Docker Basics

About the Author

Mamta Jha (Cloud Solutions Architect, CloudThat Technologies) has more than a decade of experience in Telecom domain and has been working in Cloud domain since last four years. She has worked on first ever Cloud RNC (WCE – Wireless Cloud Element). She was the part of the pilot project of Alcatel-Lucent in moving the entire wireless domain to cloud.

She is an expert in Configuration Management with Chef and Ansible. OpenSource Technologies and Linux is her passion. She has around four years of experience in deploying and managing OpenStack infrastructure primarily on Red Hat Platform, Centos, and Ubuntu. She has worked with organizations on setting up their complete private cloud on OpenStack using PackStack / TripleO OpenStack deployment methods. And even integrating Ceph as object, block and file storage as one unified system for OpenStack.

Mamta is currently working on architecting cloud strategy for enterprises, which includes latest cloud technologies like Chef, Ansible, Docker, Kubernetes, OpenShift, etc. She is tech-savvy and has attained several certifications. She is Red Hat Certified System Admin in RHEL7, Red Hat Certified Engineer in RHEL7, Red Hat Certified System Administrator in OpenStack, Red Hat Certified Engineer – OpenStack and Red Hat Certificate of Expertise in PaaS (OpenShift).

Table of Contents

Lab 1: Launching Container with Ubuntu Image from Docker Hub Public Images

Topics

- Launch a Virtual Machine and install Docker on an Ubuntu 16.04 Machine
- Creating a Docker container and installing packages
- Port mapping and volume mapping while creating the container

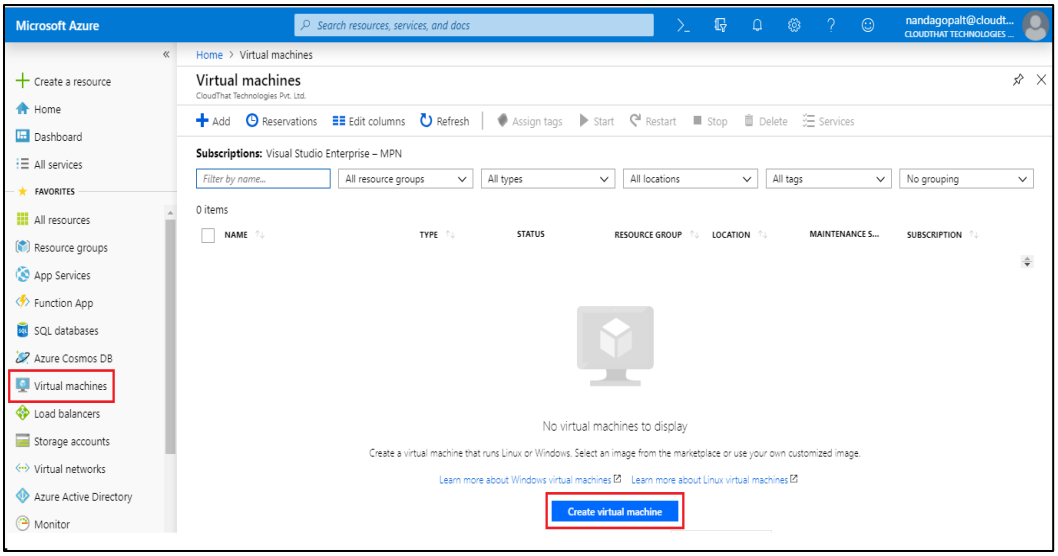
Task 1: Launch a Virtual Machine and Install Docker on Ubuntu 16.04

In this task, you will launch an VM using **Ubuntu 16.04** from the Classic Wizard.

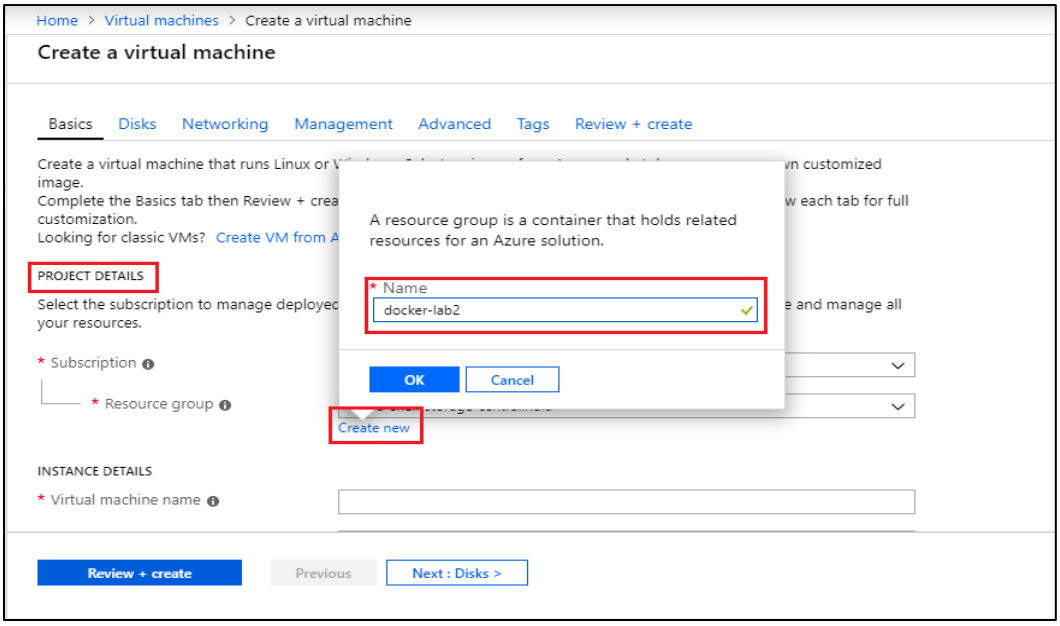
1. Open a web browser and enter https://portal.azure.com in the address bar and log in to your account

2. Click Virtual Machines

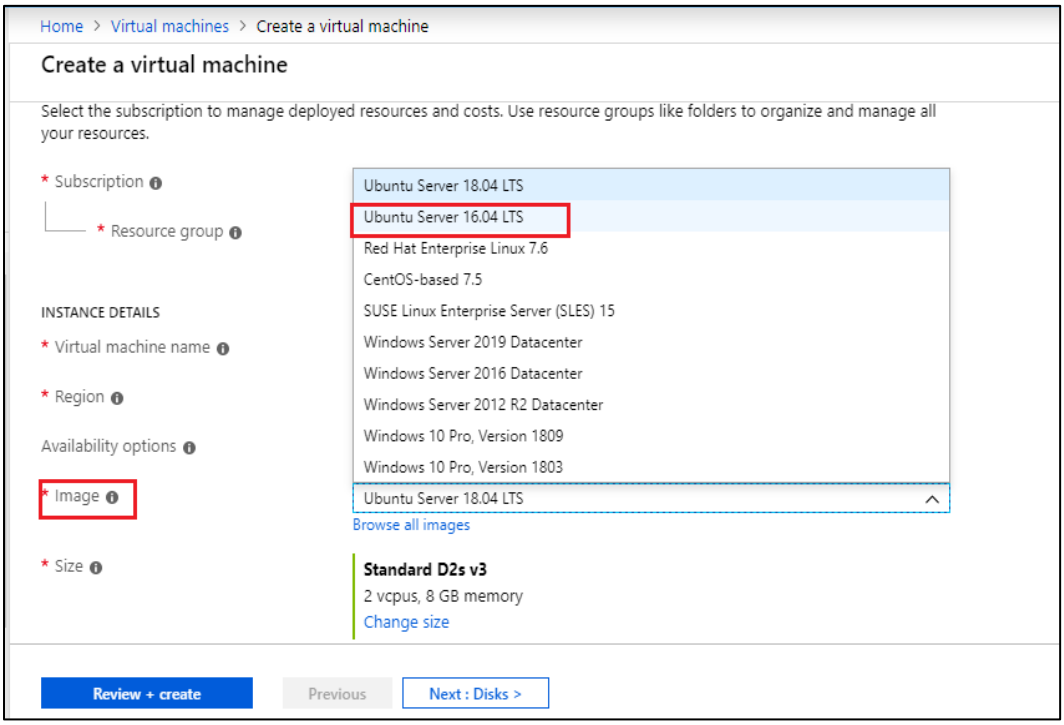
3. Click **Create Virtual Machines** under Virtual Machines section



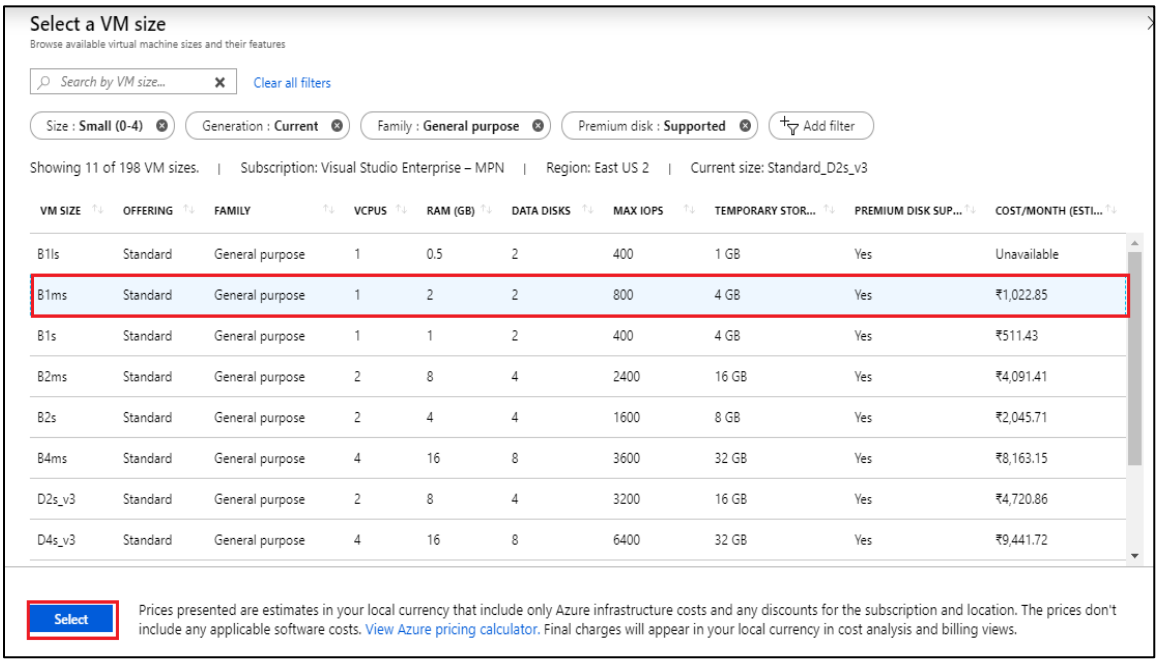
4. Click **Create New** under Project Details and create a separate resource group for this lab



5. Scroll down to Instance details and type the VM name as **manager** and select the image as **Ubuntu 16.04** from the drop-down list

