# 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: o Git Bash with SSH client from <http://www.git-scm.com/downloads>o [PuTTY,](http://www.putty.org/)  o [AnyConnect](http://anyconnect.net/Default.aspx)
* Docker evaluation license file: <http://emails.microsoft.com/o00sW00Oh00mmG1TTEQEqq0>

## 

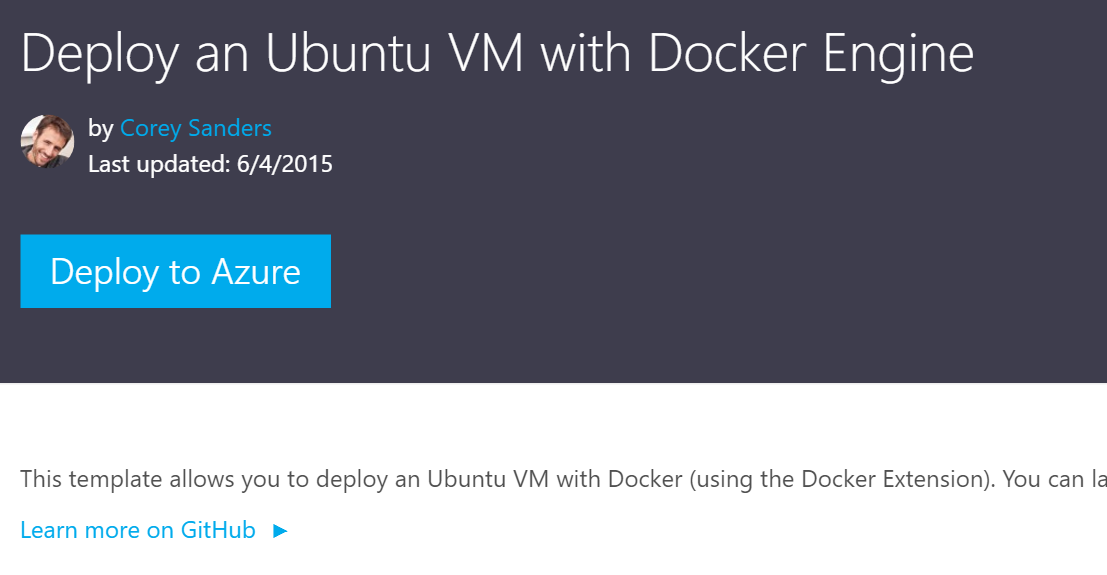
# Create the Docker Host VM in Azure using the CLI

First, we will configure a Linux machine in azure with the Docker extensions to function as our Docker Host. There are several ways to accomplish this task.

