Week 05: Infrastructure as Code Part 2

Learning Layers of AWS IaC

- **Start with CloudFormation** to understand the basic unit of IaC—stacks. Learn how to create, update, troubleshoot, and manage stacks including using change sets, handling drift, and deploying nested stacks or stack sets.
- **Expand with the CloudFormation Registry & EventBridge.** Leverage extensions to bring custom or third-party functionalities into your stacks and use EventBridge to react to infrastructure events.
- **Transition to AWS CDK** for a code-first approach. Use constructs to build stacks and apps, taking advantage of higher-level abstractions (L2 and L3) that simplify development.

- **Apply Testing Practices** to your CDK apps using TDD, fine-grained assertions, and snapshot tests to ensure reliable, bug-free deployments.

AWS CloudFormation Stacks

Definition & Purpose:

- A **stack** is a collection of AWS resources (e.g., EC2, S3, Lambda) defined by a CloudFormation template (JSON or YAML) and managed as a single unit.
- **Key benefits:** repeatable deployments, environment duplication, centralized resource management.

Stack Lifecycle & Operations:

- **Creation:** Use the AWS Management Console or CLI to upload a template, name your stack, set parameters, and deploy.
- **Change Sets:** Previews of proposed changes before updating a stack. They let you verify modifications (like resource replacements) to avoid unintended impacts.
- **Drift Detection:** Identifies when stack resources have changed outside CloudFormation (drift). Remediation options include updating the template or importing existing changes.
- **Failure Handling & Rollback:** When a stack operation fails, you can retry, update, or roll back to the last known stable state. Options allow preserving successful resources for troubleshooting.

Advanced Concepts:

- **Nested Stacks:** Split large templates into smaller, reusable stacks that are referenced within a root stack. Ideal for reusing common components (e.g., load balancers).
- **Stack Sets:** Deploy and manage stacks across multiple AWS accounts and Regions with one operation.

- **Exporting/Importing Values:** Share resource outputs (like VPC IDs) between stacks to keep configurations consistent.

Expanding CloudFormation Usage: Registry & EventBridge

CloudFormation Registry:

- **Purpose:** A searchable collection of extensions that enhance CloudFormation's capabilities.
- **Extensions:**

Resource Types: Custom AWS resources that encapsulate provisioning logic.
Modules: Pre-packaged resource configurations for reuse across templates.
Hooks: Pre-provisioning checks (e.g., for compliance and security).
Extensions vs Hooks: Extension ***extends*** CloudFormation's capability. **Hook** is a specific type of extension and it acts like a ***pre-flight check*** for resource operations. Hooks are a subset of extensions, focused specifically on pre-operation validation.

- **Public vs. Private Extensions:**

Public: Managed by AWS or third parties, available to all users.
Private: Registered and activated within your account, either custom-built or shared.

Amazon EventBridge:

- **Core Concepts:**

Events: JSON objects representing changes (e.g., EC2 state changes).
Event Bus: The "pipeline" that receives events.
Rules: Define patterns to match events and route them to targets.
Targets: Resources (Lambda, SQS, etc.) that respond to events.

- **Integration with CloudFormation:**

CloudFormation emits events (e.g., stack status changes, drift detections) to EventBridge.
You can generate CloudFormation templates from existing EventBridge buses, rules, or pipes to automate event-driven workflows.

AWS Cloud Development Kit (CDK)

Overview

- **AWS CDK** is an open-source toolkit that uses familiar programming languages (TypeScript, Python, Java, C#, Go) to define cloud infrastructure.

- **Abstraction:** Translates imperative code (with logic, loops, conditionals) into declarative CloudFormation templates.

Core Constructs

- **Project:** Complete development environment
- **Apps:** An application is a collection of one or more stacks. Entry point of application.
- **Stacks:** Collections of constructs; each CDK stack corresponds to a CloudFormation stack.
- **Constructs:** Reusable, pre-written components that represent AWS resources. Building blocks. Constructs are the ***deployable units*** in CDK.
- **Resources:** Individual AWS services. Defined within constructs.

Levels of Constructs

- **L1 (CFN Resources):** Direct mappings to CloudFormation resources (e.g., `CfnBucket` for S3).
- **L2 (Curated Constructs):** Provide simplified, intent-based APIs with sensible defaults (e.g., `s3.Bucket`).
- **L3 (Patterns):** High-level constructs that bundle multiple resources to solve common use cases (e.g., ECS service with load balancing).

CDK Toolkit Commands

- **`cdk init`** – Start a new CDK project.
- **`cdk synth`** – Generate CloudFormation templates from your CDK app.
- **`cdk diff`** – Compare current changes with the deployed infrastructure.
- **`cdk deploy`** – Deploy stacks to AWS.

Key Concepts

- **Identifiers:** Ensure unique naming within a construct's scope.
- **Environments (env):** Target AWS account and Region for deployment.
- **Contexts & Assets:** Pass configuration values and bundle local files or Docker images with your app.

Testing AWS CDK Constructs

Test-Driven Development (TDD)

- **Process:**

1. Write a failing test (e.g., check for the existence of an S3 bucket).
2. Write minimal code to pass the test.

3. Refactor and repeat.

- **Benefits:** Rapid feedback, modular code, and increased confidence in infrastructure changes.

Types of Tests

- **Fine-Grained Assertions:** Validate specific properties of the synthesized CloudFormation template (e.g., ensuring versioning is enabled on an S3 bucket).
- **Snapshot Tests:** Compare the entire generated template against a stored baseline to detect unintended changes.

Testing Tools & Frameworks

- **AWS CDK Assertions Module:** Provides functions to write assertions against your CloudFormation templates.
- **Frameworks:** Use Jest for TypeScript/JavaScript or Pytest for Python to automate tests.
- **Best Practices:** Keep tests modular, use helper functions to reduce duplication, and clearly name tests to indicate their purpose.