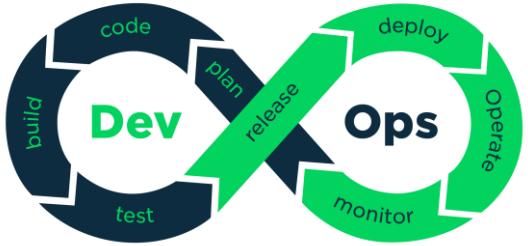
DevOps – Let the Journey Begin



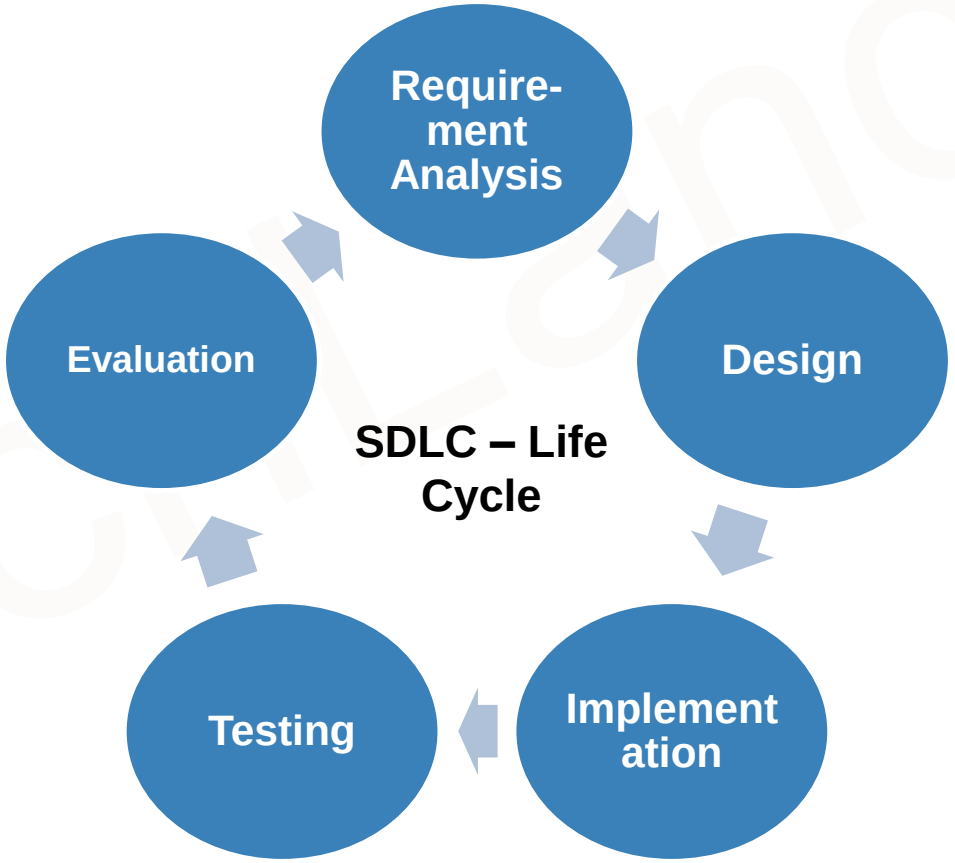
Raman Khanna

DevOps

What is DevOps?

