Name: Lujain Zia
Roll no: 2023-BSE-034
Date: December 2, 2025

Fatima Jinnah Women University

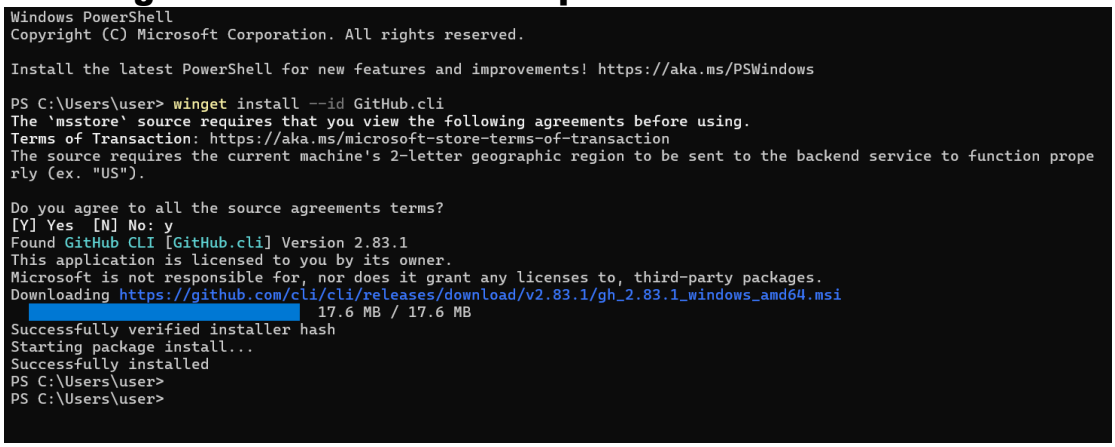
Lab 9

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.



2. (Local) Authenticate GH CLI for Codespaces:

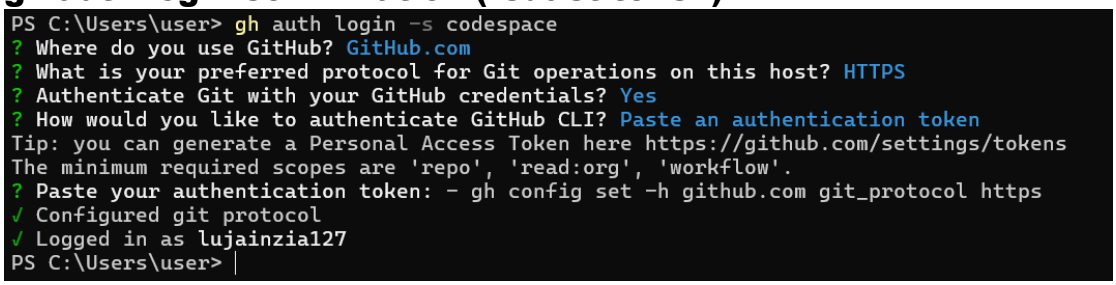
gh auth login -s codespace

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).



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

gh codespace list

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

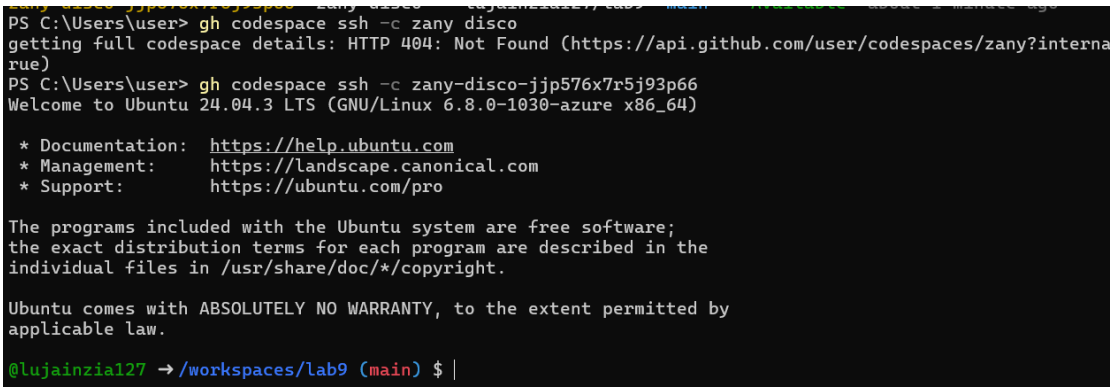
4. (Local) Create or connect to a Codespace. To create a new codespace from the current repo:

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>.

