Cloud Storage

DA 410/510

Richard Kelley

Devices and Filesystems in Linux

“Everything is a File”

- In Linux, hardware devices are accessed through special files in /dev
- These are not normal files — they are interfaces to drivers
- The kernel exposes devices using two main abstractions:
  - Block devices
  - Character devices

Example (in an EC2 instance)

- `ls -l /dev`
- Look at the first character
  - b → block device
  - c → character device

Character Devices

- A ***character device*** provides a stream of bytes, accessed sequentially.
- Properties
  - Data is read/written in order
  - No concept of random access
  - No fixed-size blocks
  - Often represents interactive or streaming hardware
- Examples
  - `/dev/tty` — terminal
  - `/dev/null` — data sink
  - `/dev/random` — random byte stream

Block Devices

- A ***block device*** stores data in fixed-size blocks and supports random access.
- Properties
  - Data stored in fixed-size blocks (e.g., 4KB)
  - Can read or write any block independently
  - Designed for storage (disks, SSDs)
  - Supports buffering and caching
- Examples
  - `/dev/sda`
  - `/dev/nvme0n1`
  - `/dev/xvda` (common on EC2)

Why Block Devices Matter

- A block device does not contain files.
- It contains raw blocks.
- A ***filesystem*** is written on top of a block device.
- Layering:
  - Hardware (or virtual disk)
  - → Block device (/dev/nvme1n1)
  - → Filesystem (ext4, xfs, etc.)
  - → Mounted directory (/data)
  - → Files

Two Commands for Working with Block Devices in Linux

- `lsblk`
  - What disks (block devices) exist?
- `df`
  - What filesystems are mounted and how full are they?

lsblk

- Lists block devices known to the kernel
- Shows disks, partitions, and mount points
- Displays the tree structure of storage

```
ubuntu@ip-172-31-23-90:~$ lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
loop0           7:0    0 27.6M  1 loop /snap/amazon-ssm-agent/11797
loop1           7:1    0 50.9M  1 loop /snap/snapd/25577
loop2           7:2    0   74M  1 loop /snap/core22/2163
nvme0n1       259:0    0    8G  0 disk
├─nvme0n1p1   259:1    0    7G  0 part /
├─nvme0n1p14  259:2    0    4M  0 part
├─nvme0n1p15  259:3    0  106M  0 part /boot/efi
└─nvme0n1p16  259:4    0  913M  0 part /boot
ubuntu@ip-172-31-23-90:~$
```

df -h

- Reports disk space usage
- Shows mounted filesystems
- Displays total size, used space, available space
- `-h` means “human-readable” (MB, GB instead of blocks)

```
ubuntu@ip-172-31-23-90:~$ df -h
Filesystem       Size  Used Avail Use% Mounted on
/dev/root        6.8G  1.8G  5.0G  27% /
tmpfs            458M     0  458M   0% /dev/shm
tmpfs            183M  872K  182M   1% /run
tmpfs            5.0M     0  5.0M   0% /run/lock
efivarfs         128K  3.6K  120K   3% /sys/firmware/efi/efivars
/dev/nvme0n1p16  881M   89M  730M  11% /boot
/dev/nvme0n1p15  105M  6.2M   99M   6% /boot/efi
tmpfs             92M   12K   92M   1% /run/user/1000
ubuntu@ip-172-31-23-90:~$
```

Device Naming Schemes

- There are 2 in use now
  - Old-Style (SCSI/SATA Emulation)
    - Devices: `/dev/sda`, `/dev/sdb`, `/dev/sdc`
    - Partitions: `/dev/sda1`, `/dev/sda2`
    - Mostly used on older machines.
  - Modern NVMe Naming
    - Devices: `/dev/nvme0n1`, `/dev/nvme1n1`
    - Partitions: `/dev/nvme0n1p1`, `/dev/nvme0n1p2`
    - Used on modern hardware

Filesystems in Linux — Abstracting from Block Devices

- Core problem:
  - A block device is just a sequence of numbered blocks.
- It does not know about:
  - Filenames
  - Directories
  - Permissions
  - Timestamps

