Cisco
Cloud Partner VT

Cloud Native
Development Lab

Overview

Lab Objective

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

What are the lab requirements

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

What components are we going to use

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Github

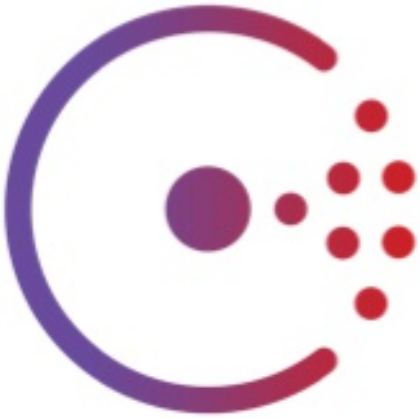
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Docker

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Consul

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

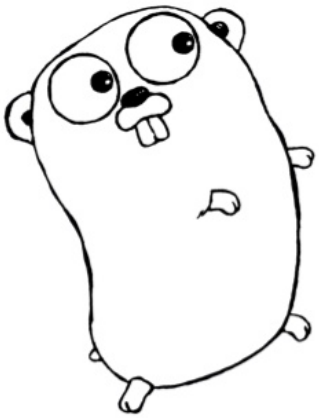


Fabio

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Redis

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.



Golang

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

What is the lab outcome?

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Any assumptions about the lab?

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Through this lab we will use and refer to certain syntax, boxes and commands, below is a summary these for your reference;

```
#
```

Any commands that you need to enter will be shown in boxes as shown above.

A single # will mean to be entered onto the host machine, <NAME># will refer to commands that need to be entered into the respective container, for example;

```
webserver#
```

Will mean the following command will need to be entered into the docker container called webserver.

Throughout this lab guide there will also be boxes with additional information and tips and some text that will be highlighted to signify its importance, they will look like the following;

Important information will be presented in boxes like this.

Important information will be in here

Step 1 - Docker Setup

At this stage you should have a working copy of docker installed either on your machine or accessible from your machine.

To check this is working as expected, please try the following;

If you recieved some output similar to the above, then this lab is going to be very easy!

Please note you may receive different versions than the above, this is not too much of a concern as we will not be using any version specific features.

Step 1 - Docker Setup

Lets make sure we have a clean environment for running this lab, so make sure we have no running containers and no images available. (This is optional, but is going to make life easier for this lab)

```
# docker ps -a
```

We will be using this command a lot.

The output from the command should look similar to the below;

```
❯ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

Now check the images available;

```
# docker images
```

The output from the command should look similar to the below;

```
❯ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

If you need to remove a running container, copy the ID from the "CONTAINER ID" column in the first command and use it in the following format;

```
# docker rm -f <ID>
```

If you need to remove an image, copy the ID from the "IMAGE ID" column in the second command and use it in the following format;

```
# docker rmi <ID>
```

Finally rerun the previous two commands to ensure you have a clean environment;

```
# docker ps -a
# docker images
```

Step 1 - Docker Setup

We can now pull a couple of images we will need for this lab. Please be patient, it will take a few minutes for each of these commands to complete;

```
# docker pull golang
```

During this process you should see output that resembles;

Continue and pull the other three images we need;

```
# docker pull consul
```

And

```
# docker pull redis
```

And

```
# docker pull magiconair/fabio
```

Step 1 - Docker Setup - Troubleshooting

You may receive an error which states something like;
"Cannot connect to the Docker daemon. Is the docker daemon running on this host?"

This is todo with the docker process not running or not being linked correctly. If running on a Mac, Check your output looks like the following;

```
# ls -la /var/run/docker.sock

lrwxr-xr-x  1 root  daemon  59  2 Aug 11:44 /var/run/docker.sock ->
/var/root/Library/Containers/com.docker.docker/Data/s60
```

This is to do with a link being created incorrectly. The important part to check is the piece highlighted, if your outut matches this, please proceed to fix it with the following;

```
# sudo ln -sf ~/Library/Containers/com.docker.docker/Data/s60 /var/run/docker.sock
```

Afterwards the output should look like the following;

```
# ls -la /var/run/docker.sock

lrwxr-xr-x  1 root  daemon  59  2 Aug 11:44 /var/run/docker.sock ->
/users/<USERNAME>/Library/Containers/com.docker.docker/Data/s60
```

Step 2 - Consul Setup

Consul does not need any specific configuration or changes to be made for this lab, so we can go ahead and run it and then check that it is running as expected.

```
# docker run -d -p 8500:8500 --name consul consul
```

So what have we just done;

- docker
    This command just tells the terminal to invoke the docker application

- run
    This command tells the docker application the activity we are going to action

- -d
    This flag tells docker the container is going to run in detached mode, ie: we will run it and connect to it or interact with it in another way, do not drop us into the terminal.

