# Compute on AWS

Data replication among AZs (availability zones) in the same Region

not shared with other Regions unless explicitly specified

Data compliance (policy), Latency (distance to target user), Pricing & Service availability (both vary among Regions) are factors to consider when choosing Region

Edge Location to replicate frequent-accessed content in cache for high-latency areas

AWS Methods

AWS Management Console: user-friendly, point-and-click interface. No coding required

AWS Command Line Interface (CLI): good for production

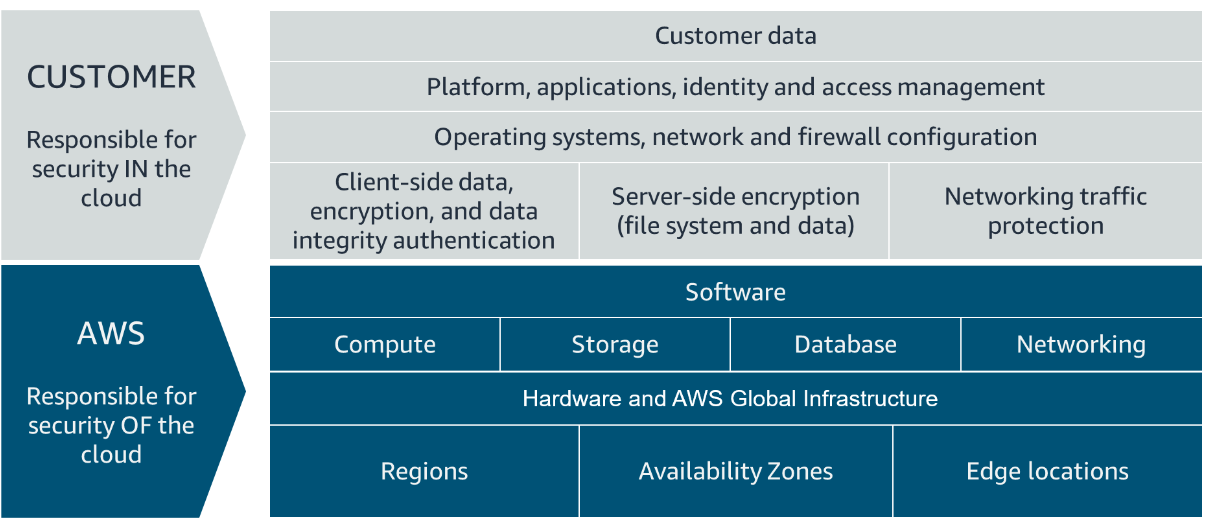
command-line tool (terminal/AWS Cloud Shell) to access AWS services

AWS Software Development Kits (SDKs)

Use APIs (Python, Java, Node.js, etc.) to interact with AWS services

Security (shared responsibility)

AWS makes sure the building is well-build and safe-to-live. Customer makes sure rented apartment is locked for security (firewall, access management, encryption…)



Security Best Practices

**Enable MFA for the Root User**: Immediately after creating your AWS account.

**Create IAM** (Identity and Access Management) **Users**: For regular use and administrative tasks.

**Use Strong Passwords**: For all accounts, especially the root user.

**Minimize Root User Use**: Avoid root user for daily tasks

**Disable access key for Root User**

Create an IAM user with admin permissions for administrative tasks.

3 authentications

Application Access Control: separate access management

Users need valid credentials (username and password) to log into the employee directory application.

IAM manages access to AWS account

IAM roles manages access to API: All API calls must be signed and authenticated to enable (Role-Based Access Between AWS Services)

IAM Policies & Groups:

Permissions are managed using IAM policies (JSON-based documents)

Policies cannot be applied to the root user but can be to IAM users.

Users can be organized into groups. Policies attached to groups.

Best Practices:

Role-Based Access:

EC2 instances need credentials for API calls to S3, but not via IAM users with usernames and passwords.

Temporary access via IAM roles is used for such cases.

IAM Roles:

Provide temporary access and are ideal for applications and services needing to perform actions without long-term credentials.

EC2 (Elastic Cloud)

web service that provides secure, resizable compute capacity in the cloud

EC2 instances are live instantiations of what is defined in an AMI, much like a cake is a live instantiation of a cake recipe.

compute-optimized, memory-optimized, storage-optimized.

Example: G instance family for graphics-intensive apps, M5 for balanced resources.

Instance types include designations like T3, A1, followed by sizes (nano to extra large).

