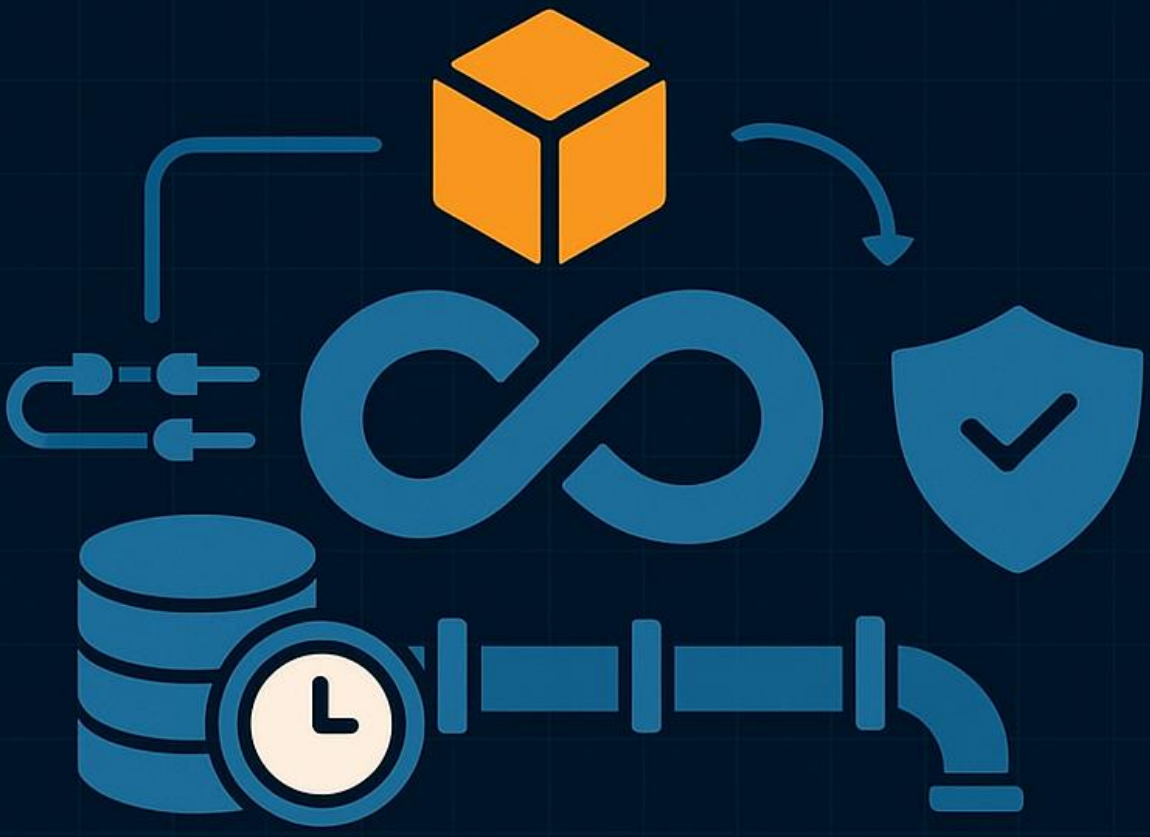
THE ULTIMATE AWS DEVOPS PLAYBOOK



Pipelines, Monitoring, Security & Infrastructure

MANISH KUMAR

The Ultimate AWS DevOps Playbook: Pipelines, Monitoring, Security & Infrastructure

By Manish Kumar

Table of Contents

📘 Chapter 1: Introduction to DevOps

1.1 What is DevOps?

DevOps is a set of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity. It unifies **development (Dev)** and **operations (Ops)** teams for better collaboration, automation, and continuous delivery.

✅ Key Characteristics of DevOps:

- **Collaboration:** Developers and operations work together across the lifecycle.
- **Automation:** Infrastructure provisioning, code deployment, and testing are automated.
- **Continuous Improvement:** Through metrics, feedback, and monitoring.

1.2 DevOps Lifecycle

Stage	Purpose	Example Tools
Plan	Define project roadmap	Jira, GitHub Projects
Develop	Write and version code	Git, GitHub, CodeCommit
Build	Compile and build applications	CodeBuild, Jenkins
Test	Validate quality and functionality	Selenium, CodeBuild
Release	Prepare for deployment	CodePipeline, GitHub Actions
Deploy	Deploy to production	CodeDeploy, ECS, Lambda
Operate	Manage infrastructure and runtime	CloudWatch, Systems Manager
Monitor	Track logs, metrics, errors	CloudWatch, X-Ray

1.3 Key DevOps Concepts

🔹 CI/CD (Continuous Integration and Continuous Deployment)

- **CI:** Automatically build and test code every time a change is committed.
- **CD:** Automatically deploy applications to test or production environments.

✅ CI/CD Example:

```
# CLI-based example: Commit and push code to trigger CI/CD
git add .
```

```
git commit -m "Added login functionality"
git push origin main
```

Result: This push can trigger AWS CodePipeline → CodeBuild → CodeDeploy.

🔹 Infrastructure as Code (IaC)

IaC means managing and provisioning infrastructure using code instead of manual processes.

✅ IaC Example: CloudFormation (YAML)

```
Resources:
  MyBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: my-devops-bucket
```

✅ IaC Example: Terraform

```
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-devops-bucket"
}
```

✅ GUI Equivalent:

1. Open AWS Console → S3 → Create Bucket → Fill in details manually.

🔹 Automation

Automating repetitive tasks like:

- Code testing
- Infrastructure provisioning
- Deployments

Example:

```
aws cloudformation deploy \
  --template-file template.yaml \
  --stack-name my-stack \
  --capabilities CAPABILITY_IAM
```

🔹 Monitoring and Feedback

Use tools like Amazon CloudWatch to collect logs, metrics, and set up alarms.

✅ **CLI Example:**

```
aws cloudwatch put-metric-alarm \
  --alarm-name CPU_Utilization_Alarm \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-0123456789abcdef0 \
  --evaluation-periods 2 \
  --alarm-actions arn:aws:sns:us-east-1:123456789012:MyTopic \
  --unit Percent
```

1.4 Benefits of DevOps on AWS

Benefit	How AWS Helps
Scalability	Auto Scaling, Elastic Load Balancing
Automation	CodePipeline, CodeBuild, CodeDeploy
Infrastructure as Code	CloudFormation, CDK, Terraform
Monitoring & Logging	CloudWatch, X-Ray, CloudTrail
Security & Compliance	IAM, AWS Config, Secrets Manager
High Availability	Multi-AZ deployments, Elastic IPs
Developer Agility	Rapid provisioning of environments

1.5 AWS DevOps Tools (Preview)

Service	Functionality
CodeCommit	Git-based source control
CodeBuild	Build and test automation
CodeDeploy	Deploy to EC2, Lambda, ECS
CodePipeline	CI/CD orchestration
CodeStar	DevOps project templates
CloudFormation	IaC templates
CloudWatch	Logs, metrics, dashboards

CodeArtifact	Private package repositories

Each of these will be explored in detail in the next chapters.

1.6 DevOps on AWS vs Traditional Setup

Feature	Traditional Setup	AWS DevOps Approach
Infrastructure	Manually provisioned	IaC via CloudFormation
Builds	On local servers	CodeBuild (scalable builds)
Deployments	Manual FTP or scripts	Automated via CodeDeploy
Monitoring	Basic logging	CloudWatch, X-Ray
Scalability	Limited	Elastic scaling

1.7 Sample End-to-End Workflow (Diagram)

```
Developer → CodeCommit → CodePipeline
          → CodeBuild → CodeDeploy → EC2/ECS/Lambda
                                          ↓
                                Monitor via CloudWatch
```

1.8 Summary

- DevOps is the cultural and toolset shift to streamline software delivery.
- AWS provides native services for each DevOps phase.
- Automation, monitoring, and IaC are core to AWS DevOps.
- CLI, SDK, and console all offer interfaces to manage DevOps workflows.

📘 Chapter 2: AWS DevOps Services Landscape

2.1 Overview

AWS offers a fully managed set of services that allow you to implement complete DevOps pipelines, from code commit to production deployment, with integrated monitoring and security.

These services are categorized into:

- Source Control
- Build & Test Automation
- Deployment
- CI/CD Orchestration
- Infrastructure as Code (IaC)
- Monitoring & Logging
- Package Management

2.2 Source Control – AWS CodeCommit

✅ What is CodeCommit?

AWS CodeCommit is a fully managed Git-based source control system, similar to GitHub and GitLab, but hosted within AWS.

🔹 Features

- Private and secure Git repositories
- Integration with IAM for fine-grained permissions
- Supports Git clients and HTTPS/SSH access
- Triggers for Lambda, SNS, or CodePipeline

🛠️ GUI Example

1. Go to AWS Console → CodeCommit → Create Repository
2. Name: `my-web-app-repo`
3. Click **Create**
4. Follow the instructions to clone it using Git.

💻 CLI Example

```
aws codecommit create-repository \
  --repository-name my-web-app-repo \
  --repository-description "DevOps demo project"
```

🧩 Hands-on Case

Create a repository, push code using Git, and configure a trigger to start a build with CodeBuild.

2.3 Build Automation – AWS CodeBuild

✅ What is CodeBuild?

A fully managed continuous integration service that compiles source code, runs tests, and produces deployable packages.

🔹 Features

- Scales automatically
- Supports Docker, Java, Python, Node.js, Go, etc.
- Supports custom build environments
- Integrated with CodePipeline and CloudWatch

🛠️ GUI Example

1. AWS Console → CodeBuild → Create Build Project
2. Connect to CodeCommit repo
3. Define `buildspec.yml` or provide inline build commands
4. Choose environment (e.g., `aws/codebuild/standard:5.0`)
5. Enable CloudWatch logs

💻 CLI Example

```
aws codebuild start-build --project-name my-web-app-build
```

🧩 Hands-on Case

Use a `buildspec.yml` to package a Node.js app and upload artifacts to an S3 bucket.

2.4 CI/CD Orchestration – AWS CodePipeline

✅ What is CodePipeline?

A fully managed continuous integration and delivery (CI/CD) service that automates your software release processes.

🔹 Features

- Connect CodeCommit, GitHub, S3, CodeBuild, CodeDeploy
- Supports manual approval stages
- JSON/YAML templates supported
- Monitors status and failure points

🛠️ GUI Example

1. AWS Console → CodePipeline → Create Pipeline
2. Source: CodeCommit
3. Build: CodeBuild
4. Deploy: CodeDeploy or Lambda
5. Add manual approval step if needed

💻 CLI Example

```
aws codepipeline create-pipeline \
  --cli-input-json file://pipeline-definition.json
```

🧩 Hands-on Case

Automate the release pipeline: CodeCommit → CodeBuild → CodeDeploy → EC2

2.5 Deployment – AWS CodeDeploy

✅ What is CodeDeploy?

CodeDeploy automates the deployment of applications to Amazon EC2, ECS, Lambda, or on-prem servers.

🔹 Features

- Supports in-place and blue/green deployments
- Lifecycle hooks
- Rollbacks on failure
- Can deploy artifacts from S3 or CodePipeline

🛠️ GUI Example

1. AWS Console → CodeDeploy → Create Application
2. Choose compute platform (EC2 or Lambda)
3. Create Deployment Group
4. Attach IAM role and tags

💻 CLI Example

```
aws deploy create-application --application-name MyApp --compute-platform
Lambda
```

🧩 Hands-on Case

Deploy a Lambda function update using blue/green deployment strategy with traffic shifting.

2.6 Infrastructure as Code – AWS CloudFormation & CDK

✅ What is CloudFormation?

CloudFormation enables you to define and provision AWS infrastructure using YAML or JSON templates.

✅ What is CDK?

The AWS Cloud Development Kit (CDK) allows you to define infrastructure using programming languages like Python, TypeScript, Java.

🛠️ CloudFormation YAML Example

```
Resources:
  MyBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: devops-bucket
aws cloudformation deploy \
  --template-file template.yaml \
  --stack-name my-devops-stack \
  --capabilities CAPABILITY_NAMED_IAM
```

🛠️ CDK TypeScript Example

```
new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-cdk-bucket',
  versioned: true,
```

```
});
cdk synth
cdk deploy
```

🧩 Hands-on Case

Deploy a VPC, EC2 instance, and RDS via CloudFormation; repeat using CDK.

2.7 Monitoring – Amazon CloudWatch

✅ What is CloudWatch?

Amazon CloudWatch monitors your AWS resources and applications in real time.

🔹 Features

- Metrics, logs, dashboards
- Alarms and anomaly detection
- Custom metrics and logs from apps
- Integrated with SNS for alerts

🛠️ GUI Example

1. AWS Console → CloudWatch → Alarms → Create Alarm
2. Choose metric (e.g., EC2 CPUUtilization)
3. Set threshold and action (e.g., send SNS notification)

💻 CLI Example

```
aws logs create-log-group --log-group-name /app/logs

aws cloudwatch put-metric-alarm \
  --alarm-name HighCPU \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --threshold 70 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-periods 2 \
  --period 300 \
  --alarm-actions arn:aws:sns:... \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0
```

🧩 Hands-on Case

Create a CloudWatch dashboard with EC2, Lambda, and RDS metrics. Configure alarm + SNS.

2.8 Package Management – AWS CodeArtifact

✅ What is CodeArtifact?

AWS CodeArtifact is a fully managed artifact repository for secure storage and sharing of software packages.

🔹 Features

- Supports npm, Maven, pip, NuGet
- Integrates with CodeBuild and IDEs
- Usage-based pricing

🛠️ CLI Example

```
aws codeartifact create-domain --domain devops-domain
aws codeartifact create-repository --domain devops-domain --repository my-repo
```

🧩 Hands-on Case

Publish and consume a Python package via CodeArtifact using pip.

2.9 AWS CodeStar (Optional)

✅ What is CodeStar?

An all-in-one service to manage DevOps projects via templates and dashboards. It helps rapidly set up CI/CD pipelines, Git repositories, and Cloud9 IDEs.

Not used heavily in large-scale enterprise DevOps, but helpful for beginners.

2.10 Summary

Service	Category	CLI Support	GUI Support	Hands-on Use Case
CodeCommit	Source Control	✅	✅	Git repo management

CodeBuild	Build/Test Automation	✅	✅	Build and test app
CodePipeline	CI/CD Orchestration	✅	✅	End-to-end pipeline
CodeDeploy	Deployment	✅	✅	EC2/Lambda deploys
CloudFormation	Infrastructure as Code	✅	✅	Deploy environments
CDK	Infra-as-code SDK	✅	❌ (uses CLI)	Programmatic IaC
CloudWatch	Monitoring	✅	✅	Logs, alarms, dashboards
CodeArtifact	Package Management	✅	✅	Private pip/npm registry

📘 Chapter 3: Setting Up the DevOps Environment

3.1 Overview

Before launching any DevOps pipelines, you must prepare a secure, scalable, and networked AWS environment. This setup includes:

- IAM roles and policies for DevOps services
- VPC, subnets, security groups
- S3 buckets for artifacts
- Key management (KMS)
- Audit and security baseline (CloudTrail, AWS Config)

3.2 IAM Configuration for DevOps

IAM (Identity and Access Management) is the backbone of secure access control in AWS. Each DevOps tool (CodeBuild, CodePipeline, etc.) needs a dedicated role with proper policies.

✅ Best Practices

- Use least privilege principle
- Use managed policies when possible
- Enable MFA for human users
- Rotate access keys regularly

🔹 IAM Role for CodeBuild

🛠️ GUI

1. AWS Console → IAM → Roles → Create Role
2. Use Case: **CodeBuild**
3. Attach managed policy: `AWSCodeBuildDeveloperAccess`
4. Add inline policy if S3/VPC access is needed

💻 CLI

```
aws iam create-role \
  --role-name CodeBuildServiceRole \
  --assume-role-policy-document file://trust-policy.json
```

trust-policy.json

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Service": "codebuild.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }]
}
```

🔹 IAM Role for CodePipeline

Attach:

- `AWSCodePipelineFullAccess`
- `AmazonS3FullAccess` (for artifacts)
- `AWSCodeBuildDeveloperAccess`
- Inline permission for CodeDeploy or Lambda, as needed

3.3 VPC & Networking Setup

To securely host your EC2, Lambda (VPC mode), RDS, and ECS resources, a VPC must be defined.

✅ Architecture:

```
VPC (10.0.0.0/16)
├── Public Subnet (10.0.1.0/24)
│    └── NAT Gateway / Bastion Host
├── Private Subnet (10.0.2.0/24)
     └── EC2, RDS, Lambda
```

🔹 Terraform Example

```
resource "aws_vpc" "devops_vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "public_subnet" {
  vpc_id     = aws_vpc.devops_vpc.id
  cidr_block = "10.0.1.0/24"
  map_public_ip_on_launch = true
}
```

```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.devops_vpc.id
}
```

🔹 CloudFormation Example

```
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16

  PublicSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock: 10.0.1.0/24
      VpcId: !Ref VPC
```

3.4 S3 Buckets for Artifacts and Logs

DevOps pipelines use Amazon S3 to store build artifacts, templates, and logs.

✅ Use Cases

- Store CodePipeline artifacts
- Upload packaged Lambda/zipped app
- Archive logs

💻 CLI

```
aws s3api create-bucket --bucket devops-artifacts-bucket --region us-east-1
```

🛠️ CloudFormation

```
Resources:
  ArtifactBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: devops-artifacts-bucket
```

3.5 KMS Key for Secure Encryption

Use AWS KMS to encrypt S3 buckets, CodeBuild secrets, and CodePipeline artifacts.

💻 **CLI**

```
aws kms create-key --description "Key for DevOps artifacts"
```

3.6 CloudTrail for Auditing

CloudTrail records all API calls in your AWS account—critical for tracking changes and ensuring compliance.

🛠️ **GUI**

1. AWS Console → CloudTrail → Create Trail
2. Store logs in a new/existing S3 bucket
3. Enable management and data events

💻 **CLI**

```
aws cloudtrail create-trail \
  --name DevOpsTrail \
  --s3-bucket-name devops-audit-logs
aws cloudtrail start-logging --name DevOpsTrail
```

3.7 AWS Config for Baseline Compliance

AWS Config tracks configuration changes to resources and alerts when violations occur.

🛠️ **GUI**

1. AWS Console → Config → Set Up
2. Choose all resources
3. Store data in a dedicated S3 bucket
4. Create rules like "restricted SSH" or "S3 must be encrypted"

💻 **CLI**

