AWS DevOps Digital Guide

Author: Sainath Shivaji Mitalakar *Edition: 2025*

1 Introduction to AWS and DevOps

Welcome to the **AWS DevOps Digital Guide**, designed for engineers, architects, and IT professionals. This book will help you master AWS services, DevOps principles, automation, and cost optimization through practical examples and clear explanations.

1.1 What is AWS DevOps?

AWS DevOps is the practice of implementing DevOps principles on Amazon Web Services. It allows automation of deployments, efficient monitoring, and cost optimization.

1.2 Key Benefits

- **Automation:** Infrastructure as Code, CI/CD pipelines
- **Scalability:** Horizontal & vertical scaling
- **Security:** IAM, KMS, MFA
- **Cost Optimization:** Budgets, Reserved/Spot instances, S3 lifecycle
- **Monitoring:** CloudWatch, CloudTrail, SNS notifications

1.3 Core AWS DevOps Services

- **Compute:** EC2, Lambda, ECS, EKS
- **Storage:** S3, EBS, EFS
- **Networking:** VPC, Route53, ELB
- **CI/CD:** CodePipeline, CodeBuild, CodeDeploy
- **Monitoring:** CloudWatch, CloudTrail

1.4 AWS CLI Commands Example

Configure AWS CLI:

```
$ aws configure
AWS Access Key ID [None]: AKIAEXAMPLE
AWS Secret Access Key [None]: abc123EXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]: json
```

Output:

```
Configuration saved successfully.
```

List S3 Buckets:

```
$ aws s3 ls
```

Output:

```
2025-09-01 00:00:00 my-first-bucket
2025-09-01 01:20:10 my-second-bucket
```

Tip: Use –query and –output table to make CLI outputs more readable. Example:

```
$ aws ec2 describe-instances --query "Reservations[*].Instances[*].[InstanceId,State
```

2 IAM and Security

IAM (Identity and Access Management) controls access to AWS services securely. It allows you to create users, groups, roles, and policies for fine-grained permission management.

2.1 Core Concepts

- **Users:** Individual accounts for human users or services.
- **Groups:** Collection of users with shared permissions.
- **Roles:** Temporary credentials for applications, services, or cross-account access.
- **Policies:** JSON documents defining permissions.

2.2 Creating IAM Users and Groups

Create a new IAM user:

```
$ aws iam create-user --user-name DevOpsUser
```

Output:

```
{
    "User": {
        "UserName": "DevOpsUser",
        "UserId": "AIDAEXAMPLEID",
        "Arn": "arn:aws:iam::123456:user/DevOpsUser",
        "CreateDate": "2025-09-01T10:00:00Z"
    }
}
```

Create a group:

```
$ aws iam create-group --group-name DevOpsGroup
```

Output:

```
{
    "Group": {
        "GroupName": "DevOpsGroup",
        "GroupId": "AGPAEXAMPLEID",
        "Arn": "arn:aws:iam::123456:group/DevOpsGroup",
        "CreateDate": "2025-09-01T10:05:00Z"
    }
}
```

2.3 Attaching Policies to Users or Groups

Attach Administrator Access policy to user:

```
$ aws iam attach-user-policy --user-name DevOpsUser --policy-arn arn:aws:iam::aws:po
```

Output:

```
Successfully attached policy AdministratorAccess to user DevOpsUser
```

2.4 Creating Roles for EC2

Create a role for EC2:

```
$ aws iam create-role --role-name EC2Role --assume-role-policy-document file://trust
```

Output:

```
{
    "Role": {
        "RoleName": "EC2Role",
        "RoleId": "AROAEXAMPLEID",
        "Arn": "arn:aws:iam::123456:role/EC2Role",
        "CreateDate": "2025-09-01T10:10:00Z"
    }
}
```

2.5 Security Best Practices

- Follow the **least privilege principle**.
- Enable **MFA** for all users with sensitive access.
- Rotate access keys regularly.
- Monitor and review **CloudTrail logs** frequently.

3 EC2 Instances, Key Pairs, and Security Groups

3.1 Introduction to EC2

Amazon EC2 (Elastic Compute Cloud) provides scalable virtual servers. Key concepts:

- **Instance Types:** t2.micro, m5.large, etc.
- **AMI (Amazon Machine Image):** Pre-configured OS and software.
- **Key Pairs:** Secure SSH login credentials.
- **Security Groups:** Virtual firewall for instances.

3.2 Launching an EC2 Instance

Launch a new EC2 instance using AWS CLI:

```
$ aws ec2 run-instances
  --image-id ami-0abcdef1234567890
  --count 1
  --instance-type t2.micro
  --key-name DevOpsKey
  --security-group-ids sg-0abc1234def567890
  --subnet-id subnet-0ab12c345def67890
```

Output:

```
{
    "Instances": [
        {
            "InstanceId": "i-0abcdef1234567890",
            "ImageId": "ami-0abcdef1234567890",
            "State": {"Code": 0, "Name": "pending"},
            "InstanceType": "t2.micro",
            "KeyName": "DevOpsKey",
            "SubnetId": "subnet-0ab12c345def67890",
            "SecurityGroups": [{"GroupName": "DevOpsSG","GroupId": "sg-0abc1234def56
        }
    ]
}
```

3.3 Creating Key Pairs

Create a new key pair for SSH access:

```
$ aws ec2 create-key-pair --key-name DevOpsKey
$ aws ec2 create-key-pair --key-name DevOpsKey --query KeyMaterial --output text > D
```

Output:

```
<PrivateKey data saved in DevOpsKey.pem>
```

Tip: Always set correct permissions for the key file before using SSH:

```
$ chmod 400 DevOpsKey.pem
```

3.4 Creating Security Groups

Create a security group:

```
$ aws ec2 create-security-group
  --group-name DevOpsSG
  --description "DevOps EC2 Security Group"
  --vpc-id vpc-0abcd1234efgh5678
```

Output:

```
{
    "GroupId": "sg-0abc1234def567890"
}
```

3.5 Adding Inbound Rules

Allow SSH (port 22) access:

```
$ aws ec2 authorize-security-group-ingress
  --group-id sg-0abc1234def567890
  --protocol tcp
  --port 22
  --cidr 0.0.0.0/0
```

Output:

```
{
    "Return": true
}
```

Allow HTTP (port 80) access:

```
$ aws ec2 authorize-security-group-ingress
  --group-id sg-0abc1234def567890
  --protocol tcp
  --port 80
  --cidr 0.0.0.0/0
```

Output:

```
{
    "Return": true
}
```

Tip: Limit SSH access to your IP for security instead of 0.0.0.0/0.

4 EBS Volumes, Snapshots, and AMI Management

4.1 Introduction to EBS

Amazon Elastic Block Store (EBS) provides persistent block storage for EC2 instances. Key concepts:

- **Volume Types:** General Purpose SSD (gp3), Provisioned IOPS SSD (io2), Magnetic.
- **Snapshots:** Point-in-time backups of volumes.
- **Attachment:** Volumes must be attached to EC2 instances.

4.2 Creating an EBS Volume

Create a new 8GB EBS volume:

```
$ aws ec2 create-volume
  --size 8
  --region us-east-1
  --availability-zone us-east-1a
  --volume-type gp3
```

Output:

```
{
    "AvailabilityZone": "us-east-1a",
    "VolumeId": "vol-0abcd1234ef567890",
    "Size": 8,
    "State": "creating",
    "VolumeType": "gp3"
}
```

4.3 Attaching an EBS Volume to EC2

Attach the volume to an EC2 instance:

```
$ aws ec2 attach-volume
  --volume-id vol-0abcd1234ef567890
  --instance-id i-0abcdef1234567890
  --device /dev/sdf
```

Output:

```
{
    "State": "attaching",
    "AttachTime": "2025-09-01T12:00:00Z",
    "InstanceId": "i-0abcdef1234567890",
    "VolumeId": "vol-0abcd1234ef567890",
    "Device": "/dev/sdf"
}
```

4.4 Creating a Snapshot of EBS Volume

Create a snapshot for backup:

```
$ aws ec2 create-snapshot
  --volume-id vol-0abcd1234ef567890
  --description "Backup of DevOps EBS volume"
```

Output:

```
{
    "SnapshotId": "snap-0abc1234def567890",
    "State": "pending",
    "VolumeId": "vol-0abcd1234ef567890",
    "StartTime": "2025-09-01T12:30:00Z",
    "Description": "Backup of DevOps EBS volume"
}
```

4.5 Creating an AMI from EC2 Instance

Create an Amazon Machine Image (AMI):

```
$ aws ec2 create-image
  --instance-id i-0abcdef1234567890
  --name "DevOpsServerAMI"
  --description "AMI for DevOps EC2 server"
```

Output:

```
{
    "ImageId": "ami-0abc1234def567890"
}
```

4.6 Listing Snapshots and AMIs

List all snapshots in your account:

```
$ aws ec2 describe-snapshots --owner-ids self
```

Output:

```
[
    {
        "SnapshotId": "snap-0abc1234def567890",
        "VolumeId": "vol-0abcd1234ef567890",
        "State": "completed",
        "StartTime": "2025-09-01T12:30:00Z"
    }
]
```

List all AMIs you own:

```
$ aws ec2 describe-images --owners self
```

Output:

```
{
    "Images": [
        {
            "ImageId": "ami-0abc1234def567890",
            "Name": "DevOpsServerAMI",
            "State": "available",
            "CreationDate": "2025-09-01T12:45:00Z"
        }
    ]
}
```

Tip: Use snapshots to recover from accidental data loss and to create new AMIs quickly.

5 VPC, Subnets, and Internet Gateway Setup

5.1 Introduction to VPC

A Virtual Private Cloud (VPC) allows you to provision a logically isolated section of AWS Cloud. Key concepts:

- **VPC:** Virtual network with CIDR block.
- **Subnet:** Subdivision of VPC to group resources.

- **Internet Gateway (IGW):** Enables internet connectivity for VPC.
- **Route Table:** Controls traffic routing in the VPC.

5.2 Creating a VPC

Create a new VPC:

```
$ aws ec2 create-vpc
  --cidr-block 10.0.0.0/16
```

Output:

```
{
    "Vpc": {
        "VpcId": "vpc-0abc1234def567890",
        "State": "pending",
        "CidrBlock": "10.0.0.0/16",
        "IsDefault": false
    }
}
```

5.3 Creating Subnets

Create a public subnet:

```
$ aws ec2 create-subnet
  --vpc-id vpc-0abc1234def567890
  --cidr-block 10.0.1.0/24
  --availability-zone us-east-1a
```

Output:

```
{
    "Subnet": {
        "SubnetId": "subnet-0abcd1234ef567890",
        "VpcId": "vpc-0abc1234def567890",
        "CidrBlock": "10.0.1.0/24",
        "AvailabilityZone": "us-east-1a",
        "State": "available"
    }
}
```

5.4 Creating an Internet Gateway (IGW)

Create an Internet Gateway:

```
$ aws ec2 create-internet-gateway
```

Output:

```
{
    "InternetGateway": {
        "InternetGatewayId": "igw-0abc1234def567890",
        "Attachments": []
    }
}
```

5.5 Attaching IGW to VPC

Attach Internet Gateway to the VPC:

```
$ aws ec2 attach-internet-gateway
  --vpc-id vpc-0abc1234def567890
  --internet-gateway-id igw-0abc1234def567890
```

Output:

```
{
    "Return": true
}
```

5.6 Creating a Route Table and Route to IGW

Create a route table:

```
$ aws ec2 create-route-table
  --vpc-id vpc-0abc1234def567890
```

Output:

```
{
    "RouteTable": {
        "RouteTableId": "rtb-0abc1234def567890",
        "VpcId": "vpc-0abc1234def567890",
        "Routes": [
            {"DestinationCidrBlock": "10.0.0.0/16", "GatewayId": "local"}
        ]
    }
}
```

Add route to IGW:

```
$ aws ec2 create-route
  --route-table-id rtb-0abc1234def567890
  --destination-cidr-block 0.0.0.0/0
  --gateway-id igw-0abc1234def567890
```

Output:

```
{
    "Return": true
}
```

5.7 Associating Subnet with Route Table

Associate subnet with route table:

```
$ aws ec2 associate-route-table
  --subnet-id subnet-0abcd1234ef567890
  --route-table-id rtb-0abc1234def567890
```

Output:

```
{
    "AssociationId": "rtbassoc-0abc1234def567890"
}
```

Tip: Always check subnet's route table to ensure proper internet connectivity for public resources.

