MODULE 13 · STUDY NOTES

Automating Deployment Using CI/CD Pipelines

DevOps culture and practices, the AWS code services that automate CI/CD, and provisioning applications with AWS CloudFormation and the AWS Serverless Application Model (AWS SAM)

Sofía's café website is in production, and she wants the release process to be less prone to human error. The answer her AWS colleague offers is **DevOps** — and this module unpacks what that means on AWS. In the cloud, an application is never just its code: it is the code *plus* all of the infrastructure it needs to run, so building and releasing software becomes a problem of automating both together. DevOps is the culture, practices, and tooling that make that automation possible; **CI/CD** (continuous integration, delivery, and deployment) is the practice at its center; and AWS supplies concrete services to implement it — the **code services** (CodeBuild, CodeDeploy, CodePipeline) that build and ship changes, **AWS CloudFormation** for provisioning infrastructure as code, and the **AWS Serverless Application Model (AWS SAM)** for packaging and deploying serverless applications. Together they turn a manual, error-prone release into a repeatable, automated pipeline.

LEARNING OBJECTIVES

- Describe DevOps
- Recognize AWS code services for continuous integration and continuous delivery (CI/CD)
- Describe how AWS CloudFormation is used to deploy applications
- Describe how the AWS Serverless Application Model (AWS SAM) is used to deploy serverless applications

13.1 Introducing DevOps

In the cloud, **your application is not just your application** — it is the application plus all of the associated infrastructure. That infrastructure can include virtual private clouds (VPCs), Elastic Load Balancing, Auto Scaling groups, Amazon RDS databases, Amazon S3 buckets and their data, Amazon ElastiCache servers, and any other AWS and non-AWS resources the application needs to run. Treating the two as a single unit is a change from how software used to be built.

For much of the past 20 years, software development (**Dev**) and system operations (**SysOps**) teams worked independently. Developers made sure applications worked in their dev environments and left it to system administrators to stand up the corresponding production environment. That disconnected hand-off produced expensive, time-consuming conversations as operations teams struggled to host — scalably and securely — whatever the developers had built. Cloud technology changed the dynamic: developers can now provision their own infrastructure, test code on it at scale, and turn it off when it is not in use, creating environments that closely mirror production. New methodologies followed — **Agile, DevOps, and DevSecOps**.

KEY TERMS

DevOps
The combination of Development and Operations — cultural philosophies, practices, and tools that support cloud application development and automate the development and release process.

DevSecOps
DevOps taken one step further by integrating security into the same combination of philosophies, practices, and tools.

DevOps is more a mentality than a specific toolset. Its cultural philosophy is captured in the motto *"People over process over tools."* The point is to remove the barrier between development and operations teams and get them communicating, working together under a model of **shared responsibility** to optimize both developer productivity and operational reliability.

DEVOPS PRACTICES

- **Microservice architecture** — instead of a monolithic application whose processes are tightly coupled and run as a single service, build discrete microservices that run independently in containers or as Lambda functions, so components can be added, removed, or updated without rewriting the whole code base.
- **Continuous integration and continuous delivery (CI/CD)** — regularly merge code to a central repository and automatically build, test, and prepare it for release.
- **Automate every phase** — automating each phase of the software development lifecycle reduces human effort and the errors that come from manual processes, delivering business value faster.
- **Continuous monitoring and improvement** — deployed code is watched and refined continuously.
- **Infrastructure as code** — use CloudFormation to create a repeatable method for deploying your cloud architecture.

The payoff is code that is secure, reliable, and maintainable. Under DevOps, development and operations are often merged into a single team of engineers with cross-functional skills, and quality-assurance and security teams integrate more tightly across the lifecycle. Automated testing catches bugs and audits changes; automated, production-mirroring dev environments make releases more reliable and end the *"it worked on my machine"* problem; and a microservice architecture keeps code maintainable and scalable. The result is faster delivery of value to customers and the ability to scale with the pace of change — a set of benefits usually summarized as improved collaboration, rapid delivery, and software that is scalable, secure, reliable, and maintainable.

TABLE 13.1 AWS DevOps service categories

CATEGORY	AWS SERVICES
CI/CD	AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy, AWS CodeConnections
Microservices	Amazon ECS, AWS Lambda, AWS Fargate
Platform as a service	AWS Elastic Beanstalk
Infrastructure as code	AWS CloudFormation, AWS OpsWorks, AWS Systems Manager
Monitoring and logging	Amazon CloudWatch, AWS CloudTrail, AWS X-Ray, AWS Config

This module focuses on the AWS **code services** that support CI/CD and on the **AWS CloudFormation** service.