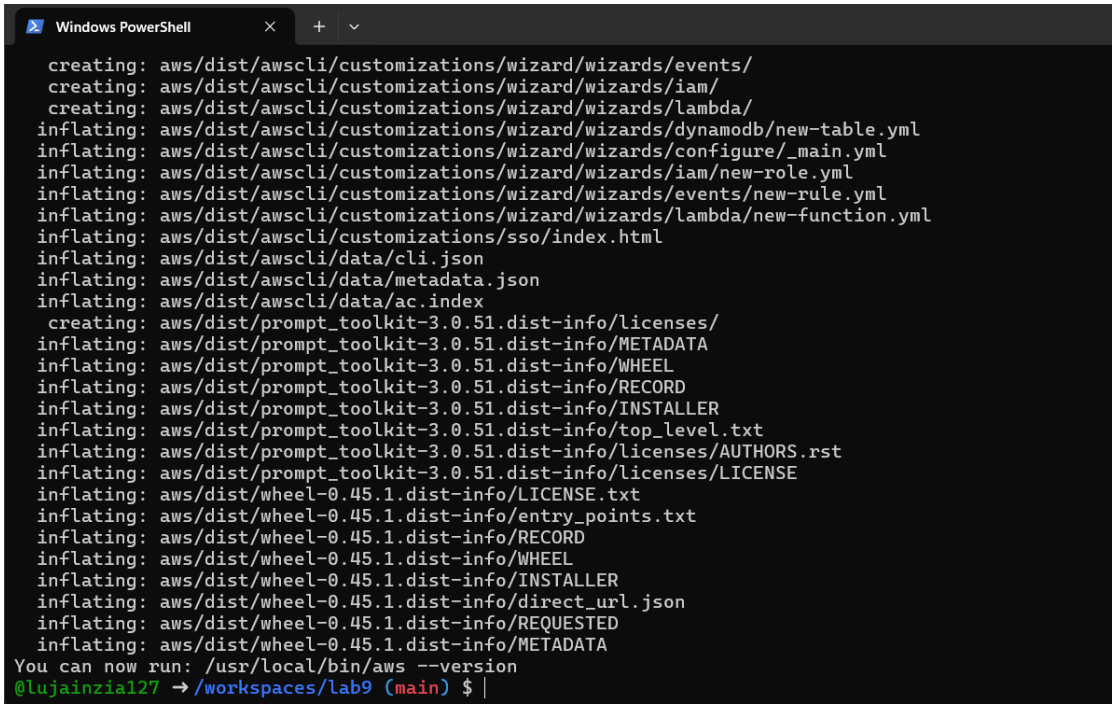


Task 2 — Install AWS CLI inside the Codespace and configure it

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

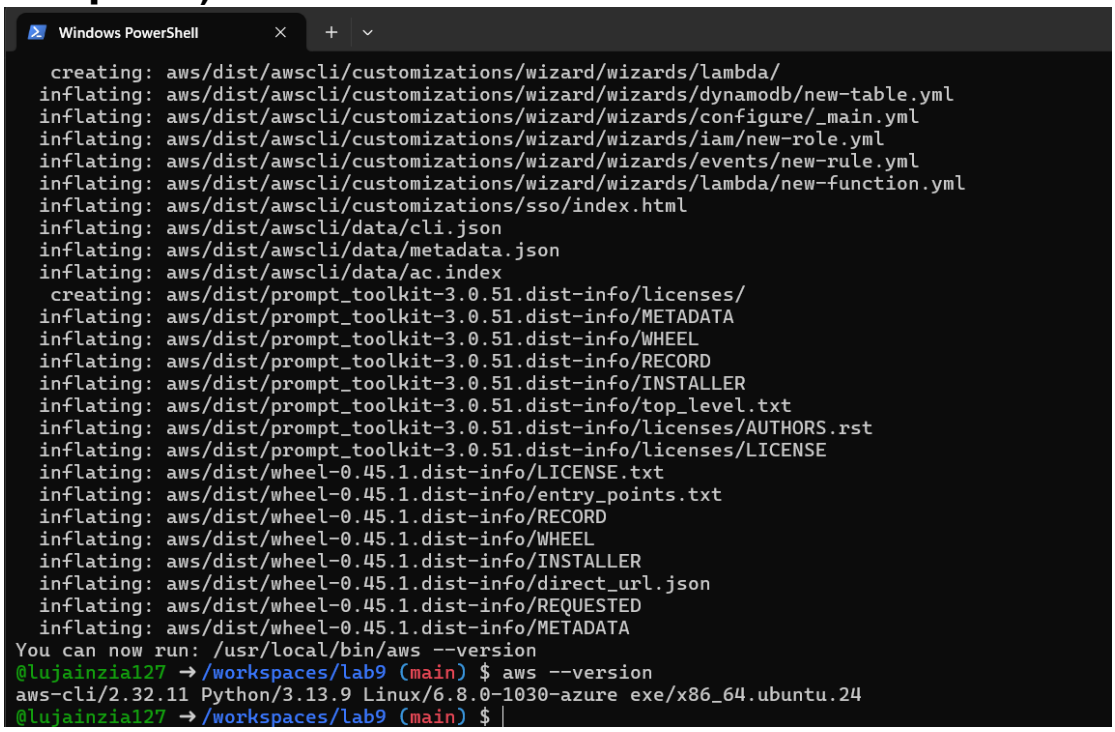
Save screenshot as: task2_aws_install_and_version.png — terminal showing the download/unzip/install output.



2. Verify installation:

aws --version

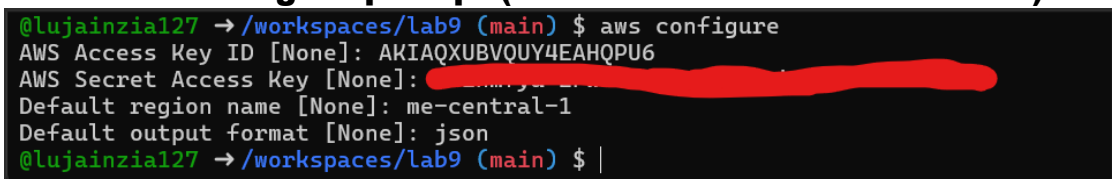
Save screenshot as: task2_aws_install_and_version.png — include aws --version output (same file as install output is acceptable).



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):

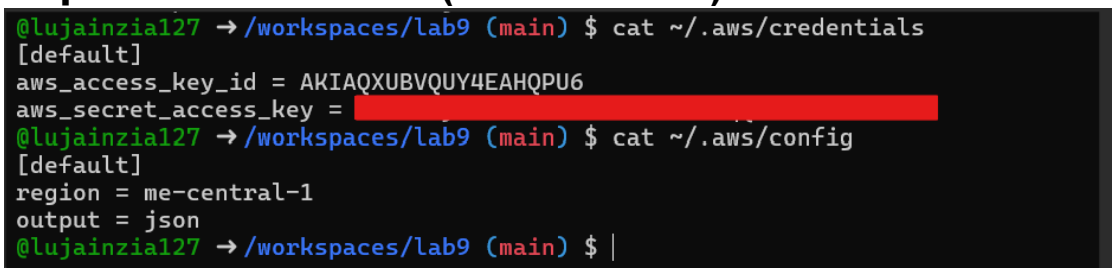
aws configure

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).



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).



