WHAT IS AWS (AMAZON WEB SERVICES)?

DEFINITION

AWS is a cloud provider that provides computing services over the internet.

What is cloud? Instead of buying and maintaining your own physical servers, you rent computing resources from AWS.

AWS SERVICE MODELS

AWS falls into multiple categories:

1. IaaS (Infrastructure as a Service)

What it means: AWS provides the basic infrastructure (virtual servers, storage, networking), and you manage everything else.

Example: EC2 (Elastic Compute Cloud) - Virtual servers. You get a blank virtual machine. You install everything yourself. Full control, but more responsibility.

---

2. PaaS (Platform as a Service)

What it means: AWS provides the infrastructure AND the platform (operating system, runtime, tools), you just deploy your application.

Example: Elastic Beanstalk - Just upload your code. AWS manages servers, scaling, updates. You focus only on your application.

---

3. SaaS (Software as a Service)

What it means: AWS provides complete ready-to-use software applications.

Example: Amazon WorkMail - Email service. Amazon Chime - Video conferencing. You just use the software, no setup needed.

THE MAIN IDEA OF AWS

AWS makes life easy by providing your app in service model.

Traditional approach (hard way):

1. Buy physical servers
2. Set up data center
3. Install operating system
4. Configure networking
5. Install Kubernetes
6. Manage Kubernetes
7. Update Kubernetes
8. Monitor Kubernetes
9. Fix issues
10. Scale manually

This is overhead - takes months and requires specialized team.

AWS approach (easy way):

1. Go to AWS platform
2. Click a button
3. Get Kubernetes service (EKS)
4. Just scale up or down
5. AWS handles everything behind the scenes

This takes minutes and requires less expertise.

KUBERNETES EXAMPLE

Problem without AWS:

- Installing Kubernetes: Complex, takes days
- Management: Need dedicated team
- Updates: Manual, risky
- Scaling: Manual configuration
- Monitoring: Set up yourself
- High availability: Configure yourself

- Security: Your responsibility
- Cost: Expensive infrastructure

Solution with AWS EKS (Kubernetes as a Service):

- Installation: Click button, done in 10 minutes
- Management: AWS handles it
- Updates: Automatic
- Scaling: Click button or auto-scale
- Monitoring: Built-in CloudWatch
- High availability: AWS guarantees it
- Security: AWS best practices applied
- Cost: Pay only for what you use

You focus on: Deploying your applications AWS handles: All the complex infrastructure

---

AWS SERVICE COUNT

AWS has more than 200+ services.

Categories include:

- Compute (servers, containers)
- Storage (files, databases)
- Networking (connectivity, CDN)
- Security (IAM, encryption)
- Machine Learning (AI services)
- IoT (Internet of Things)
- Analytics (data processing)
- And many more

As a DevOps engineer: It is practically impossible to remember all these services.

What you need: Learn AWS DevOps-related services that you'll actually use.

DEVOPS PHILOSOPHY

DevOps is always about:

1. Automation - Reduce manual tasks
2. Increasing efficiency - Do more with less time and effort

AWS provides tools to achieve both.

---

15 ESSENTIAL AWS SERVICES FOR DEVOPS

---

1. EC2 (ELASTIC COMPUTE CLOUD)

What it is: Virtual servers (virtual machines) in the cloud.

Definition: You rent a computer that exists in AWS data center. You control it remotely via internet.

What you can do:

- Install any operating system (Linux, Windows)
- Install any software
- Run applications
- Host websites
- Run databases
- Process data

Example: You have a Node.js application:

1. Create EC2 instance
2. Install Node.js
3. Deploy your app
4. Your app is now running 24/7

Key features:

Elastic:

- Scale up (more powerful server) or down (less powerful)
- Add more servers during high traffic
- Remove servers during low traffic

Pricing:

- Pay only for hours used
- Stop instance when not needed = no charges
- Different instance types for different needs

Instance types:

t2.micro: 1 CPU, 1GB RAM - Good for testing

t2.medium: 2 CPU, 4GB RAM - Good for small apps

c5.large: 2 CPU, 4GB RAM - Compute optimized

r5.large: 2 CPU, 16GB RAM - Memory optimized

Use cases for DevOps:

- Application servers
- CI/CD build servers
- Testing environments
- Development servers
- Jump boxes (bastion hosts)

---

2. VPC (VIRTUAL PRIVATE CLOUD)

What it is: Your own private network in AWS cloud.

Definition: Your private, isolated section of AWS. Your private data center in the cloud.

