Module 13
Computing as a Service and Virtual Machines (EC2) – Part 1

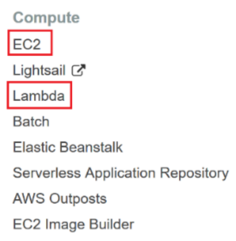
Overview

Serverless computing, such as AWS Lambda is part of PaaS (Platform as a Service), where the cloud provider is providing us (the users) with a platform for our computational needs. In the AWS Lambda modules, we saw how versatile and easy it is to set up this computation environment in the cloud. There were no provisioning of physical servers, operating systems choice, etc. With few clicks, a Lambda function can be set up to run our code. And by using serverless computing we gained 2 advantages out of the box:

1. Economic: we are charged for the time of execution only. If our Lambda code executes for a total of 5 minutes per day, we only pay for those 5 minutes.

2. Design and Engineering: Lambda automatically scales our function depending on how heavily the function is being called. When AWS detects and determines that the Lambda function is being heavily called, the AWS infrastructure spins additional instances of our Lambda function and load-balance the workload among these instances. And these new instances can run on the same machine or different machines. This is a huge benefit as developers can dedicate their efforts to the logic of their applications. For example, Alexa skills code run in Lambda functions. If you write a skill that is not widely used then scalability might not be an issue. But if you write an Alexa skill that becomes a huge hit with millions of users worldwide, Lambda takes care of scaling your Lambda computing environment to handle millions of Alexa questions being asked of your skill. This relieves you from the burden of addressing scalability and, instead, concentrate on the business rules of your application.

Another cloud compute service allows users to provision Virtual Machines (VMs) in the cloud with minimal effort, and scale their computing environments as demand asks for it. The difference from serverless computing is that users assumes control of their VMs characteristics: operating system and version, number of CPUs, memory size, software, etc. In AWS this service is known as **EC2** (**Elastic Compute Cloud**). Unlike serverless which fits under PaaS, EC2 fits under IaaS (Infrastructure as a Service). Users are basically renting infrastructure from a cloud provider such as Amazon AWS or Microsoft Azure. Both serverless (Lambda) and VMs (EC2) are under the Compute family of services. EC2 predates Lambda.



Companies rent VMs for many reasons. It can be as an alternative to desktop computers, to provision a fleet of VMs with different OSs for testing purposes, to set up clusters with certain hardware characteristics for a short time, to scale for bursts of business activities without the need to commit and purchase lots of servers that sit idle for the majority of time, etc.

Going forward, we will use the terms **VM** and **EC2 machine** interchangeably. They mean the same thing in the context of AWS.

When you provision a VM from a cloud provider you do not get a physical computer the same way you do when you buy one from a computer hardware vendor. Instead you get a Virtual Machine (VM). A VM is a software image of a complete machine that runs on a server like any other program. A VM on a physical machine is sometime referred to as the ***guest*** operating system (to distinguish it from the physical machine itself which is usually referred to as the ***host***). One physical computer (the host) can have multiple VMs (guests) running on it (Figure 1). The host uses a software program known as **hypervisor** to manage the VMs and control the sharing of the physical machine's resources (e.g., disk, memory, etc.) among the VMs (Figure 2). Examples of hypervisors are Citrix Xen, Microsoft Hyper-V, and VMWare ESXi.

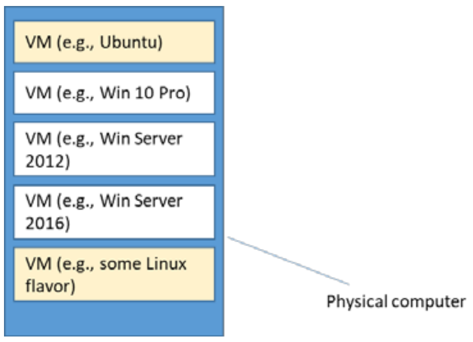

VM (e.g., Ubuntu)
VM (e.g., Win 10 Pro)
VM (e.g., Win Server 2012)
VM (e.g., Win Server 2016)
VM (e.g., some Linux flavor)
Physical computer

Figure 1: One physical computer can have multiple VMs running on it. In this example, there are 3 Windows VMs and two Linux VMs all running on a single physical computer.

