



# Achieving Continuous DevOps – CI/CD

---

NAGP – Experienced Batch'20

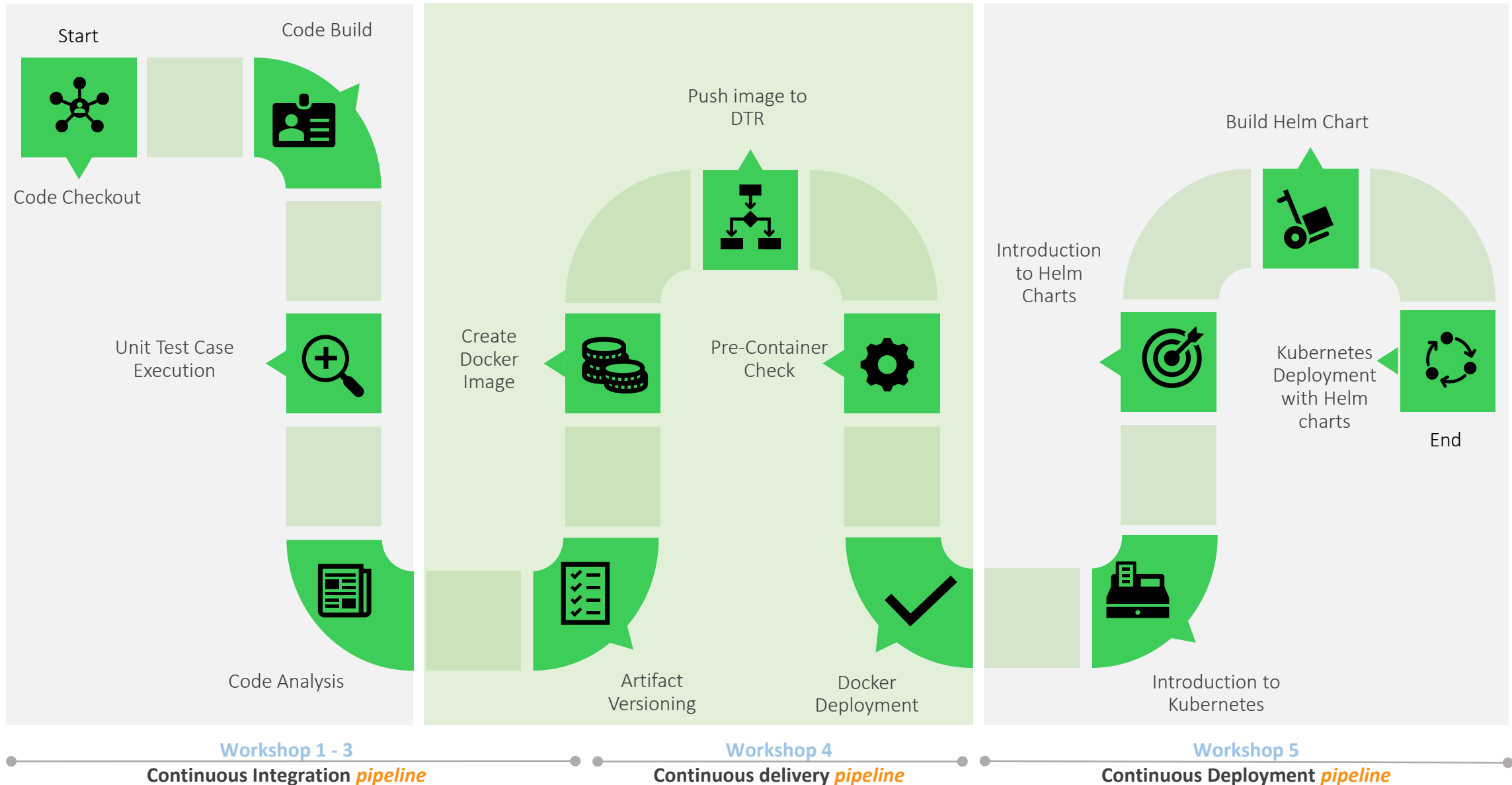


THINKING  
BREAKTHROUGHS

---

# Achieving Continuous DevOps ...Recap

## Agenda



# Sample Jenkinsfile

An example Jenkinsfile looks like below

*Jenkinsfile (Declarative Pipeline)*

```
pipeline { ❶
  agent any ❷
  options {
    skipStagesAfterUnstable()
  }
  stages {
    stage('Build') { ❸
      steps { ❹
        sh 'make' ❺
      }
    }
    stage('Test'){
      steps {
        sh 'make check'
        junit 'reports/**/*.xml' ❻
      }
    }
    stage('Deploy') {
      steps {
        sh 'make publish'
      }
    }
  }
}
```

- ❶ `pipeline` is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.
  - ❷ `agent` is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline.
  - ❸ `stage` is a syntax block that describes a [stage of this Pipeline](#). Read more about `stage` blocks in Declarative Pipeline syntax on the [Pipeline syntax](#) page. As mentioned [above](#), `stage` blocks are optional in Scripted Pipeline syntax.
  - ❹ `steps` is Declarative Pipeline-specific syntax that describes the steps to be run in this `stage`.
  - ❺ `sh` is a Pipeline [step](#) (provided by the [Pipeline: Nodes and Processes plugin](#)) that executes the given shell command.
  - ❻ `junit` is another a Pipeline [step](#) (provided by the [JUnit plugin](#)) for aggregating test reports.
- `node` is Scripted Pipeline-specific syntax that instructs Jenkins to execute this Pipeline (and any stages contained within it), on any available agent/node. This is effectively equivalent to `agent` in Declarative Pipeline-specific syntax.

# Achieving Continuous Integration Pipeline (CI)

Code checkout → code build → AUT → sonar code coverage → Artifact upload

**A** Click on new Item on the left side of Jenkins,

Enter an item name

com.nagp2020.emp.name.emp.id **1**

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

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

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

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

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

Bitbucket Team/Project  
Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.

