

AWS DOCS

Problem 1: EC2 Key Pair Recovery

Using CLI

Part 1: Preparation

- Ensure AWS CLI is installed on Windows

```
aws --version
```

- Configure AWS CLI

```
aws configure
```

- Enter Access Key
- Enter Secret Key
- Region: us-east-1
- Output: json

- Verify no resources exist

```
aws ec2 describe-instances  
aws ec2 describe-key-pairs
```

Part 2: Creating Required Resources

To simulate key pair recovery, we must create a new EC2 instance first (since the account is empty).

- Create a new key pair (this key pair will be deleted to simulate loss)

```
aws ec2 create-key-pair --key-name temp-key --query "KeyMaterial" --output text >  
temp-key.pem
```

- Create security group

```
aws ec2 create-security-group --group-name recovery-sg --description "SG for key  
recovery"
```

- Allow SSH

```
aws ec2 authorize-security-group-ingress --group-name recovery-sg --protocol tcp --  
port 22 --cidr 0.0.0.0/0
```

- Launch instance

```
aws ec2 run-instances --image-id ami-0e001c9271cf7f3b9 --instance-type t2.micro --  
key-name temp-key --security-groups recovery-sg --count 1
```

Now assume: **temp-key.pem is lost** and cannot be used.

Part 3: Configurations (Recovery Technique)

Recovery is done using EC2 Image + Launch New Instance Method:

Step 1 – Create Image From Instance

- Find instance ID

```
aws ec2 describe-instances --query "Reservations[].Instances[].InstanceId"
```

- Create AMI

```
aws ec2 create-image --instance-id <instance-id> --name "recovery-ami"
```

Step 2 – Create New Key Pair

```
aws ec2 create-key-pair --key-name recovered-key --query "KeyMaterial" --output text  
> recovered-key.pem
```

Step 3 – Launch New Instance From AMI

```
aws ec2 run-instances --image-id <ami-id> --instance-type t2.micro --key-name  
recovered-key --security-groups recovery-sg
```

Part 4: Validation

- Verify instance is running

```
aws ec2 describe-instances --query "Reservations[].Instances[].State.Name"
```

- Retrieve public IP

```
aws ec2 describe-instances --query "Reservations[].Instances[].PublicIpAddress"
```

- Connect using new key

```
ssh -i recovered-key.pem ec2-user@<public-ip>
```

Using Console (GUI)

Part 1: Preparation

- Log in to **AWS Console**
- Go to **EC2 Dashboard**
- Ensure no existing instances or key pairs exist

Part 2: Resource Creation

1. Go to **EC2 → Key Pairs → Create key pair**
 - Name: temp-key
 - Format: .pem
 2. Go to **EC2 → Security Groups → Create security group**
 - Name: recovery-sg
 - Allow port 22
 3. Go to **EC2 → Instances → Launch Instances**
 - AMI: Amazon Linux
 - Type: t2.micro
 - Key pair: temp-key
 - SG: recovery-sg
 - Launch
 4. Delete temp-key from your computer (simulate key loss)
-

Part 3: Configuration (Recovery)

Step 1 – Create Image

1. Go to EC2 → Instances
2. Select your instance
3. Click **Actions → Image and templates → Create image**
4. Name: recovery-ami
5. Create

Step 2 – Launch New Key Pair + Instance

1. Go to **EC2 → Key Pairs → Create key pair**
 - Name: recovered-key
 2. Go to **EC2 → AMIs**
 3. Select recovery-ami
 4. Click **Launch instance**
 5. Select recovered-key
 6. Launch
-

Part 4: Validation

- Go to **Instances**
- Select the new instance
- Copy its **Public IP**
- Connect using:

```
ssh -i recovered-key.pem ec2-user@<public-ip>
```

Problem 2: VPC Components Using Console

Part 1: Preparation

- Open AWS Console
 - Search for **VPC**
 - Ensure **no VPCs, no subnets, no IGW, no route tables** exist
(You will always see a “default VPC”, but we ignore it)
-

Part 2: Resource Creation

Step 1 — Create VPC

1. Open **VPC Dashboard**
 2. Click **Create VPC**
 3. Select **VPC Only**
 4. Enter:
 - Name: my-vpc
 - IPv4 CIDR: 10.0.0.0/16
 5. Click **Create VPC**
-

