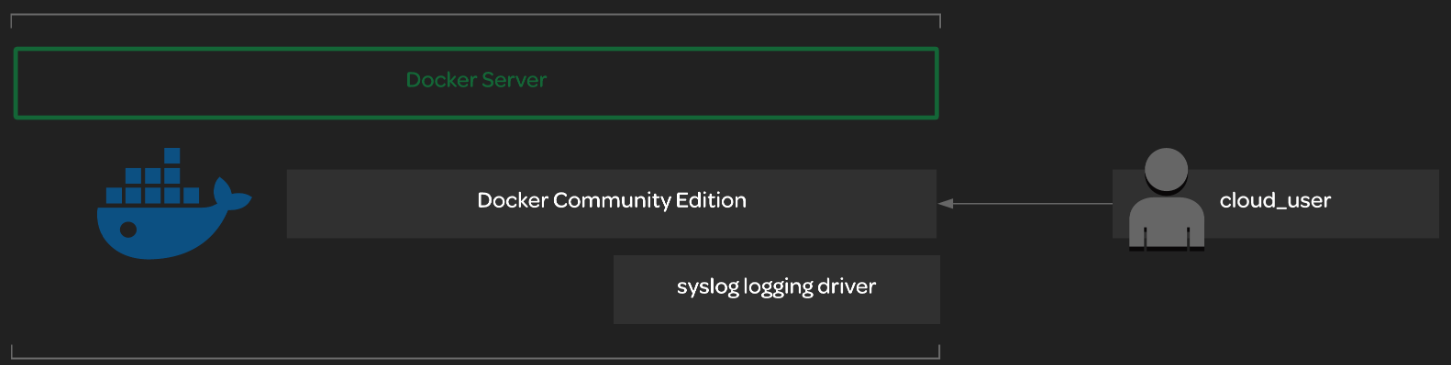
Installing and Configuring the Docker Engine

Docker CE is a great way to get started using the Docker engine. It is free and open-source, but provides a high-quality container runtime. This lab will help you practice the steps involved in installing and configuring the Docker Engine. You will practice installing Docker CE and configuring a logging driver. This lab will help provide you with some basic insight into how the Docker Engine is installed and configured on systems in the real world.

### **Instructions**

Your company is ready to start using Docker on some of their servers. In order to get started, they want you to set up and configure Docker CE on a server that has already been set up. You will need to make sure that the server meets the following specifications:

* Docker CE is installed and running on the server.
* Use Docker CE version 5:18.09.5~3-0~ubuntu-bionic.
* The user cloud\_user has permission to run docker commands.
* The default logging driver is set to syslog.

If you get stuck, feel free to check out the solution video, or the detailed instructions under each objective. Good luck!

## **Solution**

Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Install Docker CE on the server**

1. First, set up the Docker Repository.

sudo apt-get update

sudo apt-get -y install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

1. Install docker packages.

sudo apt-get update

sudo apt-get install -y docker-ce=5:18.09.5~3-0~ubuntu-bionic docker-ce-cli=5:18.09.5~3-0~ubuntu-bionic containerd.io

1. Verify that your installation is working.

sudo docker version

### **Give cloud\_user access to run docker commands**

1. Add cloud\_user to the docker group.

sudo usermod -a -G docker cloud\_user

Log out of the server, then log back in.

1. Once you are logged back on, you can verify cloud-user's access:

docker version

### **Set the default logging driver to syslog**

1. Edit daemon.json:

sudo vi /etc/docker/daemon.json

1. Add configuration to daemon.json to set the default logging driver.
2. {
3. "log-driver": "syslog"

}

1. Restart docker.

sudo systemctl restart docker

1. Verify that the logging driver was set properly like so:

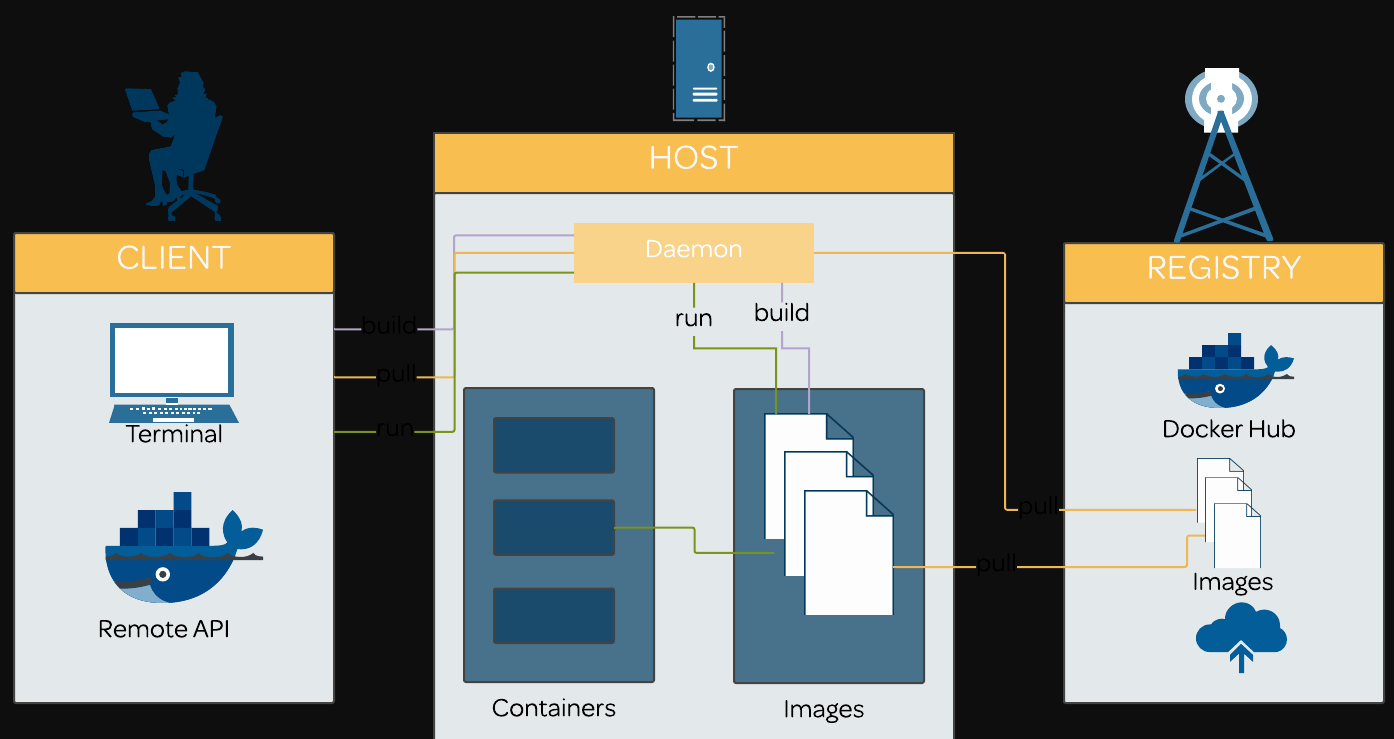
docker info | grep Logging

1. This command should return a line that says:

Logging Driver: syslog

Running Your First Docker Container

|  |
| --- |
| * Install Docker from the Default CentOS 7 Repository * Enable & start Docker service * User Permissions * hello-world * Pull Images |



### **Install docker from the default CentOS 7 repository**

After logging into the server, install the latest version of Docker using yum.

$ sudo yum -y install docker

### **enable & start docker service**

Once installation completes, enable & start the service using systemd.

$ systemctl enable docker

$ systemctl start docker

### **user permissions**

Create a new group named docker, then add the cloud\_user user to the group.

$ sudo groupadd docker

$ sudo usermod -aG docker cloud\_user

### **hello-world**

After starting \*& enabling Docker, run the hello-world container image to verify installation.

$ docker run docker.io/hello-world

### **pull images**

After getting a valid return message from the hello-world image, pull the following images into your Docker repository to prepare for the next exercise.

* 06kellyjac/nyancat
* jeremy646/doge

docker pull 06kellyjac/nyancat

docker pull jeremy646/doge

## Exercise: Exposing Ports to Your Host System

|  |
| --- |
| You are being asked to run an Nginx container listening on port 80, but that port is not available on the host. Complete the following tasks:  1. Using the appropriate Docker CE commands, download the latest 'nginx' webserver image from Docker Hub.  2. Instantiate a container based on the 'nginx' image from the previous step. This container should have the following characteristics:    \* when started, the container should run in 'detached' mode    \* name the container 'nginxtest'    \* use the appropriate option to allow Docker to map all container service ports to random host ports over 32768    \* the container is based on the 'nginx' image from step one  3. Verify the container is running and find the host port that is mapped to the web server's port 80  4. Using any tool you wish, verify that the default site page is served (NOTE: use the cloud server's PRIVATE ADDRESS for your testing) |

1. Using the appropriate Docker CE commands, download the latest 'nginx' webserver image from Docker Hub.

[user@tcox6 ~]$ docker pull nginx

Using default tag: latest

latest: Pulling from library/nginx

e7bb522d92ff: Pull complete

0f4d7753723e: Pull complete

91470a14d63f: Pull complete

Digest: sha256:edc8182581fdaa985a39b3021836aa09a69f9b966d1a0ff2f338be6f2fbfe238

Status: Downloaded newer image for nginx:latest

2. Instantiate a container based on the 'nginx' image from the previous step. This container should have the following characteristics:

  \* when started, the container should run in 'detached' mode

  \* name the container 'nginxtest'

  \* use the appropriate option to allow Docker to map all container service ports to random host ports over 32768

  \* the container is based on the 'nginx' image from step one

[user@tcox6 ~]$ docker run -d --name nginxtests -P nginx

590b38619f5db236b2ee77d2594ec23da0d7a041c80a86c7c29ad56abb0f8885

3. Verify the container is running and find the host port that is mapped to the web server's port 80

[user@tcox6 ~]$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

590b38619f5d nginx "nginx -g 'daemon ..." 31 seconds ago Up 30 seconds 0.0.0.0:32768->80/tcp nginxtests

4. Using any tool you wish, verify that the default site page is served (NOTE: use the cloud server's PRIVATE ADDRESS for your testing)

[user@tcox6 ~]$ elinks http://172.31.19.60:32768

Thank you for using nginx.

# Deploying a Static Website to the Container (Docker)

**Pull Spacebones Image**

Student should pull the docker image located [here](https://hub.docker.com/r/spacebones/doge). This can be done via the command:

docker pull spacebones/doge

Since it is stored at Docker Hub, this will find the image required.

**Start the Website Container and Redirect HTTP Port 80 to the Host**

The student will be starting a container from the spacebones/doge image installed locally in the previous step, named 'treatseekers', in disconnected mode, redirecting container port 80 to host port 80. The command to accomplish this is:

docker run -d --name treatseekers -p 80:80 spacebones/doge

You can verify it is running after with

docker ps

Which will tell you the container, name and port that is currently running

che**Container Called 'Treatseekers' is Running**keyboard\_arrow\_up

As part of your launch, you were asked to name your container 'treatseekers'. You can verify that by reviewing the content of the following command:

docker ps

Which will show you all running containers, 'treatseekers' should now appear with the port information running on.

# Building Container Images

In this activity, you are tasked with migrating the SpaceBones official Government website from Ubuntu to CentOS 6 by preparing a CentOS 6 image found on DockerHub to run Apache. Use the website contents found in the [content-dockerquest-spacebones](https://github.com/linuxacademy/content-dockerquest-spacebones/) GitHub repository (specifically contents found under the /doge/ directory) to recreate the website on CentOS. Good luck!

After logging into your server, pull the centos:6 Docker image

docker pull centos:6

Once the pull completes, start the Docker container in interactive mode, name it 'websetup'.

docker run -it --name websetup centos:6 /bin/bash

Update the system, then install Apache & Git.

yum -y update

yum -y install httpd git

Once installation completes, clone the content-dockerquest-spacebones repository, then copy the contents of the /doge/ subdirectory to /var/www/html.

git clone https://github.com/linuxacademy/content-dockerquest-spacebones

cp content-dockerquest-spacebones/site /var/www/html

In order for the site to display correctly, rename the default welcome.conf file to welcome.conf.bak.

mv /etc/httpd/conf.d/welcome.conf /etc/httpd/conf.d/welcome.conf.bak

Enable & start the Apache service, then exit the container environment by typing exit or hitting CTRL+C.

chkconfig httpd on && service httpd start

exit

Save the edited image, then pat yourself on the back for being awesome.

docker commit websetup spacebones:thewebsite

# 

# Creating Your Own Docker Image

Docker Hub provides many useful, pre-made images which you can use for a variety of applications. However, if you want to use Docker in the real world, you will likely be required to design and build your own Docker images, either to customize existing images or to run your own software.

In this lab, you will have the opportunity to work with Docker images by designing your own image to a set of specifications using a Dockerfile. You will then be able to run a container using your image to verify that it works.

## **Solution**

1. Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Create a Dockerfile to define the image and build it**

1. Change to the project directory and create a Dockerfile.
2. cd ~/fruit-list

vi Dockerfile

1. Build a Dockerfile that meets the provided specifications.
2. FROM nginx:1.15.8
3. ADD static/fruit.json /usr/share/nginx/html/fruit.json
4. ADD nginx.conf /etc/nginx/nginx.conf
5. EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

1. Build the image.

docker build -t fruit-list:1.0.0 .

### **Run a container with the image in detached mode and verify that it works**

1. Run a container in detached mode using the newly-created image.

docker run --name fruit-list -d -p 8080:80 fruit-list:1.0.0

1. Make a request to the container and verify that you get some JSON with a list of fruits.

curl localhost:8080

# Dockerizing a node.js application

Using the example Dockerfile included in activity instructions, use Docker to build a new Node.js app image using the files under the content-dockerquest-spacebones/nodejs-app subdirectory, named baconator. Be sure to tag the image as dev. Good luck!

* Clone the content-dockerquest-spacebones GitHub repo

git clone https://github.com/linuxacademy/content-dockerquest-spacebones

* Move into the nodejs-app subdirectory

cd ~/content-dockerquest-spacebones/nodejs-app

* Use the Dockerfile below to build a new image

FROM node:7

WORKDIR /app

COPY package.json /app

RUN npm install

COPY . /app

CMD node index.js

EXPOSE 8081

* build container image

docker build -t baconator:dev .

