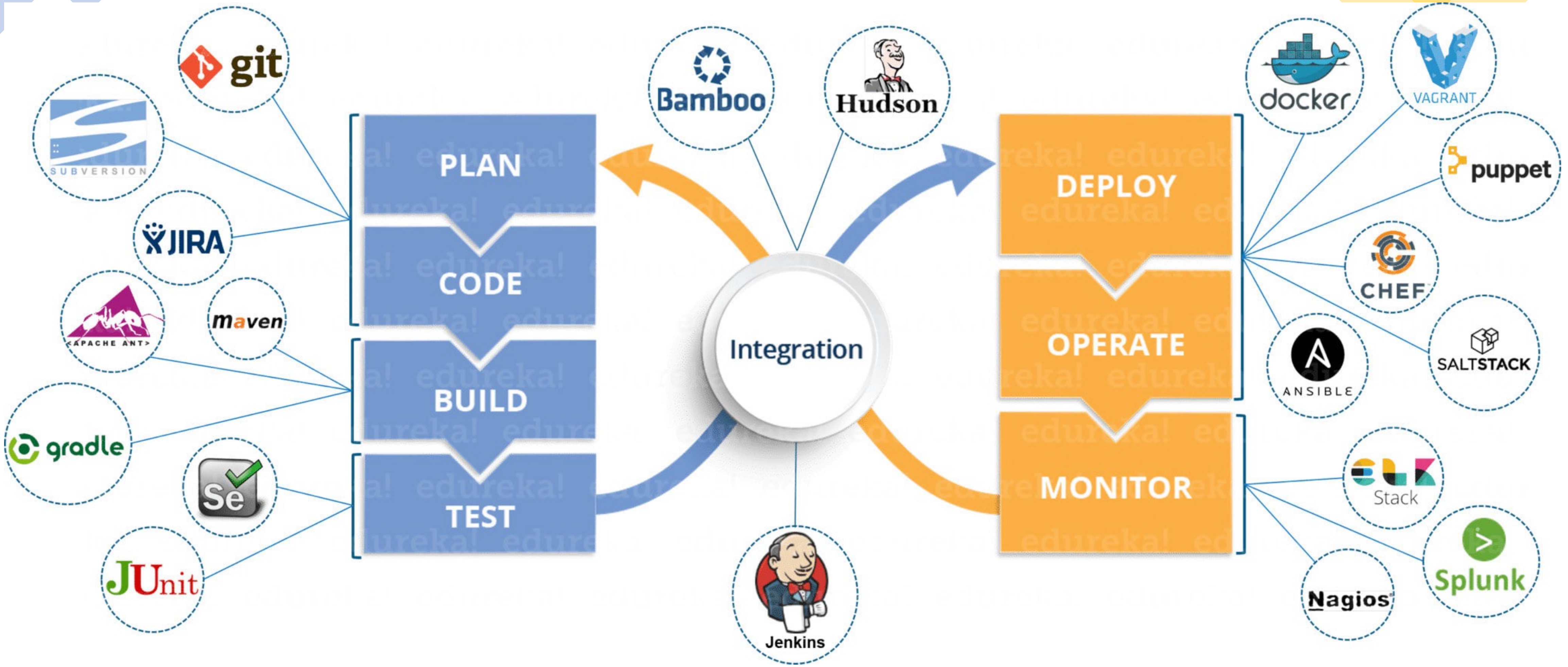
By Praveen Singampalli

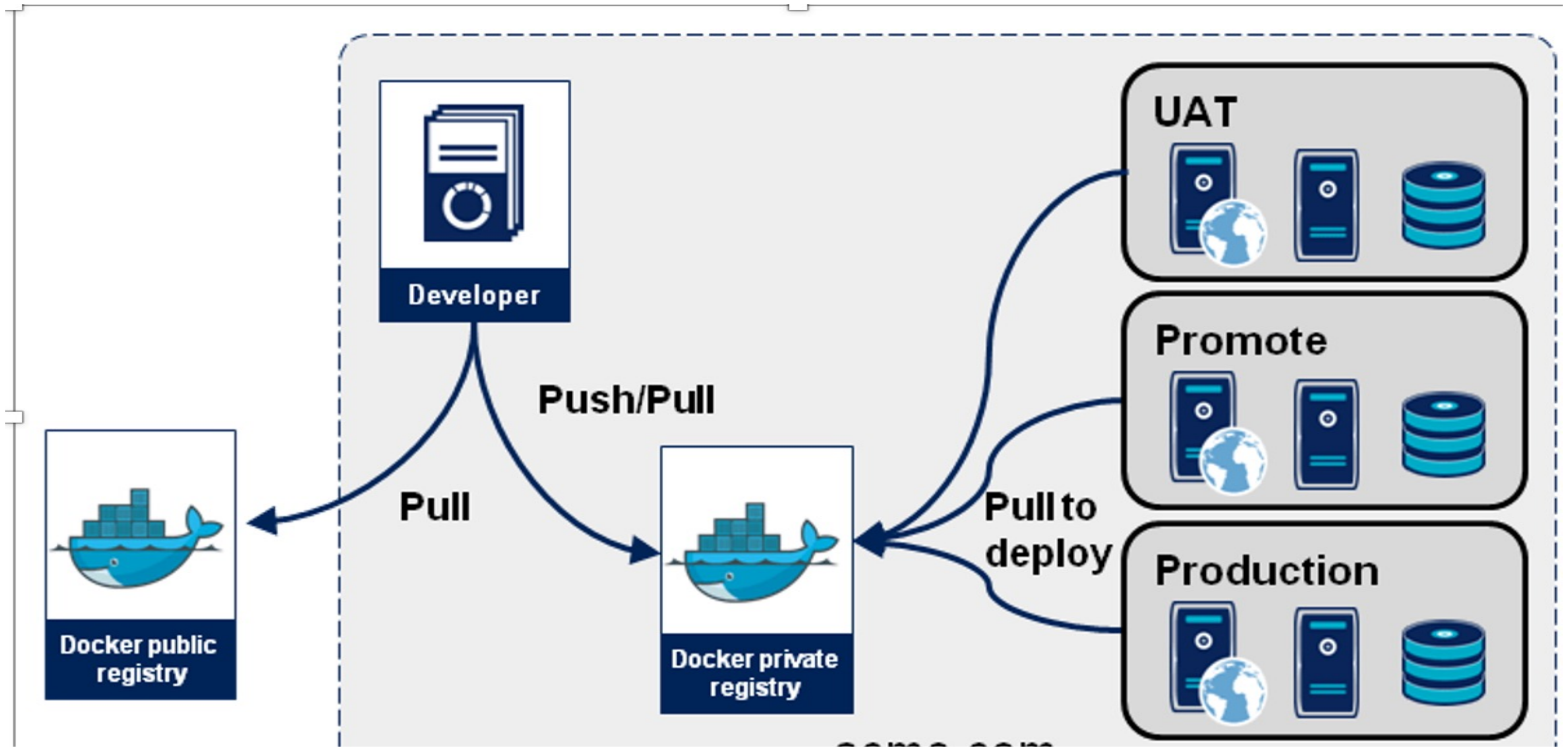
#DOCKER CD SESSION3

Continuous Delivery Tools



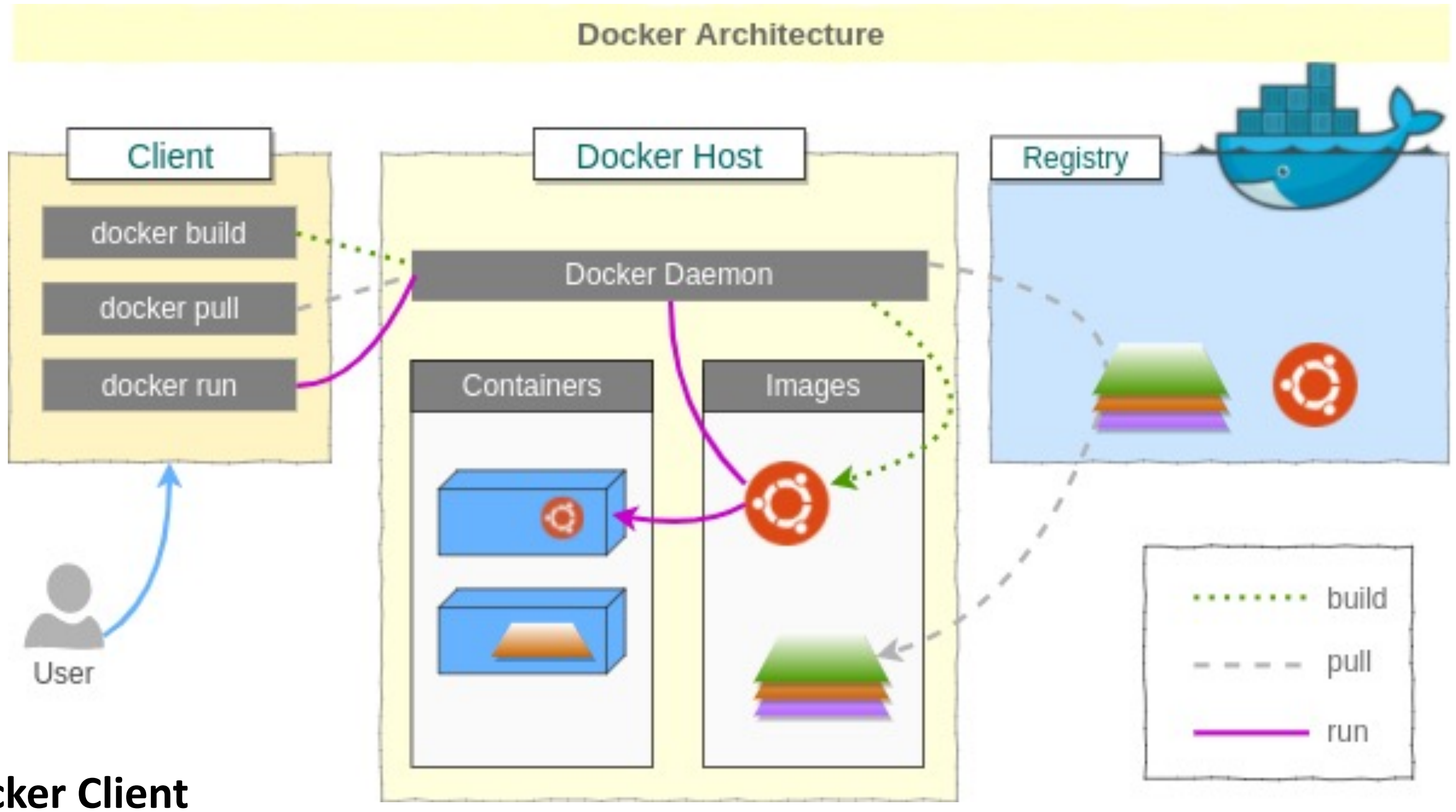
What is Docker?

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables to separate the applications from infrastructure so that the delivery of software is quick.
- Docker provides the ability to package and run an application in a loosely isolated environment called a container.
- The isolation and security allow you to run many containers simultaneously on a given host.