Flexibility and Resizing features have

Amazon Machine Images (AMI): good ← reusable

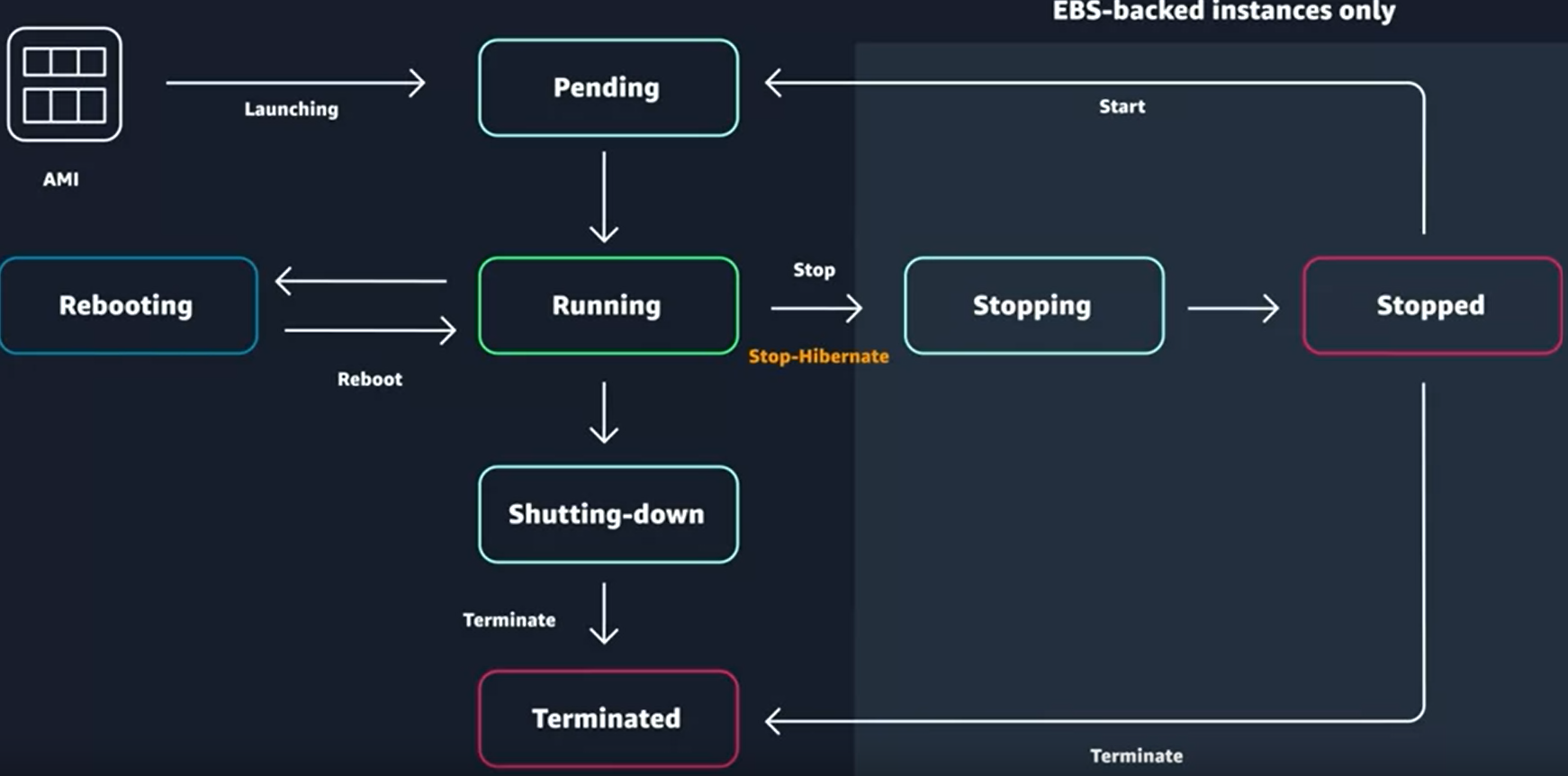
Contains instance configuration (OS, applications, other settings).

AWS provides some AMIs; others are community-provided or custom-built.

Launch multiple instances in one AMI

EC2 Instance Lifecycle

lives in Amazon Virtual Private Cloud (VPC)



Launch: Start from an Amazon Machine Image (AMI).

Pending State: Initial boot-up phase.

Running State: Instance is ready and operational; charges apply.

Instance States and Actions

Reboot: = rebooting a laptop.

Instance turns off and on again.

Stop → stopping → stopped

Can be restarted, going through pending to running states.

Stop-Hibernate

State saved in memory, quick restart without boot sequence.

Terminate → shutting down → terminated

Instance is permanently deleted, data is lost

Enable termination protection to prevent accidental deletion.

can be a good choice if you want to replace it with a new instance

Use Cases of Different EC2

General purpose

Scale-out workloads such as web servers, containerized microservices, caching fleets, distributed data stores, and development environments.

Compute optimized

High-performance web servers, scientific modeling, batch processing, distributed analytics, high-performance computing (HPC), machine/deep learning, ad serving, highly scalable multiplayer gaming.

Memory optimized (fast for workloads that process large data sets in memory)

Memory-intensive applications such as high-performance databases, distributed web-scale in-memory caches, mid-size in-memory databases, real-time big-data analytics, and other enterprise applications.

Accelerated computing

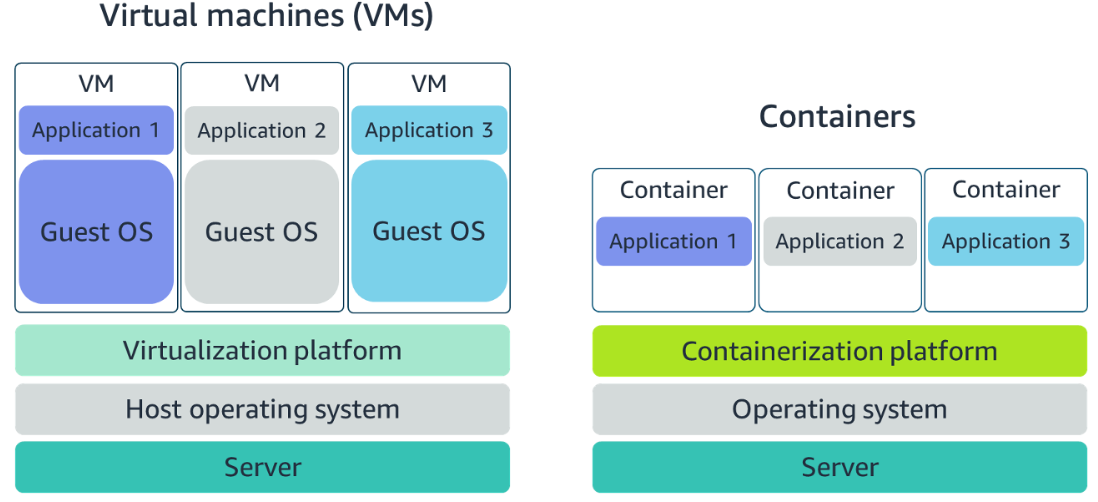
Use hardware accelerators or co-processors to perform functions such as floating-point number calculations, graphics processing, or data pattern matching more efficiently than is possible with conventional CPUs.

3D visualizations, graphics-intensive remote workstations, 3D rendering, application streaming, video encoding, and other server-side graphics workloads.

Storage optimized (high, sequential read and write access to large data sets on local storage)

NoSQL databases, such as Cassandra, MongoDB, and Redis, in-memory databases, scale-out transactional databases, data warehousing, Elasticsearch, and analytics.