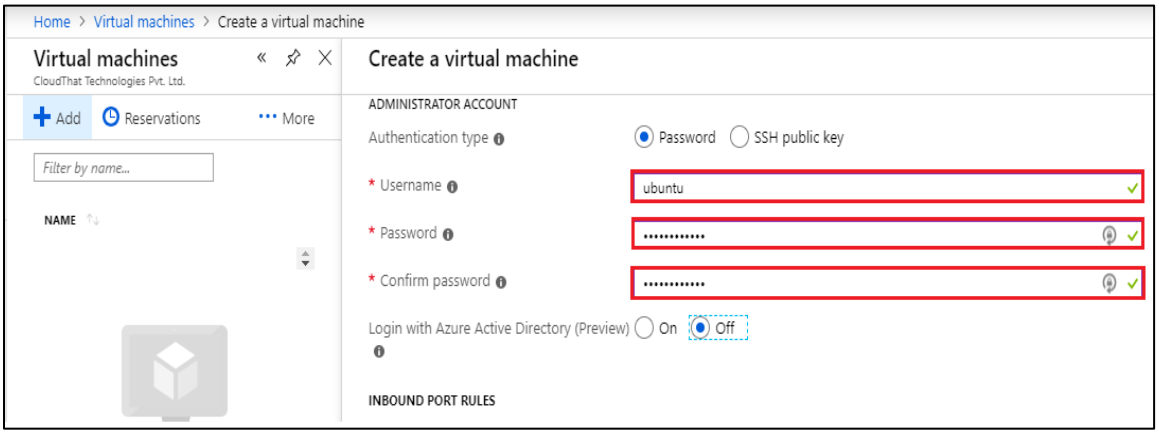


6. Choose an Instance Size and ensure that **VM Size** is selected as **B1ms**

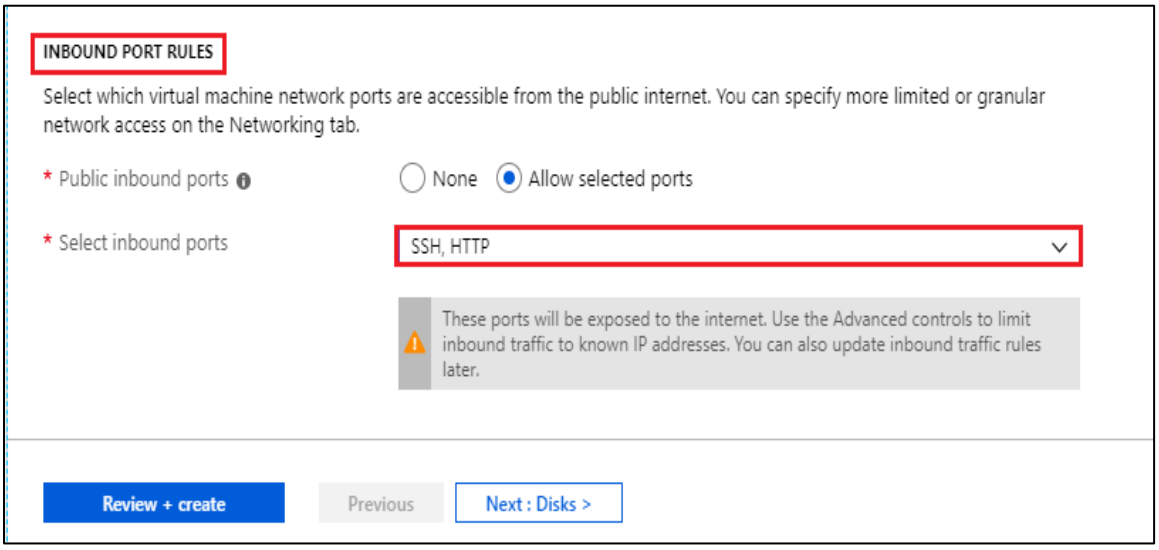


VM SIZE	OFFERING	FAMILY	VCPUS	RAM (GB)	DATA DISKS	MAX IOPS	TEMPORARY STOR...	PREMIUM DISK SUP...	COST/MONTH (ESTI...
B1ls	Standard	General purpose	1	0.5	2	400	1 GB	Yes	Unavailable
B1ms	Standard	General purpose	1	2	2	800	4 GB	Yes	₹1,022.85
B1s	Standard	General purpose	1	1	2	400	4 GB	Yes	₹511.43
B2ms	Standard	General purpose	2	8	4	2400	16 GB	Yes	₹4,091.41
B2s	Standard	General purpose	2	4	4	1600	8 GB	Yes	₹2,045.71
B4ms	Standard	General purpose	4	16	8	3600	32 GB	Yes	₹8,163.15
D2s_v3	Standard	General purpose	2	8	4	3200	16 GB	Yes	₹4,720.86
D4s_v3	Standard	General purpose	4	16	8	6400	32 GB	Yes	₹9,441.72

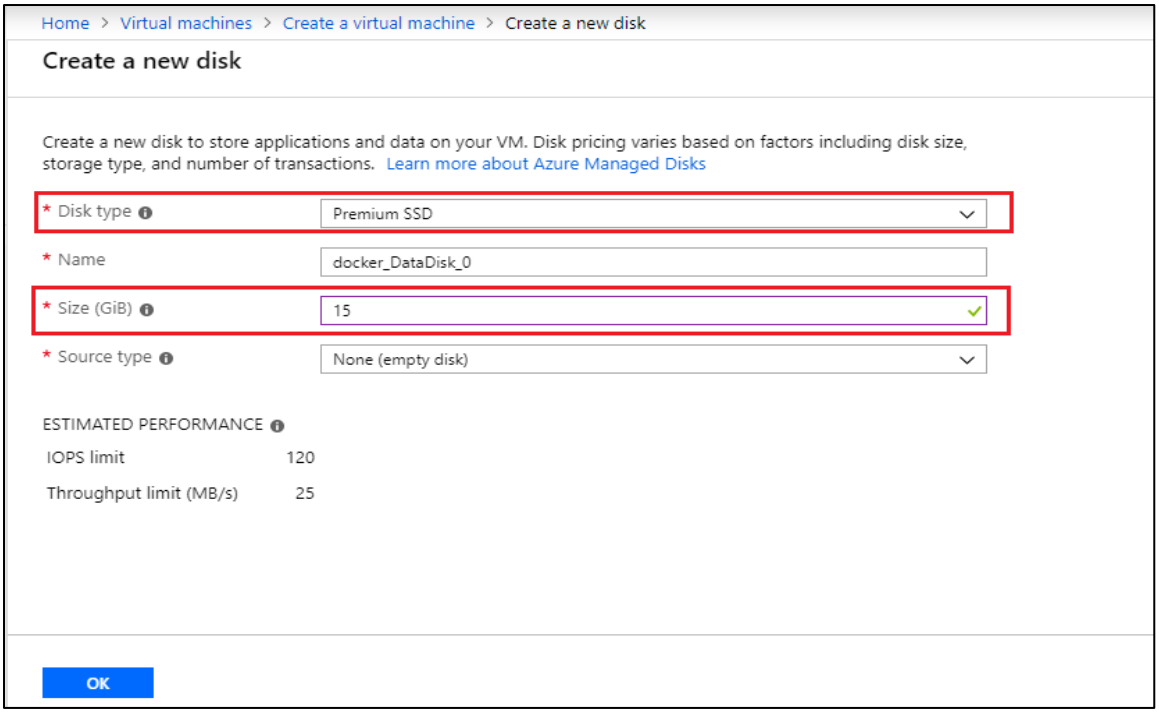
7. Create a sudo account for **Administrator** by typing a username as **ubuntu** and password as **Admin123456!**



8. Scroll Down and Open the ports for **SSH** and **HTTP**

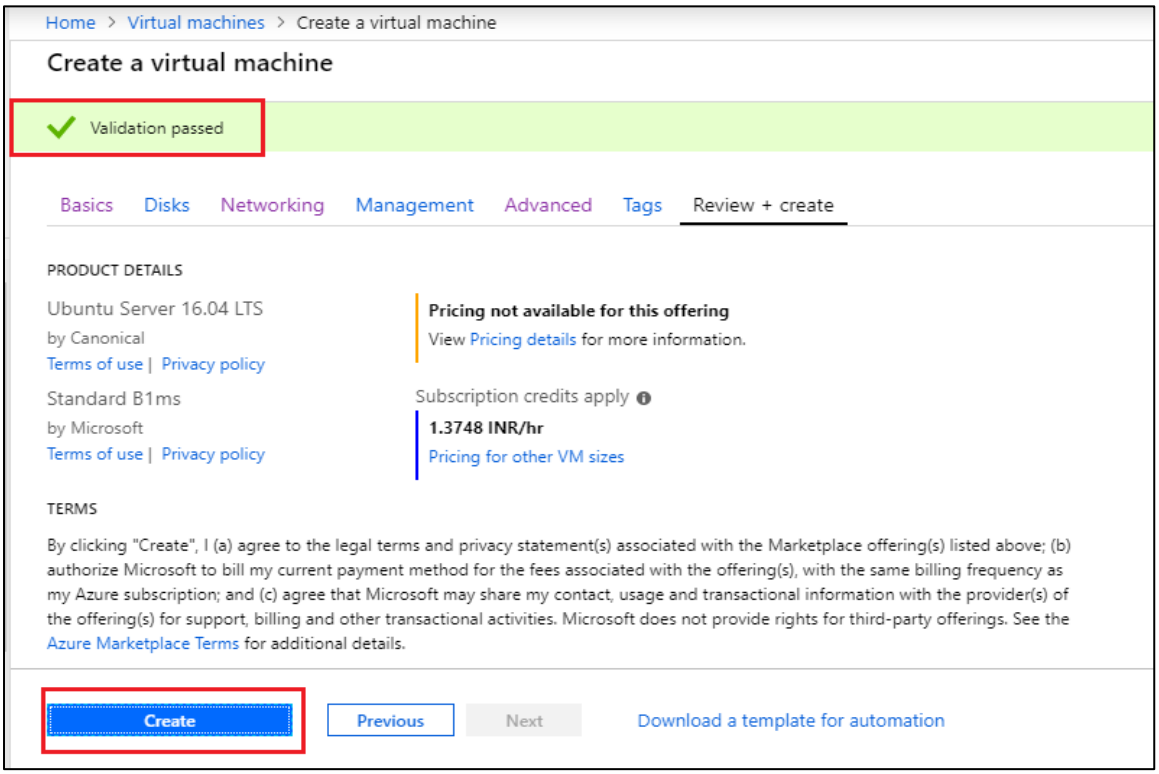


9. Click **Next**: **Disks**, add a Disk of 15 GiB by clicking **Create and Attach a New Disk**. Select the appropriate Disk type and add the value for the Disk

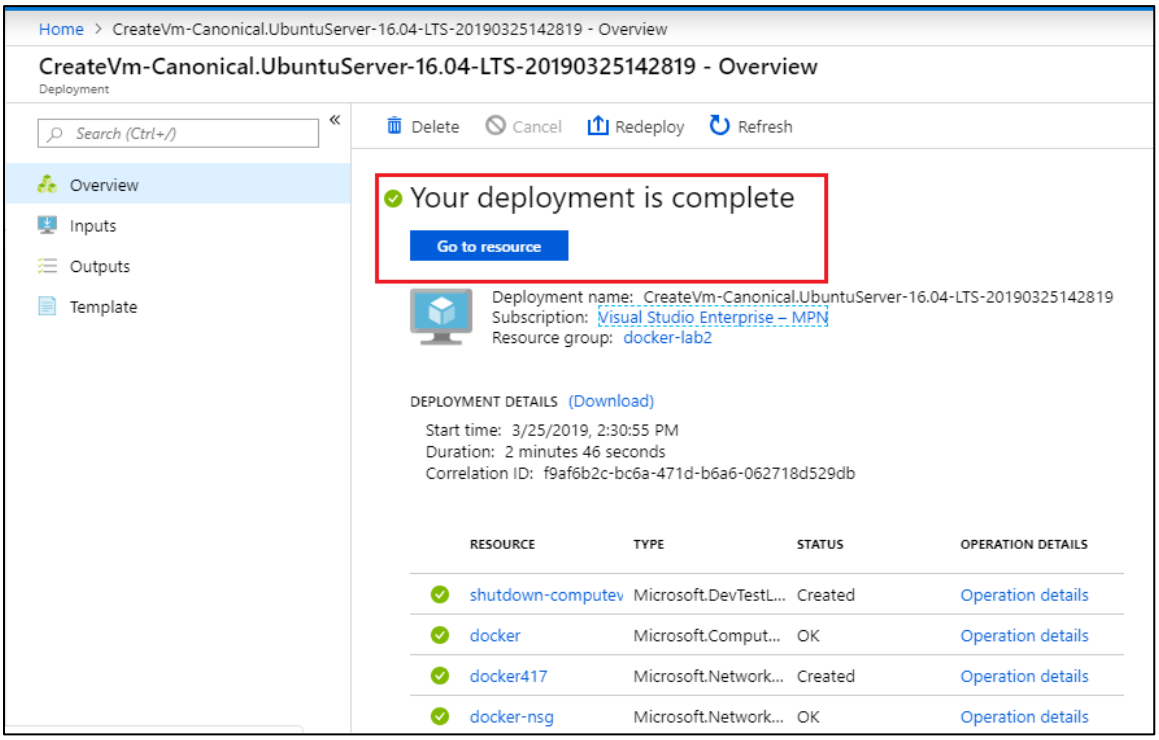


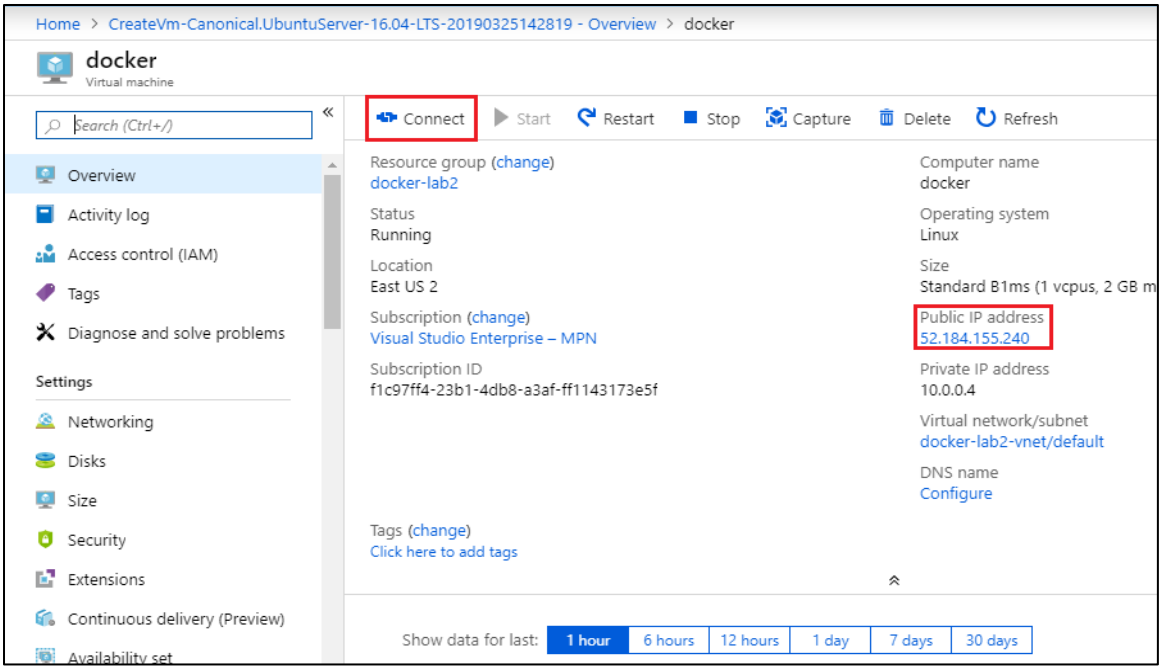
10. Skip the two tabs **Networking, Management, Advanced, Tags** by clicking Next

