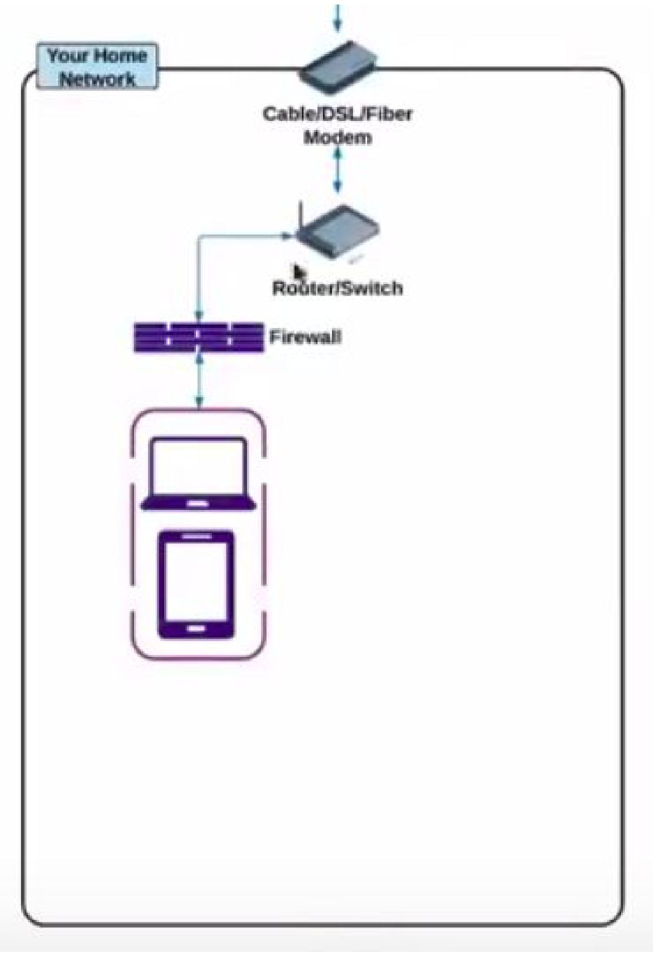
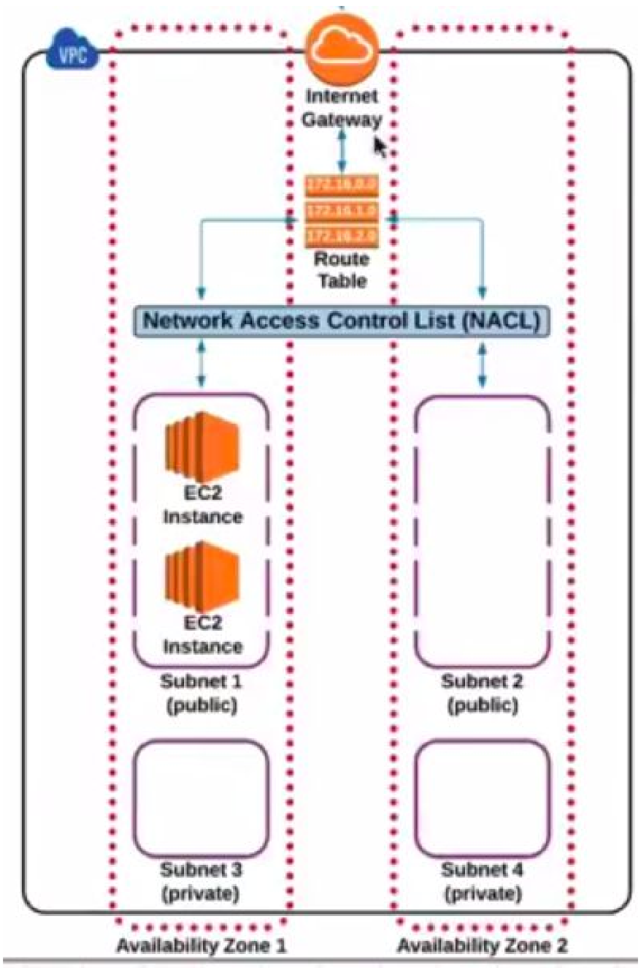
AWS