6 NAT Gateway, Private Subnets, and Elastic IPs

6.1 Introduction

In a VPC setup, NAT Gateways allow instances in private subnets to access the internet for updates or downloads without exposing them publicly. Key concepts:

- **Private Subnet:** Subnet without direct internet access.
- **NAT Gateway:** Enables internet access for private instances.
- **Elastic IP (EIP):** Static public IP to associate with NAT or EC2.

6.2 Allocate an Elastic IP

Allocate a new Elastic IP:

```
$ aws ec2 allocate-address --domain vpc
```

Output:

```
{
    "PublicIp": "203.0.113.25",
    "AllocationId": "eipalloc-0abc1234def567890"
}
```

6.3 Create a NAT Gateway

Create NAT Gateway in a public subnet:

```
$ aws ec2 create-nat-gateway
  --subnet-id subnet-0abcd1234ef567890
  --allocation-id eipalloc-0abc1234def567890
```

Output:

```
{
    "NatGateway": {
        "NatGatewayId": "nat-0abc1234def567890",
        "State": "pending",
        "SubnetId": "subnet-0abcd1234ef567890",
        "VpcId": "vpc-0abc1234def567890",
        "NatGatewayAddresses": [
            {"PublicIp": "203.0.113.25", "AllocationId": "eipalloc-0abc1234def56789
        ]
    }
}
```

6.4 Create Private Subnet

Create a private subnet:

```
$ aws ec2 create-subnet
  --vpc-id vpc-0abc1234def567890
  --cidr-block 10.0.2.0/24
  --availability-zone us-east-1a
```

Output:

```
{
    "Subnet": {
        "SubnetId": "subnet-0abcd5678efgh1234",
        "VpcId": "vpc-0abc1234def567890",
        "CidrBlock": "10.0.2.0/24",
        "AvailabilityZone": "us-east-1a",
        "State": "available"
    }
}
```

6.5 Create a Route Table for Private Subnet

Create a route table:

```
$ aws ec2 create-route-table
  --vpc-id vpc-0abc1234def567890
```

Output:

```
{
    "RouteTable": {
        "RouteTableId": "rtb-0abcd5678efgh1234",
        "VpcId": "vpc-0abc1234def567890"
    }
}
```

6.6 Add Route to NAT Gateway

Add route for private subnet to NAT Gateway:

```
$ aws ec2 create-route
  --route-table-id rtb-0abcd5678efgh1234
  --destination-cidr-block 0.0.0.0/0
  --nat-gateway-id nat-0abc1234def567890
```

Output:

```
{
    "Return": true
}
```

6.7 Associate Private Subnet with Route Table

Associate private subnet with the route table:

```
$ aws ec2 associate-route-table
  --subnet-id subnet-0abcd5678efgh1234
  --route-table-id rtb-0abcd5678efgh1234
```

Output:

```
{
    "AssociationId": "rtbassoc-0abcd5678efgh1234"
}
```

Tip: NAT Gateways are billed hourly. Use them only when necessary for cost optimization.

7 Security Groups, NACLs, and Bastion Hosts

7.1 Introduction

Securing your AWS environment is crucial. Security Groups (SG) and Network ACLs (NACLs) help control traffic. Bastion Hosts allow secure SSH access to instances in private subnets. Key concepts:

- **Security Groups:** Stateful firewall for EC2 instances.
- **Network ACLs (NACLs):** Stateless subnet-level firewall.
- **Bastion Host:** Jump server for accessing private instances.

7.2 Security Group Example

Create a security group for private instances:

```
$ aws ec2 create-security-group
  --group-name PrivateSG
  --description "Security group for private instances"
  --vpc-id vpc-0abc1234def567890
```

Output:

```
{
    "GroupId": "sg-0abcd5678efgh1234"
}
```

7.3 Add Inbound Rule for Bastion Access

Allow SSH from bastion host IP:

```
$ aws ec2 authorize-security-group-ingress
  --group-id sg-0abcd5678efgh1234
  --protocol tcp
  --port 22
  --cidr 203.0.113.25/32
```

Output:

```
{
    "Return": true
}
```

7.4 Network ACLs (NACLs)

Create a Network ACL:

```
$ aws ec2 create-network-acl --vpc-id vpc-0abc1234def567890
```

Output:

```
{
    "NetworkAcl": {
        "NetworkAclId": "acl-0abc1234def567890",
        "VpcId": "vpc-0abc1234def567890"
    }
}
```

7.5 Add NACL Rules

Allow inbound HTTP/HTTPS:

```
$ aws ec2 create-network-acl-entry
  --network-acl-id acl-0abc1234def567890
  --rule-number 100
  --protocol tcp
  --port-range From=80,To=443
  --egress false
  --rule-action allow
```

Output:

```
{
    "Return": true
}
```

Deny all other inbound traffic:

```
$ aws ec2 create-network-acl-entry
  --network-acl-id acl-0abc1234def567890
  --rule-number 200
  --protocol -1
  --egress false
  --rule-action deny
```

Output:

```
{
    "Return": true
}
```

7.6 Bastion Host Setup

Launch a Bastion Host in public subnet:

```
$ aws ec2 run-instances
  --image-id ami-0abcdef1234567890
  --count 1
  --instance-type t2.micro
  --key-name DevOpsKey
  --security-group-ids sg-0abc1234def567890
  --subnet-id subnet-0abcd1234ef567890
```

Output:

```
{
    "Instances": [
        {
            "InstanceId": "i-0abcdef5678901234",
            "State": {"Code": 0, "Name": "pending"},
            "InstanceType": "t2.micro",
            "KeyName": "DevOpsKey"
        }
    ]
}
```

Tip: Use Bastion Host to SSH into private instances using:

```
$ ssh -i DevOpsKey.pem ec2-user@<Private-Instance-IP> -J ec2-user@<Bastion-IP>
```

8 RDS Setup, Subnet Groups, and Security

8.1 Introduction

Amazon RDS (Relational Database Service) allows you to run managed databases in AWS. Key concepts:

- **RDS Instance Types:** db.t2.micro, db.m5.large, etc.
- **Subnet Groups:** Defines which subnets RDS instances can use.
- **Security:** Controlled via Security Groups and IAM roles.

8.2 Create an RDS Subnet Group

Create a DB Subnet Group:

```
$ aws rds create-db-subnet-group
  --db-subnet-group-name DevOpsSubnetGroup
  --db-subnet-group-description "Subnet group for RDS instances"
  --subnet-ids subnet-0abcd1234ef567890 subnet-0abcd5678efgh1234
```

Output:

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupName": "DevOpsSubnetGroup",
        "DBSubnetGroupDescription": "Subnet group for RDS instances",
        "SubnetIds": [
            "subnet-0abcd1234ef567890",
            "subnet-0abcd5678efgh1234"
        ]
    }
}
```

8.3 Launch an RDS Instance

Create a MySQL RDS instance:

```
$ aws rds create-db-instance
  --db-instance-identifier DevOpsDB
  --db-instance-class db.t2.micro
  --engine mysql
  --master-username admin
  --master-user-password Admin1234
  --allocated-storage 20
  --vpc-security-group-ids sg-0abcd5678efgh1234
  --db-subnet-group-name DevOpsSubnetGroup
```

Output:

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "DevOpsDB",
        "DBInstanceStatus": "creating",
        "DBInstanceClass": "db.t2.micro",
        "Engine": "mysql",
        "MasterUsername": "admin"
    }
}
```

8.4 Check RDS Instance Status

Check if RDS instance is available:

```
$ aws rds describe-db-instances
  --db-instance-identifier DevOpsDB
```

Output:

```
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "DevOpsDB",
            "DBInstanceStatus": "available",
            "Endpoint": {
                "Address": "devopsdb.abcdefgh.us-east-1.rds.amazonaws.com",
                "Port": 3306
            },
            "DBInstanceClass": "db.t2.micro"
        }
    ]
}
```

8.5 RDS Security Best Practices

- Enable encryption at rest using KMS.
- Apply security group rules to restrict access.

- Enable automated backups and snapshots.
- Regularly rotate master credentials.

Tip: Use private subnets for RDS instances to prevent direct internet exposure.

9 S3 Buckets, Versioning, and Lifecycle Policies

9.1 Introduction

Amazon S3 (Simple Storage Service) is a scalable object storage service. Key concepts:

- **Buckets:** Containers for storing objects.
- **Versioning:** Keep multiple versions of objects.
- **Lifecycle Policies:** Automate deletion or transition of objects.

9.2 Create an S3 Bucket

Create a new S3 bucket:

```
$ aws s3 mb s3://my-devops-bucket
```

Output:

```
make_bucket: my-devops-bucket
```

9.3 Enable Versioning on Bucket

Enable versioning:

```
$ aws s3api put-bucket-versioning
  --bucket my-devops-bucket
  --versioning-configuration Status=Enabled
```

Output:

```
{}
```

9.4 Upload an Object to S3

Upload a file to S3 bucket:

```
$ aws s3 cp myfile.txt s3://my-devops-bucket/
```

Output:

```
upload: ./myfile.txt to s3://my-devops-bucket/myfile.txt
```

9.5 List Objects in S3 Bucket

List all objects:

```
$ aws s3 ls s3://my-devops-bucket/
```

Output:

```
2025-09-01 12:00:00  myfile.txt
```

9.6 S3 Lifecycle Policy Example

Create a lifecycle policy to transition objects to Glacier:

```
$ aws s3api put-bucket-lifecycle-configuration
  --bucket my-devops-bucket
  --lifecycle-configuration '{
      "Rules": [
          {
              "ID": "ArchiveToGlacier",
              "Status": "Enabled",
              "Prefix": "",
              "Transitions": [
                  {"Days": 30, "StorageClass": "GLACIER"}
              ]
          }
      ]
  }'
```

Output:

```
{}
```

9.7 S3 Best Practices

- Enable bucket versioning for critical data.
- Use server-side encryption (SSE) for security.
- Implement lifecycle policies to optimize storage costs.
- Apply bucket policies and IAM roles for secure access.

Tip: Use `--recursive` option for bulk uploads and downloads.

10 CloudWatch Monitoring, Alarms, and Logs

10.1 Introduction

Amazon CloudWatch allows you to monitor AWS resources and applications in real-time. Key concepts:

- **Metrics:** Quantitative data about resources (CPU, Memory, Network).
- **Alarms:** Trigger actions based on thresholds.
- **Logs:** Centralized logging for troubleshooting and analysis.

