aws Training Module

01

AWS Foundations & Developer Setup

Focus: Governance, Security, and Automation First

Today's Agenda

- AWS Cloud & Developer Environment
- AWS CLI Deep Dive & Config
- IAM Basics (Users, Groups, Policies)
- Shared Responsibility Model

Learning Outcomes

- Master AWS Shared Responsibility
- Configure CLI securely (SSO/MFA)
- Automate IAM user provisioning
- Understand Identity Principals

Prerequisites: AWS Account | Terminal Access | Admin Permissions

AWS Professional

10-Day Comprehensive Program

REF: AWS-PRO-2026

AWS Overview for Enterprises

Module 01: Foundations

Global Infrastructure

AWS spans **33 Geographic Regions** and **105 Availability Zones (AZs)**. Each Region is a physical location with multiple AZs designed for high availability and fault tolerance.

- ✓ AZs are isolated locations within a region with independent power/cooling
- ✓ Edge Locations (400+) deliver content via CloudFront with low latency

Core Services & Building Blocks

Comprehensive suite of IaaS, PaaS, and SaaS offerings categorized by functionality.

- Compute (EC2, Lambda)
- Storage (S3, EBS, EFS)
- Database (RDS, DynamoDB)
- Networking (VPC, Route53)

Well-Architected Framework

Six pillars ensuring systems are secure, high-performing, resilient, and efficient: Operational Excellence, Security, Reliability, Performance Efficiency, Cost Optimization, Sustainability.

Enterprise Governance

AWS Organizations

Central governance and billing for multiple AWS accounts. Enables Service Control Policies (SCPs) to enforce guardrails across the organization.

Control Tower

Automates the setup of a multi-account environment (Landing Zone) based on best practices blueprints.

Compliance Programs

AWS supports 143 security standards and compliance certifications including PCI-DSS, HIPAA/HITECH, FedRAMP, GDPR, FIPS 140-2, and NIST 800-171.

"Security is job zero." - AWS Core Philosophy

© 2026 AWS Training | Confidential & Proprietary

1 Account Creation & Verification

Navigate to aws.amazon.com/free. You will need a valid email, credit/debit card, and phone number.

AWS will charge a temporary $1.00 hold to verify identity. Ensure your card allows international transactions if applicable.

2 Secure Root User (MFA)

Login as Root. Go to **IAM Identity Center** or Security Credentials.

Menu > Security Credentials > Multi-factor authentication (MFA) > Assign MFA

Use Google Authenticator or Authy app on your mobile device to scan the QR code.

3 Billing Preferences & Alerts

Enable PDF invoices and Free Tier usage alerts to get notified via email when limits are approached.

✓ Receive PDF Invoice ✓ Free Tier Alerts

Verification Checklist

- ☐ Root account secured with MFA
- ☐ Billing currency set correctly
- ☐ $0.00 Budget created & active
- ☐ Region checks (Use us-east-1 for billing)

Common Pitfalls

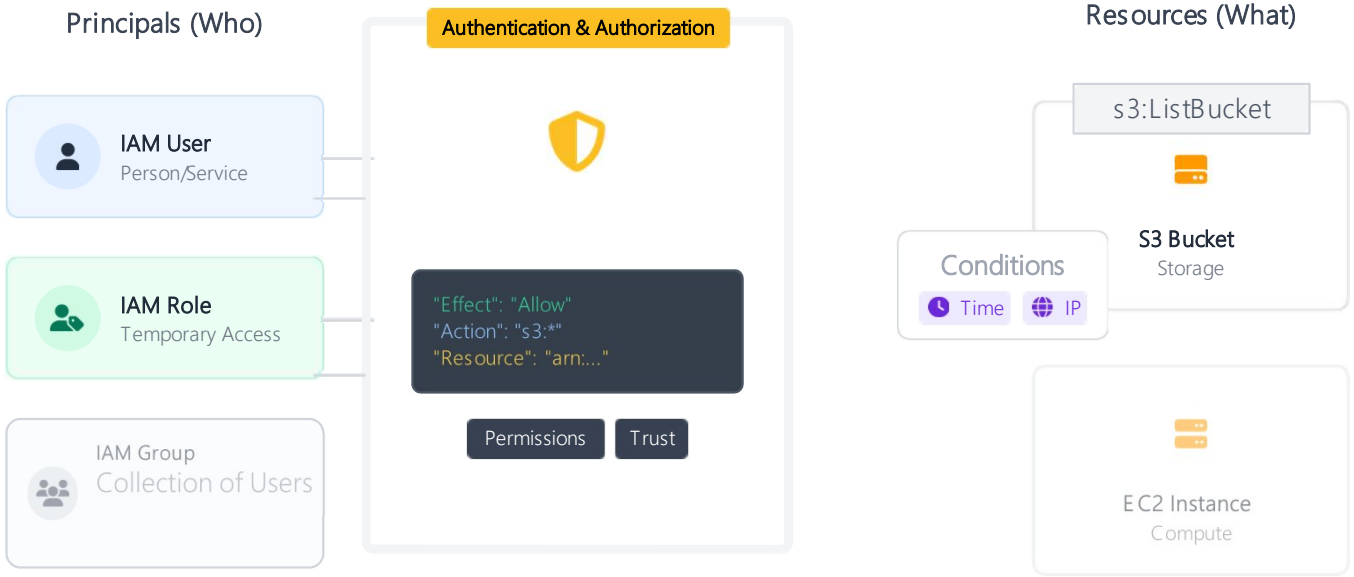
- **Support Plan:** Selecting "Developer" or "Business" support by mistake (costs $29+/mo). Stick to "Basic".
- **Region Blindness:** Resources created in the wrong region (e.g., Ohio vs N. Virginia) may be forgotten and billable.
- **Active Services:** Not terminating the NAT Gateway or Elastic IP after labs.

© 2026 AWS Training | Confidential & Proprietary

IAM Basics: Principals & Policies

Module 01: Foundations

IAM Authorization Flow



Key Concepts

Principals

An entity (User, Service, App) that is authenticated and allowed to make requests.

Policies (JSON)

Documents that define permissions.

- Identity-based (Attached to User/Role)
- Resource-based (Attached to S3/SQS)

Trust vs Permission

Trust Policy:
"Who can assume this role?"

```
Principal: {"AWS": "arn:aws:iam::123:user/Bob"}
```

Permission Policy:
"What can they do?"

```
Action: "s3:ListBucket"
```

IAM Policy JSON Examples

Module 01: Foundations

secure-policy.json

```
{
"Version" :"2012-10-17" ,
"Statement" :[
// Statement 1: Allow Read Access to Specific Bucket
{
"Sid" :"AllowScopedRead" ,
"Effect" :"Allow",
"Action" :[
"s3:ListBucket" ,
"s3:GetObject"
],
"Resource" :[
"arn:aws:s3:::finance-reports" ,// Bucket Level
"arn:aws:s3:::finance-reports/*" // Object Level
]
},
// Statement 2: Explicit Deny for Non-SSL Traffic
{
"Sid" :"EnforceTLS" ,
"Effect" :"Deny",
"Action" :"s3:*",
"Resource" :"arn:aws:s3:::finance-reports/*" ,
"Condition" :{
```

Policy Anatomy

1. **Version**
   Always use 2012-10-17 es.
2. **Sid (Statement ID)**
   Optional ID. Useful for tracking & organization.
3. **Effect**
   Allow or Deny. *Implicit Deny* is default.
4. **Condition**
   "When does this apply?" e.g., IP range, MFA, SSL.