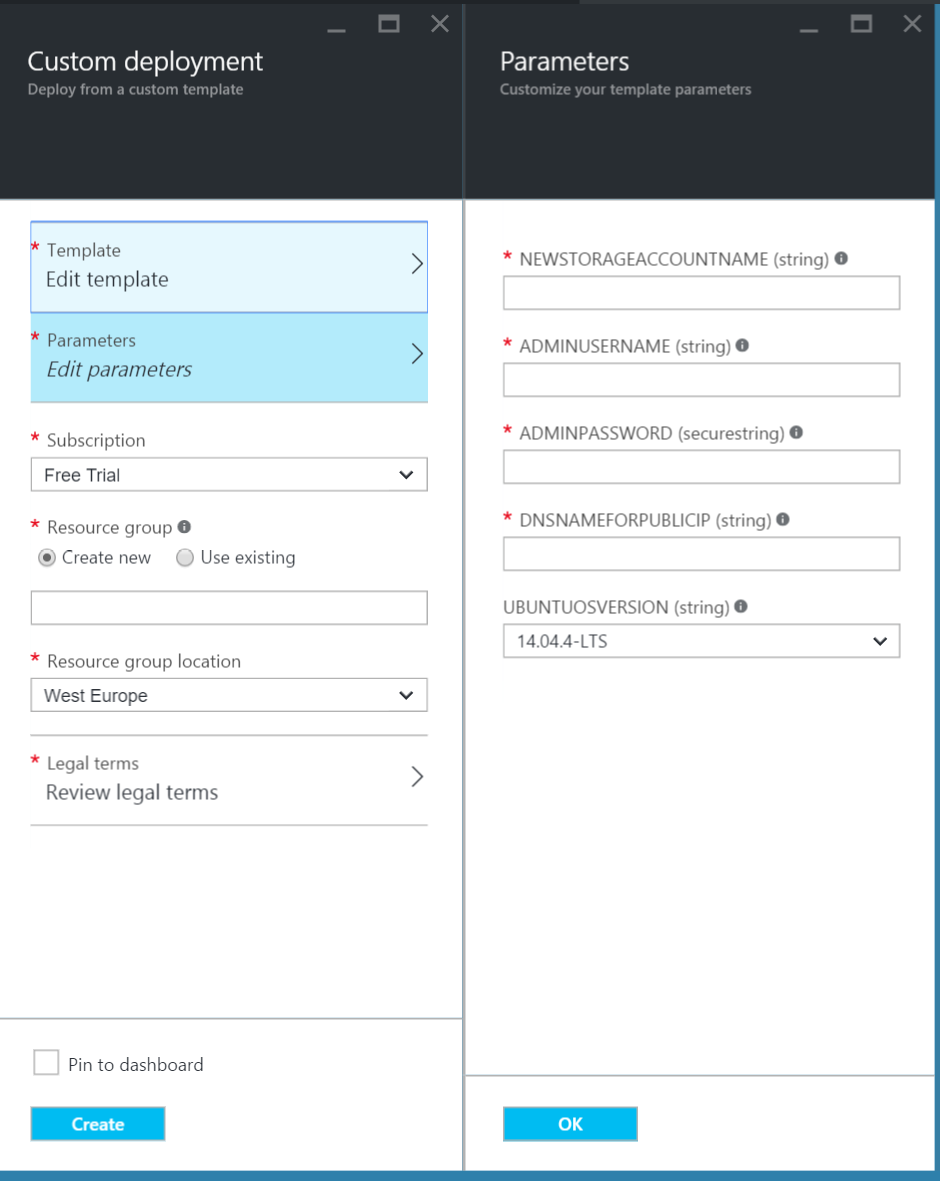
## Deploy directly from the portal

You can use the docker-simple-on-ubuntu quick template and deploy directly from the Azure Portal :

Go to <https://azure.microsoft.com/en-us/documentation/templates/docker-simple-on-ubuntu/> and click the “Deploy to Azure” button.



This starts a custom deployment based on this Microsoft template, written by Corey Sanders. Just fill out the required parameters and click create.



**When the docker VM host has been deployed correctly, please continue to the step : Deploy your first container.**

## **Alternatively use the Azure CLI to deploy this Docker Host environment**

You can use the Azure CLI to deploy this host with the azure group command, making use of the same template on github.

First delete the resource group created in the previous exercises.

* Important: all resources you previously created will be lost.

View all resource groups in your subscription

azure group list

Then delete the specific resource group:

azure group delete [resource group name]

## **Create the docker host using the CLI**

****azure group create --name accelerate[uniqueid] --location "West Europe" --template-uri**** <https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/docker-simple-on-ubuntu/azuredeploy.json>

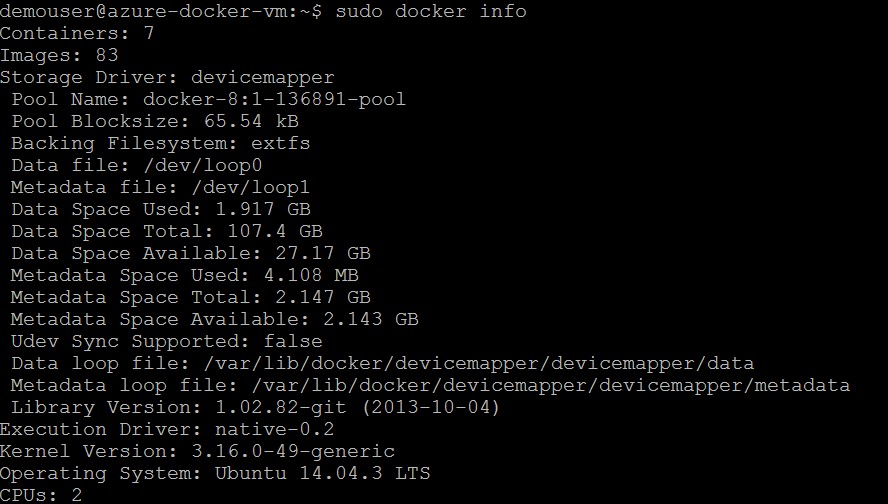
## Get started with Docker

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>**

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



sudo docker info

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

sudo docker version



1. 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

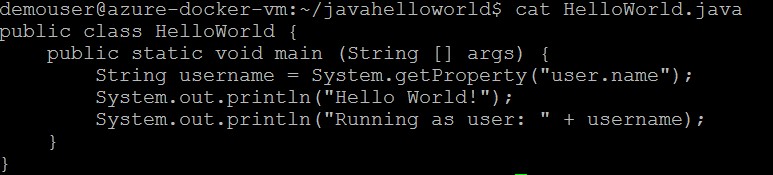
1. Switch to the javahelloworld directory.

cd javahelloworld

1. 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.



