EC2 Linux

Dr Peadar Grant

September 16, 2021

Infrastructural services consist of compute, low-level storage and the networking required to connect them. Compute is provided on AWS using a service called EC2. Virtual Machines on EC2 are called Instances. Instances are grouped into a virtual network called a VPC. VPCs are important - without them we wouldn't be able to connect to our virtual machines (as AWS has no emulated console video / keyboard).

1 EC2

The backbone IaaS compute service offered by AWS is Elastic Compute 2. This allows you to create Virtual Machines on the AWS platform. Many of their other services rely on EC2. Other major cloud platforms (Azure, IBM, Google) all offer a basic virtual-machine compute service. Virtual Machines created on AWS are known as EC2 instances (or just instances).

1.1 Compute instances

To launch an instance, we must decide:

Instance Type which sets the hardware configuration. This includes CPU family, number of cores, clock speed, RAM.

Operating System template as available as an Amazon Machine Image (or AMI) that will be used to clone the image from.

Storage: The instance's virtual hard disk.

VPC and Subnet that the instance will be launched into.

Security group which defines what traffic is allowed in/out of individual EC2 instances. Try to use shared/template ones rather than one per instance.

1.2 Amazon Machine Image (AMI)

- On physical servers (and when dealing with on-site virtualisation systems) we often install an OS using an installer disk and work through the steps of the installer.
- Cloud compute instances are normally created by cloning an image and running some minor post-installation tasks.
- AWS uses the Amazon Machine Image (AMI) to clone a compute instance's virtual hard disk from.

1.3 Amazon Linux

There are a few different Linux distributions on AWS as AMIs, (as well as Windows). One possible Linux flavour is Amazon Linux which is stripped-down for use as a cloud server and is maintained by AWS. Some notes about it:

- By default, a user named `ec2-user` is provisioned. You can of course set up other users as you wish as on any OS.
- It uses the RedHat `yum` package manager rather than `apt` as in Ubuntu / Debian. *(You should be comfortable at this stage working with similar non-identical tools to broaden your experience and horizon!)*

1.4 Remote Access

Unlike some virtualisation solutions (e.g. Xen server, Hyper-V), AWS provide no emulated Keyboard / Video / Mouse (KVM). (You can however take a screenshot.) This means that all operations must be carried out using in-band remote protocols like Secure Shell (SSH) or Remote Desktop Protocol.

1.5 Key pairs

Some AMIs have a default username/password, but most including Amazon Linux use a key-pair:

1. You generate a public / private key pair. (Using PuTTY, ssh-keygen or the AWS console-based generator.)
2. You use the public key when creating EC2 instances.
3. Log in using PuTTY (windows) / ssh (mac/linux) using your private key. Username for Amazon Linux is `ec2-user`.

1.6 Upload key pair to AWS

Uploading a key pair is normally only done once. Make sure you are in the `.ssh` directory in Powershell/Bash. Then issue the command:

```
aws ec2 import-key-pair --key-name LAB_KEY --public-key-material fileb://id_isa.pub
```

The `fileb` is not a typo! If the command is successful, the output should look like:

```
{
    "KeyFingerprint": "93:17:be:5c:ed:df:c1:45:5a:93:82:fa:db:76:aa:53",
    "KeyName": "LAB_KEY",
    "KeyPairId": "key-01400961a75915709"
}
```

(If you run it a second time it should fail as the key exists.) You can see your key pair(s) stored in AWS using the command:

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

This will give JSON output similar to:

```
{
    "KeyPairs": [
        {
            "KeyPairId": "key-01400961a75915709",
            "KeyFingerprint": "93:17:be:5c:ed:df:c1:45:5a:93:82:fa:db:76:aa:53",
            "KeyName": "LAB_KEY",
            "Tags": []
        }
    ]
}
```

2 Virtual private clouds

IaaS components (like EC2) are launched in Virtual Private Clouds (VPCs).

2.1 Key components

VPC (Virtual Private Cloud): is a virtual software-defined network to which your compute instances connect. Essentially it's a software-defined multi-site data entre environment. Each VPC ...

- ... is associated with a single specific region.
- ... is owned by a single AWS account.
- ... has a CIDR block of addresses, such as 10.0.0.0/16.

Subnets within the VPC. Each subnet:

- is associated with a single specific Availability Zone.
- has a single CIDR-block of addresses, such as 10.0.1.0/24.

EC2 (Compute instances): virtual machines provisioned on-demand in the cloud. Each EC2 instance must be attached to a particular subnet (and therefore belongs to a particular VPC).

Network Access Control Lists (ACLs) define what traffic is allowed flow between subnets of a single VPC.

Route table(s) specify where traffic for specific IP ranges should be routed.

Internet Gateway connects VPC to the internet. (Target for 0.0.0.0/0 traffic in route table.)

