

Working with Docker on Microsoft Azure

Lab Overview

In this lab you will create a Docker enabled virtual machine from the Azure Marketplace. You will then go through basic Docker commands. After that, you will learn how to Dockerize a sample application. You will also configure a multi-container application using Docker compose.

Prerequisites

- Microsoft Azure Subscription: <http://azure.microsoft.com/en-us/pricing/free-trial/>
- Windows client computers will need an SSH client to complete the lab such as:
 - Git Bash with SSH client from <http://www.git-scm.com/downloads>
 - [PuTTY](#),
 - [AnyConnect](#)
- Docker evaluation license file: <http://emails.microsoft.com/o00sW00Oh00mmG1TTEQEqq0>

Time Estimate

- 45 minutes

Technical Support

Having trouble with this lab or have a question? Please contact SuperHuman_Help@microsoft.com for technical assistance.

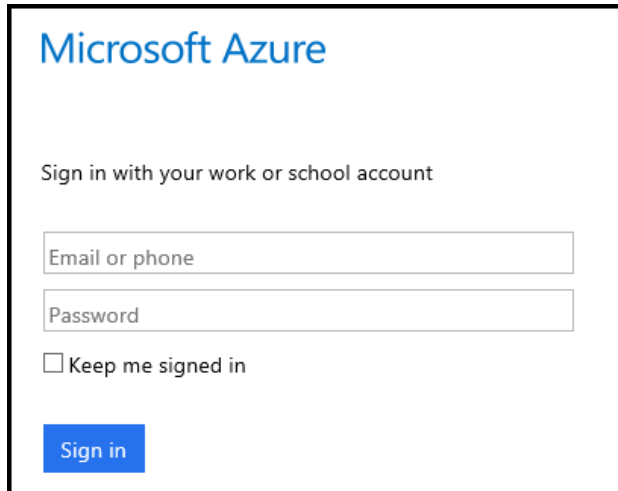
Table of Contents

Working with Docker on Microsoft Azure.....	1
Lab Overview.....	1
Prerequisites	1
Time Estimate.....	1
Technical Support.....	1
Table of Contents	1
Exercise 1: Login to the Microsoft Azure Portal	2
Exercise 2: Create Virtual Machine with Docker Engine	3
Exercise 3: Get started with Docker	11
Exercise 4: Create and Dockerize your Application.....	12
Exercise 5: Manage multi container application with Docker Compose	15
Validate Lab Completion	19
ssh demouser@<public ip address>	19
cd HelloRedis	19
Lab Summary	20

Exercise 1: Login to the Microsoft Azure Portal

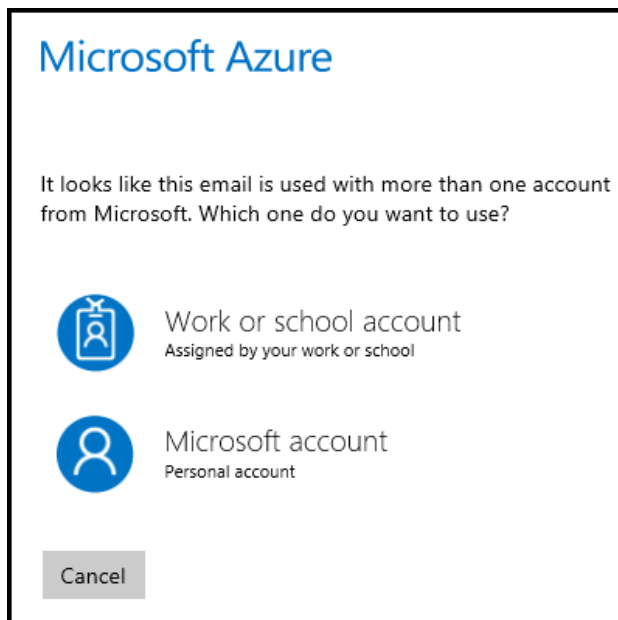
In this exercise you will use your Microsoft or Organization account to login to the Azure Portal to start the lab exercise.

1. Open your browser and navigate to <https://portal.azure.com/>
2. Enter the account associated with your Microsoft Azure subscription.



The image shows the Microsoft Azure login page. At the top, the text "Microsoft Azure" is displayed in blue. Below it, the instruction "Sign in with your work or school account" is shown. There are two input fields: "Email or phone" and "Password". Below these fields is a checkbox labeled "Keep me signed in". At the bottom left, there is a blue button labeled "Sign in".

3. If your account is associated with an organization account and a Microsoft account you may be prompted to choose which one to authenticate with for your Microsoft Azure account.

