

AWS Zero to Hero - Notes 01

Introduction, Virtualization, Cloud Computing & IAM Fundamentals

Table of Contents

1. [Course Introduction](#)
 2. [Virtualization Concepts](#)
 3. [Cloud Computing Fundamentals](#)
 4. [AWS Overview](#)
 5. [IAM \(Identity & Access Management\)](#)
 6. [MFA \(Multi-Factor Authentication\)](#)
 7. [Important Q&A](#)
-

Course Introduction

What You'll Learn:

- **IT Fundamentals** - Basic understanding required for cloud
- **Website Development & Deployment** - How websites work in real world
- **Networking Basics** - IP, Ports, Routing, Internet Gateway
- **Cloud Computing** - What it is and how it works
- **AWS Services** - Hands-on with various AWS services
- **Infrastructure as Code (IaC)** - Terraform basics
- **Containers** - Docker and Kubernetes (EKS)
- **Full Stack Development** - Complete deployment on cloud

Who Should Watch:

- IT professionals wanting to learn cloud
- Developers wanting AWS knowledge

- Anyone in any IT role
 - Support engineers
 - Those preparing for AWS certifications
-

Virtualization Concepts

◆ What is Virtualization?

Definition: Virtualization is the process of creating virtual (software-based) versions of physical hardware resources like servers, storage, networks, and operating systems.

In-Depth Explanation:

Think of virtualization like a building with multiple apartments. The building (physical server) has fixed resources - electricity, water, space. Instead of one family using the entire building, multiple families (virtual machines) share these resources efficiently, each with their own private space.

Real-World Analogy:

- **Without Virtualization:** One company buys 10 servers, each running at 10% capacity = 90% waste
- **With Virtualization:** One powerful server runs 10 VMs, each getting resources they need = 90% efficiency

Technical Deep Dive:

Virtualization works by inserting a **Hypervisor** (Virtual Machine Monitor) between hardware and operating systems. The hypervisor:

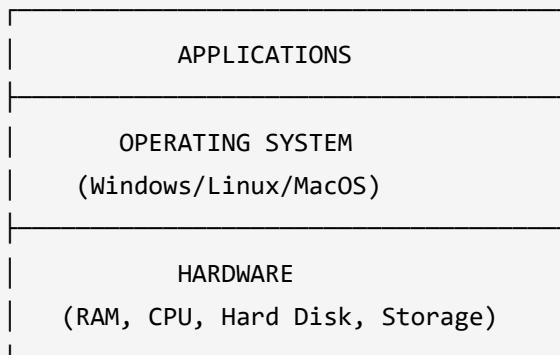
1. Abstracts physical hardware into virtual resources
2. Allocates CPU, memory, storage to each VM
3. Ensures isolation between VMs
4. Manages resource scheduling

Types of Hypervisors:

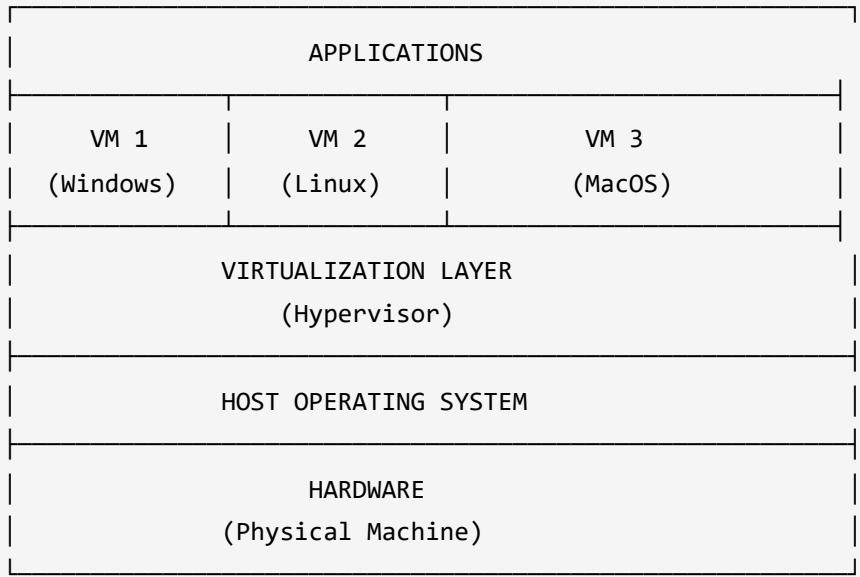
Type	Description	Examples	Use Case
Type 1 (Bare Metal)	Runs directly on hardware	VMware ESXi, Microsoft Hyper-V, Xen	Enterprise data centers

Type	Description	Examples	Use Case
Type 2 (Hosted)	Runs on top of OS	VirtualBox, VMware Workstation	Development, testing

Physical Machine Components:



How Virtualization Works:



Key Points:

- **Hypervisor** - Software that creates and manages virtual machines
- Multiple OS can run on single physical hardware
- Each VM gets allocated resources (RAM, CPU, Storage)
- VMs are isolated from each other

Benefits of Virtualization:

1. **Resource Optimization** - Better utilization of hardware (from 10-15% to 70-80%)

2. **Cost Savings** - Less physical hardware, reduced power and cooling costs
3. **Flexibility** - Run different OS on same hardware simultaneously
4. **Testing** - Test applications in different environments safely
5. **Isolation** - Problems in one VM don't affect others
6. **Disaster Recovery** - Easy backup and restoration of entire VMs
7. **Rapid Deployment** - Create new servers in minutes, not weeks

Virtualization vs Containerization:

Aspect	Virtualization	Containerization
Isolation	Complete OS isolation	Process-level isolation
Resource Usage	Heavy (full OS per VM)	Lightweight (shared OS kernel)
Boot Time	Minutes	Seconds
Use Case	Different OS environments	Microservices, same OS
Example	VMware, VirtualBox	Docker, Kubernetes

Cloud Computing Fundamentals

◆ What is Cloud Computing?

Definition: Cloud computing is the delivery of computing services (servers, storage, databases, networking, software) over the internet ("the cloud") on a pay-as-you-go basis.

In-Depth Explanation:

Cloud computing is like using electricity from the power grid instead of running your own generator:

- **Before Cloud:** Companies bought servers, hired IT staff, maintained data centers, planned capacity years in advance
- **With Cloud:** Rent computing power by the hour/minute, scale instantly, pay only for what you use

The Five Essential Characteristics of Cloud (NIST Definition):

1. **On-Demand Self-Service:** Provision resources automatically without human interaction
2. **Broad Network Access:** Available over the network from any device
3. **Resource Pooling:** Provider's resources serve multiple customers (multi-tenancy)
4. **Rapid Elasticity:** Scale up/down quickly based on demand
5. **Measured Service:** Pay only for what you consume (metering)

Cloud Deployment Models:

Model	Description	Use Case	Example
Public Cloud	Resources owned by third-party	Startups, variable workloads	AWS, Azure, GCP
Private Cloud	Dedicated to single organization	Banks, healthcare, government	On-premise OpenStack
Hybrid Cloud	Mix of public and private	Sensitive + variable workloads	AWS Outposts
Multi-Cloud	Using multiple cloud providers	Avoid vendor lock-in	AWS + Azure together

Traditional IT vs Cloud Computing:

Traditional IT	Cloud Computing
Buy hardware upfront	Pay as you go
Manage your own servers	Provider manages infrastructure
Fixed capacity	Scalable capacity
High CAPEX	OPEX model
Long setup time	Quick provisioning

Types of Cloud Services:

1. IaaS (Infrastructure as a Service)

- **What:** Raw computing resources (VMs, Storage, Networks)
- **Example:** AWS EC2, Azure VMs
- **You Manage:** OS, Applications, Data
- **Provider Manages:** Hardware, Networking, Storage

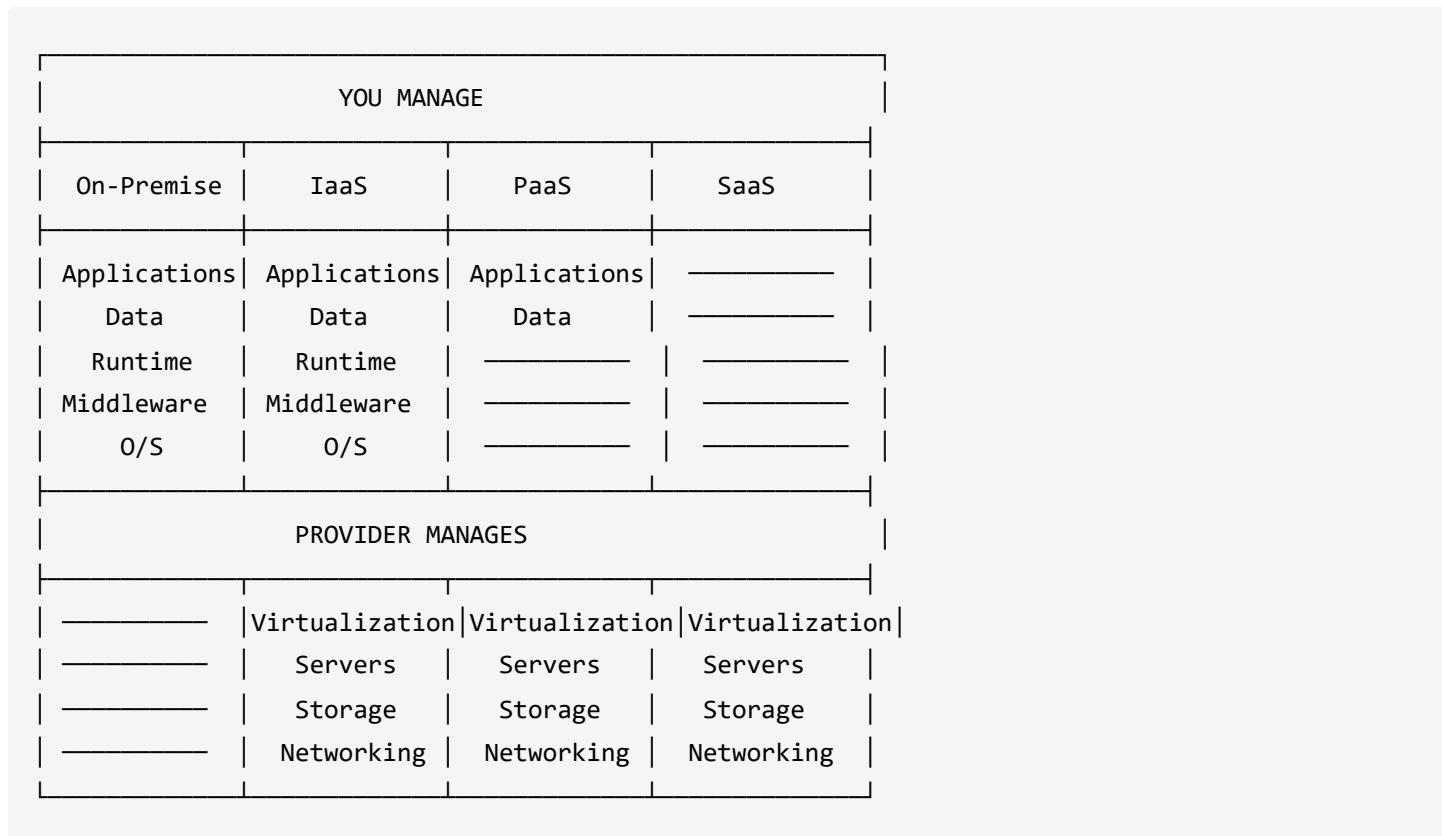
2. PaaS (Platform as a Service)

- **What:** Platform for building applications
- **Example:** AWS Elastic Beanstalk, Heroku
- **You Manage:** Applications, Data
- **Provider Manages:** OS, Runtime, Hardware

3. SaaS (Software as a Service)

- **What:** Complete software delivered via internet
- **Example:** Gmail, Salesforce, Office 365
- **You Manage:** Just use the software
- **Provider Manages:** Everything

Cloud Service Model Diagram:



Types of Scaling:

Vertical Scaling (Scale Up)

- Increase resources of existing server
- Add more RAM, CPU, Storage
- Has limitations (hardware max)
- **Example:** Upgrading from t2.micro to t2.large

Horizontal Scaling (Scale Out)

- Add more servers/instances
- Distribute load across multiple servers
- Preferred for high availability
- **Example:** Adding more EC2 instances behind a load balancer

CapEx vs OpEx Model:

Aspect	CapEx (Traditional)	OpEx (Cloud)
Payment	Large upfront investment	Pay monthly/hourly
Ownership	You own the hardware	Rent resources
Risk	High (capacity planning)	Low (adjust as needed)
Flexibility	Limited	High
Tax Treatment	Depreciation over years	Operating expense

AWS Overview

◆ What is AWS?

Amazon Web Services (AWS) is the world's most comprehensive and widely adopted cloud platform, offering over 200 fully featured services from data centers globally.

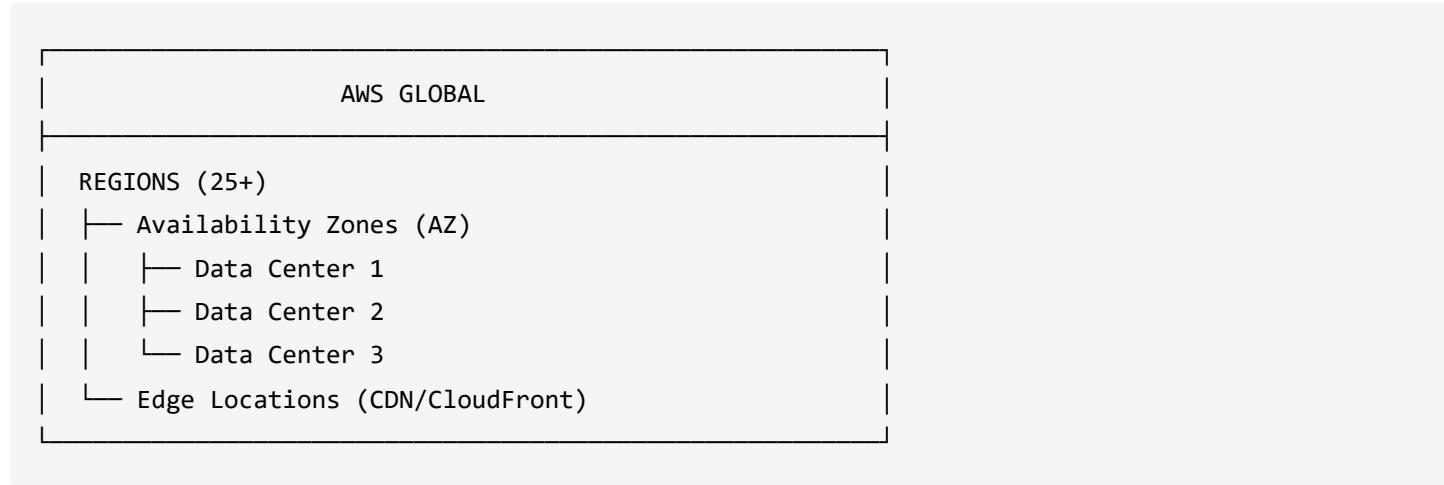
AWS History:

- **2002:** Amazon began offering web services
- **2004:** SQS (Simple Queue Service) launched
- **2006:** AWS officially launched with S3 and EC2
- **2024+:** Market leader with 32%+ market share

Why Companies Choose AWS:

1. **Market Leader** - Largest cloud provider with most services
2. **Global Infrastructure** - 33+ regions, 100+ availability zones
3. **200+ Services** - Compute, storage, ML, IoT, analytics
4. **Security** - Meets compliance standards (HIPAA, PCI, SOC)
5. **Cost Effective** - Pay-as-you-go with various discount options
6. **Innovation** - Constantly adding new services
7. **Community** - Large ecosystem, documentation, support

AWS Global Infrastructure:



Key Concepts:

- **Region:** Geographic area with multiple AZs
- **Availability Zone (AZ):** One or more data centers
- **Edge Location:** CDN endpoints for faster content delivery

AWS Free Tier:

- **12 Months Free:** EC2, S3, RDS (limited)
- **Always Free:** Lambda, DynamoDB (limited)
- **Trials:** Short-term free trials of services

IAM (Identity & Access Management)

◆ What is IAM?

IAM is AWS's identity and access management service that helps you securely control access to AWS resources. It answers two fundamental questions:

1. **Authentication:** WHO are you? (Identity)
2. **Authorization:** WHAT can you do? (Permissions)

In-Depth Explanation:

Think of IAM as the security system of an office building:

- **Root Account** = Building owner (has all keys, shouldn't be used daily)
- **IAM Users** = Employees with ID badges
- **IAM Groups** = Departments (Engineering, HR, Finance)

- **IAM Policies** = Access rules (who can enter which rooms)
- **IAM Roles** = Temporary visitor passes

Why IAM is Critical:

- **Security:** Prevent unauthorized access to resources
- **Compliance:** Audit trail of who did what
- **Principle of Least Privilege:** Give minimum permissions needed
- **Cost Control:** Prevent accidental resource creation

IAM Components:

1. Root Account

- Created when you first set up AWS account
- Has **complete access** to all AWS services
- **⚠ NEVER use for daily tasks**
- Enable MFA immediately
- Create IAM users instead

2. IAM Users

- Individual identities for people/applications
- Can have:
 - Console access (password)
 - Programmatic access (Access Keys)
- Principle of Least Privilege

3. IAM Groups

- Collection of users
- Assign permissions to group
- Users inherit group permissions
- Makes permission management easier

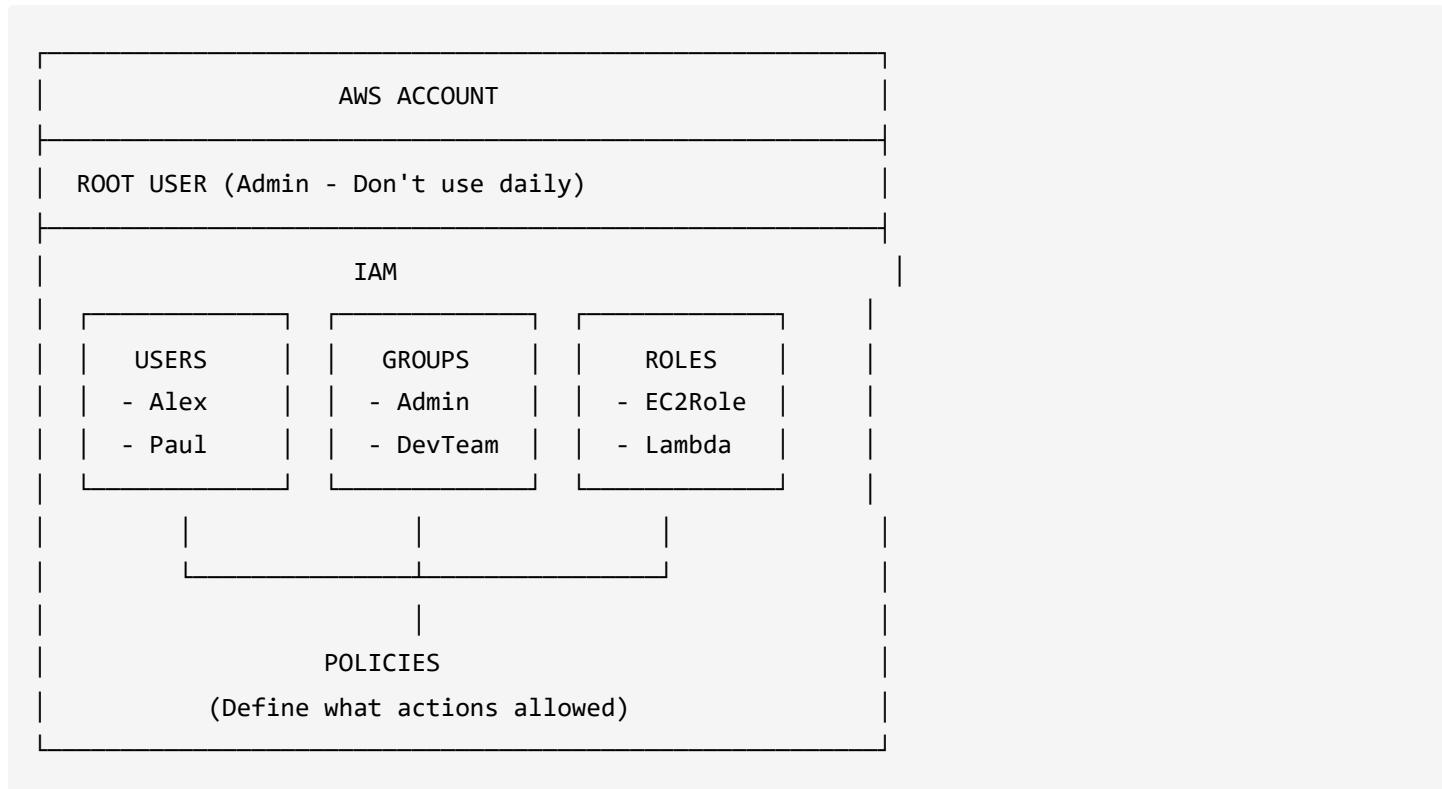
4. IAM Policies

- JSON documents defining permissions
- Attached to Users, Groups, or Roles
- Types:
 - AWS Managed Policies
 - Customer Managed Policies
 - Inline Policies

5. IAM Roles

- Temporary permissions
- For AWS services or external identities
- No permanent credentials
- Assumed when needed

IAM Architecture:



IAM Policy Example (JSON):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:Describe*",
        "ec2:StartInstances",
        "ec2:StopInstances"
      ],
      "Resource": "*"
    }
  ]
}
```

IAM Best Practices:

1. Enable MFA on root account
2. Create individual IAM users
3. Use groups to assign permissions
4. Grant least privilege
5. Use roles for applications
6. Rotate credentials regularly
7. Use strong password policy
8. Never share credentials

Creating IAM User - Steps:

1. Go to IAM Dashboard
2. Click "Users" → "Create User"
3. Enter username
4. Set permissions (Add to group or attach policy)
5. Set password policy
6. Enable console access if needed
7. Download credentials (CSV)

User vs Group Permissions:

Aspect	User Direct	Through Group
Management	Complex with many users	Easy - one change affects all
Best Practice	Avoid for teams	Recommended
Flexibility	Individual control	Standardized access

MFA (Multi-Factor Authentication)

◆ What is MFA?

Multi-Factor Authentication adds an extra layer of security beyond username and password.

MFA Factors:

1. **Something you know** - Password
2. **Something you have** - Phone/Token device
3. **Something you are** - Biometrics

MFA Options in AWS:

- **Virtual MFA** - Google Authenticator, Authy
- **Hardware MFA** - Physical token device
- **U2F Security Key** - YubiKey

Why Enable MFA?

- Protects against password compromise
- **Critical for root account**
- Recommended for all IAM users
- Industry best practice

Enabling MFA Steps:

1. Go to IAM → Users → Select user
 2. Security credentials tab
 3. Assign MFA device
 4. Follow setup wizard
 5. Enter two consecutive codes to verify
-

Important Q&A (Real Interview Questions)

Q1: What is the difference between Root User and IAM User?

Answer:

Root User	IAM User
Created with AWS account	Created by admin/root
Has complete, unrestricted access	Has limited, defined access
Cannot be restricted by policies	Can be restricted by policies
Only for billing and emergency	For daily administrative tasks
One per AWS account	Multiple allowed (up to 5000)
Should have MFA enabled	Should have MFA enabled

Interview Tip: Always mention that root user should NEVER be used for daily tasks.

Q2: What is the Principle of Least Privilege?

Answer: The principle of least privilege means giving users/applications only the minimum permissions they need to perform their job functions. This reduces the attack surface and limits damage from compromised credentials.

Example: A developer who only needs to read S3 buckets should NOT have `s3:*` (all S3 permissions) but only `s3:GetObject` and `s3>ListBucket`.

Q3: Explain the difference between IAM Users, Groups, Roles, and Policies.

Answer:

- **IAM User:** An identity representing a person or application with permanent credentials
- **IAM Group:** A collection of users; makes permission management easier
- **IAM Role:** A set of permissions that can be assumed temporarily by users, applications, or AWS services
- **IAM Policy:** A JSON document that defines permissions (what actions are allowed/denied on which resources)

Key Point: Users and Roles are identities. Policies are permissions. Groups are for organizing users.

Q4: What is the difference between IAM Role and IAM User?

Answer:

IAM User	IAM Role
Has permanent long-term credentials	Has temporary credentials
For people or applications	For AWS services or cross-account access
Has password and/or access keys	No permanent credentials
Directly assigned permissions	Permissions assumed when needed
Cannot be assumed	Can be assumed by users, services, or accounts

Use Case Example:

- EC2 instance needs to access S3 → Use IAM Role (not access keys)
- Developer needs AWS CLI access → Use IAM User

Q5: What happens when you attach multiple policies to a user?

Answer: AWS uses a **union** of all permissions. If any policy allows an action, it's allowed (unless explicitly denied). The evaluation order is:

1. Explicit Deny (always wins)
2. Explicit Allow
3. Implicit Deny (default)

Q6: What is IAM Policy Structure?

Answer: An IAM policy has these elements:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UniqueIdentifier",
      "Effect": "Allow/Deny",
      "Action": ["service:action"],
      "Resource": "arn:aws:service:region:account:resource",
      "Condition": {"operator": {"key": "value"}}
    }
  ]
}
```

- **Version:** Always use "2012-10-17"
- **Effect:** Allow or Deny
- **Action:** What API calls are permitted
- **Resource:** Which AWS resources are affected
- **Condition:** When the policy applies (optional)

Q7: What is AWS STS and how does it relate to IAM Roles?

Answer: AWS Security Token Service (STS) is the service that issues temporary credentials when a role is assumed. When you call `AssumeRole`, STS returns:

- Temporary Access Key ID
- Temporary Secret Access Key
- Session Token
- Expiration time (default 1 hour, max 12 hours)

Q8: How would you grant an EC2 instance access to S3?

Answer:

1. Best Practice (Use IAM Role):

- Create an IAM role with S3 permissions
- Attach the role to the EC2 instance
- The instance automatically gets temporary credentials

2. NOT Recommended (Access Keys):

- Never store access keys on EC2 instances
- If instance is compromised, keys are exposed

Q9: What is the difference between AWS Managed Policy and Customer Managed Policy?**Answer:**

AWS Managed Policy	Customer Managed Policy
Created and managed by AWS	Created and managed by you
Cannot be modified	Fully customizable
Automatically updated by AWS	You must update manually
Good for common use cases	For specific requirements
Example: AmazonS3ReadOnlyAccess	Custom policy for your application

Q10: What is Cross-Account Access and how do you implement it?

Answer: Cross-account access allows resources in one AWS account to access resources in another account.

Implementation:

1. Account B creates an IAM role with trust policy allowing Account A
2. Attach permissions policy to the role
3. Account A users assume the role using `sts:AssumeRole`

Q11: What is MFA and why is it important?

Answer: Multi-Factor Authentication adds an extra layer of security requiring:

1. Something you know (password)
2. Something you have (MFA device/phone)

Why Important:

- Protects against password compromise
- Required for compliance (PCI-DSS, HIPAA)
- Especially critical for root and admin accounts
- Can be required for specific actions (e.g., deleting resources)

Q12: What is Identity Federation in AWS?

Answer: Identity Federation allows users from external identity providers to access AWS resources without creating IAM users. Types:

- **SAML 2.0:** Enterprise identity providers (Active Directory, Okta)
- **Web Identity:** Social logins (Google, Facebook) via Cognito
- **Custom Identity Broker:** For non-SAML enterprise systems

Q13: What are IAM Access Keys and when should you use them?

Answer: Access keys consist of Access Key ID + Secret Access Key, used for programmatic access (CLI, SDK, API).

Best Practices:

- Never embed in code
- Rotate regularly (90 days recommended)
- Use IAM roles instead when possible
- Delete unused access keys
- Never share or commit to version control

Q14: Explain the IAM Policy Evaluation Logic.

Answer: AWS evaluates policies in this order:

1. **Explicit Deny:** If any policy explicitly denies → DENIED
2. **Organization SCP:** If SCP denies → DENIED
3. **Resource-based Policy:** Check if resource policy allows
4. **Identity-based Policy:** Check user/role policies
5. **IAM Permission Boundary:** If defined, must also allow
6. **Session Policy:** For role sessions, must also allow
7. **Default:** If nothing allows → DENIED (implicit deny)

Q15: What is the difference between Resource-based Policy and Identity-based Policy?

Answer:

Identity-based Policy	Resource-based Policy
Attached to IAM users, groups, roles	Attached to AWS resources
Specifies what actions identity can perform	Specifies who can access the resource
Example: IAM user policy	Example: S3 bucket policy
Must use ARN for resources	Must specify Principal

Key Takeaways

1. **Virtualization** enables running multiple OS on single hardware
 2. **Cloud Computing** provides on-demand IT resources over internet
 3. **AWS** is the leading cloud provider with 200+ services
 4. **IAM** controls who can access what in AWS
 5. **Root Account** should be protected with MFA and rarely used
 6. **Groups** simplify permission management for multiple users
 7. **Policies** define permissions in JSON format
 8. **MFA** adds critical security layer
-

Next Topics to Cover

- EC2 (Elastic Compute Cloud)
 - VPC (Virtual Private Cloud)
 - S3 (Simple Storage Service)
 - Networking in AWS
-

Notes created from AWS Zero to Hero Course - File 1

AWS Zero to Hero - Notes File 2

Topics: MFA Device Setup, AWS CLI, EC2 Instances & EBS Storage

Table of Contents

1. [MFA Device Setup](#)
 2. [AWS Access Methods](#)
 3. [AWS CLI Setup & Configuration](#)
 4. [EC2 Instances - Deep Dive](#)
 5. [Instance Types & Categories](#)
 6. [Purchasing Options](#)
 7. [EBS - Elastic Block Store](#)
 8. [Important Q&A](#)
-

1. MFA Device Setup

What is MFA?

Multi-Factor Authentication (MFA) adds an extra layer of security to your AWS account by requiring:

1. Something you know (password)
2. Something you have (MFA device)

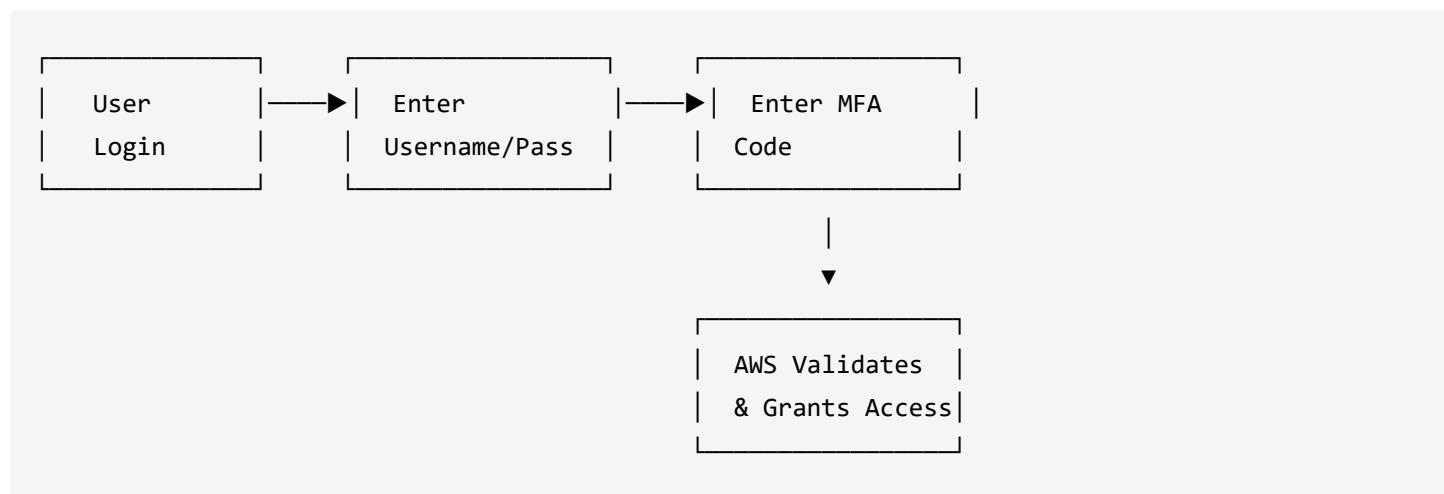
MFA Device Options

Option	Description	Use Case
Authenticator App	Google Authenticator, Authy	Most common, mobile-based
Hardware TOTP Token	Physical token device	High security environments
Security Keys	FIDO U2F keys	Enterprise security
SMS (Deprecated)	Text message codes	Not recommended

Setting Up MFA (Step-by-Step)

1. Go to IAM → Users → Select User → Security Credentials
2. Under MFA, click "Assign MFA device"
3. Give the device a name
4. Choose device type (Authenticator App recommended)
5. Install Google Authenticator on your mobile
6. Scan the QR code displayed
7. Enter two consecutive MFA codes (30 seconds apart)
8. Click "Add MFA"

Diagram: MFA Authentication Flow



2. AWS Access Methods

AWS provides **three ways** to access the platform:

Method 1: Management Console (Browser)

- GUI-based interface
- Good for beginners
- Manual operations
- Visual dashboard

Method 2: AWS CLI (Command Line Interface)

- Text-based commands
- Automation capable
- Scripting support
- Used by DevOps engineers

Method 3: SDKs & APIs

- Programmatic access
- Integration with applications
- Supports multiple languages (Python, Java, Node.js, etc.)
- Used for application development

Comparison Table

Feature	Console	CLI	SDK/API
Ease of Use	★★★★★	★★★☆☆	★★☆☆☆
Automation	X	✓	✓
Scripting	X	✓	✓
Bulk Operations	X	✓	✓
Learning Curve	Low	Medium	High

3. AWS CLI Setup & Configuration

Installation Steps

Windows:

```
# Download MSI installer from AWS website
# Or use winget
winget install Amazon.AWSCLI

# Verify installation
aws --version
```

Linux/Mac:

```
# Using curl
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

# Verify
aws --version
```

CLI Configuration

```
# Configure AWS CLI
aws configure

# You'll be prompted for:
# 1. AWS Access Key ID
# 2. AWS Secret Access Key
# 3. Default region (e.g., ap-south-1)
# 4. Default output format (json/yaml/table/text)
```

Important AWS CLI Commands

```
# IAM Commands
aws iam list-users          # List all IAM users
aws iam create-user --user-name alex # Create new user
aws iam delete-user --user-name alex # Delete user

# EC2 Commands
aws ec2 describe-instances      # List all instances
aws ec2 start-instances --instance-ids i-xxx
aws ec2 stop-instances --instance-ids i-xxx

# S3 Commands
aws s3 ls                      # List all buckets
aws s3 cp file.txt s3://bucket-name/ # Upload file
```

Access Key Best Practices

⚠ SECURITY WARNINGS:

- Never share access keys
- Never commit keys to Git
- Rotate keys regularly
- Use IAM roles when possible
- Delete unused access keys

4. EC2 Instances - Deep Dive

What is EC2?

Elastic Compute Cloud (EC2) is AWS's core compute service that provides resizable virtual servers in the cloud.

In-Depth Explanation:

Think of EC2 as renting a computer:

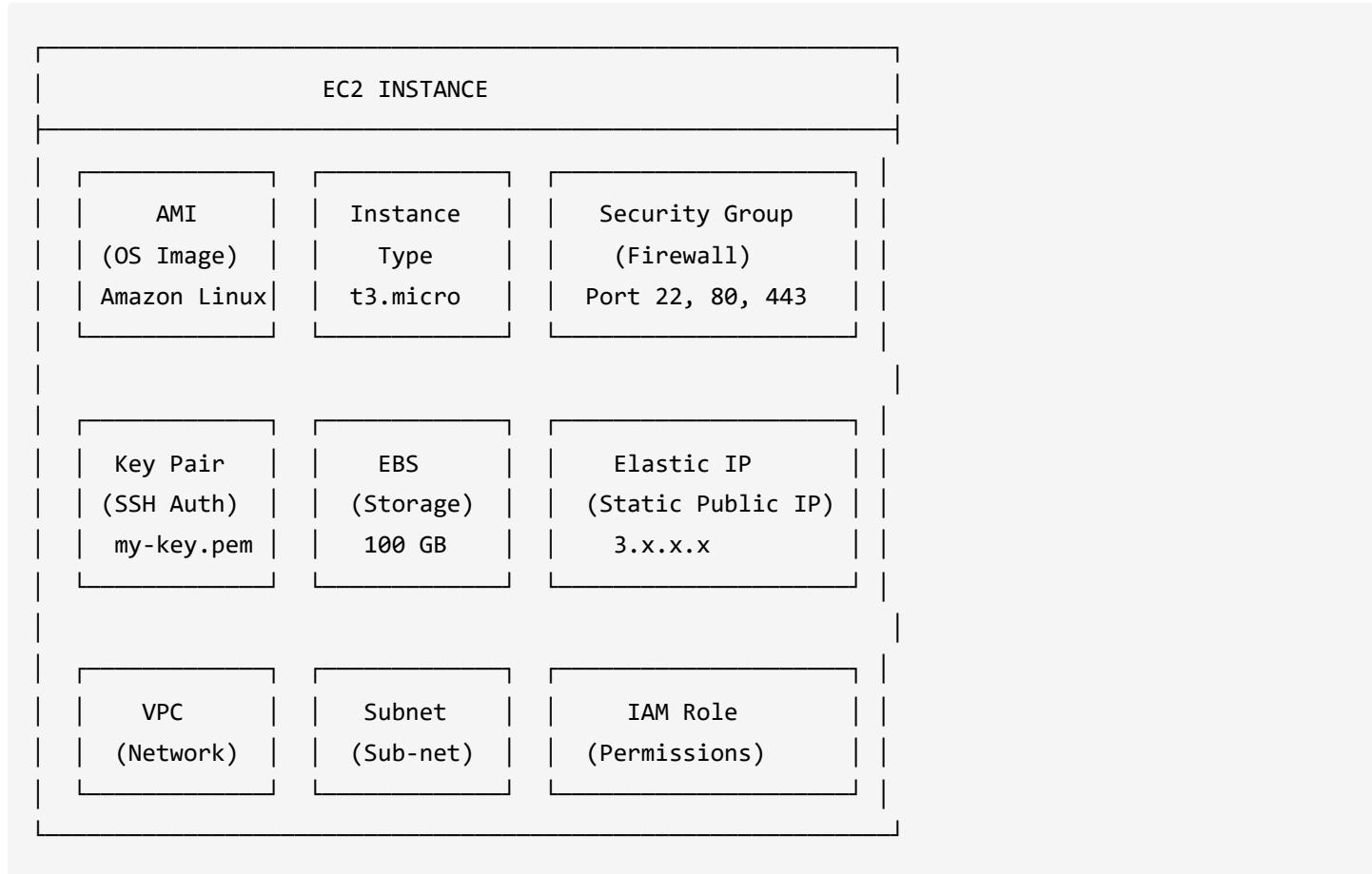
- **Before EC2:** Buy servers (\$10,000+), wait weeks for delivery, manage hardware, pay even when idle
- **With EC2:** Launch a server in minutes, pay by the hour/second, scale up/down instantly, terminate when not needed

Why "Elastic"?

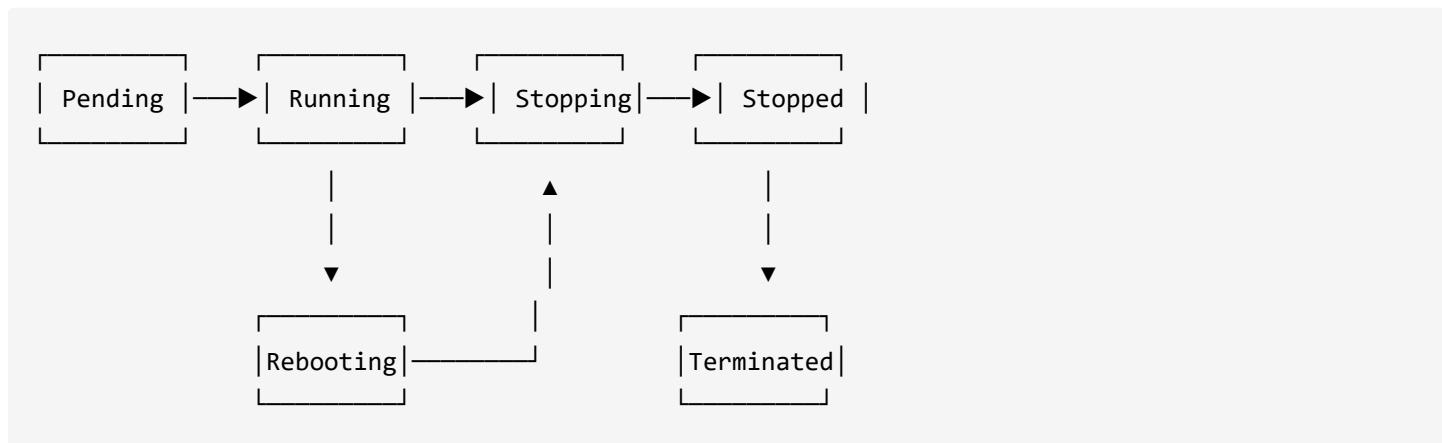
The term "Elastic" means you can:

1. **Scale vertically:** Change instance type (t3.micro → t3.large)
2. **Scale horizontally:** Add/remove instances based on demand
3. **Pay elastically:** Only pay for what you use

EC2 Architecture:



EC2 Instance Lifecycle



Key Components

1. **AMI (Amazon Machine Image)** - OS template
2. **Instance Type** - Hardware configuration
3. **Security Groups** - Virtual firewall
4. **Key Pairs** - SSH authentication
5. **EBS Volumes** - Storage

5. Instance Types & Categories

Instance Naming Convention

Example: t3.large

```
t = Instance Family (General Purpose)
3 = Generation (3rd generation)
. = Separator
large = Size
```

Instance Families

Family	Purpose	Use Cases
T (General Purpose)	Balanced compute, memory, networking	Web servers, small databases
M (General Purpose)	Balanced resources	Application servers
C (Compute Optimized)	High CPU performance	Batch processing, gaming
R (Memory Optimized)	Large memory	Databases, caching
G/P (Accelerated Computing)	GPU instances	Machine learning, graphics
I (Storage Optimized)	High I/O	Data warehousing

T-Series Sizes Comparison

Instance	vCPUs	Memory	Use Case
t3.nano	2	0.5 GB	Micro
t3.micro	2	1 GB	Small
t3.small	2	2 GB	Dev/Test
t3.medium	2	4 GB	Small App
t3.large	2	8 GB	Production
t3.xlarge	4	16 GB	Large App
t3.2xlarge	8	32 GB	High Load

Use Case Examples

1. Simple Website/Blog

Instance Type: t3.micro or t3.small

- Low traffic
- Static content
- Simple CMS

2. E-commerce Application

Instance Type: m5.large or m5.xlarge

- Higher traffic
- Database operations
- More memory needed

3. Video Rendering/Streaming

Instance Type: g5 series (GPU)

- Real-time processing
- High compute power
- GPU acceleration needed

4. In-Memory Database (Real-time Analytics)

Instance Type: r6g series (Memory Optimized)

- Large RAM requirements
- Fast data access
- Analytics workloads

6. Purchasing Options

Pricing Models Comparison

Option	Commitment	Discount	Best For
On-Demand	None	0%	Variable workloads
Reserved	1-3 years	Up to 75%	Steady-state workloads
Spot	None	Up to 90%	Flexible, interruptible

Option	Commitment	Discount	Best For
Savings Plans	1-3 years	Up to 72%	Flexible commitment
Dedicated Hosts	Varies	Varies	Compliance requirements

On-Demand Instances

- ✓ Pay per second (Linux) or per hour (Windows)
- ✓ No upfront commitment
- ✓ Most flexible
- ✓ Best for: Testing, unpredictable workloads

- X Most expensive option

Reserved Instances

- ✓ Up to 75% discount
- ✓ Predictable costs
- ✓ Best for: Steady, predictable workloads

- X Requires 1-3 year commitment
- X Less flexible

Spot Instances

- ✓ Up to 90% discount
- ✓ Access to spare capacity
- ✓ Best for: Batch jobs, CI/CD, fault-tolerant apps

- X Can be interrupted with 2-minute warning
- X Not suitable for critical applications

Dedicated Hosts

- ✓ Physical server dedicated to you
- ✓ Hardware control
- ✓ Best for: Compliance, licensing requirements

- X Most expensive
- X Limited flexibility

7. EBS - Elastic Block Store

What is EBS?

Elastic Block Store (EBS) provides persistent block storage volumes for EC2 instances - think of it as a virtual hard drive that exists independently of the EC2 instance.

In-Depth Explanation:

Analogy: EBS is like an external hard drive for your computer:

- It stores data persistently (data survives power off)
- You can detach from one computer and attach to another
- You can make backups (snapshots)
- It exists independently from the computer

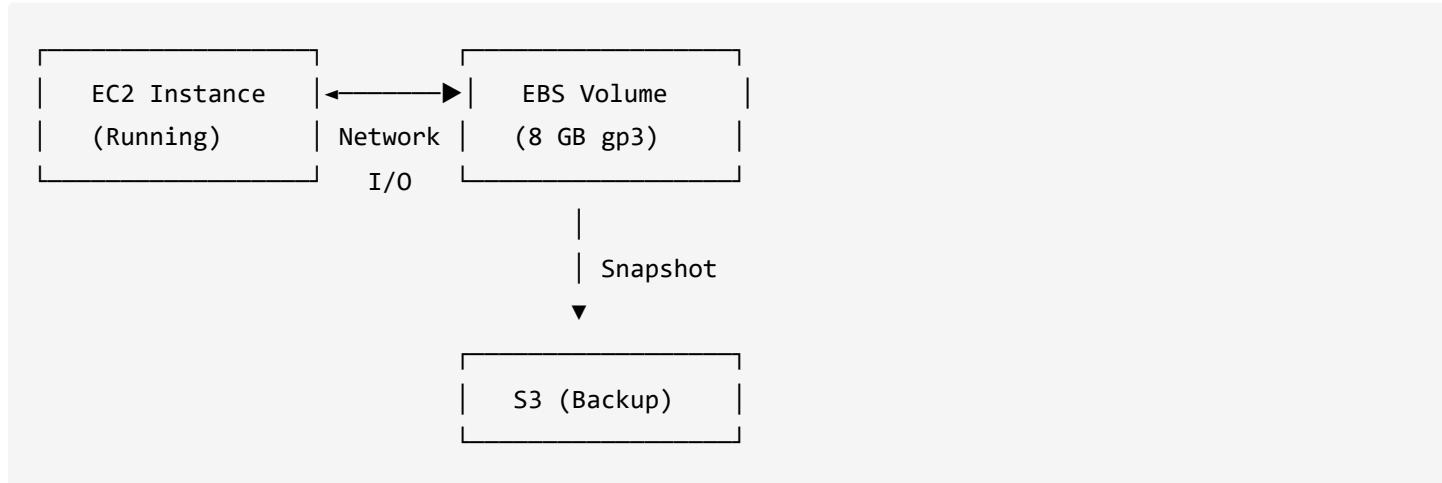
Why EBS is Important:

1. **Persistence:** Data survives instance stop/start
2. **Flexibility:** Resize, change type without data loss
3. **Backup:** Snapshots for disaster recovery
4. **Encryption:** Protect sensitive data at rest
5. **High Availability:** Automatically replicated within AZ

Key Characteristics

- Works like a virtual hard drive
- Data persists independently from instance
- Can be attached/detached from instances
- Supports snapshots for backup
- Available in different types for various workloads

How EBS Works



EBS Volume Types

Type	Name	Use Case	IOPS	Throughput
gp3	General Purpose SSD	General workloads	3,000-16,000	125-1000 MB/s
gp2	General Purpose SSD	Boot volumes	100-16,000	128-250 MB/s
io1/io2	Provisioned IOPS	Databases	Up to 64,000	1,000 MB/s
st1	Throughput Optimized HDD	Big data	500	500 MB/s
sc1	Cold HDD	Infrequent access	250	250 MB/s

When to Use Which Type

gp3 (Recommended Default):

- Web servers
- Development environments
- Small to medium databases
- Boot volumes

io1/io2 (High Performance):

- Production databases (MySQL, PostgreSQL)
- I/O intensive applications
- Applications requiring consistent IOPS
- Mission-critical workloads

st1 (Throughput Optimized):

- Big data processing
- Data warehouses
- Log processing
- Streaming workloads

sc1 (Cold Storage):

- Infrequently accessed data
- Archive storage
- Lowest cost requirement

EBS Features

1. Snapshots

- Point-in-time backup of EBS volume
- Stored in S3 (managed by AWS)
- Can create new volumes from snapshots
- Cross-region copy supported
- Incremental backups (only changed blocks)

2. Encryption

- Data-at-rest encryption
- Data-in-transit encryption
- Uses AWS KMS keys
- No performance impact
- Enabled at volume creation

3. Resizing

- Can increase size without downtime
- Can change volume type
- No need to restart instance
- Changes take time to apply

EBS Best Practices

- ✓ Use gp3 as default (better performance, lower cost)
- ✓ Enable encryption for sensitive data
- ✓ Take regular snapshots
- ✓ Use io2 for critical databases
- ✓ Monitor performance metrics
- ✓ Delete unused volumes to save costs

EBS vs Instance Store

Feature	EBS	Instance Store
Persistence	✓	X
Data survives stop	✓	X
Snapshots	✓	X
Network attached	✓	X (Local)
Performance	Good	Very High
Use case	General	Temporary cache

8. Important Q&A (Real Interview Questions)

Q1: What is EC2 and why is it important?

Answer: EC2 (Elastic Compute Cloud) is AWS's core compute service that provides virtual servers in the cloud. It's important because:

- Eliminates need to invest in hardware upfront
- Scale capacity in minutes, not weeks
- Pay only for capacity you use
- Foundation for many AWS architectures

Q2: Explain EC2 instance states and their differences.

Answer:

State	Description	Billing
Pending	Instance is launching	No charge
Running	Instance is active	Charged
Stopping	Instance is stopping	Charged until stopped
Stopped	Instance is off	No compute charge, EBS charged
Terminated	Instance is deleted	No charge
Rebooting	Instance is restarting	Charged

Key Point: Stopped instances don't incur compute charges but EBS volumes are still charged.

Q3: What happens to EBS data when EC2 is terminated?

Answer: By default:

- **Root volume:** Deleted (DeleteOnTermination = true)
- **Additional volumes:** Retained (DeleteOnTermination = false)

You can modify this behavior during launch or via CLI:

```
aws ec2 modify-instance-attribute --instance-id i-xxx \
--block-device-mappings "[{\\"DeviceName\": \"/dev/xvda\", \\"Ebs\": {\\"DeleteOnTermination\": false}}
```

Q4: Can we attach one EBS volume to multiple EC2 instances?

Answer:

- **Standard EBS:** No, one volume to one instance only
- **EBS Multi-Attach (io1/io2 only):** Yes, up to 16 Nitro instances in the same AZ

Use Case for Multi-Attach: Clustered applications like Oracle RAC, shared storage for high availability.

Q5: What is the difference between gp2 and gp3?

Answer:

Feature	gp2	gp3
Baseline IOPS	3 IOPS/GB (scales with size)	3,000 IOPS (independent)
Max IOPS	16,000	16,000
Throughput	128-250 MB/s	125-1000 MB/s
Cost	Higher	20% cheaper
Configuration	Size-based	Independent IOPS/throughput

Interview Tip: Always recommend gp3 for new deployments unless specific gp2 requirements exist.

Q6: What is the difference between Instance Store and EBS?

Answer:

Aspect	EBS	Instance Store
Persistence	Data persists	Data lost on stop/terminate
Attachment	Network-attached	Physically attached
Reboot behavior	Data preserved	Data preserved
Stop/Terminate	Data preserved (EBS)	Data lost
Use Case	Databases, boot volumes	Temp data, cache, buffers
Snapshots	Supported	Not supported

Q7: How do you choose the right EC2 instance type?

Answer: Consider these factors:

1. **Workload type:** Compute, memory, storage, or GPU intensive?
2. **Resource requirements:** How much CPU, RAM needed?
3. **Budget:** Cost constraints?
4. **Compliance:** Any specific hardware requirements?

Quick Guide:

- General purpose (T, M) → Web servers, small databases
- Compute optimized (C) → Batch processing, high-performance computing
- Memory optimized (R, X) → Large databases, in-memory caching
- Storage optimized (I, D) → Data warehousing, distributed file systems
- GPU (P, G) → Machine learning, graphics rendering

Q8: Explain the difference between On-Demand, Reserved, and Spot instances.

Answer:

Type	Commitment	Discount	Interruption	Best For
On-Demand	None	0%	Never	Variable workloads
Reserved	1-3 years	Up to 75%	Never	Steady-state
Spot	None	Up to 90%	Yes (2 min notice)	Flexible, fault-tolerant

Interview Tip: Mention that production databases should NEVER use Spot instances.

Q9: What is a Security Group? How is it different from NACL?

Answer:

Aspect	Security Group	NACL
Level	Instance level	Subnet level
State	Stateful	Stateless
Rules	Allow only	Allow and Deny
Default	Deny all inbound	Allow all
Evaluation	All rules evaluated	Rules evaluated in order

Q10: What happens when you stop vs terminate an EC2 instance?

Answer:

Action	Stop	Terminate
Instance state	Stopped	Terminated
EBS root volume	Preserved	Deleted (by default)
Instance charges	No	No
EBS charges	Yes	No (if deleted)
Public IP	Released	Released
Elastic IP	Retained	Retained (but charged)

Action	Stop	Terminate
Can restart?	Yes	No

Q11: How do you connect to an EC2 instance?

Answer:

- **Linux:** SSH using key pair

```
ssh -i mykey.pem ec2-user@<public-ip>
```

- **Windows:** RDP using password (retrieved with key pair)
- **Session Manager:** Browser-based, no open ports required (requires IAM role)
- **EC2 Instance Connect:** Browser-based SSH

Q12: What is User Data in EC2?

Answer: User Data is a script that runs automatically when an instance first launches. Used for:

- Installing software
- Applying patches
- Configuring applications

```
#!/bin/bash
yum update -y
yum install -y httpd
systemctl start httpd
echo "Hello World" > /var/www/html/index.html
```

Q13: What is EC2 Placement Group?

Answer: Placement groups control how instances are placed on underlying hardware:

Type	Description	Use Case
Cluster	Instances close together in single AZ	Low latency, HPC
Spread	Instances on different hardware	High availability
Partition	Instances in logical partitions	Hadoop, Cassandra

Q14: How does EBS pricing work?

Answer: EBS pricing components:

1. **Volume storage:** \$/GB-month
2. **Provisioned IOPS:** \$/IOPS-month (io1/io2)
3. **Provisioned throughput:** \$/MB/s-month (gp3)
4. **Snapshots:** \$/GB-month (S3 storage)
5. **Data transfer:** Cross-AZ transfer charged

Q15: What is the maximum size of an EBS volume?

Answer:

- **Maximum size:** 64 TiB (64,000 GB)
- **Minimum size:** 1 GiB (gp2/gp3), 4 GiB (io1/io2)

Note: You can only increase EBS size, not decrease.

Quick Reference Commands

```
# EC2 Operations
aws ec2 describe-instances
aws ec2 run-instances --image-id ami-xxx --instance-type t3.micro
aws ec2 terminate-instances --instance-ids i-xxx

# EBS Operations
aws ec2 describe-volumes
aws ec2 create-volume --size 100 --volume-type gp3 --availability-zone ap-south-1a
aws ec2 attach-volume --volume-id vol-xxx --instance-id i-xxx --device /dev/sdf
aws ec2 create-snapshot --volume-id vol-xxx --description "Backup"

# Check AWS CLI configuration
aws configure list
aws sts get-caller-identity
```

Summary

Topic	Key Points
MFA	Additional security layer, use Authenticator apps
AWS CLI	Command-line access, automation, scripting

Topic	Key Points
EC2	Virtual servers, multiple types, flexible pricing
Instance Types	T (general), C (compute), R (memory), G (GPU)
Pricing	On-Demand, Reserved, Spot, Savings Plans
EBS	Persistent storage, gp3 default, snapshots for backup

Notes from AWS Zero to Hero - File 2

AWS Zero to Hero - Notes 03

EBS Advanced Features, Snapshots, Storage Mounting & Availability Zones

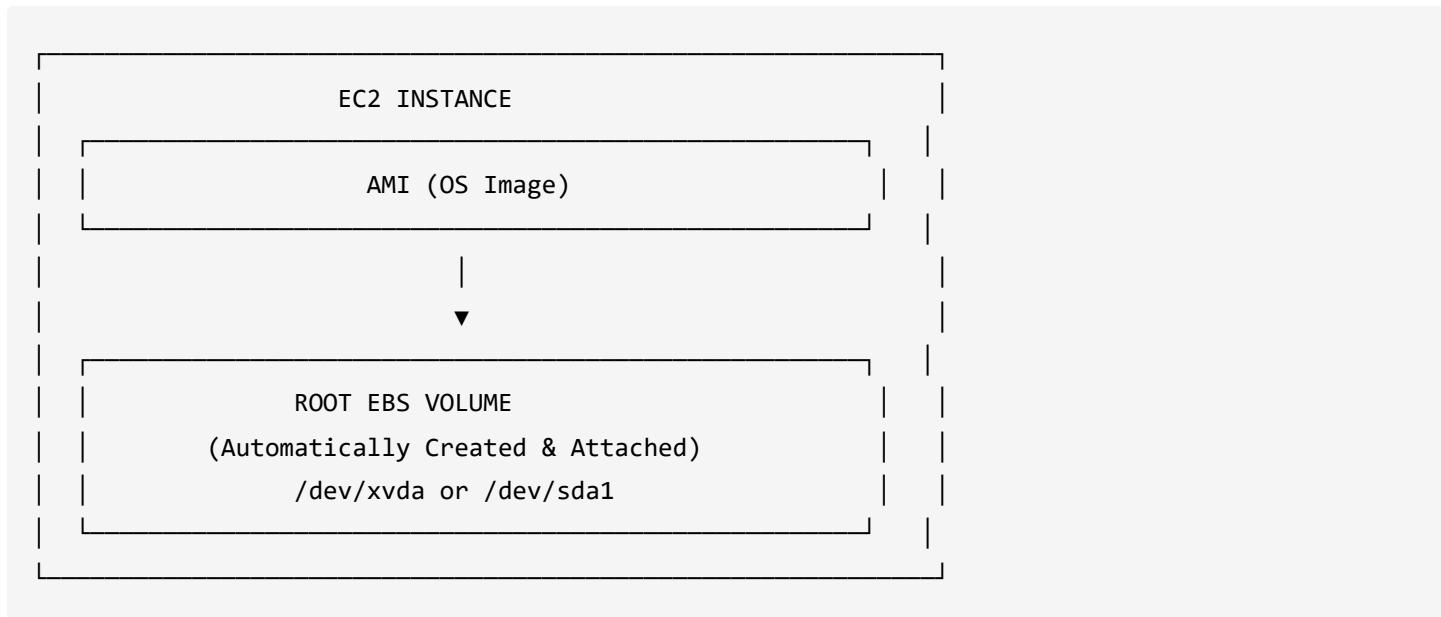
Table of Contents

1. [EBS Volume Attachment & Instance Relationship](#)
 2. [Delete on Termination Behavior](#)
 3. [Instance & EBS Availability Zone Requirement](#)
 4. [Creating Additional EBS Volumes](#)
 5. [Attaching EBS Volumes to Running Instances](#)
 6. [EBS Volume Modification \(Resize\)](#)
 7. [Storage Operations on EC2 \(Linux\)](#)
 8. [EBS Snapshots Deep Dive](#)
 9. [Cross-Region Snapshot Copy](#)
 10. [Creating EBS from Snapshots](#)
 11. [Multiple EBS Attachment to Single Instance](#)
 12. [Important Questions & Answers](#)
-

1. EBS Volume Attachment & Instance Relationship

Understanding Volume-Instance Connection

When you launch an EC2 instance, an EBS volume (root volume) is automatically created and attached.



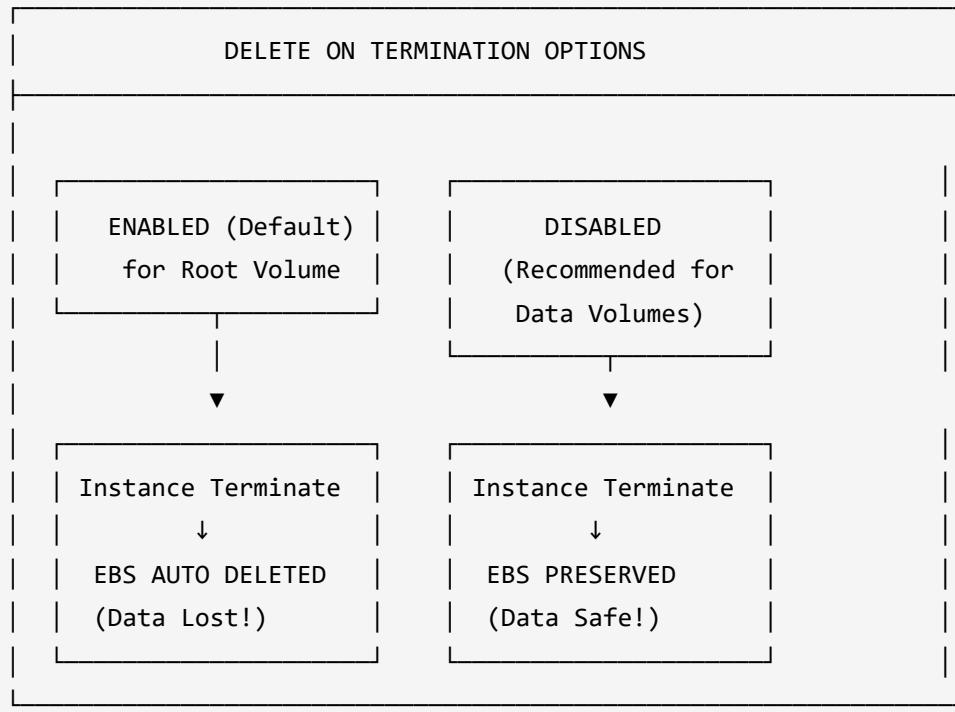
Key Points:

- **Root Volume:** Created automatically with instance launch
- **Volume ID:** Each EBS gets unique identifier (vol-xxxxxx)
- **Attachment Point:** Shows in EC2 console under Storage section
- **Device Name:** Linux uses `/dev/xvda`, `/dev/nvme0n1`, etc.

2. Delete on Termination Behavior

What is "Delete on Termination"?

This setting controls whether an EBS volume is automatically deleted when the attached EC2 instance is terminated.



Configuration Location:

1. **During Launch:** Advanced Settings → Storage → Configure
2. **After Launch:** Actions → Storage → Delete on Termination → Change

Practical Example:

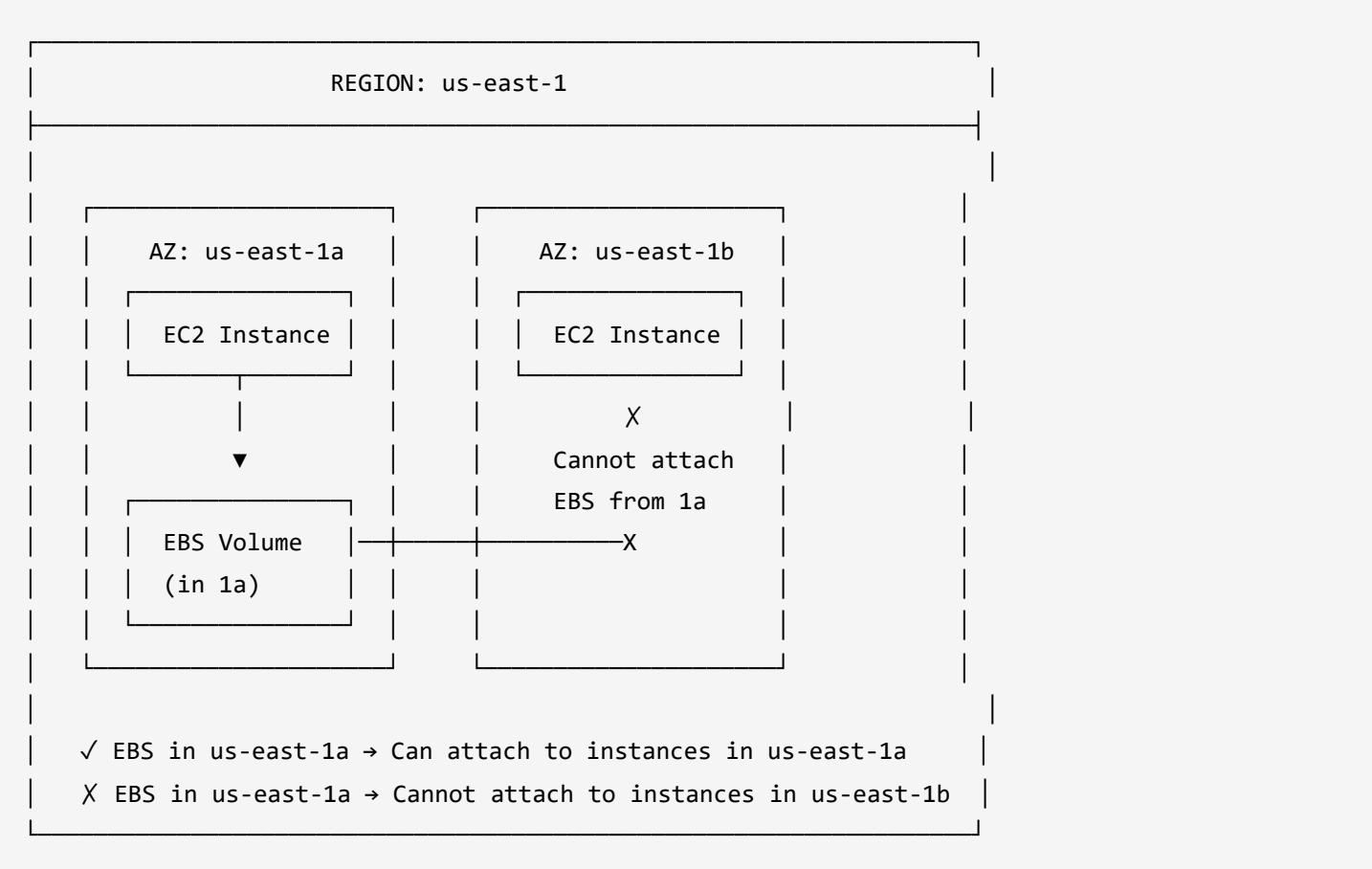
Scenario: Production Database

- X DELETE ON TERMINATION = Yes (RISKY!)
 → Instance terminated → Database LOST
- ✓ DELETE ON TERMINATION = No (SAFE)
 → Instance terminated → EBS remains → Attach to new instance

3. Instance & EBS Availability Zone Requirement

Critical Rule: Same AZ Requirement

⚠️ IMPORTANT: An EBS volume can ONLY be attached to an EC2 instance in the **same Availability Zone (AZ)**.



Why This Limitation?

- **Physical Proximity:** EBS volumes are stored in specific data centers
- **Latency:** Cross-AZ attachment would cause unacceptable latency
- **Solution:** Use **Snapshots** to move data between AZs or Regions

4. Creating Additional EBS Volumes

Step-by-Step Process:

1. **Navigate:** EC2 Console → Elastic Block Store → Volumes
2. **Click:** Create Volume
3. **Configure:**
 - **Volume Type:** gp3 (recommended) / gp2 / io1 / io2
 - **Size:** Specify in GiB (e.g., 5 GiB, 100 GiB)
 - **Availability Zone:** Must match target EC2 instance AZ
 - **Encryption:** Enable for security (recommended)
4. **Create:** Click Create Volume

Volume Configuration Example:

CREATE EBS VOLUME		
Volume Type:	[gp3]	▼
Size (GiB):	[5]	
IOPS:	[3000]	
Throughput:	[125 MiB/s]	
Availability Zone:	[us-east-1a]	▼
Encryption:	[✓] Enable	
KMS Key:	[Default]	▼
Tags:		
Name:	[my-data-volume]	
[Create Volume]		

5. Attaching EBS Volumes to Running Instances

Attachment Process:

State: "Available" → Action: Attach → State: "In-Use"

Steps:

1. **Select Volume:** In EBS Volumes, select the volume
2. **Actions:** Click "Attach Volume"
3. **Select Instance:** Choose the EC2 instance (same AZ)
4. **Device Name:** Auto-assigned (e.g., /dev/sdf , /dev/xvdf)
5. **Attach:** Confirm attachment

Device Naming Convention:

LINUX DEVICE NAMING

Root Volume: /dev/xvda or /dev/nvme0n1

Additional Volumes:

First: /dev/xvdf or /dev/nvme1n1

Second: /dev/xvdg or /dev/nvme2n1

Third: /dev/xvdh or /dev/nvme3n1

Note: AWS suggests /dev/sdf but Linux may show as xvdf

6. EBS Volume Modification (Resize)

Key Points About Resizing:

⚠️ IMPORTANT: EBS volumes can only be **increased** in size, **never decreased**.

Modification Process:

1. **Select Volume:** In Volumes section
2. **Actions:** Modify Volume
3. **Change Size:** Enter new size (must be larger)
4. **Modify:** Click Modify
5. **State Change:** Volume state shows "optimizing"

EBS VOLUME RESIZE

Original: 5 GiB

▼

MODIFY	
Can change:	
5 → 6 GiB ✓	
5 → 100 GiB ✓	
5 → 4 GiB X	← Cannot decrease!

