Lab 6 - Working with Docker Containers

Friday, December 25, 2020 10:46 AM

Developing on AWS - Lab 6 - Working with Docker Containers

2 hours 30 minutes Free ★★★★☆ Rate Lab

aws training and certification

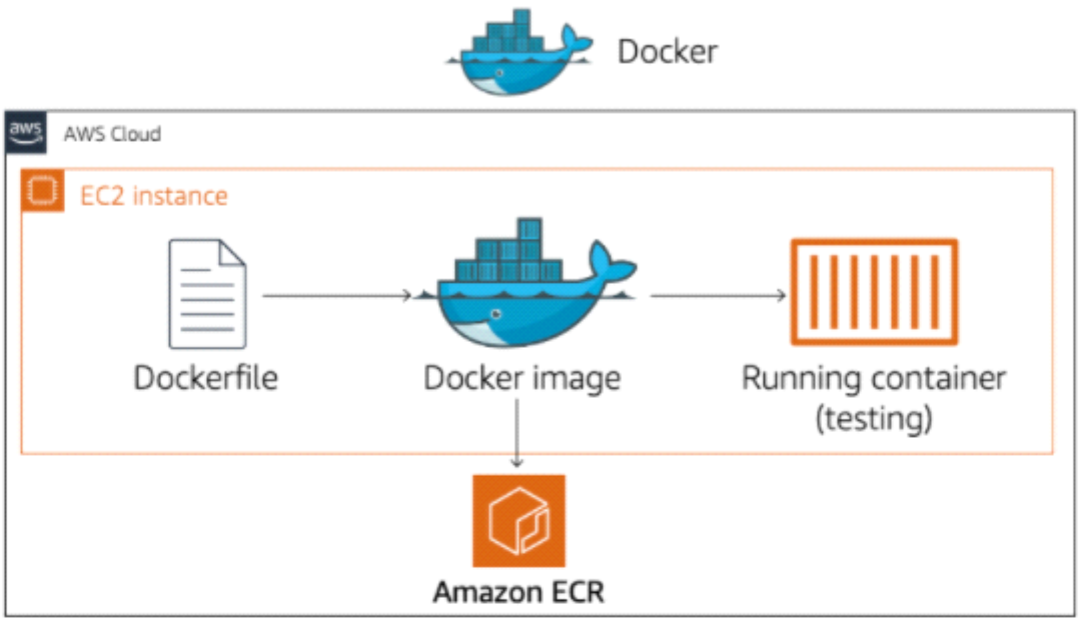
Corrections, feedback, or other questions? Contact us at *AWS Training and Certification*.

Overview

Overview

In this lab, you will learn how to host a basic website by using Docker containers.

Understanding how to create Docker images and launch containers is a very useful skill for a developer. You will be starting out with a base EC2 instance with Docker installed. Using this instance, you will create a Docker image and then launch a container. You will then create an ECR Repository to send the Docker image you will build.



Objectives

After completing this lab, you will be able to:

- Create a Dockerfile.
- Create a Docker image by using a dockerfile.
- Run a container from a Docker image.
- Interact with and administer your containers.
- Create an ECR Repository.
- Authenticate the Docker client to ECR.
- Push a Docker image to ECR.

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi running Microsoft Windows, macOS, or Linux (Ubuntu, SuSE, or Red Hat).
- An Internet browser such as Chrome, Firefox, or IE9+ (previous versions of Internet Explorer are not supported).
- You will need either an SSH client, such as PuTTY to connect to your development EC2 instance. Or, you can connect by using AWS Cloud9.

Note
You can use an iPad or tablet device to access these directions in the lab console.

Duration

This lab will require around **45 minutes** to complete.

Start Lab

1. At the top of your screen, launch your lab by choosing **Start Lab**

   This starts the process of provisioning your lab resources. An estimated amount of time to provision your lab resources is displayed. You must wait for your resources to be provisioned before continuing.

   If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. Open your lab by choosing **Open Console**

   This opens an AWS Management Console sign-in page.

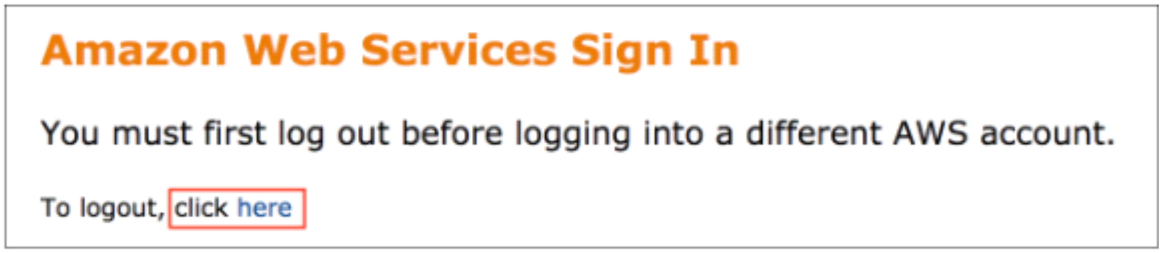
3. On the sign-in page, configure:

   - **IAM user name:** `awsstudent`
   - **Password:** Paste the value of **Password** from the left side of the lab page
   - Choose **Sign In**

⚠ **Do not change the Region unless instructed.**

Common Login Errors

Error: You must first log out



Amazon Web Services Sign In

You must first log out before logging into a different AWS account.

To logout, click here

If you see the message, **You must first log out before logging into a different AWS account:**

- Choose **click here**
- Close your browser tab to return to your initial lab window
- Choose **Open Console** again

Task 1: Connecting to Your Development

Environment

4. There are multiple ways for you to complete this lab. You can either use AWS Cloud9 or connect directly to your Dev Linux instance.

   If you don't have access to a local SSH client, or your access is restricted by a firewall, you can use a web browser and use the AWS Cloud9 service to complete this lab.
   However, it's a great way to learn about AWS Cloud9 by using this option for the lab.

