Welcome!

DevOps

by

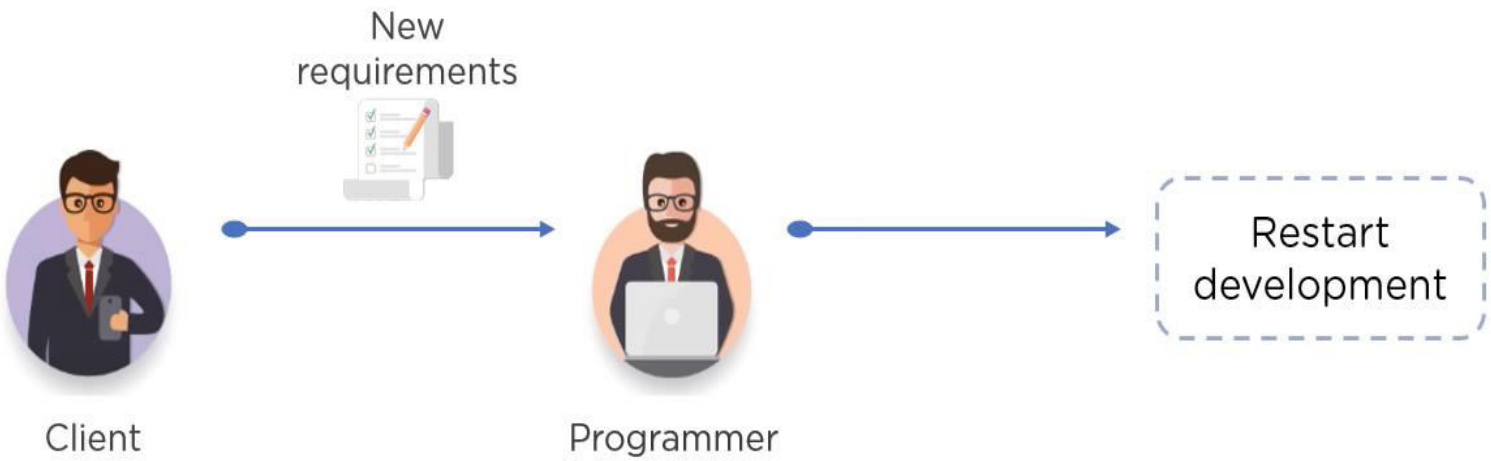
Girish Godbole

DevOps

Waterfall Model

Disadvantage of waterfall model

Any new requirements from the client will restart the development cycle

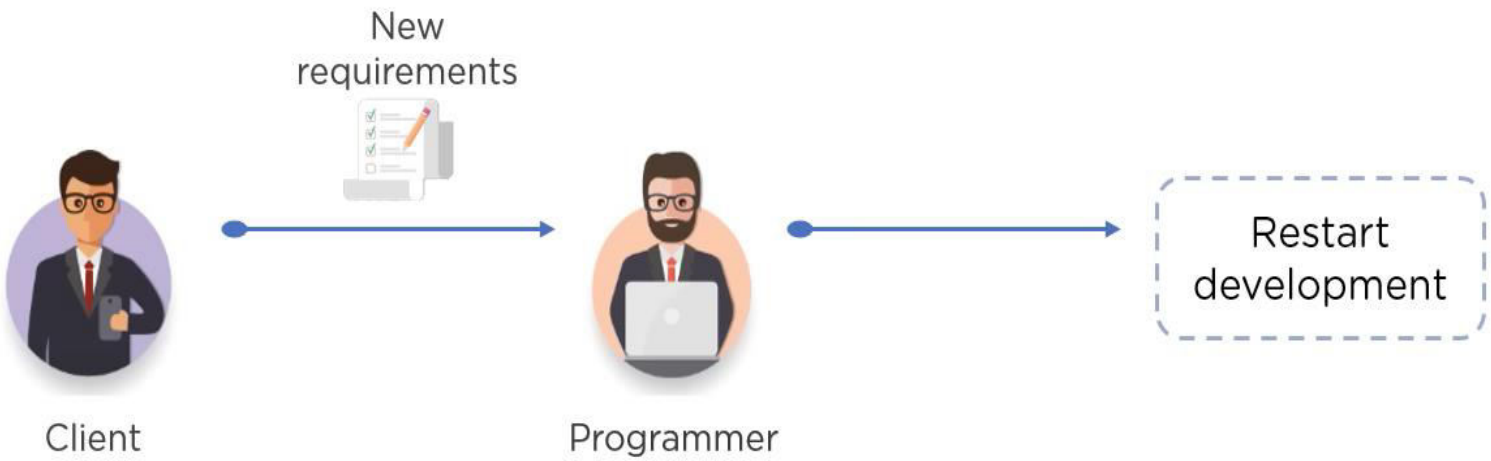


