



Name: Lujain Zia

Roll no: 2023-BSE-034

Date: December 2, 2025

Lab 9

Task 1 — GitHub CLI, Codespace setup and authentication

1. (Local desktop) Install GitHub CLI (Windows example via winget):

```
winget install --id GitHub.cli
```

Save screenshot as: task1_gh_install.png — terminal showing the winget install command output.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user> winget install --id GitHub.cli
The 'msstore' source requires that you view the following agreements before using.
Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction
The source requires the current machine's 2-letter geographic region to be sent to the backend service to function properly (ex. "US").

Do you agree to all the source agreements terms?
[Y] Yes [N] No: y
Found GitHub CLI [GitHub.cli] Version 2.83.1
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Downloading https://github.com/cli/cli/releases/download/v2.83.1/gh_2.83.1_windows_amd64.msi
██████████████████ 17.6 MB / 17.6 MB
Successfully verified installer hash
Starting package install...
Successfully installed
PS C:\Users\user>
PS C:\Users\user>
```

2. (Local) Authenticate GH CLI for Codespaces:

```
gh auth login -s codespace
```

When prompted, generate a GitHub Access Token (classic) in the browser with the following Token scopes:

```
admin:org
codespace
repo
```

Copy the token into the GH CLI prompt to complete login.

Save screenshot as: task1_gh_auth_login.png — screenshot of gh auth login confirmation (redact token).

```
PS C:\Users\user> gh auth login -s codespace
? Where do you use GitHub? GitHub.com
? What is your preferred protocol for Git operations on this host? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Paste an authentication token
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'workflow'.
? Paste your authentication token: - gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as lujainzia127
PS C:\Users\user>
```

3. (Local) List available Codespaces (optional verification):

```
gh codespace list
```

Save screenshot as: task1_codespace_list.png — gh codespace list output (show codespace name).

4. (Local) Create or connect to a Codespace. To create a new codespace from the current repo:

```
gh codespace create --repo <owner>/<repo> --branch main --machine basicLinux32gb
```

Or to open an existing codespace:

```
gh codespace ssh -c <name_of_codespace>
```

After connecting via gh codespace ssh, you will be inside a Linux shell that the rest of this lab assumes.

Save screenshot as: task1_codespace_ssh_connected.png — terminal inside the Codespace shell after gh codespace ssh -c <name>.

```
PS C:\Users\user> gh codespace ssh -c zany disco
getting full codespace details: HTTP 404: Not Found (https://api.github.com/user/codespaces/zany?internarue)
PS C:\Users\user> gh codespace ssh -c zany-disco-jjp576x7r5j93p66
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-1030-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

Task 2 — Install AWS CLI inside the Codespace and configure it

1. Download and install AWS CLI:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

```
unzip awscliv2.zip
```

```
sudo ./aws/install
```

Save screenshot as: task2_aws_install_and_version.png — terminal showing the download/unzip/install output.

```
Windows PowerShell + ^

creating: aws/dist/awscli/customizations/wizard/wizards/events/
creating: aws/dist/awscli/customizations/wizard/wizards/iam/
creating: aws/dist/awscli/customizations/wizard/wizards/lambda/
inflating: aws/dist/awscli/customizations/wizard/wizards/dynamodb/new-table.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/configure/_main.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/iam/new-role.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/events/new-rule.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/lambda/new-function.yml
inflating: aws/dist/awscli/customizations/sso/index.html
inflating: aws/dist/awscli/data/cli.json
inflating: aws/dist/awscli/data/metadata.json
inflating: aws/dist/awscli/data/ac.index
    creating: aws/dist/prompt_toolkit-3.0.51.dist-info/licenses/
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/METADATA
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/WHEEL
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/RECORD
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/INSTALLER
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/top_level.txt
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/licenses/AUTHORS.rst
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/licenses/LICENSE
inflating: aws/dist/wheel-0.45.1.dist-info/LICENSE.txt
inflating: aws/dist/wheel-0.45.1.dist-info/entry_points.txt
inflating: aws/dist/wheel-0.45.1.dist-info/RECORD
inflating: aws/dist/wheel-0.45.1.dist-info/WHEEL
inflating: aws/dist/wheel-0.45.1.dist-info/INSTALLER
inflating: aws/dist/wheel-0.45.1.dist-info/direct_url.json
inflating: aws/dist/wheel-0.45.1.dist-info/REQUESTED
inflating: aws/dist/wheel-0.45.1.dist-info/METADATA
You can now run: /usr/local/bin/aws --version
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

2. Verify installation:

aws --version

Save screenshot as: task2_aws_install_and_version.png — include aws --version output (same file as install output is acceptable).

```
Windows PowerShell + ^

creating: aws/dist/awscli/customizations/wizard/wizards/lambda/
inflating: aws/dist/awscli/customizations/wizard/wizards/dynamodb/new-table.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/configure/_main.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/iam/new-role.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/events/new-rule.yml
inflating: aws/dist/awscli/customizations/wizard/wizards/lambda/new-function.yml
inflating: aws/dist/awscli/customizations/sso/index.html
inflating: aws/dist/awscli/data/cli.json
inflating: aws/dist/awscli/data/metadata.json
inflating: aws/dist/awscli/data/ac.index
    creating: aws/dist/prompt_toolkit-3.0.51.dist-info/licenses/
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/METADATA
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/WHEEL
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/RECORD
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/INSTALLER
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/top_level.txt
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/licenses/AUTHORS.rst
inflating: aws/dist/prompt_toolkit-3.0.51.dist-info/licenses/LICENSE
inflating: aws/dist/wheel-0.45.1.dist-info/LICENSE.txt
inflating: aws/dist/wheel-0.45.1.dist-info/entry_points.txt
inflating: aws/dist/wheel-0.45.1.dist-info/RECORD
inflating: aws/dist/wheel-0.45.1.dist-info/WHEEL
inflating: aws/dist/wheel-0.45.1.dist-info/INSTALLER
inflating: aws/dist/wheel-0.45.1.dist-info/direct_url.json
inflating: aws/dist/wheel-0.45.1.dist-info/REQUESTED
inflating: aws/dist/wheel-0.45.1.dist-info/METADATA
You can now run: /usr/local/bin/aws --version
@lujainzia127 ~ /workspaces/lab9 (main) $ aws --version
aws-cli/2.32.11 Python/3.13.9 Linux/6.8.0-1030-azure exe/x86_64.ubuntu.24
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

3. Configure the AWS CLI (you will provide Access Key ID and Secret Access Key for a user with permissions, or use root/IAM user you prepared for the lab):

```
aws configure
```

Enter:

AWS Access Key ID
AWS Secret Access Key
Default region name (e.g., me-central-1)
Default output format (e.g., json)