Tania Batista

AWS Intro

Home Network Example	Amazon AWS Equivalent
1. Internet	1. Internet / VPC
2. Modem: Connected to internet	2. Internet Gateway: Connects you to the URL
3. Router: Allows communication & file-sharing between devices. Requires password for this access. Can work without internet.	3. Route Table: Connects the specific computer
4. Firewall: Security	4. Network Access Control List: Security layer
5. Cell phone/computer	5. Subnet > Instance: public/private
	

Sources:

https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-subnets-commands-example.html

www.LinuxAcademy.com

1. Setting Up the AWS Command Line Interface

1.1 install JQ so you can process json files

```
$ sudo apt-get install python3 python3-pip jq
```

1.2 install AWS command line as a python package

```
$ pip3 install awscli --upgrade --user
```

1.3 check version: should be aws-cli/1.14.32 Python/3.5.2 Linux/4.4.0-112-generic botocore/1.8.36

```
$ aws --version
```

1.4 configure CLI

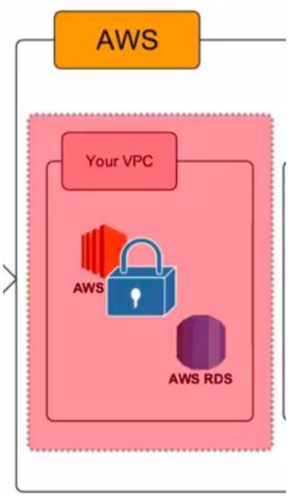
```
$ aws configure
```

```
    # AWS Access Key ID [None]:
    AKIAIRV7VMXGHLSPAXNA

    # AWS Secret Access Key [None]:
    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

    # Default region name [None]:
    eu-west-1
```

2. THE VPC

A Virtual Private Network encompasses everything:

- EC2 (used for webhosting by instances/subnets),
- and RDS (which catalogues the information for various EC2 users.)
- You can add an extra security layer.



2.1 Create the VPC & Configure the CIDR (Classless Inter-Domain Block) to /16

- **CIDR provides routing prefix aggregation. This reduces the number of routes that have to be advertised.**
- **For example, sixteen /24 networks can be added to a larger network as a single /20 routing table entry, if the first 20 bits of their network prefixes match.**

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

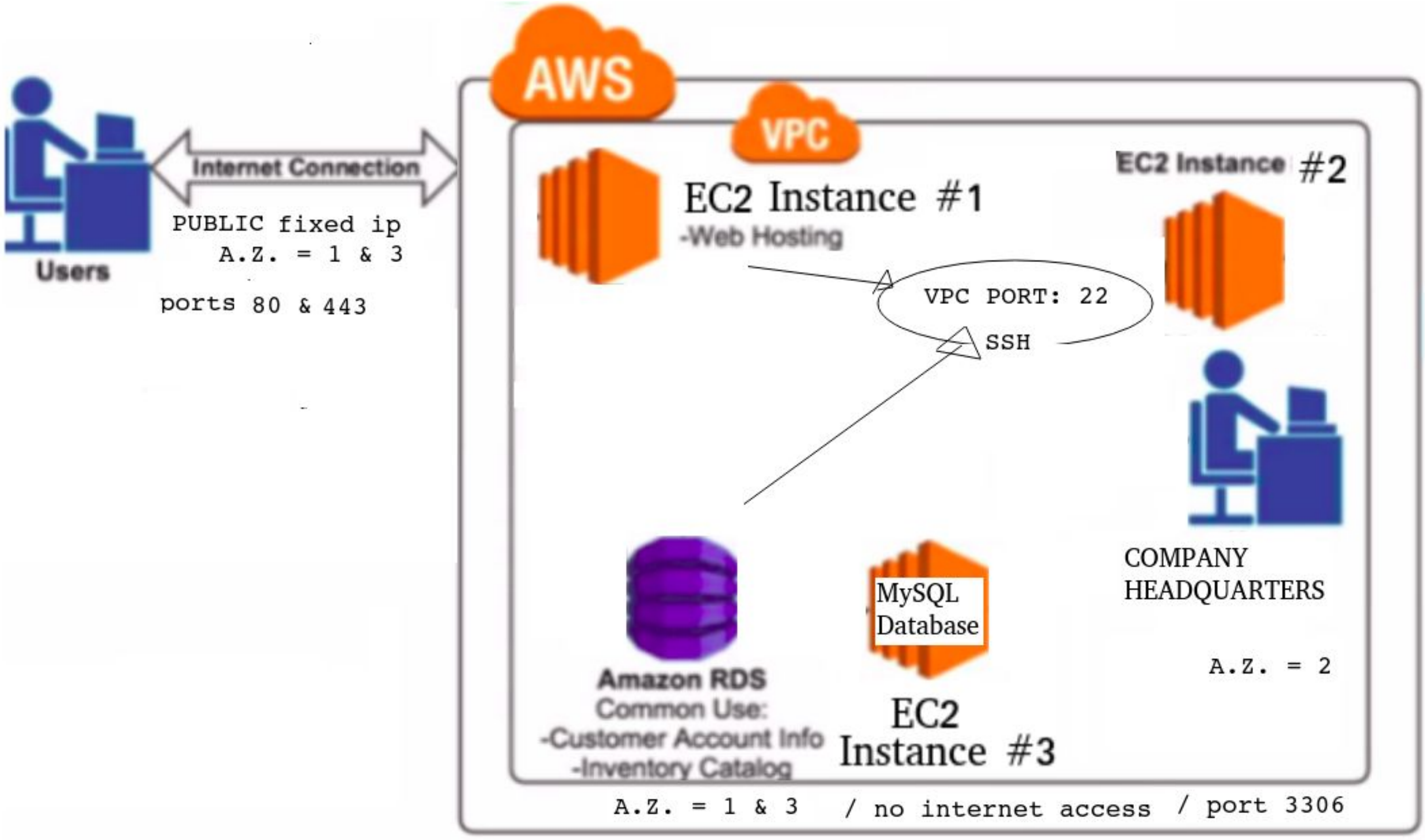
2.2 Check if the VPC has been created. (Leave the ID empty to see all of the available VPC IDs)

```
$ aws ec2 describe-vpcs --vpc-ids
```