   To complete the lab by using AWS Cloud9 (recommended), see the following directions:

   - Completing the Lab By Using AWS Cloud9

   To complete the lab by connecting to your Linux Dev instance, see the following directions:

   - Completing the Lab By Connecting to Your Amazon EC2 Linux Instance

Completing the Lab By Using AWS Cloud9

AWS Cloud9 allows you to write, run, and debug your code by using only a web browser. With AWS Cloud9, you have immediate access to a rich code editor, integrated debugger, and built-in terminal with a pre-configured AWS CLI. You can get started in minutes and no longer have to spend the time to install local applications or configure your development machine.

For this lab, you create and use a Cloud9 instance. All of the steps for this lab are completed in this section for working with Docker containers.

5. Choose **Services ⌄** and select **Cloud9**.

   If the screen you see next starts with **Region Unsupported**, select on the closest region to your location supported from the list of **Supported Regions** and continue with the next steps. If you are in doubt, use US East (N. Virginia).

6. Choose **Create environment**.

7. On the **Name environment** page, for **Name**, enter: `Docker`

8. Choose **Next step**.

9. On the **Configure settings** page, choose the following options:

   - **Environment type**: ● Create a new EC2 instance for environment (direct access).

   - **Instance type**: ● t3.small (2 GiB RAM + 2 vCPU).

   Since you will run a Docker container within that instance, more RAM is preferable. Due to this, the t3.small instance type is selected.

10. Leave the remaining settings as default and select **Next step**.

11. Review the settings and choose **Create environment**.

    It should take around two minutes for the environment to be created. Wait for it to complete. Once it's ready, you will be connected to your new Cloud9 environment.

12. The Cloud9 deployment creates an EC2 instance in your account. You will need to modify the Security Group (firewall) for that EC2 instance to allow you to test the application that you will deploy. Make sure that you take note

of the Public IP address of that EC2 instance to do that test.

To modify the Security Group, you need to go back to the AWS Management Console. In Cloud9, to return to the AWS Management Console, complete the following steps:

- At the top, you should see the Cloud9 menu bar.

Hint If you do not see the menu bar, it might be hidden. Select at the top of the Cloud9 environment to unhide the menu bar.

- Select **AWS Cloud9 -> Go To Your Dashboard**.
- Choose **Services** and select **EC2**.

13. In the left navigation menu, select **Instances**. You should see an instance with a name that starts with `aws-cloud9-Docker-...`. If you don't see that instance, you are currently in the wrong region. Select the same region that you selected when you had to create your Cloud9 environment.

14. Select **aws-cloud9-Docker-...**.

15. In the lower section, take note of the **Public IPv4 address** and save it for future use.

16. Select the **Security** tab.

17. Select the name of the **Security group** which has `aws-cloud9-Docker-...` in the name.

    This will take you to the Security Group section of the **EC2** management console.

18. Select the **Inbound rules** tab.

19. Select **Edit inbound rules**.

20. Select **Add rule** and input the following:

- **Type**: `HTTP`.
- **Source**: `Anywhere`.

The other settings will be automatically populated for you.

21. Select **Save rules**.

22. Go back to the **Cloud9** environment.

    If you have closed this tab in your browser, you can go back by navigating to the AWS Management Console. Choose **Services**, select **Cloud9**, and then choose **Open IDE** in the Docker card.

23. At the bottom of the screen, you will see the **bash** terminal. You can use this terminal to complete the steps in the lab.

    Keep in mind, you will be using the Cloud9 instance to complete this lab. You are not using it to connect to the remote Linux instance.

Cloud9 Task 2: Creating a Docker Image

Docker is already installed on the Cloud9 EC2 instance. You can verify that Docker is installed and see the version information.

The next step is to create your Docker image.

24. To verify that Docker is installed, enter the following command:

```
docker -v
```

You should see a response similar to the following:

Docker version 18.06.1-ce, build e68fc7a215d7133c34aa18e3b72b4a21fd0c6136

25. To make a directory called 'webapp' and change directory into it, enter the following command:

```
mkdir webapp && cd ./webapp
```

26. To download a static website from S3, enter the following command:

```
wget http://us-west-2-tcprod.s3.amazonaws.com/courses/ILT-TF-100-DODEVA/v3.3.10/lab-6-docker/scripts/website.zip
```

27. Enter the following command to unzip your static website:

```
unzip website.zip && rm -f website.zip
```

28. Enter the following command to change back into the parent directory:

```
cd ..
```

29. To create a new dockerfile, you will use the Cloud9 IDE. In the Cloud9 menu bar, select **File -> New File** and enter the following into the new **Untitled1** tab that just opened.

```
FROM ubuntu

#Set environment variable for tzdata to run non-interactive
ENV DEBIAN_FRONTEND=noninteractive

# Install apache and remove the list of packages downloaded from
```

```
apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get update -y && \
apt-get install -y apache2 && \
rm -r /var/lib/apt/lists/*

# Copy the website into the apache web root directory
COPY webapp /var/www/html

EXPOSE 80

CMD ["apachectl", "-D", "FOREGROUND"]
```

This is an example of a Dockerfile. Your Dockerfile does the following:

- Downloads the image ubuntu from an image repository.
- Installs apache and removes the packages downloaded from apt-get update.
- Copies your web application into the image.
- Exposes tcp/80 to allow HTTP connections inbound.
- Starts apache.

30. To save the file, select **File -> Save**. For **Filename**, enter: `dockerfile` and select **Save**.

31. The next step is to build an image for your application. Go back to the bash terminal. To build an image, enter the following command (make sure to copy the whole command including the '.')

```
docker build -t webapp-image .
```

This command builds an image from a Dockerfile located in '.' (the current directory). Then, it will tag the image with a name *webapp*.

