MODULE 1 · STUDY NOTES

Understanding Systems Operations on AWS

Automation, core services, resource scope and service limits, IAM security, and the AWS CLI — the SysOps foundation for the rest of the course

Systems operations — often shortened to SysOps — is the work of building, testing, deploying, monitoring, maintaining, and safeguarding complex computing systems, and in the cloud that work is done through automation rather than by hand. This module lays the groundwork for the rest of the course. It starts with why packaging infrastructure as scripts, programs, or templates makes deployments repeatable and consistent; introduces the three service areas every SysOps engineer touches — networking (Amazon VPC), compute (Amazon EC2), and access control (IAM); explains how far each AWS resource reaches (global, Regional, or Availability Zone) and the account service limits that keep spending and the AWS network in check; goes deep on IAM as the service that secures who can do what in an account; and finishes with the AWS CLI, the language-agnostic tool that turns all of it into automation. A running project — a troubleshooting knowledge base for the fictional Mom & Pop Café & Bakery — puts the concepts to work.

LEARNING OBJECTIVES

- Describe system operations in the cloud related to automated and repeatable deployments
- Explain AWS Regions and edge locations, and the criteria for selecting them
- Describe core services related to system operations, including services for network, compute, and access
- Explain how AWS Identity and Access Management (IAM) provides security over AWS account resources
- Describe AWS Command Line Interface (AWS CLI) features

1.1 Systems operations in the cloud

Systems operations involves the responsibilities and tasks required to build (create), test, deploy, monitor, maintain, and safeguard complex computing systems. The *build* — or *create* — activity is where a great deal of the cloud advantage lives: instead of assembling every piece of a deployment by hand, you develop **reusable infrastructure templates** that are then tested, deployed, monitored, maintained, and safeguarded like any other artifact.

It is a best practice to use a file or an application to package the description of the network and the components that will run in it. Varying degrees of automation are possible through **scripts, programs, or templates** — a Linux shell script, a Python or Ruby application, a C# application, or a template format such as **AWS CloudFormation** (covered in depth later in the course). Whether you specify the infrastructure **declaratively** or **programmatically**, the payoff is the same: you can create as many deployments of it as you need, when you need them.

AUTOMATION OF SYSTEM OPERATIONS ENABLES

- **Repeatable deployment** of infrastructure and applications on demand.
- Creation of **self-describing systems** — the definition is the documentation.
- Building **well-tested, secure systems** by treating infrastructure like code.

WORKED EXAMPLE – ONE TEMPLATE, THREE ENVIRONMENTS

A team captures a network and its EC2 instances in a single CloudFormation template. Because the definition is repeatable, they deploy it once to stand up **production**, again to create a **development** environment, and again for **test** — each one consistently matching the others. Later they deploy the same template into a second Region for disaster recovery. No manual clicking, no drift between environments: the template is the single source of truth.

1.2 Core services: network, compute, and access

Three areas are of primary interest to systems operations — **network**, **compute**, and **controlling access** — and each maps to a core AWS service. These services are introduced here and revisited in depth throughout the course.

TABLE 1.1 Core services for systems operations

AREA	SERVICE	WHAT IT PROVIDES
Network	Amazon VPC	A logically isolated virtual network for launching resources; configurable IP ranges, routing, gateways, and security settings
Compute	Amazon EC2	Virtual computing environments (instances) launched from AMIs; scalable and optimizable
Access	AWS IAM	Manages authentication and access — users, groups, roles, policies, and identity federation

When you configure a network environment in the cloud, **Amazon VPC** is the service you use. A VPC is the network environment into which resources such as EC2 instances are launched, designed to give you greater control over isolating the environment and its resources from one another. Each VPC is **defined within a Region**; you can have multiple VPCs in a Region and VPCs in as many Regions as you want, and the **subnets** you define inside a VPC can be either **public** or **private**.

Amazon EC2 lets you provision virtual computing environments — virtually anything you can do with a server, you can do with an EC2 instance. Most server operating systems are supported, including Windows, Red Hat, SUSE, Ubuntu, and Amazon Linux. You can capture an image of a server at any time as an **Amazon Machine Image (AMI)** and reuse it to launch future instances. EC2 is **scalable** (add instances when needed and terminate them when not) and **optimizable** (choose instance types tuned for CPU, memory, storage, networking, or graphics, plus general purpose — each in a variety of sizes).

