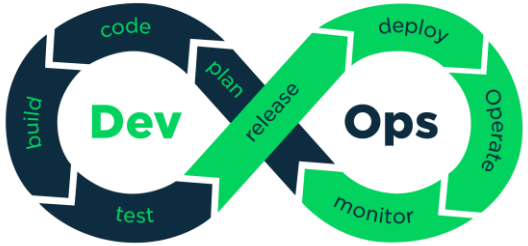
DevOps – Let the Journey Begin



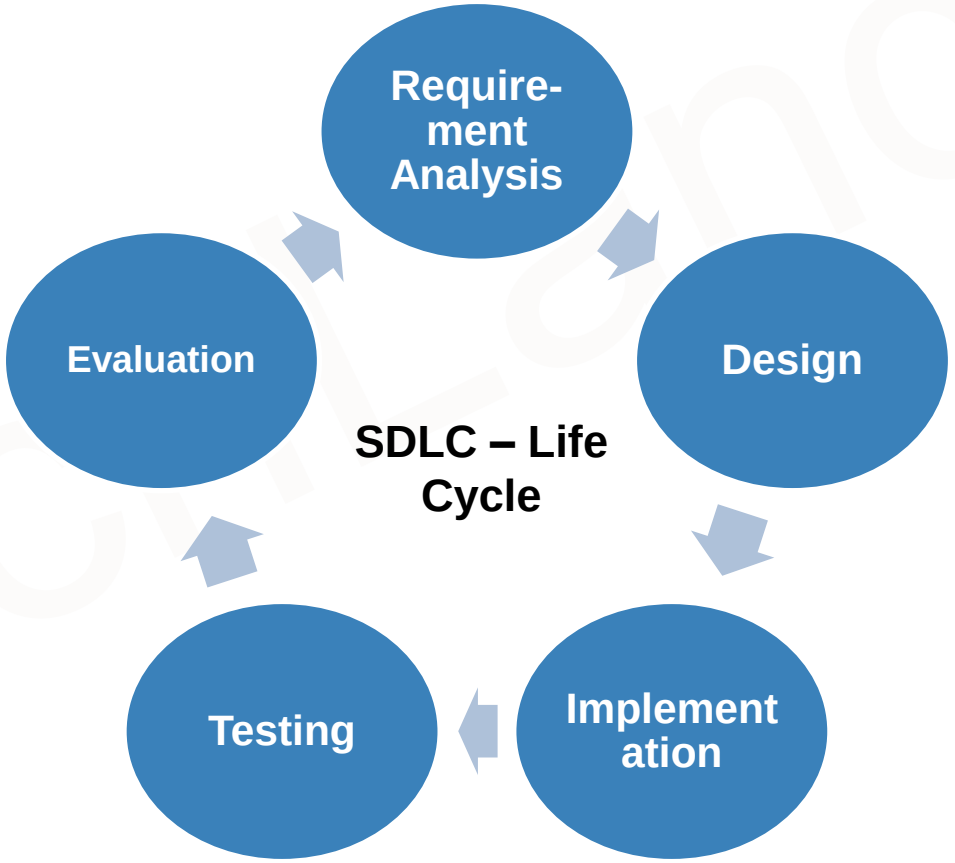
Raman Khanna

DevOps

What is DevOps?

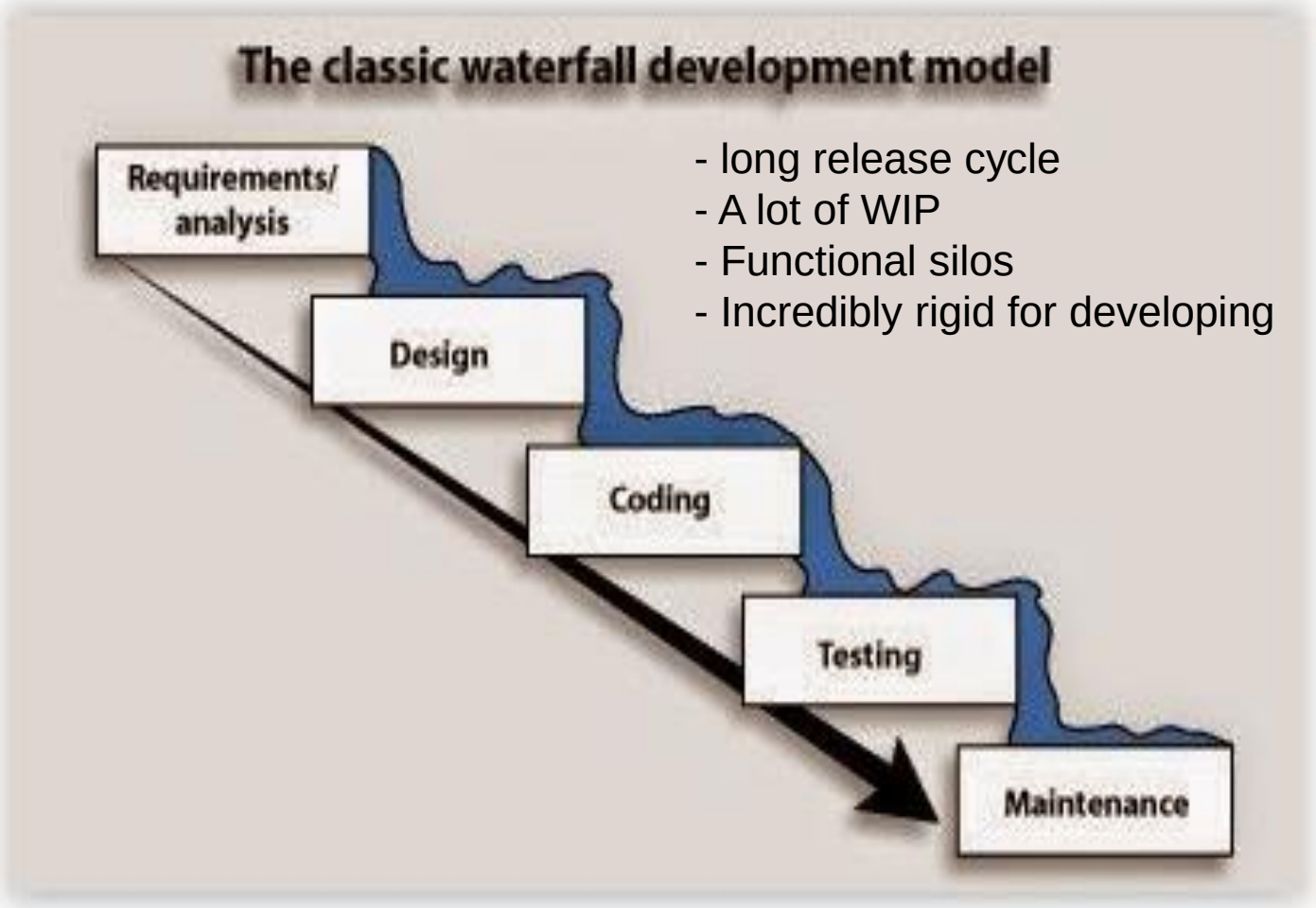
SDLC Model

- A systems development life cycle is composed of **several clearly defined and distinct work phases** which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems

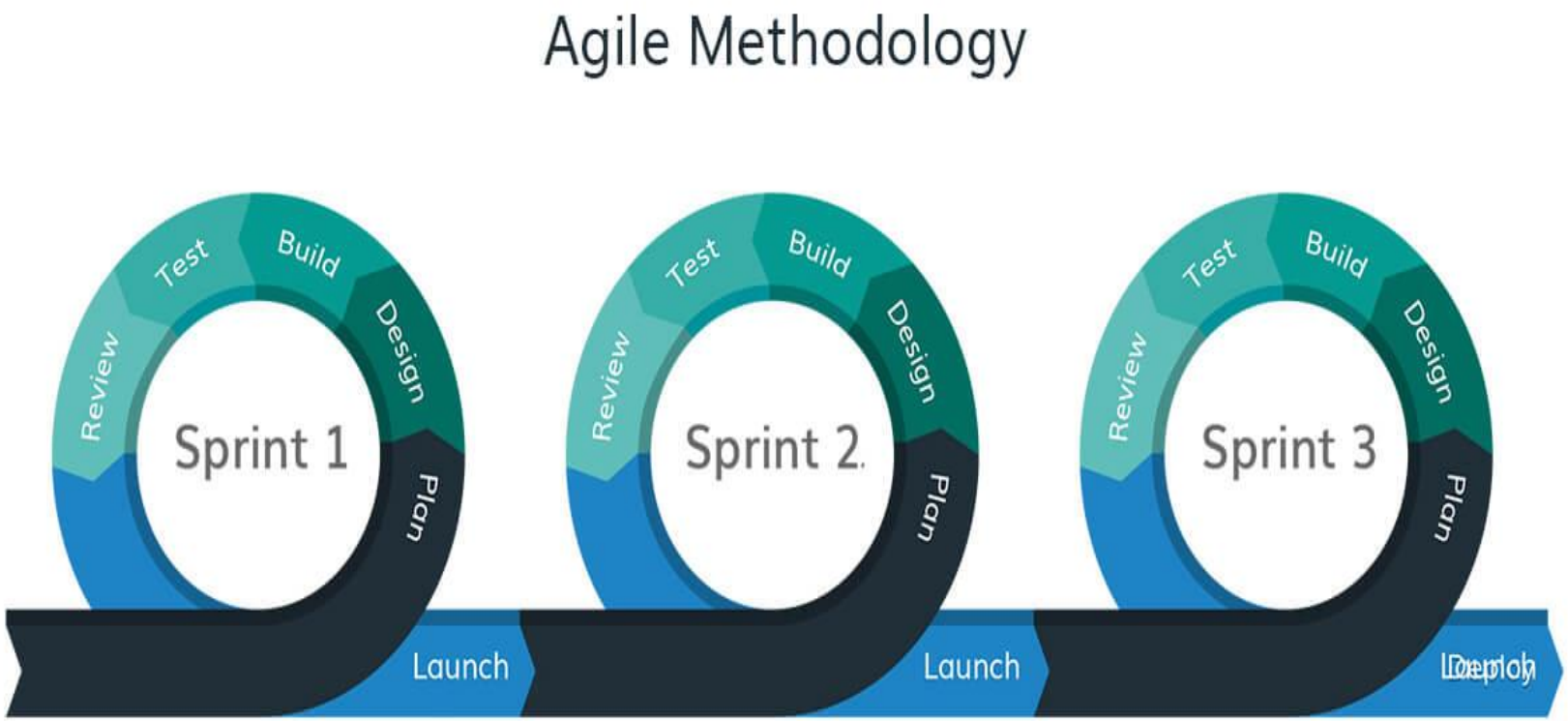


Waterfall Model

1. Determine the Requirements
2. Complete the design
3. Do the coding and testing (unit tests)
4. Perform other tests (functional tests, non-functional tests, Performance testing, bug fixes etc.)
5. At last deploy and maintain

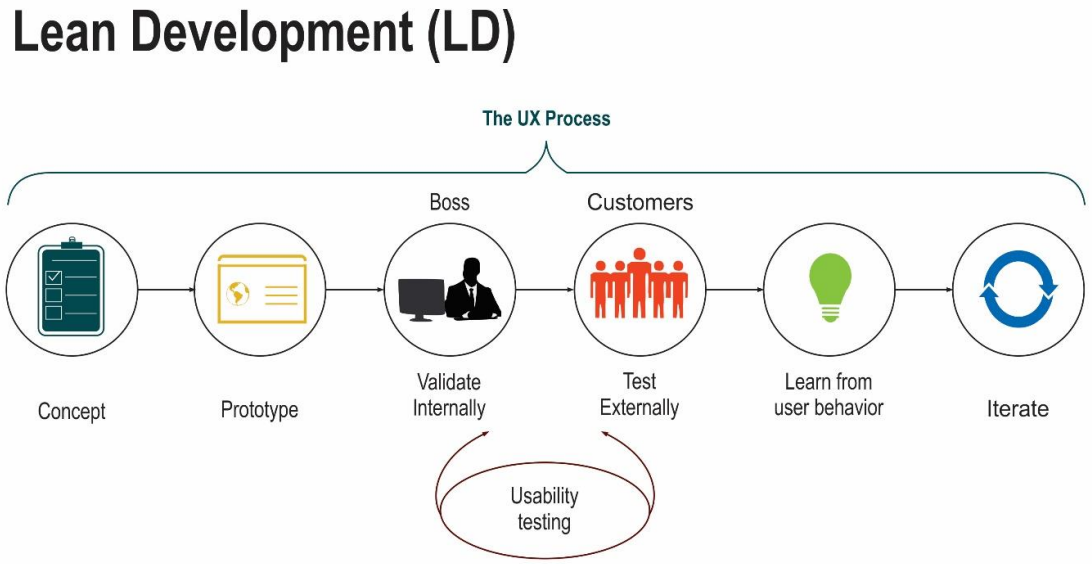


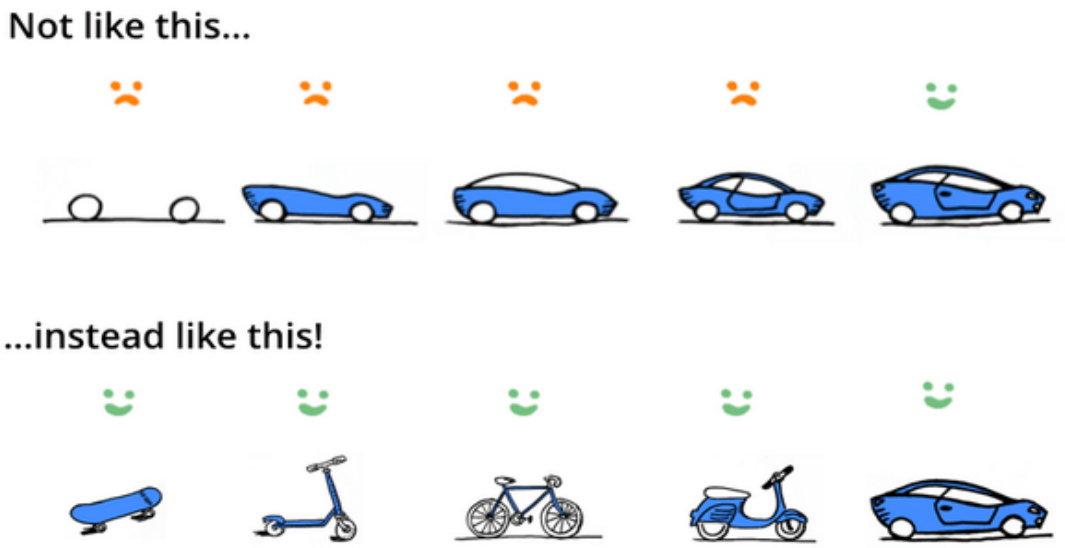
Agile



- Shorter release cycle
- Small batch sizes (MVP)
- Cross-functional teams
- Incredibly agile

Lean Development

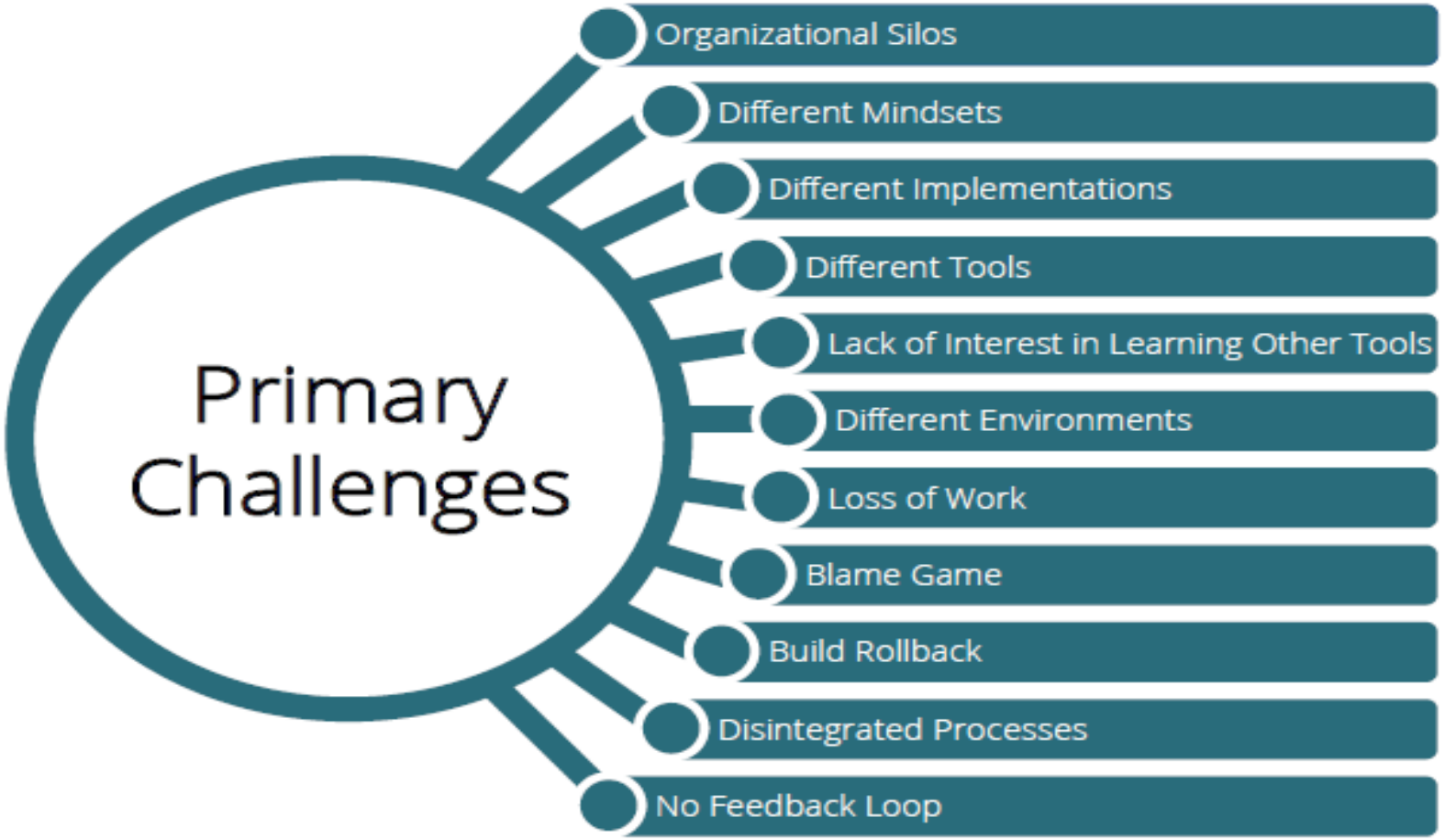




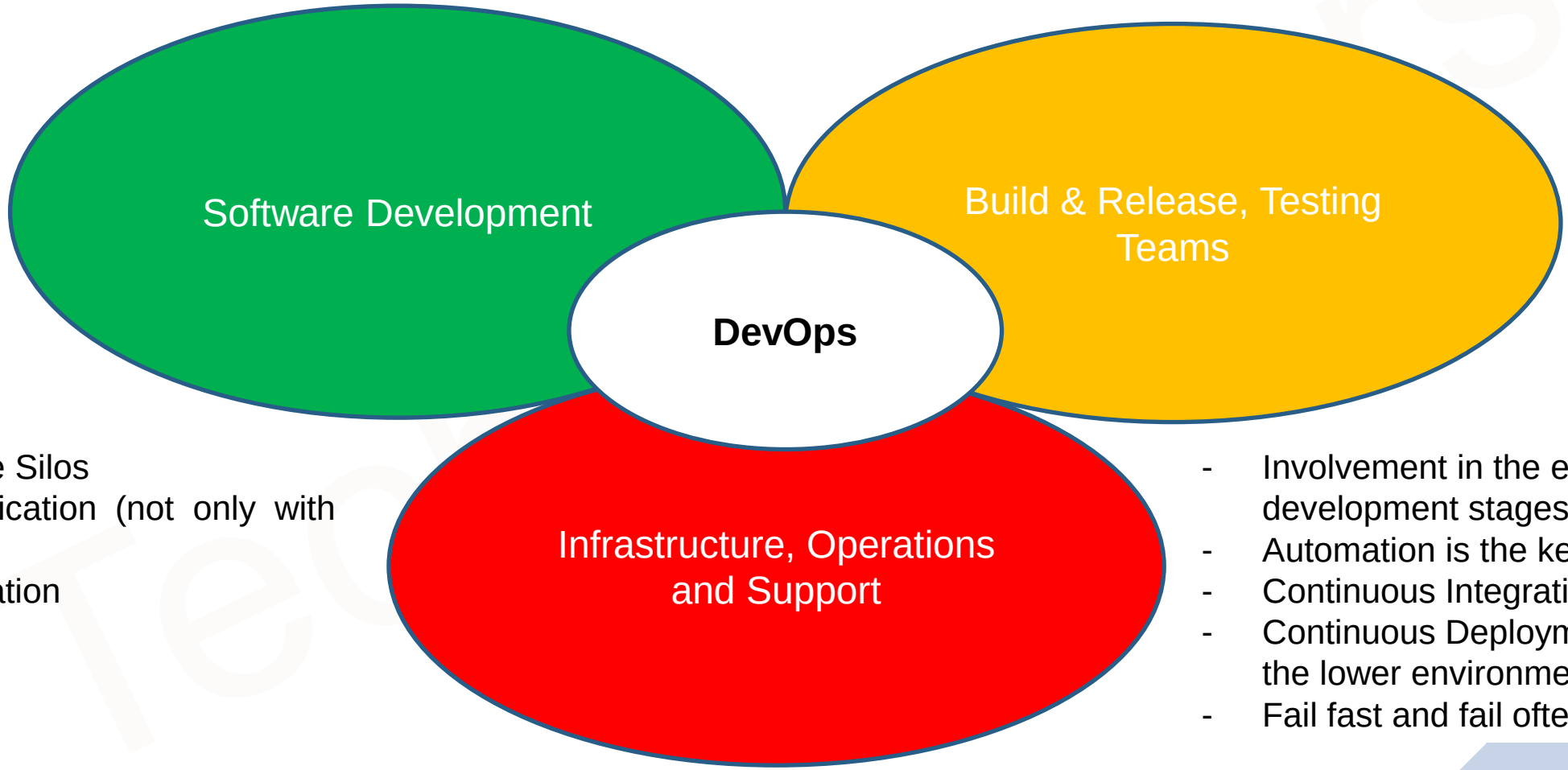
- Suddenly ops was the bottleneck (more release less people), again WIP is more!

Challenges

Some of the challenges with the traditional teams of Development and Operations are:



DevOps

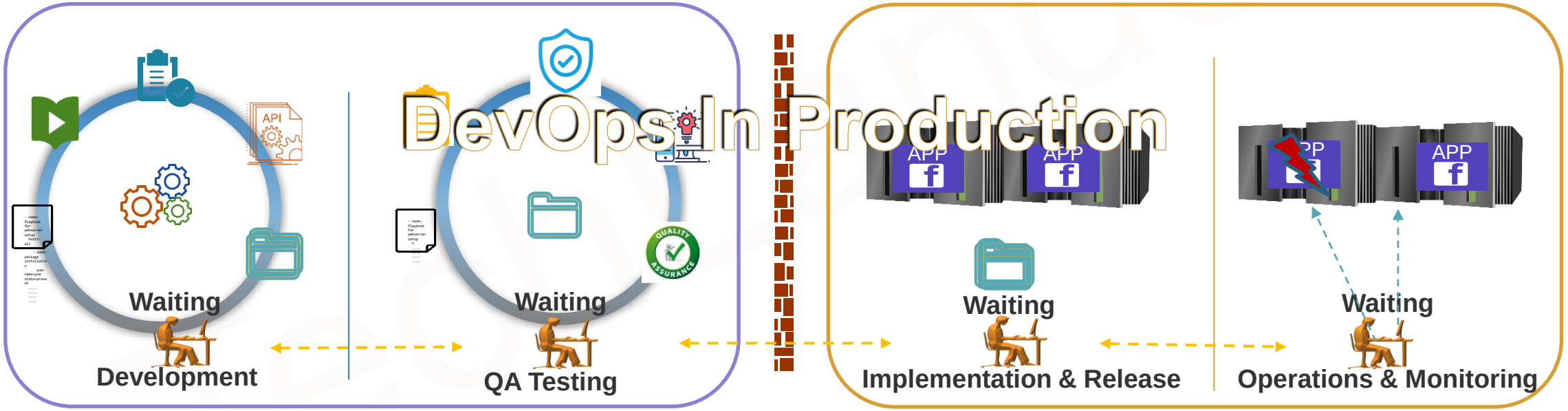


- Break the Silos
- Communication (not only with emails)
- Collaboration
- Trust

- Involvement in the early development stages
- Automation is the key
- Continuous Integration
- Continuous Deployments in the lower environments
- Fail fast and fail often

Continuous Feedback Continuous Improvement Continuous Planning

Continuous Integration Continuous Delivery Continuous Deployment Continuous Monitoring



DevOps Essence

Efficiency - Faster time to market

Predictability - Lower failure rate of new releases

Reproducibility – Version everything

Maintainability - Faster time to recovery in the event of a new release crashing or otherwise disabling the current system

DevOps Core Principals

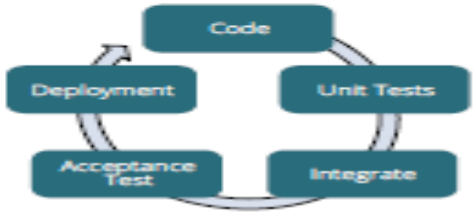
1. Customer-Centric Action

4. Cross-Functional Autonomous Teams

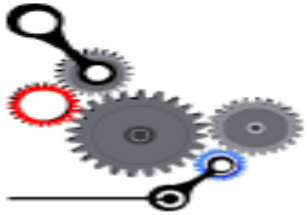
2. Create with the End in Mind

5. Continuous Improvement

3. End-to-End Responsibility



6. Automate Everything You can



How to Build DevOps Organization Culture

Retention is as important as recruitment

Establish Cross-functional team structure

Small teams are better

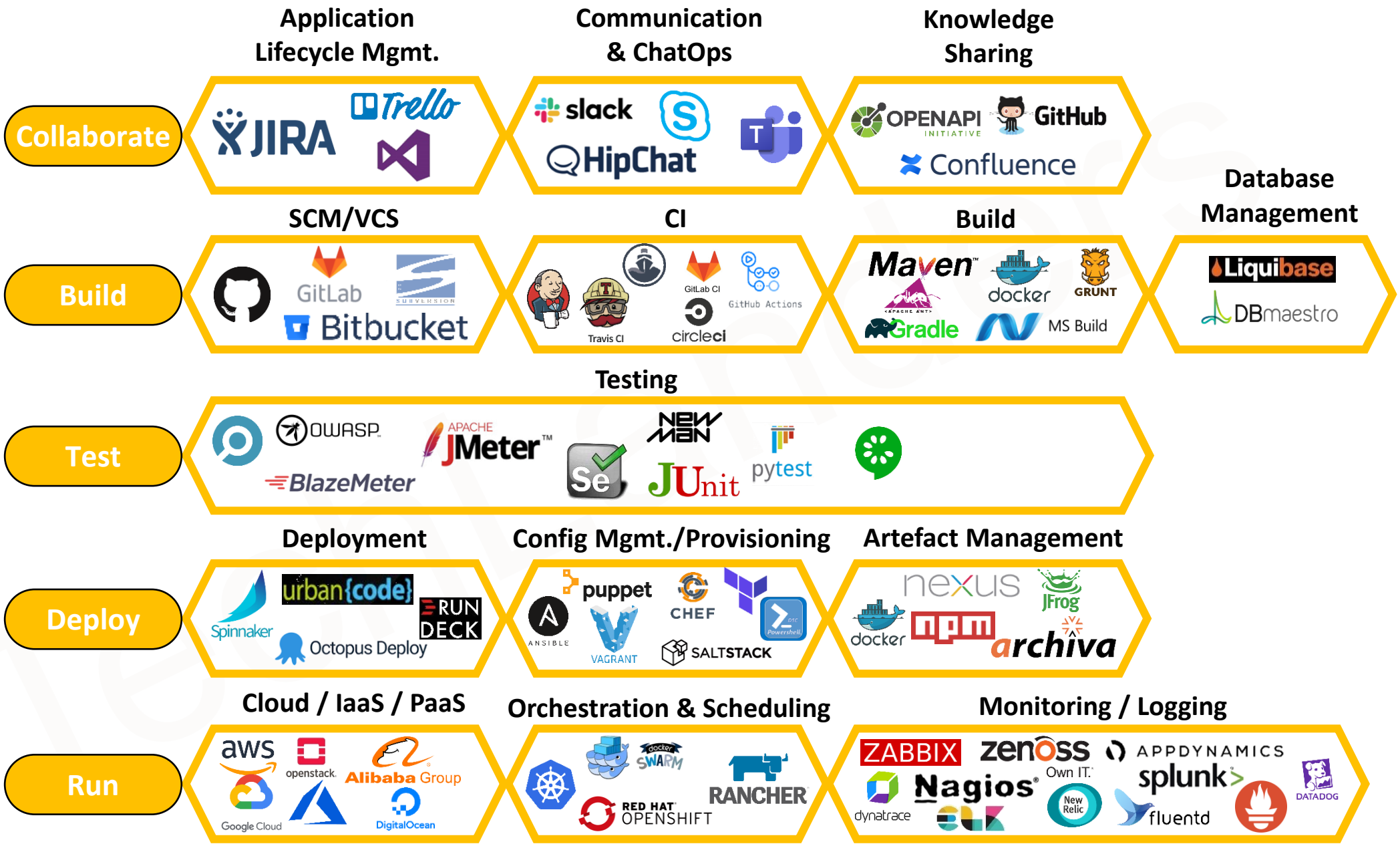
Cool tools can attract and retain

Give autonomy

Automate with existing staffs and give them a chance to learn

Take out few resources from each team, build a new virtual team for automation.

DevOps Toolsets



Version Control Systems with GIT

What is Version Control System

As name states Version Control System is the **“Management of changes to anything”.**

Version Control is way of storing files in central location accessible to all team members and enabling them to keep track of changes being done in the source code by whom, when & why. It also help teams to recover from some inevitable circumstances.

Think about traditional versioning of file with names – Login001.java, Login002.java, Login_final.java.

