K21Academy
Learn Cloud from Experts

Docker & Certified Kubernetes Administrator (CKA)

Concepts & Architecture | Storage | Networking | HA & Clusters | Scheduling
Administration | Docker Compose | Security | Troubleshooting

16 Module | 60+ Lessons | 39 Hands-On Labs | Exam Preparation | On-Job Support

Atul Kumar
Oracle ACE & Cloud Expert

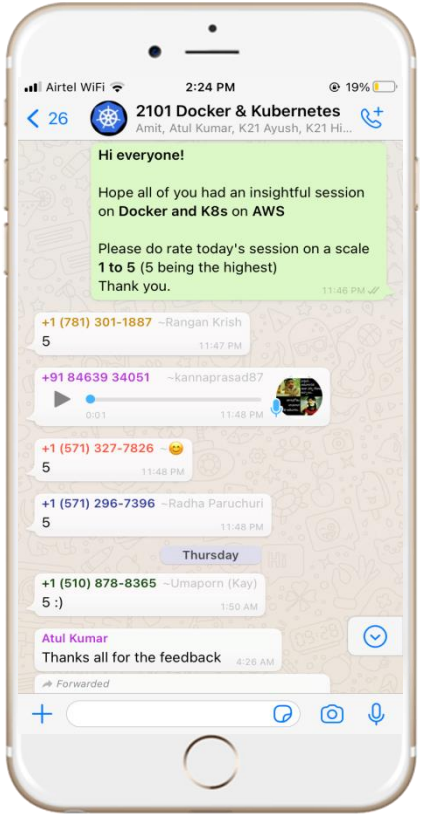
docker + kubernetes

WhatsApp & Ticketing System

K21Academy
Learn Cloud from Experts

support@k21academy.com

zendesk

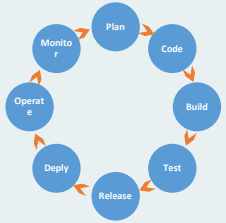


Agenda: Module

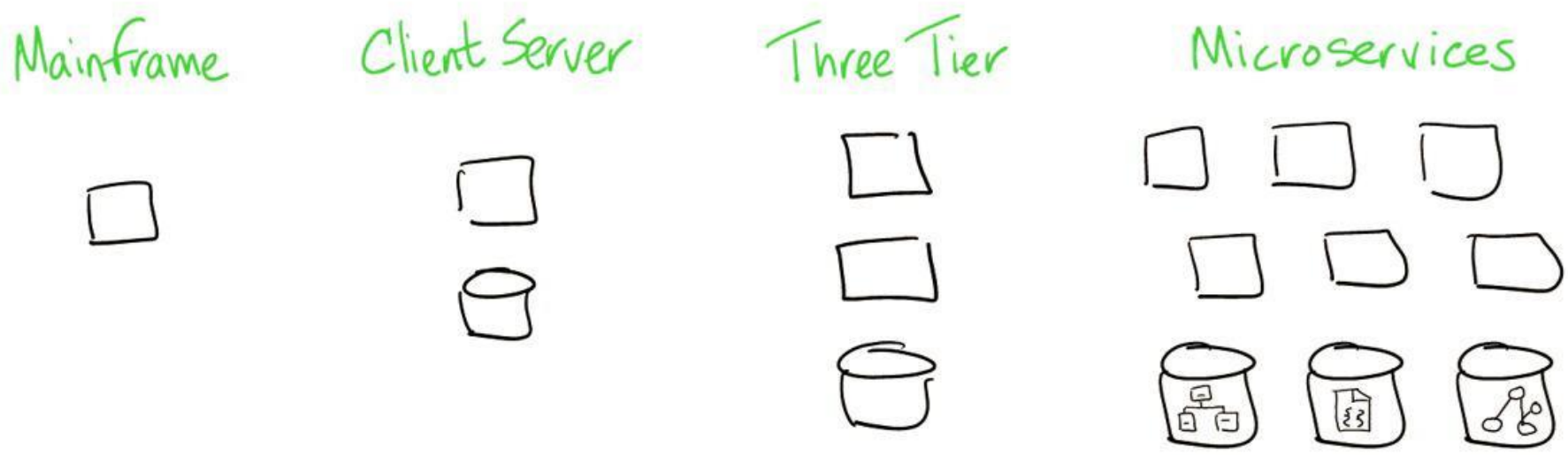
- Monolithic application Overview
- Disadvantage of Monolithic
- Monolithic Architecture
- Microservices Overview
- Microservices Architecture

Evolution of Development & Deployment

	Development Process	Application Architecture	Deployment and Packaging	Application Infrastructure
~ 1980 ~ 1990	Waterfall	Monolithic	Physical Server	Datacenter
~ 2000	Agile	N-Tier	Virtual Servers	Hosted
~ 2010 Now	DevOps	Microservices	Containers	Cloud



Evolution of Development & Deployment



docker

+

kubernetes

K21Academy
Learn Cloud from Experts

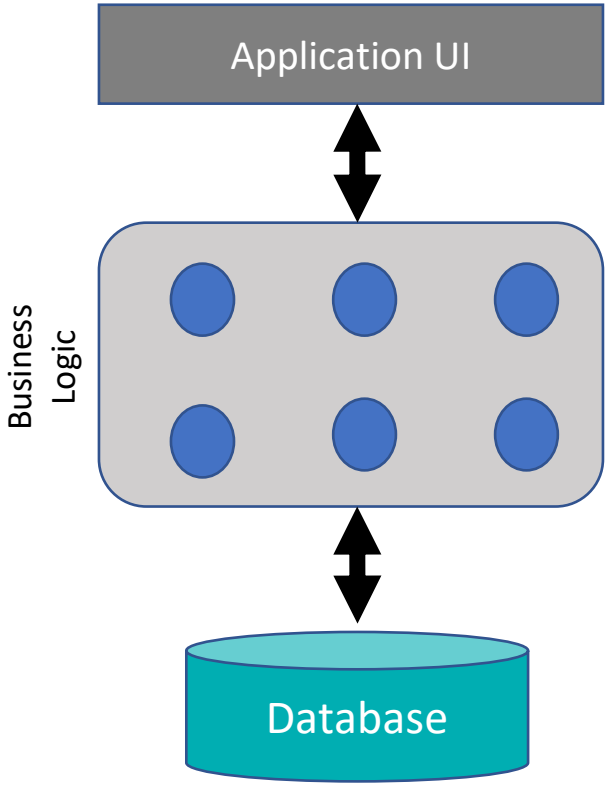
Monolithic Overview

Monolithic

K21Academy
Learn Cloud from Experts

- A monolithic application has all of its components residing together as one unit
- A web application is a software program running on a web server
- An application consists of three main components:
  - user interface (UI)
  - Database
  - Server
- All three of these components and is written and released as a single unit

Monolithic Architecture



Solution !!!

- Define an architecture that structures the application as a set of loosely coupled, collaborating services
- Each service should be:
  - Highly maintainable and testable
  - Loosely coupled with other services
  - Independently deployable
  - Capable of being developed by a small team

K21Academy
Learn Cloud from Experts

docker
+
kubernetes

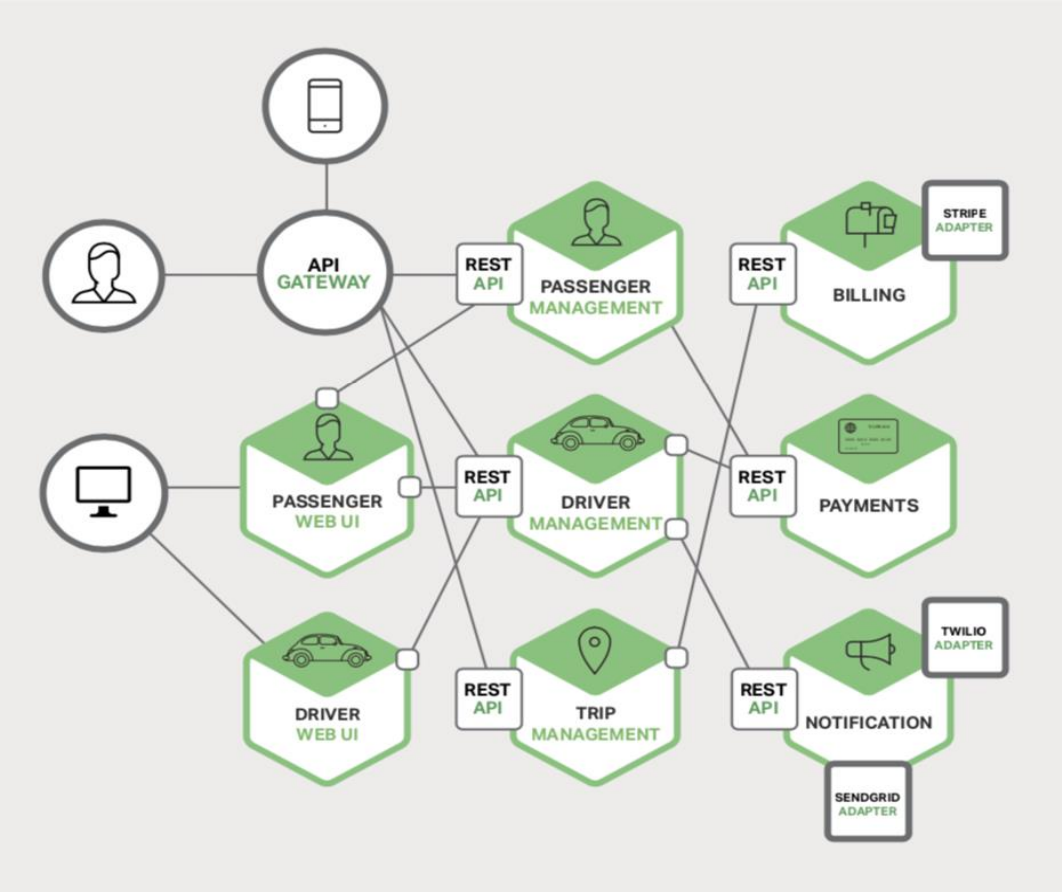
Microservices Overview

Microservices

K21Academy
Learn Cloud from Experts

- Microservices is a architectural style that structure an application as a collection of various service
- Split the application into set of smaller, interconnected services
- A service typically implements a set of distinct features or functionality, such as order management, customer management
- Each microservice is a mini-application that has its own hexagonal architecture consisting of business logic along with various adapters

Microservices Architecture



K21Academy
Learn Cloud from Experts

docker

+

kubernetes

Docker

Agenda: Module

K21Academy
Learn Cloud from Experts

- Docker Basics Concepts
- Docker Architecture
- Docker Images
- Docker Networking
- Docker Storage
- Automate Image Creation – Dockerfile
- Docker Host -Networks and volumes
- Compose tool
- Docker Cluster
- Hands-On Guides