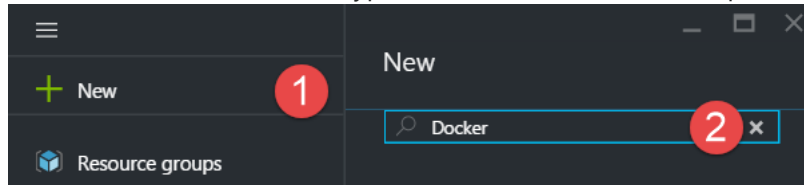


The image shows a Microsoft Azure account selection page. At the top, the text "Microsoft Azure" is displayed in blue. Below it, the message "It looks like this email is used with more than one account from Microsoft. Which one do you want to use?" is shown. There are two options, each with a blue circular icon containing a white person silhouette. The first option is "Work or school account" with the subtext "Assigned by your work or school". The second option is "Microsoft account" with the subtext "Personal account". At the bottom left, there is a grey button labeled "Cancel".

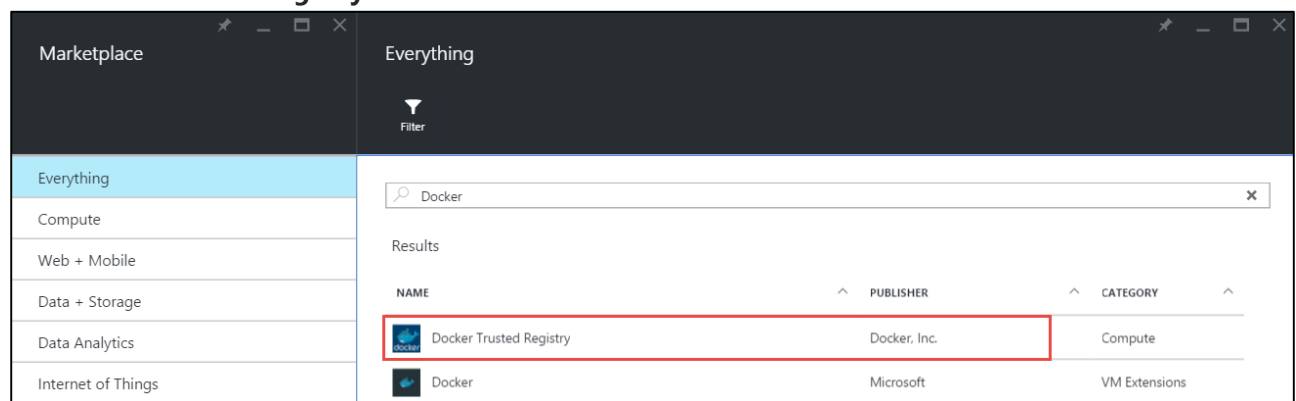
Exercise 2: Create Virtual Machine with Docker Engine

In this task you will create a virtual machine using the 'Docker Trusted Registry' image available in Azure Marketplace.

1. Download the evaluation license to your local machine if you have not already done so. The Docker evaluation license file may be obtained from here: <http://emails.microsoft.com/o00sW00Oh00mmG1TTEQEgq0>
2. Click the **+New** button, then type **Docker** in the text box and press **enter**.



3. Click **Docker Trusted Registry** from the search results.



4. After reading the description, click **Create** button. Note that deployment model is defaulted to '**Resource Manager**'.



Docker Trusted Registry

Docker, Inc.

Bring Your Own License enabled.

Docker Subscription is a solution to build, ship and run distributed applications. The subscription includes Docker Trusted Registry (DTR), Docker Engine and commercial support subscription is available with support tiers to align to your application SLAs.

Packaged as a containerized application, Docker Trusted Registry (DTR) allows you to store and manage your Docker images securely inside your firewall while integrating to your CI workflow.

Highlight Features

Run instantly on Azure. Docker Subscription allows you to run Docker Trusted Registry, Docker Engine and your Dockerized applications all within your Azure infrastructure.

Easy to use and manage. A web admin console makes it easy to manage user access, configure storage and security certificates, view system health, audit logs and metrics.

One-click upgrade: An update button will appear when patches and upgrades are ready.

Purchase licenses or access an evaluation for a Docker Subscription: <https://hub-beta.docker.com/enterprise/>

Purchase licenses for Docker Subscription here: <https://hub-beta.docker.com/enterprise/>. The subscription license includes: 1 instance of Docker Trusted Registry, Support for 10 commercially supported Docker Engines, Email Support, 1 Year License Subscription.



Docker Trusted Registry

Home System Health Settings Logs

Settings

HTTP Security Storage Authentication License

Authentication Method

Managed

Username	Password	Global Role	Delete
----------	----------	-------------	--------

Select a deployment model ⓘ

Resource Manager

Create

5. Specify the following virtual machine **basics** configuration and click **OK**.

- Name: azure-docker-vm
- User Name: demouser
- Authentication Type: Password
- Password: demo@pass1
- Resource group: DockerVMs

- Location: The Azure region closest to your location.

