IBM Cloud

Introduction to Docker

Hands-on Workshop

Lab Guide

Notices and Disclaimers

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

Document Revision History

Rev #	File Name	Date
1.0	Docker and Kubernetes - Lab	1/21/2018

Prepared & Revised by:
Louis V. Frolio - louis.frolio@ibm.com
David Solomon - dlsolomo@us.ibm.com

Table of Contents

Lab Environment Overview

Installed Software and Tools

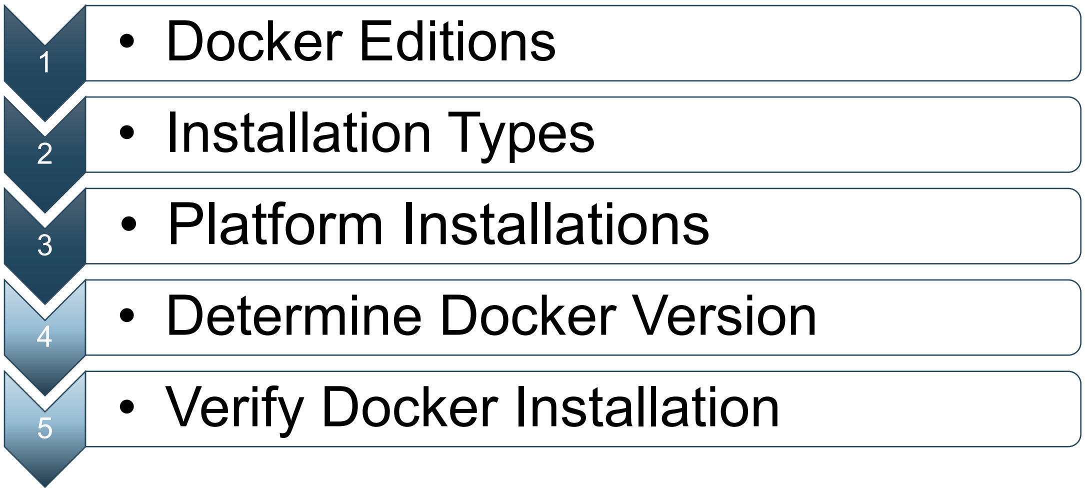
Software	Link
Docker	https://www.docker.com
VirtualBox	https://www.virtualbox.org/wiki/Downloads
Minikube	https://kubernetes.io/docs/getting-started-guides/minikube/
Docker Hub	https://hub.docker.com/

Section 1: Pre-requisites

Purpose:	This section introduces Docker editions, and installation types along with guidance on how to install Docker on its supported platforms. You will determine the version of Docker running on your laptop and then verify that Docker is running without issues.

Tasks:	Docker software for: • Discuss Docker Editions • Review supported platforms • Gather information about Docker installation • Verify Docker is running without issue

Section 1: Lab Workflow Overview



1
- Docker Editions

2
- Installation Types

3
- Platform Installations

4
- Determine Docker Version

5
- Verify Docker Installation

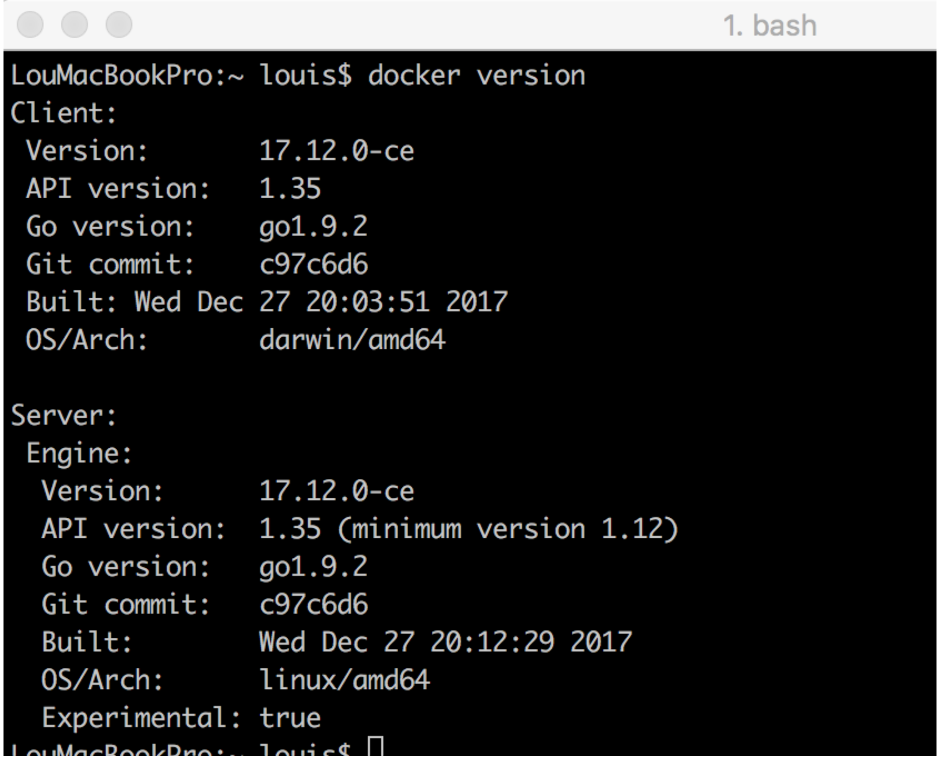
Section 1: Lab Instructions

Step	Action
1	**Docker Editions** • Community Edition (CE) Free, quarterly release cadence, no premium support (community). **This lab uses Docker CE** • Enterprise Edition (EE) Not free, quarterly release cadence, premium support available Certified on specific platforms Extra products • General Availability (GA) vs Beta (Edge) **GA** is the stable production ready release of Docker; quarterly cadence for CE and EE **Edge** is the monthly beta release, gets new functionality first. Each edge release is supported for the month, once new edge release is released prior edge cannot get support. Aggregate edge functionally changes roll up into EE GA release quarterly.
2	**Installation Types** • **Direct** Direct installation on a supported operating system. E.g. Linux, RaspPI, Mainframe, Windows Server 2016 • **Mac/Windows 10** Natively does not support “direct” installation of Docker. Small VM (transparent) is spun up to run the containers in. • **Cloud** IBM Cloud, AWS, Azure, Google Cloud Usually have proprietary features specific to cloud
3	**Platform Installations** • **Mac** Install Docker for Mac. For older Mac’s with less than OSX Yosemite 10.10.3 install the Docker Toolbox • **Windows 10 Pro/Enterprise** Install "Docker for Windows" from the Docker Store • **Windows 7, 8, 10 Home** Install Docker Toolbox. Lack of Hyper-V necessitates this type of installation

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4</td><td>

<u>Determine Docker Version</u>

a. Open terminal (MAC), Shell (Windows), or Quickstart Terminal (Docker Toolbox) then type:
~$ docker version



```
LouMacBookPro:~ louis$ docker version
Client:
 Version:       17.12.0-ce
 API version:   1.35
 Go version:    go1.9.2
 Git commit:    c97c6d6
 Built: Wed Dec 27 20:03:51 2017
 OS/Arch:       darwin/amd64

Server:
 Engine:
  Version:      17.12.0-ce
  API version:  1.35 (minimum version 1.12)
  Go version:   go1.9.2
  Git commit:   c97c6d6
  Built:        Wed Dec 27 20:12:29 2017
  OS/Arch:      linux/amd64
  Experimental: true
LouMacBookPro:~ louis$
```

b. You output should be similar.

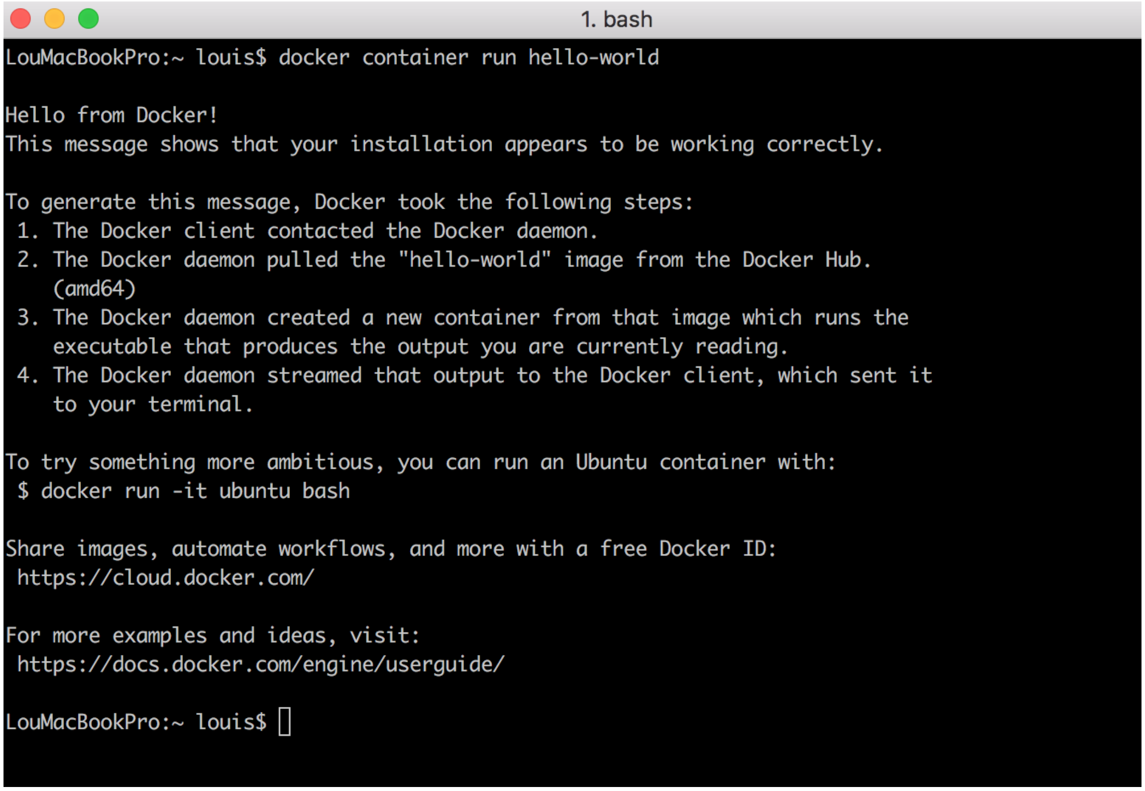
</td></tr>
<tr><td>5</td><td>

<u>Verify Docker Installation</u>

a. Run test Docker container
~$ docker container run hello-world

</td></tr>
</table>

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td></td><td>



```
LouMacBookPro:~ louis$ docker container run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/

LouMacBookPro:~ louis$ 
```

Output verifies that Docker is running and you are able to pull images from Docker Hub, and then start a container from the image.

