MODULE 11 · STUDY NOTES

Automating Your Architecture

Why manual processes create risk, how infrastructure as code fixes it, and how AWS CloudFormation models, deploys, and manages your resources — with Quick Starts, Amazon Q Developer, and Well-Architected automation practices

Most organizations start on AWS by hand — someone creates an S3 bucket in the console or launches an EC2 instance running a web server, then adds more resources over time. That works until it doesn't: manual processes are slow, error prone, and impossible to reproduce reliably, leaving you with no version control, no audit trail, and configurations that drift apart. This module is the answer. It moves from the business case for automation, to the industry practice of infrastructure as code (IaC) — writing a template that captures your infrastructure's desired state — to the AWS service that makes IaC concrete: AWS CloudFormation, which turns a template into a managed collection of resources called a stack. It then adds the tools that make CloudFormation faster to adopt (Quick Starts and the AI coding assistant Amazon Q Developer) and closes by mapping everything back to the AWS Well-Architected Framework. Throughout, the cloud architect works backward from the business need and weighs how automation decisions affect performance and cost.

LEARNING OBJECTIVES

- Recognize when to use architecture automation and why
- Identify how to use infrastructure as code (IaC) as a strategy for provisioning and managing cloud resources
- Identify how to model, create, and manage a collection of AWS resources by using AWS CloudFormation
- Identify how to use AWS Quick Start CloudFormation templates to set up an architecture
- Identify uses of Amazon Q Developer
- Use the AWS Well-Architected Framework principles when designing automation strategies

11.1 Reasons to automate

Many organizations begin using AWS by **manually** creating a resource — an S3 bucket, or an EC2 instance running a web server — and then adding more resources by hand as new needs arise. Soon it becomes challenging to manage and maintain everything this way. **Manual processes are error prone, unreliable, and inadequate to support an agile business.** They also tie up highly skilled people on repetitive configuration work when that time could go to higher-value activities. Before building an architecture by hand, a cloud architect should ask the hard questions: where should effort go, into design or implementation? How will you update production servers, roll deployments out across multiple Regions, and roll back to the last known good version when something breaks? How will you debug deployments and manage dependencies across systems — and is it realistic to do all of that manually at all?

FOUR RISKS OF MANUAL PROCESSES

- **No repeatability at scale** — manually adding features and replicating deployments to multiple Regions doesn't scale; there may not be enough people to do it.
- **No version control** — an environment built from scratch has no inherent way to roll the production stack back to a previous version in an emergency.
- **No audit trail** — giving people the ability to manually control and edit environments is dangerous for compliance and security; you cannot easily track configuration changes at the resource level.
- **Inconsistent configurations** — consistency is critical to minimizing risk, and only automation reliably keeps configurations matching across many instances.

The fix is **automation**, which addresses the full lifecycle of setup, configuration, deployment, and support of infrastructure and the applications on it. Automation **reduces manual intervention or access** to production, delivers **reproducible environments** in a standardized, repeatable configuration, **improves productivity** across testing, releasing, patching, troubleshooting, and bug fixing, and **automates testing and scaling** so environments respond elastically to customer demand.

COMMON PITFALL

Manual builds are a natural starting point, not a viable long-term strategy. The recurring theme of the module is that a cloud architect works backward from the business need — the goal is to reduce the risk of manual processes and maximize the benefit of reproducible environments, and those decisions will evolve over time as the business changes and as you optimize cost and performance.

11.2 Using infrastructure as code

Historically, infrastructure was provisioned with a combination of scripts and manual processes, so new environments were not always repeatable, reliable, or consistent. **Infrastructure as code (IaC)** replaces that with a programmatic, descriptive approach: you provision and manage cloud resources by defining them in a **template file that is both human-readable and machine-consumable**. Crucially, you define the infrastructure's *desired state* without spelling out every step to reach it. Organizations adopt IaC to control costs, reduce risks, and respond with speed to new business opportunities — and because it lets them **replicate, redeploy, and repurpose** infrastructure reliably and consistently. The key benefits are **reusability, repeatability, and maintainability**.