- -p 8500:8500
    This flag tells docker to map the host port:container port, so in this case, we explicitly want port 8500 on the host to map to 8500 of this container.

- --name consul
    Every container can have a unique name. It is an easy way to address the container rather than remembering or looking up the Container ID each time.

- consul
    This sets the container we are going to start. This is the name that you can see if you run "docker images". This will in most cases contain a /, but is not required to.

After running the above command you should receive a long number as your confirmation that the action was performed. It will look something like this;

```
❯ docker run -d -p 8500:8500 --name consul consul
b0dc2607f1f29f6aeeedf71bb4797b07f27c53bc94f898119b072a55329b336b
```

Lets check Docker correctly started the container and it is running as expected;

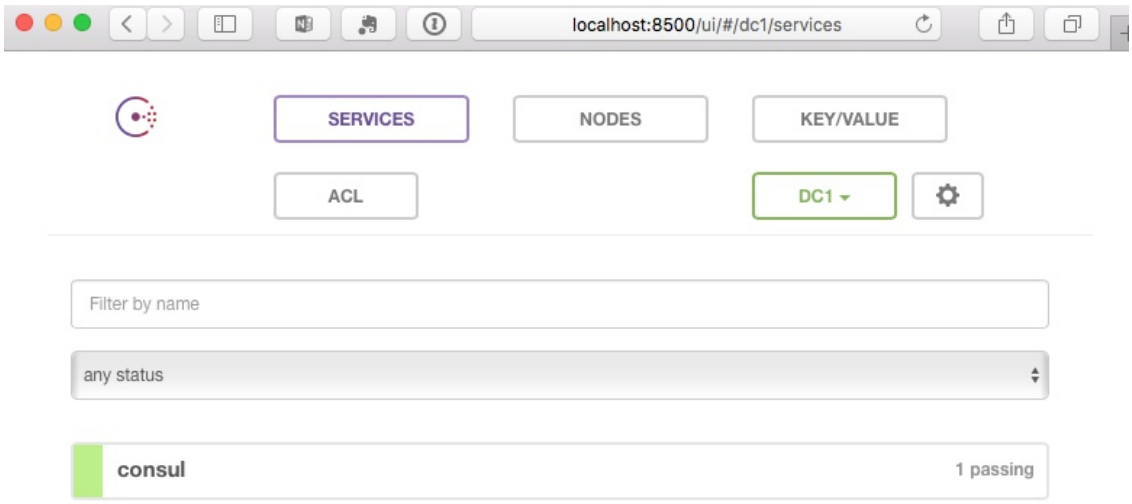
```
# docker ps -a
```

Step 2 - Consul Setup

You should see something like the below. Please note the ID number and mapped ports will be different.

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                                                                  NAMES
d07ffcee893a        consul              "docker-entrypoint.sh"   3 seconds ago       Up 2 seconds        8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8600/tcp, 8600/udp, 0.0.0.0:8500->8500/tcp   consul
```

At this point a simple test should confirm you have started your first container and port mapping is working successfully! Open a web browser and navigate to http://localhost:8500 and you should be presented with the following screen.



The final thing we need from Consul is it's IP address, if it is the first container, it is likley to be 172.17.0.2, but lets check;

```
# docker inspect -f '{{.Name}} - {{.NetworkSettings.IPAddress }}' $(docker ps -aq)
```

The output should be similar to the below;