Step 2 — Create Subnets

1. Left menu → **Subnets**
 2. Click **Create subnet**
 3. Select VPC: my-vpc
 4. Create 4 subnets:
 - public-subnet-1 → 10.0.1.0/24 → AZ us-east-1a
 - public-subnet-2 → 10.0.2.0/24 → AZ us-east-1b
 - private-subnet-1 → 10.0.3.0/24 → AZ us-east-1a
 - private-subnet-2 → 10.0.4.0/24 → AZ us-east-1b
 5. Create
-

Step 3 — Create Internet Gateway

1. Left menu → **Internet Gateways**
2. Click **Create internet gateway**
3. Name: my-igw
4. Create

5. Select IGW → Click **Actions** → **Attach to VPC**
 6. Select my-vpc → Attach
-

Step 4 — Create Route Tables

1. Left menu → **Route Tables**
 2. Click **Create route table**
 3. Name: public-rt
 4. VPC: my-vpc
 5. Create
-

Part 3: Configuration

Public Route Table

1. Select public-rt
 2. Go to **Routes** → **Edit routes**
 3. Add:
 - Destination: 0.0.0.0/0
 - Target: my-igw
 4. Save
 5. Go to **Subnet associations** → **Edit**
 6. Select:
 - public-subnet-1
 - public-subnet-2
 7. Save
-

Private Route Table

1. Create new route table: private-rt
 2. Do NOT add IGW route
 3. Go to subnet associations
 4. Select:
 - private-subnet-1
 - private-subnet-2
-

Part 4: Validation

- Go to **Subnets** → verify each subnet is associated correctly
- Go to **Route Tables** → verify public route has IGW

- No errors in console
-

Problem 3: VPC Components Using Commands (CLI)

Using CLI

Part 1: Preparation

- Ensure AWS CLI is configured

```
aws configure
```

- Verify no resources exist:

```
aws ec2 describe-vpcs
aws ec2 describe-subnets
aws ec2 describe-route-tables
aws ec2 describe-internet-gateways
```

Part 2: Creating Required Resources

1. Create VPC

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16
```

Save the VPC ID returned.

2. Create Subnets

```
aws ec2 create-subnet --vpc-id <vpc-id> --cidr-block 10.0.1.0/24 --availability-zone
us-east-1a
aws ec2 create-subnet --vpc-id <vpc-id> --cidr-block 10.0.2.0/24 --availability-zone
us-east-1b
aws ec2 create-subnet --vpc-id <vpc-id> --cidr-block 10.0.3.0/24 --availability-zone
us-east-1a
aws ec2 create-subnet --vpc-id <vpc-id> --cidr-block 10.0.4.0/24 --availability-zone
us-east-1b
```

3. Create Internet Gateway

```
aws ec2 create-internet-gateway
```

Then attach it:

```
aws ec2 attach-internet-gateway --vpc-id <vpc-id> --internet-gateway-id <igw-id>
```

4. Create Route Tables

```
aws ec2 create-route-table --vpc-id <vpc-id>
```

Save Route Table ID → public route table.

Part 3: Configurations

Public Route Table

```
aws ec2 create-route --route-table-id <public-rt-id> --destination-cidr-block  
0.0.0.0/0 --gateway-id <igw-id>
```

Associate Subnets

```
aws ec2 associate-route-table --subnet-id <subnet-id1> --route-table-id <public-rt-id>  
aws ec2 associate-route-table --subnet-id <subnet-id2> --route-table-id <public-rt-id>
```

Part 4: Validation

Check routes

