DevOps with DataPower

Harish Chadalawada
Apigee Developer
Miracle Software Systems,Inc.

Venkata Lakshmi Kondapalli
Apigee Developer
Miracle Software Systems, Inc.

www.miraclesoft.com

1. Introduction:

- Modern application development is fast and getting faster. Manual processes are necessarily being eliminated in favor of automation.
- Automation itself is moving oneoff tasks maintained by a person or an organization to automation regimes that are hosted elsewhere.
- In this application we used DevOps, CI, and CD methodology on an existing DataPower application.
- **Required Technologies**:

  → Docker

  → Docker Compose

  → Gitlab

  → Jenkins

- Here we start with an existing DataPower configuration turn it into a Docker image and save the configuration as files human-readable "source" files that we will place under version control.

- From there, we'll fully containerize the application by putting both the test client and the other servers into containers and the containers into a composed application with Docker Compose.

- Once we have the composed application, we'll set up two different ways we can work with it.

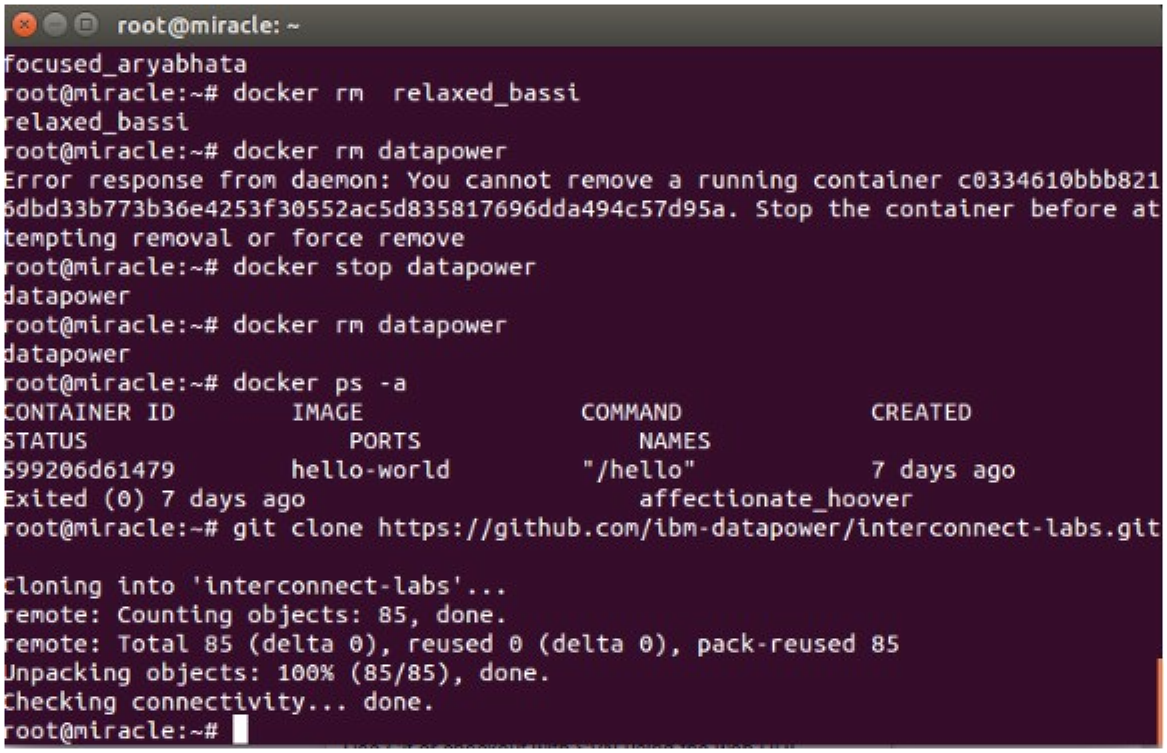
  → The first way will be as a developer would – making small changes to configuration and source and testing them live.
  → The second way we want to run our composed application is as a standalone test.

- Once implementation is complete then it is also used by the continuous integration regime to validate the application on every check-in.

- Now we will use Jenkins to perform Continuous Integration(CI) for our Docker Compose application.

Preparation:

- We used Ubuntu machine and it has all the tools required to complete the POC.
- Installed Docker and docker-compose and the official DataPower Docker image is pre-loaded.
- Both Chrome and Firefox are configured to open a tab on the GitHub **ibm-datapower/interconnect-labs** page.
- This repository contains, among other things, the DataPower application that we will use as the basis for the POC.
- To get the DataPower configuration from GitHub, use the following command:



root@miracle: ~

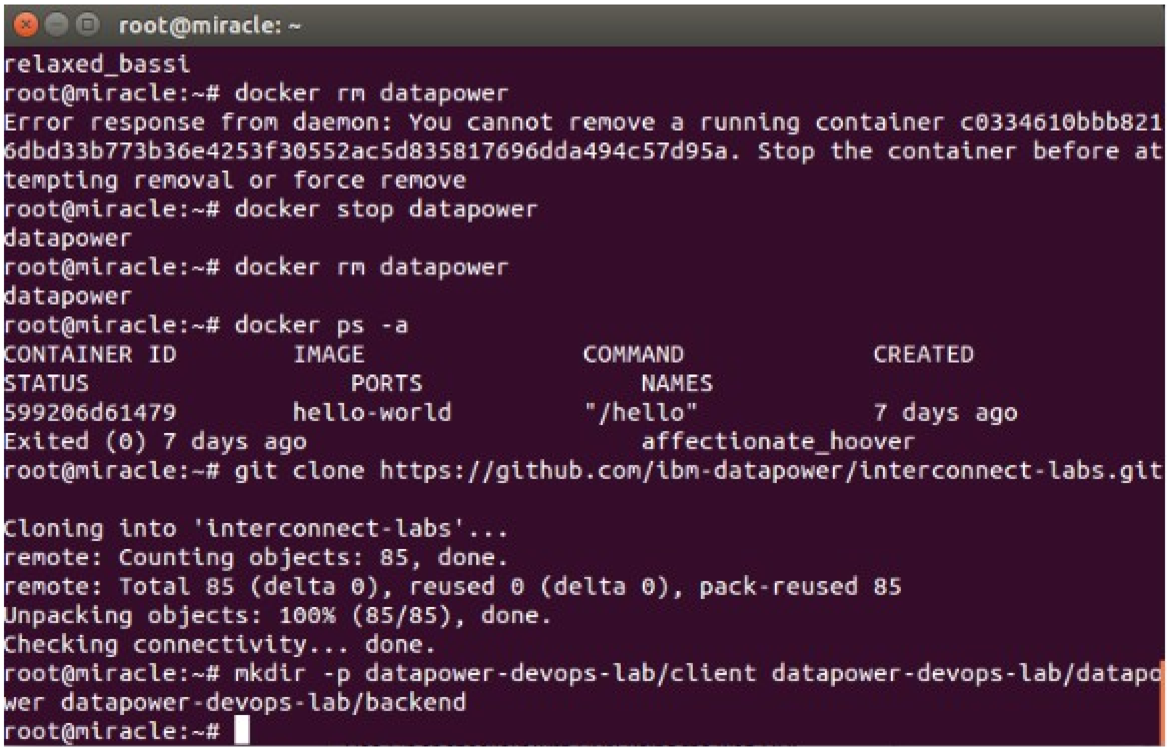
```
focused_aryabhata
root@miracle:~# docker rm  relaxed_bassi
relaxed_bassi
root@miracle:~# docker rm datapower
Error response from daemon: You cannot remove a running container c0334610bbb821
6dbd33b773b36e4253f30552ac5d835817696dda494c57d95a. Stop the container before at
tempting removal or force remove
root@miracle:~# docker stop datapower
datapower
root@miracle:~# docker rm datapower
datapower
root@miracle:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                  PORTS               NAMES
599206d61479        hello-world         "/hello"            7 days ago
Exited (0) 7 days ago                       affectionate_hoover
root@miracle:~# git clone https://github.com/ibm-datapower/interconnect-labs.git

Cloning into 'interconnect-labs'...
remote: Counting objects: 85, done.
remote: Total 85 (delta 0), reused 0 (delta 0), pack-reused 85
Unpacking objects: 100% (85/85), done.
Checking connectivity... done.
root@miracle:~# 
```

Bootstrapping a DataPower for Docker project:

- Now that we have export.zip from the lab's GitHub repository, we have all the artifacts that are required to bring the existing DataPower configuration into IDG for Docker.

Setting up the project

- we'll create a directory for use for our project – a project that will eventually include containers for DataPower, the load driver, and backend server.
- Now create a directory **datapower-devops-lab**. Under that, we'll create directories for client, datapower, and backend as swown in the below figure.



```
root@miracle: ~
relaxed_bassi
root@miracle:~# docker rm datapower
Error response from daemon: You cannot remove a running container c0334610bbb821
6dbd33b773b36e4253f30552ac5d835817696dda494c57d95a. Stop the container before at
tempting removal or force remove
root@miracle:~# docker stop datapower
datapower
root@miracle:~# docker rm datapower
datapower
root@miracle:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                    PORTS               NAMES
599206d61479        hello-world         "/hello"            7 days ago
Exited (0) 7 days ago                         affectionate_hoover
root@miracle:~# git clone https://github.com/ibm-datapower/interconnect-labs.git

Cloning into 'interconnect-labs'...
remote: Counting objects: 85, done.
remote: Total 85 (delta 0), reused 0 (delta 0), pack-reused 85
Unpacking objects: 100% (85/85), done.
Checking connectivity... done.
root@miracle:~# mkdir -p datapower-devops-lab/client datapower-devops-lab/datapo
wer datapower-devops-lab/backend
root@miracle:~# 
```

-

Running DataPower

- Now we have created the directory structure, let's get to work on datapower. For that

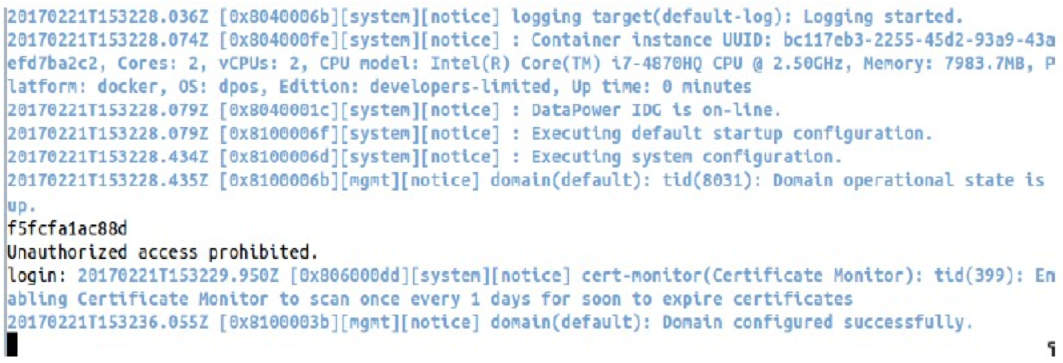
  Issue the following command:

  ```
  $ cd ~/datapower-devops-lab/datapower.
  ```

- we will use DataPower running inside Docker to create its own configuration. The configuration is contained entirely within DataPower's config: and local: directories with the following command.
- Run IDG for Docker with the following command:

  ```
  localuser@ubuntu-base:~/datapower-devops-lab/datapower$ \
     docker run -it \
     -v $PWD/config:/drouter/config \
     -v $PWD/local:/drouter/local \
     -e DATAPOWER_ACCEPT_LICENSE=true \
     -e DATAPOWER_INTERACTIVE=true \
     -e DATAPOWER_WORKER_THREADS=2 \
     -p 9090:9090 -p 80:80 -p 443:443 \
     --name datapower \ ibmcom/datapower
  ```

- Now we will see the DataPower default log as follows.



```
20170221T153228.036Z [0x8040006b][system][notice] logging target(default-log): Logging started.
20170221T153228.074Z [0x804000fe][system][notice] : Container instance UUID: bc117eb3-2255-45d2-93a9-43a
efd7ba2c2, Cores: 2, vCPUs: 2, CPU model: Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz, Memory: 7983.7MB, P
latform: docker, OS: dpos, Edition: developers-limited, Up time: 0 minutes
20170221T153228.079Z [0x8040001c][system][notice] : DataPower IDG is on-line.
20170221T153228.079Z [0x8100006f][system][notice] : Executing default startup configuration.
20170221T153228.434Z [0x8100006d][system][notice] : Executing system configuration.
20170221T153228.435Z [0x8100006b][mgmt][notice] domain(default): tid(8031): Domain operational state is
up.
f5fcfa1ac88d
Unauthorized access prohibited.
login: 20170221T153229.950Z [0x806000dd][system][notice] cert-monitor(Certificate Monitor): tid(399): En
abling Certificate Monitor to scan once every 1 days for soon to expire certificates
20170221T153236.055Z [0x8100003b][mgmt][notice] domain(default): Domain configured successfully.
```

- Next have a completely unconfigured DataPower gateway running!

DataPower Initialization

- DataPower is easier to work with using web management, but web management is off by default. We also don't want to enable web management while DataPower still has the default password.
- In this step, we log in, change the admin password, then enable web management.
- Next Log in with user admin and password admin.

```
login: admin
Password: *****

Welcome to IBM DataPower Gateway console configuration.
Copyright IBM Corporation 1999-2017

Version: IDG.7.5.2.2 build 283762 on Jan 11, 2017 7:43:36 PM
Serial number: 0000001

idg#
```

- Now we'll change the password for admin first. To do that first enter configure mode then use the user-password command.

```
idg# configure
Global configuration mode
idg(config)# user-password
Enter old password: *****
Enter new password: ********
Re-enter new password: ********
Password for user 'admin' changed
Cleared RBM cache
20170221T161125.709Z [0x8100000c][mgmt][notice] : tid(111): Saved current
configuration to 'config:///auto-user.cfg'
```

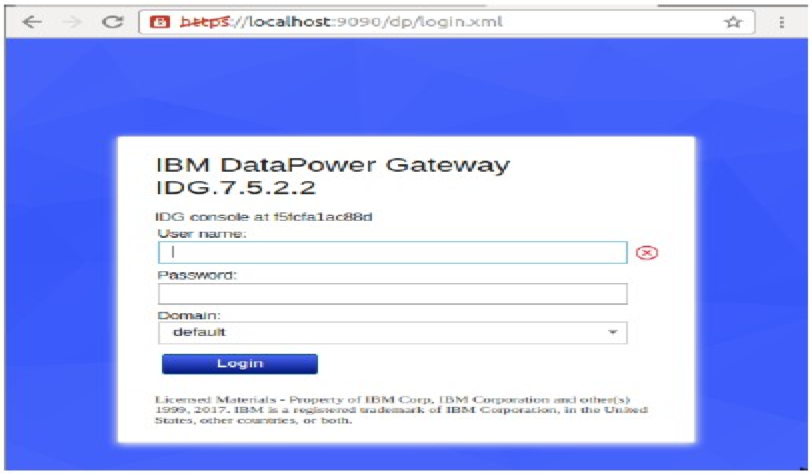
- Next enable web management with the commands web-mgmt, admin-enabled,

exit.

```
idg(config)# web-mgmt
Modify Web Management Service configuration

idg(config web-mgmt)# admin enabled
idg(config web-mgmt)# exit
idg(config)#
20170221T161604.565Z [0x8100003f][mgmt][notice] domain(default): tid(303):
Domain configuration has been modified.
20170221T161604.566Z [0x00350014][mgmt][notice] web-mgmt(WebGUI-Settings):
tid(303): Operational state up
```

Importing DataPower Configuration

- Now that web management is enabled, point your favorite browser at https://localhost:9090 and log in as admin with your new password.
- We can see the DataPower GUI now.



https://localhost:9090/dp/login.xml

IBM DataPower Gateway
IDG.7.5.2.2

IDG console at f51cfa1ac88d

User name:

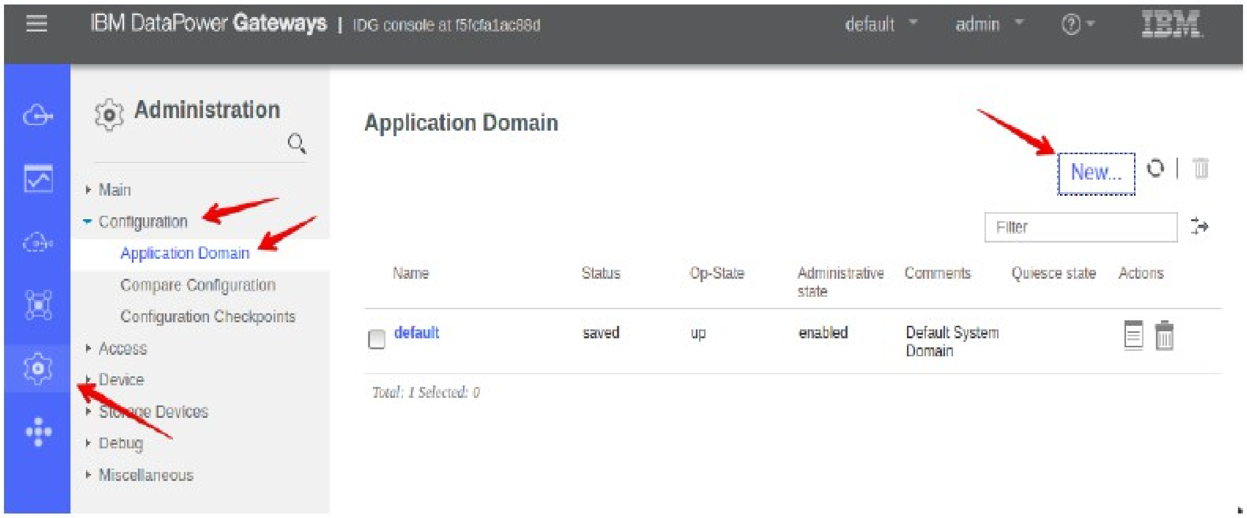
Password:

Domain:
default

Login

Licensed Materials - Property of IBM Corp, IBM Corporation and other(s) 1999, 2017. IBM is a registered trademark of IBM Corporation, in the United States, other countries, or both.

- After logging in, navigate to Administration → Configuration → Application Domain → New then the following screen will be appear.



IBM DataPower Gateways | IDG console at f51cfa1ac88d

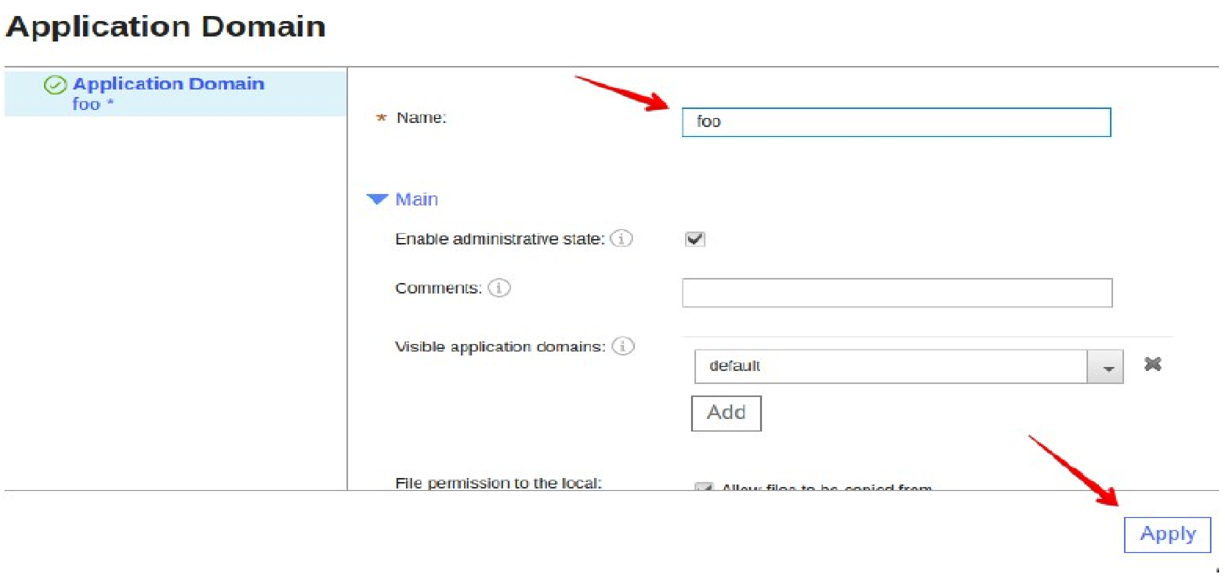
Application Domain

Name	Status	Op-State	Administrative state	Comments	Quiesce state	Actions
default	saved	up	enabled	Default System Domain		

Total: 1 Selected: 0

- Now we want to create the DataPower application in its own Application Domain since this will both keep the focus entirely on the application and it will be more like the eventual deployment target.

- Name the Application Domain **foo** then click on Apply.