* (optional) Run the image to verify functionality

docker run -d -p 80:8081 baconator:dev

## Exercise: Display Details About Your Containers and Control the Display of Output

|  |
| --- |
| You are being asked to run an Nginx container listening on port 80, but that port is not available on the host. Complete the following tasks:  1. Using the appropriate Docker CE commands, download the latest 'nginx' webserver image from Docker Hub.  2. Instantiate a container based on the 'nginx' image from the previous step. This container should have the following characteristics:    \* when started, the container should run in 'detached' mode    \* name the container 'nginxtest'    \* use the appropriate option to allow Docker to map all container service ports to random host ports over 32768    \* the container is based on the 'nginx' image from step one  3. Using the appropriate Docker command, find the CONTAINER IP address ONLY (use the aforementioned appropriate command and JUST display the IP address using its built in options) |

1. Using the appropriate Docker CE commands, download the latest 'nginx' webserver image from Docker Hub.

[user@tcox6 ~]$ docker pull nginx

Using default tag: latest

latest: Pulling from library/nginx

e7bb522d92ff: Pull complete

0f4d7753723e: Pull complete

91470a14d63f: Pull complete

Digest: sha256:edc8182581fdaa985a39b3021836aa09a69f9b966d1a0ff2f338be6f2fbfe238

Status: Downloaded newer image for nginx:latest

2. Instantiate a container based on the 'nginx' image from the previous step. This container should have the following characteristics:

  \* when started, the container should run in 'detached' mode

  \* name the container 'nginxtest'

  \* use the appropriate option to allow Docker to map all container service ports to random host ports over 32768

  \* the container is based on the 'nginx' image from step one

[user@tcox6 ~]$ docker run -d --name nginxtests -P nginx

590b38619f5db236b2ee77d2594ec23da0d7a041c80a86c7c29ad56abb0f8885

3. Verify the container is running and find the host port that is mapped to the web server's port 80

[user@tcox6 ~]$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

590b38619f5d nginx "nginx -g 'daemon ..." 31 seconds ago Up 30 seconds 0.0.0.0:32768->80/tcp nginxtests

3. Using the appropriate Docker command, find the CONTAINER IP address ONLY (use the aforementioned appropriate command and JUST display the IP address using its built in options

[user@tcox6 ~]$ docker container inspect --format="{{.NetworkSettings.Networks.bridge.IPAddress}}" nginxtests

172.17.0.2

# Container Logging

In this lab, we are going to configure syslog on a Docker instance, configure Docker to use syslog instead of the JSON logging driver, and spin up two containers to test our configuration.

Let's get started. Open your terminal application, and log in to the live environment using the credentials provided on the lab instructions page.

ssh cloud\_user@<PUBLIC\_IP>

Next, elevate privileges to root.

sudo su -

## **Configure Docker to Use Syslog**

Open the rsyslog.conf file.

vim /etc/rsyslog.conf

In the file editor, uncomment the two lines under `Provides UDP syslog reception` by removing `#`.

$ModLoad imudp

$UDPServerRun 514

Then, start the syslog service.

systemctl start rsyslog

Now that syslog is running, let's configure Docker to use syslog as the default logging driver. We'll do this by creating a file called daemon.json.

sudo mkdir /etc/docker

vi /etc/docker/daemon.json

In the vi editor, enter the following, making sure to replace <PRIVATE\_IP> with the private IP of your cloud server:

{

"log-driver": "syslog",

"log-opts": {

"syslog-address": "udp://<PRIVATE\_IP>:514"

}

}

Next, save and quit. Then, start the Docker service.

systemctl start docker

The next step is to see if there are any logs coming in from Docker.

tail /var/log/messages

The output of this command tells us that there are. Now let's test our setup.

We're going to create two new containers using the httpd image. The first one will be called syslog-logging and will use syslog for the log driver. The second will be called json-logging and will use the JSON file for the log driver.

Let's create our first container.

docker container run -d --name syslog-logging httpd

Then run docker ps to make sure our container is up and running correctly.

Next, execute the docker logs command to see what logs we have.

docker logs syslog-logging

When we run this, we receive an error that says Error response from daemon: configured logging does not support reading. This is because we're using syslog instead of JSON for logging.

To confirm this, we can check the content of /var/log/messages. Verify that the syslog-logging container is sending its logs to syslog by running tail on the message log file.

tail /var/log/messages

The output shows us the logs that are being input to syslog.

Now let's create our second test container. This time, we'll specify the log driver as the JSON file.

docker container run -d --name json-logging --log-driver json-file httpd

# Updating Containers with Watchtower ---study Guide not available

# Adding Metadata and Labels

# 

## **Introduction**

For the last six months, the Acme Anvil Corporation has been migrating some of their bare metal infrastructure to Docker containers. After the initial implementation, you mention to the team that labels can be used for storing metadata about images and containers. This is useful for keeping track of image and container attributes like the build date and application version.

Your team thinks this is a great idea, and you’ve been tasked with creating a quick demo to show how useful labels can be. You created a small weather app a few months ago while learning Node.JS, and it’s perfect for a quick demo. On your Docker workstation, you will create a Dockerfile that contains four labels: the maintainer, build date, application name, and application version. You will then build the image and push it to Docker Hub. On your Docker server, you will create a new container using the weather-app image. Once the container is running, you will update the branch of your application, rebuild the image, and push the changes to Docker Hub. On the Docker server, Watchtower will update the running container with the new version of the image.

### **Instructions**

In this lab, you will deploy a container that uses labels. In order to complete this learning activity, you will need a [Docker Hub](https://hub.docker.com/)account.

Log in to your Docker workstation and Docker server, and sudo to root. You will create the Dockerfile and image on the Docker workstation. The weather-app container will be run on your Docker server.

## **Solution**

Begin by logging in to the **Docker Server** and **Docker Workstation** lab servers using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

Become the root user:

sudo su -

### **Create a Dockerfile**

On **Docker Workstation**:

1. Create a Dockerfile:

vi Dockerfile

1. Use the following instructions:

**Note**: Replace EMAIL\_ADDRESS with your email address.

FROM node

LABEL maintainer="EMAIL\_ADDRESS"

ARG BUILD\_VERSION

ARG BUILD\_DATE

ARG APPLICATION\_NAME

LABEL org.label-schema.build-date=$BUILD\_DATE

LABEL org.label-schema.applicaiton=$APPLICATION\_NAME

LABEL org.label-schema.version=$BUILD\_VERSION

RUN mkdir -p /var/node

ADD weather-app/ /var/node/

WORKDIR /var/node

RUN npm install

EXPOSE 3000

CMD ./bin/www

### **Build the Docker image**

1. Log in to Docker Hub:

docker login

1. Build the Docker image using the following parameters:

**Note**: Be sure to replace DOCKER\_USERNAME with your Docker Hub username.

docker build -t <DOCKER\_USERNAME>;/weather-app --build-arg BUILD\_DATE=$(date -u +'%Y-%m-%dT%H:%M:%SZ') \

--build-arg APPLICATION\_NAME=weather-app --build-arg BUILD\_VERSION=v1.0 -f Dockerfile .

1. Show image ID:

docker images

1. Use image ID to inspect:

docker inspect IMAGE\_ID

### **Push the image to Docker Hub**

1. Push the weather-app image to Docker Hub.

**Note**: Be sure to replace USERNAME with your Docker Hub username.

docker push USERNAME/weather-app

### **Create the weather-app container**

On the **Docker Server**:

1. Start the weather-app container.

**Note**: Be sure to replace USERNAME with your Docker Hub username.

docker run -d --name demo-app -p 80:3000 --restart always USERNAME/weather-app

1. Verify that the image is running:

docker ps

### **Check out version v1.1 of the weather app**

On the **Docker Workstation**:

1. In the weather-app directory, check out version v1.1 of the weather app.

cd weather-app

git checkout v1.1

cd ../

### **Rebuild the weather-app image**

1. Rebuild and push the weather-app image.

**Note**: Be sure to replace USERNAME with your Docker Hub username.

docker build -t USERNAME/weather-app --build-arg BUILD\_DATE=$(date -u +'%Y-%m-%dT%H:%M:%SZ') /

--build-arg APPLICATION\_NAME=weather-app --build-arg BUILD\_VERSION=v1.1 -f Dockerfile .

docker push USERNAME/weather-app

On the **Docker Server**:

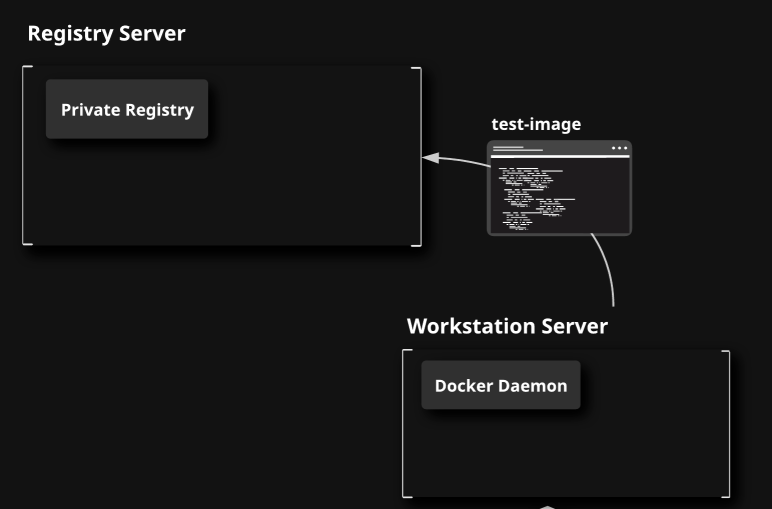
1. Show image status:

docker ps

1. Using the container ID from the previous command, inspect the weather-app image:

docker inspect IMAGE\_ID

# Building a Private Docker Registry



Docker registries provide a powerful way to manage and distribute your Docker images. Docker offers a free registry at Docker Hub, but in many scenarios, you may want greater control of your images, not to mention that it is not free to have more than one private repository on Docker Hub. Fortunately, you can build and manage your own private registries, allowing you full control over where your images are housed and how they can be accessed.

In this lab, you will have the opportunity to work with a private registry. You will build your own private registry, and you will have a chance to practice some advanced setup to ensure that your private registry is secure. After completing this lab, you will know how to set up a simple but secure private Docker registry.

## **Solution**

1. Begin by logging in to the lab servers using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

It is a good idea to log in to both servers at the same tab in separate tabs in your terminal application.

### **Set up the private registry**

1. Create an htpasswd file containing the login credentials for the initial account.

mkdir -p ~/registry/auth

docker run --entrypoint htpasswd \

registry:2 -Bbn docker d0ck3rrU73z > ~/registry/auth/htpasswd

1. Create a self-signed certificate for the registry. For common name, enter the hostname of the registry server, which is ip-10-0-1-101. For the other prompts, just hit **enter** to accept the default.

mkdir -p ~/registry/certs

openssl req \

-newkey rsa:4096 -nodes -sha256 -keyout ~/registry/certs/domain.key \

-x509 -days 365 -out ~/registry/certs/domain.crt

1. Create a container to run the registry.
2. docker run -d -p 443:443 --restart=always --name registry \
3. -v /home/cloud\_user/registry/certs:/certs \
4. -v /home/cloud\_user/registry/auth:/auth \
5. -e REGISTRY\_HTTP\_ADDR=0.0.0.0:443 \
6. -e REGISTRY\_HTTP\_TLS\_CERTIFICATE=/certs/domain.crt \
7. -e REGISTRY\_HTTP\_TLS\_KEY=/certs/domain.key \
8. -e REGISTRY\_AUTH=htpasswd \
9. -e "REGISTRY\_AUTH\_HTPASSWD\_REALM=Registry Realm" \
10. -e REGISTRY\_AUTH\_HTPASSWD\_PATH=/auth/htpasswd \

registry:2

1. Once the registry starts up, verify that it is responsive. It's OK if this command returns nothing, just make sure it does not fail.

curl -k https://localhost:443

### **Test the registry from the Docker workstation server**

1. Get the public hostname from the registry server. It should be ip-10-0-1-101.

echo $HOSTNAME

1. On the **Workstation** server, add the registry's public self-signed certificate to /etc/docker/certs.d. The scpcommand is copying the file from the registry server to the workstation. The password is the normal cloud\_userpassword provided by the lab.

**Note**: The following steps should be completed from the **Workstation** server.

sudo mkdir -p /etc/docker/certs.d/ip-10-0-1-101:443

sudo scp cloud\_user@ip-10-0-1-101:/home/cloud\_user/registry/certs/domain.crt /etc/docker/certs.d/ip-10-0-1-101:443

1. Log in to the private registry from the workstation. The credentials should be username docker and password d0ck3rrU73z.

docker login ip-10-0-1-101:443

1. Test the registry by pushing an image to it. You can pull any image from Docker hub and tag it appropriately to push it to the registry as a test image.

docker pull ubuntu

docker tag ubuntu ip-10-0-1-101:443/test-image:1

docker push ip-10-0-1-101:443/test-image:1

1. Verify image pulling by deleting the image locally and re-pulling it from the private repository.

docker image rm ip-10-0-1-101:443/test-image:1

docker image rm ubuntu:latest

docker pull ip-10-0-1-101:443/test-image:1

## Exercise: Creating and Working With Volumes

|  |
| --- |
| 1. Using the appropriate Docker command, create a storage volume for use by your containers, call the volume 'test-volume'  2. Display all the Docker storage volumes that exist on your local system  3. Execute the Docker command that will allow you to display all the attributes of that newly created 'test-volume'  4. Display the location on the host file system where that 'test-volume' exists and note the permissions  5. Remove the newly created 'test-volume' and then run the command to verify that the volume has been deleted |

1. Using the appropriate Docker command, create a storage volume for use by your containers, call the volume 'test-volume'

[user@tcox2 ~]$ docker volume create test-volume

test-volume

2. Display all the Docker storage volumes that exist on your local system

[user@tcox2 ~]$ docker volume ls

DRIVER VOLUME NAME

local test-volume

3. Execute the Docker command that will allow you to display all the attributes of that newly created 'test-volume'

[user@tcox2 ~]$ docker volume inspect test-volume

[

{

"CreatedAt": "2017-10-18T19:31:42Z",

"Driver": "local",

"Labels": {},

"Mountpoint": "/var/lib/docker/volumes/test-volume/\_data",

"Name": "test-volume",

"Options": {},

"Scope": "local"

}

]

4. Display the location on the host file system where that 'test-volume' exists and note the permissions

[user@tcox2 ~]$ ls -al /var/lib/docker/volumes/test-volume/\_data

ls: cannot access /var/lib/docker/volumes/test-volume/\_data: Permission denied

[user@tcox2 ~]$ sudo ls -al /var/lib/docker/volumes/test-volume/\_data

[sudo] password for user:

total 0

drwxr-xr-x. 2 root root 6 Oct 18 19:31 .

drwxr-xr-x. 3 root root 18 Oct 18 19:31 ..

