Table of Contents

1. INTRODUCTION

1.1 What are Containers

Containers are a method of operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes. Containers allow you to easily package an application's code, configurations, and dependencies into easy to use building blocks that deliver environmental consistency, operational efficiency, developer productivity, and version control. Containers can help ensure that applications deploy quickly, reliably, and consistently regardless of deployment environment. Containers also give you more granular control over resources giving your infrastructure improved efficiency.

1.2 Benefits of Containers:

1.2.1 **Environment Consistency**:

Containers can help you get more from your computing resources by allowing to you easily run multiple applications on the same instance. With containers, you can specify the exact amount of memory, disk space, and CPU to be used by a container on an instance.

1.2.2 **Operational Efficiency**:

Containers can help you get more from your computing resources by allowing to you easily run multiple applications on the same instance. With containers, you can specify the exact amount of memory, disk space, and CPU to be used by a container on an instance. Containers

have fast boot times because each container is only a process on the operating system running an application and its dependencies.

1.2.3 **Developer Productivity**:

Containers increase developer productivity by removing cross-service dependencies and conflicts. Each application component can be broken into different containers running a different microservice. Containers are isolated from one another, so you don't have to worry about libraries or dependencies being in sync for each service.

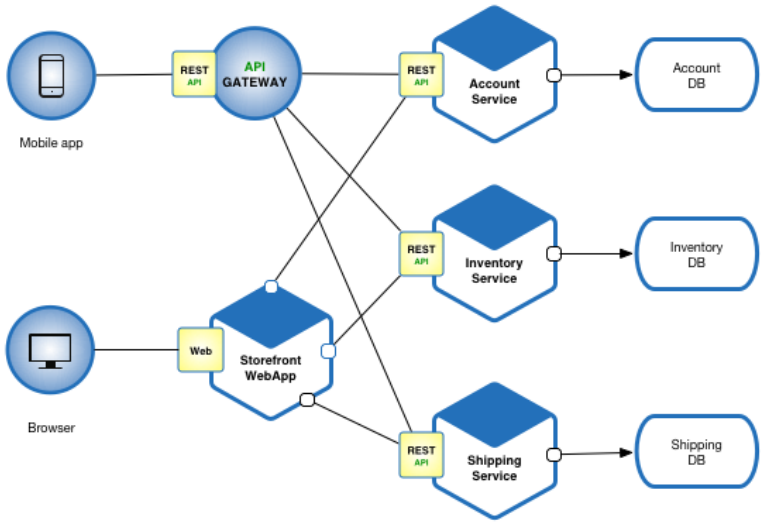
1.2.4 **Version Control**:

Containers allow you to track versions of your application code and their dependencies. Container images have a manifest file (Dockerfile) that allows you to easily maintain and track versions of a container, inspect differences between versions, and roll-back to previous versions.

1.3 What are Microservices

Microservices is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack.

Example of Microservices



Some Examples of microservices we will be using:

Jupyter Notebook
Client-host Application

1.3.1 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.

Notebook provides the multiple language platform along with the libraries and their packages

Following are the features of the Jupyter Notebook

1. Language of choice:
   The Notebook has support for over 40 programming languages, including those popular in Data Science such as Python, R, Julia and Scala.

2. Share notebooks:
   Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

3. Interactive widgets
   Code can produce rich output such as images, videos, LaTeX, and JavaScript. Interactive widgets can be used to manipulate and visualize data in real time.

4. Big data integration
   Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, dplyr, etc.

Another Application of the Jupyter Notebook is JupyterHub.

1.3.2 JupyterHub:

A multi-user version of the notebook designed for companies, classrooms and research labs. With JupyterHub you can create a **multi-user Hub** which spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server.

1.3.2.1 Three main actors make up JupyterHub:

- multi-user Hub (tornado process)
- configurable http proxy (node-http-proxy)
- multiple single-user Jupyter notebook servers (Python/IPython/tornado)

1.3.2.2 Basic principles for operation:

- Hub spawns a proxy.
- Proxy forwards all requests to Hub by default.
- Hub handles login, and spawns single-user servers on demand.
- Hub configures proxy to forward url prefixes to the single-user notebook servers.
- JupyterHub also provides a REST API for administration of the Hub and its users.

1.3.2.3 Features of JupyterHub:

- Pluggable authentication:
  Manage users and authentication with PAM, OAuth or integrate with your own directory service system. Collaborate with others through the Linux permission model.
- Centralized deployment:
  Deploy the Jupyter Notebook to all of the users in your organization on centralized servers on- or off-site.
- Container friendly:
  Use Docker containers to scale your deployment and isolate user processes using a growing ecosystem of prebuilt Docker containers
- Code meets data
  Deploy the Notebook next to your data to provide unified software management and data access within your organization.

Hub: Manages user accounts, authentication, and coordinates single user notebook servers using a spawner. Hub can offer notebook servers to a class of students, a corporate data science work group or a high-performance computing group.

1.3.2.3 JupyterHub Installation

Step1: Jupyterhub can be installed with pip, and the proxy with npm:

```
npm install -g configurable-http-proxy
pip3 install jupyterhub
```

Step 2: Run the Hub server:

To start the Hub server, run the command:

```
jupyterhub
```

Step 3: Visit https://localhost:8000 in your browser and sign in with your Github OAuth credentials.

OAuthenticator:

OAuth + JupyterHub Authenticator = OAuthenticator

Step 4: Create a GitHub OAuth application. Make sure the callback URL is:

```
http[s]://[your-host]/hub/oauth_callback
```

Step 5: Then, add the following to your jupyterhub_config.py file:

```
c.JupyterHub.authenticator_class = 'oauthenticator.GitHubOAuthenticator'

c.GitHubOAuthenticator.oauth_callback_url = os.environ['OAUTH_CALLBACK_URL']

c.GitHubOAuthenticator.client_id = os.environ['GITHUB_CLIENT_ID']

c.GitHubOAuthenticator.client_secret = os.environ['GITHUB_CLIENT_SECRET']
```

Which should look something like this:

```
root@ubuntu:/# cat jupyterhub_config.py
# Configuration file for jupyterhub.

#------------------------------------------------------------------------------
# Application(SingletonConfigurable) configuration
#------------------------------------------------------------------------------

## This is an application.

c.JupyterHub.authenticator_class = 'oauthenticator.GitHubOAuthenticator'
c.GitHubOAuthenticator.oauth_callback_url = 'https://192.168.222.132:8000/hub/oauth_callback'
c.GitHubOAuthenticator.client_id= 'd1101d1a5ca82fdb8d9e'
c.GitHubOAuthenticator.client_secret = 'b7c5a5245ef5eee11fc3544dff1c2c77249dc928'
c.LocalAuthenticator.create_system_users = True
c.Authenticator.whitelist = {'maansi13','max030','rutikaranjolkar'}
c.Authenticator.admin_users = {'maansi13','max030','rutikaranjolkar'}
c.Spawner.notebook_dir = '~/notebooks'
c.JupyterHub.ssl_cert = 'mycert.pem'
c.JupyterHub.ssl_key = 'mykey.key'
```

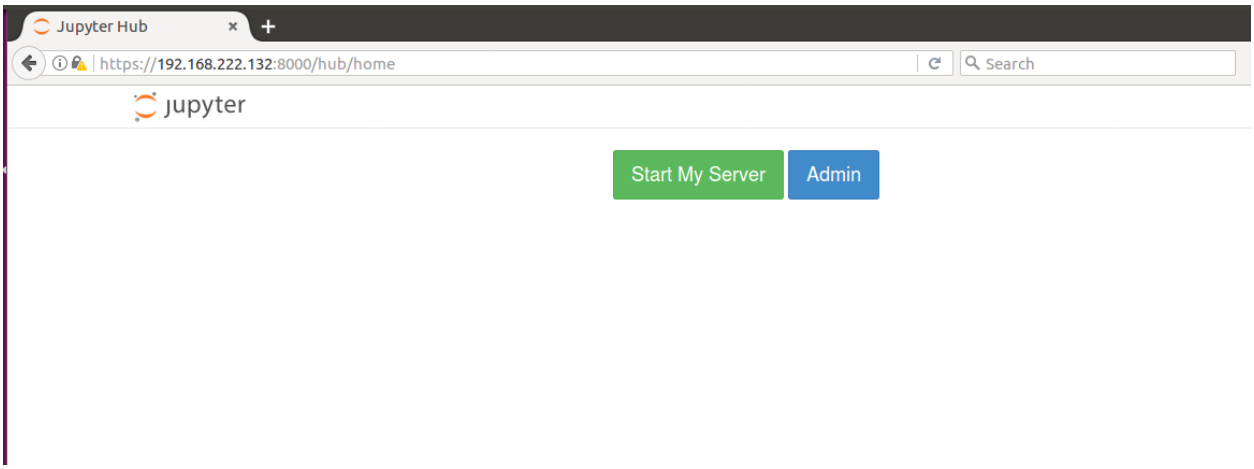
Step 6: Add Hub users to the Userlist like this:

```
root@ubuntu: /
maansi13 admin
max030 admin
rutikaranjolkar admin
~
~
~
```

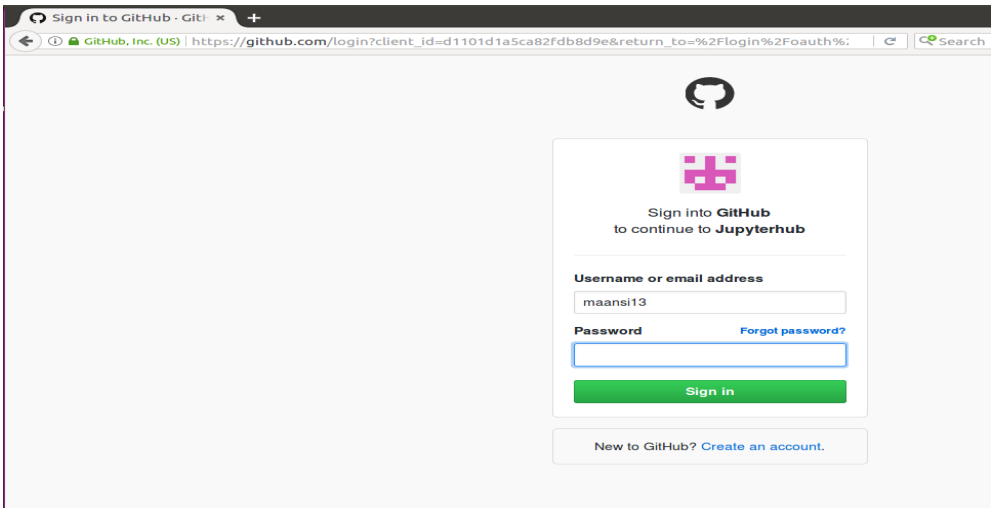
Step 7: Run the jupyterhub like this:

```
root@ubuntu:/# jupyterhub
[I 2017-05-31 18:53:44.798 JupyterHub app:724] Loading cookie_secret from /jupyterhub_cookie_secret
[W 2017-05-31 18:53:44.835 JupyterHub app:365]
    Generating CONFIGPROXY_AUTH_TOKEN. Restarting the Hub will require restarting the proxy.
    Set CONFIGPROXY_AUTH_TOKEN env or JupyterHub.proxy_auth_token config to avoid this message.

{'trait': <traitlets.traitlets.Unicode object at 0x7f8e5cbd9978>, 'owner': <jupyterhub.spawner.LocalProcessSpaw
ner object at 0x7f8e5ef373c8>, 'value': '~/notebooks'}
{'trait': <traitlets.traitlets.Unicode object at 0x7f8e5cbd9978>, 'owner': <jupyterhub.spawner.LocalProcessSpaw
ner object at 0x7f8e5ef370b8>, 'value': '~/notebooks'}
{'trait': <traitlets.traitlets.Unicode object at 0x7f8e5cbd9978>, 'owner': <jupyterhub.spawner.LocalProcessSpaw
ner object at 0x7f8e5ef371d0>, 'value': '~/notebooks'}
[I 2017-05-31 18:53:44.878 JupyterHub app:1453] Hub API listening on http://127.0.0.1:8081/hub/
[I 2017-05-31 18:53:44.885 JupyterHub app:1176] Starting proxy @ http://*:8000/
18:53:45.053 - info: [ConfigProxy] Proxying https://*:8000 to http://127.0.0.1:8081
18:53:45.057 - info: [ConfigProxy] Proxy API at http://127.0.0.1:8001/api/routes
[I 2017-05-31 18:53:45.094 JupyterHub app:1485] JupyterHub is now running at http://127.0.0.1:8000/
```

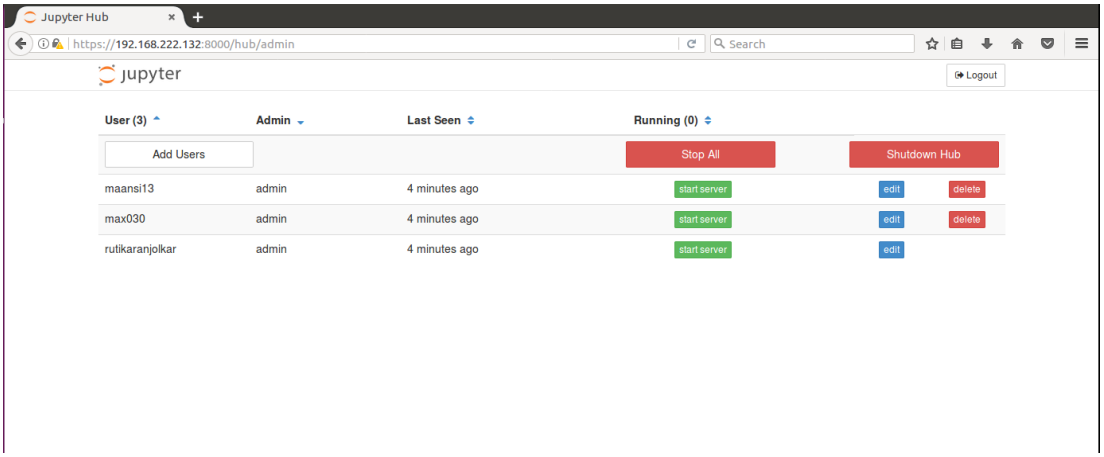
Step 8: Browser URL – Enter localhost at port 80 will look like this:



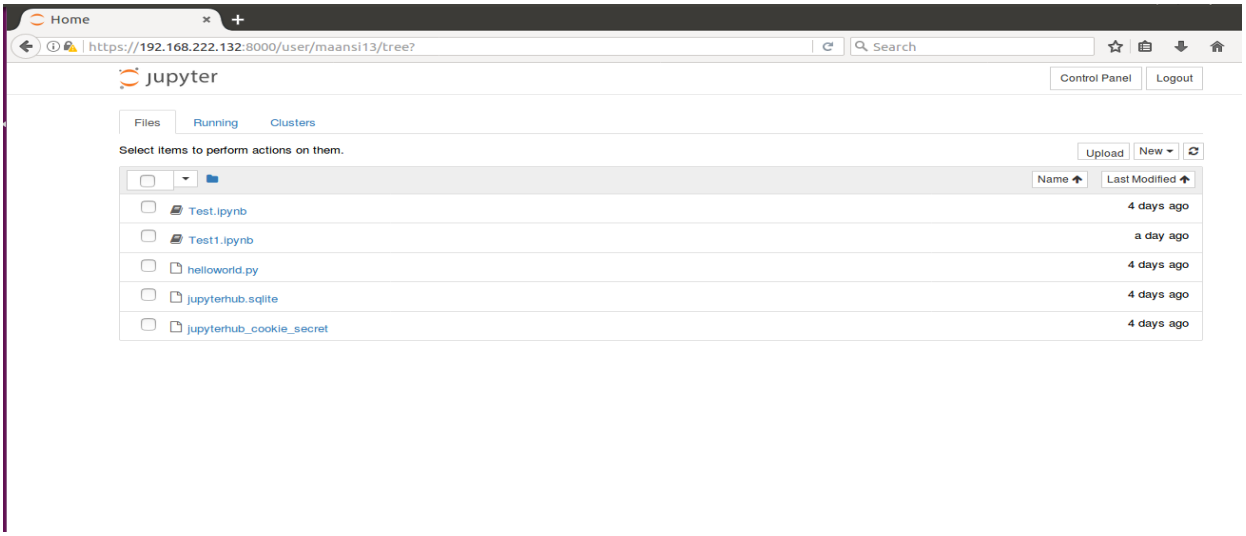
Step 9: GitHub OAuth login to authorize Hub users:



Step 10: Added user's dashboard (Each can start its own server):



Step 11: After starting the server, each user can access its own notebook space and add python files to it like this:



Step 12: All the communication between HTTP looks like this:

```
[I 2017-05-31 18:57:43.572 JupyterHub log:100] 302 GET /hub/logout (rutikaranjolkar@::ffff:192.168.222.132) 6.6
3ms
[I 2017-05-31 18:57:43.586 JupyterHub log:100] 302 GET /hub/ (@::ffff:192.168.222.132) 1.60ms
[I 2017-05-31 18:57:43.600 JupyterHub log:100] 302 GET /oauth_login (@::ffff:192.168.222.132) 1.65ms
[I 2017-05-31 18:57:43.605 JupyterHub oauth2:37] oauth redirect: 'https://192.168.222.132:8000/hub/oauth_callba
ck'
[I 2017-05-31 18:57:43.613 JupyterHub log:100] 302 GET /hub/oauth_login (@::ffff:192.168.222.132) 3.10ms
[I 2017-05-31 18:57:43.931 JupyterHub log:100] 302 GET /hub/oauth_callback?code=1c70fa3846742e1d207c (@::ffff:1
92.168.222.132) 227.23ms
[I 2017-05-31 18:57:55.655 JupyterHub login:19] User logged out: rutikaranjolkar
[I 2017-05-31 18:57:55.682 JupyterHub log:100] 302 GET /hub/logout (rutikaranjolkar@::ffff:192.168.222.132) 13.
25ms
[I 2017-05-31 18:57:55.724 JupyterHub log:100] 302 GET /hub/ (@::ffff:192.168.222.132) 3.71ms
[I 2017-05-31 18:57:55.763 JupyterHub log:100] 302 GET /oauth_login (@::ffff:192.168.222.132) 7.02ms
[I 2017-05-31 18:57:55.768 JupyterHub oauth2:37] oauth redirect: 'https://192.168.222.132:8000/hub/oauth_callba
ck'
[I 2017-05-31 18:57:55.775 JupyterHub log:100] 302 GET /hub/oauth_login (@::ffff:192.168.222.132) 1.61ms
[I 2017-05-31 18:59:48.896 JupyterHub log:100] 302 GET /hub/oauth_callback?code=5b0d2c65fc6ab7fcf307 (@::ffff:1
92.168.222.132) 296.17ms
[I 2017-05-31 18:59:48.932 JupyterHub log:100] 200 GET /hub/home (maansi13@::ffff:192.168.222.132) 15.43ms
[I 2017-05-31 18:59:51.519 JupyterHub log:100] 200 GET /hub/admin (maansi13@::ffff:192.168.222.132) 66.36ms
[I 2017-05-31 18:59:53.927 JupyterHub spawner:783] Spawning jupyterhub-singleuser '--user="maansi13"' '--cookie
-name="jupyter-hub-token-maansi13"' '--base-url="/user/maansi13"' '--hub-host=""' '--hub-prefix="/hub/"' '--hub
-api-url="http://127.0.0.1:8081/hub/api"' '--ip="127.0.0.1"' --port=42097 '--notebook-dir="~/notebooks"'
[W 2017-05-31 18:59:59.120 maansi13 login:225] All authentication is disabled.  Anyone who can connect to this
server will be able to run code.
[I 2017-05-31 18:59:59.155 maansi13 notebookapp:1366] Serving notebooks from local directory: /home/maansi13/no
tebooks
[I 2017-05-31 18:59:59.155 maansi13 notebookapp:1366] 0 active kernels
[I 2017-05-31 18:59:59.155 maansi13 notebookapp:1366] The Jupyter Notebook is running at: http://127.0.0.1:4209
7/user/maansi13/
[I 2017-05-31 18:59:59.155 maansi13 notebookapp:1367] Use Control-C to stop this server and shut down all kerne
ls (twice to skip confirmation).
[I 2017-05-31 18:59:59.199 JupyterHub base:322] User maansi13 server took 5.391 seconds to start
[I 2017-05-31 18:59:59.200 JupyterHub orm:188] Adding user maansi13 to proxy /user/maansi13 => http://127.0.0.1
:42097
[I 2017-05-31 18:59:59.200 maansi13 log:47] 302 GET /user/maansi13 (127.0.0.1) 3.57ms
[I 2017-05-31 18:59:59.215 JupyterHub log:100] 201 POST /hub/api/users/maansi13/server (maansi13@::ffff:192.168
.222.132) 5408.22ms
[I 2017-05-31 19:01:24.521 JupyterHub log:100] 302 GET /hub/ (maansi13@::ffff:192.168.222.132) 25.51ms
[I 2017-05-31 19:01:24.550 maansi13 log:47] 302 GET /user/maansi13 (::ffff:192.168.222.132) 3.09ms
[I 2017-05-31 19:01:24.624 JupyterHub log:100] 200 GET /hub/api/authorizations/cookie/jupyter-hub-token-maansi1
3/[secret] (maansi13@127.0.0.1) 17.20ms
```

2. DOCKER

2.1 What is Docker

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

2.1.1 Benefits of Docker

- **Ship More Software Faster**
  Docker users on average ship software 7x more frequently than non-Docker users. Docker enables developers to ship isolated services as often as needed by eliminating the headaches of software dependencies.
- **Improve Developer Productivity**
  Docker reduces the time spent setting up new environments or troubleshooting differences between environments.

