AMAZON WEB SERVICES (AWS)

Amazon EC2 — Networking, Integrations & Automation

Part 2: Practical Lab Manual | Intermediate EC2 Use Cases 31–60

Lab detail	Description
Level	Intermediate hands-on EC2 networking, AWS service integration, high availability & IaC
Services	EC2, VPC, RDS, S3, DynamoDB, SNS, SES, SQS, EFS, ALB, Auto Scaling, Route 53, ACM, CloudWatch, Systems Manager, AWS Backup, Ansible, Terraform
Methods	AWS Management Console, SSH/SSM Session Manager, and AWS CLI / IaC Tools
Suggested Region	ap-south-1 (Mumbai), unless your lab uses another Region
Goal	Integrate EC2 with managed AWS databases/storage/messaging, secure private subnets, scale across AZs, and automate ops

How to use this manual

- Complete all labs in a dedicated sandbox or training AWS account. Do not experiment on production infrastructure.
- Replace placeholders such as ACCOUNT_ID, VPC_ID, PUBLIC_SUBNET_ID, PRIVATE_SUBNET_ID, INSTANCE_ID, RDS_ENDPOINT, BUCKET_NAME, ALB_DNS, and DOMAIN_NAME with your actual values.
- Attach least-privilege IAM instance profiles to EC2 instances instead of embedding IAM user access keys on servers.
- Paid-tier note: Resources like Amazon RDS, EFS, ALB, NAT Gateway/Instance, and AWS Backup incur hourly or storage charges beyond Free Tier limits—clean them up immediately after finishing your lab session.

Suggested lab naming

Item	Suggested value
VPC & Subnets	Lab-VPC (10.0.0.0/16), Lab-Public-Subnet-1a, Lab-Private-Subnet-1a
Web / App Instances	Lab-EC2-Web-01, Lab-EC2-Private-01, Lab-Bastion-Host, Lab-NAT-Instance
Database & Storage	lab-mysql-db (RDS), lab-app-bucket-ACCOUNT_ID (S3), Lab-App-Table (DynamoDB), Lab-Shared-EFS
Load Balancing & ASG	Lab-App-ALB, Lab-Web-TG (Target Group), Lab-Web-ASG

Item	Suggested value
IAM Instance Profile	Lab-EC2-App-Integration-Role
Region	ap-south-1

Quick reference — Part 2: Intermediate EC2 use cases (31–60)

No.	Use case	Practical implementation
31	Host WordPress	Run WordPress using EC2, a web server and a database.
32	Host multiple websites	Configure Nginx or Apache virtual hosts.
33	Application with RDS	Connect an EC2-hosted application to Amazon RDS for MySQL.
34	Application with S3	Upload and retrieve application files from Amazon S3.
35	Application with DynamoDB	Read and write application data in DynamoDB.
36	Email notifications	Send application notifications using Amazon SNS or SES.
37	Message processing	Process messages from Amazon SQS.
38	Private EC2 instance	Launch an instance without a public IP address.
39	Bastion host	Provide controlled administrative access to private instances.
40	NAT instance	Provide outbound connectivity for private subnets using a self-managed NAT instance.
41	Configure VPC peering	Connect workloads across VPCs using VPC peering.
42	Configure VPN access	Connect on-premises networks to AWS using VPN connectivity.
43	Configure EFS	Share a file system across multiple EC2 instances.
44	Configure NFS server	Share files between Linux systems using NFS.
45	Configure load balancing	Distribute application traffic using an Application Load Balancer.
46	Configure Auto Scaling	Automatically add or remove EC2 capacity based on demand.
47	Multi-AZ application	Deploy application instances across multiple Availability Zones.
48	Configure Route 53	Route a domain name to an application endpoint.
49	Configure HTTPS	Terminate TLS at an Application Load Balancer using ACM certificates.
50	Configure SSL on EC2	Install and manage certificates on a web server when TLS terminates on the instance.
51	Configure CloudWatch alarms	Alert on high CPU utilization or other monitored metrics.
52	Configure SNS alerts	Send notifications when monitoring alarms trigger.

No.	Use case	Practical implementation
53	Configure Systems Manager	Manage EC2 instances without relying on inbound SSH.
54	Patch management	Automate operating-system patching using Systems Manager.
55	Configure backup automation	Schedule EBS and application backups using AWS Backup.
56	Configure log management	Send application and operating-system logs to CloudWatch Logs.
57	S3 lifecycle integration	Archive backups and logs using S3 lifecycle policies.
58	Deploy a Git-based application	Pull application code from GitHub and deploy it to EC2.
59	Configure Ansible	Automate EC2 configuration and application setup.
60	Configure Terraform	Provision EC2 infrastructure using infrastructure as code.

Before starting

- Ensure you have completed Part 1 fundamentals (launching Linux EC2, SSH, Security Groups, and IAM Roles).
- Prepare a VPC with at least two public subnets and two private subnets across two Availability Zones (ap-south-1a and ap-south-1b) for ALB, RDS, and Multi-AZ labs.
- Verify AWS CLI v2 is installed locally and on your management instance (aws sts get-caller-identity).

Part 2 — Intermediate EC2 practical lab implementations (31–60)

Use Case 31 — Host WordPress

Objective: Deploy a full WordPress CMS site on EC2 backed by Apache, PHP, and a MySQL/MariaDB database.

Prerequisites: Linux EC2 instance (t3.micro or t3.small) with HTTP (TCP 80) and SSH (TCP 22) allowed.