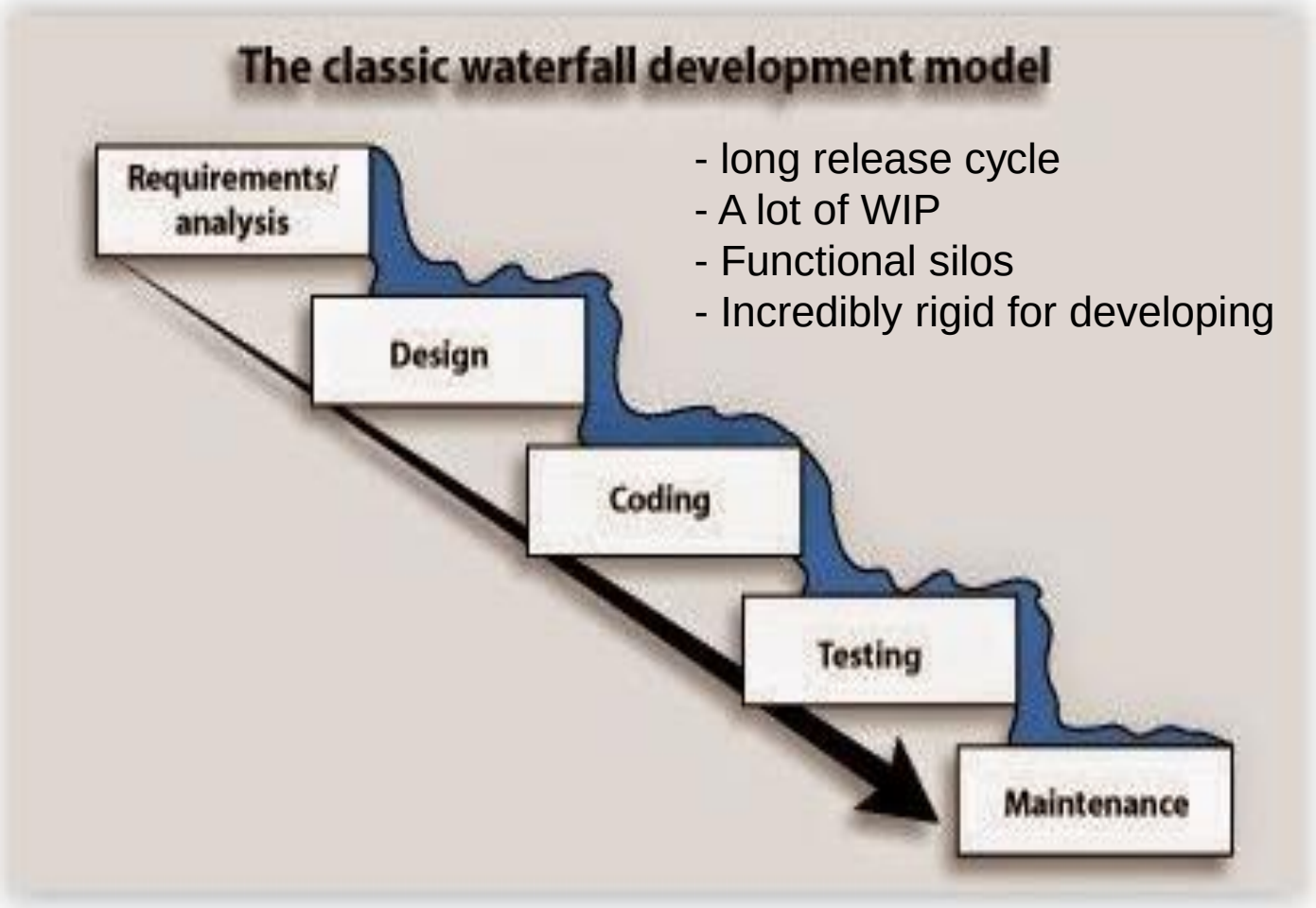
SDLC Model

- 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

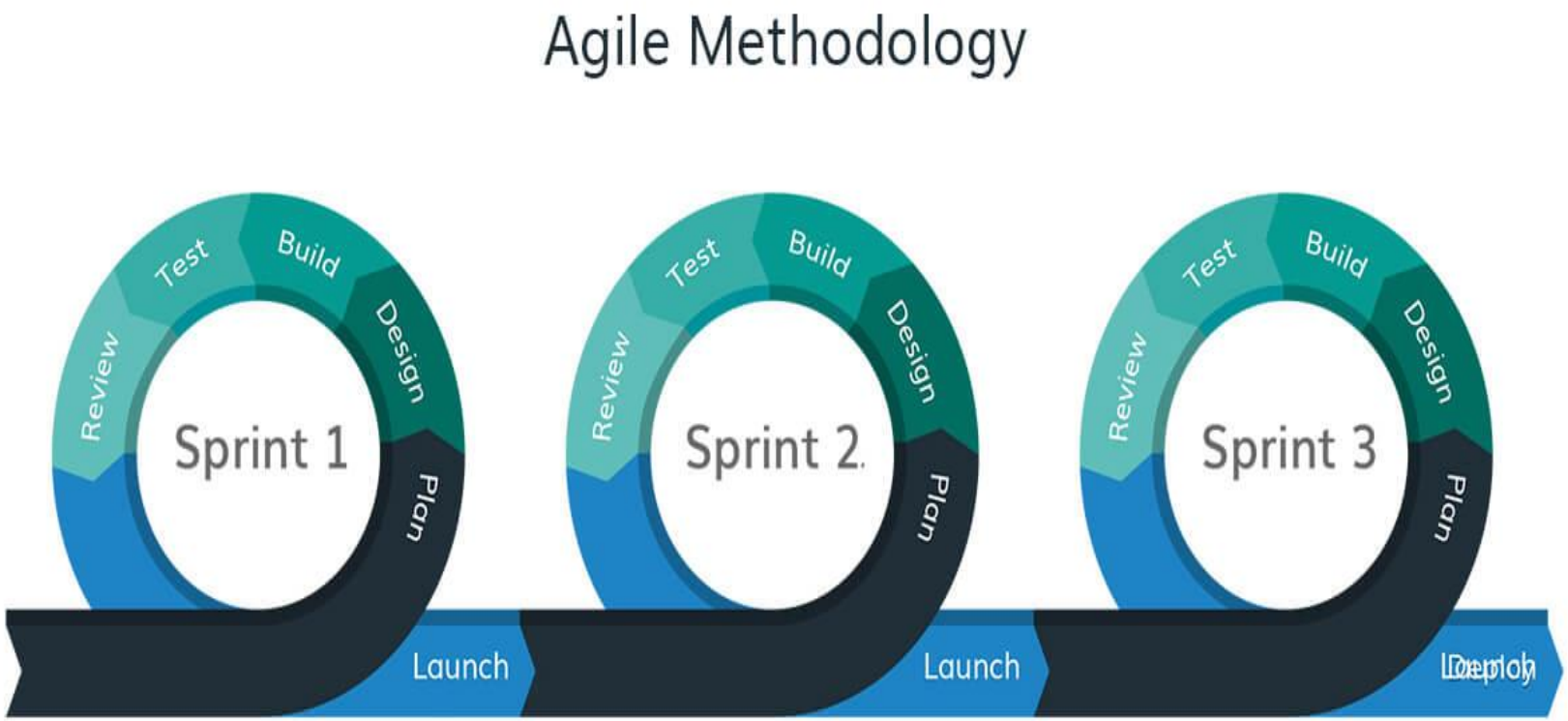


Waterfall Model

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

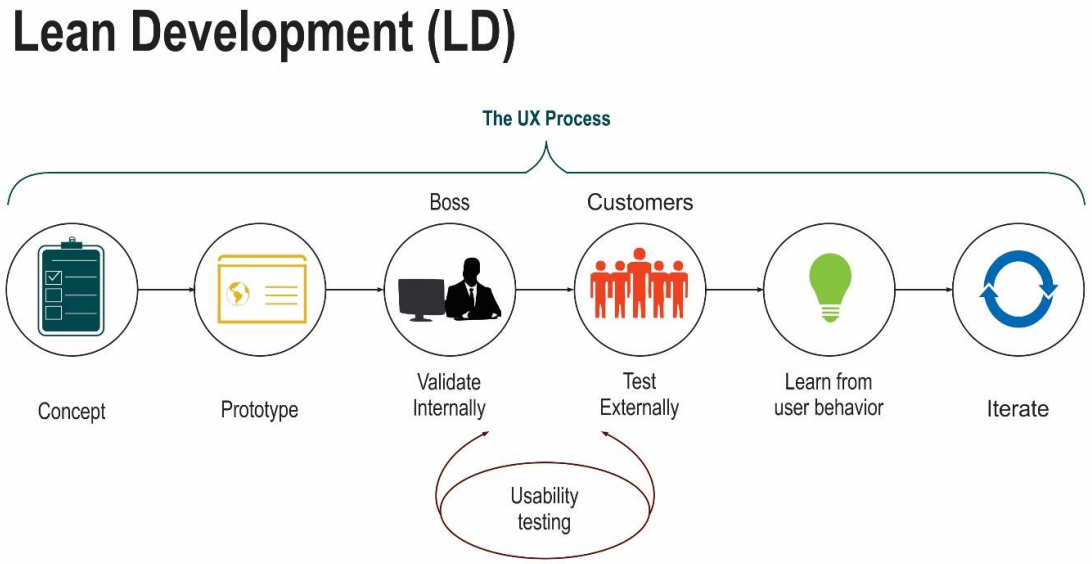


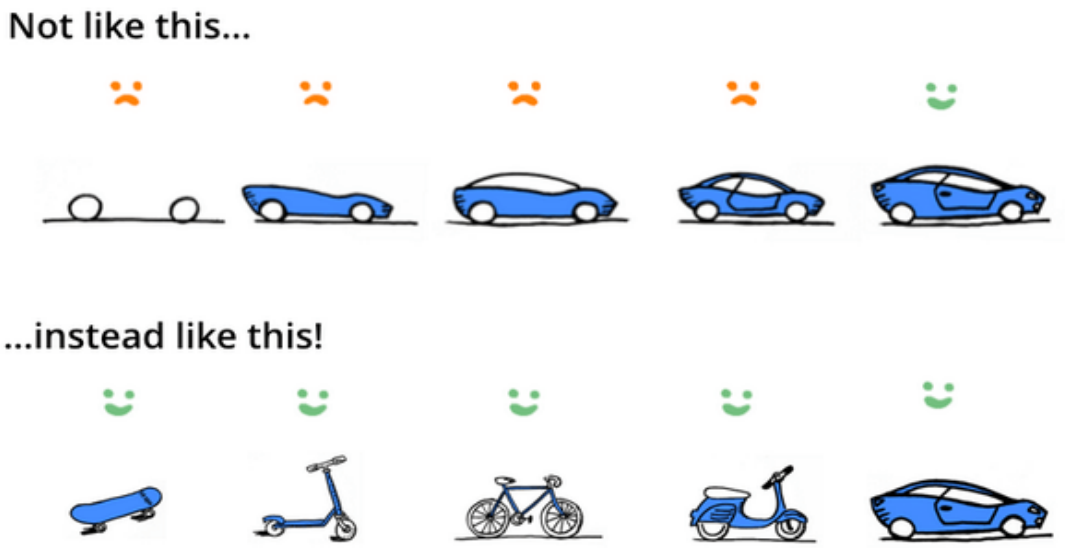
Agile



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

Lean Development

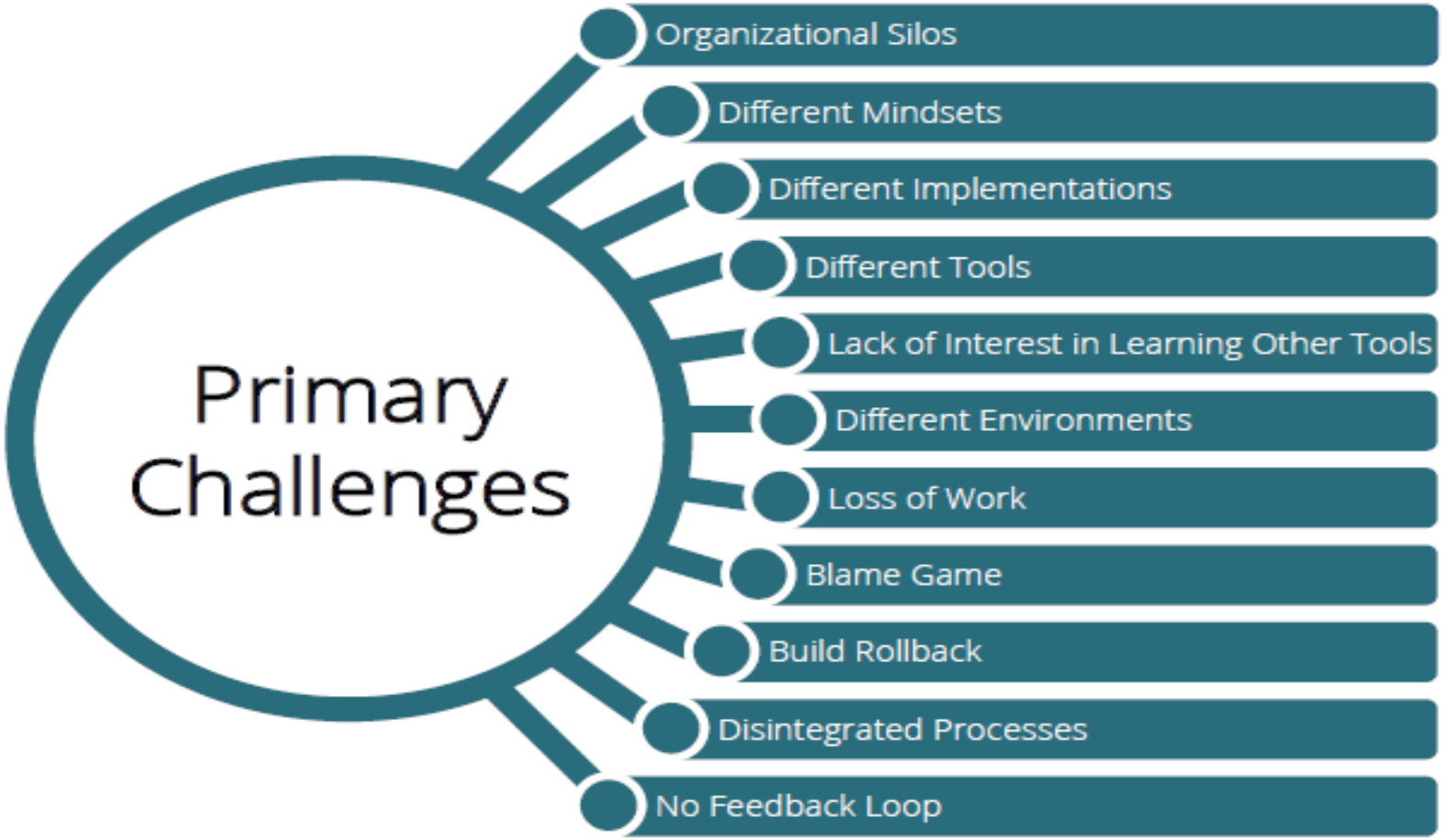




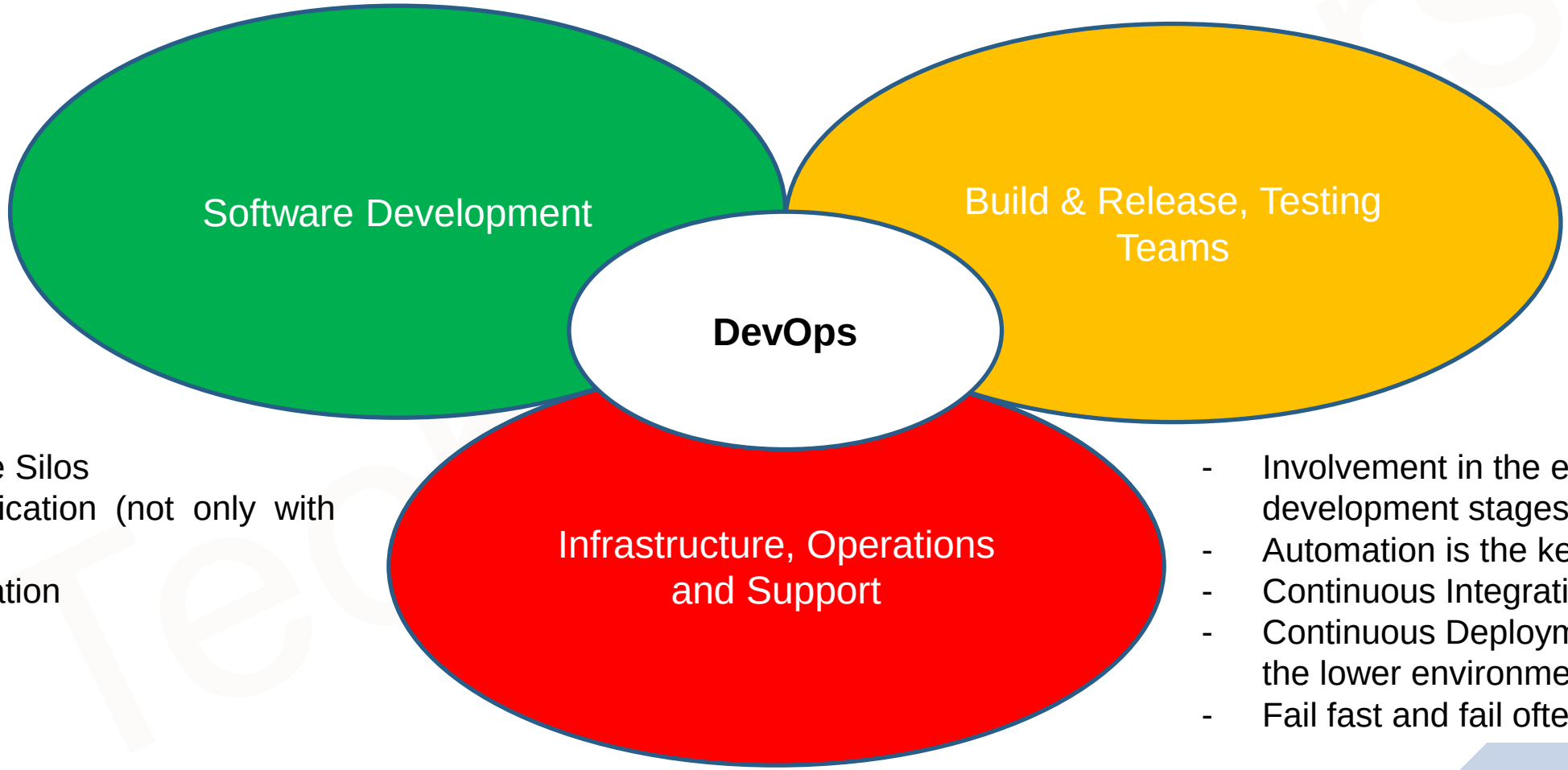
- Suddenly ops was the bottleneck (more release less people), again WIP is more!