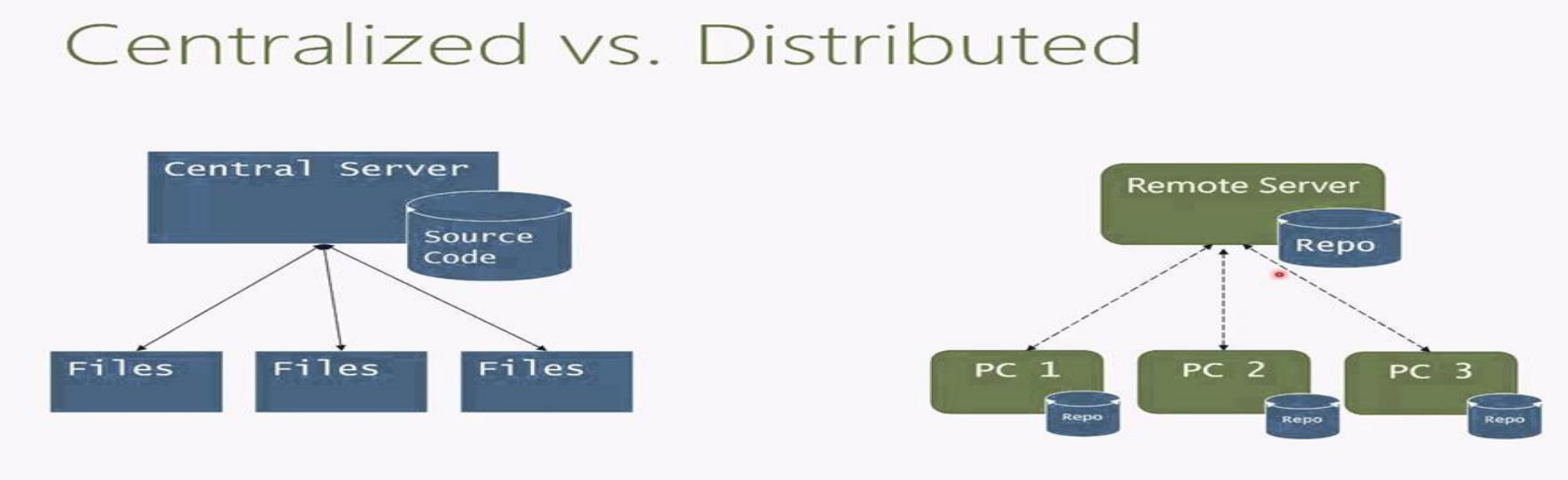
Its not just for code, it also helps in

- Backups & Restoration
- Synchronization
- Reverts
- Track Changes
- Most importantly in Parallel Development

Types of VCS

Majorly VCS is divided into two parts:

- Centralized Version Control System – CVS, Subversion, Visual Source Safe
- Distributed Version Control System – Mercurial, Bitkeeper, Git



Git History

Linus uses BitKeeper to manage Linux code

Ran into BitKeeper licensing issue
Liked functionality
Looked at CVS as how not to do things

April 5, 2005 - Linus sends out email showing first version

June 15, 2005 - Git used for Linux version control

Why Git?

- **Branching:** gives developers a great flexibility to work on a replica of master branch.

- **Distributed Architecture:** The main advantage of DVCS is **"no requirement of network connections to central repository"** while development of a product.

- **Open-Source:** Free to use.

- **Integration with CI:** Gives faster product life cycle with even faster minor changes.

Git Installation

- yum install autoconf libcurl-devel expat-devel gcc gettext-devel kernel-headers openssl-devel perl-devel zlib-devel –y
- Visit git release page - https://github.com/git/git/releases and pick desired version.
- curl -O -L https://github.com/git/git/archive/v2.14.0.tar.gz
- tar -zxvf v2.14.2.tar.gz
- cd git-v2.14.0
- make clean
- make configure
- ./configure --prefix=/usr/local
- make
- make install
- ln -s /usr/local/bin/git /usr/bin/git

Git Commands

- git –version

- [root@techlanders ~]# git config --global user.email “Gagandeep.singh@techlanders.com”
- [root@techlanders ~]# git config --global user.name “Gagandeep Singh”
- [root@techlanders ~]# git config --global -l
- user.name=Gagandeep Singh
- user.email=Gagandeep.singh@techlanders.com
- [root@techlanders ~]#

- //Initializing a repo
- [root@master git]# mkdir /Repo1
- [root@master git]# cd /Repo1/
- [root@master Repo1]# git init
- Initialized empty Git repository in /Repo1/.git/
- [root@master Repo1]#

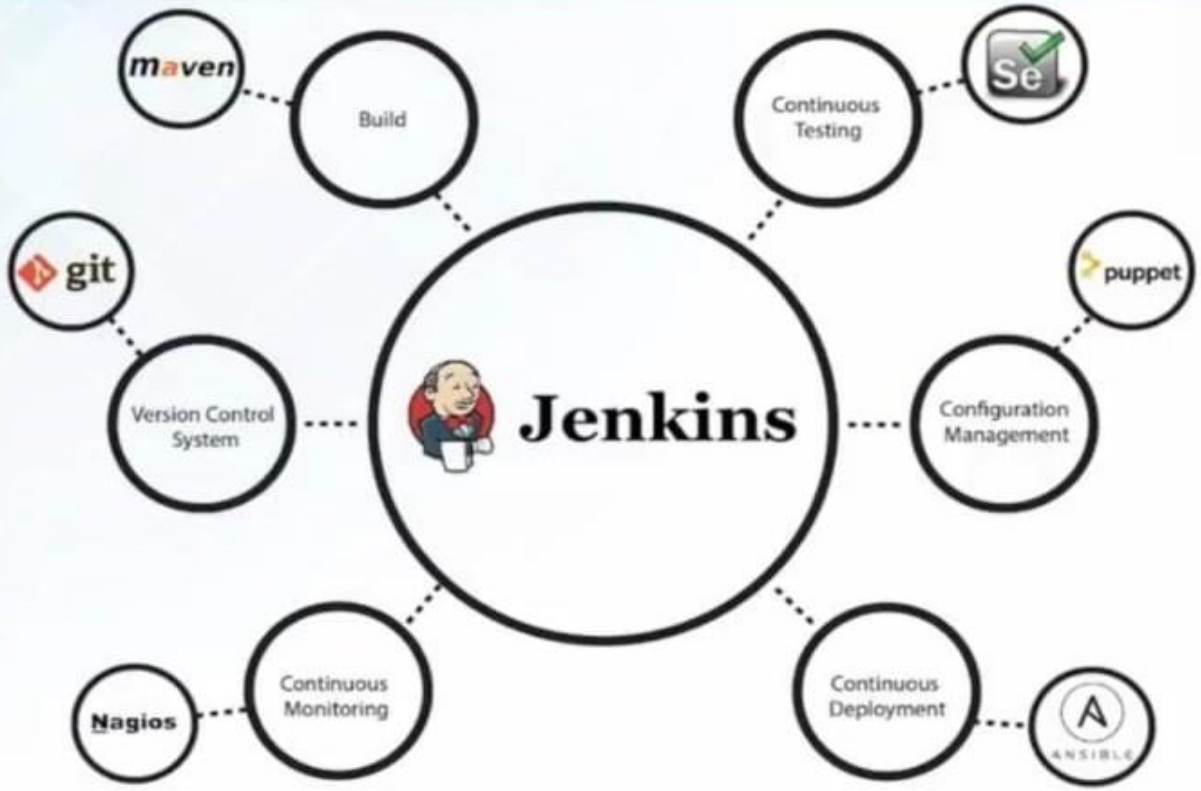
Github/Bitbucket

- **What is Bitbucket /GitHub/Gitlab?.**
- Bitbucket is a Git solution for professional teams. In simple layman language its a UI for Git, offered by Atlassian, similarly we have different available UI solutions from Github (most famous) and Gitlab.
- GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
- **Host in the cloud**: free for small teams (till 5 users) and paid for larger teams.
- **Host on Your server**: One-Time pay for most solutions.
- Visit "https://bitbucket.org/" and click "Get Started" to sign up for free account.
- Visit "https://github.com/" for Github details

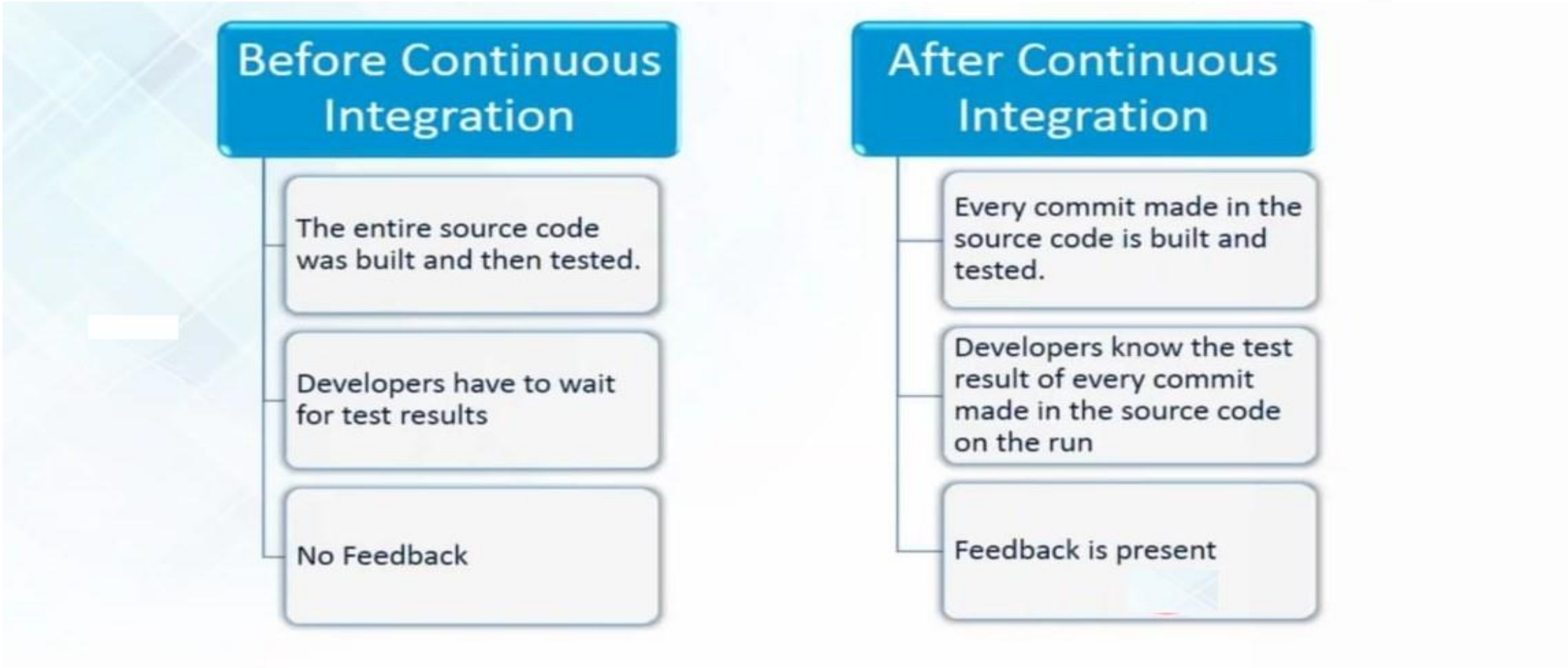
Continuous Integration with Jenkins

What is Jenkins?

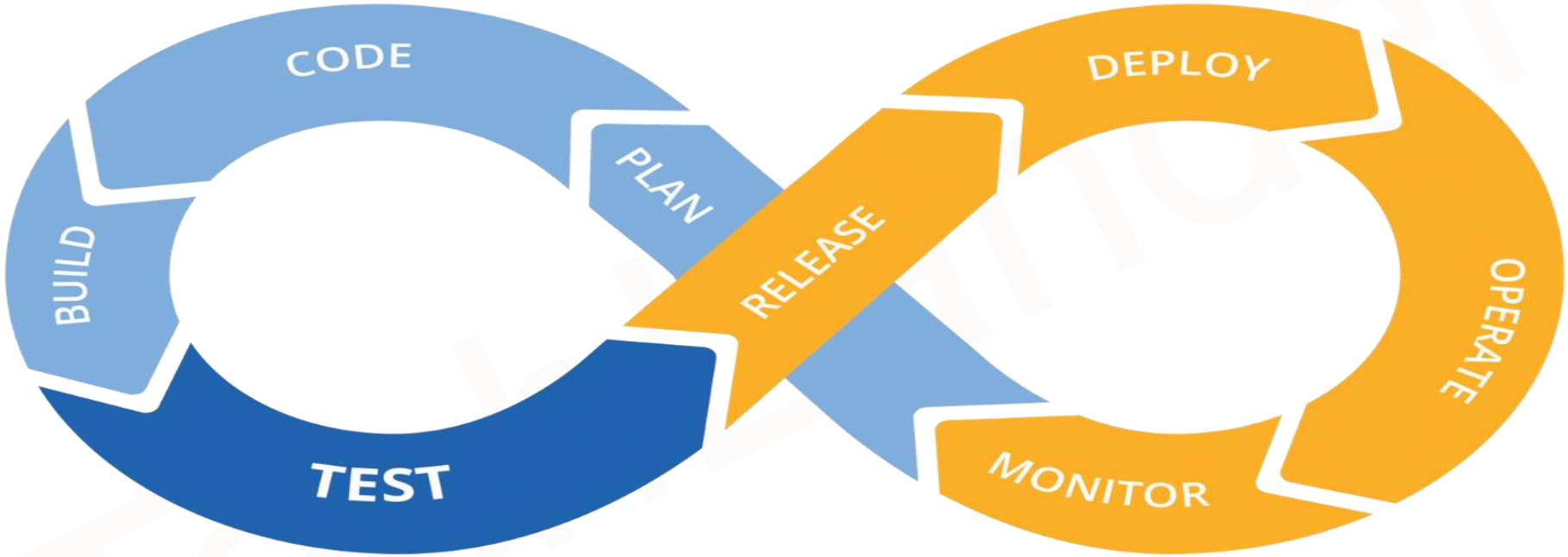
Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Plugins allows integration of various DevOps stages.



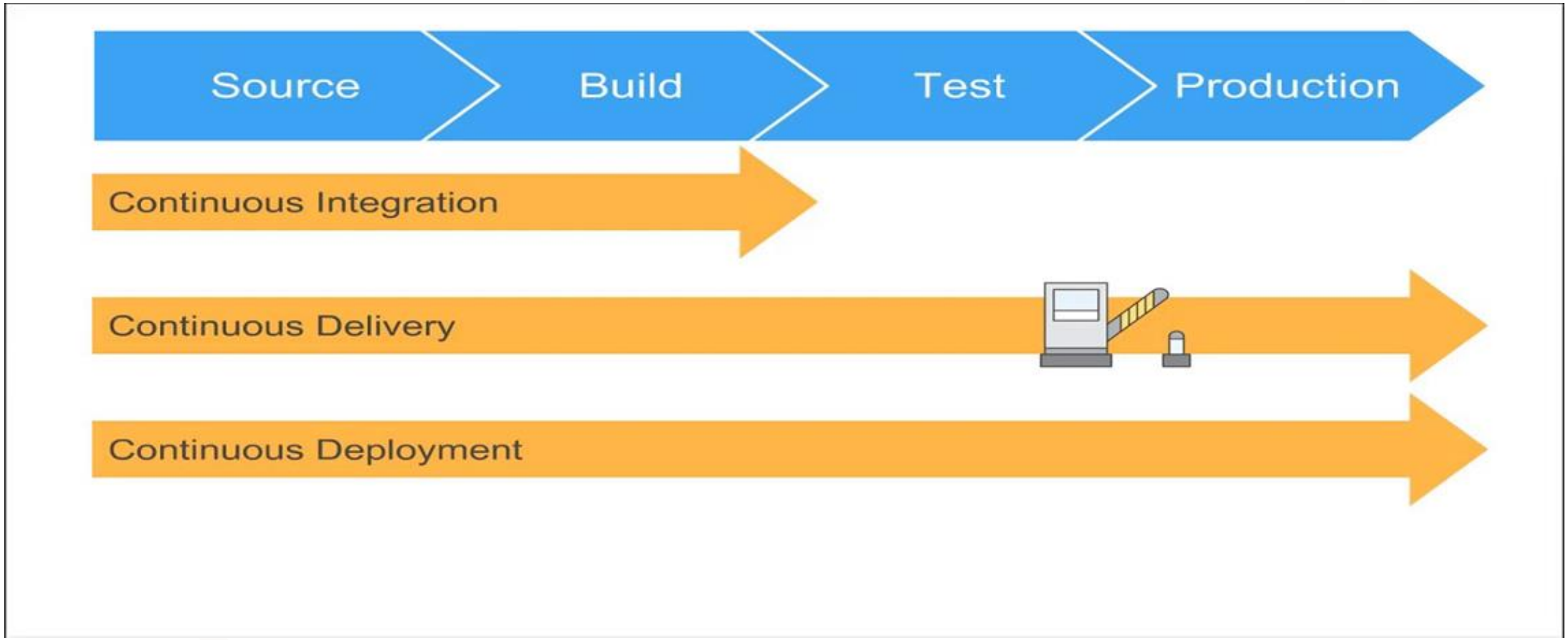
What is Continuous Integration?



CI/CD pipeline overview



CI/CD pipeline overview



CI/CD pipeline overview

What is Continuous Integration

- In Simple term Continuous Integration is a term which means “checking the compatibility of a change you have made with the remaining code base or modules of an application or to check the impact of that change on the application functionality”.
- Continuous Integration is not helping us in fixing the bugs but it helps us to Identify those bugs in the early phases of a development lifecycle and with much faster rate.
- Continuous Integration is not the sole responsibility of the developers but every individual working in the team is responsible for the Healthy Continuous Integration.

Why Continuous Integration

- CI helps in reduction of efforts which will increase exponentially with the increase in:
  - Number of components
  - Number of Bugs
  - Parallel development
  - Time from last Integration

Benefits

- Project Management
  - Detect system development problems in earlier stages of development
  - Reduce risks of increase in cost or budget
  - Reduce the risk of missing delivery timelines
- Code Quality
  - Measurable and visible code quality
  - Continuous automatic regression and unit test
  - Visible code coverage
- Automation
  - Automated pipelines
  - Early time to reach market
- Monitoring
  - Complete Visibility to projects
  - Time to react to changes
- And much more . .

Jenkins History

- Jenkins was originally developed as the Hudson project. Hudson's creation started in summer of 2004 at Sun Microsystems. Kohsuke Kawaguchi, the current CTO of CloudBees, "*A Company providing Enterprise Level Support for Jenkins, when he was working with Sun Micro Systems*"

- Certain clashes between Sun Micro Systems and Hudson Community for the Management of the Project

- Today Jenkins is the most used Continuous Integration tool in the market, which provide variety of functions to run with the help of large Jenkins Community.

Brief about Jenkins

- An Open Source CI Server with MIT (Massachusetts Institute of Technology)Licensing
  Large Community Base
- More than 300,000 installations on record
- More than 1300 plug-in support which make to compatible with almost every delivery tool available in market
- Annual meet-up with the Jenkins World
- CloudBees is the biggest promoter and Contributor of Jenkins Community and also provide Enterprise Support for Jenkins
- *Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and deploying software.*

Create our first project

- Install Jenkins
- *Run Jenkins as root user by modifying /etc/sysconfig/Jenkins file*
- *Create your first project in Jenkins*
- *Understand plugins and global configuration*
- *Install git and integrate github with Jenkins*
- *Install Docker and integrate docker with Jenkins*
- *Create a pipeline to build a docker image, run containers from the image and all code to be pulled for github repo*

SonarQube – Software testing - SAST

What is Software Testing?

What is Software Testing?

Software Testing is a part of Software Development Lifecycle, which is aimed to ensure that code to be deployed is of high quality and standards, with no bugs, logical errors and issues.

It is always better to spend a little in early stages on testing than spending a lot at a later stage.

Code should be clean and reusable and have security in place.

SAST vs DAST

Sr No	Static Application Security Testing (SAST)	Dynamic Application Security Testing (DAST)
1	White Box Security Testing	Black Box Security Testing
2	Source Code Is Required	A Runtime Application is required
3	Vulnerabilities Found Earlier in SDLC	Vulnerabilities Found Later in SDLC
4	Less Expensive to Fix	More Expensive to Fix
5	Unable to Identify Timing- and Environment-Related Issues	Can Identify Run-Time and Environment-Related Issues
6	Generally Supports all Kinds of Software	Typically only scans Web applications and Services

Few Other benefits of SAST

- Detects Overcomplexity in the code
- Smells the code for future failures/issues
- Enforces Best Coding Practices
- Project Specific Rules can be created
- Avoid Technical Debts

SonarQube

SonarQube (formerly Sonar) is an open-source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on multiple programming languages.

It offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities.

27+ Languages supported.

SonarQube can record metrics history and provides evolution graphs.

Fully automated analysis and integration with Maven, Ant, Gradle, MSBuild and continuous integration tools (Bamboo, Jenkins, Azure DevOps, Hudson, etc.).

Open to all tools via WebAPI and Webhooks.

SonarQube

SonarQube is available for free under the GNU Lesser General Public License. An enterprise version for paid licensing also exists, as well as a data center edition that supports high availability.

SonarQube integrates with Eclipse, Visual Studio, and IntelliJ IDEA development environments through the SonarLint plug-ins

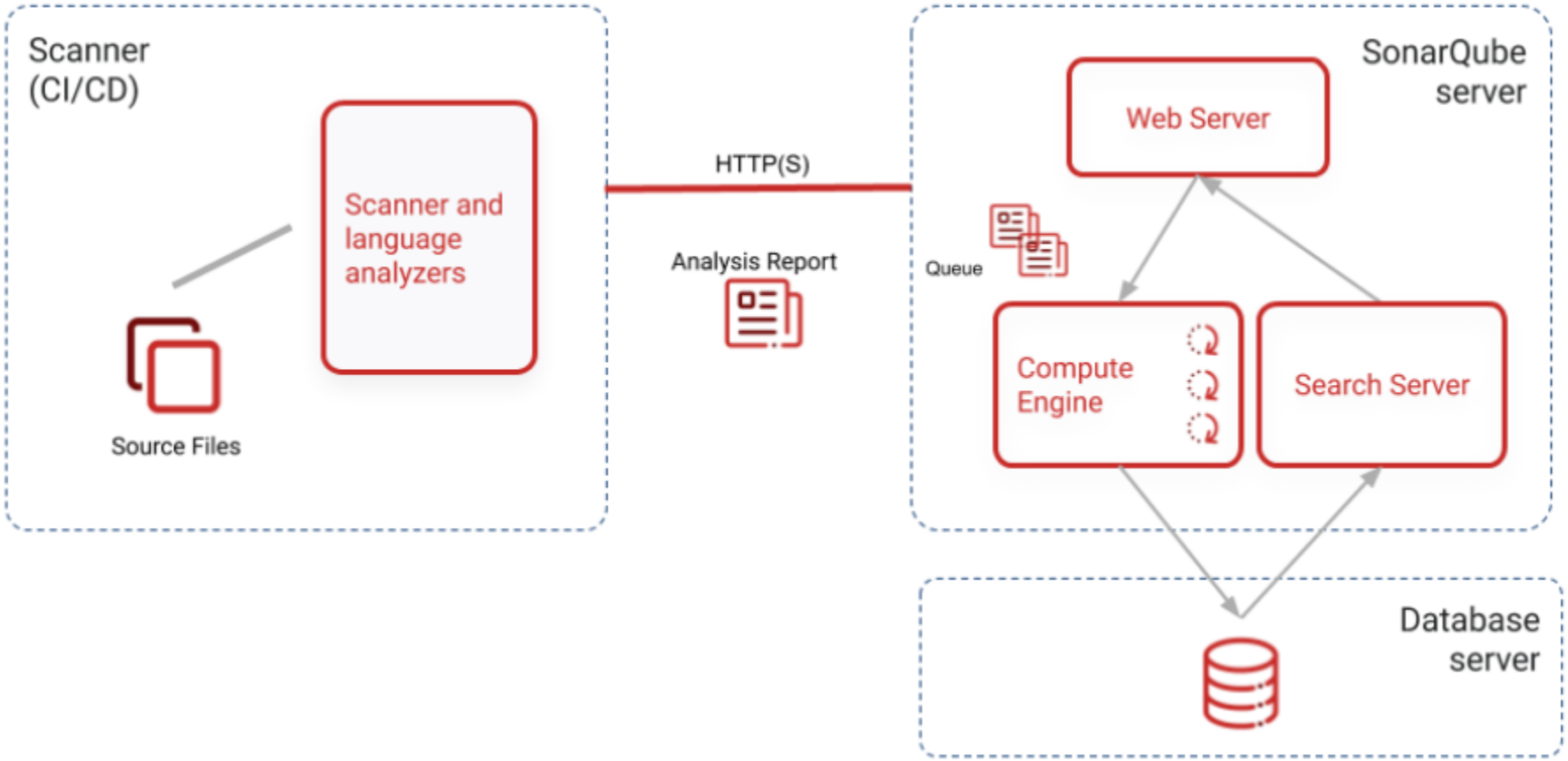
Analyzes all branches of a VCS

Discover Memory Leaks

Quality gates and Quality profiles for customized requirements

Good Visualization.

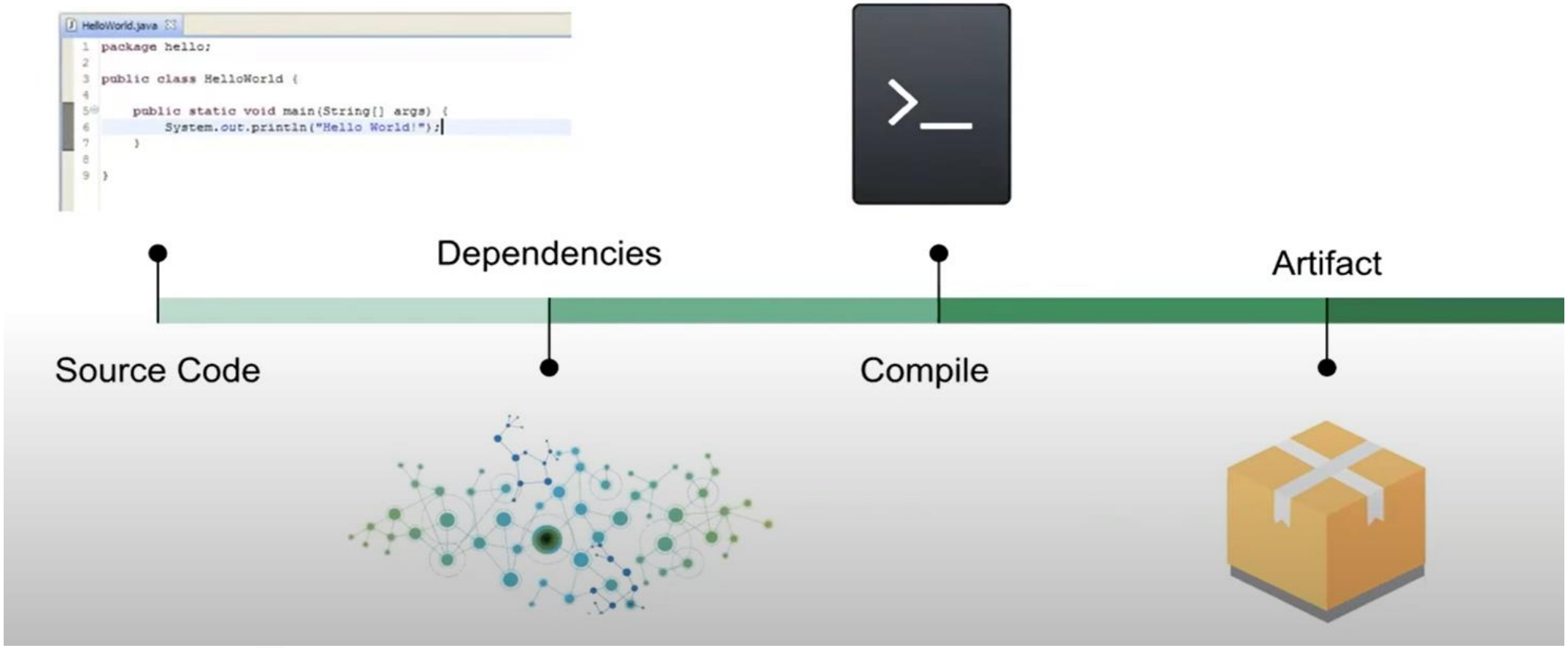
SonarQube Components



https://docs.sonarqube.org/latest/setup/install-server/

Nexus –repo manager| A Binary Artifacts manager

Code to Release



Artifact Management

To Version control your artifacts

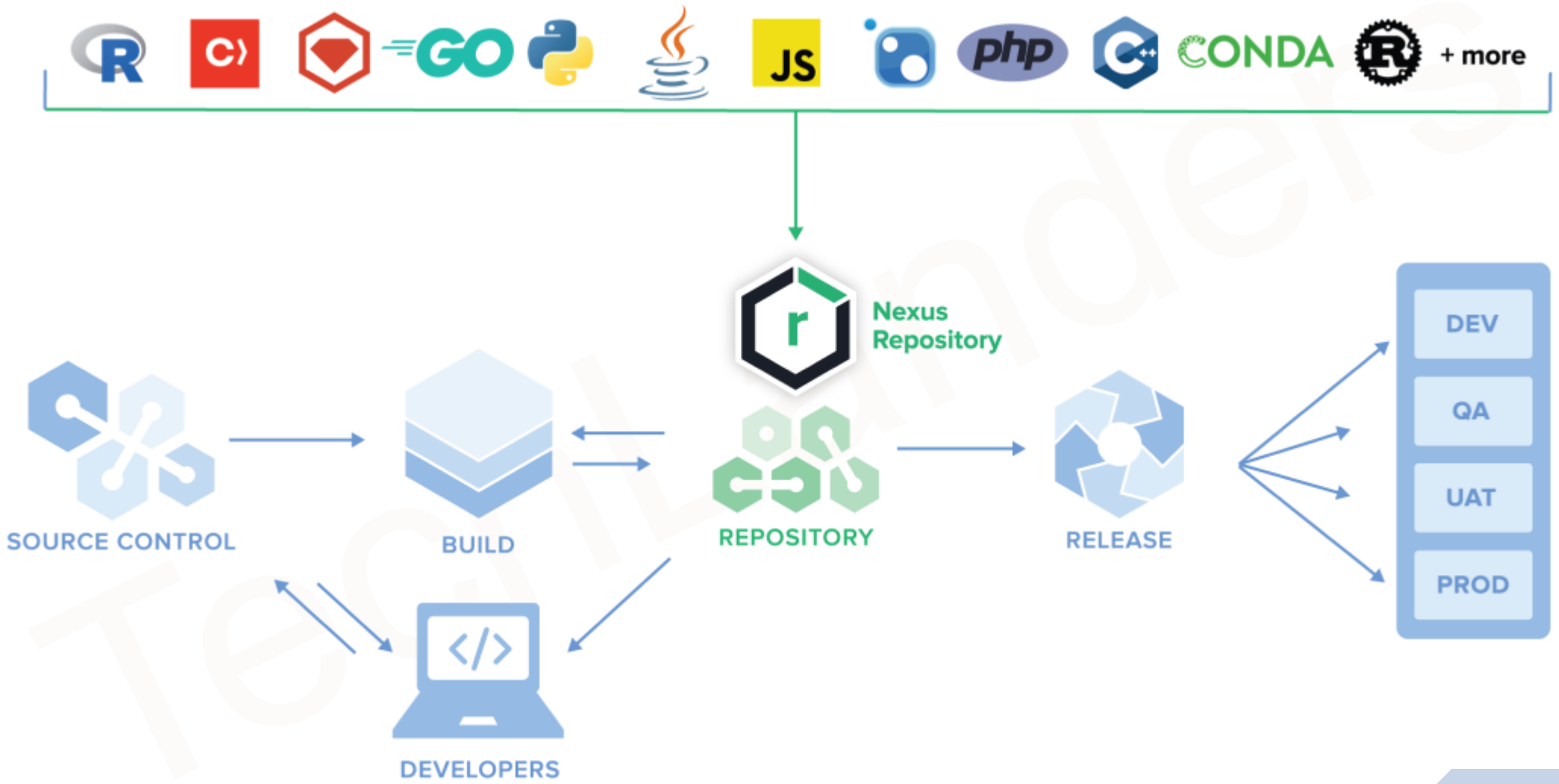
Manage them at remote Central Location

Manage Dependencies and Artifacts together

Binary Repository Manager: A collection of binary software artifacts and metadata, stored in a way which can be used by clients, package managers and CI Servers, to retrieve/store binaries during the development and build process.