Application Domain

Application Domain
foo *

* Name: foo

Main

Enable administrative state: ✓

Comments:

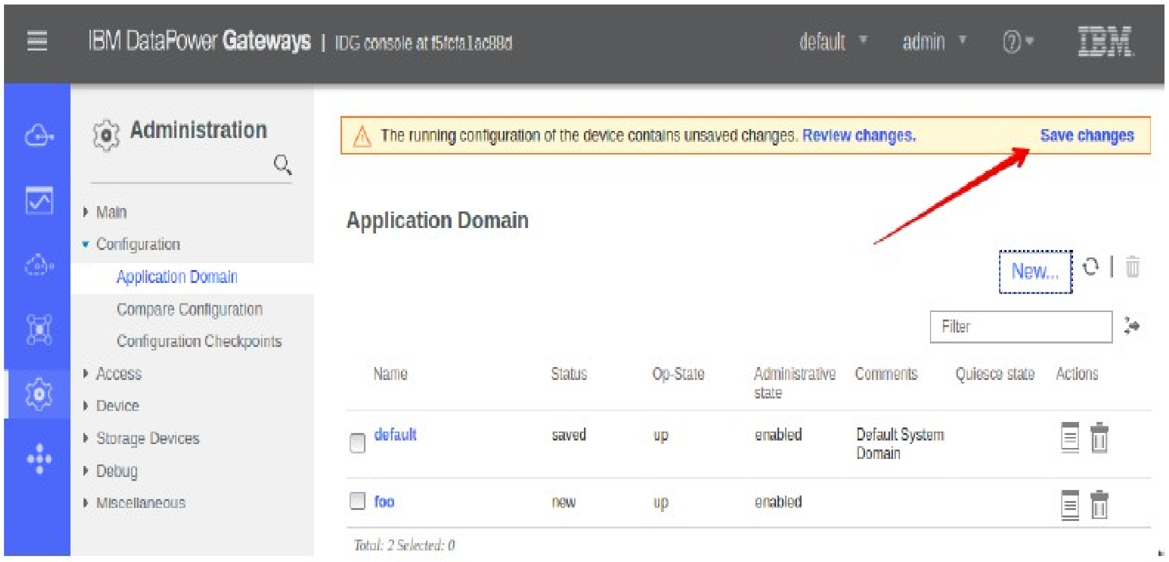
Visible application domains: default

Add

File permission to the local: Allow files to be copied from

Apply

- After the application domain is created, and click on Save changes.



IBM DataPower Gateways | IDG console at f5fcfa1ac88d   default   admin

Administration

Main
Configuration
Application Domain
Compare Configuration
Configuration Checkpoints
Access
Device
Storage Devices
Debug
Miscellaneous

The running configuration of the device contains unsaved changes. Review changes. Save changes

Application Domain

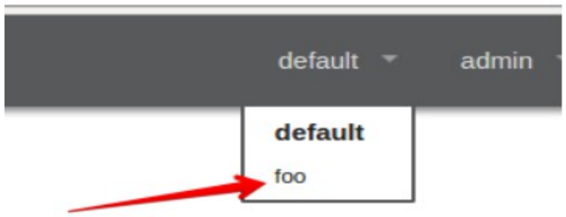
New...

Filter

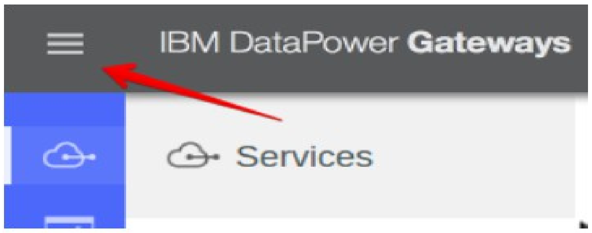
Name	Status	Op-State	Administrative state	Comments	Quiesce state	Actions
default	saved	up	enabled	Default System Domain		
foo	new	up	enabled			

Total: 2 Selected: 0

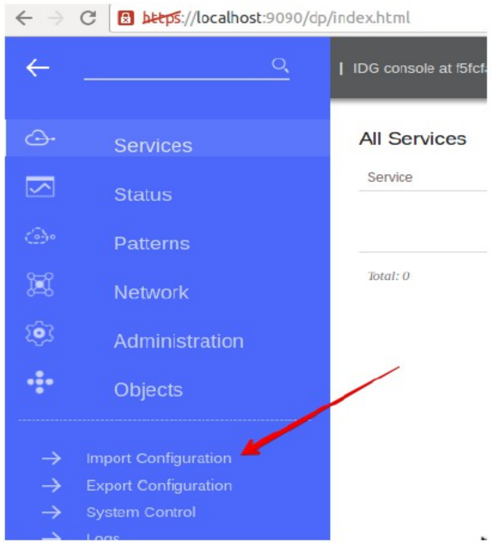
- Next we have to import the configuration into the domain, For that we first switch into the domain.
- For that drop down the default menu at the right corner in the web UI and select **foo** domain as follows.



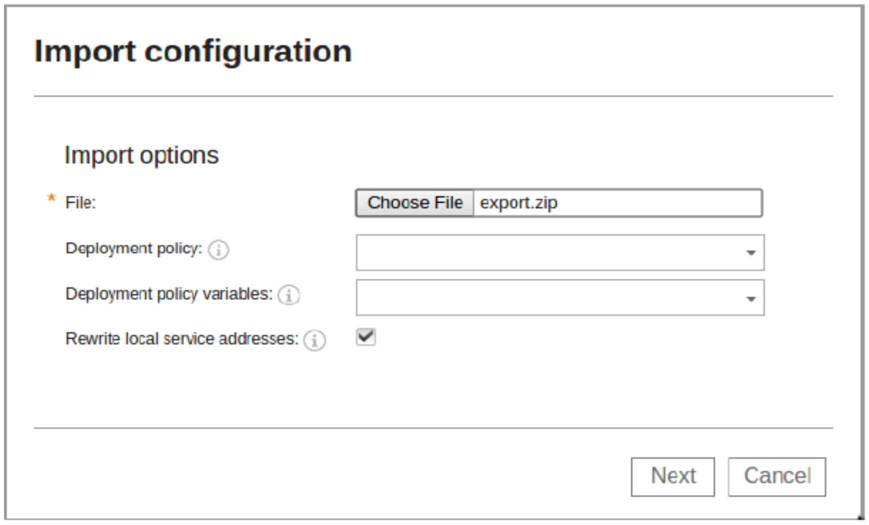
- we can import the configuration that is in the **interconnect-labs** GitHub repository.
- To import the configuration, first click on the "hamburger" icon in the upper left corner of the management screen.



- Then press Import Configuration on the resulting pane.



- On the Import configuration screen, click on Choose File button and navigate to the localuser home directory.
- From there, select **interconnect-labs/IC17-6381-DataPower-DevOps/export.zip** and choose Next as follows.



Import configuration

Import options

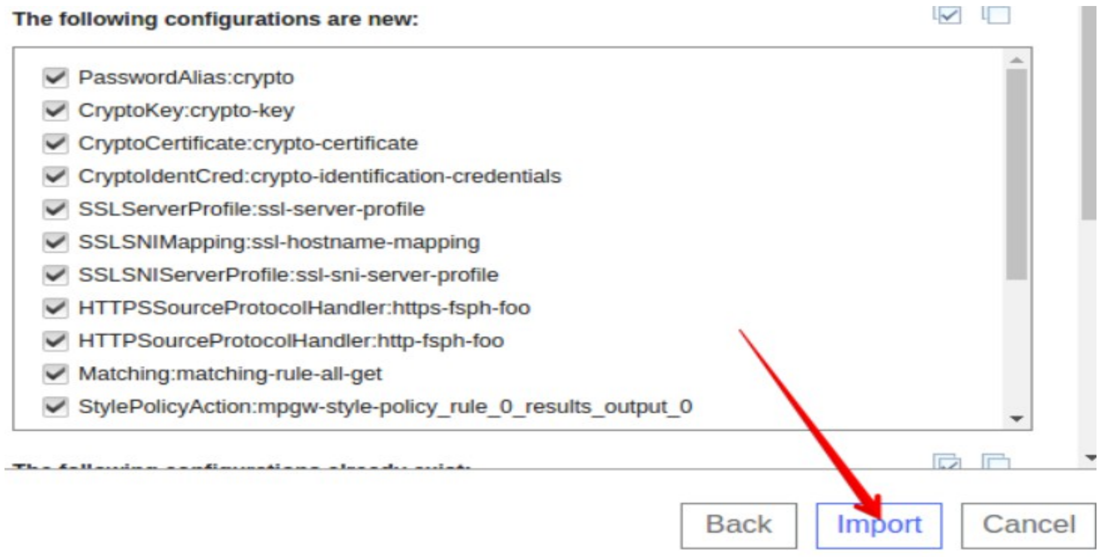
* File: Choose File export.zip

Deployment policy:

Deployment policy variables:

Rewrite local service addresses: ✓

- After you have reviewed the configuration and files that will be imported, click **Import** as follows**.**

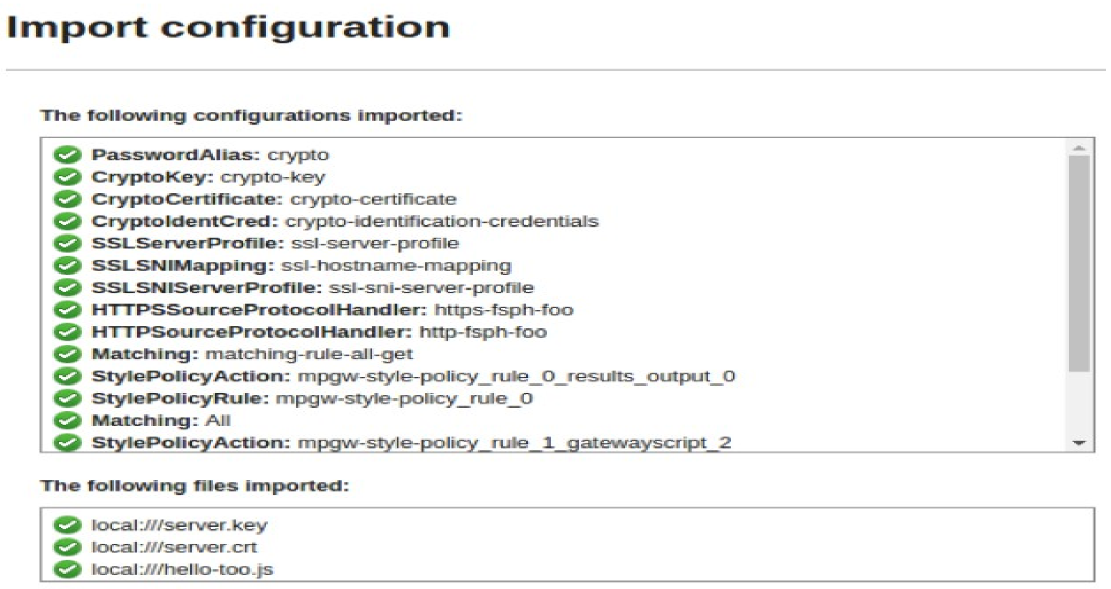


The following configurations are new:

- ✓ PasswordAlias:crypto
- ✓ CryptoKey:crypto-key
- ✓ CryptoCertificate:crypto-certificate
- ✓ CryptoIdentCred:crypto-identification-credentials
- ✓ SSLServerProfile:ssl-server-profile
- ✓ SSLSNIMapping:ssl-hostname-mapping
- ✓ SSLSNIServerProfile:ssl-sni-server-profile
- ✓ HTTPSSourceProtocolHandler:https-fsph-foo
- ✓ HTTPSourceProtocolHandler:http-fsph-foo
- ✓ Matching:matching-rule-all-get
- ✓ StylePolicyAction:mpgw-style-policy_rule_0_results_output_0

The following configurations already exist:

- Then see that all the configuration objects and files were successfully imported and close the import screen.



Import configuration

The following configurations imported:

```
✓ PasswordAlias: crypto
✓ CryptoKey: crypto-key
✓ CryptoCertificate: crypto-certificate
✓ CryptoIdentCred: crypto-identification-credentials
✓ SSLServerProfile: ssl-server-profile
✓ SSLSNIMapping: ssl-hostname-mapping
✓ SSLSNIServerProfile: ssl-sni-server-profile
✓ HTTPSSourceProtocolHandler: https-fsph-foo
✓ HTTPSourceProtocolHandler: http-fsph-foo
✓ Matching: matching-rule-all-get
✓ StylePolicyAction: mpgw-style-policy_rule_0_results_output_0
✓ StylePolicyRule: mpgw-style-policy_rule_0
✓ Matching: All
✓ StylePolicyAction: mpgw-style-policy_rule_1_gatewayscript_2
```

The following files imported:

```
✓ local:///server.key
✓ local:///server.crt
✓ local:///hello-too.js
```

- Now domain has been successfully imported.

Testing the Imported Application

- Now we have to check the application whether it is working or not by pointing curl at the application.
- It should run on both the ports 81 and 443, and since we mapped those ports on our docker run command.
- Next run the following command

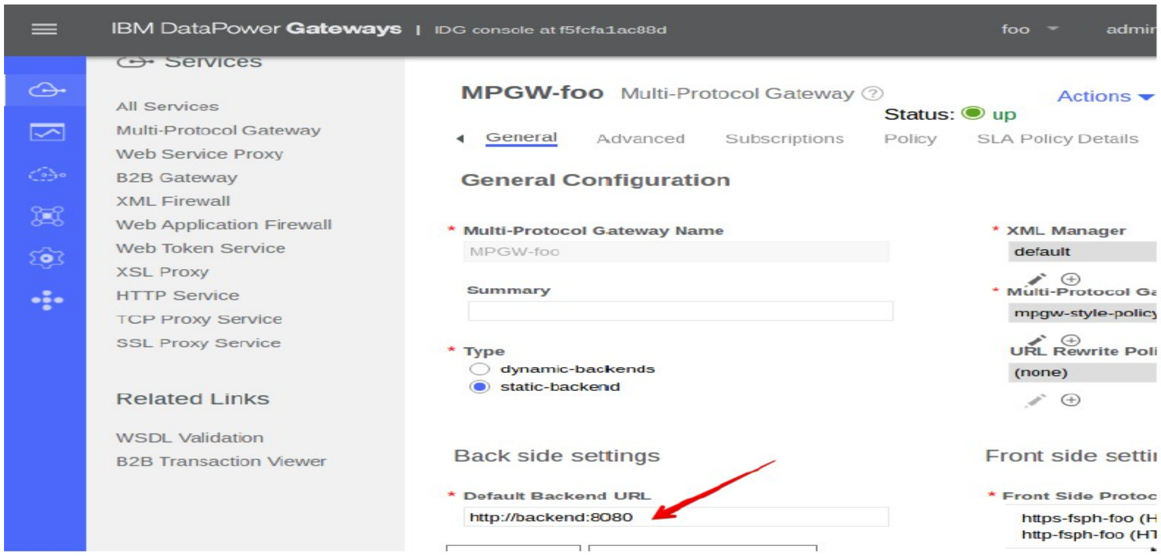
```
localuser@ubuntu-base $ curl http://127.0.0.1
```

- And it didn't work and it shows the following result.

```
<?xml version='1.0' ?>
<env:Envelope      xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
<env:Body>
<env:Fault>
<faultcode>env:Client</faultcode>
<faultstring>Internal Error</faultstring>
```

```
</env:Fault>
</env:Body>
</env:Envelope>
```

- It is showing error because here we did not change the backend. It had the value that was correct for the exported system, but it is not correct here.
- To fix it, drill into MPGW-foo in the web management interface and choose a better value for the Default Backend URL.
- Value might be http://www.ibm.com/. After make all the changes, click Apply as follows.



Status: up

* Multi-Protocol Gateway Name: MPGW-foo

Summary:

* Type: dynamic-backends [✓ static-backend]

* XML Manager: default

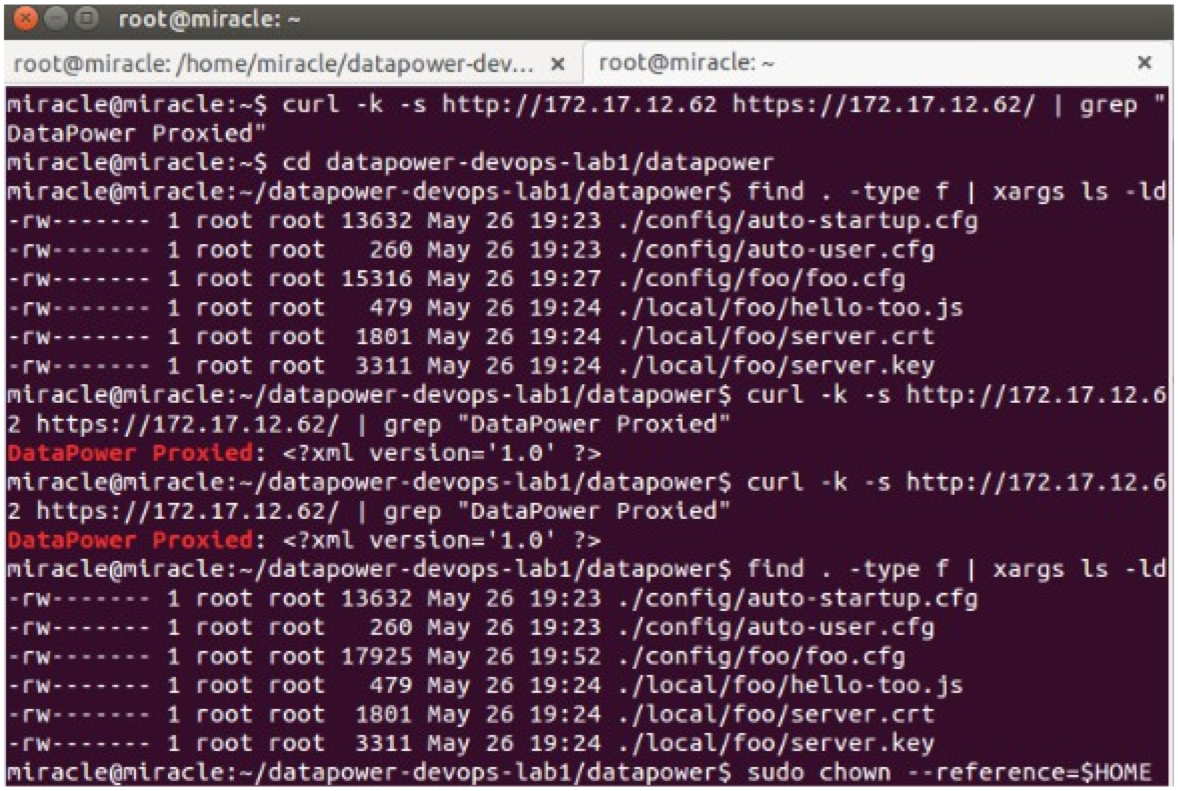
* Multi-Protocol Ga: mpgw-style-policy

URL Rewrite Poli: (none)

* Default Backend URL: http://backend:8080

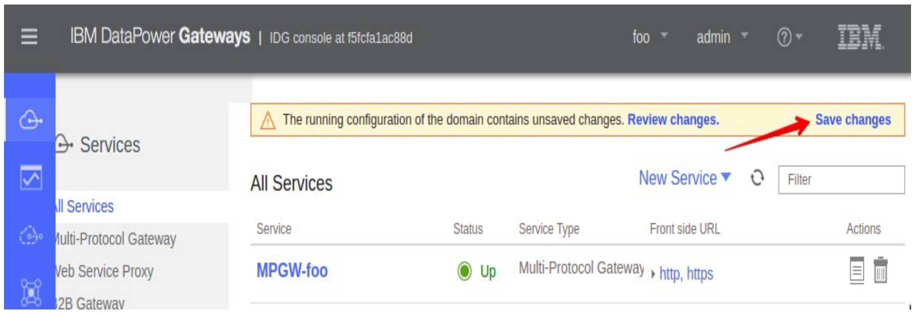
* Front Side Protoc: https-fsph-foo (H, http-fsph-foo (HT

- Now test with both http and https with the following commands.



```
miracle@miracle:~$ curl -k -s http://172.17.12.62 https://172.17.12.62/ | grep "
DataPower Proxied"
miracle@miracle:~$ cd datapower-devops-lab1/datapower
miracle@miracle:~/datapower-devops-lab1/datapower$ find . -type f | xargs ls -ld
-rw------- 1 root root 13632 May 26 19:23 ./config/auto-startup.cfg
-rw------- 1 root root   260 May 26 19:23 ./config/auto-user.cfg
-rw------- 1 root root 15316 May 26 19:27 ./config/foo/foo.cfg
-rw------- 1 root root   479 May 26 19:24 ./local/foo/hello-too.js
-rw------- 1 root root  1801 May 26 19:24 ./local/foo/server.crt
-rw------- 1 root root  3311 May 26 19:24 ./local/foo/server.key
miracle@miracle:~/datapower-devops-lab1/datapower$ curl -k -s http://172.17.12.6
2 https://172.17.12.62/ | grep "DataPower Proxied"
DataPower Proxied: <?xml version='1.0' ?>
miracle@miracle:~/datapower-devops-lab1/datapower$ curl -k -s http://172.17.12.6
2 https://172.17.12.62/ | grep "DataPower Proxied"
DataPower Proxied: <?xml version='1.0' ?>
miracle@miracle:~/datapower-devops-lab1/datapower$ find . -type f | xargs ls -ld
-rw------- 1 root root 13632 May 26 19:23 ./config/auto-startup.cfg
-rw------- 1 root root   260 May 26 19:23 ./config/auto-user.cfg
-rw------- 1 root root 17925 May 26 19:52 ./config/foo/foo.cfg
-rw------- 1 root root   479 May 26 19:24 ./local/foo/hello-too.js
-rw------- 1 root root  1801 May 26 19:24 ./local/foo/server.crt
-rw------- 1 root root  3311 May 26 19:24 ./local/foo/server.key
miracle@miracle:~/datapower-devops-lab1/datapower$ sudo chown --reference=$HOME
```

- Now click on the **Save changes** in UI part as shown in the following screen.