[user@tcox2 ~]$ sudo ls -al /var/lib/docker/volumes/test-volume/

total 0

drwxr-xr-x. 3 root root 18 Oct 18 19:31 .

drwx------. 3 root root 42 Oct 18 19:31 ..

drwxr-xr-x. 2 root root 6 Oct 18 19:31 \_data

5. Remove the newly created 'test-volume' and then run the command to verify that the volume has been deleted

[user@tcox2 ~]$ docker volume rm test-volume

test-volume

[user@tcox2 ~]$ docker volume ls

DRIVER VOLUME NAME

## Exercise: Using External Volumes Within Your Containers

|  |
| --- |
| 1. Create a Docker volume called 'http-files' and then list all volumes to confirm it was created  2. Execute the appropriate Docker command to display ALL information on the 'http-files' volume, make a note of the filesystem location that volume is linked to on your host  3. Pull the 'httpd' image from the standard Docker repository and verify it was installed locally  4. Create an 'index.html' file of your choosing and copy it to the HOST directory that your 'http-files' volume is linked to (obtained in Step #2 above)  5. Start a container based on the 'httpd' image with the following characteristics:    - the container should run in the background (i.e. you are not connected to it in the current terminal)    - name the container 'test-web'    - associate the created volume 'http-files' with the container directory path of /usr/local/apache2/htdocs  6. Using the appropriate Docker command, find out the container's IP address and note it  7. Execute the 'curl' command against that IP address to display the Apache website running on the container, verify the output is from your created 'index.html' file  8. Make a change to the 'index.html' file on the container's host and save the file. Rerun the 'curl' command to verify the container's website is now displaying the new value |

1. Create a Docker volume called 'http-files' and then list all volumes to confirm it was created

[user@tcox3 ~]$ docker volume create http-files

http-files

[user@tcox3 ~]$ docker volume ls

DRIVER VOLUME NAME

local http-files

2. Execute the appropriate Docker command to display ALL information on the 'http-files' volume, make a note of the filesystem location that volume is linked to on your host

[user@tcox3 ~]$ docker volume inspect http-files

[

{

"CreatedAt": "2017-10-19T21:20:43Z",

"Driver": "local",

"Labels": {},

"Mountpoint": "/var/lib/docker/volumes/http-files/\_data",

"Name": "http-files",

"Options": {},

"Scope": "local"

}

]

3. Pull the 'httpd' image from the standard Docker repository and verify it was installed locally

[user@tcox3 ~]$ docker pull httpd

Using default tag: latest

latest: Pulling from library/httpd

85b1f47fba49: Pull complete

3dee1a596b5f: Pull complete

86144720cb98: Pull complete

23273e61b31a: Pull complete

011f98a84808: Pull complete

771652f83cbc: Pull complete

0036b8043a1c: Pull complete

Digest: sha256:b2cf2ae2400c2b49c43bbead8931c489566cf5f150164990447d36c831976a5c

Status: Downloaded newer image for httpd:latest  
[user@tcox3 ~]$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

httpd latest c24f66af34b4 7 days ago 177MB

4. Create an 'index.html' file of your choosing and copy it to the HOST directory that your 'http-files' volume is linked to (obtained in Step #2 above)

NOTE: This command is done as ROOT user  
[root@tcox3 ~]# echo "This is my test website index file" > /var/lib/docker/volumes/http-files/\_data/index.html

[root@tcox3 ~]# cat /var/lib/docker/volumes/http-files/\_data/index.html

This is my test website index file

5. Start a container based on the 'httpd' image with the following characteristics:

  - the container should run in the background (i.e. you are not connected to it in the current terminal)

  - name the container 'test-web'

  - associate the created volume 'http-files' with the container directory path of /usr/local/apache2/htdocs

[user@tcox3 ~]$ docker run -d --name test-web --mount source=http-files,target=/usr/local/apache2/htdocs httpd

09fd2cf1b701287939065cba06d531ae6119b022459b0a865ccfed17e6b28ab0

[user@tcox3 ~]$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

09fd2cf1b701 httpd "httpd-foreground" 4 seconds ago Up 3 seconds 80/tcp test-web

6. Using the appropriate Docker command, find out the container's IP address and note it

[user@tcox3 ~]$ docker inspect test-web | grep IPAddress

"SecondaryIPAddresses": null,

"IPAddress": "172.17.0.2",

"IPAddress": "172.17.0.2",

7. Execute the 'curl' command against that IP address to display the Apache website running on the container, verify the output is from your created 'index.html' file

[user@tcox3 ~]$ curl http://172.17.0.2

This is my test website index file

8. Make a change to the 'index.html' file on the container's host and save the file. Rerun the 'curl' command to verify the container's website is now displaying the new value

NOTE: These commands are run as ROOT user to access the HOST directory the index file is in  
[root@tcox3 ~]# echo "This is a CHANGED website file" > /var/lib/docker/volumes/http-files/\_data/index.html

[root@tcox3 ~]# curl http://172.17.0.2

This is a CHANGED website file

## Exercise: Creating a Bind Mount to Link Container Filesystem to Host Filesystem

|  |
| --- |
| You will use Docker CE to create a container that contains a link to a mount on the host (and it's contents). Please complete the following tasks:  1. Create a directory in your home directory called 'content'. Within this directory, place a file called 'index.html' containing any text you wish.  2. Using the appropriate Docker CE command, download the image called 'httpd:latest' to your system.  3. Install the 'elinks' web browser to complete testing of your site  4. Instantiate a container on this single host with the following characteristics:    \* name the container 'testweb'    \* map container port 80 to the host port 80    \* create a bind mapping from the container directory of /usr/local/apache2/htdocs to the local host directory you created above using the complete path    \* base the container on the 'httpd' image downloaded above    \* run the contained in 'detached' mode  5. Verify the container is running  6. Use 'elinks' to connect to your local IP over port 80 and verify the created index file created above is displayed |

You will use Docker CE to create a container that contains a link to a mount on the host (and it's contents). Please complete the following tasks:

1. Create a directory in your home directory called 'content'. Within this directory, place a file called 'index.html' containing any text you wish.

[user@tcox6 ~]$ mkdir content

[user@tcox6 ~]$ echo "This is a test web site in a container" > content/index.html

[user@tcox6 ~]$ ls -al content

total 8

drwxrwxr-x. 2 user user 23 Dec 21 16:32 .

drwx------. 10 user user 4096 Dec 21 16:31 ..

-rw-rw-r--. 1 user user 39 Dec 21 16:32 index.html

2. Using the appropriate Docker CE command, download the image called 'httpd:latest' to your system.

[user@tcox6 ~]$ docker pull httpd

Using default tag: latest

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

3. Install the 'elinks' web browser to complete testing of your site

[user@tcox6 ~]$ sudo yum install elinks

[sudo] password for user:

Loaded plugins: fastestmirror

Loading mirror speeds from cached hostfile

\* base: mirror.lax.hugeserver.com

4. Instantiate a container on this single host with the following characteristics:

  \* name the container 'testweb'

  \* map container port 80 to the host port 80

  \* create a bind mapping from the container directory of /usr/local/apache2/htdocs to the local host directory you created above using the complete path

  \* base the container on the 'httpd' image downloaded above

  \* run the contained in 'detached' mode

[user@tcox6 ~]$ docker run -d --name testweb -p 80:80 --mount type=bind,source=/home/user/content,target=/usr/local/apache2/htdocs httpd

054b6dc6c49aaeb6a7ef7000794c0212eba5ec773b04ba3ff637a402a812220a

5. Verify the container is running

[user@tcox6 ~]$ docker ps

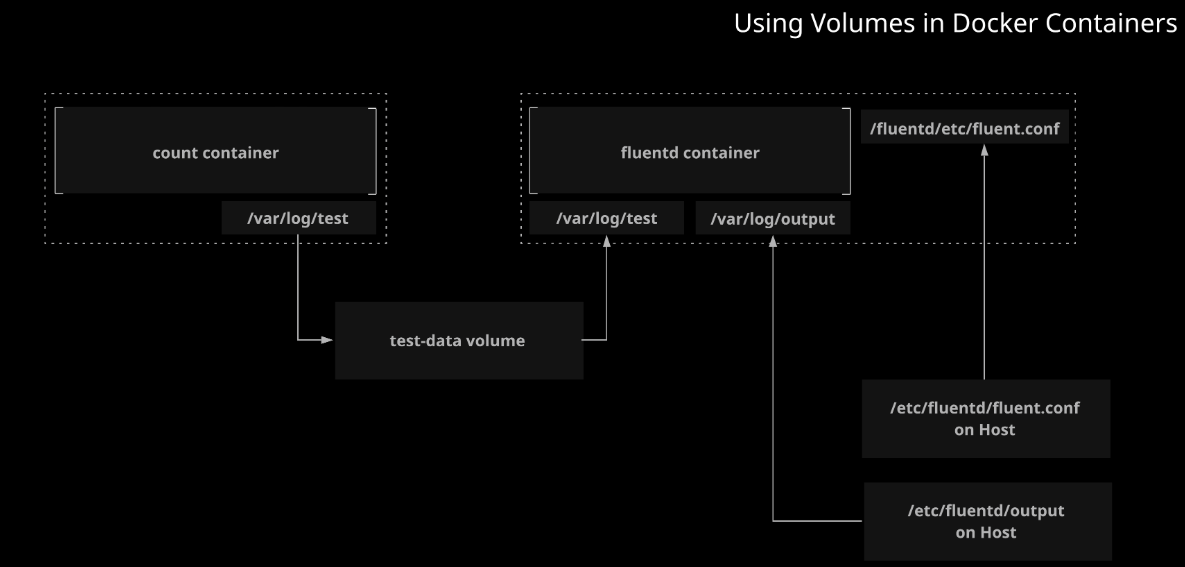
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

054b6dc6c49a httpd "httpd-foreground" 29 seconds ago Up 27 seconds 0.0.0.0:80->80/tcp testweb

6. Use 'elinks' to connect to your local IP over port 80 and verify the created index file created above is displayed

[user@tcox6 ~]$ elinks http://172.31.19.60  
NOTE: Will display the created file in the console browser if commands completed correctly!

# Using Volumes in Docker Containers



Containers are designed to be ephemeral, so when you need persistent data, it is usually not a good idea to store it directly in the container's file system. This is where Docker volumes come into play. Docker volumes allow you to store persistent data outside the container itself, providing greater flexibility in what you can do with your data.

In this lab, you will have the opportunity to solve a complex problem using Docker volumes. You will have a chance to work with both shared volumes and bind mounts in order to implement two containers which work together, one container transforming data created by the other. This will give you some practice in working with volumes, as well as some insight into the many ways in which Docker volumes can be used.

## **Solution**

1. Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Create the Shared Volume and Counter Container**

1. Create a shared volume.

docker volume create test-data

1. Create the counter container with the provided command, and mount the shared volume to the container.
2. docker run --name counter -d \
3. --mount type=volume,source=test-data,destination=/var/log/test \
4. busybox \

sh -c 'i=0; while true; do echo "$i: $(date)" >> /var/log/test/1.log; i=$((i+1)); sleep 1; done'

You can confirm that the counter container is generating data by examining the file inside the container:

docker exec counter cat /var/log/test/1.log

### **Create the fluentd Container**

1. Create the fluentd container and mount the shared volume, the config file, and the output directory to it.
2. docker run --name fluentd -d \
3. --mount type=volume,source=test-data,destination=/var/log/input \
4. --mount type=bind,source=/etc/fluentd/fluent.conf,destination=/fluentd/etc/fluent.conf \
5. --mount type=bind,source=/etc/fluentd/output,destination=/var/log/output \
6. --env FLUENTD\_ARGS="-c /fluentd/etc/fluent.conf" \

k8s.gcr.io/fluentd-gcp:1.30

1. Verify that the fluentd container is generating output on the Docker host.

ls /etc/fluentd/output

You should see some files containing the transformed log data.

# Persistent Data Volumes

# 

### **SpaceBones 2 - The Quest for More Yummies**

Great job, cadet! SpaceBonians have created a special message for you! To view it, create a Docker volume containing the contents of /content-dockerquest-spacebones/volumes/ named missionstatus, then mount the volume on a new httpd container named fishin-mission running on port :80 to view it!

### **Instructions**

* if not already present, clone the content-dockerquest-spacebones repo on your machine

git clone https://github.com/linuxacademy/content-dockerquest-spacebones.git

* create a new volume named missionstatus

docker volume create missionstatus

* Use Docker commandline to print all information on the missionstatus volume.

docker volume inspect missionstatus

* drop to root, then copy the contents of content-dockerquest-spacebones/volumes/ to missionstatus data directory

sudo -i

cp -r /home/user/content-dockerquest-spacebones/volumes/ /var/lib/docker/volumes/missionstatus/\_data/

* Exit root, then start a new container named fishin-mission using the httpd base image available on DockerHub, running on port 80.

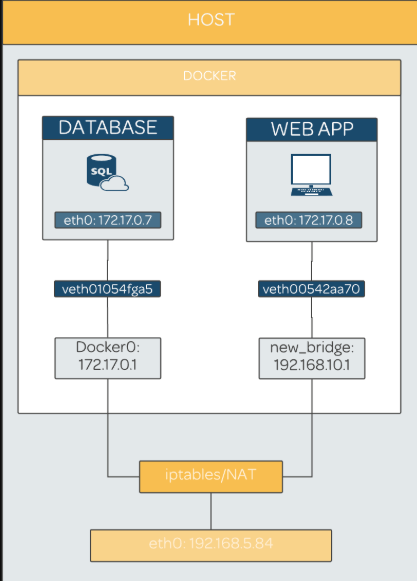
docker run -d -p 80:80 --name fishin-mission \

--mount source=missionstatus,target=/usr/local/apache2/htdocs httpd

* Visit the server IP in your browser for a special thank you from SpaceBones residents!

# Container Networking with Networks

# 

Negotiations between President Squawk & Droolidian officials are now complete, & now Droolidia is preparing to deliver the first treat rations to the nation of SpaceBones. Using the details below, create a new Docker network named borkspace using the 192.168.10.0/24 network range, with the gateway IP address 192.168.10.250. Once the borkspace network is created, use it to launch a new app named treattransfer in interactive mode using the spacebones/nyancat:latest. Once the container is running, you should see a live view of Droolidian cadets running to the rescue! 