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
```

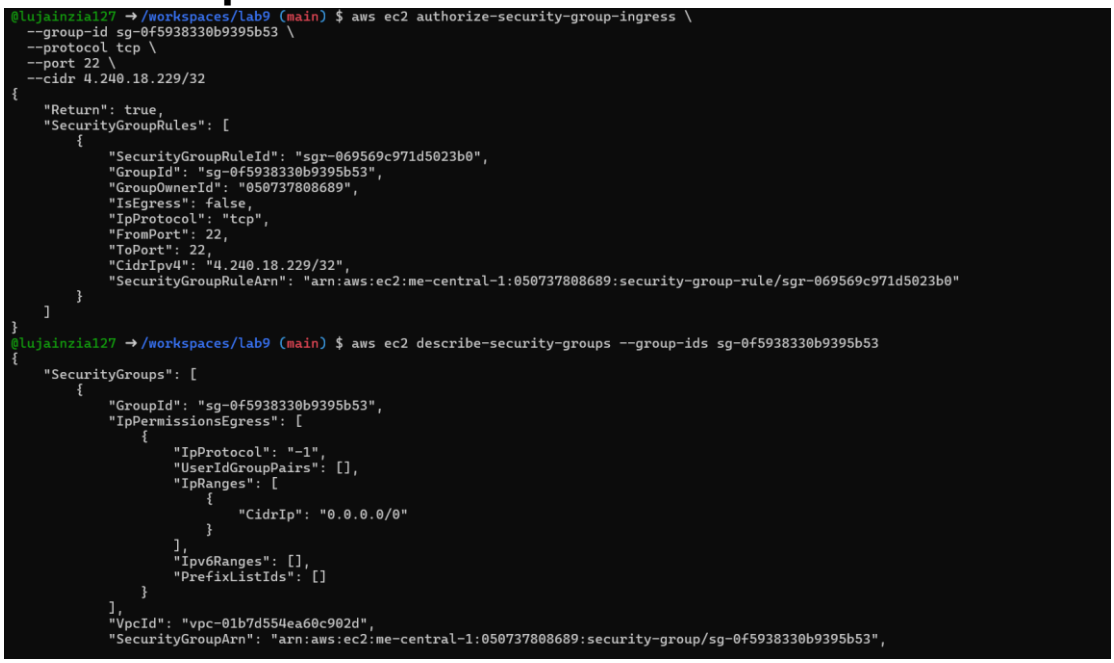
Save screenshot as: task3_codespace_public_ip.png — output of curl icanhazip.com.

```
@lujainzia127 ➜ /workspaces/lab9 (main) $ curl icanhazip.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.

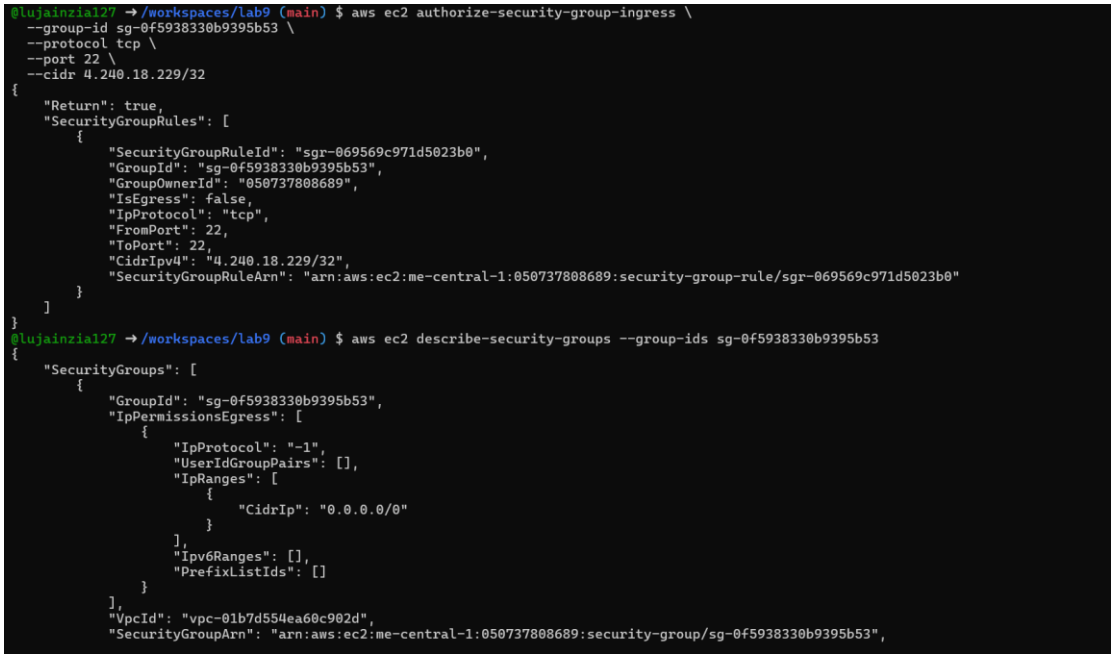


```
@lujainzia127 ➜ /workspaces/lab9 (main) $ aws ec2 authorize-security-group-ingress \
  --group-id sg-0f5938330b9395b53 \
  --protocol tcp \
  --port 22 \
  --cidr 4.240.18.229/32
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-069569c971d5023b0",
            "GroupId": "sg-0f5938330b9395b53",
            "GroupOwnerId": "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",
```

5. Verify ingress rule appears:

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

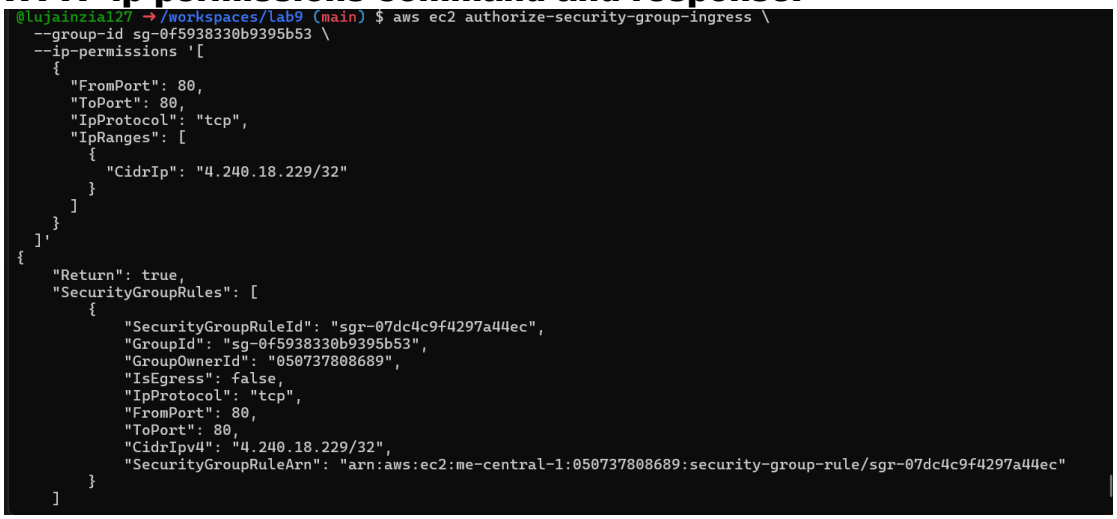
Save screenshot as: task3_authorize_ssh_and_describe.png — describe output showing SSH ingress (same file as previous step is acceptable).