Why not VCS?

Source Code Managers (VCS)	Binary/Artifacts Repository Management
Text	Binary
Diffable	Not Diffable
Versioned By content	Versioned by Name
Mutable	Immutable


+ more
Nexus Repository
SOURCE CONTROL
BUILD
REPOSITORY
RELEASE
DEV
QA
UAT
PROD
DEVELOPERS

Containerization

Raman

Agenda

- Introduction
- Docker Components
- Classroom Environment
- Containers
- Docker – Images
- Docker - Building Images
- Deep Dive – Images
- Deep Dive – Containers
- MicroServices Example
- Container Network Model
- Docker Volumes

Session: 1

Introduction

What is Container?

Before proceeding with the new terms and technologies lets have a look on the history/traditional approaches used for the application since ages.

A History Lesson

- The problem got the solution by a technology called “Hypervisor-Based Virtualization”.
- One physical server can contain multiple running applications.
- Each application need a VM to run the application binaries.

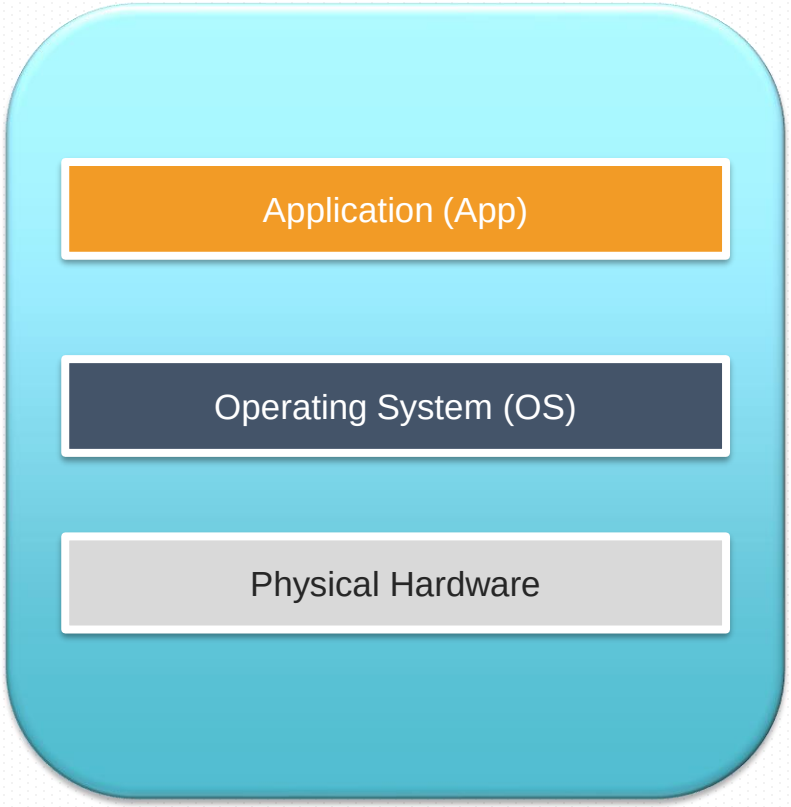
A History Lesson

- In the traditional ages of development and deployment of application

Unused Resources

Difficult Migrations

Vendor dependency



Slow Deployment time

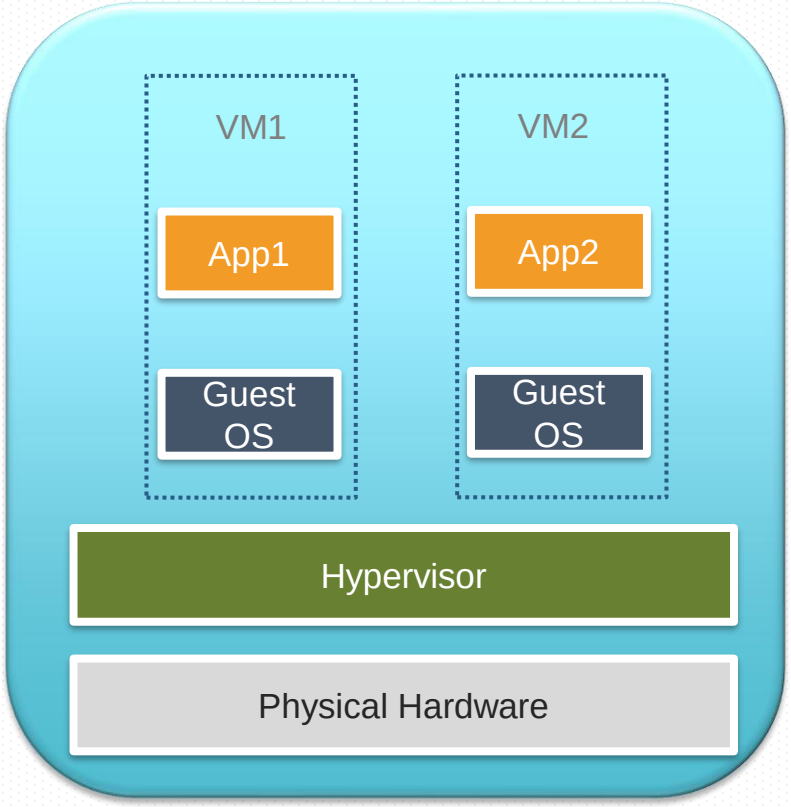
Difficult to Scale

Huge cost in Infrastructure

Traditional Approach

Virtualization

- OverHead
- Portability issues
- Boot-time still in Minutes
- Scaling issue in Hybrid Env
- Migrations still failing
- Costly Solution



Server Virtualization

- Better Resource Pooling
- Easier to Scale
- Flexibility & Easy Migration
- Faster Deployments
- Faster Boot time

Containerization

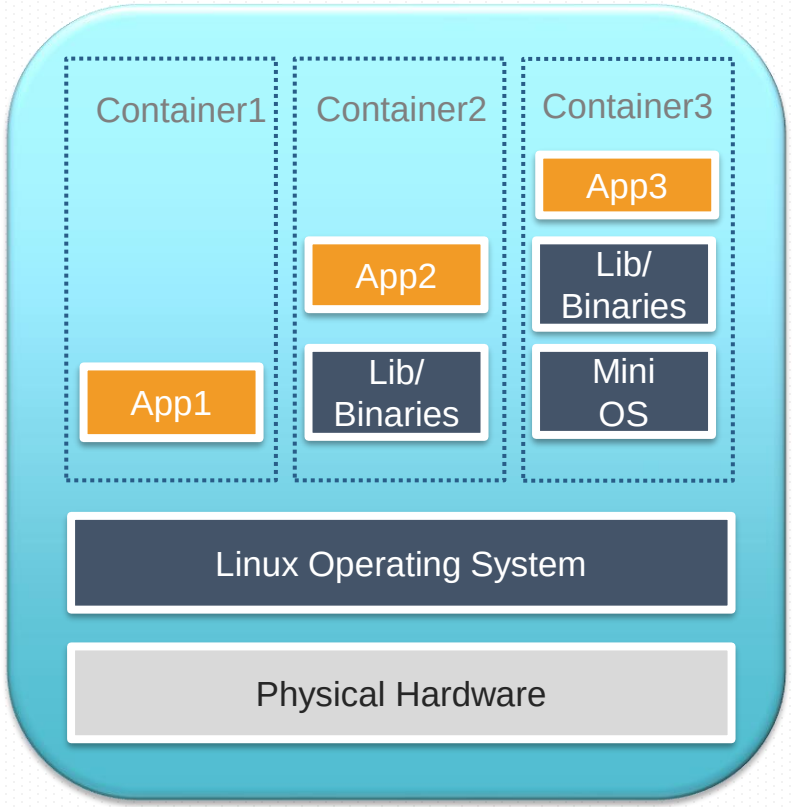
Less OverHead

Highly Portable

Scaling in Hybrid Env

High Migrations success ratio

Cost Effective Solution



Containerization

Much Better Resource Pooling

Extended Scaling

Flexibility & Easy Migration

Faster Deployments (Seconds)

Faster Boot time (Seconds)

Introducing Containers

- Container based virtualization uses the kernel on the host's operating system to run multiple guest instances

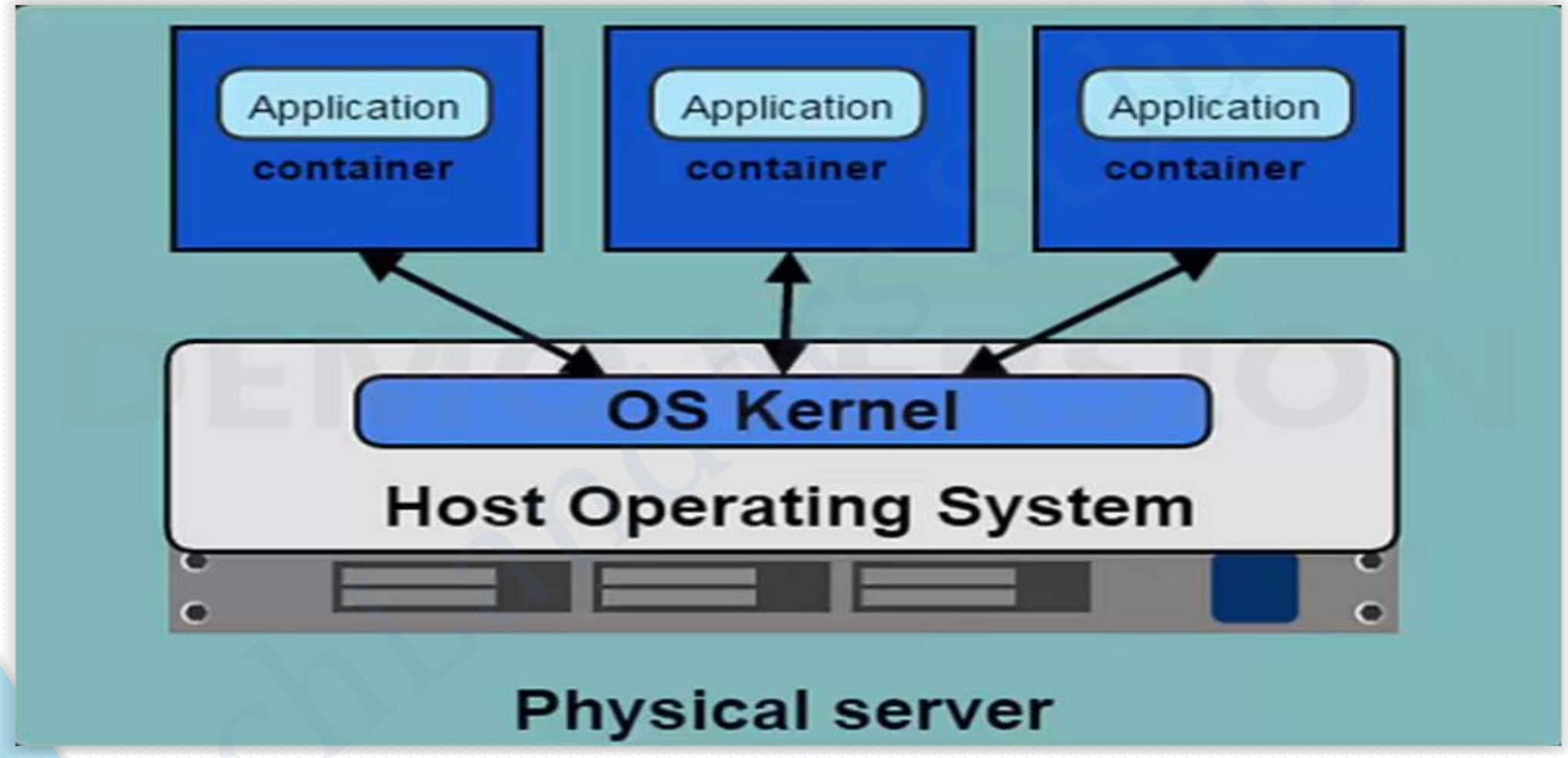
- Each guest instance is called a "Container"

- Each container has its own

- Root Filesystem
- Processes
- Memory
- Devices
- Network Ports

- From outside its looks like a VM but its not a VM

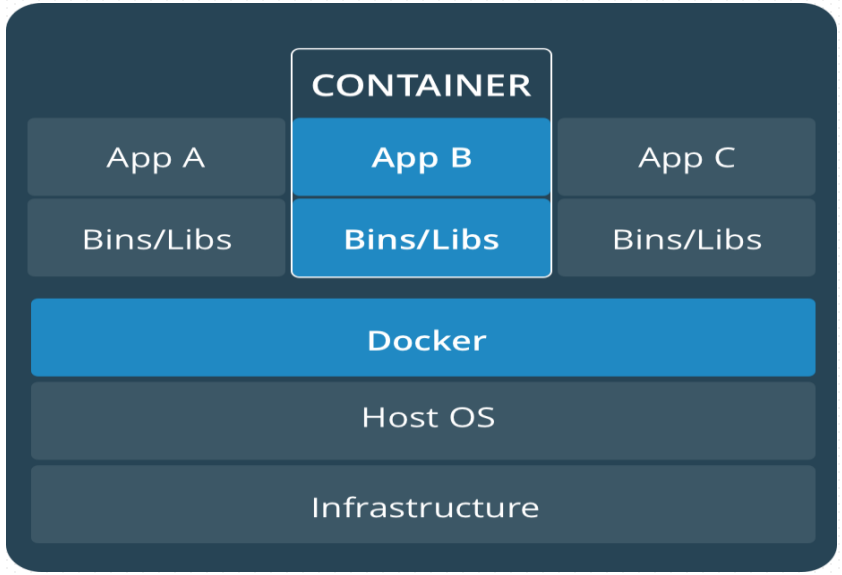
Overview of Containers

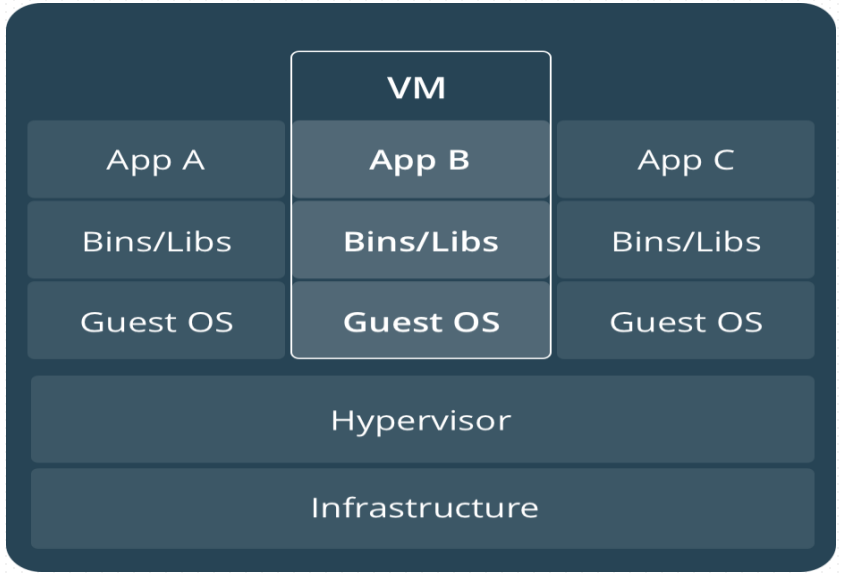


Containers VS VM's

- Container are more light weight
- No need to install dedicated guest OS, no virtualization like VM is required
- Stop/Start time is very fast
- Less CPU, RAM, Storage Space required
- More containers per machine than VM's
- Great Portability

Containers VS VM's

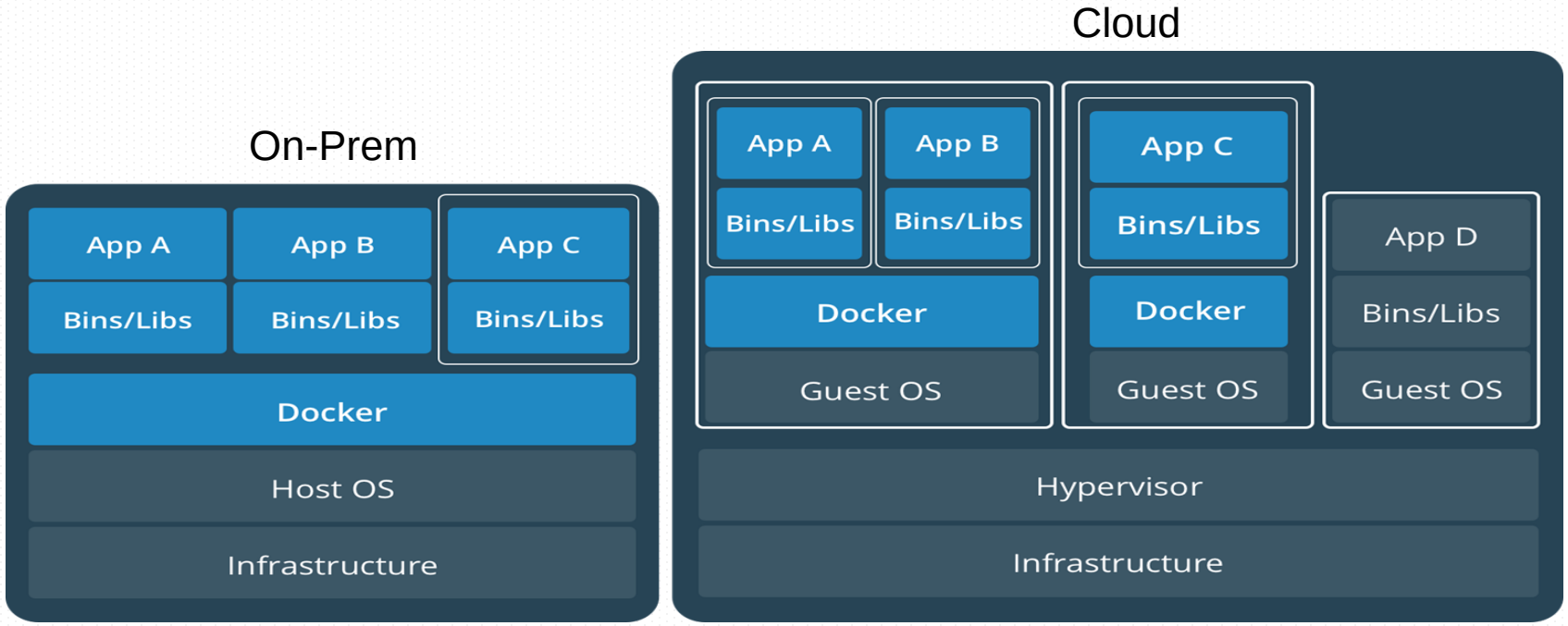




Containers are an app level construct

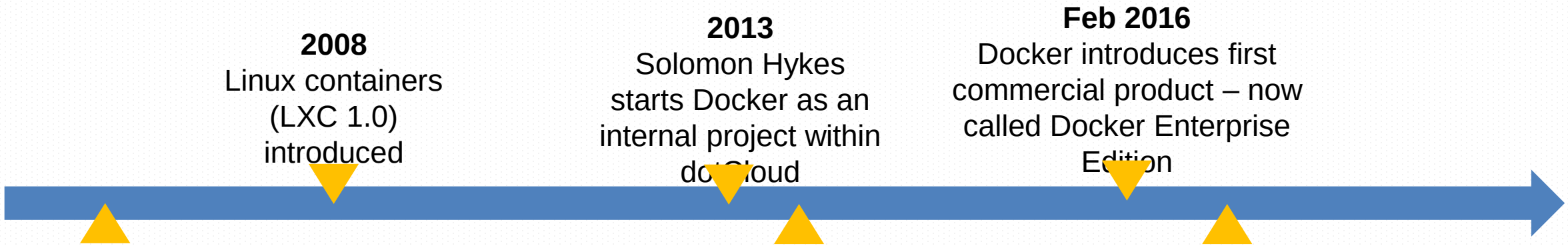
VMs are an infrastructure level construct to turn one machine into many servers

Containers and VM's together



Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

Origins of Docker Project



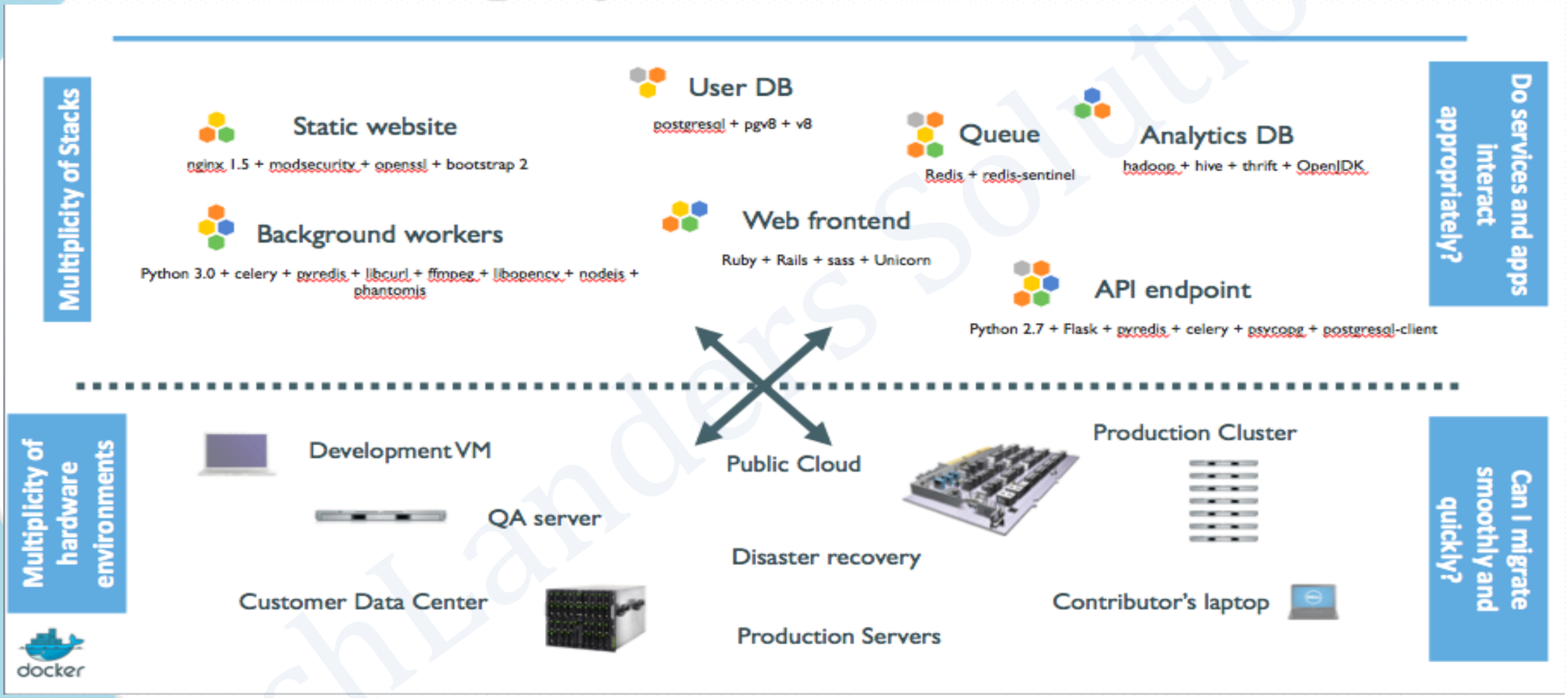
Origins of Docker Project

- ‘dotCloud’ was operating a PaaS platform, using a custom container engine.
- This engine was based on ‘OpenVZ’ (and later, LXC) and AUFS.
- It started (circa 2008) as a single Python script.
- By 2012, the engine had multiple (~10) Python components. (and ~100 other micro-services!)
- End of 2012, ‘dotCloud’ refractors this container engine.
- The codename for this project is "Docker."

About Docker Inc.

- Docker Inc. Formerly ‘dotCloud’ Inc, used to be a French company
- Docker Inc. is the primary sponsor and contributor to the Docker Project:
  - Hires maintainers and contributors.
  - Provides infrastructure for the project.
  - Runs the Docker Hub.
- HQ in San Francisco.
- Backed by more than 100M in venture capital.

Deployment Problem

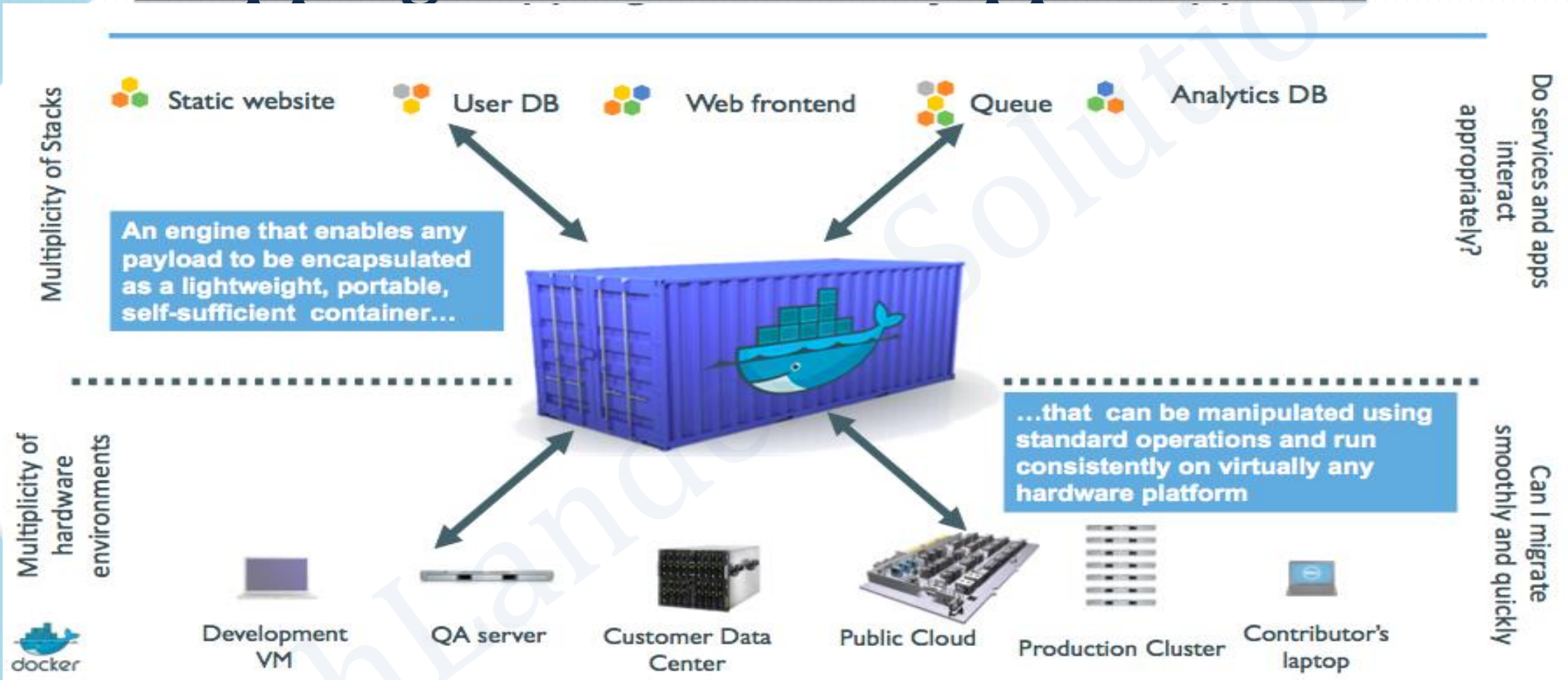


Matrix Checks

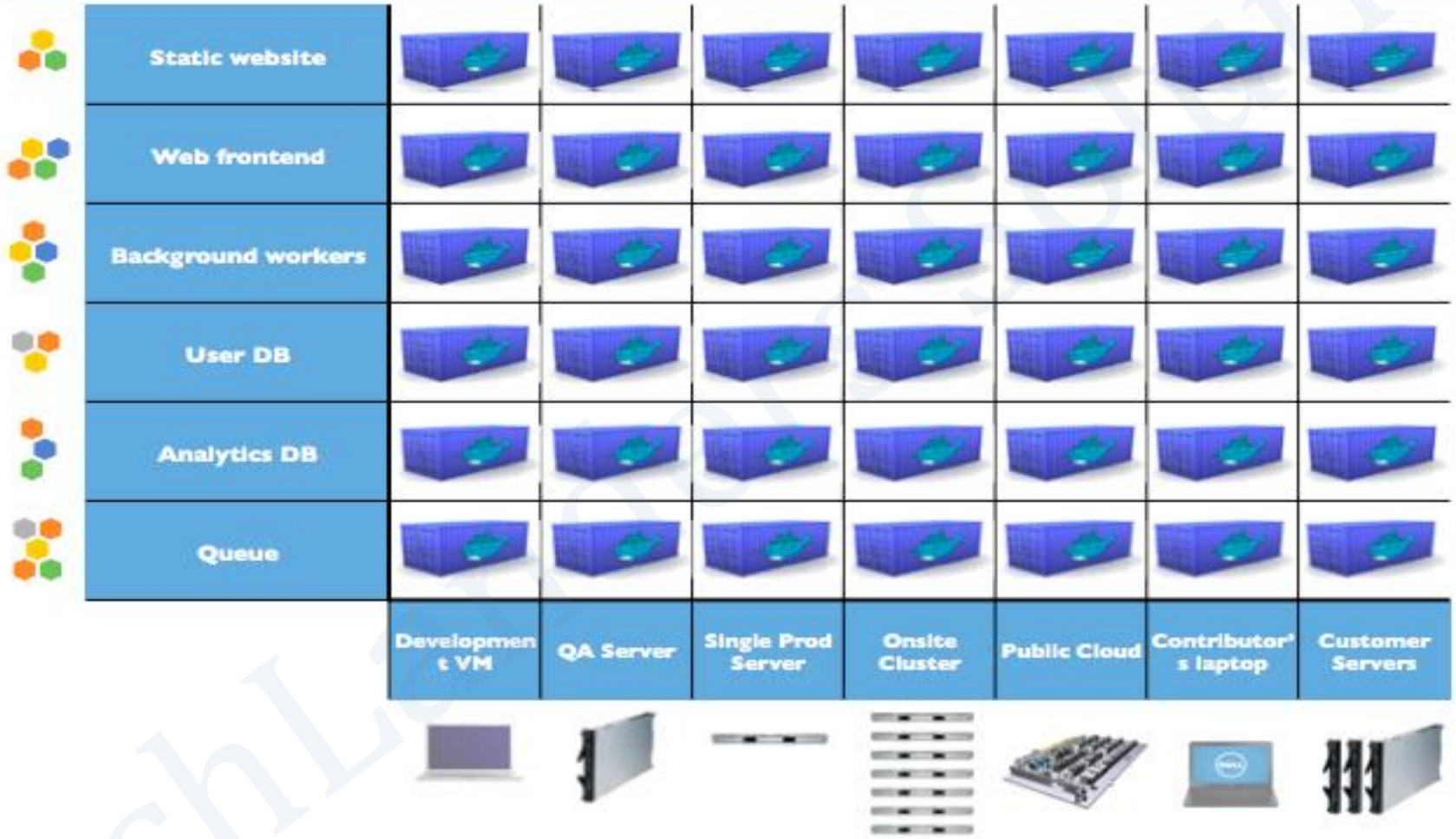
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Intermodal Shipping Containers

Shipping Container for Applications



Eliminate the Matrix



Results

Speed

- No OS to boot = applications online in seconds

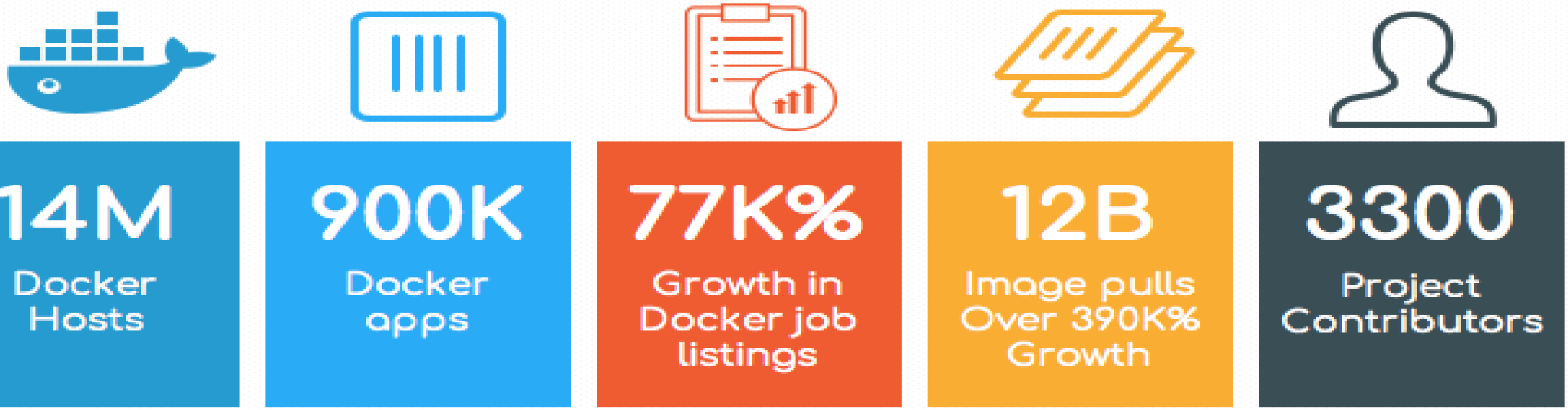
Portability

- Less dependencies between process layers = ability to move between infrastructure

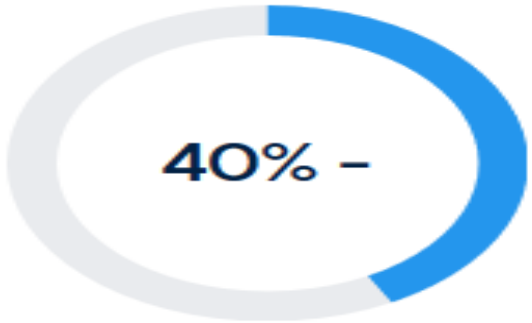
Efficiency

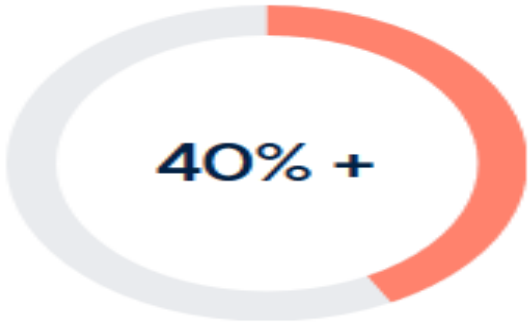
- Less OS overhead
- Improved VM density

Adoption in Just 4 years



Why Docker?