DevOps

Waterfall Model

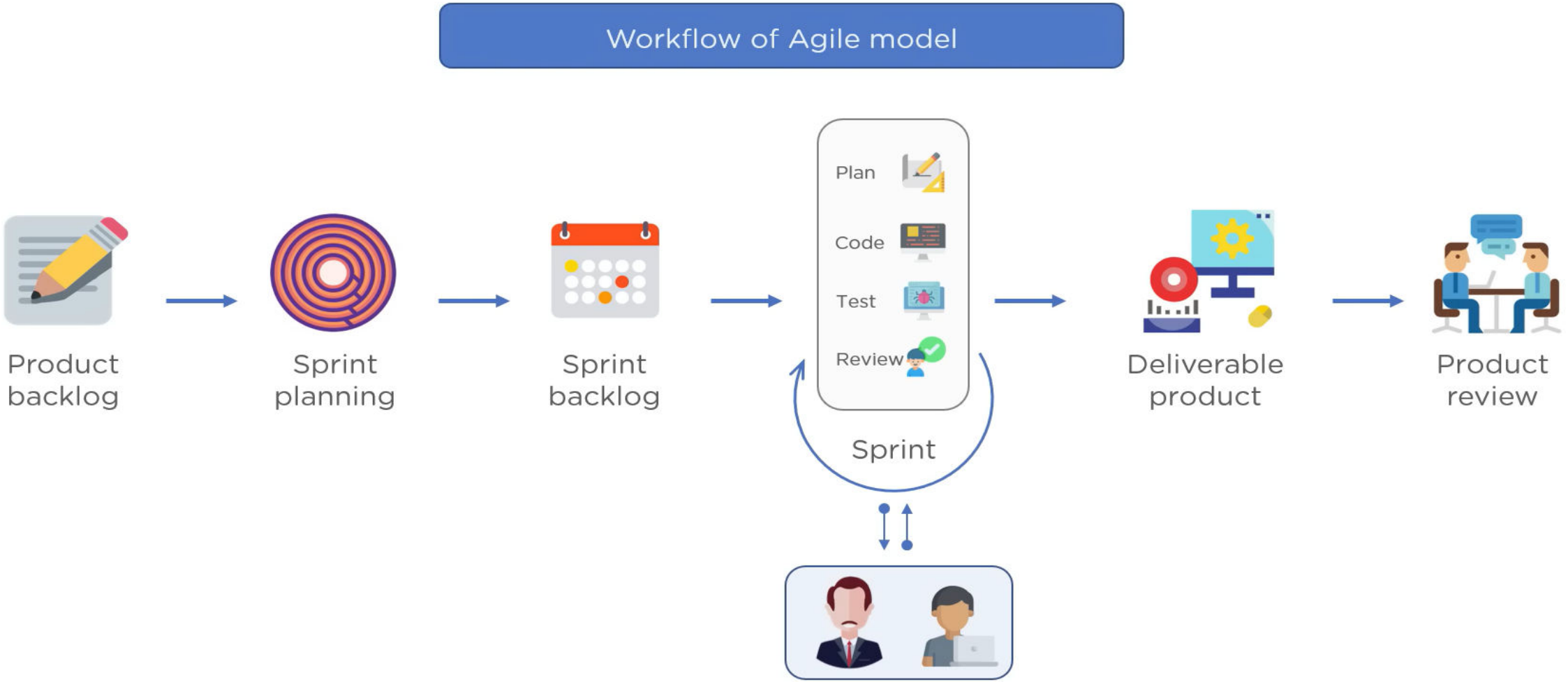
Disadvantage of waterfall model

Any new requirements from the client will restart the development cycle



DevOps

Agile Model



DevOps

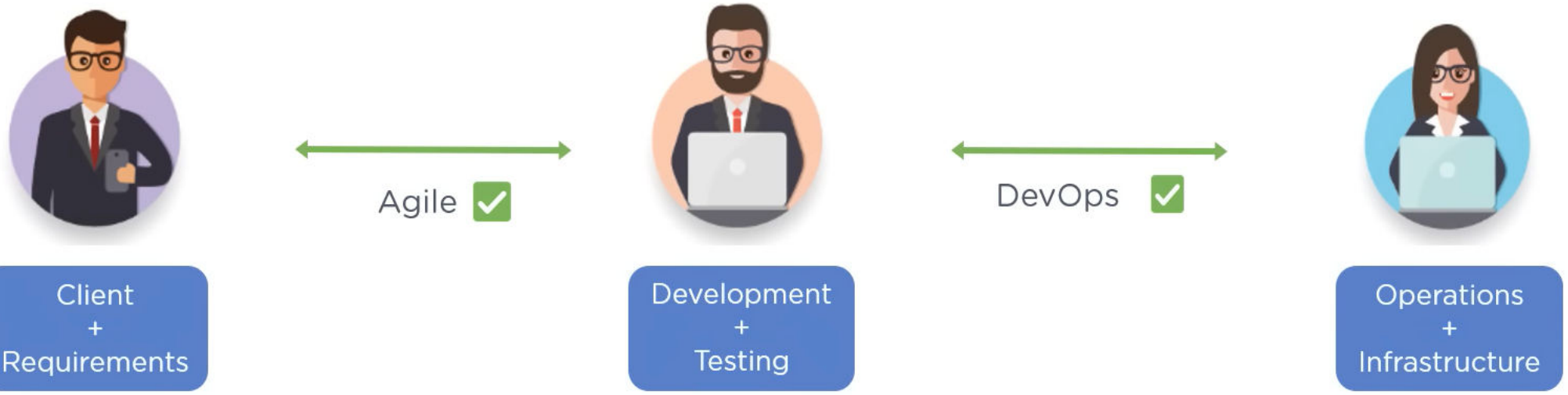
