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.

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml

Other company, product and service names may be trademarks or service marks of others

Document Revision History

Rev #	File Name	Date
1.0	Docker and Kubernetes - Lab	1/21/2018
2.0	Docker and Kubernetes – Lab	1/26/2018

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.

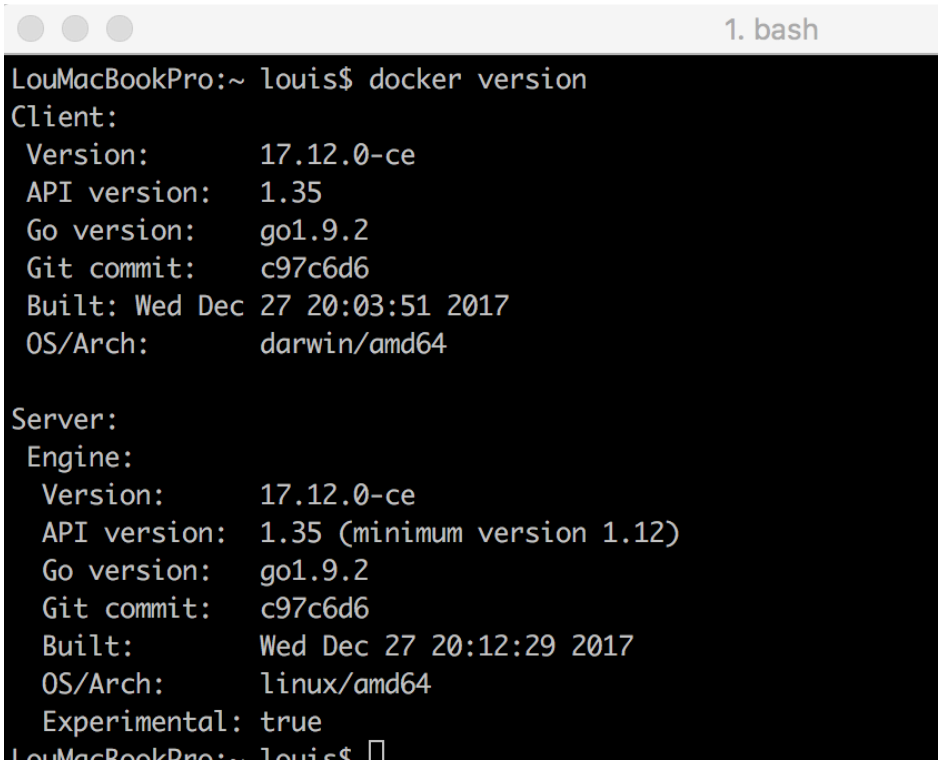
Tasks:	Docker software for: • Discuss Docker Editions • Review supported platforms • Gather information about Docker installation • Verify Docker is running without issue

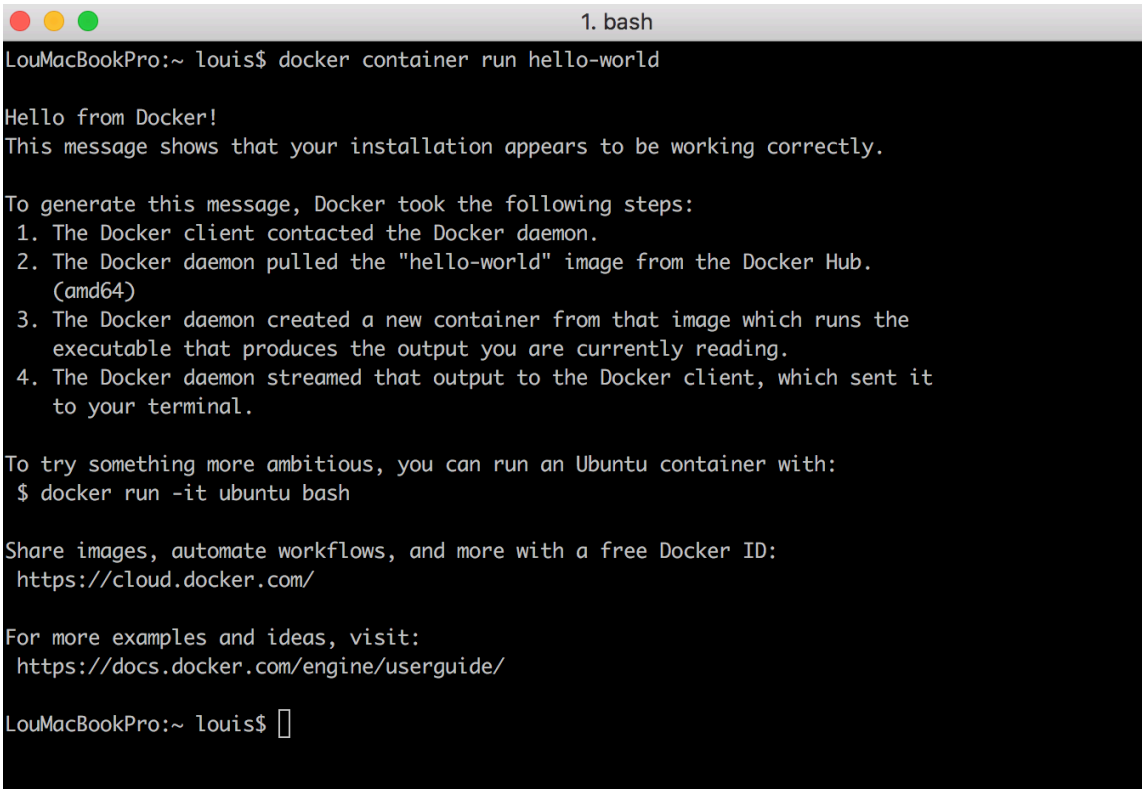
Section 1: Lab Workflow Overview

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

Step	Action
4	**Determine Docker Version** a. Open terminal (MAC), Shell (Windows), or Quickstart Terminal (Docker Toolbox) then type: ~$ docker version  b. You output should be similar.
5	**Verify Docker Installation** a. Run test Docker container ~$ docker container run hello-world