Save screenshot as: task2_aws_configure_and_files.png — show aws configure prompt (redact secret values if visible).

```
@lujainzia127 → /workspaces/lab9 (main) $ aws configure
AWS Access Key ID [None]: AKIAQXUBVQUY4EAHQPU6
AWS Secret Access Key [None]: ██████████
Default region name [None]: me-central-1
Default output format [None]: json
@lujainzia127 → /workspaces/lab9 (main) $ |
```

4. Verify credentials/config files:

```
cat ~/.aws/credentials
cat ~/.aws/config
```

Save screenshot as: task2_aws_configure_and_files.png — output of cat commands (redact secrets).

```
@lujainzia127 → /workspaces/lab9 (main) $ cat ~/.aws/credentials
[default]
aws_access_key_id = AKIAQXUBVQUY4EAHQPU6
aws_secret_access_key = ██████████
@lujainzia127 → /workspaces/lab9 (main) $ cat ~/.aws/config
[default]
region = me-central-1
output = json
@lujainzia127 → /workspaces/lab9 (main) $ |
```

5. Verify connectivity (you should see a JSON result showing your caller identity):

```
aws sts get-caller-identity
```

Expected output (example):

```
{
    "UserId": "EXAMPLEUSERID1234567890",
    "Account": "123456789012",
    "Arn": "arn:aws:iam::123456789012:user/ExampleUser"
}
```

Save screenshot as: task2_aws_get_caller_identity.png — aws sts get-caller-identity output.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws sts get-caller-identity
{
    "UserId": "AIDAQXUBVQUY4D5JLHRBT",
    "Account": "050737808689",
    "Arn": "arn:aws:iam::050737808689:user/lab9"
}
@lujainzia127 → /workspaces/lab9 (main) $ |
```

Task 3 — Create security group and add ingress rules using Codespace IP

1. Create a security group (substitute your VPC id):

```
aws ec2 create-security-group --group-name 'MySecurityGroup' \
--description 'My Security Group' \
--vpc-id 'vpc-EXAMPLE1234567890'
```

This command returns a security group id, for example sg-EXAMPLE1234567890.

Save screenshot

as: task3_create_security_group_output.png — output of create-security-group.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 create-security-group --group-name 'MySecurityGroup' --description 'My Security Group' --vpc-id vpc-01b7d554ea60c902d
{
    "GroupId": "sg-0f5938330b9395b53",
    "SecurityGroupArn": "arn:aws:ec2:me-central-1:050737808689:security-group/sg-0f5938330b9395b53"
}
@lujainzia127 → /workspaces/lab9 (main) $ |
```

2. Inspect the security group (initially ingress will be empty or default):

```
aws ec2 describe-security-groups --group-ids sg-EXAMPLE1234567890
```

Save screenshot as: task3_describe_sg_before_ingress.png — describe-security-groups before adding rules.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-security-groups --group-ids sg-0f5938330b9395b53
{
    "SecurityGroups": [
        {
            "GroupId": "sg-0f5938330b9395b53",
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "UserIdGroupPairs": [],
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": []
                }
            ],
            "VpcId": "vpc-01b7d554ea60c902d",
            "SecurityGroupArn": "arn:aws:ec2:me-central-1:050737808689:security-group/sg-0f5938330b9395b53",
            "OwnerId": "050737808689",
            "GroupName": "MySecurityGroup",
            "Description": "My Security Group",
            "IpPermissions": []
        }
    ]
}
@lujainzia127 → /workspaces/lab9 (main) $ |
```

3. Get your Codespace public IP (from inside the Codespace):

```
curl icanhazip.com
```

Save screenshot as: task3_codespace_public_ip.png — output of curl icanhazip.com.

```
'@lujainzia127 ~ /workspaces/lab9 (main) $ curl ianhazip.com
4.240.18.229
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

4. Authorize SSH inbound on port 22 from your Codespace IP:

```
aws ec2 authorize-security-group-ingress \
--group-id sg-EXAMPLE1234567890 \
--protocol tcp \
--port 22 \
--cidr <XXX.XXX.XXX.XXX>/32
```

Save screenshot as: task3_authorize_ssh_and_describe.png — authorize-security-group-ingress for SSH and subsequent describe output.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 authorize-security-group-ingress \
--group-id sg-0f5938330b9395b53 \
--protocol tcp \
--port 22 \
--cidr 4.240.18.229/32
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-069569c971d5023b0",
            "GroupId": "sg-0f5938330b9395b53",
            "GroupOwnerId": "650737808689",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIpv4": "4.240.18.229/32",
            "SecurityGroupRuleArn": "arn:aws:ec2:me-central-1:050737808689:security-group-rule/sgr-069569c971d5023b0"
        }
    ]
}
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 describe-security-groups --group-ids sg-0f5938330b9395b53
{
    "SecurityGroups": [
        {
            "GroupId": "sg-0f5938330b9395b53",
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "UserIdGroupPairs": [],
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": []
                }
            ],
            "VpcId": "vpc-01b7d554ea60c902d",
            "SecurityGroupArn": "arn:aws:ec2:me-central-1:050737808689:security-group/sg-0f5938330b9395b53",
            "OwnerId": "650737808689"
        }
    ]
}
```

5. Verify ingress rule appears:

```
aws ec2 describe-security-groups --group-ids sg-EXAMPLE1234567890
Save screenshot as: task3_authorize_ssh_and_describe.png — describe output showing SSH ingress (same file as previous step is acceptable).
```

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 authorize-security-group-ingress \
--group-id sg-0f5938330b9395b53 \
--protocol tcp \
--port 22 \
--cidr 4.240.18.229/32
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-069569c971d5023b0",
            "GroupId": "sg-0f5938330b9395b53",
            "GroupOwnerId": "050737808689",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIpv4": "4.240.18.229/32",
            "SecurityGroupRuleArn": "arn:aws:ec2:me-central-1:050737808689:security-group-rule/sgr-069569c971d5023b0"
        }
    ]
}
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-security-groups --group-ids sg-0f5938330b9395b53
{
    "SecurityGroups": [
        {
            "GroupId": "sg-0f5938330b9395b53",
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "UserIdGroupPairs": [],
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": []
                }
            ],
            "VpcId": "vpc-01b7d554ea60c902d",
            "SecurityGroupArn": "arn:aws:ec2:me-central-1:050737808689:security-group/sg-0f5938330b9395b53",
            "GroupName": "sg-0f5938330b9395b53"
        }
    ]
}
```

6. Add an HTTP rule (port 80) using ip-permissions JSON:

```
aws ec2 authorize-security-group-ingress \
--group-id 'sg-EXAMPLE1234567890' \
--ip-permissions
'{"FromPort":80,"ToPort":80,"IpProtocol":"tcp","IpRanges":[{"CidrIp":<XXX.XX.X.XXX.XXX>/32"}]}
```