Filesystems in Linux

- A filesystem is a data structure written onto a block device that organizes raw blocks into:
  - Files
  - Directories
  - Metadata (ownership, permissions, timestamps)
- Layered model:
  - Block device
  - → Filesystem (ext4, xfs, etc.)
  - → Mounted directory (e.g., /, /data)
  - → Files and subdirectories

The Filesystem as Interface to a Block Device

Silicon Graphics
3D File System
Navigator for IRIX

Filesystems in Linux

- Examples of Linux filesystems:
  - ext4 (common default)
  - xfs
  - btrfs
- Filesystems on devices have to be attached to the directory structure of a computer.
  - This is called ***mounting*** the filesystem.

Mounting a Filesystem

- Linux has a single unified directory tree starting at: / (root)
- Before mounting you may have: /data
  - This would just be an empty directory
- And you may have a block device: /dev/nvme1n1
- If the block device is new (or a new EBS volume), you have to create a filesystem for it.
- And then you have to mount the filesystem.

Creating and Mounting a Filesystem

- Creating a filesystem

```
sudo mkfs.ext4 /dev/nvme1n1
```

- Mounting a filesystem

```
sudo mount /dev/nvme1n1 /data
```

- Unmounting

```
sudo umount /data
```

Permissions

- You may (probably will) need to change ownership/permissions to use the new filesystem:

```
sudo chown -R $USER:$USER /data
```

- This is because by default the owner:group of the filesystem is root:root.

EBS

Every EC2 Instance Has a Root Filesystem

- When you launch an EC2 instance, it boots from a ***root volume***.
  - It is mounted at /
  - It contains:
    - The operating system
    - System libraries
    - Installed packages
    - Your home directory

The Problem: Instance Lifecycle vs Data Lifecycle

- An EC2 instance is:
  - A virtual machine
  - Compute + memory
  - Ephemeral by design
- You can:
  - Stop it
  - Terminate it
  - Replace it
- What happens to data stored in /home if the instance is terminated?

What happens to /home if an instance is terminated?

- Depending on configuration:
  - The root volume may be deleted
  - The instance disappears
  - Your data disappears
- Even if it persists, it is tightly coupled to that one instance.

- Idea: Compute should be replaceable. *Persistent data should not be tightly coupled to compute.*

Why We Need Separate Storage

- Suppose you are running:
  - A database
  - Gitea
  - A web application with user uploads
- If everything lives on the root volume:
  - Replacing the instance risks data loss
  - Scaling horizontally becomes difficult
  - Backups become messy
- Architectural principle in cloud systems: *Separate compute from storage.*

Amazon Elastic Block Store (EBS)

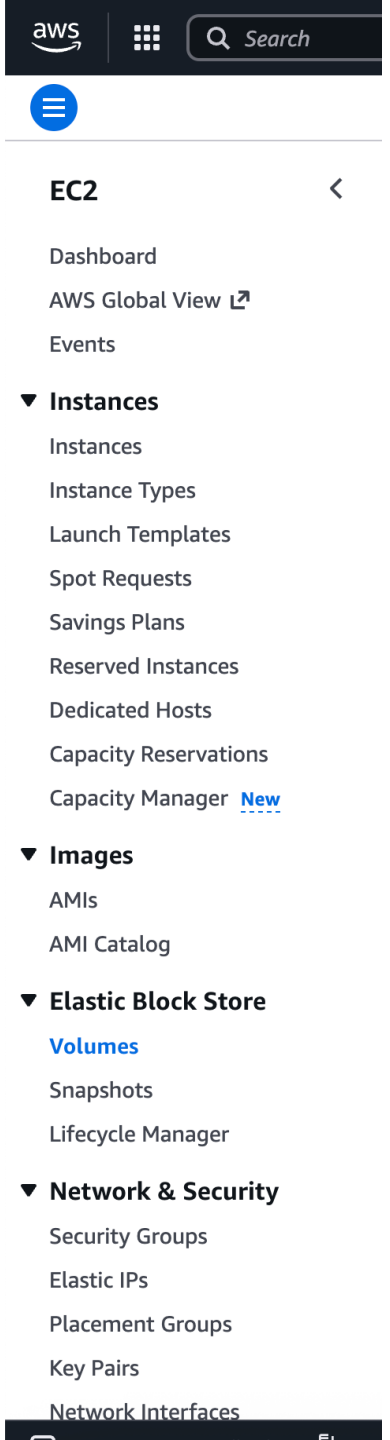
- EBS allows you to:
  - Attach storage independently of the instance lifecycle
  - Detach it
  - Reattach it to another instance
  - Snapshot it