10.2 List CloudWatch Metrics

List available metrics:

```
$ aws cloudwatch list-metrics
```

Sample Output:

```
{
    "Metrics": [
        {
            "Namespace": "AWS/EC2",
            "MetricName": "CPUUtilization",
            "Dimensions": [{"Name":"InstanceId","Value":"i-0abcdef5678901234"}]
        }
    ]
}
```

10.3 Create a CloudWatch Alarm

Create alarm for high CPU utilization:

```
$ aws cloudwatch put-metric-alarm
  --alarm-name HighCPU
  --metric-name CPUUtilization
  --namespace AWS/EC2
  --statistic Average
  --period 300
  --threshold 80
  --comparison-operator GreaterThanThreshold
  --evaluation-periods 2
  --alarm-actions arn:aws:sns:us-east-1:123456789012:NotifyMe
  --dimensions Name=InstanceId,Value=i-0abcdef5678901234
```

Output:

```
{}
```

10.4 Enable CloudWatch Logs for EC2

Install CloudWatch agent on EC2:

```
$ sudo yum install amazon-cloudwatch-agent -y
```

Configure CloudWatch agent:

```
$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
```

Start the CloudWatch agent:

```
$ sudo systemctl start amazon-cloudwatch-agent
$ sudo systemctl enable amazon-cloudwatch-agent
```

10.5 View Logs in CloudWatch

List log groups:

```
$ aws logs describe-log-groups
```

View log streams:

```
$ aws logs describe-log-streams --log-group-name /var/log/messages
```

Get latest logs:

```
$ aws logs get-log-events
  --log-group-name /var/log/messages
  --log-stream-name <log-stream-name>
  --limit 20
```

10.6 Best Practices

- Set meaningful thresholds for alarms to avoid false positives.
- Use tags to organize metrics and logs.
- Centralize logs across multiple accounts for better observability.
- Archive old logs to S3 for long-term storage.

Tip: Combine CloudWatch Alarms with SNS to receive instant notifications on critical events.

11 CodePipeline, CodeBuild, and CI/CD Basics

11.1 Introduction

AWS CodePipeline and CodeBuild enable automated Continuous Integration and Continuous Deployment (CI/CD). Key concepts:

- **CodePipeline:** Orchestrates the build, test, and deployment process.
- **CodeBuild:** Compiles source code, runs tests, and produces artifacts.
- **CI/CD:** Automates code delivery and reduces manual errors.

11.2 Create a CodeBuild Project

Create a CodeBuild project:

```
$ aws codebuild create-project
  --name DevOpsBuildProject
  --source type=GITHUB,location=https://github.com/user/repo.git
  --artifacts type=NO_ARTIFACTS
  --environment type=LINUX_CONTAINER,image=aws/codebuild/standard:5.0,computeType=BU
```

Output:

```
{
    "project": {
        "name": "DevOpsBuildProject",
        "arn": "arn:aws:codebuild:us-east-1:123456789012:project/DevOpsBuildProject"
        "created": "2025-09-01T14:00:00Z"
    }
}
```

11.3 Create a Simple CodePipeline

Create pipeline with source, build, and deploy stages:

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

Output:

```
{
    "pipeline": {
        "name": "DevOpsPipeline",
        "version": 1,
        "created": "2025-09-01T14:05:00Z"
    }
}
```

11.4 Example pipeline-definition.json

```
{
  "pipeline": {
    "name": "DevOpsPipeline",
    "roleArn": "arn:aws:iam::123456789012:role/AWSCodePipelineServiceRole",
    "stages": [
      {
        "name": "Source",
        "actions": [
          {
            "name": "SourceAction",
            "actionTypeId": {
              "category": "Source",
              "owner": "ThirdParty",
              "provider": "GitHub",
              "version": "1"
            },
            "outputArtifacts": [{"name": "SourceArtifact"}],
            "configuration": {
              "Owner": "user",
              "Repo": "repo",
              "Branch": "main",
              "OAuthToken": "********"
            }
          }
        ]
      },
      {
        "name": "Build",
        "actions": [
          {
            "name": "BuildAction",
            "actionTypeId": {
              "category": "Build",
              "owner": "AWS",
              "provider": "CodeBuild",
              "version": "1"
            },
            "inputArtifacts": [{"name": "SourceArtifact"}],
            "outputArtifacts": [{"name": "BuildArtifact"}],
            "configuration": {"ProjectName": "DevOpsBuildProject"}
          }
        ]
      }
    ]
  }
}
```

11.5 CI/CD Best Practices

- Use separate AWS accounts or stages for dev, test, and prod.
- Integrate automated tests in the build stage.
- Keep pipelines declarative and version-controlled.
- Monitor pipeline status with CloudWatch events and SNS notifications.

Tip: Always use IAM roles with least privilege for pipeline and build projects.

12 ECS and Fargate Deployment Basics

12.1 Introduction

Amazon ECS (Elastic Container Service) allows you to run and manage Docker containers on AWS. Fargate is a serverless compute engine for ECS that eliminates the need to manage EC2 instances. Key concepts:

- **Task Definition:** Blueprint for your container(s) including CPU, memory, and Docker image.

- **Service:** Manages running tasks and ensures desired count.
- **Cluster:** Logical grouping of tasks or services.

12.2 Create ECS Cluster

Create a new ECS cluster:

```
$ aws ecs create-cluster --cluster-name DevOpsCluster
```

Output:

```
{
    "cluster": {
        "clusterName": "DevOpsCluster",
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/DevOpsCluster",
        "status": "ACTIVE"
    }
}
```

12.3 Register a Task Definition

Register a simple Fargate task definition:

```
$ aws ecs register-task-definition
  --family DevOpsTask
  --network-mode awsvpc
  --requires-compatibilities FARGATE
  --cpu 256
  --memory 512
  --container-definitions '[
      {
          "name": "web-app",
          "image": "nginx:latest",
          "portMappings": [{"containerPort": 80, "protocol": "tcp"}]
      }
  ]'
```

Output:

```
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/Dev
        "family": "DevOpsTask",
        "revision": 1
    }
}
```

12.4 Run ECS Fargate Service

Create a Fargate service:

```
$ aws ecs create-service
  --cluster DevOpsCluster
  --service-name WebAppService
  --task-definition DevOpsTask
  --desired-count 2
  --launch-type FARGATE
  --network-configuration '{
      "awsvpcConfiguration": {
          "subnets": ["subnet-0abcd1234ef567890"],
          "securityGroups": ["sg-0abcd5678efgh1234"],
          "assignPublicIp": "ENABLED"
      }
  }'
```

Output:

```
{
    "service": {
        "serviceName": "WebAppService",
        "status": "ACTIVE",
        "desiredCount": 2,
        "runningCount": 2
    }
}
```

12.5 Best Practices for ECS + Fargate

- Use IAM roles for tasks to limit permissions.
- Enable CloudWatch logging for container output.
- Set up auto-scaling policies based on CPU/memory metrics.
- Use multiple availability zones for high availability.

Tip: Fargate simplifies container management but monitor costs for high-scale deployments.

13 Lambda Functions, IAM Roles, and Event Triggers

13.1 Introduction

AWS Lambda allows you to run code without provisioning or managing servers. Key concepts:

- **Lambda Function:** Your code executed on-demand.
- **IAM Role:** Permissions for Lambda to access AWS resources.
- **Event Trigger:** Initiates Lambda execution automatically.

13.2 Create an IAM Role for Lambda

Create IAM role for Lambda with basic execution policy:

```
$ aws iam create-role --role-name LambdaExecRole
  --assume-role-policy-document '{
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Principal": {"Service": "lambda.amazonaws.com"},
              "Action": "sts:AssumeRole"
          }
      ]
  }'
```

Attach AWSLambdaBasicExecutionRole policy:

```
$ aws iam attach-role-policy
  --role-name LambdaExecRole
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Output:

```
Successfully created role LambdaExecRole
Successfully attached policy AWSLambdaBasicExecutionRole
```

13.3 Create a Lambda Function

Create Lambda function using Python:

```
$ aws lambda create-function
  --function-name DevOpsHello
  --runtime python3.9
  --role arn:aws:iam::123456789012:role/LambdaExecRole
  --handler lambda_function.lambda_handler
  --zip-file fileb://lambda_function.zip
```

Output:

```
{
    "FunctionName": "DevOpsHello",
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:DevOpsHello",
    "Runtime": "python3.9",
    "Role": "arn:aws:iam::123456789012:role/LambdaExecRole",
    "Handler": "lambda_function.lambda_handler"
}
```

13.4 Invoke Lambda Function

Invoke Lambda function:

```
$ aws lambda invoke
  --function-name DevOpsHello
  output.txt
```

Output:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

13.5 Add Event Trigger (S3 Upload)

Add S3 event to trigger Lambda on object creation:

```
$ aws s3api put-bucket-notification-configuration
  --bucket my-devops-bucket
  --notification-configuration '{
      "LambdaFunctionConfigurations": [
          {
              "LambdaFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:D
              "Events": ["s3:ObjectCreated:*"]
          }
      ]
  }'
```

13.6 Best Practices for Lambda

- Keep Lambda functions small and single-purpose.
- Use environment variables for configuration.
- Monitor execution using CloudWatch Logs.
- Use IAM roles with least privilege to access resources.
- Consider cost for high-frequency invocations.

Tip: Combine Lambda with S3, DynamoDB, and SNS for event-driven architectures.

14 CloudFormation Basics and Stack Deployment

14.1 Introduction

AWS CloudFormation allows you to model, provision, and manage AWS resources using code. Key concepts:

- **Template:** JSON or YAML file defining AWS resources.
- **Stack:** Collection of resources created and managed as a single unit.
- **Change Set:** Preview changes before updating stacks.

14.2 Create a Simple CloudFormation Template

```
AWSTemplateFormatVersion: '2010-09-09'
Description: DevOps Sample EC2 Instance
Resources:
  DevOpsEC2:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t2.micro
      ImageId: ami-0abcdef1234567890
      KeyName: DevOpsKey
```

14.3 Deploy CloudFormation Stack

Create a new stack:

```
$ aws cloudformation create-stack
  --stack-name DevOpsStack
  --template-body file://devops-template.yaml
```

Output:

```
{
    "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/DevOpsStack/abcd
}
```

14.4 Check Stack Status

Describe stack events and status:

```
$ aws cloudformation describe-stacks --stack-name DevOpsStack
```

Sample Output:

```
{
    "Stacks": [
        {
            "StackName": "DevOpsStack",
            "StackStatus": "CREATE_COMPLETE",
            "CreationTime": "2025-09-01T15:00:00Z"
        }
    ]
}
```

14.5 Update a Stack using Change Set

Create a change set:

```
$ aws cloudformation create-change-set
  --stack-name DevOpsStack
  --change-set-name UpdateInstanceType
  --template-body file://devops-template-update.yaml
  --change-set-type UPDATE
```

Execute the change set:

```
$ aws cloudformation execute-change-set
  --stack-name DevOpsStack
  --change-set-name UpdateInstanceType
```

14.6 CloudFormation Best Practices

- Keep templates modular and reusable.
- Use parameters and mappings to customize stacks.
- Version control templates using Git.
- Monitor stack events and logs for errors.
- Test templates in dev/test accounts before production deployment.

Tip: Use CloudFormation drift detection to monitor manual changes in resources.

15 Elastic Beanstalk Basics and Application Deployment

15.1 Introduction

AWS Elastic Beanstalk is a Platform-as-a-Service (PaaS) that simplifies application deployment. Key concepts:

- **Application:** Container for environments and versions.
- **Environment:** Deployed instances of an application.
- **Version:** Specific code bundle deployed to environment.

15.2 Create an Elastic Beanstalk Application

Create a new application:

```
$ aws elasticbeanstalk create-application
  --application-name DevOpsApp
  --description "Sample DevOps Application"
```

Output:

```
{
    "Application": {
        "ApplicationName": "DevOpsApp",
        "Description": "Sample DevOps Application",
        "DateCreated": "2025-09-01T16:00:00Z"
    }
}
```

15.3 Create an Environment and Deploy Application

Deploy a web application using Python:

```
$ aws elasticbeanstalk create-environment
  --application-name DevOpsApp
  --environment-name DevOpsAppEnv
  --solution-stack-name "64bit Amazon Linux 2 v5.4.7 running Python 3.9"
  --version-label v1
  --option-settings file://options.json
```

Output:

```
{
    "EnvironmentName": "DevOpsAppEnv",
    "EnvironmentId": "e-abc123xyz",
    "Status": "Launching",
    "Health": "Green"
}
```

15.4 Update Application Version

Upload new version to S3 and deploy:

```
$ aws elasticbeanstalk create-application-version
  --application-name DevOpsApp
  --version-label v2
  --source-bundle S3Bucket="my-devops-bucket",S3Key="app-v2.zip"

$ aws elasticbeanstalk update-environment
  --environment-name DevOpsAppEnv
  --version-label v2
```

15.5 Elastic Beanstalk Best Practices

- Use separate environments for dev, test, and prod.
- Enable enhanced health monitoring for better visibility.
- Store application versions in S3 for version control.
- Use environment variables for configuration instead of hardcoding.

- Monitor logs and events for troubleshooting deployment issues.

Tip: Use rolling deployments to minimize downtime during updates.

16 CloudTrail and Auditing for Compliance

16.1 Introduction

AWS CloudTrail allows you to monitor, log, and retain account activity across your AWS infrastructure. Key concepts:

- **Trail:** Configuration to capture and store API activity.
- **Event:** Records of API calls including user, time, source IP, and parameters.
- **Log File Validation:** Ensures integrity of logs for auditing.

16.2 Create a CloudTrail Trail

Create a new CloudTrail trail:

```
$ aws cloudtrail create-trail
  --name DevOpsTrail
  --s3-bucket-name my-devops-logs
  --include-global-service-events
```

Output:

```
{
    "Name": "DevOpsTrail",
    "S3BucketName": "my-devops-logs",
    "IncludeGlobalServiceEvents": true,
    "IsMultiRegionTrail": false
}
```

16.3 Start Logging

Enable CloudTrail logging:

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

Output:

```
{
    "ResponseMetadata": {
        "HTTPStatusCode": 200
    }
}
```

16.4 View Trail Events

Lookup events for auditing:

```
$ aws cloudtrail lookup-events
  --lookup-attributes AttributeKey=Username,AttributeValue=DevOpsUser
  --max-results 5
```

Sample Output:

```
{
    "Events": [
        {
            "EventId": "abcd1234-ef56-7890-gh12-ijkl34567890",
            "EventName": "CreateBucket",
            "Username": "DevOpsUser",
            "EventTime": "2025-09-01T17:00:00Z",
            "Resources": [{"ResourceName": "my-first-bucket"}]
        }
    ]
}
```

16.5 CloudTrail Best Practices

- Enable multi-region trails for complete coverage.
- Encrypt logs using SSE-KMS for security.
- Enable log file validation for compliance auditing.
- Integrate with CloudWatch Logs to monitor events in real-time.
- Regularly review trails for unusual activity.

Tip: Use CloudTrail insights to detect unusual API activity patterns automatically.

17 AWS Config: Resource Inventory and Compliance

17.1 Introduction

AWS Config helps you track AWS resource configurations, compliance, and changes over time. Key concepts:

- **Configuration Recorder:** Records resource configurations continuously.
- **Delivery Channel:** Sends recorded configurations to an S3 bucket.
- **Rules:** Evaluate resource configurations against desired policies.

17.2 Set Up AWS Config Recorder

Create a configuration recorder:

```
$ aws configservice put-configuration-recorder
  --configuration-recorder '{
      "name": "DevOpsRecorder",
      "roleARN": "arn:aws:iam::123456789012:role/ConfigRole",
      "recordingGroup": {
          "allSupported": true,
          "includeGlobalResourceTypes": true
      }
  }'
```

17.3 Set Up Delivery Channel

Create delivery channel to S3 bucket:

```
$ aws configservice put-delivery-channel
  --delivery-channel '{
      "name": "DevOpsChannel",
      "s3BucketName": "my-config-bucket"
  }'
```

17.4 Start Recording Configurations

Start configuration recorder:

```
$ aws configservice start-configuration-recorder
  --configuration-recorder-name DevOpsRecorder
```

Output:

```
{
    "ResponseMetadata": {
        "HTTPStatusCode": 200
    }
}
```

17.5 Add AWS Config Rules for Compliance

Add a managed rule to check S3 bucket encryption:

```
$ aws configservice put-config-rule
  --config-rule '{
      "ConfigRuleName": "s3-bucket-encrypted",
      "Description": "Check whether S3 buckets have encryption enabled",
      "Scope": {},
      "Source": {
          "Owner": "AWS",
          "SourceIdentifier": "S3_BUCKET_SERVER_SIDE_ENCRYPTION_ENABLED"
      }
  }'
```

Output:

```
{
    "ConfigRule": {
        "ConfigRuleName": "s3-bucket-encrypted",
        "ConfigRuleArn": "arn:aws:config:us-east-1:123456789012:config-rule/config-r
    }
}
```

17.6 AWS Config Best Practices

- Enable all supported resource types to track comprehensive inventory.
- Integrate Config with CloudWatch Events for real-time alerts.
- Use managed rules for common compliance standards (PCI, HIPAA, CIS).
- Periodically review compliance dashboards to enforce governance.
- Store historical configurations for audit and rollback purposes.

Tip: Combine AWS Config with CloudTrail for end-to-end auditing and governance.

18 Amazon SNS: Notifications and Messaging

18.1 Introduction

Amazon SNS is a fully managed pub/sub messaging service for sending notifications. Key concepts:

- **Topic:** Logical access point for publishing messages.
- **Subscription:** Receivers of messages (email, SMS, Lambda, SQS).
- **Publisher:** Sends messages to a topic.

18.2 Create SNS Topic

Create a new topic:

```
$ aws sns create-topic --name DevOpsNotifications
```

Output:

```
{
    "TopicArn": "arn:aws:sns:us-east-1:123456789012:DevOpsNotifications"
}
```

18.3 Subscribe to SNS Topic

Subscribe email endpoint to topic:

```
$ aws sns subscribe
  --topic-arn arn:aws:sns:us-east-1:123456789012:DevOpsNotifications
  --protocol email
  --notification-endpoint user@example.com
```

18.4 Publish a Message

Send a notification to topic subscribers:

```
$ aws sns publish
  --topic-arn arn:aws:sns:us-east-1:123456789012:DevOpsNotifications
  --message "AWS DevOps Alert: Deployment Successful"
```

Tip: Combine SNS with CloudWatch alarms for automated notifications.

19 Amazon SQS and EventBridge: Event-Driven Architectures

19.1 Amazon SQS (Simple Queue Service)

SQS allows decoupling of components using message queues. Key concepts:

- **Queue:** Stores messages until consumed by a receiver.
- **Producer:** Sends messages to queue.
- **Consumer:** Receives and processes messages.

19.2 Create an SQS Queue

Create a standard queue:

```
$ aws sqs create-queue --queue-name DevOpsQueue
```

Output:

```
{
    "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/DevOpsQueue"
}
```

19.3 Send and Receive Messages

Send message to SQS queue:

```
$ aws sqs send-message
  --queue-url https://sqs.us-east-1.amazonaws.com/123456789012/DevOpsQueue
  --message-body "Deploy App Version 2"
```

Receive message from queue:

```
$ aws sqs receive-message
  --queue-url https://sqs.us-east-1.amazonaws.com/123456789012/DevOpsQueue
```

19.4 Amazon EventBridge (CloudWatch Events)

EventBridge allows routing events from AWS services or custom applications. Key concepts:

- **Rule:** Defines which events trigger actions.
- **Target:** Lambda, SQS, SNS, or other services that respond to events.

19.5 Create EventBridge Rule

Route S3 upload events to Lambda:

```
$ aws events put-rule
  --name DevOpsS3UploadRule
  --event-pattern '{
      "source": ["aws.s3"],
      "detail-type": ["Object Created"]
  }'

$ aws events put-targets
  --rule DevOpsS3UploadRule
  --targets '[
      {
          "Id": "LambdaTarget",
          "Arn": "arn:aws:lambda:us-east-1:123456789012:function:DevOpsHello"
      }
  ]'
```

19.6 Best Practices

- Use SNS for fan-out messaging to multiple subscribers.
- Use SQS for decoupling and reliable message processing.
- Use EventBridge for event-driven automation and workflows.
- Monitor queue length and delivery failures for troubleshooting.

20 Amazon DynamoDB: Basics and Tables

20.1 Introduction

DynamoDB is a fully managed NoSQL database offering high performance at scale. Key concepts:

- **Table:** Collection of items (rows).
- **Item:** Single record in a table.
- **Attribute:** Data fields in an item.
- **Primary Key:** Unique identifier for items (Partition Key, optionally Sort Key).

20.2 Create a DynamoDB Table

Create a table with Partition Key "UserId":

```
$ aws dynamodb create-table
  --table-name DevOpsUsers
  --attribute-definitions AttributeName=UserId,AttributeType=S
  --key-schema AttributeName=UserId,KeyType=HASH
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

Output:

```
{
    "TableDescription": {
        "TableName": "DevOpsUsers",
        "TableStatus": "CREATING"
    }
}
```

20.3 Check Table Status

Describe table:

```
$ aws dynamodb describe-table --table-name DevOpsUsers
```

Output:

```
{
    "Table": {
        "TableName": "DevOpsUsers",
        "TableStatus": "ACTIVE",
        "ItemCount": 0
    }
}
```

21 DynamoDB CRUD Operations

21.1 Add Item

Insert an item into DevOpsUsers:

```
$ aws dynamodb put-item
  --table-name DevOpsUsers
  --item '{"UserId": {"S": "U1001"}, "Name": {"S": "Sainath"}, "Role": {"S": "DevOps
```

Output:

```
{}
```

21.2 Read Item

Get an item by UserId:

```
$ aws dynamodb get-item
  --table-name DevOpsUsers
  --key '{"UserId": {"S": "U1001"}}'
```

Output:

```
{
    "Item": {
        "UserId": {"S": "U1001"},
        "Name": {"S": "Sainath"},
        "Role": {"S": "DevOps"}
    }
}
```

21.3 Update Item

Update the Role of a user:

```
$ aws dynamodb update-item
  --table-name DevOpsUsers
  --key '{"UserId": {"S": "U1001"}}'
  --update-expression "SET Role = :r"
  --expression-attribute-values '{":r": {"S": "Senior DevOps"}}'
```

21.4 Delete Item

Delete a user by UserId:

```
$ aws dynamodb delete-item
  --table-name DevOpsUsers
  --key '{"UserId": {"S": "U1001"}}'
```

22 DynamoDB Advanced Features

22.1 Global Secondary Index (GSI)

Add a GSI on "Role" attribute:

```
$ aws dynamodb update-table
  --table-name DevOpsUsers
  --attribute-definitions AttributeName=Role,AttributeType=S
  --global-secondary-index-updates '[
      {
          "Create": {
              "IndexName": "RoleIndex",
              "KeySchema": [{"AttributeName": "Role","KeyType":"HASH"}],
              "Projection": {"ProjectionType":"ALL"},
              "ProvisionedThroughput": {"ReadCapacityUnits":5,"WriteCapacityUnits":5
          }
      }
  ]'
```

22.2 Query by GSI

Query users by Role:

```
$ aws dynamodb query
  --table-name DevOpsUsers
  --index-name RoleIndex
  --key-condition-expression "Role = :r"
  --expression-attribute-values '{":r": {"S": "DevOps"}}'
```

22.3 Best Practices

- Choose partition keys wisely to avoid hot partitions.
- Use GSIs sparingly for flexible queries.
- Monitor read/write capacity and enable auto-scaling.
- Use DynamoDB Streams for event-driven architectures.
- Enable point-in-time recovery (PITR) for backups.

23 Amazon RDS: Relational Database Service Basics

23.1 Introduction

Amazon RDS is a managed relational database service supporting multiple engines (MySQL, PostgreSQL, MariaDB, Oracle, SQL Server). Key concepts:

- **DB Instance:** Managed database server.
- **DB Snapshot:** Backup of the DB instance.
- **Multi-AZ:** High availability with failover support.
- **Read Replica:** Offload read queries for scalability.

