JTC01 Docker

Lab0 - Lab information

Docker is an open platform for developing, shipping, and running applications.

Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

In this Lab you will learn the basic operations for building and running Docker containers.

Lab sources

All the source code for the lab is available here:

https://github.com/niklaushirt/training

Lab overview

- In this Lab you will learn the basics of Docker:

1. Docker basics
2. Build a Docker image
3. Run a Docker image
4. Use the Portainer tool
5. Deploy a more complex Docker application
6. Push a Docker image into a registry
7. Docker internals

Lab0 - Lab semantics

Nomenclatures

Shell Commands

The commands that you are going to execute to progress the Labs will look like this:

THIS IS AN EXAMPLE - DO NOT EXECUTE THIS!

Bash

```bash
kubectl create -f redis-slave-service.yaml

 > Output Line 1
 > Output Line 2
 > Output Line 3
 ...
```

> **IMPORTANT NOTE:** The example output of a command is prefixed by ">" in order to make it more distinguishable.
>
> So in the above example you would only enter/copy-paste `kubectl create -f redis-slave-service.yaml` and the output from the command is "Output Line 1" to "Output Line 3"

Code Examples

Code examples are presented like this:

YAML

```yaml
apiVersion: lab.ibm.com/v1beta1
kind: MyResource
metadata:
  name: example
spec:
  size: 3
  image: busybox
```

This is only for illustration and is not being actively used in the Labs.

Lab 0 - Prepare the Lab environment

Before starting the Labs, let's make sure that we have the latest source code from the GitHub repository:

https://github.com/niklaushirt/training

1. Open a Terminal window by clicking on the Termnial icon in the bottom dock - we will use this extensively later as well

2. Execute the following commands to initialize your Training Environment

   Bash
   ```bash
   ./welcome.sh
   ```

   This will

   - pull the latest example code from my GitHub repository
   - start minikube if not already running
   - installs the registry
   - installs the Network Plugin (Cilium)
   - starts the Personal Training Environment

   > During this you will have to provide a name (your name) that will be used to show your progress in the Instructor Dashboard in order to better assist you.

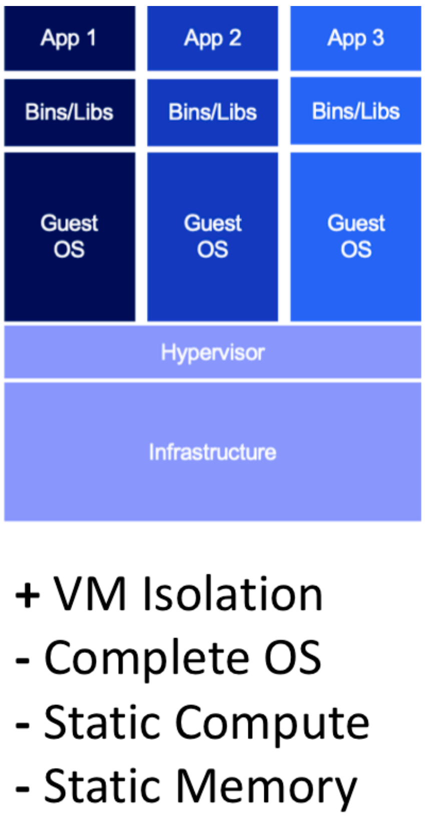
Lab 1 - Get to know Docker

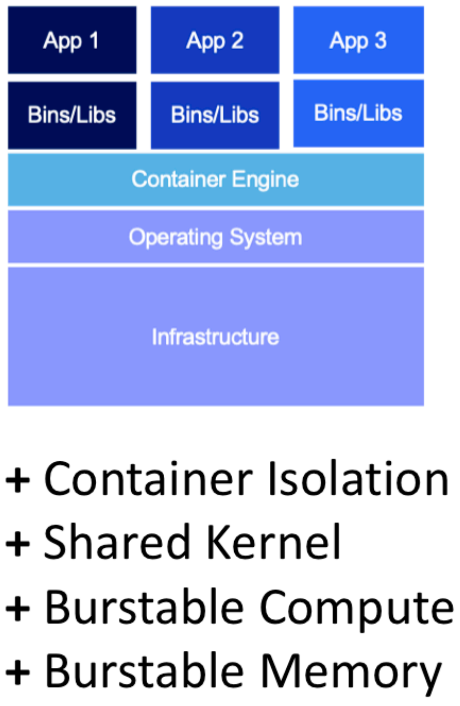
Docker is an open platform for developing, shipping, and running applications.

Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

More details on `Docker` can be found here.





Dockerfile

A `Dockerfile` is a text document that contains all the commands a user could call on the command line to assemble an image.