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.XXX.XXX.XXX>/32"}]}'

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



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
```

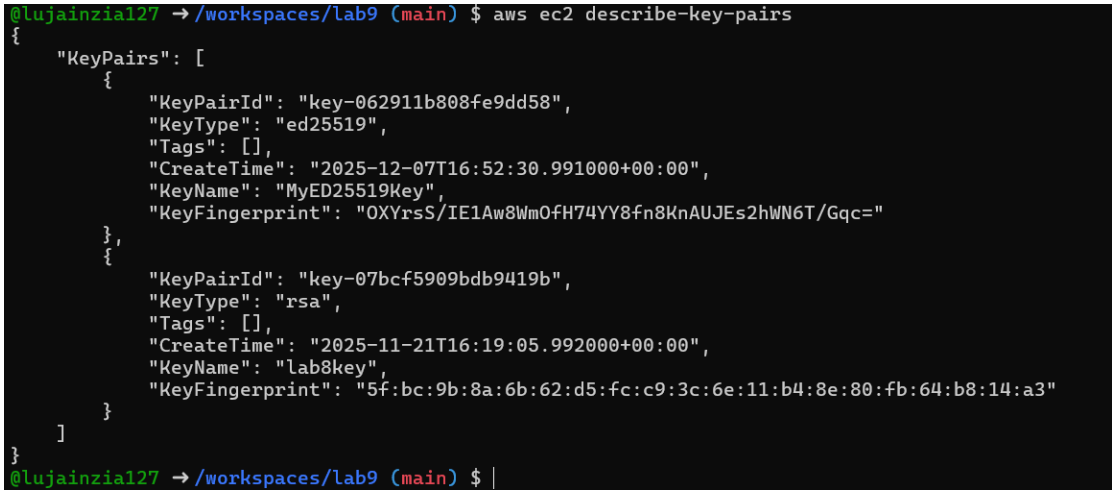
Save screenshot as: task4_create_keypair_output.png — output of create-key-pair and ls -l MyED25519Key.pem.

```
@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.



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-05e66df2bafcb7dea \
  --count 1 \
  --instance-type t3.micro \
  --key-name MyED25519Key \
  --security-group-ids sg-0f5938330b9395b53 \
  --subnet-id subnet-0b11cd485731a00f7 \
  --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"
                        }
                    ],
:...skipping...
{
    "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.