The running configuration of the domain contains unsaved changes. Review changes. Save changes

All Services

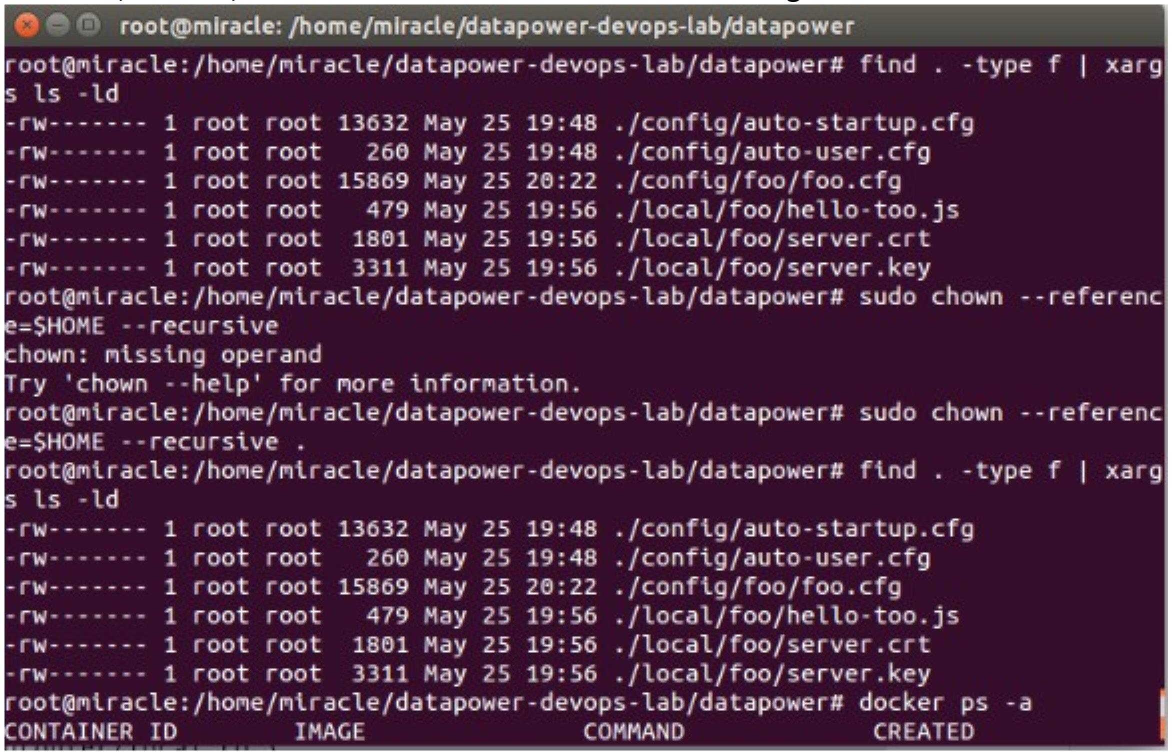
Service	Status	Service Type	Front side URL	Actions
MPGW-foo	Up	Multi-Protocol Gateway	http, https	

- Now we dont need to run this container, for that we have stop the container with the following commands.

```
idg(config)# exit

idg# exit
Goodbye.
f5fcfa1ac88d
```

```
Unauthorized access prohibited.
login: ^C
localuser@ubuntu-base:~/datapower-devops-lab/datapower$
```

- This process was to build a DataPower configuration suitable for use in Docker.
- Now, we first, check the files we made with the following command.



root@miracle: /home/miracle/datapower-devops-lab/datapower

```
root@miracle:/home/miracle/datapower-devops-lab/datapower# find . -type f | xarg
s ls -ld
-rw------- 1 root root 13632 May 25 19:48 ./config/auto-startup.cfg
-rw------- 1 root root   260 May 25 19:48 ./config/auto-user.cfg
-rw------- 1 root root 15869 May 25 20:22 ./config/foo/foo.cfg
-rw------- 1 root root   479 May 25 19:56 ./local/foo/hello-too.js
-rw------- 1 root root  1801 May 25 19:56 ./local/foo/server.crt
-rw------- 1 root root  3311 May 25 19:56 ./local/foo/server.key
root@miracle:/home/miracle/datapower-devops-lab/datapower# sudo chown --referenc
e=$HOME --recursive
chown: missing operand
Try 'chown --help' for more information.
root@miracle:/home/miracle/datapower-devops-lab/datapower# sudo chown --referenc
e=$HOME --recursive .
root@miracle:/home/miracle/datapower-devops-lab/datapower# find . -type f | xarg
s ls -ld
-rw------- 1 root root 13632 May 25 19:48 ./config/auto-startup.cfg
-rw------- 1 root root   260 May 25 19:48 ./config/auto-user.cfg
-rw------- 1 root root 15869 May 25 20:22 ./config/foo/foo.cfg
-rw------- 1 root root   479 May 25 19:56 ./local/foo/hello-too.js
-rw------- 1 root root  1801 May 25 19:56 ./local/foo/server.crt
-rw------- 1 root root  3311 May 25 19:56 ./local/foo/server.key
root@miracle:/home/miracle/datapower-devops-lab/datapower# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
```

→ The file **autostartup.cfg** contains the default domain configuration.
→ The file **auto-user.cfg** contains the hash of the admin password.
→ The file **foo.cfg** contains the domain foo configuration.
→ In **local** we have the GatewayScript and crypto material.

- The files are all owned by the **root.**

Docker Build and the DataPower Application

Creating the Dockerfile

Now we need a Dockerfile and we need to docker build. In our datapower directory, create a file named Dockerfile with the following command.

```
localuser@ubuntu-base:~/datapower-devops-lab/datapower$ atom Dockerfile
```

- Now add the text into that file as follows.

```
FROM ibmcom/datapower:latest
ENV  DATAPOWER_ACCEPT_LICENSE=true \
     DATAPOWER_WORKER_THREADS=2 \
     DATAPOWER_INTERACTIVE=true
COPY config/ /drouter/config/
COPY local/ /drouter/local/
EXPOSE 80 443 9090
```

- Now we compares the **foo** service in DataPower – and to the docker run commands we've been using.

→ The FROM line indicates the starting point for the new image. We are staring from **ibmcom/datapower:latest** since that is what we have been running so far.

→ The **ENV** line allows us to build in environment variables instead of pass them in every time.

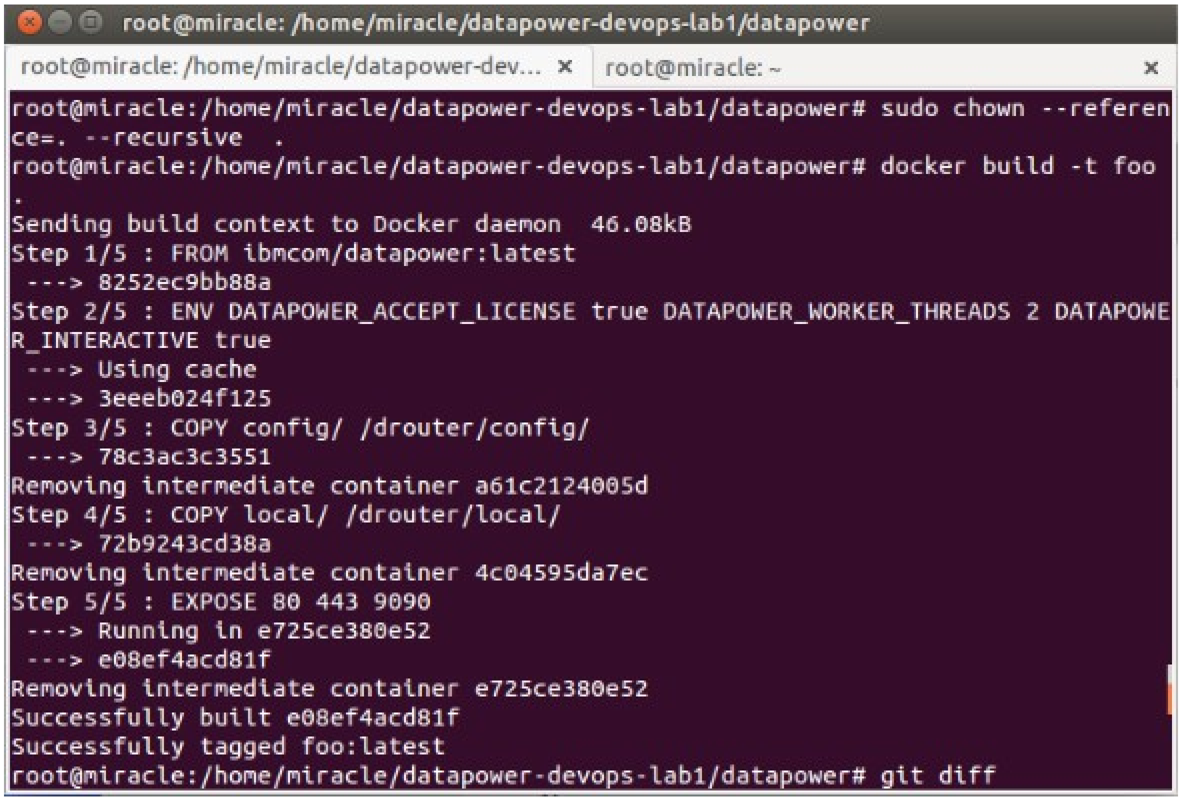
→ Next we COPY the **./config** and **./local** directories into the new image.

→ Lastly, we **EXPOSE** the ports that make up our application. These ports will be automatically mapped by Docker if the –P option is used.

Building the Customized DataPower Docker Image

- Now that we have both the Dockerfile and our source.

- Next we have to build our DataPower Docker image. It contains the application called Foo, so we'll tag the resulting image as **foo**.


root@miracle: /home/miracle/datapower-devops-lab1/datapower
root@miracle: /home/miracle/datapower-dev...
root@miracle: ~

```
root@miracle:/home/miracle/datapower-devops-lab1/datapower# sudo chown --referen
ce=. --recursive  .
root@miracle:/home/miracle/datapower-devops-lab1/datapower# docker build -t foo
.
Sending build context to Docker daemon  46.08kB
Step 1/5 : FROM ibmcom/datapower:latest
 ---> 8252ec9bb88a
Step 2/5 : ENV DATAPOWER_ACCEPT_LICENSE true DATAPOWER_WORKER_THREADS 2 DATAPOWE
R_INTERACTIVE true
 ---> Using cache
 ---> 3eeeb024f125
Step 3/5 : COPY config/ /drouter/config/
 ---> 78c3ac3c3551
Removing intermediate container a61c2124005d
Step 4/5 : COPY local/ /drouter/local/
 ---> 72b9243cd38a
Removing intermediate container 4c04595da7ec
Step 5/5 : EXPOSE 80 443 9090
 ---> Running in e725ce380e52
 ---> e08ef4acd81f
Removing intermediate container e725ce380e52
Successfully built e08ef4acd81f
Successfully tagged foo:latest
root@miracle:/home/miracle/datapower-devops-lab1/datapower# git diff
```

- Now we can check docker images and it will see foo in the list.

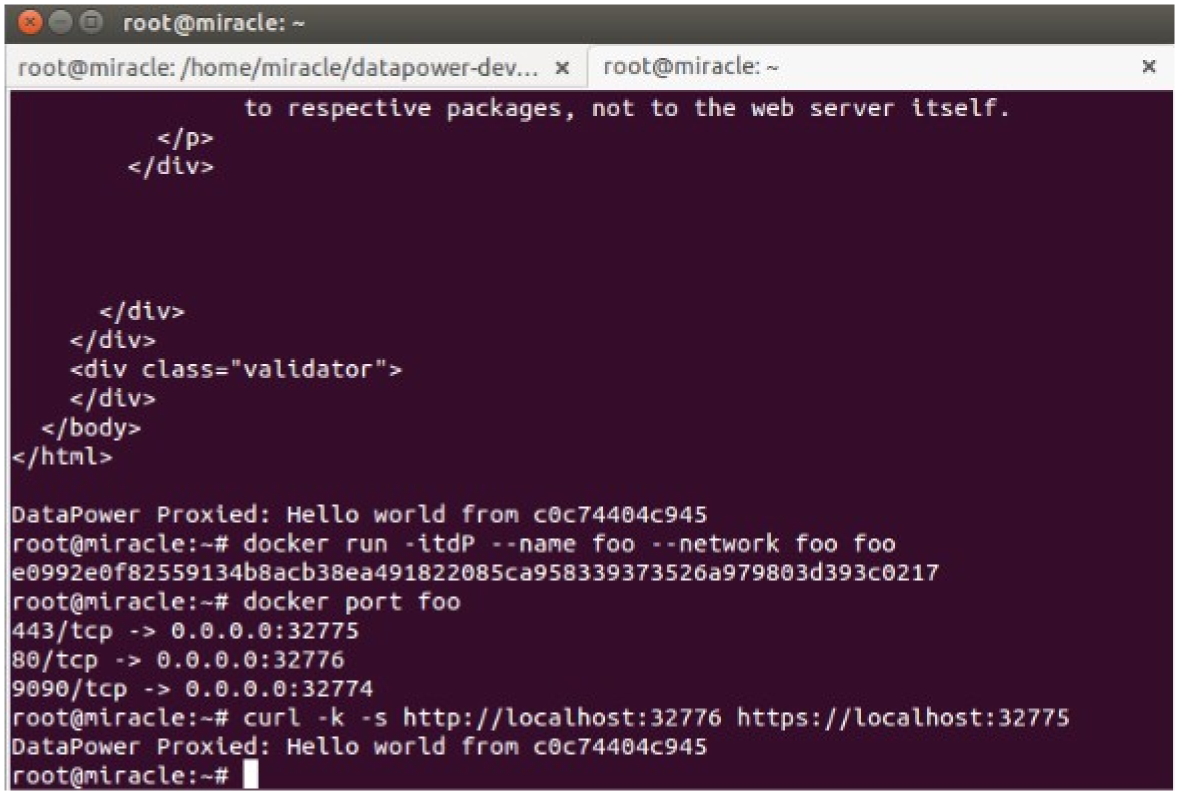
Testing the Build Image

- Now we can test the image and will be just a bit different because now we will let Docker map the ports for us.

- Next the container is running but we don’t know which ports have been mapped.

We use the following command to find out the docker port.

Now that we know which ports are mapped, we can use the ports in the URLs in the curl command to test the application



```
                to respective packages, not to the web server itself.
          </p>
        </div>




      </div>
    </div>
    <div class="validator">
    </div>
  </body>
</html>

DataPower Proxied: Hello world from c0c74404c945
root@miracle:~# docker run -itdP --name foo --network foo foo
e0992e0f82559134b8acb38ea491822085ca958339373526a979803d393c0217
root@miracle:~# docker port foo
443/tcp -> 0.0.0.0:32775
80/tcp -> 0.0.0.0:32776
9090/tcp -> 0.0.0.0:32774
root@miracle:~# curl -k -s http://localhost:32776 https://localhost:32775
DataPower Proxied: Hello world from c0c74404c945
root@miracle:~#
```

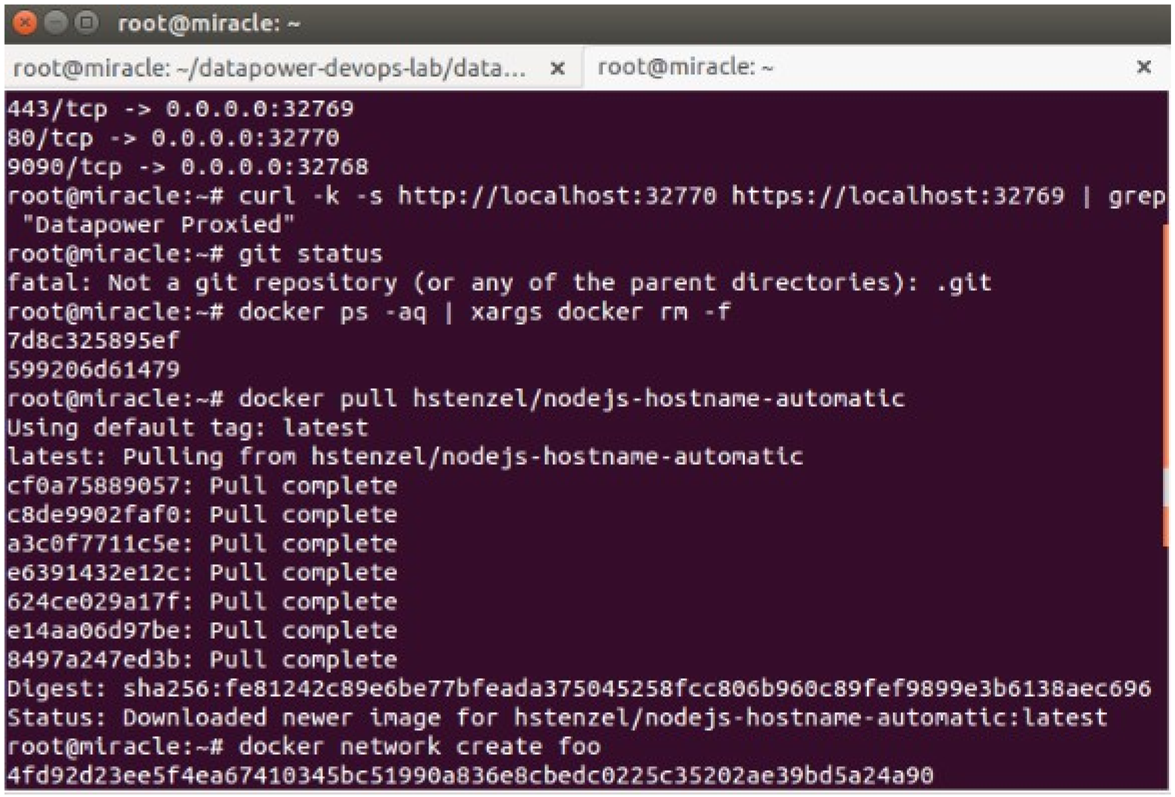
Now its all done. We have few containers running out there without any use. We need to remove the unwanted containers running.

```
$ docker ps –aq | xargs docker rm -f
```

Containerize the Backend:

- We have to connect with the backend so that we need to have a backend. Till now we dont deal with the backend except with the docker commands.
- In this case we are using the default backend image from the docker hub given below. We need to pull it.

https://hub.docker.com/r/hstenzel/nodejs-hostname-automatic/



```
443/tcp -> 0.0.0.0:32769
80/tcp -> 0.0.0.0:32770
9090/tcp -> 0.0.0.0:32768
root@miracle:~# curl -k -s http://localhost:32770 https://localhost:32769 | grep
 "Datapower Proxied"
root@miracle:~# git status
fatal: Not a git repository (or any of the parent directories): .git
root@miracle:~# docker ps -aq | xargs docker rm -f
7d8c325895ef
599206d61479
root@miracle:~# docker pull hstenzel/nodejs-hostname-automatic
Using default tag: latest
latest: Pulling from hstenzel/nodejs-hostname-automatic
cf0a75889057: Pull complete
c8de9902faf0: Pull complete
a3c0f7711c5e: Pull complete
e6391432e12c: Pull complete
624ce029a17f: Pull complete
e14aa06d97be: Pull complete
8497a247ed3b: Pull complete
Digest: sha256:fe81242c89e6be77bfeada375045258fcc806b960c89fef9899e3b6138aec696
Status: Downloaded newer image for hstenzel/nodejs-hostname-automatic:latest
root@miracle:~# docker network create foo
4fd92d23ee5f4ea67410345bc51990a836e8cbedc0225c35202ae39bd5a24a90
```

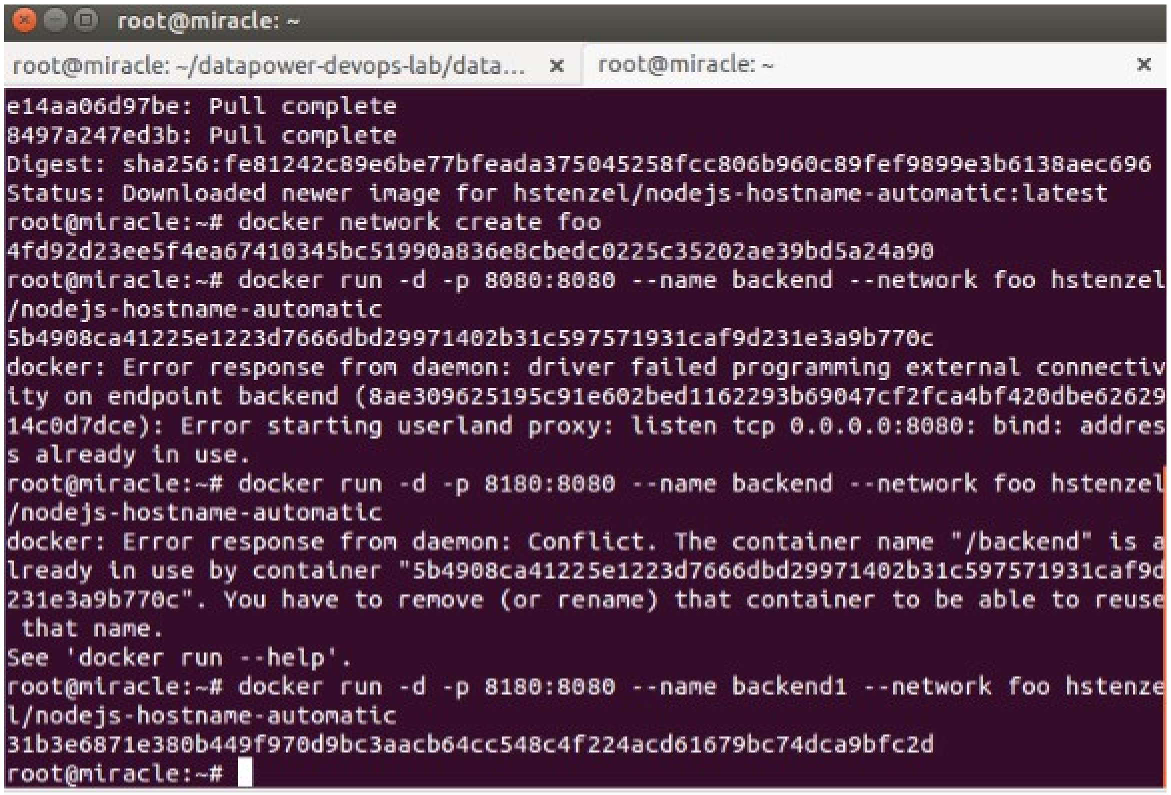
Here we have a conflict that the configuration in our project does not know how to use the containerized backend. We must fix that. But first, let’s prepare to run both DataPower and the backend in the same network.