11. Wait for the **Validation** to be passed and after that click **Create**



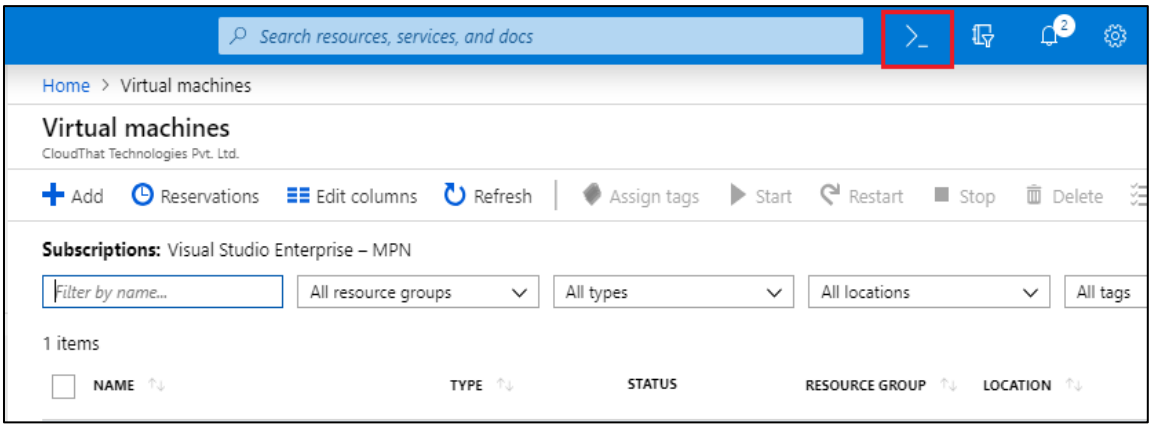
12. The Launch Wizard displays the message that your machine is now launched. Click **Go to Resource** and login to the instance using SSH





13. SSH into the VM using **CloudConsole** with **Public IP**. Use **Connect** from the console to get the command in-order to use in any Linux Terminal

14. Click the **CloudConsole** button on the top right corner and SSH into the VM



```
[redacted]:~$ ssh ubuntu@104.46.99.181
ubuntu@104.46.99.181's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1041-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud


To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@manager:~$
```

a) Install Docker using the following command

```
$ sudo apt-get update && sudo curl -SSL https://get.docker.com/
| sh
```

b) Start the Docker service and test the installation

```
$ sudo service docker start
```

c) The Docker daemon binds to a UNIX socket, which is owned by root and other users, needs to use sudo to access it. To avoid using sudo every time, adding user to docker group , using the following command **, logout form cloudshell & login back.**

```
$ sudo usermod -aG docker ubuntu
```

Task 2: Creating a Docker Container by Downloading a Public Ubuntu Image from the Docker Public Registry

Now as the Docker is installed on your VM, let's create a Docker container of an existing image of ubuntu:14.04 and then install apache on it.

1. Run a Docker container from an existing ubuntu image via Docker hub public registry. Use the following command

```
docker run -it --name ubuntu -p 80:80 ubuntu:14.04 bash
```

Note: The options -it is to get into the interactive shell and pseudo-TTY for the bash, and -p is for port mapping for apache web server and --name is used to give a name to the container.

Ubuntu Docker container is started

```
ubuntu@manager:~$ docker run -it --name ubuntu -p 80:80 ubuntu:14.04 bash
Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
e082d4499130: Pull complete
371450624c9e: Pull complete
c8a555b3a57c: Pull complete
1456d810d42e: Pull complete
Digest: sha256:6612de24437f6f01d6a2988ed9a36b3603df06e8d2c0493678f3ee696bc4bb2d
Status: Downloaded newer image for ubuntu:14.04
root@7a2e43ab51db:/#
```

Note: The Docker image was not found locally, so it was pulled from the Docker hub registry.

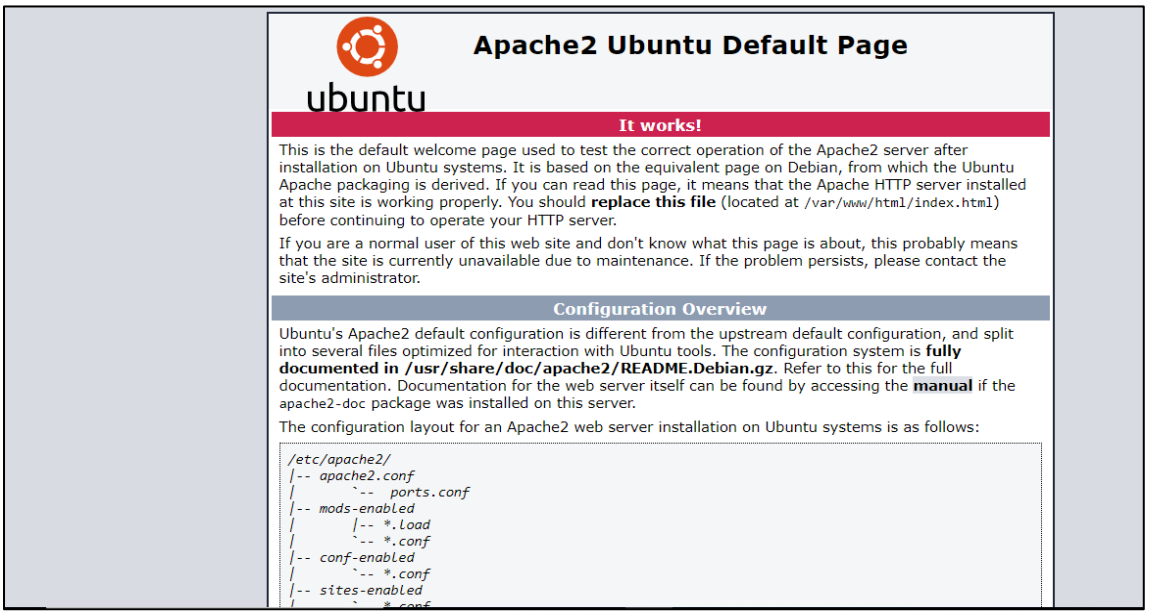
2. Now you are inside the container. Now let us install apache2 inside the Docker container using the following commands

```
# apt-get update -y
# apt-get install apache2 -y
```

3. Once Apache is installed, start the apache service inside the Docker container using the following command

```
# service apache2 start
```

4. Now go to your browser, paste the VM machine's public DNS, you should be able to see something like this



5. Log out of the Ubuntu container

```
# exit
```

6. Find the Ubuntu container id by listing the containers and note the container id. Use the id in the next command

```
$ docker ps -a
CONTAINER ID                    IMAGE                COMMAND
CREATED              STATUS                         PORTS
NAMES
d654738afc84         ubuntu:14.04        "bash"
17 minutes ago       Exited (127) 3 seconds ago
ubuntu
```

7. Remove the containers using the below command. Replace the container id below with the actual container id you obtained from the previous command

```
$ docker rm -vf <container_id>
```

Task 3: Starting Docker Containers with Volume Mapping such that the Container Shares the Host Machine Volume

Now let us start another Docker container with volume mapping and share the host machine volume with the container.

1. Create a directory **"share"** in home directory of your host machine

```
$ mkdir /home/ubuntu/share
```

2. Start a docker container with volume mapped. The directory "/var/www/html" inside the docker container will be mapped to the directory "/home/ubuntu/share" on the host machine

3. Use the following command to achieve the task

```
$ docker run -it -p 80:80 -v /home/ubuntu/share:/var/www/html -
-name container1 ubuntu:14.04 /bin/bash
```

Note: The above command will create a Docker container with name container1 and volume mapped.

4. We will install apache2 inside the Docker container, as we did in the previous task, using the following commands

```
# apt-get update -y
# apt-get install apache2 -y
```

5. Once Apache is installed, start the apache service inside the Docker container using the following command

```
# service apache2 start
```

6. Paste the public DNS URL of the VM in a browser and make sure the default page is loading

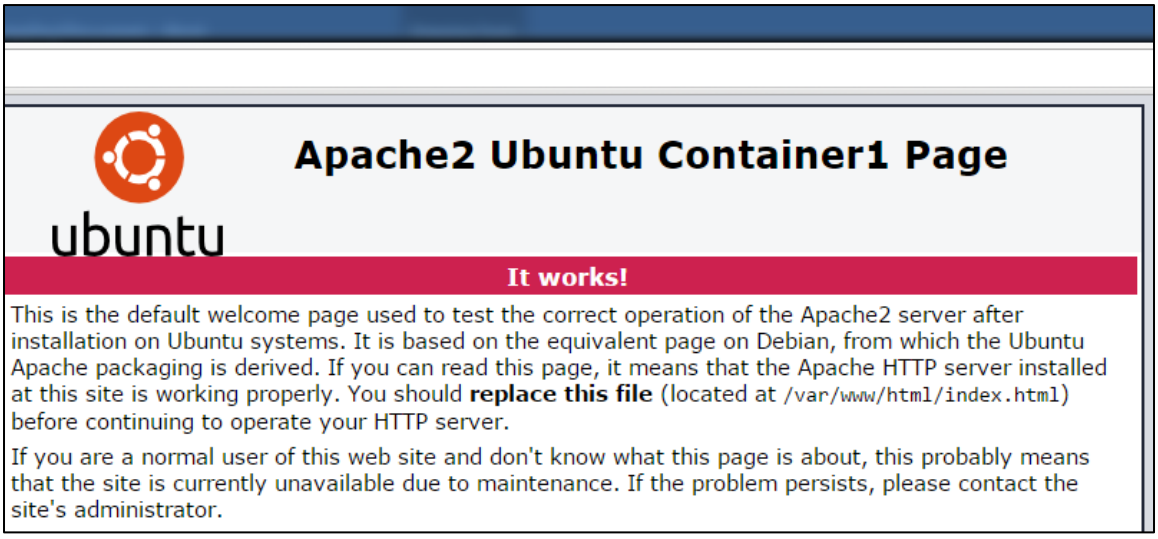
7. Install vim in the container1 and then open the /var/www/html/index.html file inside the container1 for edit

```
# apt-get install vim -y
# vim /var/www/html/index.html
```

8. At line number 198 change the text from “Apache2 Ubuntu Default Page” to “Apache2 Ubuntu **Container1** Page”

```
  <body>
    <div class="main_page">
      <div class="page_header floating_element">
        <img src="/icons/ubuntu-logo.png" alt="Ubuntu Logo" class="floating_element"/>
        <span class="floating_element">
          Apache2 container1 Ubuntu Default Page
        </span>
      </div>
<!--      <div class="table_of_contents floating_element">
        <div class="section_header section_header_grey">
          TABLE OF CONTENTS
        </div>
        <div class="table_of_contents_item floating_element">
          <a href="#about">About</a>
        </div>
        <div class="table_of_contents_item floating_element">
          <a href="#changes">Changes</a>
        </div>
```

9. Save the file and stay inside the container. Now go to the browser, and put the public DNS of VM, and check that the change is reflected to the web page



10. Now open another terminal and log in to the VM. Let us start another container, which will also be mapped to the same folder on the host machine to which container1 is mapped. Use the following command

```
$ docker run -it -v /home/ubuntu/share:/var/www/html --name
container2 ubuntu:14.04
```

> **Note**: This will create a container2, which is also mapped to the same folder as the container one. But no port mapping is done to the container2, so you cannot access it from the browser.