Challenges

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



DevOps

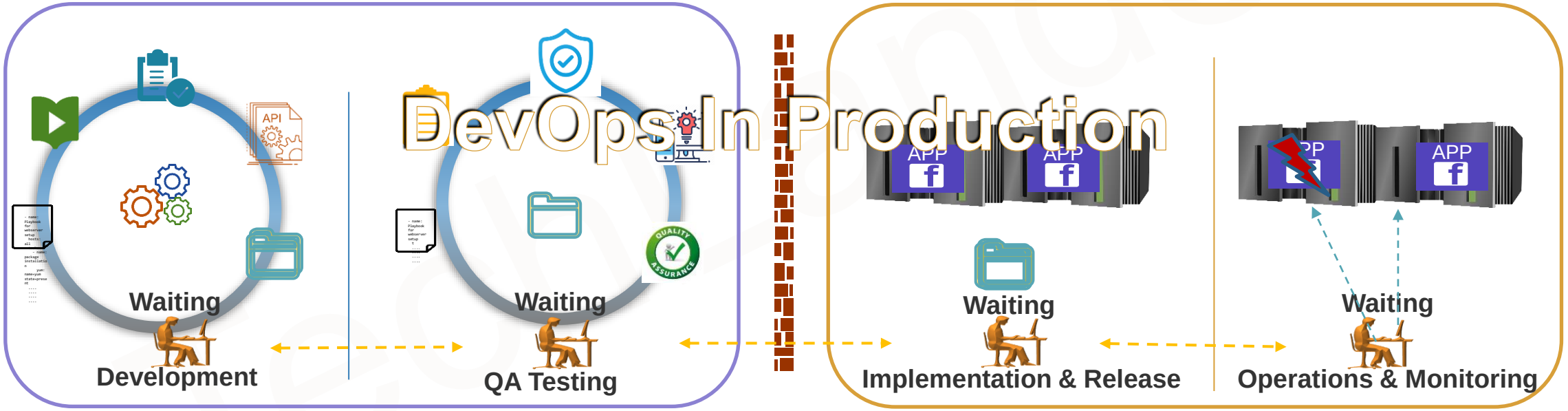


- Break the Silos
- Communication (not only with emails)
- Collaboration
- Trust

- 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

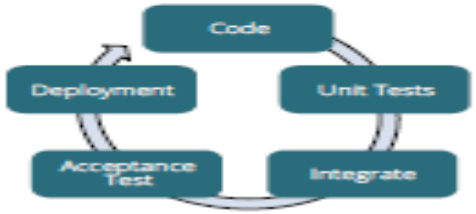
1. Customer-Centric Action

4. Cross-Functional Autonomous Teams

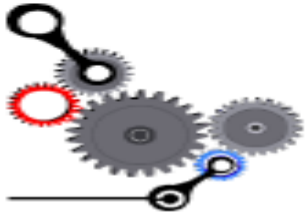
2. Create with the End in Mind

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

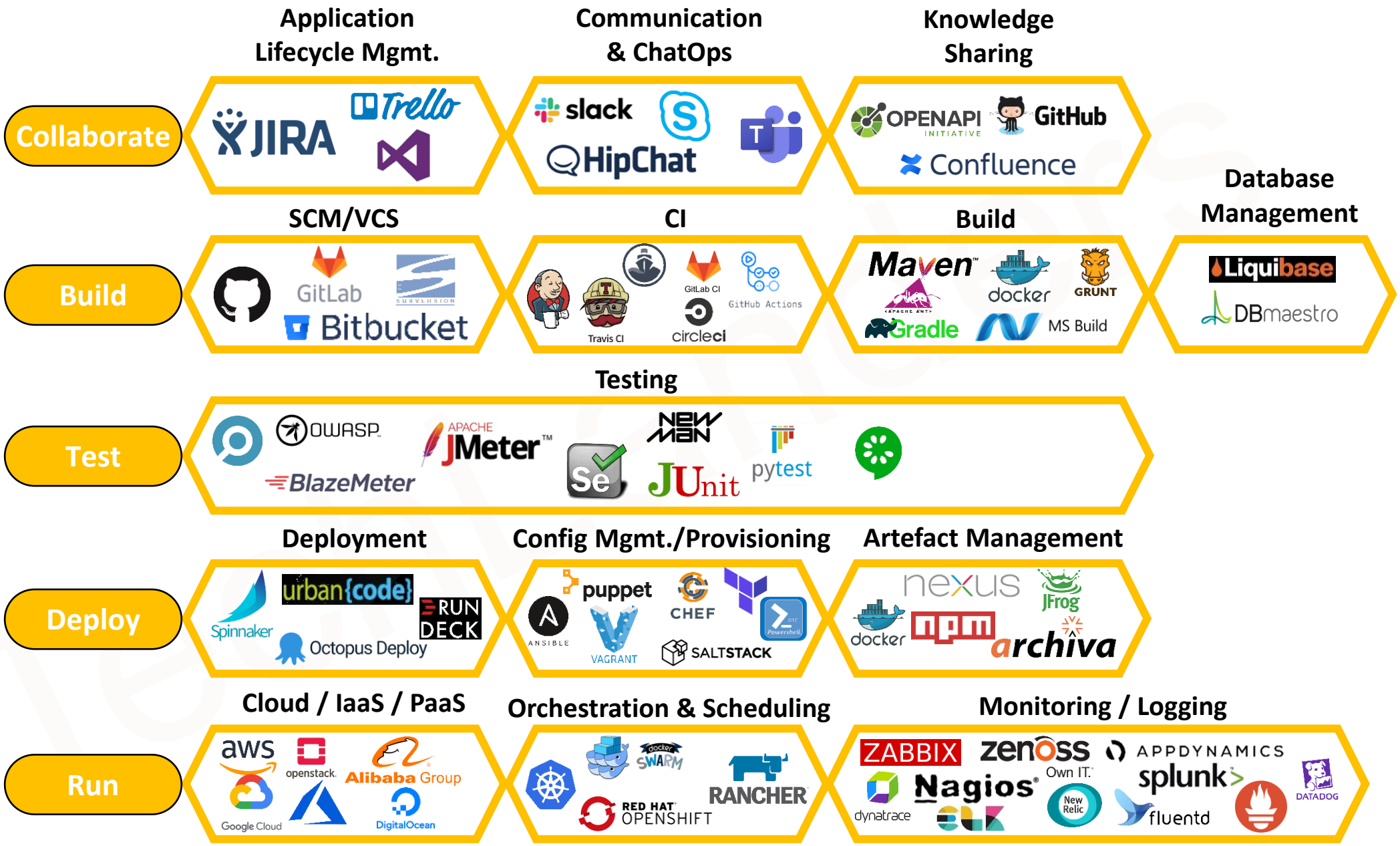
Cool tools can attract and retain

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.

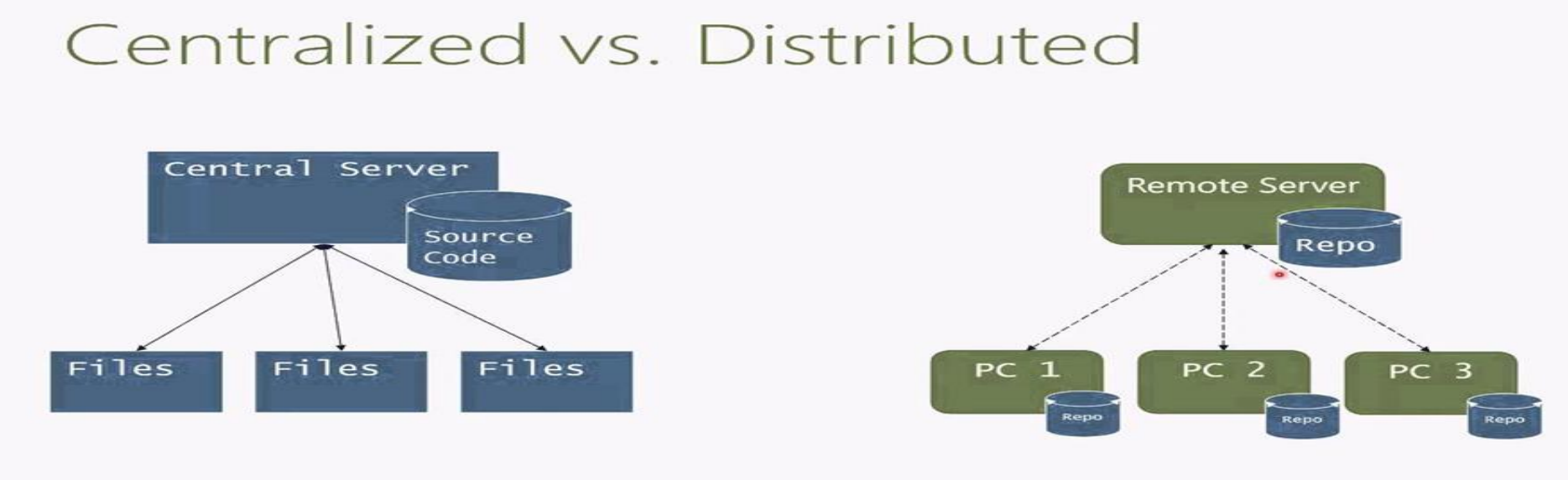
Its not just for code, it also helps in

- 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]#

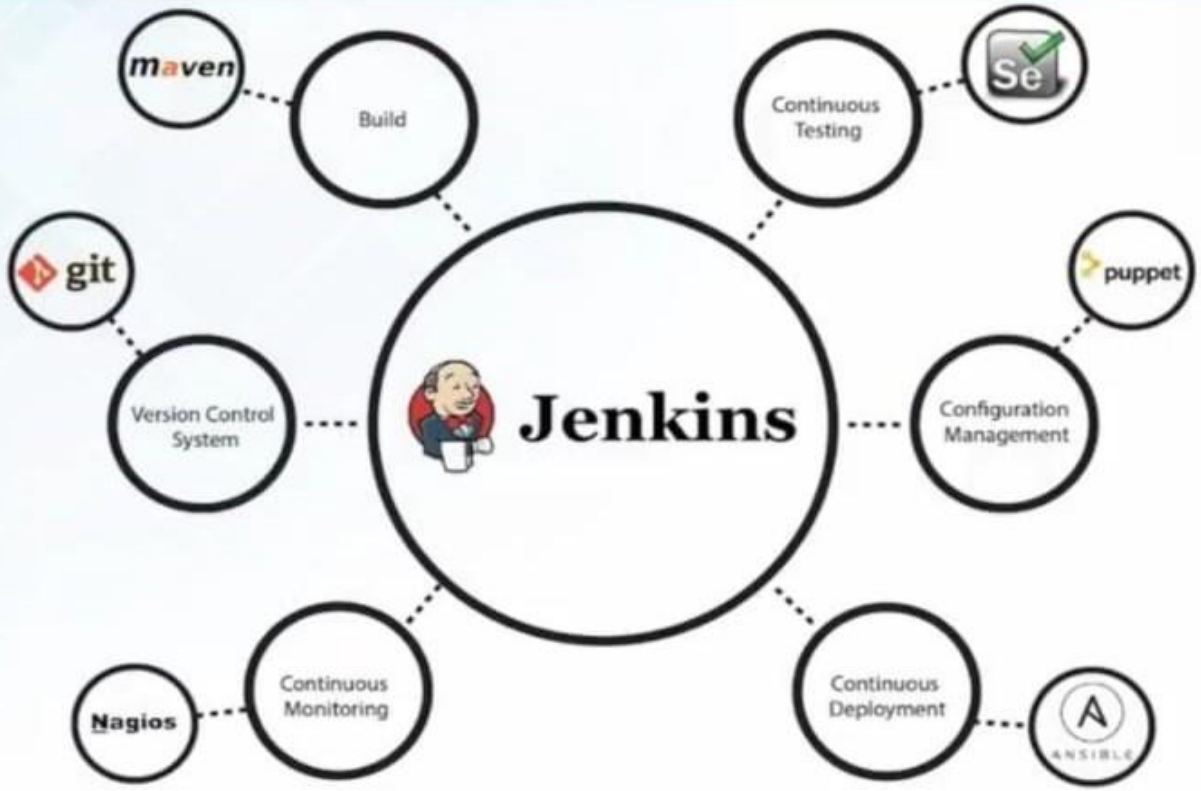
Github/Bitbucket

- **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

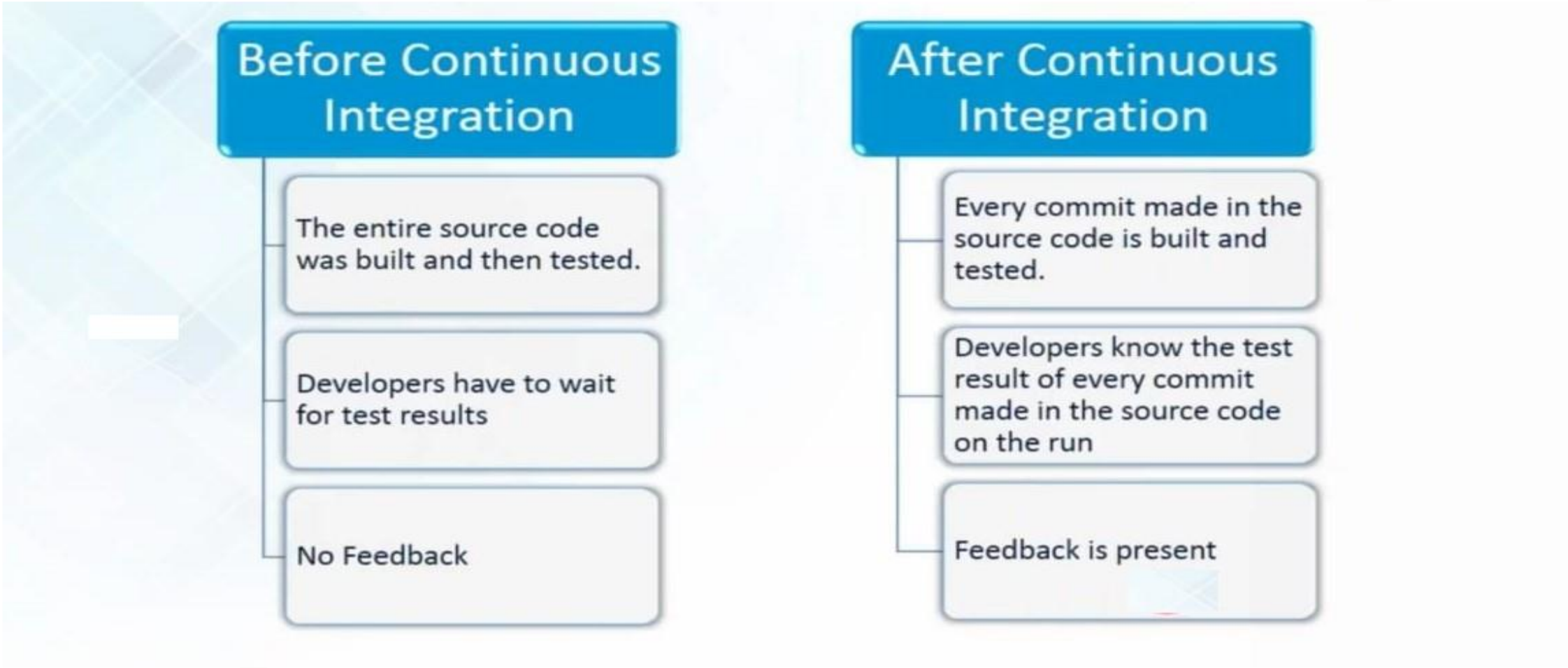
Continuous Integration with Jenkins

What is Jenkins?

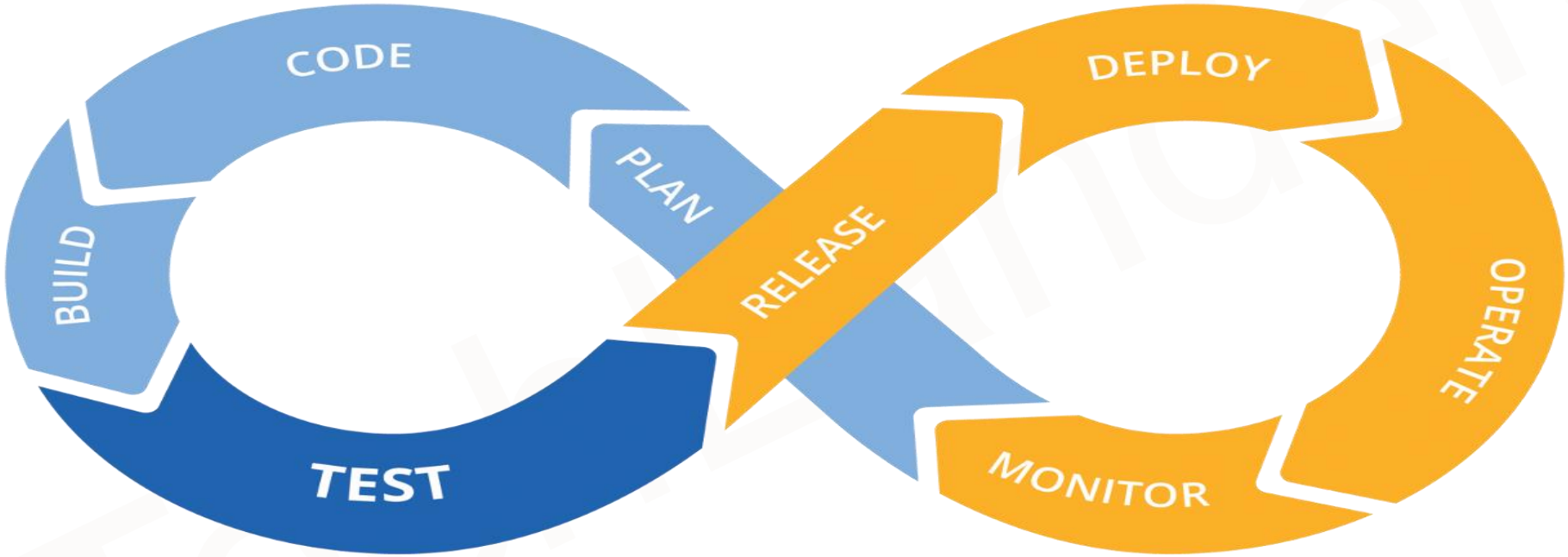
Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Plugins allows integration of various DevOps stages.