Creating a network:

Up to now, every container we have run has been in the default Docker network. But what we want is for DataPower-foo and the backend to run all by themselves in the same dedicated network. This way, Docker will treat the names of the containers as their DNS names so the containers can talk to each other without NAT and without being exposed to other containers.

Add the Backend to the Network :

Next, start the backend in the foo network



```
e14aa06d97be: Pull complete
8497a247ed3b: Pull complete
Digest: sha256:fe81242c89e6be77bfeada375045258fcc806b960c89fef9899e3b6138aec696
Status: Downloaded newer image for hstenzel/nodejs-hostname-automatic:latest
root@miracle:~# docker network create foo
4fd92d23ee5f4ea67410345bc51990a836e8cbedc0225c35202ae39bd5a24a90
root@miracle:~# docker run -d -p 8080:8080 --name backend --network foo hstenzel
/nodejs-hostname-automatic
5b4908ca41225e1223d7666dbd29971402b31c597571931caf9d231e3a9b770c
docker: Error response from daemon: driver failed programming external connectiv
ity on endpoint backend (8ae309625195c91e602bed1162293b69047cf2fca4bf420dbe62629
14c0d7dce): Error starting userland proxy: listen tcp 0.0.0.0:8080: bind: addres
s already in use.
root@miracle:~# docker run -d -p 8180:8080 --name backend --network foo hstenzel
/nodejs-hostname-automatic
docker: Error response from daemon: Conflict. The container name "/backend" is a
lready in use by container "5b4908ca41225e1223d7666dbd29971402b31c597571931caf9d
231e3a9b770c". You have to remove (or rename) that container to be able to reuse
 that name.
See 'docker run --help'.
root@miracle:~# docker run -d -p 8180:8080 --name backend1 --network foo hstenze
l/nodejs-hostname-automatic
31b3e6871e380b449f970d9bc3aacb64cc548c4f224acd61679bc74dca9bfc2d
root@miracle:~#
```

Reconfigure DataPower to use the Backend:

With the backend on the private network and responding to traffic, the next step is for DataPower to use http://backend:8080 instead of http://www.ibm.com/ for the backend URL of MPGW-foo.

In order to use the web management to reconfigure DataPower, we'll do the same thing we did before, only this time we'll run DataPower inside our new network. Notice the addition of --network foo and –d to run detached in the background.

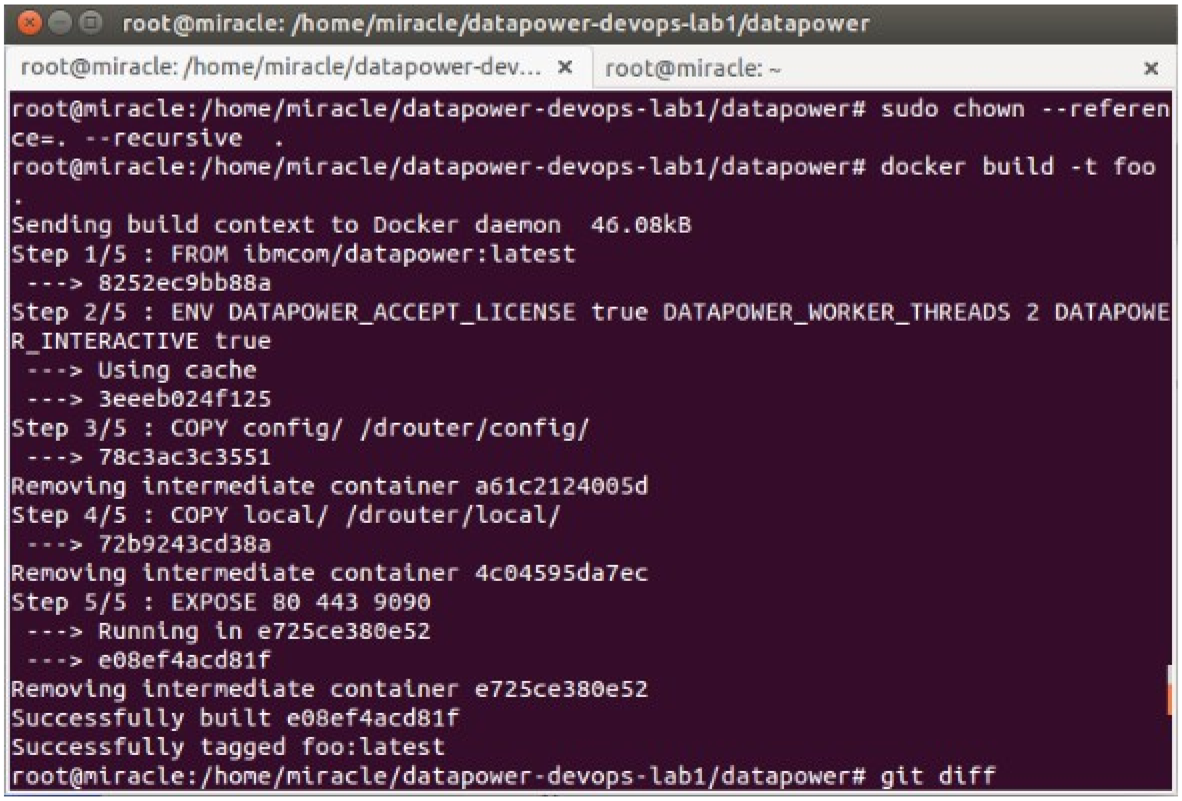
```
localuser@ubuntu-base:~/datapower-devops-lab/datapower$ docker run -itd \
-v $PWD/config:/drouter/config \
-v $PWD/local:/drouter/local \
-e DATAPOWER_ACCEPT_LICENSE=true \
-e DATAPOWER_INTERACTIVE=true \
-e DATAPOWER_WORKER_THREADS=2 \
```

*-p 9090:9090 –p 81:80 –p 443:443 *
*--name datapower *
*--network foo *
ibmcom/datapower

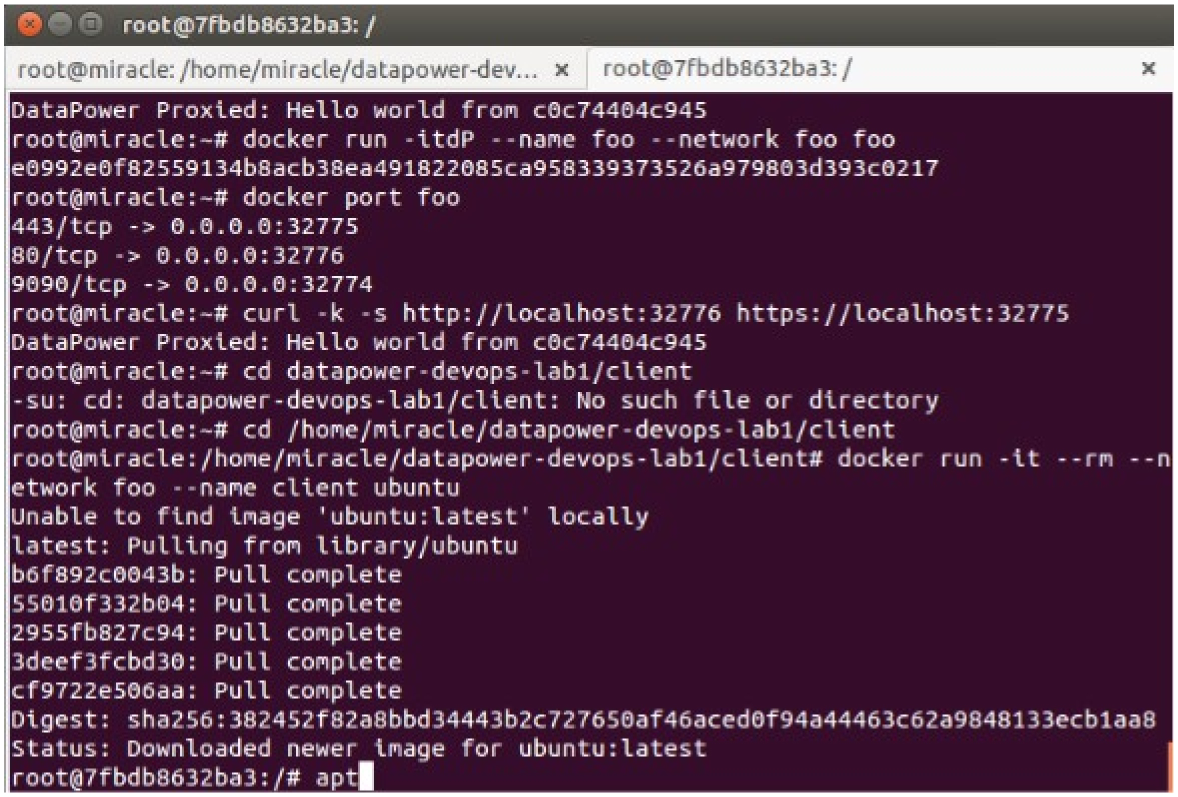
After pointing your browser to http://localhost:9090 there is a need to change the backend url to http://backend:8080.

Test the application using the following commands.

Next let's build the foo image. Remember that we will have to fix permissions since we saved configuration changes in DataPower


root@miracle: /home/miracle/datapower-devops-lab1/datapower
root@miracle: /home/miracle/datapower-dev...
root@miracle: ~

```
root@miracle:/home/miracle/datapower-devops-lab1/datapower# sudo chown --referen
ce=. --recursive  .
root@miracle:/home/miracle/datapower-devops-lab1/datapower# docker build -t foo
.
Sending build context to Docker daemon  46.08kB
Step 1/5 : FROM ibmcom/datapower:latest
 ---> 8252ec9bb88a
Step 2/5 : ENV DATAPOWER_ACCEPT_LICENSE true DATAPOWER_WORKER_THREADS 2 DATAPOWE
R_INTERACTIVE true
 ---> Using cache
 ---> 3eeeb024f125
Step 3/5 : COPY config/ /drouter/config/
 ---> 78c3ac3c3551
Removing intermediate container a61c2124005d
Step 4/5 : COPY local/ /drouter/local/
 ---> 72b9243cd38a
Removing intermediate container 4c04595da7ec
Step 5/5 : EXPOSE 80 443 9090
 ---> Running in e725ce380e52
 ---> e08ef4acd81f
Removing intermediate container e725ce380e52
Successfully built e08ef4acd81f
Successfully tagged foo:latest
root@miracle:/home/miracle/datapower-devops-lab1/datapower# git diff
```



```
DataPower Proxied: Hello world from c0c74404c945
root@miracle:~# docker run -itdP --name foo --network foo foo
e0992e0f82559134b8acb38ea491822085ca958339373526a979803d393c0217
root@miracle:~# docker port foo
443/tcp -> 0.0.0.0:32775
80/tcp -> 0.0.0.0:32776
9090/tcp -> 0.0.0.0:32774
root@miracle:~# curl -k -s http://localhost:32776 https://localhost:32775
DataPower Proxied: Hello world from c0c74404c945
root@miracle:~# cd datapower-devops-lab1/client
-su: cd: datapower-devops-lab1/client: No such file or directory
root@miracle:~# cd /home/miracle/datapower-devops-lab1/client
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo --name client ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
b6f892c0043b: Pull complete
55010f332b04: Pull complete
2955fb827c94: Pull complete
3deef3fcbd30: Pull complete
cf9722e506aa: Pull complete
Digest: sha256:382452f82a8bbd34443b2c727650af46aced0f94a44463c62a9848133ecb1aa8
Status: Downloaded newer image for ubuntu:latest
root@7fbdb8632ba3:/# apt
```

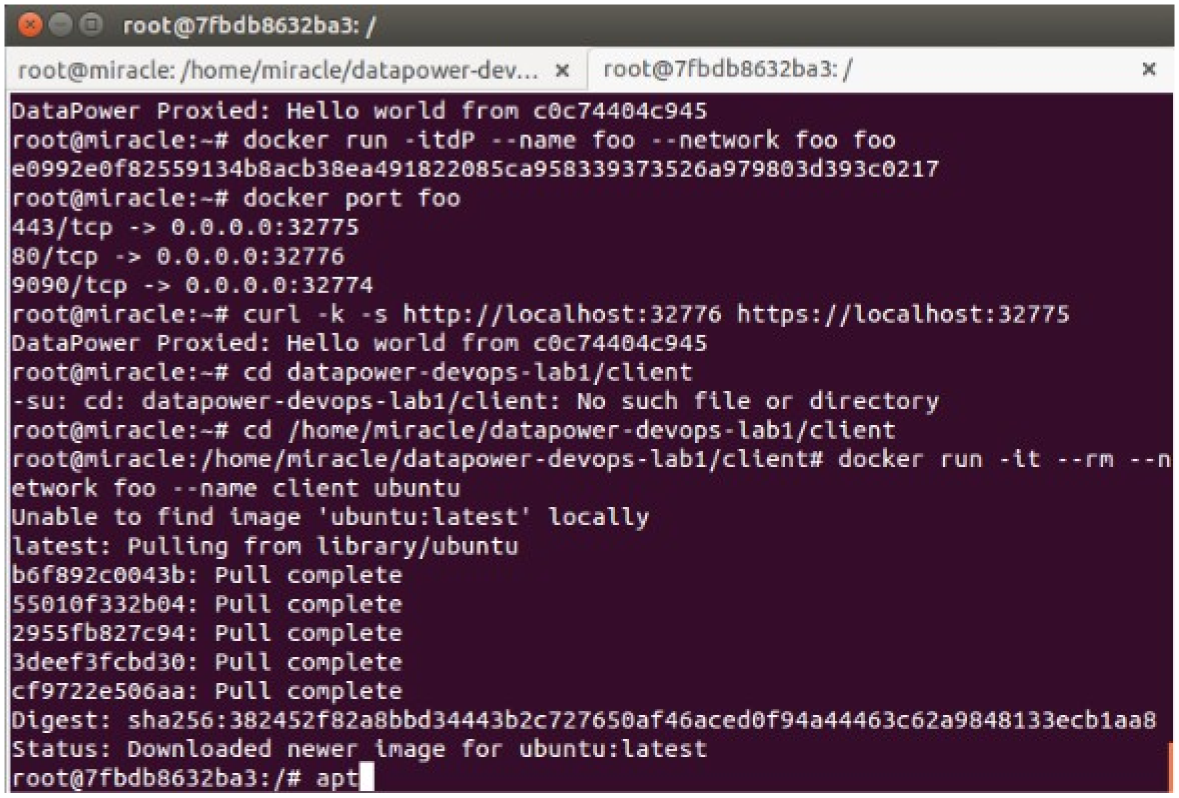
Lastly, run the new container as before, but this time on network foo . Also, we can’t reuse the ports for the development container, so let Docker choose them for us with –P . This means we must look at docker port to see how the ports are mapped and that we must change the URLs in curl to account for the different ports.

Containerize the Client:

Before we containerize the backend, now its time to Containerize the Client.
In production, clients are connecting from all over the world, why do we want it in the container at all?

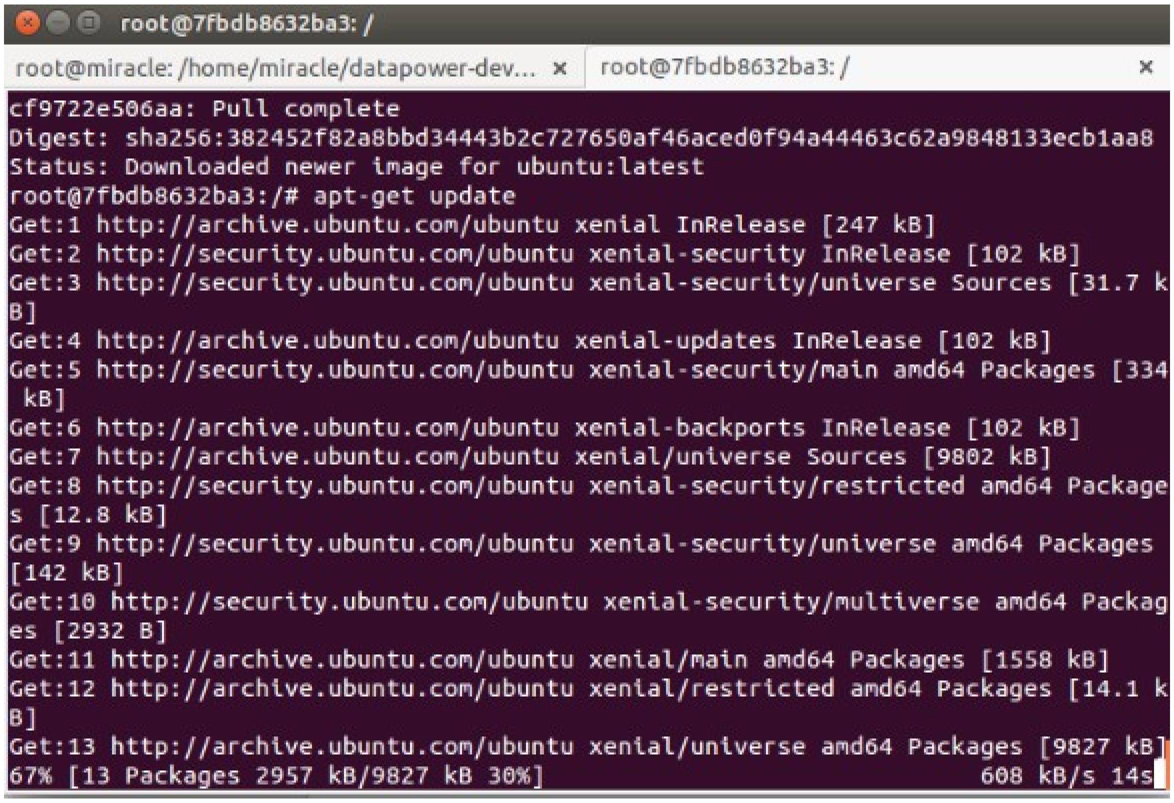
The answer is DevOps. It is Continuous Integration. Let me explain. The goal of building a composed application is that all the required pieces are tied together in a way that they can be deployed together. So far, we have that for the backend and for the Foo DataPower application. But we do not have that for curl . Is curl part of the application? Granted it is not a part that is in line with client data the way DataPower and the backend are. But it is test code, and test code is a deliverable.

First, let's get a container to start trying things out with. We're eventually going to write a Dockerfile to automate this, so we'll start by trying out the steps manually.