EBS + EC2

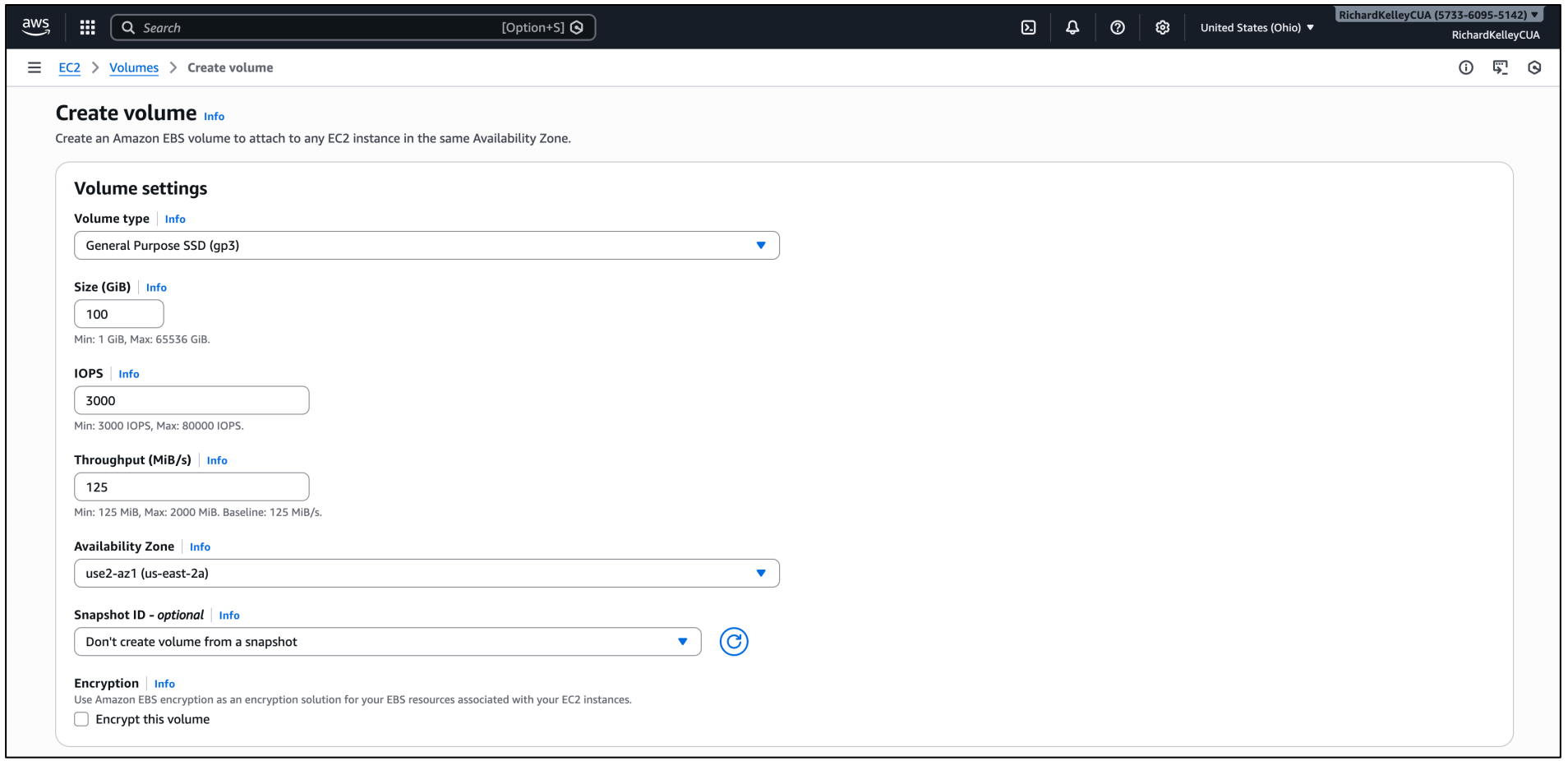
- Root volume:
  - Required for boot
  - OS + system state
  - Often disposable
- Additional EBS volumes:
  - Application data
  - Databases
  - Persistent state
- The root disk makes the machine run. EBS volumes make the data survive.