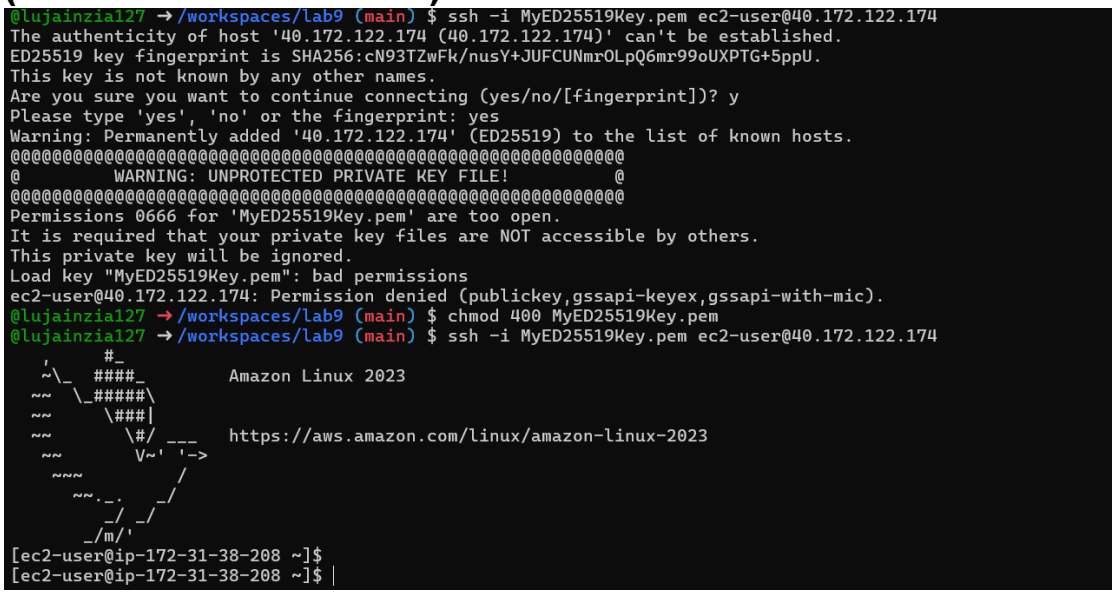
```
@lujainzia127 ➜ /workspaces/lab9 (main) $ aws ec2 describe-instances \
  --query "Reservations[*].Instances[*].[InstanceId,PublicIpAddress]" \
  --output table
-------------------------------------------
|            DescribeInstances            |
+----------------------+------------------+
|  i-0eef7a2fd01f47780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

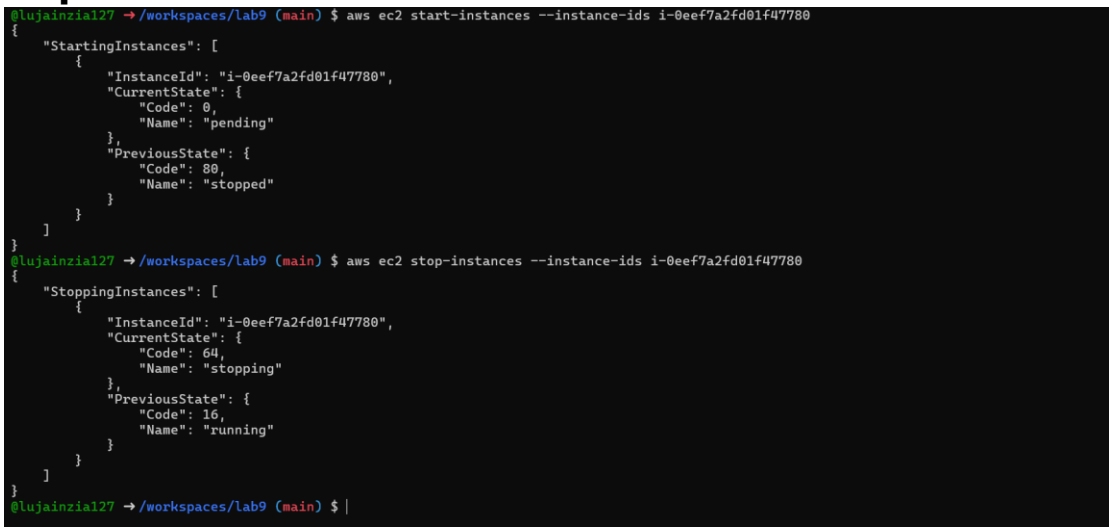
Save screenshot as: task4_ssh_permission_error_and_fix.png — show permission error and chmod 400 then successful SSH (redact sensitive details).



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.

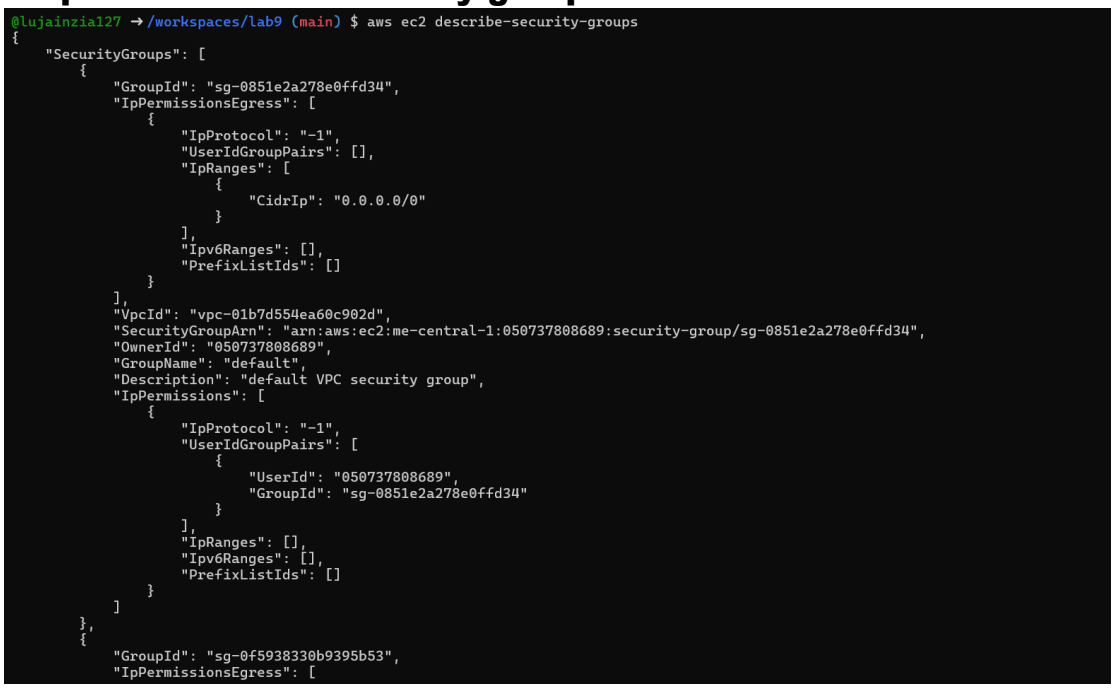


Task 5 — Understand AWS describe-* commands

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

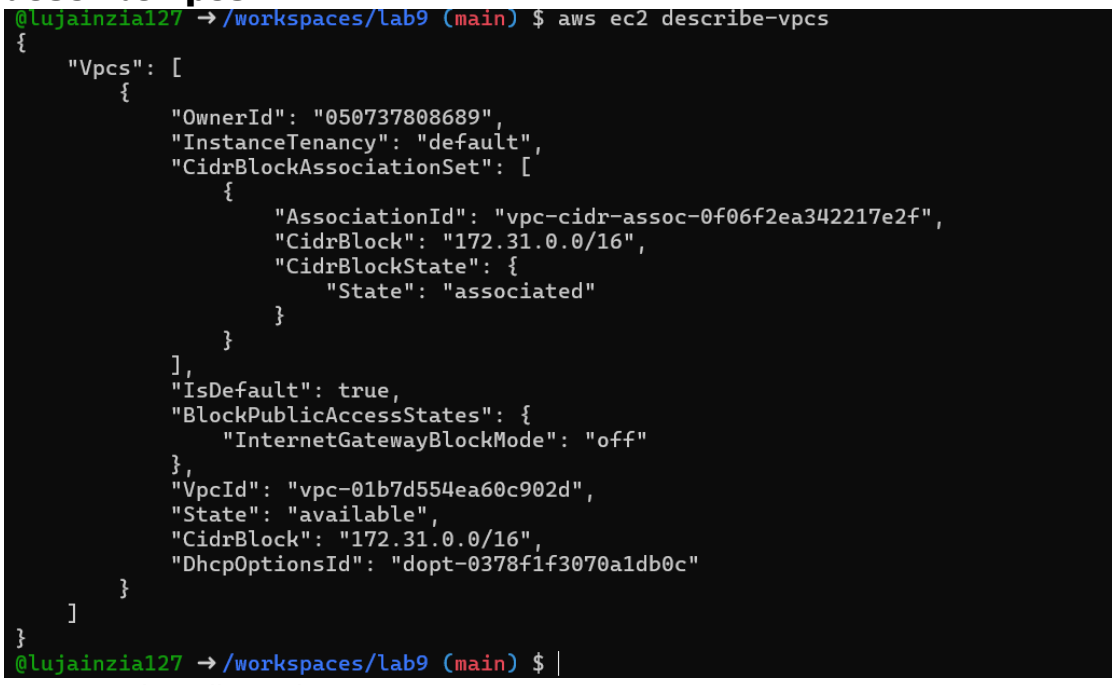
aws ec2 describe-security-groups

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



aws ec2 describe-vpcs

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



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"
                                }
```