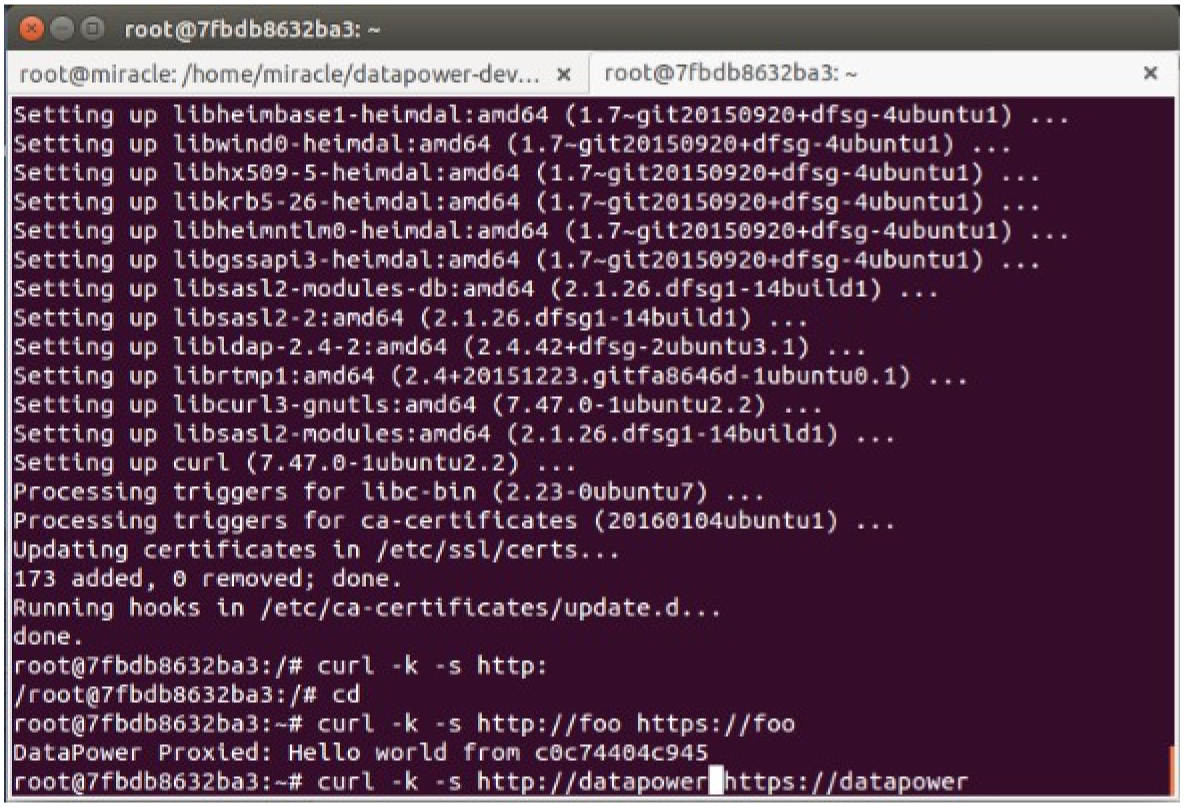
```
DataPower Proxied: Hello world from c0c74404c945
root@miracle:~# docker run -itdP --name foo --network foo foo
e0992e0f82559134b8acb38ea491822085ca958339373526a979803d393c0217
root@miracle:~# docker port foo
443/tcp -> 0.0.0.0:32775
80/tcp -> 0.0.0.0:32776
9090/tcp -> 0.0.0.0:32774
root@miracle:~# curl -k -s http://localhost:32776 https://localhost:32775
DataPower Proxied: Hello world from c0c74404c945
root@miracle:~# cd datapower-devops-lab1/client
-su: cd: datapower-devops-lab1/client: No such file or directory
root@miracle:~# cd /home/miracle/datapower-devops-lab1/client
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo --name client ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
b6f892c0043b: Pull complete
55010f332b04: Pull complete
2955fb827c94: Pull complete
3deef3fcbd30: Pull complete
cf9722e506aa: Pull complete
Digest: sha256:382452f82a8bbd34443b2c727650af46aced0f94a44463c62a9848133ecb1aa8
Status: Downloaded newer image for ubuntu:latest
root@7fbdb8632ba3:/# apt
```

Now to get cURL and try it out. We left both the datapower and foo containers running in the previous step, so we have some built-in choices



```
cf9722e506aa: Pull complete
Digest: sha256:382452f82a8bbd34443b2c727650af46aced0f94a44463c62a9848133ecb1aa8
Status: Downloaded newer image for ubuntu:latest
root@7fbdb8632ba3:/# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [31.7 k
B]
Get:4 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [334
 kB]
Get:6 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:7 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Package
s [12.8 kB]
Get:9 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages
[142 kB]
Get:10 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packag
es [2932 B]
Get:11 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 k
B]
Get:13 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
67% [13 Packages 2957 kB/9827 kB 30%]                              608 kB/s 14s
```

Now we know what we must put into the Dockerfile . So first open Dockerfile in an editor



```
Setting up libheimbase1-heimdal:amd64 (1.7~git20150920+dfsg-4ubuntu1) ...
Setting up libwind0-heimdal:amd64 (1.7~git20150920+dfsg-4ubuntu1) ...
Setting up libhx509-5-heimdal:amd64 (1.7~git20150920+dfsg-4ubuntu1) ...
Setting up libkrb5-26-heimdal:amd64 (1.7~git20150920+dfsg-4ubuntu1) ...
Setting up libheimntlm0-heimdal:amd64 (1.7~git20150920+dfsg-4ubuntu1) ...
Setting up libgssapi3-heimdal:amd64 (1.7~git20150920+dfsg-4ubuntu1) ...
Setting up libsasl2-modules-db:amd64 (2.1.26.dfsg1-14build1) ...
Setting up libsasl2-2:amd64 (2.1.26.dfsg1-14build1) ...
Setting up libldap-2.4-2:amd64 (2.4.42+dfsg-2ubuntu3.1) ...
Setting up librtmp1:amd64 (2.4+20151223.gitfa8646d-1ubuntu0.1) ...
Setting up libcurl3-gnutls:amd64 (7.47.0-1ubuntu2.2) ...
Setting up libsasl2-modules:amd64 (2.1.26.dfsg1-14build1) ...
Setting up curl (7.47.0-1ubuntu2.2) ...
Processing triggers for libc-bin (2.23-0ubuntu7) ...
Processing triggers for ca-certificates (20160104ubuntu1) ...
Updating certificates in /etc/ssl/certs...
173 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
root@7fbdb8632ba3:/# curl -k -s http:
/root@7fbdb8632ba3:/# cd
root@7fbdb8632ba3:~# curl -k -s http://foo https://foo
DataPower Proxied: Hello world from c0c74404c945
root@7fbdb8632ba3:~# curl -k -s http://datapower https://datapower
```

Now type the command **atom dockerfile.** Then add the text as below.

```
FROM ubuntu:latest
RUN  apt-get update && \
     apt-get -y install curl
CMD  ["curl","-k","-s","http://datapower","https://datapower"]
```

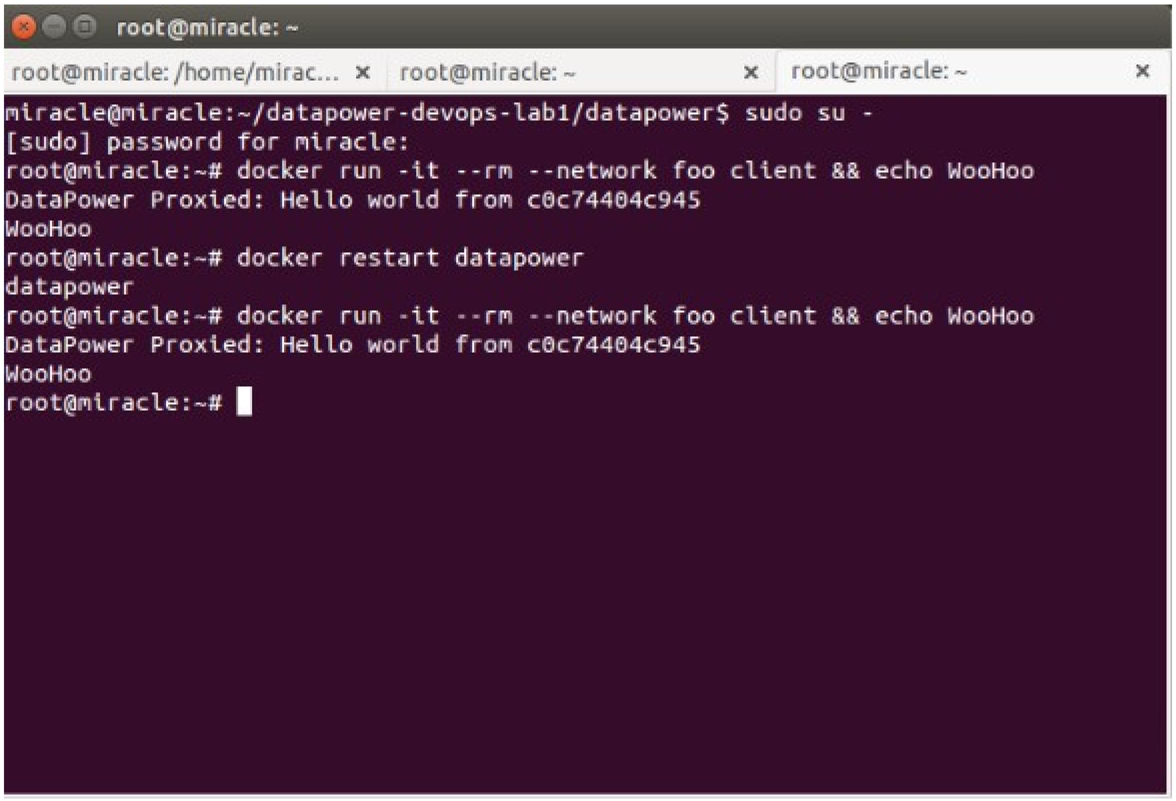
Build the client image:

localuser@ubuntu-base:~/datapower-devops-lab/client$ docker build -t client .
Sending build context to Docker daemon 3.584 kB
Step 1/3 : FROM ubuntu:latest
---> f49eec89601e
Step 2/3 : RUN apt-get update &&

apt-get install -y curl
---> Using cache
---> 4edd1295ea7a
Step 3/3 : CMD /usr/bin/curl -k -s http://datapower https://datapower
---> Using cache
---> 0c0cbf38822c
Successfully built 0c0cbf38822c

Check it by using commands

But we don't yet know if the test succeeds in all circumstances. Let's try it while restarting the datapower container



```
miracle@miracle:~/datapower-devops-lab1/datapower$ sudo su -
[sudo] password for miracle:
root@miracle:~# docker run -it --rm --network foo client && echo WooHoo
DataPower Proxied: Hello world from c0c74404c945
WooHoo
root@miracle:~# docker restart datapower
datapower
root@miracle:~# docker run -it --rm --network foo client && echo WooHoo
DataPower Proxied: Hello world from c0c74404c945
WooHoo
root@miracle:~#
```

The good news is that there are no success messages on failures. The bad news is that the client should retry enough that the test won't incorrectly fail in a startup situation.

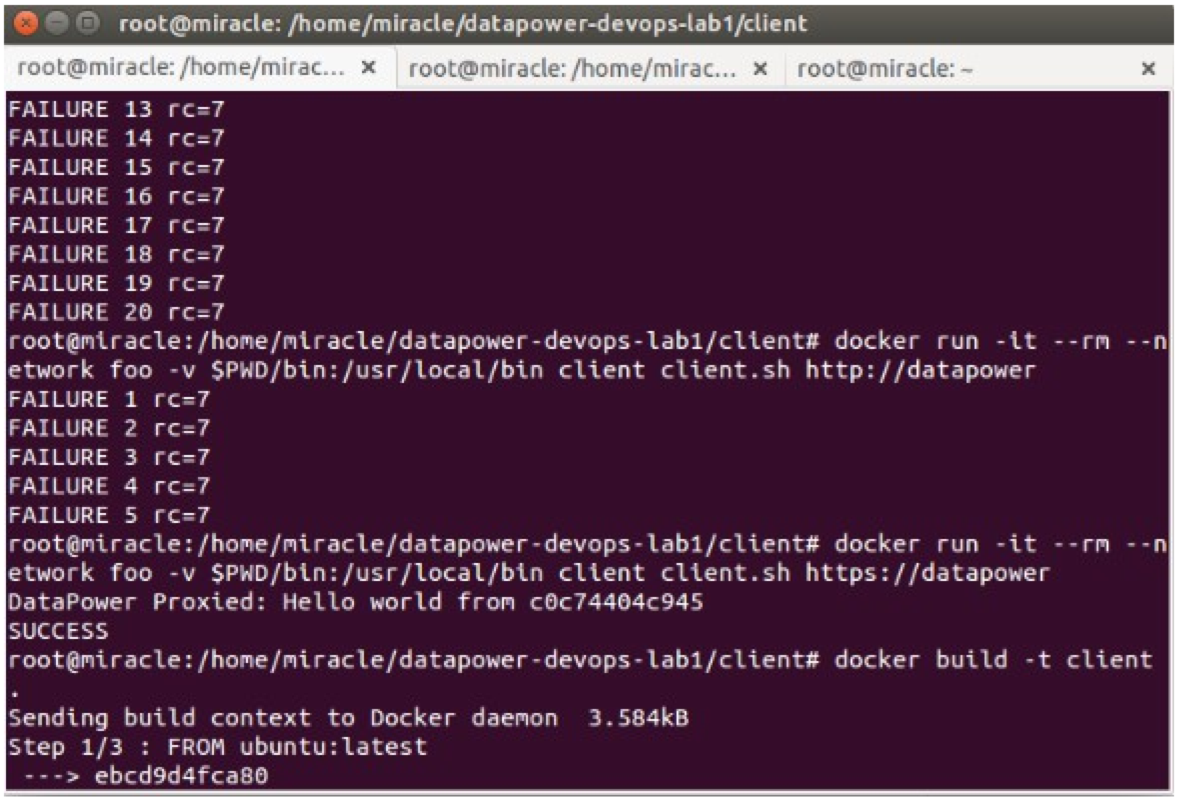
This is a job for a script!

```
localuser@ubuntu-base:~/datapower-devops-lab/client$ mkdir -p bin
localuser@ubuntu-base:~/datapower-devops-lab/client$ touch bin/client.sh
localuser@ubuntu-base:~/datapower-devops-lab/client$ \
chmod a+x bin/client.sh
localuser@ubuntu-base:~/datapower-devops-lab/client$ atom bin/client.sh
```

Inside the script put,

```
#!/bin/bash

# Loop for up to 20 consecutive failures
for (( failures=0; failures<20; ))
do
  # Issue the curl command with urls from script args
  curl -k -s "$@"
  # save the exit code of curl for use in comparison and exit
  rc=$?
  if [ $rc = 0 ]
  then
    failures=0
    if [ ! "$CONTINUOUS" = "true" ]
    then
      echo SUCCESS
      break
    fi
  else
    failures=$((failures + 1))
    echo FAILURE $failures rc=$rc
  fi
  # Otherwise wait a second and try again
  sleep 5
done
```

Now we have to test it. Use the commands as shown below.

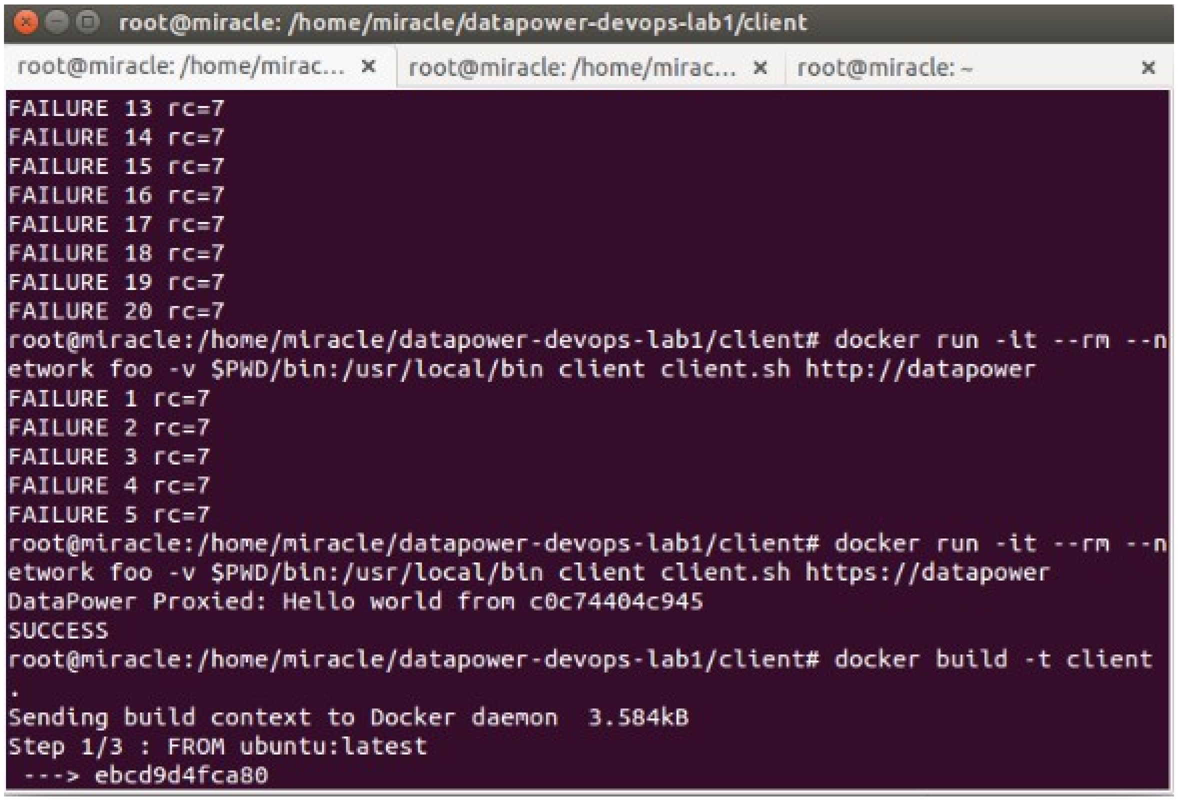


root@miracle: /home/miracle/datapower-devops-lab1/client

root@miracle: /home/mirac... | root@miracle: /home/mirac... | root@miracle: ~

```
FAILURE 13 rc=7
FAILURE 14 rc=7
FAILURE 15 rc=7
FAILURE 16 rc=7
FAILURE 17 rc=7
FAILURE 18 rc=7
FAILURE 19 rc=7
FAILURE 20 rc=7
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo -v $PWD/bin:/usr/local/bin client client.sh http://datapower
FAILURE 1 rc=7
FAILURE 2 rc=7
FAILURE 3 rc=7
FAILURE 4 rc=7
FAILURE 5 rc=7
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo -v $PWD/bin:/usr/local/bin client client.sh https://datapower
DataPower Proxied: Hello world from c0c74404c945
SUCCESS
root@miracle:/home/miracle/datapower-devops-lab1/client# docker build -t client
.
Sending build context to Docker daemon  3.584kB
Step 1/3 : FROM ubuntu:latest
 ---> ebcd9d4fca80
```

- Great, we can see that it retries then exits on first success. Now that it’s up, make sure that it exits with success immediately.
- After that we want to start the docker build client as shown above.


root@miracle: /home/miracle/datapower-devops-lab1/client
root@miracle: /home/mirac...
root@miracle: /home/mirac...
root@miracle: ~

```
FAILURE 13 rc=7
FAILURE 14 rc=7
FAILURE 15 rc=7
FAILURE 16 rc=7
FAILURE 17 rc=7
FAILURE 18 rc=7
FAILURE 19 rc=7
FAILURE 20 rc=7
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo -v $PWD/bin:/usr/local/bin client client.sh http://datapower
FAILURE 1 rc=7
FAILURE 2 rc=7
FAILURE 3 rc=7
FAILURE 4 rc=7
FAILURE 5 rc=7
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo -v $PWD/bin:/usr/local/bin client client.sh https://datapower
DataPower Proxied: Hello world from c0c74404c945
SUCCESS
root@miracle:/home/miracle/datapower-devops-lab1/client# docker build -t client
.
Sending build context to Docker daemon  3.584kB
Step 1/3 : FROM ubuntu:latest
 ---> ebcd9d4fca80
```

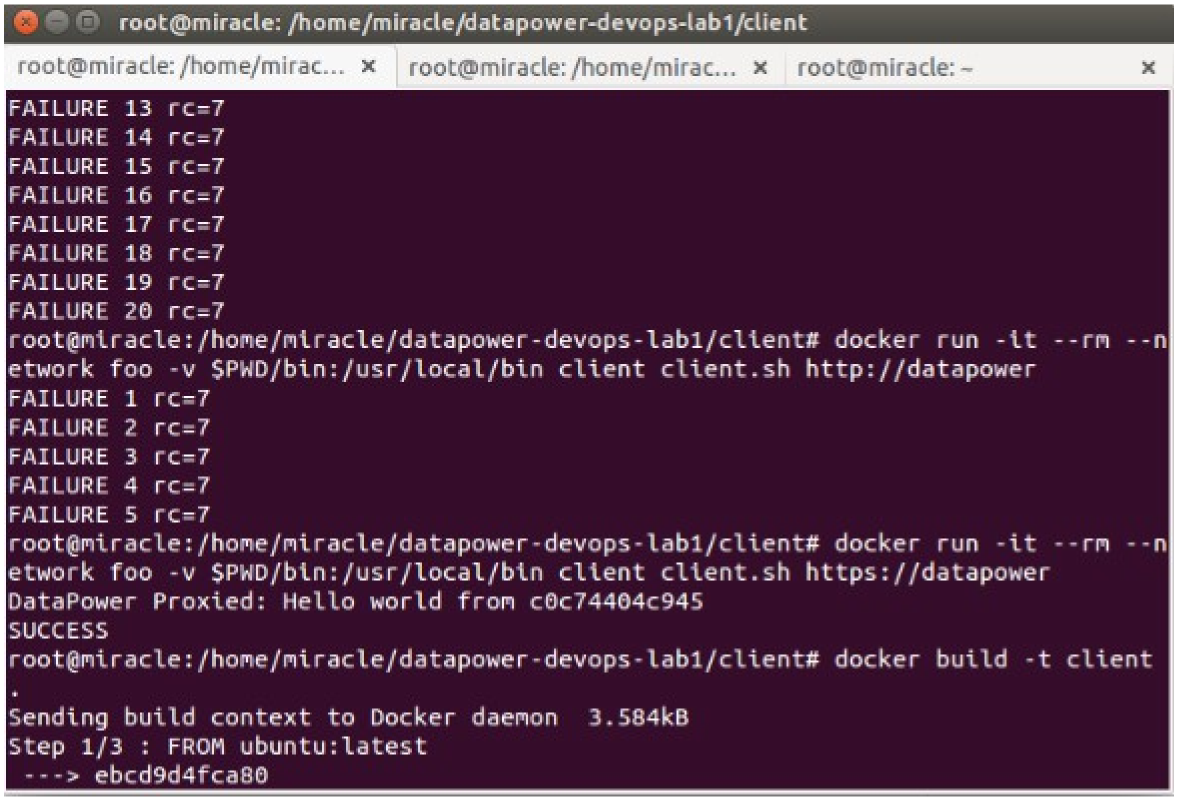
Cleaning Up :

Since we don't need our existing containers any longer, remove them with the **command**

docker ps -aq | xargs docker rm –f

Since the fix is to add the missing -f option in client.sh , we make that change, test it, then check in the fix:

1. From the client directory, run the atom bin/client.sh command.
2. Add the missing -f option to curl and save the client.sh script.
3. Build the client image with the **docker build -t client .** command.



root@miracle: /home/miracle/datapower-devops-lab1/client

root@miracle: /home/mirac... × root@miracle: /home/mirac... × root@miracle: ~ ×

```
FAILURE 13 rc=7
FAILURE 14 rc=7
FAILURE 15 rc=7
FAILURE 16 rc=7
FAILURE 17 rc=7
FAILURE 18 rc=7
FAILURE 19 rc=7
FAILURE 20 rc=7
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo -v $PWD/bin:/usr/local/bin client client.sh http://datapower
FAILURE 1 rc=7
FAILURE 2 rc=7
FAILURE 3 rc=7
FAILURE 4 rc=7
FAILURE 5 rc=7
root@miracle:/home/miracle/datapower-devops-lab1/client# docker run -it --rm --n
etwork foo -v $PWD/bin:/usr/local/bin client client.sh https://datapower
DataPower Proxied: Hello world from c0c74404c945
SUCCESS
root@miracle:/home/miracle/datapower-devops-lab1/client# docker build -t client
.
Sending build context to Docker daemon  3.584kB
Step 1/3 : FROM ubuntu:latest
 ---> ebcd9d4fca80
```

4. Test that it works correctly under failure with the docker run --rm -- network foo client ; echo $? command.
5. Test that it works correctly under success by starting the backend with the docker run -d --name backend --network foo hstenzel/nodejs-hostname-automatic ; echo $? command.
6. Check in with the git add . && git commit -m 'Add --fail option to curl to fix false success bug' command.