1. 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

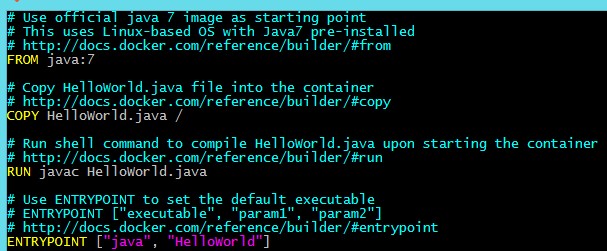
COPY HelloWorld.java /

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.



1. Build the Dockerfile

Build a new image with name javahelloworld

sudo docker build javahelloworld

1. Run the container with the javahelloworld image

sudo docker run javahelloworld

Here container is launched straight into the java application.



1. 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.

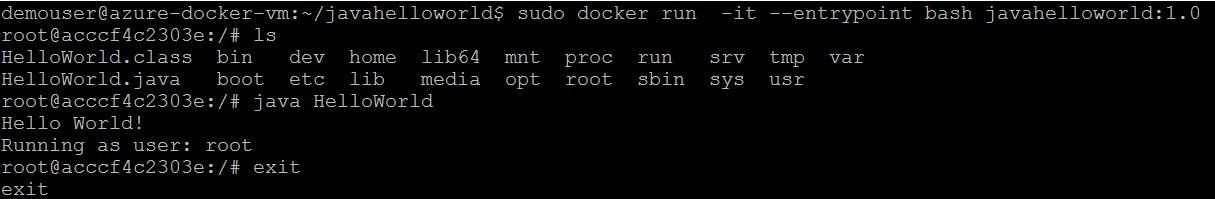
sudo docker run -it --entrypoint bash javahelloworld:1.0

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

ls

java HelloWorld

exit



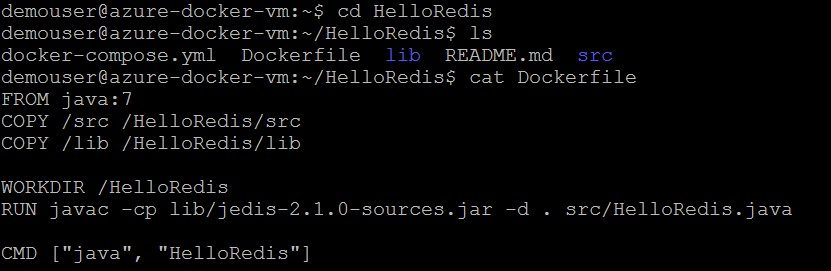
## Exercise 5: Manage multi container application with Docker Compose

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 |



1. Build the docker image and name it helloredis

sudo docker build -t helloredis .

1. Run the sample helloredis application

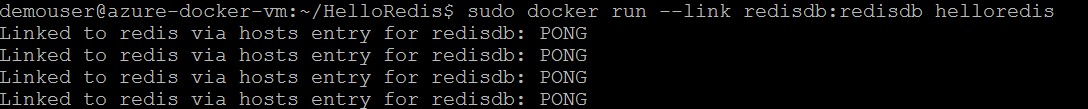
HelloRedis application depends on redis, an open source key-value store that functions as a data structure server).

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



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

1. 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/.](http://docs.docker.com/compose/)

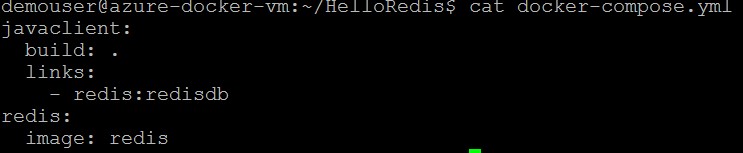
To install compose on the Docker Trusted Registry instance run the following commands:

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



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

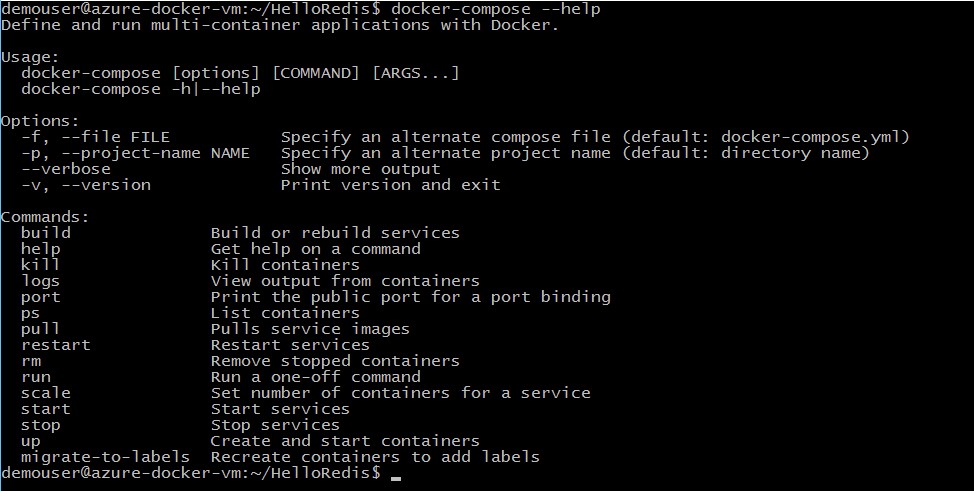
You can manage the containers created using docker-compose, with **docker**. Try out the following commands.

sudo docker ps

sudo docker logs helloredis\_redis\_1

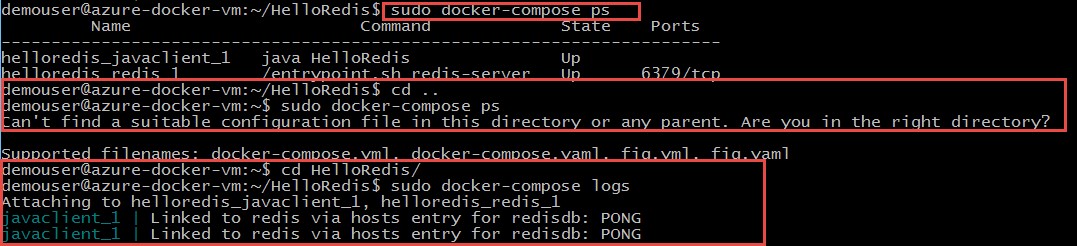
sudo docker logs helloredis\_javaclient\_1

docker-compose, supports commands similar to docker for managing containers (ps, logs, …). However, dockercompose 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.



Execute following commands:

|  |
| --- |
| sudo docker-compose ps  cd ..    sudo docker-compose ps    cd HelloRedis    sudo docker-compose logs    # Use Ctrl+C to exit |



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:

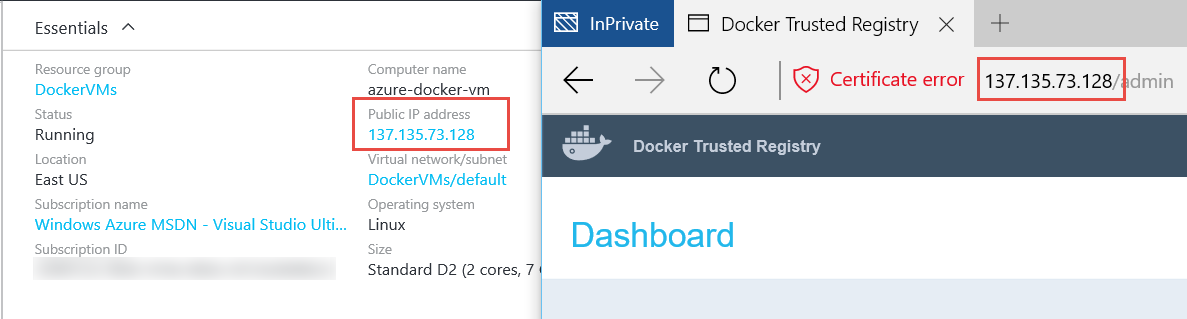
sudo docker-compose stop

## 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.

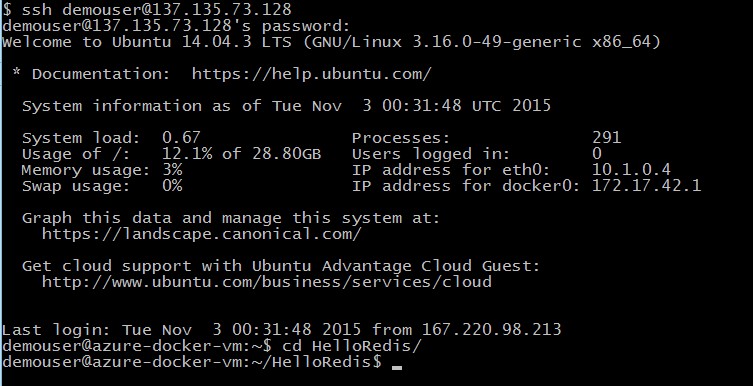
Please save your lab screenshots as either a .jpeg or .png. Upload your screenshots in one .zip file, [here.](http://www.tryazuremarketplace.com/docker/submit)



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



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.