2.2 Names and IDs

Names are important to identify components, particularly when we use command-line interfaces. We already know that AWS use text code names for the different regions and AZs. As we create and use resources it's important to note the distinction between names we assign and Ids that are assigned by AWS:

Names are assigned by you when creating components like VPCs, subnets, internet gateways, ec2 instances.

Ids are assigned by AWS when you create the same resources.

You can name things any way you like, but I tend to suggest you follow a pattern and avoid spaces! Table 1 shows suggested suffix patterns I use (with the corresponding AWS prefixes for the ids)

3 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!

Component type	Suggested name suffix	AWS name prefix
VPC	_VPC	vpc-
Subnet	_SN	sn-
Internet gateway	_IGW	igw-
Route table	_RTB	rtb-

Table 1: Naming suggestions and AWS Id prefixes

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_IGW 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_IGW

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.

3.1 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-71d01c16",
        "State": "pending",
        "VpcId": "vpc-0afdc142d97b1eaaa",
        "OwnerId": "637116340434",
        "InstanceTenancy": "default",
        "Ipv6CidrBlockAssociationSet": [],
        "CidrBlockAssociationSet": [
            {
                "AssociationId": "vpc-cidr-assoc-0d6ab9228eec46921",
                "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.

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

```
aws ec2 create-tags --resources vpc-0afdc142d97b1eaaa --tags Key=Name,Value=LAB_VPC
```

Obviously you'll have to change the VPC ID in the above command to your own. Hitting refresh in the console should show the name now appearing.

3.2 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 vpc-0afdc142d97b1eaaa --cidr-block 10.0.0.0/24
```

Just like the VPC, the subnet creation will return:

```
{
    "Subnet": {
        "AvailabilityZone": "eu-west-1b",
        "AvailabilityZoneId": "euw1-az3",
        "AvailableIpAddressCount": 65531,
        "CidrBlock": "10.0.0.0/24",
        "DefaultForAz": false,
        "MapPublicIpOnLaunch": false,
        "State": "available",
        "SubnetId": "subnet-0530a2180c3f71e62",
        "VpcId": "vpc-0afdc142d97b1eaaa",
        "OwnerId": "637116340434",
        "AssignIpv6AddressOnCreation": false,
        "Ipv6CidrBlockAssociationSet": [],
        "SubnetArn": "arn:aws:ec2:eu-west-1:637116340434:subnet/subnet-0530a2180c3f71e62"
    }
}
```

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-az3`. 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`.

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

```
aws ec2 create-tags --resources subnet-0530a2180c3f71e62 --tags Key=Name,Value=LAB_SN
```

3.3 Internet gateway

When a VPC (and its subnets) are created, it's actually isolated from the internet. There are some circumstances we'll see later on where we actually don't want a VPC to connect to the internet, but generally we do. This is through a resource called an 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-01fe2befea2cd8a27",
        "OwnerId": "637116340434",
        "Tags": []
    }
}
```

We can name the internet gateway the same as before:

```
aws ec2 create-tags --resources igw-01fe2befea2cd8a27 --tags Key=Name,Value='LAB_IGW'
```

Finally we can attach the internet gateway to a VPC. The `aws ec2 attach-internet-gateway` command needs two IDs: the internet gateway and the VPC.

```
aws ec2 attach-internet-gateway --internet-gateway-id igw-01fe2befea2cd8a27  --vpc-id vpc-0afdc
```

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

3.4 Route tables

Route tables control how traffic is directed among the different subnets and into/out of the VPC. They're not difficult to understand, but a misconfigured route table will generally cause problems.

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=vpc-0afdc142d97b1eaaa
```

This will give us the route table in JSON:

```
{
    "RouteTables": [
        {
            "Associations": [
                {
                    "Main": true,
                    "RouteTableAssociationId": "rtbassoc-08fd06faf732ee3ca",
                    "RouteTableId": "rtb-0a895efdab0ad7591",
                    "AssociationState": {
                        "State": "associated"
                    }
                }
            ],
            "PropagatingVgws": [],
            "RouteTableId": "rtb-0a895efdab0ad7591",
            "Routes": [
                {
                    "DestinationCidrBlock": "10.0.0.0/16",
                    "GatewayId": "local",
                    "Origin": "CreateRouteTable",
                    "State": "active"
                }
            ],
            "Tags": [],
            "VpcId": "vpc-0afdc142d97b1eaaa",
            "OwnerId": "637116340434"
        }
    ]
}
```

First let's name the main route table:

```
aws ec2 create-tags --resources rtb-0a895efdab0ad7591 --tags Key=Name,Value=LAB_RTB
```

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 rtb-0a895efdab0ad7591 --destination-cidr-block 0.0.0.0/0
```

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

4 Parsing JSON

By now you can see that most AWS commands by default return JSON responses that contain a lot of information. Other formats are possible.