Docker Architecture

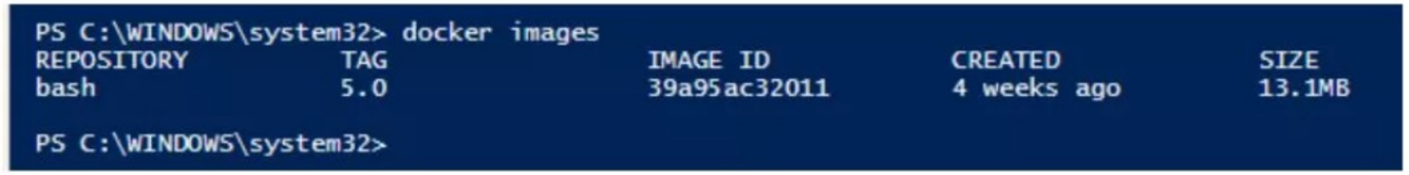
- Docker uses a client-server architecture.
- The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



1) **Docker Client**
2) **Docker Daemon**
3) **Docker Registry**
4) **Docker Image**
5) **Docker Container**

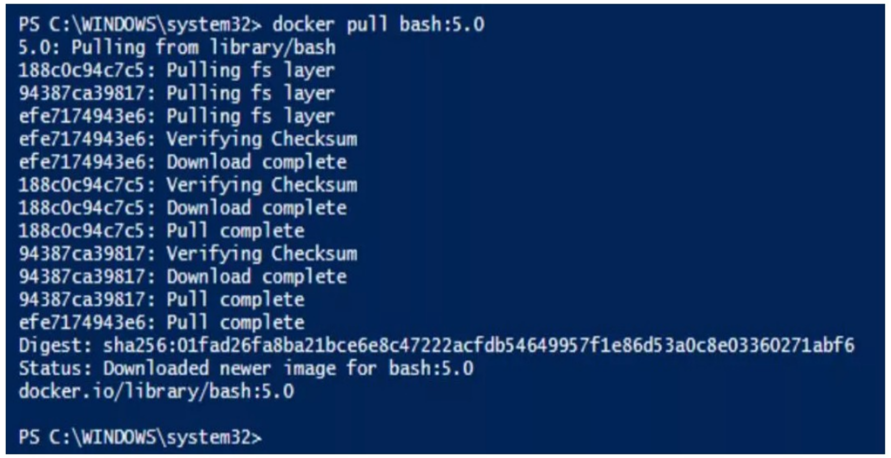
Docker Image/Container and DockerFile Commands

- A Docker image is a read-only, inert template that comes with instructions for deploying containers. In Docker, everything basically revolves around images.
- An image consists of a collection of files (or layers) that pack together all the necessities—such as dependencies, source code, and libraries—needed to set up a completely functional container environment.
- Images are stored on a Docker registry, such as the Docker Hub, or on a local registry.

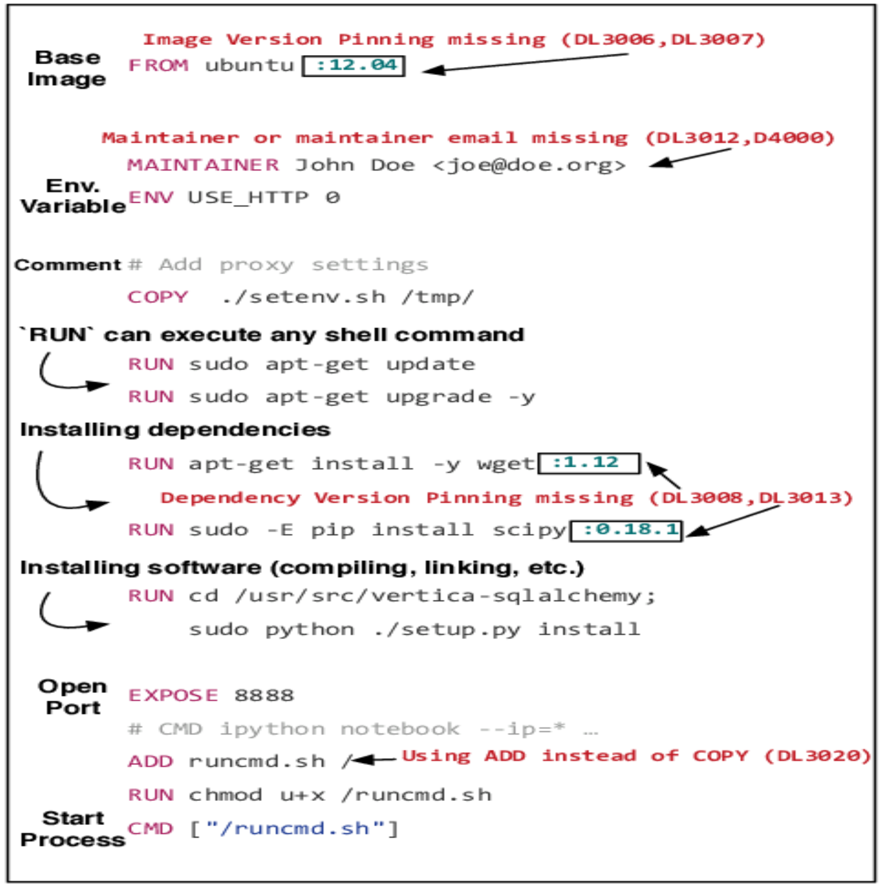


```
PS C:\WINDOWS\system32> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bash                5.0                 39a95ac32011        4 weeks ago         13.1MB

PS C:\WINDOWS\system32>
```

A Docker container is a virtualized runtime environment that provides isolation capabilities for separating the execution of applications from the underpinning system. It's an instance of a Docker image.



```
PS C:\WINDOWS\system32> docker pull bash:5.0
5.0: Pulling from library/bash
188c0c94c7c5: Pulling fs layer
94387ca39817: Pulling fs layer
efe7174943e6: Pulling fs layer
efe7174943e6: Verifying Checksum
efe7174943e6: Download complete
188c0c94c7c5: Verifying Checksum
188c0c94c7c5: Download complete
188c0c94c7c5: Pull complete
94387ca39817: Verifying Checksum
94387ca39817: Download complete
94387ca39817: Pull complete
efe7174943e6: Pull complete
Digest: sha256:01fad26fa8ba21bce6e8c47222acfdb54649957f1e86d53a0c8e03360271abf6
Status: Downloaded newer image for bash:5.0
docker.io/library/bash:5.0

PS C:\WINDOWS\system32>
```