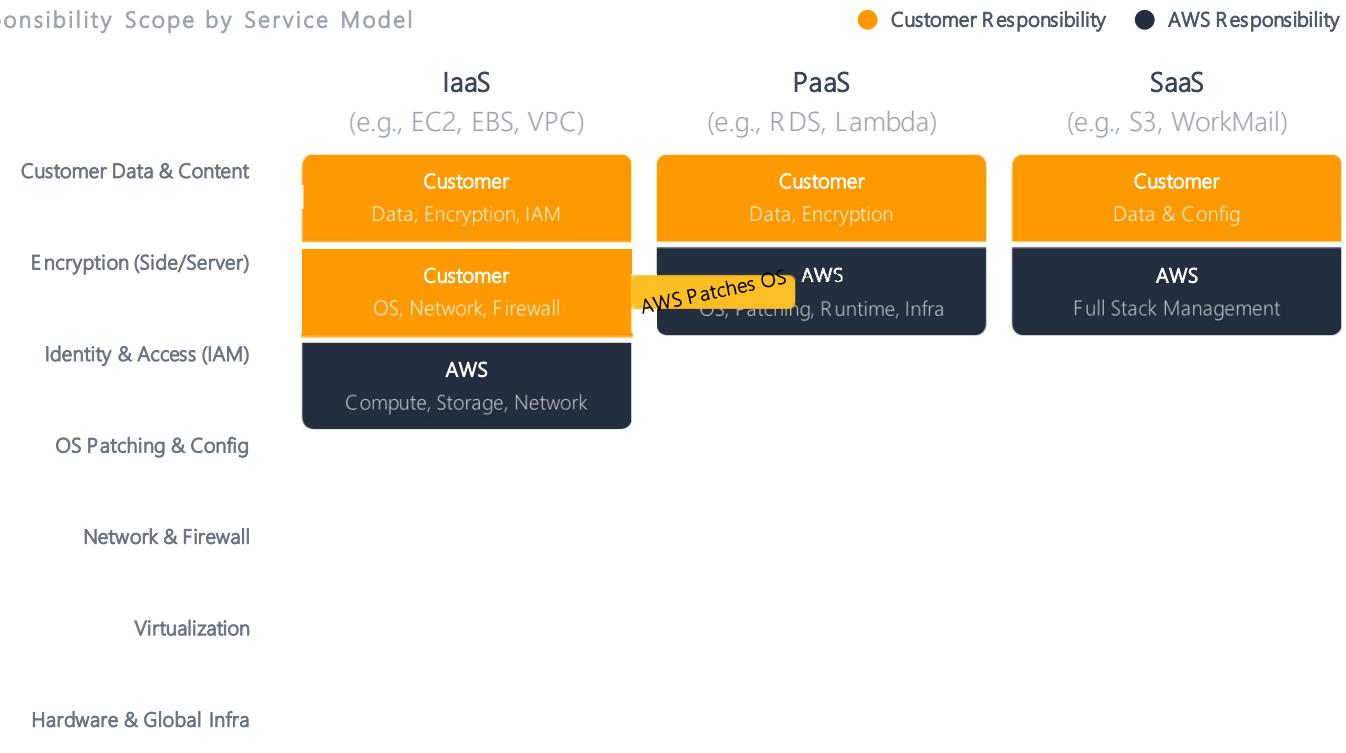
Pro Tips

- **Explicit Deny** always overrides Allow. Use it for guardrails.
- Avoid "Action": "*" in production. Be specific (Least Privilege).
- Use **Policy Variables** like ${aws:username} for dynamic scaling.

Shared Responsibility Model

Module 01: Foundations

Responsibility Scope by Service Model



Note: Data Classification and Client-side Encryption are ALWAYS the customer's responsibility, regardless of the service model.

Governance & Ops

Compliance Mapping

AWS manages security *of* the cloud (Physical, Network), reducing the audit scope for compliance (PCI-DSS, HIPAA).

Example: For RDS (PaaS), AWS handles OS patching compliance, customer handles DB user access compliance.

OS Patch Management

IaaS: You control the update schedule (yum update).

PaaS: AWS applies minor version patches automatically during maintenance windows.

Encryption Keys

Customer owns the keys (KMS CMK) and decides who can use them. AWS manages the hardware

You cannot [illegible] responsibility for your Data or

Developer Tools: Console vs CLI vs SDK

Module 01: Foundations

Feature / Tool	AWS Console Web UI (GUI)	AWS CLI Command Line	AWS SDK Code Libraries
Speed & Automation	**Slow / Manual** Click-ops is prone to human error. Cannot be automated easily.	**Fast / Scriptable** Great for ad-hoc scripts and repeatable tasks.	**Fastest / Integrated** Embedded in app logic. High performance.
Repeatability	**Low** Difficult to reproduce exact steps across regions/accounts.	**High** Scripts can be version controlled and re-run.	**Very High** Part of application codebase and build process.
Learning Curve	**Low (Beginner)** Visual, intuitive discovery of features.	**Medium** Requires knowing commands and parameters.	**High (Developer)** Requires programming knowledge (Python, JS, etc.).
Auth & Security	**SSO / MFA / Password** Session based. Auto-logout.	**Access Keys / SSO** Keys stored locally. Risk of leakage if not managed.	**IAM Roles** Best practice. Uses instance metadata or env vars.
Primary Use Case	Initial exploration Billing & Cost Mgmt One-off diagnostics	CI/CD Pipelines Shell Automation Quick bulk operations	Application Logic Custom Tools Complex workflows

Console

CLI

SDK

Why Use AWS CLI? Enterprise Scenarios

Module 01: Foundations

Automation & Scale

Critical for managing large-scale environments where Console actions are inefficient.

- Rapidly configure baselines (VPC, IAM, Config) for new accounts without manual clicks.
- Iterate through thousands of resources (e.g., tagging instances, cleaning up S3 buckets) using shell loops.

DevOps & Integrity

Integrating AWS operations directly into development pipelines and verification workflows.

- Embed CLI commands in Jenkins, GitLab, or GitHub Actions for lightweight infrastructure deployment steps.
- Scripted verification to ensure resource configurations match the expected state (e.g., security group rules).

Repeatability & Reliability

Ensure consistent execution across environments.

- Write scripts that handle existence checks to safely run multiple times without side effects.

Power User Capabilities

Demo Scenarios

For architects and leads, the CLI is faster for "live demos" or rapid prototyping than writing full SDK code for one-off tests.

Shell Integration

Pipe output directly to tools like `jq` `grep` `awk`

```
aws s3 ls | grep "2024" | awk '{print $4}'
```

JMESPath Querying

Server-side filtering prevents fetching unnecessary data, reducing latency and cost.

```
--query 'Reservations[].Instances[].InstanceId'
```

© 2026 AWS Training | Confidential & Proprietary

Configure CLI: Access Keys, Region, Profiles

Module 01: Foundations

1 Standard Configuration vs. SSO

Choose your authentication method. SSO is recommended for enterprise environments.

Method A: Static Keys

`aws configure`

Requires Access Key ID & Secret Key

Method B: SSO (Recommended)

`aws configure sso`

Auto-rotates credentials via portal

2 Using Named Profiles

Isolate environments (Dev, Prod) by appending `--profile` to commands.

```
$ aws configure --profile prod-user
# Set region to us-west-2, output to json
```

Usage: ***aws s3 ls --profile prod-user***

3 Configuration Files Location

AWS CLI stores settings in two separate files in your home directory `~/.aws/`

config

Region, output format, sso_start_url

credentials

Access Key ID, Secret Access Key (Sensitive!)

Verification Checklist

- ☐ Verify Identity
  `aws sts get-caller-identity`
- ☐ Check Local Files
  `cat ~/.aws/credentials`
- ☐ List Buckets (Test Access)
  `aws s3 ls`

Common Pitfalls

- **Committing Secrets:** Never commit `.aws/credentials` or `.env` files to Git. Use `.gitignore`.
- **Region Mismatch:** CLI defaults to `us-east-1` if not set. Resources in Ohio (`us-east-2`) won't show up.
- **Hidden Env Vars:** Troubleshooting why profile settings aren't working? Check if `AWS_ACCESS_KEY_ID` is set in your shell.

©2026 AWS Training | Confidential & Proprietary

CLI Command Structure & Examples

Module 01: Foundations

```bash
# 1. General Syntax Pattern
# aws <service> <operation> [parameters]
aws s3 ls

# 2. Advanced Query: Filter Running Instances & Format Output
aws ec2 describe-instances \
--filters "Name=instance-state-name,Values=running"  \
--query "Reservations[*].Instances[*].[InstanceId, PublicIpAddress, Tags[?Key=='Name'].Value|[0]]"  \
--output table

# 3. Pagination & Limits
aws s3api list-objects-v2 \
--bucket enterprise-logs-archive  \
--max-items 50\
--starting-token "eyJOZXh0VG9rZW4iOiBudWxsfQ=="

# 4. Security: Using Specific Named Profiles
aws iam list-users \
--profile prod-admin \
--region us-west-2

# 5. Debugging: Dry Run
aws ec2 run-instances --dry-run --image-id ami-12345678
```

Command Anatomy

1. **Service**
   The AWS service namespace (e.g., `ec2`, `s3`)
2. **Operation**
   The API action to perform (e.g., `describe-instances`).
3. **Parameters**
   Options starting with `--` to filter or configure.