```
aws configservice put-configuration-recorder \
  --configuration-recorder name=devops-
recorder,roleARN=arn:aws:iam::123456789012:role/aws-config-role

aws configservice start-configuration-recorder \
  --configuration-recorder-name devops-recorder
```

3.8 Optional: Secrets Manager

Use AWS Secrets Manager to store credentials for GitHub tokens, database passwords, etc.

💻 CLI

```
aws secretsmanager create-secret \
  --name DevOpsGithubToken \
  --secret-string '{"token":"ghp_ABC123..."}'
```

3.9 Recommended Security Checklist

Resource	Security Practice
IAM	Least privilege, MFA, no root use
VPC	Use private subnets for sensitive resources
S3	Block public access, enable encryption
CloudTrail	Always enabled
KMS	Encrypt S3, EBS, RDS, etc.
Secrets	Use Secrets Manager, not plaintext
CI/CD Access	Service roles with explicit permissions

3.10 Summary

Task	CLI Support	IaC Support	GUI Available
IAM Role Creation	✅	✅	✅
VPC + Subnets	✅	✅	✅
S3 Bucket Creation	✅	✅	✅
KMS Key	✅	❌ (limited)	✅
CloudTrail Setup	✅	✅ (manual)	✅
AWS Config Rules	✅	❌	✅
Secrets Manager	✅	❌	✅

📘 Chapter 4: CI/CD with CodePipeline, CodeBuild, and CodeDeploy

4.1 Overview

CI/CD (Continuous Integration and Continuous Deployment) automates the process of building, testing, and releasing code. On AWS, the native stack for CI/CD includes:

Tool	Purpose
CodeCommit	Source Control (Git)
CodeBuild	Build and Test automation
CodePipeline	CI/CD Orchestration
CodeDeploy	Automated deployment engine

This chapter builds a real-world pipeline to:

- Automatically trigger on a code commit
- Build and package a Node.js or Python application
- Deploy it to EC2 or Lambda
- Monitor the pipeline with CloudWatch

4.2 Architecture of the CI/CD Pipeline

```
Developer → CodeCommit → CodePipeline
                         ├── CodeBuild
                         └── CodeDeploy → EC2 / Lambda
```

4.3 Prerequisites Checklist

Resource	Setup
CodeCommit Repo	Yes
EC2 Instances / Lambda	Yes (based on target)
S3 Bucket	For artifacts

IAM Roles	CodeBuild, CodePipeline, CodeDeploy roles created
Application Code	With buildspec.yml and appspec.yml

4.4 Step 1 – Set Up CodeCommit Repository

🛠️ GUI Steps

1. Go to AWS Console → CodeCommit → Create Repository
2. Name: `sample-webapp`
3. Clone the repo locally

💻 CLI

```
aws codecommit create-repository \
  --repository-name sample-webapp \
  --repository-description "CI/CD demo app"
```

📂 Sample Repo Structure

```
sample-webapp/
├── index.html
├── scripts/
│   └── start.sh
├── buildspec.yml
├── appspec.yml
```

4.5 Step 2 – Create buildspec.yml (for CodeBuild)

This file defines how CodeBuild should build your app.

```
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 18
    commands:
      - echo Installing packages...
      - npm install
  build:
    commands:
      - echo Building the app...
      - npm run build
artifacts:
```

```
  files:
    - '**/*'
```

4.6 Step 3 – Create appspec.yml (for CodeDeploy)

Defines how CodeDeploy will deploy and start the app.

```
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/html
hooks:
  AfterInstall:
    - location: scripts/start.sh
      timeout: 300
      runas: root
```

4.7 Step 4 – Configure CodeBuild

🛠️ GUI Steps

1. AWS Console → CodeBuild → Create Project
2. Source: CodeCommit repo `sample-webapp`
3. Environment:
    - OS: Amazon Linux 2
    - Runtime: Node.js, Python, etc.
4. Use buildspec.yml from source
5. IAM Role: `CodeBuildServiceRole`

💻 CLI

```
aws codebuild create-project \
  --name sample-webapp-build \
  --source type=CODECOMMIT,location=https://git-codecommit.us-east-
1.amazonaws.com/v1/repos/sample-webapp \
  --environment
type=LINUX_CONTAINER,computeType=BUILD_GENERAL1_SMALL,image=aws/codebuild/s
tandard:5.0 \
  --service-role arn:aws:iam::<account-id>:role/CodeBuildServiceRole
```

4.8 Step 5 – Configure CodeDeploy

🛠️ GUI Steps

1. AWS Console → CodeDeploy → Applications → Create Application
2. Platform: EC2 (or Lambda)
3. Create Deployment Group
    - Select EC2 tags or Auto Scaling group
    - Attach IAM role: `CodeDeployServiceRole`

💻 CLI

```
aws deploy create-application \
  --application-name sample-webapp \
  --compute-platform Server

aws deploy create-deployment-group \
  --application-name sample-webapp \
  --deployment-group-name WebAppGroup \
  --service-role-arn arn:aws:iam::<account-id>:role/CodeDeployServiceRole \
  --ec2-tag-filters Key=Environment,Value=Dev,Type=KEY_AND_VALUE \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --auto-scaling-groups "" \
  --deployment-style
deploymentType=IN_PLACE,deploymentOption=WITHOUT_TRAFFIC_CONTROL
```

4.9 Step 6 – Create CodePipeline

🛠️ GUI Steps

1. AWS Console → CodePipeline → Create Pipeline
2. Source Stage: CodeCommit → `sample-webapp`
3. Build Stage: CodeBuild → `sample-webapp-build`
4. Deploy Stage: CodeDeploy → `sample-webapp` → `WebAppGroup`
5. Artifact bucket: select or create

💻 CLI

Create a pipeline JSON like `pipeline-definition.json`:

```
{
  "pipeline": {
    "name": "WebAppPipeline",
    "roleArn": "arn:aws:iam::<account-id>:role/CodePipelineServiceRole",
    "artifactStore": {
      "type": "S3",
      "location": "devops-artifacts-bucket"
    },
```

```
"stages": [
  {
    "name": "Source",
    "actions": [{
      "name": "SourceAction",
      "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
      },
      "outputArtifacts": [{ "name": "SourceOutput" }],
      "configuration": {
        "RepositoryName": "sample-webapp",
        "BranchName": "main"
      },
      "runOrder": 1
    }]
  },
  {
    "name": "Build",
    "actions": [{
      "name": "BuildAction",
      "actionTypeId": {
        "category": "Build",
        "owner": "AWS",
        "provider": "CodeBuild",
        "version": "1"
      },
      "inputArtifacts": [{ "name": "SourceOutput" }],
      "outputArtifacts": [{ "name": "BuildOutput" }],
      "configuration": {
        "ProjectName": "sample-webapp-build"
      },
      "runOrder": 1
    }]
  },
  {
    "name": "Deploy",
    "actions": [{
      "name": "DeployAction",
      "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "provider": "CodeDeploy",
        "version": "1"
      },
      "inputArtifacts": [{ "name": "BuildOutput" }],
      "configuration": {
        "ApplicationName": "sample-webapp",
        "DeploymentGroupName": "WebAppGroup"
      },
      "runOrder": 1
```

```
        }]
      }
    ],
    "version": 1
  }
}
```

Then run:

```
aws codepipeline create-pipeline --cli-input-json file://pipeline-definition.json
```

4.10 Notifications and Monitoring

- **CloudWatch Logs** for CodeBuild and CodeDeploy
- **SNS** integration for deployment failure alerts
- **Pipeline History** visible in CodePipeline console

4.11 Rollbacks and Approvals

- Use **CodeDeploy**'s `failure threshold` and `rollback configuration`
- Insert **Manual Approval** step in CodePipeline:
    - Add stage → Approval → Use SNS topic

4.12 Full Workflow Summary

Stage	Tool	Trigger	Output
Source	CodeCommit	Git Push (webhook)	Source artifact
Build	CodeBuild	On source change	Built/packaged app artifact
Deploy	CodeDeploy	On successful build	App deployed to EC2/Lambda

📘 Chapter 5: Advanced Deployment Patterns on AWS

5.1 Overview

Modern DevOps demands advanced, low-risk deployment strategies that reduce downtime and accelerate feedback. In this chapter, you'll learn how to implement:

Pattern	Use Case
Blue/Green	Deploy with zero-downtime and quick rollback
Canary	Gradual rollout to a subset of users
Multi-environment	Dev → Staging → Production pipelines
External Git Source	Trigger pipeline from GitHub/GitLab

5.2 Blue/Green Deployment (CodeDeploy + EC2/ECS)

✅ Benefits:

- Instant rollback
- No downtime
- A/B testing possible

🛠️ GUI (ECS Example):

1. AWS Console → CodeDeploy → Create Application (ECS)
2. Deployment type: **Blue/Green**
3. Choose your ECS service + Load Balancer
4. Specify test traffic, production traffic shift

💻 CLI (EC2 Example):

```
aws deploy create-deployment-group \
  --application-name sample-webapp \
  --deployment-group-name WebAppGroup \
  --deployment-style
deploymentType=BLUE_GREEN,deploymentOption=WITH_TRAFFIC_CONTROL \
```

```
  --blue-green-deployment-configuration
"terminateBlueInstancesOnDeploymentSuccess={action=TERMINATE,terminationWai
tTimeInMinutes=5}" \
  --load-balancer-info "targetGroupInfoList=[{name=BlueGreenTargetGroup}]"
\
  --service-role-arn arn:aws:iam::<account-id>:role/CodeDeployRole
```

💡 Pro Tips:

- Use ELB to split traffic between versions
- CodeDeploy handles traffic shifting and rollback

5.3 Canary Deployment (Lambda or ECS)

Canary deployments roll out new versions gradually.

🌊 Canary with Lambda:

💻 CLI

```
aws lambda update-alias \
  --function-name myLambdaFunction \
  --name live \
  --routing-config '{"AdditionalVersionWeights": {"2": 0.1}}'
```

This sends 10% traffic to version 2 and 90% to version 1.

🛠️ GUI:

1. Lambda Console → Aliases → Create alias `live`
2. Add routing configuration

🌊 Canary with ECS/CodeDeploy:

1. Set deployment config to **Canary10Percent5Minutes**
2. Initial 10% traffic goes to new version → wait 5 mins → full traffic

💻 CLI:

```
aws deploy create-deployment-group \
  --application-name sampleECSApp \
  --deployment-group-name ECSCanaryGroup \
```

```
  --deployment-config-name CodeDeployDefault.ECSCanary10Percent5Minutes \
  ...
```

5.4 Multi-Environment CI/CD Pipelines

Commonly used to isolate development, staging, and production workflows.

🧩 Strategy:

```
CodeCommit → CodePipeline (Dev)
                    ↓
              Manual Approval
                    ↓
          CodePipeline (Staging)
                    ↓
              Manual Approval
                    ↓
          CodePipeline (Prod)
```

🛠️ Pipeline Enhancement:

Add **Manual Approval** between stages using SNS:

```
{
  "name": "ApproveStage",
  "actions": [{
    "name": "ManualApproval",
    "actionTypeId": {
      "category": "Approval",
      "owner": "AWS",
      "provider": "Manual",
      "version": "1"
    },
    "configuration": {
      "NotificationArn": "arn:aws:sns:us-east-
1:123456789012:PipelineApproval"
    },
    "runOrder": 1
  }]
}
```

5.5 External Git Integration (GitHub/GitLab)

You can replace CodeCommit with GitHub or GitLab.

🔗 GitHub as Source (CodePipeline)

🛠️ **GUI:**

1. Source provider: **GitHub (Version 2)**
2. Connect GitHub account via OAuth
3. Choose repo and branch

💻 **CLI (GitHub Integration):**

```
"configuration": {
  "Owner": "your-github-user",
  "Repo": "your-repo-name",
  "Branch": "main",
  "OAuthToken": "encrypted_token_here"
}
```

Note: GitHub token should be stored securely using **Secrets Manager**.

5.6 Monitoring Advanced Deployments

Tool	Purpose
CloudWatch	Logs for Lambda, CodeBuild, ECS
CodeDeploy	Deployment status, error tracking
SNS	Notify on failures or approvals
CloudTrail	Audits of all pipeline activities

5.7 Rollback Mechanisms

Method	Rollback Support
CodeDeploy	Yes (Auto/Manual)
Lambda Alias	Yes (re-point alias)
ECS (Blue/Green)	Yes (switch target group)

5.8 Summary Table

Feature	CodeDeploy	Lambda	ECS Fargate
Blue/Green	✅	➖	✅

Canary	✅	✅	✅
Auto Rollback	✅	✅	✅
Load Balancer Support	✅	N/A	✅
Manual Approval Support	✅	N/A	✅

Chapter 6: Integrating Testing, Code Quality, and Security into CI/CD Pipelines

6.1 Overview

Modern software delivery isn't just about deploying fast — it's about deploying **secure**, **stable**, and **high-quality** applications. In this chapter, you'll learn how to:

- Add automated **unit tests**, **linting**, and **code coverage** reports
- Integrate tools for **static code analysis**, **open-source vulnerability detection**, and **security scanning**
- Use AWS-native tools like **Amazon CodeGuru**, **Secrets Manager**, **Inspector**, and **third-party tools** (like SonarQube, Trivy, or Checkov)
- Implement **"Shift-Left"** testing in your CI/CD lifecycle

6.2 Testing in CodeBuild

🧪 Add Unit Testing to `buildspec.yml`

📁 Sample: `buildspec.yml` with Jest (Node.js)

```
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 18
    commands:
      - npm install
  pre_build:
    commands:
      - echo "Running tests..."
      - npm run test -- --coverage
  build:
    commands:
      - echo "Building..."
      - npm run build
reports:
  jest_reports:
    files:
      - coverage/lcov-report/index.html
```

```
    base-directory: coverage
    discard-paths: yes
```

✅ Reports show up in CodeBuild's *Test Reports* section.

6.3 Code Quality Checks (Linting + Style)

🎯 Add ESLint or Pylint to your pipeline

📁 Node.js Example:

```
npm install eslint --save-dev
npx eslint .
```

📁 Python Example:

```
pip install pylint
pylint myapp/
```

Add to `buildspec.yml`:

```
  pre_build:
    commands:
      - echo "Linting the code"
      - npx eslint .
```

6.4 Static Code Analysis with Amazon CodeGuru

💡 CodeGuru Reviewer

Helps detect code quality and security issues using ML.

🔧 Requirements:

- Repo must be in CodeCommit or GitHub
- Enable CodeGuru Reviewer analysis

💻 CLI:

```
aws codeguru-reviewer create-code-review \
  --name MyReview \
  --type RepositoryAnalysis \
  --repository AssociationType=CodeCommit,Name=sample-webapp
```

6.5 Security in CI/CD

🔐 Secrets Detection with AWS Secrets Manager

1. **Store DB/API credentials securely:**

```
aws secretsmanager create-secret \
  --name prod/dbPassword \
  --secret-string '{"username":"admin","password":"secret123"}'
```

5. **Reference in `buildspec.yml`:**

```
env:
  secrets-manager:
    DB_PASSWORD: "prod/dbPassword:password"
```

6.6 Open-Source Dependency Scanning

✅ Use Trivy or Snyk in CodeBuild

🐬 Trivy (Docker & Dependency Scan):

```
curl -sfL
https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.s
h | sh
trivy fs .
```

📁 Add to `buildspec.yml`:

```
  pre_build:
    commands:
      - echo "Scanning dependencies for vulnerabilities"
      - trivy fs .
```

6.7 Infrastructure Security with Checkov

Check your Terraform or CloudFormation templates for misconfigurations.

```
pip install checkov
checkov -d .
```

Include in your build pipeline:

```
  pre_build:
    commands:
```

```
      - checkov -d .
```

6.8 Container Image Scanning in ECR

ECR supports **automated vulnerability scanning** using Amazon Inspector.

🛠️ Enable scanning:

```
aws ecr put-image-scanning-configuration \
  --repository-name my-app \
  --image-scanning-configuration scanOnPush=true
```

📊 View results:

AWS Console → ECR → Image Details → Vulnerabilities tab

6.9 Shift-Left Security and Testing Strategy

Practice	Benefit
Unit tests in CodeBuild	Catch bugs early
Lint + Style checks	Maintain consistency
Secrets in Secrets Manager	No hardcoded credentials
Code scanning via CodeGuru	Detect logic and security issues
Image scan (Trivy/Inspector)	Prevent vulnerable container deploys
IaC checks (Checkov, tfsec)	Infrastructure hygiene

6.10 Example: Secure `buildspec.yml` with All Stages

```
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 18
    commands:
      - npm install
      - curl -sfL
https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.s
h | sh
```

```
  pre_build:
    commands:
      - echo "Running lint"
      - npx eslint .
      - echo "Running unit tests"
      - npm run test
      - echo "Scanning dependencies"
      - trivy fs .
  build:
    commands:
      - echo "Building app"
      - npm run build
reports:
  jest_reports:
    files:
      - coverage/lcov-report/index.html
    base-directory: coverage
    discard-paths: yes
```

📘 Chapter 7: Observability — Monitoring, Logging, and Alarming in CI/CD and Deployment Pipelines

7.1 Overview

"**You can't improve what you can't measure**." Observability in DevOps means tracking the *health*, *performance*, and *errors* across your entire delivery pipeline, environments, and services.

In this chapter, you'll learn how to:

- Centralize logs with **CloudWatch Logs**
- Monitor metrics using **CloudWatch Metrics**
- Detect issues with **CloudWatch Alarms**
- Visualize system health using **CloudWatch Dashboards**
- Use **X-Ray**, **CloudTrail**, and **SNS** for tracing and alerting

7.2 Centralized Logging with CloudWatch Logs

Every AWS service — from CodeBuild to Lambda and ECS — emits logs to **CloudWatch Logs**.

🔧 CodeBuild Logs:

Automatically streamed to CloudWatch when you enable logging.

✅ CLI:

```
aws codebuild start-build --project-name my-app-build
aws logs get-log-events \
  --log-group-name /aws/codebuild/my-app-build \
  --log-stream-name build-log-stream-name
```

🔧 Lambda Logs:

Logs are automatically available in `/aws/lambda/function-name`.