Base Image
Image Version Pinning missing (DL3006,DL3007)
Maintainer or maintainer email missing (DL3012,D4000)
Env. Variable
Comment
`RUN` can execute any shell command
Installing dependencies
Dependency Version Pinning missing (DL3008,DL3013)
Installing software (compiling, linking, etc.)
Open Port
Using ADD instead of COPY (DL3020)
Start Process

```
FROM ubuntu:12.04

MAINTAINER John Doe <joe@doe.org>
ENV USE_HTTP 0

# Add proxy settings
COPY  ./setenv.sh /tmp/

RUN sudo apt-get update
RUN sudo apt-get upgrade -y

RUN apt-get install -y wget:1.12

RUN sudo -E pip install scipy:0.18.1

RUN cd /usr/src/vertica-sqlalchemy;
    sudo python ./setup.py install

EXPOSE 8888
# CMD ipython notebook --ip=* …
ADD runcmd.sh /
RUN chmod u+x /runcmd.sh
CMD ["/runcmd.sh"]
```

- RUN is an image build step
- CMD is the command the container executes by
default when you launch the built image

- CMD: Sets default parameters that can be overridden from the
Docker Command Line Interface (CLI) while running a docker container.
- ENTRYPOINT: Default parameters that cannot be overridden
while executing Docker Containers with CLI parameters.’

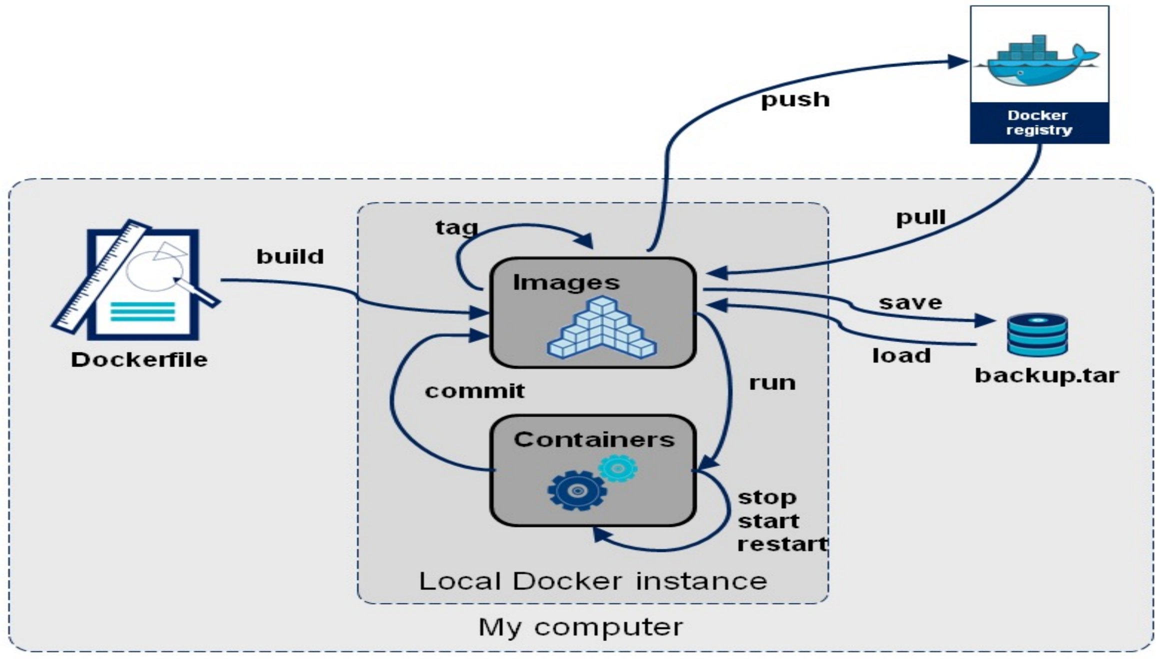
- COPY is a docker file command that copies files from a
local source location to a destination in the Docker container.
- ADD command is used to copy files/directories into a Docker image.

Docker Main Commands

- **docker pull** ubuntu. (To pull image from hub/repo)
- **docker run** –it –name c1 –d –p 82:80 ubuntu (To run an image as container)
- docker exec –it c1 bash (To login into container)

  Backup of container as Image

- **docker commit** c1 apache-on-ubuntu:1.0(To save the container data as new Image)
- **docker save** apache-on-ubuntu:1.0 --output backup.tar (To save the image as tar)
- **docker load** -i backup.tar (To unzip the image from tar)
- **docker start/stop/restart** c1 (Conatiner commands)
- **docker push** image-name (To push the image to conatiner)
- **docker build** Dockerfile



Docker Container States

- **Created -** Docker assigns the *created* state to the containers that were never started ever since they were created. Hence, no CPU or memory is used by the containers in this state.
- **Running -** When we start a container having created a state using the docker start command, it attains the running state. This state signifies that the processes are running in the isolated environment inside the container.
- **Restarting -** Docker supports four types of restart policies, namely – no, on-failure, always, unless-stopped. Restart policy decides the behaviour of the container when it exit. By default, the restart policy is set to no, which means that the container will not be started automatically after it exits.
- **Exited -** This state is achieved when the process inside the container terminates. **No CPU and memory are consumed** by the container.