Why needed: Without VPC, all your resources would be on public internet - huge security risk.

Components of VPC:

a) CIDR (Classless Inter-Domain Routing):

What it is: Defines the IP address range for your VPC.

Example:

VPC CIDR: 10.0.0.0/16

This gives you: 65,536 IP addresses

Range: 10.0.0.0 to 10.0.255.255

Why important:

- Determines how many resources you can create
- Ensures no IP conflicts
- Organizes your network

---

b) Subnets:

What they are: Smaller networks within your VPC.

Types:

Public Subnet:

- Has internet access
- Resources can be accessed from internet
- Example: Web servers

Private Subnet:

- No direct internet access
- More secure
- Example: Databases

Example setup:

VPC: 10.0.0.0/16

Public Subnet 1: 10.0.1.0/24 (256 IPs) - Web servers

Public Subnet 2: 10.0.2.0/24 (256 IPs) - Load balancers

Private Subnet 1: 10.0.10.0/24 (256 IPs) - Databases

Private Subnet 2: 10.0.11.0/24 (256 IPs) - App servers

---

c) Security Groups:

What they are: Virtual firewalls for your EC2 instances.

How they work: Control which traffic is allowed in (inbound) and out (outbound).

Example rules:

Inbound:

- Allow SSH (port 22) from my office IP only

- Allow HTTP (port 80) from anywhere

- Allow HTTPS (port 443) from anywhere

- Deny everything else

Outbound:

- Allow all traffic (default)

Real-world scenario:

Web Server Security Group:

Inbound:

- Port 80 (HTTP) from 0.0.0.0/0 (anywhere)

- Port 443 (HTTPS) from 0.0.0.0/0 (anywhere)

- Port 22 (SSH) from 203.0.113.5 (your office IP only)

Database Security Group:

Inbound:

- Port 3306 (MySQL) from Web Server Security Group only

- No internet access

Security benefit: Even if hacker gets into web server, they cannot access database from outside.

---

d) Internet Gateway:

What it is: Door to the internet for your VPC.

Purpose: Allows resources in public subnet to access internet.

---

e) NAT Gateway:

What it is: Allows resources in private subnet to access internet (one-way).

Use case:

Private database needs to:

- Download updates

- Access external APIs

But should NOT be accessible from internet

NAT Gateway allows outbound traffic but blocks inbound.

VPC Use Cases for DevOps:

Example 1: Three-tier application

Internet

Internet Gateway

Public Subnet: Load Balancer

Private Subnet: Application Servers

Private Subnet: Database Servers

Example 2: Dev/Test/Prod separation

VPC 1: Development (10.1.0.0/16)

VPC 2: Testing (10.2.0.0/16)

VPC 3: Production (10.3.0.0/16)

Each completely isolated

---

3. EBS (ELASTIC BLOCK STORE)

What it is: Storage volumes (hard drives) for EC2 instances.

Definition: External hard drives you attach to your virtual servers.

Why needed: EC2 instances by default have temporary storage that's deleted when instance stops. EBS provides permanent storage.

Types of EBS volumes:

a) General Purpose SSD (gp3/gp2):

- Balanced price and performance
- Good for most workloads
- Use case: Operating system, applications

b) Provisioned IOPS SSD (io2/io1):

- High performance
- Expensive
- Use case: Databases, high-traffic applications

c) Throughput Optimized HDD (st1):

- Good for large sequential reads/writes
- Cheaper than SSD
- Use case: Big data, log processing

d) Cold HDD (sc1):

- Lowest cost
- Infrequently accessed data
- Use case: Backups, archives

EBS Features:

Snapshots:

- Backup of your volume
- Can create new volume from snapshot
- Stored in S3
- Incremental (only changes are saved)

Example:

1. Create EBS volume (100GB)
2. Attach to EC2 instance
3. Install application and data
4. Create snapshot (backup)
5. Instance crashes? Create new instance from snapshot

Encryption:

- Data encrypted at rest
- Meet compliance requirements
- No performance impact

Resizing:

- Increase size without downtime
- Change type (gp2 to io2)
- Flexible

DevOps Use Cases:

Use Case 1: Database storage

EC2 instance with MySQL

EBS volume for database files

Daily snapshots for backup

Use Case 2: CI/CD build artifacts

Jenkins server on EC2

EBS volume for build cache

Speeds up builds

Use Case 3: Disaster recovery

Production server with EBS

Daily snapshots

Server fails? Create new from snapshot

Downtime: Minutes instead of hours

---

4. S3 (SIMPLE STORAGE SERVICE)

What it is: Object storage service - store files in the cloud.