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

**3** OK ☒ Add to current view  
GitHub Organization

**1** Type pipeline name following above naming convention

**2** Select pipeline as type of job

**3** Click OK with checked Add to current view

**B** Write pipeline script, and click Save

Jenkins > com.nagp2019.shreysangal.3146997 >

General Build Triggers Advanced Project Options **Pipeline**

Definition Pipeline script

Script

```
1 pipeline{
2   agent any
3   stages{
4     stage('Code checkout'){
5       steps{
6         checkout([
7           $class: 'GitSCM',
8           branches: [[name: '*/master']],
9           doGenerateSubmoduleConfig: false,
10          extensions: [],
11          submoduleCfg: [],
12          userRemoteConfigs: [[
13            credentialsId: 'Git_credentials',
14            url: 'https://git.nagp2020.com:8443/emp.name/emp.id'
15          ]]
16        })
17      }
18    }
19  }
```

**a** Save Apply

☒ Use Groovy Sandbox

**a** Code checkout

**b** Code build

**c** Automated unit testing

**d** Code Analysis

**e** Artifact versioning

# Achieving Continuous Integration Pipeline (CI).. contd

Code checkout → code build → AUT → sonar code coverage → Artifact upload



Click Build now, and check console logs

A screenshot of the Jenkins web interface showing the console output of a pipeline. The Jenkins logo and name are at the top, followed by a red box with the number '4' and a search bar. Below the header, the console output is displayed in a monospaced font. The output shows the pipeline starting, running on a test node, and executing several stages: Code checkout, Code build, AUT, Code Analysis, and Artifact Versioning. Each stage is marked with a letter in a black circle (a, b, c, d, e). The pipeline ends with 'Finished: SUCCESS'.

```
Jenkins
com.nagp2019.shreysangal.3146997 #7

Console Output


Started by user Shrey Sangal
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on test in /home/anupamagarwal/workspace/com.nagp2019.shreysangal.3146997
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Code checkout) a
[Pipeline] sh
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Code build) b
[Pipeline] sh
[Pipeline] stage
[Pipeline] { (AUT) c
[Pipeline] sh
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Code Analysis) d
[Pipeline] sh
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Artifact Versioning) e
[Pipeline] sh
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

# Automated unit testing

Jenkins pipeline triggers automated unit testing

Navigation steps:

1. Goto test results on the left pane
2. Click on test module


 **Jenkins**


4


[?](#)


Shrey Sangal | [log out](#)


Jenkins > com.nagp2019.shreysangal.3146997 > #8 > Test Results > com.devopssample.service > TestClass [ENABLE AUTO REFRESH](#)


 History


 Git Build Data


 No Tags


 Test Result


 Artifactory Build Info


 Open Blue Ocean

 Restart from Stage

 Replay


 Pipeline Steps

 Workspaces

 Previous Build

## Test Result : TestClass

0 failures (±0)

3 tests (±0)  
[Took 19 ms.](#)  
 [add description](#)

### All Tests

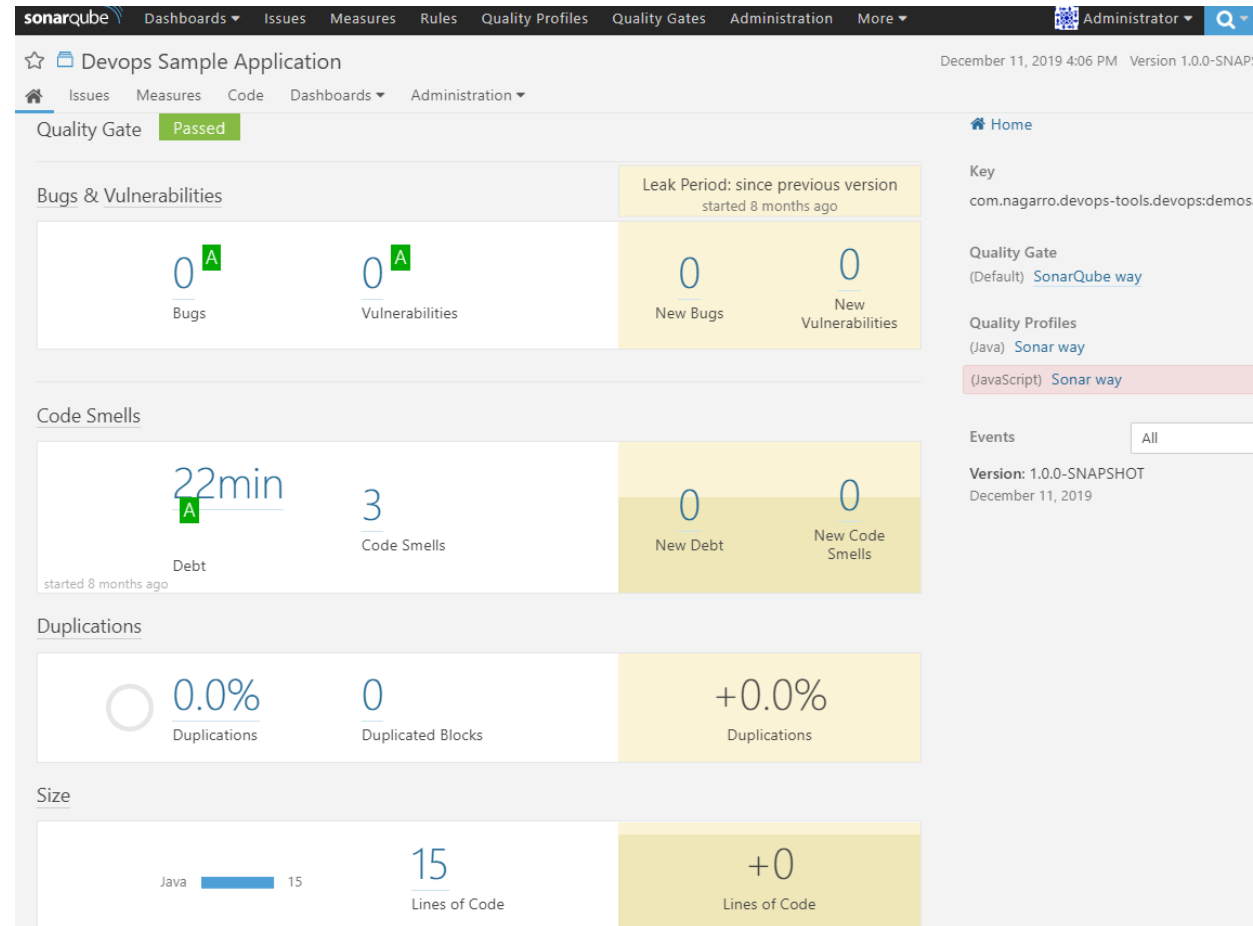
Test name	Duration	Status
<a href="#">testAdd</a>	16 ms	Passed
<a href="#">testFailedAdd</a>	2 ms	Passed