COMMON PITFALL

Don't reduce DevOps to a set of tools. It is first a **cultural** shift — people over process over tools — that removes the wall between development and operations. And remember the cloud definition of an application: the code **plus all of its infrastructure**, deployed and managed together.

13.2 Using AWS code services for CI/CD

CI/CD spans the develop and deploy stages of the software development lifecycle — that is, code, build, and test on the develop side, and the release stages on the deploy side. The practice comes in three escalating forms.

Continuous integration (CI) is the practice of continuously checking code into a central code repository (for example, a Git repository) and verifying each change with an automated build and test process. Previously, developers worked in isolation for long stretches and merged into the mainline branch only after a feature was finished; batching changes that way made both the business logic and the test logic hard to merge. CI requires teams to write automated tests, which raises software quality and shortens the time to validate that a new version is good. Although definitions vary, **CI is commonly thought to stop at the build stage.**

Continuous delivery extends continuous integration by deploying out to production-like stages and running verification testing against those deployments. It may reach all the way to production, but **some form of manual intervention occurs between when code is checked in and when it is released to customers.** It is a major step beyond CI because it gives teams far greater certainty that the software will work in production. **Continuous deployment** goes the last step: it is the fully automated release of software to customers, extending from check-in through production **without human intervention**, which shortens the time to deliver value and to collect customer feedback.

TABLE 13.2 CI/CD across the lifecycle

PRACTICE	HOW FAR IT GOES	HUMAN STEP?
Continuous integration	Merge to a central repo, then automated build and test — stops at the build stage	Not applicable
Continuous delivery	Adds deployment to production-like stages with verification testing	Manual intervention before release
Continuous deployment	Automated release to customers, from check-in through production	None

AWS **code services** implement this pipeline as managed, pay-as-you-go building blocks; you combine them to build and automate a CI/CD pipeline.

TABLE 13.3 AWS code services

SERVICE	TYPE	WHAT IT DOES
AWS CodeBuild	Continuous integration	Fully managed service that compiles source code, runs tests, and produces packages ready to deploy; scales continuously and runs concurrent builds so none wait in a queue — no build servers to provision or manage
AWS CodeDeploy	Deployment	Fully managed service that automates software deployments to EC2, AWS Fargate, AWS Lambda, and on-premises servers; helps avoid downtime and eliminates error-prone manual operations
AWS CodePipeline	Continuous delivery	Fully managed service that automates the build, test, and deploy phases on every code change, per the release model you define; integrates with third-party sources such as GitHub and has no upfront fees

WORKED EXAMPLE – THE CAFÉ PIPELINE

In the module's lab, Sofía centralizes the café website code in a **code repository** with version control, then uses **CodePipeline** to automate updates: when new website code is pushed to the repository, the pipeline builds and deploys it to the S3 bucket that hosts the site, using a second S3 bucket to hold the pipeline's artifacts. A previously manual, error-prone update becomes an automated, repeatable release.

COMMON PITFALL

Keep continuous **delivery** and continuous **deployment** distinct — delivery keeps a manual approval gate before customers see the change; deployment removes it. And CI is not a release step: it is commonly thought to stop at the build stage.

13.3 Deploying applications with CloudFormation

Infrastructure as code (IaC) is a method to automate creating, updating, and deleting AWS infrastructure. Nearly everything you can build in the AWS Management Console can also be created programmatically through an SDK or the AWS CLI, so you can define your infrastructure as a series of scripts or other code artifacts and build it repeatably. From one code base you can stand up identical development, testing, staging, and production environments on demand — using the same code for production that you used everywhere else.

AWS CloudFormation is the fully managed service that delivers IaC on AWS. It provides a common language to describe and provision all of the infrastructure resources in your cloud environment, and it creates, updates, and deletes those resources in environments called **stacks**. CloudFormation is all about automated resource provisioning — it simplifies repeatedly and predictably creating groups of related resources — and it supports many application types, from existing enterprise and legacy applications to container-based solutions, including those built with AWS Elastic Beanstalk.

HOW CLOUDFORMATION WORKS

- You author a **template file** that lists the resources to provision.
- CloudFormation reads the template and constructs the listed resources.
- The output is your environment — a **stack** — which can hold a single resource or hundreds of them.
- You interact with CloudFormation through the console, the AWS CLI, or the AWS SDKs and APIs directly.
- Use the `DependsOn` attribute to set the order of creation — for example, a database server before a web server.