State: "In-Use" → "In-Use (optimizing)" → "In-Use"

⚠ After resize, extend filesystem on OS level

Post-Resize OS Commands (Linux):

```
# Check current disk usage
df -h

# List block devices
lsblk

# Extend partition (if using partition)
sudo growpart /dev/xvdf 1

# Extend filesystem
# For ext4:
sudo resize2fs /dev/xvdf1

# For XFS:
sudo xfs_growfs /mount_point
```

7. Storage Operations on EC2 (Linux)

Complete Flow: Attach → Format → Mount → Use

EBS USAGE WORKFLOW ON LINUX

1. ATTACH VOLUME

AWS Console: Actions → Attach Volume



2. VERIFY ATTACHMENT

\$ lsblk

NAME	SIZE	TYPE	MOUNTPOINT
xvda	8G	disk	
└─xvda1	8G	part	/
xvdf	5G	disk	← New volume (no mount)



3. CHECK IF FILESYSTEM EXISTS

\$ sudo file -s /dev/xvdf

→ "data" means no filesystem

→ "XFS" or "ext4" means has filesystem



4. CREATE FILESYSTEM (if needed)

\$ sudo mkfs -t xfs /dev/xvdf

OR

\$ sudo mkfs -t ext4 /dev/xvdf



5. CREATE MOUNT POINT

\$ sudo mkdir /mybackup



6. MOUNT VOLUME

\$ sudo mount /dev/xvdf /mybackup



7. VERIFY

\$ df -h

/dev/xvdf 5G used avail /mybackup



8. USE STORAGE

```
| $ cd /mybackup  
| $ echo "Hello" > myfile.txt  
|
```

Essential Linux Commands:

```
# List block devices  
lsblk  
  
# Check filesystem  
sudo file -s /dev/xvdf  
  
# Format with XFS  
sudo mkfs -t xfs /dev/xvdf  
  
# Create mount directory  
sudo mkdir /mybackup  
  
# Mount the volume  
sudo mount /dev/xvdf /mybackup  
  
# Check disk space  
df -h  
  
# Unmount volume  
sudo umount /mybackup  
  
# Make mount persistent (add to fstab)  
# Get UUID first:  
sudo blkid /dev/xvdf  
  
# Add to /etc/fstab:  
# UUID=xxxxxx /mybackup xfs defaults,nofail 0 2
```

UUID and Persistent Mounting:

```
# Why use UUID instead of device name?  
# Device names can change across reboots  
# UUID is unique and permanent  
  
# Get UUID  
$ sudo blkid /dev/xvdf  
/dev/xvdf: UUID="abc-123-def" TYPE="xfs"  
  
# Add to /etc/fstab for automatic mount on reboot  
UUID=abc-123-def /mybackup xfs defaults,nofail 0 2
```

8. EBS Snapshots Deep Dive

What is a Snapshot?

A snapshot is a **point-in-time backup** of an EBS volume stored in Amazon S3 (managed by AWS).

In-Depth Explanation:

Think of a snapshot like taking a photograph of your hard drive at a specific moment:

- Captures all data exactly as it existed at that moment
- Stored durably in S3 (99.999999999% durability - 11 9s)
- Can restore data to that exact state anytime

How Snapshots Work (Incremental):

Snapshots are **incremental** - only changed blocks since the last snapshot are saved:

- **First Snapshot:** Copies all blocks (full backup)
- **Subsequent Snapshots:** Only changed blocks (incremental)
- **Benefit:** Faster, cheaper, less storage

INCREMENTAL SNAPSHOT CONCEPT

Volume: [A][B][C][D][E] (100 GB)

Snapshot 1 (Full): Copies all → 100 GB stored

↓

Volume Changes: [A][B*][C][D][E*] (* = changed)

Snapshot 2 (Incremental): Only B*, E* → ~2 GB stored

↓

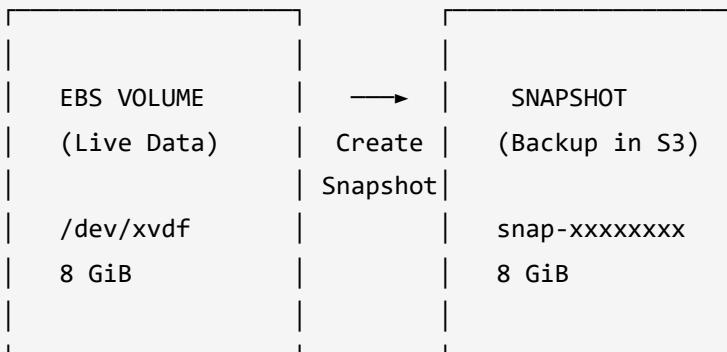
Volume Changes: [A][B*][C*][D][E*] (* = changed)

Snapshot 3 (Incremental): Only C* → ~1 GB stored

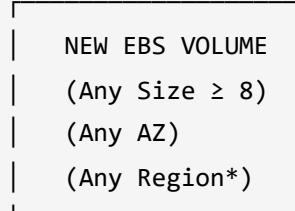
✓ Each snapshot is INDEPENDENT - can delete any

✓ Deleting Snap 1 moves needed blocks to Snap 2

EBS SNAPSHOT CONCEPT



Can Create



*After copying snapshot

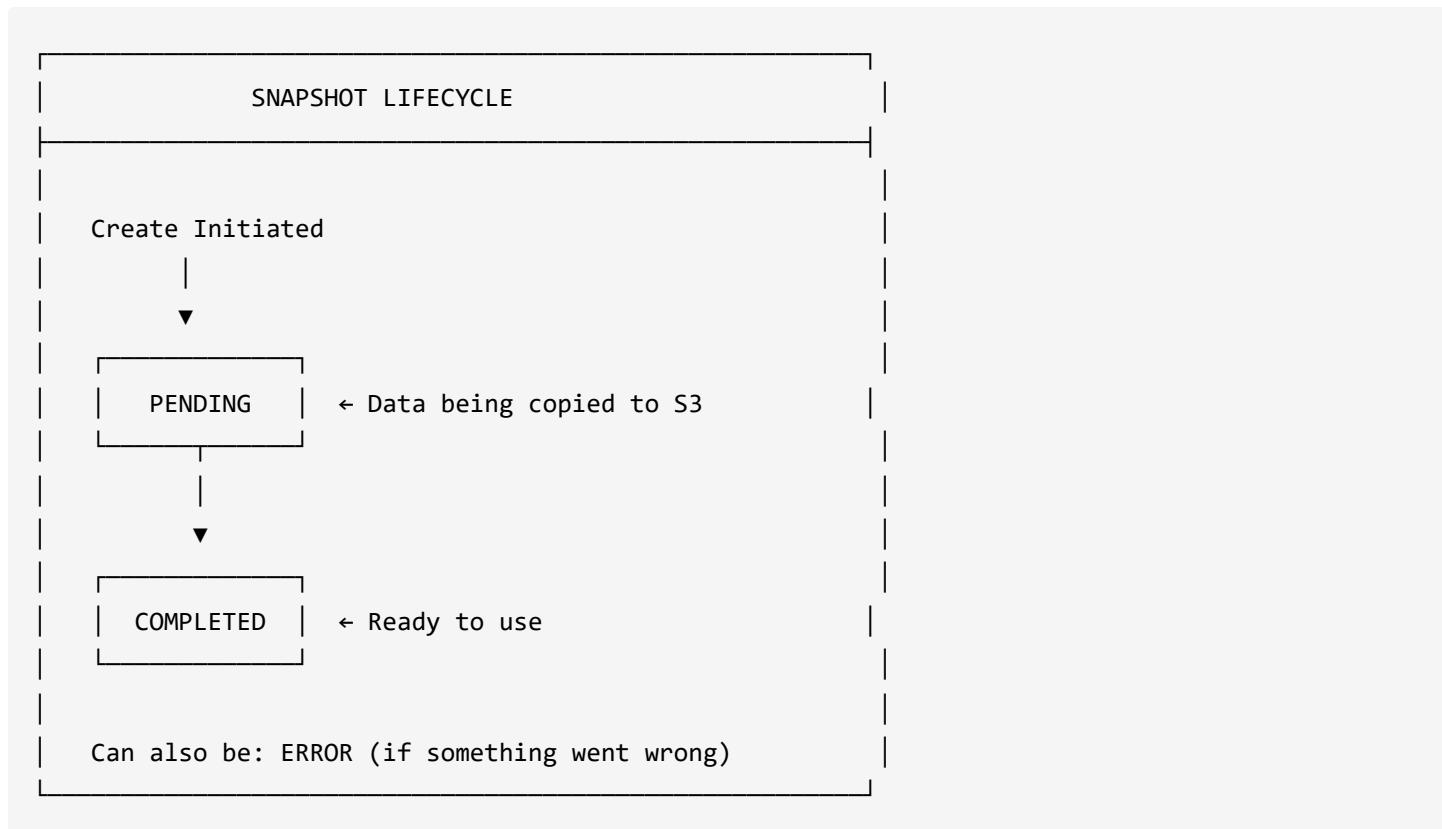
Snapshot Use Cases:

Use Case	Description
Disaster Recovery	Backup data before critical operations
Migration	Move data between AZs
Cross-Region Backup	Copy to different region for DR
Create AMI	Base for custom AMIs
Data Cloning	Create copies for testing

Creating a Snapshot:

1. **Navigate:** EC2 → Volumes → Select Volume
2. **Actions:** Create Snapshot
3. **Add Description:** "Backup before upgrade"
4. **Create:** Snapshot creation starts
5. **Status:** Pending → Completed

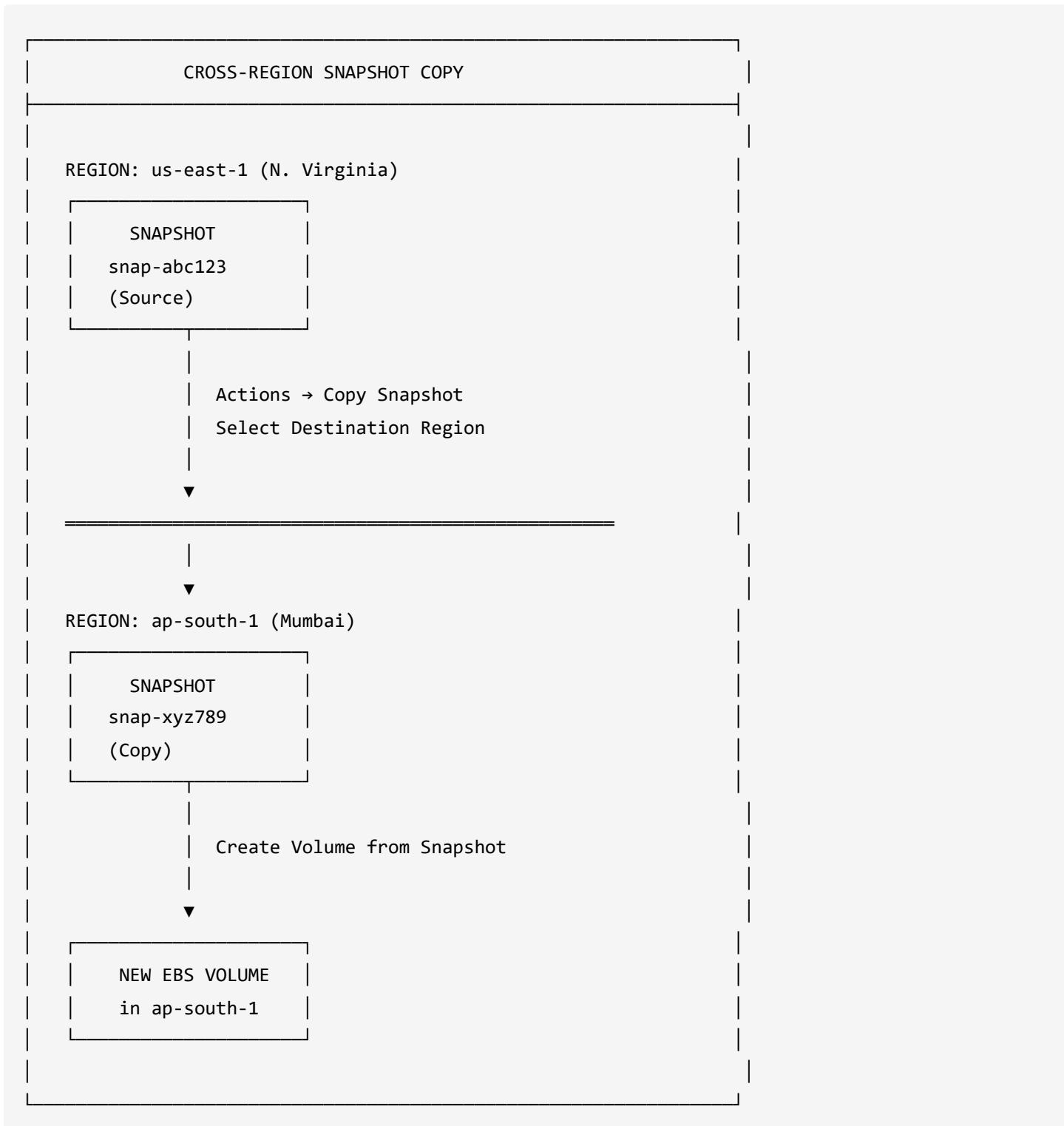
Snapshot States:



9. Cross-Region Snapshot Copy

Why Copy Snapshots Across Regions?

- **Disaster Recovery:** Different geographic location
- **Compliance:** Data residency requirements
- **Migration:** Move workloads to another region
- **Lower Latency:** Serve users in different regions



Steps to Copy Snapshot:

1. **Select Snapshot:** In source region
 2. **Actions:** Copy Snapshot
 3. **Destination Region:** Select target region
 4. **Encryption:** Can encrypt during copy
 5. **Copy:** Initiate copy process
-

10. Creating EBS from Snapshots

Creating Volume from Snapshot:



Creating Instance Using Snapshot:

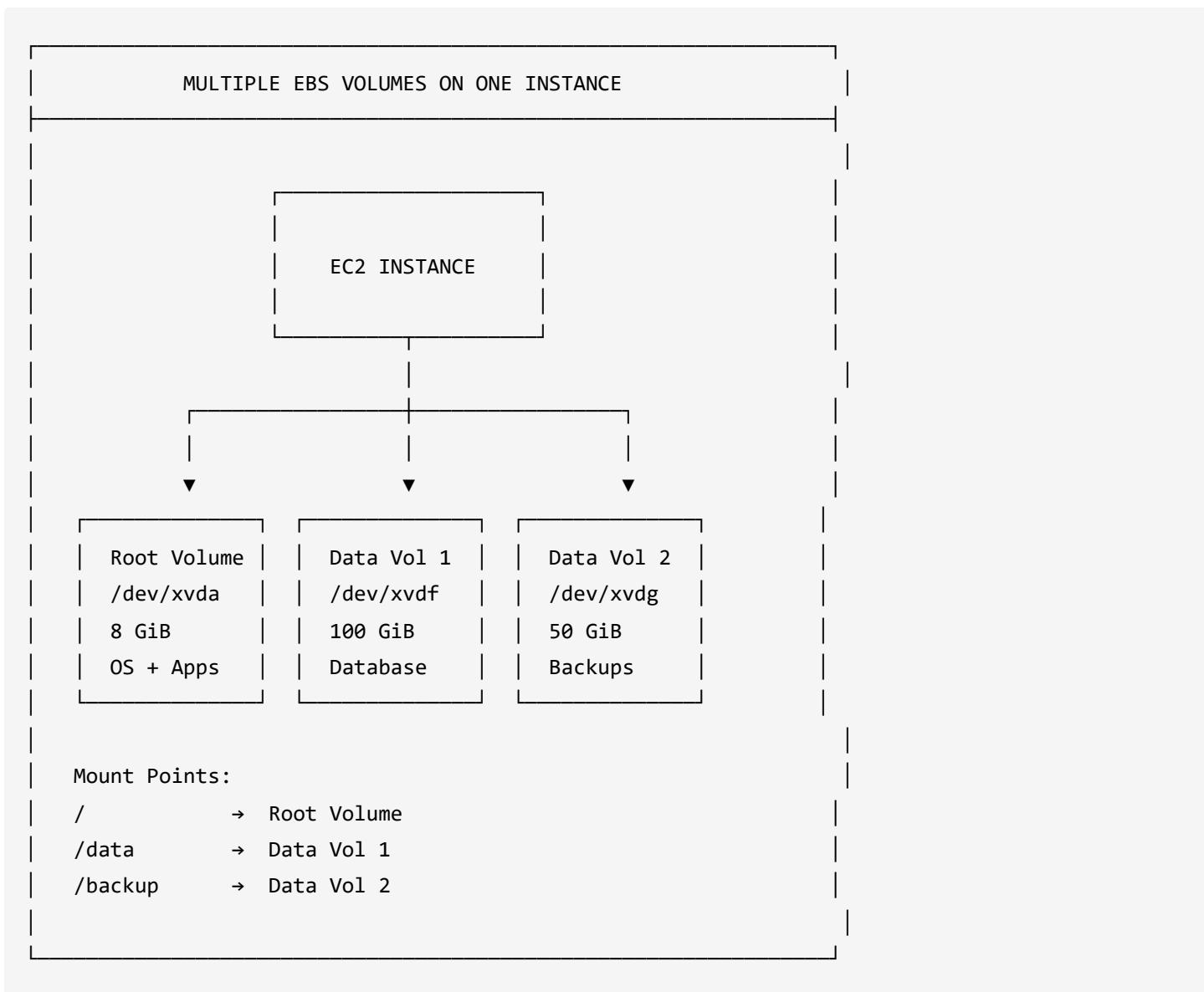
You can also create an EC2 instance directly from a snapshot by:

1. Creating an AMI from the snapshot
 2. Launching instance from that AMI
-

11. Multiple EBS Attachment to Single Instance

Can One Instance Have Multiple EBS Volumes?

YES! An EC2 instance can have multiple EBS volumes attached simultaneously.



Limits:

- Maximum 27 EBS volumes per instance (varies by instance type)
 - Nitro-based instances support more attachments
-

12. Real Interview Questions & Answers

Q1: Can I attach an EBS volume to an instance in a different Availability Zone?

Answer: No, an EBS volume can ONLY be attached to an instance in the **same Availability Zone**. This is a fundamental architectural constraint.

To migrate data to a different AZ:

1. Create a snapshot of the volume
2. Create a new volume from the snapshot in the desired AZ
3. Attach the new volume to the instance in that AZ

Why this limitation exists: EBS volumes are physically located in specific AZs for low-latency access. Cross-AZ attachments would introduce network latency.

Q2: What happens to my EBS volume when I terminate an EC2 instance?

Answer: It depends on the "Delete on Termination" attribute:

Attribute Setting	Root Volume	Additional Volumes
Enabled (Yes)	Deleted automatically	Deleted automatically
Disabled (No)	Preserved as "Available"	Preserved as "Available"
Default	Enabled	Disabled

Best Practice: Always disable "Delete on Termination" for volumes containing critical data.

Q3: Explain EBS Snapshots. Are they incremental or full backups?

Answer: EBS Snapshots are **incremental backups**:

- **First Snapshot:** Full copy of all data blocks
- **Subsequent Snapshots:** Only changed blocks since last snapshot
- **Storage:** Only unique, changed blocks are stored
- **Each snapshot is self-sufficient:** Can restore complete volume from any snapshot

Benefits of Incremental:

- Faster backup creation
- Lower storage costs
- Reduced backup window

Important: Even though incremental, each snapshot is independent. Deleting an earlier snapshot doesn't affect later ones—AWS automatically manages block references.

Q4: What is the difference between Snapshot and AMI?

Answer:

Feature	EBS Snapshot	AMI (Amazon Machine Image)
What it captures	Single EBS volume data	Complete instance blueprint
Contents	Raw data blocks	OS, configuration, all volumes, metadata
Use case	Backup/restore individual volumes	Launch identical instances
Creates	New EBS volume	New EC2 instance
Region scope	Region-specific (can copy)	Region-specific (can copy)
Boot capability	No (just data)	Yes (bootable)

Relationship: An AMI uses snapshots internally to store volume data.

Q5: How do you make EBS mount persistent across reboots?

Answer: Add an entry to `/etc/fstab` using the volume's UUID:

```
# Step 1: Get the UUID
sudo blkid /dev/xvdf

# Step 2: Add to /etc/fstab
UUID=abc123-def456-789 /mydata xfs defaults,nofail 0 2
```

Why UUID instead of device name?

- Device names (like `/dev/xvdf`) can change across reboots
- UUID is a unique identifier that never changes for that filesystem

- Guarantees correct mounting even if device order changes

Important flags:

- `nofail` : System boots even if this volume isn't available
 - `0 2` : Dump and fsck order settings
-

Q6: Can you decrease the size of an EBS volume?

Answer: No, you cannot decrease EBS volume size. This is a permanent limitation.

If you need a smaller volume:

1. Create a new smaller EBS volume
2. Attach to instance
3. Copy data from old volume to new
4. Update fstab or application configuration
5. Detach and delete old volume

Why this limitation?

- Filesystem shrinking is complex and risky
 - Data integrity could be compromised
 - AWS chose to prevent this to avoid data loss
-

Q7: How do you copy an EBS volume to another region?

Answer: You cannot directly copy an EBS volume, but you can use snapshots:

1. **Create Snapshot** of the source volume
2. **Copy Snapshot** to destination region (Actions → Copy Snapshot)
3. **Create Volume** from the copied snapshot in the new region
4. **Attach** to an instance in that region

Use Cases:

- Disaster Recovery (DR)
 - Geographic expansion
 - Compliance (data residency requirements)
 - Migration projects
-

Q8: What are EBS Snapshot States and what do they mean?

Answer:

State	Meaning
Pending	Snapshot is being created (first snapshot takes longer)
Completed	Snapshot is ready for use
Error	Snapshot creation failed

Important: You can create a volume from a snapshot while it's still "pending," but the volume will be slower until the snapshot completes (lazy loading).

Q9: Explain "Lazy Loading" in EBS Snapshots.

Answer: When you create a volume from a snapshot, data is loaded **lazily** (on-demand):

- Volume is immediately available
- Data blocks are fetched from S3 (where snapshots are stored) as they're accessed
- First access to each block is slower
- Subsequent accesses are at full EBS speed

To avoid lazy loading latency:

- Use **Fast Snapshot Restore (FSR)** - pre-initializes volumes instantly
- Run `dd` or `fio` to read all blocks before production use

Cost: FSR charges per AZ per hour it's enabled.

Q10: What is EBS Multi-Attach and when would you use it?

Answer: **EBS Multi-Attach** allows a single EBS volume to be attached to **multiple EC2 instances simultaneously**.

Constraints:

- Only io1/io2 (Provisioned IOPS SSD) volumes
- Maximum 16 instances simultaneously
- All instances must be in same AZ
- Requires cluster-aware filesystem (not ext4/xfs)

Use Cases:

- Clustered databases (Oracle RAC)
- High-availability applications
- Shared storage for tightly coupled workloads

Important: Applications must manage concurrent write access to prevent data corruption.

Q11: How do you increase EBS volume size without downtime?

Answer: EBS Elastic Volumes allows live modification:

1. **Modify Volume:** Console → Volumes → Modify → Enter new size
2. **Wait for Optimization:** State changes to "optimizing" (volume is usable)
3. **Extend Filesystem:** OS doesn't auto-detect the new space

```
# For ext4
sudo resize2fs /dev/xvdf

# For XFS
sudo xfs_growfs /mountpoint
```

Limitations:

- 6-hour wait between modifications
 - Can only increase (never decrease)
 - Works while volume is attached and in-use
-

Q12: What's the difference between IOPS and Throughput?

Answer:

Metric	IOPS	Throughput
Measures	Operations per second	Data transfer per second
Unit	Number (e.g., 3000 IOPS)	MB/s (e.g., 125 MB/s)
Best for	Small, random I/O (databases)	Large, sequential I/O (big data)
Analogy	How many trips per hour	How much cargo per trip

Example Scenario:

- Database with many small queries → needs high IOPS
 - Log processing reading large files → needs high throughput
-

Q13: How is EBS Snapshot pricing calculated?

Answer: You pay only for the **actual data stored**:

- **First snapshot:** Full volume size (e.g., 100 GB volume = ~100 GB stored)
- **Subsequent snapshots:** Only changed blocks (could be just a few GB)
- **Pricing:** Per GB-month of data stored

Cost Optimization Tips:

- Delete old, unnecessary snapshots
 - Use Amazon Data Lifecycle Manager (DLM) for automated cleanup
 - Incremental nature means recent changes cost less
-

Q14: Can you encrypt an existing unencrypted EBS volume?

Answer: Not directly. You must follow this process:

1. Create a snapshot of the unencrypted volume
2. Copy the snapshot with encryption enabled
3. Create a new volume from the encrypted snapshot
4. Swap the volumes (attach new, detach old)

Note: Once encrypted, a volume cannot be unencrypted. Encryption is permanent.

Q15: What happens if the Availability Zone where my EBS volume is located goes down?

Answer:

- **The volume becomes inaccessible** until the AZ recovers
- **Your data is NOT lost** (EBS replicates within the AZ)
- **To recover faster:** Use snapshots stored in S3 (region-wide)

High Availability Strategy:

1. Take regular snapshots (automated with DLM)

2. Keep snapshots up to date
3. In case of AZ failure, create new volume from snapshot in another AZ
4. Update your application to use the new instance/volume

Best Practice: For critical data, maintain multi-AZ architecture using snapshots or database replication

📌 Quick Reference: Common Commands

```
# View attached volumes
lsblk

# Check disk space
df -h

# Check filesystem type
sudo file -s /dev/xvdf

# Format volume (XFS)
sudo mkfs -t xfs /dev/xvdf

# Create mount point
sudo mkdir /mydata

# Mount volume
sudo mount /dev/xvdf /mydata

# Unmount volume
sudo umount /mydata

# Get UUID
sudo blkid /dev/xvdf

# Extend filesystem after resize (ext4)
sudo resize2fs /dev/xvdf

# Extend filesystem after resize (XFS)
sudo xfs_growfs /mydata
```



Summary

Topic	Key Points
EBS-Instance AZ	Must be in same Availability Zone
Delete on Termination	Default ON for root; disable for data safety
Volume Resize	Can only increase, never decrease
Snapshots	Point-in-time backup; stored in S3
Cross-Region	Copy snapshot to another region
Mounting	Format → Create directory → Mount → Add to fstab
Multiple Volumes	One instance can have many EBS volumes

Notes compiled from AWS Zero to Hero Session 3

AWS Notes - File 04: Elastic Load Balancing (ELB), Auto Scaling Groups (ASG) & Launch Templates



Table of Contents

1. [Introduction to Load Balancing](#)
2. [Why Load Balancer is Needed](#)
3. [Types of Load Balancers in AWS](#)
4. [Application Load Balancer \(ALB\) Deep Dive](#)
5. [Network Load Balancer \(NLB\)](#)
6. [Gateway Load Balancer \(GWLB\)](#)
7. [Target Groups](#)
8. [Health Checks](#)
9. [Auto Scaling Groups \(ASG\)](#)

10. [Launch Templates](#)
 11. [AMI \(Amazon Machine Image\)](#)
 12. [Scaling Policies](#)
 13. [ASG Configuration & Setup](#)
 14. [VPC & Subnet Selection for ASG](#)
 15. [Important Q&A](#)
-

1. Introduction to Load Balancing

What is Load Balancing?

Load Balancing is a technique to **distribute incoming traffic** across multiple servers (EC2 instances) to ensure:

- **High Availability** - Application remains accessible even if some servers fail
- **Fault Tolerance** - Automatic failover to healthy instances
- **Scalability** - Handle increased load by distributing traffic
- **Better Performance** - No single server is overloaded

In-Depth Explanation: The Restaurant Analogy

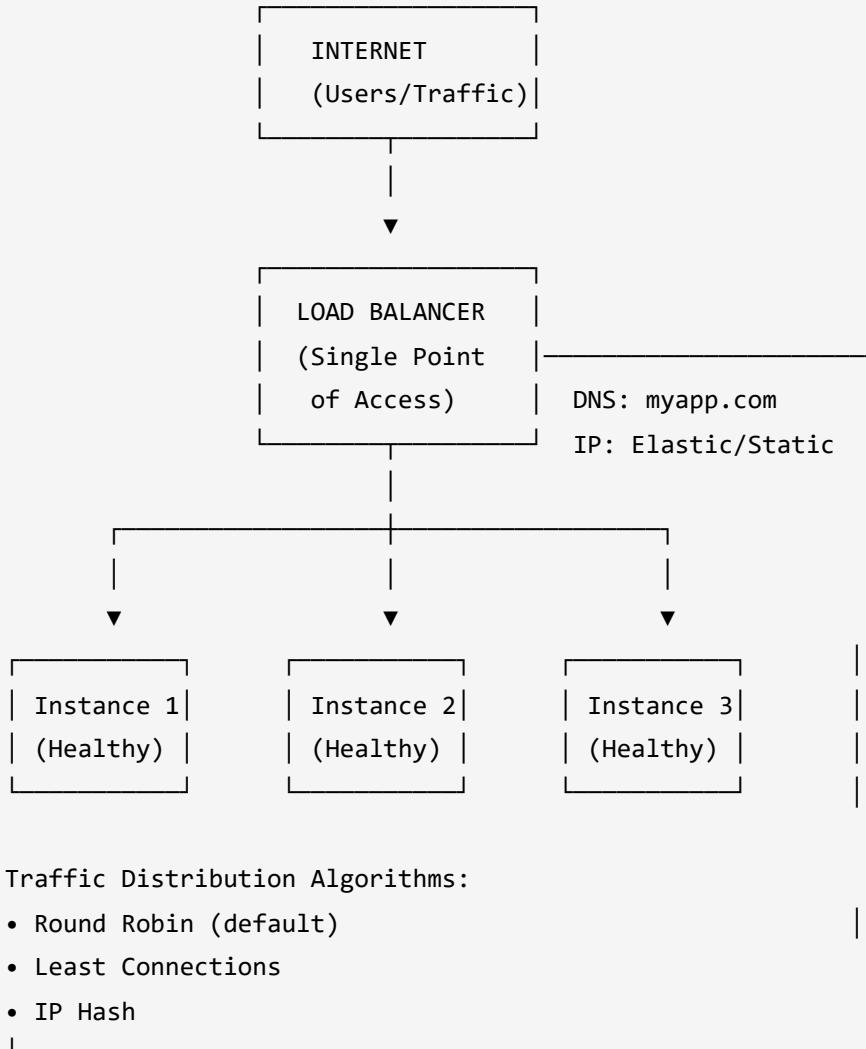
Think of a Load Balancer like a **head waiter at a busy restaurant**:

Restaurant	AWS
Customers arriving	Incoming requests
Head waiter	Load Balancer
Tables/servers	EC2 Instances
Seating guests evenly	Distributing traffic
Knowing which tables are available	Health checks

Without a head waiter: Customers would crowd at one table, overwhelming that server.

With a head waiter: Customers are directed to available tables, ensuring smooth service.

How Load Balancing Works Internally



Key Concepts

Term	Description
Listeners	Define port and protocol for incoming connections
Target Groups	Group of EC2 instances receiving traffic
Health Checks	Verify if instances are healthy
Rules	Define how to route requests

2. Why Load Balancer is Needed

Problems Without Load Balancer

1. Single Point of Failure (SPOF)

- If one server goes down, entire application becomes unavailable
- No automatic failover mechanism

2. No Traffic Distribution

- One server handles all requests
- Server can become overloaded during peak times

3. Poor User Experience

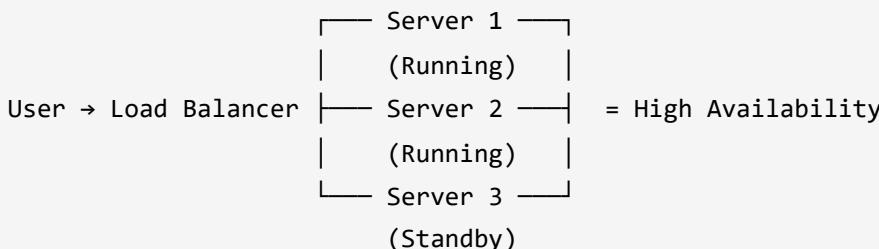
- Slow response times
- Request timeouts

Benefits of Load Balancer

WITHOUT LOAD BALANCER:

User → Single Server (If Down = Application Down)

WITH LOAD BALANCER:



If Server 1 fails → Load Balancer routes to Server 2

Key Benefits Summary

Benefit	Description
High Availability	Application remains accessible
Fault Tolerance	Auto-failover to healthy instances
Scalability	Easy to add/remove instances
Cost Optimization	Scale based on demand
Security	Single point of exposure (DNS)

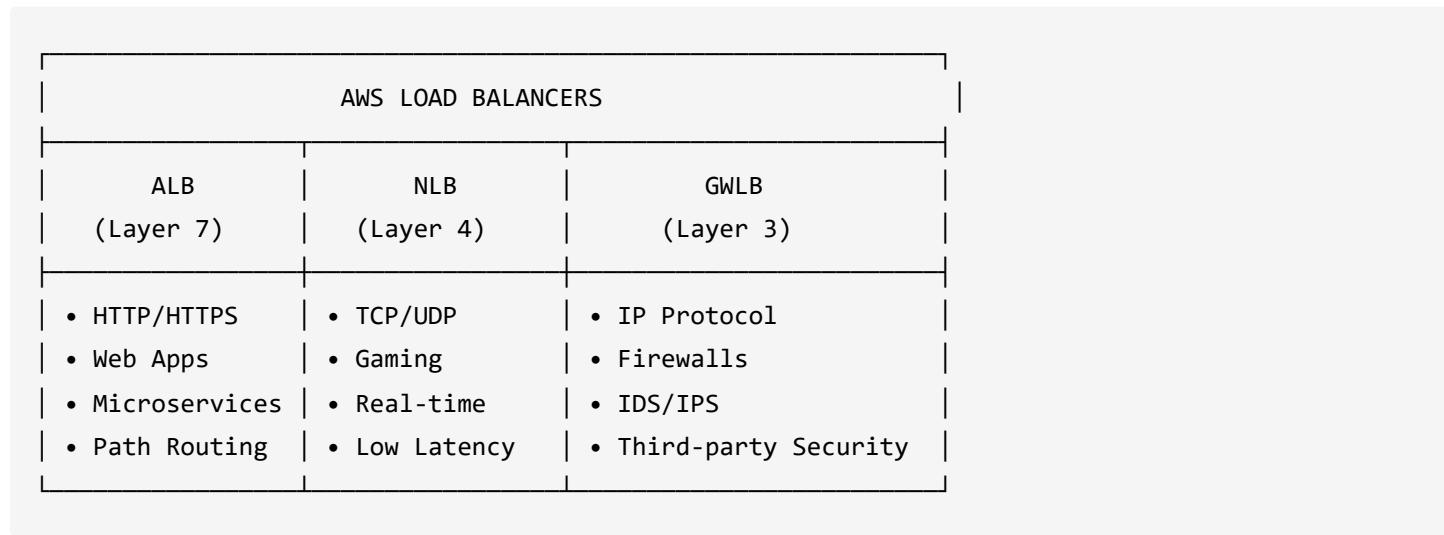
3. Types of Load Balancers in AWS

AWS provides **3 types of Elastic Load Balancers:**

Comparison Table

Feature	ALB	NLB	GWLB
Layer	Layer 7 (Application)	Layer 4 (Transport)	Layer 3 (Gateway)
Protocol	HTTP, HTTPS	TCP, UDP, TLS	IP Protocol
Use Case	Web Applications	High Performance	Third-party Appliances
Latency	Moderate	Ultra-low	Low
Static IP	No (use Global Accelerator)	Yes	No
Path Routing	Yes	No	No
Host Routing	Yes	No	No

Visual Representation



4. Application Load Balancer (ALB) Deep Dive

Overview

- Works at **OSI Layer 7** (Application Layer)
- Best suited for **HTTP/HTTPS** based applications

- Supports **content-based routing** (path, host, headers)
- Most commonly used for web applications

Key Features

1. Path-based Routing

```
example.com/api/*      → API Server Target Group
example.com/images/*   → Image Server Target Group
example.com/*          → Web Server Target Group
```

2. Host-based Routing

```
api.example.com     → API Servers
www.example.com    → Web Servers
admin.example.com  → Admin Servers
```

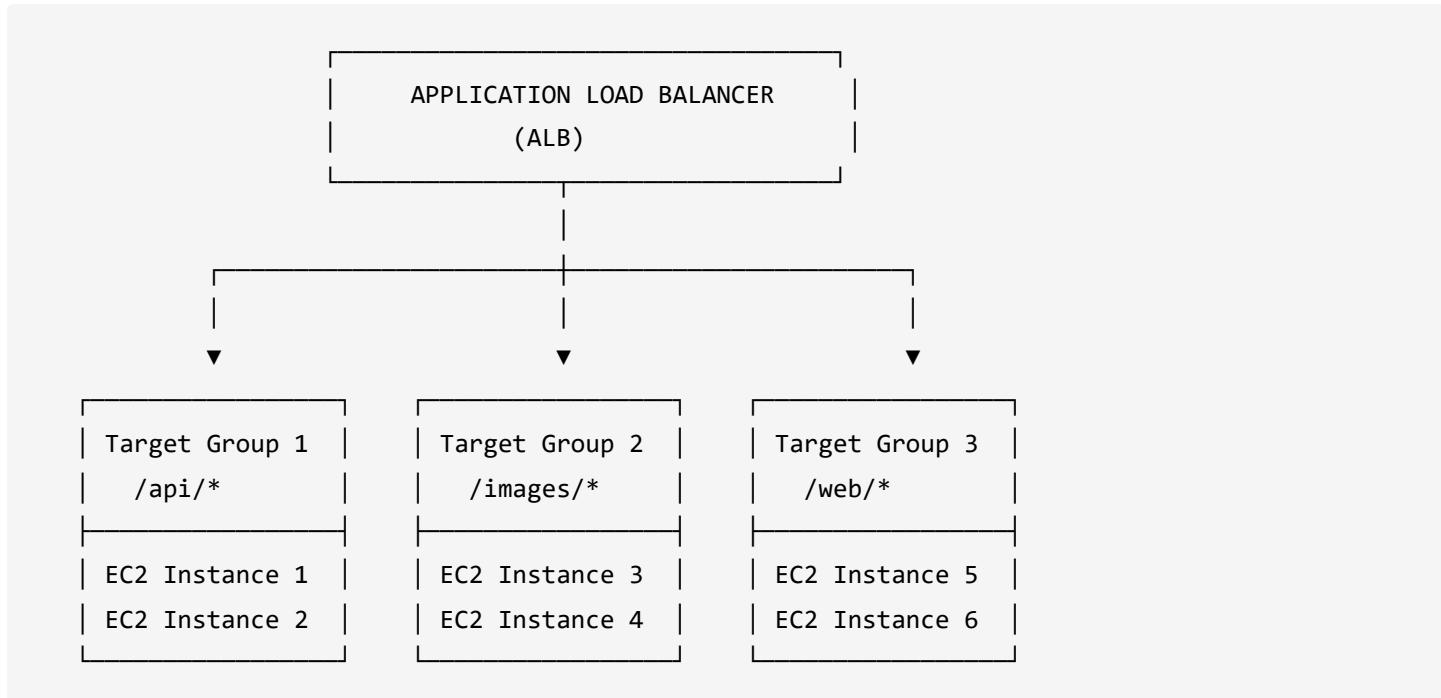
3. HTTP Header/Method Routing

4. Query String Routing

5. WebSocket Support

6. gRPC Support

ALB Architecture



Creating ALB - Step by Step

1. Go to EC2 Dashboard → Load Balancers
2. Click "Create Load Balancer"
3. Select "Application Load Balancer"

4. Configure:

- Name: my-web-server-lb
- Scheme: Internet-facing (for public access)
- IP address type: IPv4
- VPC: Select your VPC
- Availability Zones: Select multiple AZs
- Security Group: Allow HTTP (80), HTTPS (443)

5. Configure Target Group

6. Register Targets (EC2 Instances)

7. Review and Create

5. Network Load Balancer (NLB)

Overview

- Works at **OSI Layer 4** (Transport Layer)
- Handles **TCP, UDP, TLS** protocols
- **Ultra-low latency** (millions of requests per second)
- Provides **static IP addresses**

Use Cases

Use Case	Description
Gaming Applications	Requires ultra-low latency
Financial Applications	Time-sensitive transactions
IoT	High throughput, low latency
Media Streaming	UDP-based streaming

NLB vs ALB

ALB (Layer 7):

Request → ALB → Inspects HTTP Content → Routes to Target

NLB (Layer 4):

Request → NLB → Routes based on IP/Port → Target (Faster!)

6. Gateway Load Balancer (GWLB)

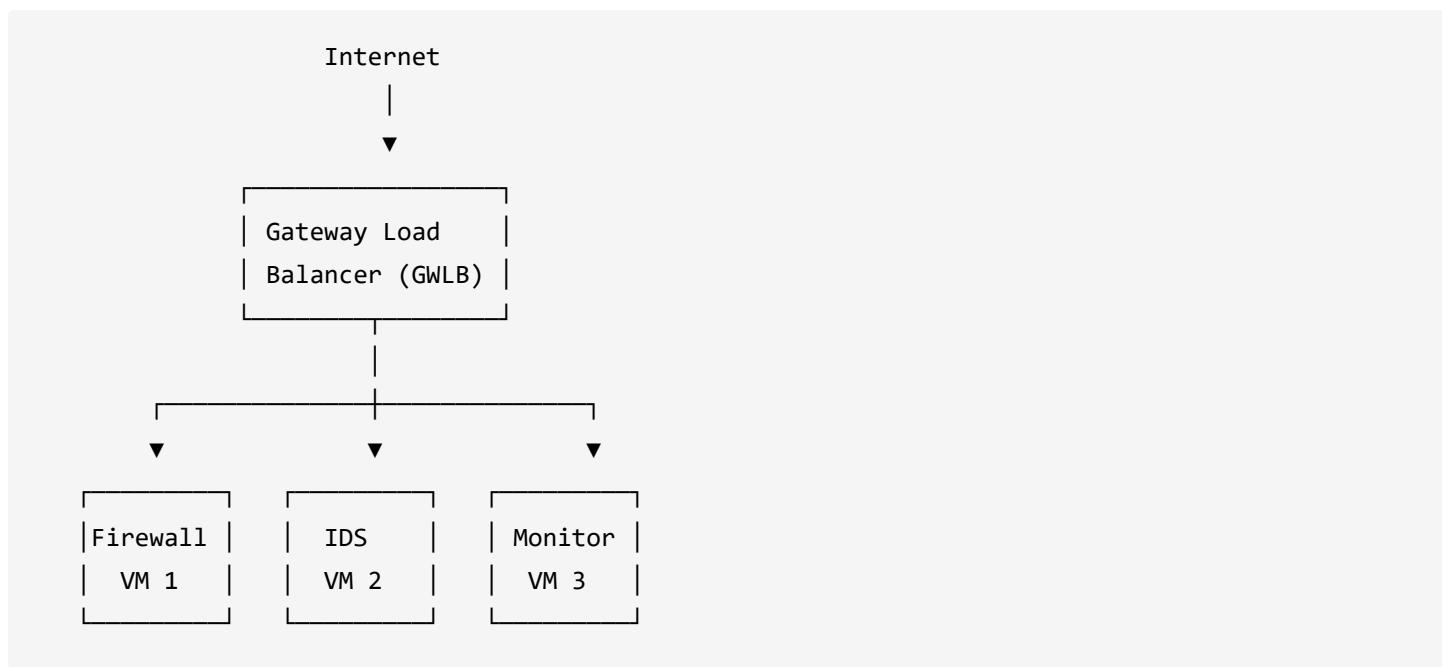
Overview

- Works at **OSI Layer 3** (Network Layer)
- Used with **third-party virtual appliances**
- Common for security appliances (Firewalls, IDS/IPS)

Use Cases

1. **Firewall Integration** (Palo Alto, Fortinet)
2. **Intrusion Detection/Prevention** (IDS/IPS)
3. **Deep Packet Inspection**
4. **Network Monitoring**

Architecture with Third-Party Appliances



7. Target Groups

What is a Target Group?

A **Target Group** is a collection of resources that receive traffic from Load Balancer:

- EC2 Instances
- IP Addresses

- Lambda Functions
- Other ALBs

Target Group Configuration

TARGET GROUP			
Name: my-web-server-tg			
Protocol: HTTP			
Port: 80			
VPC: vpc-12345			
REGISTERED TARGETS:			
Instance	Port	AZ	Health
i-abc123	80	ap-1a	Healthy
i-def456	80	ap-1b	Healthy
i-ghi789	80	ap-1c	Unhealthy

Creating Target Group

1. Navigate to **EC2 → Target Groups**
2. Click **Create Target Group**
3. Select target type: **Instances**
4. Configure:
 - Name: `my-web-server-tg`
 - Protocol: HTTP
 - Port: 80
 - VPC: Select VPC
5. Configure Health Checks
6. Register Targets

8. Health Checks

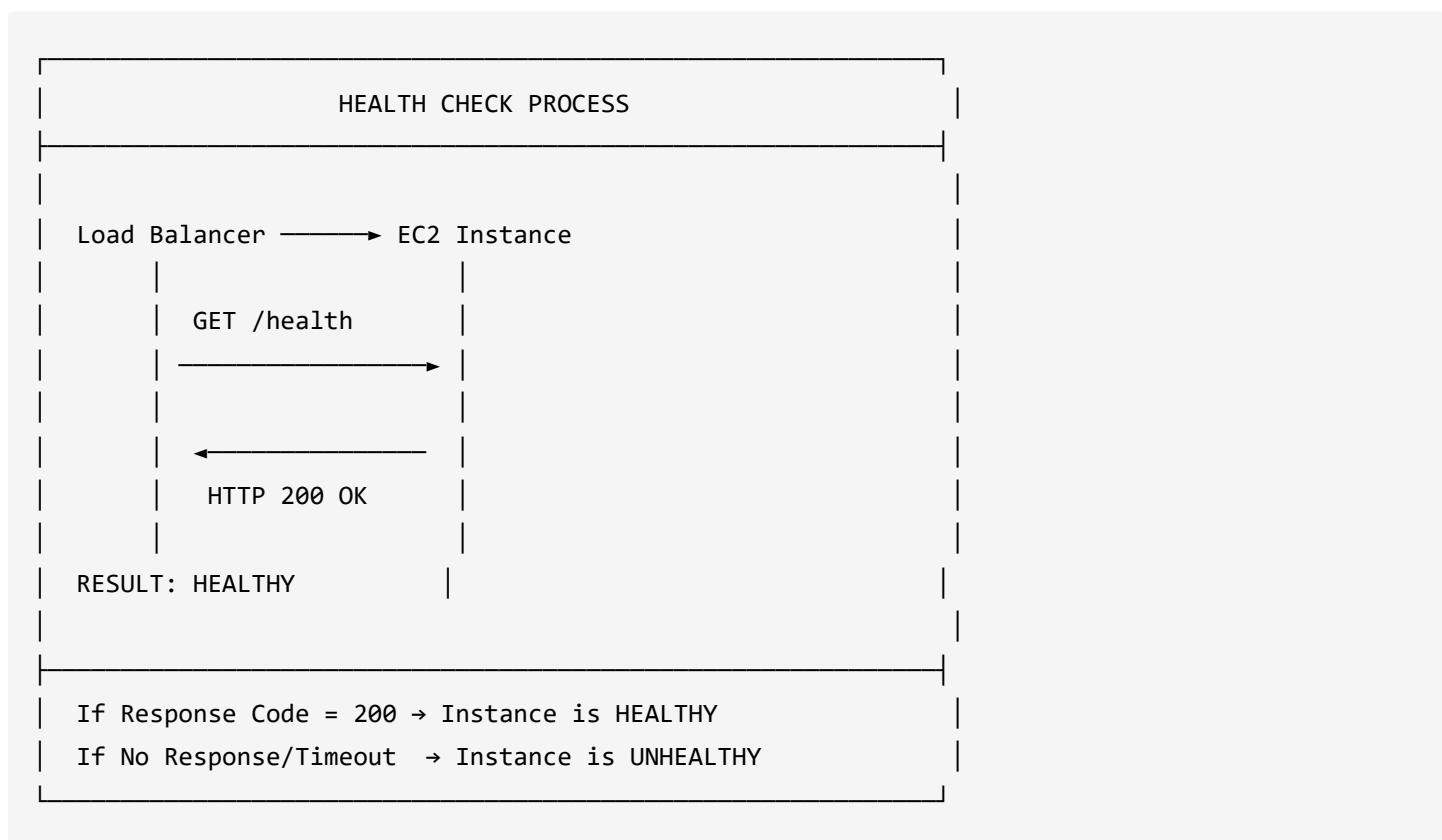
What are Health Checks?

Health checks verify that registered targets are healthy and able to receive traffic.

Health Check Configuration

Parameter	Description	Example
Protocol	HTTP, HTTPS, TCP	HTTP
Path	Health check endpoint	/health
Port	Port to check	80
Healthy Threshold	Consecutive successful checks	2
Unhealthy Threshold	Consecutive failed checks	3
Timeout	Time to wait for response	5 seconds
Interval	Time between checks	30 seconds

Health Check Flow



Health Check States

State	Description
Healthy	Instance passed health checks
Unhealthy	Instance failed health checks
Draining	Instance is being deregistered

State	Description
Initial	Health check in progress

9. Auto Scaling Groups (ASG)

What is Auto Scaling Group?

Auto Scaling Group (ASG) is a service that **automatically adjusts** the number of EC2 instances based on:

- **Demand/Load** (CPU, Memory, Request Count)
- **Schedule** (Time-based scaling)
- **Health Status** (Replace unhealthy instances)

Why ASG is a Game Changer

Without ASG (Traditional Infrastructure):

- Manual capacity planning
- Over-provisioning (wasting money) or under-provisioning (poor performance)
- Manual intervention during traffic spikes
- No automatic recovery from failures

With ASG:

- Automatic capacity adjustment
- Pay only for what you use
- Self-healing infrastructure
- Handle any traffic pattern

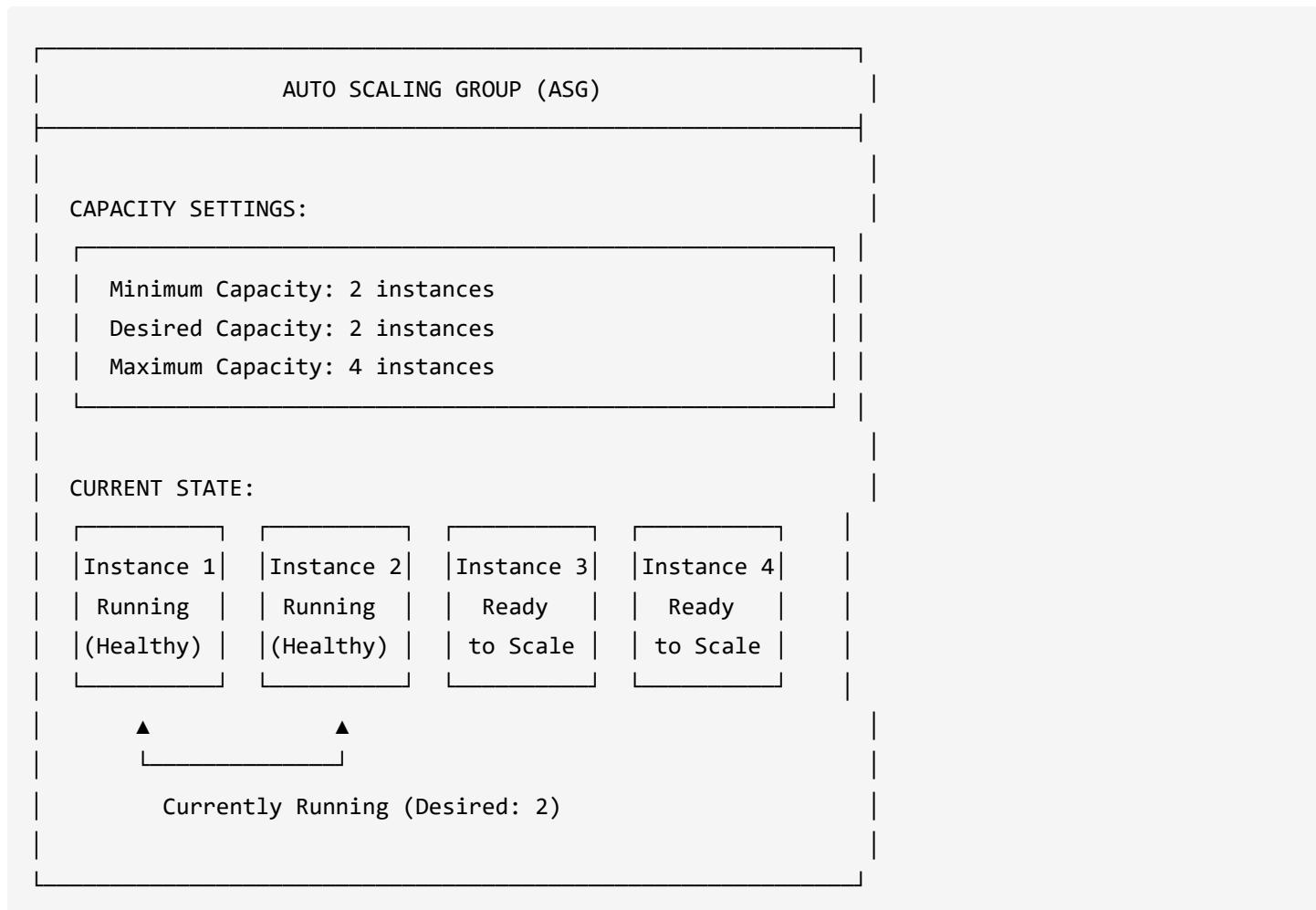
Real-World Analogy: The Smart Taxi Fleet

Scenario	Traditional	With ASG
Morning rush	Fixed 10 taxis (some customers wait)	Auto-add taxis (no waiting)
Night time	Same 10 taxis (wasting fuel)	Scale down to 3 taxis (save money)
Taxi breaks down	Manually dispatch replacement	Auto-replace with new taxi
Special event	Scramble for more taxis	Pre-schedule extra taxis

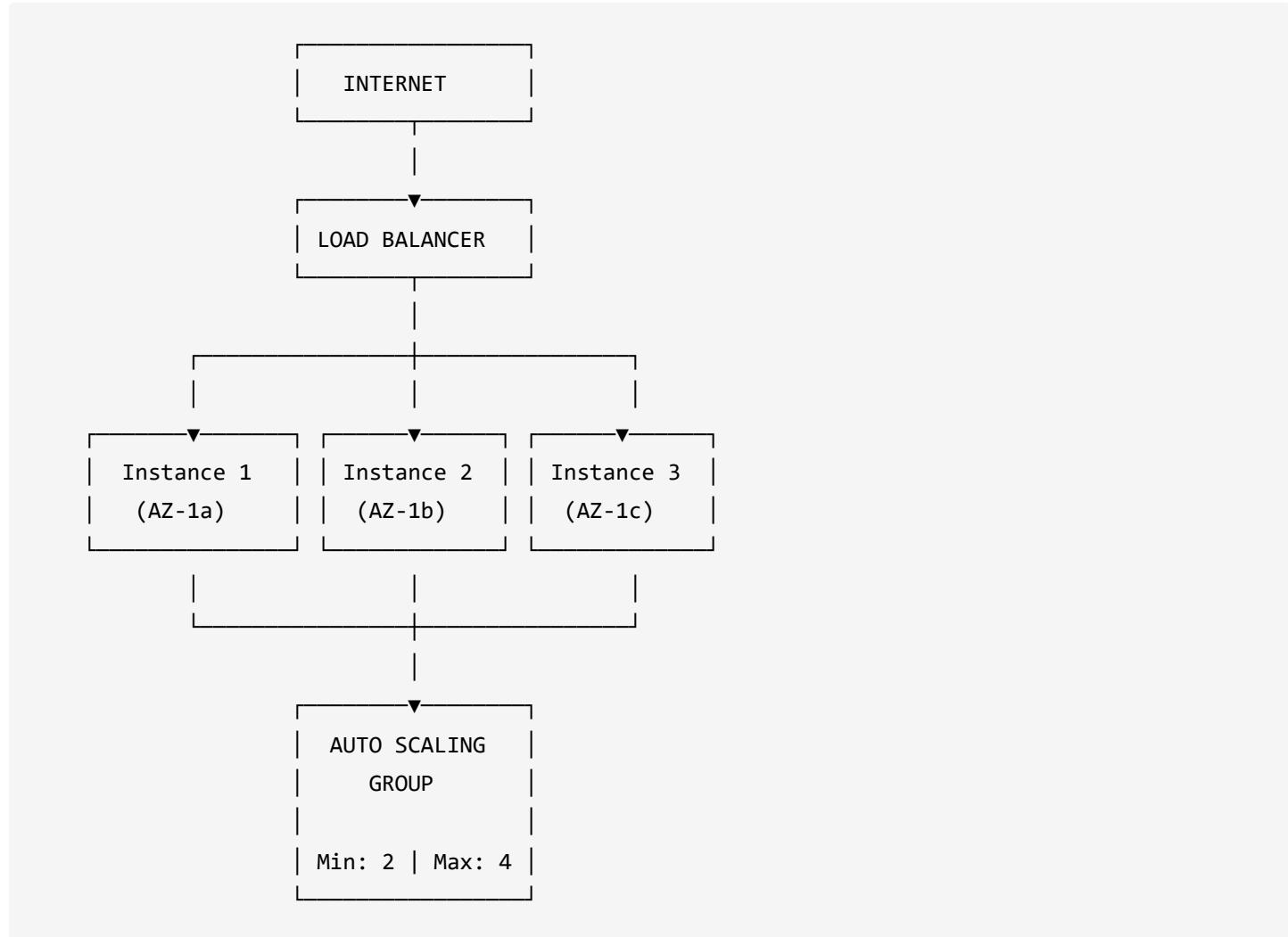
Key Benefits

Benefit	Description
High Availability	Maintains desired capacity
Cost Optimization	Scale down during low demand
Fault Tolerance	Auto-replaces unhealthy instances
Elasticity	Scale up/down automatically

ASG Architecture



ASG with Load Balancer Integration



10. Launch Templates

What is Launch Template?

Launch Template defines the configuration for EC2 instances that ASG will launch:

- AMI ID
- Instance Type
- Key Pair
- Security Groups
- User Data (Bootstrap scripts)
- Network settings

Launch Template vs Launch Configuration

Feature	Launch Template	Launch Configuration
Versioning	✓ Supported	✗ Not Supported
AWS Recommendation	✓ Recommended	⚠ Legacy
Spot Instances	✓ Supported	Limited
Multiple Instance Types	✓ Supported	✗ Not Supported

Creating Launch Template

```

LAUNCH TEMPLATE

Name: my-web-server-template
Version: 1

CONFIGURATION:
└─ AMI: ami-12345678 (My Web Server Image)
└─ Instance Type: t3.micro
└─ Key Pair: my-key-pair
└─ Security Group: my-web-server-sg (HTTP/HTTPS enabled)
└─ Storage: 8 GB gp3
└─ User Data:
    #!/bin/bash
    yum update -y
    yum install -y httpd
    systemctl start httpd
    systemctl enable httpd

```

Steps to Create Launch Template

1. Go to **EC2 → Launch Templates**
2. Click **Create Launch Template**
3. Configure:
 - Template name: `my-web-server-lt`
 - AMI: Select your AMI (or My AMIs if created)
 - Instance Type: `t3.micro` (Free Tier eligible)
 - Key Pair: Select existing or create new
 - Network Settings:
 - VPC: Select your VPC

- Subnet: Select subnet
- Security Group: Select or create (enable HTTP/HTTPS)
- User Data: Add bootstrap script if needed

4. Click **Create Launch Template**

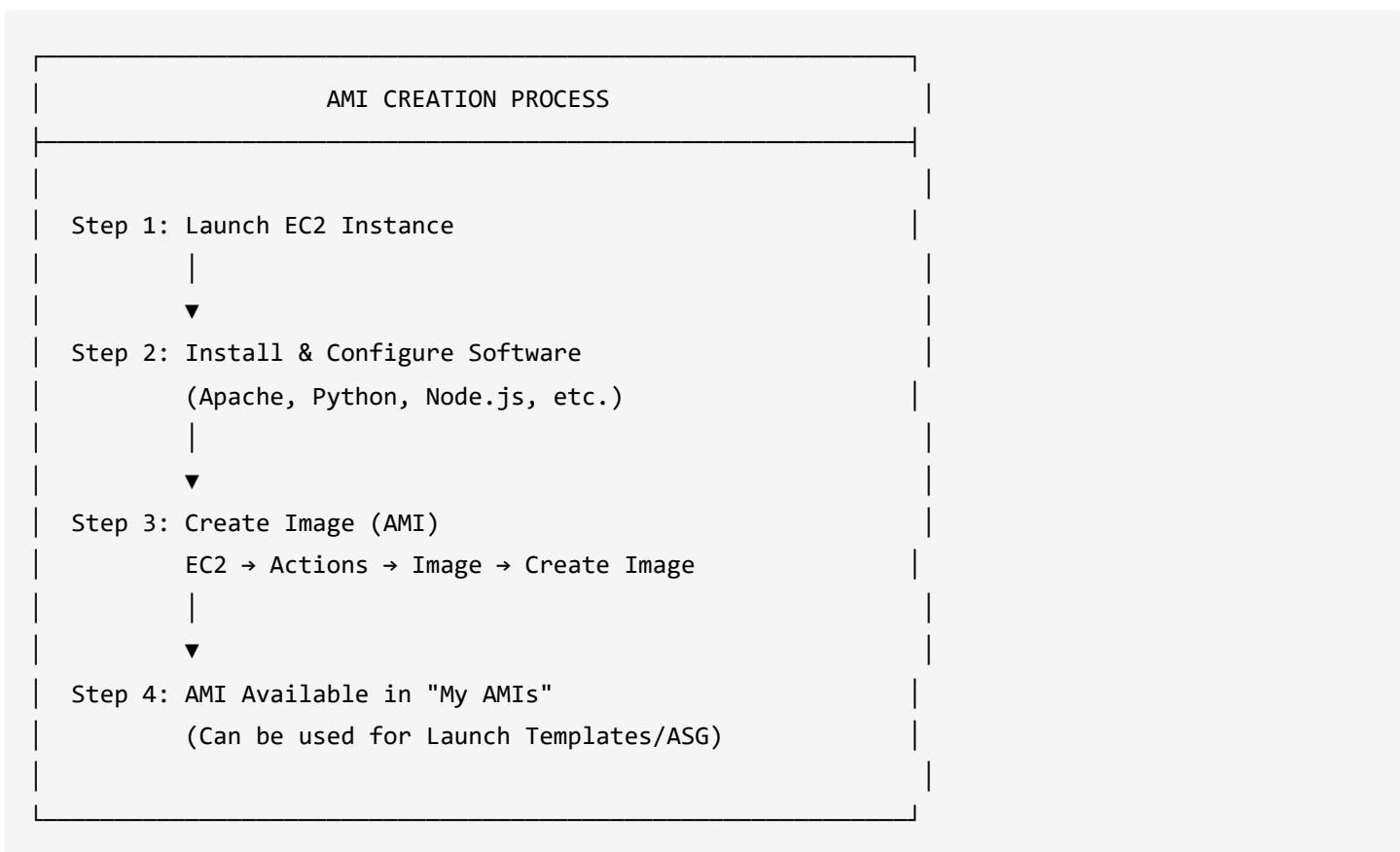
11. AMI (Amazon Machine Image)

What is AMI?

AMI (Amazon Machine Image) is a pre-configured template containing:

- Operating System
- Application software
- Configurations
- Data volumes

Creating Custom AMI



AMI Types

Type	Description
Amazon Quick Start	AWS provided AMIs (Amazon Linux, Ubuntu, Windows)
My AMIs	Your custom created AMIs
AWS Marketplace	Third-party vendor AMIs
Community AMIs	Public shared AMIs

12. Scaling Policies

Types of Scaling Policies

Policy Type	Description	Use Case
Target Tracking	Maintain metric at target value	Keep CPU at 50%
Step Scaling	Scale based on alarm thresholds	Scale in steps
Simple Scaling	Single adjustment	Basic scaling
Scheduled Scaling	Scale at specific times	Known traffic patterns

Target Tracking Policy Example

TARGET TRACKING SCALING POLICY

Target: Average CPU Utilization = 50%

SCENARIO:

Current CPU: 70% (Above Target)

Action: SCALE OUT - Add instances

Current CPU: 30% (Below Target)

Action: SCALE IN - Remove instances

Scaling Metrics

Metric	Description
CPUUtilization	Average CPU usage across instances
RequestCountPerTarget	Number of requests per target
ALBRequestCountPerTarget	ALB specific request count
NetworkIn/Out	Network traffic

Scheduled Scaling Example

SCHEDULED SCALING

Peak Hours (6 PM - 12 AM):

- └─ Min: 3
- └─ Desired: 4
- └─ Max: 6

Off-Peak Hours (12 AM - 6 PM):

- └─ Min: 1
- └─ Desired: 2
- └─ Max: 3

13. ASG Configuration & Setup

Step-by-Step ASG Creation

1. **Create Launch Template** (Required first)
2. **Go to EC2 → Auto Scaling Groups**
3. **Click "Create Auto Scaling Group"**
4. **Configure:**

```
ASG CONFIGURATION WIZARD

STEP 1: Choose Launch Template
└─ Name: my-web-server-asg
└─ Launch Template: my-web-server-lt

STEP 2: Choose Instance Launch Options
└─ VPC: Select VPC
└─ Availability Zones: Select multiple AZs
    └─ ap-south-1a
    └─ ap-south-1b
    └─ ap-south-1c

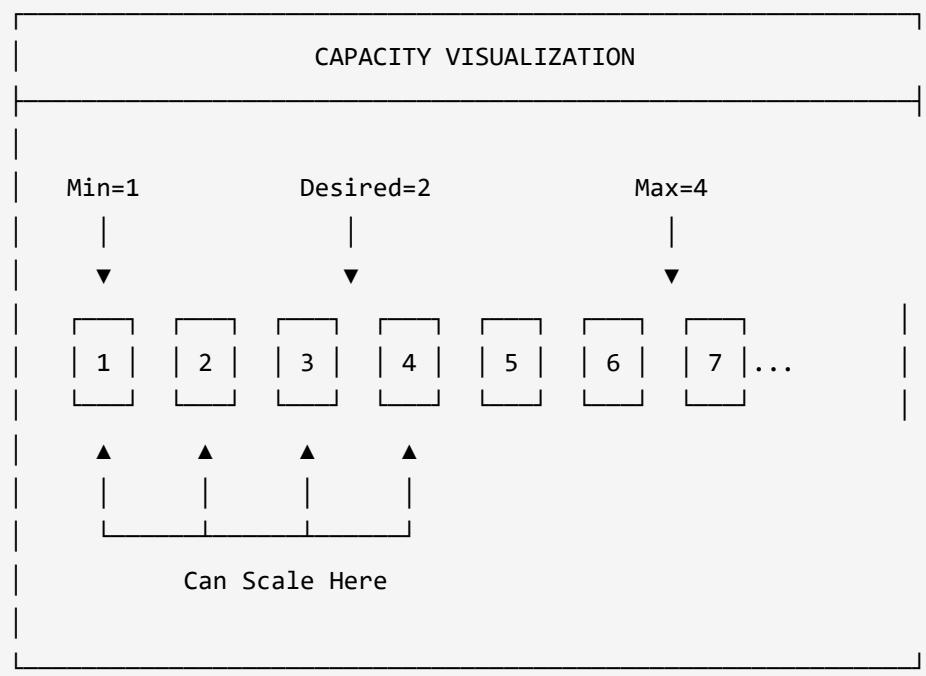
STEP 3: Configure Advanced Options
└─ Load Balancer: Attach to existing ALB
└─ Health Check Type: ELB

STEP 4: Configure Group Size
└─ Desired Capacity: 2
└─ Minimum Capacity: 1
└─ Maximum Capacity: 4

STEP 5: Configure Scaling Policies
└─ Target Tracking: CPU = 50%
```

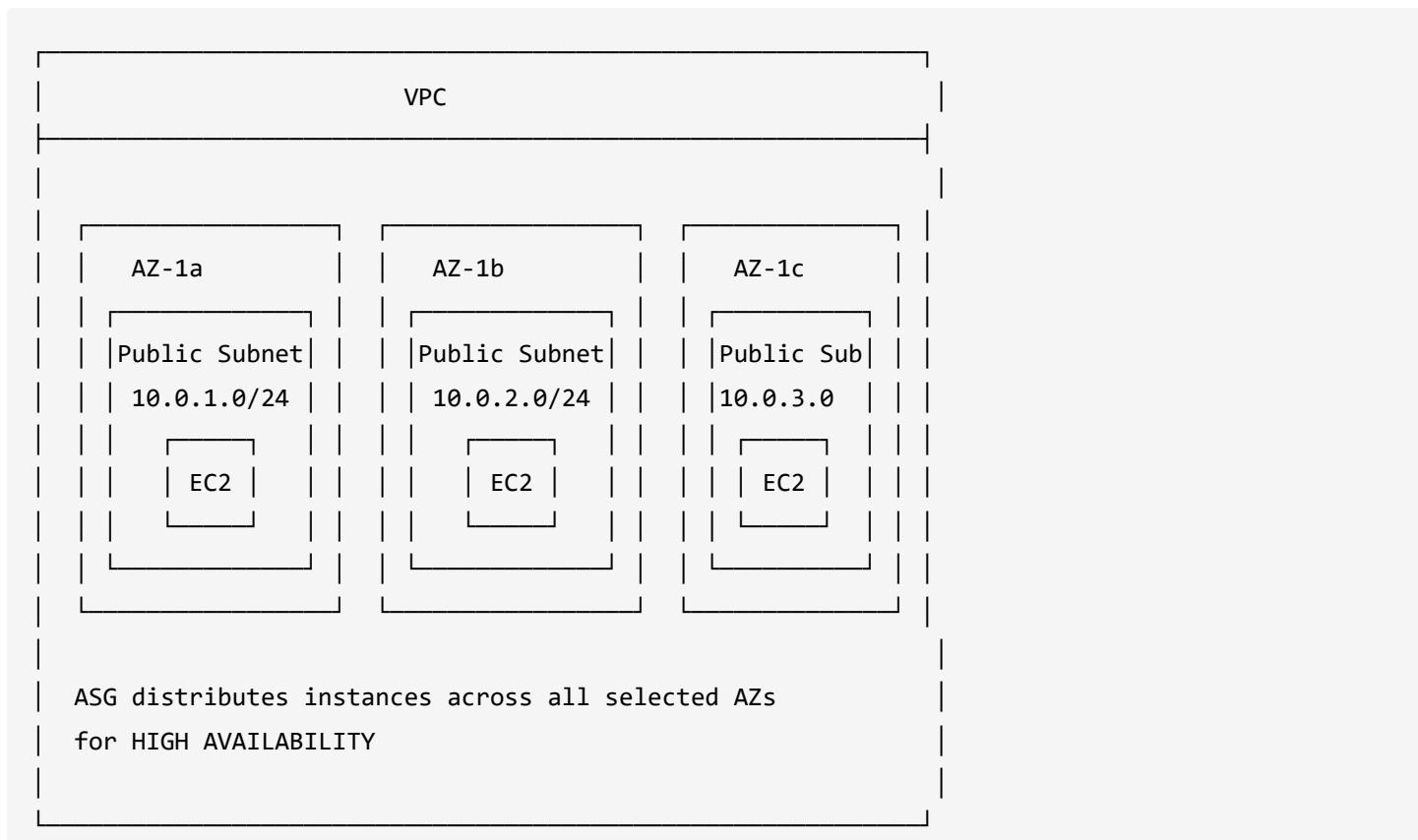
Capacity Settings Explained

Setting	Description
Minimum	Lowest number of instances (even during scale-in)
Desired	Target number of instances to maintain
Maximum	Highest number of instances (even during scale-out)



14. VPC & Subnet Selection for ASG

Multi-AZ Deployment



Why Multiple AZs?