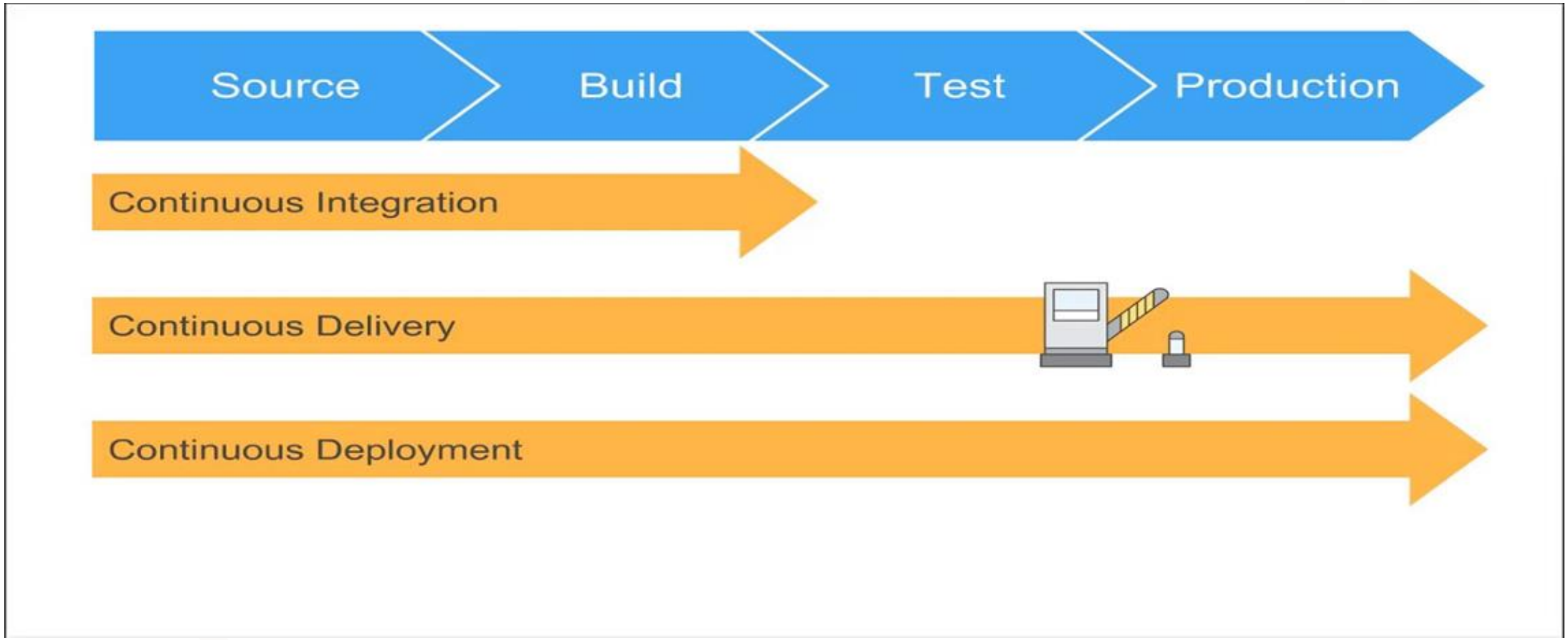
What is Continuous Integration?



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

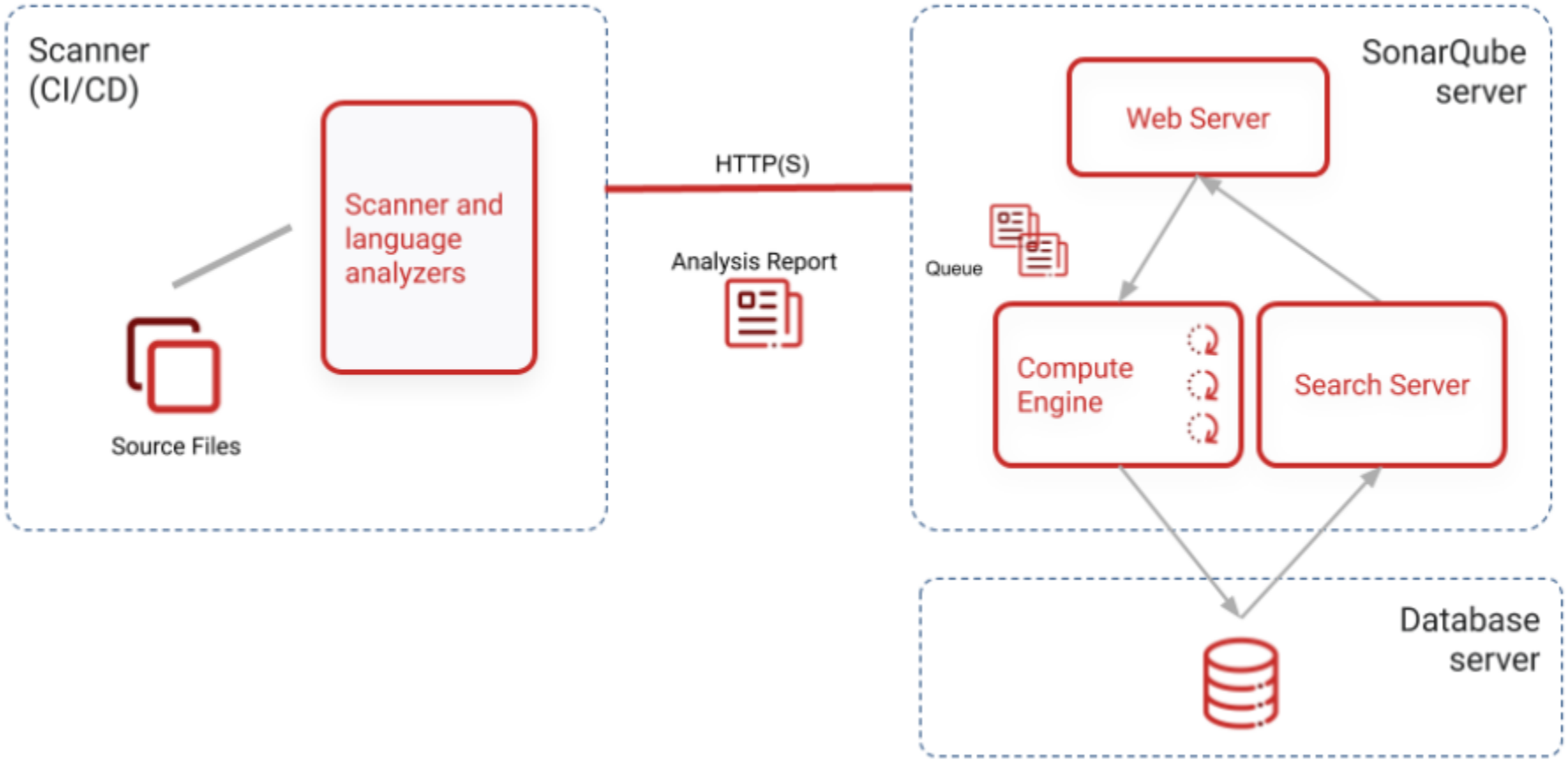
Analyzes all branches of a VCS

Discover Memory Leaks

Quality gates and Quality profiles for customized requirements

Good Visualization.

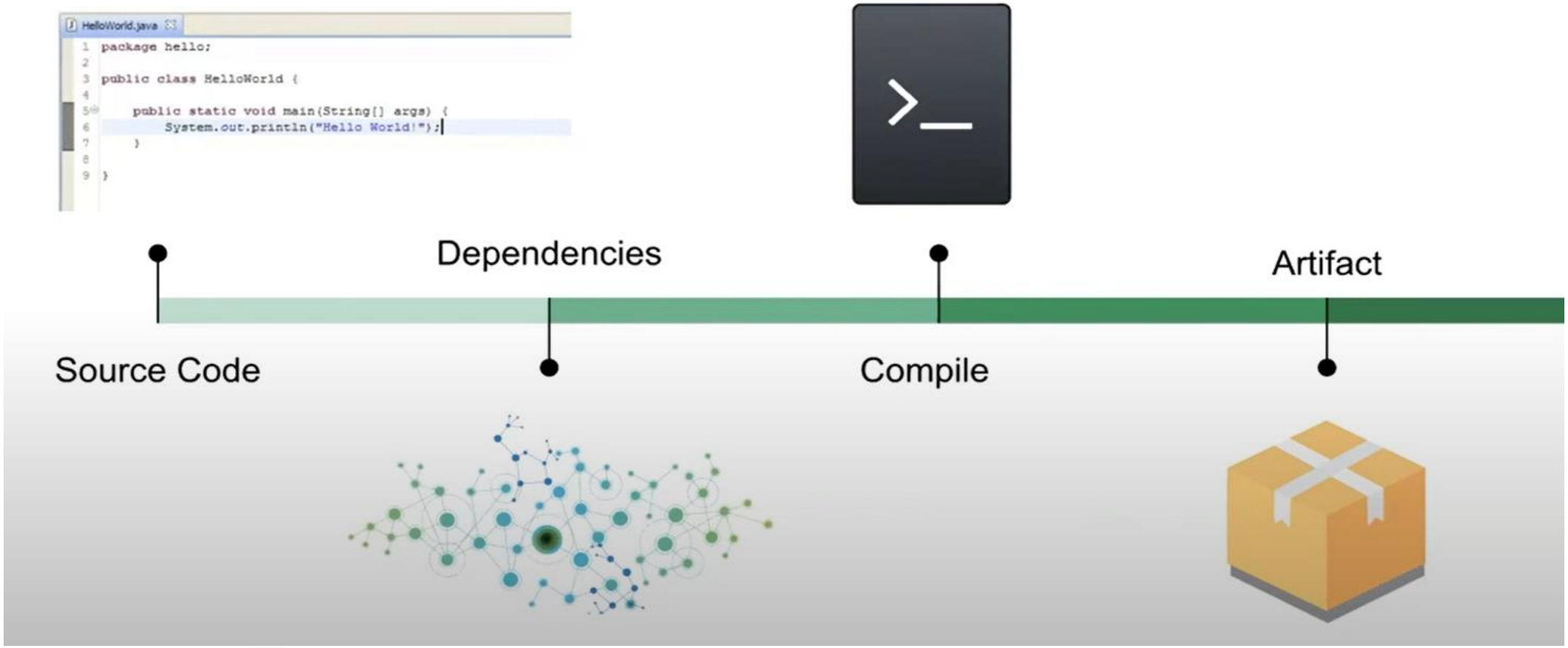
SonarQube Components



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

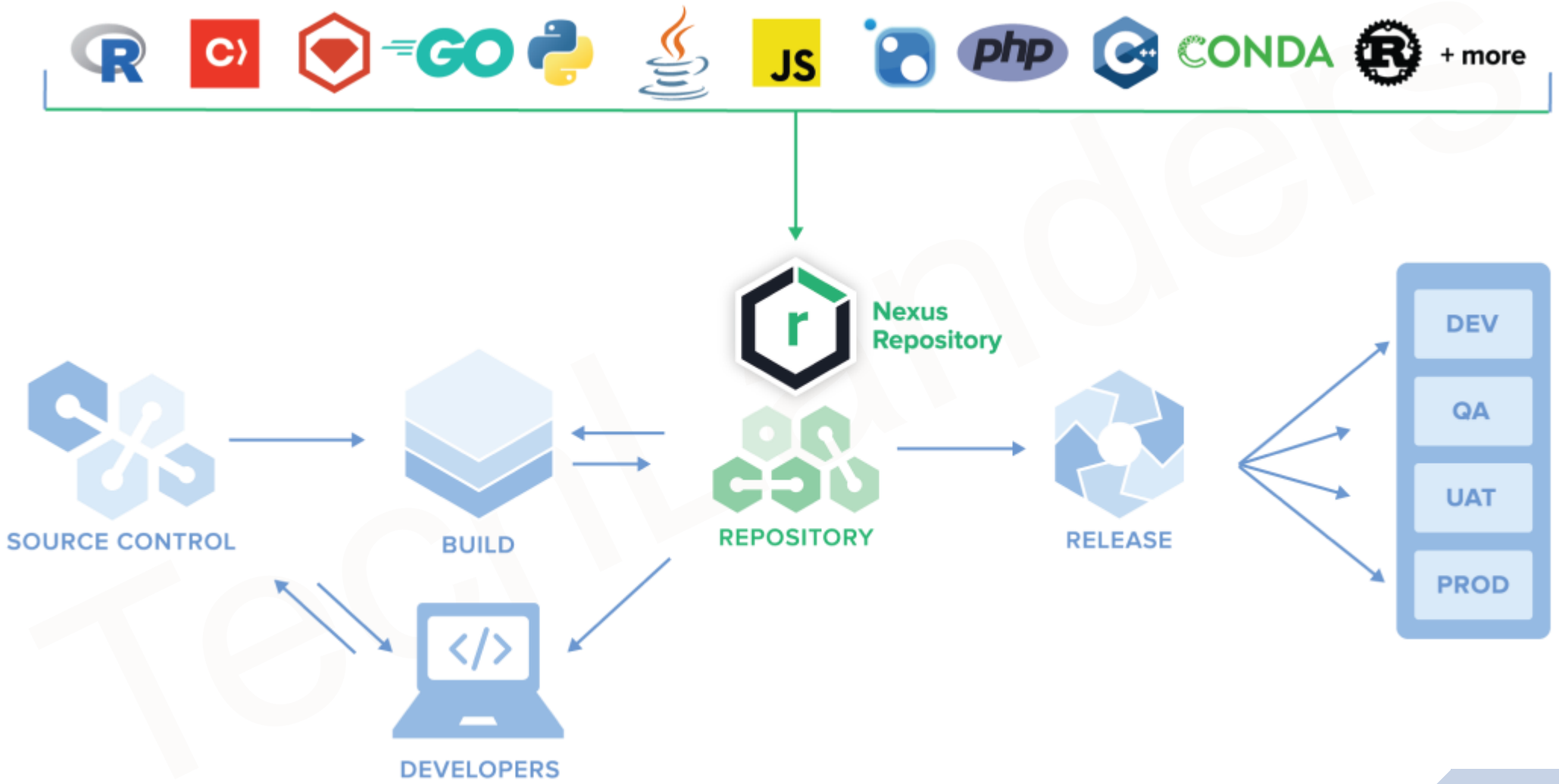
Manage them at remote Central Location

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.