> Output: gives the ID for your newly-created VPC.

```
"Vpcs": [
    {
      "CidrBlock": "172.31.1.0/16",
      "DhcpOptionsId": "dopt-7abe0212",
      "State": "available",
      "VpcId": "vpc-b901a7d1",
      "InstanceTenancy": "default",
      "CidrBlockAssociationSet": [
         {
            "AssociationId": "vpc-cidr-assoc-5c5eef34",
            "CidrBlock": "172.31.1.0/16",
            "CidrBlockState": {
               "State": "associated"
            }
         }
      ],
      "IsDefault": true
```

3. SUBNETS



Configuration	Subnet 1 subnet-15cc050f	Subnet 2 subnet-26bb161fc	Subnet 3 subnet-37ee050f
Access	Internet	Internet	No internet access. Can only access other networks inside the VPC.
Availability Zone	Same as Subnet #3	Different from #1 and #3	Same as Subnet #1
Image	$AMI	$AMI	$AMI
Requirements	Instance 1	Instance 2	Instance 3
Installation USE	**Web application** that provides service for users	**Business** admin	MySQL-type Database
Incoming Traffic	From anywhere in the world	OFF	From subnet #1
…. Ports	80 443	-	3306
Inbound/Outbound Traffic to allow the server to connect to other hosts via SSH	to Subnet 2	Company Headquarters	to Subnet 2 for administration purposes
… Ports	22	22	22
Public IP	Fixed	ON (yes) Doesn't have to be fixed	OFF (none) Private?

SUBNET 1

- **VPC ID: vpc-b901a7d1**
- **CIDR association: "172.31.1.0/16"**

```
$ aws ec2 create-subnet \
        --cidr-block 172.31.1.0/16 \
        --vpc-id vpc-b901a7d1
```

```
> Output:
  "Subnet": {
    "AvailabilityZone": "eu-west-2a",
    "AvailableIpAddressCount": 65531,
    "CidrBlock": "172.31.1.0/16",
    "DefaultForAz": false,
    "MapPublicIpOnLaunch": false,
    "State": "pending",
    "SubnetId": "subnet-15cc050f",
    "VpcId": "vpc-b901a7d1",
    "AssignIpv6AddressOnCreation": false,
    "Ipv6CidrBlockAssociationSet": []
  }
```