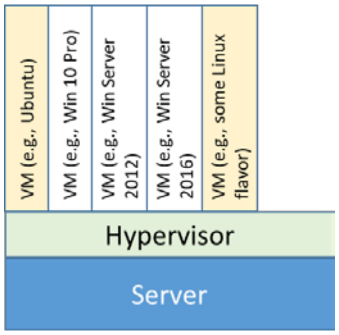

VM (e.g., Ubuntu)
VM (e.g., Win 10 Pro)
VM (e.g., Win Server 2012)
VM (e.g., Win Server 2016)
VM (e.g., some Linux flavor)
Hypervisor
Server

Figure 2: The physical server uses a software known as hypervisor to manage the VMs and to allocate resources to them.

Because each VM is running as an isolated program, if one VM crashes it does not crash the entire server. Applications running on one VM are not aware of the existence of other VMs.

Because a VM is a complete OS instance it can take a few minutes to start up. This is one reason, among others, that makes VMs not suitable for certain scenarios where speed is essential. In cases where speed is essential, another level of virtualization are **containers**. Containers are extremely light-weight, and allow you to package an application and all its dependencies and data into a single unit. Containers are not complete OSs like VMs. Instead, a single VM can have multiple containers (Figure 3).

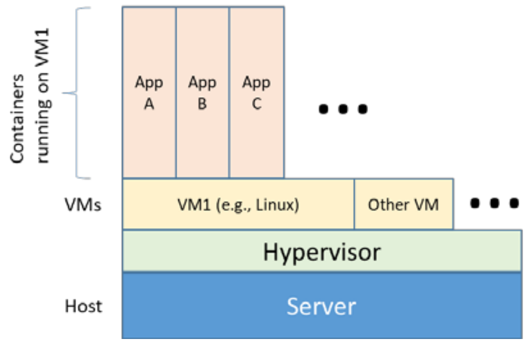


Figure 3: Containers are another level of virtualization that run on top of an OS.

As the numbers of containers increase, managing them becomes cumbersome and tricky. Many tools such as **Kubernetes** have been developed to address this challenge.

The downside of containers is that they are less secure than VMs. Because containers share the same OS, two containers running on one host are less isolated than 2 VMs running on one host.

Note that a physical computer in a data center is not necessarily dedicated to one customer. For example, the cloud provider can use the physical computer in Figures 2 to allocate the 5 VMs to 2 different customers: customer1 is a pharmaceutical company in Oregon that provisioned the 3 Windows VMs, while customer2 is a software company in Washington that provisioned the 2 Linux VMs.

Even some applications are shared among customers, a concept known as **Multi-Tenancy**. Let's give an example to clarify this concept: Assume an organization has a need for a database. In a traditional non-cloud setting, the customer buys a database server software (e.g., Oracle, SQL Server, etc.) and creates one or more databases on it – maybe 2 databases: SalesDB and EmployeesDB. In a cloud environment, the cloud provider buys a database server software license (e.g., Oracle, SQL Server, etc.) and creates on such database server multiple databases for different customers (Figure 4). Multi-tenancy is economical because licensing, software development, and maintenance costs are shared.

Sharing of resources is fundamental to cloud computing and is found throughout the cloud computing landscape: sharing a physical computer among multiple VMs, sharing an application among multiple customers, etc.