Session: 2

Docker Components

Docker Overview

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- By using Docker's methodologies for shipping, testing, and deploying, you can reduce time of customer delivery.

Docker Platform

- 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.

- Because of the **lightweight** nature of containers, which run without the extra load of a hypervisor, you can run more containers on a given hardware combination than if you were using virtual machines.

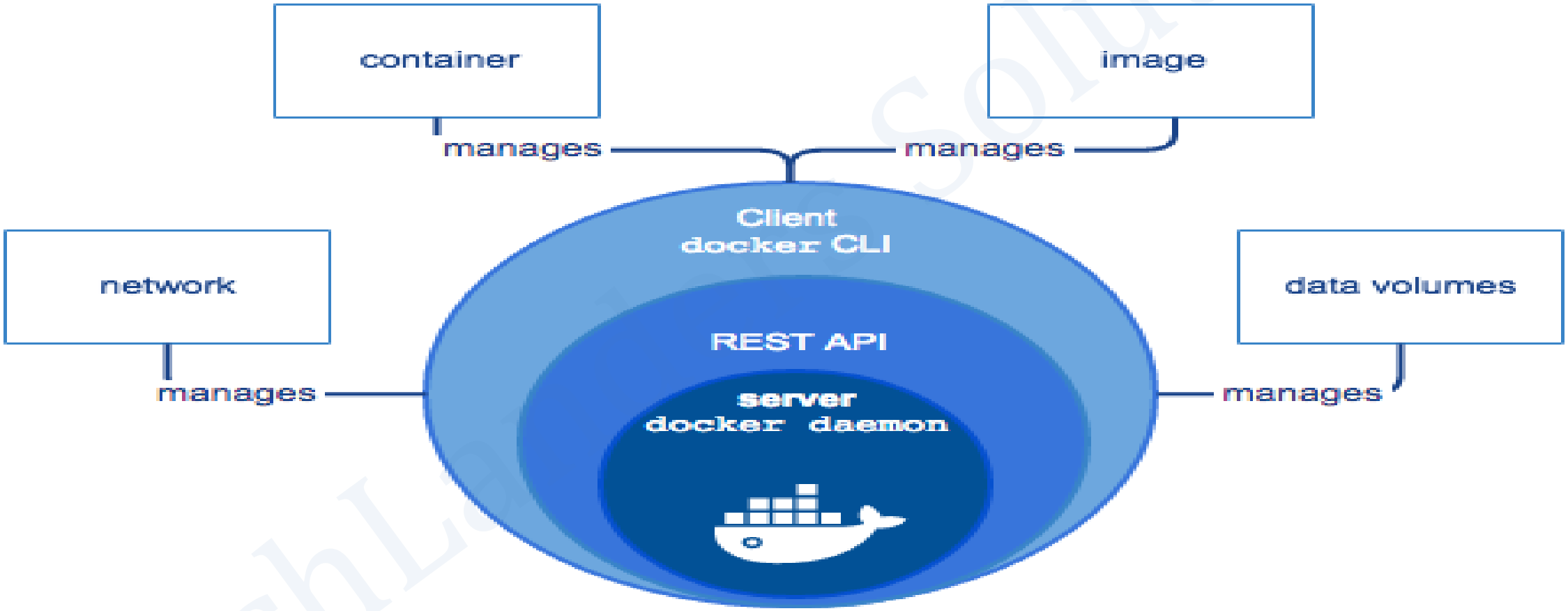
Docker Test

- docker --version
- Test the docker functioning: docker run hello-world

Docker Platform

- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Encapsulate your applications (and supporting components) into Docker containers
  - Distribute and ship those containers to your teams for further development and testing
  - Deploy those applications to your production environment, whether it is in a local data center or the Cloud

Docker Engine



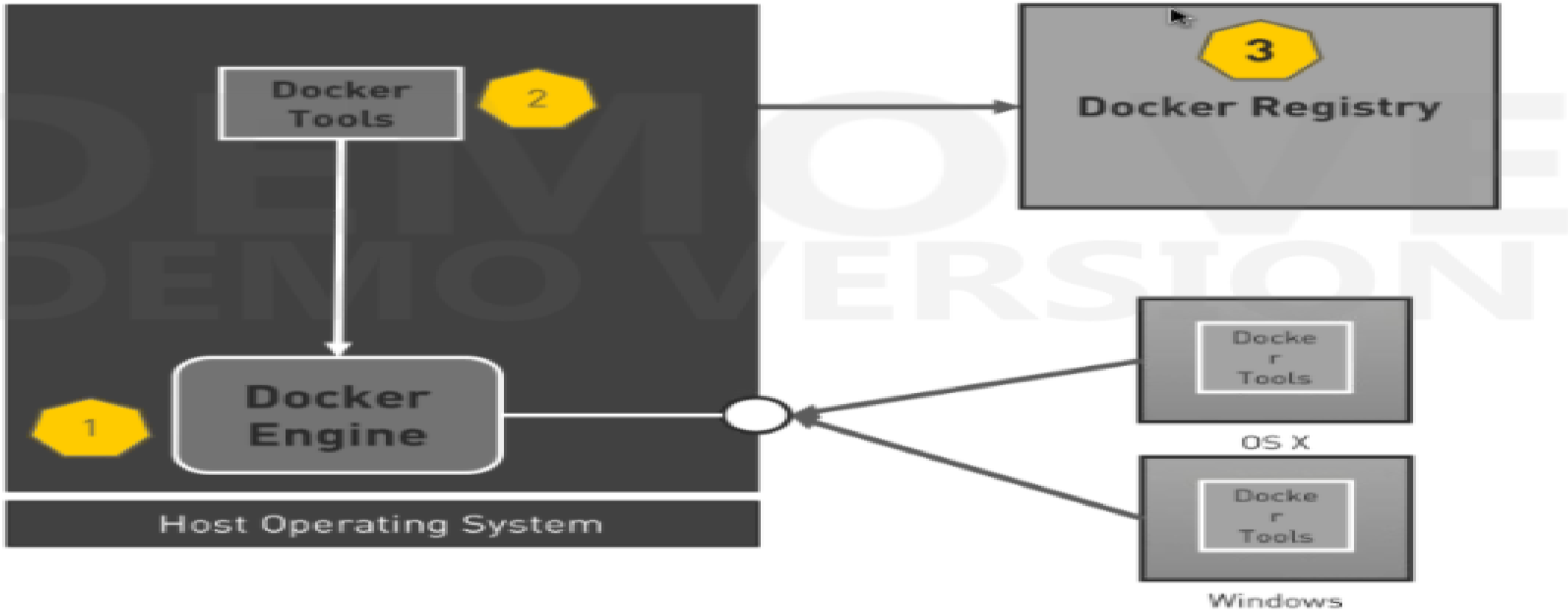
Docker Engine

- Docker Engine is a client-server application with these major components:
  - A server which is a type of long-running program called a daemon process.
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  - A command line interface (CLI) client.

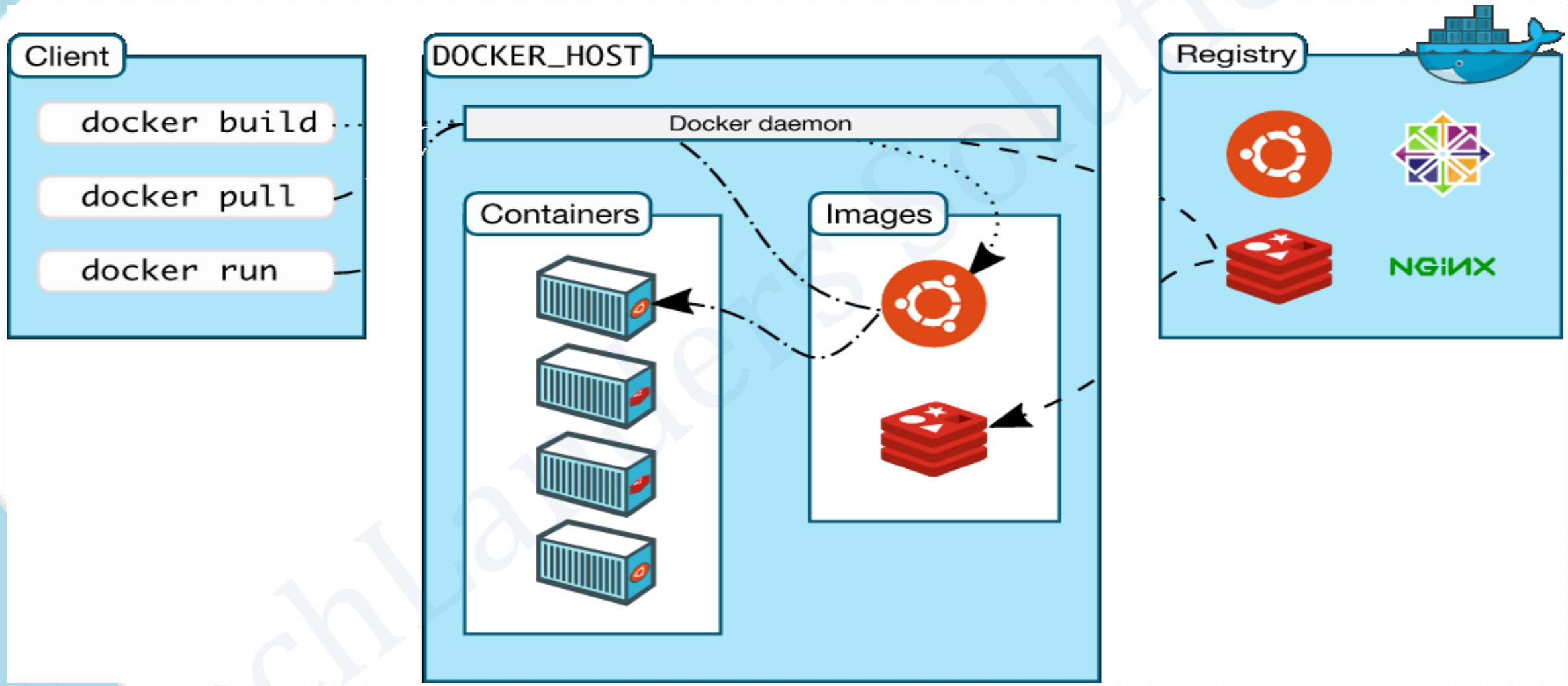
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands

- The daemon creates and manages Docker objects, such as images, containers, networks, and data volumes

Docker Architecture



Docker Architecture



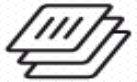
Docker Architecture

- 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.

Docker Architecture

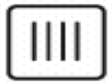
- The Docker Daemon
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- The Docker Client
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker.
  - It accepts commands and configuration flags from the user and communicates with a Docker daemon.

Docker Architecture



Image

The basis of a Docker container. The content at rest.



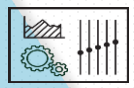
Container

The image when it is ‘running.’ The standard unit for app service

Engine

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

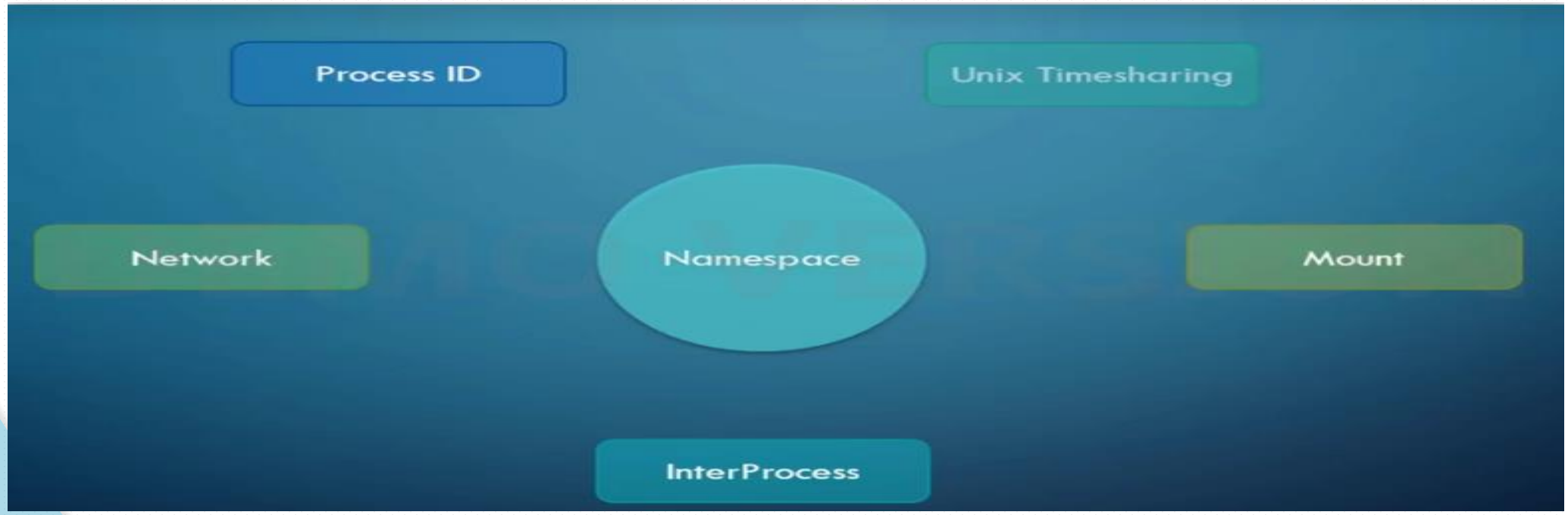
Registry



Stores, distributes and manages Docker images

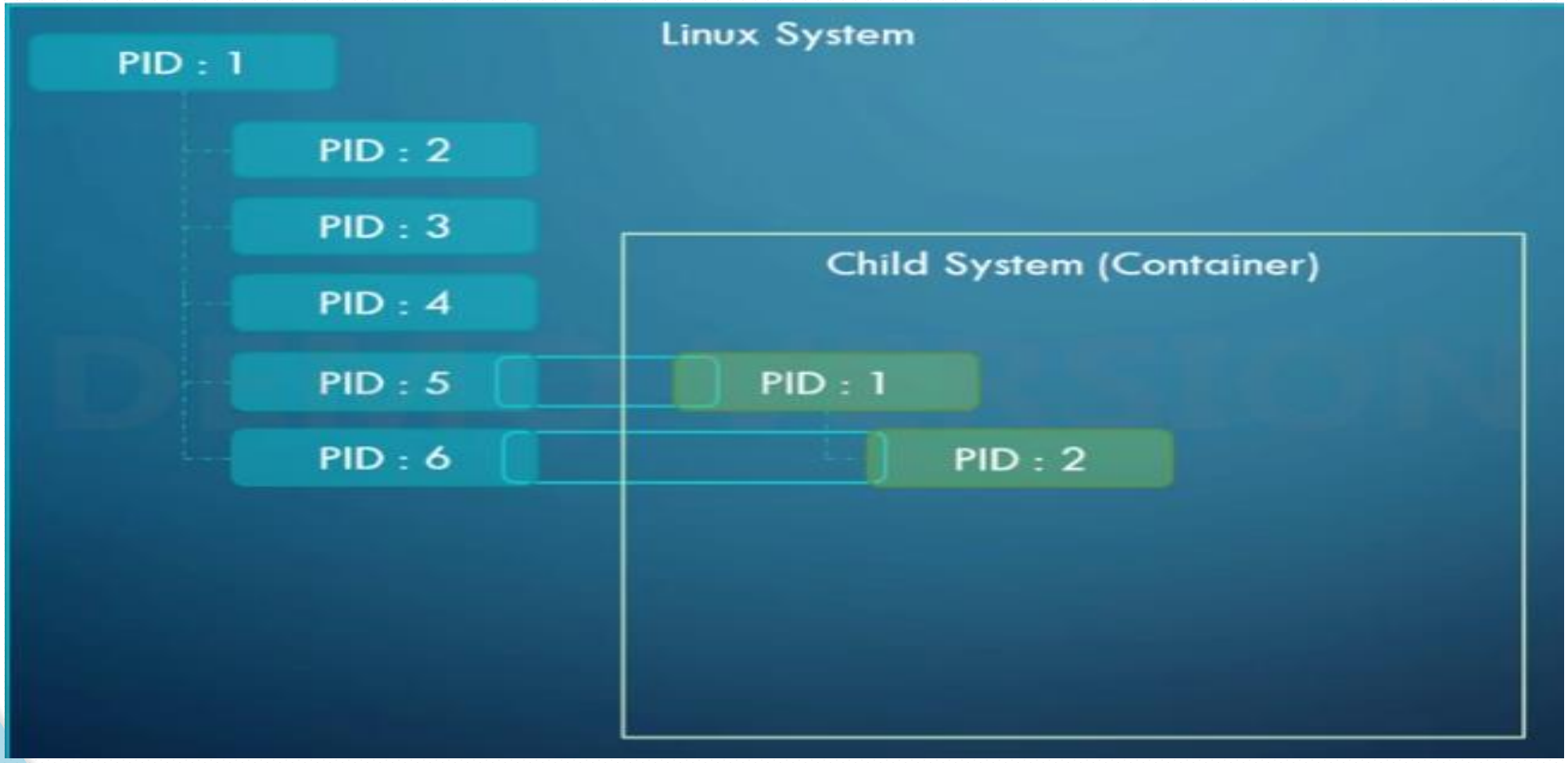
Docker Architecture

- Lets understand how does applications works in isolation under the hood.
- Docker uses namespace which is a nothing but an isolated environment like VM, but on top of VM called container.



Docker Architecture

- The demonstration will be given once we understand the container operations.



Docker Images

- A Docker image is a read-only template with instructions for creating a Docker container.
- For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others.
- A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.
- Docker images are the build component of Docker.

Docker Containers

- A Docker container is a running instance of a Docker image.
- You can run, start, stop, move, or delete a container using Docker API or CLI commands.
- When you run a container, you can provide configuration metadata such as networking information or environment variables.
- Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases.
- Docker containers are the run component of Docker.

Docker Registries

- A docker registry is a library of images.
- A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server.
- Docker registries are the distribution component of Docker.
- “Docker Hub” is known as global registry.

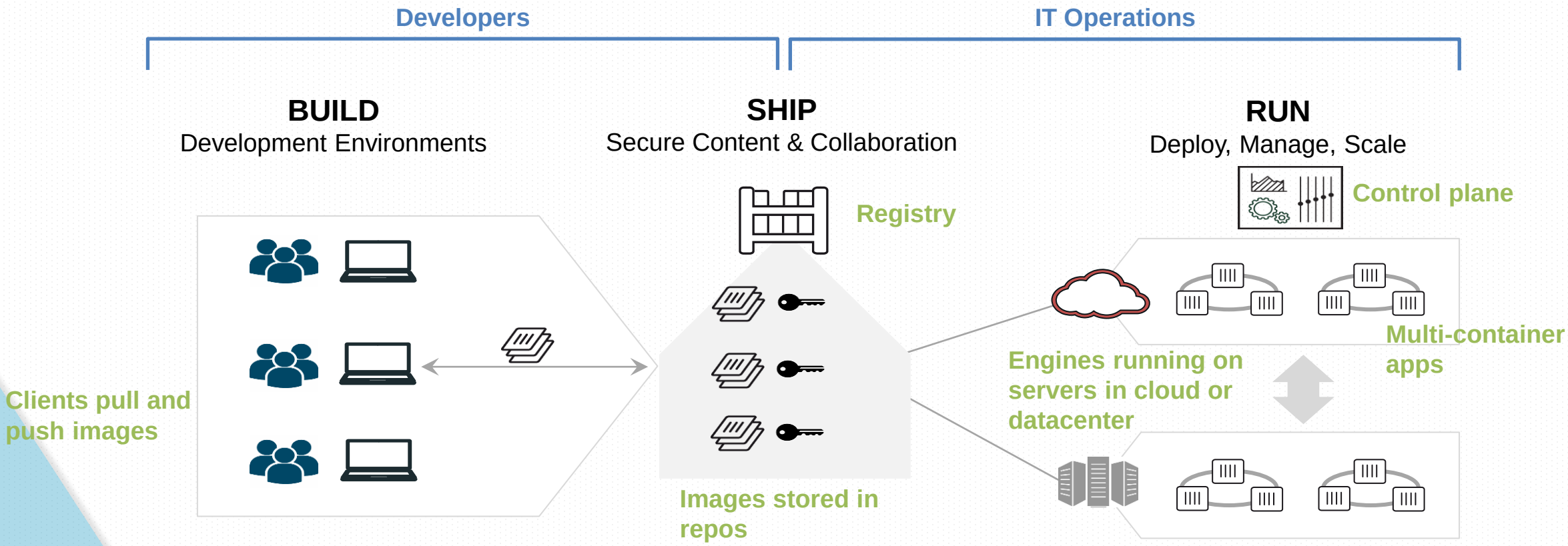
Docker Features

- Lightweight
  - Containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.

- Open
  - Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.

- Secure
  - Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application.

Container as a Service



Session: 3

Environment

Docker Engine Install Demo

- Docker Engine/Client would be installed on Training Environment as demo LAB.
- https://docs.docker.com/engine/installation/linux/centos/
- https://www.cyberciti.biz/faq/how-to-install-docker-on-amazon-linux-2/

Docker Installation Key Points

- docker.service Systemd File:

[root@TechLanders lib]# more /usr/lib/systemd/system/docker.service

- Docker Socket file:

[root@TechLanders lib]# file /var/run/docker.sock
/var/run/docker.sock: socket

- Docker PID file and Docker Container PID file (This is Docker Daemon which will have pid1)

root@TechLanders run]# cat /var/run/docker.pid
3715
[root@TechLanders libcontainerd]# cat /var/run/docker/containerd/docker-containerd.pid
4251

- Docker Detailed Information:

[root@TechLanders libnetwork]# docker info

Manage Docker as a non-root user

- The docker daemon binds to a Unix socket instead of a TCP port.
- By default that Unix socket is owned by the user ”root” and other users can only access it using sudo.
- The docker daemon always runs as the root user.
- If you don’t want to use sudo when you use the docker command, add users to Unix group called ”docker” example: usermod -aG docker <user-name>
- When the docker daemon starts, it makes the ownership of the Unix socket read/writable by the docker group.

Session: 4

Containers

How Container works

- A container uses the host machine's Linux kernel, and consists of any extra files you add when the image is created, along with metadata associated with the container at creation or when the container is started.
- Each container is built from an image.
- The image defines the container's contents, which process to run when the container is launched, and a variety of other configuration details.
- The Docker image is read-only. When Docker runs a container from an image, it adds a read-write layer on top of the image (using a UnionFS) in which your application runs.

How Container works

- When you use the ”docker run” CLI command, the Docker Engine client instructs the Docker daemon to run a container.

```
docker run ubuntu ps ax
```

- This example tells the Docker daemon to run a container using the centos Docker image, to remain in the foreground in interactive mode (-i), provide a tty terminal (-t) and to run the /bin/bash command.

```
docker run -i -t centos /bin/bash
```

Because if you exit the current running process /bin/bash (pid 1), container will stop/exit. Use “Ctrl pq” to safe exit without stopping the container.

How Container works

```
[root@TechLanders libcontainerd]# docker run -i -t centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
d9aaf4d82f24: Pull complete
Digest: sha256:eba772bac22c86d7d6e72421b4700c3f894ab6e35475a34014ff8de74c10872e
Status: Downloaded newer image for centos:latest
[root@9a06b1a61fc5 /]#
```

```
[root@TechLanders overlay]# ls -lrt /var/lib/docker/image/overlay2/repositories.json
-rw-------. 1 root root 545 Sep 18 03:17 /var/lib/docker/image/overlay2/repositories.json
```

How Container works

[root@TechLanders overlay]# cat repositories.json

{"Repositories":{"centos":{"centos:latest":"**sha256:196e0ce0c9fbb31da595b893dd39bc9fd4aa78a474bbdc21459a3ebe855b7768**","centos@sha256:**eba772bac22c86d7d6e72421b4700c3f894ab6e35475a34014ff8de74c10872e**":"sha256:196e0ce0c9fbb31da595b893dd39bc9fd4aa78a474bbdc21459a3ebe855b7768"},"hello-world":{"hello-world:latest":"sha256:05a3bd381fc2470695a35f230afefd7bf978b566253199c4ae5cc96fafa29b37","hello-world@sha256:1f19634d26995c320618d94e6f29c09c6589d5df3c063287a00e6de8458f8242":"sha256:05a3bd381fc2470695a35f230afefd7bf978b566253199c4ae5cc96fafa29b37"}}}

[root@TechLanders overlay]# docker image list

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
centos	latest	**196e0ce0c9fb**	3 days ago	197MB
hello-world	latest	05a3bd381fc2	5 days ago	1.84kB

[root@TechLanders overlay]# docker image inspect centos

More Useful - Container

- Do something in our container:

  Lets suppose we try to use “talk” for communication.

- Let’s check how many packages are installed:

  rpm –qa | wc –l

A non interactive - Container

- In your Docker environment, just run the following command:

  docker run jpetazzo/clock

- This container just displays the time every second.
- This container will run forever.
- To stop it, press ^C.
- Docker has automatically downloaded the image jpetazzo/clock.

Run in background - Container

- Containers can be started in the background, with the -d flag (daemon mode):

  docker run -d jpetazzo/clock

- We don't see the output of the container.
- But don't worry: Docker collects that output and logs it!
- docker ps -a
- docker logs <conatiner-id>
- Docker gives us the ID of the container.

List Running Containers

- With docker ps, just like the UNIX ps command, lists running processes.

  docker ps
  docker ps –l
  docker ps –a
  docker ps -q

- The (truncated) ID of our container.
- The image used to start the container.
- That our container has been running (Up) for a couple of minutes.
- Now, start multiple containers and use “docker ps” to list them.

Stop our Container

- There are two ways we can terminate our detached container.
  - Killing it using the docker “kill” command.
  - Stopping it using the docker “stop” command.
- The first one stops the container immediately, by using the KILL signal.
- The second one is more graceful. It sends a TERM signal, and after 10 seconds, if the container has not stopped, it sends KILL.

Removing Container

- Let’s remove our container:

```
docker rm <yourContainerID>
```

Attaching to a Container

- You can attach to a container:

  docker attach <containerID>

- The container must be running.

- There *can be multiple clients attached to the same container.*

Restarting a Container

- When a container has exited, it is in stopped state.
- It can then be restarted with the “start” command.

  docker start <containerID>

- The container must be running.
- You can also use restart command

  docker restart <container-id>

LAB1

- Create one Ubuntu Container in interactive and terminal mode
- Exit out of the container
- Check the status of container
- Restart the container
- Get attached to the container
- Get out of container without exiting „ stop the container
- Remove the container
- Create the container now in detached mode and follow the same steps ..

LAB2

- Create a new nginx container using (-d) option.
- Docker run –d nginx
- Now check the difference using docker ps –a command
- Try to access this container
- Docker attach <container-ID>

  cat /etc/os-release (THIS WILL NOT WORK) (BECAUSE WE ARE NOT USING INETRACTIVE IN THE CREATION)
- Use docker exec -it <CONATINER-ID> /bin/bash
- Remove everything
- Docker stop
- Docker rm
- Docker run –dt nginx (YOU NEED TO PROVIDE A TERMINAL HERE) → THIS IS TRUE WITH BASE IMAGES
- Docker attach <container-ID>

  cat /etc/os-release

LAB3

- *On your host machine install httpd package*

  #For centos

  – Yum install httpd –y

  #For ubuntu :

  -- apt-get install apache2

- *Verify installation:*

  rpm –qa | grep –i httpd

  #For ubuntu

  find / -name apache2 or dpkg –list | grep –i apache2

- *Create a container :*
- #For ubuntu :

docker run –it –name c1 ubuntu

docker run –it centos

Learning - Containers have isolated independent root filesystem

LAB3 cont..