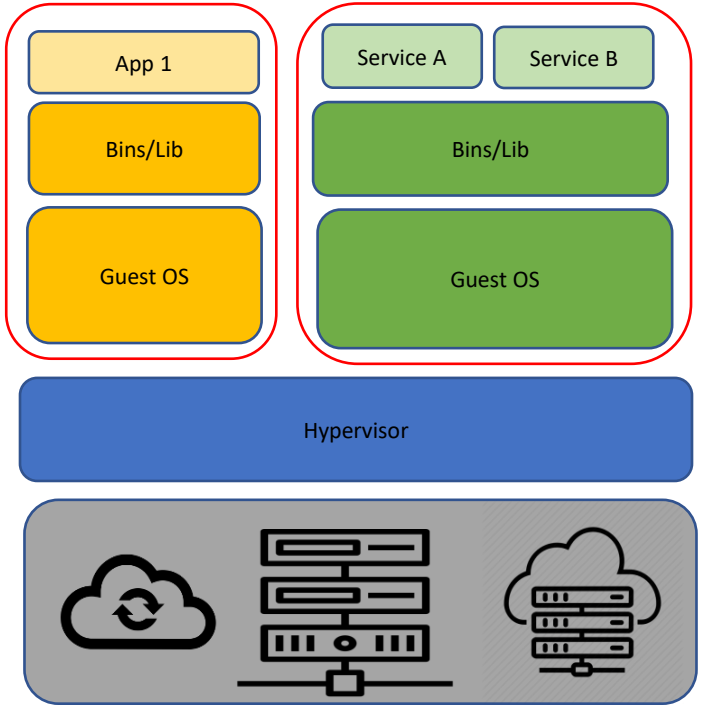
K21Academy
Learn Cloud from Experts

docker
+
kubernetes

Introduction to Containers

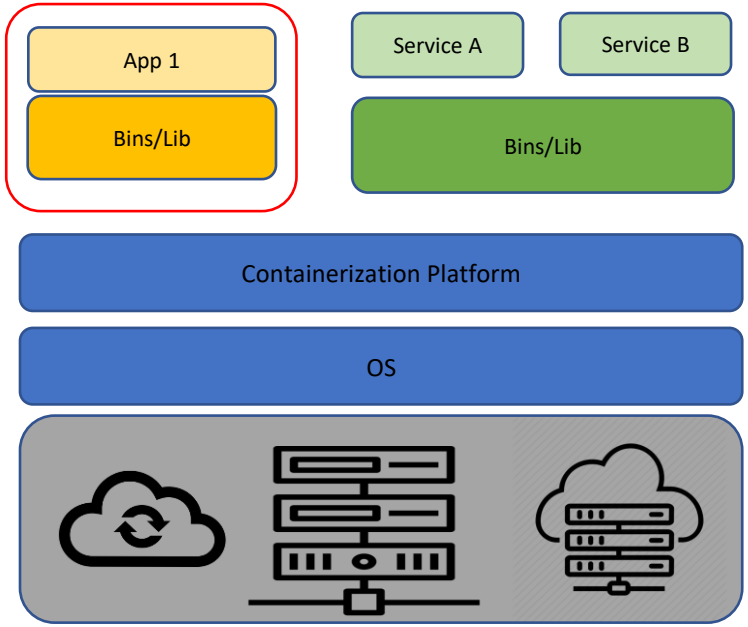
Virtual Machine

- Virtualization is hardware-level based
- Dedicated Operating System for each VM
- Startup takes a few minutes
- Packaging is bulky

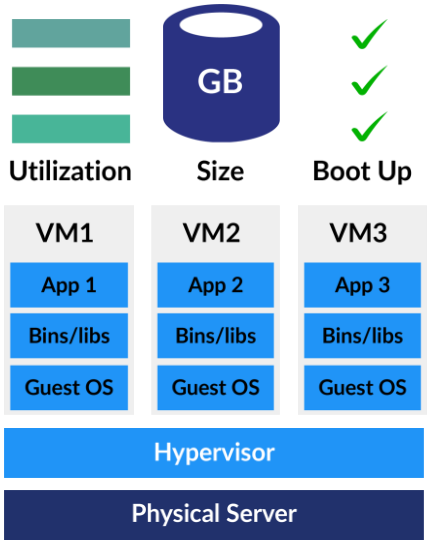


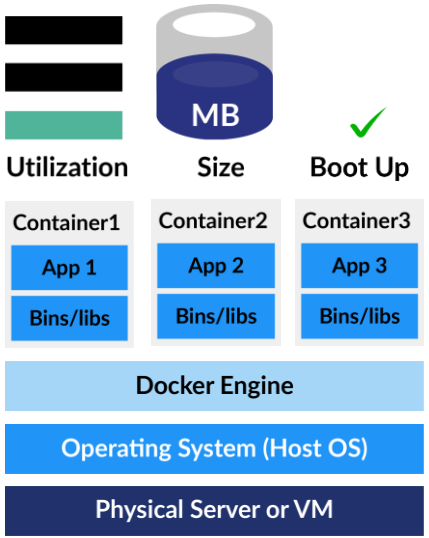
Containers

- Host OS is shared by all the containers
- Application is decoupled from the OS
- Startup takes a few milliseconds
- Packing is not bulky



VM vs Docker





Containers

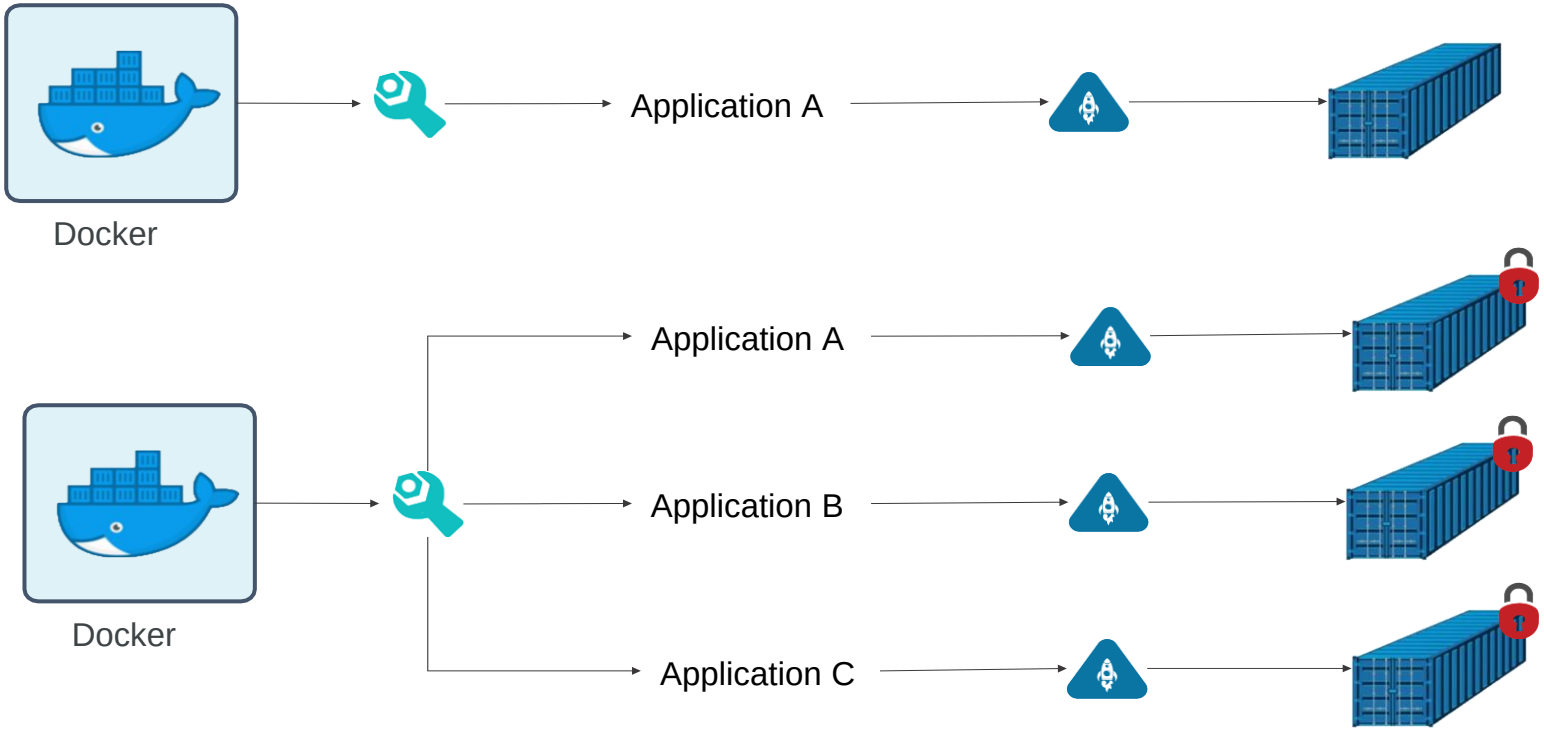
- Containers are lightweight because they don't need the extra load of a hypervisor
- Run directly within the host machine's kernel
- Run Docker containers within virtual machines
- Develop your application and its supporting components using containers
- Container is the basic unit for distributing and testing your application

K21Academy
Learn Cloud from Experts

docker
+
kubernetes

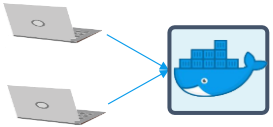
Docker Overview

What Is Docker?