Console implementation

1. Launch an Amazon Linux 2023 EC2 instance with inbound HTTP (port 80) open in its Security Group.

2. Connect via SSH and install Apache (httpd), PHP with php-mysqlnd, and MariaDB server (or connect to Amazon RDS).

3. Start httpd and mariadb, secure the database, and create a wordpress database and user.

4. Download and extract the latest WordPress archive into /var/www/html/ and configure wp-config.php.

5. Open http://PUBLIC_IP in your browser and complete the WordPress 5-minute web installation wizard.

CLI / Configuration examples

```
sudo dnf install -y httpd php php-mysqlnd php-gd php-xml mariadb105-server wget
sudo systemctl enable --now httpd mariadb
sudo mysql -e "CREATE DATABASE wordpress; CREATE USER 'wpuser'@'localhost' IDENTIFIED BY 'StrongPass123!';
GRANT ALL PRIVILEGES ON wordpress.* TO 'wpuser'@'localhost'; FLUSH PRIVILEGES;"
cd /var/www/html && sudo wget https://wordpress.org/latest.tar.gz && sudo tar -xzf latest.tar.gz --strip-
```

```
components=1
sudo chown -R apache:apache /var/www/html
```

Verification

- Navigate to http://PUBLIC_IP/wp-admin, log in, and publish a test post.

Expected result: WordPress front-end and admin dashboard load and persist posts in the MySQL/MariaDB database.

> **Security & operational best practice:** For production WordPress, decouple the database onto Amazon RDS Multi-AZ and store wp-content/uploads on Amazon EFS or S3.

Use Case 32 — Host multiple websites

Objective: Configure name-based Virtual Hosts in Nginx or Apache so one EC2 instance serves multiple domains.

Prerequisites: Linux EC2 instance with Nginx or Apache installed and TCP port 80 open.

Console implementation

1. Connect via SSH and create separate document roots: /var/www/site1 and /var/www/site2 with distinct index.html files.
2. Create two server block configuration files inside /etc/nginx/conf.d/ (site1.conf and site2.conf) with distinct server_name directives.
3. Test the Nginx configuration syntax with nginx -t and reload the service.

CLI / Configuration examples

```
sudo mkdir -p /var/www/site1 /var/www/site2
echo "<h1>Welcome to Site One</h1>" | sudo tee /var/www/site1/index.html
echo "<h1>Welcome to Site Two</h1>" | sudo tee /var/www/site2/index.html
cat << 'EOF' | sudo tee /etc/nginx/conf.d/vhosts.conf
server { listen 80; server_name site1.example.com; root /var/www/site1; index index.html; }
server { listen 80; server_name site2.example.com; root /var/www/site2; index index.html; }
EOF
sudo nginx -t && sudo systemctl reload nginx
```

Verification

- Test using curl -H "Host: site1.example.com" http://PUBLIC_IP and curl -H "Host: site2.example.com" http://PUBLIC_IP.

Expected result: Each Host header routes to its corresponding document root and returns its unique webpage.

> **Security & operational best practice:** Maintain separate access/error log files and TLS certificates per virtual host for isolated troubleshooting and security.

Use Case 33 — Application with RDS

Objective: Connect an application running on an EC2 instance to a managed Amazon RDS for MySQL database.

Prerequisites: EC2 instance and Amazon RDS MySQL instance in the same VPC.

Console implementation

1. Open RDS → Create database → Standard create → MySQL → Free tier template (db.t3.micro).
2. Configure DB instance identifier (lab-mysql-db), master username (admin), and password; place it in private subnets.
3. Configure the RDS Security Group inbound rules to allow MySQL/Aurora (TCP 3306) ONLY from the EC2 instance's Security Group ID.
4. Connect to EC2 via SSH, install the MariaDB/MySQL client, and connect to the RDS endpoint.

CLI / Configuration examples

```
sudo dnf install -y mariadb105
mysql -h RDS_ENDPOINT -P 3306 -u admin -p -e "SHOW DATABASES; CREATE DATABASE appdb;"
```

Verification

- Confirm the MySQL prompt connects cleanly and SHOW DATABASES lists appdb.

Expected result: EC2 application communicates privately with Amazon RDS over port 3306 using Security Group chaining.

> **Security & operational best practice:** Store RDS credentials in AWS Secrets Manager rather than hardcoding database passwords in application source files.

Use Case 34 — Application with S3

Objective: Enable an EC2-hosted application to upload and download objects from Amazon S3 using an IAM Instance Profile.

Prerequisites: Private S3 bucket and an EC2 instance.

Console implementation

1. Create an S3 bucket named lab-app-bucket-ACCOUNT_ID with Block All Public Access enabled.
2. Create an IAM Role for EC2 with a least-privilege policy allowing s3:PutObject, s3:GetObject, and s3:ListBucket on lab-app-bucket-ACCOUNT_ID.
3. Attach the IAM Role to your EC2 instance via Actions → Security → Modify IAM role.
4. SSH into EC2 and use AWS CLI or Python (boto3) to upload and retrieve files.

CLI / Configuration examples

```
echo "Sample application payload" > payload.txt
aws s3 cp payload.txt s3://lab-app-bucket-ACCOUNT_ID/data/payload.txt
aws s3 ls s3://lab-app-bucket-ACCOUNT_ID/data/
aws s3 cp s3://lab-app-bucket-ACCOUNT_ID/data/payload.txt ./downloaded_payload.txt
```

Verification

- Confirm the object appears in the S3 Console under data/payload.txt and downloads cleanly back to EC2.

Expected result: EC2 reads and writes S3 objects seamlessly via temporary IAM role credentials without public bucket access.

> **Security & operational best practice:** Configure a VPC Gateway Endpoint for S3 so traffic between EC2 and S3 stays on the private AWS network without NAT costs.

Use Case 35 — Application with DynamoDB

Objective: Read and write NoSQL items in Amazon DynamoDB from an EC2 instance using an IAM role.

Prerequisites: EC2 instance with an IAM role granting dynamodb:PutItem, GetItem, Query, and Scan on the target table.