Create virtual machine

Basics

- 1 Basics**
Configure basic settings
- 2 Size**
Choose virtual machine size
- 3 Settings**
Configure optional features
- 4 Summary**
Docker Trusted Registry
- 5 Buy**

* Name: azure-docker-vm ✓

* User name: demouser ✓

* Authentication type: Password SSH public key

* Password: ✓

* Subscription: Windows Azure MSDN - Visual Studio Ult

* Resource group: DockerVMs

[Select existing](#)

* Location: South Central US

6. For **virtual machine size** configuration, select **D2 Standard** and click **Select**

Create virtual machine

Choose a size
Browse the available sizes and their features

Prices presented below are estimated retail prices that include both Azure infrastructure and applicable third-party software costs. Prices do not reflect applicable discounts for your subscription and may include currency conversions.

★ Recommended | [View all](#)

D2 Standard ★	D3 Standard ★
2 Cores	4 Cores
7 GB	14 GB
4 Data disks	8 Data disks
4x500 Max IOPS	8x500 Max IOPS
100 GB Local SSD	200 GB Local SSD
Load balancing	Load balancing
Auto scale	Auto scale
99.70 USD/MONTH (ESTIMATED)	199.39 USD/MONTH (ESTIMATED)

Select

7. For **Settings** (Configure optional features), leave default values and click **OK**.

Note the below.

1 Basics Done ✓

2 Size Done ✓

3 Settings Configure optional features >

4 Summary Docker Trusted Registry >

5 Buy >

Storage

Disk type ⓘ

Standard Premium (SSD)

* Storage account ⓘ >

(new) dockervms1303

Network

* Virtual network ⓘ >

(new) DockerVMs

* Subnet ⓘ >

default (172.16.0.0/24)

* Public IP address ⓘ >

(new) azure-docker-vm

* Network security group ⓘ >

(new) azure-docker-vm

Monitoring

Diagnostics ⓘ

Disabled Enabled

* Diagnostics storage account ⓘ >

(new) dockervms1303

Availability

OK

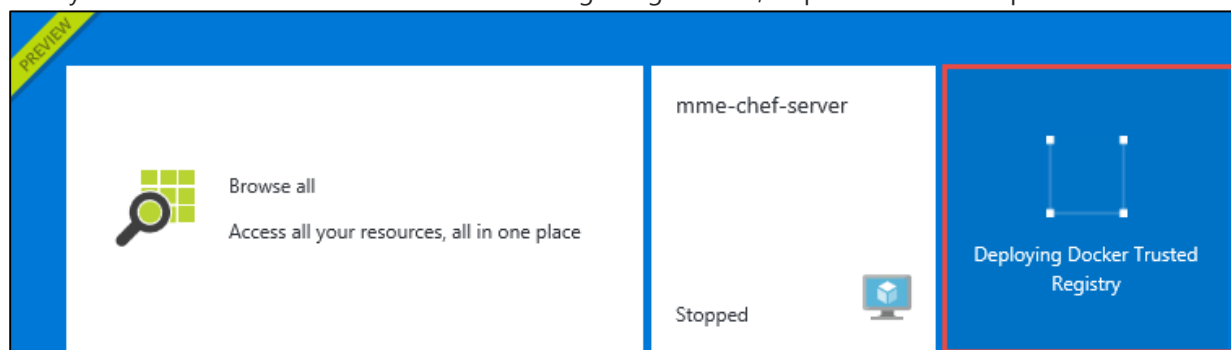
8. For **Summary** (Docker Trusted Registry) configuration, review details and click **OK**

1 Basics Done	✓	Subscription	Visual Studio Premium with MSDN
2 Size Done	✓	Resource group	(new) DockerVMs
3 Settings Done	✓	Location	South Central US
4 Summary Docker Trusted Registry	>	Computer name	azure-docker-vm
		User name	azureuser
		Size	Standard D2
		Disk type	Standard
		Storage account	(new) dockervms1303
		Virtual network	(new) DockerVMs
		Subnet	(new) default (172.16.0.0/24)
		Public IP address	(new) azure-docker-vm
		Network security group	(new) azure-docker-vm
		Availability set	None
		Diagnostics	Enabled
		Diagnostics storage account	(new) dockervms1303
5 Buy	>	OK	

9. For **Buy**, review offer details and click **Purchase**

1 Basics Done	✓	Offer details	
2 Size Done	✓	Docker Trusted Registry by Docker, Inc.	
3 Settings Done	✓	Standard D2	0.00 USD (Bring your own license)
4 Summary Docker Trusted Registry	✓	Terms of use and privacy policy	Pricing for other VM sizes
5 Buy	>	<p>Pricing above does not include Azure infrastructure costs (e.g., virtual machine compute time or storage) and is based on the pricing tier you have selected. Neither Microsoft subscription credits nor monetary commitment funds may be used to purchase the above offering(s). These purchases are billed separately. If any Microsoft products are listed above (e.g., Windows Server or SQL Server), such products are licensed by Microsoft and not by any third party.</p> <p>Terms of use</p> <p>By clicking "Purchase," I (a) agree to the legal terms and privacy statement(s) associated with each offering above, (b) authorize Microsoft to charge or bill my current payment method on a quarterly basis for the fees associated with my use of the offering(s), including applicable taxes, until I discontinue use of the offering(s), and (c) agree that Microsoft may share my contact information with any third-party vendors, if listed above. Microsoft does not provide rights for third-party products or services. See the Azure Marketplace Terms for additional terms.</p>	
		Purchase	