SUBNET 2

- **VPC ID: vpc-b901a7d1**
- **CIDR association: "172.31.1.0/16"**
- **Availability zone: unique**

```
$ aws ec2 create-subnet \
        --availability-zone eu-west-1a \
        --cidr-block 172.31.2.0/16 \
        --vpc-id vpc-b901a7d1
```

```
> Output:
  "Subnet": {
    "AvailabilityZone": "eu-west-1a",
    "AvailableIpAddressCount": 65531,
    "CidrBlock": "172.31.2.0/16",
    "DefaultForAz": false,
    "MapPublicIpOnLaunch": false,
    "State": "pending",
    "SubnetId": "subnet-26bb161fc",
    "VpcId": "vpc-b901a7d1",
    "AssignIpv6AddressOnCreation": false,
    "Ipv6CidrBlockAssociationSet": []
  }
```

SUBNET 3

- **VPC ID: vpc-b901a7d1**
- **CIDR association: "172.31.1.0/16"**
- **Availability zone: same as 1: eu-west-2a**

```
$ aws ec2 create-subnet \
        --availability-zone eu-west-2a \
        --cidr-block 172.31.3.0/16 \
        --vpc-id vpc-b901a7d1
```

```
> Output:
  "Subnet": {
    "AvailabilityZone": "eu-west-2a",
    "AvailableIpAddressCount": 65531,
    "CidrBlock": "172.31.3.0/16",
    "DefaultForAz": false,
    "MapPublicIpOnLaunch": false,
    "State": "pending",
    "SubnetId": " subnet-37ee050f",
    "VpcId": "vpc-b901a7d1",
    "AssignIpv6AddressOnCreation": false,
    "Ipv6CidrBlockAssociationSet": []
```

4. INTERNET GATEWAY:

4.1 Create ONE internet gateway (igw): this is what allows the servers/instances to access a URL

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

```
> Output
{
   "InternetGateway": {
      "Attachments": [],
      "InternetGatewayId": "igw-bc1d28d5",
      "Tags": []
   }
}
```

4.2 Add the internet gateway ID to the VPC, so we know who is going to use it

```
$ aws ec2 attach-internet-gateway \
        --internet-gateway-id igw-bc1d28d5 \
        --vpc-id vpc-b901a7d1
```

5. ROUTE TABLES

Routes = determine where traffic is directed. One Route per subnet. (?)

5.1 Find the Route Table (rtb). Look for “MAIN” and “TRUE” in the output.
These provide a device’s connection between the NACL and Internet Gateway.

```
$ aws ec2 describe-route-tables --filters "Name=vpc-id, Values=vpc-b901a7d1"

                > Output route table:
                rtb-76a6321e

                > Output internet gateway:
                igw-1775597e
```

5.2 Add the default route for IPV4: the CIDR block 0.0.0.0/0

```
$ aws ec2 create-route \
        --route-table-id rtb-76a6321e \
        --destination-cidr-block 0.0.0.0/0 \
        --gateway-id igw-1775597e
```

5.3 Associate SUBNET1 to the route table: this allows traffic between subnets

```
$ aws ec2 associate-route-table \
        --route-table-id rtb-76a6321e \
        --subnet-id subnet-15cc050f

                > Output:
                "AssociationId": "rtbassoc-3c273954"

        # SUBNET1: Configure to PUBLIC IP  addresses
        $ aws ec2 modify-subnet-attribute
                --subnet-id subnet-15cc050f
                --mappublic-ip-on-launch
```

5.4 Associate SUBNET2 to the route table

```
$ aws ec2 associate-route-table \
        --route-table-id rtb-76a6321e \
        --subnet-id subnet-26bb161fc

        # SUBNET2: Configure to PUBLIC IP  addresses
        $ aws ec2 modify-subnet-attribute
                --subnet-id subnet-26bb161fc
                --mappublic-ip-on-launch
```

5.5 Associate SUBNET3 to the route table: NOT public

```
# Build another route table specifically for PRIVATE ACCESS routes    (?)
$ aws ec2 create-route-table  \
        --vpc-id vpc-b901a7d1
$ aws ec2 associate-route-table \
        --route-table-id rtb-NEW-PRIVATE-ROUTE \
        --subnet-id subnet-37ee050f

        # SUBNET2: Configure to PUBLIC IP  addresses
        $ aws ec2 modify-subnet-attribute \
                        --subnet-id subnet-37ee050f \
                        --no-associate-public-ip-address
```