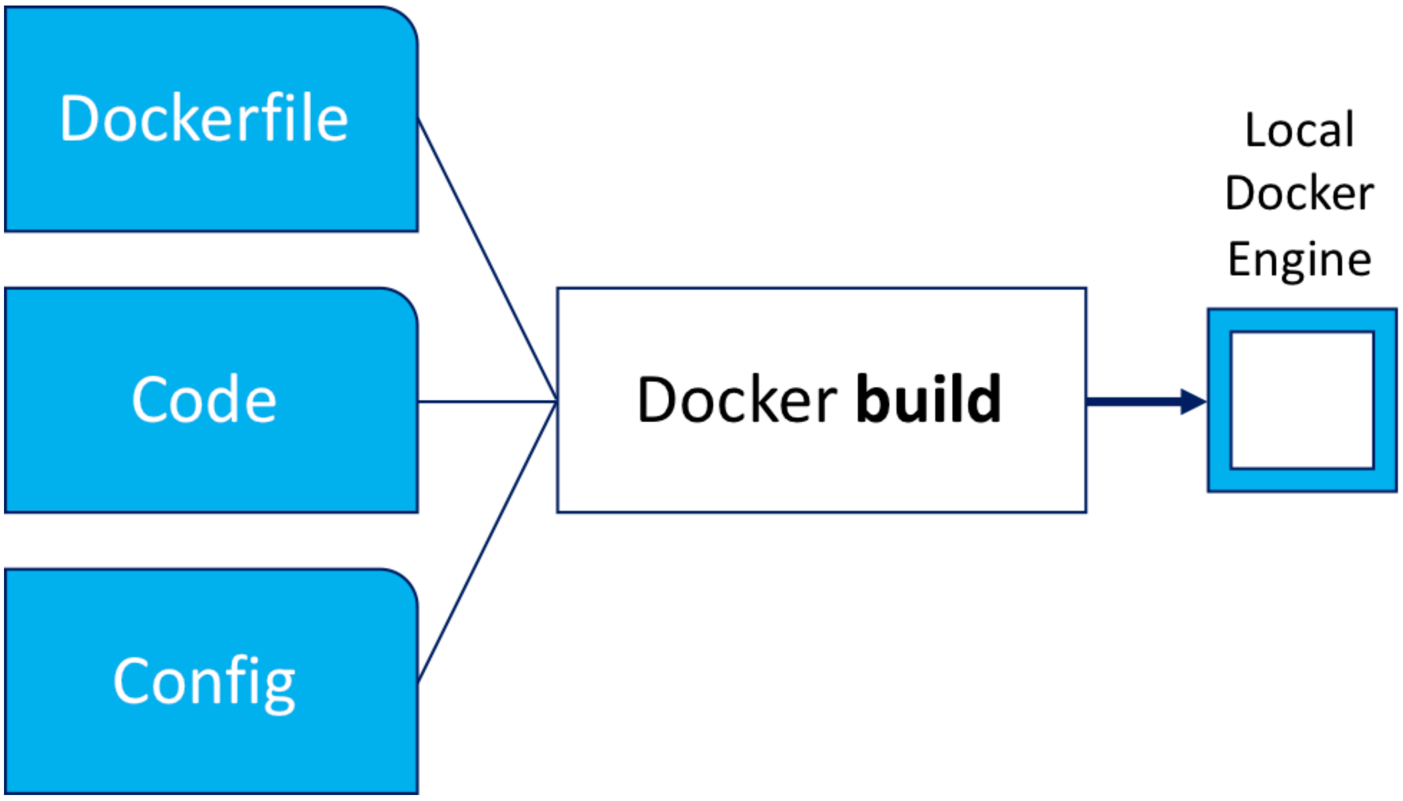
```yaml
FROM node:8-stretch

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
 && apt-get dist-upgrade -y \
 && apt-get clean \
 && echo 'Finished installing dependencies'

# Install npm production packages
COPY package.json /app/
RUN cd /app; npm install --production

COPY . /app

ENV NODE_ENV production
ENV BACKEND_URL https://api.nasa.gov/planetary/apod\?api_key\=DEMO_KEY
ENV PORT 3000

EXPOSE 3000

CMD ["npm", "start"]
```

The `docker build` command creates the container from the `Dockerfile`.

More details on Dockerfiles can be found here.