</td></tr>
</table>

Section 1: Lab Summary

In this lab, you learned about Docker Editions, installation types, and various platforms supported by Docker. You also ran Docker commands to determine the version of Docker, and to verify that the installation was successful.

Other useful Docker commands:

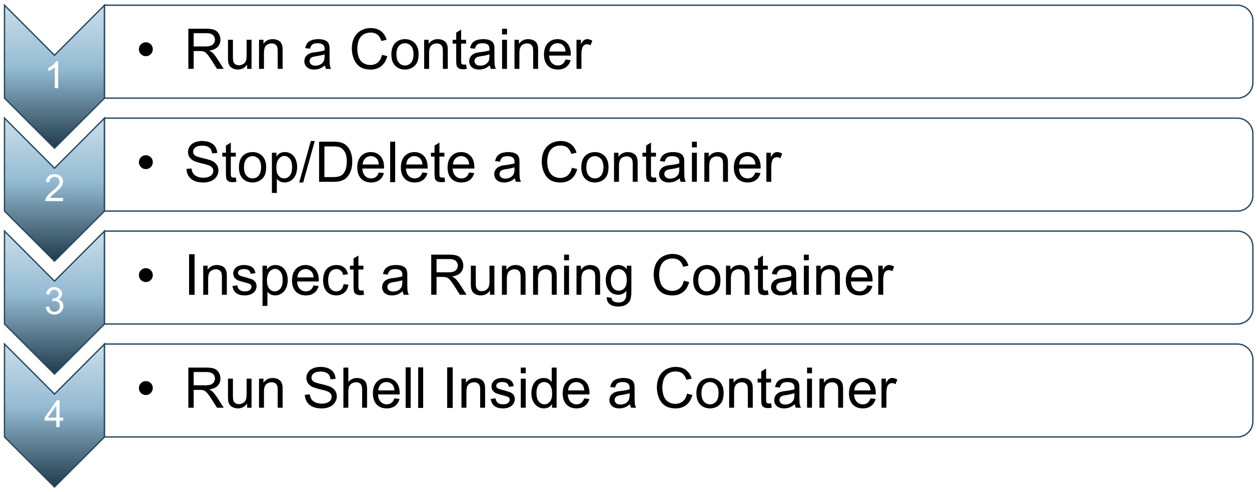
```
~$ docker Info - display Docker system-wide information
~$ docker help – display help topics available
```

Section 2: Container Basics

Purpose:	This lab introduces container basics. You will learn how to run, inspect and manage multiple containers. Also, you will work through establishing console access within the container.

Tasks:	Tasks you will complete in this lab exercise include: • Running containers • Inspecting containers • Container process monitoring • Container shell access

Section 2: Lab Workflow Overview

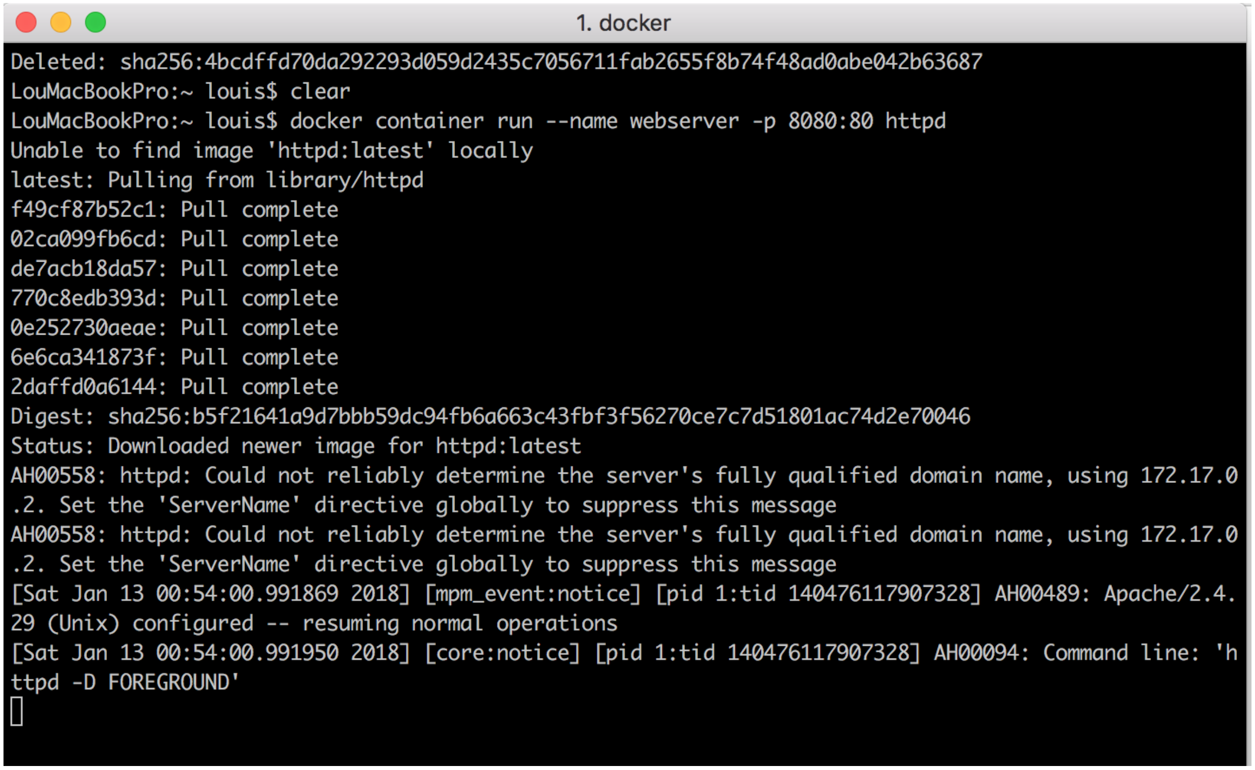


1. Run a Container
2. Stop/Delete a Container
3. Inspect a Running Container
4. Run Shell Inside a Container

Section 2: Lab Instructions

Step	Action
1	**Run a Container**

a. In a terminal window type the following:
~$ docker container run --name webserver -p 8080:80 httpd



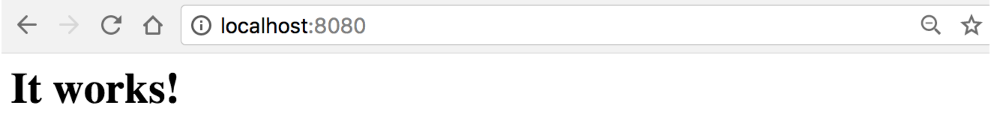
```
Deleted: sha256:4bcdffd70da292293d059d2435c7056711fab2655f8b74f48ad0abe042b63687
LouMacBookPro:~ louis$ clear
LouMacBookPro:~ louis$ docker container run --name webserver -p 8080:80 httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
f49cf87b52c1: Pull complete
02ca099fb6cd: Pull complete
de7acb18da57: Pull complete
770c8edb393d: Pull complete
0e252730aeae: Pull complete
6e6ca341873f: Pull complete
2daffd0a6144: Pull complete
Digest: sha256:b5f21641a9d7bbb59dc94fb6a663c43fbf3f56270ce7c7d51801ac74d2e70046
Status: Downloaded newer image for httpd:latest
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0
.2. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0
.2. Set the 'ServerName' directive globally to suppress this message
[Sat Jan 13 00:54:00.991869 2018] [mpm_event:notice] [pid 1:tid 140476117907328] AH00489: Apache/2.4.
29 (Unix) configured -- resuming normal operations
[Sat Jan 13 00:54:00.991950 2018] [core:notice] [pid 1:tid 140476117907328] AH00094: Command line: 'h
ttpd -D FOREGROUND'
```

The container “webserver” is an instance of the image “httpd” running as a process. The image is pulled (first time) from the default Docker registry called Docker Hub.
There is no limit to the number of containers that can be run from an image.

Commands:
--name – Specify a unique name for the container service. If omitted Docker will create a random, human readable name.
-p – Specify that the container internal port (80) be exposed to port 8080 on the host.

b. Open a browser and navigate to: localhost:8080. You should be presented with a message returned from the web browser “It Works”

Step	Action



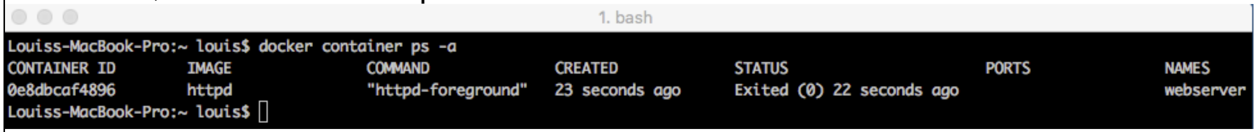
It works!

2 **Stop/Delete a Container**

a. You stop the container by typing cntrl-c
~$ <Cntrl-c>

b. Verify that the container is no longer running:
~$ docker container ps

c. Although the container is not running it still exists:
~$ docker container ps -a



```
Louiss-MacBook-Pro:~ louis$ docker container ps -a
CONTAINER ID        IMAGE               COMMAND              CREATED             STATUS                      PORTS               NAMES
0e8dbcaf4896        httpd               "httpd-foreground"   23 seconds ago      Exited (0) 22 seconds ago                       webserver
Louiss-MacBook-Pro:~ louis$ 
```

-a, --all: Show all containers (default shows just running)

d. Remove the container:
~$ docker container rm webserver

Containers can be removed either by their name or container id

3 **Inspect a Running Container**

a. Run a new Docker container:
~$ docker run --publish 80:80 --detach --name webserver nginx

You should be brought back to the terminal prompt.