Definition: Cloud storage for files and objects at massive scale.

Key concepts:

Buckets:

- Container for files
- Globally unique name

Objects:

- Files you store
- Can be any type (images, videos, logs, backups)
- Up to 5TB per file

S3 Features:

1. Durability: 99.999999999% (11 nines)

- AWS stores copies across multiple data centers
- Virtually impossible to lose data

2. Availability: 99.99%

- Always accessible
- High performance

3. Scalability:

- Store unlimited data
- Handle millions of requests

4. Versioning:

- Keep multiple versions of files
- Recover from accidental deletion
- Roll back to previous version

5. Lifecycle policies:

- Automatically move old data to cheaper storage
- Delete old files automatically

6. Static website hosting:

- Host simple websites directly from S3
- No need for web server

Storage Classes:

S3 Standard:

- Frequently accessed data
- Most expensive
- Use: Active files, website content

S3 Intelligent-Tiering:

- Automatically moves data between access tiers
- Cost optimization
- Use: Unknown access patterns

S3 Standard-IA (Infrequent Access):

- Less frequently accessed
- Cheaper than Standard
- Use: Backups, disaster recovery

S3 Glacier:

- Archival storage
- Very cheap
- Retrieval takes minutes to hours
- Use: Long-term archives, compliance data

S3 Glacier Deep Archive:

- Cheapest storage
- Retrieval takes 12 hours
- Use: Data you rarely/never access

DevOps Use Cases:

Use Case 1: Static website hosting

Store HTML, CSS, JS files in S3

Enable static website hosting

Access via S3 URL or custom domain

Cost: Very low

Use Case 2: Application logs

Application writes logs to S3

Centralized log storage

Analyze with AWS Athena

Lifecycle policy: Delete logs after 90 days

Use Case 3: Backup and disaster recovery

Database backups stored in S3

Daily snapshots stored in S3

Instant recovery if needed

Use Case 4: CI/CD artifacts

Build artifacts stored in S3

Lambda functions code in S3

CloudFormation templates in S3

Version control for deployments

Use Case 5: Data lake

Store all company data in S3

Analyze with analytics tools

Machine learning training data

Centralized data repository

S3 Pricing:

- Pay for storage used (GB-month)
- Pay for data transfer out
- Pay for requests (GET, PUT)
- Very cost-effective

---

5. IAM (IDENTITY AND ACCESS MANAGEMENT)

What it is: Security service to control who can access what in AWS.

Definition: Controls authentication (who you are) and authorization (what you can do) in AWS.

Why IAM is Critical:

Problem without IAM:

Everyone uses root account

Full access to everything

Can delete production database

Can create massive bills

Cannot track who did what

Security nightmare

Solution with IAM:

Create individual users

Give only needed permissions

Track all activities

Revoke access easily

Secure and controlled

IAM Components:

1. Users:

- Individual people or applications
- Each has unique credentials
- Example: john-developer, sarah-admin

2. Groups:

- Collection of users
- Assign permissions to group

- Example:
    - Developers group: Can deploy code
    - Admins group: Can create resources
    - QA group: Can view logs

3. Roles:

- Temporary permissions
- For AWS services
- Example: EC2 instance accessing S3

4. Policies:

- JSON documents defining permissions
- Attached to users, groups, or roles

IAM Policy Example:

Allow read-only access to S3:

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

```
 ]
}
```

Allow EC2 management:

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

IAM Best Practices:

1. Never use root account for daily work
2. Enable MFA (Multi-Factor Authentication)
3. Principle of least privilege
4. Use roles for EC2 instances
5. Rotate credentials regularly
6. Use groups, not individual users

DevOps Use Cases:

Use Case 1: CI/CD pipeline

Create IAM role for Jenkins

Give permissions to:

- Deploy to EC2

- Update S3

- Invoke Lambda

- Update databases

Use Case 2: Application accessing AWS services

Node.js app needs to upload to S3

Create IAM role for EC2 instance

Attach role with S3 write permissions

App uses role automatically (no credentials in code)

Use Case 3: Team structure

Developers group:

- Deploy to dev/test environments

- View all resources

- Cannot delete production

DevOps group:

- Full access to all environments

- Create infrastructure

- Manage costs

QA group:

- Read-only access

- View logs

- Run tests

---

6. CLOUDWATCH (MONITORING SERVICE)

What it is: Monitoring and observability service for AWS resources and applications.