Single AZ	Multiple AZs
If AZ fails, all instances down	Instances in other AZs continue working
No fault tolerance	High fault tolerance
Not recommended for production	Recommended for production

15. Real Interview Questions & Answers

Q1: What is the difference between ALB, NLB, and GWLB?

Answer:

Feature	ALB	NLB	GWLB
OSI Layer	Layer 7 (Application)	Layer 4 (Transport)	Layer 3 (Network)
Protocols	HTTP, HTTPS, gRPC	TCP, UDP, TLS	IP Protocol
Routing	Content-based (path, host, headers)	Connection-based (IP, port)	IP-based
Latency	Higher (inspects content)	Ultra-low (~100ms)	Low
Static IP	No (use Global Accelerator)	Yes (1 per AZ)	No
Use Cases	Web apps, microservices, APIs	Gaming, IoT, real-time	Firewalls, IDS/IPS

Key Decision: Use ALB for web apps, NLB when you need extreme performance or static IPs.

Q2: Explain how ALB path-based routing works with an example.

Answer: Path-based routing directs traffic to different target groups based on URL path:

Example: e-commerce application

```
/api/*      → API Server Target Group (handles backend)
/images/*   → Image Server Target Group (serves static files)
/checkout/* → Payment Server Target Group (PCI compliant)
/*          → Web Server Target Group (default)
```

Benefits:

- Single domain serves multiple applications
 - Different scaling per component
 - Cost-effective (one ALB, multiple services)
 - Microservices architecture support
-

Q3: What is Target Group and why is it needed?

Answer:

Target Group is a logical collection of resources that receive traffic from a Load Balancer.

Why needed:

1. **Grouping:** Organize instances by function (API servers, web servers)
2. **Health Checks:** Configure health monitoring per group
3. **Routing:** Direct traffic based on rules
4. **Management:** Easy scaling and deployment

Target Types:

- **Instance:** EC2 instances
 - **IP:** Private IPs (useful for on-premises)
 - **Lambda:** Serverless functions
 - **ALB:** Another Application Load Balancer
-

Q4: How do ELB Health Checks work? What's the difference between EC2 and ELB health checks?

Answer:

Type	EC2 Health Check	ELB Health Check
What it checks	Instance running status	Application responding

Type	EC2 Health Check	ELB Health Check
Level	System/hardware	Application
Detection	Instance stopped/terminated	App crashed but instance running
Example	Server powered on	Web server returning HTTP 200

ELB Health Check Parameters:

- **Protocol/Port:** HTTP:80
- **Path:** /health
- **Healthy Threshold:** 2 consecutive successes
- **Unhealthy Threshold:** 3 consecutive failures
- **Interval:** 30 seconds
- **Timeout:** 5 seconds

Best Practice: Always use ELB health checks for production—they detect application issues, not just hardware.

Q5: What is Auto Scaling Group and how does it work?

Answer:

ASG automatically adjusts EC2 capacity to maintain application availability and optimize costs.

How it works:

1. **Define Launch Template:** Specifies instance configuration
2. **Set Capacity:** Min, Desired, Max instance counts
3. **Configure Scaling:** Policies for scale-out/scale-in
4. **Monitor:** CloudWatch triggers scaling actions
5. **Execute:** ASG launches/terminates instances

Key Components:

```
Min Capacity = 2 (Floor: never below this)
Desired Capacity = 3 (Target: maintains this)
Max Capacity = 6 (Ceiling: never above this)
```

Q6: Explain the different scaling policies in ASG.

Answer:

Policy	Description	Use Case
Target Tracking	Maintain metric at target	Keep CPU at 50%
Step Scaling	Scale in steps based on alarm	Add 2 instances if CPU > 70%, add 4 if > 90%
Simple Scaling	Single adjustment per alarm	Add 1 instance when alarm triggers
Scheduled	Scale at specific times	Scale up at 9 AM, down at 6 PM
Predictive	ML-based forecasting	Anticipate traffic patterns

Best Practice: Use Target Tracking for most cases—it's simplest and most effective.

Q7: What happens when an instance in ASG fails health checks?

Answer:

The process follows these steps:

1. **Detection:** Health check fails (threshold met)
2. **Marking:** Instance marked "Unhealthy"
3. **Deregistration:** Removed from Target Group (stops receiving traffic)
4. **Termination:** ASG terminates unhealthy instance
5. **Replacement:** ASG launches new instance from Launch Template
6. **Registration:** New instance added to Target Group
7. **Health Check:** New instance passes health checks
8. **Active:** Load Balancer sends traffic to new instance

Time Impact: Typically 3-5 minutes for full replacement

Q8: What is the difference between Launch Configuration and Launch Template?

Answer:

Feature	Launch Configuration	Launch Template
Status	Legacy (not recommended)	Current (AWS recommended)
Versioning	✗ No	✓ Yes

Feature	Launch Configuration	Launch Template
Multiple Instance Types	✗ No	✓ Yes (Mixed Instances)
Spot + On-Demand Mix	✗ No	✓ Yes
T2/T3 Unlimited	✗ No	✓ Yes
Placement Groups	✗ No	✓ Yes
Capacity Reservations	✗ No	✓ Yes

Key Point: Launch Configurations cannot be modified after creation; Launch Templates support versioning.

Q9: Can ASG work without a Load Balancer? When would you use ASG alone?

Answer:

Yes, ASG can work independently without a Load Balancer.

Use cases for ASG without Load Balancer:

- Batch processing workers (pull jobs from SQS)
- Background job processing
- Scheduled tasks
- Single-purpose workers that don't need traffic distribution

When to use ASG with Load Balancer:

- Web applications (most common)
 - APIs
 - Any service receiving direct user traffic
 - Microservices
-

Q10: How do you implement zero-downtime deployment with ALB and ASG?

Answer:

Several strategies:

1. Rolling Deployment:

- ASG launches new instances with new version
- Waits for health checks to pass
- Gradually terminates old instances
- No manual intervention required

2. Blue-Green Deployment:

- Create new ASG with new version (Green)
- Test Green environment
- Switch ALB to point to Green target group
- Keep Blue for rollback

3. Canary Deployment:

- Route small % of traffic to new version
- Monitor for errors
- Gradually increase traffic
- Roll back if issues detected

Q11: What is Connection Draining (Deregistration Delay)?

Answer:

Connection Draining ensures in-flight requests complete before an instance is deregistered.

How it works:

1. Instance marked for deregistration
2. Stop sending NEW requests to this instance
3. Allow EXISTING requests to complete (up to timeout)
4. After timeout, forcefully close connections
5. Instance fully deregistered

Default: 300 seconds (5 minutes)

Best Practice: Set based on longest expected request time

Q12: What is Sticky Sessions (Session Affinity) and when should you use it?

Answer:

Sticky Sessions route all requests from a user to the same instance.

How it works:

- ALB generates a cookie (AWSALB)
- Cookie sent to client
- Subsequent requests include cookie
- ALB routes to same target

When to use:

- Stateful applications (shopping carts stored in memory)
- Legacy apps not designed for load balancing
- WebSocket connections

When NOT to use:

- Stateless applications (use Redis/DynamoDB for state)
- High availability requirements (sticky = single point of failure)

Q13: How does ALB handle HTTPS traffic?

Answer:

Option 1: SSL Termination at ALB (Recommended)

Client → HTTPS → ALB → HTTP → EC2 (Port 80)

- ALB handles SSL certificate
- Reduced CPU load on EC2
- Easier certificate management (ACM)

Option 2: SSL Passthrough (End-to-End)

Client → HTTPS → NLB → HTTPS → EC2 (Port 443)

- Use NLB for passthrough
- EC2 handles SSL
- Required for compliance that mandates end-to-end encryption

Q14: What are Cross-Zone Load Balancing and its implications?

Answer:

Without Cross-Zone:

- Each AZ's load balancer only sends traffic to instances in its AZ
- Uneven distribution if AZs have different instance counts

With Cross-Zone (Default for ALB):

- Traffic distributed evenly across ALL instances in ALL AZs
- Better distribution regardless of instance count per AZ

Example: 3 AZs with 2, 2, and 6 instances

Without Cross-Zone: AZ-c gets 10% per instance (underutilized)

With Cross-Zone: All instances get 10% each (balanced)

Cost: ALB: Free | NLB: Charges for cross-zone data

Q15: How do you troubleshoot if instances are showing unhealthy in Target Group?

Answer:

Systematic troubleshooting:

1. Check Security Groups:

- ALB SG allows outbound to instance port
- Instance SG allows inbound from ALB SG

2. Verify Health Check Path:

- Path exists (/health, /, /status)
- Returns HTTP 200

3. Check Application:

- App running on correct port
- App responding to requests

4. Network Issues:

- Instance in correct subnet
- NACL allows traffic
- Route tables configured

5. Health Check Settings:

- Timeout not too short
- Interval appropriate
- Thresholds reasonable

Quick Test: SSH to instance and curl localhost:port/health-path

Quick Reference Summary

Load Balancer Types

ALB (Layer 7) → HTTP/HTTPS → Web Apps
NLB (Layer 4) → TCP/UDP → Gaming/Real-time
GWLB (Layer 3) → IP → Firewall/Security

ASG Key Settings

Minimum: Floor (never below)
Desired: Target (maintain this)
Maximum: Ceiling (never above)

ASG + LB Integration Steps

1. Create Launch Template (AMI, Instance Type, Security Groups)
2. Create Target Group (Health Checks)
3. Create Load Balancer (Attach Target Group)
4. Create Auto Scaling Group (Attach Load Balancer)

Scaling Policy Priority

Target Tracking > Step Scaling > Simple Scaling
(Recommended) (Flexible) (Basic)

Key Takeaways

1. **Load Balancer** provides high availability and fault tolerance
2. **ALB** is best for HTTP/HTTPS web applications
3. **NLB** is best for ultra-low latency requirements
4. **Target Groups** logically group resources for routing
5. **Health Checks** ensure only healthy instances receive traffic
6. **ASG** automatically scales instances based on demand
7. **Launch Templates** define instance configuration
8. **Multiple AZs** are essential for high availability

9. **Target Tracking** is the recommended scaling policy
 10. **Always use ASG with Load Balancer** for production
-

End of Notes - File 04

AWS Zero to Hero - Notes 05

S3 Storage Classes, Lifecycle Policies, Snow Family & RDS

Table of Contents

1. [S3 Storage Classes Overview](#)
 2. [S3 Standard](#)
 3. [S3 Standard-IA \(Infrequent Access\)](#)
 4. [S3 One Zone-IA](#)
 5. [S3 Glacier Storage Classes](#)
 6. [S3 Intelligent Tiering](#)
 7. [S3 Lifecycle Policies](#)
 8. [AWS Snow Family](#)
 9. [AWS Storage Gateway](#)
 10. [Amazon RDS Introduction](#)
 11. [RDS Database Engines](#)
 12. [RDS Instance Configuration](#)
 13. [RDS Multi-AZ Deployment](#)
 14. [RDS Read Replicas](#)
 15. [Important Q&A](#)
-

1. S3 Storage Classes Overview

What are Storage Classes?

Storage classes in Amazon S3 allow you to optimize costs based on data access patterns and retention requirements.

In-Depth Explanation: The Library Analogy

Think of S3 Storage Classes like **sections of a library**:

Library Section	S3 Storage Class	Description
Open shelves	S3 Standard	Books you access daily, always visible
Back room	Standard-IA	Books requested occasionally, takes a minute to get
Basement storage	Glacier	Archived books, takes hours to retrieve
Off-site warehouse	Deep Archive	Rarely-needed historical records, takes days
Smart sorting	Intelligent-Tiering	Library auto-moves books based on popularity

Why Multiple Storage Classes Matter (Cost Example)

Storing 10 TB of data for 1 month:

Storage Class	Monthly Cost	Savings vs Standard
S3 Standard	\$230	-
Standard-IA	\$125	46%
One Zone-IA	\$100	57%
Glacier Instant	\$40	83%
Glacier Deep Archive	\$10	96%

Choosing wrong class = Wasting money OR losing accessibility!

Key Selection Criteria

Factor	Consideration
Access Frequency	How often will data be accessed?
Retrieval Time	How quickly do you need the data?

Factor	Consideration
Retention Period	How long will data be stored?
Compliance	Are there regulatory requirements?
Cost	What's the budget constraint?

2. S3 Standard

Characteristics

- **Use Case:** Frequently accessed data
- **Availability:** 99.99%
- **Durability:** 99.999999999% (11 9's)
- **Minimum Storage Duration:** None
- **Retrieval Fee:** None

When to Use

S3 STANDARD USE CASES

- Active application data
- Content distribution
- Big data analytics
- Mobile/gaming applications
- Data that's accessed multiple times per day

Pricing Model

- Pay for storage used (GB/month)
- No retrieval charges
- Request charges apply
- Most expensive storage class

3. S3 Standard-IA (Infrequent Access)

Characteristics

- **Use Case:** Data accessed less frequently but needs rapid access when required
- **Availability:** 99.9%
- **Minimum Storage Duration:** 30 days
- **Retrieval Fee:** Per GB retrieved

Cost Structure

S3 STANDARD-IA COSTING	
Storage Cost:	Lower than S3 Standard
Retrieval Cost:	Charged per GB
Best for:	<ul style="list-style-type: none">• Backups• DR data• Older archives
Minimum:	30 days storage (charged even if deleted early)

Important Note

- If you delete data before 30 days, you're still charged for 30 days minimum
- Ideal for data accessed once per month or less

4. S3 One Zone-IA

Characteristics

- **Use Case:** Infrequently accessed data that doesn't require multi-AZ resilience
- **Availability:** 99.5%
- **Storage:** Single Availability Zone only

- **Minimum Storage Duration:** 30 days

Risk vs Cost Trade-off

S3 ONE ZONE-IA	
SINGLE AZ	← Data stored in ONE zone only
Data	If AZ fails = DATA LOST!
✓ 20% cheaper than Standard-IA	
X No cross-AZ redundancy	
Best For:	
<ul style="list-style-type: none"> • Secondary backup copies • Data that can be recreated • Non-critical data 	

5. S3 Glacier Storage Classes

Overview

Glacier classes are designed for long-term archival storage with varying retrieval times.

Glacier Classes Comparison

Class	Retrieval Time	Use Case	Min Storage
Glacier Instant	Milliseconds	Archive needing instant access	90 days
Glacier Flexible	Minutes to 12 hours	General archival	90 days
Glacier Deep Archive	12-48 hours	Long-term compliance	180 days

Glacier Deep Archive

GLACIER DEEP ARCHIVE

CHEAPEST STORAGE in AWS S3

Ideal For:

- 10+ year data retention
- Compliance/regulatory archives
- Data rarely (if ever) accessed
- Financial records
- Healthcare records

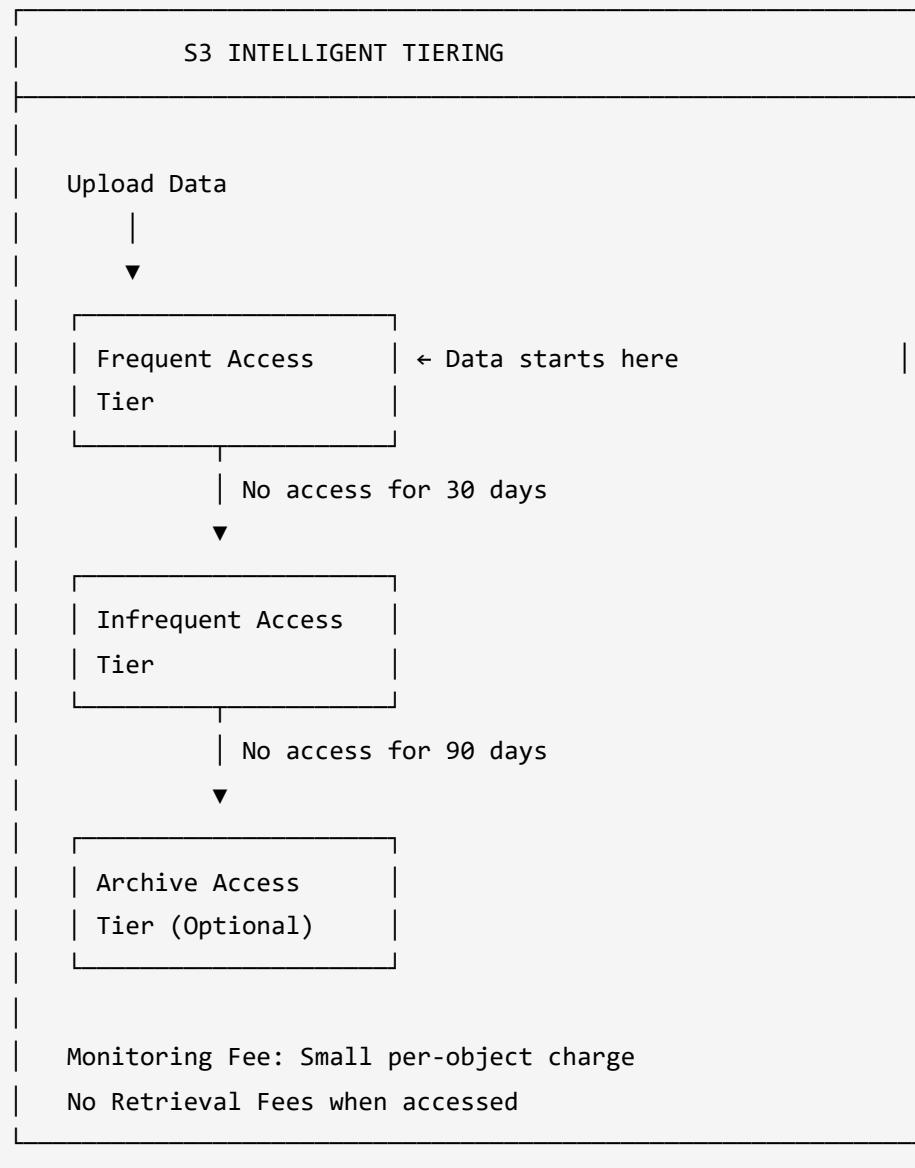
Retrieval: 12-48 hours (plan ahead!)

Cost: ~\$1 per TB per month

6. S3 Intelligent Tiering

How It Works

S3 Intelligent Tiering automatically moves data between access tiers based on access patterns.



Transition Timeline

- **Day 0-30:** Frequent Access Tier
- **Day 30-60:** Standard-IA equivalent
- **Day 90+:** Can transition to Archive tier

Best Use Case

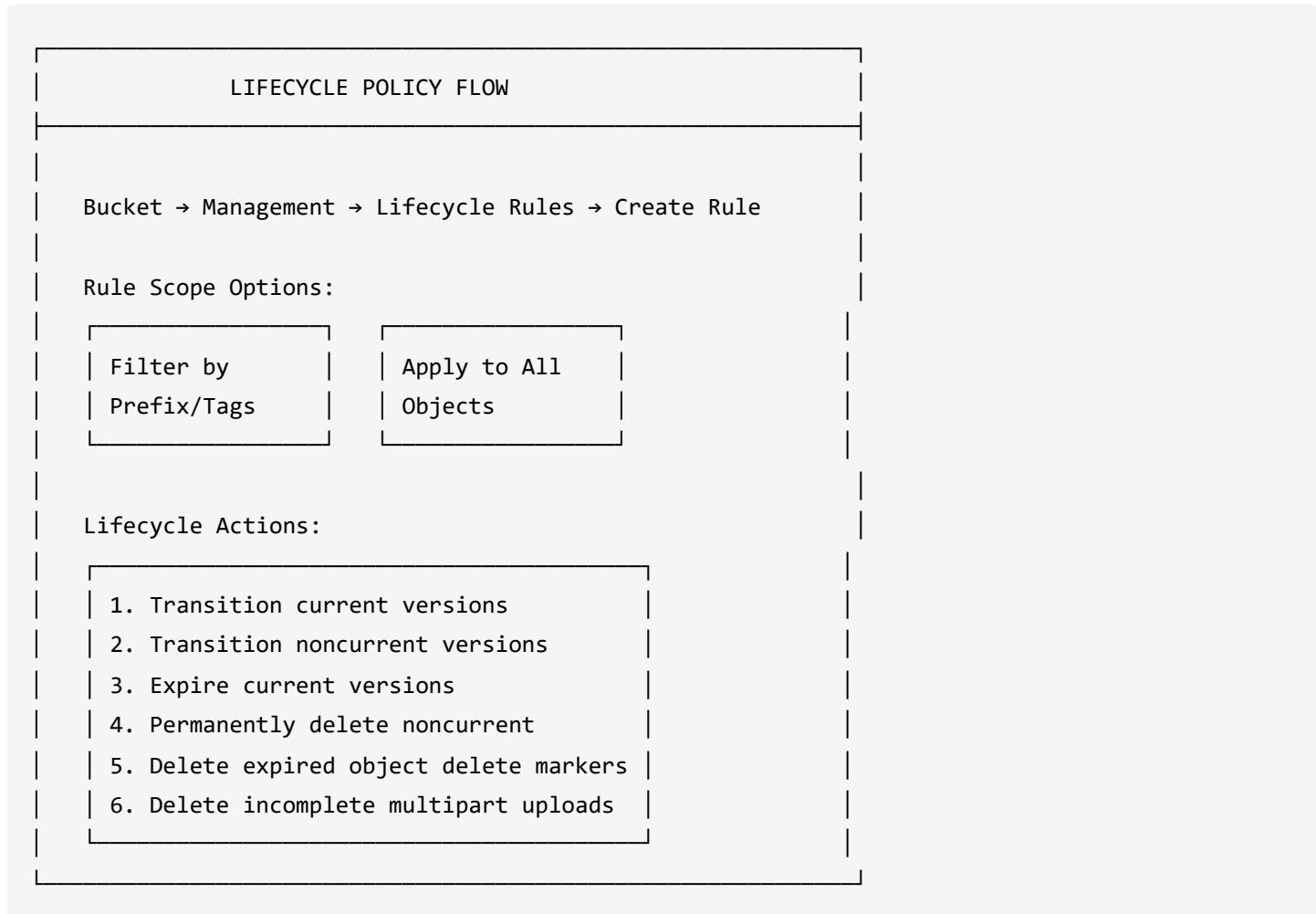
- When you're unsure of access patterns
- Unpredictable workloads
- Changing data access patterns

7. S3 Lifecycle Policies

What are Lifecycle Policies?

Automated rules that transition objects between storage classes or delete them based on specified conditions.

Creating Lifecycle Rules



Example Lifecycle Configuration

Lifecycle Rule Example:

Name: "archive-old-data"

Scope: All objects in bucket

Transitions:

- After 30 days → Standard-IA
- After 60 days → Glacier Flexible
- After 180 days → Glacier Deep Archive

Expiration:

- After 365 days → Delete permanently

Transition Actions

Action	Description
Expire Current Versions	Delete objects after X days
Transition Current Versions	Move to different storage class
Permanently Delete	Remove noncurrent versions

8. AWS Snow Family

Overview

Physical devices for large-scale data transfer when network transfer is impractical.

Why Snow Family?

WHY USE SNOW FAMILY?

Problem: 10 TB data to upload to S3

Internet Transfer:

- | 10 TB over 100 Mbps = ~10+ DAYS!
- | Network costs, reliability issues
- | Bandwidth limitations

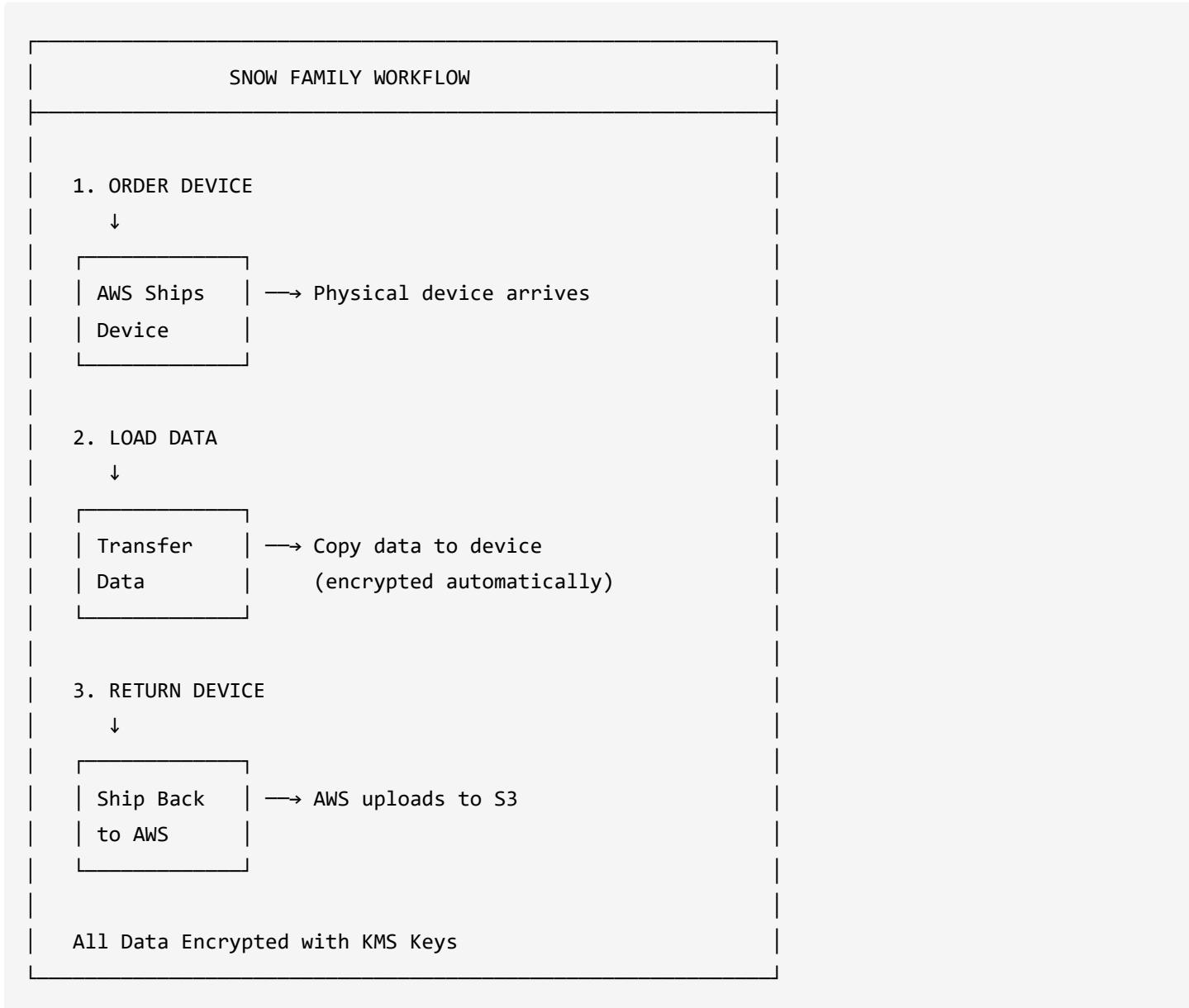
Snow Device Transfer:

- | Ship device → Load data → Ship back
- | Days instead of weeks/months
- | Secure, encrypted transfer

Snow Family Devices

Device	Capacity	Use Case
Snowcone	8-14 TB	Small, portable, edge computing
Snowball	80 TB (HDD)	Medium-scale data migration
Snowball Edge	80-210 TB	Compute + Storage, edge computing
Snowmobile	100 PB	Massive data center migrations

Snow Device Architecture



Snowball Edge Variations

- **Storage Optimized:** Maximum storage capacity
- **Compute Optimized:** More CPU/memory for edge computing

Snowmobile

SNOWMOBILE

[======TRUCK=====]

- 45-foot shipping container
- 100 PETABYTES capacity
- Used for data center migrations
- GPS tracking, 24/7 security
- Typically used by large enterprises

When to use: Exabyte-scale migrations

9. AWS Storage Gateway

Overview

Hybrid cloud storage service connecting on-premises environments to AWS cloud storage.

Types of Storage Gateway

STORAGE GATEWAY TYPES

FILE GATEWAY

→ NFS/SMB → S3 Buckets

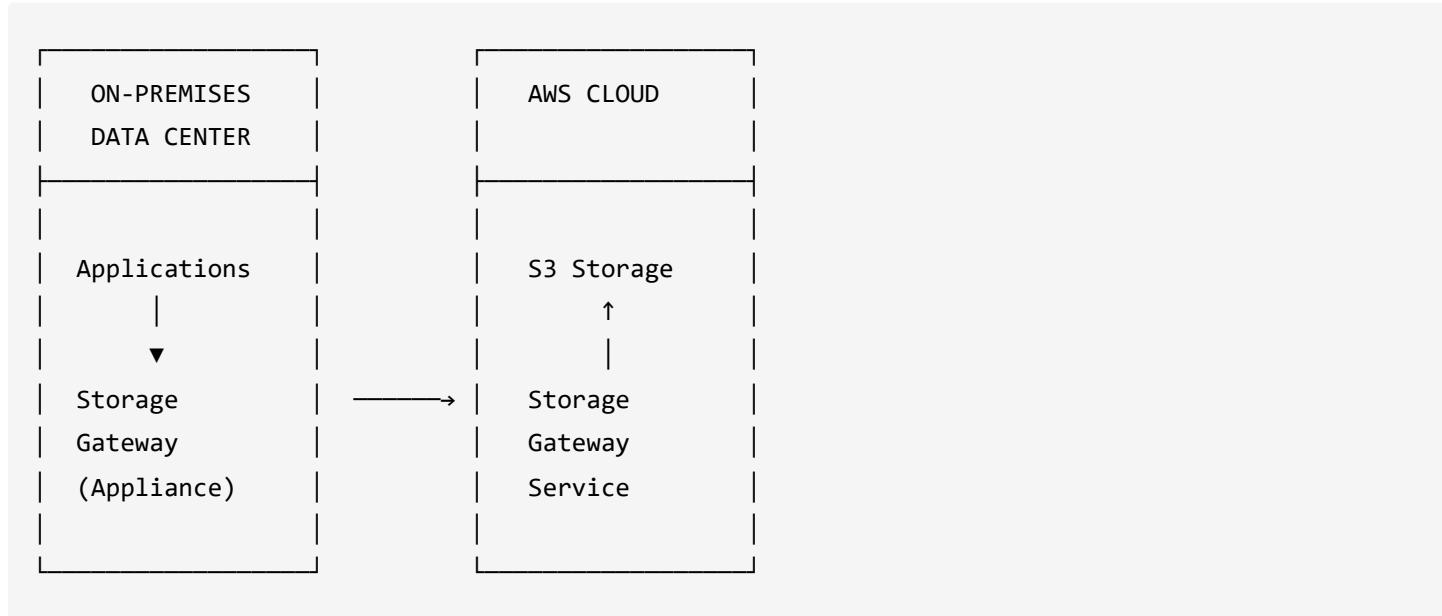
VOLUME GATEWAY

→ iSCSI → EBS Snapshots

TAPE GATEWAY

→ Virtual Tapes → S3 Glacier

Use Case: Hybrid Cloud Integration

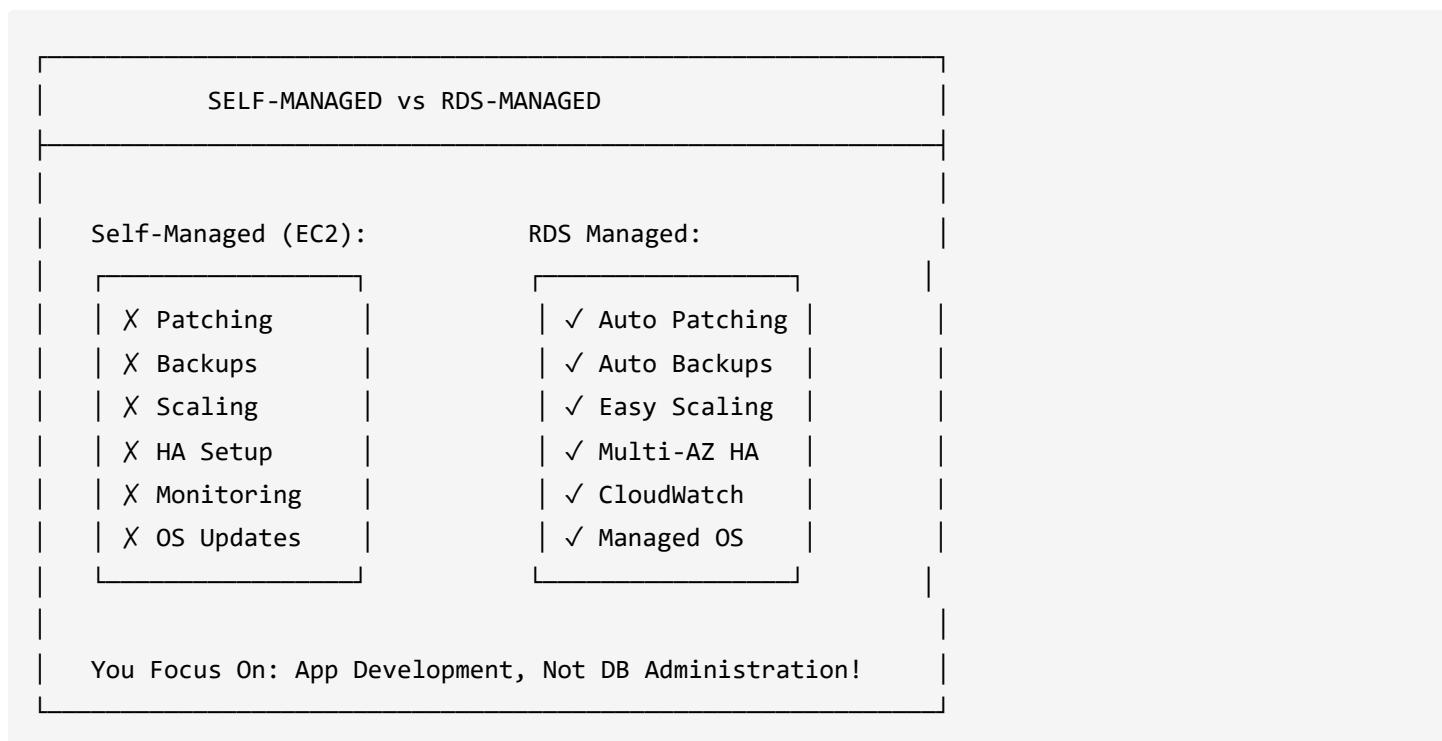


10. Amazon RDS Introduction

What is RDS?

Amazon RDS (Relational Database Service) is a managed database service that handles database administration tasks.

Why Use RDS?

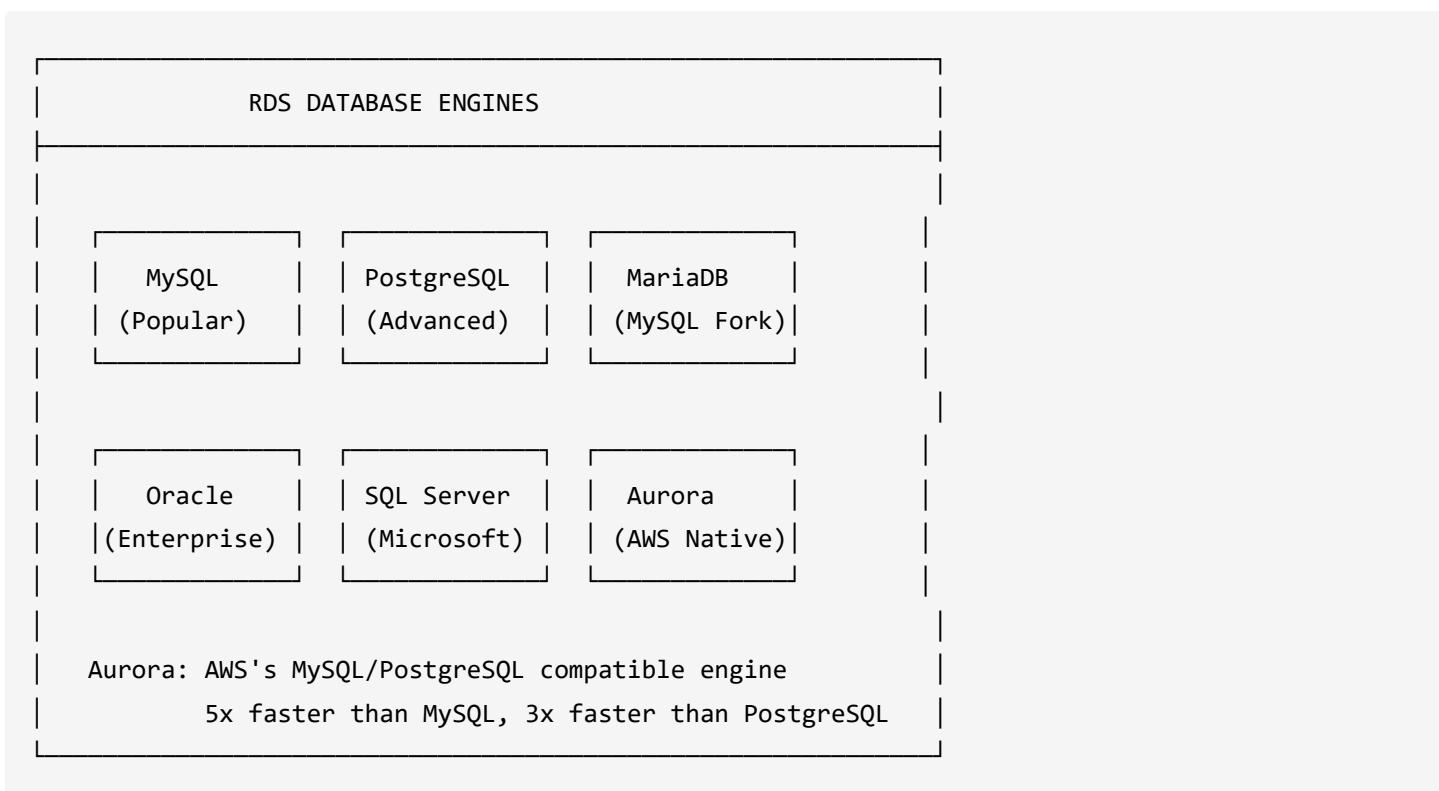


Relational Database Concepts

- **Tables:** Structured data in rows and columns
- **Relationships:** Foreign keys linking tables
- **SQL:** Query language for data manipulation
- **ACID:** Atomicity, Consistency, Isolation, Durability

11. RDS Database Engines

Supported Engines



Engine Selection Guide

Engine	Best For
MySQL	General-purpose, web applications
PostgreSQL	Complex queries, GIS data
MariaDB	MySQL alternative, community-driven
Oracle	Enterprise applications, legacy systems
SQL Server	Microsoft ecosystem integration
Aurora	High performance, scalability

12. RDS Instance Configuration

Creating RDS Instance

RDS INSTANCE CREATION

Step 1: Choose Database Engine

- Standard Create / Easy Create
- Select: MySQL, PostgreSQL, etc.

Step 2: Choose Template

- Production (High Availability)
- Dev/Test (Lower Cost)
- Free Tier (Learning)

Step 3: Instance Configuration

- DB Instance Class (CPU/Memory)
- Storage Type (SSD/IOPS)
- Allocated Storage (GB)

Instance Classes

Instance Class Types:

Class	Description
db.t3.micro	Burstable, Free Tier eligible
db.m5	General purpose, balanced
db.r5	Memory optimized, read-heavy
db.x1	Memory optimized, in-memory databases

Credentials Configuration

Database Credentials:

Master Username: admin (default)
Master Password: Your secure password

Options:

- Self-managed (you set password)
- AWS Secrets Manager (recommended for production)

Storage Configuration

Storage Options:

Storage Type:

General Purpose	Provisioned
SSD (gp2/gp3)	IOPS (io1)

Allocated Storage: 20 GB - 64 TB

Storage Autoscaling: Enable to auto-increase

Maximum Storage Threshold: Set limit (e.g., 1000 GB)

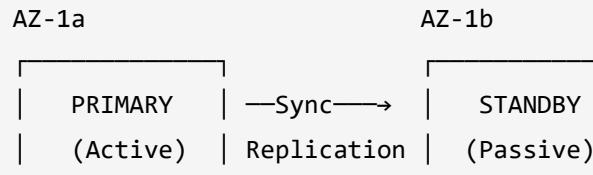
13. RDS Multi-AZ Deployment

What is Multi-AZ?

Multi-AZ provides high availability by maintaining a synchronous standby replica in a different Availability Zone.

RDS MULTI-AZ ARCHITECTURE

Region: ap-south-1



Read/Write Operations → Automatic Failover

Failover: Automatic (60-120 seconds)
DNS: Same endpoint, automatic redirect

Multi-AZ Benefits

Feature	Benefit
Automatic Failover	No manual intervention needed
Synchronous Replication	Zero data loss
Same Endpoint	No application changes required
Automated Backups	From standby, no primary impact

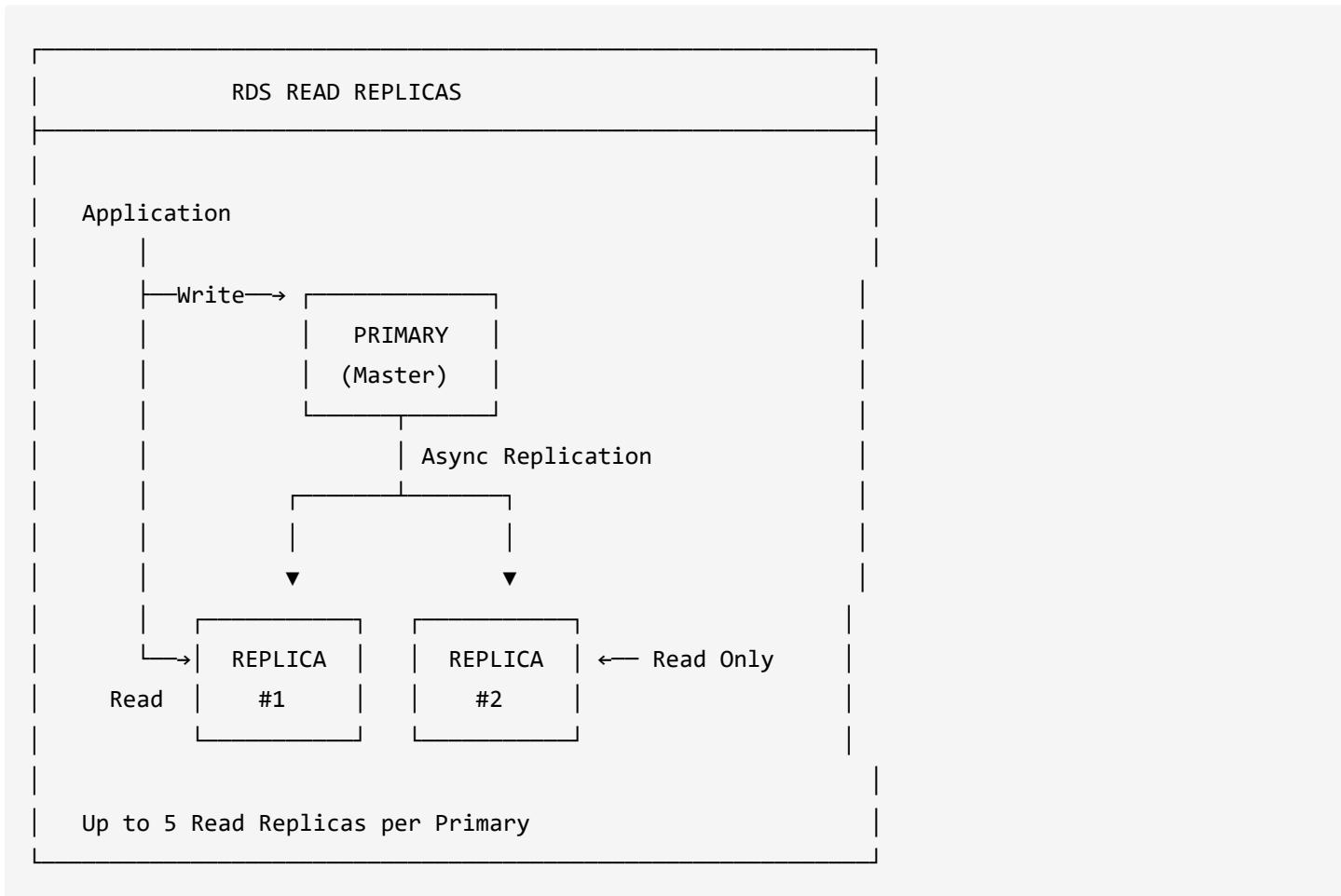
Multi-AZ vs Single Instance

SINGLE INSTANCE	MULTI-AZ
Lower cost	Higher cost (~2x)
Single point failure	High availability
Dev/Test suitable	Production ready
Manual recovery	Auto failover

14. RDS Read Replicas

What are Read Replicas?

Read replicas are read-only copies of your database that help scale read operations.



Read Replica vs Multi-AZ

Feature	Read Replica	Multi-AZ
Purpose	Read scaling	High availability
Replication	Asynchronous	Synchronous
Read Traffic	Yes, serves reads	No, standby only
Cross-Region	Yes	No (same region)
Failover	Manual promotion	Automatic

Use Cases for Read Replicas

1. **Analytics Queries:** Run reports without impacting primary
 2. **Geographic Distribution:** Place replicas closer to users
 3. **Read Scaling:** Distribute read load
 4. **Disaster Recovery:** Cross-region replica
-

15. Real Interview Questions & Answers

Q1: What are the S3 storage classes? Explain when to use each.

Answer:

Storage Class	Access Pattern	Retrieval Time	Min Duration	Use Case
S3 Standard	Frequent	Milliseconds	None	Active data, websites
Standard-IA	Infrequent	Milliseconds	30 days	Backups, DR
One Zone-IA	Infrequent	Milliseconds	30 days	Secondary backups
Glacier Instant	Archive, instant access	Milliseconds	90 days	Medical images, news archives
Glacier Flexible	Archive	1-12 hours	90 days	Long-term backups
Glacier Deep Archive	Rarely accessed	12-48 hours	180 days	Compliance, 10+ year retention
Intelligent-Tiering	Unknown	Milliseconds	None	Unpredictable access

Key Decision: If you don't know access patterns → Use Intelligent-Tiering.

Q2: What is S3 Intelligent-Tiering and how does it work?

Answer:

S3 Intelligent-Tiering automatically moves objects between access tiers based on access patterns:

Automatic Tiers:

1. **Frequent Access** (default) → Standard-like pricing

2. **Infrequent Access** (after 30 days no access) → IA-like pricing
3. **Archive Instant Access** (after 90 days) → Glacier Instant-like
4. **Archive Access** (optional, 90+ days) → Glacier-like
5. **Deep Archive** (optional, 180+ days) → Deep Archive-like

Costs:

- Small monitoring fee per object (~\$0.0025 per 1,000 objects)
- NO retrieval fees
- No minimum storage duration

When to use: Unknown/changing access patterns, datasets with mixed hot/cold data.

Q3: Explain S3 Lifecycle Policies with an example.

Answer:

Lifecycle policies automate object transitions and deletions.

Example: Media Company Workflow

Rule: "media-archive-policy"

Scope: Objects with prefix "videos/"

Transitions:

- Day 0-30: S3 Standard (active editing)
- Day 30: → S3 Standard-IA (finished editing)
- Day 90: → S3 Glacier Instant (reference archive)
- Day 365: → S3 Glacier Deep Archive (compliance)

Expiration:

- Day 2555 (7 years): Permanently delete

Key Actions:

- Transition current versions
- Transition noncurrent versions (for versioned buckets)
- Expire current versions (delete)
- Delete incomplete multipart uploads

Q4: What is the difference between S3 One Zone-IA and S3 Standard-IA?

Answer:

Aspect	S3 Standard-IA	S3 One Zone-IA
AZ Distribution	3+ AZs	Single AZ
Durability	99.999999999% (11 9s)	99.999999999% (within 1 AZ)
Availability	99.9%	99.5%
Cost	Higher	20% cheaper
Risk	Survives AZ failure	Data LOST if AZ fails
Use Case	Primary backups	Secondary copies, recreatable data

Key Point: Never store primary/irreplaceable data in One Zone-IA.

Q5: What is AWS Snow Family? When would you use each device?

Answer:

Device	Capacity	Use Case	Time to Transfer 100 TB
Snowcone	8-14 TB	Edge locations, IoT, portable	Multiple trips
Snowball Edge	80-210 TB	Medium migrations, edge compute	1-2 devices
Snowmobile	100 PB	Data center migrations	Overkill

When to use Snow vs. Internet:

Rule of Thumb:

If upload would take > 1 week on internet → Consider Snow

10 TB at 100 Mbps = ~10 days

100 TB at 100 Mbps = ~100 days → Use Snowball

10 PB = Snowmobile territory

Snow Family Features:

- 256-bit encryption

- Tamper-resistant enclosure
 - End-to-end tracking
 - Compute capability (Snowball Edge)
-

Q6: What is RDS Multi-AZ? How does failover work?

Answer:

Multi-AZ creates a **synchronous standby replica** in another AZ for high availability.

How Failover Works:

1. Primary DB becomes unavailable (failure, maintenance, etc.)
2. AWS detects failure within seconds
3. Standby is automatically promoted to primary
4. DNS endpoint points to new primary
5. Applications reconnect automatically
6. Total failover time: 60-120 seconds

Important Points:

- Synchronous replication = zero data loss
 - Same DNS endpoint (no application changes)
 - Standby CANNOT serve read traffic (use Read Replicas for that)
 - ~2x cost of single instance
-

Q7: What is the difference between RDS Read Replicas and Multi-AZ?

Answer:

Feature	Read Replicas	Multi-AZ
Purpose	Read scaling	High availability
Replication	Asynchronous	Synchronous
Serves Traffic	Yes (read-only)	No (standby only)
Data Loss Risk	Possible (lag)	Zero
Cross-Region	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
Failover	Manual promotion	Automatic

Feature	Read Replicas	Multi-AZ
Endpoint	Separate endpoint	Same endpoint
Count	Up to 5	1 standby
Cost	Per replica	~2x primary

Combined Architecture: Use both for HA + read scaling.

Q8: How do you secure data in S3?

Answer:

1. Access Control:

- **Bucket Policies:** JSON policies for bucket-level access
- **IAM Policies:** User/role-based permissions
- **ACLs:** Legacy, object-level (not recommended)
- **Block Public Access:** Account/bucket-level blocking

2. Encryption:

- **SSE-S3:** AWS-managed keys (default)
- **SSE-KMS:** Customer-managed KMS keys (audit trail)
- **SSE-C:** Customer-provided keys
- **Client-side:** Encrypt before upload

3. Network Security:

- **VPC Endpoints:** Private connectivity
- **Presigned URLs:** Time-limited access
- **HTTPS:** Enforce encryption in transit

4. Auditing:

- **S3 Access Logs:** Request logging
 - **CloudTrail:** API activity logging
-

Q9: What is S3 Versioning? Why is it important?

Answer:

Versioning keeps multiple variants of an object in the same bucket.

Benefits:

- **Accidental Deletion Protection:** Delete creates delete marker, not permanent deletion
- **Recovery:** Restore previous versions
- **Audit Trail:** Track changes over time

How it works:

Object: report.pdf

Version 1: Original upload

Version 2: Modified version

Version 3: Latest version

Delete operation → Creates "Delete Marker"

→ Versions still exist

→ Can restore by removing delete marker

MFA Delete: Require MFA to permanently delete versions (extra protection).

Q10: What RDS database engines are supported? Which should you choose?

Answer:

Engine	Best For	Key Features
MySQL	Web apps, general purpose	Most popular, mature
PostgreSQL	Complex queries, GIS	Advanced features, JSONB
MariaDB	MySQL alternative	Open-source, community
Oracle	Enterprise, legacy	OLTP, compatibility
SQL Server	Microsoft ecosystem	.NET integration
Aurora	High performance	5x MySQL, 3x PostgreSQL speed

Aurora Specifics:

- AWS proprietary, MySQL/PostgreSQL compatible
- Auto-scaling storage (up to 128 TB)
- 6 copies across 3 AZs
- Serverless option available

Q11: How do you connect to an RDS database from an EC2 instance?

Answer:

Prerequisites:

1. **Same VPC:** EC2 and RDS in same VPC (or VPC peering)
2. **Security Group:** RDS SG allows inbound from EC2 SG
3. **Network:** Correct subnet routing

Security Group Configuration:

RDS Security Group:

Inbound Rule:

Type: MySQL/Aurora (or your engine)
Port: 3306 (or engine port)
Source: EC2 Security Group ID (sg-xxxxx)

Connection Command:

```
# Install client
sudo yum install mysql -y

# Connect
mysql -h <rds-endpoint>.rds.amazonaws.com -u admin -p
```

Common Issues:

- Security group not allowing traffic
- RDS not publicly accessible (if connecting from internet)
- Wrong VPC/subnet configuration

Q12: Explain S3 Transfer Acceleration.

Answer:

S3 Transfer Acceleration uses **CloudFront edge locations** to speed up uploads.

How it works:

1. Client uploads to nearest CloudFront edge
2. Data travels over AWS backbone (optimized network)
3. Reaches S3 bucket faster than public internet

Use Cases:

- Users uploading from distant locations
- Large file uploads
- Regular uploads from various global locations

Cost: Additional per-GB charge only when faster than regular upload.

Enable: Bucket → Properties → Transfer Acceleration

URL Change: `bucketname.s3-accelerate.amazonaws.com`

Q13: What is RDS automated backup vs manual snapshot?

Answer:

Feature	Automated Backup	Manual Snapshot
Creation	Automatic (daily)	User-initiated
Retention	1-35 days (configurable)	Until deleted
Point-in-Time Recovery	<input checked="" type="checkbox"/> Yes (5-minute granularity)	<input type="checkbox"/> No (point of snapshot only)
Performance Impact	During backup window	Can impact I/O
Cost	Free up to DB size	Pay for storage
Deletion Behavior	Deleted with instance	Persists after deletion

Best Practice: Enable automated backups + take manual snapshots before major changes.

Q14: What is the minimum storage duration charge for Glacier Deep Archive?

Answer:

180 days minimum.

If you delete an object before 180 days, you're charged for the full 180 days.

All Minimum Storage Durations:

- S3 Standard: None
- Standard-IA: 30 days

- One Zone-IA: 30 days
- Glacier Instant: 90 days
- Glacier Flexible: 90 days
- Glacier Deep Archive: **180 days**
- Intelligent-Tiering: None

Planning Tip: Only use Deep Archive for data you're sure to keep 180+ days.

Q15: How do you migrate a large database to RDS?

Answer:

Option 1: AWS DMS (Database Migration Service)

- Supports heterogeneous migrations (Oracle → MySQL)
- Minimal downtime (continuous replication)
- Schema conversion tool available

Option 2: Native Database Tools

```
# MySQL example
# Export from source
mysqldump -h source-db -u user -p database > backup.sql

# Import to RDS
mysql -h rds-endpoint -u admin -p database < backup.sql
```

Option 3: AWS Snowball (for very large databases)

- Physical transfer for multi-TB databases
- Use when network transfer would take too long

Best Practices:

- Test migration in non-production first
 - Plan maintenance window
 - Validate data integrity post-migration
 - Keep source available for rollback
-

Quick Reference Card

S3 & RDS QUICK REFERENCE

S3 STORAGE CLASSES (Cost: High → Low)

Standard → Standard-IA → One Zone-IA → Glacier → Deep

LIFECYCLE MINIMUM DURATIONS

- Standard-IA: 30 days
- One Zone-IA: 30 days
- Glacier: 90 days
- Deep Archive: 180 days

SNOW FAMILY (Data Transfer)

- Snowcone: 8-14 TB (portable)
- Snowball Edge: 80-210 TB (medium)
- Snowmobile: 100 PB (massive)

RDS HIGH AVAILABILITY

- Multi-AZ: Sync replication, auto failover
- Read Replicas: Async, read scaling, up to 5

RDS ENGINES

MySQL | PostgreSQL | MariaDB | Oracle | SQL Server | Aurora

FREE TIER RDS

- db.t3.micro (or db.t2.micro)
- 750 hours/month
- 20 GB storage

End of Notes 05 - S3 Storage Classes, Lifecycle Policies, Snow Family & RDS

AWS Zero to Hero - Notes 06: DynamoDB,

AWS Lambda & CloudWatch

Table of Contents

1. [Amazon DynamoDB Overview](#)
 2. [DynamoDB Data Model](#)
 3. [DynamoDB DAX \(Accelerator\)](#)
 4. [DynamoDB Global Tables](#)
 5. [AWS Lambda Introduction](#)
 6. [Lambda Function Components](#)
 7. [Lambda Execution Model](#)
 8. [Lambda Triggers and Events](#)
 9. [Lambda IAM Roles and Permissions](#)
 10. [Lambda Practical Implementation](#)
 11. [Amazon CloudWatch Overview](#)
 12. [CloudWatch Logs](#)
 13. [CloudWatch Monitoring](#)
 14. [Important Q&A](#)
-

1. Amazon DynamoDB Overview

What is DynamoDB?

DynamoDB is a fully managed **NoSQL database service** provided by AWS that offers:

- Single-digit millisecond performance
- Flexible data model (key-value and document)
- Automatic scaling
- Built-in security and backup

In-Depth Explanation: The Library Card Catalog Analogy

Think of DynamoDB like a **super-fast library card catalog system**:

Library Concept	DynamoDB Equivalent
Card Catalog Drawer	Partition (data grouped together)
Card	Item (single record)

Library Concept	DynamoDB Equivalent
Card's Book ID	Partition Key (unique identifier)
Alphabetical order within drawer	Sort Key (ordering within partition)
Information on card	Attributes (flexible fields)
Finding a specific card	GetItem (instant lookup)
Browsing cards in a drawer	Query (scan within partition)

Why DynamoDB vs Relational (RDS)?

Use RDS when you need:

- Complex relationships (JOINS between tables)
- ACID transactions across multiple tables
- SQL querying flexibility
- Well-defined, unchanging schema

Use DynamoDB when you need:

- Millisecond response at any scale
- Flexible schema (different items can have different attributes)
- Automatic scaling without management
- Simple key-value or document access patterns

DynamoDB vs Traditional Databases

DATABASE COMPARISON	
RELATIONAL (RDS)	NoSQL (DynamoDB)
Fixed Schema	Flexible Schema
SQL Queries	API-based Queries
ACID Transactions	Eventually Consistent (configurable)
Vertical Scaling	Horizontal Scaling
Complex Joins	No Joins (denormalized data)
Good for Complex Relationships	Good for High-throughput, Low-latency Applications

Key Features

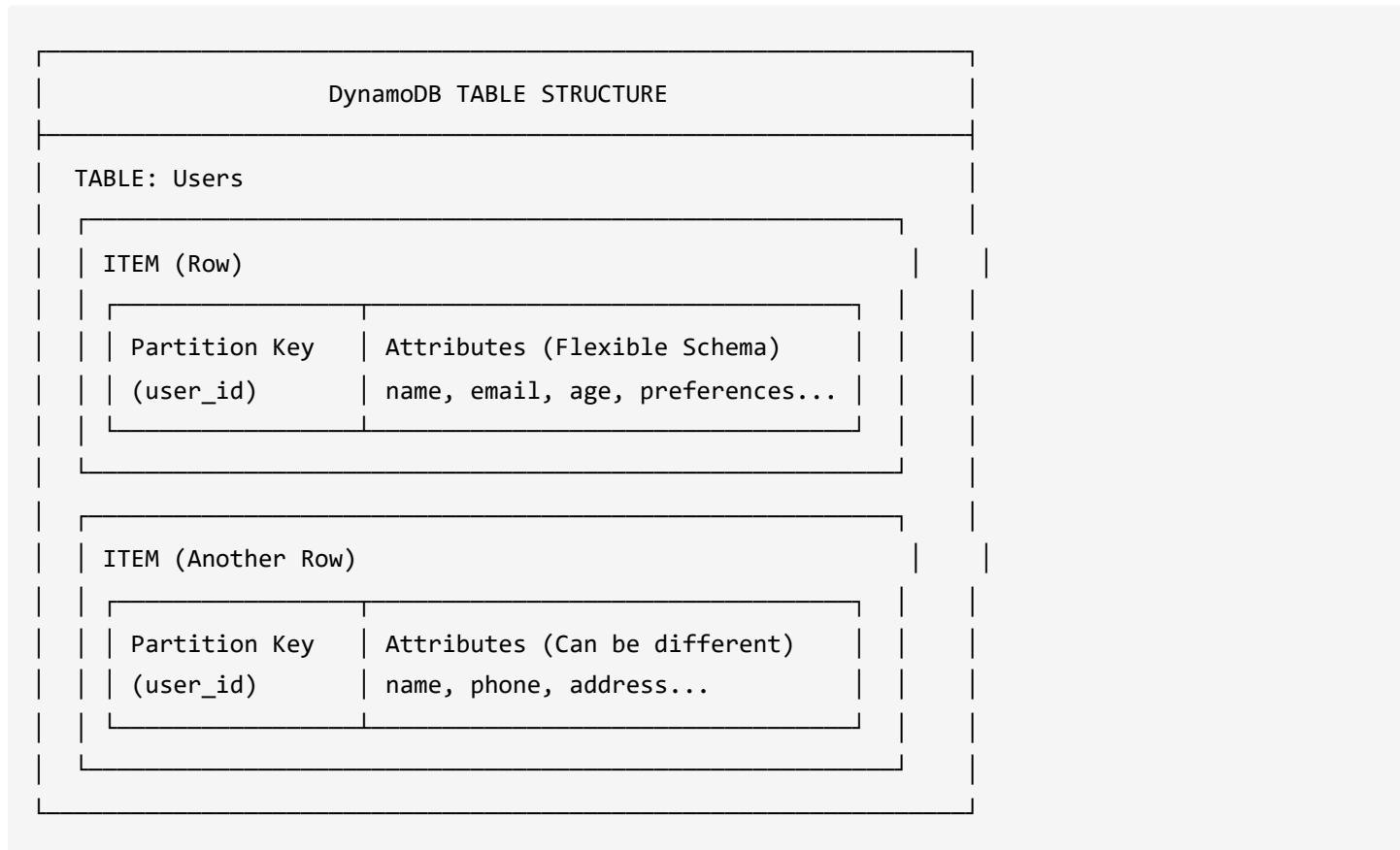
Feature	Description
Performance	Single-digit millisecond latency
Scalability	Automatic scaling based on demand
Availability	Multi-AZ by default
Security	IAM integration, encryption at rest
Backup	Point-in-time recovery, on-demand backup

Use Cases for DynamoDB

- Mobile and Web Applications
 - Gaming (leaderboards, session state)
 - IoT (high-volume sensor data)
 - Real-time analytics
 - Content management
 - E-commerce (shopping carts, user profiles)
-

2. DynamoDB Data Model

Core Components



Key Types

Key Type	Description	Example
Partition Key	Primary key (unique identifier)	<code>user_id</code> , <code>order_id</code>
Sort Key	Optional secondary key for ordering	<code>timestamp</code> , <code>order_number</code>
Composite Key	Partition Key + Sort Key combination	<code>user_id</code> + <code>order_date</code>

Creating a DynamoDB Table

```
# Using AWS CLI
aws dynamodb create-table \
--table-name Users \
--attribute-definitions \
  AttributeName=user_id,AttributeType=S \
--key-schema \
  AttributeName=user_id,KeyType=HASH \
--provisioned-throughput \
  ReadCapacityUnits=5,WriteCapacityUnits=5
```

DynamoDB Operations

Operation	Description	CLI Command
PutItem	Insert/replace item	aws dynamodb put-item
GetItem	Retrieve single item	aws dynamodb get-item
Query	Retrieve items with same partition key	aws dynamodb query
Scan	Read entire table	aws dynamodb scan
UpdateItem	Modify existing item	aws dynamodb update-item
DeleteItem	Remove item	aws dynamodb delete-item

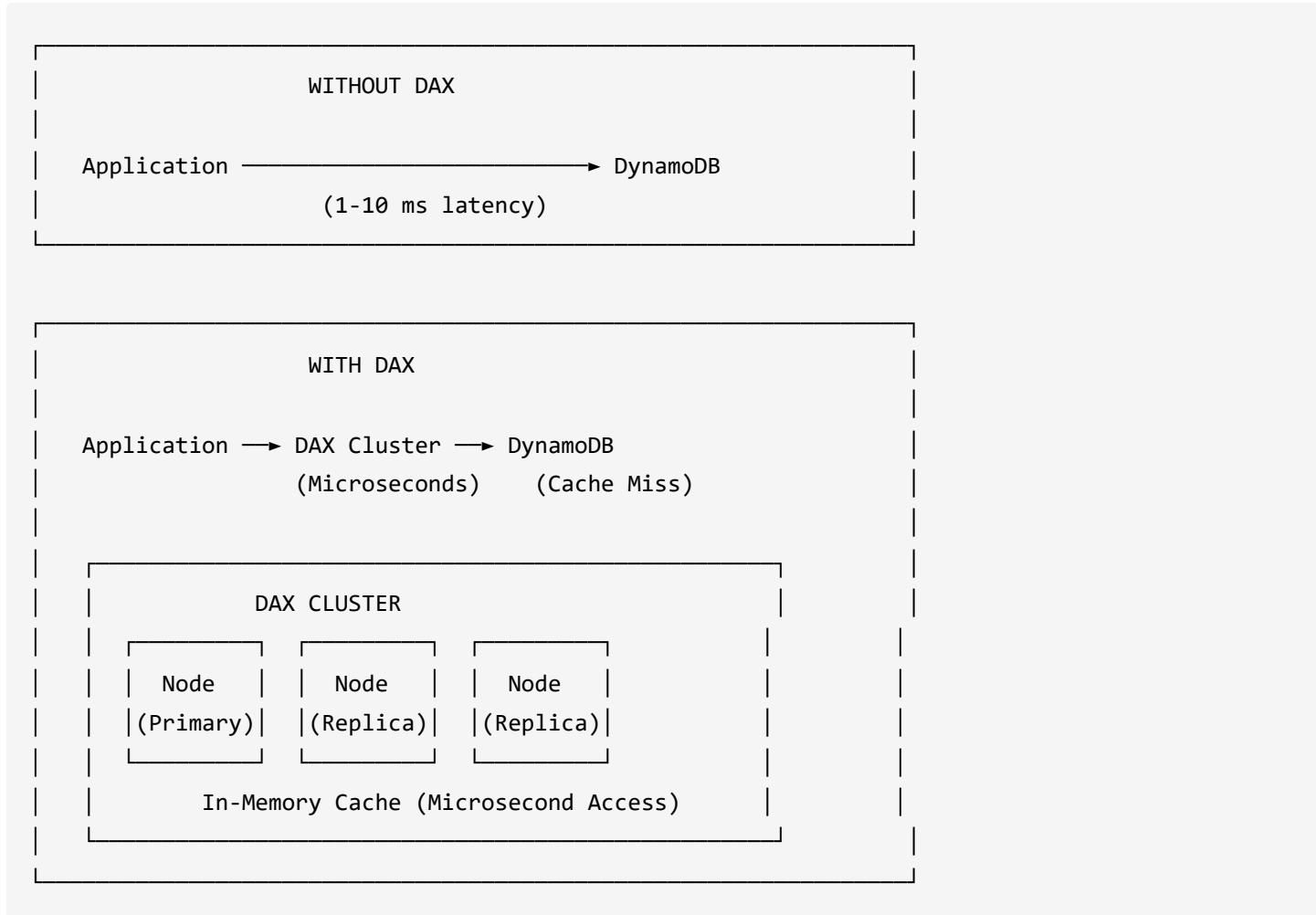
3. DynamoDB DAX (Accelerator)

What is DAX?

DynamoDB Accelerator (DAX) is a fully managed, in-memory cache for DynamoDB that provides:

- 10x performance improvement (microseconds instead of milliseconds)
- No application code changes required
- Fully managed (no cache management needed)

DAX Architecture



DAX Use Cases

- Read-heavy workloads
- Applications requiring consistent response times
- Gaming leaderboards
- Real-time bidding platforms
- Social media feeds

DAX vs ElastiCache

Feature	DAX	ElastiCache
Purpose	DynamoDB-specific caching	General-purpose caching
Integration	Seamless with DynamoDB	Requires application changes
API Compatibility	DynamoDB API compatible	Redis/Memcached APIs
Use Case	DynamoDB acceleration only	Any caching scenario

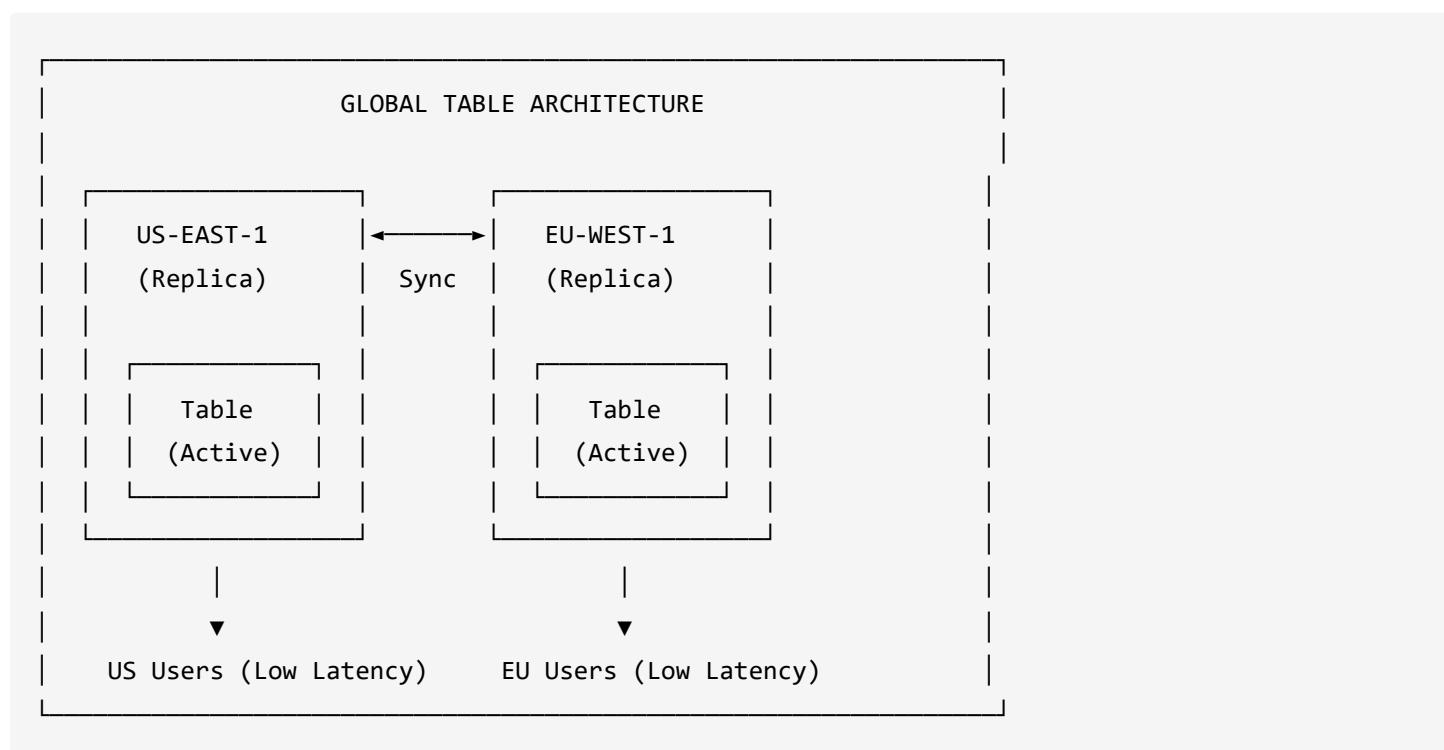
4. DynamoDB Global Tables

What are Global Tables?