Step	Action

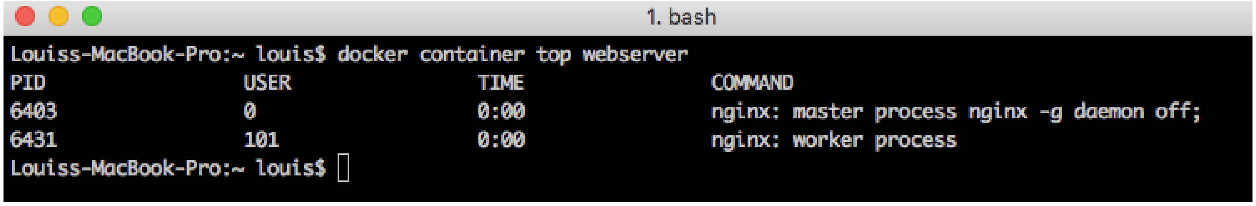
b. Open a browser and navigate to “localhost”. You should be prompted with “Welcome to nginx!”

c. Inspect the log file for the running container “webserver”:
~$ docker container logs webserver

Click refresh on the browser then re-run the “docker logs …” command. Notice that there is a new log entry for the event.

d. Examine the processes running in the container:
~$ docker container top webserver

The output shows two nginx processes running in the container. One is the master process, the other is a child process



```
Louiss-MacBook-Pro:~ louis$ docker container top webserver
PID                 USER                TIME                COMMAND
6403                0                   0:00                nginx: master process nginx -g daemon off;
6431                101                 0:00                nginx: worker process
Louiss-MacBook-Pro:~ louis$ 
```

e. Inspect meta-data for running container:
~$ docker container inspect webserver

f. Stream live performance container metrics:
~$ docker container stats webserver

g. Clean up
~$ docker container rm -f webserver

Commands:
-d, --detach - Run the container in the background.

4

Run Shell Inside a Container

a. Run container interactively:
~$ docker container run -it --name linuxlight alpine

Alpine is a lightweight linux distribution

Step	Action
	b. Run Linux commands in container: # ls -tal // List directories and files # df -h // List file systems and their usage # apk add --no-cache curl // Install curl # curl localhost # exit // Exit shel and stop container c. Start an existing (stopped) container & run Linux commands: ~$ docker container start -ai linuxlight # curl localhost # exit d. Access an already running container: ~$ docker container run -p 80:80 -d --name webserver nginx ~$ docker container ls -a ~$ docker container exec -it webserver bash # ls -tal # exit ~$ docker container ls -a Commands: -i - Run interactively -t - Create pseudo tty -a - Attach to STDIN, STDOUT or STDERR exec - Run a command in a running container run - Run a command in a new container

Section 2: Lab Summary

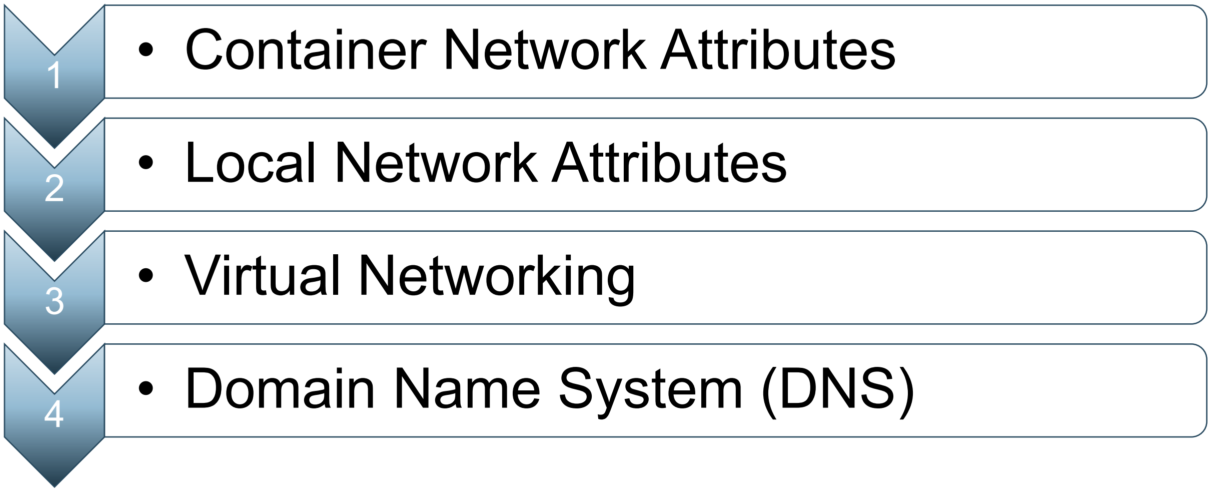
In this section you learned how to create new containers based on images stored in Docker Hub. You also learned how to interact with containers both from the outside (top, inspect, stats, …), and from the inside (docker exec and run). Access to the Docker service via tty was demonstrated and you learned how to run Linux commands inside the container just as if you were working with a Linux OS.

Section 3: Container Networking Basics

Purpose:	In this lab you will learn basic skills as it relates to networking (security & DNS) with Docker containers:

Tasks:	Tasks you will complete in this lab exercise include: • Determine active ports • Identify container ip address • Create virtual network • Attach container to virtual network

Section 3: Lab Workflow Overview



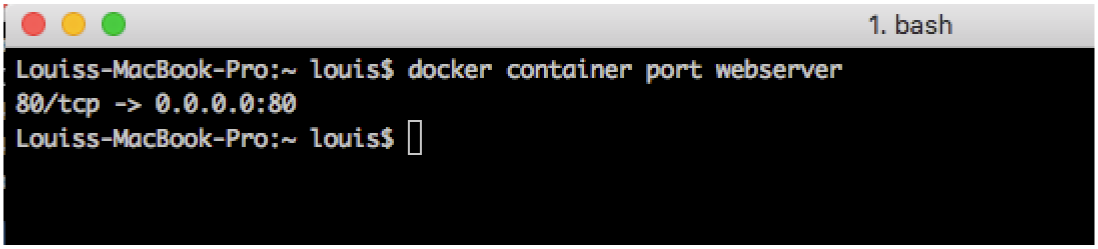
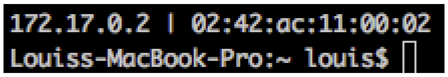
1
- Container Network Attributes

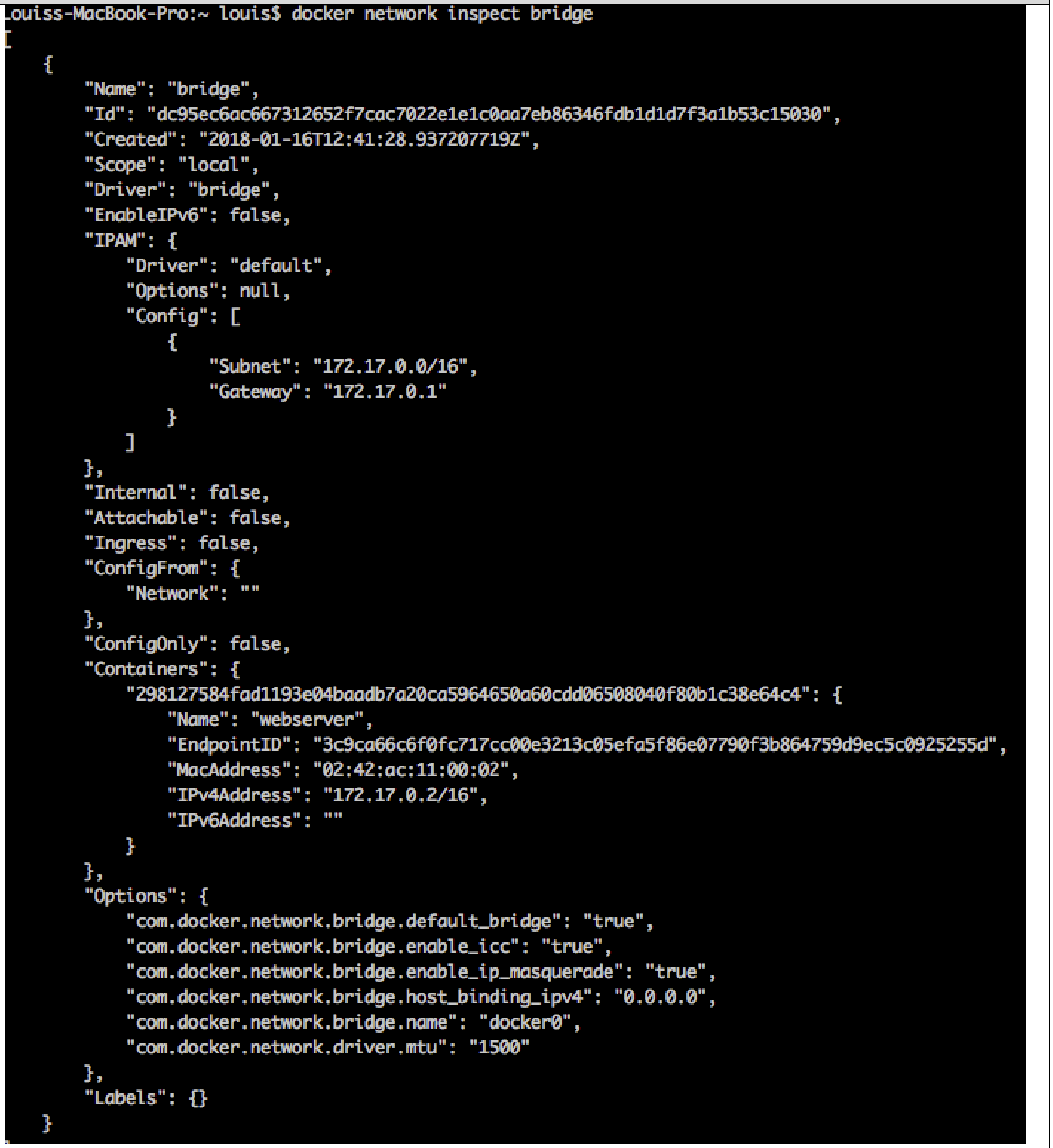
2
- Local Network Attributes

3
- Virtual Networking

4
- Domain Name System (DNS)