```
aws logs tail /aws/lambda/myLambdaFunction --follow
```

7.3 Metrics Monitoring with CloudWatch

Service	Key Metrics
CodeBuild	BuildSuccess, Duration, FailedPhases
Lambda	Invocations, Errors, Duration, Throttles
ECS/EKS	CPU, Memory usage, Container restarts
CodePipeline	PipelineExecutionStart, PipelineExecutionFailed
Application	Custom business or error metrics via SDK

📊 Custom Metrics (CLI):

```
aws cloudwatch put-metric-data \
  --namespace "MyApp" \
  --metric-name "SuccessfulLogin" \
  --value 1 \
  --unit Count
```

7.4 Alarming on Failures and Thresholds

Use CloudWatch Alarms to get notified when thresholds are crossed.

📉 Alarm for Failed CodeBuild:

✅ CLI:

```
aws cloudwatch put-metric-alarm \
  --alarm-name "BuildFailureAlarm" \
  --metric-name "FailedBuilds" \
  --namespace "AWS/CodeBuild" \
  --statistic Sum \
  --period 300 \
  --threshold 1 \
  --comparison-operator GreaterThanOrEqualToThreshold \
  --evaluation-periods 1 \
  --dimensions Name=ProjectName,Value=my-app-build \
  --alarm-actions arn:aws:sns:us-east-1:123456789012:NotifyTeam
```

🔔 Set up SNS for Alerting:

```
aws sns create-topic --name NotifyTeam
```

```
aws sns subscribe \
  --topic-arn arn:aws:sns:us-east-1:123456789012:NotifyTeam \
  --protocol email \
  --notification-endpoint your-team@company.com
```

7.5 Dashboards for Deployment Health

Use **CloudWatch Dashboards** to monitor multiple CI/CD metrics and resource performance on one screen.

🛠️ Sample JSON Dashboard:

```
{
  "widgets": [
    {
      "type": "metric",
      "properties": {
        "metrics": [
          ["AWS/CodeBuild", "BuildDuration", "ProjectName", "my-app-
build"],
          ["AWS/CodeBuild", "FailedBuilds", "ProjectName", "my-app-build"]
        ],
        "title": "CodeBuild Overview",
        "view": "timeSeries",
        "region": "us-east-1"
      }
    }
  ]
}
```

CLI:

```
aws cloudwatch put-dashboard \
  --dashboard-name "CICD-Dashboard" \
  --dashboard-body file://dashboard.json
```

7.6 Tracing and Root Cause Analysis

📍 AWS X-Ray for Lambda & ECS:

- Auto-traces requests across microservices
- Tracks performance bottlenecks, latency spikes, and downstream errors

Enable via:

- Lambda Console → Monitor → Enable X-Ray

- ECS Task Definition → Enable X-Ray daemon

📜 CloudTrail:

Use for auditing who triggered a deployment or pipeline change.

```
aws cloudtrail lookup-events \
  --lookup-attributes
AttributeKey=EventName,AttributeValue=StartPipelineExecution
```

7.7 Monitoring Pipelines: CodePipeline & CodeDeploy

✅ Common Failure Metrics:

Tool	Metric
CodePipeline	PipelineExecutionFailed
CodeDeploy	DeploymentFailure, InstanceError
CodeBuild	FailedBuilds, Duration

View in AWS Console:

- Go to **CloudWatch → Metrics → AWS/CodePipeline**
- Set Alarm on "PipelineExecutionFailed"

7.8 Integrating with Third-Party Observability Tools

Tool	Purpose
DataDog	Cloud-native monitoring
Prometheus	Kubernetes metrics
Grafana	Visualize CloudWatch via plugin
ELK Stack	Centralize logs from AWS services
Sentry	Error tracking in frontend/backend

Example: Export logs from Lambda to ELK:

```
aws logs create-export-task \
```

```
--log-group-name /aws/lambda/my-function \
--from 1671120000000 \
--to 1671123600000 \
--destination s3-bucket-name \
--destination-prefix logs/lambda
```

7.9 Real-World Use Case: Full Monitoring Stack

📝 DevOps Workflow:

- CodeCommit → CodeBuild → CodePipeline → ECS
- CloudWatch Logs: Build & ECS logs
- Alarms: Build failed, CPU > 80%
- Dashboard: Build time, error counts
- X-Ray: Traces latency across services
- SNS: Sends alert to Slack via Lambda

7.10 Summary & Best Practices

✅ Checklist:

- Enable CloudWatch Logs for all stages
- Create dashboards for real-time health
- Alarm on errors & performance thresholds
- Use X-Ray and CloudTrail for deep root cause
- Integrate with Slack, Email, PagerDuty, or Opsgenie

📘 Chapter 8: Infrastructure as Code (IaC) for AWS DevOps – Automating Everything

8.1 What is Infrastructure as Code (IaC)?

Infrastructure as Code (IaC) means managing and provisioning cloud resources (servers, databases, networks, pipelines, etc.) through machine-readable configuration files.

Instead of manually creating AWS resources via the console, you use tools like:

- **AWS CloudFormation** (native, declarative YAML/JSON)
- **Terraform** (open-source, cloud-agnostic, HCL-based)
- **AWS CDK (Cloud Development Kit)** (programmatic IaC using Python, TypeScript, etc.)

Benefits:

- Version-controllable (stored in Git)
- Repeatable & auditable
- Environment consistency
- Quick disaster recovery

8.2 AWS CloudFormation for DevOps

🏗️ CloudFormation Template to Create a CodePipeline

pipeline.yaml

```
Resources:
  MyPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn: arn:aws:iam::123456789012:role/CodePipelineServiceRole
      Stages:
        - Name: Source
          Actions:
            - Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
```

```
              Provider: CodeCommit
              Version: '1'
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              RepositoryName: my-repo
              BranchName: main
      - Name: Build
        Actions:
          - Name: BuildAction
            ActionTypeId:
              Category: Build
              Owner: AWS
              Provider: CodeBuild
              Version: '1'
            InputArtifacts:
              - Name: SourceOutput
            Configuration:
              ProjectName: my-build-project
```

Deploy the stack:

```
aws cloudformation deploy \
  --template-file pipeline.yaml \
  --stack-name devops-pipeline \
  --capabilities CAPABILITY_IAM
```

8.3 Terraform for DevOps Automation

🌱 Define an ECR Repository and CodeBuild Project

main.tf

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_ecr_repository" "app_repo" {
  name = "my-app-repo"
}

resource "aws_codebuild_project" "build_app" {
  name          = "my-app-build"
  service_role  = aws_iam_role.codebuild_role.arn
  artifacts {
    type = "NO_ARTIFACTS"
  }
  environment {
    compute_type = "BUILD_GENERAL1_SMALL"
    image        = "aws/codebuild/standard:6.0"
```

```
    type          = "LINUX_CONTAINER"
  }
  source {
    type      = "GITHUB"
    location  = "https://github.com/my-org/my-app"
    buildspec = "buildspec.yml"
  }
}

resource "aws_iam_role" "codebuild_role" {
  name = "codebuild-service-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = { Service = "codebuild.amazonaws.com" }
      Action = "sts:AssumeRole"
    }]
  })
}
```

Deploy with Terraform:

```
terraform init
terraform apply
```

8.4 AWS CDK for Programmatic DevOps

CDK lets you use your favorite programming language to define infrastructure.

🧱 CDK CodePipeline in Python

Install:

```
pip install aws-cdk-lib constructs
cdk init app --language python
```

```
pipeline_stack.py
```

```
from aws_cdk import (
    Stack, pipelines, aws_codecommit as codecommit
)
from constructs import Construct

class PipelineStack(Stack):
    def __init__(self, scope: Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        repo = codecommit.Repository.from_repository_name(
```

```
            self, "Repo", "my-repo"
        )

        pipeline = pipelines.CodePipeline(self, "Pipeline",
            synth=pipelines.ShellStep("Synth",
                input=pipelines.CodePipelineSource.code_commit(repo,
"main"),
                commands=["npm install -g aws-cdk", "cdk synth"]
            )
        )
```

Deploy:

```
cdk deploy
```

8.5 Real-World Use Case: ECS Service with IaC

- Use Terraform to define:
  - VPC, Subnets
  - ECS Cluster + Fargate service
  - ECR + CodeBuild + CodePipeline
  - CloudWatch Dashboards & Alarms
- Use CloudFormation or CDK to:
  - Add SNS notifications
  - Add Lambda approvals
  - Secure pipeline roles

8.6 Best Practices for IaC in DevOps

Practice	Why It's Important
Version-control all IaC	Enables rollback and traceability
Use remote state for Terraform	Prevents state file corruption
Modularize IaC code	Easier reuse and testing
Use IAM least privilege	Avoid wide-open permissions
Validate configs before apply	Use `terraform validate`, `cfn-lint`
Tag all resources	Helps in cost analysis & tracking

8.7 Tools You Should Know

Tool	Purpose
Terraform	Multi-cloud IaC tool (HCL)
AWS CloudFormation	AWS-native YAML/JSON IaC
AWS CDK	Dev-friendly IaC (Python/TS)
Pulumi	Alternative to CDK with more language support
CFN Lint	Linting for CloudFormation
TFLint / Checkov	Security + compliance for IaC

8.8 IaC Automation for CI/CD Itself

Yes, even your **CI/CD pipelines** should be created and versioned using IaC!

✅ Things to automate:

- IAM Roles & Policies
- CodePipeline, CodeBuild, CodeDeploy
- SNS topics for notifications
- CloudWatch dashboards for pipelines
- Lambda approvals

📘 Chapter 9: Containers & Orchestration — Automating Deployments with ECR, ECS, and EKS

9.1 Why Containers in DevOps?

Containers have become foundational in modern software engineering because they:

✅ **Package Applications and Dependencies Together** — making deployments reproducible across environments.
✅ **Achieve Portability** — run reliably across development, test, and production environments.
✅ **Provide Scalability** — launch, update, and terminate instances quickly based on load.
✅ **Support Isolation** — separate workloads and reduce cross-application impacts.

In AWS DevOps, containers standardize application packaging and deployment, making them ideal for automating pipelines with **CI/CD** and **IaC** tools.

9.2 AWS Container Services Overview

Service	Description	Use Case
ECR	Elastic Container Registry	Store and version Docker images securely
ECS	Elastic Container Service	Run containers on EC2 instances or Fargate
EKS	Elastic Kubernetes Service	Run Kubernetes workloads at scale
Fargate	Serverless compute for containers	Run ECS/EKS tasks/pods with no servers to manage

9.3 Working with Amazon ECR

ECR acts as a **secure, highly available, and managed registry** for your container images.

✅ Push an Image to ECR

Step 1: Create the Repository

```
aws ecr create-repository --repository-name my-app
```

Step 2: Authenticate Docker to ECR

```
aws ecr get-login-password \
  | docker login --username AWS \
    --password-stdin <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com
```

Step 3: Build, Tag, and Push

```
docker build -t my-app .
docker tag my-app:latest <aws_account_id>.dkr.ecr.us-east-
1.amazonaws.com/my-app:latest
docker push <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-app:latest
```

✅ Terraform ECR Module

```
resource "aws_ecr_repository" "app" {
  name = "my-app"
}
```

9.4 Running Containers on ECS

ECS is a robust, fully managed container orchestration service for running workloads. It supports:

- **EC2 launch type** — You manage servers.
- **Fargate launch type** — No servers to manage.

✅ ECS Fargate Deployment Example

```
resource "aws_ecs_cluster" "app_cluster" {
  name = "app-cluster"
}

resource "aws_ecs_task_definition" "app_task" {
  family                   = "app-task"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "256"
  memory                   = "512"
  network_mode             = "awsvpc"
  execution_role_arn       = aws_iam_role.ecs_task_exec.arn
  container_definitions    = jsonencode([{
    name      = "my-app"
    image     = "<aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-
app:latest"
    essential = true
    portMappings = [{ containerPort = 80 }]
  }])
}
```

```
resource "aws_ecs_service" "app_service" {
  name            = "app-service"
  cluster         = aws_ecs_cluster.app_cluster.id
  task_definition = aws_ecs_task_definition.app_task.arn
  desired_count   = 2
  launch_type     = "FARGATE"

  network_configuration {
    subnets         = [aws_subnet.public.id]
    assign_public_ip = true
    security_groups = [aws_security_group.ecs_sg.id]
  }
}
```

9.5 Automating ECS Deployments with AWS CodePipeline

✅ End-to-End Flow

1. **CodeCommit ➔** Push code.
2. **CodeBuild ➔** Build Docker image and Push to ECR.
3. **CodeDeploy/ECS ➔** Deploy new version of service.

✅ Sample buildspec.yml

```
version: 0.2
phases:
  pre_build:
    commands:
      - $(aws ecr get-login-password | docker login \
          --username AWS \
          --password-stdin <account>.dkr.ecr.<region>.amazonaws.com)
  build:
    commands:
      - docker build -t my-app .
      - docker tag my-app:latest <ecr_repo_uri>:latest
  post_build:
    commands:
      - docker push <ecr_repo_uri>:latest
      - printf '[{"name":"my-app","imageUri":"%s"}]' \
          <ecr_repo_uri>:latest > imagedefinitions.json
artifacts:
  files: imagedefinitions.json
```

9.6 Amazon EKS — Kubernetes on AWS

EKS provides a fully managed Kubernetes control plane. It's ideal if your team already uses **kubectl**, Helm charts, or Kubernetes CRDs.

✅ Simple EKS Deployment

Create an EKS Cluster

```
eksctl create cluster \
  --name my-eks-cluster \
  --region us-east-1 \
  --node-type t3.medium \
  --nodes 2
```

Deploy an App

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

Example Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-
app:latest
        ports:
        - containerPort: 80
```

9.7 Choosing Between ECS and EKS

Feature	ECS	EKS
Simplicity	✅ Easier to operate	❌ Steeper learning curve
Flexibility	❌ AWS-only	✅ Kubernetes-standard
Cost	✅ Cheaper (no control node)	❌ Additional control-plane cost

Ecosystem	❌ Proprietary tooling	✅ Rich Kubernetes ecosystem
Use Case	Simple APIs, batch jobs	Complex microservices, multi-cloud deployments

9.8 Real-World CI/CD Use Case (ECS + ECR)

📝 Scenario:

✅ App source code stored in GitHub.
✅ Build and push a Docker image to ECR using **CodeBuild**.
✅ Deploy to ECS Fargate Service using **CodeDeploy**.
✅ Monitor using **CloudWatch Logs** and **X-Ray**.

🔥 Tools Involved:

- **Terraform**: Create ECR, ECS, IAM roles.
- **buildspec.yml**: Build/push Docker image.
- **imagedefinitions.json**: ECS Service deployment.
- **CloudWatch Alarms**: Alerting & auto-recovery.

9.9 Best Practices for Containerized Applications

Practice	Description
🤿 **Scan Images Before Deployment**	Use Amazon Inspector or ECR built-in scanning.
🔐 **Use Least Privilege IAM**	Assign task roles vs EC2 roles for fine-grained access.
🔑 **Use AWS Secrets Manager**	Never hardcode secrets in images or deployments.
📈 **Enable Logging**	Route ECS/EKS logs to Amazon CloudWatch Logs.
⚡ **Use Least Privilege Ports**	Expose only required ports and protocols.

✅ Summary

In this chapter, you learned:

- Why containers matter for DevOps.
- The differences between **ECR**, **ECS**, and **EKS** and when to use each.
- A hands-on example of deploying containers via ECS Fargate.
- How to build and push images to ECR, and use CodePipeline to enable fully automated deployments.

- Best practices for making deployments secure and resilient.

With these patterns and best practices, you're well-prepared to implement container-based deployments for any AWS-powered application.

📘 Chapter 10: DevSecOps – Integrating Security into Every Step of AWS DevOps

10.1 What is DevSecOps?

DevSecOps integrates **security practices into the DevOps lifecycle**, ensuring that code, infrastructure, and deployment processes are **secure by default**.

🔒 DevSecOps = DevOps + Continuous Security

- **Shift Left**: Identify vulnerabilities early (code, build)
- **Automate**: Integrate scanning into CI/CD
- **Monitor**: Ongoing logging, alerts, audit trails
- **Comply**: Automate governance and policy enforcement

10.2 Key Pillars of AWS DevSecOps

Area	AWS Service/Tool	Purpose
Code Scanning	CodeGuru, SonarQube	Catch security bugs, code smells
IaC Security	AWS Config, Checkov, tfsec	Detect drift, validate IaC
Container Scanning	Amazon Inspector, Trivy	Scan ECR images for CVEs
Secrets Mgmt	AWS Secrets Manager, SSM	Store credentials securely
Policy Enforcement	SCPs, IAM policies	Limit risky operations
Logging & Audit	CloudTrail, GuardDuty	Detect malicious activity

10.3 Securing the CI/CD Pipeline

🛡️ Step-by-Step Security Integration

1. Pre-Commit Hooks Use pre-commit to enforce security checks:

```
repos:
- repo: https://github.com/antonbabenko/pre-commit-terraform
  rev: v1.77.0
  hooks:
    - id: terraform_fmt
    - id: terraform_validate
    - id: terraform_tflint
```

2. Code Scanning (CI)

- **AWS CodeGuru**:

```
aws codeguru-reviewer create-code-review \
  --name my-review \
  --type RepositoryAnalysis \
  --repository-association-arn arn:aws:codeguru:...
```

- **SonarQube in CodeBuild**:

```
phases:
  install:
    commands:
      - wget https://binaries.sonarsource.com/...
  build:
    commands:
      - sonar-scanner -Dsonar.projectKey=myApp ...
```

3. Image Scanning in ECR

- Enable **Amazon Inspector**:

```
aws inspector2 enable --resource-types ECR
```

- Automatically scans images on push.

10.4 Securing Infrastructure as Code (IaC)

✅ **Tools:**

- **Checkov** (Python-based IaC scanner)
- **TFLint / tfsec** (Terraform-focused)
- **AWS Config** (Cloud compliance as code)

Example: Checkov with GitHub Actions

```
- name: Checkov IaC Scan
```

```
  uses: bridgecrewio/checkov-action@v12
  with:
    directory: ./terraform
```

Terraform IAM Best Practice