KEY TERMS

Stack
A unit of deployment: the collection of resources generated by a template. You create stacks, update them by running a modified template, and delete them.

Template
A JSON- or YAML-formatted text file that describes your AWS infrastructure and doubles as self-documenting for the environment.

A stack is a unit of deployment, and its resources are generated by a template. Deleting a stack **deletes all of the resources in it by default** (though that behavior can be reconfigured and overridden). The number of stacks you can deploy may be limited by **quotas** (formerly called limits); knowing them in advance avoids limitation errors that could force you to redesign templates or stacks, and if you need more you can request an increase through the **Support Center** in the console.

CloudFormation templates are text files formatted in **JSON or YAML**. They designate the resources to provision, act as documentation for the environment, and directly support the DevOps practice of infrastructure as code. CloudFormation follows the ECMA-404 JSON standard and the YAML 1.1 specification, with a few exceptions — it does not support binary, omap, pairs, set, and timestamp tags; aliases; or hash merges.

TABLE 13.4 The sections of a CloudFormation template

SECTION	PURPOSE
AWSTemplateFormatVersion	The template format version identifying its capabilities; `2010-09-09` is the latest and only valid value
Description	A text string that describes the template
Metadata	Objects that provide additional information about the template
Parameters	Values to pass to the template at runtime when you create or update a stack
Mappings	A lookup table of keys and associated values, useful for conditional parameter values
Conditions	Control whether resources are created or properties assigned during a stack create or update — for example, prod versus test
Transform	For serverless (Lambda-based) apps, specifies the AWS SAM version and enables AWS SAM syntax
Resources	The stack resources and their properties (such as an EC2 instance or S3 bucket) — the only required section
Outputs	The values returned whenever you view the stack's properties

SAMPLE EXAM QUESTION, WORKED

A developer has reached the account quota for the number of CloudFormation stacks in a Region. How could they increase the quota? The key phrases are **account quota** and **CloudFormation stacks**. Options such as using the AWS CLI or emailing an address do not raise a quota, and it is false that all service quotas are fixed. The correct answer is to use the **Support Center in the AWS Management Console**, where you can request an increase for service quotas that do not have a fixed maximum capacity.

COMMON PITFALL

On a CloudFormation template, **only the Resources section is required** — everything else (format version, description, parameters, outputs, and so on) is optional. And deleting a stack is destructive: by default it **removes every resource** the stack created.

13.4 Deploying serverless applications with AWS SAM

Serverless applications pose a deployment problem: you must supply everything needed to deploy a function — the code, its dependencies, and the blueprint that sets up the infrastructure — yet you cannot build that blueprint's environment locally or connect to a specific server to debug, and you need to deploy the stack into each AWS account. The **AWS Serverless Application Model (AWS SAM)** is the framework that solves this.

AWS SAM is an open-source framework for building serverless applications, and it is an extension of AWS CloudFormation — so it inherits CloudFormation's reliable deployment capabilities. You use SAM to build templates that define your serverless applications and then **deploy those templates with CloudFormation.** SAM has deep integration with CI/CD tools: you can pair it with **CodeBuild, CodeDeploy, and CodePipeline** to build a deployment pipeline for serverless apps. It also provides a **local invocation environment** similar to Lambda, which is useful for step-through debugging and catching issues early, before later in the process.

THE TWO COMPONENTS OF AWS SAM

- **Template specification** — a shorthand syntax that describes the functions, APIs, permissions, configurations, and events that make up a serverless application, treated as a single, deployable, versioned entity.
- **CLI** — a command-line tool that verifies SAM templates against the specification, invokes and step-through-debugs Lambda functions locally, and packages and deploys serverless applications to the AWS Cloud.

A SAM template follows the CloudFormation template format with a few differences: the transform declaration `Transform: AWS::Serverless-2016-10-31` is **required** and identifies the file as a SAM template; a **Globals** section (unique to SAM) defines properties common to all functions and APIs; the **Resources** section holds resources for both CloudFormation and SAM; and the **Parameters** section declares objects and presents additional prompts to the user.