<a href="#">testSub</a>	1 ms	Passed

# Code Coverage

Sonarqube code coverage execution within Jenkins pipeline

Navigation steps:

1. Navigate to **URL**
2. Login using your AD credentials



# Artifact Versioning

Artifact get uploaded from jenkins build pipeline

Navigation steps:

1. Navigate to **URL**
2. Login using your AD credentials

The screenshot displays the JFrog Artifactory web interface. The top navigation bar is green with the JFrog logo, a search icon, and links for 'Help' and 'Welcome, admin'. Below this is a grey header for the 'Artifact Repository Browser' with 'Set Me Up' and 'Deploy' links. The left sidebar contains icons for home, library, repository, search, settings, and user profile. The main content area is divided into two panels. The left panel shows a tree view of the repository structure: 'workshop' > 'com' > 'devopssampleapplication' > 'META-INF', 'resources', and 'WEB-INF'. The right panel shows the details for the selected artifact, 'devopssampleapplication.war'. It includes a 'Download' link and an 'Actions' dropdown. The 'General' tab is active, displaying the following information:

Info	
Name:	devopssampleapplication.war
Repository Path:	workshop/devopssampleapplication.war
Module ID:	N/A
Deployed By:	admin
Size:	25.56 MB
Created:	01-08-19 12:57:32 +05:30
Last Modified:	01-08-19 12:57:31 +05:30
Downloads:	1



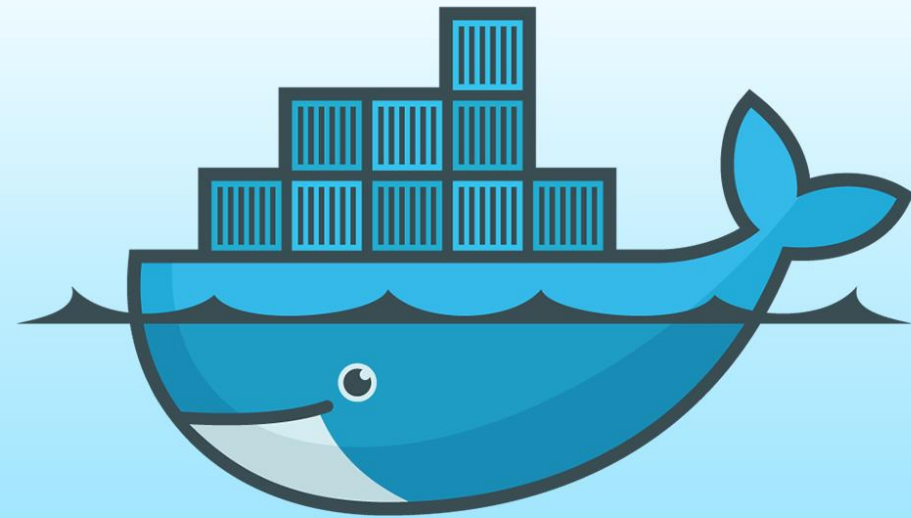
# Overview

## What is Docker

Docker is a container management service. The keywords of Docker are **develop**, **ship** and **run** anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere. The initial release of Docker was in **March 2013** and since then, it has become the buzzword for modern world development, especially in the face of Agile-based projects.

### Build, Ship and Run Any App, Anywhere

Docker - An open platform for distributed applications for developers and sysadmins.

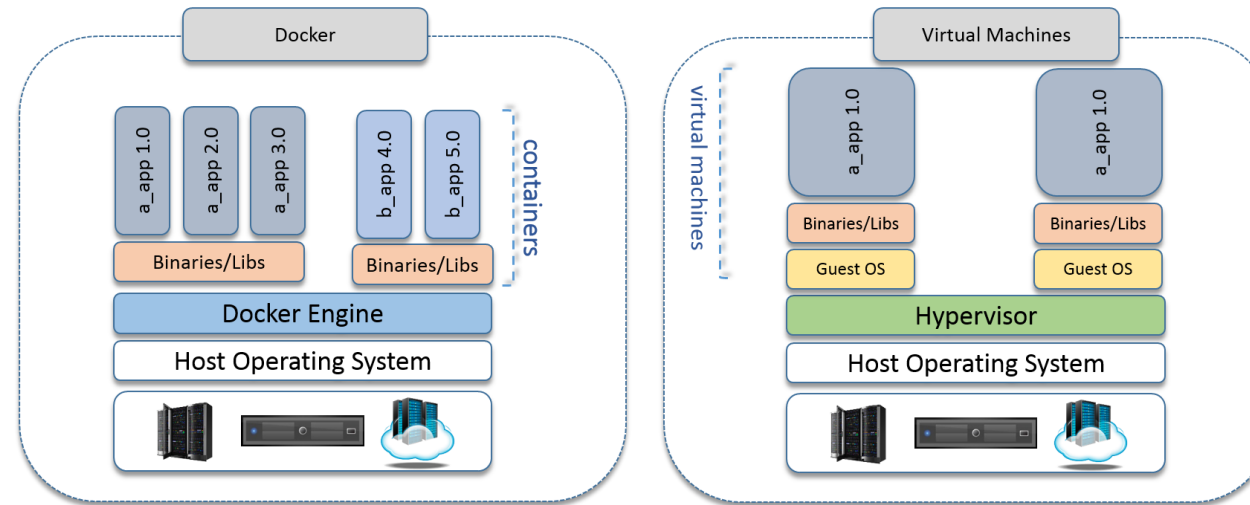
[What is Docker?](#)[Try It!](#)

# How Docker is different from VMs

Docker allows to package an application with all of its dependencies into a standardized unit for software development.

- Virtual machines

- Have a full **OS** with its own memory management installed with the associated overhead of virtual device drivers.
- In a virtual machine, valuable resources are **emulated** for the guest OS and **hypervisor**, which makes it possible to run many instances of one or more operating systems in parallel on a single machine (or host).
- Every guest OS runs as an **individual entity** from the host system.



- On the other hand Docker containers

- Are executed with the **Docker engine** rather than the hypervisor.
- Containers are therefore **smaller** than Virtual Machines and enable **faster** start up with better **performance**, less isolation and greater compatibility possible due to sharing of the host's kernel.
- Docker Containers are able to **share** a single **kernel** and share application **libraries**

# Docker images

A Docker image is built up from a series of layers and each layer contains a change..

Each layer represents an instruction in the image's Dockerfile.

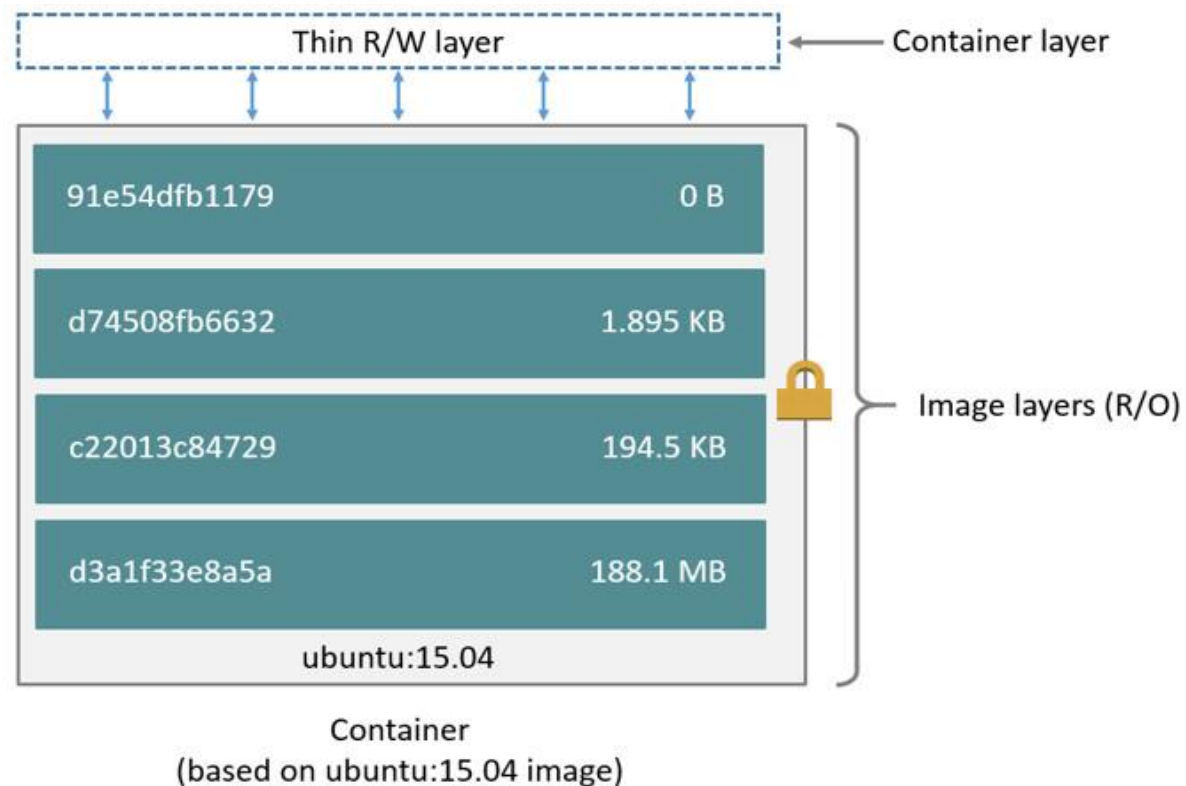
Consider the following Dockerfile:

```
FROM ubuntu:15.04
```

```
COPY ./app
```

```
RUN make /app
```

```
CMD python /app/app.py
```



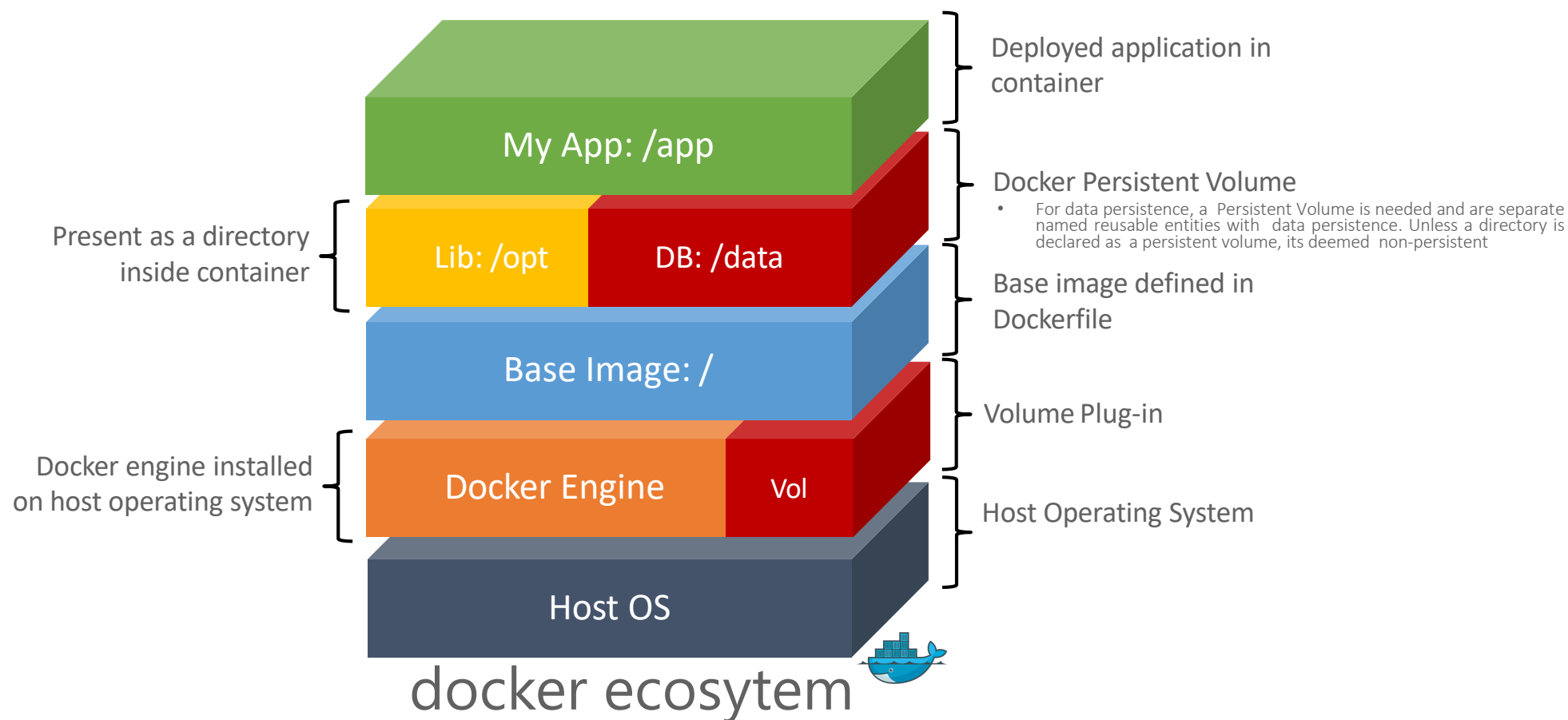
This Dockerfile contains four commands, each of which creates a layer. Each layer is only a set of differences from the layer before it. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers.

TAR (archive) file containing all the binary and configuration files needed to run a container eg: docker save/load.

# Containers

- Each container has its own
- Root filesystem
  - Processes
  - Memory
  - Devices
  - Network ports

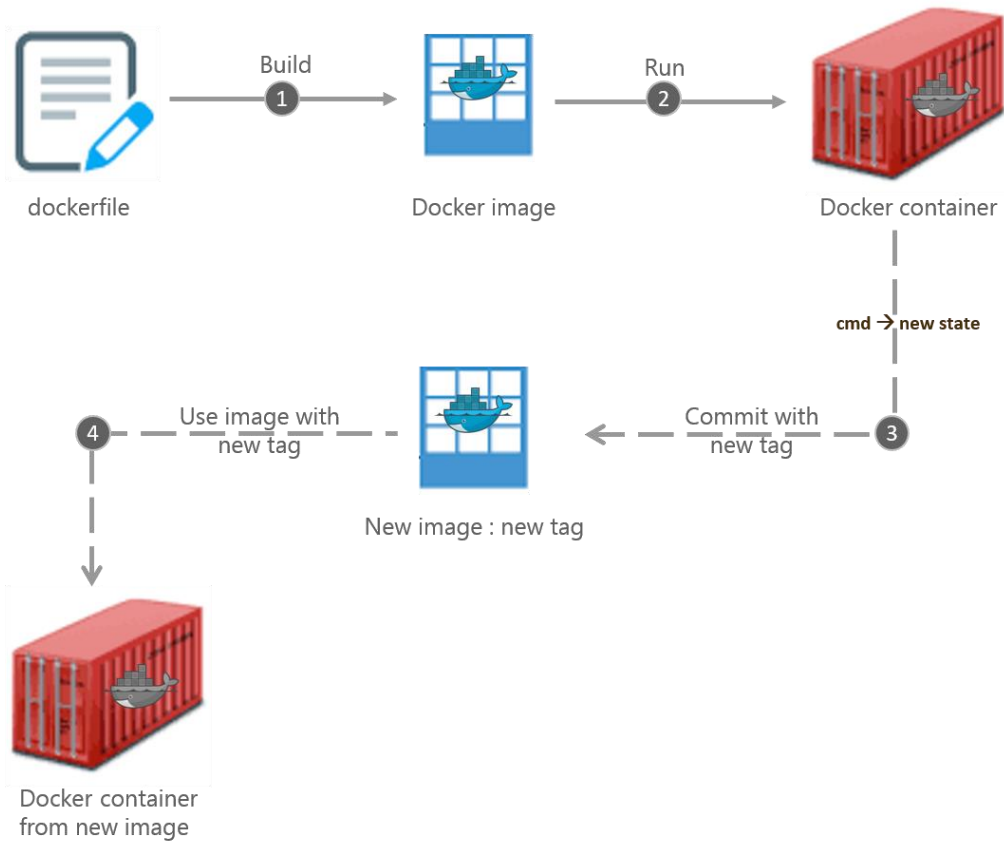
Containers are an operating system level virtualization technology for providing multiple isolated environments on a single host. Unlike virtual machines (VMs), containers do not run dedicated guest operating systems. Rather, they share the host operating system kernel and make use of the guest operating system libraries for providing the required OS capabilities. Since there is no dedicated operating system, containers start much faster than VMs.



*Note that whilst persistent volumes can be assigned to more than one container concurrently; you need to manage data change collisions/conflicts as there is no locking mechanism in Docker*

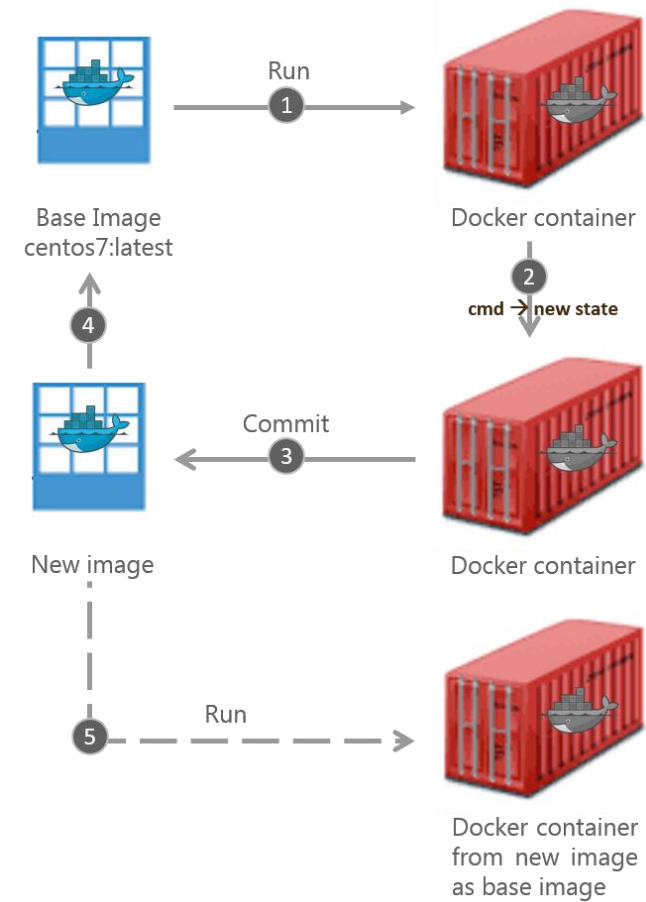
# Image vs container

## Scenario 1



Source: Dockerfile

