

DOCKER

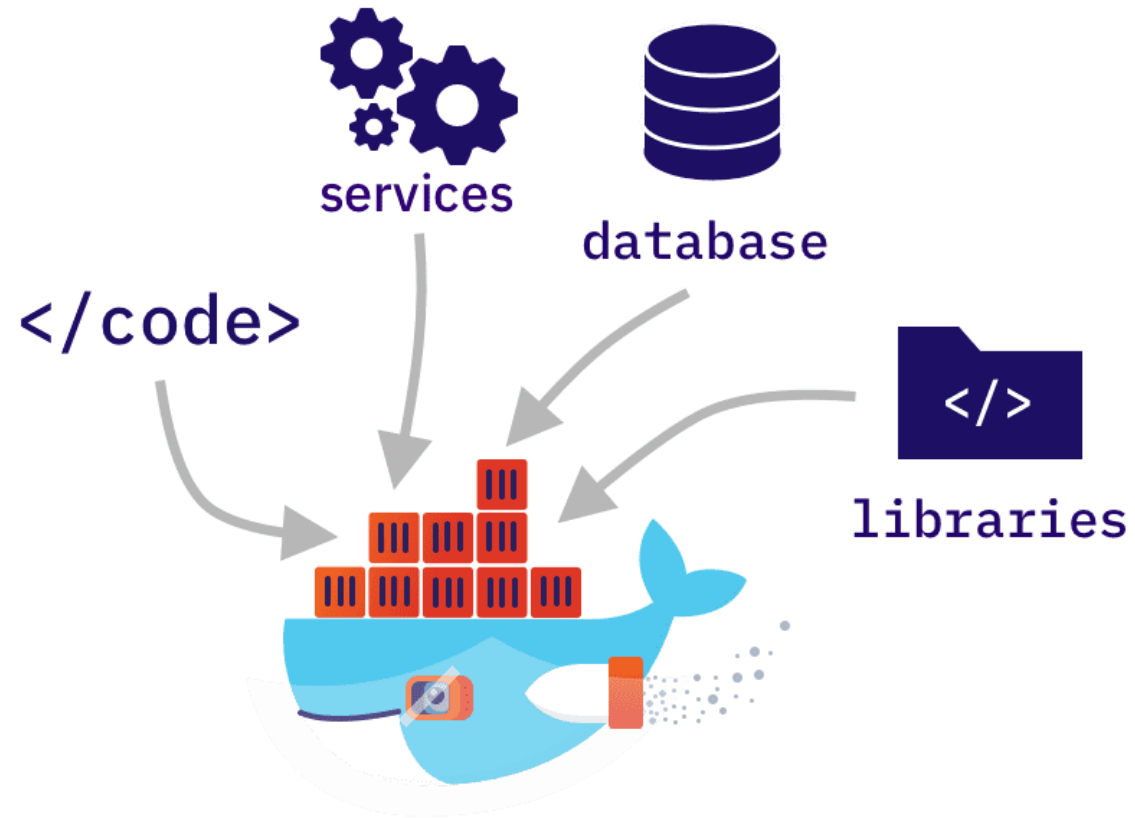




Docker - Introduction

- Tool that allows developers to **bundle all the ingredients of software they create into unified boxes** - called Docker containers.
- The key concern is usually the code, but most apps or websites won't work without a database, configuration files, runtime libraries or other third-party software.
- Putting these together in a box makes it **easier to launch the app on another developer's computer or server.**

Docker





Docker - History

- Docker was originally created by [Solomon Hykes](#) from dotCloud Inc. (later [Docker Inc.](#)) as an open-source project and since its launch in 2013 Docker is available for free.
- In March 2017 the open-source version of Docker was rebranded to Docker CE (Community Edition) and Docker Enterprise Edition (EE) was released.

Docker File

- Docker builds images automatically by reading the instructions from a Dockerfile which is a text file that contains all commands, in order, needed to build a given image.
- A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.