Section 3: Lab Instructions

Step	Action
1	**<u>Container Network Attributes</u>** a. On what port is container listening: ~$ docker container port webserver  1. bash Louiss-MacBook-Pro:~ louis$ docker container port webserver 80/tcp -> 0.0.0.0:80 Louiss-MacBook-Pro:~ louis$ b. Determine ip and mac address of container (Go templating): ~$ docker container inspect --format '{{.NetworkSettings.IPAddress }} | {{.NetworkSettings.MacAddress}}' webserver  172.17.0.2 | 02:42:ac:11:00:02 Louiss-MacBook-Pro:~ louis$
2	**<u>Local Network Attributes</u>** a. List all networks available to Docker: ~$ docker network ls Network named “bridge” is the default network available to Docker b. Get details for local network “bridge”: ~$ docker network inspect bridge Shows containers attached to the “bridge” network. “bridge” is the default network that bridges through NAT firewall to the physical network to which the host is connected.

Step	Action
	

```
Louiss-MacBook-Pro:~ louis$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "dc95ec6ac667312652f7cac7022e1e1c0aa7eb86346fdb1d1d7f3a1b53c15030",
        "Created": "2018-01-16T12:41:28.937207719Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "298127584fad1193e04baadb7a20ca5964650a60cdd06508040f80b1c38e64c4": {
                "Name": "webserver",
                "EndpointID": "3c9ca66c6f0fc717cc00e3213c05efa5f86e07790f3b864759d9ec5c0925255d",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
```

	c. Fill in
3	**Virtual Networking** a. Create Virtual Network:

Step	Action
	~$ docker network create mynetwork ~$ docker network ls b. Create new container and run it on “mynetwork”: ~$ docker container run -d --name webserver2 --network mynetwork nginx c. Verify new container is running on “mynetwork”: ~$ docker network inspect mynetwork d. Connect container from “bridge” to “mynetwork”: ~$ docker container inspect webserver | tail -30 ~$ docker network connect mynetwork webserver e. Ajfd;lak
4	**Domain Name System (DNS)** In a Docker environment you cannot have two containers with the same name. Further, because of the dynamic nature of Docker you cannot assign (nor would you want to) a static IP to a container. Instead, you rely on unique container names and reference those in your environment. This is what we have been doing throughout this tutorial. However, you may find that you want to load balance a workload across several containers, but you don’t want the hassle of having to choose. In this case Docker provides the ability to create a network alias that will permit two or more containers to be referenced by a network alias. a. Create a new virtual network called “icecream”: ~$ docker network create icecream ~$ docker network ls b. Create two Elasticsearch containers on the network “icecream”. We don’t specify a port for these, we will work within the virtual network. ~$ docker container run -d --net icecream --net-alias search elasticsearch:2 ~$ docker container run -d --net icecream --net-alias search elasticsearch:2 c. Run a DNS lookup to verify the two containers are on the same network, and share the same DNS name. ~$ docker container run --rm --net icecream alpine nslookup search

Step	Action
	d. Test that DNS round robin is working: ~$ docker container run --rm --net icecream centos curl -s search:9200 Run this command several times and you will see that the Elasticsearch server name will change. Due to DNS caching you may get the same server a few times in a row. e. Clean up: ~$ docker container rm -f <cont1> <cont2>

Section 3: Lab Summary

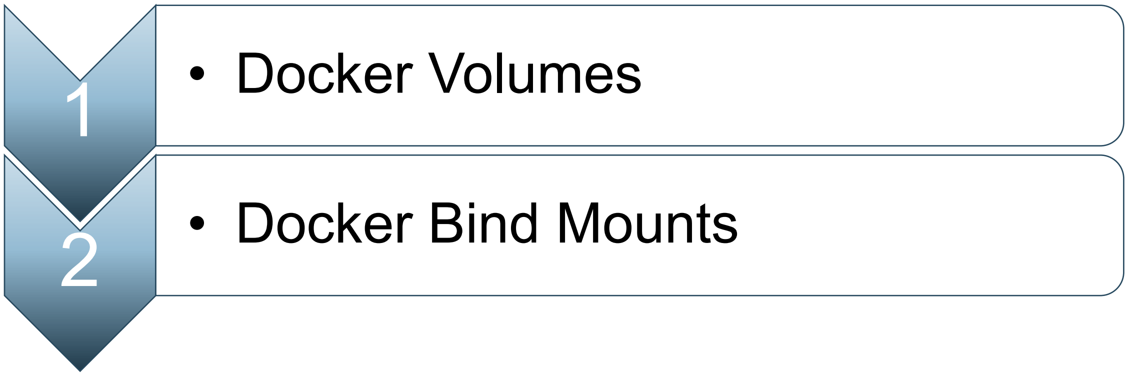
In this section you were introduce to virtual networking in Docker and how to create and manage them. Also, you learned how to create network aliases and how they can be inspected and used for load balancing

Section 4: Data Persistence in Docker

Purpose:	In this section you will learn the different ways in which Docker can handle and manage data persistence.

Tasks:	Tasks you will complete in this lab exercise include: • Create Docker volumes • Create Bind mounts • Work with volumes and bind mounts in containers

Section 4: Lab Workflow Overview



1

- Docker Volumes

2

- Docker Bind Mounts

Section 4: Lab Instructions

Step	Action
1	<u>**Docker Volumes**</u> a. Create MySQL container: ~$ docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True mysql b. Open bash shell on container create database objects: ~$ docker container exec -it mysql bash # mysql -u root mysql> show databases; mysql> create database apple; mysql> exit; # exit c. Stop, then restart container to verify that db changes still exist: ~$ docker container stop mysql ~$ docker container start mysql ~$ docker container exec -it mysql bash # mysql -u root mysql> show databases; mysql> exit; # exit ~$ docker container rm -f mysql d. Create a MySQL database and specify volume creation: ~$ docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql-db:/var/lib/mysql mysql ~$ docker container ls ~$ docker volume ls ~$ docker volume inspect mysql-db ~$ docker container inspect mysql ** Note that the command above is wrapped. There are no CRLF’s. ** The path specified is for a Mac, Windows will be different e. Open bash shell on container create database objects: ~$ docker container exec -it mysql bash # mysql -u root

Step	Action
	mysql> show databases; mysql> create database apple; mysql> exit; # exit f. Delete container “mysql” then recreate and verify that db changes remain: ~$ docker container rm -f mysql ~$ docker volume ls ~$ docker container run -d --name mysql3 -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql-db:/var/lib/mysql mysql ~$ docker container exec -it mysql3 bash # mysql -u root mysql> show databases; mysql> create database apple; mysql> exit; # exit g. Clean up ~$ docker container rm -f mysql3 ~$ docker volume rm mysql-db ~$ docker volume ls ~$ docker container ls -a Volumes are extremely useful for local development projects. You can maintain several volumes to which you can attach a new database that fits a specific purpose.
2	**Docker Bind Mounts** Bind mounts differ from volumes in that they are mappings from the host’s file or directory into a container file or directory. Like volumes, when you delete a container the data is not lost. a. Pull down in index.html from Github ~$ wget https://github.com/team-wolfpack/Docker-and-Kubernetes-Hands-On/blob/master/index.html Inspect the contents of the html file.

Step	Action
	b. Create nginx container and bind mount local directory to specified path within container: ~$ docker container run -d --name nginx -p 80:80 -v $(pwd):/usr/share/nginx/html nginx c. Verify that the correct html is being used: ~$ curl localhost You should see body text of the html file. d. Get to shell prompt in container and verify bind mount mapping ~$ docker container exec -it mysql bash # cd /user/share/nginx/html # cat index.html e. Open another terminal window on the host and change the index.html file. ~$ docker container exec -it mysql bash # cd /user/share/nginx/html # cat index.html

Section 4: Lab Summary

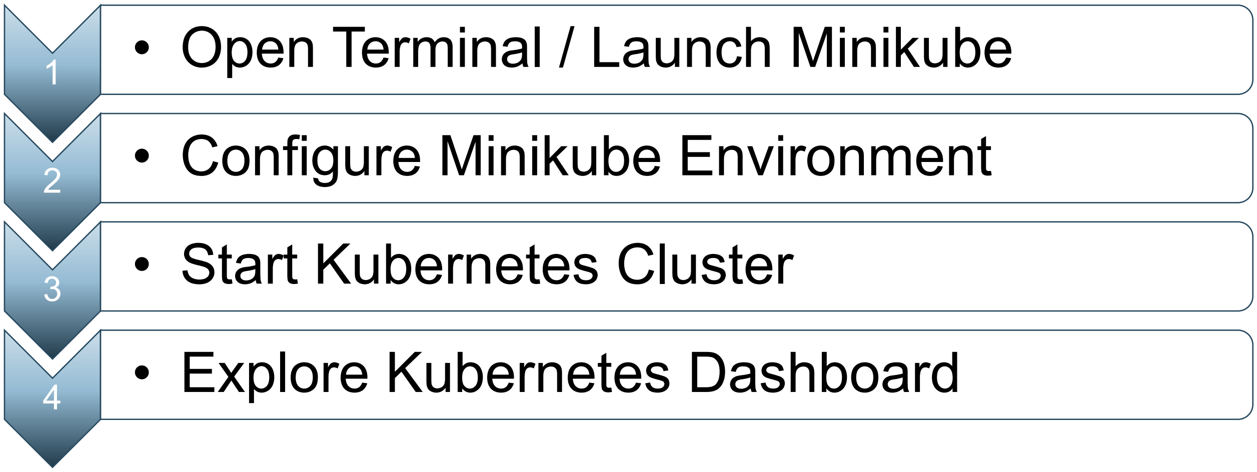
In this lab you were introduced to two approaches to persist data on the host file system. With volumes the container references a volume object on the local file system. Bind mounts create a common link between the local file system and the file system in the container. A bound mount can reference either a file or directory on the host file system.