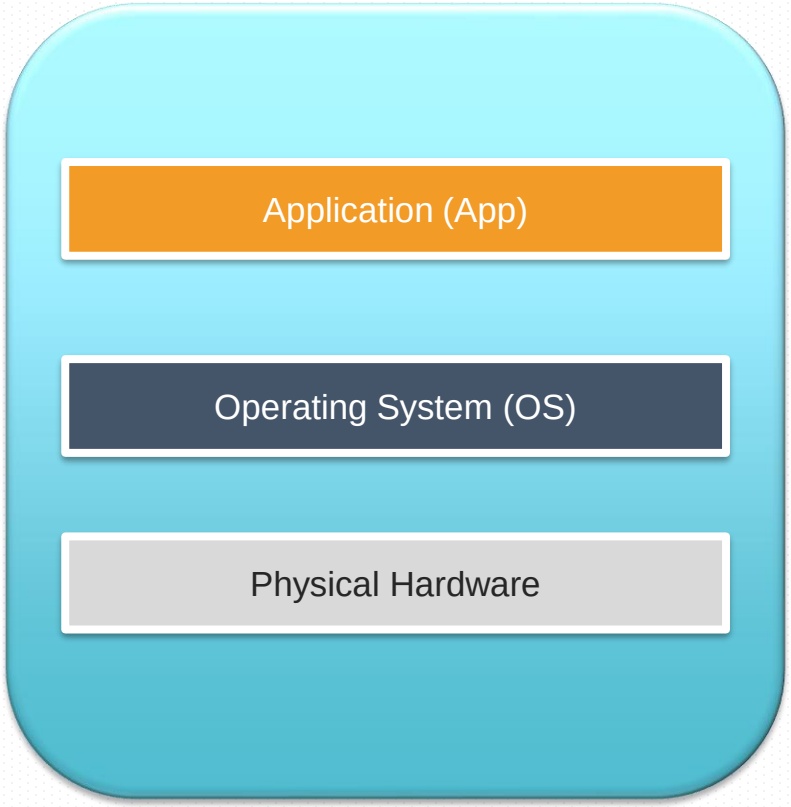
A History Lesson

- In the traditional ages of development and deployment of application

Unused Resources

Difficult Migrations

Vendor dependency



Slow Deployment time

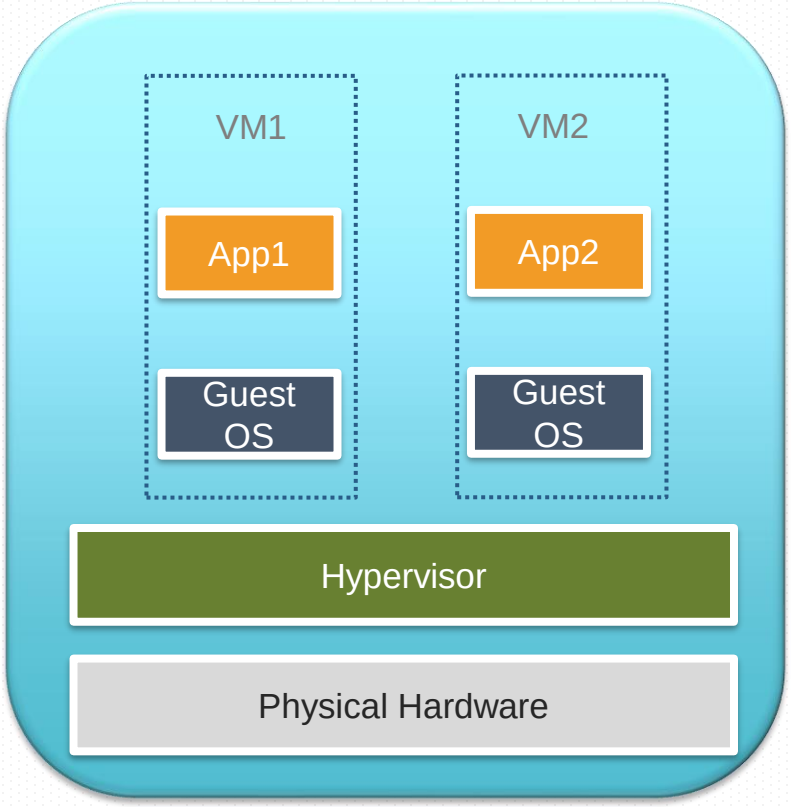
Difficult to Scale

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

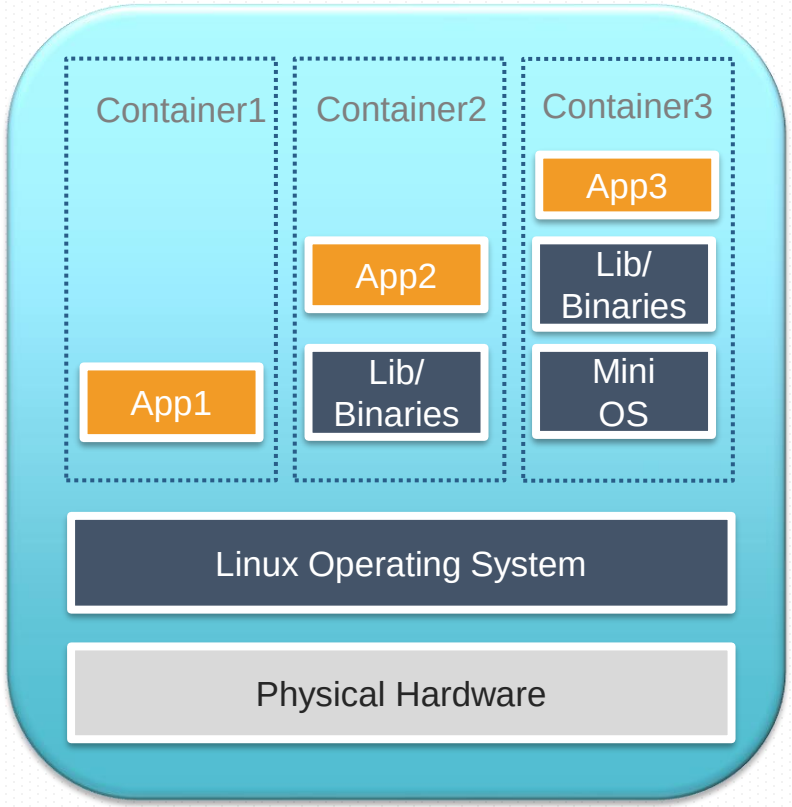
Less OverHead

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

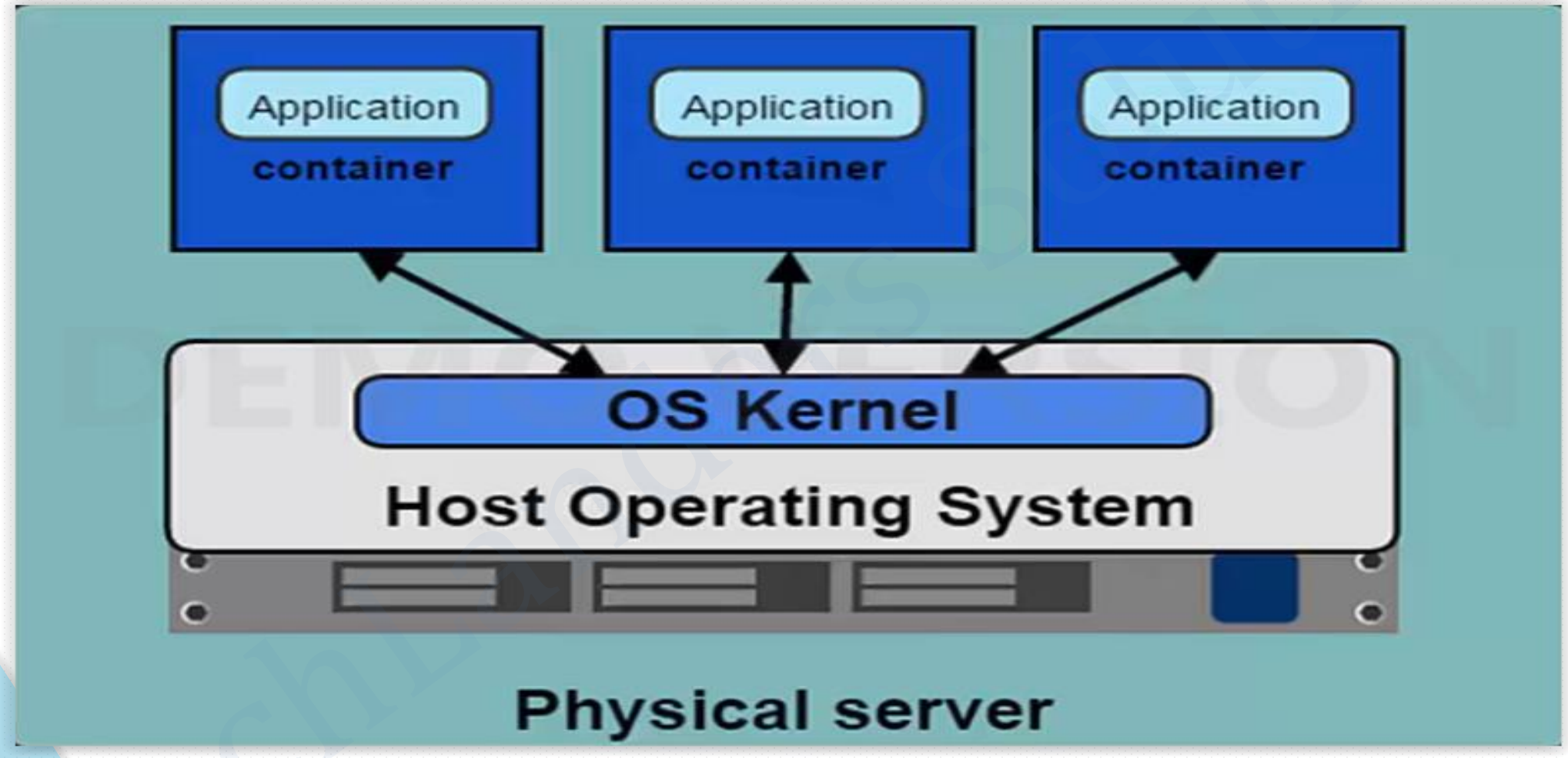
- Each guest instance is called a "Container"

