EC2 CLI

Dr Peadar Grant

October 10, 2022

EC2 and all other AWS services can be scripted through the Command-Line Interface. Here we will look at the setup of a VPC, Security Group and EC2 instance.

1 Powershell usage

1.1 Working directory

The working directory (folder) is shown at the PowerShell prompt. On Windows, directories exist within drives.

1.2 Changing directory

```
# change into foldername folder
cd foldername

# go back up to the parent directory
cd ..

# go back to your home directory (C:\Users\yourname)
cd ~

# go to a directory relative to your home directory
# e.g. to go to the Desktop folder
cd ~/Desktop

# change to a different drive (e.g. O:)
O:
```

1.3 Command history

You can use the up cursor key to view previously entered commands and re-run them after editing them.

1.4 AWS CLI

The AWS CLI consists of the `aws` command and its subcommands grouped by service. Almost all of the commands we'll use are in the `aws ec2` category. To get help:

```
# help on the AWS command
aws help

# help on the commands available in ec2 category
aws ec2 help

# help on an individual command
aws ec2 describe-instances help
```

2 Creating a VPC

Before creating any resources on AWS, you should draw out your VPC as best you can on paper. When creating VPCs and other resources, try to be systematic with your naming!

We will work through an example of a VPC (named `LAB_VPC`) with the CIDR block 10.0.0.0/16 with the 10.0.1.0/24 address range assigned to a single subnet `LAB_1_SN`. This VPC will be setup so that it connects to the public internet.

- VPC 10.0.0.0/16 named `LAB_VPC`
- Subnet 10.0.1.0/24 `LAB_1_SN`
  - Auto-assign public IP is turned on. This means that EC2 instances launched in this subnet will also receive a public IP that is transparently routed to their private IP by AWS using NAT.
- Internet gateway `LAB_GATEWAY` attached to VPC
- Route table (named `LAB_RT`) with routes for:
  - Local traffic 10.0.0.0/16 sent locally
  - All traffic 0.0.0.0/0 routed via `LAB_GATEWAY`

These notes show how to create a VPC using the CLI. The first thing to remember is to use the `help` subcommand of `aws` liberally. Some commands have required arguments (things that must specify) and will tell you if you've omitted them. To see what effect the commands are having you can follow along at the same time in the console and see the effect of your changes.

2.1 VPC creation

To create a VPC using the CIDR block 10.0.0.0/16 we can say:

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

This will return a JSON-formatted response similar to:

```
{
    "Vpc": {
```

```
        "CidrBlock": "10.0.0.0/16",
        "DhcpOptionsId": "dopt-0e49fce64f2659c15",
        "State": "pending",
        "VpcId": "vpc-08ae22f0a2cc414a0",
        "OwnerId": "593472368051",
        "InstanceTenancy": "default",
        "Ipv6CidrBlockAssociationSet": [],
        "CidrBlockAssociationSet": [
            {
                "AssociationId": "vpc-cidr-assoc-058fd321447f841b1",
                "CidrBlock": "10.0.0.0/16",
                "CidrBlockState": {
                    "State": "associated"
                }
            }
        ],
        "IsDefault": false
    }
}
```


A few things to note about the response:

- Most AWS commands will return output formatted as JSON.
- The VPC ID is assigned by AWS, not us.

If you look at the console following creation of your VPC you'll see it listed if you hit refresh.

2.1.1 Using shell variables

We are going to need the VPC ID in a few places. You should copy and paste it to assign it to a variable. Powershell:

```
$VpcId="vpc-08ae22f0a2cc414a0"
```

Bash version:

```
VpcId="vpc-08ae22f0a2cc414a0"
```

2.1.2 Naming

To name our VPC as `LAB_VPC` we issue the command:

```
aws ec2 create-tags --resources $VpcId --tags Key=Name,Value=LAB_VPC
```

Hitting refresh in the console should show the name now appearing.