11. Install vim in the container2 and then open the /var/www/html/index.html file inside the container2 for edit

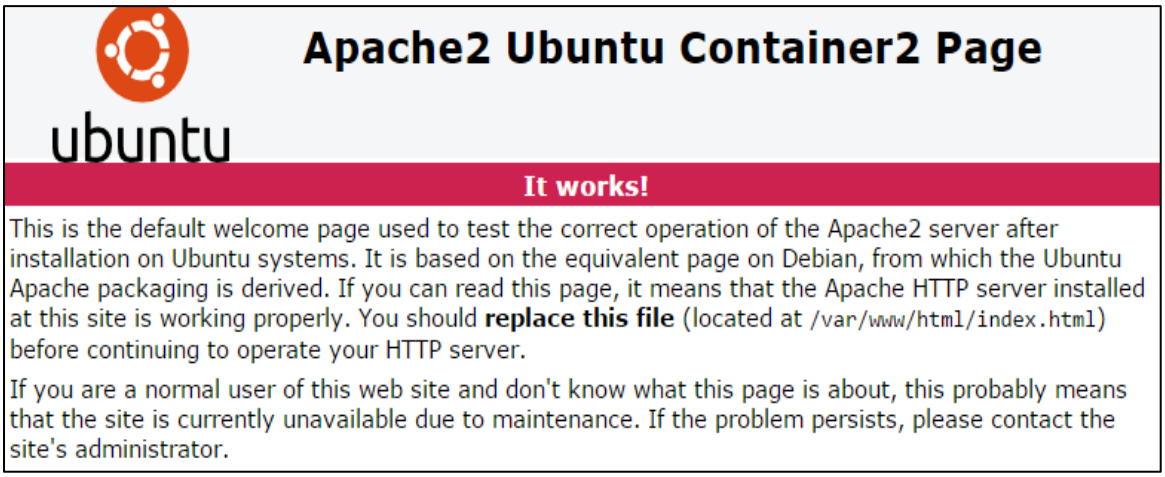
```
# apt-get update -y
# apt-get install vim -y
# vim /var/www/html/index.html
```

12. At line number 198 change the text to “Apache2 Ubuntu **Container1** Page” to “Apache2 Ubuntu **Container2** Page”

```
  <body>
    <div class="main_page">
      <div class="page_header floating_element">
        <img src="/icons/ubuntu-logo.png" alt="Ubuntu Logo" class="floating_element"/>
        <span class="floating_element">
          Apache2 container2 Ubuntu Default Page
        </span>
      </div>
<!--      <div class="table_of_contents floating_element">
        <div class="section_header section_header_grey">
          TABLE OF CONTENTS
        </div>
        <div class="table_of_contents_item floating_element">
          <a href="#about">About</a>
        </div>
        <div class="table_of_contents_item floating_element">
          <a href="#changes">Changes</a>
```

13. We just made changes to the /var/www/html/index.html file, which in turn will change the contents of the index.html file in container1. This is because both the containers are mapped to the same volume of the host

14. Now open your browser and paste the public DNS of the VM. The index.html page of container1 will be accessed which will show something like as below



Note: In the index.html of container1, we had inserted container1, and later we changed it to container2 from the index.html of container2. This shows both the containers are mapped with the same volume.

15. Log out of the Ubuntu container. Find the Ubuntu container id by listing the containers and note the id. Use the id in the next command

```
# exit
```

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS                      PORTS
NAMES
ffe1d735809e        ubuntu:14.04        "/bin/bash"         2
minutes ago       Exited (0) 5 seconds ago
container2
d654738afc84        ubuntu:14.04        "bash"
          3 minutes ago      Exited (127) 3 seconds ago
container1
```

16. Remove the container with the help of command written below

```
$ docker rm -vf container_id
```

Task 4: Creating a new docker volume and inspection

1. For creating a new “docker volume” use the syntax provided below

```
$ docker volume create <volume name>
$ docker volume ls
```

```
ubuntu@manager:~$ docker volume create cloudthat
cloudthat
ubuntu@manager:~$ docker volume ls
DRIVER              VOLUME NAME
local               cloudthat
ubuntu@manager:~$
```

2. Inspecting the Docker Volume

```
$ docker volume inspect <volume name>
```

```
ubuntu@manager:~$ docker volume inspect cloudthat
[
    {
        "CreatedAt": "2019-04-22T07:37:12Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/cloudthat/_data",
        "Name": "cloudthat",
        "Options": {},
        "Scope": "local"
    }
]
ubuntu@manager:~$
```

Task 5: Launching a Nginx container mapped to a specific docker volume and verification

1. Run the command provided below for launching a container named “Nginx” mapped to a specified volume and serving content from that location

```
$ docker run -d -p 80:80 --name=Nginx --mount
source=cloudthat,destination=/usr/share/nginx/html
nginx:latest

$ docker ps
```

```
ubuntu@manager:~$ docker run -d -p 80:80 --name=Nginx --mount source=cloudthat,destination=/usr/share/nginx/html nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
27833a3ba0a5: Pull complete
ea005e36e544: Pull complete
d172c7f0578d: Pull complete
Digest: sha256:e71b1bf4281f25533cf15e6e5f9be4dac74d2328152edf7ecde23abc54e16c1c
Status: Downloaded newer image for nginx:latest
750a1504cafb8e2621fc6ee8f412e05a3378b14f8d1caa4c523c8705a006a2f4
ubuntu@manager:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
750a1504cafb        nginx:latest        "nginx -g 'daemon of…"   5 seconds ago       Up 1 second         0.0.0.0:80->80/tcp   Nginx
ubuntu@manager:~$
```

2. Checking volume attached to the Nginx container

```
$ docker inspect <container name>
```

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "cloudthat",
        "Source": "/var/lib/docker/volumes/cloudthat/_data",
        "Destination": "/usr/share/nginx/html",
        "Driver": "local",
        "Mode": "z",
        "RW": true,
        "Propagation": ""
    }
],
```

3. Checking for contents and verifying mount at “/var/lib/docker/<volume name>/_data”

```
$ sudo su
# ls /var/lib/docker/volumes/<volume name>/_data/
```

```
ubuntu@manager:~$ sudo su
root@manager:/home/ubuntu# ls /var/lib/docker/volumes/cloudthat/_data/
50x.html  index.html
root@manager:/home/ubuntu#
```

4. Now we will use the public IP of our VM to access the Nginx webpage

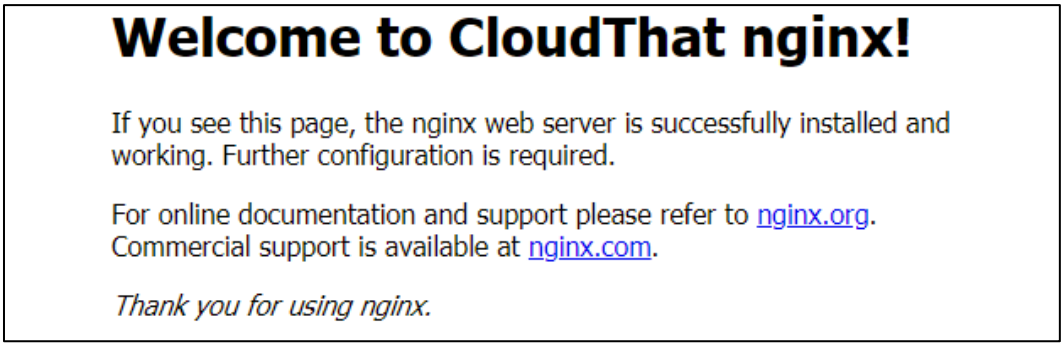
5. Now change the contents on the mapped volume and verify the changes. We will change the headings section in a file named "index.html"

```
$ sudo su
# vi /var/lib/docker/volume/<volume name>/_data/index.html
```

The file will look like this

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to CloudThat nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

6. No again use the public IP of the instance and check for the changes



Task 6: Deleting container and volume

1. Deleting Container and verifying

```
$ docker stop <container name>
$ docker rm <container name>
$ docker ps -a
```

```
root@manager:/home/ubuntu# docker stop Nginx
Nginx
root@manager:/home/ubuntu# docker rm Nginx
Nginx
root@manager:/home/ubuntu# docker ps
CONTAINER ID        IMAGE               COMMAND
root@manager:/home/ubuntu#
```

2. Deleting Volume and verifying

```
$ docker volume ls
$ docker volume rm <volume name>
$ docker volume ls
```

```
root@manager:/home/ubuntu# docker volume ls
DRIVER              VOLUME NAME
local               cloudthat
root@manager:/home/ubuntu# docker volume rm cloudthat
cloudthat
root@manager:/home/ubuntu# docker volume ls
DRIVER              VOLUME NAME
root@manager:/home/ubuntu#
```

Lab 2: Working with Application in Docker

Topics

- Start a container with an Ubuntu image and configure it
- Create a database for WordPress
- Download and configure WordPress
- Create custom WordPress image
- Creating a new docker volume and inspection
- Launching a Nginx container mapped to a specific docker volume and verification
- Deleting container and volume

Task 1: Start a Docker Container with Ubuntu Image from Public Repository and Configure it

1. Let's start a container with fresh Ubuntu 14.04 image and map host port 80 with port 80 of the container by Ubuntu user

```
$ docker run -it --name wp -p 80:80 ubuntu:16.04 /bin/bash
```

2. Now we are inside the container; this command will update the repositories in the container

```
# apt-get update -y
```

3. Here we are installing apache2, mysql, php5 in the **container**

```
# apt-get install -y apache2 mysql-server php php-fpm php-mysql
libapache2-mod-php wget vim
```

Note: While installing MySQL please provide a password for root user.

4. Start the services after installation

```
# service apache2 start
```

```
# service mysql start
```

Task 2: Download and Configure

In this task, we will download and configure the container to run WordPress.

1. Inside **container** login to mysql and create a database for it. The below command will ask for a password. Enter the password you provided during the installation of mysql

```
# mysql -u root -p
```

```
mysql> CREATE DATABASE wordpress;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE USER 'user'@'localhost' IDENTIFIED BY
'qwerty123';
Query OK, 0 rows affected (0.00 sec)
mysql> GRANT ALL PRIVILEGES ON wordpress.* TO
'user'@'localhost' IDENTIFIED BY 'qwerty123';
Query OK, 0 rows affected (0.00 sec)
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql>exit
Bye
```

2. Go to html directory in the container and then download WordPress

```
# cd /var/www/html
```

3. Download WordPress with the following command and extract it

```
# wget http://wordpress.org/latest.tar.gz
# tar xzvf latest.tar.gz
```

4. Move the contents of WordPress directory to /var/www/html and remove index.html file

```
# mv wordpress/* /var/www/html
# rm index.html
```

5. Changing the ownership so that apache can access the files in WordPress directory

```
# chown -R www-data:www-data *
```

6. Complete the installation
   a) Copy the VM`s dns and paste it into the browser
   b) Provide db_name("wordpress"), db_user("user"), db_password("qwerty123")
   c) Let's keep the host as localhost and click submit
   d) Please provide a site name, password, and email address to complete the process

7. If the installation of WordPress has issues in creating the wp-config.php. Create a file named **wp-config.php** in the /var/www/html folder inside your **container** and copy the contents mentioned in the page

Task 3: Create Custom WordPress Image

1. Now we have a full-fledged WordPress container with mysql configured

2. Let's create an image out of it

3. To create an image, come out from the container **using ctrl+p+q** and give the container name or container id with docker commit command

4. Check the docker image present in the local repository of the server

```
$ docker commit wp my-custom-wordpress
$ docker images
```

```
ubuntu@manager:~$ docker commit wp my-custom-wordpress
docker imagessha256:fffbfd645a4c00a94ec05d8daeab2b55acfb4587b9600100b57dc7eddd00019d
ubuntu@manager:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my-custom-wordpress latest              fffbfd645a4c        4 seconds ago       467MB
nginx               latest              27a188018e18        5 days ago          109MB
ubuntu              14.04               390582d83ead        5 weeks ago         188MB
ubuntu@manager:~$
```

5. Remove the old wp container and run a new container using image we have committed in the previous step

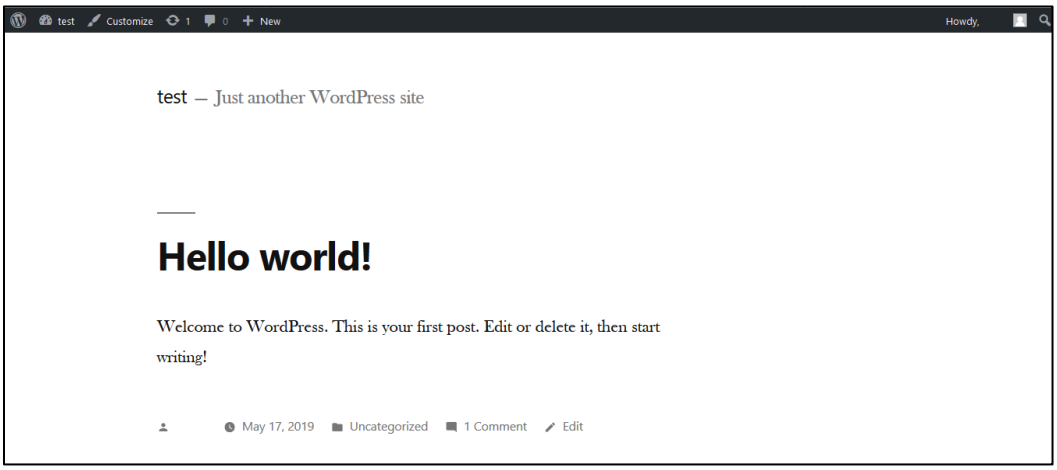
```
$ docker rm -f <wp container id>
$ docker run -it --name wp -p 80:80 my-custom-wordpress
/bin/bash
```

6. Start apache2 and mysql service inside the container

```
# service apache2 start
# find /var/lib/mysql -type f -exec touch {} \; && service
mysql start
```

```
ubuntu@manager:~$ docker run -it --name wp -p 80:80 my-custom-wordpress /bin/bash
root@e3297c05d46e:/# service apache2 start
 * Starting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name,
age
 *
root@e3297c05d46e:/# find /var/lib/mysql -type f -exec touch {} \; && service mysql start
 * Starting MySQL database server mysqld
 * Checking for tables which need an upgrade, are corrupt or were
not closed cleanly.
root@e3297c05d46e:/#
```

