

CLOUD COURSE

AWS Intermediate/Advanced

Account structure, IAM and KMS, CloudFormation and CDK, containers, APIs, messaging, edge, delivery and a Well-Architected capstone, with every example labelled by how it was verified.

12 Chapters

Philip Osztromok

Generated with Claude

AWS INTERMEDIATE/ADVANCED

Contents

Chapter 1	Account Strategy	2
Chapter 2	IAM Deep Dive	21
Chapter 3	Encryption and Secrets	47
Chapter 4	CloudFormation	72
Chapter 5	CDK	95
Chapter 6	Containers	118
Chapter 7	EKS	140
Chapter 8	API Gateway	159
Chapter 9	Messaging	183
Chapter 10	Edge and DNS	207
Chapter 11	Delivery and Operations	230
Chapter 12	Capstone	255

AWS Intermediate/Advanced

Chapter 1 · Multi-Account Foundations

AWS Fundamentals taught you to build in one account. That is the right way to learn, and the wrong way to run anything that matters. The first decision a team makes, before any server or database, is how many AWS accounts it will have and who may do what in each. This chapter covers the tools for that decision: AWS Organizations (the structure), service control policies (the guardrails), IAM Identity Center (how people sign in), Control Tower (a ready-made version of all of it), and tags (how you find out who is spending what). It ends with an account layout for a small team that you can check with a script.

> **HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED**
>
> **There is no AWS account and no AWS CLI on the machine that wrote this course**, so nothing here was run against AWS. The labels used throughout:
>
> - **Run here (level 1):** Python 3.14.0 and cfn-lint 1.57.2 on Windows 11, in a scratch folder outside this site. Output is pasted from the real run.
> - **Needs an AWS account (level 2):** none in this chapter. Nothing here creates, changes or costs anything.
> - **From the documentation, not run (level 3):** everything about how AWS itself behaves. I read the AWS Organizations, IAM Identity Center, Control Tower and Billing pages on 2026-10-12 through a page-fetch tool that summarises pages, so quotes below are as that tool returned them. **AWS changes limits and defaults often; check the current page before you rely on a number.**
>
> Two scripts below are **models of the documented rules**, not AWS. When one says "allowed", that means "allowed by my reading of the documentation", and the chapter says so each time.

Why One Account Is Not Enough

An AWS account is a boundary. Everything inside it shares one bill, one set of service quotas, one root user and one blast radius: a mistake or a stolen key anywhere in the account can reach everything else in it. Putting production, testing, experiments and your audit logs in one account means a developer experimenting with a test database sits one wrong click from your customers' data, and the bill is one number nobody can split by team. Separate accounts are the strongest dividing line AWS gives you. The reasons teams give, which are reasoning rather than measurements:

CONCERN	ONE ACCOUNT	SEVERAL ACCOUNTS
Blast radius	A bad deploy, a leaked key or a deleted stack can touch every environment	Damage stops at the account boundary unless you built a path across it
Access	Policies must keep prod and dev apart inside one set of users and roles	Different people hold different access per account; production can be very few people
Billing	One bill; splitting it relies on every resource being tagged	Each account is a line on the bill without any tagging at all
Limits	Service quotas are shared, so a load test can use up production's headroom	Quotas are generally per account, so environments do not compete
Audit and compliance	Logs live next to the things being logged, and can be deleted by the same admins	Logs go to a separate account that workload admins cannot reach

The price is administration: ten accounts means ten places to set up the same baseline. That is exactly what the rest of this chapter removes.

AWS Organizations

AWS Organizations lets you create and group accounts and apply policy to the groups. The vocabulary:

TERM	MEANING
Organization	The whole structure, with exactly one **root** at the top
Management account	The account that created the organization. It pays the bills and manages the structure
Member account	Every other account in the organization
Organizational unit (OU)	A folder of accounts and other OUs. Policies attached to an OU apply to everything beneath it
Policy	A document attached to the root, an OU or an account. Service control policies are the type this chapter covers

All features, not just billing. An organization can be set up for consolidated billing only, or with all features. The documentation is explicit that SCPs "are available only in an organization

that has all features enabled" and "aren't available if your organization has enabled only the consolidated billing features". Use all features.

Limits to design around

These are the default quotas from the Organizations quotas page (read 2026-10-12). Several are adjustable through Service Quotas; the hard ones are marked.

LIMIT	VALUE ON 2026-10-12	ADJUSTABLE?
Accounts in an organization	10 by default (newly created organizations may start lower)	Yes, on request from the management account, up to 50,000 depending on qualification
Roots	1	No
OUs in an organization	2,000	Not stated
OU nesting	Five levels deep under the root	Not stated
SCP size	10,240 characters	Not stated
SCPs attached directly to one root, OU or account	10, "all policy limits are hard limits"; every entity must keep at least one	No
Tags on a root, OU or account	50	Not stated
Member accounts created at once	5	Not stated

The 10-account default matters more than it looks: *a closed account keeps counting until it is permanently closed*, and an invitation you send counts too, according to the same page. A layout with eleven accounts needs a quota increase requested before you create the eleventh. The layout checker in Exercise 3 warns about this.

Keep the management account empty

The Organizations best-practice page says to use the management account "only for tasks that *require* the management account" and to "store all of your AWS resources in other AWS accounts". One reason it gives is the next section's subject: SCPs "do not work to restrict any users or roles in the management account", so anything you run there is outside your guardrails.

The same page recommends limiting who can reach it, reviewing that list quarterly or monthly, and delegating services such as CloudFormation StackSets to a member account where the service supports it.

Service Control Policies

A service control policy (SCP) sets the **maximum** permissions for the users and roles in the accounts it covers. The documentation is clear about the part people get wrong: "SCPs do not grant permissions". An SCP is a ceiling. Nobody gets anything from an SCP; they get it from an identity-based or resource-based policy, and only if the SCP ceiling also allows it.

SCPS DO	SCPS DO NOT
Restrict IAM users and roles in member accounts, **including each member account's root user**	Affect anything in the **management account**
Apply to member accounts that are delegated administrators	Affect **service-linked roles**
Combine with permission boundaries: the boundary, the SCP and the identity policy must all allow the action	Directly affect resource-based policies, or users and roles from accounts **outside** the organization
Apply to everything beneath the OU or root they are attached to	Restrict a short list of tasks, such as registering for the Enterprise support plan as the root user

Allow lists and deny lists

When SCPs are enabled, AWS attaches an AWS-managed policy called **FullAWSAccess** to every root, OU and account. It allows everything, so by default nothing changes. You then choose a strategy:

- **Deny list.** Keep FullAWSAccess and attach SCPs that `Deny` specific things (leaving the organization, turning off logging, unapproved regions). Everything not denied stays allowed. This is the easier one to run.
- **Allow list.** Detach FullAWSAccess and attach SCPs that `Allow` only the services you approve. Everything else is denied, including new AWS services until you add them. This is tighter and more work. The documentation warns: "You should not remove the FullAWSAccess policy unless you modify or replace it with a separate policy with allowed actions, otherwise all AWS actions from member accounts will fail".

Two examples straight from the documentation's own pages (the first is the one AWS now attaches by default to organizations created in the console, see below):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyLeaveOrganizationAndCloseAccount",
            "Effect": "Deny",
            "Action": [
                "organizations:LeaveOrganization",
                "account:CloseAccount"
            ],
            "Resource": "*"
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyAllOutsideEU",
            "Effect": "Deny",
            "NotAction": [
                "cloudfront:*",
                "iam:*",
                "route53:*",
                "support:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestedRegion": [
                        "eu-central-1",
                        "eu-west-1"
                    ]
                }
            }
        }
    ]
}
```

The second is a deny list in a clever disguise: `Deny` with `NotAction` and `StringNotEquals` reads as "deny everything *except* these global services, when the region is *not* one of these two". The excepted services (`iam`, `cloudfront`, `route53`, `support`) are the ones the documentation's example leaves out of the region lock. Reading double negatives slowly is a core SCP skill.

> **NEW IN 2026: A DEFAULT SCP FOR CONSOLE-CREATED ORGANIZATIONS**
>
> The Organizations documentation says that organizations created **through the console after July 10, 2026** automatically get SCPs enabled and the first policy above attached at the root, so member accounts cannot leave the organization or close themselves without action from the management account or a delegated administrator. It does **not** apply to organizations created with the CLI, SDKs or CloudFormation, or to older ones: those need it attached by hand. AWS recommends keeping it at the root.

The syntax rules, and a checker for them

SCP syntax is IAM policy syntax with restrictions. These rules are stated on the SCP syntax page; the checker below applies them (my implementation of the documentation, not AWS's validator):

RULE	WHAT THE DOCS SAY
Version	Must be `"2012-10-17"`
Effect	`Allow` or `Deny`
Action / NotAction	`service:Action` strings; `*` and `?` wildcards
Resource	Required (or `NotResource`). In an `Allow` statement only `"*"` is accepted; a `Deny` may name ARNs
Condition	Allowed in both `Allow` and `Deny`
Principal, NotPrincipal	Not supported
Size	Every character counts against 10,240; the console strips extra whitespace, the CLI and SDKs save the text exactly as given

Run here (level 1). I wrote 17 SCP files: five taken from or modelled on the documentation, nine with one deliberate mistake each, and three for the size and duplicate-key edge cases. The

checker's verdicts:

```
py -X utf8 scp_lint.py scp/*.json
```

```
FAIL  bad_action_format.json       237 chars as written, 112 compact
        error: R9 statement 1: Action value 's3' is not service:Action
        error: R9 statement 1: Action value 'ec2:Describe Instances' is not service:Acti
FAIL  bad_allow_with_arn.json      189 chars as written, 111 compact
        error: R8 statement 1: Allow with Resource 'arn:aws:s3:::my-bucket/*'; only '*'
FAIL  bad_effect_typo.json         167 chars as written, 89 compact
        error: R4 statement 1: Effect is 'Denied'
FAIL  bad_empty_statement.json     52 chars as written, 39 compact
        error: R3 Statement missing, empty or not an object/array
FAIL  bad_no_resource.json         136 chars as written, 72 compact
        error: R6 statement 1: no Resource or NotResource
FAIL  bad_no_version.json          136 chars as written, 64 compact
        error: R2 Version is None, must be '2012-10-17'
FAIL  bad_old_version.json         165 chars as written, 87 compact
        error: R2 Version is '2008-10-17', must be '2012-10-17'
FAIL  bad_principal.json           195 chars as written, 103 compact
        error: R7 statement 1: Principal is not supported in SCPs
FAIL  bad_trailing_comma.json
        error: R1 not valid JSON: Illegal trailing comma before end of array (line 1 col
PASS  good_allow_list.json         402 chars as written, 200 compact
PASS  good_default_controls.json   323 chars as written, 184 compact
PASS  good_protect_role.json       513 chars as written, 276 compact
PASS  good_region_lock.json        570 chars as written, 242 compact
PASS  good_single_statement_object.json 104 chars as written, 104 compact
PASS  size_fits_only_compact.json  18932 chars as written, 10222 compact
        warn:  R10 18932 chars as written is over 10240, but 10222 without whitespace fi
FAIL  size_too_big.json            11022 chars as written, 11022 compact
        error: R10 11022 chars even without whitespace is over the 10240 limit
PASS  warn_duplicate_key.json      114 chars as written, 87 compact
        warn:  R1 duplicate key(s) ['Effect']: docs do not say how AWS treats them
```

Two lines are worth reading closely. `size_fits_only_compact.json` is 18,932 characters as written (indented) but 10,222 without whitespace: it fits the 10,240 limit only if the whitespace is stripped, which the console does and the CLI does not. That is a practical trap: a policy that saves fine from the console can be refused when the same file is sent with `aws organizations create-policy`. And the duplicate `Effect` key passes with a warning, because the

documentation does not say what AWS does with it (Node, Python and jq all keep the last, see *Working with JSON 11*). I did not guess.

Linting the template that deploys an SCP

You will usually deploy SCPs with infrastructure as code (*AWS Intermediate/Advanced 4* covers CloudFormation). `AWS::Organizations::Policy` carries the SCP in a `Content` property. This template is the default-controls policy, and `cfn-lint` passes it:

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Attach the default-controls SCP (deny LeaveOrganization and CloseAccount) t
Parameters:
  TargetId:
    Type: String
    Description: Root, OU or account ID the SCP is attached to
    AllowedPattern: "^(r-[0-9a-z]{4,32}|ou-[0-9a-z]{4,32}-[0-9a-z]{8,32}|[0-9]{12})$"
Resources:
  DenyLeaveAndClose:
    Type: AWS::Organizations::Policy
    Properties:
      Name: deny-leave-and-close
      Description: Member accounts cannot leave the organization or close themselves
      Type: SERVICE_CONTROL_POLICY
      TargetIds:
        - !Ref TargetId
      Content:
        Version: "2012-10-17"
        Statement:
          - Sid: DenyLeaveOrganizationAndCloseAccount
            Effect: Deny
            Action:
              - organizations:LeaveOrganization
              - account:CloseAccount
            Resource: "*"
```

Run here (level 1). I linted that template, then six broken copies. cfn-lint 1.57.2 said:

```
py -X utf8 lint_templates.py
```

```
=== OK (unchanged): exit status 0
(no findings)
```

```
=== policy Type misspelt: exit status 2
E3030 'SERVICE_CONTROL_POLICIES' is not one of ['AISERVICES_OPT_OUT_POLICY', 'BACKUP_POL
tmpl_variant.yaml:14:7
=== Name removed: exit status 2
E3003 'Name' is a required property
tmpl_variant.yaml:11:5
=== TargetIds removed: exit status 4
W2001 Parameter TargetId not used.
tmpl_variant.yaml:4:3
=== Content is a string: exit status 0
(no findings)
=== Principal inside Content: exit status 0
(no findings)
=== Effect misspelt in Content: exit status 0
(no findings)
```

cfn-lint caught the misspelt policy type and the missing `Name`, and only hinted at the removed `TargetIds` (it noticed the parameter was no longer used, which is not the same as knowing a target is missing). It **missed all three mistakes inside** `Content`: a string instead of a policy, a `Principal`, and `Effect: Denied` all gave exit status 0. For this version, the template linter checks the shape of the CloudFormation resource but not the policy language inside it. That is why the SCP checker above is a separate step, and why neither tool replaces testing on one account first (see below). Not run: other cfn-lint versions.

How SCPs combine

The evaluation page gives the rules, and they are worth stating exactly:

- **Allow needs a yes at every level.** "There must be an explicit `Allow` statement at every level from the root through each OU in the direct path to the account (including the target account itself)."
- **Deny needs a no at any level.** "Any SCP from the root through each OU in the direct path to the account (including the target account itself) can deny that permission." A deny at the root beats an allow below it.
- **Several SCPs at the same level add together.** Scenario 7 on the page has two allow lists at the root, one for IAM and EC2 and one for S3 and CloudWatch, and the accounts get all four services.

I turned the page's scenarios into a small model and ran them. The "docs say" column is my reading of each scenario's prose (the figures are images I could not see, so for scenarios 1 and 4

I assumed the accounts not mentioned as having extra SCPs have only FullAWSAccess). The model is the evaluator code below; it handles `Action`, `NotAction`, and multiple SCPs per level, and **refuses** to guess about `Condition`.

```
def level_verdict(scps, action):
    allow = any(s["Effect"] == "Allow" and stmt_hits(s, action) for p in scps for s in p
    deny = any(s["Effect"] == "Deny" and stmt_hits(s, action) for p in scps for s in p)
    return allow, deny


def scp_allows(path_levels, action):
    """path_levels: list (root ... account) of lists of policies (each policy = list of 
    for lv in path_levels:
        allow, deny = level_verdict(lv, action)
        if deny or not allow:
            return False
    return True


def effective(path_levels, identity_allow, action, management=False):
    if management:
        return match(identity_allow, action)          # SCPs do not apply
    return scp_allows(path_levels, action) and match(identity_allow, action)
```

Run here (level 1). The model against the documentation's scenarios (a set of five probe actions: `ec2:RunInstances`, `s3:GetObject`, `iam:CreateRole`, `cloudwatch:PutMetricData`, `lambda:InvokeFunction`):

```
py -X utf8 scp_eval.py
```

```
scenario                       account         docs say allowed
1 deny at OU and account       A               cloudwatch,ec2,iam,lambda
1 deny at OU and account       B               cloudwatch,iam,lambda
2 allow-list at the OU         A               ec2
2 allow-list at the OU         B               ec2
3 root has only a Deny         A               (none)
4 layered deny, two OU levels  D (Test)        cloudwatch,iam,lambda,s3
4 layered deny, two OU levels  E (Production)  cloudwatch,ec2,iam,lambda,s3
6 root deny beats lower allow  D (Test)        (none)
6 root deny beats lower allow  E (Production)  cloudwatch,ec2,iam,lambda
7 two allow-list SCPs at the root A            cloudwatch,ec2,iam,s3
```

```
  mismatches: 0 of 10

  control (one expectation flipped on purpose): mismatches 1 of 10
```

All ten account results match what the documentation says. The control line shows the comparison can fail: with one expectation flipped on purpose, exactly one mismatch is reported. The rest of the run shows the two ideas people get wrong:

```
  py -X utf8 scp_eval.py (continued)
```

```
  -- the SCP grants nothing: same account, three identity policies
    no identity policy             s3:GetObject -> False,  ec2:RunInstances -> False
    identity policy allows s3:*    s3:GetObject -> True,  ec2:RunInstances -> False
    AdministratorAccess (*)        s3:GetObject -> True,  ec2:RunInstances -> True
  -- same account, but an SCP Deny on ec2, with AdministratorAccess
    ec2:RunInstances -> False,  s3:GetObject -> True
  -- the management account with the same deny
    ec2:RunInstances -> True  (SCPs do not apply there)
  -- what the model refuses to guess
    NotModelled: Condition
```

An account with no identity policy can do nothing even though the SCP allows everything (the ceiling is not a grant); an administrator with `*` still cannot run EC2 once an SCP denies it; and the same deny does not touch the management account. This is a model of the documented rules. It says nothing about how real requests are evaluated once conditions, resource policies, boundaries or session policies join in: *AWS Intermediate/Advanced 2* takes the full evaluation order.

> **TEST AN SCP ON ONE ACCOUNT FIRST**
>
> The documentation says AWS "strongly recommends that you don't attach SCPs to the root of your organization without thoroughly testing the impact". Create an OU, move one account in, attach the SCP there, and use CloudTrail or IAM's "service last accessed" data to see what the account really uses before widening it. An SCP that blocks a service a pipeline depends on fails the pipeline, not the person who wrote the policy.

Signing In: IAM Identity Center

In *AWS Fundamentals 2* you met IAM users, with passwords and access keys. With many accounts that does not scale: a user per person per account, each with their own long-lived keys. AWS IAM Identity Center is the replacement for people (workforce users). The documentation describes it as the way to "manage centrally your workforce access to AWS accounts":

- **One sign-in point.** Users sign in at the AWS access portal and see the accounts and roles assigned to them. You can connect an existing identity provider (it supports SAML 2.0) or create users in Identity Center itself.
- **Permission sets.** A permission set is a reusable bundle of permissions for a job function such as administrator or read-only. You provision permission sets across accounts and assign them to users and groups. The documentation says this feature "is available only for organization instances".
- **Organization instance.** The documentation calls this "the best practice": it is deployed in the management account, and it is the only kind of instance that manages access to AWS accounts. An *account instance* is bound to one account and is for isolated application use.
- **Old name.** It was AWS Single Sign-On until 2022-07-26. The CLI commands and API namespaces still use `sso` (for example `aws configure sso`), so older tutorials still apply.

The usual reason for this design is that people work with *short-lived* credentials from the portal rather than permanent access keys. That is how it is commonly described and how I understand it; the Identity Center pages I read did not contain that sentence, so I am not quoting it. Level 3 for everything here: setting it up needs an Organizations management account, and I have none. On a machine with the CLI you would run `aws configure sso` and answer its prompts; I did not run it and did not check its current wording.

AWS Control Tower

Everything above can be done by hand. AWS Control Tower does a recommended version of it for you. The documentation says it "orchestrates the capabilities of several other AWS services, including AWS Organizations, AWS Service Catalog, and AWS IAM Identity Center, to build a landing zone in less than an hour". A **landing zone** is the multi-account environment it builds: OUs, accounts and baseline settings. The other pieces:

- **Controls (guardrails).** High-level rules, in three kinds: preventive, detective and proactive. The page I read does not say which mechanism each kind uses, so I do not claim one.
- **Account Factory.** A template for creating new accounts that already match your standards.

- **Dashboard.** Shows accounts, enabled controls and non-compliant resources.

The documentation's own advice: "If you are hosting more than a handful of accounts, it's beneficial to have an orchestration layer". For a team of three with four accounts, building the structure by hand from this chapter is reasonable and teaches more; for a company with dozens, use Control Tower. Level 3: not run. I did not check what the service costs beyond the resources it creates, so check the pricing page before enabling it.

Tags and Cost Allocation

Accounts split the bill by team and environment for free. Tags split it *inside* an account and across accounts (for example "everything for the shop product"). A tag is a key and a value on a resource. Billing's rules, from the Cost allocation tags page:

- A tag does nothing for billing until you **activate it** as a cost allocation tag in the Billing and Cost Management console. User-defined and AWS-generated tags are activated separately. User-defined tags appear with a `user:` prefix, AWS-generated ones with `aws:`.
- "All tags can take up to 24 hours to appear in the Billing and Cost Management console."
- In an organization, only the management account (or a standalone account) can reach the cost allocation tags manager.
- The cost allocation report "includes both tagged and untagged resources", and its total reconciles with the Bills page. Untagged spend does not vanish; it is a line you can measure.
- "As a best practice, don't include sensitive information in tags."

Tag *policies* (a policy type in Organizations, up to 10 attached per entity) can require tag keys and values to follow a standard. I did not read their syntax page, so I do not show one.

Measuring tag coverage

The point of tagging is to find the gaps. Here is the whole technique on **invented data**: a fixed-seed generator makes a 48-row file shaped like a cost report (account, service, tag columns, cost). The dollar amounts are not AWS prices; only the method is real. The script groups the cost by tag with an explicit "(untagged)" bucket, checks that every grouping adds up to the file total, and lists the biggest untagged rows.

```
py -X utf8 cost_report.py
```

```
first three lines of costs.csv:
account,service,user:owner,user:env,cost_usd
acme-management,Amazon EC2,platform,mgmt,0.71
acme-management,Amazon S3,platform,mgmt,2.27
acme-management,Amazon RDS,platform,mgmt,0.65
```

```
py -X utf8 cost_report.py (continued)
```

```
rows 48, total 1,643.56 (synthetic dollars)
-- by owner tag
   shop                     1,018.79   62.0%
   (untagged)                 376.22   22.9%
   platform                   146.17    8.9%
   security                   102.38    6.2%
```

```
py -X utf8 cost_report.py (continued)
```

```
-- the untagged rows, biggest first (where the next tagging effort should go)
   acme-shop-prod          AWS Lambda             171.84
   acme-shop-prod          Amazon CloudWatch       84.68
   acme-shop-staging       Amazon EC2              65.66
   acme-shared-services    Amazon EC2              29.48
   acme-shop-dev           AWS Data Transfer       14.14
totals reconcile: owner buckets = account buckets = env buckets = file total

-- control: drop one row from the data before grouping, and the reconciliation must fail
   file total 164356 cents, after dropping a row 164285 cents -> DETECTED
```

About a fifth of the rows had lost their `owner` tag by design, and they hold **22.9% of the spend**. The unrelated groupings (by account, by environment) are unaffected, which is the argument for doing both: accounts give you a total that cannot be mistagged, tags give you the cross-cutting view. The last line is a control: dropping one row changes the total, and the reconciliation catches it.

A Layout for a Small Team

A good first layout is boring. Separate the things that must never be touched by workload admins (logs, security tools), the things shared by everyone (network, pipelines), the workloads by environment, and a place to experiment. Eight accounts for one product:

```
py -X utf8 layout_check.py
```

```
Root   (management account: acme-management)
├── Security/   SCPs: deny-leave-and-close, deny-disable-logging
│   • acme-log-archive
│   • acme-security-tooling
├── Infrastructure/   SCPs: deny-leave-and-close
│   • acme-shared-services
├── Workloads/   SCPs: deny-leave-and-close, region-lock-eu
│   ├── Prod/   SCPs: deny-delete-backups
│   │   • acme-shop-prod
│   └── NonProd/   SCPs: -
│       • acme-shop-staging
│       • acme-shop-dev
└── Sandbox/   SCPs: deny-leave-and-close, region-lock-eu, deny-expensive-services
    • acme-sandbox
```

OU	WHY IT EXISTS	SCPS IN THIS PLAN
Security	Log archive and security tooling. Workload admins get no access, so the evidence cannot be altered by the people it is evidence about	leave/close denied; a deny on turning off logging
Infrastructure	Shared services: networking, CI/CD runners, shared images	leave/close denied
Workloads / Prod	Production, with few people and the strictest guardrails	leave/close and region lock (inherited), plus protection of backups
Workloads / NonProd	Staging and development, where people may try things	inherits from Workloads
Sandbox	Throw-away experiments, with budget alerts and a block on expensive services	leave/close, region lock, deny expensive services

Notice what the nesting buys. An SCP attached to `Workloads` reaches Prod and NonProd without being attached to either, and the "Number of policies you can attach" limit only counts

policies attached *directly*, not inherited ones. The management account sits at the root and holds no workload. The plan names five SCPs; this chapter wrote two of them as files (the leave/close one and the region lock), so the rest are names in a plan, not tested policies.

Run here (level 1). The checker passes this layout and rejects the five broken copies I made (a missing tag, a workload in the management account, a shared e-mail address, an eleventh account, OUs nested six deep):

```
py -X utf8 layout_check.py (continued)
```

```
=== the layout: 8 accounts, 6 OUs, deepest OU path 2 -> PASS
=== broken 1: new account without a cost-center tag: 9 accounts, 6 OUs, deepest OU path 2
   error: L4 x is missing tags ['cost-center']
=== broken 2: workload placed in the management account: 8 accounts, 6 OUs, deepest OU pa
   error: L3 management account acme-management hosts a workload
=== broken 3: two accounts share one e-mail: 8 accounts, 6 OUs, deepest OU path 2 -> FAIL
   error: L6 duplicate e-mail aws+shop-prod@example.com
=== broken 4: 11 accounts: 11 accounts, 6 OUs, deepest OU path 2 -> PASS
   warn:  L2 11 accounts is over the default quota of 10: a Service Quotas increase must
=== broken 5: OUs nested six deep, and an empty OU chain: 8 accounts, 12 OUs, deepest OU
   error: L1 OU nesting is 6 levels, limit 5
   warn:  L7 OU L0 has no account under it
   warn:  L7 OU L0/L1 has no account under it
   warn:  L7 OU L0/L1/L2 has no account under it
   warn:  L7 OU L0/L1/L2/L3 has no account under it
   warn:  L7 OU L0/L1/L2/L3/L4 has no account under it
   warn:  L7 OU L0/L1/L2/L3/L4/L5 has no account under it
```

The 11-account copy passes with a warning rather than failing, because the quota is a default that can be raised. Two of the checker's rules are mine rather than AWS's: the required tags are a team convention, and the check for duplicate e-mail addresses is a sanity check (I did not find an e-mail rule on the pages I read, so I do not claim one).

Cost and Clean-Up

Nothing in this chapter creates anything in AWS, so there is nothing to delete and nothing to pay. When you do build this for real, start with a budget alert (*AWS Fundamentals 9*) on the management account, create the OUs and accounts, and test each SCP on the sandbox account

before the root. I did not check what AWS Organizations, Identity Center or Control Tower charge for themselves (some of it may be free, and the resources they create may not be): read the pricing pages before enabling them.

WHAT WAS AND WAS NOT CHECKED

Run here: the SCP checker (17 files), cfn-lint on one template and six broken copies, the evaluation model against ten documented results, the layout checker (one layout and five broken ones) and the tag report on invented data. **From the documentation, not run:** every statement about AWS behaviour, limits and defaults (read 2026-10-12, possibly summarised by the fetch tool), the July 10, 2026 default SCP, Identity Center and Control Tower. **Not run at all:** anything in a real organization, `aws organizations` commands, the access portal, Control Tower, tag activation, and any other version of the tools used.

Practice

EXERCISE 1

Write a checker for SCP files that applies the documented syntax rules (version, effect, action format, resource rules for `Allow`, no `Principal`, size both as written and with whitespace removed). Run it on five good policies and ten with one mistake each, and report the size trap. Then lint a CloudFormation template that deploys an SCP, along with six broken copies, and record which mistakes cfn-lint catches and which it misses.

View solution

EXERCISE 2

Model the documented SCP evaluation rules (an allow at every level, a deny at any level, union of SCPs at one level, management account exempt) and run the documentation's scenarios as tests. Write your expected results first, count mismatches, and add a control that flips one expectation. Show that an SCP grants nothing and that the model refuses to guess about conditions.

View solution

EXERCISE 3

Describe an account layout for a small team as JSON and write a checker (OU depth, account count against the default quota, no workload in the management account, required tags, distinct names, unknown OUs and SCP names). Show it rejecting five broken layouts. Then group an invented cost file by tag, measure the untagged share, and prove the totals reconcile.

View solution

CHAPTER 1 QUICK REFERENCE

- One account is one blast radius, one bill and one set of quotas; separate accounts are the strongest boundary and the bill splits itself
- Organizations: one root, a management account, member accounts, OUs (five levels deep at most); use **all features**, because SCPs do not exist in consolidated-billing-only mode
- Default quotas (2026-10-12): 10 accounts (adjustable), 10,240-character SCPs, 10 SCPs attached per root, OU or account; closed accounts count until permanently closed
- Keep workloads out of the management account: SCPs do not apply to it
- An SCP is a ceiling, never a grant. Allowed = an Allow at every level from root to account, no Deny anywhere, and an identity or resource policy that grants it
- SCPs apply to member-account roots too; they do not touch service-linked roles or the management account; Allow statements accept only `Resource: "*"`; no `Principal`
- Deny list = keep FullAWSAccess, deny specifics (easy). Allow list = replace FullAWSAccess with approved services (tight). Never remove FullAWSAccess without a replacement
- Whitespace counts toward the size limit unless the console strips it, so a policy can save from the console and be refused by the CLI
- Console-created organizations after July 10, 2026 get a leave/close deny at the root; older or CLI/CloudFormation-created ones need it attached by hand
- IAM Identity Center (organization instance in the management account) gives people portal sign-in and permission sets per account; CLI namespace is still `sso`
- Tags only reach the bill after activation, and take up to 24 hours; measure the untagged share and make buckets reconcile
- cfn-lint 1.57.2 checks an SCP resource's shape but not the policy inside `Content`; lint the policy separately and test on one account first

COMING NEXT

Chapter 2 is the IAM deep dive: the full policy evaluation order (where SCPs, permission boundaries, session policies, identity and resource policies each sit), conditions and policy variables, roles and STS, cross-account access, and the tools for proving a policy does what you think. Start with *AWS Intermediate/Advanced 2*.

AWS Intermediate/Advanced

Chapter 2 · IAM Policy Evaluation and Cross-Account Access

AWS Fundamentals 2 taught you to write an IAM policy that allows something. This chapter is about the harder question: given *all* the policies that could apply to a request, will it be allowed? Those policies include the service control policies from *AWS Intermediate/Advanced 1*, permissions boundaries, session policies, identity-based policies and resource-based policies, in two accounts when the request crosses an account. You will learn the evaluation order, the rules for conditions and policy variables, how roles and STS (the Security Token Service) hand out temporary access, how to let another account in without opening a door for strangers, and a workflow for shrinking a broad policy to what a workload really uses.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS CLI on the machine that wrote this course, so nothing here was run against AWS.

- **Run here (level 1):** Python 3.14.0 on Windows 11, in a scratch folder outside this site, with `boto3` / `botocore` 1.43.112 (used offline, through its test Stubber) and the open-source linter `parliament` 1.6.4 (offline, with the setuptools pin that it needs on this Python). Output is pasted from the real runs.
- **Needs an AWS account (level 2):** none in this chapter. Nothing here creates, changes or costs anything.
- **From the documentation, not run (level 3):** everything about how AWS itself behaves. I read the IAM, STS and Access Analyzer pages on 2026-10-12 through a page-fetch tool that summarises them, so quotes are as that tool returned them. **The copy I could read of one page ended early**; the chapter says where that matters.

The central tool is **a model I wrote of the documented evaluation rules (`iam_eval.py`), not AWS**. I tested it against the documentation's own worked examples and tables. When it says "Allow", that means "allowed by my reading of the documentation". It refuses (raises an error) for anything it does not model, instead of guessing.

The Order AWS Evaluates a Request In

Every request starts as an implicit **deny**. The documentation says "By default, all requests are implicitly denied with the exception of the AWS account root user, which has full access", and

"An explicit deny overrides an explicit allow". The enforcement code then works through the policy types in this order (the page's own list, in my words):

#	STEP	OUTCOME IF IT FAILS
1	**Explicit deny.** Look through *every* applicable policy (SCPs, RCPs, resource-based, identity-based, permissions boundaries, session policies) for a `Deny` that matches	One matching `Deny` anywhere gives a final **Deny**
2	**Resource control policies (RCPs).** An Organizations policy type attached to *resources* of the accounts it covers; `RCPFullAWSAccess` is attached automatically when RCPs are enabled	No applicable `Allow`: Deny
3	**Service control policies (SCPs).** Apply to the *principals* of the accounts they cover (*AWS Intermediate/Advanced 1*)	No applicable `Allow`: Deny
4	**Resource-based policy.** Depending on the principal it names, an `Allow` here can be enough on its own (next section)	Continues to step 5, or ends in Allow
5	**Identity-based policy.** Some statement must allow the action	None: implicit Deny
6	**Permissions boundary,** if the user or role has one	It does not allow the action: implicit Deny
7	**Session policy,** if the principal is a session and one was passed	It does not allow the action: implicit Deny; otherwise **Allow**

Three rules follow from the page, and they are the whole of the chapter in miniature. **Inside one account**, identity-based and resource-based policies are a *union*: "If an action is allowed by an identity-based policy, a resource-based policy, or both, then AWS allows the action." Identity policies and permissions boundaries are an *intersection*: "the resulting permissions are the intersection of the two categories". And SCPs, boundaries and identity policies are all ceilings and grants together: "an action must be allowed by all three policy types". **Across accounts**, there are two evaluations, one in each account, and "the request is allowed only if both evaluations return a decision of Allow".

Who the resource policy names changes what it can do

This is the part most people have never heard, and it is stated plainly on the evaluation page. Within the same account, a resource-based policy that names the requester can grant access

even when an identity policy, boundary or session policy would not, depending on which kind of principal it names:

THE RESOURCE POLICY NAMES	LIMITED BY THE PRINCIPAL'S IDENTITY POLICY?	LIMITED BY A BOUNDARY OR SESSION POLICY?	DOCUMENTED EXAMPLE ARN
An IAM **user**	No	No (a user has no session policy)	`arn:aws:iam::111122223333:user/exampleuser`
An IAM **role** (by ARN, or with `aws:PrincipalArn`)	No	**Yes**: limited by an implicit deny in a boundary or session policy	`arn:aws:iam::111122223333:role/examplerole`
A **role session**	No	No: "Permissions granted directly to a session are not limited"	`arn:aws:sts::111122223333:assumed-role/examplerole/examplerolesessionname`
A **federated user session** (`GetFederationToken`)	No	No	`arn:aws:sts::111122223333:federated-user/exampleuser`
The IAM user **who federated**	No	**Yes**	`arn:aws:iam::111122223333:user/exampleuser`

The page explains the role-versus-session split: "When you assume a role and make a request, the principal making the request is the IAM role session ARN and not the ARN of the role itself." Two exceptions are also stated. **Role trust policies and KMS key policies** "must explicitly allow access for principals": an identity policy alone is never enough for them. And "Resource-based policies for services other than IAM and AWS KMS may also require an explicit Allow statement within the same account", so for any given service you check its own documentation (*AWS Intermediate/Advanced 3* returns to key policies).

A model of those rules, tested against the documentation

I turned the steps into code and fed it the documentation's own worked examples: the boundary on user Shirley, the cross-account request from Carlos's `Demo` role to an S3 bucket, the delegated user-creation example (Maria, Zhang and Nikhil), the principal types in the table above, and the three-way SCP, boundary and identity combinations. The core of it (the full file is in Exercise 1):

```
def decide(req, S, explicit_resource_allow=False):
    """same-account evaluation. req: who, action, resource, ctx. S: scp (list of levels)
    explicit_resource_allow: a role trust policy or KMS key policy, which must itself al
    who = req["who"]
    if side_deny(req, S):
        return False, "explicit Deny"
    for i, lv in enumerate(S.get("scp") or []):
        if not hits(lv, req, "Allow"):
            return False, "SCP level %d has no Allow" % (i + 1)
    kinds = [grant_kind(s, who) for p in S.get("resource", []) for s in stmts(p) if s.ge
    if any(k == "wildcard" for k in kinds):
        raise NotModelled("wildcard Principal in an Allow")
    kinds = [k for k in kinds if k]
    if explicit_resource_allow and not kinds:
        return False, "this resource policy must explicitly allow the principal (it does
    for k in kinds:
        if k in DIRECT:
            return True, "resource-based policy grants the %s directly" % k
    limited = any(k in LIMITED for k in kinds)
    if not limited:
        if not hits(S.get("identity", []), req, "Allow"):
            return False, "no identity-based Allow (implicit deny)"
    if S.get("boundary") and not hits([S["boundary"]], req, "Allow"):
        return False, "permissions boundary does not allow (implicit deny)"
    if who["type"] in ("role-session", "federated-session") and S.get("session") and not
        return False, "session policy does not allow (implicit deny)"
    return True, "allowed (via the %s grant)" % kinds[0] if limited else "allowed by ide
```

Run here (level 1). Each row's "docs" value was written down before the model ran:

```
py -X utf8 t_principals.py
```

```
case (source: AWS IAM page named in the script)
boundary: identity allows iam:CreateUser, boundary does not -> iam:CreateUser
boundary: identity allows s3:*, boundary allows s3:* -> s3:GetObject
```

```
boundary: a boundary alone grants nothing -> s3:GetObject with no identity policy
boundary updated to include iam:CreateUser -> iam:CreateUser
cross-account: PutObject to the production-logs bucket -> Deny (DenyS3Logs in the reques
cross-account: PutObject to the production bucket -> Allow (both accounts allow)
cross-account: DeleteObject on the production bucket -> Deny ('can create and edit, but
cross-account: bucket policy allows the role but the role has no identity policy -> Deny
cross-account: identity policy allows but the bucket policy is empty -> Deny
Nikhil: bucket policy allows PutObject on 'logs', boundary has an explicit Deny on it ->
Nikhil: secret's resource policy names his user ARN; boundary does not mention Secrets M
resource policy names an IAM USER ARN; identity none, boundary no -> Allow
resource policy names an IAM ROLE ARN; boundary and session policy do not allow -> Deny
resource policy names the ROLE SESSION ARN; boundary and session policy do not allow ->
resource policy names the FEDERATED USER session ARN -> Allow
resource policy names the IAM user who federated, not the session -> Deny
```

```
py -X utf8 t_principals.py (continued)
```

```
SCP allows / boundary allows / identity allows -> s3:GetObject
SCP allows / boundary allows / identity omits -> s3:GetObject
SCP allows / boundary omits / identity allows -> s3:GetObject
SCP allows / boundary omits / identity omits -> s3:GetObject
SCP omits / boundary allows / identity allows -> s3:GetObject
SCP omits / boundary allows / identity omits -> s3:GetObject
SCP omits / boundary omits / identity allows -> s3:GetObject
SCP omits / boundary omits / identity omits -> s3:GetObject
all three allow, but an explicit Deny in the scp -> Deny
all three allow, but an explicit Deny in the boundary -> Deny
all three allow, but an explicit Deny in the identity -> Deny
Zhang: iam:CreateUser WITHOUT setting a boundary -> Deny
Zhang: iam:CreateUser with the XCompanyBoundaries boundary -> Allow
Zhang: iam:CreateUser with a different boundary policy -> Deny
Zhang: iam:DeleteUserPermissionsBoundary -> Deny (explicit)

31 cases, mismatches: 0
```

All 31 results match the documentation. The control, running the script with one expectation flipped on purpose, reports `31 cases, mismatches: 1`, so the comparison can fail. One thing to know about how I got here: **the first run reported a mismatch**, and it was my test data, not the model: I had written an SCP level as a plain list of statements where the model expected a list of policies, and the model silently read it as "no statements". That is the dangerous failure for any

policy tool (a malformed policy quietly treated as empty), so the model now raises an error on anything that is not a policy or a statement, and the table above is from the fixed version. The model does not handle resource control policies, wildcard principals in an `Allow`, account principals in ordinary resource policies, or role chaining; it raises an error for those.

Conditions and Policy Variables

A `Condition` compares keys in the **request context** with values in the policy. The rules below are on the Condition and Condition-operators pages. The ones that cause real incidents are about keys that are *missing*:

SITUATION	DOCUMENTED RESULT
Key missing, ordinary operator (`StringEquals`, `Bool`, `NumericLessThan`, `IpAddress`, ...)	"The values do not match and the condition is *false*"
Key missing, **negated** operator (`StringNotEquals`, `StringNotLike`, `ArnNotLike`, ...)	The condition is *true*
Operator ends in `IfExists`	Key missing: true. Key present: evaluated normally. "If you are using an `"Effect": "Deny"` element with a negated condition operator like `StringNotEqualsIfExists`, the request is still denied even if the condition key is not present"
`Null` operator	Tests only whether the key exists (`"false"` means "must exist")
`ForAllValues:` qualifier	True if every request value matches, **and also true if the key is missing or empty**
`ForAnyValue:` qualifier	True if at least one request value matches; false if the key is missing
Several keys or operators in one `Condition`	All must hold (the page's tables show a statement failing when any one part fails)
Several values for one key	Any one may match (the page's own example: "as long as the `s3:prefix` matches any one of the specified patterns")
Policy variables such as `${aws:username}`	Supported in String, ARN and Boolean operators; **not supported** in Numeric, Date, Binary and IP address operators or in `Null`

One rule I could not verify. How a *plain* negated operator combines several policy values (does `StringNotEquals` with two values mean "equals neither"?) is a sentence on the Condition page that was past the end of the copy I could read. The set-operator forms are documented ("None of the values of the context key in the request can match any of the context key values in your policy"), and my model implements the plain form the same way, but treat that one as **unverified**. Also note the page's advice on key types: "Multivalued context keys require a condition set operator", and `ForAllValues` or `ForAnyValue` must not be used on single-valued keys.

The part of the model that implements the missing-key and set-operator rules:

```
  def clause(op, key, polvals, ctx):
      """one operator/key pair -> True/False"""
      set_op = None
      if op.startswith("ForAllValues:"):
          set_op, op = "all", op[len("ForAllValues:"):]
      elif op.startswith("ForAnyValue:"):
          set_op, op = "any", op[len("ForAnyValue:"):]
      ifex = op.endswith("IfExists")
      if ifex:
          op = op[:-len("IfExists")]
      k = key.lower()
      if op == "Null":
          present = k in ctx and ctx[k] not in (None, "", [])
          return (not present) == (str(L(polvals)[0]).lower() == "true")
      neg = op in NEG
      base = NEG.get(op, op)
      if base not in POS:
          raise NotModelled("operator " + op)
      if k not in ctx:
          if ifex or set_op == "all":
              return True                    # "IfExists": missing key is true; ForAllVal
          if set_op == "any":
              return False
          return neg                         # missing key: positive operators false, neg
      reqs = L(ctx[k])
      try:
          pol = []
          for v in L(polvals):
              if isinstance(v, str) and "${" in v and not base.startswith(VAR_OK):
                  raise NotModelled("policy variable in " + op)
              pol.append(subst(v, ctx))
      except Missing:
```

```
        return False                        # a variable with no value: documented table
    m = POS[base]
    if set_op == "all":
        return all((not any(m(r, p) for p in pol)) if neg else any(m(r, p) for p in pol)
    if set_op == "any":
        return any(all(not m(r, p) for p in pol) if neg else any(m(r, p) for p in pol) f
    if len(reqs) != 1:
        raise NotModelled("multivalued key %s needs ForAllValues or ForAnyValue" % key)
    r = reqs[0]
    return (not any(m(r, p) for p in pol)) if neg else any(m(r, p) for p in pol)
```

Run here (level 1). I ran every table I could find on those pages through it: the tag-key examples, the `${aws:username}` home-directory example, the numeric, date, Boolean, IP and ARN examples, `IfExists`, `Null` and both set operators. Again the documentation's answer was written first.

```
py -X utf8 t_conditions.py
```

```
case                                                                docs      model     r
StringEquals PrincipalTag: iamuser-admin                            Match     Match     m
StringEquals PrincipalTag: dev-ops                                  No match  No match  m
StringEquals PrincipalTag: key absent                               No match  No match  m
variable ${aws:username}: martha_rivera lists home/martha_rivera/   Match     Match     m
variable ${aws:username}: nikki_wolf lists home/martha_rivera/      No match  No match  m
variable ${aws:username}: no username in the request                No match  No match  m
tags: TagKeys=[environment], environment=preprod                    Match     Match     m
tags: TagKeys=[environment,costcenter], environment=PreProd         Match     Match     m
tags: TagKeys=[Environment] (capital E)                             No match  No match  m
tags: TagKeys=[costcenter]                                          No match  No match  m
tags: no TagKeys in the request                                     No match  No match  m
tags: no RequestTag/environment                                     No match  No match  m
tags: neither key present                                           No match  No match  m
Numeric: MFA age 2800 <= 3600                                       Match     Match     m
Numeric: MFA age 4000                                               No match  No match  m
Numeric: key absent (access keys, no MFA)                           No match  No match  m
Date: token issued 2026 > 2020                                      Match     Match     m
Date: key absent (long-term access keys)                            No match  No match  m
Bool: SecureTransport=false                                         Match     Match     m
Bool: SecureTransport=true                                          No match  No match  m
Bool: SecureTransport absent                                        No match  No match  m
IpAddress: 198.51.100.1                                             No match  No match  m
```

```
  ArnEquals: same topic                                              Match     Match     m
  ArnEquals: topic in another account                                No match  No match  m
  ArnEquals: key absent (a person called, not a service)             No match  No match  m
  ForAllValues:ArnLike: one ARN in the account                       Match     Match     m
  ForAllValues:ArnLike: two ARNs in the account                      Match     Match     m
  ForAllValues:ArnLike: ARN in another account                       No match  No match  m
  ForAllValues:ArnLike: key absent -> TRUE (the page says so)        Match     Match     m
  IfExists: t1.micro                                                 Match     Match     m
  IfExists: m2.micro                                                 No match  No match  m
  IfExists: key absent -> true                                       Match     Match     m
  Null false: temporary credentials (key present)                    Match     Match     m
  Null false: long-term credentials (key absent)                     No match  No match  m
  ForAllValues+Null: [environment]                                   Match     Match     m
  ForAllValues+Null: [cost-center]                                   Match     Match     m
  ForAllValues+Null: [environment, cost-center]                      Match     Match     m
  ForAllValues+Null: [environment, dept]                             No match  No match  m
  ForAllValues+Null: no TagKeys                                      No match  No match  m
  ForAnyValue: [environment]                                         Match     Match     m
  ForAnyValue: [cost-center]                                         Match     Match     m
  ForAnyValue: [environment, cost-center]                            Match     Match     m
  ForAnyValue: [environment, dept]                                   Match     Match     m
  ForAnyValue: [dept]                                                No match  No match  m
  ForAnyValue: no TagKeys                                            No match  No match  m
  StringNotLike with the key absent -> true                          Match     Match     m
  ArnNotLike with the key absent -> true                             Match     Match     m
  StringNotEquals, key present and equal -> false                    No match  No match  m

  49 cases, mismatches: 0
```

49 of 49 match. **Three rows I did not count**, and why: for the `aws:ResourceOrgPaths` example the page's table says *Match* for a path inside `sportsou/` and *No match* for a path under `mediaou/` and for a missing key. `ForAllValues:StringNotLike` is true only when *no* request value matches *any* pattern, and the page itself says `ForAllValues` is true on a missing key, so by the page's own rules those three results come out the other way round (below). It reads to me as an error in that table, and in a Deny statement the page's intent (deny access *outside* the listed OUs) agrees with the model. I cannot be certain, so I show it rather than bury it:

```
  py -X utf8 t_conditions.py (continued)

  NOT counted: the aws:ResourceOrgPaths example. The page's table says Match for sportsou/
  The operator ForAllValues:StringNotLike is true only when NO request value matches ANY p
    orgpaths: .../sportsou/costcenter/       page table: Match     model: No match
```

```
  orgpaths: .../mediaou/costcenter/        page table: No match  model: Match
  orgpaths: key absent                     page table: No match  model: Match
```

The ARN rows use well-formed ARNs: the page's `ForAllValues:ArnLike` table prints `arn:aws::cloudfront:...` with an extra colon on both the policy and the request side, which matches itself but is not an ARN; I used the form in the policy JSON above it.

Mistakes the documentation warns about, run

These are consequences of the rules above. They are the model's answers, derived from the documented rules (not measurements from AWS). The `ec2:RunInstances` case is the page's own example: a statement is "evaluated against every resource that is required to launch the instance", and an instance-type key exists only on the instance.

```
py -X utf8 t_traps.py
```

```
1. ForAllValues on a request with no tags (the page: true 'if there are no context keys
   an Allow statement, no Null guard, request carries NO tag keys
   the same with "Null": {"aws:TagKeys": "false"}, request carries NO tag keys
   with the guard, a request tagging only 'environment'

2. One statement is checked against EVERY resource the action touches (the page's ec2:Ru
   StringLike (the page's 'does not work' policy), instance type t2.micro
   StringLike (the page's 'does not work' policy), instance type m2.micro
   StringLikeIfExists, instance type t2.micro
   StringLikeIfExists, instance type m2.micro

3. Negated operators and a missing key (the page: negated operators are true when the ke
   Deny + StringNotEquals region, request has NO aws:RequestedRegion
   Deny + StringNotEqualsIfExists region, request has NO aws:RequestedRegion (the page:
   Deny + StringNotEquals region, request region eu-west-1

4. Things the model refuses to guess
   policy variable inside a Numeric operator (the page: not supported)
   multivalued key without ForAllValues / ForAnyValue
   an operator it does not implement (BinaryEquals)

5. A malformed policy is an error, not an empty policy
   a statement with no Effect
```

The practical rules: guard every `ForAllValues` `Allow` with `Null`; use `IfExists` when one statement spans resource types that do not all carry the key; remember that a `Deny` with a negated operator also catches requests that lack the key; and know that `aws:SourceIp` is the caller's IP address, so a service acting on your behalf (the page's example is CloudFormation calling EC2) is seen with the *service's* address. The documentation says there is "currently no way" to pass the original through.

Permissions Boundaries and Session Policies

Both are *ceilings*, not grants. A **permissions boundary** is a managed policy attached to a user or role "to set the maximum permissions that an identity-based policy can grant". The documentation's example gives user Shirley this boundary:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*",
        "cloudwatch:*",
        "ec2:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Shirley "can never perform operations in any other service, including IAM, even if she has a permissions policy that allows it", and the boundary alone gives her nothing. Both statements ran above (the first four rows of the table). A **session policy** is passed when you create temporary credentials (`AssumeRole`'s `Policy` and `PolicyArns` parameters). "The resulting session's permissions are the intersection of the role's identity-based policy and the session policies. ... You cannot use session policies to grant more permissions than those allowed by the identity-based policy of the role". It is how a tool that holds a powerful role hands a narrower one to a job:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::reports/*"
    }
  ]
}
```

Delegating without losing control is the use that justifies boundaries. In the documentation's example the administrator lets Zhang create users, but only if every new user carries the boundary `XCompanyBoundaries`. The trick is a condition on the request itself (`iam:PermissionsBoundary`) in *Zhang's own* boundary, plus a `Deny` on removing boundaries. These statements (modelled on the page's) are the last four rows of the Exercise 1 table: creating a user without a boundary is denied, with the right boundary allowed, with a different boundary denied, and removing a boundary denied.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateOrChangeOnlyWithBoundary",
      "Effect": "Allow",
      "Action": [
        "iam:AttachUserPolicy",
        "iam:CreateUser",
        "iam:PutUserPermissionsBoundary",
        "iam:PutUserPolicy"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PermissionsBoundary": "arn:aws:iam::111111111111:policy/XCompanyBoundarie
        }
      }
    },
    {
      "Sid": "NoBoundaryUserDelete",
      "Effect": "Deny",
```

```
      "Action": "iam:DeleteUserPermissionsBoundary",
      "Resource": "*"
    }
  ]
}
```

Two warnings from the page. First, never use `NotPrincipal` with `Deny` in a resource-based policy for principals that have boundaries: "The `NotPrincipal` element with a `Deny` effect will always deny any IAM principal that has a permissions boundary policy attached"; use `ArnNotEquals` on `aws:PrincipalArn` instead. Second, **a boundary's implicit deny does not block a resource policy that names the user**: in the page's example, a Secrets Manager secret whose policy names Nikhil lets him read it even though his boundary never mentions Secrets Manager (a row in the table above). A boundary is not a way to stop access that resource policies grant directly: use an explicit `Deny` for that.

Roles and STS

A role has two policies, and the `AssumeRole` page says so: "a role trust policy that specifies *who* can assume the role, and a permissions policy that specifies *what* can be done with the role". The trust policy is a resource-based policy. Calling `AssumeRole` returns temporary credentials (an access key ID, a secret key and a session token).

Who may assume a role (from the same page):

- **Same account:** either the trust policy names the user directly, or the trust policy trusts the account and the user has an identity policy that allows `sts:AssumeRole` on the role. "When a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required."
- **Another account:** the role's trust policy must trust that account *and* "A user who wants to access a role in a different account must also have permissions that are delegated from the account administrator": an identity policy in their own account allowing `AssumeRole` on the role's ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:user/dev"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Run here (level 1). The assumption rules, the external ID, the service-principal case and the session-policy rule from the pages, through the model. The last three rows are marked *derived*: they combine two documented rules and report whether the `Deny` statement applies.

```
py -X utf8 t_trust.py
```

```
case
same account: trust names the USER directly, user has no identity policy
same account: trust names the ACCOUNT, user has no identity policy
same account: trust names the ACCOUNT, user's policy allows sts:AssumeRole on this role
same account: trust names the account, user's policy allows a DIFFERENT role
same account: the trust policy does not name the user or account at all
same account: trust allows, but the user has an explicit Deny on sts:AssumeRole
cross account: trust in B names account A, user in A may AssumeRole on the role
cross account: trust in B names account A, but the user has no policy in A
cross account: user may AssumeRole in A, but B's trust policy does not name A
cross account: even if B's trust names the user ARN, the user still needs a policy in A
external ID: AssumeRole sent with ExternalId 12345 (the trust policy requires it)
external ID: AssumeRole sent with the other customer's ExternalId 67890
external ID: AssumeRole sent with no ExternalId
no external ID condition: the SAME caller with ExternalId 67890 is let in (the confused
service principal, no condition: CloudTrail acting for the owner's account 111122223333
service principal, no condition: CloudTrail acting for an UNKNOWN account 999999999999
with aws:SourceAccount=111122223333: CloudTrail acting for 111122223333
with aws:SourceAccount=111122223333: CloudTrail acting for 999999999999
with aws:SourceAccount=111122223333: a request that carries no SourceAccount at all
session policy allows s3:GetObject only; request s3:GetObject
session policy allows s3:GetObject only; request s3:PutObject (role allows it)
session policy allows ec2:*, the role does not -> a session policy cannot add permission
no session policy passed -> the role's own permissions apply
Deny if MFA age > 3600: caller used access keys (the key is absent) -> the Deny does NOT
same Deny written with NumericGreaterThanIfExists -> the Deny applies to the keys-only c
same Deny written with NumericGreaterThan: caller with a 5000-second-old MFA session ->
```

```
26 cases, mismatches: 0
```

All 26 match (and a flipped control reports one mismatch). Read the *derived* rows twice. `aws:MultiFactorAuthAge` is, per the Condition page, absent when the caller uses long-term access keys; a numeric operator on a missing key is false; so a `Deny` that says "deny if the MFA is older than an hour" simply **does not apply to the caller with no MFA at all**. Adding `IfExists` makes the missing key count as true and closes the gap. (Whether you want that `Deny` at all is a separate decision: it would also catch every role session that has no MFA.)

Session length and what the SDK checks

The `DurationSeconds` rules, as written on the page, as a function:

```
py -X utf8 duration.py
```

```
DurationSeconds  role max  chained  result
(not given)      3600      False    ok: 3600 s (1.00 h)
3600             3600      False    ok: 3600 s (1.00 h)
7200             3600      False    fails: above the role's maximum session duration (36
7200             14400     False    ok: 7200 s (2.00 h)
43200            43200     False    ok: 43200 s (12.00 h)
43200            14400     False    fails: above the role's maximum session duration (14
899              3600      False    fails: below the 900-second minimum
900              3600      False    ok: 900 s (0.25 h)
3600             43200     True     ok: 3600 s (1.00 h)
7200             43200     True     fails: role chaining limits the session to one hour
(not given)      43200     True     ok: 3600 s (1.00 h)
(not given)      43200     False    ok: 3600 s (1.00 h)
```

The other documented limits: a session policy is plain text of at most 2,048 characters (inline and managed together), up to 10 managed policy ARNs; `RoleSessionName` is 2 to 64 characters from `[\w+=,.@-]`; `ExternalId` is 2 to 1,224 characters from `[\w+=,.@:/-]`; `SourceIdentity` persists across chained sessions and may not start with `aws:`. Also "In cross-account scenarios, the role session name is visible to, and can be logged by the account that owns the role": do not put anything sensitive in it.

Run here (level 1). How much of that does the AWS SDK check before sending anything? I called `assume_role` through botocore's test `Stubber` (no network) with values that break each documented rule:

```
py -X utf8 sts_params.py
```

```
botocore 1.43.112, boto3 1.43.112; no network (Stubber)

case                                  botocore (client side)
a valid call                          accepted
RoleSessionName of 1 character        REJECTED (valid min length: 2)
RoleSessionName of 65 characters      accepted
RoleSessionName with a space          accepted
DurationSeconds 899                   REJECTED (valid min value: 900)
DurationSeconds 43201                 accepted
DurationSeconds 3600                  accepted
ExternalId of 1 character             REJECTED (valid min length: 2)
ExternalId with a space               accepted
session Policy of 2,049 characters    accepted
11 managed session policy ARNs        accepted
TokenCode of 5 digits                 REJECTED (valid min length: 6)
TokenCode of 6 letters                accepted
RoleArn of 19 characters              REJECTED (valid min length: 20)

cases the documentation forbids but botocore accepted without complaint: 7
```

botocore enforces the minimum lengths and the minimum duration and nothing else. A 65-character session name, a 43,201-second duration, a space in a session name or external ID, a 2,049-character session policy and 11 managed policy ARNs all went through; AWS would reject them (according to the page), but you would find out only from the service's error. This is a client-side check in the installed SDK version, not a statement about AWS: I could not send a request to see AWS's response.

Cross-Account Access, and the Confused Deputy

The Carlos example from the Cross-account page is worth restating, because it is the cleanest picture of the two-evaluation rule. A `Demo` role in account 111111111111 may use S3, but its policy denies any bucket with `log` in its name. A bucket policy in account 222222222222 allows

the `Demo` role to get and put (not delete) objects. Putting to the `production-logs` bucket fails in *his own account* (explicit deny). Putting to the production bucket works because both accounts allow it. Deleting an object fails in the *bucket's* account because the bucket policy never allowed deletes, even though his own policy would. All three are rows in the Exercise 1 table, with two more: a bucket policy that allows the role but a role that has no identity policy (denied), and the reverse (denied).

The external ID

Giving a third party a role in your account has a trap. The role ARN "isn't a secret", so another customer of the same vendor can tell the vendor to use *your* role ARN, and the vendor, trusted by your role, acts on your account for them: the vendor is a **confused deputy**. The fix is an `sts:ExternalId` condition in the trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "12345"
        }
      }
    }
  ]
}
```

The documentation is specific about who chooses the value: it "must be unique among Example Corp's customers and controlled by Example Corp, not its customers. This is why you get it from Example Corp and you don't come up with it on your own." The model's rows above show the effect: with the condition, only a caller that sends `12345` gets in; without it, the same caller sending the other customer's ID is let in.

Services as principals

The same problem exists between AWS services. If a bucket policy trusts `cloudtrail.amazonaws.com` with no condition, a CloudTrail trail in *any* account that knows the bucket name could write there. The documentation recommends `aws:SourceArn`, `aws:SourceAccount`, `aws:SourceOrgID` or `aws:SourceOrgPaths` "whenever access to one of your resources is granted to an AWS service principal":

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSCloudTrailWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudtrail.amazonaws.com"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1/*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111111111111"
        }
      }
    }
  ]
}
```

In the table above, with this condition the owner's account is let in and an unknown account (or a request carrying no `SourceAccount` at all) is not. The page adds that some services have their own protection: for a KMS grant given to an AWS service, "AWS KMS uses the encryption context associated with the resource, and the key grant". Resource control policies can enforce the same check across an organization (an example RCP is on the page; I did not run it, and my model does not handle RCPs).