TABLE 13.5 Reading a SAM template

DECLARATION	WHAT IT DOES
`Transform: AWS::Serverless-2016-10-31`	Tells CloudFormation this is a SAM template it needs to transform
`AWS::Serverless::Function`	Creates a Lambda function from the referenced .zip code, runtime, and handler, using the referenced IAM policy
`Events` with `Type: Api`	Creates an Amazon API Gateway endpoint; the SAM transform performs all necessary mappings and permissions
`AWS::Serverless::SimpleTable`	Creates an Amazon DynamoDB table

COMMON PITFALL

AWS SAM does not replace CloudFormation — it **extends** it. You still deploy through CloudFormation, and the required `Transform: AWS::Serverless-2016-10-31` line is what tells CloudFormation to expand the SAM shorthand into full resources.

Module summary

- In the cloud, an application is the code **plus all of its associated infrastructure**; DevOps — Development and Operations — is the culture, practices, and tools that build and release the two together, under the motto *people over process over tools*. DevSecOps adds security.
- DevOps practices include microservice architecture, CI/CD, automating every phase of the lifecycle, continuous monitoring and improvement, and infrastructure as code — yielding secure, reliable, maintainable, and scalable software delivered rapidly.
- CI/CD spans the develop and deploy stages: continuous integration merges to a central repo and runs automated build and test (stopping at the build stage); continuous delivery adds production-like testing with a manual gate before release; continuous deployment automates release to customers with no human intervention.
- AWS code services implement the pipeline: **CodeBuild** (build and test), **CodeDeploy** (deploy to EC2, Fargate, Lambda, on-premises), and **CodePipeline** (orchestrate build, test, and deploy on every change).
- Infrastructure as code automates creating, updating, and deleting AWS infrastructure so identical environments come from one code base.
- **AWS CloudFormation** is the fully managed IaC service: JSON- or YAML-formatted templates provision groups of related resources into **stacks** (a stack is a unit of deployment); deleting a stack deletes its resources by default, and stack quotas can be raised through the Support Center in the console.
- A CloudFormation template has nine possible sections — format version, description, metadata, parameters, mappings, conditions, transform, resources, outputs — of which **Resources is the only required** one; the `DependsOn` attribute sets creation order.
- **AWS SAM** is an open-source, CloudFormation-based framework for serverless apps: the required `Transform: AWS::Serverless-2016-10-31` declaration marks a template as SAM, deployment runs through CloudFormation, a local invocation environment aids debugging, and it integrates with CodeBuild, CodeDeploy, and CodePipeline.

Review questions

1. **Why is "your application" a bigger thing in the cloud, and what practice manages it?**

   Answer: In the cloud your application is the code plus all of the associated infrastructure it needs to run — VPCs, load balancers, Auto Scaling groups, databases, storage, and more. DevOps is the culture, practices, and tools that build, deploy, and manage the application and its infrastructure together.

2. **Explain the three forms of CI/CD and where each one stops.**

   Answer: Continuous integration merges every change into a central repository and runs an automated build and test, commonly stopping at the build stage. Continuous delivery extends that to production-like testing but keeps a manual intervention step before releasing to customers. Continuous deployment automates the release all the way to production customers with no human intervention.

3. **Match each AWS code service to its job: CodeBuild, CodeDeploy, CodePipeline.**

   Answer: CodeBuild compiles source code, runs tests, and produces deployable packages (continuous integration). CodeDeploy automates deployments to EC2, Fargate, Lambda, and on-premises servers. CodePipeline automates the build, test, and deploy phases into a release pipeline that runs on every code change (continuous delivery).

4. **What is a CloudFormation stack, and what happens to its resources when it is deleted?**

   Answer: A stack is a unit of deployment — the collection of related resources generated from a template. By default, deleting the stack deletes all of the resources in it, although that behavior can be reconfigured and overridden.

5. **In what two formats can a CloudFormation template be written, and which section is required?**

   Answer: A template is a text file formatted in JSON or YAML. The Resources section — which specifies the stack resources and their properties — is the only required section.

6. **How does AWS SAM relate to CloudFormation, and which one declaration is mandatory in a SAM template?**

   Answer: AWS SAM is an open-source framework and an extension of CloudFormation; you deploy SAM templates with CloudFormation and gain its reliable deployment capabilities. The Transform: AWS::Serverless-2016-10-31 declaration is required to identify the file as a SAM template.

7. **A developer has hit the account quota for CloudFormation stacks in a Region. How do they get more?**

   Answer: Request a quota increase through the Support Center in the AWS Management Console, which works for service quotas that do not have a fixed maximum capacity.