SJ GREEN
27TH January 2025

Demonstration of the AWS CLI to Launch a VPC

Important Preparation: Way before the session begins, and you are displaying your screen, get the AWS Access Key and Secret Access Key for the IAM user you intend to use, into position. Paste them in Notepad, in a window. If you have them in a file on your local computer, then you will have to expose all your embarrassing filenames to everyone watching.

Note 1 - These instructions are written specifically for a demonstration using CMD on Windows. When writing out the command to execute, in this article, I have always used a new line for a parameter. However, generally, with CMD on Windows you should not do this. Do not have your commands spread across multiple lines. Instead, simply type one space and then give the name of the parameter. Two dashes are written before the name of a parameter.

Note 2 – The best demonstrations are the ones where things go wrong. You want to prove to students that you possess a stable capability to use the technology. This is why I choose not to record a video of myself and play it – a video is not real. So, try to relax and model good troubleshooting skills. Usually, you are four minutes from a solution, and it will be a space character or a comma or an uppercase letter that should not be there.

Note 3 – Using the AWS CLI on Windows is the kind of activity that it helps to warm up with. If you have not executed any commands for several months, then things will be difficult on a first pass. You will type `Instanceid` instead of `InstanceId` and then realise you meant to type `ImageId` anyhow; you'll forget the difference between `--filter` and `--query` and countless other minutiae. If this is the state you are in, then start by reading through this entire article. Then take a first attempt at the activities, and a second attempt. Give yourself a few uninterrupted hours to really "get into" it.

Begin by explaining that I will be demonstrating the launch of an EC2 Instance. If I am to launch an EC2 Instance, then I need to know the ID for an Amazon Machine Image (AMI). The question raised is how I can come to have this knowledge.

Execute the following command:

```
aws ec2 describe-images
```

This will provide the opportunity to demonstrate a command is deficient in two respects. First, it takes a long time to provide a response. Second, the response lacks precision.

Explain that it is still useful to run the command, because we can learn about the keys in the response. (For example, there is an Images object, and within it there is an `ImageId` key and other helpful keys.)

Do not wait for too long. If you get not response, then move on. It the unrefined command (aws ec2 describe-images) has not provided a response, then use `Ctrl+C` to cease waiting for the response. Point out how, exactly, we **know** that the command has not yet responded (we know because there is no prompt displayed). A novice might worry about how we distinguish a command that is responding, albeit slowly, from a command that was successful but had no observable response. Explain, therefore, that this command is responding slowly.

Use a more refined command:

```
aws ec2 describe-images
      --query Images[0:2]
```

The response from this command is important, because it tells us the material we have to play with. Explain that we are using an index expression to only return 2 objects the (0^{th} and 1^{st}). Take a look at all the keys related to particular images. For example, there is `ImageId`, `Hypervisor`, `Name`, `State`,

Execute a more refined command (ensure you look at all the lines of the command printed below, especially is this text is near to the edge of a page):

```
aws ec2 describe-images

      --query Images[*].[ImageId, Hypervisor, Name]
```

You may need to put the value of the `query` parameter inside speech marks.

Explore the variety of values for Name. This gives insight into the various types of Amazon Machine Image there are.

The Need For the Filter Parameter

Explain that in order to prevent the response from being too long, we really need to be more precise in what we ask for. The `--query` parameter does not help us in this regard. It means that we only see certain keys for each object, but we are still asking for an enormous number of objects to be returned.

Explain that we will now use this command, to give a **first** example of how the `--filter` parameter works:

```
aws ec2 describe-images

      --query Images[*].[ImageId, Hypervisor, Name, State]

      --filter "Name=state,Values=available"
```

To use the `--filter` parameter, we provide a Name and some Values.

Here, we are interested only in images whose state is available.

Even though in the object, the key is "State", with an initial upper case letter, we write `Name=state`. For some reason, the `--filter` parameter demands that its names are in all-lowercase.

Here is a second example, using the `--filter` parameter. This time, we are only interested in cases where the name begins `amzn2-ami-hvm-*`

So, we type:

```
aws ec2 describe-images
      --owners amazon
      --filters "Name=name,Values=amzn2-ami-hvm-*"
      --query Images[*].[ImageId,Name]
      --output table
```

You will notice that we have introduced a few other parameters. We have used the `--owners` parameter and the `--output` parameter.

Sometimes, Windows CMD has a problem if you include a space character between the `Name` and `Values` specification within `--filter`.

In other words, this will be unacceptable:

```
--filters Name=name, Values=amzn2-ami-hvm-*
```

This, meanwhile, will be acceptable:

```
--filters Name=name,Values=amzn2-ami-hvm-*
```

Notice how there is a comma and then an uppercase V, with no spaces interrupting.