Check the package inside it:

dpkg --list | grep -i apache

*Install the package if not there :
apt-get update –y
apt-get install apache2
dpkg --list | grep apache2

#For centos :
rpm –qa | grep –i httpd

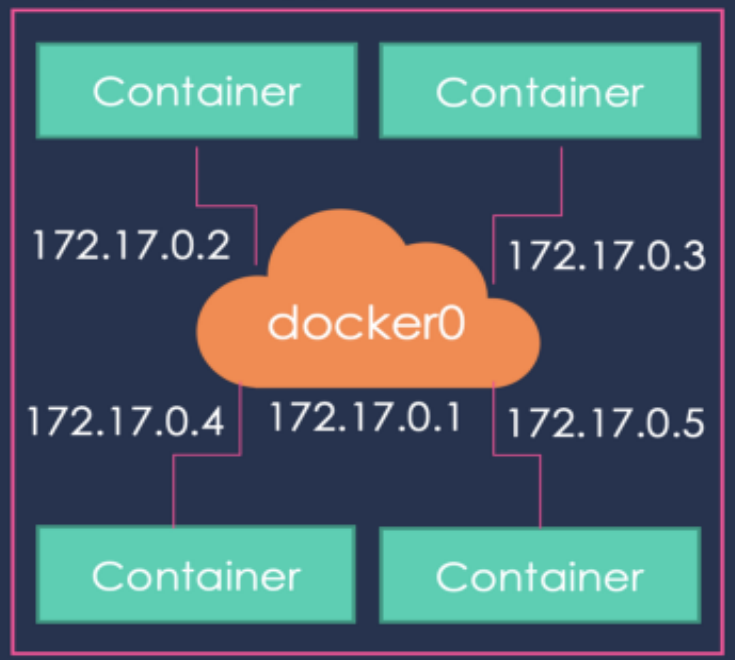
•If facing an error while installing httpd in centos container run below :

sudo sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
sudo sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*
sudo yum update –y

Bridge

```
docker run ubuntu
```

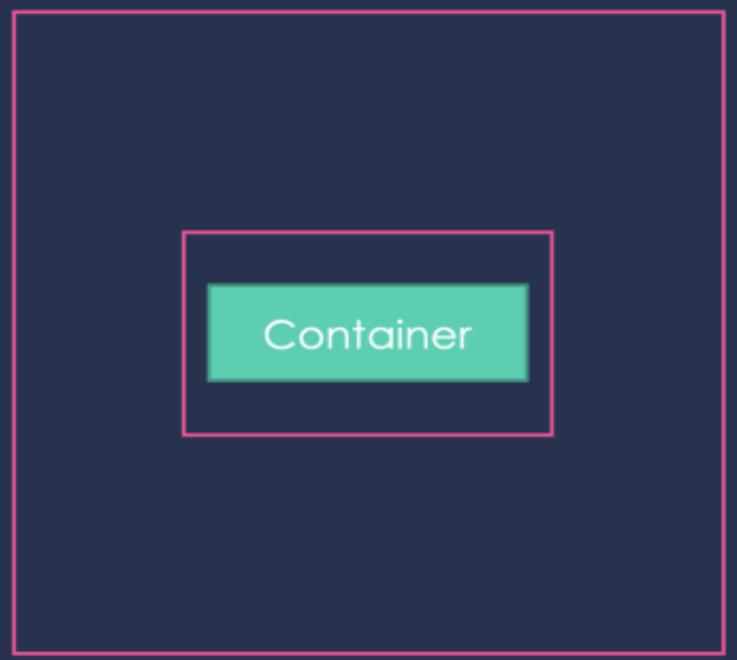
Docker Host



None

```
docker run \
--network=none
ubuntu
```

Docker Host



Host

```
docker run \
--network=host
ubuntu
```

Docker Host

Session: 5

Docker – Containers Advanced

Checking Info

Checking System usage:
$ docker system df // use –v for verbose run

Checking Docker Information:
$ docker system info

Checking Docker events:
$ docker system events //Checking runtime events

Checking Logs:
$ docker logs {containerid}

Cleaning up Unused resources:
$ docker system prune //use –a for deleting all unused images
docker volume prune //to remove all unused volumes

Container

- *Set a Name to container*

  docker run --name my-redis -d redis

- *Setting a hostname from outside:*

  docker run -dit --name container1 --hostname c1 centos

- *Finding stats for container:*

  docker top container-name
  docker stats

- *Inspecting Docker containers*

  docker inspect container-name

- *Homedirectory*

  /var/lib/docker

Resource binding to a Container

Limiting memory(Quota):

$ docker run -dit -m 300M --name c1 redis

Verify :
Docker stats c1

Limiting CPUs:

docker run -dit --cpus 0.02 --name c2 redis

Check the nanocpus

Docker inspect c2 | grep –I cpu

Session: 5

Ports and Volumes

Working with Volumes

- Containers have ephemeral storage
- Data shared within the container is not accessible after it is terminated
- Volumes provide persistence to containers
- Each volume is implicitly or explicitly mapped to a host directory

Working with Volumes

- Question is who is responsible for performing all these operations in Docker.
- And the answer is “Storage Drivers”, some of the common drivers are like “aufs, zfs, overlay, overlay2, device mapper etc.)
- The choice of device driver depends upon the underlying OS.
- “aufs” is the default Storage Driver for Ubuntu, however this is not present in Centos, in such cases “device mapper” is a better option to go with.
- The key part is “docker will pick the best storage driver for you automatically”.
- Execute docker info | more to know about the drivers used by the docket for your host.

Volume mapping

- cd /home/ubuntu/raman/
- touch hostfile

- #On the command-line, with the -v flag for “docker run”.

  docker run -it --name c1 -v /home/ubuntu/raman:/appdata centos

Edit the file hostfile in appdata folder in container

#Exit out the container safely

See the changes getting reflected in local as well.

Listing Volumes

- If a container is stopped, its volumes still exist and are available.
- Since Docker 1.9, we can see all existing volumes:
  docker volume ls

Docker Ports and Volumes

- Two most important things in docker are Ports and Volumes.
- docker images
- docker run -d nginx:latest
- docker ps
- docker inspect <container-id>| grep -i ip
- docker ps
- docker stop <container-id>
- docker rm `docker ps -a -q`

Docker Ports and Volumes

- Lets expose the containers to a random port by docker itself
- docker run -d --name=nginxserver -P nginx:latest
- docker ps (you will get a port mapped by the docker for you)
- docker port <conatainer-name> $CONTAINERPORT
- docker stop yogeshserver

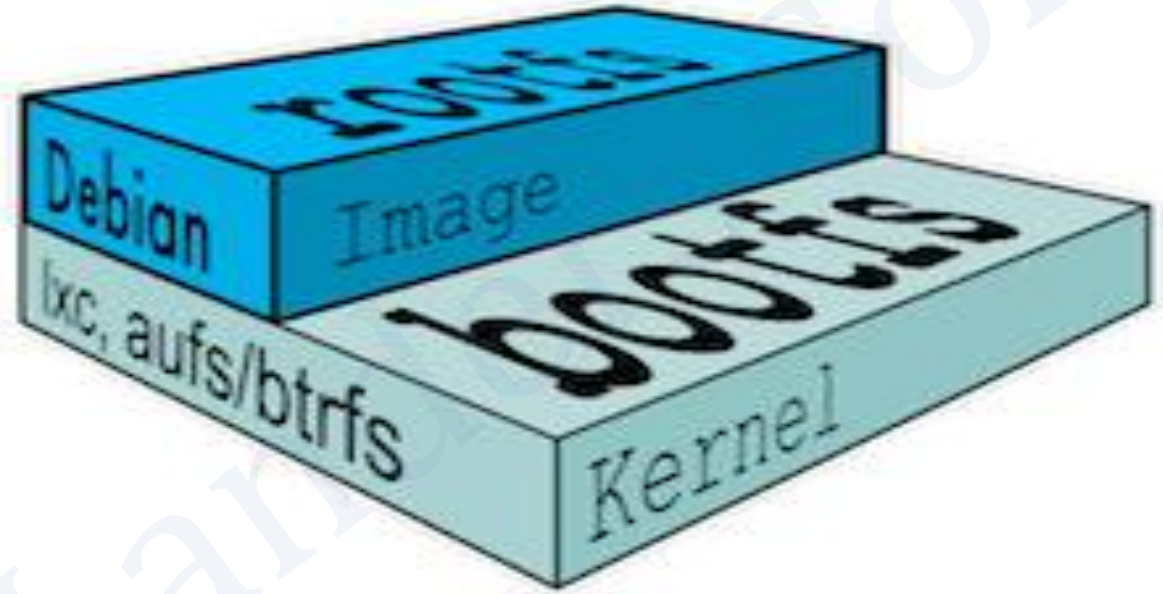
Docker Ports and Volumes

- But how to bound the container with a particulat port:
- docker run -d -p 8080:80 nginx
- Even you can bound multiple ports:
- docker run -d -p 8080:80,8081:443 nginx
- Once done stop and remove the container

Session: 6

Docker - Images

Docker Images



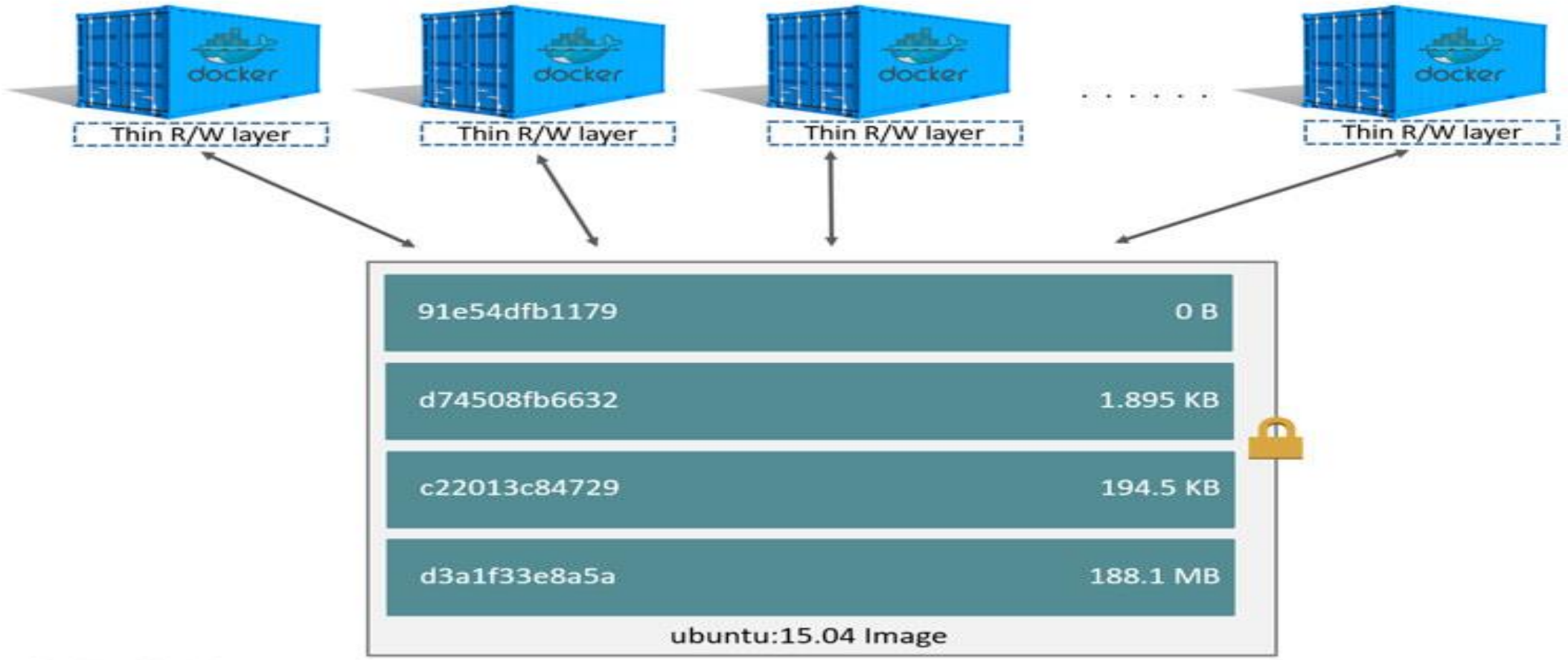
What is an image?

- An image is a collection of files + some meta data.
- Images are made of *layers, conceptually stacked on top of each other.*
- Each layer can add, change, and remove files.
- Images can share layers to optimize disk usage, transfer times, and memory use.

Container vs Image

- An image is a read-only filesystem.
- A container is an encapsulated set of processes running in a read-write copy of that filesystem.
- docker run starts a container from a given image.

Container vs Image



Store & manage images

- Images can be stored:
  - On your Docker host.
  - In a Docker registry.
- You can use the Docker client to download (pull) or upload (push) images.
- To be more accurate: you can use the Docker client to tell a Docker server to push and pull images to and from a registry.
- Lets explore docker public registry called “docker hub”

Showing current images

- Let's look at what images are on our host now.

  docker images

  docker image list

Searching for images

- We cannot list all images on a remote registry, but we can search for a specific keyword:

  docker search zookeeper

- "Stars" indicate the popularity of the image.

Downloading images

- There are two ways to download images.
  - Explicitly, with “docker pull”.
  - Implicitly, when executing “docker run” and the image is not found locally.

- Pulling an image.

  docker pull debian:jessie

- Images can have tags.
- Tags define image versions or variants.
- “docker pull ubuntu” will refer to “ubuntu:latest”.
- The :latest tag is generally updated often.

Docker Image LifeCycle

- To list docker images: docker images
- To remove docker images locally: docker rmi <image-name>
- Special Cases:
- Lets try to remove docker images used by current container: docker rmi <image-name>
- This time you will get an error, but the image can be removed forcefully using “-f” option.
- Docker rmi -f <image-name> (Note: It wont work with Image ID)

Docker Image Information

- Docker Image details:
- docker image list

- Detailed Infromation about docker image:
- ls –lrt /var/lib/docker/image/overlay2/imagedb/content/sha256

- High level certificate sign information (SHA):
- cat /var/lib/docker/image/overlay2/repositories.json

- All details with command line:
- docker image inspect <image-name>

- Even you can check all of the information about the docker image:
- docker history <image-id>

Docker Image and Conatiner LifeCycle

- To list docker: docker ps -a
- To list docker images: docker images
- To remove container: docker rm <container-id> or <Name>
- To remove multiple container: docker rm `docker ps –a –q`
- To remove docker images locally: docker rmi <image-name>

Docker Image and Conatiner LifeCycle

- Special Cases:
- Lets try to remove docker images used by current container: docker rmi <image-name>
- This time you will get an error, but the image can be removed forcefully using “-f” option.
- Docker rmi -f <image-name> (Note: It wont work with Image ID)
- The best part is though you have removed the image forcefully, but this will not impact the current container as the container preserves the metadata in containers folder.
- You containers are safe!.

Session: 7

Building Images

Building Images Interactively

- Let's have a Use Case:
  - We will build an image that has httpd.
  - First, we will do it manually with docker commit.
  - Then, we will use a Dockerfile and "docker build".

Create a new container

- Let's start from base image "centos":
  docker run -dit --name c1 centos:7

  docker attach c1
  yum update –y

  yum install –y httpd
  exit

- Inspect the changes:
  docker diff <yourContainerId>

- Commit the changes:

docker commit -m "added httpd and updated" -a "raman khanna" c1 ramancentosimage:v1

Dockerfile overview

- A Dockerfile is a build recipe for a Docker image.
- It contains a series of instructions telling Docker how an image is constructed.
- The “docker build” command builds an image from a Dockerfile.

First Dockerfile

- Create a directory to hold our Dockerfile.
  mkdir myimage

- Create a Dockerfile inside this directory.
  cd myimage
  vi Dockerfile

- Write below in our Dockerfile

```
FROM centos:7
RUN yum update -y
RUN yum -y install httpd
```

First Dockerfile

- “FROM” indicates the base image for our build.
- Each “RUN” line will be executed by Docker during the build.
- No input can be provided to Docker during the build.

First Dockerfile

- Build the Dockerfile:

  docker build -t httpd . Or docker build -t httpd /home/ubuntu/raman/myimage/

- -t indicates the tag to apply to the image.

- . indicates the location of the Directory of Dockerfile.

Run & Tag the image

- Let's run the new images:
  docker run -it <newImageId>
  rpm –qa | grep –i httpd

Using Image & viewing history

- The history command lists all the layers composing an image.
- For each layer, it shows its creation time, size, and creation command.
- When an image was built with a Dockerfile, each layer corresponds to a line of the Dockerfile.

  docker history httpd

Dockerfile

- Dockerfile:

```
FROM centos:7
MAINTAINER  Raman Khanna raman.khanna@TechLanders.com
RUN mkdir /data
RUN yum update -y
RUN yum -y install httpd   php
RUN echo " TechLanders Solutions Deals in DevOps and Cloud" > /var/www/html/index.html
EXPOSE 80
VOLUME  /data
RUN echo "httpd" >> /root/.bashrc
CMD ["/bin/bash"]
```

- Build the image:

```
docker build -t webapp:v1 .

docker run -dit --name c1 -p 8080:80 webapp:v1

curl 172.31.84.13:8080
```

Browse in browser as well

COPY Instruction

- For Use Case, let's build a container that copy file from localhost

  echo " TechLanders Solutions Deals in DevOps and Cloud " > /home/ubuntu/image/index.html

Dockerfile content

```
FROM centos:7
RUN yum update -y
RUN yum install -y httpd
COPY ./index.html /var/www/html/index.html
EXPOSE 80
WORKDIR /var/www/html
CMD ["httpd","-D","FOREGROUND"]
```

COPY Instruction

- For Use Case, let's build a container that compiles a basic "Hello world" program in C.
- hello.c

```
[root@TechLanders yogesh]# cat hello.c
#include<stdio.h>
int main () {
      puts("Hello, TechLanders!");
      return 0;
}
```

- Dockerfile

```
[root@TechLanders yogesh]# cat Dockerfile
FROM ubuntu
RUN apt-get update
RUN apt-get install -y build-essential
COPY hello.c /
RUN make hello
CMD /hello
```

Note: Using COPY keyword we can copy the files from Docker Host to a container.

COPY Instruction

- docker build -t helloworld:v1 .

- docker run -it –name prodv4 helloworld:v1 /bin/bash

- root@519a9d815a29:/# ls -lrt /hello
- -rwxr-xr-x. 1 root root 8600 Sep 18 11:54 /hello

- root@519a9d815a29:/# ./hello
- Hello, world!

ENTRYPOINT vs. CMD

CMD

ENTRYPOINT

Specify the command to run at container startup

```
CMD ["somecommand.sh"]
```

```
> docker run myimage

Running somecommand.sh
...
```

```
ENTRYPOINT ["somecommand.sh"]
```

```
> docker run myimage

Running somecommand.sh
...
```

Dockerfile >

```
FROM debian:buster

COPY . /myproject

RUN apt-get update ...

CMD ["./cmd1.sh"]
```

> "docker run my-image"

Exec ==> `./cmd1.sh`

> "docker run my-image cmd2.sh"

Exec ==> `cmd2.sh`

Dockerfile >

```
FROM debian:buster

COPY . /myproject

RUN apt-get update ...

ENTRYPOINT ["entrypoint1.sh"]

CMD ["param1", "param2"]
```

> "docker run my-container"

Exec ==>

```
entrypoint1.sh param1 param2
```

> "docker run my-container cmd2"

Exec ==>

```
entrypoint1.sh cmd2
```

Docker Restart Policy

$ docker run -dit --restart unless-stopped centos

Flag	Description
no	Do not automatically restart the container. (the default)
on-failure	Restart the container if it exits due to an error, which manifests as a non-zero exit code.
always	Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted. (See the second bullet listed in restart policy details)
unless-stopped	Similar to always, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.

Docker Registries

Local Registry (Local to Host)

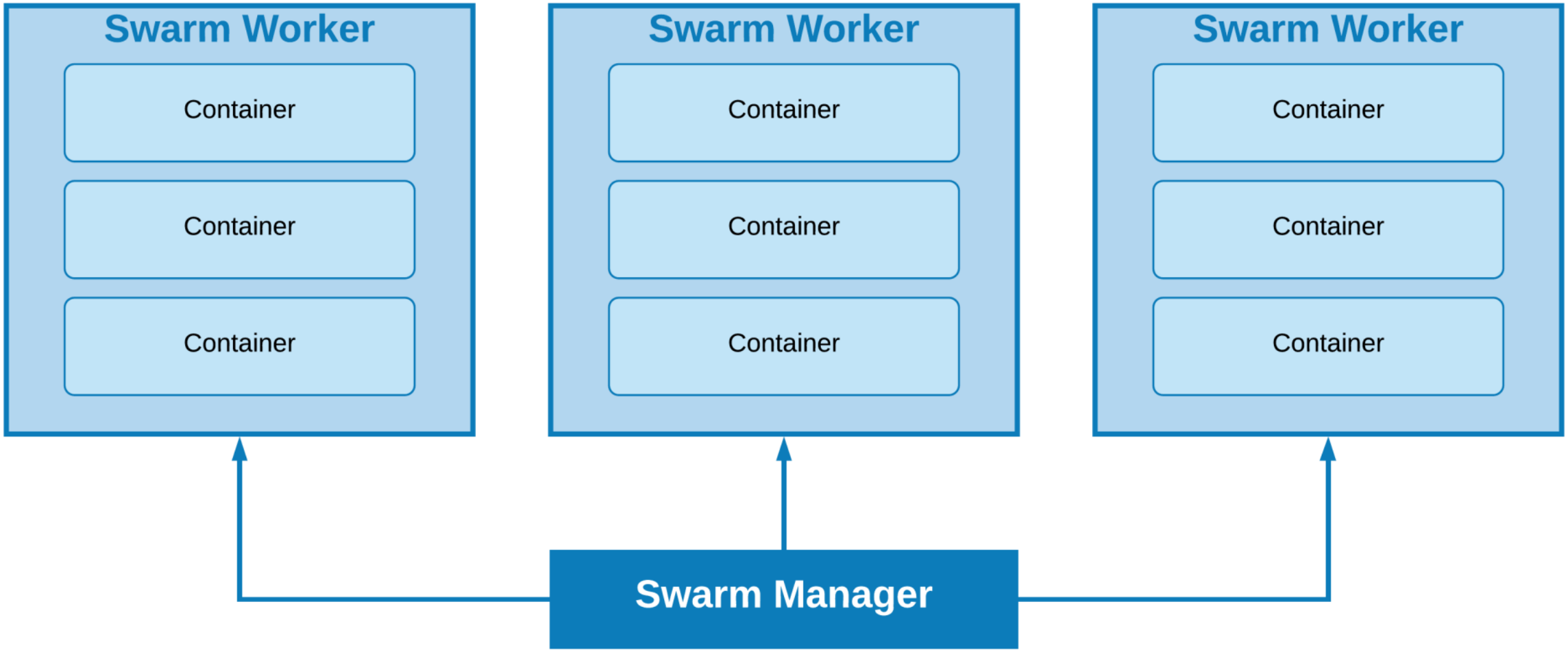
Remote Registry (Private)

Global Registry (Public)

Docker Trusted Registry

- Enterprise-grade image storage solution from Docker
- Highly Secure
- Image and job management with CICD
- HA Availability
- Efficiency with Near to user storage and bandwidth sharing
- Security Scanning
- Similar tools in market Sonatype Nexus (open source), AWS ECR (PaaS), azure Container Registry, GCP container Registry etc

Docker Swarm



Docker Limitations

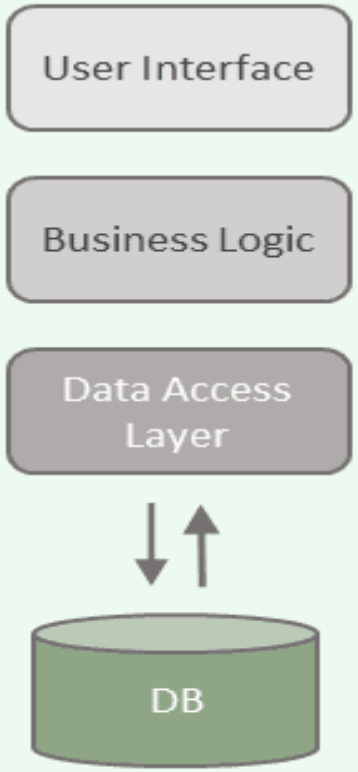
- Hardware Issues? High Availability?
- How IP address will be managed for failover?
- Scaling?
- Auto Healing?
- Autoscaling?
- No Application Management - Only Containerization
- Updation of application/management

Why Kubernetes

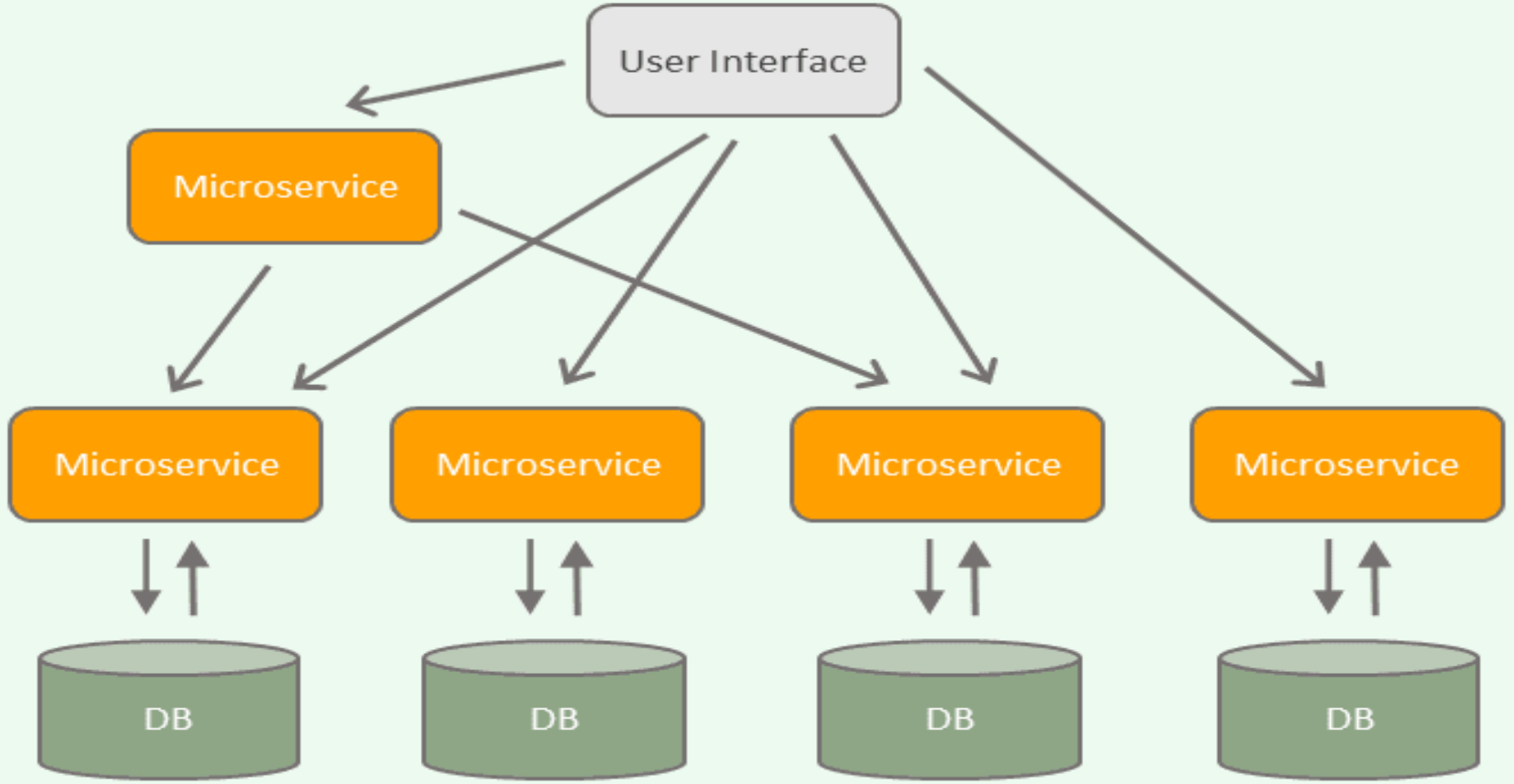
- Kubernetes can schedule and run application containers on clusters of physical or virtual machines.
- **host-centric** infrastructure to a **container-centric** infrastructure.
- Orchestrator
- Load balancing
- Auto Scaling
- Application Health checks
- Rolling updates

Questions & Answers

MONOLITHIC ARCHITECTURE



MICROSERVICES ARCHITECTURE

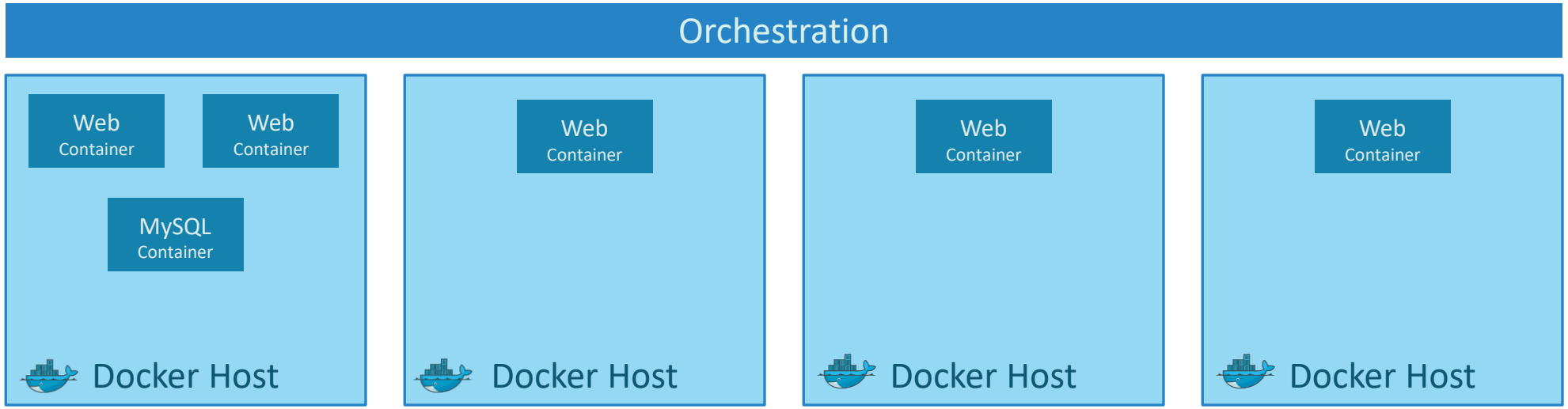


Container Orchestration

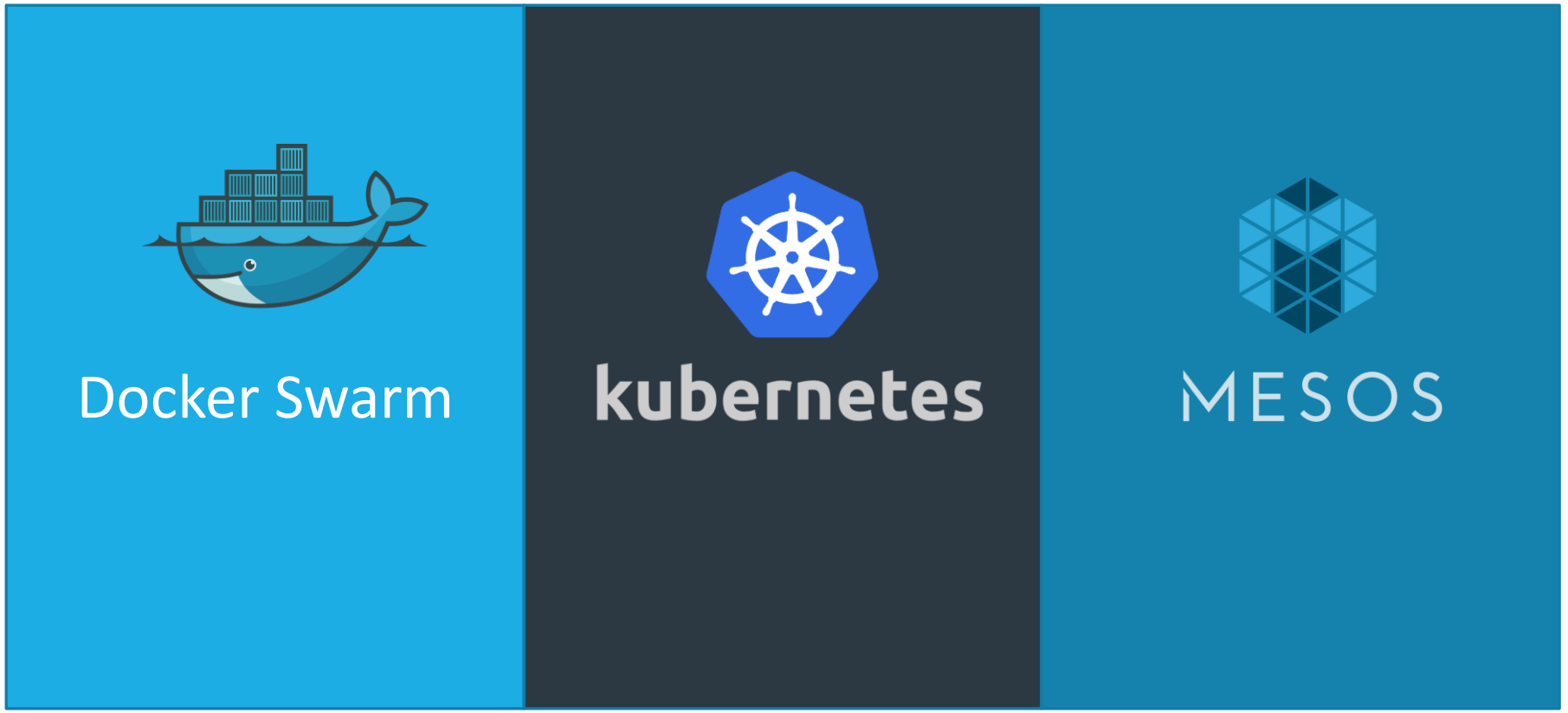
Containers Limitation?