10. Now you will see that the new virtual machine is getting created, as per the status on portal dashboard.

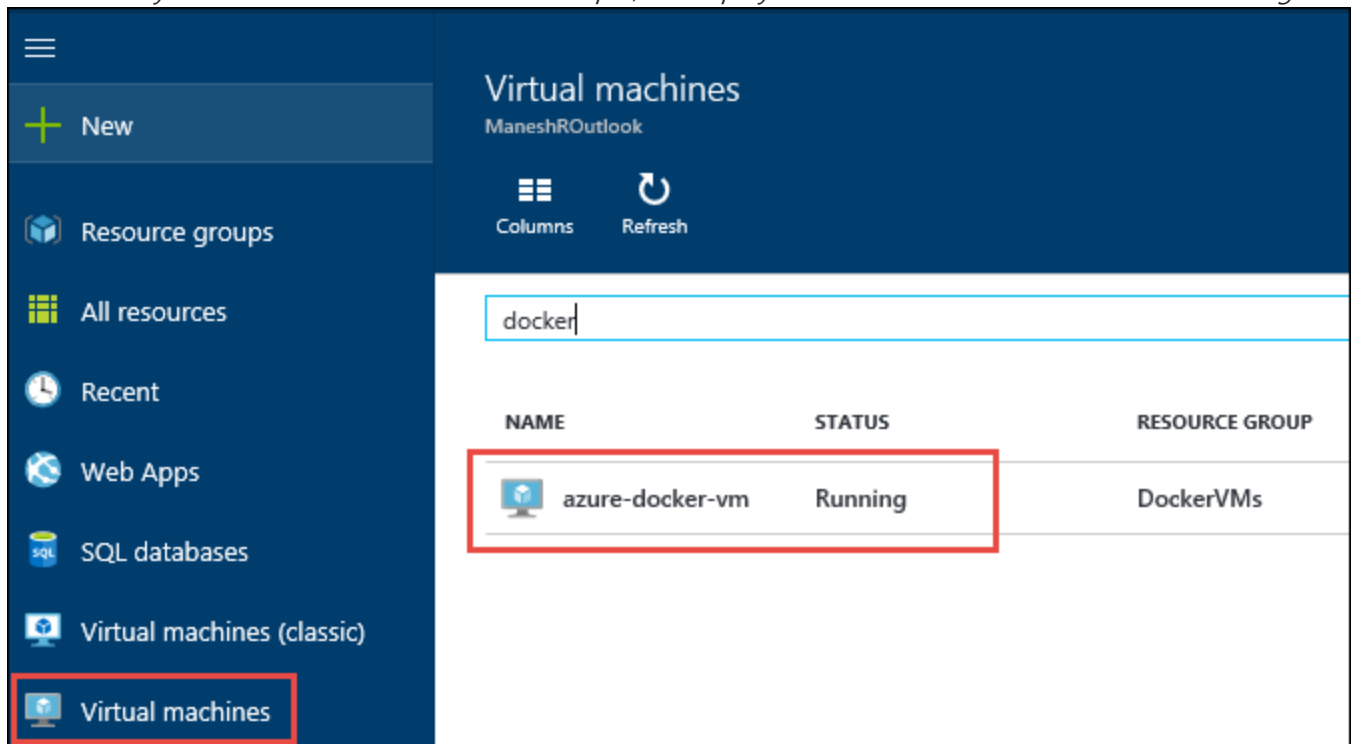


11. Wait until the status of created virtual machine is '**Running**'.



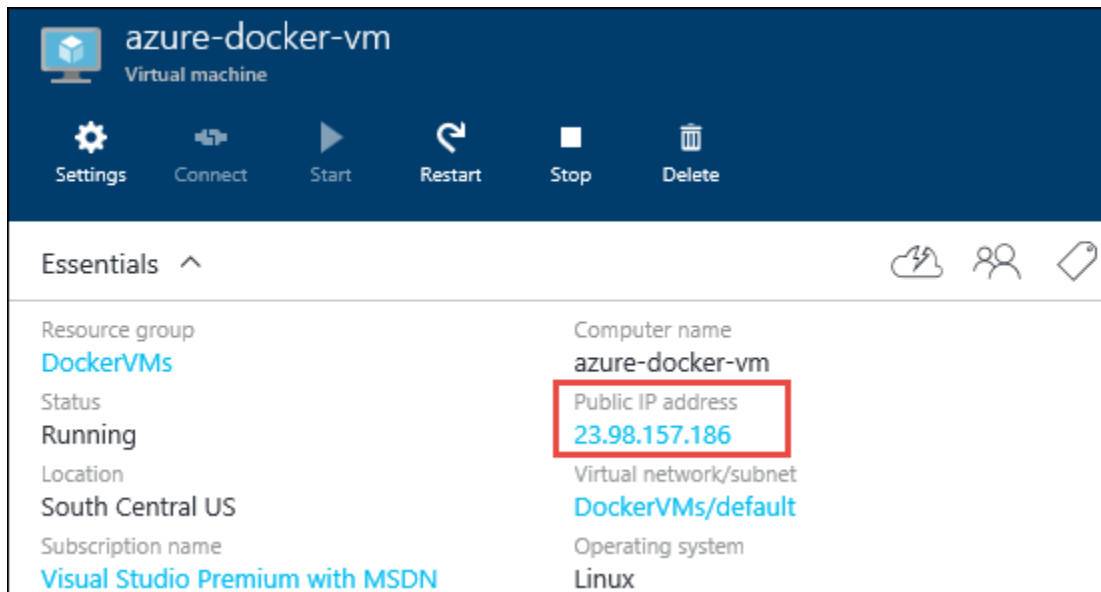
12. Sometimes, portal dashboard takes time to refresh. You can also click **Virtual Machines** to see the latest status of the virtual machine. Wait until the status turns to '**Running**'.

Note: When you created the virtual machine in step 4, the deployment model was chosen as Resource Manager.

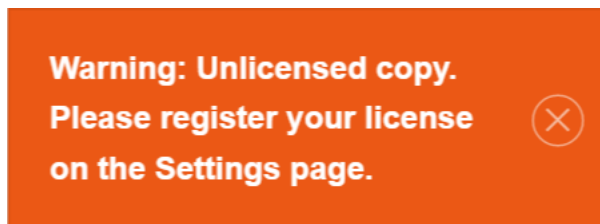


13. Once virtual machine is in status '**Running**', click on the **virtual machine name** and go to details. Note down the **Public IP Address** of the virtual machine.

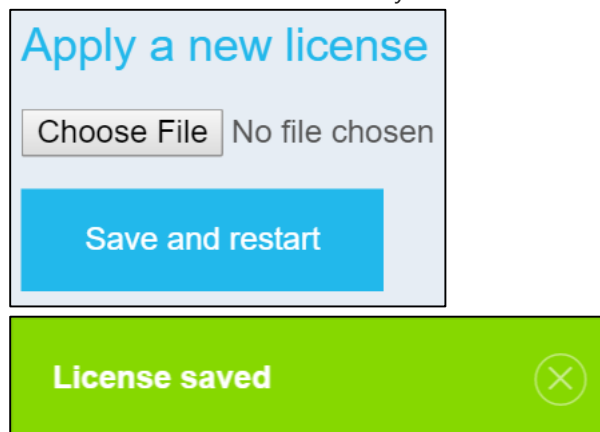
Note: You will be using this IP address to browse to https endpoint to apply the license, as well as for connecting to virtual machine over SSH.



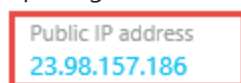
14. Launch a browser and navigate to the public IP address of the virtual machine (<https://<public-ip-address>/>). Continue on past the SSL warning.
15. Click the '**Warning: Unlicensed copy**' to navigate to the settings page.



16. Click **Choose File** and browse to your trial license file and then click **Save and Restart**.