Resource-Based Policies in Practice

Which resources take a resource-based policy? The documentation's Access Analyzer page lists the resource types it checks for external access, which is a good list of the places a resource policy can open a door: S3 buckets (and directory buckets), IAM roles (trust policies), KMS keys,

Lambda functions and layers, SQS queues, Secrets Manager secrets, SNS topics, EBS and RDS snapshots, ECR repositories, EFS file systems, and DynamoDB tables and streams. The commonest design question is the one in the table at the start of this chapter: *do I put the grant in the identity policy, the resource policy, or both?* Same account, for most resources, either is enough; for another account you need both; for a role trust policy or a KMS key policy the resource side is mandatory.

The Condition page's SNS-to-SQS example shows the usual shape for a service writing to a queue: the queue policy names `sns.amazonaws.com` and an `ArnEquals` on `aws:SourceArn` pins it to one topic. That is the confused-deputy guard from the previous section in its most common form. (From the documentation, not run; *AWS Intermediate/Advanced 9* builds it.)

Access Analyzer and the Policy Simulator

Two AWS tools check policies. Neither could be used here (level 3), so this is the documentation, with its caveats.

IAM Access Analyzer does six things (all from its overview page):

- **External access analyzers** find resources shared outside your *zone of trust* (your account or organization) by reasoning about resource policies, and generate a finding for each. They look only at the Region where the analyzer is enabled, so you need one per Region. A new or changed policy is analysed "within about 30 minutes".
- **Internal access analyzers** show which principals inside your organization can reach selected critical resources (S3 buckets, DynamoDB tables, RDS snapshots among them).
- **Unused access analyzers** report unused roles, access keys and passwords and, for active roles, unused services and actions.
- **Policy validation** checks a policy against IAM grammar and best practices.
- **Custom policy checks** test a policy against standards you set, for example "this update grants no new access" or "these critical actions are not allowed".
- **Policy generation** builds a policy from the actions a user or role used in your CloudTrail logs over a date range.

On cost, the page says charges apply to unused-access analysis (per role and user analyzed, per analyzer, per month), internal-access analysis (per resource monitored) and custom policy checks (per API request); it points to the pricing page for the rest, which I did not read.

The IAM policy simulator evaluates "identity-based policies, IAM permissions boundaries, service control policies (SCPs), and resource-based policies that you provide" without sending a real request. Its documented limits matter: "The policy simulator results can differ from your live AWS environment"; it "does not support resource control policies (RCPs)" and can differ for VPC endpoint policies, role chaining or multiple resource-based policies on one resource; for SCPs it gives the decision but "does not surface the matched statements". You supply the context key values that your conditions use, or the condition is not satisfied and the action is denied. It is a good check, not a final one: the page itself recommends confirming against the live environment.

Offline Linting with parliament

You can catch a class of mistakes before any AWS call. `parliament` is an open-source linter that ships its own copy of AWS's action and condition-key lists. **Run here (level 1):**

```
py -X utf8 lint_policies.py
```

```
parliament 1.6.4

-- core checks
  correct
       (no findings)
  misspelt action (GetObjekt)
       UNKNOWN_ACTION: Unknown action s3:GetObjekt
  bucket ARN where an object ARN is needed
       RESOURCE_MISMATCH: [{'action': 's3:GetObject', 'required_format': 'arn:*:s3:::*..
  Effect misspelt (Allowed)
       MALFORMED: Unknown Effect used. Effect must be either Allow or Deny
  no Resource
       MALFORMED: Statement contains neither Resource nor NotResource
  condition operator misspelt
       RESOURCE_STAR: ['s3:GetObject']
       UNKNOWN_OPERATOR: StringEqualz
  EC2 condition key on an S3 action
       RESOURCE_STAR: ['s3:GetObject']
       UNKNOWN_CONDITION_FOR_ACTION: Unknown condition ec2:InstanceType for action s3:Ge
  iam:* on every resource
       RESOURCE_STAR: ['iam:AddClientIDToOpenIDConnectProvider', 'iam:AddRoleToIns...
  Action * on Resource *
       (no findings)
```

```
  Chapter 1's default-controls SCP (actions added to AWS in recent years)
       (no findings)

-- community auditors (off by default), on a policy that can escalate privileges
   core only:         ["RESOURCE_STAR: ['iam:PassRole', 'lambda:CreateFunction', 'lambda
   with community on: ['PERMISSIONS_MANAGEMENT_ACTIONS', 'PRIVILEGE_ESCALATION', "RESOUR

-- the limit: a role trust policy is not an identity policy
   ['MALFORMED: Statement contains neither Resource nor NotResource']
```

What it caught: a misspelt action, a bucket ARN where an object action needs an object ARN, a misspelt effect, a missing `Resource`, a misspelt condition operator, an EC2 condition key on an S3 action, a `Resource: "*"` on `iam:*`, and (only with the community auditors switched on) a combination of actions that can escalate privileges. What it did **not** do: it reported nothing for `Action: *` on `Resource: *`; it reported nothing for the SCP from *AWS Intermediate/Advanced 1* (it knows the `account:CloseAccount` action, but SCP-specific rules are outside what it checks); and because it treats every input as an identity policy it called a role trust policy "malformed". It also called the documentation's own `iam:PermissionsBoundary` condition "mismatched type but usable". Treat its output as a first filter. The data inside it is as current as the installed package, which I did not date.

The Least-Privilege Workflow

"Least privilege" is a process, not a policy you write once: start with a policy that works, observe what the workload really does, cut the policy to that, and keep checking. The AWS tools that fit each step are on the documentation page above: *generate* a policy from CloudTrail activity, *validate* it, *check* it against your standards, and watch *unused access* findings afterwards. I cannot run those, so here is the method on **invented data**: twelve events in the general shape of CloudTrail records (the log is made up; I have no CloudTrail to confirm that real events carry exactly these names), turned into a policy, linted, replayed and attacked.

```
py -X utf8 least_priv.py (steps 1 and 2)
```

```
12 invented events written to events.json

1. naive actions (service:eventName): ['dynamodb:GetItem', 'dynamodb:PutItem', 'dynamodb
2. parliament on the naive policy:
```

```
        ('UNKNOWN_ACTION', 'Unknown action s3:ListObjects')
        ('UNKNOWN_ACTION', 'Unknown action s3:ListObjectsV2')
```

The first lesson is in step 2. An event's name is *not always* an IAM action: the log records `ListObjects` and `ListObjectsV2`, and parliament says `s3:ListObjects` and `s3:ListObjectsV2` are not actions (the IAM action for listing a bucket is `s3:ListBucket`). I fixed those two by hand, from my own knowledge, accepting the fix only because parliament then recognised the action. Next, resources are taken from the request parameters so the policy is scoped (a bucket for `ListBucket`, `bucket/*` for object actions):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query"
      ],
      "Resource": [
        "arn:aws:dynamodb:eu-west-1:111111111111:table/orders"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::reports"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::reports-out/*"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::reports/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage"
      ],
      "Resource": [
        "arn:aws:sqs:eu-west-1:111111111111:jobs"
      ]
    }
  ]
}
```

```
py -X utf8 least_priv.py (steps 5 to 8)
```

```
5. parliament on the derived policy: (no findings) | community auditors: (no findings)

6. replay: 12 of 12 logged requests are allowed by the derived policy (model)
7. requests that were NOT in the log:
      s3:DeleteObject       arn:aws:s3:::reports/a.csv                                  der
      s3:GetObject          arn:aws:s3:::other-bucket/a.csv                             der
      s3:PutObject          arn:aws:s3:::reports/new.csv                                der
      dynamodb:DeleteTable  arn:aws:dynamodb:eu-west-1:111111111111:table/orders        der
      dynamodb:GetItem      arn:aws:dynamodb:eu-west-1:111111111111:table/users         der
      sqs:PurgeQueue        arn:aws:sqs:eu-west-1:111111111111:jobs                     der
      iam:CreateUser        *                                                           der

8. the action lists inside parliament: s3:* = 163 actions, dynamodb:* = 71 actions, sqs:*
   a broad policy (s3:*, dynamodb:*, sqs:* on Resource *) allows 254 actions on every re
   3.1% of the actions, and each limited to the resources actually used
```

Every logged request is allowed (a policy that breaks the workload is no good), and the requests that were *not* in the log are denied, including the neighbouring ones that a broad policy would have allowed: another bucket, another table, a destructive action on the right table. The last step counts what a "broad" version would permit using parliament's own action lists: `s3:*`, `dynamodb:*` and `sqs:*` on every resource is 254 actions; the derived policy is 8 actions on 5 resources. **The caveats are the point:** a log shows what happened in the window you looked at, so a monthly job or an error-handling path that did not run is missing and will fail when it first runs; read access can be wider than the log shows (the log lists each object read, the policy allows the prefix); and the derived policy contains *no conditions*, which is where the next gains (a source VPC, an MFA requirement, a required tag) come from. I believe object-level S3 and DynamoDB data events must be switched on in CloudTrail before they are logged at all, but I did not check that today.

Cost and Clean-Up

Nothing in this chapter creates anything in AWS, so there is nothing to delete and nothing to pay. The Access Analyzer features that charge are listed above; check the pricing pages before enabling them. When you do try these in a real account, do it in the sandbox account from *AWS Intermediate/Advanced 1*, with a budget alert (*AWS Fundamentals 9*) in place first.

WHAT WAS AND WAS NOT CHECKED

Run here: the evaluation model against 31 documented outcomes, 49 documented condition results, 26 role and cross-account cases and the derived MFA cases; botocore's parameter checks (14 cases, offline); the documented session-duration rules; parliament on 12 policies; and the log-to-policy method on invented events. **From the documentation, not run:** every statement about how AWS behaves, including Access Analyzer, the policy simulator and RCPs. **Unverified:** how a plain negated operator combines several values; that real CloudTrail events have the names I invented; the dating of parliament's data. **Not run at all:** any real request, any role assumption, Access Analyzer, the simulator, other SDK versions or other languages' SDKs.

Practice

EXERCISE 1

Write an evaluator for the documented evaluation order (explicit deny first, SCP levels, resource-based policy with the principal-type rules, identity policy, permissions boundary, session policy; two evaluations for cross-account requests) that raises an error for anything it does not model. Run it against the documentation's worked examples (Shirley, Carlos, Nikhil, the principal-type table, the SCP, boundary and identity combinations, Maria and Zhang), writing the expected result first, then flip one expectation as a control.

View solution

EXERCISE 2

Add condition evaluation (string, ARN, IP, Boolean, numeric and date operators, `IfExists`, `Null`, `ForAllValues` and `ForAnyValue`, policy variables, the missing-key rules). Run every table from the AWS condition pages as a test and report any row you believe the page has wrong. Then show five traps: `ForAllValues` without a `Null` guard, `ec2:RunInstances` across several resources, a `Deny` with a negated operator on a missing key, and the cases your evaluator refuses.

View solution

EXERCISE 3

Model role assumption (same account, cross-account, external ID, service principals with `aws:SourceAccount`, session policies, the MFA-age trap), check which `AssumeRole` limits the AWS SDK enforces offline, apply the documented session-duration rules, lint policies with `parliament`, and derive a least-privilege policy from an event log, then replay the log and try requests that were never logged.

View solution

CHAPTER 2 QUICK REFERENCE

- Default is deny (except the account root user). Order: any explicit Deny, RCPs, SCPs, resource-based policy, identity-based policy, permissions boundary, session policy. A Deny anywhere wins
- Same account: identity and resource policies are a **union**; identity policy, boundary and session policy are an **intersection**; SCP and RCP allows are needed too. Another account: **both** accounts must allow
- A resource policy naming a **user**, a **role session** or a federated session grants on its own; one naming a **role** is still limited by the boundary and session policy. Trust policies and KMS key policies must name

the principal

- Missing key: ordinary operators are false, negated operators true, `IfExists` true, `ForAllValues` true, `ForAnyValue` false. Guard `ForAllValues` with `Null`
- Policy variables work in String, ARN and Boolean operators, not in Numeric, Date, Binary, IP or `Null`. Multivalued keys need a set operator
- One statement is checked against every resource the action touches: use `IfExists` for keys only some resources carry
- Boundaries and session policies only *limit*. A boundary's implicit deny does not stop a resource policy that names the user; use an explicit Deny. Never `NotPrincipal` + `Deny` with boundaries
- `AssumeRole`: session 900 s up to the role's maximum (1 to 12 h), default 3,600 s, role chaining 1 h; session policy at most 2,048 characters; botocore checked only minimum lengths offline
- Third party in your account: `sts:ExternalId` in the trust policy, a value the **third party** generates. AWS service in your resource policy: `aws:SourceArn` or `aws:SourceAccount`
- Access Analyzer: external, internal and unused-access analyzers, policy validation, custom checks, generation from CloudTrail. The simulator can differ from live and does not support RCPs
- Least privilege is a loop: start broad, observe, derive, lint, replay, restrict resources, add conditions, watch unused access. Event names are not always action names

COMING NEXT

Chapter 3 is encryption and secrets: KMS keys and key policies (the other policy that must name its principals), envelope encryption, Secrets Manager versus Parameter Store, rotation and certificates. Start with *AWS Intermediate/Advanced 3*.

AWS Intermediate/Advanced

Chapter 3 · KMS, Secrets Manager and Parameter Store

Almost everything you store in AWS is encrypted, and most of the time you never see it happen. This chapter is about the times you do see it: when a policy has to say who may use a key, when a key is rotated or scheduled for deletion, when an application needs a database password and you must decide where it lives, and when an auditor asks who used what. It covers AWS KMS (the key service), the defaults for S3, EBS and RDS, **envelope encryption** (the pattern that makes encrypting big data with a small key service possible), Secrets Manager, Parameter Store, certificates, and the CloudTrail records that prove what happened.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS CLI on the machine that wrote this course, so nothing here was run against AWS.

- **Run here (level 1):** Python 3.14.0 on Windows 11, in a scratch folder outside this site, with `cryptography` 50.0.2 (AES-GCM), the `aws-encryption-sdk` 4.0.7 library, `boto3` / `botocore` 1.43.112 (offline, through its test Stubber) and `parliament` 1.6.4 (an offline policy linter). Output is pasted from the real runs. **Docker is installed but its engine was not running**, so no container example was run.
- **Needs an AWS account (level 2):** none in this chapter. Nothing here creates, changes or costs anything.
- **From the documentation, not run (level 3):** everything about how AWS KMS, Secrets Manager, Parameter Store, ACM and CloudTrail behave. I read their pages and the pricing pages on 2026-10-12 through a page-fetch tool that summarises pages, so quotes are as that tool returned them. **AWS changes limits and prices often; check the current page before relying on a number.**

Three things here are **models or stand-ins, not AWS**: `LocalKms` (a teaching stand-in for the key service), the policy evaluator from *AWS Intermediate/Advanced 2* applied to key policies, and a small model of the Secrets Manager rotation protocol. Each says what it copies from the documentation and what it does not.

Encryption at Rest: What AWS Does Before You Ask

The defaults differ by service, and the differences matter when you are asked "is it encrypted?":

SERVICE	DEFAULT	WHAT YOU CAN AND CANNOT CHANGE (FROM THE DOCUMENTATION)
Amazon S3	"Starting January 5, 2023, all new object uploads to Amazon S3 are automatically encrypted at no additional cost" with SSE-S3 (AES-256)	You cannot disable encryption for new uploads. Objects uploaded *before* that date in an unencrypted bucket are not encrypted automatically: use S3 Batch Operations to make encrypted copies. CloudTrail records `"SSEApplied":"Default_SSE_S3"`
Amazon EBS	Not on by default: "encryption by default" is a **Region-specific account setting**. When enabled, new volumes and snapshot copies are encrypted	"Encryption by default has no effect on existing EBS volumes or snapshots." It cannot be disabled for individual volumes in that Region. The default key is the AWS managed key `aws/ebs`, which you can change to a customer managed key
Amazon RDS	You choose at creation (`--storage-encrypted`)	"You can only encrypt an Amazon RDS DB instance when you create it, not after." You cannot turn it off; you cannot change the KMS key of an existing encrypted instance. The workaround: snapshot, encrypted copy, restore (the page also mentions blue/green deployments). Logs, backups, snapshots and read replicas are encrypted too

One RDS detail shows why key management is an availability matter, not only a security one. If the KMS key becomes unusable (for example it is disabled), the page says an encrypted instance with backups enabled "goes into the `inaccessible-encryption-credentials-recoverable` state 2 hours after detection", stays stopped for seven days while you can re-enable the key and restart it, and after that is no longer usable and can only be restored from a snapshot or backup.

KMS Keys

A **KMS key** is a logical key in the AWS Key Management Service; its secret material is kept inside AWS's hardware security modules and is "designed never to be exported from the HSM in plaintext". You ask the service to use it; you do not get it. There are three kinds, and the documentation's comparison is the one to remember:

	CUSTOMER MANAGED KEY	AWS MANAGED KEY	AWS OWNED KEY
Who controls the key policy	You, exclusively	The service; you can view it	The AWS service only; you cannot even view it
In your account?	Yes	Yes, alias form `aws/<service>` (e.g. `aws/ebs`)	No (a service account)
Rotation	Optional; you set it	Required, every year (about 365 days); you cannot change it	The service decides
Audit in CloudTrail	Yes	Yes	Not viewable by you
Cost (as read 2026-10-12)	$1 per key per month (pro-rated hourly), plus requests	No monthly fee; per-use fee (some services pay it for you)	None
Sharing across accounts	Yes, through the key policy (below)	No: "You cannot share resources encrypted under an AWS managed key with other accounts"	Easy cross-account sharing

The documentation's advice is "use customer managed keys when control is important, but use AWS owned keys when convenience is most important". It also says AWS managed keys are "a legacy key type that is no longer being created for new AWS services as of 2021". Prices from the KMS pricing page: **$1 per month per key**, **$0.03 per 10,000 requests**, a free tier of 20,000 requests per month across all Regions (asymmetric operations are excluded from it), and **$1 per month extra for the first and second rotation of key material** (later rotations are not billed). The page named no Region for the request price. You refer to a key by key ID, key ARN, alias name (`alias/Name`) or alias ARN; to use a key in *another* account you must use the key ARN or alias ARN.

Envelope Encryption

KMS will encrypt at most **4,096 bytes** directly ("Encrypts plaintext of up to 4,096 bytes"). That is enough for a password and not enough for a file. The standard answer is **envelope encryption**: ask KMS for a *data key* (`GenerateDataKey` returns a plaintext copy and a copy encrypted under your KMS key), encrypt the big data locally with the plaintext key, throw the plaintext key away,

and store the encrypted data key next to the data. To read it back, ask KMS to decrypt the stored data key, then decrypt locally. The key service sees only a 32-byte key per object; the bulk bytes never leave your process.

I built this by hand with a **local stand-in for KMS** (`LocalKms`: it keeps keys in memory and copies the documented behaviours listed at the top of the file, such as the 4,096-byte limit, the key and key version recorded inside the ciphertext blob, the encryption context as authenticated data, and rotation that adds key material without touching data). It is **not KMS**. What is real is the AES-256-GCM encryption, the sizes and the failures. The core:

```
def envelope_encrypt(kms, kid, data, ctx):
    dk, wrapped = kms.generate_data_key(kid, ctx)
    nonce = os.urandom(12)
    ct = AESGCM(dk).encrypt(nonce, data, canon(ctx))
    return {"wrapped_key": wrapped, "nonce": nonce, "ciphertext": ct, "ctx": ctx}


def envelope_decrypt(kms, obj, ctx):
    dk = kms.decrypt(obj["wrapped_key"], ctx)
    return AESGCM(dk).decrypt(obj["nonce"], obj["ciphertext"], canon(ctx))
```

Run here (level 1).

```
py -X utf8 envelope.py
```

```
1. direct encryption has a small limit (4,096 bytes, as documented for KMS)
  encrypt 4,096 bytes directly                                          -> ok (4
  encrypt 4,097 bytes directly                                          -> KmsE

2. envelope encryption of 50 MB: only the 32-byte data key goes through the key service
  plaintext 52,428,800 bytes; ciphertext 52,428,816 bytes (+16); nonce 12; wrapped data k
  round trip identical: True; encrypt 0.01 s, decrypt 0.01 s on this machine (single run

3. the encryption context is bound to the data
  decrypt with the same context                                         -> ok (1
  decrypt with tenant t-18 instead of t-17                              -> KmsE
  decrypt with no context at all                                        -> KmsE
  decrypt after flipping ONE bit of the ciphertext                      -> Inva
  decrypt after swapping in another object's wrapped data key           -> Inva

4. rotation changes the KEY MATERIAL, not your data (the Rotate keys page)
```

```
    wrapped key made with key version 1; rotating the key ...
    key now has 2 versions
    decrypt the OLD object after rotation                                  -> ok (
    a NEW object is wrapped with key version 2
    re-encrypting only the wrapped data key moves it to version 2; bulk ciphertext hash be
    decrypt the object after the wrapped key was moved                     -> ok (
    decrypt an object still wrapped under the DESTROYED version 1          -> KmsE
    decrypt the object that had been moved to version 2                    -> ok (

  5. a key that is not enabled cannot unwrap data keys (the Deleting keys page: pending de
    a fresh process asks the key service to unwrap                         -> KmsE
    the running app that cached the plaintext data key                     -> ok (
    generate a NEW data key under the disabled key                         -> KmsE
    after re-enabling the key                                              -> ok (
```

What the run shows. **(1)** 4,096 bytes go through directly and 4,097 do not. **(2)** 50 MB is encrypted with 16 bytes of overhead (the GCM tag), a 12-byte nonce and a 72-byte wrapped key in this stand-in's format; the timing is one run on this machine and says nothing about KMS, whose calls cross a network. **(3)** The *encryption context* is bound to the data: the wrong tenant, or no context, fails; one flipped bit fails; swapping in another object's wrapped key fails. **(4)** Rotation (as documented) adds key material and keeps the old, so old data still decrypts; "Key rotation has no effect on the data that the KMS key protects. It does not rotate the data keys ... or re-encrypt any data". Re-encrypting just the *wrapped key* (the `ReEncrypt` idea) moves an object to the current key version while the bulk ciphertext stays byte-for-byte the same (same hash). And if old key material were destroyed, anything still wrapped under it would be lost. **(5)** A key that is not enabled cannot unwrap or generate data keys, yet an application that already holds a plaintext data key in memory keeps working until it restarts. That matches the documentation's warning that scheduling deletion "might not immediately affect data keys encrypted by the KMS key", and it is why a key change can look harmless for hours and then break a restart.

The same pattern in a real library

The **AWS Encryption SDK** implements envelope encryption for you. It can use a KMS key, or a raw AES key you hold, and I could only use the second kind (no AWS account). So this run shows what the SDK does, *not* how a KMS keyring behaves. **Run here (level 1):** aws-encryption-sdk 4.0.7 with a raw AES-256 key.

```
py -X utf8 esdk_demo.py
```

```
aws-encryption-sdk 4.0.7

plaintext 29,000 bytes -> message 29,674 bytes (+674)
message format version: SerializationVersion.V2 | algorithm: AES_256_GCM_HKDF_SHA512_COM
encrypted data keys in the header: 1 | commitment policy: REQUIRE_ENCRYPT_REQUIRE_DECRYP
encryption context in the header: {'app': 'invoices', 'tenant': 't-17', 'aws-crypto-publ
  the bytes b'invoices' appear in the ciphertext message: True
  the bytes b't-17' appear in the ciphertext message: True
  the bytes b'1,250.00' appear in the ciphertext message: False

decrypting
  with the right key                                                    -> ok (29
  with the wrong key (key-B only)                                       -> Decryp
  the decrypt result also returns the context: True
  after flipping one bit near the end of the message                    -> Invali
  after flipping one bit in the header                                  -> Invali

requiring the context on decrypt (the application's job: the SDK returns it, you must ch
  expect tenant t-17                                                    -> ok (co
  expect tenant t-99                                                    -> ValueE

two master keys: either one decrypts (a message can be readable by two keys, e.g. two Re
  encrypted data keys: 2
  decrypt with key-A only                                               -> ok (sh
  decrypt with key-B only                                               -> ok (sh
```

Three findings. **The encryption context is not secret**: the words `invoices` and `t-17` appear in clear text in the encrypted message (the KMS pages say the same: "Do not include confidential or sensitive information in this field"). The data itself does not appear. **The SDK returns the context on decrypt, but you must check it**: the "expect tenant t-99" line fails only because my code compared it. And damage anywhere fails: a flipped bit in the header or near the end of the message both fail (the library signs the message with its default algorithm suite; the algorithm name is in the output). A message can carry two encrypted data keys, so either of two keys (for example in two Regions) can decrypt it.

Key Policies, IAM Policies and Grants

KMS has an access model that surprises people who know IAM, and *AWS Intermediate/Advanced 2* prepared you for it: **a KMS key policy is a resource policy that must name its principals**. The documentation:

- "Key policies are the primary way to control access to KMS keys. Every KMS key must have exactly one key policy."
- "No AWS principal, including the account root user or key creator, has any permissions to a KMS key unless they are explicitly allowed, and never denied, in a key policy, IAM policy, or grant."
- "Unless the key policy explicitly allows it, you cannot use IAM policies to *allow* access to a KMS key. ... (You can use an IAM policy to *deny* a permission to a KMS key without permission from a key policy.)"
- "Unlike IAM policies, which are global, key policies are Regional."

The default key policy for a key created through the API is one statement. The principal is the account *root*, and that does not give the root user the key: "When the principal in a key policy statement is the account principal, the policy statement doesn't give any IAM principal permission to use the KMS key. Instead, it allows the account to use IAM policies to *delegate* the permissions". Delete it and "IAM policies that allow access to the key are ineffective, although IAM policies that deny access to the key are still effective" (and you risk an unmanageable key).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    }
  ]
}
```

The console adds more: a **key administrators** statement (manage the key, *not* use it), a **key users** statement (use it for cryptographic operations), and one that lets users create grants for AWS services, only when `kms:GrantIsForAWSResource` is true:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow access for Key Administrators",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/KeyAdmin"
      },
      "Action": [
        "kms:Create*",
        "kms:Describe*",
        "kms:Enable*",
        "kms:List*",
        "kms:Put*",
        "kms:Update*",
        "kms:Revoke*",
        "kms:Disable*",
        "kms:Get*",
        "kms:Delete*",
        "kms:TagResource",
        "kms:UntagResource",
        "kms:ScheduleKeyDeletion",
        "kms:CancelKeyDeletion",
        "kms:RotateKeyOnDemand"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/KeyUser"
      },
      "Action": [
```

```
          "kms:Encrypt",
          "kms:Decrypt",
          "kms:ReEncrypt*",
          "kms:GenerateDataKey*",
          "kms:DescribeKey"
        ],
        "Resource": "*"
      },
      {
        "Sid": "Allow attachment of persistent resources",
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::111122223333:role/KeyUser"
        },
        "Action": [
          "kms:CreateGrant",
          "kms:ListGrants",
          "kms:RevokeGrant"
        ],
        "Resource": "*",
        "Condition": {
          "Bool": {
            "kms:GrantIsForAWSResource": true
          }
        }
      }
    ]
  }
```

A warning on the page that is easy to miss: "Because key administrators have permission to change the key policy and create grants, they can give themselves and others AWS KMS permissions not specified in this policy." Separating administrators from users stops *accidents*, not a determined administrator.

Grants

A **grant** is a second, temporary route to a key. It gives a grantee (an IAM identity, or an AWS service principal) specific operations on exactly one key; it can allow but never deny; it needs no change to the key policy or any IAM policy; and "an authorized principal can delete the grant (retire or revoke it)", which removes everything it allowed. AWS services use them: they create a grant on your behalf, use it, and retire it. Details from the page: the KMS API is eventually consistent, so a new grant may not work at once ("typically less than a few seconds ... in some

cases it can take several minutes"); a **grant token** lets you use it immediately; and there is a limit of 50,000 grants per key. Treat `kms:CreateGrant` like `kms:PutKeyPolicy`: "Permission to create grants has security implications".

The rules as tests

I ran these rules through the *AWS Intermediate/Advanced 2* evaluator, using its "this resource policy must explicitly allow the principal" mode (`explicit_resource_allow`), which was built for exactly this case, and added grants and key state around it. As before, each expected answer comes from a sentence on the pages and was written first. (Level 1: a model of the documented rules, not AWS.)

```
py -X utf8 kms_policy_model.py
```

```
case (source: AWS KMS developer guide page named in the script)
default key policy (root statement) + IAM policy allows kms:Decrypt for the role -> Decry
default key policy, role has NO IAM policy -> Decrypt (the root statement only DELEGATES
key policy names only one user (no root statement); IAM policy allows the role -> Decrypt
key policy names the role directly for Decrypt; role has no IAM policy -> Decrypt
key policy names the role for Decrypt only -> Encrypt (not listed)
IAM Deny, and the key policy has no statement for the role at all -> Decrypt (a Deny nee
key policy has an explicit Deny for the role even though IAM allows -> Decrypt
administrator role: kms:Decrypt (the admins statement has no cryptographic actions)
administrator role: kms:PutKeyPolicy
administrator role: kms:CreateGrant (kms:Create* includes it)
administrator role: kms:ScheduleKeyDeletion
key user role: kms:Decrypt
key user role: kms:GenerateDataKeyWithoutPlaintext (GenerateDataKey* wildcard)
key user role: kms:PutKeyPolicy
key user role: kms:ScheduleKeyDeletion
the warning: an administrator creates a grant for themselves, THEN calls kms:Decrypt
key policy names account B's root AND B's role has an IAM policy for the key ARN -> Decry
key policy names account B's root, but B's role has no IAM policy -> Decrypt
B's role has an IAM policy, but the key policy does not name account B -> Decrypt
a grant for the role, and NO key policy statement or IAM policy for it -> Decrypt
same grant -> Encrypt (not among the grant's operations)
the grant, but an IAM Deny on kms:Decrypt -> Decrypt (a grant cannot override a Deny)
the grant is retired (removed from the list) -> Decrypt
key is pending deletion; policies and grants all allow -> Decrypt
key is pending deletion -> kms:CancelKeyDeletion by the administrator (management, not c
a grant with a constraint -> the model refuses (not modelled)
key user calls kms:CreateGrant directly (kms:GrantIsForAWSResource is absent)
key user creates the grant through an AWS service (kms:GrantIsForAWSResource = true)
```

```
  identity policy requires context app=invoices; request carries app=invoices -> Decrypt
  same policy; request carries app=payroll -> Decrypt
  same policy; request carries no encryption context at all -> Decrypt (the key is missing

  31 cases, mismatches: 0
```

All 31 match; the control with one flipped expectation reports `31 cases, mismatches: 1`. Read it as a checklist. The root statement *delegates* (no IAM policy, no access). A key policy that names only one user makes everyone else's IAM allows useless. An administrator cannot decrypt, but can put a key policy, create a grant and schedule deletion, and the "warning" row shows the two-step path from administrator to user. A cross-account request needs *both* the key policy naming the other account and an IAM policy in that account. A grant works with no policy statement but cannot beat a `Deny`. A key pending deletion refuses cryptographic operations whatever allows them. Grants with constraints are refused by the model (not implemented).

The cross-account pair, as policy files. In the key's account, the key policy names the other account; in the other account, a role needs its own policy naming the key ARN:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow account B to use the key",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:root"
      },
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey"
      ],
      "Resource": "*"
```

```
    }
  ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey"
      ],
      "Resource": "arn:aws:kms:eu-west-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456
    }
  ]
}
```

One more control you can put in an *identity* policy: bind use of a key to an encryption context with the condition key form `kms:EncryptionContext:<key>` (the last three rows of the table above: the matching context is allowed, a different one is denied, and a request with no context at all is denied because a missing key makes the condition false):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:eu-west-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:app": "invoices"
        }
      }
    }
```

```
    ]
  }
```

Rotation and Deletion of Keys

Rotation (the Rotate keys page):

- Automatic rotation is **optional for customer managed keys**, on symmetric encryption keys whose key material AWS KMS generated. When you enable it, "by default ... AWS KMS generates new cryptographic material for the KMS key every year"; the default rotation period is 365 days, and you can set a custom period. **On-demand rotation** rotates immediately, even with automatic rotation off.
- Asymmetric keys, HMAC keys and keys in custom key stores cannot be rotated automatically or on demand; you create a new key and switch your applications or alias to it (**manual rotation**).
- Rotation changes only the *current key material*. "AWS KMS retains all key material for a KMS key with `AWS_KMS` origin, even if key rotation is disabled", the key ID and ARN do not change, and decryption "automatically chooses the correct version". It does **not** re-encrypt data or rotate data keys, and "will not mitigate the effect of a compromised data key".
- It is recorded: a `RotateKey` event in CloudTrail and a `KMS CMK Rotation` event in EventBridge. Cost: see above ($1 per month extra for the first two rotations).

The page is candid about why: KMS keys are mostly *wrapping keys* that are used rarely, so "they ... are almost never reused enough to risk key exhaustion", and you rotate mainly "due to business or contract rules or government regulations". The envelope-encryption run above is the reason it is cheap: rotation never touches the bulk data.

Deletion (the Deleting keys page) is the one operation you cannot undo:

- "After a KMS key is deleted, you can no longer decrypt the data that was encrypted under that KMS key, which means that data becomes unrecoverable." For a symmetric key this holds even if you have the same key material, because "metadata unique to each KMS key is cryptographically bound to each symmetric ciphertext".
- You *schedule* deletion with a waiting period of **7 to 30 days (default 30)**; the real wait "might be up to 24 hours longer". During it the key is *Pending deletion*: it cannot be used for cryptographic operations and is not rotated, and you can cancel. The page suggests disabling

the key if you are not sure, and a CloudWatch alarm that warns if anything tries to use a key during the waiting period.

- You can delete only customer managed keys. Closing the AWS account makes keys inaccessible and stops the billing.

Secrets Manager and Parameter Store

The Secrets Manager introduction contains a useful decision table of its own: "AWS credentials: we recommend IAM. Encryption keys: we recommend KMS. SSH keys: we recommend Amazon EC2 Instance Connect. Private keys and certificates: we recommend AWS Certificate Manager." What is left for Secrets Manager is "database credentials, application credentials, OAuth tokens, API keys, and other secrets", and its pitch is rotation: "you replace hard-coded credentials with a runtime call to the Secrets Manager service". Systems Manager **Parameter Store** holds configuration and small secrets more cheaply. The facts, from the documentation:

	SECRETS MANAGER	PARAMETER STORE, STANDARD	PARAMETER STORE, ADVANCED
Maximum value size	65,536 bytes	4 KB	8 KB
How many	500,000 secrets per Region	10,000 per account and Region	100,000 per account and Region
Cost (as read 2026-10-12)	$0.40 per secret per month and $0.05 per 10,000 API calls; the page also says a free plan is "available for 6 months after account creation". Encrypting with the free AWS managed key `aws/secretsmanager` costs nothing extra; your own KMS key is charged at KMS rates. No charge for secrets marked for deletion	"No additional charge"	"Charges apply" (the price was not read)
Rotation	Built in: managed rotation, or a Lambda function you run (charged at Lambda rates)	None built in	None built in
Versions	Up to 100 versions, staging labels (20 labels across all versions)	(Not read)	

	SECRETS MANAGER	PARAMETER STORE, STANDARD	PARAMETER STORE, ADVANCED
Policies and sharing	Resource-based policy up to 20,480 characters; replication to other Regions is an API operation	No parameter policies; no sharing across accounts	Parameter policies; sharing across accounts
Changing tier			Standard can be upgraded; advanced cannot be downgraded, because it would truncate the value to 4 KB, drop the policies, and "advanced and standard parameters use a different form of encryption"

Run here (level 1): will real key material fit? I generated RSA, EC and certificate PEM text and measured it. The costs below are arithmetic on the prices above (not a quote; no Region named; the free plan is not applied).

```
py -X utf8 secrets_compare.py
```

```
1. does it fit? (bytes of the PEM / JSON text)
  item                                  bytes  Parameter Store standard    Parameter S
  database credentials as JSON            113  fits                        fits
  EC P-256 private key (PEM)              241  fits                        fits
  RSA-2048 private key (PEM)             1704  fits                        fits
  RSA-4096 private key (PEM)             3272  fits                        fits
  RSA-2048 key + 3-certificate chain     4767  TOO BIG                     fits
  RSA-4096 key + 3-certificate chain     6335  TOO BIG                     fits

2. what reading a secret costs: arithmetic on the prices above (Secrets Manager only; Pa
  each of 200 container starts a day reads every secret once                        150
  each request reads every secret: 10 requests/s                                648,000
  the same 10 requests/s, with a 5-minute cache in each of 20 instances           4,320
  (25 secrets; the free-trial rules and Regional price differences were not stated on th
```

The fit table is a measurement, and it contains a trap: a 4,096-bit RSA key fits Parameter Store standard on its own (3,272 bytes), but the same key with a three-certificate chain does not

(6,335 bytes). The cost table is the lesson about *where you call*: reading 25 secrets on every request at 10 requests per second is about $3,250 a month in API calls, while a five-minute cache in 20 instances is about $32. Cache secrets in memory and re-read on a schedule or on authentication failure. (How rotation interacts with a cache is the next section.)

The rotation protocol

A rotation Lambda function does four jobs, and Secrets Manager tracks them with **staging labels**: `AWSCURRENT` is the version applications read; `AWSPENDING` is the candidate; `AWSPREVIOUS` is the last good one. From the page:

1. `createSecret` checks whether a version with this request token exists; if not it makes a new value and stores it with the label `AWSPENDING`. "Storing the new secret value in `AWSPENDING` helps ensure idempotency."
2. `setSecret` changes the credential in the database or service to match `AWSPENDING`. The function is "a privileged deputy", so before it acts it must check that `AWSCURRENT` is valid, that both versions are for the same user, and that the target host is the same, otherwise someone who can edit a secret could point the function at another resource (the confused deputy problem from *AWS Intermediate/Advanced 2*).
3. `testSecret` uses `AWSPENDING` against the database (the templates use read access).
4. `finishSecret` moves `AWSCURRENT` to the new version; Secrets Manager adds `AWSPREVIOUS` to the old one. "Don't remove `AWSPENDING` before this point". If `AWSPENDING` is left on a different version than `AWSCURRENT`, "any later invocation of rotation assumes that a previous rotation request is still in progress and returns an error".

Other documented rules: only the rotation function can rotate the secret (it cannot call another Lambda function to do it); a provisioned concurrency below 10 "can cause throttling"; and be careful what you log, because logs go to CloudWatch. I turned these into a small model:

```
    def createSecret():
        if token not in sec.versions:                    # idempotent: reuse the pending
            c = parse(sec.get())
            sec.put(f"{c['user']}:{pysecrets.token_urlsafe(8)}:{c['host']}", token, ["AW
    def setSecret():
        c, p = parse(sec.get()), parse(sec.versions[token]["value"])
        if not db.connect(c["user"], c["password"], c["host"]):
            raise SmError("abandoned: the AWSCURRENT credential is not valid")
        if (c["user"], c["host"]) != (p["user"], p["host"]):
            raise SmError(f"abandoned: AWSCURRENT and AWSPENDING are for different resou
```

```
        db.password = p["password"]
    def testSecret():
        p = parse(sec.versions[token]["value"])
        if not db.connect(p["user"], p["password"], p["host"]):
            raise SmError("the AWSPENDING credential does not work")
    def finishSecret():
        sec.move_current(token)
    for name, fn in (("createSecret", createSecret), ("setSecret", setSecret), ("testSec
```

Run here (level 1): a model of the documented protocol, not AWS. "My reading" marks two things the page does not state outright: that a retry with the same token resumes, and that a put which cannot make room under the 100-version quota fails.

```
py -X utf8 rotation_model.py
```

```
1. a normal rotation, step by step (the model prints the state after each step)
  labels: v1=AWSCURRENT                                app reading AWSCURRENT can co
  labels: v1=AWSCURRENT, v2=AWSPENDING                 app reading AWSCURRENT can co
  labels: v1=AWSCURRENT, v2=AWSPENDING                 app reading AWSCURRENT can co
  labels: v2=AWSCURRENT/AWSPENDING, v1=AWSPREVIOUS     app reading AWSCURRENT can co

2. failure inside setSecret, then a retry with the same token
  rotate(token v2), failing in setSecret                              -> SmError: simu
  labels: v1=AWSCURRENT, v2=AWSPENDING                 app reading AWSCURRENT can co
  retry rotate(token v2)                                              -> ok ['createS
  the retry reused the pending value: True
  labels: v2=AWSCURRENT/AWSPENDING, v1=AWSPREVIOUS     app reading AWSCURRENT can co

3. a rotation left in progress blocks a rotation with a different token
  first rotation fails in testSecret                                  -> SmError: simu
  a new rotation with token v3                                        -> SmError: a pr
  the same token v2 again                                             -> SmError: aba

4. the confused-deputy checks in setSecret
  an attacker edits the AWSPENDING version to user 'admin' on the same host -> SmError:
  database password unchanged: True
  an attacker points AWSPENDING at another host                       -> SmError: aba
  the AWSCURRENT credential no longer works                           -> SmError: aba

5. quotas (value size, and the 100-version limit with the 24-hour rule)
  put a value of 65,536 bytes                                         -> ok
```

```
    put a value of 65,537 bytes                                          -> SmError: sec
    an update every  5 min: failed at update 101 (after 8.3 h): LimitExceeded: 100 version
    an update every 10 min: failed at update 101 (after 16.7 h): LimitExceeded: 100 versio
    an update every 15 min for 3 days: 286 updates, 100 versions kept, never hit the limit
```

Four things to take from it. **First**, between `setSecret` and `finishSecret` the database already has the new password while `AWSCURRENT` still holds the old one, so an application that reads `AWSCURRENT` in that window cannot connect. That window is a consequence of the four steps as written (derived, not a quoted sentence); the documentation lists a second rotation strategy ("alternating users") and a managed rotation, which I did not read, so I do not describe how they avoid it. **Second**, a failure inside `setSecret` leaves the application untouched and a retry with the same token reuses the pending value. **Third**, a failure after the database change (the `testSecret` case) leaves `AWSCURRENT` wrong, and then even a retry is abandoned by the "is AWSCURRENT valid" check: a real operational hazard that the model makes visible, and why your application should retry with a fresh read. **Fourth**, the quota line: updating a secret more often than the page recommends (once every 10 minutes at most) eventually runs out of versions, because unlabelled versions younger than 24 hours are never removed; at one update every five minutes the model fails after 8.3 hours.

Certificates: ACM

AWS Certificate Manager "handles the complexity of creating, storing, and renewing public and private SSL/TLS X.509 certificates and keys". Facts from its overview page (level 3, not run): certificates can be issued by ACM or imported; wildcard certificates are supported; there is "no additional charge for SSL/TLS certificates that you manage with AWS Certificate Manager" (you pay for the resources that use them); **certificates** [illegible] and "you cannot copy a certificate between regions"; and to use one with CloudFront "you must request or import the certificate in the US East (N. Virginia) region". ACM certificates signed by AWS Private CA can be exported; for publicly trusted certificates on your own servers (EC2, say) the page points to the ACME protocol. I did not read the validity period or renewal rules, so I do not state them. *HTTPS/TLS Fundamentals* covers what the certificate itself is.

Secrets in Environment Variables

"Put it in an environment variable" is the common first step away from hard-coding. It is better than source code and worse than most people think. These are **real experiments** with child processes on this Windows machine (the password is made up):

```
py -X utf8 secret_leaks.py
```

```
1. an environment variable is inherited by every child process, and by their children
   child sees: hunter2
   grandchild sees: hunter2
   child started with the variable removed: None

2. a 'debug dump' of the environment leaks it; redacting by key name misses a password i
   raw dump contains the secret: True
   after redacting variable NAMES that look sensitive: {"DATABASE_URL": "postgres://app:
   the secret still appears: True (inside DATABASE_URL: the name has no PASS/SECRET/TOKE

3. a command-line argument is readable by any other process of the same user, while the
   another process reads its command line: <python.exe> -c "import time; time.sleep(25)"

4. an alternative: pass a FILE PATH (not the secret), or send the secret over a pipe at
   another process reads its command line: <python.exe> -c "import sys,time; print('read
   the secret appears in that command line: False
   child got the secret over a pipe, length 7 | environment copy in the child: None
   (a file or a pipe still has to be protected: the file needs permissions, and anything
```

What they show. **Inheritance:** every child and grandchild sees the variable unless you remove it deliberately. **Dumps:** a diagnostic that prints the environment leaks it, and filtering by variable *name* fails when the secret hides inside `DATABASE_URL`. **Command lines:** an argument is readable by another process of the same user while it runs (here through `Get-CimInstance Win32_Process`), so never pass a secret as an argument. **Alternatives:** passing a file path, or writing the secret to the child's standard input at start-up, keeps it out of the command line and the environment, but the file needs tight permissions and anything the program prints can still leak it. I did not run `docker inspect` (the engine was not running); I am not claiming what it shows. The documented pattern is to fetch the secret at run time, cache it in memory, and refresh it (see the cost table above).

Evidence: CloudTrail and KMS

CloudTrail "records all calls to AWS KMS by users, roles, and other AWS services", including read-only calls, key management and cryptographic operations such as `GenerateDataKey` and `Decrypt`, and internal operations such as `RotateKey` and `DeleteKey`. Details: failed calls are logged "in some scenarios"; cross-account operations are logged in *both* accounts, but cross-account requests rejected for access are logged "only in the caller's account"; some fields are omitted ("the `Plaintext` parameter of an `Encrypt` request, and the response to ... any cryptographic operation"); and **the encryption context may appear in plain text**. One warning to remember: a trail can exclude all KMS events (`ExcludeManagementEventSources` set to `kms.amazonaws.com`), and "excluding AWS KMS events from a CloudTrail Log can obscure actions that use your KMS keys", so guard `cloudtrail:PutEventSelectors`. Look for a `PutEventSelectors` event with that setting if KMS events seem to be missing.

Run here (level 1): I took the three example log entries the KMS documentation prints for `Decrypt` and `GenerateDataKey`, and added 18 invented events in the same layout (marked synthetic: I had no real CloudTrail data) so there is something to find. Note the first result: the documentation's own examples are not all valid JSON.

```
py -X utf8 ct_analyze.py
```

```
1. the documentation's own example events: do they parse?
  decrypt_ok.json                    valid JSON as printed
  decrypt_fail_as_printed.json       NOT valid JSON as printed: Expecting ',' delimiter
                                     after inserting one comma: valid, 18 top-level fiel
  generatedatakey_as_printed.json    NOT valid JSON as printed: Expecting ',' delimiter
                                     after inserting one comma: valid, 20 top-level fiel

  21 events in the analysis: 3 from the documentation, 18 synthetic

2. data-quality check on the documentation's events: does the key ARN's Region match the
  Decrypt          awsRegion us-west-2  key us-west-2
  Decrypt          awsRegion us-west-2  key us-west-2
  GenerateDataKey  awsRegion us-east-1  key us-west-2        <-- differs

3. who used which key (events with a principal and a key; counts per principal)
  role App/i-0abc        key ...1234567890ab  Decrypt               6
  role App/i-0abc        key ...1234567890ab  GenerateDataKey       6
  role Batch/job-7       key ...1234567890ab  Decrypt               1
  role Batch/job-7       key ...999999999999  Decrypt               3
  role KeyAdmin/carol    key ...999999999999  CancelKeyDeletion     1
  role KeyAdmin/carol    key ...999999999999  ScheduleKeyDeletion   1
  user Alice             key ...1234567890ab  Decrypt               2
```

```
  user Alice             key ...1234567890ab  GenerateDataKey       1

4. failures (errorCode) and what the request said
  2022-11-24T18:57:43Z  Decrypt    Alice    IncorrectKeyException        context {'Depar
  2026-10-11T14:00:00Z  Decrypt    Batch    InvalidCiphertextException   context {'app':
  2026-10-11T17:10:00Z  Decrypt    Batch    KMSInvalidStateException     context {'app':
  2026-10-11T17:40:00Z  Decrypt    Batch    KMSInvalidStateException     context {'app':

5. the encryption context is plain text in the log: values that appear
  Department: ['Engineering']
  Project: ['Alpha']
  app: ['invoices', 'payroll']
  tenant: ['t-17', 't-99']

6. the documented alarm idea: any attempt to use a key that is pending deletion
  key ...999999999999 scheduled for deletion 2026-10-11T15:00:00Z, cancelled 2026-10-11T
    2026-10-11T17:10:00Z Decrypt by Batch -> KMSInvalidStateException
    2026-10-11T17:40:00Z Decrypt by Batch -> KMSInvalidStateException
```

Two of the three printed examples are missing a comma, so any tool that loads them as given fails (see *Working with JSON 1*; I inserted one comma in each to analyse them). The check that compared the event's Region with the key ARN's Region flags the `GenerateDataKey` example as inconsistent (`us-east-1` against a `us-west-2` key): it is an example, but it shows the sort of mismatch worth alerting on in real data. The questions that matter are answerable from the fields: who used which key (`userIdentity` and `resources`), which calls failed and why (`errorCode`), what context they used, and, from the last section, whether anything touched a key that was pending deletion.

Offline Checks: the SDK and parliament

Two more things you can check before any AWS call. The first is how much of the KMS request limits the SDK enforces; the second is what a policy linter says about the files in this chapter.
Run here (level 1):

```
py -X utf8 kms_botocore_lint.py
```

```
botocore 1.43.112; every call goes to Stubber, nothing is sent

case                                                what the SDK did
```

```
  Encrypt: 1 byte of plaintext                          accepted
  Encrypt: 0 bytes (documented minimum is 1)            accepted
  Encrypt: 4,096 bytes (documented maximum)             accepted
  Encrypt: 4,097 bytes (over the documented maximum)    accepted
  Encrypt: encryption context with a non-string value   REJECTED: Invalid type for param
  Encrypt: 11 grant tokens (documented maximum 10)      accepted
  Encrypt: no KeyId                                     REJECTED: Missing required param

  parliament on the policy files in pol/ (it reads each as an IDENTITY policy)
    iam_policy_account_b_role.json          ['(no findings)']
    iam_policy_context_bound.json           ['(no findings)']
    key_policy_allow_other_account.json     ['INVALID_SID', 'INVALID_SID']
    key_policy_console_style.json           ['INVALID_SID', 'INVALID_SID', 'INVALID_SID',
    key_policy_default.json                 ['INVALID_SID']
```

botocore accepted a 0-byte and a 4,097-byte `Plaintext` and 11 grant tokens: here it checked types and required fields only, not the lengths the documentation states (AWS would reject those, according to the pages; I could not send a request to see). Consistent with *AWS Intermediate/Advanced 2*: do not treat an SDK's silence as validation. And parliament, which reads everything as an identity policy, flagged the Sid of the documented default key policy (`"Enable IAM User Permissions"` contains spaces) as `INVALID_SID`. That is its identity-policy rule applied to a key policy: key policies are a different policy type, and the linter does not know. The two identity policies (the cross-account role policy and the context-bound policy) had no findings.

Cost and Clean-Up

Nothing in this chapter creates anything in AWS, so there is nothing to delete and nothing to pay. When you do build it for real, the recurring costs to know are: **$1 per month per customer managed key** (plus $1 for each of the first two rotations), a per-request price, and for Secrets Manager **$0.40 per secret per month** plus calls (prices as read 2026-10-12, no Region stated, free plans not applied). A key you no longer need costs money until it is deleted, and deletion takes 7 to 30 days and cannot be undone; disable first. Put a budget alert in place before you start (*AWS Fundamentals 9*), and use the sandbox account from *AWS Intermediate/Advanced 1*.

> **WHAT WAS AND WAS NOT CHECKED**

Run here: AES-GCM envelope encryption with a stand-in key service (sizes, failures, rotation and disabled-key behaviour); the AWS Encryption SDK with a raw key; 31 documented key-policy, IAM and grant outcomes through the evaluator model; the rotation protocol model; key and certificate sizes against the documented limits; the environment-variable and command-line experiments; CloudTrail event analysis on three documented examples plus 18 invented events; botocore offline parameter checks; parliament on five policy files. **From the documentation, not run:** every statement about KMS, S3, EBS, RDS, Secrets Manager, Parameter Store, ACM and CloudTrail behaviour, and every price. **Not read:** Parameter Store's SecureString details, Secrets Manager's alternating-users and managed rotation, ACM validity and renewal, advanced Parameter Store prices. **Not run at all:** any real KMS call, key, grant, rotation or secret; `docker inspect` (engine not running); Lambda or ECS environment settings; other SDKs or versions.

Practice

EXERCISE 1

Build envelope encryption by hand with AES-GCM and a stand-in key service that copies the documented behaviours (4,096-byte direct limit, key version stored in the blob, encryption context as authenticated data, rotation that keeps old material, a disabled key). Show the context, tamper and swapped-key failures, a rotation that leaves the bulk ciphertext unchanged, and a running process that keeps working on a cached data key. Then repeat the pattern with the AWS Encryption SDK and a raw key, and show what is and is not visible in the encrypted message.

View solution

EXERCISE 2

Write the KMS key policies from the documentation (default, console-style, cross-account) and run the documented access rules as tests through an evaluator: delegation by the root statement, a key policy that names only one user, administrators versus users, the administrator-grants-themselves path, cross-account use needing both sides, grants that allow but never deny, a key pending deletion, and an encryption-context condition. Add the offline SDK and linter checks and report what each does and does not catch.

View solution

EXERCISE 3

Model the Secrets Manager rotation protocol (labels, four steps, the confused-deputy checks, the in-progress rule, the version quota), measure real key and certificate sizes against the Parameter Store and Secrets Manager limits and compute reading costs, run the environment-variable and command-line leak experiments, and analyse KMS CloudTrail events, including the documentation's own examples that fail to parse.

View solution

CHAPTER 3 QUICK REFERENCE

- Defaults: S3 encrypts all new objects (SSE-S3) since 2023-01-05 and cannot be turned off; EBS needs the per-Region "encryption by default" setting and does not change existing volumes; RDS encrypts only at creation and the key cannot be changed afterwards
- KMS keys: customer managed ($1/month, full control, optional rotation), AWS managed (free to exist, yearly rotation, not shareable), AWS owned (invisible, free). Direct `Encrypt` is limited to 4,096 bytes: use envelope encryption (`GenerateDataKey`)
- The encryption context is authenticated but **not secret**: it must match exactly on decrypt, and it appears in clear text in messages and CloudTrail. Never put secrets in it
- A key policy must name its principals. The root statement only **delegates** to IAM policies; without it, IAM allows do nothing (IAM denies still work). Key policies are Regional
- Key administrators manage but cannot use a key, yet can edit the key policy and create grants, so they can give themselves use. Cross-account needs the key policy naming the other account **and** an IAM policy in that account
- Grants allow a grantee specific operations on one key, never deny, and vanish when retired or revoked; they are eventually consistent (use a grant token to act at once)
- Rotation changes key material only: old material is kept, data and data keys are not re-encrypted. Deletion is irreversible, with a 7 to 30 day waiting period (default 30): disable first and alarm on use during the wait
- Secrets Manager: rotation built in, 65,536-byte values, $0.40 per secret per month plus calls. Parameter Store: standard 4 KB and free, advanced 8 KB with policies and sharing; no built-in rotation. Certificates belong in ACM, encryption keys in KMS, AWS credentials in IAM
- Rotation = createSecret, setSecret, testSecret, finishSecret with labels AWSPENDING, AWSCURRENT, AWSPREVIOUS; the function must check the same user and host before acting; a stale `AWSPENDING` blocks the next rotation. Cache secrets: reading on every request is the expensive mistake
- Environment variables are inherited by every child, leak through diagnostic dumps (even when you redact by name), and command-line arguments are readable by other processes: fetch at run time, keep in memory

- CloudTrail logs KMS calls (both accounts for cross-account, denied cross-account only for the caller); do not exclude KMS events from a trail without a reason, and do not trust an SDK's silence as validation

COMING NEXT

Chapter 4 is infrastructure as code with CloudFormation: templates, parameters, conditions, change sets, rollback and drift, with every template linted for real. Everything in this chapter (keys, key policies, secrets, rotation) is something you will want to define in a template rather than click. Start with *AWS Intermediate/Advanced 4*.

AWS Intermediate/Advanced

Chapter 4 · Infrastructure as Code with CloudFormation

The first three chapters described things you would create by hand: accounts and guardrails, roles and policies, keys and secrets. Doing that by clicking does not scale, cannot be reviewed, and cannot be repeated. AWS CloudFormation is AWS's own answer: you describe the resources in a **template**, CloudFormation creates them together as a **stack**, and later changes go through the same template. This chapter builds one real template (the encrypted bucket, queues and role from the last two chapters), then goes through parameters, mappings, conditions, change sets, failure and rollback, deletion policies, drift and quotas. Every template here is checked with `cfn-lint`, and the parts of CloudFormation that need an account are modelled or labelled.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS CLI on the machine that wrote this course, so no stack was created, changed or deleted.

- **Run here (level 1):** Python 3.14.0 on Windows 11, in a scratch folder outside this site, with `cfn-lint` 1.57.2 (a template checker that works offline), `cfn-flip` 1.3.0 (YAML and JSON conversion) and PyYAML. cfn-lint output is pasted from real runs.
- **Needs an AWS account (level 2):** `aws cloudformation deploy`, change sets, rollback, drift detection and every real stack operation. Shown with their documented options, **not run here**.
- **From the documentation, not run (level 3):** every statement about how CloudFormation behaves. I read the template, parameter, condition, change set, failure, deletion-policy, drift and quota pages and the CLI reference on 2026-10-12 and 2026-10-13 through a page-fetch tool that summarises pages. **One page (Outputs) came back only as a paraphrase from the tool's general knowledge, and is marked where used.**

Three tools in this chapter are **models, not AWS**: a template resolver (`cfn_resolve.py`), an approximation of a change set, and a failed-deployment simulator. Each states which documented rules it copies and which it does not.

A Template, Section by Section

The documentation's rule is short: the **Resources** section "is required in every CloudFormation template"; everything else is optional. The sections: `Description`, `Metadata`, `Parameters` (values supplied when you create or update the stack), `Rules` (checks on parameter

combinations), `Mappings` ("lookup tables"), `Conditions`, `Transform` (macros, including the Serverless Application Model), `Resources` and `Outputs` (values returned for viewing or for other stacks). Each resource has a logical ID unique in the template, a type and its properties.

The template below is the running example. It builds, for one environment, the pieces *AWS Intermediate/Advanced 3* argued for: a KMS key with the root statement that *delegates* to IAM policies and a separate administrator statement, an S3 bucket encrypted with that key (and refusing plain-HTTP requests), a work queue with a dead-letter queue, a worker role that gets exactly what it needs, and a backlog alarm that exists only in production. First the inputs:

```
Parameters:
  Environment:
    Type: String
    Default: dev
    AllowedValues: [dev, staging, prod]
    Description: Which environment this stack is for
  RetentionDays:
    Type: Number
    Default: 30
    MinValue: 1
    MaxValue: 3650
    Description: Days before report objects expire
  KeyAdminRoleArn:
    Type: String
    AllowedPattern: "^arn:aws:iam::[0-9]{12}:role/[A-Za-z0-9+=,.@/_-]+$"
    ConstraintDescription: Must be the ARN of an IAM role
    Description: Role that may administer (not use) the key

Mappings:
  EnvSettings:
    dev:     { VisibilityTimeout: 30, MaxReceive: 3 }
    staging: { VisibilityTimeout: 60, MaxReceive: 5 }
    prod:    { VisibilityTimeout: 120, MaxReceive: 5 }

Conditions:
  IsProd: !Equals [!Ref Environment, prod]
```

Then the key and its alias. The key policy has no statement naming the worker role, on purpose: the role gets `kms:Decrypt` from its own identity policy because the root statement delegates. Naming the role in the key policy would make the key depend on the role and the role (whose policy refers to the key) depend on the key, which is a circular dependency that cfn-lint rejects

(see the experiment below). `DeletionPolicy` and `UpdateReplacePolicy` are both `Retain` for the data stores.