High Availability?
Overlay Network?
Versioning of Application – Rollout, Rollback?
Scaling?
Autoscaling?
Monitoring?
Dependency between containers?

Container orchestration



Orchestration Technologies



What is Kubernetes?

The Kubernetes project was started by Google in 2014.

Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale.

Kubernetes can run on a range of platforms, from your laptop, to VMs on a cloud provider, to rack of bare metal servers.

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

portable: with all public, private, hybrid, community cloud

self-healing: auto-placement, auto-restart, auto-replication, auto-scaling

Why Kubernetes

Kubernetes can schedule and run application containers on clusters of physical or virtual machines.

host-centric infrastructure to a **container-centric** infrastructure.

Orchestrator

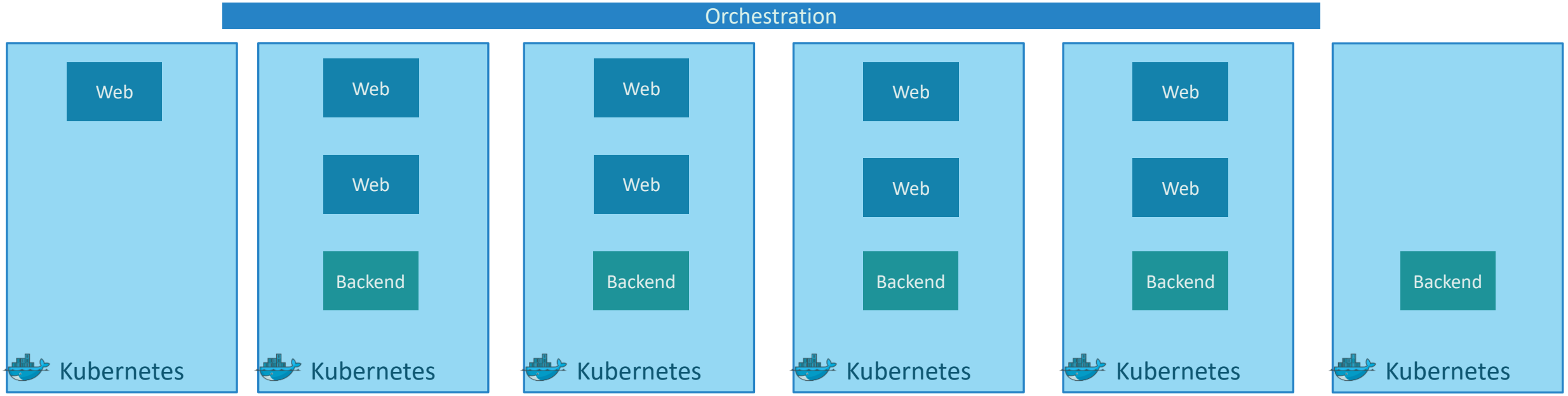
Load balancing

Auto Scaling

Application Health checks

Rolling updates

Kubernetes Advantage



And that is kubernetes..

Setup

Minikube

Kubeadm

Google Cloud Platform

Amazon Web Services

play-with-k8s.com

Setup Kubernetes

Setup - kubeadm

Kubernetes Cluster

A Kubernetes cluster consists of two types of resources:

Master: Which coordinates with the cluster

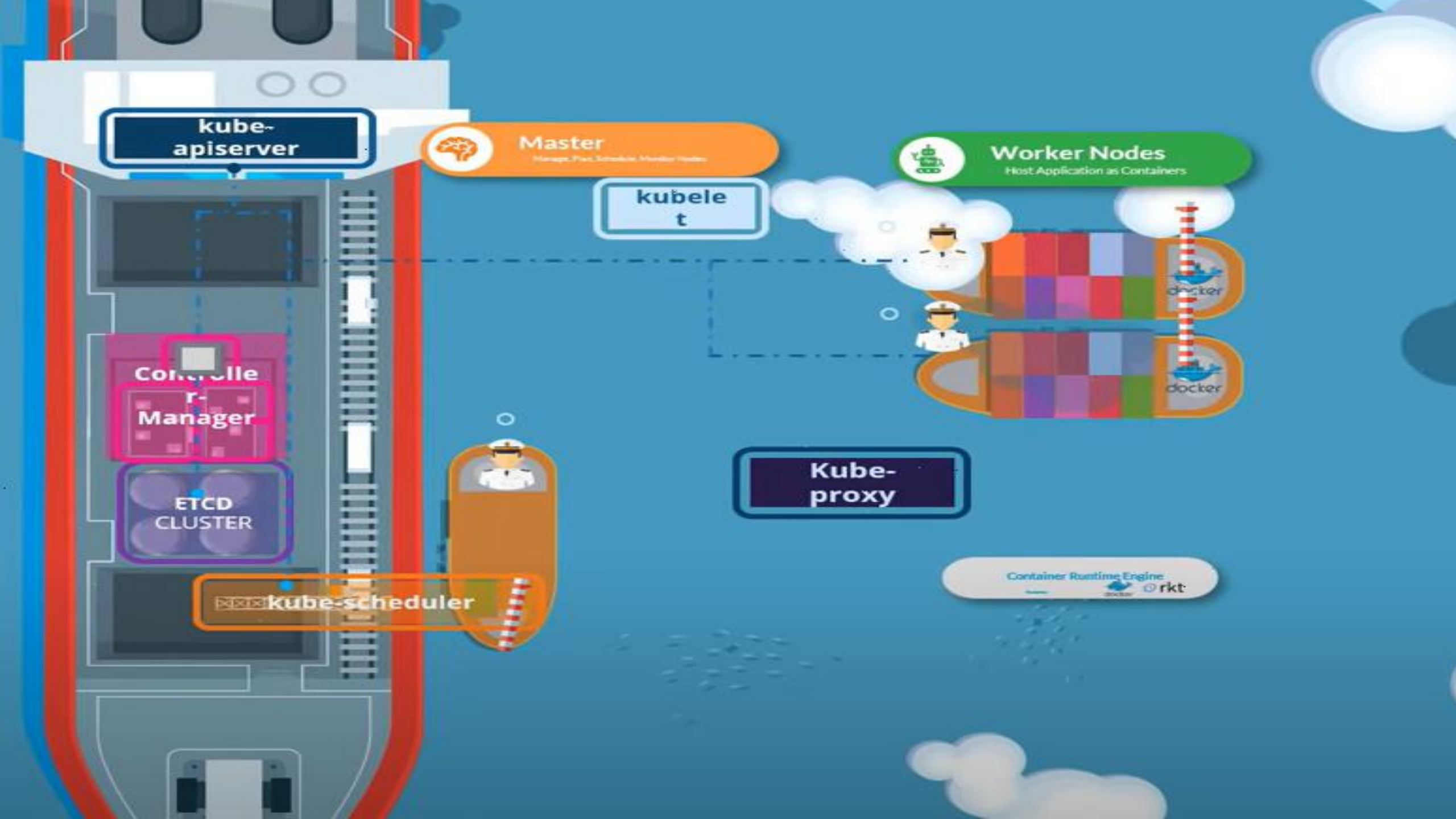
The Master is responsible for managing the cluster. The master coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

Nodes: Are the workers that run application

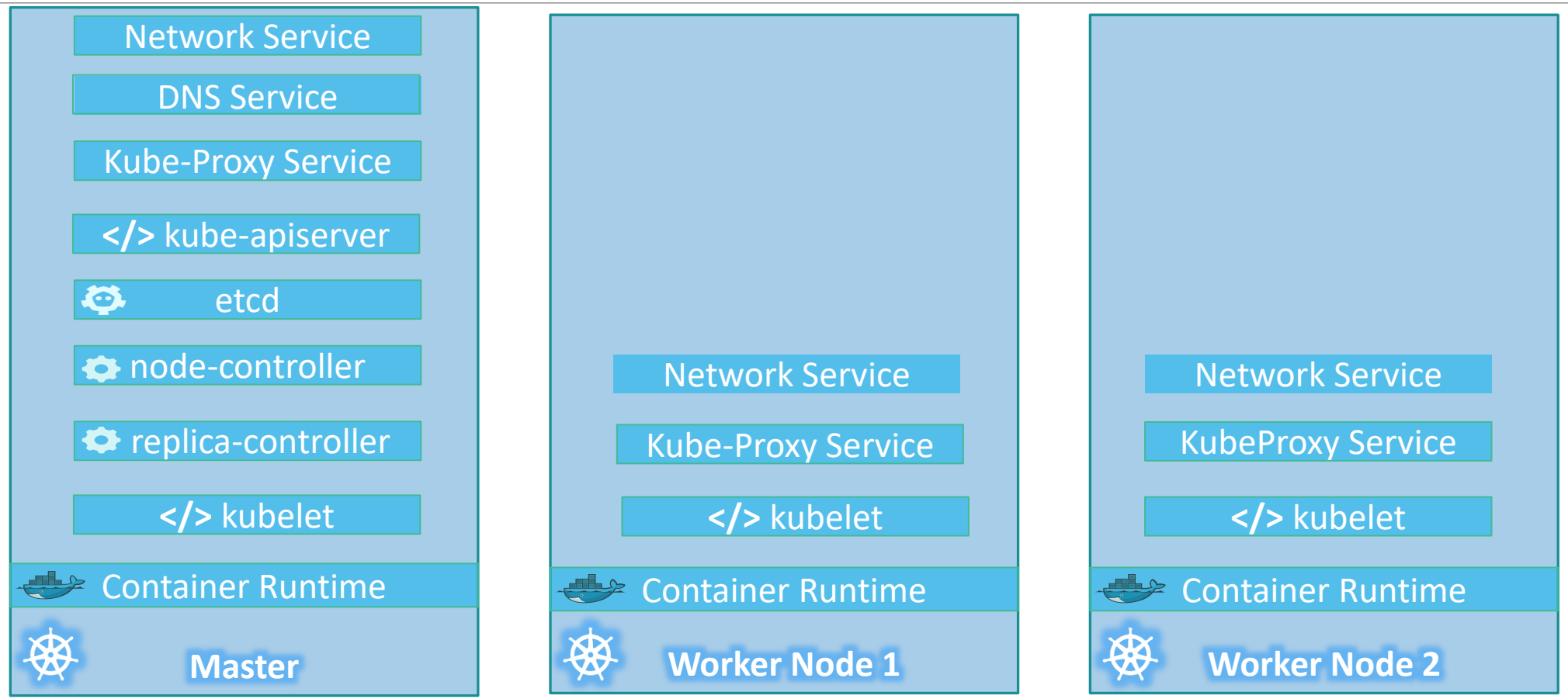
A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.

Masters manage the cluster and the nodes are used to host the running applications.

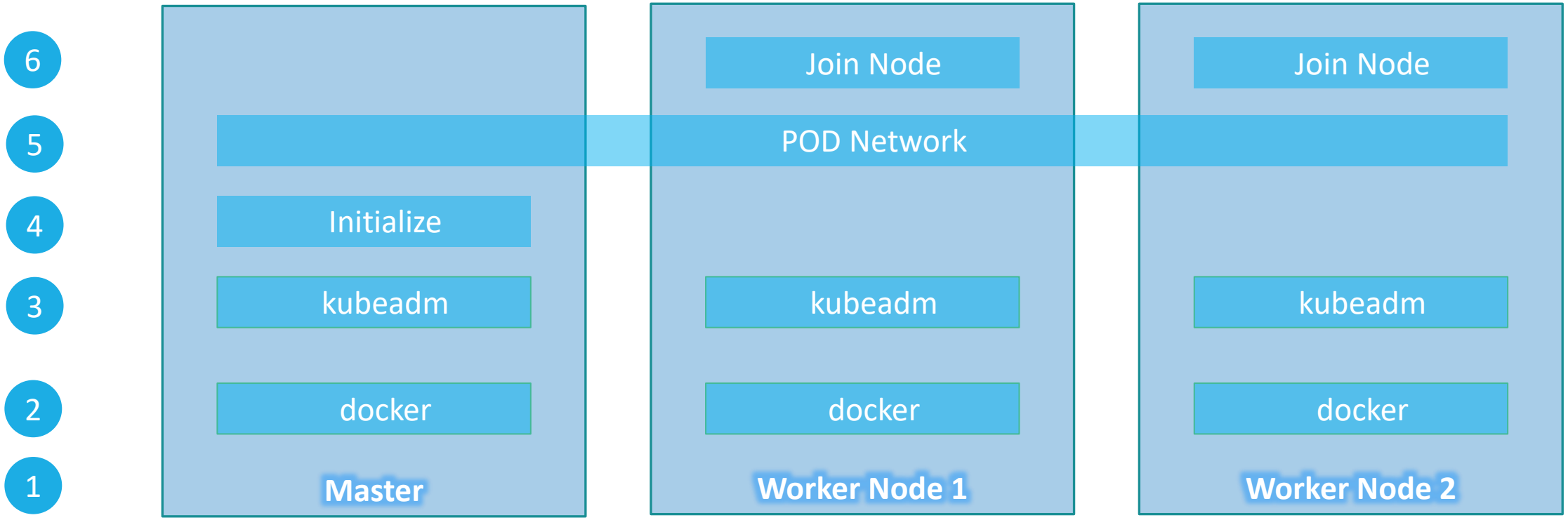
The nodes communicate with the master using the Kubernetes API, which the master exposes.


kube-
apiserver
Master
kubele
t
Worker Nodes
Host Application as Containers
docker
docker
Controlle
r-
Manager
ETCD
CLUSTER
kube-scheduler
Kube-
proxy
Container Runtime Engine
rkt

kubeadm


Network Service
DNS Service
Kube-Proxy Service
</> kube-apiserver
etcd
node-controller
replica-controller
</> kubelet
Container Runtime
Master
Network Service
Kube-Proxy Service
</> kubelet
Container Runtime
Worker Node 1
Network Service
KubeProxy Service
</> kubelet
Container Runtime
Worker Node 2

Steps


6
5
4
3
2
1
Join Node
Join Node
POD Network
Initialize
kubeadm
kubeadm
kubeadm
docker
docker
docker
Master
Worker Node 1
Worker Node 2

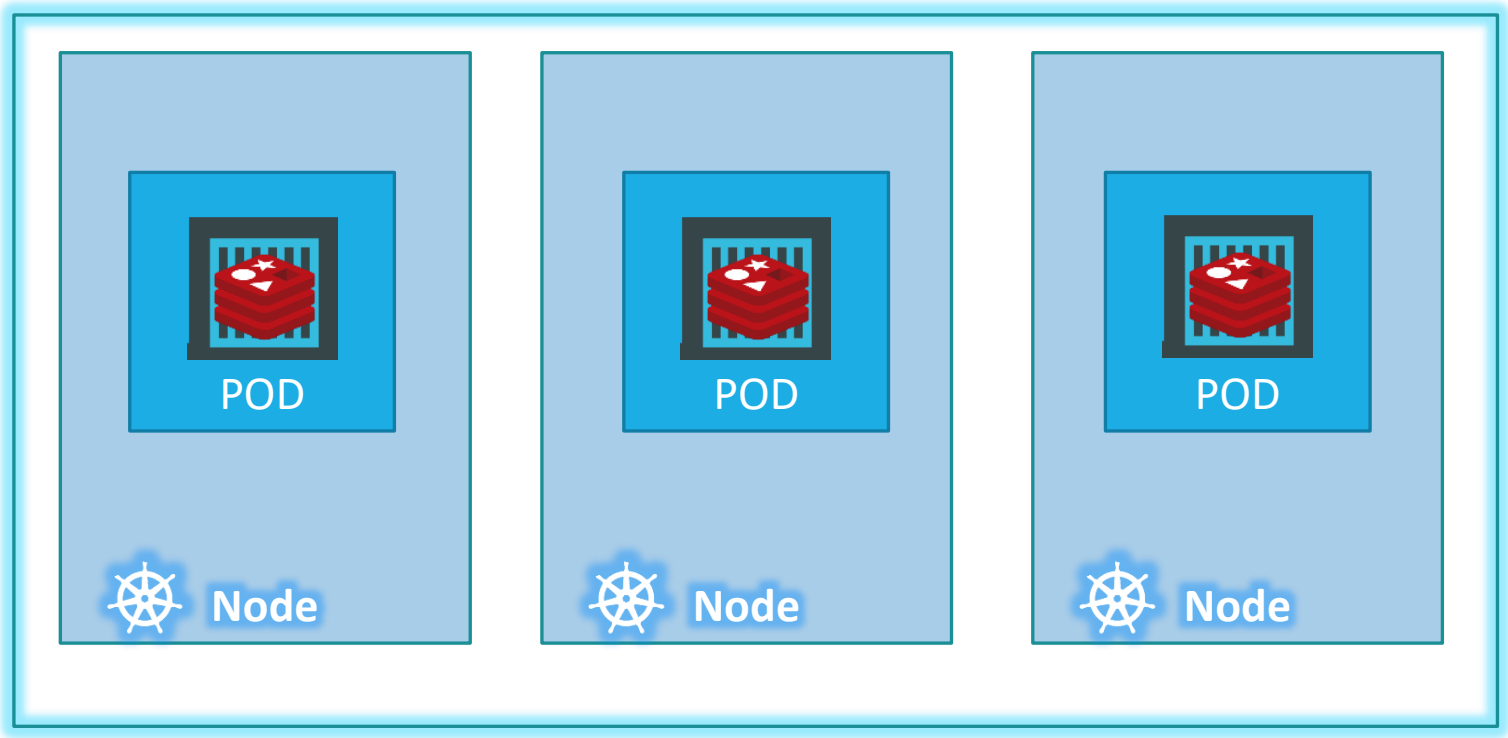
POD

Assumptions

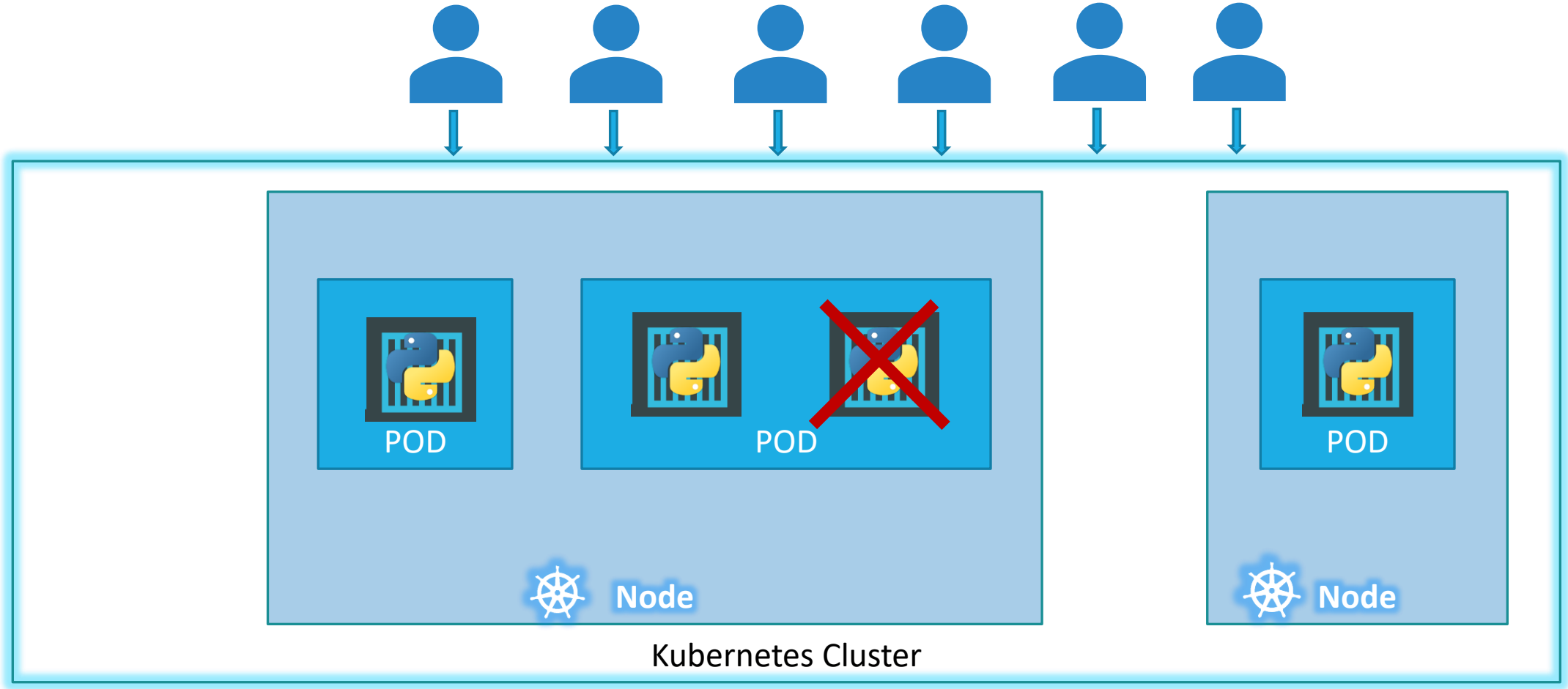
Docker Image

Kubernetes Cluster

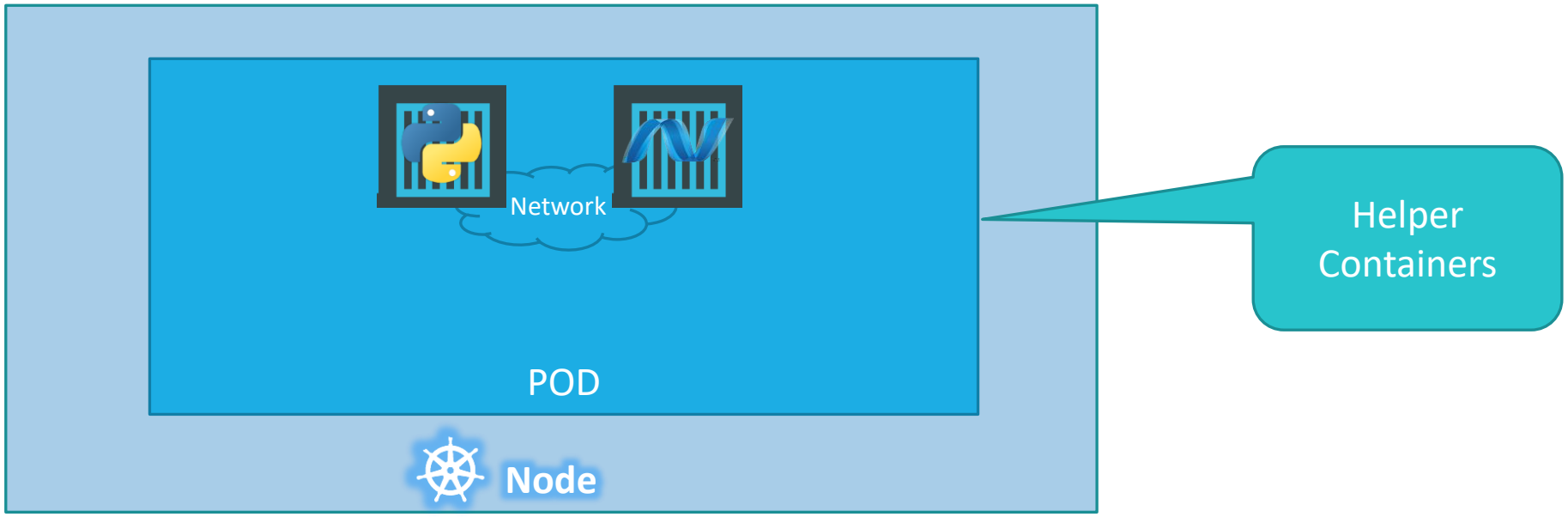
POD



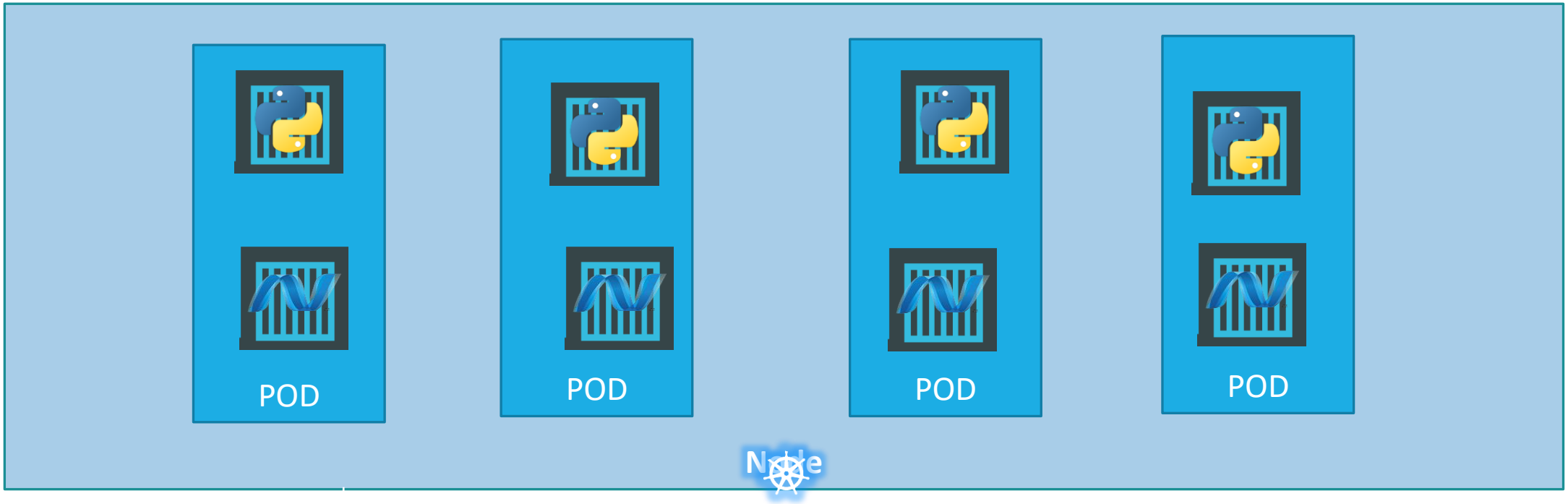
POD



Multi-Container PODs



PODs Again!



Note: I am avoiding networking and load balancing details to keep explanation simple.