32. You should now have a Docker image. You can verify your image by entering the following command:

```
docker images
```

Expected output:

```
REPOSITORY          TAG                 IMAGE ID
CREATED              SIZE
webapp-image        latest              eeb99f639bd2        About
a minute ago   196MB
lambci/lambda       python3.8           f33b8c321346        10
days ago          521MB
lambci/lambda       nodejs12.x          ad65b6f59aaa        10
days ago          384MB
lambci/lambda       nodejs10.x          91e2fa511e23        10
days ago          379MB
lambci/lambda       python3.7           89450c5a34a0        10
days ago          945MB
lambci/lambda       python3.6           927a0dba7c51        10
days ago          893MB
lambci/lambda       python2.7           0672c3ecc763        10
days ago          763MB
lambci/lambda       nodejs8.10          3d20c119c61d        10
days ago          813MB
ubuntu              latest              d70eaf7277ea        11
days ago          72.9MB
```

Cloud9 Task 3: Running a Docker Container

In this section, you will launch a container from the Docker image that you built.

33. To launch a container, enter the following command:

```
docker run --name webapp -d -p 80:80 webapp-image
```

This command requests Docker to run a container, with the name *webapp*, in daemon mode (non-interactive) and map tcp/80 outside the container to tcp/80 on the inside of the container.

34. To see if you container is running, enter the following command:

```
docker ps -a
```

Expected output:

```
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS                    NAMES
4febafa7a43e        webapp-image        "apachectl -D FOREGR…"
15 seconds ago      Up 14 seconds       0.0.0.0:80->80/tcp
webapp
```

35. Next, you will test the website. To do this, use the **Public IPv4 address** that you noted earlier and paste it into a web browser.

For example: http://54.70.180.170

You should see a web page with a galaxy for a background and the following two phrases:

DEVELOPING ON **AWS**

Hi, I'm running your container. I'm very athletic.

Congratulations! You have launched your first container.

Cloud9 Task 4: Interacting with your

Docker Container

In this section, you use various commands to interact with your Docker container.

36. Go back to your **Cloud9** environment tab.

37. Open a bash prompt on your running container by using the following command:

```
docker exec -i -t webapp /bin/bash
```

The **docker exec** command runs a new command in a running container. The flag *i* makes it interactive by redirecting the STDIN.

You should now be logged into your container. This is great for troubleshooting. The prompt will look like the following:

root@container-id:/#

38. Try taking a look at the file system. Look at the */var/www/html* folder to see that your webapp code is now in it. Enter the following command:

```
ls /var/www/html
```

39. To exit from your container, type **exit** and press enter.

40. To view your running containers, enter the following command:

```
docker ps -a
```

Note
Take note of your *container id*. You can use the container id to start and stop

your containers.

41. To stop your container, you can simply enter:

    ```
    docker stop <container-id>
    ```

    or

    ```
    docker stop <name>
    ```

    For example, if your container-id is *4b6c759654cd* and the name is *webapp*, you could stop your container using any of the following commands:

    ```
    docker stop webapp
    ```

    or

    ```
    docker stop 4b6c759654cd
    ```

    or

    ```
    docker stop 4b
    ```

    Note
    The last command works because Docker can see that there is only one container running that has a container id starting with '4b'. This can be useful and faster than using the entire id, particularly for testing.

42. Start your container again. To start the container, enter the following command:

    ```
    docker start webapp
    ```

43. Next, view the logs for your container. To view your container logs, enter the following command:

```
docker logs webapp
```

This is only to show you that you can see the logs of apache. No need to be worried by the logs you are seeing as you already know that your container works when you tested it using your browser in the previous task.

44. List the port mappings for your container. To list the port mappings, enter the following command:

```
docker port webapp
```

This should return the following: *80/tcp -> 0.0.0.0:80*

Cloud9 Task 5: Creating an ECR Repository and Pushing Your Image

Now that you have an image, the next step a Developer would normally do is to send the image into a repository.

In this section, you will be using ECR as your private repository. You will:

- Create a repository.
- Tag your image with that new repository.
- Authenticate the docker client to your ECR repository.
- Push your image in that repository.

45. The first step is to verify the region you are in. Create the ECR repository called 'webapp'. To create the repository:

    - Paste `aws ecr create-repository --repository-name webapp --region R`
    - Replace **REGION** with the value of **REGION** located to the left of these instructions.
    - Press **Enter**

```
{
    "repository": {
        "repositoryUri": "012345678901.dkr.ecr.us-east-
1.amazonaws.com/webapp",
        "registryId": "012345678901",
        "imageTagMutability": "MUTABLE",
        "repositoryArn": "arn:aws:ecr:us-east-
1:401983724912:repository/webapp",
        "repositoryName": "webapp",
        "createdAt": 1572555829.0
    }
}
```

    Take note of the **repositoryUri** without the double-quotes as you will use it in the following steps. In this example, the repositoryUri to use in the following steps would be: *012345678901.dkr.ecr.us-east-1.amazonaws.com/webapp*

46. Next, tag the webapp-image ECR repository that you have just created. To tag the image, enter the following command:

```
docker tag webapp-image change_me-repositoryUri
```

    You will need to replace **change_me-repositoryUri** with the **repositoryUri** that you noted in a previous step.

47. To be able to do a push into ECR, the first step is to authenticate the docker client. AWS created an AWS CLI command, *aws ecr get-login*, to simplify this process which returns the *docker login -u ...* command. Instead of having

you copy the return statement and paste it back, you will run the command by using *eval* which will run the command returned by *aws ecr get-login*.

To authenticate the *docker* client, enter the following command:

```
eval $(aws ecr get-login --no-include-email)
```

The output should indicate *Login Succeeded*. You can ignore the Warning as using the *eval* command allowed you to not paste the password on standard input and to be visible in the history.

48. Now that you are logged in, you can push the image into the ECR repository. To push the image, enter the following command:

```
docker push change_me-repositoryUri
```

You will need to replace **change_me-repositoryUri** with the **repositoryUri** you noted in a previous step.