If you wanted to script the above operation you would need to be able to parse the JSON to extract things like the VPC, subnet, internet gateway and route table IDs. (You can also use other utilities with the text format option.) AWS provide some guidance in the CLI usage output page.

4.1 Powershell

In Powershell you can use the `ConvertFrom-Json` cmdlet to convert the JSON output from the AWS command to powershell objects. See the Powershell documentation for full details on its usage. We could thus convert the output from `aws ec2 describe-vpcs` to a variable named `VPCs`:

```
# VPCs as a PowerShell object
$VPCs = aws ec2 describe-vpcs | ConvertFrom-Json

# The VPC ID of the first VPC would then be found by:
$VPCs.Vpcs[0].VpcId
```

If you're wondering why the `Vpcs` object is there take a look back at the JSON returned for the `aws ec2 describe-vp`
command. It may be more natural to de-reference the Vpcs root object immediately when converting:

```
# remove the Vpcs root object by de-referencing on conversion:
$vpcs = (aws ec2 describe-vpcs | ConvertFrom-Json).Vpcs

# The VPC ID of the first VPC would then be found by:
$VPCs[0].VpcId
# (simpler usage than leaving the Vpcs root object there)
```

We will use this pattern for most AWS command output to make the powershell commands later on easier to read.

4.2 Bash

The easiest way to parse JSON in Bash is to use `jq`. If you're on Linux/UNIX you can install `jq` using the package manager (rpm, apt etc). MacPorts on Mac can install `jq` for you or you can download it from GitHub.

Usage is like:

```
# store command output in variable (to avoid repeated lookups)
VPCS=$(aws ec2 describe-vpcs)

# pull out the first vpc ID (same pattern for other variables)
echo $VPCS | jq -r .Vpcs[0].VpcId
# or capture / store that
VPCID=$(echo $VPCS | jq -r .Vpcs[0].VpcId)

# then use as normal
echo vpc id is $VPCID
```

The script `ec2_linux_vpc_setup.ps1` will set up this environment. You can also run the commands manually or create the above setup in the console.

You may need to run `Unblock-File` to let the Powershell script run after downloading as described in Microsoft's documentation.

5 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.

5.1 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 security group' --vpc-id
```

Successful output will look like:

```
{
    "GroupId": "sg-08cafd37327645e81"
}
```

5.2 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 sg-08cafd37327645e81 --protocol tcp --port
```

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!

5.3 Egress rules

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

6 Instance setup

We will setup an EC2 instance as follows:

- AMI: Amazon Linux
- Configuration: `t2.micro`
- VPC: `LAB_VPC`
- Subnet: `LAB_1_SN`

- Security group: `LAB_SG`

Using the command:

```
aws ec2 run-instances \
 --subnet-id subnet-08f1cd51b749c3ce2 \
--image-id ami-0bb3fad3c0286ebd5 \
--instance-type t2.micro \
--key-name LAB_KEY \
--security-group-ids sg-0ea635cfaa7ab103e
```

- The `image-id` is the AMI
- Instance type is `t2.micro` (good for general purpose stuff)
- The SSH key `LAB_KEY` is to be used.
- Security group is the ID of the `LAB_SG`.

Image IDs are region and account-dependent. They also get updated as Amazon update the images.

6.1 Available image names

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[
```

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

6.2 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 `
--image-id resolve:ssm:/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2
# other parameters as normal
```

6.3 Instance information

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

```
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-0bb3fad3c0286ebd5",
            "InstanceId": "i-04acdf703ecc03471",    "OwnerId": "637116340434",
            "...": "...",
            "ReservationId": "r-0536a89f39c4c6324"
}
```

The public key is copied by AWS into the ec2-user's authorized_keys file. This uses a combination of cloud-init on the EC2 instance and the AWS Instance Meta Data Service (IMDS), both of which we will meet later on.

7 Connecting to instance

7.1 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 i-07affd402ac286fe6
```

The public IP is listed with the network interface.

We can make some sense of this using PowerShell.

```
# First we capture the JSON and convert to PowerShell objects.
$reservations = $(aws ec2 describe-instances --instance-id i-07affd402ac286fe6 | ConvertFrom-Js
# Then we extract the public IP.
$publicIp = $reservations.Reservations.Instances[0].NetworkInterfaces[0].Association.publicIp
```

7.2 Connecting to instance over SSH

In PowerShell/Bash we can just use the SSH command to connect to the instance. We will connect as the ec2-user. By default, SSH will try all private keys so we don't need to specify which.

```
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 ~]$
```

8 EC2 instance termination

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

```
aws ec2 terminate-instances --instance-ids i-07affd402ac286fe6
```