* create network

docker network create --driver=bridge --subnet=192.168.10.0/24 --gateway=192.168.10.250 borkspace

* verify that network is created

docker network ls

docker network inspect borkspace

* launch treattransfer container using the borkspace network

docker run -it --name treattransfer --network=borkspace -p 80:80 nyancat

* exit the container

CTRL+c

## Exercise: Utilize External DNS With Your Containers

|  |
| --- |
| Your company's DNS servers are having issues but you need to launch a web container that can resolve outside named resources. Your tasks are as follows:  1. Using the Docker base image for Ubuntu, create a container with the following characteristics:    - Interactive    - Attached to Terminal    - Using Google Public DNS    - Named 'mycontainer1'  2. Exit the container from Step #1. Using the Docker base image for Ubuntu, create a container with the following characteristics:    - Interactive    - Attached to Terminal    - Using Google Public DNS    - Using Domain Search "mydomain.local"    - Named 'mycontainer2'  3. Exit the container from Step #2. List all the containers. List all characteristics inspected from 'mycontainer2' and then remove and verify removal of all containers. |

Your company's DNS servers are having issues but you need to launch a web container that can resolve outside named resources. Your tasks are as follows:

1. Using the Docker base image for Ubuntu, create a container with the following characteristics:

  - Interactive

  - Attached to Terminal

  - Using Google Public DNS

  - Named 'mycontainer1'

[user@tcox6 ~]$ docker container run -it --name mycontainer1 --dns 8.8.8.8 ubuntu

root@fa6b1f65e621:/# cat /etc/resolv.conf

search mylabserver.com

nameserver 8.8.8.8

root@fa6b1f65e621:/# exit

exit

2. Exit the container from Step #1. Using the Docker base image for Ubuntu, create a container with the following characteristics:

  - Interactive

  - Attached to Terminal

  - Using Google Public DNS

  - Using Domain Search "mydomain.local"

  - Named 'mycontainer2'

[user@tcox6 ~]$ docker container run -it --name mycontainer2 --dns 8.8.8.8 --dns-search "mydomain.local" ubuntu

root@64892f5624cb:/# cat /etc/resolv.conf

search mydomain.local

nameserver 8.8.8.8

root@64892f5624cb:/# exit

exit

3. Exit the container from Step #2. List all the containers. List all characteristics inspected from 'mycontainer2' and then remove and verify removal of all containers.

[user@tcox6 ~]$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

64892f5624cb ubuntu "/bin/bash" About a minute ago Exited (0) About a minute ago mycontainer2

fa6b1f65e621 ubuntu "/bin/bash" 3 minutes ago Exited (0) 2 minutes ago mycontainer1

[user@tcox6 ~]$ docker container inspect mycontainer2

[

{

"Id": "64892f5624cb257063987203b8985a15039e14b0e4ad1b12ab651f6a84bc74b6",

"Created": "2017-12-21T18:12:42.598971852Z",

"Path": "/bin/bash",

[user@tcox6 ~]$ docker rm `docker ps -a -q`

64892f5624cb

fa6b1f65e621

[user@tcox6 ~]$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[user@tcox6 ~]$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

## Exercise: Create a New Bridge Network and Assign a Container To It

|  |
| --- |
| Your development team would like a separate container test network on their Docker host to assign and test certain workloads. You have been asked to create one and verify it works as expected. Your tasks are as follows:  1. Display existing Docker networks and names on your host  2. Create a new network as follows:    \* use the 'bridge' driver    \* assigns IP addresses in the '192.168.1.0/24' network range    \* uses a gateway address of 192.168.1.250    \* called 'dev\_bridge'  3. Display all Docker networks on the host  4. Pull the 'httpd' image and install locally  5. Create a container called 'testweb' based on the image in the previous step as follows:    \* assigned to new 'dev\_bridge' network on start  6. Using the appropriately formatted Docker command output, display the container's IP(s) to include the new network |

Your development team would like a separate container test network on their Docker host to assign and test certain workloads. You have been asked to create one and verify it works as expected. Your tasks are as follows:

1. Display existing Docker networks and names on your host

[user@tcox4 ~]$ docker network ls

NETWORK ID NAME DRIVER SCOPE

3092c22e2f99 bridge bridge local

5ee8aa70a57c docker\_gwbridge bridge local

d50effa7e823 host host local

koijigc21abd ingress overlay swarm

1c1832dfeb9f none null local

2. Create a new network as follows:

  \* use the 'bridge' driver

  \* assigns IP addresses in the '192.168.1.0/24' network range

  \* uses a gateway address of 192.168.1.250

  \* called 'dev\_bridge'

[user@tcox4 ~]$ docker network create --driver=bridge --subnet=192.168.1.0/24 --gateway=192.168.1.250 dev\_bridge

9172f9fed208df5717df937dc7d5e284ce6a32b2e7ce72ffeaa325c169fdb842

3. Display all Docker networks on the host

[user@tcox4 ~]$ docker network ls

NETWORK ID NAME DRIVER SCOPE

3092c22e2f99 bridge bridge local

9172f9fed208 dev\_bridge bridge local

5ee8aa70a57c docker\_gwbridge bridge local

d50effa7e823 host host local

koijigc21abd ingress overlay swarm

1c1832dfeb9f none null local

4. Pull the 'httpd' image and install locally

[user@tcox4 ~]$ docker pull httpd

Using default tag: latest

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

5. Create a container called 'testweb' based on the image in the previous step as follows:

  \* assigned to new 'dev\_bridge' network on start

[user@tcox4 ~]$ docker run -d --name testweb --network=dev\_bridge httpd

8d6ce33684248269da414fc3f8cd2b084ce6b7c787461278673637764c008a6a

6. Using the appropriately formatted Docker command output, display the container's IP(s) to include the new network

[user@tcox4 ~]$ docker container inspect --format="{{.NetworkSettings.Networks.dev\_bridge.IPAddress}}" testweb

192.168.1.1

# Using a Docker Bridge Network

## **Introduction**

By default, all containers on a host can communicate with one another over a default bridge network. However, in some cases, you may want to isolate groups of containers by allowing them to communicate over their own isolated network.

In this lab, you will have the opportunity to create a custom bridge network designed to facilitate communication between containers on a Docker host.

## **Solution**

1. Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Create the Bridge Network**

1. Create a bridge network called prices-net.

docker network create --driver bridge prices-net

### **Create the base-price Container**

1. Create a container for the component that serves base prices.

docker run --name base-price -d --network prices-net linuxacademycontent/prices-base-price:1

### **Create the sales Container**

1. Create a container for the component that serves products on sale.

docker run --name sales -d --network prices-net linuxacademycontent/prices-sales:1

### **Create the total-price Container**

1. Create a container for the component that serves the total prices of products.

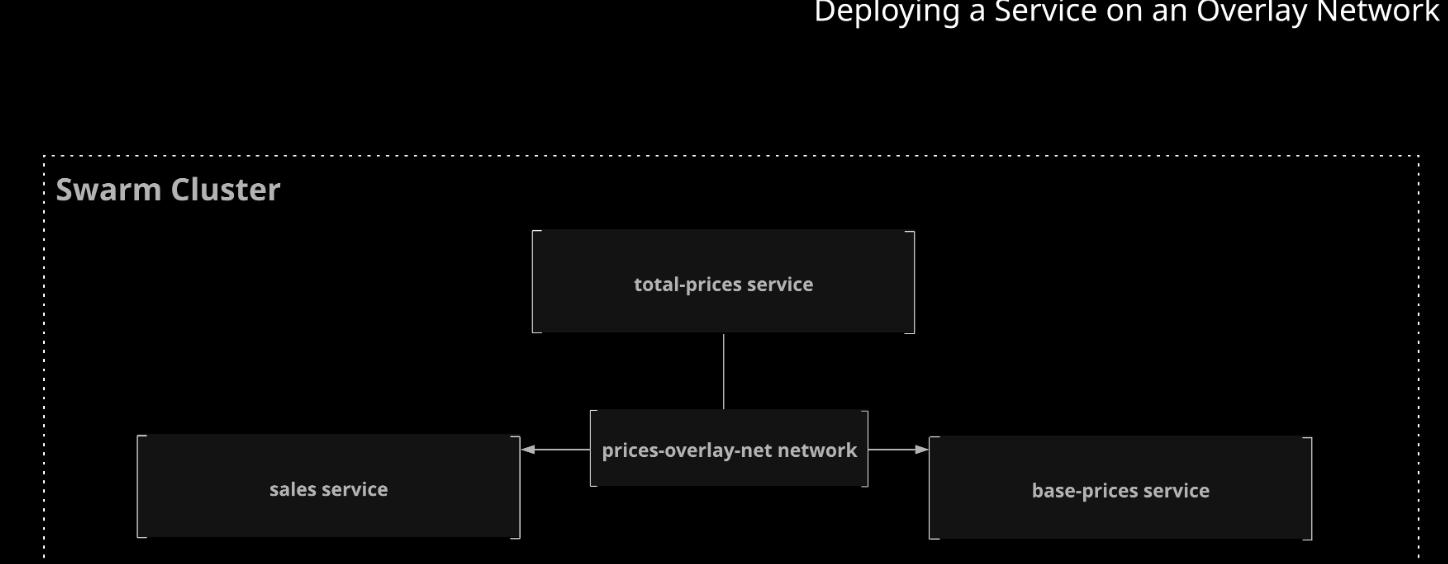
docker run --name total-price -d --network prices-net -p 8080:80 linuxacademycontent/prices-total-price:1

1. Verify that everything is set up correctly.

curl localhost:8080

You should get a list of products and their final prices. The total-prices container calculates these prices by first querying the other two containers. This communication takes place over the prices-net bridge network.

**Deploying a Service on an Overlay Network**



Bridge networks are a powerful tool for controlling communication between containers on a single host, but what if you need to provide isolated networking between containers in Docker Swarm? With Docker Swarm, you can use custom overlay networks to allow groups of containers to communicate transparently, even if they are running on different swarm nodes.

In this lab, you will have the opportunity to work with overlay networks. You will set up a custom overlay network and deploy three different services that communicate with each other using this network.

Solution

1. Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

Create the Overlay Network

1. Create the prices-overlay-net overlay network.

docker network create --driver overlay prices-overlay-net

Create the base-price Service

1. Create the base-price service.

docker service create --name base-price --network prices-overlay-net --replicas 3 linuxacademycontent/prices-base-price:1

Create the sales Service

1. Create the sales service.

docker service create --name sales --network prices-overlay-net --replicas 3 linuxacademycontent/prices-sales:1

Create the total-price Service

1. Create the total-price service.

docker service create --name total-price --network prices-overlay-net --replicas 2 -p 8080:80 linuxacademycontent/prices-total-price:1

1. Verify that you get the total price data.

curl localhost:8080

You should see a list of products and the total price for each. These prices are calculated by communicating with the base-price and sales services using the custom overlay network.

# Build Services with Docker Compose

## **Introduction**

For the last six months, the Acme Anvil Corporation has been migrating some of their bare metal infrastructure to Docker containers. Your team wants to find an easier way to deploy applications that consist of multiple containers and has decided to use Docker Compose. You have been tasked with setting up an internal blog so the team can write technical articles. This blog will consist of two services: a Ghost Blog service and a MySQL service. Both services will use volumes for persistent storage.

## **Solution**

Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

Become the root user:

sudo su -

### **Create a Ghost Blog and MySQL Service**

1. Create a docker-compose.yml file in the root directory.

vi docker-compose.yml

1. Add the following contents to it:
2. version: '3'
3. services:
4. ghost:
5. image: ghost:1-alpine
6. container\_name: ghost-blog
7. restart: always
8. ports:
9. - 80:2368
10. environment:
11. database\_\_client: mysql
12. database\_\_connection\_\_host: mysql
13. database\_\_connection\_\_user: root
14. database\_\_connection\_\_password: P4sSw0rd0!
15. database\_\_connection\_\_database: ghost
16. volumes:
17. - ghost-volume:/var/lib/ghost
18. depends\_on:
19. - mysql
20. mysql:
21. image: mysql:5.7
22. container\_name: ghost-db
23. restart: always
24. environment:
25. MYSQL\_ROOT\_PASSWORD: P4sSw0rd0!
26. volumes:
27. - mysql-volume:/var/lib/mysql
28. volumes:
29. ghost-volume:

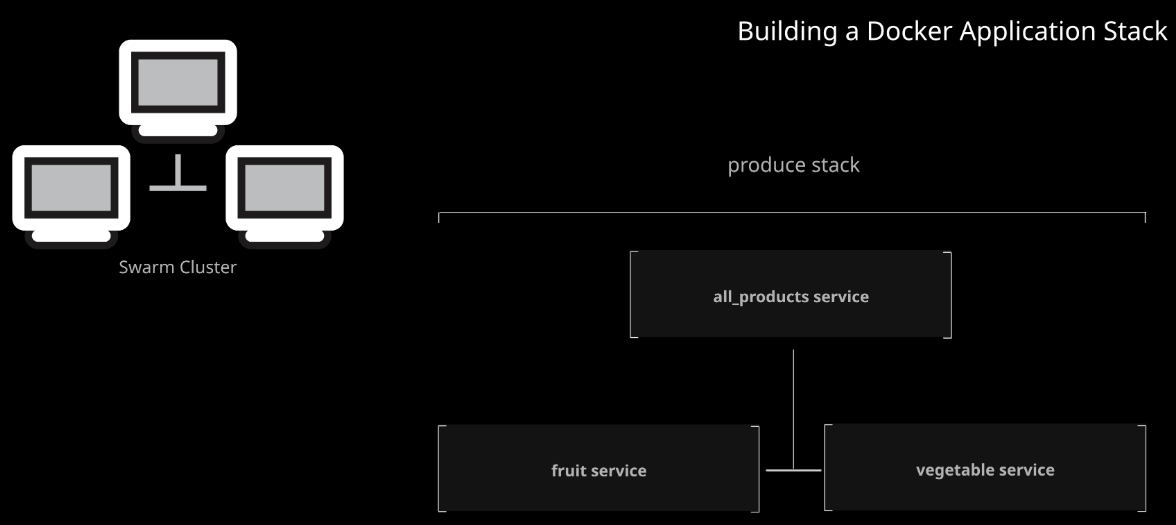
mysql-volume:

### **Bring Up the Ghost Blog Service**

1. Start up the Docker Compose service.

docker-compose up -d

# Building a Docker Application Stack



## **Introduction**

Stacks are one of the most powerful orchestration features available in Docker Swarm. They allow you to easily manage complex applications consisting of multiple interdependent components running in separate containers.

In this lab, you will have the opportunity to work with Docker stacks by building a multi-component application as a Docker stack. You will also learn how to manage existing stacks by scaling a stack's services after it has already been deployed. This will give you some hands-on insight into Docker stacks.