Docker = container runtime that simplifies the management of the entire OS stack needed for container isolation



ORCHESTRATE CONTAINERS

Amazon ECS is an end-to-end container orchestration service that allows you to quickly spin up new containers and manage them across a cluster of EC2 instances.

task definition = text file (json format) to specify \* containers = blueprint that describes the resources you need to run that container

**{**

**"family": "webserver",**

**"containerDefinitions": [ {**

**"name": "web",**

**"image": "nginx",**

**"memory": "100",**

**"cpu": "99"**

**} ],**

**"requiresCompatibilities": [ "FARGATE" ],**

**"networkMode": "awsvpc",**

**"memory": "512",**

**"cpu": "256"**

**}**

Amazon Kubernetes Service (AKS) to manage containerized workloads and services.

EC2 instance with ECS Agent installed configured = container instance = (EKS) worker node

ECS Container = task = (EKS) pod.

ECS runs on AWS native technology. EKS runs on top of Kubernetes.

Serverless (Lamda, Fargate) no need for VPC

Abstracts underlying infrastructure.

AWS manages provisioning, scaling, fault tolerance, and maintenance.

vertical scaling (instance resize), horizontal scaling (additional instances)

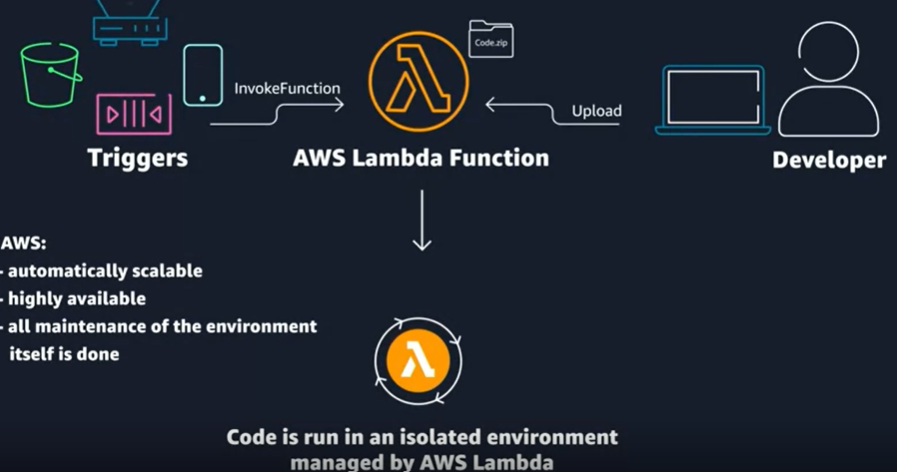
Users focus solely on application development.

Lamda

Lambda functions only run in response to triggers

Suitable for quick processing tasks like web backend handling, report processing, or microservices, but not for long-running processes (max runtime is 15 minutes)

Demo



Ex. automate the process of adding new entries based on spreadsheet

run frequently → EC2 container to get new files and update

run once a quarter → EC2 is not cost effective → Lambda (a PUT trigger from S3 bucket when new file is uploaded)

Ex. on-premises data center + linux → migrate to AWS (heavy loading)