```
  DataKey:
    Type: AWS::KMS::Key
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      Description: !Sub "Data key for ${AWS::StackName} (${Environment})"
      EnableKeyRotation: true
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          - Sid: Enable IAM User Permissions
            Effect: Allow
            Principal:
              AWS: !Sub "arn:${AWS::Partition}:iam::${AWS::AccountId}:root"
            Action: "kms:*"
            Resource: "*"
          - Sid: Allow access for Key Administrators
            Effect: Allow
            Principal:
              AWS: !Ref KeyAdminRoleArn
            Action:
              - "kms:Create*"
              - "kms:Describe*"
              - "kms:Enable*"
              - "kms:List*"
              - "kms:Put*"
              - "kms:Update*"
              - "kms:Revoke*"
              - "kms:Disable*"
              - "kms:Get*"
              - "kms:Delete*"
              - "kms:TagResource"
              - "kms:UntagResource"
              - "kms:ScheduleKeyDeletion"
              - "kms:CancelKeyDeletion"
            Resource: "*"

  DataKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
```

```
      AliasName: !Sub "alias/${AWS::StackName}-data"
      TargetKeyId: !Ref DataKey
```

The bucket, its policy and the two queues:

```
  ReportBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - BucketKeyEnabled: true
            ServerSideEncryptionByDefault:
              SSEAlgorithm: "aws:kms"
              KMSMasterKeyID: !GetAtt DataKey.Arn
      VersioningConfiguration:
        Status: Enabled
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
      LifecycleConfiguration:
        Rules:
          - Id: expire-reports
            Status: Enabled
            ExpirationInDays: !Ref RetentionDays
      Tags:
        - Key: environment
          Value: !Ref Environment

  ReportBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref ReportBucket
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Sid: DenyInsecureTransport
            Effect: Deny
            Principal: "*"
            Action: "s3:*"
            Resource:
```

```
              - !GetAtt ReportBucket.Arn
              - !Sub "${ReportBucket.Arn}/*"
            Condition:
              Bool:
                "aws:SecureTransport": "false"

  WorkDeadLetterQueue:
    Type: AWS::SQS::Queue
    Properties:
      MessageRetentionPeriod: 1209600
      KmsMasterKeyId: !Ref DataKey

  WorkQueue:
    Type: AWS::SQS::Queue
    Properties:
      VisibilityTimeout: !FindInMap [EnvSettings, !Ref Environment, VisibilityTimeout]
      KmsMasterKeyId: !Ref DataKey
      RedrivePolicy:
        deadLetterTargetArn: !GetAtt WorkDeadLetterQueue.Arn
        maxReceiveCount: !FindInMap [EnvSettings, !Ref Environment, MaxReceive]
```

Run here (level 1). The whole template, linted:

```
cfn-lint app_stack.yaml
```

```
cfn-lint exit status: 0
```

Clean, with exit status 0. Everything below that is "cfn-lint says" means that exit status and those rule IDs; I observed 0 for clean, 2 for errors, 4 for warnings and 6 for both.

Parameters

Parameters are inputs you supply when you create or update a stack. From the documentation: the only required attribute is `Type`; `Number` "is validated as a number; however, when you use the parameter elsewhere ... the parameter value becomes a string"; `CommaDelimitedList` gives "an array of literal strings that are separated by commas" with each member "space trimmed"; `List<Number>` becomes a list of strings. The constraints are `AllowedValues`, `AllowedPattern` (for a `String` "the pattern must match the entire parameter value"), `MinLength`, `MaxLength`, `MinValue`, `MaxValue`, `ConstraintDescription` and `Default`. A parameter with no default and

no value fails the stack with `Parameters: [KeyName] must have values`. A template may have at most **200** parameters. There are also AWS-specific types (key pair names, VPC IDs) and Systems Manager parameter types that look up the latest value in Parameter Store.

> **NOECHO HIDES A VALUE FROM DISPLAY, NOT FROM THE TEMPLATE**
>
> `NoEcho: true` masks the value "as asterisks" in describe calls and stack events, but "does not mask any information stored in" the `Metadata` section, the `Outputs` section, or a resource's `Metadata` attribute. The page says "We strongly recommend you do not use these mechanisms to include sensitive information" and to use dynamic references to Systems Manager Parameter Store or Secrets Manager instead (the page for those, which lists a limit of 60 dynamic references per template, I did not read). It also warns against putting a `NoEcho` parameter into a property that forms a resource's primary identifier, because the real value "may" end up in the resource ID. For a password, the secret from *AWS Intermediate/Advanced 3* is the answer, not a parameter.

Run here (level 1). To check my understanding of these rules I wrote a small resolver: it takes a template and parameter values, applies the documented rules, and shows what the template becomes. Values only AWS knows (a resource's ID or attribute) stay as placeholders. It is a *model*, tested below. Here it renders the template for two environments and rejects four bad parameter sets:

```
py -X utf8 render_app.py
```

```
== Environment=dev: 7 resources: DataKey, DataKeyAlias, ReportBucket, ReportBucketPolicy
   WorkQueue    : {"VisibilityTimeout": 30, "KmsMasterKeyId": "<Ref DataKey>", "RedriveP
   DataKey desc.: Data key for reports (dev)
   DataKeyAlias : {'AliasName': 'alias/reports-data', 'TargetKeyId': '<Ref DataKey>'}
   Outputs      : {"BucketName": "<Ref ReportBucket>", "QueueUrl": "<Ref WorkQueue>", "K
== Environment=prod: 8 resources: BacklogAlarm, DataKey, DataKeyAlias, ReportBucket, Rep
   WorkQueue    : {"VisibilityTimeout": 120, "KmsMasterKeyId": "<Ref DataKey>", "Redrive
   DataKey desc.: Data key for reports (prod)
   DataKeyAlias : {'AliasName': 'alias/reports-data', 'TargetKeyId': '<Ref DataKey>'}
   Outputs      : {"BucketName": "<Ref ReportBucket>", "QueueUrl": "<Ref WorkQueue>", "K
   RetentionDays 0 (MinValue is 1): CfnError: Parameter RetentionDays: 0 is outside MinV
   Environment qa: CfnError: Parameter Environment: 'qa' is not one of AllowedValues ['d
   KeyAdminRoleArn that is not a role ARN: CfnError: Malformed input-Parameter KeyAdminR
   no KeyAdminRoleArn (no Default): CfnError: Parameters: [KeyAdminRoleArn] must have va
```

Production gets a longer visibility timeout, a different redrive count and an alarm and output that development does not have, all from one template. Note how each bad input fails: a value below `MinValue`, a value outside `AllowedValues`, a pattern failure (the message uses the page's wording, with the template's `ConstraintDescription`), and a missing value with the page's exact message.

Mappings, Conditions and Intrinsic Functions

Mappings are two-level lookup tables used with `Fn::FindInMap`; the template above uses one to choose queue settings by environment. **Conditions** are named true/false values built with `Fn::Equals`, `Fn::And`, `Fn::Or`, `Fn::Not`, and you attach them to resources and outputs (`Condition: IsProd`) or to property values with `Fn::If`. Rules from the Condition functions page:

- `Fn::And` and `Fn::Or` take "the minimum number of conditions ... is 2, and the maximum is 10".
- Inside `Fn::Equals` and the other condition functions you may use only `Ref`, `Fn::FindInMap` and other condition functions. (`Fn::If` itself accepts a longer list, including `Fn::Sub` and `Fn::GetAtt`.)
- `Fn::If` can return `AWS::NoValue` to remove a property entirely.
- "You can use intrinsic functions only in specific parts of a template": resource properties, outputs, metadata attributes and update policy attributes, and to conditionally create resources. That list does not include `DeletionPolicy`.

The full list of functions on the reference page: `Fn::Base64`, `Fn::Cidr`, the condition functions, `Fn::FindInMap`, `Fn::ForEach`, `Fn::GetAtt`, `Fn::GetAZs`, `Fn::GetStackOutput`, `Fn::ImportValue`, `Fn::Join`, `Fn::Length`, `Fn::Select`, `Fn::Split`, `Fn::Sub`, `Fn::ToJsonString`, `Fn::Transform`, `Ref` and rule functions. The resolver models the first group; it refuses `Fn::GetAZs`, `Fn::Cidr`, `Fn::Transform`, `Fn::ForEach`, `Fn::Length` and `Fn::ToJsonString` rather than guess.

Run here (level 1). I ran the documentation's own examples through the resolver (the parameter type examples, the `EnvType` condition sample for `prod`, `dev` and `test`, the subnet `Select`/`Sub` example, the `NoValue` snippet, the condition-function limits) and wrote each expected answer first:

```
py -X utf8 t_resolve.py
```

```
result    case
match     Number 8888, then Ref: 'becomes a string'
match     List<Number> '80,20': a Ref 'would result in ["80","20"]'
match     CommaDelimitedList 'test,dev,prod'
match     CommaDelimitedList members are space-trimmed
match     Number parameter given 'abc'
match     AllowedPattern [A-Za-z0-9]+ and the value abc123
match     the page's message for a failed AllowedPattern
match     the pattern must match the ENTIRE value: 'xxabc123yy' against 'abc123'
match     a parameter with no Default and no value
match     DbSubnet1 AvailabilityZone: Select 0 of the AZ list
match     DbSubnet1 CidrBlock: Select 0 of the CIDR list
match     DbSubnet3 CidrBlock: Select 2
FINDING   FINDING (not counted): DbSubnet2 AvailabilityZone, ${AWS::Region}${AZ} with th
match     EnvType prod: InstanceType (nested Fn::If)
match     EnvType prod: MountPoint and NewVolume are created only for prod
match     EnvType dev: InstanceType (nested Fn::If)
match     EnvType dev: MountPoint and NewVolume are created only for prod
match     EnvType test: InstanceType (nested Fn::If)
match     EnvType test: MountPoint and NewVolume are created only for prod
match     EnvType 'qa' is not among AllowedValues
match     NoValue: no snapshot name -> the property is removed
match     NoValue: snapshot name 'snap-1' -> the property is present
match     Fn::Not of Fn::Equals (the page's EnvCondition shape)
match     Fn::And with ONE condition (the minimum is 2)
match     Fn::Or with 11 conditions (the maximum is 10)
match     Fn::Sub inside Fn::Equals (only Ref, Fn::FindInMap and condition functions are
match     a condition that names another condition
match     YAML: AllowedValues [Yes, No] without quotes is read by PyYAML as booleans
match     YAML: with single quotes the strings survive
match     cfn-flip's loader on AllowedValues [Yes, No] (what a CloudFormation tool sees)
match     Fn::Join ['-', [a, b, c]]
match     Fn::Split [',', 'x,y,z'] then Select 1
match     Sub with an escaped literal ${!Literal}
match     Fn::GetAZs is not modelled (it needs the account's Regions)

33 counted cases, mismatches: 0; 1 finding(s) shown but not counted
```

All 33 counted cases match, and the control (one expectation flipped) reports one mismatch.

One result is a finding, deliberately not counted: the Parameters page's subnet example

builds an availability zone with `${AWS::Region}${AZ}` where `AZ` is one of the page's own default values, `us-west-2b`. With the Region `us-west-2` that resolves to `us-west-2us-west-2b`. The example only makes sense if the list held bare letters like `b`. cfn-lint did not notice (it reported only a warning about the default VPC value). I also checked the YAML trap the page warns about: an unquoted `[Yes, No]` is read as booleans by PyYAML and by cfn-flip's loader, so `AllowedValues` with `Yes` and `No` needs single quotes. Three small cases (`Fn::Join`, `Fn::Split` with `Fn::Select`, and the `${!Literal}` escape in `Fn::Sub`) come from my own knowledge of those functions, not from pages I read.

Linting: What cfn-lint Catches and What It Does Not

`cfn-lint` checks a template against the resource schemas that CloudFormation publishes, plus rules of its own, without any AWS call. To learn what it really catches I broke `app_stack.yaml` in 22 specific ways (each a plain text replacement) and generated six templates that break documented quotas. A mutation that is not flagged is a gap in the tool, not in the experiment:

```
py -X utf8 mutate_lint.py
```

```
python.exe C:\lwaws\c4\.venv\Scripts\cfn-lint 1.57.2

mutation of app_stack.yaml                                              exit  ru
(the unchanged template)                                                exit 0
unknown property name (EnableKeyRotations)                              exit 2
wrong type (a word where a number belongs)                              exit 2
Ref to a resource that does not exist                                   exit 2
GetAtt of an attribute that does not exist                              exit 2
Default not among AllowedValues                                         exit 2
required property removed (Alias TargetKeyId)                           exit 2
circular dependency (key policy names the role that uses the key)       exit 2
Sub with a variable nothing defines                                     exit 2
Fn::Sub inside Fn::Equals (the docs allow only Ref and FindInMap there) exit 0
Condition name that is not defined                                      exit 2
invalid DeletionPolicy value (Keep)                                     exit 2
DeletionPolicy written with !If                                         exit 0
unused parameter                                                        exit 4
unused condition                                                        exit 4
output of a conditional resource without the condition                  exit 4
bucket name that breaks the naming rules                                exit 2
IAM action without the service prefix                                   exit 4
IAM Effect misspelt (Allowed)                                           exit 2
```

```
  FindInMap with a second-level key that does not exist                          exit 0
  Number parameter with a non-number default                                     exit 0
  KMS key policy with Principal missing                                          exit 6
  SQS retention above the documented service maximum (1209600 -> 99999999)       exit 2

  mutations cfn-lint did NOT flag at all (exit status 0): 4 of 22
     - Fn::Sub inside Fn::Equals (the docs allow only Ref and FindInMap there)
     - DeletionPolicy written with !If
     - FindInMap with a second-level key that does not exist
     - Number parameter with a non-number default

  the documented quotas
  501 resources (documented maximum is 500)                                      exit 2
  500 resources (at the maximum)                                                 exit 0
  201 parameters (documented maximum is 200)                                     exit 4
  201 outputs (documented maximum is 200)                                        exit 2
  description of 1,100 bytes (documented maximum is 1,024)                       exit 2
  template body of about 60,000 bytes (documented direct-request maximum is 51,200) exit 0
```

Read the result in three parts. **Caught well:** a misspelt property, a wrong type, a reference to something that does not exist, a missing required property, a bad `GetAtt`, a Default outside `AllowedValues`, an invalid `DeletionPolicy`, an undefined condition, a malformed S3 bucket name, an IAM `Effect` typo, a value above a service maximum, and the circular dependency (E3004 names the loop). **Caught only as a warning** (exit status 4): an unused parameter or condition, an output that refers to a conditional resource without the condition, an IAM action missing its service prefix. A pipeline that fails only on errors will let these through. **Not caught at all (4 of 22):** `Fn::Sub` inside `Fn::Equals` (the documentation says it is not allowed), `DeletionPolicy` written with `!If`, a `FindInMap` key that exists in no mapping, and a `Number` parameter whose Default is the word `thirty`. For the quotas, cfn-lint flagged 501 resources, 201 outputs and a 1,100-byte description, accepted exactly 500 resources, and did **not** flag 201 parameters (only an unused-parameter warning) or a 62 KB template body. Each of those is a reason to keep a second check, such as the resolver below, and a reason to test on one account before the first real deploy.

CHECK THE CHECKER

My first limits run padded the template to "501 resources" by assuming it started with ten, but it had eight, so it actually tested 499 and 498 and cfn-lint seemed to accept an over-limit template. I only noticed because a second tool (the quota checker further down) counted and disagreed. The

generator now counts the template's real resources first, and cfn-lint does flag 501 (rule E3010). When two tools disagree, one of them is not measuring what you think.

Next, the documentation's own example templates, which you would expect to be clean:

```
cfn-lint (the documentation's examples and Chapter 1's SCP template)
```

```
== doc/conditions_sample.yaml
exit status: 0
== doc/subnets_example.yaml
W1030 {'Ref': 'VPC'} is not a 'AWS::EC2::VPC.Id' with pattern '^vpc-(([0-9A-Fa-f]{8})|([0
doc\subnets_example.yaml:18:7

W1030 {'Ref': 'VPC'} is not a 'AWS::EC2::VPC.Id' with pattern '^vpc-(([0-9A-Fa-f]{8})|([0
doc\subnets_example.yaml:30:7

W1030 {'Ref': 'VPC'} is not a 'AWS::EC2::VPC.Id' with pattern '^vpc-(([0-9A-Fa-f]{8})|([0
doc\subnets_example.yaml:42:7

exit status: 4
== doc/deletionpolicy.yaml
W3011 Both 'UpdateReplacePolicy' and 'DeletionPolicy' are needed to protect resource fro
doc\deletionpolicy.yaml:3:3

exit status: 4
== doc/ch1_scp_stack.yaml
exit status: 0
```

The `EnvType` sample and Chapter 1's SCP template are clean. The subnet example gets warnings because its default `VPC` value `vpc-123456` is not a valid VPC ID pattern, and the page's `DeletionPolicy: Retain` example gets `W3011`: "Both 'UpdateReplacePolicy' and 'DeletionPolicy' are needed to protect resource from deletion". That second warning is why the main template sets both.

Quotas to Design Around

From the CloudFormation quotas page (read 2026-10-12):

QUOTA	VALUE	THE PAGE'S SUGGESTION WHEN YOU HIT IT
Resources per template	500	Split into nested stacks
Parameters per template	200	Use mappings or lists to carry several values in one parameter
Outputs per template	200	
Mappings per template, attributes per mapping	200 and 200	Split into more mappings or nested stacks
Template description	1,024 bytes	
Template body in a request	51,200 bytes	Upload it to S3 (limit 1 MB), or split into nested stacks
Parameter value	4,096 bytes	Several parameters joined with `Fn::Join`
Stacks per account (per Region)	2,000	Delete unused stacks or request an increase
Stack name	128 characters	
Resources changed in one nested-stack operation	2,500	Split the hierarchy
StackSets: sets, instances per set	1,000 and 100,000	Delete or request an increase
Dynamic references per template	60	

The `deploy` command in the CLI needs `--s3-bucket` for templates "greater than 51,200 bytes". My own quota checker (below) applied those numbers to the template and to the generated ones and agrees with the experiment above, including the two cfn-lint misses (parameters and body size).

Stacks, Change Sets and Deploying

A **stack** is the set of resources created from one template. Updating means giving CloudFormation a changed template or new parameter values. The safe route is a **change set**: "Change sets allow you to preview how proposed changes to a stack might impact your running resources". The flow on the page: create the change set (nothing changes yet), view it (which resources will be *added*, *modified* or *removed*, with a before/after for properties), optionally

create more, and execute the one you want. After an execution "CloudFormation removes all change sets that are associated with the stack". One caution stands out: "Change sets don't guarantee that CloudFormation will successfully update a stack". Pre-deployment validation catches common causes (syntax errors, name conflicts, quota limits) but "failures that occur because of runtime conditions ... might still occur during execution".

A modification "can cause the resource to be interrupted or replaced (recreated)", which is the field you read most carefully. The page's example output shows `"Action": "Modify"` and `"Replacement": "False"` with property-level details. `aws cloudformation deploy` wraps all of this: it "deploys the specified AWS CloudFormation template by creating and then executing a change set", and with `--no-execute-changeset` it stops after creating it so you can review it. Its documented options include `--parameter-overrides Key=Value`, `--capabilities` (`CAPABILITY_IAM` or `CAPABILITY_NAMED_IAM`; without it, an IAM template fails with `InsufficientCapabilities`), `--fail-on-empty-changeset` (by default an empty change set exits with status 0) and `--s3-bucket`. **Level 2, not run here:**

```
# level 2: needs an AWS account; NOT RUN here. Options as documented on the CLI referenc
# 1. review the change set without applying it
aws cloudformation deploy \
  --template-file app_stack.yaml \
  --stack-name reports-dev \
  --parameter-overrides Environment=dev KeyAdminRoleArn=arn:aws:iam::111122223333:role/K
  --capabilities CAPABILITY_IAM \
  --no-execute-changeset

# 2. apply it (creates and executes a change set)
aws cloudformation deploy \
  --template-file app_stack.yaml \
  --stack-name reports-dev \
  --parameter-overrides Environment=dev KeyAdminRoleArn=arn:aws:iam::111122223333:role/K
  --capabilities CAPABILITY_IAM

# 3. a failed create that keeps what worked (documented option)
aws cloudformation create-stack --stack-name reports-dev --template-body file://app_stac

# 4. drift (documented three-step flow)
aws cloudformation detect-stack-drift --stack-name reports-dev
aws cloudformation describe-stack-drift-detection-status --stack-drift-detection-id <id
aws cloudformation describe-stack-resource-drifts --stack-name reports-dev --stack-resou
```

Which capability does a template need?

The page says: "If you have IAM resources, you can specify either capability. If you have IAM resources with custom names, you must specify CAPABILITY_NAMED_IAM." I wrote that rule as a checker (my list of name properties, since the page says only "custom names": `RoleName`, `UserName`, `GroupName`, `InstanceProfileName`, `ManagedPolicyName`). Its results are in section 3 of the output further down: `app_stack.yaml` needs `CAPABILITY_IAM`, adding `RoleName` makes it `CAPABILITY_NAMED_IAM`, and Chapter 1's SCP template needs neither.

An approximation of a change set, computed locally

You cannot create a change set without an account, but you can approximate the most important question: *will this edit replace something?* The resource schemas bundled in cfn-lint list each type's **create-only properties**, the ones that cannot be changed on an existing resource. My approximation renders the old and new template, lists Add, Remove and Modify with property-level differences, and calls a modification `Replacement: True` when a changed property is create-only, `Conditional` when it is conditionally create-only, and `False` otherwise. **It is not a change set**: a `False` can still be an interruption (the schema does not record those), and I did not read the page's definitions of `True` and `Conditional`, so those two meanings are my own.

```
py -X utf8 changeset_model.py
```

```
schemas from cfn-lint 1.57.2


-- 1. parameter change: Environment dev -> prod
   Add    BacklogAlarm          AWS::CloudWatch::Alarm    Replacement: -
   Modify DataKey               AWS::KMS::Key             Replacement: False
            Modify /Properties/Description                                  "Data k
   Modify ReportBucket          AWS::S3::Bucket           Replacement: False
            Modify /Properties/Tags/0/Value                                 "dev" -
   Modify WorkQueue             AWS::SQS::Queue           Replacement: False
            Modify /Properties/RedrivePolicy/maxReceiveCount                3 -> 5
            Modify /Properties/VisibilityTimeout                            30 -> 1

-- 2. parameter change: RetentionDays 30 -> 90
   Modify ReportBucket          AWS::S3::Bucket           Replacement: False
            Modify /Properties/LifecycleConfiguration/Rules/0/ExpirationInDays   "30" ->

-- 3. template edit: give the bucket a fixed BucketName
```

```
    Modify ReportBucket          AWS::S3::Bucket          Replacement: True
             Add    /Properties/BucketName                                  null ->

  -- 4. template edit: give the work queue a fixed QueueName
    Modify WorkQueue             AWS::SQS::Queue          Replacement: True
             Add    /Properties/QueueName                                   null ->

  -- 5. template edit: rename only the bucket's logical ID, ReportBucket to Reports (the b
    Remove ReportBucket          AWS::S3::Bucket          Replacement: -
    Modify ReportBucketPolicy    AWS::S3::BucketPolicy    Replacement: True
             Modify /Properties/Bucket                                       <unknow
             Modify /Properties/PolicyDocument/Statement/0/Resource/0        <unknow
             Modify /Properties/PolicyDocument/Statement/0/Resource/1        <unknow
    Add    Reports               AWS::S3::Bucket          Replacement: -
    Modify WorkerRole            AWS::IAM::Role           Replacement: False
             Modify /Properties/Policies/0/PolicyDocument/Statement/1/Resource    <unknow

  -- 6. parameter change: a different key administrator role
    Modify DataKey               AWS::KMS::Key            Replacement: False
             Modify /Properties/KeyPolicy/Statement/1/Principal/AWS          "arn:aw

  create-only properties in the bundled schemas (a change to one of these means Replacemen
    AWS::S3::Bucket           ['BucketName', 'BucketNamePrefix', 'BucketNamespace']
    AWS::SQS::Queue           ['FifoQueue', 'QueueName']
    AWS::KMS::Key             (none listed)
    AWS::KMS::Alias           ['AliasName']
    AWS::IAM::Role            ['Path', 'RoleName']
    AWS::CloudWatch::Alarm    ['AlarmName']
    AWS::S3::BucketPolicy     ['Bucket']
```

Four lessons. **(1)** Changing a parameter from `dev` to `prod` adds the alarm and modifies three resources, with no replacement. **(2)** Adding a fixed `BucketName` or `QueueName` to an existing resource is a *replacement*: the properties are create-only, and naming resources is the classic way to make a harmless-looking edit destructive. **(3)** Renaming a logical ID is not a rename: it is a remove plus an add, and everything that referred to it changes too (here the bucket policy's `Bucket` is create-only, so it is replaced, and the role's policy changes). That chain is derived from the documented Add/Remove behaviour, not a quoted sentence. **(4)** The schema lists no create-only properties for `AWS::KMS::Key`; I report that and make no claim about what real CloudFormation does when a key's other properties change.

When a Deployment Fails

The "Stack failure options" page describes how a failed operation behaves. CloudFormation "identifies dependencies between resources to parallelize independent provisioning actions", and "stops at the first failure in each independent provisioning path. A failure in one path doesn't affect other provisioning paths". If you choose **Preserve successfully provisioned resources** (`--disable-rollback`, or `on-failure DO_NOTHING`), the successful resources stay, the failed one is left in a failed state (`CREATE_FAILED`), and you can *Retry* (for a cause such as a missing IAM permission), *Update* (the template was wrong) or *Roll back*. Without that option the default is the rollback you are used to. The CLI command `rollback-stack` returns a failed stack to "its last stable state" and "will delete a stack if it doesn't contain a last known stable state". With express mode, rollback is off unless you turn it back on.

Rollback triggers let CloudWatch alarms stop a deployment: you name up to **5** alarms (or composite alarms) and a monitoring time of 0 to 180 minutes after the deployment; if any goes to `ALARM` during the operation or that period, "CloudFormation rolls back the entire stack operation". They are not monitored while a rollback is happening, and a missing alarm makes the whole operation fail and roll back.

Run here (level 1): I extracted the dependency graph from the template (references, `GetAtt`, `Sub` variables and `DependsOn`), and applied the documented behaviour to it: which resources get created, which fail, and which are never attempted when one resource fails. This is a model of those sentences, not CloudFormation. Section 3 shows the capability checker and section 4 the quota checker.

```
py -X utf8 stack_graph.py
```

```
1. dependency graph of app_stack.yaml (Environment=prod: BacklogAlarm exists)
   BacklogAlarm           depends on ['WorkQueue']
   DataKey                depends on -
   DataKeyAlias           depends on ['DataKey']
   ReportBucket           depends on ['DataKey']
   ReportBucketPolicy     depends on ['ReportBucket']
   WorkDeadLetterQueue    depends on ['DataKey']
   WorkQueue              depends on ['DataKey', 'WorkDeadLetterQueue']
   WorkerRole             depends on ['DataKey', 'ReportBucket', 'WorkQueue']
   waves (each wave can be provisioned in parallel once the earlier waves are done):
     wave 1: DataKey
     wave 2: DataKeyAlias, ReportBucket, WorkDeadLetterQueue
```

```
      wave 3: ReportBucketPolicy, WorkQueue
      wave 4: BacklogAlarm, WorkerRole
    cycle in the real template: None
    cycle after letting the key policy name the role: ['DataKey', 'WorkerRole', 'DataKey'

  2. a failed deployment, preserve-successful-resources model (which resources get created
    DataKey              fails -> created: []
                                  not attempted: ['BacklogAlarm', 'DataKeyAlias', 'Report
    WorkDeadLetterQueue  fails -> created: ['DataKey', 'DataKeyAlias', 'ReportBucket', 'R
                                  not attempted: ['BacklogAlarm', 'WorkQueue', 'WorkerRol
    ReportBucket         fails -> created: ['BacklogAlarm', 'DataKey', 'DataKeyAlias', 'W
                                  not attempted: ['ReportBucketPolicy', 'WorkerRole']
    WorkerRole           fails -> created: ['BacklogAlarm', 'DataKey', 'DataKeyAlias', 'R
                                  not attempted: []
    BacklogAlarm         fails -> created: ['DataKey', 'DataKeyAlias', 'ReportBucket', 'R
                                  not attempted: []

  3. which capability a deployment needs
    app_stack.yaml: CAPABILITY_IAM  (IAM resources: ['WorkerRole'])
    same template with RoleName: work-role: CAPABILITY_NAMED_IAM
    Chapter 1's SCP template (AWS::Organizations::Policy): (none)

  4. the documented quotas, applied to the template and to the cfn-lint experiment's gener
    app_stack.yaml                                                                  with
    501 resources (documented maximum is 500)                                       reso
    500 resources (at the maximum)                                                  with
    201 parameters (documented maximum is 200)                                      para
    201 outputs (documented maximum is 200)                                         outp
    description of 1,100 bytes (documented maximum is 1,024)                        desc
    template body of about 60,000 bytes (documented direct-request maximum is 51,2   body
```

Read the waves: the key must exist first, then the alias, bucket and dead-letter queue can go in parallel, and so on. When the key fails, nothing else is attempted. When the dead-letter queue fails, the key, alias, bucket and bucket policy are still created (an independent path) but the work queue, role and alarm are never attempted. When the worker role fails, everything else exists. The circular-dependency check finds the same loop as cfn-lint's E3004, which is the cross-check I trust more than either alone. The quota section agrees with the cfn-lint experiment.

Deletion and Replacement Policies

`DeletionPolicy` controls what happens to a resource when its stack is deleted (and when you remove the resource from the template). The documentation:

- `Delete` (the default): the resource is deleted. Exceptions: the default is `Snapshot` for `AWS::RDS::DBCluster` and for `AWS::RDS::DBInstance` without a `DBClusterIdentifier`. For an S3 bucket "you must delete all objects in the bucket for deletion to succeed".
- `Retain`: keep the resource, "continue to incur applicable charges until you delete those resources". Allowed on any resource.
- `RetainExceptOnCreate`: like `Retain`, except that a rollback of the stack operation that *created* the resource still deletes it (new empty resources go, in-use ones stay).
- `Snapshot`: for resources that support it (EBS volumes, RDS instances and clusters, ElastiCache, Redshift, DocumentDB and Neptune clusters are on the page's list), take a snapshot first; the snapshot continues to incur charges.

Two traps. The page says the policy "doesn't apply to resources whose physical instance is replaced during stack update operations", so a harmless-looking edit that forces a replacement (previous section) is not protected by `DeletionPolicy` alone; `UpdateReplacePolicy` is the other half, and cfn-lint's `W3011` says to set both (I did not read the `UpdateReplacePolicy` page itself). And `DeletionPolicy` takes a plain string: the experiment above shows cfn-lint accepting `!If` there, which the documentation's list of places for intrinsic functions does not allow. The page also advises: if you want to change retained resources outside CloudFormation, use `Retain` then delete the stack, "otherwise, your resources might get out of sync with your CloudFormation template".

Drift

Drift detection compares what a template says with what exists. The page: "Performing a drift detection operation on a stack determines whether the stack has drifted from its expected template configuration, and returns detailed information about the drift status of each resource in the stack that supports drift detection". A stack "is considered to have drifted if one or more of its resources have drifted". You run `detect-stack-drift`, watch it with `describe-stack-drift-detection-status` (it "may take several minutes"; one run at a time per stack), and read the details with `describe-stack-resource-drifts`. The page also warns to confirm the stack's **Last drift check time** is earlier than the resource results, "to prevent the use of stale data". Detection reports; it does not fix anything.

Run here (level 1): I took the page's own example (a queue whose expected and actual properties are printed as JSON) and computed the property differences. The result reproduces the page's output exactly: `/DelaySeconds` actual 120, expected 20, and `/RedrivePolicy/maxReceiveCount` actual 12, expected 10. This only *compares two documents*; real drift detection reads the real resources. `ADD` and `REMOVE` are my extension of the comparison.

```
py -X utf8 drift_demo.py
```

```
computed from the page's ExpectedProperties and ActualProperties:
  /DelaySeconds                    actual 120    expected 20     NOT_EQUAL
  /RedrivePolicy/maxReceiveCount   actual 12     expected 10     NOT_EQUAL
matches the PropertyDifferences printed on the page: True

what else a comparison like this reports (on top of the two documented differences)
  a property added by hand to the live resource  : [('/Tags', 'ADD')]
  a property missing from the live resource      : [('/VisibilityTimeout', 'REMOVE')]
  the expected document compared with itself      : [] (no drift)

ADD and REMOVE are my extension; the page's example prints only NOT_EQUAL. The page says
```

Nested Stacks, StackSets, Exports and Terraform

Nested stacks split a large template into smaller ones; the quotas page notes you can have "a nested stack hierarchy with more than 2500 total resources", but not change more than 2,500 in a single operation. **StackSets** deploy a stack to many accounts and Regions; their limits are in the table above, and "StackSets does not currently support creating or updating stack sets with service-managed permissions from templates that contain macros", which includes transforms. Both are level 3 here. **Outputs and exports:** the outputs page did not load in the tool, so I did not read it; what follows is the tool's paraphrase from general knowledge and is **unverified**: export names must be unique within an account and Region, and a stack whose exported output another stack imports cannot be deleted or have that export changed. Check the page before relying on either. The function list above shows `Fn::ImportValue` and the newer `Fn::GetStackOutput`.

CloudFormation or Terraform? *Terraform / Infrastructure as Code* covers Terraform's own plan, apply and state. I will not repeat that here, and I did not verify claims about Terraform in this

session. The choice usually turns on questions you can answer yourself: does your estate span clouds or services outside AWS, who will operate the tool and hold its state, and which preview (a change set here, a plan there) your reviewers will read. *AWS Intermediate/Advanced 5* adds the AWS CDK, which generates templates like this one from code.

YAML and JSON

A template can be YAML or JSON. YAML allows short-form functions (`!Ref`, `!Sub`) and is much shorter; JSON has no comments and no short forms. **Run here (level 1):** cfn-flip converted the main template to JSON and back.

```
py -X utf8 flip_roundtrip.py
```

```
cfn-flip 1.3.0
parsed data equal: yaml == json: True; yaml -> json -> yaml == original: True
size: YAML 6,187 bytes, JSON 14,023 bytes (227% of the YAML)
short forms in the YAML: 29 | the same functions in the JSON: 29
cfn-lint app_stack.yaml               exit 0 (no findings)
cfn-lint app_stack.json               exit 0 (no findings)
cfn-lint app_stack_roundtrip.yaml     exit 0 (no findings)
cfn-lint bad.yaml                     exit 2 E3002 Additional properties are not allowed
cfn-lint bad.json                     exit 2 E3002 Additional properties are not allowed
```

The data is identical in both forms and after the round trip, the JSON is about 2.3 times the size, and cfn-lint gives the same answer on both (the same rule, E3002, but at a different line, so line numbers in a report refer to whichever file you gave it). The converted JSON is also larger against the 51,200-byte request limit.

Cost and Clean-Up

Nothing in this chapter created anything in AWS. If you deploy `app_stack.yaml` in a sandbox account, it creates a KMS key ($1 a month while it exists, from *AWS Intermediate/Advanced 3*), a bucket and two queues, a role and, in production, an alarm. Because the key and bucket have `Retain` policies, **deleting the stack leaves them behind** and they keep costing money until you remove them. Not run here:

```
# level 2: NOT RUN here. Deleting the stack leaves the Retain resources behind; remove t
aws cloudformation delete-stack --stack-name reports-dev
# then, for each retained resource (steps not spelled out here because I did not read th
#   - the bucket: it has versioning on, so empty it of ALL object versions (the docs say
#   - the key: schedule its deletion (waiting period 7 to 30 days, default 30; irreversi
#   - the queues, role and alarm are deleted with the stack because they have no Retain
```

A key scheduled for deletion waits 7 to 30 days (default 30) and cannot be recovered after that (*AWS Intermediate/Advanced 3*). Put a budget alert in place first (*AWS Fundamentals 9*) and use the sandbox account from *AWS Intermediate/Advanced 1*.

WHAT WAS AND WAS NOT CHECKED

Run here: cfn-lint 1.57.2 on the main template, 22 broken copies, six limit-breaking templates, three documentation examples and Chapter 1's SCP template; the resolver model against 33 documented cases (and one finding); parameter rendering for two environments and four bad inputs; the change set approximation on six edits using cfn-lint's bundled resource schemas; the dependency graph, failure model, capability checker and quota checker; the drift example comparison; the YAML/JSON round trip. **From the documentation, not run:** every statement about change sets, rollback, rollback triggers, deletion policies, drift detection, stack operations and quotas. **Unverified or not read:** the Outputs page (export rules), the Fn::Sub and UpdateReplacePolicy pages, the definitions of the Replacement values True and Conditional, the dynamic references page, nested stacks and StackSets beyond their limits. **Not run at all:** any stack, change set, rollback, drift detection, `deploy` command, nested stack or StackSet.

Practice

EXERCISE 1

Write a template for an encrypted bucket, two queues, a worker role and a production-only alarm, with parameters (constrained), a mapping, a condition, outputs and both deletion policies. Lint it, then break it in about twenty specific ways (misspelt property, wrong type, bad Ref, missing required property, circular dependency, bad condition, bad deletion policy, over a documented quota) and record for each whether cfn-lint caught it, only warned, or missed it. Lint the documentation's own example templates as well.

View solution

EXERCISE 2

Write a resolver for the documented template rules (parameter types and constraints, Ref, Sub, Join, Select, Split, FindInMap, If, the condition functions with their limits, NoValue, conditional resources and outputs) that refuses what it does not model. Run the documentation's own examples as tests with the expected result written first, report anything in the documentation that seems wrong, and render your template for two environments. Convert it to JSON and back and confirm the data survives.

View solution

EXERCISE 3

Approximate a change set locally by diffing two rendered templates and using the create-only properties from cfn-lint's bundled schemas to flag replacements; try six edits including naming a resource and renaming a logical ID. Extract the dependency graph, find cycles, and simulate which resources are created, fail or are never attempted when one fails. Add a capability checker, a quota checker and the drift-example comparison, and label what each cannot know.

View solution

CHAPTER 4 QUICK REFERENCE

- Only `Resources` is required. Parameters (types, constraints, 200 max), mappings, conditions and outputs make one template serve several environments
- `Number` parameters become strings under `Ref`; `AllowedPattern` must match the whole value; a parameter with no value or default fails with `Parameters: [Name] must have values`; quote `'Yes'` and `'No'` in YAML `AllowedValues`
- `NoEcho` does not mask `Outputs` or `Metadata`: keep secrets in Secrets Manager or Parameter Store and use dynamic references
- Condition functions: `And` / `Or` take 2 to 10 conditions; inside `Equals` use only `Ref`, `FindInMap` and condition functions; `Fn::If` with `AWS::NoValue` removes a property; intrinsic functions are not allowed in `DeletionPolicy`
- Quotas: 500 resources, 200 parameters, 200 outputs, 1,024-byte description, 51,200-byte template in a request (1 MB from S3), 2,000 stacks, 128-character stack name
- cfn-lint: exit 2 errors, 4 warnings, 6 both. It missed Sub inside Equals, !If in DeletionPolicy, a bad FindInMap key and a non-numeric Number default, and does not check the parameter count or body size: add a second check

- Change set: create, view (Add, Modify, Remove, property-level), execute; it does not guarantee success. `deploy` creates and executes one (`--no-execute-changeset` to review); IAM templates need `--capabilities`, named IAM resources need `CAPABILITY_NAMED_IAM`
- Naming a resource (`BucketName`, `QueueName`) or renaming a logical ID can replace it; read the Replacement field before executing
- Failure: independent paths continue, dependents of a failure are never attempted; `--disable-rollback` keeps successful resources; rollback triggers use up to 5 CloudWatch alarms and a 0 to 180 minute monitoring time
- `DeletionPolicy`: `Delete` (default; `Snapshot` for RDS clusters and standalone instances), `Retain`, `RetainExceptOnCreate`, `Snapshot`; it does not protect against replacement, so set `UpdateReplacePolicy` too; retained resources keep costing money
- Drift detection reports differences (`detect-stack-drift`, then `describe-stack-resource-drifts`); it does not fix them, and you check the last drift check time

COMING NEXT

Chapter 5 is the AWS CDK: write the same stack in TypeScript, run `cdk synth` for real to produce a template like this one, read what it generated, test the stack with assertions, and decide between CDK, CloudFormation and Terraform. Start with *AWS Intermediate/Advanced 5*.

AWS Intermediate/Advanced

Chapter 5 · The AWS CDK

AWS Intermediate/Advanced 4 wrote a stack as a YAML template. The AWS Cloud Development Kit (CDK) lets you write the same thing as ordinary code (here TypeScript): you build objects, run the program, and it *generates* the CloudFormation template. Deployment is still CloudFormation, so everything in chapter 4 still applies. What changes is how you write the template, how you test it, and which surprises are waiting. This chapter rebuilds the chapter 4 stack in the CDK, runs `cdk synth` for real, reads what comes out, and measures it against the hand-written version.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS credentials on the machine that wrote this course, so nothing was deployed.

- **Run here (level 1):** Node 24.18.0 on Windows 11, with `aws-cdk` (the CLI) 2.1145.0, `aws-cdk-lib` 2.273.0, `constructs` 10.8.1, TypeScript 7.0.2 and Jest 30.5.2, installed into a scratch folder outside this site (the install is about 277 MB; nothing was installed globally). `cdk synth` was run for real, the tests were run for real, and the generated templates were checked with the cfn-lint and models from chapter 4.
- **Needs an AWS account (level 2):** `cdk bootstrap`, `cdk deploy`, `cdk diff`, `cdk destroy` and context lookups. I ran them anyway, with no credentials, so the chapter shows how they fail, not how they succeed.
- **From the documentation, not run (level 3):** what the CDK guide says about constructs, apps, bootstrapping, the CLI, testing and escape hatches (read 2026-10-13 through a page-fetch tool that summarises pages).

I chose TypeScript because the outline's default was TypeScript; nothing here depends on that. I did not use `cdk init`, so this project is hand-made and its feature flags are unset: that matters, and the chapter shows how.

Apps, Stacks and Constructs

The CDK guide's definitions. An **app** "is a collection of one or more CDK stacks. Stacks are a collection of one or more constructs, which define AWS resources and properties." `App` "is the only construct that can be used as the root", and "all constructs that represent AWS resources

must be defined, directly or indirectly, within the scope of a `Stack`". "Apps are then synthesized to create AWS CloudFormation templates for your stacks." Constructs come in three levels:

LEVEL	WHAT THE GUIDE SAYS	EXAMPLE IN THIS CHAPTER
L1 (CFN resources)	"the lowest-level construct and offer no abstraction. Each L1 construct maps directly to a single AWS CloudFormation resource"; names start with `Cfn`	`s3.CfnBucket`
L2 (curated)	Higher-level constructs with sensible defaults and "helper methods for most resources that make it simpler and quicker to define properties, permissions, event-based interactions"	`s3.Bucket`, `kms.Key`, `sqs.Queue`, `iam.Role`
L3 (patterns)	"the highest-level of abstraction ... a collection of resources that are configured to work together to accomplish a specific task"	The guide's example: `ecsPatterns.ApplicationLoadBalancedFargateService` (not used here)

Constructs form a **construct tree**: every construct is created with a `scope` (its parent) and an `id`, and the app is the root. "The AWS CDK uses the IDs of all constructs in the path from the tree's root to each child construct to generate the unique IDs required by AWS CloudFormation." That sentence has a consequence, which the guide states and this chapter measures: "if you move a construct to a different scope, its generated stack-unique ID changes, and AWS CloudFormation won't consider it the same resource" (see Construct IDs below). The newer CDK also has *Mixins* and *Facades* (the guide describes grants as a facade); I did not use them.

The Project

The layout is small: `bin/app.ts` (creates the app and the stack), `lib/reports-stack.ts` (the stack), `lib/aspects.ts`, `test/`, `cdk.json` (tells the CLI how to run the app) and `package.json`. Two practical notes from building it with TypeScript 7.0.2: its compiler **rejects** `"moduleResolution": "node"` ("Option 'moduleResolution=node10' has been removed"), so the project uses `node16`; and because I did not want a TypeScript runner in the loop, the app is compiled first and `cdk.json` runs the compiled file:

```
{
  "app": "node dist/bin/app.js",
  "context": {
    "environment": "dev"
  }
}
```

The CLI reads `cdk.json`, runs `node dist/bin/app.js`, and the app writes a *cloud assembly* (a folder, `cdk.out` by default, holding one template per stack). The environment name comes in as context (`-c environment=prod`).

The Stack in TypeScript

The same design as chapter 4: a key, an encrypted bucket, two queues, a worker role, and an alarm in production only. Note where the chapter 4 features went. The Mappings table is a plain object; the condition is a plain `if`; the parameter constraints are a check that throws; there are no CloudFormation Parameters or Conditions of my own, because those decisions are made *while the program runs*, not when CloudFormation deploys.

```
    this.key = new kms.Key(this, "DataKey", {
      description: `Data key for ${this.stackName} (${props.environment})`,
      enableKeyRotation: true,
      alias: `alias/${this.stackName}-data`,
      removalPolicy: cdk.RemovalPolicy.RETAIN,
    });
    this.key.addToResourcePolicy(new iam.PolicyStatement({
      sid: "Allow access for Key Administrators",
      principals: [new iam.ArnPrincipal(props.keyAdminRoleArn)],
      actions: ["kms:Create*", "kms:Describe*", "kms:Enable*", "kms:List*", "kms:Put*",
                "kms:Get*", "kms:Delete*", "kms:TagResource", "kms:UntagResource", "kms:
      resources: ["*"],
    }));

    this.bucket = new s3.Bucket(this, "ReportBucket", {
      encryption: s3.BucketEncryption.KMS,
      encryptionKey: this.key,
      bucketKeyEnabled: true,
      versioned: true,
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
```

```
      enforceSSL: true,
      lifecycleRules: [{ id: "expire-reports", expiration: cdk.Duration.days(props.reten
      removalPolicy: cdk.RemovalPolicy.RETAIN,
    });

    const dlq = new sqs.Queue(this, "WorkDeadLetterQueue", {
      retentionPeriod: cdk.Duration.days(14),
      encryption: sqs.QueueEncryption.KMS,
      encryptionMasterKey: this.key,
    });
    this.queue = new sqs.Queue(this, "WorkQueue", {
      visibilityTimeout: cdk.Duration.seconds(s.visibilityTimeout),
      encryption: sqs.QueueEncryption.KMS,
      encryptionMasterKey: this.key,
      deadLetterQueue: { queue: dlq, maxReceiveCount: s.maxReceive },
    });

    const role = new iam.Role(this, "WorkerRole", { assumedBy: new iam.ServicePrincipal(
    this.queue.grantConsumeMessages(role);      // intent-based grants: CDK writes the IA
    this.bucket.grantRead(role);                 // grantReadWrite also allows s3:Delete
    this.bucket.grantPut(role);
```

Three things to notice before running anything. **Grants are intent-based:** `queue.grantConsumeMessages(role)` says what the role should be able to do, and the CDK writes the IAM statements (and, for the key, the extra permissions the bucket's encryption needs). **Removal policies:** `RemovalPolicy.RETAIN` is the chapter 4 `DeletionPolicy` and `UpdateReplacePolicy` pair. **Safety options are one word:** `enforceSSL: true` becomes a bucket policy and `blockPublicAccess` becomes four properties. The rest of the file (the alarm and the outputs):

```
    if (props.environment === "prod") {          // an ordinary 'if', evaluated when the
      const alarm = new cloudwatch.Alarm(this, "BacklogAlarm", {
        alarmDescription: "Work queue backlog is growing",
        metric: this.queue.metricApproximateNumberOfMessagesVisible({ period: cdk.Durati
        threshold: 1000,
        evaluationPeriods: 1,
        comparisonOperator: cloudwatch.ComparisonOperator.GREATER_THAN_THRESHOLD,
        treatMissingData: cloudwatch.TreatMissingData.NOT_BREACHING,
      });
      new cdk.CfnOutput(this, "AlarmName", { value: alarm.alarmName, description: "Backl
    }
```

```
    new cdk.CfnOutput(this, "BucketName", { value: this.bucket.bucketName, description:
    new cdk.CfnOutput(this, "QueueUrl", { value: this.queue.queueUrl, description: "URL
    new cdk.CfnOutput(this, "KeyArn", { value: this.key.keyArn, description: "ARN of the
  }
}
```

cdk synth, for Real

Run here (level 1): `cdk synth -c environment=dev`. It succeeded (about 18 seconds the first time, 7 to 8 seconds later) and printed two lines on its error stream that are not part of the template: `AI agent detected` and `84 feature flags are not configured. Run 'cdk flags --unstable=flags' to learn more.` (I report them as printed; the second one is explained under Feature Flags). The generated template, 372 lines of YAML, in its two interesting parts: the worker role and the policy the grants wrote, and the extra sections the CDK adds.

```
WorkerRole8DD27D41:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Statement:
        - Action: sts:AssumeRole
          Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
      Version: "2012-10-17"
    Tags:
      - Key: project
        Value: reports
  Metadata:
    aws:cdk:path: Reports-dev/WorkerRole/Resource
WorkerRoleDefaultPolicy1750E153:
  Type: AWS::IAM::Policy
  Properties:
    PolicyDocument:
      Statement:
        - Action:
            - sqs:ReceiveMessage
            - sqs:ChangeMessageVisibility
            - sqs:GetQueueUrl
```

```
        - sqs:DeleteMessage
        - sqs:GetQueueAttributes
      Effect: Allow
      Resource:
        Fn::GetAtt:
          - WorkQueue94013F35
          - Arn
    - Action: kms:Decrypt
      Effect: Allow
      Resource:
        Fn::GetAtt:
          - DataKey17AFBB84
          - Arn
    - Action:
        - s3:GetObject*
        - s3:GetBucket*
        - s3:List*
      Effect: Allow
      Resource:
        - Fn::GetAtt:
            - ReportBucket577F0FCD
            - Arn
        - Fn::Join:
            - ""
            - - Fn::GetAtt:
                  - ReportBucket577F0FCD
                  - Arn
              - /*
    - Action:
        - kms:Decrypt
        - kms:DescribeKey
      Effect: Allow
      Resource:
        Fn::GetAtt:
          - DataKey17AFBB84
          - Arn
    - Action:
        - s3:PutObject
        - s3:PutObjectLegalHold
        - s3:PutObjectRetention
        - s3:PutObjectTagging
        - s3:PutObjectVersionTagging
        - s3:Abort*
      Effect: Allow
      Resource:
```

```
              Fn::Join:
                - ""
                - - Fn::GetAtt:
                      - ReportBucket577F0FCD
                      - Arn
                  - /*
            - Action:
                - kms:Encrypt
                - kms:ReEncrypt*
                - kms:GenerateDataKey*
                - kms:Decrypt
              Effect: Allow
              Resource:
                Fn::GetAtt:
                  - DataKey17AFBB84
                  - Arn
          Version: "2012-10-17"
        PolicyName: WorkerRoleDefaultPolicy1750E153
        Roles:
          - Ref: WorkerRole8DD27D41
      Metadata:
        aws:cdk:path: Reports-dev/WorkerRole/DefaultPolicy/Resource
    CDKMetadata:
      Type: AWS::CDK::Metadata
```

```
            - us-west-2
  Parameters:
    BootstrapVersion:
      Type: AWS::SSM::Parameter::Value<String>
      Default: /cdk-bootstrap/hnb659fds/version
      Description: Version of the CDK Bootstrap resources in this environment, automatical
```

What you are looking at. **Logical IDs** such as `WorkerRole8DD27D41` are the construct path plus a hash, and each resource carries `aws:cdk:path` metadata so you can find the code that made it. The **role's permissions are a separate** `AWS::IAM::Policy` resource (the hand-written template used an inline policy). There is an `AWS::CDK::Metadata` resource (version reporting, on by default in the CLI documentation's options) and a `BootstrapVersion` **parameter** that reads `/cdk-bootstrap/hnb659fds/version` from Parameter Store at deploy time: the template itself expects the environment to have been bootstrapped. The tag from `Tags.of(app)` was applied to every taggable resource without any resource mentioning it.

Against the hand-written template

I compared the generated templates (dev and prod) with the chapter 4 template, using the chapter 4 tools: the resolver renders the hand-written one, the change set approximation compares the two generated ones, and cfn-lint checks them. (Output of a script that reads only real files.)

```
C:/lwaws/c4/.venv/Scripts/python.exe compare_templates.py
```

```
1. resource types: hand-written template (rendered by the chapter 4 resolver) against wh
   dev: hand-written 7 resources, CDK 9 resources
      AWS::CDK::Metadata           hand-written 0   CDK 1   <-- differs
      AWS::IAM::Policy             hand-written 0   CDK 1   <-- differs
      AWS::IAM::Role               hand-written 1   CDK 1
      AWS::KMS::Alias              hand-written 1   CDK 1
      AWS::KMS::Key                hand-written 1   CDK 1
      AWS::S3::Bucket              hand-written 1   CDK 1
      AWS::S3::BucketPolicy        hand-written 1   CDK 1
      AWS::SQS::Queue              hand-written 2   CDK 2
   prod: hand-written 8 resources, CDK 10 resources
      AWS::CDK::Metadata           hand-written 0   CDK 1   <-- differs
      AWS::CloudWatch::Alarm       hand-written 1   CDK 1
      AWS::IAM::Policy             hand-written 0   CDK 1   <-- differs
      AWS::IAM::Role               hand-written 1   CDK 1
      AWS::KMS::Alias              hand-written 1   CDK 1
      AWS::KMS::Key                hand-written 1   CDK 1
      AWS::S3::Bucket              hand-written 1   CDK 1
      AWS::S3::BucketPolicy        hand-written 1   CDK 1
      AWS::SQS::Queue              hand-written 2   CDK 2

2. sections of the generated template: CDK made these decisions while the app ran, so th
   dev: sections ['Conditions', 'Outputs', 'Parameters', 'Resources', 'Rules']; Paramete
        hand-written template sections: ['AWSTemplateFormatVersion', 'Conditions', 'Desc
   prod: sections ['Conditions', 'Outputs', 'Parameters', 'Resources', 'Rules']; Paramet
        hand-written template sections: ['AWSTemplateFormatVersion', 'Conditions', 'Desc

3. the worker role's allowed actions: hand-written inline policy against the policy CDK'
   hand-written (7): ['kms:Decrypt', 'kms:GenerateDataKey', 's3:GetObject', 's3:PutObjec
   CDK          (19): ['kms:Decrypt', 'kms:DescribeKey', 'kms:Encrypt', 'kms:GenerateDat
   only in CDK's policy:         ['kms:DescribeKey', 'kms:Encrypt', 'kms:GenerateDataKey
   only in the hand-written one: ['kms:GenerateDataKey', 's3:GetObject']

4. DeletionPolicy / UpdateReplacePolicy per resource
   CDK AWS::KMS::Key            DeletionPolicy Retain   UpdateReplacePolicy Retain
```

```
    CDK AWS::KMS::Alias          DeletionPolicy -        UpdateReplacePolicy -
    CDK AWS::S3::Bucket          DeletionPolicy Retain   UpdateReplacePolicy Retain
    CDK AWS::S3::BucketPolicy    DeletionPolicy -        UpdateReplacePolicy -
    CDK AWS::SQS::Queue          DeletionPolicy Delete   UpdateReplacePolicy Delete
    CDK AWS::SQS::Queue          DeletionPolicy Delete   UpdateReplacePolicy Delete
    CDK AWS::IAM::Role           DeletionPolicy -        UpdateReplacePolicy -
    CDK AWS::IAM::Policy         DeletionPolicy -        UpdateReplacePolicy -
    CDK AWS::CDK::Metadata       DeletionPolicy -        UpdateReplacePolicy -

  5. dev -> prod, as the chapter 4 change set approximation sees the two CDK templates

  -- CDK Reports-dev -> Reports-prod (the stack name also changes, so names derived from i
     Add    BacklogAlarm1E83516A  AWS::CloudWatch::Alarm    Replacement: -
     Modify DataKey17AFBB84       AWS::KMS::Key             Replacement: False
              Modify /Properties/Description                                  "Data k
     Modify DataKeyAliasF989EB84  AWS::KMS::Alias           Replacement: True
              Modify /Properties/AliasName                                    "alias/
     Modify WorkQueue94013F35     AWS::SQS::Queue           Replacement: False
              Modify /Properties/RedrivePolicy/maxReceiveCount                3 -> 5
              Modify /Properties/VisibilityTimeout                            30 -> 1

  6. capabilities and quotas, with the chapter 4 checkers
     dev: capability CAPABILITY_IAM; over a quota: none; template 13,372 bytes
     prod: capability CAPABILITY_IAM; over a quota: none; template 14,298 bytes

  7. cfn-lint (chapter 4 environment) on the generated templates
     Reports-dev.template.json: exit 0 (no findings)
     Reports-prod.template.json: exit 0 (no findings)
```

The same resource types appear, plus the policy resource and the metadata resource: 9 resources against 7 in dev, 10 against 8 in prod. **Both generated templates lint clean** (cfn-lint exit status 0) and need only `CAPABILITY_IAM`. The generated template has no `Mappings`, no `Parameters` of its own and, apart from the CDK's metadata one, no `Conditions`. Two differences deserve attention. **The role's permissions are much wider** than the hand-written ones: 19 distinct actions against 7, even after I narrowed them (next section). **Defaults you did not write:** the queues get `DeletionPolicy: Delete` and `UpdateReplacePolicy: Delete` (the hand-written template left them unset), while the key and bucket got `Retain` only because I asked. The dev-to-prod comparison shows the alias name changing: here that is only because the stack name contains the environment, and dev and prod are separate stacks.

Grants, and the Test That Caught One

Grants save a lot of typing and hide a lot. My first version of the stack used `bucket.grantReadWrite(role)`. It looked right, and I had written a test for least privilege: "the worker role may not delete objects and has no wildcard action". The test failed, which is what it is for. **Run here (level 1):**

```
jest (first run: 13 passed, 1 failed on purpose)
```

```
FAIL dist/test/reports-stack.test.js (8.296 s)
  ● ReportsStack, fine-grained assertions › least privilege: the worker role is not allow

    expect(received).toEqual(expected) // deep equality

    - Expected  - 1
    + Received  + 3

    - Array []
    + Array [
    +   "s3:DeleteObject*",
    + ]

      101 |         const allowed = policies.flatMap((p) => p.Properties.PolicyDocument.S
      102 |         expect(allowed.filter((a) => a === "*" || a.endsWith(":*"))).toEqual
    > 103 |         expect(allowed.filter((a) => a.startsWith("s3:Delete"))).toEqual([])
          |                                                                  ^
      104 |     });
      105 |     test("the template has no CloudFormation Conditions or Mappings of its ow
      106 |         const j = dev.toJSON();

      at Object.toEqual (dist/test/reports-stack.test.js:103:66)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 13 passed, 14 total
Snapshots:   1 passed, 1 total
Time:        9.654 s
Ran all test suites.
```

`grantReadWrite` had added `s3:DeleteObject*`. The worker only reads and writes, so I replaced it with `grantRead` plus `grantPut`. That changed the generated template, and the *snapshot* test

(a stored copy of the whole template) failed in a different way, which is exactly what the guide warns about:

```
jest (second run: after narrowing the grants)
```

```
FAIL dist/test/reports-stack.test.js (8.296 s)
  ● snapshot › the dev template matches the stored baseline

    expect(received).toMatchSnapshot()

    Snapshot name: `snapshot the dev template matches the stored baseline 1`

    - Snapshot  -  7
    + Received  + 37

    @@ -281,17 +281,10 @@
                  {
                    "Action": [
                      "s3:GetObject*",
                      "s3:GetBucket*",
                      "s3:List*",
    -                 "s3:DeleteObject*",
    -                 "s3:PutObject",
    -                 "s3:PutObjectLegalHold",
    -                 "s3:PutObjectRetention",
    -                 "s3:PutObjectTagging",
    -                 "s3:PutObjectVersionTagging",
    -                 "s3:Abort*",
                    ],
                    "Effect": "Allow",
                    "Resource": [
                      {
                        "Fn::GetAtt": [
```

```
jest (second run, summary)
```

```
 › 1 snapshot failed.
Snapshot Summary
 › 1 snapshot failed from 1 test suite. Inspect your code changes or re-run jest with `-

Test Suites: 1 failed, 1 total
```

```
Tests:       1 failed, 13 passed, 14 total
Snapshots:   1 failed, 1 total
Time:        9.679 s
Ran all test suites.
```

The failing snapshot shows the delete permission gone and the put permissions moved to their own statement. The change was intentional, so I accepted the new baseline with `jest -u`, and then everything passed:

```
jest -u
```

```
Snapshot Summary
 › 1 snapshot updated from 1 test suite.