- Each container has its own

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

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

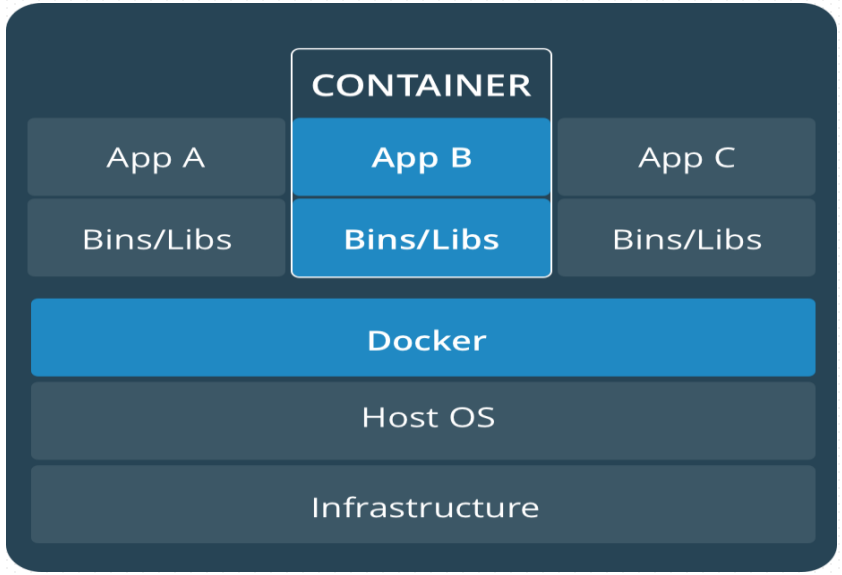
Overview of Containers

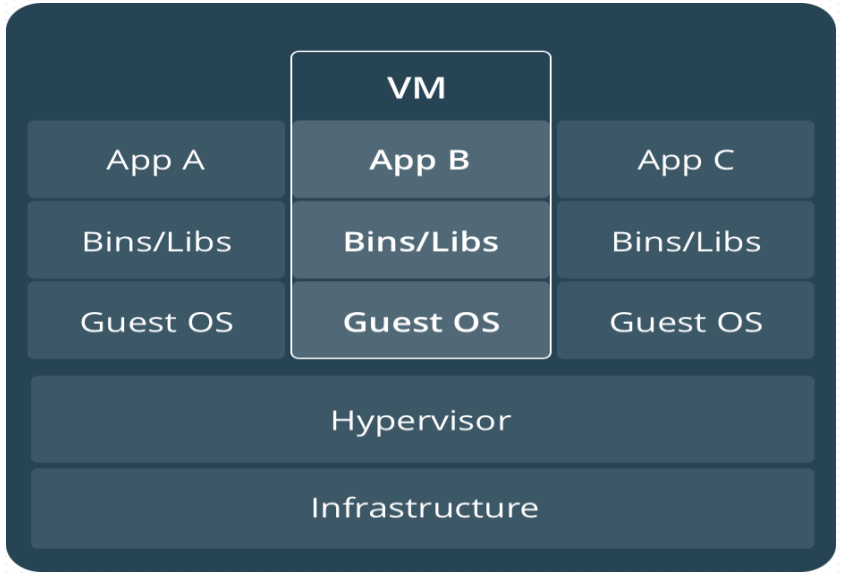


Containers VS VM's

- 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

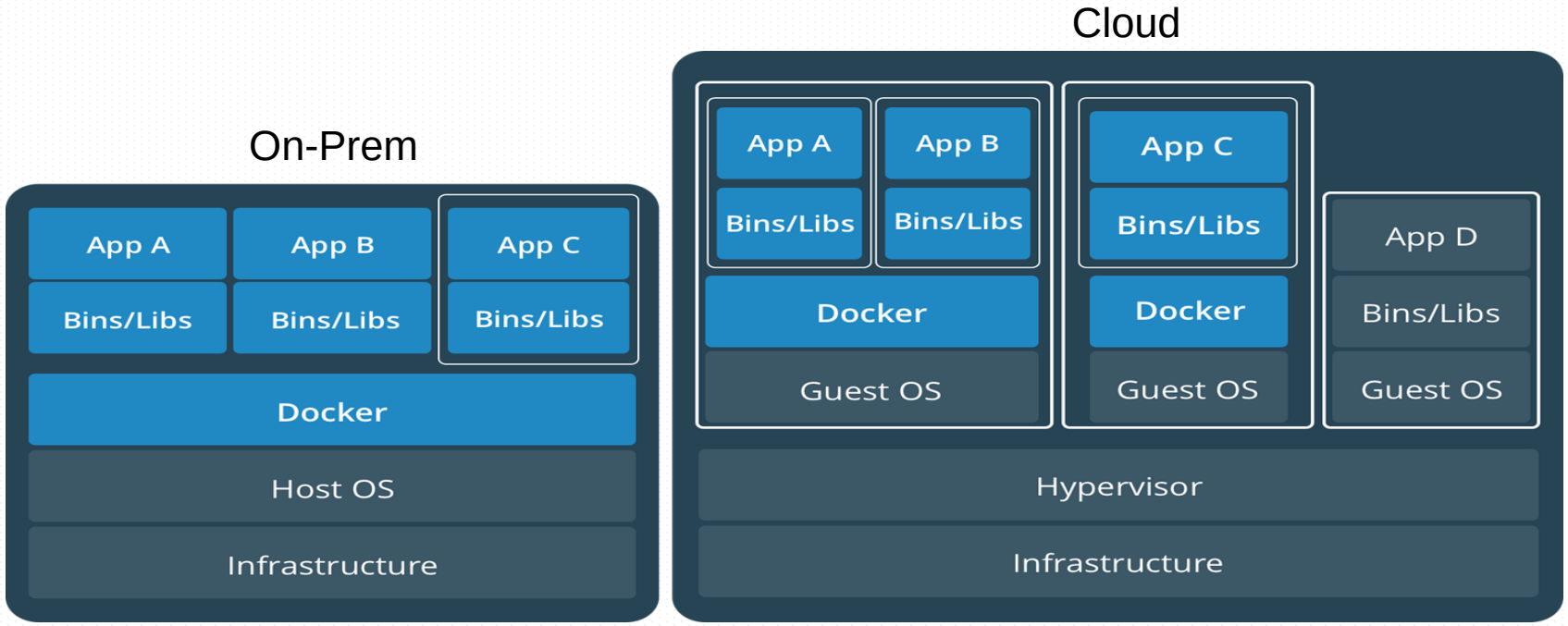




Containers are an app level construct

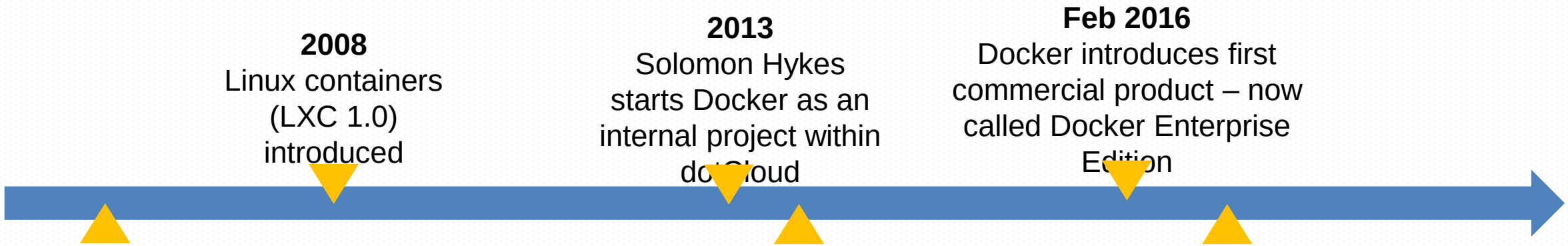
VMs are an infrastructure level construct to turn one machine into many servers