```
 ❯ docker inspect -f '{{.Name}} - {{.NetworkSettings.IPAddress }}' $(docker ps -aq)
/consul - 172.17.0.2
```

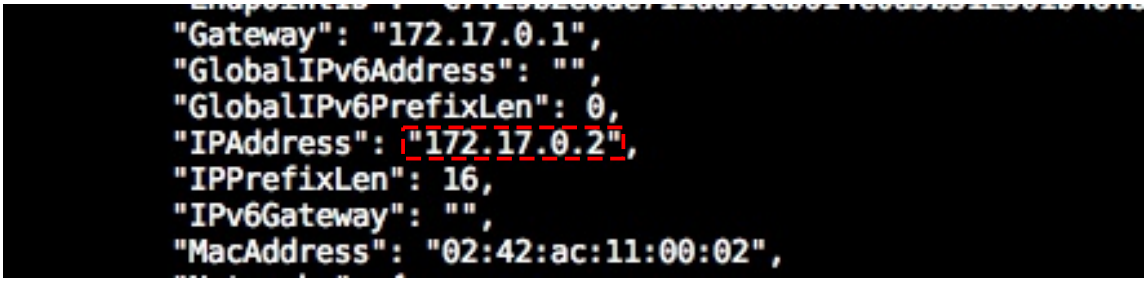
Now we know the IP that has been assigned to our Consul container. Make a note of it as you will need it later in the lab.

Step 2 - Consul Setup

An alternative way to get this information is;

```
# docker inspect consul
```

The output will contain amongst other things the IP, it should be similar to the below;



Step 3 - Fabio Setup

Fabio requires some configuration for our environment and we will achieve this via a simple text configuration file. There are many ways we could point fabio to the config file, but we will do this via sharing a folder in to the container. So the first thing we need is a shared folder.

```
# mkdir /tmp/lab/fabio
```

We have created it in the tmp folder to avoid any user access or priviledge issues.Next, lets create our configuration file;

```
# nano /tmp/lab/fabio/fabio.properties
```

In this instance I am using nano to create the text file, however please use whichever text editor you are comfortable with and have installed on your system.

Paste the following into the file;

```
registry.consul.addr = <CONSUL IP>:8500
```

Thats all we need in the config file and it is to ensure Fabio can correctly locate our Consul server. Finally we need to start our Fabio server, with the any ports we need and the config file we just created.

```
# docker run -d -p 9999:9999 -p 9998:9998 -v /tmp/lab/fabio:/etc/fabio --name fabio magiconair/fabio
```

So what have we just achieved;

-p 9999:9999
: Map host port 9999 to container port 9999, this is the port we will access any service on.

-p 9998:9998
: Map host port 9998 to container port 9998, this is the port we will access to view the services.

-v /tmp/lab/fabio:/etc/fabio
: This is the folder map to ensure our config file is loaded by our Fabio server, essentially map local folder /tmp/lab/fabio to the container folder /etc/fabio.

--name fabio
: Lets name the container to something that we can remember.

magiconair/fabio
: This is the name of the container image we will run.

Step 3 - Fabio Setup

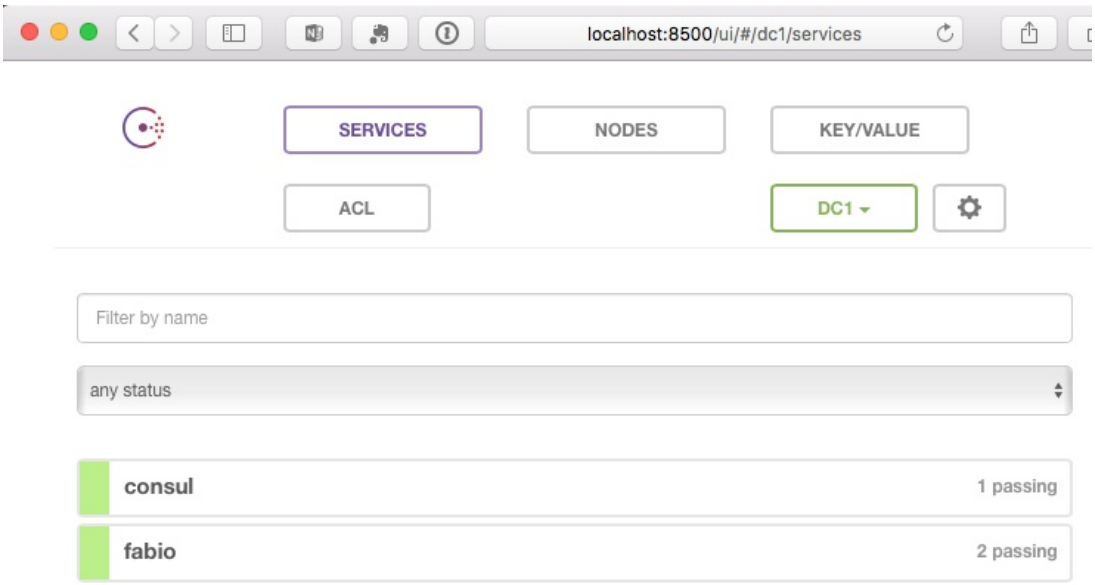
If everything worked successfully you should recieve the long string back and running;