```
resource "aws_iam_policy" "least_privilege" {
  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Action   = ["s3:PutObject", "s3:GetObject"],
      Effect   = "Allow",
      Resource = "arn:aws:s3:::my-bucket/*"
    }]
  })
}
```

10.5 Secrets Management in DevOps

🔐 **Use:**

- **AWS Secrets Manager**
- **AWS SSM Parameter Store (SecureString)**

CLI Example:

```
aws secretsmanager create-secret \
  --name my-db-password \
  --secret-string '{"username":"admin","password":"Secr3tPass"}'
```

Use in ECS task:

```
"secrets": [
  {
    "name": "DB_PASSWORD",
    "valueFrom": "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-
db-password"
  }
]
```

10.6 Real-time Threat Detection

Tool	What It Does
CloudTrail	Audits API calls, tracks user actions

CloudWatch Logs	Monitors app logs and metrics
GuardDuty	Detects threats using AI models
Security Hub	Aggregates all security findings

GuardDuty Enablement (CLI)

```
aws guardduty create-detector --enable
```

CloudTrail Log Bucket Policy (Restrict tampering)

```
{
  "Statement": [
    {
      "Sid": "AllowCloudTrailWrite",
      "Effect": "Allow",
      "Principal": { "Service": "cloudtrail.amazonaws.com" },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::my-logs-bucket/AWSLogs/*",
      "Condition": { "StringEquals": { "s3:x-amz-acl": "bucket-owner-full-
control" } }
    }
  ]
}
```

10.7 Enforcing Policies & Compliance

✅ Use:

- **IAM Policies**
- **Service Control Policies (SCPs)**
- **AWS Config Rules**

SCP Example: Deny All Unless Tagged

```
{
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:TagKeys": ["CostCenter"]
        }
      }
    }
```

```
  ]
}
```

10.8 DevSecOps with GitHub + Terraform + AWS

Real-World Flow:

- Developer commits Terraform & app code
- GitHub Actions run Checkov, tfsec, Trivy
- CodeBuild builds and scans Docker image
- If clean, deploys to ECS/EKS via Terraform
- Slack alert on scan results (via Lambda or EventBridge)

10.9 DevSecOps Dashboarding and Alerts

- **Security Hub** + **EventBridge** → Alert on severity
- **CloudWatch Dashboards** → Visualize resource metrics
- **Trusted Advisor** → Security checks on S3, IAM, etc.
- **AWS Chatbot** → Slack notifications

10.10 Best Practices Summary

Practice	Benefit
Automate security scans	Enforces compliance early
Store secrets securely	Avoids hardcoded credentials
Use IAM least privilege	Minimizes blast radius
Enable centralized logging	Helps during audits/investigations
Scan IaC & containers	Detect config drift and CVEs
Enable all native services	Leverage GuardDuty, Inspector, etc.

📘 Chapter 11: Monitoring, Logging & Observability in AWS DevOps

11.1 Why Monitoring & Observability Matter

Modern DevOps practices depend on:

- **Monitoring**: Detecting performance anomalies
- **Logging**: Auditing behavior and debugging issues
- **Tracing**: Understanding distributed interactions
- **Alerting**: Reacting in real-time to outages

These are key to achieving **Mean Time to Detection (MTTD)** and **Mean Time to Recovery (MTTR)** goals.

11.2 Core AWS Monitoring & Observability Services

Service	Purpose
CloudWatch	Metrics, logs, dashboards, and alarms
CloudTrail	Governance: who did what, where, and when
AWS X-Ray	Distributed tracing of microservices
OpenTelemetry on AWS	Vendor-neutral observability
AWS Distro for OpenTelemetry (ADOT)	Export metrics/traces to CloudWatch, X-Ray, etc.

11.3 Amazon CloudWatch in DevOps

🛠️ Core Features:

- **Metrics**: CPU, memory, disk, custom metrics
- **Logs**: Real-time log collection and retention
- **Dashboards**: Visualize system performance
- **Alarms**: Notify on threshold breaches

- **Insights**: Search and analyze logs

📌 Use Case: Monitor EC2 CPU and Send Slack Alert

✅ Step 1: Create Alarm via CLI

```
aws cloudwatch put-metric-alarm \
  --alarm-name "HighCPUUtilization" \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-0abc123def456gh78 \
  --evaluation-periods 1 \
  --alarm-actions arn:aws:sns:us-east-1:123456789012:my-sns-topic
```

✅ Step 2: Configure SNS to trigger Lambda or Slack Notification

🛠️ CloudWatch Dashboard (Terraform Example)

```
resource "aws_cloudwatch_dashboard" "app_dash" {
  dashboard_name = "AppDashboard"
  dashboard_body = jsonencode({
    widgets = [
      {
        type = "metric",
        x = 0, y = 0, width = 12, height = 6,
        properties = {
          metrics = [["AWS/EC2", "CPUUtilization", "InstanceId", "i-
0abc123def456gh78"]],
          period = 300,
          stat = "Average",
          region = "us-east-1",
          title = "EC2 CPU Usage"
        }
      }
    ]
  })
}
```

11.4 CloudWatch Logs

✅ Real-time log ingestion and search

- EC2, ECS, Lambda, EKS
- Log retention and metric filters

CLI Example: Push Custom Log

```
aws logs create-log-group --log-group-name /devops/myapp
aws logs create-log-stream --log-group-name /devops/myapp --log-stream-name
stream1

aws logs put-log-events \
  --log-group-name /devops/myapp \
  --log-stream-name stream1 \
  --log-events timestamp=$(date +%s000),message="Test log entry"
```

Log Retention Settings:

```
aws logs put-retention-policy \
  --log-group-name /devops/myapp \
  --retention-in-days 30
```

11.5 CloudTrail – Governance & Auditing

CloudTrail records all API actions and user activities across your AWS account.

Capability	Benefit
Track user access	Who changed what, when
Detect misconfig	Alert on unauthorized actions
Feed into SIEMs	Centralize security logging

Enable CloudTrail (CLI)

```
aws cloudtrail create-trail \
  --name DevOpsTrail \
  --s3-bucket-name my-cloudtrail-bucket

aws cloudtrail start-logging --name DevOpsTrail
```

11.6 AWS X-Ray – Distributed Tracing

X-Ray helps visualize and debug distributed applications (e.g., microservices on ECS, EKS, Lambda).

Feature	Description
Service map	Trace user requests across components
Annotations	Add metadata for filtering and grouping
Latency heatmap	Pinpoint bottlenecks

Integration Example:

In Lambda:

```
from aws_xray_sdk.core import patch_all, xray_recorder
patch_all()

def lambda_handler(event, context):
    xray_recorder.begin_subsegment('DBQuery')
    # Do something
    xray_recorder.end_subsegment()
```

In ECS (with sidecar or agent):

- Use the X-Ray daemon

11.7 AWS Distro for OpenTelemetry (ADOT)

- **Vendor-neutral metrics and traces**
- Export to: CloudWatch, Prometheus, Grafana, Datadog
- Works with ECS, EKS, EC2, Lambda

Example ADOT Collector Config:

```
receivers:
  otlp:
    protocols:
      grpc:
      http:

exporters:
  awsxray:
  awscloudwatch:
    log_group_name: /adot/logs

service:
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [awsxray]
```

```
  metrics:
    receivers: [otlp]
    exporters: [awscloudwatch]
```

11.8 Observability Dashboards & Real-Time Alerts

🔔 Alert Routing (Amazon EventBridge + SNS + Lambda)

- **CloudWatch Alarm → EventBridge**
- **EventBridge Rule → SNS Topic**
- **SNS → Slack / Teams via Lambda Webhook**

11.9 Cost and Log Retention Management

Tip	Description
Set log retention	Don’t store logs indefinitely
Archive to S3	Use lifecycle rules
Use filters	Reduce ingestion costs
Delete old metrics	Unused custom metrics incur cost

✅ Summary: Best Practices

Practice	Why It Matters
Use dashboards	Visualize trends in real time
Enable X-Ray	Trace requests end to end
Retain only what’s needed	Reduce costs
Monitor everything	Proactive issue detection
Integrate with alerts	React immediately

📘 Chapter 12: Hands-On CI/CD Pipelines Using AWS DevOps Tools

12.1 Introduction to AWS CI/CD Stack

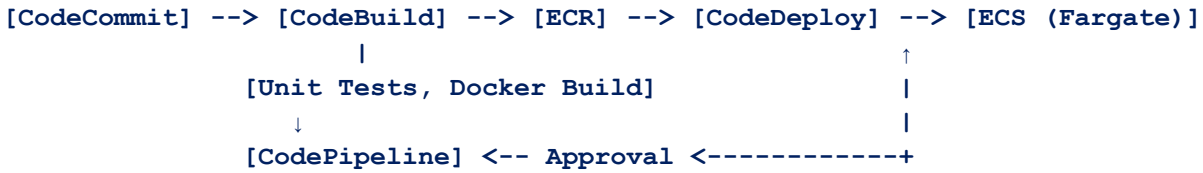
AWS Tool	Purpose
CodeCommit	Git repository (like GitHub)
CodeBuild	Build & test code
CodeDeploy	Deploy to EC2, Lambda, ECS
CodePipeline	Orchestrate CI/CD flow
ECR	Docker image registry

12.2 CI/CD Pipeline Overview

🎯 Use Case:

Deploy a containerized Node.js app to **Amazon ECS (Fargate)** when a developer pushes to the `main` branch.

12.3 Architecture Diagram



12.4 Step-by-Step Implementation

Step 1: Create a CodeCommit Repository

🖥️ **GUI:**

- Go to AWS CodeCommit → Create Repository

✅ **CLI:**

```
aws codecommit create-repository \
  --repository-name nodejs-ecs-pipeline \
  --repository-description "CI/CD repo for ECS"
```

Step 2: Create Dockerfile in Repo

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "app.js"]
EXPOSE 3000
```

Step 3: Create `buildspec.yml` for CodeBuild

```
version: 0.2

phases:
  install:
    runtime-versions:
      docker: 18
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
      - REPO_NAME=nodejs-ecs
      - IMAGE_TAG=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
  build:
    commands:
      - docker build -t $REPO_NAME:$IMAGE_TAG .
      - docker tag $REPO_NAME:$IMAGE_TAG
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - docker push
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$REPO_NAME:$IMAGE_TAG
      - printf '[{"name":"nodejs-app","imageUri":"%s"}]'
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$REPO_NAME:$IMAGE_TAG >
imagedefinitions.json
artifacts:
  files: imagedefinitions.json
```

Step 4: Create an ECR Repository

```
aws ecr create-repository \
  --repository-name nodejs-ecs \
  --image-scanning-configuration scanOnPush=true
```

Step 5: ECS Task Definition Template (`taskdef.json`)

```
{
  "family": "nodejs-ecs-task",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256",
  "memory": "512",
  "containerDefinitions": [
    {
      "name": "nodejs-app",
      "image": "",
      "portMappings": [{ "containerPort": 3000 }]
    }
  ]
}
```

Step 6: Create CodeDeploy App and Deployment Group

```
aws deploy create-application \
  --application-name nodejs-app \
  --compute-platform ECS

aws deploy create-deployment-group \
  --application-name nodejs-app \
  --deployment-group-name nodejs-deploy-group \
  --service-role-arn arn:aws:iam::<account>:role/CodeDeployRole \
  --deployment-config-name CodeDeployDefault.ECSAllAtOnce \
  --ecs-services serviceName=nodejs-service,clusterName=nodejs-cluster \
  --target-group-pair-info "..."  # Optional for Blue/Green
```

Step 7: Create CodePipeline

```
aws codepipeline create-pipeline --cli-input-json file://pipeline.json
```

`pipeline.json` **(simplified):**

```
{
  "pipeline": {
```

```
"name": "NodejsECSPipeline",
"roleArn": "arn:aws:iam::123456789012:role/CodePipelineRole",
"artifactStore": {
  "type": "S3",
  "location": "my-pipeline-artifacts"
},
"stages": [
  {
    "name": "Source",
    "actions": [{
      "name": "Source",
      "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
      },
      "outputArtifacts": [{ "name": "SourceOutput" }],
      "configuration": {
        "RepositoryName": "nodejs-ecs-pipeline",
        "BranchName": "main"
      }
    }]
  },
  {
    "name": "Build",
    "actions": [{
      "name": "DockerBuild",
      "actionTypeId": {
        "category": "Build",
        "owner": "AWS",
        "provider": "CodeBuild",
        "version": "1"
      },
      "inputArtifacts": [{ "name": "SourceOutput" }],
      "outputArtifacts": [{ "name": "BuildOutput" }],
      "configuration": {
        "ProjectName": "NodejsBuildProject"
      }
    }]
  },
  {
    "name": "Deploy",
    "actions": [{
      "name": "ECSDeploy",
      "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "provider": "CodeDeployToECS",
        "version": "1"
      },
      "inputArtifacts": [{ "name": "BuildOutput" }],
      "configuration": {
```

```
            "ApplicationName": "nodejs-app",
            "DeploymentGroupName": "nodejs-deploy-group",
            "TaskDefinitionTemplateArtifact": "BuildOutput",
            "AppSpecTemplateArtifact": "BuildOutput"
          }
        }]
      }
    ]
  }
}
```

12.5 Optional: Manual Approval Stage

```
{
  "name": "Approval",
  "actions": [{
    "name": "ManualApproval",
    "actionTypeId": {
      "category": "Approval",
      "owner": "AWS",
      "provider": "Manual",
      "version": "1"
    },
    "runOrder": 1
  }]
}
```

Add it after "Build" and before "Deploy" stages.

12.6 Monitor Your Pipeline

- View in **CodePipeline Console**
- Use **CloudWatch Events** to trigger notifications on failure
- Create **Alarms** for build duration or status

12.7 Full GitHub-Based Variant (Optional)

You can replace CodeCommit with GitHub in the "Source" stage:

```
"configuration": {
  "Owner": "your-github-username",
  "Repo": "your-repo-name",
  "Branch": "main",
```

```
  "OAuthToken": "****"
}
```

✅ Summary

Step	Tool Used	Outcome
Version control	CodeCommit or GitHub	Source of truth
Build & Scan	CodeBuild	Secure Docker image
Artifact Storage	ECR + S3	Image + metadata
Orchestration	CodePipeline	CI/CD Flow
Deployment	CodeDeploy	Live update ECS

📘 Chapter 13: Deployment Strategies in AWS DevOps

🧭 13.1 Introduction

In DevOps, deploying updates to production **without downtime**, **safely**, and with **rollback options** is a key goal. AWS offers multiple deployment strategies that align with different risk profiles and environments (web apps, APIs, microservices, etc.).

This chapter explores the major deployment strategies used in AWS DevOps pipelines, including their advantages, trade-offs, and **step-by-step implementations** using AWS services like CodeDeploy, ECS, EKS, and Lambda.

🚦 13.2 Types of Deployment Strategies

Strategy	Description	Use Case
All-at-Once	Deploys to all instances immediately	Fast updates, high risk
Rolling	Updates a few instances at a time	Gradual rollout, less risk
Blue/Green	Swaps traffic between old and new environments	Zero-downtime, safe rollback
Canary	Routes traffic to a small % of users first	Measure impact before full rollout
A/B Testing	Divides traffic for testing different versions/features	Marketing, user behavior testing
Shadow	New version receives traffic but doesn't affect real users	Observability testing
Feature Toggle	Feature is live but hidden/controlled by flags	Safe experimentation

📐 13.3 Implementing Deployment Strategies in AWS

🚀 A. All-at-Once Deployment (CodeDeploy)

Quickest but highest risk. Useful in dev/staging.

```
"deploymentConfigName": "CodeDeployDefault.AllAtOnce"
```

CLI Example:

```
aws deploy create-deployment \
  --application-name MyApp \
  --deployment-group-name MyDG \
  --s3-location bucket=my-bucket,key=myapp.zip,bundleType=zip \
  --deployment-config-name CodeDeployDefault.AllAtOnce
```

🔁 B. Rolling Deployment

Deploys in batches, e.g., 2 instances at a time.

```
"deploymentConfigName": "CodeDeployDefault.HalfAtATime"
```

Ideal for EC2 & ECS

🌐 C. Blue/Green Deployment

✅ For EC2/ECS (with CodeDeploy)

Blue: Live version
Green: New version, ready to switch

```
aws deploy create-deployment \
  --deployment-group-name MyBlueGreenGroup \
  --deployment-config-name CodeDeployDefault.ECSAllAtOnce \
  --auto-rollback-configuration enabled=true
```

For ECS, use a **load balancer** with a **listener rule switch**.

✅ For Lambda (Native Support)

```
aws lambda update-alias \
  --function-name my-function \
  --name LIVE \
  --function-version 2 \
```

```
  --routing-config '{"AdditionalVersionWeights":{"1":0.1}}'
```

- `version 2` gets 90%
- `version 1` keeps 10% (Canary)

🐤 D. Canary Deployment (Lambda + ECS)

Start with a small % of traffic, then increase if stable.

For Lambda (Native Canary):

```
"DeploymentPreference": {
  "Type": "Canary10Percent5Minutes",
  "Alarms": [...],
  "Hooks": {
    "PreTraffic": "TestHook",
    "PostTraffic": "MonitorHook"
  }
}
```

With CodeDeploy on ECS:

Set `canary` configuration:

```
"deploymentConfigName": "CodeDeployDefault.ECSCanary10Percent5Minutes"
```

🧪 E. A/B Testing

Use **Amazon CloudFront**, **Lambda@Edge**, or **Application Load Balancer** to split traffic.

Lambda@Edge Example to redirect 20% of users:

```
if (Math.random() < 0.2) {
  response.headers['location'] = [{ key: 'Location', value: '/v2/' }];
}
```

🔀 F. Feature Toggles

Use **LaunchDarkly**, **Unleash**, or custom config (e.g., in DynamoDB or SSM Parameter Store) to control feature rollout.