Features of Docker

Application isolation and security management

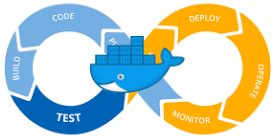


Multitenancy

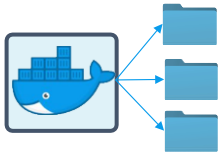
Simplified configuration

Features

Rapid deployment

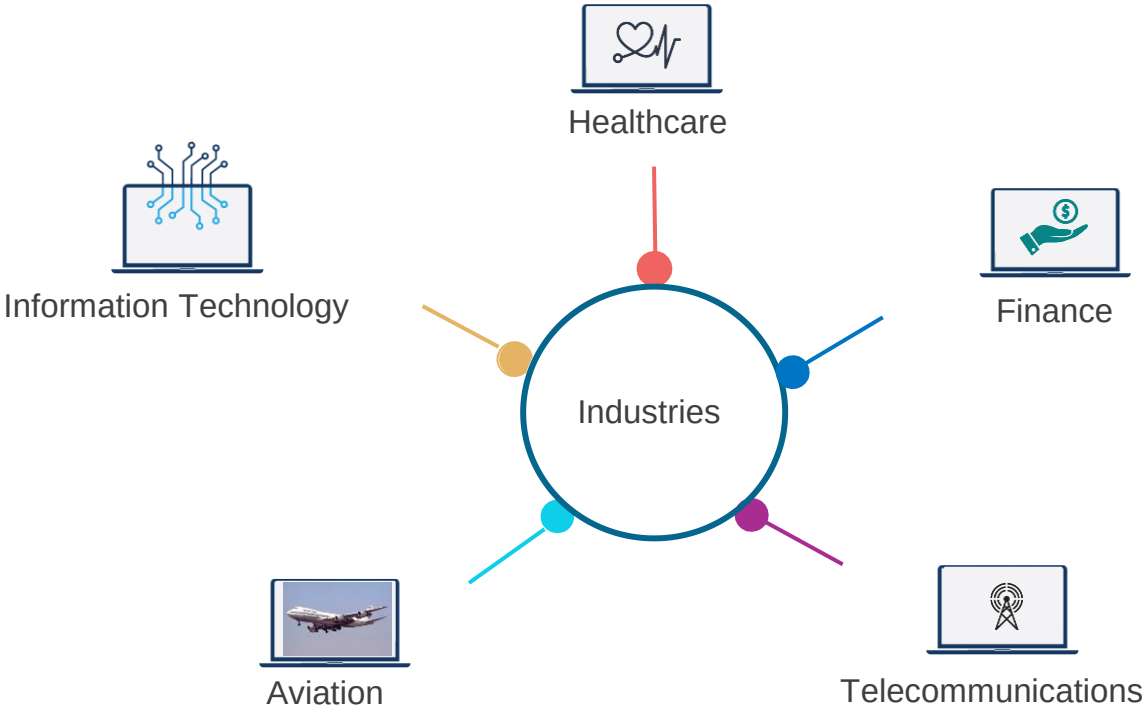


Container CI/CD



Increased productivity

Industries Using Docker



docker

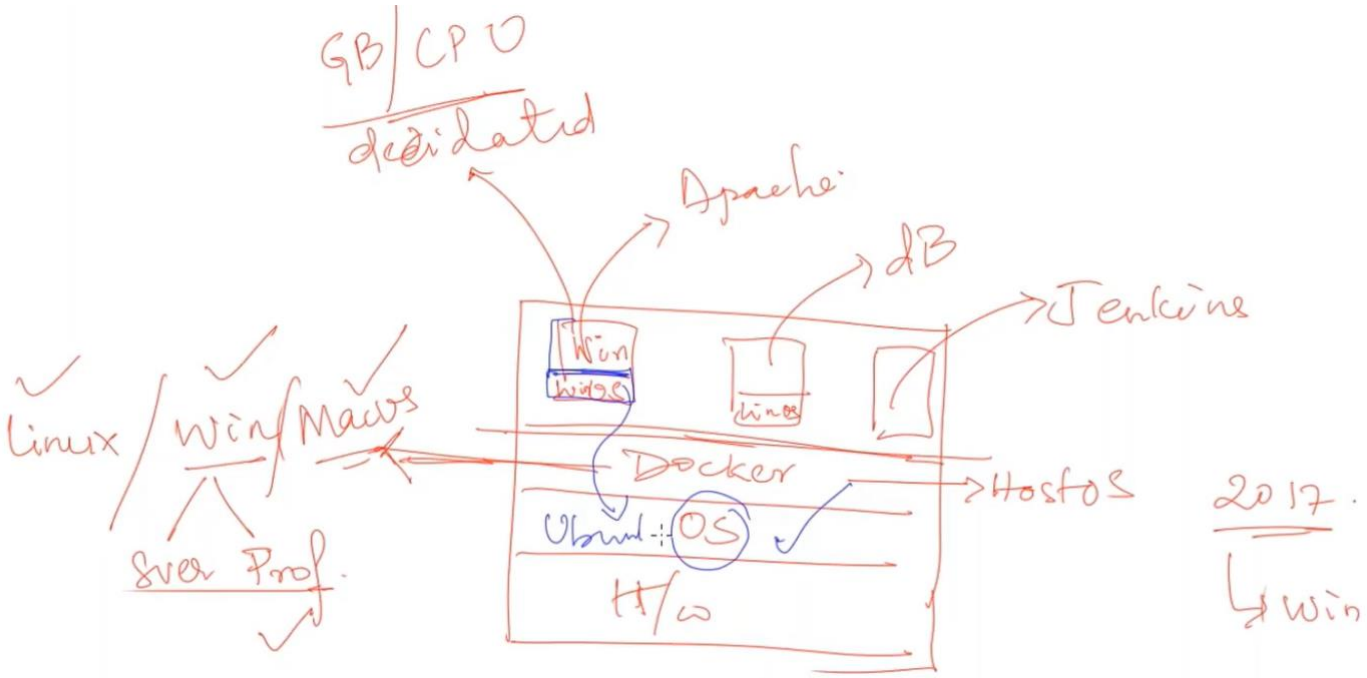
+

kubernetes

K21Academy
Learn Cloud from Experts

Installing Docker

Docker Install Options

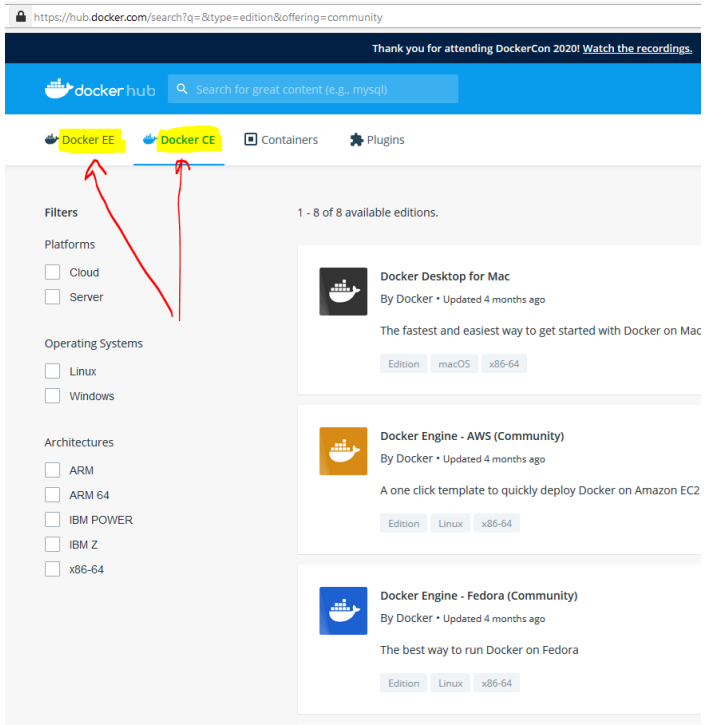


Docker Installation Options

- Docker Edition
  - Community Edition (CE): FREE
  - Enterprise Edition (EE): Paid

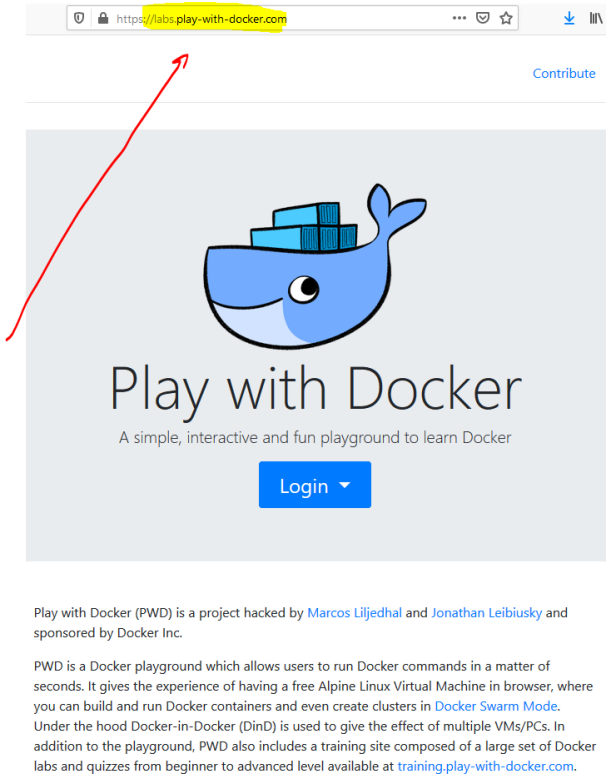
- Install Location
  - On-Premise (Data Centre)
  - Cloud (Azure, Oracle, AWS, Google..)
  - Laptop/Desktop

- Supported O.S.
  - Linux
  - Windows
  - Mac



Docker For Practice

- Docker Desktop
  - Windows
  - Mac
- Server Install
  - Linux
  - Windows Server
- Sandbox
  - labs.play-with-docker.com



Docker Install: Linux

1. First Update Software Repositories

```
$ sudo apt-get update -y
```

2. Uninstall Old Versions of Docker (***Optional:*** *Only if docker was already installed on this host and you want to configure it again*)

```
$ sudo apt-get remove docker docker-engine docker.io
```

3. Install Docker

```
$ sudo apt install docker.io
```

4. Start and Enable Docker

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

5. Check Docker status

```
$ sudo systemctl status docker
```

6. Identify the user id of user that will run container

```
$ id
```

```
DockerMachineUser@DockerMachine:~$ id
uid=1000(DockerMachineUser) gid=1000(DockerMachineUser) groups=1000(DockerMachin
eUser),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(vid
eo),46(plugdev),108(lxd),114(netdev)
DockerMachineUser@DockerMachine:~$
```

7. Add above user to docker group

```
$ sudo usermod -a -G docker <userid>

$ sudo usermod -aG docker DockerMachineUser
```

8. Logout and check if user has group docker assigned

9. Check Docker Version

```
docker version
```

```
DockerMachineUser@DockerMachine:~$ docker version
Client:
 Version:           19.03.6
 API version:       1.40
 Go version:        go1.12.17
 Git commit:        369ce74a3c
 Built:             Fri Feb 28 23:45:43 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          19.03.6
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.17
  Git commit:       369ce74a3c
  Built:            Wed Feb 19 01:06:16 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.3.3-0ubuntu1~18.04.2
  GitCommit:
 runc:
  Version:          spec: 1.0.1-dev
  GitCommit:
 docker-init:
  Version:          0.18.0
  GitCommit:
DockerMachineUser@DockerMachine:~$
```

K21Academy
Learn Cloud from Experts

docker

+

kubernetes

Step-by-Step Activity Guide

Lab Exercise: On Portal

K21Academy
Learn Cloud from Experts

3 Module 1: [Lab] Docker Installation, Containers & Images

- Lesson 1: Docker Installation Overview Options: Windows/Mac/Linu min)
- Lesson 2: Register Free Azure Trial Account (18:49 min)
- Lesson 3: Create and Configure Ubuntu Machine on Azure Cloud (1
- Lesson 4: Connect Azure Ubuntu Machine From MAC (03:54 min)
- Lesson 5: Install and Configure Docker o
- Lesson 6: Docker Container Images (32:
- Activity Guide (Lab): Register For Azure Cloud Account & Accessing
- Activity Guide (Lab): Create & Connect To Ubuntu 18.04 Server On Account
- Activity Guide (Lab): Create & Manage Docker on Ubuntu
- Activity Guide (Optional) : Register for AWS Free Tier Account And AWS Console
- Activity Guide (Optional): Creating & Connect to Ubuntu EC2 Instar
- Docker Notes by Trainer Ramesh

Activity Guide for Create Cloud Account and ubuntu machine on Azure

3 Module 1: [Lab] Docker Installation, Containers & Images

- Lesson 1: Docker Installation Overview Options: Windows/Mac/L min)
- Lesson 2: Register Free Azure Trial Account (18:49 min)
- Lesson 3: Create and Configure Ubuntu Machine on Azure Cloud
- Lesson 4: Connect Azure Ubuntu Machine From MAC (03:54 min
- Lesson 5: Install and Configure Docker on Ubuntu Machine (16:
- Lesson 6: Docker Container Images (32:20 min)
- Activity Guide (Lab): Register For Azure Cloud Account & Access
- Activity Guide (Lab): Create & Connect Account
- Activity Guide (Lab): Create & Manage
- Activity Guide (Optional) : Register for AWS Free Tier Account A AWS Console
- Activity Guide (Optional): Creating & Connect to Ubuntu EC2 Ins
- Docker Notes by Trainer Ramesh

Activity Guide for Create Cloud Account and ubuntu machine on AWS

3 Module 1: [Lab] Docker Installation, Containers & Images ✓

- Lesson 1: Docker Installation Overview Options: Windows/Mac/Linux (19:56 min)
- Lesson 2: Register Free Azure Trial Account (18:49 min)
- Lesson 3: Create and Configure Ubuntu Machine on Azure Cloud (19:27 min)
- Lesson 4: Connect Azure Ubuntu Machine From MAC (03:54 min)
- Lesson 5: Install and Configure Docker on Ubuntu Machine (16:04 min)
- Lesson 6: Docker Container Images (32:20 min)
- Activity Guide (Lab): Register For Azure Console
- Activity Guide (Lab): Create & Connect To Azure Cloud Account
- Activity Guide (Lab): Create & Manage Docker on Ubuntu
- Activity Guide (Optional) : Register for AWS Free Tier Account And Login to AWS Console
- Activity Guide (Optional): Creating & Connect to Ubuntu EC2 Instance
- Docker Notes by Trainer Ramesh

Activity Guide for Docker

Lab Exercise: Create Azure Cloud Account

- Follow Activity Guide to
  - Create Azure Cloud Account
  - Access Azure Console

K21Academy
Learn Oracle from Experts

ORACLE Gold Partner

Lab Exercise: Create & Access Machine

K21Academy
Learn Cloud from Experts

- Follow Activity Guide to
  - Create VM On Azure Portal
  - Connect to Ubuntu Machine
  - Start/Stop or Terminate Ubuntu Machine