```
# docker ps -a
```

We should now see our two containers running. Checking the status column to ensure they are both up and running and neither have exited.

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                            NAMES
6a459c48cf15        magiconair/fabio    "/fabio -cfg /etc/fab"   2 minutes ago       Up 2 minutes        0.0.0.0:9998-9999->9998-9999/tcp                 fabio
09cbcf2693ae        consul              "docker-entrypoint.sh"   4 minutes ago       Up 4 minutes        8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8600/tcp, 8600/udp, 0.0.0.0:8500->8500/tcp   consul
```

At this point our containers are up and running, now we need to check that Fabio has registered with Consul successfully. Open a web browser and go to http://localhost:8500 and check the services tab.



Lastly we can browse to Fabio to ensure its ready for our application. Browse to http://localhost:9998

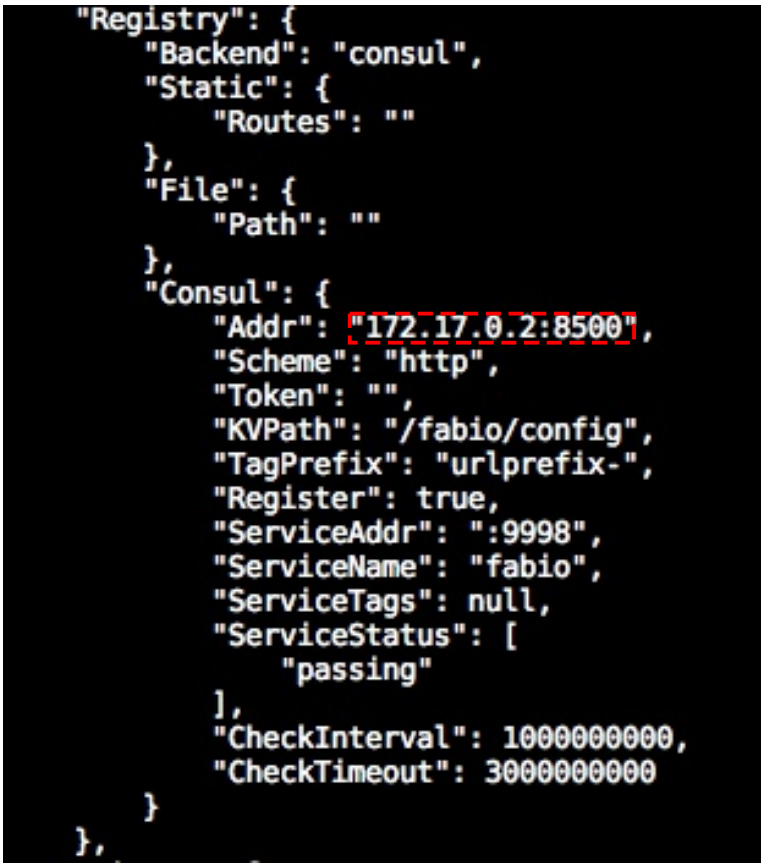
Step 3 - Fabio Setup - Troubleshooting

If you do not see Fabio listed on the on the services tab, there could be a number of things that are not working, however the most common is that Fabio was not able to read the configuration succesfully and that would usually be down to an incorrect folder mapping.

A quick way to check is to look at the config Fabio has, to do that, run the following command;

```
# docker logs fabio
```

You may find some indication at the end of the log that tells you why or if the Fabio container is operating or not. If there are no obvious reasons, the Fabio config file will also be displayed. If you find the section in the config of Registry, then Consul and look for the Addr, this should map to the address you entered into the config file /tmp/lab/fabio/fabio.properties.