Definition: Watches everything in your AWS environment and alerts you if something goes wrong.

What CloudWatch monitors:

1. AWS Resources:

- EC2: CPU, memory, disk usage
- RDS: Database performance
- Lambda: Function executions
- ELB: Load balancer traffic

2. Application Metrics:

- Custom metrics from your app
- API response times
- Business metrics (orders, signups)

3. Logs:

- Application logs
- System logs
- Access logs

CloudWatch Components:

1. Metrics:

What they are: Numerical data points over time.

Examples:

EC2 CPU Utilization: 45%, 67%, 89%

Application error count: 5, 12, 3

Website visitors: 1000, 1500, 2000

2. Alarms:

What they are: Notifications when metrics cross thresholds.

Example:

If CPU > 80% for 5 minutes

Send email to admin

Auto-scale (add more servers)

3. Logs:

What they are: Text data from applications and systems.

Example:

[2024-11-07 10:00:00] INFO: User logged in

[2024-11-07 10:00:05] ERROR: Database connection failed

[2024-11-07 10:00:10] WARN: High memory usage

4. Dashboards:

What they are: Visual displays of metrics showing CPU usage, disk space, alarms, and other monitoring data.

DevOps Use Cases:

Use Case 1: Auto-scaling based on metrics

Monitor: EC2 CPU usage

If CPU > 75%: Add more EC2 instances

If CPU < 25%: Remove EC2 instances

Result: Automatic scaling based on demand

Use Case 2: Application monitoring

Send custom metrics from application:

- Number of orders per minute

- Average response time

- Error rate

Set alarms for abnormal values

Get notified immediately

Use Case 3: Centralized logging

All EC2 instances send logs to CloudWatch

All Lambda functions send logs to CloudWatch

Search across all logs from one place

Debug issues faster

Use Case 4: Cost monitoring

Track resource usage

Set billing alarms

Get notified if costs exceed budget

Optimize resource usage

---

7. LAMBDA (SERVERLESS COMPUTE)

What it is: Run code without managing servers.

Definition: Upload your code, AWS runs it automatically when triggered. You pay only for execution time.

How Lambda Works:

Traditional approach:

1. Create EC2 instance

2. Install runtime (Node.js, Python)

3. Deploy code

4. Keep server running 24/7

5. Pay for 24/7 even if code runs 5 minutes/day

Lambda approach:

1. Upload code to Lambda

2. Code runs only when triggered

3. AWS handles everything

4. Pay only for execution time (milliseconds)

Lambda Characteristics:

Serverless:

- No servers to manage
- AWS handles infrastructure
- Automatic scaling

Event-driven:

- Triggered by events
- S3 file upload triggers Lambda
- API Gateway request triggers Lambda
- CloudWatch alarm triggers Lambda

Pay-per-use:

First 1 million requests/month: Free

After that: $0.20 per 1 million requests

Execution time: $0.00001667 per GB-second

Example:

Function runs 1 million times/month

Each execution: 100ms, 128MB

Cost: ~$0.21/month

Lambda Supported Languages:

- Node.js
- Python
- Java
- Go
- Ruby
- C# (.NET)
- Custom runtimes

Lambda Limits:

Execution time:

- Maximum: 15 minutes
- Not for long-running tasks

Memory:

- 128 MB to 10 GB

Package size:

- 50 MB (zipped)
- 250 MB (unzipped)

DevOps Use Cases:

Use Case 1: Automated backups

CloudWatch Event: Every day at 2 AM

Triggers Lambda function

Lambda creates EC2 snapshots

Lambda sends notification email

Use Case 2: Image processing

User uploads image to S3

S3 triggers Lambda

Lambda resizes image

Lambda saves thumbnail to S3

Use Case 3: API backend

API Gateway receives HTTP request

Triggers Lambda function

Lambda processes request

Lambda queries database

Lambda returns response

Serverless API with auto-scaling

Use Case 4: Log processing

Application writes logs to S3

S3 triggers Lambda

Lambda parses logs

Lambda sends errors to Slack

Lambda stores metrics in CloudWatch

Use Case 5: CI/CD automation

Developer pushes code to GitHub

GitHub webhook triggers Lambda

Lambda runs tests

Lambda deploys if tests pass

Lambda sends notification

---

8. AWS CODE BUILD SERVICES (CI/CD SERVICES)

What they are: AWS services for building, testing, and deploying applications automatically.

Three main services:

1. AWS CodePipeline
2. AWS CodeBuild
3. AWS CodeDeploy

---

a) AWS CODEPIPELINE

What it is: Continuous Integration and Continuous Delivery service.

