DevOps – Laboratory 7

Configuration Management

In this lab you will focus on Configuration Management. Imagine you are in charge of 100 brand new servers which all have to set up so that they can be used to host your applications. Setting these machines up individually would take weeks or even months to achieve. Configuration Management takes care of this, as well as a number of other issues you may face.

Using a Configuration Management tool you can define exactly how you want a server to be configured using only code. This code can then be Version Controlled with the rest of our project.

Today you will utilize a Configuration Management tool called Ansible. You will use this tool to manage an AWS EC2 instance and configure it automatically. Due to the resource available in AWS, you will look at Configuration Management in the context of a single machine but keep in mind that this could easily be extended to multiple machines.

If you had access to more resource, other VMs could be connected as Nodes to the VM you will use today. In that case, the scripts you run to configure your single VM would be automatically run on the Node machines as well, ensuring that the configuration is identical, without you having to interact with them directly.

So for the purposes of this Lab, try to imagine that you are managing a whole collection of VMs and that the scripts you run today are being run across all of these servers.

# Getting Started

You will need to launch an AWS EC2 Instance on which you will complete this Lab.

Use the guidance in Lab 1 to configure an EC2 instance now but select the **t2.medium** size. You do not need to open any other ports for this lab.

When you have successfully connected to your new VM via SSH you are ready to continue with this Lab.

**PLEASE REMEMBER TO STOP OR TERMINATE YOUR VM AT THE END OF THIS LAB SO THAT YOU DO NOT USE UNNECCESARY CREDIT!!**

# Important Information

This Lab documents will contain commands that you should run in the bash terminal (both locally and on your VM). This code will be provided in boxes as shown below:

|  |
| --- |
| This is a sample command that fits on one line  This is a much longer sample command that covers multiple lines but should still be copied as a single command |

As you can see, some commands will be short and clearly displayed on a single line. This can be copied directly into the Terminal. Other commands such as the second one are much longer and do not fit on one line. If you are going to copy and paste these commands make sure to copy all lines and paste as a single command. Each new command is separated by a blank line so that each command is clearly distinguished.

**If you are unfamiliar with using the Terminal or Bash it is highly recommended that you complete the following Tutorial:**

[**https://ubuntu.com/tutorials/command-line-for-beginners**](https://ubuntu.com/tutorials/command-line-for-beginners)

This tutorial is written for Ubuntu but as the Terminal we are using (GitBash for Windows or Terminal for MacOS/Linux) also uses Bash, the commands are the same and the tutorial is very relevant. The Virtual Machines you create on AWS will utilise the Ubuntu Operating system and therefore this tutorial will also be useful when interacting with your VMS.

Section 7 of this Tutorial covers SuperUser which is not applicable to Windows Systems but will be important on your VMs so reading is recommended.

The following Cheat Sheet is also helpful for remembering Bash commands and will be a good point of reference throughout this Module:

<https://cheatography.com/davechild/cheat-sheets/linux-command-line/pdf/>

**If you encounter any issued during this process, please contact the Module Tutor.**

# Installing Ansible

From this point in the Lab it is assumed that you have created an AWS EC2 VM and have connected to it via SSH. The remainder of this Lab you should perform all command line tasks on the VM. If you have not already done this, please see the information earlier in this Lab.

1. Ansible is available through the Python package manager known as Pip. Use the following commands to update the package manager, install Python and Pip and then install Ansible:

|  |
| --- |
| sudo apt-get update  sudo apt-get install -y libssl-dev libffi-dev python3 python3-pip  sudo pip install ansible |

Now that Ansible is installed, you can use it to manage a VM in a number of ways including all of the manual configuration you have already seen in this module. This includes installing applications, downloading content and interacting with other applications such as Docker.

Ansible defines the configuration of a machine using a ‘playbook’ which is written in a markup language called YML (sometimes YAML - Yet Another Markup Language). These YML scripts are very sensitive to indentation which is done using spaces. **As a result, even a single incorrect space can mean that a playbook will not work correctly.**

When working through this lab and creating your playbooks, make sure that you copy absolutely everything (including spaces that you may not be able to see) from the code boxes provided.

# Ansible Playbooks

An Ansible Playbook is written in YML and can be used to define the configuration of a machine (or Virtual Machine in the case of this lab).

As previously mentioned, these playbooks are very sensitive to indentation but the structure is also very important.