## Scenario 2



Source: Base docker Image

# Docker registry

## Docker Hub

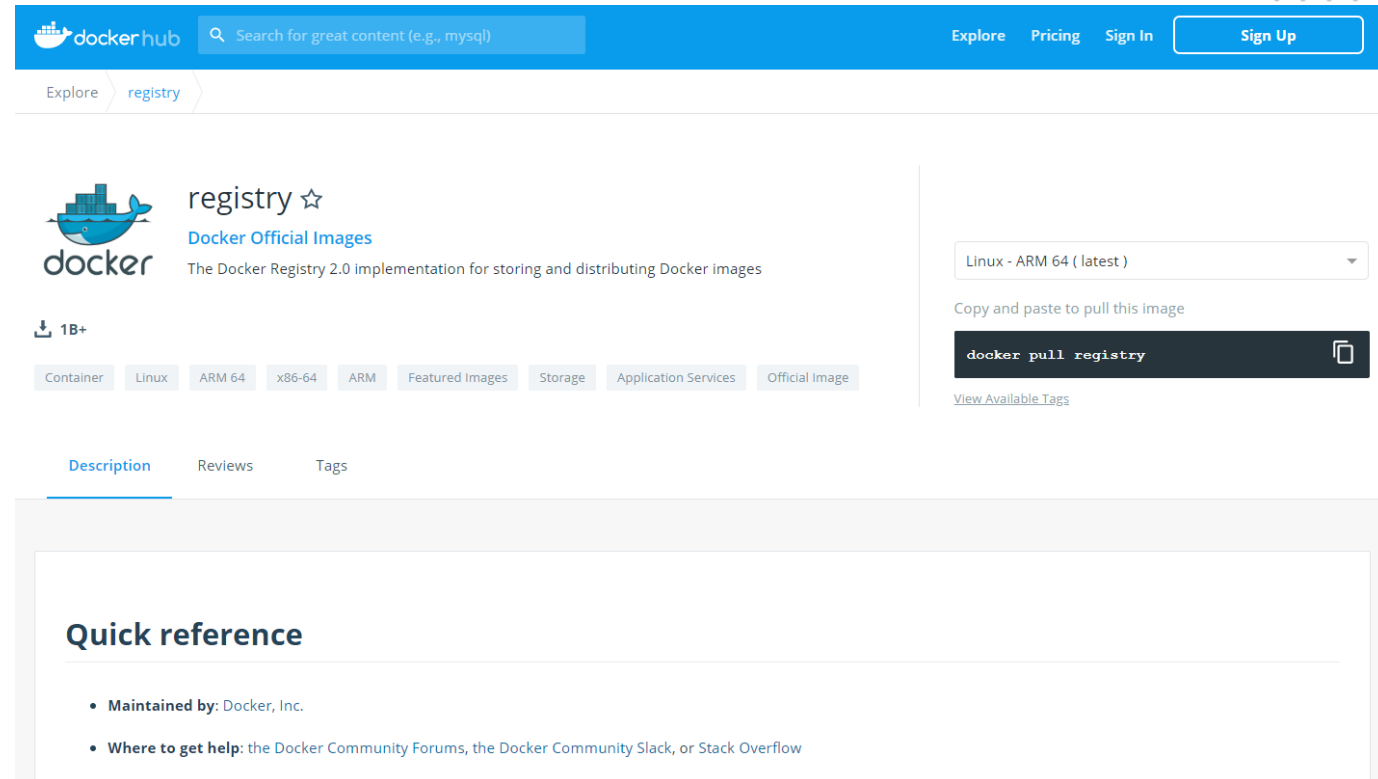
Docker Hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution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

**Example:** `docker pull ubuntu` instructs docker to pull an image named ubuntu from the official Docker Hub. This is simply a shortcut for the longer `docker pull docker.io/library/ubuntu` command

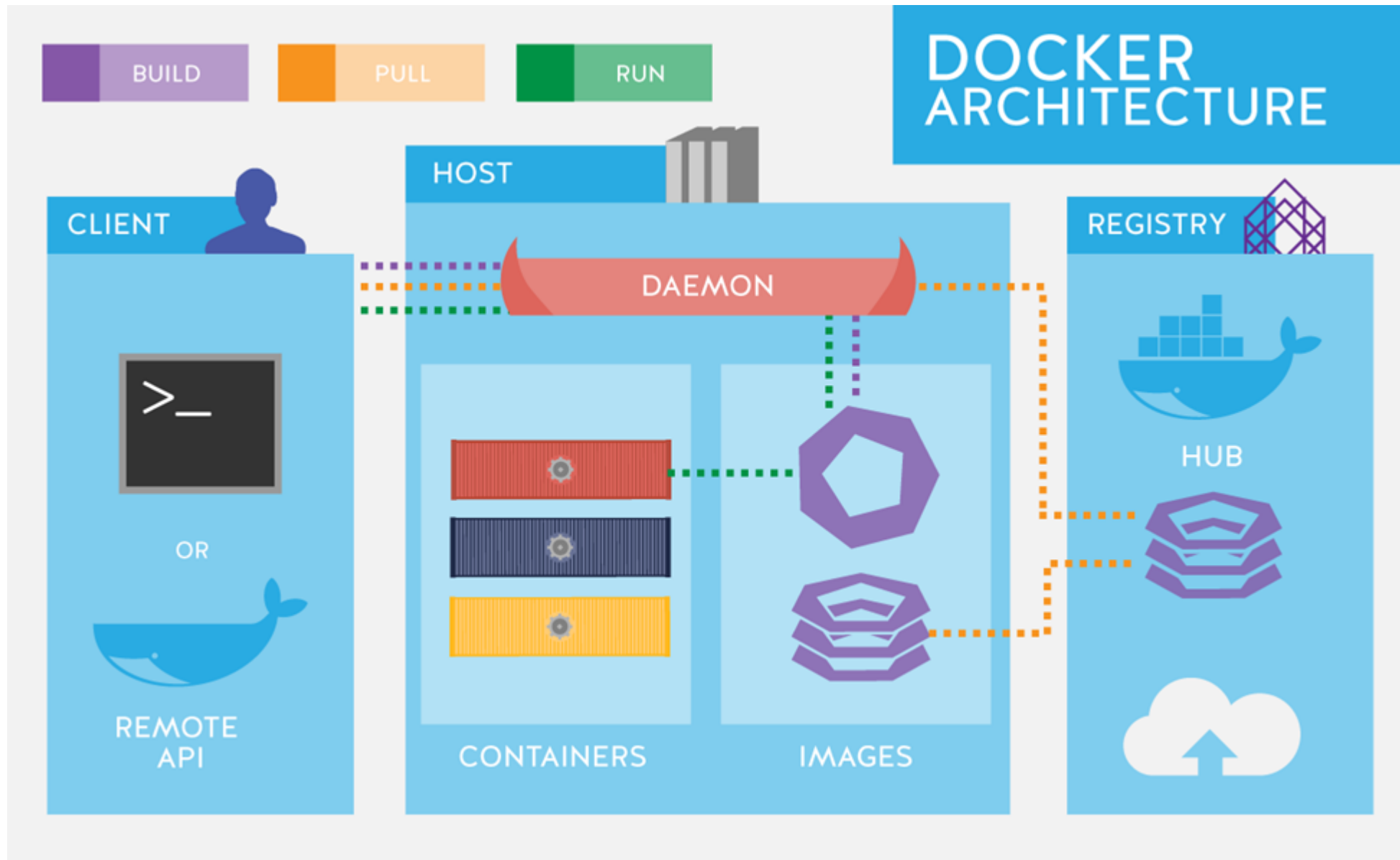
## Private Registry

The Registry is a stateless, highly scalable server-side application that stores and lets you distribute Docker images. The Registry is open-source, under the permissive Apache license.

**Example:** `docker pull myregistrydomain:port/foo/bar` instructs docker to contact the registry located at `myregistrydomain:port` to find the image `foo/bar`

The screenshot shows the Docker Hub interface for the 'registry' image. At the top is a blue navigation bar with the Docker Hub logo, a search bar, and links for 'Explore', 'Pricing', 'Sign In', and 'Sign Up'. Below the navigation bar, the 'registry' page is displayed. It features the Docker logo, the word 'registry' with a star, and the text 'Docker Official Images' and 'The Docker Registry 2.0 Implementation for storing and distributing Docker Images'. There are 18+ downloads