Composed Testing :

The docker-compose.yml file describes a composed application in a way that Docker Compose can manage it. There are a ton of great resources available about Docker Compose and docker-compose.yml.

- Now go to datapower-devops-lab folder and open a test editor to write the docker-compose script in it.
- Then add the script like this.

```
version: '3'
services:
  client:
    build: client
    depends_on:
      - datapower
  datapower:
    build: datapower
    depends_on:
      - backend
  backend:
    image: hstenzel/nodejs-hostname-automatic
```

- First, notice that we have defined three different services, client , datapower , and backend . These are the three images that we run as containers that make up our composed application.
- We also give Compose dependency knowledge. We say that datapower depends on backend and client depends on datapower because it does not make sense for this application to have a datapower container unless there is a backend container to support it.

We need to build and pull the Docker-compose file to start the services.

Use the commands in the root folder as

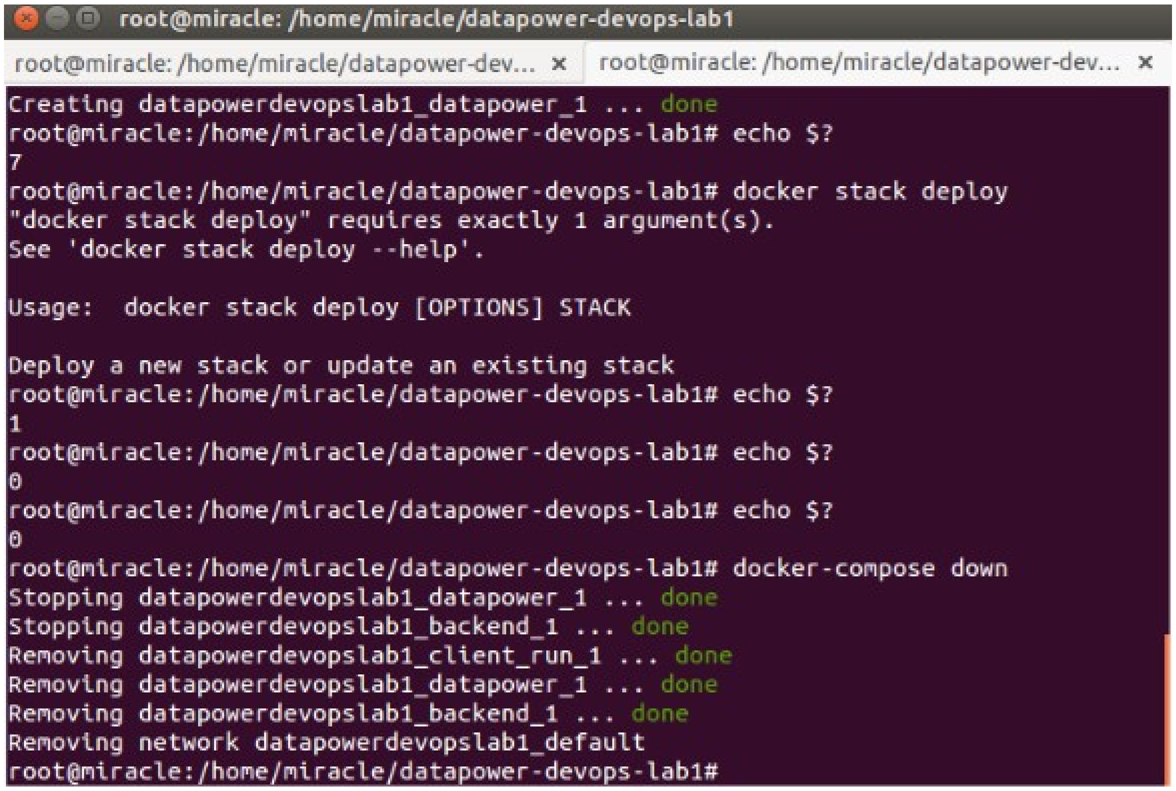
- **docker-compose build**
- **docker-compose pull**

Now if you want to run the client or datapower or backend give the command like

docker-compose run client

After giving this it should prompt like starting the datapower and backend services, because in the docker-compose file we had written client depends on datapower and datapower depends on backend.

Now stop the docker-compose. When you buils it, it is in running state.



root@miracle: /home/miracle/datapower-devops-lab1
root@miracle: /home/miracle/datapower-dev...
root@miracle: /home/miracle/datapower-dev...

```
Creating datapowerdevopslab1_datapower_1 ... done
root@miracle:/home/miracle/datapower-devops-lab1# echo $?
7
root@miracle:/home/miracle/datapower-devops-lab1# docker stack deploy
"docker stack deploy" requires exactly 1 argument(s).
See 'docker stack deploy --help'.

Usage:  docker stack deploy [OPTIONS] STACK

Deploy a new stack or update an existing stack
root@miracle:/home/miracle/datapower-devops-lab1# echo $?
1
root@miracle:/home/miracle/datapower-devops-lab1# echo $?
0
root@miracle:/home/miracle/datapower-devops-lab1# echo $?
0
root@miracle:/home/miracle/datapower-devops-lab1# docker-compose down
Stopping datapowerdevopslab1_datapower_1 ... done
Stopping datapowerdevopslab1_backend_1 ... done
Removing datapowerdevopslab1_client_run_1 ... done
Removing datapowerdevopslab1_datapower_1 ... done
Removing datapowerdevopslab1_backend_1 ... done
Removing network datapowerdevopslab1_default
root@miracle:/home/miracle/datapower-devops-lab1#
```

Composed Development

While the developer will use the composed testing process, that alone is not enough. There still must exist some way to access web management, some way that DataPower configuration can be saved to version control, and some way that allows easy testing of changes to the service.

So now we have docker-compose file but we don't have an environment to run it.

So write the script inside the docker-compose-dev.yml

```
version: '3'
services:
  client:
    build: client
    depends_on:
      - datapower
    environment:
      - CONTINUOUS=true
  datapower:
    build: datapower
    depends_on:
      - backend
    volumes:
      - ./datapower/config:/drouter/config
      - ./datapower/local:/drouter/local
    ports:
      - "80:80"
      - "443:443"
      - "9090:9090"
  backend:
    image: hstenzel/nodejs-hostname-automatic
```

Now using the docker-compose-dev run the command,

docker-compose -f docker-compose-dev.yml up

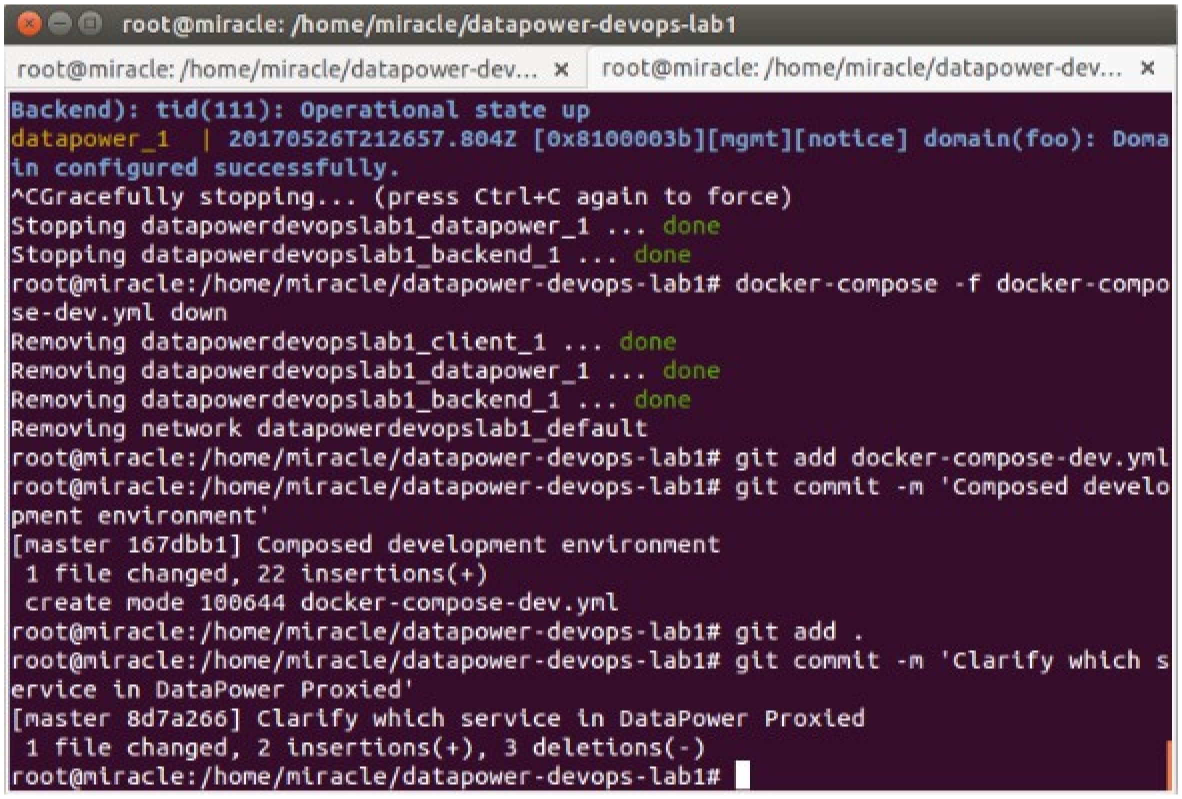
After running the command we can have all the containers available are running and we can also have a datapower UI in the web-browser.

After having this we can check whether the continous development is working or not. Open hellotoo.js in the datapower folder and make some changes in the application.

We can observe the changes in the service that done automatically in the datapower web UI.

For now stop the development environment using the command

docker-compose -f docker-compose-dev.yml down


root@miracle: /home/miracle/datapower-devops-lab1
root@miracle: /home/miracle/datapower-dev...
root@miracle: /home/miracle/datapower-dev...

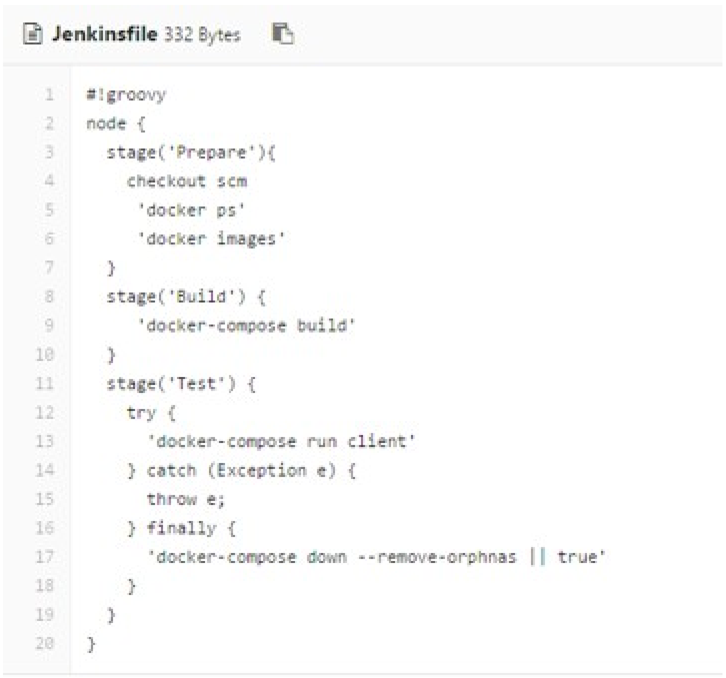
```
Backend): tid(111): Operational state up
datapower_1  | 20170526T212657.804Z [0x8100003b][mgmt][notice] domain(foo): Doma
in configured successfully.
^CGracefully stopping... (press Ctrl+C again to force)
Stopping datapowerdevopslab1_datapower_1 ... done
Stopping datapowerdevopslab1_backend_1 ... done
root@miracle:/home/miracle/datapower-devops-lab1# docker-compose -f docker-compo
se-dev.yml down
Removing datapowerdevopslab1_client_1 ... done
Removing datapowerdevopslab1_datapower_1 ... done
Removing datapowerdevopslab1_backend_1 ... done
Removing network datapowerdevopslab1_default
root@miracle:/home/miracle/datapower-devops-lab1# git add docker-compose-dev.yml
root@miracle:/home/miracle/datapower-devops-lab1# git commit -m 'Composed develo
pment environment'
[master 167dbb1] Composed development environment
 1 file changed, 22 insertions(+)
 create mode 100644 docker-compose-dev.yml
root@miracle:/home/miracle/datapower-devops-lab1# git add .
root@miracle:/home/miracle/datapower-devops-lab1# git commit -m 'Clarify which s
ervice in DataPower Proxied'
[master 8d7a266] Clarify which service in DataPower Proxied
 1 file changed, 2 insertions(+), 3 deletions(-)
root@miracle:/home/miracle/datapower-devops-lab1# 
```

Now the continuous development is done . Its time for continuous Integration. Now we need to configure Gitlab and jenkins in docker for the setup.

Add jenkins integration:

The Jenkins integration will be checked into version control at the root of the

repository, and the file is called Jenkinsfile . Let's create our Jenkinsfile.



```
#!groovy
node {
  stage('Prepare'){
    checkout scm
     'docker ps'
     'docker images'
  }
  stage('Build') {
     'docker-compose build'
  }
  stage('Test') {
    try {
      'docker-compose run client'
    } catch (Exception e) {
      throw e;
    } finally {
      'docker-compose down --remove-orphnas || true'
    }
  }
}
```

Configure Gitlab in docker:

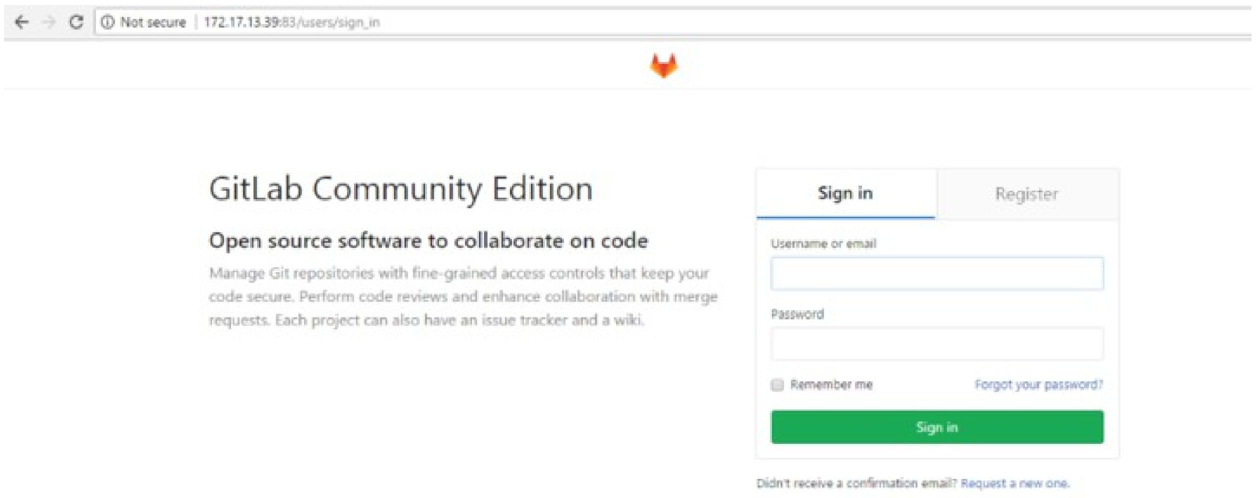
To install gitlab in docker we have to install the openssh-server and curl .

Now run the command like this

docker run -d -p 443:443 -p 83:80 -p 23:22 –name gitlab gitlab/gitlab-ce:latest

Now open the browser and point to localhost:83. You will see a login page there you can set a password. The username is root.

1. Now login to gitlab and create a new project.
2. Name it as datapower-devops-lab under public.



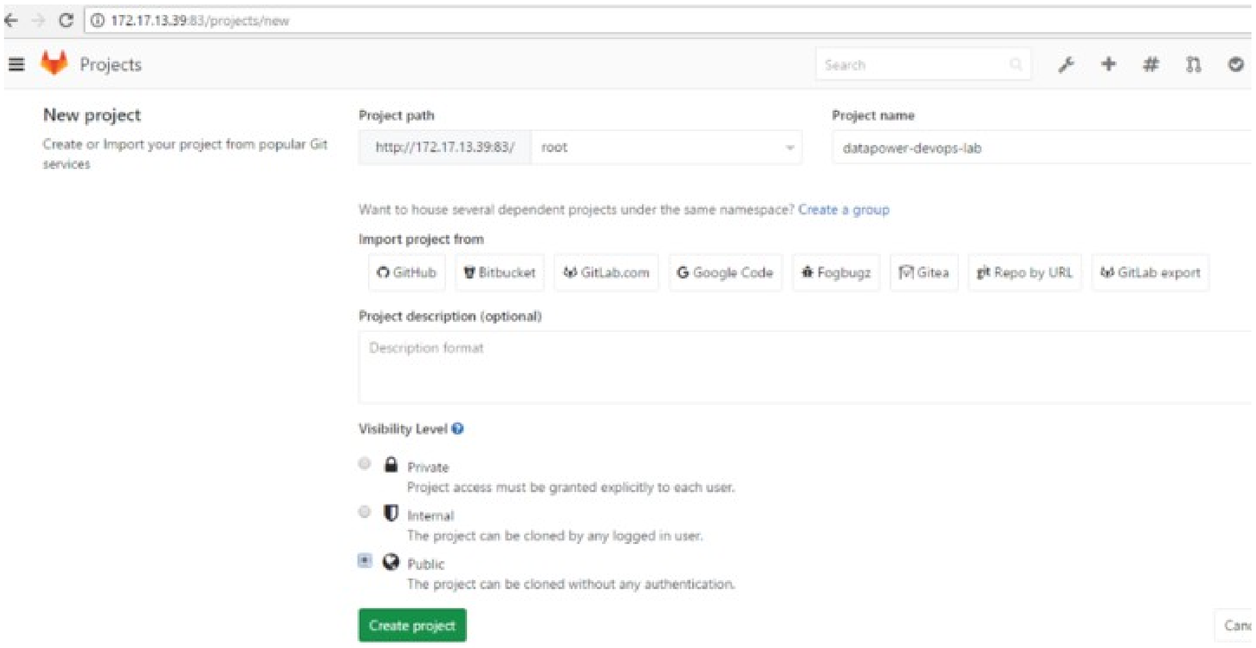
Not secure | 172.17.13.39:83/users/sign_in

GitLab Community Edition

Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Didn't receive a confirmation email? Request a new one.



172.17.13.39:83/projects/new

New project

Create or Import your project from popular Git services

Project path: http://172.17.13.39:83/ root

Project name: datapower-devops-lab

Want to house several dependent projects under the same namespace? Create a group

Import project from

Project description (optional)

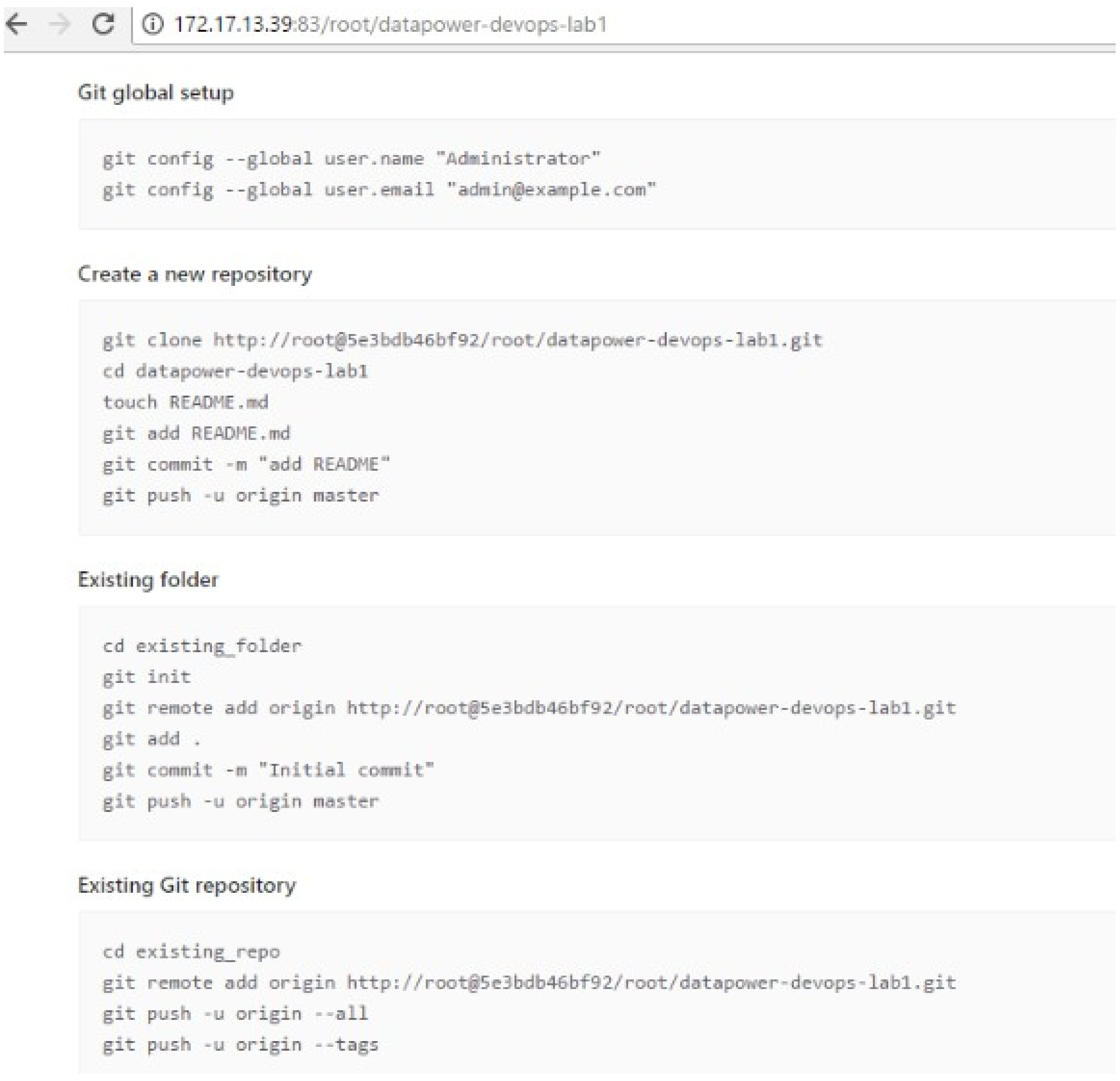
Visibility Level

Private
Project access must be granted explicitly to each user.