Contents

Lab Exercise: Create & Manage Docker

K21Academy
Learn Cloud from Experts

Follow Activity Guide to :

- Docker Installation Steps on Ubuntu
- Working with Container
- Working with Docker Images
- Docker Default Bridge Networking
- Creating Custom Bridge Network
- Docker Host Network
- Docker Storage – Host Path Mounting
- Docker Volume
- Configuring External DNS, Logging and Storage Driver
- Working with Dockerfile
- Working with Application stack

K21Academy
Learn Cloud from Experts

ORACLE Gold Partner

Contents

K21Academy
Learn Cloud from Experts

docker

+

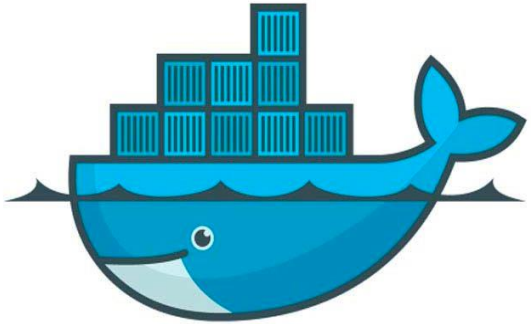
kubernetes

Introduction to Docker

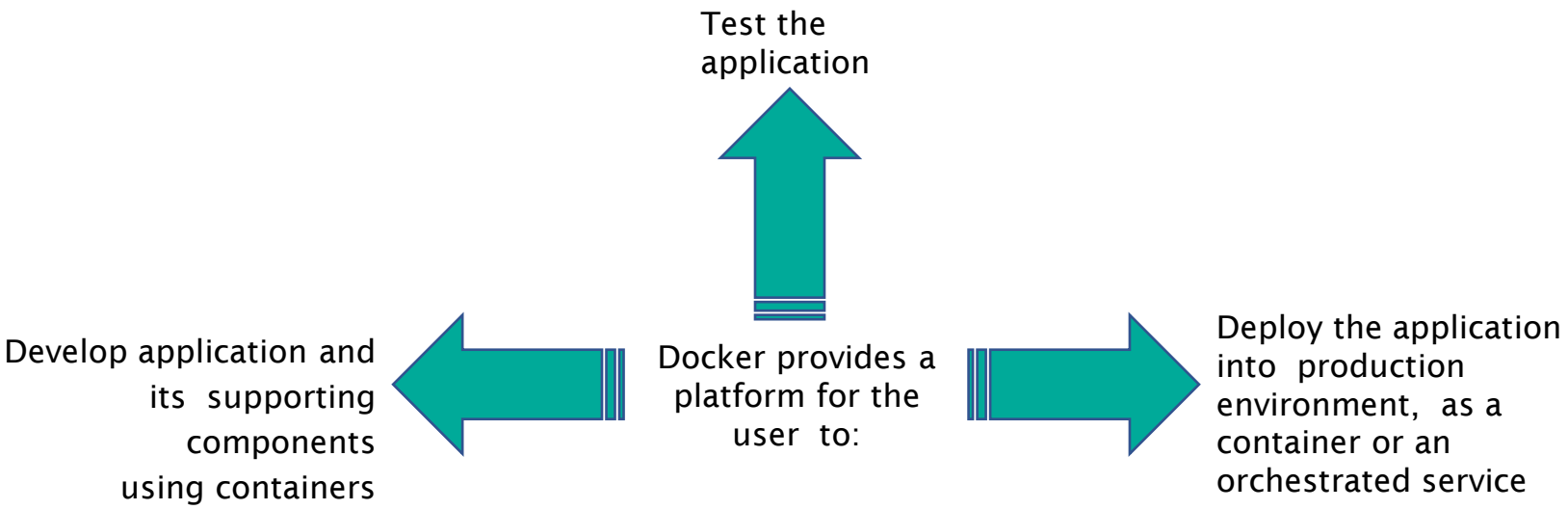
Introduction to Docker

K21Academy
Learn Cloud from Experts

- Docker is a platform for developers and sysadmins to develop, ship, and run applications by using containers
- Docker helps the user to quickly assemble applications from its components and eliminates the friction during code shipping
- Docker aids the user to test and deploy the code into production



Docker Functionalities



Underlying Technology

K21Academy
Learn Cloud from Experts

- **Control groups** limits an application to a specific set of resources
- Allow Docker Engine to share available hardware resources to containers
- Enforce limits and constraints
- **Union file systems** file systems that operate by creating layers, making it very lightweight and fast
- **Container Format**
- Docker Engine combines the namespaces, control groups and UnionFS into a wrapper called a container format
- The default container format is libcontainer

Underlying Technology

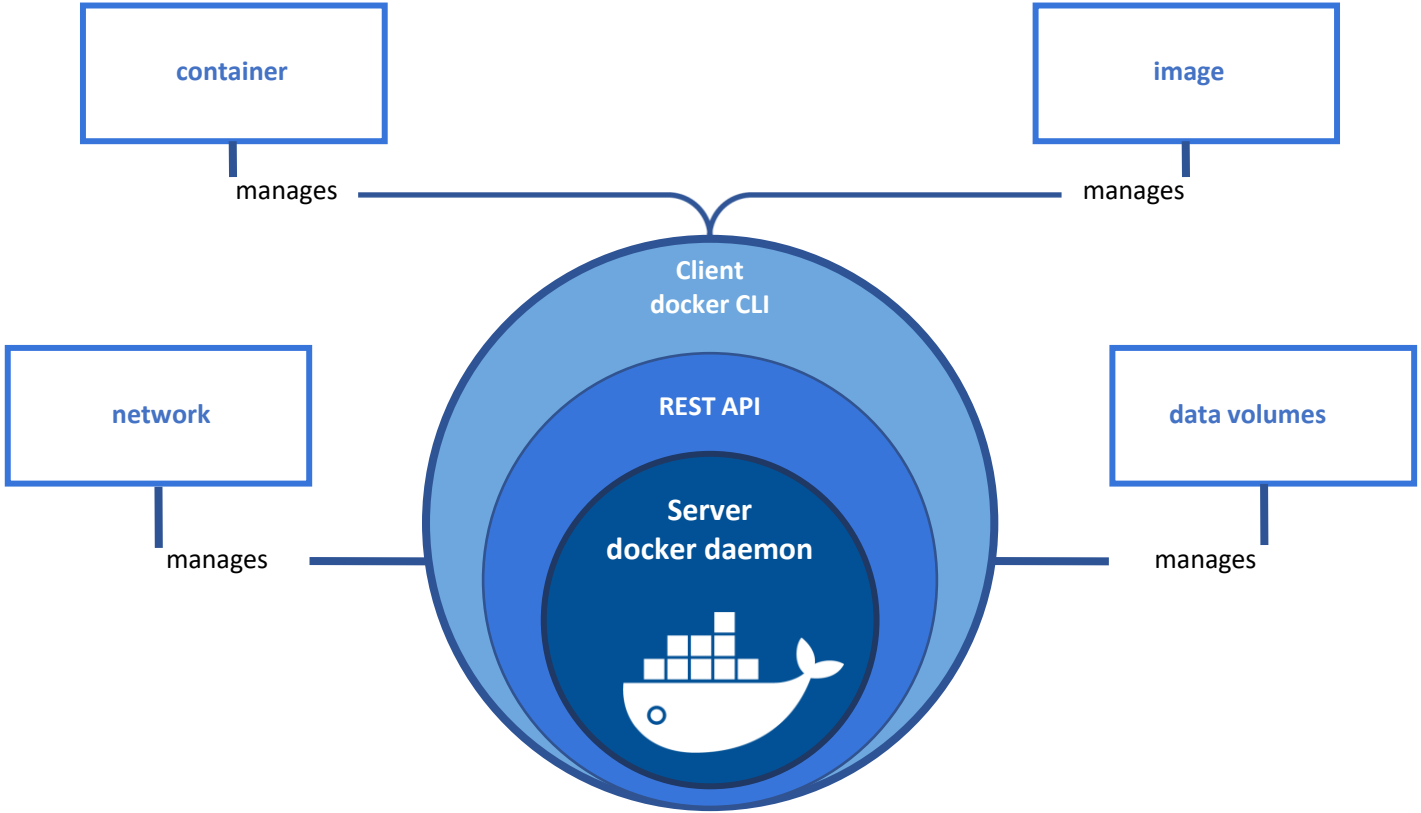
K21Academy
Learn Cloud from Experts

- **Namespaces** provides a layer of isolation
- Provide isolated workspace called the container
- Docker creates a set of namespaces for the container
- Docker Engine uses namespaces such as
  - The pid namespace: Process isolation (PID: Process ID).
  - The net namespace: Managing network interfaces (NET: Networking).
  - The ipc namespace: Managing access to IPC resources (IPC: InterProcess Communication).
  - The mnt namespace: Managing filesystem mount points (MNT: Mount).
  - The uts namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System)

Docker Use Cases

- Fast, consistent delivery of your applications
- Responsive deployment and scaling
- Running more workloads on the same hardware

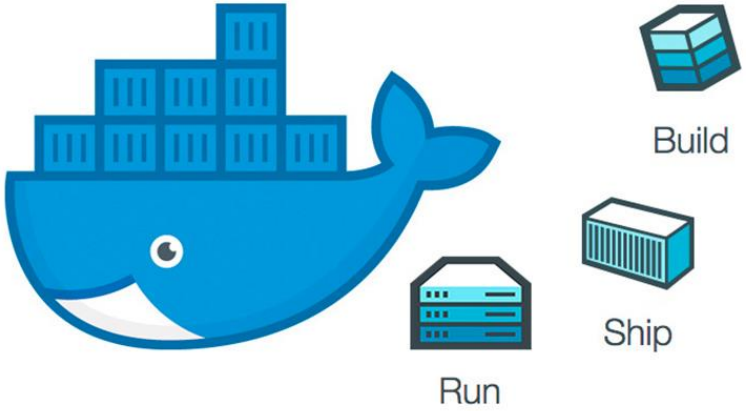
Docker Engine



Docker Benefits

Provides fast delivery of the applications

Is deployable and scalable



Has high density and runs more workloads

Aids in quick deployment for easy management

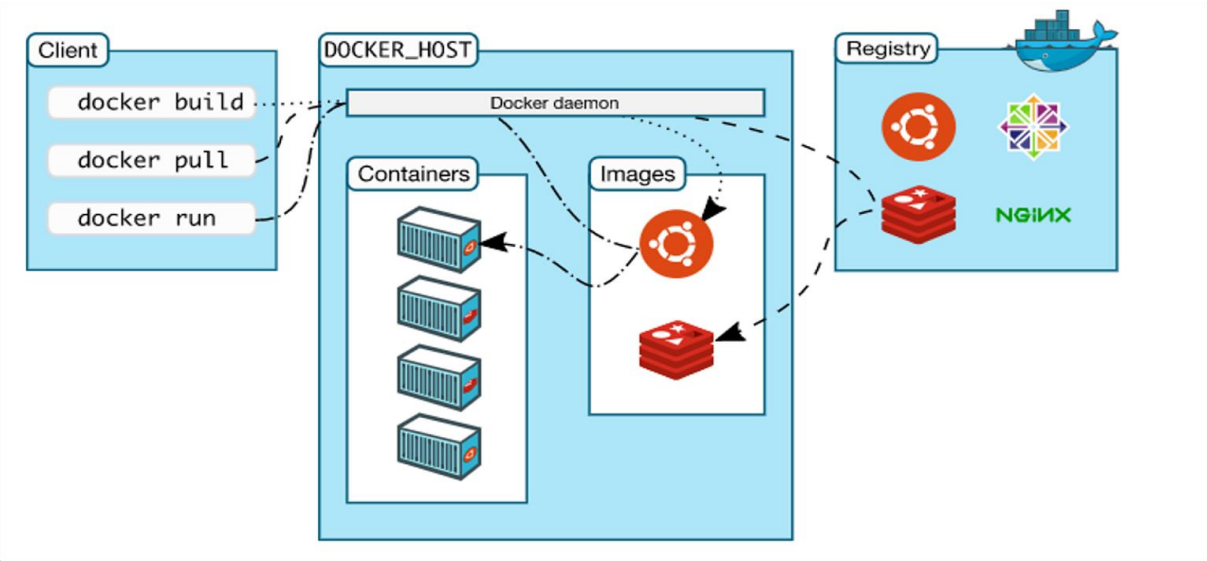
K21Academy
Learn Cloud from Experts

docker
+
kubernetes

Docker Architecture

Docker Architecture

- Docker uses a client-server architecture
- The docker client interacts with the Docker daemon that performs running, heavy lifting of building, and distribution of Docker containers