Lab 2 - Docker

Lab 2 - Create your first Image

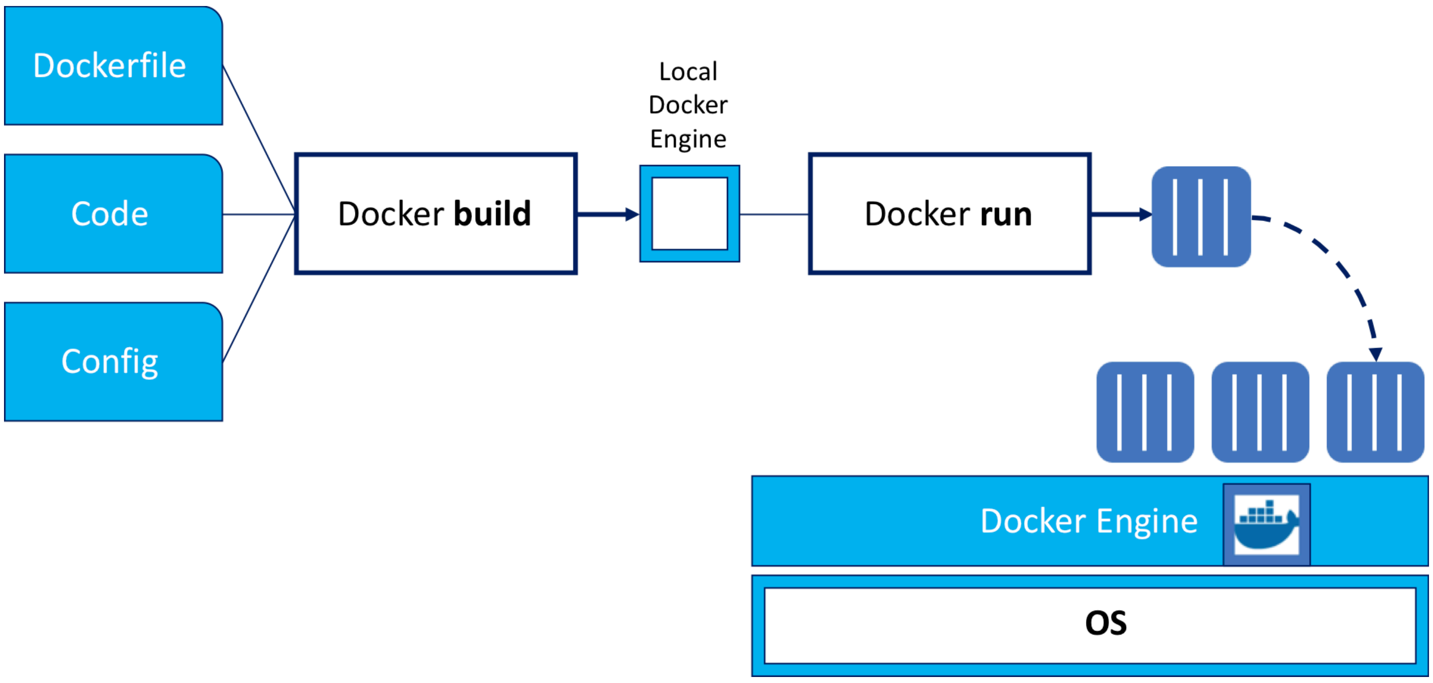


Let's create our first image (the `k8sdemo-backend` image) from this `Dockerfile`:

YAML
```
FROM node:8-stretch

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
 && apt-get dist-upgrade -y \
 && apt-get clean \
 && echo 'Finished installing dependencies'

# Install npm production packages
COPY package.json /app/
RUN cd /app; npm install --production

COPY . /app

ENV NODE_ENV production
ENV BACKEND_MESSAGE HelloWorld

ENV PORT 3000

EXPOSE 3000

CMD ["npm", "start"]
```

Bash
```
cd ~/training/demo-app/k8sdemo_backend

docker build -t k8sdemo-backend:lab .

> Sending build context to Docker daemon  6.975MB
> Step 1/11 : FROM node:8-stretch
>  ---> 7a9afc16a57f
> Step 2/11 : WORKDIR "/app"
>  ---> Using cache
>  ---> a2515f8a3ec5
...
> Step 11/11 : CMD ["npm", "start"]
>  ---> Using cache
>  ---> b9b0f3fea9f7
> Successfully built b9b0f3fea9f7
> Successfully tagged k8sdemo-backend:lab
```

Lab 2 - Run your first Image



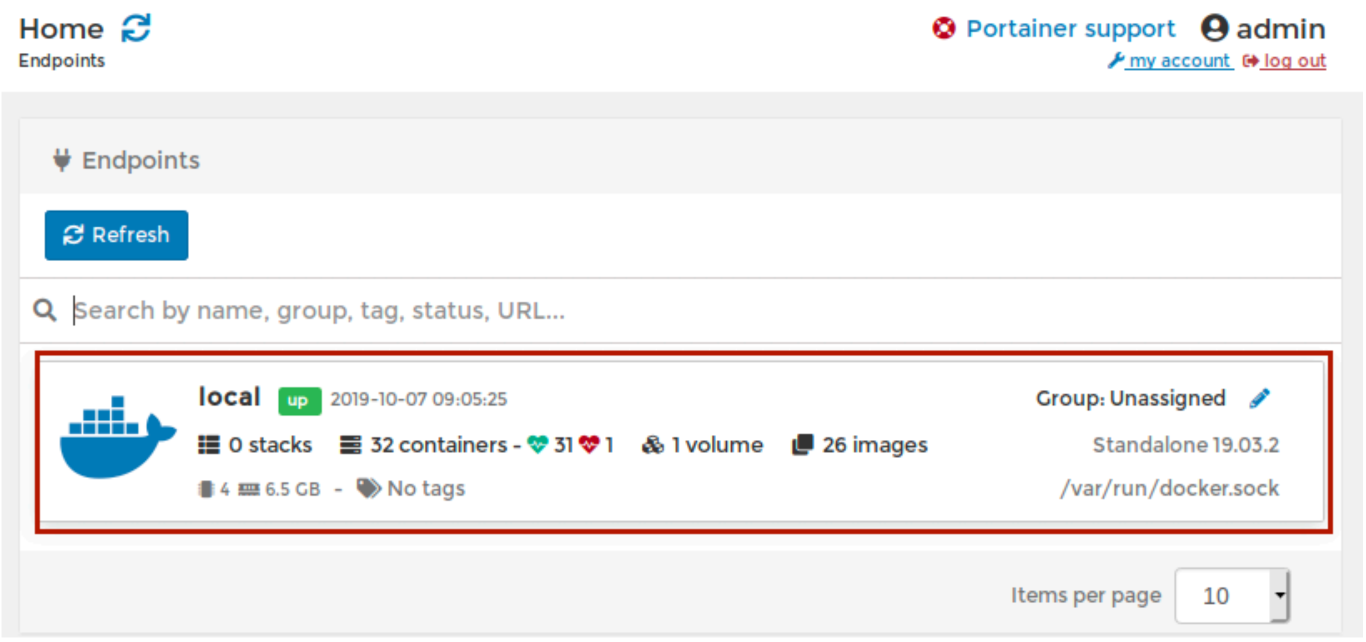
```bash
docker run --rm --name k8sdemo-backend -p 3001:3000 k8sdemo-backend:lab

> test@0.0.0 start /app
> node ./bin/www
```

This command runs the backend server: * --rm makes sure that the container is deleted once it's stopped * --name gives the container a fixed name (otherwise you get some pretty funny, automatically generated names - think drunken-weasel) * -p exposes the container port 3000 to the outside port 3001 (we do this so that it does not conflict with port 3000 of the k8sdemo web application we will start later) * k8sdemo-backend:lab is the image we created before

Lab 2 - Use Portainer

Portainer Community Edition is a powerful, open-source management toolset that allows you to easily build, manage and maintain Docker environments.