Save screenshot as: task3_authorize_http_and_describe.png — HTTP ip-permissions command and response.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 authorize-security-group-ingress \
--group-id sg-0f5938330b9395b53 \
--ip-permissions '[
    {
        "FromPort": 80,
        "ToPort": 80,
        "IpProtocol": "tcp",
        "IpRanges": [
            {
                "CidrIp": "4.240.18.229/32"
            }
        ]
    }
]
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-07dc4c9f4297a44ec",
            "GroupId": "sg-0f5938330b9395b53",
            "GroupOwnerId": "050737808689",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIpv4": "4.240.18.229/32",
            "SecurityGroupRuleArn": "arn:aws:ec2:me-central-1:050737808689:security-group-rule/sgr-07dc4c9f4297a44ec"
        }
    ]
}
```

7. Verify both ingress rules are present:

```
aws ec2 describe-security-groups --group-ids sg-EXAMPLE1234567890
```

Save screenshot as: task3_describe_sg_final.png — final describe showing both ingress rules.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-security-groups --group-ids sg-0f5938330b9395b53
{
    "SecurityGroups": [
        {
            "GroupId": "sg-0f5938330b9395b53",
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "UserIdGroupPairs": [],
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": []
                }
            ],
            "VpcId": "vpc-01b7d554ea60c902d",
            "SecurityGroupArn": "arn:aws:ec2:me-central-1:050737808689:security-group/sg-0f5938330b9395b53",
            "OwnerId": "050737808689",
            "GroupName": "MySecurityGroup",
            "Description": "My Security Group",
            "IpPermissions": [
                {
                    "IpProtocol": "tcp",
                    "FromPort": 80,
                    "ToPort": 80,
                    "UserIdGroupPairs": [],
                    "IpRanges": [

```

Task 4 — Create a key pair, describe key pairs, and launch EC2 instance

1. Create the key pair and save the PEM file into the Codespace workspace:

```
aws ec2 create-key-pair \
--key-name MyED25519Key \
--key-type ed25519 \
--key-format pem \
--query 'KeyMaterial' \
--output text > MyED25519Key.pem
```

Save screenshot as: task4_create_keypair_output.png — output of create-key-pair and ls -l MyED25519Key.pem.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 create-key-pair \
--key-name MyED25519Key \
--key-type ed25519 \
--key-format pem \
--query 'KeyMaterial' \
--output text > MyED25519Key.pem
@lujainzia127 → /workspaces/lab9 (main) $ ls -l MyED25519Key.pem
-rw-rw-rw- 1 codespace codespace 388 Dec  7 16:52 MyED25519Key.pem
@lujainzia127 → /workspaces/lab9 (main) $ |
```

2. View created key pairs:

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

Save screenshot as: task4_describe_keypairs.png — describe-key-pairs output.

```

@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-key-pairs
{
    "KeyPairs": [
        {
            "KeyPairId": "key-062911b808fe9dd58",
            "KeyType": "ed25519",
            "Tags": [],
            "CreateTime": "2025-12-07T16:52:30.991000+00:00",
            "KeyName": "MyED25519Key",
            "KeyFingerprint": "OXYrsS/IE1Aw8Wm0fH74YY8fn8KnAUJEs2hWN6T/Gqc="
        },
        {
            "KeyPairId": "key-07bcf5909bdb9419b",
            "KeyType": "rsa",
            "Tags": [],
            "CreateTime": "2025-11-21T16:19:05.992000+00:00",
            "KeyName": "lab8key",
            "KeyFingerprint": "5f:bc:9b:8a:6b:62:d5:fc:c9:3c:6e:11:b4:8e:80:fb:64:b8:14:a3"
        }
    ]
}
@lujainzia127 → /workspaces/lab9 (main) $ |

```

3. (Do not) Delete key pair:

aws ec2 delete-key-pair --key-name MyED25519Key # Info: shows how to delete

Save screenshot as: task4_delete_keypair_optional.png — output of delete-key-pair (if performed).

4. Launch an EC2 instance (example values — replace IDs with ones from your account/region):

```

aws ec2 run-instances \
--image-id ami-05e66df2bafcb7dea \
--count 1 \
--instance-type t3.micro \
--key-name MyED25519Key \
--security-group-ids sg-EXAMPLE1234567890 \
--subnet-id subnet-EXAMPLE1234567890 \
--tag-specifications
"ResourceType=instance,Tags=[{Key=Name,Value=MyServer}]"

```

Save screenshot as: task4_run_instances_output.png — run-instances output with instance id.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 run-instances \
--image-id ami-05e66df2baefcb7dea \
--count 1 \
--instance-type t3.micro \
--key-name MyED25519Key \
--security-group-ids sg-0f5938330b9395b53 \
--subnet-id subnet-0b11cd485731a0ef7 \
--tag-specifications "ResourceType=instance,Tags=[{Key=Name,Value=MyServer}]"
{
  "ReservationId": "r-092d4ffd14ed07acf",
  "OwnerId": "050737808689",
  "Groups": [],
  "Instances": [
    {
      "Architecture": "x86_64",
      "BlockDeviceMappings": [],
      "ClientToken": "6e58d8be-a3dc-4429-ba3d-a88a48437e81",
      "EbsOptimized": false,
      "EnaSupport": true,
      "Hypervisor": "xen",
      "NetworkInterfaces": [
        {
          "Attachment": {
            "AttachTime": "2025-12-07T17:02:32+00:00",
            "AttachmentId": "eni-attach-0f93125eaba782e35",
            "DeleteOnTermination": true,
            "DeviceIndex": 0,
            "Status": "attaching",
            "NetworkCardIndex": 0
          },
          "Description": "",
          "Groups": [
            {
              "GroupId": "sg-0f5938330b9395b53",
              "GroupName": "MySecurityGroup"
            }
          ],
        },
        ...
      ],
      "ReservationId": "r-092d4ffd14ed07acf",
    }
  ]
}
```

5. Get the public IP address of your instance:

```
aws ec2 describe-instances \
--query "Reservations[*].Instances[*].[InstanceId,PublicIpAddress]" \
--output table
```

Save screenshot as: task4_describe_instances_public_ip.png — describe-instances table with public IP.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-instances \
--query "Reservations[*].Instances[*].[InstanceId,PublicIpAddress]" \
--output table
-----
|             DescribeInstances           |
+-----+-----+
| i-0eef7a2fd01f47780 | 40.172.122.174 |
+-----+-----+
@lujainzia127 → /workspaces/lab9 (main) $
```

6. Attempt SSH into the instance from the Codespace or from a machine whose IP is allowed in the security group:

```
ssh -i MyED25519Key.pem ec2-user@<public-ip-address>
```

If you see the error:

Permissions 0644 for 'MyED25519Key.pem' are too open. It is required that your private key files are NOT accessible by others.

fix permissions:

```
chmod 400 MyED25519Key.pem
```

```
ssh -i MyED25519Key.pem ec2-user@<public-ip-address>
```

Save screenshot as: task4_ssh_permission_error_and_fix.png — show permission error and chmod 400 then successful SSH (redact sensitive details).