2.2 Subnet creation

Creating a subnet requires two pieces of information: the VPC ID and the CIDR block (which must be a subset of the VPC CIDR block). For a simple VPC with one subnet, the subnet's CIDR block can be the same as the VPC's CIDR block. But normally it's good to have it smaller (giving us the option later

to add another subnet). Here we will create a single subnet with the CIDR block 10.0.0.0/24 inside the VPC (CIDR block 10.0.0.0/16).

```
aws ec2 create-subnet --vpc-id $VpcId --cidr-block 10.0.1.0/24
```

Just like the VPC, the subnet creation will return:


```
{
    "Subnet": {
        "AvailabilityZone": "eu-west-1b",
        "AvailabilityZoneId": "euw1-az1",
        "AvailableIpAddressCount": 251,
        "CidrBlock": "10.0.1.0/24",
        "DefaultForAz": false,
        "MapPublicIpOnLaunch": false,
        "State": "available",
        "SubnetId": "subnet-085a8c631d6a9b174",
        "VpcId": "vpc-08ae22f0a2cc414a0",
        "OwnerId": "593472368051",
        "AssignIpv6AddressOnCreation": false,
        "Ipv6CidrBlockAssociationSet": [],
        "SubnetArn": "arn:aws:ec2:eu-west-1:593472368051:subnet/subnet-085a8c631d6a9b174"
    }
}
```


A few interesting things to note about the response:

- AWS has assigned a `SubnetId` for us.
- Each subnet is actually linked to a physical availability zone, here `eu-west-1b` (within the `eu-west-1` region).
- Note also how `eu-west-1b` is actually known as `euw1-az1`. This is because the `a, b, c` availability zones are actually rotated between different accounts to balance load. Your `eu-west-1b` might be another person's `eu-west-1c`.

2.2.1 Subnet ID

```
$SubnetId="subnet-085a8c631d6a9b174"
```

2.2.2 Naming subnet

We can name the subnet the same way as the VPC:

```
aws ec2 create-tags --resources $SubnetId --tags Key=Name,Value=LAB_SUBNET_1
```

2.2.3 Public IP

```
aws ec2 modify-subnet-attribute `
--subnet-id $SubnetId `
--map-public-ip-on-launch
```

2.3 Internet gateway

An internet gateway needs to be created and then attached to the correct VPC. To create an internet gateway we type:

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

with JSON response like:


```
{
    "InternetGateway": {
        "Attachments": [],
        "InternetGatewayId": "igw-065bc8d541f711ce1",
        "OwnerId": "593472368051",
        "Tags": []
    }
}
```


Save the gateway ID

```
$GatewayId="igw-065bc8d541f711ce1"
```

We can name the internet gateway the same as before:

```
aws ec2 create-tags --resources $GatewayId  --tags Key=Name,Value=LAB_GATEWAY
```

Finally we can attach the internet gateway to a VPC.

```
aws ec2 attach-internet-gateway --internet-gateway-id $GatewayId --vpc-id $VpcId
```

This won't return any output unless there's an error.

2.4 Route tables

In the CLI, we can get a list of all route tables using:

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

which will show us all the route tables. To ensure we see only the route table(s) for our VPC we should add a filter to the command (as described in the help):

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

This will give us the route table in JSON:


```
{
    "RouteTables": [
        {
            "Associations": [
                {
                    "Main": true,
                    "RouteTableAssociationId": "rtbassoc-0eb5e7a4ea9666bd3",
                    "RouteTableId": "rtb-0b7ce39addb6017be",
                    "AssociationState": {
                        "State": "associated"
                    }
```

```
                }
            ],
            "PropagatingVgws": [],
            "RouteTableId": "rtb-0b7ce39addb6017be",
            "Routes": [
                {
                    "DestinationCidrBlock": "10.0.0.0/16",
                    "GatewayId": "local",
                    "Origin": "CreateRouteTable",
                    "State": "active"
                }
            ],
            "Tags": [],
            "VpcId": "vpc-08ae22f0a2cc414a0",
            "OwnerId": "593472368051"
        }
    ]
}
```


2.4.1 Route table ID

We grab the route table ID:

```
$RouteTableId="rtb-0b7ce39addb6017be"
```

2.4.2 Naming the route table

First let's name the main route table:

```
aws ec2 create-tags --resources $RouteTableId --tags Key=Name,Value=LAB_RTB
```

2.4.3 Adding route

Looking at the JSON output, the Routes list contains one entry. This routes all traffic to 10.0.0.0/16 addresses locally. We need to add a route for traffic elsewhere (0.0.0.0/0) to go through the internet gateway.

```
aws ec2 create-route `
--route-table-id $RouteTableId `
--destination-cidr-block 0.0.0.0/0 `
--gateway-id $GatewayId
```

Re-running the describe route table command should now show two routes.