WORKED EXAMPLE – ONE TEMPLATE, THREE ENVIRONMENTS

A single template (or a combination of templates) can build the same complex environment repeatedly — for instance, all the resources for a web application: a web server, a database, and networking rules. Use one template to create three stacks: stack 1 is the **development** environment (flexible and dynamic, for prototyping and debugging), stack 2 is the **test** environment (simulates production to test functionality, performance, and reliability), and stack 3 is **production** (available to end users). Because all three come from one template, you gain confidence that what passed in test will behave the same in production, and you minimize the risk that test is configured differently. Need a change? Push it to the template and it propagates to every stack. Done with the web application? Delete the stack and all its related resources are removed — reducing cost and keeping the account clean.

AWS CloudFormation is the AWS IaC service at the center of this module. It provides a simplified way to **model, create, and manage a collection of AWS resources** — a collection is called a **stack** — and there is **no extra charge** beyond the resources you create. You author a document that describes what your infrastructure should be (a *model*), and CloudFormation uses that model to create the reality in your account. You can create, update, and delete stacks,

giving orderly and predictable provisioning. Because the template is just code, you can author it in any editor, check it into a version control system such as GitHub, and review it with teammates before deploying — and if you check it in, you can always delete a stack, check out an older version of the document, and recreate the stack from it, which is what gives you essential **rollback** capabilities.

TABLE 11.1 AWS IaC services that use CloudFormation

SERVICE	WHAT IT DOES	RELATIONSHIP TO CLOUDFORMATION
AWS Elastic Beanstalk	Fully managed service that launches an AWS environment from your uploaded application code (Go, Java, .NET, Node.js, PHP, Python, Ruby); handles provisioning, load balancing, scaling, and health	Provisions resources such as EC2 via CloudFormation
AWS Quick Starts	Automated reference architectures that automate hundreds of manual procedures using AWS best practices	Based on CloudFormation templates
AWS SAM	Toolkit for serverless apps: shorthand-syntax templates plus the SAM CLI	An extension of CloudFormation; deploys directly to it
AWS Amplify	Framework for full-stack web and mobile apps with less coding and less backend knowledge	Amplify deployments build CloudFormation stacks
AWS CDK	Open source framework to define infrastructure using familiar programming languages	CDK projects synthesize into CloudFormation templates

Whichever of these you choose, **behind the scenes AWS uses CloudFormation to deploy your resources.** Each service abstracts away a different amount of the coding involved, so the decision comes down to the relative level of **convenience and control** you need and the **skills of your team**.

> **COMMON PITFALL**
>
> Choose the IaC solution to fit the use case — not the other way around. Elastic Beanstalk, Amplify, SAM, and the CDK all sit at different points on a convenience-versus-control spectrum, but they share the same engine. If a question hinges on which service deploys resources under the hood, the answer for all of them is CloudFormation.

11.3 Customizing with CloudFormation

CloudFormation follows a consistent four-step flow. First, **define** the AWS resources you want in a template — for example, a few EC2 instances, a load balancer, an Auto Scaling group, and a Route 53 hosted zone — writing it from scratch or starting from a prebuilt template. Second, **upload** the template to CloudFormation, or store it in Amazon S3 and point CloudFormation to it. Third, run the **create stack** action: CloudFormation reads the template and creates the resources in your account, across **multiple services in a single Region**, as a running environment. Finally, the **stack retains control** of what it created and acts like a handle to those resources, so you can later update the stack, detect drift, or delete it. (Where CloudFormation lacks a specific feature, you can invoke a Lambda function during the build that calls the AWS SDK for full API coverage.)