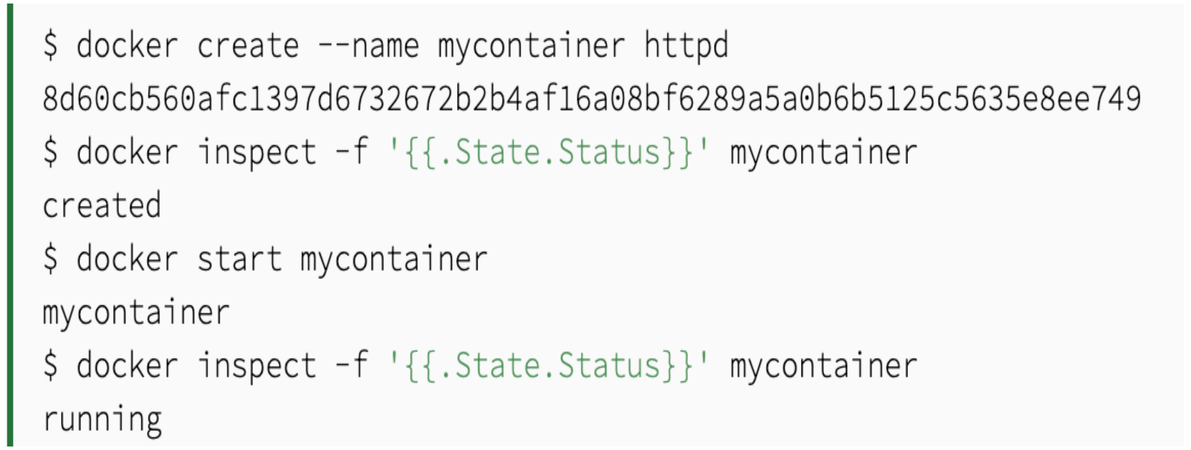
The process inside the container was completed, and so it exited.

The process inside the container encountered an exception while running.

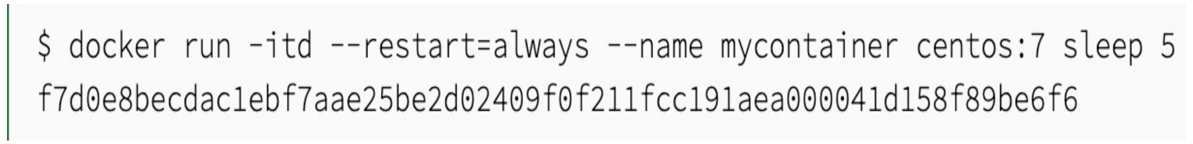
A container is intentionally stopped using the docker stop command.

No interactive terminal was set to a container running bash.

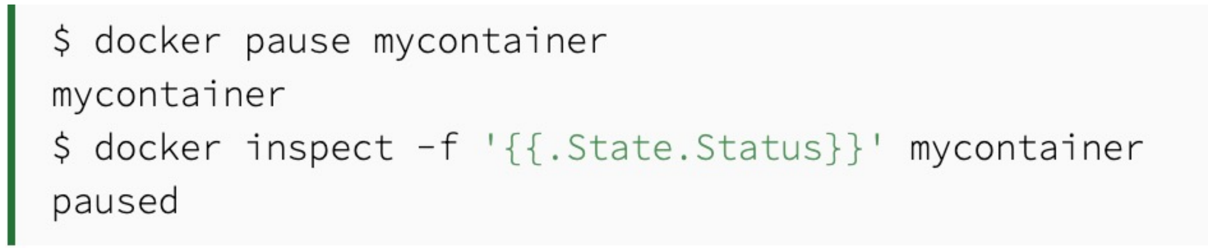
- **Pause** - A paused container consumes the same memory used while running the container, but the CPU is released completely



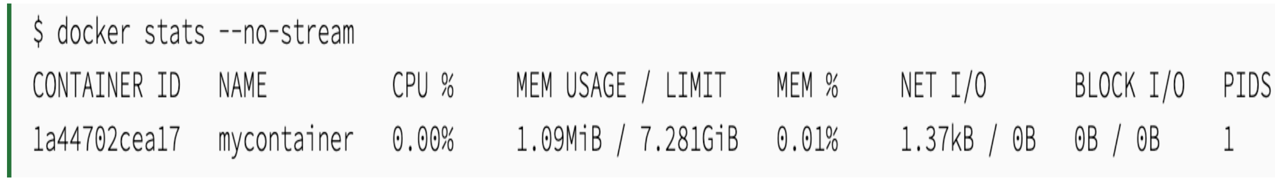
```
$ docker create --name mycontainer httpd
8d60cb560afc1397d6732672b2b4af16a08bf6289a5a0b6b5125c5635e8ee749
$ docker inspect -f '{{.State.Status}}' mycontainer
created
$ docker start mycontainer
mycontainer
$ docker inspect -f '{{.State.Status}}' mycontainer
running
```



```
$ docker run -itd --restart=always --name mycontainer centos:7 sleep 5
f7d0e8becdac1ebf7aae25be2d02409f0f211fcc191aea000041d158f89be6f6
```



```
$ docker pause mycontainer
mycontainer
$ docker inspect -f '{{.State.Status}}' mycontainer
paused
```



```
$ docker stats --no-stream
CONTAINER ID   NAME          CPU %    MEM USAGE / LIMIT     MEM %    NET I/O        BLOCK I/O   PIDS
1a44702cea17   mycontainer   0.00%    1.09MiB / 7.281GiB    0.01%    1.37kB / 0B    0B / 0B     1
```