Docker Architecture

K21Academy
Learn Cloud from Experts

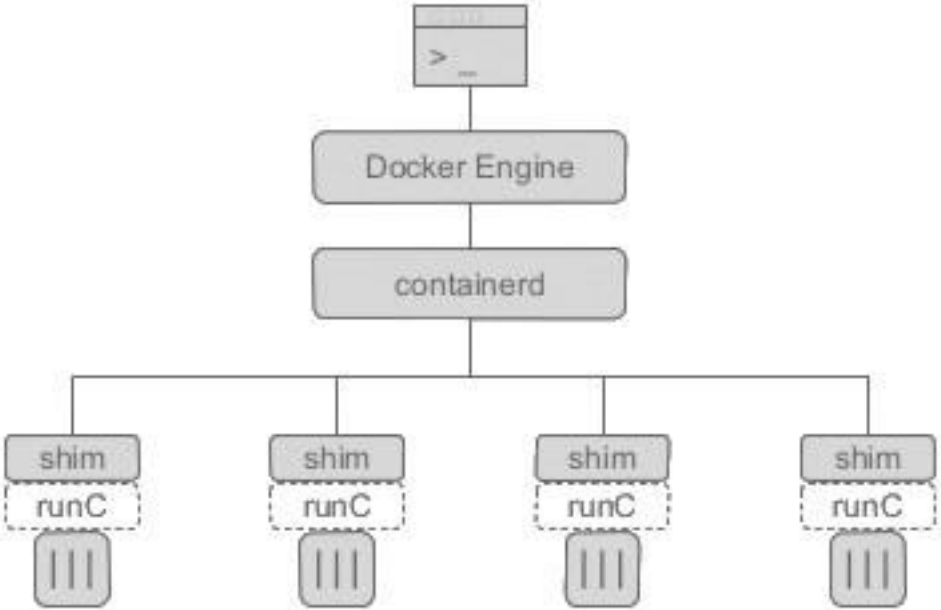
- **Docker daemon**

- Docker uses a client-server architecture
- The docker client interacts with the Docker daemon that performs running, heavy lifting of building, and distribution of Docker containers

- **Docker client**

- It is the primary path for Docker users to interact with the Docker application
- the client sends these commands to dockerd

Docker Dissection



Docker Daemon Architecture

K21Academy
Learn Cloud from Experts

- **dockerd**: Listen for Docker Engine API requests
- **containerd**:
  - Introduced in Docker 1.11
  - Responsibilty of managing containers life-cycle
  - containerd is the executor for containers
  - Executing of Containers by calling **runc** with the right parameters to run containers
- **runc**:
  - containerd uses runc to do all the Linux work
  - Starts the container and exits
- **containerd-shim:**
  - It allows you to run daemonless containers
  - STDIO and other FDs are kept open in the event that containerd
  - Reports the containers exit status to containerd.

Questions

K21Academy
Learn Cloud from Experts

docker

+

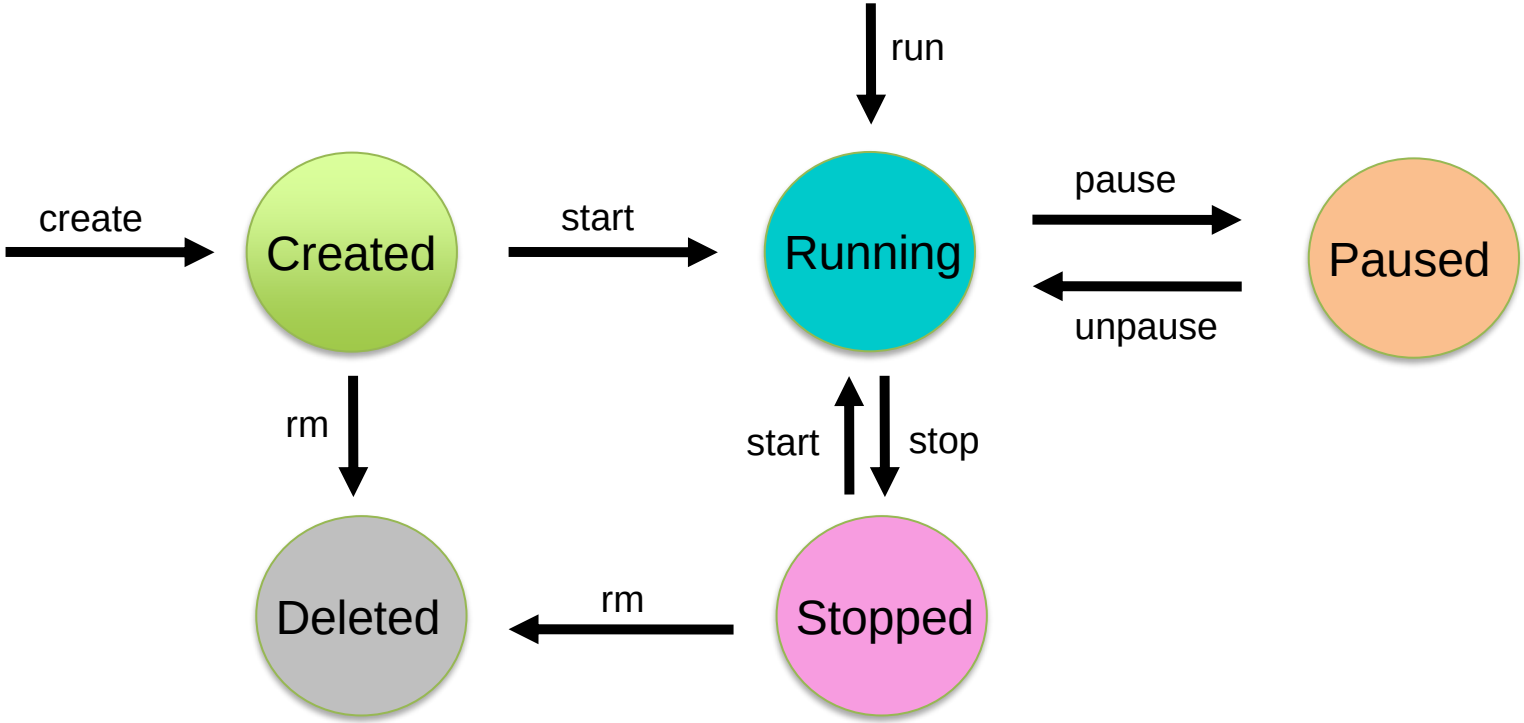
kubernetes

Working with Docker

Docker Commands

- docker –version
- docker info
- docker log path : /var/lib/docker
- docker daemon setting: /etc/docker

Container Lifecycle



Docker Container Commands

K21Academy
Learn Cloud from Experts

- docker run: Create Container
  - -it: interactive
  - **docker run -it ubuntu bash**
  - -dt: detached
  - **docker run -dt ubuntu bash**

- docker ps: List all running containers
- docker ps –a: Lists all containers running/stopped/paused

Process View of Container

K21Academy
Learn Cloud from Experts

- docker ps

```
ubuntu@docker:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
2216ffcc8b68        ubuntu              "bash"              About an hour ago   Up About an hour                        upbeat_heisenber
g
9b6cbd84d0ad        ubuntu              "bash"              About an hour ago   Up About an hour                        bold_mestorf
ubuntu@docker:~$
```