- **Seamlessly Move Applications**

  Docker-based applications can be seamlessly moved from local development machines to production deployments on cloud.

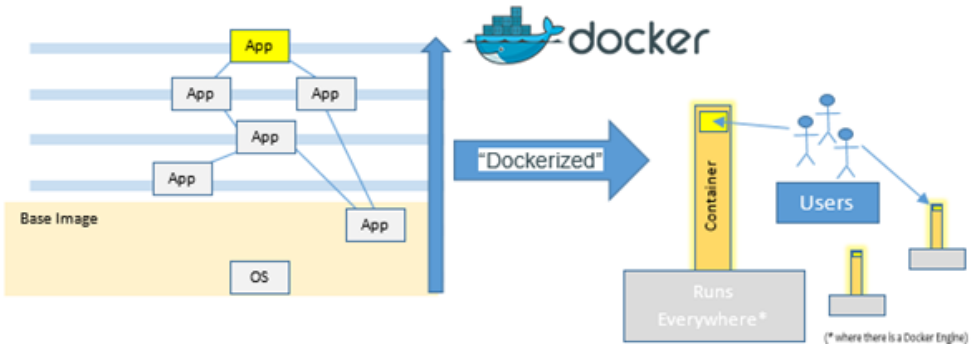
- **Standardize Application Operations**
  Small containerized applications make it easy to deploy, identify issues, and roll back for remediation.

- **Responsive deployment and scaling**

  Docker's container-based platform allows for highly portable workloads. It can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments. It makes it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

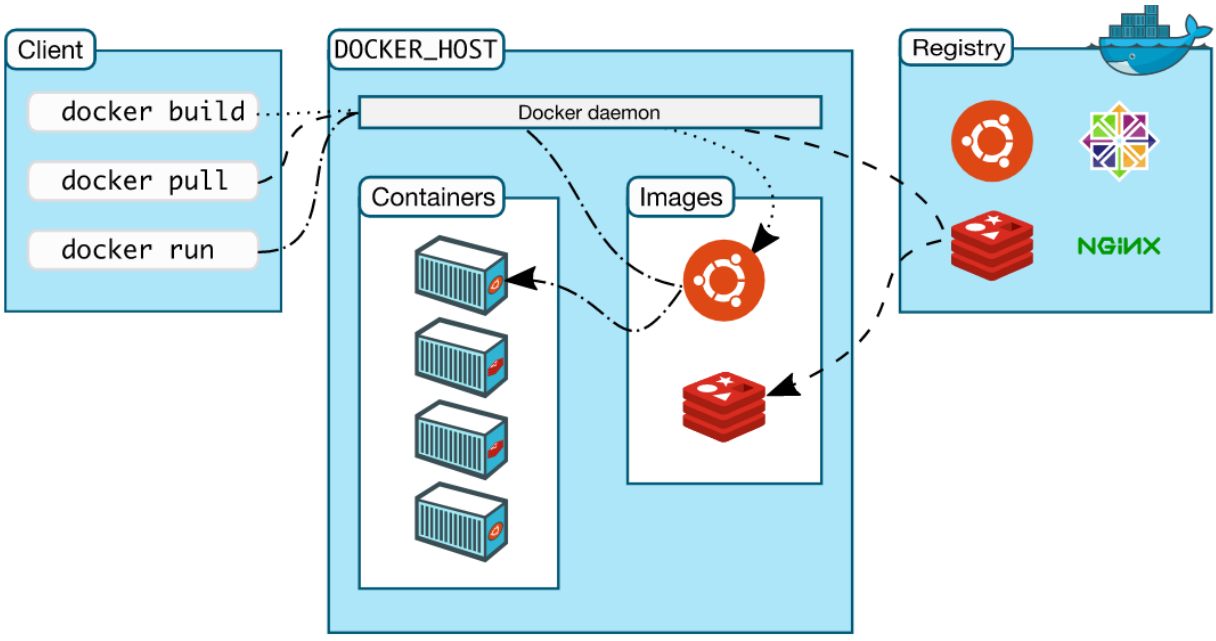
- **Running more workloads on the same hardware**

  Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals.



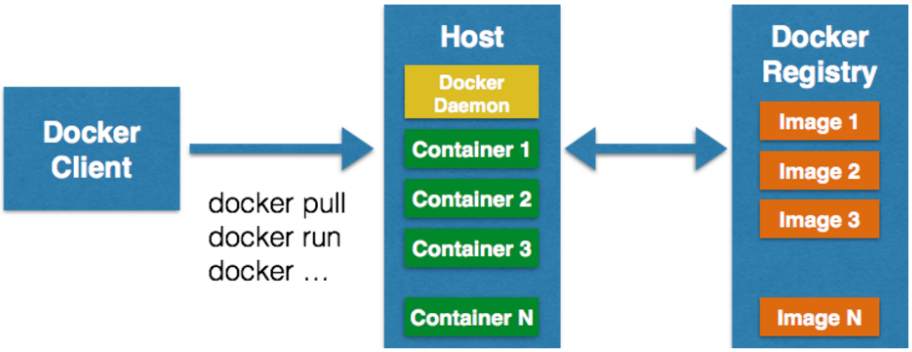
2.2 Docker Architecture:

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.



2.2.1 Docker Working:

Docker has a client-server architecture. Docker Daemon or server is responsible for all the actions that are related to containers. The daemon receives the commands from the Docker client through CLI or REST API's. Docker client can be on the same host as a daemon or it can be present on any other host.



2.2.2 Docker Components:

Docker is composed of following four components

- Docker Client and Daemon
- Images: Template for creating the environment that you want. It has the OS, software, application code etc. bundled up in a file
- Docker registries: A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default.
- Containers: running instance of an image
- Dockerfile: Images are defined using a dockerfile. It is a text file with a list of steps to perform in order to create that image.
- After writing the dockerfile, you build an image using that dockerfile. And after running the image, you get a container. Images are the basic building blocks of Docker. Containers are built from images. Images can be configured with applications and used as a template for creating containers. Images are organized in a layered manner. Every change in an image is added as a layer on top of it.

2.2.3 Basic Commands in Docker

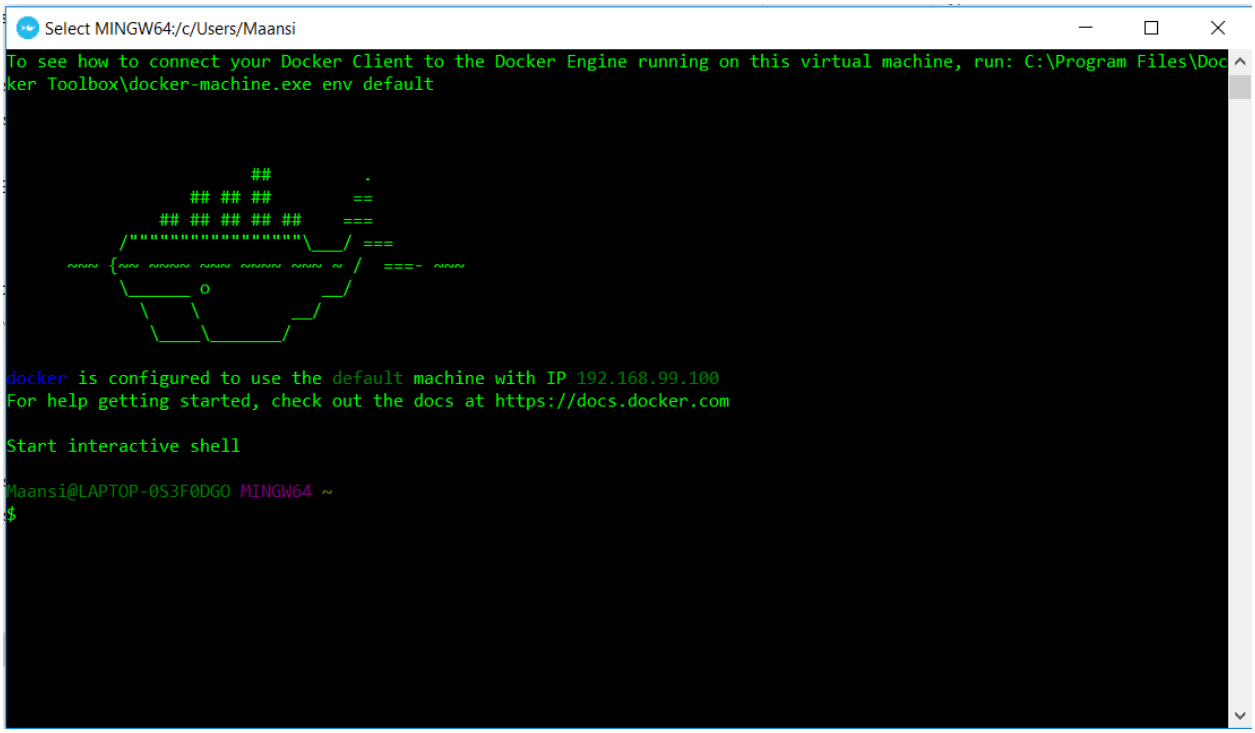
1. ***docker run <image>:*** It will take any image, create a container from the image, and it will start that container. If you use the 'run' command 4 times, 4 containers will be created from that image
2. ***docker start<name|id>:*** starts an existing container that is not running right now
3. ***docker stop<name|id>***: stops a running container. To restart a container, say docker start<name|id>
4. ***docker ps [-a gives all stopped containers]*** : gives a list of all running containers
5. ***docker rm<name|id>***: removes a container
6. ***docker rmi <image name/imageid>*** removes an image

2.3 How to run docker on local machine

- Download docker tool box for your machine
- To check docker installation on system run these commands on Powershell

```
PS C:\Users\Docker> docker -version
```

- Now run the docker Terminal
- Docker Quick Start Terminal



- Run Command

```
docker run hello-world
```

```
Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/


Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$
```

- Now run the Ubuntu Image

  $ `docker run -ti ubuntu bash`

```
Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker run -ti ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
aafe6b5e13de: Pull complete
0a2b43a72660: Pull complete
18bdd1e546d2: Pull complete
8198342c3e05: Pull complete
f56970a44fd4: Pull complete
Digest: sha256:f3a61450ae43896c4332bda5e78b453f4a93179045f20c8181043b26b5e79028
Status: Downloaded newer image for ubuntu:latest
root@7ea20b5676ce:/# pwd
/
root@7ea20b5676ce:/#
```

- To check the images on docker machine

  $ `docker images`

  List Docker Images

```
Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              f7b3f317ec73        2 weeks ago         117MB
hello-world         latest              48b5124b2768        4 months ago        1.84kB

Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$
```

- Start Docker Machine: it will start your machine and generate IP locally.

  `docker-machine start`

```
Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker-machine start
Starting "default"...
Machine "default" is already running.

Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker-machine stop
Stopping "default"...
Machine "default" was stopped.

Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker-machine start
Starting "default"...
(default) Check network to re-create if needed...
(default) Waiting for an IP...
Machine "default" was started.
Waiting for SSH to be available...
Detecting the provisioner...
Started machines may have new IP addresses. You may need to re-run the `docker-machine env` command.


Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$
```

- Stop Docker Machine: It will stop current docker-machine

  `docker-machine stop`

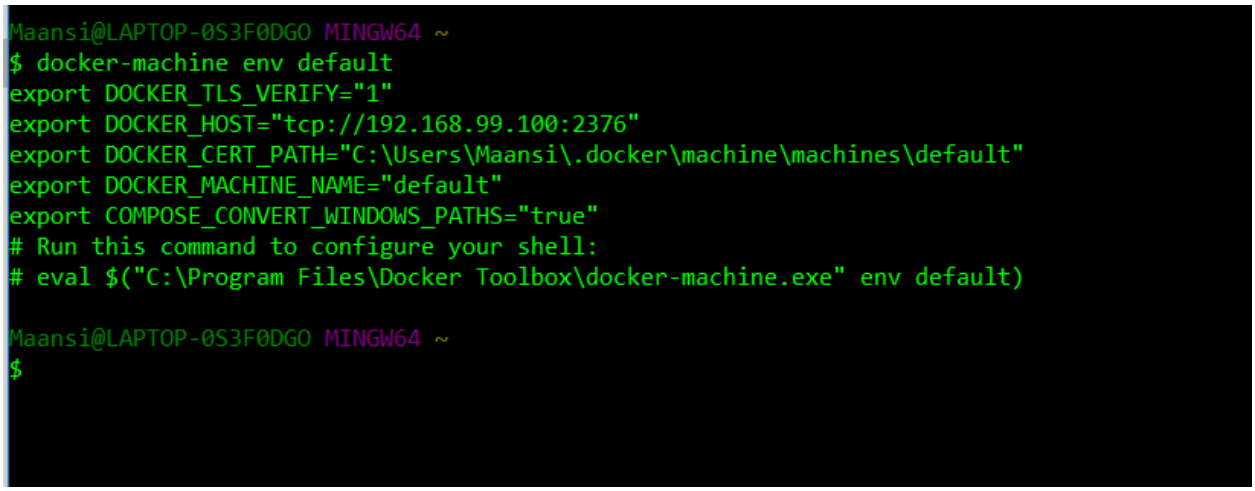
```
Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker-machine stop
Stopping "default"...
Machine "default" was stopped.
```

- Connection to Docker Machine: Using ssh will start the docker for the particular user/docker-machine
  In this case it is ssh as default.

  `docker-machine ssh <Machine-name>`

- Setting docker machine as default

  `docker-machine env default`



- Enabling internet access on docker machine by setting it to no proxy:

  Here the docker machine setting is set to no proxy setting.
  `docker-machine env-no-proxy default`

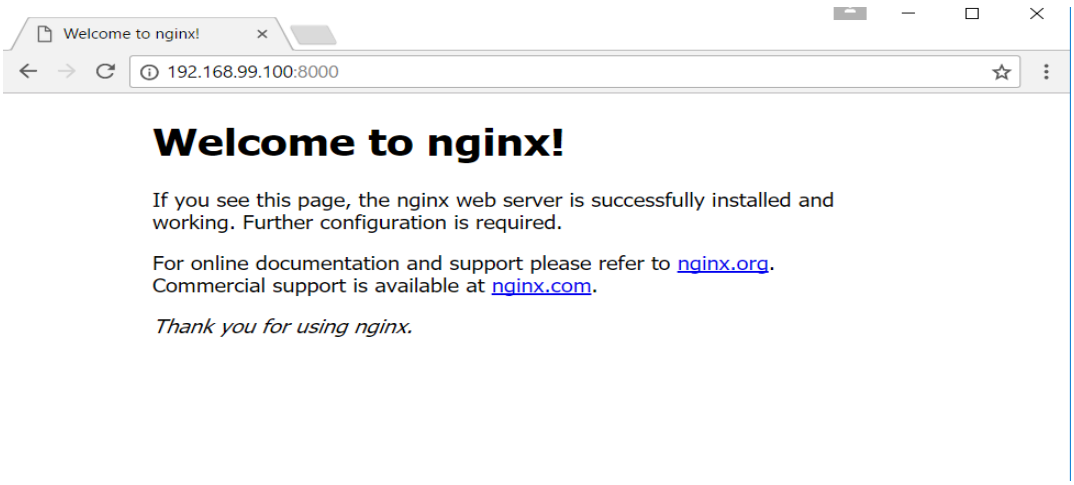
```
Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker-machine env --no-proxy default
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="C:\Users\Maansi\.docker\machine\machines\default"
export DOCKER_MACHINE_NAME="default"
export COMPOSE_CONVERT_WINDOWS_PATHS="true"
export no_proxy="192.168.99.100"
# Run this command to configure your shell:
# eval $("C:\Program Files\Docker Toolbox\docker-machine.exe" env --no-proxy default)

Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$
```

- Running nginx server on docker: Running a small nginx service on docker to run on the given port :8000 locally

  `docker` run -d -p 8000:80 nginx

```
Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$ docker run -d -p 8000:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
ff3d52d8f55f: Pull complete
b05436c68d6a: Pull complete
961dd3f5d836: Pull complete
Digest: sha256:12d30ce421ad530494d588f87b2328ddc3cae666e77ea1ae5ac3a6661e52cde6
Status: Downloaded newer image for nginx:latest
d26ecede676fa64b181ab15bf280320d5a110a14c849cc56977c63d88db61744

Maansi@LAPTOP-0S3F0DGO MINGW64 ~
$
```

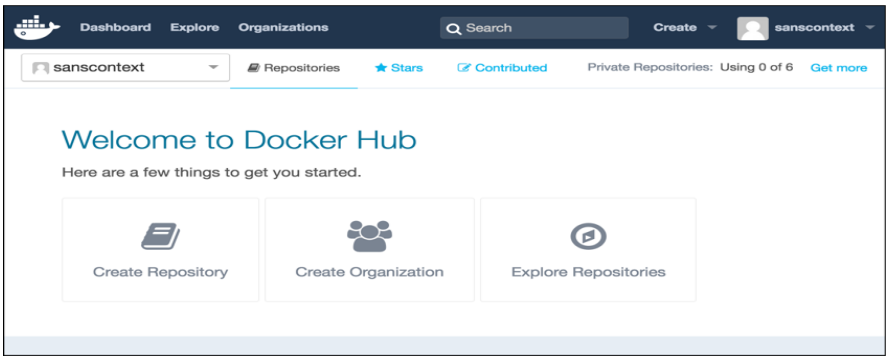
- Connecting to nginx server via browser (IP address and Port number)



2.4 DockerHub

It's a cloud-based registry service which allows which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

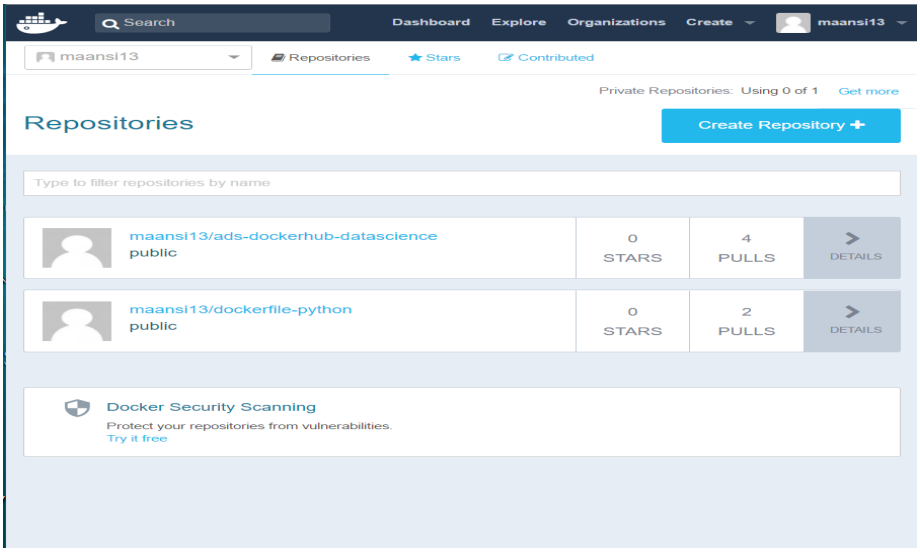
Log in to Docker Hub and Docker Cloud using your free Docker ID.