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

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

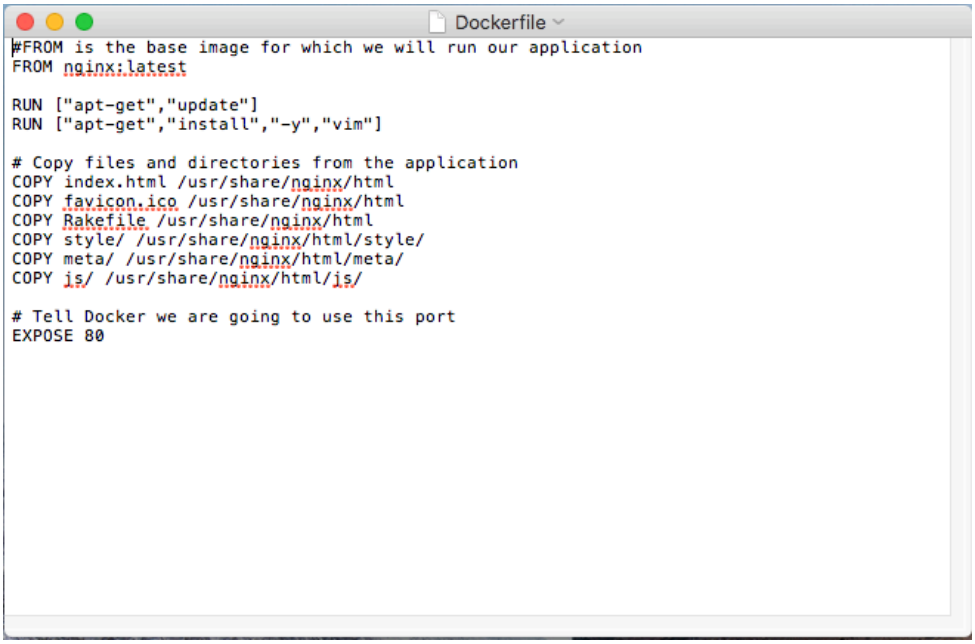
Purpose:	Throughout the remaining sections of this lab, we will be using a sample application, a variation of the mobile game 2048. You will see how we create a Docker image from this application and run it as a container. In later sections of this lab, you will learn how to deploy this container into a Kubernetes environment. This section introduces container basics. You will learn how to create, run, inspect and manage containers. Also, you will work through establishing console access within the container.

Tasks:	Tasks you will complete in this lab exercise include: • Creating a Docker Image for an Application • Running containers • Inspecting containers • Container process monitoring • Container shell access

Section 2: Lab Workflow Overview

1. Build a Docker Image of an Application
2. Run a Container
3. Stop/Delete a Container
4. Inspect a Running Container
5. Run Shell Inside a Container

Section 2: Lab Instructions

Step	Action
1	**Build a Docker Image for an Application** a. In order to build an image, you will first need to download the 2048 application from the following URL: http://bit.ly/2GuzEAP b. Unzip the files into a directory on your local machine and open a terminal and cd to that directory. c. These files are the application code required to run the game. Notice there is a file called "Dockerfile" in the top directory of the unzipped files. The Dockerfile is the file you create that instructs Docker how to create and package the application into a Docker image. In this case, the file has already been created for you. Open the file and browse its contents. It will look similar to the figure below:  The commands in this file instruct Docker to use a simple web service (nginx) as a base image (nginx is automatically pulled from Docker Hub when the image is built. The file then copies the application code into a directory structure within the image (in /usr/share). Finally, port 80 is exposed in order to enable access to the game from our Web Browser.

Step	Action
	d. Now you can build the image by running the following command: ~ $ docker build -t 2048_game . *(don't forget the "." at the end of the command)* e. Docker will now build the image. You can confirm this by running the following command and observing that an image named "2048_game" is listed: ~$ docker images Davids-MacBook-Pro-2:2048-master davidsolomon$ docker images REPOSITORY TAG IMAGE ID CREATED SIZE 2048_game latest 5aebd0e1585b About an hour ago 109MB <none> <none> 119fd5eefae3 28 hours ago 663MB mysql latest f008d8ff927d 11 days ago 409MB You have now successfully taken an existing application and created a docker image from it.
2	**<u>Run a Container</u>** a. Now that you have an image, we will now run the 2048 application as a container. To do this, run the following command: ~$ docker container run –name mygame -p 8080:80 2048_game The container "mygame" is an instance of the image "2048_game" running as a process. 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. A page will open with the game, as shown below:

Step	Action
	 You have now successfully run your first container!!

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
	`Davids-MacBook-Pro-2:2048-master davidsolomon$ docker ps -a`
	`CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES`
	`efd7d14bc574 2048_game "nginx -g 'daemon of…" 27 seconds ago Exited (0) 15 seconds ago mygame`
	`Davids-MacBook-Pro-2:2048-master davidsolomon$`
	-a, --all: Show all containers (default shows just running)
	
	d. Remove the container:
	~$ docker container rm mygame
	
	Containers can be removed either by their name or container id

3	**Inspect a Running Container** a. Run a new Docker container for the game:

Step	Action
	~$ docker run --publish 8080:80 --detach --name mygame 2048_game You should be brought back to the terminal prompt (the “detach” option runs the container as a background process) b. Open a browser and navigate to “localhost:8080”. You should be prompted with the game again. c. You can run a variety of commands to get information on the status of a running container. These commands. can be useful when troubleshoot an environment or application. For example, inspecting the meta-data for running container: ~$ docker container inspect mygame and, Stream live performance container metrics: ~$ docker container stats mygame d. Clean up ~$ docker container rm -f mygame Commands: **-d, --detach** - Run the container in the background.
4	**Run Shell Inside a Container** a. We can also directly access a container via a command shell. It allows you to directly login to the container’s command prompt; enabling you to troubleshoot application issues or update the content of a running container. First run the container again: ~$ docker container run –name mygame -d -p 8080:80 2048_game b. Next, we will use the following command to open a shell prompt into the container: ~$ docker exec -it mygame bash

Step	Action
	c. Run Linux commands in container: For example, # ls -tal // List directories and files. # exit // Exit shell d. Delete the container: ~$ docker rm -f mygame 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