Global Tables provide **multi-region, multi-active replication** for DynamoDB, allowing:

- Global users with low-latency access
- Disaster recovery across regions
- Active-active database architecture

Global Table Architecture



Key Benefits

Benefit	Description
Multi-Region	Data replicated across multiple AWS regions
Active-Active	Read/write from any region
Low Latency	Users access nearest region
Disaster Recovery	Automatic failover capability
Conflict Resolution	Last-writer-wins for concurrent updates

Creating Global Table (Console Steps)

1. Create a DynamoDB table with streams enabled
 2. Go to Global Tables tab
 3. Click "Create replica"
 4. Select target region
 5. Wait for synchronization
-

5. AWS Lambda Introduction

What is AWS Lambda?

AWS Lambda is a **serverless compute service** that:

- Runs code without provisioning servers
- Automatically scales based on requests
- Charges only for compute time used
- Supports multiple programming languages

In-Depth Explanation: The "Kitchen Staff" Analogy

Think of Lambda like **on-demand kitchen staff**:

Traditional (EC2)	Serverless (Lambda)
Hire full-time chefs	Order food from service
Pay salary even when no orders	Pay only when cooking
You manage kitchen equipment	Kitchen is managed for you
Limited by kitchen size	Infinite kitchens available
Chefs sit idle during slow times	Staff appears only when needed

When to Use Lambda vs EC2

Scenario	Lambda	EC2
Short tasks (<15 min)	<input checked="" type="checkbox"/> Perfect	Overkill
Bursty workloads	<input checked="" type="checkbox"/> Auto-scales instantly	Takes time to scale
Always-on applications	<input checked="" type="checkbox"/> Can be expensive	<input checked="" type="checkbox"/> Better cost

Scenario	Lambda	EC2
Heavy computation	✗ Memory/time limits	✓ Full control
Event-driven processing	✓ Built for this	Possible but complex
Container/custom runtime	✓ Supported	✓ Full flexibility

Lambda Limits (Important for Interviews!)

Limit	Value
Execution timeout	15 minutes max
Memory	128 MB to 10,240 MB
Deployment package	50 MB (zipped), 250 MB (unzipped)
Environment variables	4 KB total
Concurrent executions	1,000 per region (default, can increase)
Temp storage (/tmp)	512 MB - 10 GB

Serverless Computing Concept

TRADITIONAL vs SERVERLESS

TRADITIONAL (EC2):

You Manage:

- Server provisioning
- OS patching
- Scaling configuration
- Always running (paying even when idle)

SERVERLESS (Lambda):

AWS Manages:

- All infrastructure
- Auto-scaling
- High availability
- Pay only when code executes

You Focus:

- Just upload your code!

Supported Programming Languages

Language	Runtime
Python	python3.9, python3.10, python3.11
Node.js	nodejs16.x, nodejs18.x, nodejs20.x
Java	java11, java17, java21
Go	provided.al2
.NET	dotnet6, dotnet8
Ruby	ruby3.2
Custom	provided.al2 (Any language)

Lambda Pricing Model

LAMBDA PRICING

Pay Per Use:

- Number of requests
- Duration of execution (in GB-seconds)
- Memory allocated

Free Tier (Monthly):

- 1 million free requests
- 400,000 GB-seconds of compute time

Formula: Cost = (Requests × \$0.20/million) +
(GB-seconds × \$0.0000166667)

6. Lambda Function Components

Function Structure

```
# Python Lambda Function Example

import json

def lambda_handler(event, context):
    """
    Main handler function - Entry point for Lambda

    Parameters:
    - event: Input data (JSON format)
    - context: Runtime information

    Returns:
    - Response object (typically JSON)
    """

    # Your business logic here
    name = event.get('name', 'World')
    message = f"Hello from Lambda, {name}!"

    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': message
        })
    }
```

Node.js Lambda Example

```
// Node.js Lambda Function
exports.handler = async (event, context) => {

    const name = event.name || 'World';
    const message = `Hello from Lambda, ${name}!`;

    return {
        statusCode: 200,
        body: JSON.stringify({
            message: message
        })
    };
};
```

Lambda Function Components

LAMBDA FUNCTION ANATOMY

HANDLER

- Entry point for Lambda execution
- Format: filename.function_name
- Example: lambda_function.lambda_handler

EVENT

- Input data passed to function
- JSON format
- Contains trigger-specific data

CONTEXT

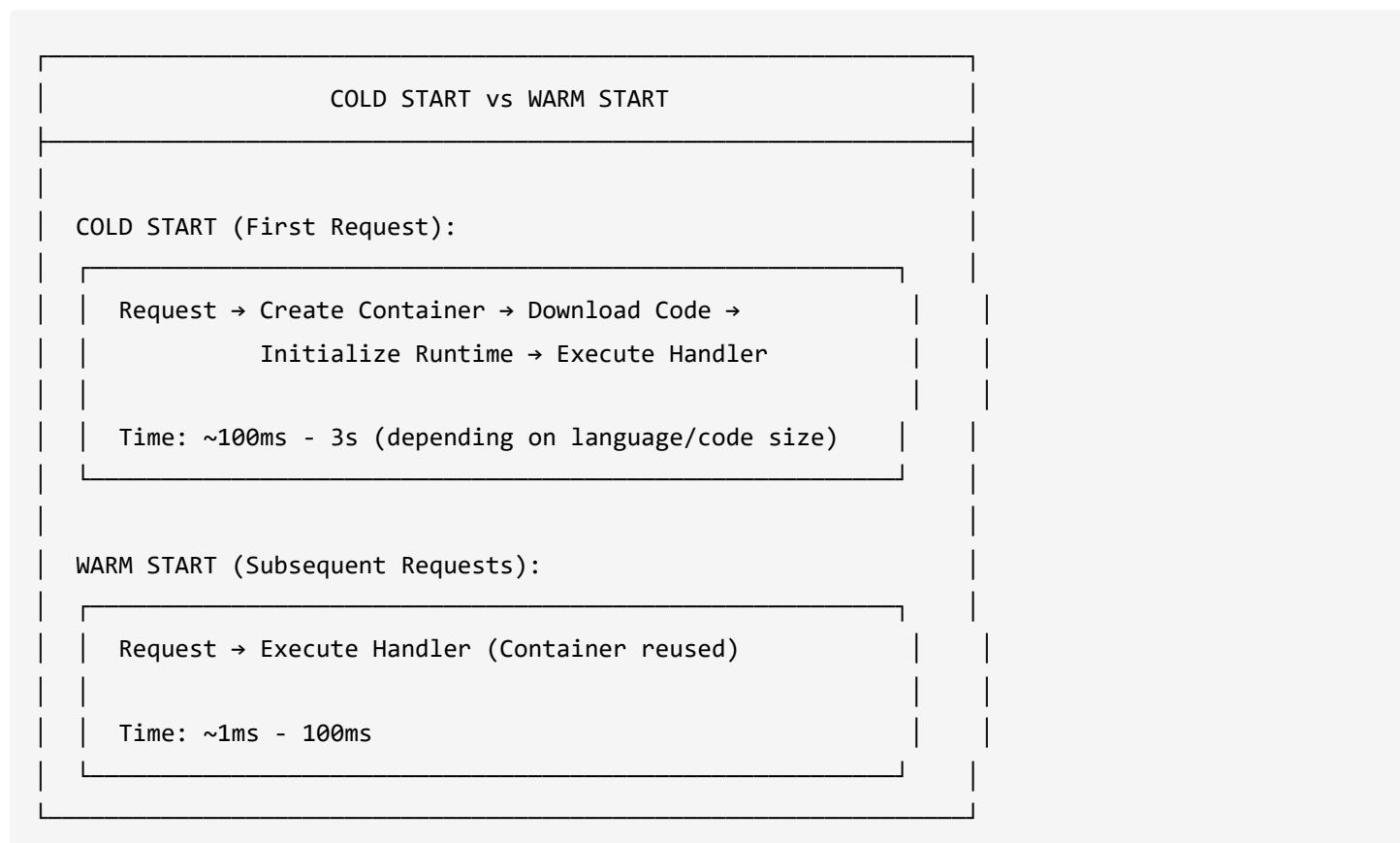
- Runtime information
- Request ID
- Function name
- Remaining execution time

Lambda Configuration Options

Setting	Description	Limits
Memory	Allocated memory (128 MB - 10,240 MB)	Affects CPU allocation
Timeout	Maximum execution time	1 sec - 15 min
Environment Variables	Configuration values	4 KB total
Concurrency	Parallel executions	Default: 1000/region
Layers	Shared code/libraries	Up to 5 layers

7. Lambda Execution Model

Cold Start vs Warm Start



Lambda Execution Lifecycle

LAMBDA EXECUTION LIFECYCLE

1. INIT PHASE

- └─ Load function code
- └─ Initialize runtime
- └─ Run initialization code (outside handler)

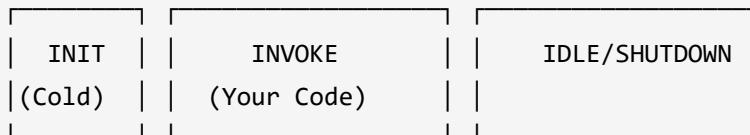
2. INVOKE PHASE

- └─ Receive event
- └─ Execute handler function
- └─ Return response

3. SHUTDOWN PHASE (if idle)

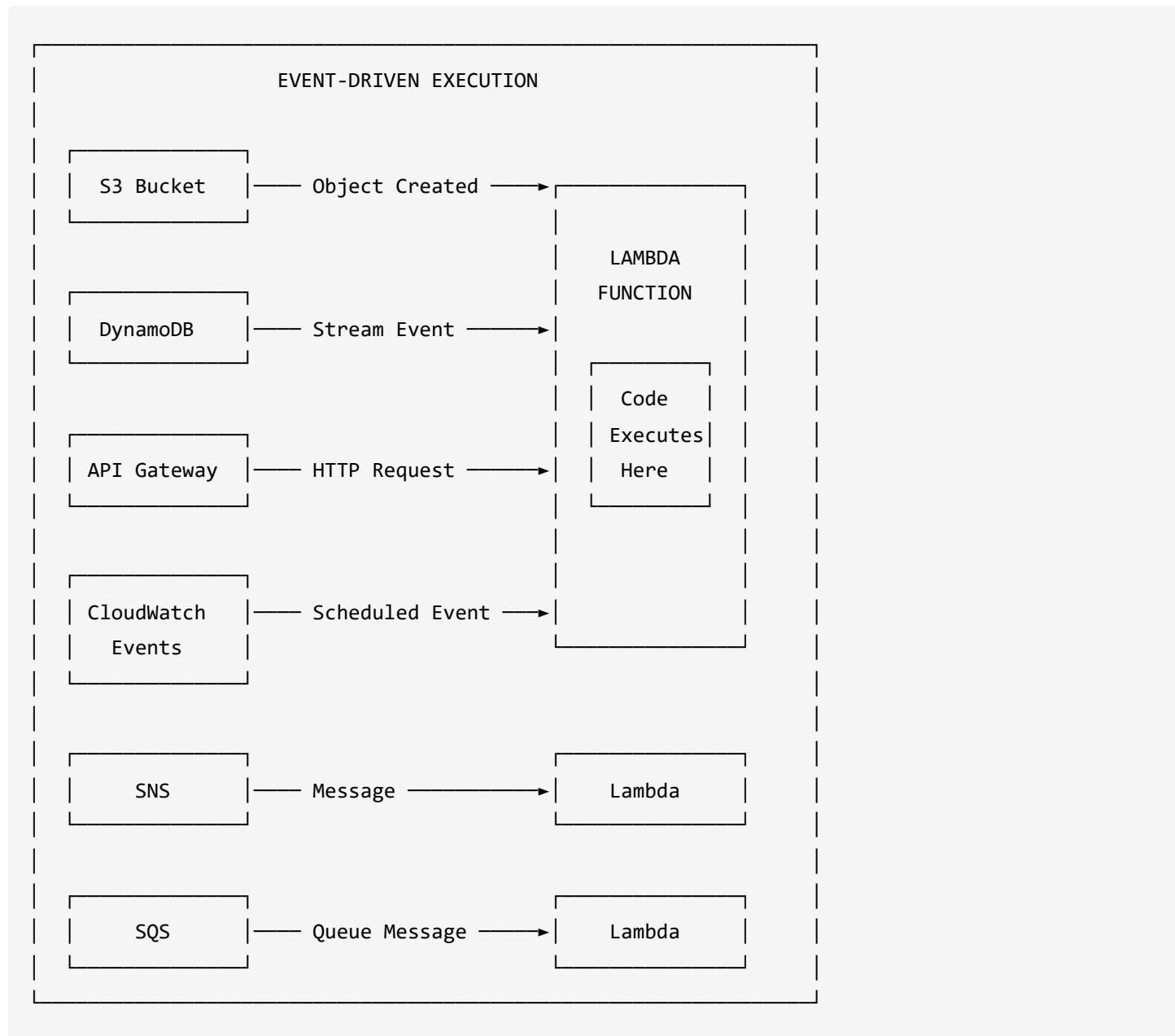
- └─ Container terminated

Timeline:



8. Lambda Triggers and Events

Event-Driven Architecture



Common Trigger Types

Trigger Type	Description	Use Case
S3	Object created/deleted/modified	Image processing, file validation
API Gateway	HTTP requests	REST APIs, webhooks
DynamoDB Streams	Table changes	Real-time data processing
CloudWatch Events	Scheduled/rule-based	Cron jobs, automation
SNS	Pub/sub messaging	Notifications, fan-out

Trigger Type	Description	Use Case
SQS	Queue messages	Async processing, decoupling
Kinesis	Real-time streaming	Log analysis, IoT
Cognito	User pool triggers	Authentication workflows

S3 Trigger Example (Event JSON)

```
{
  "Records": [
    {
      "eventSource": "aws:s3",
      "eventName": "ObjectCreated:Put",
      "s3": {
        "bucket": {
          "name": "my-bucket"
        },
        "object": {
          "key": "uploads/image.jpg",
          "size": 1024
        }
      }
    }
  ]
}
```

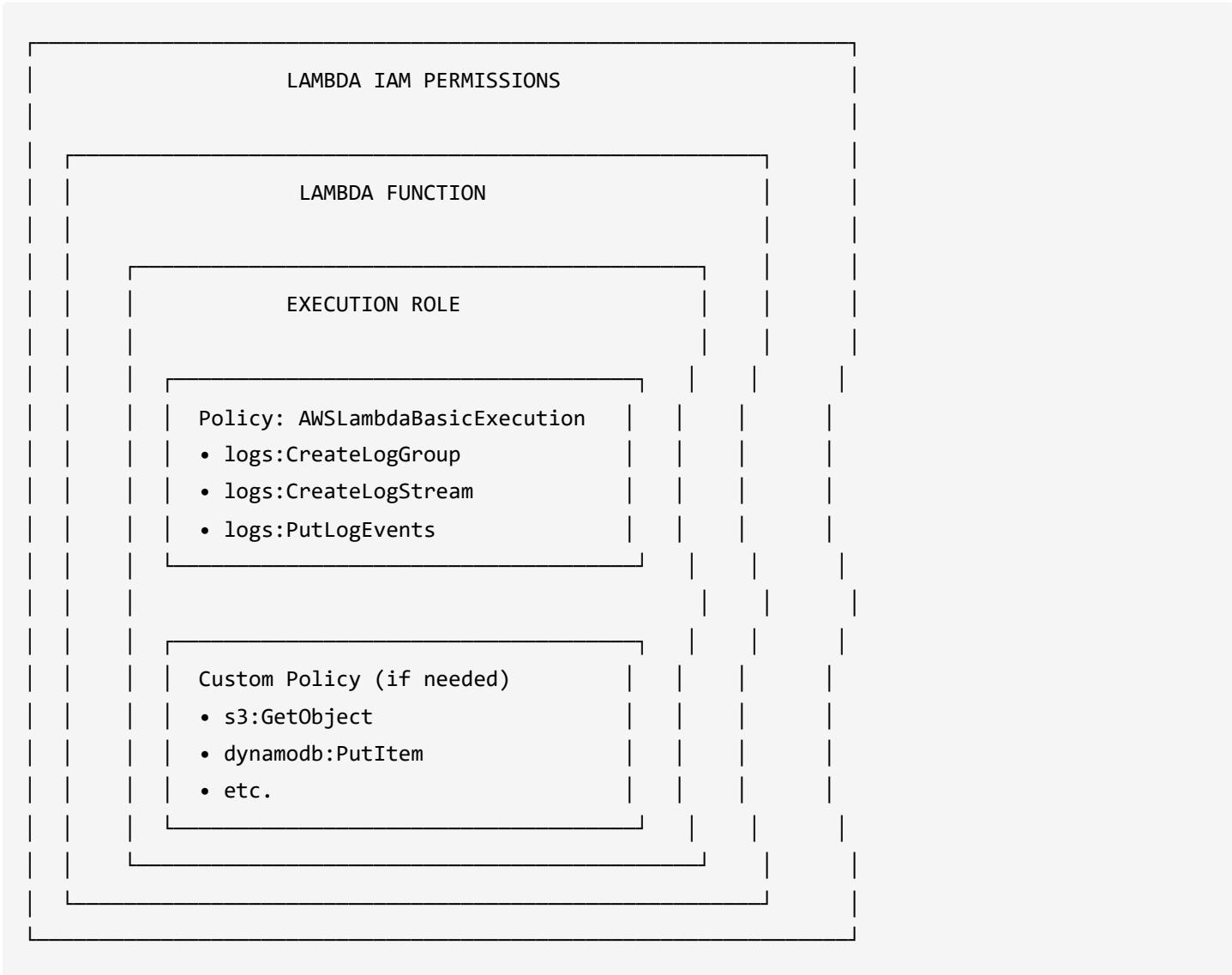
9. Lambda IAM Roles and Permissions

Execution Role

Lambda needs an **IAM Execution Role** to:

- Write logs to CloudWatch
- Access other AWS services (S3, DynamoDB, etc.)

IAM Role Architecture



Creating IAM Role for Lambda

Step 1: Create Role

1. Go to IAM → Roles → Create Role
2. Select "AWS Service" → "Lambda"
3. Attach required policies
4. Name the role

Step 2: Attach Basic Policy

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
                "logs>PutLogEvents"  
            ],  
            "Resource": "arn:aws:logs:*:*:  
        }  
    ]  
}
```

Step 3: Add Custom Permissions (Example for S3)

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

10. Lambda Practical Implementation

Creating a Lambda Function (Console Steps)

1. Navigate to Lambda

- Search "Lambda" in AWS Console
- Click "Create function"

2. Choose Creation Method

- Author from scratch (new code)
- Use a blueprint (pre-built templates)
- Container image

- Browse serverless app repository

3. Configure Function

- Function name: demo-function
- Runtime: Python 3.11
- Architecture: x86_64
- Execution role: Create new or use existing

4. Write Code

```
import json

def lambda_handler(event, context):
    print("Hello from Lambda!")
    print(f"Event: {json.dumps(event)}")

    return {
        'statusCode': 200,
        'body': json.dumps('Hello from M Prashant!')
    }
```

5. Deploy and Test

- Click "Deploy"
- Click "Test" → Create test event
- View execution results

Adding S3 Trigger (Console Steps)

1. Go to Lambda function → Configuration → Triggers
2. Click "Add trigger"
3. Select "S3"
4. Choose bucket
5. Select event type (All object create events)
6. Acknowledge recursive invocation warning
7. Click "Add"

Lambda Function with S3 Processing

```
import json
import boto3

def lambda_handler(event, context):
    # Get S3 event details
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    print(f"File uploaded: {key} to bucket: {bucket}")

    # Process the file (example: get file info)
    s3 = boto3.client('s3')
    response = s3.head_object(Bucket=bucket, Key=key)

    file_size = response['ContentLength']
    content_type = response['ContentType']

    print(f"File size: {file_size} bytes")
    print(f"Content type: {content_type}")

    return {
        'statusCode': 200,
        'body': json.dumps({
            'bucket': bucket,
            'key': key,
            'size': file_size
        })
    }
```

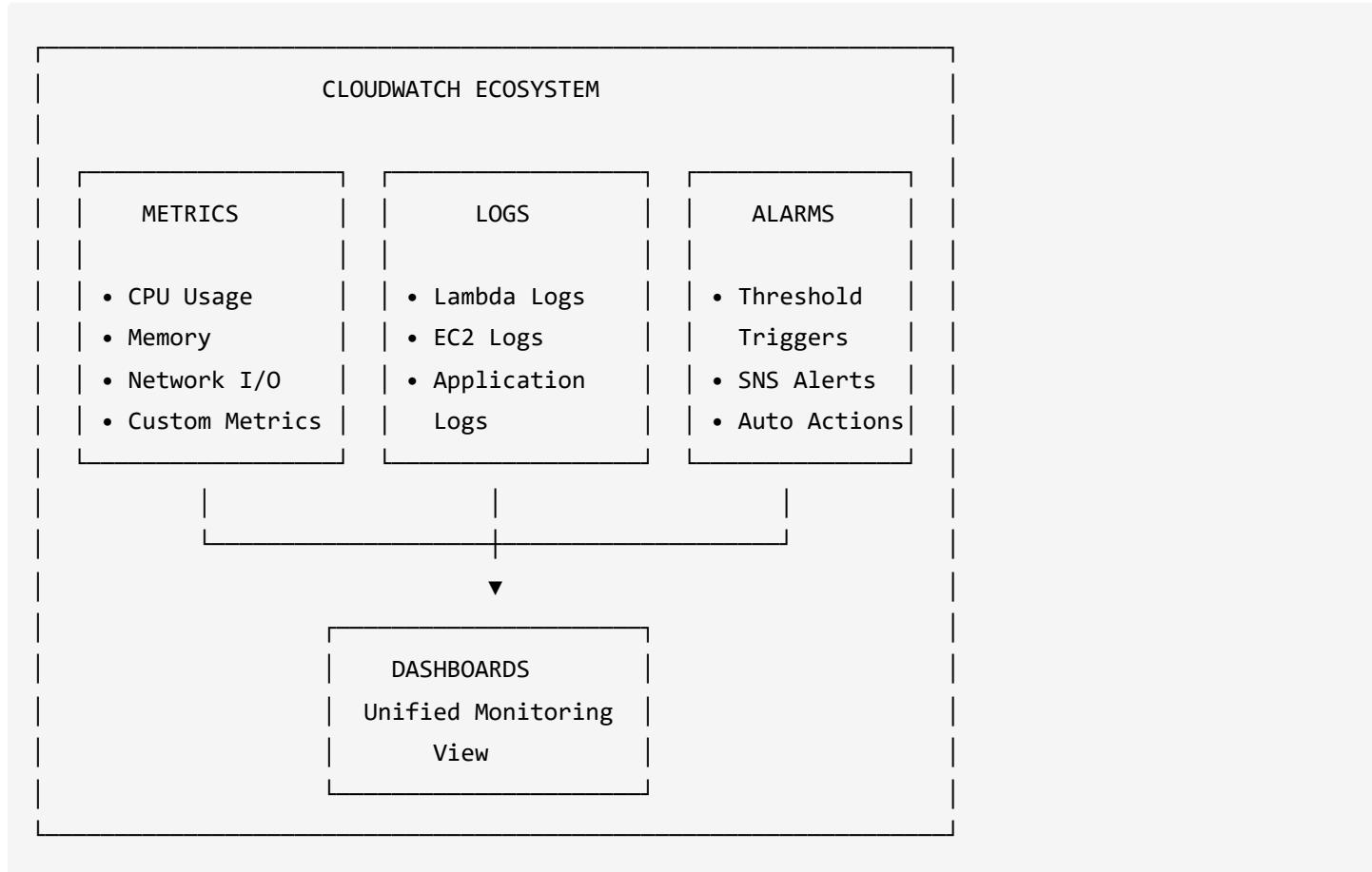
11. Amazon CloudWatch Overview

What is CloudWatch?

CloudWatch is AWS's **monitoring and observability service** that provides:

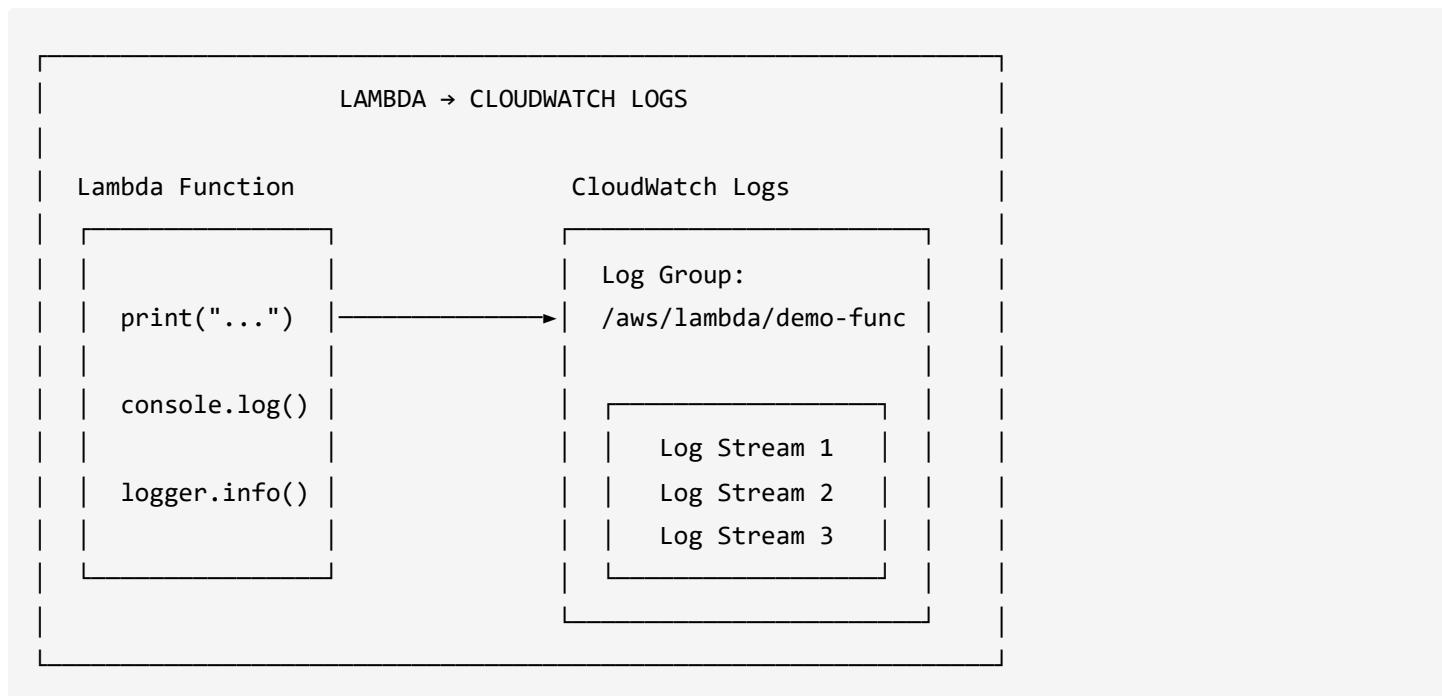
- Metrics collection and visualization
- Log aggregation and analysis
- Alarms and notifications
- Automated actions

CloudWatch Components



12. CloudWatch Logs

How Lambda Logs Work



Log Structure

Component	Description
Log Group	Container for log streams (usually per function)
Log Stream	Sequence of log events (per execution context)
Log Event	Single log entry with timestamp and message

Viewing Lambda Logs

Method 1: CloudWatch Console

1. Go to CloudWatch → Log groups
2. Find `/aws/lambda/your-function-name`
3. Click on log stream
4. View log events

Method 2: Lambda Console

1. Go to Lambda function
2. Click "Monitor" tab
3. Click "View logs in CloudWatch"

Log Insights Query Example

```
-- Find errors in Lambda logs
fields @timestamp, @message
| filter @message like /ERROR/
| sort @timestamp desc
| limit 20

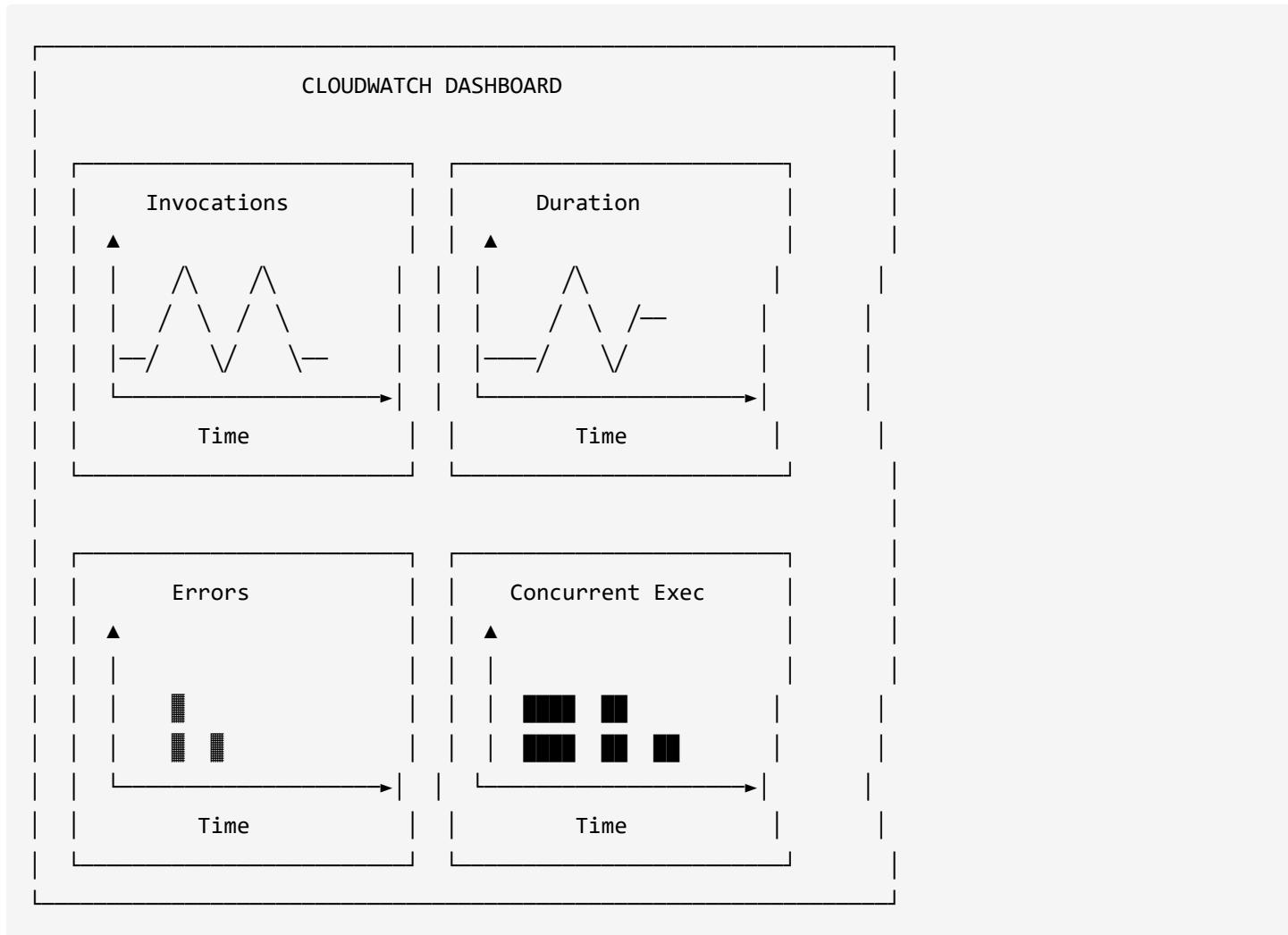
-- Calculate function duration
fields @timestamp, @duration
| filter @type = "REPORT"
| stats avg(@duration), max(@duration), min(@duration)
```

13. CloudWatch Monitoring

Lambda Metrics in CloudWatch

Metric	Description
Invocations	Number of function calls
Duration	Execution time
Errors	Failed executions
Throttles	Rejected due to concurrency limits
ConcurrentExecutions	Parallel executions
UnreservedConcurrentExecutions	Concurrency using unreserved pool

Monitoring Dashboard



Creating CloudWatch Alarm (Console)

1. Go to CloudWatch → Alarms → Create alarm
2. Select metric (e.g., Lambda Errors)
3. Set threshold (e.g., Errors > 5)
4. Configure actions (SNS notification)
5. Name and create alarm

CloudWatch Alarm Example (CLI)

```
aws cloudwatch put-metric-alarm \
--alarm-name "Lambda-High-Error-Rate" \
--alarm-description "Alarm when error rate exceeds threshold" \
--metric-name Errors \
--namespace AWS/Lambda \
--statistic Sum \
--period 300 \
--threshold 5 \
--comparison-operator GreaterThanThreshold \
--dimensions Name=FunctionName,Value=demo-function \
--evaluation-periods 1 \
--alarm-actions arn:aws:sns:region:account:my-topic
```

14. Real Interview Questions & Answers

Q1: What is DynamoDB and how is it different from RDS?

Answer:

DynamoDB is a fully managed NoSQL database service.

Aspect	DynamoDB (NoSQL)	RDS (Relational)
Data Model	Key-value, document	Tables with relationships
Schema	Flexible (schemaless)	Fixed schema
Scaling	Automatic horizontal	Manual vertical
Queries	Primary key lookups, limited queries	Full SQL support
Transactions	Single-item atomic, limited multi-item	Full ACID transactions
Performance	Single-digit millisecond	Varies with complexity

Aspect	DynamoDB (NoSQL)	RDS (Relational)
Cost Model	Pay per request/capacity	Instance-based
Best For	High-throughput, simple queries	Complex relationships

Key Point: Choose DynamoDB for speed and scale, RDS for complex queries.

Q2: Explain Partition Key and Sort Key in DynamoDB.

Answer:

Partition Key (Hash Key):

- Uniquely identifies an item
- DynamoDB uses it to determine physical storage partition
- Must be specified for every item
- Should have high cardinality (many unique values)

Sort Key (Range Key):

- Optional second part of primary key
- Items with same partition key are stored together, sorted by sort key
- Enables range queries within a partition

Example:

```
Table: Orders
Partition Key: customer_id
Sort Key: order_date
```

```
Query: All orders for customer_id=123 in 2024
      → Efficient scan within single partition
```

Q3: What is the difference between Query and Scan in DynamoDB?

Answer:

Operation	Query	Scan
Requires	Partition key	Nothing (reads all)

Operation	Query	Scan
Performance	Fast (single partition)	Slow (entire table)
Cost	Lower (reads fewer items)	Higher (reads everything)
Use Case	Known key lookup	Analytics, migrations
Filtering	On sort key (efficient)	After reading (inefficient)

Best Practice: Design schema to use Query, avoid Scan in production.

Q4: What is DAX and when would you use it?

Answer:

DAX (DynamoDB Accelerator) is a fully managed in-memory cache for DynamoDB.

Benefits:

- 10x performance (microseconds instead of milliseconds)
- API-compatible (no code changes)
- Fully managed (no cache invalidation logic)

Use Cases:

- Read-heavy workloads (90%+ reads)
- Same data accessed frequently (hot partitions)
- Latency-sensitive applications
- Gaming leaderboards, real-time bidding

When NOT to use:

- Write-heavy workloads
 - Strongly consistent reads required
 - Data changes frequently (cache ineffective)
-

Q5: What is AWS Lambda? Explain cold start.

Answer:

Lambda is a serverless compute service that runs code in response to events.

Cold Start:

When Lambda creates a new execution environment:

1. **Download** function code
2. **Initialize** runtime (Python, Node.js, etc.)
3. **Execute** initialization code (outside handler)
4. **Run** handler function

Cold start occurs when:

- First invocation after deployment
- Scaling out to handle more requests
- After ~15 minutes of inactivity

Mitigation Strategies:

- **Provisioned Concurrency**: Pre-warm instances (\$\$\$)
 - **Smaller packages**: Faster download
 - **Initialization outside handler**: Reused on warm starts
 - **Choose faster runtimes**: Python, Node.js faster than Java
-

Q6: What triggers can invoke a Lambda function?

Answer:

Trigger	Event Type	Use Case
API Gateway	HTTP request	REST APIs, webhooks
S3	Object create/delete	Image processing, ETL
DynamoDB Streams	Table changes	Real-time replication
SQS	Queue message	Async processing
SNS	Notification	Fan-out, alerting
CloudWatch Events	Schedule/rule	Cron jobs, automation
Kinesis	Stream data	Real-time analytics
Cognito	Auth events	User registration workflows
ALB	HTTP request	Load-balanced serverless

Q7: What is the Lambda execution role?

Answer:

The execution role is an **IAM role** that Lambda assumes to access AWS services.

Must have:

- Trust policy allowing Lambda to assume the role
- Permissions for CloudWatch Logs (basic)
- Additional permissions based on what Lambda accesses

Minimum Policy (AWSLambdaBasicExecutionRole):

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:*:*:*"
  }]
}
```

Add more permissions for S3, DynamoDB, etc. as needed.

Q8: Explain Lambda concurrency types.

Answer:

Type	Description	Use Case
Unreserved	Shared pool (default 1000/region)	Most functions
Reserved	Dedicated portion of account limit	Guarantee capacity
Provisioned	Pre-initialized instances	Eliminate cold starts

Throttling:

- Occurs when concurrency limit reached
- Returns 429 (Too Many Requests)
- SQS/Kinesis retry automatically

Provisioned Concurrency Cost:

- ~\$0.000004463 per provisioned instance per second
 - Use for latency-sensitive, predictable workloads
-

Q9: How does Lambda pricing work?

Answer:

Two pricing dimensions:

1. **Requests:** \$0.20 per million requests
2. **Duration:** \$0.0000166667 per GB-second

Free Tier (monthly, forever):

- 1 million requests
- 400,000 GB-seconds

Example Calculation:

Function: 512 MB memory, 200ms duration, 5M invocations/month

Requests: $5M \times \$0.20/M = \1.00

Duration: $5M \times 0.5GB \times 0.2s \times \$0.0000166667 = \$8.33$

Total: \$9.33/month

Q10: What is CloudWatch and its main components?

Answer:

Component	Purpose	Example
Metrics	Numerical time-series data	CPU%, request count
Logs	Text-based application logs	Lambda print statements
Alarms	Threshold-based notifications	Alert if errors > 10
Dashboards	Visualization	Custom monitoring views
Events/EventBridge	Event routing	Trigger Lambda on schedule
Log Insights	Query language for logs	Find errors across functions

Q11: What is DynamoDB Global Tables?

Answer:

Global Tables provide **multi-region, active-active replication**.

Features:

- Data replicated across selected regions
- Read/write from any region
- Sub-second replication
- Last-writer-wins conflict resolution

Requirements:

- DynamoDB Streams enabled
- Same table name in all regions
- Same key schema

Use Cases:

- Disaster recovery
 - Low-latency global access
 - Regional compliance
-

Q12: How do you debug Lambda functions?

Answer:

1. CloudWatch Logs:

- All `print()` statements logged
- Automatic logging of invocations, errors
- Use Log Insights for queries

2. X-Ray Tracing:

- Enable active tracing
- Visualize service calls
- Identify bottlenecks

3. Local Testing:

- AWS SAM CLI (`sam local invoke`)

- Docker for Lambda environment
- Unit tests with mocked AWS services

4. Error Handling:

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        logger.info(f"Event: {event}")
        # Your code
    except Exception as e:
        logger.error(f"Error: {str(e)}")
        raise
```

Q13: What are DynamoDB capacity modes?

Answer:

Mode	Description	Best For
On-Demand	Pay per request, auto-scales	Unpredictable traffic
Provisioned	Pre-defined RCU/WCU	Predictable traffic
Provisioned + Auto Scaling	Auto-adjust within range	Variable but predictable

Capacity Units:

- **RCU (Read Capacity Unit):** 1 strongly consistent read (4 KB) per second
- **WCU (Write Capacity Unit):** 1 write (1 KB) per second

On-Demand Pricing: ~5x more expensive than provisioned at steady load.

Q14: What is the difference between CloudWatch Logs and Metrics?

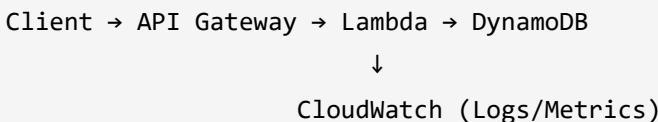
Answer:

Feature	CloudWatch Logs	CloudWatch Metrics
Data Type	Text (log entries)	Numerical (data points)
Source	Applications, Lambda, EC2	AWS services, custom
Structure	Log Groups → Streams → Events	Namespace → Dimensions → Values
Querying	Log Insights (SQL-like)	GetMetricData API
Use Case	Debugging, audit trail	Monitoring, alerting
Retention	Configurable (1 day - forever)	15 months (high-res shorter)

Q15: How would you design a serverless API with Lambda?

Answer:

Architecture:



Implementation Steps:

1. **Create DynamoDB table** for data storage
2. **Create Lambda function** with business logic
3. **Attach IAM role** with DynamoDB permissions
4. **Create API Gateway** REST or HTTP API
5. **Connect API Gateway** to Lambda
6. **Enable CloudWatch** for monitoring

Best Practices:

- Use environment variables for configuration
- Implement error handling and logging
- Use Lambda Layers for shared code
- Enable X-Ray for tracing
- Set appropriate timeouts and memory

Cost Optimization:

- Right-size memory (more memory = more CPU = faster = cheaper sometimes)
- Use provisioned capacity for DynamoDB if predictable

- Implement caching (DAX, API Gateway caching)
-

Quick Reference Summary

DynamoDB Key Commands

```
# Create table
aws dynamodb create-table --table-name Users --attribute-definitions ...

# Put item
aws dynamodb put-item --table-name Users --item '{"id":{"S":"1"}}'

# Get item
aws dynamodb get-item --table-name Users --key '{"id":{"S":"1"}}'

# Query
aws dynamodb query --table-name Users --key-condition-expression "id = :v" ...
```

Lambda Key Commands

```
# Create function
aws lambda create-function --function-name demo \
--runtime python3.11 --handler lambda_function.lambda_handler \
--role arn:aws:iam::xxx:role/lambda-role --zip-file fileb://code.zip

# Invoke function
aws lambda invoke --function-name demo output.txt

# Update code
aws lambda update-function-code --function-name demo --zip-file fileb://code.zip

# Add permission (for S3 trigger)
aws lambda add-permission --function-name demo \
--statement-id s3-trigger --action lambda:InvokeFunction \
--principal s3.amazonaws.com --source-arn arn:aws:s3:::my-bucket
```

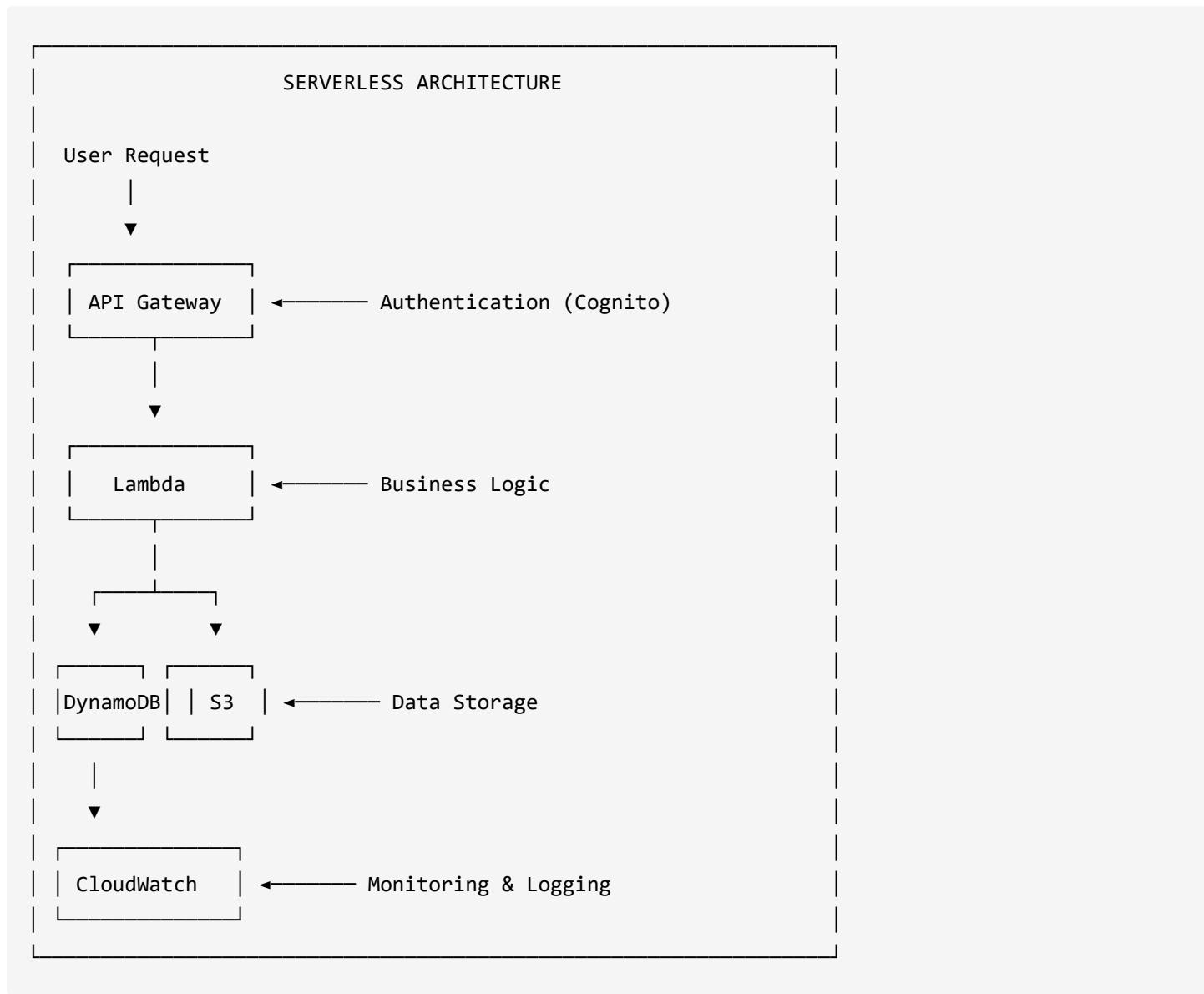
CloudWatch Key Commands

```
# View logs
aws logs describe-log-groups
aws logs get-log-events --log-group-name /aws/lambda/demo --log-stream-name xxx

# Create alarm
aws cloudwatch put-metric-alarm --alarm-name HighErrors ...

# Get metrics
aws cloudwatch get-metric-statistics --namespace AWS/Lambda \
--metric-name Invocations --dimensions Name=FunctionName,Value=demo ...
```

Architecture Pattern: Serverless Application



AWS Zero to Hero - Notes 07

Amazon Route 53, DNS Management & CloudFront CDN

Table of Contents

1. [DNS Fundamentals](#)
 2. [Amazon Route 53 Overview](#)
 3. [Domain Registration with Route 53](#)
 4. [Hosted Zones](#)
 5. [DNS Record Types](#)
 6. [Routing Policies](#)
 7. [Route 53 Health Checks](#)
 8. [Route 53 Use Cases](#)
 9. [Amazon CloudFront Overview](#)
 10. [CloudFront Architecture](#)
 11. [CloudFront with S3 Integration](#)
 12. [Route 53 Billing](#)
 13. [Important Q&A](#)
-

1. DNS Fundamentals

What is DNS?

DNS (Domain Name System) converts human-readable domain names into machine-readable IP addresses.

In-Depth Explanation: The "Phone Directory" Analogy

Think of DNS like a **giant phone directory for the internet**:

Real World	DNS
Person's name	Domain name (google.com)
Phone number	IP address (142.250.190.78)
Phone book	DNS servers
Looking up a number	DNS query
Multiple listings for a business	Multiple A records

Why DNS matters:

- IP addresses are hard to remember (142.250.190.78)
- Domain names are easy (google.com)
- IP addresses can change; domain names stay constant
- Load balancing and failover become possible

How DNS Resolution Works Step-by-Step

User types: www.example.com

|
v

STEP 1: Browser Cache

Check if IP is already cached locally
(Usually cached for a few minutes to hours based on TTL)

| Not found
v

STEP 2: Operating System Cache

Check hosts file and system DNS cache

| Not found
v

STEP 3: ISP's DNS Resolver (Recursive Resolver)

ISP's server does the heavy lifting
May have cached response from previous queries

| Not found - begins recursive resolution
v

STEP 4: Root DNS Servers (13 global root servers)

"I don't know www.example.com, but ask .com TLD servers"

|
v

STEP 5: TLD DNS Servers (.com, .org, .net, .io)

"I don't know www.example.com, but example.com's
authoritative server is ns1.route53.amazonaws.com"

|
v

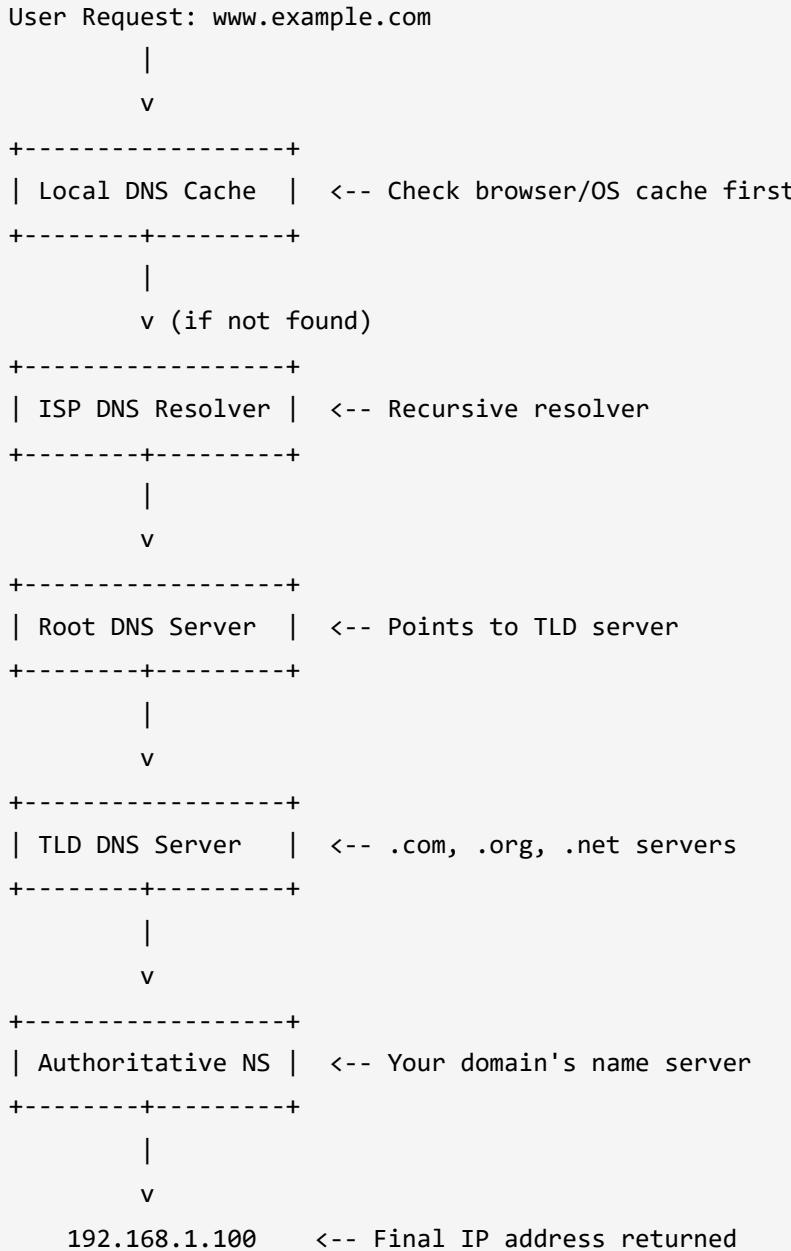
STEP 6: Authoritative DNS Server (Route 53)

"www.example.com is 192.168.1.100" (finally!)

|
v

Response cached at each level
Browser connects to 192.168.1.100

DNS Resolution Flow



2. Amazon Route 53 Overview

What is Route 53?

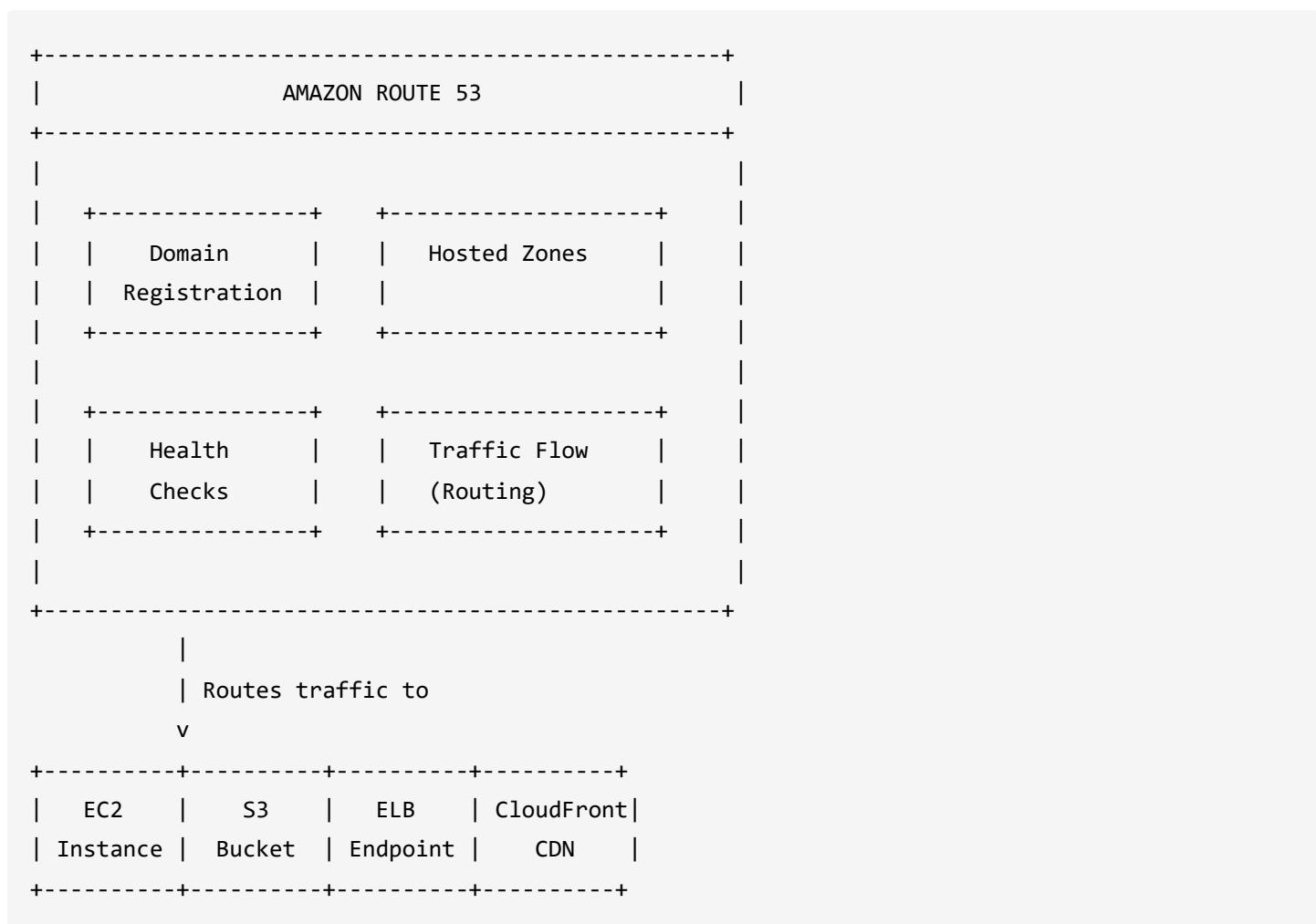
Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service provided by AWS.

Why "Route 53"? DNS operates on port 53, hence the name Route 53.

Key Features

Feature	Description
Domain Registration	Register new domain names directly
DNS Management	Manage DNS records for your domains
Health Checks	Monitor endpoint health and trigger failover
Traffic Routing	Route traffic using various policies
High Availability	100% SLA availability
Global Reach	Anycast network across the globe

Route 53 Architecture



3. Domain Registration with Route 53

Steps to Register a Domain

1. Open Route 53 Console
2. Navigate to "Domain Registration" → "Register Domain"
3. Search for your desired domain name
4. Check availability and pricing
5. Provide contact information (Registrant, Admin, Tech)
6. Complete purchase

Domain Registration Details

```
Domain: myexamplewebsite.com
```

```
|
```

```
v
```

REGISTRATION INFO	
Registrar: Amazon Route 53	
Status: Active	
Expiry: 1 Year (Auto-renew)	
DNSSEC: Optional	
Privacy Protection: Enabled	

Important Points

- Domain registration can take **up to 24-48 hours** to propagate
- AWS charges annual fees for domain registration
- Auto-renewal is recommended to prevent domain expiry
- Transfer domains from other registrars (GoDaddy, Hostinger, etc.)

4. Hosted Zones

What is a Hosted Zone?

A **Hosted Zone** is a container that holds information about how you want to route traffic for a domain and its subdomains.

Types of Hosted Zones

Type	Description	Use Case
Public Hosted Zone	Routes internet traffic	Publicly accessible websites
Private Hosted Zone	Routes traffic within VPC	Internal applications

Hosted Zone Structure

```
+-----+
|      HOSTED ZONE: example.com      |
+-----+
|
|
+-----+ |
|  Record Sets (DNS Records)   | |
+-----+ |
|  - NS Record (Name Servers)  | |
|  - SOA Record                | |
|  - A Record (IPv4)           | |
|  - AAAA Record (IPv6)        | |
|  - CNAME Record              | |
|  - MX Record (Mail)          | |
|  - Alias Record              | |
+-----+ |
|
+-----+
```

Creating a Hosted Zone

```
Route 53 Console
|
v
Hosted Zones → Create Hosted Zone
|
+
--- Domain Name: example.com
--- Type: Public / Private
--- VPC (if Private)
|
v
Hosted Zone Created with:
- NS Records (Name Servers)
- SOA Record (Start of Authority)
```

5. DNS Record Types

Common Record Types

RECORD TYPE	DESCRIPTION
A	Maps domain name to IPv4 address example.com → 192.168.1.100
AAAA	Maps domain name to IPv6 address example.com → 2001:0db8:85a3:0000:....
CNAME	Maps domain name to another domain name www.example.com → example.com (Cannot use for root/apex domain)
NS	Name Server records - defines authoritative DNS servers for the domain
MX	Mail Exchange - specifies mail servers example.com → mail.example.com (priority 10)
TXT	Text records - verification, SPF, DKIM Used for email verification, etc.
SRV	Service records - defines service location
SOA	Start of Authority - zone metadata

Alias Records (AWS Specific)

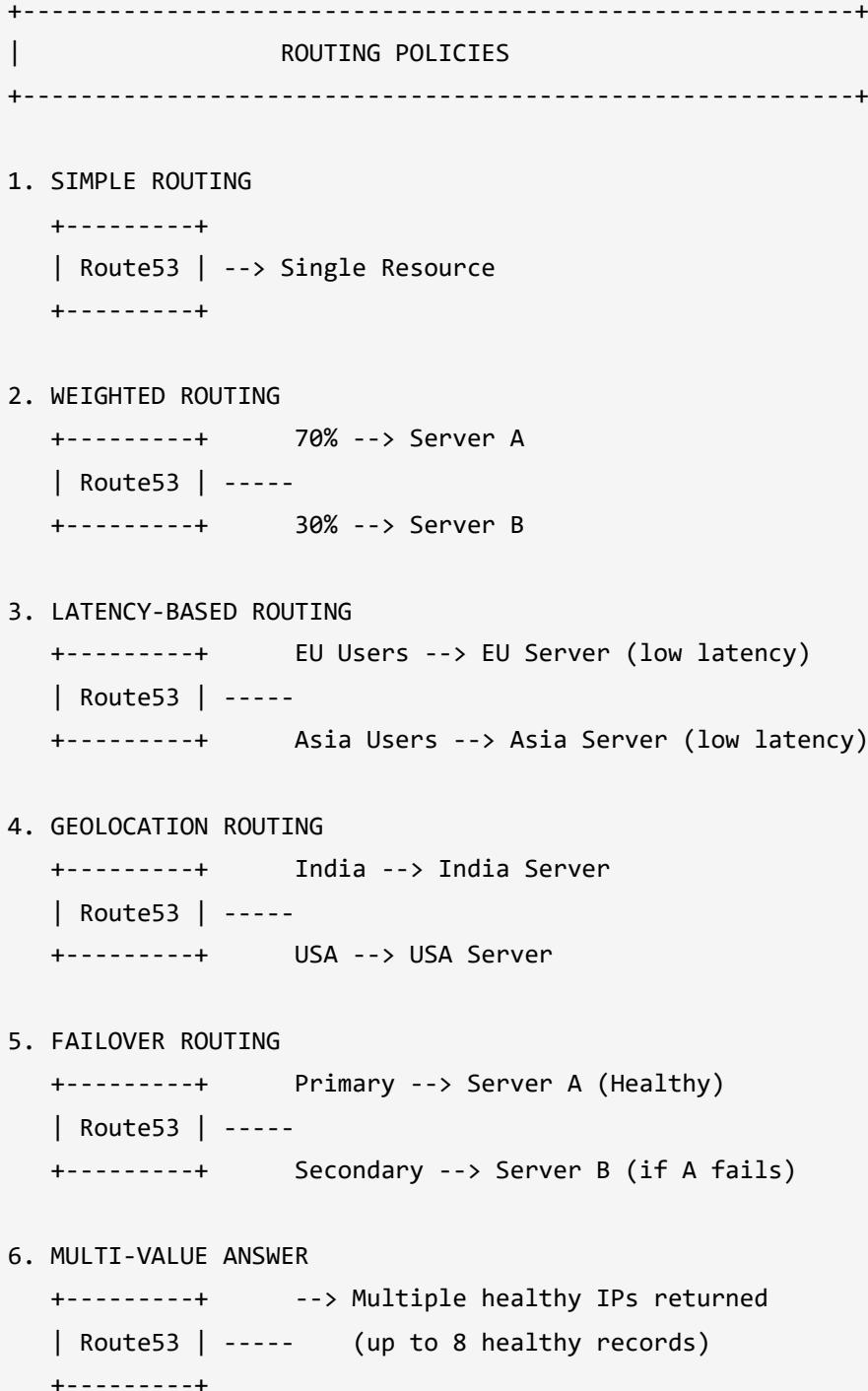
ALIAS RECORD	
Special Route 53 record type that maps to:	
✓ CloudFront Distributions	
✓ Elastic Load Balancers	
✓ S3 Website Endpoints	
✓ API Gateway	
✓ Another Route 53 Record	
Benefits:	
- Free of charge for AWS resources	
- Works with root domain (apex)	
- Native health check integration	

A Record vs CNAME vs Alias

Feature	A Record	CNAME	Alias
Points to	IP Address	Domain Name	AWS Resource
Root Domain	✓ Yes	X No	✓ Yes
Free for AWS	X Charged	X Charged	✓ Free
Health Check	Manual	Manual	Native

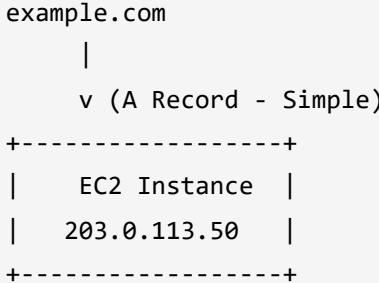
6. Routing Policies

Types of Routing Policies



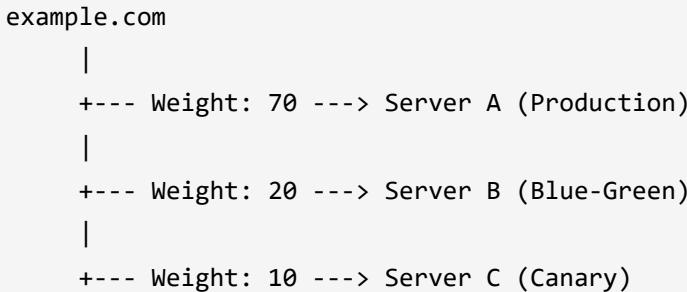
1. Simple Routing Policy

Use Case: Single resource, basic DNS routing



2. Weighted Routing Policy

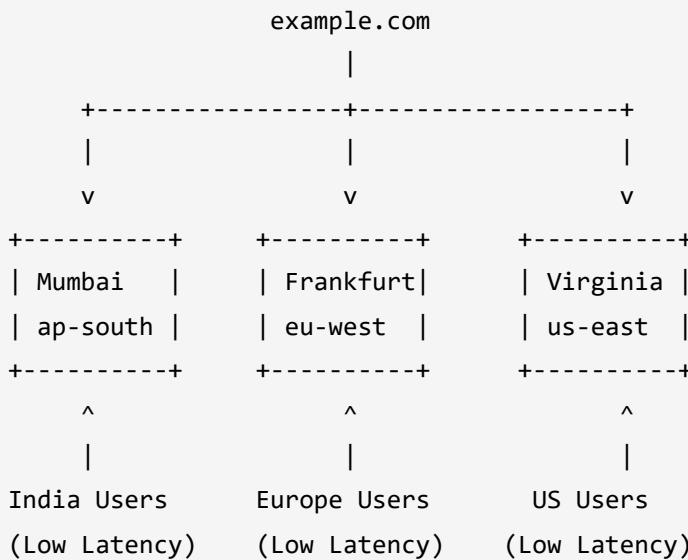
Use Case: A/B testing, gradual deployments, load distribution



Formula: Traffic % = (Record Weight / Sum of All Weights) × 100

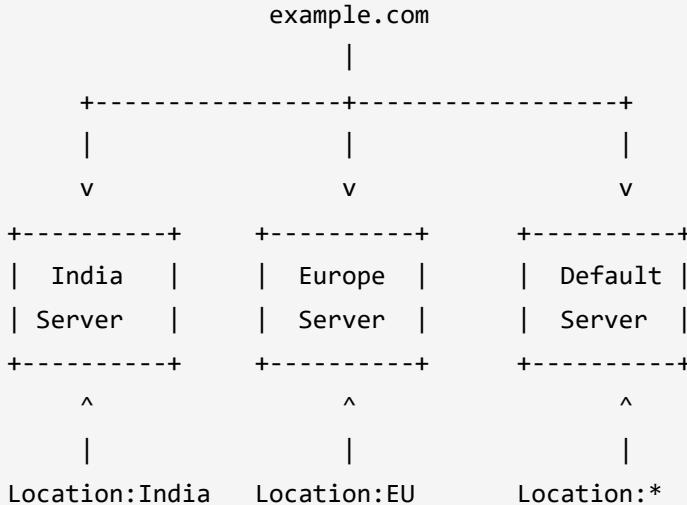
3. Latency-Based Routing

Use Case: Global applications requiring lowest latency



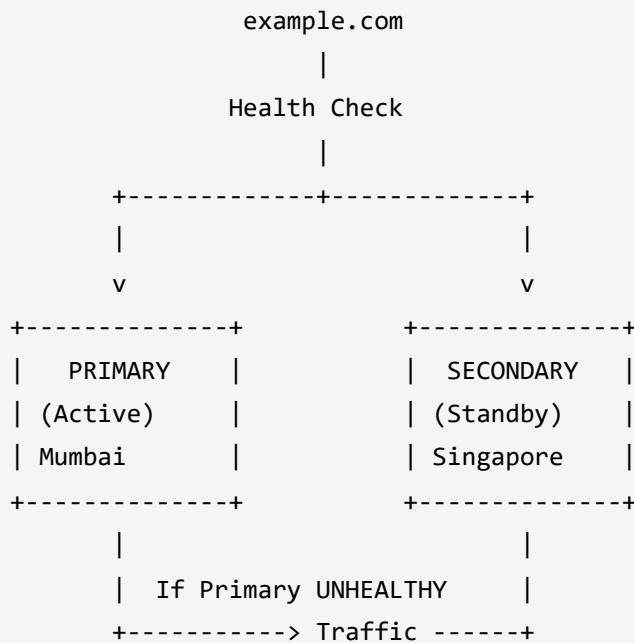
4. Geolocation Routing

Use Case: Content localization, compliance, regional restrictions



5. Failover Routing Policy

Use Case: Disaster recovery, high availability

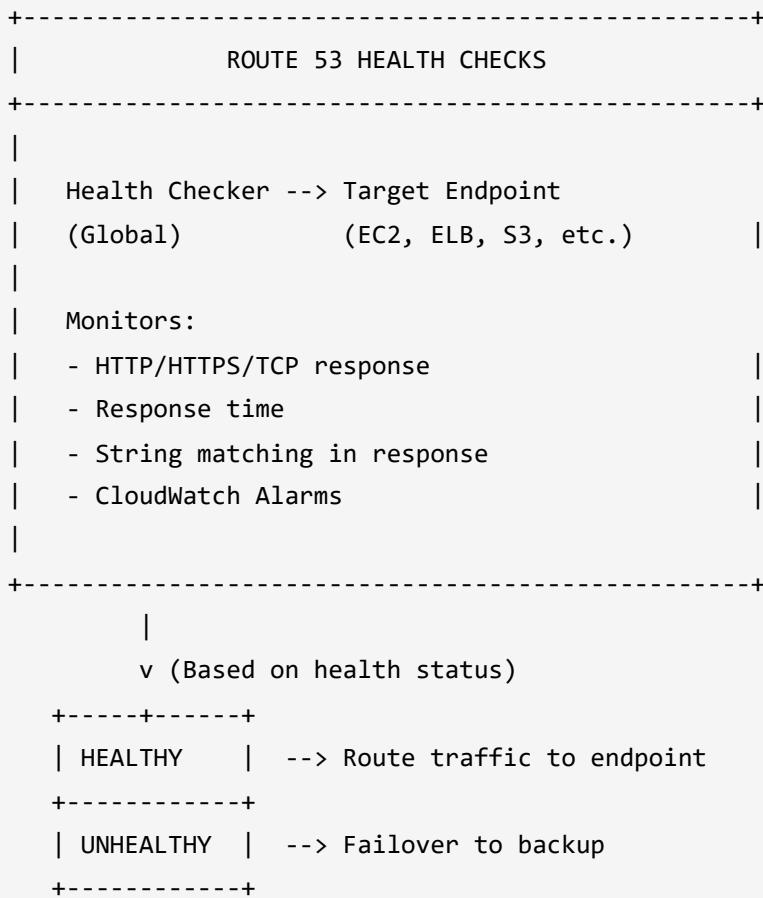


7. Route 53 Health Checks

What are Health Checks?

Health checks monitor the health and performance of your endpoints and trigger failover when issues are detected.

Health Check Architecture



Health Check Configuration

Health Check Settings:

Endpoint Type: HTTP / HTTPS / TCP	
IP/Domain: 192.168.1.100	
Port: 80 / 443	
Path: /health (for HTTP)	
Request Interval: 10 sec / 30 sec	
Failure Threshold: 3 (consecutive)	
String Matching: "OK" (optional)	
Regions: Multiple AWS regions	

Health Check Types

Type	Description
Endpoint Health Check	Monitors specific URL/IP
Calculated Health Check	Combines multiple health checks

Type	Description
CloudWatch Alarm Health Check	Based on CloudWatch metrics

Creating Health Check