YAML	JSON
choose for a human-readable format ▪ Optimized for readability — fewer lines (no braces, fewer quotation marks) ▪ Natively supports embedded comments ▪ Easier to debug — no hunting for a missing comma or brace	*choose for cross compatibility* ▪ More widely used by other computer systems, such as APIs ▪ Usually less complex to generate and parse ▪ Comments are possible, but require the metadata attribute

A template can be authored in **either JSON or YAML** — same template, different notation — so pick the language that suits your use case, business needs, and experience. It is recommended that you **treat templates as source code** and store them in a code repository. You can also author templates in **AWS CloudFormation Designer**, a graphical interface in the console: drag resources from the Resource types pane onto a canvas, edit details in an integrated JSON and YAML editor, see the interrelationships between resources, and convert a valid template from JSON to YAML or back. A Messages pane reports success, conversion results, and validation errors.

When you write a template, you choose which sections to include based on your workload. The **only required section is `Resources`**; all others are optional. A fixed order isn't required, but a logical order helps because values in one section can reference values from an earlier one.

TABLE 11.2 CloudFormation template sections, in suggested order

SECTION	REQUIRED?	WHAT IT DOES
AWSTemplateFormatVersion	Optional	The template format version the template conforms to (independent of the API/WSDL version)
Description	Optional	A text string describing the template; must follow the format version
Metadata	Optional	Objects that provide additional information about the template
Parameters	Optional	Values to pass to the template at runtime, when you create or update a stack
Rules	Optional	Validates a parameter or combination of parameters during create or update
Mappings	Optional	A lookup table of keys and associated values for conditional parameter values
Conditions	Optional	Controls whether resources are created or properties assigned during create/update
Transform	Optional	For serverless applications, specifies the version of AWS SAM to use
Resources	REQUIRED	The stack's resources and their properties (e.g., an EC2 instance or an S3 bucket)
Outputs	Optional	The values returned when you view your stack's properties

WORKED EXAMPLE – A RESOURCES + OUTPUTS TEMPLATE

A minimal template to create an EC2 instance uses two sections. In `Resources` (required), a resource named Ec2Instance of `Type: AWS::EC2::Instance` is declared with statically defined properties (ImageId such as ami-9d23aeea, InstanceType such as m3.medium) plus a `Ref` to a KeyPair parameter. In `Outputs`, an InstanceId output returns the value `Ref` Ec2Instance. After the stack is created, you can read that output in the console, by running the `aws cloudformation describe-stacks` CLI command, or through the SDKs — a common use is returning the instance ID or the public IP address of an EC2 instance. The flow of the template is identical whether you write it in JSON or YAML.

Two features make updates safe and intentional. **Conditions** let one template serve different environments: the same template can build a production environment across **two Availability Zones** and a development environment in **one Availability Zone**, while keeping the same application binaries, language version, and database version so the app behaves identically. **Change sets** let you preview changes before you implement them — create a change set from your proposed edits, view exactly which settings and resources will change, then run the change set to apply it. Pair change sets with the `DeletionPolicy` attribute to preserve (or back up) a resource when its stack is deleted or updated; a resource with no `DeletionPolicy` is deleted.

Drift detection catches the opposite problem — when reality diverges from the model. Consider a stack that created an application environment. Later, someone manually adds an inbound TCP port to the stack's security group using the EC2 console, outside CloudFormation. Running **Detect Drift** from the Stack actions menu reports the drift status of each supported resource: every resource returns **IN_SYNC** except the security group, which returns **MODIFIED** with details. Note that deleting a stack that has drift does not clean up the drifted change, and unresolved dependencies can even cause the delete to fail.

COMMON PITFALL

The right way to change a stack's resources is through the template. In the drift scenario, the better approach is to modify the security group settings **in the CloudFormation template** and run **Update Stack** — CloudFormation updates the security group and keeps the model deployment synchronized with the actual deployment. Editing the resource directly in the console is what created the drift in the first place.