```
@lujainzia127 → /workspaces/lab9 (main) $ ssh -i MyED25519Key.pem ec2-user@40.172.122.174
The authenticity of host '40.172.122.174 (40.172.122.174)' can't be established.
ED25519 key fingerprint is SHA256:cN93TzwFk/nusY+JUFCUNmrOLpQ6mr99oUXPTG+5ppU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '40.172.122.174' (ED25519) to the list of known hosts.
@          WARNING: UNPROTECTED PRIVATE KEY FILE!          @
Permissions 0666 for 'MyED25519Key.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "MyED25519Key.pem": bad permissions
ec2-user@40.172.122.174: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
@lujainzia127 → /workspaces/lab9 (main) $ chmod 400 MyED25519Key.pem
@lujainzia127 → /workspaces/lab9 (main) $ ssh -i MyED25519Key.pem ec2-user@40.172.122.174
#_
~\_ #####_      Amazon Linux 2023
~~ \#####\
~~ \###|
~~ \#/ --- https://aws.amazon.com/linux/amazon-linux-2023
~~ \~'`-->
~~ .-. /-
~/`/`/
[ec2-user@ip-172-31-38-208 ~]$ 
[ec2-user@ip-172-31-38-208 ~]$ |
```

7. Stop, start and (optionally) terminate the instance:

```
aws ec2 stop-instances --instance-ids i-EXAMPLE1234567890
aws ec2 start-instances --instance-ids i-EXAMPLE1234567890
aws ec2 terminate-instances --instance-ids i-EXAMPLE1234567890 # Don't
run this command
```

Save screenshot

as: task4_stop_start_terminate_commands.png — outputs for stop/start/terminate.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 start-instances --instance-ids i-0eef7a2fd01f47780
{
  "StartingInstances": [
    {
      "InstanceId": "i-0eef7a2fd01f47780",
      "CurrentState": {
        "Code": 0,
        "Name": "pending"
      },
      "PreviousState": {
        "Code": 80,
        "Name": "stopped"
      }
    }
  ]
}
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 stop-instances --instance-ids i-0eef7a2fd01f47780
{
  "StoppingInstances": [
    {
      "InstanceId": "i-0eef7a2fd01f47780",
      "CurrentState": {
        "Code": 64,
        "Name": "stopping"
      },
      "PreviousState": {
        "Code": 16,
        "Name": "running"
      }
    }
  ]
}
@lujainzia127 → /workspaces/lab9 (main) $ |
```

Task 5 — Understand AWS describe-* commands

1. Run and understand these commands (run each, then capture screenshot immediately after):

```
aws ec2 describe-security-groups
```

Save screenshot as: task5_describe_security_groups.png — output of describe-security-groups.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-security-groups
{
    "SecurityGroups": [
        {
            "GroupId": "sg-0851e2a278e0ffd34",
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "UserIdGroupPairs": [],
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": []
                }
            ],
            "VpcId": "vpc-01b7d554ea60c902d",
            "SecurityGroupArn": "arn:aws:ec2:me-central-1:050737808689:security-group/sg-0851e2a278e0ffd34",
            "OwnerId": "050737808689",
            "GroupName": "default",
            "Description": "default VPC security group",
            "IpPermissions": [
                {
                    "IpProtocol": "-1",
                    "UserIdGroupPairs": [
                        {
                            "UserId": "050737808689",
                            "GroupId": "sg-0851e2a278e0ffd34"
                        }
                    ],
                    "IpRanges": [],
                    "Ipv6Ranges": [],
                    "PrefixListIds": []
                }
            ]
        },
        {
            "GroupId": "sg-0f5938330b9395b53",
            "IpPermissionsEgress": [

```

aws ec2 describe-vpcs

Save screenshot as: task5_describe_vpcs.png — output of describe-vpcs.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-vpcs
{
    "Vpcs": [
        {
            "OwnerId": "050737808689",
            "InstanceTenancy": "default",
            "CidrBlockAssociationSet": [
                {
                    "AssociationId": "vpc-cidr-assoc-0f06f2ea342217e2f",
                    "CidrBlock": "172.31.0.0/16",
                    "CidrBlockState": {
                        "State": "associated"
                    }
                }
            ],
            "IsDefault": true,
            "BlockPublicAccessStates": {
                "InternetGatewayBlockMode": "off"
            },
            "VpcId": "vpc-01b7d554ea60c902d",
            "State": "available",
            "CidrBlock": "172.31.0.0/16",
            "DhcpOptionsId": "dopt-0378f1f3070a1db0c"
        }
    ]
}
@lujainzia127 → /workspaces/lab9 (main) $ |
```

aws ec2 describe-subnets

Save screenshot as: task5_describe_subnets.png — output of describe-subnets.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-subnets
{
    "Subnets": [
        {
            "AvailabilityZoneId": "mec1-az1",
            "MapCustomerOwnedIpOnLaunch": false,
            "OwnerId": "050737808689",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn": "arn:aws:ec2:me-central-1:050737808689:subnet/subnet-0b11cd485731a00f7",
            "EnableDns64": false,
            "Ipv6Native": false,
            "PrivateDnsNameOptionsOnLaunch": {
                "HostnameType": "ip-name",
                "EnableResourceNameDnsARecord": false,
                "EnableResourceNameDnsAAAARecord": false
            },
            "BlockPublicAccessStates": {
                "InternetGatewayBlockMode": "off"
            },
            "SubnetId": "subnet-0b11cd485731a00f7",
            "State": "available",
            "VpcId": "vpc-01b7d554ea60c902d",
            "CidrBlock": "172.31.32.0/20",
            "AvailableIpAddressCount": 4090,
            "AvailabilityZone": "me-central-1a",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true
        },
        {
            "AvailabilityZoneId": "mec1-az3",
            "MapCustomerOwnedIpOnLaunch": false,
            "OwnerId": "050737808689",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn": "arn:aws:ec2:me-central-1:050737808689:subnet/subnet-09a23e3b14a74df6e",
            "EnableDns64": false,
            "Ipv6Native": false,
            "PrivateDnsNameOptionsOnLaunch": {
                "HostnameType": "ip-name",
            }
        }
    ]
}
```

aws ec2 describe-instances

Save screenshot as: task5_describe_instances.png — output of describe-instances..

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-instances
{
    "Reservations": [
        {
            "ReservationId": "r-092d4ffd14ed07acf",
            "OwnerId": "050737808689",
            "Groups": [],
            "Instances": [
                {
                    "Architecture": "x86_64",
                    "BlockDeviceMappings": [
                        {
                            "DeviceName": "/dev/xvda",
                            "Ebs": {
                                "AttachTime": "2025-12-07T17:02:32+00:00",
                                "DeleteOnTermination": true,
                                "Status": "attached",
                                "VolumeId": "vol-0963f0fd3962da182"
                            }
                        }
                    ],
                    "ClientToken": "6e58d8be-a3dc-4429-ba3d-a88a48437e81",
                    "EbsOptimized": false,
                    "EnaSupport": true,
                    "Hypervisor": "xen",
                    "NetworkInterfaces": [
                        {
                            "Attachment": {
                                "AttachTime": "2025-12-07T17:02:32+00:00",
                                "AttachmentId": "eni-attach-0f93125eaba782e35",
                                "DeleteOnTermination": true,
                                "DeviceIndex": 0,
                                "Status": "attached",
                                "NetworkCardIndex": 0
                            },
                            "Description": "",
                            "Groups": [
                                {
                                    "GroupId": "sg-0f5938330b9395b53",
                                    "GroupName": "MySecurityGroup"
                                }
                            ]
                        }
                    ],
                    "State": "running"
                }
            ]
        }
    ]
}
```