17. For connecting to the virtual machine, you should use the public IP address. The public IP can be obtained by opening the Azure Virtual Machine in the Azure Portal (step 12).



Exercise 3: Get started with Docker

In this task you will login to the virtual machine and explore basic docker commands.

1. Use **ssh** to connect to the virtual machine using the following command. Change the name of your virtual machine in the script as required. Ignore the authenticity warning, if prompted, by confirming **'yes'** to continue connecting to server. Provide the **password** of demouser, when prompted.

```
ssh demouser@<public-ip-address>
```

2. To verify Docker is installed, use the following command:

```
sudo docker info
```

```
demouser@azure-docker-vm:~$ sudo docker info
Containers: 7
Images: 83
Storage Driver: devicemapper
 Pool Name: docker-8:1-136891-pool
 Pool Blocksize: 65.54 kB
 Backing Filesystem: extfs
 Data file: /dev/loop0
 Metadata file: /dev/loop1
 Data Space Used: 1.917 GB
 Data Space Total: 107.4 GB
 Data Space Available: 27.17 GB
 Metadata Space Used: 4.108 MB
 Metadata Space Total: 2.147 GB
 Metadata Space Available: 2.143 GB
 Udev Sync Supported: false
 Data loop file: /var/lib/docker/devicemapper/devicemapper/data
 Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
 Library Version: 1.02.82-git (2013-10-04)
Execution Driver: native-0.2
Kernel Version: 3.16.0-49-generic
Operating System: Ubuntu 14.04.3 LTS
CPUs: 2
```