- ps fxa | grep docker -A 3

```
ubuntu@docker:~$ ps fxa | grep docker -A 3
  3643 ?        Sl     0:00  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1.linux/moby/9b6cbd8
4d0adcd0ae8910d5cc8fc9834560fc00d7bdf17fcaec72b45556551ea -address /run/containerd/containerd.sock -containerd-binary /usr/bin/container
d -runtime-root /var/run/docker/runtime-runc
  3664 ?        Ss+    0:00  |   \_ bash
  3732 ?        Sl     0:00  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1.linux/moby/2216ffc
c8b68078b0c4722fb27ed48a6adb8c61f1fe3530e8aada60f8be5a8ad -address /run/containerd/containerd.sock -containerd-binary /usr/bin/container
d -runtime-root /var/run/docker/runtime-runc
  3750 ?        Ss+    0:00      \_ bash
  9137 ?        Ss     0:00          \_ /usr/sbin/apache2 -k start
  9140 ?        Sl     0:00              \_ /usr/sbin/apache2 -k start
--
 10155 pts/1    S+     0:00              \_ grep --color=auto docker -A 3
  1853 ?        Ss     0:00 /lib/systemd/systemd --user
  1863 ?        S      0:00  \_ (sd-pam)
  2014 ?        Ssl    0:03 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Docker Container Commands

K21Academy
Learn Cloud from Experts

- docker rm: Removes the container
- docker attach : Connect to running container
- docker exec: Connect to running container

- **docker exec vs attach**

Docker Images

K21Academy
Learn Cloud from Experts

- A Docker image is a file used to execute code in a Docker container
- A read only template
- Built from the instructions for a complete and executable version of an application
- A Docker image is made up of multiple layers
- Docker images start with a base image
- The docker run command creates a container from a given image
- The docker commit command creates image from a container

Docker Images Layers

K21Academy
Learn Cloud from Experts

- A Docker image consists of layer built on top of each other
- Docker uses Union File System (UFS) to build an image
- Docker image is immutable and when changed made adds new layer on the top
- Image is shared across containers
- Container = Image + writable top layer

Docker Image & Container

Docker Image

Package Template Plan

Docker Container #1

Docker Container #2

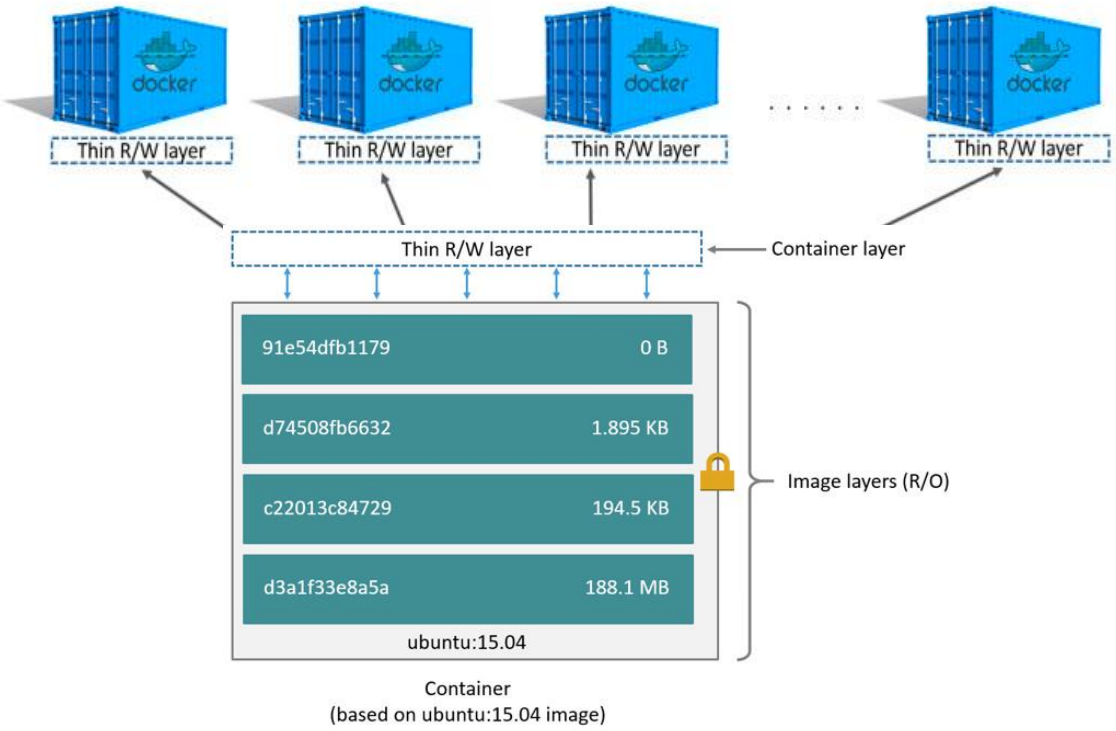
Docker Container

Docker Images Commands

K21Academy
Learn Cloud from Experts

- docker pull: Pulls image from registry
- docker images: lists all images in the local registry
- docker export: Export a container's filesystem as a tar archive
- docker import : Import the contents from a tarball to create a filesystem image
- docker rmi: Remove one or more images

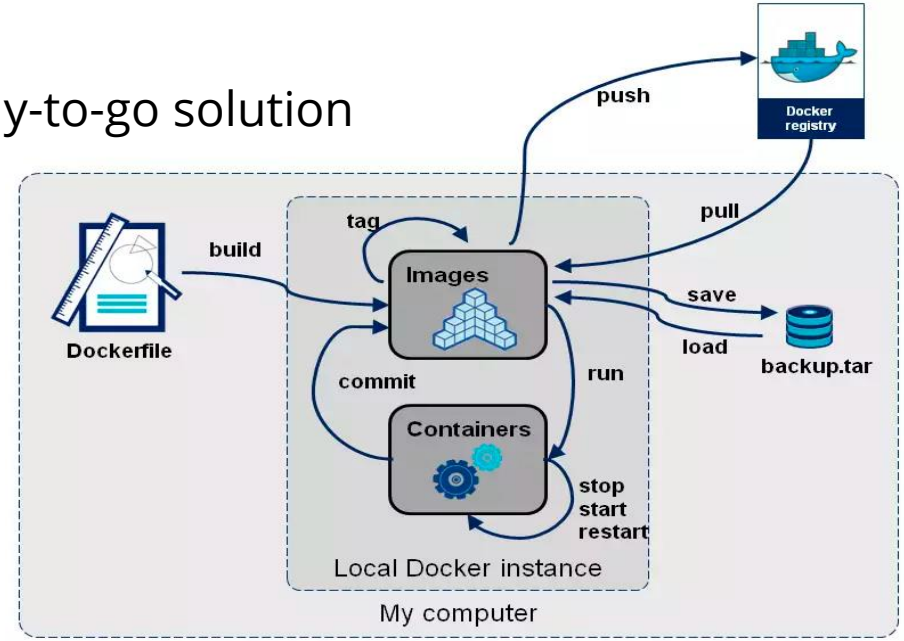
Image Layering with CoW



Docker Registry

- The Registry is a stateless, highly scalable server-side application that stores and lets you distribute Docker images

- Docker Hub - zero maintenance, ready-to-go solution



Questions

K21Academy
Learn Cloud from Experts

docker
+
kubernetes

Docker Networking

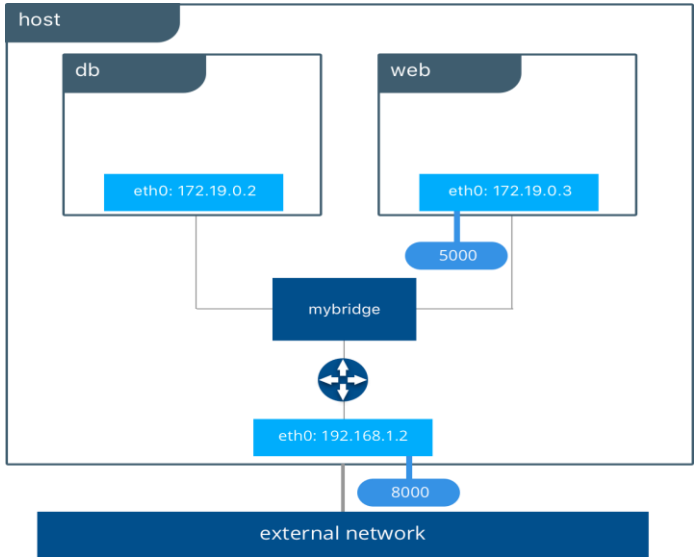
Docker Networking

K21Academy
Learn Cloud from Experts

- Docker's networking subsystem is pluggable, using drivers
- **bridge:** The default network driver
- **host:** For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly
- **none:** For this container, disable all networking.

Bridge Network

- Link Layer device which forwards traffic between network segments
- Software bridge which allows containers connected to the same bridge network to communicate
- Provides isolation from containers which are not connected to that bridge network
- Can't modify network setting of Default bridge network
- Can create user defined bridge

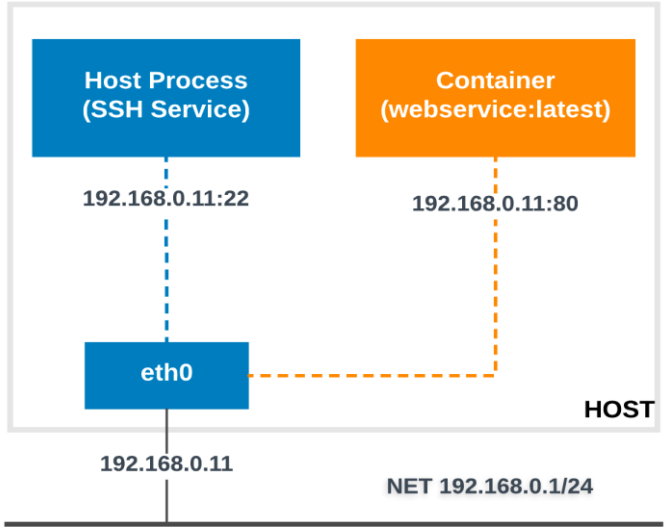


Custom Bridge Network

- docker network create --driver=bridge --subnet=172.18.0.0/16 --gateway=172.18.0.1 custom-bridge
- docker run -dt --network custom-bridge --name=Container1 ubuntu bash

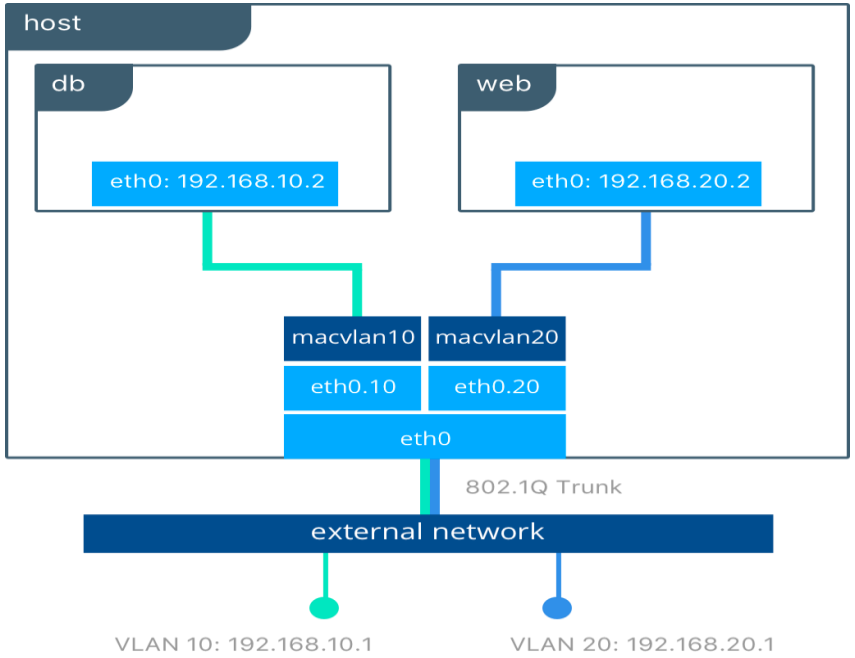
Host Network

- Container's network stack is not isolated from the Docker host
- Container does not get its own IP-address allocated
- Useful to optimize performance
- When container needs to handle a large range of ports
- Does not require network address translation (NAT)



macvlan Network

- Directly connected to the physical network
- Assigns a MAC address to each container's virtual network interface
- Useful to optimize performance
- Used in legacy applications needing MAC address



Docker Network Commands

docker network connect	Connect a container to a network
docker network create	Create a network
docker network disconnect	Disconnect a container from a network
docker network inspect	Display detailed information on one or more networks
docker network ls	List networks
docker network prune	Remove all unused networks
docker network rm	Remove one or more networks

Lab Exercise Videos: On Portal

K21Academy
Learn Cloud from Experts

Docker Notes by Trainer Ramesh

4 Module 2: [Theory] Docker Networking & Storage

- Lesson 1: Module Overview (06:41 min)