7. Copy the VM`s dns and paste it into the browser and click on **Install Wordpress** , Provide the Details, Login to Wordpress , Click on **Publish** . access the wordpress website



8. Remove the running container using the command below

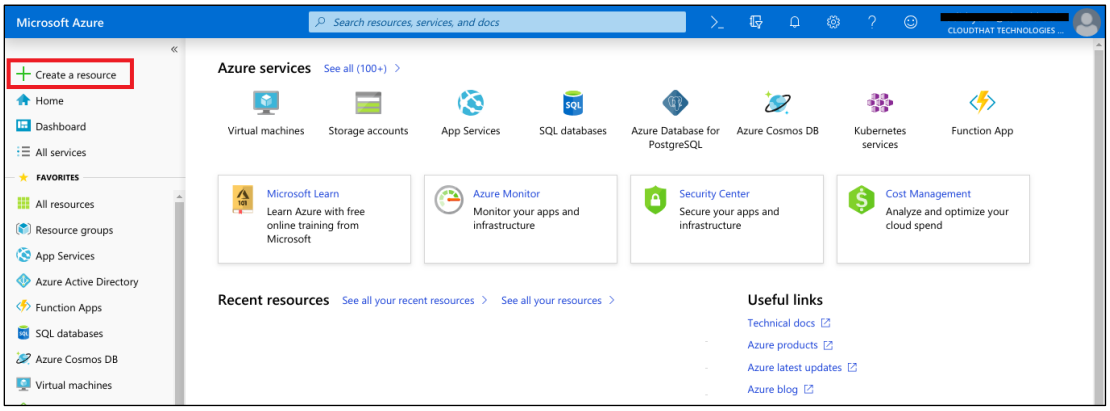
```
$ docker rm -f <container id>
```

Lab 3: Creating private Docker registry for Docker containers in ACR

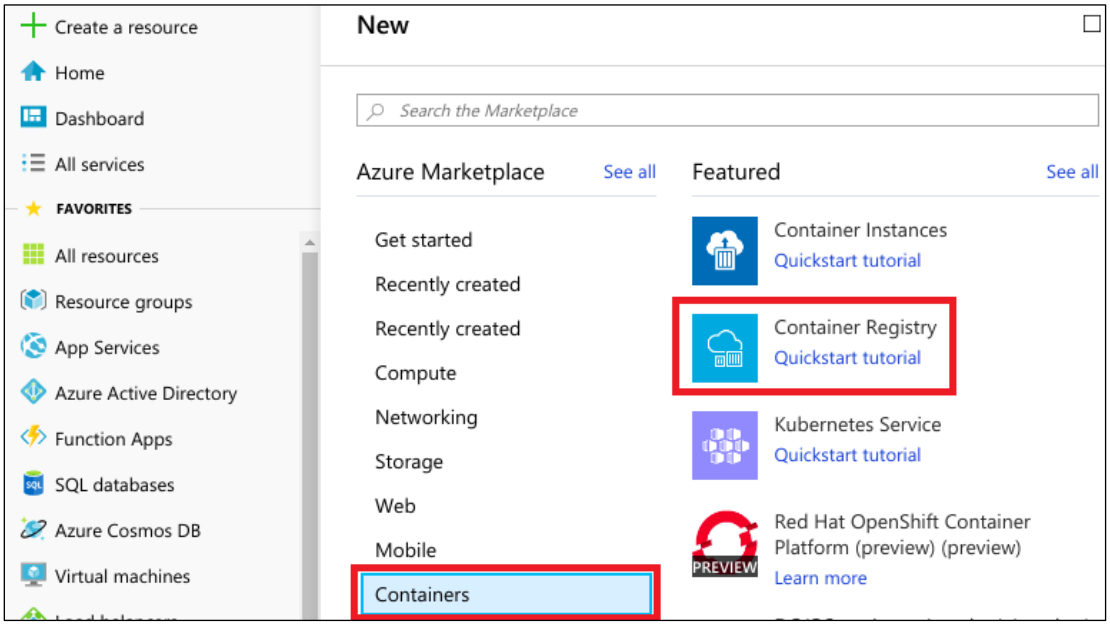
In this lab, a Docker image is created and pushed to Azure Container Registry.

Task 1: Creating a repository in Azure Container Registry

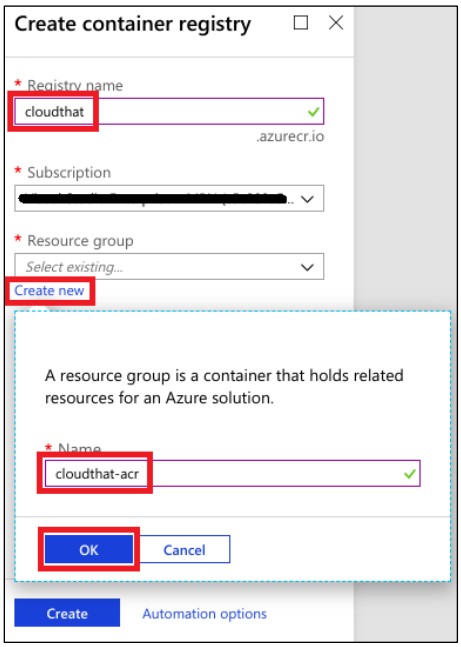
1. Access the Azure portal and click **Create a resource**



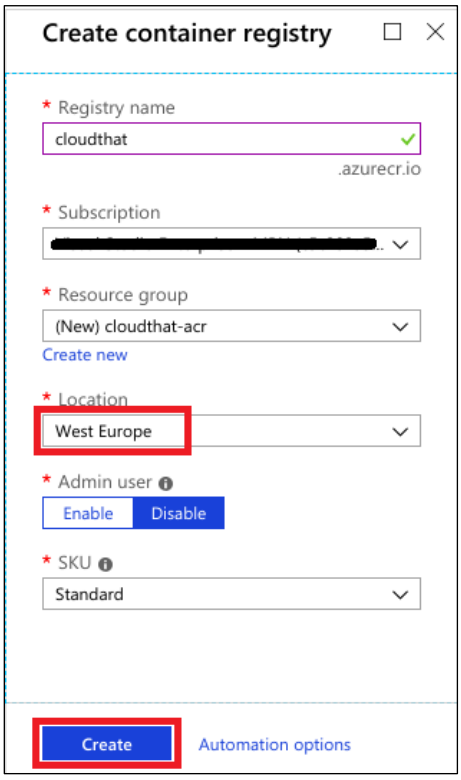
2. Select **Containers** and then click on **Container Registry**



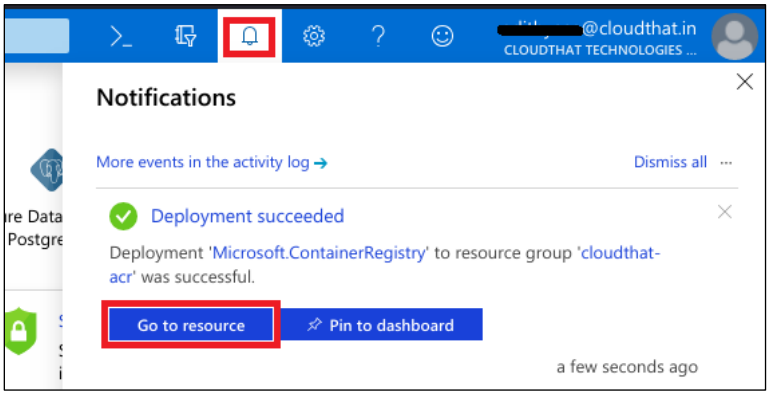
3. Enter a **UNIQUE Registry name,** select a **Subscription.** Under **Resource Group**, click **Create new**, enter a **Name** for the resource group and click **OK**

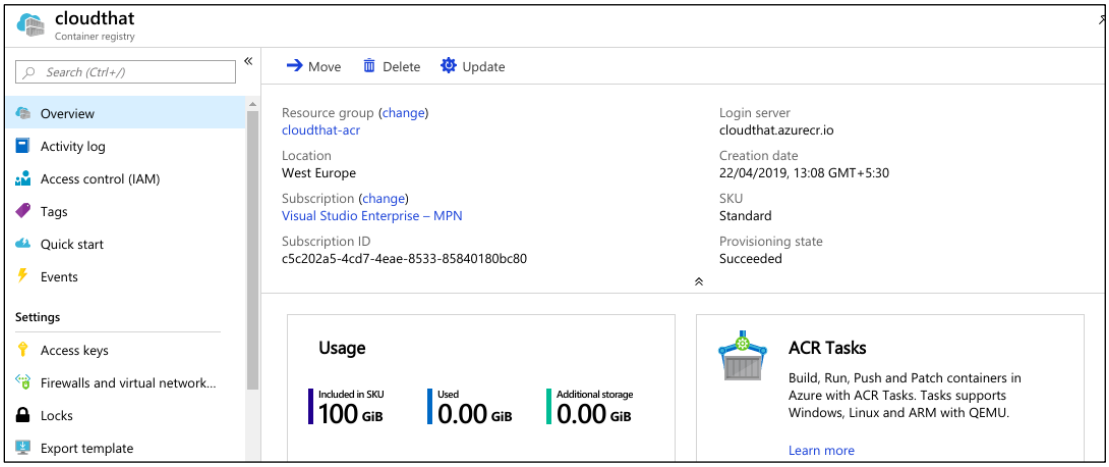


Select any **Location** and click **Create**

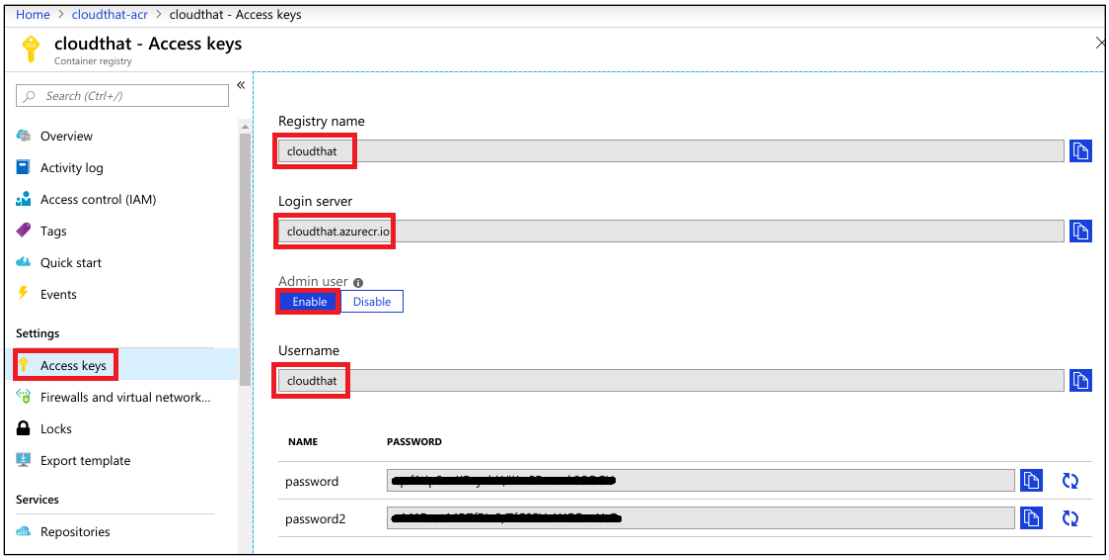


4. In **Notifications**, click **Go to resource** once the **Deployment** is **successful**



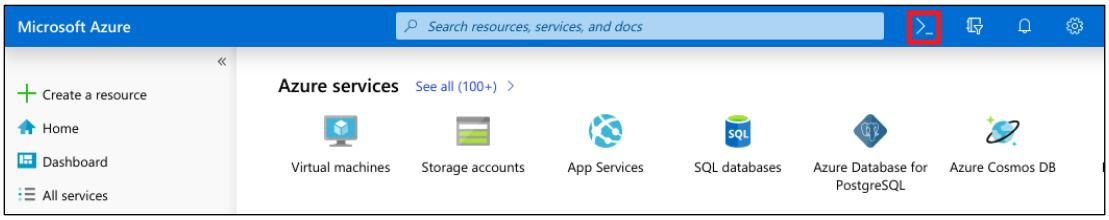


5. Click on **Access keys** and make a note of the **Registry name**, **Login server** and **Username**
   Enable Admin user and make a note of the two **Passwords**



Task 2: Working with application with Docker

1. Access the Azure cloud shell from the Azure portal



2. SSH to the VM created earlier using its **Public IP**

```
nandagopal@Azure:~$ ssh ubuntu@104.46.99.181
ubuntu@104.46.99.181's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1041-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud


To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@manager:~$
```