Expected output:

```
The push refers to repository [012345678901.dkr.ecr.us-east-
1.amazonaws.com/webapp]
2fc92baef76f: Pushed
9d66cbb6a6ab: Pushed
268a067217b5: Pushed
c01d74f99de4: Pushed
ccd4d61916aa: Pushed
8f2b771487e9: Pushed
f49017d4d5ce: Pushed
latest: digest:
sha256:d5f7beae13aed16d307629a943f3d687ae30e437db3ee3cf4a565672d2b39
size: 1779
```

Congratulations!

You have successfully created an ECR Repository and pushed your image to it.

End Lab

Follow these steps to close the console, end your lab, and evaluate the experience.

49. Return to the AWS Management Console.

50. On the navigation bar, choose **awsstudent@<AccountNumber>**, and then choose **Sign Out**.

51. Choose End Lab

52. Choose OK

53. (Optional):

- Select the applicable number of stars ☆
- Type a comment
- Choose **Submit**

    - 1 star = Very dissatisfied
    - 2 stars = Dissatisfied
    - 3 stars = Neutral
    - 4 stars = Satisfied
    - 5 stars = Very satisfied

You may close the window if you don't want to provide feedback.
Congratulations! You have successfully created and interacted with Docker

containers by using AWS Cloud9.

Additional Resources

For more information about AWS Training and Certification, see *http://aws.amazon.com/training/*.

Your feedback is welcome and appreciated.

If you would like to share any feedback, suggestions, or corrections, please provide the details in our *AWS Training and Certification Contact Form*.

Completing the Lab By Connecting to Your Amazon EC2 Linux Instance

For this lab, you connect to an Amazon EC2 Linux instance for this lab. All of the steps for this lab are completed in this section for working with Docker containers.

The first step is to connect to your Linux Dev instance. On the Linux EC2 instance, you will also have access to Vi and Nano text editors. To connect to the Linux EC2 instance, see the following directions:

- Connect to Linux Dev Instance from a Windows Machine
- Connect to Linux Dev Instance from a macOs Machine

Connect to Linux Dev Instance from a Windows Machine

Access to the Dev Linux instance that is used in your labs requires a secure connection by using an SSH client. The following instructions walk through the process of connecting to Amazon EC2 Linux instances.

Note: Perform the steps in this task only if you are connecting to **Linux Dev Instance** from a Windows machine.

54. In the **Connection Details** section in the lab console, under **EC2 Key Pair Private Key**, to connect to Amazon EC2 Linux select **Download PPK** .

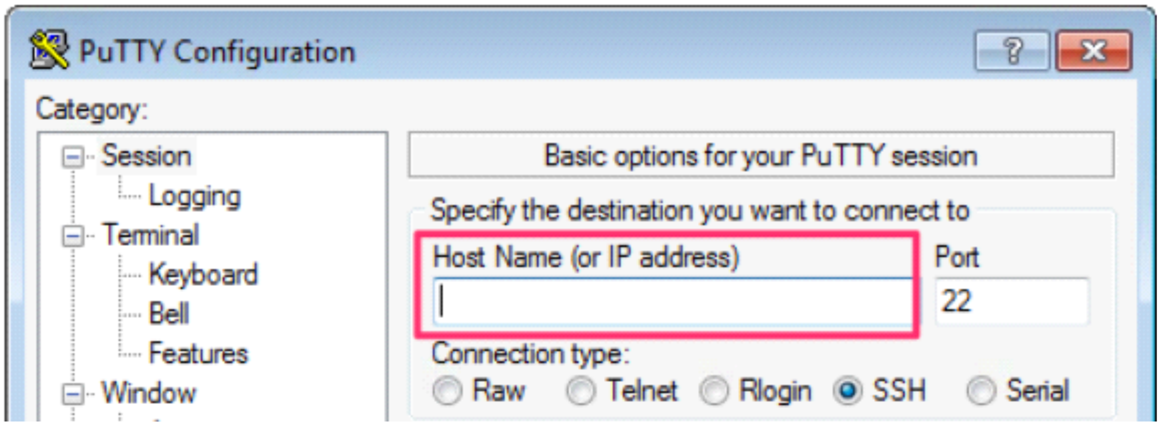
55. Go to the bottom of the **Connection Details** section in the lab console. Copy the **LinuxInstanceIP** to the clipboard.

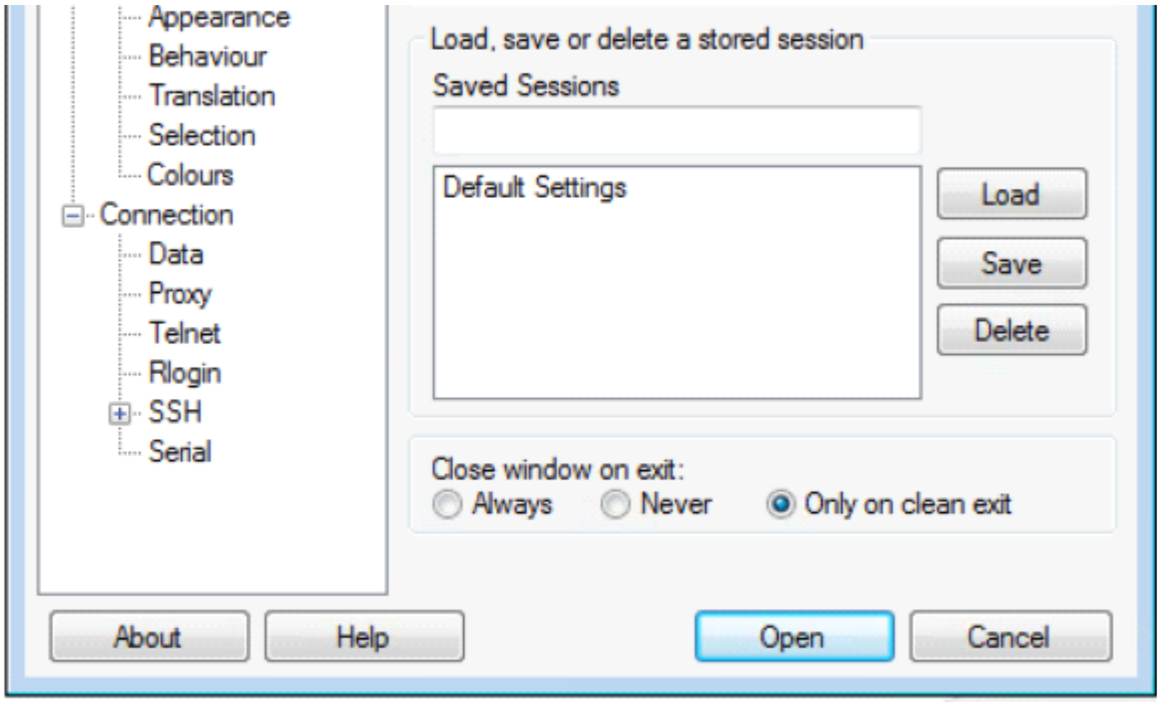
56. If it is not already installed, download the **PuTTY.exe** client to a folder of your choice from the following URL: PuTTY