Console implementation

1. Open DynamoDB → Tables → Create table → Name: Lab-App-Table, Partition key: UserId (String) → Create table.

2. Attach an IAM policy allowing DynamoDB read/write actions on Lab-App-Table to your EC2 IAM role.

3. SSH into EC2 and insert/query records via AWS CLI or boto3.

CLI / Configuration examples

```
aws dynamodb put-item --table-name Lab-App-Table --item '{"UserId": {"S": "user-101"}, "Role": {"S": "Architect"}}' --region ap-south-1
aws dynamodb get-item --table-name Lab-App-Table --key '{"UserId": {"S": "user-101"}}' --region ap-south-1
```

Verification

- Check the DynamoDB Console → Explore table items to confirm user-101 appears.

Expected result: EC2 application performs low-latency NoSQL read/write operations against DynamoDB.

> **Security & operational best practice:** Use VPC Gateway Endpoints for DynamoDB and scope IAM resource ARNs to specific table names.

Use Case 36 — Email notifications

Objective: Publish application notifications from EC2 using Amazon SNS (or send transactional emails via Amazon SES).

Prerequisites: EC2 IAM role with sns:Publish (or ses:SendEmail) permissions and a verified email subscription.

Console implementation

1. Open Amazon SNS → Topics → Create topic → Standard → Name: Lab-App-Notifications.

2. Create an Email subscription with your email address and click the confirmation link in your inbox.

3. Attach sns:Publish permission for the topic ARN to your EC2 IAM role.

4. Publish a test notification from the EC2 instance.

CLI / Configuration examples

```
aws sns publish --topic-arn arn:aws:sns:ap-south-1:ACCOUNT_ID:Lab-App-Notifications --subject "EC2 App Alert" --message "Batch job completed successfully on EC2." --region ap-south-1
```

Verification

- Check your email inbox to confirm delivery of the notification message.

Expected result: Application events on EC2 trigger immediate email notifications via Amazon SNS.

> **Security & operational best practice:** Use Amazon SES when you need rich HTML customer emails, custom sender domains (DKIM/SPF), and bounce/complaint tracking.

Use Case 37 — Message processing

Objective: Decouple application workloads by polling, processing, and deleting messages from an Amazon SQS queue on EC2.

Prerequisites: EC2 IAM role with sqs:SendMessage, sqs:ReceiveMessage, and sqs:DeleteMessage permissions.

Console implementation

1. Open Amazon SQS → Create queue → Standard → Name: Lab-Work-Queue → Create queue.

2. Send a test JSON message to Lab-Work-Queue from the console or CLI.

3. On EC2, run a worker script that receives the message, processes it, and deletes it using its ReceiptHandle.

CLI / Configuration examples

```
QUEUE_URL=$(aws sqs get-queue-url --queue-name Lab-Work-Queue --query 'QueueUrl' --output text --region ap-south-1)
aws sqs send-message --queue-url "$QUEUE_URL" --message-body '{"orderId": 5001, "action": "resize_image"}' --region ap-south-1
aws sqs receive-message --queue-url "$QUEUE_URL" --max-number-of-messages 1 --wait-time-seconds 10 --region ap-south-1
```

Verification

- After deleting the processed message with delete-message, confirm Messages available returns to 0 in SQS.

Expected result: EC2 worker reliably consumes and acknowledges asynchronous jobs from SQS.

> **Security & operational best practice:** Enable SQS Long Polling (ReceiveMessageWaitTimeSeconds = 20) and configure a Dead-Letter Queue (DLQ) for failed messages.

Use Case 38 — Private EC2 instance

Objective: Launch an isolated EC2 instance inside a private subnet with no public IPv4 address for backend workloads.

Prerequisites: VPC with a private subnet whose route table has no direct route to an Internet Gateway (IGW).

Console implementation

1. Open EC2 → Instances → Launch instances → Name: Lab-EC2-Private-01.
2. Under Network settings, select your VPC and choose Lab-Private-Subnet-1a.
3. Set Auto-assign public IP to Disable.
4. Select a Security Group that allows SSH (port 22) ONLY from your Bastion Host Security Group (or VPC CIDR).
5. Click Launch instance.

CLI / Configuration examples

```
aws ec2 run-instances --image-id AMI_ID --instance-type t3.micro --key-name lab-ec2-keypair --subnet-id PRIVATE_SUBNET_ID --no-associate-public-ip-address --security-group-ids PRIVATE_SG_ID --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=Lab-EC2-Private-01}]' --region ap-south-1
```

Verification

- Inspect instance details and confirm Public IPv4 address is blank while Private IPv4 address is assigned.

Expected result: Backend EC2 instance is completely unreachable directly from the public internet.

> **Security & operational best practice:** Always place application servers and databases in private subnets; expose only Load Balancers or Bastions publicly.

Use Case 39 — Bastion host

Objective: Deploy a hardened jump server in a public subnet to provide controlled SSH access to private EC2 instances.

Prerequisites: Public subnet, private subnet, and Lab-EC2-Private-01 running.

Console implementation

1. Launch Lab-Bastion-Host in Lab-Public-Subnet-1a with a public IP and inbound SSH (port 22) restricted to My IP.
2. Update Lab-EC2-Private-01's Security Group to allow inbound SSH (port 22) from the Bastion Host's Security Group ID.
3. Use SSH Agent Forwarding (ssh -A) from your local terminal to connect to the Bastion Host, then SSH to the private instance's private IP.

CLI / Configuration examples

```
# On local machine, add key to SSH agent and connect with agent forwarding (-A):
ssh-add lab-ec2-keypair.pem
```

```
ssh -A ec2-user@BASTION_PUBLIC_IP

# From inside the Bastion Host, hop to the private instance:
ssh ec2-user@PRIVATE_INSTANCE_IP
```

Verification

- Confirm you reach the shell prompt of Lab-EC2-Private-01 without copying your .pem key onto the Bastion Host.

Expected result: Administrators securely reach private subnet instances through an audited single entry point.

> **Security & operational best practice:** Never store private .pem files on the Bastion Host disk; use SSH agent forwarding or replace Bastions with AWS Systems Manager Session Manager.

Use Case 40 — NAT instance

Objective: Configure a self-managed NAT EC2 instance in a public subnet so private instances can download OS updates outbound.