Definition: Automates the steps to release your software.

How it works:

Source Stage: Get code from GitHub

Build Stage: Compile and test code

Deploy Stage: Deploy to servers

Example pipeline:

1. Developer pushes code to GitHub

2. CodePipeline detects change

3. Triggers CodeBuild to compile

4. Runs automated tests

5. If tests pass, triggers CodeDeploy

6. Deploys to production

7. Sends notification

All automatic!

Benefits:

- Automate release process
- Fast and consistent deployments
- Reduce human errors
- Deploy multiple times per day

---

b) AWS CODEBUILD

What it is: Fully managed build service.

Definition: Compiles code, runs tests, produces deployable artifacts.

How it works:

1. Get source code

2. Install dependencies

3. Run build commands

4. Run tests

5. Create deployment package

6. Store artifacts in S3

Build specification (buildspec.yml):

version: 0.2

```
phases:
 install:
  commands:
   - npm install
 build:
  commands:
   - npm run build
 post_build:
  commands:
   - npm test

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

Benefits:

- No build servers to manage
- Scales automatically
- Pay only for build time
- Parallel builds

Use case:

Node.js application:

1. CodeBuild gets code from GitHub
2. Installs npm packages
3. Runs webpack to bundle
4. Runs unit tests
5. Creates deployment package
6. Stores in S3

---

c) AWS CODEDEPLOY

What it is: Automated deployment service.

Definition: Deploys code to EC2, Lambda, or on-premises servers.

How it works:

1. Get deployment package from S3

2. Stop old application version

3. Deploy new version

4. Start new application

5. Run health checks

6. Rollback if deployment fails

Deployment strategies:

All-at-once:

- Deploy to all servers simultaneously
- Fast but risky
- Downtime if deployment fails

Rolling:

- Deploy to servers in batches
- Example: Update 2 servers at a time
- Slower but safer

Blue/Green:

- Create new environment (Green)
- Deploy to Green
- Switch traffic from Blue to Green
- Keep Blue as backup
- Zero downtime

Canary:

- Deploy to small percentage of servers first

- Example: 10% of traffic to new version
- Monitor for errors
- Gradually increase if no issues

Complete CI/CD Pipeline Example:

1. Developer pushes code to GitHub

2. CodePipeline detects change

3. CodePipeline triggers CodeBuild

4. CodeBuild:

  - Downloads code

  - Installs dependencies

  - Runs tests

  - Creates artifact

  - Stores in S3

5. CodePipeline triggers CodeDeploy

6. CodeDeploy:

  - Gets artifact from S3

  - Deploys to EC2 instances

  - Runs health checks

7. CloudWatch monitors deployment

8. SNS sends success notification

Time from code push to production: 5-10 minutes (fully automated)

DevOps Benefits:

- Faster releases
- Automated testing
- Consistent deployments
- Easy rollbacks

- Reduced manual work
- Better quality code

---

9. AWS CONFIGURATION SERVICE (AWS CONFIG)

What it is: Service to assess, audit, and evaluate configurations of AWS resources.

Definition: Tracks how your AWS resources are configured and how they change over time.

What AWS Config Does:

1. Resource Inventory:

- Lists all AWS resources in your account
- EC2 instances, S3 buckets, Security groups, etc.
- Current configuration of each resource

2. Configuration History:

- Records configuration changes
- Who made the change
- When it was made
- What was changed

3. Compliance Monitoring:

- Check if resources follow rules
- Example: All S3 buckets must be encrypted
- Alert if rule violated

4. Resource Relationships:

- Shows how resources are connected
- Example: This EC2 is in this VPC, using this security group

---

AWS Config Rules:

What they are: Desired configuration settings for your resources.

Example rules:

Rule 1: S3 buckets must have encryption enabled

If S3 bucket created without encryption

AWS Config marks it as non-compliant

Sends notification

Rule 2: EC2 instances must have tags

If EC2 instance created without required tags

AWS Config flags it

Can automatically remediate

Rule 3: Security groups must not allow 0.0.0.0/0 on port 22

If security group has SSH open to world

AWS Config alerts

Security risk identified

DevOps Use Cases:

Use Case 1: Security compliance

Requirement: All databases must be encrypted

AWS Config rule: Check RDS encryption

Alert if non-compliant

Automatic remediation: Enable encryption

Use Case 2: Change tracking

Production issue occurred at 3 PM

Check AWS Config

See that security group changed at 2:55 PM

Identify who made change

Revert change

Use Case 3: Disaster recovery

Resource accidentally deleted

AWS Config has configuration history

Recreate resource with exact same configuration

Faster recovery

Use Case 4: Audit requirements

Auditor asks: Were all S3 buckets encrypted in 2024?

AWS Config provides historical data

Generate compliance report

Pass audit

---

10. BILLING AND COSTING

What it is: Tools to monitor and manage AWS costs.

Why important: AWS charges for usage. Without monitoring, bills can become very high unexpectedly.

AWS Billing Services:

a) AWS Cost Explorer:

What it is: Visualize and analyze your AWS costs.

Features:

- View costs by service
- View costs by time period
- Forecast future costs
- Identify cost trends
- Find cost optimization opportunities

Example:

Last month costs:

EC2: $500

S3: $50

RDS: $200

Lambda: $5

Total: $755

Forecast next month: $800

---

b) AWS Budgets:

What it is: Set custom cost and usage budgets.

Features:

- Set budget limits
- Get alerts when exceeding
- Track against budget
- Prevent overspending

Example:

Budget: $1000/month

Current spend: $850

Alert at 80% ($800)

Alert at 100% ($1000)

Email sent when thresholds reached

---

c) Cost Allocation Tags:

What they are: Labels you assign to resources to track costs.

Example:

Tags:

- Environment: Production

- Team: DevOps

- Project: Website

View costs by:

- Production environment: $2000

- DevOps team: $1500

- Website project: $800

---

d) AWS Cost and Usage Report:

What it is: Detailed breakdown of AWS usage and costs.

Features:

- Hourly or daily granularity
- All services covered
- Can export to S3
- Analyze with spreadsheet or analytics tools

Cost Optimization Strategies:

1. Right-sizing:

Problem: Using t2.large (2 CPU, 8 GB) but only using 20% CPU

Solution: Switch to t2.medium (2 CPU, 4 GB)

Savings: 50% cost reduction

2. Reserved Instances:

Problem: Running EC2 24/7, paying on-demand price

Solution: Buy Reserved Instance (1 or 3 year commitment)

Savings: Up to 75% discount

3. Spot Instances:

Problem: Need compute for non-critical tasks

Solution: Use Spot Instances (spare AWS capacity)

Savings: Up to 90% discount

Risk: Can be terminated with 2-minute warning

4. S3 Lifecycle Policies:

Problem: Storing old logs in S3 Standard (expensive)

Solution: Auto-move to S3 Glacier after 90 days

Savings: 95% storage cost reduction

5. Delete unused resources:

Find and delete:

- Stopped EC2 instances (still paying for storage)

- Unattached EBS volumes

- Old snapshots

- Unused Elastic IPs

DevOps Use Cases:

Use Case 1: Department cost tracking

Tag all resources with Department tag

Dev team resources: Department=Development

QA team resources: Department=QA

View costs per department

Charge back to departments

Use Case 2: Cost alerting

Set budget: $5000/month

Alert at $4000 (80%)

DevOps team investigates

Finds unused resources

Deletes them before exceeding budget

Use Case 3: Scheduled shutdowns

Development environments only needed 9 AM - 6 PM

Lambda function:

- Stops EC2 instances at 6 PM

- Starts at 9 AM

Saves: 13 hours/day = 54% cost reduction

---

11. AWS KMS (KEY MANAGEMENT SERVICE)

What it is: Service to create and manage encryption keys.

Definition: Centralized control over cryptographic keys used to protect data.

Why KMS is Important:

Without encryption:

Data stored in plain text

If leaked, anyone can read

Compliance violations

Security breach

With KMS encryption:

Data encrypted

If leaked, unreadable without key

Meet compliance requirements

Data secure

What KMS Does:

1. Create Encryption Keys:

- Generate cryptographic keys
- Master keys for encryption

2. Manage Keys:

- Rotate keys automatically
- Disable/enable keys
- Delete keys

3. Control Access:

- Who can use which keys
- Audit key usage

4. Encrypt/Decrypt:

- Encrypt data with keys
- Decrypt data with keys

KMS Key Types:

Customer Managed Keys (CMK):

- You create and manage
- Full control
- Can rotate, delete, disable

AWS Managed Keys:

- Created by AWS services
- AWS manages for you
- Automatic rotation

How KMS Works:

Encrypting data:

1. Application has data to encrypt
2. Calls KMS with key ID
3. KMS encrypts data
4. Returns encrypted data
5. Application stores encrypted data

Decrypting data:

1. Application has encrypted data
2. Calls KMS with encrypted data
3. KMS checks permissions
4. KMS decrypts data
5. Returns plain text data

DevOps Use Cases:

Use Case 1: Encrypt EBS volumes

Create EC2 instance

Enable EBS encryption with KMS key

All data on disk encrypted

If disk stolen, data unreadable

Use Case 2: Encrypt S3 buckets

Store application logs in S3

Enable S3 encryption with KMS

Logs encrypted at rest

Meet compliance requirements

Use Case 3: Encrypt database

RDS database with sensitive data

Enable encryption with KMS

Database backups also encrypted

Snapshots encrypted

Use Case 4: Secrets management

Application needs database password

Password encrypted with KMS

Stored in Systems Manager Parameter Store

Application retrieves and decrypts

Password never in plain text

---

12. CLOUDTRAIL (AUDIT AND LOGGING SERVICE)

What it is: Records all API calls and actions in your AWS account.

Definition: Audit trail of everything that happens in AWS.

What CloudTrail Records:

1. API Calls:

- Who made the call
- When it was made
- Where it came from (IP address)
- What was the call (CreateInstance, DeleteBucket)
- What was the result (Success or Failure)

2. Console Actions:

- User logged in
- User clicked button
- Configuration changed

3. AWS Service Actions:

- Auto Scaling launched instance
- Lambda function executed
- CloudWatch triggered alarm

CloudTrail Event Example:


```
{
  "eventTime": "2024-11-07T10:30:45Z",
  "eventName": "DeleteBucket",
  "userIdentity": {
    "userName": "john-developer"
  },
  "sourceIPAddress": "203.0.113.1",
  "requestParameters": {
    "bucketName": "production-data"
  },
  "responseElements": {
    "success": true
```

}

}

This shows: User john-developer deleted bucket production-data at 10:30 AM from IP 203.0.113.1

DevOps Use Cases:

Use Case 1: Security investigation

Problem: Suspicious activity detected

Action: Check CloudTrail logs

Find: Unauthorized user accessed resources

Result: Revoke access, investigate breach

Use Case 2: Compliance audit

Auditor: "Show me all S3 bucket deletions in 2024"

Action: Query CloudTrail logs

Result: Generate report with all deletions

Compliance: Passed

Use Case 3: Troubleshooting

Problem: Resource deleted accidentally

Action: Check CloudTrail

Find: Who deleted it and when

Result: Recreate resource, prevent future accidents

Use Case 4: Change tracking

Problem: Configuration changed, system broken

Action: Check CloudTrail for recent changes

Find: Security group rule modified at 2 PM

Result: Revert change, system restored

CloudTrail Best Practices:

1. Enable CloudTrail in all regions
2. Store logs in S3 bucket
3. Enable log file integrity validation
4. Set up alerts for sensitive actions
5. Regularly review logs
6. Keep logs for compliance period

---

13. AWS EKS (ELASTIC KUBERNETES SERVICE)

What it is: Managed Kubernetes service on AWS.

Definition: AWS runs Kubernetes for you. You just deploy your applications.

What is Kubernetes: Container orchestration platform that automates deployment, scaling, and management of containerized applications.

Problem without EKS:

1. Install Kubernetes manually (complex)

2. Configure master nodes

3. Configure worker nodes

4. Set up networking

5. Configure security

6. Set up monitoring

7. Manage updates

8. Handle failures

9. Scale manually

All this takes weeks and expertise

Solution with EKS:

1. Click "Create EKS cluster"

2. AWS sets up everything

3. Deploy your applications

Everything ready in 15 minutes

EKS Architecture:

Control Plane (Master):

- Managed by AWS
- Runs Kubernetes API server
- Makes decisions about cluster
- You don't manage this

Worker Nodes:

- EC2 instances
- Run your applications
- You can manage these

Pods:

- Containers running your application
- Smallest deployable unit

How EKS Works:

1. Create EKS cluster
2. EKS creates Kubernetes control plane
3. Add worker nodes (EC2 instances)
4. Deploy application as containers
5. Kubernetes schedules containers on nodes
6. Kubernetes monitors health
7. Kubernetes auto-scales
8. Kubernetes auto-heals

DevOps Use Cases:

Use Case 1: Microservices architecture

Application with 20 microservices

Each microservice in Docker container

Deploy all to EKS cluster

Kubernetes manages:

- Load balancing

- Service discovery

- Health checks

- Auto-scaling

Use Case 2: CI/CD with containers

Developer pushes code

CodePipeline builds Docker image

Image pushed to ECR (AWS container registry)

CodeDeploy updates EKS deployment

New version rolled out gradually

Zero downtime deployment

Use Case 3: Auto-scaling applications

Traffic increases during day

Kubernetes detects high CPU

Automatically scales pods up

Traffic decreases at night

Kubernetes scales pods down

Optimized cost

---

14. FARGATE AND ECS

What they are: AWS container services.

---

a) ECS (ELASTIC CONTAINER SERVICE)

What it is: AWS proprietary container orchestration service.

Definition: Run Docker containers on AWS without managing servers.

How it works:

1. Create ECS cluster

2. Define task (which containers to run)

3. Deploy task