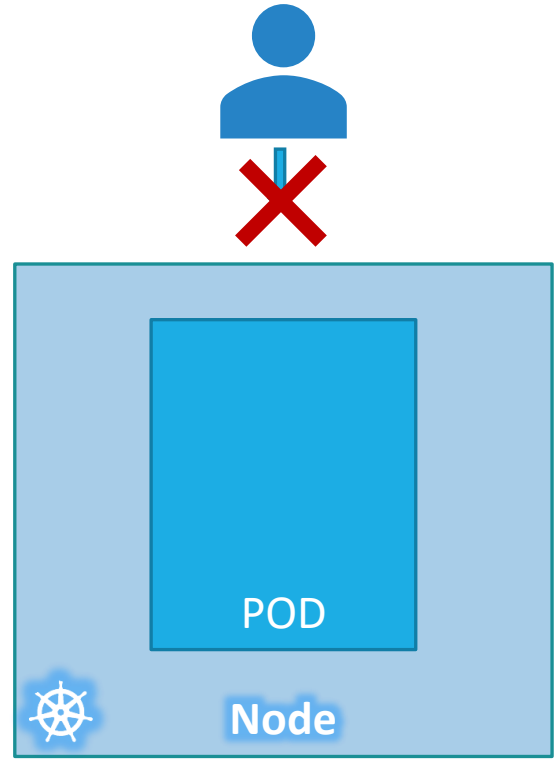
kubectl

```
kubectl run nginx --image nginx
```

```
kubectl get pods
```

```
C:\Kubernetes>kubectl get pods
NAME                     READY     STATUS              RESTARTS   AGE
nginx-8586cf59-whssr     0/1       ContainerCreating   0          3s
```

```
C:\Kubernetes>kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-8586cf59-whssr     1/1       Running   0          8s
```



YAML Introduction

TechLanders
Delivering the future

POD

With YAML

YAML in Kubernetes

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
    - name: nginx-container
      image: nginx
```

String

String

List/Array

1st Item in List

```
kubectl create -f pod-definition.yml
```

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

Commands

TechLanders
Delivering the future

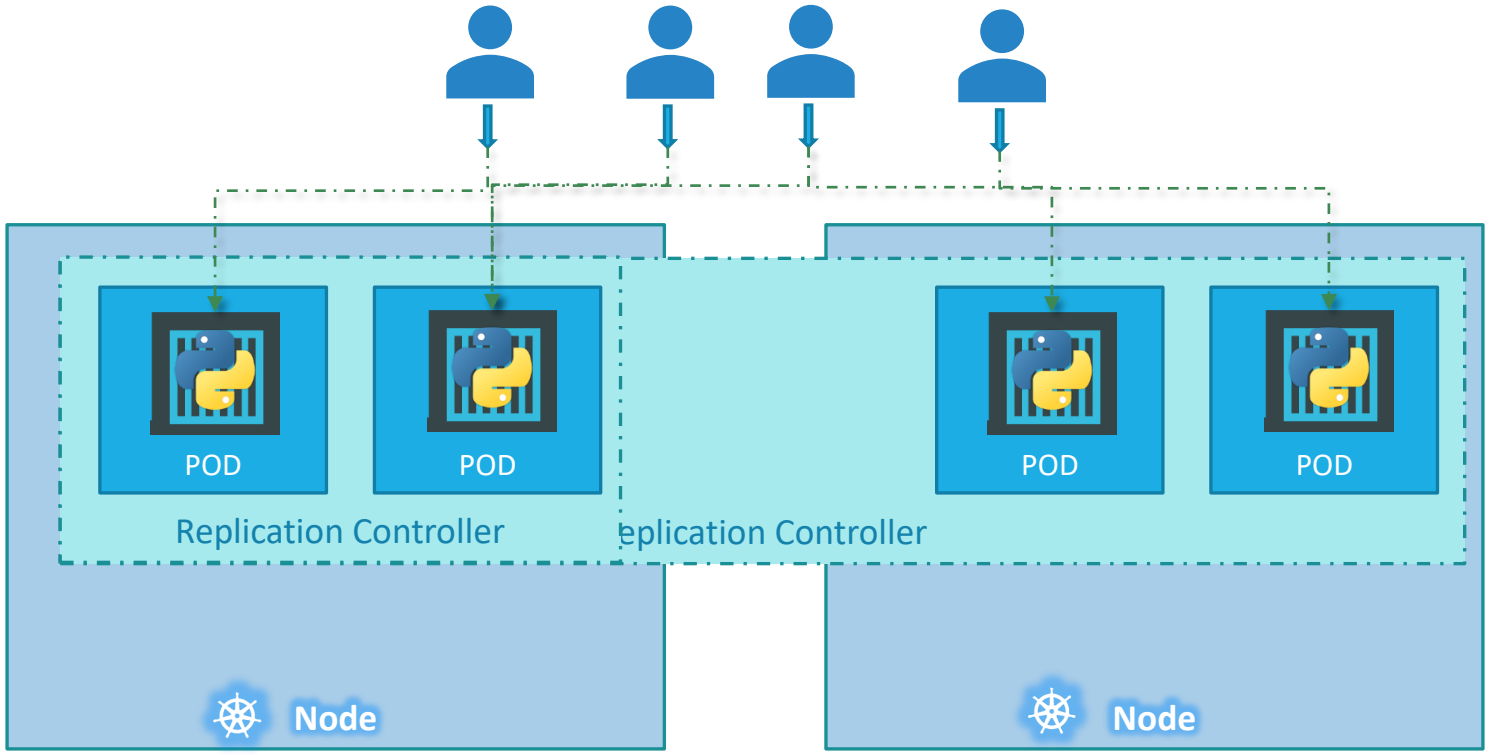
```
> kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
myapp-pod       1/1       Running   0          20s
```

```
> kubectl describe pod myapp-pod
Name:         myapp-pod
Namespace:    default
Node:         minikube/192.168.99.100
Start Time:   Sat, 03 Mar 2018 14:26:14 +0800
Labels:       app=myapp
              name=myapp-pod
Annotations:  <none>
Status:       Running
IP:           10.244.0.24
Containers:
  nginx:
    Container ID:   docker://830bb56c8c42a86b4bb70e9c1488fae1bc38663e4918b6c2f5a783e7688b8c9d
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:4771d09578c7c6a65299e110b3ee1c0a2592f5ea2618d23e4ffe7a4cab1ce5de
    Port:           <none>
    State:          Running
      Started:      Sat, 03 Mar 2018 14:26:21 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-x95w7 (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Events:
  Type    Reason                 Age   From               Message
  ----    ------                 ----  ----               -------
  Normal  Scheduled              34s   default-scheduler  Successfully assigned myapp-pod to minikube
  Normal  SuccessfulMountVolume  33s   kubelet, minikube  MountVolume.SetUp succeeded for volume "default-token-x95w7"
  Normal  Pulling                33s   kubelet, minikube  pulling image "nginx"
  Normal  Pulled                 27s   kubelet, minikube  Successfully pulled image "nginx"
  Normal  Created                27s   kubelet, minikube  Created container
  Normal  Started                27s   kubelet, minikube  Started container
```

Replication Controller

High Availability

Load Balancing & Scaling



Replication Controller

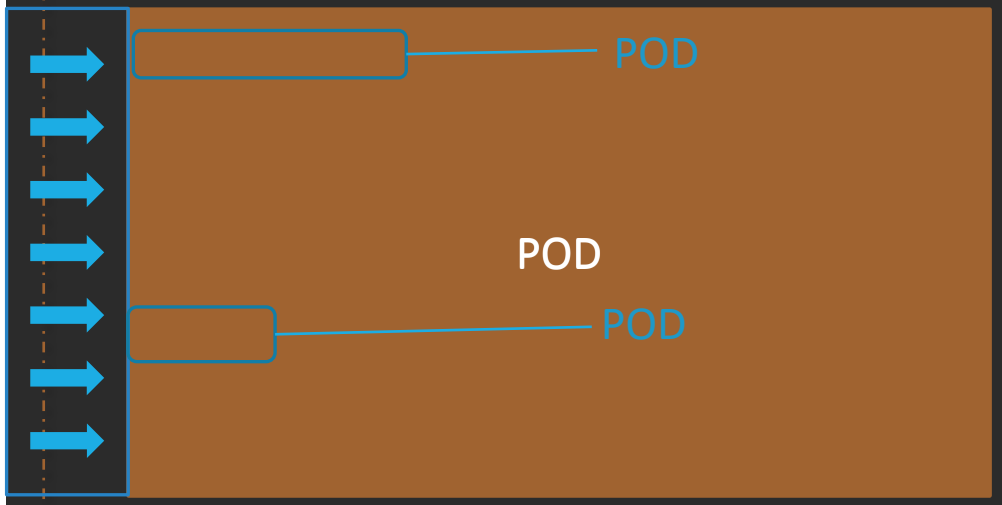
Replica Set

rc-definition.yml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
      app: myapp
      type: front-end
spec:
  template:
    replicas: 3
```

Replication Controller

Replication Controller



pod-definition.yml

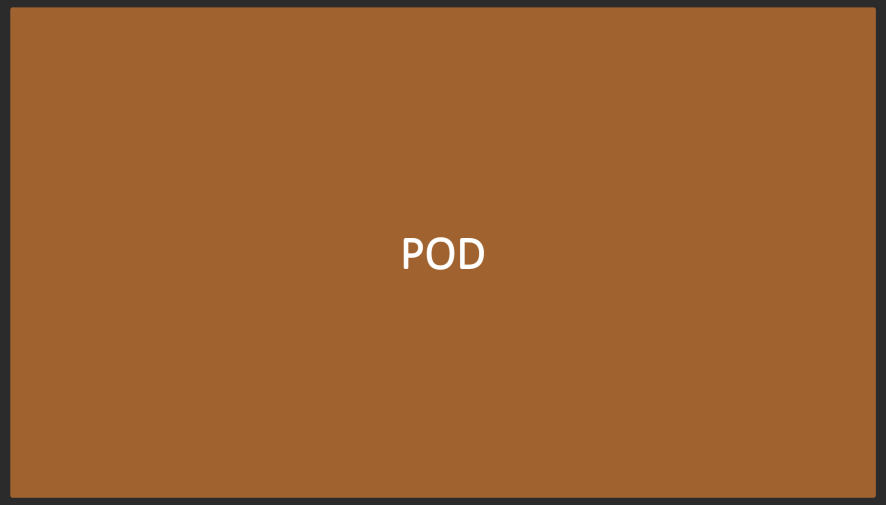
```yaml
apiVersion: v1
kind: Pod

metadata:
  name: myapp-pod
  labels:
     app: myapp
     type: front-end
spec:
  containers:
  - name: nginx-container
    image: nginx
```

```
> kubectl create -f rc-definition.yml
replicationcontroller "myapp-rc" created
```

```
> kubectl get replicationcontroller
NAME       DESIRED   CURRENT   READY     AGE
myapp-rc   3         3         3         19s
```

```
> kubectl get pods
NAME             READY     STATUS    RESTARTS   AGE
myapp-rc-4lvk9   1/1       Running   0          20s
myapp-rc-mc2mf   1/1       Running   0          20s
myapp-rc-px9pz   1/1       Running   0          20s
```

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-repl
  labels:
      app: myapp
      type: front-end
spec:
  template:
      
      
  replicas: 3
  selector:
      matchLabels:
          type: front-end
```

```
error: unable to recognize "replicaset-
definition.yml": no matches for /, Kind=ReplicaSet
```

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
      app: myapp
      type: front-end
spec:
  containers:
  - name: nginx-container
    image: nginx
```

```
> kubectl create -f replicaset-definition.yml
replicaset "myapp-replicaset" created
```

```
> kubectl get replicaset
NAME              DESIRED   CURRENT   READY     AGE
myapp-replicaset  3         3         3         19s
```

```
> kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
myapp-replicaset-9ddl9  1/1       Running   0          45s
myapp-replicaset-9jtpx  1/1       Running   0          45s
myapp-replicaset-hq84m  1/1       Running   0          45s
```

TechLanders
Delivering the future

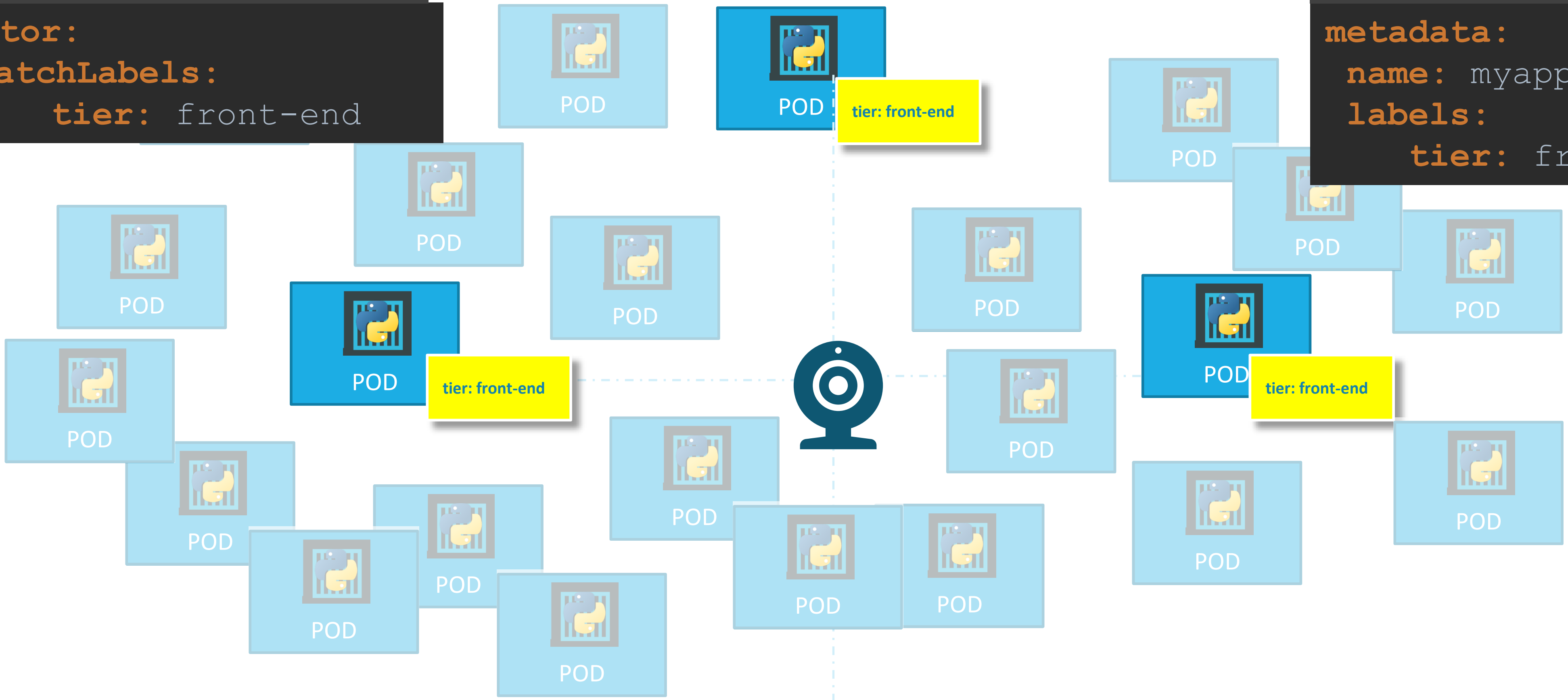
Labels and Selectors

replicaset-definition.yml

```yaml
selector:
    matchLabels:
        tier: front-end
```

pod-definition.yml

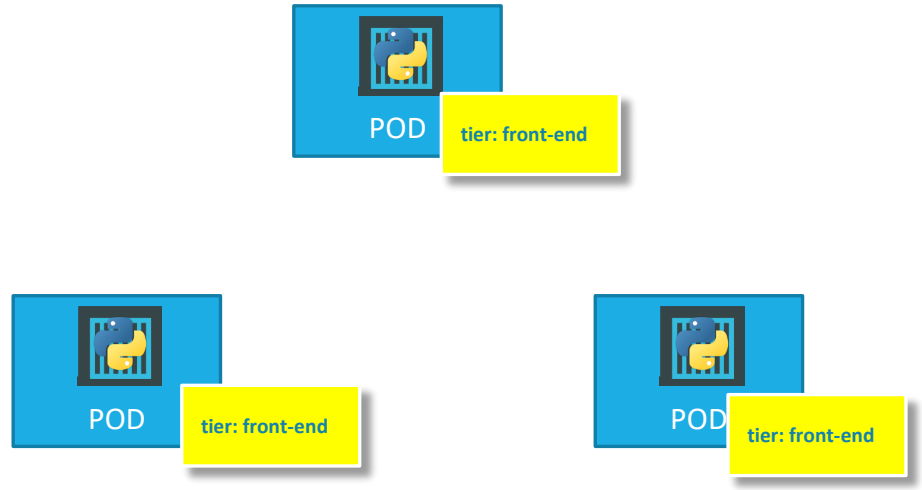
```yaml
metadata:
  name: myapp-pod
  labels:
     tier: front-end
```



replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
          app: myapp
          type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
  selector:
    matchLabels:
        type: front-end
```

Template



Scale

```
> kubectl replace -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 replicaset myapp-replicaset
```

TYPE NAME

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
     metadata:
      name: myapp-pod
      labels:
         app: myapp
         type: front-end
     spec:
       containers:
       - name: nginx-container
         image: nginx
  replicas: 6
  selector:
     matchLabels:
        type: front-end
```

commands

```
> kubectl create –f replicaset-definition.yml
```

```
> kubectl get replicaset
```

```
> kubectl delete replicaset myapp-replicaset
```

*Also deletes all underlying PODs

```
> kubectl replace -f replicaset-definition.yml
```

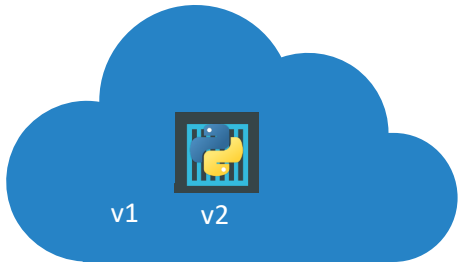
```
> kubectl scale –replicas=6 -f replicaset-definition.yml
```

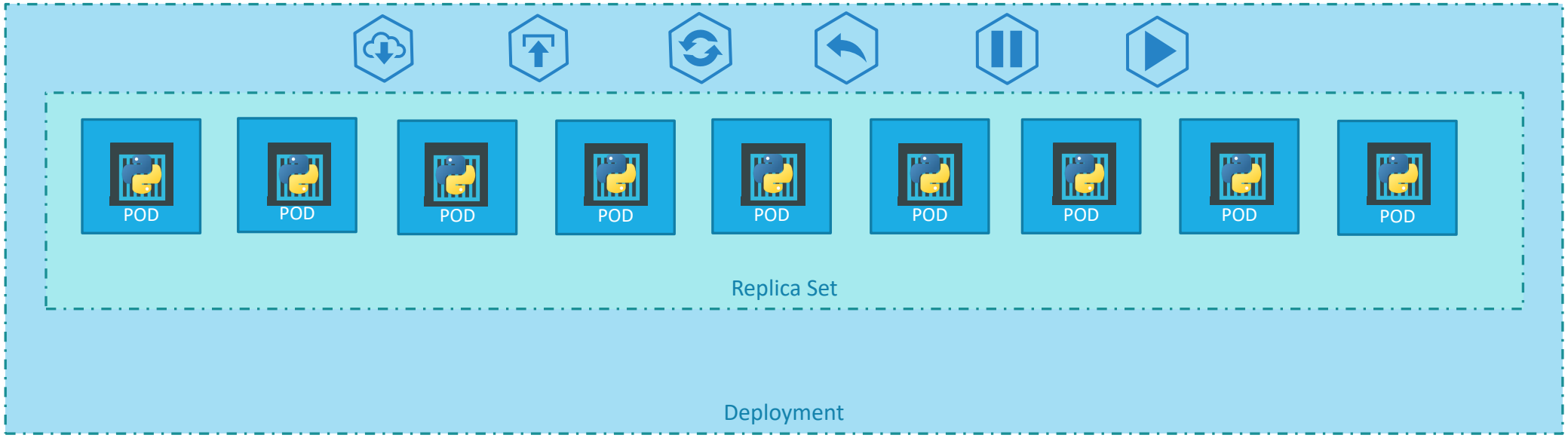
Demo

ReplicaSet

Deployment

Deployment


v1
v2


POD
POD
POD
POD
POD
POD
POD
POD
POD
Replica Set
Deployment

Definition

```
> kubectl create -f deployment-definition.yml
deployment "myapp-deployment" created
```

```
> kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
myapp-deployment   3         3         3            3           21s
```

```
> kubectl get replicaset
NAME                          DESIRED   CURRENT   READY     AGE
myapp-deployment-6795844b58   3         3         3         2m
```

```
> kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
myapp-deployment-6795844b58-5rbjl   1/1       Running   0          2m
myapp-deployment-6795844b58-h4w55   1/1       Running   0          2m
myapp-deployment-6795844b58-lfjhv   1/1       Running   0          2m
```

deployment-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-deployment
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
         app: myapp
         type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
       type: front-end
```

TechLanders

commands

```
> kubectl get all
NAME                       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/myapp-deployment    3         3         3            3           9h

NAME                               DESIRED   CURRENT   READY     AGE
rs/myapp-deployment-6795844b58     3         3         3         9h

NAME                                    READY     STATUS    RESTARTS   AGE
po/myapp-deployment-6795844b58-5rbjl    1/1       Running   0          9h
po/myapp-deployment-6795844b58-h4w55    1/1       Running   0          9h
po/myapp-deployment-6795844b58-lfjhv    1/1       Running   0          9h
```

Demo

Deployment

Demo

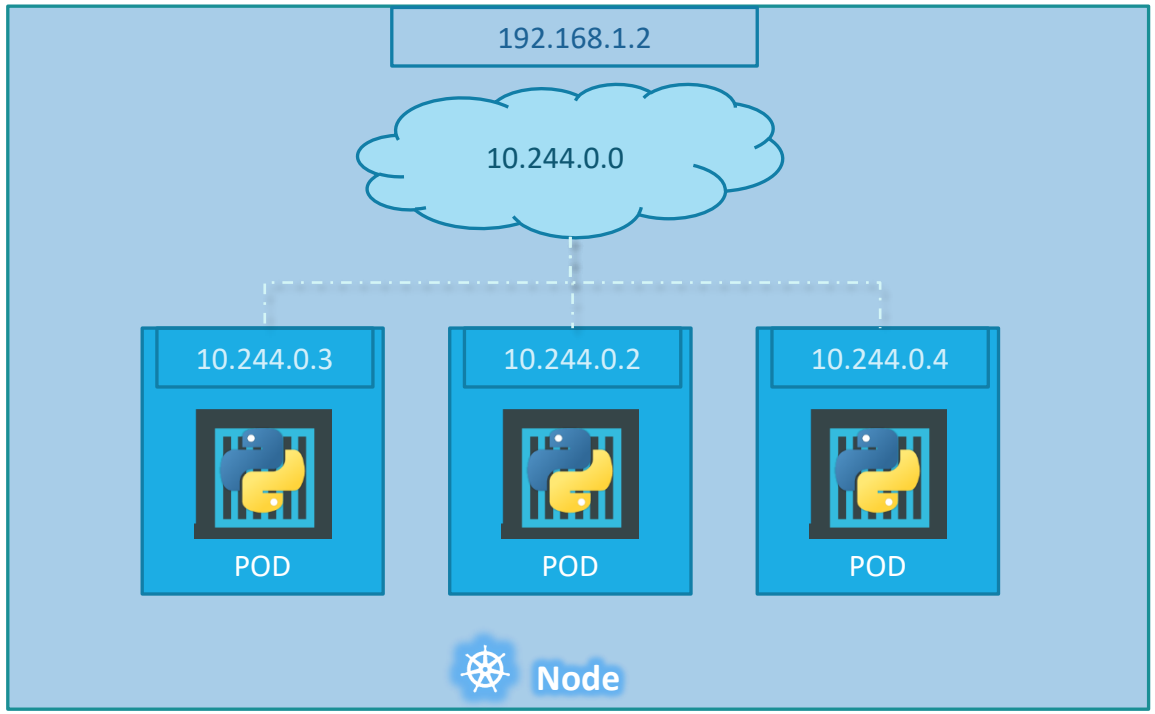
Deployment

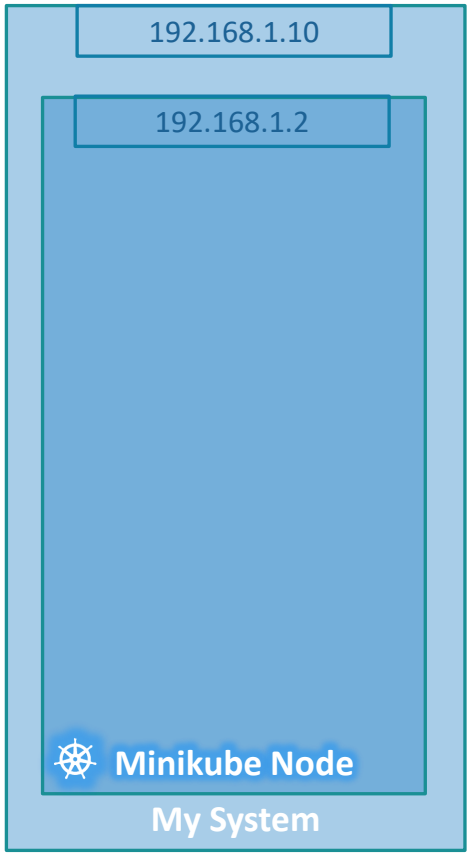
Networking 101

Raman Khanna

Kubernetes Networking - 101

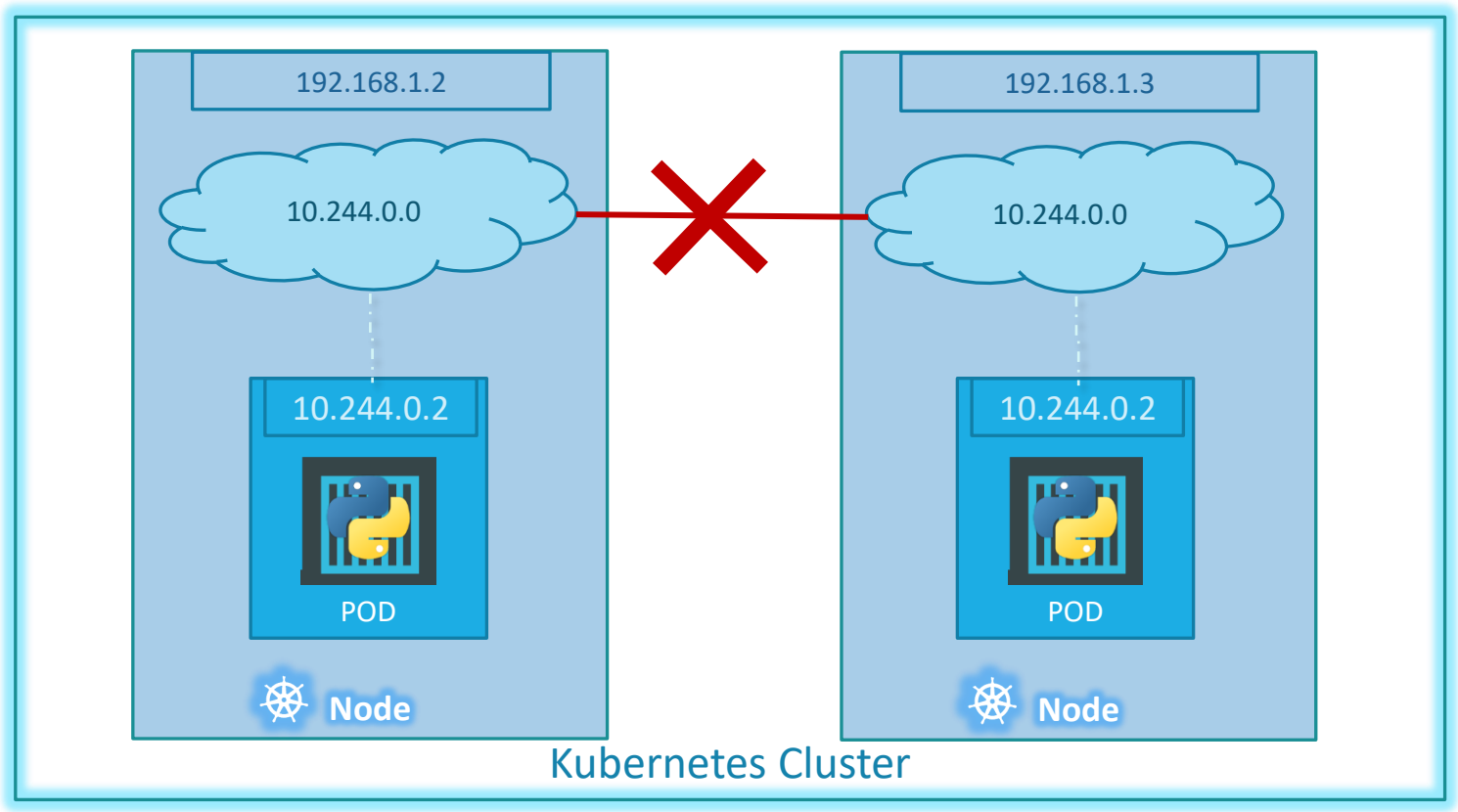
- IP Address is assigned to a POD





Cluster Networking

- All containers/PODs can communicate to one another without NAT
- All nodes can communicate with all containers and vice-versa without NAT

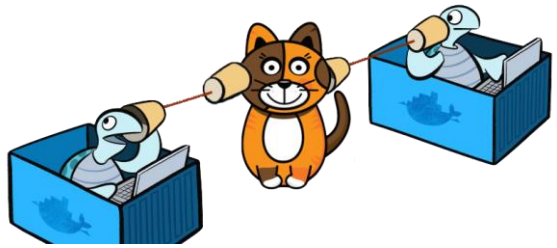



CISCO


cilium


flannel


vmware
NSX



Cluster Networking Setup

(3/4) Installing a pod network

You **MUST** install a pod network add-on so that your pods can communicate with each other.

The network must be deployed before any applications. Also, kube-dns, an internal helper service, will not start up before a network is installed. kubeadm only supports Container Network Interface (CNI) based networks (and does not support kubenet).

Several projects provide Kubernetes pod networks using CNI, some of which also support Network Policy. See the add-ons page for a complete list of available network add-ons. IPv6 support was added in CNI v0.6.0. CNI bridge and local-ipam are the only supported IPv6 network plugins in 1.9.

Note: kubeadm sets up a more secure cluster by default and enforces use of RBAC. Please make sure that the network manifest of choice supports RBAC.

You can install a pod network add-on with the following command:

```
kubectl apply -f <add-on.yaml>
```

NOTE: You can install **only one** pod network per cluster.

Choose one... | Calico | Canal | Flannel | Kube-router | Romana | Weave Net

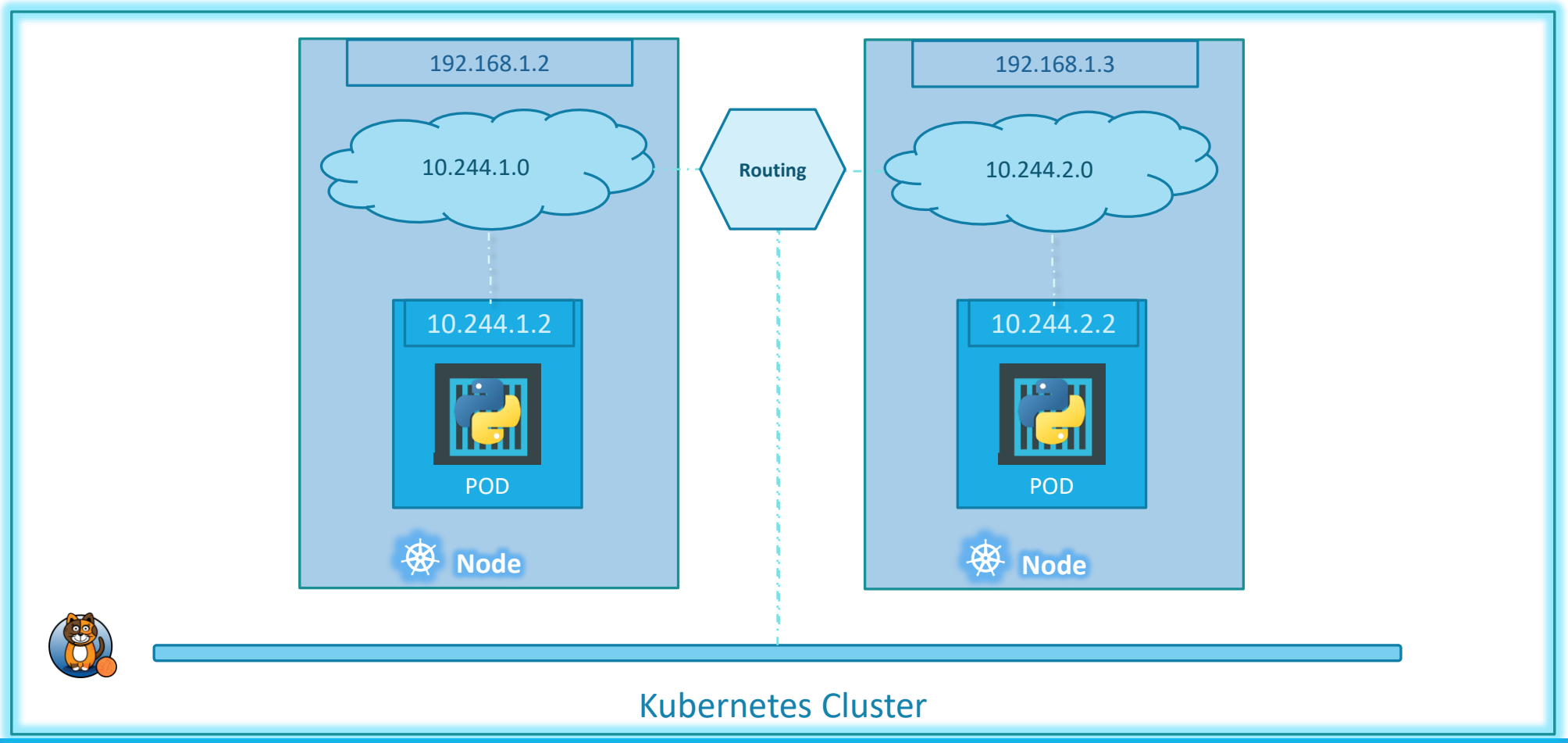
Refer to the Calico documentation for a kubeadm quickstart, a kubeadm installation guide, and other resources.

Note:

- In order for Network Policy to work correctly, you need to pass `--pod-network-cidr=192.168.0.0/16` to `kubeadm init`.
- Calico works on `amd64` only.

