**Docker-Notes**

Docker is a platform for developers and sysadmins to **develop, deploy, and run** applications with containers. The use of Linux containers to deploy applications is called *containerization*. Containers are not new, but their use for easily deploying applications is.

Containerization is increasingly popular because containers are:

* Flexible: Even the most complex applications can be containerized.
* Lightweight: Containers leverage and share the host kernel.
* Interchangeable: You can deploy updates and upgrades on-the-fly.
* Portable: You can build locally, deploy to the cloud, and run anywhere.
* Scalable: You can increase and automatically distribute container replicas.
* Stackable: You can stack services vertically and on-the-fly.

**Containers and Images**

A container is launched by running an image. An **image** is an executable package that includes everything needed to run an application–the code, a runtime, libraries, environment variables, and configuration files.

A **container** is a runtime instance of an image–what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, docker ps, just as you would in Linux.

**Containers and Virtual machines**

A **container** runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

By contrast, a **virtual machine** (VM) runs a full-blown “guest” operating system with virtual access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.

**Docker commands:**

1. docker
2. docker container –help
3. docker –version
4. docker version
5. docker info
6. docker run hello-world
7. docker images ls
8. docker container ls
9. docker container ls –all
10. docker container ls - a-q
11. **Containers**
12. **Service**
13. **Swarm**
14. **Stacks**
15. **Deploy app PROD**

**Containers:**

Create Dockerfile for your application and this docker file is used to build image . Below the sample Dockerfile for Python app.

# Use an official Python runtime as a parent image

FROM python:2.7-slim

# Set the working directory to /app

WORKDIR /app

# Copy the current directory contents into the container at /app

ADD . /app

# Install any needed packages specified in requirements.txt

RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container

EXPOSE 80

# Define environment variable

ENV NAME World

# Run app.py when the container launches

CMD ["python", "app.py"]

Create requirements.txt and app.py with below contents

**requirements.txt**

flask

redis

**app.py**

from flask import Flask

from redis import Redis, RedisError

import os

import socket

# Connect to Redis

redis = Redis(host="redis", db=0, socket\_connect\_timeout=2, socket\_timeout=2)

app = Flask(\_\_name\_\_)

@app.route("/")

def hello():

try:

visits = redis.incr("counter")

except RedisError:

visits = "<i>cannot connect to Redis, counter disabled</i>"

html = "<h3>Hello {name}!</h3>" \

"<b>Hostname:</b> {hostname}<br/>" \

"<b>Visits:</b> {visits}"

return html.format(name=os.getenv("NAME", "world"), hostname=socket.gethostname(), visits=visits)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=80)

Build the APP:

docker build –t firstapp .

docker images ls

docker run –it –p 4000:80 firstapp

Run the container in background

docker run –d –p 4000:80 firstapp

docker container ls

docker container stop <image>

**Share the docker image:**

docker login

docker tag image username/repository:tag

example:

docker tag firstapp username/repository:100

Push the image to remote repository

docker push username/repository:100

Run from anywhere now

docker run –p 4000:80 username/repository:100

**Run Docker container as Service:**