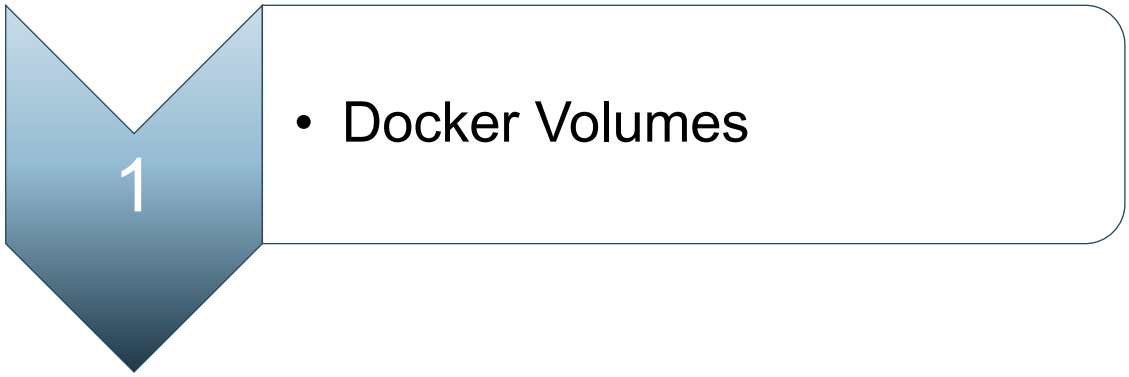
In this section you learned how to create new containers based on images stored in Docker Hub. You also learned how to interact with containers both from the outside (top, inspect, stats, …), and from the inside (docker exec and run). Access to the Docker service via tty was demonstrated and you learned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

Section 3: Data Persistence in Docker

Purpose:	In this section, you will see one method of how data from a container can be persisted, even after a container is removed. Unless such persistence is established, any changes made to a container's data are deleted once the container is deleted. The method we will use below is Docker Volumes. With Volumes, Docker controls a location for persistent storage on your local machine that persists once a container is deleted.

Tasks:	Tasks you will complete in this lab exercise include: • Create and work with Docker volumes

Section 3: Lab Workflow Overview



Section 3: Lab Instructions

Step	Action
1	**Docker Volumes** a. Let's run our game application in a new container, except this time we will include an option (-v (or volume)) to instruct Docker to persist the content of a specific directory on your local machine: ~$ docker container run -d --name mygame -p 8080:80 -v myvol:/usr/share/nginx/html 2048_game b. Open bash shell on container and navigate the /usr/share/nginx/html directory: ~$ docker container exec -it mygame bash # cd /usr/share/nginx/html c. Create a new file in the html folder containing the phrase, "This is my file". # echo "This is my file" > myfile Confirm the file "myfile" is listed in the directory and exit the container. # ls root@1f5d5f84c4a4:/usr/share/nginx/html# ls 50x.html Rakefile favicon.ico index.html js meta myfile style root@1f5d5f84c4a4:/usr/share/nginx/html# # exit d. We will now remove the container using the command: ~$ docker rm -f mygame e. Now, we can create a new container, referencing the persistent volume and confirm that our file is still present: ~$ docker container run -d --name mygame -p 8080:80 -v myvol:/usr/share/nginx/html 2048_game

Step	Action
	~$ docker container exec -it mygame bash # cd /usr/share/nginx/html # ls root@1f5d5f84c4a4:/usr/share/nginx/html# ls 50x.html Rakefile favicon.ico index.html js meta myfile style root@1f5d5f84c4a4:/usr/share/nginx/html# # cat myfile root@a9703c89b049:/usr/share/nginx/html# cat myfile This is my file root@a9703c89b049:/usr/share/nginx/html# Volumes are extremely useful for local development projects. You can maintain several volumes to which you can attach a new directory or database that fits a specific purpose.

Section 3: Lab Summary

In this lab you were introduced to one way to persist data on the host file system. With volumes the container references a volume object on the local file system.

Section 4: Getting Started with Minikube

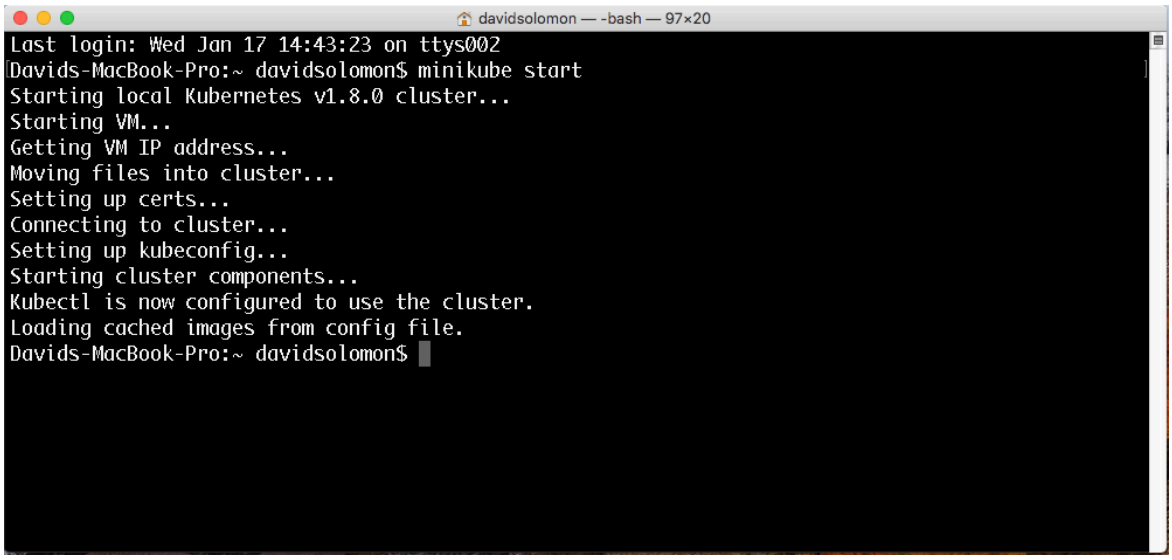
Purpose:	In this lab you will learn basic skills as it relates to networking (security & DNS) with Docker containers:

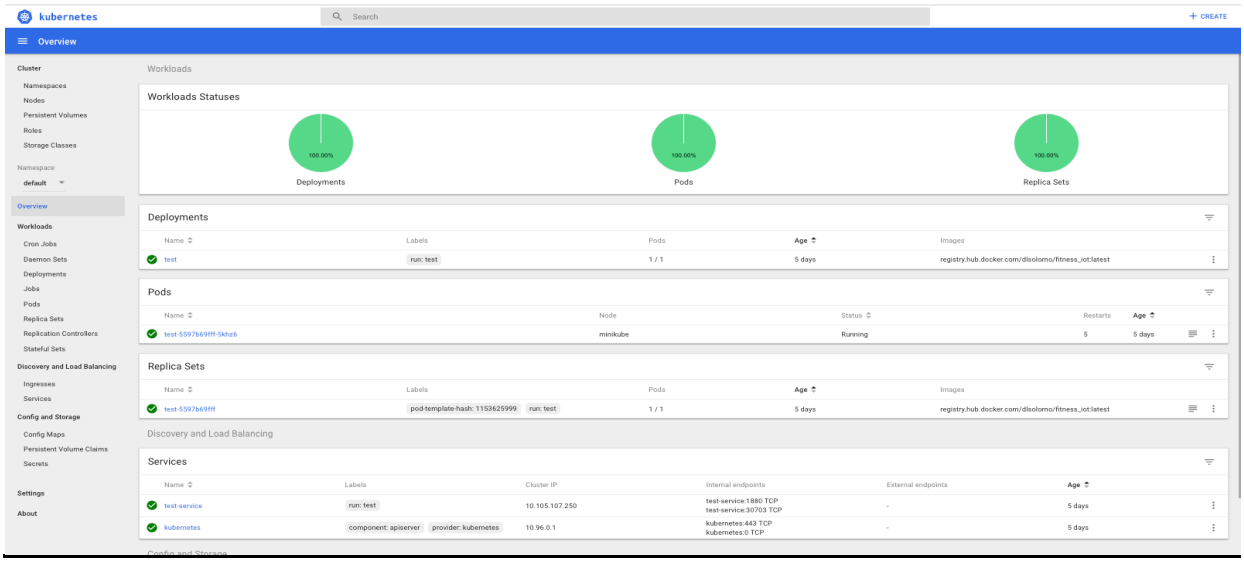
Tasks:	Tasks you will complete in this lab exercise include: • Start Minikube • Configure Kubernetes • Launch & Explore Kubernetes Dashboard

Section 4: Getting Started with Minikube

1. Open Terminal / Launch Minikube
2. Configure Minikube Environment
3. Start Kubernetes Cluster
4. Explore Kubernetes Dashboard

Section 4: 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  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
	
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: 

Section 4: Lab Summary

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

Section 5: Deploy your Application to Kubernetes

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

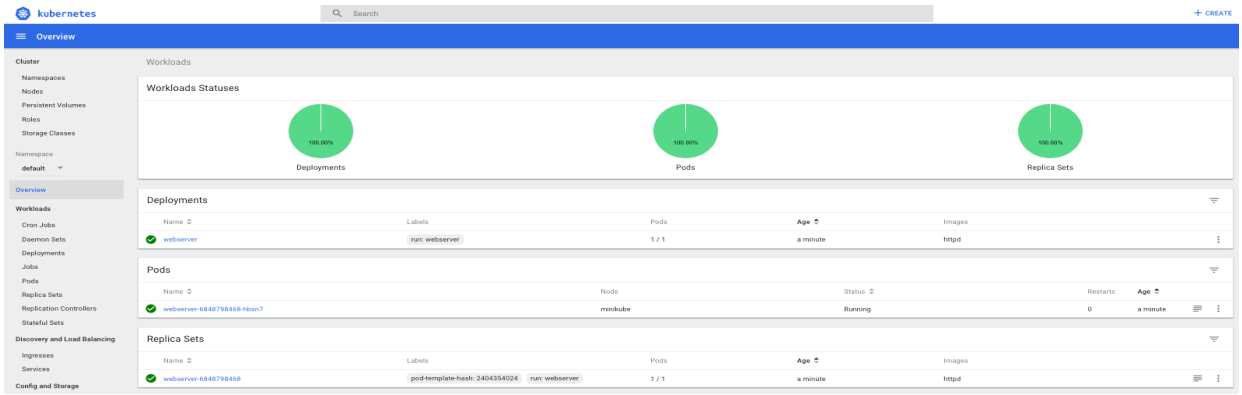
Tasks:	Tasks you will complete in this lab exercise include: • Deploy a Docker application to Kubernetes • Expose the application through a service • Access the running application

Section 5: Deploy an Application to Kubernetes

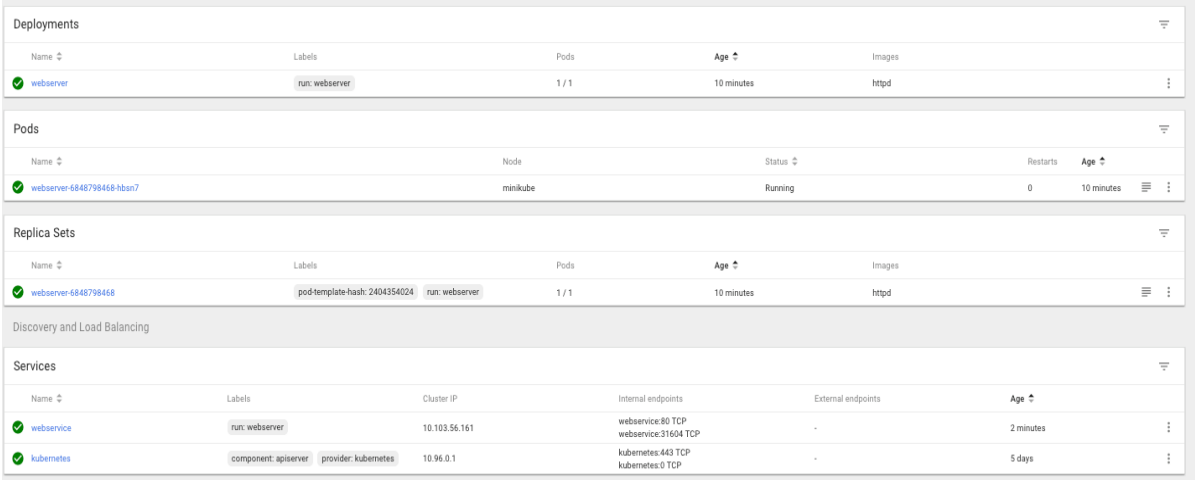
1. Deploy a Docker application to Kubernetes
2. Expose Application through Service
3. Access the Running Application

Section 5: Lab Instructions

Step	Action

1	**Deploy a Docker application to the Kubernetes cluster**
	
	a. We will now deploy the same 2048 game application to your cluster. To do this, enter the following command to create a new deployment called “mygame”, using a version of the image already deployed on Docker Hub.
	