CLI Example: Feature Toggle via Parameter Store

```
aws ssm put-parameter \
  --name "/features/new_ui" \
  --value "enabled" \
  --type "String"
```

App logic:

```
if get_param("/features/new_ui") == "enabled":
    show_new_ui()
else:
    show_old_ui()
```

📦 13.4 Deployment Strategy in CodePipeline

Example Pipeline JSON for Blue/Green:

```
"Actions": [
  {
    "Name": "DeployApp",
    "ActionTypeId": {
      "Category": "Deploy",
      "Owner": "AWS",
      "Provider": "CodeDeploy",
      "Version": "1"
    },
    "Configuration": {
      "ApplicationName": "MyApp",
      "DeploymentGroupName": "MyBGGroup"
    }
  }
]
```

🧠 13.5 Best Practices by Strategy

Strategy	Best Practice
All-at-Once	Use only in dev/staging
Rolling	Monitor health per batch
Blue/Green	Automate DNS/LB switch with rollback
Canary	Use alarms for auto rollback
A/B Testing	Use analytics to measure impact
Feature Toggle	Store toggles in config service (e.g., SSM, DynamoDB)

🛡️ 13.6 Rollback Mechanisms

Always enable **auto rollback** in CodeDeploy:

```
"autoRollbackConfiguration": {
  "enabled": true,
  "events": ["DEPLOYMENT_FAILURE", "DEPLOYMENT_STOP_ON_ALARM"]
}
```

Enable **alarms** in CloudWatch for quick rollback:

```
aws cloudwatch put-metric-alarm \
  --alarm-name High5xxErrors \
  --metric-name 5XXErrorRate \
  --threshold 5 \
  --comparison-operator GreaterThanThreshold
```

🔧 13.7 Hands-On Scenario: ECS Blue/Green Deployment

1. Push app image to ECR
2. Register new ECS Task Definition
3. Create a new version of service as “green”
4. Use CodeDeploy with LoadBalancer and switch traffic
5. Auto rollback if health fails

```
aws ecs update-service \
  --cluster my-cluster \
  --service my-service \
  --task-definition my-app:v2 \
  --deployment-configuration "maximumPercent=200,minimumHealthyPercent=100"
```

✅ 13.8 Summary

Strategy	Downtime Risk	Rollback Ease	Setup Complexity
All-at-Once	High	Low	Simple
Rolling	Low	Medium	Moderate
Blue/Green	None	High	Moderate-High
Canary	Low	High	High
A/B Testing	None	Manual	High
Feature Toggle	None	High	App logic required

📘 Chapter 14: Cost Optimization in AWS DevOps Pipelines

💰 14.1 Introduction

Cost optimization in DevOps isn't just about reducing bills — it's about **maximizing efficiency**, **paying only for what you use**, and ensuring that **speed and reliability** aren't sacrificed for savings.

AWS offers multiple services and tools to help you **monitor**, **analyze**, and **automate cost controls** across your DevOps pipelines — from source to production.

🧱 14.2 Areas to Optimize in DevOps Pipelines

Area	Common Cost Drains	Optimization Techniques
CodeBuild	Unused compute time, high specs	Right-size build environments, timeout config
CodePipeline	Idle pipelines, frequent triggers	Event-driven, use source filters
Testing	Long test runtimes, resource-heavy tests	Parallelism, granular testing, serverless test
Deployments	Over-provisioned EC2, rollback retries	Use Fargate/Lambda, optimize rollback attempts
Artifact Storage	Large binary logs in S3, long retention	Lifecycle policies, compress artifacts
Monitoring & Logs	Too many log streams, high retention	Use filters, set log retention policies

⚙️ 14.3 CodeBuild Cost Optimization

✅ Choose Right Build Environment

Use only what you need:

```
environment:
  computeType: BUILD_GENERAL1_SMALL  # or .MEDIUM, .LARGE
```

Type	vCPUs	Memory	Cost/hour (approx)
`BUILD_GENERAL1_SMALL`	2	3 GB	$0.005/min
`BUILD_GENERAL1_LARGE`	8	15 GB	$0.02/min

✅ Timeout Control

Limit idle build time:

```
timeoutInMinutes: 10
```

✅ Use Spot Builds

```
aws codebuild start-build \
  --project-name MyProject \
  --environment-variables-override name=USE_SPOT,value=true
```

In `buildspec.yml`:

```
environment:
  type: LINUX_CONTAINER
  privilegedMode: true
  computeType: BUILD_GENERAL1_SMALL
```

🛠️ 14.4 Optimizing CodePipeline Usage

✅ Event-Based Triggers

Use S3 or GitHub Webhooks instead of polling:

```
"PollForSourceChanges": false
```

Enable trigger filter for branch/tag:

```
"GitHubSource": {
  "Branch": "main",
  "WebhookFilters": [
    {
      "Type": "EVENT",
      "Pattern": "PUSH"
    }
  ]
```

```
}
```

✅ Disable Unused Pipelines Temporarily

```
aws codepipeline disable-pipeline --name MyPipeline
```

🧪 14.5 Testing Cost Optimization

✅ Use Serverless Testing with Lambda

Split tests across multiple Lambda functions or containers.

✅ Parallelize Testing

In `buildspec.yml`:

```
phases:
  build:
    commands:
      - pytest tests/unit/
      - pytest tests/integration/ &
```

✅ Use Caching

Avoid redownloading dependencies:

```
cache:
  paths:
    - '/root/.m2/**/*'
    - 'node_modules/**/*'
```

📦 14.6 Artifact & Storage Cost Controls

✅ Compress Artifacts

```
zip -r app.zip ./build
```

Use GZIP or Brotli for logs and reports.

✅ Set S3 Lifecycle Policy

```
{
  "Rules": [
```

```
    {
      "ID": "ExpireArtifacts",
      "Prefix": "artifacts/",
      "Status": "Enabled",
      "Expiration": {
        "Days": 7
      }
    }
  ]
}
```

📊 14.7 Logging & Monitoring Cost Optimization

✅ Set Retention Periods for Logs

```
aws logs put-retention-policy \
  --log-group-name /aws/codebuild/myproject \
  --retention-in-days 7
```

✅ Use Filters Instead of Full Streams

Only capture logs you need using filters or CloudWatch Metric Filters.

🧠 14.8 Real-World Tips

Practice	Tool or Service	Cost Impact
Use Lambda for approvals	AWS Lambda	Avoid EC2 overhead
Use Step Functions instead of EC2	AWS Step Functions	Serverless cost
Scan repos during commit, not build	Git hooks + Snyk/ESLint	Faster pipeline
Schedule test pipelines off hours	EventBridge	Avoid idle builds
Use CodeArtifact instead of S3	AWS CodeArtifact	Granular billing

🔐 14.9 Cost Guardrails in DevOps

- **Budgets**: Set AWS Budgets for pipelines and notifications
- **Quotas**: Use Service Quotas for build limits
- **Alerts**: Use CloudWatch + SNS to alert on usage

Example: Create Budget for CodeBuild

```
aws budgets create-budget \
  --account-id 1234567890 \
  --budget-name "CodeBuildBudget" \
  --budget-type COST \
  --time-unit MONTHLY \
  --budget-limit Amount=10,Unit=USD
```

🧹 14.10 Automation: Clean Up Unused Resources

Schedule Lambda or CLI jobs via EventBridge to:

- Delete old CodeBuild logs
- Prune ECR images
- Delete stale EC2 deployments
- Clean up S3 build logs

```
aws ecr batch-delete-image \
  --repository-name myapp \
  --image-ids imageTag=old
```

✅ Summary

Area	Optimization Methods
Compute	Right-size build machines, use spot instances
Triggering	Event-based triggers, branch filters
Testing	Parallel execution, Lambda, cache
Logging	Retention policies, targeted filters
Storage	Compress artifacts, S3 lifecycle rules
Monitoring	Use budgets, alerts, and quotas

📘 Chapter 15: Advanced GitOps and Continuous Delivery Patterns in AWS

🚀 15.1 Introduction

GitOps is the modern approach to Continuous Delivery (CD) where **Git is the single source of truth** for declarative infrastructure and application deployment.

This chapter dives deep into GitOps principles, advanced delivery patterns, and how to implement them using AWS-native services and tools like **EKS**, **CodePipeline**, **Argo CD**, **Flux**, and **CloudFormation/CDK** — all integrated tightly with Git repositories.

🧭 15.2 What is GitOps?

Concept	Description
Git as source of truth	All infrastructure and deployment configurations live in Git repositories
Pull-based deployments	Agents (e.g., ArgoCD) pull latest changes and apply them to clusters
Declarative management	Systems are described using manifests (YAML, HCL, CDK)
Automated reconciliation	Continuous syncing between Git and deployed state

📏 15.3 GitOps vs Traditional CI/CD

Feature	Traditional CD	GitOps
Trigger Model	Push-based (pipeline pushes to infra)	Pull-based (agent pulls from Git)
Source of Truth	Pipelines + config elsewhere	Git repo only
Security Model	CI/CD tools need infra access	Infra agent needs Git access only
Rollback	Manual pipeline re-run	Git revert auto-applies
Observability	Pipeline logs	Git history + sync status

⚙️ 15.4 Tools Used for GitOps in AWS

Tool	Description	Deployment Target
Argo CD	Declarative GitOps controller for Kubernetes	EKS
Flux CD	CNCF GitOps project for K8s	EKS
CDK Pipelines	Git-based CD pipeline using AWS CDK	CloudFormation
CodePipeline + GitHub	Git-triggered AWS pipeline	All AWS services

🌐 15.5 GitOps Workflow in AWS

🔁 1. Developer Workflow

1. Dev pushes Kubernetes manifest to Git
2. GitOps agent (ArgoCD/Flux) detects the change
3. Agent pulls, compares, and reconciles changes
4. Updates EKS environment accordingly

Example GitOps Repo Structure

```
├── apps/
│   └── frontend/
│       └── kustomization.yaml
│       └── deployment.yaml
│       └── service.yaml
├── infra/
│   └── eks/
│       └── eks-cluster.yaml
```

🛠️ 15.6 Implementing GitOps with ArgoCD on EKS

Step 1: Install Argo CD on EKS

```
kubectl create namespace argocd
```

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

Step 2: Login to Argo CD

```
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

Browser: https://localhost:8080
Default login:

```
kubectl -n argocd get secret argocd-initial-admin-secret \
  -o jsonpath="{.data.password}" | base64 -d
```

Step 3: Deploy Application from Git

```
argocd app create my-app \
  --repo https://github.com/myorg/gitops-repo.git \
  --path apps/frontend \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace default
```

Step 4: Sync and Monitor

```
argocd app sync my-app
argocd app get my-app
```

📦 15.7 CDK-Based GitOps (Pull + Push Hybrid)

CDK Pipelines lets you define your entire pipeline in TypeScript/Python. When pushed to GitHub, it auto-triggers changes.

CDK Example:

```
import { CodePipeline, CodePipelineSource, ShellStep } from 'aws-cdk-lib/pipelines';

new CodePipeline(this, 'Pipeline', {
  pipelineName: 'MyCDKPipeline',
  synth: new ShellStep('Synth', {
    input: CodePipelineSource.gitHub('myorg/myrepo', 'main'),
    commands: ['npm ci', 'npm run build', 'npx cdk synth']
  })
});
```

🌈 15.8 Advanced GitOps Patterns

Pattern	Description
Multi-Environment Sync	Separate branches/folders for dev/stage/prod
Automated Drift Detection	Alert if actual infra deviates from Git config
Progressive Delivery	Combine GitOps with canary/blue-green
Promotion via Pull Request	Merge from `dev` to `prod` triggers promotion
Secrets Management	Integrate with SOPS + KMS or AWS Secrets Manager

🔐 Secrets in GitOps (SOPS + KMS)

Encrypt secrets in Git using Mozilla SOPS:

```
sops -e secrets.yaml > secrets.enc.yaml
```

Decrypt automatically in GitOps agent via IAM/KMS access.

🧠 15.9 Real-World GitOps Tips

Tip	Benefit
Use separate repos per team/app	Isolation, cleaner PR process
Use commit signatures and approvals	Change tracking and auditability
Apply retry/backoff in sync policies	Avoid race conditions
Monitor ArgoCD metrics via Prometheus	Health and sync observability
Define App-of-Apps pattern for large systems	Multi-app management from a central app

🧪 15.10 Hands-On: GitOps EKS Deployment with Argo CD

1. Deploy a sample app via `deployment.yaml` in GitHub
2. Point Argo CD to `main` branch of the repo
3. Make a change (e.g., replicas: 2 → 3)
4. Push to Git → ArgoCD detects → EKS updates pods
5. Revert in Git → ArgoCD rolls back the change

💡 **Rollback is just a Git revert!**

✅ 15.11 Summary

Component	GitOps Implementation
EKS	Argo CD, Flux
EC2/CloudFormation	CDK Pipelines, CodePipeline from Git
Secrets	SOPS + KMS/Secrets Manager
Monitoring	ArgoCD + Prometheus + CloudWatch
Rollbacks	Git history → auto reconciliation

📘 Chapter 16: Infrastructure as Code in DevOps – CDK, CloudFormation, Terraform

🧱 16.1 Introduction

Infrastructure as Code (IaC) is the DevOps foundation that enables teams to **define, manage, and provision cloud infrastructure using machine-readable code**. In AWS DevOps, IaC helps to standardize, automate, and version infrastructure using code.

This chapter will guide you through core IaC tools supported in AWS:

- **AWS CloudFormation**
- **AWS CDK (Cloud Development Kit)**
- **Terraform (by HashiCorp)**

We'll cover practical syntax, workflows, CLI tools, and Git-based automation examples.

🧠 16.2 Why IaC in AWS DevOps?

Benefit	Description
Consistency	Environments are repeatable and version-controlled
Automation	No manual configuration — ideal for CI/CD pipelines
Auditable	Git-based code serves as single source of truth
Testable	IaC can be validated using linting and testing tools
Scalable	Easily replicate resources across multiple regions/accounts

🧰 16.3 AWS CloudFormation (YAML/JSON)

CloudFormation is the native AWS IaC tool to manage resources using **YAML or JSON** templates.

✅ Example: Launch EC2 via CloudFormation

```
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t2.micro
      ImageId: ami-0abcd1234efgh5678
```

📦 Deploy using AWS CLI:

```
aws cloudformation create-stack \
  --stack-name my-stack \
  --template-body file://template.yaml
```

📌 CloudFormation Features

- StackSets for multi-account deployments
- Drift Detection
- Change Sets (dry run deployments)
- Parameter Store integration

🖥️ 16.4 AWS CDK (Cloud Development Kit)

CDK allows developers to define AWS resources using **TypeScript, Python, Java, or C#** instead of YAML.

✅ CDK Example in Python

```
from aws_cdk import aws_ec2 as ec2, core

class MyStack(core.Stack):
    def __init__(self, scope, id, **kwargs):
        super().__init__(scope, id, **kwargs)

        ec2.Instance(self, "Instance",
            instance_type=ec2.InstanceType("t2.micro"),
            machine_image=ec2.MachineImage.latest_amazon_linux(),
            vpc=ec2.Vpc.from_lookup(self, "VPC", is_default=True)
        )
```

🔧 CDK CLI Commands

```
cdk init app --language python
cdk synth       # Generates CloudFormation YAML
```

```
cdk deploy       # Deploys the stack
cdk destroy      # Destroys stack
```

🚀 Git Integration for CDK Pipelines

```
from aws_cdk.pipelines import CodePipeline, CodePipelineSource, ShellStep

pipeline = CodePipeline(self, "MyPipeline",
    synth=ShellStep("Synth",
        input=CodePipelineSource.git_hub("myorg/myrepo", "main"),
        commands=["npm ci", "npm run build", "npx cdk synth"]
    )
)
```

🌍 16.5 Terraform with AWS

Terraform is a **cloud-agnostic IaC tool** that integrates with AWS via the AWS provider.

✅ Sample Terraform Code

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "web" {
  ami           = "ami-0abcd1234efgh5678"
  instance_type = "t2.micro"
}
```

🔧 Terraform CLI Workflow

```
terraform init      # Initialize provider plugins
terraform plan      # Preview changes
terraform apply     # Create resources
terraform destroy   # Remove resources
```

🔄 16.6 IaC in DevOps Pipelines

☁️ With CodePipeline + Terraform

```
phases:
  install:
    runtime-versions:
      python: 3.8
  build:
```

```
    commands:
      - terraform init
      - terraform validate
      - terraform plan
      - terraform apply -auto-approve
```

Trigger pipeline on Git push to apply infra changes.

🔐 16.7 Managing Secrets & Variables

Tool	Feature	Example
SSM Parameter Store	Store plain/secure parameters	`aws ssm put-parameter`
Secrets Manager	Encrypt secrets w/ KMS	Use in CloudFormation or Terraform
Environment Variables	Used in CDK or Terraform	`TF_VAR_key=value`, `cdk.context.json`

🧪 16.8 Testing and Validation for IaC

Tool	Use Case
`cfn-lint`	Validate CloudFormation templates
`cdk diff`	Check CDK-generated changes
`tflint`, `tfsec`	Security & format checks in Terraform
`cfripper`	CloudFormation security scanning

🧱 16.9 Modular & Reusable IaC

Terraform Module Structure

```
module "vpc" {
  source  = "./modules/vpc"
  cidr    = "10.0.0.0/16"
}
```

CDK Constructs

```
from constructs import Construct
class MyVpc(Construct):
    ...
```

CloudFormation Nested Stacks

```
Resources:
  VPCStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: "https://s3.amazonaws.com/mybucket/vpc.yaml"
```

🚨 16.10 Drift Detection & Rollbacks

- **CloudFormation Drift Detection**:
- `aws cloudformation detect-stack-drift --stack-name my-stack`
- **Terraform State Drift**:
- `terraform refresh`
- **Rollback in CDK**: Simply revert code and re-deploy.

📊 16.11 Observability for IaC Deployments

Tool	Monitoring
AWS CloudTrail	Logs all API changes
AWS Config	Resource state & compliance logs
CDK Pipelines	Tracks status of IaC deploys
Terraform Cloud	Audit logs + policy enforcement

🧠 16.12 Real-World Tips

- Store all IaC code in **Git with branching**
- Run **automated scans** on PRs before merging
- Use **backend storage for state** (e.g., S3 + DynamoDB for Terraform)
- Always test with **dry-run** (`plan`, `synth`, `diff`)
- Integrate IaC deployments into **CI/CD** pipelines

✅ 16.13 Summary

Tool	Strengths	Best For
CloudFormation	AWS-native, YAML/JSON based	Simple, AWS-only deployments
CDK	Code-first, integrates with Git natively	Dev-friendly IaC
Terraform	Multi-cloud, rich module ecosystem	Complex, cross-platform infra

📘 Chapter 17: Managing Multi-Account & Multi-Region AWS DevOps Setups

🌎 17.1 Introduction

In large-scale AWS DevOps environments, enterprises often need to:

- Use **multiple AWS accounts** (per environment/team)
- Operate across **multiple AWS regions**
- Enforce **centralized security, billing, and governance**
- Enable **cross-account CI/CD** and **resource sharing**

This chapter covers practical strategies, architectures, and tools to **design, deploy, and manage scalable multi-account and multi-region AWS DevOps environments** with consistency and compliance.