```
aws ec2 describe-route-tables
```

Check subnets

```
aws ec2 describe-subnets
```

Problem 4: All Types of IAM Operations Using Console

Using Console

Part 1: Preparation

- Open **AWS Console**
- In the search bar → type **IAM** → open **IAM Dashboard**
- Ensure the account is clean:
 - Go to **Users** → Confirm no custom users exist
 - Go to **User Groups** → Confirm no groups exist
 - Go to **Roles** → Only AWS default roles exist
 - Go to **Policies** → AWS-managed policies only

Part 2: Resource Creation

Step 1 — Create a User

1. In IAM Dashboard → Left menu → **Users**
2. Click **Create user**
3. Enter username:
 - example-user
4. Click **Next**
5. Select **Attach policies directly**
6. Search for and select:
 - AmazonEC2FullAccess
 - AmazonS3FullAccess
7. Click **Next**
8. Click **Create user**

Step 2 — Create a User Group

1. IAM Dashboard → Left menu → **User groups**
2. Click **Create group**
3. Enter group name:
 - example-group
4. Search policies:
 - AmazonEC2ReadOnlyAccess
 - AmazonS3ReadOnlyAccess
5. Select both
6. Create group

Step 3 — Create a Role

1. IAM Dashboard → Left panel → **Roles**
 2. Click **Create role**
 3. Choose **AWS service**
 4. Select **EC2** (most common use-case)
 5. Click **Next**
 6. Attach policy:
 - AmazonS3ReadOnlyAccess
 7. Click **Next**
 8. Name the role:
 - ec2-s3-role
 9. Click **Create role**
-

Step 4 — Create Policy (Custom Policy)

1. IAM → Left menu → **Policies**
2. Click **Create policy**
3. Choose **JSON**
4. Paste the following:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "ec2:Describe*",  
      "Resource": "*"  
    }  
  ]  
}
```

5. Click **Next**
 6. Enter name:
 - describe-ec2-only
 7. Click **Create policy**
-

Part 3: Configuration

1. Add User to Group

1. Go to **Users**
2. Select **example-user**
3. Go to **Groups** tab
4. Click **Add user to groups**
5. Select **example-group**

6. Add

2. Add MFA (Optional but Required in Tasks)

1. Select user **example-user**
 2. Click **Security credentials**
 3. Scroll to **MFA**
 4. Click **Assign MFA device**
 5. Choose **Authenticator App**
 6. Scan QR Code in Google Authenticator
 7. Enter two consecutive OTPs
 8. Click **Assign**
-

3. Add Inline Policy to User

1. Users → example-user
2. Permissions tab
3. Scroll to **Inline policies**
4. Click **Add inline policy**
5. Go to **JSON** and paste:

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": "ec2:StartInstances",  
        "Resource": "*"  
    }]  
}
```

6. Click **Review policy**
 7. Name it: allow-start-instances
 8. Click **Create policy**
-

Part 4: Validation

- Go to IAM → Users → example-user
- Check:
 - Managed policies → Should see 2 policies
 - Group membership → example-group
 - Inline policy → allow-start-instances
- Go to IAM → Roles → ec2-s3-role
 - Verify S3 read-only policy is attached
- Go to IAM → Groups → example-group

- Verify 2 attached policies
-

Problem 5: IAM Operations Using Commands (CLI)

Using CLI

Part 1: Preparation

- Verify AWS CLI setup:

```
aws configure
```

- Check existing IAM resources:

```
aws iam list-users
aws iam list-groups
aws iam list-roles
aws iam list-policies
```

Everything should show no custom resources.

Part 2: Creating Required Resources

Step 1 — Create a User

```
aws iam create-user --user-name example-user
```

Step 2 — Create User Group

```
aws iam create-group --group-name example-group
```

Step 3 — Attach Policies to Group

```
aws iam attach-group-policy --group-name example-group --policy-arn
arn:aws:iam::aws:policy/AmazonEC2ReadOnlyAccess
```

```
aws iam attach-group-policy --group-name example-group --policy-arn
arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Step 4 — Add User to Group

```
aws iam add-user-to-group --user-name example-user --group-name example-group
```

Step 5 — Attach Managed Policies to User

```
aws iam attach-user-policy --user-name example-user --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess
```

```
aws iam attach-user-policy --user-name example-user --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess
```

Step 6 — Create Custom Policy

Save JSON in Windows as **policy.json**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
      "Resource": "*"
    }
  ]
}
```

Run:

```
aws iam create-policy --policy-name describe-ec2-only --policy-document file://policy.json
```

Step 7 — Create Role

Create assume-role document **trust.json**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com" },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Run:

```
aws iam create-role --role-name ec2-s3-role --assume-role-policy-document file://trust.json
```

Attach policy:

```
aws iam attach-role-policy --role-name ec2-s3-role --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Part 3: Configuration

Inline policy

Create `inline.json`:

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": "ec2:StartInstances",  
        "Resource": "*"  
    }]  
}
```

Attach:

```
aws iam put-user-policy --user-name example-user --policy-name allow-start-instances --policy-document file://inline.json
```

Part 4: Validation