## **Solution**

1. Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Build and deploy the application stack**

1. Create an empty project directory with a Docker compose YAML file inside.
2. cd ~/
3. mkdir produce
4. cd produce

vi produce.yml

1. Build a stack definition in produce.yml to meet the provided specifications.
2. version: '3'
3. services:
4. fruit:
5. image: linuxacademycontent/fruit-service:1.0.1
6. vegetables:
7. image: linuxacademycontent/vegetable-service:1.0.0
8. all\_products:
9. image: linuxacademycontent/all-products:1.0.0
10. ports:
11. - "8080:80"
12. environment:
13. - FRUIT\_HOST=fruit
14. - FRUIT\_PORT=80
15. - VEGETABLE\_HOST=vegetables

- VEGETABLE\_PORT=80

1. Deploy the stack using the compose file.

docker stack deploy -c produce.yml produce

1. Verify that the stack is working.

curl localhost:8080

Note that after deploying, it may take a few moments for the stack to become responsive. You can check the status of the services with docker stack services produce. Once the services are up and running, you should get some JSON data containing a combined list of fruits and vegetables.

### **Scale the Fruit and Vegetable services in the stack**

1. Set the number of replicas to 3 for the Fruit and Vegetable services in the compose file.

vi produce.yml

version: '3'

services:

fruit:

image: linuxacademycontent/fruit-service:1.0.1

deploy:

replicas: 3

vegetables:

image: linuxacademycontent/vegetable-service:1.0.0

deploy:

replicas: 3

all\_products:

image: linuxacademycontent/all-products:1.0.0

ports:

- "8080:80"

environment:

- FRUIT\_HOST=fruit

- FRUIT\_PORT=80

- VEGETABLE\_HOST=vegetables

- VEGETABLE\_PORT=80

1. Redeploy the stack using the compose file.

docker stack deploy -c produce.yml produce

1. Verify that the stack is still working.

curl localhost:8080

You should get some JSON data containing a combined list of fruits and vegetables.

Use docker stack services produce to see that the number of replicas for the Fruit and Vegetable services is now 3.

# Building Services in Docker

## 

Services are the most basic and straightforward way to run containers using a Docker swarm. They allow you to execute multiple replica containers across all nodes in the Swarm cluster.

In this lab, you will have the opportunity to work with Docker services. You will practice scaling services by changing the number of replicas for an existing service. You will also have the opportunity to create a new service and run it in the cluster.

## **Solution**

1. Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Scale the products-fruit service to 5 replicas**

1. Scale the service.

docker service update --replicas 5 products-fruit

You can also do it this way (both do the same thing):

docker service scale products-fruit=5

### **Create the products-vegetables service**

1. Create the products-vegetables service.

docker service create --name products-vegetables -p 8081:80 --replicas 3 linuxacademycontent/vegetable-service:1.0.0

1. Verify that the service is working.

curl localhost:8081

You should see some JSON data containing a list of vegetables.

# Building a Docker Swarm

Docker swarm allows you to quickly move beyond simply using Docker to run containers. With swarm, you can easily set up a cluster of Docker servers capable of providing useful orchestration features. This lab will allow you to become familiar with the process of setting up a simple swarm cluster on a set of servers. You will configure a swarm master and two worker nodes, forming a working swarm cluster.

### **Instructions**

Your company is ready to move forward with using Docker to run their applications. However, they have some complex container apps that can take advantage of the cluster management and orchestration features of Docker swarm. You have been asked to stand up a simple Docker swarm cluster to be used for some initial testing. A set of servers has already been provisioned for this purpose. The swarm cluster should meet the following criteria:

* One Swarm manager.
* Two worker nodes.
* All nodes should use Docker CE version 5:18.09.5~3-0~ubuntu-bionic.
* Both worker nodes should be joined to the cluster.
* cloud\_user should be able to run docker commands on all three servers.

If you get stuck, feel free to check out the solution video, or the detailed instructions under each objective. Good luck!

## **Solution**

Begin by logging in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Install Docker CE on all three nodes**

1. On all three servers, install Docker CE.

sudo apt-get update

sudo apt-get -y install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

sudo apt-get update

sudo apt-get install -y docker-ce=5:18.09.5~3-0~ubuntu-bionic docker-ce-cli=5:18.09.5~3-0~ubuntu-bionic containerd.io

1. Add cloud\_user to the Docker group so that you can run docker commands as cloud\_user.

sudo usermod -a -G docker cloud\_user

Log out each server, then log back in.

1. You can verify the installation on each server like so:

docker version

### **Configure the swarm manager**

1. On the swarm manager server, initialize the swarm. Be sure to replace <swarm manager private IP> in this command with the actual Private IP of the swarm manager (NOT the public IP).

docker swarm init --advertise-addr <swarm manager private IP>;

### **Join the worker nodes to the cluster**

1. On the swarm manager, get a join command with a token:

docker swarm join-token worker

This should provide a command that begins docker swarm join .... Copy that command and run it on both worker servers.

1. Go back to the swarm manager and list the nodes.

docker node ls

Verify that you can see all three servers listed (including the manager). All three should have a status of READY. Once all three servers are ready, you have built your own Docker swarm cluster!

## Create a Docker Service on Your Swarm and Expose Service Ports to Each Host

|  |
| --- |
| You will be using a THREE NODE Docker Swarm (a Single Manager and Two Worker Nodes). If you do not have that set up at this point, please see the exercise in doing so or complete that setup before you begin this exercise.  The tasks that need to be completed include the following:  1. Using the appropriate command, pull the latest 'httpd' image from Docker Hub.  2. Create a service on your cluster that meets the following requirements:    \* name the service 'testweb'    \* map the service web port of 80 to the underlying service hosts port of 80    \* base it on the 'httpd' service above    \* initialize the service with three replicas  3. Verify the service is running  4. List the all nodes and verify replicas are running on all three  5. Using whatever method you choose, check that the default site in the service is being served when you check port 80 on all three nodes |

You will be using a THREE NODE Docker Swarm (a Single Manager and Two Worker Nodes). If you do not have that set up at this point, please see the exercise in doing so or complete that setup before you begin this exercise.

The tasks that need to be completed include the following:

1. Using the appropriate command, pull the latest 'httpd' image from Docker Hub.

[user@tcox4 ~]$ docker pull httpd

Using default tag: latest

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

2. Create a service on your cluster that meets the following requirements:

  \* name the service 'testweb'

  \* map the service web port of 80 to the underlying service hosts port of 80

  \* base it on the 'httpd' service above

  \* initialize the service with three replicas

[user@tcox4 ~]$ docker service create --name testweb -p 80:80 --replicas 3 httpd

xeyuhxsv2j5uyt9rsv3uj3ue9

3. Verify the service is running

[user@tcox4 ~]$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

xeyuhxsv2j5u testweb replicated 3/3 httpd:latest \*:80->80/tcp

4. List the all nodes and verify replicas are running on all three

[user@tcox4 ~]$ docker service ps testweb

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

qfbrk9v1tl4l testweb.1 httpd:latest tcox4.mylabserver.com Running Running 59 seconds ago

omuirsvknr6m testweb.2 httpd:latest tcox5.mylabserver.com Running Running 28 seconds ago

0fey7m5i4dkd testweb.3 httpd:latest tcox6.mylabserver.com Running Running 58 seconds ago

5. Using whatever method you choose, check that the default site in the service is being served when you check port 80 on all three nodes

[user@tcox4 ~]$ elinks http://tcox4  
[user@tcox4 ~]$ elinks http://tcox5  
[user@tcox4 ~]$ elinks http://tcox6  
NOTE: All should show the 'It Works' default page

## Exercise: Create a Swarm Cluster

|  |
| --- |
| 1. You will need to choose one of the three servers you have provisioned as the 'master' or 'manager' node in your cluster. The other two will function as nodes. On your manager, initialize the swarm cluster for your environment with the appropriate Docker command.  2. Clear your screen, do NOT note the token value. In order to demonstrate how to retrieve the previously displayed token, from your terminal, issue the command on your master that will retrieve the token value and command that each node will need to join the swarm cluster.  3. On each of your other two node servers, join the cluster (i.e. the manager) that you initialized above using the necessary token.  4. On your manager node, display all of the nodes within your cluster.  5. On your manager node, display ALL information about the system that Docker can see along with the details of the cluster name, management nodes and nodes joined. |

1. You will need to choose one of the three servers you have provisioned as the 'master' or 'manager' node in your cluster. The other two will function as nodes. On your manager, initialize the swarm cluster for your environment with the appropriate Docker command.

[user@tcox1 ~]$ sudo docker swarm init --advertise-addr 172.31.118.114  
  
Swarm initialized: current node (31uejwo9gyiklfry3htehj65h) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join \

--token SWMTKN-1-2kl42kf4xeb1t08vcvwqekovknbmw68ny5o01rpyoepmvkrwtv-8q0chh9moe4hs4v6g8ogewmgv \

172.31.118.144:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

2. Clear your screen, do NOT note the token value. In order to demonstrate how to retrieve the previously displayed token, from your terminal, issue the command on your master that will retrieve the token value and command that each node will need to join the swarm cluster.

[user@tcox1 swarm-info]$ sudo docker swarm join-token worker

To add a worker to this swarm, run the following command:

docker swarm join \

--token SWMTKN-1-2kl42kf4xeb1t08vcvwqekovknbmw68ny5o01rpyoepmvkrwtv-8q0chh9moe4hs4v6g8ogewmgv \  
 172.31.118.144:2377

3. On each of your other two node servers, join the cluster (i.e. the manager) that you initialized above using the necessary token.

[user@tcox4 ~]$ sudo docker swarm join --token SWMTKN-1-1anvcitaky0o7sp39qhzraucmcjm0wm2828tbzdr7wa0pauz5d-08pkm4jd4pbp1oyfbolxy0bav 172.31.118.144:2377

This node joined a swarm as a worker.  
  
[user@tcox5 ~]$ sudo docker swarm join --token SWMTKN-1-1anvcitaky0o7sp39qhzraucmcjm0wm2828tbzdr7wa0pauz5d-08pkm4jd4pbp1oyfbolxy0bav 172.31.118.144:2377

This node joined a swarm as a worker.

4. On your manager node, display all of the nodes within your cluster.

[user@tcox1 swarm-info]$ sudo docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

2gv6ttpmmqi7s41yv62e7zpxv tcox5.mylabserver.com Ready Active

89s726omssxe2b7jr6sp1zhp3 \* tcox1.mylabserver.com Ready Active Leader

ats02vit0ylzm8b9hf0b64dad tcox4.mylabserver.com Ready Active

5. On your manager node, display ALL information about the system that Docker can see along with the details of the cluster name, management nodes and nodes joined.

[user@tcox1 swarm-info]$ sudo docker info

Containers: 0

Running: 0

**Start a Service and Scale It Within Your Swarm**

|  |
| --- |
| *NOTE: This exercise can be completed on the Linux Academy Cloud Servers running CentOS 7. You will need a total of THREE Cloud Servers running CentOS 7 started to complete this exercise and you will need to have installed Docker and started the Docker service on each one. Although we are performing the tasks as the 'user' account on each server, you may use 'root' or any other user account with sufficient*sudo*privileges. Finally, add EACH Cloud Server's name and PRIVATE IP ADDRESS to the /etc/hosts file on ALL THREE servers. We need to use the private IP addresses for this exercise. To pick up where this exercise begins, you will either need to install a manager and two nodes for use in your swarm OR use the servers from the previous exercise here - https://linuxacademy.com/cp/exercises/view/id/601/module/150 -  NOTE: You can use the Linux Academy 'Docker Certification' Cloud Server in lieu of completing a fresh installation and configuration of your environment.*   * Execute the appropriate command on your manager to display the current manager and node list in your swarm cluster * Create a service in your swarm cluster with the following attributes:   1. Name the application "devweb"   2. Remap/Redirect the underlying host environment's port 80 to the service application port 80   3. Base the service off of the default 'httpd' image that is available on the Docker Hub (default repository) * Display the current services that are running on your cluster with the number of replicas each has * Execute the command that will tell you which node the service is currently running on * Scale up your service so that it is running THREE replicas and verify by listing the nodes that the service is running on in your environment * Scale back down your service so that it is running TWO replicas and verify by listing the nodes that the service is running on in your environment |

* Execute the appropriate command on your manager to display the current manager and node list in your swarm cluster

[user@tcox2 ~]$ sudo docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

brz5mgidk77hg6hpn5w4yp3j2 tcox4.mylabserver.com Ready Active

eb3sio0bu3lzhfgpcu5pscq9y tcox3.mylabserver.com Ready Active

ebcivaccyqsxwkgxvqqkzelv5 \* tcox2.mylabserver.com Ready Active Leader

* Create a service in your swarm cluster with the following attributes:
  1. Name the application "devweb"
  2. Remap/Redirect the underlying host environment's port 80 to the service application port 80
  3. Base the service off of the default 'httpd' image that is available on the Docker Hub (default repository)

[user@tcox2 ~]$ sudo docker service create --name devweb -p 80:80 httpd

[sudo] password for user:

0zad69e49ghj5wbs8cmi4io8a

* Display the current services that are running on your cluster with the number of replicas each has

[user@tcox2 ~]$ sudo docker service ls

ID NAME REPLICAS IMAGE COMMAND

0zad69e49ghj devweb 1/1 httpd

* Execute the command that will tell you which node the service is currently running on

[user@tcox2 ~]$ sudo docker service ps devweb ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR 377vxg5ay18p38ow3clq32syp devweb.1 httpd tcox2.mylabserver.com Running Running 5 minutes ago

* Scale up your service so that it is running THREE replicas and verify by listing the nodes that the service is running on in your environment

[user@tcox2 ~]$ sudo docker service scale devweb=3 devweb scaled to 3

[user@tcox2 ~]$ sudo docker service ps devweb

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR

377vxg5ay18p38ow3clq32syp devweb.1 httpd tcox2.mylabserver.com Running Running 7 minutes ago

52m31m8u611ip4ncgp07so3nj devweb.2 httpd tcox4.mylabserver.com Running Running 33 seconds ago

enzhtf6e6htgwemwvj3o8gcdz devweb.3 httpd tcox3.mylabserver.com Running Running 32 seconds ago

* Scale back down your service so that it is running TWO replicas and verify by listing the nodes that the service is running on in your environment

[user@tcox2 ~]$ sudo docker service scale devweb=2 devweb scaled to 2 [user@tcox2 ~]$ sudo docker service ps devweb ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR 377vxg5ay18p38ow3clq32syp devweb.1 httpd tcox2.mylabserver.com Running Running 10 minutes ago 52m31m8u611ip4ncgp07so3nj devweb.2 httpd tcox4.mylabserver.com Running Running 3 minutes ago enzhtf6e6htgwemwvj3o8gcdz devweb.3 httpd tcox3.mylabserver.com Shutdown Shutdown 1 seconds ago

## Exercise: Demonstrate How Failure Affects Service Replicas in a Swarm

|  |
| --- |
| NOTE: Although you can complete this exercise using Ubuntu 16.04LTS or CentOS 7 Linux Academy Cloud Servers (which you would then have to have installed Docker CE as detailed in the course already), we have made available a special pre-configured 'Docker Certification' CentOS 7 system for your use with the course and all exercises.  You will be using a THREE NODE Docker Swarm (a Single Manager and Two Worker Nodes). If you do not have that set up at this point, please see the exercise in doing so or complete that setup before you begin this exercise.  The tasks that need to be completed include the following:  1. Using the appropriate command, pull the latest 'httpd' image from Docker Hub.  2. Create a service on your cluster that meets the following requirements:    \* name the service 'testweb'    \* map the service web port of 80 to the underlying service hosts port of 80    \* base it on the 'httpd' service above    \* initialize the service with three replicas  3. Verify the service is running  4. List the all nodes and verify replicas are running on all three  5. STOP the Docker service completely on the third node  6. Check the 'testweb' service created above and display once all replicas are listed as running again |

You will be using a THREE NODE Docker Swarm (a Single Manager and Two Worker Nodes). If you do not have that set up at this point, please see the exercise in doing so or complete that setup before you begin this exercise.