indicated. A horizontal menu includes 'Container', 'Linux', 'ARM 64', 'x86-64', 'ARM', 'Featured Images', 'Storage', 'Application Services', and 'Official Image'. The 'Description' tab is selected. On the right, a dropdown menu shows 'Linux - ARM 64 ( latest )'. Below it, a text box says 'Copy and paste to pull this image' with a code block containing 'docker pull registry' and a copy icon. A link 'View Available Tags' is also present. At the bottom, a 'Quick reference' section lists: 'Maintained by: Docker, Inc.' and 'Where to get help: the Docker Community Forums, the Docker Community Slack, or Stack Overflow'.

# Docker architecture



# Docker commands

Command	Function
<code>docker info</code>	Show information on the docker engine
<code>docker version</code>	Show the version of docker running
<code>docker run</code>	Start / deploy new containers
<code>docker ps</code> (and <code>ps -a</code> )	Show running (or stopped) containers
<code>docker stats &lt;containerid&gt;</code>	Show performance info for a container
<code>docker logs &lt;containerid&gt;</code>	Show logs for a container
<code>docker rename</code> (because we always forget to name our containers)	Rename a deployed container
<code>docker start</code> , <code>docker stop</code>	Stop and start an existing container
<code>docker kill</code>	Hard stop (kill) a running container