Pro Tips

- **Server-Side Filtering:** Use `--query` (JMESPath) to filter data on AWS side, reducing bandwidth.
- **Safe Switching:** Always use `--profile` or export `AWS_PROFILE` to prevent accidental prod commands.
- **Autocomplete:** Enable with `complete -C '/usr/bin/aws_completer' aws`.

Console vs CLI vs SDK: Decision Matrix

Technical Assessment Criteria

FACTOR	CONSOLE (GUI)	CLI (SHELL)	SDK (CODE)
Frequency	Ad-hoc / One-time	Recurring / Batch	Continuous / High-freq
Complexity	Simple / Visual	Moderate / Scripted	High / Logic-heavy
Audit Trail	CloudTrail (User)	CloudTrail (User/Role)	CloudTrail (App Role)
Scale	Single Resource	10s - 100s Resources	Unlimited / Parallel
Latency	High (Human speed)	Medium (Process overhead)	Low (Native integration)
Governance	Hard to enforce limits	Guardrails via wrappers	Embedded compliance logic
Dependency	Browser only	Python/Installer	Language Runtime
Error Handling	Visual Feedback	Exit Codes / Text	Try-Catch / Exceptions

Selection Logic

Scenario: Visual Verification

"I need to check why an EC2 instance isn't starting and look at screenshots."

Use Console

Scenario: Ops Automation

"I need to rotate keys for 50 IAM users and generate a report CSV."

Use CLI

Scenario: App Integration

"My web app needs to upload user avatars to S3 and generate presigned URLs."

Use SDK

Pro Tip: Start exploring with Console, automate ops with CLI, build products with SDK.

Enterprise Recommendation

For production environments, restrict **Console** access. Mandate **IaC (Terraform/CloudFormation)** for provisioning and **SDKs** for application logic to ensure reproducibility and auditability.

developer-setup.sh

```bash
# 1. PREFERRED: Configure SSO (Avoids long-lived static keys)
# Interactive setup for AWS IAM Identity Center
aws configure sso

# Example Prompts & Inputs:
# SSO session name (Recommended): my-sso
# SSO start URL: https://my-org.awsapps.com/start
# SSO region: us-east-1

# 2. Switch Contexts safely using Environment Variables
export AWS_PROFILE="dev-profile"
export AWS_REGION="us-west-2"

# 3. Verify Identity (Check you are not using Root!)
aws sts get-caller-identity
# Output should show: "Arn": "arn:aws:sts::...:assumed-role/..."

# 4. ANTI-PATTERN: DO NOT DO THIS
# Never hardcode secrets in scripts or export them manually
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE" # <-- BAD!
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

# 5. Handling Secrets for Apps
```

Core Principles

1. **SSO > Static Keys**
   Use Identity Center. Short-term creds auto-rotate.
2. **Least Privilege**
   Don't use Admin. Scope permissions to exact needs.
3. **MFA Everywhere**
   Enforce MFA for Console and CLI access.
4. **CloudTrail**
   Ensure all API actions are logged for audit.

Critical Warnings

- **NEVER** commit `.env` files or credentials to Git. Use `.gitignore`.
- Do not use the **Root User** for daily tasks. Lock it away.
- Use tools like `git-secrets`.

Mini Project: IAM User Automation

Module 01: Foundations

IAM Provisioner

45 Mins Intermediate

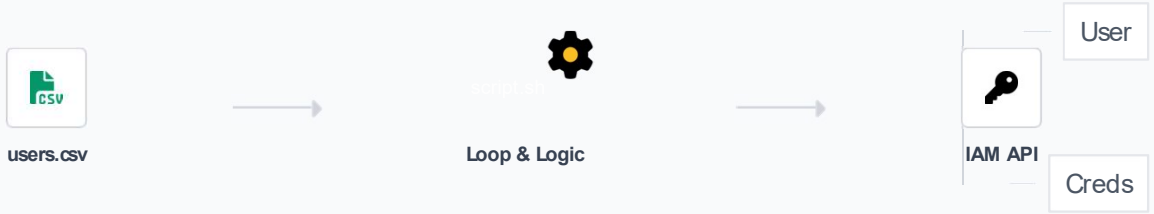
Create a robust shell script to onboard new employees by parsing a CSV file and provisioning their IAM resources automatically.

Bash/Python | AWS CLI | IAM | CSV

Objectives

- Parse a `users.csv` file containing names and departments.
- Create IAM Users idempotently (check if exists first).
- Assign users to groups (Devs, Ops) based on department.
- Create initial login profile with forced password change.

Logic Flow



provision_users.sh — Bash

```bash
#!/bin/bash
# Usage: ./provision_users.sh users.csv
# CSV Format: username,group
INPUT_FILE=$1
while IFS=, read -r user group; do
# 1. Check if user exists
 aws iam get-user --user-name "$user" &>/dev/null
if [ $? -eq 0 ]; then
 echo "User $user already exists. Skipping."
else
# 2. Create User
 aws iam create-user --user-name "$user"
# 3. Add to Group
 aws iam add-user-to-group --user-name "$user" --group-name "$group"
# 4. Create Login Profile
```

aws Training Module

02

AWS Professional

10-Day Comprehensive Program

REF: AWS-PRO-2026

Terraform Automation & KMS Resource Creation

Focus: Infrastructure as Code (IaC) & Security Automation

Today's Agenda

- IaC Concepts & Patterns
- Terraform Workflow (Init to Apply)
- Variables & Precedence Strategy
- AWS KMS Resource Creation
- Multi-Environment Delivery

Learning Outcomes

- Implement Declarative Provisioning
- Master Terraform State Management
- Automate Security Keys (KMS)
- Manage Multi-Env Deployments

Requirements Terraform CLI | VS Code / IDE | AWS CLI Configured

IaC Concepts: Why Terraform?

Module 02: Infrastructure Automation

Repeatability & Consistency

Eliminate manual configuration errors ("ClickOps") and ensure environments (Dev, Stage, Prod) are mathematically identical.

- **Idempotency:** Apply the same code multiple times without side effects.
- Disaster Recovery becomes a "terraform apply" operation.

Reviewability via VCS (GitOps)

Infrastructure is treated as software code, residing in version control systems like Git/CodeCommit.

- Enforce peer reviews (PRs) before infrastructure changes occur.
- Full audit trail of *who* changed *what* and *when* via Git history.

Modularity & Reusability

Encapsulate complex architecture into reusable modules to standardize patterns across the enterprise.

- Centralize best practices (e.g., "Hardened VPC Module").
- Write once, deploy everywhere (DRY principle).

Enterprise Implications

Drift Reduction

Detects unauthorized manual changes ("drift") by comparing the live environment state against the code. Terraform can report on or automatically revert these changes to maintain integrity.

Policy-as-Code

Integrate tools like **Sentinel** or **OPA** (Open Policy Agent) to enforce compliance rules automatically during the plan phase (e.g., "Prohibit S3 buckets without AES-256 encryption").