Docker File

innovate

achieve

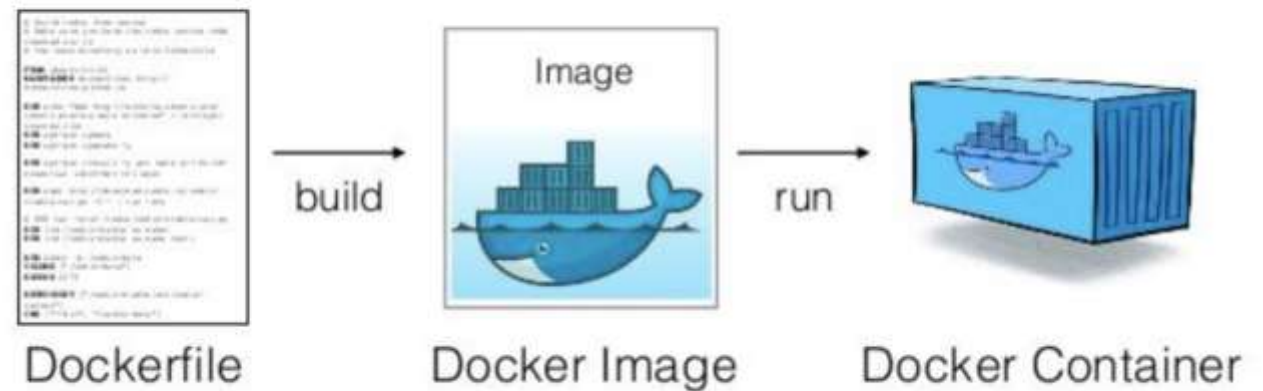
lead

Dockerfile - myhttpd:0.1

```
1 # A simple web app served by httpd
2 FROM httpd:2.4
3
4 LABEL AUTHOR=user@example.com
5
6 LABEL VERSION=0.1
7
8 # COPY mypage.html /usr/local/apache2/htdocs/mypage.html
9 # WORKDIR /usr/local/apache2
10
11 COPY mypage.html htdocs/mypage.html
```

Docker - Images

- Docker images are a lightweight, standalone, executable package of software that includes everything needed to run an application, code, runtime, system tools, system libraries and settings.





Docker Container

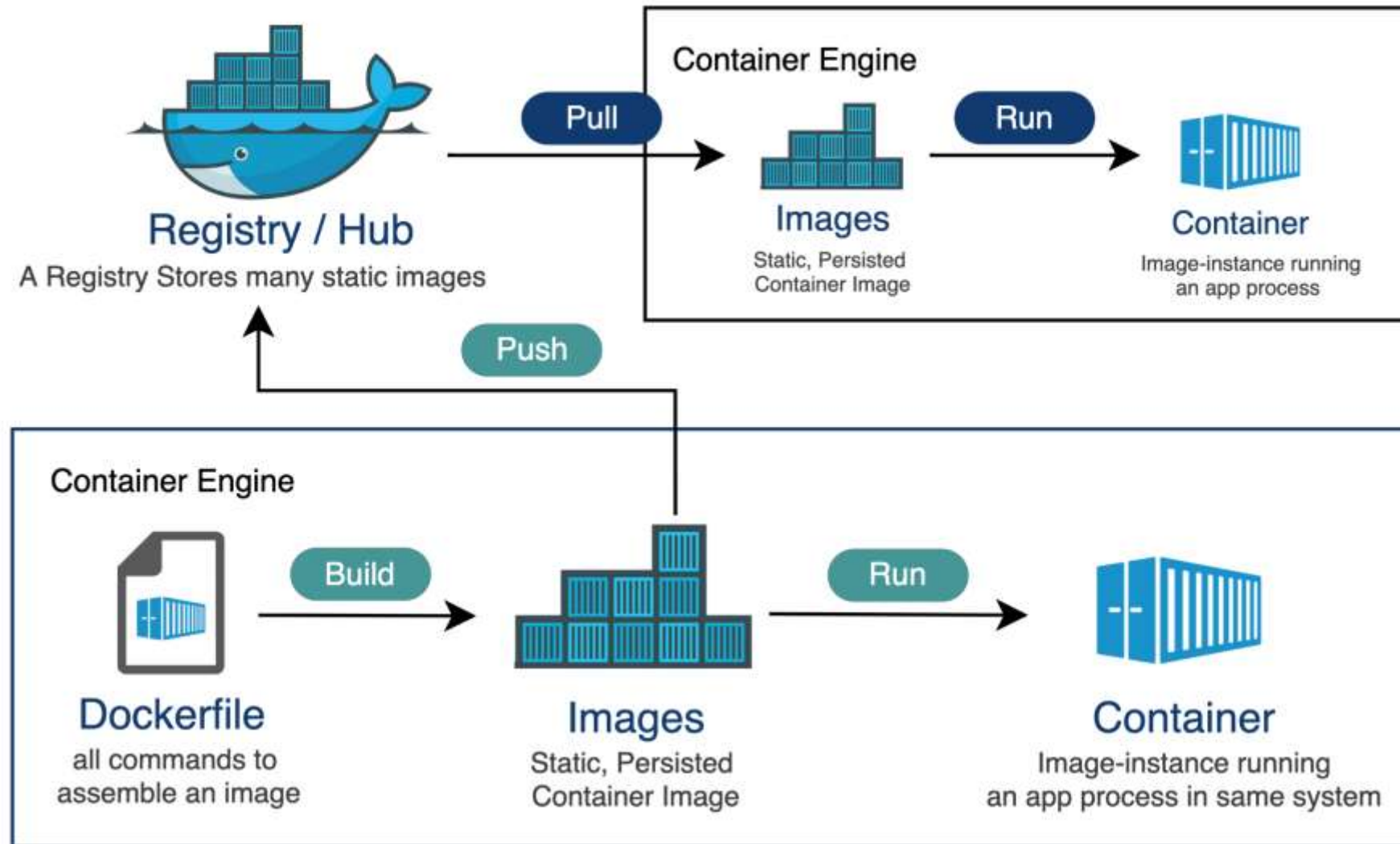
- A container is a runtime instance of a docker image. A container will always run the same, regardless of the infrastructure.
- Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.