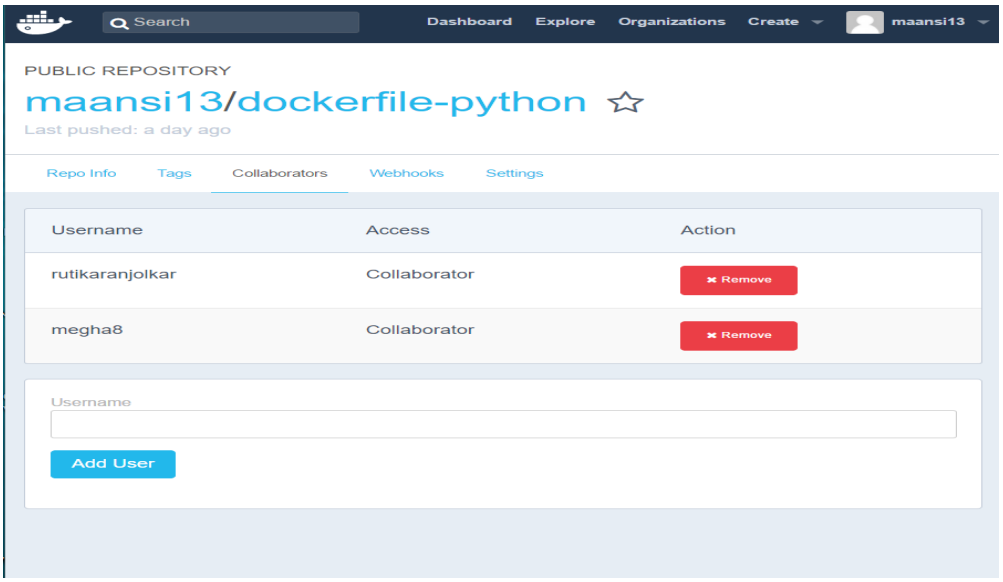
2.4.1 Docker Hub provides the following major features:

- **Image Repositories:** Find and pull images from community and official libraries, and manage, push to, and pull from private image libraries to which you have access.
- **Automated Builds:** Automatically create new images when you make changes to a source code repository.
- **Webhooks:** A feature of Automated Builds, Webhooks let you trigger actions after a successful push to a repository.
- **Organizations:** Create work groups to manage access to image repositories.
- **GitHub and Bitbucket Integration:** Add the Hub and your Docker Images to your current workflows.

Following are the images pushed from docker to docker hub:



Under collaborators tab we can add team members which can then pull images and make changes to the existing images and commit them like this:



3. KUBERNETES

3.1 What is Kubernetes

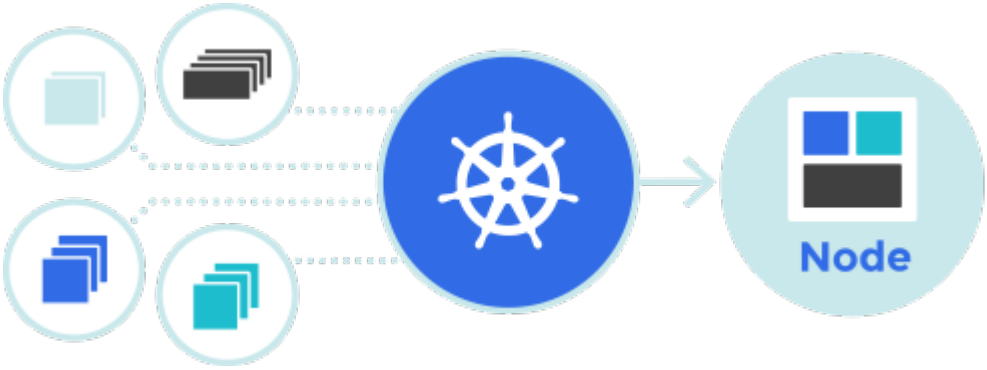
Kubernetes is an open source platform for automating deployment, scaling and operations of application containers across clusters of hosts, providing container-centric infrastructure.

Kubernetes is:

- Portable: public, private, hybrid, multi-cloud
- Extensible: modular, pluggable, composable
- Self-healing: auto-placement, auto-restart, auto-replication, auto-scaling

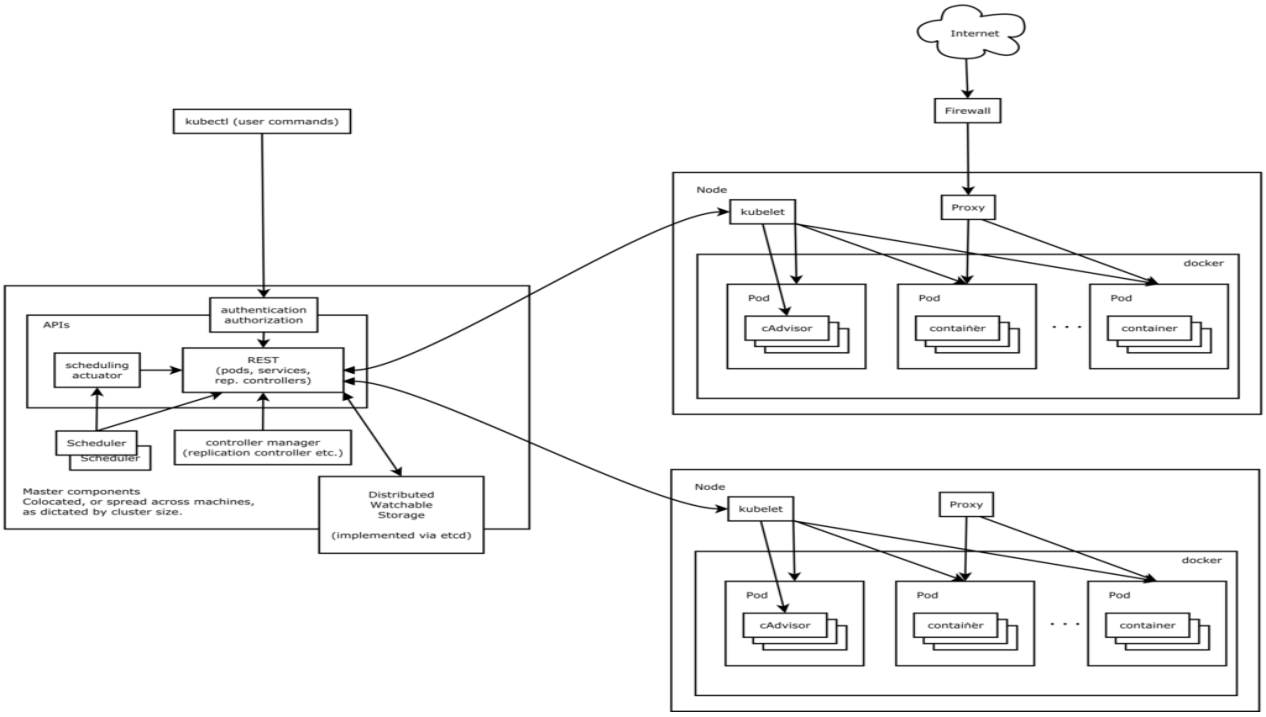
With Kubernetes, we can

- Deploy applications quickly and predictably.
- Scale applications on the fly.
- Roll out new features seamlessly.
- Limit hardware usage to required resources only.
- Our goal is to foster an ecosystem of components and tools that relieve the burden of running applications in public and private clouds.



3.2 Kubernetes Architecture

Kubernetes follows the master-slave architecture. The components of Kubernetes can be divided into those that manage an individual node and those that are part of the control plane. Kubernetes installation consists of both Master and Worker nodes:



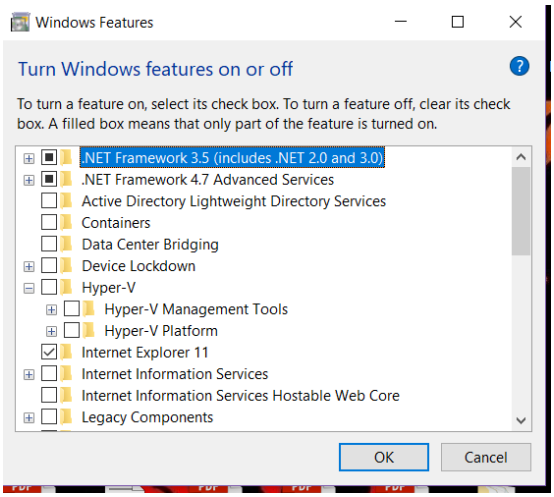
3.2.1 Kubernetes Work Units:

- **Pods:** A pod is the basic unit that Kubernetes deals with and runs on a minion. Closely related containers are grouped together in a pod. It is the basic unit of manipulation in kubernetes.
- **Master:** The master is the host that contain the master components, including the API server, controller manager server. The master manages nodes in its Kubernetes cluster and schedules pods to run on nodes.
- **Node:** A node provides the runtime environments for containers. Each node in a Kubernetes cluster has the required services to be managed by the master. Nodes also have the required services to run pods, including the Docker service, a kubelet, and a service proxy.
- **Kubelet**: Each node has a kubelet that updates the node as specified by a container manifest, which is a YAML file that describes a pod. The kubelet uses a set of manifests to ensure that its containers are started and that they continue to run.
- **Replication Controller**: It ensures that the requested number of pods are running on minions at all times.
- **Label:** An arbitrary key/value pair that the Replication Controller uses for service discovery
- **kubectl**: The command line config tool
- **Service**: An endpoint that provides load balancing across a replicated group of pods

3.3 Kubernetes on local machine

3.3.1 Prerequisites:

- Download VirtualBox for your system from www.virtualbox.org
- Disable Hyper-v from windows10 as the container uses its own vm and it does not go well with VirtualBox, hence you will need to disable it. Go to Windows features on or off. A dialog with list of Windows features as shown below. Navigate to the Hyper-V section and disable it completely
- Minikube
- Kubectl



Note: Enable VT-X/Amd-v in BIOS

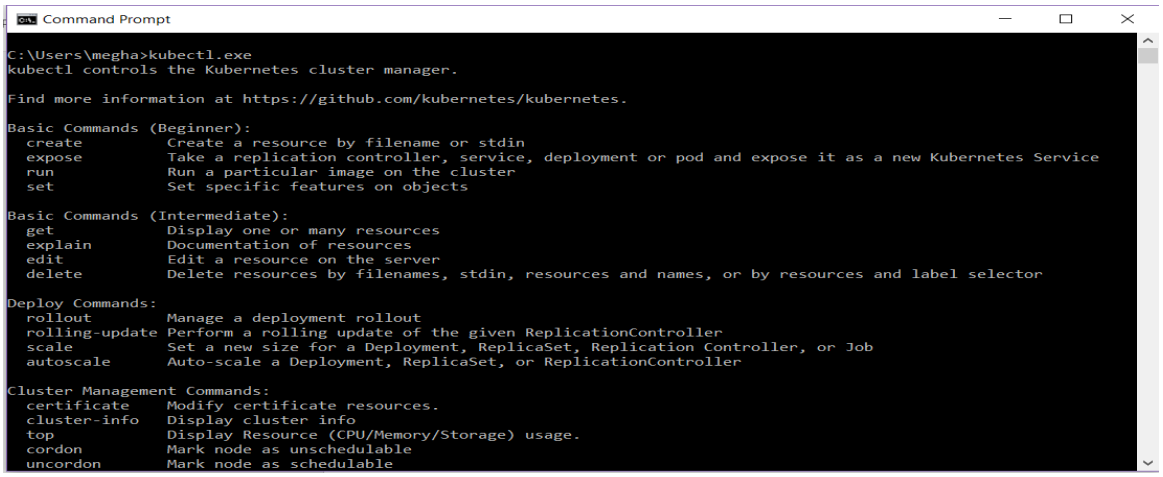
3.3.2 Steps:

- Install kubectl:

  http://storage.googleapis.com/kubernetes-release/release/v0.16.1/bin/windows/amd64/kubectl.exe
  to download the .exe file.
- Place that in the C:\ folder and make it available in the environment PATH variable.

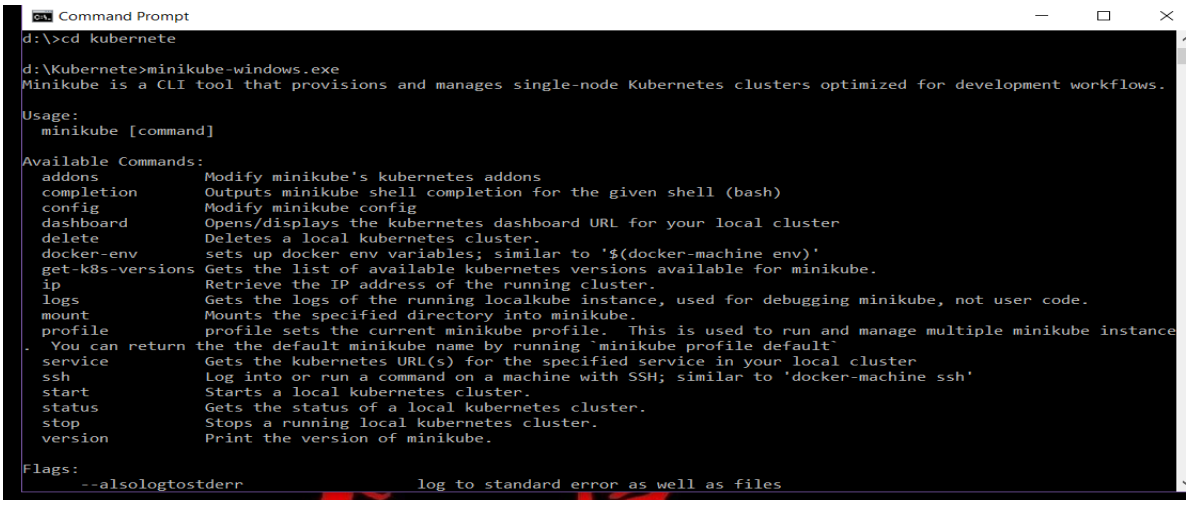
- Download kubctl:

- Run kubectl from cmd.exe from the defined path

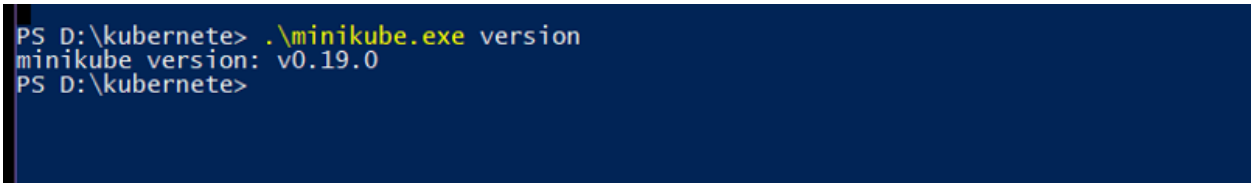


- Minikube:

  https://github.com/kubernetes/minikube/releases
  run minikube-windows.exe using cmd



- Check installed minikube on powershell

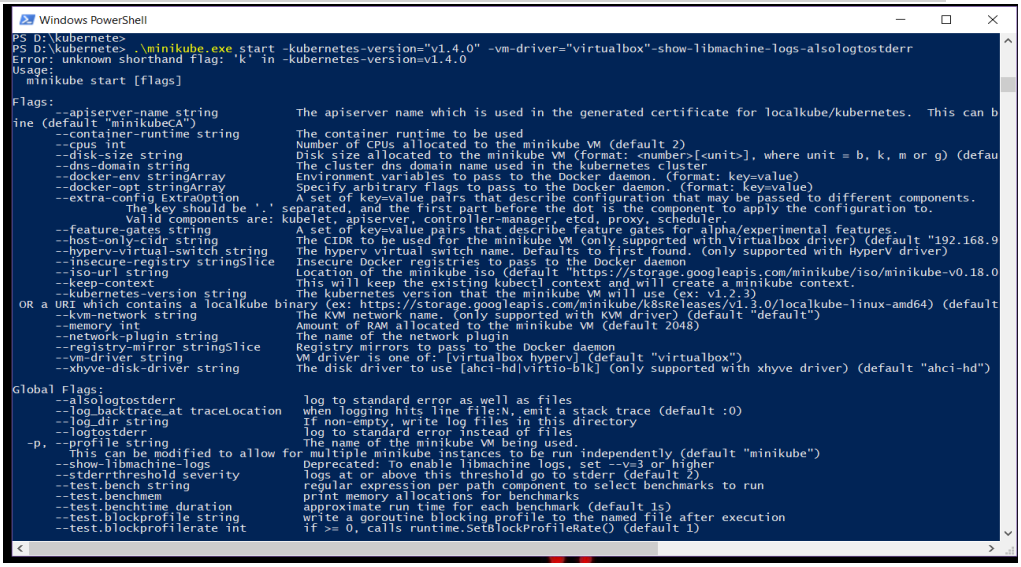


You are all set to launch a local Kubernetes single node cluster but before let's check if minikube is installed correctly.

3.4 Running Kubernetes using minikube locally:

On powershell: run the following command for cluster

```
.\minikube.exe start --kubernetes-version="v1.4.0" --vm-driver="virtualbox" --show-libmachine-logs -alsologtostderr
```



It prints out a message that kubectl is configured to talk to your local Kubernetes cluster.

Check if the Cluster is running

Execute the following command to check the status of the cluster:

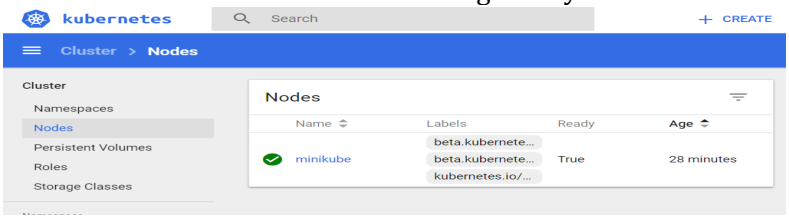
- To Check the status of minikube if it is running
  `.\minikube.exe status`

- To check the ip of minikube
  `.\minikube.exe ip`

- To open minikube dashboard on local machine
  `.\minikube.exe dashboard`

```
Opening kubernetes dashboard in default browser...
PS C:\WINDOWS\system32> .\minikube.exe dashboard
Opening kubernetes dashboard in default browser...
PS C:\WINDOWS\system32> .\minikube.exe version
minikube version: v0.19.1
PS C:\WINDOWS\system32> .\minikube.exe ip
192.168.99.100
PS C:\WINDOWS\system32>
```

This ensures the kubernetes is running locally via minikube on your system.