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",
```

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
AWSServiceRoleForEC2ScheduledInstances
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

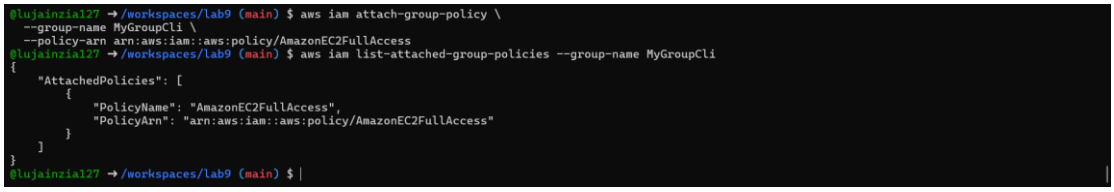
Save screenshot as: task6_policy_list_and_attach.png — ARN query output.

```
@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

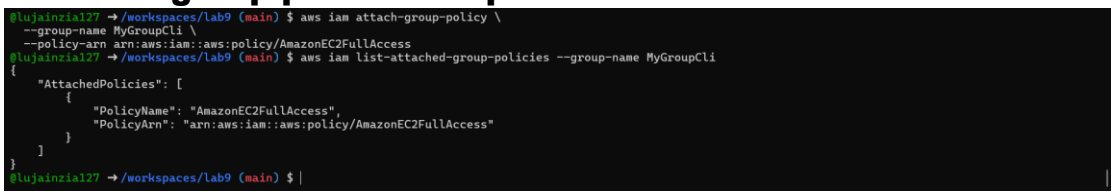
Save screenshot as: task6_policy_list_and_attach.png — attach-group-policy output.



10. List attached policies for the group:

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

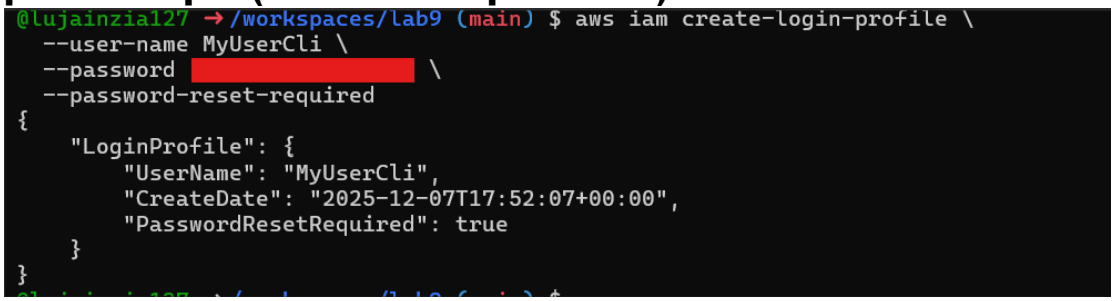
Save screenshot as: task6_policy_list_and_attach.png — list-attached-group-policies output.



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).