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



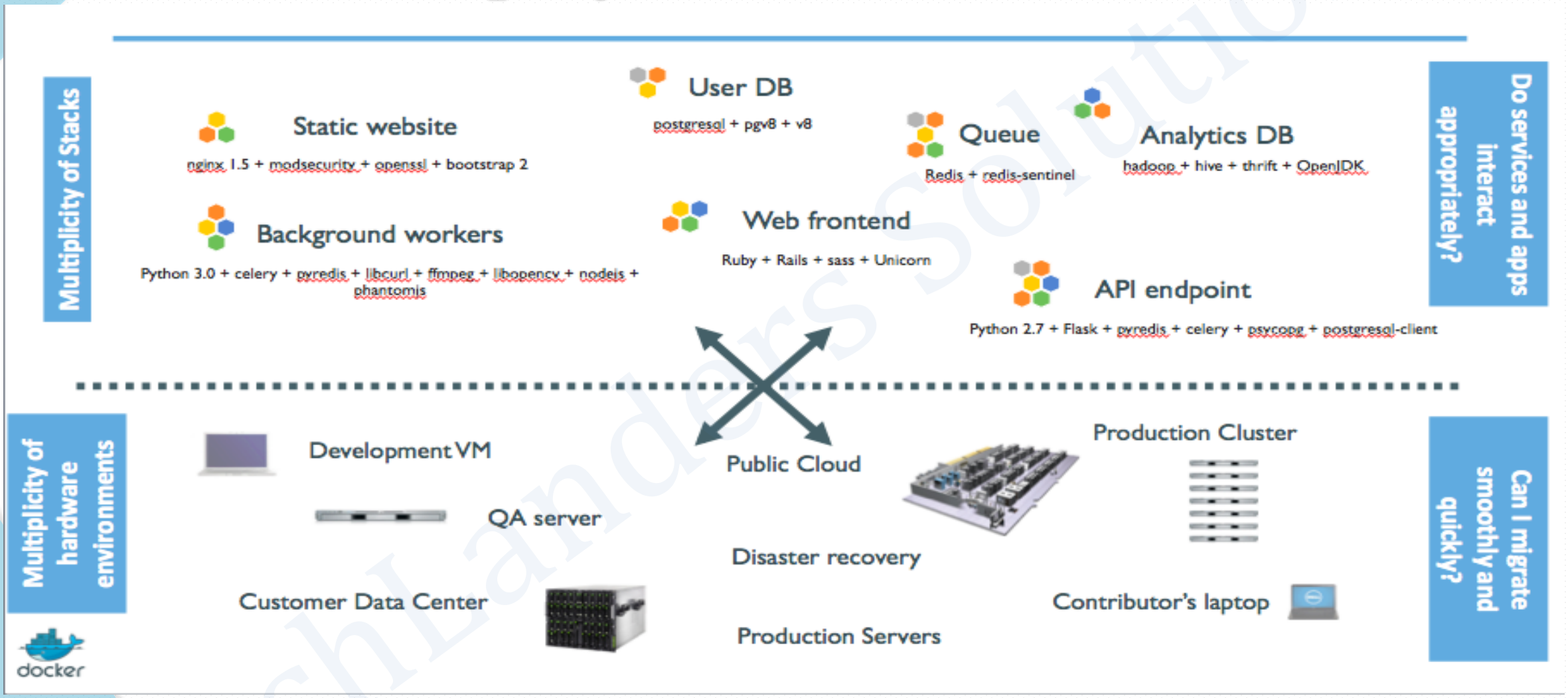
Origins of Docker Project

- ‘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

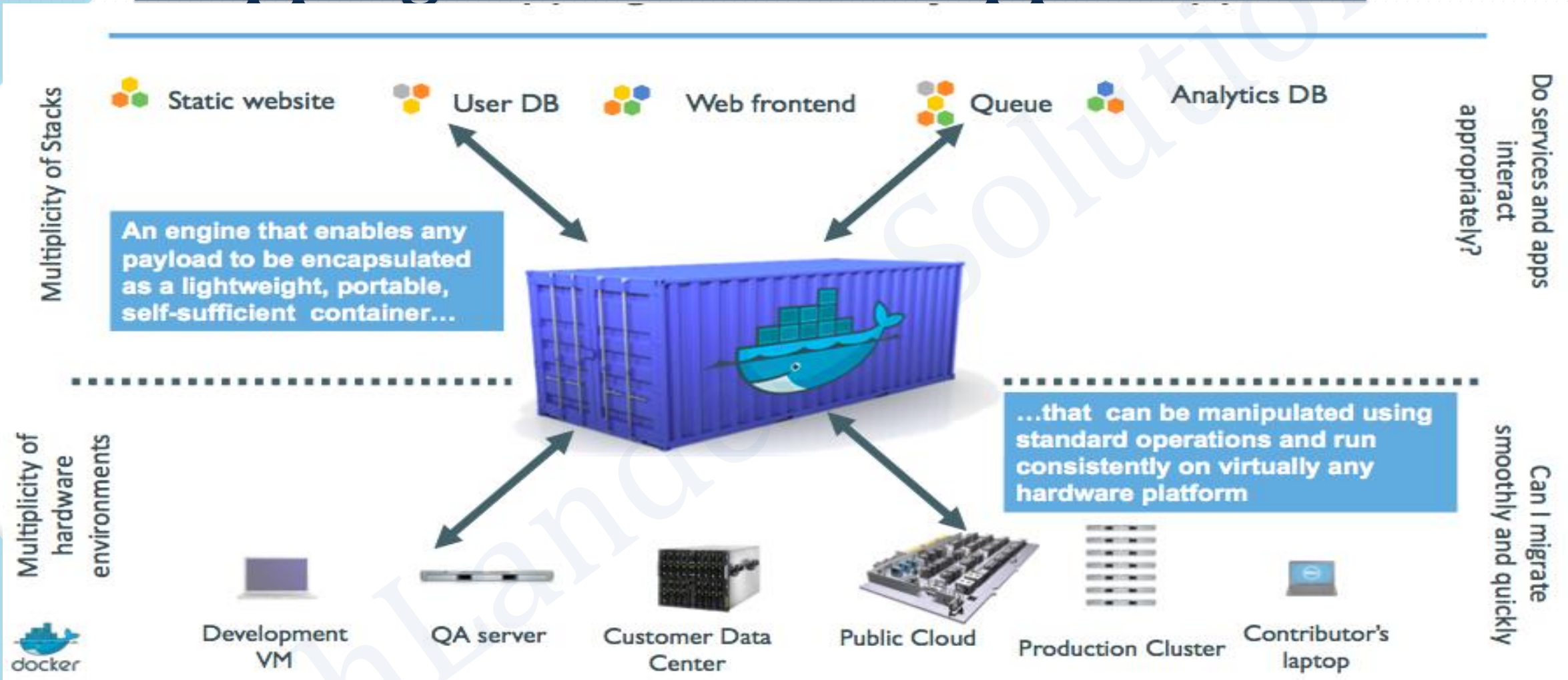


Matrix Checks

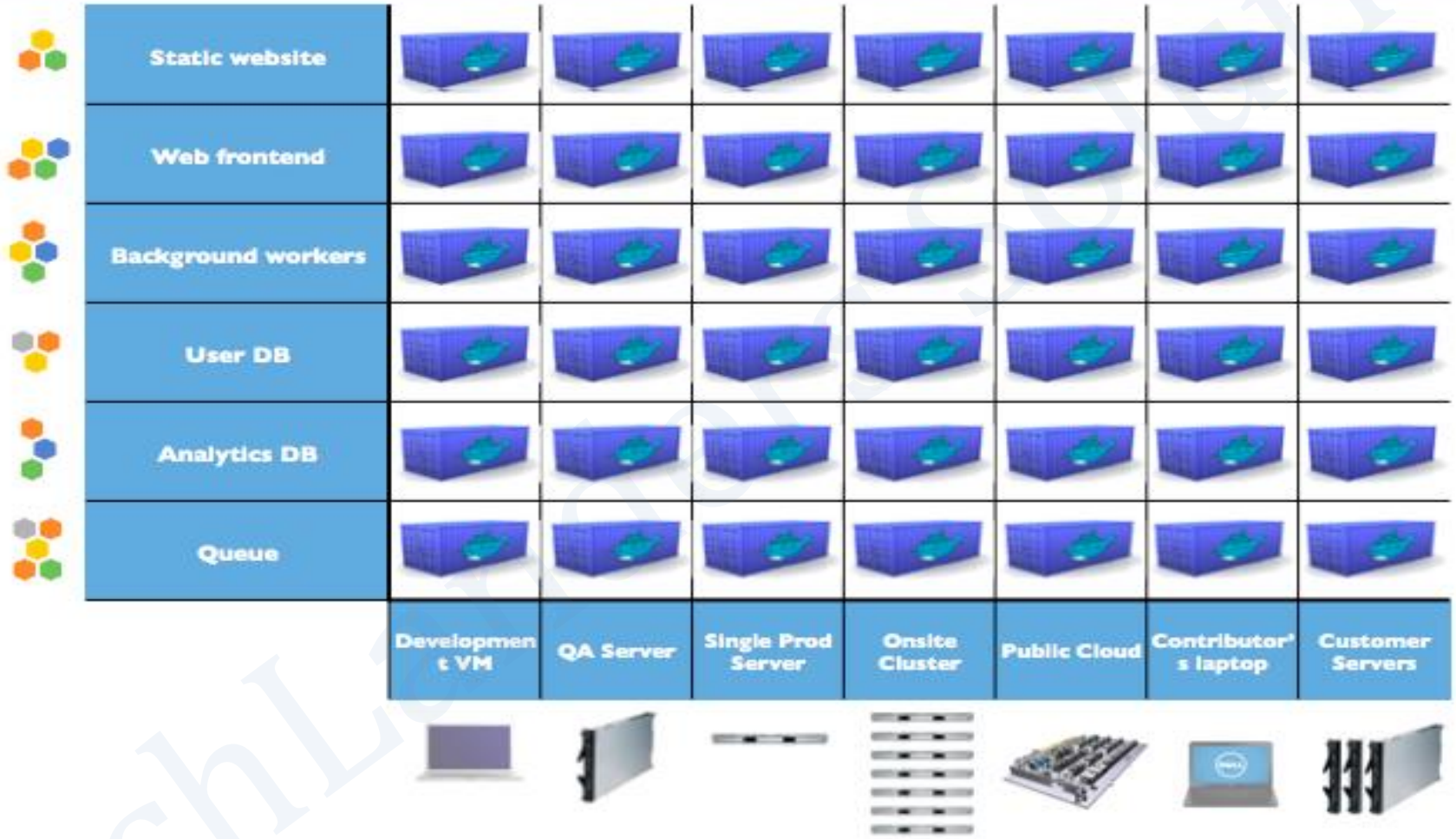
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Intermodal Shipping Containers

Shipping Container for Applications



Eliminate the Matrix



Results

Speed

- No OS to boot = applications online in seconds

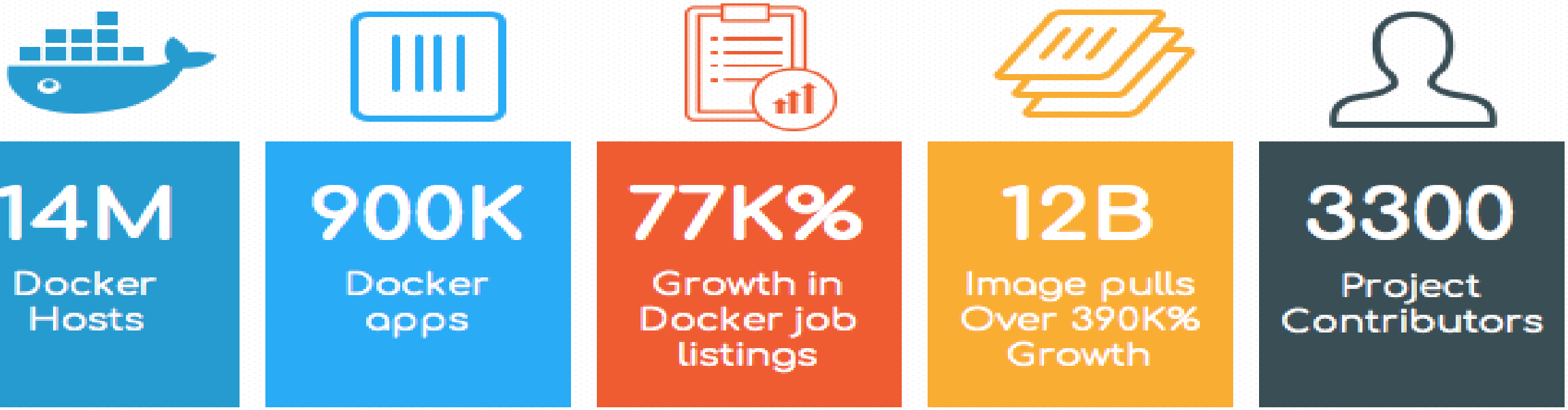
Portability

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

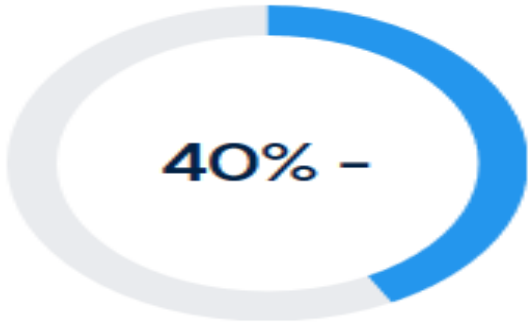
Efficiency

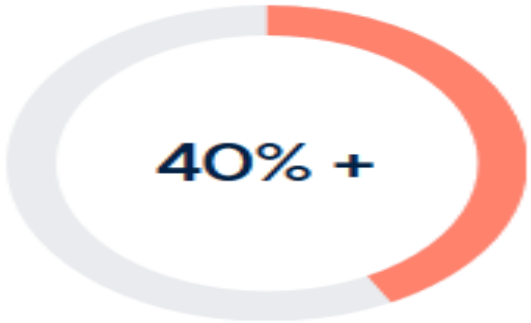
- Less OS overhead
- Improved VM density

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.

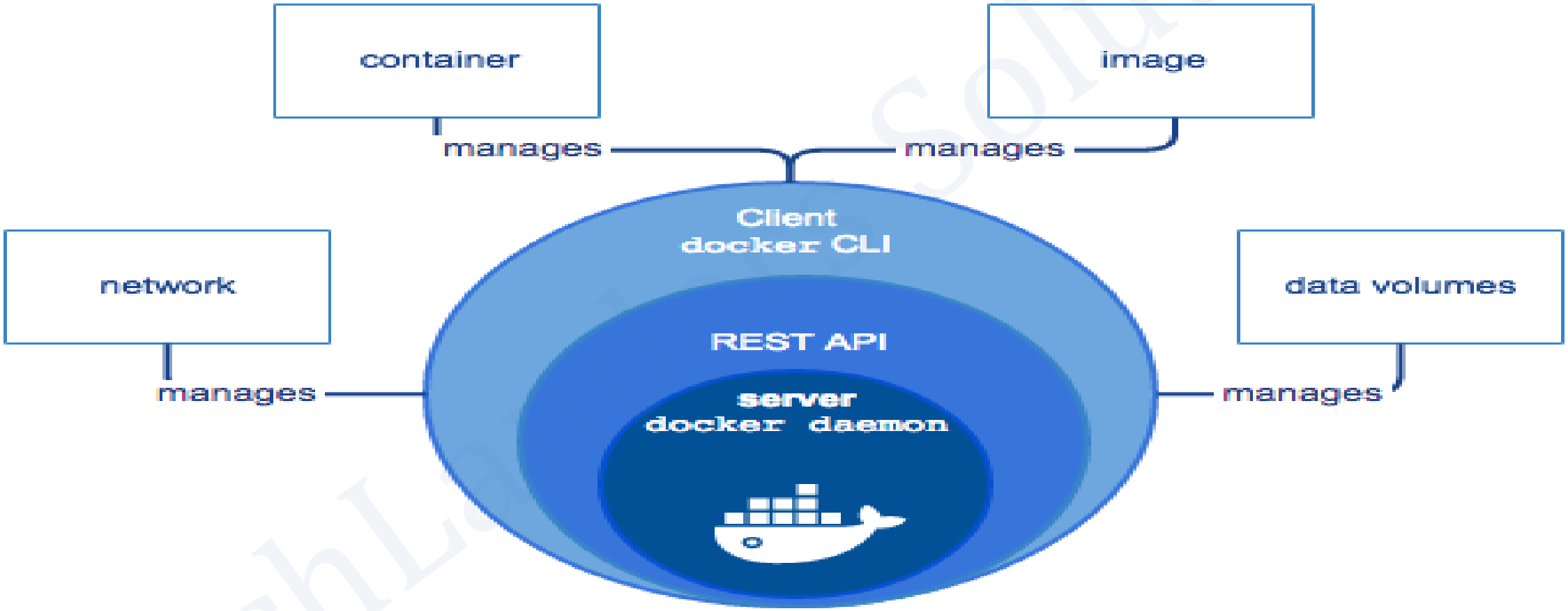
Docker Test

- 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



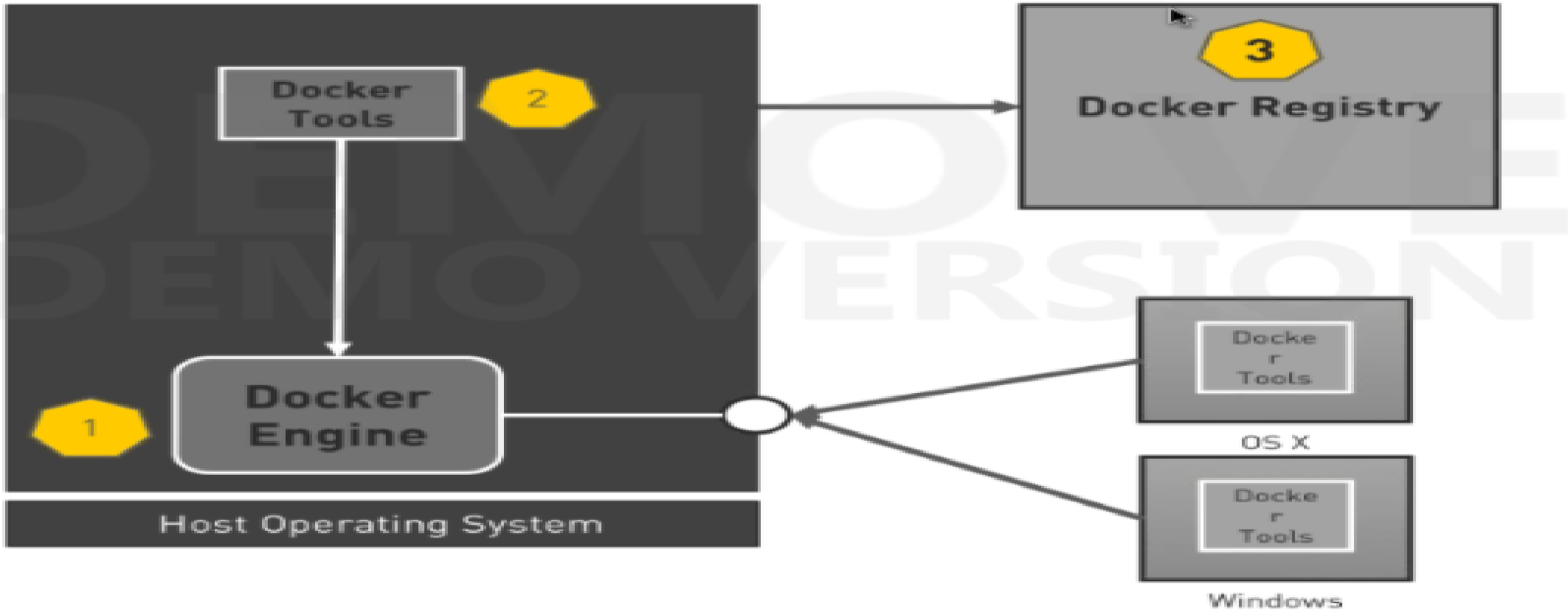
Docker Engine

- 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.