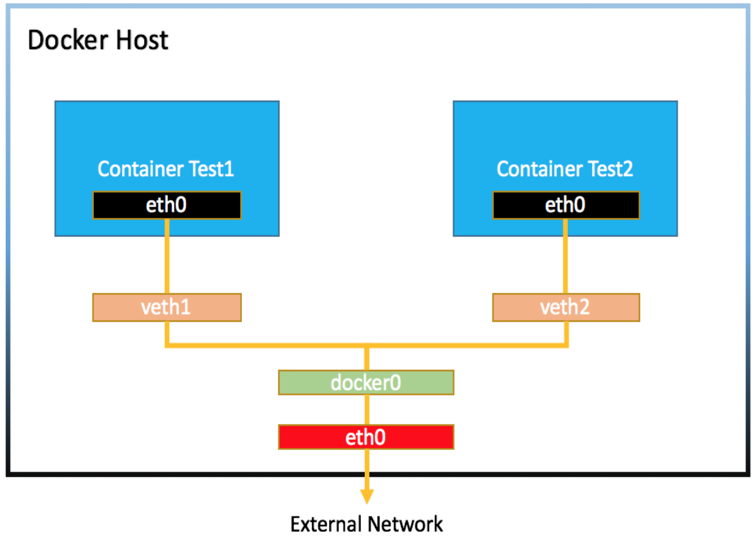
Docker Networking

1) Bridge: The default network driver. Bridge networks apply to containers running on the same Docker daemon host

2) Host: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly

3) Overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons.

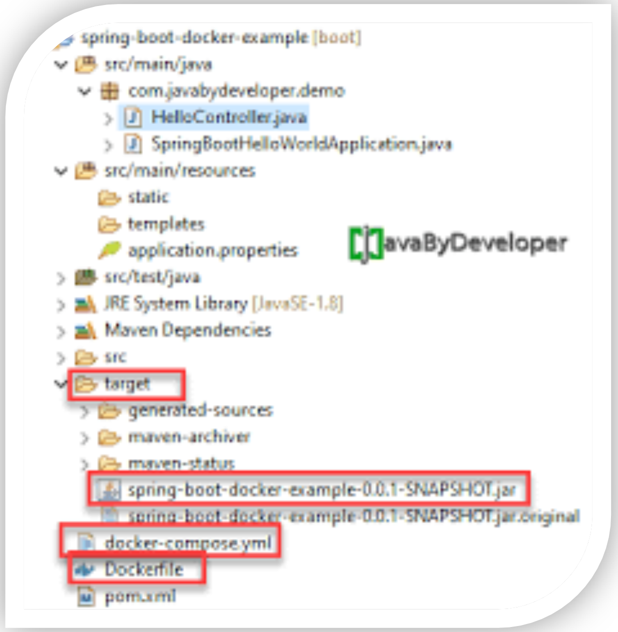
- **User-defined bridges** provide better isolation
- Containers can be attached and detached from user-defined networks on the fly.
- Each user-defined network creates a configurable bridge.



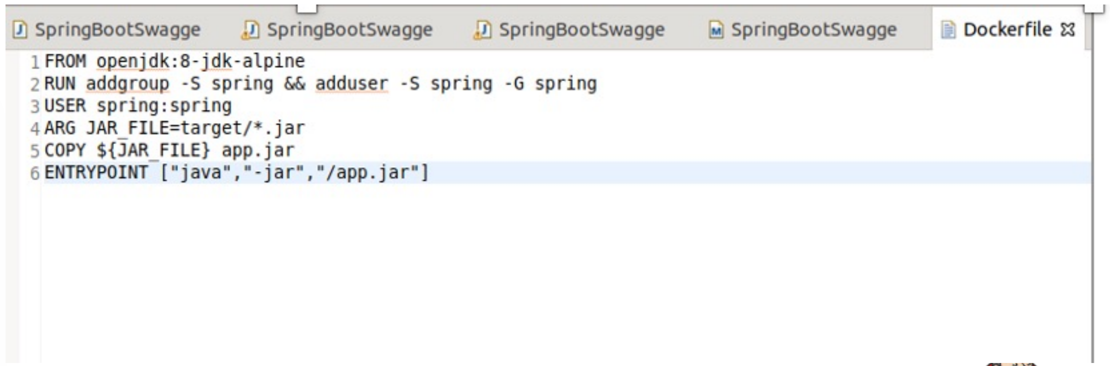
Managing User defined bridge network

- docker network create -d bridge my-bridge-network (Basic command)
- docker network rm my-net (To remove the network)
- docker network create -d overlay my-bridge-network (To connect to multiple daemons)
- docker network create --driver=bridge --subnet=172.28.0.0/16 --ip-range=172.28.5.0/24 --gateway=172.28.5.254 my-bridge-network
- docker create --name my-nginx --network my-bridge-network --publish 8080:80 nginx:latest (To create a container without start state)
- docker network disconnect **docker0** my-nginx