```
aws iam list-users  
aws iam list-groups  
aws iam list-attached-group-policies --group-name example-group  
aws iam list-attached-user-policies --user-name example-user  
aws iam list-user-policies --user-name example-user  
aws iam list-roles
```

Everything should match expected output.

Problem 6: S3 Multipart Upload

Using CLI

Part 1: Preparation

- Ensure AWS CLI is configured
- Create a large test file (Windows):

```
fsutil file createnew bigfile.bin 104857600
```

- Check no bucket exists:

```
aws s3 ls
```

Part 2: Creating Required Resources

Create S3 Bucket

```
aws s3 mb s3://multipart-demo-bucket-112233
```

Part 3: Configuration (Multipart Upload)

Simplest Method — Automatic Multipart Upload

```
aws s3 cp bigfile.bin s3://multipart-demo-bucket-112233/
```

AWS automatically splits the file into multiple parts (>5MB triggers multipart).

Part 4: Validation

- List bucket contents:

```
aws s3 ls s3://multipart-demo-bucket-112233/
```

- Console validation:

- Go to S3 Console
- Open the bucket
- Confirm file is uploaded

Using Console

Part 1: Preparation

- Go to **S3 Console**
 - Check no buckets exist
 - Ensure you are in the correct region
-

Part 2: Resource Creation

1. Click **Create bucket**
 2. Enter name:
 - multipart-demo-bucket-112233
 3. Region: us-east-1
 4. Keep all defaults
 5. Click **Create bucket**
-

Part 3: Configuration

1. Open the bucket
 2. Click **Upload**
 3. Add file bigfile.bin
 4. Click **Upload**
 5. AWS automatically performs a multipart upload
-

Part 4: Validation

- Check file listed successfully
 - Properties → Look for:
 - “Multipart Upload” metadata
-

Problem 7: Application Load Balancer Using Console

Using Console

Part 1: Preparation

1. Log in to the **AWS Management Console**.
2. In the search bar → type **EC2** → open the **EC2 Dashboard**.

3. Verify the account has:
 - No EC2 instances
 - No security groups (except default)
 - No load balancers
 - No target groups
 4. Ensure default VPC exists (AWS always provides one).
-

Part 2: Resource Creation

Step 1 — Create Security Group

1. EC2 Dashboard → Left menu → **Security Groups**
 2. Click **Create security group**
 3. Enter:
 - Name: alb-sg
 - Description: security group for ALB
 - VPC: default
 4. In **Inbound rules**:
 - Add rule → HTTP → Port 80 → Source 0.0.0.0/0
 5. Create security group
-

Step 2 — Launch Two EC2 Instances

1. EC2 Dashboard → **Instances** → **Launch instances**
2. Configure:
 - Name: web-server-1
 - AMI: Amazon Linux 2023 / Amazon Linux 2
 - Instance type: t2.micro
 - Key pair: create or select
 - Network: default VPC
 - Subnet: us-east-1a (for instance 1)
 - Security group: alb-sg
3. Select **Advanced details** → **User data**, paste:

```
#!/bin/bash
yum install httpd -y
systemctl start httpd
systemctl enable httpd
echo "<h1>Server 1</h1>" > /var/www/html/index.html
```

4. Launch instance

Repeat for second instance:

- Name: web-server-2

- Subnet: us-east-1b
 - User data: same but change header to “Server 2”.
-

Part 3: Configuration

Step 3 — Create Target Group

1. EC2 → Load Balancing → **Target Groups**
 2. Click **Create target group**
 3. Choose:
 - Target type: Instance
 - Name: web-tg
 - Protocol: HTTP
 - Port: 80
 - VPC: default
 4. Click **Next**
 5. Select both web-server-1 and web-server-2
 6. Click **Include as pending**
 7. Click **Create target group**
-

Step 4 — Create Application Load Balancer

1. EC2 → Load Balancers → **Create Load Balancer**
 2. Choose **Application Load Balancer**
 3. Enter:
 - Name: web-alb
 - Scheme: Internet-facing
 - IP type: IPv4
 4. Network mapping:
 - Select two subnets (us-east-1a, us-east-1b)
 5. Security group:
 - Select **alb-sg**
 6. Listeners:
 - HTTP 80 → Default action → Forward to **web-tg**
 7. Click **Create load balancer**
-

Part 4: Validation

1. Go to **Load Balancers**
2. Select **web-alb**
3. Copy its DNS name

4. Paste in browser
 5. Refresh repeatedly:
 - You should see alternating:
 - “Server 1”
 - “Server 2”
 6. Go to **Target Groups** → **web-tg** → **Targets**
 - Verify both instances show **healthy**
-

Problem 8: Auto Scaling Groups Using Console

Using Console

Part 1: Preparation

1. Log in to AWS Console.
 2. Search for **EC2**.
 3. Ensure:
 - No launch templates exist.
 - No Auto Scaling groups exist.
 - Default VPC exists.
-

Part 2: Resource Creation

Step 1 — Create Launch Template

1. EC2 → **Launch Templates**
2. Click **Create launch template**
3. Enter:
 - Name: web-template
4. Template details:
 - AMI: Amazon Linux 2023
 - Type: t2.micro
 - Key pair: choose or create
5. Choose **Security group**:
 - Create one: web-sg
 - Allow HTTP 80 from 0.0.0.0/0
6. Scroll to **Advanced details** → **User data**
7. Enter:

```
#!/bin/bash
yum install httpd -y
systemctl start httpd
systemctl enable httpd
echo "<h1>Instance from ASG</h1>" > /var/www/html/index.html
```

8. Click **Create launch template**
-

Step 2 — Create Auto Scaling Group

1. EC2 → Auto Scaling → **Auto Scaling groups**

2. Click **Create Auto Scaling group**

3. Choose:

- Name: web-asg
- Launch template: web-template

4. Click **Next**

5. Select VPC: default

6. Select 2 subnets (us-east-1a, us-east-1b)

7. Desired capacity: 2

8. Min: 1

9. Max: 3

10. Click **Next**

11. Keep defaults until the end

12. Click **Create Auto Scaling group**
-

Part 3: Configuration

Attach Load Balancer (Optional but recommended)

1. ASG → Select web-asg

2. Click **Edit** under **Load balancing**

3. Choose:

- Application Load Balancer
- Select an existing target group or create new

4. Save
-

Set Scaling Policies

1. In ASG → Choose **Automatic scaling**

2. Add a policy:

- CPU > 70% → Add instance
- CPU < 40% → Remove instance

Part 4: Validation

1. EC2 → Instances → verify **2 instances** running
 2. Terminate 1 manually
 - ASG should launch a new one automatically
 3. Simulate high CPU load (optional)
 4. Verify scaling events in:
 - ASG → Activity history
-

Problem 9: S3 Operations Using Console and Commands

Using CLI

Part 1: Preparation

- Verify CLI configuration:

```
aws configure
```

- Check S3 buckets:

```
aws s3 ls
```

Part 2: Creating Required Resources

Create S3 Bucket

```
aws s3 mb s3://my-cli-bucket-778899
```

Upload Files

```
aws s3 cp file.txt s3://my-cli-bucket-778899/
```

Create Folder

S3 doesn't have real folders. Just prefix object:

```
aws s3 cp file.txt s3://my-cli-bucket-778899/folder1/
```

Part 3: Configuration

Enable Versioning

```
aws s3api put-bucket-versioning --bucket my-cli-bucket-778899 --versioning-configuration Status=Enabled
```

Enable Encryption

```
aws s3api put-bucket-encryption --bucket my-cli-bucket-778899 --server-side-encryption-configuration "{
  \"Rules\": [
    {
      \"ApplyServerSideEncryptionByDefault\": {
        \"SSEAlgorithm\": \"AES256\"
      }
    }
  ]
}"
```

Part 4: Validation

```
aws s3 ls s3://my-cli-bucket-778899 --recursive
aws s3api get-bucket-versioning --bucket my-cli-bucket-778899
aws s3api get-bucket-encryption --bucket my-cli-bucket-778899
```

Using Console

Part 1: Preparation

1. Open AWS Console
2. Search **S3**
3. Ensure no buckets exist

Part 2: Resource Creation

Create Bucket

1. Click **Create bucket**
 2. Enter:
 - Name: my-console-bucket-778899
 - Region: us-east-1
 3. Leave defaults
 4. Click **Create bucket**
-

Upload Files

1. Click bucket name
 2. Click **Upload**
 3. Add file(s)
 4. Upload
-

Create Folder

1. Inside bucket → Click **Create folder**
 2. Name: documents
 3. Click **Create**
-

Part 3: Configuration

Enable Versioning

1. Bucket → **Properties**
 2. Scroll to **Versioning**
 3. Click **Enable**
-

Enable Encryption

1. Bucket → Properties
 2. Scroll to **Default encryption**
 3. Choose:
 - SSE-S3
 4. Save
-

Part 4: Validation

- Check versioning status
 - Check object metadata
 - Re-upload file and verify version count increases
-

Problem 10: Application Load Balancer Using Commands (CLI)

Using CLI

Part 1: Preparation

- Ensure AWS CLI is configured
- Verify:

```
aws ec2 describe-vpcs  
aws ec2 describe-subnets
```

- Identify:
 - default VPC ID
 - two subnet IDs
-

Part 2: Creating Required Resources

Step 1 — Create Security Group

```
aws ec2 create-security-group --group-name alb-sg-cli --description "ALB SG" --vpc-id <vpc-id>
```

Allow HTTP:

```
aws ec2 authorize-security-group-ingress --group-name alb-sg-cli --protocol tcp --port 80 --cidr 0.0.0.0/0
```

Step 2 — Create Target Group

```
aws elbv2 create-target-group --name cli-tg --protocol HTTP --port 80 --vpc-id <vpc-id> --target-type instance
```

Take note of:

- Target Group ARN
-

Step 3 — Launch EC2 Instances

```
aws ec2 run-instances --image-id <ami-id> --instance-type t2.micro --count 2 --  
security-group-ids <sg-id> --subnet-id <subnet-id>
```

Register instances:

```
aws elbv2 register-targets --target-group-arn <tg-arn> --targets Id=i-xxx Id=i-yyy
```

Part 3: Configuration

Create ALB

```
aws elbv2 create-load-balancer --name cli-alb --subnets <subnet-1> <subnet-2> --  
security-groups <sg-id>
```

Copy ALB ARN.

Create Listener

```
aws elbv2 create-listener --load-balancer-arn <alb-arn> --protocol HTTP --port 80 --  
default-actions Type=forward,TargetGroupArn=<tg-arn>
```

Part 4: Validation

```
aws elbv2 describe-load-balancers  
aws elbv2 describe-target-health --target-group-arn <tg-arn>
```

Get DNS:

```
aws elbv2 describe-load-balancers --query "LoadBalancers[*].DNSName"
```

Test in browser.

Problem 11: Auto Scaling Groups Using Commands (CLI)

Using CLI

Part 1: Preparation

- Ensure AWS CLI is configured
- Identify:

```
aws ec2 describe-vpcs  
aws ec2 describe-subnets
```

Part 2: Creating Required Resources

Step 1 — Create Launch Template

Create file `template.json`:

```
{  
  "ImageId": "<ami-id>",  
  "InstanceType": "t2.micro",  
  "SecurityGroupIds": ["<sg-id>"],  
  "UserData":  
    "IyEvYmluL2Jhc2gKeXVtIGluc3RhbGwgaHR0cGQgLXkKc3lzdGVtY3RsIHN0YXJ0IGH0dHBkCnN5c3RlbWN0  
    bCBlbmFibGUgaHR0cGQKZWNo byAiPEgxPkFXUyBBU0cgSw5zdGFuY2U8L0gxPiIgPiAvdmFyL3d3dy9odG1sL  
    2luZGV4Lmh0bWw="  
}
```

Run:

```
aws ec2 create-launch-template --launch-template-name cli-template --version-  
description v1 --launch-template-data file://template.json
```

Step 2 — Create Auto Scaling Group

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name cli-asg --launch-  
template LaunchTemplateName=cli-template,Version=1 --min-size 1 --max-size 3 --  
desired-capacity 2 --vpc-zone-identifier "<subnet1>,<subnet2>"
```

Part 3: Configuration

Add Scaling Policies

Scale Out:

```
aws autoscaling put-scaling-policy --auto-scaling-group-name cli-asg --policy-name scaleout --policy-type SimpleScaling --scaling-adjustment 1 --adjustment-type ChangeInCapacity
```

Scale In:

```
aws autoscaling put-scaling-policy --auto-scaling-group-name cli-asg --policy-name scalein --policy-type SimpleScaling --scaling-adjustment -1 --adjustment-type ChangeInCapacity
```

Part 4: Validation

```
aws autoscaling describe-auto-scaling-groups  
aws ec2 describe-instances  
aws autoscaling describe-scaling-activities
```

Problem 12: Auto Scaling Groups Using Commands (CLI)

(Already covered in Problem 11—ASG via CLI)