<code>docker rm</code> (and <code>rmi</code> )	Delete a container or image
<code>docker network</code>	Work with docker networks
<code>docker volume</code>	Work with docker volumes
<code>docker exec</code> / <code>docker attach</code>	Interact with a running container
<code>docker system prune</code>	Delete all “unused” containers/volumes/images

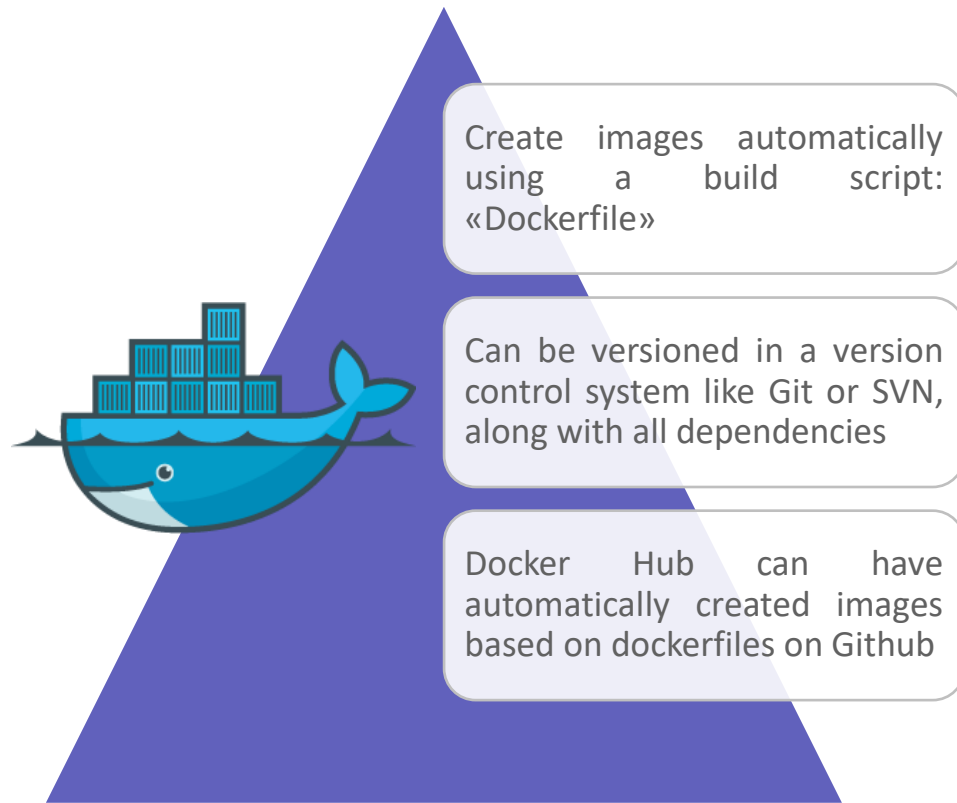


# Common docker run Options

Command	Function
--name	Give a friendly name to the container <i>Note double dashes</i>
-d	Run the container in the background (detached)
-i -t	Start the container interactive, and with tty support
--restart= always/on-failure/unless-stopped/no	Restart the container automatically when the condition is met (note this is not a cluster restart policy, only on a single host) <i>Note double dashes</i>
-v	Map a volume to a container
-p ip:port:port / -P	Publish ports externally (hostport:containerport) <ul style="list-style-type: none"><li>• -P means publish all ports defined in the dockerfile.</li><li>• Using -p :80 means randomly assign a host port</li><li>• Using -p 192.168.1.20:80:80 means expose as port 80 on the hosts ip address 192.168.1.20 (if the host had more than 1 IP)</li></ul>
-e	Pass an environment variable into the container

# Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.



## Example 1