Internal
The project can be cloned by any logged in user.

[✓ Public]
The project can be cloned without any authentication.

Now we created a empty project and we have the list of commands to perform.



Git global setup

```
git config --global user.name "Administrator"
git config --global user.email "admin@example.com"
```

Create a new repository

```
git clone http://root@5e3bdb46bf92/root/datapower-devops-lab1.git
cd datapower-devops-lab1
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

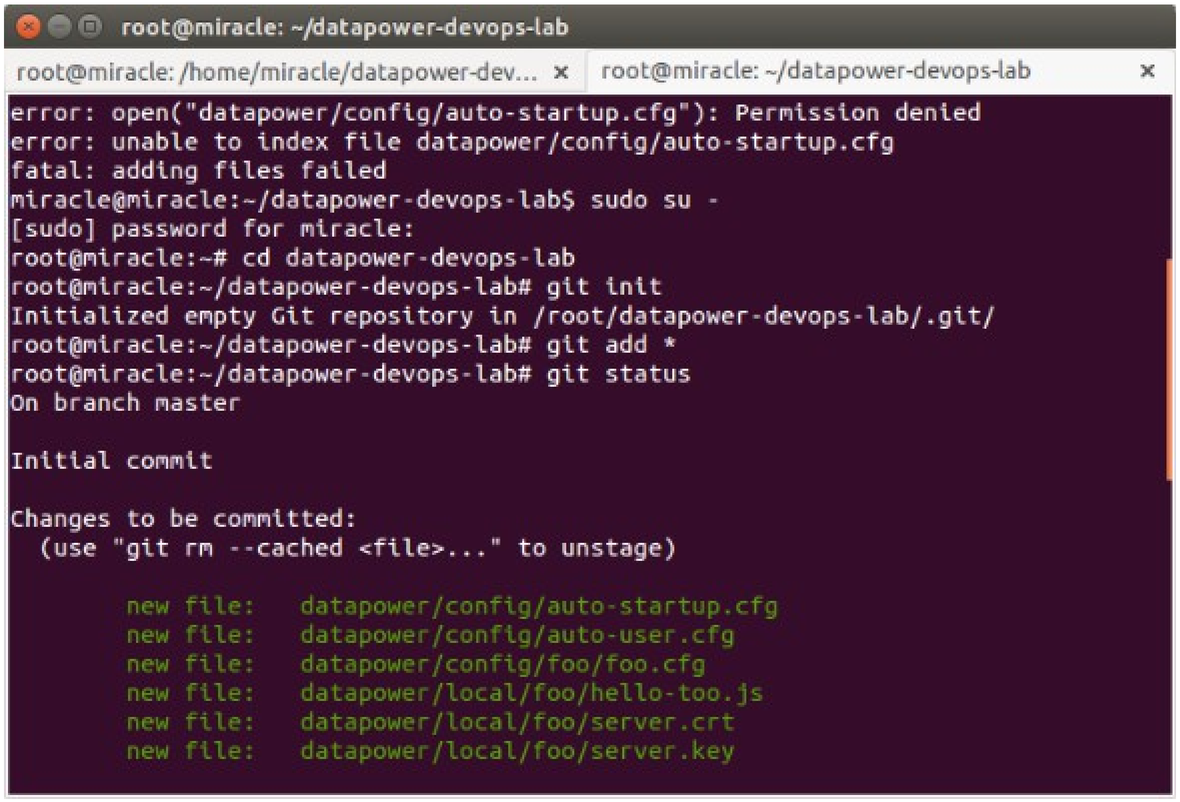
Existing folder

```
cd existing_folder
git init
git remote add origin http://root@5e3bdb46bf92/root/datapower-devops-lab1.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

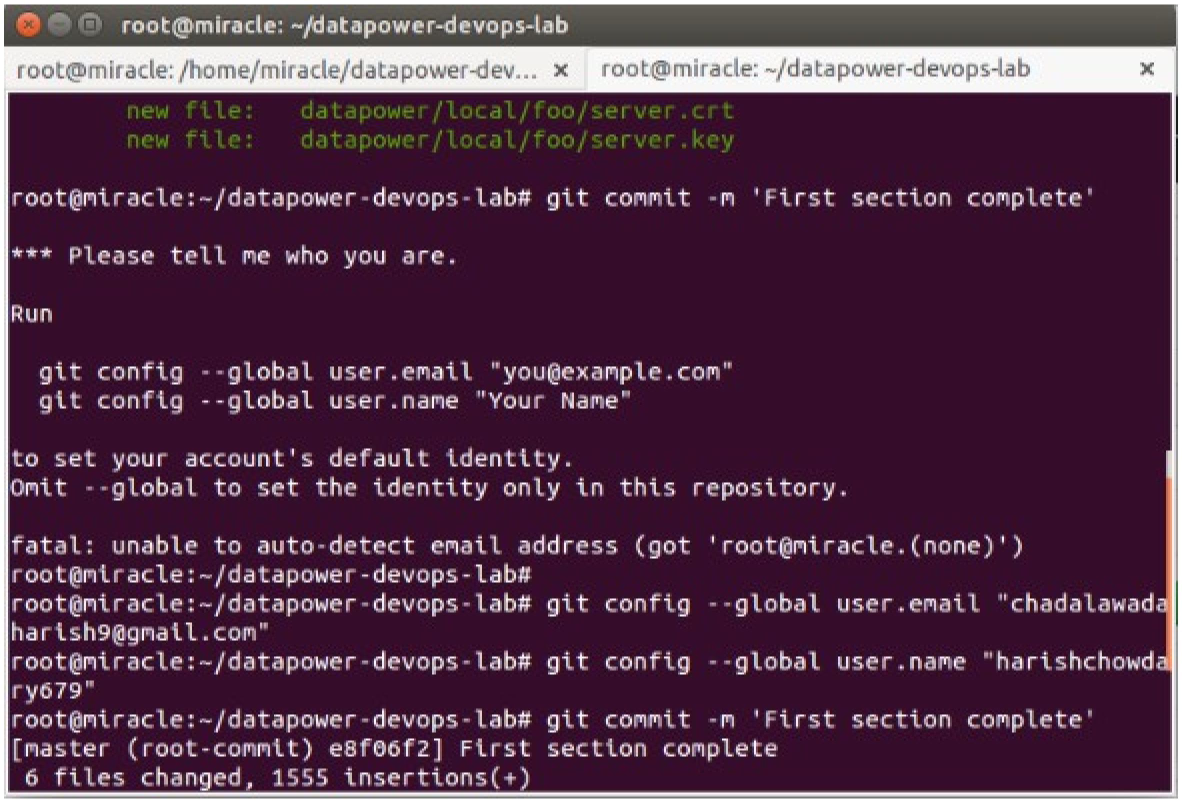
Existing Git repository

```
cd existing_repo
git remote add origin http://root@5e3bdb46bf92/root/datapower-devops-lab1.git
git push -u origin --all
git push -u origin --tags
```

Here we have already having a folder. So execute the commands under Existing folder.


root@miracle: ~/datapower-devops-lab
root@miracle: /home/miracle/datapower-dev...
root@miracle: ~/datapower-devops-lab

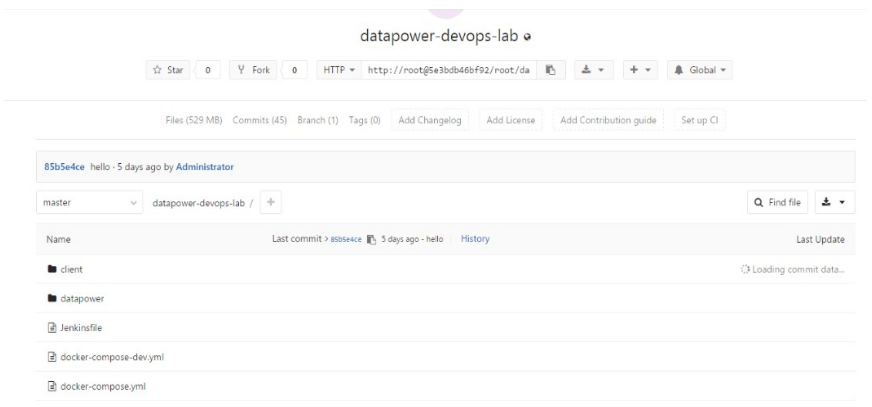
```
error: open("datapower/config/auto-startup.cfg"): Permission denied
error: unable to index file datapower/config/auto-startup.cfg
fatal: adding files failed
miracle@miracle:~/datapower-devops-lab$ sudo su -
[sudo] password for miracle:
root@miracle:~# cd datapower-devops-lab
root@miracle:~/datapower-devops-lab# git init
Initialized empty Git repository in /root/datapower-devops-lab/.git/
root@miracle:~/datapower-devops-lab# git add *
root@miracle:~/datapower-devops-lab# git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   datapower/config/auto-startup.cfg
        new file:   datapower/config/auto-user.cfg
        new file:   datapower/config/foo/foo.cfg
        new file:   datapower/local/foo/hello-too.js
        new file:   datapower/local/foo/server.crt
        new file:   datapower/local/foo/server.key
```


root@miracle: ~/datapower-devops-lab
root@miracle: /home/miracle/datapower-dev...
root@miracle: ~/datapower-devops-lab

```
        new file:   datapower/local/foo/server.crt
        new file:   datapower/local/foo/server.key

root@miracle:~/datapower-devops-lab# git commit -m 'First section complete'

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'root@miracle.(none)')
root@miracle:~/datapower-devops-lab#
root@miracle:~/datapower-devops-lab# git config --global user.email "chadalawada
harish9@gmail.com"
root@miracle:~/datapower-devops-lab# git config --global user.name "harishchowda
ry679"
root@miracle:~/datapower-devops-lab# git commit -m 'First section complete'
[master (root-commit) e8f06f2] First section complete
 6 files changed, 1555 insertions(+)
```

Now we have committed the changes in the folder. Push the application into gitlab using **git push -u origin**

After push the application we can check in Git lab whether the folder structure was created or not.



http://root@5e3bdb46bf92/root/da

85b5e4ce hello · 5 days ago by Administrator

master datapower-devops-lab /

Name	Last commit > 85b5e4ce 5 days ago - hello History	Last Update
client		Loading commit data...
datapower		
Jenkinsfile		
docker-compose-dev.yml		
docker-compose.yml		

Now its time for continuous integration:

- Jenkins is a powerful application that allows "continuous integration" and "continuous delivery" of projects.

Jenkins is a Software that allows continuous integration and continuous delivery of projects, regardless of the platform.

We can integrate Jenkins with a number of testing and deployment technologies.

2. Why Jenkins:

- Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place.

3. Continuous Integration:

- Continuous Integration is a development practice that requires developers to integrate code into a shared repository at several times per day. Continuous Integration improves Software Quality and Reduces the Risk.
- Committing code frequently.
- Categorizing developer tests.
- Using a dedicated integration build machine.
- Using continuous feedback mechanisms.
- Staging builds.

8. Jenkins Installation Steps:

- For Installing Jenkins in Windows we have to follow the below steps.
- Download the war file from the Jenkins website(Let the war file name be "jenkins.war")
- Now, go to the location where we have the war file and run it using the followinf command.
- Java -jar jenkins.war

9. Creating user in Jenkins:

- So now Jenkins is installed, updated and it is in running state.
- Go to browser and give "http://localhost:8080".
- Give your details as shown in the following screen shot.
- And click save and finish.
- Now you can see the Jenkins screen as shown below.
- Click "start using Jenkins.

Fig 2. Login to Jenkins

10. Adding required Plugins in Jenkins:

- For Integrating Git Hub with Jenkins install all the required plugins in Jenkins like
    - github plugin
    - git plugin

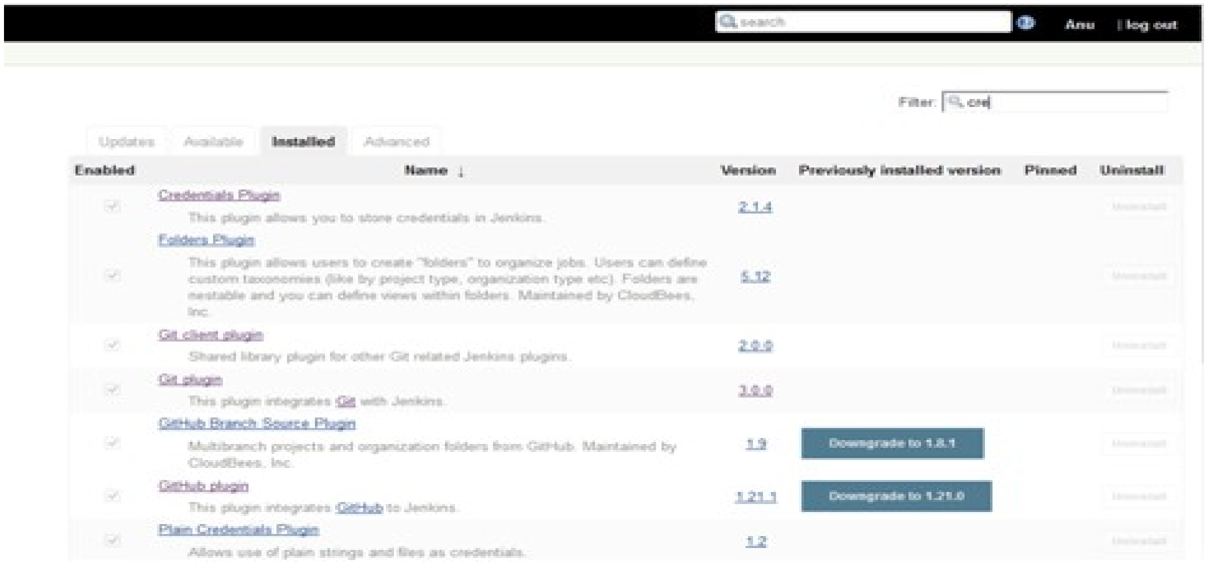
GitHub Plugin:

- This plugins is mainly used for integrates Jenkins with Git Hub project.

- For installing the required plugins go to
- "Manage Jenkins (in the left navigation menu) --> Manage plugins".

Fig 3. Adding Plugins

- Search for the required plugin to be installed in Available tab. If the plugin is already installed it was in Installed tab.



search | Anu | log out

Filter: cre

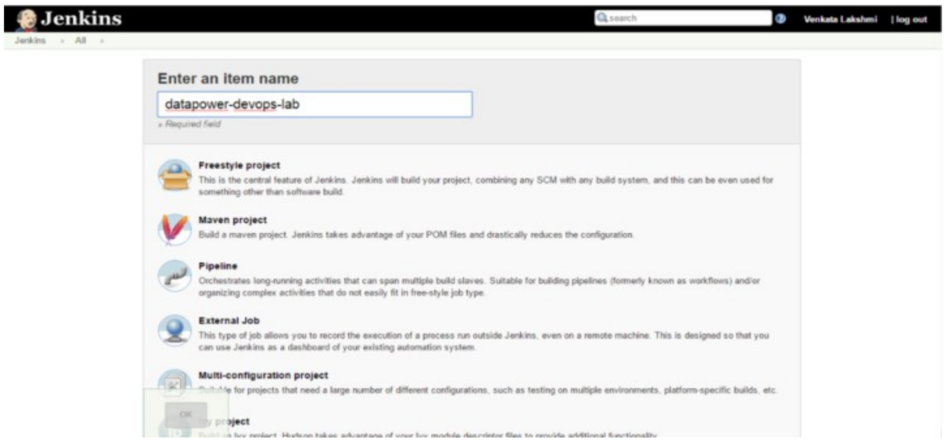
Updates | Available | **Installed** | Advanced

Enabled	Name ↓	Version	Previously installed version	Pinned	Uninstall
✓	Credentials Plugin This plugin allows you to store credentials in Jenkins.	2.1.4			Uninstall
✓	Folders Plugin This plugin allows users to create "folders" to organize jobs. Users can define custom taxonomies (like by project type, organization type etc). Folders are nestable and you can define views within folders. Maintained by CloudBees, Inc.	5.12			Uninstall
✓	Git client plugin Shared library plugin for other Git related Jenkins plugins.	2.0.0			Uninstall
✓	Git plugin This plugin integrates Git with Jenkins.	3.0.0			Uninstall
✓	GitHub Branch Source Plugin Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc.	1.9	Downgrade to 1.8.1		Uninstall
✓	GitHub plugin This plugin integrates GitHub to Jenkins.	1.21.1	Downgrade to 1.21.0		Uninstall
✓	Plain Credentials Plugin Allows use of plain strings and files as credentials.	1.2			Uninstall

Fig 4. Installed Plugins

11. Creating Job:

- Click on new item and give the name then select a Multibranch pipeline with defaults and click on Ok.



Jenkins | search | Venkata Lakshmi | log out

Jenkins › All ›

Enter an item name

datapower-devops-lab

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

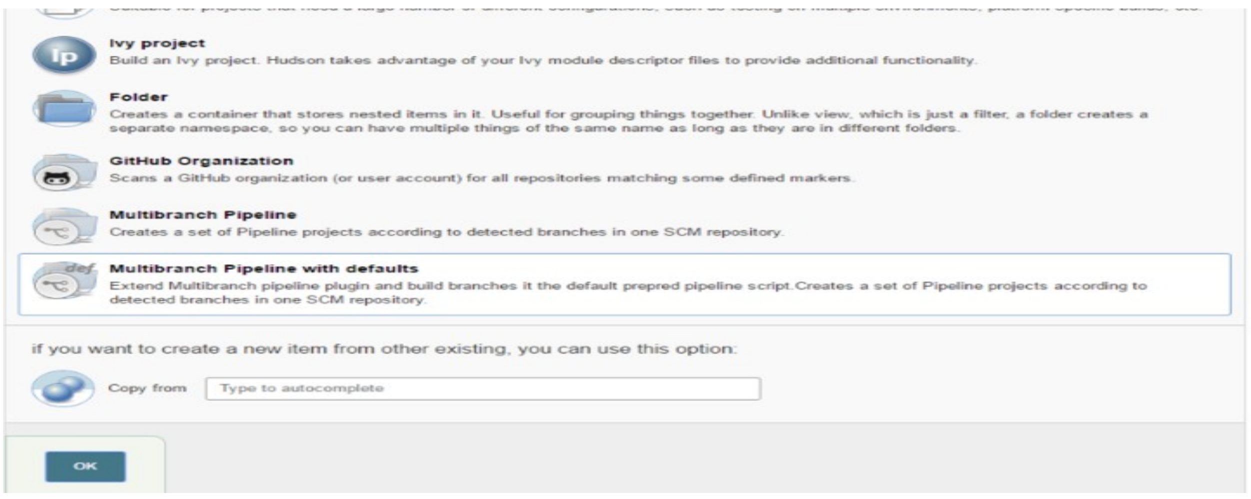
Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

OK

Ivy project
Build an Ivy project. Hudson takes advantage of your Ivy module descriptor files to provide additional functionality.



