Assignment: Persistent Gitea on EC2 (Docker + EBS + S3 Backup)

Due: February 27 in class (submit by email with GitHub repo link)

Overview

In this assignment you will deploy **Gitea** on an **AWS EC2** instance using **Docker**, store all persistent application data on a separate **EBS volume**, and implement a small **S3-based backup and restore** workflow using the **AWS CLI**.

The goal is for you to practice the core cloud idea: *separating compute (EC2) from state (EBS/S3)* so services can be rebuilt or moved without losing data.

Learning Objectives

By the end of this assignment, you should be able to:

- Attach, format, mount, and persistently mount an EBS volume on Linux.
- Run a stateful service in Docker with a host bind mount backed by EBS.
- Explain the difference between *block storage* (EBS) and *object storage* (S3).
- Store and retrieve backup artifacts in S3 without embedding credentials in scripts.
- Create a backup artifact from live service data and restore it to recover service state.

What You Will Build

A single-node deployment:

- One EC2 instance running Docker
- One EBS data volume mounted at `~/data` on the host
- Gitea container using a bind mount: `~/data` (host) → `/data` (container)
- An S3 bucket used for storing compressed backups (`.tar.gz`)
- A simple backup script and a restore procedure

Constraints / Requirements

- You must run Gitea **in Docker**.
- Persistent data must live on the **separate EBS volume**, not only on the root disk.

- Backups must be uploaded to **S3** using **AWS CLI** from the instance.
- Do not hardcode AWS credentials in scripts or commit secrets to your submission.

Part A: EC2 + EBS Persistent Data (Core)

A1. Provision EC2

Launch an EC2 instance (Ubuntu recommended). Ensure you can SSH into it.

Configure the Security Group minimally:

- TCP 22 (SSH): restrict to your IP if possible.
- TCP 3000 (Gitea web): for testing (or use a reverse proxy if you already have one).

A2. Create and Attach an EBS Volume

Create a new EBS volume in the same Availability Zone as your EC2 instance and attach it.

A3. Format and Mount

On the instance, identify the device name (e.g., `/dev/nvme1n1`).

Format it as ext4 and mount it at `~/data`. Example commands (you may adapt):

```
sudo lsblk
sudo mkfs.ext4 /dev/nvme1n1
mkdir -p ~/data
sudo mount /dev/nvme1n1 ~/data
df -h
```

A4. Make the Mount Persistent

Linux uses `/etc/fstab` as a startup mount table. Each line tells the OS what to mount, where to mount it, and with which options during boot.

Use the volume `UUID` (not device names like `/dev/nvme1n1`) because device names can change after reboot.

Configure `/etc/fstab` so the volume mounts at boot:

```
sudo blkid
echo $HOME
sudo nano /etc/fstab
# Add a line using your absolute home path (not ~)
# UUID=... /home/ubuntu/data ext4 defaults,nofail 0 2
# Example:
# UUID=1234-ABCD /home/ubuntu/data ext4 defaults,nofail 0 2
sudo mount -a
df -h
```

The `nofail` option helps the instance continue booting even if the volume is temporarily unavailable.

After `sudo mount -a` succeeds without errors, reboot the instance and verify `~/data` is still mounted.

Part B: Run Gitea via Docker with EBS-backed Bind Mount

B1. Install Docker

Install Docker on the instance. Confirm:

```
docker --version
docker ps
```

B2. Run Gitea

Run Gitea in a container such that Gitea's **data directory is mounted from host `~/data`**. Remember that a Docker container's writable layer is ephemeral. If you recreate the container, data stored only inside the container filesystem can be lost. A bind mount maps a host path into the container, so writes to container `/data` are actually stored on host `~/data` (your EBS volume).

You may use a Docker Compose file or `docker run`. If using `docker run`, it should include a bind mount of the form:

```
-v ~/data:/data
```

After starting the container, verify the mount is present:

```
docker inspect <gitea-container-name> --format '{{json .Mounts}}'
```

If the bind mount is missing, stop and recreate the container with the correct mount before continuing.

Once running:

- Complete the Gitea initial setup in the browser.
- Create a test repository.
- Make at least one commit (e.g., add a README).

B3. Persistence Test (Container Lifecycle)

Demonstrate that your repo survives:

- Stopping and restarting the container
- Removing and recreating the container (recreate it with the same `~/data` bind mount)

Part C: S3 Backup and Restore (AWS CLI)

C1. Create an S3 Bucket

Create an S3 bucket for backups. Name it something globally unique.

Recommended S3 settings:

- Block public access: ON
- Versioning: optional but encouraged
- Default encryption: optional but encouraged

C2. Use the Provided Backup Script

Use the following `backup.sh` script as-is so you can focus on the S3 workflow:

```
#!/usr/bin/env bash
set -euo pipefail

TS="$(date -u +%Y%m%dT%H%M%SZ)"
ARCHIVE="/tmp/gitea-backup-${TS}.tar.gz"

sudo tar -czf "${ARCHIVE}" -C "$HOME/data" .
echo "Created backup archive: ${ARCHIVE}"
```

Make it executable and run it:

```
chmod +x backup.sh
./backup.sh
```

C3. Upload Backup to S3 (AWS CLI)

Run `backup.sh`, then upload the newest archive to S3 from the instance:

```
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}/"
```

If AWS CLI is not installed yet, install it first:

```
sudo snap install aws-cli --classic
aws --version
```

If `aws s3 cp` fails with credentials errors, configure AWS credentials or attach an instance role with S3 access before continuing.

One possible verification workflow:

1. Run `aws s3 cp` to upload the tarball.
2. Run `aws s3 ls` on the same prefix.
3. Confirm the uploaded object appears in the listing.

C4. Restore Procedure

Write a restore procedure (a short set of steps) that:

- Downloads the chosen backup tarball from S3 using `aws s3 cp`
- Restores it back into `$HOME/data` (careful to stop the container first)
- Restarts the container and confirms the repo is present

At minimum, demonstrate restore by intentionally deleting a file or repo and recovering it from your backup.

Deliverables

Create a GitHub repository for this assignment and include:

1. **README.md** with:
   - Your architecture summary (1 short paragraph)
   - Step-by-step deployment instructions
   - Your backup and restore instructions
2. **docker-compose.yml** or the exact **docker run** command(s) used
3. **backup.sh** (and any helper scripts)
4. **Evidence** folder with screenshots or terminal transcripts showing:
   - `lsblk` showing the attached volume
   - `df -h` showing your home data mount (`/home/<user>/data`)
   - A Gitea repo visible in the web UI
   - `aws s3 ls` output showing uploaded backup object
   - Proof of persistence or restore (before/after)

After your repository is complete, email me the GitHub repository URL.

Notes and Hints

- Use `sudo chown` to ensure Docker/Gitea can write to `~/data` if you encounter permission issues.
- If you use a reverse proxy, ensure `ROOT_URL` (or equivalent) is configured appropriately for Gitea.
- Be careful when restoring: stop the container first to avoid inconsistent state.
- Keep security groups tight. Avoid opening unnecessary ports to the world.

Academic Integrity

You may consult official documentation and course resources. You may discuss high-level approaches with classmates, but your deployment, scripts, and write-up must be your own work. Cite any external sources you used in your README. You are encouraged to use AI tools to help you get unstuck or troubleshoot specific issues. If you use AI tools, include a brief note in your `README.md` describing what you used and how it helped.