EC2 because it might take longer time than Lambda can handle

Ex. new app with microservice + can scale up/down + lower risk of deploying

ECS / EKS

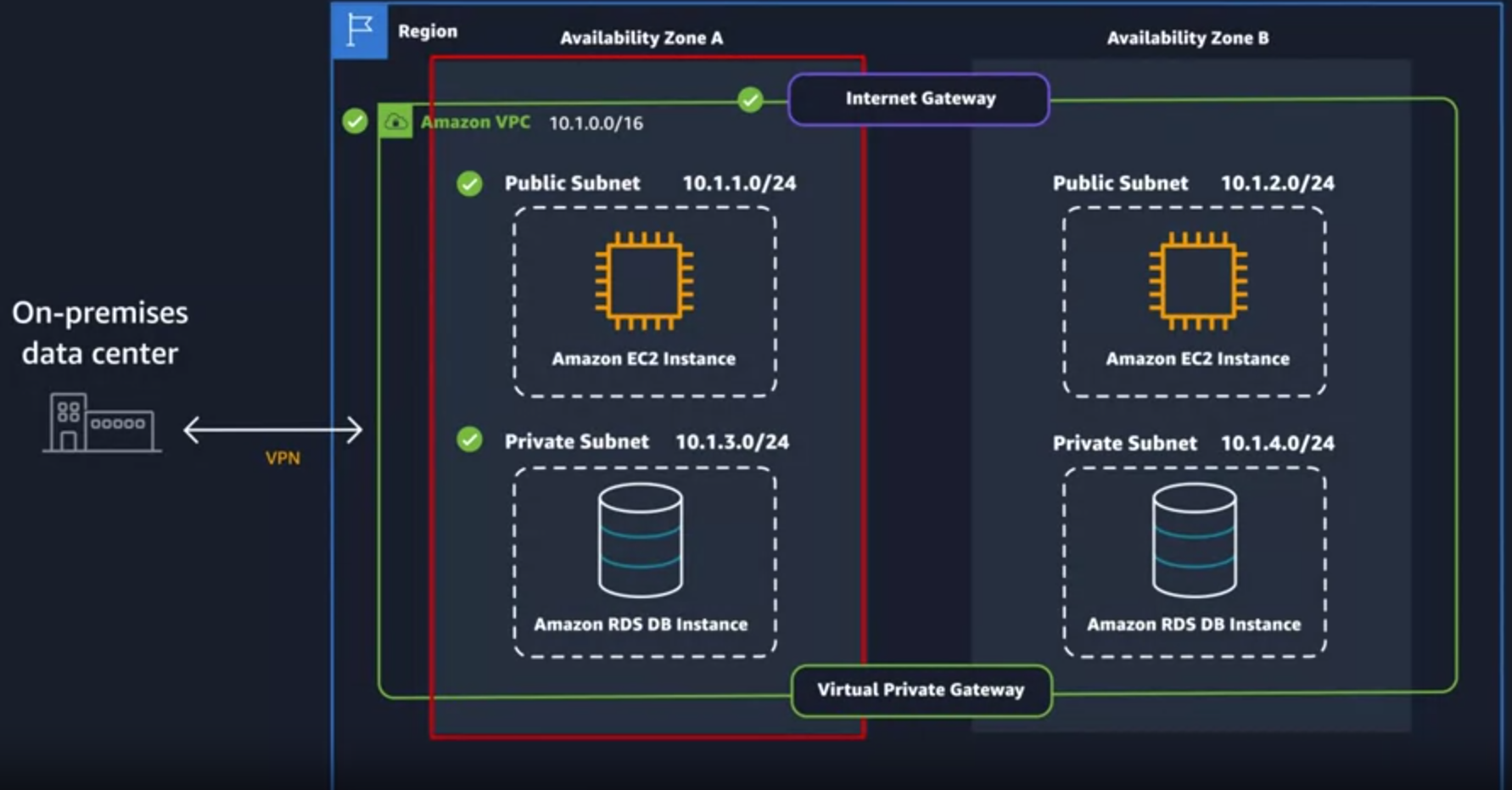
# Networking on AWS

CIDR notation

32 total bits 24 fixed and 8 flexible = 192.168.1.0/24 is LESS flexible than 192.168.1.0/16.

AWS provides smallest IP from /28 to largest IP to /16

Ex.



AZ 1 with IP for VPC is larger range 10.1.0.0/**16**

public subnet = VPC + AZ + IP (10.1.1.0/**24**)

private subnet (non overlapping) = VPC + AZ + IP (10.1.**3**.0/**24**)

AZ 2 can be a replication of AZ 1 to handle

internet gateway MUST attach to VPC to function

**VPG (virtual private gateway)** only allow access between on-premises data center and subnets

route table match traffic from internet gateway to subnet

default:

traffic to main subnet.

subnets local only ⇒ VPC isolation

**route table defines what subnet is public and private**

public subnet by route table

0.0.0.0/0 (take traffic from anywhere) to internet gateway

associate public subnets to this route table

network ACL (access control list)

firewall at subnet level (defines in/out-bound rules)

stateless ⇒ **MUST create outbound rules** to deliver output to internet

default: allow incoming + outgoing traffic from a subnet

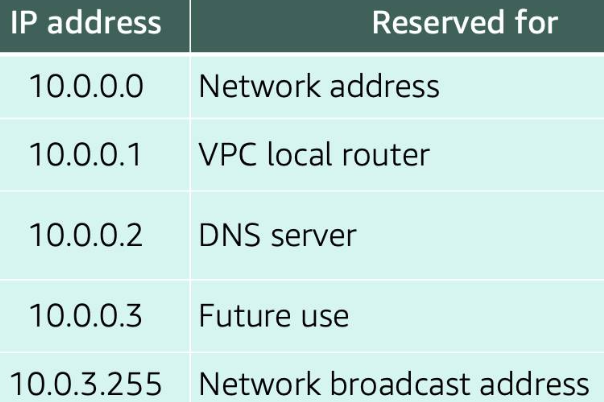
security groups **mandatory**

firewall at EC2 instance level

stateful ⇒ no outbound needed

default: no inbound traffic + allows all outbound traffic

reserved ip



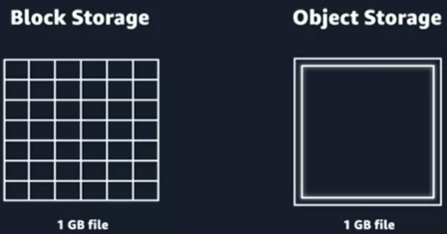
# Storage on AWS

Storage categories

Block storage: split file into chunks (blocks) (ex. good for text file modification)

direct-attached storage (DAS) or a storage area network (SAN)

Object storage: network attached storage (NAS) server. (ex. good for image file)



Storage types

Instance Store: directly attached to instance. FAST but gone after instance terminates

Amazon Elastic Block Store (EBS): data persistence (like a USB)

can attach multiple EBS volumes to a single **EC2** instance

two types = SSD-backed and HDD-backed volumes

**backup** via EBS Snapshots

S3 for larger scalable distributed flat storage

buckets to hold multiple files in S3 (region specified, unique name across all AWS buckets)

AIM + bucket policy (action allowed/denied on this bucket)

EFS (elastic file system) to multiple **EC2 instances**

FSx = Fully managed Lustre file system that integrates with S3.

ex. lambda function to transform images to new ones. Both should stored for a year

object storage on S3 (EBS is for EC2)

ex. MySQL on EC2 stores customer information with quickly updating information.

EBS

ex. web app to perform calculation. temporary data

Instance Store

ex. wordpress on multiple instances for uploading images. Multiple instance in shared storage

EFS (elastic file system)

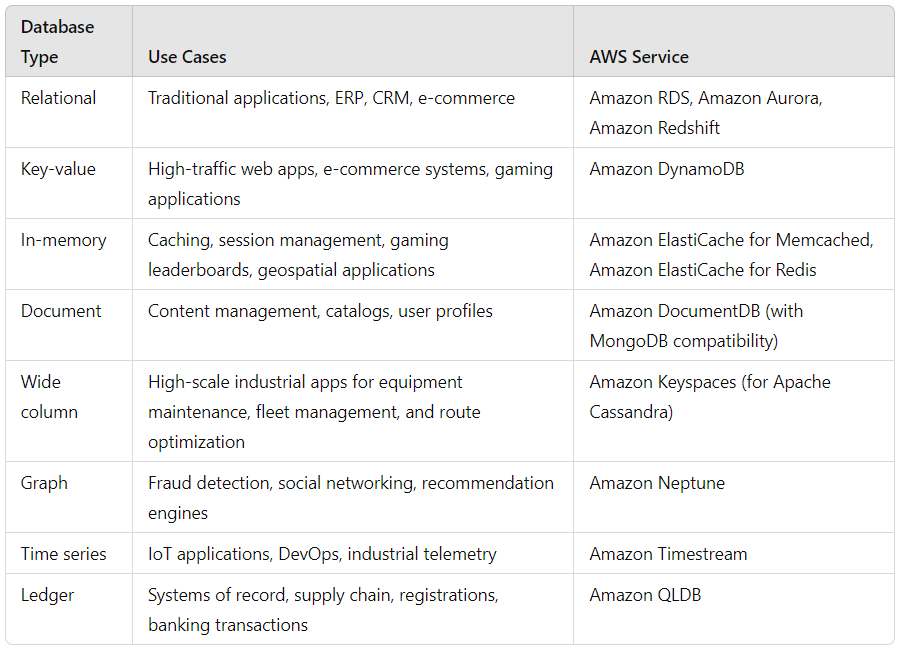
Database Types & Use Cases

DynamoDB NoSQL (charge by usage of table instead of by time)