Test Suites: 1 passed, 1 total
Tests:       14 passed, 14 total
Snapshots:   1 updated, 1 total
Time:        7.705 s, estimated 9 s
Ran all test suites.
```

Testing a Stack

The guide describes two kinds of tests: "**Fine-grained assertions** test specific aspects of the generated AWS CloudFormation template" and are "the most frequently used"; "**Snapshot tests** test the synthesized AWS CloudFormation template against a previously stored baseline", which let you refactor freely, but "CDK upgrades can also cause synthesized templates to change, so you can't rely only on snapshots". The tools are `Template.fromStack`, `hasResourceProperties` (a *partial* match by default), `resourceCountIs` and the `Match` helpers. My 14 tests, all passing at the end:

```
jest results (names)
```

```
passed  ReportsStack, fine-grained assertions the data key rotates automatically and is
passed  ReportsStack, fine-grained assertions the key policy keeps the root statement th
passed  ReportsStack, fine-grained assertions the bucket is encrypted with the key, vers
passed  ReportsStack, fine-grained assertions the bucket policy denies requests that do
passed  ReportsStack, fine-grained assertions every queue is encrypted with the key
```

```
  passed  ReportsStack, fine-grained assertions the work queue's settings come from the en
  passed  ReportsStack, fine-grained assertions the backlog alarm exists only in productio
  passed  ReportsStack, fine-grained assertions a retention outside 1 to 3650 days stops s
  passed  ReportsStack, fine-grained assertions least privilege: the worker role is not al
  passed  ReportsStack, fine-grained assertions the template has no CloudFormation Conditi
  passed  snapshot the dev template matches the stored baseline
  passed  aspects and environments the versioning aspect turns an unversioned bucket into
  passed  aspects and environments the same aspect is quiet for a versioned bucket
  passed  aspects and environments an environment-agnostic stack leaves the Region as a to

  14 passed, 0 failed of 14
```

How they are built: a helper creates an `App` and the stack for a given environment and returns `Template.fromStack(stack)`, and each test asserts one thing. The least-privilege test is the one that mattered:

```
  test("least privilege: the worker role is not allowed to delete objects or use a wildc
    const policies = Object.values(dev.findResources("AWS::IAM::Policy"));
    const allowed: string[] = policies.flatMap((p) => p.Properties.PolicyDocument.Statem
    expect(allowed.filter((a) => a === "*" || a.endsWith(":*"))).toEqual([]);
    expect(allowed.filter((a) => a.startsWith("s3:Delete"))).toEqual([]);
  });
```

The tests ran against compiled JavaScript (Jest finds the compiled files in `dist/test`), because I did not want a TypeScript transform layer between Jest and TypeScript 7. Also: a test template does not contain the CLI's `CDKMetadata` resource (my first version of the "no Conditions" test assumed it did and failed; the test now ignores that one name). Tests need no AWS account and ran in about 10 seconds.

The guide's own testing example

I typed in the guide's example (a stack with a state machine, a Lambda function and an SNS subscription) and the assertions it prints. It compiles, and the test fails as printed:

```
jest on the guide's example (selected lines)
```

```
[WARNING] aws-cdk-lib.aws_stepfunctions.StateMachineProps#definition is deprecated.
use definitionBody: DefinitionBody.fromChainable()
WARNING Runtime: Runtime 'nodejs18.x' was deprecated on '2025-09-01'. Creation will be d
```

```
Template has 1 resources with type AWS::Lambda::Function, but none match as expected.
!!   Expected nodejs14.x but received nodejs18.x
Tests:       1 failed, 1 total
```

The stack in the guide creates the function with `Runtime.NODEJS_18_X` and the assertion on the same page expects `"nodejs14.x"`, so the example cannot pass. It also uses the `definition` property, which the library now marks deprecated ("use definitionBody"), and Node.js 18 as a Lambda runtime, which the CloudFormation validation notice says was deprecated on 2025-09-01. The lesson is not about that page: documentation examples age, and a test that fails with a clear diff ("Expected nodejs14.x but received nodejs18.x") is how you find out.

Aspects: Rules That Apply to the Whole Tree

An **Aspect** visits every construct in the tree (the guide: "a simple way to visit every node in the construct tree and perform an operation on each one"). Mine checks every bucket for versioning and, if it is missing, adds an *error annotation*:

```
import * as cdk from "aws-cdk-lib";
import * as s3 from "aws-cdk-lib/aws-s3";
import { IConstruct } from "constructs";

/** An Aspect visits every construct in the tree. This one turns a missing rule into a sy
export class RequireBucketVersioning implements cdk.IAspect {
  visit(node: IConstruct): void {
    if (node instanceof s3.CfnBucket) {
      const v = node.versioningConfiguration as s3.CfnBucket.VersioningConfigurationProp
      if (!v || v.status !== "Enabled") {
        cdk.Annotations.of(node).addError("Bucket versioning must be Enabled (organisati
      }
    }
  }
}
```

Run here (level 1): `cdk synth` on an app with an unversioned bucket, and on the real app with the aspect attached:

```
cdk synth (the aspect fails an unversioned bucket) and the no-credentials runs
```

```
== cdk synth of an app that breaks the organisation rule (unversioned bucket)  -> exit s
AI agent detected
ERROR Bucket versioning must be Enabled (organisation rule) (Construct Annotations)
   Unversioned/Plain/Resource
   Rule Annotation::error-annotation
Synthesis finished with errors
   (the cloud assembly files were still written: cdk.out manifest.json tree.json Unversi
```

The synthesis printed the error and **exited with status 1**, but the cloud assembly files, including a template for the bad stack, were still written. A pipeline must test the exit status, never the existence of a template. Everything else the aspect did is checked in the tests above (a versioned bucket is quiet).

Escape Hatches

When an L2 construct lacks something, the guide lists the ways out: use the `Cfn` L1 directly; reach the L1 inside an L2 with `construct.node.defaultChild` and change it; use raw overrides (`addOverride`, `addPropertyOverride`, `addPropertyDeletionOverride`); or use `CfnResource` for a type the CDK does not know. There are also "un-escape hatches" (`Bucket.fromCfnBucket`) and, for features CloudFormation lacks, custom resources. I tried the raw route on a bucket created as `versioned: true`:

```
import * as cdk from "aws-cdk-lib";
import * as s3 from "aws-cdk-lib/aws-s3";
import { RequireBucketVersioning } from "../lib/aspects";

// escape hatches: reach the L1 resource inside an L2 and rewrite it with raw overrides
const app = new cdk.App();
const stack = new cdk.Stack(app, "Hatch");
const bucket = new s3.Bucket(stack, "Reports", { versioned: true });
const cfn = bucket.node.defaultChild as s3.CfnBucket;
cfn.addPropertyOverride("VersioningConfiguration.Status", "Suspended");      // raw over
cfn.addPropertyOverride("ObjectLockEnabled", true);                          // a proper
cfn.addMetadata("reviewed-by", "platform-team");                             // template
```

```
cfn.cfnOptions.deletionPolicy = cdk.CfnDeletionPolicy.RETAIN;                // a CloudF
cdk.Aspects.of(app).add(new RequireBucketVersioning());                      // the orga
```

Run here (level 1): `cdk synth` exited with status 0, and wrote:

```
Resources:
  Reports65402A0A:
    Type: AWS::S3::Bucket
    Properties:
      ObjectLockEnabled: true
      VersioningConfiguration:
        Status: Suspended
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
    Metadata:
      aws:cdk:path: Hatch/Reports/Resource
      reviewed-by: platform-team
  CDKMetadata:
```

The override **won**: the template says `Status: Suspended`, a property the L2 never set (`ObjectLockEnabled`) appeared, and the metadata and `Retain` policy came through. And **the organisation's versioning aspect did not complain**, because it reads the typed property (still `Enabled`) while the raw override changes only the final template. So an escape hatch is a way around the CDK's own safety checks as well as its limits. If you use one, put a test on the generated template (as above) rather than relying on the aspect.

Construct IDs and Refactoring

The guide's warning about moving a construct is easy to check. This program prints the logical IDs of "the same bucket" in different places in the tree:

```
console.log("bucket directly in the stack  :", ids((s) => new s3.Bucket(s, "ReportBucket
console.log("same code run again          :", ids((s) => new s3.Bucket(s, "ReportBucket
console.log("bucket inside a construct     :", ids((s) => new Storage(s, "Storage")));
console.log("construct renamed to Archive  :", ids((s) => new Storage(s, "Archive")));
console.log("bucket id changed to Reports  :", ids((s) => new s3.Bucket(s, "Reports")));
```

```
node dist/bin/ids.js
```

```
bucket directly in the stack  : [ 'ReportBucket577F0FCD' ]
same code run again           : [ 'ReportBucket577F0FCD' ]
bucket inside a construct     : [ 'StorageReportBucketC5ACC6CD' ]
construct renamed to Archive  : [ 'ArchiveReportBucketFA2C8983' ]
bucket id changed to Reports  : [ 'Reports65402A0A' ]
```

The IDs are stable when the code is the same, and different the moment the bucket moves into a wrapper construct, the wrapper is renamed, or the id changes. CloudFormation sees a different logical ID as a different resource: a *remove* and an *add*, as chapter 4's change set experiment showed, and for a bucket or a key with data in it that is a serious event. The CLI has a `cdk refactor` command described as "Preserve deployed resources when refactoring code in your CDK application"; I did not read or run it. The safe habit: treat construct IDs and scopes as part of the stack's identity, and pin the logical ID or test for it before refactoring.

Environments, Context and Bootstrapping

A stack can be **environment-agnostic** (no account or Region given: the Region and account stay as tokens resolved at deploy time) or **environment-specific** (`env: { account, region }`). My tests check both: the Region is an unresolved token in the first case and `eu-west-1` in the second. Both synthesize without credentials as long as nothing needs to *ask AWS*. Things that do (looking up a VPC, an AMI, a Route 53 zone) are **context lookups**: the CLI option reference says `--lookups` defaults to true and "Synthesis will fail if this value is `false` and context lookups need to be performed". (A file named `cdk.context.json` appeared in the project holding just `{}` after the failed lookup below; I did not read the page on context, so I will not describe its caching rules.)

Bootstrapping prepares an environment: "Before you deploy a CDK stack into an AWS environment, the environment must first be bootstrapped." It deploys a stack named `CDKToolkit` containing "an Amazon S3 bucket" (for files and assets), "an Amazon ECR repository" (for Docker images) and "IAM roles" that the CDK uses to deploy; "each environment that you want to use with the AWS CDK must first be bootstrapped". (I did not read what the bootstrap resources cost.) The generated template's `BootstrapVersion` parameter is how a stack checks that.

Run here (level 1), with no credentials: what the account-dependent commands do. Every one printed the same message and exited with status 1:

```
cdk diff, deploy, bootstrap, destroy and a context lookup, without credentials
```

```
== cdk diff (no AWS credentials on this machine)  -> exit status 1
AI agent detected
Need to perform AWS calls for account 111122223333, but no credentials have been configu
Need to perform AWS calls for account 111122223333, but no credentials have been configu
== cdk deploy (no AWS credentials on this machine)  -> exit status 1
AI agent detected

✨  Synthesis time: 8.17s

Need to perform AWS calls for account 111122223333, but no credentials have been configu
== cdk bootstrap (no AWS credentials on this machine)  -> exit status 1
AI agent detected
⏳  Bootstrapping environment aws://111122223333/eu-west-1...
❌  Environment aws://111122223333/eu-west-1 failed bootstrapping: NoCredentials: Need
Need to perform AWS calls for account 111122223333, but no credentials have been configu
== cdk destroy (no AWS credentials on this machine)  -> exit status 1
AI agent detected
Reports-dev: destroying... [1/1]
❌  Reports-dev: destroy failed NoCredentials: Need to perform AWS calls for account 111
Need to perform AWS calls for account 111122223333, but no credentials have been configu
== cdk synth of an app that does a context lookup (Vpc.fromLookup)  -> exit status 1
AI agent detected
ERROR Need to perform AWS calls for account 111122223333, but no credentials have been c
   Lookup
   Rule Annotation::error-annotation
Synthesis finished with errors
   (the cloud assembly files were still written: cdk.out Lookup.assets.json Lookup.metad
```

The CLI command list in the guide (acknowledge, bootstrap, context, deploy, destroy, diff, docs, doctor, drift, flags, import, init, list, metadata, migrate, notices, orphan, publish-assets, refactor, synthesize, watch) is the map. The commands that need an account are `bootstrap`, `deploy`, `destroy`, `diff`, `drift`, `import` and anything that performs lookups. Level 2, **not run here** (account and Region are placeholders):

```
# level 2: need AWS credentials and an account; NOT RUN here (they failed with "no crede
# one-time per environment: create the CDKToolkit stack (bucket, ECR repository, IAM rol
```

```
cdk bootstrap aws://111122223333/eu-west-1

# see what would change, then deploy; the environment name is passed as context
cdk diff    -c environment=prod -c account=111122223333 -c region=eu-west-1
cdk deploy  -c environment=prod -c account=111122223333 -c region=eu-west-1

# remove what the stack created (the key and the bucket stay: RemovalPolicy.RETAIN)
cdk destroy -c environment=prod -c account=111122223333 -c region=eu-west-1
```

Feature flags

The CDK changes some defaults over time, and to avoid changing existing templates it hides the new behaviour behind **feature flags**. A project created with `cdk init` is, I understand, set up with the current flags (I did not use it or read that page); mine was made by hand, which is why `cdk synth` said "84 feature flags are not configured". **Run here (level 1):** `cdk flags --unstable=flags` lists them. The table has four columns (the flag, the recommended value, the value you set, and the effective value) and for my project 84 flags show `<unset>` with an effective value of `false`, including ones that touch this stack, such as `@aws-cdk/aws-iam:minimizePolicies`, `@aws-cdk/aws-kms:aliasNameRef` and `@aws-cdk/aws-s3:publicAccessBlockedByDefault`. To see what one flag does, I set `@aws-cdk/aws-iam:minimizePolicies` on the command line and compared the worker's policy:

```
node (compare the policy with and without the flag)
```

```
flag unset (default)      6 statements: 5 actions, 1 actions, 3 actions, 2 actions, 6 ac
minimizePolicies = true   4 statements: 5 actions, 5 actions, 3 actions, 6 actions
actions listed (with repeats): 21 -> 19; distinct: 19 -> 19
```

The same 19 distinct actions, in 4 statements instead of 6. That one is harmless, but the point is that the same code can produce different templates depending on flags, so set them deliberately (the CLI offers `cdk flags --set ...` and a `--safe` option for flags "that don't change templates"; I did not run those), keep them in `cdk.json`, and let a snapshot test tell you when they change.

CDK, CloudFormation or Terraform?

What this chapter measured, as a list of trade-offs rather than a verdict:

CDK GAVE	CDK COST (MEASURED ABOVE)
One word for a safety setting (`enforceSSL`, `blockPublicAccess`); intent-based grants	Grants wider than I would write by hand (19 actions against 7); a test caught one delete permission
Logic in a real language: tables, `if`, loops, types, unit tests with Jest	No CloudFormation Conditions or Parameters: decisions are fixed at synth time, so one generated template is one environment
A template you can still read, lint and diff (it linted clean)	Extra generated resources and metadata (9 resources against 7), defaults you did not write (`Delete` on queues)
Aspects that apply a rule to the whole tree	Raw overrides bypass the typed API and the aspects (the `Suspended` bucket)
Fast feedback: synth in 8 to 18 seconds, tests in about 10, no account	Account-dependent steps (lookups, diff, deploy) fail without credentials; documentation examples that no longer pass; a version-coupled CLI and library and 84 feature flags
Identity risk: construct IDs and scopes generate logical IDs, so a tidy refactor can be a delete and a create	

Hand-written CloudFormation is the most direct and the least abstract; the CDK is worth it when you write many similar stacks or want real tests and reuse; *Terraform / Infrastructure as Code* covers the third option, which I did not test here, so I make no claims about it beyond that pointer. In every case the thing that gets deployed is the same kind of object, a template, and chapter 4's tools (lint, change set, failure model) apply to it.

Cost and Clean-Up

Nothing in this chapter created anything in AWS. The only thing left on this machine is the scratch folder (the 277 MB of installed packages and the generated `cdk.out*` folders); it is outside this site and can be deleted. If you deploy this stack for real (level 2, not run), `cdk destroy` removes what the stack created, except the key and the bucket, because their removal policy is `RETAIN`: the chapter 4 clean-up notes apply (empty and delete the bucket, schedule the key's deletion, 7 to 30 days, irreversible). Bootstrapping an environment leaves a

`CDKToolkit` stack and its bucket, repository and roles in the account until you remove them; I did not check their cost.

WHAT WAS AND WAS NOT CHECKED

Run here: `cdk synth` for dev and prod, for an aspect-failing app, an escape-hatch app and a lookup app; 14 Jest tests (including the deliberate failure, the snapshot failure and the update); the guide's own testing example; the generated templates compared with the chapter 4 template and linted with cfn-lint; the construct ID experiment; `cdk ls`, `cdk doctor` and `cdk flags`; the effect of one feature flag; and `cdk diff`, `deploy`, `bootstrap` and `destroy` without credentials (failures only). **From the documentation, not run:** the construct levels, app and bootstrapping descriptions, the CLI command list and options, the testing guidance and the escape hatch API (read 2026-10-13). **Not read or not run:** the Aspects, Stacks, Environments and feature-flag pages themselves; `cdk init`, `cdk refactor`, `cdk watch`, `cdk drift`, `cdk import`; cross-stack references; custom resources; Mixins; other languages; and every command that needs an account. **Not checked at all:** a real deployment of this stack.

Practice

EXERCISE 1

Create a CDK project by hand (TypeScript, scratch folder, nothing installed globally) and rebuild the encrypted bucket, two queues, worker role and production-only alarm from the CloudFormation chapter. Run `cdk synth` for dev and prod, lint the generated templates with cfn-lint, and compare them with the hand-written template: resource types, sections, the role's allowed actions and the deletion policies. Report every difference you did not write yourself.

View solution

EXERCISE 2

Write Jest tests for the stack with fine-grained assertions and one snapshot. Include a least-privilege test, let it fail, fix the stack, watch the snapshot fail, and update it. Add an Aspect that rejects unversioned buckets and test both the failing and the passing case. Then run the CDK guide's own testing example as printed and report what happens.

View solution

EXERCISE 3

Explore the edges: run `cdk diff`, `deploy`, `bootstrap` and `destroy` with no credentials, and a context lookup; show what an aspect error leaves in the output folder; use raw overrides and show they bypass the aspect; print the logical IDs for the same bucket in four places in the tree; list the unset feature flags and measure what one of them changes.

View solution

CHAPTER 5 QUICK REFERENCE

- App → stacks → constructs; L1 (`Cfn*`, one CloudFormation resource), L2 (curated defaults and helpers), L3 (patterns). Every resource lives in a `Stack` inside an `App`; the `scope` and `id` build the construct tree
- `cdk synth` runs your program and writes a cloud assembly (`cdk.out`) with one CloudFormation template per stack; deployment is still CloudFormation. It needs no account unless something does a context lookup
- Tables, conditions and parameter checks are ordinary code: the generated template has none of them (apart from the CLI's own `CDKMetadata` condition and the `BootstrapVersion` parameter and rule)
- Logical IDs come from the construct path plus a hash: moving a construct, renaming it or changing its ID makes CloudFormation see a remove and an add. Same code gives the same IDs
- Grants are convenient and broad: `grantReadWrite` included `s3:DeleteObject*`; test least privilege on the generated template. Defaults you did not set show up (`Delete` on queues)
- Tests: `Template.fromStack`, `hasResourceProperties` (partial match), `resourceCountIs`, `Match`; snapshots catch every change, including intended ones and CDK upgrades, so use them sparingly. No account needed
- Aspects visit every construct; an error annotation makes synth exit with status 1 but still writes the assembly: check the exit status. Raw overrides (`addPropertyOverride`) bypass both the typed API and aspects
- Bootstrapping (`CDKToolkit`: bucket, ECR repository, IAM roles) is per environment and needed before deploying. `bootstrap`, `deploy`, `diff`, `destroy` and lookups need credentials and failed with exit status 1 without them
- Feature flags change generated templates: set them deliberately in `cdk.json` (`cdk flags` lists them; 84 were unset in a hand-made project); the CLI prints a warning when they are unconfigured
- Documentation examples age: the guide's own testing example fails as printed (Node 14 against Node 18) and uses a deprecated property

COMING NEXT

Chapter 6 is containers on AWS: ECR repositories and image scanning, ECS clusters, task definitions and services, Fargate versus EC2 capacity, task roles versus execution roles, logs, and a task definition written and synthesised with the CDK. Docker is installed on this machine but its engine was not running while I wrote this chapter, so chapter 6 starts by starting it. Start with *AWS Intermediate/Advanced 6*.

AWS Intermediate/Advanced

Chapter 6 · Containers on AWS: ECR, ECS and Fargate

You already know how to build an image (*Docker for Beginners*, *Docker Intermediate/Advanced*) and how a load balancer and health checks fit together (*AWS Fundamentals* 8). This chapter is about what changes when the container runs on AWS: **Amazon ECR** stores the image, **Amazon ECS** decides what runs, and **AWS Fargate** supplies the machines so that you do not manage servers. Everything ECS does to a container is something Docker can already do, so most of this chapter runs the container *locally* and shows the Docker setting that each ECS setting becomes.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS credentials on the machine that wrote this course, so nothing was pushed to ECR or deployed to ECS.

- **Run here (level 1):** Docker 28.0.4 (the engine was started for this chapter) built and ran the images; Node 24.18.0 with `aws-cdk` 2.1145.0 and `aws-cdk-lib` 2.273.0 synthesised the stacks (reusing the install from chapter 5); cfn-lint 1.57.2, Jest 30.5.2 and `botocore` 1.43.112 (offline, with a stubbed client) checked them. Two small Python models encode rules from the ECS documentation and were cross-checked against cfn-lint.
- **Needs an AWS account (level 2), not run:** pushing an image, creating the cluster, service and load balancer, scanning, logging to CloudWatch, and everything about how ECS and Fargate really behave. What ECS does is described from the documentation pages I read on 2026-10-13 (through a page-fetch tool that summarises pages), and each such statement says so.
- **Not an ECS test:** a local Docker container is *not* a Fargate task. Where the two could differ (limits, signals, health), the chapter says what was observed in Docker and what the ECS documentation says, and never mixes them.

What Runs Where

The ECS pages I read list four places a task can run: **Fargate**, **Amazon EC2** instances you manage, **Amazon ECS Managed Instances** and **external instances**. This chapter uses Fargate. EKS (Kubernetes) is chapter 7; running a container image as a Lambda function was covered in *AWS Fundamentals* 7 and not revisited. I did not read the pages that compare these options, so I make no claim about which one to pick beyond what the tests below show.

ECS WORD	WHAT IT IS	CLOSEST DOCKER IDEA
Task definition	A versioned recipe (a *family* plus a revision number) with container settings, CPU and memory, roles and network mode	The flags of `docker run`, written down and versioned
Task	One running copy of a task definition. The documentation gives its life cycle: PROVISIONING, PENDING, ACTIVATING, RUNNING, DEACTIVATING, STOPPING, DEPROVISIONING, STOPPED	One `docker run`
Service	Keeps a desired number of tasks running and replaces unhealthy ones; carries the rolling-update settings	Nothing in plain Docker (closest is a restart policy plus a script)
Cluster	The logical group the tasks and services belong to	The Docker host(s)
ECR repository	Where the image is stored, with scanning and lifecycle rules	Docker Hub, but private to your account

The Image: Two Dockerfiles, One Mistake

Run here (level 1). A tiny web service (`server.py`, in the exercise solution) answers on port 8080, has a `/health` endpoint that can be told to fail, can try to write a file, can allocate memory on request, and can optionally exit cleanly on `SIGTERM`. It is built twice: a *good* image (non-root user, exec-form `CMD`, a `HEALTHCHECK`) and a *naive* one (root, shell-form `CMD`).

```
# the "good" image: non-root user, exec-form CMD, health check
FROM python:3.13-slim
RUN useradd --uid 10001 --no-create-home app
WORKDIR /app
COPY server.py .
USER 10001
EXPOSE 8080
HEALTHCHECK --interval=2s --timeout=2s --retries=2 --start-period=1s \
  CMD ["python", "-c", "import urllib.request,sys; sys.exit(0 if urllib.request.urlopen(
CMD ["python", "server.py"]
```

```
# the "naive" image: runs as root, shell-form CMD
FROM python:3.13-slim
WORKDIR /app
```

```
COPY server.py .
EXPOSE 8080
CMD python server.py
```

```
bash exp_local.sh   # first block
```

```
aws2-6/naive:latest  176MB
aws2-6/good:latest  176MB
good image runs as uid 10001
naive image runs as uid 0
good: User=10001 Cmd=["python","server.py"] Entrypoint=null
naive: User=[] Cmd=["/bin/sh","-c","python server.py"]
```

Both images are 176 MB. The difference is in how they behave, and everything below depends on it. Notice the naive image's command: `["/bin/sh","-c","python server.py"]`. The shell, not Python, is the container's first process.

A Task Definition Is docker run in JSON

The ECS parameters I read map onto Docker settings. The documentation states some of the mappings itself (`user` maps to `--user`; `readonlyRootFilesystem` maps to `--read-only`); the others are the ones I used in the experiments.

TASK DEFINITION	DOCKER RUN	WHAT THE DOCUMENTATION SAYS
`user`	`--user`	"As a security best practice, always use a non-root user"
`readonlyRootFilesystem`	`--read-only`	Default `false`; not supported for Windows containers
`linuxParameters.initProcessEnabled`	`--init`	(the parameter page excerpt I kept did not cover it)
`stopTimeout`	`--stop-timeout`	Seconds before the container is forcefully killed; default 30, maximum 120
`healthCheck`	`HEALTHCHECK`	Interval 5 to 300 s (default 30), timeout 2 to 60 (5), retries 1 to 10 (3), startPeriod 0 to 300 (disabled)

TASK DEFINITION	DOCKER RUN	WHAT THE DOCUMENTATION SAYS
task `cpu`, `memory`	`--cpus`, `--memory`	Only certain combinations are valid on Fargate (below)

To check the mapping I applied the same settings as flags (non-root, read-only, init, 512 MiB, 0.25 CPU, stop timeout 20 s) and asked Docker what it recorded:

```
docker run -d --name task1 --user 10001 --read-only --init --memory 512m --cpus 0.25 --s
```

```
uid/pid: hello from pid 7 as uid 10001
write:   write failed: [Errno 30] Read-only file system: '/app/data.txt'
health:  healthy
limits: memory=536870912 bytes, nanoCpus=250000000, readonly=true, init=true, user=10001
```

That is evidence about *Docker*: the flags take effect and the service works under them. It is not evidence that ECS applies them the same way.

Stopping a Task: SIGTERM, stopTimeout and Exit Codes

From the documentation (level 3): in the STOPPING state, for Linux containers, "the container agent will send the stop signal defined in your container image ... This is `SIGTERM` by default. Then it will send a `SIGKILL` after waiting the `StopTimeout` duration set in the task definition." **Run here:** I stopped containers with `docker stop -t 5` (the same idea: signal, wait, kill) and recorded how long it took and the exit code. The "took" figure includes about two seconds of Docker overhead that every row has; the rows that waited the full timeout are about five seconds longer.

```
bash exp_local.sh   # second block
```

```
good image, no SIGTERM handler                  stop took   7.1s  exit code 137   proce
good image, handler (HANDLE_SIGTERM=1)          stop took   2.1s  exit code 0   process
good image, no handler, --init                  stop took   2.3s  exit code 143   proce
naive image (shell-form CMD), no handler        stop took   7.3s  exit code 137   proce
naive image, handler                            stop took   7.3s  exit code 137   proce
```

Read the rows carefully.

- **No handler, python as the first process (137, waited the full timeout).** The program was killed, not asked. My understanding is that Linux gives the first process of a container no default action for `SIGTERM`, so unless the program installs a handler the signal does nothing; the rows are consistent with that, but I did not test it separately.
- **With a handler (exit code 0, no wait).** The program saw the signal and left cleanly: this is what you want in a service, because it can finish requests first.
- **With `--init` (exit code 143, no wait).** A tiny init process (`docker-init`) became the first process and the program terminated on the signal (143 = 128 + 15). The ECS equivalent is `initProcessEnabled` (not run on ECS).
- **The shell-form `CMD` was the trap.** With `/bin/sh -c` as the first process, *even the image with a handler* waited the full timeout and was killed (137): the shell did not pass the signal on. This is the cheapest way to turn a clean shutdown into a forced one, and it only shows up when something stops the container.

Why it matters on ECS: every deployment, scale-in and spot-style interruption stops tasks. If the app ignores the signal, each stop costs the full `stopTimeout` (default 30 s, maximum 120 s on the page I read) and ends in a kill, with in-flight requests dropped. Not run: a real ECS stop.

Memory Limits

Run here: a container with `--memory 64m` (no swap) was asked to hold 20 MiB, which worked, and then 200 MiB:

```
bash exp_local.sh   # third block
```

```
holding 20 MiB   <- 20 MiB request
   <- 200 MiB request (curl exit 52)
State.Status=exited ExitCode=137 OOMKilled=true
```

The request never got an answer (curl exit status 52, empty reply), the container exited with 137 and Docker recorded `OOMKilled=true`. On Docker, then, the memory figure is a hard wall. I read that Fargate's task `memory` must come from a table (below), but I did *not* read what Fargate does when a container exceeds it, so treat the local result as an illustration of the idea and look up the ECS behaviour before relying on it.

Health Checks

The documentation says the check command is run in the container, `CMD` or `CMD-SHELL`, and "an exit code of 0, with no `stderr` output, indicates success". I did not test the `stderr` condition. **Run here**, with a Docker `HEALTHCHECK` every 2 s (2 retries):

```
bash exp_local.sh   # fourth block
```

```
right after start: starting
after 5 s:         healthy
now unhealthy
after /break + 7s: unhealthy  failing streak=4
container is still: running   (Docker alone does not restart an unhealthy container)
last health check exit codes: 1 1 1 1 1
```

Docker marked the container unhealthy after the service began failing, and **left it running**. On its own, a failing health check restarts nothing; something has to act on it. From the ECS deployment page: during a rolling deployment "when tasks become unhealthy, Amazon ECS replaces them to maintain your service's `minimumHealthyPercent`". So in ECS the service is the thing that acts. Two other health checks exist on a load-balanced service (the target group's check, from *AWS Fundamentals* 8); I did not read how ECS combines the container check with the target group's, so that is not claimed here.

Non-Root and Read-Only

Run here: the same `/write` request against four settings.

```
bash exp_local.sh   # fifth block
```

```
naive (root), writable fs      -> wrote /app/data.txt
naive (root), --read-only      -> write failed: [Errno 30] Read-only file system: '/app/
good (uid 10001), writable fs  -> write failed: [Errno 13] Permission denied: '/app/data
good (uid 10001), --read-only  -> write failed: [Errno 30] Read-only file system: '/app/
```

Root with a writable file system can write; root with `--read-only` cannot; the non-root user cannot write to `/app` even when it is writable (permission denied) and cannot when it is read-

only either. Both settings are cheap to apply and each blocks a different thing. If your app needs scratch space, give it a place to write on purpose (a volume, or a temporary directory) rather than loosening the root file system; the volume options of ECS were not read.

CPU Architecture

The task definition's `cpuArchitecture` is `X86_64` (the default) or `ARM64`, and "all task definitions that are used in a service must have the same value". **Run here:** this machine is x86-64 and Docker ran the `linux/arm64` variant of `python:3.13-slim` through emulation:

```
docker run --platform linux/arm64 python:3.13-slim python -c "import platform;print(plat
```

```
x86_64
daemon arch: x86_64
ran as aarch64
exit status: 0
```

So you can test an ARM image on this machine, slowly. The reverse mistake is the one that bites: an image built on an ARM laptop only runs on an `ARM64` task definition. Not run: a mismatch on Fargate.

Task Size: The Fargate Table

The task-size page I read lists the valid Linux CPU and memory combinations for Fargate: 256 (0.25 vCPU) with 512 MiB, 1 GB or 2 GB; 512 with 1 to 4 GB; 1024 with 2 to 8 GB; 2048 with 4 to 16 GB; 4096 with 8 to 30 GB; and for 8192, 16384 and 32768 larger ranges that need platform version 1.4.0 or later. An invalid pair returns `Invalid 'cpu' setting for task` at registration.

I wrote the table as a function (`fargate_model.py`). The spot checks in that file only restate my own encoding of the table (the page gives no worked examples for it), so they prove little. What makes it a real check is a second, independent source: **cfn-lint** has its own rules for this (E3047 for an incompatible pair, E3048 for a CPU value that is not on the list). I generated one task definition for every pair of 10 CPU values (the 8 on the list plus 128 and 300) and every memory value in steps of 512 MiB up to 250 GB, 5,000 pairs, and compared:

```
python cross_check.py
```

```
combinations checked: 5000
model says valid: 77  cfn-lint accepts: 77
disagreements: 0
```

Two separate encodings agree on every pair: 77 valid combinations. That is evidence that my reading of the page and cfn-lint's schema match each other, not that either matches AWS today.

What the tools do with a bad size (CDK synth of the stack in the next section, with `-c cpu=… -c memory=…`):

```
cat out/sizes.txt
```

```
cpu=256 memory=512: cdk synth exit 0; cdk message: ; cfn-lint: nothing beyond W2001
cpu=256 memory=4096: cdk synth exit 0; cdk message: [Warning] Template validation found
cpu=512 memory=512: cdk synth exit 0; cdk message: [Warning] Template validation found i
cpu=300 memory=512: cdk synth exit 0; cdk message: [Warning] Template validation found i
cpu=1024 memory=8192: cdk synth exit 0; cdk message: ; cfn-lint: nothing beyond W2001
cpu=2048 memory=3072: cdk synth exit 0; cdk message: [Warning] Template validation found
```

`cdk synth` **exits 0 for every bad pair** and prints only a warning that template validation found issues. If your pipeline checks the exit status, a bad size gets through synthesis. cfn-lint reports the pair (and the invalid CPU value). Lint the template as a separate, failing step.

The Stack: ECR, Cluster, Task, Service

Run here: a CDK stack with no pattern construct, so that every resource is one you can see. It creates a small VPC with public subnets only and no NAT gateway, a cluster, an ECR repository, an S3 bucket the task reads, a task role, a Fargate task definition, a log group, a security group and a service.

```
import * as cdk from "aws-cdk-lib";
import { Construct } from "constructs";
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as ecr from "aws-cdk-lib/aws-ecr";
import * as ecs from "aws-cdk-lib/aws-ecs";
```

```
import * as iam from "aws-cdk-lib/aws-iam";
import * as logs from "aws-cdk-lib/aws-logs";
import * as s3 from "aws-cdk-lib/aws-s3";

export interface ServiceProps extends cdk.StackProps {
  cpu: number;
  memoryMiB: number;
  desiredCount: number;
}

// A deliberately explicit Fargate service: nothing is created for us that we cannot see
export class ServiceStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props: ServiceProps) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, "Vpc", { maxAzs: 2, natGateways: 0, subnetConfiguratio
    const cluster = new ecs.Cluster(this, "Cluster", { vpc });

    const repo = new ecr.Repository(this, "Repo", {
      imageScanOnPush: true,
      imageTagMutability: ecr.TagMutability.IMMUTABLE,
      removalPolicy: cdk.RemovalPolicy.DESTROY,
      emptyOnDelete: true,
      lifecycleRules: [{ maxImageCount: 10, description: "keep the last 10 images" }],
    });

    const dataBucket = new s3.Bucket(this, "Data", { blockPublicAccess: s3.BlockPublicAc

    const taskRole = new iam.Role(this, "TaskRole", { assumedBy: new iam.ServicePrincipa
    // explicit instead of dataBucket.grantRead(taskRole), which generates s3:GetObject*
    taskRole.addToPolicy(new iam.PolicyStatement({ actions: ["s3:GetObject"], resources:
    taskRole.addToPolicy(new iam.PolicyStatement({ actions: ["s3:ListBucket"], resources

    const task = new ecs.FargateTaskDefinition(this, "Task", {
      cpu: props.cpu,
      memoryLimitMiB: props.memoryMiB,
      taskRole,
      runtimePlatform: { cpuArchitecture: ecs.CpuArchitecture.X86_64, operatingSystemFam
    });

    const logGroup = new logs.LogGroup(this, "Logs", { retention: logs.RetentionDays.ONE

    task.addContainer("web", {
      image: ecs.ContainerImage.fromEcrRepository(repo, "1.0.0"),
      portMappings: [{ containerPort: 8080 }],
```

```
        readonlyRootFilesystem: true,
        user: "10001",
        stopTimeout: cdk.Duration.seconds(20),
        linuxParameters: new ecs.LinuxParameters(this, "Linux", { initProcessEnabled: true
        logging: ecs.LogDrivers.awsLogs({ streamPrefix: "web", logGroup }),
        healthCheck: {
          command: ["CMD-SHELL", "python -c \"import urllib.request;urllib.request.urlopen
          interval: cdk.Duration.seconds(15), timeout: cdk.Duration.seconds(5), retries: 3
        },
      });

      const sg = new ec2.SecurityGroup(this, "ServiceSg", { vpc, allowAllOutbound: true })
      sg.addIngressRule(ec2.Peer.ipv4(vpc.vpcCidrBlock), ec2.Port.tcp(8080), "from inside

      new ecs.FargateService(this, "Service", {
        cluster, taskDefinition: task, desiredCount: props.desiredCount, securityGroups: [
        assignPublicIp: true, minHealthyPercent: 100, maxHealthyPercent: 200,
        circuitBreaker: { rollback: true },
      });
    }
  }
```

`cdk synth` succeeded and the template has 24 resources of 18 types. What it created that I did not write is worth reading: a **task execution role** with its own policy, and the image reference built from the repository.

```
    Task79114B6B:
      Type: AWS::ECS::TaskDefinition
      Properties:
        ContainerDefinitions:
          - Essential: true
            HealthCheck:
              Command:
                - CMD-SHELL
                - python -c "import urllib.request;urllib.request.urlopen('http://127.0.0.
              Interval: 15
              Retries: 3
              StartPeriod: 10
              Timeout: 5
            Image:
              Fn::Join:
                - ""
```

```
            - - Fn::Select:
                  - 4
                  - Fn::Split:
                      - ":"
                      - Fn::GetAtt:
                          - Repo02AC86CF
                          - Arn
              - .dkr.ecr.
              - Fn::Select:
                  - 3
                  - Fn::Split:
                      - ":"
                      - Fn::GetAtt:
                          - Repo02AC86CF
                          - Arn
              - "."
              - Ref: AWS::URLSuffix
              - /
              - Ref: Repo02AC86CF
              - :1.0.0
          LinuxParameters:
            Capabilities: {}
            InitProcessEnabled: true
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group:
                Ref: Logs6819BB44
              awslogs-stream-prefix: web
              awslogs-region:
                Ref: AWS::Region
          Name: web
          PortMappings:
            - ContainerPort: 8080
              Protocol: tcp
          ReadonlyRootFilesystem: true
          StopTimeout: 20
          User: "10001"
      Cpu: "256"
      ExecutionRoleArn:
        Fn::GetAtt:
          - TaskExecutionRoleEF964934
          - Arn
      Family: ServiceTask4E7131ED
      Memory: "512"
```

```
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        CpuArchitecture: X86_64
        OperatingSystemFamily: LINUX
      TaskRoleArn:
        Fn::GetAtt:
          - TaskRole30FC0FBB
          - Arn
    Metadata:
      aws:cdk:path: Service/Task/Resource
```

```
  TaskExecutionRoleDefaultPolicy72339918:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - ecr:BatchCheckLayerAvailability
              - ecr:GetDownloadUrlForLayer
              - ecr:BatchGetImage
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - Repo02AC86CF
                - Arn
          - Action: ecr:GetAuthorizationToken
            Effect: Allow
            Resource: "*"
          - Action:
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - Logs6819BB44
                - Arn
        Version: "2012-10-17"
      PolicyName: TaskExecutionRoleDefaultPolicy72339918
      Roles:
        - Ref: TaskExecutionRoleEF964934
```

Lint:

```
cfn-lint cdk.out/Service.template.json   # and the stdout YAML
```

```
$ cfn-lint cdk.out/Service.template.json
(exit status 0)

$ cfn-lint out/synth_service.yaml   (the YAML that cdk synth prints on stdout)
W2001 Parameter BootstrapVersion not used.
out\synth_service.yaml:516:3

(exit status 4)

Resources identical in both files: True | sections only in cdk.out: ['Rules']
```

The template in `cdk.out` lints clean. The YAML that `cdk synth` prints on standard output has the same resources but not the `Rules` section, so cfn-lint reports "BootstrapVersion not used" for it: lint the file in `cdk.out`, not the printed text.

Two roles, two jobs

From the documentation: the task *execution* role "grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your behalf", and its permissions "aren't directly accessible by the containers in the task". Typical uses: pulling the image from a private ECR repository, sending logs with the `awslogs` driver, reading Secrets Manager or Parameter Store values that the task definition references. The *task* role is what your application code uses. The managed policy for the former is `AmazonECSTaskExecutionRolePolicy`.

Run here: the synthesised stack has two different roles. The execution role (created by CDK) got exactly: `ecr:BatchCheckLayerAvailability`, `ecr:GetDownloadUrlForLayer`, `ecr:BatchGetImage` on the one repository, `ecr:GetAuthorizationToken` on `*` (an action that does not take a resource), and `logs:CreateLogStream`, `logs:PutLogEvents` on the one log group. Nothing more. I did not compare it with the managed policy's contents.

The task role is where the chapter 5 lesson came back. My first version gave it `dataBucket.grantRead(taskRole)`, and the test "task role can only read the data bucket" failed:

```
cat out/jest_names_1.txt
```

```
passed 9 failed 1
PASSED - task role and execution role are different roles
PASSED - execution role can pull images and write logs, and nothing else
FAILED - task role can only read the data bucket
    Error: expect(received).not.toMatch(expected)
PASSED - container: non-root, read-only root fs, init process, stop timeout within the d
PASSED - health check is defined and the image is a pinned tag, not latest
PASSED - logs go to a log group with a retention period
PASSED - repository scans on push, tags are immutable, and a lifecycle rule exists
PASSED - rolling update settings leave room to start a task, and the circuit breaker rol
PASSED - no NAT gateway and no load balancer in this stack
PASSED - the service security group allows only 8080 from the VPC
```

The grant produced `s3:GetObject*`, `s3:GetBucket*` and `s3:List*`: three wildcards, more than "read the objects and list the bucket". I replaced it with two explicit statements (`s3:GetObject` on the objects, `s3:ListBucket` on the bucket) and the ten tests passed:

```
cat out/jest_names_2.txt
```

```
passed 10 failed 0
PASSED - task role and execution role are different roles
PASSED - execution role can pull images and write logs, and nothing else
PASSED - task role can only read the data bucket
PASSED - container: non-root, read-only root fs, init process, stop timeout within the d
PASSED - health check is defined and the image is a pinned tag, not latest
PASSED - logs go to a log group with a retention period
PASSED - repository scans on push, tags are immutable, and a lifecycle rule exists
PASSED - rolling update settings leave room to start a task, and the circuit breaker rol
PASSED - no NAT gateway and no load balancer in this stack
PASSED - the service security group allows only 8080 from the VPC
```

The tests check the properties this chapter has argued for: different roles, an exact execution-role action list, non-root, read-only root file system, init process, a stop timeout of 20 s (the test's title says "within the documented 120 s" but it only asserts the value 20), a health check, a pinned image tag, log retention, scan on push and immutable tags, rolling-update settings, no NAT gateway or load balancer, and a security group that only admits port 8080 from the VPC.

ECR

The repository in the stack sets scan on push, immutable tags, and a lifecycle rule that keeps the last ten images. **From the documentation (ECR scanning page):** there are two scanning types. *Basic* scanning uses the CVE database to scan operating-system packages and runs on push or manually. *Enhanced* scanning integrates with Amazon Inspector, covers operating-system and programming-language packages, and can run on push or continuously; Inspector emits an event to EventBridge as new vulnerabilities appear. "Switching between Enhanced scanning and Basic scanning will cause previously established scans to no longer be available." **Not read:** the price of enhanced scanning, what the lifecycle rule syntax allows beyond the rule I used, tag immutability behaviour, and anything about replication.

Pushing an image needs an account, so it is level 2 and was not run:

```
# level 2: needs an AWS account, credentials and Docker. NOT RUN. Written from memory of
# Account ID and Region are placeholders.
aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-s
docker tag aws2-6/good:latest 111122223333.dkr.ecr.eu-west-1.amazonaws.com/web:1.0.0
docker push 111122223333.dkr.ecr.eu-west-1.amazonaws.com/web:1.0.0
# the repository in the stack has immutable tags: pushing 1.0.0 a second time should be
```

Those commands are written from my knowledge of the AWS CLI and the Docker CLI; I did not read the ECR push documentation in this session, so check them against the current page before using them.

Rolling Deployments

From the documentation: `minimumHealthyPercent` is a lower limit on running healthy tasks, "rounded up"; `maximumPercent` is an upper limit on tasks, "rounded down". The page's four examples became a test of a small function:

```
py fargate_model.py
```

```
ok   50% min, 4 desired: can stop two before starting two -> may_stop=2 (documented 2)
ok   75% min, 2 desired: can't stop any (result is two) -> may_stop=0 (documented 0)
```

```
  ok   200% max, 4 desired: can start four -> may_start=4 (documented 4)
  ok   125% max, 3 desired: can't start any (result is three) -> may_start=0 (documented 0
```

The same page warns: "you should ensure that the scheduler can stop or start at least one task when a deployment is initiated." Settings that allow neither leave a deployment stuck, and ECS sends a service event. The stack uses 100 / 200 with two tasks, so it may start two new tasks before stopping any old ones: with 100 / 100 the function says it may neither stop nor start any, which is exactly the stuck case:

```
py -c "from fargate_model import rolling_limits as r; print(r(2,100,100)); print(r(2,100
```

```
desired 2, min 100, max 100: {'may_stop': 0, 'may_start': 0, 'min_running': 2, 'max_runn
desired 2, min 100, max 200: {'may_stop': 0, 'may_start': 2, 'min_running': 2, 'max_runn
```

(the first line is the stuck case).

The stack also turns on the **deployment circuit breaker with rollback**. The page describes two ways to detect a failed rolling deployment: the circuit breaker ("use this method when you want to stop a deployment when the tasks can't start") and CloudWatch alarms ("when you want to stop a deployment based on application metrics"), and "both methods support rolling back to the previous service revision". I did not read the circuit breaker page itself, so its thresholds are not stated here.

Image digests. By default ECS resolves the image tag in the task definition to a digest when a deployment starts and uses that digest for every task, so all tasks run identical images even if the tag moves; that is a documented behaviour (with agent and platform-version conditions), not tested here. Immutable tags and pinned versions (as the stack uses) avoid relying on it.

The Pattern Construct and Its Bill

The CDK has an L3 construct, `ApplicationLoadBalancedFargateService` (chapter 5 named it). **Run here:** I synthesised it with an image and a port and nothing else, then counted what it created:

```
cat out/pattern_resources.txt
```

```
resources: 38
  1  AWS::CDK::Metadata
  2  AWS::EC2::EIP
  1  AWS::EC2::InternetGateway
  2  AWS::EC2::NatGateway
  4  AWS::EC2::Route
  4  AWS::EC2::RouteTable
  2  AWS::EC2::SecurityGroup
  1  AWS::EC2::SecurityGroupEgress
  1  AWS::EC2::SecurityGroupIngress
  4  AWS::EC2::Subnet
  4  AWS::EC2::SubnetRouteTableAssociation
  1  AWS::EC2::VPC
  1  AWS::EC2::VPCGatewayAttachment
  1  AWS::ECS::Cluster
  1  AWS::ECS::Service
  1  AWS::ECS::TaskDefinition
  1  AWS::ElasticLoadBalancingV2::Listener
  1  AWS::ElasticLoadBalancingV2::LoadBalancer
  1  AWS::ElasticLoadBalancingV2::TargetGroup
  1  AWS::IAM::Policy
  2  AWS::IAM::Role
  1  AWS::Logs::LogGroup
NAT gateways: 2 | elastic IPs: 2
task size: cpu 256 memory 512 | read-only root fs: None | user: None
service: desired None | deployment config: {"Alarms": {"AlarmNames": [], "Enable": false
target group health check: {} | deregistration delay attrs: [{'Key': 'stickiness.enabled
listener port 80 HTTP
```

A few lines of code gave **38 resources**, including **two NAT gateways and two Elastic IPs** (the default VPC the pattern builds has private subnets), an internet-facing load balancer with an **HTTP-only listener on port 80**, a task with no read-only root file system and no non-root user, and a service with `MinimumHealthyPercent` 50. The target group health check was not set in the template, so whatever the service default is applies (I did not read it). cfn-lint reported one warning (W3005, a redundant dependency, exit status 4) and no errors.

The cost of those NAT gateways is the part people miss. Arithmetic from prices I read on 2026-10-13 (the page-fetch tool summarised the pages and converted the Fargate figures; the NAT price listed was for US East (Ohio) only, so the two Regions are mixed here, which is a flaw of this sketch):

```
py cost_sketch.py
```

```
Fargate, 2 tasks of 0.25 vCPU + 0.5 GB, one month, X86 : $18.02
Fargate, 2 tasks of 0.25 vCPU + 0.5 GB, one month, ARM : $14.42
Fargate, 2 tasks of 1 vCPU + 2 GB, one month, X86      : $72.08
2 NAT gateways, one month, hourly charge only (Ohio rate): $65.70
2 public IPv4 addresses on those NAT gateways, one month: $7.30
so the two NAT gateways plus their addresses cost about 4x the two small tasks (hourly c
the default L3 pattern keeps them running whether or not any traffic flows
```

So the idle plumbing of the default pattern costs more per month than two small tasks, before the load balancer (not priced here: I did not read its price). That is the argument for either choosing the VPC deliberately (public subnets with a tight security group, or VPC endpoints) or reading the pattern's output before deploying it.

Validating a Task Definition Without AWS

If you register task definitions with the CLI or SDK rather than the CDK, you can check the JSON offline. `botocore` validates the request's shape before it sends anything; I used a stubbed client so that nothing is sent, then ran the file through the documented ranges as well.

```
{
  "family": "web",
  "requiresCompatibilities": ["FARGATE"],
  "networkMode": "awsvpc",
  "cpu": "256",
  "memory": "512",
  "runtimePlatform": {"cpuArchitecture": "X86_64", "operatingSystemFamily": "LINUX"},
  "taskRoleArn": "arn:aws:iam::111122223333:role/web-task",
  "executionRoleArn": "arn:aws:iam::111122223333:role/web-exec",
  "containerDefinitions": [{
    "name": "web",
    "image": "111122223333.dkr.ecr.eu-west-1.amazonaws.com/web:1.0.0",
    "essential": true,
    "user": "10001",
    "readonlyRootFilesystem": true,
    "stopTimeout": 20,
    "linuxParameters": {"initProcessEnabled": true},
    "portMappings": [{"containerPort": 8080, "protocol": "tcp"}],
```

```
    "healthCheck": {"command": ["CMD-SHELL", "python -c \"import urllib.request;urllib.r
    "logConfiguration": {"logDriver": "awslogs", "options": {"awslogs-group": "/ecs/web"
  }]
}
```

```
python taskdef_check.py
```

```
case                                  | botocore (offline)
the file as written                   | accepted
cpu as a number instead of a string   | rejected: Invalid type for parameter cpu, val
size 256 / 4096                       | accepted
stopTimeout 300                       | accepted
health check interval 1               | accepted
health check without command          | rejected: Missing required parameter in conta
containerPort as a string             | rejected: Invalid type for parameter containe
no family                             | rejected: Missing required parameter in input
misspelt key readOnlyRootFilesystem   | rejected: Unknown parameter in containerDefin
```

`botocore` catches types, missing required fields and misspelt keys, and **accepts** the invalid size, those three but does not look at types or required fields. The third layer, cfn-lint on the CDK template, caught the size but **not** a `StopTimeout` of 300 or a health-check interval of 1 (I tried both, plus retries 20 and timeout 90: exit status 0 each time). No one tool covers it; and a real `RegisterTaskDefinition` call is the only check against the live rules (not run).

Cost and Clean-Up

Nothing in this chapter created anything in AWS. What is left on this machine is outside the site: the scratch folder, two local Docker images (`aws2-6/good` and `aws2-6/naive`) and a pulled `arm64` variant of `python:3.13-slim`. If you deploy the stack (level 2, not run): the Fargate tasks, the log group and the ECR images are billed while they exist; `cdk destroy` removes the stack (the repository is set to be emptied and deleted and the log group is removed; the data bucket has no removal policy set, so the CDK default applies and I expect it to be kept, which I did not test), and the CDK bootstrap resources remain. Create a budget alert first (*AWS Fundamentals* 9).

WHAT WAS AND WAS NOT CHECKED

Run here: two images built and run; stop behaviour under five configurations; the memory limit; the Docker health check; non-root and read-only combinations; an ARM image under emulation; the task settings as Docker flags; the CDK stack synthesised, linted and tested (10 tests, one deliberate failure found by the first version); the Fargate size table cross-checked against cfn-lint on 5,000 pairs; bad sizes through `cdk synth`; out-of-range stop timeout and health check values through cfn-lint; the L3 pattern synthesised and counted; `botocore` offline validation of a task definition. **From the documentation, not run:** the task size table, the execution role description, stop behaviour, health check parameters, the rolling-update percentages, the circuit breaker and alarm methods, image digest resolution, ECR scanning types, the Fargate and public IPv4 prices and the Ohio NAT price. **Not read or not run:** pushing to ECR, ECS itself, the awslogs driver, Container Insights, ECS Exec, secrets injection, service connect and discovery, auto scaling, blue/green deployments, Fargate Spot, EC2 and Managed Instances capacity, ECR replication and enhanced-scanning prices, the load balancer price and target group defaults, what Fargate does at a memory limit, and all behaviour of a real task. **Not checked at all:** a real deployment.

Practice

EXERCISE 1

Build a small web service as a good and a naive image. Measure stop time and exit code for five combinations (handler or not, exec or shell form, with and without `--init`), the memory limit, the Docker health check, and the four non-root / read-only combinations. Then apply the task-definition settings as `docker run` flags and ask Docker to confirm them.

View solution

EXERCISE 2

Write a Fargate service in the CDK without a pattern construct. Synthesise and lint it, write tests (including least privilege for the task role), let one fail, fix the stack, and rerun. Encode the Fargate size table, cross-check it against cfn-lint over every combination, and show what `cdk synth` does with an invalid size.

View solution

EXERCISE 3

Synthesise the load-balanced Fargate pattern with minimal arguments and count what it creates (NAT gateways, load balancer, listener protocol, security settings). Estimate the idle monthly cost from published prices and state what you did not price. Validate a hand-written task definition JSON with `botocore` offline and compare its verdicts with the documented rules and cfn-lint. Test the rolling-update percentages against the documentation's examples.

View solution

CHAPTER 6 QUICK REFERENCE

- ECR stores images; ECS runs them; Fargate supplies the machines. A task definition is versioned `docker run` settings; a task is one copy; a service keeps the desired count and replaces unhealthy tasks
- Use exec-form `CMD` (or an init process) so `SIGTERM` reaches your program, and handle it. Otherwise every stop waits for `stopTimeout` (default 30 s, maximum 120 s) and ends in a kill (exit 137)
- A failing health check restarts nothing in plain Docker; in ECS the service replaces unhealthy tasks. Health check ranges: interval 5 to 300 s, timeout 2 to 60, retries 1 to 10, startPeriod 0 to 300
- Run as non-root and with a read-only root file system; give the app a deliberate place to write if it needs one
- Fargate task sizes must come from the table (256 CPU: 512 MiB, 1 or 2 GB; 512: 1 to 4 GB; 1024: 2 to 8 GB; and so on). `cdk synth` exits 0 on a bad pair, cfn-lint flags it (E3047, E3048); neither flags an out-of-range stop timeout or health interval
- Execution role: the agents pull the image, write logs, fetch referenced secrets. Task role: your code. Keep them separate and test the task role's actions on the synthesised template (a `grantRead` gave three wildcard actions)
- Rolling updates: `minimumHealthyPercent` rounds up, `maximumPercent` rounds down; make sure at least one task can start or stop. Add the circuit breaker with rollback
- The L3 pattern with defaults created 38 resources, two NAT gateways and an HTTP-only load balancer. Read the synthesised template before deploying a pattern
- Lint the template in `cdk.out`; the YAML printed by `cdk synth` lacks the `Rules` section

COMING NEXT

Chapter 7 is Kubernetes on AWS: Amazon EKS. An EKS cluster is not free, so most of it will be manifests and definitions written and validated locally, and the chapter will point to *Kubernetes*

Fundamentals and *Kubernetes Intermediate/Advanced* for everything that is not AWS-specific. Start with *AWS Intermediate/Advanced 7*.

AWS Intermediate/Advanced

Chapter 7 · Kubernetes on AWS: EKS

AWS Intermediate/Advanced 6 ran containers on ECS and Fargate. Amazon EKS (Elastic Kubernetes Service) runs *Kubernetes* instead: AWS operates the Kubernetes control plane, and you decide how the worker nodes are provided. This chapter covers only what is AWS-specific. How Kubernetes itself works (pods, deployments, services, probes, disruption budgets) is in *Kubernetes Fundamentals* and *Kubernetes Intermediate/Advanced*, and is assumed.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS credentials on the machine that wrote this course, and an EKS cluster costs money while it exists, so this chapter is mostly level 3 and no cluster was created.

- **Run here (level 1):** `eksctl` 0.231.0 (its configuration schema and its failure without credentials), the CDK (`aws-cdk` 2.1145.0, `aws-cdk-lib` 2.273.0, reusing the chapter 5 install) synthesising an L1-only and an L2 EKS stack, cfn-lint 1.57.2, Jest 30.5.2, and four small Python models: pod limits per node, version support, trust-policy conditions, and manifest checks. The first two models were cross-checked against published data.
- **Planned but not completed:** I created a local Kubernetes cluster with `kind` 0.31.0 (Kubernetes v1.35.0, first one node, then three), to run the manifests against a real API server. Docker Desktop's engine stopped before the experiments could run and would not come back (its log said "no connected VM"; I tried `docker desktop start` and `restart`), so **nothing from that cluster is reported**. Where the chapter says what a real API server would do, it is my expectation and is marked as not tested.
- **From the documentation, not run (level 3):** what EKS, Auto Mode, Pod Identity, IRSA, the VPC CNI and the Load Balancer Controller do. Read on 2026-10-13 through a page-fetch tool that summarises pages; each such statement says so.

Two tools were downloaded as binaries into the scratch folder, not installed: `eksctl` and `kind`. I checked each file's SHA-256 against the checksum published next to it (both matched), which only shows the download was not corrupted, because the checksum comes from the same place as the file.

EKS or ECS?

I did not read a page that compares the two, so this table sticks to what the chapter shows or the pages I read state. The last row is my judgement and is not tested.

TOPIC	ECS ON FARGATE (CHAPTER 6)	EKS
What you describe	A task definition and a service	Kubernetes objects (Deployment, Service, Ingress, PodDisruptionBudget…), so the tools and manifests you already know
Control plane fee	Not read; no claim made	Per cluster per hour: $0.10 in standard version support, $0.60 in extended support (pricing page, 2026-10-13, no Region stated). About $73 or $438 for a 730-hour month
Pod addressing	Not covered here	Each pod gets its own VPC IP address with the Amazon VPC CNI, so subnet size and instance type limit pod counts (below)
Version lifecycle	Not applicable	A Kubernetes minor version gets 14 months of standard support plus 12 of extended; nodes are not upgraded for you (except in Auto Mode)
Identity for pods	Task role (chapter 6)	IRSA or EKS Pod Identity (below)
When EKS earns its cost (judgement)	When you already run Kubernetes, need its ecosystem or portability, or have many teams sharing a platform. For one small service, the extra concepts and the fixed fee are the price. Not tested.	

What EKS Runs, and What You Choose

From the documentation (level 3): AWS manages the Kubernetes API server and the etcd database. You choose how pods get compute. Two choices appear on the pages I read: **managed node groups** (EC2 instances in your account, created and updated through the EKS API) and **EKS Auto Mode**. Pods can also run on Fargate, with restrictions: the Pod Identity page says Linux and Windows pods on Fargate are not supported for Pod Identity. I did not read the Fargate profile pages, and hybrid nodes were only seen in the pricing page.

The Auto Mode page says AWS "takes over the configuration, management, security, and scaling of the EC2 instances launched as well as cluster capabilities for load balancing, IP address management, networking policy, and block storage". Specific statements: nodes use Bottlerocket variants with a read-only root file system and no SSH or SSM access; nodes have a **maximum lifetime of 21 days** (which you can reduce) and are then replaced; scaling is Karpenter-based; upgrades respect your Pod Disruption Budgets; "you do not have to install the EKS Pod Identity

Agent on EKS Auto Mode clusters". It limits pods per node to "the lower of" 110 and `ENIs × (IPs per ENI − 1)`. The pricing page says you pay "based on the duration and type of Amazon EC2 instances launched and managed by EKS Auto Mode" and lists per-instance Auto Mode fees only in an example (US West (Oregon)); I did not work out a total.

Managed node groups are not upgraded when the control plane is. The version page says so for managed node groups, self-managed nodes, Fargate pods and hybrid nodes alike: "These instances aren't automatically upgraded when you or Amazon EKS update your control plane." The page adds that Kubernetes tests compatibility between control plane and nodes for up to three minor versions, and recommends the same version for both.

Defining a Cluster

eksctl: one YAML file, checked against its own schema

`eksctl` takes a `ClusterConfig` file. **Run here:** `eksctl create cluster -f cluster.yaml --dry-run` needs AWS access even for a dry run: it exited with status 1 and "*checking AWS STS access – cannot get role ARN for current session*". What does work offline is `eksctl utils schema`, which prints a JSON schema of the configuration. I validated the file against it with the `jsonschema` package, then changed one thing at a time.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: demo
  region: eu-west-1
  version: "1.35"
availabilityZones: ["eu-west-1a", "eu-west-1b"]
iam:
  withOIDC: true
managedNodeGroups:
  - name: ng-1
    instanceType: t3.medium
    desiredCapacity: 2
    minSize: 1
    maxSize: 3
    privateNetworking: true
```

```
python validate_eksctl.py
```

```
yaml.safe_load('1.30') -> 1.3 | yaml.safe_load('"1.30"') -> '1.30'
change                                                    schema validation
the file as written                                       valid
version written as a YAML number, 1.35                    1 error(s): 1.35 is not of type
version 1.30 written unquoted (YAML reads it as 1.3)      1 error(s): 1.3 is not of type
misspelt key managedNodeGroup                             1 error(s): Additional properti
desiredCapacity 'two'                                     1 error(s): 'two' is not of typ
an unknown field on the node group                        1 error(s): Additional properti
minSize 5 above maxSize 3                                 valid
instanceType 't3.huge'                                    valid
region 'eu-west-9'                                        valid
withOIDC 'yes'                                            1 error(s): 'yes' is not of typ
kind 'ClusterConfigs'                                     1 error(s): 'ClusterConfigs' is
```

The schema caught 7 of the 10 changes: typos, wrong types and unknown fields, including the **YAML number trap**: an unquoted `version: 1.35` is a number, and the unquoted `1.30` becomes `1.3`, a different version. Quote versions. It did **not** catch a `minSize` above `maxSize`, an instance type that does not exist, or a Region that does not exist: those need AWS (or a list of valid values you maintain).