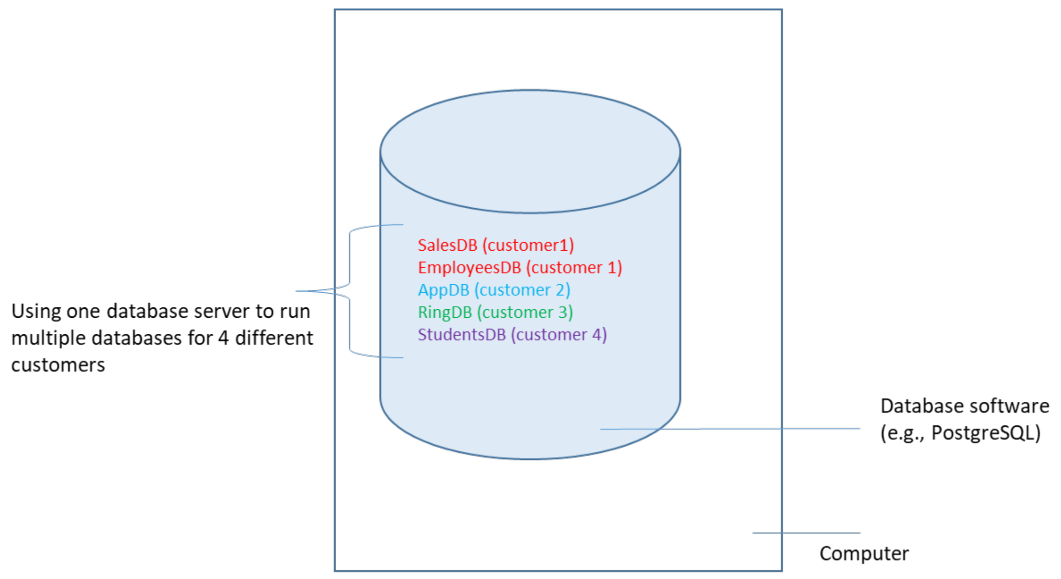

SalesDB (customer1)
EmployeesDB (customer 1)
AppDB (customer 2)
RingDB (customer 3)
StudentsDB (customer 4)
Using one database server to run multiple databases for 4 different customers
Database software (e.g., PostgreSQL)
Computer

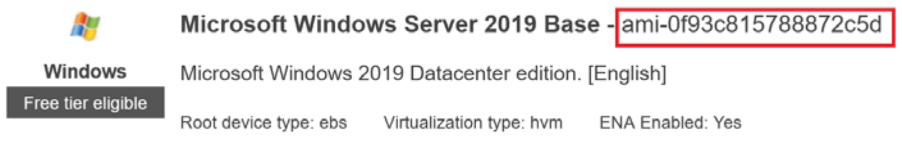
Figure 4: Multi-tenancy example: one database server shared among many customers.

Create an EC2 instance from the AWS Console

1. Login to the AWS Management Console and go to the EC2 service page.

2. Creating a Windows EC2 instance:

    a. Click the **Launch instance → Launch instance** menu button.
    This takes you to a page where you are presented with the choices of OSs and versions. Notice to the right of each choice is the presence of an **AMI** (Amazon Machine Image) number as shown below:



Microsoft Windows Server 2019 Base - ami-0f93c815788872c5d

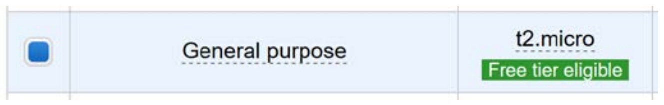
Microsoft Windows 2019 Datacenter edition. [English]

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

We will cover AMIs later where you will create your own AMI. In short, it is an ID that identifies a certain image configuration.

b. Find the **Microsoft Windows Server 2019 Base** and click the **Select** button. This takes you to a list of instance types. Instance types are VM configurations (number of CPUs, memory size, etc.) optimized for different use cases. The list of EC2 instance types and their configuration properties is found here.

c. Leave the default choice of **t2.micro** (or **t3.micro**) checked. Click the **Review and Launch** button.

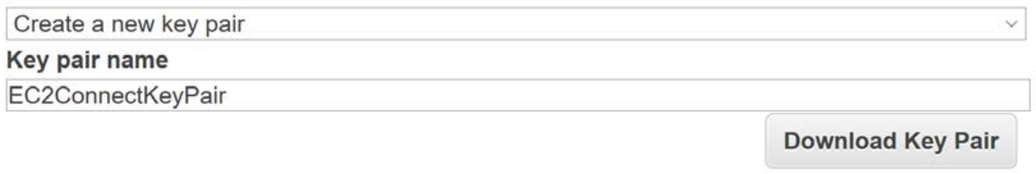


d. The review screen gives you a summary of what you have selected so far. Scroll down to the bottom of the page and expand the **Tags** section. Like many AWS resources (e.g., files in an S3 bucket) you can define custom key/value tags and associate them with a VM. This can be useful if your application is launching dozens of VMs with different purposes and you programmatically want to know to which group a given VM belong.

Click the **Launch** button.

A dialog titled “Select an existing key pair or create a new key pair” opens.

e. Select “Create a new key pair”. For the Key pair name, enter: EC2ConnectKeyPair



Create a new key pair

Key pair name

EC2ConnectKeyPair

f. Click the **Download Key Pair** button and save file **EC2ConnectKeyPair.pem** to a location on your computer (you will need this file later to be able to connect to the VM you are about to launch).

g. Click the **Launch Instances** button. (Note that EC2 instances take a few minutes to launch).

h. Click the **View Instances** button in the lower right corner of the window.