AWS IAM helps you securely control access to AWS resources for both people and programs. You use it to control **who** can use resources (authentication) and **what** they can do with them, in what ways (authorization) — with granular control all the way down to which API calls each service can make. Create additional users and assign permissions following the **least-privilege** principle, give each user unique credentials so there is no need to share, and use **identity federation** for single sign-on with corporate-directory credentials. IAM is the focus of a full section later in this module.

KEY TERMS

Instance
A running virtual server in Amazon EC2, launched from an AMI

Subnet
A range of IP addresses within a VPC; can be public or private

User / group / role
IAM identities — a person or application (user), a collection of users (group), or an assumable identity with temporary credentials (role)

1.3 Scope of AWS services and service limits

Some AWS service features are **Regional**, others are **global**, and others exist only within the **Availability Zone (AZ)** in which they are created. Understanding this scope is essential. For example, when you define an S3 bucket you must specify its Region; when you capture an EC2 instance as an AMI, that AMI is available only in the Region where it was captured, so launching from it elsewhere requires **copying the AMI** to the other Region first. Every Region has **at least two Availability Zones** — launch an instance in us-west-1 and it lands in an AZ such as us-west-1b, and to attach a new EBS volume you must create that volume in the **same AZ** (us-west-1b) or it will not attach.

TABLE 1.2 Scope of AWS service features

SCOPE	EXAMPLES	NOTES
Global	IAM users, groups, roles; Route 53 hosted zones; CloudFront distributions	Referenced across all Regions in the account
Regional	S3 buckets; AMIs; CloudWatch metrics; EBS snapshots; ElastiCache clusters; VPCs	You choose the Region when you create them
Availability Zone	EC2 instances; EBS volumes; subnets; RDS DB instances	Defined within a single AZ

WHAT INFLUENCES REGION SELECTION

- **Cost** — pricing varies by Region.
- **Legal and data-governance requirements** — a country's laws and regulations can affect what data may cross geographic boundaries and which services are offered.
- **Latency** — choose a Region with geographic proximity to your customers to reduce it.

New accounts start with **default service limits** that limit potential excessive charges and keep the AWS network healthy. Some are **soft limits**: for example, the per-Region cap on running t2.large On-Demand instances is 20 — try to run a 21st while 20 are running and you get a capacity error. You can open a case in the AWS Support Center and request a **Service Limit Increase**, and such requests are often approved. Other limits are **hard limits** that cannot be changed — for example, the Amazon SNS API throttles at 100 Subscribe transactions per second.

COMMON PITFALL

Do not treat every limit as immovable — or as negotiable. A soft limit (like the 20 On-Demand instances of a type per Region) is raised with a Support case; a hard limit (like SNS 100 Subscribe transactions per second) cannot be changed at all. On the exam, the phrase *capacity error* after hitting a default cap points to a soft limit and a Service Limit Increase request.

1.4 AWS Identity and Access Management (IAM)

IAM lets you centrally manage authentication and access to AWS resources, and it is offered as a feature of an AWS account at no charge. You create **users, groups, and roles** and apply **policies** to them to control access. IAM is not only for user authentication — applications and other AWS services also use it for access. It eliminates the need to share passwords or access keys when granting rights, makes it easy to enable or disable a user's access over time, and provides granular control not just over which resources can be accessed but over **which actions** can be performed on them (for example, *who can terminate EC2 instances?*).

Every request turns on two questions. **Authentication** is the first step — it establishes who is making the request. **Authorization**, based on knowing who the user is, determines which resources they can access and what they can do with them. IAM users can be given **programmatic access** (an access key ID and secret access key for the API, CLI, and SDKs), **AWS Management Console access** (an account ID or alias, user name, and password, with an MFA code prompt if MFA is enabled), or both.