	~$ kubectl run mygame --image=dlsolomo/2048_game --port=80
	
	b. Confirm the output is as shown below:
	
	`Davids-MacBook-Pro-2:2048-master davidsolomon$ kubectl run mygame --image=dlsolomo/2048_game --port=80`
	`deployment "mygame" created`
	`Davids-MacBook-Pro-2:2048-master 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.
	
	 

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 “mygame-service”: ~$ kubectl expose deployment mygame --type=NodePort --name *mygame-service* b. Confirm the output is as shown below:

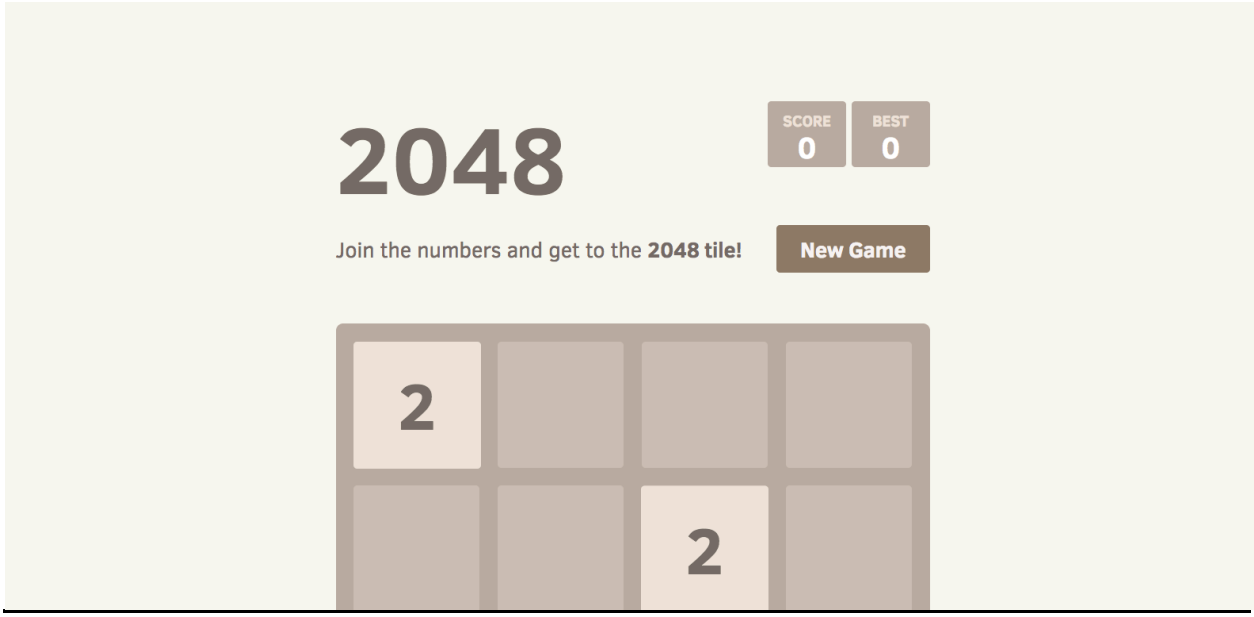
Step	Action
	Davids-MacBook-Pro-2:2048-master davidsolomon$ kubectl expose deployment mygame --type=NodePort --name mygame-service service "mygame-service" exposed Davids-MacBook-Pro-2:2048-master davidsolomon$
	c. Return to the Kubernetes dashboard. You will see that a new service has been created for the application, as shown below, indicating that the application now has a service for accessing it from outside the cluster.
	
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 mygame-service 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 (30752 in the example below).