Creating an EBS Volume — Core Parameters

- You can do this through the AWS console or the CLI (more on the CLI later).
- When you create a volume, you are defining a block device with specific performance and location properties.
- Creating EBS volumes through the AWS console is done via EC2.



Creating a Volume



Creating a Volume

- Critical fields:
  - Volume Type
    - gp3 (general purpose, recommended default)
    - io1/io2 (high IOPS, databases)
    - st1/sc1 (throughput or cold HDD)
  - Size (GiB)
    - • Determines capacity
    - • Also affects performance limits for some types
  - Performance (gp3, io1, io2)
    - IOPS → how many operations per second
    - Throughput (gp3) → MB/s
  - Availability Zone (AZ)
    - Must match the EC2 instance AZ
    - Volumes are AZ-scoped
  - Snapshot (optional)
    - Empty volume (fresh block device)
    - Or restore from snapshot (pre-populated data)

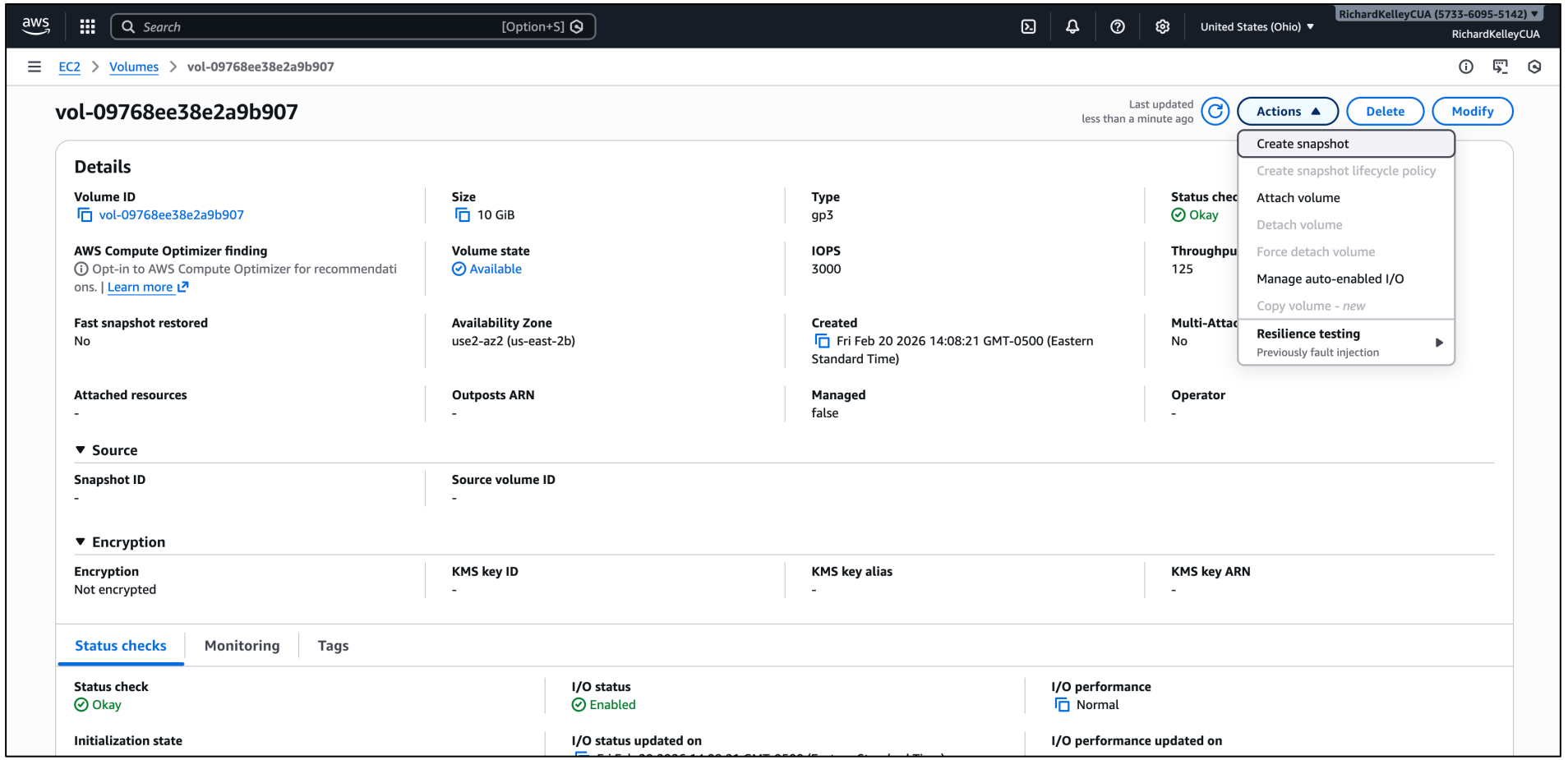
After Creation — How It Behaves

- Availability Zone Constraint
  - ***Volume and EC2 must be in same AZ***
  - EBS is not region-wide attachable
- Encryption
  - Often enabled by default
  - Transparent to the OS
  - No change to Linux commands
- Multi-Attach (io1/io2 only)
  - Specialized feature
  - Multiple instances can attach
  - Requires cluster-aware software
- Attachment
  - Volume must reach “available” state
  - Then attach to an EC2 instance