aws ec2 describe-regions

Save screenshot as: task5_describe_regions.png — output of describe-regions.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-regions
{
    "Regions": [
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "ap-south-1",
            "Endpoint": "ec2.ap-south-1.amazonaws.com"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "eu-north-1",
            "Endpoint": "ec2.eu-north-1.amazonaws.com"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "eu-west-3",
            "Endpoint": "ec2.eu-west-3.amazonaws.com"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "eu-west-2",
            "Endpoint": "ec2.eu-west-2.amazonaws.com"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "eu-west-1",
            "Endpoint": "ec2.eu-west-1.amazonaws.com"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "ap-northeast-3",
            "Endpoint": "ec2.ap-northeast-3.amazonaws.com"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "ap-northeast-2",
            "Endpoint": "ec2.ap-northeast-2.amazonaws.com"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "RegionName": "ap-northeast-1",
            "Endpoint": "ec2.ap-northeast-1.amazonaws.com"
        }
    ]
}
```

aws ec2 describe-availability-zones

Save screenshot as: task5_describe_availability_zones.png — output of describe-availability-zones.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-availability-zones
{
    "AvailabilityZones": [
        {
            "OptInStatus": "opt-in-not-required",
            "Messages": [],
            "RegionName": "me-central-1",
            "ZoneName": "me-central-1a",
            "ZoneId": "mec1-az1",
            "GroupName": "me-central-1-zg-1",
            "NetworkBorderGroup": "me-central-1",
            "ZoneType": "availability-zone",
            "GroupLongName": "Middle East (UAE) 1",
            "State": "available"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "Messages": [],
            "RegionName": "me-central-1",
            "ZoneName": "me-central-1b",
            "ZoneId": "mec1-az2",
            "GroupName": "me-central-1-zg-1",
            "NetworkBorderGroup": "me-central-1",
            "ZoneType": "availability-zone",
            "GroupLongName": "Middle East (UAE) 1",
            "State": "available"
        },
        {
            "OptInStatus": "opt-in-not-required",
            "Messages": [],
            "RegionName": "me-central-1",
            "ZoneName": "me-central-1c",
            "ZoneId": "mec1-az3",
            "GroupName": "me-central-1-zg-1",
            "NetworkBorderGroup": "me-central-1",
            "ZoneType": "availability-zone",
            "GroupLongName": "Middle East (UAE) 1",
            "State": "available"
        }
    ]
}
```

Task 6 — IAM: create group, user, attach policies, create console login & keys

1. Create group:

```
aws iam create-group --group-name MyGroupCli
```

Save screenshot as: task6_create_group_and_user.png — create-group output.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws iam create-group --group-name MyGroupCli
{
    "Group": {
        "Path": "/",
        "GroupName": "MyGroupCli",
        "GroupId": "AGPAQXUBVQUYSIROVQDCY",
        "Arn": "arn:aws:iam::050737808689:group/MyGroupCli",
        "CreateDate": "2025-12-07T17:30:20+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $
```

2. Get group details:

```
aws iam get-group --group-name MyGroupCli
```

Save screenshot as: task6_create_group_and_user.png — get-group output.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws iam get-group --group-name MyGroupCli
{
    "Users": [],
    "Group": {
        "Path": "/",
        "GroupName": "MyGroupCli",
        "GroupId": "AGPAQXUBVQUYSIROVQDCY",
        "Arn": "arn:aws:iam::050737808689:group/MyGroupCli",
        "CreateDate": "2025-12-07T17:30:20+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $
```

3. Create user:

```
aws iam create-user --user-name MyUserCli
```

Save screenshot as: task6_create_group_and_user.png — create-user output.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws iam create-user --user-name MyUserCli
{
    "User": {
        "Path": "/",
        "UserName": "MyUserCli",
        "UserId": "AIDAQXUBVQUY7KWMZSQEK",
        "Arn": "arn:aws:iam::050737808689:user/MyUserCli",
        "CreateDate": "2025-12-07T17:46:30+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $ aws iam get-user --user-name MyUserCli
{
    "User": {
        "Path": "/",
        "UserName": "MyUserCli",
        "UserId": "AIDAQXUBVQUY7KWMZSQEK",
        "Arn": "arn:aws:iam::050737808689:user/MyUserCli",
        "CreateDate": "2025-12-07T17:46:30+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $
```

4. Get user details:

```
aws iam get-user --user-name MyUserCli
```

Save screenshot as: task6_create_group_and_user.png — get-user output.

```

@lujainzia127 → /workspaces/lab9 (main) $ aws iam create-user --user-name MyUserCli
{
    "User": {
        "Path": "/",
        "UserName": "MyUserCli",
        "UserId": "AIDAQXUBVQUY7KWMZSQEK",
        "Arn": "arn:aws:iam::050737808689:user/MyUserCli",
        "CreateDate": "2025-12-07T17:46:30+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $ aws iam get-user --user-name MyUserCli
{
    "User": {
        "Path": "/",
        "UserName": "MyUserCli",
        "UserId": "AIDAQXUBVQUY7KWMZSQEK",
        "Arn": "arn:aws:iam::050737808689:user/MyUserCli",
        "CreateDate": "2025-12-07T17:46:30+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $

```

5. Add user to group:

aws iam add-user-to-group --user-name MyUserCli --group-name MyGroupCli

Save screenshot

as: task6_add_user_to_group_and_verify.png — add-user-to-group and verify with get-group.

```

@lujainzia127 → /workspaces/lab9 (main) $ aws iam add-user-to-group --user-name MyUserCli --group-name MyGroupCli
@lujainzia127 → /workspaces/lab9 (main) $ aws iam get-group --group-name MyGroupCli
{
    "Users": [
        {
            "Path": "/",
            "UserName": "MyUserCli",
            "UserId": "AIDAQXUBVQUY7KWMZSQEK",
            "Arn": "arn:aws:iam::050737808689:user/MyUserCli",
            "CreateDate": "2025-12-07T17:46:30+00:00"
        }
    ],
    "Group": {
        "Path": "/",
        "GroupName": "MyGroupCli",
        "GroupId": "AGPAQXUBVQUYSIROVQDCY",
        "Arn": "arn:aws:iam::050737808689:group/MyGroupCli",
        "CreateDate": "2025-12-07T17:30:20+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $

```

6. See group again:

aws iam get-group --group-name MyGroupCli

Save screenshot

as: task6_add_user_to_group_and_verify.png — get-group showing user present.