Hopefully, in the output, you see a table. In the table there are specific AMI IDs. Copy one of the AMI IDs and paste it on Notepad for later. *You will need it when launching an EC2 Instance.*

Part 2 – Launching a VPC

We can attempt the following command:

```
aws ec2 create-default-vpc
```

You might see the following result:

```
An error occurred (DefaultVpcAlreadyExists) when calling the
CreateDefaultVpc operation: A Default VPC already exists for
this account in this region.
```

Discuss why there might already be a VPC in existence.

You might then run this command:

```
aws ec2 describe-vpcs
```

Explore the properties of the VPC. Note down the VPC ID.

Bonus Question: Why do we not see the Region in which this VPC is situated? The answer is that the AWS CLI is already scoped down to a specific Region. Consider the `aws configure` command which allows you to do this.

We are now building up to executing the following command, but do not execute the command yet:

```
aws ec2 launch-instances
```

Type the following:

```
aws ec2 launch-instances –help
```

This is an example of being able to type `--help` after a specific command.

There are a few things to draw attention to in the response. First, it should say somewhere:

```
aws: error: argument operation: Invalid choice, valid choices
are:
```

This is the AWS CLI trying to provide feedback to us. Specifically, it is telling us that the command we typed was an invalid choice. It is correct because there is no such command as `launch-instances`. Second, it should say:

```
Invalid choice: 'launch-instances', maybe you meant:

  * run-instances
```

This is an example of the AWS CLI providing a suggestion about the command we may have intended. Now attempt the regular and rectified run command:

```
aws ec2 run-instances
```

You will be met with the following:

```
An error occurred (MissingParameter) when calling the
RunInstances operation: The request must contain the parameter
ImageId
```

Now, let's think about this for a moment. Presumably, there are lots of parameters that are required, and not just ImageId. Learning about each parameter one-by-one, via the response of the AWS CLI is a tedious way to do things. We might ask: Is there a way to find out all of the parameters that are required, in a single act?

The answer is that there is. We can execute:

```
aws ec2 run-instances --help
```

I find that the above command returns an error. The word `help` should have no dashes in front of it. Execute this instead:

```
aws ec2 run-instances help
```

If you repeatedly press the `Enter` key on your keyboard, you should eventually see a list of parameters. Examples include:

`--block-device-mappings`

`--image-id`

`--instance-type`

`--ipv6-address-count`

`--ipv6-addresses`

Some of these parameters will be in square brackets. I have found that all the parameters are in square brackets. This is some sort of mistake with the AWS CLI. After all, we know that the `--image-id` parameter is required.

It turns out there is a way to discover all the parameters which are required. You can use `--generate-cli-skeleton`. This is an incredibly useful parameter, so please pay attention here. Let's look at how the AWS CLI reference describes it:

> `--generate-cli-skeleton` (string) Prints a JSON skeleton to standard output without sending an API request.
>
> If provided with no value or the value input, prints a sample input JSON that can be used as an argument for `--cli-input-json`.
>
> Similarly, if provided yaml-input it will print a sample input YAML that can be used with `--cli-input-yaml`. If provided with the value output, it validates the command inputs and returns a sample output JSON for that command.
>
> The generated JSON skeleton is not stable between versions of the AWS CLI and there are no backwards compatibility guarantees in the JSON skeleton generated.

So, the AWS CLI provides you with an example of the kind of thing *you* might input.

However, the above text is misleading. It states the inclusion of the parameter "prints a sample input JSON that can be used as an argument for `--cli-input-json`."

This might lead you to believe that you can directly paste the JSON in, to be the value of `--cli-input-json`. Well, you are free to attempt this.

When I attempt this, I get back an error line for each part of the JSON:

```
Error parsing parameter 'cli-input-json': Invalid JSON
received.

"BlockDeviceMappings": ['"BlockDeviceMappings":' is not
recognized as an internal or external command, operable
program or batch file.

[…]
```

An alternative way of doing things is to use the `file://` notation. So, we *can* include the sample JSON. But we are going to point to it, rather than paste the JSON in directly.

Do not run this command yet:

```
aws ec2 run-instances –cli-input-json file://run--
instances.json
```

Before executing the command, take a look over the file. Specify an instance type such as `t3.medium`. Paste in your AMI Id which you saved to Notepad earlier.

I am told the following:

```
Parameter validation failed:

Invalid value for parameter
ElasticInferenceAccelerators[0].Count, value: 0, valid min
value: 1
```

Therefore, I go and specify that we have 1 Elastic Inference Accelerator.