Step	Action

```
Davids-MacBook-Pro-2:2048-master davidsolomon$ kubectl describe services mygame-service
Name:                     mygame-service
Namespace:                default
Labels:                   run=mygame
Annotations:              <none>
Selector:                 run=mygame
Type:                     NodePort
IP:                       10.106.58.241
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30752/TCP
Endpoints:                172.17.0.2:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
Davids-MacBook-Pro-2:2048-master 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*>



d. Delete the deployment and the service, using the following commands:

~$ kubectl delete deployment mygame

~$ kubectl delete service mygame-service

Step	Action

Section 5: 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 6: Observing Kubernetes Resiliency

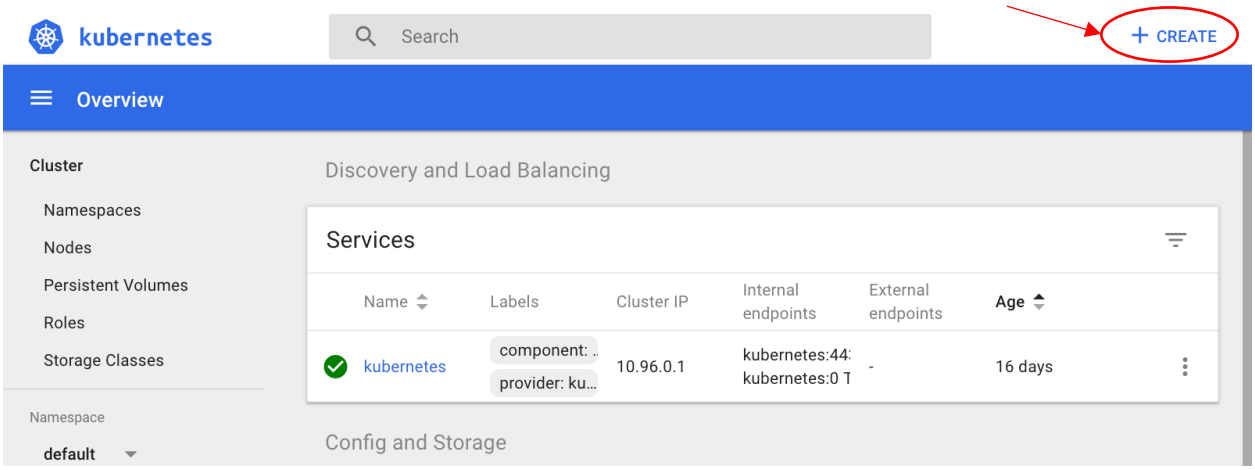
Purpose:	In this lab, you will learn how Kubernetes recovers from a container failure.

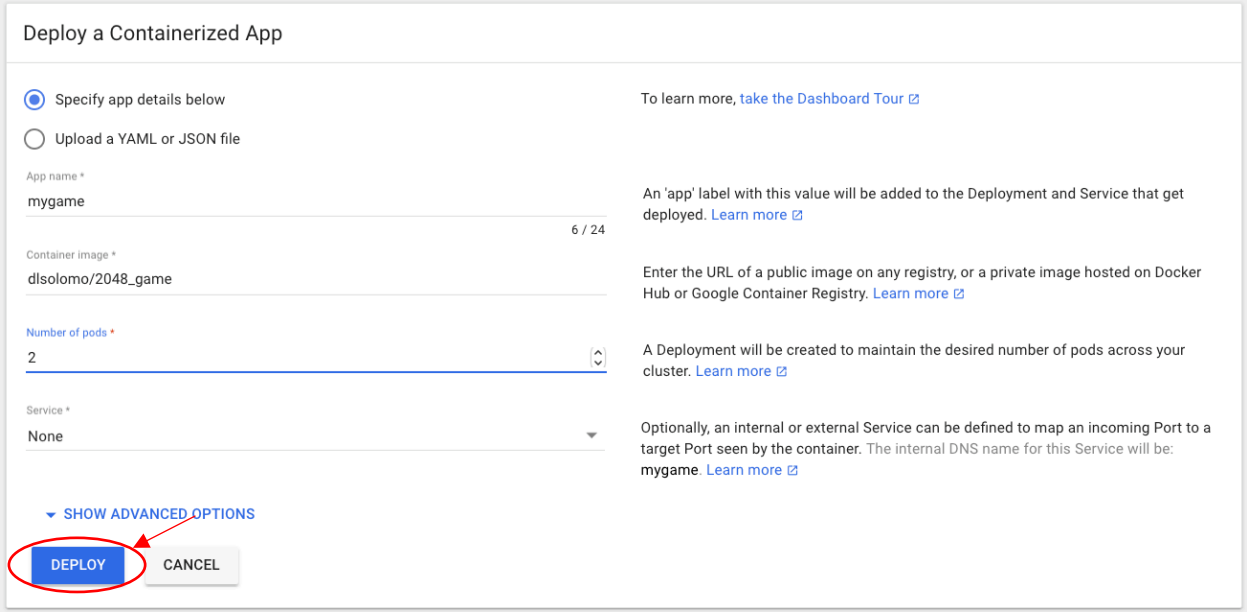
Tasks:	Tasks you will complete in this lab exercise include: • Create a new deployment with multiple Pods • Explore the ReplicaSet policy • Simulate a pod failure • Observer how the cluster quickly recovers from the failure to retain the number of available pods

Section 6: Observing Kubernetes Resiliency

1. Create a new deployment with multipe Pods
2. Explore the ReplicaSet Policy
3. Simulate a Pod Failure
4. Observe How the Cluster Quickly Recovers from Failure

Section 6: Lab Instructions

Step	Action
1	**Create a new deployment** a. We will now create a new deployment using the Kubernetes web GUI interface. This time, however, we will specify the use of 2 Pods Access http://192.168.99.100:30000 on your browser b. Select the “Create” button on the upper right of the page:  c. A create deployment form will appear. Complete the form as shown below and click “Deploy”. This will create a deployment with 2 Pods:

Step	Action
	

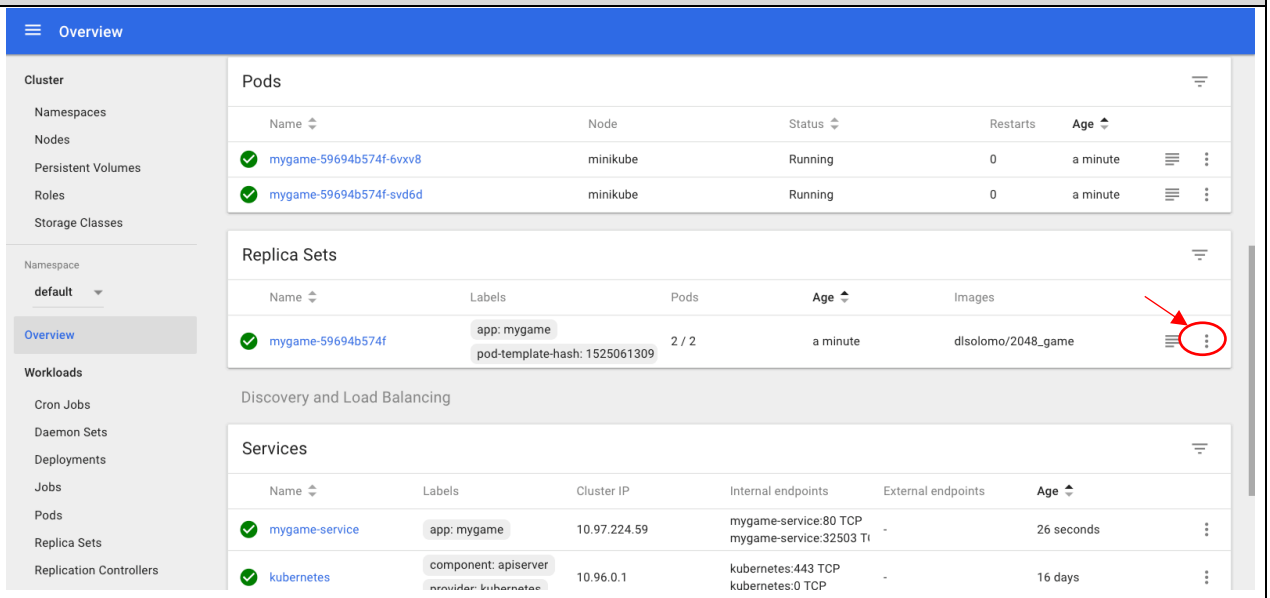
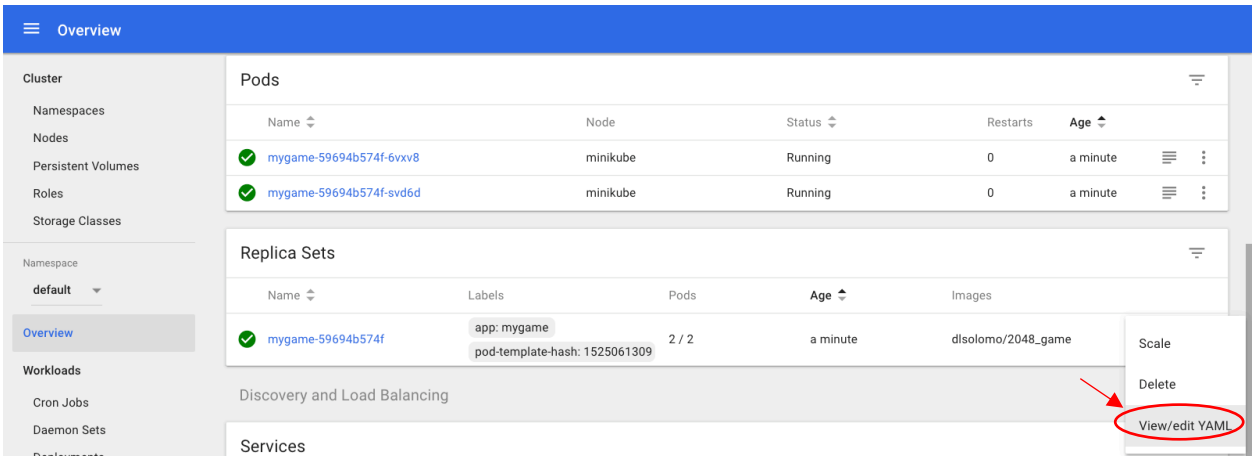
d. Create a new service as before using the following command:

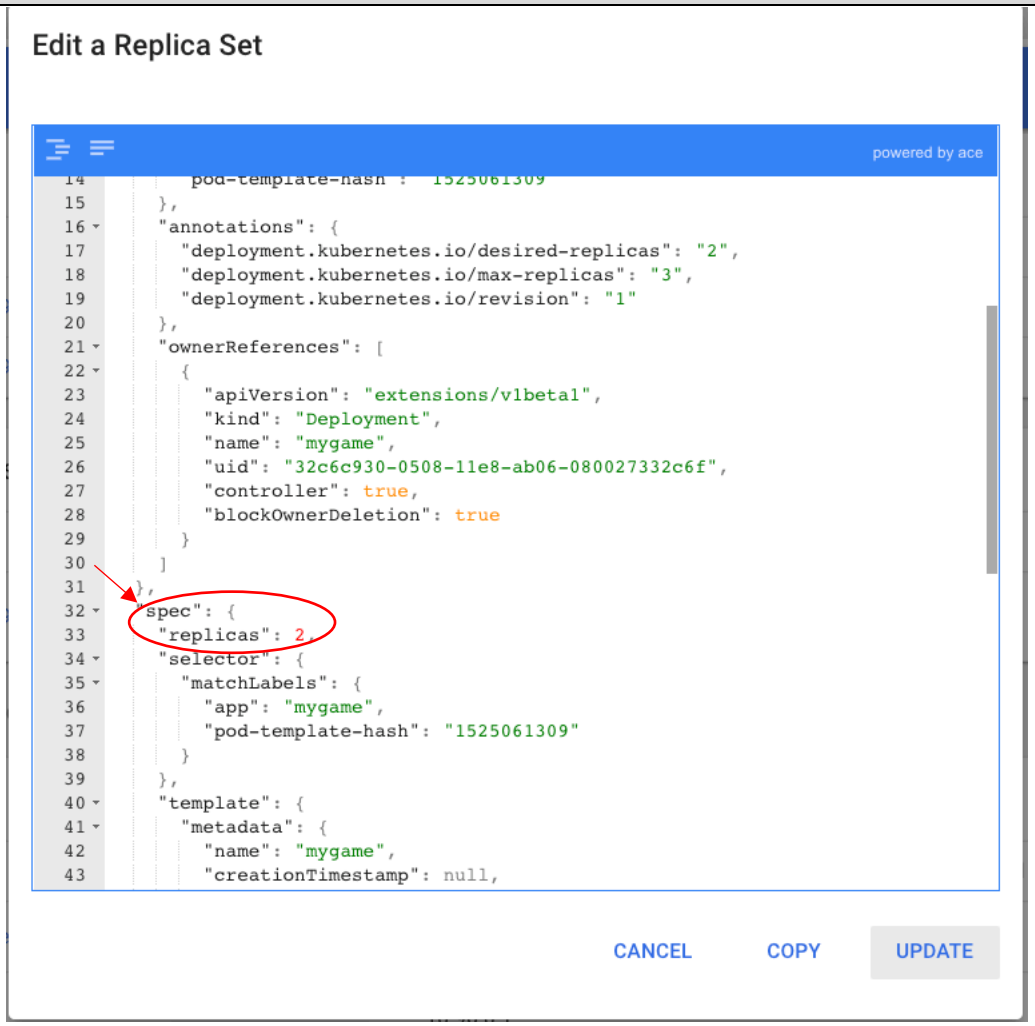
~$ kubectl expose deployment mygame --type=NodePort --port 80 --name *mygame-service*

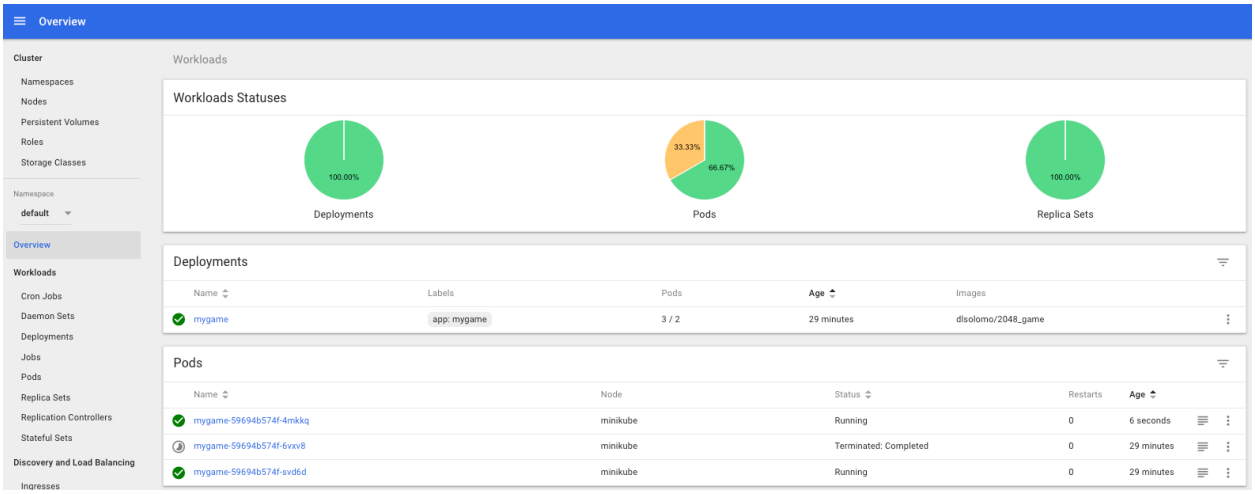
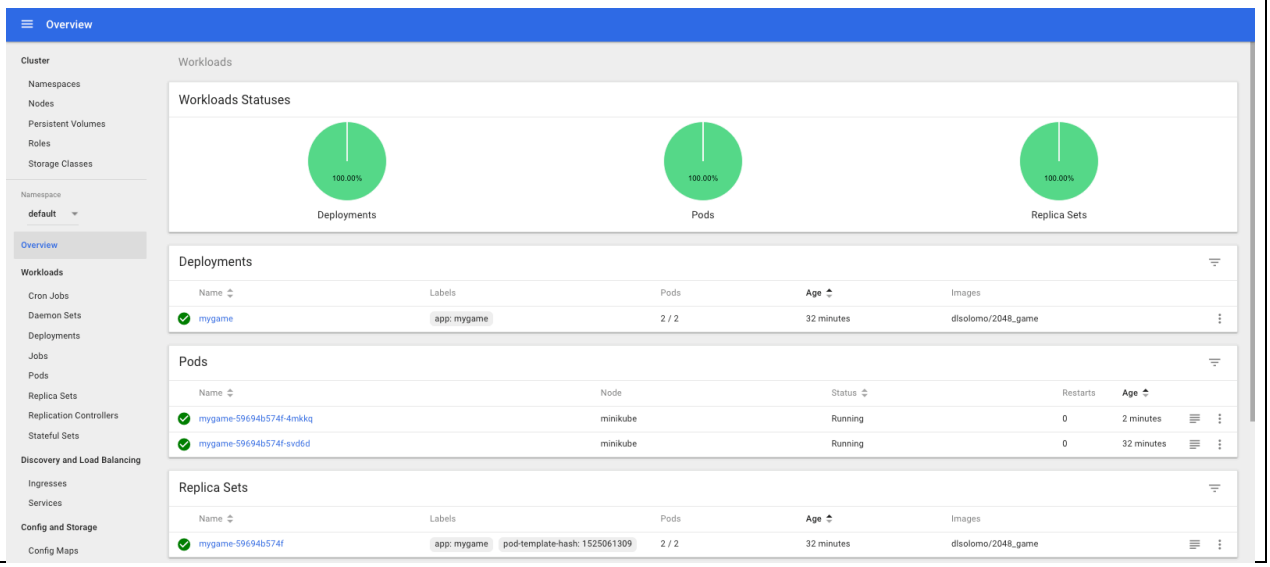
e. Identify the port for the new service:

~$ kubectl describe services mygame-service

Step	Action
	Davids-MacBook-Pro-2:2048-master davidsolomon$ kubectl describe services mygame-service Name: mygame-service Namespace: default Labels: run=mygame Annotations: <none> Selector: run=mygame Type: NodePort IP: 10.106.58.241 Port: <unset> 80/TCP TargetPort: 80/TCP NodePort: <unset> 30752/TCP Endpoints: 172.17.0.2:80 Session Affinity: None External Traffic Policy: Cluster Events: <none> Davids-MacBook-Pro-2:2048-master davidsolomon$ f. Confirm that the application is now deployed: http:// 192.168.99.100:<*your port number*>
2	**Explore the ReplicaSet Policy** a. We will now examine the ReplicaSet in more detail. As you may recall, a ReplicaSet manages a policy that governs the how and when Pods are deployed, including the recovery of a failed Pod. This recovery is based a policy established during or after a deployment. The policy is specified in a YAML formatted file that is location within the Kubernetes Cluster. With the proper permissions, this file can be viewed or edited within the Kubernetes dashboard. Access the Kubernetes dashboard: http://192.168.99.100:30000 b. Navigate to the ReplicaSet for the game application (it was created automatically when you performed the deployment). Click on the ellipsis (the 3 dot) symbol on the right hand side and select “View/Edit YAML”.