🧭 17.2 Why Multi-Account & Multi-Region?

Reason	Multi-Account	Multi-Region
Security Isolation	Isolates workloads	Isolates by geography
Resource Limits	Avoids hitting service limits	Distributes load across regions
Cost Allocation	Better chargeback/showback	Localized billing
Resiliency/Disaster Recovery	Limits blast radius	Ensures geo-failover
Compliance	Follows regulatory rules (e.g., GDPR)	Data residency requirements

🛠️ 17.3 Core Tools for Multi-Account/Region DevOps

Tool/Service	Use Case

AWS Organizations	Account hierarchy and policy control
AWS Control Tower	Landing zone creation, governance
AWS CodePipeline	Cross-account CI/CD orchestration
AWS SSO / IAM Identity Center	Federated access across accounts
CloudFormation StackSets	Cross-account and cross-region IaC
AWS Config / CloudTrail	Centralized compliance and auditing

📐 17.4 Multi-Account Setup Using AWS Organizations

📌 Key Concepts

- **Organization Root**: Parent of all AWS accounts
- **Organizational Units (OUs)**: Group accounts logically (e.g., Prod, Dev, Security)
- **Service Control Policies (SCPs)**: Guardrails that restrict account actions

🧪 Hands-on: Create an Organization

```
aws organizations create-organization --feature-set ALL
aws organizations create-account \
  --email "team1@example.com" \
  --account-name "Dev-Team1-Account"
```

🧱 17.5 Use AWS Control Tower for Governance

AWS Control Tower automates the creation of well-architected multi-account environments.

✅ Features:

- Account vending machine (automated account creation)
- Pre-configured **guardrails** (mandatory & elective)
- Centralized **dashboard** for visibility
- Automates **SSO setup, SCPs, and AWS Config**

🧭 17.6 DevOps Patterns Across Accounts

🏞️ Common Account Structure:

```
Root Account (Master)
├── Security Account (GuardDuty, CloudTrail)
├── Shared Services Account (DNS, AD)
├── Logging Account (S3, CloudWatch)
├── Dev/Stage/Prod Application Accounts
```

📦 Pipeline Strategy:

Component	Pattern
Code Repository	Central GitHub/CodeCommit
CI/CD	CodeBuild in central account
Deployments	Cross-account IAM roles
Artifacts	Centralized S3 or CodeArtifact

👥 Cross-Account IAM Roles

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "AWS":
"arn:aws:iam::<CentralAccountId>:role/CodePipelineRole" },
    "Action": "sts:AssumeRole"
  }]
}
```

🌎 17.7 Multi-Region Design Principles

Principle	Explanation
Data Sovereignty	Keep sensitive data in region
Latency Optimization	Serve users from nearest region
Disaster Recovery (DR)	Use active-passive or active-active
Failover Configuration	Route 53 health checks, multi-region ALB
Deployment	StackSets, replicated pipelines

🔁 StackSets for Multi-Region Deployment

```
aws cloudformation create-stack-set \
  --stack-set-name my-multiregion-stack \
```

```
  --template-body file://infra.yaml \
  --capabilities CAPABILITY_IAM
aws cloudformation create-stack-instances \
  --stack-set-name my-multiregion-stack \
  --regions us-east-1 us-west-2 \
  --accounts 111111111111 222222222222
```

🚀 17.8 CI/CD Pipeline Across Accounts/Regions

Scenario: Central DevOps Account builds → Deploys to App Accounts

Step	Description
1	Developer pushes code to GitHub
2	CodePipeline in DevOps account runs build
3	Build artifact uploaded to S3
4	Pipeline assumes role in target account
5	CodeDeploy/CloudFormation/CDK applies changes in target region/account

🛡️ 17.9 Centralized Logging & Security

- Enable **CloudTrail Organization Trail**:
- `aws cloudtrail create-trail --is-multi-region-trail`
- Use **AWS Config Aggregator** to collect config data from all accounts/regions
- Deploy **Security Hub**, **GuardDuty**, and **Macie** in security account and link members

🔐 17.10 Secure Access Across Accounts

Tool	Functionality
AWS SSO	Unified access using Identity Provider
IAM Role Assumption	Temporary access via trusted role
SCPs	Block risky APIs across accounts

📊 17.11 Monitoring Multi-Account/Region Infra

Tool	Use Case
AWS CloudWatch	Aggregate logs across regions via Log Insights
CloudWatch Dashboards	View global metrics
X-Ray	Trace across services and accounts
CloudTrail	Auditing and access tracing

📦 17.12 Cost Management in Multi-Account Setup

- Enable **Consolidated Billing**
- Use **AWS Budgets** per OU/account
- Integrate **Cost Anomaly Detection**
- Export **Cost Explorer** reports to S3

```
aws ce get-cost-and-usage \
  --time-period Start=2025-06-01,End=2025-06-15 \
  --granularity MONTHLY \
  --metrics "BlendedCost"
```

✅ 17.13 Summary

Practice	Tool/Pattern Used
Multi-Account Governance	AWS Organizations + Control Tower
CI/CD Cross Account	IAM Role Assumption + CodePipeline
Multi-Region IaC	CloudFormation StackSets / Terraform Workspaces
Logging & Security	CloudTrail, Config, GuardDuty
Cost Optimization	Budgets, Billing, Anomaly Detection

📘 Chapter 18: Managing Hybrid and On-Premises Integrations in AWS DevOps

🌉 18.1 Introduction

In real-world enterprise environments, full cloud migration is often **gradual** or **incomplete** due to:

- Legacy systems
- Regulatory constraints
- Latency-sensitive workloads
- Existing on-premises investments

Hybrid cloud DevOps strategies enable seamless integration between **on-premises infrastructure and AWS services**—allowing unified CI/CD pipelines, consistent configuration, and centralized monitoring across environments.

🧭 18.2 Key Use Cases for Hybrid AWS DevOps

Use Case	Description
CI/CD pipelines for hybrid apps	Build in cloud, deploy on-prem or both
Centralized monitoring/logging	Aggregate metrics/logs from all environments
Secrets and credential sharing	Secure secrets across environments using AWS tools
Hybrid networking & identity	Use shared VPN/Direct Connect, LDAP or AD federation

🛠️ 18.3 AWS Services for Hybrid Integration

Service	Purpose

AWS Direct Connect	High-speed private link to AWS
AWS Site-to-Site VPN	Encrypted tunnel over public internet
AWS Outposts	Run AWS infrastructure on-prem
AWS Storage Gateway	Local access to cloud storage
AWS Systems Manager (SSM)	Agent-based ops across cloud & on-prem
AWS CodeDeploy (Hybrid Mode)	App deployment to on-prem EC2/VMs

🧱 18.4 Hybrid Networking – Direct Connect & VPN

🧪 Hands-On: Configure Site-to-Site VPN

1. **Create a Virtual Private Gateway (VGW) on AWS:**
2. `aws ec2 create-vpn-gateway --type ipsec.1`
3. **Create a Customer Gateway (CGW) with on-prem IP:**
4. `aws ec2 create-customer-gateway --type ipsec.1 --public-ip <on-prem-IP>`
5. **Establish the VPN connection:**
6. `aws ec2 create-vpn-connection \`
7. `  --customer-gateway-id cgw-xyz \`
8. `  --vpn-gateway-id vgw-abc \`
9. `  --type ipsec.1`
10. **Download the config and apply it on your on-prem router.**

🧩 18.5 Hybrid CI/CD Pipelines

Scenario: Cloud-based Build + On-Prem Deployment

1. Developers push to GitHub
2. AWS CodePipeline/CodeBuild runs build stage
3. CodeDeploy agent on on-prem VM receives deployment

🧪 CodeDeploy Hybrid Agent Setup (Ubuntu On-Prem)

```
sudo apt update && sudo apt install ruby wget
cd /tmp && wget https://aws-codedeploy-<region>.s3.amazonaws.com/latest/install
chmod +x ./install
sudo ./install auto
```

Register On-Prem Instance:

```
aws deploy register-on-premises-instance \
```

```
  --instance-name MyOnPremInstance \
  --iam-user-arn arn:aws:iam::<account-id>:user/CodeDeployUser
```

Deploy Using CLI:

```
aws deploy create-deployment \
  --application-name MyHybridApp \
  --deployment-group-name MyOnPremDG \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --s3-location bucket=mybucket,key=myapp.zip,bundleType=zip
```

🖥️ 18.6 AWS Systems Manager for Hybrid Fleet

- Install **SSM Agent** on on-prem machines
- Register instance with AWS using hybrid activation
- Execute commands, patch, monitor, or automate ops

🧪 Hybrid Activation:

```
aws ssm create-activation \
  --default-instance-name "OnPremHybrid" \
  --iam-role "SSMServiceRole" \
  --registration-limit 10
```

On-prem:

```
sudo amazon-ssm-agent -register -code "ABC" -id "xyz" -region us-east-1
```

Remotely Run Command:

```
aws ssm send-command \
  --instance-ids "mi-xxxxxxxxxx" \
  --document-name "AWS-RunShellScript" \
  --parameters 'commands=["uptime"]'
```

🧳 18.7 AWS Storage Gateway for Hybrid Storage

Three types of gateways:

Gateway Type	Use Case
File Gateway	NFS/SMB to Amazon S3
Volume Gateway	Block storage with cloud backups

Tape Gateway	Replace physical tape backups with S3 Glacier

🧪 Mount File Gateway:

1. Deploy File Gateway VM on-prem
2. Activate via AWS Console
3. Mount with NFS:

```
4. mount -t nfs <gateway-ip>:/path /mnt/storage
```

🔐 18.8 Identity Federation

Enable users to access AWS using on-prem credentials:

- Use **AWS IAM Identity Center** with Active Directory (AD)
- Federate through **SAML 2.0**

🧪 SAML Setup Overview:

1. Configure AD/LDAP to support SAML
2. Register IdP in AWS IAM Identity Center
3. Assign roles to users/groups
4. Use AWS Console or CLI for access

📈 18.9 Centralized Monitoring in Hybrid Environments

Use **CloudWatch Agent** or **SSM Agent** on-prem:

CloudWatch Agent on On-Prem Server:

```
sudo yum install amazon-cloudwatch-agent
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
sudo systemctl start amazon-cloudwatch-agent
```

You can:

- Collect on-prem logs and metrics
- View them on CloudWatch Dashboard
- Set alarms or integrate with EventBridge

🧪 18.10 GitOps in Hybrid Environments

- Use GitHub/GitLab/Bitbucket as source of truth
- Push infra code that deploys either on-prem or AWS
- Implement agents on-prem that **pull or receive** jobs (e.g., Jenkins agents)

🧠 18.11 Real-World Tips

Tip	Why It Matters
Use SSM and CloudWatch Agents together	Unified management and monitoring
Secure hybrid connections with Direct Connect + VPN	Encrypted and stable networking
Avoid hardcoding IPs; use Route 53 Private Hosted Zones	Easier maintenance and automation
Enforce least privilege IAM across environments	Prevent cross-breach exposure
Implement central logging via Kinesis/Firehose	Visibility across all systems

✅ 18.12 Summary

Category	AWS Tool/Approach
CI/CD	CodePipeline + CodeDeploy hybrid agents
Ops & Monitoring	SSM Agent, CloudWatch Agent
Networking	Direct Connect, VPN, Route 53
Storage	AWS Storage Gateway
Identity	IAM Identity Center, SAML Federation

📘 Chapter 19: DevOps Governance, Compliance & Auditing in AWS

🔐 19.1 Introduction

Governance and compliance are essential pillars in DevOps, especially in regulated industries (e.g., healthcare, finance, public sector). DevOps must ensure that **speed and automation** do not compromise **security, policies, auditability**, and **regulatory compliance**.

This chapter explores how AWS supports DevOps teams in achieving **automated, auditable, and scalable governance** across all resources using native tools and best practices.

🧭 19.2 Key Concepts

Term	Definition
Governance	Enforcing organizational policies, standards, and access controls
Compliance	Adherence to laws, regulations, or internal policies (e.g., GDPR, HIPAA, ISO)
Auditability	Ability to track who did what, when, and where
Policy-as-Code	Managing governance using IaC for consistent enforcement

🧱 19.3 Governance Structure in AWS

Layer	Tool/Service
Account-level	AWS Organizations, SCPs
IAM-level	IAM Policies, Permission Boundaries
Resource-level	Resource Policies, Tag Policies
Pipeline-level	CodePipeline Approval Gates
Monitoring	AWS Config, CloudTrail, AWS Audit Manager

🛠️ 19.4 AWS Tools for DevOps Governance

Service	Purpose
AWS Organizations	Group accounts, apply policies via SCPs
Service Control Policies (SCPs)	Control permissions at OU/account level
AWS Config	Monitor config drift, evaluate compliance
CloudTrail	Record all API activity across accounts
Audit Manager	Automate evidence collection for audits
IAM Access Analyzer	Detect overly broad permissions
Tag Policies	Enforce consistent tagging for resources
CodePipeline Manual Approval	Enforce security review gates

📌 19.5 AWS Config – Configuration Governance

Features:

- Detects non-compliant changes (e.g., open ports, public S3)
- Evaluates resources using AWS-managed or custom rules

🧪 Example: Enable AWS Config via CLI

```
aws configservice put-configuration-recorder \
  --name default \
  --role-arn arn:aws:iam::123456789012:role/aws-config-role \
  --recording-group allSupported=true,includeGlobalResourceTypes=true
```

🧪 Example Rule: Ensure S3 buckets are encrypted

```
aws configservice put-config-rule \
  --config-rule file://s3-encryption-rule.json

s3-encryption-rule.json:

{
  "ConfigRuleName": "s3-bucket-encryption-enabled",
  "Source": {
    "Owner": "AWS",
    "SourceIdentifier": "S3_BUCKET_SERVER_SIDE_ENCRYPTION_ENABLED"
  }
}
```

🔍 19.6 AWS CloudTrail – Auditing Activities

Features:

- Tracks **who did what, when, and from where**
- Stores logs in S3 or sends to CloudWatch
- Works across **all accounts and regions** using **organization trail**

🧪 Create Organization Trail

```
aws cloudtrail create-trail \
  --name org-trail \
  --s3-bucket-name my-cloudtrail-logs \
  --is-multi-region-trail
```

Enable in all accounts:

```
aws cloudtrail update-trail --name org-trail --is-organization-trail
```

📋 19.7 AWS Audit Manager – Compliance Frameworks

Features:

- Automates evidence collection (e.g., user access, encryption, logging)
- Supports prebuilt frameworks (PCI DSS, HIPAA, ISO 27001)
- Integrates with AWS Config, CloudTrail, and IAM

🧪 Create Assessment (Console or CLI):

```
aws auditmanager create-assessment \
  --name "PCI_Assessment" \
  --framework-id <pci-framework-id> \
  --role-arn arn:aws:iam::123456789012:role/AuditManagerRole
```

Export evidence reports as PDFs or CSVs for compliance audits.

🔐 19.8 IAM Governance Features

Feature	Use Case
IAM Access Analyzer	Identify public or cross-account access

IAM Permission Boundaries	Prevent users from exceeding limits
IAM Policy Simulator	Test access before applying policy

🧪 Example: IAM Access Analyzer

```
aws accessanalyzer create-analyzer \
  --analyzer-name MyAnalyzer \
  --type ACCOUNT
```

🏷️ 19.9 Tag-Based Governance & Resource Control

Benefits:

- Enable **cost allocation**, **resource organization**, and **policy enforcement**
- Use **Tag Policies** to enforce tag presence and format

🧪 Example: Tag Policy

```
{
  "tags": {
    "Environment": {
      "tag_key": {
        "enforced_for": ["ec2:instance"]
      },
      "tag_value": {
        "enforced_for": ["ec2:instance"],
        "values": ["dev", "stage", "prod"]
      }
    }
  }
}
```

Apply via AWS Organizations in the management account.

🚦 19.10 Manual Approval Gates in CI/CD

Add **approval steps** in CodePipeline to pause for security/governance reviews.