3.5 Running jupyter notebook on kubernetes locally:

Step 1: minikube start

```
C:\WINDOWS\system32>minikube start
Starting local Kubernetes v1.6.4 cluster...
Starting VM...
Downloading Minikube ISO
 89.51 MB / 89.51 MB [============================================]
Moving files into cluster...
Setting up certs...
Starting cluster components...
Connecting to cluster...
Setting up kubeconfig...
Kubectl is now configured to use the cluster.

C:\WINDOWS\system32>
```

Step 2: Check minikube is running on local machine

```
minikube version
minikube ip
minikube status
```

Step 3: Now install pip on kubernetes to run Jupyter Notebooks

```
Pip install kubernetes
```

```
PS C:\WINDOWS\system32> pip install kubernetes
Collecting kubernetes
  Downloading kubernetes-2.0.0-py2.py3-none-any.whl (718kB)
    100% |████████████████████████████████| 727kB 369kB/s
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from kubernetes)
Collecting ipaddress (from kubernetes)
  Downloading ipaddress-1.0.18.tar.gz
Collecting oauth2client (from kubernetes)
  Downloading oauth2client-4.1.0-py2.py3-none-any.whl (185kB)
    100% |████████████████████████████████| 194kB 3.1MB/s
Collecting certifi (from kubernetes)
  Downloading certifi-2017.4.17-py2.py3-none-any.whl (375kB)
    100% |████████████████████████████████| 378kB 2.0MB/s
Requirement already satisfied: pyyaml in c:\programdata\anaconda3\lib\site-packages (from kubernetes)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib\site-packages\setuptools-27.2.0-py3.6.egg (from kubernetes)
Collecting urllib3>=1.21 (from kubernetes)
  Downloading urllib3-1.21.1-py2.py3-none-any.whl (131kB)
    100% |████████████████████████████████| 133kB 4.0MB/s
Collecting websocket-client (from kubernetes)
  Downloading websocket_client-0.40.0.tar.gz (196kB)
    100% |████████████████████████████████| 204kB 3.3MB/s
Requirement already satisfied: python-dateutil in c:\programdata\anaconda3\lib\site-packages (from kubernetes)
Collecting pyasn1>=0.1.7 (from oauth2client->kubernetes)
  Downloading pyasn1-0.2.3-py2.py3-none-any.whl (53kB)
    100% |████████████████████████████████| 61kB 3.7MB/s
Collecting pyasn1-modules>=0.0.5 (from oauth2client->kubernetes)
  Downloading pyasn1_modules-0.0.9-py2.py3-none-any.whl (60kB)
    100% |████████████████████████████████| 61kB 3.2MB/s
Collecting httplib2>=0.9.1 (from oauth2client->kubernetes)
  Downloading httplib2-0.10.3.tar.gz (204kB)
    100% |████████████████████████████████| 204kB 3.4MB/s
Collecting rsa>=3.1.4 (from oauth2client->kubernetes)
  Downloading rsa-3.4.2-py2.py3-none-any.whl (46kB)
    100% |████████████████████████████████| 51kB 3.5MB/s
Building wheels for collected packages: ipaddress, websocket-client, httplib2
  Running setup.py bdist_wheel for ipaddress ... done
  Stored in directory: C:\Users\megha\AppData\Local\pip\Cache\wheels\d7\f9\02\033feb2ff3e12e6d0682a5cd38c07a17e78d361c29ee52d61e
  Running setup.py bdist_wheel for websocket-client ... done
  Stored in directory: C:\Users\megha\AppData\Local\pip\Cache\wheels\d1\5e\dd\93da015a0ecc8375278b05ad7f0452eff574a044bcea2a95d2
  Running setup.py bdist_wheel for httplib2 ... done
  Stored in directory: C:\Users\megha\AppData\Local\pip\Cache\wheels\ca\ac\5f\749651f7925b231103f5316cacca82a487810c22d30f011c0c
Successfully built ipaddress websocket-client httplib2
Installing collected packages: ipaddress, pyasn1, pyasn1-modules, httplib2, rsa, oauth2client, certifi, urllib3, websocket-client, kubernetes
Successfully installed certifi-2017.4.17 httplib2-0.10.3 ipaddress-1.0.18 kubernetes-2.0.0 oauth2client-4.1.0 pyasn1-0.2.3 pyasn1-modules-0.0.9 rsa-3.4.2
urllib3-1.21.1 websocket-client-0.40.0
```

Step 4: Check minikube status

```
PS C:\WINDOWS\system32> .\minikube.exe status
minikube: Running
localkube: Running
```

Step 5: Now Install jupyter on Kubernetes using pip

```
Pip install jupyter
```

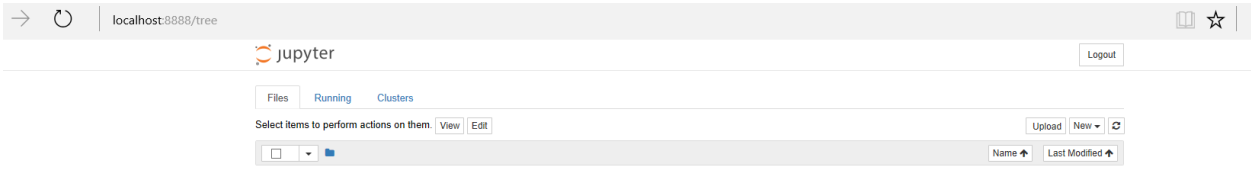
```
PS C:\WINDOWS\system32> pip install jupyter
Requirement already satisfied: jupyter in c:\programdata\anaconda3\lib\site-packages
PS C:\WINDOWS\system32> pip install jupyter
```

Step 6: Now run jupyter notebook on minikube

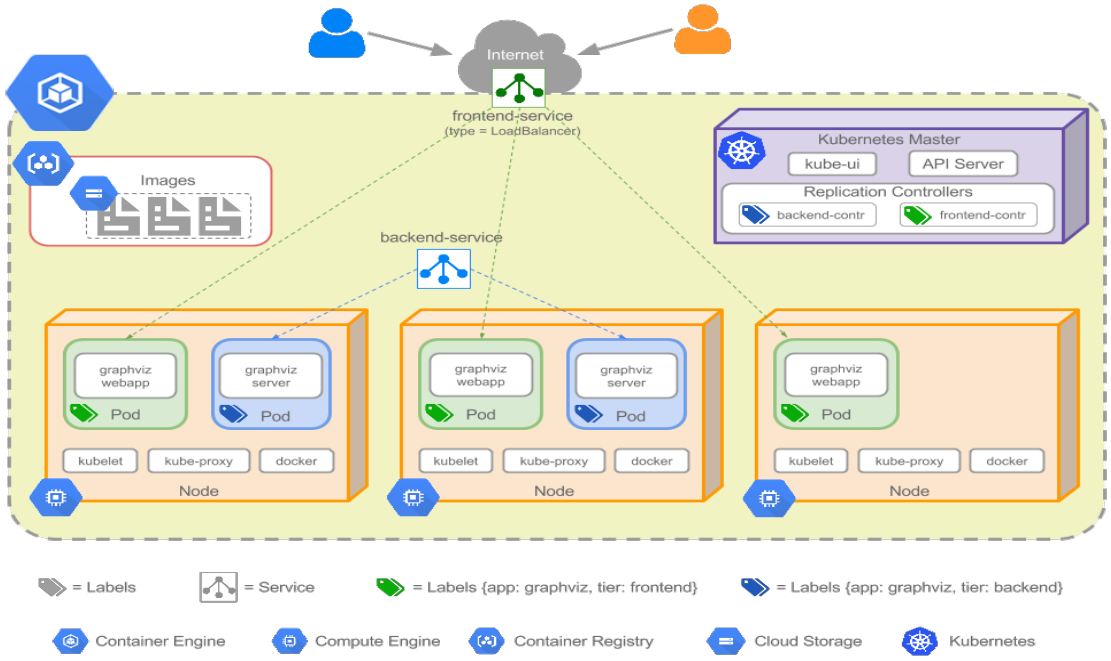
```
Jupyter notebook
```

```
PS C:\WINDOWS\system32> jupyter notebook
[I 22:28:15.771 NotebookApp] Serving notebooks from local directory: C:\WINDOWS\system32
[I 22:28:15.771 NotebookApp] 0 active kernels
[I 22:28:15.772 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/?token=0e8c861c43c5ce7532750a0e20f198d0b6465f5357a0c530
[I 22:28:15.776 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 22:28:15.778 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=0e8c861c43c5ce7532750a0e20f198d0b6465f5357a0c530
[I 22:28:16.325 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

Step 7: It will open on your local browser:



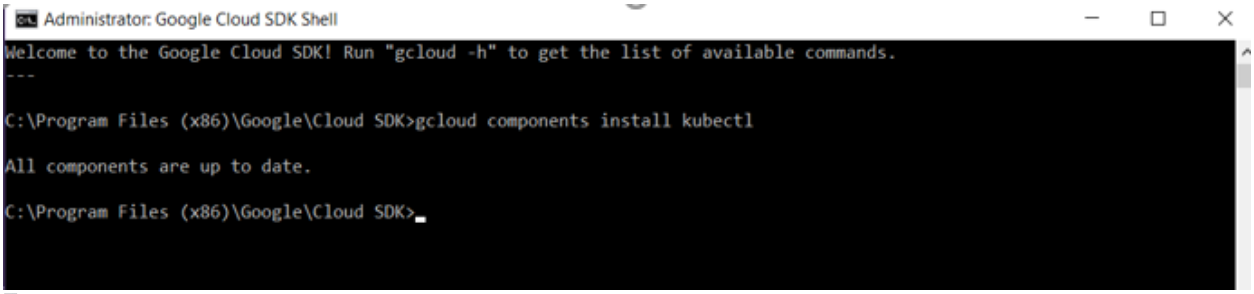
3.6 Kubernetes on Google Container Engine

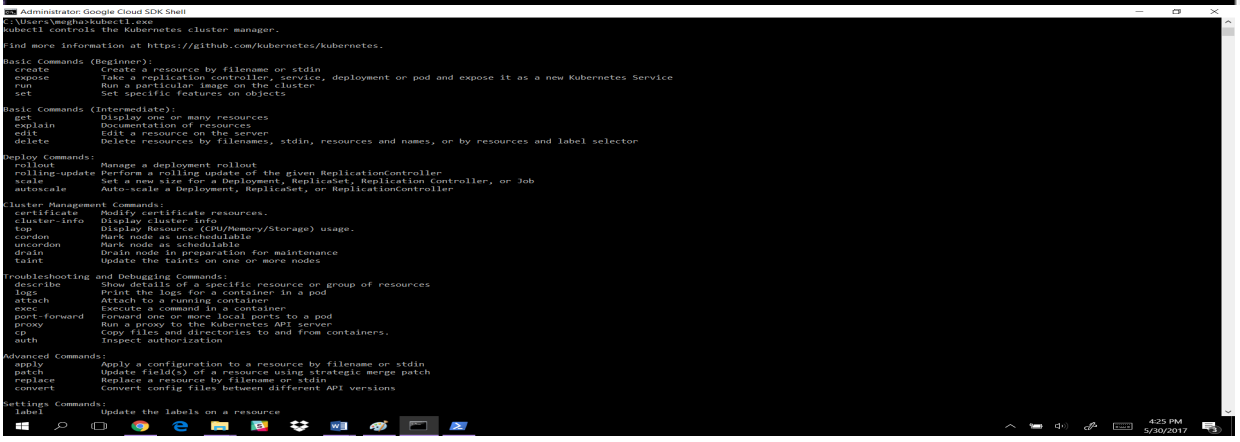


On Googlecloud:

Run on gcloud sdk

```
gcloud components install kubectl
```





OR **Run on gcloud**

Run The following commands

Install kubectl on gcloud

```
gcloud components install kubectl
```

```
megha_singh8008@atomic-graph-169317:~$ gcloud components install kubectl

All components are up to date.
megha_singh8008@atomic-graph-169317:~$
```

- We can create a pod by using the following command. This command starts up the docker image on one of the nodes in the cluster.

  Kubectl is setup !!

- Create the container

```
gcloud container clusters create example-cluster
```

- Run the container:

```
kubectl run hello-node --image=gcr.io/google-samples/node-hello:1.0
--port=8080
```

```
megha_singh8008@atomic-graph-169317:~$ kubectl run hello-node --image=gcr.io/google-samples/node-hello:1.0 --port=8080
deployment "hello-node" created
megha_singh8008@atomic-graph-169317:~$
```

- Expose the node

```
kubectl expose deployment hello-node --type="LoadBalancer"
```

```
megha_singh8008@atomic-graph-169317:~$ kubectl expose deployment hello-node --type="LoadBalancer"
service "hello-node" exposed
megha_singh8008@atomic-graph-169317:~$ kubectl get service hello-node
NAME         CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
hello-node   10.7.254.73   <pending>     8080:30583/TCP   15s
megha_singh8008@atomic-graph-169317:~$
```

Note: You might need to wait several minutes to get an external IP address. If you don't get an external IP address, run

```
gcloud config list project
```

```
Welcome to Cloud Shell! Type "help" to get started.
megha_singh8008@atomic-graph-169317:~$ gcloud config list project
[core]
project = atomic-graph-169317

Your active configuration is: [cloudshell-8732]
megha_singh8008@atomic-graph-169317:~$
```

`gcloud auth login`

```
Your active configuration is: [cloudshell-8732]
megha_singh8008@atomic-graph-169317:~$ gcloud auth login

         You are already authenticated with gcloud when running
         inside the Cloud Shell and so do not need to run this
         command.

         Do you wish to proceed anyway?


Do you want to continue (Y/n)?  Y

Go to the following link in your browser:

    https://accounts.google.com/o/oauth2/auth?redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&prompt=select_account&response_
type=code&client_id=32555940559.apps.googleusercontent.com&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+http
s%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fw
ww.googleapis.com%2Fauth%2Fcompute+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Faccounts.reauth&access_type=offline


Enter verification code:
```

- We get the number of pods running

`kubectl get pods`

```
megha_singh8008@atomic-graph-169317:~$ kubectl get pods
NAME                                 READY     STATUS    RESTARTS   AGE
demo-deployment-2988296882-233tk     1/1       Running   0          3h
hello-node-3526609615-63cph          1/1       Running   0          3h
megha_singh8008@atomic-graph-169317:~$
```

3.7 To clean up the unused services on gcloud

To avoid incurring charges to your Google Cloud Platform account for the resources used in this quickstart:

- Delete the services you created to remove the load balancing resources that are provisioned to serve them:

  `kubectl delete service hello-node`

  ```
  megha_singh8008@atomic-graph-169317:~$ kubectl delete service hello-node
  service "hello-node" deleted
  megha_singh8008@atomic-graph-169317:~$
  ```

- Wait a few minutes for the service to spin down, then use the following command to delete the cluster you created:

```
gcloud container clusters delete example-cluster
```

```
megha_singh8008@atomic-graph-169317:~$ gcloud container clusters delete test2
The following clusters will be deleted.
 - [test2] in [us-central1-a]

