Week 04: Infrastructure as Code Part 1

Introduction to IaC

- **Definition**: Infrastructure as Code (IaC) is the practice of **automating infrastructure provisioning and management using code** instead of manual processes.
- **Benefits**:

Automates provisioning and deployment.
Ensures **consistency** across environments.
Reduces **human errors** and configuration drift.
Enables **version control**, rollback, and collaboration.
Improves **scalability**, disaster recovery, and compliance.

Approach	Description	Examples
Imperative	**Specifies the exact steps** to create infrastructure. More control but complex.	Bash scripts, AWS CLI, Ansible playbooks (ad hoc tasks)
Declarative	**Describes the desired state** of infrastructure, and the system determines how to reach it. Easier to maintain.	AWS CloudFormation, Terraform, Kubernetes manifests

IaC in DevOps

- **Reproducibility**: Enables **consistent** infrastructure deployment across development, staging, and production.
- **CI/CD Integration**: **Infrastructure changes** can be version-controlled, tested, and deployed as part of a **CI/CD pipeline**.
- **Version Control**:

Without IaC: Infrastructure changes are made manually via AWS Console or CLI.
With IaC: Infrastructure **code is stored in repositories** (e.g., AWS CodeCommit, Git) and changes can be tracked, reviewed, and reverted.

- **Approval Gates**:

Continuous Deployment: Fully automated, no manual intervention.
Continuous Delivery: Requires **manual approval** before deployment (more control for regulated environments).

AWS and Third-Party IaC Tools

Tool	Description
AWS CloudFormation	**AWS-native declarative IaC service** that uses JSON/YAML templates to provision AWS resources.

Tool	Description
AWS CDK (Cloud Development Kit)	Allows **programming infrastructure** using languages like Python, TypeScript, and Java instead of JSON/YAML.
AWS CDK for Kubernetes (CDK8s)	Defines **Kubernetes applications using code** instead of writing YAML manifests manually.
AWS CDK for Terraform (CDKTF)	Uses programming languages **instead of HashiCorp Configuration Language (HCL)** to write Terraform configurations.
Terraform	**Popular third-party IaC tool** for multi-cloud provisioning using HCL.
AWS Cloud Control API	Standardized API for **managing AWS and third-party resources**.
AWS Serverless Application Model (AWS SAM)	Framework for defining and deploying **serverless applications** using CloudFormation.

AWS CloudFormation Basics

- **Definition**: AWS CloudFormation automates the creation of AWS resources using JSON/YAML **templates**.
- **Workflow**:

Template Creation – Define resources in a CloudFormation template.
Template Validation – Ensure the template syntax is correct.
Resource Provisioning – CloudFormation **orchestrates** the creation of AWS resources.
Stack Management – The resources are grouped into a **stack**, which can be updated or deleted as a unit.

AWS CloudFormation Template Structure

Section	Purpose
Format Version (Optional)	Specifies the CloudFormation **template version**.
Description (Optional)	Provides a **summary** of the template.
Metadata (Optional)	Stores **additional information** about the template.
Parameters (Optional)	Allows **user input** when creating/updating stacks.
Rules (Optional)	Validates input **parameter values** before stack creation.
Mappings (Optional)	Acts as a **lookup table** for static values (e.g., region-based AMI IDs).
Conditions (Optional)	Defines **conditions for resource creation** (e.g., only create an instance in production).
Transform (Optional)	Enables **AWS SAM** for serverless applications.
Resources (Required)	Declares **AWS resources** (e.g., EC2, S3, RDS).

Section	Purpose
Outputs (Optional)	Returns stack values **after deployment** (e.g., an S3 bucket URL).

CloudFormation Key Concepts

Resources

- **Required** section in a CloudFormation template.
- Each resource has:
  - **Logical ID** (unique within the template).
  - **Resource Type** (e.g., `AWS::EC2::Instance`).
  - **Properties** (e.g., `InstanceType: t2.micro`).

Feature	Purpose
Rules	Validate **input parameters** before deploying resources.
Conditions	Dynamically create resources **based on conditions** (e.g., only create a DB in production).

Intrinsic Functions in CloudFormation

Function	Purpose
`Ref`	Returns **parameter/resource value**.
`Fn::GetAtt`	Retrieves **attribute values** from a resource.
`Fn::Join`	Concatenates values into a **single string**.
`Fn::Select`	Picks an item from a **list by index**.
`Fn::Split`	Splits a **string into a list**.
`Fn::Sub`	Performs **string substitution**.

Function	Purpose
`Fn::And`	Evaluates to `true` if **all conditions are true**.
`Fn::Or`	Evaluates to `true` if **any condition is true**.
`Fn::Not`	**Negates** a condition.
`Fn::If`	Includes/excludes **resources based on conditions**.
`Fn::Equals`	Compares **two values** for equality.

CloudFormation Advanced Features

Macros

- **Custom processing tools** that modify templates before deployment.
- Powered by **AWS Lambda**.
- Used for **reusable template logic** (e.g., auto-generating common configurations).

Modules

- **Reusable CloudFormation templates** that define best practices.
- Used like standard CloudFormation resources.
- Available in the **CloudFormation Registry**.

Feature	Macros	Modules
Purpose	**Transform templates** dynamically.	**Encapsulate best practices** and reusable patterns.
Implementation	Uses **AWS Lambda**.	Uses **nested stacks**.
Execution Time	Before deployment.	During stack creation.
Use Cases	Validations, string manipulations, custom logic.	Common infrastructure patterns (e.g., standard EC2 setup).

CloudFormation Designer

- **Graphical tool** for creating and modifying CloudFormation templates.
- Features:
  - **Drag and drop** resources.
  - **Real-time template validation**.
  - **Visual representation** of resource relationships.
  - Integrated **JSON/YAML editor**.