```

@lujainzia127 → /workspaces/lab9 (main) $ aws iam add-user-to-group --user-name MyUserCli --group-name MyGroupCli
@lujainzia127 → /workspaces/lab9 (main) $ aws iam get-group --group-name MyGroupCli
{
    "Users": [
        {
            "Path": "/",
            "UserName": "MyUserCli",
            "UserId": "AIDAQXUBVQUY7KWMZSQEK",
            "Arn": "arn:aws:iam::050737808689:user/MyUserCli",
            "CreateDate": "2025-12-07T17:46:30+00:00"
        }
    ],
    "Group": {
        "Path": "/",
        "GroupName": "MyGroupCli",
        "GroupId": "AGPAQXUBVQUYSIROVQDCY",
        "Arn": "arn:aws:iam::050737808689:group/MyGroupCli",
        "CreateDate": "2025-12-07T17:30:20+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $

```

7. List policies that mention EC2:

aws iam list-policies \

```
--query "Policies[?contains(PolicyName, 'EC2')].{Name:PolicyName}" \
--output text
```

Save screenshot as: task6_policy_list_and_attach.png — policy list output.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws iam list-policies \
--query "Policies[?contains(PolicyName, 'EC2')].{Name:PolicyName}" \
--output text
AmazonEC2FullAccess
AmazonEC2ReadOnlyAccess
AmazonElasticMapReduceforEC2Role
AmazonEC2RoleforDataPipelineRole
AmazonEC2ContainerServiceforEC2Role
AmazonEC2ContainerServiceRole
AmazonEC2RoleforAWSCodeDeploy
AmazonEC2RoleForSSM
CloudWatchActionsEC2Access
AmazonEC2ContainerRegistryReadOnly
AmazonEC2ContainerRegistryPowerUser
AmazonEC2ContainerRegistryFullAccess
AmazonEC2ContainerServiceAutoscaleRole
AmazonEC2SpotFleetAutoscaleRole
AWSElasticBeanstalkCustomPlatformforEC2Role
AmazonEC2ContainerServiceEventsRole
AmazonEC2SpotFleetTaggingRole
AWSEC2SpotServiceRolePolicy
AWSERVICERoleForEC2ScheduledInstances
AWSEC2SpotFleetServiceRolePolicy
AWSApplicationAutoScalingEC2SpotFleetRequestPolicy
AWSEC2FleetServiceRolePolicy
AWSAutoScalingPlansEC2AutoScalingPolicy
EC2InstanceConnect
AmazonEC2RolePolicyForLaunchWizard
EC2InstanceProfileForImageBuilder
EC2FleetTimeShiftableServiceRolePolicy
AmazonEC2RoleforAWSCodeDeployLimited
EC2InstanceProfileForImageBuilderECRContainerBuilds
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

8. Get ARN for AmazonEC2FullAccess (example query):

```
aws iam list-policies --query
'Policies[?PolicyName==`AmazonEC2FullAccess`].{Name:PolicyName,
ARN:Arn}' --output table
```

Save screenshot as: task6_policy_list_and_attach.png — ARN query output.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws iam list-policies \
--query 'Policies[?PolicyName==`AmazonEC2FullAccess`].{Name:PolicyName, ARN:Arn}' \
--output table
+-----+-----+
|      ListPolicies      |
+-----+-----+
|      ARN      |      Name      |
+-----+-----+
| arn:aws:iam::aws:policy/AmazonEC2FullAccess | AmazonEC2FullAccess |
+-----+-----+
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

9. Attach policy to group (use the ARN you retrieved):

```
aws iam attach-group-policy \
--group-name MyGroupCli \
--policy-arn arn:aws:iam::aws:policy/EXAMPLEPolicyName
```

Save screenshot as: task6_policy_list_and_attach.png — attach-group-policy output.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws iam attach-group-policy \
--group-name MyGroupCli \
--policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess
@lujainzia127 ~ /workspaces/lab9 (main) $ aws iam list-attached-group-policies --group-name MyGroupCli
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonEC2FullAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonEC2FullAccess"
        }
    ]
}
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

10. List attached policies for the group:

aws iam list-attached-group-policies --group-name MyGroupCli

Save screenshot as: task6_policy_list_and_attach.png — list-attached-group-policies output.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws iam attach-group-policy \
--group-name MyGroupCli \
--policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess
@lujainzia127 ~ /workspaces/lab9 (main) $ aws iam list-attached-group-policies --group-name MyGroupCli
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonEC2FullAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonEC2FullAccess"
        }
    ]
}
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

11. Create a console login profile for the user:

```
aws iam create-login-profile \
--user-name MyUserCli \
--password <PASSWORD_VALUE> \
--password-reset-required
```

Save screenshot

as: task6_create_login_profile_and_signin.png — create-login-profile output (do not show password).

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws iam create-login-profile \
--user-name MyUserCli \
--password [REDACTED] \
--password-reset-required
{
    "LoginProfile": {
        "UserName": "MyUserCli",
        "CreateDate": "2025-12-07T17:52:07+00:00",
        "PasswordResetRequired": true
    }
}
```

12. If the user cannot change password, attach

IAMUserChangePassword to the group temporarily:

```
aws iam attach-group-policy --group-name MyGroupCli --policy-arn
arn:aws:iam::aws:policy/IAMUserChangePassword
```

After the user logs in and resets password, detach that policy:

```
aws iam detach-group-policy --group-name MyGroupCli --policy-arn
arn:aws:iam::aws:policy/IAMUserChangePassword
```

Save screenshot

as: task6_create_login_profile_and_signin.png — attach/detach outputs and a screenshot showing the user login (redact password).

```

@lujainzia127 → /workspaces/lab9 (main) $ aws iam attach-group-policy \
    --group-name MyGroupCli \
    --policy-arn arn:aws:iam::aws:policy/IAMUserChangePassword
@lujainzia127 → /workspaces/lab9 (main) $ aws iam detach-group-policy \
    --group-name MyGroupCli \
    --policy-arn arn:aws:iam::aws:policy/IAMUserChangePassword
@lujainzia127 → /workspaces/lab9 (main) $ |