Do you want to continue (Y/n)?
```

The cluster is deleted

4. VAGRANT

4.1 What is vagrant?

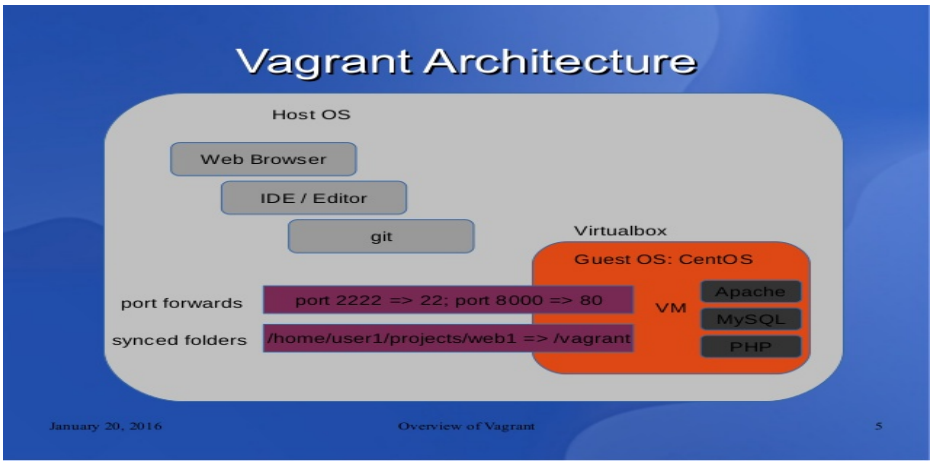
Vagrant is a tool for building and managing virtual machine environments in a single workflow. With an easy-to-use workflow and focus on automation, Vagrant lowers development environment setup time, increases production parity.

Vagrant provides easy to configure, reproducible, and portable work environments built on top of industry-standard technology and controlled by a single consistent workflow to help maximize the productivity and flexibility of you and your team.

4.2 Benefits of Vagrant

- Unified workflow
- Simple and Powerful
- Vagrant provides the same, easy workflow regardless of work as a developer, operator, or designer. It leverages a declarative configuration file which describes all your software requirements, packages, operating system configuration, users, and more.
- Enforce consistency
- Production Parity
- Vagrant aims to mirror production environments by providing the same operating system, packages, users, and configurations, all while giving users the flexibility to use their favorite editor, IDE, and browser.
- Cross-Platform
- Works where you work
- Vagrant works on Mac, Linux, Windows, and more. Remote development environments force users to give up their favorite editors and programs.
- Easily code in your favorite text editor, edit images in your favorite manipulation program, and debug using your favorite tools, all from the comfort of your local laptop.

4.3 Vagrant Architecture

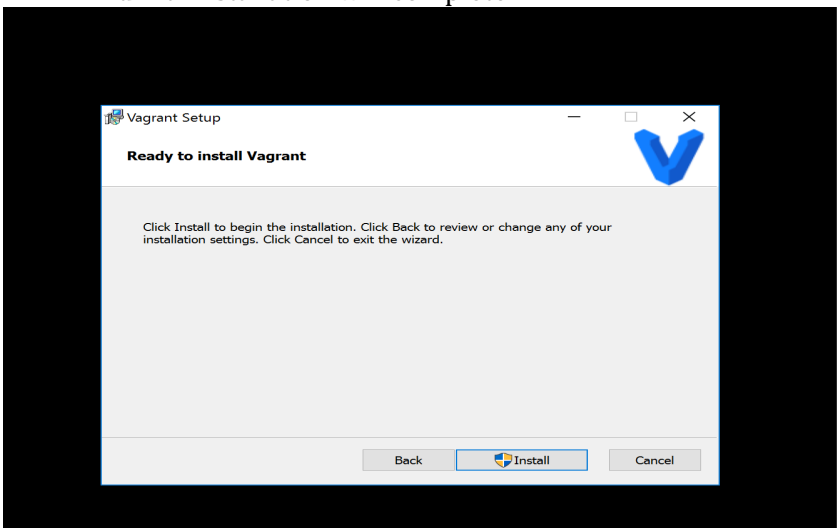


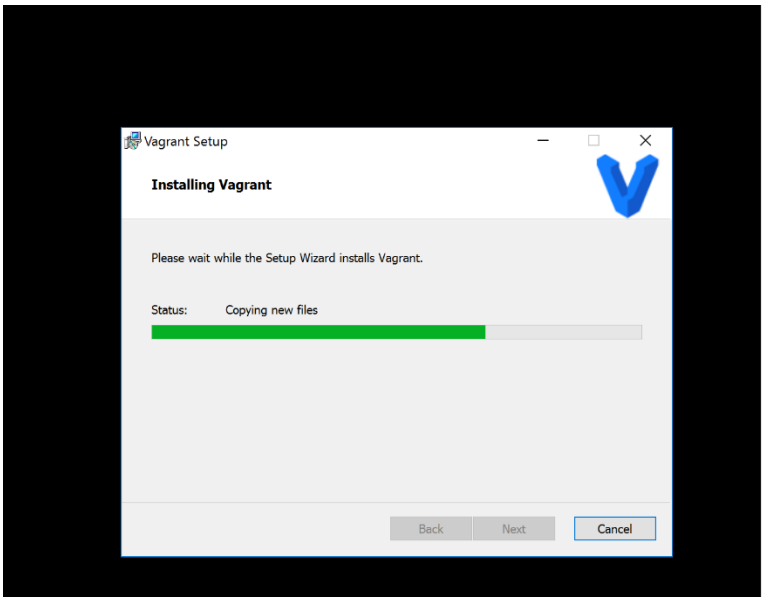
4.3.1 Components of Vagrant

- **Box:** A box is a packaged Vagrant environment, typically a virtual machine.
- **Provider**: A provider is the location in which the virtual environment runs. It can be local (the default is to use VirtualBox), remote, or even a special case like a Docker container.
- **Provisioner**: A provisioner is a tool to set up the virtual environment, and can be as simple as a shell script, but alternatively a more advanced tool like Chef, Puppet, or Ansible can be used.

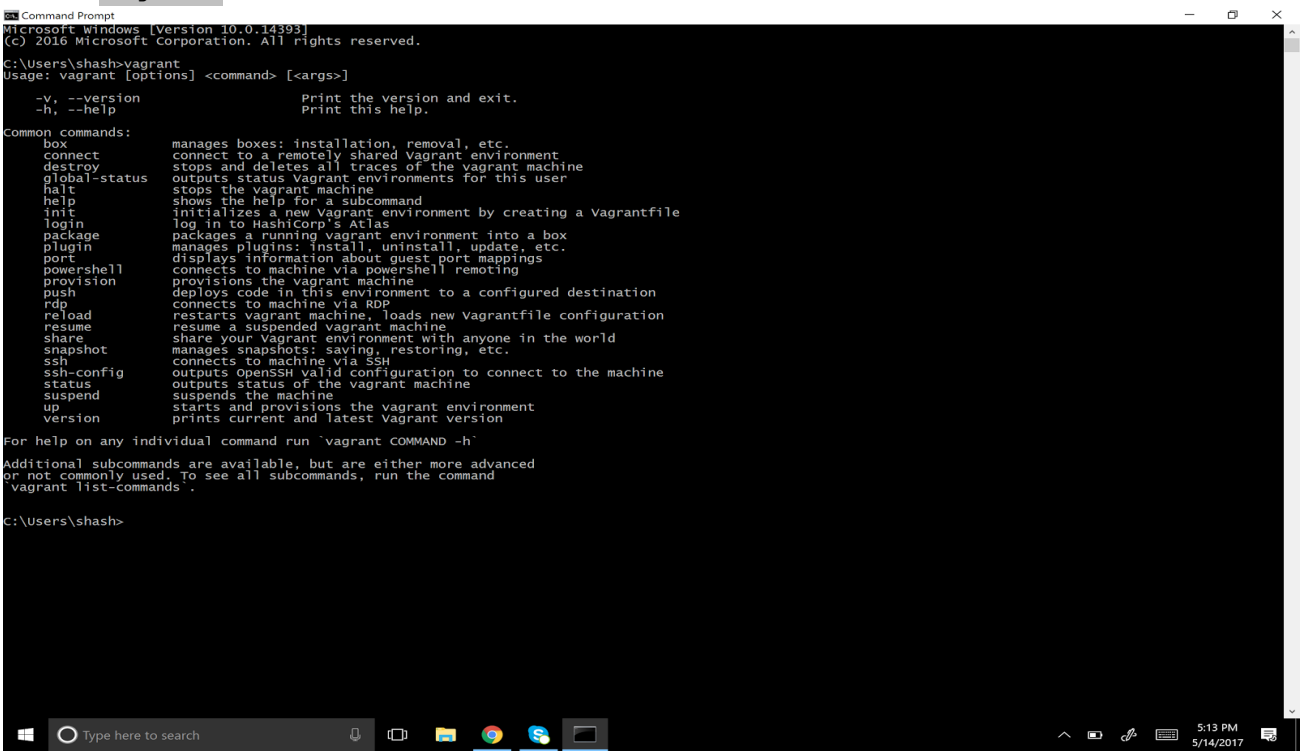
4.3.2 Vagrant on local machine.

- First, Download the vagrant file from the given link on your system https://www.vagrantup.com/downloads.html
- Download .msi file
- Run it- installation will complete

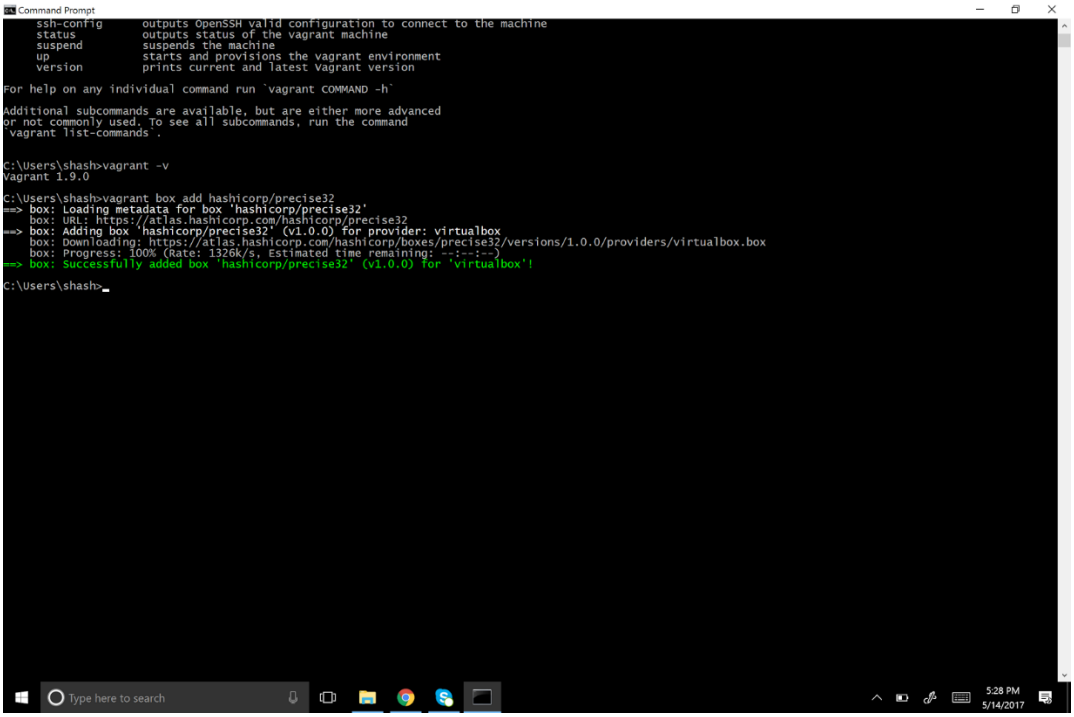




- Restart the machine, if necessary
- On the command prompt, verify the installation `vagrant`



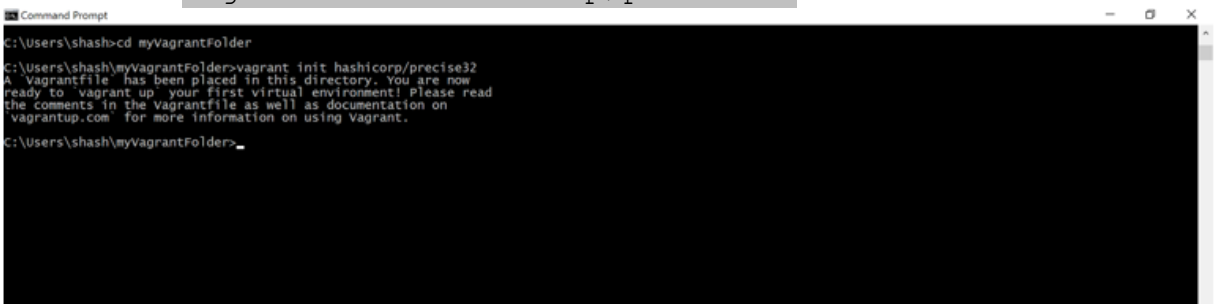
- Install Box on the system. We choose 32-bit box since it consumes less RAM on the computer.
- The command `vagrant box add hashicorp/precise32` downloads the box image on the machine. We can now install as many 32 bit boxes as we want, as long as the machine supports it.



- We now initialize one 32 mbit box from inside the folder.

```
cd myVagrantFolder
```

```
vagrant init hashicorp/precise32
```



Fire up the instance using the following command:

```
vagrant up
```

- After sshing in the machine, try `ls /vagrant/`

It should give 'Vagrantfile' as the output.

If u dont get it, there's a mount error.
Try `sudo mount -t vboxsf vagrant /vagrant`

- The vagrant file is created inside the folder.

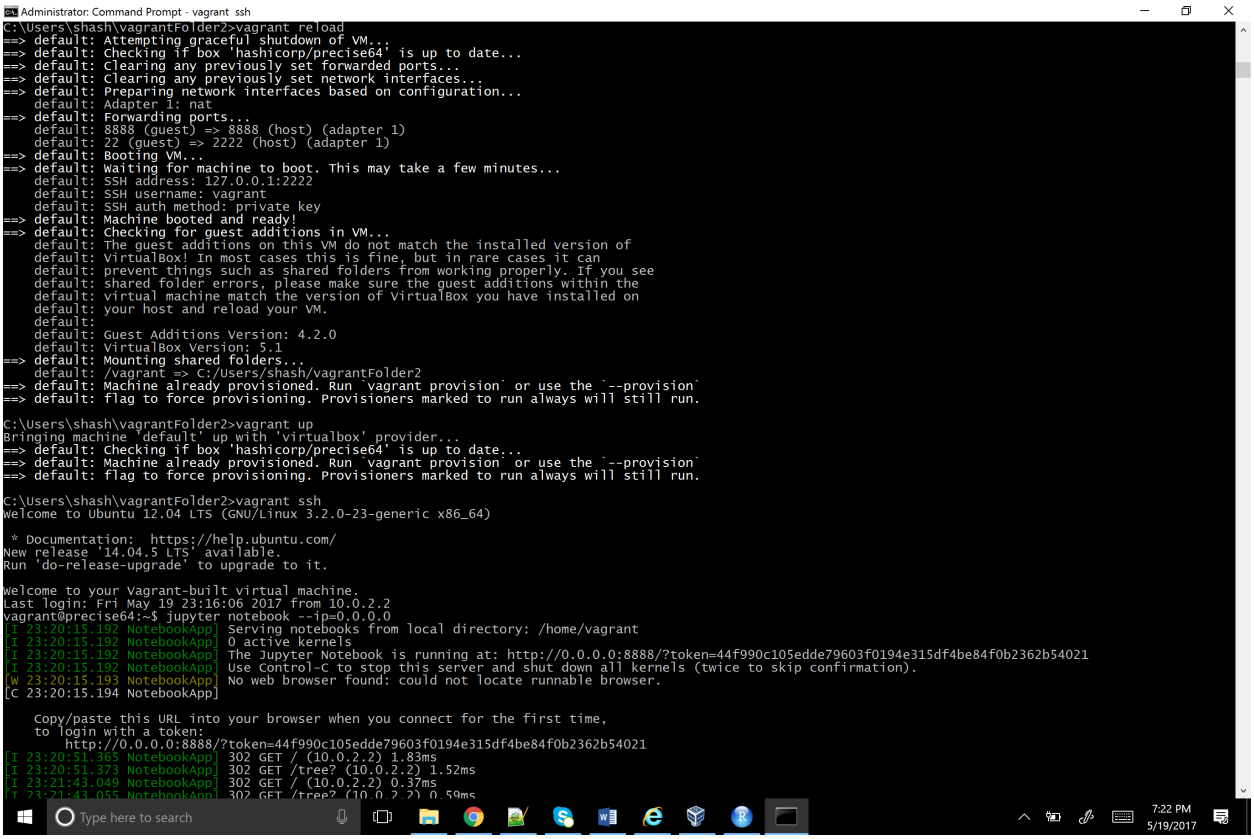
```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented belo
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search fo
  # boxes at https://atlas.hashicorp.com/search.
  config.vm.box = "hashicorp/precise64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example belo
  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 8888, host: 8888

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  # config.vm.network "private_network", ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
```

- By default, Vagrant shares the project directory (the directory with the Vagrantfile) to /vagrant inside the virtual machine. After SSHing into the virtual machine, this can be verified by listing the files in that directory:

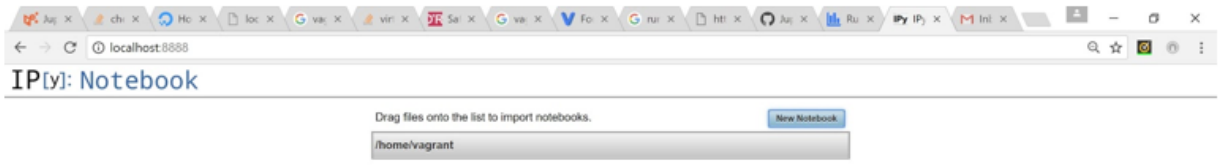
```
vagrant@precise32:~$ ls /vagrant/Vagrantfile
```

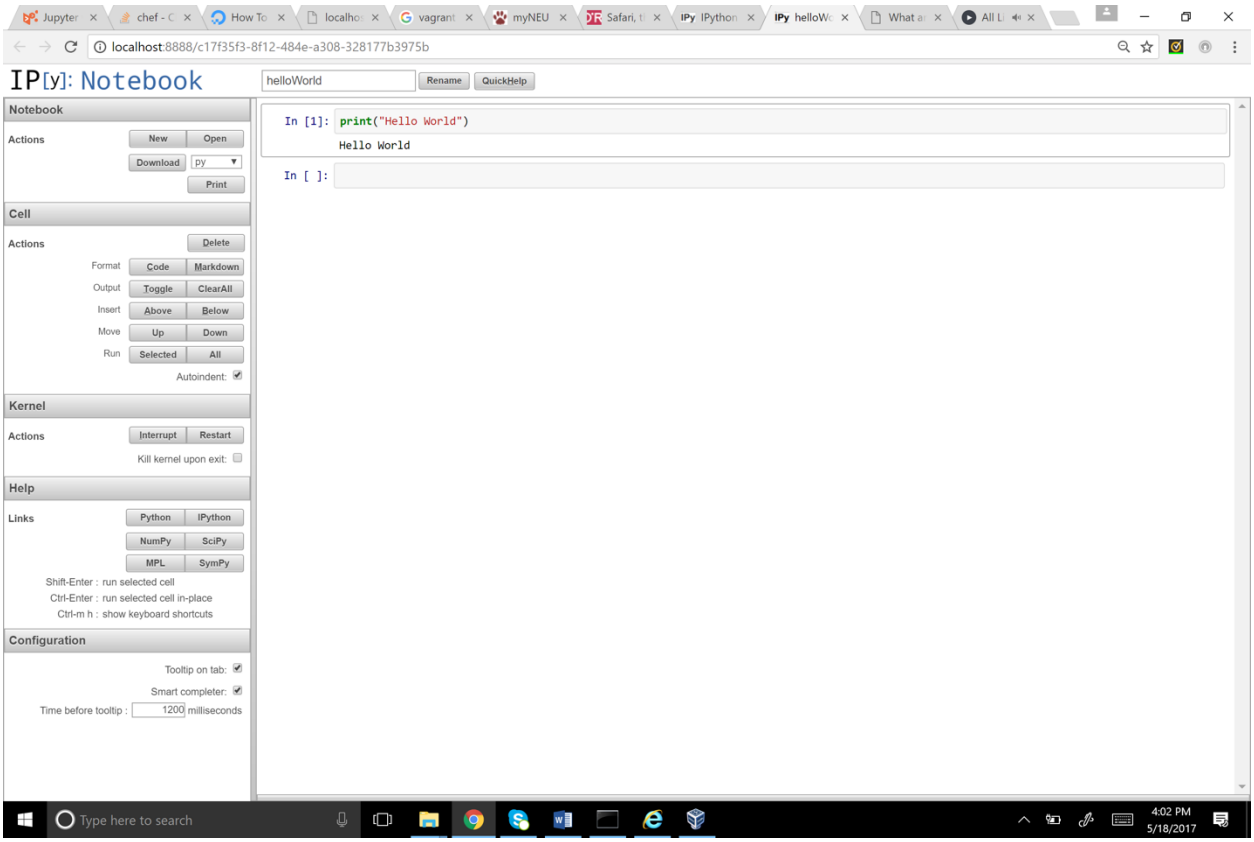
The Vagrant file in that directory is a Vagrantfile from the project directory. If a file is created on the host machine or within the Vagrant machine, the changes will be mirrored from host to guest and vice versa.

- Install python and jupyter notebooks after sshing in the guest machine.



- Log in to the localhost url with the set port number.





5. DOCKER SWARM

Swarm mode exists natively for Docker Engine, the layer between the OS and container images. Swarm mode integrates the orchestration capabilities of Docker Swarm.

Clustering: An important feature for container technology as it creates a cooperative group of systems that can provide redundancy, enabling Docker Swarm failover if one or more nodes experience an outage.

5.1 Features of Docker Swarm:

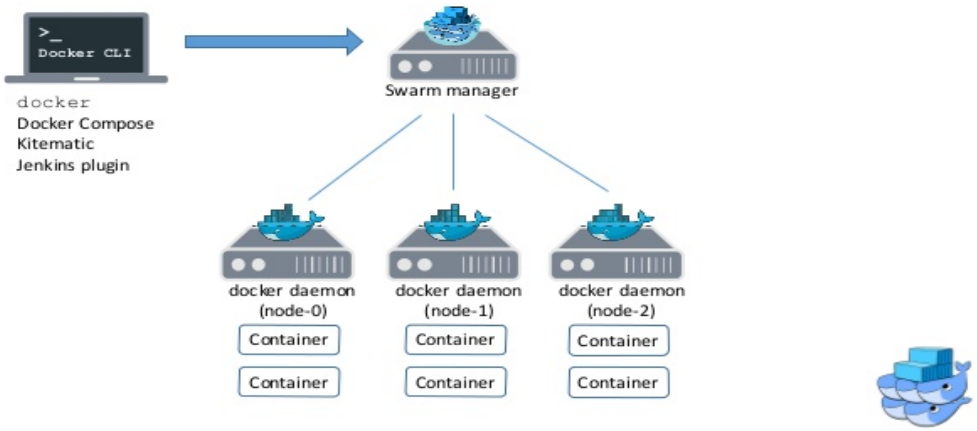
- Its cluster provides administrators and developers with the ability to add or subtract container iterations as computing demands change.
- It has two main components: Manager and Worker
- Swarm Manager Node: Allows a user to create a primary manager instance and multiple replica instances in case the primary instance fails. It manages the orchestration of tasks(containers) and maintaining Swarm Cluster itself.
- Swarm Worker Node: A worker node is a member of the swarm cluster whose role is to run tasks. Tasks are containers.

5.1.2 Services can be deployed to Swarm

- A service is a long-running docker container that can be deployed to any node worker.
- Service is something that either remote systems or other containers within the swarm can connect to and consume.

5.2 Swarm Architecture

1. Cluster management integrated with Docker Engine
2. Decentralized design
3. Declarative service model
4. Scaling and Load Balancing
5. Desired state reconciliation



5.3 Steps to create Docker swarm

5.3.1 Activities performed in docker swarm

- initializing a cluster of Docker Engines in swarm mode
- adding nodes to the swarm
- deploying application services to the swarm
- managing the swarm once you have everything running

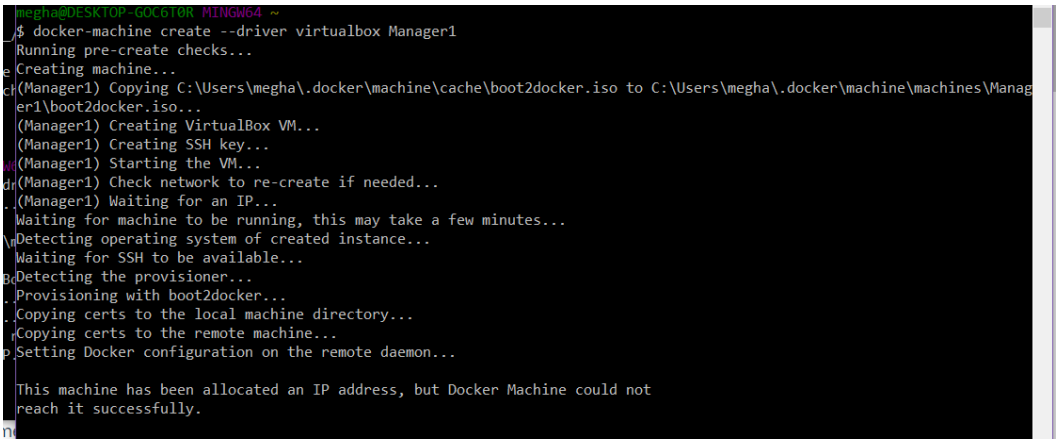
- Create the Vm machines for all nodes:
  Manager Machine: This command will create a new Manager Machine on your VM