  - Appears inside Linux as a new block device

AWS Regions and Availability Zones

- Core structure of AWS infrastructure.
- Region
  - A geographic area (e.g., us-east-1, us-west-2)
  - Physically separated from other regions
  - Used for latency, compliance, and disaster recovery decisions
- Availability Zone (AZ)
  - A distinct data center (or group of data centers)
  - Multiple AZs per region (e.g., us-east-1a, 1b, 1c)
  - Independent power, networking, and cooling
- Why it matters:
  - EC2 instances run in a specific AZ
  - EBS volumes are created in a specific AZ
  - An EBS volume can only attach to an instance in the same AZ

Attaching a Volume



Once You Attach a Volume, You're Not Done

- The volume will show up as block device in your instance.
- But you don't have a filesystem yet, so you can't do anything.
- You have to create the filesystem, mount it, and then you're good to go.

Stopping, Termination, and Volumes

- You need to separate three lifecycles:
  - EC2 instance (compute)
  - Root EBS volume
  - Additional EBS volumes
- First: Instance Stop vs Terminate
  - Stop:
    - VM shuts down
    - Instance state = stopped
    - EBS volumes remain attached
    - No compute charges (storage charges continue)
  - Terminate:
    - VM is destroyed
    - Root volume is usually deleted (default setting)
    - Additional EBS volumes may or may not be deleted depending on configuration

EC2 vs EBS: What Do You Pay For?