5.6 Describe the new table associated to the 3 subnetworks

```
$ aws ec2 describe-route-tables \
        --filters "Name=vpc-id, Values=vpc-b901a7d1"
```

6. SUBNET INSTANCES & NETWORK ACCESS CONTROL LISTS

(1 security group per instance)

SUBNET1's INSTANCE 1: Ports 80, 443 & 22 to SUBNET2

1.A SECURITY GROUP: controls secure communication between route tables and EC2 instances

```
$ aws ec2 create-security-group \
        --description "Acceso por SSH" \
        --group-name AccesoSSH \
        --vpc-id vpc-b901a7d1
```

```
> output
   "GroupId": "sg-eacfa581"
```

1.B PORTS: Add instances to security group sg-eacfa581

# Add Port 80 (worldwide)

```
$ aws ec2 authorize-security-group-ingress \
        --group-id sg-eacfa581  \
        --protocol tcp  \
        --port 80  \
        --cidr 0.0.0.0/0
```

# Add Port 443 (worldwide)

```
$ aws ec2 authorize-security-group-ingress \
        --group-id sg-eacfa581  \
        --protocol tcp  \
        --port 443  \
        --cidr 0.0.0.0/0
```

# Add Port 22 (secure SSH - for subnet1): HAS TO BE FIXED PUBLIC IP

```
$ aws ec2 authorize-security-group-ingress \
        --group-id sg-eacfa581  \
        --protocol tcp  \
        --port 22  \
        --cidr 172.31.1.0/16
```

# Confirm the security-group configuration for INSTANCE 1

```
$ aws ec2 describe-security-groups  \
        --group-id sg-eacfa581
```

1.C SSH ACCESS

```
$ aws ec2 create-key-pair \
        --key-name "Instance1Key" \
        --query '{KeyMaterial:KeyMaterial}'  \
        --output text > Instance1Key.pem
```

```
> output:
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAtZynaEbJMeqcWSao8jK7+5AJIOT5iiETwUfxtU0cJxinbi2flWfGiE1Y…...
```

# change the Linux SSH commands & check your key pairs

```
$ chmod 600 Instance1Key.pem
$ ls -lisa Instance1Key.pem
```

```
> Output
   "KeyPairs": [
      {
         "KeyFingerprint": "a5:d0:40:df:2b:2a:b6:44:98:82:29:a3:be:c5:97:90:15:99:2c:98",
         "KeyName": "funprl"
      },
      {
         "KeyFingerprint": "82:02:be:a7:ba:4c:40:1f:73:4d:bb:0b:d7:56:2f:dc:32:f6:67:f9",
         "KeyName": "Instance1Key"
```

1.D Deploy AMI Instances: Refer to subnet1 & its security-group

# Find out the AMI ID for creating the instance

```
$ aws ec2 describe-images \
        --filters "Name=virtualization-type,Values=hvm" "Name=is-public,Values=true" \
        --query 'Images[*].{ID:ImageId, Description:Description, \
                Name:Name, CreationDate:CreationDate}.sort_by(@, &CreationDate)'
        --output text | grep -v testing | grep -v None | grep \
                "ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server"
```

> output AMI Instance1 for Subnet1:
> **ami-fffef49b**

# Describe the EC2 imageID characteristics

```
$ aws ec2 describe-images \
        --image-id ami-fffef49b
```

> output:

```
{
   "Images": [
      {
         "Architecture": "x86_64",
         "CreationDate": "2016-12-31T03:54:05.000Z",
         "ImageId": "ami-fffef49b",
         "ImageLocation": "099720109477/ubuntu/images-testing/ebs-ssd/ubuntu-zesty-daily-amd64-server-20161231",
         "ImageType": "machine",
         "Public": true,
         "KernelId": "aki-8b6369ef",
         "OwnerId": "099720109477",
         "State": "available",
         "BlockDeviceMappings": [
            {
               "DeviceName": "/dev/sda1",
               "Ebs": {
                  "Encrypted": false,
                  "DeleteOnTermination": true,
                  "SnapshotId": "snap-07bef6f6399f6b45d",
                  "VolumeSize": 8,
                  "VolumeType": "gp2"
               }
            },
            {
               "DeviceName": "/dev/sdb",
               "VirtualName": "ephemeral0"
            }
         ],
         "Description": "Canonical, Ubuntu, None, UNSUPPORTED daily amd64 zesty image build on 2016-12-31",
         "Hypervisor": "xen",
         "Name": "ubuntu/images-testing/ebs-ssd/ubuntu-zesty-daily-amd64-server-20161231",
         "RootDeviceName": "/dev/sda1",
         "RootDeviceType": "ebs",
         "VirtualizationType": "paravirtual"
      }
   ]
```