```
kubectl apply -f https://docs.projectcalico.org/v3.0/getting-started/kubernetes/installation/hosted/kubeadm/1.7/calico.yaml
```

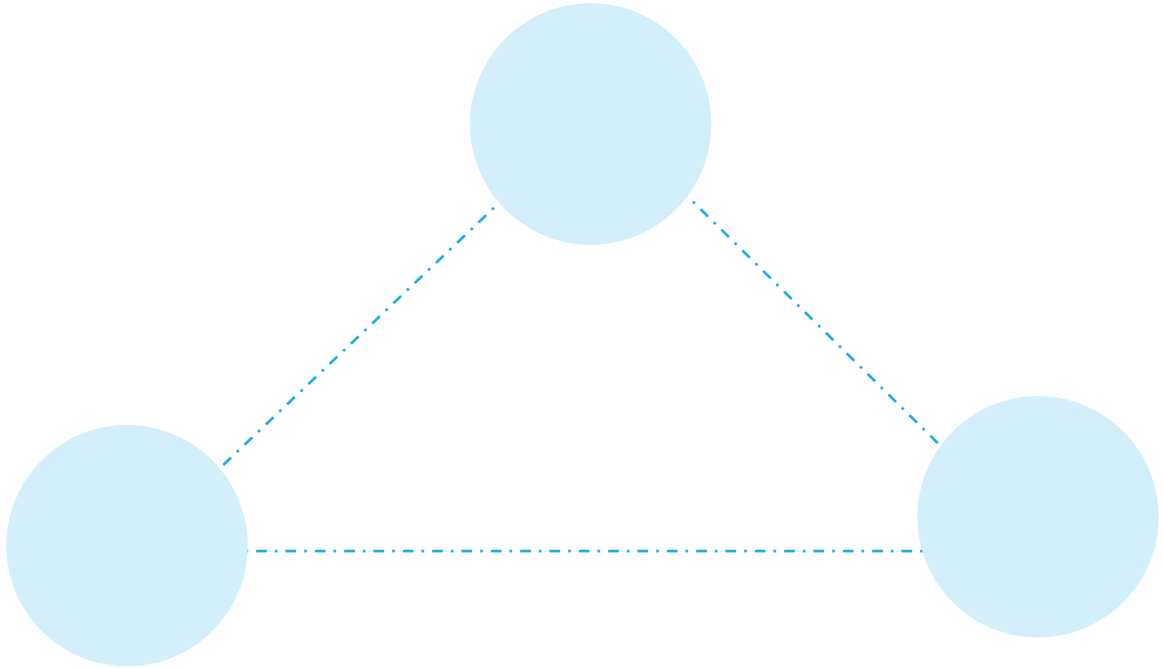
Cluster Networking



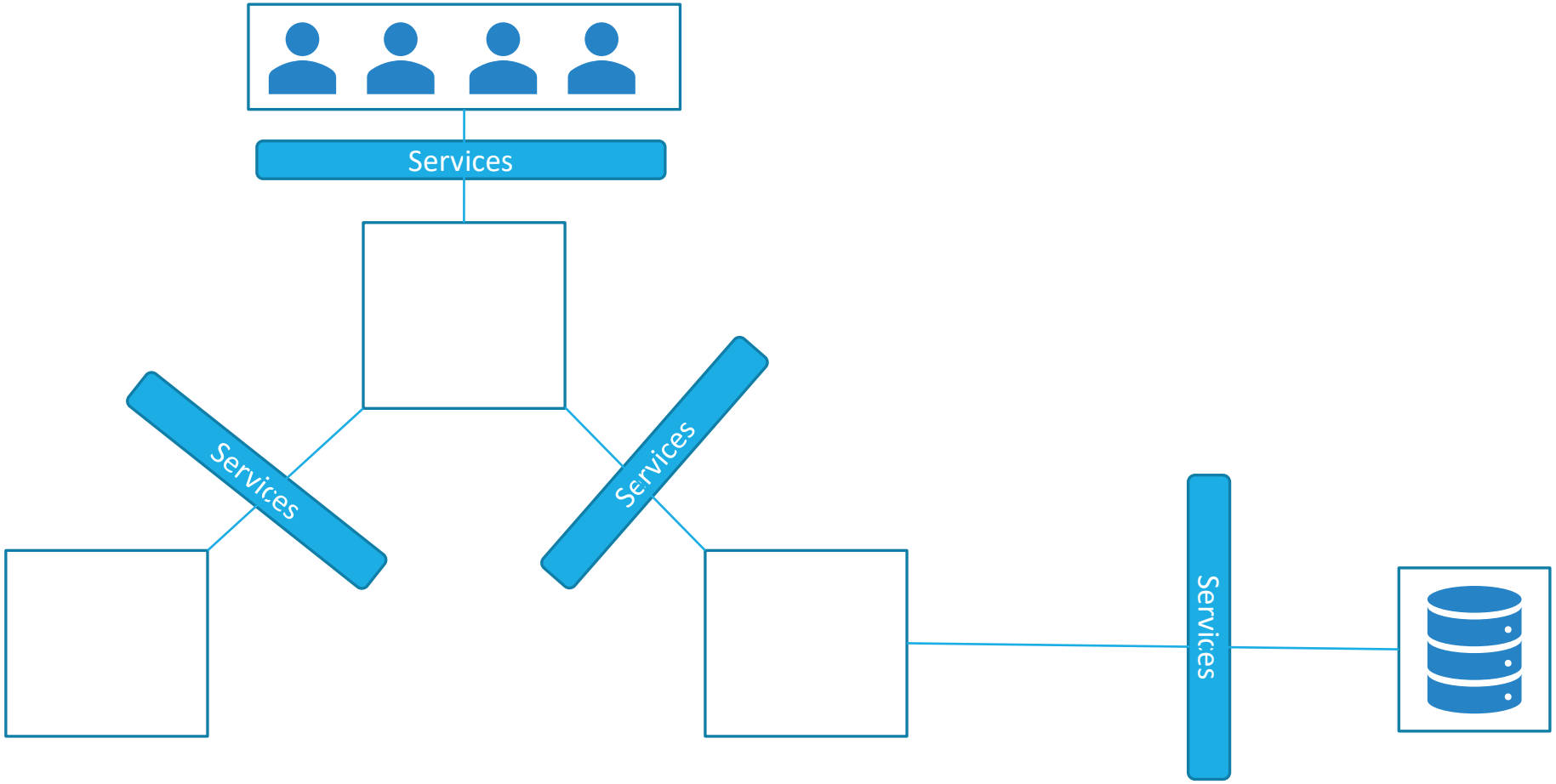
Demo

Networking

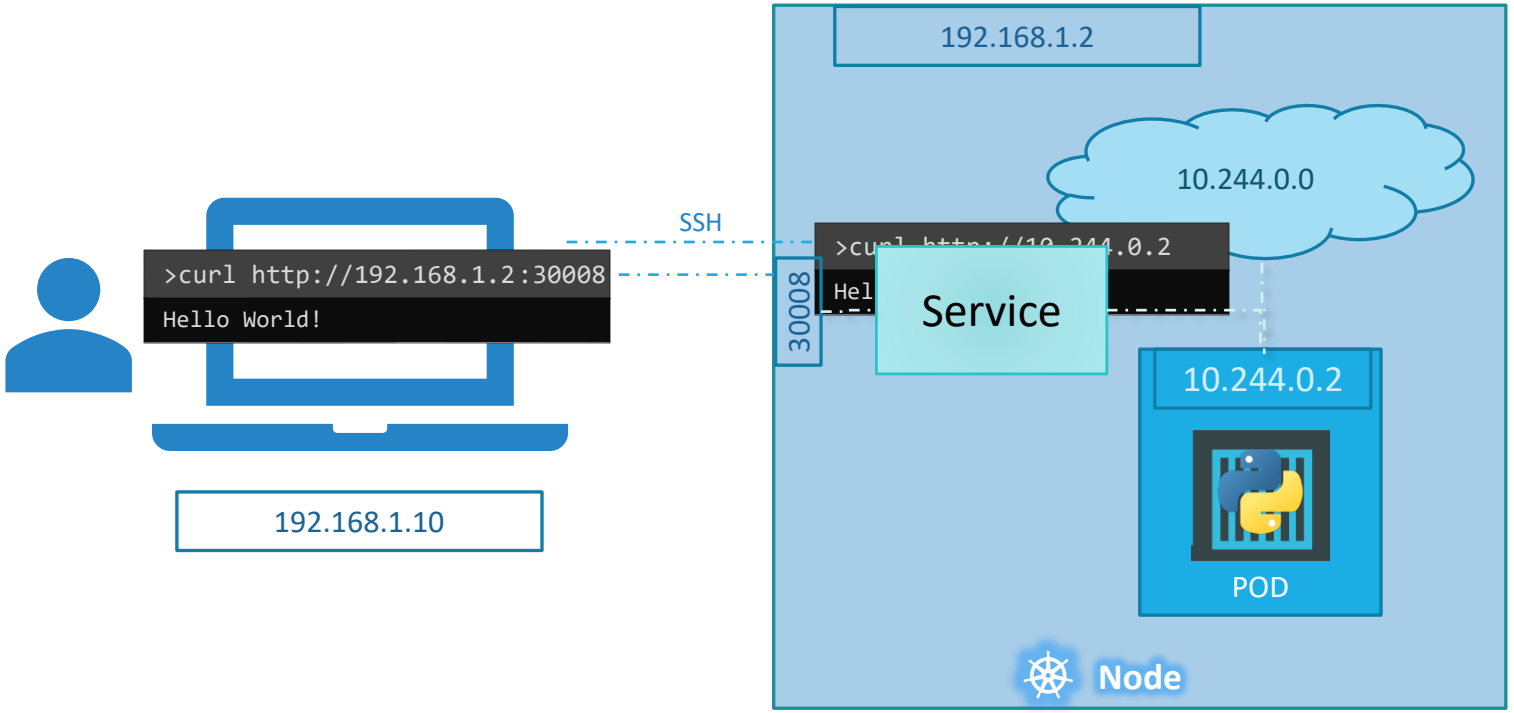
Services



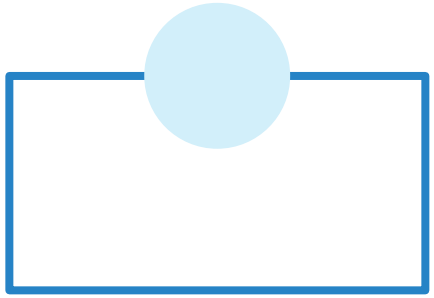
Services



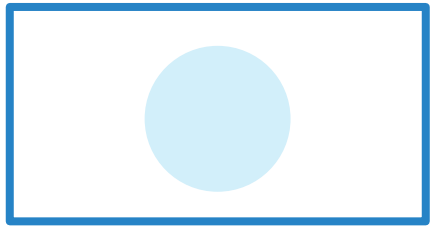
Service


>curl http://192.168.1.2:30008
Hello World!
192.168.1.10
SSH
192.168.1.2
10.244.0.0
30008
Service
10.244.0.2
POD
Node

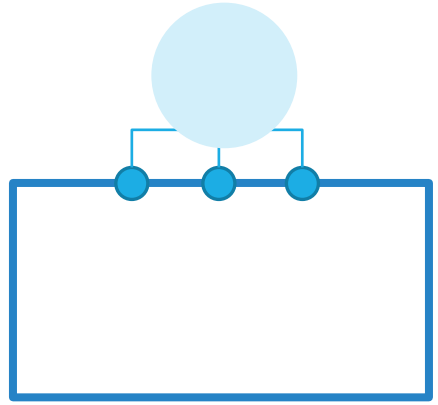
Services Types



NodePort

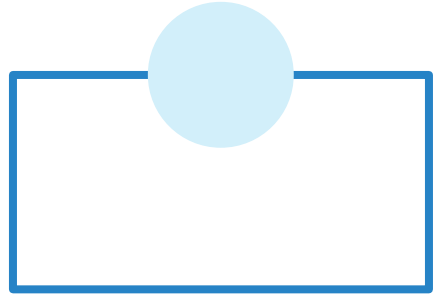


ClusterIP

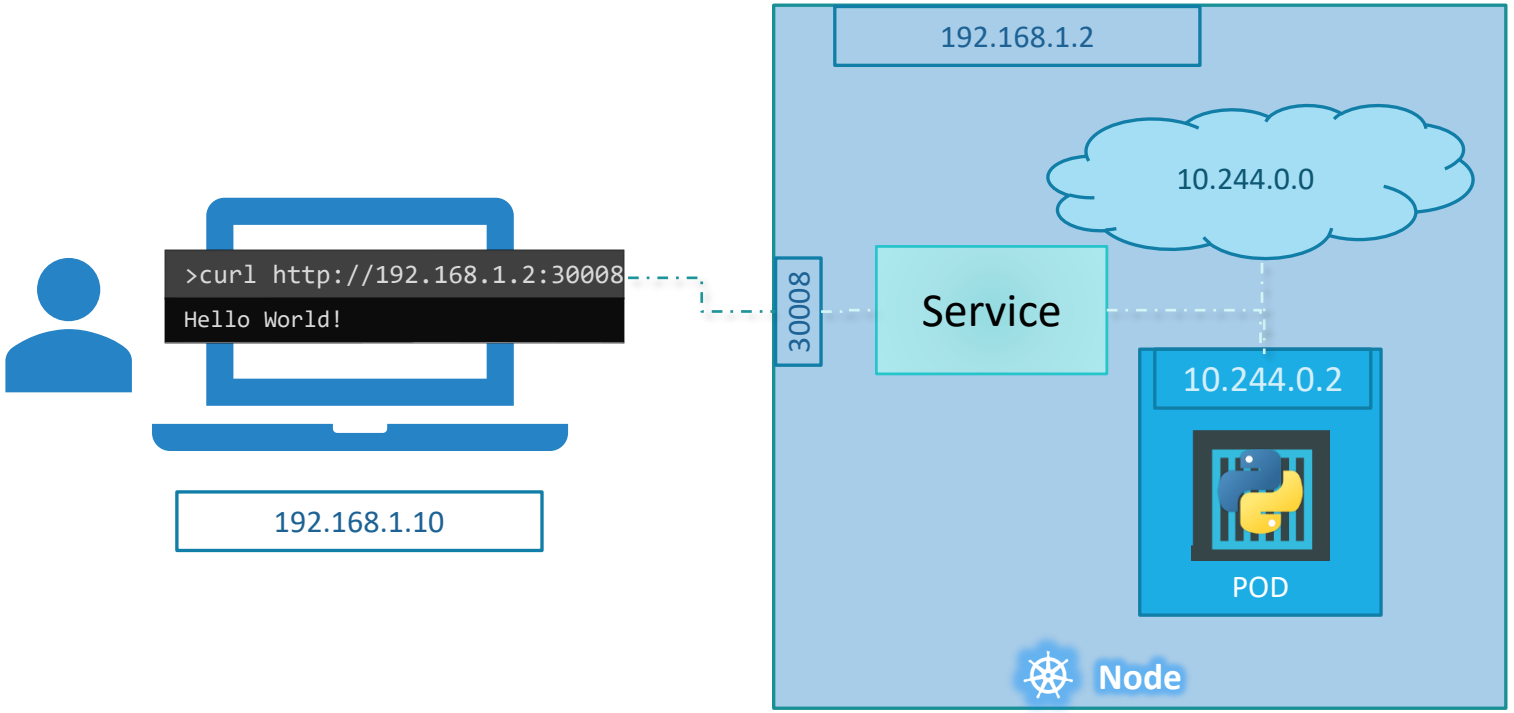


LoadBalancer

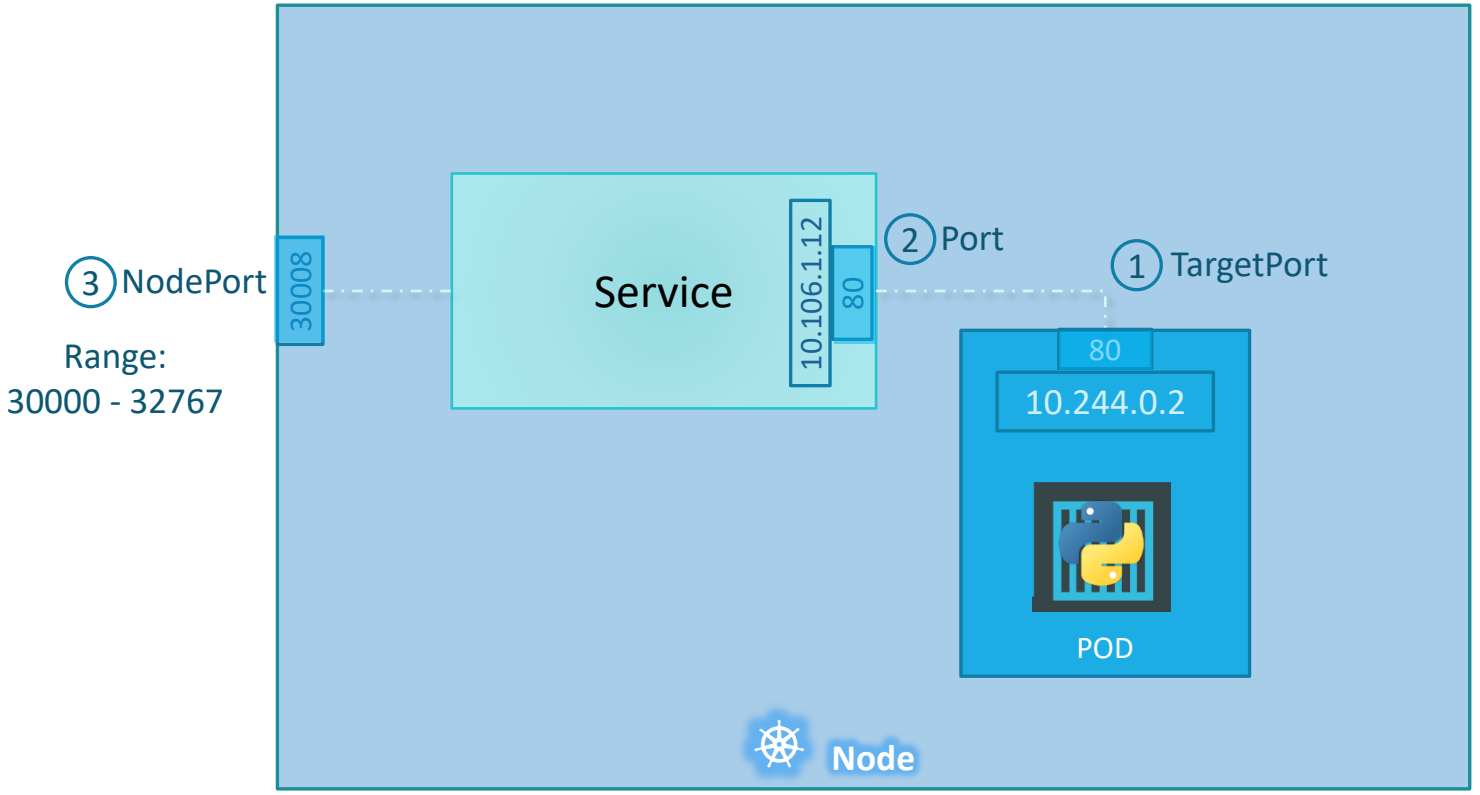
NodePort



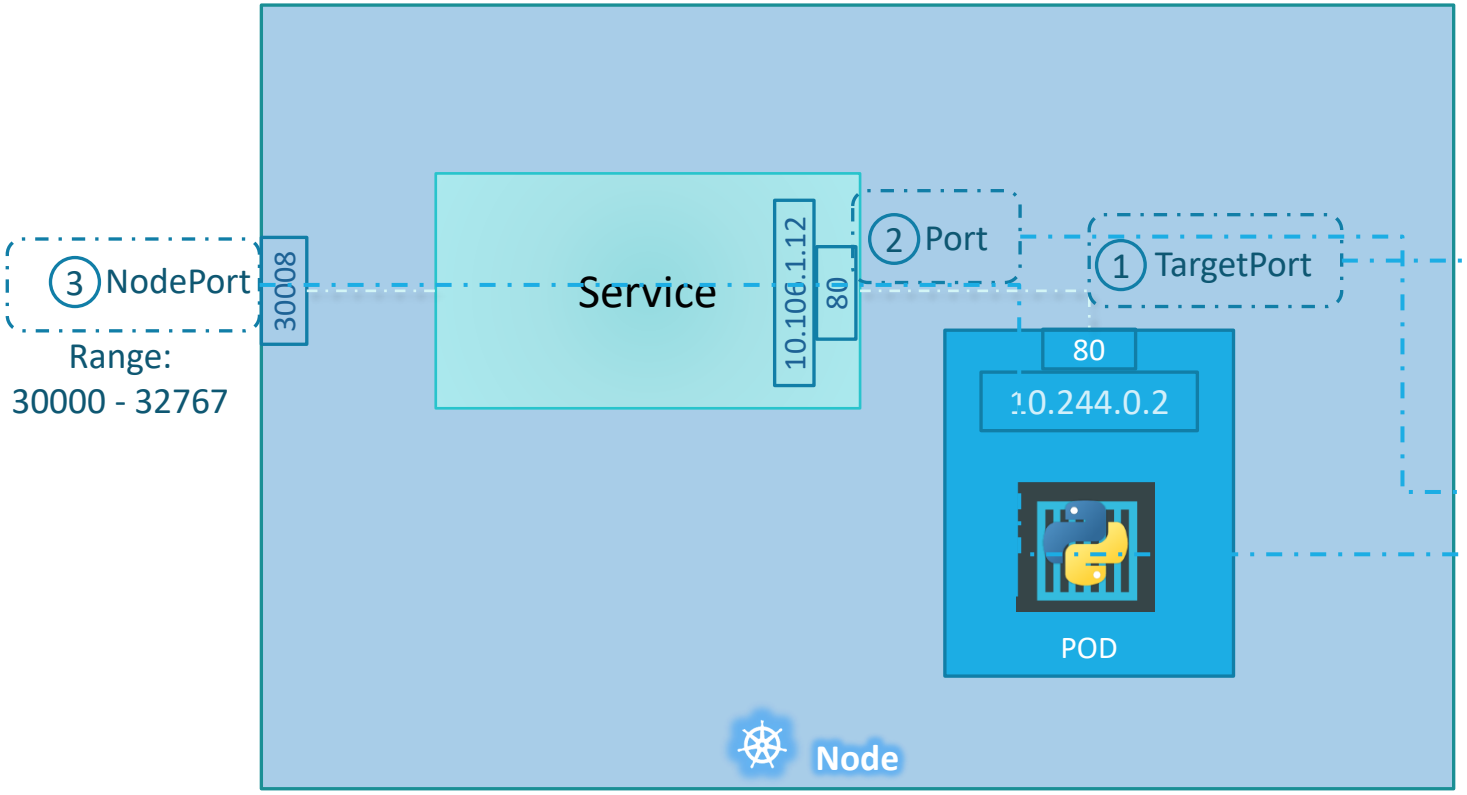
Service - NodePort



Service - NodePort



Service - NodePort



service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
      - targetPort: 80
        port: 80
        nodePort: 30008
```

Service - NodePort

service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
     - targetPort: 80
       port: 80
       nodePort: 30008
    selector:
```

pod-definition.yml

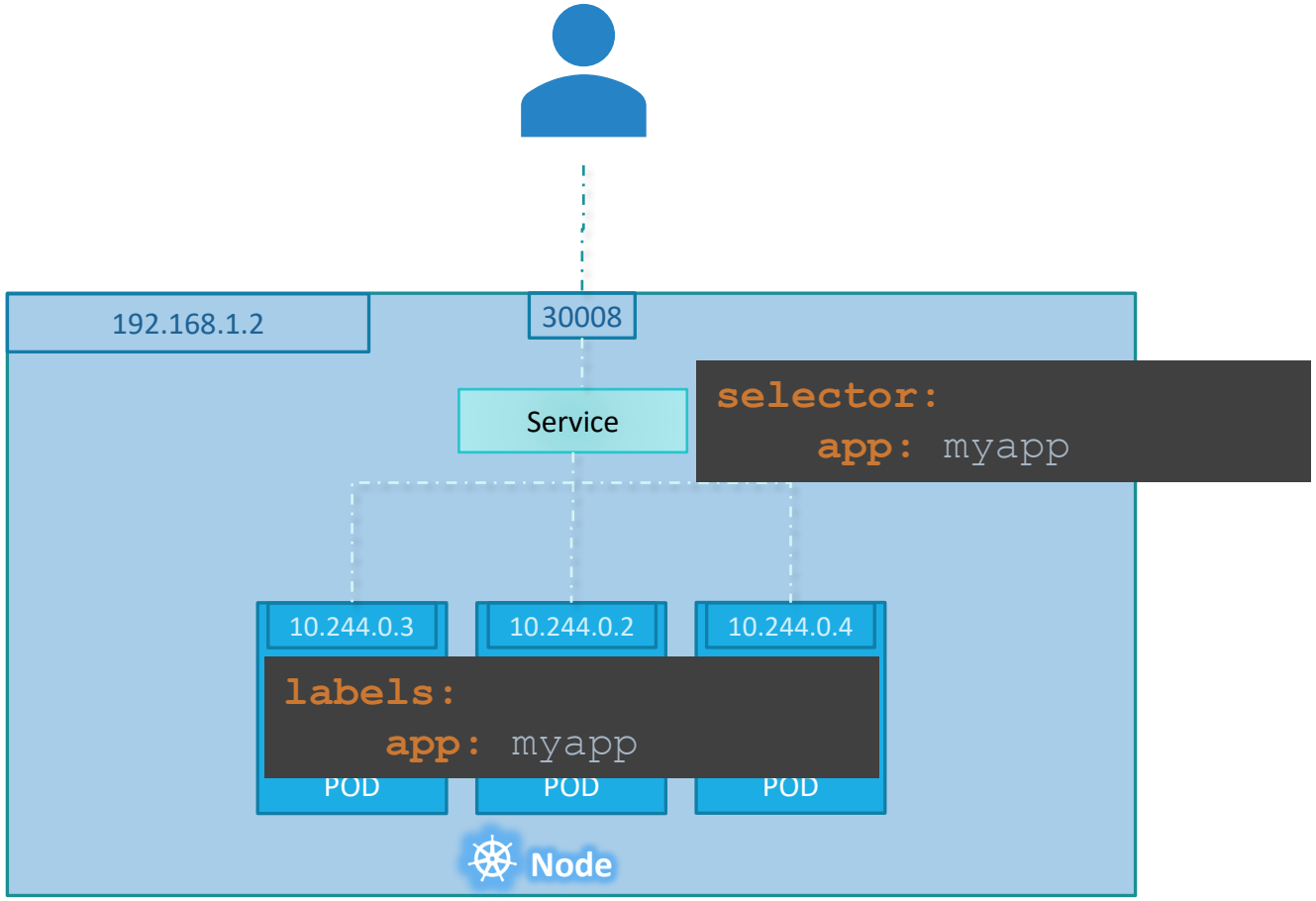
```
> kubectl create -f service-definition.yml
service "myapp-service" created

> kubectl get services
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        16d
myapp-service   NodePort    10.106.127.123   <none>        80:30008/TCP   5m
```

```
app: myapp
```

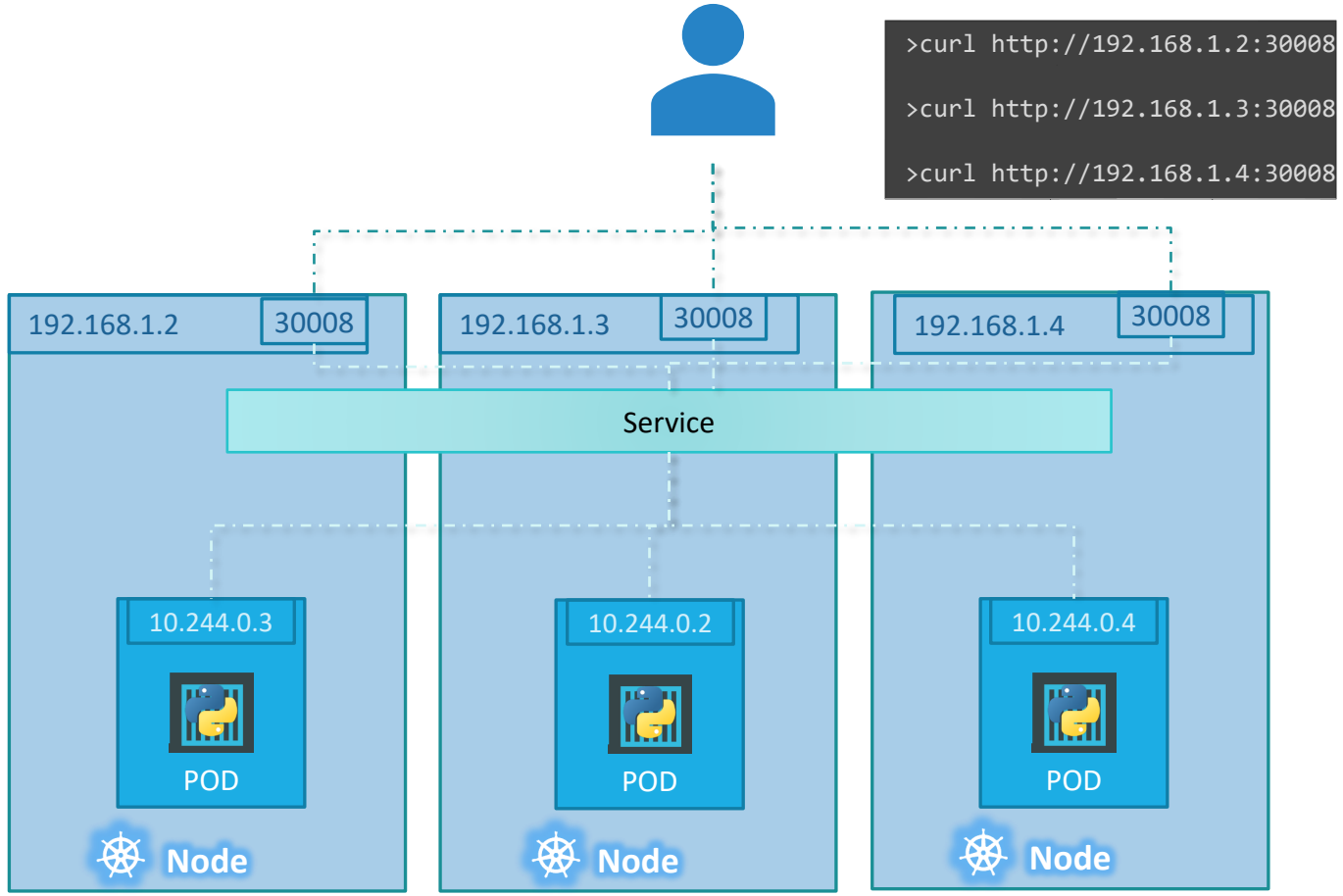
```
> curl http://192.168.1.2:30008
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
```

Service - NodePort

Algorithm: Random
SessionAffinity: Yes



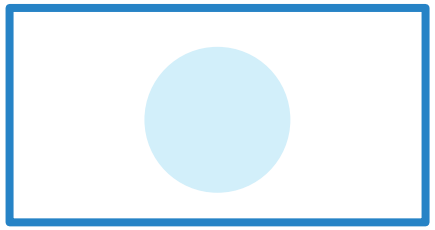
Service - NodePort



Demo

Service - NodePort

ClusterIP



ClusterIP

front-end

back-end

redis