1. Open URL or use the Portainer Bookmark
2. Login in with `admin` / `passw0rd` (already prefilled)
3. Select `local` for our Endpoint



Home
Endpoints

Portainer support admin
my account log out

Endpoints

Refresh

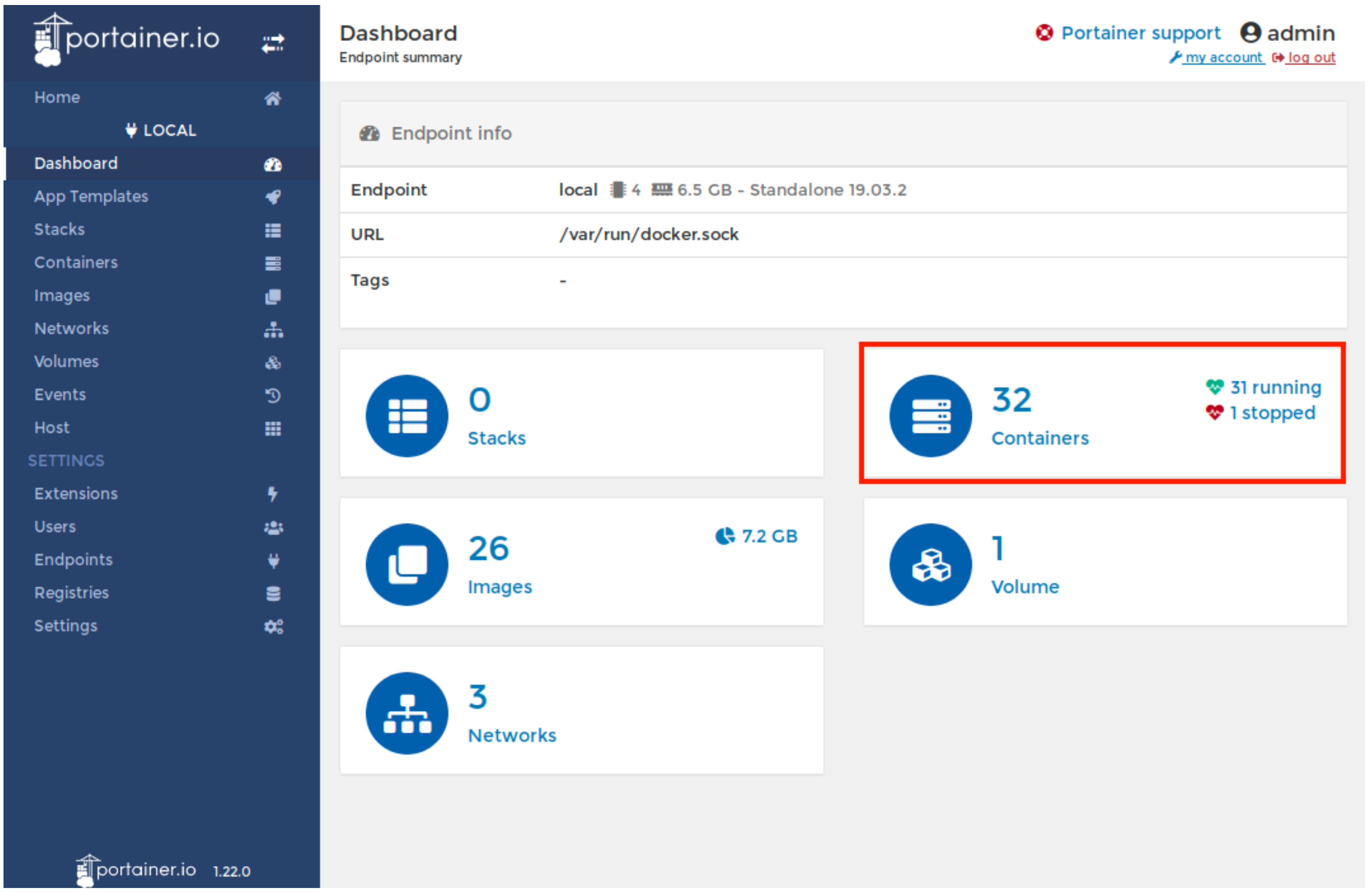
Search by name, group, tag, status, URL...

local up 2019-10-07 09:05:25 Group: Unassigned
0 stacks 32 containers - 31 1 1 volume 26 images Standalone 19.03.2
4 6.5 GB - No tags /var/run/docker.sock