3. Start a container with Ubuntu 14.04 image and map host port 80 with port 80 of the container

```
$ docker run -it --name wp -p 80:80 ubuntu:14.04 /bin/bash
```

```
ubuntu@manager:~$
ubuntu@manager:~$ docker run -it --name wp -p 80:80 ubuntu:14.04 /bin/bash
Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
e082d4499130: Pull complete
371450624c9e: Pull complete
c8a555b3a57c: Pull complete
1456d810d42e: Pull complete
Digest: sha256:6612de24437f6f01d6a2988ed9a36b3603df06e8d2c0493678f3ee696bc4bb2d
Status: Downloaded newer image for ubuntu:14.04
root@0848a65b4d38:/#
```

Update the repositories in the container

```
# apt update
```

```
root@0848a65b4d38:/# apt update
Ign http://archive.ubuntu.com trusty InRelease
Get:1 http://archive.ubuntu.com trusty-updates InRelease [65.9 kB]
Get:2 http://archive.ubuntu.com trusty-backports InRelease [65.9 kB]
Get:3 http://archive.ubuntu.com trusty Release.gpg [933 B]
Get:4 http://archive.ubuntu.com trusty-updates/main amd64 Packages [1446 kB]
Get:5 http://security.ubuntu.com trusty-security InRelease [65.9 kB]
Get:6 http://archive.ubuntu.com trusty-updates/restricted amd64 Packages [21.4 kB]
Get:7 http://archive.ubuntu.com trusty-updates/universe amd64 Packages [668 kB]
Get:8 http://archive.ubuntu.com trusty-updates/multiverse amd64 Packages [16.1 kB]
Get:9 http://archive.ubuntu.com trusty-backports/main amd64 Packages [14.7 kB]
```

4. Install apache2, mysql, php5 in the **container**. When prompted for database root password enter **rootpasswd** and confirm again when prompted

```
# apt install -y apache2 mysql-server php php-fpm php-mysql
libapache2-mod-php wget
```

```
root@0848a65b4d38:/# apt-get install -y apache2 mysql-server php5 php5-fpm php5-mysql libapache2-mod-php5 wget
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  apache2-bin apache2-data ca-certificates libaio1 libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap libasn1-8-heimdal libdbd-mysql-perl
  libdbi-perl libedit2 libgssapi3-heimdal libhcrypto4-heimdal
  libheimbase1-heimdal libheimntlm0-heimdal libhtml-template-perl
  libhx509-5-heimdal libidn11 libkrb5-26-heimdal libldap-2.4-2
  libmysqlclient18 libroken18-heimdal libsasl2-2 libsasl2-modules
  libsasl2-modules-db libsystemd-daemon0 libterm-readkey-perl libwind0-heimdal
  libwrap0 libxml2 lsof mysql-client-5.5 mysql-client-core-5.5 mysql-common
```

5. Start the **apache2** and **mysql** services

```
# service apache2 start
# service mysql start
```

```
root@0848a65b4d38:/# service apache2 start
 * Starting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.
 suppress this message
 *
root@0848a65b4d38:/# service mysql start
 * Starting MySQL database server mysqld
root@0848a65b4d38:/#
```

6. Inside **container**, login to mysql and create a database for Wordpress.

   Enter the password provided during the installation of MySQL

```
# mysql -u root -p
```

```
root@0848a65b4d38:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 47
Server version: 5.5.62-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

```
mysql> CREATE DATABASE wordpress;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE USER 'user'@'localhost' IDENTIFIED BY
'qwerty123';
Query OK, 0 rows affected (0.00 sec)
mysql> GRANT ALL PRIVILEGES ON wordpress.* TO
'user'@'localhost' IDENTIFIED BY 'qwerty123';
Query OK, 0 rows affected (0.00 sec)
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql>EXIT
Bye
```

```
mysql> CREATE DATABASE wordpress;
Query OK, 1 row affected (0.00 sec)

mysql> CREATE USER 'user'@'localhost' IDENTIFIED BY 'qwerty123';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

```
mysql> GRANT ALL PRIVILEGES ON wordpress.* TO 'user'@'localhost' IDENTIFIED BY 'qwerty123';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> EXIT
Bye
root@0848a65b4d38:/#
```

7. Go to /html directory in the container then download WordPress

```
# cd /var/www/html/
```

```
root@0848a65b4d38:/# cd /var/www/html/
root@0848a65b4d38:/var/www/html#
```

8. Download WordPress with the following command and extract it

```
# wget https://wordpress.org/latest.tar.gz
# tar xzf latest.tar.gz
```

```
root@0848a65b4d38:/var/www/html# wget https://wordpress.org/latest.tar.gz
--2019-04-22 07:55:17--  https://wordpress.org/latest.tar.gz
Resolving wordpress.org (wordpress.org)... 198.143.164.252
Connecting to wordpress.org (wordpress.org)|198.143.164.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10654780 (10M) [application/octet-stream]
Saving to: 'latest.tar.gz'

100%[===================================================================================

2019-04-22 07:55:19 (10.4 MB/s) - 'latest.tar.gz' saved [10654780/10654780]

root@0848a65b4d38:/var/www/html#
```

```
root@0848a65b4d38:/var/www/html# tar xzf latest.tar.gz
root@0848a65b4d38:/var/www/html#
```

9. Move the contents of WordPress directory to /var/www/html
   Delete index.html file

```
# mv wordpress/* .
# rm index.html
```

```
root@0848a65b4d38:/var/www/html# mv wordpress/* .
root@0848a65b4d38:/var/www/html#
```

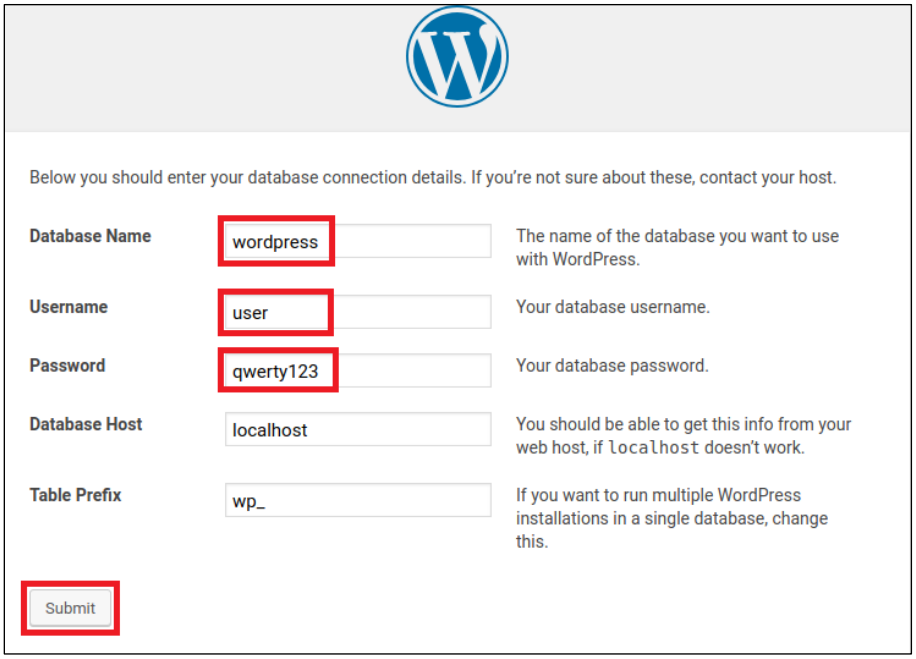
```
root@0848a65b4d38:/var/www/html# rm index.html
root@0848a65b4d38:/var/www/html#
```

10. Changing the ownership so that apache can access the files in WordPress directory

```
# chown -R www-data:www-data *
```

```
root@0848a65b4d38:/var/www/html# chown -R www-data:www-data *
root@0848a65b4d38:/var/www/html#
```

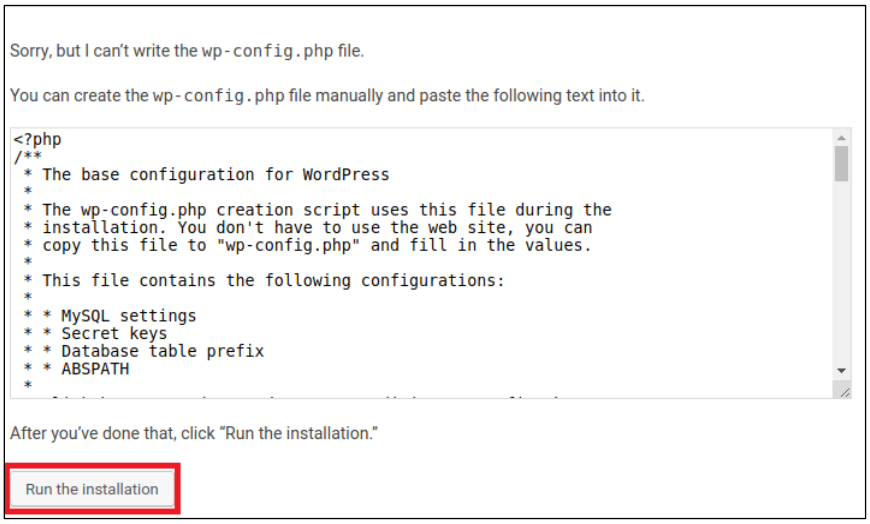
11. Complete Wordpress installation
    a) Copy the public IP address of the Azure VM and paste it into the browser
    b) Enter **Database Name** as **wordpress**, **Username** as **user** and **Password** as **qwerty123**
    c) Click **Submit**



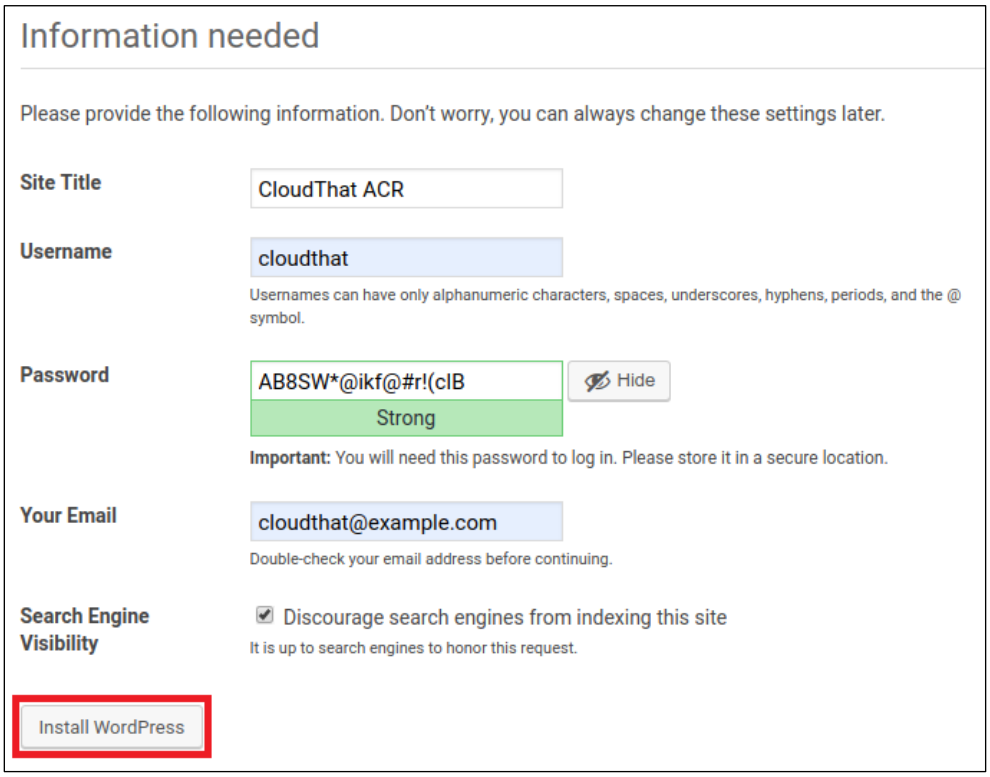
12. If the installation of WordPress has issues in creating the wp-config.php. Create a file named **wp-config.php** in the /var/www/html folder inside your **container** and copy the contents mentioned in the page

```
root@0848a65b4d38:/var/www/html# vi wp-config.php
```

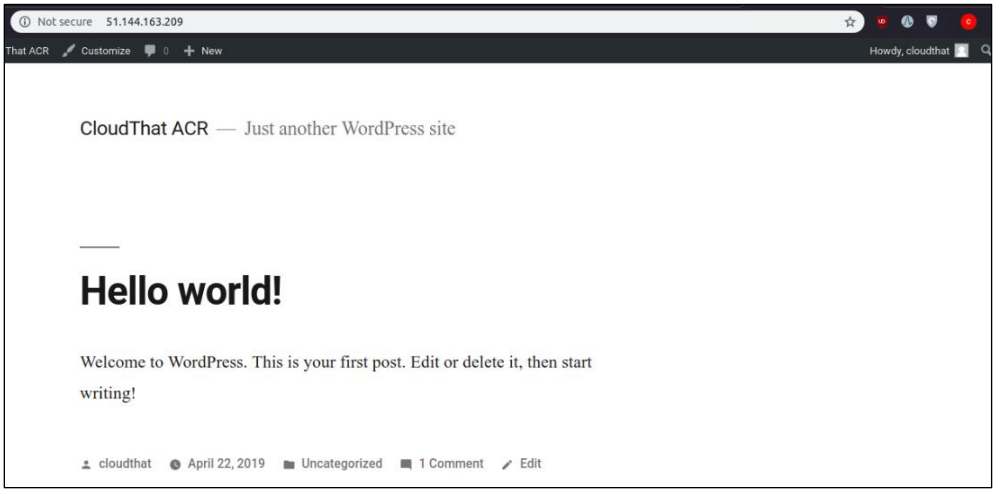
Then click **Run the Installation**



13. Fill in the additional information and click **Install WordPress**



14. Access the Public IP in the browser



Task 3: Create Custom WordPress Image

Now we have a full-fledged WordPress container with mysql configured.Let's create an image out of it.
To create an image, come out from the container **using ctrl+p+q** and give the container name or container id with docker commit command Check the docker image present in the local repository of the server.