Prerequisites: Public subnet with IGW route and private subnet with a custom route table.

Console implementation

1. Launch an Amazon Linux EC2 instance (Lab-NAT-Instance) in Lab-Public-Subnet-1a with a Public IP.
2. Select Lab-NAT-Instance → Actions → Networking → Change source/destination check → Check Stop (disable source/dest check) → Save.
3. Configure its Security Group to allow inbound HTTP/HTTPS/All TCP from the private subnet CIDR (e.g., 10.0.2.0/24).
4. SSH into Lab-NAT-Instance and enable IPv4 forwarding and iptables MASQUERADE.
5. In VPC → Route Tables, edit the private subnet route table and add route 0.0.0.0/0 → Target: Instance (Lab-NAT-Instance).

CLI / Configuration examples

```
# Run on Lab-NAT-Instance:
sudo dnf install -y iptables-services
echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.conf && sudo sysctl -p
PRIMARY_IF=$(ip route | grep default | awk '{print $5}')
sudo iptables -t nat -A POSTROUTING -o $PRIMARY_IF -j MASQUERADE
sudo service iptables save
```

Verification

- From Lab-EC2-Private-01 (via Bastion), run curl -I https://aws.amazon.com to verify outbound internet access.

Expected result: Private instances initiate outbound internet connections while remaining unreachable from inbound internet traffic.

> **Security & operational best practice:** For production workloads, prefer a managed AWS NAT Gateway over a self-managed NAT instance for high availability and bandwidth.

Use Case 41 — Configure VPC peering

Objective: Establish private IP routing between two non-overlapping VPCs using an AWS VPC Peering connection.

Prerequisites: Two VPCs with non-overlapping CIDR blocks (e.g., VPC-A: 10.0.0.0/16 and VPC-B: 172.16.0.0/16).

Console implementation

1. Open VPC → Peering connections → Create peering connection.
2. Select Requester VPC (VPC-A) and Accepter VPC (VPC-B), then click Create peering connection.
3. Select the peering connection → Actions → Accept request.
4. Update Route Tables in both VPCs: add 172.16.0.0/16 → pcx-id in VPC-A, and 10.0.0.0/16 → pcx-id in VPC-B.
5. Update Security Groups in both VPCs to allow traffic from the peer CIDR block.

CLI / Configuration examples

```
aws ec2 create-vpc-peering-connection --vpc-id VPC_A_ID --peer-vpc-id VPC_B_ID --region ap-south-1
aws ec2 accept-vpc-peering-connection --vpc-peering-connection-id PCX_ID --region ap-south-1
```

Verification

- Ping or SSH across private IP addresses between an EC2 instance in VPC-A and an instance in VPC-B.

Expected result: Traffic routes privately between both VPCs over the AWS backbone without traversing the public internet.

> **Security & operational best practice:** Remember VPC peering is non-transitive; use AWS Transit Gateway when connecting three or more VPCs at scale.

Use Case 42 — Configure VPN access

Objective: Connect a remote network or workstation securely to a private VPC using AWS Site-to-Site VPN or Client VPN.

Prerequisites: VPC with private subnets and remote public gateway IP (for Site-to-Site VPN) or ACM certificates (for Client VPN).

Console implementation

1. Open VPC → Virtual Private Network (VPN) → Customer gateways → Create customer gateway with your remote router's static public IP.
2. Open Virtual private gateways → Create virtual private gateway and attach it to your Lab-VPC.
3. Open Site-to-Site VPN connections → Create VPN connection linking the VGW and CGW (select Static routing and enter remote CIDR).

4. Enable Route Propagation on your VPC Route Table and download the router configuration file.

CLI / Configuration examples

```
aws ec2 create-vpn-connection --type ipsec.1 --customer-gateway-id CGW_ID --vpn-gateway-id VGW_ID --options '{"StaticRoutesOnly":true}' --region ap-south-1
```

Verification

- Confirm Tunnel 1 or Tunnel 2 Status transitions to UP in the Site-to-Site VPN console.

Expected result: Encrypted IPsec tunnels link the remote network to private EC2 instances inside AWS.

> **Security & operational best practice:** Configure both IPSec tunnels in active/passive or BGP ECMP mode for high availability across AWS endpoints.

Use Case 43 — Configure EFS

Objective: Create an elastic, multi-AZ Amazon EFS shared file system and mount it concurrently on multiple EC2 instances.

Prerequisites: Two Linux EC2 instances and a Security Group allowing inbound NFS (TCP 2049) from the EC2 Security Group.

Console implementation

1. Open EFS → File systems → Create file system → Name: Lab-Shared-EFS, select your VPC, and click Create.

2. Ensure the EFS mount targets use a Security Group allowing inbound TCP port 2049 from your EC2 instances.

3. SSH into both EC2 instances, install amazon-efs-utils, and mount the EFS file system to /mnt/efs.

CLI / Configuration examples

```
sudo dnf install -y amazon-efs-utils
sudo mkdir -p /mnt/efs
sudo mount -t efs -o tls EFS_FILE_SYSTEM_ID:/ /mnt/efs
echo "Shared across EC2 fleet" | sudo tee /mnt/efs/shared_test.txt
```

Verification

- On the second EC2 instance, run cat /mnt/efs/shared_test.txt and confirm the file appears immediately.

Expected result: Multiple EC2 instances read and write concurrently to a shared, auto-scaling POSIX file system.

> **Security & operational best practice:** Enable Encryption in transit (-o tls) and Encryption at rest, and add the EFS mount entry to /etc/fstab with _netdev.

Use Case 44 — Configure NFS server

Objective: Configure a self-managed Linux EC2 instance as an NFS server exporting a directory to Linux client instances.

Prerequisites: Two Linux EC2 instances in the same VPC; NFS server SG allows TCP/UDP 2049 from the client's private IP/SG.