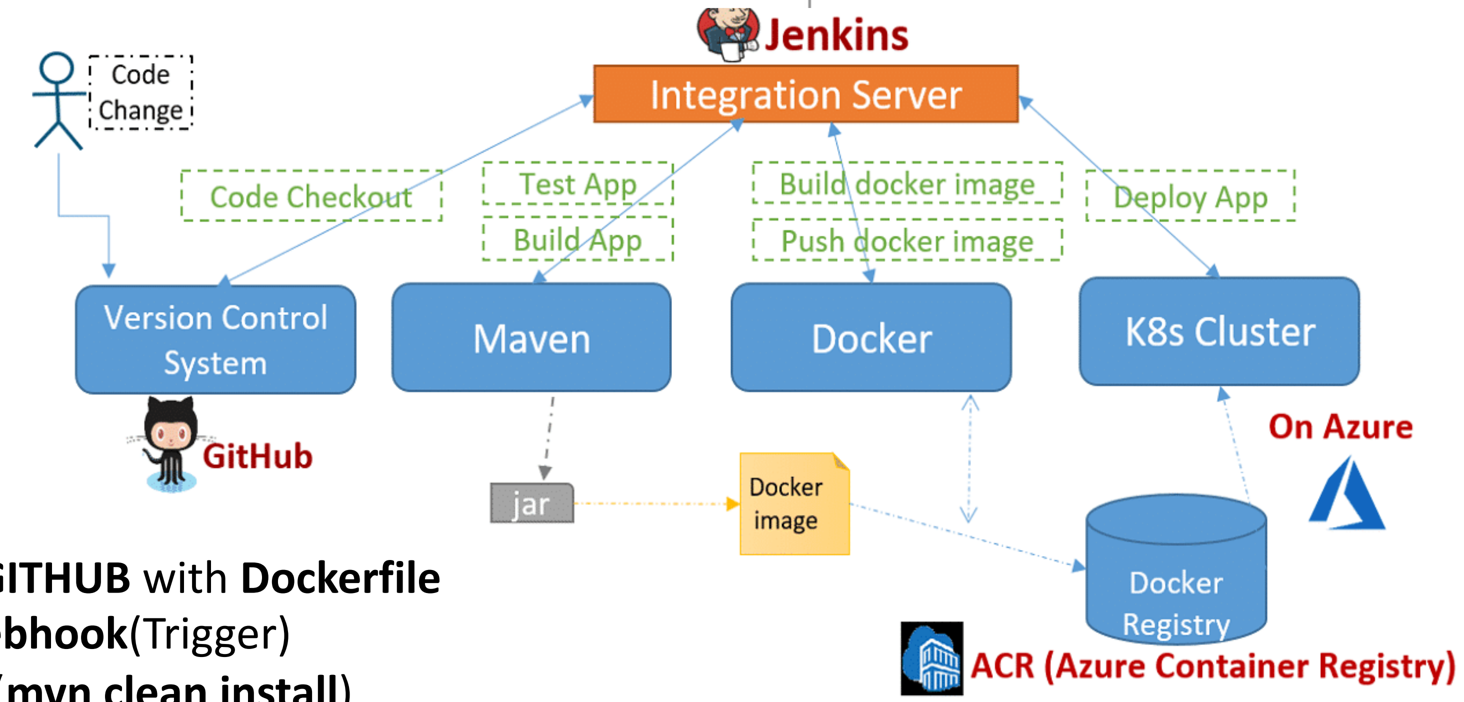
DockerFile



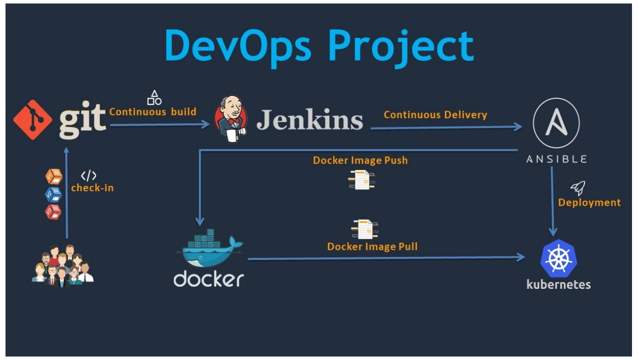
```
spring-boot-docker-example [boot]
  src/main/java
    com.javabydeveloper.demo
      HelloController.java
      SpringBootHelloWorldApplication.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
    generated-sources
    maven-archiver
    maven-status
    spring-boot-docker-example-0.0.1-SNAPSHOT.jar
    spring-boot-docker-example-0.0.1-SNAPSHOT.jar.original
  docker-compose.yml
  Dockerfile
  pom.xml
```