```
{
  "Name": "ApproveDeploy",
  "Actions": [
    {
      "Name": "ManualApproval",
```

```
      "ActionTypeId": {
        "Category": "Approval",
        "Owner": "AWS",
        "Provider": "Manual",
        "Version": "1"
      },
      "RunOrder": 1
    }
  ]
}
```

🧠 19.11 Policy-as-Code for Governance

Use tools like:

- **CloudFormation Guard (cfn-guard)**
- **Terraform Sentinel**
- **OPA (Open Policy Agent) + Rego**

🧪 Example: `cfn-guard` rule

```
rule check_instance_type {
  resourceType == "AWS::EC2::Instance"
  properties.InstanceType == "t3.micro"
}
```

Run:

```
cfn-guard validate --data cf-template.json --rules rules.guard
```

🧪 19.12 Governance as Code with Terraform Sentinel (Pro feature)

Example Sentinel Policy:

```
import "tfplan"
main = rule { all tfplan.resources.aws_s3_bucket as bucket {
bucket.config.acl is "private" } }
```

📊 19.13 Dashboards and Reporting

Tool	Use

CloudWatch Dashboards	Real-time metrics visualization
AWS Config Aggregator	Multi-account compliance view
AWS Audit Manager Reports	Generate reports for auditors

✅ 19.14 Summary

Goal	AWS Service/Method
Policy Enforcement	SCPs, Tag Policies, Permission Boundaries
Compliance Monitoring	AWS Config, Audit Manager
Auditing & Forensics	CloudTrail, Access Analyzer
CI/CD Security	Manual Approvals, Policy-as-Code
Unified Governance	AWS Organizations + Central Management

📘 Chapter 20: Resilience Engineering and Disaster Recovery with AWS DevOps

🧭 20.1 Introduction

Resilience Engineering in DevOps focuses on **designing systems that gracefully recover from failure**. AWS provides numerous services and architectural patterns to ensure that applications can **withstand disruptions**—whether due to system bugs, hardware failures, or entire region outages.

Disaster Recovery (DR) is a subset of resilience—focused on restoring services and data after major incidents.

This chapter walks through AWS-native ways to build resilient systems and implement disaster recovery strategies within DevOps workflows.

🛠️ 20.2 Core Concepts

Concept	Explanation
Resilience	System's ability to withstand and recover from disruptions
Disaster Recovery (DR)	Strategy to restore services/data after catastrophic failure
High Availability (HA)	Ensure minimal downtime by using redundant infrastructure
Fault Tolerance	System continues to operate even when components fail

🧰 20.3 AWS Services Supporting Resilience

Service	Purpose
Auto Scaling	Replace failed EC2 instances automatically
Elastic Load Balancer (ELB)	Distribute traffic to healthy targets
Amazon Route 53	DNS-level failover routing
AWS Backup	Centralized backup and restore

Amazon RDS Multi-AZ	Automatic failover for databases
S3 Cross-Region Replication (CRR)	Protect data from regional failure
AWS Elastic Disaster Recovery (DRS)	Real-time, low RTO failback
CloudFormation & Terraform	Rebuild infra quickly from IaC

🧪 20.4 Hands-On: High Availability Setup for EC2

Step 1: Launch EC2 in Auto Scaling Group (ASG)

```
aws autoscaling create-auto-scaling-group \
  --auto-scaling-group-name web-asg \
  --launch-template LaunchTemplateId=lt-12345678 \
  --min-size 2 \
  --max-size 4 \
  --desired-capacity 2 \
  --vpc-zone-identifier subnet-1234abcd,subnet-5678efgh
```

Step 2: Attach Load Balancer

```
aws autoscaling attach-load-balancer-target-groups \
  --auto-scaling-group-name web-asg \
  --target-group-arns arn:aws:elasticloadbalancing:...
```

🌎 20.5 Route 53 DNS Failover Example

Failover between two endpoints (primary in `us-east-1`, secondary in `us-west-2`):

1. Configure Health Checks for both endpoints.
2. Create two Route 53 records:
    - Primary: Failover = PRIMARY
    - Secondary: Failover = SECONDARY

🧪 20.6 Resilient Database Setup – RDS Multi-AZ

```
aws rds create-db-instance \
  --db-instance-identifier mydb \
  --engine mysql \
  --multi-az \
  --allocated-storage 20 \
  --db-instance-class db.t3.medium \
```

```
  --master-username admin \
  --master-user-password password
```

Benefits:

- Automatic failover
- No manual intervention
- Same endpoint for app connection

📦 20.7 Backup Strategies

🧪 AWS Backup Configuration

```
aws backup create-backup-plan \
  --backup-plan file://backup-plan.json
```

backup-plan.json:

```
{
  "BackupPlanName": "DailyEC2Backup",
  "Rules": [{
    "RuleName": "DailyBackup",
    "TargetBackupVaultName": "Default",
    "ScheduleExpression": "cron(0 12 * * ? *)",
    "StartWindowMinutes": 60,
    "CompletionWindowMinutes": 180,
    "Lifecycle": {
      "DeleteAfterDays": 30
    }
  }]
}
```

🧪 20.8 S3 Cross-Region Replication

Enable replication from `us-east-1` to `us-west-2`:

1. Enable versioning on both source and destination buckets.
2. Attach IAM role for replication.
3. Apply replication rule:

```
{
  "Rules": [{
    "ID": "CRRRule",
    "Status": "Enabled",
```

```
    "Prefix": "",
    "Destination": {
      "Bucket": "arn:aws:s3:::my-destination-bucket"
    }
  }]
}
```

🔄 20.9 Elastic Disaster Recovery (DRS)

AWS DRS continuously replicates EC2/VMs to AWS for rapid failback.

Features:

- RPO: Seconds
- RTO: Minutes
- Works across AWS and on-prem

Setup Flow:

1. Install DRS agent on source VM.
2. Configure recovery instance template.
3. Launch test or real failover with single click.

💾 20.10 Infrastructure as Code for Resilience

Maintain CloudFormation/Terraform templates to **rebuild your entire infrastructure** in any region during outages.

Benefits:

- Reproducible recovery
- Fast failover/failback
- Lower RTO/RPO

🧪 20.11 Chaos Engineering for Resilience Testing

Tools: **AWS Fault Injection Simulator**

Example Fault Injection

- Simulate EC2 termination
- Introduce network latency
- Validate recovery workflows

```
aws fis create-experiment-template \
  --description "Terminate EC2" \
  --targets file://targets.json \
  --actions file://actions.json \
  --role-arn arn:aws:iam::123456789012:role/FISRole
```

📋 20.12 Disaster Recovery Scenarios & RTO/RPO

Strategy	RTO	RPO	Cost	Example
Backup & Restore	Hours	Hours	Low	S3, RDS backups
Pilot Light	< 1 hr	Minutes	Medium	Replicate DB, keep app idle
Warm Standby	Minutes	Minutes	Higher	Auto-Scaling + replicated DB
Multi-Site Active	Seconds	Seconds	Highest	Active-active regions

🧠 20.13 Real-World Resilience Tips

Tip	Why It Matters
Test DR plan quarterly	Validate your failover capabilities
Use health checks in ELB & Route53	Prevent routing to failed targets
Use lifecycle hooks in ASG	Avoid losing stateful data
Store backups in separate regions	Protect from regional disasters
Use IaC versioning	Ensure consistent infra recovery

✅ 20.14 Summary

Goal	AWS Services/Tools
Auto recovery & healing	ASG, ELB, Route 53
Reliable data backups	AWS Backup, RDS
Disaster Recovery	DRS, S3 CRR
Global Resilience	Multi-region IaC
Resilience testing	AWS Fault Injection Simulator

📘 Chapter 21: DevOps Economics – Cost-Aware Engineering & Optimization

💡 21.1 Introduction

DevOps isn't just about faster deployments or increased agility—**cost awareness** is a critical dimension of software delivery. Engineering teams must understand the **economic impact of their architecture and deployment choices.**

This chapter focuses on **how DevOps can drive cost-efficiency** without sacrificing speed, scalability, or security, using AWS-native tools, IaC, policies, and practices.

📝 21.2 Key Terms

Term	Meaning
FinOps	Financial Operations – bridging finance and engineering to optimize cloud spend
Unit Economics	Cost per unit of value (e.g., per user, per request, per container)
Showback / Chargeback	Allocation of cloud cost to teams based on usage
Cost Anomaly Detection	Alerting when costs deviate from expected patterns
Cost-aware Engineering	Designing systems with cost trade-offs in mind

📊 21.3 AWS Services for Cost Visibility

Service	Purpose
AWS Cost Explorer	View and filter cost breakdowns
AWS Budgets	Set budgets, track and alert overspend
Cost Anomaly Detection	AI-powered abnormal cost detection
AWS CUR (Cost & Usage Reports)	Raw detailed billing data
AWS Pricing Calculator	Estimate costs pre-deployment

AWS Compute Optimizer	Recommends instance rightsizing
AWS Trusted Advisor	Cost, security, fault tolerance checks

💸 21.4 Hands-On: Cost Allocation Using Tags

Step 1: Apply Tags to Resources (CLI)

```
aws ec2 create-tags --resources i-1234567890abcdef0 \
  --tags Key=Environment,Value=Prod Key=Team,Value=DevOps
```

Step 2: Activate Tags for Billing

1. Go to AWS Console → Billing → Cost Allocation Tags
2. Activate `Team`, `Environment`, etc.
3. Use in **Cost Explorer → Group By → Tag**

📈 21.5 Using AWS Budgets

Example: Create Budget via CLI

```
aws budgets create-budget --account-id 123456789012 \
  --budget file://budget.json
```

```
budget.json:
```

```
{
  "BudgetName": "DevOpsEC2Budget",
  "BudgetLimit": {
    "Amount": "100",
    "Unit": "USD"
  },
  "TimeUnit": "MONTHLY",
  "BudgetType": "COST"
}
```

Add notifications to alert via email or SNS.

⚠️ 21.6 Enable Cost Anomaly Detection

```
aws ce create-anomaly-monitor \
  --anomaly-monitor-name "EC2SpikeMonitor" \
  --monitor-type DIMENSIONAL \
  --monitor-dimension SERVICE
```

🧠 21.7 Cost-Aware DevOps Design Principles

Principle	Practice
Use serverless when idle is costly	Lambda, Fargate
Prefer Spot Instances for CI/CD	Save up to 90%
Auto scale everything	Use ASG and Lambda concurrency limits
Optimize container utilization	Bin packing in ECS/EKS
Control data transfer costs	Use CloudFront, avoid cross-AZ data transfer
Rightsize resources continuously	Use Compute Optimizer & CloudWatch

🔧 21.8 Integrating Cost Awareness into CI/CD

Example: Pipeline Step to Check Instance Cost

```
aws pricing get-products \
  --service-code AmazonEC2 \
  --filters Type=TERM_MATCH,Field=instanceType,Value=t3.medium
```

Integrate into Jenkins/GitHub Action to compare price before deploy.

📦 21.9 Unit Cost Analysis in DevOps

Calculate:

- Cost per deploy
- Cost per 1000 requests
- Cost per team/app/account

Example: Divide S3 cost by object count:

```
aws ce get-cost-and-usage \
  --time-period Start=2025-06-01,End=2025-06-30 \
  --granularity MONTHLY \
  --metrics "UsageQuantity" \
  --filter file://filter.json
```

🛡️ 21.10 Enforce Cost Policies via IaC & Governance

Example: SCP to deny large instance types

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "ec2:RunInstances",
    "Resource": "*",
    "Condition": {
      "StringEqualsIfExists": {
        "ec2:InstanceType": ["m5.24xlarge", "r5.24xlarge"]
      }
    }
  }]
}
```

Apply to Dev/Test OUs to avoid misuse.

🚀 21.11 Automation for Optimization

Task	Tool
Idle EC2/DB cleanup	Lambda + Scheduler
Snapshot retention management	AWS Backup or Lifecycle policies
Unused EBS volume detection	Trusted Advisor + CLI
Budget enforcement	Budgets + Lambda + IAM policies

🧰 21.12 DevOps Toolkit for Cost Management

Tool	Purpose
Terraform Cost Estimation	`infracost.io` plugin
Jenkins Budget Check Plugin	Pipeline stage to block based on budget
GitHub Cost Actions	Comment on PR with estimated AWS cost
Cloud Custodian	Policy engine for auto-remediation

✅ 21.13 Summary

Goal	Method or Tool
Budgeting & alerts	AWS Budgets, SNS
Visibility & analysis	Cost Explorer, CUR
Detect anomalies	Cost Anomaly Detection
Automate optimization	Lambda, Compute Optimizer

Showback/chargeback	Cost allocation tags
Cost control in DevOps	SCP, IaC validation, policy enforcement

🔍 Real World Example

Case: CI/CD pipeline on large EC2s was overshooting budget.

- Added `spot` instances to Jenkins workers
- Added step in GitHub Actions to comment cost estimate before merge
- Applied `AWS Budgets` with email alerts for each deploy

Result: Cost dropped by 65% in 2 months

📘 22: DevOps: One-Liner Questions & Answers for Quick Revision

Q1. What does DevOps stand for?
A1. Development and Operations.

Q2. What is the goal of DevOps?
A2. Faster, reliable, and automated software delivery.

Q3. Which AWS service manages source code repositories?
A3. AWS CodeCommit.

Q4. What does CI/CD mean?
A4. Continuous Integration and Continuous Delivery/Deployment.

Q5. What is a buildspec.yml file?
A5. A YAML file defining build instructions for CodeBuild.

Q6. What AWS service compiles and tests code?
A6. AWS CodeBuild.

Q7. Which CLI command triggers a CodeBuild build?
A7. `aws codebuild start-build --project-name <name>`

Q8. What is the primary purpose of CodeDeploy?
A8. Automate deployment of apps to EC2, ECS, or Lambda.

Q9. What file defines CodeDeploy lifecycle hooks?
A9. `appspec.yml`

Q10. Which service manages the CI/CD workflow in AWS?
A10. AWS CodePipeline.

Q11. What is AWS ECR used for?
A11. Private Docker image registry.

Q12. Which command authenticates Docker to ECR?
A12. `aws ecr get-login-password | docker login ...`

Q13. Which AWS services support container orchestration?
A13. ECS and EKS.

Q14. What is ECS Fargate?
A14. Serverless compute for containers.

Q15. What is imagedefinitions.json used for?
A15. It maps containers to images in ECS deployments.

Q16. What is EKS?
A16. Managed Kubernetes service on AWS.

Q17. How do you create an EKS cluster quickly?
A17. `eksctl create cluster`

Q18. Which tool is used to manage Kubernetes resources?
A18. `kubectl`

Q19. Can EKS use Fargate?
A19. Yes, for serverless pod execution.

Q20. What does a service.yaml file define?
A20. Kubernetes service for exposing your app.

Q21. What is Terraform?
A21. Open-source IaC tool using HCL.

Q22. What is AWS CDK?
A22. Code-based IaC using TypeScript, Python, Java, etc.

Q23. What file format does CloudFormation use?
A23. YAML or JSON.

Q24. Which command deploys a CloudFormation stack?
A24. `aws cloudformation deploy --template-file template.yaml`

Q25. What is state management in Terraform?
A25. It tracks infrastructure changes in `.tfstate`.

Q26. What tool is used for code linting in Node.js?
A26. ESLint.

Q27. What is SonarQube used for?
A27. Static code analysis and quality checks.

Q28. What is CodeGuru Reviewer?
A28. AWS ML tool for code quality and security feedback.

Q29. What phase runs tests in `buildspec.yml`?
A29. `pre_build`.

Q30. What tool helps with Python code quality?
A30. Pylint.

Q31. What is DevSecOps?
A31. Integrating security at every step of DevOps.

Q32. How does AWS Secrets Manager help in pipelines?
A32. Securely stores API keys and DB passwords.

Q33. What is Trivy?
A33. Tool for container image and dependency scanning.

Q34. What is Checkov used for?
A34. Static analysis for IaC misconfigurations.

Q35. What AWS service scans container images for CVEs?
A35. Amazon Inspector.

Q36. What is Amazon CloudWatch?
A36. AWS service for metrics, logs, alarms, dashboards.

Q37. What is CloudTrail?
A37. Service to track AWS API activity.

Q38. What does AWS X-Ray do?
A38. Provides distributed tracing for microservices.

Q39. Which metric detects failed deployments in CodeDeploy?
A39. `DeploymentFailure`.

Q40. What does OpenTelemetry offer?
A40. Vendor-neutral observability for metrics and traces.

Q41. What is Amazon SNS?
A41. Notification service for emails, SMS, Lambda triggers.

Q42. What triggers a manual approval in CodePipeline?
A42. An Approval action using an SNS notification.

Q43. How to monitor high EC2 CPU usage?
A43. Use CloudWatch alarm on `CPUUtilization`.

Q44. How can Slack receive AWS alerts?
A44. SNS → Lambda → Slack webhook.

Q45. What is EventBridge used for in monitoring?
A45. To route events for automation and alerting.

Q46. What is Blue/Green deployment?
A46. Shift traffic from old to new environment for zero downtime.

Q47. What is Canary deployment?
A47. Slowly rollout to small traffic, then increase.

Q48. What is Rolling deployment?
A48. Replaces instances in batches.

Q49. What is Linear deployment?
A49. Gradual update at equal intervals.

Q50. Which deployment strategy allows instant rollback?
A50. Blue/Green.

Q51. What is AWS Config?
A51. Monitors resource compliance and drift.

Q52. What are Service Control Policies (SCPs)?
A52. Org-level policies to control permissions in AWS Org.

Q53. What does AWS GuardDuty do?
A53. Detects threats via anomaly detection.

Q54. What is AWS Security Hub?
A54. Aggregates and prioritizes security alerts.

Q55. What is least privilege IAM?
A55. Users get only permissions necessary for their tasks.

Q56. What is the purpose of `build`, `pre_build`, `post_build` in buildspec.yml?
A56. Define command phases in CodeBuild.

Q57. How do you rollback a Lambda deployment?
A57. Re-point the alias to an older version.

Q58. Which service hosts private NPM or Python packages?
A58. AWS CodeArtifact.

Q59. Can we create pipelines using Terraform?
A59. Yes, using `aws_codepipeline` resource.

Q60. What is the advantage of Fargate over EC2?
A60. No server provisioning or scaling needed.

Q61. What is a manual approval action used for?
A61. Pause pipeline for human review.

Q62. How is CDK different from CloudFormation?
A62. CDK uses code (Python, TS) while CloudFormation is YAML/JSON.

Q63. How do you reduce log storage costs?
A63. Set log retention and archive old logs to S3.

Q64. What AWS tool gives unified visibility of cost and usage?
A64. AWS Cost Explorer.

Q65. What is ADOT?
A65. AWS Distro for OpenTelemetry, used for observability.

Q66. Can GitHub be used as a source in CodePipeline?
A66. Yes, via GitHub (v2) source provider or webhook.

Q67. How do you define custom stages in CodePipeline?
A67. By specifying `stages` in the pipeline definition JSON.

Q68. What is the role of build artifacts in pipelines?
A68. They pass outputs (e.g., binaries, images) between pipeline stages.

Q69. What does CodePipeline use to connect services securely?
A69. IAM service roles for each stage.

Q70. Can a pipeline span multiple AWS accounts?
A70. Yes, using cross-account roles and artifacts.

Q71. How can you trigger a build only on a specific branch?
A71. Use filters or specify the branch in source action config.

Q72. How do you securely pass GitHub tokens to AWS?
A72. Store them in Secrets Manager and use environment variables.

Q73. What is the difference between `git push` and `CodePipeline` execution?
A73. `git push` commits code; CodePipeline builds and deploys it.

Q74. Can you auto-trigger pipelines from GitHub commits?
A74. Yes, using GitHub webhooks or CodePipeline integration.

Q75. What does "infrastructure drift" refer to?
A75. When deployed infrastructure differs from the IaC definition.