```
FROM tomcat:8
MAINTAINER Devops Team
RUN wget -O /usr/local/tomcat/webapps/demosampleapplication.war
<WAR FILE URL>
CMD /usr/local/tomcat/bin/catalina.sh run
```

## Example 2

```
FROM ubuntu
MAINTAINER XYZ
RUN apt-get update
RUN apt-get install -y nginx
COPY index.html /usr/share/nginx/html/
ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]
EXPOSE 80
```

# Docker – Example 1

## Checking container logs

- docker ps (see the container id)
- docker logs <containerid>

```
[root@localhost html]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
71f50830826c       nginx:latest       "nginx -g 'daemon of..." 6 minutes ago
, 0.0.0.0:443->443/tcp mynginx
[root@localhost html]#
```

```
[root@localhost html]# docker logs 71f
2018/11/19 19:01:49 [error] 6#6: *1 directory index of "/usr/share/nginx/html/"
: localhost, request: "GET / HTTP/1.1", host: "192.168.1.71"
192.168.1.253 - - [19/Nov/2018:19:01:49 +0000] "GET / HTTP/1.1" 403 153 "-" "Moz
.0) Gecko/20100101 Firefox/63.0" "-"
2018/11/19 19:01:49 [error] 6#6: *1 open() "/usr/share/nginx/html/favicon.ico" f
t: 192.168.1.253, server: localhost, request: "GET /favicon.ico HTTP/1.1", host:
192.168.1.253 - - [19/Nov/2018:19:01:49 +0000] "GET /favicon.ico HTTP/1.1" 404 1
x64; rv:63.0) Gecko/20100101 Firefox/63.0" "-"
192.168.1.253 - - [19/Nov/2018:19:05:46 +0000] "GET / HTTP/1.1" 200 15 "-" "Mozi
0) Gecko/20100101 Firefox/63.0" "-"
2018/11/19 19:05:46 [error] 6#6: *2 open() "/usr/share/nginx/html/favicon.ico" f
t: 192.168.1.253, server: localhost, request: "GET /favicon.ico HTTP/1.1", host:
192.168.1.253 - - [19/Nov/2018:19:05:46 +0000] "GET /favicon.ico HTTP/1.1" 404 1
x64; rv:63.0) Gecko/20100101 Firefox/63.0" "-"
[root@localhost html]#
```

# Docker – Example 2

## Docker attach

- Get your container id again
- Docker attach <containerid>
  - Note you just get a flashing cursor – you are inside the container, however there is no shell..
  - Press ctrl c to exit
  - Now do a docker ps
  - What happened to your container ? It got killed as you send the kill command whilst attached to the container
- Do docker ps -a to see stopped containers
- Do a docker start <containerid> to restart

```
[root@localhost html]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
71f50830826c       nginx:latest       "nginx -g 'daemon of..." 6 minutes ago
, 0.0.0.0:443->443/tcp mynginx
[root@localhost html]#
```

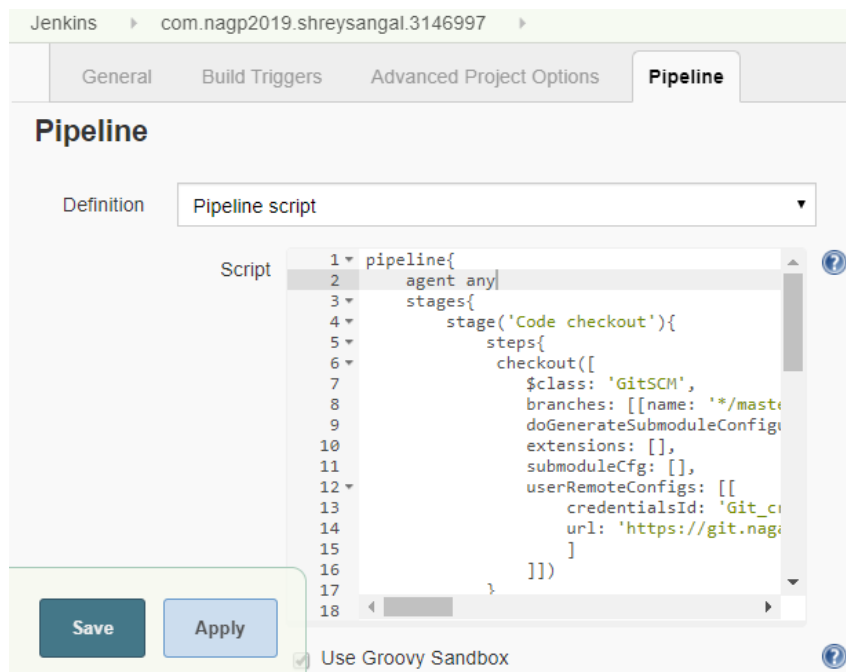
```
[root@localhost html]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
71f50830826c       nginx:latest       "nginx -g 'daemon of..." 11 minutes ago
, 0.0.0.0:443->443/tcp mynginx
[root@localhost html]# docker attach 71f

^C[root@localhost html]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
ES
[root@localhost html]#
```

# Extending CI Pipeline with Docker

Existing CI Pipeline → Create Docker image → Push image to DTR → Pre-Container Check → Docker deployment

- 1 Extend pipeline script with docker stages, and click Save



- a Create Docker Image
- b Push Docker Image to DTR
- c Pre-Container Check
- d Docker Deployment

- 2 Click Build now, and check console logs