```
docker-machine create --driver virtualbox Manager1
```

To check the machine running on the docker we use the command:

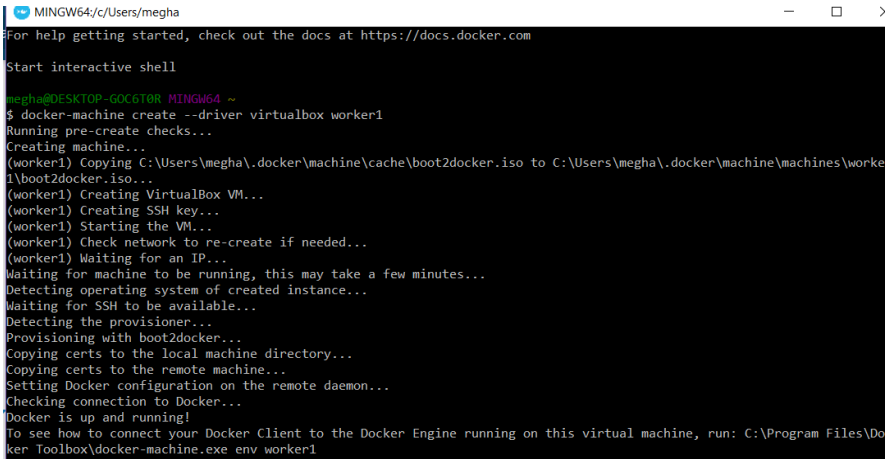
```
docker-machine ls
```

- After creating the Manager Machine which will be the leader/master of the swarm



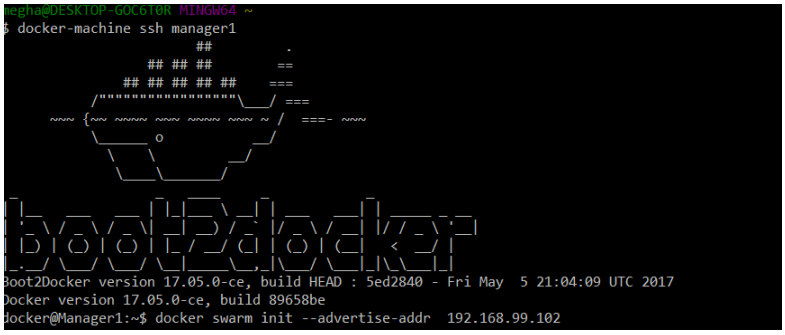
- Similarly create worker1 and worker2 as the slave nodes in swarm
    - Create Worker1 and worker 2

```
docker-machine create --driver virtualbox worker1
docker-machine create --driver virtualbox worker2
```

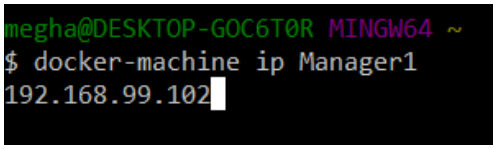


- SSH on Manager to create swarm

  `docker-machine ssh manager1`

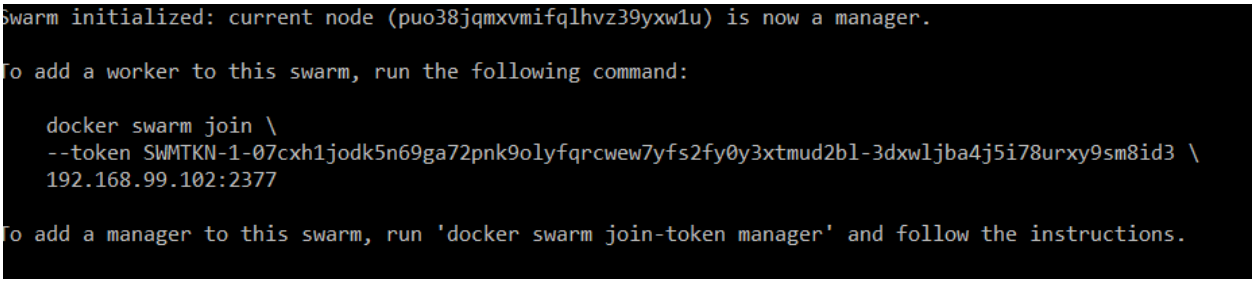


  `docker-machine ip`



- Run the following command to create a new swarm:

  `docker swarm init --advertise-addr <MANAGER-IP> {Here it is 192.168.99.102}`



- Check the manager node joined the swarm
- The following command creates a swarm on the manager1 machine:

```
$ docker swarm init --advertise-addr 192.168.99.101
```

```
docker@Manager1:~$ docker swarm init --advertise-addr  192.168.99.102
Swarm initialized: current node (puo38jqmxvmifqlhvz39yxw1u) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-07cxh1jodk5n69ga72pnk9olyfqrcwew7yfs2fy0y3xtmud2bl-3dxwljba4j5i78urxy9sm8id3 \
    192.168.99.102:2377

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

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

```
docker@Manager1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-07cxh1jodk5n69ga72pnk9olyfqrcwew7yfs2fy0y3xtmud2bl-cuh61qg19qu6ab6ffdc83tt9i \
    192.168.99.102:2377

docker@Manager1:~$ docker swarm join \
>     --token SWMTKN-1-07cxh1jodk5n69ga72pnk9olyfqrcwew7yfs2fy0y3xtmud2bl-cuh61qg19qu6ab6ffdc83tt9i \
>     192.168.99.102:2377
Error response from daemon: This node is already part of a swarm. Use "docker swarm leave" to leave this swarm and join
another one.
```

```
Docker node ls
```

```
docker@Manager:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
d5tny4i27m9d144gj2juflg1o *   Manager             Ready               Active              Leader
docker@Manager:~$
```

- Similarly, ssh on worker1 and worker2
- Add worker nodes using the command

```
docker swarm join \--token SWMTKN-1-
07cxh1jodk5n69ga72pnk9olyfqrcwew7yfs2fy0y3xtmud2bl-
3dxwljba4j5i78urxy9sm8id3 \ 192.168.99.102:2377
```

```
oot2Docker version 17.05.0-ce, build HEAD : 5ed2840 - Fri May  5 21:04:09 UTC 2017
ocker version 17.05.0-ce, build 89658be
ocker@worker2:~$  docker swarm join \
>     --token SWMTKN-1-1ricwlvzhn0fynaob8z5980wmk59u53z5xh6zn8lzde2x9p8dc-ahd20j2l20kt6s87xsy5d28pq \
>     192.168.99.102:2377
his node joined a swarm as a worker.
ocker@worker2:~$ exit
```

- To check all nodes joined in swarm

```
Docker-machine ls
```

- All the nodes joined in the swarm

```
megha@DESKTOP-GOC6T0R MINGW64 ~
$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
default    *        virtualbox   Running   tcp://192.168.99.101:2376           v17.05.0-ce
Manager1   -        virtualbox   Running   tcp://192.168.99.102:2376           v17.05.0-ce
worker1    -        virtualbox   Running   tcp://192.168.99.103:2376           v17.05.0-ce
worker2    -        virtualbox   Running   tcp://192.168.99.104:2376           v17.05.0-ce
```

5.3.2 Create a service in Docker Swarm

First, ssh on the node where you want to start a service its generally the default/Manager node, but to start or inspect the service can be ssh on worker nodes as well.

- Create a service

```
docker@Manager:~$ docker service create --name my_web nginx
1a195hj3mef25ou8yab7cd7uj
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
docker@Manager:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
c92tr81mvthadk705ekidqvu7     Worker1             Ready               Active              Reachable
cr9mtr3v5rq04d3abvd1mwpx3 *   Manager             Ready               Active              Leader
m5d49sun43dh5n5gf8jh44lwi     Worker2             Ready               Active
docker@Manager:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
docker@Manager:~$ docker service ls
ID                  NAME                    MODE                REPLICAS            IMAGE               PORTS
09w9ox9atqz7        peaceful_stallman       replicated          0/1                 hello
0jddzmgjoi7u        helloworld              replicated          1/1                 alpine:latest
1a195hj3mef2        my_web                  replicated          1/1                 nginx:latest
en86cbv26pd8        determined_ardinghelli  replicated          0/1                 hello
docker@Manager:~$
```

- Start A Service

```
docker@Manager:~$ docker service create --replicas 1 --name helloworld alpine ping docker.com
0jddzmgjoi7u3r3h26y4uf2ca
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
docker@Manager:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
0jddzmgjoi7u        helloworld          replicated          1/1                 alpine:latest
```

- Inspect the service

```
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
0jddzmgjoi7u        helloworld          replicated          1/1                 alpine:latest
docker@Manager:~$ docker service inspect --pretty 0jddzmgjoi7u

ID:             0jddzmgjoi7u3r3h26y4uf2ca
Name:           helloworld
Service Mode:   Replicated
 Replicas:      1
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         alpine:latest@sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c
 Args:          ping docker.com
Resources:
Endpoint Mode:  vip
```

- To check which nodes are running the service:

```
docker@Manager:~$ docker service ps 0jddzmgjoi7u
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE           ERROR
yfpl1rxqqep7        helloworld.1        alpine:latest       Worker1             Running             Running 3 minutes ago
docker@Manager:~$
```

6. WEB APPLICATION USING DOCKER

Web Application: (Client Server communication)

Docker Engine is a client-server application with these major components: A server which is a type of long-running program called a daemon process. A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do. A command line interface (CLI) client.

Web-app: Web services are services that are exposed to the internet for programmatic access. Since microservices are independent components in the web app, we can build each microservice inside a container. This is called dockerization. Simply put, it is the process of modifying an application to run within a Docker container. Each microservice is dockerized.

Docker containers that comprise of microservices are then connected in order to create a complete web app.

6.1 How to Dockerize the web application

Step1: A clear vision and understanding of the app is required before beginning the process. Define the nature of the app, the inputs, deliverables and individual components.

Step2: Choose the best suited technologies for each of the microservice and its components.

Step3: Create a Dockerfile to customize each container. This dockerfile will have an underlying base image/images from DockerHub. A great way to start out to use one of these as a base in the FROM statement. It is like inheriting the image for your app.

Specific customizations can be added after adding the base image. The Dockerfile will contain all the necessary commands in order to build a custom container.

A word of caution about the base images! Be careful about what you pull from Docker Hub and be sure to review the code for malicious content. Anyone can push to Docker Hub!

Step 4: Creating additional code files to go with the Dockerfile. Any dependencies or code are included in these files.

Step5: Create an image for each of the microservices. It is good practice to tag and push these images on DockerHub for reuse and version control.

Step 6: Run the images to make containers for each microservice.

Step 7: Link the containers using the *"link"* command

6.2 'Greetings, User!'

For our example, we are building a web app *'Greetings, User!'* that takes a username and password and verifies it with a preset database. If the unique combination of the username and password is present in the table, our web app will return a specific message directed to that user.

6.2.1 Steps for Server-Client Communication on docker

For the client-server communication for web-app following steps are performed:
The individual microservices in this app are:

1. authentication and
2. database.

For the *'Greetings, User!'* web app, we choose MySQL as an ideal candidate for database operations. For authentication and front end, Node.js is chosen.

6.3 Steps to create microservice in Greetings User

6.3.1 Database: Create a database microservice in docker

- Creating a Dockerfile:

```
FROM mysql
RUN mkdir -p /docker-entrypoint-initdb.d
ADD mysql-init.sql /docker-entrypoint-initdb.d/
```

- Create a sql file

```
CREATE DATABASE ads;

USE ads;

CREATE TABLE credentials (
  username varchar(30),
  password varchar(30),
  message varchar(100)
);

INSERT INTO credentials (
 username,
 password,
 message
) VALUES (
 'john',
 'password',
 'Johny, Johny, are you eating sugar?'
);

INSERT INTO credentials (
 username,
 password,
 message
) VALUES (
 'albert',
 'password',
 'How is your relativity research going on?'
);
```

- Build an image using the two files

  `sudo docker build -t adsmysql .`

  The image name is *adsmysql*

- The image is tagged and pushed to DockerHub as rutikaranjolkar/adsmysql

- `sudo docker login`

  `sudo docker tag adsmysql rutikaranjolkar/adsmysql:latest`

  `sudo docker push rutikaranjolkar/adsmysql:latest`

- Pull the image from DockerHub.

```
sudo docker pull rutikaranjolkar/adsmysql
```

```
rutika@rutika-VirtualBox:~$ sudo docker pull rutikaranjolkar/adsmysql
Using default tag: latest
latest: Pulling from rutikaranjolkar/adsmysql
10a267c67f42: Pull complete
c2dcc7bb2a88: Pull complete
17e7a0445698: Pull complete
9a61839a176f: Pull complete
a1033d2f1825: Pull complete
0d6792140dcc: Pull complete
cd3adf03d6e6: Pull complete
d79d216fd92b: Pull complete
b3c25bdeb4f4: Pull complete
02556e8f331f: Pull complete
4bed508a9e77: Pull complete
eae098b311a0: Pull complete
Digest: sha256:22d7a28b7ecd8bb81e1fecc14091908302aaeda030017304d11b19d98a12683b
Status: Downloaded newer image for rutikaranjolkar/adsmysql:latest
rutika@rutika-VirtualBox:~$
```

- Run the image that we just built.

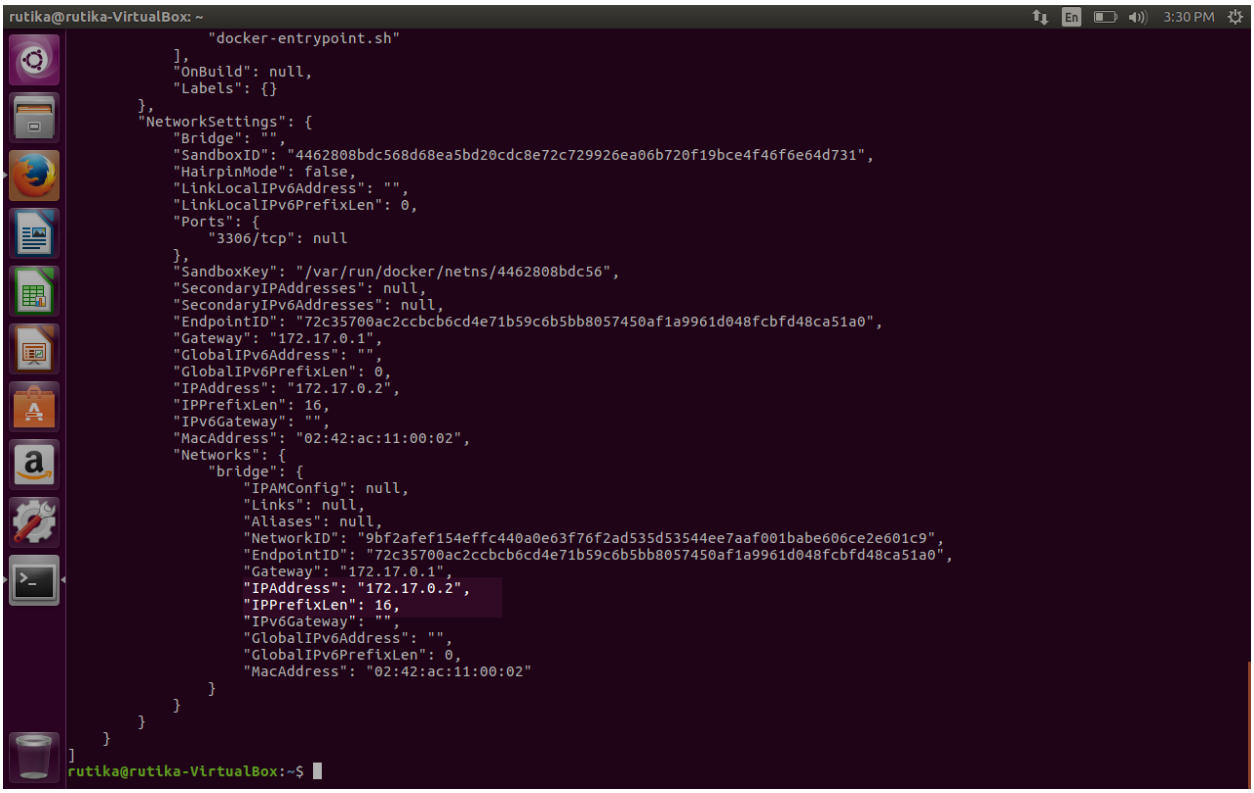
```
sudo docker run -d -name adsmysql -env "MYSQL_ROOT_PASSWORD=mypass" rutikaranjolkar/adsmysql:latest
```

```
rutika@rutika-VirtualBox:~$ sudo docker run -d --name adsmysql --env "MYSQL_ROOT_PASSWORD=mypass" rutikaranjolkar/adsmysql:latest
[sudo] password for rutika:
85b0ccfbcffd781a99e8eae6b939630b8604ae4a1789cf0b8adef9b5303d6c6e
rutika@rutika-VirtualBox:~$
```

Now we have a running container named adsmysql that has all the features listed in the Dockerfile. It also has a database named ads. Inside ads is a table called credentials that store username, password and message.

- Inspect the container to get the IP address of the microservice.
  `sudo docker ps`

```
rutika@rutika-VirtualBox:~$ sudo docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS
     NAMES
85b0ccfbcffd        rutikaranjolkar/adsmysql:latest   "docker-entrypoint..."   6 minutes ago       Up 6 minutes        3306/tcp
     adsmysql
rutika@rutika-VirtualBox:~$ sudo docker inspect adsmysql
[
    {
        "Id": "85b0ccfbcffd781a99e8eae6b939630b8604ae4a1789cf0b8adef9b5303d6c6e",
        "Created": "2017-06-01T19:17:49.4094539182",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "mysqld"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 3033,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2017-06-01T19:17:49.810603882Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:a9f47c1064cad77ac9524b65fb043d370f72fee4a1f6576e27eaed0f5cfd10b4",
        "ResolvConfPath": "/var/lib/docker/containers/85b0ccfbcffd781a99e8eae6b939630b8604ae4a1789cf0b8adef9b5303d6c6e/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/85b0ccfbcffd781a99e8eae6b939630b8604ae4a1789cf0b8adef9b5303d6c6e/hostname",
        "HostsPath": "/var/lib/docker/containers/85b0ccfbcffd781a99e8eae6b939630b8604ae4a1789cf0b8adef9b5303d6c6e/hosts",
```



IPAddress: “172.17.0.2”: for database microservice

Here, Next Step is to create another microservice in docker container for “authentication”

6.3.2 Create another microservice for authentication

- Create a Dockerfile.

```
FROM node:boron

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

- Create the following files:

Application code: server.js

```
var mysql      = require('mysql');
var express = require('express');

var mysqlIpAddress = process.env.MYSQL_IP_ADDRESS;
console.log("Connecting to MySQL on port 3306 at: ", mysqlIpAddress);

function getMessage(username, password) {
  var connection = mysql.createConnection({
    host     : mysqlIpAddress,
    user     : 'root',
    password : 'mypass',
    database : 'ads'
  });

  connection.connect();
  var theQuery = "SELECT * FROM credentials WHERE username = '" + username + "' AND password = '" + password +"'";
  console.log("The query is ", theQuery);

  var messageToBeReturned;
  connection.query(theQuery, function (error, results, fields) {
    if(error) throw error;
    console.log("Results = ", JSON.stringify(results));
    if(results.length > 0) {
      messageToBeReturned = results[0].message;
    }
  });
  connection.end();
  return messageToBeReturned;
};

var app = express();

app.get('/', function (req, res) {
  var username = req.query.username;
  var password = req.query.password;
  console.log("Username = ", username, "Password = ", password);

  var connection = mysql.createConnection({
    host     : '172.17.0.2',
    user     : 'root',
    password : 'mypass',
    database : 'ads'
  });
  connection.connect();
  var theQuery = "SELECT * FROM credentials WHERE username = '" + username + "' AND password = '" + password +"'";
  console.log("The query is ", theQuery);

  connection.query(theQuery, function (error, results, fields) {
    if(error) throw error;
    console.log("Results = ", JSON.stringify(results));
    if(results.length > 0) {
      var message = results[0].message;
      console.log("Got message : ", message);
      res.json({
        success: true,
        message: message
      });
    } else {
      res.json({
        success: false
      });
```

Application dependencies are written in package.json

- Create an image using the above files.