23.2 Create RDS Database Instance

Create MySQL RDS instance:

```
$ aws rds create-db-instance
  --db-instance-identifier DevOpsRDS
  --db-instance-class db.t3.micro
  --engine mysql
  --master-username admin
  --master-user-password MyPassword123
  --allocated-storage 20
  --backup-retention-period 7
  --publicly-accessible
```

Output:

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "DevOpsRDS",
        "DBInstanceStatus": "creating"
    }
}
```

24 RDS Backup and Restore

24.1 Create DB Snapshot

Take snapshot of RDS instance:

```
$ aws rds create-db-snapshot
  --db-instance-identifier DevOpsRDS
  --db-snapshot-identifier DevOpsRDSBackup1
```

Output:

```
{
    "DBSnapshot": {
        "DBSnapshotIdentifier": "DevOpsRDSBackup1",
        "DBInstanceIdentifier": "DevOpsRDS",
        "Status": "creating"
    }
}
```

24.2 Restore from Snapshot

Restore DB from snapshot:

```
$ aws rds restore-db-instance-from-db-snapshot
  --db-instance-identifier DevOpsRDSRestore
  --db-snapshot-identifier DevOpsRDSBackup1
```

25 RDS Read Replicas

25.1 Create Read Replica

Create a read replica of DevOpsRDS:

```
$ aws rds create-db-instance-read-replica
  --db-instance-identifier DevOpsRDSReplica
  --source-db-instance-identifier DevOpsRDS
```

Output:

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "DevOpsRDSReplica",
        "DBInstanceStatus": "creating"
    }
}
```

25.2 Best Practices for RDS

- Enable Multi-AZ for production workloads.
- Schedule automated backups and retain snapshots.
- Use read replicas to scale read-heavy applications.
- Monitor CPU, storage, and connections with CloudWatch.
- Use IAM roles for RDS access when possible.

26 RDS Security and Maintenance

26.1 Enable Encryption

Create encrypted RDS instance:

```
$ aws rds create-db-instance
  --db-instance-identifier DevOpsRDSEncrypted
  --db-instance-class db.t3.micro
  --engine mysql
  --master-username admin
  --master-user-password MyPassword123
  --allocated-storage 20
  --storage-encrypted
  --kms-key-id arn:aws:kms:us-east-1:123456789012:key/abcd-1234
```

26.2 Apply Patches and Maintenance

Apply pending maintenance:

```
$ aws rds apply-pending-maintenance-action
  --resource-identifier arn:aws:rds:us-east-1:123456789012:db:DevOpsRDS
  --apply-action system-update
  --opt-in-type immediate
```

26.3 Best Practices

- Enable encryption for sensitive data.
- Keep automated backups enabled with retention period.
- Schedule maintenance windows during low traffic periods.
- Use monitoring and alarms for performance and storage.

27 Amazon EKS: Kubernetes on AWS

27.1 Introduction

Amazon EKS is a managed Kubernetes service for deploying, managing, and scaling containerized applications. Key concepts:

- **Cluster:** Control plane to manage worker nodes and Kubernetes resources.

- **Node Group:** EC2 instances running containers in the cluster.
- **Fargate:** Serverless compute for running containers without managing nodes.

28 EKS Cluster Creation

Create EKS cluster using AWS CLI:

```
$ aws eks create-cluster
  --name DevOpsEKS
  --role-arn arn:aws:iam::123456789012:role/EKSClusterRole
  --resources-vpc-config subnetIds=subnet-123,subnet-456,securityGroupIds=sg-123
```

Output:

```
{
    "cluster": {
        "name": "DevOpsEKS",
        "status": "CREATING"
    }
}
```

29 EKS Node Groups

Create managed node group:

```
$ aws eks create-nodegroup
  --cluster-name DevOpsEKS
  --nodegroup-name DevOpsNodes
  --subnets subnet-123 subnet-456
  --instance-types t3.medium
  --scaling-config minSize=2,maxSize=5,desiredSize=2
```

29.1 Best Practices

- Use multiple subnets for high availability.
- Enable autoscaling for workload fluctuations.
- Tag nodes for cost allocation and monitoring.

30 EKS Fargate Profiles

Create Fargate profile for serverless workloads:

```
$ aws eks create-fargate-profile
  --cluster-name DevOpsEKS
  --fargate-profile-name DevOpsFargate
  --pod-execution-role-arn arn:aws:iam::123456789012:role/EKSFargateRole
  --subnets subnet-123 subnet-456
  --selectors namespace=default
```

31 EKS Cluster Autoscaling

Enable cluster autoscaler using Helm:

```
$ helm repo add autoscaler https://kubernetes.github.io/autoscaler
$ helm install cluster-autoscaler autoscaler/cluster-autoscaler
  --namespace kube-system
  --set autoDiscovery.clusterName=DevOpsEKS
  --set awsRegion=us-east-1
```

32 ConfigMaps in Kubernetes

Create ConfigMap for application config:

```
$ kubectl create configmap app-config
  --from-literal=LOG_LEVEL=DEBUG
  --from-literal=ENV=DEV
```

View ConfigMap:

```
$ kubectl get configmap app-config -o yaml
```

33 Secrets in Kubernetes

Create Secret for sensitive info:

```
$ kubectl create secret generic db-secret
  --from-literal=username=admin
  --from-literal=password=MySecret123
```

View Secret (base64 encoded):

```
$ kubectl get secret db-secret -o yaml
```

34 Deploy Applications on EKS

Create Deployment and Service:

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

Sample Deployment Status:

```
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
devops-app    3/3     3            3           2m
```

35 EKS Best Practices

- Always use IAM roles for service accounts (IRSA).
- Separate workloads between node groups and Fargate profiles.
- Enable logging with CloudWatch Container Insights.
- Use namespaces and labels for organization and RBAC policies.
- Regularly update worker nodes and control plane versions.

36 AWS CodePipeline: Continuous Integration and Deployment

36.1 Introduction

AWS CodePipeline is a fully managed CI/CD service that automates build, test, and deployment phases. Key concepts:

- **Pipeline:** Defines workflow from source to deployment.
- **Stage:** Logical division of pipeline steps (Source, Build, Deploy).
- **Action:** Individual tasks within a stage.

36.2 Create a Pipeline

Create a new CodePipeline using AWS CLI:

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

Output:

```
{
    "pipeline": {
        "name": "DevOpsPipeline",
        "roleArn": "arn:aws:iam::123456789012:role/CodePipelineRole",
        "artifactStore": { "type": "S3", "location": "devops-pipeline-artifacts" }
    }
}
```

37 AWS CodeBuild: Build and Test

37.1 Introduction

CodeBuild is a fully managed build service for compiling code, running tests, and producing artifacts. Key concepts:

- **Project:** Defines source, build commands, and environment.
- **Environment:** OS, runtime, and compute type.
- **Buildspec:** YAML file defining build commands.

37.2 Create a CodeBuild Project

Create CodeBuild project using AWS CLI:

```
$ aws codebuild create-project \
  --name DevOpsBuild \
  --source type=GITHUB,location=https://github.com/username/repo \
  --artifacts type=NO_ARTIFACTS \
  --environment type=LINUX_CONTAINER,computeType=BUILD_GENERAL1_SMALL,image=aws/code
```

37.3 Build Project

Start a build:

```
$ aws codebuild start-build --project-name DevOpsBuild
```

Sample Output:

```
{
    "build": {
        "id": "DevOpsBuild:12345-abcde-67890",
        "buildStatus": "IN_PROGRESS"
    }
}
```

38 AWS CodeDeploy: Deployment Automation

38.1 Introduction

CodeDeploy automates deployment to EC2, Lambda, or ECS. Key concepts:

- **Application:** Container for deployment artifacts.
- **Deployment Group:** Target instances for deployment.
- **Revision:** Application version to deploy.

38.2 Create CodeDeploy Application

Create application and deployment:

```
$ aws deploy create-application --application-name DevOpsApp

$ aws deploy create-deployment \
  --application-name DevOpsApp \
  --deployment-group-name DevOpsGroup \
  --revision revisionType=S3,s3Location={bucket=devops-artifacts,bundleType=zip,key=
```

38.3 Best Practices

- Use versioning and S3 for deployment artifacts.
- Test in staging before deploying to production.
- Monitor deployment status using AWS CLI or console.
- Combine with CodePipeline for end-to-end CI/CD automation.

39 AWS CloudFormation: Infrastructure as Code

39.1 Introduction

CloudFormation allows you to define AWS infrastructure as code using JSON or YAML templates. Key concepts:

- **Stack:** Collection of AWS resources deployed as a single unit.
- **Template:** JSON or YAML file defining resources.
- **Change Set:** Preview changes before applying them to stacks.

40 Creating a CloudFormation Stack

Deploy stack using AWS CLI:

```
$ aws cloudformation create-stack \
  --stack-name DevOpsStack \
  --template-body file://devops-template.yaml \
  --parameters ParameterKey=InstanceType,ParameterValue=t3.micro
```

Output:

```
{
    "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/DevOpsStack/abc1
}
```

41 Update and Delete CloudFormation Stacks

Update stack:

```
$ aws cloudformation update-stack \
  --stack-name DevOpsStack \
  --template-body file://devops-template.yaml
```

Delete stack:

```
$ aws cloudformation delete-stack --stack-name DevOpsStack
```

42 Stack Outputs and Resources

View stack outputs:

```
$ aws cloudformation describe-stacks --stack-name DevOpsStack
```

List resources in stack:

```
$ aws cloudformation list-stack-resources --stack-name DevOpsStack
```

43 AWS Cost Management and Billing

43.1 Introduction

Effective cost management is essential for AWS workloads. Key tools:

- **Cost Explorer:** Analyze historical costs.
- **Budgets:** Set spending thresholds.
- **Tags:** Allocate costs by project, department, or team.

44 AWS Cost Explorer

Enable Cost Explorer and view costs:

```
$ aws ce get-cost-and-usage \
  --time-period Start=2025-09-01,End=2025-09-30 \
  --granularity MONTHLY \
  --metrics "BlendedCost" "UnblendedCost"
```

Sample Output:

```
{
    "ResultsByTime": [
        {
            "TimePeriod": {"Start": "2025-09-01", "End": "2025-09-30"},
            "Total": {"BlendedCost": {"Amount": "50.25", "Unit": "USD"}}
        }
    ]
}
```

45 AWS Budgets

Create a monthly budget:

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

Tip: Use SNS notifications to alert when budget thresholds are reached.

46 AWS Resource Tagging

Tag EC2 instance for cost allocation:

```
$ aws ec2 create-tags \
  --resources i-0123456789abcdef0 \
  --tags Key=Project,Value=DevOpsGuide
```

46.1 Best Practices

- Tag all resources with Project, Environment, and Owner.
- Use tags in Cost Explorer and billing reports.
- Enforce tagging via AWS Config rules.

47 Reserved Instances and Savings Plans

- Purchase Reserved Instances (RI) for long-term workloads to save costs.
- Use Savings Plans for compute flexibility across EC2, Lambda, and Fargate.
- Monitor usage and adjust commitments accordingly.

48 Cost Optimization Tips

- Right-size instances using CloudWatch metrics.
- Turn off unused resources (EC2, RDS, EBS).
- Use Spot instances for non-critical workloads.
- Enable S3 lifecycle policies to move old data to Glacier.
- Automate budget alerts and monitoring.

49 Cost Reports and Analytics

Generate cost report by service:

```
$ aws ce get-cost-and-usage \
  --time-period Start=2025-09-01,End=2025-09-30 \
  --granularity MONTHLY \
  --group-by Type=DIMENSION,Key=SERVICE
```

Sample Output:

```
{
  "ResultsByTime": [
    {
      "Groups": [
        {"Keys": ["AmazonEC2"], "Metrics": {"BlendedCost": {"Amount": "20.10"}}},
        {"Keys": ["AmazonS3"], "Metrics": {"BlendedCost": {"Amount": "5.25"}}}
      ]
    }
  ]
}
```

50 AWS Billing and Cost Management Summary

- Monitor and optimize AWS costs using Cost Explorer, Budgets, and Tags.
- Implement automation to reduce wastage and overprovisioning.
- Use Reserved Instances and Savings Plans for predictable workloads.
- Review monthly reports to identify trends and optimize usage.