Step	Action
	  c. Navigate to the “Spec” section of the YAML and notice that the number of replicas specified is 2. This means that the policy of this ReplicaSet is set to maintain 2 active Pods running the game application. This is because we specified 2 Pods when we created the deployment in the previous step.

Step	Action
	 d. Click “cancel” to close the YAML file.
2	**Simulate a Pod Failure** a. We will now use a kubectl command to simulate the failure of a Pod. To do this, find the Pod IDs for the running Pods using the following command: ~$ kubectl get pods The command will list the 2 running pods and their names. b. Enter the following command to delete one of the Pods (it does not matter which one). Copy the name from the output of the previous step.

Step	Action
	~$ kubectl delete pods *<the name of one of your Pods>*. c. Immediately return to and refresh the Kubernetes dashboard. While the container will only take a few seconds to recover, you may see the failure while the old Pod is being shutdown reflected as shown below. However, notice that ReplicaSet automatically recovered by create a new Pod with a new name. 
3	**<u>Observe how quickly the cluster recovers from the failure</u>** a. Wait approximately 1 minute and refresh the dashboard page again. Notice that the status is now green again, and are up and running. 

Step	Action
	b. You should also still be able to access the game via the URL: http://192.168.99.100:<*your port number*>

Section 6: Lab Summary

In this section, you learned how Kubernetes can quickly recover from a Pod failure.