Console implementation

1. On the NFS Server EC2 instance, install nfs-utils, create /srv/nfs_share, and configure /etc/exports.

2. Start and enable nfs-server and verify exported paths using exportfs -v.

3. On the NFS Client EC2 instance, install nfs-utils and mount SERVER_PRIVATE_IP:/srv/nfs_share to /mnt/nfs.

CLI / Configuration examples

```
# On NFS Server:
sudo dnf install -y nfs-utils
sudo mkdir -p /srv/nfs_share && sudo chmod 777 /srv/nfs_share
echo "/srv/nfs_share 10.0.0.0/16(rw,sync,no_root_squash)" | sudo tee /etc/exports
sudo exportfs -rav && sudo systemctl enable --now nfs-server

# On NFS Client:
sudo dnf install -y nfs-utils && sudo mkdir -p /mnt/nfs
sudo mount -t nfs SERVER_PRIVATE_IP:/srv/nfs_share /mnt/nfs
```

Verification

- Create a file in /mnt/nfs on the client and verify it appears in /srv/nfs_share on the server.

Expected result: Linux EC2 client mounts and shares files over NFSv4 with the EC2 NFS server.

> **Security & operational best practice:** Restrict /etc/exports CIDR ranges strictly to trusted subnets and never expose port 2049 publicly.

Use Case 45 — Configure load balancing

Objective: Distribute incoming HTTP application traffic across multiple EC2 instances using an Application Load Balancer (ALB).

Prerequisites: VPC with at least two public subnets in different Availability Zones and two web-serving EC2 instances.

Console implementation

1. Open EC2 → Load Balancing → Target Groups → Create target group → Instances → Name: Lab-Web-TG, Protocol HTTP:80 → Register your two EC2 instances.

2. Open Load Balancers → Create load balancer → Application Load Balancer → Name: Lab-App-ALB, Internet-facing.

3. Select two public subnets (ap-south-1a and ap-south-1b) and a Security Group allowing inbound HTTP port 80 from 0.0.0.0/0.

4. Configure Listener HTTP:80 to forward traffic to Lab-Web-TG and click Create load balancer.

5. Update the EC2 instances' Security Group to accept HTTP port 80 traffic from the ALB's Security Group.

CLI / Configuration examples

```
aws elbv2 describe-target-health --target-group-arn TARGET_GROUP_ARN --region ap-south-1
```

Verification

- Wait until Target Group health status shows Healthy, then browse to http://ALB_DNS_NAME and refresh to observe load distribution.

Expected result: ALB routes incoming requests evenly across healthy backend EC2 instances and automatically stops routing to unhealthy ones.

> **Security & operational best practice:** Lock down backend EC2 Security Groups so they only accept HTTP traffic from the ALB Security Group, bypassing direct IP access.

Use Case 46 — Configure Auto Scaling

Objective: Automatically scale EC2 instance count up and down based on CPU utilization using an Auto Scaling Group (ASG).

Prerequisites: Launch Template with web server User Data, multi-AZ subnets, and an ALB Target Group.

Console implementation

1. Open EC2 → Auto Scaling Groups → Create Auto Scaling group → Name: Lab-Web-ASG.

2. Select your Launch Template (Lab-Web-Launch-Template) → Next.

3. Select your VPC and at least two subnets across different AZs → Next.

4. Attach to existing load balancer Target Group (Lab-Web-TG) and enable ELB health checks → Next.

5. Set Desired capacity: 2, Min: 1, Max: 4, and add a Target tracking scaling policy keeping Average CPU Utilization at 50%.

6. Review and click Create Auto Scaling group.

CLI / Configuration examples

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names Lab-Web-ASG --region ap-south-1
```

Verification

- Terminate one ASG instance manually or run stress-ng to spike CPU above 50%, and observe ASG launching replacement/additional capacity.

Expected result: Auto Scaling automatically maintains desired fleet health and scales capacity dynamically under load.

> **Security & operational best practice:** Always configure both a minimum capacity >= 2 across multiple AZs and a sensible maximum capacity cap to prevent runaway billing.

Use Case 47 — Multi-AZ application

Objective: Deploy an application across multiple Availability Zones (ap-south-1a and ap-south-1b) for single-AZ fault tolerance.

Prerequisites: VPC with subnets in at least two AZs, an ALB, and an Auto Scaling Group.

Console implementation

1. Verify your ALB and Auto Scaling Group span at least two Availability Zones (ap-south-1a and ap-south-1b).

2. Configure each instance's webpage (via IMDSv2) to display its Availability Zone name.

3. Simulate an AZ or instance failure by stopping the instance in ap-south-1a.

CLI / Configuration examples

```
# Add AZ display to instance index.html using IMDSv2:
TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
AZ=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/placement/availability-zone)
echo "<h1>Served from Availability Zone: $AZ</h1>" | sudo tee /var/www/html/index.html
```

Verification

- Refresh http://ALB_DNS_NAME repeatedly to see responses alternating between ap-south-1a and ap-south-1b, and confirm zero downtime when one instance stops.

Expected result: Application remains continuously available even if an entire Availability Zone experiences an outage.

> **Security & operational best practice:** Enable Cross-Zone Load Balancing and pair Multi-AZ EC2 tiers with Multi-AZ RDS deployments.

Use Case 48 — Configure Route 53

Objective: Map a custom DNS domain name to an Application Load Balancer or EC2 instance using Amazon Route 53.

Prerequisites: Registered domain name and a Route 53 Public Hosted Zone.

Console implementation

1. Open Route 53 → Hosted zones → Select your domain (e.g., example.com).

2. Click Create record → Record name: app (creating app.example.com), Record type: A.

3. Toggle Alias to Yes → Route traffic to: Alias to Application and Classic Load Balancer → Select ap-south-1 and Lab-App-ALB.

4. Click Create records.

CLI / Configuration examples