**First we will talk through a playbook and then you will build and run it.**

Each playbook starts with a Header section which looks something like this. Note that the - name is not indented but the other three lines are indented by two spaces indicating that they belong to this group of commands:

|  |
| --- |
| - name: A sample playbook header  hosts: localhost  user: ubuntu  become: yes |

In this Header; the playbook is given a name, the host is defined as localhost (although this would change in the event of multiple machines) and the user is defined as Ubuntu which is the default user in the VMs you create in AWS. The final line (become: yes - where become refers to becoming the root user) tells the playbook to execute the commands using Super User (sudo) privileges which may or may not be necessary depending on the task you are performing.

The next section of the playbook that is defined is the Variables. These are optional but they make life much easier, especially when playbooks become more complex.

|  |
| --- |
| vars:  myPath: /home/ubuntu/ |

Note that in the box above, vars is indented by two spaces (two spaces indented from the header name) and the variables four spaces (i.e. two spaces indented from vars).

Now that the Header and Variables are configured it is time to look at the main section of the playbook. Any time you would like Ansible to perform an action you must define a Task. Have a look at the code below.

|  |
| --- |
| tasks:    - name: Update Package Manager  apt:  update\_cache: yes |

In this task, Ansible will update the package manager which is the equivalent to running the command ‘sudo apt-get update’ which you should be familiar with.

Now have a look at this task:

|  |
| --- |
| - name: Install an application  apt:  update\_cache: yes  name: default-jdk  - name: Remove an application  apt:  update\_cache: yes  name: default-jdk  state: absent |

This first task is the equivalent to ‘sudo apt-get install default-jdk’ which you used in the Docker Lab. You can remove packages using a similar task but defining the ‘state’ of the package as absent as also shown above.

This is fine for one package but what about installing several? You can use a new set of variables within a task to do this as shown below:

|  |
| --- |
| - name: Install an application  apt:  name={{ item }}  with\_items:  - libssl-dev  - libffi-dev  - python3  - python3-pip |

This task is equivalent to the command ‘sudo apt-get install -y libssl-dev libffi-dev python-dev python-pip’ which you used earlier before installing Ansible.

Note the use of {{ item }} and with\_items. This is the equivalent of a loop in ansible where {{ item }} will be sequentially replaced with the packages in the list defined. The task above will run the installation for each of the packages defined in the list.

Obviously you can do a lot more with Ansible than just use the package manager. Have a look at the task below which utilizes the ‘shell’ function which allows you to run a command exactly as you would in the terminal. The second task then uses the ‘file’ command to change the permissions of the file so that you can use it.

|  |
| --- |
| - name: Create an empty file using Ansible  shell:  cmd: touch "{{ myPath }}test\_file.txt"  - name: Change file ownership  file:  path: ../test\_file.txt  owner: ubuntu  group: ubuntu  mode: '777' |

You can also move files like this:

|  |
| --- |
| - name: Copy a file in Ansible  copy:  src: /home/ubuntu/test\_file.txt  dest: "{{ myPath }}test\_file\_2.txt" |

In the command above, the source file is defined by an absolute path which may be appropriate in some cases but typically it would be better to define common paths as variables. This can be seen in the destination command where the variable defined at the start of the playbook is utilized. Note that when using the variable as part of the path, the entire path must be enclosed in “ ”.

1. Create a new folder on your VM called ansible and change directory so that you are inside it. You should stay inside this directory for the remainder of the lab.
2. Create a new playbook called ansible-demo.yml using nano and paste the sections above in order into it. **Take care to copy the complete contents of each text box or you may encounter syntax issues.**

Once you have added all of the above sections to your playbook and saved the changes, you can run the playbook using the following command:

|  |
| --- |
| ansible-playbook ansible-demo.yml |

Take some time to look at the output that Ansible gives you. You can see that Ansible attempts each task in turn. Sometimes no changes are required for a task and Ansible will return text in green. Yellow text shows when a change has occurred, red for a failure and magenta is generally warnings which do not need to be addressed right now.

# Ansible for Docker Installation

Now that you have some experience with the basic commands in Ansible you can write your own playbooks for all sorts of tasks. The box below gives the set of commands that you have used to install Docker on your VM each week.

|  |
| --- |
| sudo apt-get update  sudo apt-get -y remove unscd  sudo apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-properties-common  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  sudo apt-key fingerprint 0EBFCD88  sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"  sudo apt-get update  sudo apt-get -y install docker-ce docker-ce-cli containerd.io  sudo groupadd docker  sudo usermod -aG docker $USER  sudo addgroup --system docker  sudo adduser $USER docker  newgrp docker |

In this section of the lab you will write an Ansible Playbook to run the Docker installation for you.