DocumentDB (full context management system)

Neptune (graph structure)

QLDB (ledger DB) immutable



# ASW Monitoring

CloudWatch

Metrics = data collected and analyzed over time to form statistics

diff types

EC2 (CPU/network/mem utilization, disk performance)

S3 Buckets (overall size, # objects, and read/write operations)

RDS (database connections, CPU utilization, and disk consumption)

benefit

Proactive Response (detect and address issues before end users)

Performance and Reliability (identify bottlenecks and inefficiencies)

Security Threat: (spot anomalies beyond baseline of normal activity)

Data-Driven Decisions: (gain insights into user behavior and feature usage)

Cost Optimization (identify underutilized resources)

Log Structure:

Log Event: A record of activity with a timestamp.

Log Stream: A sequence of log events for a specific resource.

Log Group: Collection of log streams with shared retention and permissions settings.

CloudWatch Alarms

Alarm Configuration: Set thresholds and time periods for metrics to trigger alarms.

Alarm States: OK, ALARM, and INSUFFICIENT\_DATA.

Actions: initiate actions like EC2 actions, Auto Scaling actions, or SNS notifications.

ex. alarms for metrics like 500-error responses. Alarm can trigger automated actions like rebooting instances or calling AWS APIs via Lambda functions.

Optimization

Load Balancer distributes incoming application traffic across multiple targets

Application Load Balancer (ALB) for HTTP/HTTPS

listener (port #)

target group = backend resources (EC2 instances, lambda func…)

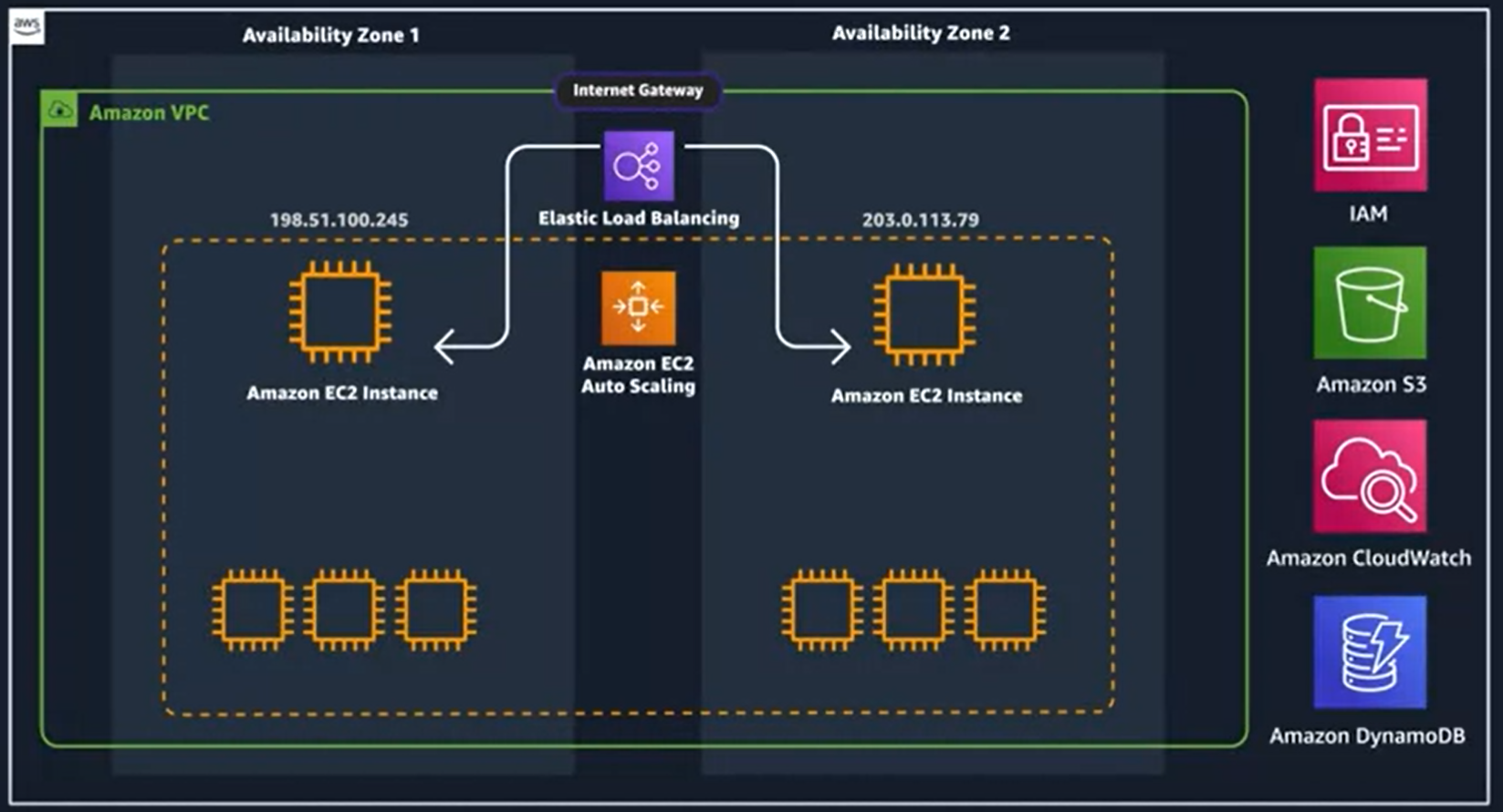
Rules ⇒ how requests are routed to targets.

schema = internet facing (for user request) / internal (between resources)

Ex. ec2 – load balancer

Network Load Balancer (NLB) for TCP/UDP/TLS

Gateway Load Balancer for IP (third-party?)