4. AWS runs containers

ECS vs EKS:

- ECS: AWS proprietary, simpler, AWS-specific
- EKS: Standard Kubernetes, portable, more complex

---

b) AWS FARGATE

What it is: Serverless compute engine for containers.

Definition: Run containers without managing servers at all.

Traditional container approach:

1. Create EC2 instances

2. Install Docker

3. Join to ECS/EKS cluster

4. Manage instances

5. Patch and update

Fargate approach:

1. Define container

2. Fargate runs it

3. No servers to manage

How Fargate Works:

1. Create container image

2. Define task (CPU, memory needed)

3. Launch on Fargate

4. AWS provisions compute

5. Container runs

6. AWS manages everything

7. You pay per second of usage

ECS vs Fargate:

ECS with EC2:

- You manage EC2 instances
- More control
- Can be cheaper at scale
- More operational overhead

ECS with Fargate:

- No servers to manage
- Completely serverless
- Pay per task
- Less operational overhead
- Slightly more expensive

DevOps Use Cases:

Use Case 1: Batch processing

Need to process 1000 files

Fargate task for each file

All processed in parallel

Tasks terminate when done

No idle servers

Use Case 2: API backend

Containerized API application

Run on Fargate

Auto-scales based on traffic

Pay only when requests come

Serverless containers

Use Case 3: Scheduled jobs

Need to run report generation daily

Fargate task at 2 AM

Runs report

Sends email

Task terminates

Pay for 5 minutes of execution

---

15. ELK (ELASTICSEARCH, LOGSTASH, KIBANA)

What it is: Log management and analytics stack.

Note: ELK is not an AWS service, but AWS offers managed version called Amazon Elasticsearch Service (now Amazon OpenSearch Service).

ELK Components:

E = Elasticsearch:

- Search and analytics engine
- Stores log data
- Fast searching

L = Logstash:

- Log collection and processing
- Collects logs from multiple sources
- Transforms and sends to Elasticsearch

K = Kibana:

- Visualization tool
- Create dashboards

- Search logs
- Analyze data

How ELK Works:

1. Applications generate logs

2. Logstash collects logs

3. Logstash processes logs

4. Logs sent to Elasticsearch

5. Elasticsearch stores and indexes

6. Kibana visualizes data

7. DevOps team searches and analyzes

AWS OpenSearch (Managed ELK):

What AWS provides:

- Fully managed Elasticsearch
- Automatic backups
- Scaling
- Security
- No server management

DevOps Use Cases:

Use Case 1: Centralized logging

50 microservices running

All send logs to Logstash

Logstash sends to Elasticsearch

Search all logs from Kibana

Find errors across all services

Debug faster

Use Case 2: Application monitoring

Track application metrics

Response times

Error rates

User activity

Visualize in Kibana dashboards

Real-time monitoring

Use Case 3: Security monitoring

Collect security logs

Failed login attempts

Unauthorized access

Suspicious activities

Alert on patterns

Improve security

Use Case 4: Business analytics

Track business metrics

User signups

Purchases

Feature usage

Create business dashboards

Data-driven decisions

SUMMARY OF 15 AWS SERVICES FOR DEVOPS

1. EC2: Virtual servers in cloud

2. VPC: Private network (subnets, security groups, CIDR)

3. EBS: Storage volumes for EC2

4. S3: Object storage for files

5. IAM: Security and access control

6. CloudWatch: Monitoring and logging

7. Lambda: Serverless compute

8. Code Services: CI/CD (CodePipeline, CodeBuild, CodeDeploy)

9. AWS Config: Resource configuration tracking

10. Billing: Cost monitoring and optimization

11. KMS: Encryption key management

12. CloudTrail: Audit trail of API calls

13. EKS: Managed Kubernetes

14. Fargate/ECS: Container services

15. ELK: Log management and analytics

\\\\\\\