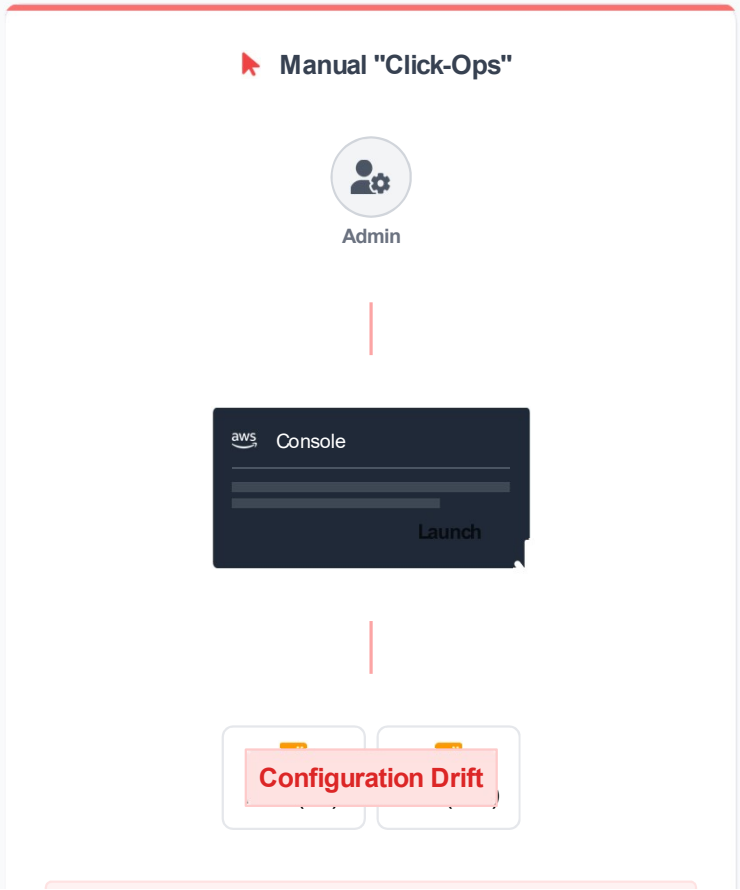
Lifecycle Management

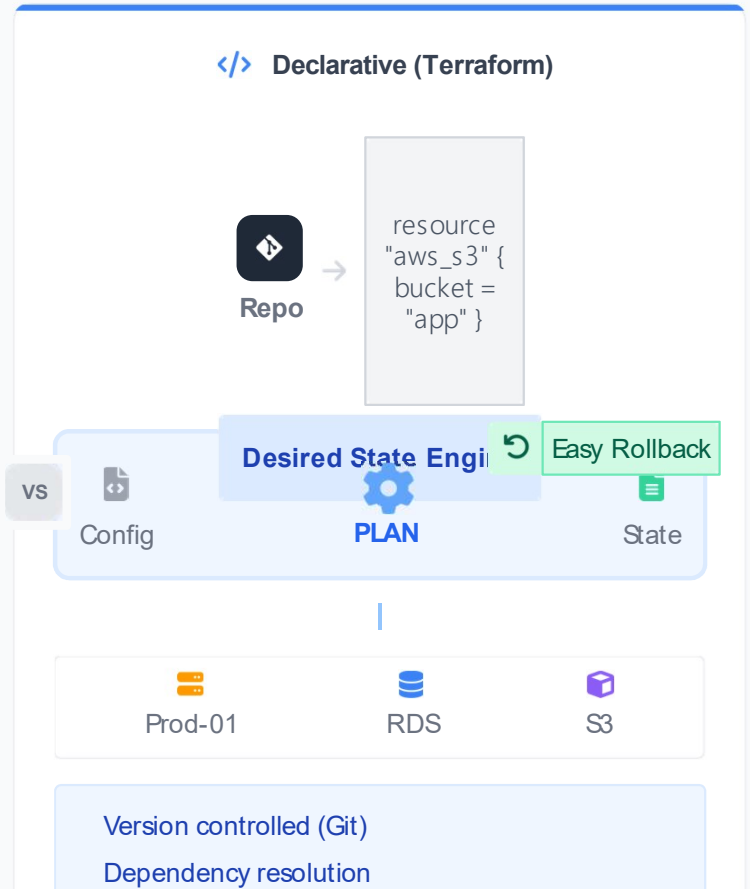
Manage the entire lifecycle of resources, from provisioning to updates and eventual decommissioning (destroy), ensuring no "zombie" resources remain.

© 2026 AWS Training | Confidential & Proprietary

Manual vs Declarative Provisioning

Module 02: Infrastructure as Code





Key Concepts

Desired State

You define **WHAT** the infrastructure should look like, not **HOW** to build it step-by-step.

Goal: 3 EC2 Instances Current: 1 EC2 Instance Action: Add 2 Instances

Action: Add 2 Instances

Dependency Graph

Terraform automatically calculates the order of operations based on resource references.

e.g., VPC must exist before Subnet, Subnet before EC2.

Rollback Story

In IaC, a "rollback" is just applying a previous version of the code committed to Git.

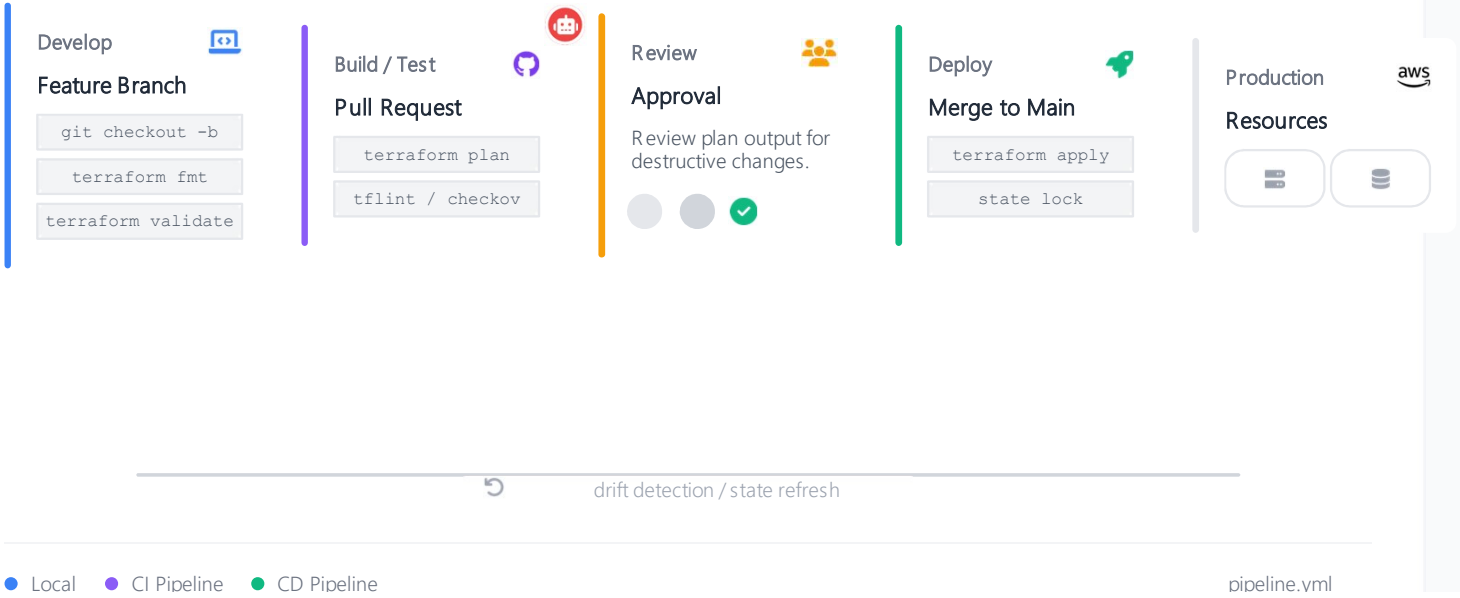
git revert → terraform apply

Process

Version-Controlled Deployments (Git Workflow)

Module 02: Infrastructure Automation

Infrastructure CI/CD Pipeline



Module Versioning

Production infrastructure should never use `latest` or `main` branch for modules.

```
module "vpc" {
source = "git::...vpc.git?ref=v1.2.0"

}
```

Use Semantic Versioning (Major.Minor.Patch)