The tasks that need to be completed include the following:

1. Using the appropriate command, pull the latest 'httpd' image from Docker Hub.

[user@tcox4 ~]$ docker pull httpd

Using default tag: latest

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

2. Create a service on your cluster that meets the following requirements:

  \* name the service 'testweb'

  \* map the service web port of 80 to the underlying service hosts port of 80

  \* base it on the 'httpd' service above

  \* initialize the service with three replicas

[user@tcox4 ~]$ docker service create --name testweb -p 80:80 --replicas 3 httpd

xeyuhxsv2j5uyt9rsv3uj3ue9

3. Verify the service is running

[user@tcox4 ~]$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

xeyuhxsv2j5u testweb replicated 3/3 httpd:latest \*:80->80/tcp

4. List the all nodes and verify replicas are running on all three

[user@tcox4 ~]$ docker service ps testweb

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

qfbrk9v1tl4l testweb.1 httpd:latest tcox4.mylabserver.com Running Running 59 seconds ago

omuirsvknr6m testweb.2 httpd:latest tcox5.mylabserver.com Running Running 28 seconds ago

0fey7m5i4dkd testweb.3 httpd:latest tcox6.mylabserver.com Running Running 58 seconds ago

5. STOP the Docker service completely on the third node

[user@tcox6 ~]$ sudo systemctl stop docker

[sudo] password for user:

6. Check the 'testweb' service created above and display once all replicas are listed as running again

[user@tcox4 ~]$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

xeyuhxsv2j5u testweb replicated 3/3 httpd:latest \*:80->80/tcp

[user@tcox4 ~]$ docker service ps testweb

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

qfbrk9v1tl4l testweb.1 httpd:latest tcox4.mylabserver.com Running Running 25 minutes ago

omuirsvknr6m testweb.2 httpd:latest tcox5.mylabserver.com Running Running 24 minutes ago

knhltztskrpv testweb.3 httpd:latest tcox4.mylabserver.com Running Running 28 seconds ago

0fey7m5i4dkd \\_ testweb.3 httpd:latest tcox6.mylabserver.com Shutdown Running 48 seconds ago

## Exercise: Reassign a Swarm Worker to Manager

|  |
| --- |
| You will be using a THREE NODE Docker Swarm (a Single Manager and Two Worker Nodes). If you do not have that set up at this point, please see the exercise in doing so or complete that setup before you begin this exercise.  Your swarm needs to be reconfigured, the first step of which will be to reassign existing resources. Your tasks are:  1. On ONE of the worker nodes, force the node to leave the swarm  2. Find the command and token on the manager node to allow other managers to join the swarm  3. Execute the command in the previous step so that your swarm will have an additional manager node  4. Verify the node has been promoted to manager and is reporting as such in the cluster information |

You will be using a THREE NODE Docker Swarm (a Single Manager and Two Worker Nodes). If you do not have that set up at this point, please see the exercise in doing so or complete that setup before you begin this exercise.

Your swarm needs to be reconfigured, the first step of which will be to reassign existing resources. Your tasks are:

1. On ONE of the worker nodes, force the node to leave the swarm

[user@tcox6 ~]$ docker swarm leave

Node left the swarm.  
  
[user@tcox4 ~]$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

q413g1p0qr2dtclbcz7v0kevb \* tcox4.mylabserver.com Ready Active Leader

q5jptu0fqam87s2jb1wmunzug tcox5.mylabserver.com Ready Active

4ucfai1vty001mjca2pxnmvig tcox6.mylabserver.com Down Active

2. Find the command and token on the manager node to allow other managers to join the swarm

[user@tcox4 ~]$ docker swarm join-token manager

To add a manager to this swarm, run the following command:

docker swarm join --token SWMTKN-1-4j26zpgz4zajn9k1ivvwk337fhn1dru9jswk83gxe956q192km-3hsamjxspn0hgctzfv7sfae1i 172.31.24.209:2377

3. Execute the command in the previous step so that your swarm will have an additional manager node

[user@tcox6 ~]$ docker swarm join --token SWMTKN-1-4j26zpgz4zajn9k1ivvwk337fhn1dru9jswk83gxe956q192km-3hsamjxspn0hgctzfv7sfae1i 172.31.24.209:2377

This node joined a swarm as a manager.

4. Verify the node has been promoted to manager and is reporting as such in the cluster information

[user@tcox4 ~]$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

q413g1p0qr2dtclbcz7v0kevb \* tcox4.mylabserver.com Ready Active Leader

q5jptu0fqam87s2jb1wmunzug tcox5.mylabserver.com Ready Active

4ucfai1vty001mjca2pxnmvig tcox6.mylabserver.com Down Active

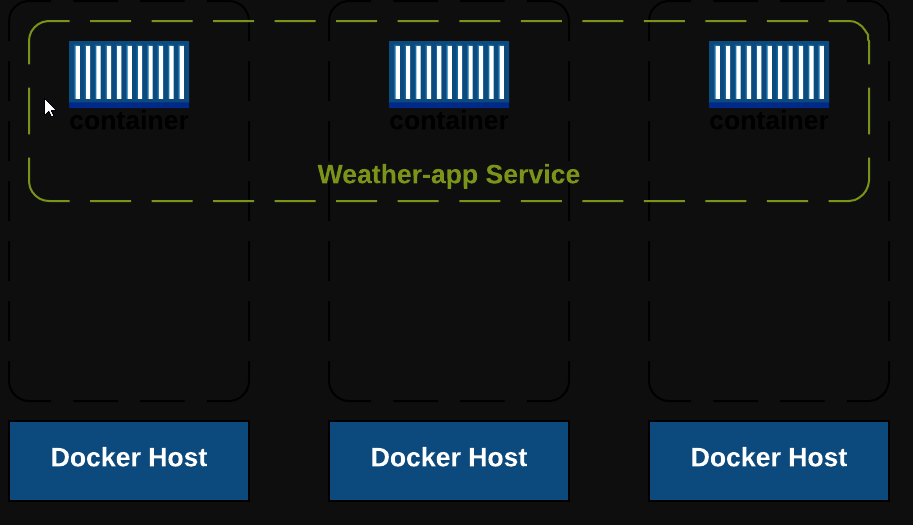
68qi7ik7hvhk7josnzp31kyex tcox6.mylabserver.com Ready Active Reachable

## Hands-On Lab: Configure a Swarm and Scale Services Within Your Cluster

**Setting Up a Docker Swarm**

|  |
| --- |
| **Initialize the Docker swarm.**  Swarm Server 1 will be the swarm master.  Initialize the Docker swarm.  docker swarm init  che**Add additional nodes to the swarm.**keyboard\_arrow\_up  Add your worker nodes to the swarm.  docker swarm join --token TOKEN IP\_ADDRESS:2377  che**Create a swarm service.**keyboard\_arrow\_up  From the master node, create a service to test your swarm configuration.  docker service create --name weather-app --publish published=80,target=3000 --replicas=3 weather-app |

**Setting Up a Docker Swarm**



Introduction

For the last six months, the Acme Anvil Corporation has been migrating some of their bare metal infrastructure to Docker containers. A schism has developed between the members of your team on whether to use Docker Swarm or Kubernetes. Your manager has decided to settle the dispute by creating two competing demos for Docker Swarm and Kubernetes. You have been tasked with helping to create the Docker Swarm demo. Create a swarm with three nodes and then create a service using the weather-app image.

Instructions

In this lab, you are tasked with setting up a Docker swarm.

1. Log in to Swarm Server 1. This will be your swarm master.
2. Initialize a new Docker swarm.
3. Add two worker nodes to the swarm.
4. Create a new swarm service using the weather-app image.
5. Name the service weather-app.
6. Publish port 80 and map it to port 3000 on the container.
7. Create 3 replicas.

Solution

Begin by logging in to **Swarm Server 1** using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

Become the root user:

sudo su -

Repeat these steps for **Swarm Server 2** and **Swarm Server 3** in new tabs.

Initialize the Docker swarm

On **Swarm Server 1**:

1. Initialize the Docker swarm.

docker swarm init

Copy the docker swarm join command that is displayed for the next step.

Add additional nodes to the swarm

Add your worker nodes to the swarm.

1. Run the following command on **Swarm Server 2** and **Swarm Server 3**:

docker swarm join --token TOKEN IP\_ADDRESS:2377

**Note**: This is the command that was copied from the previous step.

Create a swarm service

1. From the master node (**Swarm Server 1**), create a service to test your swarm configuration.

docker service create --name weather-app --publish published=80,target=3000 --replicas=3 weather-app

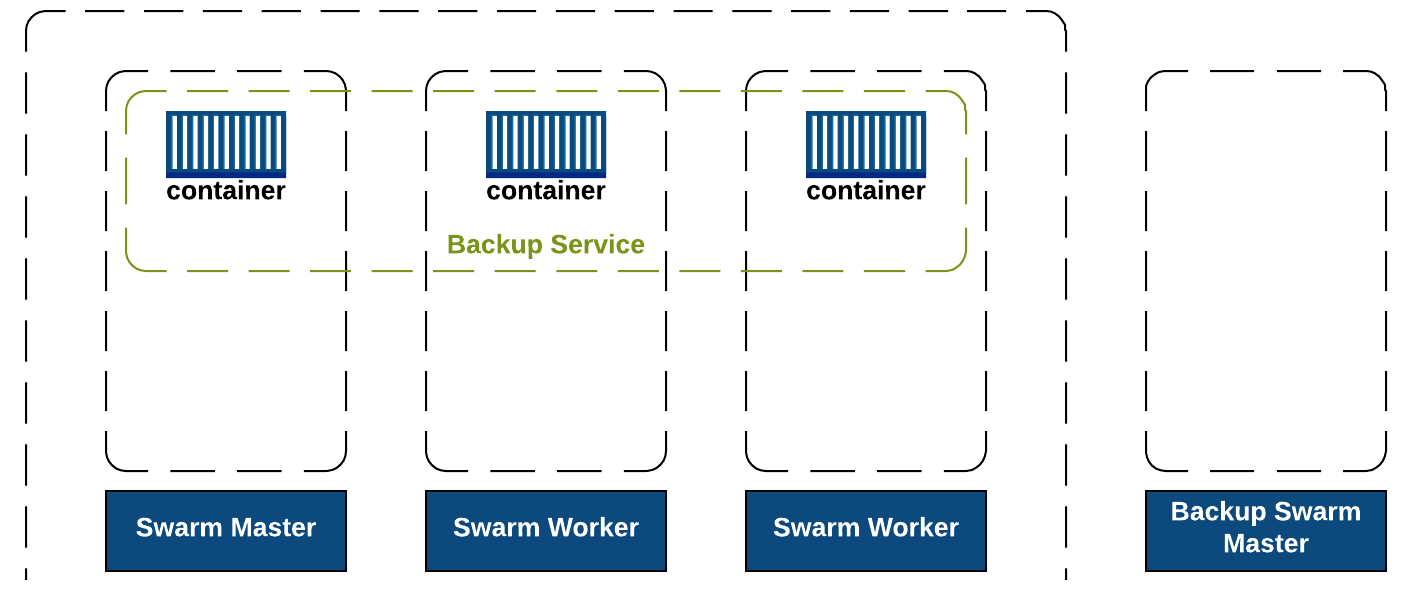
1. Verify that this completed successfully:

docker service ls

We can also paste one of the public IP addresses of our worker nodes in a browser to view the weather app.

Backing Up and Restoring a Docker Swarm

|  |
| --- |
| le**Back up the swarm master.**keyboard\_arrow\_up  Stop the docker service on the master node.  systemctl stop docker  Back up the /var/lib/docker/swarm/ directory.  systemctl stop docker  tar czvf swarm.tgz /var/lib/docker/swarm/  che**Restore the swarm on the backup master.**keyboard\_arrow\_up  Copy the swarm backup from the master node to the backup master:  scp swarm.tar.tgz cloud\_user@BACKUP\_IP\_ADDRESS:/home/cloud\_user/  From the backup master, extract the backup file:  tar xzvf swarm.tar.tgz  Copy the swarm directory to /var/lib/docker/swarm:  cd /var/lib/docker  cp -rf swarm/ /var/lib/docker/  Reinitialize the swarm:  docker swarm init --force-new-cluster  che**Add the worker nodes to the restored cluster.**keyboard\_arrow\_up  Remove each node from the old swarm:  docker swarm leave  Add each node to the backup swarm:  docker swarm join --token TOKEN IP\_ADDRESS:2377  che**Distribute the replicas across the swarm.**keyboard\_arrow\_up  Scale the replicas down to 1:  docker service scale backup=1  Next, scale the replicas up to 3 to distribute the replicas across the swarm:  docker service scale backup=3 |



**Backing Up and Restoring a Docker Swarm**

Introduction

For the last six months, the Acme Anvil Corporation has been migrating some of their bare metal infrastructure to Docker containers. A schism has developed between the members of your team on whether to use Docker Swarm or Kubernetes. To settle the dispute, your manager has decided to create a series of challenges. You have been tasked with creating a demo on how to back up and restore a Docker swarm. You are to set up a Docker swarm with 3 nodes, scale the backup service up to 3 nodes, and back up your master node and restore it to a backup instance.