TABLE 11.3 Scoping and organizing templates

CATEGORY	TYPICAL RESOURCES
Frontend services	Web interfaces, mobile access, and analytics dashboards
Backend services	Search, payments, reviews, and recommendations
Shared services	CRM databases, common monitoring, alarms, subnets, and security groups
Network	VPCs, internet gateways, VPNs, and NAT devices
Security	IAM policies, users, groups, and roles

As you use more templates, define a **strategy** for the scope of each one — group tightly connected components together, much as you would organize a large enterprise application into parts. When the same components appear in multiple templates, factor them into dedicated templates and reference them as **nested stacks** (a stack created as part of another stack). However you scope them, treat all templates as code that needs version control and store them in a source control system.

WORKED EXAMPLE – THE GUIDED AND CAFÉ LABS

The module's labs deploy in **layers**. In the guided lab, one template deploys a VPC networking layer and a second deploys an application layer that **references** it; you then explore the templates in Designer and delete a stack that has a deletion policy. The networking stack exports two outputs — the VPC ID and the public subnet ID — so other stacks build EC2 instances and security groups inside them. The café challenge lab goes further: it stores templates in AWS CodeCommit, uses Git to invoke AWS CodePipeline to create or update stacks, and uses conditions to build different infrastructures from environment variables. Duplicating the application layer to the Oregon (us-west-2) Region first requires a `cafe-oregon` EC2 key pair, because the instance's KeyName references a RegionMap Mappings entry — a reminder that some resources are Region-specific.

11.4 Using AWS Quick Starts

AWS Quick Starts are **gold-standard deployments** — automated reference architectures built by AWS solutions architects and partners, based on AWS **best practices for security and high availability**. Each one provides a CloudFormation template for a full solution, so it can provision an entire architecture in your account in **less than an hour** — often in minutes — that would otherwise take hundreds of discrete manual procedures. Because Quick Starts use IaC with CloudFormation as the foundation, you get well-architected environments **without building your own template**, and because they are open source and modular, you can use them for experimentation or as the basis for your own architectures.

HOW A QUICK START IS PACKAGED AND USED

- Each Quick Start consists of a **CloudFormation template** and a **deployment guide**.
- The guide covers deployment options, how to configure the deployment, **security considerations**, and **estimated costs**.
- You customize the deployment to match your needs and create the stack; it finishes in minutes or a few hours.
- Even if you don't deploy one, reviewing a few shows useful patterns and practices — you can borrow a section and embed it in your own template.

WORKED EXAMPLE – SERVERLESS IMAGE HANDLER

This Quick Start deploys a serverless architecture for cost-effective image processing, with dynamic content delivery, content moderation, and smart cropping. The CloudFormation template deploys an **Amazon CloudFront** distribution as a caching layer to cut cost and latency; **Amazon API Gateway** provides endpoints and initiates a **Lambda** function; the Lambda function retrieves an image from the customer's existing S3 bucket and returns a modified version; a separate S3 bucket stores logs; **AWS Secrets Manager** holds the secret used to validate an image URL signature; and **Amazon Rekognition** analyzes images for the smart-crop and content-moderation features. The whole architecture deploys in under an hour.

COMMON PITFALL

AWS Quick Starts are not the same as AWS Marketplace AMIs. Marketplace AMIs are **single-vendor** solutions launched from the EC2 console that run on EC2 instances. Quick Starts are **modular and more customizable** reference architectures that might — or might not — use Amazon EC2 at all.

11.5 Customizing with Amazon Q Developer

Writing infrastructure as code has real challenges: **human error** (people introduce mistakes that are hard to find), **differing skill levels** and coding styles across developers, the **size and complexity** of templates (even a basic three-tier architecture gets complicated fast, and you can get lost in the details of an existing template or Quick Start), and **security vulnerabilities** that slip in when the focus is on delivering error-free code rather than secure code. **Amazon Q Developer** is AWS's response — a **generative AI-powered coding assistant** for developers and IT professionals that improves productivity by recommending code from natural-language comments and prior code.