If any of these details are incorrect, remove the container using "docker rm -f <ID>" and then try again, paying particular attention to the -v path of the docker run command.

Step 4 - Redis Setup

As we have already pulled the Redis image (This is not actually required. The following command will pull the image itself if it is not found on the local filesystem).

Running the following command will do everything we need for Redis.

```
# docker run -d -p 6379:6379 --name redis redis
```

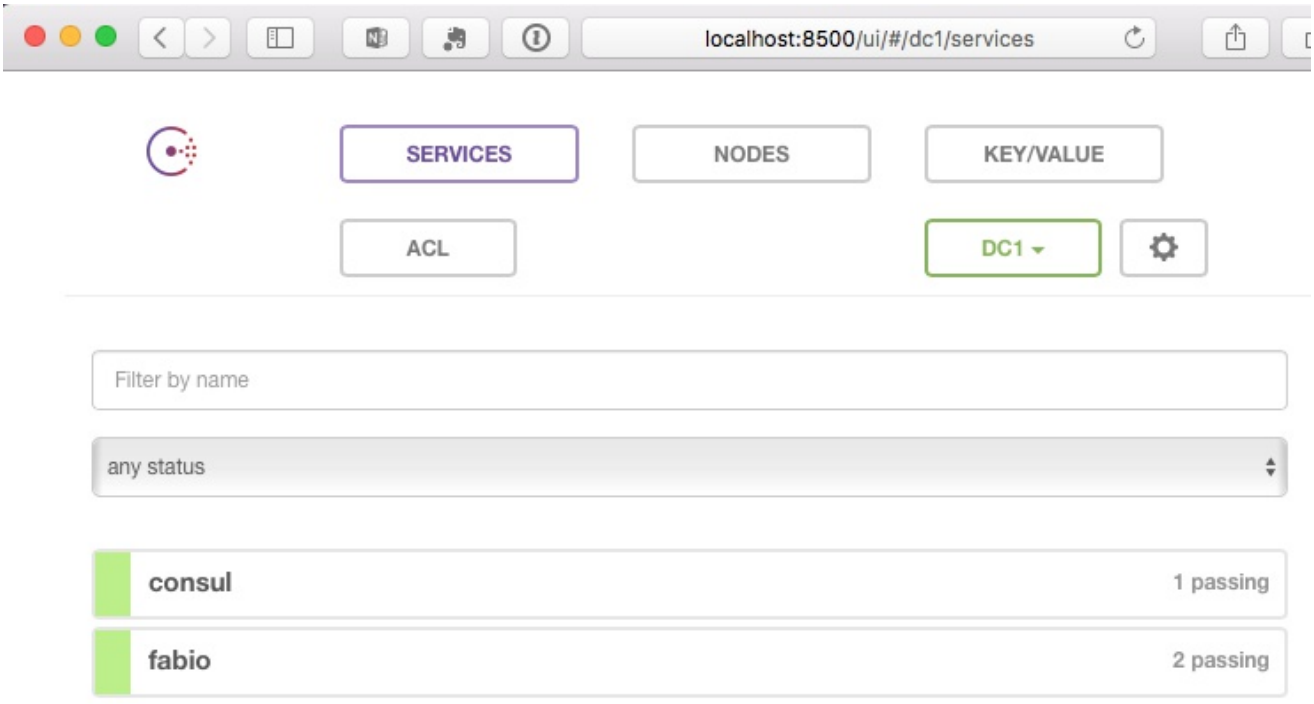
Lets test quickly to ensure it is running;

```
# docker ps -a
```

Hopefully the output now looks like the following;