57. Open **PuTTY.exe**.

58. In the *Basic options for your PuTTY Session* pane, for **Host Name (or IP address)**, type `ec2-user@<ip-address>`, where `<ip-address>` is the **LinuxInstanceIP** of your Amazon EC2 instance that you copied earlier.

    This will log you in to the remote server as the user `ec2-user`, which is the default user on Amazon Linux on EC2.





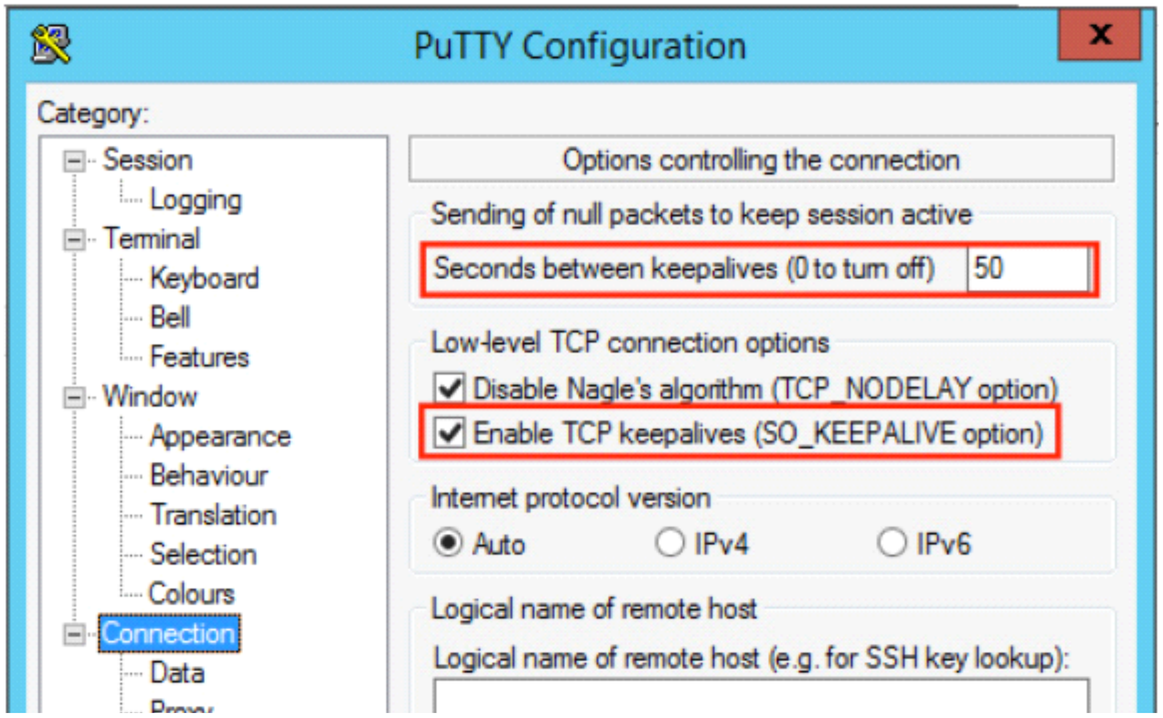
Default Settings

Close window on exit:
( ) Always ( ) Never [✓ Only on clean exit]

59. In the left navigation menu, select **Connection**.

60. Set the following options to keep your SSH connection active as you complete the lab:

- For **Seconds between keepalives (0 to turn off)**, **set the value** to `50`.
- For **Enable TCP keepalives (SO_KEEPALIVE option)**, enable the checkbox.

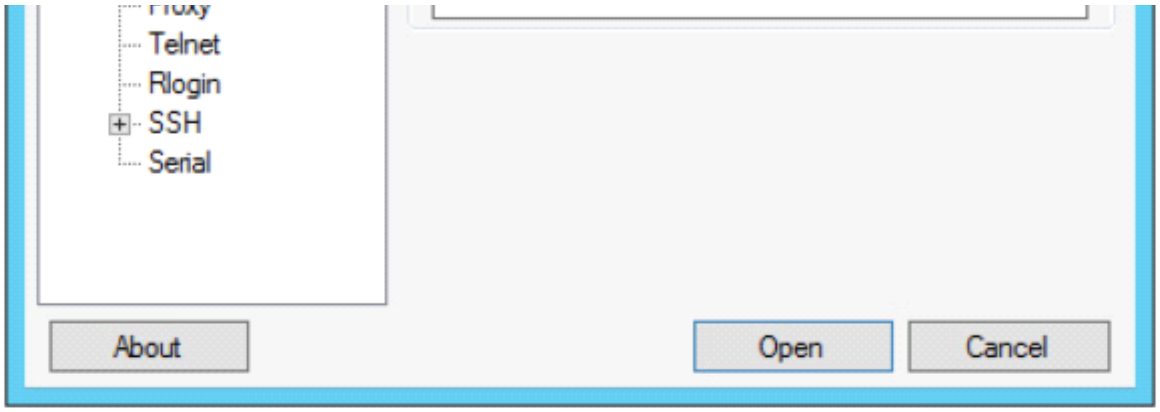

PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Options controlling the connection
Sending of null packets to keep session active
Low-level TCP connection options
Internet protocol version
Logical name of remote host
Logical name of remote host (e.g. for SSH key lookup):