51 AWS Lambda: Serverless Compute

51.1 Introduction

AWS Lambda allows running code without provisioning or managing servers. Key features:

- Event-driven execution
- Automatic scaling
- Pay-per-use pricing

52 Creating a Lambda Function

Create Lambda function using AWS CLI:

```
$ aws lambda create-function \
  --function-name DevOpsFunction \
  --runtime python3.9 \
  --role arn:aws:iam::123456789012:role/LambdaExecRole \
  --handler lambda_function.lambda_handler \
  --zip-file fileb://function.zip
```

Output:

```
{
    "FunctionName": "DevOpsFunction",
    "Runtime": "python3.9",
    "Role": "arn:aws:iam::123456789012:role/LambdaExecRole",
    "Handler": "lambda_function.lambda_handler",
    "State": "Active"
}
```

53 EventBridge Triggers

53.1 Introduction

EventBridge (formerly CloudWatch Events) allows triggering Lambda functions based on events.

Create rule to trigger Lambda every day at 10 AM:

```
$ aws events put-rule \
  --name DailyTrigger \
  --schedule-expression "cron(0 10 * * ? *)"
```

Add Lambda as target:

```
$ aws events put-targets \
  --rule DailyTrigger \
  --targets "Id"="1","Arn"="arn:aws:lambda:us-east-1:123456789012:function:DevOpsFun
```

54 API Gateway: REST APIs for Lambda

54.1 Introduction

API Gateway allows exposing Lambda functions as RESTful APIs with authorization and throttling.

Key features:

- Create REST or HTTP APIs
- Integrate with Lambda, ECS, or other endpoints
- Enable authentication (Cognito, IAM, Lambda Authorizers)

55 API Gateway Integration

Create API Gateway REST API:

```
$ aws apigateway create-rest-api --name DevOpsAPI
```

Integrate Lambda function with API:

```
$ aws apigateway put-integration \
  --rest-api-id abc123 \
  --resource-id xyz789 \
  --http-method POST \
  --type AWS_PROXY \
  --integration-http-method POST \
  --uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda
```

56 Lambda Authorizers

- Used for custom authentication for API Gateway
- Can validate tokens or headers

Create Lambda authorizer:

```
$ aws apigateway create-authorizer \
  --rest-api-id abc123 \
  --name DevOpsAuth \
  --type TOKEN \
  --authorizer-uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn
  --identity-source method.request.header.Authorization
```

57 AWS Step Functions: Workflow Automation

57.1 Introduction

Step Functions allows building state machines for orchestrating multiple AWS services.

Key features:

- Visual workflows
- Parallel execution
- Error handling and retries

58 Creating Step Functions

Define state machine in JSON:

```
{
  "Comment": "DevOps Workflow",
  "StartAt": "Task1",
  "States": {
    "Task1": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:DevOpsFunction",
      "Next": "Task2"
    },
    "Task2": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:AnotherFunction",
      "End": true
    }
  }
}
```

59 Deploy Step Function

Create state machine using AWS CLI:

```
$ aws stepfunctions create-state-machine \
  --name DevOpsWorkflow \
  --definition file://state-machine.json \
  --role-arn arn:aws:iam::123456789012:role/StepFunctionsRole
```

60 Step Functions Best Practices

- Use meaningful state names for clarity
- Implement retries and catch blocks for error handling
- Keep workflows modular with separate Lambda functions
- Monitor executions using CloudWatch Logs

61 Amazon SQS: Simple Queue Service

61.1 Introduction

Amazon SQS is a fully managed message queuing service that enables decoupling microservices, distributed systems, and serverless applications. It allows **asynchronous communication** between components, ensuring messages are not lost and systems remain resilient.

61.2 Key Features

- **Standard Queues:** High throughput, at-least-once delivery, best-effort ordering.
- **FIFO Queues:** Exactly-once processing, preserves message order.
- **Dead-letter Queues (DLQ):** Capture messages that cannot be processed successfully.
- **Visibility Timeout:** Temporarily hides messages from other consumers to avoid duplicate processing.

61.3 CLI Commands with Outputs

Create Standard Queue:

```
$ aws sqs create-queue --queue-name DevOpsQueue
```

Output:

```
{
  "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/DevOpsQueue"
}
```

Send a message:

```
$ aws sqs send-message \
  --queue-url https://sqs.us-east-1.amazonaws.com/123456789012/DevOpsQueue \
  --message-body "Deploy task started"
```

Output:

```
{
  "MD5OfMessageBody": "fae0b27c451c728867a567e8c1bb4e53",
  "MessageId": "1234abcd-5678-efgh-9012-ijkl34567890"
}
```

61.4 Best Practices

- Use DLQs for error handling.
- FIFO queues for order-sensitive workflows.
- Monitor CloudWatch metrics: NumberOfMessagesSent/Received.

61.5 Use Case

CI/CD pipeline pushes deployment tasks to SQS. Multiple worker Lambdas process messages in parallel without overloading resources.

61.6 Interview Tips

Explain differences between Standard and FIFO queues, visibility timeout, DLQs, and integration with Lambda.

62 Amazon SNS: Simple Notification Service

62.1 Introduction

SNS is a fully managed **pub/sub messaging service**. It allows sending notifications to multiple endpoints, such as email, SMS, SQS, and Lambda.

62.2 Key Features

- Broadcast messages to multiple subscribers.
- Integrates with SQS, Lambda, and HTTP/S endpoints.
- Supports filtering policies and message attributes.
- Reliable delivery with retries.

62.3 CLI Commands with Outputs

Create a topic:

```
$ aws sns create-topic --name DevOpsTopic
```

Output:

```
{
  "TopicArn": "arn:aws:sns:us-east-1:123456789012:DevOpsTopic"
}
```

Subscribe email endpoint:

```
$ aws sns subscribe \
  --topic-arn arn:aws:sns:us-east-1:123456789012:DevOpsTopic \
  --protocol email \
  --notification-endpoint admin@example.com
```

62.4 Use Case

Send deployment alerts to team email/SMS whenever a new Lambda executes or EC2 instance changes state.

62.5 Best Practices

- Use message attributes for filtering.
- Combine SNS + SQS for decoupled architectures.
- Monitor using CloudWatch metrics: NumberOfMessagesPublished/Delivered.

63 Event-driven Architectures on AWS

63.1 Introduction

Event-driven architecture allows building **loosely coupled applications** that react to events rather than polling continuously.

63.2 AWS Services

- **SQS:** Queue messages between components.
- **SNS:** Broadcast messages to multiple subscribers.
- **EventBridge:** Trigger Lambda or other services on specific events.
- **Lambda:** Serverless execution on events.

63.3 Example

- Developer pushes code to S3 (EventBridge triggers Lambda)
- Lambda processes files and sends success notifications to SNS
- SQS queues failed messages for retry processing

64 Amazon CloudFront: Content Delivery Network

64.1 Introduction

CloudFront is a **global CDN** that delivers web content with low latency and high transfer speeds.

64.2 Key Features

- Edge caching for static and dynamic content.
- Integrates with S3, EC2, ALB, and Lambda@Edge.
- HTTPS support and custom domain names.
- Real-time metrics and logging.

64.3 CLI Commands

Create distribution pointing to S3 bucket:

```
$ aws cloudfront create-distribution \
  --origin-domain-name my-bucket.s3.amazonaws.com
```

64.4 Best Practices

- Set appropriate TTL for caching objects.
- Use Lambda@Edge for request/response modification.
- Enable logging for security and analytics.

64.5 Use Case

Deliver static website hosted on S3 globally with low latency and HTTPS.

65 CloudFront Caching Strategies

- Cache static content for long durations.
- Use Cache-Control headers for dynamic content.
- Invalidate cache for updates using CLI or console:

```
$ aws cloudfront create-invalidation \
  --distribution-id EDFDVBD632BHDS5 \
  --paths "/index.html" "/css/*"
```

66 Amazon Route 53: DNS Service

66.1 Introduction

Route 53 is a scalable DNS service with routing policies, health checks, and domain registration.

66.2 Key Features

- Latency-based routing
- Weighted routing
- Failover routing
- Health checks and alarms

66.3 CLI Examples

Create Hosted Zone:

```
$ aws route53 create-hosted-zone \
  --name example.com \
  --caller-reference "unique123"
```

67 Route 53 Health Checks

- Monitor endpoint health and automatically failover.
- CLI to create health check:

```
$ aws route53 create-health-check \
  --caller-reference "hc1" \
  --health-check-config IPAddress=192.0.2.44,Port=80,Type=HTTP
```

68 Failover Routing

- Primary site receives traffic if healthy.
- Secondary site receives traffic if primary fails.
- Integrate with CloudWatch for automatic switching.

69 Elastic Load Balancing (ELB)

69.1 Introduction

ELB distributes incoming traffic across multiple targets for high availability.

69.2 Types

- Classic Load Balancer
- Application Load Balancer (ALB) – HTTP/HTTPS, path-based routing
- Network Load Balancer (NLB) – TCP/UDP, low latency

70 Application Load Balancer (ALB)

CLI to create ALB:

```
$ aws elbv2 create-load-balancer \
  --name DevOpsALB \
  --subnets subnet-123 subnet-456 \
  --security-groups sg-12345678 \
  --scheme internet-facing
```

70.1 Best Practices

- Use path-based routing for microservices.
- Enable WAF for security.
- Monitor target health with CloudWatch.

71 Network Load Balancer (NLB)

CLI to create NLB:

```
$ aws elbv2 create-load-balancer \
  --name DevOpsNLB \
  --type network \
  --subnets subnet-123 subnet-456
```

71.1 Best Practices

- Use for TCP/UDP traffic with extremely low latency.
- Combine with Auto Scaling groups for high availability.
- Monitor using CloudWatch metrics: ActiveFlowCount, NewFlowCount.

72 Summary

- Use SQS + SNS + EventBridge for event-driven applications.
- CloudFront accelerates global content delivery.
- Route 53 ensures DNS reliability with health checks and failover.
- ELB (ALB/NLB) distributes traffic efficiently, ensuring high availability.

73 Auto Scaling Groups (ASG)

73.1 Introduction

Auto Scaling Groups automatically adjust the number of EC2 instances based on demand, ensuring high availability, cost efficiency, and scalability.

73.2 Key Features

- Automatic scaling up/down based on metrics.
- Integration with ELB to distribute traffic.
- Health checks and instance replacement.
- Launch templates or configurations for standardization.

73.3 CLI Commands

Create Launch Configuration:

```
$ aws autoscaling create-launch-configuration \
  --launch-configuration-name DevOpsLaunchConfig \
  --image-id ami-12345678 \
  --instance-type t2.micro \
  --key-name DevOpsKeyPair \
  --security-groups sg-12345678
```

Create Auto Scaling Group:

```
$ aws autoscaling create-auto-scaling-group \
  --auto-scaling-group-name DevOpsASG \
  --launch-configuration-name DevOpsLaunchConfig \
  --min-size 1 --max-size 5 \
  --desired-capacity 2 \
  --vpc-zone-identifier subnet-123,subnet-456
```

73.4 Metrics Policies

- Scale out if CPU ¿ 70%.
- Scale in if CPU ¡ 20%.
- Monitor metrics via CloudWatch.

73.5 Best Practices

- Use Launch Templates over configurations for flexibility.
- Combine ASG with ELB for health-aware routing.
- Monitor scaling activities and set alarms for unusual behavior.