TABLE 1.3 Types of security credentials

CREDENTIAL	USED FOR
Email address and password	Associated with the AWS account (the root user)
IAM user name and password	Accessing the AWS Management Console
Access keys	Programmatic requests — the AWS CLI, APIs, and SDKs
Multi-factor authentication (MFA)	An extra layer of security; can be enabled for the root account and IAM users
Key pairs	Used only for specific AWS services, such as Amazon EC2

ROOT USER AND IAM BEST PRACTICES

- Every account has a **root user** tied to a sign-up email; do not use it for everyday tasks. Use it only to create the first IAM user, then **lock the root credentials away** for the rare tasks that require them.
- Follow **least privilege** — grant access only to the services and actions that are needed; more can be granted over time.
- Use **IAM roles** to provide cross-account access instead of sharing credentials.
- Configure **strong password policies**, enable **MFA** for privileged users, and **rotate credentials** regularly.

Policies define the permissions attached to identities. The main policy types, roughly in order of how often they are used, are **permission policies, permission boundaries, managed policies, and inline policies**. Two broad categories matter most: **identity-based policies** are attached to IAM identities — users, the groups they belong to, and roles — as managed or inline policies; **resource-based policies** are attached to resources (the most common examples are S3 bucket policies and IAM role trust policies) and are JSON documents. **Resource-level permissions** offer granular control over specific objects for some services — a policy can name specific EC2 instances and EBS volumes for actions like Reboot, Start, Stop, and Terminate, but not for **RunInstances**, because the instance ID does not exist until the run completes. **Service control policies (SCPs)** apply permission boundaries across AWS Organizations or organizational units, and **access control lists (ACLs)** control which principals can access a resource — the only policy type that does not use the JSON document structure.

WORKED EXAMPLE – ANATOMY OF AN IAM POLICY

A short JSON policy shows the moving parts: a top-level `Version` and a `Statement` array; inside a statement, a `Sid` (statement ID), an `Effect` of `Allow`, an `Action` covering all Amazon EC2 actions, a `Resource` of all resources, and a `Condition` that permits the request only when `aws:MultiFactorAuthPresent` is true and the call comes from a specific `aws:SourceIp`. Evaluation logic is fixed: authenticate the principal → process the request context (actions, resources, principal, environment) → evaluate the applicable policies by type and category → allow or deny. Two rules to memorize: the **order in which policies are evaluated has no effect on the outcome**, and an explicit **Deny overrides any Allow**.

IAM roles can be used three ways: interactively in the IAM console, programmatically with the AWS CLI, or through the SDKs (API calls). An application or a service such as EC2 can **assume a role** by requesting temporary security credentials to make programmatic requests — this is how you grant an EC2 instance's application access to other AWS services without embedding long-lived keys. When an IAM user assumes a role, they **give up their original permissions** and take on the role's; exiting the role restores the original permissions. Switching roles also enables **cross-account access**; when the other account belongs to a third party, add a shared secret called an **external ID** so one account cannot guess a role ARN and hand it to a deputized account. A role's permissions policy specifies services and actions and can add **conditions** based on time, date, source IP, and more; **managed policies** usually make ongoing management easier than inline policies.

COMMON PITFALL

Roles are not users, and access keys are not console passwords. **IAM roles** provide *temporary* credentials that a user, application, or service assumes — assign one to an EC2 instance to let its application reach other AWS services. **Access keys** provide *programmatic* access for the AWS CLI and SDKs and are distinct from the user name and password used for the console.

1.5 The AWS Command Line Interface (AWS CLI)

There are **three ways to use AWS services**, all built on a common, REST-like API. The **AWS Management Console** is a rich graphical interface for most products (occasionally a brand-new feature is not fully exposed there yet). The **AWS CLI** is a suite of utilities that run from a command program on Linux, macOS, or Windows. The **SDKs** are packages that access AWS from popular languages such as Python, Ruby, .NET, or Java. All three can be used interchangeably — an instance created via an SDK call can be described with `aws ec2 describe-instances` and later terminated from the console — though a change made through the CLI or API may take a few seconds or minutes to appear in the console.

AWS MANAGEMENT CONSOLE	AWS CLI (AND SDKS)
point-and-click; good for exploring and one-off tasks ▪ A rich graphical interface for most AWS services ▪ Simplest path for learning and occasional tasks ▪ Some brand-new features may not be fully exposed yet	*scriptable; good for automation and scale* ▪ **Automate** the creation and building of an infrastructure ▪ **Expedite** administration by running hundreds of commands via a script or file ▪ Language-agnostic and repeatable across Linux, Windows, and Mac ▪ The solution architecture is **unchanged** by choosing the CLI over the console

On Linux, install with `pip3 install awscli --upgrade --user` — the `--user` switch places the install in `~/.local/bin` — and verify with `aws --version`. The CLI runs on Linux, Windows, and Mac and comes preinstalled on Amazon Linux instances. After installing, run `aws configure` to set defaults for all commands: the **access key ID**, the **secret access key**, a **default Region**, and the **default output format**.