Now, after running the command again, I am told:

```
An error occurred (UnknownParameter) when calling the
RunInstances operation: The parameter SpreadDomain is not
recognized
```

Where does the demand for an Elastic Inference Accelerator come from? Is it because I'm using a particular AMI or a particular Instance Type?

This is probably because of the Instance Type.

New Attempt

Let's suppose we make a new attempt. We just run the following:

```
aws ec2 run-instances --image-id ami-008524bd0e265fbab --
instance-type t3.micro
```

So, we have just decided to use two parameters. (I do this because Ivo Pinto did it on page X of his book AWS Practical Projects.) It seems we can get away with this.

But we get this error:

```
An error occurred (InvalidParameterValue) when calling the
RunInstances operation:

The architecture 'x86_64' of the specified instance type does
not match the architecture 'arm64' of the specified AMI.

Specify an instance type and an AMI that have matching
architectures, and try again.
```

```
You can use 'describe-instance-types' or 'describe-images' to
discover the architecture of the instance type or AMI.
```

How can we find out the architecture of the AMI we used? I can use the following command:

```
aws ec2 describe-images --image-ids ami-008524bd0e265fbab --
query "Images[0].Architecture"
```

Where the AMI ID provided is the AMI I used. It turns out the AMI I used employs an ARM64 architecture. A t3.micro instance, on the other hand, uses an x86_64 architecture.

Ivo Pinto uses `ami-0c101f26f147fa7fd` on page 15 of his textbook ("AWS Cloud Projects")

Unfortunately, this does not work. When I try to launch an instance using `run-instances` and specify this AMI and t2.micro I am told:

```
An error occurred (InvalidAMIID.NotFound) when calling the
RunInstances operation: The image id '[ami-0c101f26f147fa7fd]'
does not exist
```

So, we cannot cheat! We really have to get on top of this and work out for ourselves how to find an Amazon Machine Image that is appropriate.

Properly Selecting an AMI, for Ourselves

Let's suppose we want to select an AMI. First, we should fix the instance type we are using. I want to use a simple and cheap one, because this is just a demonstration.

I can find the instance types available using the AWS CLI (the User Guide for EC2 gave me this idea):

```
aws ec2 describe-instance-types --region us-west-2
```

Notice how I am using the Region parameter to ensure I am looking at instance types in London. The response is fascinating. For each instance type, I am told things such as:

- Whether it is `FreeTierEligible`,
- whether it is the `CurrentGeneration`,
- the network performance in gigabits (e.g. 37.5 gigabits)
- Whether it supports `BareMetal` usage
- `PlacementGroupInfo` tells me whether it supports the cluster placement strategy or the partition placement strategy or the spread placement strategy
- `MemoryInfo` tells me the memory in MiB
- `Hypervisor` tells me the hypervisor used (e.g. Xen)

It is difficult to use this CLI response solely to select an instance type. This is because you need that knowledge from the documentation which tells you what `t3` means, and what `m5` or `c5` means.

The EC2 User Guide tells me that General Purpose instances are found in `t3.micro`. So I type:

```
aws ec2 describe-instance-types
        --region us-west-2
        --instance-types t3.micro
```

This response to this allows me to fix a few details, in anticipation of locating an AMI. For example, I can confirm using `Hypervisor` that a nitro hypervisor is used, I can confirm using the `FreeTierEligible` key that it is not in fact eligible for the free tier, `ProcessorInfo` tells me that the manufacturer of the processor is Intel and that the only supported architecture is `x86_64`.

This is really important to note. The only supported architecture on the instance type I intend to use is x86_64. This is a 64-bit architecture. It is also known as AMD64 or x64.

Selecting an AMI

Now that we have fixed these details related to the instance type, let's search through some candidates for our AMI.

We're going to use `aws ec2 describe-images` again. But let's see what parameters we can use here. Type:

```
aws ec2 describe-images help
```

I try this command:

```
aws ec2 describe-images
--filters Name=architecture,Values=x86_64
--owners amazon
--page-size 5
--max-items 5
```

Remember that within `--filter` it must be `Values` not `Value`.

One thing that is disappointing is that my t3.micro has a nitro hypervisor. Yet I am not able to filter for a nitro hypervisor. The aws ec2 describe-images command tells me that the only options for "`hypervisor`" are "`ovm`" and "`xen`". Well, that is not very helpful. I cannot select for instances that use a "`nitro`" hypervisor, it seems.

We probably want to add another thing within our `--filter` parameter. How can we do this? Do we need some sort of delimiter character? The answer is that you can just use a space.