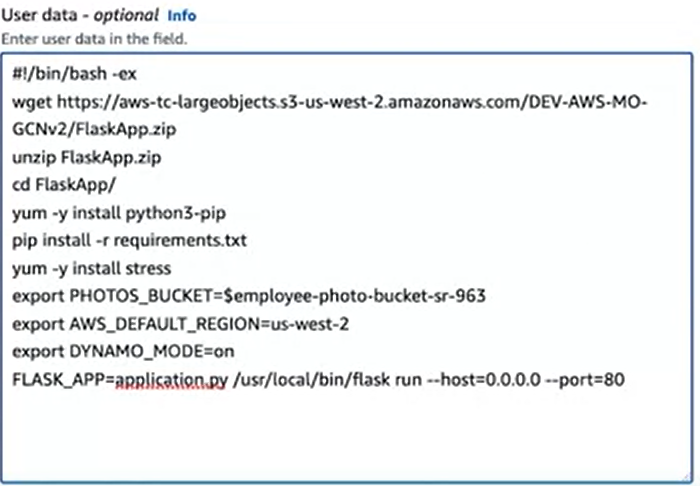


Auto Scaling

Launch template + scaling policies + EC2 Auto Scaling group

EC2 – instances → launch instance template with EC2 Audio Scaling

grab source code and run it on EC2



Auto Scaling Group

if manually deleting, MUST delete target groups when scaling down (otherwise more instances will be created)

Scaling Policies

Simple Scaling Policy: Uses a CloudWatch alarm to specify actions when triggered

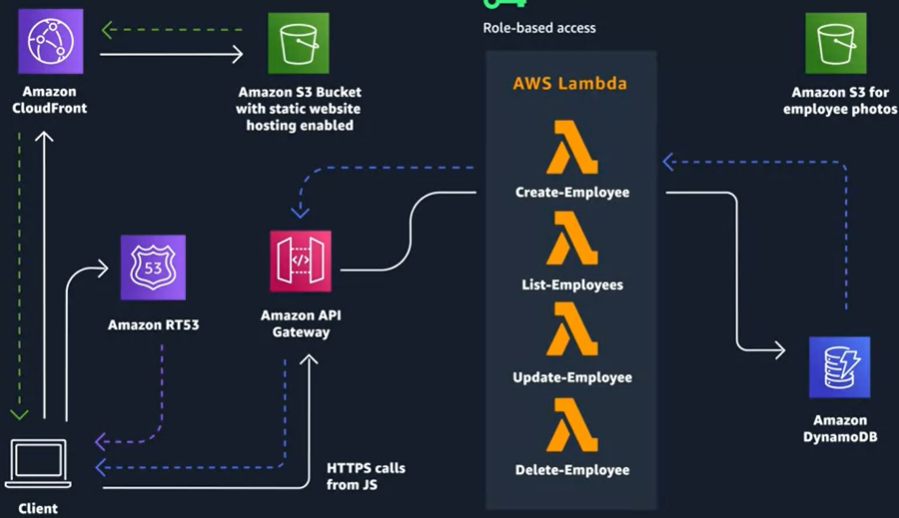
Step Scaling Policy: Responds to multiple alarms and can take different actions based on the severity of the metric

This policy type allows more granular scaling adjustments compared to simple scaling.

Target Tracking Scaling Policy

Ideal for applications scaling based on average metrics like CPU utilization, network utilization, or request count. You set a target value, and AWS automatically creates and manages the necessary CloudWatch alarms and scaling actions.

Serverless



VPC security group are auto-handled by default

Amazon CloudFront = content delivery network (CDN) service for content distribution and caching to improve performance.

Amazon RT53 for DNS management to route end-user requests to the CloudFront distribution.

Graph database

Q: API Gateway. Application Load Balancer

Project

IAM - Roles – Add – AWS Service, EC2, S3FullAccess + DynamoDBFullAccess

IAM - Users – Add – enable console access – add user to a group – (create group – EC2FullAccess) –

created user – security – create access key – CLI

Labs

1. In this task, you access the IAM dashboard and explore the existing groups, users, roles, and policies. You learn how to manage users and groups. Lastly, you learn how to add users to groups, allowing users to inherit specific group permissions.

In this task, you work with the users and groups, enabling permissions to support a business scenario.

1. In this task, you create a VPC, an Internet gateway, a Route Table with a public route to the Internet, and two public subnets in a two separate Availability Zones (AZs).

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you’ve defined. This virtual network closely resembles a traditional network that you’d operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

An Internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between your VPC and the internet. It supports IPv4 and IPv6 traffic. It does not cause availability risks or bandwidth constraints on your network traffic.

After creating a VPC, you add subnets. Each subnet resides entirely within one Availability Zone and cannot span multiple Availability Zones. A subnet is a range of IP addresses in your VPC. You can launch AWS resources into a specified subnet. Use a public subnet for resources that must be connected to the internet, and a private subnet for resources that won’t be connected to the Internet. You will not use private subnets in this lab.

# Code, Build, Test

CodeCommit

While there are many kinds of application tests, the three mentioned in this week are:

Regression testing - Tests to ensure that previously developed applications don’t break with new changes

Integration testing - Tests to ensure separately developed modules work together as expected

Unit testing - Tests a specific section of code to ensure the code does what it is expected to do

[Exercise 1](https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/DEV-AWS-MO-DevOps-C1/exercise-1.html)

AWS SAM CLI = templates to build and run your serverless applications.