Items per page 10

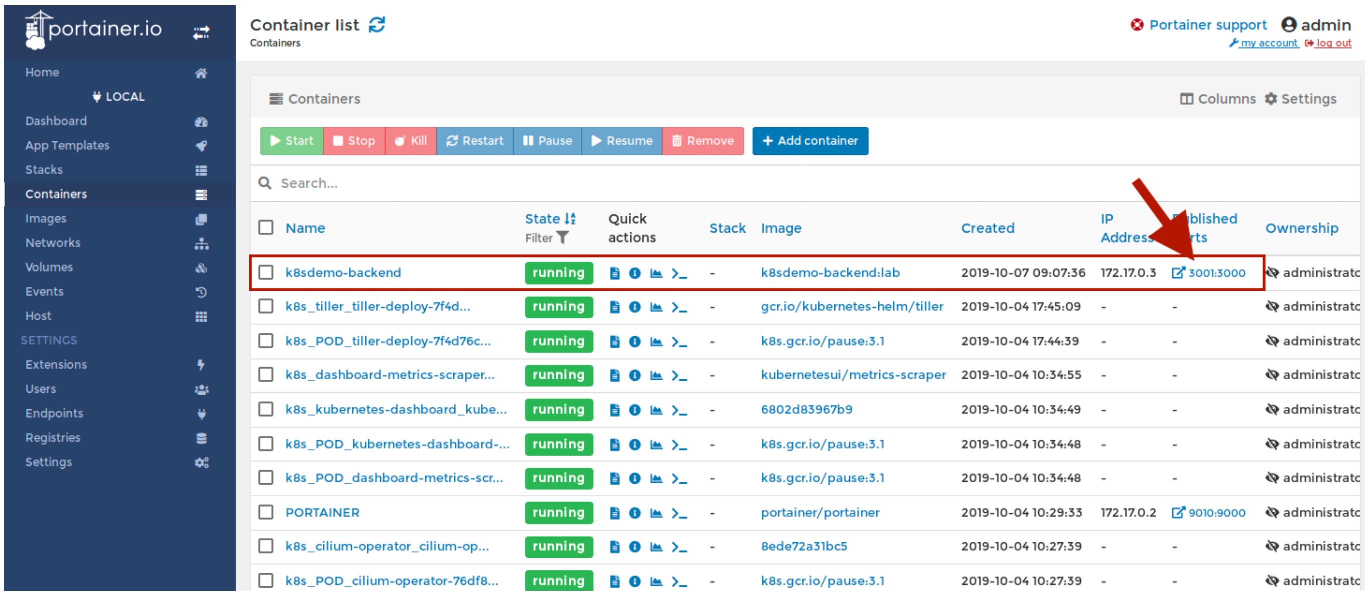
Now you get an overview of your local Docker instance.

4. Select `Containers`



Endpoint	local 4 6.5 GB - Standalone 19.03.2
URL	/var/run/docker.sock
Tags	-

5. You get a list of all running containers and you can see our `k8sdemo-backend` container running.



	Name	State Filter	Quick actions	Stack	Image	Created	IP Addres	blished rts	Ownership
☐	k8sdemo-backend	running	📄 ℹ 📈 >_	-	k8sdemo-backend:lab	2019-10-07 09:07:36	172.17.0.3	3001:3000	administratc
☐	k8s_tiller_tiller-deploy-7f4d...	running	📄 ℹ 📈 >_	-	gcr.io/kubernetes-helm/tiller	2019-10-04 17:45:09	-	-	administratc
☐	k8s_POD_tiller-deploy-7f4d76c...	running	📄 ℹ 📈 >_	-	k8s.gcr.io/pause:3.1	2019-10-04 17:44:39	-	-	administratc
☐	k8s_dashboard-metrics-scraper...	running	📄 ℹ 📈 >_	-	kubernetesui/metrics-scraper	2019-10-04 10:34:55	-	-	administratc
☐	k8s_kubernetes-dashboard_kube...	running	📄 ℹ 📈 >_	-	6802d83967b9	2019-10-04 10:34:49	-	-	administratc
☐	k8s_POD_kubernetes-dashboard-...	running	📄 ℹ 📈 >_	-	k8s.gcr.io/pause:3.1	2019-10-04 10:34:48	-	-	administratc
☐	k8s_POD_dashboard-metrics-scr...	running	📄 ℹ 📈 >_	-	k8s.gcr.io/pause:3.1	2019-10-04 10:34:48	-	-	administratc
☐	PORTAINER	running	📄 ℹ 📈 >_	-	portainer/portainer	2019-10-04 10:29:33	172.17.0.2	9010:9000	administratc
☐	k8s_cilium-operator_cilium-op...	running	📄 ℹ 📈 >_	-	8ede72a31bc5	2019-10-04 10:27:39	-	-	administratc
☐	k8s_POD_cilium-operator-76df8...	running	📄 ℹ 📈 >_	-	k8s.gcr.io/pause:3.1	2019-10-04 10:27:39	-	-	administratc

6. Click on the PublishedPorts 3001:3000 to open the backend web interface.

7. In your terminal you will see that this generated some traffic:

```bash
docker run --rm --name k8sdemo-backend -p 3001:3000 k8sdemo-backend:lab

> test@0.0.0 start /app
> node ./bin/www

> GET / 304 225.805 ms - -
> GET /stylesheets/style.css 304 2.175 ms - -
```

8. Stop the container by hitting `CTRL-C` in the terminal (several times if needed)

So now we have tested our backend component.