3. Use docker version command to get information on currently installed Docker client and daemon

```
sudo docker version
```

```
demouser@azure-docker-vm:~$ sudo docker version
Client version: 1.6.2-cs5
Client API version: 1.18
Go version (client): go1.4.2
Git commit (client): 6f88399
OS/Arch (client): linux/amd64
Server version: 1.6.2-cs5
Server API version: 1.18
Go version (server): go1.4.2
Git commit (server): 6f88399
OS/Arch (server): linux/amd64
demouser@azure-docker-vm:~$
```

4. Use following commands to learn about working with containers.

```
# Start a new container. If container image was not present, it will download the image.
# -d option will Run container in background and return container ID
# You are running command /bin/sh from ubuntu image
JOB=$(sudo docker run -d ubuntu /bin/sh -c "while true; do echo Hello world; sleep 1; done")

# Print ID of container just created
echo $JOB

# List all the running containers. You will see the above container running.
sudo docker ps

# Stop the container
sudo docker stop $JOB

# List all the running containers. You will not see the above container, as it is stopped.
sudo docker ps

# List all containers, including the stopped ones.
sudo docker ps -a

# Start the container
sudo docker start $JOB

# Restart the container
sudo docker restart $JOB

# SIGKILL a container
sudo docker kill $JOB

# Remove a container (stop it first and then remove)
sudo docker stop $JOB
sudo docker rm $JOB
```

Exercise 4: Create and Dockerize your Application

In this exercise you will create a sample application and explore some functionality available in Dockerfile.

1. Clone the sample java application from GitHub.

For this exercise, you will use a small Java application and explore how to Dockerize that application. Clone the Git repository with the following command.

```
git clone https://github.com/danielbergamin/javahelloworld.git
```

2. Switch to the javahelloworld directory.

```
cd javahelloworld
```

3. Examine the content of the HelloWorld.java file

```
cat HelloWorld.java
```

You can see that this application prints Hello World and the current user name.

```
demouser@azure-docker-vm:~/javahelloworld$ cat HelloWorld.java
public class HelloWorld {
    public static void main (String [] args) {
        String username = System.getProperty("user.name");
        System.out.println("Hello World!");
        System.out.println("Running as user: " + username);
    }
}
```

4. Create Dockerfile and edit content

```
touch Dockerfile
```

```
vim Dockerfile
```

To start inserting text in vim, press the **i** key to enter insert mode.

Enter following content to the Dockerfile.

```
FROM java:7

COPY HelloWorld.java /

RUN javac HelloWorld.java

ENTRYPOINT ["java", "HelloWorld"]
```

Once you have finished entering the content, press **ESC** to exit the insert mode, then type **:wq** and press **Enter** to save and exit from the vim editor.

Here is a more descriptive version with comments and help links.

```
# Use official java 7 image as starting point
# This uses Linux-based OS with Java7 pre-installed
# http://docs.docker.com/reference/builder/#from
FROM java:7

# Copy HelloWorld.java file into the container
# http://docs.docker.com/reference/builder/#copy
COPY HelloWorld.java /

# Run shell command to compile HelloWorld.java upon starting the container
# http://docs.docker.com/reference/builder/#run
RUN javac HelloWorld.java

# Use ENTRYPOINT to set the default executable
# ENTRYPOINT ["executable", "param1", "param2"]
# http://docs.docker.com/reference/builder/#entrypoint
ENTRYPOINT ["java", "HelloWorld"]
```

5. Build the Dockerfile

Build a new image with name javahelloworld and tag 1.0

```
sudo docker build -t javahelloworld:1.0 .
```

6. Run the container with the javahelloworld image

```
sudo docker run javahelloworld:1.0
```

Here container is launched straight into the java application.

```
demouser@azure-docker-vm:~/javahelloworld$ sudo docker run javahelloworld:1.0
Hello World!
Running as user: root
```

7. Change container entry point to shell and look inside the container

Docker allows to override the native entry point for the image. Use following command to launch bash terminal as the entry point.

```
sudo docker run -it --entrypoint bash javahelloworld:1.0
```

Take a look at the files inside the container, run the HelloWorld application and terminate the container with exit command.

```
ls

java HelloWorld

exit
```

```

demouser@azure-docker-vm:~/javahelloworld$ sudo docker run -it --entrypoint bash javahelloworld:1.0
root@acccf4c2303e:/# ls
HelloWorld.class  bin    dev    home  lib64  mnt    proc  run    srv    tmp    var
HelloWorld.java   boot  etc    lib    media  opt    root  sbin   sys    usr
root@acccf4c2303e:/# java HelloWorld
Hello World!
Running as user: root
root@acccf4c2303e:/# exit
exit

```

Exercise 5: Manage multi container application with Docker Compose

In this exercise you will use compose to define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running.