Q76. What is shift-left testing?
A76. Running tests earlier in the SDLC to catch issues sooner.

Q77. Can integration tests be run in CodeBuild?
A77. Yes, using custom commands in `buildspec.yml`.

Q78. Where are test reports stored in CodeBuild?
A78. In CodeBuild Reports or exported to S3.

Q79. What is coverage testing?
A79. Measures how much of the code is tested.

Q80. How do you include a unit testing step?
A80. Add it to the `pre_build` or `build` phase in `buildspec.yml`.

Q81. Where should DB passwords be stored?
A81. AWS Secrets Manager or Parameter Store.

Q82. What does `env.secrets-manager` in buildspec do?
A82. Injects secrets into the CodeBuild environment.

Q83. What's the benefit of using SSM Parameter Store?
A83. Centralized, encrypted configuration and secrets management.

Q84. Can IAM roles access secrets directly?
A84. Yes, if allowed via Secrets Manager policy.

Q85. What is the risk of hardcoded secrets?
A85. They can be exposed, leading to security breaches.

Q86. What metric namespace is used for EC2 in CloudWatch?
A86. `AWS/EC2`.

Q87. What tool helps visualize all alarms together?
A87. CloudWatch Dashboards.

Q88. How can you trace a user's API actions?
A88. Using CloudTrail event history.

Q89. What is a custom metric?
A89. User-defined metric pushed to CloudWatch.

Q90. What CLI command sets up a CloudWatch alarm?
A90. `aws cloudwatch put-metric-alarm`

Q91. Why is IaC version-controlled?
A91. To allow auditing, rollback, and reproducibility.

Q92. What tool lints CloudFormation templates?
A92. `cfn-lint`.

Q93. What is a Terraform module?
A93. A reusable unit of Terraform configuration.

Q94. How do you prevent state file corruption in Terraform?
A94. Use remote state backends like S3 with locking via DynamoDB.

Q95. What's the command to validate Terraform code?
A95. `terraform validate`.

Q96. What's one way to detect idle resources in AWS?
A96. Use Trusted Advisor or build custom CloudWatch metrics.

Q97. How do tags help in DevOps?
A97. Tags help with cost allocation, automation, and filtering.

Q98. What is the purpose of lifecycle policies in S3?
A98. Automatically transition or delete older objects.

Q99. How can ECR storage costs be reduced?
A99. Enable lifecycle policies to delete untagged or old images.

Q100. What AWS tool can alert on budget thresholds?
A100. AWS Budgets with SNS notifications.

Q101. What is the final trigger for deploying in CodePipeline?
A101. A successful build artifact and optional approval.

Q102. Can Lambda be used in pipelines?
A102. Yes, as deployment targets or approval logic.

Q103. What AWS service lets you run pre-deployment validation?
A103. CodeDeploy with lifecycle hooks (e.g., BeforeInstall).

Q104. Can you visualize deployment history in AWS?
A104. Yes, in CodeDeploy or CodePipeline consoles.

Q105. What service helps define governance rules as code?
A105. AWS Config.

📘 Chapter 23: References

The following references and official documentation have been used to ensure accuracy, reliability, and technical completeness of the concepts, commands, and infrastructure-as-code examples included in this book. All links are valid as of the time of writing.

🔹 AWS Official Documentation

- **AWS DevOps Services Overview**
  https://aws.amazon.com/devops
- **AWS CodePipeline User Guide**
  https://docs.aws.amazon.com/codepipeline
- **AWS CodeBuild User Guide**
  https://docs.aws.amazon.com/codebuild
- **AWS CodeDeploy Documentation**
  https://docs.aws.amazon.com/codedeploy
- **AWS CloudFormation User Guide**
  https://docs.aws.amazon.com/cloudformation
- **AWS CDK (Cloud Development Kit) Docs**
  https://docs.aws.amazon.com/cdk
- **AWS Lambda Developer Guide**
  https://docs.aws.amazon.com/lambda
- **Amazon ECS Developer Guide**
  https://docs.aws.amazon.com/ecs
- **Amazon EKS Documentation**
  https://docs.aws.amazon.com/eks
- **AWS CloudWatch Logs and Metrics**
  https://docs.aws.amazon.com/cloudwatch
- **AWS Secrets Manager Guide**
  https://docs.aws.amazon.com/secretsmanager
- **AWS IAM Best Practices**
  https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html
- **Amazon Inspector and ECR Scanning**
  https://docs.aws.amazon.com/inspector/latest/user/inspector_ecr

🔹 Terraform Registry & IaC References

- **Terraform AWS Provider Documentation**
  https://registry.terraform.io/providers/hashicorp/aws/latest/docs
- **Terraform EKS Module**
  https://github.com/terraform-aws-modules/terraform-aws-eks

- **Terraform CodePipeline Examples**
  https://registry.terraform.io/modules/terraform-aws-modules/codepipeline/aws/latest

CDK Examples & Open Source Repositories

- **AWS CDK Examples (GitHub)**
  https://github.com/aws-samples/aws-cdk-examples
- **CDK Pipelines Module**
  https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines-readme.html
- **CDK Construct Hub**
  https://constructs.dev

Open Source Tools Used for Security & Quality

- **Trivy – Aqua Security**
  https://github.com/aquasecurity/trivy
- **Checkov – Infrastructure Scanning**
  https://github.com/bridgecrewio/checkov
- **SonarQube – Code Quality**
  https://www.sonarsource.com/products/sonarqube
- **ESLint – JavaScript Linting**
  https://eslint.org
- **Pylint – Python Linter**
  https://pylint.pycqa.org

Additional Industry-Recognized Learning Resources

- **AWS Well-Architected Framework**
  https://aws.amazon.com/architecture/well-architected
- **AWS DevOps Blog**
  https://aws.amazon.com/blogs/devops
- **AWS Workshop Portal**
  https://workshops.aws

📘 Chapter 24: Mini-Projects for AWS DevOps Hands-On Practice

These mini-projects are designed to give you practical, step-by-step experience using core AWS DevOps tools, services, and strategies. Each project includes objective, architecture, instructions, CLI, and IaC commands wherever applicable.

🔹 Project 1: CI/CD Pipeline for Static Website Hosting on S3

💼 Objective:

Automate the deployment of a static website to an S3 bucket using CodePipeline and CodeBuild.

🌐 Tools Used:

- S3, CodeCommit, CodeBuild, CodePipeline, CloudFront (optional)

✅ Instructions:

1. **Create S3 Bucket:**

```
aws s3api create-bucket --bucket my-static-site-bucket --region us-east-1
```

2. **Enable Static Website Hosting:**

```
aws s3 website s3://my-static-site-bucket/ --index-document index.html
```

3. **Create CodeCommit Repo:**

```
aws codecommit create-repository --repository-name StaticSiteRepo
```

4. **Upload Site Code (HTML/JS):**

```
git clone https://git-codecommit.us-east-
1.amazonaws.com/v1/repos/StaticSiteRepo
cd StaticSiteRepo
echo "<html>Welcome</html>" > index.html
git add . && git commit -m "initial commit" && git push
```

5. **Create buildspec.yml:**

```
version: 0.2
phases:
  build:
    commands:
      - echo Building static website
artifacts:
  files:
    - '**/*'
```

6. **Create CodeBuild Project:**
   Configure to build from CodeCommit and output to S3.
7. **Create CodePipeline:**
   - Source: CodeCommit
   - Build: CodeBuild
   - Deploy: S3 bucket
8. **(Optional)** Setup CloudFront:

```
aws cloudfront create-distribution --origin-domain-name my-static-site-bucket.s3.amazonaws.com
```

🔹 Project 2: Dockerized Node.js App CI/CD to ECS

💼 Objective:

Build a Docker image, push to ECR, and deploy to ECS via CodePipeline.

🌐 Tools:

- CodeCommit, CodeBuild, CodePipeline, ECR, ECS Fargate

✅ Instructions:

1. **Dockerfile (in repo):**

```
FROM node:18
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "app.js"]
```

2. **ECR Repo:**

```
aws ecr create-repository --repository-name node-app
```

3. **Push Code:**

```
git commit -am "Docker app" && git push
```

4. **buildspec.yml:**

```
version: 0.2
env:
  variables:
    IMAGE_REPO_NAME: "node-app"
    IMAGE_TAG: latest
phases:
  pre_build:
    commands:
      - $(aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin <account>.dkr.ecr.us-east-1.amazonaws.com)
  build:
    commands:
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG <account>.dkr.ecr.us-east-
1.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - docker push <account>.dkr.ecr.us-east-
1.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG
      - echo '[{"name":"node-container","imageUri":"<account>.dkr.ecr.us-
east-1.amazonaws.com/node-app:latest"}]' > imagedefinitions.json
artifacts:
  files:
    - imagedefinitions.json
```

5. **ECS Task Definition & Fargate Service:** Create in ECS console.
6. **CodePipeline:** Source (CodeCommit) → Build (CodeBuild) → Deploy (ECS).

🔹 Project 3: Infrastructure as Code Using Terraform

💼 Objective:

Provision S3, EC2, and IAM with Terraform.

✅ Instructions:

1. **Directory Structure:**

```
terraform_project/
├── main.tf
├── variables.tf
```

```
├── outputs.tf
```

2. **main.tf:**

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_s3_bucket" "mybucket" {
  bucket = "my-terraform-bucket-123"
}

resource "aws_instance" "myec2" {
  ami           = "ami-0c02fb55956c7d316"
  instance_type = "t2.micro"
}
```

3. **Deploy:**

```
terraform init
terraform plan
terraform apply
```

4. **Add outputs & variables for dynamic config.**

🔹 Project 4: Lambda Blue/Green Deployment with Traffic Shifting

💼 Objective:

Deploy Lambda with versioning and traffic shifting using CodeDeploy.

✅ Instructions:

1. **Lambda Function with Alias:**

```
aws lambda publish-version --function-name my-func
aws lambda create-alias --function-name my-func --name live --function-
version 1
```

2. **appspec.yml:**

```
version: 0.0
Resources:
  - myLambda:
      Type: AWS::Lambda::Function
```

```
      Properties:
        Name: my-func
        Alias: live
        CurrentVersion: 1
        TargetVersion: 2
```

3. **deployment-config.json:**

```
{
  "deploymentConfigName": "CodeDeployDefault.LambdaCanary10Percent5Minutes"
}
```

4. **Create Deployment App:**

```
aws deploy create-application --application-name my-lambda-app --compute-platform Lambda
```

5. **Trigger via CodePipeline** after CodeBuild builds the zip.

🔹 Project 5: GitHub CI/CD for Helm App on EKS

💼 Objective:

Push Docker image, package Helm chart, and deploy to EKS via GitHub Actions.

✅ Instructions:

1. **Create EKS Cluster:**

```
eksctl create cluster --name helm-cluster --region us-east-1
```

2. **Dockerfile + Helm Chart in Repo:**

```
charts/myapp/
  ├── Chart.yaml
  ├── values.yaml
  └── templates/deployment.yaml
```

3. **GitHub Action Workflow:** `.github/workflows/deploy.yaml`

```
name: Deploy to EKS
on:
  push:
    branches:
      - main
jobs:
```

```
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-1
      - name: Deploy with Helm
        run: |
          helm upgrade --install myapp ./charts/myapp --namespace default
```

4. **Access App via LoadBalancer Service.**

🔹 Project 6: Multi-Account AWS Control Tower Setup

💼 Objective:

Set up secure, compliant multi-account AWS environment.

✅ Instructions:

1. **Enable AWS Control Tower in Management Account.**
2. **Create Organizational Units (OUs)** like `Security`, `Sandbox`, `Workload`.
3. **Create AWS Accounts** via Account Factory.
4. **Apply SCPs:**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "ec2:*",
      "Resource": "*",
      "Condition": { "StringNotEquals": { "aws:RequestedRegion": "us-east-
1" } }
    }
  ]
}
```

5. **Enable AWS Config + CloudTrail** in all accounts.
6. **Centralize logs and billing.**

📘 Glossary of Terms

🔤 A

- **ADOT (AWS Distro for OpenTelemetry)**: A secure, production-ready distribution of the OpenTelemetry project for AWS observability.
- **Alias (Lambda)**: A pointer to a specific version of a Lambda function, enabling versioned deployment strategies.
- **API Gateway**: AWS service that enables creation and monitoring of APIs.
- **AppSpec File**: YAML or JSON file used by CodeDeploy to define deployment actions.

🔤 B

- **Blue/Green Deployment**: A deployment strategy with two environments—one live (green), one idle (blue)—used for zero-downtime deployment.
- **BuildSpec File**: A YAML configuration file that defines the build commands and settings for AWS CodeBuild.

🔤 C

- **Canary Deployment**: Gradual release of a new version to a subset of users to minimize impact from failures.
- **CI/CD (Continuous Integration/Continuous Deployment)**: Automating the integration and delivery of code changes through pipelines.
- **CloudFormation**: AWS service that lets you model and provision AWS resources using YAML or JSON templates.
- **CloudTrail**: AWS service that logs API calls and activity across your AWS environment.
- **CloudWatch**: AWS monitoring and observability service for metrics, logs, alarms, and dashboards.
- **CodeArtifact**: AWS artifact repository for packages like npm, pip, Maven.
- **CodeBuild**: AWS service for compiling source code, running tests, and producing build artifacts.
- **CodeCommit**: AWS Git-based source control system.
- **CodeDeploy**: AWS service for automating application deployment to EC2, Lambda, or ECS.
- **CodeGuru**: ML-based AWS tool that reviews code for security and performance issues.
- **CodePipeline**: Orchestration service for automating CI/CD workflows in AWS.
- **Config (AWS Config)**: A service that enables assessment and auditing of resource configurations.

- **Container**: A lightweight, standalone, executable software package that includes everything needed to run a piece of software.
- **Custom Metrics**: User-defined metrics published to CloudWatch for observability.

D

- **DevOps**: A culture and set of practices that integrate software development (Dev) and IT operations (Ops).
- **DevSecOps**: Integration of security practices into the DevOps lifecycle.
- **Docker**: Platform to build, deploy, and manage containerized applications.

E

- **EBS (Elastic Block Store)**: AWS block storage service for EC2 instances.
- **EC2 (Elastic Compute Cloud)**: AWS service for virtual machines.
- **ECR (Elastic Container Registry)**: Managed Docker container registry by AWS.
- **ECS (Elastic Container Service)**: AWS native container orchestration service.
- **EKS (Elastic Kubernetes Service)**: AWS managed Kubernetes service.
- **Environment Variables**: Key-value pairs used for configuration in code or build environments.

G

- **Git**: Distributed version control system for source code.
- **GitHub Actions**: CI/CD service from GitHub for automating workflows.
- **GitOps**: Infrastructure and deployment automation based on Git repositories.
- **GuardDuty**: Threat detection service that uses machine learning and threat intelligence.

H

- **Helm**: Kubernetes package manager for deploying charts (pre-configured apps).

I

- **IaC (Infrastructure as Code)**: Managing and provisioning infrastructure using machine-readable configuration files.
- **IAM (Identity and Access Management)**: AWS service for managing user access and permissions.
- **ImageDefinitions.json**: File used by CodeDeploy to map Docker containers to image URIs for ECS deployments.
- **Inspector (Amazon Inspector)**: AWS vulnerability scanning service for EC2, Lambda, and ECR.

J

- **Jenkins**: Popular open-source automation server for building CI/CD pipelines.

K

- **KMS (Key Management Service)**: AWS service for managing encryption keys.
- **Kubernetes**: Open-source container orchestration system for automating deployment, scaling, and management.

L

- **Lambda (AWS Lambda)**: Serverless compute service that runs code in response to events.
- **Logs**: Output generated by applications, builds, and services used for debugging and monitoring.

M

- **Metrics**: Quantitative measures (e.g., CPU usage, memory) tracked over time for monitoring system health.
- **Monitoring**: Observing the state and performance of systems and applications.

N

- **Node.js**: JavaScript runtime built on Chrome's V8 engine, often used in DevOps pipelines.

O

- **OpenTelemetry**: A set of APIs, libraries, and agents to collect distributed traces and metrics.
- **Observability**: Measure of how well internal states of a system can be inferred from outputs (metrics, logs, traces).

P

- **Pipeline**: An automated workflow that builds, tests, and deploys code changes.
- **Pre-Build Phase**: Step in a buildspec where setup tasks like installing dependencies are done.
- **Prometheus**: Monitoring system and time series database used for Kubernetes and cloud-native applications.

R

- **Repository**: A version-controlled collection of code, often Git-based.
- **Rollout Strategy**: Approach to releasing new code (e.g., blue/green, canary, rolling).
- **Rollback**: Reverting to a previous version after a failed deployment.

S

- **S3 (Simple Storage Service)**: AWS object storage used for logs, artifacts, templates.
- **Secrets Manager**: AWS service to store and manage access to secrets like API keys and passwords.
- **Security Hub**: Central dashboard to view and manage security findings from AWS services.
- **Service Control Policies (SCPs)**: Organization-wide policies to manage account permissions.
- **Shift Left**: Practice of integrating testing/security early in the software development process.
- **SonarQube**: Tool for static code analysis and quality checks.
- **Stack**: A collection of AWS resources defined and managed as a single unit in CloudFormation.
- **SNS (Simple Notification Service)**: Messaging service for alerts and notifications.
- **Source Stage**: Initial stage in a CI/CD pipeline that retrieves code from source control.

T

- **Terraform**: Open-source IaC tool for provisioning cloud infrastructure using HCL.
- **Trace**: A record of request flow through distributed systems used for debugging and performance tuning.
- **Trivy**: Open-source vulnerability scanner for containers, IaC, and dependencies.
- **Trigger**: An event that initiates a pipeline or build process.

U

- **Unit Test**: Automated test that verifies the smallest piece of code behaves as expected.
- **User Data**: Script or commands passed to an EC2 instance at launch for initialization.

V

- **VPC (Virtual Private Cloud)**: AWS's isolated network environment for launching resources.

- **Versioning (Lambda)**: Ability to publish and manage different versions of a Lambda function.

W

- **Webhook**: A mechanism to send real-time updates to external systems (e.g., GitHub to trigger CodePipeline).

X

- **X-Ray (AWS X-Ray)**: AWS service for distributed tracing of microservices to diagnose errors and latency.