> **WORKED EXAMPLE – CONFIGURING THE CLI AND PICKING AN OUTPUT FORMAT**
>
> Running `aws configure` prompts for the access key ID, secret access key (both from an IAM user), a default Region such as us-west-2, and a default output format. **JSON** is the default and is best when output will be processed by a program. Two other formats are available: a **table** format (the most human-readable; set with `AWS_DEFAULT_OUTPUT="table"`) and a **text** format (tab-delimited lines that work well with Unix text-processing tools). Any command can also override the format with the `--output` option.

A command breaks into parts: `aws <service> <operation> <parameters> <options>`. You name the top-level **service** (for example `ec2`), the **operation** to perform on it (`stop-instances`, `run-instances`), the **parameters** it needs (each preceded by two dashes, e.g. `--instance-id i-1234567890abcdef0`), and finally the **options** (such as `--output json`). Every command's syntax and examples are available through the built-in help — `aws help`, `aws ec2 help`, or `aws ec2 describe-instances help`.

`aws ec2 describe-instances` returns an array of **Reservations**, each holding an array of one or more **Instances**, with each instance's properties as name-value pairs. Three options shape what you get back. `--query` limits which fields are **displayed** on the **client side** using **JMESPath**, addressing the path from the top element down with zero-based indexing — for example `Reservations[0].Instances[0].State.Name` returns just the state name of the first instance; put a wildcard in place of an index to return the value for every member of an array. `--filter` restricts the result set on the **server side** (for example, only instances whose platform is Windows) and can be combined with `--query`. `--dry-run` checks whether you have the required permissions **without making the request**, returning **DryRunOperation** if you do and **UnauthorizedOperation** if you do not.

> **COMMON AWS CLI COMMANDS**
>
> - **Amazon EC2**: `aws ec2 run-instances` (launch instances from an AMI), `aws ec2 describe-instances`, `aws ec2 create-volume` (an EBS volume in the same AZ), `aws ec2 create-vpc` (with a CIDR block).
> - **Amazon S3**: `aws s3 ls` (list buckets/objects), `aws s3 cp` (copy to/from/between S3), `aws s3 mv` (move), `aws s3 rm` (delete an object).
> - Note: **Amazon VPC and Amazon EBS actions are performed with the `aws ec2` command.**

> **COMMON PITFALL**
>
> Keep the two result-shaping options straight: `--filter` runs on the **server** and controls which resources come back, while `--query` runs on the **client** and controls how much of what came back is displayed. They can be used together, and confusing which side each runs on is a classic knowledge-check trap.

1.6 Test strategies, the course project, and the use case

The course threads two applied elements through every module. The **troubleshooting knowledge base** is a project you build up over time: as you hit situations, you add entries so the document becomes a reference guide. Module 1's focus categories are **Security & Compliance** (from the IAM material) and **Foundational IT** (from installing and configuring the AWS CLI). The running business scenario is **Mom & Pop Café & Bakery**, whose owners want to use AWS — with

help from an AWS team of a TAM, a Solutions Architect, a SysOps Engineer, and a Developer — to automate the business, starting with installing the AWS CLI and establishing IAM controls.

WORKED EXAMPLE – THE MODULE SCENARIO, READ FOR KEYWORDS

A federal contractor in Seattle is expanding into Germany, which has strict data-sovereignty rules. The contractor has thousands of staff, many AWS accounts, and uses a directory service to manage AWS account access. Auditors need access to all systems that store and process German data. How should access be provided? Read for keywords before the answer choices: **directory service**, **many accounts**, **thousands of staff**, **external auditors**. The best answer: **create a federated identity for each auditor in the directory and assign each auditor a role, with appropriate permissions, in each account.** Federation reuses the existing corporate directory (SSO via SAML), and roles grant scoped, temporary, cross-account access. The distractors fail on the keywords — creating an IAM user per account ignores the directory and the scale; SSH keys to EC2 hosts and handing over root passwords violate least privilege outright.