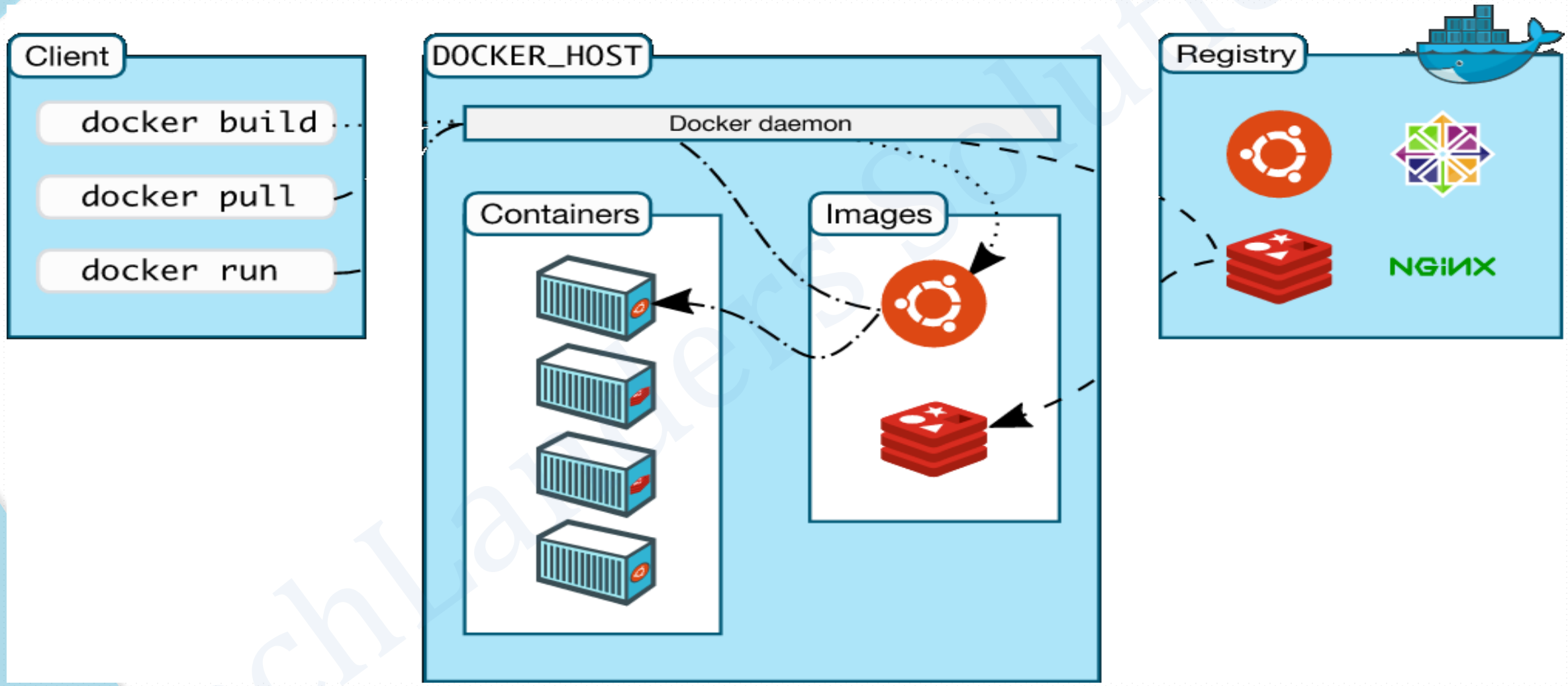
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands

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

Docker Architecture



Docker Architecture



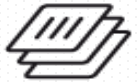
Docker Architecture

- 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

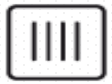
- The Docker Daemon
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- The Docker Client
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker.
  - It accepts commands and configuration flags from the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

Docker Architecture



Image

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



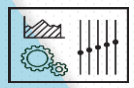
Container

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.

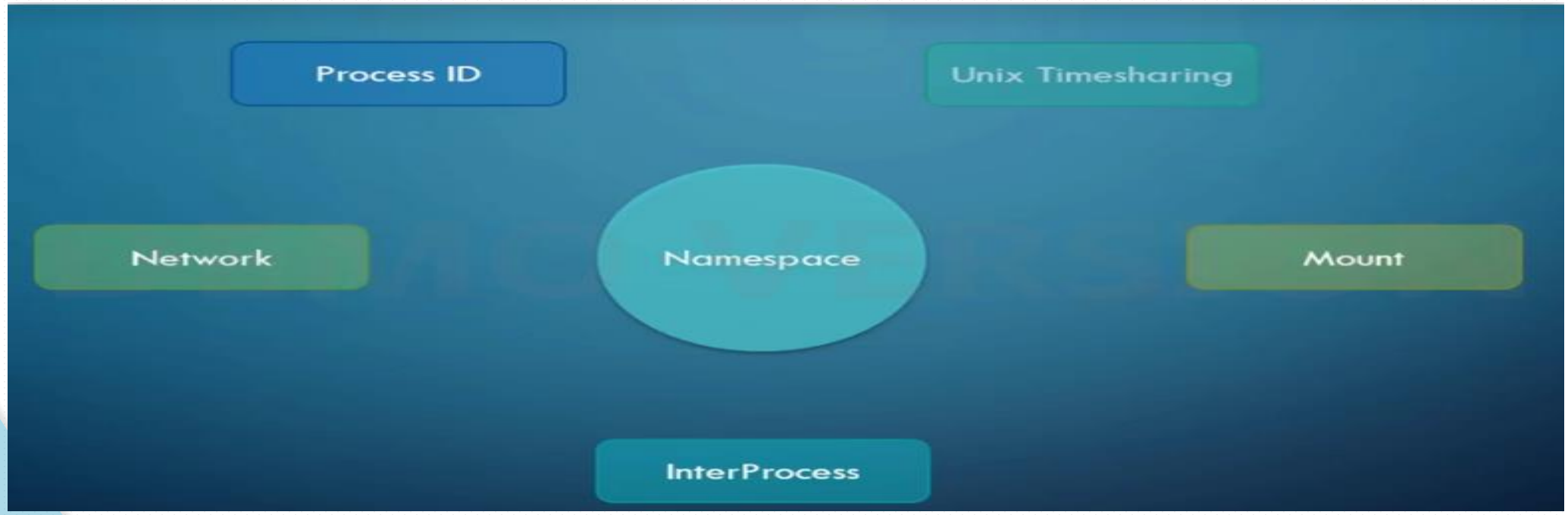
Registry



Stores, distributes and manages Docker images

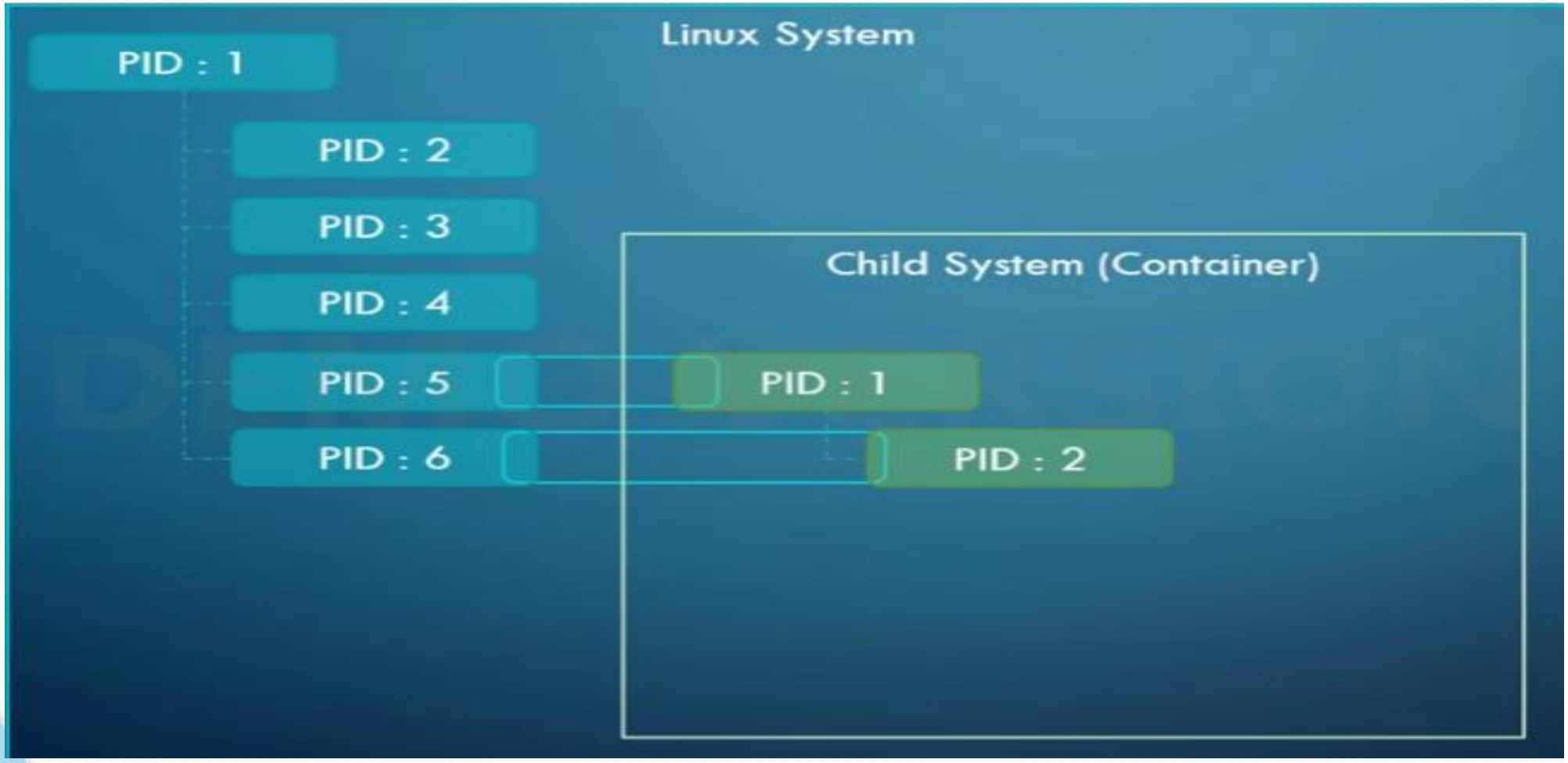
Docker Architecture

- 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.

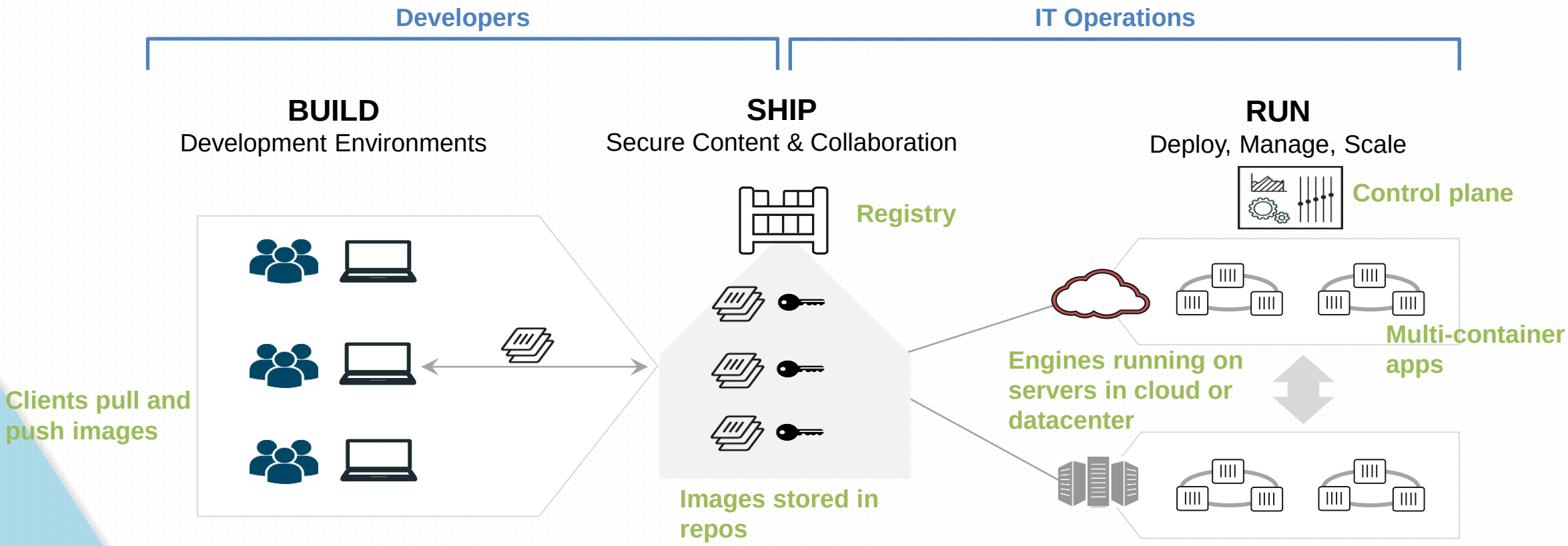
Docker Features

- 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.