[Exercise 2](https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/DEV-AWS-MO-DevOps-C1/exercise-2.html)

How to debug the build

* logfile
* Docker for local testing pass



cd app

go build

* session manager to pause and debug build, then resume the build

In yaml file

breakpoint to pause



session manager to access

[Exercise 3](https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/DEV-AWS-MO-DevOps-C1/exercise-3.html)

CodePipeline

orchestrator tool (for github, jenkins…) to run test for every commit with CodeBuild

[Exercise 4](https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/DEV-AWS-MO-DevOps-C1/exercise-4.html)

Deploy

CodeDeploy on Amazon Elastic Compute Cloud (Amazon EC2) instances or on-premises servers



–no–fail-on-empty-changeset (don’t change or deploy if no fail)

# Release & Deploy

CI/CD

C delivery (auto test/deploy to non-production + manual release)

C deploy (auto test/deploy/release to production)

Deployment Decisions

IAM roles

Style: In-Place or Blue-Green Deployment

Config…

Deployment Strategies

AWS Serverless Application Model (SAM):

Integrates with AWS CodeDeploy for gradual Lambda deployments.

Automatically deploys new Lambda function versions and creates aliases.

Gradually shifts traffic to the new version or rolls back if issues are detected.

Uses pre-traffic and post-traffic test functions and CloudWatch alarms.

Traffic-Shifting Options:

Canary: Traffic shifted in two increments with a predefined percentage and time interval.

Linear: Traffic shifted in equal increments with equal time intervals.

All-at-once: Instant traffic shift to the updated Lambda function version.

Comparison: Canary and linear options offer safer, more gradual deployments compared to all-at-once.

# AWS Monitor

Recap

### **Code**

[AWS CodeCommit](https://aws.amazon.com/codecommit/) is a secure, highly scalable, managed source control service that hosts private Git repositories. It makes it easy for teams to securely collaborate on code, with contributions encrypted in transit and at rest. CodeCommit reduces the need for you to manage your own source-control system or worry about scaling its infrastructure.

### **Build**

[AWS CodeBuild](https://aws.amazon.com/codebuild) is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. You don’t need to provision, manage, and scale your own build servers. CodeBuild scales near continuously and can process multiple builds concurrently.

### **Test**

[AWS CodePipeline](https://aws.amazon.com/codepipeline/) is a fully managed service that helps you automate your release pipelines for fast and reliable application and infrastructure updates so that you can deliver features and updates rapidly and reliably. For context, in the first course, CodePipeline was used to test the build before incrementing a continuous delivery pipeline.

### **Release**

[AWS CloudFormation](https://aws.amazon.com/cloudformation/) gives you a way to model a collection of related AWS and third-party resources, provision them quickly and consistently, and manage them throughout their lifecycles by treating infrastructure as code.

### **Deploy**

[AWS CodeDeploy](https://aws.amazon.com/codedeploy/) automates code deployments to any instance, including Amazon Elastic Compute Cloud (Amazon EC2) instances and on-premises servers. AWS CodeDeploy makes it easier for you to rapidly release new features, helps you avoid downtime during application deployment, and handles the complexity of updating your applications.

EventBridge = CodeWatch Events

triggered by event

create rules via CodeFormation by using AWS::Events::Rule

CodePipeline to manage multiple Event Bridge

AWS Config to monitor config changes (delta) before and after infra changes

CouldTrail gives the evidence of the delta

CloudTrai logs API activities made by AWS account

CloudWatch for custom metrics specific to your application



AWS X-Ray to view requests in application and to identify issues



Enabling X-Ray:

Lambda and API Gateway: Check a box in the console.

EC2: Install X-Ray daemon on the instance.

Segments and Traces:

Segments: Details about requests and work done.

Traces: Grouped segments providing comprehensive request information.

Service Map:

Visualizes each AWS resource sending data to X-Ray.

Useful for understanding complex, distributed applications.

ServiceLens Map combines CloudWatch and X-Ray data

Network Monitoring

CloudFront Logs Options

Standard Logs (Access Logs): Detailed records of requests, delivered to S3.

Real-Time Logs: Detailed records delivered within seconds to Kinesis Data Streams. Can filter requests.

API Gateway Logs:

Access Logs: Info on who accessed the API and how.

Execution Logs: Info on errors, execution traces, payloads. Logs created in CloudWatch Logs.

Data Analysis:

Logs in S3: Use Amazon Athena, Amazon QuickSight, or Elasticsearch with Kibana for querying and visualization.

Logs in CloudWatch Logs: Use CloudWatch Logs Insights for searching and querying log data to troubleshoot operational issues.

[Exercise: Monitoring pipeline changes](https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/DEV-AWS-MO-DevOps-C3/exercise-1.html)

# AWS Operate

**AWS Systems Manager:**

* **AWS Service Quotas:**
  + Provides a list of AWS default quota values and the applied quota values in your account.
  + Helps monitor and manage resource limits to prevent update failures.
* **CloudFormation Change Sets:**
  + A feature that allows you to preview changes before executing stack updates.
  + **Three Methods to Change AWS Environments:**
    - Directly modifying resources (not recommended).
    - Updating the stack directly.
    - Using change sets for a more sophisticated approach.

**Key Points:**

1. **CloudFormation and Manual Changes:**
   * CloudFormation manages infrastructure as code (IaC), and manual changes to resources created by CloudFormation can cause issues.
   * Manual changes (e.g., adding tags) can be forgotten and lead to inconsistencies when updating the stack.
2. **Resource Replacement:**
   * Some updates (e.g., changing subnets) require replacing the resource, which deletes the existing resource and creates a new one.
   * If the manually added tags are not included in the CloudFormation template, they will be lost in the new resource.
3. **CloudFormation Drift:**
   * **Definition:** Drift occurs when the actual state of infrastructure deviates from the state defined in the CloudFormation template due to manual changes.
   * **Implications:** Drift can lead to unexpected behaviors, security issues, and failed deployments.
4. **Drift Detection:**
   * CloudFormation provides a drift detection feature to identify discrepancies between the template and the actual state of resources.
   * Running drift detection helps ensure that manual changes (e.g., tags) are identified and addressed before making further updates.