Section 5: Getting Started with Minikube

Purpose:	In this lab you will learn basic skills as it relates to networking (security & DNS) with Docker containers:

Tasks:	Tasks you will complete in this lab exercise include: • Start Minikube • Configure Kubernetes • Launch & Explore Kubernetes Dashboard

Section 5: Getting Started with Minikube



1
- Open Terminal / Launch Minikube

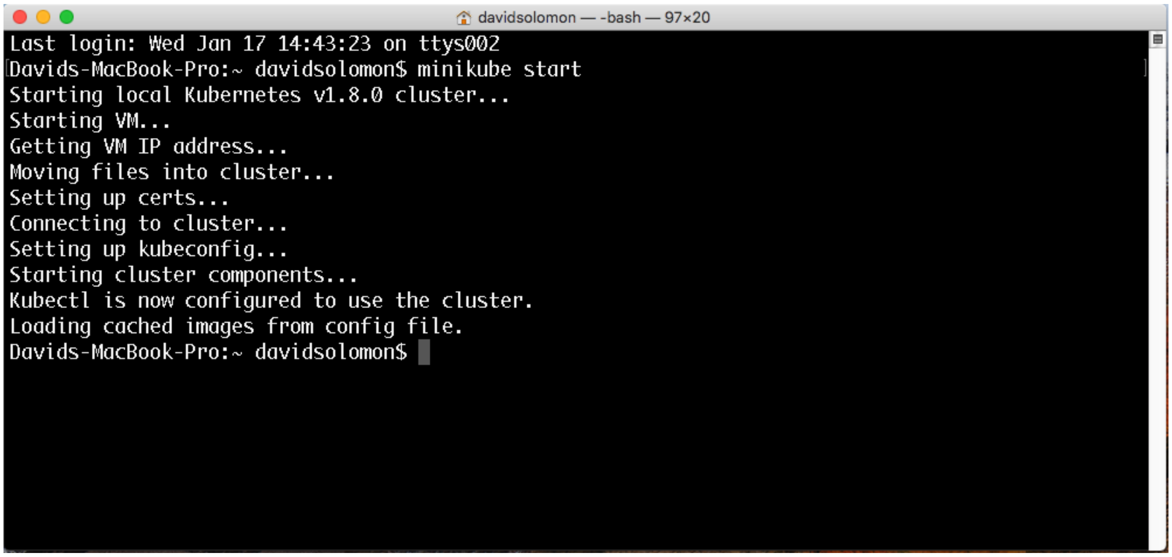
2
- Configure Minikube Environment

3
- Start Kubernetes Cluster

4
- Explore Kubernetes Dashboard

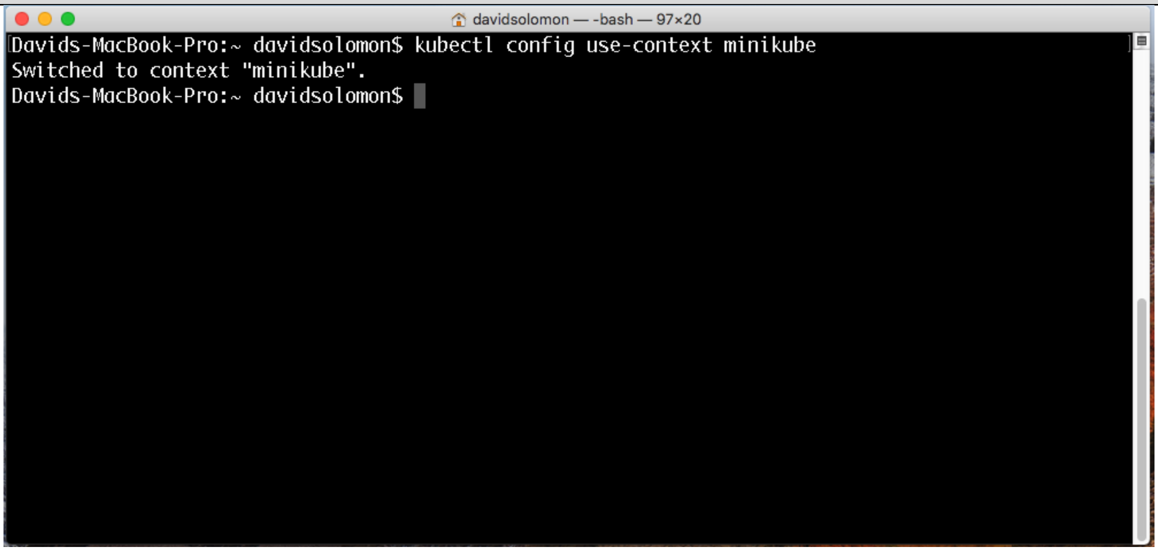
Section 5: Lab Instructions

Step	Action

1	**Start Kubernetes**
	a. Open a terminal window and type the following command. This will start the Kubernetes cluster.
	~$ minikube start
	
	`Last login: Wed Jan 17 14:43:23 on ttys002`
	`Davids-MacBook-Pro:~ davidsolomon$ minikube start`
	`Starting local Kubernetes v1.8.0 cluster...`
	`Starting VM...`
	`Getting VM IP address...`
	`Moving files into cluster...`
	`Setting up certs...`
	`Connecting to cluster...`
	`Setting up kubeconfig...`
	`Starting cluster components...`
	`Kubectl is now configured to use the cluster.`
	`Loading cached images from config file.`
	`Davids-MacBook-Pro:~ davidsolomon$`
	b. Confirm that Minikube has successfully started, as shown above.

2	**Configure the Kubernetes Environment** a. Run the following command to set the current Kubernetes environment to our local cluster (the default cluster name is "minikube") ~$ kubectl config use-context minikube b. Confirm the output is as shown below:

Step	Action



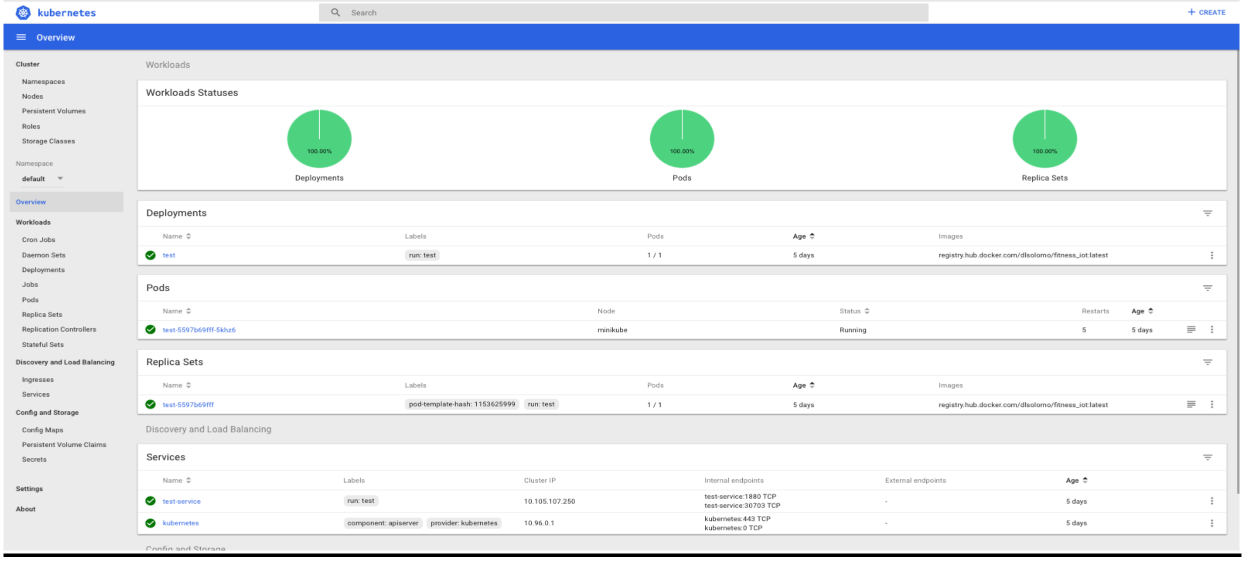
```
Davids-MacBook-Pro:~ davidsolomon$ kubectl config use-context minikube
Switched to context "minikube".
Davids-MacBook-Pro:~ davidsolomon$ 
```

3

Start the Kubernetes Dashboard

a. Start the Kubernetes web dashboard by entering the following command:
~$ minikube dashboard

b. This will open the dashboard in your default browser. You will see in the dashboard the status of any existing deployments, PODS, and services, as shown below:



Deployments

Name	Labels	Pods	Age	Images
test	run: test	1 / 1	5 days	registry.hub.docker.com/dlsolomo/fitness_iot:latest

Pods

Name	Node	Status	Restarts	Age
test-5597b69fff-5khz6	minikube	Running	5	5 days

Replica Sets

Name	Labels	Pods	Age	Images
test-5597b69fff	pod-template-hash: 1153625999 run: test	1 / 1	5 days	registry.hub.docker.com/dlsolomo/fitness_iot:latest

Discovery and Load Balancing

Services

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
test-service	run: test	10.105.107.250	test-service:1880 TCP test-service:30703 TCP	-	5 days
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	5 days