1. Clone the sample application from GitHub.

For this exercise, you will use another Java application, which depends on Redis. Clone the repository and explore the contents of the Git repository. Run 'cd' as the first command to switch to home directory.

```

cd

git clone https://github.com/danielbergamin/HelloRedis.git

cd HelloRedis

ls

cat Dockerfile

```

```

demouser@azure-docker-vm:~$ cd HelloRedis
demouser@azure-docker-vm:~/HelloRedis$ ls
docker-compose.yml  Dockerfile  lib  README.md  src
demouser@azure-docker-vm:~/HelloRedis$ cat Dockerfile
FROM java:7
COPY /src /HelloRedis/src
COPY /lib /HelloRedis/lib

WORKDIR /HelloRedis
RUN javac -cp lib/jedis-2.1.0-sources.jar -d . src/HelloRedis.java

CMD ["java", "HelloRedis"]

```

2. Build the docker image and name it helloredis

```

sudo docker build -t helloredis .

```

3. Run the sample helloredis application

HelloRedis application depends on redis, an open source key-value store that functions as a data structure server). Start a redis container with name redisdb. Refer: https://hub.docker.com/_/redis/

```

sudo docker run -d --name redisdb redis

```

Run helloredis image and link it to the redis container. The link command is used to connect two containers.

```
sudo docker run --link redisdb:redisdb helloredis
```

```
demouser@azure-docker-vm:~/HelloRedis$ sudo docker run --link redisdb:redisdb helloredis
Linked to redis via hosts entry for redisdb: PONG
Linked to redis via hosts entry for redisdb: PONG
Linked to redis via hosts entry for redisdb: PONG
Linked to redis via hosts entry for redisdb: PONG
```

Press Ctrl+C to terminate the helloredis container.

Run docker stop to terminate the redis container.

```
sudo docker stop redisdb
```

Check if any containers are still running.

```
sudo docker ps
```

4. Run the sample helloredis application using compose

You executed several commands to start and stop the two container application (redis & helloredis) with only one link. Consider the amount of commands to run if you had multiple applications. Docker compose helps us to manage this easier.

Compose is a tool for defining and running multi-container applications with Docker. With Compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running. You can find more details at

<http://docs.docker.com/compose/>.

To install compose on the Docker Trusted Registry instance run the following commands:

```
sudo apt-get install python-pip

sudo pip install -U docker-compose==1.3.3
```

Examine the docker-compose.yml file

```
cat docker-compose.yml
```

```
demouser@azure-docker-vm:~/HelloRedis$ cat docker-compose.yml
javaclient:
  build: .
  links:
    - redis:redisdb
redis:
  image: redis
```

Breaking down compose file: For more details refer: <https://docs.docker.com/compose/yml/>

Command	Function
javaclient:	Defines a new service called javaclient

build: .	Path to the directory containing Dockerfile. When path is supplied as a relative path, it is interpreted as relative to the local of the yml file itself. In this case '.' will cause it to look for Dockerfile in the same path as the yml.
links:	Specifies which services to link the defined parent service to.
- redis:redisdb	Specifies the service to link to the form service:alias. Service is the name of the service defined in the Dockerfile, alias will be the name on the hosts file entry created. A service may consist of multiple instances when scaled, each one being one container.
redis:	Defines a new service called redis
image:	Specifies which image should be used for the service.

Start the application using compose.

```
sudo docker-compose up
```

Press **Ctrl+C** to terminate the applications and containers.

5. Manage a compose application

Start the compose application in detached mode

```
sudo docker-compose up -d
```

You can manage the containers created using docker-compose, with **docker**. Try out the following commands.

```
sudo docker ps

sudo docker logs helloredis_redis_1

sudo docker logs helloredis_javaclient_1
```

docker-compose, supports commands similar to docker for managing containers (ps, logs, ...). However, docker-compose only work with the containers that are defined in the docker-compose.yml file found in the working directory. Command with return error, if it can't find docker-compose files in the current directory. You can find all supported commands for docker-compose, with --help option.

```

demouser@azure-docker-vm:~/HelloRedis$ docker-compose --help
Define and run multi-container applications with Docker.

Usage:
  docker-compose [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE           Specify an alternate compose file (default: docker-compose.yml)
  -p, --project-name NAME    Specify an alternate project name (default: directory name)
  --verbose                  Show more output
  -v, --version              Print version and exit

Commands:
  build                      Build or rebuild services
  help                      Get help on a command
  kill                      Kill containers
  logs                      View output from containers
  port                      Print the public port for a port binding
  ps                         List containers
  pull                      Pulls service images
  restart                   Restart services
  rm                        Remove stopped containers
  run                      Run a one-off command
  scale                     Set number of containers for a service
  start                     Start services
  stop                      Stop services
  up                       Create and start containers
  migrate-to-labels         Recreate containers to add labels
demouser@azure-docker-vm:~/HelloRedis$