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                                                                                NAMES
930b6128cfeb        redis               "docker-entrypoint.sh"   22 hours ago        Up 18 hours         0.0.0.0:6379->6379/tcp                                                                               redis
6a459c48cf15        magiconair/fabio    "/fabio -cfg /etc/fab"   7 days ago          Up 5 days           0.0.0.0:9998-9999->9998-9999/tcp                                                                     fabio
09cbcf2693ae        consul              "docker-entrypoint.sh"   7 days ago          Up 5 days           8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8600/tcp, 8600/udp, 0.0.0.0:8500->8500/tcp                   consul
```

However if we check Consul it will not show our Redis server.

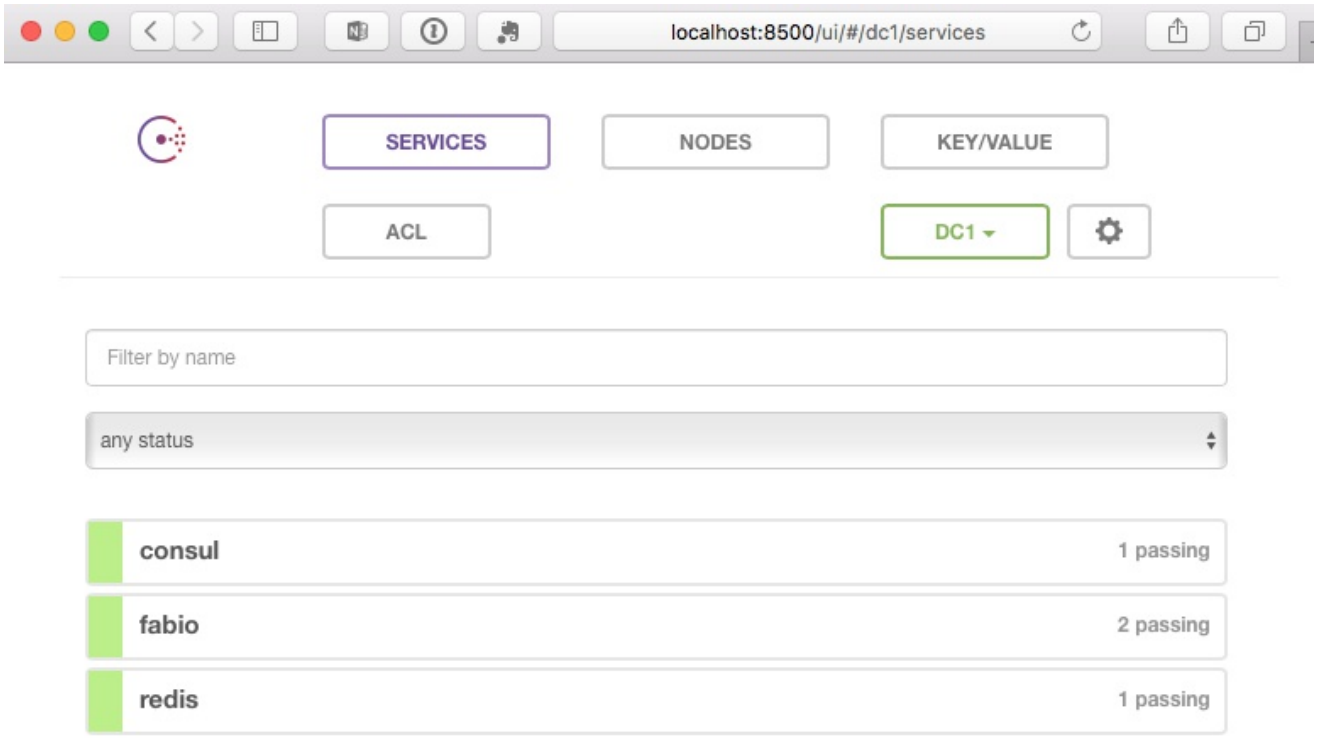


Step 4 - Redis Setup

As we will need to query Consul for the address of our Redis server we need to make Consul aware of it. There are different ways to achieve it, but using the default Redis image, we can just register it manually.

```
# curl -XPUT http://localhost:8500/v1/agent/service/register -d
'{"name":"redis","address":"<REDIS_IP>","port":6379}'
```

Nothing should happen at this point, but we can go to Consul and check to ensure that our Redis server now appears correctly.



Step 5 - Golang Setup

This section take us through installing and using Go within a container. If you have a local instance of GO then you could use it, but this is a simple and easy way to use a container for creating and building Go applications.

One of the great things about containers and one of the objectives is to try and keep the size to an absolute minimum. This helps with pushing and pulling and starting up new containers, Go can really help us here!

Go can compile the application and include all of the dependencies needed so the application can run on any system, this we will take advantage of with our simple webserver.

First off, open a new terminal window and then create a new folder for our work;

```
# mkdir /tmp/lab/golang
```

Lets create a very simple test case and ensure everything is working as expected.