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

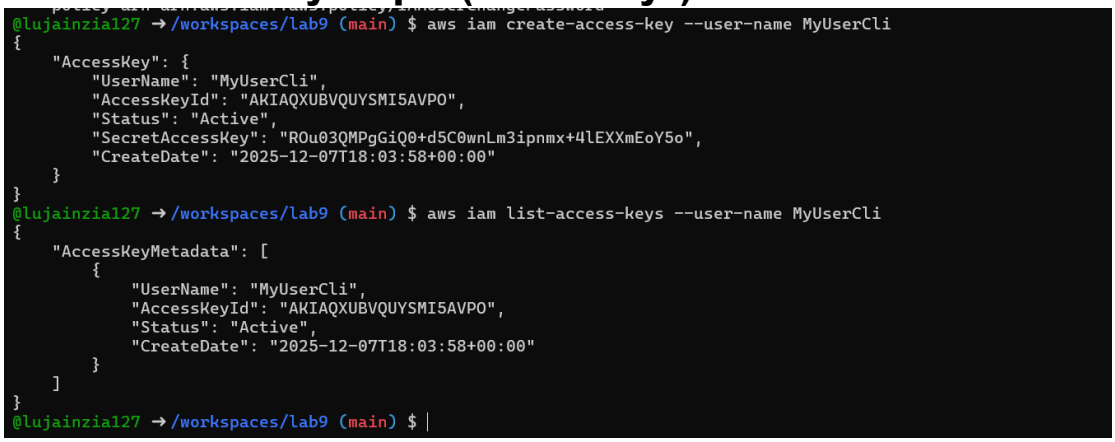
Save screenshot as: task6_create_login_profile_and_signin.png — attach/detach outputs and a screenshot showing the user login (redact password).

```
@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

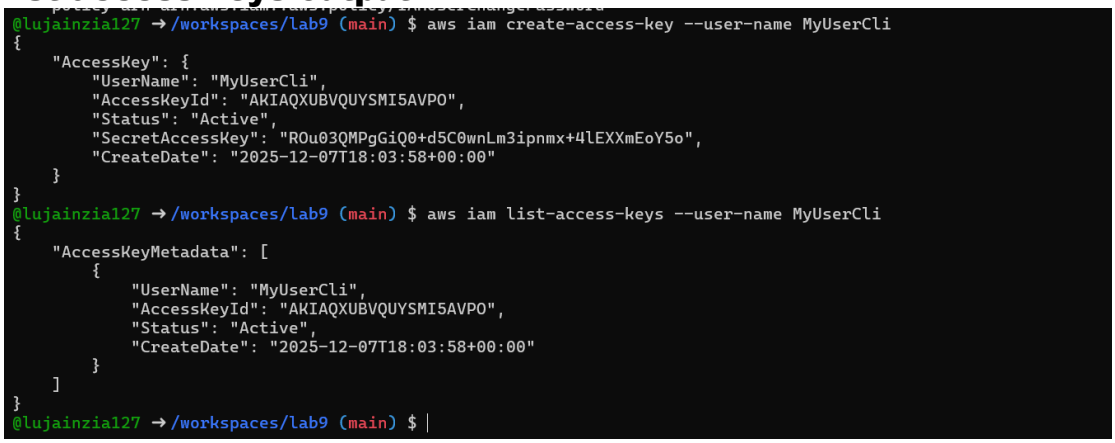
Save screenshot as: task6_create_access_key_output.png — create-access-key output (redact keys).



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.



15. (Don't) Delete access key:

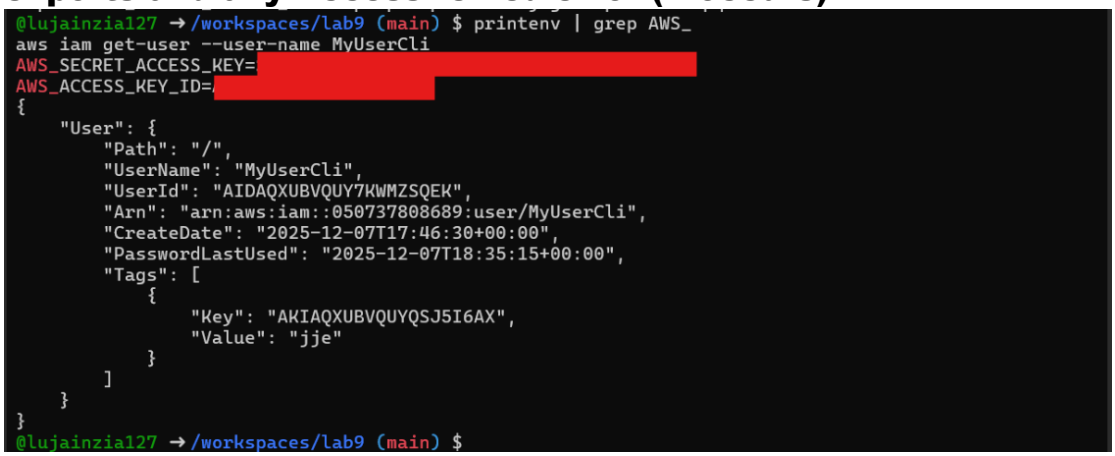
aws iam delete-access-key --user-name MyUserCli --access-key-id <AccessKeyId> # Don't run this command

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

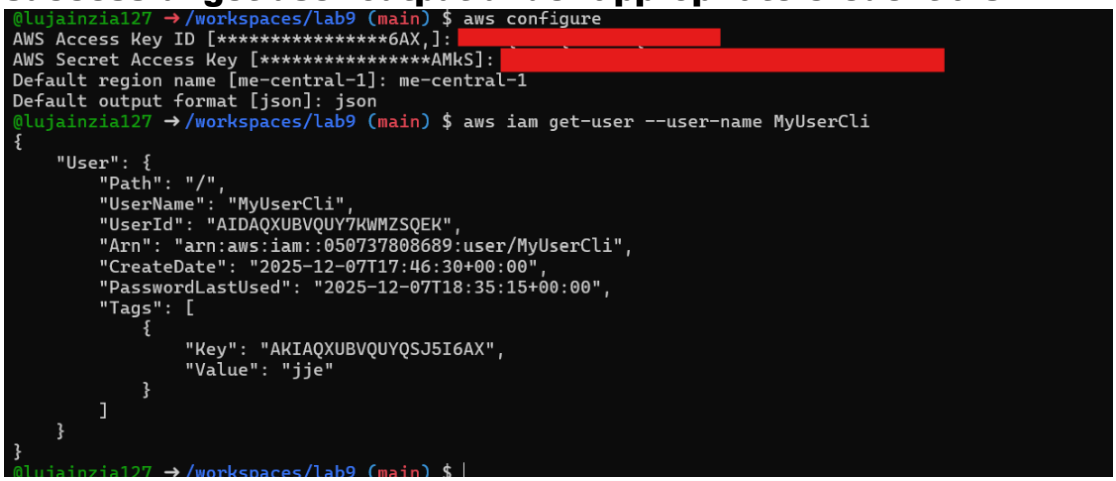
16. Use environment variables to authenticate as that user in the Codespace:

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).



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.

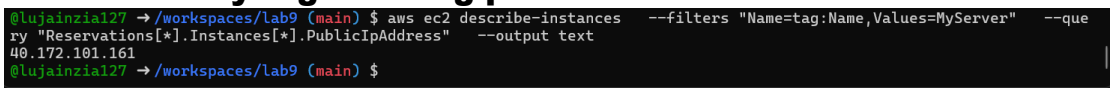


Task 7 — Filters: query with filters to find instances and their attributes

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

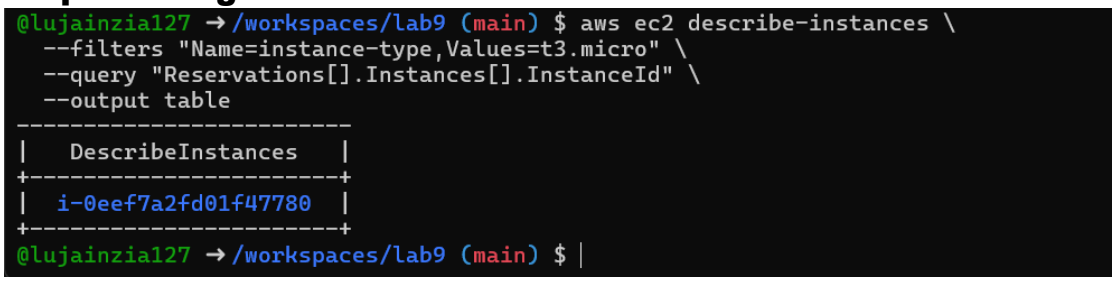
```
aws ec2 describe-instances \
  --filters "Name=tag:Name,Values=MyServer" \
  --query "Reservations[*].Instances[*].PublicIpAddress" \
  --output text
```

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



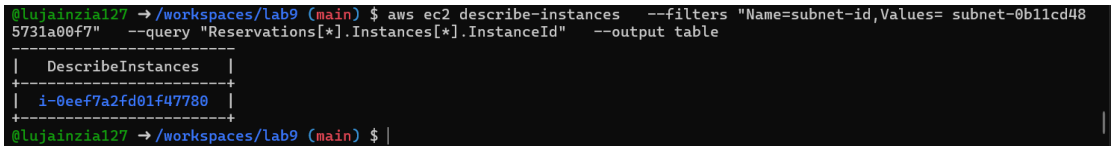