Lab 2 - Create the frontend Image

Let's create our first image (the `k8sdemo` frontend image) from this `Dockerfile`:

YAML

```
FROM node:8-stretch

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
 && apt-get dist-upgrade -y \
 && apt-get clean \
 && echo 'Finished installing dependencies'

# Install npm production packages
COPY package.json /app/
RUN cd /app; npm install --production

COPY . /app

ENV NODE_ENV production
ENV BACKEND_URL https://api.nasa.gov/planetary/apod\?api_key\=DEMO_KEY
ENV PORT 3000

EXPOSE 3000

CMD ["npm", "start"]
```

Bash

```bash
cd ~/training/demo-app/k8sdemo

docker build -t k8sdemo:lab .

> Sending build context to Docker daemon  13.23MB
> Step 1/11 : FROM node:8-stretch
>  ---> 7a9afc16a57f
> Step 2/11 : WORKDIR "/app"
>  ---> Using cache
>  ---> a2515f8a3ec5
...
> Step 11/11 : CMD ["npm", "start"]
>  ---> Using cache
>  ---> 5293cb32d1f6
> Successfully built 5293cb32d1f6
> Successfully tagged k8sdemo:lab
```

Lab 2 - Run the frontend Image

1. First let's run the backend container again, but this time in the background

   ```bash
   docker run --rm -d --name k8sdemo-backend -p 3001:3000 k8sdemo-backend:lab

   > 444b0570058b97f0532ef89c92963bb7da6aa1f2d3e27bf8c989da5fb8277fe0
   ```

   This command runs the backend server:

   - -d runs the container in the background (as a daemon)

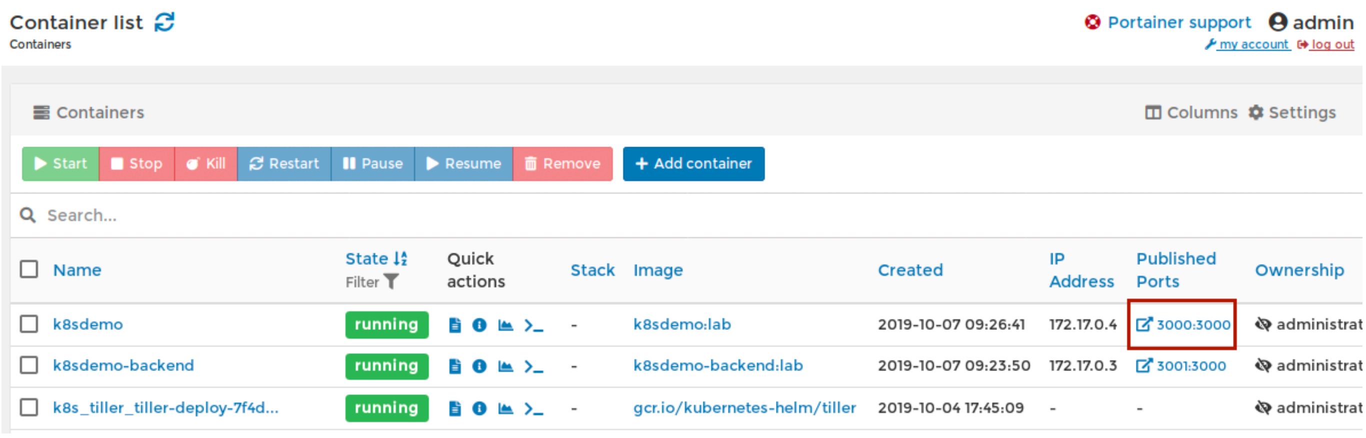
2. Then we start the new Web Frontend container

   ```bash
   docker run --rm --name k8sdemo -p 3000:3000 --env BACKEND_URL=http://172.17.0.
   1:3001/api k8sdemo:lab

   > test@0.0.0 start /app
   > node ./bin/www
   ```

   This command runs the frontend server:

   - --rm makes sure that the container is deleted once it's stopped
   - --name gives the container a fixed name
   - --env defines the environment variable that points to the `k8sdemo-backend` server API
   - -p exposes the container port 3000 to the outside port 3000
   - k8sdemo:lab is the image we created before

3. Go back to Portainer and refresh the browser



Name	State Filter	Quick actions	Stack	Image	Created	IP Address	Published Ports	Ownership
k8sdemo	running		-	k8sdemo:lab	2019-10-07 09:26:41	172.17.0.4	3000:3000	administrat
k8sdemo-backend	running		-	k8sdemo-backend:lab	2019-10-07 09:23:50	172.17.0.3	3001:3000	administrat
k8s_tiller_tiller-deploy-7f4d...	running		-	gcr.io/kubernetes-helm/tiller	2019-10-04 17:45:09	-	-	administrat