Instructions

In this lab, you are tasked with backing up and restoring a Docker swarm.

1. Use cat on the file swarm-token.txt to show the join token for your worker nodes.
2. Add 2 worker nodes to the swarm.
3. Scale the backup service up to 3 replicas.
4. Stop the Docker service on the master node and use tar to compress the swarm directory.
5. Use scp to securely copy the swarm tarball to /home/cloud\_user on the backup master node.
6. Use tar to extract the tarball.
7. Stop the Docker service, restore the swarm directory, and start up the Docker service.
8. Force the swarm to reinitialize a new cluster.
9. Add the two worker nodes to the swarm.

Solution

Begin by logging in to all lab servers using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

Become the root user:

sudo su -

Join the worker nodes to the swarm

On **Swarm Server 1**

1. View the docker swarm join command:

cat swarm-token.txt

Copy the docker swarm join command provided in this output.

1. Run the command from the previous step on **Swarm Server 2** and **Swarm Server 3**.
2. View the running services:

docker service ls

1. Scale the backup service up to 3 nodes:

docker service scale backup=3

1. Show the service across our nodes:

docker service ps backup

Back up the swarm master

1. Stop the docker service on the master node.

systemctl stop docker

1. Back up the /var/lib/docker/swarm/ directory.

tar czvf swarm.tgz /var/lib/docker/swarm/

Restore the swarm on the backup master

1. Copy the swarm backup from the master node to the backup master:

**Note**: Be sure to replace BACKUP\_IP\_ADDRESS with the private IP address of the **Backup** server.

scp swarm.tar.tgz cloud\_user@BACKUP\_IP\_ADDRESS:/home/cloud\_user/

1. From the **Backup Swarm Server**, extract the backup file:

tar xzvf swarm.tar.tgz

1. Copy the swarm directory to /var/lib/docker/swarm:

cd /var/lib/docker

cp -rf swarm/ /var/lib/docker/

1. Restart the Docker service:

systemctl restart docker

1. Reinitialize the swarm:

docker swarm init --force-new-cluster

Copy the docker swarm join command from this output.

Add the worker nodes to the restored cluster

On **Swarm Server 2** and **Swarm Server 3**

1. Remove each node from the old swarm:

docker swarm leave

1. Add each node to the backup swarm:

**Note**: This will be the command that was copied from the previous task. You will need to replace the IP address with the private IP address of the **Backup Swarm Server**.

docker swarm join --token TOKEN IP\_ADDRESS:2377

Distribute the replicas across the swarm

On **Backup Swarm Server**

1. Show the current service list:

docker service ps backup

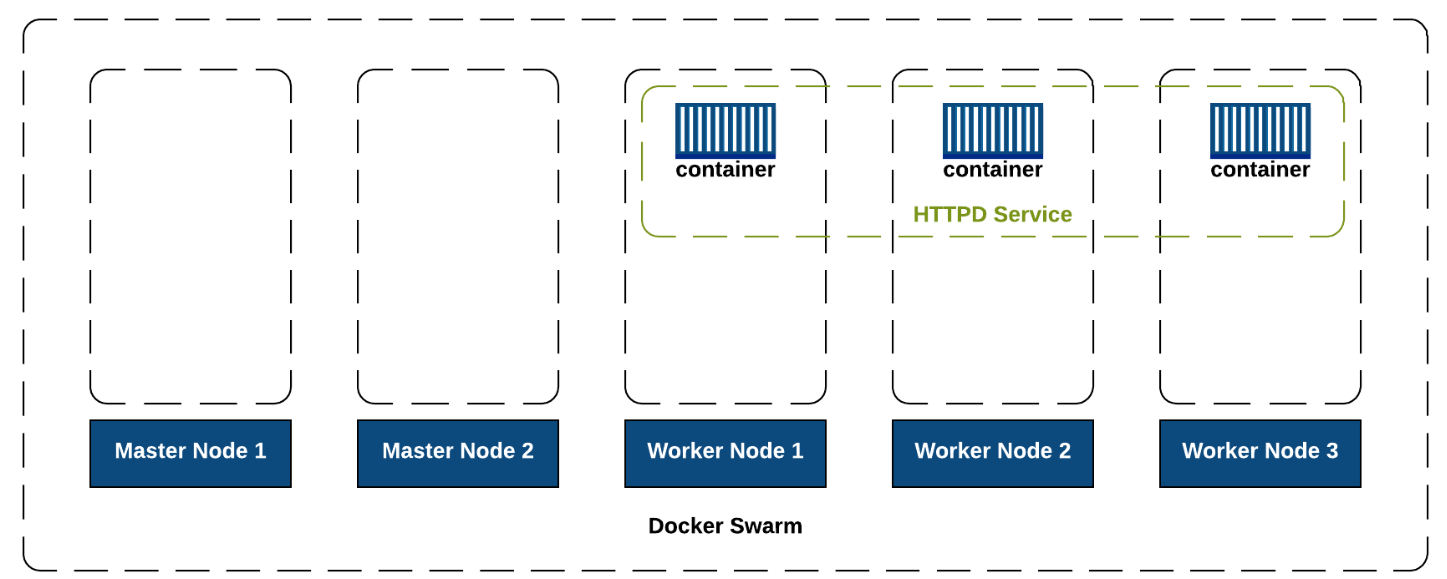
1. Scale the replicas down to 1:

docker service scale backup=1

1. Next, scale the replicas up to 3 to distribute the replicas across the swarm:

docker service scale backup=3

Scaling a Docker Swarm Service

 **Create a Swarm**keyboard\_arrow\_up

Create a swarm with 2 masters and 3 worker nodes.

Initilize the swarm.

docker swarm init

Use the join command to add the 3 worker nodes.

docker swarm join --token TOKEN IP\_ADDRESS:2377

Generate the master token.

docker swarm join-token manager

che**Drain the Masters**keyboard\_arrow\_up

Set the availability to drain for Master1.

docker node update --availability drain MASTER1

Set the availability to drain for Master2.

docker node update --availability drain MASTER2

che**Create a Service**keyboard\_arrow\_up

Create a service with 3 replicas.

docker service create --name httpd -p 80:80 --replicas 3 httpd

che**Scale the Service up to 5 Replicas**keyboard\_arrow\_up

Scale the httpd service up to 5 replicas.

docker service scale httpd=5

che**Scale the Service Down to 2 Replicas**keyboard\_arrow\_up

Scale the httpd service down to 2 replicas.

docker service scale httpd=2

**Scaling a Docker Swarm Service**

Hi there. Welcome to the lab. This one looks to be fairly simple. We've got to create a couple of Docker swarm master nodes, and three worker nodes, then practice scaling replicas up and down.

There are five servers we'll need to log into: two will be swarm masters, and three will be swarm workers. Keep the Linux Academy lab page handy. We'll need it for login credentials, and to help keep track of which server is doing what. The command prompts in this lab guide are just general "user@host" types, but knowing which host has which IP will be helpful.

If we fire up five terminals, and log into each of the servers, we'll be able to switch back and forth quickly. And since we'll need to be root in all of them, we can just run a sudo su - as soon as we log into each one. Once we're in, we can get moving with Docker.

Create a Swarm

On our first server, we're going to create a swarm master. Initialize the swarm with this:

[root@swarmmaster1]# docker swarm init

There's going to be some output, including a docker swarm join command that we can copy.

Add Worker Nodes

On our worker node servers, we can join the swarm by pasting in the command that our swarm master spit out. Add the 3 worker nodes with these:

Worker 1

[root@swarmworker1]# docker swarm join --token <TOKEN> <IP\_ADDRESS:2377>

Worker 2

[root@swarmworker2]# docker swarm join --token <TOKEN> <IP\_ADDRESS:2377>

Worker 3

[root@swarmworker3]# docker swarm join --token <TOKEN> <IP\_ADDRESS:2377>

The Master Token

Back on swarmmaster1, we need to generate the master token:

[root@swarmmaster1]# docker swarm join-token manager

This will output yet another command. The first one we copied and pasted was for getting a worker node joined. This one is for when we want a master node to join. Let's log into the second master, and paste that command:

[root@swarmmaster2]# docker swarm join --token <TOKEN> <IP\_ADDRESS:2377>

Check Our Work

A quick docker node ls from our first master server will show us that there are five nodes running. Three are workers, and two have a *MANAGER STATUS*: *Leader* (the first master) and *Reachable* (the second master).

Ensure that the Two Masters Will Only Function as Masters

We want the masters to just be masters, not workers too. To do that we run these (one for each master):

Set the availability to drain for Master1.

[root@swarmmaster1]# docker node update --availability drain <MASTER1 ID>

Set the availability to drain for Master2.

[root@swarmmaster1]# docker node update --availability drain <MASTER2 ID>

<MASTER ID> here would be the *ID* column in that docker node ls command we ran earlier.

Create a Service

We're going to create a service, and replicate it three times.

[root@swarmmaster1]# docker service create --name httpd -p 80:80 --replicas 3 httpd

In this command, we created the service, named it httpd, ran it on port 80, created three replicas, and used an image called httpd that was pulled from Docker Hub.

Check What Is Running

We can look at what actually fired up with another docker command:

[root@swarmmaster1]# docker service ps httpd

The output shows that we have three replicas running, one on each of our worker nodes, and that they're *only* running on our worker nodes. Great. Now let's scale up.

Scale the httpd Service up to Five Nodes

Instead of just running three replicas, let's scale the httpd service up to five:

[root@swarmmaster1]# docker service scale httpd=5

If we run another docker service ps httpd, we'll see there are now five replicas running. If we look at the nodes, we'll also see that it's still only the worker nodes running these. This is what we intended though: we want our master being masters, and just want the workers working.

Let's scale down and see what happens.

Scale the httpd Service Back Down to Two Nodes

We're going to cut our workforce by three, so that we're only running two replicas:

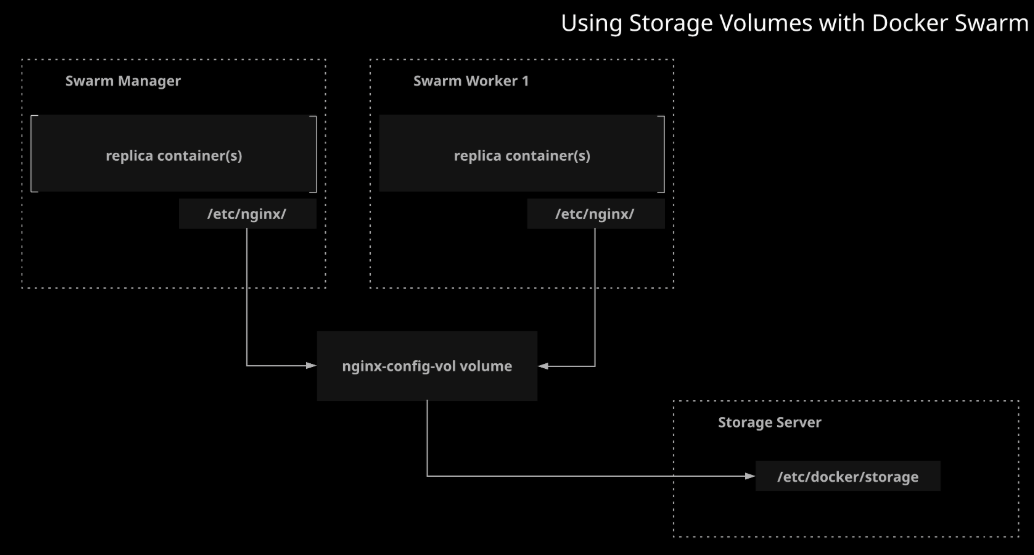
[root@swarmmaster1]# docker service scale httpd=2

Now we can run docker service ps httpd again, and we'll see that three of the replicas have disappeared, leaving us with two.

Conclusion

Well, we've done it. We created a Docker swarm, containing two masters and three workers, and figured out how to scale the number of replicas up and down.

# Using Storage Volumes with Docker Swarm



Storage volumes provide a powerful and flexible way to add persistent storage to your containers, but what if you need to share storage volumes across multiple Docker hosts, such as a Swarm cluster? In this lab, you will have the opportunity to work with a simple method of creating shared volumes usable across multiple swarm nodes using the sshfs volume driver.

## **Solution**

1. Begin by logging in to the storage server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Set up the External Storage Location**

1. On the storage server, create the storage directory.
2. sudo mkdir -p /etc/docker/storage

sudo chown cloud\_user:cloud\_user /etc/docker/storage

1. Copy the nginx configuration file into the storage directory.

cp /home/cloud\_user/nginx.conf /etc/docker/storage/

### **Install the vieux/sshfs Plugin**

1. Install the vieux/sshfs plugin on the **swarm manager** and **worker node**.

docker plugin install --grant-all-permissions vieux/sshfs

### **Create the nginx Service That Uses the Shared Volume**

1. Create the nginx-web service on the **swarm manager**.

Create the container. Be sure to replace <cloud\_user password> with the actual password.

docker service create -d \

--replicas=3 \

--name nginx-web \

-p 8080:9773 \

--mount volume-driver=vieux/sshfs,source=nginx-config-vol,target=/etc/nginx/,volume-opt=sshcmd=cloud\_user@10.0.1.103:/etc/docker/storage,volume-opt=password=<cloud\_user password> nginx:latest

1. Verify that the service is working properly.

curl localhost:8080

If everything is set up correctly, you should see HTML from the nginx Welcome page.

# Load Balancing Containers

# 

## **Introduction**

For the last six months, the Acme Anvil Corporation has been migrating some of their bare metal infrastructure to Docker containers. The initial implementation was very basic and lacked any kind of load balancing. Your manager has tasked you with creating two proofs of concept. For the first proof of concept, you are to use Docker Compose to create an Nginx load balancer and three instances using your weather-app image. Nginx will use port 80 and send traffic to port 3000 on the weather-app containers. For the second proof of concept, you are to create a Docker Swarm service called nginx-app that has two replicas using the Nginx image. The service should be published to port 8080 on the host and target port 80 on the containers.