Seconds between keepalives (0 to turn off) 50

[✓] Disable Nagle's algorithm (TCP_NODELAY option)

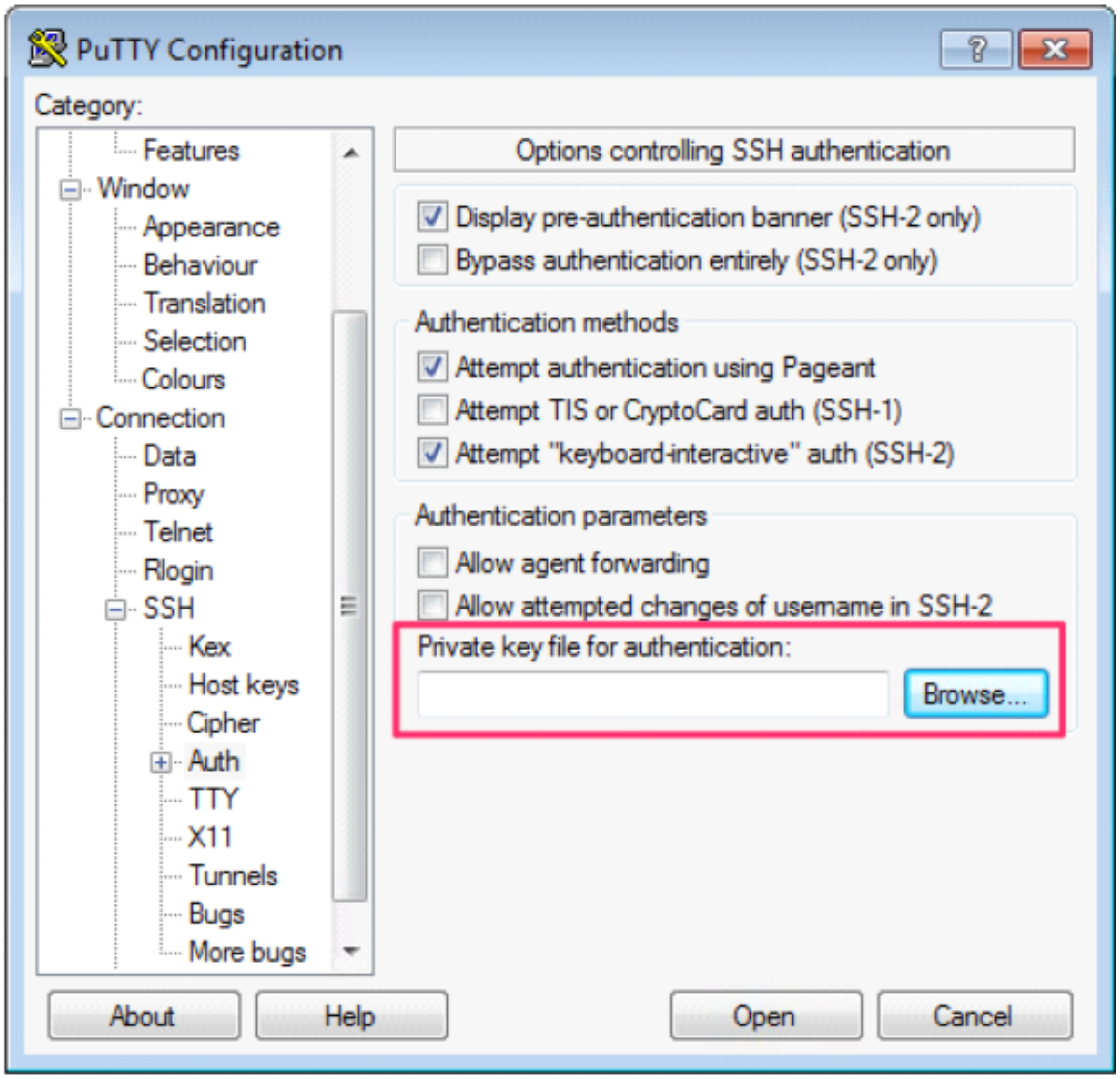
[✓] Enable TCP keepalives (SO_KEEPALIVE option)

[✓ Auto] ( ) IPv4 ( ) IPv6



61. In the left navigation menu, select **Connection -> SSH -> Auth**.

62. Select **Browse** in the section for **Private key file for authentication**.


PuTTY Configuration

Category:

Options controlling SSH authentication

- [✓] Display pre-authentication banner (SSH-2 only)
- [ ] Bypass authentication entirely (SSH-2 only)

Authentication methods

- [✓] Attempt authentication using Pageant
- [ ] Attempt TIS or CryptoCard auth (SSH-1)
- [✓] Attempt "keyboard-interactive" auth (SSH-2)

Authentication parameters

- [ ] Allow agent forwarding
- [ ] Allow attempted changes of username in SSH-2

Private key file for authentication:

63. Select the PPK file you downloaded earlier and then, select **Open**.

This file is usually located in the *Downloads* folder on your PC.

64. Select **Open** to initiate the remote session.

65. PuTTY will ask whether you wish to cache the server's host key. Select **Yes**.

    Result

    You are now connected to **Linux Dev Instance**. When lab instructions in subsequent sections require a command window, continue using your SSH terminal window.

    To continue this lab, move on to Task 2-Linux: Creating a Docker Image.

Connect to Linux Dev Instance from a Linux or macOS Machine

In this section, you will connect to a Linux EC2 instance from your Linux or macOS machine using SSH.

Note
Perform the steps in this task only if you are connecting to **Linux Dev Instance** from a Linux or macOS machine.

66. In the **Connection Details** section in the lab console, under **EC2 Key Pair Private Key**, to connect to Amazon EC2 Linux select **Download PEM** .