```
# Verify DNS resolution once propagated:
nslookup app.example.com
```

Verification

- Open http://app.example.com in a browser and confirm it loads your ALB-backed application.

Expected result: Custom human-readable domain resolves directly to your AWS load balancer endpoint.

> **Security & operational best practice:** Always use Route 53 Alias records (rather than CNAMEs) when pointing zone apex domains (example.com) to AWS resources.

Use Case 49 — Configure HTTPS

Objective: Provision a free SSL/TLS certificate in AWS Certificate Manager (ACM) and terminate HTTPS traffic at an ALB.

Prerequisites: Domain name in Route 53 and an Application Load Balancer.

Console implementation

1. Open AWS Certificate Manager (ACM) in ap-south-1 → Request certificate → Public certificate → Enter app.example.com → DNS validation.
2. Click Create records in Route 53 inside the ACM certificate details to complete DNS validation automatically.
3. Once certificate status is Issued, open EC2 → Load Balancers → Lab-App-ALB → Listeners → Add listener.
4. Select Protocol HTTPS : 443, forward to Lab-Web-TG, select your ACM certificate, and save.
5. Edit the HTTP : 80 listener rule to redirect all HTTP requests to HTTPS : 443 (HTTP 301).

CLI / Configuration examples

```
aws acm list-certificates --region ap-south-1
```

Verification

- Browse to http://app.example.com and confirm it automatically redirects to https://app.example.com with a valid TLS padlock.

Expected result: End-to-end encrypted HTTPS is served at the ALB with automatic ACM certificate renewal.

> **Security & operational best practice:** Offloading TLS at the ALB simplifies certificate rotation and reduces CPU overhead on backend EC2 instances.

Use Case 50 — Configure SSL on EC2

Objective: Install and configure an SSL/TLS certificate directly on an Nginx or Apache web server on EC2.

Prerequisites: Linux EC2 instance with inbound HTTPS (TCP 443) open and a domain name pointing to its public IP.

Console implementation

1. Open TCP port 443 in the EC2 instance's Security Group.

2. SSH into the EC2 instance and install Certbot (for Let's Encrypt) or generate a self-signed certificate for lab testing.
3. Configure Nginx/Apache SSL virtual host on port 443 and reload the web server.

CLI / Configuration examples

```
# Self-signed TLS certificate lab example for Nginx:
sudo mkdir -p /etc/nginx/ssl
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/ssl/ec2.key -out
/etc/nginx/ssl/ec2.crt -subj "/CN=ec2-lab.local"
cat << 'EOF' | sudo tee /etc/nginx/conf.d/ssl.conf
server {
  listen 443 ssl;
  ssl_certificate /etc/nginx/ssl/ec2.crt;
  ssl_certificate_key /etc/nginx/ssl/ec2.key;
  root /usr/share/nginx/html;
}
EOF
sudo nginx -t && sudo systemctl reload nginx
```

Verification

- Run curl -vk https://PUBLIC_IP to inspect the TLS handshake and certificate subject.

Expected result: Web server on EC2 terminates TLS directly on TCP port 443.

> **Security & operational best practice:** Disable deprecated TLS 1.0/1.1 protocols and enforce TLS 1.2 or TLS 1.3 cipher suites.

Use Case 51 — Configure CloudWatch alarms

Objective: Create a metric alarm in Amazon CloudWatch to monitor high CPU utilization on an EC2 instance.

Prerequisites: Running EC2 instance.

Console implementation

1. Open CloudWatch → Alarms → All alarms → Create alarm.
2. Click Select metric → EC2 → Per-Instance Metrics → Select CPUUtilization for Lab-EC2-Linux-01.
3. Set Statistic: Average, Period: 1 minute (or 5 minutes), Threshold type: Static, Whenever CPUUtilization is Greater than 75.
4. Configure notification action (or skip to test state transitions), name it Lab-High-CPU-Alarm, and click Create alarm.

CLI / Configuration examples

```
aws cloudwatch put-metric-alarm --alarm-name "Lab-High-CPU-Alarm" --metric-name CPUUtilization --namespace
AWS/EC2 --statistic Average --period 300 --threshold 75 --comparison-operator GreaterThanThreshold --dimensions
Name=InstanceId,Value=INSTANCE_ID --evaluation-periods 2 --region ap-south-1
```

Verification

- Generate load on EC2 (e.g., yes > /dev/null &) and watch the alarm transition from OK to In alarm.

Expected result: CloudWatch continuously evaluates CPU metrics and enters In alarm state when thresholds are breached.

> **Security & operational best practice:** Use evaluation periods of 2 or 3 datapoints (e.g., 2 out of 2) to avoid false alarms from brief startup CPU spikes.

Use Case 52 — Configure SNS alerts

Objective: Connect a CloudWatch Alarm to an Amazon SNS topic so operations teams receive immediate email/SMS alerts.

Prerequisites: Confirmed Amazon SNS topic subscription and a CloudWatch alarm.

Console implementation

1. Open CloudWatch → Alarms → Select Lab-High-CPU-Alarm → Actions → Edit.

2. On the Configure actions step, under Notification, select In alarm.

3. Choose Select an existing SNS topic → Lab-App-Notifications → Update alarm.

4. Trigger the alarm (or use aws cloudwatch set-alarm-state) to test end-to-end delivery.

CLI / Configuration examples

```
aws cloudwatch set-alarm-state --alarm-name "Lab-High-CPU-Alarm" --state-value ALARM --state-reason "Testing SNS alert delivery" --region ap-south-1
```

Verification

- Confirm an automated CloudWatch Alarm notification email arrives in your inbox within seconds.

Expected result: Metric threshold breaches automatically dispatch operational alerts through Amazon SNS.

> **Security & operational best practice:** Configure both 'In alarm' and 'OK' notification actions so responders know when an incident has self-resolved.

Use Case 53 — Configure Systems Manager

Objective: Connect to and manage EC2 instances securely via AWS Systems Manager Session Manager without opening inbound SSH ports.

Prerequisites: EC2 instance with SSM Agent installed and an IAM role with AmazonSSMManagedInstanceCore attached.

Console implementation

1. Attach the AWS managed policy AmazonSSMManagedInstanceCore to your EC2 instance's IAM Role.

2. Remove inbound SSH (port 22) from the instance's Security Group to verify zero-port access.

3. Open AWS Systems Manager → Node Management → Fleet Manager (or Session Manager → Start session).

4. Select Lab-EC2-Linux-01 and click Start session to open a browser-based interactive shell.

CLI / Configuration examples