1. Create a new playbook called docker-playbook.yml in your ansible directory. Copy and paste the template playbook below into the file and save the changes.

|  |
| --- |
| - name: A playbook to install Docker on a VM  hosts: localhost  user: ubuntu  become: yes  tasks:  - name: 1. Update APT Package Manager  - name: 2. Uninstall the unscd package  - name: 3. Install dependency packages  apt:  name={{ item }}  with\_items:  -  - name: 4. Get APT Key  shell:  cmd: curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  - name: 5. Confirm Fingerprint  shell:  cmd:  - name: 6. Add APT Repository  apt\_repository:  repo: "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"  - name: 7. Update Package Manager  apt:  update\_cache: yes  - name: 8. Install Docker packages    - name: 9. Create bash Script for the Permissions Commands  become: no  copy:  dest: ‘/home/ubuntu/ansible/docker.sh’  content: |  #! /bin/bash  sudo groupadd docker  sudo usermod -aG docker $USER  sudo addgroup --system docker  sudo adduser $USER docker  newgrp docker  - name: 10. Change File permissions so that it can be executed  file: |

The playbook above has a total of 11 tasks. The name of each task has been numbered for you. The below table looks at each of these tasks and what commands in the Docker installation they are dealing with.

|  |  |  |
| --- | --- | --- |
| **Ansible Task** | **Docker Commands** | **Details** |
| 1. Update APT Package Manager | sudo apt-get update | Updates the package manager |
| 2. Uninstall the unscd package | sudo apt-get -y remove unscd | Remove the unscd package which is not required |
| 3. Install dependency packages | sudo apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-properties-common | Install various dependencies |
| 4. Get APT Key | curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - | Get the APT verification key for the Docker repository |
| 5. Confirm Fingerprint | sudo apt-key fingerprint 0EBFCD88 | Confirm that the APT Key is correct |
| 6. Add APT Repository | sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | Add the location of the Docker app repository to the APT package manager list |
| 7. Update Package Manager | sudo apt-get update | Update the package manager |
| 8. Install Docker packages | sudo apt-get -y install docker-ce docker-ce-cli containerd.io | Install Docker |
| 9. Create bash Script for the Permissions Commands | sudo groupadd docker  sudo usermod -aG docker $USER  sudo addgroup --system docker  sudo adduser $USER docker  newgrp docker | Create a bash script that contains the 5 commands which set the Docker user permissions |
| 10. Change File permissions so that it can be executed |  | Change the permissions of the bash script so that it can be executed |
| 11. Execute Script |  | Run the bash script to fix the Docker permissions |

The template has been written such that it incorporates a range of ansible functions.

Some of the tasks are incomplete (highlighted in red) and you should use the information in this lab to complete them appropriately. **Please take the time to understand the commands and how ansible is executing them.**

1. Use the above table to complete the docker-playbook.yml file. When you run the playbook you should get no errors (no red output) and the output of the command ‘docker version’ should look like this:

|  |
| --- |
| Client: Docker Engine - Community  Version: 19.03.2  API version: 1.40  Go version: go1.12.8  Git commit: 6a30dfc  Built: Thu Aug 29 05:26:49 2019  OS/Arch: darwin/amd64  Experimental: false  Server: Docker Engine - Community  Engine:  Version: 19.03.2  API version: 1.40 (minimum version 1.12)  Go version: go1.12.8  Git commit: 6a30dfc  Built: Thu Aug 29 05:32:21 2019  OS/Arch: linux/amd64  Experimental: false  containerd:  Version: v1.2.6  GitCommit: 894b81a4b802e4eb2a91d1ce216b8817763c29fb  runc:  Version: 1.0.0-rc8  GitCommit: 425e105d5a03fabd737a126ad93d62a9eeede87f  docker-init:  Version: 0.18.0  GitCommit: fec3683 |

If instead of the above output you see the following, you may need to run the docker.sh script manually using ‘./docker.sh’. This appears to be a permissions issue in AWS that means the permissions are not being set properly by Ansible in some cases. If you have a different error than the one below please speak to the lab tutor.

|  |
| --- |
| Client: Docker Engine - Community  Version: 19.03.2  API version: 1.40  Go version: go1.12.8  Git commit: 6a30dfc  Built: Thu Aug 29 05:26:49 2019  OS/Arch: darwin/amd64  Experimental: false  Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.40/version: dial unix /var/run/docker.sock: connect: permission denied |

Congratulations!! You have completed the seventh lab in this module. As you can see, using tools like Ansible to manage instances makes life much easier, especially if you imagine that you were using these tools to configure multiple nodes simultaneously. You now have all of the skills required to manage a server using Ansible and have seen how to, manage packages using the package manager as well as create, manipulate and execute files.

**PLEASE REMEMBER TO STOP OR TERMINATE YOUR VM AT THE END OF THIS LAB SO THAT YOU DO NOT USE UNNECCESARY CREDIT!!**