Docker Hub

- Docker Hub is a service provided by Docker for finding and sharing container images with team.
- Readily available images can be found at <https://hub.docker.com>

Putting all Docker terminologies together



Docker Desktop - Containers



Docker Desktop interface showing the Containers tab. The left sidebar contains navigation options: Containers, Images, Volumes, Dev Environments (BETA), Docker Scout, Learning center, Extensions, and Add Extensions. The main area displays a table of containers. Above the table, there are sections for Container CPU usage and Container memory usage, both indicating 'No containers are running.' and a 'Show charts' link. A search bar and a toggle for 'Only show running containers' are also present.

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	awesome_noyce ab5d2606c45e	504568707713.dkr.ecr.ap-south-1.amazon	Exited (255)	N/A	80:80	18 days ago	▶ ⋮ 🗑
<input type="checkbox"/>	jolly_mirzakhani 2b37963de452	504568707713.dkr.ecr.ap-south-1.amazon	Exited	N/A		18 days ago	▶ ⋮ 🗑

Showing 2 items

RAM 0.00 GB CPU 0.00% Not signed in v4.23.0 2

Docker Desktop - Images



Docker Desktop interface showing the Images section.

Search for images, containers, volumes, extensions and more... **Ctrl+K**

Containers
Images
Volumes
Dev Environments **BETA**
Docker Scout
Learning center
Extensions
Add Extensions

Images [Give feedback](#)

Local Hub Artifactory **EARLY ACCESS**

12.04 GB / 14.22 GB in use 3 images Last refresh: 11 minutes ago

Search

<input type="checkbox"/>	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	504568707713.dkr.ecr.ap-south-1.amazonaws.com/charakdtrepo 2795d328c005	nov21	Unused	17 days ago	6.48 GB	▶ ⋮ 🗑️
<input type="checkbox"/>	504568707713.dkr.ecr.ap-south-1.amazonaws.com/charakdtrepo 42b0ead02634	nov20	In use	18 days ago	6.48 GB	▶ ⋮ 🗑️
<input type="checkbox"/>	504568707713.dkr.ecr.ap-south-1.amazonaws.com/charakdtrepo c956b0c9694a	latest	In use	2 months ago	6.48 GB	▶ ⋮ 🗑️