WHY USE THE AWS CLI INSTEAD OF THE CONSOLE?

- **Automate** the creation and building of an infrastructure.
- **Expedite** administrative tasks by running hundreds of commands through a script or file.
- The activity (installing, configuring, and practicing the CLI on a Red Hat EC2 instance over SSH) shows that the underlying **solution architecture does not change** — only how you drive it.

Module summary

- Systems operations builds, tests, deploys, monitors, maintains, and safeguards systems; in the cloud, automation via scripts, programs, or templates (such as CloudFormation) makes deployments repeatable, self-describing, and consistent across Regions and dev/test/production.
- The three core service areas are networking (Amazon VPC — a logically isolated virtual network defined in a Region), compute (Amazon EC2 — scalable, optimizable virtual servers launched from AMIs), and access (AWS IAM).
- Resource scope matters: IAM, Route 53 hosted zones, and CloudFront are global; S3 buckets, AMIs, EBS snapshots, ElastiCache clusters, CloudWatch metrics, and VPCs are Regional; EC2 instances, EBS volumes, subnets, and RDS instances live in an Availability Zone — and an EBS volume attaches only within its AZ.
- Region selection weighs cost (varies by Region), legal and data-governance requirements, and latency (proximity to users); every Region has at least two Availability Zones.
- Service limits protect against excessive charges and keep the network healthy: soft limits (e.g. 20 On-Demand instances of a type per Region) can be raised through a Support case; hard limits (e.g. SNS 100 Subscribe transactions per second) cannot.
- IAM (a free account feature) centrally manages authentication (who) and authorization (what actions on which resources) through users, groups, roles, policies, and identity federation; use least privilege, and an explicit Deny always overrides any Allow regardless of evaluation order.
- Security credentials include the root email/password, IAM user name/password (console), access keys (programmatic — API/CLI/SDK), MFA (an extra layer for root and IAM users), and key pairs (specific services like EC2); reserve the root user for tasks that require it and enable MFA.

- IAM roles grant temporary credentials and can be assumed by users, applications, or services (for example, an EC2 instance) to enable cross-account access and SAML 2.0 federation without sharing long-lived keys.
- The AWS CLI and SDKs sit on a common REST-like API and automate AWS: aws <service> <operation> --parameters; --filter filters server-side, --query (JMESPath) limits displayed fields client-side, --dry-run checks permissions, and output can be JSON, table, or text.

Review questions

1. Why is automating systems operations preferable to building each deployment by hand?

Answer: Packaging infrastructure as a script, program, or template makes deployments repeatable on demand, produces self-describing and well-tested secure systems, and lets you recreate an identical environment in different Regions or as matching development, test, and production environments.

2. A colleague created an AMI in us-east-1 and wants to launch an instance from it in eu-west-1. What must happen first, and why?

Answer: AMIs are Regional, so the AMI must be copied to eu-west-1 before an instance can be launched from it there.

3. Distinguish a soft service limit from a hard service limit, with an example of each.

Answer: A soft limit (for example, 20 On-Demand instances of a type per Region) can be raised by requesting a Service Limit Increase through the AWS Support Center; a hard limit (for example, Amazon SNS's 100 Subscribe transactions per second) cannot be changed.

4. What is the difference between authentication and authorization in IAM?

Answer: Authentication establishes who the principal is (the first step); authorization determines which resources they may use and what actions they may perform, down to individual API calls.

5. An IAM user's permissions include both an Allow and an explicit Deny that apply to the same action. What happens, and does the order the policies are evaluated in change the result?

Answer: The explicit Deny overrides the Allow, so the action is denied; the evaluation order has no effect on the outcome.

6. Name the three ways to interact with AWS services and the foundation they share.

Answer: The AWS Management Console, the AWS CLI, and the SDKs — all built on a common, REST-like API, and usable interchangeably.

7. For a large organization whose staff are managed in a corporate directory and that must give external auditors access across many AWS accounts, what IAM approach fits, and why?

Answer: Create a federated identity for each auditor in the directory and assign each one a role with appropriate permissions in each account — federation reuses the existing directory via SAML/SSO, and roles grant scoped, temporary cross-account access without creating and sharing long-lived credentials.