   You should see a table that lists your instances. The Instance State column tells you the state of the instance. When it becomes "running" it indicates that it is ready to be connected to.

   Also note some of the properties of your instance. For example, the instance type is t2.micro (or t3.micro). In my case the AZ is us-east-1e (which means it is in the **us-east-1** region (Northern Virginia) in AZ numbered **e**). Remember that the Northern Virginia AWS region (code-named us-east-1) has 6 AZs. In your case, your VM might get created in a different AZ (that is, other than **e**).

i. Check the checkbox at the start of the table row, then click the **Connect** button. A dialog opens.

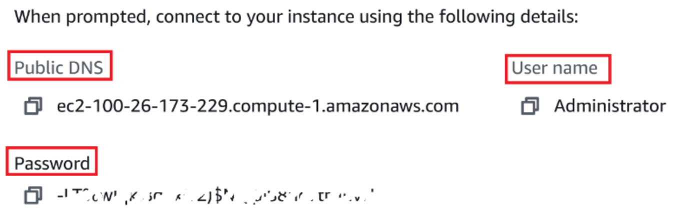
j. Click the **RDP client** tab.

k. Click the **Get Password** button.

l. Click the Browse… button and browse to the file **EC2ConnectKeyPair.pem** you saved earlier in step f.

m. Click the **Decrypt Password** button.

   The dialog should now show you the instance public DNS name, the Administrator user and its password.



When prompted, connect to your instance using the following details:

Public DNS

ec2-100-26-173-229.compute-1.amazonaws.com

User name

Administrator

Password

[illegible]

   Copy these 3 pieces of information and save them in a file (call it EC2ConnectCredentials.txt).

n. Click the **Cancel** button to close the dialog.

o. In Windows, open the Remote Desktop Connection app (Note that if you are using mac, you can download it – go here).

p. In the Remote Desktop Connection window:

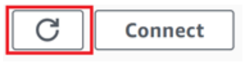
   i. Click the Show Options button in the lower left.

ii. For the Computer field, enter the EC2 Public DNS name.
iii. For username, enter: Administrator
iv. Check the checkbox **Allow me to save credentials**.
v. Click the Connect button.
vi. Enter the password you saved in step m above. Click Connect/OK.
vii. If you get a warning dialog (identity of remote computer cannot be verified…), check the checkbox "Don't ask me again…" and click the **Yes** button.

q. If successful, you should connect to the Windows EC2 machine you created in region us-east-1.

You can disconnect from this VM by clicking the Windows button in the lower left of the VM, then right-clicking on the Power icon, and then selecting the **Disconnect** menu.

Note that this does not shut down the VM. Go back to the EC2 page and view your instances table. The state of the VM should still be "running". You can refresh the table data by clicking the refresh button above the table.



AMI creation: configure the VM the way we like it.

In almost all cases the VMs used in your cloud application will need some additional software or configuration beyond the bare bone OS in order to do something useful. For example, you might have a VM that needs to execute some Python script using a specific version of Python. Therefore, not only you need to install your Python script, but you also need to make sure that the OS has the correct Python version installed. One impractical option is to just install Python and your script after creating the VM. There are 3 major inconveniences and problems with this approach:

1. If say you need to create 10 VMs, then you need to install Python 10 times and install your script 10 times.
2. If you need much more than just Python (e.g., additional software, libraries, settings, environment variables, folder structure, etc.) it will take very long to get a VM ready. Such delays mean that you cannot scale your application quickly.
3. As the configuration of a VM becomes more involved you are likely to make a mistake between configuring one VM and another. Ending up with many VMs (with different configurations) that say need to do identical things increases the likelihood of bugs.

Another more practical apprpoach is to first create a VM with all needed software and configuration. Then from this VM create an **Amazon Machine Image** (**AMI**). The AMI is a VM template. The next time you need a second or third VM, you just tell the EC2 service that you want a VM based on your AMI.

And the new VMs that get created will be identical to the one you used to create the AMI. This solves the problems outlined above.

1. Remote Connect to your EC2 instance that you created in the previous section.

2. To test the AMI concept we will assume that the VM needed by our cloud application needs 3 things:

   a. Python.
   b. A folder named Data with 2 subfolders Current and Archived.