```
# Start an SSM terminal session from local CLI (requires Session Manager plugin):
aws ssm start-session --target INSTANCE_ID --region ap-south-1
```

Verification

- Run whoami and hostname inside the Session Manager browser terminal while port 22 remains closed.

Expected result: Full shell access works over outbound HTTPS (port 443) with IAM authentication and CloudTrail session auditing.

> **Security & operational best practice:** Adopt Systems Manager Session Manager as your standard administrative access method to eliminate bastion hosts and exposed SSH ports.

Use Case 54 — Patch management

Objective: Automate OS security patch scanning and installation across EC2 fleets using Systems Manager Patch Manager.

Prerequisites: Managed EC2 instances online in AWS Systems Manager with tag PatchGroup=Linux-Lab.

Console implementation

1. Tag your Linux EC2 instance with Key=PatchGroup, Value=Linux-Lab.

2. Open Systems Manager → Node Management → Patch Manager → Click Patch now.

3. Select Patching operation: Scan and install, Target instances: Specify instance tags (PatchGroup = Linux-Lab).

4. Click Patch now and monitor execution progress in Run Command.

CLI / Configuration examples

```
aws ssm send-command --document-name "AWS-RunPatchBaseline" --targets "Key=tag:PatchGroup,Values=Linux-Lab" --parameters '{"Operation":["Scan"]}' --region ap-south-1
```

Verification

- Check Patch Manager → Dashboard to verify Compliance status reports Compliant with 0 missing critical patches.

Expected result: Systems Manager scans and applies approved OS patches automatically according to the Patch Baseline.

> **Security & operational best practice:** Schedule recurring patching inside Systems Manager Maintenance Windows during off-peak hours.

Use Case 55 — Configure backup automation

Objective: Automate scheduled EBS and EC2 backups with retention policies using AWS Backup.

Prerequisites: Tagged EC2 instance (e.g., Backup=Daily) and AWS Backup default service role.

Console implementation

1. Open AWS Backup → Backup plans → Create backup plan → Start from template (Daily-35day-Retention).

2. Name the plan Lab-EC2-Daily-Backup and click Create plan.

3. Click Assign resources → Name: Lab-EC2-Selection, IAM role: Default role.

4. Under Resource selection, select Include specific resource types → EC2 → Filter by tag Backup = Daily → Assign resources.

CLI / Configuration examples

```
aws backup list-backup-plans --region ap-south-1
```

Verification

- Click Create on-demand backup on your EC2 instance to verify a recovery point is created inside the Backup Vault.

Expected result: Automated, policy-driven backups run on schedule and store immutable recovery points in AWS Backup Vault.

> **Security & operational best practice:** Enable Cross-Region Backup Copy and Vault Lock for ransomware protection and disaster recovery compliance.

Use Case 56 — Configure log management

Objective: Stream operating-system and web server logs from EC2 to Amazon CloudWatch Logs for centralized retention and search.

Prerequisites: EC2 instance with CloudWatchAgentServerPolicy IAM role and CloudWatch Agent installed.

Console implementation

1. Create a CloudWatch Agent JSON configuration specifying log files (e.g., /var/log/messages or /var/log/httpd/access_log) and target log_group_name (/aws/ec2/lab-web-logs).

2. Load the configuration into the CloudWatch Agent and restart it.

3. Open CloudWatch → Logs → Log groups → /aws/ec2/lab-web-logs to inspect live log streams.

CLI / Configuration examples

```
cat << 'EOF' | sudo tee /opt/aws/amazon-cloudwatch-agent/bin/logs-config.json
{
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
```

```
        {"file_path": "/var/log/httpd/access_log", "log_group_name": "/aws/ec2/lab-web-logs", "log_stream_name":
"{instance_id}-access"}
      ]
    }
   }
  }
}
EOF
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c
file:/opt/aws/amazon-cloudwatch-agent/bin/logs-config.json
```

Verification

- Refresh your website URL a few times and verify HTTP request log entries appear in CloudWatch Logs.

Expected result: Application and OS logs persist centrally in CloudWatch Logs even if the EC2 instance is terminated.

> **Security & operational best practice:** Set explicit Log Group retention policies (e.g., 30 or 90 days) instead of 'Never expire' to control log storage costs.

Use Case 57 — S3 lifecycle integration

Objective: Archive rotated EC2 logs or backup exports to Amazon S3 and automate storage tier transitions and expiration.

Prerequisites: S3 bucket storing archived logs/backups and EC2 IAM write access.

Console implementation

1. Upload archived EC2 log files to s3://lab-app-bucket-ACCOUNT_ID/archived-logs/.

2. Open S3 → Select bucket → Management tab → Create lifecycle rule.

3. Name it Archive-And-Expire-Logs, scope to prefix archived-logs/.

4. Add Transition actions: Move to Standard-IA after 30 days, Glacier Flexible Retrieval after 90 days, and Expire objects after 365 days.

CLI / Configuration examples

```
aws s3 sync /var/log/httpd/ s3://lab-app-bucket-ACCOUNT_ID/archived-logs/ --exclude "*" --include "*.gz"
```

Verification

- Confirm the lifecycle rule shows Status: Enabled under the bucket's Management tab.

Expected result: Historical log archives automatically transition to lower-cost storage classes and expire according to retention rules.

> **Security & operational best practice:** Pair cron/logrotate on EC2 with aws s3 sync to offload compressed historical logs nightly.

Use Case 58 — Deploy a Git-based application

Objective: Install Git on an EC2 instance, clone an application repository from GitHub, and deploy updates cleanly.

Prerequisites: Linux EC2 instance with web server or runtime installed and outbound HTTPS access.

Console implementation

1. Connect to your Linux EC2 instance via SSH.

2. Install git (sudo dnf install -y git).

3. Clone your repository into /var/www/html (or an application directory) and start/reload the service.

CLI / Configuration examples