3 Security groups

Security groups are essentially a firewall controlling what traffic is allowed into or out of each instance. For a good description of security groups type:

```
aws ec2 create-security-group help
```

Each instance may have one or more security groups attached.

Every instance created can have a default security group attached but this leads to a few problems:

- Hard to get an overview of allowed/denied traffic to instances (security risk)
- Hard to reconfigure allowed/denied traffic to a number of instances (time consuming, nuisance)

Instead we will create a security group and attach it as needed to instances in our VPC. We will create a LAB_SG security group to allow in SSH access.

3.1 Creating security group

Security groups are associated with a VPC, so your VPC must be set up before creating the security group.

```
aws ec2 create-security-group --group-name 'LAB_SG' --description 'LAB_SG' --vpc-id $VpcId
```

Successful output will look like:

```
{
    "GroupId": "sg-0d26fc12d5641121c"
}
```

```
$GroupId='sg-0d26fc12d5641121c'
```

3.2 Adding ingress rules

We now add an ingress rule to our security group to permit inbound TCP traffic on Port 22 (SSH) using the command:

```
aws ec2 authorize-security-group-ingress `
--group-id $GroupId `
--protocol tcp `
--port 22 --cidr 0.0.0.0/0
```

Note we used 0.0.0.0/0 as the source, meaning from anywhere. We can lock this down to specific IP addresses or IP ranges (e.g. your ISP). This is an exercise for another time!

3.3 Egress rules

By default, security groups allow egress of all traffic from instances, so this doesn't need to be set up.

4 Instance setup

We will setup an EC2 instance as follows:

- AMI: Amazon Linux
- Configuration: `t2.micro`
- VPC: LAB_VPC

- Subnet: LAB_1_SUBNET
- Security group: LAB_SG
- Key: MAIN_KEY

4.1 Available image names

Image IDs are region and account-dependent. They also get updated as Amazon update the images. We can use a tool called Systems Manager to look up available AMIs:

```
# print out list of Linux AMIs
aws ssm get-parameters-by-path `
--path /aws/service/ami-amazon-linux-latest `
--query "Parameters[].Name"
```

The “standard” linux image we will use is amzn2-ami-hvm-x86_64-gp2.

4.2 Launching by name

We can use Systems Manager to launch an instance using the following syntax. Instead of giving an AMI directly we use resolve:ssm: to tell AWS to look this value up in SSM.

```
aws ec2 run-instances `
--subnet-id $SubnetId `
--image-id resolve:ssm:/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2 `
--instance-type t2.micro `
--key-name MAIN_KEY `
--security-group-ids $GroupId
```

4.3 Instance information

Launching an instance will give a very long JSON in the format:

```
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-0bb3fad3c0286ebd5",
            "InstanceId": "i-062914b3061527245",
            "...": "...",
            "ReservationId": "r-0536a89f39c4c6324"
}
```

We grab the instance Id:

```
$InstanceId="i-062914b3061527245"
```

5 Connecting to instance

5.1 Finding instance public IP

Once our instance is Running, it will have a public IPv4 that AWS transparently routes through to its private IP using NAT.

To find this out we can look it up in the console, or use the following commands:

```
aws ec2 describe-instances --instance-id $InstanceId
```

The public IP is listed with the network interface:


```
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "...": "...",
                    "PublicIpAddress": "54.170.85.144",
                    "State": {
                        "Code": 16,
                        "Name": "running"
                    }
                }
            ],
            "OwnerId": "593472368051",
            "ReservationId": "r-04094affdd1662bba"
        }
    ]
}
```


We copy the public IP:

```
$PublicIp="54.170.85.144"
```

5.2 Connecting to instance over SSH

```
ssh ec2-user@$PublicIp
```

The first time you connect to an instance you'll get a warning:

```
The authenticity of host '54.78.220.233 (54.78.220.233)' can't be established.
ECDSA key fingerprint is SHA256:8omkD5RLibZNgJJ/B7MAnL7IbEcrmCmIWFdQXbjJf60.
Are you sure you want to continue connecting (yes/no)?
```

Just type `yes` here. Your local SSH client is just confirming it hasn't seen this machine before. If a different key fingerprint shows for the same IP you'll get a warning, which means a machine has been changed for another.

If you see something like the following then you're connected:

```
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
2 package(s) needed for security, out of 13 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-0-1-80 ~]$
```

6 EC2 instance termination

Instances can be terminated when no longer needed using the command:

```
aws ec2 terminate-instances --instance-ids $InstanceId
```