73.6 Use Case

Web application receives variable traffic. ASG ensures 2–5 EC2 instances run automatically based on load.

73.7 Interview Tips

Explain differences between Launch Configuration and Launch Template, metrics-based policies, and health check integration.

74 AWS Secrets Manager

74.1 Introduction

Secrets Manager securely stores, rotates, and manages secrets such as database credentials, API keys, and passwords.

74.2 Key Features

- Automatic secret rotation.
- Integrated with Lambda for custom rotation logic.
- Fine-grained access control using IAM.
- Secure retrieval via AWS SDK/CLI.

74.3 CLI Commands

Create Secret:

```
$ aws secretsmanager create-secret \
  --name DevOpsDBSecret \
  --secret-string '{"username":"admin","password":"P@ssw0rd"}'
```

Retrieve Secret:

```
$ aws secretsmanager get-secret-value \
  --secret-id DevOpsDBSecret
```

74.4 Best Practices

- Rotate secrets automatically every 30–60 days.
- Use IAM policies to restrict access.
- Avoid hardcoding credentials in code or scripts.

74.5 Use Case

Lambda function retrieves DB credentials from Secrets Manager, eliminating the need to store them in code.

74.6 Interview Tips

Explain difference between Secrets Manager and Parameter Store, and how rotation works.

75 AWS Systems Manager Parameter Store

75.1 Introduction

Parameter Store provides centralized storage for configuration data and secrets with encryption support.

75.2 Key Features

- Store plaintext or encrypted values.
- Version control of parameters.
- Integrated with CloudFormation, Lambda, and EC2.

75.3 CLI Commands

Create Parameter:

```
$ aws ssm put-parameter \
  --name "/devops/db/username" \
  --value "admin" \
  --type "SecureString"
```

Retrieve Parameter:

```
$ aws ssm get-parameter \
  --name "/devops/db/username" \
  --with-decryption
```

75.4 Best Practices

- Use Parameter Store for configs and Secrets Manager for credentials.
- Encrypt sensitive data using KMS.
- Version control critical parameters.

75.5 Use Case

EC2 instances or Lambda functions read database credentials and configuration parameters securely from Parameter Store.

76 Encryption with AWS

76.1 Introduction

AWS provides encryption at rest (S3, EBS, RDS) and in transit (TLS/HTTPS).

76.2 Key Points

- Use KMS keys for centralized key management.
- Enable default encryption on S3 buckets and EBS volumes.
- Rotate keys periodically for security compliance.

76.3 CLI Example: Encrypt S3 Bucket

```
$ aws s3api put-bucket-encryption \
  --bucket my-devops-bucket \
  --server-side-encryption-configuration '{"Rules":[{"ApplyServerSideEncryptionByDefa
```

77 VPC Peering

77.1 Introduction

VPC Peering connects two VPCs privately using AWS network without internet gateway, VPN, or firewall.

77.2 CLI Commands

Create VPC Peering Connection:

```
$ aws ec2 create-vpc-peering-connection \
  --vpc-id vpc-11111111 \
  --peer-vpc-id vpc-22222222
```

77.3 Best Practices

- Use route tables to allow traffic between VPCs.
- Avoid overlapping CIDR ranges.
- Monitor peering connections for health and usage.

78 AWS Transit Gateway

78.1 Introduction

Transit Gateway connects multiple VPCs and on-prem networks through a single gateway for simplified network management.

78.2 CLI Commands

```
$ aws ec2 create-transit-gateway \
  --description "DevOpsTransitGateway" \
  --options "AmazonSideAsn=64512"
```

78.3 Use Case

Connect 10+ VPCs in multiple accounts for centralized routing and security policies.

79 VPC Security: Security Groups NACLs

- Security Groups: Virtual firewall for instances, stateful.
- NACLs: Network-level firewall, stateless, controls subnet traffic.
- Combine Security Groups + NACLs for defense-in-depth.

79.1 CLI Example: Security Group

```
$ aws ec2 create-security-group \
  --group-name DevOpsSG \
  --description "Security group for DevOps" \
  --vpc-id vpc-11111111
```

80 Elastic Beanstalk

80.1 Introduction

Elastic Beanstalk automates application deployment, capacity provisioning, load balancing, scaling, and monitoring.

80.2 CLI Example: Deploy App

```
$ eb init DevOpsApp --platform python-3.9 --region us-east-1
$ eb create DevOpsApp-env
```

80.3 Best Practices

- Use environment variables for secrets/configs.
- Monitor health via Beanstalk console and CloudWatch.
- Version control deployments to rollback if needed.

81 Application Deployment in Beanstalk

- Upload code package (.zip or .war).
- Auto-deploy via CLI or console.
- Integration with CI/CD pipelines using CodePipeline.

81.1 CLI Example: Deploy New Version

```
$ eb deploy
```

82 Monitoring Logs

- Monitor application health and instance metrics.
- Configure alarms in CloudWatch for CPU, latency, errors.
- Enable enhanced logging for troubleshooting.

82.1 Use Case

Deploy Python web app, auto-scale based on traffic, monitor logs and metrics for health and errors.

82.2 Interview Tips

Explain difference between Elastic Beanstalk and ECS, monitoring strategies, and CI/CD integration.

83 Amazon RDS: Relational Database Service

83.1 Introduction

Amazon RDS is a fully managed relational database service supporting multiple engines: MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server. It automates provisioning, patching, backup, and scaling.

83.2 Key Features

- Multi-AZ deployment for high availability.
- Automated backups and snapshots.
- Read replicas for horizontal scaling.
- Monitoring via CloudWatch.

83.3 CLI Commands

Create MySQL Database:

```
$ aws rds create-db-instance \
  --db-instance-identifier DevOpsRDS \
  --db-instance-class db.t3.micro \
  --engine mysql \
  --master-username admin \
  --master-user-password P@ssw0rd \
  --allocated-storage 20
```

Output:

```
{
  "DBInstance": {
    "DBInstanceIdentifier": "DevOpsRDS",
    "DBInstanceStatus": "creating",
    "Engine": "mysql",
    ...
  }
}
```

83.4 Best Practices

- Use Multi-AZ deployment for production.
- Enable automated backups and snapshot retention.
- Monitor CPU, storage, and connections via CloudWatch.

83.5 Use Case

Deploy MySQL database for web application with automated failover and backups.

83.6 Interview Tips

Explain difference between Multi-AZ and Read Replica, automated backup vs snapshot, and scaling strategies.

84 Amazon Aurora

84.1 Introduction

Aurora is a MySQL and PostgreSQL-compatible high-performance relational database with **up to 5x faster** throughput and managed replication.

84.2 Key Features

- Multi-AZ replication with 6-way durability.
- Auto-scaling storage.
- Global databases for cross-region replication.
- Backtrack to restore to specific point in time.

84.3 CLI Commands

Create Aurora Cluster:

```
$ aws rds create-db-cluster \
  --db-cluster-identifier DevOpsAurora \
  --engine aurora-mysql \
  --master-username admin \
  --master-user-password P@ssw0rd
```

84.4 Use Case

Global web application needing high availability and low latency.

85 RDS Backup and Restore

85.1 Automated Backups

Enable automatic backups for point-in-time recovery:

```
$ aws rds modify-db-instance \
  --db-instance-identifier DevOpsRDS \
  --backup-retention-period 7
```

85.2 Manual Snapshot

```
$ aws rds create-db-snapshot \
  --db-snapshot-identifier DevOpsRDS-Snapshot \
  --db-instance-identifier DevOpsRDS
```

85.3 Restore from Snapshot

```
$ aws rds restore-db-instance-from-db-snapshot \
  --db-instance-identifier RestoredDB \
  --db-snapshot-identifier DevOpsRDS-Snapshot
```

86 Multi-AZ Deployment

86.1 Introduction

Multi-AZ RDS ensures high availability by replicating instances synchronously across availability zones.

86.2 CLI Commands

Enable Multi-AZ:

```
$ aws rds modify-db-instance \
  --db-instance-identifier DevOpsRDS \
  --multi-az \
  --apply-immediately
```

86.3 Best Practices

- Use for production workloads.
- Monitor failover events in CloudWatch.
- Combine with automated backups.

87 Read Replicas

87.1 Introduction

Read replicas improve read scalability and offload queries from primary database.

87.2 CLI Commands

```
$ aws rds create-db-instance-read-replica \
  --db-instance-identifier DevOpsRDS-Replica \
  --source-db-instance-identifier DevOpsRDS
```

87.3 Use Case

High-traffic web applications require multiple read replicas for reporting or analytics queries.

88 Amazon DynamoDB

88.1 Introduction

DynamoDB is a fully managed NoSQL database with single-digit millisecond latency.

88.2 Key Features

- Serverless with auto-scaling.
- Global tables for cross-region replication.
- Integrated with Lambda and API Gateway.

88.3 CLI Commands

Create Table:

```
$ aws dynamodb create-table \
  --table-name DevOpsTable \
  --attribute-definitions AttributeName=ID,AttributeType=S \
  --key-schema AttributeName=ID,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

88.4 CRUD Operations

Insert Item:

```
$ aws dynamodb put-item \
  --table-name DevOpsTable \
  --item '{"ID":{"S":"123"},"Name":{"S":"TestUser"}}'
```

Query Item:

```
$ aws dynamodb get-item \
  --table-name DevOpsTable \
  --key '{"ID":{"S":"123"}}'
```

89 Amazon Redshift

89.1 Introduction

Redshift is a fully managed data warehouse for analytics workloads.

89.2 Key Features

- Columnar storage for fast queries.
- Integrates with S3, Glue, QuickSight.
- Automated snapshots and scaling.

89.3 CLI Commands

Create Cluster:

```
$ aws redshift create-cluster \
  --cluster-identifier DevOpsRedshift \
  --node-type dc2.large \
  --master-username admin \
  --master-user-password P@ssw0rd \
  --number-of-nodes 2
```

89.4 Connect and Query

- Connect using SQL clients via JDBC/ODBC.
- Load data from S3 using COPY command.

90 AWS Glue

90.1 Introduction

Glue is a serverless ETL service that discovers, prepares, and transforms data for analytics.

90.2 Key Features

- Automatic schema discovery with Glue Crawlers.
- ETL jobs using Python or Spark.
- Integration with S3, Redshift, DynamoDB.

90.3 CLI Commands

Create Glue Crawler:

```
$ aws glue create-crawler \
  --name DevOpsCrawler \
  --role AWSGlueServiceRole \
  --database-name devopsdb \
  --targets S3Targets=[{Path="s3://my-bucket/"}]
```

90.4 Create and Run ETL Job

CLI Example:

```
$ aws glue create-job \
  --name DevOpsETLJob \
  --role AWSGlueServiceRole \
  --command Name=glueetl,ScriptLocation=s3://scripts/etl_script.py \
  --max-capacity 2
```

Run Job:

```
$ aws glue start-job-run --job-name DevOpsETLJob
```

90.5 Use Case

ETL pipeline processes raw S3 logs into structured data in Redshift for analytics.

90.6 Best Practices

- Use version-controlled scripts for reproducibility.
- Monitor job logs and failures in CloudWatch.
- Partition data in S3 for optimized processing.

90.7 Interview Tips

Explain the difference between glue and lambda, ETL concepts, and integration with Redshift/S3.

91 AWS CloudFormation: Infrastructure as Code

91.1 Introduction

CloudFormation allows you to define AWS resources using **JSON or YAML templates**, enabling automated, repeatable infrastructure deployment.

91.2 Key Features

- Create, update, and delete entire stacks.
- Version-controlled infrastructure.
- Supports almost all AWS resources.
- Rollback on failure to maintain stability.

91.3 CLI Commands

Create Stack:

```
$ aws cloudformation create-stack \
  --stack-name DevOpsStack \
  --template-body file://template.yaml \
  --parameters ParameterKey=InstanceType,ParameterValue=t2.micro
```

Output Example:

```
{
  "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/DevOpsStack/abc123"
}
```

Update Stack:

```
$ aws cloudformation update-stack \
  --stack-name DevOpsStack \
  --template-body file://template.yaml \
  --parameters ParameterKey=InstanceType,ParameterValue=t2.small
```

91.4 Best Practices

- Store templates in version control (Git).
- Modularize templates using nested stacks.
- Test in a dev environment before production deployment.

91.5 Use Case

Deploy multi-tier web applications with EC2, RDS, and S3 using a single CloudFormation template.

91.6 Interview Tips

Explain difference between CloudFormation and Terraform, benefits of IaC, rollback policies, and nested stacks.

92 AWS CDK: Cloud Development Kit

92.1 Introduction

CDK allows developers to define cloud infrastructure using **programming languages** like Python, TypeScript, or Java.

92.2 Key Features

- Strongly typed constructs for resources.
- Reusable components for multiple projects.
- Synthesizes templates for CloudFormation deployment.

92.3 CLI Commands

Initialize CDK App (Python):

```
$ cdk init app --language python
```

Add Resources:

```
from aws_cdk import aws_s3 as s3, core

bucket = s3.Bucket(self, "DevOpsBucket",
                   versioned=True,
                   removal_policy=core.RemovalPolicy.DESTROY)
```

Deploy CDK Stack:

```
$ cdk deploy
```

92.4 Best Practices

- Use version control for CDK code.
- Leverage constructs for reusable patterns.
- Test changes locally using CDK diff before deployment.

92.5 Use Case

Programmatically deploy S3, Lambda, and API Gateway resources with code-based constructs.

92.6 Interview Tips

Be ready to explain difference between CDK, CloudFormation, and Terraform, pros of programmatic IaC, and how to integrate with CI/CD pipelines.

93 Cost Optimization and Billing

93.1 Introduction

AWS cost management ensures efficient resource usage and reduces unnecessary spending.

93.2 Key Features

- AWS Budgets and Cost Explorer for tracking expenses.
- Reserved Instances, Spot Instances, and Savings Plans for savings.
- Tagging resources for cost allocation.
- Cost anomaly detection.

93.3 CLI Commands

Get Cost and Usage:

```
$ aws ce get-cost-and-usage \
  --time-period Start=2025-09-01,End=2025-09-30 \
  --granularity MONTHLY \
  --metrics "BlendedCost" "UsageQuantity"
```

Output Example:

```
{
  "ResultsByTime": [
    {
      "TimePeriod": {"Start": "2025-09-01","End": "2025-09-30"},
      "Total": {"BlendedCost": {"Amount": "120.50","Unit": "USD"}}
    }
  ]
}
```

93.4 Best Practices

- Use tagging to allocate costs to teams/projects.
- Implement Reserved Instances for predictable workloads.
- Utilize Spot Instances for short-lived or batch workloads.
- Continuously monitor usage with Cost Explorer and budgets.

93.5 Use Case

Automate monthly cost reports, enforce budgets, and reduce unused resource costs using tagging.

93.6 Interview Tips

Explain differences between On-Demand, Reserved, and Spot Instances, and describe cost-saving strategies in AWS.

94 AWS Security Best Practices and Compliance

94.1 Introduction

Security and compliance are critical in AWS DevOps, ensuring protection of resources and meeting regulatory requirements.

94.2 Key Practices

- IAM Policies: Apply least privilege and role-based access.
- Enable MFA for all privileged users.
- Regularly rotate access keys and secrets.
- Monitor CloudTrail logs for auditing.
- Use GuardDuty, Security Hub, and Config Rules.

94.3 CLI Commands

Attach Policy to User:

```
$ aws iam attach-user-policy \
  --user-name DevOpsUser \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

Enable MFA for User:

```
$ aws iam enable-mfa-device \
  --user-name DevOpsUser \
  --serial-number arn:aws:iam::123456:mfa/DevOpsUser \
  --authentication-code1 123456 \
  --authentication-code2 654321
```

94.4 Best Practices

- Use IAM roles instead of long-lived credentials.
- Enable logging and monitoring for all accounts.
- Regularly audit policies and remove unused permissions.

94.5 Use Case

Maintain compliance in multi-account AWS environments by enforcing MFA, least privilege, and CloudTrail monitoring.

94.6 Interview Tips

Explain IAM policy types, best practices for DevOps security, and how to implement compliance checks using AWS Config and CloudTrail.

95 AWS IAM Policies: Access Control and Best Practices

95.1 Introduction

IAM Policies are **JSON documents** that define **permissions** for AWS resources. They are attached to **users, groups, or roles** to control access securely.

95.2 Types of IAM Policies

- **Managed Policies:** AWS-provided or customer-managed reusable policies.
- **Inline Policies:** Policies embedded directly into a single user, group, or role.
- **Permission Boundaries:** Upper limit of permissions for IAM entities.

95.3 Key Concepts

- **Effect:** Allow or Deny permissions.
- **Action:** Specifies which actions (e.g., s3:PutObject, ec2:StartInstances) are allowed or denied.
- **Resource:** Specifies which resources the actions apply to.
- **Condition:** Optional JSON object to apply conditional access.

95.4 CLI Commands Examples

Attach Managed Policy to User:

```
$ aws iam attach-user-policy \
  --user-name DevOpsUser \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

Output Example:

```
{
    "ResponseMetadata": {
        "RequestId": "abc123-example",
        "HTTPStatusCode": 200,
        ...
    }
}
```

Create Custom Managed Policy:

```
$ aws iam create-policy \
  --policy-name DevOpsS3ReadOnly \
  --policy-document file://s3-readonly-policy.json
```

s3-readonly-policy.json:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject","s3:ListBucket"],
      "Resource": ["arn:aws:s3:::my-bucket","arn:aws:s3:::my-bucket/*"]
    }
  ]
}
```

Attach Custom Policy to Group:

```
$ aws iam attach-group-policy \
  --group-name DevOpsGroup \
  --policy-arn arn:aws:iam::123456789012:policy/DevOpsS3ReadOnly
```

95.5 Best Practices

- Follow the **Principle of Least Privilege**: only grant necessary permissions.
- Prefer **Managed Policies** for reusability and simplicity.
- Use **Permission Boundaries** for controlled delegation.
- Rotate and audit credentials regularly.
- Monitor IAM actions with **CloudTrail**.

95.6 Use Cases

- Allow developers read-only access to S3 buckets.
- Assign Lambda functions roles with limited permissions.
- Separate production vs development access using groups and policies.

95.7 Interview Tips

- Explain **Managed vs Inline policies**.
- Describe **Policy evaluation logic** (Allow ¿ Deny ¿ Default Deny).
- Discuss **Permission boundaries and their advantages**.
- Demonstrate creating and attaching a custom IAM policy.

96 Top 10 Real-Time AWS DevOps Interview Scenarios with Solutions

96.1 Scenario 1: Automate EC2 Deployment using CLI

Problem: Deploy a new EC2 instance in a specific VPC with a security group and key pair.

Solution:

```
$ aws ec2 run-instances \
  --image-id ami-0abcdef1234567890 \
  --count 1 \
  --instance-type t2.micro \
  --key-name DevOpsKey \
  --security-group-ids sg-0123456789abcdef0 \
  --subnet-id subnet-0abc1234def567890
```

Output Example:

```
{
  "Instances": [
    {
      "InstanceId": "i-0123456789abcdef0",
      "State": {"Name": "pending"},
      ...
    }
  ]
}
```

Notes:

- Always check availability of AMI in the chosen region.
- Security groups define port access (e.g., 22 for SSH, 80 for HTTP).

—

96.2 Scenario 2: Setup Auto Scaling for Web Application

Problem: Automatically scale EC2 instances based on CPU utilization.

Solution:

```
$ aws autoscaling create-auto-scaling-group \
  --auto-scaling-group-name WebAppASG \
  --launch-configuration-name WebAppLC \
  --min-size 1 \
  --max-size 5 \
  --vpc-zone-identifier subnet-0abc1234
```

Best Practices:

- Use CloudWatch alarms to trigger scaling.
- Define both min and max limits to avoid over-provisioning.

—

96.3 Scenario 3: Implement S3 Bucket Lifecycle Policy

Problem: Move old objects to Glacier for cost optimization.

Solution:

```
$ aws s3api put-bucket-lifecycle-configuration \
  --bucket my-bucket \
  --lifecycle-configuration file://lifecycle.json
```

lifecycle.json Example:

```
{
  "Rules": [
    {
      "ID": "ArchiveOldObjects",
      "Prefix": "",
      "Status": "Enabled",
      "Transitions": [
        {
          "Days": 30,
          "StorageClass": "GLACIER"
        }
      ]
    }
  ]
}
```

Notes: Reduces storage costs for infrequently accessed data.

—

96.4 Scenario 4: Create IAM Role for Lambda with S3 Access

Problem: Lambda function needs to read objects from an S3 bucket.

Solution:

```
$ aws iam create-role --role-name LambdaS3Role \
  --assume-role-policy-document file://trust-policy.json
$ aws iam attach-role-policy \
  --role-name LambdaS3Role \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Notes: Always follow least privilege principle.

—

96.5 Scenario 5: Deploy Serverless Application using CloudFormation

Problem: Automate deployment of Lambda + API Gateway.

Solution:

```
$ aws cloudformation deploy \
  --stack-name ServerlessApp \
  --template-file serverless-template.yaml \
  --capabilities CAPABILITY_NAMED_IAM
```

Best Practices:

- Use parameters for environment-specific configuration.
- Test stacks in dev account before production.

—

96.6 Scenario 6: Monitor EC2 and Set Alarm

Problem: Alert when CPU exceeds 70%.

Solution:

```
$ aws cloudwatch put-metric-alarm \
  --alarm-name HighCPU \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --threshold 70 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-0123456789abcdef0 \
  --evaluation-periods 2 \
  --alarm-actions arn:aws:sns:us-east-1:123456789012:NotifyMe
```

Notes: Helps proactive resource scaling and troubleshooting.

—

96.7 Scenario 7: Implement CloudFront with S3 Origin

Problem: Serve static website content with low latency globally.

Solution:

```
$ aws cloudfront create-distribution \
  --origin-domain-name my-bucket.s3.amazonaws.com
```

Best Practices:

- Use caching behaviors to reduce origin load.
- Enable HTTPS for secure content delivery.

—

96.8 Scenario 8: Deploy RDS with Multi-AZ for High Availability

Problem: Ensure database uptime with Multi-AZ deployment.

Solution:

```
$ aws rds create-db-instance \
  --db-instance-identifier mydb \
  --db-instance-class db.t3.micro \
  --engine mysql \
  --allocated-storage 20 \
  --master-username admin \
  --master-user-password password \
  --multi-az
```

Notes: Automatic failover ensures minimal downtime.

—

96.9 Scenario 9: Implement Cost Optimization with Spot Instances

Problem: Reduce EC2 cost for batch jobs.

Solution:

```
$ aws ec2 request-spot-instances \
  --instance-count 2 \
  --type "one-time" \
  --launch-specification file://spot-spec.json
```

Notes: Use Spot for non-critical workloads; combine with Auto Scaling.

—

96.10 Scenario 10: CI/CD Pipeline using CodePipeline and CodeBuild

Problem: Automate deployment from GitHub to EC2.

Solution:

```
# Create Pipeline
$ aws codepipeline create-pipeline --cli-input-json file://pipeline.json
# Trigger Build
$ aws codebuild start-build --project-name DevOpsProject
```

Best Practices:

- Integrate testing before deployment.
- Use versioning for rollback.

—

Author's Note

Thank You for reading this AWS DevOps Digital Guide!

Author: Sainath Shivaji Mitalakar
Edition: 2025
xcolor

"Dream, dream, dream. Dreams transform into thoughts and thoughts result in action."
– APJ Abdul Kalam