      Data
        Current
        Archived

   c. An environment variable (let's call it DATA_PATH) that has the path of the **Data** folder created in (b).

3. Verify that Python is not installed on your VM (there are many ways to check: type **`python --version`** in the command window or look in the Windows registry by launching the Registry Editor app).
4. On the VM go to this URL: https://www.python.org/downloads/ (you may need to add pages to trusted sites if you get a security dialog from windows – click the Add button on the warning dialog).

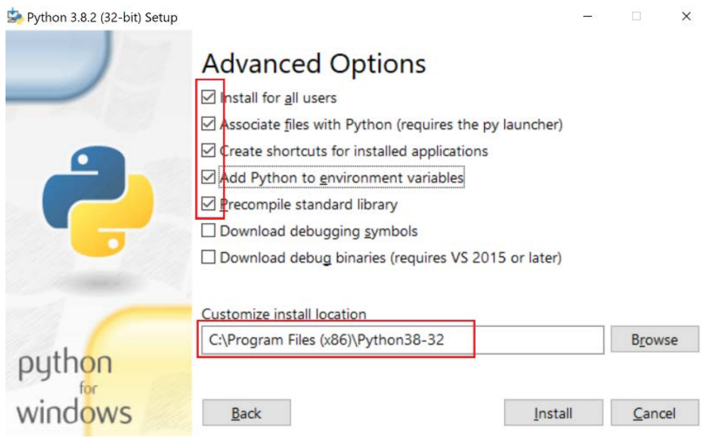
   Download the Windows installer.

   Double-click it to launch it.

   Choose the **Customize installation** on the Welcome screen.

   On the Optional Features screen, leave things as they are and click the **Next** button.

   On the Advanced Option screen, check the first 5 checkboxes and change the installation directory to C:\Program Files (x86)\... as shown below:



☑ Install for all users
☑ Associate files with Python (requires the py launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2015 or later)

C:\Program Files (x86)\Python38-32

Click the **Install** button.

5. When installation is complete test that Python is installed: `python --version`

6. Under the C:\ drive, create a folder named **Data**.

7. Under C:\Data, create 2 subfolders: **Latest** and **Archive**.

8. Click the Windows button and start typing: Edit environment variables.
   Select the Edit environment variables for your account menu.

9. In the Environment Variables dialog, click **New…**

   For Variable name, enter: DATA_PATH
   For Variable value, enter: C:\Data

   Click OK. Then close the Environment Variables dialog.

10. So far we made the following configuration changes to our VM:
    a. Installed Python.
    b. Created the Data folder (under C:\) with 2 sub-folders.
    c. Created an environment variable, DATA_PATH, that points to the location of the Data folder.

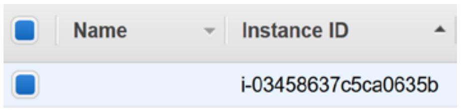
    We assume that this is all what our cloud application needs.
    We now want to create an AMI out of this VM.

11. You can now disconnect from the VM (click the Windows button in the lower left, then right-clicking on the Power icon, and then selecting the Disconnect menu).

AMI creation: create the AMI image.

1. Go back to the EC2 table that lists your instances.

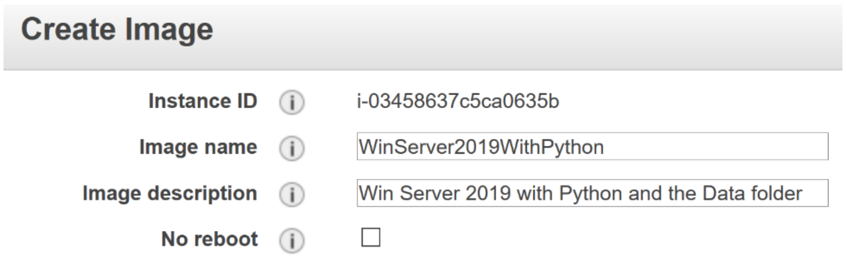
2. Make sure you have the instance you have been working with checked.