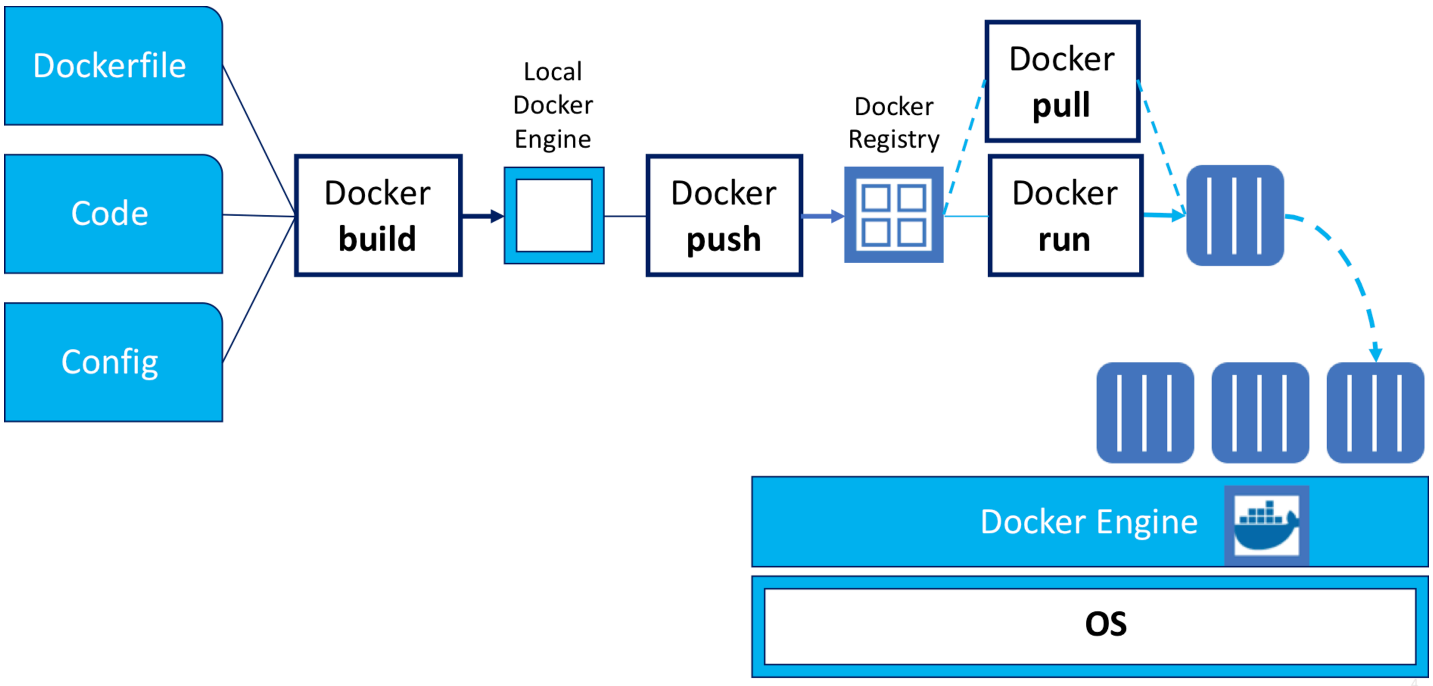
4. Click on the PublishedPorts 3000:3000 for `k8sdemo` to open the web interface.

5. Refresh several times and check in the terminal that some traffic is being generated

```
GET / 304 6.314 ms - -
GET /public/bootstrap.min.css 304 0.975 ms - -
GET /public/bootstrap-theme.min.css 304 0.843 ms - -
GET /public/stylesheets/style.css 304 2.568 ms - -
GET /public/images/ibm_cloud.png 304 0.522 ms - -
GET /public/images/cloud_private.png 304 1.057 ms - -
GET /public/images/back.png 304 0.411 ms - -
The value of BACKEND_URL is: http://k8sdemo-backend
Error: getaddrinfo ENOTFOUND k8sdemo-backend k8sdemo-backend:80
```

6. Stop the container by hitting `CTRL-C` in the terminal

Lab 2 - Push the frontend Image to the registry



1. Let's tag the image with the address of the local Docker registry (localhost:5000).

   ```bash
   docker tag k8sdemo:lab localhost:5000/k8sdemo:lab
   ```

2. Expose the local Docker registry.

   First execute this in order to be able to access the private registry:

   ```bash
   kubectl port-forward --namespace kube-system $(kubectl get po -n kube-system |
    grep kube-registry-v0 | \awk '{print $1;}') 5000:5000  > /dev/null&
   ```

   This exposes the Docker Registry to the Terminal we are using.

   > If you don't get a command line prompt, press `enter` several times

3. And now push the image to the local registry:

   ```bash
   docker push localhost:5000/k8sdemo:lab
   ```

   Messages like `Handling connection for 5000` and `Retrying in x seconds` are due to the Lab setup and can be ignored.

Now the image is available to be aquired (pulled) from any machine that has access to the Docker registry.

Lab 2 - Run the frontend Image from the registry

1. Now let's start the Web Frontend container with the image from the registry

   ```bash
   docker run -d --rm --name k8sdemo -p 3000:3000 --env BACKEND_URL=http://172.17
   .0.1:3001/api localhost:5000/k8sdemo:lab

   > b6e46d8bd60978af7e9e45260111e938da63a64247c9cff3b4e398a6498670a6
   ```

   This command runs the frontend server:

   - --rm makes sure that the container is deleted once it's stopped
   - --name gives the container a fixed name
   - --env defines the environment variable that points to the `k8sdemo-backend` server API
   - -p exposes the container port 3000 to the outside port 3000
   - -d runs the container in the background (as a daemon)
   - localhost:5000/k8sdemo:lab is the image we have pushed to the registry before

2. Go back to your browser and refresh the `k8sdemo` web application to make sure that the container has been started.

Lab 3 - Docker Internals

Now let's have a more in depth look at the running containers.