```

Route 53 Dashboard
|
v
Health Checks → Create Health Check
|
+-- Name: my-website-health
+-- Type: Endpoint
+-- Protocol: HTTP/HTTPS
+-- IP/Domain: ec2-xxx.compute.amazonaws.com
+-- Port: 80
+-- Path: /health
+-- Advanced:
    +-- Request Interval: 30 seconds
    +-- Failure Threshold: 3
    +-- Regions: Select multiple

```

8. Route 53 Use Cases

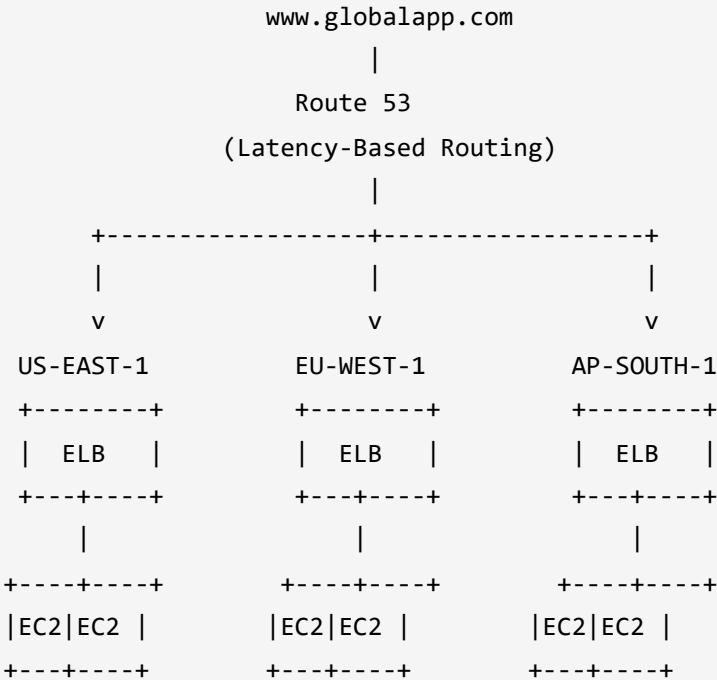
Use Case 1: Website Hosting with Custom Domain

```

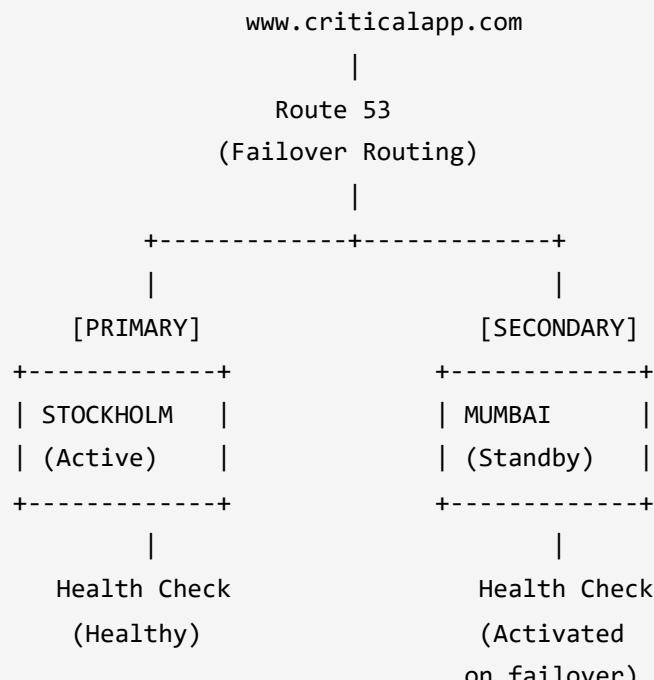
User Browser
|
| www.mywebsite.com
v
+-----+
|     Route 53      |
| (Hosted Zone)   |
+-----+
|
| A Record / Alias
v
+-----+
|     EC2 / ELB      |
| (Web Server)    |
+-----+

```

Use Case 2: Global Load Distribution with Latency Routing



Use Case 3: Disaster Recovery with Failover



9. Amazon CloudFront Overview

What is CloudFront?

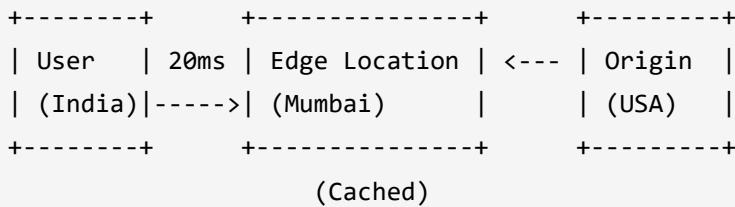
Amazon CloudFront is a fast Content Delivery Network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency and high transfer speeds.

Why Use CloudFront?

WITHOUT CDN:



WITH CLOUDFRONT:

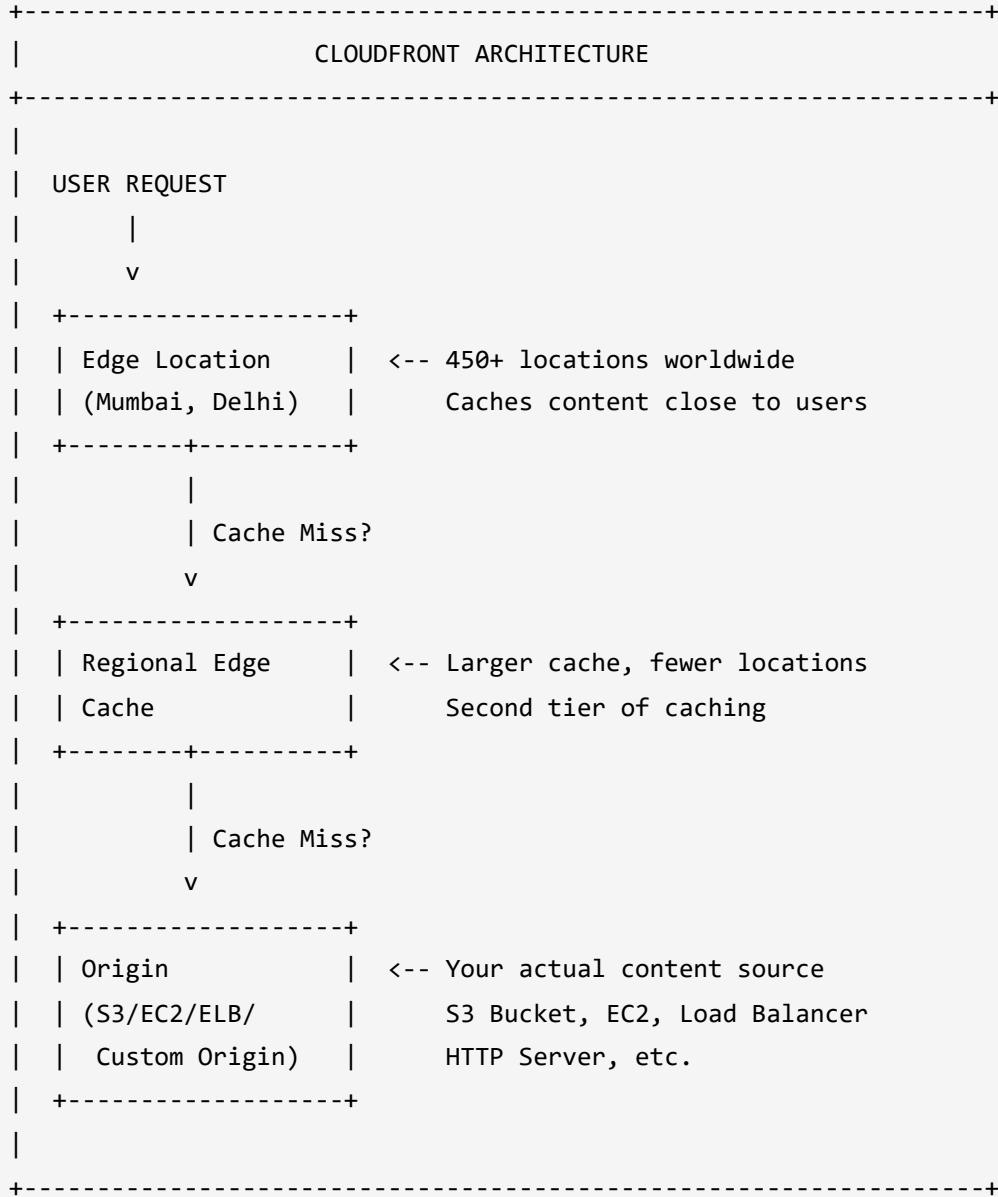


CloudFront Benefits

Benefit	Description
Low Latency	Content served from nearest edge location
High Performance	450+ edge locations globally
Security	DDoS protection, HTTPS, AWS Shield
Cost Effective	Pay only for data transfer
Integration	Works with S3, EC2, ELB, Lambda@Edge
Global Reach	Serves users worldwide efficiently

10. CloudFront Architecture

CloudFront Components



CloudFront Distribution

CLOUDFRONT DISTRIBUTION

```
+-----+
| |
| Distribution ID: E1234567890ABC      |
| Domain: d1234567890.cloudfront.net    |
| |
| Origins:                                |
+-----+
| | Origin 1: S3 Bucket (static content)   |
| | Origin 2: EC2/ELB (dynamic content)     |
| +-----+
| |
| Behaviors:                               |
+-----+
| | /*.jpg, /*.png --> Origin 1 (S3)       |
| | /api/* --> Origin 2 (EC2/ELB)           |
| | Default --> Origin 1                   |
| +-----+
| |
| Settings:                                |
+-----+
| | Price Class: All Edge Locations        |
| | SSL Certificate: Custom or Default     |
| | HTTP/2: Enabled                         |
| | Alternate Domain: www.example.com       |
| +-----+
| |
+-----+
```

Edge Location Distribution

GLOBAL EDGE LOCATIONS (450+)

NORTH AMERICA	EUROPE	ASIA
- Virginia	- London	- Tokyo
- California	- Frankfurt	- Mumbai
- Oregon	- Paris	- Singapore
- Ohio	- Milan	- Sydney
SOUTH AMERICA	MIDDLE EAST	AFRICA
- São Paulo	- Dubai	- Cape Town
- Buenos Aires	- Bahrain	- Nairobi

11. CloudFront with S3 Integration

S3 Static Website Hosting with CloudFront

USERS (Global)



Setup Steps

Step 1: Create S3 Bucket with Static Website Hosting

- Enable Static Website Hosting
- Upload website files
- Note the S3 Website Endpoint

Step 2: Create CloudFront Distribution

- Origin: S3 Website Endpoint
- Viewer Protocol: Redirect HTTP to HTTPS
- Cache Policy: Optimized for static content

Step 3: Configure Route 53 (Optional)

- Create Alias Record
- Point to CloudFront Distribution
- www.example.com → d123456.cloudfront.net

Benefits of CloudFront + S3

CLOUDFRONT + S3 BENEFITS	
✓ Global Content Delivery	- Content cached at 450+ edge locations
✓ Reduced Latency	- Users get content from nearest edge
✓ Lower S3 Costs	- Fewer requests to origin S3 bucket
✓ HTTPS/SSL Support	- Free SSL with ACM
✓ DDoS Protection	- AWS Shield Standard included
✓ Custom Domain	- Use your own domain name

12. Route 53 Billing

Pricing Components

ROUTE 53 PRICING																	
CHARGED COMPONENTS:																	
<table border="1"> <tr> <td>Hosted Zones</td> <td>\$0.50/month/zone</td> <td></td> </tr> <tr> <td>DNS Queries</td> <td>\$0.40/million</td> <td></td> </tr> <tr> <td>Health Checks</td> <td>\$0.50-\$2/month</td> <td></td> </tr> <tr> <td>Domain Registration</td> <td>\$9-\$35/year</td> <td></td> </tr> <tr> <td>Traffic Flow</td> <td>\$50/policy/month</td> <td></td> </tr> </table>			Hosted Zones	\$0.50/month/zone		DNS Queries	\$0.40/million		Health Checks	\$0.50-\$2/month		Domain Registration	\$9-\$35/year		Traffic Flow	\$50/policy/month	
Hosted Zones	\$0.50/month/zone																
DNS Queries	\$0.40/million																
Health Checks	\$0.50-\$2/month																
Domain Registration	\$9-\$35/year																
Traffic Flow	\$50/policy/month																
FREE COMPONENTS:																	
<table border="1"> <tr> <td>Alias queries to AWS resources (CloudFront, ELB, S3, etc.)</td> <td></td> <td></td> </tr> </table>			Alias queries to AWS resources (CloudFront, ELB, S3, etc.)														
Alias queries to AWS resources (CloudFront, ELB, S3, etc.)																	

Cost Optimization Tips

Tip	Description
Use Alias Records	Free queries to AWS resources
Consolidate Hosted Zones	Reduce number of zones
Optimize TTL	Higher TTL = fewer queries
Review Health Checks	Remove unused health checks

13. Real Interview Questions & Answers

Q1: What is DNS and how does it work?

Answer:

DNS (Domain Name System) is the internet's "phone book" that translates human-readable domain names to IP addresses.

Resolution Process:

1. User types domain → Browser checks cache
2. If not cached → Query ISP's recursive resolver
3. Resolver queries Root DNS servers → TLD servers → Authoritative servers
4. IP address returned and cached at each level

Key Concepts:

- **TTL (Time to Live):** How long to cache the response
 - **Recursive Resolver:** Does lookup on behalf of client
 - **Authoritative Server:** Has definitive answer for a domain
-

Q2: What is Amazon Route 53 and why is it called "53"?

Answer:

Route 53 is AWS's highly available, scalable DNS web service.

Why "53": DNS protocol uses **port 53** for both TCP and UDP.

Key Features:

- Domain registration
 - DNS management (hosted zones)
 - Health checks & failover
 - Traffic routing policies
 - 100% SLA availability
-

Q3: Explain the different Route 53 routing policies.

Answer:

Policy	How It Works	Use Case
Simple	Returns one record	Single resource
Weighted	% traffic to each endpoint	Blue-green, A/B testing
Latency	Routes to lowest latency region	Global apps
Geolocation	Routes by user's location	Compliance, localization
Geoproximity	Routes by proximity + bias	Shift traffic between regions
Failover	Primary/secondary setup	Disaster recovery
Multi-value	Returns multiple healthy IPs	Simple load balancing

Interview Tip: Know when to use each:

- **Performance** → Latency-based
 - **Compliance** → Geolocation
 - **DR** → Failover
 - **A/B testing** → Weighted
-

Q4: What is the difference between A Record, CNAME, and Alias?

Answer:

Feature	A Record	CNAME	Alias (AWS)
Points to	IP address	Another domain	AWS resource
Root domain	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	<input checked="" type="checkbox"/> Yes
Query cost	Charged	Charged	Free (for AWS)
TTL	You set it	You set it	AWS manages
Example	example.com → 1.2.3.4	www → example.com	example.com → ELB

Key Point: Use Alias for AWS resources—it's free, supports root domain, and health-check integrated.

Q5: What is a Hosted Zone?

Answer:

A Hosted Zone is a **container for DNS records** for a specific domain.

Types:

- **Public Hosted Zone:** Internet-facing resources
- **Private Hosted Zone:** VPC-internal resources

What's created automatically:

- NS records (4 name servers)
- SOA record (Start of Authority)

Cost: \$0.50/month per hosted zone + query charges.

Q6: How do Route 53 Health Checks work?

Answer:

Health checks monitor endpoint availability and trigger failover.

Types:

1. **Endpoint Health Check:** HTTP/HTTPS/TCP to your resource
2. **Calculated Health Check:** Combines child health checks (OR/AND logic)
3. **CloudWatch Alarm Health Check:** Based on CloudWatch metrics

How it works:

- Health checkers in 15+ global locations
- Configurable interval (10s or 30s)
- Failure threshold (e.g., 3 consecutive failures)
- Can check HTTP status code, response body

Integration: Connects to Failover routing for automatic DR.

Q7: What is Amazon CloudFront?

Answer:

CloudFront is AWS's **Content Delivery Network (CDN)** service.

How it works:

1. User requests content
2. Request routed to nearest edge location (450+ globally)
3. If cached → serve immediately
4. If not cached → fetch from origin, cache, serve

Benefits:

- Reduced latency (content served from nearby)
 - DDoS protection (AWS Shield)
 - HTTPS/SSL support
 - Lower origin load (caching)
-

Q8: What is the difference between Edge Location and Regional Edge Cache?

Answer:

Component	Description	Cache Size	Count
Edge Location	Primary cache, closest to users	Smaller	450+
Regional Edge Cache	Secondary tier, between edge and origin	Larger	~13

Request Flow:

User → Edge Location (miss) → Regional Edge (miss) → Origin

Benefit: Regional Edge keeps less-popular content cached, reducing origin requests.

Q9: How do you secure content in CloudFront?

Answer:

1. HTTPS:

- Use ACM (free SSL certificates)
- Redirect HTTP to HTTPS
- Custom SSL for own domain

2. Origin Access Control (OAC):

- Restrict S3 access to CloudFront only

- Prevents direct S3 URL access

3. Signed URLs/Cookies:

- Time-limited access
- IP-restricted access
- For premium/private content

4. Geo-Restriction:

- Allow/block by country
- Compliance requirements

5. WAF Integration:

- Block malicious requests
- Rate limiting
- SQL injection protection

Q10: What is the difference between Latency-based and Geolocation routing?

Answer:

Aspect	Latency-Based	Geolocation
Routing Decision	Network latency measurement	User's physical location
Goal	Best performance	Location-based content
Example	Route to fastest server	India users → India server
Use Case	Global app performance	Compliance, localization
Fallback	Picks any healthy endpoint	Needs explicit default

Key Difference: Latency is about speed; Geolocation is about where the user IS.

Q11: How do you integrate Route 53 with CloudFront?

Answer:

Steps:

1. Create CloudFront distribution (origin: S3/ELB/EC2)
2. Get CloudFront domain (d123456.cloudfront.net)
3. In Route 53, create **Alias record** pointing to CloudFront
4. Use your custom domain (www.example.com)

Benefits:

- No DNS query charges (Alias to AWS resource)
 - Automatic health integration
 - Works with root domain
-

Q12: What is TTL in DNS and how does it affect performance?

Answer:

TTL (Time to Live) = how long DNS resolvers cache a record.

TTL Setting	Pros	Cons
Low (60s)	Fast DNS changes, quick failover	More DNS queries, higher cost
High (86400s/1 day)	Fewer queries, lower cost	Slow propagation of changes

Best Practices:

- Normal operations: 300-600 seconds
 - Before migration: Lower to 60s
 - After migration: Raise back up
-

Q13: Can Route 53 be used for private DNS?

Answer:

Yes! Private Hosted Zones provide DNS for VPC resources.

Use Cases:

- Internal application names (api.internal.company.com)
- Override public DNS within VPC
- Hybrid cloud DNS (with Route 53 Resolver)

Requirements:

- Associate with one or more VPCs
- VPC must have DNS support enabled

- Can span multiple VPCs (even different accounts)
-

Q14: What happens if all Route 53 name servers go down?

Answer:

This is extremely unlikely due to Route 53's architecture:

- 4 name servers assigned per hosted zone
- Servers distributed across multiple TLDs (.com, .net, .org, .co.uk)
- Hosted on AWS's global anycast network
- **100% SLA** (AWS guarantees availability)

Even if one NS fails: Others continue serving. Route 53 is designed for no single point of failure.

Q15: How do you troubleshoot DNS issues with Route 53?

Answer:

1. Check DNS propagation:

```
# Using dig
dig +short example.com
dig @8.8.8.8 example.com # Query specific resolver

# Using nslookup
nslookup example.com
```

2. Verify hosted zone records:

- Check record exists
- Correct record type (A vs CNAME vs Alias)
- Correct value

3. Check health checks:

- Health check status in console
- Health checker endpoints accessible?

4. TTL considerations:

- Old DNS may be cached
- Wait for TTL to expire after changes

5. Name server configuration:

- Domain registrar pointing to correct Route 53 NS
 - NS records in hosted zone match
-

Quick Reference Summary

Route 53 Services

Service	Purpose
Domain Registration	Register/transfer domain names
Hosted Zones	Container for DNS records
Health Checks	Monitor endpoint health
Traffic Flow	Visual routing policy editor
Resolver	Hybrid DNS for VPC

Record Type Quick Reference

A	→ Domain → IPv4
AAAA	→ Domain → IPv6
CNAME	→ Domain → Domain (no apex)
Alias	→ Domain → AWS Resource (free, supports apex)
MX	→ Domain → Mail Server
NS	→ Domain → Name Servers
TXT	→ Domain → Text (verification)

CLI Commands

```
# List hosted zones
aws route53 list-hosted-zones

# List records in hosted zone
aws route53 list-resource-record-sets --hosted-zone-id Z123456789

# Create health check
aws route53 create-health-check --caller-reference "unique-string" \
--health-check-config file://health-check-config.json

# List CloudFront distributions
aws cloudfront list-distributions

# Create CloudFront invalidation
aws cloudfront create-invalidation \
--distribution-id E123456789 \
--paths "/*"
```

End of Notes 07 - Route 53, DNS & CloudFront

AWS VPC (Virtual Private Cloud) - Complete Networking Guide

Table of Contents

1. [VPC Fundamentals](#)
2. [CIDR Block & IP Addressing](#)
3. [Subnets - Public & Private](#)
4. [Internet Gateway \(IGW\)](#)
5. [Route Tables](#)
6. [NAT Gateway & NAT Instance](#)
7. [Security Groups](#)
8. [Network ACLs \(NACLs\)](#)
9. [VPC Peering](#)

10. [VPC Endpoints](#)
 11. [Bastion Host / Jump Server](#)
 12. [Elastic IP Address](#)
 13. [VPC Flow Logs](#)
 14. [AWS Direct Connect](#)
 15. [AWS VPN \(Client VPN & Site-to-Site\)](#)
 16. [Important Q&A Section](#)
-

1. VPC Fundamentals

What is VPC?

Virtual Private Cloud (VPC) is a logically isolated virtual network in AWS that you define. It's your own private cloud within AWS where you can launch AWS resources.

In-Depth Explanation: The "Office Building" Analogy

Think of VPC like building your own **private office building**:

Office Building	VPC
Your building	VPC (isolated from others)
Floors	Subnets
Lobby (public entrance)	Public subnet
Server room (restricted)	Private subnet
Main entrance door	Internet Gateway
Security guards	Security Groups + NACLs
Service elevator (staff only)	NAT Gateway
Connecting bridge to another building	VPC Peering
Your address	CIDR block

Why VPC Matters (The Foundation of AWS)

Without understanding VPC:

- Cannot secure resources properly
- Cannot design scalable architectures

- Cannot troubleshoot network issues
- Cannot pass AWS certifications

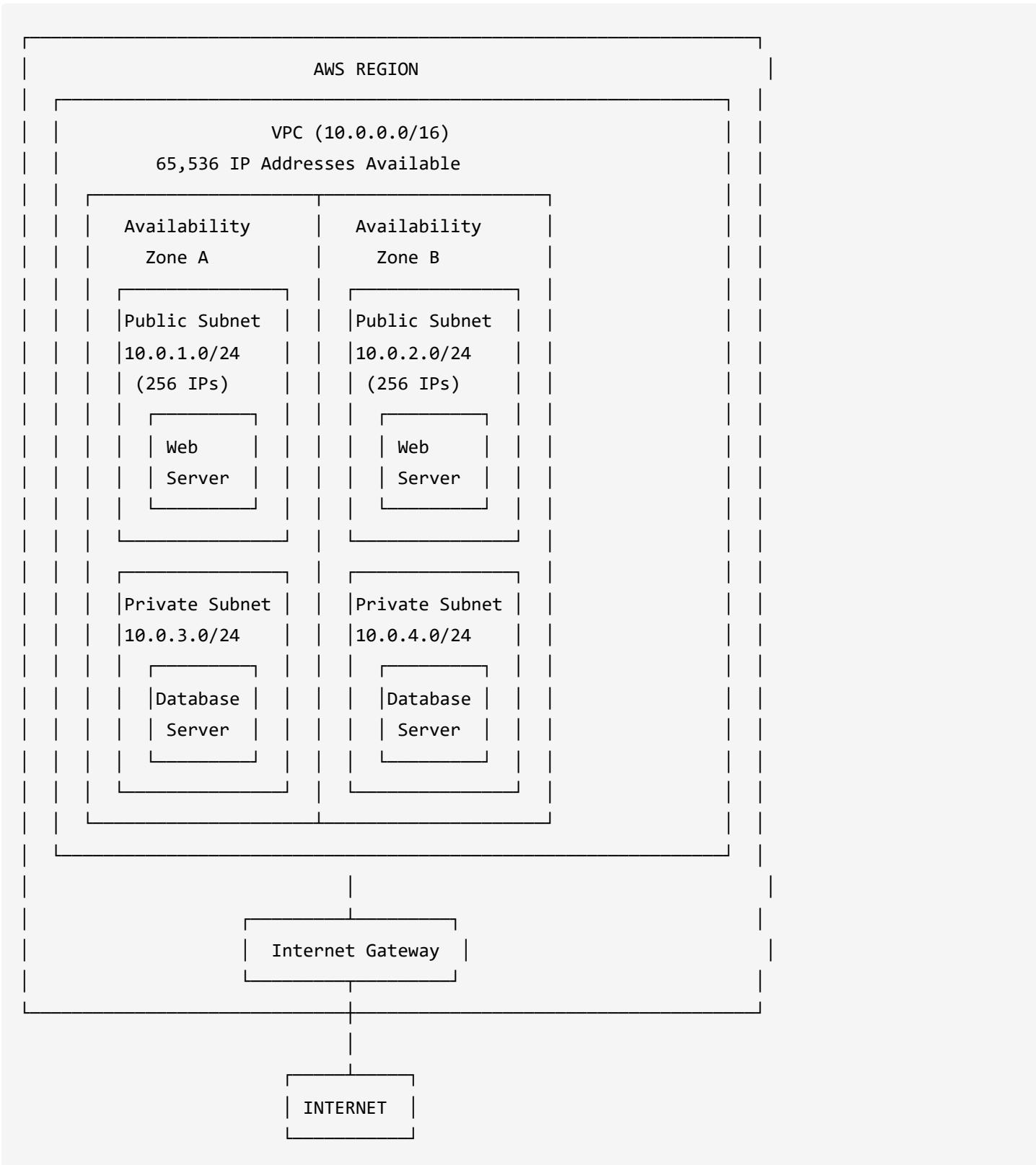
VPC is involved in:

- EC2 instances (must be in a VPC)
- RDS databases (live in VPC subnets)
- Lambda (can be VPC-connected)
- Load Balancers (require VPC)
- Essentially EVERYTHING in AWS

Key Characteristics:

- **Isolation:** Complete network isolation from other AWS customers
- **Regional Service:** VPC spans all Availability Zones in a region
- **Customizable:** Define your own IP address range, subnets, route tables
- **Default VPC:** AWS provides a default VPC in each region (auto-created)
- **IP Range:** Supports up to 65,536 IP addresses (using /16 CIDR)

VPC Architecture Diagram:



VPC Components Summary:

Component	Purpose
VPC	Isolated virtual network
Subnet	Sub-division of VPC

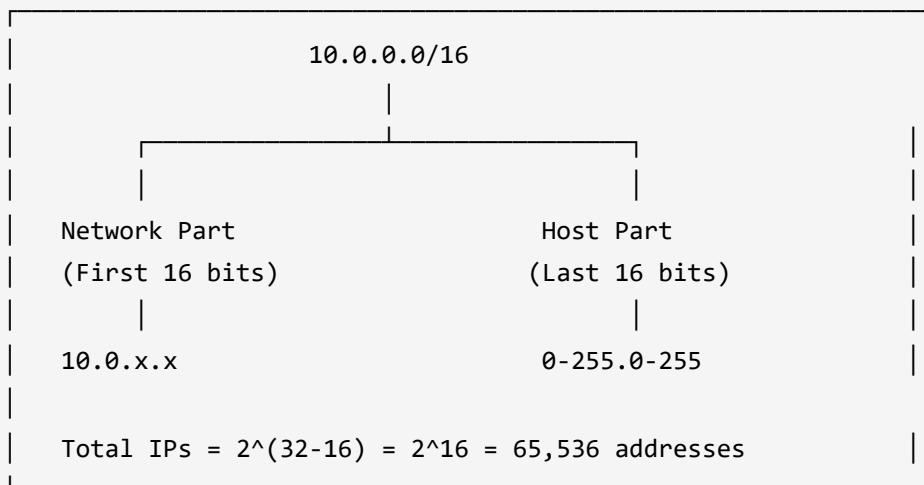
Component	Purpose
Internet Gateway	Internet connectivity
Route Table	Traffic routing rules
Security Group	Instance-level firewall
NACL	Subnet-level firewall
NAT Gateway	Outbound internet for private subnets
VPC Peering	Connect two VPCs
VPC Endpoint	Private AWS service access

2. CIDR Block & IP Addressing

What is CIDR?

CIDR (Classless Inter-Domain Routing) is a method for allocating IP addresses and routing.

CIDR Notation:



AWS VPC CIDR Rules:

- **Minimum:** /28 (16 IP addresses)
- **Maximum:** /16 (65,536 IP addresses)
- **IPv4 & IPv6:** VPC supports both

Common CIDR Examples:

CIDR Block	Subnet Mask	Total IPs	Usable IPs
/16	255.255.0.0	65,536	65,531
/20	255.255.240.0	4,096	4,091
/24	255.255.255.0	256	251
/28	255.255.255.240	16	11

Reserved IP Addresses (AWS reserves 5 IPs per subnet):

Example: Subnet 10.0.1.0/24

- 10.0.1.0 → Network address
- 10.0.1.1 → Reserved for VPC router
- 10.0.1.2 → Reserved for DNS
- 10.0.1.3 → Reserved for future use
- 10.0.1.255 → Broadcast address (not supported in VPC)

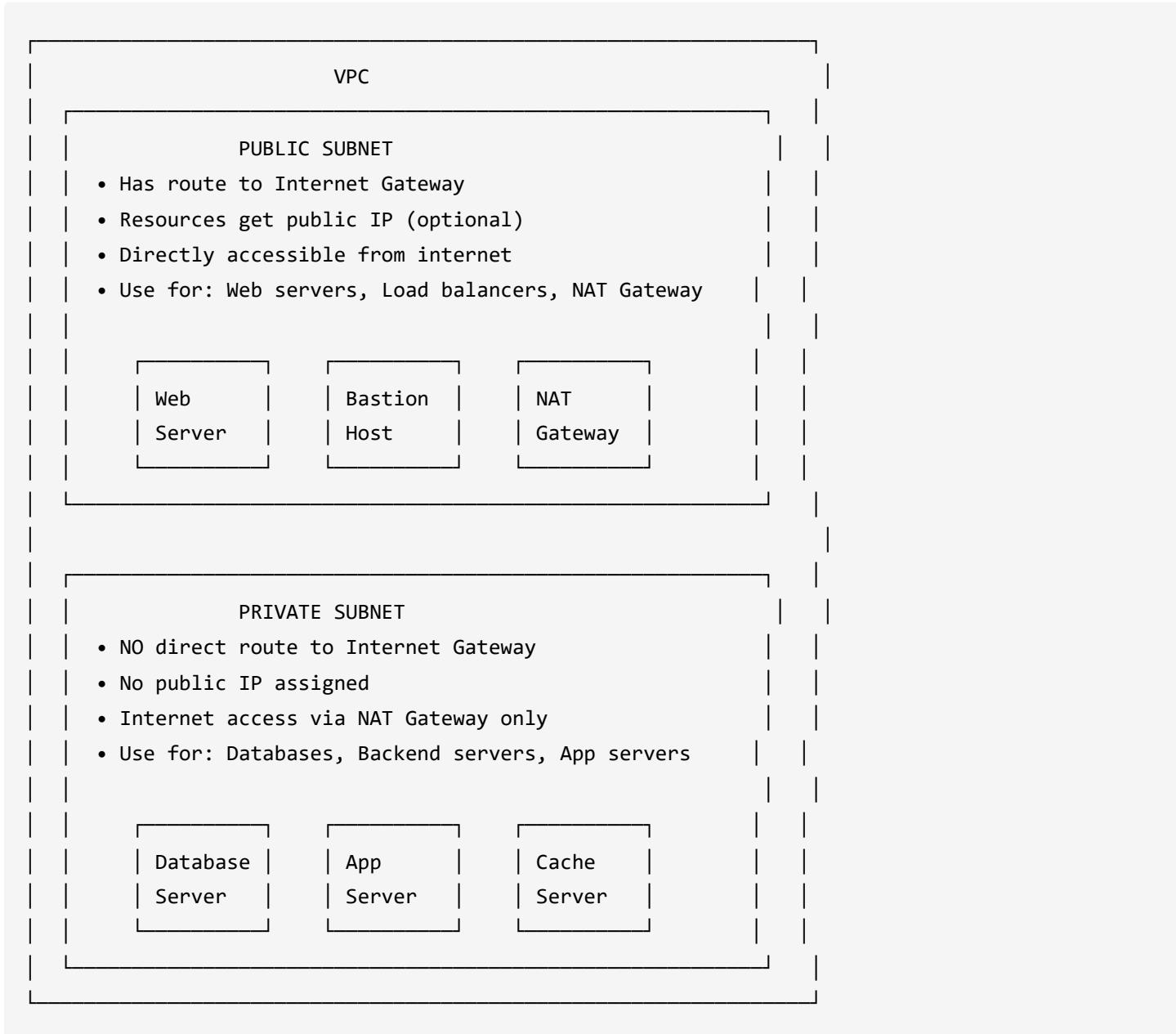
Available: 10.0.1.4 to 10.0.1.254 = 251 usable IPs

3. Subnets - Public & Private

What is a Subnet?

A **Subnet** is a range of IP addresses in your VPC. It's a sub-division of your VPC network.

Types of Subnets:



Key Differences:

Feature	Public Subnet	Private Subnet
Internet Gateway Route	Yes (0.0.0.0/0 → IGW)	No
Public IP	Can be assigned	Not assigned
Direct Internet Access	Yes	No
Internet Access Method	Direct via IGW	Via NAT Gateway
Use Case	Web servers, LBs	Databases, Backend

Subnet Best Practices:

1. **Multi-AZ**: Create subnets in multiple Availability Zones
 2. **Separation**: Keep public and private resources separated
 3. **Sizing**: Plan IP addresses based on future growth
 4. **Naming**: Use clear naming convention (prod-public-1a, prod-private-1a)
-

4. Internet Gateway (IGW)

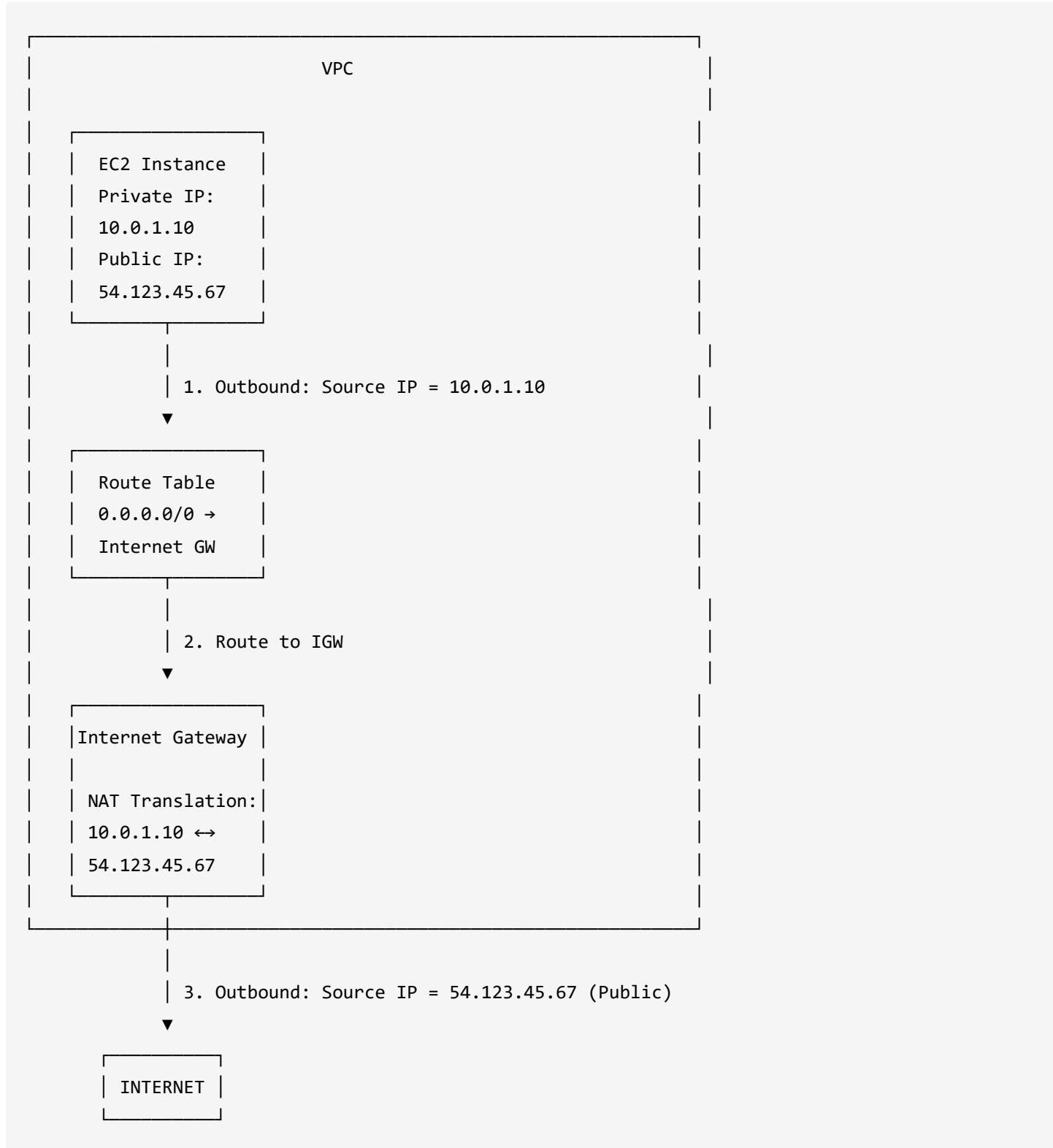
What is Internet Gateway?

Internet Gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between your VPC and the internet.

Internet Gateway Characteristics:

- **One per VPC**: Only one IGW can be attached to a VPC
- **Highly Available**: AWS manages redundancy automatically
- **No Bandwidth Constraints**: No availability risks or bandwidth limits
- **Free**: No additional charges for IGW

How Internet Gateway Works:



IGW Key Functions:

1. **Network Address Translation (NAT)**: Translates private IP to public IP
2. **Routing Target**: Acts as target for internet-bound traffic
3. **Bidirectional**: Supports both inbound and outbound traffic

5. Route Tables

What is a Route Table?

A **Route Table** contains a set of rules (routes) that determine where network traffic is directed.

Route Table Structure:

ROUTE TABLE		
Destination	Target	Status
10.0.0.0/16 (VPC CIDR)	local (within VPC)	Active
0.0.0.0/0 (All traffic)	igw-xxxxxxxx (Internet GW)	Active (Public RT only)
0.0.0.0/0 (All traffic)	nat-xxxxxxxx (NAT Gateway)	Active (Private RT only)

Types of Route Tables:

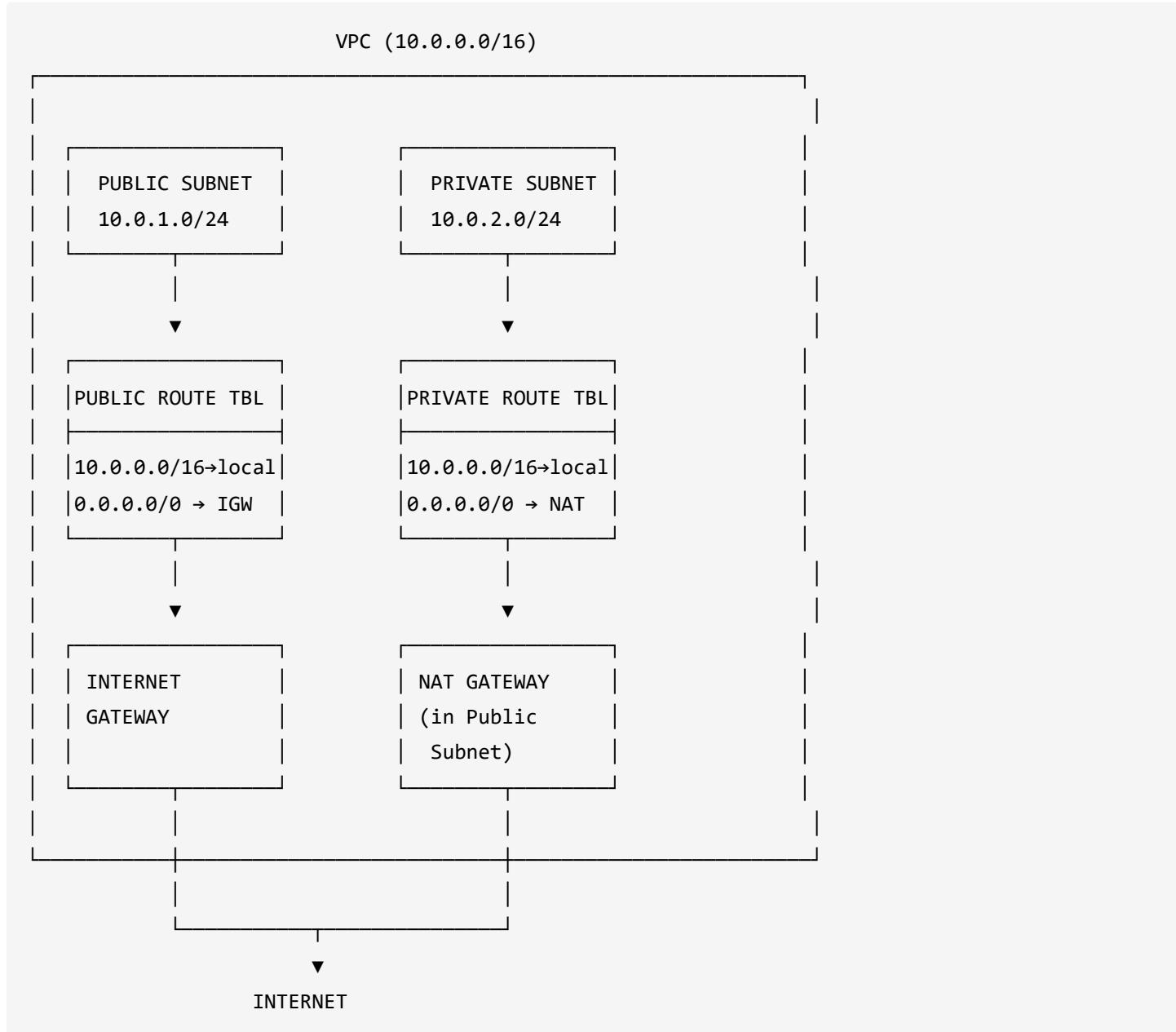
Main Route Table (Default):

- Auto-created with VPC
- Associated with subnets that don't have explicit association
- Called the "implicit" route table

Custom Route Table:

- User-created for specific subnet requirements
- Explicitly associated with subnets
- Allows different routing for different subnets

Route Table Architecture:



Subnet Association Rules:

- Each subnet MUST be associated with exactly ONE route table
- Multiple subnets can share one route table
- If no explicit association, subnet uses Main Route Table

6. NAT Gateway & NAT Instance

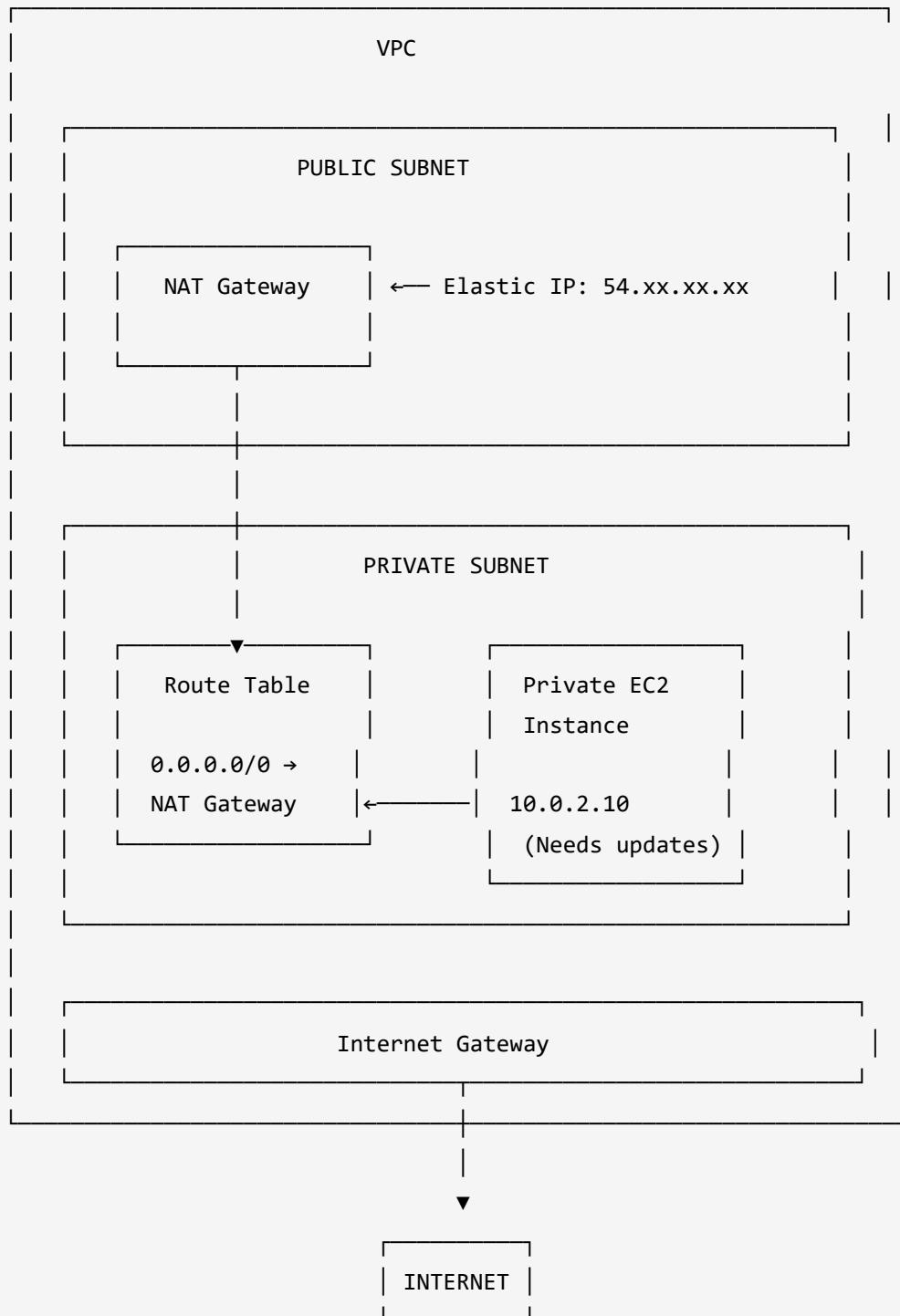
What is NAT?

NAT (Network Address Translation) enables instances in private subnets to connect to the internet while preventing the internet from initiating connections to those instances.

NAT Gateway vs NAT Instance:

Feature	NAT Gateway	NAT Instance
Managed by	AWS (Fully managed)	You (Self-managed)
Availability	Highly available in AZ	Must configure manually
Bandwidth	Up to 100 Gbps	Depends on instance type
Maintenance	No patching	You must patch
Cost	Hourly + data transfer	Instance cost + data
Security Group	Not supported	Supported
Bastion Host	Cannot be used as	Can be used as

NAT Gateway Architecture:



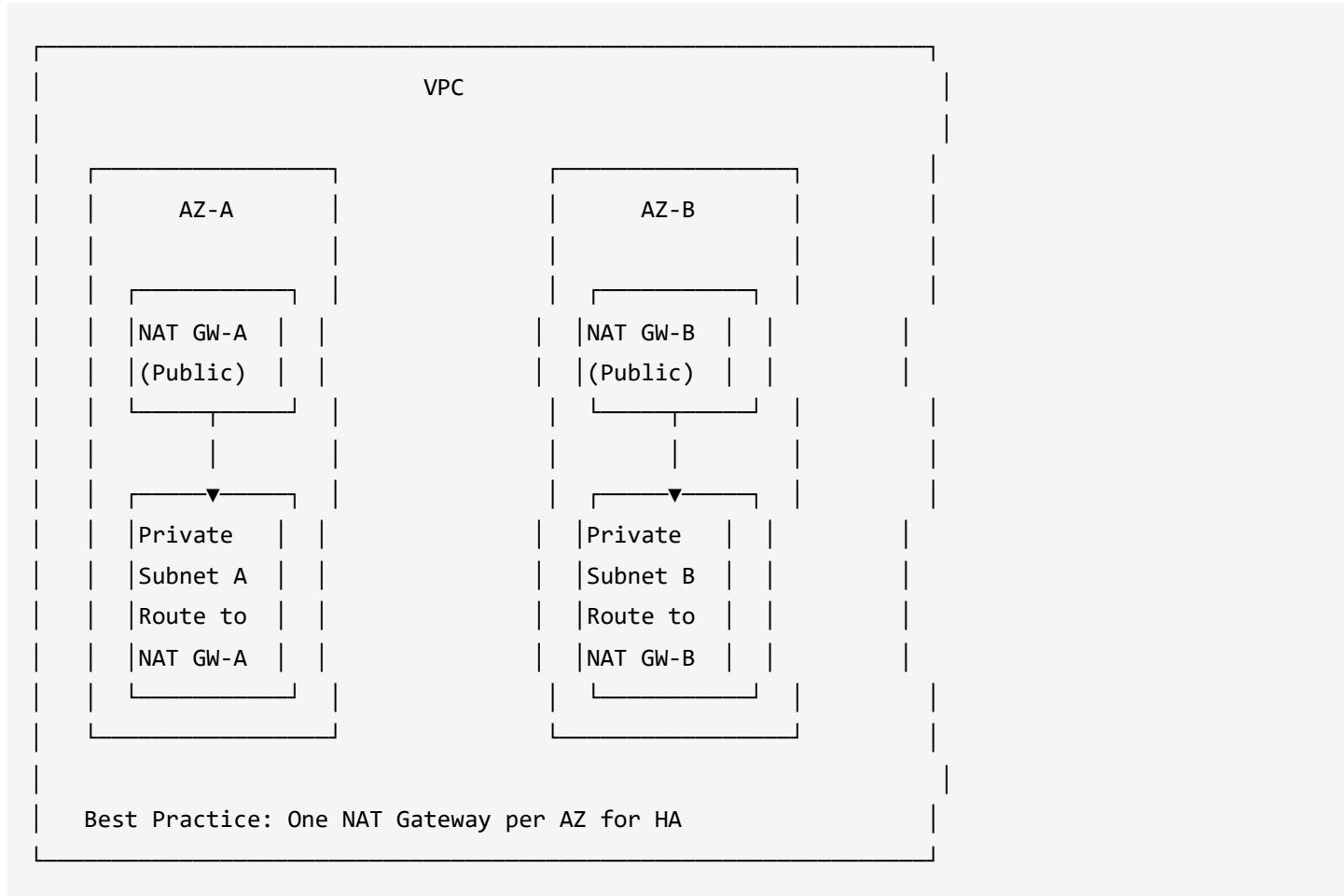
Traffic Flow:

1. Private EC2 (10.0.2.10) sends request to internet
2. Route Table forwards to NAT Gateway
3. NAT Gateway translates: 10.0.2.10 → 54.xx.xx.xx
4. Request goes through IGW to internet
5. Response comes back via same path

NAT Gateway Setup Steps:

1. Create NAT Gateway in PUBLIC subnet
2. Allocate Elastic IP to NAT Gateway
3. Update PRIVATE subnet's route table
4. Add route: 0.0.0.0/0 → NAT Gateway

High Availability for NAT Gateway:



7. Security Groups

What is a Security Group?

Security Group acts as a virtual firewall at the INSTANCE level, controlling inbound and outbound traffic.

Key Characteristics:

- **Stateful:** Return traffic automatically allowed
- **Allow Rules Only:** Cannot create deny rules

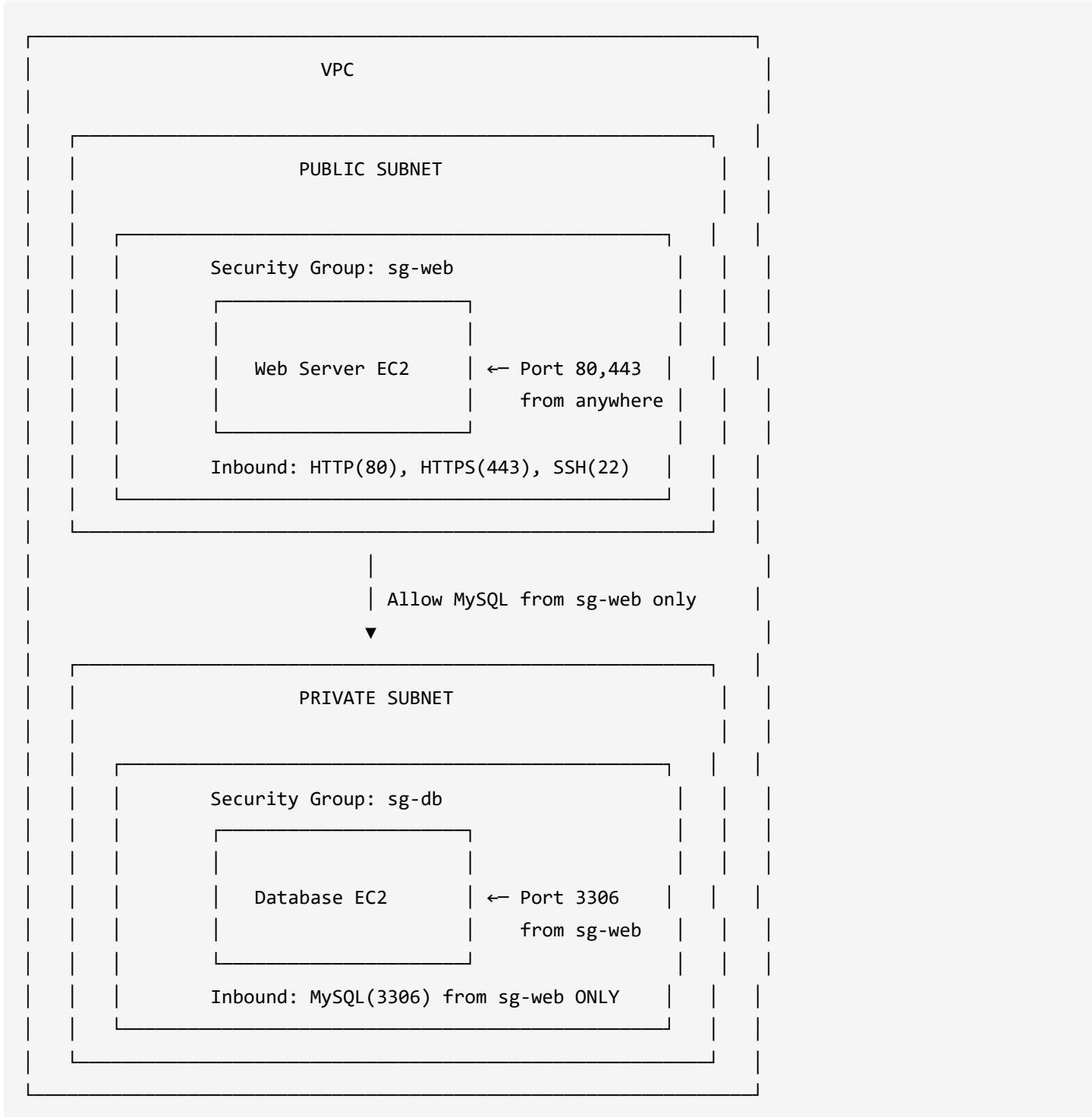
- **Instance Level:** Attached to ENI (Network Interface)
- **Default:** Deny all inbound, Allow all outbound
- **Multiple SGs:** Instance can have multiple security groups

Security Group Rules:

SECURITY GROUP			
INBOUND RULES			
Type	Protocol	Port Range	Source
SSH	TCP	22	My IP (203.0.113.0/32)
HTTP	TCP	80	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
Custom	TCP	3306	sg-webapp (SG ID)

OUTBOUND RULES			
Type	Protocol	Port Range	Destination
All	All	All	0.0.0.0/0 (Default)

Security Group Architecture:



Common Security Group Patterns:

```
# Web Server Security Group
```

Inbound:

- SSH (22) → My IP only
- HTTP (80) → 0.0.0.0/0
- HTTPS (443) → 0.0.0.0/0

```
# Database Security Group
```

Inbound:

- MySQL (3306) → sg-web-servers
- SSH (22) → sg-bastion-host