Showing 3 items

RAM 1.88 GB CPU 0.05% Not signed in v4.23.0 2

Docker Desktop - Images

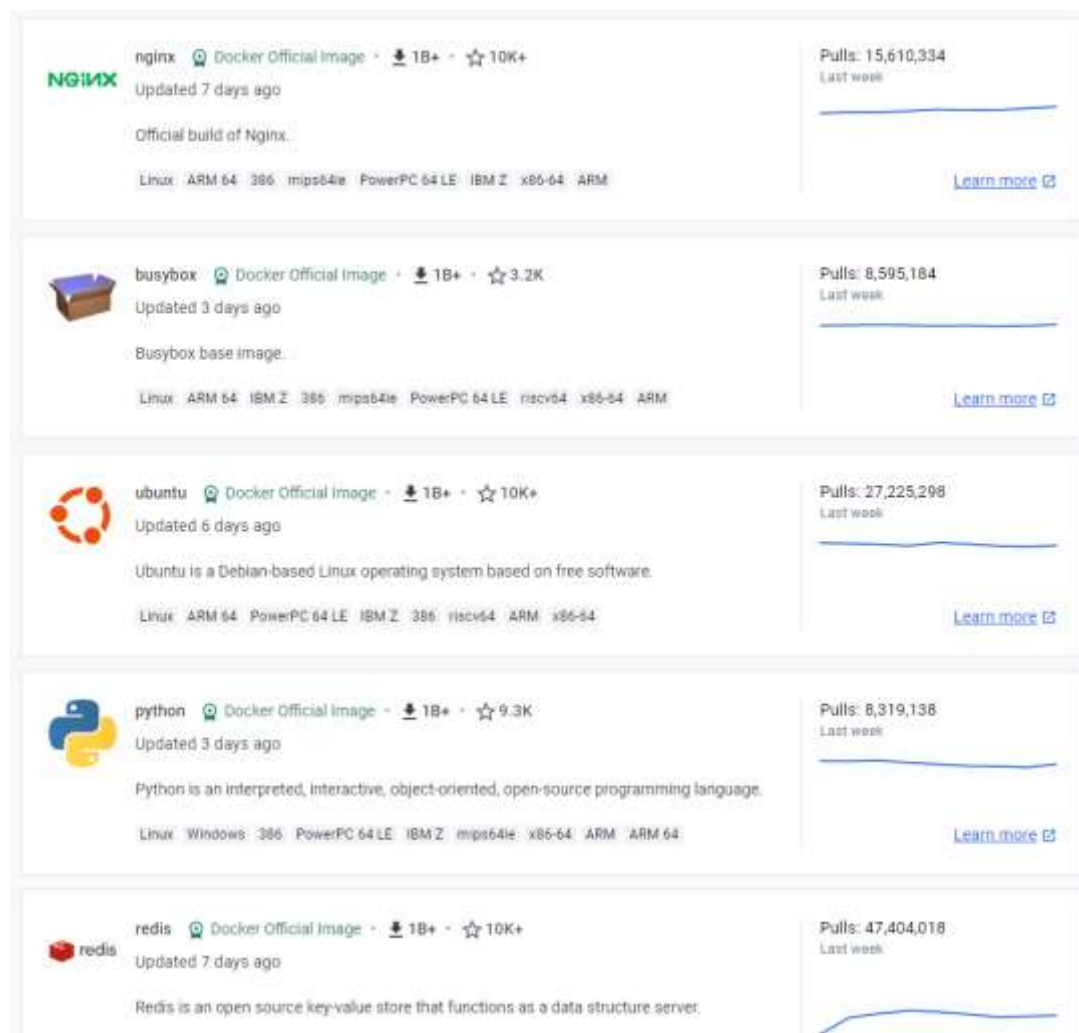


Image Name	Stars	Pulls (Last Week)	Architecture
nginx	10K+	15,610,334	Linux, ARM 64, 386, mips64le, PowerPC 64 LE, IBM Z, x86-64, ARM
busybox	3.2K	8,595,184	Linux, ARM 64, IBM Z, 386, mips64le, PowerPC 64 LE, riscv64, x86-64, ARM
ubuntu	10K+	27,225,298	Linux, ARM 64, PowerPC 64 LE, IBM Z, 386, riscv64, ARM, x86-64
python	9.3K	8,319,138	Linux, Windows, 386, PowerPC 64 LE, IBM Z, mips64le, x86-64, ARM, ARM 64
redis	10K+	47,404,018	Linux, ARM 64, 386, mips64le, PowerPC 64 LE, IBM Z, x86-64, ARM

Amazon Machine Images

innovate

achieve

lead

1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Add Tags
6. Configure Security Group
7. Review

Step 1: Choose an Amazon Machine Image (AMI)

[Cancel and Exit](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace, or you can select one of your own AMIs.