- Lesson 2: Bridge Network (19:44 min)
- Lesson 3: Host Network (7:31 min)
- Lesson 4: macvlan Network (7:06 min)
- Lesson 5: Docker Storage & Bind Mount (20:12 min)
- Lesson 6: Storage Volume & tmpfs (10:35 min)
- Lesson 7: DNS in Docker (13:59 min)

5 Module 2: [Lab] Docker Networking & Storage

- Lesson 1: Networking Bridge (13:49 min)
- Lesson 2: Networking Custom Bridge & Host (13:12 min)
- Lesson 3: Storage: Host Path Mounting (04:29 min)
- Lesson 4: Storage: Volume & tmpfs (06:41 min)

6 Module 3: [Theory & Lab] Containerize Apps Using Docker, Docker Compose &

Docker Networking Videos and Labs Exercise

K21Academy
Learn Cloud from Experts

ORACLE Gold Partner

Contents

Questions

K21Academy
Learn Cloud from Experts

docker
+
kubernetes

Docker Storage

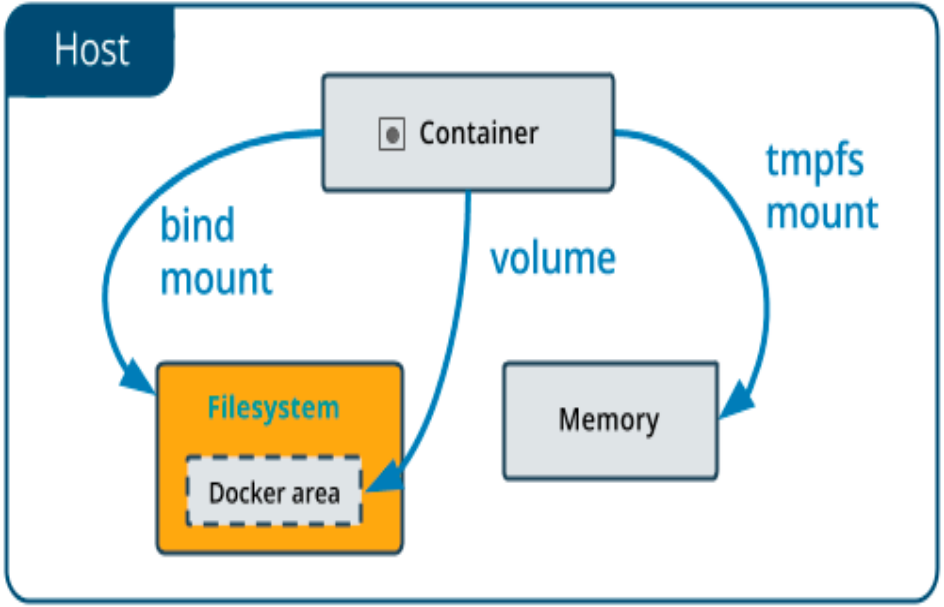
Docker Storage

K21Academy
Learn Cloud from Experts

- By default all files created inside a container are stored on a writable container layer
- The data doesn't persist when that container no longer exists
- A container's writable layer is tightly coupled to the host machine where the container is running
- Need data persistency in Containers

Docker Bind Mount

- A file or directory on the host machine is mounted into a container
- The file or directory does not need to exist on the Docker host already
- It is created on demand if it does not yet exist with -v option.
- If does not exist, it does not create with –mount option
- Can mount in RW or RO access method



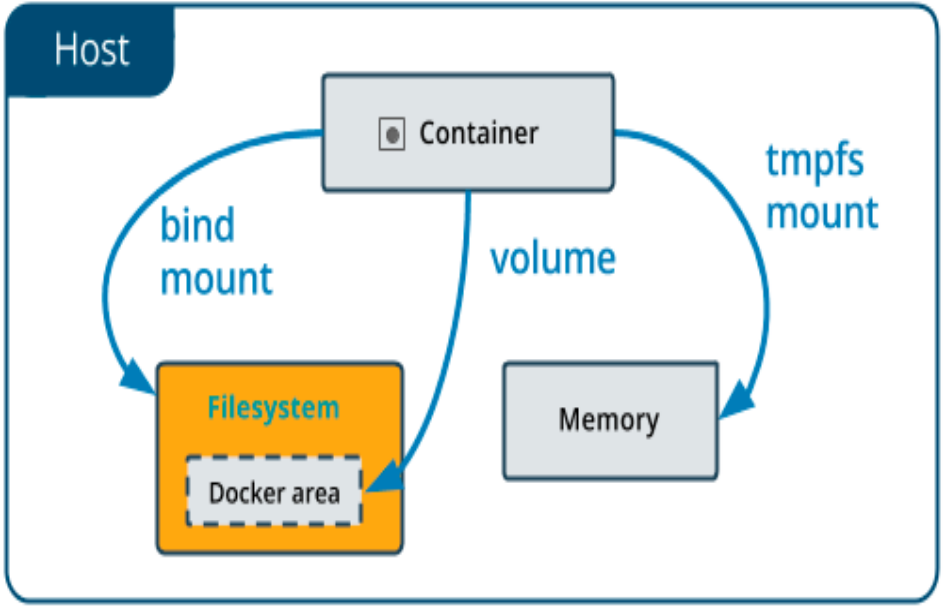
Docker Bind Mount

- docker run -d --name devtest **--mount type=bind**,source="$(pwd)”/target,target=/app nginx:latest

- docker run -d --name devtest **-v** "$(pwd)"/target:/app:ro nginx:latest

Docker tmpfs

- A tmpfs mount is not persisted on disk
- It can be used by a container during the lifetime of the container
- Stores non-persistent state or sensitive information
- Can't share tmpfs mounts between containers
- Only available if you're running Docker on Linux



Docker tmpfs Mount

K21Academy
Learn Cloud from Experts

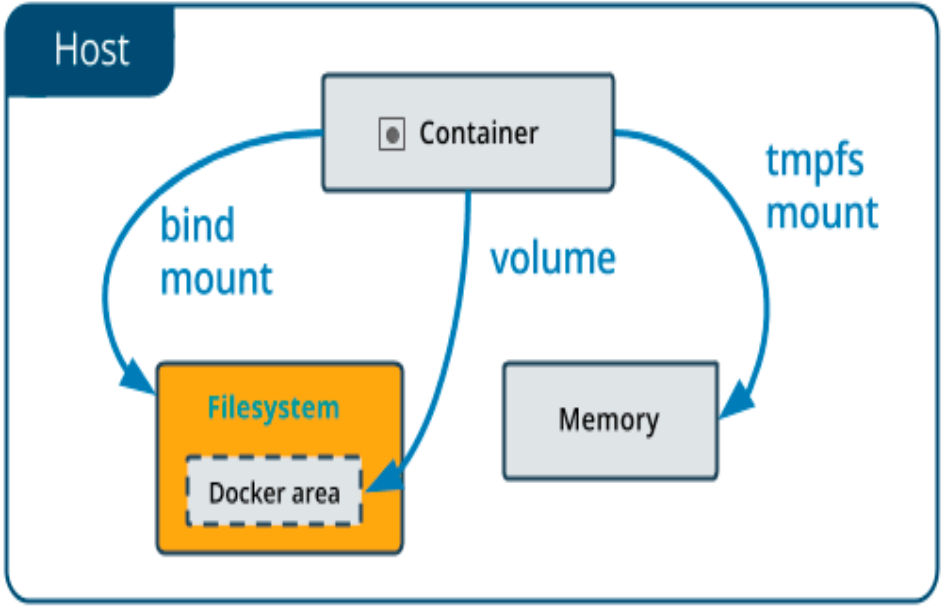
- docker run -d --name tmptest **--mount type=tmpfs**,destination=/app nginx:latest

- docker run -d --name tmptest **--tmpfs** /app nginx:latest

tmpfs-size	Size of the tmpfs mount in bytes. Unlimited by default.
tmpfs-mode	File mode of the tmpfs in octal. For instance, 700 or 0770. Defaults to 1777 or world-writable.

Docker Volume

- Preferred mechanism for persisting data generated by and used by Docker containers
- Volumes are completely managed by Docker
- Volume can be mounted into multiple containers simultaneously



Docker Volume

K21Academy
Learn Cloud from Experts

- docker volume create my-vol
- docker volume create --driver local --opt type=tmpfs --opt device=tmpfs --opt o=size=100m my-tmpfs
- docker volume create --driver local --opt type=btrfs --opt device=/dev/sda2 my-vol
- docker volume create --driver local --opt type=nfs --opt o=addr=192.168.1.1,rw --opt device=:/path/to/dir my-nfs
- docker run -d --name devtest --mount source=my-vol,target=/app nginx:latest
- docker run -d --name=nginxtest -v my-vol:/app:ro nginx:latest

Lab Exercise Videos: On Portal

K21Academy
Learn Cloud from Experts

Docker Notes by Trainer Ramesh

4 Module 2: [Theory] Docker Networking & Storage

- Lesson 1: Module Overview (06:41 min)
- Lesson 2: Bridge Network (19:44 min)
- Lesson 3: Host Network (7:31 min)
- Lesson 4: macvlan Network (7:06 min)
- Lesson 5: Docker Storage & Bind Mount (20:12 min)
- Lesson 6: Storage Volume & tmpfs (10:35 min)
- Lesson 7: DNS in Docker (13:59 min)

5 Module 2: [Lab] Docker Networking & Storage

- Lesson 1: Networking Bridge (13:49 min)
- Lesson 2: Networking Custom Bridge & Host (13:12 min)
- Lesson 3: Storage: Host Path Mounting (04:29 min)
- Lesson 4: Storage: Volume & tmpfs (06:41 min)

6 Module 3: [Theory & Lab] Containerize Apps Using Docker, Docker Compose &

Docker Storage Videos and Labs Exercise

K21Academy
Learn Cloud from Experts

ORACLE Gold Partner

Contents

Questions

K21Academy
Learn Cloud from Experts

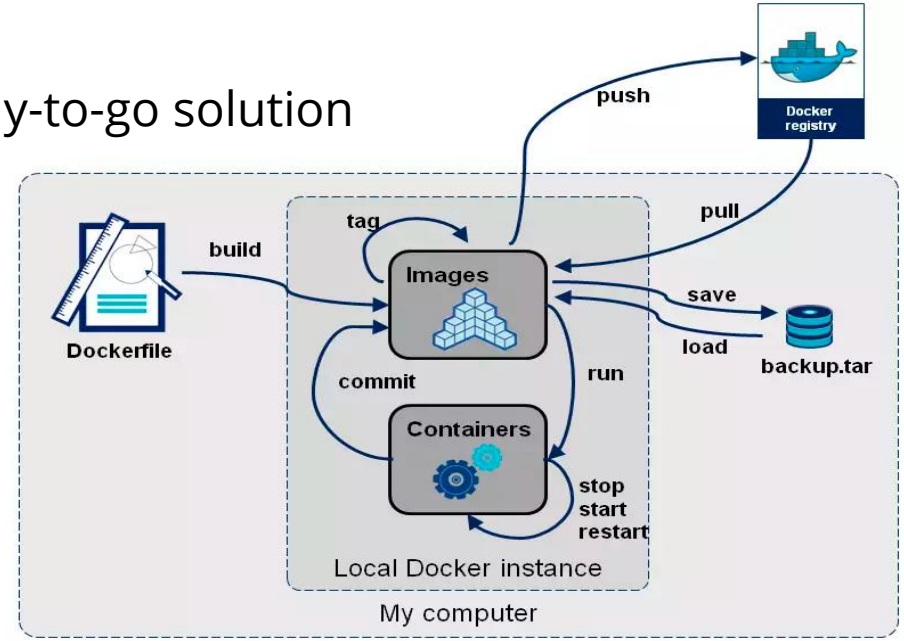
docker
+
kubernetes

Building Docker Images

Docker Registry

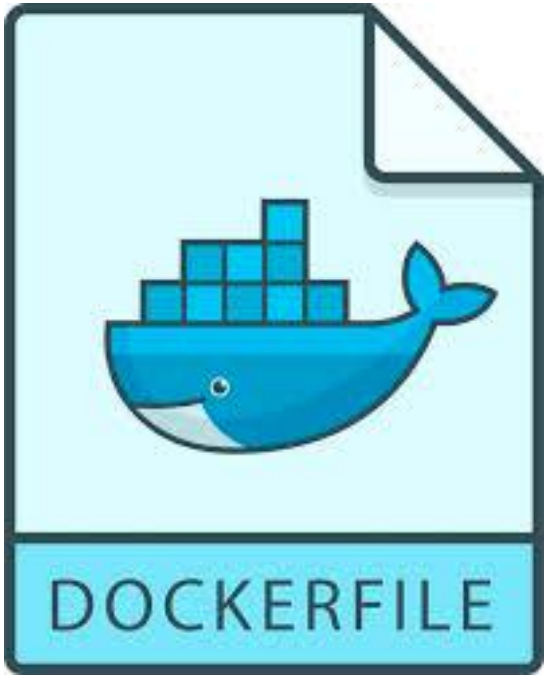
- The Registry is a stateless, highly scalable server-side application that stores and lets you distribute Docker images

- Docker Hub - zero maintenance, ready-to-go solution



Dockerfile

- Build images automatically by reading the instructions from a Dockerfile
- A text document that contains all the commands a user could call on the command line to assemble an image
- The docker build command builds an image from a Dockerfile
- The build is run by the Docker daemon
- The Docker daemon runs the instructions in the Dockerfile one-by-one, committing the result of each instruction to a new image if necessary, before finally outputting the ID of your new image.



Dockerfile Instructions

K21Academy
Learn Cloud from Experts

FROM

ARG