```

13. Create access keys for the user (save AccessKeyId and SecretAccessKey securely):

aws iam create-access-key --user-name MyUserCli

Save screenshot as: task6_create_access_key_output.png — create-access-key output (redact keys).

```

@lujainzia127 → /workspaces/lab9 (main) $ aws iam create-access-key --user-name MyUserCli
{
    "AccessKey": {
        "UserName": "MyUserCli",
        "AccessKeyId": "AKIAQXUBVQUYSMI5AVPO",
        "Status": "Active",
        "SecretAccessKey": "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",
        "CreateDate": "2025-12-07T18:03:58+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $ aws iam list-access-keys --user-name MyUserCli
{
    "AccessKeyMetadata": [
        {
            "UserName": "MyUserCli",
            "AccessKeyId": "AKIAQXUBVQUYSMI5AVPO",
            "Status": "Active",
            "CreateDate": "2025-12-07T18:03:58+00:00"
        }
    ]
}
@lujainzia127 → /workspaces/lab9 (main) $ |

```

14. List access keys:

aws iam list-access-keys --user-name MyUserCli

Save screenshot as: task6_create_access_key_output.png — list-access-keys output.

```

@lujainzia127 → /workspaces/lab9 (main) $ aws iam create-access-key --user-name MyUserCli
{
    "AccessKey": {
        "UserName": "MyUserCli",
        "AccessKeyId": "AKIAQXUBVQUYSMI5AVPO",
        "Status": "Active",
        "SecretAccessKey": "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",
        "CreateDate": "2025-12-07T18:03:58+00:00"
    }
}
@lujainzia127 → /workspaces/lab9 (main) $ aws iam list-access-keys --user-name MyUserCli
{
    "AccessKeyMetadata": [
        {
            "UserName": "MyUserCli",
            "AccessKeyId": "AKIAQXUBVQUYSMI5AVPO",
            "Status": "Active",
            "CreateDate": "2025-12-07T18:03:58+00:00"
        }
    ]
}
@lujainzia127 → /workspaces/lab9 (main) $ |

```

15. (Don't) Delete access key:

aws iam delete-access-key --user-name MyUserCli --access-key-id <AccessKeyId> # Don't run this command

Save screenshot as: task6_create_access_key_output.png — delete-access-key output (if performed).

16. Use environment variables to authenticate as that user in the Codespace:

```
export AWS_ACCESS_KEY_ID=<YOUR_ACCESS_KEY_ID>export  
AWS_SECRET_ACCESS_KEY=<YOUR_SECRET_ACCESS_KEY>  
printenv | grep AWS_  
aws iam get-user --user-name MyUserCli # may fail if no permissionsexit  
# to clear exports
```

Save screenshot

as: task6_env_exports_and_get_user_error.png — show env exports and any AccessDenied error (if occurs).

```
@lujainzia127 → /workspaces/lab9 (main) $ printenv | grep AWS_  
aws iam get-user --user-name MyUserCli  
AWS_SECRET_ACCESS_KEY=  
AWS_ACCESS_KEY_ID=  
{  
    "User": {  
        "Path": "/",  
        "UserName": "MyUserCli",  
        "UserId": "AIDAQXUBVQUY7KWMZSQEK",  
        "Arn": "arn:aws:iam:050737808689:user/MyUserCli",  
        "CreateDate": "2025-12-07T17:46:30+00:00",  
        "PasswordLastUsed": "2025-12-07T18:35:15+00:00",  
        "Tags": [  
            {  
                "Key": "AKIAQXUBVQUYQSJ5I6AX",  
                "Value": "jje"  
            }  
        ]  
    }  
}  
@lujainzia127 → /workspaces/lab9 (main) $
```

After clearing or switching credentials, repeat get-user and save:

Save screenshot

as: task6_after_logout_and_get_user_success.png — successful get-user output under appropriate credentials.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws configure  
AWS Access Key ID [*****6AX]:  
AWS Secret Access Key [*****AMkS]:  
Default region name [me-central-1]: me-central-1  
Default output format [json]: json  
@lujainzia127 → /workspaces/lab9 (main) $ aws iam get-user --user-name MyUserCli  
{  
    "User": {  
        "Path": "/",  
        "UserName": "MyUserCli",  
        "UserId": "AIDAQXUBVQUY7KWMZSQEK",  
        "Arn": "arn:aws:iam:050737808689:user/MyUserCli",  
        "CreateDate": "2025-12-07T17:46:30+00:00",  
        "PasswordLastUsed": "2025-12-07T18:35:15+00:00",  
        "Tags": [  
            {  
                "Key": "AKIAQXUBVQUYQSJ5I6AX",  
                "Value": "jje"  
            }  
        ]  
    }  
}  
@lujainzia127 → /workspaces/lab9 (main) $ |
```

Task 7 — Filters: query with filters to find instances and their attributes

Examples (run each and take a screenshot immediately after):

```
aws ec2 describe-instances \
--filters "Name=tag:Name,Values=MyServer" \
--query "Reservations[*].Instances[*].PublicIpAddress" \
--output text
```

Save screenshot as: task7_filter_by_tag_public_ip.png — output of the filter by tag showing public IP.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 describe-instances --filters "Name=tag:Name,Values=MyServer" --query "Reservations[*].Instances[*].PublicIpAddress" --output text
40.172.101.161
@lujainzia127 ~ /workspaces/lab9 (main) $
```

```
aws ec2 describe-instances \
--filters "Name=instance-type,Values=t3.micro" \
--query "Reservations[].Instances[].InstanceId" \
--output table
```

Save screenshot as: task7_filter_by_instance_type.png — output listing instance IDs.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 describe-instances \
--filters "Name=instance-type,Values=t3.micro" \
--query "Reservations[].Instances[].InstanceId" \
--output table
-----
|  DescribeInstances   |
+-----+
| i-0eef7a2fd01f47780 |
+-----+
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

```
aws ec2 describe-instances \
--filters "Name=subnet-id,Values=subnet-0600df5fa8ce60857" \
--query "Reservations[*].Instances[*].InstanceId" \
--output table
```

Save screenshot as: task7_filter_by_subnet.png — output for subnet filter.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 describe-instances --filters "Name=subnet-id,Values= subnet-0b11cd485731a00f7" --query "Reservations[*].Instances[*].InstanceId" --output table
-----
|  DescribeInstances   |
+-----+
| i-0eef7a2fd01f47780 |
+-----+
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

```
aws ec2 describe-instances \
--filters "Name=vpc-id,Values=vpc-06be85cd81b657192" \
--query "Reservations[*].Instances[*].InstanceId" \
--output table
```

Save screenshot as: task7_filter_by_vpc.png — output for VPC filter.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 describe-instances --filters "Name=vpc-id,Values=vpc-01b7d554ea60c902d" --query "Reservations[*].Instances[*].InstanceId" --output table
-----
|  DescribeInstances   |
+-----+
| i-0eef7a2fd01f47780 |
+-----+
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

Task 8 — Use --query to format outputs for reporting

Examples (run each and take a screenshot immediately after):

```
aws ec2 describe-instances \
--filters "Name=tag:Name,Values=MyServer" \
```

```
--query  
"Reservations[*].Instances[*].[InstanceId,PublicIpAddress,Tags[?Key=='Name'  
].Value|[0]]" \  
--output table
```