```

Execute following commands:

```

sudo docker-compose ps

cd ..

sudo docker-compose ps

cd HelloRedis

sudo docker-compose logs

# Use Ctrl+C to exit

```

```

demouser@azure-docker-vm:~/HelloRedis$ sudo docker-compose ps

```

Name	Command	State	Ports
helloworld_javaclient_1	java HelloRedis	Up	
helloworld_redis_1	/entrypoint.sh redis-server	Up	6379/tcp

```

demouser@azure-docker-vm:~/HelloRedis$ cd ..
demouser@azure-docker-vm:~$ sudo docker-compose ps
Can't find a suitable configuration file in this directory or any parent. Are you in the right directory?

Supported filenames: docker-compose.yml, docker-compose.yaml, fia.yml, fia.yaml
demouser@azure-docker-vm:~$ cd HelloRedis/
demouser@azure-docker-vm:~/HelloRedis$ sudo docker-compose logs
Attaching to helloworld_javaclient_1, helloworld_redis_1
javaclient_1 | Linked to redis via hosts entry for redisdb: PONG
javaclient_1 | Linked to redis via hosts entry for redisdb: PONG

```

docker-compose do not support all the management commands of docker. The docker-compose application potentially has multiple containers running. The attach command — which connects to the stdin/out of a container — would be ambiguous here, and as such, it is not supported.

```

sudo docker-compose attach

```

Terminate the application (both containers) with one command:

```

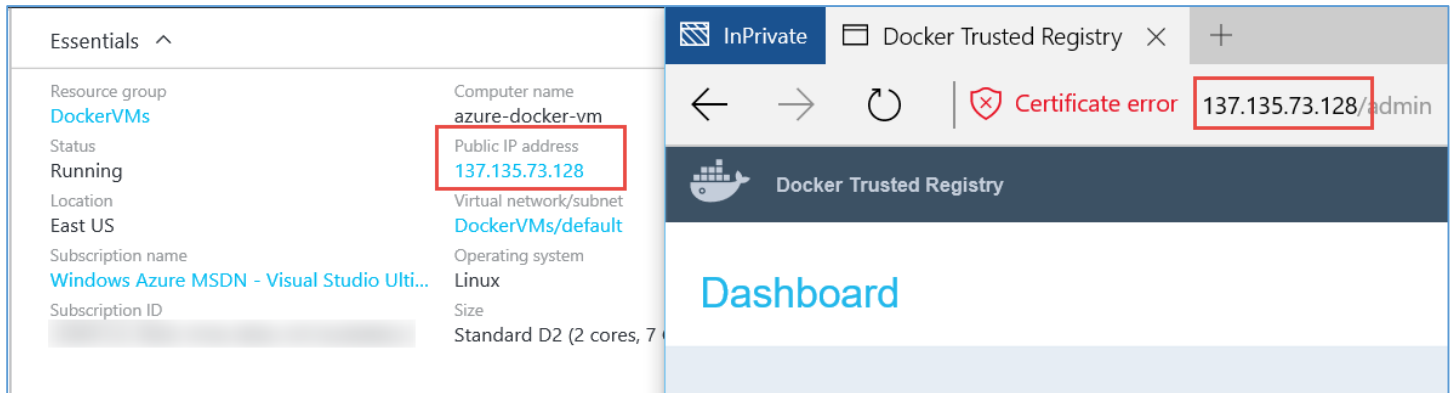
sudo docker-compose stop

```

Validate Lab Completion

Create a screenshot that shows the essentials panel from within the Azure Portal of your Docker Trusted Registry virtual machine instance as well as a screenshot of the deployed Docker Trusted Registry instance with the Dashboard page shown. Both the essentials panel and the Docker Trusted Registry should show the IP address.

Please save your lab screenshots as either a .jpeg or .png. Upload your screenshots in one .zip file, [here](#).



A screenshot of an SSH session connecting to the Docker Trusted Registry instance using the same IP address as the previous screenshots and executing the following commands should be provided.

```
ssh demouser@<public ip address>
cd HelloRedis
```

```
$ ssh demouser@137.135.73.128
demouser@137.135.73.128's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.16.0-49-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Tue Nov  3 00:31:48 UTC 2015

System load:  0.67           Processes:            291
Usage of /:   12.1% of 28.80GB Users logged in:        0
Memory usage: 3%            IP address for eth0:  10.1.0.4
Swap usage:   0%            IP address for docker0: 172.17.42.1

Graph this data and manage this system at:
https://landscape.canonical.com/

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

Last login: Tue Nov  3 00:31:48 2015 from 167.220.98.213
demouser@azure-docker-vm:~$ cd HelloRedis/
demouser@azure-docker-vm:~/HelloRedis$
```

Lab Summary

In this lab you have created a Docker enabled virtual machine from the Microsoft Azure Marketplace and explored basic Docker commands, Dockerize your application and used the compose feature to manage a multi-container application.