67. Go to the bottom of the **Connection Details** section in the lab console. Copy the **LinuxInstanceIP** to the clipboard.

68. Open the **Terminal** application.

    Complete the remaining connection steps in the terminal window.

69. Change directory to the folder where the PEM fille has been downloaded (ie `cd ~/Downloads`)

70. Change the permissions on the PEM file using the command below.

    Replace `<pem-file>` with the name of the PEM file you downloaded.

    ```
    chmod 400 <pem-file>
    ```

71. Use the command below to log in to the remote instance as user `ec2-user` .

    Replace `<pem-file>` with the name of the PEM file you downloaded and **replace** `<ip-address>` with the IP address of your EC2 instance that you copied to the clipboard.

    ```
    ssh -i <pem-file> ec2-user@<ip-address>
    ```

    Result

    You are now connected to **Linux Dev Instance**. When lab instructions in subsequent sections require a command window, continue using your SSH terminal window.

    To continue this lab, move on to Task 2-Linux: Creating a Docker Image.

Linux Task 2: Creating a Docker Image

Docker is already installed on the EC2 instance. You can verify that Docker is installed and see the version information.

Now, the next step is to create your first Docker image.

72. To verify that Docker is installed, enter the following command:

```
docker -v
```

You should see a response similar to the following:

Docker version 18.06.1-ce, build e68fc7a215d7133c34aa18e3b72b4a21fd0c6136

73. To create your Docker image, enter the following commands:

- Make a directory called 'webapp' and change into it:

```
mkdir webapp && cd ./webapp
```

- Download a static website from S3:

```
wget http://us-west-2-tcprod.s3.amazonaws.com/courses/ILT-TF-100-
DODEVA/v3.3.10/lab-6-docker/scripts/website.zip
```

- Unzip your static website:

```
unzip website.zip && rm -f website.zip
```

- Change back into the parent directory:

```
cd ..
```

- Create a new Dockerfile by using nano. To create the file, enter the following command:

```
nano dockerfile
```

Note

You can use any editor of choice, such as vi.

- Copy and paste the following into your dockerfile:

```
FROM ubuntu

ENV TZ=Europe/Minsk
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ >
/etc/timezone

# Install apache and remove the list of packages downloaded from
apt-get update
RUN apt-get update -y && \
apt-get install -y apache2 && \
rm -r /var/lib/apt/lists/*

# Copy the website into the apache web root directory
COPY webapp /var/www/html

EXPOSE 80

CMD ["apachectl", "-D", "FOREGROUND"]
```

- **Save** your file and quit the editor.

This is an example of a Dockerfile. Your dockerfile does the following:

- Downloads the image ubuntu from an image repository.
- Installs apache and removes the packages downloaded from apt-get update.
- Copies your web application into the image.
- Exposes tcp/80 to allow HTTP connections inbound.
- Starts apache.

74. The next step is to build an image for your application. To build an image, enter the following command:

```
docker build -t webapp-image
docker build -t webapp-image .
```

This command builds an image from a Dockerfile located in '.' (the current directory). Then, it will tag the image with a name *webapp*.

75. You should now have a Docker image. You can verify your image by entering the following command:

```
docker images
```

You should a list showing a couple of images.

Linux Task 3: Running a Docker Container

In this section, you will launch a container from the Docker image that you built.

76. To launch a container, enter the following command:

```
docker run --name webapp -d -p 80:80 webapp-image
```

This command requests Docker to run a container, with the name *webapp*, in daemon mode (non-interactive) and map tcp/80 outside the container to tcp/80 on the inside of the container.

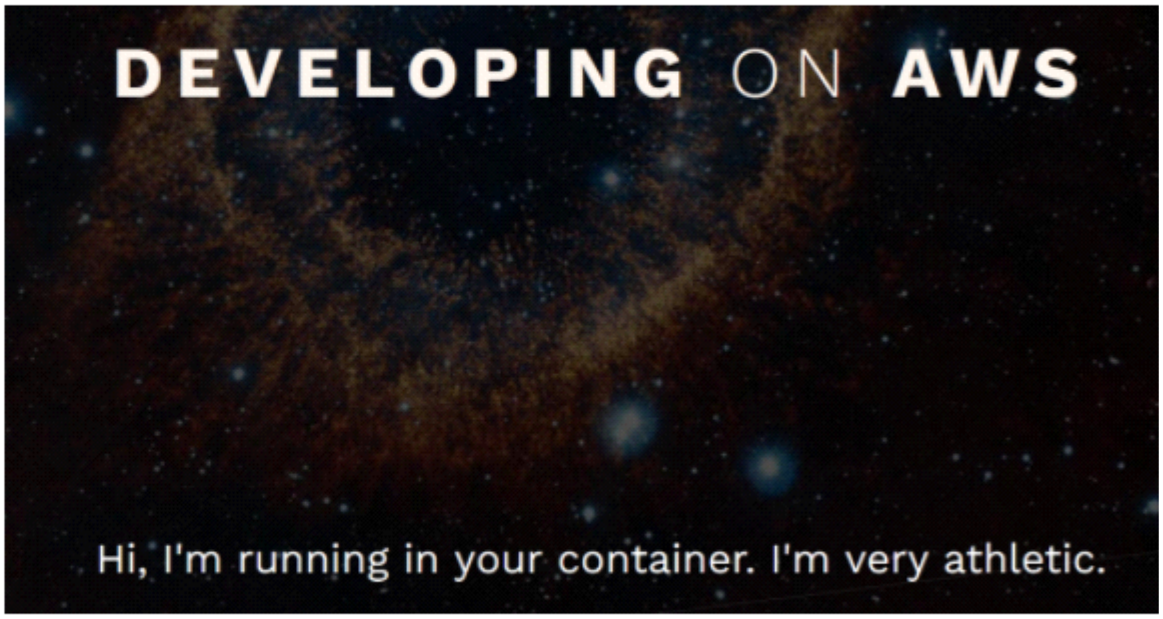
77. To see if you container is running, enter the following command:

```
docker ps -a
```

78. Next, you will need to test the website. To do this, go to the bottom of the **Connection Details** section in the lab console. Copy the **LinuxInstanceIP** to the clipboard and paste it into a web browser.