CI Gate Checks

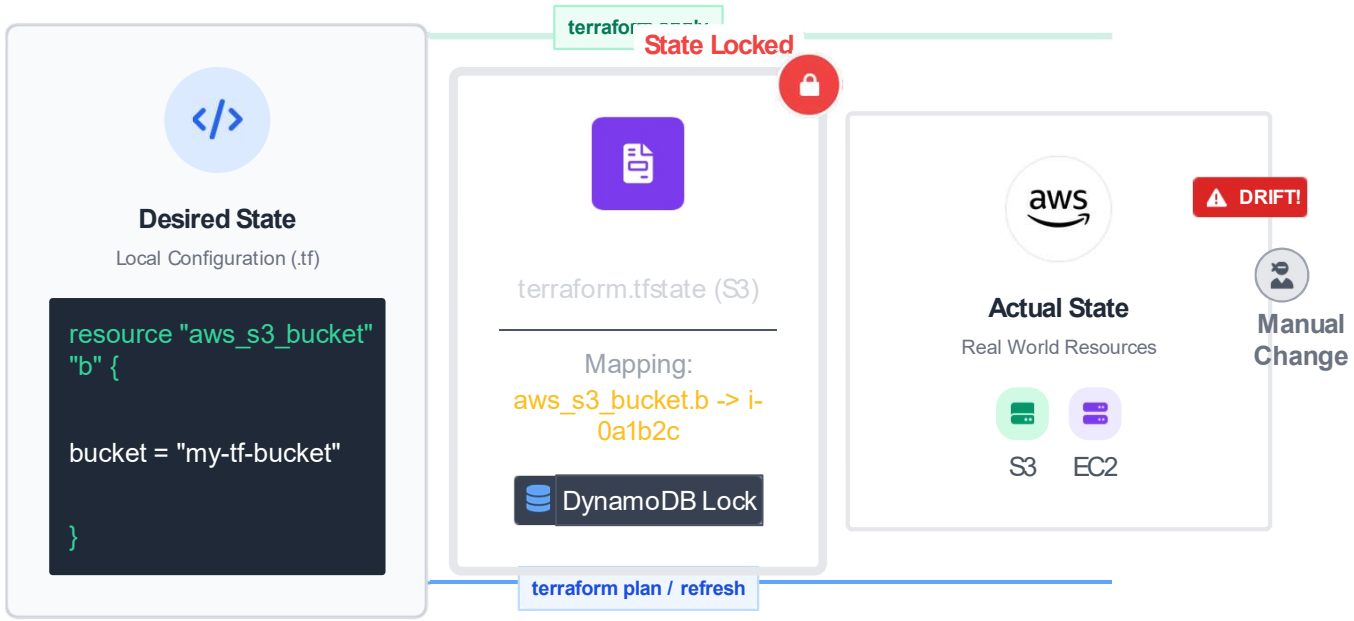
Terraform Fmt
Ensures consistent style (HCL standards).

Terraform Validate
Checks syntax and internal consistency.

Terraform Plan
Speculative execution to visualize changes in PR.

Security Note: Ensure the CI runner has minimal IAM permissions (Least Privilege) to

Terraform State Reconciliation Loop



Concepts & commands

Reconciliation

Terraform compares **Desired State** (code) against **State File** and **Actual State** (Cloud) to generate a plan.

State Locking

Prevents concurrent operations (e.g., two developers applying at once) using a DynamoDB table. Essential for teams.

Drift Management

Occurs when resources are modified outside Terraform (ClickOps).

```
# Detect drift only
terraform plan -refresh-only

# Accept drift to state
terraform apply -refresh-only
```

Terraform Workflow Lifecycle

Module 02: Infrastructure Automation

Standard Lifecycle Operations

Initialize

`terraform init`

Prepares the working directory.

- Downloads Providers
- Installs Modules
- Configures Backend

Plan

`terraform plan`

Creates execution plan (Dry Run).

- Refreshes State
- Calculates Diff
- Preview Changes

Apply

`terraform apply`

Provisions infrastructure.

- Calls AWS APIs
- Updates tfstate
- Locks State

Destroy

`terraform destroy`

Removes all resources.

- Irreversible Action
- Updates tfstate

terraform.tfstate (Remote Backend: S3 + DynamoDB)

Read by Plan	Written by Apply

Workspaces

Isolate state files for different environments without changing code structure.

```
$ terraform workspace new dev
Created and switched to workspace "dev"!
```

default | dev | prod

Plan Output Legend

+ Create
Resource will be added.

~ Update in-place
Modification of existing resource.

- Destroy
Resource will be removed.

Project Layout

root_module/

- **main.tf** — Resources & Modules
- **variables.tf** — Input Declarations
- **outputs.tf** — Return Values
- **versions.tf** — Constraints & Plugins

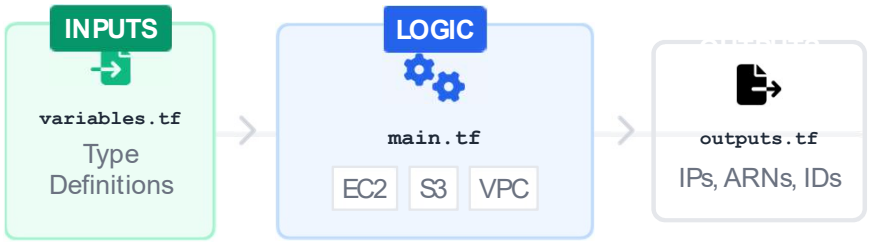
modules/

- vpc/
- security/

Functional Workflow

providers.tf
Configures AWS region, aliases, and authentication methods.

versions.tf
Pins Terraform core version and provider versions (e.g., aws ~> 5.0).



Remote Modules
Registry / Git

```
source = "..."
```

terraform.tfstate
Managed Automatically

Standard Pattern

The Root Module

```
terraform apply
```

root).

Key Files

`main.tf` — Primary entry point. Contains resource definitions and module blocks.

`variables.tf` — Declarations for input variables. Defines types, descriptions, and defaults.

`outputs.tf` — Values returned to the CLI or passed to other resources/modules.

Best Practices

- Split complex `main.tf` into logical files (e.g., `network.tf`, `storage.tf`)
- Always pin provider versions in `versions.tf`

Terraform Variables: Types with Examples

Module 02: Infrastructure as Code

variables.tf

```hcl
variable "aws_region" {
type=string
default ="us-east-1"
description ="Target deployment region"
}

variable "instance_count" {
type=number
default =2
validation {
condition =var.instance_count >0  >0
error_message ="Instance count must be at least 1."
}
}

variable "enable_monitoring" {
type=bool
default =true
}

variable "resource_tags" {
type=map(string)
default ={
```

Type System

P **Primitives**
Simple values: `string` `number` `bool`

C **Collections**
Grouped values: `list` `map` `set`

S **Structural**
Complex types: `object` `tuple`

Best Practices

- Always define **type** constraints to catch errors early during *plan*.
- Use **validation** blocks for custom rules (e.g., enforce instance sizes).
- Mark passwords/keys as **sensitive = true** to hide from logs.

Terraform Variable Precedence

Module 02: Infrastructure as Code

Precedence Hierarchy

Scenarios & Patterns

Secrets Handling

Never commit terraform.tfvars to Git.

Best Practice: Use Env Variables (TF_VAR_db_pass) or fetch from AWS Secrets Manager dynamically.

Workspaces

When using Terraform Workspaces (dev, stage, prod), Terraform looks for specific variable definitions.

```
terraform plan -var-file="prod.tfvars"
```

Override Files

Files named *_override.tf d override portions of the configuration, not just variables. Use sparingly for debugging.

Source Commit?

AWS KMS Resource Creation

Module 02: Infrastructure as Code

main.tf

```hcl
resource "aws_kms_key" "app_key" {
description ="KMS key for App Data encryption"
deletion_window_in_days =10
enable_key_rotation =true

policy=jsonencode ({
Version ="2012-10-17"
Statement =[{
Sid="Enable IAM User Permissions"
Effect ="Allow"
Principal ={AWS="arn:aws:iam::${var.account_id}:root"  }
Action ="kms:*"
Resource ="*"
}]
})
}

# Create an alias for easier reference
resource "aws_kms_alias" "app_key_alias" {
name="alias/app-data-key"
target_key_id =aws_kms_key .app_key .key_id
}

```

Resource Components

1. **enable_key_rotation**
   Essential for compliance. Rotates backing key material annually automatically.
2. **jsonencode()**
   Terraform function to render the policy. Cleaner than heredoc syntax.
3. **aws_kms_alias**
   Creates a display name (e.g., alias/my-key) for easier CLI/SDK usage.

Security Pro Tips

- **Root Access:** Always allow the root user in the key policy to avoid creating an unmanageable key.
- **Deletion Window:** Default is 30 days. Set shorter (e.g., 7 days) for non-prod environments.
- **Outputs:** Output the arn or key_id so dependent modules (like RDS/S3) can reference it.

Hands-on: Provision AWS Resources with Terraform

Module 02: Infrastructure Automation

Lab Details

Est. Time: 30 min

Level: Intermediate

Objectives

- Configure S3 + DynamoDB Backend for state locking
- Run full lifecycle: init plan apply
- Simulate & Resolve state lock conflict

Prerequisites

- Terraform v1.5+
- AWS CLI v2

1 Configure Remote Backend (S3 & DynamoDB)

backend.tf

```
terraform { backend "s3" { bucket = "my-terraform-state-prod"
```

Why Locking?

DynamoDB locking prevents concurrent operations (e.g., two developers applying at the same time) which could corrupt the state file.