The CDK, two ways

Run here: the same cluster as an **L1-only** stack (every property is one you write) and with the L2 `eks.Cluster` construct. Both synthesised offline. The L1 stack:

```
import * as cdk from "aws-cdk-lib";
import { Construct } from "constructs";
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as eks from "aws-cdk-lib/aws-eks";
import * as iam from "aws-cdk-lib/aws-iam";

// Only L1 constructs for the EKS parts: every property is one you wrote, so the templat
export class EksL1Stack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);
    const vpc = new ec2.Vpc(this, "Vpc", { maxAzs: 2, natGateways: 0, subnetConfiguratio

    const clusterRole = new iam.Role(this, "ClusterRole", {
      assumedBy: new iam.ServicePrincipal("eks.amazonaws.com"),
```

```
    managedPolicies: [iam.ManagedPolicy.fromAwsManagedPolicyName("AmazonEKSClusterPoli
});
const nodeRole = new iam.Role(this, "NodeRole", {
  assumedBy: new iam.ServicePrincipal("ec2.amazonaws.com"),
  managedPolicies: [
    iam.ManagedPolicy.fromAwsManagedPolicyName("AmazonEKSWorkerNodePolicy"),
    iam.ManagedPolicy.fromAwsManagedPolicyName("AmazonEC2ContainerRegistryReadOnly"),
    iam.ManagedPolicy.fromAwsManagedPolicyName("AmazonEKS_CNI_Policy"),
  ],
});

const cluster = new eks.CfnCluster(this, "Cluster", {
  name: "demo",
  version: "1.35",
  roleArn: clusterRole.roleArn,
  resourcesVpcConfig: {
    subnetIds: vpc.publicSubnets.map((s) => s.subnetId),
    endpointPublicAccess: true,
    endpointPrivateAccess: true,
    publicAccessCidrs: ["203.0.113.0/24"],
  },
  accessConfig: { authenticationMode: "API" },
  upgradePolicy: { supportType: "STANDARD" },
  logging: { clusterLogging: { enabledTypes: [{ type: "api" }, { type: "audit" }, {
});

const nodes = new eks.CfnNodegroup(this, "Nodes", {
  clusterName: cluster.ref,
  nodeRole: nodeRole.roleArn,
  subnets: vpc.publicSubnets.map((s) => s.subnetId),
  instanceTypes: ["t3.medium"],
  amiType: "AL2023_x86_64_STANDARD",
  scalingConfig: { minSize: 1, maxSize: 3, desiredSize: 2 },
  updateConfig: { maxUnavailable: 1 },
});

for (const addon of ["vpc-cni", "kube-proxy", "coredns", "eks-pod-identity-agent"]) {
  const a = new eks.CfnAddon(this, "Addon-" + addon, { clusterName: cluster.ref, add
  a.addDependency(nodes);
}

const appRole = new iam.Role(this, "AppRole", { assumedBy: new iam.ServicePrincipal(
appRole.assumeRolePolicy!.addStatements(new iam.PolicyStatement({
  principals: [new iam.ServicePrincipal("pods.eks.amazonaws.com")], actions: ["sts:T
new eks.CfnPodIdentityAssociation(this, "Assoc", { clusterName: cluster.ref, namespa
```

```
  }
}
```

```
python analyze.py
```

```
== EksL1 (L1 only): 22 resources, 13 types
    1 AWS::CDK::Metadata
    1 AWS::EC2::InternetGateway
    2 AWS::EC2::Route
    2 AWS::EC2::RouteTable
    2 AWS::EC2::Subnet
    2 AWS::EC2::SubnetRouteTableAssociation
    1 AWS::EC2::VPC
    1 AWS::EC2::VPCGatewayAttachment
    4 AWS::EKS::Addon
    1 AWS::EKS::Cluster
    1 AWS::EKS::Nodegroup
    1 AWS::EKS::PodIdentityAssociation
    3 AWS::IAM::Role
== EksL2 (L2 Cluster): 43 resources, 21 types
   top-level types: CDK::Metadata x1, CloudFormation::Stack x2, EC2::EIP x1, EC2::Interne
== nested template EksL2awscdkawseksClusterResourceProviderC158A3E5.nested.temp: 18 reso
== nested template EksL2awscdkawseksKubectlProvider86078FD4.nested.template.jso: 6 resou

L2 stack: custom resources: ['AWS::CloudFormation::Stack', 'Custom::AWSCDK-EKS-Cluster',
L2 stack: NAT gateways: 1 | Lambda functions in nested stacks: 7
managed policies, L1 (my names): ['AmazonEC2ContainerRegistryReadOnly', 'AmazonEKSCluste
managed policies, L2 (library's): ['AmazonEC2ContainerRegistryPullOnly', 'AmazonEC2Conta
```

The L1 template has 22 resources; the L2 stack has 43, plus 24 more in two nested stacks (67 in all). The L2 construct does not use `AWS::EKS::Cluster` at all: it creates the cluster through a *custom resource* (`Custom::AWSCDK-EKS-Cluster`) backed by Lambda functions and a Step Functions state machine, plus a kubectl handler, an OIDC provider custom resource and a NAT gateway. A test in the exercise pins this behaviour. Reasons to prefer one or the other are yours to weigh; the facts are the size, the extra Lambda code that runs in your account during deployment, and that the cluster is not a plain CloudFormation resource. Two more things surfaced:

- **The L2 construct requires a `kubectlLayer`** (a Lambda layer containing `kubectl`). I passed a placeholder ARN so that the stack would synthesise; it could never deploy as written.
- **The library lags the service.** `aws-cdk-lib` 2.273.0 lists Kubernetes versions up to `V1_36`, while the lifecycle page lists 1.37 (released on EKS on 2026-10-01). The construct has a way to name another version, which I did not try.
- **Deprecation warning:** `CfnResource#addDependency` printed "is deprecated. Use `addResourceDependency` instead" at synth. I used the deprecated name in the L1 stack and left it, so the chapter shows the warning; synthesis still exited 0.

The managed policies are a useful cross-check. I wrote the L1 role policy names from memory; the library's own roles carry the same ones:

```
python analyze.py   # roles
```

```
managed policies attached to each IAM role in the EksL2 stack (top-level and nested templ
  ClusterKubectlHandlerRole94549F93               service-role/AWSLambdaBasicExecutionR
  ClusterRoleFA261979                             AmazonEKSClusterPolicy
  ClusterNodegroupNodesNodeGroupRole7AC14175      AmazonEKSWorkerNodePolicy, AmazonEKS_
(roles that only carry AWSLambdaBasicExecutionRole are the other Lambda execution roles
```

The cluster role has `AmazonEKSClusterPolicy`; the node role has `AmazonEKSWorkerNodePolicy`, `AmazonEKS_CNI_Policy` and `AmazonEC2ContainerRegistryReadOnly`. That shows my names match the library's, not that they are the right ones for your cluster (I did not read the node role page).

What cfn-lint says

```
cfn-lint cdk.out/EksL1.template.json   # first block of the lint run
```

```
$ cfn-lint EksL1.template.json
(exit status 0)
```

The L1 template lints clean (exit status 0); the L2 templates produce only W3005 warnings (a dependency that is already implied), exit status 4. cfn-lint validates against the AWS resource schemas it bundles, so it knows the allowed values. I changed one property at a time in the L1 template:

```
python mutate_lint.py
```

```
change                                                          cfn-lint
the template as synthesised                                     exit 0  -
cluster version 1.35 -> 135                                     exit 2  E3031:'135' does
cluster version 1.35 -> 'latest'                                exit 2  E3031:'latest' do
upgradePolicy.supportType 'STANDARD' -> 'PREMIUM'               exit 2  E3030:'PREMIUM' i
accessConfig.authenticationMode -> 'SAML'                       exit 2  E3030:'SAML' is n
logging type 'api' -> 'apis'                                    exit 2  E3030:'apis' is n
endpointPublicAccess 'true' -> 'yes'                            exit 2  E3012:'yes' is no
publicAccessCidrs -> 'anywhere'                                 exit 0  -
nodegroup amiType -> 'AL3'                                      exit 2  E3030:'AL3' is no
nodegroup minSize 1 -> 5 (above maxSize 3)                      exit 0  -
nodegroup scalingConfig.desiredSize -> 'two'                    exit 2  E3012:'two' is no
nodegroup instance type 't3.medium' -> 't3.huge'                exit 0  -
nodegroup updateConfig.maxUnavailable -> 0                      exit 2  E3034:0 is less t
addon name -> 'vpc-cni2'                                        exit 0  -
pod identity namespace missing                                  exit 2  E3003:'Namespace'
pod identity serviceAccount -> 'Reports_SA' (not a DNS name)    exit 0  -
```

cfn-lint caught 10 of the 15 changes and missed 5: a public-access CIDR that is not a CIDR, a node group whose minimum exceeds its maximum, an instance type that does not exist, an add-on name that does not exist, and a service account name that is not a valid Kubernetes name. Types and enumerations are checked; meaning is not. The version pattern it enforces (`1\.\d\d`) is a format check, not a check that the version is available.

Tests on the synthesised L1 stack check what the chapter argues for: a pinned version string and `STANDARD` upgrade policy, a public endpoint limited to listed CIDRs (never `0.0.0.0/0`), API-only authentication mode, audit and authenticator logs, exactly the three worker policies on the node role, a pod role that trusts only `pods.eks.amazonaws.com`, one node replaced at a time, the Pod Identity agent add-on, and no NAT gateway. All 10 passed the first time, which proves little on its own, so I ran a control: changing the allowed CIDR to `0.0.0.0/0` made the endpoint test fail (1 failed, 9 skipped), and rebuilding made it pass again.

Pod Networking: the VPC CNI and How Many Pods Fit

From the documentation: "Since each Pod is assigned its own IP address, the number of IP addresses supported by an instance type is a factor in determining the number of Pods that can

run on the instance." The default maximum is `(number of ENIs × (IPs per ENI − 1)) + 2`, where "the +2 accounts for the Amazon VPC CNI and kube-proxy running on every node, which don't consume a Pod IP address". A managed node group without a custom AMI also caps it at 110 for instances with fewer than 30 vCPUs and 250 above, and that cap "takes precedence over any other maxPods configuration". Prefix delegation (VPC CNI 1.9.0 or later) assigns more addresses; I did not read that page.

Run here: a model of the formula with data from two published files in the VPC CNI repository (the limits file the CNI itself uses, and a published list of max pods per instance type), both fetched on 2026-10-13:

```
py max_pods.py
```

```
instance types in the limits file: 1392 | in the published list: 1392
types in both: 1392 | formula disagrees with the list for: 0
only in limits file: 0 | only in list: 0

type            ENIs   IPs/ENI  max pods
t3.small           3         4        11
t3.medium          3         6        17
t3.large           3        12        35
m5.large           3        10        29
m5.xlarge          4        15        58
m5.4xlarge         8        30       234
c5.large           3        10        29
m6g.large          3        10        29
m5.24xlarge       15        50       737

types whose formula exceeds 110: 952 of 1392 (a managed node group without a custom AMI
```

The formula and the published list agree for all 1,392 instance types. They come from the same repository, so that shows internal consistency, not that they match what AWS runs today. The practical reading: a `t3.medium` holds 17 pods, an `m5.large` holds 29, and many small pods per node need a bigger instance or prefix delegation. The 952 types whose formula exceeds 110 are subject to the managed node group cap (I do not have their vCPU counts here).

The other side is the subnet. Every pod takes an address from a subnet in your VPC. *AWS Fundamentals* 5 states that AWS reserves 5 addresses in every subnet, so a `/24` offers 251, and these must also cover the nodes and the load balancer. I did not read how many addresses the

CNI keeps in reserve per node, so the number of pods a subnet really holds is lower than a naive count suggests, and I made no estimate.

Identity: IRSA and EKS Pod Identity

Both give pods AWS permissions by tying an IAM role to a Kubernetes service account instead of using the node's instance role. **From the documentation:**

	IRSA (IAM roles for service accounts)	EKS Pod Identity
Mechanism	An OIDC provider for the cluster; the pod's projected token is exchanged with `AssumeRoleWithWebIdentity`	An association (role to service account to namespace, per cluster) made in EKS; the Pod Identity Agent on each node issues credentials
Trust policy	A principal per cluster (the OIDC provider)	One principal for all clusters: `"Service": "pods.eks.amazonaws.com"`
Setup	Create the OIDC provider once per cluster; annotate the service account	Install the agent add-on (not needed on Auto Mode) and create the association
Limits and exclusions	Not for local clusters on Outposts	Up to 5,000 associations per cluster; role must be in the cluster's account; Linux EC2 nodes only (not Fargate, not Windows)
Consistency	Immediate (not stated)	Associations are eventually consistent, "potential delays of several seconds"

Both pages also say that without restricting access to the instance metadata service, a pod can reach the *node's* role, and that "containers are not a security boundary". The Pod Identity page says its association environment variables configure the AWS SDKs, so the workload must use a supported SDK version and the default credential chain.

Run here: the IRSA trust policy is where mistakes hide. The condition on the token's `sub` claim is what limits *which* service account may use the role. I wrote a small evaluator on top of the chapter 2 condition model and tried the cases. (The JSON shape is the commonly documented web-identity format; the EKS page I read did not print it.)

```
python trust_eval.py
```

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.eu-west-1.amazonaws.c
   },
   "Action": "sts:AssumeRoleWithWebIdentity",
   "Condition": {
    "StringEquals": {
     "oidc.eks.eu-west-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system
     "oidc.eks.eu-west-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.am
    }
   }
  }
 ]
}

ok   strict policy, apps/reports                              -> allowed
ok   strict policy, wrong namespace (dev/reports)             -> denied
ok   strict policy, wrong service account (apps/other)        -> denied
ok   strict policy, wrong audience                            -> denied
ok   strict policy, token from another cluster's provider     -> denied
ok   StringLike with * in the namespace, dev/reports          -> allowed
ok   StringLike with * in the namespace, kube-system/reports  -> allowed
ok   StringLike with * in the namespace, apps/other           -> denied
ok   no sub condition at all, any service account             -> allowed
expectation mismatches: 0
```

The strict policy (`StringEquals` on the full `system:serviceaccount:apps:reports` and on the audience) allowed only that service account and refused a wrong namespace, a wrong name, a wrong audience and a token from another cluster's provider. A `StringLike` with `*` in the namespace let *any* namespace's `reports` account in, including `kube-system`; and a policy with no `sub` condition let *every* service account in the cluster assume the role. With Pod Identity the trust policy has no such condition, because the association carries the restriction, so who may create associations matters; I did not read the permissions required to create one (`iam:PassRole` is the thing to check).

The CDK's L2 `addServiceAccount` hides this. The trust policy it generated has a `StringEquals` whose value is `Fn::GetAtt` of a custom resource (`Custom::AWSCDKCfnJson`): the actual `sub`

and `aud` condition is built when the stack deploys, so it cannot be read or tested in the synthesised template (a test in the exercise pins that). The L1 route with a Pod Identity association is plain and inspectable.

Manifests, and What Validates Them Without a Cluster

Run here: a set of manifests for the service from chapter 6 (service account with the IRSA annotation, a three-replica Deployment with probes, a read-only root file system and a zone spread, a PodDisruptionBudget, a Service and an Ingress for the AWS Load Balancer Controller):

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: reports
  namespace: apps
  annotations:
    # IRSA only (Pod Identity needs no annotation): the role the pods may assume through
    eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/reports-irsa
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reports
  namespace: apps
spec:
  replicas: 3
  selector:
    matchLabels: {app: reports}
  strategy:
    type: RollingUpdate
    rollingUpdate: {maxUnavailable: 0, maxSurge: 1}
  template:
    metadata:
      labels: {app: reports}
    spec:
      serviceAccountName: reports
      terminationGracePeriodSeconds: 20
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: topology.kubernetes.io/zone
          whenUnsatisfiable: ScheduleAnyway
          labelSelector:
```

```
            matchLabels: {app: reports}
      containers:
        - name: web
          image: 111122223333.dkr.ecr.eu-west-1.amazonaws.com/web:1.0.0
          ports:
            - containerPort: 8080
          env:
            - {name: HANDLE_SIGTERM, value: "1"}
          readinessProbe:
            httpGet: {path: /health, port: 8080}
            periodSeconds: 5
          livenessProbe:
            httpGet: {path: /health, port: 8080}
            initialDelaySeconds: 10
            periodSeconds: 15
          resources:
            requests: {cpu: 100m, memory: 128Mi}
            limits: {memory: 256Mi}
          securityContext:
            runAsNonRoot: true
            runAsUser: 10001
            readOnlyRootFilesystem: true
            allowPrivilegeEscalation: false
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: reports
  namespace: apps
spec:
  minAvailable: 2
  selector:
    matchLabels: {app: reports}
---
apiVersion: v1
kind: Service
metadata:
  name: reports
  namespace: apps
spec:
  selector: {app: reports}
  ports:
    - port: 80
      targetPort: 8080
---
```

```
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: reports
    namespace: apps
    annotations:
      alb.ingress.kubernetes.io/scheme: internet-facing
      alb.ingress.kubernetes.io/target-type: ip
      alb.ingress.kubernetes.io/healthcheck-path: /health
      alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS": 443}]'
      alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:eu-west-1:111122223333:certif
  spec:
    ingressClassName: alb
    rules:
      - http:
          paths:
            - path: /
              pathType: Prefix
              backend:
                service: {name: reports, port: {number: 80}}
```

The Load Balancer Controller, from the documentation: it "watches for Kubernetes Ingress or Service resources" and creates load balancers: an Application Load Balancer for an Ingress, a Network Load Balancer for a Service of type `LoadBalancer`, and (from version 2.14.0) an ALB for a Gateway. If you do not install it, Kubernetes uses a legacy cloud provider that creates Classic Load Balancers. The annotations page gives these defaults and values: `scheme` defaults to `internal` (values `internal` or `internet-facing`), `target-type` defaults to `instance` (values `instance` or `ip`), and `listen-ports` defaults to HTTP 80, or HTTPS 443 if a certificate ARN is set. Not read: the subnet tags the controller needs, its IAM policy, installation, and the Gateway configuration.

I validated the manifests offline in two layers: JSON schemas for Kubernetes v1.35.0 from a third-party project that publishes them (`yannh/kubernetes-json-schema`, strict variant), and checks I wrote from the annotation values above plus one consistency rule (the ALB health check path must be a path the readiness probe serves). Then I changed one thing at a time:

```
python validate_k8s.py
```

```
change                                                  JSON schema (v1.35.0 stri
the manifests as written                                valid
```

```
misspelt 'readinessprobe' on the container                    1 error: Deployment: spec
replicas: '3' (a string)                                      1 error: Deployment: spec
containerPort: 'http'                                         1 error: Deployment: spec
Deployment without a selector                                 1 error: Deployment: spec
selector that matches no pod label (app: other)               valid
PDB minAvailable 5 with 3 replicas                            valid
PDB with both minAvailable and maxUnavailable                 valid
memory limit below the request (64Mi < 128Mi)                 valid
Ingress scheme 'public'                                       valid
Ingress target-type 'pod'                                     valid
ALB health check path '/healthz' (readiness serves /health)   valid
misspelt annotation key 'alb.ingress.kubernetes.io/schema'    valid
```

The two layers together caught 7 of the 12 changes and missed 5: a selector that matches no pod label, a PodDisruptionBudget asking for more pods than exist, a PDB with both `minAvailable` and `maxUnavailable`, a memory limit below the request, and a misspelt annotation key (which the controller would simply ignore, leaving the default `internal` scheme, so the ALB would silently not be public). My expectation is that a real API server rejects the selector mismatch, the double PDB field and the limit below the request; I did not test it because the cluster was not available. The PDB with `minAvailable: 5` and three replicas is accepted by the schema and, in my understanding, would block every voluntary eviction, which is exactly what stops a node drain. A real API server (`kubectl apply --dry-run=server`) is the check to add, and it needs a cluster.

Versions, Upgrades and the Bill

From the documentation: a minor version is under standard support for 14 months after its EKS release, then extended support for 12 more (26 in all); extended support is enabled by default (upgrade policy `EXTENDED`); a cluster still on a version at the end of extended support is automatically upgraded (the control plane only). In-place upgrades can be rolled back within 7 days. I encoded the rule and tested it on every row of the page's release calendar:

```
py versions.py
```

```
1.37 release October 1, 2026    -> standard ends 2027-12-01 (printed 2027-12-01)  extend
1.36 release June 2, 2026       -> standard ends 2027-08-02 (printed 2027-08-02)  extend
1.35 release January 27, 2026   -> standard ends 2027-03-27 (printed 2027-03-27)  extend
1.34 release October 2, 2025    -> standard ends 2026-12-02 (printed 2026-12-02)  extend
```

```
  1.33 release May 29, 2025       -> standard ends 2026-07-29 (printed 2026-07-29)  extend
  1.32 release January 23, 2025   -> standard ends 2026-03-23 (printed 2026-03-23)  extend
  1.31 release September 26, 2024 -> standard ends 2025-11-26 (printed 2025-11-26)  extend
  rows where the 14 + 12 month rule disagrees with the page: 0

  status on 2026-10-13
    1.37 standard   (standard ends in 414 days)
    1.36 standard   (standard ends in 293 days)
    1.35 standard   (standard ends in 165 days)
    1.34 standard   (standard ends in 50 days)
    1.33 extended
    1.32 extended
    1.31 extended

  control plane fee for one cluster for one month (730 h): standard $73.00, extended $438.
  the cluster.yaml first drafted for this chapter used version 1.33, which is in: extended
```

The rule reproduced all 7 printed pairs of dates. On 2026-10-13, 1.34 had 50 days of standard support left and 1.33 was already in extended support: the `cluster.yaml` I first drafted for this chapter used 1.33 and would have started on the $0.60 rate. That is the kind of surprise to check before you create a cluster, and the reason the L1 stack sets the upgrade policy to `STANDARD` (the cluster is then upgraded at the end of standard support, which has its own risk: test upgrades before they are forced).

Upgrades, from the documentation: control plane and nodes are separate. After a control plane upgrade you must update managed node groups, self-managed nodes, add-ons and Fargate pods yourself; Auto Mode nodes follow automatically and respect Pod Disruption Budgets, with intervention needed if a blocking PDB prevents replacement within the 21-day lifetime. I did not read the node group update procedure or what `maxUnavailable` does when a PDB blocks eviction.

Cost and Clean-Up

Nothing in this chapter created anything in AWS. On this machine, outside the site: the scratch folder with its downloaded tools (about 196 MB of binaries and a 211 MB Python environment), a junction to chapter 5's `node_modules`, and whatever `kind` left in Docker (its containers and the local images from chapter 6 vanished from my view when the engine stopped, so I cannot say what remains). If you create a cluster for real (level 2, not run): the

control plane fee runs from creation (about $73 for a 730-hour month in standard support, $438 in extended), plus nodes, volumes, NAT gateways and load balancers, which I did not price here. Delete the node groups and then the cluster, and set a budget alert first (*AWS Fundamentals* 9).

WHAT WAS AND WAS NOT CHECKED

Run here: `eksctl` schema validation with 10 changes and its dry run without credentials; the CDK L1 and L2 stacks synthesised, linted (15 changes) and tested (10 tests plus a control); the max-pods model against two published files (1,392 instance types); the version rule against 7 calendar rows; the IRSA trust evaluator on 9 cases; the manifests against Kubernetes v1.35.0 schemas and annotation rules (12 changes); checksums of the two downloaded tools. **From the documentation, not run:** EKS Auto Mode, Pod Identity, IRSA, the max-pods rules, the Load Balancer Controller, the version lifecycle and prices (Region not stated for the control plane fee). **Not read or not run:** every command that needs an account; a Kubernetes API server (`kind` was created but unusable for the experiments); node group update behaviour; Fargate profiles; prefix delegation; the controller's IAM policy and subnet tags; access entries; add-on versions; networking policy; cluster logging costs; EC2 node prices; Karpenter. **Not checked at all:** a real EKS cluster.

Practice

EXERCISE 1

Write an `eksctl` `ClusterConfig`, print eksctl's own JSON schema and validate the file against it, then make ten one-change mutations (including a number-typed version and an unquoted `1.30`) and report which the schema catches. Run `eksctl create cluster --dry-run` without credentials and report what happens. Compare the version you chose with the support calendar using a small model of the 14 + 12 month rule, tested on every row of the calendar.

View solution

EXERCISE 2

Define the cluster in the CDK twice: with L1 constructs only and with the L2 `Cluster`. Synthesise both, count resources and types, find what the L2 version creates that you did not write, and lint every template. Mutate the L1 template fifteen times and report which cfn-lint catches. Write Jest tests for the L1 stack (include a control that proves one test can fail) and tests that pin the L2 surprises.

View solution

EXERCISE 3

Model the maximum pods per node from the VPC CNI's published limits and cross-check it against the published list. Write an IRSA trust policy and evaluate nine cases, including a wildcard namespace and a missing `sub` condition. Write the manifests for a service, validate them against Kubernetes schemas plus the Load Balancer Controller's documented annotation values, make twelve one-change mutations and report what each layer misses, and say what only a real API server could check.

View solution

CHAPTER 7 QUICK REFERENCE

- EKS runs the Kubernetes API server and etcd; you choose compute: managed node groups, Auto Mode (Bottlerocket nodes, 21-day maximum lifetime, Karpenter scaling), or Fargate pods with restrictions. Control plane fee $0.10 per cluster-hour in standard support, $0.60 in extended (no Region stated)
- Versions: 14 months standard + 12 extended; extended support is on by default; node groups, add-ons and Fargate pods are not upgraded with the control plane (Auto Mode nodes are). Check a version's support status before creating a cluster
- Quote Kubernetes versions in YAML: unquoted `1.35` is a number and `1.30` becomes `1.3`
- Each pod gets a VPC IP address: max pods = ENIs × (IPs per ENI − 1) + 2, capped at 110 or 250 on managed node groups without a custom AMI. A `t3.medium` holds 17 pods, an `m5.large` 29
- IRSA: OIDC provider per cluster, trust policy with `sub` and `aud` conditions (use `StringEquals` on the exact service account; a `*` namespace or no `sub` opens the role wider). Pod Identity: one trust principal (`pods.eks.amazonaws.com`), an association per service account, the agent add-on (not on Auto Mode), Linux EC2 nodes only
- The Load Balancer Controller makes an ALB for an Ingress and an NLB for a `LoadBalancer` Service; the `scheme` default is `internal`, so an Ingress meant to be public needs `internet-facing`, and a misspelt annotation is silently ignored
- The L2 CDK `Cluster` deployed as a custom resource with Lambda and Step Functions (67 resources against 22 for L1) and needs a kubectl layer; its IRSA condition is not visible in the template
- Offline checks catch types, enumerations and typos (cfn-lint caught 10 of 15 changes, the eksctl schema 7 of 10, schemas plus annotation rules 7 of 12); they do not catch nonexistent instance types or Regions, min above max, or relationships between objects. Add a real API server and the AWS service to your checks

COMING NEXT

Chapter 8 is API Gateway and serverless APIs: REST, HTTP and WebSocket APIs, integrations, stages, authorisation, throttling and the SAM CLI. SAM runs Lambda locally in Docker, and Docker Desktop's engine was not running when this chapter ended, so please start it (and check that `docker info` answers) before chapter 8. Start with *AWS Intermediate/Advanced 8*.

AWS Intermediate/Advanced

Chapter 8 · Serverless APIs: API Gateway and SAM

AWS Intermediate/Advanced 7 covered Kubernetes on EKS. This chapter goes the other way: an API with no servers or clusters of your own. Amazon API Gateway receives HTTP requests, decides who may make them, and hands them to a backend, usually a Lambda function (*AWS Fundamentals* 7). The AWS SAM (Serverless Application Model) CLI lets you describe the API in a short template, run it on your own machine with Docker, and later deploy it. Infrastructure as code itself is in *AWS Intermediate/Advanced* 4 (CloudFormation) and 5 (the CDK); SAM is a third way to write the same thing, and this chapter shows what it expands into.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS credentials on the machine that wrote this course, so nothing in this chapter was deployed and no real API Gateway was called. Docker Desktop's engine was running (it was empty at the start: 0 containers, 0 images), so, unlike chapter 7, the local experiments did run.

- **Run here (level 1):** SAM CLI 1.168.0 with AWS SAM Translator 1.114.0: `sam validate`, `sam build`, `sam local start-api` and `sam local invoke` against real Lambda container images for Python 3.13 in Docker 29.8.2; cfn-lint in two versions (1.53.3 and 1.57.2); the SAM translator run offline to see what a template expands into; four small Python models (route selection, token-bucket throttling, authorizer caching, request cost).
- **From the documentation, not run (level 3):** what the real service does. API Gateway pages were read on 2026-10-13 through a page-fetch tool that summarises pages, so each such statement says it comes from the docs. Pricing came from one page that has no price table, only worked examples (said again where it matters).
- **Needs an account, not run (level 2):** `sam deploy`, `sam sync`, `sam logs`, `sam delete`. I checked with `--help` that the commands and the flags shown exist; I did not run them.

The local emulator is a development tool, and the chapter shows that it does not behave like the service in at least four ways. Treat "it worked locally" as a statement about your function code and routing, not about your API's security.

Which Kind of API?

API Gateway sells three products: **REST APIs**, **HTTP APIs** and **WebSocket APIs**. From the documentation: "REST APIs support more features than HTTP APIs, while HTTP APIs are

designed with minimal features so that they can be offered at a lower price. Choose REST APIs if you need features such as API keys, per-client throttling, request validation, AWS WAF integration, or private API endpoints. Choose HTTP APIs if you don't need the features included with REST APIs." The rows below are the ones that matter for this chapter, read from the page's comparison tables:

FEATURE (DOCS, 2026-10-13)	REST API	HTTP API
Endpoint types	Edge-optimized, Regional, Private	Regional only
IAM, Lambda authorizers, Cognito	Yes (Cognito directly)	Yes (Cognito through a JWT authorizer)
JWT authorizer	No (use a Lambda authorizer to check JWTs)	Yes
Resource policies; AWS WAF	Yes; Yes	No; No
API keys, per-client throttling, usage plans	Yes	No
Request validation	Yes	No
Request body transformation	Yes (mapping templates)	No (parameter mapping only)
Caching; canary releases; custom gateway responses; test invocations	Yes	No
Automatic deployments	No (you deploy)	Yes
Execution logs; X-Ray tracing; mock integrations	Yes	No (access logs only)
Private integration with AWS Cloud Map	No	Yes
Maximum integration timeout (quotas pages)	50 ms to 29 s; can be raised above 29 s for Regional and private APIs, which "might require a reduction in your Region-level throttle quota"	30 s, not adjustable
Payload size	10 MB	10 MB

WebSocket APIs keep a connection open for two-way messages. I did not read the WebSocket developer pages, so this chapter says nothing about how they work; the pricing page lists them at $1.00 per million messages and $0.25 per million connection minutes (US East (N. Virginia)), with messages metered in 32 KB steps and a 128 KB maximum.

Price. From the pricing page's worked examples (the page has no standalone price table, and the fetch tool inferred the tier boundaries): REST APIs $3.50 per million requests for the first 333 million, $2.80 for the next 667 million, $2.38 above a billion (Regions named in the examples: US East (N. Virginia and Ohio), US West (Oregon), Asia Pacific (Mumbai)); HTTP APIs $1.00 per million for the first 300 million and $0.90 above that (**no Region stated**). I modelled only the request charge; data transfer, caching, Lambda, logs, WAF and the free tier (1 million REST and 1 million HTTP calls a month for 12 months, for new customers only) are left out:

```
py cost_model.py
```

```
   requests/month       REST $       HTTP $  REST/HTTP
        1,000,000         3.50         1.00       3.50
       10,000,000        35.00        10.00       3.50
      100,000,000       350.00       100.00       3.50
      400,000,000     1,353.10       390.00       3.47
    1,200,000,000     3,509.10     1,110.00       3.16
```

At these rates a REST request costs 3.5 times an HTTP request until the cheaper tiers start. That is a reason to ask which of the REST-only features you really need, not a reason to choose either by default: **a judgement, not tested here:** for a small public API that only needs routing, a JWT check and CORS, an HTTP API does the job for less; the moment you need API keys with per-client quotas, request validation or WAF, the REST API's extra cost is paying for features HTTP APIs do not have.

Defining the APIs with SAM

A SAM template is a CloudFormation template with one extra line, `Transform: AWS::Serverless-2016-10-31`, which lets it use short resource types such as `AWS::Serverless::Function`, `AWS::Serverless::Api` (a REST API) and `AWS::Serverless::HttpApi` (an HTTP API). A *Globals* section sets defaults for every function. **Run here:** a REST API with five kinds of route (open, a path parameter, a Lambda authorizer, an API key, and a validated JSON body) plus two

deliberately broken routes used later, and a separate HTTP API. They are two templates on purpose; the reason appears in the local-run section.

```
AWSTemplateFormatVersion: "2010-09-09"
Transform: AWS::Serverless-2016-10-31
Description: Chapter 8 REST API experiment

Globals:
  Function:
    Runtime: python3.13
    Timeout: 5
    MemorySize: 128

Resources:
  HelloFn:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ../app/hello/
      Handler: app.handler
      Environment:
        Variables:
          STAGE_NAME: dev
      Events:
        Hello:
          Type: Api
          Properties: {RestApiId: !Ref RestApi, Path: /hello, Method: get}
        Item:
          Type: Api
          Properties: {RestApiId: !Ref RestApi, Path: "/items/{id}", Method: get}
        Secure:
          Type: Api
          Properties:
            RestApiId: !Ref RestApi
            Path: /secure
            Method: get
            Auth:
              Authorizer: TokenAuth
        Keyed:
          Type: Api
          Properties:
            RestApiId: !Ref RestApi
            Path: /keyed
            Method: get
            Auth:
```

```
            ApiKeyRequired: true
      Create:
        Type: Api
        Properties:
          RestApiId: !Ref RestApi
          Path: /items
          Method: post
          RequestModel:
            Model: ItemModel
            Required: true
            ValidateBody: true
            ValidateParameters: true
```

```
RestApi:
  Type: AWS::Serverless::Api
  Properties:
    StageName: dev
    Cors:
      AllowMethods: "'GET,POST,OPTIONS'"
      AllowHeaders: "'Content-Type,Authorization'"
      AllowOrigin: "'https://example.com'"
    MethodSettings:
      - ResourcePath: /*
        HttpMethod: "*"
        ThrottlingBurstLimit: 5
        ThrottlingRateLimit: 2
    Models:
      ItemModel:
        type: object
        required: [name]
        properties:
          name: {type: string, minLength: 1}
          qty: {type: integer, minimum: 1}
        additionalProperties: false
    Auth:
      ApiKeyRequired: false
      UsagePlan:
        CreateUsagePlan: PER_API
        Throttle: {BurstLimit: 5, RateLimit: 2}
        Quota: {Limit: 1000, Period: MONTH}
      Authorizers:
        TokenAuth:
          FunctionArn: !GetAtt AuthFn.Arn
          Identity:
```

```
              Header: Authorization
              ReauthorizeEvery: 0
```

The Lambda functions behind them are tiny: `HelloFn` returns the event's own shape so the experiments can see it, and the authorizer allows only the token `allow-me`:

```
def handler(event, context):
    token = event.get("authorizationToken", "")
    effect = "Allow" if token == "allow-me" else "Deny"
    return {
        "principalId": "user-1",
        "policyDocument": {"Version": "2012-10-17", "Statement": [{"Action": "execute-ap
        "context": {"plan": "gold"},
    }
```

The HTTP API template is shorter because HTTP APIs have fewer settings. It has a JWT authorizer (`Jwt`), a greedy route (`/files/{proxy+}` for any method), CORS and a default throttle:

```
AWSTemplateFormatVersion: "2010-09-09"
Transform: AWS::Serverless-2016-10-31
Description: Chapter 8 HTTP API experiment

Globals:
  Function:
    Runtime: python3.13
    Timeout: 5
    MemorySize: 128

Resources:
  HttpFn:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ../app/hello/
      Handler: app.handler
      Events:
        Hello:
          Type: HttpApi
          Properties: {ApiId: !Ref HttpApi, Path: /hello, Method: get}
        Item:
          Type: HttpApi
          Properties: {ApiId: !Ref HttpApi, Path: "/items/{id}", Method: get}
```

```
        Secure:
          Type: HttpApi
          Properties: {ApiId: !Ref HttpApi, Path: /secure, Method: get, Auth: {Authorize
        Any:
          Type: HttpApi
          Properties: {ApiId: !Ref HttpApi, Path: "/files/{proxy+}", Method: any}

  HttpApi:
    Type: AWS::Serverless::HttpApi
    Properties:
      StageName: dev
      CorsConfiguration:
        AllowOrigins: ["https://example.com"]
        AllowMethods: [GET, POST]
        AllowHeaders: [Content-Type]
      DefaultRouteSettings:
        ThrottlingBurstLimit: 5
        ThrottlingRateLimit: 2
      Auth:
        Authorizers:
          Jwt:
            IdentitySource: $request.header.Authorization
            JwtConfiguration:
              issuer: https://issuer.example.com/
              audience: [my-api]
```

What the templates expand into

`sam validate` checks a template, and with `--lint` runs cfn-lint on it. **Run here:** the HTTP template passed ("*is a valid SAM Template*", exit status 0). For the REST template, `sam validate` alone passed, but `--lint` exited with status 1 because cfn-lint 1.53.3 (the version the SAM CLI installed) raised **W3005 warnings (an obsolete `DependsOn`) on resources SAM itself generates**, which you cannot edit. With cfn-lint 1.57.2 (the version chapters 1 to 7 used) the same template was clean. That is the first sign that tool versions change what you see, and the section on mistakes the checkers miss comes back to it.

The SAM translator can also run offline, which shows what the short template is hiding. I ran it on both templates (the code locations were replaced with placeholder S3 addresses in memory, as `sam package` would produce):

```
py transform.py rest/template.yaml
py transform.py http/template.yaml
```

```
rest/template.yaml: 22 resources, 9 types
   1 AWS::ApiGateway::ApiKey
   1 AWS::ApiGateway::Deployment
   1 AWS::ApiGateway::RestApi
   1 AWS::ApiGateway::Stage
   1 AWS::ApiGateway::UsagePlan
   1 AWS::ApiGateway::UsagePlanKey
   4 AWS::IAM::Role
   4 AWS::Lambda::Function
   8 AWS::Lambda::Permission
http/template.yaml: 8 resources, 5 types
   1 AWS::ApiGatewayV2::Api
   1 AWS::ApiGatewayV2::Stage
   1 AWS::IAM::Role
   1 AWS::Lambda::Function
   4 AWS::Lambda::Permission
```

The REST template declares 8 resources and becomes 22; the HTTP template declares 2 and becomes 8. The extras are the Lambda permissions that let API Gateway call each function (8 of them in the REST case), the IAM role for each function, and, for REST, the deployment, the stage, a usage plan and an API key. Neither API contains separate resources for routes: SAM writes the whole API as an OpenAPI document inside the API resource. The REST validator and model, and the HTTP API's JWT scheme, are in that document:

```
REST: paths['/items'].post in the OpenAPI body SAM generates
{
 "x-amazon-apigateway-integration": {
  "httpMethod": "POST",
  "type": "aws_proxy",
  "uri": {
   "Fn::Sub": "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${Hell
  }
 },
 "x-amazon-apigateway-request-validator": "body-and-params",
 "responses": {},
 "parameters": [
  {
   "required": true,
```

```
    "in": "body",
    "name": "itemmodel",
    "schema": {
     "$ref": "#/definitions/itemmodel"
    }
   }
 ]
}

REST: x-amazon-apigateway-request-validators
{
 "body-and-params": {
  "validateRequestParameters": true,
  "validateRequestBody": true
 }
}

REST: definitions (the model)
{
 "itemmodel": {
  "additionalProperties": false,
  "required": [
   "name"
  ],
  "type": "object",
  "properties": {
   "name": {
    "minLength": 1,
    "type": "string"
   },
   "qty": {
    "minimum": 1,
    "type": "integer"
   }
  }
 }
}

REST: deployment logical ID: RestApiDeployment2981488895  (the number changes when the AI
```

```
HTTP: components.securitySchemes and the /secure operation in the generated OpenAPI body
{
 "securitySchemes": {
```

```
    "Jwt": {
     "type": "oauth2",
     "x-amazon-apigateway-authorizer": {
      "jwtConfiguration": {
       "issuer": "https://issuer.example.com/",
       "audience": [
        "my-api"
       ]
      },
      "identitySource": "$request.header.Authorization",
      "type": "jwt"
     }
    }
   },
   "/secure get security": [
    {
     "Jwt": []
    }
   ],
   "payloadFormatVersion": "2.0"
  }

  HTTP: AWS::ApiGatewayV2::Stage properties
  {
   "DefaultRouteSettings": {
    "ThrottlingBurstLimit": 5,
    "ThrottlingRateLimit": 2
   },
   "ApiId": {
    "Ref": "HttpApi"
   },
   "StageName": "dev",
   "Tags": {
    "httpapi:createdBy": "SAM"
   },
   "AutoDeploy": true
  }
```

Two things in that output matter. First, the REST deployment's logical ID ends in a number that **changed when I renamed a route and did not change when I changed a function's timeout**:

```
  py (transform before and after one change)
```

```
logical IDs of the REST deployment and stage in the template SAM generates
as written                      : RestApiDeployment2981488895, RestApidevStage
a route changed (/hello -> /hello2): RestApiDeployment11800fc711, RestApidevStage
a function's Timeout 5 -> 7     : RestApiDeployment2981488895, RestApidevStage
```

That is how SAM gets around a REST API rule from the documentation: "Every time you update an API, you must redeploy the API to an existing stage or to a new stage. Updating an API includes modifying routes, methods, integrations, authorizers, resource policies, and anything else other than stage settings." A new deployment resource is a new logical ID, so CloudFormation creates it and repoints the stage. I did not read how SAM computes the number, only saw that it follows the API definition. Second, the generated HTTP stage has `AutoDeploy: true`, which matches the docs' table (HTTP APIs have automatic deployments, REST APIs do not).

Running the API Locally

Run here: `sam build` copied each function into `.aws-sam/build`, then `sam local start-api` started a web server on a local port and one Docker container per function. The first start built an emulation image from AWS's public Lambda Python 3.13 base image (two images of 795 MB and 834 MB stayed in Docker afterwards), and `--warm-containers EAGER` started the containers before the first request. Here is every request I sent to the REST API and what came back:

```
./probe_rest.sh   # compact view of the saved run
```

```
GET /hello                                200 OK                event 1.0 (REST), CO
GET /items/42?x=1                         200 OK                event 1.0 (REST), CO
GET /secure (no token)                    401 UNAUTHORIZED
GET /secure (token allow-me)              200 OK                event 1.0 (REST), CO
GET /secure (token nope)                  403 FORBIDDEN
GET /keyed (no API key)                   200 OK                event 1.0 (REST), CO
POST /items valid                         200 OK                event 1.0 (REST), CO
POST /items missing name                  200 OK                event 1.0 (REST), CO
POST /items qty 0                         200 OK                event 1.0 (REST), CO
POST /items extra property                200 OK                event 1.0 (REST), CO
POST /items not JSON                      200 OK                event 1.0 (REST), CO
POST /items no body                       200 OK                event 1.0 (REST), CO
OPTIONS /hello origin example.com         200 OK                CORS headers
OPTIONS /hello origin evil.example        200 OK                CORS headers
```

```
GET /nope                                403 FORBIDDEN
DELETE /hello                            403 FORBIDDEN
GET /bad (no statusCode)                 502 BAD GATEWAY
GET /slow (8 s, timeout 5 s)             502 BAD GATEWAY
```

What the local server did like the service, and what it did not. Routing and path parameters worked; the Lambda authorizer worked (401 with no token, 403 for a token the authorizer denied, and the authorizer's `context` reached the function); CORS response headers were added; a function that returned something other than a proxy response (`/bad`) and one that ran over its timeout (`/slow`, 8 s against 5 s) both gave 502, and unknown routes gave the 403 "*Missing Authentication Token*" message. What it did **not** do:

- **Request validation was not enforced.** Five bodies that break the model (missing `name`, `qty` 0, an extra property, text that is not JSON, no body) all got `200` from the function. The documentation says the real service "immediately fails the request, returns a 400 error response to the caller" before the integration runs; I could not test that.
- **API keys were not enforced.** `/keyed` is marked `ApiKeyRequired` and answered 200 with no key.
- **Throttling was not applied.** The stage allows a burst of 5 and 2 requests per second; I sent 30 requests back to back and all 30 returned 200.
- **CORS did not filter by origin:** a preflight from `https://evil.example` got the same `Access-Control-Allow-Origin: https://example.com` header as the allowed origin, so the server only echoes your configuration; the browser, not the server, is what refuses a mismatched origin.

The HTTP API, run on its own, had the most worrying result:

```
./probe_http.sh   # compact view of the saved run
```

```
GET /hello                               200 OK              event 2.0
GET /items/42?x=1                        200 OK              event 2.0
GET /secure (no token)                   200 OK              event 2.0
GET /secure (Bearer junk)                200 OK              event 2.0
GET /files/a/b/c.txt (proxy+)            200 OK              event 2.0
POST /files/x                            200 OK              event 2.0
OPTIONS /hello origin example.com        200 OK              CORS headers
```

```
GET /nope                                403 FORBIDDEN
DELETE /hello                            403 FORBIDDEN
```

The JWT-protected route `/secure` answered 200 with no token at all, and with the junk token `Bearer junk`. The only sign was one line in the server log: "*Authorizer 'Jwt' for route '/secure' was not found, skipping*". The route was served unprotected. In the runs that had a Lambda authorizer the server also printed a banner ("*does not guarantee 100% fidelity between authorizers locally and authorizers deployed on AWS*"), but in the HTTP-only run the one skipping line was all I found about the JWT route. If you test an HTTP API locally, read the log and do not use "it returned 200" as evidence that a JWT route is protected. The DELETE and unknown-route requests got the REST-style 403 locally, while the documentation says an unmatched request to an HTTP API returns `{"message":"Not Found"}`, so the local server's error shapes are not a guide to the real HTTP API either.

One template with both API types is a trap. When I first put the REST and HTTP APIs in one template, both defined `GET /hello` and the one local server served only one of them:

```
combined template, one sam local start-api
```

```
combined template (a REST API and an HTTP API that both define GET /hello), one 'sam loc
--- GET /hello (REST API defines it with HelloFn, HTTP API defines it with HttpFn)
{"version": "2.0", "path": "/hello", "method": "GET", "pathParameters": {}, "query": nul
--- OPTIONS /hello CORS headers (REST says Content-Type,Authorization; HTTP says Content
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: GET,OPTIONS,POST
Access-Control-Allow-Headers: Content-Type
```

The request went to the HTTP API's function (event version 2.0), and the CORS headers came from the HTTP API's configuration, not the REST API's. On the real service the two APIs have separate URLs and never collide, so the local server's single namespace is a limit of the tool. Keep one API per template while testing locally.

Events: two shapes, and generated fixtures are samples

REST APIs (and HTTP APIs with payload format 1.0) send the Lambda function the 1.0 event; HTTP APIs default to 2.0 in the console and require you to say which when you use the CLI, CloudFormation or an SDK (docs). The documented differences: 2.0 has `rawPath`, a `cookies` list, `requestContext.http`, no `multiValueHeaders`, and duplicate headers or query values joined

with commas. `sam local generate-event` makes sample events. **Run here:** I generated one of each type and listed the keys:

```
py mk8.py
```

```
REST fixture (sam local generate-event apigateway aws-proxy)
  top-level keys: body, resource, path, httpMethod, isBase64Encoded, queryStringParamete
  requestContext keys: accountId, resourceId, stage, requestId, requestTime, requestTime
  version field: (none: REST proxy events have no version field) | isBase64Encoded: True
HTTP API fixture (generate-event apigateway http-api-proxy)
  top-level keys: version, routeKey, rawPath, rawQueryString, cookies, headers, queryStr
  requestContext keys: accountId, apiId, authentication, authorizer, domainName, domainP
  version field: 2.0 | isBase64Encoded: True | pathParameters: {'parameter1': 'value1'}
```

The fixtures are canned samples, not requests built from your route. I asked for `--path hello`, yet the REST fixture's `pathParameters` is `{'proxy': '/hello'}`, the HTTP fixture has `{'parameter1': 'value1'}`, and both have `isBase64Encoded: true` and a sample body. Edit them to look like the request you want to test before you trust a `sam local invoke` result.

Responses. With a REST (1.0) proxy integration the function must return `statusCode`, `headers` and `body`; `/bad` returned a dictionary without `statusCode` and the local server gave 502. With payload format 2.0 the docs say API Gateway "can infer the response format": valid JSON without a `statusCode` becomes a 200 response with `content-type: application/json` and the output as the body. **Run here:** locally an HTTP API function that returned `{"message": "no statusCode here"}` got 200 and `Content-Type: application/json`, and one that returned the string `plain string` got 200 with that text, as the docs describe. The same function on a REST API gave 502, so the difference is real, and a handler written for one API type is not automatically correct for the other.

Routes in HTTP APIs

A route is an HTTP method plus a path (`GET /pets/{id}`), `ANY` matches other methods, `{proxy+}` (greedy, last in the path) matches everything below it, and `$default` catches the rest. From the docs, the selection order is: "1. Full match for a route and method. 2. Match for a route and method with a greedy path variable ({proxy+}). 3. The $default route", and with no

match the API returns `{"message":"Not Found"}`. I wrote a small model of that and tested it on the four rows of the docs' example table:

```
def select(routes, method, path):
    """routes: list of 'METHOD /path' keys or '$default'."""
    best = None
    for key in routes:
        if key == "$default":
            continue
        m, p = key.split(" ", 1)
        if m not in (method, "ANY"):
            continue
        r = match_path(p, path)
        if not r:
            continue
        kind, statics, depth = r
        # order: full before greedy; exact method before ANY; more static segments; long
        rank = (kind == "full", m == method, statics, depth)
        if best is None or rank > best[0]:
            best = (rank, key)
    if best: return best[1]
    return "$default" if "$default" in routes else None
```

```
py route_model.py
```

```
ok   GET   /pets/dog/1    -> GET /pets/dog/1  (docs: GET /pets/dog/1)
ok   GET   /pets/dog/2    -> GET /pets/dog/{id}  (docs: GET /pets/dog/{id})
ok   GET   /pets/cat/1    -> GET /pets/{proxy+}  (docs: GET /pets/{proxy+})
ok   POST  /test/5        -> ANY /{proxy+}  (docs: ANY /{proxy+})
extra cases (assumptions, not in the docs table):
  routes=['$default'] GET /anything -> $default
  routes=['GET /a'] GET /b -> None
  routes=['GET /a'] POST /a -> None
  routes=['ANY /a', 'GET /a'] GET /a -> GET /a
  routes=['GET /{proxy+}', 'GET /pets/{proxy+}'] GET /pets/x/y -> GET /pets/{proxy+}
docs-table rows disagreeing with the model: 0
```

The model reproduced all four documented rows. The "extra cases" are my additions (a static segment beating a variable, the longer greedy prefix winning) and are not on the page. I then ran the same five routes through `sam local` and the emulator picked the same route for all the

documented requests, with one difference worth noting: for `POST /test/5` its event's `routeKey` was `POST /{proxy+}`, not the `ANY /{proxy+}` route you configured. I did not check what the real service puts in `routeKey`, so do not write a handler that depends on it.

Authorisation

From the documentation: both API types can use IAM (SigV4-signed requests), Lambda authorizers and Cognito (REST directly; HTTP through a JWT authorizer); REST adds resource policies; HTTP adds the JWT authorizer. I ran only one of them locally, the REST Lambda authorizer, and the results are in the probe table above.

Lambda authorizers

A Lambda authorizer receives the caller's identity and returns an IAM policy, a principal ID and optionally a `context` object that is passed to the integration. The docs describe two types: `TOKEN` (a bearer token in one header; the header name is the cache key) and `REQUEST` (headers, query strings, stage variables and context variables; "We recommend that you use a REQUEST authorizer", because it can use several identity sources). With caching on for a `REQUEST` authorizer, a missing identity source makes API Gateway return 401 without calling your function. In the docs' TOKEN example, a `Deny` policy gives 403 outside the test environment, the token `unauthorized` or an empty string gives 401, and any other token gives 500 (an error from the function). My local run agreed on 401 (no token) and 403 (denied).

The caching gotcha, from the docs: if you enable authorization caching, API Gateway "caches the policy so that the Lambda authorizer function isn't invoked again. Ensure that your policy is applicable to all resources and methods across your API." An authorizer that returns `Allow` for only the method ARN it was asked about will allow the first request and then refuse the same caller on other routes while the cache lasts. I modelled it (my reading of the docs, not run against the service) with the documented method-ARN shape and IAM-style wildcards:

```
py authz_cache.py
```

```
no cache, policy names only the requested ARN
  authorizer invoked 3x; results: GET /hello -> 200, GET /items/42 -> 200, POST /items -
cache 300 s, policy names only the requested ARN
  authorizer invoked 1x; results: GET /hello -> 200, GET /items/42 -> 403, POST /items -
```

```
  cache 300 s, policy covers the whole stage (./dev/*)
    authorizer invoked 1x; results: GET /hello -> 200, GET /items/42 -> 200, POST /items -
```

With no cache the narrow policy works, because the authorizer is called for every request, which is also why testing without caching hides the problem. With a 300-second cache the second and third requests were denied; a policy for `.../dev/*` fixed it. (The sample local run used `ReauthorizeEvery: 0`, caching off, so this effect never showed locally.)

JWT authorizers (HTTP APIs)

From the documentation: a JWT authorizer reads the token from the identity source (the token alone, or prefixed `Bearer`), checks the algorithm and signature against the issuer's `jwks_uri` (only RSA-based algorithms are supported; keys are cached for up to two hours), then validates `kid`, `iss` (must equal the configured issuer), `aud` or `client_id` (must match a configured audience; `client_id` is checked only if `aud` is absent), `exp`, `nbf`, `iat` and, if the route configures scopes, `scope` or `scp` (at least one must match). The claims reach a Lambda function in `requestContext.authorizer.jwt.claims`. Quotas on the HTTP API quotas page: 10 authorizers per API, 50 audiences per authorizer, 10 scopes per route, a 1,500 ms timeout for the key-set endpoint, and a 10,000 ms timeout for a Lambda authorizer's response.

The docs also warn that "there is no standard mechanism to differentiate JWT access tokens from other types of JWTs, such as OpenID Connect ID tokens", and recommend requiring authorization scopes on routes. In my reading, a route with a JWT authorizer and no scopes accepts any valid token from that issuer for that audience, ID tokens included; **I did not test this against the service.** The local emulator, as shown above, skipped the authorizer altogether.

Throttling, Usage Plans and Quotas

From the documentation: API Gateway "throttles requests to your API using the token bucket algorithm, where a token counts for a request". The *rate* is how fast tokens are added each second and the *burst* is the bucket's size. When the bucket is empty clients "may receive 429 Too Many Requests". The page calls limits "targets rather than guaranteed request ceilings". Four kinds of limit exist (AWS Regional limits, per-account per-Region limits, per-API per-stage limits for a method, and per-client limits tied to API keys in a usage plan), applied in this order: "1. per-client or per-method limits in a usage plan, 2. per-method limits for a stage, 3. account-level per Region, 4. AWS Regional". The account default is 10,000 requests per second with a maximum bucket of 5,000 across all API types in a Region (2,500 and 1,250 in a list of newer Regions), and

the burst quota "is not a quota that a customer can control". The docs list no per-client throttling for HTTP APIs, so for them a stage or route setting is what the HTTP template uses (`DefaultRouteSettings`).

A model of the documented algorithm shows what rate and burst mean in practice. It is a model of the idea, not of the service's behaviour at the margins, which the page says is best effort:

```
py bucket_model.py
```

```
rate 2/s, burst 5, 30 requests all at once (spacing 0.0 s):
  200s: 5  429s: 25
rate 2/s, burst 5, 30 requests at 10 per second (spacing 0.1 s):
  200s: 10  429s: 20   first 429 at request # 7
rate 2/s, burst 5, 30 requests at 2 per second (spacing 0.5 s):
  200s: 30  429s: 0
steady state: sustained ceiling is the rate; burst is how far a short spike may overrun
account default (docs): 10,000 rps, bucket 5,000; at spacing 0 the first 5,000 of 12,000
  200s: 5000  429s: 7000
```

A burst of 30 at once let 5 through; 30 requests at ten per second let 10 through and refused the 7th; 30 requests at two per second all passed. The local server applied none of this (30 of 30 returned 200), so throttling needs the real service (or a load test against it) to check.

Quotas worth knowing (docs, 2026-10-13)

QUOTA	REST API	HTTP API
Routes / resources per API	300 resources (adjustable)	300 routes (adjustable); 300 integrations (not adjustable)
Stages per API	10 (adjustable)	10 (adjustable)
Authorizers per API	10 (adjustable via Support)	10 (adjustable via Support)
API keys; usage plans	10,000 keys per account per Region; 300 usage plans (adjustable)	Not applicable
Header size	20,480 bytes in total	10,240 bytes for request line plus headers
Mapping template; model	300 KB; 400 KB per API	Not applicable

QUOTA	REST API	HTTP API
Method ARN length	1,600 bytes (longer: 414 response)	Not read

Request Validation, Mapping and CORS

REST request validation, from the docs. A *request validator* assigned to a method can check that required parameters in the URI, query string and headers are "included and not blank" ("only the existence of a parameter, not its type or format") and that the body "adheres to the configured JSON schema" for the request's content type; "if no matching content type is found, request validation is not performed" (use `$default` as the content type to apply one model to every type). A failure stops the request before the integration. SAM wrote exactly that into the generated document (`body-and-params` validator, the model under `definitions`, the validator on `POST /items`): see the OpenAPI output above. **HTTP APIs have no request validation**, so the function has to validate its own input, and so does anything the REST validator does not cover (types of query strings, business rules, a body whose content type differs from the one in the model).