    For example: http://xxx.xxx.xxx.xxx

    You should see a web page similar to the following:


DEVELOPING ON AWS

Hi, I'm running in your container. I'm very athletic.

Congratulations! You have launched your first container.

Linux Task 4: Interacting with your Docker Container

In this section, you use various commands to interact with your Docker container.

79. Open a bash prompt on your running container by using the following

79. Open a bash prompt on your running container by using the following command:

```
docker exec -i -t webapp /bin/bash
```

The **docker exec** command runs a new command in a running container. The flag *i* makes it interactive by redirecting the STDIN.

You should now be logged into your container. For troubleshooting, this is as great as you can get it! The prompt will look like the following:

root@container-id:/#

80. Look at the */var/www/html* folder to see that your webapp code is now in it. Enter the following command:

```
ls /var/www/html
```

81. To exit from your container, type **exit** and press enter.

82. To view your running containers, enter the following command:

```
docker ps -a
```

Note
Take note of your *container id*. You can use the container id to start and stop your containers.

83. To stop your container, you can simply enter:

```
docker stop <container-id>
```

or

```
docker stop <name>
```

For example, if your container-id is *4b6c759654cd* and the name is *webapp*, you could stop your container using any of the following commands:

```
docker stop webapp
```

or

```
docker stop 4b6c759654cd
```

or

```
docker stop 4b
```

Note
The last command works because Docker can see that there is only one container running that has a container id starting with '4b'. This can be useful and faster than using the entire id, particularly for testing.

84. Start your container again. To start the container, enter the following command:

```
docker start webapp
```

85. Next, view the logs for your container. To view your container logs, enter the following command:

```
docker logs webapp
```

This is only to show you that you can see the logs of apache. No need to be worried by the logs you are seeing as you already know that your container works when you tested it using your browser in the previous task.

86. List the port mappings for your container. To list the port mappings, enter the following command:

```
docker port webapp
```

This should return the following: *80/tcp -> 0.0.0.0:80*

Congratulations! You have successfully created and interacted with Docker containers.

Linux Task 5: Creating an ECR Repository and Pushing Your Image

Now that you have an image, the next step a Developer would normally do is to send the image into a repository.

In this section, you will be using ECR as your private repository. You will:

- Create a repository.
- Tag your image with that new repository.
- Authenticate the docker client to your ECR repository.
- Push your image in that repository.

87. The first step is to create the ECR repository called 'webapp'. To create the repository, enter the following command:

```
aws ecr create-repository --repository-name webapp
```

The return value should be similar to the following:

```
{
    "repository": {
        "registryId": "012345678901",
        "repositoryName": "webapp",
        "repositoryArn": "arn:aws:ecr:us-east-
1:012345678901:repository/webapp",
        "createdAt": 1533951996.0,
        "repositoryUri": "012345678901.dkr.ecr.us-east-
1.amazonaws.com/webapp"
    }
}
```

Take note of the **repositoryUri** without the double-quotes as you will use it in the following steps. In this example, the repositoryUri to use in the following steps would be: *012345678901.dkr.ecr.us-east-1.amazonaws.com/webapp*

88. Next, tag the webapp-image ECR repository that you have just created. To tag the image, enter the following command:

```
docker tag webapp-image change_me-repositoryUri
```

You will need to replace **change_me-repositoryUri** with the **repositoryUri** that you noted in a previous step.

89. To be able to do a push into ECR, the first step is to authenticate the docker client. AWS created an AWS CLI command, *aws ecr get-login*, to simplify this process which returns the *docker login -u ...* command. Instead of having you copy the return statement and paste it back, you will run the command by using *eval* which will run the command returned by *aws ecr get-login*.

To authenticate the *docker* client, enter the following command:

```
eval $(aws ecr get-login --no-include-email)
```

The output should indicate *Login Succeeded*. You can ignore the Warning as using the *eval* command allowed you to not paste the password on standard input and to be visible in the history.

90. Now that you are logged in, you can push the image into the ECR repository. To push the image, enter the following command:

    ```
    docker push change_me-repositoryUri
    ```

    You will need to replace **change_me-repositoryUri** with the **repositoryUri** you noted in a previous step.

    The output of this command should look similar to the following:

    ```
    The push refers to repository [012345678901.dkr.ecr.us-east-
    1.amazonaws.com/webapp]
    2fc92baef76f: Pushed
    9d66cbb6a6ab: Pushed
    268a067217b5: Pushed
    c01d74f99de4: Pushed
    ccd4d61916aa: Pushed
    8f2b771487e9: Pushed
    f49017d4d5ce: Pushed
    latest: digest:
    sha256:d5f7beae13aed16d307629a943f3d687ae30e437db3ee3cf4a565672d2b39
    size: 1779
    ```

    Congratulations! You have successfully created an ECR Repository and pushed your image to it.

End Lab

Follow these steps to close the console, end your lab, and evaluate the experience.

91. Return to the AWS Management Console.

92. On the navigation bar, choose **awsstudent@<AccountNumber>**, and then choose **Sign Out**.

93. Choose **End Lab**

94. Choose **OK**

95. (Optional):

- Select the applicable number of stars ☆
- Type a comment
- Choose **Submit**

    - 1 star = Very dissatisfied
    - 2 stars = Dissatisfied
    - 3 stars = Neutral
    - 4 stars = Satisfied
    - 5 stars = Very satisfied

You may close the window if you don't want to provide feedback.

Congratulations! You have successfully created and interacted with Docker containers by using Amazon EC2.
For more information about AWS Training and Certification, see *http://aws.amazon.com/training/*.

Your feedback is welcome and appreciated.

If you would like to share any feedback, suggestions, or corrections, please provide the details in our *AWS Training and Certification Contact Form*.