WHAT AMAZON Q DEVELOPER DOES

- Generates code and helps you understand, build, extend, and operate AWS applications.
- Chats about code, provides in-line code completions, and generates new code.
- **Scans code for security vulnerabilities** and suggests remediations — and is secure and private by design.
- Makes code upgrades and improvements: language updates, debugging, and optimizations.
- Lets you describe an application in natural language and generates suggested code — a no-code path for simple web apps, business-process automation, and prototypes.

TABLE 11.4 Amazon Q Developer across the software development lifecycle (SDLC)

SDLC PHASE	HOW AMAZON Q DEVELOPER HELPS
Plan	Ask questions and get referenceable, contextual guidance; explain code with conversational coding
Create	Receive in-line code recommendations for multiple languages; implement features through comments or code prompts; chat in your IDE
Test and secure	Generate unit tests; scan project code for security vulnerabilities and get remediation suggestions
Operate	Troubleshoot errors; troubleshoot network connectivity with VPC Reachability Analyzer
Maintain and modernize	Modernize code with the Amazon Q Developer Agent for code transformation (e.g., upgrade to newer language versions)

WORKED EXAMPLE – AMAZON Q DEVELOPER WITH A TEMPLATE

To modify a CloudFormation stack template, a developer types the start of a desired resource inside a YAML file. Amazon Q Developer makes a suggestion, the developer accepts it, and the accepted code snippet is added into the template — with suggestions offered in **either YAML or JSON** to fit seamlessly. Amazon Q Developer can also help write **Lambda functions** for a CloudFormation stack: once activated in the Lambda console, it makes on-demand code recommendations in the Lambda code editor as you develop the function.

COMMON PITFALL

Amazon Q Developer accelerates template authoring — it does not replace understanding the template. Its outputs vary over time because the model continually updates, and it can generate insecure or incorrect code, so you still review, test, and (using its own scanning) check suggestions for security vulnerabilities before deploying.

11.6 Applying AWS Well-Architected Framework principles

The **AWS Well-Architected Framework** has **six pillars**, each with best practices and a set of questions to consider when you architect solutions. Automation touches most of them: in AWS you can view your **entire workload** — applications, infrastructure, policy, governance, and operations — as code, and apply the same engineering discipline you use for application code to every element of the stack. The best practices below are the ones this module maps to most directly.

TABLE 11.5 Automation best practices by Well-Architected pillar

PILLAR / APPROACH	BEST PRACTICES	WHY IT MATTERS
Operational Excellence — design for operations	Perform operations as code; make frequent, small, reversible changes; fully automate integration and deployment	Limits human error and gives consistent event responses; small reversible changes ease troubleshooting and rollback; automated CI/CD leaves a full audit trail for governance
Security	Automate security best practices	Software-based, version-controlled controls scale securely and cost-effectively; automating identification and classification reduces human error and exposure
Reliability — implement change	Deploy changes with automation; use automation when obtaining or scaling resources	Controlled, automated changes reduce the risk of production change; managed services (S3, CloudFront, Auto Scaling, Lambda, DynamoDB, Fargate, Route 53) automate scaling
Cost Optimization — optimize over time	Perform automation for operations	Automating repetitive, high-value, error-prone admin tasks lowers the cost of human effort and frees resources for higher-value work

The through line is that **using infrastructure as code — and CloudFormation in particular — is itself a Well-Architected practice.** Editing a template and rolling the change out consistently to every stack is how you deploy changes with automation, maintain consistency into production, and keep an auditable, reversible record of what changed. When you deploy CloudFormation templates, make sure security is considered at every step so that automation reinforces, rather than undermines, your security posture.

> **COMMON PITFALL**
>
> "Automate everything at once" is not the guidance. Cost Optimization says to **prioritize**: look at the total cost of human actions and start by automating the repetitive, high-value activities — especially those that carry a higher risk of human error, since that risk often becomes an unwanted operational cost.