```
# nano /tmp/lab/golang/main.go
```

Paste or copy in the following;

```
package main

import (
    "fmt"
    "net/http"
    "log"
)

func sayhelloName(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "<html><head></head><body bgcolor='red'></body></html>")
}

func main() {
    http.HandleFunc("/", sayhelloName) // set router
    err := http.ListenAndServe(":8080", nil) // set listen port
    if err != nil {
        log.Fatal("ListenAndServe: ", err)
    }
}
```

Now we have our temporary folders and files created, lets start the Golang container;

```
# docker run -it --name golang -v /tmp/lab/golang:/tmp golang
```

This time we have started the container with different parameters and no ports mapped. The parameters I will explain next, but as we will be using this container purely as our GO compiler, we do not need any tcp/udp ports for remote connections.

Step 5 - Golang Setup

-it

This is the main difference when we run the golang container. This could also have been written as -i -t. These two parameters mean;

-i

Technically means keep the STDIN open always. However it is more commonly referred to as Interactive mode.

-t

Tells Docker to allocate a TTY to the container. This could be called a PTY, providing an emulation of a physical terminal.

--name

This is the same as all the other examples, we will specificy a name, rather than having to

When you run this command, it will not return a container ID, it instead drops you at the terminal window for your container.

```
❯ docker run -it --name golang -v /tmp/golang:/tmp golang
root@33733b7dd51d:/go#
```

To ensure all has worked, cd to /tmp and do an ls. You should see your main.go file you created earlier in this step.

```
root@33733b7dd51d:/go# cd /tmp
root@33733b7dd51d:/tmp# ls
main.go
root@33733b7dd51d:/tmp#
```

Lastly, just to ensure we can build our future code without errors, try to build our main.go file.

```
root@33733b7dd51d:/tmp# go build main.go
root@33733b7dd51d:/tmp# ls
main  main.go
root@33733b7dd51d:/tmp#
```

Now we have our compiled binary for a very simple webserver we need some way of running it inside a container. This next example is a very simple way of doing it, but it has its issues which we will discuss later.

```
# nano /tmp/lab/golang/Dockerfile
```

First of all, we need to create a new file which will container the information needed to build our very first customer container. The name of the file "Dockerfile" is fixed and needs to be entered exactly as is.

Step 5 - Golang Setup

Paste the following into the file;

```
FROM ubuntu
ADD main /
CMD ["/main"]
```

This is telling Docker to perform a few different actions;

FROM ubuntu
This means we will be using Ubuntu as our base image. So if the latest Ubuntu does not exist on the system, pull it from Docker hub.

ADD main /
As we have our compiled binary called "main", we want to copy it into the container so it can be executed.

CMD ["/main"]
Lastly, we are telling the container to execute this command, which in our case is the binary we copied in, in the previous command.

Now we need to build our new container;

```
# docker build -t robjporter/golang1 .
```

You can replace the "robjporter" piece with whatever you would like, this is for identification purposes only.

```
 > docker build -t robjporter/golang1 .
Sending build context to Docker daemon 5.693 MB
Step 1 : FROM ubuntu
 ---> bd3d4369aebc
Step 2 : ADD main /
 ---> b0c745d9b363
Removing intermediate container 5ee0659cce22
Step 3 : CMD /main
 ---> Running in 211ad5bb6291
 ---> b1dd9d71ea0d
Removing intermediate container 211ad5bb6291
Successfully built b1dd9d71ea0d
```

If you saw the last message and it contained "Successfully built" you will now see your new container image listed with the others we have pulled for this lab. Trying listing the images;