Save screenshot

as: task8_query_table_instances_name_ip.png — table showing InstanceId, PublicIpAddress, Name.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-instances \  
  --filters "Name=tag:Name,Values=MyServer" \  
  --query "Reservations[*].Instances[*].[InstanceId,PublicIpAddress,Tags[?Key=='Name'].Value|[0]]" \  
  --output table  
-----  
|           DescribeInstances           |  
+-----+-----+-----+  
| i-0eef7a2fd01f47780 | 40.172.101.161 | MyServer |  
+-----+-----+-----+  
@lujainzia127 → /workspaces/lab9 (main) $  
aws ec2 describe-instances \  
  --query "Reservations[*].Instances[*].[InstanceId,State.Name]" \  
  --output table
```

Save screenshot as: task8_query_table_instance_state.png — table showing InstanceId and State.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-instances \  
  --query "Reservations[*].Instances[*].[InstanceId,State.Name]" \  
  --output table  
-----  
|           DescribeInstances           |  
+-----+-----+  
| i-0eef7a2fd01f47780 | running |  
+-----+-----+  
@lujainzia127 → /workspaces/lab9 (main) $  
aws ec2 describe-instances \  
  --query  
"Reservations[*].Instances[*].[InstanceId,InstanceType,Placement.Availability  
Zone]" \  
  --output table
```

Save screenshot

as: task8_query_table_instance_type_az.png — table showing InstanceId, InstanceType, AvailabilityZone.

```
@lujainzia127 → /workspaces/lab9 (main) $ aws ec2 describe-instances \  
  --query "Reservations[*].Instances[*].[InstanceId,InstanceType,Placement.AvailabilityZone]" \  
  --output table  
-----  
|           DescribeInstances           |  
+-----+-----+-----+  
| i-0eef7a2fd01f47780 | t3.micro | me-central-1a |  
+-----+-----+-----+  
@lujainzia127 → /workspaces/lab9 (main) $ |
```

Cleanup -- Remove resources to avoid charges

After verification, terminate and delete everything you created in AWS. Capture screenshots for each step.

Cleanup steps and required screenshots (take each screenshot immediately after running the command):

1. Terminate instances:

```
aws ec2 terminate-instances --instance-ids i-EXAMPLE1234567890
```

Save screenshot as: cleanup_terminate_instance.png — terminate-instances output/confirmation.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 terminate-instances --instance-ids i-0eef7a2fd01f47780
{
    "TerminatingInstances": [
        {
            "InstanceId": "i-0eef7a2fd01f47780",
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
@lujainzia127 ~ /workspaces/lab9 (main) $
```

2. Delete EBS volumes & snapshots (if any):

Save screenshot as: cleanup_delete_volumes_snapshots.png — confirmation or listing showing volumes/snapshots deleted.

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 delete-volume --volume-id vol-0963f0fd3962da182
An error occurred (InvalidVolume.NotFound) when calling the DeleteVolume operation: The volume 'vol-0963f0fd3962da182' does not exist.
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 describe-volumes --query "Volumes[*].{ID:VolumeId,State:State}" --output table
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 describe-snapshots --owner-ids self --query "Snapshots[*].SnapshotId" --output table
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

3. Delete security group and key pair:

```
aws ec2 delete-security-group --group-id sg-EXAMPLE1234567890
aws ec2 delete-key-pair --key-name MyED25519Key
```

Save screenshot as: cleanup_delete_security_group_and_keypair.png — deletion confirmation(s).

```
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 delete-security-group --group-id sg-0f5938330b9395b53
{
    "Return": true,
    "GroupId": "sg-0f5938330b9395b53"
}
@lujainzia127 ~ /workspaces/lab9 (main) $ aws ec2 delete-key-pair --key-name MyED25519Key
{
    "Return": true,
    "KeyPairId": "key-062911b808fe9dd58"
}
@lujainzia127 ~ /workspaces/lab9 (main) $ |
```

4. Remove IAM users, access keys, groups:

```
aws iam delete-access-key --user-name MyUserCli --access-key-id <AccessKeyId>
aws iam delete-login-profile --user-name MyUserCli
aws iam remove-user-from-group --user-name MyUserCli --group-name MyGroupCli
aws iam delete-user --user-name MyUserCli
aws iam detach-group-policy --group-name MyGroupCli --policy-arn arn:aws:iam::aws:policy/EXAMPLEPolicyName
aws iam delete-group --group-name MyGroupCli
```

Save screenshot as: cleanup_iam_users_deleted.png — IAM deletion commands and confirmation.

```
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam delete-access-key --user-name MyUserCli --access-key-id AKIAQXUBVQUYQ5J5I6AX
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam delete-login-profile --user-name MyUserCli
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam remove-user-from-group --user-name MyUserCli --group-name MyGroupCli
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam delete-user --user-name MyUserCli

An error occurred (DeleteConflict) when calling the DeleteUser operation: Cannot delete entity, must detach all policies first.
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam detach-group-policy --group-name MyGroupCli --policy-arn arn:aws:iam::aws:policy/EXAMPLEPolicyName

An error occurred (NoSuchEntity) when calling the DetachGroupPolicy operation: Policy arn:aws:iam::aws:policy/EXAMPLEPolicyName was not found.
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam detach-group-policy --group-name MyGroupCli --policy-arn arn:aws:iam::aws:policy/AdministratorAccess

An error occurred (NoSuchEntity) when calling the DetachGroupPolicy operation: Policy arn:aws:iam::aws:policy/AdministratorAccess was not found.
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam detach-group-policy --group-name MyGroupCli --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess
[@lujainzia127 ~] /workspaces/lab9 {main} $ Connection to localhost closed by remote host.

Connection to localhost closed.
shell closed: exit status 0xffffffff
PS C:\Users\user> gh codespace ssh -c zany-disco-jjp576x7r5j93p66
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-1030-azure x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro
Last login: Sun Dec  7 18:48:55 2025 from ::1
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam detach-group-policy --group-name MyGroupCli --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess

An error occurred (NoSuchEntity) when calling the DetachGroupPolicy operation: Policy arn:aws:iam::aws:policy/AmazonEC2FullAccess was not found.
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam delete-user --user-name MyUserCli
[@lujainzia127 ~] /workspaces/lab9 {main} $ aws iam delete-group --group-name MyGroupCli
[@lujainzia127 ~] /
```

5. Final verification (billing/resource groups):

Save screenshot as: cleanup_summary.png — final console verification (billing/resource groups) showing no active resources if possible.

Page refresh time: Monday, December 8, 2025 at 12:29:05 AM GMT+5

AWS estimated bill summary Info		
Total charges and payment information		
Account ID 050737808689	Billing period Info December 1 - December 31, 2025	Bill status Info Pending
Service provider Amazon Web Services, Inc.		Total in USD USD 0.00
Estimated grand total:		USD 0.00
▶ Payment information Info		