```
sudo dnf install -y git
sudo rm -rf /var/www/html/*
sudo git clone https://github.com/YOUR_USERNAME/YOUR_STATIC_SITE_REPO.git /var/www/html
sudo systemctl restart httpd
```

Verification

- Open http://PUBLIC_IP in a browser to confirm the repository's web files are live; run git pull to test updates.

Expected result: Application code is deployed directly from version control onto the EC2 instance.

> **Security & operational best practice:** Use SSH Deploy Keys (read-only) or AWS CodeDeploy for private repositories instead of storing personal GitHub tokens on servers.

Use Case 59 — Configure Ansible

Objective: Use Ansible to automate idempotent package installation and web server configuration on target EC2 instances.

Prerequisites: Control node (local machine or control EC2 instance) with Ansible installed and SSH key access to target EC2.

Console implementation

1. Install Ansible on your control machine.

2. Create an inventory.ini file containing the target EC2 public/private IP and SSH key path.

3. Write a playbook (setup_web.yml) that installs Nginx, starts the service, and deploys an index.html file.

4. Execute ansible-playbook -i inventory.ini setup_web.yml.

CLI / Configuration examples

```
cat << 'EOF' > inventory.ini
[webservers]
PUBLIC_IP ansible_user=ec2-user ansible_ssh_private_key_file=lab-ec2-keypair.pem
EOF
```

```
cat << 'EOF' > setup_web.yml
- hosts: webservers
  become: yes
  tasks:
    - name: Install Nginx
      package: name=nginx state=present
    - name: Start Nginx
      service: name=nginx state=started enabled=yes
EOF
ansible-playbook -i inventory.ini setup_web.yml
```

Verification

- Re-run the playbook a second time and verify changed=0 (confirming idempotency) and browse to http://PUBLIC_IP.

Expected result: Ansible configures target EC2 servers repeatably without manual SSH commands.

> **Security & operational best practice:** Use the amazon.aws.aws_ec2 dynamic inventory plugin so Ansible automatically discovers tagged instances in Auto Scaling Groups.

Use Case 60 — Configure Terraform

Objective: Provision an EC2 instance and Security Group declaratively using HashiCorp Terraform Infrastructure as Code (IaC).

Prerequisites: Terraform CLI and configured AWS CLI credentials on your workstation.

Console implementation

1. Create a directory terraform-ec2-lab and write a main.tf defining the AWS provider (ap-south-1), an aws_security_group, and an aws_instance.
2. Run terraform init to download the AWS provider plugin.
3. Run terraform plan to preview the infrastructure changes.
4. Run terraform apply -auto-approve to create the EC2 instance, then terraform destroy when finished.

CLI / Configuration examples

```
cat << 'EOF' > main.tf
provider "aws" {
  region = "ap-south-1"
}
resource "aws_instance" "lab_tf_ec2" {
  ami           = "AMI_ID"
  instance_type = "t3.micro"
  tags = { Name = "Lab-Terraform-EC2" }
}
output "instance_public_ip" {
  value = aws_instance.lab_tf_ec2.public_ip
}
EOF
```

```
terraform init && terraform plan && terraform apply -auto-approve
```

Verification

- Confirm Lab-Terraform-EC2 appears in the EC2 Console and matches the outputted public IP.

Expected result: EC2 infrastructure is provisioned reproducibly from version-controlled HCL code and tracked in Terraform state.

> **Security & operational best practice:** Store Terraform state remotely in an encrypted S3 bucket with state locking (use_lockfile = true or DynamoDB) for team collaboration.

Common troubleshooting

Problem	Checks & resolution
ALB Targets show Unhealthy	Verify the web server is running on port 80, returns HTTP 200 on the health check path (/), and the EC2 Security Group allows port 80 from the ALB Security Group.
EC2 cannot connect to RDS	Confirm both resources are in the same VPC and the RDS Security Group allows inbound TCP 3306 from the EC2 Security Group ID.
Private instance has no internet access	Check that the private subnet route table points 0.0.0.0/0 to a NAT instance/gateway in a public subnet and Source/Destination check is disabled on a NAT instance.
SSM Session Manager target missing	Verify AmazonSSMManagedInstanceCore is attached to the EC2 IAM Role, SSM Agent is running, and outbound HTTPS (443) is allowed.
EFS mount hangs or times out	Ensure the EFS mount target exists in the EC2 instance's Availability Zone and allows inbound NFS (TCP 2049) from the EC2 Security Group.

Cleanup guidance

- Run terraform destroy for Terraform-managed labs, and delete Auto Scaling Groups and Application Load Balancers first so dependent instances terminate cleanly.
- Delete Amazon RDS instances (skipping final snapshot in lab environments) and delete Amazon EFS file systems.
- Terminate Bastion Hosts, NAT instances, and standalone EC2 instances, and release any remaining Elastic IPs.
- Delete test SQS queues, SNS topics, CloudWatch alarms, and S3 lab buckets after emptying objects.

AWS documentation

- Elastic Load Balancing (ALB) User Guide: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/

- Amazon EC2 Auto Scaling User Guide: https://docs.aws.amazon.com/autoscaling/ec2/userguide/
- Amazon VPC Peering & VPN Documentation: https://docs.aws.amazon.com/vpc/latest/peering/
- AWS Systems Manager Session & Patch Manager: https://docs.aws.amazon.com/systems-manager/latest/userguide/
- AWS Backup Developer Guide: https://docs.aws.amazon.com/aws-backup/latest/devguide/