```
 > docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
robjporter/golang1  latest              b1dd9d71ea0d        3 seconds ago       132.3 MB
```

The size is pretty good, but our application is only 5.5MB, so why is it taking 132.3MB? There must be a better way and a more MicrosService way of doing it?

Step 5 - Golang Setup

The last step in our testing is to run our new container. We do this in the same way as any other container.

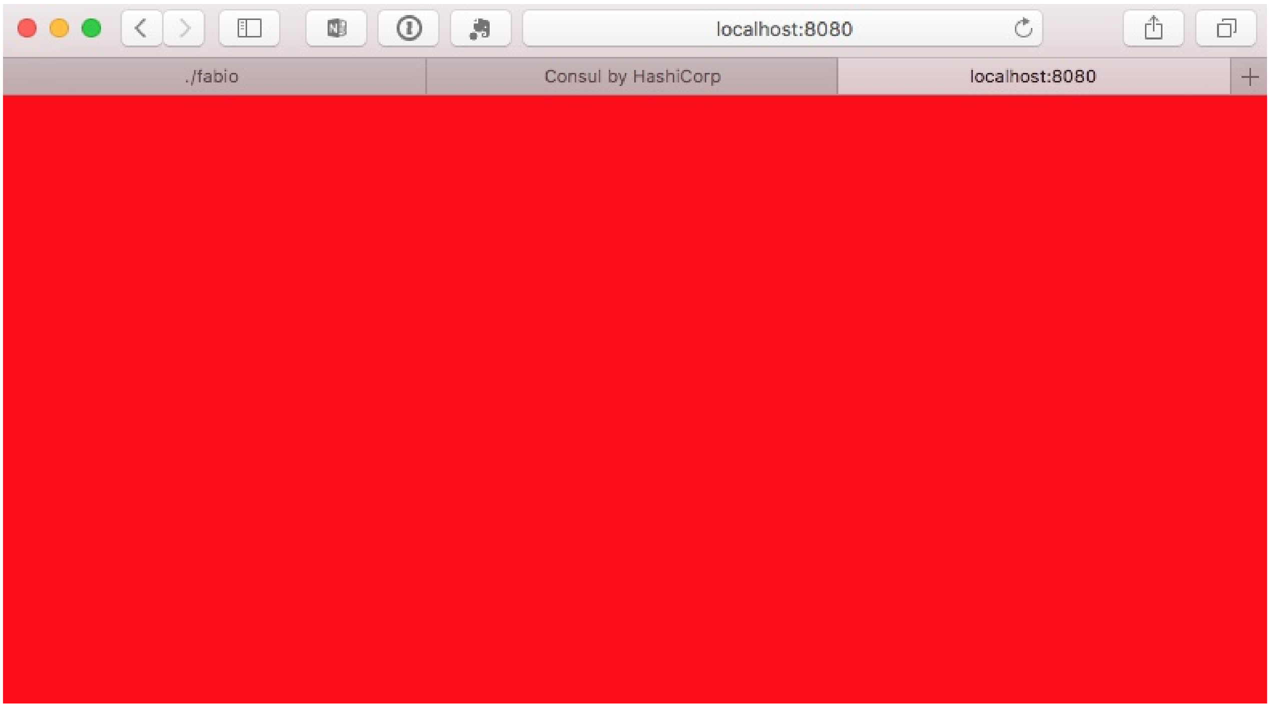
```
❯ docker run -d -p 8080:8080 --name golang1 robjporter/golang1
df99a0707d726e7800b930a1f803a76c295b6c240b958e1f7e659bbbb0410b44
```

We have seen all of these parameters before, we are mapping localhost port 8080 to container port 8080, which is what we specifed inside our GO application and we are referring to the container name, we used earlier, in my case "robjporter/golang1".

Lastly, lets check it is working as expected;

```
❯ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED            STATUS           PORTS                                                                          NAMES
df99a0707d72   robjporter/golang1   "/main"                  20 minutes ago     Up 20 minutes    0.0.0.0:8080->8080/tcp                                                         golang1
33733b7dd51d   golang               "/bin/bash"              40 minutes ago     Up 40 minutes                                                                                   golang
930b6128cfeb   redis                "docker-entrypoint.sh"   36 hours ago       Up 36 hours      0.0.0.0:6379->6379/tcp                                                         redis
6a459c48cf15   magiconair/fabio     "/fabio -cfg /etc/fab"   8 days ago         Up 6 days        0.0.0.0:9998-9999->9998-9999/tcp                                               fabio
09cbcf2693ae   consul               "docker-entrypoint.sh"   8 days ago         Up 6 days        8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8600/tcp, 8600/udp, 0.0.0.0:8500->8500/tcp   consul
```

And;



Step 6 - Bringing it all together