```
ARG VERSION=latest
FROM busybox:$VERSION
ARG VERSION
RUN echo $VERSION > image_version
```

RUN

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

```
RUN ["/bin/bash", "-c", "echo hello"]
```

LABEL

```
LABEL multi.label1="value1" multi.label2="value2" other="value3"
```

Dockerfile Instructions

K21Academy
Learn Cloud from Experts

EXPOSE

```
EXPOSE 80/tcp
EXPOSE 80/udp
```

- Does not actually publish the port
- It functions as a type of documentation between the person who builds the image and the person who runs the container

ENV

```
ENV myName John Doe
ENV myDog Rex The Dog
ENV myCat fluffy
```

ADD

```
ADD test.txt relativeDir/
```

Container as Executable

- Both ENTRYPOINT and CMD give you a way to identify which executable should be run when a container is started from your image
- Trying to run an image which doesn't have an ENTRYPOINT or CMD declared will result in an error

```
$ docker run alpine
FATA[0000] Error response from daemon: No command specified
```

- Most of the Linux distro base images that you find on the Docker Hub will use a shell like **/bin/sh** or **/bin/bash** as the the CMD executable
- So when we runs such images will get dropped into an interactive shell by default

Dockerfile - CMD

K21Academy
Learn Cloud from Experts

- The ENTRYPOINT or CMD that you specify in your Dockerfile identify the default executable for your image
- The user has the option to override either of these values at run time

For example, let's say that we have the following Dockerfile

```
FROM ubuntu:trusty
CMD ping localhost
```

If we build this image (with tag "demo") and run it we would see the following output:

```
$ docker run -t demo
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.038 ms
^C
--- localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.026/0.032/0.039/0.008 ms
```

You can see that the *ping* executable was run automatically when the container was started. However, we can override the default CMD by specifying an argument **after** the image name when starting the container:

```
$ docker run demo hostname
6c1573c0d4c0
```

In this case, *hostname* was run in place of *ping*

Dockerfile - CMD

- Use CMD when we want the user of your image to have the flexibility to run whichever executable they choose when starting the container
- There can only be one CMD instruction in a Dockerfile
- If you list more than one CMD then only the last CMD will take effect

For example, let's say that we have the following Dockerfile

```
FROM ubuntu:trusty
CMD ping localhost
```

If we build this image (with tag "demo") and run it we would see the following output:

```
$ docker run -t demo
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.038 ms
^C
--- localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.026/0.032/0.039/0.008 ms
```

You can see that the *ping* executable was run automatically when the container was started. However, we can override the default CMD by specifying an argument **after** the image name when starting the container:

```
$ docker run demo hostname
6c1573c0d4c0
```

In this case, *hostname* was run in place of *ping*

Override CMD and ENTRYPOINT

K21Academy
Learn Cloud from Experts

```
FROM ubuntu:trusty
CMD ping -c3 localhost
```

```
$
$ docker run demo_cmd
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.035 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.099 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.099 ms

--- localhost ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2117ms
rtt min/avg/max/mdev = 0.035/0.077/0.099/0.031 ms
$
```

OR

```
$
$ docker run demo_cmd hostname
63a109a9ef95
$
```

```
FROM ubuntu:trusty
ENTRYPOINT ping -c3 localhost
```

```
$
$ docker run demo
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.086 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.043 ms

--- localhost ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2118ms
rtt min/avg/max/mdev = 0.041/0.056/0.086/0.022 ms
$
$
```

OR

```
$ docker run --entrypoint google.com demo
docker: Error response from daemon: OCI runtime create failed: container_linux.go:348: starting container process caused
"exec: \"google.com\": executable file not found in $PATH": unknown.
ERRO[0001] error waiting for container: context canceled
$
```

Override

K21Academy
Learn Cloud from Experts

- We can override either of these values at run time
- Much easier it is to override the CMD
- Use CMD when we want the user of your image to have the flexibility to run whichever executable they choose when starting the container
- ENTRYPOINT should be used in scenarios where you want the container to behave exclusively as if it were the executable it's wrapping
- When we don't want or expect the user to override the executable you've specified

CMD shell vs. Exec form

- CMD instructions supports two different forms:
  - *shell form –>* **CMD ping localhost**

```
$ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
88000b0c7087        demo_cmd            "/bin/sh -c 'ping -c…"   2 seconds ago       Up 2 seconds                            elastic_fermi
$ docker exec 88000 ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 02:39 ?        00:00:00 /bin/sh -c ping -c10 localhost
root         7     1  0 02:39 ?        00:00:00 ping -c10 localhost
root         8     0  0 02:39 ?        00:00:00 ps -f
$
```

  - *exec form –>* **CMD ["/bin/ping","localhost"]**

```
$
$ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
0e73a98d03e2        cmd_exec            "/bin/ping localhost"    28 seconds ago      Up 28 seconds                           relaxed_dijkstra
$ docker exec 0e73a ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 02:42 ?        00:00:00 /bin/ping localhost
root         7     0  0 02:42 ?        00:00:00 ps -f
$
```

ENTRYPOINT and CMD

K21Academy
Learn Cloud from Experts

- Combining ENTRYPOINT and CMD allows you to specify the default executable for your image, let's also provide default arguments to that executable which may be overridden by the user

```
FROM ubuntu:trusty
ENTRYPOINT ["/bin/ping","-c","20"]
CMD ["localhost"]
~
```

```
$ docker run cmd_etp
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.096 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.101 ms
```

```
$ docker run cmd_etp google.com
PING google.com (172.217.31.206) 56(84) bytes of data.
64 bytes from maa03s28-in-f14.1e100.net (172.217.31.206): icmp_seq=1 ttl=37 time=38.4 ms
64 bytes from maa03s28-in-f14.1e100.net (172.217.31.206): icmp_seq=2 ttl=37 time=36.9 ms
64 bytes from maa03s28-in-f14.1e100.net (172.217.31.206): icmp_seq=3 ttl=37 time=39.2 ms
```

```
$
$ docker run -it --entrypoint /bin/bash cmd_etp
root@5d41d0a621ef:/#
root@5d41d0a621ef:/#
root@5d41d0a621ef:/#
root@5d41d0a621ef:/#
```

Dockerfile Instructions

K21Academy
Learn Cloud from Experts

VOLUME

- Creates a mount point with the specified name

```
VOLUME ["/data"]
```

USER

```
USER <user>[:<group>]
```

- Sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile

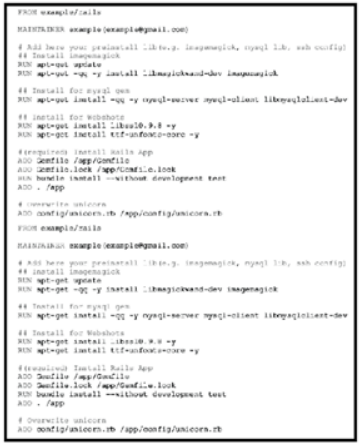
WORKDIR

```
WORKDIR /path/to/workdir
```

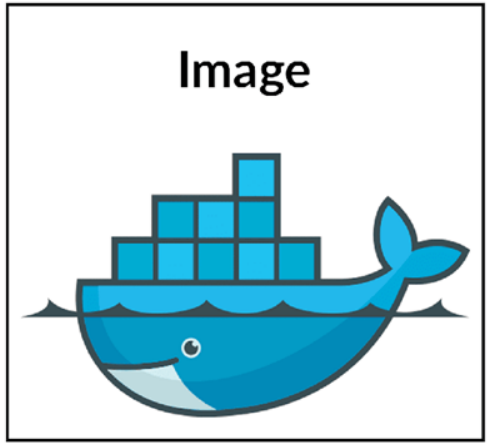
Docker Build Command

- docker build .
- docker build -t apache:2.0 .
- docker build -f Dockerfile1 .
- docker build -f dockerfiles/Dockerfile1 -t new_image .

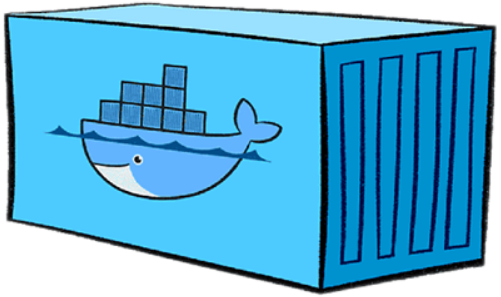
Build Image



Dockerfile



Docker Image



Docker Container

docker

+

kubernetes

K21Academy
Learn Cloud from Experts

Docker Compose

Docker Compose

- Deploy complete application stack to the swarm
- Compose is a tool for defining and running multi-container Docker applications
- With Compose, we use a YAML file to configure your application's services
- With a single command, you create and start all the services from your configuration
- Run docker-compose up and Compose starts and runs your entire app

```
version: '2.0'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Lab Exercise Videos: On Portal

K21Academy
Learn Cloud from Experts

Lesson 4: Storage: Volume & tmpfs (06:41 min)

6 Module 3: [Theory & Lab] Containerize Apps Using Docker, Docker Compose & Swarm

Lesson 1: Automate Image Creation Using Dockerfile (36:09 min)

Lesson 2: Docker Compose & YAML (17:10 min)

Lesson 3: Clustering Docker using Docker Swarm (03:2

Lesson 4: YAML Basics for Docker

Create Docker Image Videos and Labs Exercise

K21Academy
Learn Cloud from Experts

ORACLE Gold Partner

Contents

Questions

K21Academy
Learn Cloud from Experts

docker
+
kubernetes

Docker DNS

Docker Native DNS

K21Academy
Learn Cloud from Experts

- Inbuilt DNS which automatically resolves IP to container names
- In user-defined docker network DNS resolution to container names happens automatically
- When you run new container on the docker host without any DNS related option in command
- It simply copies host's /etc/resolv.conf into container
- Docker daemon smartly adds Google's public nameservers 8.8.8.8 and 8.8.4.4 into file and use it within the container
- Docker daemon takes help from file change notifier
- Makes necessary changes in container's resolve file when there are changes made in host's file

External DNS

- Define the external DNS IP in docker daemon configuration file /etc/docker/daemon.json
- Restart docker daemon to pick up these new changes

Logging Driver

- Docker captures the standard output of all containers and writes them in files using the JSON format
- Update logging driver for all containers in /etc/docker/daemon.json file
- Use –logging-driver to start container with specific driver

Storage Driver

- The storage driver controls how images and containers are stored and managed on your Docker host

- Storage drivers allows to create data in the writable layer of the container

K21Academy
Learn Cloud from Experts

docker

+

kubernetes

Docker Cluster

Swarm Cluster

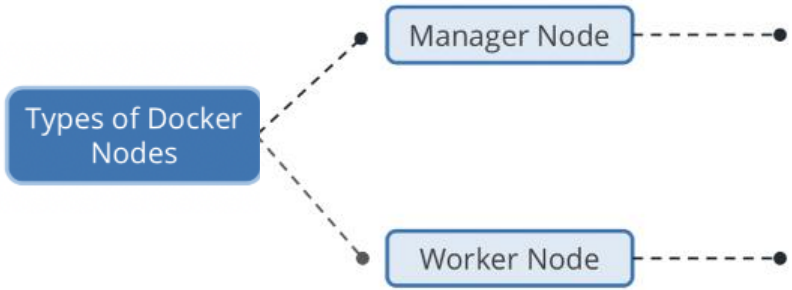
K21Academy
Learn Cloud from Experts

- A swarm consists of multiple Docker hosts which run in swarm mode and act as
  - Managers
  - Workers

- A Docker host can be a manager, a worker, or perform both roles

- Key advantage of swarm services over standalone container is that we modify configuration

Swarm Nodes



A manager node must receive a service definition in order to deploy an application to a swarm

Tasks dispatched from manager nodes are received and executed

Questions

Summary: Module

- Docker Basics Concepts
- Docker Architecture
- Docker Images
- Docker Networking
- Docker Storage
- Automate Image Creation – Dockerfile
- Docker Host -Networks and volumes
- Compose tool
- Docker Cluster
- Hands-On Guides

Find Us

https://www.facebook.com/K21Academy

http://twitter.com/k21Academy

https://www.linkedin.com/company/k21academy

https://www.youtube.com/k21academy

https://www.instagram.com/k21academy