Docker process inspection

Docker `top` gives you the running processes inside a container. We can see that we have the node server running inside the container.

```
Bash
docker top k8sdemo

> UID                  PID                 PPID                C
 STIME                TTY                 TIME                CMD
> www                  35532               35512               0
 10:48                ?                   00:00:00            npm
> www                  35606               35532               0
 10:48                ?                   00:00:00            sh -c node ./bin/www
> www                  35607               35606               0
 10:48                ?                   00:00:00            node ./bin/www
```

Docker `inspect` outputs the detailed configuration of the running container.

```
Bash
docker inspect k8sdemo

> [
>     {
>         "Id": "48f5cd2708b27da27b0edcc1aaa3b6308249e6de94d45c0b123bf5fc7f2efbbf
",
>         "Created": "2020-05-19T08:48:03.867340977Z",
>         "Path": "docker-entrypoint.sh",
>         "Args": [
>             "npm",
>             "start"
>         ],
>         "State": {
>             "Status": "running",
>  ...
```

Docker logs

In order to access the logs of the container (especially if it runs in the background with the `-d` option) you can use docker `logs`.

```bash
docker logs k8sdemo

> test@0.0.0 start /app
> node ./bin/www
```

Docker processes

As we have seen, running a Docker container starts an isolated process on the host system. So we are able to see those processed running in the VM. One for the frontend and one for the backend.

```bash
ps -Alf | grep "sh -c node"

> 4 S root        9474    9411  0  80   0 -  1070 -      10:00 ?        00:00:00
sh -c node ./bin/www
> 4 S root       12020   11960  0  80   0 -  1070 -      10:04 ?        00:00:00
sh -c node ./bin/www
```

The pstree command shows the process in the complete process tree. If you execute the following command you can see that the processes from our demo application are running under the containerd process, which is the container runtime used by Docker for this lab.

```bash
pstree | grep node

>         |-containerd-+-containerd-shim-+-portainer---10*[{portainer}]
>
>   ...
>
>         |             |-containerd-shim-+-npm-+-sh---node---9*[{node}]
>         |             |                 `-9*[{containerd-shim}]
>         |             |-containerd-shim-+-npm-+-sh---node---5*[{node}]
>         |             |                 `-9*[{containerd-shim}]
>         |             `-17*[{containerd}]
```

Docker exec

Docker `exec` allows to run commands inside already running containers. This is very useful for debugging.

```bash
docker exec -it k8sdemo /bin/bash
```

You get a prompt inside the container. Now you can explore the content of the running container.

1. List the files in the home directory

```bash
root@9bffd6c12fb0:/app# ls -al

> total 84
> drwxr-xr-x 1 root root  4096 May 14 07:25 .
> drwxr-xr-x 1 root root  4096 May 19 08:04 ..
> -rw-rw-r-- 1 root root  6148 May 14 06:55 .DS_Store
> -rw-rw-r-- 1 root root   496 May 14 06:55 Dockerfile
> -rw-rw-r-- 1 root root    78 May 14 06:55 README
> -rw-rw-r-- 1 root root  1226 May 14 06:55 app.js
> drwxrwxr-x 2 root root  4096 May 14 06:55 bin
> -rw-rw-r-- 1 root root   306 May 14 06:55 build.sh
> drwxrwxr-x 1 root root  4096 May 14 06:55 node_modules
> -rw-rw-r-- 1 root root 26382 May 14 06:55 package-lock.json
> -rw-rw-r-- 1 root root   307 May 14 06:55 package.json
> drwxrwxr-x 5 root root  4096 May 14 06:55 public
> drwxrwxr-x 2 root root  4096 May 14 06:55 routes
> drwxrwxr-x 2 root root  4096 May 14 06:55 views
```

2. List the web files for the node application

```bash
root@9bffd6c12fb0:/app# cd public/
```

```bash
root@9bffd6c12fb0:/app# ls -al

> total 144
> drwxrwxr-x 1 root root  4096 May 14 06:55 .
> drwxr-xr-x 1 root root  4096 May 14 07:25 ..
> -rw-rw-r-- 1 root root  6148 May 14 06:55 .DS_Store
> -rw-rw-r-- 1 root root 32939 May 14 06:55 404.html
> -rw-rw-r-- 1 root root 55515 May 14 06:55 500.html
> drwxrwxr-x 2 root root  4096 May 14 06:55 images
> -rwxrwxr-x 1 root root  6240 May 19 08:22 index.html
> -rwxrwxr-x 1 root root  1188 May 14 06:55 index.js
> drwxrwxr-x 4 root root  4096 May 14 06:55 public
> -rwxrwxr-x 1 root root  1157 May 14 06:55 style.css
> drwxrwxr-x 2 root root  4096 May 14 06:55 stylesheets
```

3. Modify the HTML code for the running container.

```bash
root@9bffd6c12fb0:/app# echo "HELLO" >> index.html
```

4. Reload the Web App and look for the HELLO text displayed in the lower left part of the page.

5. Type `exit` in order to get back to the commandline

Lab 4 - Cleanup

To conclude this Lab we have to clean up the containers that we have created

1. Terminate the frontend

   ```bash
   docker kill k8sdemo

   > k8sdemo
   ```

2. And Terminate the backend

   ```bash
   docker kill k8sdemo-backend

   > k8sdemo-backend
   ```

3. Verify that the two containers have been terminated

   ```bash
   docker ps | grep k8sdemo
   ```

   This command must return no result.

Congratulations!!! This concludes the Labs on Docker