Quick Start

My AMIs

AWS Marketplace

Community AMIs

Operating system

☐ Amazon Linux

☐ Cent OS

☐ Debian

☐ Fedora

☐ Gentoo

☐ OpenSUSE

☐ Other Linux

☐ Red Hat

☐ SUSE Linux

☒ Ubuntu

☐ Windows

Architecture

☐ 32-bit


☐ 64-bit

Root device type

☐ ebs

ami-274dc831

1 to 50 of 29,403 AMIs




ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20170221 - ami-a58d0dc5

Root device type: ebs Virtualization type: hvm

64-bit

Select




ubuntu/images/hvm-ssd/ubuntu-trusty-14.04-amd64-server-20170110 - ami-5e63d13e

Root device type: ebs Virtualization type: hvm

64-bit

Select




ubuntu/images-testing/ebs-ssd/ubuntu-zesty-daily-amd64-server-20161108 - ami-0104a461

Canonical, Ubuntu, None, UNSUPPORTED daily amd64 zesty image build on 2016-11-08

Root device type: ebs Virtualization type: paravirtual

64-bit

Select




ubuntu/images-testing-dev/hvm-ssd/ubuntu-xenial-16.04-testing-amd64-server-20170118 - ami-013b8761

Canonical, Ubuntu, 16.04 LTS, amd64 xenial image build on 2017-01-18

Root device type: ebs Virtualization type: hvm

64-bit

Select




ubuntu/images-testing/ebs-ssd/ubuntu-zesty-daily-amd64-server-20161103 - ami-019e3f61

Canonical, Ubuntu, None, UNSUPPORTED daily amd64 zesty image build on 2016-11-03

Root device type: ebs Virtualization type: paravirtual

64-bit

Select



ubuntu-15.04-Snappy-core-edge-15.04-20150922.1552 - ami-01cad331

ubuntu-15.04-Snappy-core-edge-15.04-20150922.1552

64-bit

Select

Yet another analogy with a difference



Docker Image



Docker Container



Docker - Commands - Images

- **Build an Image from a Dockerfile**
docker build -t <image_name>
- **Build an Image from a Dockerfile without the cache**
docker build -t <image_name> . --no-cache
- **List local images**
docker images
- **Delete an Image**
docker rmi <image_name>
- **Remove all unused images**
docker image prune



Docker - Commands - Containers

- **Create and run a container from an image, with a custom name:**
`docker run --name <container_name> <image_name>`
- **Run a container with and publish a container's port(s) to the host.**
`docker run -p <host_port>:<container_port> <image_name>`
- **Run a container in the background**
`docker run -d <image_name>`
- **Start or stop an existing container:**
`docker start|stop <container_name> (or <container-id>)`
- **Remove a stopped container:**
`docker rm <container_name>`
- **Open a shell inside a running container:**
`docker exec -it <container_name> sh`
- **Fetch and follow the logs of a container:**
`docker logs -f <container_name>`
- **To inspect a running container:**
`docker inspect <container_name> (or <container_id>)`
- **To list currently running containers:**
`docker ps`
- **List all docker containers (running and stopped):**
`docker ps --all`
- **View resource usage stats**
`docker container stats`

Commands - Docker Hub

- **Login into Docker**
docker login -u <username>
- **Publish an image to Docker Hub**
docker push <username>/<image_name>
- **Search Hub for an image**
docker search <image_name>
- **Pull an image from a Docker Hub**
docker pull <image_name>



Docker - General Commands

- **Start the docker daemon**
docker -d
- **Get help on Docker. Can also use –help on all subcommands**
docker --help
- **Display system-wide information**
docker info

**IS DOCKER A
ONE-SIZE-FITS-ALL
SOLUTION?**





When to Use Docker?