	Name	Instance ID
■		i-03458637c5ca0635b

Click menu **Actions → Image and templates → Create image**

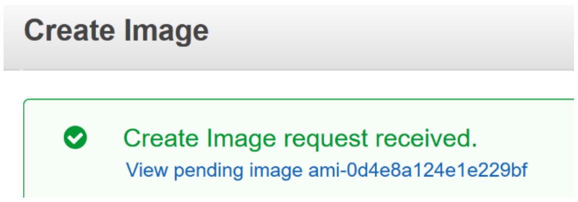
3. Fill in the Create Image dialog like this (your Instance ID will be different than mine):



Create Image

Instance ID	i-03458637c5ca0635b
Image name	WinServer2019WithPython
Image description	Win Server 2019 with Python and the Data folder
No reboot	☐

4. Click the **Create Image** button.
   You should see something like:



Create Image

Create Image request received.
View pending image ami-0d4e8a124e1e229bf

   Click the View pending images… link

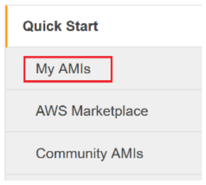
   This should take you to the AMI pages. The status should be **pending**.
   Wait until the status changes to **available**.

5. From the navigation bar on the left, click the **Instances** link.

   You have only one EC2 instance (the one you created at the beginning). We will now create another EC2 instance. The difference this time is that you will choose your AMI as a template (and not a generic one that AWS has prepared).

6. Click the **Launch Instance** button.

7. From the left navigation bar, click My AMIs.



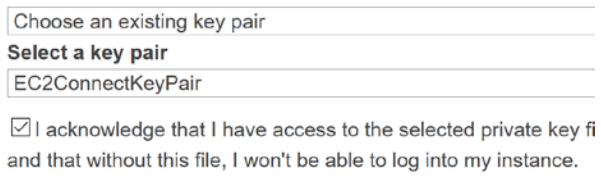
   You should see listed the AMI you created earlier: **WinServer2019WithPython.**
   Notice that AWS gave it some AMI ID (ami-…). This is the ID that uniquely identifies this AMI.

8. Click the **Select** button on the right of the AMI.

9. Leave the default t2.micro 1 GB memory instance type selected. Click the **Review and Launch** button.

10. On the Review screen, click the **Launch** button.

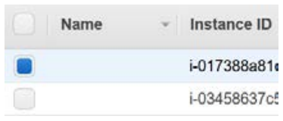
11. In the "Select an existing key pair or create a new key pair" dialog that opens, choose the Choose an existing key pair option (the one you created earlier: EC2ConnectKeyPair)



Choose an existing key pair
Select a key pair
EC2ConnectKeyPair
☑ I acknowledge that I have access to the selected private key fi
and that without this file, I won't be able to log into my instance.

12. Click the **Launch Instances** button.

13. Go back to your Instances table. Check the checkbox to <u>select the instance that just got created</u> (you can tell the new one by scrolling to the right of the table and inspecting the **Launch Time** column.



	Name	Instance ID
☑		i-017388a81
☐		i-03458637c

14. In the **Description** tab at the bottom of the page, copy the **Public DNS (IPv4)** name. You will need this to connect to this new VM.

15. Open the Remote Desktop Connection app.
    In the Computer field, paste the name you copies in step 14/

16. Click the **Connect** button.

17. Enter the same password you used with the first VM you created (the password you saved in file EC2ConnectionCredentials.txt).

18. When you login, verify the following:

    a. Python is installed.
    b. There is folder C:\Data with 2 subfolders: Archived and Latest
    c. There is an environment variable named DATA_PATH set to value C:\Data.

19. You can disconnect from this VM.

20. What we just demonstrated is that you do not need to use a generic AMI from Amazon. You can create your own image template to suit your needs. This is the common scenario since the machine you are creating is likely to need things to accomplish whatever your cloud app needs to do. And Amazon gives you this capability.

21. You get charged for EC2 instances even if you don't use them. Since we don't need these 2 instances, go ahead and delete both of them from the Instances table on the EC2 service page.

    To delete an instance, select it in the table, then choose menu **Actions → Manage instance state**.

    Choose option **Terminate**.
    Click the **Change state** button. Then click **Terminate** again on the warning dialog.

    After a minute or so the instances should indicate a status of "terminated".

22. Note that the instances are not related to your AMIs (which is a template). From the navigation bar on the left, click AMIs.

    You should still see your WinServer2019WithPython with a status of "available".

    Do not delete this AMI. You will needed in a future module.

For this module, there is no exercise to do and submit ☺.