```
# Bastion Host Security Group
```

Inbound:

- SSH (22) → Corporate IP range only

8. Network ACLs (NACLs)

What is NACL?

Network Access Control List (NACL) is an optional layer of security that acts as a firewall for controlling traffic in and out of SUBNETS.

Security Group vs NACL:

Feature	Security Group	NACL
Level	Instance (ENI)	Subnet
State	Stateful	Stateless
Rules	Allow only	Allow AND Deny
Rule Processing	All rules evaluated	Rules processed in order
Default	Deny all inbound	Allow all
Association	Multiple per instance	One per subnet

NACL Rule Structure:

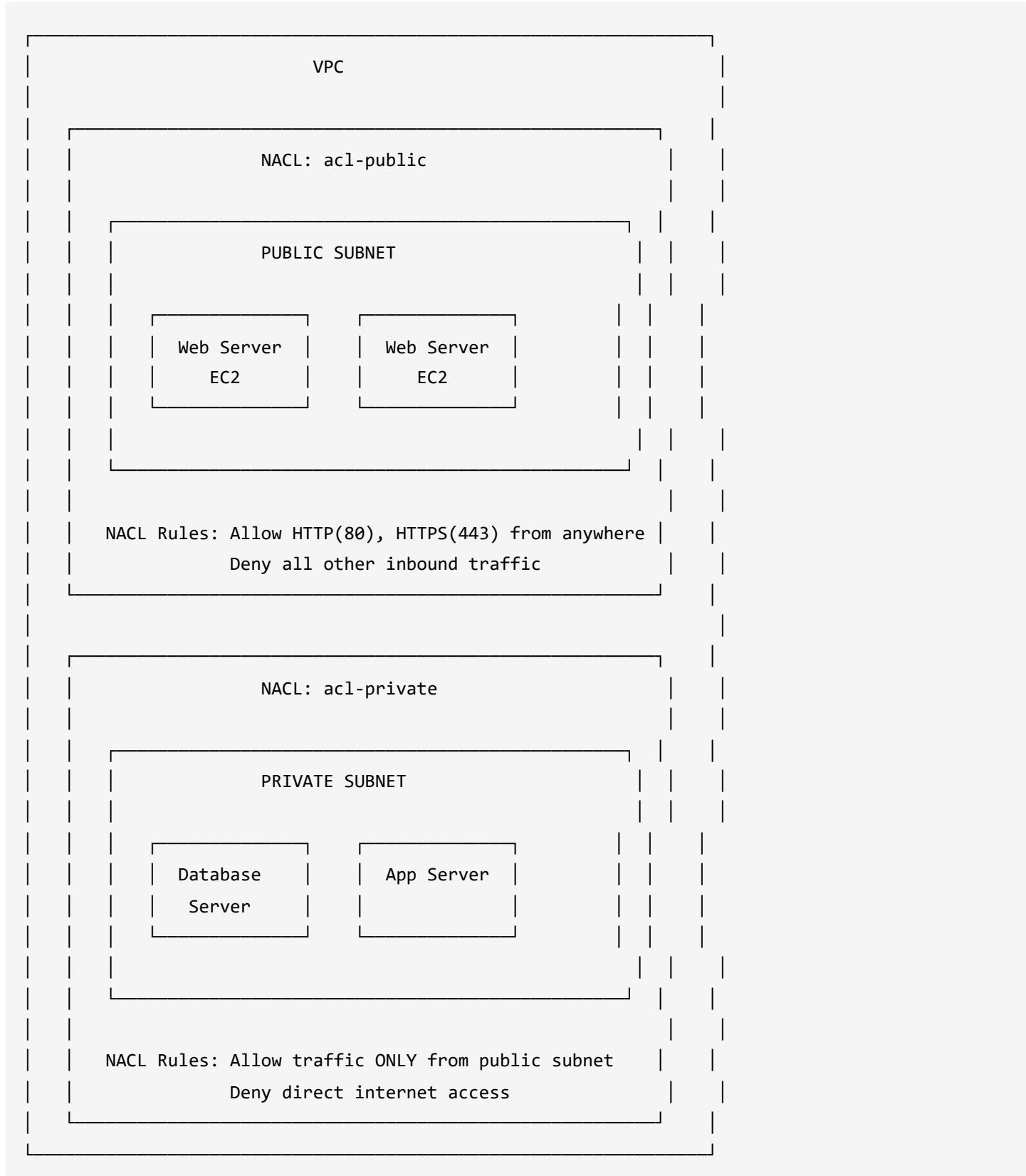
NETWORK ACL						
INBOUND RULES						
Rule	Type	Protocol	Port Range	Source	Allow/Deny	
100	HTTP	TCP	80	0.0.0.0/0	ALLOW	
110	HTTPS	TCP	443	0.0.0.0/0	ALLOW	
120	SSH	TCP	22	10.0.0.0/16	ALLOW	
*	ALL	ALL	ALL	0.0.0.0/0	DENY	

OUTBOUND RULES						
Rule	Type	Protocol	Port Range	Destination	Allow/Deny	
100	HTTP	TCP	80	0.0.0.0/0	ALLOW	
110	HTTPS	TCP	443	0.0.0.0/0	ALLOW	
120	Custom	TCP	1024-65535	0.0.0.0/0	ALLOW	
*	ALL	ALL	ALL	0.0.0.0/0	DENY	

Rule Processing: Rules evaluated in order (lowest number first)

First matching rule is applied

NACL Architecture:



NACL Important Points:

- Stateless:** Must define both inbound AND outbound rules
- Ephemeral Ports:** Remember to allow 1024-65535 for return traffic
- Rule Order:** Lower number = higher priority
- Default NACL:** Allows all traffic (created with VPC)

5. **Custom NACL:** Denies all traffic by default

9. VPC Peering

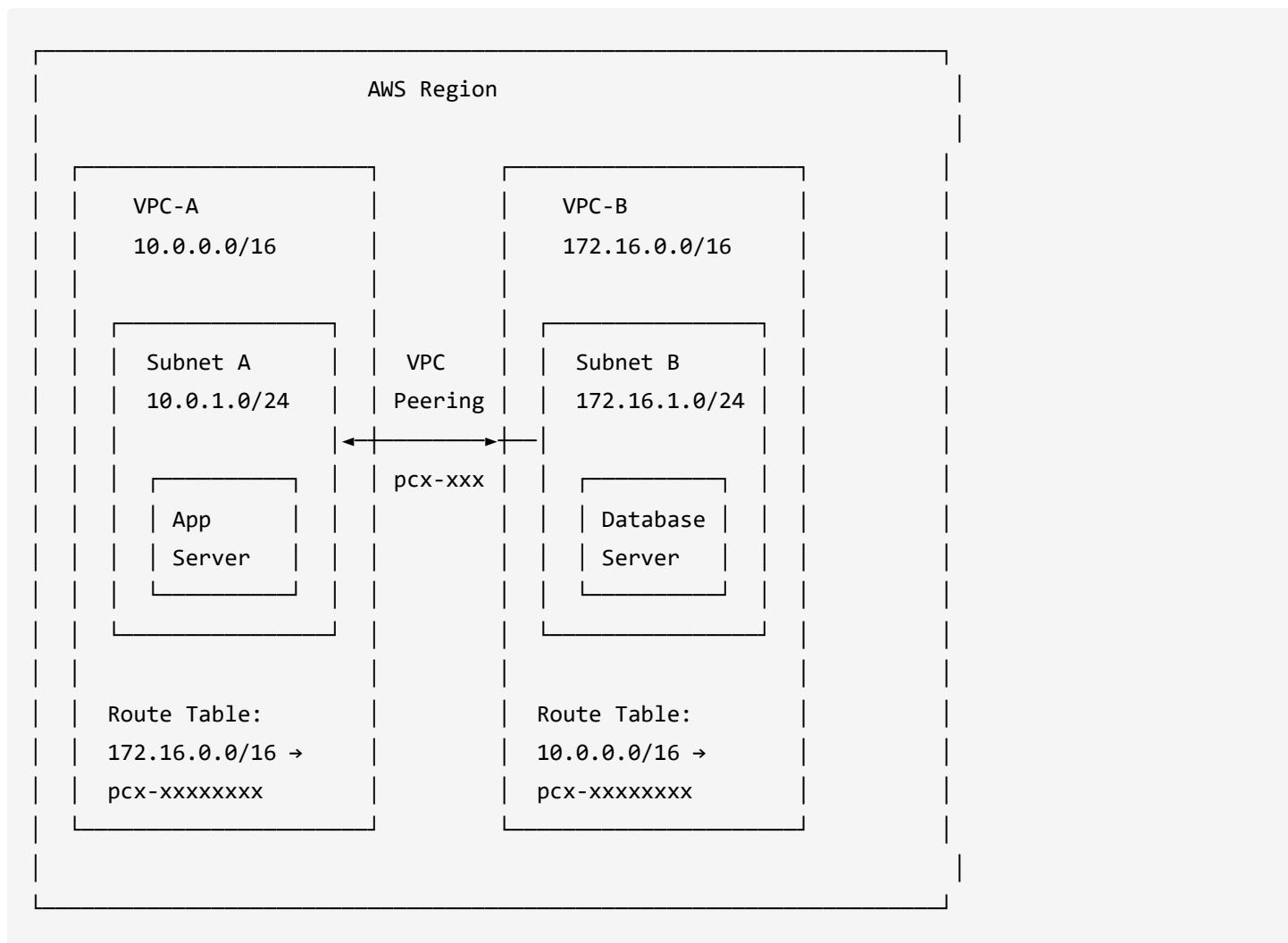
What is VPC Peering?

VPC Peering is a networking connection between two VPCs that enables you to route traffic between them using private IP addresses.

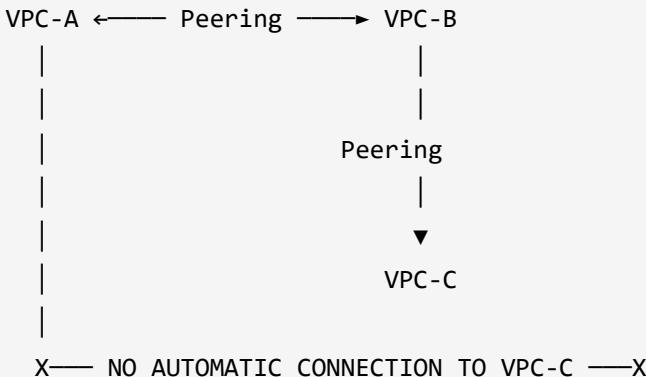
VPC Peering Characteristics:

- **Private Communication:** Traffic stays on AWS backbone
- **No Single Point of Failure:** Highly available
- **Cross-Region:** Supports inter-region peering
- **Cross-Account:** Can peer VPCs in different AWS accounts
- **Not Transitive:** A-B peered and B-C peered ≠ A-C peered

VPC Peering Architecture:



VPC Peering - Non-Transitive Nature:



To connect A to C: Must create separate peering between A and C

VPC Peering Setup Steps:

1. Create peering connection request from VPC-A
2. Accept request in VPC-B
3. Update Route Tables in BOTH VPCs
4. Update Security Groups to allow traffic

VPC Peering Limitations:

- **No Overlapping CIDR:** Cannot peer VPCs with overlapping IP ranges
- **No Edge-to-Edge Routing:** Cannot use one VPC to access another's IGW/VPN
- **Limit:** 125 peering connections per VPC

10. VPC Endpoints

What is VPC Endpoint?

VPC Endpoint enables private connections between your VPC and supported AWS services without requiring an internet gateway, NAT device, VPN, or AWS Direct Connect.

Types of VPC Endpoints:

1. Gateway Endpoint (Free):

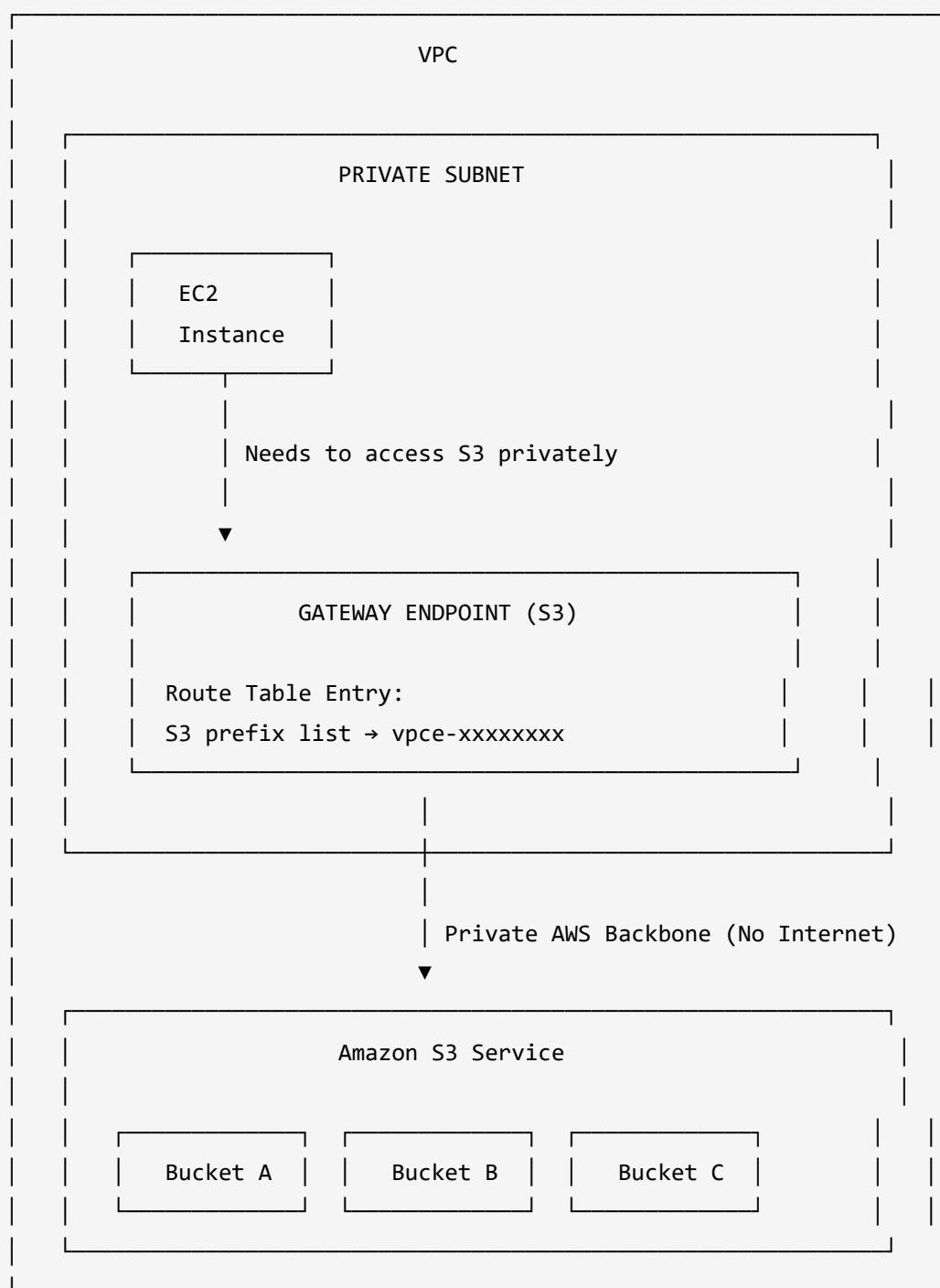
- Supported Services: **S3** and **DynamoDB** only
- Uses route table entries
- No Elastic Network Interface (ENI)

- Free to use

2. Interface Endpoint (Charged):

- Supported Services: Most AWS services
- Uses ENI with private IP
- Powered by AWS PrivateLink
- Hourly + data processing charges

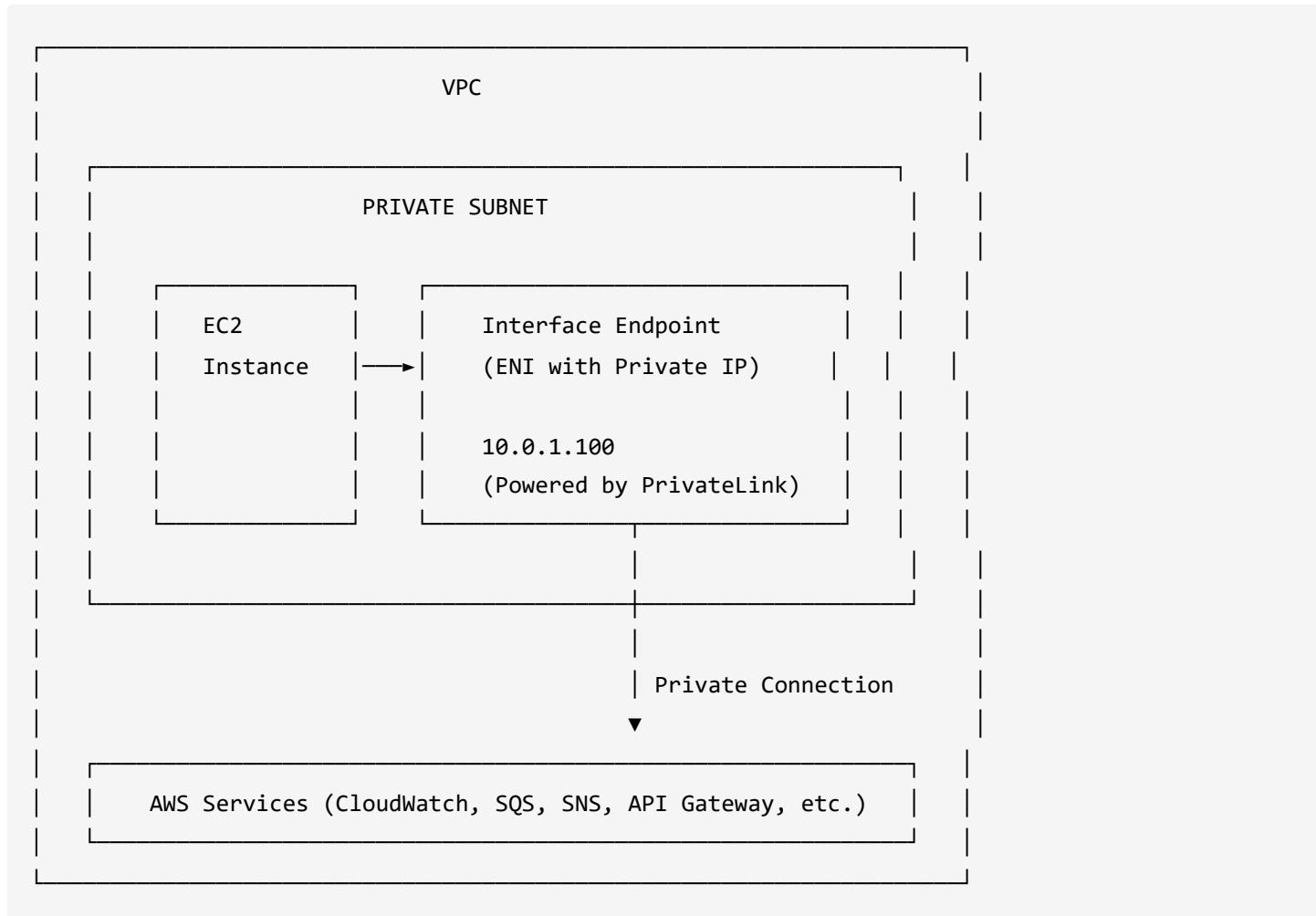
VPC Endpoint Architecture:



Benefits:

- ✓ Traffic never leaves AWS network
- ✓ No internet gateway needed
- ✓ No NAT gateway needed
- ✓ More secure
- ✓ Lower latency
- ✓ FREE for Gateway Endpoints

Interface Endpoint Architecture:



VPC Endpoint Benefits:

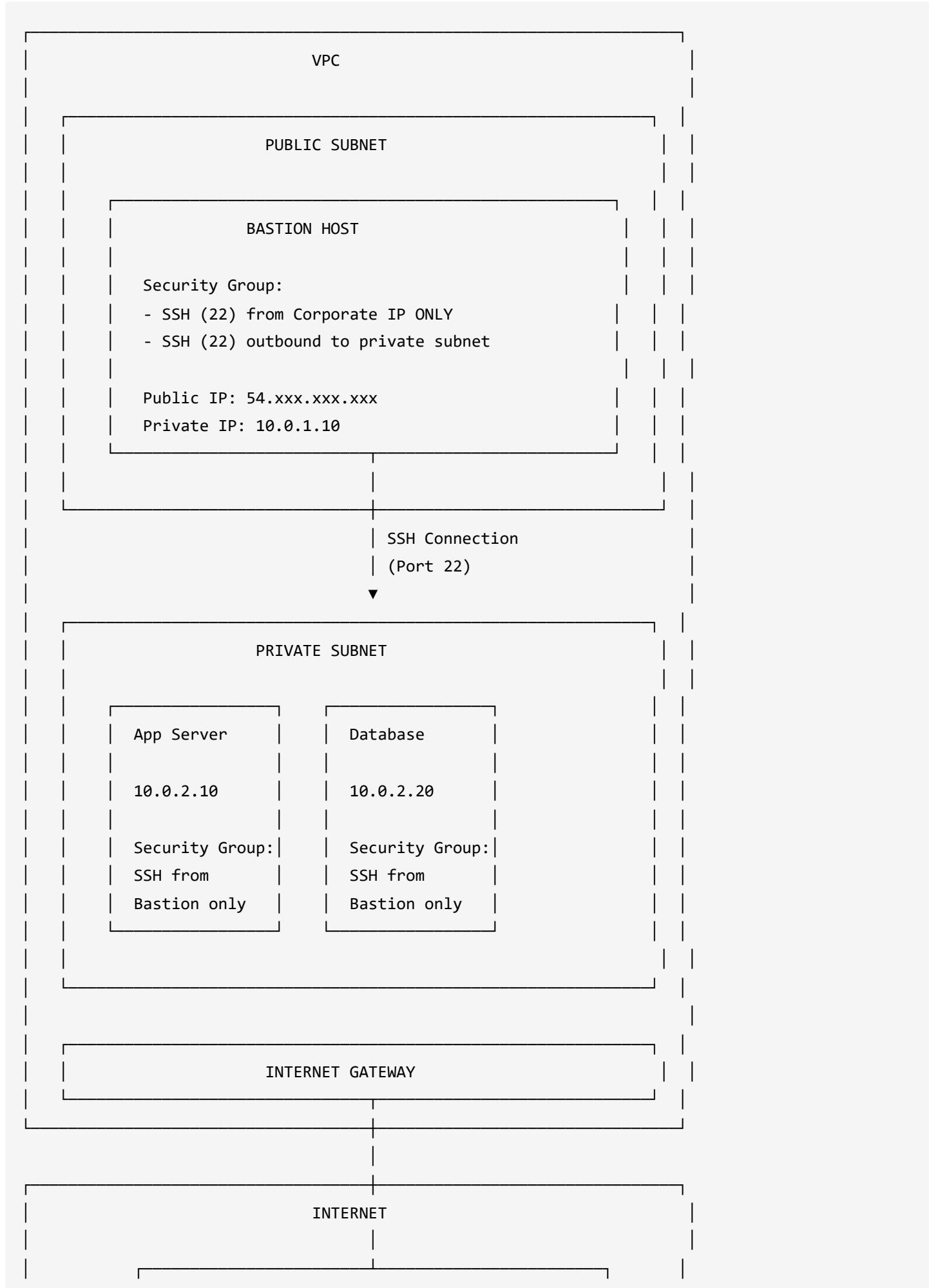
- Security**: Traffic doesn't traverse public internet
- Lower Latency**: Direct AWS backbone connection
- Cost**: No NAT gateway charges for S3/DynamoDB access
- Simplicity**: No need to manage NAT devices

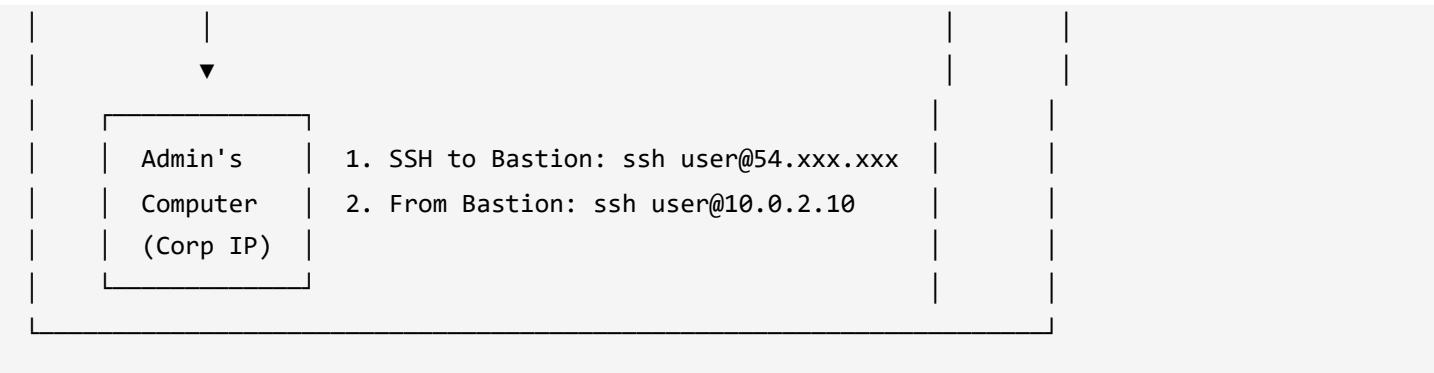
11. Bastion Host / Jump Server

What is Bastion Host?

Bastion Host (also called Jump Server) is a special-purpose EC2 instance in a public subnet that serves as a secure gateway to access instances in private subnets.

Bastion Host Architecture:





Bastion Host Security Best Practices:

Security Group for Bastion Host:

INBOUND:
SSH (22) → Corporate IP Range (e.g., 203.0.113.0/24)

OUTBOUND:
SSH (22) → Private Subnet CIDR (10.0.2.0/24)

Security Group for Private Instances:

INBOUND:
SSH (22) → Bastion Host Security Group (sg-bastion)

OUTBOUND:
As needed for application

Bastion Host Best Practices:

1. **Minimal Access:** Only SSH port open
2. **Restricted Source:** Allow only corporate/trusted IPs
3. **Hardening:** Keep patched, minimal software
4. **Logging:** Enable detailed logging
5. **MFA:** Use multi-factor authentication
6. **Session Manager:** Consider AWS Systems Manager Session Manager as alternative

12. Elastic IP Address

What is Elastic IP?

Elastic IP (EIP) is a static, public IPv4 address designed for dynamic cloud computing. It's associated with your AWS account, not a specific instance.

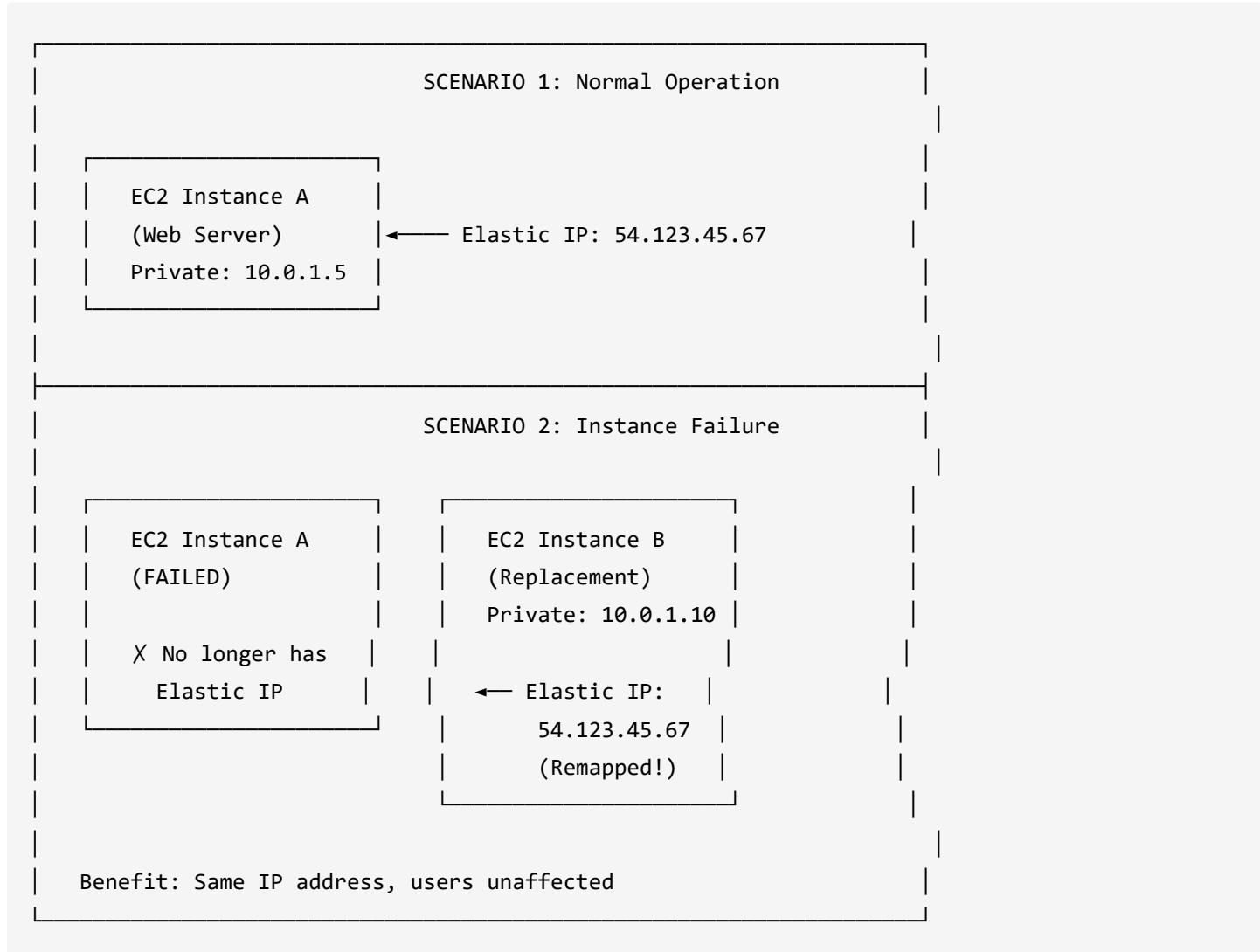
Elastic IP Characteristics:

- **Static:** Doesn't change like dynamic public IPs
- **Portable:** Can be remapped to different instances
- **Regional:** Specific to an AWS region
- **Limited:** 5 EIPs per region by default
- **Cost:** Free if attached and instance is running

Elastic IP vs Dynamic Public IP:

Feature	Elastic IP	Dynamic Public IP
Persistence	Remains after stop/start	Changes after stop/start
Reassignment	Can move between instances	Tied to instance lifecycle
Cost (running)	Free	Free
Cost (not attached)	Charged	N/A
Use Case	Fixed IP needed	Temporary/dynamic

Elastic IP Architecture:



Elastic IP Use Cases:

1. **Failover:** Quick IP remapping during instance failure
2. **DNS:** Point domain to a static IP
3. **Whitelisting:** When external systems whitelist your IP
4. **NAT Gateway:** Required for NAT Gateway

Elastic IP Costs:

ELASTIC IP PRICING	
Scenario	Cost
EIP attached to running instance	FREE
EIP attached to stopped instance	~\$0.005/hour
EIP not attached (idle)	~\$0.005/hour
Additional EIP on running instance	~\$0.005/hour
Tip: Release unused EIPs to avoid charges	

13. VPC Flow Logs

What are VPC Flow Logs?

VPC Flow Logs capture information about IP traffic going to and from network interfaces in your VPC. They're used for monitoring, troubleshooting, and security analysis.

Flow Log Levels:

VPC FLOW LOG LEVELS

VPC Level

Captures ALL traffic in entire VPC

Subnet Level

Captures traffic in specific subnet

ENI Level

Captures traffic on specific
Network Interface

Flow Log Record Format:

FLOW LOG RECORD EXAMPLE

```
version account-id interface-id srcaddr dstaddr srcport dstport
protocol packets bytes start end action log-status
```

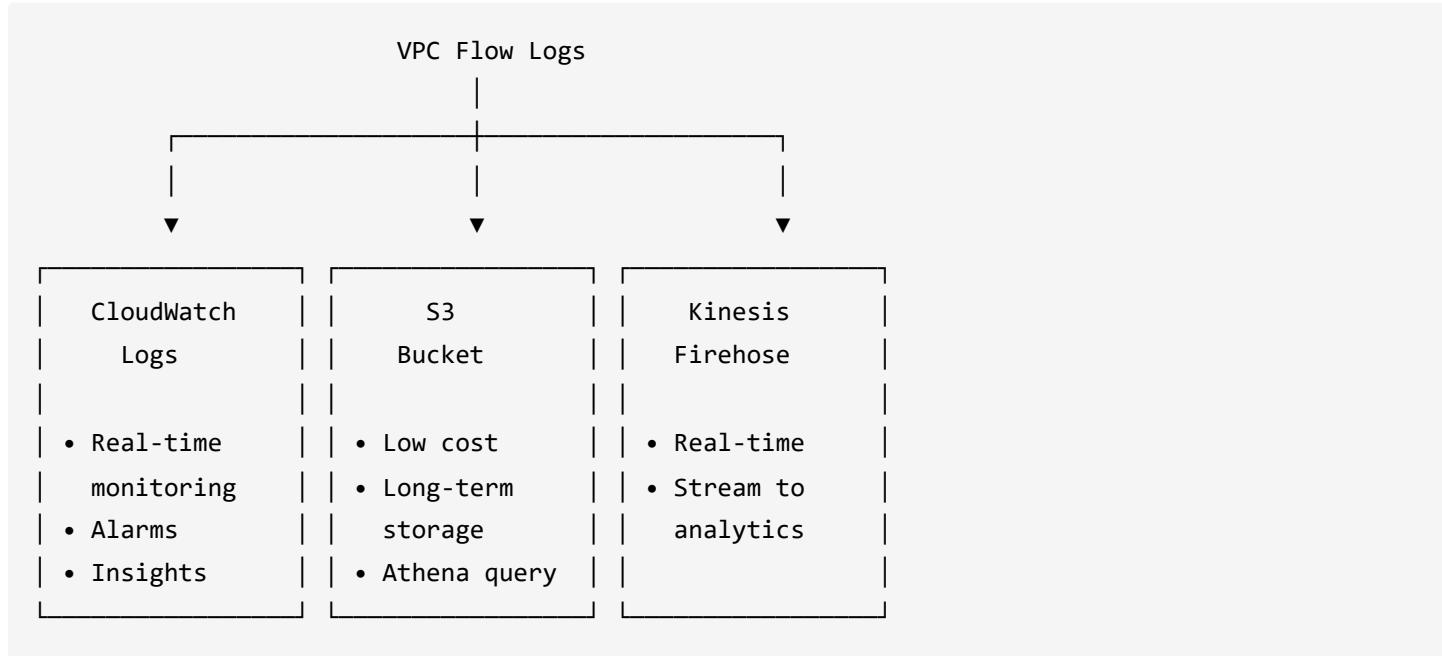
Example:

```
2 123456789012 eni-abc123 10.0.1.5 52.94.76.89 49152 443 6
10 5000 1620140761 1620140821 ACCEPT OK
```

Explanation:

- Source: 10.0.1.5:49152
- Dest: 52.94.76.89:443 (HTTPS)
- Protocol: 6 (TCP)
- Packets: 10, Bytes: 5000
- Action: ACCEPT (allowed by SG/NACL)

Flow Log Destinations:



Flow Log Use Cases:

1. **Security Analysis:** Detect unusual traffic patterns
2. **Troubleshooting:** Debug connectivity issues
3. **Compliance:** Audit network access
4. **Cost Optimization:** Identify heavy data transfer

Flow Log Limitations:

- Not captured: DHCP traffic, DNS to Amazon DNS, Windows license activation
- Not real-time: 10-15 minute delay
- Costs: Storage charges apply

14. AWS Direct Connect

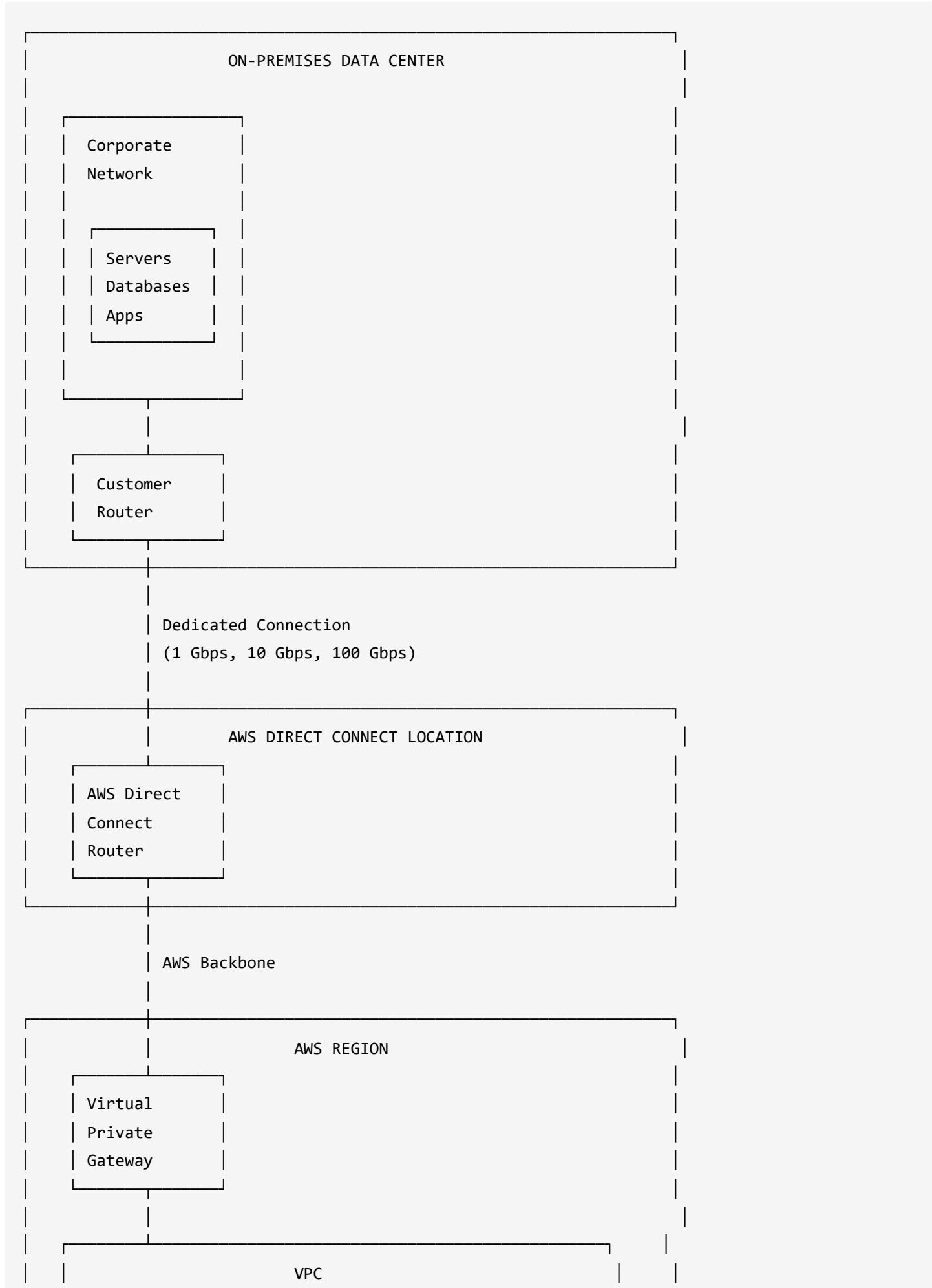
What is Direct Connect?

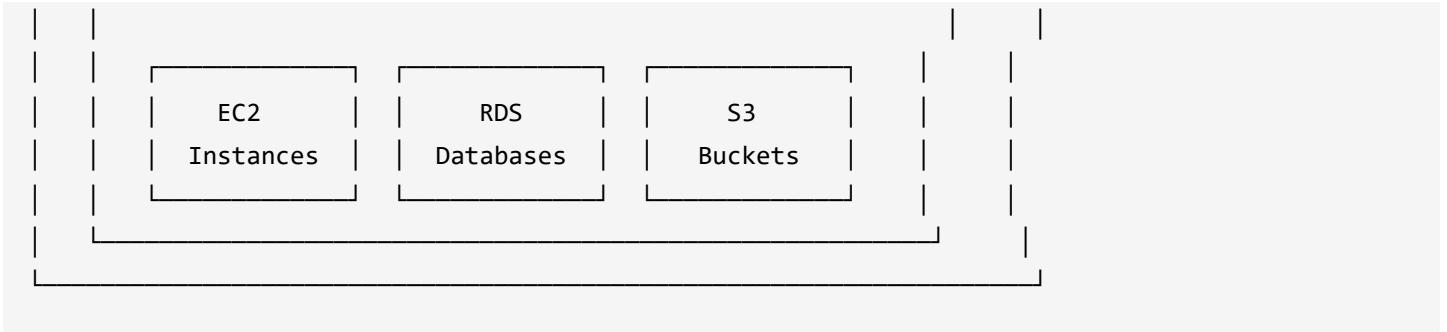
AWS Direct Connect establishes a dedicated private network connection from your on-premises data center to AWS.

Direct Connect vs VPN:

Feature	Direct Connect	Site-to-Site VPN
Connection	Dedicated physical	Over internet
Bandwidth	1 Gbps - 100 Gbps	Limited by internet
Latency	Consistent, low	Variable
Cost	Higher (dedicated)	Lower
Setup Time	Weeks/months	Minutes/hours
Encryption	Not by default	Encrypted

Direct Connect Architecture:





Direct Connect Benefits:

1. **Consistent Network Performance:** Dedicated bandwidth
 2. **Reduced Bandwidth Costs:** Lower data transfer rates
 3. **Private Connectivity:** Bypass public internet
 4. **Hybrid Cloud:** Seamless on-premises to AWS integration
-

15. AWS VPN (Client VPN & Site-to-Site)

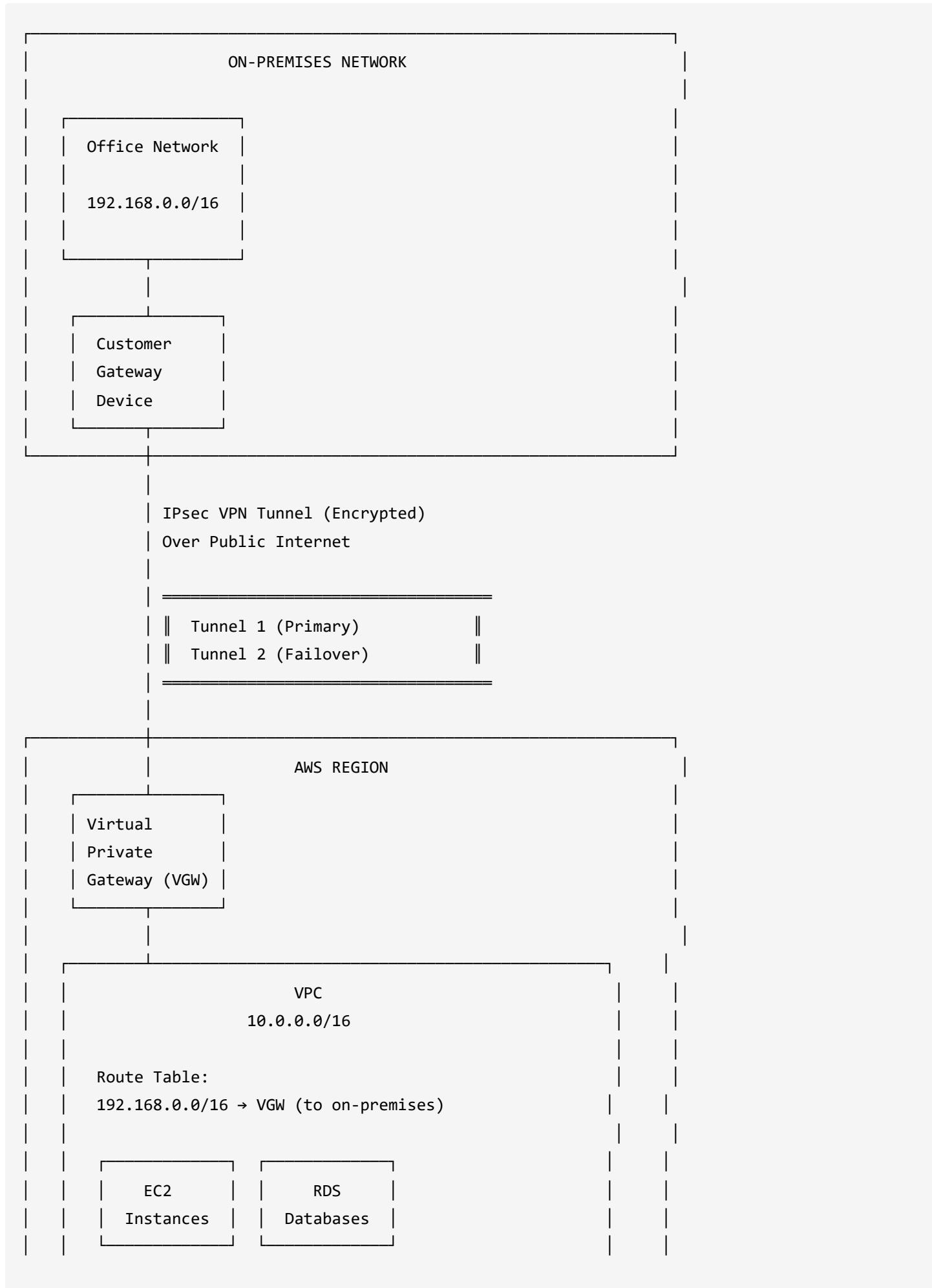
Site-to-Site VPN:

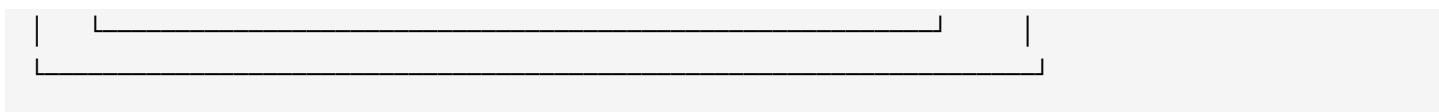
Connects your on-premises network to your VPC over an encrypted IPsec tunnel.

AWS Client VPN:

Enables remote users to securely access AWS resources and on-premises networks.

Site-to-Site VPN Architecture:





VPN Components:

Component	Description
Virtual Private Gateway (VGW)	AWS side of the VPN tunnel
Customer Gateway (CGW)	On-premises side device info
VPN Connection	Links VGW and CGW
IPsec Tunnels	Two tunnels for redundancy

16. Real AWS VPC Interview Questions (15 Questions)

Q1: What is the difference between Security Group and NACL? When would you use each?

Answer:

Aspect	Security Group	NACL
Level	Instance/ENI level	Subnet level
State	Stateful (return traffic auto-allowed)	Stateless (must explicitly allow both directions)
Rules	Allow rules ONLY	Allow AND Deny rules
Default	Deny all inbound, allow all outbound	Allow all (default NACL)
Processing	All rules evaluated together	Rules processed in order by rule number
Scope	Applied to specific instances	Applied to ALL instances in subnet

When to use:

- **Security Groups:** Primary firewall for most use cases. Use for application-level rules (e.g., allow HTTP on port 80 for web servers).
- **NACLs:** Use for subnet-wide blocking (e.g., block specific malicious IP ranges), as a secondary layer of defense, or when you need explicit deny rules.

Interview Tip: Always mention stateful vs stateless - this is the #1 follow-up question.

Q2: Explain how NAT Gateway works and why private instances need it?

Answer:

NAT Gateway translates private IPs to public IPs for outbound internet access.

How it works:

```
Private Instance (10.0.2.5) → NAT Gateway (Elastic IP: 54.1.2.3) → Internet
```

```
Response: Internet → NAT Gateway → Private Instance
```

Why needed:

- Private instances have NO public IP
- Cannot directly reach the internet
- NAT Gateway acts as a "translator" - internet sees NAT's Elastic IP, not the private IP
- Return traffic comes to NAT Gateway which routes back to the private instance

Key Points:

- Must be in public subnet
- Needs Elastic IP
- One-way: internet cannot initiate connections TO private instances
- Highly available within single AZ (deploy in each AZ for HA)

Q3: Your application in a private subnet cannot connect to the internet. How do you troubleshoot?

Answer:

Troubleshooting Checklist:

1. **NAT Gateway existence:** Is there a NAT Gateway in a public subnet?
2. **Route Table:** Does private subnet's route table have `0.0.0.0/0 → NAT Gateway` ?
3. **NAT Gateway's subnet:** Is NAT Gateway in a public subnet with route to IGW?
4. **Security Group:** Does the instance's SG allow outbound traffic?
5. **NACL:** Does the NACL allow outbound AND inbound return traffic?
6. **NAT Gateway status:** Is the NAT Gateway in "available" state?
7. **Elastic IP:** Does NAT Gateway have an Elastic IP attached?

Most Common Issues:

- Missing route to NAT Gateway in private subnet's route table
 - NAT Gateway in wrong subnet (private instead of public)
 - NACL blocking return traffic (ephemeral ports 1024-65535)
-

Q4: What are VPC Endpoints and when would you use Interface vs Gateway endpoints?

Answer:

VPC Endpoints provide private connectivity to AWS services without using the internet.

Feature	Gateway Endpoint	Interface Endpoint
Services	S3, DynamoDB ONLY	Most AWS services (100+)
Implementation	Route table entry	ENI with private IP in subnet
Cost	FREE	~\$0.01/hour + data processing
How it works	Routes traffic via AWS backbone	Uses PrivateLink
AZ consideration	Regional	AZ-specific (deploy per AZ)

Use Cases:

- **Gateway Endpoint:** S3/DynamoDB access from private subnets (always use - it's free!)
- **Interface Endpoint:** Services like Secrets Manager, SSM, KMS, ECR from private subnets

Interview Tip: Always mention S3 Gateway Endpoint is FREE - shows cost awareness.

Q5: How would you design a VPC for a 3-tier web application?

Answer:

VPC: 10.0.0.0/16	
AZ-1 (us-east-1a)	AZ-2 (us-east-1b)
Public Subnet	Public Subnet
10.0.1.0/24	10.0.2.0/24
- ALB	- ALB
- NAT Gateway	- NAT Gateway
- Bastion Host	
Private Subnet (Web)	Private Subnet (Web)
10.0.3.0/24	10.0.4.0/24
- Web Servers (ASG)	- Web Servers (ASG)
Private Subnet (App)	Private Subnet (App)
10.0.5.0/24	10.0.6.0/24
- Application Servers	- Application Servers
Private Subnet (DB)	Private Subnet (DB)
10.0.7.0/24	10.0.8.0/24
- RDS Primary	- RDS Standby (Multi-AZ)

Security Groups:

- ALB SG: Allow 80/443 from 0.0.0.0/0
- Web SG: Allow 80/443 from ALB SG only
- App SG: Allow app port from Web SG only
- DB SG: Allow 3306/5432 from App SG only

Q6: What are the 5 reserved IP addresses in a subnet and why?

Answer:

In every subnet, AWS reserves 5 IP addresses:

Reserved IP	Purpose	Example (/24)
First IP	Network address	10.0.1.0
First IP + 1	VPC Router	10.0.1.1
First IP + 2	DNS Server	10.0.1.2
First IP + 3	Future use	10.0.1.3

Reserved IP	Purpose	Example (/24)
Last IP	Broadcast	10.0.1.255

Calculation:

- /24 = 256 IPs - 5 reserved = **251 usable**
- /28 = 16 IPs - 5 reserved = **11 usable**
- /27 = 32 IPs - 5 reserved = **27 usable**

Interview Tip: This comes up often with CIDR questions. Know the formula: $2^{(32-\text{prefix})} - 5$

Q7: Explain VPC Peering limitations. What alternative would you use for complex connectivity?

Answer:

VPC Peering Limitations:

1. **Non-transitive:** VPC-A ↔ VPC-B and VPC-B ↔ VPC-C does NOT mean VPC-A ↔ VPC-C
2. **No overlapping CIDR:** Cannot peer VPCs with same IP ranges
3. **No edge-to-edge routing:** Cannot use peer's IGW, VPN, or Direct Connect
4. **125 peering connections max** per VPC
5. **Cross-region:** Supported but data transfer charges apply

Alternatives for Complex Connectivity:

Scenario	Solution
Many VPCs (hub-spoke)	AWS Transit Gateway
Private service sharing	PrivateLink
Full mesh connectivity	Transit Gateway
Cross-account services	PrivateLink or Resource Sharing

Transit Gateway Benefits:

- Central hub for thousands of VPCs
 - Transitive routing works
 - Single point of management
 - Works with VPN and Direct Connect
-

Q8: What is the difference between a public and private subnet?

Answer:

Technical Difference:

Aspect	Public Subnet	Private Subnet
Route to IGW	Yes (0.0.0.0/0 → IGW)	No
Public IP assignment	Can have public/Elastic IP	No public IP
Direct internet access	Yes (both inbound and outbound)	No
Internet access method	Directly via IGW	Via NAT Gateway

Key Insight: The ONLY difference is the route table configuration. A subnet becomes "public" when it has a route to an Internet Gateway.

Public Subnet Route Table:

10.0.0.0/16 → local

0.0.0.0/0 → igw-xxxxx ← Makes it PUBLIC

Private Subnet Route Table:

10.0.0.0/16 → local

0.0.0.0/0 → nat-xxxxx ← Routes to NAT (optional)

Q9: How do you secure SSH access to instances in a private subnet?

Answer:

Option 1: Bastion Host (Traditional)

Internet → Bastion (Public Subnet) → Private Instance

- Place Bastion in public subnet
- Restrict Bastion SG to your IP only
- Private instance SG allows SSH from Bastion SG only

Option 2: AWS Systems Manager Session Manager (Recommended)

- No Bastion needed
- No inbound ports required
- Audit trail in CloudTrail

- Works via SSM agent on instance
- Requires VPC Endpoint or NAT for SSM connectivity

Option 3: AWS Client VPN

- VPN into VPC
- Direct access to private instances
- Good for teams needing regular access

Best Practice: Use Session Manager for most cases - more secure, no SSH key management, full audit logging.

Q10: What happens to traffic when both Security Group and NACL have conflicting rules?

Answer:

Traffic must pass BOTH - they work in layers:

Inbound Traffic Flow:

Internet → NACL (checked first) → Security Group (checked second) → Instance

Outbound Traffic Flow:

Instance → Security Group (checked first) → NACL (checked second) → Internet

Example Scenario:

- NACL allows port 22 from 0.0.0.0/0
- Security Group denies port 22

Result: Traffic is BLOCKED. Both must allow for traffic to pass.

Another Scenario:

- NACL denies port 80 from specific IP
- Security Group allows port 80

Result: Traffic is BLOCKED at NACL before reaching Security Group.

Key Point: NACLs are evaluated first for inbound traffic. If NACL denies, Security Group is never reached.

Q11: How would you set up VPC Flow Logs for security monitoring?

Answer:

Setup Options:

1. **VPC Level:** Capture all traffic (comprehensive but high volume)
2. **Subnet Level:** Capture specific subnet traffic
3. **ENI Level:** Capture specific instance traffic

Best Practice Architecture:

```
VPC Flow Logs → S3 Bucket → Athena (for queries)
    → AWS Security Hub (alerts)
    → Third-party SIEM
```

```
VPC Flow Logs → CloudWatch Logs → Metric Filters → Alarms
```

Security Monitoring Use Cases:

- Detect port scans (many REJECT entries from same source)
- Identify unusual outbound traffic (data exfiltration)
- Track access to sensitive subnets
- Audit compliance requirements

Log Format Key Fields:

```
srcaddr, dstaddr, srcport, dstport, protocol, action (ACCEPT/REJECT)
```

Q12: Explain AWS Direct Connect vs Site-to-Site VPN. When to use each?

Answer:

Aspect	Direct Connect	Site-to-Site VPN
Connection	Dedicated physical fiber	Encrypted tunnel over internet
Bandwidth	1 Gbps - 100 Gbps	Limited by internet (~1.25 Gbps max)
Latency	Consistent, low	Variable (depends on internet)
Setup Time	Weeks to months	Minutes to hours
Monthly Cost		\$ (hourly + data)

Aspect	Direct Connect	Site-to-Site VPN
	(port hours + data)	
Encryption	Optional (add IPsec)	Built-in IPsec
Redundancy	Requires second connection	Two tunnels by default

When to Use Direct Connect:

- High bandwidth needs (>500 Mbps consistently)
- Latency-sensitive applications
- Large data migrations (TB/PB scale)
- Compliance requiring private connectivity
- Cost savings on high data transfer

When to Use VPN:

- Quick setup needed
- Lower bandwidth requirements
- Backup for Direct Connect
- Remote office connectivity
- Cost-sensitive scenarios

Best Practice: Use both - Direct Connect as primary, VPN as backup.

Q13: What is a CIDR block and how do you plan VPC CIDR ranges?

Answer:

CIDR (Classless Inter-Domain Routing) notation defines IP address ranges: 10.0.0.0/16

Understanding CIDR:

```
/16 = 65,536 IPs (10.0.0.0 - 10.0.255.255)
/24 = 256 IPs (10.0.1.0 - 10.0.1.255)
/28 = 16 IPs (smallest for AWS subnets)
```

VPC CIDR Planning Best Practices:

1. **Size appropriately:** Start with /16 for production (room to grow)
2. **Avoid overlaps:** Don't use 10.0.0.0/16 if on-premises uses it
3. **Plan for peering:** Peered VPCs cannot overlap
4. **Leave room:** Don't use consecutive ranges - leave gaps for expansion

Recommended Ranges:

```

Production VPC: 10.0.0.0/16
Development VPC: 10.1.0.0/16
Staging VPC: 10.2.0.0/16
(Leave 10.3.0.0/16 - 10.9.0.0/16 for future)

```

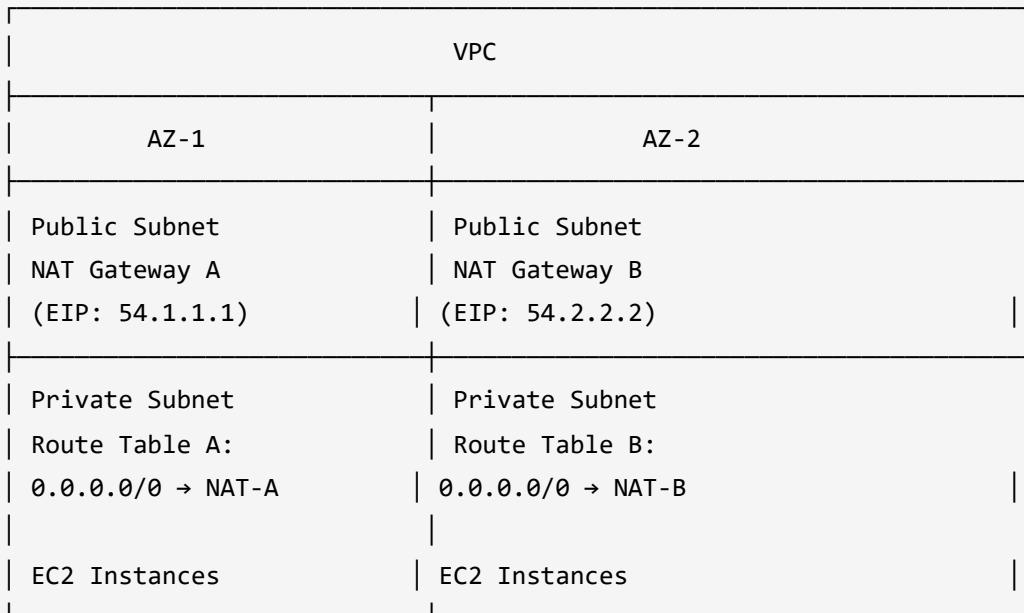
Avoid: 172.17.0.0/16 (Docker default) and any ranges used on-premises.

Q14: How do you achieve high availability for NAT Gateway?

Answer:

Problem: NAT Gateway is AZ-specific. If AZ fails, private instances in that AZ lose internet access.

Solution: NAT Gateway per AZ



Key Points:

- Create separate route table for each AZ's private subnets
 - Each route table points to its AZ's NAT Gateway
 - If AZ-1 fails, AZ-2 continues functioning independently
 - Cost: ~\$32/month per NAT Gateway + data processing
-

Q15: A customer reports they cannot access your application. How do you use VPC Flow Logs to troubleshoot?

Answer:

Troubleshooting Steps:

1. Enable Flow Logs (if not already):

```
aws ec2 create-flow-logs --resource-type VPC --resource-ids vpc-xxx \
--traffic-type ALL --log-destination-type cloud-watch-logs
```

2. Query for customer's IP (using CloudWatch Logs Insights):

```
fields @timestamp, srcAddr, dstAddr, srcPort, dstPort, action
| filter srcAddr = "203.0.113.50" # Customer IP
| sort @timestamp desc
| limit 100
```

3. Analyze Results:

Scenario A: "REJECT" entries

- Check Security Group rules
- Check NACL rules
- Verify the port is correct

Scenario B: "ACCEPT" but no response

- Application issue (not network)
- Check application logs
- Verify application is listening

Scenario C: No entries at all

- Traffic not reaching VPC
- Check DNS resolution
- Check route tables/IGW
- Customer-side issue (firewall, etc.)

4. Common Findings:

- NACL blocking ephemeral ports (1024-65535) for return traffic
- Security Group missing the specific port
- Wrong subnet/route table configuration

Quick Reference - AWS CLI Commands

```
# VPC Commands
aws ec2 create-vpc --cidr-block 10.0.0.0/16
aws ec2 describe-vpcs
aws ec2 delete-vpc --vpc-id vpc-xxxxxxxx

# Subnet Commands
aws ec2 create-subnet --vpc-id vpc-xxx --cidr-block 10.0.1.0/24 --availability-zone us-east-1a
aws ec2 describe-subnets

# Internet Gateway
aws ec2 create-internet-gateway
aws ec2 attach-internet-gateway --vpc-id vpc-xxx --internet-gateway-id igw-xxx

# Route Table
aws ec2 create-route-table --vpc-id vpc-xxx
aws ec2 create-route --route-table-id rtb-xxx --destination-cidr-block 0.0.0.0/0 --gateway-id igw-xxx

# NAT Gateway
aws ec2 create-nat-gateway --subnet-id subnet-xxx --allocation-id eipalloc-xxx

# Security Group
aws ec2 create-security-group --group-name MySecurityGroup --description "My SG" --vpc-id vpc-xxx
aws ec2 authorize-security-group-ingress --group-id sg-xxx --protocol tcp --port 22 --cidr 0.0.0.0/0

# VPC Peering
aws ec2 create-vpc-peering-connection --vpc-id vpc-xxx --peer-vpc-id vpc-yyy

# VPC Endpoint
aws ec2 create-vpc-endpoint --vpc-id vpc-xxx --service-name com.amazonaws.us-east-1.s3 --route-table-id rtb-xxx

# Flow Logs
aws ec2 create-flow-logs --resource-type VPC --resource-ids vpc-xxx --traffic-type ALL --log-destination arn:aws:logs:us-east-1:123456789012:log-group:/aws/vpc/flowLogs/vpc-xxx

# Elastic IP
aws ec2 allocate-address --domain vpc
aws ec2 associate-address --instance-id i-xxx --allocation-id eipalloc-xxx
```

Summary

Component	Purpose	Key Points
VPC	Isolated virtual network	Regional, 5 per region default
Subnet	Network subdivision	Public/Private, AZ-specific
Internet Gateway	Internet connectivity	One per VPC, free
NAT Gateway	Private outbound internet	In public subnet, needs EIP
Route Table	Traffic routing	Main vs Custom
Security Group	Instance firewall	Stateful, allow only
NACL	Subnet firewall	Stateless, allow/deny
VPC Peering	Connect two VPCs	Non-transitive
VPC Endpoint	Private AWS access	Gateway (free) vs Interface
Bastion Host	Secure access point	In public subnet
Elastic IP	Static public IP	Persistent across stop/start
Flow Logs	Traffic monitoring	To CW, S3, or Kinesis
Direct Connect	Dedicated connection	1-100 Gbps
Site-to-Site VPN	Encrypted tunnel	IPsec over internet

This completes the comprehensive guide to AWS VPC and Networking!

AWS ECS (Elastic Container Service) - Complete Notes

Table of Contents

1. [ECS Overview](#)
2. [Container Basics](#)

3. ECS Architecture
 4. Clusters
 5. Task Definitions
 6. Tasks
 7. Services
 8. Launch Types - Fargate vs EC2
 9. ECS Service Auto Scaling
 10. AWS Amplify
 11. Amazon Cognito
 12. Sandbox Environment
 13. ECR (Elastic Container Registry)
 14. Port Mapping & Networking
 15. Important Q&A
-

1. ECS Overview

What is ECS?

Amazon Elastic Container Service (ECS) is a fully managed container orchestration service that makes it easy to deploy, manage, and scale containerized applications.

In-Depth Explanation: The "Shipping Container" Analogy

Think of ECS like a **shipping port managing cargo containers**:

Shipping Industry	ECS Concept
Shipping Port	ECS Cluster
Shipping manifest (what's in the container)	Task Definition
Actual cargo container being shipped	Task (running container)
Shipping company managing multiple containers	Service
Warehouse where containers are stored	ECR (Container Registry)
Choice of ships (big/small)	Launch Type (EC2/Fargate)
Port authority managing everything	ECS Control Plane

Why ECS Matters (vs Managing Containers Yourself)

Without ECS (Manual Container Management):

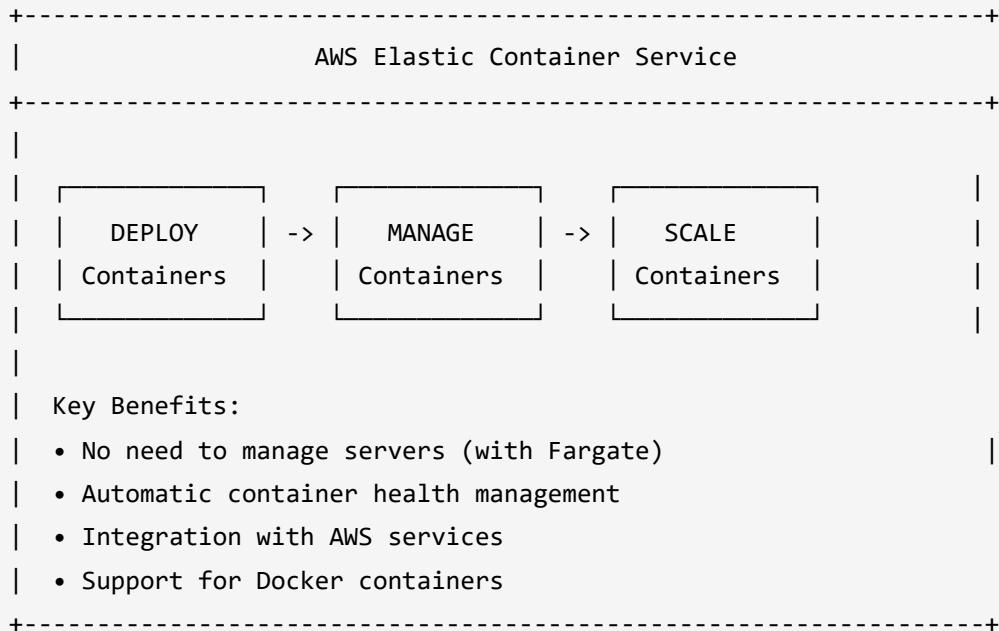
- Start container manually with docker run
- Monitor container health yourself
- Restart crashed containers manually
- Scale by running more docker commands
- No load balancing built-in
- Complex networking setup

With ECS:

- Define what you want (Task Definition)
- Tell ECS "keep 5 containers running"
- ECS handles: health, restarts, scaling, networking, load balancing
- Integrate with ALB, CloudWatch, IAM automatically

The ECS Value Proposition

Scenario	Without ECS	With ECS
Container crashes	Manual restart	Auto-restart in seconds
Traffic spike	Manual scaling	Auto Scaling policies
New deployment	SSH, docker stop/start	Rolling update, zero downtime
Monitoring	Custom setup	CloudWatch integration
Security	DIY IAM, networking	Built-in IAM roles, VPC



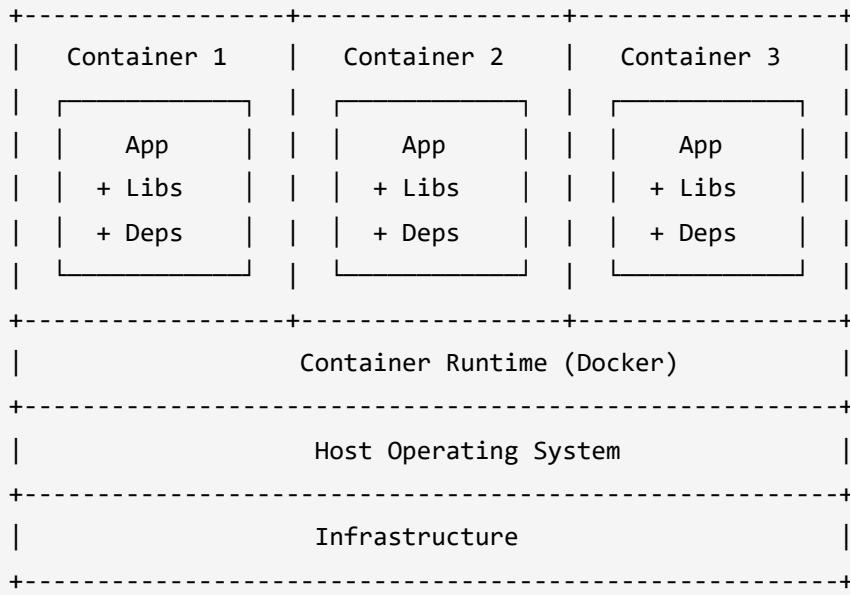
Key Features

Feature	Description
Fully Managed	AWS handles the control plane
Docker Support	Run standard Docker containers
AWS Integration	Works with IAM, VPC, CloudWatch, ALB
Flexible Launch	Choose between Fargate and EC2
Auto Scaling	Scale based on demand

2. Container Basics

What is a Container?

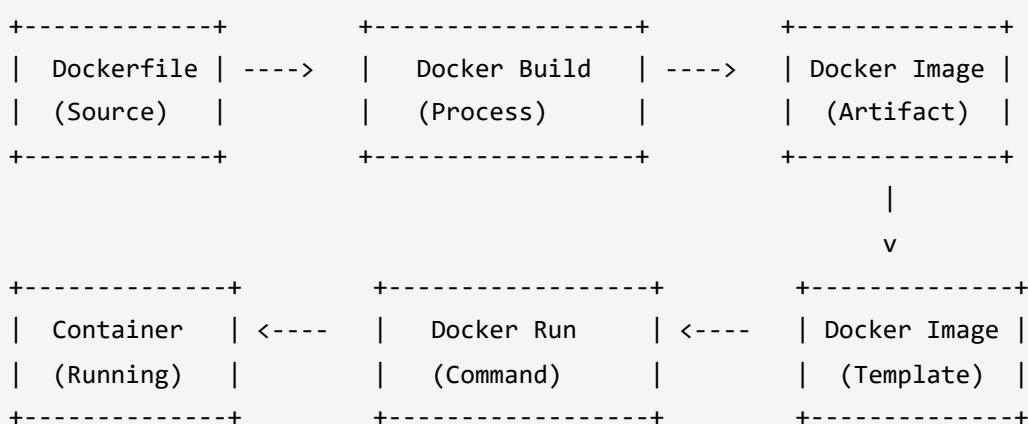
A **container** is a lightweight, standalone, executable package that includes everything needed to run an application: code, runtime, system tools, libraries, and settings.



Docker vs Virtual Machine

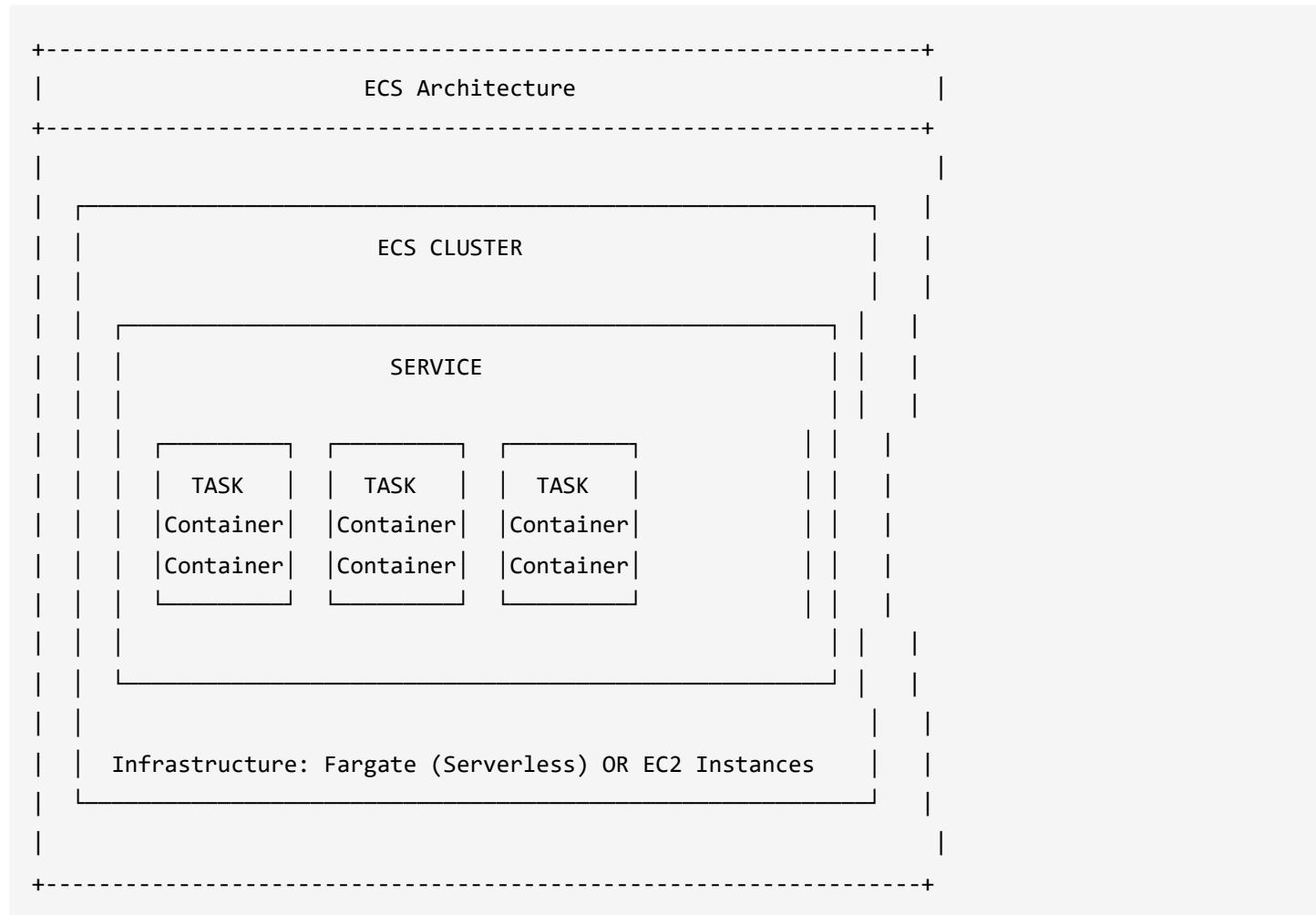
Aspect	Container (Docker)	Virtual Machine
Boot Time	Seconds	Minutes
Size	MBs	GBs
OS	Shares host OS kernel	Has own OS
Isolation	Process-level	Hardware-level
Performance	Near native	Overhead
Portability	Highly portable	Less portable

Docker Image Workflow

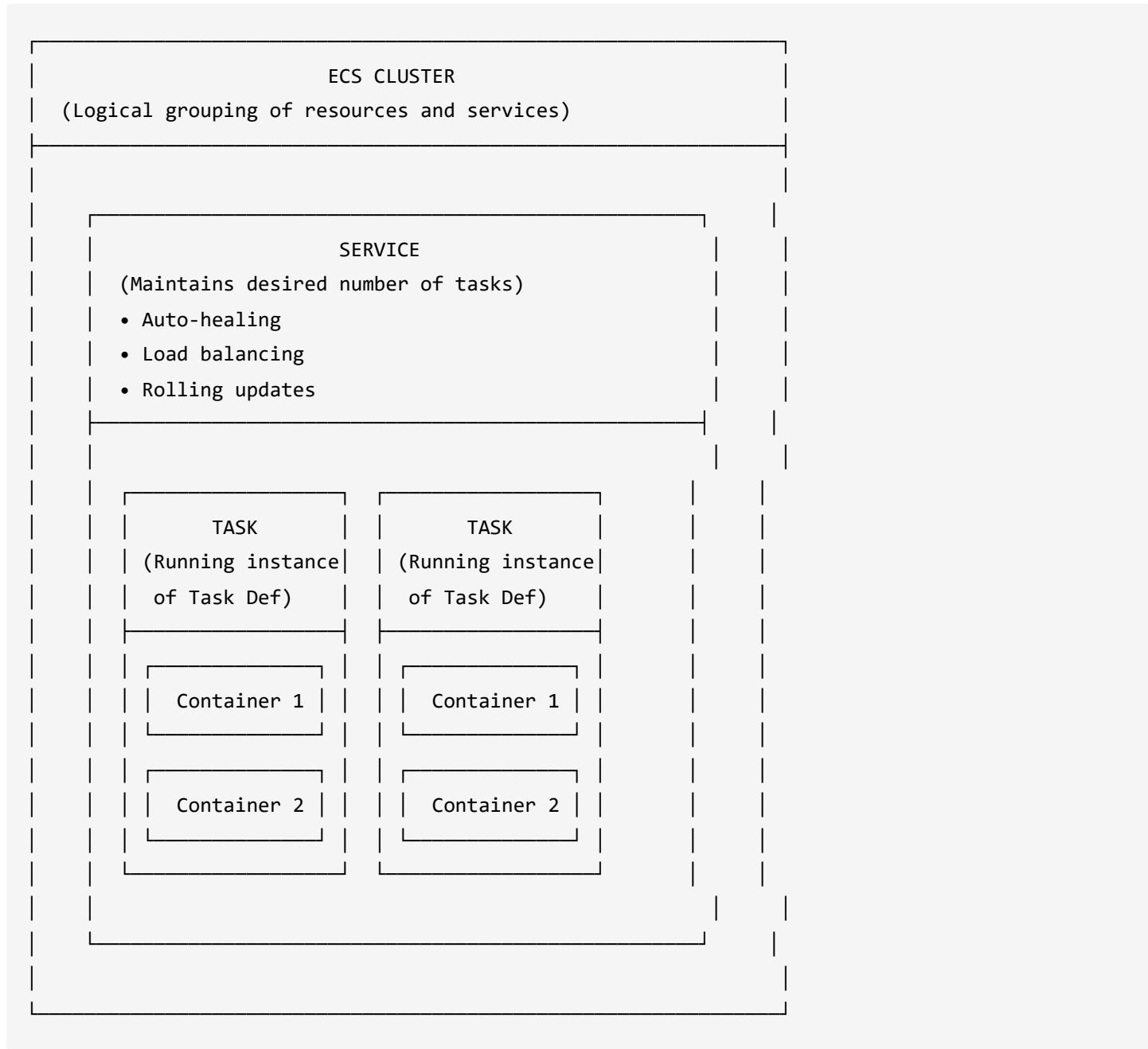


3. ECS Architecture

High-Level Architecture



ECS Components Hierarchy



4. Clusters

What is a Cluster?

A **Cluster** is a logical grouping of tasks and services. It acts as a boundary for your resources.

Cluster Types

ECS CLUSTER TYPES

1. FARGATE CLUSTER (Serverless)

- No EC2 instances to manage
- Pay per task
- AWS manages infrastructure
- Best for: Variable workloads

2. EC2 CLUSTER (Self-managed)

- You manage EC2 instances
- More control over infrastructure
- Good for: Persistent workloads
- Can use spot instances

3. EXTERNAL (ECS Anywhere)

- Run on your own servers
- On-premises support
- Hybrid cloud scenarios

Creating a Cluster (AWS CLI)

```
# Create a Fargate cluster
aws ecs create-cluster \
--cluster-name my-cluster \
--capacity-providers FARGATE FARGATE_SPOT

# Create cluster with EC2 instances
aws ecs create-cluster \
--cluster-name my-ec2-cluster \
--capacity-providers FARGATE

# List clusters
aws ecs list-clusters

# Describe cluster
aws ecs describe-clusters --clusters my-cluster
```

5. Task Definitions

What is a Task Definition?

A **Task Definition** is a blueprint for your application. It's a JSON document that describes one or more containers.

TASK DEFINITION

(Blueprint for your application)

Container Definitions:

Container 1

- Image: nginx
- CPU: 256
- Memory: 512MB
- Port: 80
- Env Variables

Container 2

- Image: redis
- CPU: 256
- Memory: 512MB
- Port: 6379
- Env Variables

Task Level Settings:

- Task Role (IAM permissions)
- Network Mode (awsvpc, bridge, host)
- Launch Type (Fargate, EC2)
- CPU & Memory (Task-level allocation)
- Volumes

Task Definition JSON Example

```
{
  "family": "my-app",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256",
  "memory": "512",
  "executionRoleArn": "arn:aws:iam::123456789:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "web-app",
      "image": "nginx:latest",
      "cpu": 256,
      "memory": 512,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ],
      "environment": [
        {
          "name": "ENV",
          "value": "production"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/my-app",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ]
}
```

Task Definition Key Parameters

Parameter	Description	Example
family	Task definition name	"my-app"

Parameter	Description	Example
cpu	Total CPU units	"256", "512", "1024"
memory	Total memory (MB)	"512", "1024", "2048"
networkMode	Networking type	"awsvpc", "bridge", "host"
image	Docker image	"nginx:latest"
portMappings	Container port config	containerPort: 80
essential	Stop task if container fails	true/false
environment	Environment variables	key-value pairs

Register Task Definition (CLI)

```
# Register task definition
aws ecs register-task-definition \
--cli-input-json file://task-definition.json