Pro Tip:

Ensure the DynamoDB table has a partition key named (String) LockID

2 Initialize, Plan, & Apply

bash - zsh

```
terraform init
Initializing the backend...

Terraform has been successfully initialized!

terraform plan -out=tfplan
Plan: 3 to add, 0 to change, 0 to destroy.

Saved the plan to: tfplan

terraform apply "tfplan"
aws_kms_key.example: Creating...
```

©2026 AWS Training | Confidential & Proprietary

Terraform Workspaces

60 Mins Advanced

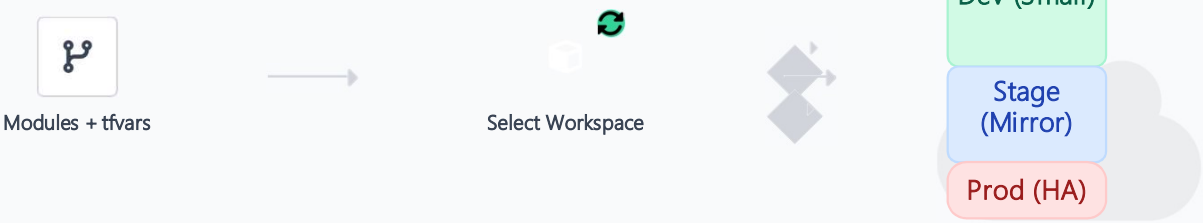
Architect a scalable infrastructure pipeline using Terraform Workspaces to deploy distinct Dev, Staging, and Production environments from a single codebase.

Terraform | Workspaces | S3 Backend | CI/CD

Objectives

- Configure remote state in S3 with DynamoDB locking.
- Create shared modules for VPC and EC2 resources.
- Implement `.tfvars` for environment differences.
- Deploy concurrent environments using Workspaces.

Deployment Flow



Project Structure & Logic

```
root/
├── modules/
└── vpc/
├── main.tf
├── outputs.tf
├── dev.tfvars
├── prod.tfvars
└── backend.tf
```

HCL

```hcl
# main.tf: Dynamic instance count based on workspace
locals {
 env_name = terraform.workspace
instance_counts = {
 default = 1
 prod = 3
}
}
module"app_cluster" {
 source = "./modules/ec2"
 count = lookup(local.instance_counts, local.env_name, 1)
tags = {
Environment = local.env_name
}
```

Key Takeaways & Best Practices

Module 02: Infrastructure Automation

Core Concepts & Best Practices

State Security

Never commit state files to Git. Use Remote Backends (S3) with KMS encryption and DynamoDB locking.

```
backend "s3" { encrypt = true }
```

Version Pinning

Pin provider and module versions to prevent breaking changes in production pipelines.

```
version = "~> 3.0"
```

Drift Management

Avoid "ClickOps". Manual changes in the Console cause drift. Run regularly to detect divergences.

terraform plan

⚠ Console changes will be overwritten

Modular Design

Don't build monoliths. Break infrastructure into reusable modules (VPC, App, DB) with clear inputs/outputs.

```
source = "./modules/vpc"
```

Common Pitfalls

Avoid These Patterns

Hard-Coded Secrets

Never put AWS keys or DB passwords in plain text **.tf** files.

Fix: Use AWS Secrets Manager

Monolithic State Files

Managing all Prod resources in one state file makes **plan** slow and risky (high blast radius).

Fix: Split by Env / Layer

Unpinned "Latest" Versions

Using **latest** means your infra might break on a random Tuesday when a provider updates.

Fix: Use .tf.lock.hcl

"Infrastructure as Code is only as good as the discipline maintaining it."

©2026 AWS Training | Confidential & Proprietary

aws Training Module

03

AWS SDK & S3 Deep Dive

Focus: Infrastructure Automation & Object Storage at Scale

Today's Agenda

- AWS SDK Automation (Boto3/Node.js)
- S3 Storage Classes & Intelligent Tiering
- Encryption, Lifecycles & Security
- Mini Project: Automated S3 Backup

Learning Outcomes

- Script resource creation programmatically
- Architect cost-optimized storage tiers
- Implement robust data security at rest
- Automate lifecycle management policies

Prerequisites AWS CLI Configured | Python/Node.js | IDE Setup

AWS Professional

10-Day Comprehensive Program

REF: AWS-PRO-2026

Core Components: Clients, Resources & Paginators

The SDK provides low-level 1:1 API mappings (Clients) and high-level object-oriented abstractions (Resources, e.g., in Boto3).

- **Paginators:** Automatically handle m `NextToken` rate over thousands of resources without manual loop logic.
- **Waiters:** Poll resource states (e.g.,) b `instance_running` il a desired state is reached.

Resilience: Retries & Timeouts

SDKs implement built-in retry logic for throttled requests (HTTP 429) and server errors (HTTP 5xx).

- Exponential Backoff
- Jitter (Randomness)
- Connect Timeouts
- Read Timeouts

Default Credential Provider Chain

SDKs automatically look for credentials in a specific order, enabling seamless transitions from local dev to production.

```
1. Environment Vars (AWS_ACCESS_KEY_ID) 2. Shared Config (~/.aws/config, SSO) 3. Container Provider (ECS
```

SDK Best Practices

Reuse Client Objects

Initializing a client is expensive. Create clients once (e.g., outside the Lambda handler) to reuse TCP connections (Keep-Alive).

Use Bounded Parallelism

When processing S3 objects or DynamoDB items, use tools like `Promise.all()` (Node) or `ThreadPoolExecutor` (Python) but respect API rate limits.

Specific Error Handling

Catch specific exceptions (e.g., `ClientError`, `ProvisionedThroughputExceeded`) rather than generic catch-all blocks to handle retries correctly.

© 2026 AWS Training | Confidential & Proprietary

Security Considerations for SDK Usage

Module 03: AWS SDK & S3 Deep Dive

IAM Scoping & Least Privilege

Never hardcode credentials. Use the default credential provider chain to leverage IAM Roles (Instance Profiles/IRSA).

- **Scope Permissions:** Grant access only to specific resources (ARNs) and actions necessary for the task.
- **Condition Keys:** Enforce aws:SourceIp, aws:RequestedRegion, or aws:PrincipalOrgID.

Secret Loading & KMS Integration

Use AWS Secrets Manager or SSM Parameter Store for configuration, not environment variables.

- **Envelope Encryption:** Use KMS for client-side encryption before data leaves the application.
- **TLS Enforcement:** Ensure SDK clients are configured to use TLS 1.2+ for all API calls.

Resilience: Timeouts & Retries

Configure clients to fail fast and retry intelligently to prevent resource exhaustion.

- **Exponential Backoff + Jitter:** Prevent "thundering herd" problems by randomizing retry intervals.
- **Connection Timeouts:** Default timeouts are often too long. Tune connect_timeout and read_timeout.

Resilience Patterns

Exponential Jitter Formula

```
Wait = min(Cap, Base * 2^Attempt) + Random
```

Adding randomness (jitter) desynchronizes client retries, flattening traffic spikes during recovery.

Throttling Exception Handling

SDKs handle 5xx errors automatically, but ProvisionedThroughputExceededException (DynamoDB) or RequestLimitExceeded (EC2) require custom retry logic in high-throughput apps.