Module summary

- Manual processes are error prone, unreliable, and inadequate for an agile business; they lack repeatability at scale, version control, audit trails, and consistency — automation reduces manual intervention, gives reproducible environments, improves productivity, and automates testing and scaling.
- Infrastructure as code (IaC) provisions and manages cloud resources with a human-readable, machine-consumable template that captures the desired state, enabling reusability, repeatability, and maintainability, and letting you replicate, redeploy, and repurpose infrastructure.

- AWS CloudFormation is the AWS IaC service: it models, creates, and manages a collection of resources called a stack at no extra charge, one stack spanning multiple services in a single Region, with create/update/delete actions and version-controlled rollback.
- Elastic Beanstalk, Quick Starts, AWS SAM, Amplify, and the AWS CDK are IaC services that all deploy through CloudFormation behind the scenes; choose among them by the balance of convenience and control and your team's skills.
- A template is written in JSON or YAML; the only required section is Resources, while Parameters, Mappings, Conditions, Transform, Outputs, and others are optional — Conditions decides whether resources are created (e.g., two AZs vs. one).
- Manage stacks through the template: preview edits with change sets, keep resources from deletion with a DeletionPolicy, and use drift detection (IN_SYNC vs. MODIFIED) to find and correct out-of-band changes by updating the template and running Update Stack.
- AWS Quick Starts pair a CloudFormation template with a deployment guide (options, security, cost) to deploy gold-standard, best-practice architectures in under an hour; Amazon Q Developer is a generative AI coding assistant that suggests and scans CloudFormation and Lambda code across the SDLC.
- Automation aligns with the Well-Architected Framework: perform operations as code; make frequent, small, reversible changes; fully automate integration and deployment; automate security best practices; deploy changes and scale with automation; and automate operations to optimize cost.

Review questions

1. **List the four risks that manual processes create, and the corresponding benefits automation provides.**

   Answer: Risks: no repeatability at scale, no version control, no audit trail, and inconsistent configurations. Benefits: reduces manual intervention/access, allows reproducible environments, improves productivity, and automates testing and scaling.

2. **Define infrastructure as code, and name its three key benefits.**

   Answer: IaC provisions and manages cloud resources by defining them in a template file that is both human-readable and machine-consumable, capturing the desired state without every step. Key benefits: reusability, repeatability, and maintainability.

3. **What is a CloudFormation stack, and does CloudFormation add a charge?**

   Answer: A stack is a collection of AWS resources you create, update, delete, and manage as a single unit. CloudFormation adds no extra charge — you pay only for the resources it creates.

4. **Which template section is required, and what do Conditions and Outputs do?**

   Answer: Resources is the only required section. Conditions control whether certain resources are created or properties assigned; Outputs describe the values returned when you view the stack's properties.

5. **How do change sets, drift detection, and DeletionPolicy each help manage a stack?**

   Answer: A change set previews the changes an update will make so you can approve them before running it; drift detection reports whether resources still match the template (IN_SYNC vs. MODIFIED); a DeletionPolicy preserves or backs up a resource when the stack is deleted or updated.

6. **Someone manually opened a port on a stack's security group. What is the better approach, and why?**

   Answer: Modify the security group settings in the CloudFormation template and run Update Stack. This keeps the model deployment synchronized with the actual deployment, whereas the manual console edit caused the drift.

7. **What are AWS Quick Starts, and what two things does each one include?**

   Answer: Quick Starts are automated reference architectures built on AWS best practices for security and high availability that can deploy an entire architecture in under an hour. Each includes a CloudFormation template and a deployment guide (deployment options, security considerations, and estimated costs).

8. **Name two ways Amazon Q Developer supports work with CloudFormation.**

   Answer: Any two of: suggesting resource code in YAML or JSON as you type in a template, generating and completing code from natural-language comments, scanning code for security vulnerabilities, and helping write Lambda functions for a CloudFormation stack in the Lambda console.