CORS for HTTP APIs, from the docs. When CORS is configured, API Gateway "automatically sends a response to preflight OPTIONS requests, even if there isn't an OPTIONS route", adds the configured CORS headers to the integration's response, and **ignores CORS headers your function returns**. The headers are returned only for requests that carry an `origin` header (and a preflight needs `Access-Control-Request-Method` too). One trap: if a `$default` route has an authorizer, preflight requests (which browsers send without credentials) hit it too; the docs' fix is an `OPTIONS /{proxy+}` route with no authorizer and an integration attached. I did not read the REST CORS page, so what the `Cors` option in the REST template produces on the real service is not covered here; the local server returned the headers I configured.

Mapping. REST APIs can transform request bodies with mapping templates (the docs table: yes), HTTP APIs only map parameters. This chapter uses Lambda proxy integrations throughout, where API Gateway passes the request straight through, so no mapping templates are shown, and non-proxy Lambda, HTTP and AWS-service integrations (listed in the docs' tables as available for both types) were not examined.

Stages, Deployments and Custom Domains

From the documentation: a REST API is reachable only after you "create an API deployment and associate it with a stage". A stage (`dev`, `prod`) is "a logical reference to a lifecycle state" and part of the invoke URL: `https://{restapi-id}.execute-api.{region}.amazonaws.com/{stageName}`. A custom domain name replaces the hostname, and "to support multiple APIs under the custom domain name, you must map an API stage to a base path". HTTP APIs have a `$default` stage that serves requests no named stage matches; with none, the API answers `{"message":"Not Found"}` and writes no CloudWatch logs. I did not read the pages on custom domains, certificates or canary releases, so those are listed in the docs' tables but not explained here (custom domains and certificates come back in *AWS Intermediate/Advanced* 10).

What the Checkers Miss

Run here: I changed one thing at a time in the two templates (25 changes) and ran `sam validate` (the SAM translator) and cfn-lint 1.53.3 and 1.57.2 on each. The first two rows are the unchanged templates as controls:

```
py mutate_sam.py
```

```
change                                                  sam validate cfn-lint 1.53.3
rest: the templates as written (control)                pass         warning only
http: the templates as written (control)                pass         pass
rest: Runtime python3.13 -> python3.99                  pass         error E3030
rest: Runtime python3.13 -> python3.7 (retired)         pass         error E2531
rest: Timeout 5 -> 901                                  pass         warning only
rest: MemorySize 128 -> 64                              pass         error E3034
rest: Handler line removed from HelloFn                 error        error E0001
rest: event Method get -> fetch                         pass         warning only
rest: event Path without leading slash                  pass         warning only
rest: StageName dev -> 'dev stage'                      pass         warning only
rest: CORS origin without inner quotes                  pass         warning only
rest: RequestModel names a model that does not exist    error        error E0001
rest: route uses an authorizer that is not defined      error        error E0001
rest: ReauthorizeEvery 0 -> 3601                        pass         warning only
rest: authorizer FunctionArn -> plain text              pass         warning only
rest: MethodSettings rate 2 -> 'fast'                   pass         error E3012
rest: usage plan Quota Period MONTH -> YEAR             pass         warning only
rest: usage plan burst below rate (1 < 2)               pass         warning only
rest: same path+method declared twice                   error        error E0001
rest: model type 'object' -> 'objekt'                   pass         warning only
```

```
  http: JwtConfiguration removed                          error       error E0001
  http: JWT identity source -> 'Authorization'            pass        pass
  http: JWT issuer not a URL                              pass        pass
  http: throttle burst 5 -> 'lots'                        pass        error E3012
  http: {proxy+} not at the end of the path               pass        pass
  http: route Method any -> anyy                          pass        pass
  http: CORS AllowMethods GET -> FETCH                    pass        pass

  25 one-change mutations. Errors reported by: sam validate 5; cfn-lint 1.53.3 10; cfn-lin
  Caught by sam validate or cfn-lint 1.53.3: 10; by sam validate or cfn-lint 1.57.2: 8; by
```

Of 25 changes, `sam validate` reported an error for 5, cfn-lint 1.53.3 for 10 and cfn-lint 1.57.2 for 3; 10 were caught by at least one of the three and 15 by none. The 15 that nobody flagged include a nonexistent HTTP method (`fetch`, `anyy`), a route path without a leading slash, a Lambda timeout of 901 seconds, an authorizer caching time of 3,601 seconds, an authorizer `FunctionArn` that is plain text, a stage name with a space, a model whose type is `objekt`, a JWT issuer that is not a URL, an identity source that is not a request expression, a greedy `{proxy+}` in the middle of a path (the docs say it must be last), and a usage plan with a burst smaller than its rate. "Not caught" means these tools did not say so; I did not deploy them, so I do not know which the service would reject.

Two other lessons. First, the five changes that referred to something that does not exist or is missing (an undefined authorizer or model, a duplicated route, a JWT authorizer with no configuration, a function with no handler) were caught by the SAM translator, which is what `sam validate` runs; cfn-lint 1.53.3 repeated those failures as `E0001` and 1.57.2 did not. Schema checks catch wrong types and out-of-range values (a retired Python runtime, a memory size of 64, a throttle that is a word) but not meaning. Second, **the two cfn-lint versions disagree**:

```
1.57.2 reported errors for 3 of the changes and missed 7 that 1.53.3 reported (the nonexistent
```

runtime `python3.99`, a throttle of `lots`, and the five translator failures). In 1.53.3 the unchanged REST template also had warnings that 1.57.2 did not. I used the same translator version (1.114.0) in both environments, so the translator is not the reason, but I did not look for the cause. The practical rule: pin the lint version in your project, and re-run your own deliberate-mistake test whenever you upgrade it.

Deploying (not run)

Level 2, needs an AWS account, not run here. I confirmed with `--help` that these commands and flags exist in SAM CLI 1.168.0:

```
# first time: prompts for stack name, Region, and whether SAM may create IAM roles
sam deploy --guided

# later, without prompts (the S3 bucket for the code is chosen for you)
sam deploy --stack-name reports-api --resolve-s3 --capabilities CAPABILITY_IAM

# faster loop while developing: sync code changes straight to the stack
sam sync --stack-name reports-api --watch

# follow the Lambda logs
sam logs --stack-name reports-api --tail

# remove the stack and the artifacts sam deploy created
sam delete --stack-name reports-api
```

The `sam local start-api` banner itself says that the way to test authorizers "deployed on AWS" is `sam sync`. Before any real deployment, set a budget alert (*AWS Fundamentals* 9) and check the free tier: the pricing page says its free offers are "only available to new AWS customers".

Cost and Clean-Up

Nothing in this chapter created anything in AWS. On this machine: the scratch folder `C:/lwaws/c8` (about 540 MB, mostly two Python environments of 337 MB and 206 MB), and in Docker two Lambda base images (795 MB and 834 MB; Docker was empty before) and no containers: with `--warm-containers EAGER`, stopping the server left its containers running, so I removed them after each run. If you run this yourself, `docker ps -a` after a local session and remove the leftovers. For real deployments: the request charges modelled above, plus Lambda, logs and data transfer, which I did not price.

WHAT WAS AND WAS NOT CHECKED

Run here: `sam validate` and `--lint` on two templates; the SAM translator offline (8 to 22 and 2 to 8 resources); `sam build`, `sam local start-api` and `sam local invoke` in Docker for a REST and an HTTP API (18 REST requests, 11 HTTP requests (two of them repeated), a combined-template run, a route-priority run and a 30-request burst); 25 template mutations against three tools; four models.
From the documentation, not run: every statement about what the real API Gateway does (feature

table, quotas, throttling, validation, CORS, JWT and Lambda authorizer behaviour, routing, deployment, prices). **Not read:** WebSocket APIs, REST CORS, custom domains and certificates, canary releases, mapping templates, caching, usage plan and API key mechanics, resource policies, private APIs, access logging, Lambda authorizers for HTTP APIs, IAM authorization details, the pricing of caching and data transfer. **Not checked at all:** a real API in a real account; the local server's behaviour for HTTP API Lambda authorizers.

Practice

EXERCISE 1

Write a SAM template for a REST API (open route, path parameter, Lambda authorizer, validated JSON body, throttle, usage plan) and a second one for an HTTP API (JWT authorizer, greedy route, CORS). Run `sam validate --lint`, translate both offline and count what they expand into, extract the generated OpenAPI, show how the REST deployment's logical ID changes, and make 25 one-change mutations against `sam validate` and two cfn-lint versions. Report what none of them catches.

View solution

EXERCISE 2

Build and run each API locally with `sam local start-api` (one API per template), probe it with `curl` (authorizer allowed, denied and missing, five invalid bodies, a missing API key, a 30-request burst, CORS preflights, a non-proxy response and a timeout), and list what the emulator enforces and what it does not. Show that a JWT route is served without a token and find the log line that says so. Then put both APIs in one template and show what collides.

View solution

EXERCISE 3

Model HTTP API route selection from the documented priority rules and test it on the docs' example table and against the emulator; model token-bucket throttling and show what a burst, a fast stream and a slow stream do; model a cached Lambda authorizer policy that names only one method ARN; and compute the request charge of REST and HTTP APIs at five traffic levels. For each of three scenarios you invent, choose REST or HTTP and say which docs row decided it.

View solution

CHAPTER 8 QUICK REFERENCE

- REST API: API keys, usage plans, request validation, WAF, resource policies, caching, private endpoints, mapping templates; you deploy. HTTP API: cheaper, JWT authorizer, automatic deployments, no validation, no usage plans. Check the docs' feature table before choosing
- SAM is CloudFormation plus `Transform: AWS::Serverless-2016-10-31`: 8 declared resources became 22 for the REST API, 2 became 8 for the HTTP API; routes live in a generated OpenAPI document
- `sam local start-api` runs your Lambda code in Docker with real routing and a Lambda authorizer, but did **not** enforce request validation, API keys, throttling or a JWT authorizer (the JWT route was served open and only the log said so)
- One API per template while testing locally: a REST and an HTTP API sharing `GET /hello` collided on one server
- Event versions differ: REST and payload 1.0 give `httpMethod` and `path`; HTTP payload 2.0 gives `rawPath`, `routeKey`, `cookies` and `requestContext.http`. 2.0 can infer a response; REST proxy needs `statusCode` (else 502)
- HTTP route priority: full match, then greedy `{proxy+}`, then `$default`; no match is `{"message":"Not Found"}`. With a `$default` authorizer, add an unauthenticated `OPTIONS /{proxy+}`
- Lambda authorizer caching: the cached policy must cover every route the caller may use; a policy naming one method ARN breaks the others while cached
- Throttling is a token bucket (rate = refill per second, burst = bucket size); 429 when empty; account default 10,000 rps and 5,000 burst per Region; usage-plan and per-method limits are applied first
- Checkers miss a lot: of 25 one-change template mutations, 15 were caught by none of `sam validate`, cfn-lint 1.53.3 or 1.57.2, and the two cfn-lint versions disagreed. Pin versions and keep your own mistake tests

COMING NEXT

Chapter 9 is messaging and events: SQS queues (standard and FIFO, visibility timeouts, dead-letter queues), SNS fan-out, EventBridge buses and rules, and Step Functions. An emulator for those services will only be used if it can be shown to behave correctly; otherwise the chapter will say which parts come from the documentation. Docker was working for this chapter and nothing needs starting first. Start with *AWS Intermediate/Advanced 9*.

AWS Intermediate/Advanced

Chapter 9 · Messaging and Events: SQS, SNS, EventBridge and Step Functions

AWS Intermediate/Advanced 8 put an API in front of a Lambda function that answers a request and finishes. Real systems also do work that should not make the caller wait, must survive a failure, or must reach several places. Four services cover that: **SQS** holds messages until a consumer takes them, **SNS** pushes one message to many subscribers, **EventBridge** routes events to targets by their content, and **Step Functions** runs a multi-step workflow with retries and error paths. This chapter explains each from the documentation and then tests what can be tested on a laptop. *Event-driven architecture* as a design idea is not re-taught here; the AWS-specific rules and limits are.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

There is no AWS account and no AWS credentials on the machine that wrote this course, so no real SQS queue, SNS topic, EventBridge bus or state machine was used. The course outline said to use an emulator only if it can be shown to be adequate, so a good part of this chapter is that test.

- **Run here (level 1):** the *moto* 5.2.4 emulator in server mode (SQS, SNS, EventBridge) driven through boto3 1.43.112; *Step Functions Local* (AWS's Docker image `amazon/aws-stepfunctions-local`) with a mock configuration; a real Lambda Python 3.13 container through the SAM CLI from chapter 8; cfn-lint 1.57.2; and four small programs I wrote: an EventBridge pattern matcher, an SQS consumer, a cross-resource template checker and a cost sketch. The emulators are checked against the documentation, not the other way round.
- **From the documentation, not run (level 3):** every statement about what the real services do. The pages were read on 2026-10-13 through a page-fetch tool that summarises pages; some pages came back empty or as general knowledge, and each of those is named where it matters (SNS fan-out, SQS and SNS price tables).
- **Needs an account, not run (level 2):** the same boto3 code against real AWS, and real deployments of the template.

LocalStack's free edition was retired in March 2026 (*AWS Intermediate/Advanced* course outline) and I did not try the young alternatives. AWS itself calls Step Functions Local "unsupported" and without "feature parity".

Which Service for Which Job?

The comparison below uses only facts from the pages I read; the last column is where I add a judgement, marked as such.

	SQS	SNS	EVENTBRIDGE	STEP FUNCTIONS
Shape	A queue consumers pull from	A topic that pushes to subscribers	A bus with rules that match event content	A state machine that runs steps
Keeps messages	Default 4 days; 60 s to 14 days	Delivers; archive and replay exist for FIFO topics (page not read)	Retries a failed delivery for 24 hours and up to 185 times by default, then drops it unless you configure a dead-letter queue	Standard: execution history for 90 days after it closes
Selecting what a receiver gets	Everything in the queue	Subscription filter policy on attributes or body	Event pattern on any field of the event	Choice states in the definition
Failure handling	Visibility timeout, then retry; dead-letter queue after `maxReceiveCount`	Not covered here	Retry policy and dead-letter queue per target	`Retry` and `Catch` per state
Delivery	At least once (standard); FIFO removes duplicates inside 5 minutes	Not read	Not read beyond retries	Standard: exactly once per execution; Express: at least once (async) or at most once (sync)
Good fit (judgement, not tested)	Buffering work for a consumer that may be slow or down	One event, several independent reactions	Routing many event types to many targets without code	Ordered steps with retries, waits and branches

A common chain, and the one this chapter builds, is: something publishes to an SNS topic; a filtered subscription puts matching messages on an SQS queue; a Lambda function drains the queue; messages that keep failing land in a dead-letter queue; and a state machine handles the multi-step part with retries. EventBridge sits beside it for events that come from AWS itself or from other applications.

Can the Emulator Be Trusted?

I ran the same questions against the documentation and the emulators and counted the agreements. **The result is mixed, and it decides how the rest of the chapter uses the emulator:** good for behaviour, poor for limits, and with one real defect.

AREA (EMULATOR)	WHAT I CHECKED	AGREED WITH THE DOCS
SQS behaviour (moto 5.2.4)	Visibility timeout, change visibility, dead-letter queue, FIFO group, deduplication, in-flight blocking, long polling	10 of 10
SQS limits (moto)	12 documented limits, one value just inside and one just outside	**6 of 12**
SNS (moto)	Fan-out, envelope versus raw delivery, filter-policy scopes, 12 operators	6 of 6 groups
EventBridge patterns (moto)	43 cases built from the docs' examples	**42 of 43** (a real defect on the `time` field)
Step Functions retries (Step Functions Local)	9 retry and catch scenarios, delays measured	9 of 9
Step Functions definition checks (Step Functions Local)	22 invalid definitions	rejected 21

The details follow in each service's section. The practical reading: use the emulators to test how your *code* behaves (does my consumer cope with a redelivery, does my pattern match my event), and use something else for limits and for validity (cfn-lint, your own checks, and ultimately the real service).

SQS: Queues, Visibility and Dead Letters

The rules, from the documentation

- **Visibility timeout.** A received message "remains in the queue but becomes temporarily invisible"; the default is 30 seconds, the range 0 seconds to 12 hours. If you do not delete it before the timeout, it becomes visible again. `ChangeMessageVisibility` can extend or

shorten the timeout, and setting it to 0 makes the message available at once. "Extending the timeout doesn't reset this 12-hour limit."

- **At-least-once.** "There's no absolute guarantee that a message won't be delivered more than once during the visibility timeout period." Design consumers so that seeing a message twice is harmless.
- **Dead-letter queues.** A redrive policy gives a `maxReceiveCount`: "the number of times a consumer can receive a message from a source queue before it is moved to a dead-letter queue". The dead-letter queue must be in the same account and Region. For a standard queue the message keeps its original enqueue timestamp, so "it is a best practice to always set the retention period of a dead-letter queue to be longer than the retention period of the original queue"; for FIFO queues the timestamp resets. The page warns against a dead-letter queue on a FIFO queue where the exact order must survive.
- **Polling.** Short polling (the default) "queries a subset of servers" and a call "might not return all of your messages"; long polling (a wait time above 0, maximum 20 seconds) queries all servers and returns as soon as a message is available.
- **FIFO.** `MessageGroupId` is required; messages in a group are delivered in order, and while one is in flight "subsequent messages in that group are not made available until the in-flight message is either deleted or the visibility timeout expires". A resend with the same deduplication ID inside the 5-minute interval adds nothing; content-based deduplication hashes the body but "not the attributes of the message". Each partition is limited to 300 transactions per second per action (3,000 messages per second with batching of 10) unless high-throughput mode is on.
- **Limits.** Message size 1 byte to 1,048,576 bytes (1 MiB); retention 60 seconds to 14 days; delay 0 to 15 minutes; 10 messages per batch; 10 message attributes; group ID up to 128 characters; batch entry ID up to 80 characters (alphanumerics, hyphen, underscore); about 120,000 in-flight messages on a standard queue (`OverLimit` under short polling, silently nothing under long polling).
- **Price (partial).** The page I read gave no price table. It says every action is a request, each 64 KB chunk of a payload is billed as one request (a 1 MiB message is 16 requests), and the first million requests a month are free. FIFO and standard requests have different rates that I could not read.

Run here: behaviour against the emulator

Each check sends real SQS API calls to moto and compares the result with the sentence above. Two mistakes of mine are worth keeping: my first FIFO ordering check asked for 10 messages in

one call and got the whole group back at once (`g1-1` , `g1-2` and `g2-1`), which looked like a failure; the page I read does not say whether one call may return several messages of a group, so I split it into an unscored information line and a one-message-at-a-time check that matches the quoted sentence.

```
py sqs_behaviour.py   # condensed
```

```
ok   visibility timeout hides a received message, then it reappears
       -> 1st receive 1 message, immediate 2nd receive 0, after 2.4 s receive 1 (Receive
ok   ChangeMessageVisibility to 0 makes the message available at once
       -> receive right after: 1 message
ok   after maxReceiveCount receives the message moves to the dead-letter queue
       -> receives that returned the message (3 allowed): [1, 1, 1, 0, 0, 0]; message in
ok   FIFO send without MessageGroupId fails
       -> MissingParameter
ok   FIFO drops a resend with the same deduplication id inside the 5-minute interval
       -> two sends with the same id, receive returned ['a']
ok   content-based deduplication hashes the body only, not attributes
       -> same body with different attributes sent twice, receive returned 1
info one receive asking for up to 10 messages returned ['g1-1', 'g1-2', 'g2-1'] (the pag
ok   FIFO: while a group's message is in flight, later messages of that group are not de
       -> one message per receive, nothing deleted: [['g1-1'], ['g2-1'], []]
ok   FIFO: after the in-flight message is deleted the next one of its group is delivered
       -> after deleting g1-1 a receive returned ['g1-2']
ok   long polling waits for the wait time on an empty queue
       -> WaitTimeSeconds=3 on an empty queue returned after 3.0 s with 0 messages
ok   long polling returns as soon as a message arrives
       -> message sent after 1 s, receive with WaitTimeSeconds=10 returned after 1.0 s w
10 of 10 behaviour checks agree with the documentation
```

All ten scored checks agreed. The limits are another matter. I sent one value just inside each documented limit and one just outside:

```
py sqs_quotas.py
```

```
limit (docs)                                        inside
message size max 1,048,576 bytes                    1,048,576 byte accepted
visibility timeout max 12 hours (43,200 s)          43200 accepted
retention min 60 s                                  60 accepted
retention max 1,209,600 s (14 days)                 1209600 accepted
message timer (delay) max 15 minutes                900 accepted
```

```
long polling wait max 20 s                              20 (empty queu accepted
batch max 10 messages                                   10 entries accepted
up to 10 message attributes                             10 attributes accepted
FIFO MessageGroupId max 128 characters                  128 accepted
batched message ID max 80 characters, [A-Za-z0-9_-]     80 accepted
batched message ID only alphanumerics, - and _          'a-b_1' accepted
message content: control character #x1 not allowed      'plain text' accepted

6 of 12 limits enforced exactly as documented
```

moto enforced only 6 of the 12 limits. It accepted a visibility timeout of 43,201 seconds, retention of 59 and 1,209,601 seconds, 11 message attributes, a 129-character group ID and a control character in the body. A test suite on this emulator would pass code that the real service rejects. This is why the template tests later in the chapter use cfn-lint, which caught the queue-attribute ranges.

At-least-once in practice: a slow consumer

Run here: a queue with a 2-second visibility timeout, a message that takes 3.5 seconds to process, and two workers. The first worker is still busy when the message reappears and the second takes it:

```
def worker(q, name, process_seconds, handler, stop, log):
    while not stop.is_set():
        r = sqs.receive_message(QueueUrl=q, MaxNumberOfMessages=1, WaitTimeSeconds=1, At
        for m in r:
            body = json.loads(m["Body"])
            done = handler(name, body, process_seconds, m)
            log.append((round(time.time() - T0, 1), name, body["order"], "receive#" + m[
            try:
                sqs.delete_message(QueueUrl=q, ReceiptHandle=m["ReceiptHandle"])
            except Exception as e:
                log.append((round(time.time() - T0, 1), name, body["order"], "delete fai


EFFECTS = []


def naive(name, body, secs, m):
    time.sleep(secs)
    EFFECTS.append(body["order"])      # the side effect (charge the card)
    return "charged"
```

```
PROCESSED, LOCK = set(), threading.Lock()


def idempotent(name, body, secs, m):
    with LOCK:
        if body["order"] in PROCESSED:
            return "skipped (already processed)"
        PROCESSED.add(body["order"])
    time.sleep(secs)
    EFFECTS.append(body["order"])
    return "charged"
```

```
py consumer_demo.py
```

```
-- naive consumer (visibility timeout 2 s, processing takes 3.5 s, two workers)
   t= 3.5s worker-1  order A-1001  receive#1     charged
   t= 5.6s worker-2  order A-1001  receive#2     charged
   side effects performed: 2 -> ['A-1001', 'A-1001']
-- idempotent consumer (visibility timeout 2 s, processing takes 3.5 s, two workers)
   t= 2.1s worker-2  order A-1001  receive#2     skipped (already processed)
   t= 3.5s worker-1  order A-1001  receive#1     charged
   side effects performed: 1 -> ['A-1001']

naive consumer charged order A-1001 2 times; idempotent consumer 1 time(s)
```

The naive consumer charged order `A-1001` twice. The idempotent one saw the repeat at 2.1 seconds, recognised the order ID and skipped it, so the charge happened once. Here the "already processed" check is an in-memory set; in a real system it must be shared and atomic (a

crashed because I forgot to ask for the `ApproximateReceiveCount` attribute: the workers died after the side effect and before the delete, so the first numbers were meaningless until I fixed and re-ran it.

SNS: Fan-Out and Filter Policies

From the documentation: "By default, an Amazon SNS topic subscriber receives every message that's published to the topic." A subscription can carry a *filter policy*; with scope `MessageAttributes` it is matched against the attributes, with `MessageBody` against the message's JSON body. "If any single property in this policy doesn't match an attribute assigned to the message, the policy rejects the message." Raw message delivery to SQS strips the SNS metadata so "the message is sent as is" (otherwise the receiver gets a JSON envelope); with raw delivery to an SQS queue at most 10 message attributes can be sent, and messages with more are discarded. The pricing page says SNS does not charge for per-message delivery to SQS and Lambda, but gave no price table, so the publish price is not stated here.

Run here: the page's own example message and its two policies (one that should accept it, one that should reject it because it requires an `encrypted` attribute the message does not have), in both scopes, plus twelve operators one at a time:

```
py sns_tests.py   # condensed
```

```
ok   one publish reaches every subscribed queue
       -> queue 1 got 1, queue 2 got 1
ok   default delivery wraps the message in an SNS JSON envelope
       -> envelope keys: Message, MessageId, Signature, SignatureVersion, SigningCertURL
ok   raw message delivery sends the message 'as is'
       -> body: 'This is a test message.'
ok   filter policy scope MessageAttributes: docs' accepting policy, rejecting policy, no
       -> accepting policy delivered 1, rejecting policy delivered 0, no policy delivere
ok   filter policy scope MessageBody: docs' accepting policy, rejecting policy, no polic
       -> accepting policy delivered 1, rejecting policy delivered 0, no policy delivere
operator checks on one message (attribute scope): 12 of 12 as expected
6 of 6 groups agree with the documentation (0 operator differences)
```

moto agreed everywhere, including the `anything-but`, numeric and `exists` operators. Two caveats. First, my message delivered to the queue without any queue access policy; the documentation page on fan-out to SQS did not load through my fetch tool (it returned general knowledge instead), so from my own understanding, not from a page I read, the real service needs the queue's policy to allow `sns.amazonaws.com` to send. The emulator cannot tell you that you forgot it. Second, the template in this chapter includes such a policy and cfn-lint cannot tell if it is missing (see the template section).

EventBridge: Rules and Event Patterns

From the documentation: an *event pattern* "has the same structure as the events they match" and either matches or does not. All fields in the pattern must match; the values of a field are an array of alternatives (OR); if the event's value is an array, the pattern matches when "the intersection of the event pattern array and the event array is non-empty". The operators page lists, among others: `prefix`, `suffix`, `equals-ignore-case`, `anything-but` (with values, lists, `prefix`, `suffix`, `equals-ignore-case` and `wildcard`), `numeric`, `exists`, `cidr`, `wildcard` (event buses only) and `$or`. Numbers are compared as 64-bit doubles, exact only up to $\pm 2^{53}$; a pattern with more than 1,000 `$or` combinations is rejected; if a key is repeated in a pattern "the last reference will be the one used". Null and empty strings "are not interchangeable". A warning on the page is worth repeating: a rule can trigger the very change it detects and loop, with "higher-than-expected charges and throttling".

Delivery, from the docs: if a target cannot be reached, EventBridge "retries sending the event for 24 hours and up to 185 times with an exponential back off and jitter" by default (both settable per target); after that "the event is dropped" unless the target has a dead-letter queue. Price (page, Region not stated, from worked examples): custom events $1.00 per million; AWS management events are free to ingest; Scheduler invocations are free for 14,000,000 a month and then $1.00 per million.

Run here: a reference matcher and the emulator

Trying to test patterns, I found that moto's `TestEventPattern` is a stub: it returned an empty 200 reply for a matching pattern and for a non-matching one alike. The test therefore had to use the real path: a rule on a fresh bus, an SQS queue as target, `put_events`, and a look at whether a message arrived. As a second opinion I wrote a reference matcher that implements only what the pages above state:

```
def _field(conds, v):
    vals = v if isinstance(v, list) else [v]
    if isinstance(v, list) and not v:
        vals = [MISSING] if any(isinstance(c, dict) and "exists" in c for c in conds) el
    return any(_op(c, x) for c in conds for x in vals)


def matches(pattern, event):
    for key, sub in pattern.items():
        if key == "$or":
```

```
            if not any(matches(p, event) for p in sub):
                return False
            continue
        v = event.get(key, MISSING) if isinstance(event, dict) else MISSING
        if isinstance(sub, dict):
            if v is MISSING or not isinstance(v, dict):
                return False
            if not matches(sub, v):
                return False
        else:
            if not _field(sub, v):
                return False
    return True
```

Then 43 cases, each with the outcome the docs' example or sentence implies:

```
py eb_pattern_tests.py
```

```
case                                                  docs  model  moto
docs' main example: source + detail-type + state      True  True   True
same, state running                                   False False  False
one wrong field among right ones                      False False  False
event array field, non-empty intersection             True  True   True
event array field, empty intersection                 False False  False
prefix on time                                        True  True   False    <--
prefix, no match                                      False False  False
anything-but a different string                       True  True   True
anything-but the actual string                        False False  False
anything-but a list containing it                     False False  False
anything-but a list without it                        True  True   True
anything-but prefix, value has the prefix             False False  False
anything-but prefix, value lacks it                   True  True   True
anything-but equals-ignore-case, other casing         False False  False
suffix .png                                           True  True   True
suffix is case-sensitive                              False False  False
suffix with equals-ignore-case                        True  True   True
anything-but suffix, matching file                    False False  False
equals-ignore-case on detail-type                     True  True   True
prefix with equals-ignore-case                        True  True   True
exists true, field present                            True  True   True
exists true, field absent                             False False  False
exists false, field absent                            True  True   True
```

```
numeric equals                                              True  True   True
numeric range (10, 20], value 20                            True  True   True
numeric range (10, 20], value 10                            False False  False
numeric range (10, 20], value 21                            False False  False
numeric 3.018e2 equals 301.8                                True  True   True
numeric against a string value                              False False  False
$or, second branch true                                     True  True   True
$or, neither branch true                                    False False  False
wildcard dir/*.png                                          True  True   True
wildcard dir/*.png, other folder                            False False  False
anything-but wildcard */lib/*, path contains /lib/          False False  False
anything-but wildcard */lib/*, path without it              True  True   True
cidr 10.0.0.0/24 inside                                     True  True   True
cidr 10.0.0.0/24 outside                                    False False  False
empty string matches [""]                                   True  True   True
null matches [null]                                         True  True   True
[""] does not match null                                    False False  False
[null] does not match an empty string                       False False  False
repeated key: the last one is used (raw JSON)               True  True   True

43 cases: reference matcher agrees with the docs on 43, moto agrees on 42; moto rejected
```

The reference matcher agreed with the documentation on all 43 cases (so it is a fair second opinion); **moto agreed on 42**. The one disagreement is a real defect in moto 5.2.4 for something the docs show directly (`"time": [{"prefix": "2017-10-02"}]` is the docs' own example). I narrowed it down:

```
py (probe of time-field matching)
```

```
prefix on source ('app.orders', prefix 'app.')              delivered=1
prefix on detail-type                                       delivered=1
prefix on time ('2017-12-22T18:43:48Z')                     delivered=0
exact value on time                                         delivered=0
exists on time                                              delivered=1
prefix on a detail string (same text)                       delivered=1
```

A prefix on `source`, on `detail-type` and on the same text inside `detail` all matched; an exact value or prefix on the top-level `time` never did, while `exists` on `time` did. moto stores the time exactly as sent, so I could not tell the cause from outside; the lesson is that an emulator

can fail on one field and agree on the rest. Not tested: IPv6 addresses, pattern size limits, the wildcard complexity limit, archives and replay, schedules.

Step Functions: Workflows with Retry and Catch

Standard or Express, from the docs: Standard workflows run up to a year, are "exactly-once", keep history for 90 days after closing, and are billed per state transition; Express workflows run up to 5 minutes, are "at-least-once" (asynchronous) or "at-most-once" (synchronous), and are billed by requests, duration and memory. Express workflows do not support the job-run (`.sync`) or callback (`.waitForTaskToken`) integration patterns, Distributed Map or activities. The workflow type cannot be changed after creation. Quotas worth knowing: a Standard execution fails if its history reaches 25,000 events; task, state and execution input or output is limited to 256 KiB; names are limited to 80 characters; 1,000,000 open executions per account per Region.

Errors, from the docs. Any state except `Pass` and `Wait` can fail, and "Step Functions defaults to failing the entire state machine execution". `Task`, `Parallel` and `Map` states can have `Retry` (a list of retriers) and `Catch`. A retrier has `ErrorEquals`, `IntervalSeconds` (default 1), `MaxAttempts` (default 3; 0 means never retry), `BackoffRate` (default 2.0), an optional `MaxDelaySeconds` (above 0) and `JitterStrategy` (`FULL` or `NONE`, default `NONE`). `States.ALL` "must appear alone in the `ErrorEquals` array and must appear in the last retrier"; it does not catch `States.DataLimitExceeded` or `States.Runtime`; `States.TaskFailed` matches any known error except `States.Timeout`. Retries run first and the matching catcher second, and "retries are treated as state transitions" for billing.

Run here: retry timing on Step Functions Local

Step Functions Local calls no real services when a mock configuration scripts each Task's result. The docs describe the file: `StateMachines` with `TestCases`, and `MockedResponses` numbered by attempt (`"0-1"` is attempts 0 and 1) with `Return` or `Throw`; every attempt a retry might make needs a mocked response, or the execution fails. The state machine, the mock file and the results:

```
  def machine(retry, catch=True):
      task = {"Type": "Task", "Resource": "arn:aws:states:::lambda:invoke", "Parameters":
      if catch:
```

```
        task["Catch"] = [{"ErrorEquals": ["States.ALL"], "ResultPath": "$.error", "Next"
    return {"StartAt": "Charge", "States": {"Charge": task, "Done": {"Type": "Succeed"},
```

```
{
  "StateMachines": {
    "RetryDemo": {
      "TestCases": {
        "SucceedAfterTwo": {"Charge": "ThrowTwiceThenOk"},
        "AlwaysFails": {"Charge": "AlwaysTransient"},
        "OtherError": {"Charge": "Declined"}
      }
    },
    "CappedRetry": {"TestCases": {"AlwaysFails": {"Charge": "AlwaysTransient"}}},
    "UncappedRetry": {"TestCases": {"AlwaysFails": {"Charge": "AlwaysTransient"}}},
    "JitterRetry": {"TestCases": {"AlwaysFails": {"Charge": "AlwaysTransient"}}},
```

```
py sfn_retry.py
```

```
scenario                                                tries waits (s)          st
2 transient failures, then success (1 s, x2, max 3)     3     [1.0, 2.0]         SUC
always fails: retries used up, then Catch               4     [1.0, 2.0, 4.0]    SUC
error name not in ErrorEquals: no retry, Catch          1     []                 SUC
interval 3, x2, max 3, MaxDelaySeconds 5                4     [3.0, 5.0, 5.0]    SUC
interval 3, x2, max 3, no cap                           4     [3.0, 6.0, 12.0]   SUC
JitterStrategy FULL, interval 2, x2, max 3 (run 1)      4     [1.0, 3.0, 8.0]    SUC
JitterStrategy FULL, interval 2, x2, max 3 (run 2)      4     [2.0, 3.0, 1.0]    SUC
docs' 'retry all except States.Timeout': Timeout thrown 1     []                 SUC
same policy: another error thrown once                  2     [1.0]              SUC
```

The waits matched the pages in every scenario: 1 then 2 seconds; 1, 2, 4 when all retries are used before the catcher takes over; **3, 5, 5 with `MaxDelaySeconds` 5 and 3, 6, 12 without it** (the docs' own two examples); an error not listed in `ErrorEquals` was not retried at all; and the docs' "retry everything except `States.Timeout`" policy did not retry a timeout and did retry another error. The two `JitterStrategy FULL` runs gave different waits (1, 3, 8 and 2, 3, 1 seconds), each inside its documented range of 0 to 2, 0 to 4 and 0 to 8; both runs produced whole seconds, which the page does not promise. The execution status was `SUCCEEDED` after a catch because

the fallback state was a `Pass` state: **a handled failure is a success as far as the execution status goes**, so alarms on status alone will miss it.

Cost sketch

From the pricing page (a table was missing from the text, so these come from its examples): Standard is $0.000025 per state transition in US East (N. Virginia), with 4,000 free a month, and "each retry will be charged as an additional state transition". Express is $1.00 per million requests (N. Virginia) plus $0.00001667 per GB-second (Region not stated). Counting my own transitions as states entered plus retries (AWS's count may differ by one):

```
py cost9.py
```

```
execution path (from the retry runs)                    states retries transitions
2 transient failures, then success (1 s, x2, max 3)     2      2       4
always fails: retries used up, then Catch               2      3       5
error name not in ErrorEquals: no retry, Catch          2      0       2
same policy: another error thrown once                  2      1       3

free tier: 4,000 transitions a month = 1,000 executions of a 4-transition path
one more retry on every execution of the 2-state path adds 50% to its transition count
EventBridge custom events: 1,000,000 events x $1.00/M = $1.00 (Region not stated on the
```

A path with three retries costs 2.5 times a path without any, so retries are not free; and the same page's free tier covers only 1,000 executions of a four-transition path.

What validates a state machine definition?

I changed one thing at a time in a definition with a Task, Choice, Wait, Parallel, Map, Succeed and Fail state, and asked two checkers: Step Functions Local (it validates on `CreateStateMachine`) and cfn-lint (the definition in a CloudFormation `DefinitionString`). The "expect" column says what the docs or the ASL (Amazon States Language) specification say; for the specification I rely on my own reading, because I did not fetch it this session:

```
py asl_mutations.py
```

```
change                                          expect   Step Functions Local   cfn
(control) the definition as written             valid    accepted               acc
state Type 'Task' -> 'Taks'                      invalid  rejected               rej
```

```
Next points to a state that does not exist          invalid  rejected          acc
StartAt names a missing state                       invalid  rejected          rej
StartAt removed                                     invalid  rejected          rej
an unreachable extra state                          invalid  rejected          acc
Choice rule without Next                            invalid  rejected          rej
Choice without a Default                            valid    accepted          acc
Task without Resource                               invalid  rejected          rej
Parallel without Branches                           invalid  rejected          rej
a state with both Next and End                      invalid  rejected          rej
Succeed state given a Next                          invalid  rejected          rej
Fail state given a Next                             invalid  rejected          rej
Wait with Seconds and Timestamp                     invalid  rejected          acc
Wait with neither Seconds nor Timestamp             invalid  rejected          acc
Map MaxConcurrency 'two'                            invalid  rejected          rej
Retry in a Pass state                               invalid  rejected          rej
Retry MaxAttempts -1                                invalid  rejected          rej
Retry IntervalSeconds 0                             invalid  accepted          acc
Retry BackoffRate 0.5                               unknown  rejected          acc
Retry JitterStrategy 'HALF'                         invalid  rejected          acc
Retry MaxDelaySeconds 0                             invalid  rejected          acc
States.ALL not last in Retry                        invalid  rejected          acc
States.ALL together with another error name         invalid  rejected          acc
Catch Next names a missing state                    invalid  rejected          acc
custom error name starting with 'States.'           unknown  accepted          acc
Fail state without Error                            valid    accepted          acc

26 changes; 22 are invalid by the docs or the ASL spec as I read it.
of those 22: rejected by Step Functions Local 21, by cfn-lint 12, by at least one 21, by
accepted by both although invalid: ['Retry IntervalSeconds 0']
'valid' or 'unknown' cases rejected by a tool: [('Retry BackoffRate 0.5', 'rejected', 'a
```

Step Functions Local rejected 21 of the 22 invalid definitions; cfn-lint rejected 12. cfn-lint missed a `Next` pointing at no state, an unreachable state, both Wait mistakes, a `Catch` to a missing state, and every retry-field mistake the error-handling page defines (a jitter strategy of `HALF`, `MaxDelaySeconds` 0, `States.ALL` not last or not alone). Both accepted a `Retry` with `IntervalSeconds` 0 although the page says the value is "a positive integer"; I do not know what the real service does. AWS's own alternatives need an account: the TestState API (the Step Functions Local page recommends it instead of Step Functions Local) and the `ValidateStateMachineDefinition` API (it appears in the quotas page; I did not read how it behaves).

Lambda as a Queue Consumer

From the documentation (Lambda SQS pages):

- Set the queue's visibility timeout "to at least six times the configuration timeout on your function"; with a batching window, six times the timeout *plus* `MaximumBatchingWindowInSeconds`. The function timeout must not exceed the queue's visibility timeout; Lambda checks this and returns an error.
- Set `maxReceiveCount` to at least 5 on a queue read by Lambda, which gives Lambda a few chances to retry before a message goes to the dead-letter queue.
- Batch size up to 10,000 on a standard queue and 10 on a FIFO queue; above 10 you must set a batching window of at least 1 second (standard queues only). A batch is also limited by the 6 MB synchronous payload.
- By default, an error fails the whole batch and every message in it returns to the queue after the visibility timeout. With `ReportBatchItemFailures` the function returns `{"batchItemFailures": [{"itemIdentifier": "id2"}]}` and only those messages return. An empty or null list is a complete success; an invalid JSON response, an empty or null `itemIdentifier`, a wrong key name or an ID that does not exist is a complete failure; an exception fails the whole batch. For a FIFO queue the function "should stop processing messages after the first failure and return all failed and unprocessed messages".

Run here: a handler that reports only the records it cannot process, run in a real Lambda Python 3.13 container with `sam local invoke` and a five-record event made from the SAM-generated SQS fixture (which has a single record with the body `Hello from SQS!`; I copied the record five times and changed the IDs and bodies):

```
import json


def handler(event, context):
    """Process each record on its own; report only the failures (ReportBatchItemFailures
    failures = []
    for record in event["Records"]:
        try:
            body = json.loads(record["body"])
            if body.get("poison"):
                raise ValueError("cannot process " + body["order"])
            print("processed", body["order"])
        except Exception as exc:
```

```
            print("failed", record["messageId"], exc)
            failures.append({"itemIdentifier": record["messageId"]})
    return {"batchItemFailures": failures}
```

```
sam local invoke BatchFn -e ev_five.json
```

```
No current session found, using default AWS::AccountId

START RequestId: c2e777de-713f-48da-a03b-0fcb7ceaf46a Version: $LATEST
processed A-1
failed id2 cannot process A-2
processed A-3
failed id4 cannot process A-4
processed A-5
END RequestId: c2e777de-713f-48da-a03b-0fcb7ceaf46a
{"batchItemFailures": [{"itemIdentifier": "id2"}, {"itemIdentifier": "id4"}]}
```

The function returned exactly the documented shape for records 2 and 4. What this run does **not** show is what the Lambda service then does with the response and the queue: that is the event source mapping, which only exists on AWS, so the rules in the list above are from the documentation.

The Whole Design as a Template

Run here: one CloudFormation template for the chain (queue with a dead-letter queue, a FIFO queue, a topic with a filtered subscription and its queue policy, an event bus with a rule and an audit queue, the Lambda consumer and its mapping, and a state machine):

```
  OrdersDLQ:
    Type: AWS::SQS::Queue
    Properties:
      QueueName: orders-dlq
      MessageRetentionPeriod: 1209600

  OrdersQueue:
    Type: AWS::SQS::Queue
    Properties:
```

```
      QueueName: orders
      VisibilityTimeout: 185
      MessageRetentionPeriod: 345600
      ReceiveMessageWaitTimeSeconds: 20
      RedrivePolicy:
        deadLetterTargetArn: !GetAtt OrdersDLQ.Arn
        maxReceiveCount: 5

  PaymentsQueue:
    Type: AWS::SQS::Queue
    Properties:
      QueueName: payments.fifo
      FifoQueue: true
      ContentBasedDeduplication: true
      VisibilityTimeout: 60
```

```
  OrdersSubscription:
    Type: AWS::SNS::Subscription
    Properties:
      TopicArn: !Ref OrdersTopic
      Protocol: sqs
      Endpoint: !GetAtt OrdersQueue.Arn
      RawMessageDelivery: true
      FilterPolicyScope: MessageAttributes
      FilterPolicy:
        store: [example_corp]
        price_usd:
          - numeric: [">=", 100]
```

It linted clean with cfn-lint 1.57.2 (exit status 0). **My first draft of this template had a mistake that cfn-lint did not report**: I set the queue's visibility timeout to 180 seconds with a 30-second function and a 5-second batching window, but the docs' rule is 6 × 30 + 5 = 185. I found it when I reread the Lambda page against the template, not through a tool, and then wrote the small checker below for the rules that need two resources together. I changed one thing at a time in the template (30 changes):

```
py mutate_stack.py   # condensed
```

```
change                                                  cfn-lint 1.57.2
(control) template as written                           exit 0
```

```
queue VisibilityTimeout 185 -> 43201                              caught E3034
queue MessageRetentionPeriod -> 59                                caught E3034
queue MessageRetentionPeriod -> 1209601                           caught E3034
ReceiveMessageWaitTimeSeconds 20 -> 21                            caught E3034
queue MaximumMessageSize above 1 MiB                              caught E3034
queue DelaySeconds 901                                            caught E3034
FIFO queue name without .fifo                                     caught E3501
ContentBasedDeduplication on a standard queue                     caught E3501
maxReceiveCount 5 -> 'five'                                       caught E3012
DLQ target ARN is a queue that does not exist                     caught E1010
DLQ target ARN is the queue's own ARN                             caught E3004
event source mapping BatchSize 10 -> 10001                        caught E3034
BatchSize 100 with no batching window                             missed
FIFO queue mapped with BatchSize 11                               caught E3705
function Timeout 30 -> 600 (queue timeout stays 185)              missed
queue VisibilityTimeout 185 -> 60 (less than 6 x 30 s)            missed
subscription Endpoint is the queue URL, not its ARN               missed
subscription RawMessageDelivery 'yes'                             missed
FilterPolicyScope 'Body'                                          missed
filter policy operator 'numerc'                                   missed
queue policy removed (SNS may not send to the queue)              missed
audit queue policy trusts the wrong service                       missed
event pattern with an unknown operator 'prefx'                    missed
event pattern numeric with a string operand                       missed
rule target RetryPolicy MaximumRetryAttempts 'many'               caught E3012
Express workflow that has a callback (.waitForTaskToken)          missed
state machine definition is not valid JSON                        missed
state machine Next names a state that does not exist              missed
Lambda runtime python3.7 (retired)                                caught E2531
consumer role trusted by the wrong service                        missed
30 one-change mutations: cfn-lint reported an error for 15; 15 passed or only warned
```

cfn-lint reported an error for 15 of 30 and passed or only warned for 15. It caught all the single-property ranges (the SQS limits above, a FIFO name without `.fifo`, batch sizes, a non-numeric retry count, a dead-letter queue that does not exist or is the queue itself). It missed everything that needs knowledge of two resources, or of what a string means: a function timeout longer than the queue's visibility timeout, a timeout shorter than six times the function's, a batch of 100 with no window, a subscription endpoint that is the queue URL instead of its ARN, `RawMessageDelivery: yes`, a filter-policy scope or operator that does not exist, a missing queue policy, a queue policy that trusts the wrong service, an event pattern with an operator that does not exist, a callback state in an Express workflow, and a state machine definition that is broken or points at a missing state.

Two of those misses I probed further, because the definition was inside `Fn::Sub`. With a plain `DefinitionString` cfn-lint also passed a definition that is not valid JSON, a `Next` to a missing state and an Express workflow with `.waitForTaskToken`; wrapped in `Fn::Sub` it printed only W1020 (a substitution rule). So the substitution is not the cause: this version does not check those things at all.

```
py (probe of DefinitionString plain and wrapped in Fn::Sub)
```

```
valid definition, plain string                              exit 0 -
valid definition, wrapped in Fn::Sub                        exit 4 W1020
broken JSON (missing brace), plain string                   exit 0 -
broken JSON (missing brace), wrapped in Fn::Sub             exit 4 W1020
Next to a missing state, plain string                       exit 0 -
Next to a missing state, wrapped in Fn::Sub                 exit 4 W1020
EXPRESS + .waitForTaskToken, plain string                   exit 0 -
EXPRESS + .waitForTaskToken, wrapped in Fn::Sub             exit 4 W1020
STANDARD + .waitForTaskToken, plain string (should be fine) exit 0 -
```

The checker I wrote reads the template, follows `Ref` and `GetAtt` between the mapping, the queue and the function, and applies the Lambda page's rules:

```
def check(text):
    t = yaml.load(text, Loader=Loader)
    R, P = t["Resources"], t.get("Parameters", {})
    out = []

    def val(v, default=None):
        n = ref_name(v)
        if n in P: return P[n].get("Default", default)
        return v if v is not None else default

    for mid, m in R.items():
        if m["Type"] != "AWS::Lambda::EventSourceMapping":
            continue
        pr = m["Properties"]
        q, fn = R.get(ref_name(pr["EventSourceArn"])), R.get(ref_name(pr["FunctionName"]
        if not q or q["Type"] != "AWS::SQS::Queue" or not fn:
            continue
        qp, fp = q["Properties"], fn["Properties"]
        vt = int(qp.get("VisibilityTimeout", 30))
        timeout = int(val(fp.get("Timeout"), 3))
```

```
        window = int(pr.get("MaximumBatchingWindowInSeconds", 0))
        batch = int(pr.get("BatchSize", 10))
        fifo = bool(qp.get("FifoQueue"))
        if timeout > vt:
            out.append(("ERROR", mid, f"function timeout {timeout} s is longer than the
        need = 6 * timeout + window
        if vt < need:
            out.append(("WARN", mid, f"visibility timeout {vt} s is below the recommende
        if fifo and batch > 10:
            out.append(("ERROR", mid, f"batch size {batch} on a FIFO queue (max 10)"))
        if not fifo and batch > 10000:
            out.append(("ERROR", mid, f"batch size {batch} on a standard queue (max 10,0
        if batch > 10 and window < 1:
            out.append(("ERROR", mid, f"batch size {batch} > 10 needs MaximumBatchingWin
        rp = qp.get("RedrivePolicy")
        if rp:
            mrc = int(rp.get("maxReceiveCount", 0))
            if mrc < 5:
                out.append(("WARN", mid, f"maxReceiveCount {mrc} is below the recommende
            d = R.get(ref_name(rp.get("deadLetterTargetArn")))
            if d and int(d["Properties"].get("MessageRetentionPeriod", 345600)) <= int(q
                out.append(("WARN", mid, "the dead-letter queue's retention is not longe
    return out
```

```
py crosscheck_demo.py
```

```
the template as written
   no findings
function Timeout 30 -> 600 (queue timeout stays 185)
   ERROR function timeout 600 s is longer than the queue's visibility timeout 185 s (Lam
   WARN  visibility timeout 185 s is below the recommended 6 x 600 s + 5 s window = 3605
queue VisibilityTimeout 185 -> 60
   WARN  visibility timeout 60 s is below the recommended 6 x 30 s + 5 s window = 185 s
queue VisibilityTimeout 180 (my first draft, with a 5 s window)
   WARN  visibility timeout 180 s is below the recommended 6 x 30 s + 5 s window = 185 s
BatchSize 100 with no batching window
   ERROR batch size 100 > 10 needs MaximumBatchingWindowInSeconds of at least 1
FIFO queue mapped with BatchSize 11
   WARN  visibility timeout 60 s is below the recommended 6 x 30 s + 5 s window = 185 s
   ERROR batch size 11 on a FIFO queue (max 10)
maxReceiveCount 5 -> 3
   WARN  maxReceiveCount 3 is below the recommended 5 for a Lambda-fed queue
```

```
  DLQ retention 1209600 -> 345600 (same as source)
    WARN  the dead-letter queue's retention is not longer than the source queue's
```

It flagged the three mutations cfn-lint missed in this group (function timeout above the queue's visibility timeout, a visibility timeout below six times the function's, a batch of 100 with no window) and my own 180, and I added three rules for good measure (FIFO batch size, `maxReceiveCount` below 5, a dead-letter queue that does not outlive its source). It is a sketch for this template's shape, not a general tool: a project that wants these checks should keep a script like it next to the template, with tests like the mutations above.

Against Real AWS (not run)

Level 2, needs an account. The boto3 calls used in this chapter are the same ones you would make against the real services: drop the `endpoint_url`, set credentials and a Region, and the code in `sqs_behaviour.py`, `sns_tests.py`, `eb_pattern_tests.py` and `consumer_demo.py` runs against AWS (queues and topics have names you must make unique, and the tests leave their resources behind, so a real run needs a clean-up step I did not write). There is no AWS CLI on this machine, so I give no CLI commands. The template deploys with `aws cloudformation deploy` or `sam deploy` (*AWS Intermediate/Advanced* 4 and 8), and costs money only for requests and state transitions at the rates above; set a budget alert first (*AWS Fundamentals* 9) and delete the stack afterwards.

Cost and Clean-Up

Nothing in this chapter created anything in AWS. On this machine: the scratch folder `C:/lwaws/c9` (a Python environment and test files), the moto server process (stopped at the end) and the Step Functions Local container (removed at the end), and three Docker images: the Step Functions Local image (1.89 GB) and the two Lambda images from chapter 8 (795 MB and 834 MB). SAM's local invoke leaves no container behind (checked with `docker ps -a`).

> **WHAT WAS AND WAS NOT CHECKED**
>
> **Run here:** 10 SQS behaviour checks; 12 SQS limit checks; SNS fan-out, raw delivery, two filter scopes and 12 operators; 43 EventBridge pattern cases against a reference matcher and moto (and a probe of the `time` field); 9 retry/catch scenarios and 26 definition mutations on Step Functions Local; 30

template mutations against cfn-lint; a cross-resource checker; a slow-consumer demonstration; a partial-batch handler in a real Lambda container; a cost sketch. **From the documentation, not run:** everything the real services do (limits, delivery, redrive, retention, throttling, prices, the event source mapping, retry behaviour on AWS). **Not read or not run:** SNS fan-out access policy, SNS FIFO topics, SQS and SNS price tables, EventBridge Scheduler, Pipes, archives and replay, Step Functions Express in practice, `Map` and `Parallel` behaviour, the TestState API, the ASL specification, moto's own Step Functions support, and **a real account** for any of it.

Practice

EXERCISE 1

Start the moto server and test SQS against the documentation: visibility timeout, `ChangeMessageVisibility`, a dead-letter queue, FIFO deduplication and group ordering, and long polling. Then test twelve documented limits, one value inside and one outside each, and report which ones the emulator enforces. Write an at-least-once consumer demonstration with a slow consumer and make the consumer idempotent.

View solution

EXERCISE 2

Test SNS fan-out, raw delivery and filter policies (both scopes and twelve operators) against the docs' example message. Write a reference matcher for EventBridge event patterns from the documented operators, build 43 test cases from the docs' examples, and compare the matcher and the emulator. Find out why one emulator case disagrees.

View solution

EXERCISE 3

Run Step Functions Local with a mock configuration and measure Retry and Catch: backoff, `MaxDelaySeconds`, jitter and the `States.Timeout` exception. Mutate a state machine definition 26 times against Step Functions Local and cfn-lint. Write a CloudFormation template for the whole chain, lint it, mutate it 30 times, and write a checker for the Lambda–SQS rules the linter cannot see. Run a partial-batch handler in a Lambda container.

View solution

CHAPTER 9 QUICK REFERENCE

- SQS: visibility timeout default 30 s (0 to 12 h); at-least-once, so make consumers idempotent; dead-letter queue after `maxReceiveCount` receives (set at least 5 for Lambda), with retention longer than the source queue's; long polling up to 20 s; FIFO needs `MessageGroupId`, deduplicates for 5 minutes, and holds back a group's later messages while one is in flight
- SNS: no filter policy means every message; scopes `MessageAttributes` or `MessageBody`; raw delivery to SQS strips the envelope and allows 10 attributes; the queue must allow SNS to send (from my understanding, page not read)
- EventBridge patterns: every pattern field must match, array values are alternatives, an event array matches on any shared element; `prefix`, `suffix`, `anything-but`, `numeric`, `exists`, `cidr`, `wildcard`, `$or`; null is not an empty string; default delivery retry is 24 h and 185 attempts, then dropped unless a dead-letter queue is set
- Step Functions: Standard (up to a year, exactly once, 25,000 history events, billed per transition) versus Express (5 minutes, at least once, no `.sync` or callback); Retry defaults 1 s, 3 attempts, rate 2.0; `States.ALL` alone and last; a caught failure still ends as SUCCEEDED; retries count as transitions
- Lambda from SQS: visibility timeout at least 6 × function timeout + batching window; batch above 10 needs a window of 1 s or more; use `ReportBatchItemFailures` and return `itemIdentifier`s; on FIFO stop at the first failure
- Emulators: moto matched SQS and SNS behaviour but enforced only 6 of 12 SQS limits and failed one EventBridge `time` match; Step Functions Local is "unsupported" but timed retries exactly as documented. Use them for your code, not for limits or validity
- cfn-lint caught 15 of 30 template mistakes: ranges and types yes, relations between resources and the meaning of strings no. Keep a cross-resource check and a deliberate-mistake test of your own

COMING NEXT

Chapter 10 is edge and DNS: CloudFront, Route 53 and WAF. Much of it needs a real account, a real domain or both, so expect a lot of "from the documentation, not run" labels, and some local checks of cache-policy and DNS record logic. Nothing needs starting beforehand. Start with *AWS Intermediate/Advanced 10*.

AWS Intermediate/Advanced

Chapter 10 · Edge and DNS: CloudFront, Route 53 and WAF

AWS Intermediate/Advanced 9 was about what happens behind an API. This chapter is about what happens in front of it: how a visitor's browser finds your site (DNS), how a copy of your content is served from a place near them (a CDN, Amazon CloudFront), and how bad requests are stopped before they reach you (a web application firewall, AWS WAF). It leans on *Web Hosting Essentials* and *HTTPS/TLS Fundamentals* for what DNS records and certificates are, and on *AWS Fundamentals* 4 (S3) and 5 (VPC) for the services behind the edge. API Gateway's own edge options were in *AWS Intermediate/Advanced* 8.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

This is the chapter where an account matters most, and there is none. CloudFront distributions take minutes to deploy, certificates need a domain you control, DNS needs a registered domain and delegated name servers, and none of that exists on the machine that wrote this course. So **nothing was deployed and no real request travelled through CloudFront, Route 53 or WAF.**

- **Run here (level 1):** models I wrote from the documentation and tested on the documentation's own examples (cache duration, cache keys, an S3 bucket policy for origin access control, weighted routing and health-check rules, WAF rule evaluation); *dnspython* 2.9.0 as a real DNS library parsing zone text; the *moto* 5.2.4 emulator (Route 53, WAF, CloudFront, ACM) checked against the docs; cfn-lint 1.57.2 on a CloudFormation template for the whole design; and the IAM evaluation model from *AWS Intermediate/Advanced* 2.
- **From the documentation, not run (level 3):** everything the real services do. The pages were read on 2026-10-13 through a page-fetch tool that summarises pages; several came back thin (the AWS WAF pages gave only an overview until I fetched their sub-pages, and the AWS WAF quotas page came from the AWS General Reference), and each gap is named where it matters.
- **Needs an account, a domain, or both, not run (level 2):** deploying the template, validating a certificate, delegating a domain and sending real requests.

Docker was not needed for this chapter.