# Run the instance

```
$ aws ec2 run-instances \
        --image-id ami-fffef49b \
        --count 1 \
        --instance-type t2.small \
        --key-name "Instance1Key" \
        --security-group-ids sg-eacfa581 \
        --subnet-id subnet-15cc050f \
        --associate-public-ip-address \
        --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=PruebaCreacion}]'
```

SUBNET2’s INSTANCE 2: none, 22 SSH & public IP

2.A SECURITY GROUP

```
$ aws ec2 create-security-group \
        --description "Acceso por SSH 2" \
        --group-name AccesoSSH2 \
        --vpc-id vpc-b901a7d1
                output>
                  "GroupId": "sg-subnet2groupID"
```

2.B PORTS: Add instances to security group sg-subnet2groupID

```
# Add Port 22 (secure SSH - for subnet2)
$ aws ec2 authorize-security-group-ingress \
        --group-id sg-subnet2groupID  \
        --protocol tcp  \
        --port 22  \
        --cidr 172.31.2.0/16 #This has to be the company’s IP address

# Confirm the security-group configuration for INSTANCE 2
$ aws ec2 describe-security-groups  \
        --group-id sg-subnet2groupID
```

2.C SSH ACCESS

```
$ aws ec2 create-key-pair \
        --key-name "Instance1Key" \
        --query '{KeyMaterial:KeyMaterial}'  \
        --output text > Instance2Key.pem

# change the Linux SSH commands
$ chmod 600 Instance2Key.pem
$ ls -lisa Instance2Key.pem
```

2.D Deploy AMI Instances

```
$ aws ec2 run-instances \
        --image-id ami-fffef49b \    #Image ID will be the same for all 3 instances
        --count 1 \
        --instance-type t2.small \
        --key-name "Instance2Key" \
        --security-group-ids sg-subnet2groupID \
        --subnet-id subnet-SUBNET2-ID \
        --instance-initiated-shutdown-behavior ForceStop
        --associate-public-ip-address \
        --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=PruebaCreacion}]'
```

SUBNET3’s INSTANCE 3: 3306, 22 to subnet2, private IP

3.A SECURITY GROUP

```
$ aws ec2 create-security-group \
        --description "Acceso por SSH 3" \
        --group-name AccesoSSH3 \
        --vpc-id vpc-b901a7d1

                output>
                  "GroupId": "sg-subnet3groupID"
```

3.B PORTS: Add instances to security group sg-subnet3groupID

# Add Port 22: (secure SSH - for subnet3 from subnet1)

```
$ aws ec2 authorize-security-group-ingress \
        --group-id sg-subnet3groupID  \
        --protocol tcp  \
        --port 22  \
        --cidr 172.31.3.0/16
```

# Add Port 3306 (for business database, subnet3)

```
$ aws ec2 authorize-security-group-ingress \
        --group-id sg-subnet3groupID  \
        --protocol tcp  \
        --port 3306  \
        --cidr 172.31.1.0/16
```

# Confirm the security-group configuration for INSTANCE 3

```
$ aws ec2 describe-security-groups  \
        --group-id sg-subnet3groupID
```

3.C SSH ACCESS

```
$ aws ec2 create-key-pair \
        --key-name "Instance3Key" \
        --query '{KeyMaterial:KeyMaterial}'  \
        --output text > Instance3Key.pem
```

# change the Linux SSH commands

```
$ chmod 600 Instance3Key.pem
```

# see all your key pairs

```
$ ls -lisa Instance3Key.pem
```

3.D Deploy AMI Instances:

```
$ aws ec2 run-instances \
        --image-id ami-fffef49b \
        --count 1 \
        --instance-type t2.small \
        --key-name "Instance3Key" \
        --security-group-ids sg-subnet3groupID \
        --subnet-id subnet3ID \
        --instance-initiated-shutdown-behavior terminate \
        --no-associate-public-ip-address \
        --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=PruebaCreacion}]'
```