Config and Storage

Section 5: Lab Summary

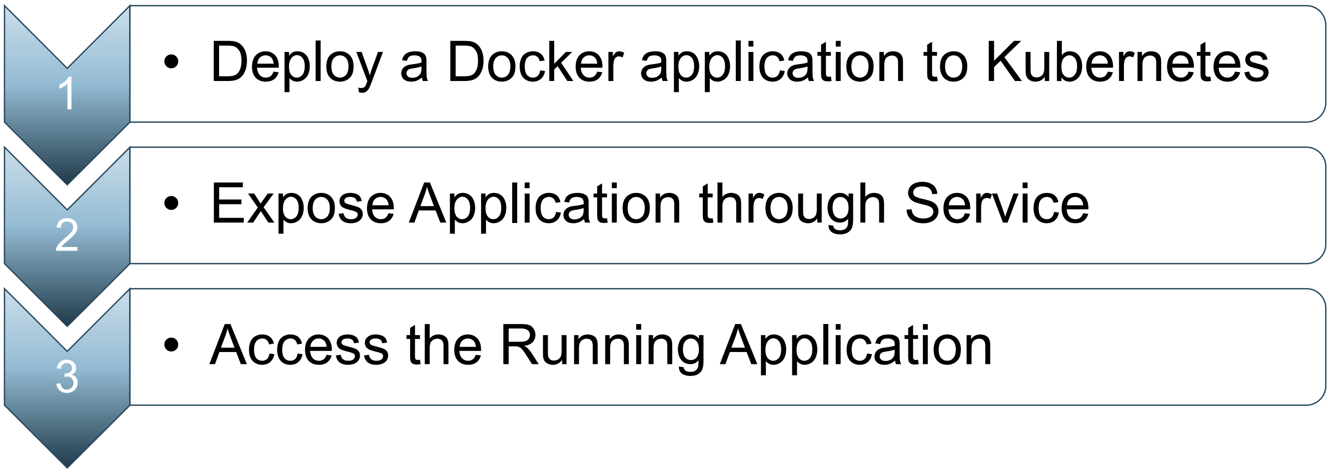
In this section, you learned how to start a Kubernetes cluster on your local machine and viewed the Kubernetes dashboard.

Section 6: Deploy an Application to Kubernetes

Purpose:	In this lab you will learn how to deploy an application to Kubernetes.

Tasks:	Tasks you will complete in this lab exercise include: • Deploy a Docker application to Kubernetes • Expose the application through a service • Access the running application

Section 6: Deploy an Application to Kubernetes



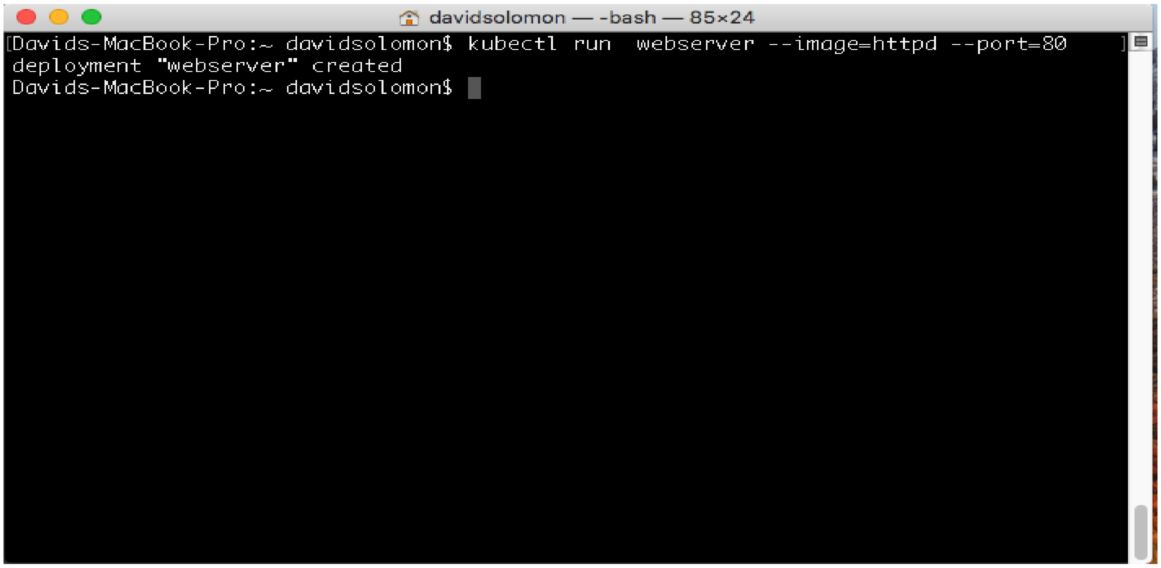
1. Deploy a Docker application to Kubernetes
2. Expose Application through Service
3. Access the Running Application

Section 6: Lab Instructions

Step	Action
1	**Deploy a Docker application to the Kubernetes cluster**

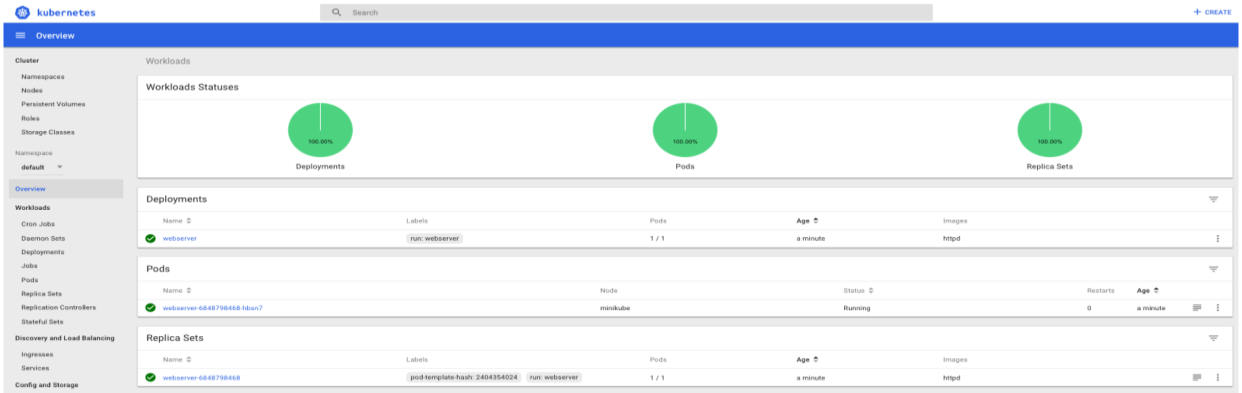
a. We will now deploy the same Docker application to your cluster. To do this, enter the following command to create a new deployment called "webserver":
~$ kubectl run webserver --image=httpd --port=80

b. Confirm the output is as shown below:



```
[Davids-MacBook-Pro:~ davidsolomon$ kubectl run  webserver --image=httpd --port=80   ]
 deployment "webserver" created
 Davids-MacBook-Pro:~ davidsolomon$ 
```

c. Return to the Kubernetes dashboard. You will see that a new deployment and Pod have been created for the application, as shown below, indicating that the application is now running in the cluster.



Deployments

Name	Labels	Pods	Age	Images
✓ webserver	run: webserver	1 / 1	a minute	httpd

Pods

Name	Node	Status	Restarts	Age
✓ webserver-6848798468-hbsn7	minikube	Running	0	a minute

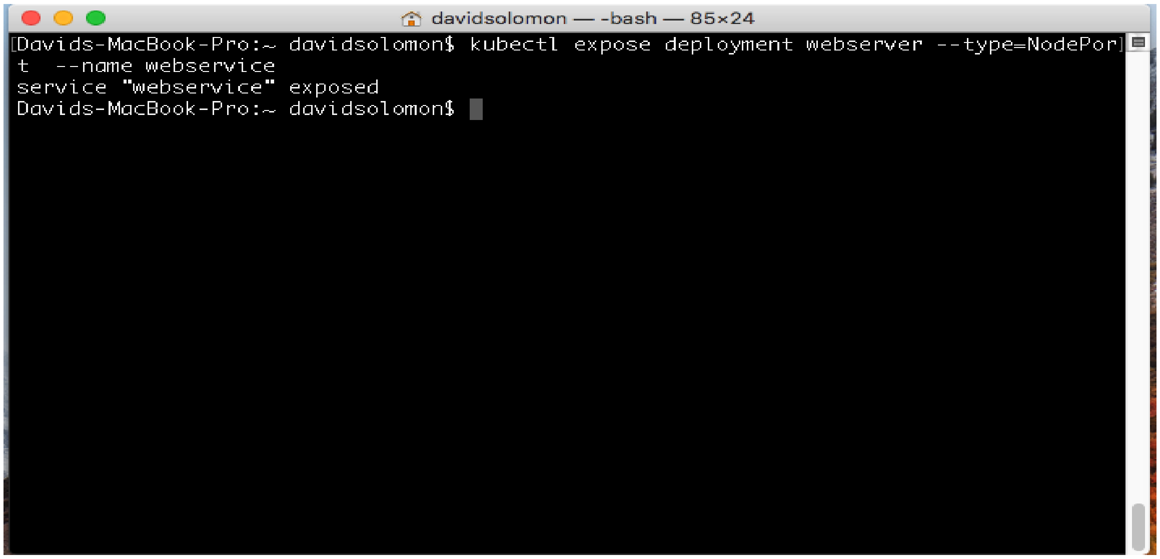
Replica Sets

Name	Labels	Pods	Age	Images
✓ webserver-6848798468	pod-template-hash: 2404354024 run: webserver	1 / 1	a minute	httpd