Common Pitfall: Default Config

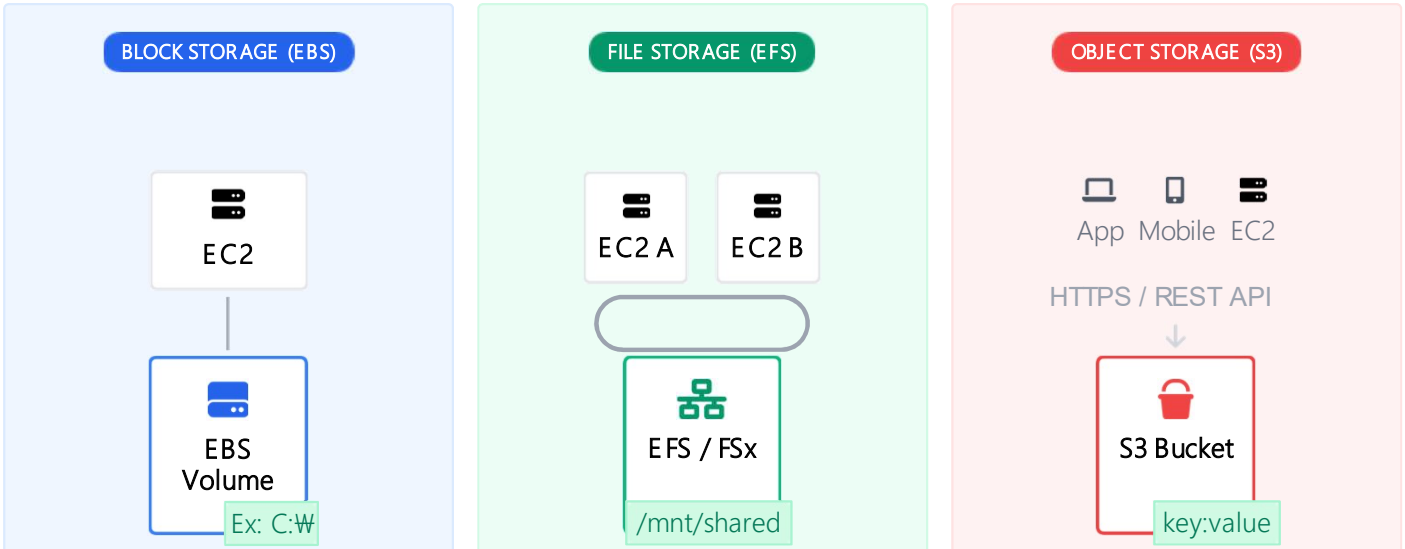
Default SDK HTTP clients often have infinite or very long timeouts. This can hang threads indefinitely if a downstream service stalls. Always set explicit timeouts.

© 2026 AWS Training | Confidential & Proprietary

Storage Models: Object vs Block vs File

Module 03: S3 Deep Dive

Storage Architecture Comparison



Characteristics

- Lowest Latency
- Single Attach (Mostly)
- Boot Volumes / DBs

Characteristics

- Shared Access
- Hierarchical
- CMS / Home Dirs

Characteristics

- Infinite Scale
- Metadata / Flat
- Web / Media / Archive

Decision Guide

When to use EBS (Block)

Think "Hard Drive". Use when you need a file system for an OS, database engine (RDS/MySQL), or low-latency disk operations.

Access: Single EC2 instance (usually)

When to use EFS (File)

Think "Network Share". Use for content management systems, shared code repositories, or home directories across a fleet.

Access: Thousands of EC2s concurrently

When to use S3 (Object)

Think "Internet Storage". Use for media assets, backups, data lakes, and static website hosting.

Access: Via API / HTTP (Anywhere)

S3 Storage Classes Deep Dive

Module 03: Storage Management

Metric	Standard Frequent Access	Intelligent Auto-Tiering	Standard-IA Infrequent	Glacier Flexible	Deep Archive Compliance
Use Case	Active Data	Unknown Patterns	Long-lived Data	Archive	Cold Archive
Latency	Milliseconds	Milliseconds	Milliseconds	Mins - Hours	12 - 48 Hrs
Min Duration	None	None*	30 Days	90 Days	180 Days
Retrieval Fee	None	None	per GB	per GB	per GB
Storage Cost	$$$$$ Highest	$$$$$ Auto-Optimized	$$$$$ ~50% of Std	$$$$$ ~20% of Std	$$$$$ Lowest (~5%)

Performance First

Use **Standard** for hot data. Use **Intelligent-Tiering** if access patterns are unknown—it automatically moves objects between frequent and infrequent access tiers without performance impact or operational overhead.

Cost Optimization

Standard-IA saves money for data accessed < 1/month but requires instant access. Be careful with small objects (< 128KB) as minimum object size charges apply for IA and Archive classes.

Archival Strategy

Glacier and **Deep Archive** offer the lowest cost but introduce retrieval times from minutes to 48 hours. Ensure your RTO (Recovery Time Objective) aligns with the retrieval latency.

© 2026 AWS Training | Confidential & Proprietary

S3 Encryption: SSE-S3 vs SSE-KMS

Module 03: Security & Storage

Envelope Encryption Process (Write)

Step 1
PUT Object
Plaintext Data
`Header: x-amz-server-side-encryption`

Step 2
Request Key
Generates Data Key
`Calls KMS (SSE-KMS) or Internal (SSE-S3)`

Step 3
Key Response
Master Key Encrypts Data Key
`Returns: Plain Key + Encrypted Key`

Step 4
Encrypt Data
Uses Plain Data Key
`AES-256 Encryption in Memory`

Step 5
Persist
Discard Plain Key
Encrypted Data
Encrypted Key

Envelope Encryption

Key Management

SSE-S3 (Default)
Keys managed by S3
Free of charge
Transparent rotation
Good for general data

SSE-KMS
Keys managed by KMS
Supports Audit (CloudTrail)
Separate permission control
Customer Managed Keys (CMK)

Performance & Limits

KMS API Quotas
SSE-KMS consumes KMS Request quota (e.g., 5,500 - 30,000 req/sec depending on region).

Bucket Keys
Enable "S3 Bucket Keys" to reduce KMS calls by 99% and lower costs.

Cost Impact: SSE-S3 is free. SSE-KMS charges for key storage ($1/mo) + API requests.

Bucket Policy & IAM Examples

Module 03: S3 Deep Dive

s3-conditions-policy.json

```
{
"Version" :"2012-10-17" ,
"Statement" :[
// Statement 1: Deny Access Outside AWS Organization
{
"Sid" :"DenyExternalOrgAccess" ,
"Effect" :"Deny",
"Principal" :"*",
"Action" :"s3:*",
"Resource" :"arn:aws:s3:::corporate-data/*" ,
"Condition" :{
"StringNotEquals" :{
"aws:PrincipalOrgID" :"o-a1b2c3d4e5"
}
}
},
// Statement 2: Allow Public Read from Specific Referer
{
"Sid" :"AllowWebAssetsHotlinking" ,
"Effect" :"Allow",
"Principal" :"*",
"Action" :"s3:GetObject" ,
"Resource" :"arn:aws:s3:::public-assets/*" ,
```

Condition Keys

1. **aws:PrincipalOrgID**
   Restricts access to principals belonging to a specific AWS Organization ID.
2. **aws:Referer**
   Limits access based on the HTTP Referer header. Useful for web assets.
3. **Deny + StringNotEquals**
   Creates a strict data perimeter. "If you are NOT in my Org, DENY."

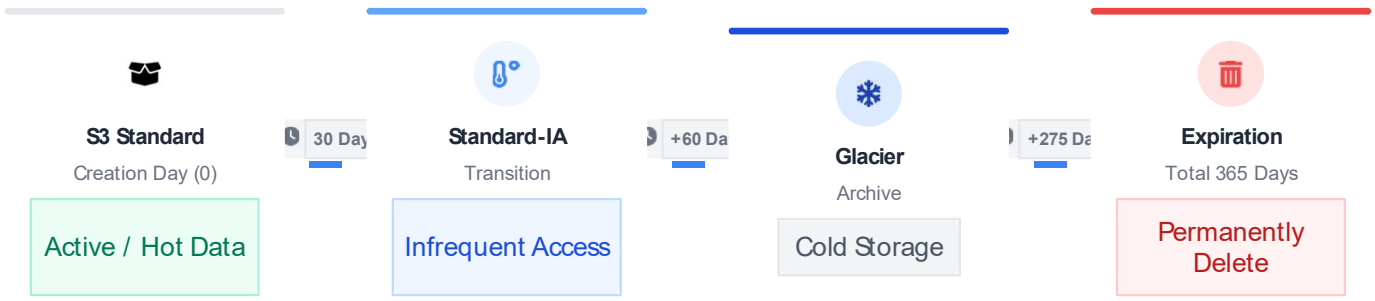
Security Pro Tips

- **Referer Spoofing:** aws:Referer can be spoofed. Never use it for private/sensitive data.
- **Block Public Access (BPA):** Account-level BPA overrides these policies. Enable BPA for all private buckets.
- Use **Service Control Policies (SCPs)** to enforce **aws:PrincipalOrgID** across accounts.