- Development team changes often
- Software runs in different environment
- Software grows and consists of many bits and pieces
- Expecting the software to be scalable and handle more users
- Hosting may change in future and if don't want to be locked
- Test new technologies for software

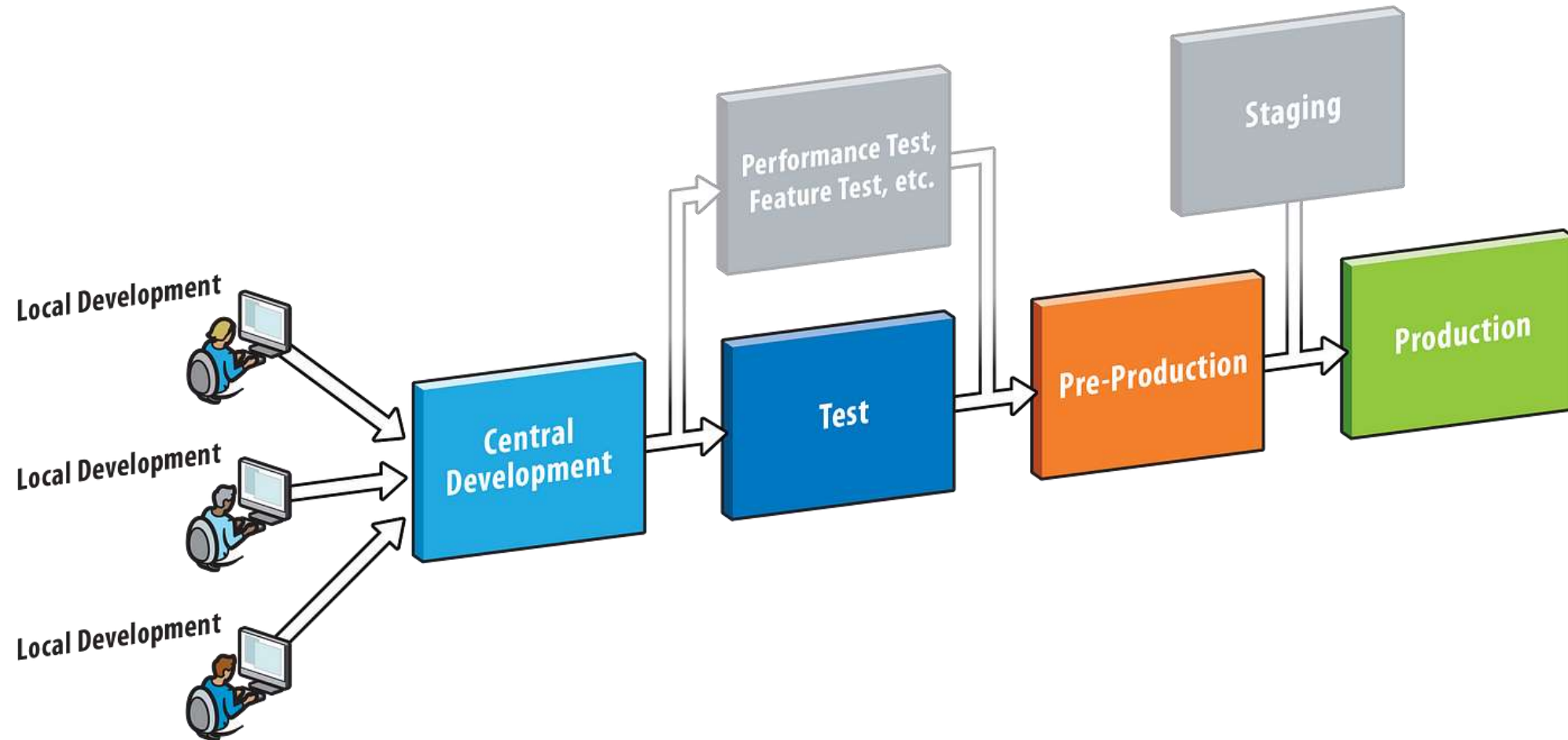


Development team changes often

- New developers need to setup their local development environment for the project - e.g. local server, database, or third-party libraries etc.
- It may take hours to many days, depending on complexity and how well is the project setup manually.
- Docker automates this by simply running one command that will prepare the development environment.