  ```
  sudo docker build adsauth .
  ```

- Tag and push the image to DockerHub.

  ```
  sudo docker tag adsauth rutikaranjolkar/adsauth:change1
  ```

  ```
  sudo docker push rutikaranjolkar/adsmysql:change1
  ```

- Pull the image from DockerHub:

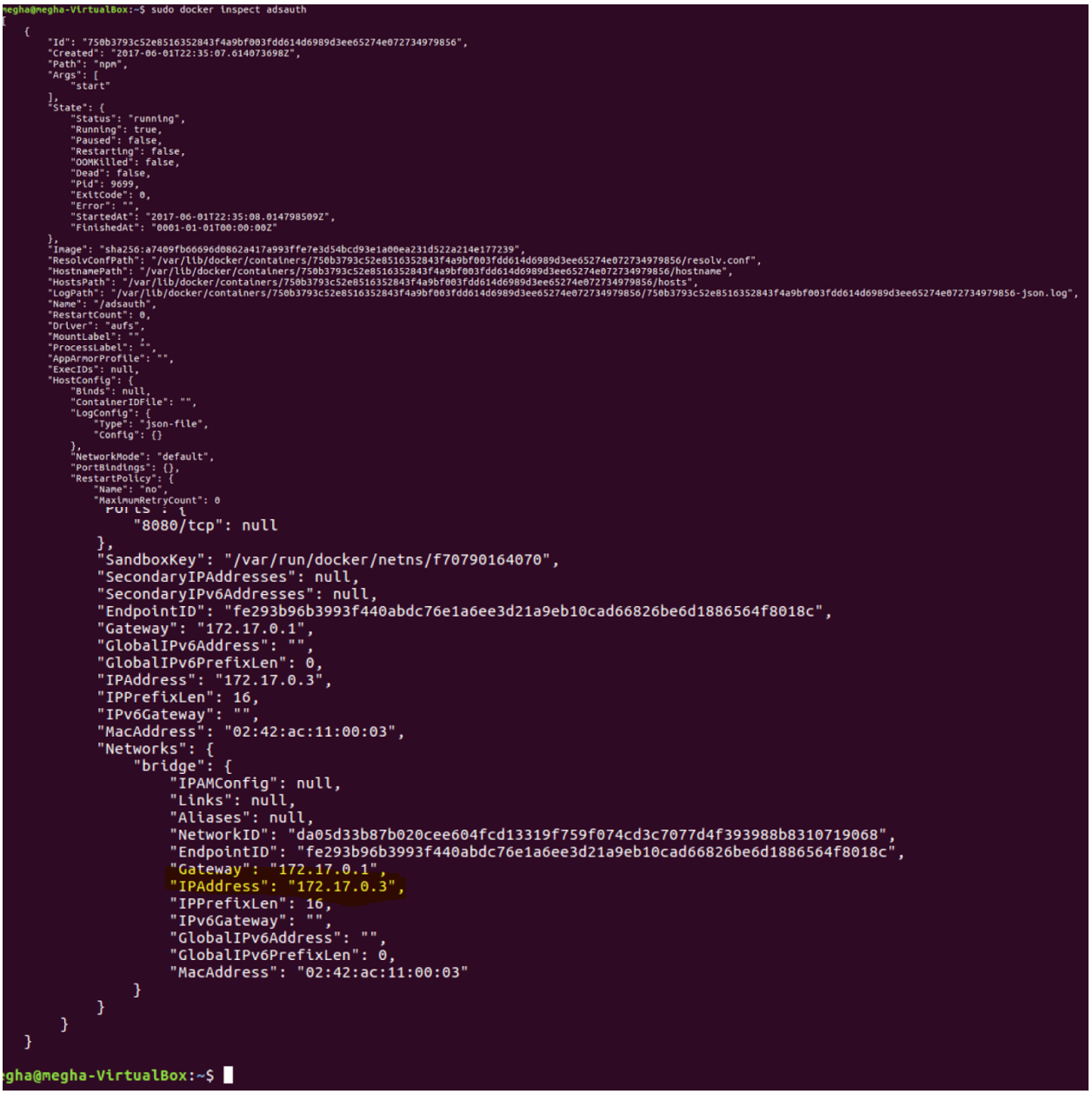
  ```
  sudo docker pull rutikaranjolkar/adsauth:change1
  ```

- We now run the docker image in order to create a container. Here, we pass the IP address of mySQL container to connect auth container to it.

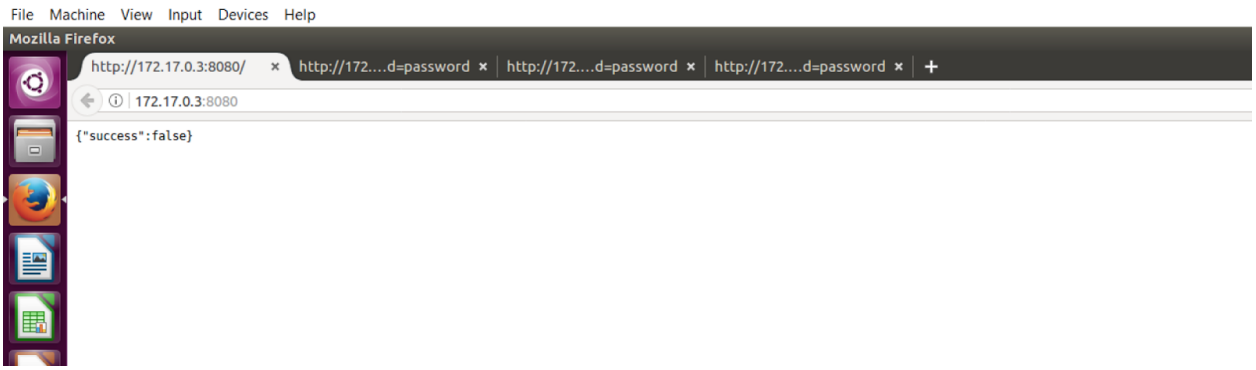
  ```
  sudo docker run -d --name adsauth --env "MYSQL_IP_ADDRESS=172.17.0.2" rutikaranjolkar/adsauth:change1
  ```

- We now inspect the IP address of the newly created adsauth container.

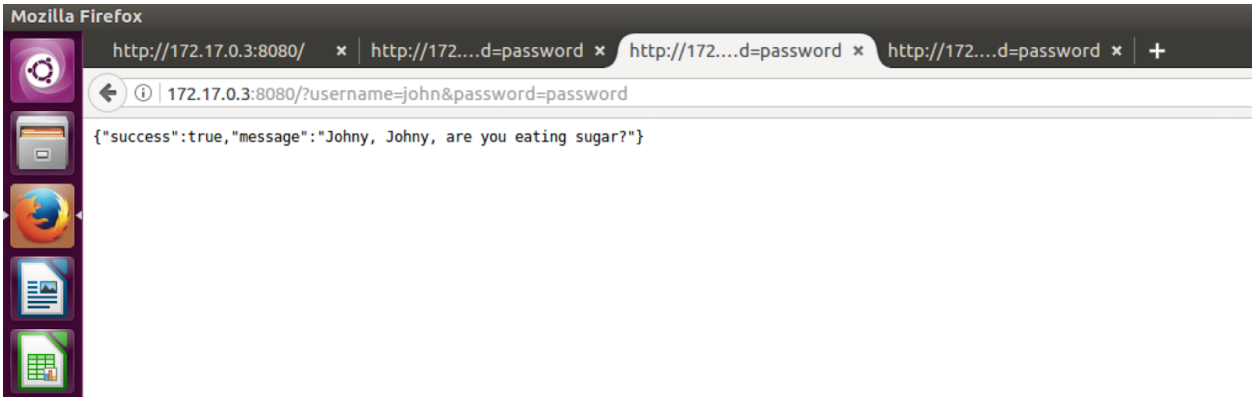
sudo docker inspect adsauth



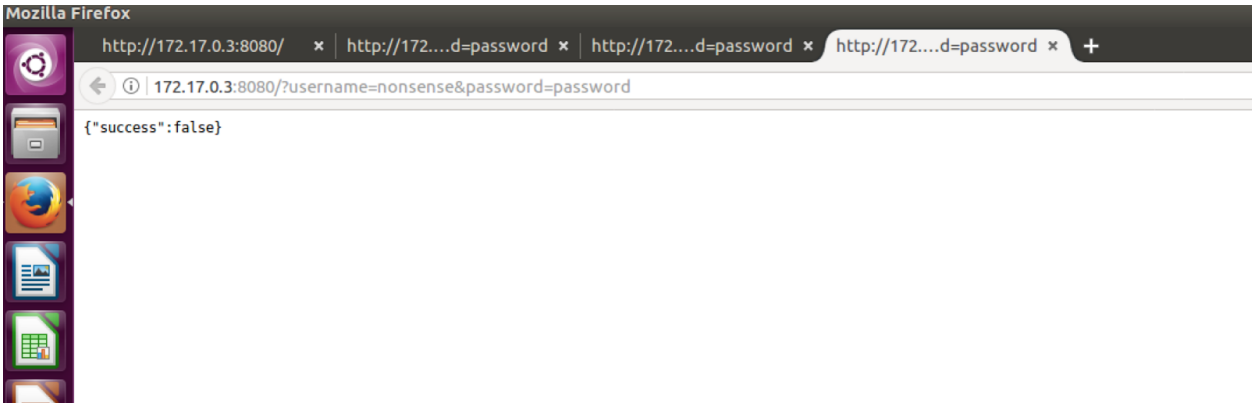
The IP address in our case is 172.17.0.3 and port is 8080.
On going visiting this address, we can see:



- And now we can pass our username and password pairs to get unique messages.



The above two users are present in the database. Hence, they receive their messages. But what happens when the credentials are not present?



6.4 Common hurdles while Dockerizing:

Configuration is typically done via environment variables. Some adjustments are needed if your app is configured via files. An advantage of environment variables is to make the app more general in purpose and can work on multiple platforms. It increases the reusability of the docker image.

Properly capturing the logging of the application. To preserve the logs, the docker container has to save the logs in docker volume.

Authoring a robust Dockerfile can be tricky. Enlisting dependencies and getting all the components you need to create the container can be challenging.

7. Connecting front end and backend using Kubernetes on Google Cloud

This task shows how to create a frontend and a backend microservice. The backend microservice is a hello greeter. The frontend and backend are connected using a Kubernetes Service object.

Pre-requisite: gcloud command line is used as a default command-line tool.
The following objectives are covered:

- Create and run a microservice using a Deployment object.
- Route traffic to the backend using a frontend.
- Use a Service object to connect the frontend application to the backend application.

Before working on gcloud the following should know:

- YAML Configuration file: It stands for "yet another markup language". It is used to create a Pod and then in deployment. YAML is a superset of JSON. Example of a simple Pod using YAML.

```
apiVersion: v1
kind: Pod
metadata:
 name: rss-site
 labels:
   app: web
spec:
 containers:
   - name: front-end
     image: nginx
     ports:
       - containerPort: 80
   - name: rss-reader
     image: nickchase/rss-php-nginx:v1
     ports:
       - containerPort: 88
```

7.1 How to connect the frontend and backend?

- Backend hello greeter microservice is deployed using the yaml configuration file which has all the app details and container details
- Kubernetes backend service object is created to which creates a persistent IP address and DNS name entry for backend microservice to reach and connect
- A service uses selector labels to find pods that it routes traffic to
- Create frontend using yaml configuration file which connects to backend worker pod using DNS name
- Pods in frontend deployment run ngnix image which finds backend service
- Services are load balanced which is mentioned in the yaml configuration files
- After all the services are deployed and external IP is generated, connect the frontend service IP from browser to see the message

7.2 Steps required to connect the frontend and backend using Kubernetes on gcloud

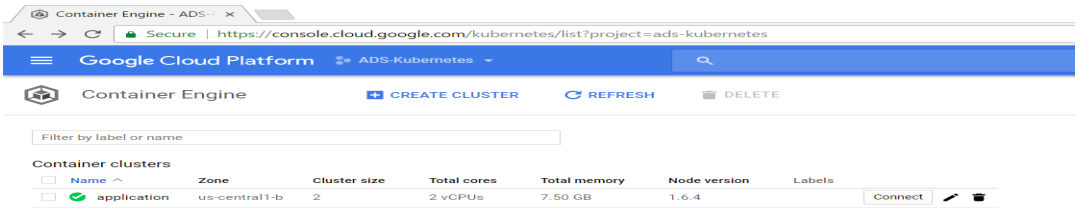
Step1: Set the default configuration values by running the following commands

```
gcloud config set project PROJECT_ID
gcloud config set compute/zone us-central1-b
```

```
maansichandira@ads-kubernetes:~$ gcloud config set project ads-kubernetes
Updated property [core/project].
maansichandira@ads-kubernetes:~$ gcloud config set compute/zone us-central1-b
Updated property [compute/zone].
maansichandira@ads-kubernetes:~$
```

Step 2: Create clusters as per node requirement of the application.

```
kubeconfig entry generated for application.
NAME         ZONE           MASTER_VERSION  MASTER_IP        MACHINE_TYPE   NODE_VERSION  NUM_NODES  STATUS
application  us-central1-b  1.6.4           104.198.139.189  n1-standard-1  1.6.4         2          RUNNING
```



Step 3: Create service with kubectl with service yaml files. First backend service deployment with hello.yaml file is created.

```
maansichandira@ads-kubernetes:~$
maansichandira@ads-kubernetes:~$ kubectl create -f https://k8s.io/docs/tasks/access-application-cluster/hello.yaml

deployment "hello" created
maansichandira@ads-kubernetes:~$
maansichandira@ads-kubernetes:~$ kubectl describe deployment hello
Name:                   hello
Namespace:              default
CreationTimestamp:      Thu, 01 Jun 2017 20:27:30 -0400
Labels:                 app=hello
                        tier=backend
                        track=stable
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=hello,tier=backend,track=stable
Replicas:               7 desired | 7 updated | 7 total | 7 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:       app=hello
                tier=backend
                track=stable
  Containers:
   hello:
    Image:              gcr.io/google-samples/hello-go-gke:1.0
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
  Progressing   True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  hello-1987912066 (7/7 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type    Reason                  Message
  ---------     --------        -----   ----                    -------------   ------- ------                  -------
  20s           20s             1       deployment-controller                   Normal  ScalingReplicaSet       Scaled up replica set hello-1987912066 to 7
maansichandira@ads-kubernetes:~$
```

Step 4: Create front-end service with hello-service.yaml file.

```
maansichandira@ads-kubernetes:~$ kubectl create -f https://k8s.io/docs/tasks/access-appli
cation-cluster/frontend.yaml
service "frontend" created
deployment "frontend" created
```

Check the port and details of the service:

```
maansichandira@ads-kubernetes:~$ kubectl get service frontend
NAME       CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
frontend   10.51.240.12   104.154.56.1    80:31784/TCP     2m
maansichandira@ads-kubernetes:~$
```

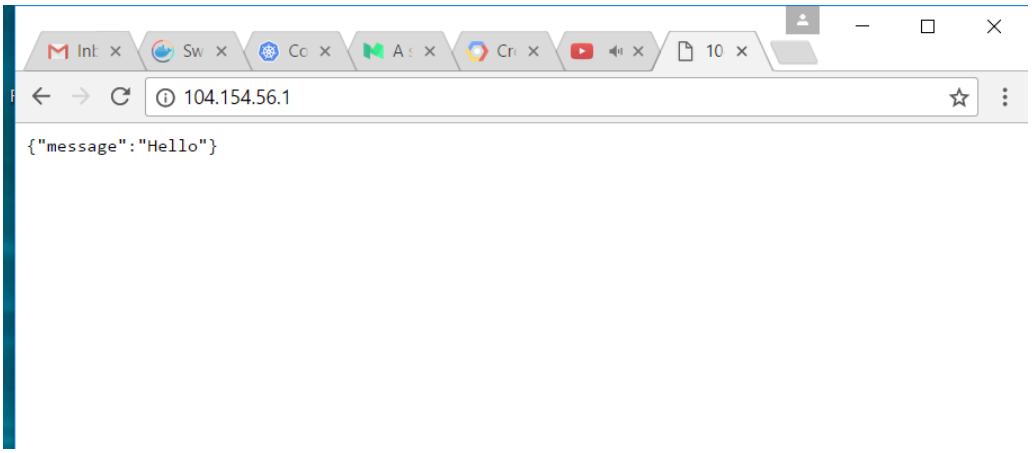
Step 5: Check the browser or use the curl command to confirm the connectivity between frontend and backend services to view the message.

Curl →

```
maansichandira@ads-kubernetes:~$ curl http://104.154.56.1
{"message":"Hello"}
```

OR

Browser →



The frontend and backends are now connected.

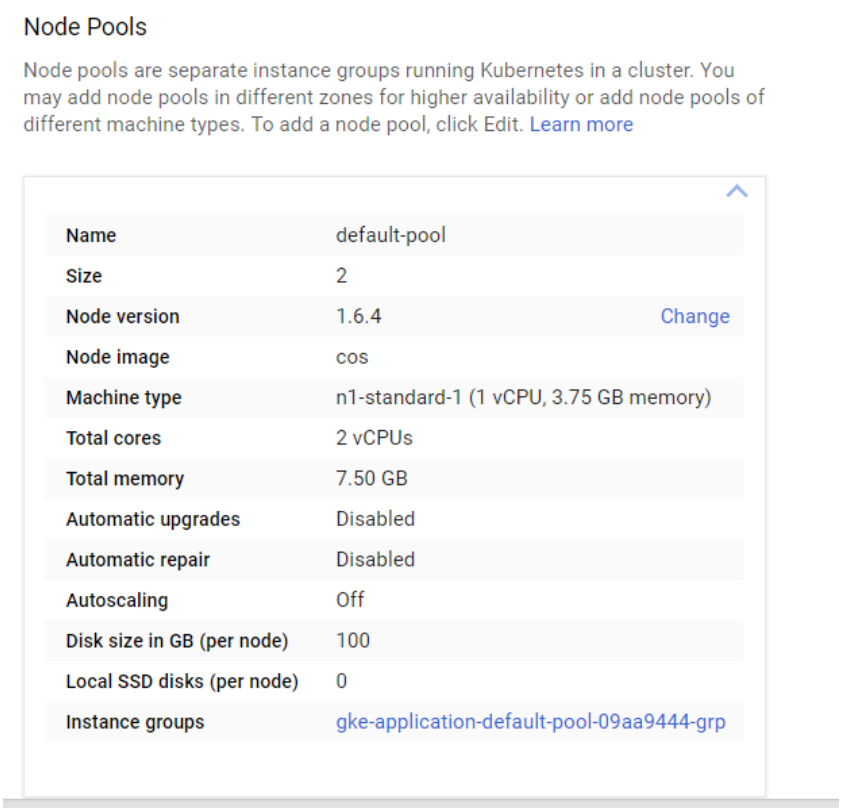
Get Pods:

```
maansichandira@ads-kubernetes:~$ kubectl get pods
NAME                        READY     STATUS    RESTARTS   AGE
frontend-3533783101-7j25p   1/1       Running   2          7m
hello-1987912066-0w0dd      1/1       Running   0          12m
hello-1987912066-139ft      1/1       Running   0          12m
hello-1987912066-7njl7      1/1       Running   0          12m
hello-1987912066-cs00r      1/1       Running   0          12m
hello-1987912066-m6zs0      1/1       Running   0          12m
hello-1987912066-q88hr      1/1       Running   0          12m
hello-1987912066-v29p7      1/1       Running   0          12m
maansichandira@ads-kubernetes:~$
```

Get services:

```
maansichandira@ads-kubernetes:~$ kubectl get service
NAME         CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
frontend     10.51.240.12    104.154.56.1    80:31784/TCP     8m
hello        10.51.251.32    <none>          80/TCP           7m
kubernetes   10.51.240.1     <none>          443/TCP          14m
maansichandira@ads-kubernetes:~$
```

Node pools:



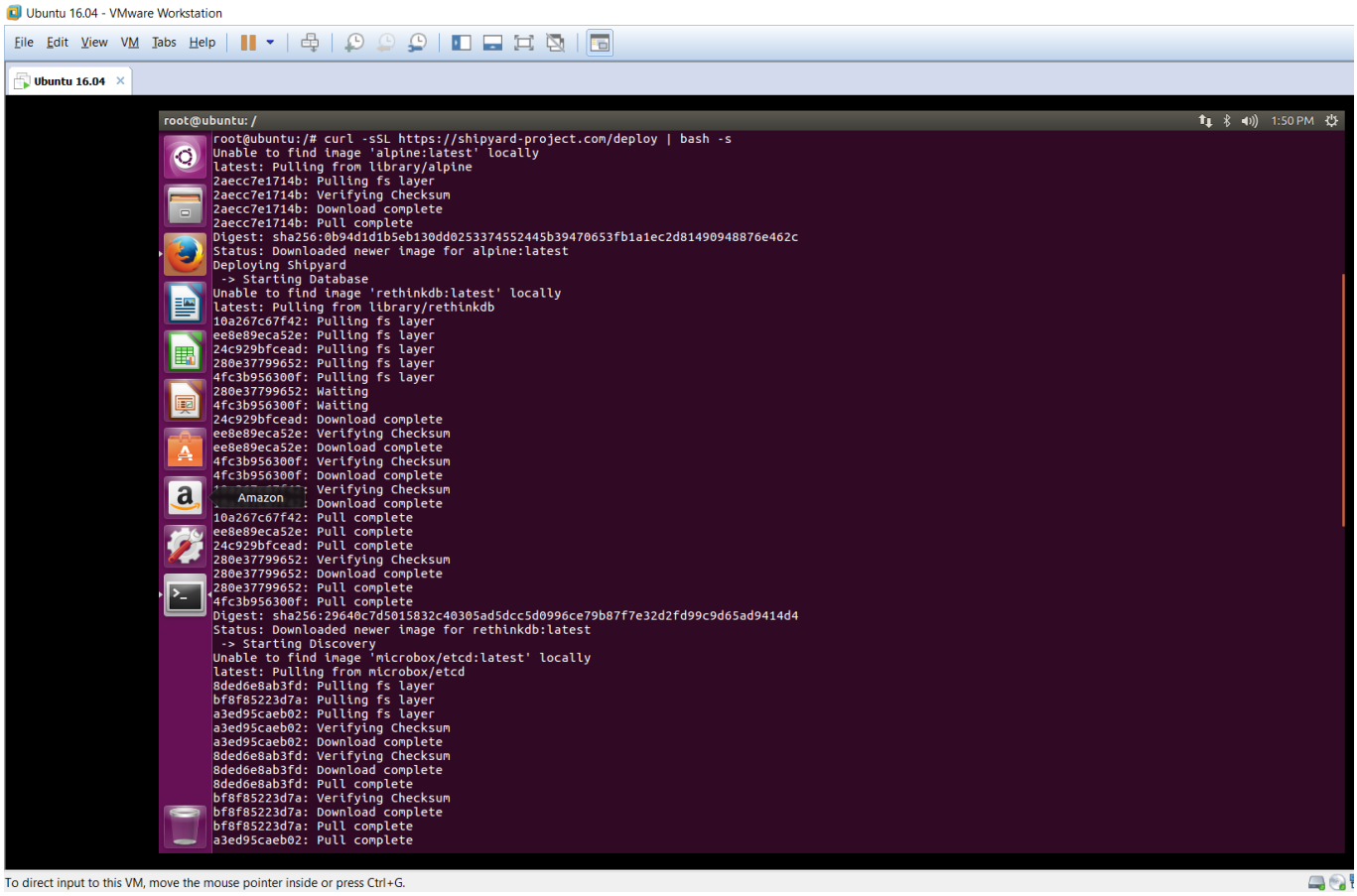
8. SHIPYARD (Composable Docker Management)

Shipyard: Built on Docker Swarm, Shipyard gives you the ability to manage Docker resources including containers, images, private registries and more.

Shipyard differs from other management applications in that it promotes composability and is 100% compatible with the Docker Remote API. Shipyard manages containers, images, nodes, private registries cluster-wide as well as providing authentication and role based access control.

- Install Shipyard:

```
curl sSl http://shipyard.project.com/deploy | bash -s
```



- Now login using username and password:

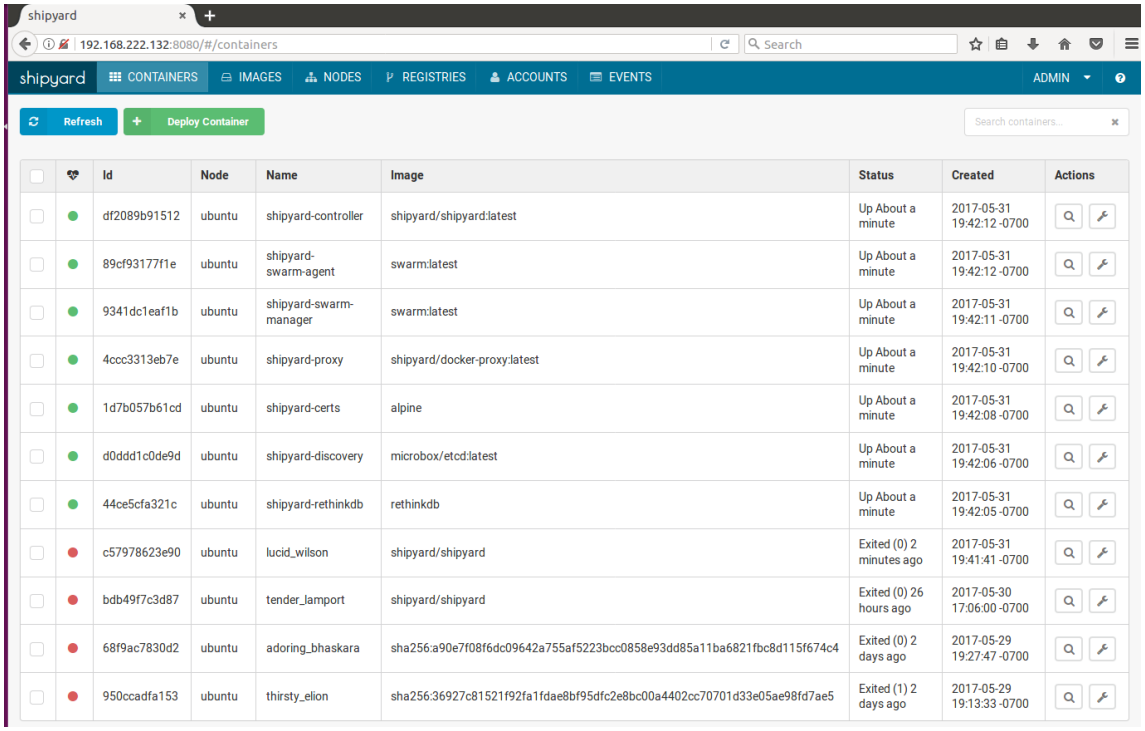
```
Digest: sha256:5f065362680fa4565dd150c8da3edd09b79a7a3010d3ceef20093c2a879187e0
Status: Downloaded newer image for shipyard/shipyard:latest
Waiting for Shipyard on 192.168.222.132:8080

Shipyard available at http://192.168.222.132:8080
Username: admin Password: shipyard
root@ubuntu:/#
```

- Shipyard starts the following things:

```
root@ubuntu:/# curl -sSL https://shipyard-project.com/deploy | bash -s
Deploying Shipyard
 -> Starting Database
 -> Starting Discovery
 -> Starting Cert Volume
 -> Starting Proxy
 -> Starting Swarm Manager
 -> Starting Swarm Agent
 -> Starting Controller
Waiting for Shipyard on 192.168.222.132:8080

Shipyard available at http://192.168.222.132:8080
Username: admin Password: shipyard
```

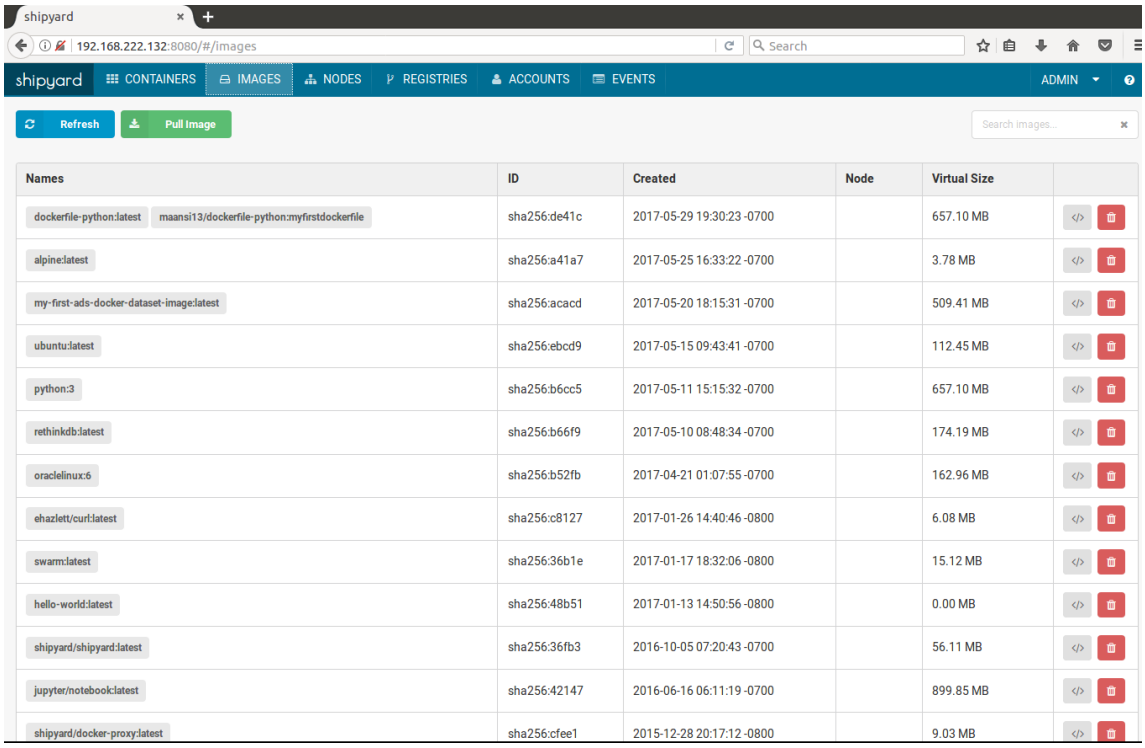
- The user interface provides simple management over your Docker cluster. You can manage containers, cluster images, private registries, authentication.

- Shipyard UI looks like this:



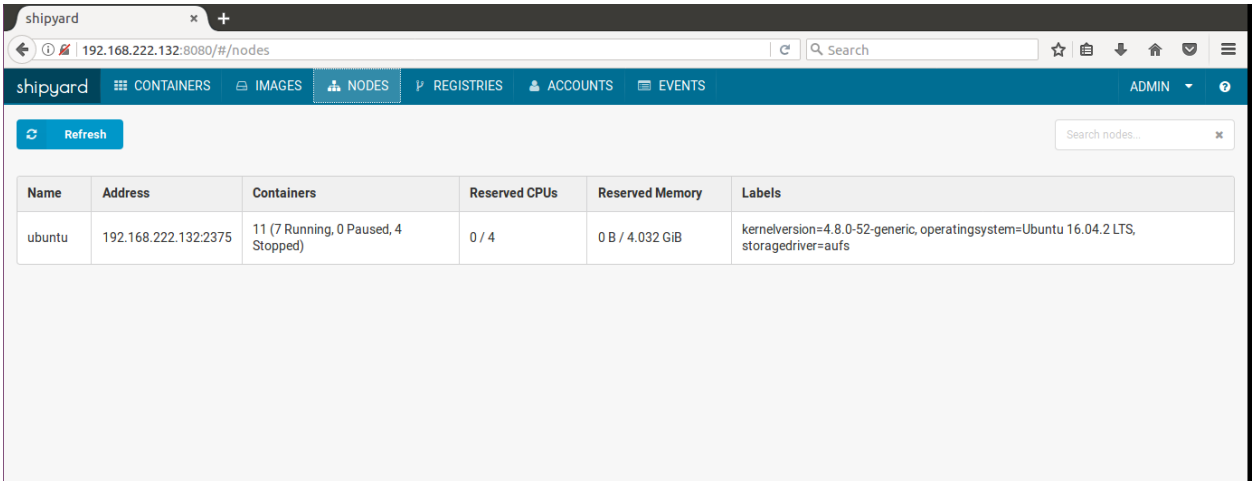
	Id	Node	Name	Image	Status	Created	Actions
	df2089b91512	ubuntu	shipyard-controller	shipyard/shipyard:latest	Up About a minute	2017-05-31 19:42:12 -0700	
	89cf93177f1e	ubuntu	shipyard-swarm-agent	swarm:latest	Up About a minute	2017-05-31 19:42:12 -0700	
	9341dc1eaf1b	ubuntu	shipyard-swarm-manager	swarm:latest	Up About a minute	2017-05-31 19:42:11 -0700	
	4ccc3313eb7e	ubuntu	shipyard-proxy	shipyard/docker-proxy:latest	Up About a minute	2017-05-31 19:42:10 -0700	
	1d7b057b61cd	ubuntu	shipyard-certs	alpine	Up About a minute	2017-05-31 19:42:08 -0700	
	d0ddd1c0de9d	ubuntu	shipyard-discovery	microbox/etcd:latest	Up About a minute	2017-05-31 19:42:06 -0700	
	44ce5cfa321c	ubuntu	shipyard-rethinkdb	rethinkdb	Up About a minute	2017-05-31 19:42:05 -0700	
	c57978623e90	ubuntu	lucid_wilson	shipyard/shipyard	Exited (0) 2 minutes ago	2017-05-31 19:41:41 -0700	
	bdb49f7c3d87	ubuntu	tender_lamport	shipyard/shipyard	Exited (0) 26 hours ago	2017-05-30 17:06:00 -0700	
	68f9ac7830d2	ubuntu	adoring_bhaskara	sha256:a90e7f08f6dc09642a755af5223bcc0858e93dd85a11ba6821fbc8d115f674c4	Exited (0) 2 days ago	2017-05-29 19:27:47 -0700	
	950ccadfa153	ubuntu	thirsty_elion	sha256:36927c81521f92fa1fdae8bf95dfc2e8bc00a4402cc70701d33e05ae98fd7ae5	Exited (1) 2 days ago	2017-05-29 19:13:33 -0700	

- Existing images can be viewed under images tab like this:



- More nodes can be added to the swarm under nodes tab with this command:

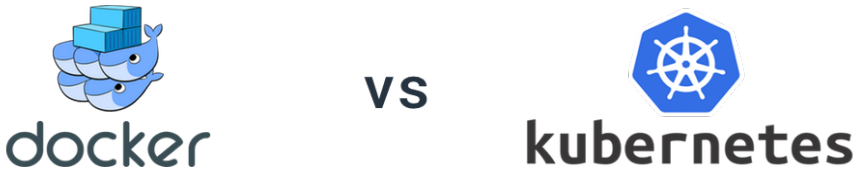
```
curl -sSL https://shipyard-project.com/deploy | ACTION=node DISCOVERY=etcd://10.0.1.10:4001 bash -s
```



- The nodes can be added using this command and can access a service shared as the slave nodes.

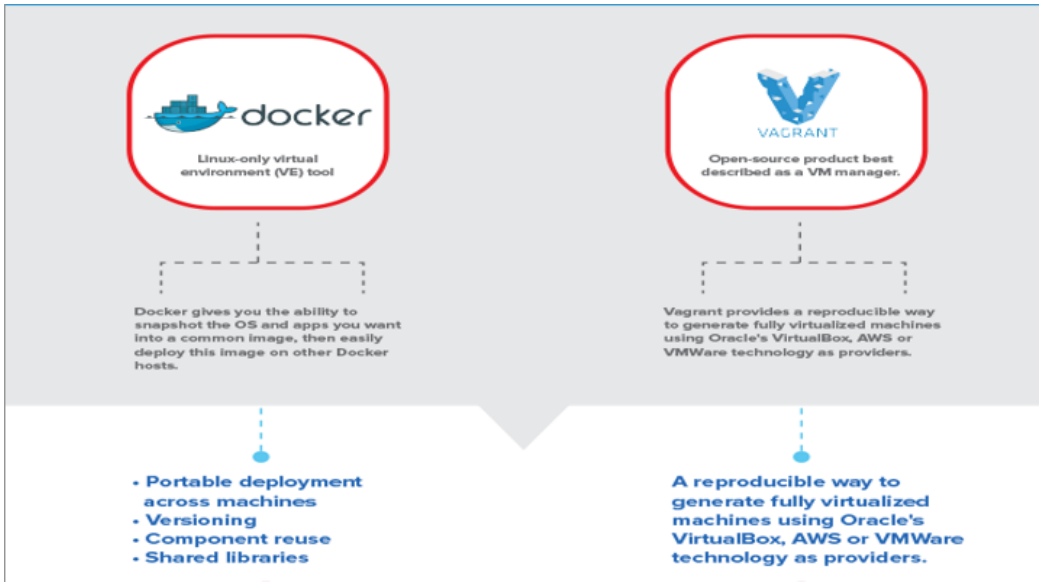
9. COMPARING CONTAINERS

9.1 Docker vs Kubernetes



Easy and fast to install and configure	Takes some work to get up and running.
Functionality is provided and limited by the Docker API	Client, API and YAML definitions are unique to Kubernetes
Quick container deployment and scaling even in very large clusters	Provides strong guarantees to cluster states at the expense of speed
Automated internal load balancing through any node in the cluster	Enabling load balancing requires manual service configuration
Process scheduling to maintain services while updating	Progressive updates and service health monitoring through the update
Simple shared local volumes	Volumes shared within pods
Automatically configured TLS authentication and container networking	TLS authentication requires manual configuration for security
Services are discoverable throughout the cluster network	Containers can be defined as services that are easily discoverable

9.2 Differences Between Docker and Vagrant



Virtualization Type is VE	Virtualization Type is VM
Docker is an open platform for building, shipping, and running distributed applications, relies on shared kernel linux containers	**Vagrant** creates and configures lightweight, reproducible, and portable development environments.

10. CITATIONS:

1) https://github.com/docker/docker
2) https://en.wikipedia.org/wiki/Operating-system-level_virtualization
3) https://docs.docker.com/
4) http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html
5) https://www.continuum.io/blog/developer-blog/anaconda-and-docker-better-together-reproducible-data-science
6) https://www.dataquest.io/blog/docker-data-science/
7) https://docs.docker.com/docker-hub/
8) https://cloud.google.com/container-engine/docs/tutorials/guestbook
9) https://kubernetes.io/docs/tasks/access-application-cluster/connecting-frontend-backend/
10) https://coreos.com/kubernetes/docs/latest/pods.html
11) https://shipyard-project.com/
12) https://www.upcloud.com/blog/docker-swarm-vs-kubernetes/
13) https://docs.docker.com/engine/swarm/
14) https://kubernetes.io/docs/getting-started-guides/gce/
15) http://jupyterhub.readthedocs.io/en/latest/quickstart.html
16) https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes
17) https://mysql.com
18) https://node.js.com
19) https://www.vagrantup.com/docs/cli/box.html
20) https:// www.virtualbox.org