1. Return to the cloud shell, and exit from the container using **Ctrl + p + q**
   Make a note of the container ID

```
$ docker container ls
```

```
ubuntu@manager:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
0848a65b4d38        ubuntu:14.04        "/bin/bash"         44 minutes ago      Up 44 minutes       0.0.0.0:80->80/tcp  wp
ubuntu@manager:~$
```

2. Commit the image using the Azure Container Registry URL from 5th step of 1st task

```
$ docker commit <container_id> <login_server>/<repo_name>
```

```
ubuntu@manager:~$ docker commit 0848a65b4d38 cloudthat.azurecr.io/custom-wordpress
sha256:3e8f2ad0d1f1aea82907807eb52dc1d58858008e6b2b93c7eca0a386be745b7a
ubuntu@manager:~$
```

3. List and verify that the image is created

```
$ docker image ls
```

```
ubuntu@manager:~$ docker image ls
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
cloudthat.azurecr.io/custom-wordpress  latest              3e8f2ad0d1f1        About a minute ago  425MB
ubuntu                              14.04               390582d83ead        5 weeks ago         188MB
ubuntu@manager:~$
```

4. Log in to the Azure Container Registry. Use the credentials from 5th step of 1st task. Use either one of the passwords

```
$ docker login -u <username> <login_server>
```

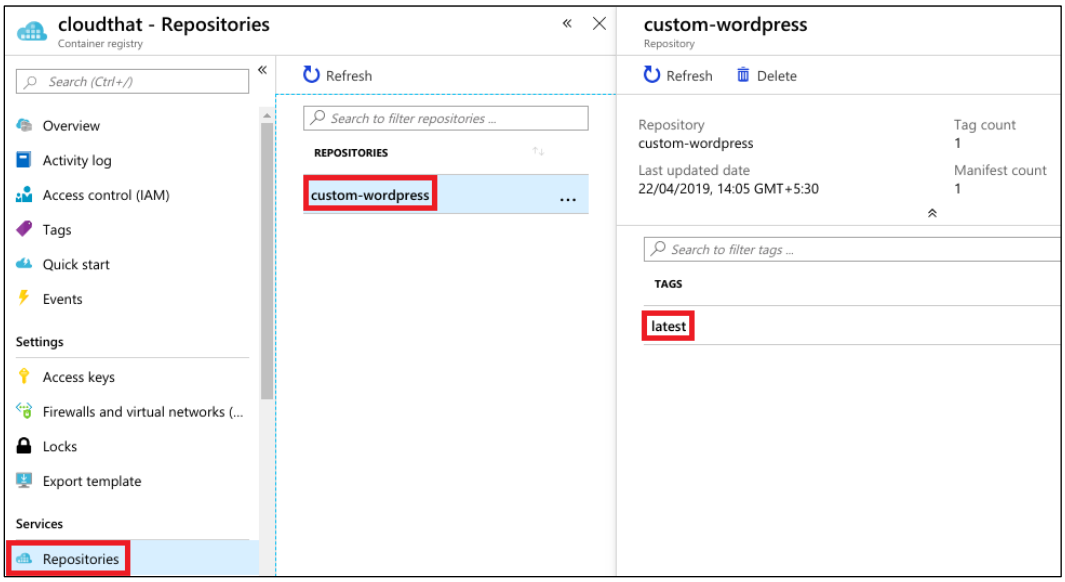
```
ubuntu@manager:~$ docker login -u cloudthat cloudthat.azurecr.io
Password:
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu@manager:~$
```

5. Once Login is Successful, push the image to Azure Container Registry

```
$ docker push <login_server>/<repo_name>
```

```
ubuntu@manager:~$ docker push cloudthat.azurecr.io/custom-wordpress
The push refers to repository [cloudthat.azurecr.io/custom-wordpress]
ecb42edb8eb6: Pushed
5f96fa66dc12: Pushed
dda5ec330bd9: Pushed
11a0c2f551fd: Pushed
eef560b4ec4f: Pushed
latest: digest: sha256:d4c0477ffdf020d016068e392e3defc59123e8cb00fcdeace71d7f7eb8043b7f size: 1364
ubuntu@manager:~$
```

6. Navigate to the **Repositories** under the previously created **Azure Container Registry** and select the **custom-wordpress** under **Repositories**



Lab 4: Building a Dockerfile to Setup an Ubuntu Container with WordPress Application

Dockerfile helps to automate the steps of launching the container and configure the container when it is launching. Docker file uses the basic DSL with instructions for building Docker images. The dockerfile approach is recommended over Docker commit because it provides a more repeatable, transparent, and idempotent mechanism for creating images.

Task 1: Writing a Dockerfile

1. Create a **directory** to save the Dockerfile and the dependent files. Create a file with name "**Dockerfile**" that has the sequence of steps for launching the container

```
$ mkdir wordpress
$ cd wordpress
$ vim Dockerfile
```

2. This directory is our build environment; this is the build context. Hence Dockerfile is placed in this directory for the build. Enter the below lines of code in the **Dockerfile**

```
FROM ubuntu:16.04
MAINTAINER Username "user@example.com"
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && \
apt-get -q -y install apache2 \
mysql-server \
php \
php-fpm \
php-mysql \
libapache2-mod-php \
wget \
vim
RUN wget http://wordpress.org/latest.tar.gz && \
tar xzvf latest.tar.gz && \
cp -R ./wordpress/* /var/www/html && \
rm /var/www/html/index.html
RUN chown -R www-data:www-data /var/www/html
EXPOSE 80
ENTRYPOINT /bin/bash
VOLUME /var/lib/mysql
```

3. This is the Dockerfile to build the container with **WordPress**. It follows the sequence of the following:
   a) **Pulls** the base **Ubuntu:14.04 image** from the public repo
   b) Installs **Apache, mysql & php** on Ubuntu
   c) Downloads **WordPress** in the container
   d) **Extract** the **zip** file for WordPress
   e) **Copies** the contents of the **WordPress** folder **to /var/www/html**, which is the default Document Root of Apache
   f) **Removes** the **default index.html** from the Document Root
   g) **Change** the **ownership** of the folder to **www-data**, which is the user for Apache
   h) **Exposes** the **port 80** of the container to the world
   i) **Volume**: It will map the specified container directory on the host
   j) **ENTRYPOINT**: this is where all commands are executed inside the container

   Run the **Docker build** in the directory where the Dockerfile is located. The command to build a container with Dockerfile:

```
$ docker build -t web .
```

   Here, build denotes the image must be built from a file named "Dockerfile". **-t** denotes the addition of a tag to the image after building. "web" is the name given to the image that is created by the Dockerfile.

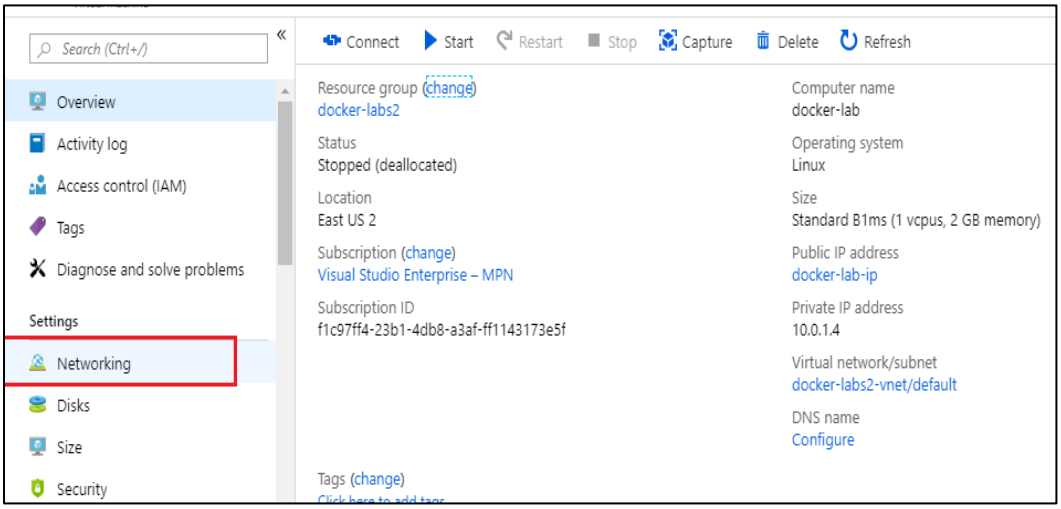
   Note: Do not forget the "." at the end of the command which is the current directory path.

4. The Dockerfile is built, and an image is generated in the Docker host. To find the image in the Docker host, enter the following command

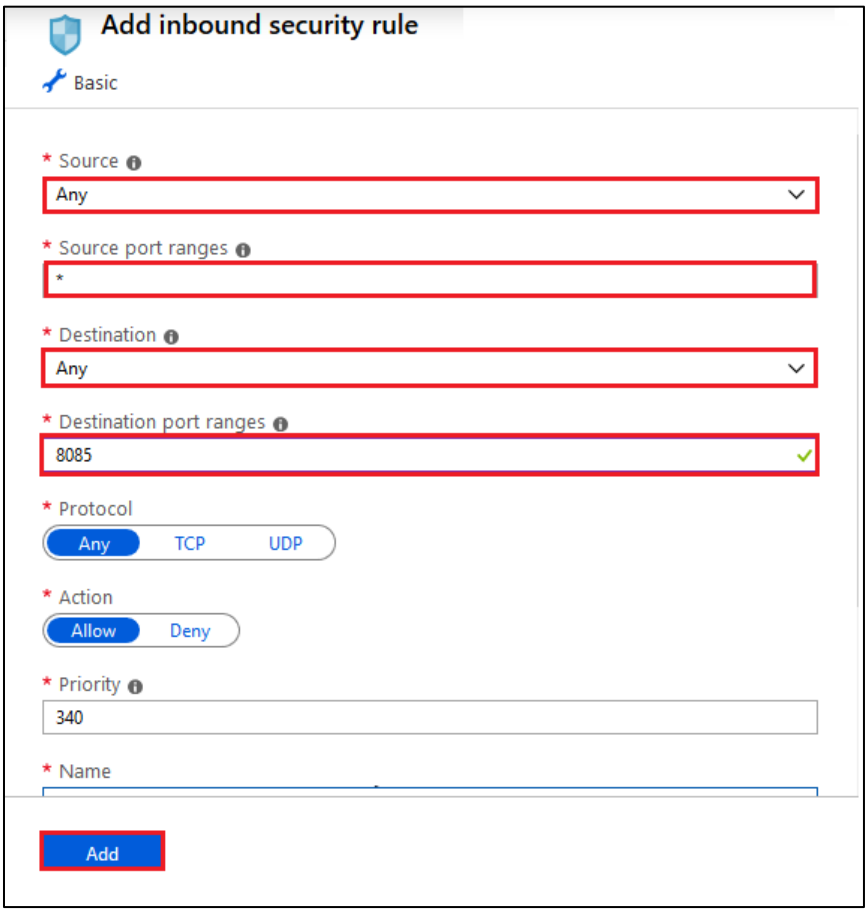
```
$ docker images
```

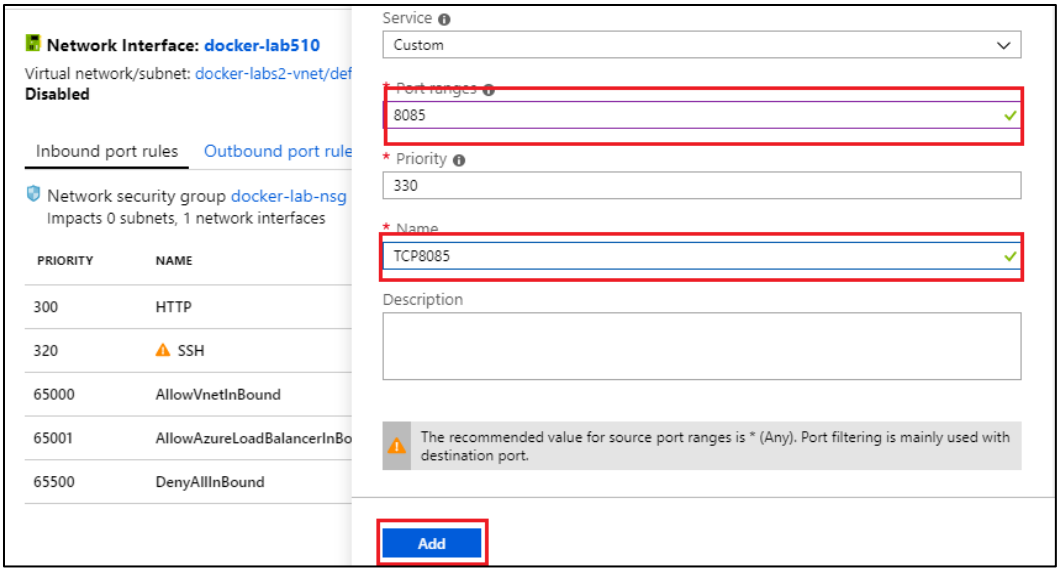
```
ubuntu@manager:~/wordpress$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
web                 latest              db71ed6b7b08        6 seconds ago       496MB
my-custom-wordpress latest              fffbfd645a4c        14 minutes ago      467MB
nginx               latest              27a188018e18        5 days ago          109MB
ubuntu              14.04               390582d83ead        5 weeks ago         188MB
ubuntu@manager:~/wordpress$
```