- EC2 (compute) and EBS (storage) are billed independently.
- State of Instance vs What You Pay
  - Instance Running
    - Pay for compute
    - Pay for EBS storage
    - Pay for provisioned IOPS (if applicable)
  - Instance Stopped
    - Do NOT pay for compute
    - Still pay for EBS storage
    - Still pay for provisioned IOPS
  - Instance Terminated
    - Compute billing stops
    - Root volume usually deleted → no storage cost (if deleted)
    - Additional volumes continue billing unless deleted
  - Detached EBS Volume
    - Not attached to any instance
    - Still billed
    - Storage charges continue

Pricing Example

- gp3 storage: $0.08 per GiB-month
- You provision: 100 GiB
- You keep it for: 10 days
- No extra IOPS or throughput beyond baseline
- Step 1 — Convert time fraction
  - AWS bills per GiB-month, prorated by time.
  - 10 days ≈ 10 / 30 ≈ 1/3 of a month
- Step 2 — Compute monthly cost
  - 100 GiB × $0.08 per GiB-month = $8.00 per month
- Step 3 — Prorate for 10 days
  - $8.00 × (10 / 30)
    ≈ $8.00 × 0.333
    ≈ $2.67

S3

Not even an introduction…

Amazon S3 — Object Storage

- S3 is not a disk and not a filesystem.
- It is ***object storage***.
- Core concepts:
  - Bucket
    - A globally named container
    - Exists at the region level
  - Object
    - A blob of bytes
    - Identified by a key (string path-like name)
    - Can be up to very large sizes

AWS S3

- Key differences from a filesystem:
  - No block devices
  - No mounting (by default)
  - No POSIX semantics
  - Accessed via HTTP API
- You interact using:
  - AWS CLI
  - SDKs
  - REST API

```
aws s3 cp backup.tar.gz s3://my-bucket/
```

S3 vs EBS: Architectural Contrast

- EBS (Block Storage)
  - Appears as a block device under /dev
  - Must be formatted and mounted
  - Attached to one instance (typically)
  - Low latency
  - AZ-scoped
  - Capacity-based pricing
- Used for:
  - Databases
  - Application data directories
  - Operating systems

- S3 (Object Storage)
  - Accessed over network via API
  - Not mounted (normally)
  - Region-scoped
  - Extremely durable
  - Virtually unlimited scale
  - Usage-based pricing
- Used for:
  - Backups
  - Static assets
  - Logs
  - Data lakes

S3 Example (from assignment)

```
BUCKET="s3://YOUR-BUCKET-NAME/backups"
ARCHIVE="$(ls -t /tmp/gitea-backup-*.tar.gz | head -n 1)”

aws s3 cp "${ARCHIVE}" "${BUCKET}/"
aws s3 ls "${BUCKET}/"
```

Identity and Access Management IAM

Again, not even an introduction

The Problem: Credentials in the Cloud

- In cloud systems, applications need to call AWS APIs:
  - Upload to S3
  - Create snapshots
  - Read from DynamoDB
- Naïve approach:
  - Store access key and secret key on the server
  - Use `aws configure`
- Problems:
  - Keys stored on disk
  - Hard to rotate
  - Easy to leak
  - Security risk

Credentials in the Cloud

- Core principle: *Compute should not store long-term credentials.*

- ***IAM roles*** let compute access AWS without storing keys.

What Is an IAM Role?

- An IAM role is:
  - An AWS identity
  - With attached policies (permissions)
  - That can be assumed by trusted entities
- For EC2:
  - The EC2 instance “assumes” the role
  - AWS provides temporary credentials automatically
- Temporary credentials
  - Automatically rotated
  - Not stored permanently

How EC2 Uses IAM Roles

- Example Workflow:
  - Create IAM role
  - Attach S3 policy
  - Attach role to EC2 instance
- Then inside the instance:

```
aws s3 ls
```

- Works — without running aws configure.

Example

- Step 1 — Create IAM Role
  - IAM → Roles → Create role
  - Trusted entity: EC2
  - Attach policy (for example):
    - AmazonS3FullAccess
    - or a more restricted custom policy
- Step 2 — Attach Role to EC2 Instance
  - EC2 → Instance → Actions → Security → Modify IAM role
  - Select the role
- Step 3 — Done