Step	Action
2	**Exposing the application through a service**

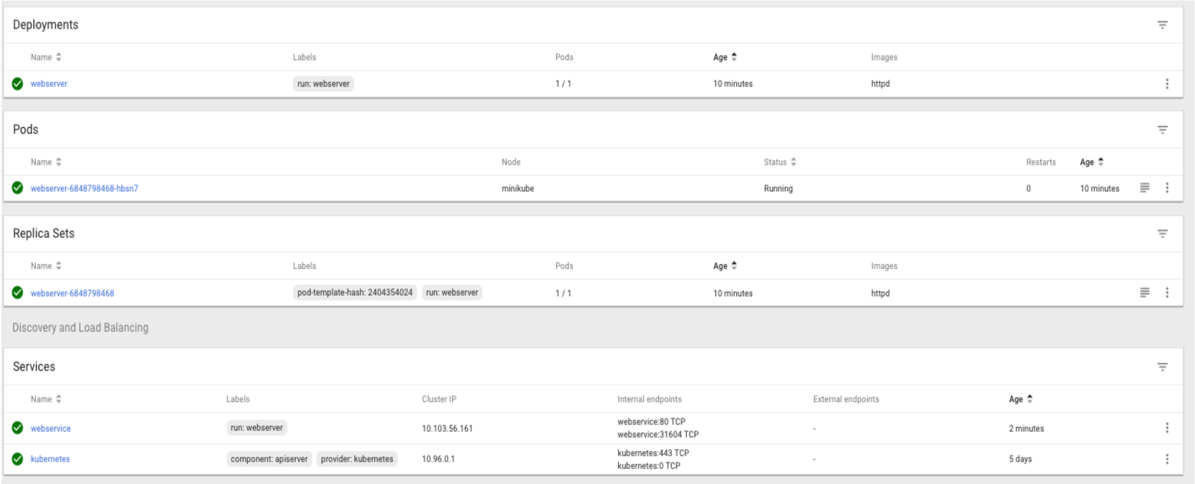
a. In order to interact with your application from outside the cluster, you will need to create a service which provide an endpoint to expose the application. To do this, enter the following command to create a new service called "webservice":
~$ kubectl expose deployment webserver --type=NodePort --name *webservice*

b. Confirm the output is as shown below:



```
[Davids-MacBook-Pro:~ davidsolomon$ kubectl expose deployment webserver --type=NodePor
t --name webservice
service "webservice" exposed
Davids-MacBook-Pro:~ davidsolomon$ 
```

c. Return to the Kubernetes dashboard. You will see that a new service has been created for the application, as shown below, indicating that the application now has a service for accessing it from outside the cluster.



Deployments

	Name	Labels	Pods	Age	Images
✓	webserver	run: webserver	1 / 1	10 minutes	httpd

Pods

	Name	Node	Status	Restarts	Age
✓	webserver-6848798468-hbsn7	minikube	Running	0	10 minutes

Replica Sets

	Name	Labels	Pods	Age	Images
✓	webserver-6848798468	pod-template-hash: 2404354024 run: webserver	1 / 1	10 minutes	httpd

Discovery and Load Balancing

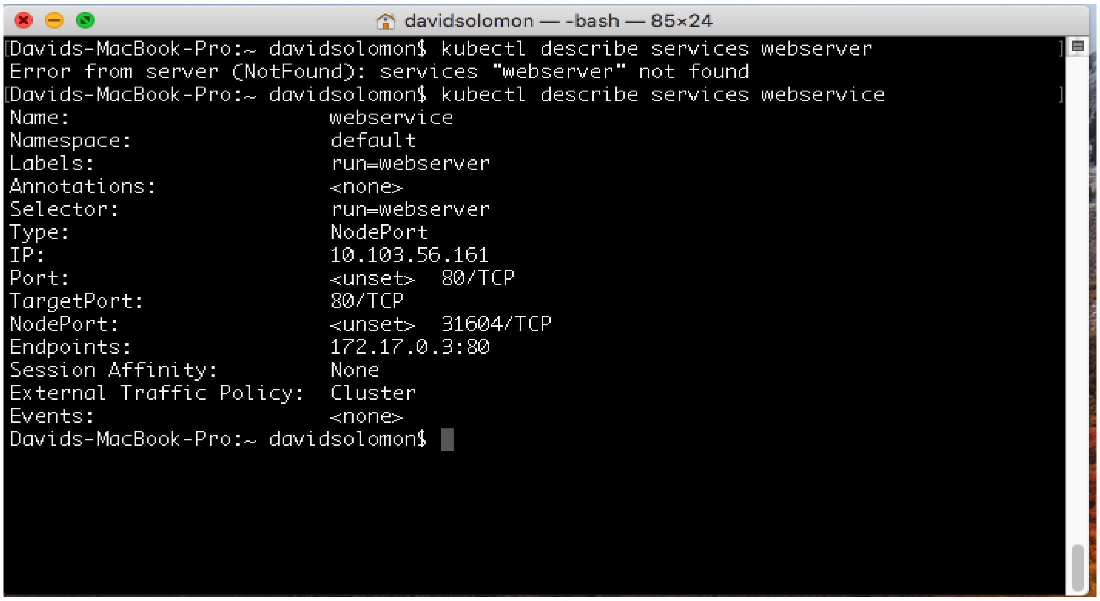
Services

	Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
✓	webservice	run: webserver	10.103.56.161	webservice:80 TCP webservice:31604 TCP	-	2 minutes
✓	kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	5 days

Step	Action
3	**Access the Running Application**

a. In order to interact with the application, Kubernetes maintains a set of ports for enabling outside access. These ports are assigned automatically when a service is created and mapped to the port the application is expecting (in this case, port 80). In order to see which port has been assigned for your deployed application, run the following command:
~$ kubectl describe services webservice

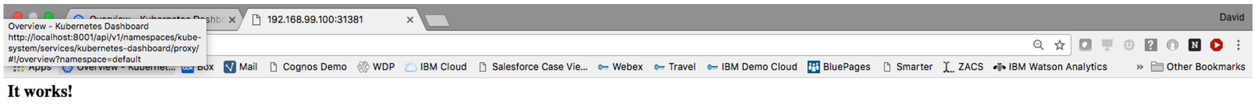
b. Once this command is entered, you will see the following output. Note the port number listed in the "NodePort" section of the output. This is the port you need to access the application (31604 in the example below).



```
[Davids-MacBook-Pro:~ davidsolomon$ kubectl describe services webserver
Error from server (NotFound): services "webserver" not found
[Davids-MacBook-Pro:~ davidsolomon$ kubectl describe services webservice
Name:                     webservice
Namespace:                default
Labels:                   run=webserver
Annotations:              <none>
Selector:                 run=webserver
Type:                     NodePort
IP:                       10.103.56.161
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  31604/TCP
Endpoints:                172.17.0.3:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
Davids-MacBook-Pro:~ davidsolomon$ 
```

c. To access the application, go to your browser and enter the following URL and verify that you can access the application, as shown below:

192.168.99.100:<*your port number*>



It works!

Section 6: Lab Summary

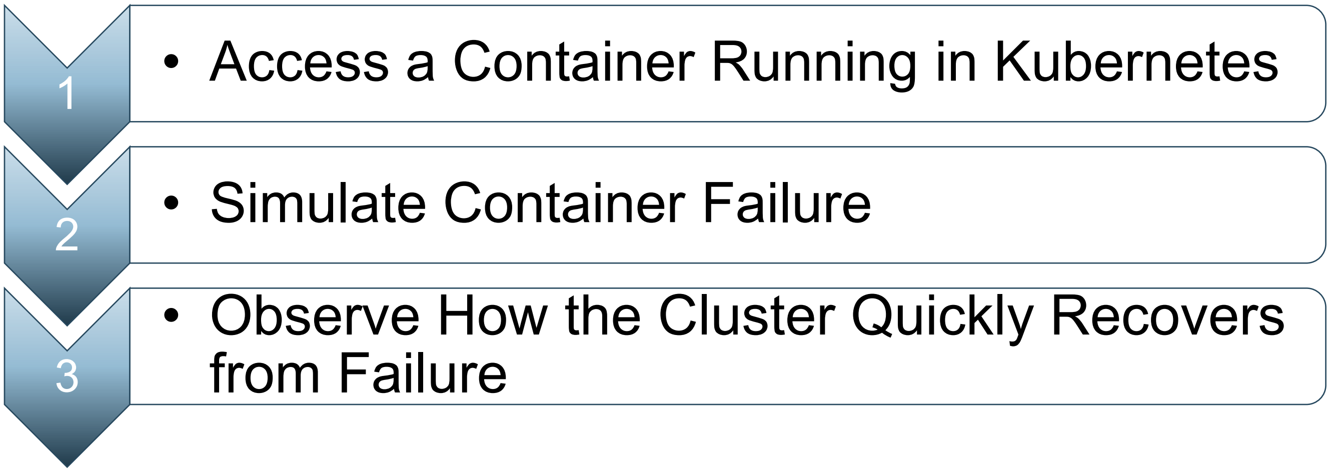
In this section, you learned how to deploy an Docker application to Kubernetes, how to enable it to be access from the outside world, and how to access it.

Section 7: Observing Kubernetes Resiliency

Purpose:	In this lab, you will learn how Kubernetes recovers from a container failure.