Our next command will be:

```
aws ec2 describe-images
    --filters Name=architecture,Values=x86_64
              Name=virtualization-type,Values=hvm
              Name=name,Values=amzn2-ami-hvm-*
              Name=ena-support,Values=true
    --owners amazon \
    --query "Images[*].[ImageId,Name,CreationDate]"
    --output table
```

You should care about the Operating System (OS) on your AMI. For this reason, we specify that the name of the AMI should begin "`amzn2-ami-hvm-*`".

Bonus question: What else can you spot in the command above?

Please really study the large, multi-line command above. It contains within it the procedural knowledge for settling upon an AMI to use. See if you can memorise the four filters.

Is there a way to scroll to the bottom of CMD on Windows? Yes, practice using `Ctrl + End` on your keyboard.

You should get a table as the output. Copy and paste an AMI from the list.

Now attempt to launch an EC2 once again:

```
aws ec2 run-instances --image-id ami-012a45249baf877bd --
instance-type t3.micro
```

If the instance launches, you should get a response. The keys includes might be things such as AmiLaunchIndex, ImageId, InstanceId, InstanceType, LaunchTime.

Paste the value for `InstanceId`, the value for `VpcId` and the value for `SubnetId`.

Scroll down to made a note of the security groups on the instance.

Firewalls and the EC2 Instance we have Just Deployed

1 - Security Groups

Run the following command:

```
aws ec2 describe-security-group-rules
```

There should be two rules. Examine the rules.

Explain that one of the rules allows all outbound traffic to *any* IPv4 address (0.0.0.0/0). The value of `-1` for the `FromPort` key and the `ToPort` key effectively means that all ports are allowed.

One of the rules looks like this:


```
{
            "SecurityGroupRuleId": "sgr-074d8d30ad81d431c",
            "GroupId": "sg-03d882f0e760f199b",
            "GroupOwnerId": "737911634202",
            "IsEgress": false,
            "IpProtocol": "-1",
            "FromPort": -1,
            "ToPort": -1,
            "ReferencedGroupInfo": {
                "GroupId": "sg-03d882f0e760f199b",
                "UserId": "737911634202"
            },
            "Tags": []
        },
```


Let's analyse the rule above. It is an ingress rule. We know this because the value of `IsEgress` is set to `false`. We can see that the value of "`IpProtocol`" is "`-1`". This tells us, perhaps counterintuitively, that all protocols are allowed.

The value of "`FromPort`" and "`ToPort`" is set to `-1`. This tells us that this rule applies to all ports.

This rule allows all protocols and all ports to communicate with this Security Group, but only if the traffic originates from the same security group. Essentially, this is a self-referencing rule, which allows instances within the same Security Group to communicate with each other.

2 - NACLs

You can attempt the following command:

```
aws ec2 describe-network-acls
```

Ask students if there is anything interesting to note about the rules.

3 - Modify the security group

Run this command:

```
aws ec2 authorize-security-group-ingress
  --group-id sg-1234567890abcdef
  --protocol tcp
  --port 80
  --cidr 0.0.0.0/0
```

Obviously you should adjust the command, providing the appropriate value for `--group-id`.

Verify that a new rule has been created by executing:

```
aws ec2 describe-security-group-rules
```

Create an isolation security group

Execute this command to create a special quarantine SG:

```
aws ec2 create-security-group
--description Security group for isolating resources.
--group-name isolation-sg
```

Now we need to alter its rule to make it restrictive:

```
aws ec2 revoke-security-group-egress
      --group-id sg-03b9ab54cff532500
      --protocol tcp
      --cidr 0.0.0.0/0
      --port 0-65535
```

We must allow ourselves to access the EC2 Instance in order to manage it, so we need to allow inbound access from elsewhere:

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

Now we need to associate our EC2 Instance with this quarantine SG:

```
aws ec2 modify-instance-attributes
      --instance-id <instanceid>
      --groups <id of security group>
```

4 - Exploring Subnets

Execute this command:

```
aws ec2 describe-subnets
```

Ask the question: where is my running EC2 Instance? Which subnet is it in? How would I find out?

Bonus question: is it possible to move an EC2 Instance from one subnet to another?

5 - Create a new subnet

6 - Create a distinct VPC

Clean up Operation

You **must** demonstrate the clean-up operation. It is part of modelling good practices. It is dangerous to put these steps off until later, because you might forget.

(1) Terminate the EC2 Instance

Execute this command:

```
aws ec2 terminate-instances --instance-id <instance-id>
```

(2) Deleting the security groups

(3) Clean up access keys

Log into the AWS Management Console on a web browser. Log in as the root user.

Delete the AWS Access Key and Secret Access key for the IAM user used for the demonstration.

It is more secure to use the Console to log in as root. You use an MFA code that expires and a password which should be memorised.