Ivy project
Build an Ivy project. Hudson takes advantage of your Ivy module descriptor files to provide additional functionality.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

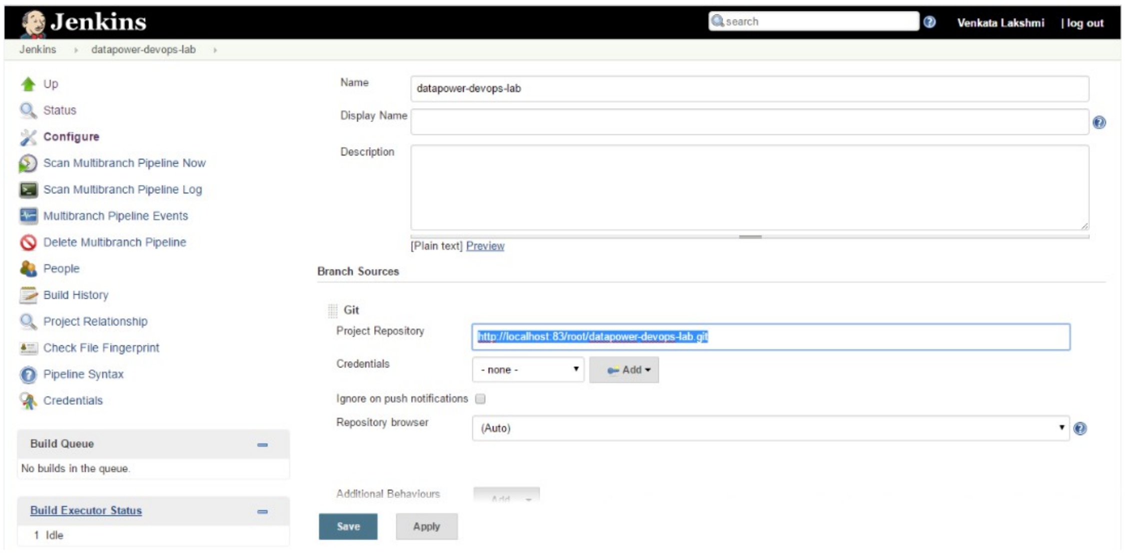
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Multibranch Pipeline with defaults
Extend Multibranch pipeline plugin and build branches it the default prepred pipeline script.Creates a set of Pipeline projects according to detected branches in one SCM repository.

if you want to create a new item from other existing, you can use this option:

Copy from: Type to autocomplete

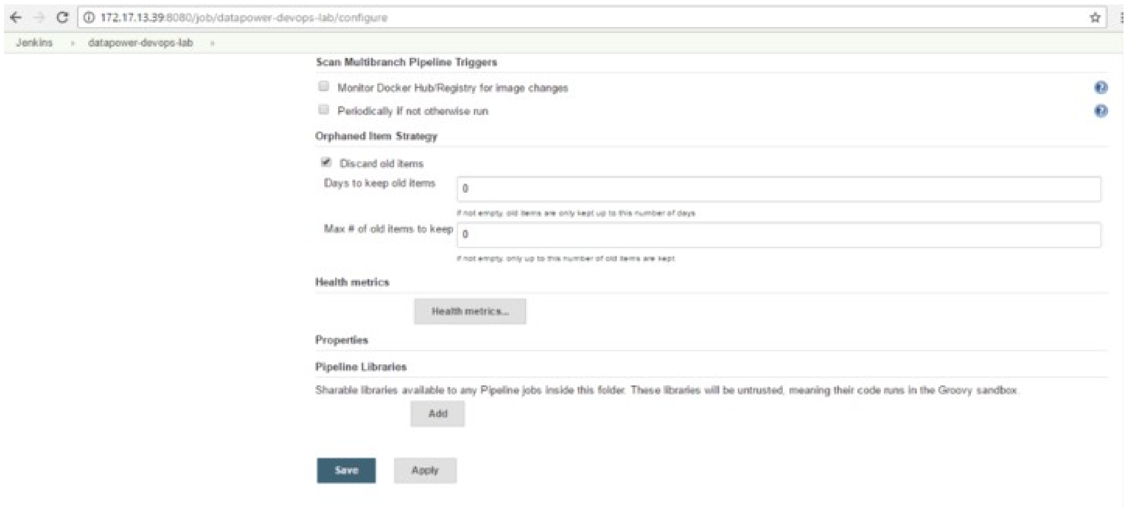
- Now configure the job with the following details
- In the **Source Code Management** part click on Git and give the URL in project repository and click on Save button.



Name: datapower-devops-lab
Display Name:
Description:
[Plain text] Preview

Branch Sources

Git
Project Repository: http://localhost:83/root/datapower-devops-lab.git
Credentials: - none - | Add
Ignore on push notifications ☐
Repository browser: (Auto)
Additional Behaviours: Add



172.17.13.39:8080/job/datapower-devops-lab/configure

Jenkins › datapower-devops-lab ›

Scan Multibranch Pipeline Triggers

☐ Monitor Docker Hub/Registry for image changes

☐ Periodically if not otherwise run

Orphaned Item Strategy

☑ Discard old items

Days to keep old items: 0

if not empty, old items are only kept up to this number of days

Max # of old items to keep: 0

if not empty, only up to this number of old items are kept

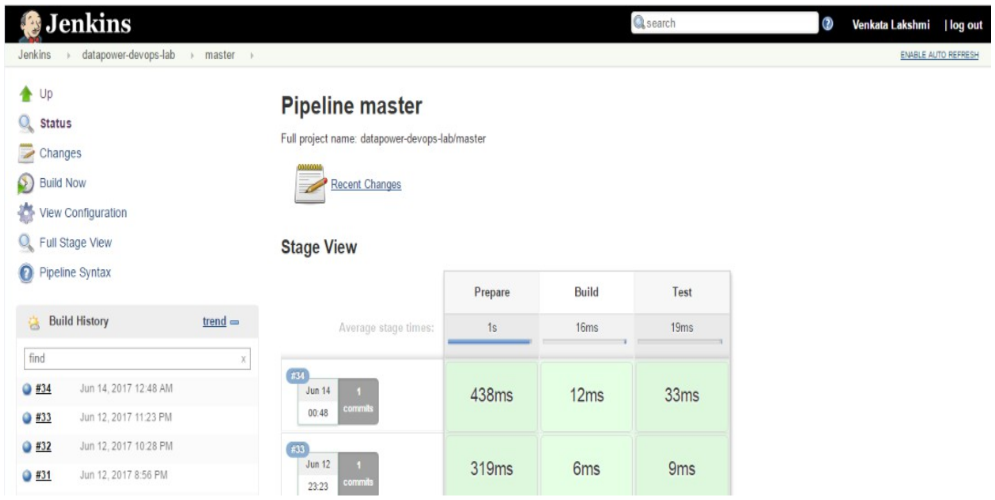
Health metrics

Properties

Pipeline Libraries

Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

- Now we can see the status of the job in jenkins as follows.



Jenkins › datapower-devops-lab › master ›

Pipeline master

Full project name: datapower-devops-lab/master

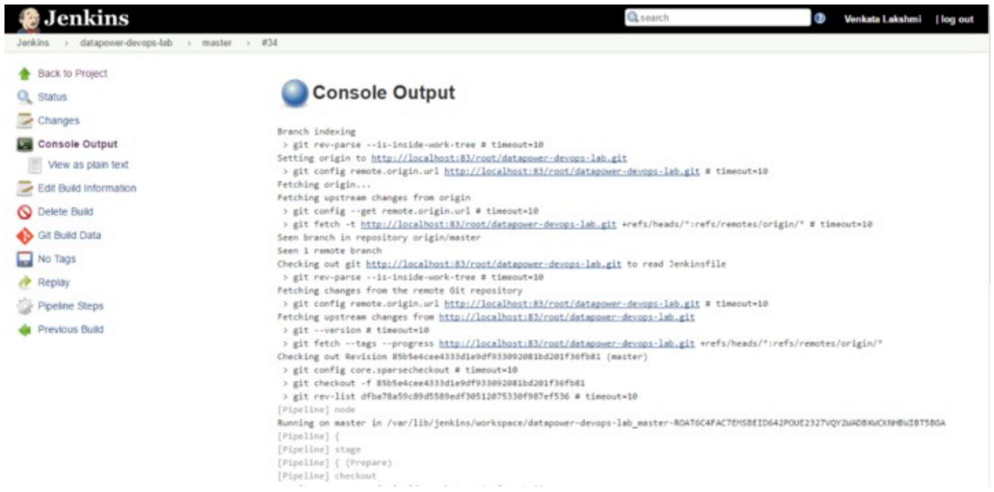
Stage View

	Prepare	Build	Test
Average stage times:	1s	16ms	19ms
#34 Jun 14 00:48 1 commits	438ms	12ms	33ms
#33 Jun 12 23:23 1 commits	319ms	6ms	9ms

Build History trend

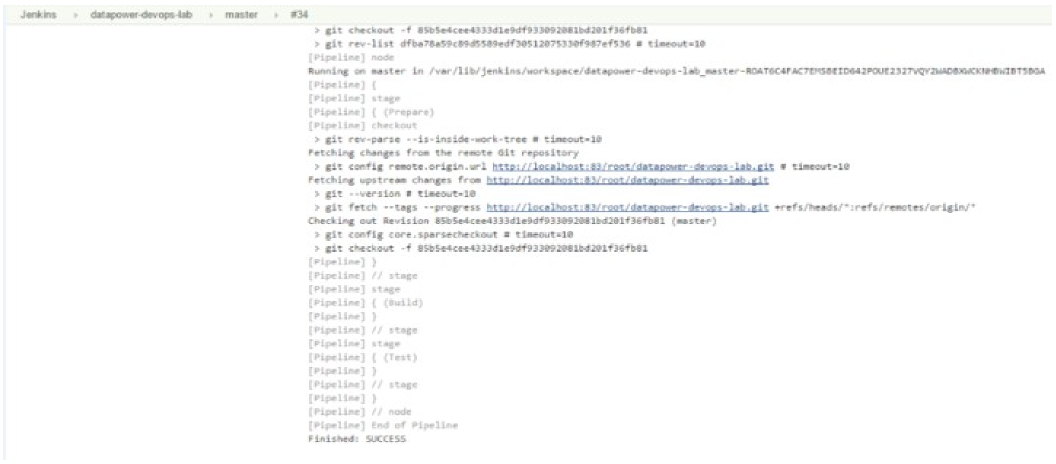
#34	Jun 14, 2017 12:48 AM
#33	Jun 12, 2017 11:23 PM
#32	Jun 12, 2017 10:28 PM
#31	Jun 12, 2017 8:56 PM

- Finally we can see the console output as **Success** as shown in the below.



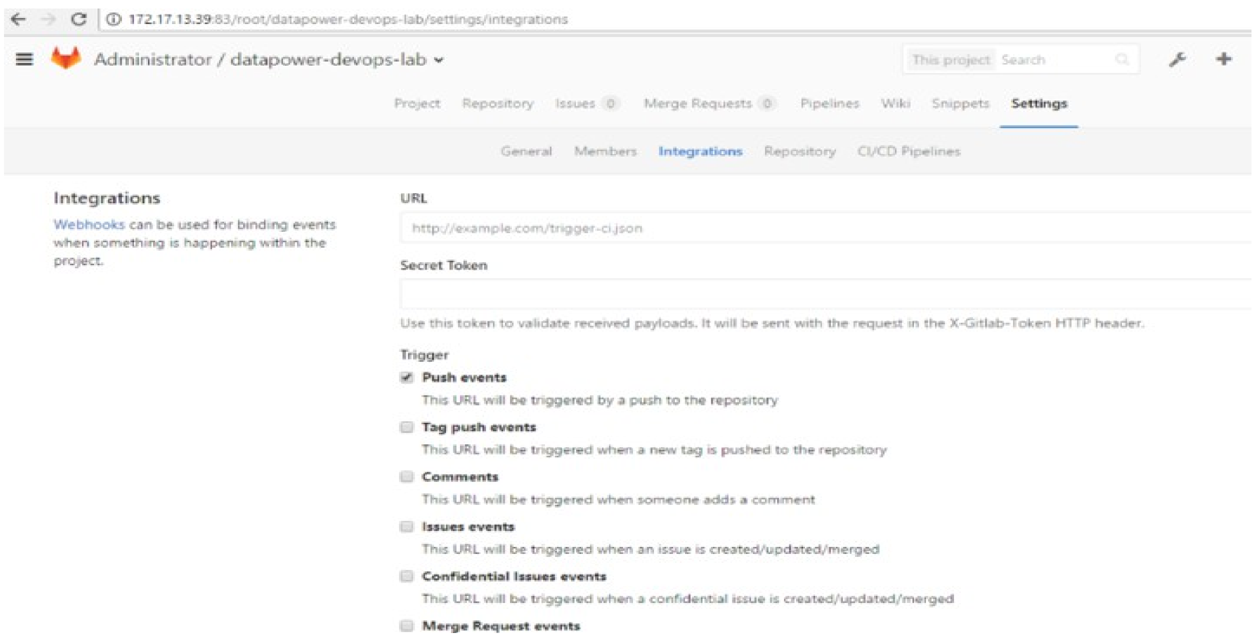
Console Output

```
Branch indexing
 > git rev-parse --is-inside-work-tree # timeout=10
Setting origin to http://localhost:83/root/datapower-devops-lab.git
 > git config remote.origin.url http://localhost:83/root/datapower-devops-lab.git # timeout=10
Fetching origin...
Fetching upstream changes from origin
 > git config --get remote.origin.url # timeout=10
 > git fetch -t http://localhost:83/root/datapower-devops-lab.git +refs/heads/*:refs/remotes/origin/* # timeout=10
Seen branch in repository origin/master
Seen 1 remote branch
Checking out git http://localhost:83/root/datapower-devops-lab.git to read Jenkinsfile
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url http://localhost:83/root/datapower-devops-lab.git # timeout=10
Fetching upstream changes from http://localhost:83/root/datapower-devops-lab.git
 > git --version # timeout=10
 > git fetch --tags --progress http://localhost:83/root/datapower-devops-lab.git +refs/heads/*:refs/remotes/origin/*
Checking out Revision 85b5e4cee4333d1e9df933092081bd201f36fb81 (master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 85b5e4cee4333d1e9df933092081bd201f36fb81
 > git rev-list dfba78a59c89d5589edf30512075330f987ef536 # timeout=10
[Pipeline] node
Running on master in /var/lib/jenkins/workspace/datapower-devops-lab_master-ROAT6C4FAC7EMSBEID642POUE2327VQY2WADBXWCKNHBWIBT5BGA
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Prepare)
[Pipeline] checkout
```



```
 > git checkout -f 85b5e4cee4333d1e9df933092081bd201f36fb81
 > git rev-list dfba78a59c89d5589edf30512075330f987ef536 # timeout=10
[Pipeline] node
Running on master in /var/lib/jenkins/workspace/datapower-devops-lab_master-ROAT6C4FAC7EMSBEID642POUE2327VQY2WADBXWCKNHBWIBT5BGA
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Prepare)
[Pipeline] checkout
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url http://localhost:83/root/datapower-devops-lab.git # timeout=10
Fetching upstream changes from http://localhost:83/root/datapower-devops-lab.git
 > git --version # timeout=10
 > git fetch --tags --progress http://localhost:83/root/datapower-devops-lab.git +refs/heads/*:refs/remotes/origin/*
Checking out Revision 85b5e4cee4333d1e9df933092081bd201f36fb81 (master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 85b5e4cee4333d1e9df933092081bd201f36fb81
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

- Now its all done. We have to integrate jenkins withlab in the gitlab integrations part.



172.17.13.39:83/root/datapower-devops-lab/settings/integrations

Administrator / datapower-devops-lab

Integrations

Webhooks can be used for binding events when something is happening within the project.

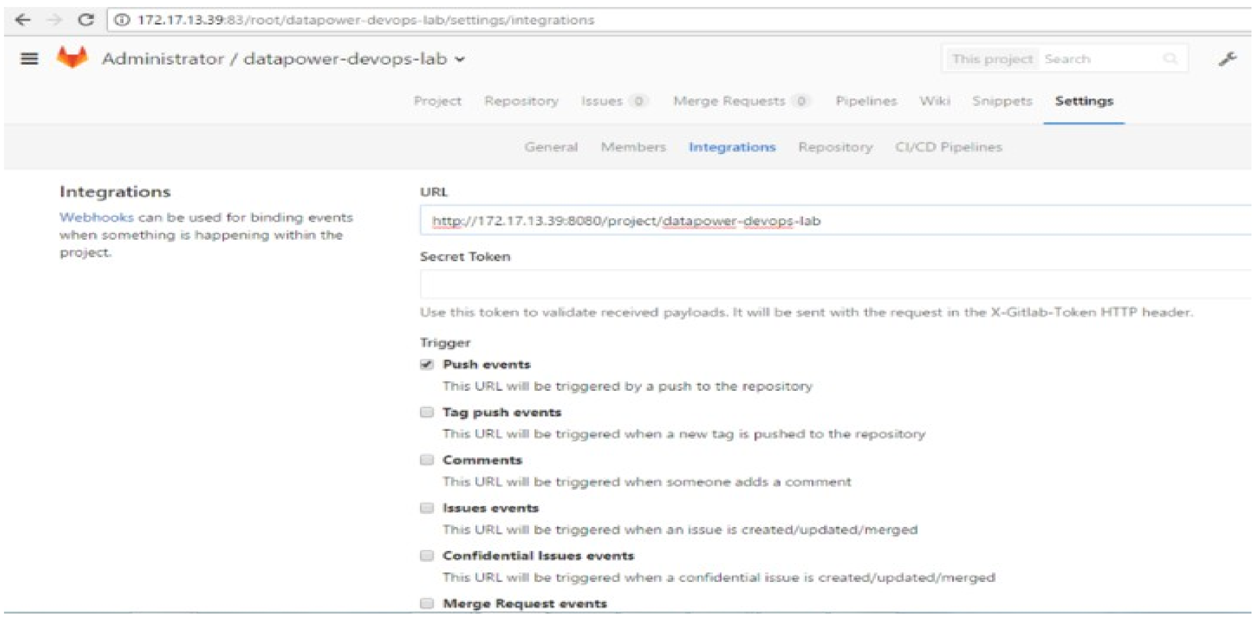
URL: http://example.com/trigger-ci.json

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

- [✓] Push events — This URL will be triggered by a push to the repository
- [ ] Tag push events — This URL will be triggered when a new tag is pushed to the repository
- [ ] Comments — This URL will be triggered when someone adds a comment
- [ ] Issues events — This URL will be triggered when an issue is created/updated/merged
- [ ] Confidential Issues events — This URL will be triggered when a confidential issue is created/updated/merged
- [ ] Merge Request events



172.17.13.39:83/root/datapower-devops-lab/settings/integrations

Administrator / datapower-devops-lab

Integrations

Webhooks can be used for binding events when something is happening within the project.

URL: http://172.17.13.39:8080/project/datapower-devops-lab

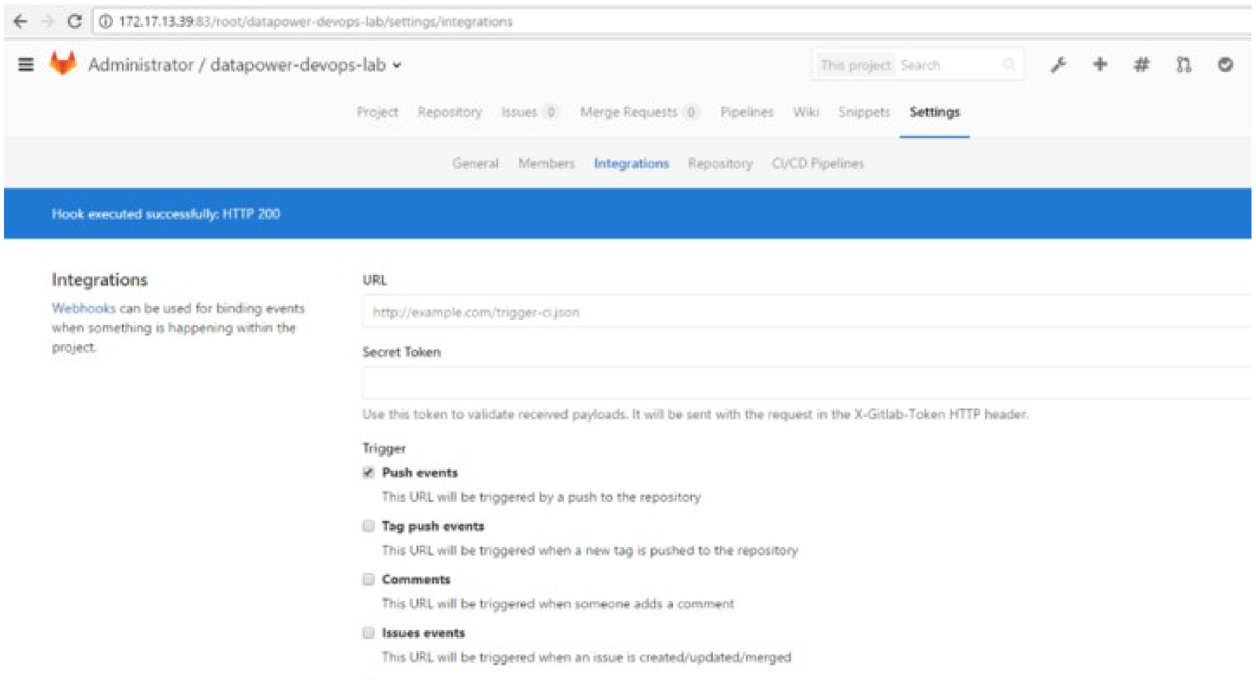
Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

- [✓] Push events — This URL will be triggered by a push to the repository
- [ ] Tag push events — This URL will be triggered when a new tag is pushed to the repository
- [ ] Comments — This URL will be triggered when someone adds a comment
- [ ] Issues events — This URL will be triggered when an issue is created/updated/merged
- [ ] Confidential Issues events — This URL will be triggered when a confidential issue is created/updated/merged
- [ ] Merge Request events

Now click on add webhook. Click the test button, it should prompt Test executed successfully as shown below.


172.17.13.39:83/root/datapower-devops-lab/settings/integrations

Administrator / datapower-devops-lab

This project Search
Project Repository Issues 0 Merge Requests 0 Pipelines Wiki Snippets Settings
General Members Integrations Repository CI/CD Pipelines

Hook executed successfully: HTTP 200

Integrations

Webhooks can be used for binding events when something is happening within the project.

URL

http://example.com/trigger-ci.json

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

- [x] **Push events**
  This URL will be triggered by a push to the repository
- [ ] **Tag push events**
  This URL will be triggered when a new tag is pushed to the repository
- [ ] **Comments**
  This URL will be triggered when someone adds a comment
- [ ] **Issues events**
  This URL will be triggered when an issue is created/updated/merged

So every time when you push some changes into gitlab, gitlab tells jenkins to automatically build the test.