Day 1 — Compute, Storage & IAM Practical Labs (CloudOps Style)

AWS CLI (Linux/macOS syntax) | Full step-by-step labs with verification & cleanup

Lab 1: EC2 Web Server Deployment (Apache)

Objective: Launch a hardened EC2 instance, install Apache, serve a static page, and validate access.
Architecture (textual):
`Internet -> Security Group (SSH, HTTP) -> EC2 (Amazon Linux 2) -> Apache -> /var/www/html`
Console Steps:
1. EC2 → Launch Instance → Select Amazon Linux 2 AMI → t3.micro (Free Tier).
2. Create or select a key pair (save pem file). Configure Security Group: allow SSH (your IP) and HTTP (0.0.0.0/0).
3. Launch and note Public IPv4 address.
CLI Steps (from your workstation):

```
# set variables
KEY=aws-key.pem
PUBIP=<public-ip>
# make key secure
chmod 400 ${KEY}

# SSH into instance
ssh -i ${KEY} ec2-user@${PUBIP}

# on instance: install and start Apache
sudo yum update -y
sudo yum install -y httpd
sudo systemctl enable --now httpd

# create simple index page
echo "Hello from EC2 - Day1 Lab" | sudo tee /var/www/html/index.html

# verify locally on instance
curl -I http://localhost
```

Verification:
- Visit http:// and confirm the page loads.
- From instance, `sudo systemctl status httpd` should show active (running).
Cleanup:

```
# Terminate instance (replace with actual instance-id)
aws ec2 terminate-instances --instance-ids i-xxxxxxxxxxxxxxx
# Optionally delete security group and key pair in console
```

Notes: For production-like setup consider using AMIs, enabling IAM roles for S3 access, and patch management.

Lab 2: IAM Role for EC2 Access to S3 (Read-Only)

Objective: Create an IAM role that allows EC2 instances to read from a specific S3 bucket, attach it to the instance, and verify access.
Architecture (textual):
```
EC2 Instance (assume Role) -> IAM Role (S3ReadOnly policy) -> S3 Bucket (data)
```
Console Steps:
1. IAM → Roles → Create role → AWS service → EC2 → Next.
2. Attach policy: AmazonS3ReadOnlyAccess (or custom policy limited to a bucket).
3. Name the role: EC2S3ReadOnlyRole. Attach role to the running EC2 instance via Actions → Security → Modify IAM Role.
CLI Steps (on EC2 with role attached):
```
# list s3 buckets (should succeed if role works)
aws s3 ls

# try to get an object (replace bucket/key)
aws s3 cp s3://my-bucket/path/to/object.txt ./object.txt
```
Verification:
- `aws s3 ls` returns buckets. If role is scoped to bucket, `aws s3 ls s3://bucket-name` should list objects.
Cleanup:
```
# detach role via console and delete role when no longer needed
# delete any test objects you uploaded to S3
aws s3 rm s3://my-bucket/path/to/object.txt
```
Notes: Prefer least-privilege; create an inline policy constrained to specific bucket ARNs.

Lab 3: S3 Cross-Region Replication (CRR) with Versioning

Objective: Configure S3 buckets for CRR to replicate objects from source region to destination region, with versioning enabled.

Architecture (textual):

```
Uploader -> Source S3 Bucket (us-east-1, versioning ON) --CRR--> Destination S3 Bucket (us-west-
```

Console Steps:

1. Create source bucket in Region A and destination bucket in Region B.
2. Enable Versioning on both buckets (Properties → Versioning).
3. Set up a replication rule on the source bucket: destination bucket ARN, IAM role to allow replication.

CLI Steps:

```
# create buckets (example)
aws s3api create-bucket --bucket my-source-bucket-<id> --region us-east-1 --create-bucket-config
aws s3api create-bucket --bucket my-dest-bucket-<id> --region us-west-2 --create-bucket-configur

# enable versioning
aws s3api put-bucket-versioning --bucket my-source-bucket-<id> --versioning-configuration Status
aws s3api put-bucket-versioning --bucket my-dest-bucket-<id> --versioning-configuration Status=E
```

Verification:

- Upload object to source bucket and confirm it appears in destination bucket with same key and a replicated timestamp.

Cleanup:

```
# remove objects and delete both buckets
aws s3 rm s3://my-source-bucket-<id> --recursive
aws s3 rm s3://my-dest-bucket-<id> --recursive
aws s3api delete-bucket --bucket my-source-bucket-<id> --region us-east-1
aws s3api delete-bucket --bucket my-dest-bucket-<id> --region us-west-2
```

Notes: CRR has costs; ensure source and destination have appropriate lifecycle rules for storage class transitions.

Lab 4: EBS Snapshot, Restore & Mount

Objective: Create an EBS snapshot, create a new volume from the snapshot, attach it to a secondary instance and mount it.

Architecture (textual):

```
Primary EC2 (volume) -> Snapshot -> New Volume -> Attach to Secondary EC2 -> Mount under /mnt/re
```

Steps (Console + CLI):

1. Identify the volume ID of the primary instance: `lsblk` or EC2 console.
2. Create snapshot:

```
# create snapshot
aws ec2 create-snapshot --volume-id vol-0123456789abcdef0 --description "backup-snapshot"

# after snapshot completes, create volume in same AZ as secondary instance
aws ec2 create-volume --snapshot-id snap-0123456789abcdef0 --availability-zone us-east-1a --volu

# attach to secondary instance
aws ec2 attach-volume --volume-id vol-0abcdef1234567890 --instance-id i-0abcdef1234567890 --devi
```

On secondary instance (mount):

```
# create mount point and mount (example)
sudo mkdir -p /mnt/restore
sudo mount /dev/xvdh1 /mnt/restore   # or appropriate device name
ls /mnt/restore
```

Verification:

- Files from original volume should be visible under /mnt/restore.

Cleanup:

```
# detach and delete the new volume
aws ec2 detach-volume --volume-id vol-0abcdef1234567890
aws ec2 delete-volume --volume-id vol-0abcdef1234567890
# delete snapshot
aws ec2 delete-snapshot --snapshot-id snap-0123456789abcdef0
```

Notes: Snapshots are incremental and stored in S3 under the hood. Consider lifecycle policies to expire old snapshots.

Lab 5: EFS Setup and Mount on EC2

Objective: Create an EFS file system, mount targets in each AZ, and mount it from an EC2 instance (shared storage).
Architecture (textual):
`EC2 Instances (multiple AZs) -> Mount Targets -> EFS (NFSv4) -> Shared /mnt/efs`
Console Steps:
1. EFS → Create file system → Choose VPC → create mount targets in required subnets.
2. Security groups: allow inbound NFS (2049) from EC2 SGs.
CLI Steps (on EC2):

```
# install NFS client (Amazon Linux)
sudo yum install -y amazon-efs-utils
# create mount point and mount using TLS (recommended)
sudo mkdir -p /mnt/efs
sudo mount -t efs -o tls fs-0123456789abcdef0:/ /mnt/efs

# verify
df -h | grep efs
echo "shared file" | sudo tee /mnt/efs/testfile.txt
cat /mnt/efs/testfile.txt
```

Verification:
- File written on one EC2 should be visible to other EC2s mounting same EFS.
Cleanup:

```
# delete test file and EFS via console
sudo rm /mnt/efs/testfile.txt
# Delete EFS in console once unmounted
```

Notes: EFS offers different performance modes. Use throughput mode and performance mode appropriate to workload.