# List task definitions
aws ecs list-task-definitions

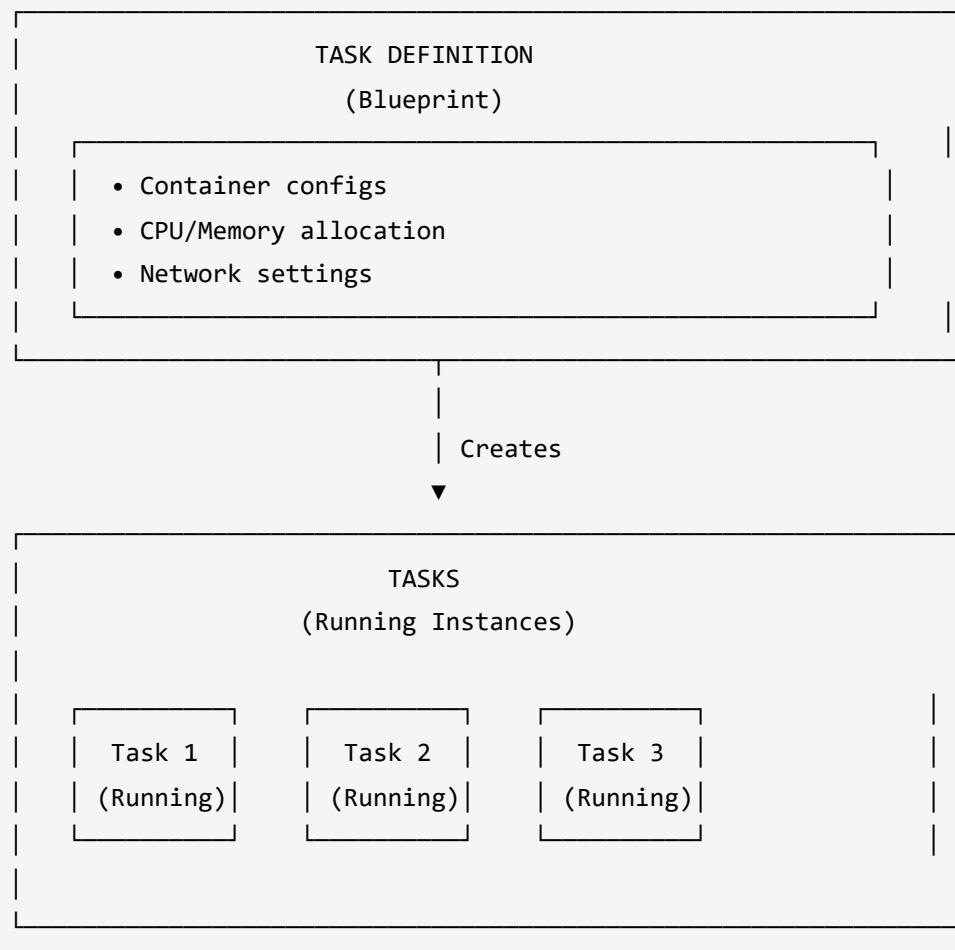
# Describe task definition
aws ecs describe-task-definition --task-definition my-app:1

# Deregister task definition
aws ecs deregister-task-definition --task-definition my-app:1
```

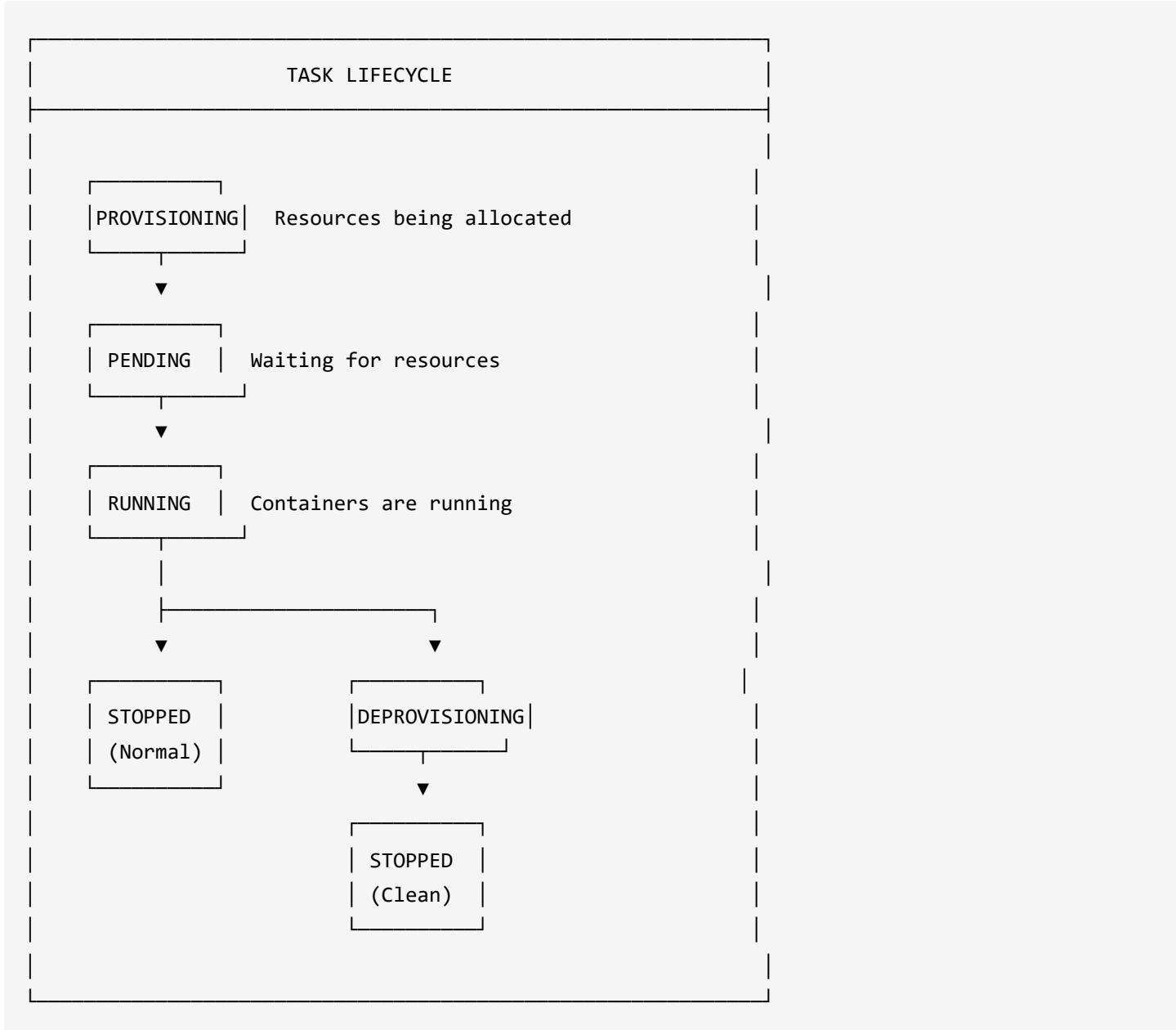
6. Tasks

What is a Task?

A **Task** is a running instance of a Task Definition. Think of Task Definition as a class and Task as an object.



Task States



Run Task (CLI)

```
# Run a task
aws ecs run-task \
--cluster my-cluster \
--task-definition my-app:1 \
--launch-type FARGATE \
--network-configuration "awsvpcConfiguration={subnets=[subnet-xxx],securityGroups=[sg-xxx],as

# List tasks
aws ecs list-tasks --cluster my-cluster

# Describe task
aws ecs describe-tasks \
--cluster my-cluster \
--tasks arn:aws:ecs:region:account:task/cluster/task-id

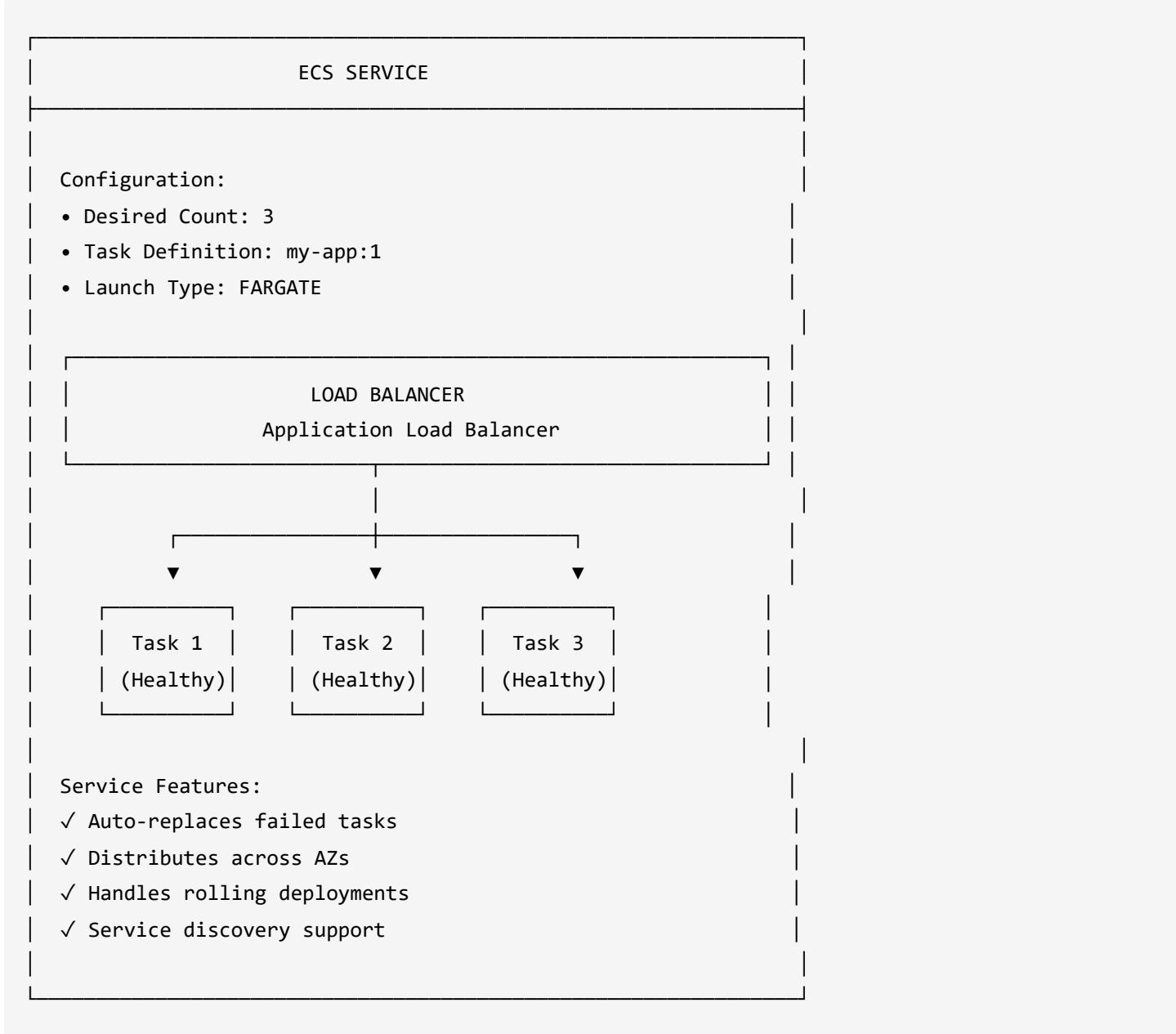
# Stop task
aws ecs stop-task \
--cluster my-cluster \
--task arn:aws:ecs:region:account:task/cluster/task-id
```

7. Services

What is a Service?

A **Service** maintains a specified number of tasks running simultaneously. It provides:

- **Desired Count:** Maintains specified number of tasks
- **Auto-healing:** Replaces unhealthy tasks
- **Load Balancing:** Distributes traffic across tasks
- **Rolling Updates:** Zero-downtime deployments



Service vs Standalone Task

Aspect	Service	Standalone Task
Management	Managed by ECS	Manual
Count	Maintains desired count	One-time execution
Auto-healing	Yes	No
Load Balancing	Supported	Not applicable
Use Case	Web apps, APIs	Batch jobs, one-off tasks

Create Service (CLI)

```
# Create service with ALB
aws ecs create-service \
--cluster my-cluster \
--service-name my-service \
--task-definition my-app:1 \
--desired-count 3 \
--launch-type FARGATE \
--network-configuration "awsvpcConfiguration={subnets=[subnet-xxx],securityGroups=[sg-xxx],as \
--load-balancers "targetGroupArn=arn:aws:elasticloadbalancing:...,containerName=web-app,conta

# Update service (rolling update)
aws ecs update-service \
--cluster my-cluster \
--service my-service \
--task-definition my-app:2

# Scale service
aws ecs update-service \
--cluster my-cluster \
--service my-service \
--desired-count 5

# Delete service
aws ecs delete-service \
--cluster my-cluster \
--service my-service \
--force
```

8. Launch Types - Fargate vs EC2

The Critical Decision: Fargate vs EC2

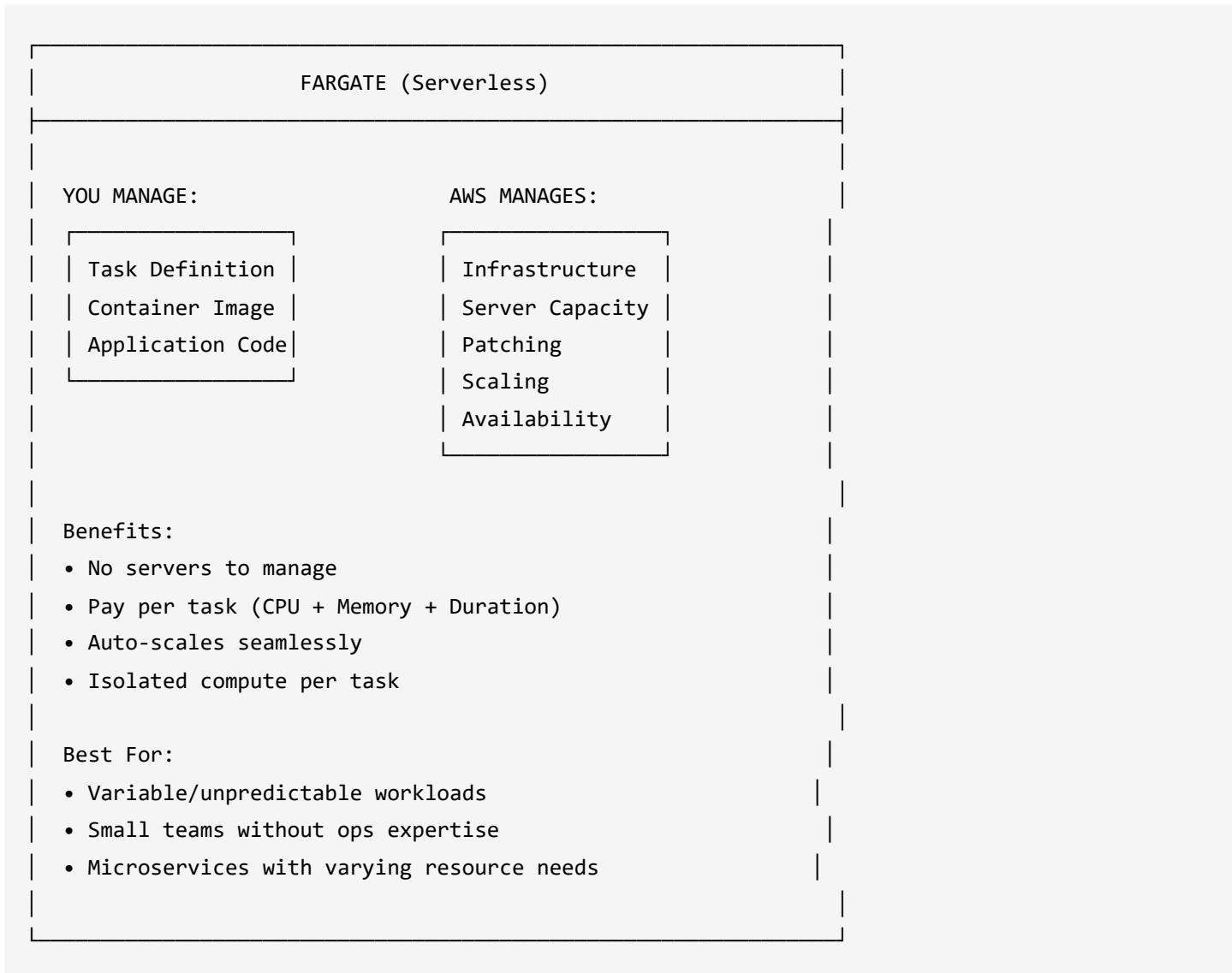
This is one of the most important decisions when using ECS. Here's a deep comparison:

Cost Comparison (Real Numbers)

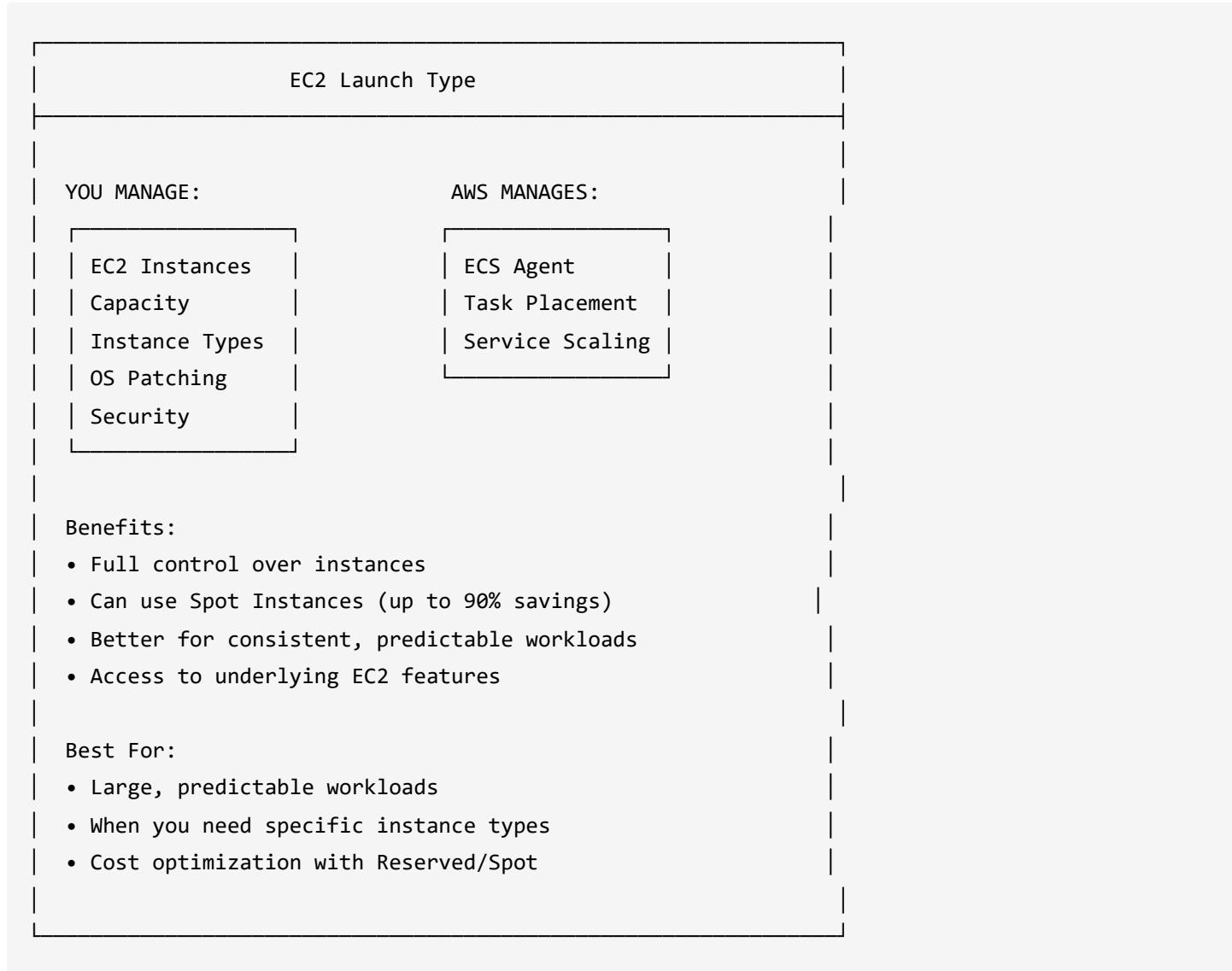
Workload	Fargate Cost/month	EC2 Cost/month	Winner
Variable load (0-100 tasks)	~\$50-200	~\$150 (always on)	Fargate

Workload	Fargate Cost/month	EC2 Cost/month	Winner
Steady 24/7 (10 tasks)	~\$300	~\$100 (Reserved)	EC2
Batch jobs (4 hrs/day)	~\$40	~\$100	Fargate
GPU workloads	Not available	~\$500	EC2

Fargate (Serverless)



EC2 Launch Type



Comparison Table

Feature	Fargate	EC2
Server Management	None (serverless)	You manage EC2
Pricing	Per task (CPU+Memory)	EC2 instance cost
Scaling	Automatic	Manual or Auto Scaling Group
Startup Time	~30-60 seconds	Depends on instance
Max Task Size	4 vCPU, 30 GB	Based on EC2 type
GPU Support	Limited	Full support
Spot Instances	Fargate Spot	EC2 Spot
Ideal For	Variable workloads	Steady workloads

When to Choose What?

DECISION FLOWCHART

START



Need GPU or specific hardware?

|
└— YES → Use EC2

|
└— NO



Predictable, consistent workload?

|
└— YES → Consider EC2 (cost effective)

|
└— NO



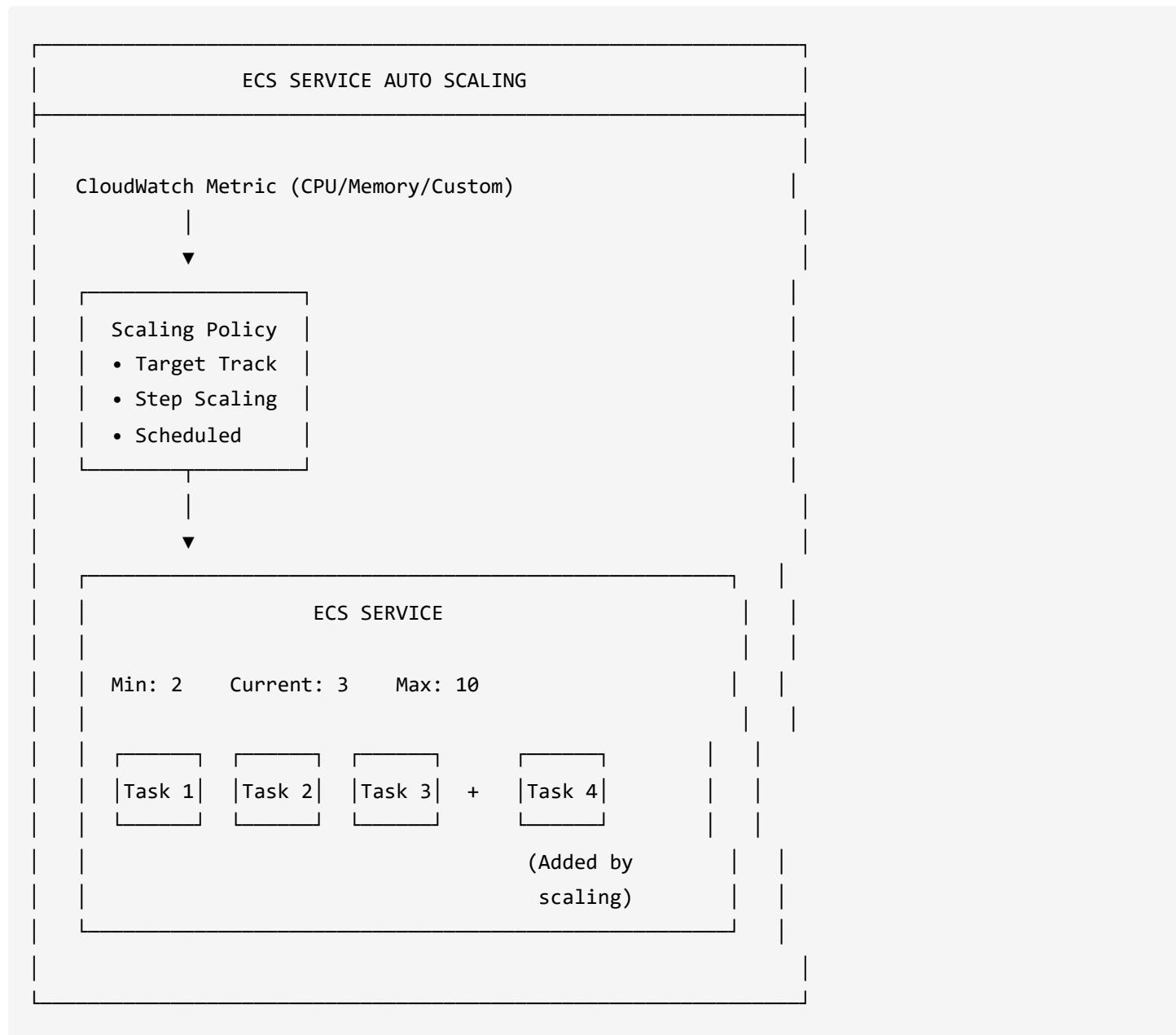
Want to minimize operational overhead?

|
└— YES → Use Fargate

|
└— NO → Use EC2 (more control)

9. ECS Service Auto Scaling

Auto Scaling Overview



Scaling Policy Types

Policy Type	Description	Use Case
Target Tracking	Maintains metric at target value	"Keep CPU at 70%"
Step Scaling	Add/remove based on alarm thresholds	Complex scaling rules
Scheduled Scaling	Time-based scaling	Known traffic patterns

Configure Auto Scaling (CLI)

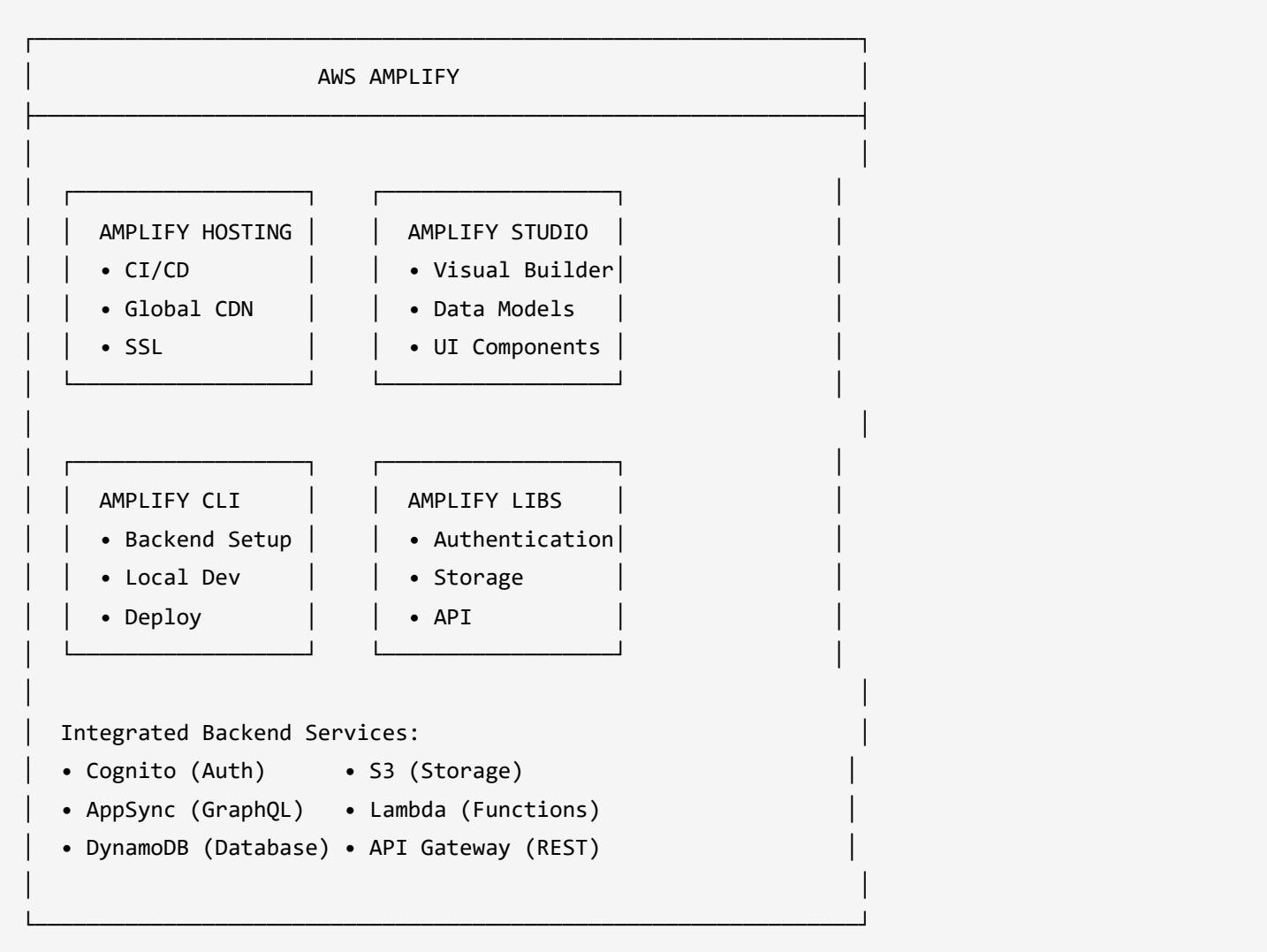
```
# Register scalable target
aws application-autoscaling register-scalable-target \
--service-namespace ecs \
--resource-id service/my-cluster/my-service \
--scalable-dimension ecs:service:DesiredCount \
--min-capacity 2 \
--max-capacity 10

# Create target tracking policy (CPU)
aws application-autoscaling put-scaling-policy \
--service-namespace ecs \
--resource-id service/my-cluster/my-service \
--scalable-dimension ecs:service:DesiredCount \
--policy-name cpu-target-tracking \
--policy-type TargetTrackingScaling \
--target-tracking-scaling-policy-configuration '{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "ECSServiceAverageCPUUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 120
}'
```

10. AWS Amplify

What is Amplify?

AWS Amplify is a set of tools and services for building full-stack web and mobile applications. It provides:



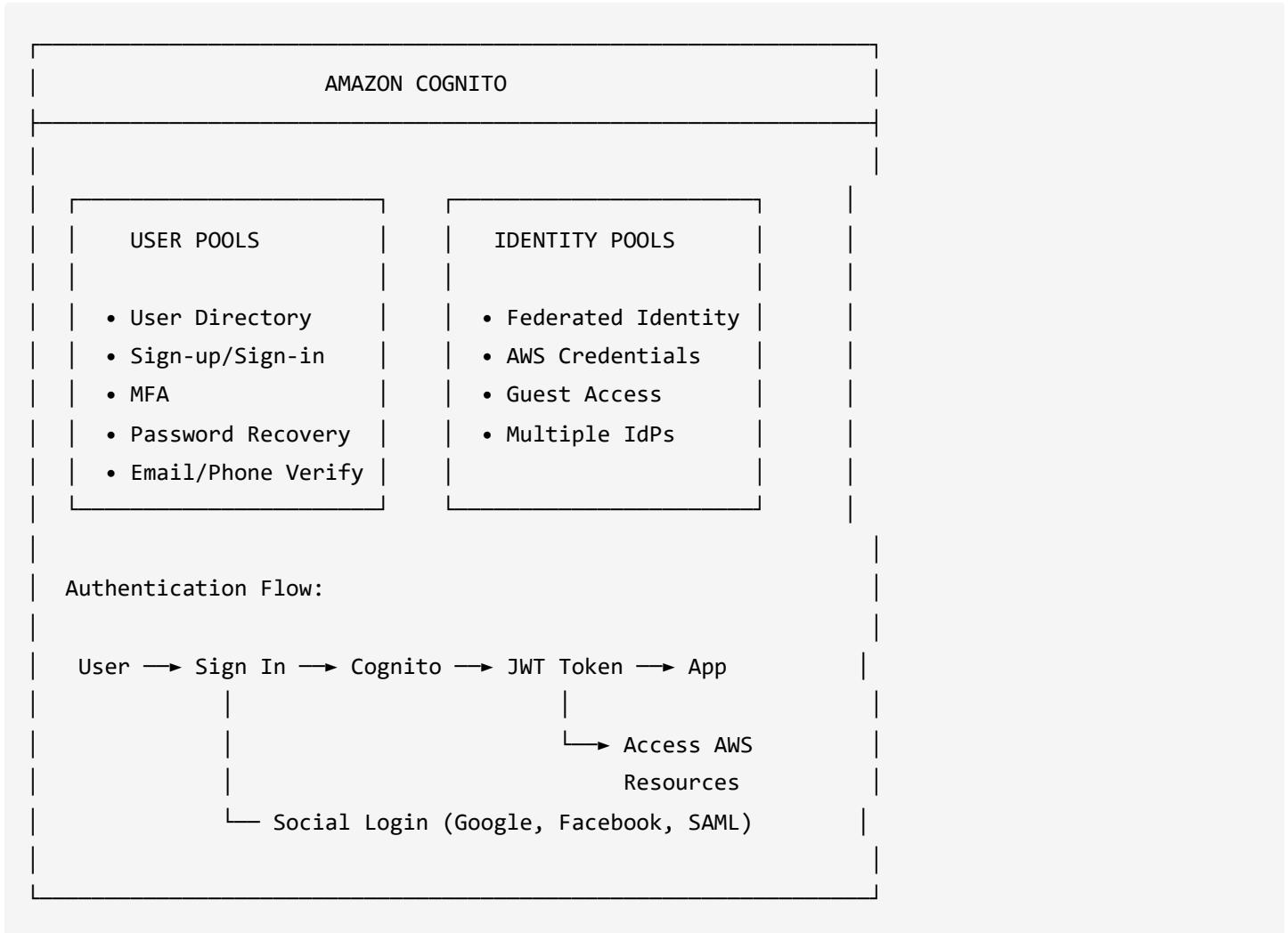
Amplify Features

Feature	Description
Hosting	Deploy web apps with CI/CD
Authentication	User sign-up, sign-in with Cognito
Data	Manage data with GraphQL/REST APIs
Storage	File storage with S3
Functions	Serverless functions with Lambda

11. Amazon Cognito

What is Cognito?

Amazon Cognito provides authentication, authorization, and user management for web and mobile apps.



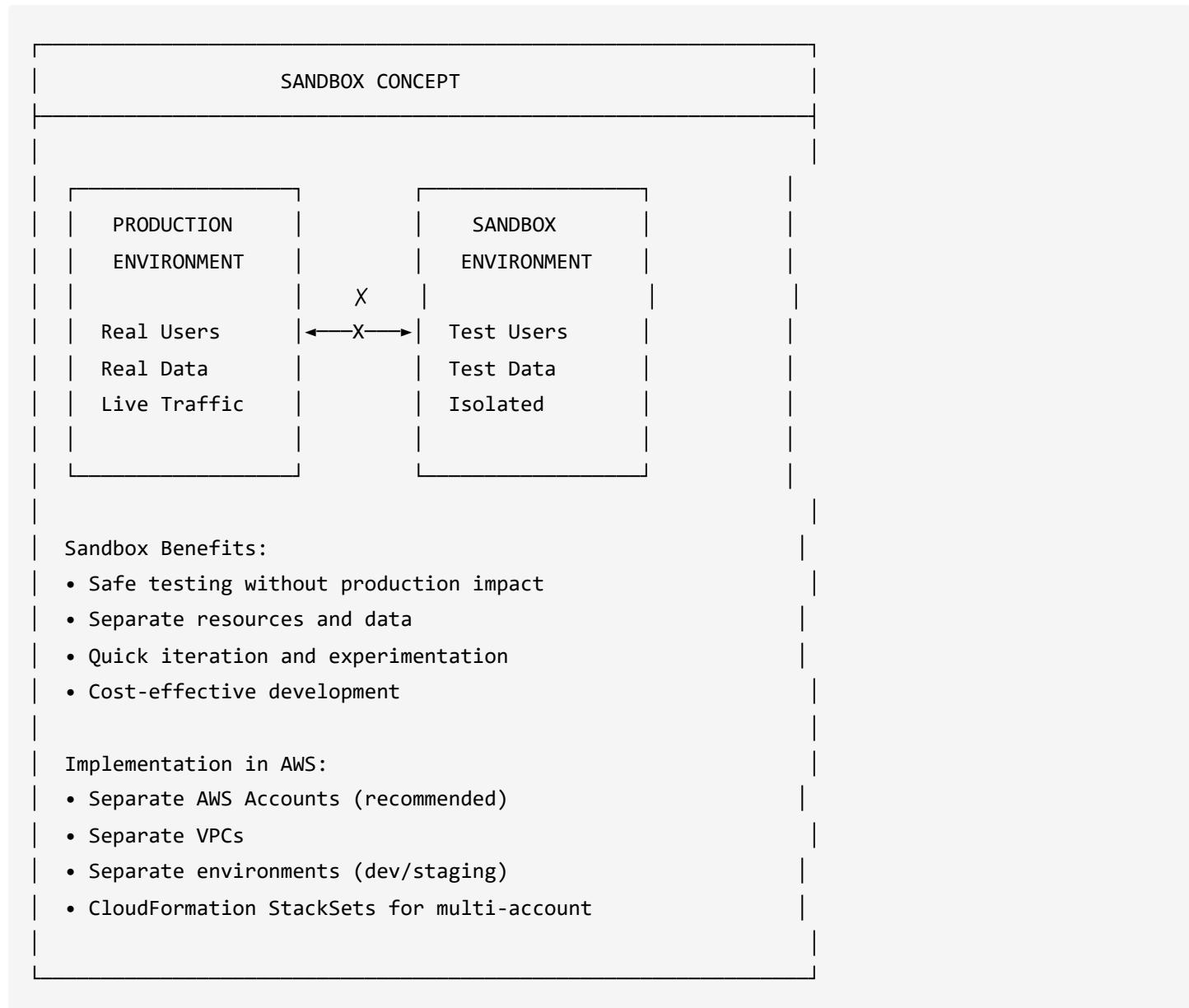
User Pool vs Identity Pool

Aspect	User Pool	Identity Pool
Purpose	User authentication	AWS resource access
Output	JWT tokens	Temporary AWS credentials
Features	Sign-up, MFA, recovery	Federated identities
Use With	API Gateway, ALB	Direct AWS service access

12. Sandbox Environment

What is a Sandbox?

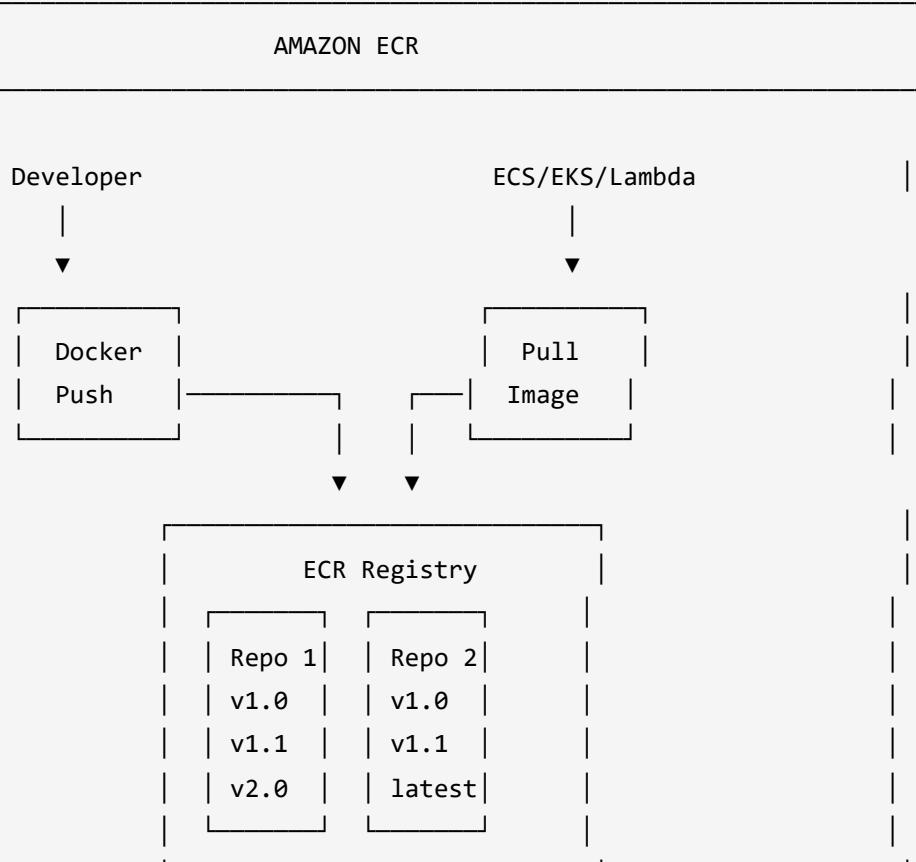
A **Sandbox** is an isolated testing environment that doesn't affect production resources.



13. ECR (Elastic Container Registry)

What is ECR?

Amazon ECR is a fully managed Docker container registry that makes it easy to store, manage, and deploy Docker container images.



ECR Commands

```
# Authenticate Docker to ECR
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 1234

# Create repository
aws ecr create-repository --repository-name my-app

# Build and tag image
docker build -t my-app .
docker tag my-app:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-app:latest

# Push image
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-app:latest

# List images
aws ecr list-images --repository-name my-app
```



14. Port Mapping & Networking

ECS Network Modes

ECS NETWORK MODES

1. awsvpc (Recommended for Fargate)

- Each task gets its own ENI
- Each task gets its own IP
- Full network isolation
- Required for Fargate

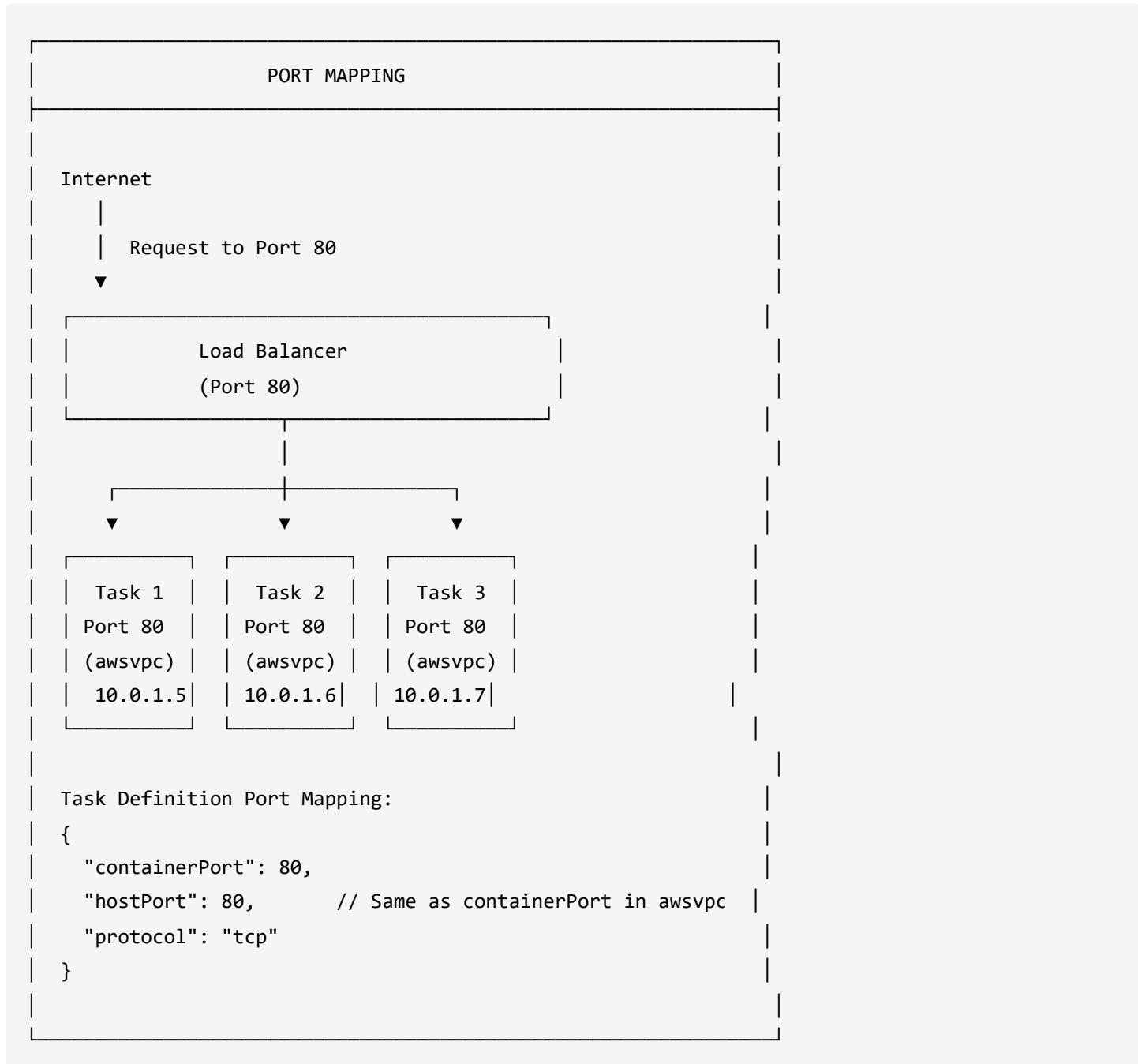
2. bridge (Default for EC2)

- Docker's built-in bridge network
- Port mapping host:container
- Multiple containers share host IP

3. host (EC2 only)

- Container uses host's network
- No port mapping needed
- Best performance

Port Mapping Example



15. Real AWS ECS Interview Questions (15 Questions)

Q1: Explain the difference between ECS and EKS. When would you choose one over the other?

Answer:

Aspect	ECS	EKS
Orchestrator	AWS proprietary	Kubernetes (CNCF standard)

Aspect	ECS	EKS
Complexity	Simpler, easier to learn	More complex, steeper learning curve
Portability	AWS-only	Multi-cloud, on-premises, hybrid
Learning Curve	1-2 weeks	1-3 months
Ecosystem	AWS-native	Huge K8s ecosystem (Helm, Istio, etc.)
Cost	No control plane fee	~\$73/month per cluster

When to Choose ECS:

- AWS-only infrastructure
- Smaller team without K8s expertise
- Simple microservices architecture
- Want faster time-to-market

When to Choose EKS:

- Multi-cloud or hybrid strategy
- Team already knows Kubernetes
- Need K8s ecosystem tools (Istio, Helm, etc.)
- Portability is critical requirement

Interview Tip: Mention that both can run the same containers - the difference is the orchestration layer.

Q2: What is the difference between Task Definition, Task, and Service in ECS?

Answer:

ANALOGY: Restaurant Kitchen

Task Definition = Recipe (ingredients, cooking instructions)

Task = Actual dish being cooked right now

Service = Kitchen manager ensuring X dishes are always ready

Detailed Comparison:

Component	Purpose	Example
Task Definition	Blueprint/template (JSON)	"Nginx container, 256 CPU, 512 MB memory, port 80"

Component	Purpose	Example
Task	Running instance of Task Definition	"Task abc123 running nginx on 10.0.1.5"
Service	Maintains desired number of tasks	"Keep 3 nginx tasks running, use ALB"

Key Points:

- Task Definition is immutable (create new revision to change)
- Task is ephemeral (can be stopped, replaced)
- Service provides self-healing and load balancing

Q3: How does Fargate pricing work and when should you use it vs EC2?

Answer:

Fargate Pricing:

- Per vCPU per second (~\$0.04/hour per vCPU)
- Per GB memory per second (~\$0.004/hour per GB)
- Minimum 1 minute billing

Example Calculation:

Task with 1 vCPU, 2 GB memory, running 24 hours:
 CPU: $1 \text{ vCPU} \times \$0.04/\text{hr} \times 24 \text{ hrs} = \0.96
 Memory: $2 \text{ GB} \times \$0.004/\text{hr} \times 24 \text{ hrs} = \0.19
 Total: ~\$1.15/day per task

Decision Matrix:

Scenario	Recommendation	Why
Variable/burst traffic	Fargate	Pay only when running
24/7 steady workload	EC2 with Reserved	Up to 70% cheaper
Short batch jobs	Fargate Spot	70% cheaper, interruption OK
GPU needed	EC2	Fargate doesn't support GPU
Small team, no ops	Fargate	Zero server management

Q4: Your ECS service keeps restarting tasks. How do you troubleshoot?

Answer:

Troubleshooting Checklist:

1. Check CloudWatch Logs

```
aws logs get-log-events --log-group-name /ecs/my-app --log-stream-name ecs/container/task-id
```

Look for: application errors, crash stacktraces, OOM kills

2. Check Task Stopped Reason

```
aws ecs describe-tasks --cluster my-cluster --tasks task-arn
```

Look at: `stoppedReason` field

3. Common Causes:

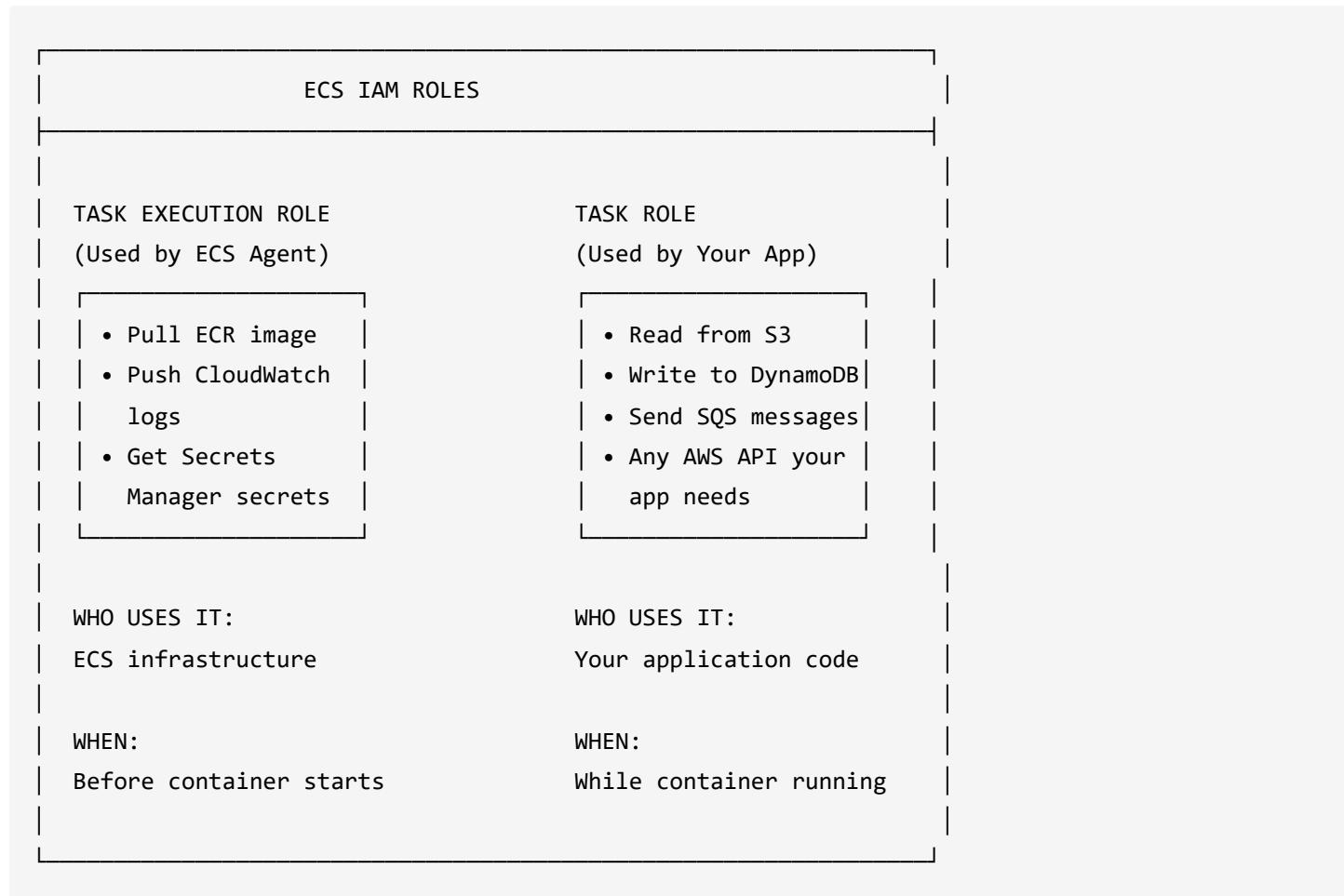
Symptom	Likely Cause	Fix
"Essential container exited"	App crash or health check fail	Check logs, fix code
"OutOfMemoryError"	Insufficient memory	Increase memory in Task Definition
"CannotPullContainerError"	ECR auth or image not found	Check IAM role, image URI
"ResourceInitializationError"	Network or ENI issue	Check VPC, subnet, SG
Repeated restarts at same point	Health check timing	Increase <code>startPeriod</code>

4. Check Health Checks:

- ALB health check path correct?
- Container health check command works?
- Start period long enough?

Q5: Explain Task Execution Role vs Task Role. Why do we need both?

Answer:



Why Two Roles (Security Principle: Least Privilege):

- Task Execution Role: ECS needs it to start your container
- Task Role: Your application only gets permissions it needs
- Separating them prevents over-privileged containers

Example:

- Your app needs to read S3 → Task Role needs s3:GetObject
- Your app does NOT need to pull images → Don't give ecr:GetDownloadUrlForLayer to Task Role

Q6: How do you implement zero-downtime deployment in ECS?

Answer:

Method 1: Rolling Update (Default)

ECS Service Settings:

- minimumHealthyPercent: 100
- maximumPercent: 200

Timeline:

1. Service has 4 tasks (v1)
2. Deploy new Task Definition (v2)
3. ECS starts 4 new tasks (v2) → now 8 total
4. ALB health checks pass for v2
5. ECS drains and stops v1 tasks
6. Final: 4 tasks (v2)

Method 2: Blue/Green with CodeDeploy

1. Blue (v1) running, serving traffic
2. Deploy Green (v2) behind separate target group
3. Test Green independently
4. CodeDeploy shifts traffic: Blue → Green
5. Options: All-at-once, Linear (10% every min), Canary (10%, then 90%)
6. Automatic rollback if health checks fail

Method 3: Canary with ALB Weighted Routing

```
{
  "Type": "forward",
  "ForwardConfig": {
    "TargetGroups": [
      {"TargetGroupArn": "v1-tg", "Weight": 90},
      {"TargetGroupArn": "v2-tg", "Weight": 10}
    ]
  }
}
```

Key Settings for Zero-Downtime:

- ALB deregistration delay: 30-60 seconds
- Container health check with proper startPeriod
- minimumHealthyPercent: 100 (don't go below current capacity)

Q7: What is awsvpc network mode and why is it important?**Answer:****awsvpc Mode:**

- Each task gets its own Elastic Network Interface (ENI)
- Each task gets its own private IP address
- **Required for Fargate** (only network mode supported)
- Recommended for EC2 launch type too

awsvpc Mode:

```
Each Task = Own ENI = Own IP = Own Security Group
Task 1: 10.0.1.5 (SG: allow 80)
Task 2: 10.0.1.6 (SG: allow 80)
Task 3: 10.0.1.7 (SG: allow 80)
```

bridge Mode (EC2 only):

```
All Tasks Share Host Network
EC2 Instance: 10.0.1.5
  └─ Container 1: hostPort 8080 → containerPort 80
  └─ Container 2: hostPort 8081 → containerPort 80
```

Why awsvpc is Important:

1. **Security:** Apply SG at task level (not instance level)
2. **Simplicity:** No port conflicts (each task has own IP)
3. **Visibility:** Each task visible in VPC Flow Logs
4. **Service Discovery:** Tasks register with their own IP

Limitation: Limited by ENIs per EC2 instance (varies by type)

Q8: How do you pass secrets to ECS containers securely?

Answer:

✗ Bad Practice:

```
"environment": [
  {"name": "DB_PASSWORD", "value": "mysecretpassword123"}
]
```

Problem: Secrets visible in Task Definition, console, logs

Best Practice: Use Secrets Manager or Parameter Store

```
"secrets": [
  {
    "name": "DB_PASSWORD",
    "valueFrom": "arn:aws:secretsmanager:us-east-1:123456789:secret:prod/db-password"
  },
  {
    "name": "API_KEY",
    "valueFrom": "arn:aws:ssm:us-east-1:123456789:parameter/prod/api-key"
  }
]
```

Required IAM (Task Execution Role):

```
{
  "Effect": "Allow",
  "Action": [
    "secretsmanager:GetSecretValue",
    "ssm:GetParameters"
  ],
  "Resource": [
    "arn:aws:secretsmanager:us-east-1:123456789:secret:prod/*",
    "arn:aws:ssm:us-east-1:123456789:parameter/prod/*"
  ]
}
```

Benefits:

- Secrets not stored in Task Definition
- Rotation support (Secrets Manager)
- Audit trail in CloudTrail
- Fine-grained access control

Q9: Explain ECS Service Auto Scaling. What metrics can you scale on?

Answer:

Scaling Target Metrics:

Metric	Use Case	Example
ECSServiceAverageCPUUtilization	CPU-bound workloads	"Scale when CPU > 70%"
ECSServiceAverageMemoryUtilization	Memory-bound apps	"Scale when Memory > 80%"
ALBRequestCountPerTarget	Request-based	"Scale when > 1000 req/target"
SQSApproximateNumberOfMessagesVisible	Queue-based	"Scale when queue > 100"
Custom CloudWatch Metric	Custom logic	"Scale when error rate > 5%"

Scaling Policy Types:

1. Target Tracking (Recommended):

```
# Keep CPU at 70%
aws application-autoscaling put-scaling-policy \
--policy-type TargetTrackingScaling \
--target-tracking-scaling-policy-configuration \
'TargetValue=70.0,PredefinedMetricSpecification={PredefinedMetricType=ECSServiceAverageCP'
```

2. Step Scaling:

CPU 70-80% → Add 1 task
 CPU 80-90% → Add 2 tasks
 CPU > 90% → Add 4 tasks

3. Scheduled Scaling:

Every weekday 9 AM → Scale to 10 tasks
 Every weekday 6 PM → Scale to 2 tasks

Q10: What happens when you update a Service with a new Task Definition?

Answer:

Rolling Update Process:

Initial State: 4 tasks running v1

Step 1: ECS starts new v2 tasks (based on maximumPercent)

Running: 4 v1 + 2 v2 = 6 tasks

Step 2: v2 tasks pass health checks, registered with ALB

Step 3: ECS begins draining v1 tasks

- ALB deregistration delay (default 300s)
- Connections drain gracefully

Step 4: v1 tasks stopped, more v2 started

Running: 2 v1 + 4 v2 = 6 tasks

Step 5: Continue until all v1 replaced

Final: 0 v1 + 4 v2 = 4 tasks

Key Parameters:

- `minimumHealthyPercent` : 100 (don't go below 4 healthy)
- `maximumPercent` : 200 (can run up to 8 during deployment)

If Deployment Fails:

- Tasks keep failing health checks
- Circuit breaker triggers (if enabled)
- Rollback to previous Task Definition

Q11: How do you debug a container that won't start in ECS?

Answer:

Systematic Debugging Steps:

Step 1: Check Task Stopped Reason

```
aws ecs describe-tasks --cluster my-cluster --tasks <task-arn> \
--query 'tasks[0].stoppedReason'
```

Step 2: Common Errors and Fixes:

Error	Cause	Fix
CannotPullContainerError	Image not found or auth failed	Check image URI, ECR permissions
ResourceInitializationError	ENI allocation failed	Check subnet has available IPs
OutOfMemoryError: Container killed	Memory limit exceeded	Increase memory in Task Definition
Essential container exited	App crashed on startup	Check CloudWatch logs for app error
CannotCreateContainerError	Invalid Task Definition	Validate JSON, check port conflicts

Step 3: Check CloudWatch Logs

```
aws logs tail /ecs/my-app --follow
```

Step 4: Run Container Locally

```
# Replicate ECS environment locally
docker run -e AWS_REGION=us-east-1 \
-p 80:80 \
123456789.dkr.ecr.us-east-1.amazonaws.com/my-app:latest
```

Step 5: ECS Exec (SSH into running container)

```
aws ecs execute-command \
--cluster my-cluster \
--task <task-id> \
--container my-container \
--interactive \
--command "/bin/sh"
```

Q12: What is the difference between Fargate and Fargate Spot?

Answer:

Aspect	Fargate	Fargate Spot
Pricing	On-demand	Up to 70% cheaper
Availability	Guaranteed	Can be interrupted
Termination	You control	AWS can terminate with 2-min warning
Use Case	Critical workloads	Fault-tolerant, batch jobs
SLA	Yes	No

When to Use Fargate Spot:

- Development/test environments
- Batch processing jobs
- Stateless web apps with auto-scaling
- Workloads that can handle interruptions

Best Practice: Mix Both

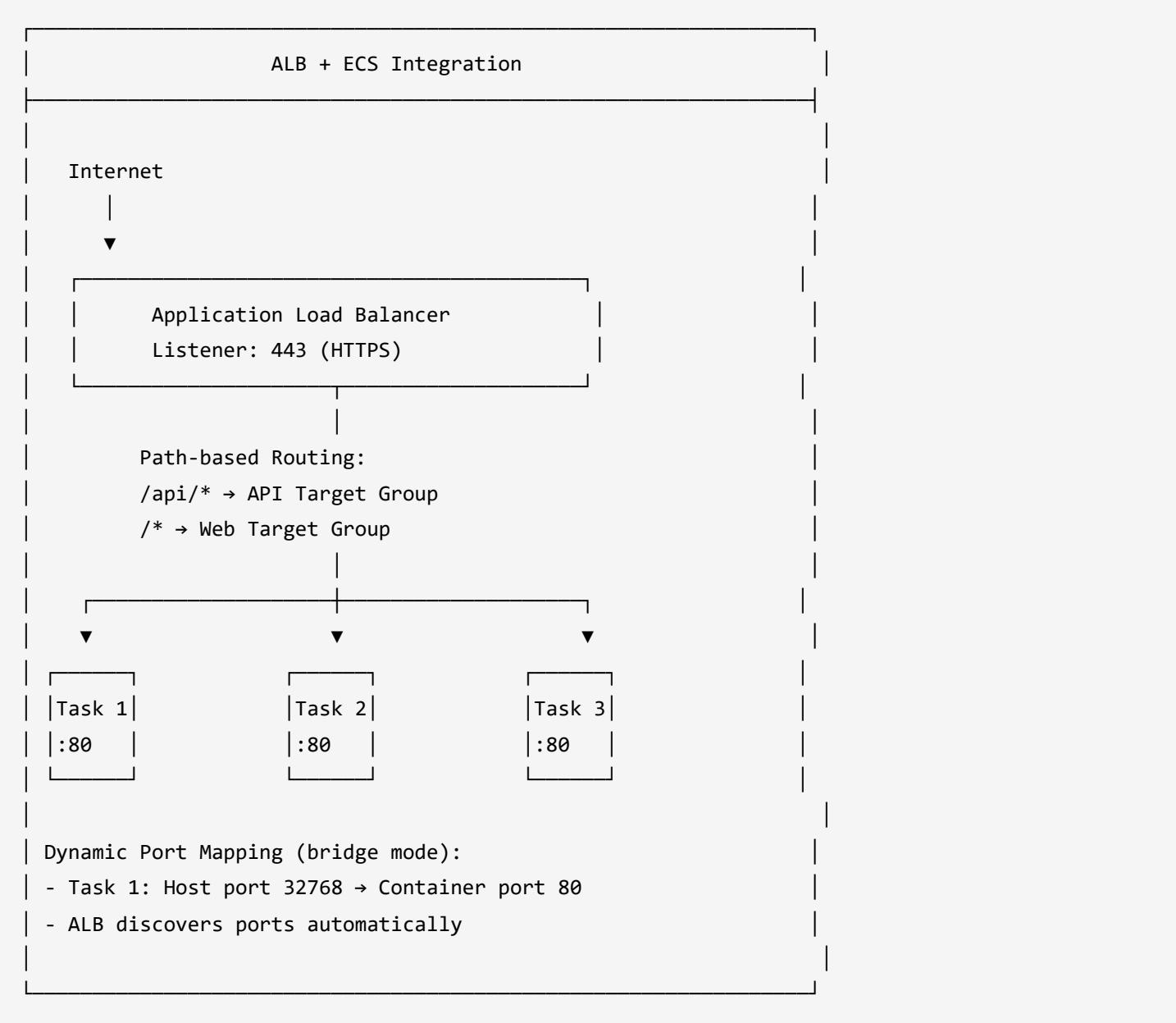
```
aws ecs create-service \
--capacity-provider-strategy \
  "capacityProvider=FARGATE,weight=1,base=2" \
  "capacityProvider=FARGATE_SPOT,weight=3"
```

This means: Always keep 2 tasks on regular Fargate, scale with 75% Spot.

Q13: How does ECS integrate with Application Load Balancer?

Answer:

Architecture:



Key Integration Points:

- Target Group:** ECS registers tasks automatically
- Health Checks:** ALB checks /health endpoint
- Draining:** ALB drains connections before task stops
- awsvpc:** Each task has unique IP, port 80

Service Definition:

```

"loadBalancers": [
    "targetGroupArn": "arn:aws:elasticloadbalancing:...",
    "containerName": "web",
    "containerPort": 80
}]

```

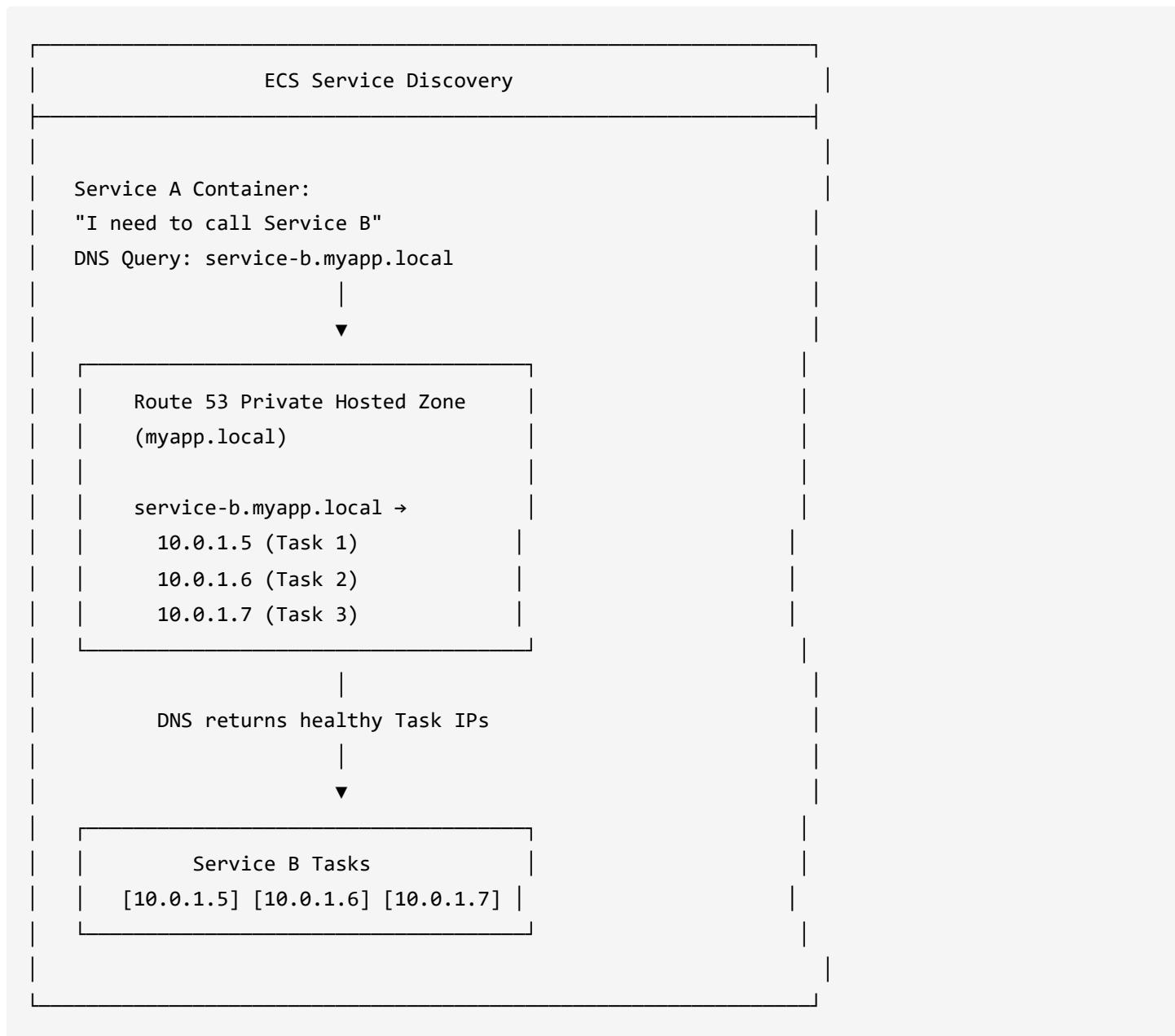
Q14: What is ECS Service Discovery and how does it work?

Answer:

Problem It Solves:

Service A needs to call Service B, but Task IPs change constantly.

Solution: AWS Cloud Map + Route 53



Setup:

1. Create Cloud Map namespace (myapp.local)
2. Create Cloud Map service (service-b)
3. Enable Service Discovery in ECS Service
4. Tasks auto-register/deregister

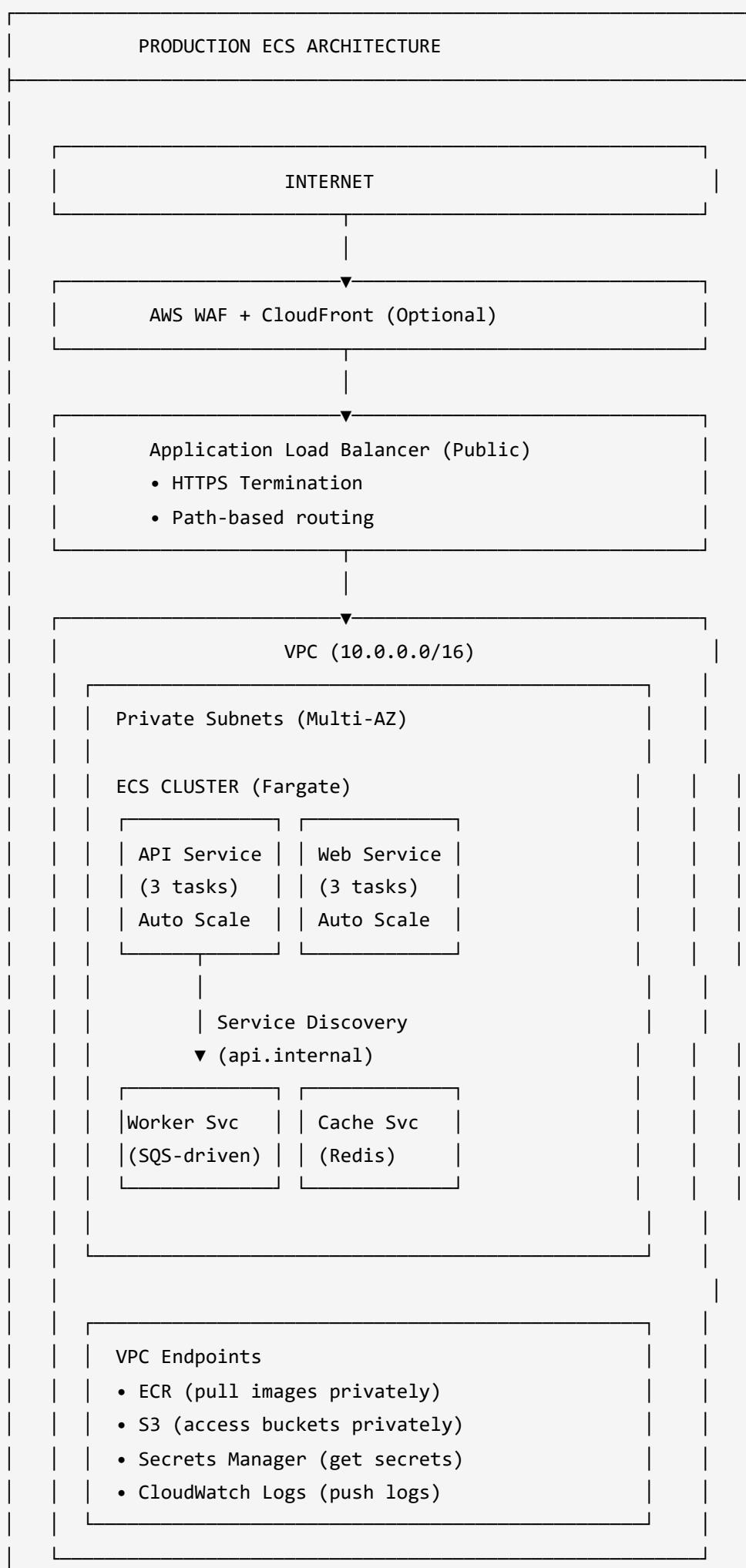
Benefits:

- No need for load balancer for internal services

- Automatic health-aware DNS
 - Works with ECS, EKS, EC2, on-premises
-

Q15: Design a production-ready ECS architecture for a microservices application

Answer:



Supporting Services:

- ECR: Container images
- Secrets Manager: Database passwords, API keys
- CloudWatch: Logs, metrics, alarms
- CodePipeline: CI/CD with Blue/Green deployment
- RDS (Multi-AZ): Database
- ElastiCache: Redis for caching

Key Design Decisions:

1. **Fargate**: No EC2 management, pay per task
 2. **Multi-AZ**: High availability
 3. **Private Subnets**: No public IPs on containers
 4. **VPC Endpoints**: Private access to AWS services
 5. **Service Discovery**: Internal service-to-service communication
 6. **Auto Scaling**: Handle traffic spikes
 7. **Blue/Green**: Zero-downtime deployments
-

Quick Reference

ECS CLI Commands Cheat Sheet