```
aws ec2 describe-instances \
  --filters "Name=instance-type,Values=t3.micro" \
  --query "Reservations[].Instances[].InstanceId" \
  --output table
```

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



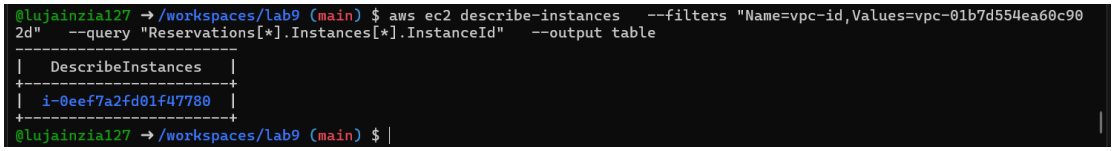

```
aws ec2 describe-instances \
  --filters "Name=subnet-id,Values=subnet-0600df5fa8ce60857" \
  --query "Reservations[*].Instances[*].InstanceId" \
  --output table
```

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




```
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.



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.AvailabilityZone]" \
--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

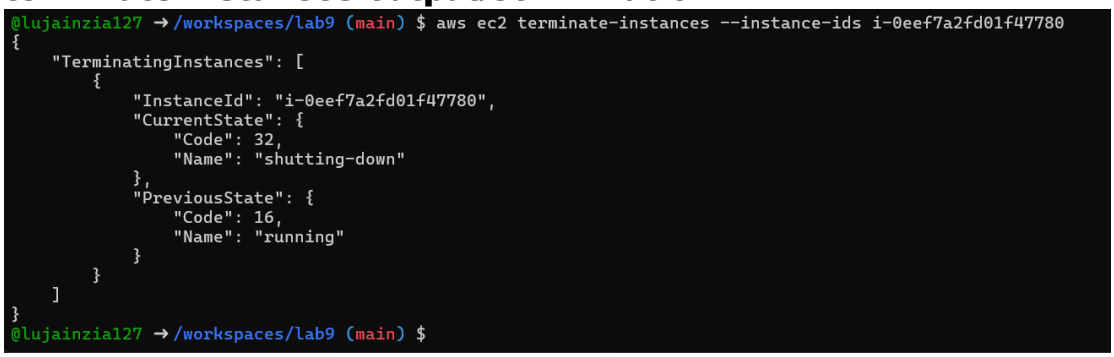
After verification, terminate and delete everything you created in AWS. Capture screenshots for each step.

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

1. Terminate instances:

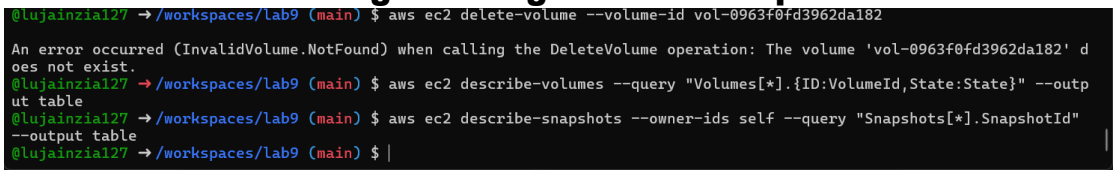
aws ec2 terminate-instances --instance-ids i-EXAMPLE1234567890

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



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

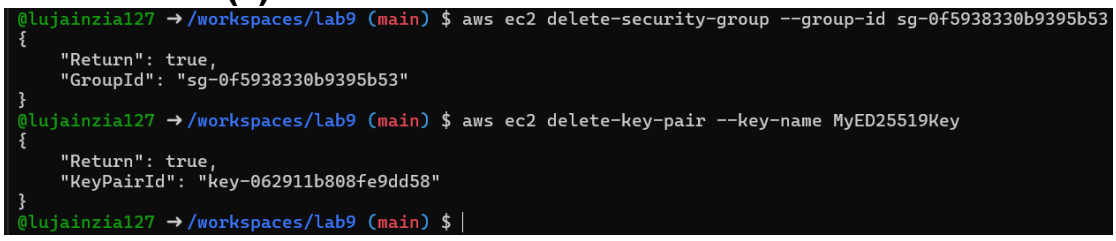
Save screenshot as: cleanup_delete_volumes_snapshots.png — confirmation or listing showing volumes/snapshots deleted.



3. Delete security group and key pair:

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

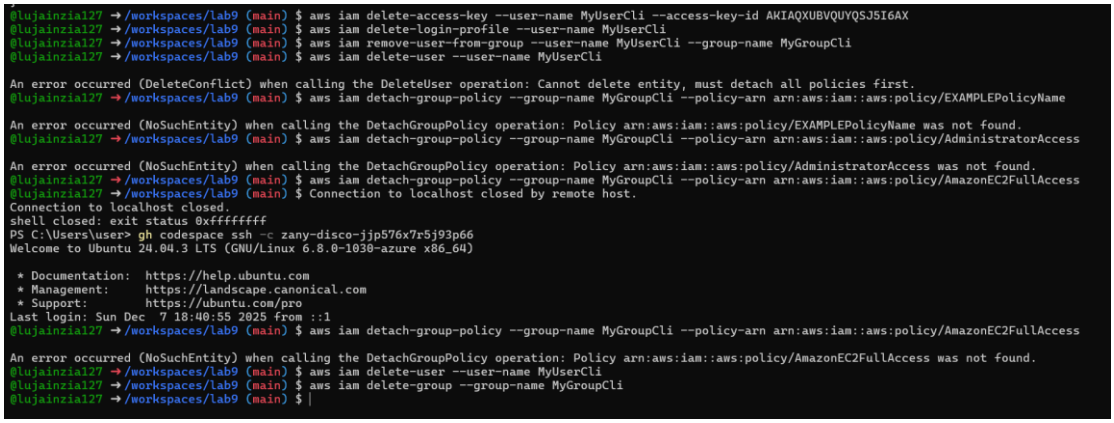
Save screenshot as: cleanup_delete_security_group_and_keypair.png — deletion confirmation(s).



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.



5. Final verification (billing/resource groups):

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