Tasks:	Tasks you will complete in this lab exercise include: • Access a container running in Kubernetes from Docker • Simulate a container failure • Observer how the cluster quickly recovers from the failure

Section 7: Observing Kubernetes Resiliency



1. Access a Container Running in Kubernetes
2. Simulate Container Failure
3. Observe How the Cluster Quickly Recovers from Failure

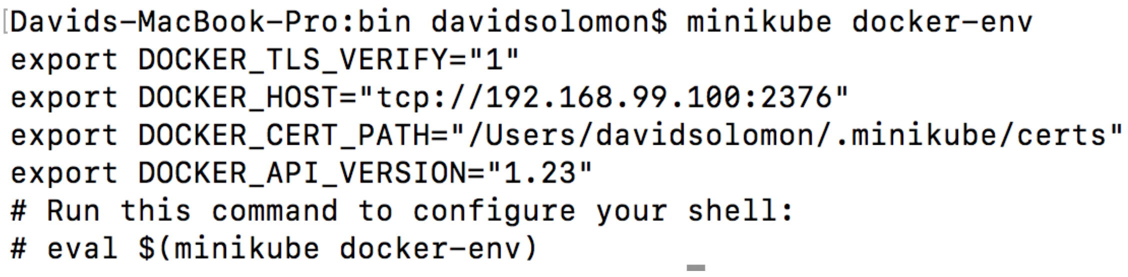
Section 7: Lab Instructions

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>

<u>Access a container running in Kubernetes from Docker</u>

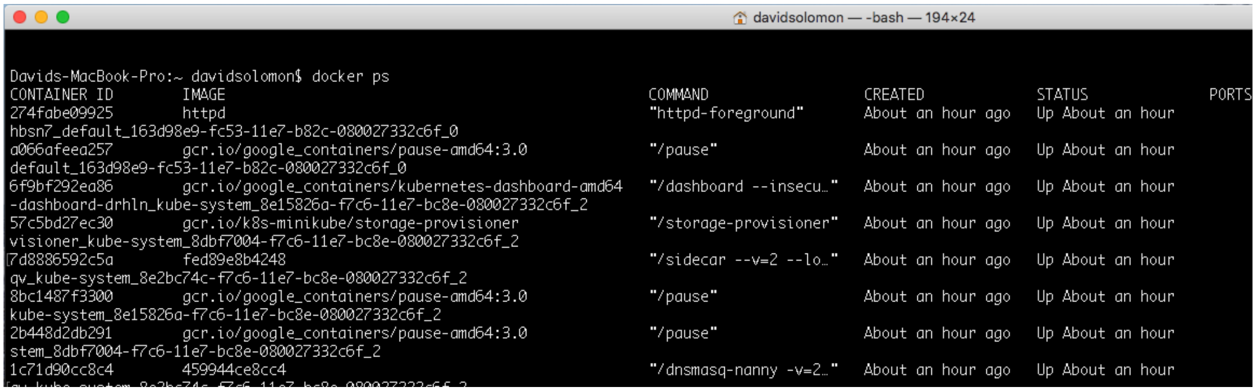
a. In order to enable the control of containers running inside Kubernetes, we will need to first set your Docker environment so that it can manage these containers. To do this, run the following command:
~$ minikube docker-env

This command will provide a list of export statements, as shown below:



```
Davids-MacBook-Pro:bin davidsolomon$ minikube docker-env
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/davidsolomon/.minikube/certs"
export DOCKER_API_VERSION="1.23"
# Run this command to configure your shell:
# eval $(minikube docker-env)
```

b. Copy the export statements and run them. Confirm that Docker is now pointing to the Kubernetes cluster by entering the following command. Your httpd application should be listed, as shown below:
~$ docker ps



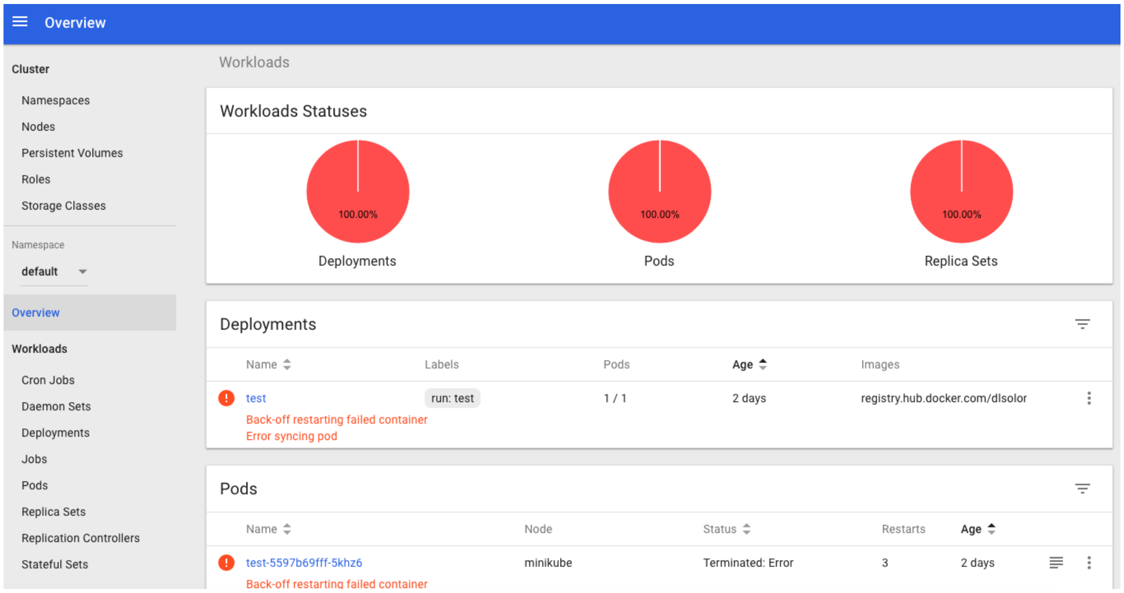
```
Davids-MacBook-Pro:~ davidsolomon$ docker ps
CONTAINER ID        IMAGE                                                 COMMAND                  CREATED             STATUS              PORTS
274fabe09925        httpd                                                 "httpd-foreground"       About an hour ago   Up About an hour
hbsn7_default_163d98e9-fc53-11e7-b82c-080027332c6f_0
a066afeea257        gcr.io/google_containers/pause-amd64:3.0              "/pause"                 About an hour ago   Up About an hour
default_163d98e9-fc53-11e7-b82c-080027332c6f_0
6f9bf292ea86        gcr.io/google_containers/kubernetes-dashboard-amd64    "/dashboard --insecu…"   About an hour ago   Up About an hour
-dashboard-drhln_kube-system_8e15826a-f7c6-11e7-bc8e-080027332c6f_2
57c5bd27ec30        gcr.io/k8s-minikube/storage-provisioner               "/storage-provisioner"   About an hour ago   Up About an hour
visioner_kube-system_8dbf7004-f7c6-11e7-bc8e-080027332c6f_2
7d8886592c5a        fed89e8b4248                                          "/sidecar --v=2 --lo…"   About an hour ago   Up About an hour
qv_kube-system_8e2bc74c-f7c6-11e7-bc8e-080027332c6f_2
8bc1487f3300        gcr.io/google_containers/pause-amd64:3.0              "/pause"                 About an hour ago   Up About an hour
kube-system_8e15826a-f7c6-11e7-bc8e-080027332c6f_2
2b448d2db291        gcr.io/google_containers/pause-amd64:3.0              "/pause"                 About an hour ago   Up About an hour
stem_8dbf7004-f7c6-11e7-bc8e-080027332c6f_2
1c71d90cc8c4        459944ce8cc4                                          "/dnsmasq-nanny -v=2…"   About an hour ago   Up About an hour
```

</td></tr>
<tr><td>2</td><td>

<u>Simulate a Container Failure</u>

a. We will now use a docker command to simulate the failure of the webserver application. To do this, find the container ID for this application from the output shown above, and enter the following command:

</td></tr>
</table>

Step	Action
	~$ docker kill *<the container ID for your application>* b. Immediately return to and refresh the Kubernetes dashboard. While the container will only take a few seconds to recover, you may see the failure reflected as shown below: 

Overview

Cluster

- Namespaces
- Nodes
- Persistent Volumes
- Roles
- Storage Classes

Namespace: default

Overview

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Workloads

Workloads Statuses

Deployments

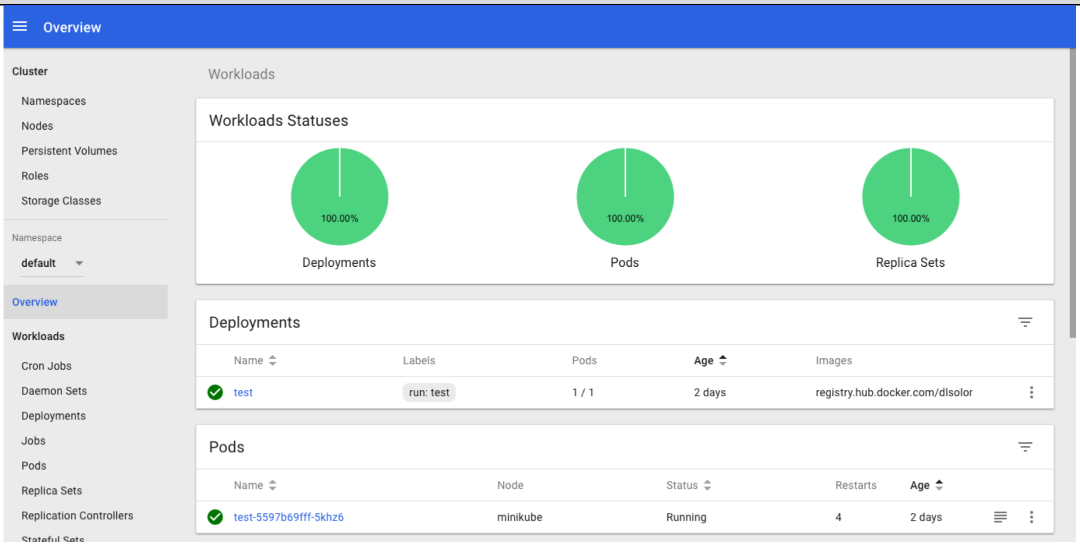
Name	Labels	Pods	Age	Images
test Back-off restarting failed container Error syncing pod	run: test	1 / 1	2 days	registry.hub.docker.com/dlsolor

Pods

Name	Node	Status	Restarts	Age
test-5597b69fff-5khz6 Back-off restarting failed container	minikube	Terminated: Error	3	2 days

Step	Action
3	**<u>Observe how quickly the cluster recovers from the failure</u>** c. Refresh the dashboard page again. Notice that the status is now green again, and the webserver deployment is now again running.

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td></td><td>



Deployments

	Name	Labels	Pods	Age	Images
✓	test	run: test	1 / 1	2 days	registry.hub.docker.com/dlsolor

Pods

	Name	Node	Status	Restarts	Age
✓	test-5597b69fff-5khz6	minikube	Running	4	2 days

d. We can also observe this behavior by entering the following command and noting that the container ID for “httpd” is now different, indicating that the application has fully recovered as a new container. However, since the exposing service has not changed, there is no change to how the application is accessed:
~$ docker ps

</td></tr>
</table>

Section 7: Lab Summary

In this section, you learned how Kubernetes can quickly recover from an application failure.