A Request's Journey

The picture below is the path this chapter follows. The order of the middle steps (WAF first, then the cache) is how I understand CloudFront and WAF to be combined; the docs pages I read

describe each piece, not the full sequence, so treat the diagram as an orientation aid rather than a quotation.

```
browser
  |  1. DNS: www.example.com ?            Route 53: an alias A/AAAA record answers with
  v
CloudFront edge location
  |  2. AWS WAF web ACL evaluates the request (rules by priority, then the default actio
  |  3. cache behaviour chosen by path; cache key built from the cache policy
  |  4a. hit and still fresh  -> answer from the edge cache
  |  4b. miss or expired      -> origin request (signed by origin access control)
  v
S3 bucket (private)  --  the response is cached for a time set by TTLs and Cache-Control
```

CloudFront

Limits to know

From the CloudFront quotas page (2026-10-13):

WHAT	DEFAULT
Distributions per account; origins per distribution; cache behaviours per distribution	500; 100; 75 (all adjustable)
Alternate domain names (CNAMEs) per distribution	100 (adjustable)
Origin response timeout ("origin read timeout")	1 to 120 seconds
Largest cacheable file per GET response; largest request body	50 GB; 64 GB
Request or origin response length with headers and query strings; URL length	32,768 bytes; 8,192 bytes
Requests per second and data rate per distribution	250,000; 150 Gbps (both can be raised; not for flat-rate plans)
Custom cache policies per account; headers, cookies and query strings per cache policy	20; 10 each
Certificates per distribution; certificates per account on dedicated IPs	1; 2 (no quota with SNI)

WHAT	DEFAULT
Invalidation: paths or tags per second; wildcard invalidations per second	150; 1
CloudFront Functions: per account; size; memory	100; 10 KB; 2 MB
Lambda@Edge: timeout; compressed size; generated response (viewer events / origin events)	30 seconds; 50 MB; 40 KB / 1 MB

Cache keys: what makes two requests the same

From the documentation: a *cache policy* says which headers, cookies and query strings go into the *cache key*, "the unique identifier for every object in the cache". A cache hit happens when a request produces the same key as an earlier one and the stored object is still valid, and "including fewer values in the cache key increases the likelihood of a cache hit". The same policy holds the TTL settings (minimum, default and maximum) and whether compressed objects are cached. The managed policies are fixed bundles of these settings:

```
py cache_tests.py   # managed policies against origin headers
```

```
== managed cache policies against origin headers (seconds CloudFront keeps an object) ==
policy                                          no header        max-age=60
CachingOptimized                                   86,400                60
CachingOptimizedForUncompressedObjects             86,400                60
CachingDisabled                                         0                 0
UseOriginCacheControlHeaders                            0                60
UseOriginCacheControlHeaders-QueryStrings               0                60
Amplify                                                 2                60
Elemental-MediaPackage                             86,400                60
```

The table is the model's result for what each managed policy does with an origin that sends nothing, `max-age=60`, `no-store`, `private`, or a three-year `max-age`. Notice two rows. `CachingOptimized` **(minimum TTL 1 second) keeps a** `no-store` **object for 1 second**, and `Amplify` (minimum 2 seconds) for 2: the docs print a warning for every managed policy with a minimum TTL above 0, because "CloudFront will cache content for at least the duration specified in the cache policy's minimum TTL, even if the `Cache-Control: no-cache`, `no-store`, or `private` directives are present". Only the policies with a minimum TTL of 0 (`CachingDisabled`, the `UseOriginCacheControlHeaders` pair and `Elemental-MediaPackage`) respect them. The

default cache key is small: `CachingOptimized` uses no query strings, no cookies and no headers, only a normalised `Accept-Encoding`.

To see the effect of the key on hit rate, I generated a synthetic log (my own, not real traffic): 20,000 requests to 30 paths with a random marketing query string on 60% of them, a version parameter on 20%, a session cookie from 200 users, and a mix of encodings and countries. For each policy I counted distinct cache keys, which gives a ceiling for the hit ratio (every repeat of a key counts as a hit, ignoring expiry and eviction):

```
py cache_tests.py   # synthetic request log
```

```
== cache key: upper bound of the hit ratio on a synthetic request log ==
policy                                          distinct keys  hit-ratio ceiling
CachingOptimized                                           90             99.6%
custom: only the 'v' query string                         269             98.7%
UseOriginCacheControlHeaders                            7,786             61.1%
UseOriginCacheControlHeaders-QueryStrings              15,458             22.7%
Amplify                                                19,240              3.8%

20,000 requests, 30 paths; 'ceiling' = 1 - distinct keys / requests, i.e. every repeat o

failed checks: 0
```

The same traffic could reach a ceiling of 99.6% with `CachingOptimized` and 3.8% with `Amplify`, because that policy puts every query string, every cookie and the `Authorization` header into the key. A custom policy that keeps only the `v` query string stayed at 98.7%. The lesson from the docs' sentence, now measurable: put in the key only what changes the response, and **check what your marketing and session parameters do to it**.

How long an object stays cached

The expiration page has a table that combines the origin's headers with the policy's minimum, default and maximum TTL. I encoded every row and tested it, using a policy of minimum 0 (then 50), default 100 and maximum 1,000 seconds, so that each branch gives a different number:

```
def edge_ttl(minttl, default, maxttl, cc=None, expires_in=None):
    """seconds CloudFront caches an object. cc: set of Cache-Control directives as {'max
    expires_in: seconds until the Expires header's date (only used when there is no max-
    cc = cc or {}
```

```
    blockers = any(cc.get(k) for k in ("no-cache", "no-store", "private"))
    if blockers:
        # table row: Min TTL = 0 -> CloudFront respects the headers (not cached); Min TT
        return minttl if minttl > 0 else 0
    if "s-maxage" in cc:
        v = cc["s-maxage"]                          # table: with both max-age and s-maxa
        return min(v, maxttl) if minttl == 0 else clamp(v, minttl, maxttl)
    if "max-age" in cc:
        v = cc["max-age"]
        return min(v, maxttl) if minttl == 0 else clamp(v, minttl, maxttl)
    if expires_in is not None:
        return min(expires_in, maxttl) if minttl == 0 else clamp(expires_in, minttl, max
    return default if minttl == 0 else max(minttl, default)
```

```
py cache_tests.py   # expiration table

== the expiration table (min 0 or >0; origin headers) ==
ok   min 0, max-age 500                                          model
ok   min 0, max-age 5000 (above max)                             model
ok   min 0, no header                                            model
ok   min 0, max-age 500 and s-maxage 200                         model
ok   min 0, s-maxage 5000                                        model
ok   min 0, Expires in 300 s                                     model
ok   min 0, Expires in 9000 s                                    model
ok   min 0, no-store                                             model
ok   min 0, private                                              model
ok   min 0, no-cache with max-age 500                            model
ok   min 50, max-age 500                                         model
ok   min 50, max-age 10                                          model
ok   min 50, max-age 5000                                        model
ok   min 50, no header, default 100                              model
ok   min 500, no header, default 100                             model
ok   min 50, s-maxage 10 (max-age 500)                           model
ok   min 50, Expires in 10 s                                     model
ok   min 50, Expires in 9000 s                                   model
ok   min 50, no-store                                            model
```

All 19 cases agree with the page. They are checks of my reading of the table (I wrote both the model and the expected values from the same page), so they show the model is faithful to the page, not that the service behaves that way. Other facts from the same page: with no cache

policy the default TTL is 24 hours; "if you specify values both for `Cache-Control max-age` and for `Expires`, CloudFront uses only the value of `Cache-Control max-age`"; a viewer's `Cache-Control` or `Pragma` header cannot force a trip to the origin; and CloudFront may evict a rarely used object before it expires.

Stale content and errors

Stale directives, from the docs. `stale-while-revalidate` lets CloudFront answer from an expired object while it fetches a fresh one; `stale-if-error` keeps serving an expired object when the origin fails or returns a 5xx error. In both cases CloudFront serves stale content "up to the value of the directive or the maximum TTL, whichever is less". The page does not say which moment that period is measured from, so my model computes only its length.

Error caching. CloudFront caches an error from the origin for the larger of the *error caching minimum TTL* (10 seconds by default) and the origin's `max-age` or `s-maxage`. It always caches 404, 414, 500, 501, 502, 503 and 504, and caches 400, 403, 405, 412 and 415 only when the origin sends one of those two directives, and only for `GET` and `HEAD` (`OPTIONS` optionally). If an object is in the cache but expired and the origin returns a 5xx error, CloudFront keeps serving the old object for the error caching minimum TTL; a 4xx is passed through instead of the object. The tests for these rules are in the output below: all agree.

```
py cache_tests.py   # stale, errors, invalidation
```

```
== stale directives and errors (page examples) ==
ok   stale-while-revalidate=600, max TTL 31,536,000                  model
ok   stale-if-error=86400, max TTL 3600                              model
ok   404, no headers                                                 model
ok   404 with max-age=60                                             model
ok   503, error min TTL 30, max-age 5                                model
ok   403 without a header                                            model
ok   403 with s-maxage=20                                            model
ok   500 for a POST                                                  model
ok   418 (not in either list)                                        model

== invalidation (page examples) ==
ok   3 distributions x 600 paths                                     model
ok   500 path + 600 tag invalidations                                model
ok   a single /* wildcard path                                       model
```

Invalidation or versioned names?

From the documentation: to replace a cached file you can invalidate it or serve a new name. The docs recommend file versioning if you update often: it works even when a browser or proxy cached the old file, it shows up in access logs, it lets you roll forward and back, and it is cheaper, because "you still have to pay for CloudFront to transfer new versions of your files to edge locations, but you don't have to pay for invalidating files". The first 1,000 invalidation paths a month are free across the account; a wildcard path such as `/images/*` counts as one, "even if it causes CloudFront to invalidate thousands of files", and tag invalidations count against the same 1,000. The page text I read did not give the price of a path above the free 1,000. Quotas: 150 paths or tags per second, and 1 wildcard invalidation per second. The model reproduces the docs' two worked examples (3 distributions × 600 paths = 800 charged; 500 + 600 = 100 charged).

A private S3 origin with origin access control

From the documentation: origin access control (OAC) lets CloudFront send *signed* requests to an S3 bucket so that the bucket can stay private. It is preferred over the older origin access identity (OAI) because it supports all Regions, SSE-KMS and `PUT`/`DELETE`. The bucket policy names the CloudFront service principal and uses a condition so that only *your* distribution is allowed: "Use a `Condition` element in the policy to allow CloudFront to access the bucket only when the request is on behalf of the CloudFront distribution that contains the S3 origin". Caveats on the page: Object Ownership must be "Bucket owner enforced" (the default for new buckets); a bucket configured as a *website endpoint* cannot use OAC at all (it is a custom origin); with the recommended signing setting `always` the S3 connection is always HTTPS; with `never` the bucket has to be publicly readable; for SSE-KMS the KMS key policy needs its own statement for CloudFront.

Run here: I evaluated the page's own statements with the IAM model from chapter 2. A mistake of mine surfaced first: the model crashed when a *service* requester was compared with a statement that names an AWS principal (a service has no ARN), instead of answering "no". I fixed it in this chapter's copy of the model. Results:

```
  SiteBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref SiteBucket
      PolicyDocument:
        Version: "2012-10-17"
```

```
        Statement:
          - Sid: AllowCloudFrontServicePrincipalReadOnly
            Effect: Allow
            Principal: {Service: cloudfront.amazonaws.com}
            Action: s3:GetObject
            Resource: !Sub "${SiteBucket.Arn}/*"
            Condition:
              StringEquals:
                "AWS:SourceArn": !Sub "arn:aws:cloudfront::${AWS::AccountId}:distributio

  OriginAccess:
    Type: AWS::CloudFront::OriginAccessControl
    Properties:
      OriginAccessControlConfig:
        Name: site-oac
        OriginAccessControlOriginType: s3
        SigningBehavior: always
        SigningProtocol: sigv4
```

```
py oac_policy.py
```

```
case                                                          expect  model
docs read-only policy: CloudFront, its own distribution, GetObject True    True   resour
same policy: CloudFront request with no SourceArn             False   False  no ide
same policy: PutObject (read-only statement)                  False   False  no ide
same policy: an unrelated user in another account             False   False  no ide
mistake: the Condition left out, another distribution         True    True   resour
mistake: Resource is the bucket, not bucket/*                 False   False  no ide
migration policy (OAC + legacy OAI): the OAI user             True    True   resour
migration policy: CloudFront with OAC                         True    True   resour
KMS key policy from the page: Decrypt for the distribution    True    True   resour
KMS key policy: Decrypt for another distribution              False   False  this r
KMS key policy: ScheduleKeyDeletion (not listed)              False   False  this r

12 of 12 cases as expected from the page
```

The model agrees with the page's intent in all 12 cases, including the two that matter: **without the `AWS:SourceArn` condition, CloudFront acting for *someone else's* distribution could read your bucket** (the "confused deputy" problem the condition exists to stop), and a `Resource` of

the bucket instead of `bucket/*` grants nothing for objects. This is a model of documented evaluation rules (*AWS Intermediate/Advanced* 2), not S3.

HTTPS and certificates

From the documentation: for a certificate from AWS Certificate Manager "make sure you request (or import) the certificate in the US East (N. Virginia) Region (`us-east-1`)", wherever your site's other resources are. When you add an alternate domain name, CloudFront checks that the certificate covers it: the SAN field must contain "an exact match for the alternate domain name, or contain a wildcard at the same level". CloudFront supports RSA keys of 1,024, 2,048, 3,072 and 4,096 bits and ECDSA 256- and 384-bit keys, but ACM issues RSA certificates only up to 2,048 bits (longer ones must be imported). A third-party certificate must be renewed or re-imported at least 24 hours before it expires, and CloudFront can take up to 24 hours to show the change; ACM-issued certificates renew themselves. If an origin's certificate has no name matching the origin domain name, CloudFront returns `502` . SNI is the default way to serve HTTPS; dedicated IP addresses are limited to 2 certificates per account.

Price

The CloudFront pricing page text I read had no pay-as-you-go prices (data transfer and request prices were on a separate page I did not get). It listed **flat-rate plans**, one monthly price per distribution with no overage charges: Free $0 (1 million requests, 100 GB), Pro $15 (10 million requests, 50 TB), Business $200 (125 million, 50 TB) and Premium $1,000 (500 million, 50 TB). The WAF page says these plans bundle CloudFront, AWS WAF and DDoS protection, bot management, Route 53 DNS, CloudWatch Logs ingestion, a TLS certificate, edge compute and S3 storage credits, and that a web ACL must stay associated while on a plan. How a plan treats a web ACL's own charges I did not read.

Route 53

Alias records, and why the apex is special

From the documentation: a CNAME says "this name is another name". "You can't create a CNAME record for example.com" (the *zone apex*), but "you can create an alias record" there. An *alias record* is a Route 53 extension: it points at selected AWS resources (CloudFront distributions, load balancers, API Gateway APIs, S3 website buckets, another record in the same zone and others), is listed in `dig` as an ordinary A or AAAA record, responds only when the query's name and type match, tracks changes in the target by itself, and you cannot set its TTL

for an AWS resource. Route 53 does not charge for alias queries to the listed AWS services; a CNAME query is charged, and a CNAME pointing at another record in a Route 53 zone counts as two.

The apex rule is not a Route 53 quirk but a DNS rule. **Run here:** dnspython, a real DNS library, parsing zone files:

```
py route53_model.py   # dnspython part
```

```
== zone text parsed by dnspython 2.9.0 ==
  a plain zone                                          accepted
  CNAME at the apex (@ IN CNAME www.example.com.)       rejected: CNAMEAndOtherData: CN
  CNAME next to an A record at the same name            rejected: CNAMEAndOtherData: rd
  CNAME at a subdomain on its own                       accepted
  two CNAMEs for one name                               accepted
  A record at the apex                                  accepted
```

The parser refuses a CNAME next to the apex's SOA and NS records, and a CNAME next to an A record on any name, and accepts a lone CNAME on a subdomain. That is the problem an alias record solves for `example.com` itself. (It also accepted two CNAMEs for one name, which I did not expect; what Route 53 does with that I do not know.)

Routing policies

The docs list eight: *simple* (one resource), *failover* (active-passive), *geolocation* (by user location), *geoproximity* (by resource location, with a bias), *latency* (best Region), *IP-based* (by the client's IP range), *multivalue answer* (up to eight healthy records at random) and *weighted* (traffic in proportions). Records with the same name and type under weighted, latency, geolocation, multivalue and IP-based routing are limited to 100 per name and type (geoproximity: 30).

Weighted routing, from the docs: a record's share is "weight for a specified record / sum of the weights for all records", so weights of 1 and 255 give 1/256 and 255/256 (a 0.4% canary); a weight of 0 stops traffic to a record. With health checks, "Route 53 initially considers only the nonzero weighted records"; if all of those are unhealthy, it considers the zero-weight ones. The page prints two tables, with and without a health check on a zero-weight record. I turned every row into a test:

```
def answer(records):
    """records: list of dicts {name, weight, hc: None|True|False}  (hc = has a health ch
    Returns the names Route 53 may answer with. Derived from the docs' tables: nonzero w
    healthy, zero-weight records; if nothing is healthy at all, the nonzero records are
    ok = lambda r: r["hc"] is None or r["hc"] is True
    nonzero = [r for r in records if r["weight"] > 0]
    zero = [r for r in records if r["weight"] == 0]
    a = [r for r in nonzero if ok(r)]
    if a: return [r["name"] for r in a]
    b = [r for r in zero if ok(r)]
    if b: return [r["name"] for r in b]
    return [r["name"] for r in nonzero]
```

```
py route53_model.py   # weighted routing
```

```
== zero-weight record WITH a health check (page table) ==
ok   health R1,R2,R3 = UUH  answered ['R3']  page: ['R3']
ok   health R1,R2,R3 = UUU  answered ['R1', 'R2']  page: ['R1', 'R2']
ok   health R1,R2,R3 = UHU  answered ['R2']  page: ['R2']
ok   health R1,R2,R3 = HHU  answered ['R1', 'R2']  page: ['R1', 'R2']
ok   health R1,R2,R3 = HHH  answered ['R1', 'R2']  page: ['R1', 'R2']
== zero-weight record WITHOUT a health check (page table) ==
ok   health R1,R2 = HH, R3 unchecked  answered ['R1', 'R2']  page: ['R1', 'R2']
ok   health R1,R2 = UU, R3 unchecked  answered ['R3']  page: ['R3']
ok   health R1,R2 = UH, R3 unchecked  answered ['R2']  page: ['R2']

== weights 1 and 255 (page example: 1/256 and 255/256) ==
model shares: {'canary': '0.00391', 'stable': '0.99609'}  1/256 = 0.00391
200,000 simulated queries: {'stable': 199200, 'canary': 800}  canary share 0.00400
weight 0 on a healthy record: {'b': 1.0}
```

All eight rows match, including the odd one: when *every* record is unhealthy, the page's table shows Route 53 still answering with the nonzero records (`U,U,U` → records 1 and 2) rather than returning nothing. This model's rule was inferred from those tables, so it is as good as the tables, and no better.

Health checks

From the documentation: checkers in many locations probe the endpoint every 10 or 30 seconds; Route 53 aggregates the answers, and "if more than 18% of health checkers report that

an endpoint is healthy, Route 53 considers it healthy". An HTTP or HTTPS check needs a TCP connection within 4 seconds and a 2xx or 3xx status within 2 seconds after that; a TCP check allows 10 seconds; string matching must find the text in the first 5,120 bytes of the body. **HTTPS health checks do not validate the certificate**, so an expired certificate does not fail them. A new health check counts as healthy "until there's enough data". A calculated health check can combine up to 255 child checks. The page does not say how many checkers there are, so I tabulated what 18% means for a few possible counts:

```
py route53_model.py   # the 18% rule
```

```
== 'more than 18% of health checkers report healthy' ==
the page does not say how many checkers there are; for a few possible counts, the number
   5 checkers: healthy once 1 of them report healthy (20%); 0 of 5 (0%) is unhealthy
  10 checkers: healthy once 2 of them report healthy (20%); 1 of 10 (10%) is unhealthy
  15 checkers: healthy once 3 of them report healthy (20%); 2 of 15 (13%) is unhealthy
  16 checkers: healthy once 3 of them report healthy (19%); 2 of 16 (12%) is unhealthy
  20 checkers: healthy once 4 of them report healthy (20%); 3 of 20 (15%) is unhealthy
```

The point of the odd threshold, per the page, is that a few locations cut off from the endpoint by network problems do not make it look dead. A related effect I did not test: answers cached by resolvers keep being used until their TTL runs out, so the time before visitors move to a healthy record is the detection time *plus* the record's TTL (my reasoning, not a page I read); an alias to an AWS resource uses that resource's TTL, which you cannot set.

Quotas and price

Quotas (2026-10-13): hosted zones, initial 500 per account; records, 10,000 per hosted zone (more costs extra); 400 records in one record set; health checks, 200 active per account; domains, 20; a `ChangeResourceRecordSets` request, at most 1,000 record elements (an `UPSERT` counts each twice) and 32,000 characters of values. Prices from the pricing page (it does not tie the standard rates to a Region): $0.50 per hosted zone per month for the first 25 and $0.10 after; standard queries $0.40 per million for the first billion; latency queries $0.60; geolocation and geoproximity $0.70; IP-based $0.80; health checks $0.50 (AWS endpoints) or $0.75 (others) plus $1.00 or $2.00 for each optional feature such as HTTPS or string matching.

AWS WAF

From the documentation: a web ACL holds rules (and rule groups, which hold rules). Each rule has a priority that must be unique "within that protection pack (web ACL)", and "inside any rule group" its own unique priorities; evaluation goes "from the lowest numeric priority setting on up until it either finds a match that terminates the evaluation or exhausts all of the rules". *Allow* and *Block* terminate; *Count* does not; CAPTCHA and Challenge terminate only when the request lacks a valid token. "If the rule evaluation doesn't result in any terminating action, then AWS WAF applies the protection pack (web ACL) default action". The page's example: an Allow rule at priority 50 and a Count rule at priority 100 means requests that match both are never counted, and "if you want count metrics from the count rule even for requests that match the allow rule, you'd need to give the count rule a lower numeric priority".

Rate-based rules, from the docs: they count requests per aggregation instance (for example per IP) over an evaluation window of 60, 120, 300 (default) or 600 seconds; the lowest limit is 10; the action can be anything except Allow; a limit is applied "near the limit that you set, but does not guarantee an exact limit match". A rate-based rule costs 2 WCUs plus 30 for each custom aggregation key. A web ACL is priced for up to 1,500 WCUs, can use up to 5,000, and a web ACL can have 10 rate-based statements (4 per rule group).

Run here: a model of the evaluation order, actions, default action, validation rules and a rate-based rule, tested on the page's own priority example (RuleGroupA with rules at 10,000 and 20,000 between Rule1 at 0 and Rule2 at 200, and so on) and on the Allow/Count example:

```
py waf_model.py
```

```
== the page's priority example ==
ok   evaluation order                                                ['Rule1'

== Allow (priority 50) and Count (priority 100), the page's example ==
ok   Allow before Count: results, count-rule hits
ok   Count before Allow (lower number): results, count-rule hits

== actions and the default action ==
ok   only a Count rule matches -> default action
ok   Count then Allow: /health
ok   Count then Allow: anything else (default Block)

== validation rules from the pages ==
ok   duplicate priorities in a web ACL
ok   rate limit 5
ok   evaluation window 90
```

```
  ok   rate-based rule with Allow

  == rate-based rule: limit 10 per 60 s per IP, 3 clients ==
     198.51.100.1   allowed   10  blocked  110
     198.51.100.2   allowed   10  blocked  350
     198.51.100.3   allowed   12  blocked    0
     (1 request/s = 60 per window, above the limit of 10; the page says real counting is '

  == capacity and price ==
  ok   rate-based rule, no custom keys
  ok   rate-based rule with 2 custom aggregation keys
  ok   5 rules/groups, 10 million requests a month: $
  ok   1 web ACL, 3 rule groups + 2 rules, 100 million requests: $

  failed checks: 0
```

The evaluation order matches the page, and the two priority arrangements give 1 and 2 count hits as the page predicts. In the rate-based demonstration the client sending one request a second was allowed 10 and then blocked for the rest; **my model counts a blocked request toward the rate, so a client that keeps sending stays blocked**. I did not read the page on how real rate limiting treats blocked requests, so do not take that behaviour from the model.

Limits and price, from the docs: 100 web ACLs per account for each of CloudFront and regional scope; 100 rule groups; 100 IP sets of up to 10,000 addresses; regex patterns of up to 200 characters; 100,000 requests per second per web ACL (adjustable). Request bodies are inspected only up to **64 KB for CloudFront, API Gateway, Cognito, App Runner and Verified Access, but 8 KB for Application Load Balancer and AppSync**, so anything beyond that limit is not inspected. Prices (from the page's examples, no Region table): $5.00 per web ACL per month, $1.00 per rule (each rule group counts), $0.60 per million requests, and extra for more than 1,500 WCUs, for body inspection beyond the default and for CAPTCHA. Which Region a web ACL for CloudFront must be created in was not on a page I read.

Can the Emulator Be Trusted Here?

The outline's rule is to use an emulator only where it is shown to be adequate. For these four services it is not. I sent moto inputs that the pages say are invalid, and a few that are valid:

```
py edge_moto_tests.py   # condensed
```

```
case                                                          docs say  moto 5.2.4
CNAME record at the zone apex                                 rejected  accepted   DIFF
CNAME record at a subdomain                                   accepted  accepted   ok
alias A record at the apex, CloudFront target                 accepted  accepted   ok
alias record that also sets a TTL                             rejected  accepted   DIFF
health check with a 20 s request interval                     rejected  accepted   DIFF
health check with a 30 s request interval                     accepted  accepted   ok
weighted record without a SetIdentifier                       rejected  accepted   DIFF
weighted record with a SetIdentifier                          accepted  accepted   ok
web ACL with a rate-based rule, limit 100, window 300         accepted  accepted   ok
rate-based limit 5                                            rejected  rejected by bot
rate-based window 90                                          rejected  accepted   DIFF
rate-based rule with the Allow action                         rejected  accepted   DIFF
two rules with the same priority                              rejected  accepted   DIFF
two rules with different priorities                           accepted  accepted   ok
distribution with a certificate requested in us-east-1        accepted  accepted   ok
distribution with a certificate requested in eu-west-1        rejected  accepted   DIFF
distribution with 101 alternate domain names                  rejected  accepted   DIFF
cache policy with 10 headers in the cache key                 accepted  call not implem
cache policy with 11 headers in the cache key                 rejected  call not implem
7 of 16 scored cases agree with the documentation (n/a = not scored: the emulator does n
```

moto accepted every one of the nine invalid inputs I could score, including a CNAME at the apex, a 20-second health check interval, a rate-based rule with the Allow action, duplicate rule priorities, a certificate from `eu-west-1` on a distribution, and 101 alternate domain names. It accepted the valid ones too, so its answers carry no information about validity. Two more notes: moto does not implement the CloudFront cache-policy calls (the server returned 404), and its rate-limit rejection came from boto3's own parameter checking, not from the emulator, so I left both out of the score. My first run counted those as emulator rejections, which would have made moto look better than it is. Emulator-based tests on these services would only prove that your code can call the API.

The Whole Design as a Template

Run here: one CloudFormation template with a private bucket and its policy, the OAC, a cache policy, a web ACL, the distribution, the alias records, a health check and two weighted records:

```
SiteCachePolicy:
  Type: AWS::CloudFront::CachePolicy
  Properties:
    CachePolicyConfig:
      Name: site-cache-policy
      MinTTL: 1
      DefaultTTL: 86400
      MaxTTL: 31536000
      ParametersInCacheKeyAndForwardedToOrigin:
        EnableAcceptEncodingGzip: true
        EnableAcceptEncodingBrotli: true
        HeadersConfig: {HeaderBehavior: none}
        CookiesConfig: {CookieBehavior: none}
        QueryStringsConfig: {QueryStringBehavior: none}
```

```
Distribution:
  Type: AWS::CloudFront::Distribution
  Properties:
    DistributionConfig:
      Enabled: true
      Comment: chapter 10 site
      Aliases: [!Ref DomainName]
      DefaultRootObject: index.html
      HttpVersion: http2and3
      PriceClass: PriceClass_100
      WebACLId: !GetAtt WebAcl.Arn
      Origins:
        - Id: site
          DomainName: !GetAtt SiteBucket.RegionalDomainName
          OriginAccessControlId: !GetAtt OriginAccess.Id
          S3OriginConfig: {OriginAccessIdentity: ""}
      DefaultCacheBehavior:
        TargetOriginId: site
        ViewerProtocolPolicy: redirect-to-https
        Compress: true
        CachePolicyId: !Ref SiteCachePolicy
      CustomErrorResponses:
        - ErrorCode: 404
          ResponseCode: 404
          ResponsePagePath: /404.html
          ErrorCachingMinTTL: 30
      ViewerCertificate:
        AcmCertificateArn: !Ref CertificateArn
```

```
          SslSupportMethod: sni-only
          MinimumProtocolVersion: TLSv1.2_2021
```

It linted clean with cfn-lint 1.57.2. Then I changed one thing at a time (32 changes, each labelled with the page behind the expectation):

```
py mutate_edge.py   # condensed
```

```
change                                                  cfn-lint 1.57.2
(control) template as written                           exit 0
cache policy MinTTL 1 -> -1                             caught E3034
cache policy MinTTL above MaxTTL                        missed
cache policy DefaultTTL 'a day'                         caught E3012
headers 'whitelist' with no header list                 missed
ViewerProtocolPolicy 'https'                            caught E3030
PriceClass 'PriceClass_50'                              caught E3030
HttpVersion 'http4'                                     caught E3030
OAC SigningBehavior 'sometimes'                         caught E3031
OAC origin type 'bucket'                                caught E3031
OAC with signing 'never' on a private bucket            missed
bucket policy without the SourceArn Condition           missed
bucket policy Resource is the bucket, not bucket/*      missed
bucket policy trusts the wrong service                  missed
certificate in eu-west-1 instead of us-east-1           missed
alias domain with no certificate                        warning W2001
custom error ErrorCachingMinTTL -5                      missed
WebACL Scope REGIONAL while attached to CloudFront      missed
WebACL without a DefaultAction                          caught E3003
rate-based Limit 1000 -> 5                              caught E3034
rate-based EvaluationWindowSec 300 -> 90                caught E3030
rate-based rule with Action Allow                       missed
two rules with the same Priority                        missed
a rule without VisibilityConfig                         caught E3003
managed rule group with a vendor that does not exist    missed
apex record becomes a CNAME                             missed
alias record that also sets a TTL                       caught E3029
alias hosted zone ID is not CloudFront's                missed
health check interval 20 s                              missed
health check FailureThreshold 'three'                   caught E3012
health check type 'HTTP3'                               caught E3030
weighted record without a SetIdentifier                 missed
```

```
weighted Weight 'heavy'                                    caught E3012
32 one-change mutations: cfn-lint reported an error for 15; 17 passed or only warned
```

cfn-lint reported an error for 15 of 32. It checks types, ranges and enumerations: a negative TTL, a protocol policy that does not exist, a rate limit of 5, a window of 90, a missing default action or visibility config, an alias record with a TTL. It misses everything that depends on meaning: a minimum TTL above the maximum, a bucket policy with no source condition or the wrong resource or service, a certificate from another Region, an apex CNAME, a web ACL with the wrong scope, a rate-based rule that allows, duplicate rule priorities, a health check interval of 20 seconds, and a weighted record without a set identifier. (The warning it printed for the distribution with no certificate was about an unused parameter, not the real problem, so I did not count it as a catch.)

Run here: I then wrote a checker for those cross-property rules from the pages above and ran both tools on all 32 changes:

```
def check(text):
    t = yaml.load(text, Loader=Loader)
    R, P = t["Resources"], t.get("Parameters", {})
    out = []
    by_type = lambda ty: [(k, v) for k, v in R.items() if v["Type"] == ty]

    # cache policies
    for k, r in by_type("AWS::CloudFront::CachePolicy"):
        c = r["Properties"]["CachePolicyConfig"]
        if isinstance(c.get("MinTTL"), (int, float)) and isinstance(c.get("MaxTTL"), (in
            out.append(("ERROR", k, f"MinTTL {c['MinTTL']} is above MaxTTL {c['MaxTTL']}
        h = c.get("ParametersInCacheKeyAndForwardedToOrigin", {}).get("HeadersConfig", {
        if h.get("HeaderBehavior") == "whitelist" and not h.get("Headers"):
            out.append(("ERROR", k, "HeaderBehavior whitelist with no Headers list (unde
    # OAC and the bucket policy
    for k, r in by_type("AWS::CloudFront::OriginAccessControl"):
        sb = r["Properties"]["OriginAccessControlConfig"].get("SigningBehavior")
        if sb in ("never", "no-override"):
            out.append(("WARN", k, f"SigningBehavior {sb}: with 'never' the bucket must
    for k, r in by_type("AWS::S3::BucketPolicy"):
        for s in r["Properties"]["PolicyDocument"]["Statement"]:
            p = s.get("Principal", {})
            if p.get("Service") == "cloudfront.amazonaws.com":
```

```
                if not s.get("Condition") or "AWS:SourceArn" not in str(s["Condition"]):
                    out.append(("ERROR", k, "CloudFront service principal without an AWS
```

```
py mutate_edge2.py   # both tools
```

```
edgecheck on the unchanged template: no findings
change                                                  cfn-lint   edgecheck
cache policy MinTTL 1 -> -1                             error      -
cache policy MinTTL above MaxTTL                        -          finding
cache policy DefaultTTL 'a day'                         error      -
headers 'whitelist' with no header list                 -          finding
ViewerProtocolPolicy 'https'                            error      -
PriceClass 'PriceClass_50'                              error      -
HttpVersion 'http4'                                     error      -
OAC SigningBehavior 'sometimes'                         error      -
OAC origin type 'bucket'                                error      -
OAC with signing 'never' on a private bucket            -          finding
bucket policy without the SourceArn Condition           -          finding
bucket policy Resource is the bucket, not bucket/*      -          finding
bucket policy trusts the wrong service                  -          finding
certificate in eu-west-1 instead of us-east-1           -          finding
alias domain with no certificate                        -          finding
custom error ErrorCachingMinTTL -5                      -          finding
WebACL Scope REGIONAL while attached to CloudFront      -          finding
WebACL without a DefaultAction                          error      -
rate-based Limit 1000 -> 5                              error      finding
rate-based EvaluationWindowSec 300 -> 90                error      finding
rate-based rule with Action Allow                       -          finding
two rules with the same Priority                        -          finding
a rule without VisibilityConfig                         error      -
managed rule group with a vendor that does not exist    -          finding
apex record becomes a CNAME                             -          finding
alias record that also sets a TTL                       error      finding
alias hosted zone ID is not CloudFront's                -          finding
health check interval 20 s                              -          finding
health check FailureThreshold 'three'                   error      -
health check type 'HTTP3'                               error      -
weighted record without a SetIdentifier                 -          finding
weighted Weight 'heavy'                                 error      -

32 changes: both tools 3; cfn-lint only 12; edgecheck only 17; neither 0
neither: []
```

Together they covered all 32 (cfn-lint alone 15, the checker alone 20; both on 3). **One caution that matters:** I wrote the checker's rules *after* I saw which changes cfn-lint missed, so "neither: 0" shows that these rules can be written, not that the checker would catch a mistake it was not written for. Two rules in it are my own understanding rather than something I read (a CloudFront web ACL needs the `CLOUDFRONT` scope, and CloudFront's alias hosted zone ID `Z2FDTNDATAQYW2`), and it is tagged that way in the code.

What a Real Run Involves (not run)

Level 2 and 3: needs an account and a domain. In outline, and partly from my own understanding of the process rather than pages I read this session: you need a domain whose name servers point at a Route 53 hosted zone (or another way to put records in the DNS); you request the certificate in `us-east-1` and prove you control the name (DNS validation adds a record); you deploy the template and wait for the distribution to finish deploying; then you check with `curl -I` that a second request is a hit (CloudFront reports it in a response header), that the S3 URL itself is refused, and with `dig` that the records resolve. The nameserver delegation steps and the exact response headers were not on pages I read, so I give no commands. Set a budget alert first (*AWS Fundamentals* 9), keep any test short and small, and delete the stack afterwards; a web ACL and a hosted zone bill by the month and the page says hosted-zone fees "are not prorated for partial months".

The priced lines I could read, for an example of a 20-million-query API record, a HTTPS health check, one web ACL with two rules and 10 million requests:

```
py cost10.py
```

```
== lines I can price (per month) ==
Route 53 hosted zone                                $   0.50
Route 53 standard queries, 20 million (api records)       $   8.00
Route 53 alias queries for the site (A and AAAA)      $   0.00   (free for CloudFront t
Route 53 health check, HTTPS, endpoint outside AWS    $   2.75   (0.75 basic + 2.00 HTT
Route 53 health check, HTTPS, endpoint inside AWS     $   1.50   (0.50 basic + 1.00 HTT
WAF: 1 web ACL + 2 rules + 10M requests                     $  13.00   (5.00 + 2.00 + 6
                                                    -------
sum of the priced lines (CloudFront delivery not included)  $  24.25

== CloudFront flat-rate plans (pricing page; one price per distribution, no overage charg
  Free      $  0.00 a month      1M requests   100 GB data transfer
```

```
  Pro       $ 15.00 a month     10M requests    50 TB data transfer
  Business  $200.00 a month    125M requests    50 TB data transfer
  Premium   $1,000.00 a month    500M requests    50 TB data transfer
The WAF page lists what the plans include: CloudFront, AWS WAF and DDoS protection, bot
CloudWatch Logs ingestion, a TLS certificate, serverless edge compute and S3 storage cre
I did not read the plan page, so I cannot say whether the WAF and Route 53 lines above w
plan treats a web ACL's own charges, health checks or weighted records. Check that befor

== invalidations ==
first 1,000 paths a month are free across the account; a /* wildcard is one path; the pr
```

Cost and Clean-Up

Nothing in this chapter created anything in AWS. On this machine: the scratch folder `C:/lwaws/c10` (a Python environment and test files) and the moto server, which I stopped. No Docker containers or images were used.

WHAT WAS AND WAS NOT CHECKED

Run here: a cache duration model (19 table cases, 9 stale and error cases, 3 invalidation cases) and a cache-key experiment on a synthetic log; 12 origin-access-control policy cases; 8 weighted-routing rows and the 18% rule; dnspython on 6 zone files; a WAF evaluation model (priority order, actions, default action, 4 validation rules, a rate-based demonstration); 16 scored emulator checks (and 3 unscored); 32 template mutations against cfn-lint and a cross-property checker; a cost sketch. **From the documentation, not run:** everything the real services do. **Not read:** CloudFront pay-as-you-go prices, the price of invalidation paths above the free 1,000, the flat-rate plan details, CloudFront Functions and Lambda@Edge restrictions, signed URLs and cookies, field-level encryption, continuous deployment, origin groups and failover, Route 53 alias target values and failover and latency routing details, DNSSEC, query logging, resolver features, how real rate limiting counts blocked requests, managed rule group contents, Shield, which Region a CloudFront web ACL must live in, how long a distribution takes to deploy. **Not checked at all:** any real request through any of these services.

Practice

EXERCISE 1

Write a model of CloudFront cache duration from the documentation's expiration table and test every cell, plus the stale-content, error-caching and invalidation-counting rules. Print what each managed

cache policy does with five origin header scenarios. Measure the hit-ratio ceiling of five cache policies on a synthetic request log. Evaluate the documentation's S3 bucket policy for origin access control against twelve requests, including the confused-deputy case.

View solution

EXERCISE 2

Model Route 53 weighted routing with health checks and test it on the two tables on the documentation page; work out what the 18% rule means for several checker counts; parse zone files with dnspython to see which CNAME placements a DNS parser rejects. Model AWS WAF evaluation (priority order, Allow/Count/Block, default action, a rate-based rule) and test it on the documentation's examples.

View solution

EXERCISE 3

Check an emulator against the documentation on Route 53, WAF, CloudFront and ACM (separate rejections by the emulator from rejections by the client library and from calls it does not implement). Write a CloudFormation template for the edge design, lint it, make 32 one-change mutations, write a checker for the cross-property rules the linter misses, and say why its score is not independent evidence. Sketch the monthly cost from the pricing pages and list what you could not read.

View solution

CHAPTER 10 QUICK REFERENCE

- CloudFront cache key = what a cache policy includes (headers, cookies, query strings, normalised encoding). Fewer values means more hits: on my synthetic log, ceilings of 99.6% (nothing in the key) against 3.8% (all query strings, all cookies, Authorization)
- Cache duration: origin `max-age` (or `s-maxage` for CloudFront) clamped by the policy's minimum and maximum TTL; no header means the default TTL (or the larger of minimum and default); a minimum TTL above 0 overrides `no-store` and `private` (`CachingOptimized` keeps them for 1 second)
- Errors are cached for at least 10 seconds by default (404, 414, 5xx always; 400, 403, 405, 412, 415 only with `max-age` / `s-maxage`); update content with versioned names rather than invalidation; the first 1,000 invalidation paths a month are free and a wildcard is one path

- Private S3 origin: OAC plus a bucket policy for `cloudfront.amazonaws.com` with an `AWS:SourceArn` condition for your distribution and `Resource` of `bucket/*`; website endpoints cannot use OAC
- CloudFront certificates from ACM must be in `us-east-1`; the certificate must cover every alternate domain name; renew imported certificates 24 hours early
- Route 53: no CNAME at the zone apex, use an alias A/AAAA (free queries to AWS targets, no TTL to set); weighted share = weight / sum of weights; health checks are healthy above 18% of checkers, probe every 10 or 30 s, and do not validate HTTPS certificates; answers cached by resolvers add the record's TTL to failover time (my reasoning)
- WAF: lowest priority first; Allow and Block terminate, Count does not; no terminating match means the default action; rate limit at least 10, window 60/120/300/600, any action but Allow; bodies inspected up to 64 KB for CloudFront but 8 KB for an ALB
- moto accepted all 9 invalid inputs on these services and does not implement CloudFront cache policies; cfn-lint caught 15 of 32 template mistakes (types, ranges, enumerations), not meaning. Keep your own cross-property checks, and test them on mistakes you did not write them for

COMING NEXT

Chapter 11 is delivery and day-2 operations: pipelines (CodePipeline, CodeBuild and CodeDeploy against GitHub Actions with OIDC to assume a role), deployment strategies, Systems Manager, CloudTrail, Config and the security services. Local checks will focus on workflow files, trust policies for OIDC and deployment-strategy arithmetic; a Docker-based check of workflows may be possible and Docker is working. Start with *AWS Intermediate/Advanced 11*.

AWS Intermediate/Advanced

Chapter 11 · Delivery and Operations: CI/CD and Day-2 Operations

The earlier chapters described what to build. This one is about how a change gets from a commit to AWS without a person's access keys in the middle, how a new version is introduced without taking everyone with it when it is wrong, and how you find out afterwards who did what and whether anything has drifted. It leans on *CI/CD Pipelines* for the general ideas (stages, artifacts, approvals), on *Observability* for alarms and logs, on *AWS Intermediate/Advanced* 2 for the IAM evaluation model and on *AWS Intermediate/Advanced* 4, 5 and 8 for CloudFormation, CDK and SAM.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

Nothing was deployed, no pipeline ran, and no token was exchanged with AWS or GitHub. There is no AWS account, no AWS CLI and no GitHub repository wired to one on the machine that wrote this course.

- **Run here (level 1):** a GitHub Actions workflow file checked by *actionlint* 1.7.12 and by the workflow schema in *check-jsonschema* 0.38.2, and by a checker I wrote; a model of the AWS trust policy for GitHub's identity tokens; the SAM translator (aws-sam-translator 1.114.0) producing the CodeDeploy resources; cfn-lint 1.57.2 on a pipeline template; arithmetic for CodeDeploy traffic shifting and ECS rolling updates; the *moto* 5.2.4 emulator for Parameter Store; and a checker for CloudTrail records, run on records I wrote myself.
- **From the documentation, not run (level 3):** what the real services do. The pages were read on 2026-10-13 through a page-fetch tool that summarises pages; some came back thin and each gap is named where it matters.
- **Needs an account, not run (level 2):** every deploy, every real role assumption, every real canary.

Docker was running but was not needed.

Two Ways to Reach AWS from a Pipeline

The outline's question for this chapter is "CodePipeline, CodeBuild and CodeDeploy, or GitHub Actions with OIDC?". They are not exclusive: a team that keeps its code on GitHub can run builds in GitHub Actions and deploy with a role it assumes over OIDC, or hand the commit to CodePipeline and let AWS run the stages. The picture:

```
A. GitHub Actions + OIDC                          B. CodePipeline + CodeBuild (+ CodeDep

commit -> workflow runs on a GitHub runner        commit -> GitHub connection -> CodePip
  |  asks GitHub for an identity token (JWT)        |  Source stage  (artifact in S3)
  |  sends it to AWS STS                            |  Build stage   (CodeBuild project,
  v                                                 |  Approval      (manual, not charge
AWS checks the token against the role's             v
trust policy -> short-lived credentials           Deploy stage (CloudFormation, CodeDepl
  |
  v  deploy with those credentials                No stored keys either: each service ha
```

What follows checks what can be checked of each, in this order: the identity link in A (workflow file, then the trust policy), the pipeline in B as a template, how new versions are shifted (CodeDeploy, ECS), and then the day-2 tools.

GitHub Actions and OIDC

What GitHub sends

From the documentation (GitHub, read 2026-10-13): a job that sets `permissions: id-token: write` can ask GitHub for a signed token. Its claims: `iss` is `https://token.actions.githubusercontent.com`; `aud` "defaults to the repository owner's URL" and a workflow can ask for another; `sub` depends on the context. The default `sub` formats:

WHEN	DEFAULT `SUB`
Job references an environment	`repo:ORG/REPO:environment:NAME` (whatever the trigger)
Pull request, job without an environment	`repo:ORG/REPO:pull_request`
Branch, no environment, not a pull request	`repo:ORG/REPO:ref:refs/heads/BRANCH`
Tag, same conditions as branch	`repo:ORG/REPO:ref:refs/tags/TAG`
Repositories created after 2026-07-15 (and renamed or	`repo:OWNER@OWNER-ID/REPO@REPO-ID:ref:refs/heads/BRANCH` (example `repo:octo-org@123456/octo-repo@456789:ref:refs/heads/main`); older

WHEN	DEFAULT `sub`
transferred ones after that date)	repositories keep the previous format unless they opt in; not on GitHub Enterprise Server

A `:` inside a value (a branch called `fix:thing`) is written `%3A`. The GitHub page I fetched first did not define the claims or the formats (it only had examples); the reference page it pointed to did, and the table above is from that second page. It did **not** say what happens with forks or `pull_request_target`, and it gave no AWS trust policy or thumbprint advice.

What AWS needs

From the AWS documentation: an IAM OIDC provider has a URL (which must be `https://`, case-sensitive and unique in the account), at least one and at most 100 audiences, and thumbprints (1 to 5). AWS says it verifies the provider's JWKS endpoint using its library of trusted root CAs and falls back to the configured thumbprints only when the certificate is not signed by one of them (or the certificate cannot be retrieved, or TLS 1.3 is required). The `aud` in the token must match an audience registered on the provider, and an OIDC provider used in a role trust policy must be in the same account as the role. The action *aws-actions/configure-aws-credentials* (README read 2026-10-13) recommends a `StringEquals` condition on **both** `token.actions.githubusercontent.com:aud` (`sts.amazonaws.com`) and `:sub`, warns against `ForAllValues:` in an `Allow` statement, gives a default session of 1 hour (900 seconds to 12 hours allowed), and says that role chaining needs both `sts:AssumeRole` and `sts:TagSession` on the first role (`sts:TagSession` can be left out if session tagging is skipped).

A workflow file, checked

Run here: a deploy workflow that tests on every push and pull request and deploys from `main` using an environment and an assumed role. It has no stored AWS keys:

```
name: deploy
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

permissions:
  contents: read

jobs:
```

```
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: echo "run the tests here"

  deploy:
    needs: test
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    environment: production
    permissions:
      id-token: write
      contents: read
    steps:
      - uses: actions/checkout@v4
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::123456789012:role/gha-deploy
          aws-region: eu-west-2
      - run: aws sts get-caller-identity
```

I then made 26 one-change mistakes and four harmless changes, and ran four checkers: PyYAML (does it parse?), the workflow JSON schema through *check-jsonschema*, *actionlint*, and `wfcheck.py`, a checker for the OIDC-specific rules. I wrote the rules in `wfcheck.py` from the pages above **before** this run (a role without `id-token: write`, `write-all`, stored keys, an action not pinned to a tag or SHA, `pull_request_target` with the pull request's code, a role ARN that is not a role ARN, untrusted event text in a `run:` step). One rule, R2, was meant to flag an `id-token: write` granted at workflow level, but I implemented it so that it never fired; the first run missed that change, and I fixed the rule afterwards. The result below is after the fix.

```
py mutate_wf.py
```

```
change                                                  yaml schema ali
(control: unchanged good.yml)                              0      0
remove id-token: write from the deploy job                 0      0
id-token: write at workflow level instead of the job       0      0
id-token: writ (typo in the value)                         0      1
permissions: write-all at the top                          0      0
permission key typo: id-tokens                             0      1
contents: reads (typo in the value)                        0      1
```

```
remove role-to-assume                                                  0      0
input typo: role-to-asume                                              0      0
static key beside the role                                             0      0
action pinned to @main                                                 0      0
action with no ref                                                     0      0
role ARN with 11-digit account                                         0      0
role ARN with partition typo (arn:ws)                                  0      0
needs names a job that does not exist                                  0      0
runs-on missing on deploy                                              0      1
expression syntax error in if                                          0      0
unclosed expression in run                                             0      0
script injection: PR title in run                                      0      0
step with uses and run together                                        0      1
timeout-minutes: abc                                                   0      1
duplicate step id                                                      0      0
pull_request_target checks out the PR head                             0      0
unknown trigger event name                                             0      1
environment name differs from the trust policy (production -> prod)    0      0
aws-region eu-west-9 (not a region)                                    0      0
both: role in the wrong account (0000...)                              0      0
OK: region changed to us-east-1                                        0      0
OK: job renamed                                                        0      0
OK: extra echo step                                                    0      0
OK: checkout pinned to a full commit SHA                               0      0

26 mistakes: caught by PyYAML 0, schema 7, actionlint 14, wfcheck 14
caught by at least one of the three off-the-shelf checks: 14; by wfcheck only: 9; by not
```

actionlint caught 14 of 26, the schema 7 and PyYAML none (they parse). actionlint's list includes things I did not expect from a linter: a typo in an action's input name (`role-to-asume`), a script-injection warning for a pull request title used in a `run:` step, and an action with no version. `wfcheck.py` caught 14, and 9 of them were missed by all three off-the-shelf checks: a missing `id-token: write` (which fails only at run time), `write-all`, a stored key beside the role, an action pinned to `@main`, a role ARN with an 11-digit account, a missing `role-to-assume`, and `pull_request_target` checking out the pull request. **Three checks missed by everything:** a Region that does not exist (`eu-west-9`; I did not read the Region list), a role in the wrong account, and an environment name that no longer matches the trust policy, which only a check across the workflow and the policy can see (below). None of the four harmless changes raised an alarm.

The trust policy, modelled

The workflow is only half of the contract: the role's trust policy decides which tokens may use it. **Run here:** a model of `sts:AssumeRoleWithWebIdentity` for GitHub's tokens, built from the three pages above. It generates the documented `sub` formats and evaluates `StringEquals` and `StringLike` on `:sub` and `:aud`; any other operator makes it stop with `NotModelled`.

```
def sub_for(repo, event, ref=None, environment=None, immutable_ids=None):
    """Default sub claim. repo is 'owner/name'; immutable_ids=(owner_id, repo_id) switch
    owner, name = repo.split("/")
    head = f"repo:{owner}@{immutable_ids[0]}/{name}@{immutable_ids[1]}" if immutable_ids
    esc = lambda s: s.replace(":", "%3A")
    if environment:
        return f"{head}:environment:{esc(environment)}"
    if event == "pull_request":
        return f"{head}:pull_request"
    if ref:
        return f"{head}:ref:{esc(ref)}"
    raise ValueError("need environment, a pull_request event or a ref")
```

```
def glob_match(pattern, value):
    # IAM StringLike: * = any run of characters, ? = one character (case-sensitive)
    rx = "".join(".*" if c == "*" else "." if c == "?" else re.escape(c) for c in patter
    return re.fullmatch(rx, value, re.S) is not None


def cond_ok(cond, claims):
    for op, kv in cond.items():
        if op not in ("StringEquals", "StringLike"):
            raise NotModelled(op)
        for key, want in kv.items():
            claim = key.split(":", 1)[1]
            vals = want if isinstance(want, list) else [want]
            have = claims.get(claim)
            if have is None:
                return False
            if op == "StringEquals":
                ok = have in vals
            else:
                ok = any(glob_match(v, have) for v in vals)
            if not ok:
```

```
                return False
    return True
```

Then a matrix of eight trust policies against ten tokens (the intended repository on a branch, a feature branch, a pull request, two environments, a tag, a different repository of the same owner, a different owner with the same repository name, a lookalike name, and a correct token with the wrong audience). My expected answers came from the documented rules, so the matrix tests the model against my reading, not against AWS.

```
py oidc_tests.py
```

```
                                    0  1  2  3  4  5  6  7  8  9
   0 = main
   1 = feature
   2 = PR
   3 = env production
   4 = env staging
   5 = tag v1.0
   6 = other repo acme/blog main
   7 = other owner evil/shop main
   8 = lookalike acme/shop-evil main
   9 = main, wrong audience
```

```
ok   A  aud only, no sub condition        Y  Y  Y  Y  Y  Y  Y  Y  Y  .   9 allowed
ok   B  sub = main branch (StringEquals)  Y  .  .  .  .  .  .  .  .  .   1 allowed
ok   C  sub = environment production      .  .  .  Y  .  .  .  .  .  .   1 allowed
ok   D  sub like repo:acme/shop:*         Y  Y  Y  Y  Y  Y  .  .  .  .   6 allowed
ok   E  sub like repo:acme/*              Y  Y  Y  Y  Y  Y  Y  .  Y  .   8 allowed
ok   F  sub like repo:acme/shop*          Y  Y  Y  Y  Y  Y  .  .  Y  .   7 allowed
ok   G  sub = main OR production (list)   Y  .  .  Y  .  .  .  .  .  .   2 allowed
ok   H  no aud condition, sub = main      Y  .  .  .  .  .  .  .  .  .   1 allowed

8 policies x 10 tokens = 80 cells; policies whose row differs from my expectation: 0

-- immutable format (repos created after 2026-07-15) --
token sub: repo:acme@123456/shop@456789:ref:refs/heads/main
policy B (old-format sub) allows it: False
policy with owner/repo IDs allows it: True
policy with owner/repo IDs allows the OLD-format token for the same repo: False
policy D (repo:acme/shop:*) allows it: False

-- ':' in a ref (documented as %3A) --
```

```
  sub: repo:acme/shop:ref:refs/heads/fix%3Athing
  policy D allows it: True

  -- NotModelled --
  ForAllValues:StringLike -> NotModelled ForAllValues:StringLike
```

Read the rows. **Policy A, with an audience condition and no `sub` condition, lets in all nine tokens that carry the right audience, including a repository owned by someone else.** The audience is a fixed string that every GitHub workflow can request, so it does not identify you; the `sub` condition does. Policy E (`repo:acme/*`) lets in every repository of the organisation, including a lookalike such as `acme/shop-evil` (an *organisation* may well own that name), and F (`repo:acme/shop*`) admits the lookalike as well. Policy D (`repo:acme/shop:*`, with the colon) is the narrowest wildcard that is still one repository. The exact-match policies B and C admit one token each, and G lets a list of two values in. The last lines of the output show two further effects: **a policy written for the old subject format stops matching a repository that uses the new, ID-based format** (it fails closed, so your deployment breaks rather than your security), and the `%3A` encoding passes a wildcard that was written without thinking about it.

The workflow and the policy meet at the environment name. I derived the `sub` that the deploy job above would send, and evaluated it against the trust policy that the pipeline template (below) contains:

```
  py cross_check.py
```

```
  good workflow, push to main                                  sub=repo:acme/shop:enviro
  good workflow, pull_request event (job has environment:)     sub=repo:acme/shop:enviro
  environment: line removed, push to main                      sub=repo:acme/shop:ref:re
  environment: line removed, pull_request event                sub=repo:acme/shop:pull_r
  environment: prod (renamed), push to main                    sub=repo:acme/shop:enviro

  note: the job's 'if: github.ref == refs/heads/main' is what keeps the pull_request run a
  the trust policy alone does not stop it, because the environment sub is the same for eve
  separate control: not read this session.)
```

Renaming the environment to `prod` in the workflow makes AWS refuse the token while every file is individually valid, and removing the `environment:` line has the same effect. The second row is the one to think about: **because a job that names an environment sends the environment `sub` whatever triggered it, a `pull_request` run of that job holds a token the**

policy accepts. In this workflow what stops a pull request from deploying is the job's `if: github.ref == 'refs/heads/main'`, not AWS. GitHub's environment protection rules are a further control that I did not read.

CodePipeline and CodeBuild as a Template

From the documentation (pricing pages, 2026-10-13): CodePipeline V1 is $1.00 per active pipeline per month (first 30 days free, one free active pipeline); V2 is $0.002 per action execution minute rounded up, with 100 free minutes shared by the account, and manual approvals are not charged; an active pipeline is one that is older than 30 days and had a code change that month. CodeBuild's page gave $0.005 per minute for the small Linux on-demand type only in a worked example, with 100 free minutes a month. Neither page named a Region. Prices for GitHub Actions were not read.

Run here: one template with the GitHub provider, a deploy role whose trust policy matches the workflow above, a site bucket, an artifact bucket, a build role, a CodeBuild project, a pipeline role and a three-stage V2 pipeline (source from a GitHub connection, build, manual approval). The trust policy and the pipeline's wiring:

```
  DeployRole:
    Type: AWS::IAM::Role
    Properties:
      MaxSessionDuration: 3600
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action: sts:AssumeRoleWithWebIdentity
            Principal:
              Federated: !Ref GitHubOidc
            Condition:
              StringEquals:
                token.actions.githubusercontent.com:aud: sts.amazonaws.com
                token.actions.githubusercontent.com:sub: !Sub "repo:${GitHubRepo}:enviro
```

```
  Pipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
```

```
      PipelineType: V2
      RoleArn: !GetAtt PipelineRole.Arn
      ArtifactStore:
        Type: S3
        Location: !Ref ArtifactBucket
      Stages:
        - Name: Source
          Actions:
            - Name: GitHub
              ActionTypeId:
                Category: Source
                Owner: AWS
                Provider: CodeStarSourceConnection
                Version: "1"
              Configuration:
                ConnectionArn: !Ref GitHubConnectionArn
                FullRepositoryId: !Ref GitHubRepo
                BranchName: main
              OutputArtifacts:
                - Name: src
        - Name: Build
          Actions:
            - Name: Build
              ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName: !Ref BuildProject
              InputArtifacts:
                - Name: src
              OutputArtifacts:
                - Name: built
        - Name: Approve
          Actions:
            - Name: Manual
              ActionTypeId:
                Category: Approval
                Owner: AWS
                Provider: Manual
                Version: "1"
```

It linted clean with cfn-lint 1.57.2. Then 26 one-change mistakes and three harmless changes, with cfn-lint on one side and `pipecheck.py` on the other. `pipecheck.py` evaluates the template's own trust policy with the model above against eight tokens that should be refused and one that should be accepted, and checks permissions and wiring. As with the workflow checker, its rules were written before this run, from the pages above and the design's intent (only the `production` environment of this repository may assume the role).

```
py mutate_pipe.py
```

```
change                                                          lint  mine
(control)                                                          0     0
sub condition: StringLike repo:acme/* (whole organisation)         0     1
sub condition removed                                              0     1
aud condition removed                                              0     0
aud typo sts.amazon.com                                            0     1
environment production -> prod                                     0     1
sub is the main branch, not the environment                        0     1
provider ClientIdList has the wrong audience                       0     1
trust Action sts:AssumeRole instead of AssumeRoleWithWebIdentity   0     1
Principal Federated Ref to a missing resource                      1     1   E1020
deploy policy s3:* on Resource *                                   0     1
CodeBuild PrivilegedMode true                                      0     1
site bucket without PublicAccessBlock                              0     1
Build input artifact renamed (src -> source)                       1     1   E3701
Build action InputArtifacts removed                                1     1   E3702
pipeline role loses codebuild:StartBuild                           0     1
build role loses logs:PutLogEvents                                 0     1
pipeline RoleArn points at BuildRole                               0     1
pipeline artifact store is the site bucket                         0     1
BuildProject TimeoutInMinutes 15 -> 0                              1     0   E3034
ComputeType typo                                                   0     0
PipelineType V3 (does not exist)                                   1     0   E3030
MaxSessionDuration 100                                             1     0   E3034
lifecycle ExpirationInDays 0                                       0     0
GitHubRepo default breaks AllowedPattern                           1     0   E2015
Source BranchName removed                                          0     0
Output artifact name used twice                                    1     0   E3701
OK: repo default changed to acme/blog                              0     0
OK: build timeout 30                                               0     0
OK: build command changed                                          0     0

26 mistakes: cfn-lint 8, pipecheck 17, both 3, only pipecheck 14, neither 4; false alarm
```

cfn-lint reported an error for 8 of 26; `pipecheck.py` for 17; both for 3; neither for 4. cfn-lint found out-of-range values (a build timeout of 0, a session duration of 100, a pipeline type that does not exist), a dangling reference, an artifact name that no action produces or one produced twice, and a pattern that the parameter default breaks. It passed every other trust-policy change, a role with `s3:*` on every resource, a privileged build container, a bucket without a public-access block and a pipeline role without the right to start the build. `pipecheck.py` crashed on one change (the repository default that breaks the parameter pattern: my helper cannot split it) and I do not count that as a catch. The four that got through are: **removing the audience condition** (in my model the provider rejects tokens whose audience is not registered, so the condition adds nothing, but the README recommends both and I cannot say whether AWS treats them as equivalent), a typo in the build compute type, a lifecycle expiry of 0 days, and a missing branch name on the source action. The same caution as in chapter 10 applies: the checker is evidence that these rules can be written, not that it would catch a mistake it was not written for.