**Best Practices to Avoid Drift:**

1. **Avoid Manual Changes:**
   * Do not manually change resources managed by CloudFormation. All changes should be made through the CloudFormation template.
   * This practice ensures that the infrastructure state remains consistent with the template.
2. **Incorporate Drift Detection in CI/CD:**
   * Integrate drift detection as part of your CI/CD pipeline to automatically detect and report drifts.
   * This proactive approach helps identify and resolve drift issues before they affect the deployment process.
3. **Include All Necessary Configurations in the Template:**
   * Ensure that all configurations, including tags, security groups, and instance types, are defined in the CloudFormation template.
   * This practice prevents loss of manual configurations during resource replacements or updates.
4. **Automate and Document Changes:**
   * Use automated scripts and tools to manage infrastructure changes, and ensure all changes are documented.
   * Automation reduces the risk of manual errors and ensures that changes are traceable.

**Core Features of AWS Systems Manager:**

* **Purpose:**
  + Automates operational tasks.
  + Centralized tracking and resolution of operational issues across AWS applications and resources.
  + Simplifies resource and application management.
  + Reduces time to detect and resolve operational problems.
  + Enhances visibility and control over infrastructure.
* **Benefits of AWS Systems Manager:**
  + Reduces time to resolve operational issues.
  + Provides easy-to-use automation.
  + Improves visibility and control over resources.
  + Manages hybrid environments (both AWS and on-premises).
  + Maintains security and compliance.
  + Connects with ITSM or ITOM software.

1. **Parameter Store:**
   * **Secure Storage:** Parameter Store allows you to store data securely in a key-value format, ideal for sensitive information such as passwords, database strings, AMI IDs, license codes, and serial numbers.
   * **Programmatic Access:** Applications can retrieve these parameters programmatically via API calls, detaching sensitive data from your code and eliminating the need for redeployment when values change.
   * **Credentials Rotation:** Facilitates compliance with password and credentials rotation policies by allowing easy updates and automatic refreshes in case of authentication failures.
   * **Integration:** Parameter Store integrates with other AWS services like Lambda and CloudFormation, enhancing security by offloading credentials and sensitive data from code and templates.
2. **Change Manager:**
   * **Operational Changes:** Simplifies tracking, approving, and implementing changes to application configurations and environments, especially useful in multi-account setups.
   * **Templates and Runbooks:** Create templates consisting of automation runbooks, SNS topics for notifications, and CloudWatch alarms for monitoring workflows.
   * **Change Requests:** Users create change requests based on available templates, with the option for manual approvals, ensuring controlled and auditable changes.
   * **CI/CD Integration:** While Change Manager complements CI/CD practices, it is recommended for non-routine operations like creating new AWS accounts or launching costly instance types, rather than daily code releases to avoid slowing down development processes.
3. **Shared Resources (SSM Documents):**
   * **Automations Repository:** SSM Documents provide a repository of pre-made actions, automations that can be executed immediately, such as closing security groups or enabling S3 bucket logging.
   * **Custom Actions:** You can define your own actions within SSM Documents, specifying required parameters and permissions, and executing them via the console or CLI.
   * **Integrated Workflows:** These documents can be part of larger workflows, such as those managed by Change Manager, facilitating complex operational tasks with predefined automation scripts.

### **AWS Systems Manager: Run Command Feature**

1. **Remote Command Execution:**
   * **Automate Tasks:** Perform tasks like adjusting time zones, cleaning logs, or configuring logrotate without SSH access.
   * **Non-Interactive:** Ideal for administrative tasks at scale.
2. **Ease of Use:**
   * **Access Methods:** Use via AWS Management Console, AWS CLI, PowerShell, or SDKs.
   * **Blueprints:** Predefined commands for common tasks like updates and Docker configuration.
3. **Flexible Targeting:**
   * **Target Options:** Specify instances by tags, instance IDs, or resource groups.
   * **Custom Parameters:** Pass specific information with your commands.
4. **Deployment Options:**
   * **Rollout Strategies:** Execute commands gradually, by percentage, or all at once.
   * **Package Management:** Handle installations without needing package manager specifics.
5. **Auditing and Monitoring:**
   * **Traceability:** Commands generate logs and trails for auditing purposes.
   * **Status Tracking:** Monitor command execution status via API calls.

**Session Manager:**

* **Enhanced Security and Auditing:**
  + Centralizes access control and logs commands in CloudWatch Logs.
  + Audits session activity in CloudTrail.
* **No SSH Keys Required:**
  + Access instances via browser or AWS CLI with one click.
  + Uses HTTPS sockets instead of SSH.
* **Setup:**
  + Create an IAM role with Amazon SSM Managed Instance Core policy.
  + Ensure Systems Manager agents are installed and running.
  + Connect through the AWS Management Console or command line.

**Patch Manager:**

* **Automated Patching:**
  + Patches OS and applications across managed instances.
  + Supports simultaneous or rolled-out patch deployment.
* **Best Practices:**
  + Integrate with AWS Security Hub to monitor patch compliance and security status.

**State Manager:**

* **Automated Configuration Management:**
  + Schedules actions via SSM documents to maintain desired state.
  + Bootstraps instances, configures network settings, joins domains, and runs scripts.
* **Integration with Other Tools:**
  + Works with open-source frameworks like Chef and Puppet to ensure agents are operational and updated.

**Usage Examples:**

* Session Manager: Initiate secure shell sessions without SSH keys.
* Patch Manager: Apply security patches across instances.
* State Manager: Automate configuration tasks and ensure instances meet compliance standards.

### **Troubleshooting Production Issues Using AWS Systems Manager Run Command**

1. **Attempt Debugging in Dev Environment:**
   * Determine if the issue can be replicated and debugged in a development environment.
   * If the issue is specific to production, move to the next step.
2. **Enable CloudWatch Logs:**
   * Check if CloudWatch Logs are already installed and configured.
   * If not, proceed to the next step.
3. **Use AWS Systems Manager Run Command:**
   * Deploy a CloudFormation template to set up necessary configurations.
   * Use AWS Systems Manager Run Command to run commands on the production servers securely.
   * Gather logs and data needed for troubleshooting without direct server access.

[Exercise: Logging](https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/DEV-AWS-MO-DevOps-C3/exercise-2.html)