5. Launch a Docker container using previously created Docker image and **enable 8085 TCP port in your NSG**. Use the following command to run the container



In Networking tab → click on Add inbound port rule, Enter the following details, Click Add






```
$ docker run -it -p 8085:80 web
```

6. Now, when we are inside the container, **check** the **status** of **Apache** by the following command

```
# service apache2 status
```

7. Start the **Apache service.** To start the service, use the command

```
# service apache2 start
```

8. Check the status of **Mysql** by the following command and start it

```
# service mysql status
* MySQL not running
# service mysql start
*MySQL is running
```

Task 2: Creating Database

1. Create a database named **WordPress.** Use this as a database store for WordPress setup. Make sure that MySQL service is running in your WordPress container

2. Inside your container login to MySQL shell using the following command with blank password

```
# mysql -u root -p
```

3. To create a database and user in MySQL, use the following command

```
mysql> CREATE DATABASE wordpress;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE USER 'user'@'localhost' IDENTIFIED BY
'qwerty123';
Query OK, 0 rows affected (0.00 sec)
mysql> GRANT ALL PRIVILEGES ON wordpress.* TO
'user'@'localhost' IDENTIFIED BY 'qwerty123';
Query OK, 0 rows affected (0.00 sec)
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql>exit
Bye
```

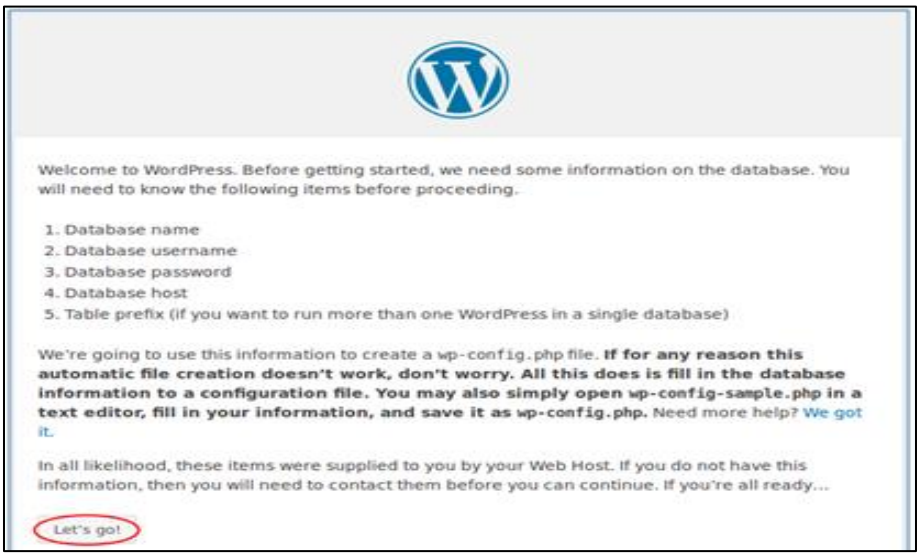
Note: The MySQL database name, username and password are essential to be remembered to configure the "**wp-config.php**" file of WordPress.

4. Exit the container using escape sequence **ctrl+p+q**

Task 3: Deploying WordPress

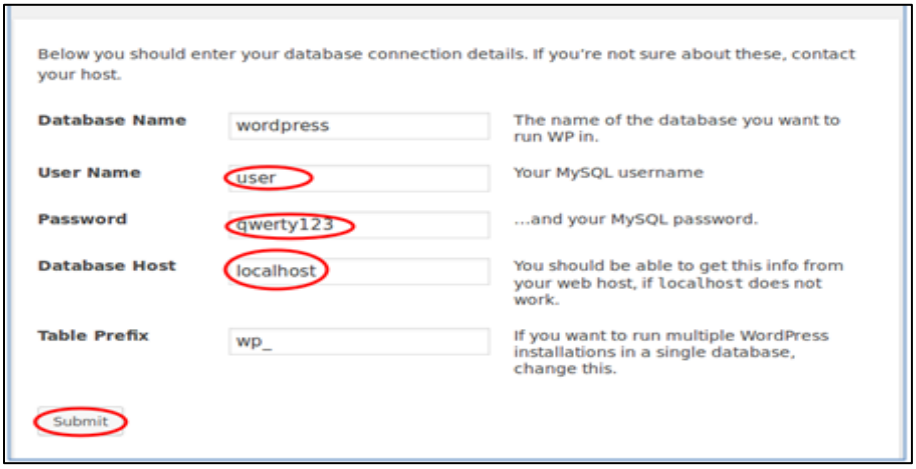
1. Visit the WordPress site with the respective port to which we have mapped the WordPress container. For example, **"Public_IP:8085"**

2. Click "**Let's go**" which takes to the database configuration page. Which requires the database details to be entered in the **"wp-config.php**" file



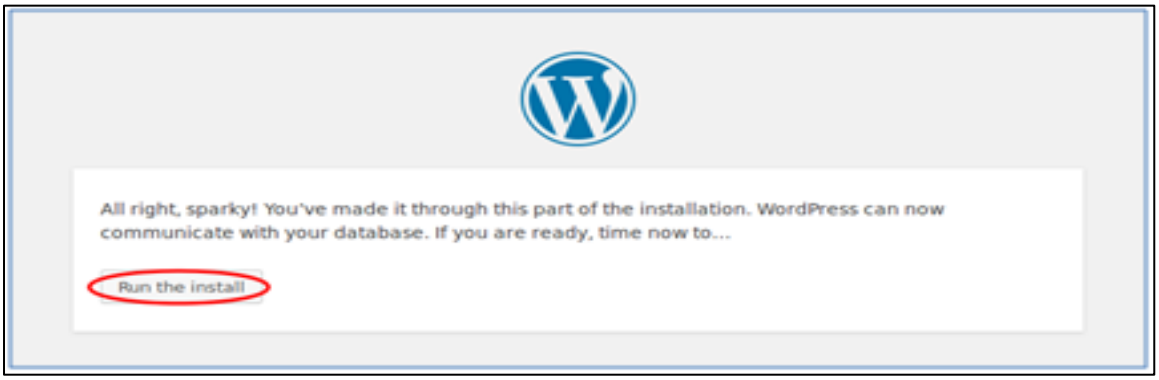
3. **Database setup**

Enter the Database details for the WordPress to store the data



4. **Installation**

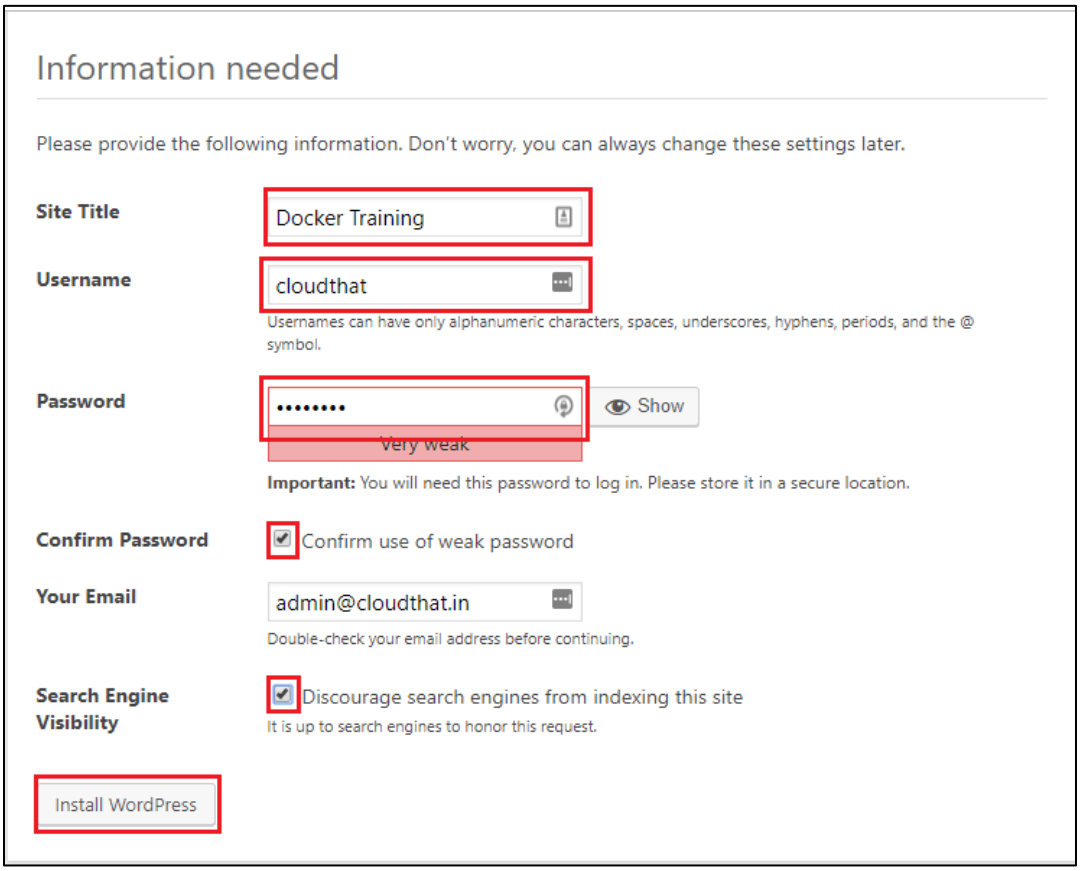
Once the database details are accepted, then move ahead to "Run the install"



Note: Remember the username and password that is set here to login the WordPress site again.

5. **Website Information**

This takes us to the Installation page of WordPress, where the required fields to log in to the WordPress site has to be mentioned

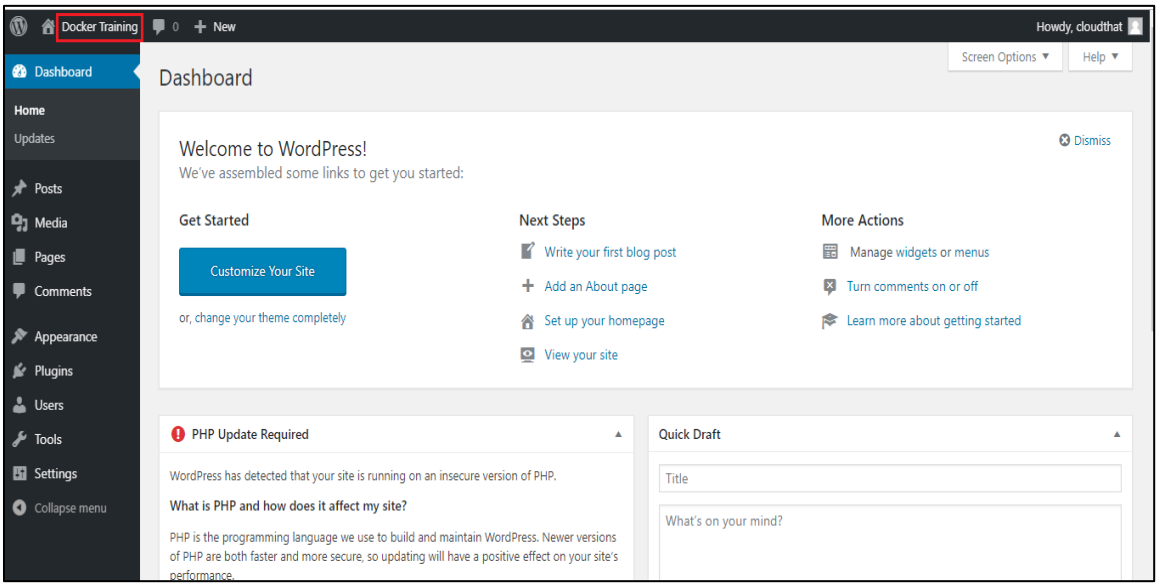


6. **Admin login**

The login page of WordPress appears once the username and password are given

7. **Admin dashboard**

   The WordPress dashboard appears with the Site name as **Docker Training** and with the username. Here we can add new posts, comments, media files, etc



8. Website's Database is hosted local to the machine
   The WordPress site is getting hosted from the container which has a local database, WordPress

9. Remove the containers using the command

```
$ docker rm -vf <container_id>
```