## **Solution**

Begin by logging in to both lab servers using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

Become the root user:

sudo su -

### **Create a Docker Compose file**

On **Swarm Server 1**

1. Change to the lb-challenge directory:

cd lb-challenge

1. Create our Docker compose file:

vi docker-compose.yml

1. The contents of your docker-compose.yml file should look like the following:
2. version: '3.2'
3. services:
4. weather-app1:
5. build: ./weather-app
6. tty: true
7. networks:
8. - frontend
9. weather-app2:
10. build: ./weather-app
11. tty: true
12. networks:
13. - frontend
14. weather-app3:
15. build: ./weather-app
16. tty: true
17. networks:
18. - frontend
19. loadbalancer:
20. build: ./load-balancer
21. image: nginx
22. tty: true
23. ports:
24. - '80:80'
25. networks:
26. - frontend
27. networks:

frontend:

### **Update nginx.conf**

1. Change to the load-balancer directory:

cd load-balancer

1. Edit the nginx.conf file:

vi nginx.conf

1. The contents of your nginx.conf file should look like the following:
2. events { worker\_connections 1024; }
3. http {
4. upstream localhost {
5. server weather-app1:3000;
6. server weather-app2:3000;
7. server weather-app3:3000;
8. }
9. server {
10. listen 80;
11. server\_name localhost;
12. location / {
13. proxy\_pass http://localhost;
14. proxy\_set\_header Host $host;
15. }
16. }

}

### **Execute docker-compose up**

1. Change directories:

cd ../

1. Execute a docker-compose up:

docker-compose up --build -d

1. Check our work:

docker ps

1. Copy the public IP address of **Swarm Server 1** and paste it into a new tab in your browser.

### **Create a Docker service using Docker Swarm**

1. Change to the root directory:

cd ~/

1. View the contents of swarm-token.txt:

cat swarm-token.txt

1. Copy the docker swarm join command from the previous step.

On **Swarm Server 2**

1. Execute the command that was copied from the previous step.

On **Swarm Server 1**

1. Create a Docker service by executing the following command:

docker service create --name nginx-app --publish published=8080,target=80 --replicas=2 nginx

1. Verify this completed successfully by running the following command on both servers:

docker ps

1. Verify that the default nginx page loads in the browser:

PUBLIC\_IP\_ADDRESS:8080

# Working with Docker Content Trust

In this lab, we will work with Docker Content Trust (DCT) by signing a previously unsigned image and running it on a system that has DCT enabled.

## **Solution**

Log in to the lab server using the credentials provided on the hands-on lab page:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

### **Generate a Trust Key and Add Yourself as a Signer to the New Repository**

1. Generate a trust key:

docker trust key generate docker

1. Create a new passphrase for your key when prompted. Note the passphrase for new docker key, since we will use it later on in this lab.
2. Add yourself as a signer to the ip-10-0-1-102:443/content-dca-tea repository:

docker trust signer add --key docker.pub docker ip-10-0-1-102:443/content-dca-tea

1. Create passphrases for the new root key and new repository key when prompted.

### **Create a New Tag for the Image, Sign It, and Push It to the Registry**

1. Create a new tag for the image:

docker tag linuxacademycontent/content-dca-tea:1 ip-10-0-1-102:443/content-dca-tea:1

1. Sign the image and push it to the registry:

docker trust sign ip-10-0-1-102:443/content-dca-tea:1

1. Enter your passphrase when prompted. This is the passphrase we created earlier by establishing your trust key using the docker trust key generate docker command.
2. Verify that you can run the signed image:

docker run -d -p 8080:80 ip-10-0-1-102:443/content-dca-tea:1

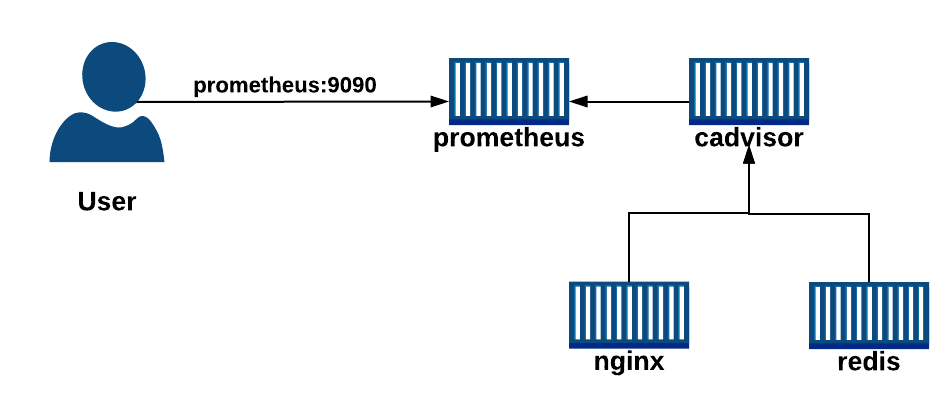
If you want to test the image further, you can query the tea list web service:

curl localhost:8080

You should see generated JSON data that contains a list of the various kinds of tea.

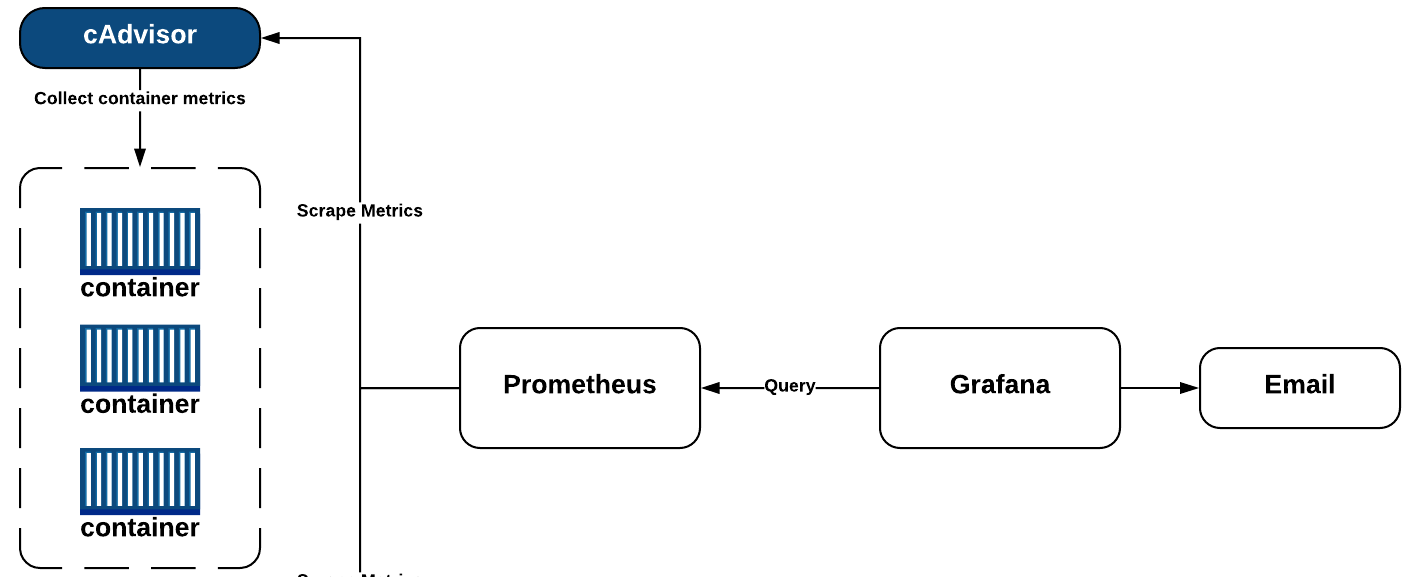
Monitoring Containers with Prometheus

|  |
| --- |
| **Create a `prometheus.yml` File**keyboard\_arrow\_up  In the root directory, create prometheus.yml.  scrape\_configs:  - job\_name: cadvisor  scrape\_interval: 5s  static\_configs:  - targets:  - cadvisor:8080  che**Create the Prometheus Services**keyboard\_arrow\_up  Create a docker-compose.yml file:  version: '3'  services:  prometheus:  image: prom/prometheus:latest  container\_name: prometheus  ports:  - 9090:9090  command:  - --config.file=/etc/prometheus/prometheus.yml  volumes:  - ./prometheus.yml:/etc/prometheus/prometheus.yml  depends\_on:  - cadvisor  cadvisor:  image: google/cadvisor:latest  container\_name: cadvisor  ports:  - 8080:8080  volumes:  - /:/rootfs:ro  - /var/run:/var/run:rw  - /sys:/sys:ro  - /var/lib/docker:/var/lib/docker:ro  Stand up the environment.  docker-compose up -d  che**Create `stats.sh`**keyboard\_arrow\_up  Create stats.sh in /root.  docker stats --format "table {{.Name}} {{.ID}} {{.MemUsage}} {{.CPUPerc}}"  Make sure the file can be executed.  chmod a+x stats.sh  Execute the script.  ./stats.sh  When you're finished, exit by pressing **Ctrl+C**. |



Using Grafana with Prometheus for Alerting and Monitoring

|  |
| --- |
| **Configure Docker.**  **Update `prometheus.yml`.**  **Update `docker-compose.yml`.**  **Install the Docker and System Monitoring dashboard.** |



In this learning activity, we will learn how to set up Prometheus and Grafana for alerting and monitoring.

Accessing the Live Environment

On the lab instructions page, copy the public IP address of the cloud server to your clipboard. Open your terminal application, and run the following command:

ssh cloud\_user@PUBLIC\_IP\_ADDRESS

At the prompt, enter yes to confirm that we wish to continue connecting.

Next, copy the password from the lab instructions page and enter it at the password prompt in your terminal. Then elevate privileges to root.

sudo su -

Enter the password again when prompted. We have now successfully logged in to the environment.

Using Grafana with Prometheus

Before we get started, let's take a look at our root directory to see what we have.

ls

Currently, we have a dashboard, the docker-compose file that was used to set up the environment, and a Prometheus configuration file. Run clear to clear your screen.

Configure Docker

The first thing we need to do is create a daemon.json file for Docker.

vi /etc/docker/daemon.json

Once /etc/docker/daemon.json is open in the vi text editor, add the following:

{

"metrics-addr" : "0.0.0.0:9323",

"experimental" : true

}

Then, save and exit the vi editor.

:wq

Restart the Docker service.

systemctl restart docker

Run clear to clear your screen.

Update Prometheus

Next, we're going to edit the prometheus.yml file in the /root directory.

vi prometheus.yml

Change the contents of the file to the following: (Be sure to provide the private IP address of your instance)

scrape\_configs:

- job\_name: prometheus

scrape\_interval: 5s

static\_configs:

- targets:

- prometheus:9090

- node-exporter:9100

- pushgateway:9091

- cadvisor:8080

- job\_name: docker

scrape\_interval: 5s

static\_configs:

- targets:

- <PRIVATE\_IP\_ADDRESS>:9323

Then, save and quit.

:wq

Update Docker Compose

The next step is to open our docker-compose file and add three new services. Open docker-compose.yml.

vi ~/docker-compose.yml

Change the contents of the file to the following:

version: '3'

services:

prometheus:

image: prom/prometheus:latest

container\_name: prometheus

ports:

- 9090:9090

command:

- --config.file=/etc/prometheus/prometheus.yml

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml:ro

depends\_on:

- cadvisor

cadvisor:

image: google/cadvisor:latest

container\_name: cadvisor

ports:

- 8080:8080

volumes:

- /:/rootfs:ro

- /var/run:/var/run:rw

- /sys:/sys:ro

- /var/lib/docker/:/var/lib/docker:ro

pushgateway:

image: prom/pushgateway

container\_name: pushgateway

ports:

- 9091:9091

node-exporter:

image: prom/node-exporter:latest

container\_name: node-exporter

restart: unless-stopped

expose:

- 9100

grafana:

image: grafana/grafana

container\_name: grafana

ports:

- 3000:3000

environment:

- GF\_SECURITY\_ADMIN\_PASSWORD=password

depends\_on:

- prometheus

- cadvisor

Then, save and quit.

:wq

Next, let's apply our changes and rebuild the environment.

docker-compose up -d

Then, let's make sure everything is running.

docker ps

Lastly, let's verify that all of our Prometheus targets are healthy. Open a web browser tab, and go to http://PUBLIC\_IP\_ADDRESS:9090. Click **Status**, and select **Targets**. Everything should be up and running correctly.

Create a New Grafana Data Source

Navigate to Grafana by entering http://PUBLIC\_IP\_ADDRESS:3000 in the address bar of your browser.

Type "admin" for *username* and "password" for *password*. Click **Log In**.

In the Grafana Home Dashboard, click the **Add data source** icon. For *Name*, type "Prometheus". Click into the *Type* field, and select **Prometheus** from the dropdown. Under *URL*, select [**http://localhost:9090**](http://localhost:9090/). (But we're going to change this in a moment.) Navigate to the lab instructions page, and copy the private IP address to your clipboard. Then, replace "localhost" in the URL with the private IP address. (It should look like this: http://PRIVATE\_IP\_ADDRESS:9090).

Click **Save & Test**.

Add the Docker Dashboard to Grafana

Click the plus sign (+) on the left side of the Grafana interface, and click **Import**. Then, navigate to the lab instructions page. Open the [JSON file](https://raw.githubusercontent.com/linuxacademy/content-intermediate-docker-quest-prometheus/master/dashboards/docker_and_system.json) included in the *Instructions & Tasks* section in a new browser tab. Copy the contents of the file to your clipboard.

Switch back to Grafana, and paste the text we just copied into the *Or paste JSON* field. Click **Load**. On the Import screen, click the dropdown menu for *Prometheus*, and select the **Prometheus** data source that we created earlier. Click **Import**.

We now have our Grafana visualization. In the upper right corner, click on **Refresh every 5m** and select **Last 5 minutes**.

Add an Email Notification Channel

In the left sidebar, click the bell icon, and select **Notification channels**. Click **Add channel**. For *Name*, type "Email". In *Email addresses*, enter your email address. Click **Save**.

Create an Alert for CPU Usage

Click the dashboard icon in the left sidebar, and select **Home**. Select the **Docker and system monitoring** dashboard. Click the **CPU Usage** graph, and select **Edit** from the dropdown.

In the *Metrics* tab, click **Add Query**. Enter the following:

sum(rate(process\_cpu\_seconds\_total[1m])) \* 100

Click the eye icon on the right side of Row E to hide the new rule from view.

Click the **Alert** tab, then **Create Alert**. Under *Conditions*, click into the query field and select **E\*\* from the dropdown. For *IS ABOVE*, type "75". Click \*\*Test Rule**.

Click **Notifications** in the left sidebar, then click the + icon next to *Sent to*. Select **Email Alerts**. Next, click into the *Message*text box, and type "CPU usage is over 75%." Click the save button at the top of the page. In the save popup that opens, check the box next to *Save current variables*. In the description box, type "Add alerting". Then click **Save**.

Conclusion

Congratulations, you've successfully completed this lab!