Versioning & Lifecycle: Transitions/Expiration

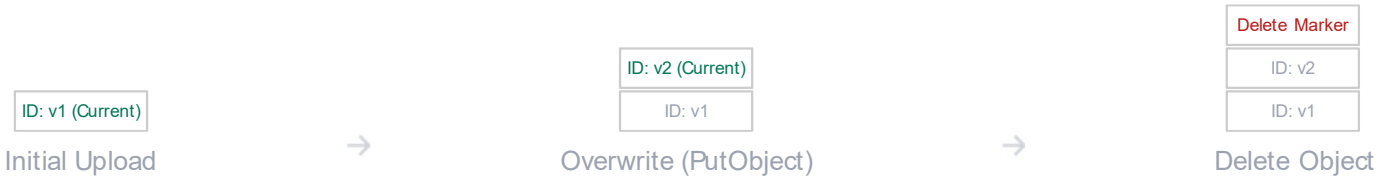
Module 03: AWS SDK + S3 Deep Dive

Object Lifecycle Policy Flow



Version Stack Behavior

What happens when you overwrite?



Lifecycle rules can target **current** versions, **non-current** versions, or **delete markers**.

Why Versioning?

Versioning enables you to recover objects from accidental deletion or overwrite.

- Protect against accidental deletes
- Required for Cross-Region Replication (CRR)

Delete Markers

"A delete marker is a placeholder for a deleted object. It has no data content."

Behavior: If you GET an object with a delete marker as the current version, S3 returns 404.

Cleanup: Use lifecycle rules to remove "expired object delete markers" to save costs.

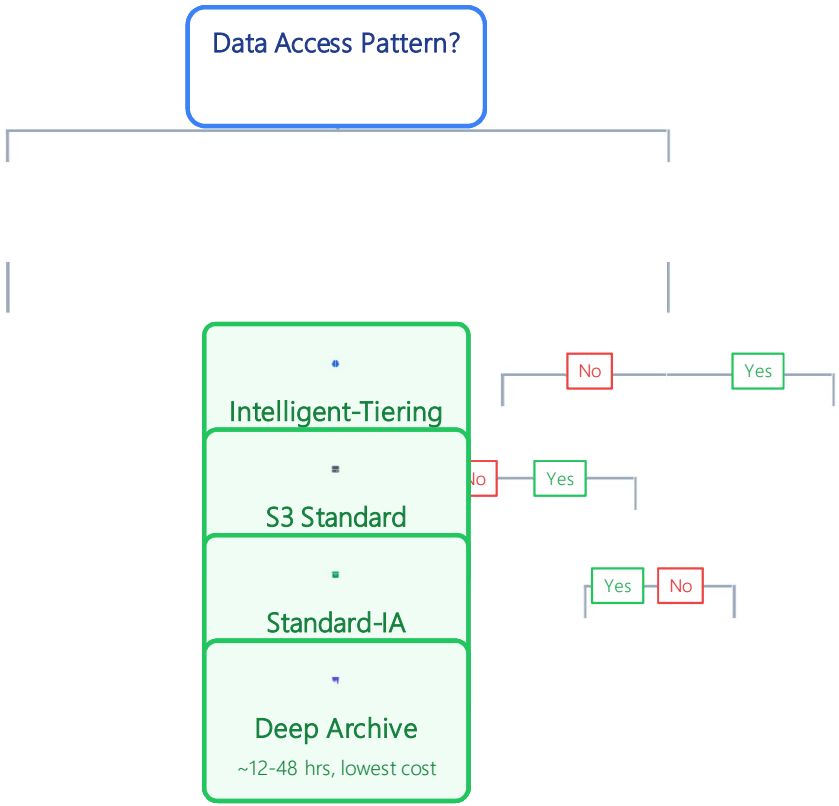
Replication Requirement

Both source and destination buckets **must** have versioning enabled for replication to work.

Cost Optimization Decision Tree (S3)

Module 03: S3 Storage & Data Management

S3 Storage Class Selection Logic



Unknown / Changing?

Predictable?

Frequently Accessed?

Long-term Archive? (>90 days)

Analytics (Athena)

Querying archived data is expensive and slow. Avoid Glacier classes if you need frequent ad-hoc queries.

Best for Analytics:

S3 Standard

Intelligent-Tiering (FA/IA)

Min. Duration Costs

andard	None
telligent-Tiering	None*
andard-IA	30 Days
Glacier Flexible	90 Days
Deep Archive	180 Days

**Monitoring fee applies per 1k objects*

Retrieval Cost Warning: accessing data from IA or Glacier tiers incurs a per-GB retrieval fee.

© 2026 AWS Training | Confidential & Proprietary

Hands-on: CLI Scripting & SDK S3 Configuration

Module 03: AWS SDK + S3 Deep Dive

Lab Details

Est. Time: 45 min

Level: Advanced

Objectives

- Provision EC2 via Python SDK (Boto3)
- Create S3 Bucket & Enable Versioning
- Apply Lifecycle Policy (Transition to Glacier)
- Validate via CLI & Clean Up

Prerequisites

- Python 3.9+
- Boto3 Lib
- AWS CLI v2

1 Provision EC2 Instance via Boto3

launch_ec2.py

```
import boto3  # Initialize EC2 Client  ec2 = boto3.  client  ( 'ec2' , region_name=  'us-east-1'  )  # Launch t3.micro instance with tags
```

2 Configure S3 (Versioning & Lifecycle)

configure_s3.py

```
import boto3 s3 = boto3.  client  ( 's3' ) bucket =  'corp-archival-lab-day3-UserXYZ'
```

Parameter Note

Ensure your bucket name is globally unique. Append your username or a random ID.

Cost Optimization

Transitioning to Glacier significantly reduces costs for rarely accessed data ($0.0036/GB) vs Standard ($0.023/GB).

© 2026 AWS Training | Confidential & Proprietary

Case Study: Media Archival System

Module 03: AWS SDK & S3 Deep Dive

The Challenge

Client
"StreamCo" - Global Media Agency

Data Profile
5 PB of Raw 8K Footage

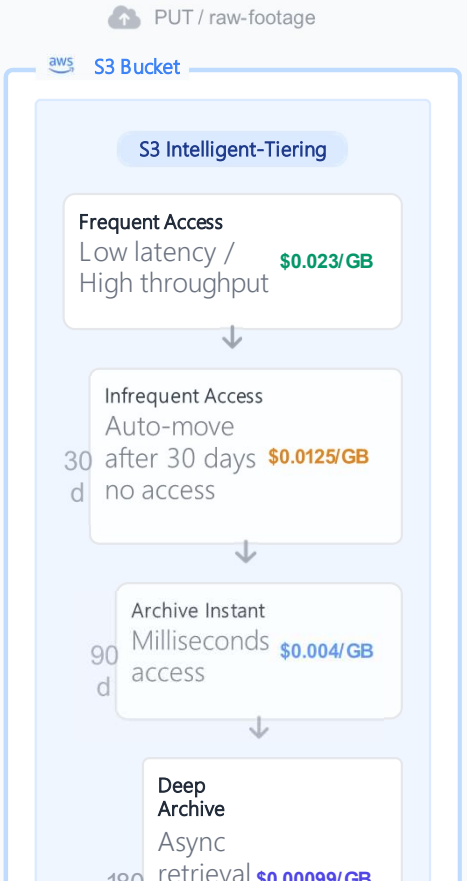
Pain Points
Exploding storage costs ($100k+/mo)

Unpredictable access patterns

Manual lifecycle rules failed (data needed back unexpectedly)

Pitfall: Small Objects

Intelligent-Tiering Workflow



Business Outcomes

Total Savings
40% reduction in TCO

Operational Overhead
Zero manual lifecycle changes

Durability
99.99...% (11 9's) across all tiers

lifecycle-config.xml

```
<LifecycleConfiguration> <Rule> <ID>MoveToIntelligentTiering</ID> <Filter> <Prefix>raw/</Prefix>
```

S3 Lifecycle Automator

60 Mins Intermediate

Design a cost-optimized backup solution that automatically transitions aging data from standard storage to cold archival storage using Terraform.

Terraform S3 Glacier KMS

Objectives

- Provision a private S3 bucket with versioning enabled.
- Enforce Server-Side Encryption (SSE-KMS) for data security.
- Configure Lifecycle Rules: Transition to IA after 30 days.
- Archive to Glacier after 90 days and expire after 1 year.

Data Lifecycle Flow

main.tf HCL

```hcl
resource"aws_s3_bucket""backup" {
 bucket = "corp-data-backup-2026"
}
resource"aws_s3_bucket_lifecycle_configuration""archive" {
bucket = aws_s3_bucket.backup.id
rule {
 id = "log-archival-policy"
 status = "Enabled"
transition {
 days = 30
 storage_class = "STANDARD_IA"
}
transition {
 days = 90
 storage_class = "GLACIER"
```