```
# Cluster Management
aws ecs create-cluster --cluster-name NAME
aws ecs list-clusters
aws ecs delete-cluster --cluster NAME

# Task Definitions
aws ecs register-task-definition --cli-input-json file://task-def.json
aws ecs list-task-definitions
aws ecs deregister-task-definition --task-definition NAME:REVISION

# Tasks
aws ecs run-task --cluster NAME --task-definition NAME
aws ecs list-tasks --cluster NAME
aws ecs stop-task --cluster NAME --task TASK_ARN

# Services
aws ecs create-service --cluster NAME --service-name NAME --task-definition NAME --desired-count N
aws ecs update-service --cluster NAME --service NAME --desired-count N
aws ecs delete-service --cluster NAME --service NAME --force

# ECR
aws ecr get-login-password | docker login --username AWS --password-stdin REGISTRY
aws ecr create-repository --repository-name NAME
aws ecr list-images --repository-name NAME
```



Important Limits

Resource	Limit
Clusters per region	10,000
Services per cluster	5,000
Tasks per service	5,000
Containers per task	10
Max task CPU (Fargate)	4 vCPU
Max task memory (Fargate)	30 GB

Notes 10: DNS Server Configuration & Domain Name System

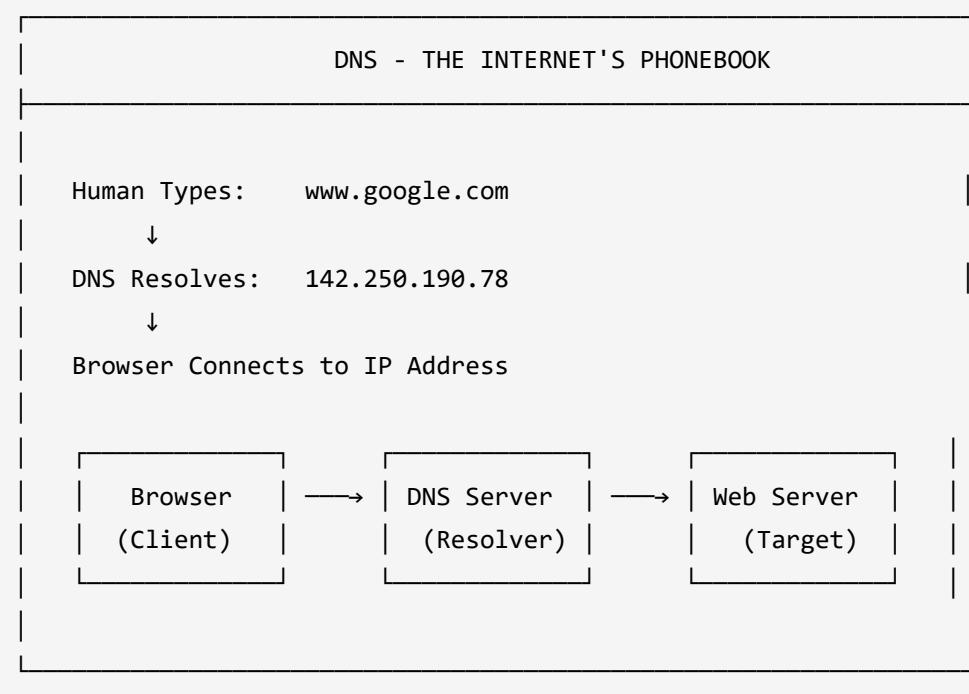
Table of Contents

1. [DNS Overview](#)
 2. [What is DNS \(Domain Name System\)](#)
 3. [How DNS Works](#)
 4. [DNS Server Components](#)
 5. [Setting Up Your Own DNS Server](#)
 6. [Installing BIND9 DNS Server](#)
 7. [DNS Configuration Files](#)
 8. [Zone Files](#)
 9. [Forward Zone Configuration](#)
 10. [Reverse Zone Configuration](#)
 11. [DNS Records Types](#)
 12. [Testing DNS Configuration](#)
 13. [Web Server Setup with Apache](#)
 14. [Firewall Configuration](#)
 15. [Terraform Cloud Introduction](#)
 16. [Important Q&A](#)
-

1. DNS Overview

What is DNS?

DNS (Domain Name System) translates human-friendly domain names into machine-readable IP addresses.



Real-World Analogy: DNS as a Global Postal System

Understanding DNS is easier when you compare it to how mail delivery works:

DNS AS A POSTAL/DELIVERY SYSTEM

POSTAL SYSTEM

House Address
"123 Main St, New York"
(Exact location coordinates)

Person's Name
"John Smith"
(Human-friendly identifier)

Phone Book/Directory
(Name → Address mapping)

Local Post Office
(Helps find addresses)

Central Registry
(Master address database)

ZIP Code Regions
(Groups addresses by area)

Neighborhood Registry
(Local address authority)

Forwarding Address
(Redirects to new location)

Address Update Time
(How often to check for moves)

DNS SYSTEM

IP Address
"142.250.190.78"
(Exact server location)

Domain Name
"google.com"
(Human-friendly identifier)

DNS Server
(Domain → IP mapping)

Recursive Resolver
(Finds IP addresses)

Root DNS Servers
(Master internet directory)

TLD Servers (.com,.org)
(Groups domains by type)

Authoritative DNS
(Domain's own DNS server)

CNAME Record
(Alias pointing elsewhere)

TTL (Time To Live)
(How long to cache records)

How the Analogy Works - Sending a Letter

MAIL DELIVERY vs DNS RESOLUTION

SENDING A LETTER TO "JOHN SMITH"

You: "I want to send letter to John Smith"

↓

Local Post Office: "Let me find his address..."

↓

Checks if already known (Cache)

↓ (Not found)

Contacts Regional Directory: "Which city is Smith in?"

↓

Regional says: "Try New York office"

↓

New York office: "John Smith lives at 123 Main St"

↓

Letter delivered to 123 Main St!

VISITING www.google.com

Browser: "I want to connect to www.google.com"

↓

DNS Resolver: "Let me find the IP address..."

↓

Checks if already known (DNS Cache)

↓ (Not found)

Contacts Root Server: "Who handles .com domains?"

↓

Root says: "Try .com TLD server"

↓

TLD Server: "google.com's DNS is ns1.google.com"

↓

Authoritative DNS: "www.google.com = 142.250.190.78"

↓

Browser connects to 142.250.190.78!

Why This Analogy Matters for Interviews

Postal Concept	DNS Equivalent	Interview Insight
Person moves, keeps name	Server IP changes, domain stays same	DNS provides abstraction layer
Multiple people at same address	Multiple domains on same IP (Virtual Hosting)	Shared hosting works via Host header
Mail forwarding	CNAME records	Aliases point to canonical names
Priority mail	MX record priority	Lower number = higher priority
Return address verification	Reverse DNS (PTR)	Used for email authentication
Address change propagation time	TTL-based DNS propagation	Low TTL = faster changes, more queries

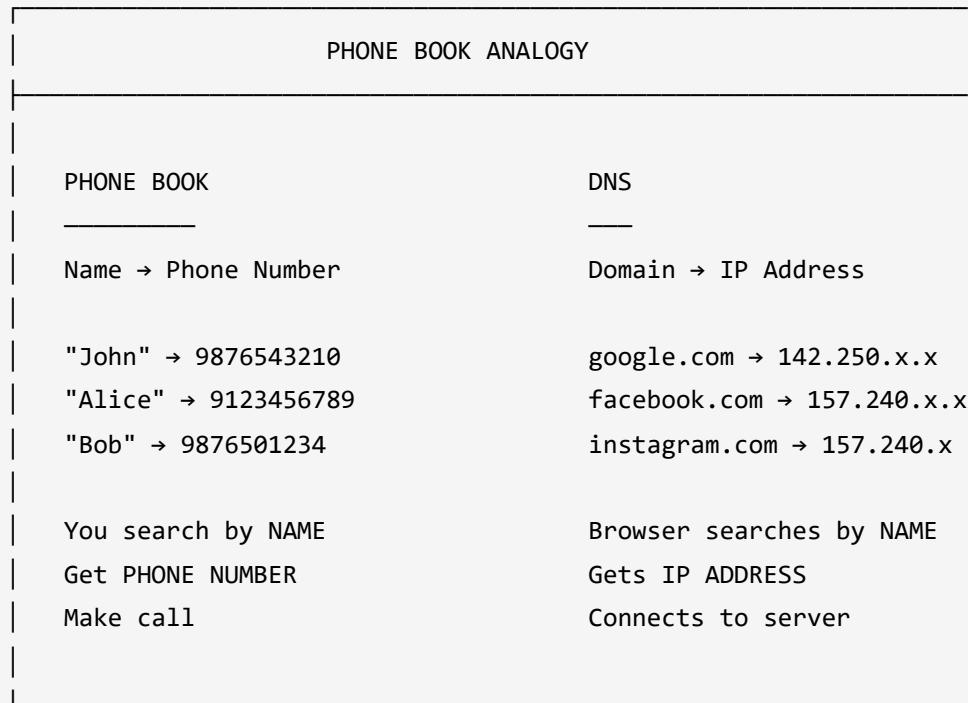
Why DNS is Important?

- **Human Memory:** Easier to remember `google.com` than `142.250.190.78`
- **Abstraction:** IP addresses can change, domain names stay same
- **Load Balancing:** Single domain can point to multiple IPs
- **Geographic Distribution:** Route users to nearest server
- **Redundancy:** Multiple DNS servers prevent single points of failure
- **Security:** DNS filtering can block malicious websites

2. What is DNS (Domain Name System)

Simple Definition

DNS is like a phonebook for the internet - you search by name, it gives you the number (IP address).

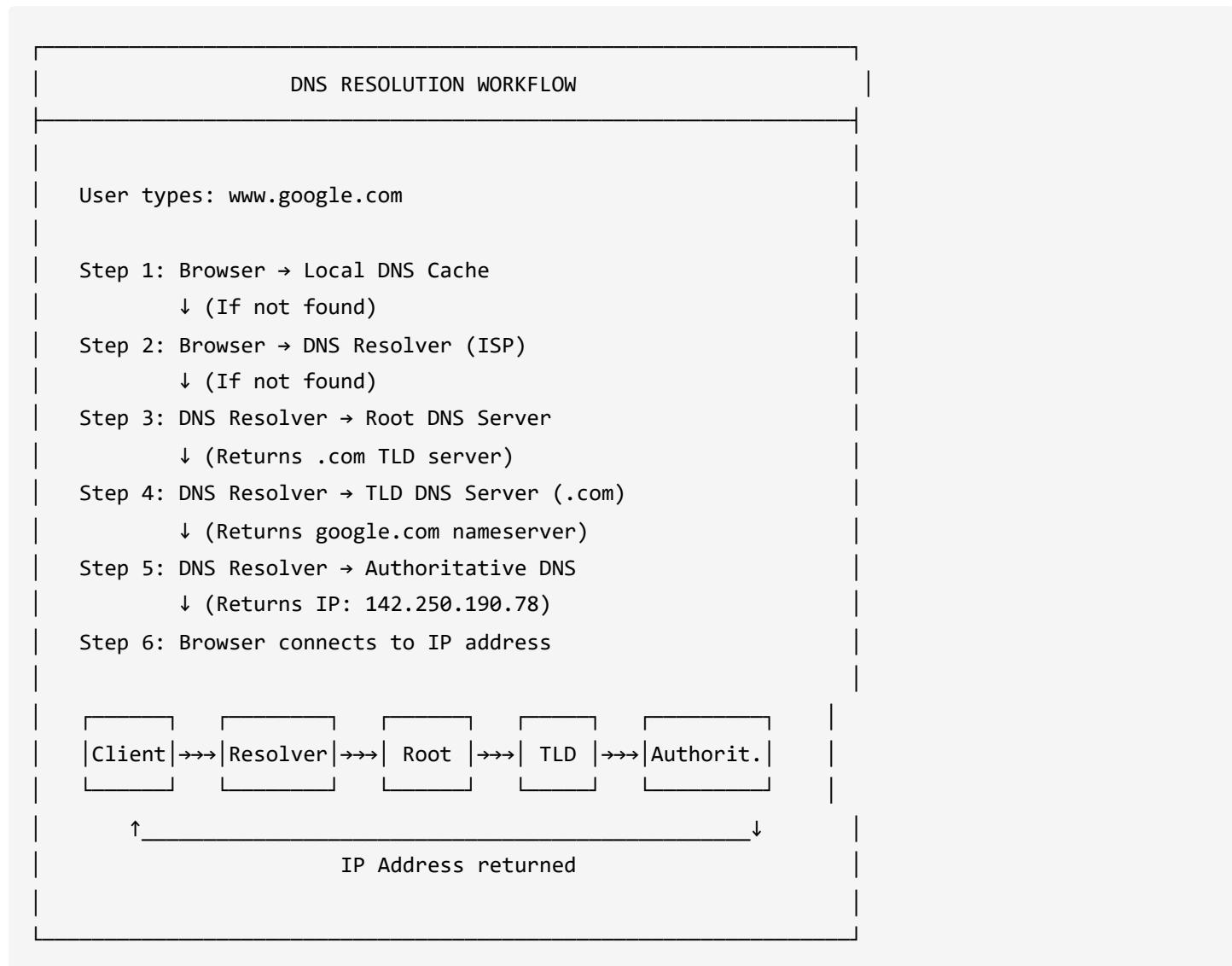


Domain Name Examples

Domain Name	Description
www.google.com	Google Search Engine
www.facebook.com	Facebook Social Network
www.instagram.com	Instagram Photo Sharing
myweb.com	Custom Domain Example

3. How DNS Works

DNS Resolution Process

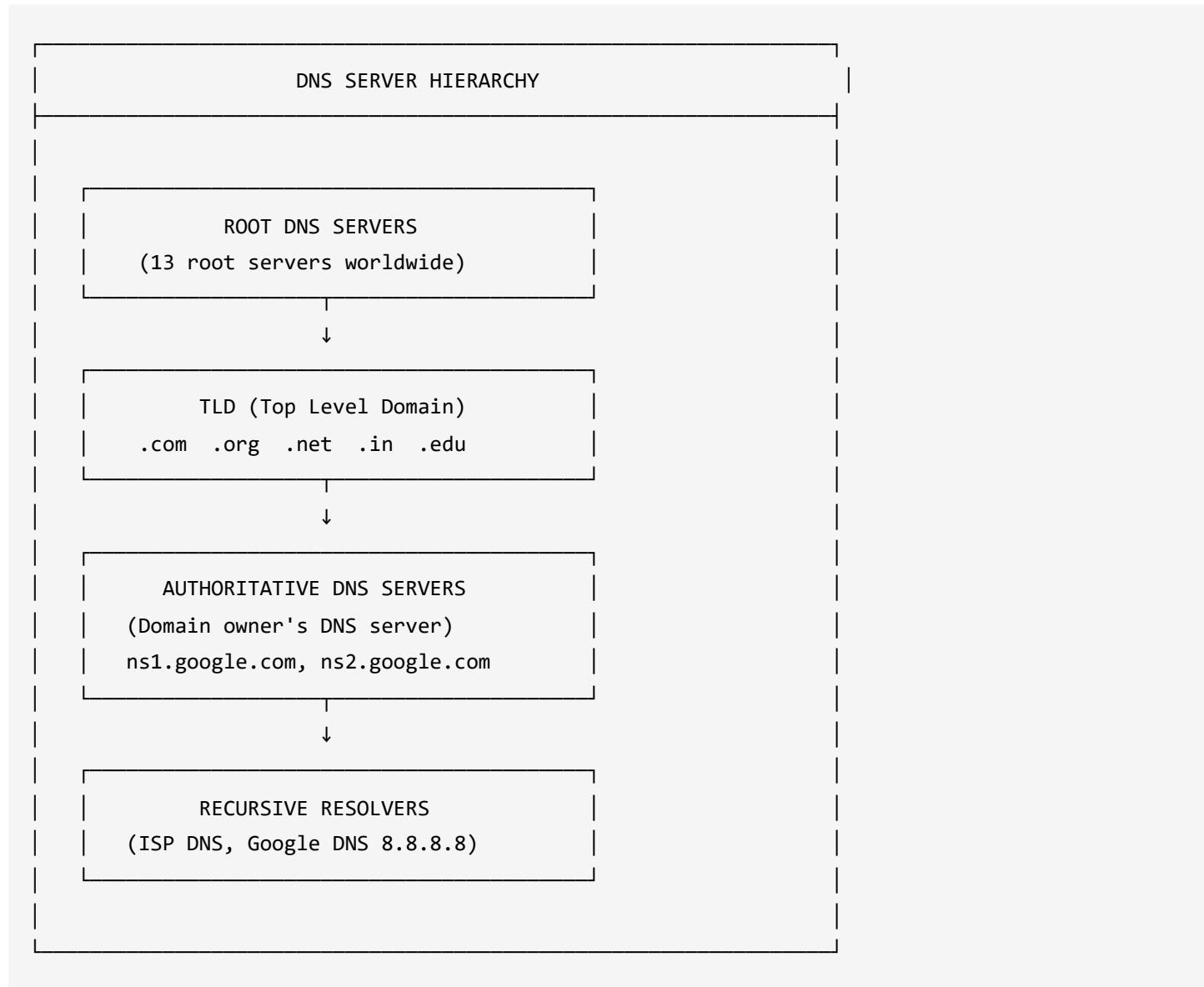


Key Concepts

- **Server Address:** Every server has an IP address (e.g., 10.21.55.1)
- **Domain Mapping:** DNS maps domain names to IP addresses
- **DNS Server:** Contains the mapping database
- **Resolution:** Process of converting domain name to IP

4. DNS Server Components

DNS Server Types



Component Comparison

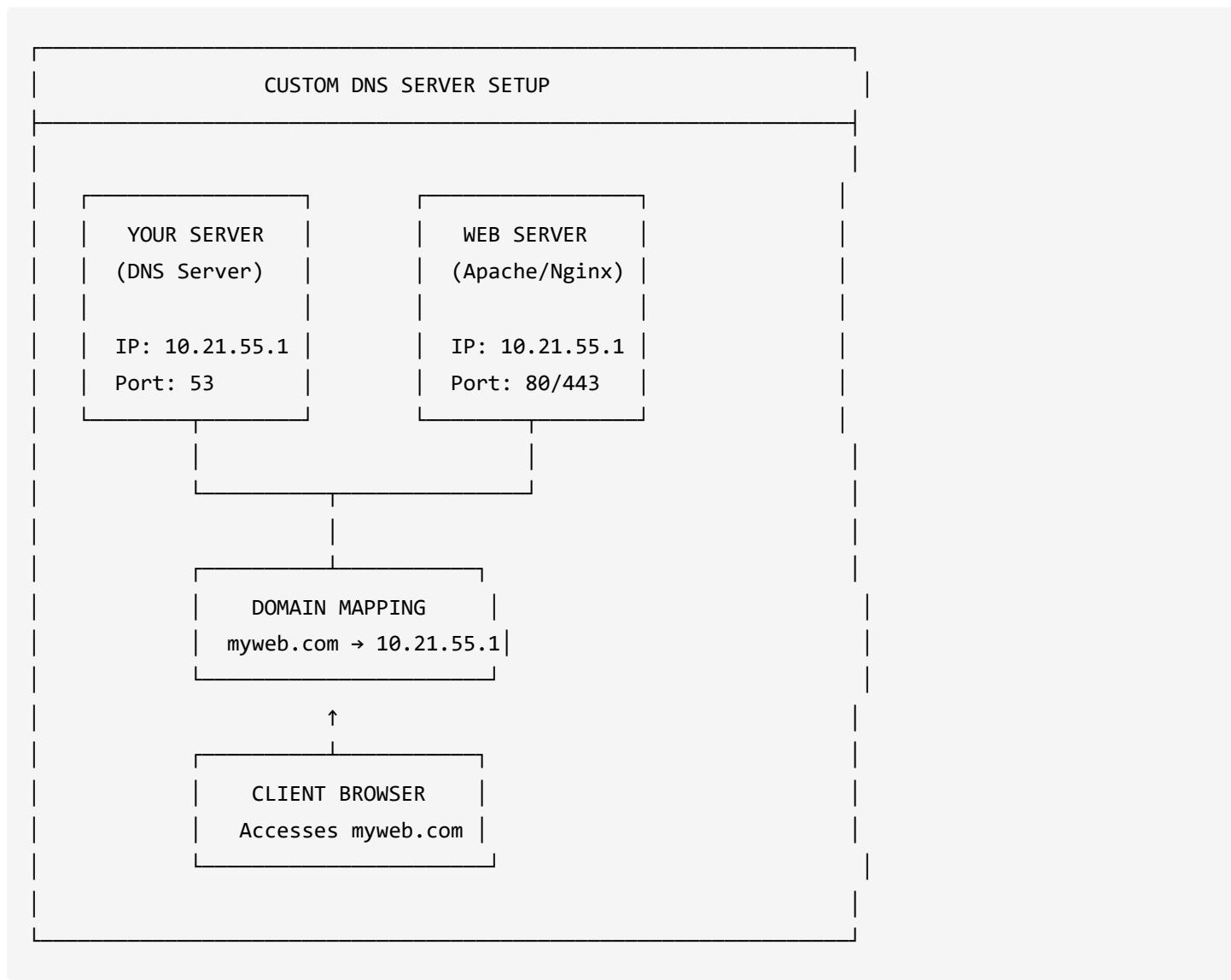
Component	Function	Example
Root Server	Top of DNS hierarchy	a.root-servers.net
TLD Server	Manages top-level domains	.com, .org, .net
Authoritative Server	Contains actual DNS records	ns1.google.com
Recursive Resolver	Queries other servers on behalf of client	8.8.8.8 (Google)
Local DNS Cache	Stores recent lookups	Browser/OS cache

5. Setting Up Your Own DNS Server

Prerequisites

1. Virtual Machine (Linux recommended)
2. Root/sudo privileges
3. Static IP address
4. Open firewall ports (53 TCP/UDP)

Architecture for Custom DNS



6. Installing BIND9 DNS Server

What is BIND9?

BIND (Berkeley Internet Name Domain) is the most widely used DNS software on the internet.

Installation Steps

```
# Step 1: Update system packages
sudo apt update

# Step 2: Install BIND9 and utilities
sudo apt install bind9 -y

# Step 3: Verify installation
rpm -qa | grep bind
# OR
apt list --installed | grep bind

# Step 4: Check BIND9 service status
systemctl status named
# OR
systemctl status bind9

# Step 5: Start BIND9 service
sudo systemctl start named
# OR
sudo systemctl start bind9

# Step 6: Enable BIND9 to start on boot
sudo systemctl enable named
```

Service Management Commands

Command	Description
systemctl start named	Start DNS service
systemctl stop named	Stop DNS service
systemctl restart named	Restart DNS service
systemctl status named	Check service status
systemctl enable named	Enable auto-start on boot

7. DNS Configuration Files

Main Configuration Files Location

BIND9 CONFIGURATION FILES

```
/etc/named.conf           Main configuration
|
|   └── /etc/named/
|       ├── named.ca        Zone files directory
|       ├── named.localhost Root hints
|       └── named.loopback  Localhost zone
|
|   └── /var/named/         Loopback zone
|       ├── forward.zone   Zone data files
|       └── reverse.zone   Forward lookup zone
|                           Reverse lookup zone
```

Alternative Locations (Ubuntu):

/etc/bind/	Bind config directory
/etc/bind/named.conf	Main config
/etc/bind/named.conf.local	Local zone definitions
/etc/bind/named.conf.options	DNS options

Main Configuration File (named.conf)

```
// /etc/named.conf - Main DNS Configuration

options {
    listen-on port 53 { 127.0.0.1; 10.21.55.1; };
    directory "/var/named";
    allow-query { any; };                                // Allow queries from any IP
    recursion yes;                                     // Enable recursive queries
};

// Forward Zone Definition
zone "myweb.com" IN {
    type master;
    file "forward.myweb.com.zone";
};

// Reverse Zone Definition
zone "55.21.10.in-addr.arpa" IN {
    type master;
    file "reverse.myweb.com.zone";
};
```

Key Configuration Parameters

Parameter	Description	Example Value
listen-on	IP addresses to listen on	port 53 { 127.0.0.1; }
directory	Location of zone files	/var/named
allow-query	Who can query this server	{ any; } or { localhost; }
recursion	Allow recursive lookups	yes / no
forwarders	Forward queries to another DNS	{ 8.8.8.8; 8.8.4.4; }

8. Zone Files

Understanding Zone Files

ZONE FILE TYPES

FORWARD ZONE

Domain Name → IP Address

myweb.com → 10.21.55.1

Used for: Normal DNS lookups

File: forward.zone or forward.myweb.com.zone

REVERSE ZONE

IP Address → Domain Name

10.21.55.1 → myweb.com

Used for: Reverse DNS lookups (PTR records)

File: reverse.zone or reverse.myweb.com.zone

9. Forward Zone Configuration

Creating Forward Zone File

```
# Location: /var/named/forward.myweb.com.zone

$TTL 86400
@ IN SOA ns1.myweb.com. admin.myweb.com. (
    2024010801 ; Serial Number (YYYYMMDDNN)
    3600        ; Refresh (1 hour)
    1800        ; Retry (30 minutes)
    604800      ; Expire (1 week)
    86400       ; Minimum TTL (1 day)
)

; Name Server Records
@ IN NS ns1.myweb.com.

; A Records (Address Records)
@ IN A 10.21.55.1
ns1 IN A 10.21.55.1
www IN A 10.21.55.1

; CNAME Records (Aliases)
mail IN CNAME www.myweb.com.
ftp IN CNAME www.myweb.com.
```

Zone File Components Explained

ZONE FILE ANATOMY

\$TTL 86400

└ Time To Live (cache duration in seconds)
86400 = 24 hours = 1 day

SOA Record (Start of Authority)

```
@ IN SOA ns1.myweb.com. admin.myweb.com. (
    ↓     ↓     ↓           ↓
Zone  Type  Primary NS      Admin Email
                  (@ replaced with .)
```

Serial Number: 2024010801

Format: YYYYMMDDNN (Year Month Day Revision Number)

MUST increment with each change!

Refresh: How often secondary checks primary (3600 = 1 hr)

Retry: Retry interval if refresh fails (1800 = 30 min)

Expire: When secondary stops responding (604800 = 1 week)

Minimum: Negative cache TTL (86400 = 1 day)

10. Reverse Zone Configuration

Creating Reverse Zone File

```
# Location: /var/named/reverse.myweb.com.zone

$TTL 86400
@ IN SOA ns1.myweb.com. admin.myweb.com. (
    2024010801 ; Serial Number
    3600        ; Refresh
    1800        ; Retry
    604800      ; Expire
    86400       ; Minimum TTL
)

; Name Server Records
@ IN NS ns1.myweb.com.

; PTR Records (Pointer Records) - Reverse DNS
1 IN PTR myweb.com.
1 IN PTR ns1.myweb.com.
1 IN PTR www.myweb.com.
```

Understanding Reverse Zone Naming

REVERSE ZONE NAMING CONVENTION

IP Address: 10.21.55.1

Reverse Zone Name Construction:

```
IP: 10.21.55.1
↓ ↓ ↓
Reverse: 55.21.10.in-addr.arpa
(Network portion reversed + .in-addr.arpa)
```

PTR Record: Last octet (1) points to domain name

```
1 IN PTR myweb.com.
```


11. DNS Records Types

Common DNS Record Types

DNS RECORD TYPES

A Record (Address)

Maps hostname to IPv4 address

www IN A 10.21.55.1

AAAA Record

Maps hostname to IPv6 address

www IN AAAA 2001:db8::1

CNAME Record (Canonical Name)

Alias for another domain

mail IN CNAME www.myweb.com.

MX Record (Mail Exchange)

Mail server for domain

@ IN MX 10 mail.myweb.com.

NS Record (Name Server)

Authoritative name servers

@ IN NS ns1.myweb.com.

PTR Record (Pointer)

Reverse DNS lookup

1 IN PTR myweb.com.

TXT Record

Text information

@ IN TXT "v=spf1 include:_spf.google.com ~all"

SOA Record (Start of Authority)

Zone administrative information

Records Summary Table

Record	Purpose	Example
A	IPv4 address mapping	www IN A 10.21.55.1
AAAA	IPv6 address mapping	www IN AAAA 2001:db8::1
CNAME	Alias/Canonical name	mail IN CNAME www
MX	Mail server	@ IN MX 10 mail.domain.com
NS	Nameserver	@ IN NS ns1.domain.com
PTR	Reverse lookup	1 IN PTR domain.com
TXT	Text record	@ IN TXT "text"
SOA	Zone authority	Required for every zone
SRV	Service location	_http._tcp IN SRV ...

12. Testing DNS Configuration

Configuration Validation Commands

```
# Check named.conf syntax
named-checkconf /etc/named.conf

# Check zone file syntax
named-checkzone myweb.com /var/named/forward.myweb.com.zone

# Restart DNS service after changes
sudo systemctl restart named

# Check DNS service status
sudo systemctl status named
```

DNS Query Testing Commands

```
# Using nslookup
nslookup myweb.com
nslookup myweb.com 10.21.55.1 # Query specific DNS server

# Using dig
dig myweb.com
dig @10.21.55.1 myweb.com      # Query specific DNS server
dig myweb.com +short            # Short output

# Using host
host myweb.com
host myweb.com 10.21.55.1

# Reverse lookup
dig -x 10.21.55.1
nslookup 10.21.55.1
```

Testing Workflow

DNS TESTING WORKFLOW

Step 1: Validate Configuration

```
$ named-checkconf /etc/named.conf  
$ named-checkzone myweb.com /var/named/forward.zone  
  
Output: OK (if no errors)
```

↓

Step 2: Restart Service

```
$ sudo systemctl restart named  
$ sudo systemctl status named  
  
Check: Active (running)
```

↓

Step 3: Test DNS Resolution

```
$ nslookup myweb.com  
  
Server: 10.21.55.1  
Address: 10.21.55.1#53  
  
Name: myweb.com  
Address: 10.21.55.1
```

13. Web Server Setup with Apache

Installing Apache HTTP Server

```
# Install Apache (httpd)
sudo apt install apache2 -y
# OR for RHEL/CentOS
sudo yum install httpd -y

# Start Apache service
sudo systemctl start httpd
# OR
sudo systemctl start apache2

# Enable auto-start on boot
sudo systemctl enable httpd

# Check service status
sudo systemctl status httpd
```

Apache Configuration

APACHE WEB SERVER SETUP

Default Document Root: /var/www/html/
Configuration File: /etc/httpd/conf/httpd.conf
Default Port: 80 (HTTP), 443 (HTTPS)

File Structure

```
/var/www/html/
    └── index.html      (Default homepage)
        └── styles.css    (Stylesheet)
            └── other files...
```

Create custom webpage:

```
$ sudo vim /var/www/html/index.html
```

Sample HTML File

```
<!-- /var/www/html/index.html -->
<!DOCTYPE html>
<html>
<head>
    <title>My Website</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Welcome to My Website!</h1>
    <p>This website is hosted on my own server with custom DNS.</p>
</body>
</html>
```

14. Firewall Configuration

Allowing DNS and HTTP Traffic

```
# Allow DNS traffic (port 53)
sudo firewall-cmd --permanent --add-service=dns
# OR
sudo firewall-cmd --permanent --add-port=53/tcp
sudo firewall-cmd --permanent --add-port=53/udp

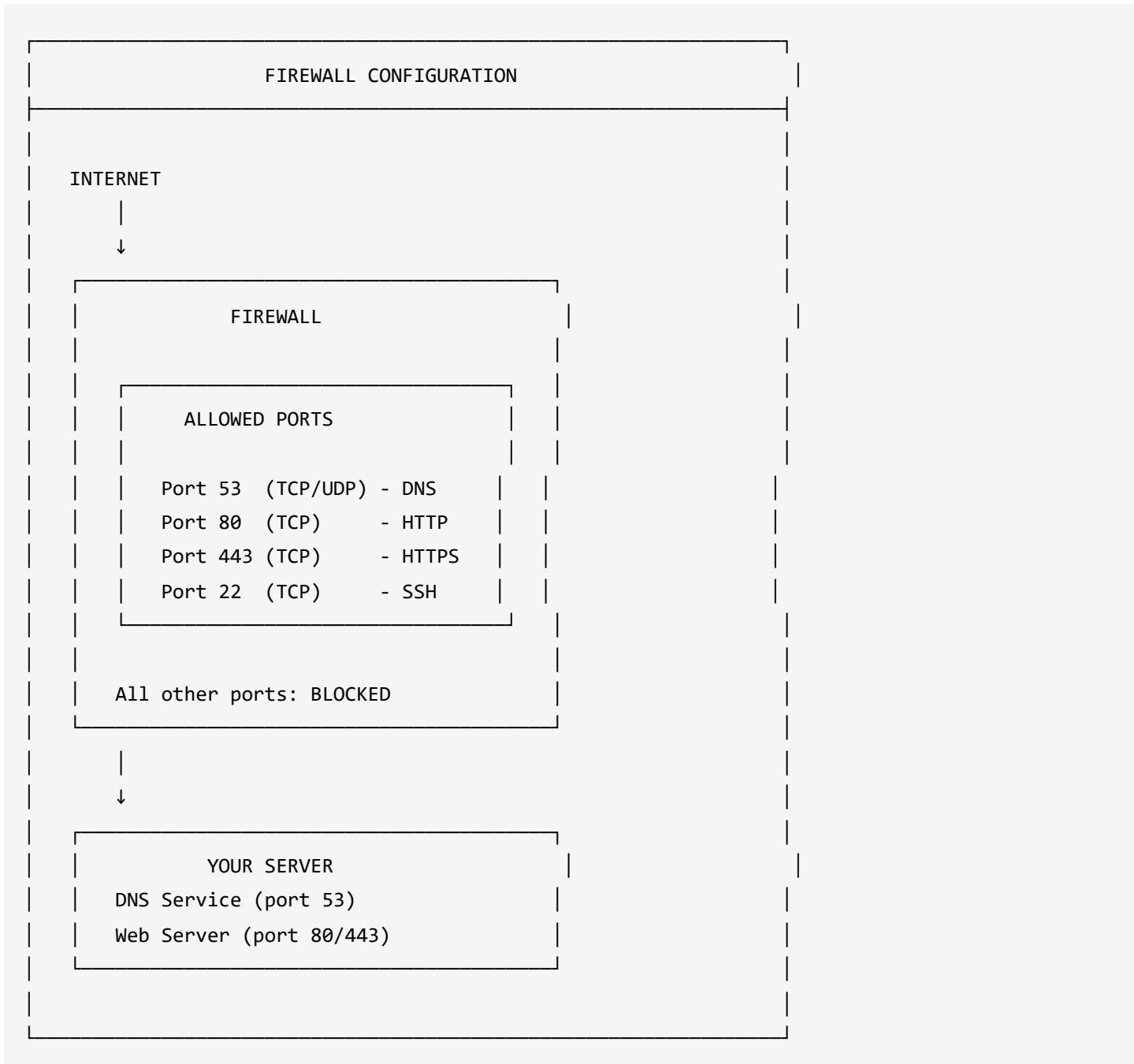
# Allow HTTP traffic (port 80)
sudo firewall-cmd --permanent --add-service=http
# OR
sudo firewall-cmd --permanent --add-port=80/tcp

# Allow HTTPS traffic (port 443)
sudo firewall-cmd --permanent --add-service=https

# Reload firewall to apply changes
sudo firewall-cmd --reload

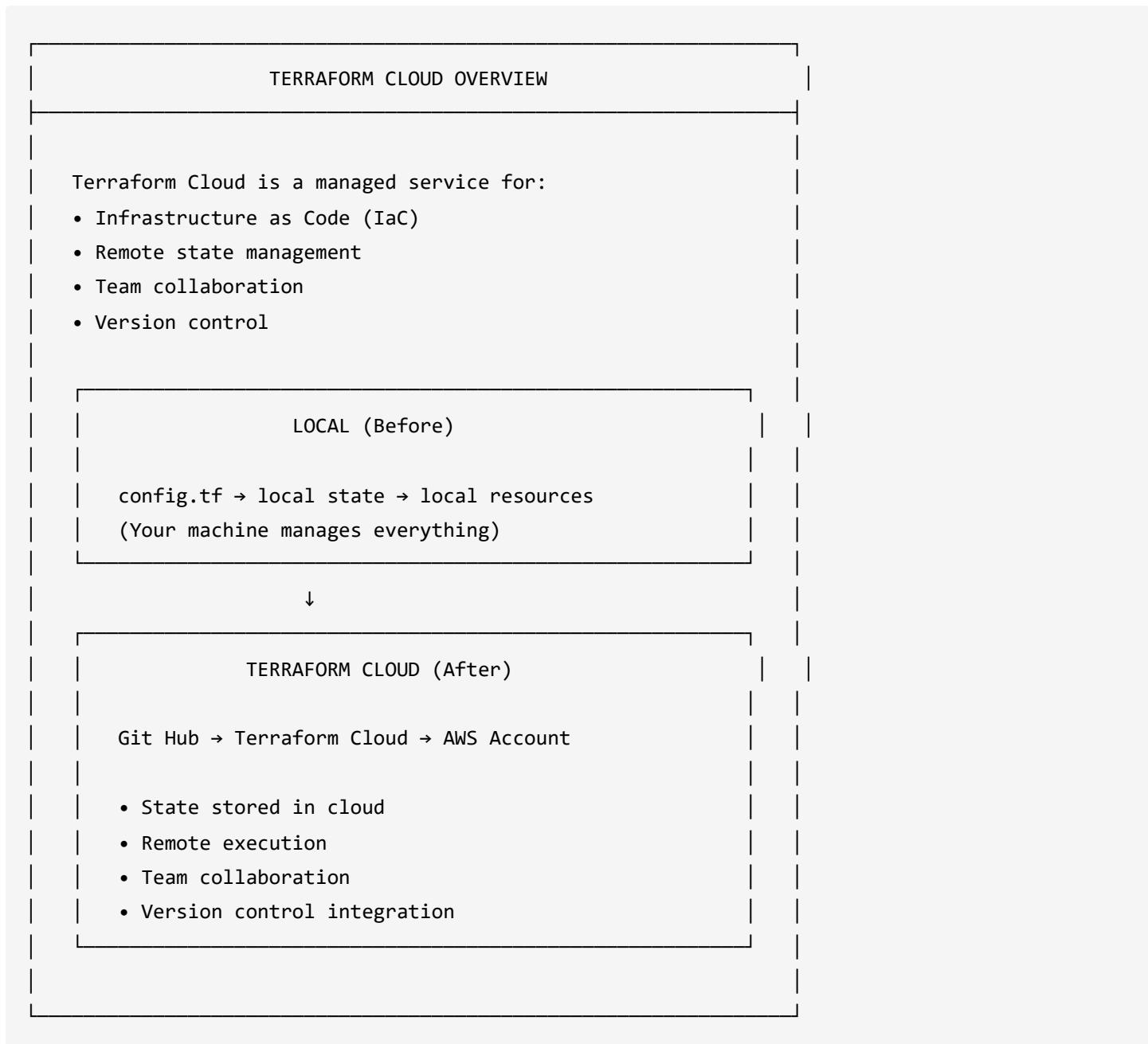
# Verify rules
sudo firewall-cmd --list-all
```

Firewall Architecture



15. Terraform Cloud Introduction

What is Terraform Cloud?



Key Benefits of Terraform Cloud

Feature	Description
Remote State	State file stored securely in cloud
Team Collaboration	Multiple users can work together
Version Control	Integration with GitHub/GitLab
Cost Estimation	Shows cost before applying

Feature	Description
Policy Enforcement	Sentinel policies for governance
Private Registry	Share modules within organization

16. Important Q&A - Real Interview Questions

Q1: Walk me through what happens when you type www.google.com in your browser - explain the complete DNS resolution process.

Answer: This is a classic interview question testing DNS fundamentals:

1. **Browser Cache Check:** Browser first checks its local DNS cache for recent lookups
2. **OS Cache Check:** If not found, checks operating system's DNS cache (`/etc/hosts` on Linux)
3. **Recursive Resolver Query:** OS contacts configured DNS resolver (ISP's DNS or 8.8.8.8)
4. **Root Server Query:** Resolver asks root servers "Who handles .com domains?"
5. **TLD Server Query:** Root returns .com TLD server address; resolver queries it
6. **Authoritative Query:** TLD returns google.com's authoritative nameserver (ns1.google.com)
7. **Final Resolution:** Authoritative server returns IP address (142.250.190.78)
8. **Caching:** Each level caches the result based on TTL
9. **Connection:** Browser establishes TCP connection to the IP address

Interview Tip: Mention caching at each level and TTL's role in propagation time.

Q2: What's the difference between Authoritative and Recursive DNS servers? Which would you use for your company's internal domains?

Answer:

Aspect	Authoritative DNS	Recursive DNS
Function	Stores and serves DNS records	Queries other servers to find answers
Data Source	Has original zone data	Has no original data, only cache
Response Type	Definitive answers	Answers from queries to others
Example	ns1.google.com , Route 53	ISP DNS, Google 8.8.8.8

Aspect	Authoritative DNS	Recursive DNS
Query Direction	Receives queries, doesn't forward	Forwards queries to find answers

For internal domains: Use **Authoritative DNS** (like BIND9 or Windows DNS) because:

- You control the zone data and records
- Internal domains (.local, .internal) won't resolve externally
- Better security - internal records aren't exposed
- Faster resolution for internal services

Best Practice: Configure internal authoritative DNS that forwards external queries to public recursive resolvers.

Q3: A DNS change you made isn't propagating globally even after 24 hours. How would you troubleshoot this?

Answer: Systematic troubleshooting approach:

1. Verify the source is correct:

```
dig @your-authoritative-dns yourdomain.com
# Ensure your DNS server has the correct record
```

2. Check TTL of OLD record:

- Old record's TTL might have been 86400 (24 hours) or higher
- Caches worldwide will hold old value until TTL expires

3. Check serial number in SOA:

```
dig SOA yourdomain.com
# Serial must be incremented for zone transfer
```

4. Test from different locations:

```
dig @8.8.8.8 yourdomain.com      # Google DNS
dig @1.1.1.1 yourdomain.com      # Cloudflare DNS
dig @9.9.9.9 yourdomain.com      # Quad9 DNS
```

5. Check parent delegation:

- Verify NS records at registrar match your nameservers
- Check if nameservers are properly delegated

6. Force cache flush (if you control resolvers):

```
rndc flush  # BIND9
```

Interview Tip: Emphasize that you can't force global propagation - you must wait for TTL expiry. Pre-migration strategy: Lower TTL to 300 seconds days before the change.

Q4: Explain DNS record types - A, AAAA, CNAME, MX, NS, TXT, PTR, SOA. When would you use each?

Answer:

Record	Purpose	Example	Use Case
A	Maps hostname to IPv4	www IN A 192.0.2.1	Web servers, any IPv4 service
AAAA	Maps hostname to IPv6	www IN AAAA 2001:db8::1	IPv6-enabled services
CNAME	Alias to another domain	blog IN CNAME www.example.com.	CDN integration, subdomains pointing to main site
MX	Mail server with priority	@ IN MX 10 mail.example.com.	Email routing, lower number = higher priority
NS	Nameserver delegation	@ IN NS ns1.example.com.	Delegate subdomain to different DNS
TXT	Arbitrary text	@ IN TXT "v=spf1..."	SPF, DKIM, domain verification
PTR	Reverse lookup (IP→Domain)	1 IN PTR mail.example.com.	Email authentication, security logging
SOA	Zone authority info	Contains serial, refresh, retry, expire	Required at zone apex, controls zone transfers

CNAME Restrictions (Important for interviews):

- Cannot exist at zone apex (root domain)
 - Cannot coexist with other records for same name
 - Use ALIAS/ANAME records (if supported) for apex CNAME-like behavior
-

Q5: Your website is experiencing slow DNS resolution times. How would you diagnose and fix this?

Answer:

Diagnosis Steps:

```
# Measure resolution time
dig +stats yourdomain.com | grep "Query time"

# Trace the full resolution path
dig +trace yourdomain.com

# Test against specific DNS servers
dig @8.8.8.8 +stats yourdomain.com
```

Common Causes and Solutions:

Issue	Diagnosis	Solution
High TTL with stale cache	Check TTL values	Set appropriate TTL (300-3600 for dynamic, 86400 for static)
Distant DNS servers	mtr or traceroute to DNS	Use GeoDNS or Anycast DNS (Route 53, Cloudflare)
DNS server overload	Check server metrics	Add more NS servers, use managed DNS
Recursive resolver latency	Test with different resolvers	Use faster public DNS (1.1.1.1, 8.8.8.8)
Missing glue records	dig NS yourdomain.com +trace	Add glue records at registrar
DNSSEC validation failures	dig +dnssec	Fix DNSSEC chain or disable if not needed

Performance Best Practices:

- Use multiple geographically distributed nameservers
- Implement Anycast for global DNS presence
- Set reasonable TTLs (not too low, not too high)
- Use managed DNS services for critical domains

Q6: What is DNS caching and at what levels does it occur? How do you flush DNS cache?

Answer:

DNS Caching Levels (from closest to furthest):

- | DNS CACHING HIERARCHY | |
|--|--|
| Level 1: BROWSER CACHE | |
| • Chrome: chrome://net-internals/#dns | |
| • TTL: Usually 60 seconds | |
| • Flush: Close browser or use internal clear | |
| Level 2: OPERATING SYSTEM CACHE | |
| • Windows: DNS Client service | |
| • Linux: systemd-resolved, nscd | |
| • macOS: mDNSResponder | |
| Level 3: LOCAL DNS RESOLVER | |
| • Corporate DNS server | |
| • Home router DNS | |
| Level 4: ISP RECURSIVE RESOLVER | |
| • ISP's caching DNS server | |
| • Public DNS (8.8.8.8, 1.1.1.1) | |

Flush Commands:

```
# Windows  
ipconfig /flushdns  
  
# macOS  
sudo dscacheutil -flushcache; sudo killall -HUP mDNSResponder  
  
# Linux (systemd)  
sudo systemd-resolve --flush-caches  
  
# BIND9 Server  
sudo rndc flush
```

Interview Tip: You can't flush ISP or public DNS caches - you must wait for TTL expiry.

Q7: What is BIND9 and how does it differ from other DNS server software? When would you choose BIND vs managed DNS?

Answer:

BIND9 (Berkeley Internet Name Domain):

- Most widely used open-source DNS server
- Supports authoritative, recursive, and caching modes
- Highly configurable with zone files
- Industry standard since 1980s

Comparison with Alternatives:

Feature	BIND9	PowerDNS	Unbound	Managed (Route 53)
Type	Full-featured	Authoritative	Recursive only	Authoritative
Backend	Zone files	Database support	Memory/config	Cloud-native
Performance	Good	Excellent	Excellent	Excellent
Management	Manual	API/DB driven	Simple	Console/API
DNSSEC	Full support	Full support	Validation only	Full support
Cost	Free	Free	Free	Pay-per-query

When to Choose:

Scenario	Best Choice	Reason
Internal corporate DNS	BIND9 or Windows DNS	Control, no external queries
High-traffic public website	Managed DNS (Route 53, Cloudflare)	Global anycast, DDoS protection
Learning/Lab environment	BIND9	Industry standard, great learning
Database-driven DNS	PowerDNS	Native database backends
Recursive resolver only	Unbound	Lightweight, secure, fast

Q8: How does DNS load balancing work? What are its limitations compared to traditional load balancers?

Answer:

DNS Load Balancing Methods:

1. **Round Robin:** Multiple A records for same hostname

```
www IN A 192.0.2.1
www IN A 192.0.2.2
www IN A 192.0.2.3
```

2. **Weighted Round Robin:** Different weights for servers (Route 53 weighted routing)
3. **GeoDNS:** Return different IPs based on client location (latency-based routing)
4. **Health-Based:** Only return healthy server IPs

Limitations vs Traditional Load Balancers:

Aspect	DNS Load Balancing	Traditional LB (ALB/NLB)
Health Checks	Slow (TTL-dependent)	Real-time (seconds)
Granularity	Per-request (cached)	Per-connection/request
Session Persistence	Difficult	Native support
SSL Termination	Not possible	Supported
Caching Issues	Can't control client cache	No caching issues
Failover Speed	Minutes (TTL)	Seconds
Cost	Low (just DNS queries)	Higher (compute resources)

Best Practice: Use DNS for geographic/global distribution, traditional LB for server-level distribution.

```
Users → DNS (Geographic) → Regional Load Balancer → Server Pool
```

Q9: Explain DNS zone transfer. What are AXFR and IXFR? How do you secure zone transfers?

Answer:

Zone Transfer Types:

Type	Full Name	Description	Use Case
AXFR	Full Zone Transfer	Transfers entire zone	Initial sync, zone corruption recovery
IXFR	Incremental Zone Transfer	Only changed records	Regular sync between primary/secondary

How Zone Transfer Works:

1. Secondary DNS checks SOA serial number
2. If primary's serial is higher, transfer is initiated
3. AXFR: Complete zone sent
4. IXFR: Only differences since last serial sent

Security Risks:

- Zone transfer reveals all DNS records (reconnaissance)
- Attackers can map entire infrastructure
- Historical breaches caused by open zone transfers

Securing Zone Transfers:

```
# In named.conf - Restrict by IP
zone "example.com" {
    type master;
    file "example.com.zone";
    allow-transfer { 192.0.2.10; 192.0.2.11; }; # Only secondary DNS IPs
    also-notify { 192.0.2.10; 192.0.2.11; };
};

# TSIG Key-based authentication (more secure)
key "transfer-key" {
    algorithm hmac-sha256;
    secret "base64encodedkey==";
};

zone "example.com" {
    type master;
    allow-transfer { key "transfer-key"; };
};
```

Test Zone Transfer Security:

```
dig @ns1.example.com example.com AXFR
# Should return "Transfer failed" if properly secured
```

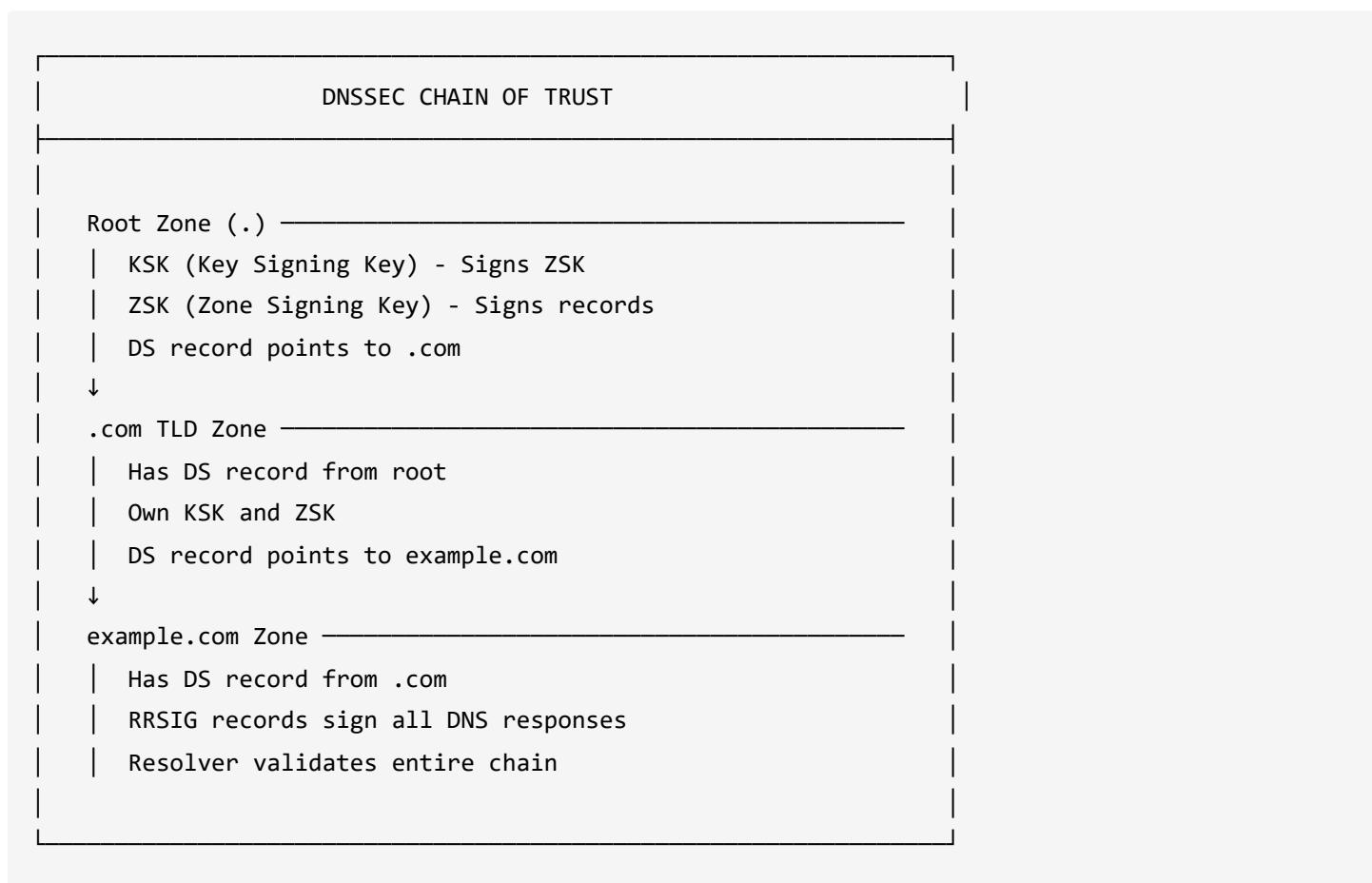
Q10: What is DNSSEC? How does it work and when would you implement it?

Answer:

DNSSEC (DNS Security Extensions):

- Adds cryptographic signatures to DNS records
- Prevents DNS spoofing and cache poisoning
- Creates chain of trust from root to your domain

How DNSSEC Works:



Key Records:

Record	Purpose
DNSKEY	Public key for the zone
RRSIG	Signature for each record set
DS	Delegation Signer (parent zone trust)
NSEC/NSEC3	Proof of non-existence

When to Implement:

- Financial services (regulatory requirement)
- Government domains
- High-security applications
- When DNS spoofing is a significant risk

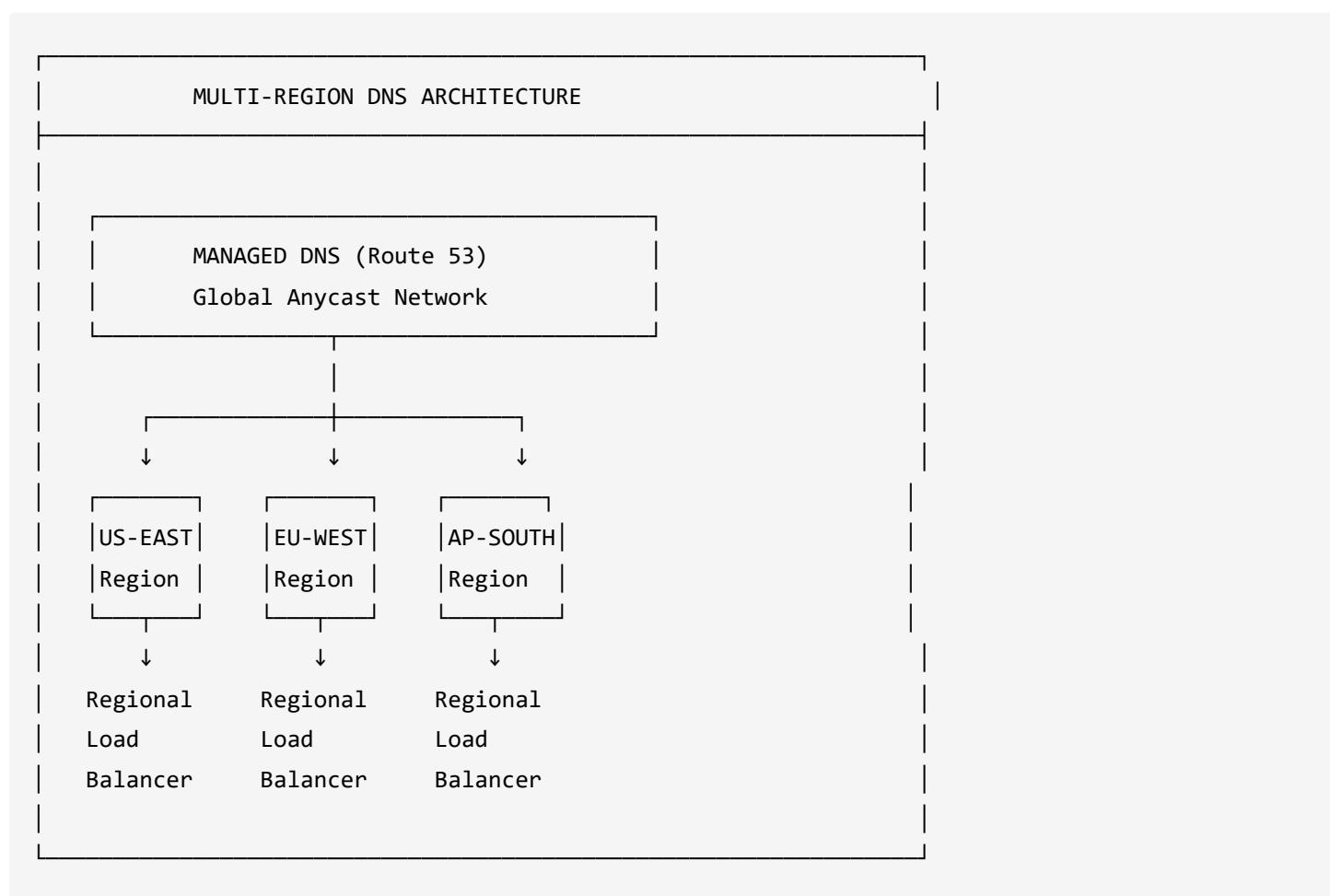
Challenges:

- Key management complexity
- Zone file size increases
- Must coordinate key rollovers
- Some old resolvers don't validate

Q11: You're setting up DNS for a new multi-region web application. Design the DNS architecture.

Answer:

Architecture Design:



DNS Records Configuration:

```

# Primary domain with latency-based routing
www.example.com:
  - Latency routing to us-east-alb.example.com (US users)
  - Latency routing to eu-west-alb.example.com (EU users)
  - Latency routing to ap-south-alb.example.com (Asia users)

# Health checks on each endpoint
# Failover to healthy regions if one goes down

# Static assets via CDN
static.example.com → CloudFront distribution

# API with weighted routing for canary deployments
api.example.com:
  - 90% weight → api-v1.example.com
  - 10% weight → api-v2.example.com (canary)

```

Best Practices:

1. **Use managed DNS** for global anycast and DDoS protection
2. **Health checks** with automatic failover
3. **Low TTL** (60-300s) for dynamic endpoints
4. **Higher TTL** (3600+) for stable records
5. **Separate DNS** for internal services (Private Hosted Zones)
6. **DNSSEC** for sensitive domains

Q12: What is the difference between public and private DNS zones? How do you implement split-horizon DNS?

Answer:

Public vs Private DNS:

Aspect	Public DNS	Private DNS
Accessibility	Internet-wide	VPC/Network internal only
Records	Public-facing services	Internal services, databases
Resolution	Any DNS resolver	Only authorized resolvers
AWS Implementation	Public Hosted Zone	Private Hosted Zone
Security	Exposed to internet	Protected within network

Split-Horizon DNS (Split-Brain):

Same domain name returns different IPs based on where the query originates.

SPLIT-HORIZON DNS

Query: app.example.com

INTERNAL USER (Corporate Network)

DNS Response: 10.0.1.50 (Private IP)

→ Direct connection to internal server

EXTERNAL USER (Internet)

DNS Response: 203.0.113.50 (Public IP)

→ Through load balancer/firewall

AWS Implementation:

```
# Public Hosted Zone (Internet queries)
app.example.com → ALB Public IP

# Private Hosted Zone (VPC queries)
app.example.com → Internal ALB IP or EC2 private IP

# Associate Private Zone with VPC
# Internal queries automatically use Private Zone
```

Use Cases:

- Internal apps accessible without going through internet
- Different configurations for testing vs production
- Security - hide internal architecture from public DNS

Q13: How do you configure DNS for email? Explain SPF, DKIM, and DMARC records.

Answer:

Email DNS Records Overview:

EMAIL DNS AUTHENTICATION

MX Record: "Where to deliver email"

```
example.com IN MX 10 mail.example.com.
```

SPF Record: "Who CAN send email for this domain"

```
example.com IN TXT "v=spf1 include:_spf.google.com ~all"
```

DKIM Record: "Cryptographic signature verification"

```
selector._domainkey.example.com IN TXT "v=DKIM1; p=..."
```

DMARC Record: "What to do with failed authentication"

```
_dmarc.example.com IN TXT "v=DMARC1; p=reject; ..."
```

Detailed Explanation:

Record	Purpose	Example	What Happens on Failure
SPF	Lists authorized mail servers	v=spf1 ip4:192.0.2.0/24 include:sendgrid.net -all	Soft fail (~) or hard fail (-)
DKIM	Cryptographic signature	Public key in DNS, signature in email header	Receiving server can't verify sender
DMARC	Policy for SPF/DKIM failures	v=DMARC1; p=reject; rua=mailto:dmarc@example.com	Reject, quarantine, or none

SPF Syntax:

```

v=spf1                                # Version
ip4:192.0.2.1                           # Specific IP allowed
include:_spf.google.com                 # Include Google's SPF
a:mail.example.com                      # A record IP allowed
mx                                      # MX record IPs allowed
-all                                    # Hard fail all others
~all                                    # Soft fail (mark suspicious)

```

Interview Tip: DMARC alignment - both SPF and DKIM must pass AND align with the From header domain for full authentication.

Q14: What is DNS TTL? How do you decide the right TTL value for different record types?

Answer:

TTL (Time To Live):

- Specifies how long (in seconds) DNS resolvers should cache a record
- Lower TTL = faster propagation, more DNS queries, higher cost
- Higher TTL = slower propagation, fewer queries, better caching

TTL Decision Matrix:

Record Type	Recommended TTL	Reason
Static website	86400 (24h)	Rarely changes
API endpoints	300-600 (5-10m)	May need quick changes
Load balancer	60-300 (1-5m)	Health check failover
Mail (MX)	3600-86400	Rarely changes
During migration	60-300	Need fast cutover
CDN/CloudFront	86400+	CloudFront handles actual routing
Development	60	Frequent changes

Migration Strategy:

```
Day -7: Lower TTL to 300 seconds
Day 0: Make DNS change
Day +1: Verify all traffic migrated
Day +2: Raise TTL back to 86400
```

Cost Implications:

- Route 53 charges per query
- TTL 60 = ~1,440 queries/day per user
- TTL 86400 = 1 query/day per user
- For high-traffic sites, this significantly impacts cost

Q15: Your DNS server is being used in a DNS amplification attack. How do you identify and mitigate this?

Answer:

DNS Amplification Attack:

- Attacker sends DNS queries with spoofed source IP (victim's IP)
- DNS server sends large responses to victim
- Amplification factor: Small query → Large response (up to 70x)

Identification Signs:

```
# High query volume from unusual sources
# Many queries for ANY or TXT records
# Queries for domains you don't host

# Check query logs
grep "ANY" /var/log/named/query.log | wc -l
```

Mitigation Steps:

- 1. Disable recursion for public (if authoritative only):**

```
# named.conf
options {
    recursion no;
    additional-from-cache no;
};
```

- 2. Implement Response Rate Limiting (RRL):**

```
# named.conf
rate-limit {
    responses-per-second 5;
    window 5;
};
```

3. Restrict ANY queries:

```
# Minimal responses to ANY
minimal-any yes;
```

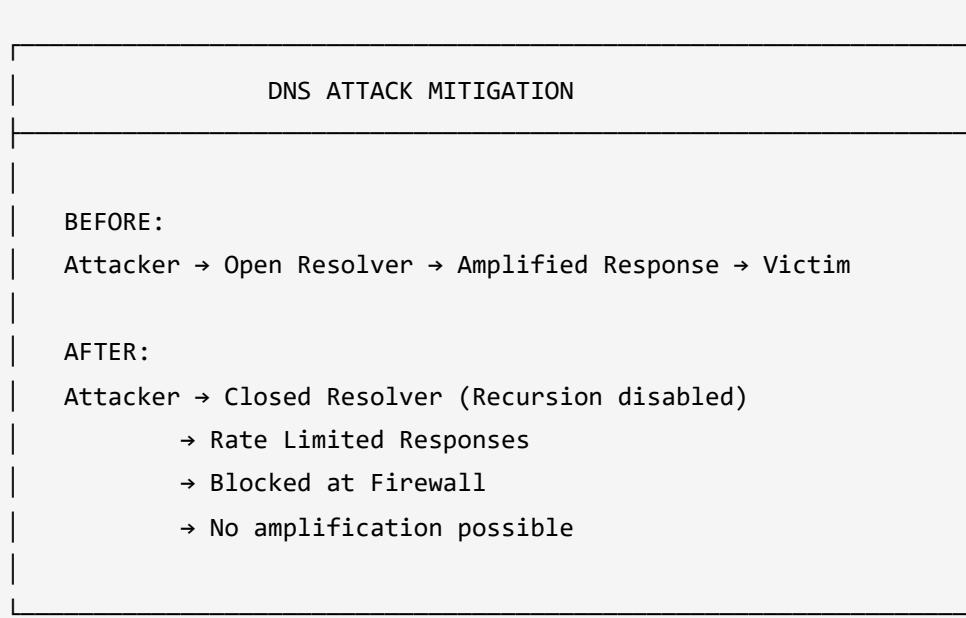
4. Block known bad sources:

```
# Using iptables
iptables -A INPUT -p udp --dport 53 -m hashlimit \
--hashlimit-above 20/sec --hashlimit-mode srcip \
-j DROP
```

5. Use managed DNS services:

- Route 53, Cloudflare have built-in DDoS protection
- Anycast absorbs attack traffic globally

Prevention Architecture:



Interview Tip: Emphasize that recursive resolvers should NOT be publicly accessible. Use separate servers for authoritative and recursive functions.

Quick Reference Commands

Installation & Service Management

```
# Install
sudo apt install bind9 -y

# Start/Stop/Restart
sudo systemctl start named
sudo systemctl stop named
sudo systemctl restart named

# Status
sudo systemctl status named
```

Configuration Testing

```
# Check syntax
named-checkconf /etc/named.conf
named-checkzone myweb.com /var/named/forward.zone

# Query DNS
nslookup myweb.com
dig @10.21.55.1 myweb.com
```

Firewall

```
# Allow DNS
sudo firewall-cmd --permanent --add-service=dns
sudo firewall-cmd --reload
```

Summary Table

Topic	Key Points
DNS	Domain Name System - translates names to IPs
BIND9	Most popular DNS server software
Port	53 (TCP/UDP)

Topic	Key Points
Forward Zone	Domain → IP mapping (A records)
Reverse Zone	IP → Domain mapping (PTR records)
TTL	Cache duration for records
SOA	Zone administrative information
named-checkconf	Validates configuration
named-checkzone	Validates zone files
Firewall	Must allow port 53 for DNS

End of Notes 10: DNS Server Configuration

This concludes the AWS Zero to Hero Series - All 10 notes files covering comprehensive AWS and DevOps topics from IAM to DNS configuration.