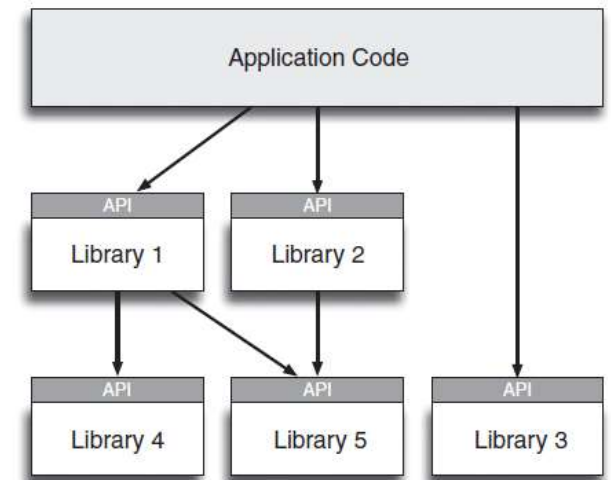
Software Runs in Different Env





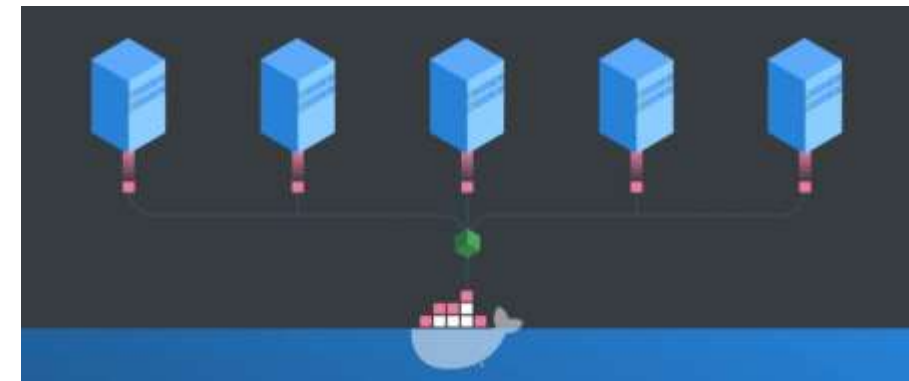
Software Consists of Many Parts

- Developers add new libraries, services, and other dependencies to the software daily.
- All the changes in project setup have to be communicated properly.
- With Docker, all requirements are there in configuration file and if something is specified to this configuration, it is automatically added for every developer working on the project.



Need Software to be Scalable

- Docker containers can be launched in many copies that may run in parallel.
- Docker compose can be used for this.



Docker Compose

```
FROM node:alpine
WORKDIR /backend/
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD [ "node", "app.js" ]
```

```
docker build -t backend .
```

```
docker run -p 3000:3000 backend
```

Backend Image

```
# Use an official Node.js runtime as the base image
FROM node:alpine
RUN npm install -g http-server
# Set the working directory in the container
WORKDIR /frontend
COPY package*.json ./
# Install the application dependencies
RUN npm install
# Copy the entire application to the container
COPY . .
RUN npm run build
EXPOSE 8080
CMD [ "http-server", "dist" ]
```

```
docker build -t frontend .
```

Frontend Image

```
version: "3"
services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - 3000:3000
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - 8080:8080
```

```
docker compose build
```

Docker Compose

Hosting may Change in Future

- When a business grows and evolves, so do the server requirements. Also, in fast paced business environment, **web infrastructure** needs to be **flexible** enough to **adapt quickly**.





Test New Technologies for Software

- Docker is quite useful if you want to experiment with different technologies. Want to adopt some new database, programming language, or other tools? Most probably there is **already a Docker template** for this (called Docker Image) created by a **docker community**.
- It is especially easy if you use one of the most popular Docker tools - **docker compose**. It brings all the project infrastructure to just a single file and run docker containers for you. No need for time-consuming installation or debugging compatibility issues.



When not to Use Docker?

- Software product is a desktop application
- Software project is relatively small and simple
- Team consists of only one developer
- Lack of skilled resources for Docker
- If want to speed up the application
- If security is on priority



Need to Speedup the Application

- Docker may speed up your development process significantly, but not necessarily your app itself.
- Although it helps with making your application scalable, so more users will be able to use it, the single instance of your app will usually be just a hint slower than without Docker.

If Security is Priority



- While isolated processes in containers promise improved security, all containers share access to a single host operating system. You risk running Docker containers with incomplete isolation. Any malicious code can get access to your computer memory.