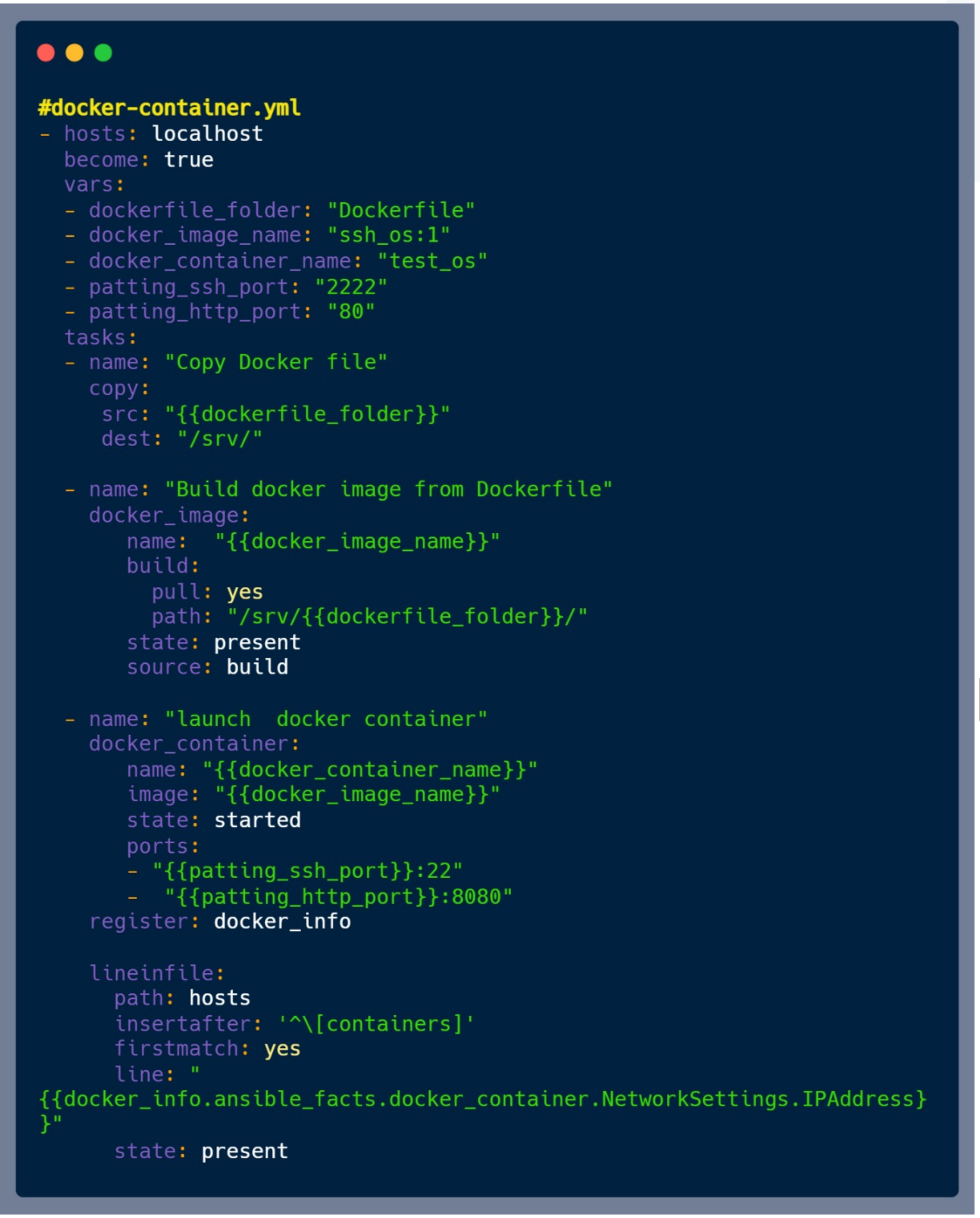


```
FROM openjdk:8-jdk-alpine
RUN addgroup -S spring && adduser -S spring -G spring
USER spring:spring
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```



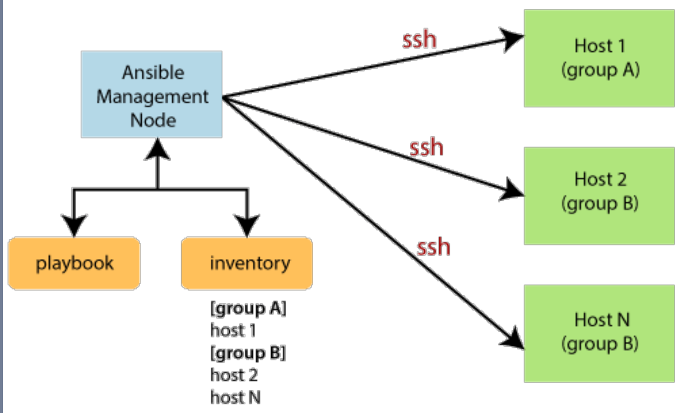
Stage 1 - Developer commits the JAVA code in **GITHUB** with **Dockerfile**
Stage 2 - **Jenkins** will check for changes with **webhook**(Trigger)
Stage 3 - **Maven** build happens with command (**mvn clean install**)
Stage 4 - **Docker build** command is given in Jenkins (docker build .)
  Stage 4.1 -The **docker image** is created with the help of dockerfile
  Stage 4.2 - **Docker push to dockerregistry or Jfrog**
Stage 5(Deploy App) - **ANSIBLE** (Which will pull the image from dockerregistry and deploy into K8s)





```
#docker-container.yml
- hosts: localhost
  become: true
  vars:
  - dockerfile_folder: "Dockerfile"
  - docker_image_name: "ssh_os:1"
  - docker_container_name: "test_os"
  - patting_ssh_port: "2222"
  - patting_http_port: "80"
  tasks:
  - name: "Copy Docker file"
    copy:
     src: "{{dockerfile_folder}}"
     dest: "/srv/"

  - name: "Build docker image from Dockerfile"
    docker_image:
       name:  "{{docker_image_name}}"
       build:
         pull: yes
         path: "/srv/{{dockerfile_folder}}/"
       state: present
       source: build

  - name: "launch  docker container"
    docker_container:
       name: "{{docker_container_name}}"
       image: "{{docker_image_name}}"
       state: started
       ports:
       - "{{patting_ssh_port}}:22"
       -  "{{patting_http_port}}:8080"
    register: docker_info

    lineinfile:
      path: hosts
      insertafter: '^\[containers]'
      firstmatch: yes
      line: "
{{docker_info.ansible_facts.docker_container.NetworkSettings.IPAddress}
}"
      state: present
```

```
- name: Download file from artifactory
  get_url:
    url: http://website:port/artifactory/platform/httpd/
    headers:
    username: admin
    password: password
    dest: /home/user_name/destination_location
    mode: 0644
```



```
- name: Tag and push to docker hub
  docker_image:
    name: pacur/centos-7
    repository: dcoppenhagan/myimage
    tag: 7.0
    push: yes

- name: Tag and push to local registry
  docker_image:
     name: centos
     repository: localhost:5000/centos
     tag: 7
     push: yes

- name: Remove image
  docker_image:
    state: absent
    name: registry.ansible.com/chouseknecht/sinatra
    tag: v1

- name: Build an image and push it to a private repo
  docker_image:
    path: ./sinatra
    name: registry.ansible.com/chouseknecht/sinatra
    tag: v1
    push: yes

- name: Archive image
  docker_image:
    name: registry.ansible.com/chouseknecht/sinatra
    tag: v1
    archive_path: my_sinatra.tar

- name: Load image from archive and push to a private registry
  docker_image:
    name: localhost:5000/myimages/sinatra
    tag: v1
    push: yes
    load_path: my_sinatra.tar

- name: Build image and with buildargs
  docker_image:
     path: /path/to/build/dir
     name: myimage
     buildargs:
       log_volume: /var/log/myapp
       listen_port: 8080
```

Thank you all for joining today's session ☺☺☺