Deploying a New Version Safely

The predefined configurations

From the documentation (CodeDeploy, 2026-10-13): for Lambda there are four canary configurations (10 percent first, the rest after 5, 10, 15 or 30 minutes), four linear ones (10 percent every 1, 2, 3 or 10 minutes) and all-at-once. For ECS there are two linear ones (10 percent every 1 or 3 minutes), two canary ones (10 percent, rest after 5 or 15 minutes) and all-at-once; with a Network Load Balancer only all-at-once is supported. For EC2 and on-premises instances the predefined choices are all-at-once, half-at-a-time and one-at-a-time (the default), which count healthy hosts rather than traffic. You can define your own canary or linear configuration for Lambda and ECS, except that with CloudFormation-managed ECS blue/green you cannot.

What SAM generates

Run here: a SAM template with one function that has `AutoPublishAlias: live` and a `DeploymentPreference` (canary, an alarm and a pre-traffic hook), the hook function, and the alarm. SAM's page says this creates a CodeDeploy application and deployment group by itself, and that the first gradual deployment should be done in two steps (publish the alias first, add the preference second) because CodeDeploy needs a previous version to shift from. The translator, run offline, turned the three declared resources into 10:

```
    Properties:
      Handler: app.handler
      CodeUri: s3://example-bucket/code.zip
      AutoPublishAlias: live
      DeploymentPreference:
        Type: Canary10Percent5Minutes
        Alarms:
          - !Ref AliasErrorsAlarm
        Hooks:
          PreTraffic: !Ref PreTrafficFn
```

```
py transform.py sam/template.yaml
```

```
sam/template.yaml: 10 resources, 7 types
   1 AWS::CloudWatch::Alarm
   1 AWS::CodeDeploy::Application
   1 AWS::CodeDeploy::DeploymentGroup
   3 AWS::IAM::Role
   1 AWS::Lambda::Alias
   2 AWS::Lambda::Function
   1 AWS::Lambda::Version
```

```
{
  "AppFnAliaslive": {
    "Type": "AWS::Lambda::Alias",
    "UpdatePolicy": {
      "CodeDeployLambdaAliasUpdate": {
        "ApplicationName": {
          "Ref": "ServerlessDeploymentApplication"
        },
        "DeploymentGroupName": {
          "Ref": "AppFnDeploymentGroup"
        },
        "BeforeAllowTrafficHook": {
          "Ref": "PreTrafficFn"
        }
      }
    },
    "Properties": {
      "Name": "live",
      "FunctionName": {
```

```
        "Ref": "AppFn"
      },
      "FunctionVersion": {
        "Fn::GetAtt": [
          "AppFnVersionc17514d5e0",
          "Version"
        ]
      }
    }
  },
  "AppFnDeploymentGroup": {
    "Type": "AWS::CodeDeploy::DeploymentGroup",
    "Properties": {
      "AlarmConfiguration": {
        "Enabled": true,
        "Alarms": [
          {
            "Name": {
              "Ref": "AliasErrorsAlarm"
            }
          }
        ]
      },
      "ApplicationName": {
        "Ref": "ServerlessDeploymentApplication"
      },
      "AutoRollbackConfiguration": {
        "Enabled": true,
        "Events": [
          "DEPLOYMENT_FAILURE",
          "DEPLOYMENT_STOP_ON_ALARM",
          "DEPLOYMENT_STOP_ON_REQUEST"
        ]
      },
      "DeploymentConfigName": {
        "Fn::Sub": [
          "CodeDeployDefault.Lambda${ConfigName}",
          {
            "ConfigName": "Canary10Percent5Minutes"
          }
        ]
      },
      "DeploymentStyle": {
        "DeploymentType": "BLUE_GREEN",
        "DeploymentOption": "WITH_TRAFFIC_CONTROL"
```

```
      },
      "ServiceRoleArn": {
        "Fn::GetAtt": [
          "CodeDeployServiceRole",
          "Arn"
        ]
      }
    }
  }
}
```

The alias carries an `UpdatePolicy` that points at the deployment group; the group names the configuration (`CodeDeployDefault.Lambda` followed by `Canary10Percent5Minutes`), turns on rollback for failure, stop-on-alarm and stop-on-request, and is a blue/green deployment with traffic control. I changed one thing at a time (16 mistakes and two valid changes) and ran the SAM translator and cfn-lint:

```
py mutate_sam11.py
```

```
change                                                                  translator
(control: unchanged template)                                                    0
Type Canary10Percent5Minutes -> Canary10Percent7Minutes                          0
Type -> Linear10PercentEvery1Minute (valid, slower)                              0
Type -> AllAtOnce (valid, no gradual shift)                                      0
remove AutoPublishAlias                                                          1
Alarms entry names a resource that does not exist                                0
PreTraffic hook names a resource that does not exist                             0
hook function name lacks the CodeDeployHook_ prefix                              0
hook role lacks PutLifecycleEventHookExecutionStatus                             0
alarm watches the function, not the alias (Dimension removed)                    0
alarm Threshold 0 -> text                                                        0
alarm ComparisonOperator typo                                                    0
alarm Period 60 -> 45 (not a valid period)                                       0
Alarms list empty                                                                0
DeploymentPreference on the hook function too (hook shifts traffic itself)       0
Runtime python3.13 -> python3.99                                                 0
Timeout 5 -> 901                                                                 0
alarm EvaluationPeriods 1 -> 0                                                   0
alarm TreatMissingData breaching (alarm fires on idle alias)                     0

16 mistakes: translator 1, cfn-lint 5, both 0, neither 10; false alarms on 2 valid chang
```

The translator caught 1 of 16 (a deployment preference without `AutoPublishAlias`), cfn-lint 5, both 0, and 10 got past both. The two tools did catch alarm-property and reference problems, but not `Canary10Percent7Minutes`, which is not in the documented list of four canary options (I did not see what CodeDeploy does with a name that does not exist, so that miss is judged against the docs' list). They also did not notice a missing alarm resource, a hook function without the `CodeDeployHook_` prefix that the SAM-created role policy expects (my understanding, not read this session), a hook role without the permission to report back, an alarm that watches the function instead of the alias, or an alarm with `TreatMissingData: breaching`. In my test, cfn-lint 1.57.2 also passed `Runtime: python3.99` and a 901-second timeout in this template; chapter 8 saw the same weak result with 1.57.2 and a stronger one with 1.53.3, and I did not investigate why. I reinstalled the translator at the version pinned for chapter 8 to rule out a version mismatch before accepting that result.

Arithmetic: how much goes wrong before the stop?

The pages give the configurations; they do not say how bad a bad release is under each. **Run here:** a small model. Its assumptions, which are mine and not on the pages: a linear configuration sends its first 10 percent at minute 0 and finishes at nine times its interval; requests are spread evenly; the bug fails every request on the new version; and the alarm stops the deployment a fixed number of minutes after traffic first reaches the new version. The first block checks the model against the wording of the pages:

```
py deploy_tests.py   # schedules and page checks
```

```
== schedules (my assumption: linear starts at minute 0 and ends at 9 x interval) ==
LambdaCanary10Percent5Minutes          steps  2   full after   5 min   0m:10% 5m:100%
LambdaCanary10Percent10Minutes         steps  2   full after  10 min   0m:10% 10m:100%
LambdaCanary10Percent15Minutes         steps  2   full after  15 min   0m:10% 15m:100%
LambdaCanary10Percent30Minutes         steps  2   full after  30 min   0m:10% 30m:100%
LambdaLinear10PercentEvery1Minute      steps 10   full after   9 min   0m:10% 1m:20% 2m:
LambdaLinear10PercentEvery2Minutes     steps 10   full after  18 min   0m:10% 2m:20% 4m:
LambdaLinear10PercentEvery3Minutes     steps 10   full after  27 min   0m:10% 3m:20% 6m:
LambdaLinear10PercentEvery10Minutes    steps 10   full after  90 min   0m:10% 10m:20% 20
LambdaAllAtOnce                        steps  1   full after   0 min   0m:100%
ECSLinear10PercentEvery1Minutes        steps 10   full after   9 min   0m:10% 1m:20% 2m:
ECSLinear10PercentEvery3Minutes        steps 10   full after  27 min   0m:10% 3m:20% 6m:
ECSCanary10Percent5Minutes             steps  2   full after   5 min   0m:10% 5m:100%
ECSCanary10Percent15Minutes            steps  2   full after  15 min   0m:10% 15m:100%
ECSAllAtOnce                           steps  1   full after   0 min   0m:100%
```

```
  == checks against the page's wording ==
  ok   Canary10Percent5Minutes: 10% first, the rest 5 minutes later
  ok   Linear10PercentEvery3Minutes: 10 equal steps of 10%
  ok   AllAtOnce: one step of 100%
  ok   ECS rolling 50% / desired 4: scheduler can stop 2
  ok   ECS rolling 75% / desired 2: cannot stop any (minimum healthy rounds up to 2)
  ok   ECS rolling max 200% / desired 4: can start 4
  ok   ECS rolling max 125% / desired 3: cannot start any (limit rounds down to 3)
```

Then the failed requests at 100 requests a second, for an alarm that needs 2 minutes and then for one that needs 10:

```
  py deploy_tests.py   # failed requests before the stop
```

```
  == failed requests before a stop (100 rps, a bug that fails every request on the new vers
  LambdaAllAtOnce                          12,000 failed requests of 12,000 in the window
  LambdaCanary10Percent5Minutes             1,200 failed requests of 12,000 in the window
  LambdaCanary10Percent30Minutes            1,200 failed requests of 12,000 in the window
  LambdaLinear10PercentEvery1Minute         1,800 failed requests of 12,000 in the window
  LambdaLinear10PercentEvery10Minutes       1,200 failed requests of 12,000 in the window

  same, but the alarm needs 10 minutes (a slow metric):
  LambdaAllAtOnce                          60,000 failed requests of 60,000 in the window
  LambdaCanary10Percent5Minutes            33,000 failed requests of 60,000 in the window
  LambdaCanary10Percent30Minutes            6,000 failed requests of 60,000 in the window
  LambdaLinear10PercentEvery1Minute        33,000 failed requests of 60,000 in the window
  LambdaLinear10PercentEvery10Minutes       6,000 failed requests of 60,000 in the window
```

The rows teach two things. With a quick alarm every gradual configuration holds the damage to 1,200 to 1,800 requests against 12,000 for all-at-once, whatever its length. With a slow alarm **a short canary does not help: the 5-minute canary ends before the 10-minute alarm has decided, the remaining 90 percent arrives, and 33,000 requests fail, as many as the 1-minute linear.** A 30-minute canary or a 10-minute linear step keeps it to 6,000. The window of the check has to be longer than the alarm's time to detect. And a canary only protects against what its alarm can see:

```
  py deploy_tests.py   # can a canary see the bug at all?
```

```
== can a canary see the bug at all? (bug fails 1 request in 1,000 on the new version) ==
expected failures on the new version during the first canary step (10% of traffic):
    1 rps,  5-minute canary: expect   0.03 failures, chance of seeing none =  97.0%
    1 rps, 15-minute canary: expect   0.09 failures, chance of seeing none =  91.4%
   10 rps,  5-minute canary: expect   0.30 failures, chance of seeing none =  74.1%
   10 rps, 15-minute canary: expect   0.90 failures, chance of seeing none =  40.7%
  100 rps,  5-minute canary: expect   3.00 failures, chance of seeing none =   5.0%
  100 rps, 15-minute canary: expect   9.00 failures, chance of seeing none =   0.0%
(Poisson arithmetic; the real alarm's threshold and evaluation period decide what counts
```

A bug that fails one request in a thousand, at 1 request a second, is invisible in a 5-minute canary 97 percent of the time (Poisson arithmetic, for a simple count of failures; the real alarm's threshold and evaluation periods decide what counts as seen). At 100 requests a second the same canary would almost always see it. Low-traffic services need a synthetic test (the pre-traffic hook) more than they need a long canary. A last check that the weighted split is a split: 13 percent of 100 requests went to the new version, 10.4 percent of 1,000 and 10.0 percent of 100,000 (random draws with a fixed seed, which says nothing about how Lambda chooses).

ECS rolling updates

From the documentation (ECS rolling update page): `minimumHealthyPercent` is "rounded up" and `maximumPercent` "rounded down"; the page's four examples (50 percent of 4 tasks lets the scheduler stop 2; 75 percent of 2 tasks lets it stop none; 200 percent of 4 lets it start 4; 125 percent of 3 lets it start none) are in the checks above. It also says to make sure the scheduler can stop or start at least one task, or the deployment is stuck. The circuit breaker and CloudWatch-alarm failure detection are named on the page and link to pages I did not read; the page says both can roll back and that when both are set, "the deployment is set to failed as soon as the failure criteria for either" is met. Blue/green and the newer linear and canary options for ECS were not read beyond the CodeDeploy table. My table of waves uses a simple model of the scheduler (start up to the maximum, stop down to the minimum), not the real one:

```
py deploy_tests.py   # rolling update table
```

```
== ECS rolling update: waves and stuck settings (my simple model of the scheduler, not t
desired  min%  max% min healthy max total can stop can start  waves
      4   100   200           4         8        0         4  1
      4    50   100           2         4        2         0  2
      4   100   100           4         4        0         0  STUCK (cannot stop or star
```

```
 4    0   100          0        4        4        0  1
 2   75   200          2        4        0        2  1
 3  100   125          3        3        0        0  STUCK (cannot stop or star
 1  100   200          1        2        0        1  1
 1    0   100          0        1        1        0  1
10   90   110          9       11        1        1  5
10  100   110         10       11        0        1  10
```

A setting of 100 and 100 percent is stuck for any task count, as is 100 and 125 percent with 3 tasks; with one task, 100 and 200 percent gives a start-first rollout (a second task is started before the first is stopped), while 0 and 100 stops the only task first, which means an outage for as long as the new task takes to become healthy. This is the arithmetic of the page's rule, not an observation of ECS.

Day-2 Operations

Systems Manager

From the documentation (Session Manager prerequisites): managed nodes need SSM Agent 2.3.68.0 or later; KMS encryption of session data needs 2.3.539.0; shell profiles 3.0.161.0; port forwarding and SSH sessions 3.0.222.0; streaming session data to CloudWatch Logs 3.0.284.0. The node must reach `ec2messages`, `ssm` and `ssmmessages` endpoints for its Region over HTTPS on port 443 (outbound), or through interface endpoints (*AWS Fundamentals* 5 covers VPC endpoints). Using the AWS CLI to start a session needs the Session Manager plugin and CLI 1.16.12 or later. The managed policy `AmazonSSMManagedInstanceCore` provides the permissions for a node. The page text I read did not discuss inbound ports. Session Manager and Patch Manager need real instances with the agent, so I could not run them here, and I did not read the Patch Manager pages.

Run here: Parameter Store through moto, against the limits read for chapter 3 (standard 4 KB, advanced 8 KB), then CloudTrail records (below):

```
py ops_checks.py   # Parameter Store
```

```
== (a) moto Parameter Store vs the documented size limits ==
ok   standard parameter, 4096 bytes                          docs say accept  moto ac
DIFF standard parameter, 4097 bytes                          docs say reject  moto ac
ok   advanced parameter, 8192 bytes                          docs say accept  moto ac
```

```
  DIFF advanced parameter, 8193 bytes                                docs say reject  moto ac
  DIFF standard tier, 4097 bytes, Tier given explicitly              docs say reject  moto ac
  ok   Type 'Secret' (not String/StringList/SecureString)            docs say reject  moto re
  observed: SecureString with decryption -> 'hunter2'; without -> 'kms:alias/aws/ssm:hunte
  observed: second put without Overwrite -> ParameterAlreadyExists (my understanding is th
  scored cases: 6; moto agrees with the page on 3
```

moto agreed with the page on 3 of 6. It accepted a 4,097-byte standard parameter, an 8,193-byte advanced parameter and a 4,097-byte value with `Tier` given explicitly as `Standard`, and rejected a type that does not exist. Two things I observed about its `SecureString` reading it without decryption returned `kms:alias/aws/ssm:hunter2`, so the "encrypted" value is the plain text with a prefix, and a second put without `Overwrite` was refused with `ParameterAlreadyExists` (that last behaviour I believe matches the service, but I did not read a page for it). The emulator is fine for testing that your code calls the API; it cannot tell you that a value is too large.

CloudTrail

From the documentation: event history is "a viewable, searchable, downloadable, and immutable record of the past 90 days of CloudTrail management events in an AWS Region"; a trail delivers events to an S3 bucket, optionally to CloudWatch Logs and EventBridge; by default trails log management events but not data events or Insights events; and "CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order". The record-contents page lists the fields. The ones marked as always present (`Optional: False`) are `eventTime` , `eventVersion` , `userIdentity` , `eventSource` , `eventName` , `awsRegion` , `sourceIPAddress` , `requestParameters` , `responseElements` , `eventID` , `eventType` and `eventCategory` . The current `eventVersion` is 1.11; the page tells you to compare the major version for equality and the minor version with greater-or-equal. `responseElements` is null for read-only APIs, and some services put error details in `responseElements` rather than in `errorCode` and `errorMessage` . The page did not say how long delivery to a trail takes, and I did not read the pricing page.

Run here: a checker for those rules, run on ten **synthetic** records that I wrote (not real CloudTrail output), including the denied `AssumeRoleWithWebIdentity` event that a wrong trust policy would produce:

```
  py ops_checks.py   # CloudTrail records
```

```
  == (b) CloudTrail record checker (fields marked 'Optional: False' on the record-contents
  complete record                              ok
  oidc assume, success                         ok
  oidc assume, denied                          ok
  no eventID                                   missing eventID
  no awsRegion and no sourceIPAddress          missing awsRegion; missing sourceIPAddress
  eventVersion 1.08                            minor 8 < 11: newer fields (eventContext) can
  eventVersion 2.0                             major version 2 differs from 1: do not parse
  errorCode without errorMessage               errorCode without errorMessage (page: some se
  readOnly but responseElements present        readOnly event with responseElements (page: n
  no eventCategory (older record)              missing eventCategory

  == (c) questions answered from the synthetic records ==
  events by name (records that passed the check): {'PutObject': 1, 'AssumeRoleWithWebIdent
  denied assume-role events: [('AssumeRoleWithWebIdentity', 'AccessDenied')]
```

Use this as the shape of a real audit query, not as a test of CloudTrail: the question "who assumed the deploy role, from which repository?" is answered from the `requestParameters` and the token claims that appear in the event, and I did not read the page that lists the claims CloudTrail records for a web identity, so I wrote none.

Config, GuardDuty and Security Hub, in a paragraph each

AWS Config (from its overview page) "provides a detailed view of the configuration of AWS resources in your AWS account", including how they are related and how they were configured in the past; it needs an S3 bucket and an IAM role (and an SNS topic if you want notifications) and lets you choose which resource types to record. Config *rules* evaluate the recorded configuration and flag a resource as noncompliant; conformance packs deploy collections of rules; aggregators collect data from several accounts and Regions. Where CloudTrail answers "who called what", Config answers "what did this resource look like then, and does it comply now" (my summary of the two pages). **Amazon GuardDuty** "continuously monitors, analyzes, and processes AWS data sources and logs": foundational sources are CloudTrail management events, VPC flow logs and DNS logs; optional protection plans cover S3, EKS, Runtime Monitoring, RDS, Lambda, malware and others. A first-time enable includes a 30-day free trial and turns on all protection plans except Runtime Monitoring. **AWS Security Hub CSPM** aggregates findings (in the AWS Security Finding Format) from GuardDuty, Inspector, Macie and partners and runs checks against standards (AWS Foundational Security Best Practices, CIS, PCI DSS, NIST); most of its controls use AWS Config rules, so Config must be enabled and recording;

it only sees findings generated after it is enabled, and only in the Region where it is enabled. I did not read pricing for any of the three, and none of them can be run without an account.

What a Real Run Involves (not run)

Level 2: needs an account and a repository. In outline: create the IAM OIDC provider (the AWS CLI command on the IAM page is `aws iam create-open-id-connect-provider`), create the role with a trust policy like the one in the template, set the repository's workflow to assume it, and look at the CloudTrail event for the assumption. I give no further commands because I did not read the CLI pages for them. Set a budget alert first (*AWS Fundamentals* 9), keep every test small, and delete the stack afterwards; the provider and role are free to hold, the pipeline and build minutes are not.

The one cost sketch I can make from the pages is for 10 builds a day on 22 working days at 3 minutes each:

```
py cost11.py
```

```
220 builds a month x 3 minutes = 660 build minutes
CodeBuild general1.small: (660 - 100 free) x $0.005 = $2.80   (free-tier minutes apply t
CodePipeline V1: $1.00 for one active pipeline (the first one is free: $0.00)
CodePipeline V2: (660 build-action minutes - 100 free) x $0.002 = $1.12   (counts only t
total, V2 pipeline + CodeBuild, assumptions above: $3.92 a month

sensitivity: builds of 6 minutes instead of 3
  CodeBuild $6.10, V2 $2.44, total $8.54

not priced: GitHub Actions minutes, CloudTrail, GuardDuty, Config, Security Hub, Session
```

Cost and Clean-Up

Nothing in this chapter created anything in AWS. On this machine: the scratch folder `C:/lwaws/c11` (a Python environment, files and saved outputs). The moto emulator ran in-process, so no server was left running. Docker was not used.

> **WHAT WAS AND WAS NOT CHECKED**

Run here: a workflow file against three checkers and 26 mistakes; a trust-policy model (8 policies by 10 tokens, the new subject format, a `%3A` value, one unmodelled operator); a workflow-to-policy cross-check; a pipeline template against cfn-lint and a rule checker on 26 mistakes; a SAM template translated and checked on 16 mistakes; deployment arithmetic (14 configurations, 7 checks against the pages, failed-request and canary-visibility tables, an ECS rolling-update table); Parameter Store in moto (6 scored cases); a CloudTrail record checker on 10 synthetic records; a cost sketch. **From the documentation, not run:** everything the real services do, including how CodeDeploy, ECS, STS and CloudTrail behave. **Not read:** GitHub Actions pricing and environment protection rules, fork and `pull_request_target` behaviour, the CodeBuild buildspec reference, CodePipeline's execution and stage rules, CodeDeploy's AppSpec reference and hooks, the ECS circuit breaker, early-success and blue/green pages, Lambda alias routing, Patch Manager, Parameter Store naming and SecureString pages, CloudTrail pricing and delivery times, AWS Config, GuardDuty and Security Hub pricing, the claims CloudTrail records for web identities. **Not checked at all:** any real token exchange, build, deployment, session, alarm or finding.

Practice

EXERCISE 1

Write a GitHub Actions deploy workflow that assumes an AWS role with OIDC and no stored keys. Check it with actionlint and the workflow schema, then make 26 one-change mistakes and write a checker for the OIDC-specific rules that those tools do not cover (write the rules before you run it, and say which you fixed afterwards). Model the AWS trust policy for GitHub's tokens, test eight policies against ten tokens, including the new subject format, and cross-check the workflow's environment against the policy.

View solution

EXERCISE 2

Model CodeDeploy's predefined Lambda and ECS traffic-shifting schedules and ECS rolling-update percentages from the documentation, test them on the pages' own examples, then measure how many requests fail before a stop for different configurations and alarm delays and how likely a canary is to see a rare bug. Write a SAM template with a canary deployment preference, translate it offline, and mutate it 16 ways against the translator and cfn-lint.

View solution

EXERCISE 3

Write a CloudFormation template for the delivery path (OIDC provider, deploy role, CodeBuild, a V2 CodePipeline), lint it, make 26 mutations, and write a checker that evaluates the template's trust policy with your model and checks permissions and wiring (write the rules first, and count a crash as a miss). Check Parameter Store in an emulator against the documented size limits, write a CloudTrail record checker from the documented fields, and sketch a monthly cost from the pricing pages.

View solution

CHAPTER 11 QUICK REFERENCE

- GitHub to AWS without keys: the job needs `permissions: id-token: write`, the role's trust policy must name the provider and restrict `:sub` (and `:aud` = `sts.amazonaws.com`) with `StringEquals`; an audience-only policy admits every GitHub repository
- Default `sub`: environment wins (`repo:O/R:environment:E`, for every trigger), else `:pull_request`, else `:ref:refs/heads/B` or `refs/tags/T`; repositories created after 2026-07-15 use `repo:OWNER@ID/REPO@ID:...`, so a policy for the old form silently stops matching
- Wildcards: `repo:acme/shop:*` is one repository; `repo:acme/shop*` and `repo:acme/*` are wider; never `ForAllValues:` in an Allow
- Linters check syntax and some inputs, not your intent: actionlint caught 14 of 26 workflow mistakes, cfn-lint 8 of 26 pipeline-template mistakes and 5 of 16 SAM-template mistakes; a rule checker needs rules written *before* you see the results
- Lambda: `AutoPublishAlias` plus `DeploymentPreference`; the first gradual deployment needs an earlier version; configurations are Canary10Percent{5,10,15,30}Minutes, Linear10PercentEvery{1,2,3,10}Minute(s), AllAtOnce; ECS has linear 1 and 3 minutes, canary 5 and 15 minutes
- A canary shorter than your alarm's detection time protects nothing; a bug that fails 1 in 1,000 at 1 request a second is unseen in a 5-minute canary 97 percent of the time (under my assumptions); low traffic needs a pre-traffic test
- ECS rolling: minimum healthy rounds up, maximum rounds down, and 100/100 can never move
- CloudTrail: 90 days of management events in event history; trails go to S3; check `eventVersion` major equal and minor at least what you need; the always-present fields are on the record-contents page
- Config records configuration over time and evaluates rules; GuardDuty detects threats from CloudTrail, flow and DNS logs and more; Security Hub CSPM aggregates findings and runs standard checks that need Config

- moto accepted values over the Parameter Store size limits and stores SecureString as prefixed plain text; use emulators to test your calls, not to find validity errors

COMING NEXT

Chapter 12 is the capstone: a small system assembled from the previous chapters (account structure, IaC, a container service, an API, messaging, edge delivery and a pipeline) and reviewed against the Well-Architected Framework, with one combined course PDF at the end. Start with *AWS Intermediate/Advanced 12*.

AWS Intermediate/Advanced

Chapter 12 · Capstone: A Well-Architected Application

This chapter builds one small system from the pieces of the course and then reviews it. The system takes orders: a static site served by CloudFront (*AWS Intermediate/Advanced* 10), an HTTP API with a Lambda function (*AWS Intermediate/Advanced* 8), a queue with a dead-letter queue and a worker that reports partial batch failures (*AWS Intermediate/Advanced* 9), a DynamoDB table and a secret encrypted with a customer-managed KMS key (*AWS Intermediate/Advanced* 3), alarms (*Observability*), all defined as code with the CDK (*AWS Intermediate/Advanced* 5). The review uses the six pillars of the AWS Well-Architected Framework.

HOW EVERY EXAMPLE IN THIS CHAPTER WAS VERIFIED

Nothing was deployed and no request travelled through anything AWS runs. There is no AWS account on the machine that wrote this course.

- **Run here (level 1):** `cdk synth` (aws-cdk 2.1145.0, aws-cdk-lib 2.273.0, TypeScript) producing two real CloudFormation templates; cfn-lint 1.57.2 on both; eight Jest assertions on the stack; the two Lambda handlers run against the *moto* 5.2.4 emulator; a checker I wrote that applies 36 checks to a synthesised template, grouped by pillar; 29 one-change regressions run against cfn-lint and that checker; and a cost sketch from pricing pages.
- **From the documentation, not run (level 3):** the framework text (read on 2026-10-13 through a page-fetch tool that summarises pages) and what every service does at run time.
- **Needs an account, not run (level 2):** `cdk bootstrap`, `diff`, `deploy` and `destroy`. The commands are in *AWS Intermediate/Advanced* 5; they were not run.

The AWS Pricing Calculator was not used (the outline asked for it; it is an interactive tool that I could not drive). The cost sketch uses rates from the pricing pages and says which lines it could not price. Docker was not needed.

The Design

```
browser --> CloudFront (one distribution, one domain)
              |  /*       -> private S3 bucket (origin access control)        static si
              |  /api/*   -> HTTP API stage "live" -> ApiFn (Lambda, arm64)    POST /api
                                                       |  validate, answer 202 + order
                                                       v
```

```
                                          SQS work queue --(5 failed receives)--> de
                                                    |  batches of 10, partial-batch
                                                    v
                                          WorkerFn (Lambda) --conditional put--> Dyn

 one customer-managed KMS key (queues, table, secret) | one Secrets Manager secret |
```

The stack is one construct class with one flag, `hardened`. With the flag off it is my first draft; with it on it contains the changes that the review (below) asked for. Same code, two templates, so the difference between them is exactly what the review found. The core of it:

```
// --- data path: API Lambda -> queue -> worker Lambda -> table
const dlq = new sqs.Queue(this, "OrdersDlq", { encryptionMasterKey: key, retentionPe
const workerTimeout = cdk.Duration.seconds(30);
const queue = new sqs.Queue(this, "OrdersQueue", {
  encryptionMasterKey: key,
  visibilityTimeout: cdk.Duration.seconds(h ? 6 * 30 + 5 : 30),   // the Lambda/SQS
  deadLetterQueue: { queue: dlq, maxReceiveCount: 5 },
  enforceSSL: h,
});
const table = new ddb.Table(this, "Orders", {
  partitionKey: { name: "orderId", type: ddb.AttributeType.STRING },
  billingMode: ddb.BillingMode.PAY_PER_REQUEST,
  encryption: ddb.TableEncryption.CUSTOMER_MANAGED,
  encryptionKey: key,
  pointInTimeRecoverySpecification: { pointInTimeRecoveryEnabled: true },
  deletionProtection: h,
  removalPolicy: keep,
});
const apiKeySecret = new secrets.Secret(this, "ApiSecret", { encryptionKey: key, des

const mkLog = (name: string) => h
  ? new logs.LogGroup(this, name, { retention: logs.RetentionDays.ONE_MONTH, removal
  : undefined;
const apiFn = new lambda.Function(this, "ApiFn", {
  runtime: lambda.Runtime.PYTHON_3_13, architecture: lambda.Architecture.ARM_64,
  handler: "app.handler", code: lambda.Code.fromAsset("app/api"),
  timeout: cdk.Duration.seconds(5), memorySize: 256,
  environment: { QUEUE_URL: queue.queueUrl, SECRET_ARN: apiKeySecret.secretArn },
  tracing: h ? lambda.Tracing.ACTIVE : lambda.Tracing.DISABLED,
  logGroup: mkLog("ApiLogs"),
});
```

```
    queue.grantSendMessages(apiFn);
    apiKeySecret.grantRead(apiFn);

    const workerFn = new lambda.Function(this, "WorkerFn", {
      runtime: lambda.Runtime.PYTHON_3_13, architecture: lambda.Architecture.ARM_64,
      handler: "worker.handler", code: lambda.Code.fromAsset("app/worker"),
      timeout: workerTimeout, memorySize: 256,
      environment: { TABLE_NAME: table.tableName },
      tracing: h ? lambda.Tracing.ACTIVE : lambda.Tracing.DISABLED,
      logGroup: mkLog("WorkerLogs"),
    });
    workerFn.addEventSource(new sources.SqsEventSource(queue, { batchSize: 10, reportBat
    table.grantWriteData(workerFn);
```

```
    const dist = new cloudfront.Distribution(this, "Cdn", {
      defaultRootObject: "index.html",
      priceClass: cloudfront.PriceClass.PRICE_CLASS_100,
      httpVersion: cloudfront.HttpVersion.HTTP2_AND_3,
      defaultBehavior: {
        origin: origins.S3BucketOrigin.withOriginAccessControl(site),
        viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
        cachePolicy: cloudfront.CachePolicy.CACHING_OPTIMIZED,
      },
      additionalBehaviors: {
        "/api/*": {
          origin: new origins.HttpOrigin(cdk.Fn.select(2, cdk.Fn.split("/", stage.url)),
          viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.HTTPS_ONLY,
          allowedMethods: cloudfront.AllowedMethods.ALLOW_ALL,
          cachePolicy: cloudfront.CachePolicy.CACHING_DISABLED,
          originRequestPolicy: cloudfront.OriginRequestPolicy.ALL_VIEWER_EXCEPT_HOST_HEA
        },
      },
    });
```

The handlers are deliberately small. The API function validates (a SKU and a quantity from 1 to 100), puts the order on the queue and answers `202` with an id. The worker writes each order with a conditional put, so a redelivered message does nothing, and returns only the failed message ids.

```
import json
import os

import boto3
from botocore.exceptions import ClientError

table = boto3.resource("dynamodb").Table(os.environ["TABLE_NAME"])


def handler(event, context):
    """SQS batch: write each order once (a conditional put makes redelivery harmless); r
    failures = []
    for rec in event["Records"]:
        try:
            m = json.loads(rec["body"])
            table.put_item(Item={"orderId": m["orderId"], "sku": m["sku"], "quantity": m
                           ConditionExpression="attribute_not_exists(orderId)")
        except ClientError as e:
            if e.response["Error"]["Code"] != "ConditionalCheckFailedException":
                failures.append({"itemIdentifier": rec["messageId"]})
        except Exception:
            failures.append({"itemIdentifier": rec["messageId"]})
    return {"batchItemFailures": failures}
```

Synthesised and Linted for Real

Run here: `cdk synth -c hardened=false` and `-c hardened=true`, each exit 0, without credentials (synthesis needs none, as in *AWS Intermediate/Advanced* 5). The first run printed one warning from the CDK itself, and it is a real finding: I had set a minimum TLS version on the distribution, and the CDK said **"Ignoring 'minimumProtocolVersion': it has no effect without a custom 'certificate'. The distribution uses the CloudFront default certificate, whose security policy is fixed at TLSv1."** A TLS floor needs a custom domain and a certificate (*AWS Intermediate/Advanced* 10), which this design does not have, so I removed the property and the warning went away. Remember that the stack therefore accepts whatever the default certificate's policy accepts.

```
cat out/resources.out
```

```
hardened stack Orders: 32 resources, 22 types (draft OrdersDraft: 24 resources)
   1 AWS::ApiGatewayV2::Api
   1 AWS::ApiGatewayV2::Integration
   1 AWS::ApiGatewayV2::Route
   1 AWS::ApiGatewayV2::Stage
   1 AWS::CDK::Metadata
   1 AWS::CloudFront::Distribution
   1 AWS::CloudFront::OriginAccessControl
   5 AWS::CloudWatch::Alarm
   1 AWS::DynamoDB::Table
   2 AWS::IAM::Policy
   2 AWS::IAM::Role
   1 AWS::KMS::Key
   1 AWS::Lambda::EventSourceMapping
   2 AWS::Lambda::Function
   1 AWS::Lambda::Permission
   2 AWS::Logs::LogGroup
   1 AWS::S3::Bucket
   1 AWS::S3::BucketPolicy
   1 AWS::SNS::Topic
   2 AWS::SQS::Queue
   2 AWS::SQS::QueuePolicy
   1 AWS::SecretsManager::Secret
```

Both templates passed cfn-lint with two `W3005` warnings each (a `DependsOn` that the CDK adds although a `GetAtt` already enforces it; harmless, and I counted them as the baseline, not as findings). Eight Jest assertions on the stack pass: the queue's visibility timeout (185 seconds, from the Lambda/SQS page's rule of six times the function timeout plus the batch window of 5), the redrive count, partial-batch reporting with a concurrency cap, the table's settings, arm64 and tracing on both functions, five alarms with actions, `Retain` on stateful resources, and the origin access control with public access blocked. **These tests assert what I already believed** (I wrote the stack and the tests together), so they guard against regressions rather than prove the design right.

Run here: the handlers against *moto* in memory. Ten checks pass: a valid order returns 202, five kinds of invalid order return 400 and are not queued, the worker writes the order, a redelivery changes nothing, and a poison message among good ones is the only one reported as failed.

```
py flow_test.py
```

```
  ok   valid order -> 202 with an orderId
  ok   invalid order (quantity 0) -> 400
  ok   invalid order (quantity 101) -> 400
  ok   invalid order (missing sku) -> 400
  ok   invalid order (quantity 'x') -> 400
  ok   invalid order (not JSON) -> 400
  ok   only the valid order was queued
  ok   worker writes it, no failures
  ok   redelivery of the same message is harmless (still 1 item, no failure)
  ok   a poison message is reported alone

  all flow checks pass
```

That tests my code's logic. It says nothing about the real services: Lambda timeouts, IAM permissions for the function to use the KMS key, throttling and partial-batch handling in the event source mapping are all run-time behaviour that chapters 8 and 9 showed an emulator does not enforce.

The Review

What the framework says

From the documentation (read 2026-10-13): the framework has six pillars: operational excellence, security, reliability, performance efficiency, cost optimization and sustainability. Each has design principles, best-practice areas and numbered questions. What I read:

PILLAR	ITS DESIGN PRINCIPLES (NAMES AS PRINTED)	BEST-PRACTICE AREAS
Operational excellence (7)	Organize teams around business outcomes; Implement observability for actionable insights; Safely automate where possible; Make frequent, small, reversible changes; Refine operations procedures frequently; Anticipate failure; Learn from all operational events and metrics; Use managed services	Organization, Prepare, Operate, Evolve
Security (7)	Implement a strong identity foundation; Maintain traceability; Apply security at all layers; Automate security best practices; Protect data in transit and at rest; Keep people away from data; Prepare for security events	Security foundations, Identity and access management, Detection, Infrastructure protection, Data protection, Incident response, Application security

PILLAR	ITS DESIGN PRINCIPLES (NAMES AS PRINTED)	BEST-PRACTICE AREAS
Reliability (5)	Automatically recover from failure; Test recovery procedures; Scale horizontally to increase aggregate workload availability; Stop guessing capacity; Manage change through automation	Foundations, Workload architecture, Change management, Failure management
Performance efficiency (5)	Democratize advanced technologies; Go global in minutes; Use serverless architectures; Experiment more often; Consider mechanical sympathy	Architecture selection, Compute and hardware, Data management, Networking and content delivery, Process and culture
Cost optimization (5)	Implement Cloud Financial Management; Adopt a consumption model; Measure overall efficiency; Stop spending money on undifferentiated heavy lifting; Analyze and attribute expenditure	Practice Cloud Financial Management, Expenditure and usage awareness, Cost-effective resources, Manage demand and supply resources, Optimize over time
Sustainability (6)	Understand your impact; Establish sustainability goals; Maximize utilization; Anticipate and adopt new, more efficient hardware and software offerings; Use managed services; Reduce the downstream impact of your cloud workloads	Region selection, Alignment to demand, Software and architecture, Data management, Hardware and services, Process and culture

The outline asked for the framework's own questions. **I read only some of them**, from the area pages that matter most for this system: SEC 2 and 3 (identities and permissions for people and machines), SEC 7 to 9 (data classification, at rest, in transit), REL 3 to 5 (service architecture and distributed-system interactions), REL 9 to 13 (backup, fault isolation, component failure, testing, disaster recovery), OPS 8 to 10 (observability, operations health, events), PERF 2 (compute), COST 5 to 8 (service selection, size, pricing models, data transfer) and SUS 5 (hardware and services). I did not read the question pages for the other areas, so this review is not a complete framework review and must not be reported as one. The pillar pages also say, correctly, that answering questions is a conversation about people and process, which a template cannot do.

What a template can and cannot show

Run here: `wa_review.py` reads a synthesised template and applies 36 scored checks, each tagged with the pillar and the design principle or earlier chapter it comes from, plus 3 informational lines. **I wrote the rules before running them on either template**, from the pages above and chapters 3, 5, 8, 9, 10 and 11. They check things like: log groups have a

retention period, an alarm exists for the dead-letter queue and for each function's errors, every bucket, queue, table and secret is encrypted, policies deny non-TLS requests, no IAM statement uses `Resource: *`, the queue's visibility timeout is at least six times the worker's timeout, the consumer reports partial batch failures, tables have point-in-time recovery and deletion protection, stateful resources are retained, the API stage throttles, functions are arm64, tables bill per request, and every taggable resource has a `project` tag. A sample of the rules:

```
    add("OPS1", "Operational excellence", len(lgs) >= len(L) and all("RetentionInDays" i
        f"every Lambda has a log group with a retention period ({len(lgs)} log groups fo
    add("OPS2", "Operational excellence", has_alarm(t, "ApproximateNumberOfMessagesVisib
    add("OPS3", "Operational excellence", sum(1 for a in alarms(t) if a["Properties"].ge
        "each Lambda function has an Errors alarm")
    add("OPS4", "Operational excellence", has_alarm(t, "ApproximateAgeOfOldestMessage",
    add("OPS5", "Operational excellence", has_alarm(t, "5xx", "AWS/ApiGateway"), "an ala
    add("OPS6", "Operational excellence", all(a["Properties"].get("AlarmActions") for a
    add("OPS7", "Operational excellence", any(v["Type"] == "AWS::CodeDeploy::DeploymentG
```

The result on the draft and on the hardened template:

```
py wa_review.py cdk.out.draft/OrdersDraft.template.json
```

```
FAIL  OPS1   Operational excellence   every Lambda has a log group with a retention peri
pass  OPS2   Operational excellence   an alarm watches the dead-letter queue
FAIL  OPS3   Operational excellence   each Lambda function has an Errors alarm
FAIL  OPS4   Operational excellence   an alarm shows the consumer falling behind (age of
FAIL  OPS5   Operational excellence   an alarm watches HTTP API 5xx
pass  OPS6   Operational excellence   every alarm has an action (1 alarms)
FAIL  OPS7   Operational excellence   gradual deployment for the functions (small, rever
pass  SEC1   Security                 data at rest is encrypted on every bucket, queue,
FAIL  SEC2   Security                 data in transit: a policy denies non-TLS requests
FAIL  SEC3   Security                 traceability: X-Ray tracing is active on every fun
pass  SEC4   Security                 no IAM statement with Resource '*' (0 found: [])
pass  SEC5   Security                 no long-term credentials (no IAM users or access k
FAIL  SEC6   Security                 defence in depth at the edge: a web ACL is attache
pass  SEC7   Security                 every CloudFront behaviour redirects or requires H
pass  SEC8   Security                 all four public-access blocks are on for every buc
pass  SEC9   Security                 automatic rotation is on for every KMS key
FAIL  SEC10  Security                 traceability at the edge: CloudFront access loggin
pass  REL1   Reliability              every work queue has a dead-letter queue
FAIL  REL2   Reliability              queue visibility timeout is at least 6 x the worke
pass  REL3   Reliability              the queue consumer reports partial batch failures
```

```
pass  REL4   Reliability             point-in-time recovery is on for every table
FAIL  REL5   Reliability             deletion protection is on for every table
pass  REL6   Reliability             every bucket, table, key and secret has DeletionPo
FAIL  REL7   Reliability             the HTTP API stage throttles requests (protects th
FAIL  REL8   Reliability             the queue consumer has a concurrency cap (protects
FAIL  REL9   Reliability             every bucket is versioned
info  REL10  Reliability             a second Region or a recovery objective: not expre
pass  PERF1  Performance efficiency   functions run on arm64 (the Lambda pricing page li
pass  PERF2  Performance efficiency   serverless compute only (no instances to size)
pass  PERF3  Performance efficiency   the API behaviour uses the managed CachingDisabled
pass  COST1  Cost optimization        tables use on-demand billing (a consumption model)
pass  COST2  Cost optimization        CloudFront price class is limited (PriceClass_100)
pass  COST3  Cost optimization        a 'project' tag is on every taggable resource (8 o
FAIL  COST4  Cost optimization        log retention bounds storage cost (same facts as O
info  COST5  Cost optimization        KMS keys: 1 (the KMS page prices a key at $1 a mon
pass  SUS1   Sustainability           efficient hardware: arm64 functions
pass  SUS2   Sustainability           managed, shared services rather than servers to ke
FAIL  SUS3   Sustainability           data you stop keeping stops costing energy: bounde
info  SUS4   Sustainability           Region selection and per-transaction impact measur

20 of 36 scored checks pass (3 informational)
```

```
py wa_review.py cdk.out.hard/Orders.template.json
```

```
pass  OPS1   Operational excellence   every Lambda has a log group with a retention peri
pass  OPS2   Operational excellence   an alarm watches the dead-letter queue
pass  OPS3   Operational excellence   each Lambda function has an Errors alarm
pass  OPS4   Operational excellence   an alarm shows the consumer falling behind (age of
```

```
diff of the two reviews
```

```
check   draft  hardened
OPS1    FAIL   pass
OPS3    FAIL   pass
OPS4    FAIL   pass
OPS5    FAIL   pass
SEC2    FAIL   pass
SEC3    FAIL   pass
SEC4    pass   FAIL
REL2    FAIL   pass
```

```
REL5    FAIL   pass
REL7    FAIL   pass
REL8    FAIL   pass
REL9    FAIL   pass
COST4   FAIL   pass
SUS3    FAIL   pass

14 checks changed state; still FAIL in both: OPS7, SEC6, SEC10
```

The draft passed 20 of 36; the hardened version passes 32 of 36. The fixes were ordinary: log retention (and the log groups to hold it), alarms for the functions, the queue's age and the API's 5xx, TLS-only policies on the bucket and queues, X-Ray tracing, a visibility timeout of 185 instead of 30, deletion protection on the table, a throttle on the API stage, a concurrency cap on the consumer and versioning on the site bucket. **One fix created a new finding:** turning on X-Ray tracing made the CDK add two statements with `Resource: *`, so SEC4 went from pass to FAIL. That is the framework's trade-off language in a small way: traceability against least privilege. I have not read whether X-Ray actions can be scoped to a resource, so I accepted it and recorded it.

A caution about the score. The rules are mine, and I fixed the stack until it satisfied them. A pass means "conforms to my checklist", not "is well-architected". Each pillar has principles that no template can show: that someone owns the alarm (OPS 10), that restores work (REL 9, REL 12), that the recovery objectives were set from business needs (REL 13), that the data was classified (SEC 7), that cost is reviewed monthly, that the Region was chosen for its impact (the sustainability pages on Region selection were not read).

The four that still fail, and the decisions behind them

CHECK	FINDING	DECISION
OPS7	The functions are deployed all at once; no alias and CodeDeploy group.	Fixable with a `lambda.Alias` and a `LambdaDeploymentGroup` and the arithmetic of *AWS Intermediate/Advanced* 11 (a canary shorter than the alarm's detection time protects nothing). Left open to keep the design small.
SEC4	Two X-Ray statements with `Resource: *`, added by the CDK for tracing.	Accepted for now; the alternative is to turn tracing off, which fails SEC3.

CHECK	FINDING	DECISION
SEC6	No web ACL on the distribution.	An open decision about cost ($5 a month per web ACL and $1 per rule from the pricing page, plus request charges) and about where a CloudFront web ACL must be created, which I did not read. Rate limiting at the API stage covers part of it.
SEC10	No CloudFront access logs.	Not added: logging needs a log bucket with its own policy, and I did not read the page on the requirements.

Three informational items stay open too: no second Region and no stated recovery objectives (REL 13 asks for them), a KMS key at about $1 a month (the KMS page), and Region selection for sustainability (not read). A finding with an owner and a date is a decision; a finding nobody wrote down is a risk.

Does the Review Catch Regressions?

A review that runs once is a document; one that runs on every change is a guardrail. **Run here:** 29 one-change edits to the hardened template, as if someone had edited the stack (visibility timeout back to 30, redrive removed, recovery off, deletion protection off, encryption removed, a public-access block off, rotation off, `allow-all` on a behaviour, an alarm without an action, an IAM user added, and so on), each run through cfn-lint and through the review. The review counts as catching a change when a check that passed now fails.

```
py mutate_wa.py
```

```
change                                                  lint  review
(control: unchanged hardened template)                     0       0   baseline FA
work queue visibility timeout 185 -> 30                    0       1
work queue loses its redrive policy                        0       1
worker timeout 30 -> 300 (queue timeout now too short)     1       1   E3505
table point-in-time recovery off                           0       1
table deletion protection off                              0       1
table encryption removed                                   0       1
table DeletionPolicy Retain -> Delete                      0       1
site bucket BlockPublicPolicy false                        0       1
KMS key rotation off                                       0       1
CloudFront default behaviour allow-all                     0       1
CloudFront price class all                                 0       1
DLQ alarm loses its action                                 0       1
```

```
DLQ alarm removed                                          0      1
consumer loses partial-batch reporting                     0      1
consumer concurrency cap removed                           0      1
worker on x86_64                                           0      1
API function tracing PassThrough                           0      1
API stage throttle removed                                 0      1
queue policy's TLS deny removed                            0      1
an IAM user and access key added                           0      1
a statement with Action * on Resource * added              0      0
API log group retention removed                            0      1
table tags removed                                         0      1
table billing PROVISIONED without capacity                 1      1   E3639
worker memory 256 -> 64                                    1      0   E3034
worker timeout 900 -> 901                                  1      1   E3505,E3717
work queue retention 14 days -> 0 seconds                  1      0   E3034
API route method POST -> POSTT                             0      0
event source batch size 10 -> 11                           0      0

29 changes: cfn-lint 5, review 24, both 3, neither 3
```

The review flagged 24 of 29; cfn-lint flagged 5; both 3; neither 3. cfn-lint caught only range and consistency errors (a memory size of 64, a retention of 0 seconds, a provisioned table without capacity, a timeout of 901 seconds) and one meaning rule that I did not expect: `E3505` reported that the queue's visibility timeout (185) was less than the function timeout (300). My chapter 9 result for a differently shaped template was that cfn-lint 1.57.2 missed that kind of mismatch; I did not investigate why this one was caught. It does not know the six-times rule, so it passed a visibility timeout of 30. Three changes were missed by both: a statement with `Action: *` on `Resource: *` added to the API function's policy, a route method typo (`POSTT`) and a batch size of 11. **The first of those is a design flaw in my harness, not just in the tools:** SEC4 already failed on the baseline because of X-Ray, so a new wildcard statement could not turn a pass into a fail. A check that is already failing cannot report a regression. If you accept a finding, put the accepted items in an allowlist so that anything else still fails.

What It Might Cost

The pricing calculator was not used. From the pricing pages (2026-10-13; mostly worked examples, US East (N. Virginia) where a Region is named): HTTP API $1.00 per million requests; Lambda $0.20 per million requests and $0.0000133334 per GB-second on Arm (that figure from

one example), with 1 million requests and 400,000 GB-seconds free; DynamoDB on demand $0.625 per million writes, $0.25 per GB stored and $0.20 per GB-month for point-in-time recovery; CloudWatch alarms $0.10 each (10 free), logs $0.50 per GB (5 GB free); KMS $1 per key and Secrets Manager $0.40 per secret. I could not read the SQS request price (only a 1 million requests free tier) or CloudFront's pay-as-you-go prices (the flat-rate plans are Free $0, Pro $15, Business $200 and Premium $1,000 per distribution). A sketch for three volumes, with the assumptions printed:

```
py cost12.py
```

```
== 100,000 orders a month (1 KB item, batch of 10, 100 ms API / 200 ms per message, 256 M
  API Gateway HTTP API   $     0.10
  Lambda requests        $     0.00   (free tier applied; without it $0.02)
  Lambda duration        $     0.00   (7,500 GB-s; without free tier $0.10)
  DynamoDB (writes, 0.5 GB stored, PITR)  $0.29
  CloudWatch alarms      $     0.00   (5 alarms inside the 10-alarm free tier; $0.50 witl
  CloudWatch logs        $     0.00   (0.2 GB ingested; $0.10 without the 5 GB free tier
  KMS key $1.00 + secret $0.40   (request charges not estimated)
  priced lines total     $     1.79   (without any free tier $2.51)
  SQS requests (not priced): 120,000   -> free tier covers 1,000,000
== 1,000,000 orders a month (1 KB item, batch of 10, 100 ms API / 200 ms per message, 25(
  API Gateway HTTP API   $     1.00
  Lambda requests        $     0.02   (free tier applied; without it $0.22)
  Lambda duration        $     0.00   (75,000 GB-s; without free tier $1.00)
  DynamoDB (writes, 0.5 GB stored, PITR)  $0.85
  CloudWatch alarms      $     0.00   (5 alarms inside the 10-alarm free tier; $0.50 witl
  CloudWatch logs        $     0.00   (2.0 GB ingested; $1.00 without the 5 GB free tier
  KMS key $1.00 + secret $0.40   (request charges not estimated)
  priced lines total     $     3.27   (without any free tier $5.97)
  SQS requests (not priced): 1,200,000   -> free tier covers 1,000,000
== 10,000,000 orders a month (1 KB item, batch of 10, 100 ms API / 200 ms per message, 2
  API Gateway HTTP API   $    10.00
  Lambda requests        $     2.00   (free tier applied; without it $2.20)
  Lambda duration        $     4.67   (750,000 GB-s; without free tier $10.00)
  DynamoDB (writes, 0.5 GB stored, PITR)  $6.47
  CloudWatch alarms      $     0.00   (5 alarms inside the 10-alarm free tier; $0.50 witl
  CloudWatch logs        $     7.50   (20.0 GB ingested; $10.00 without the 5 GB free ti
  KMS key $1.00 + secret $0.40   (request charges not estimated)
  priced lines total     $    32.04   (without any free tier $40.58)
  SQS requests (not priced): 12,000,000   -> free tier covers 1,000,000
```

```
NOT in these totals: CloudFront (flat-rate plans read: Free $0, Pro $15, Business $200, P
S3 storage and requests, X-Ray, SNS, KMS request charges, data transfer, a WAF web ACL ($
```

The priced lines stay small at these volumes (about $1.79, $3.27 and $32.04 a month for 100,000, a million and ten million orders, free tiers applied; the free tiers are not available to every account, and the "without free tier" figures are printed beside them). They **leave out** CloudFront, SQS beyond its free million requests, S3, X-Ray, SNS, KMS request charges, data transfer and any WAF. At ten million orders the logs ($7.50) and the API ($10) matter more than the table. Cost optimization in the framework starts with measuring: tag everything (check COST3 enforces the `project` tag) and look at the bill by tag.

If You Have an Account (not run)

Level 2: set a budget alert first (*AWS Fundamentals* 9). The commands for `cdk bootstrap`, `diff`, `deploy` and `destroy` are in *AWS Intermediate/Advanced* 5. Specific to this stack: the key, the site bucket, the table and the secret are `RemovalPolicy.RETAIN`, so `cdk destroy` leaves them behind and you must delete them yourself (the key's deletion is scheduled, not immediate; I did not read the waiting-period details here); deletion protection on the table must be turned off before it can be deleted; the site is empty until you upload an `index.html`; and the first deployment of a distribution takes a while. I give no copy-paste commands for the manual deletions because I did not read the CLI pages for them. Nothing was deployed, so none of this is verified.

Cost and Clean-Up

Nothing in this chapter created anything in AWS. On this machine: the scratch folder `C:/lwaws/c12` (a CDK project whose `node_modules` is a directory junction to chapter 5's, two `cdk.out` folders and a Python environment). Delete the junction itself, not through it, or delete chapter 5's folder first. moto ran in-process; Docker was not used.

WHAT WAS AND WAS NOT CHECKED

Run here: two real syntheses and cfn-lint runs; 8 Jest assertions; a 10-check handler flow test on moto; a 36-check review of both templates; 29 regression edits against cfn-lint and the review; a cost sketch.
From the documentation, not run: the framework's pillars, principles and the questions I list; every

service's run-time behaviour. **Not read:** the question pages for most best-practice areas, the Operational Excellence, Security, Reliability, Cost Optimization and Sustainability whitepapers beyond their introductions, the AWS Pricing Calculator, SQS and CloudFront pay-as-you-go prices, how a CloudFront web ACL must be created, X-Ray resource scoping, KMS key deletion details. **Not checked at all:** any deployment, any real request, any restore, any alarm firing.

Course Complete

Twelve chapters, and one question asked of every example: how do you know? The levels were: level 1 run on this machine without an account, level 2 needs an account, level 3 from the documentation. The recurring results, in the order you met them:

CHAPTER	WHAT THE TOOLS COULD CHECK	WHAT THEY MISSED
1 to 3 Accounts, IAM, KMS	Policy evaluation models tested on documented cases; policy linting	Much of the real evaluation (resource policies with wildcard principals, RCPs); how plain negated condition operators combine values was unverified
4 CloudFormation	cfn-lint caught 18 of 22 mistakes	Four meaning errors (an `Fn::Sub` in a condition, a bad mapping key)
5 and 6 CDK, containers	`cdk synth` for real; containers measured in Docker	`synth` exited 0 on an invalid Fargate size (only cfn-lint flagged it)
7 EKS	eksctl's schema caught 7 of 10 config mistakes; cfn-lint 10 of 15	The kind-cluster experiments could not run (Docker stopped)
8 API Gateway	`sam local` enforced routing, authorizers and CORS	Request validation, API keys, throttling and the JWT authorizer were not enforced locally; cfn-lint caught 10 of 25 (1.53.3)
9 Messaging	moto matched SQS and SNS behaviour; Step Functions Local reproduced retry timing exactly	moto enforced 6 of 12 SQS limits; cfn-lint caught 15 of 30
10 Edge and DNS	Models of the documented tables; a real DNS parser	moto accepted all 9 invalid inputs; cfn-lint caught 15 of 32
11 Delivery	actionlint caught 14 of 26 workflow mistakes; a trust-policy model showed what a policy admits	cfn-lint caught 8 of 26 and 5 of 16; moto accepted oversize parameters

CHAPTER	WHAT THE TOOLS COULD CHECK	WHAT THEY MISSED
12 Capstone	A 36-check review caught 24 of 29 regressions	cfn-lint caught 5 of 29; the review's own accepted finding hid a regression

Three habits carry over. **Linters check ranges and types, not meaning**: keep your own checks for the relationships between resources, and write them from the documentation before you see the results. **Emulators are lax**: use them to test that your code calls an API and handles its answers, never to learn what is valid. **Say how you know**: a statement from the documentation, a model of the documentation, a test against an emulator and an observation of the real service are four different things, and this course labelled which was which. What it could not do is the part that needs your account: deploy, watch a real alarm fire, restore a table, read a real bill.

Where next. *Kubernetes Intermediate/Advanced* continues the container material that chapter 7 sketched for EKS; *Terraform / Infrastructure as Code* is the other way to write what chapters 4 and 5 wrote in CloudFormation and CDK; *Observability* goes deeper into the alarms, logs and traces that this chapter only counted. A third AWS course (data and analytics, machine learning or networking in depth) can follow the same pattern.

Practice

EXERCISE 1

Define the order-intake system as a CDK stack in TypeScript with one `hardened` flag. Synthesise both versions without credentials, lint them, count the resources, write eight Jest assertions, and run the two Lambda handlers against an emulator, including a redelivered message and a poison message.

View solution

EXERCISE 2

Write a checker that reviews a synthesised template against the six Well-Architected pillars (write the rules before you run them), run it on the draft and the hardened stack, explain every check that changed state (including the one that got worse) and record the decisions behind the checks that still fail.

View solution

EXERCISE 3

Make 29 one-change regressions of the hardened template and score cfn-lint and your review on them; work out why the new wildcard statement was not caught and design an allowlist for accepted findings. Sketch the monthly cost for three volumes from the pricing pages, with free tiers shown both ways, and list what you could not price.

View solution

CHAPTER 12 QUICK REFERENCE

- One construct, one `hardened` flag: the difference between draft and hardened templates is exactly what the review found (20 of 36 checks passed, then 32 of 36)
- The six pillars and their principles are on the framework pages; a template can show encryption, retention, alarms, throttles, deletion protection and tags, not ownership, restores, recovery objectives or data classification
- A TLS minimum on CloudFront needs a custom certificate: the CDK says the default certificate's policy is fixed at TLSv1
- Fixing one finding can create another: tracing added wildcard X-Ray statements and SEC4 went from pass to FAIL
- A check that already fails cannot report a regression: keep accepted findings in an allowlist so anything else still fails
- On 29 regressions the review caught 24 and cfn-lint 5; both caught 3 and neither caught 3; cfn-lint's `E3505` compares the queue's visibility timeout with the function timeout but not the six-times rule
- Cost sketch (priced lines only, free tiers applied): about $1.79, $3.27 and $32.04 a month at 100,000, 1 million and 10 million orders; CloudFront, SQS beyond the free tier, S3, X-Ray, SNS and data transfer are not in it
- Retained resources (key, bucket, table, secret) survive `cdk destroy`; delete them by hand, and turn off the table's deletion protection first

COURSE COMPLETE

This is the last chapter of *AWS Intermediate/Advanced*. Pointers: *Kubernetes Intermediate/Advanced*, *Terraform / Infrastructure as Code* and *Observability*.