Agile Model

When the product fails in production servers, the operations team are clueless and send product back to the development team

DevOps

What is DevOps?

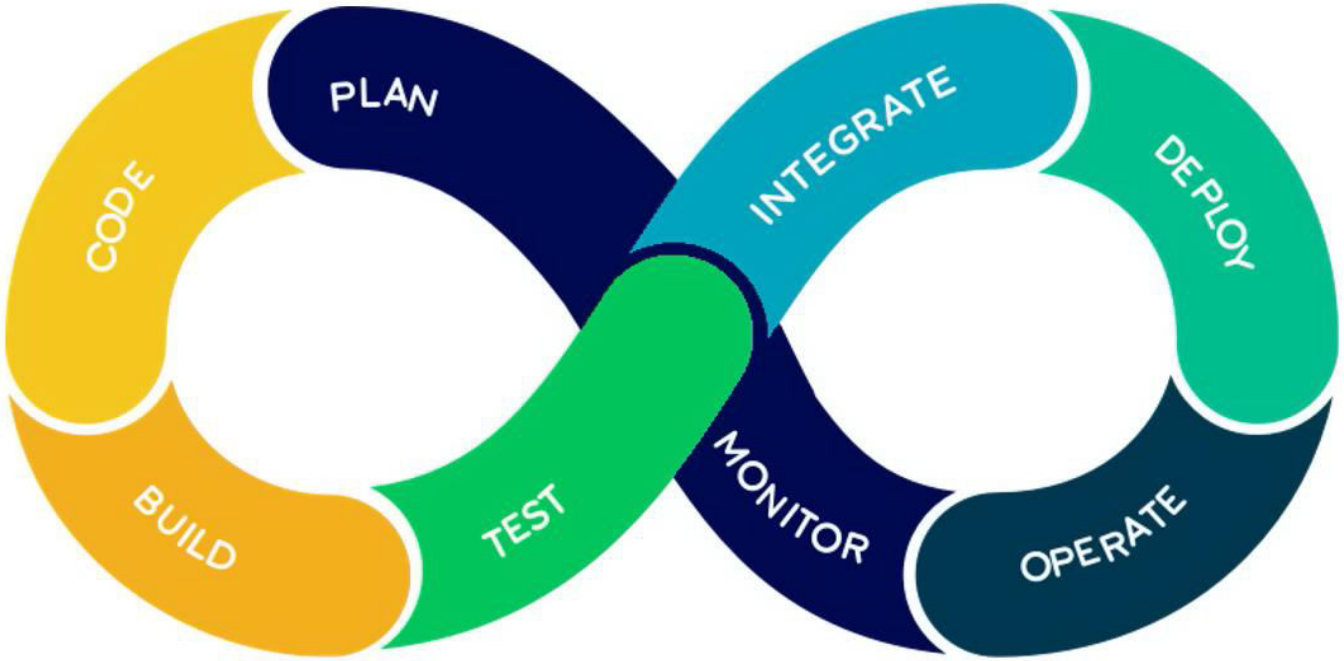
DevOps addressed the gap between Developers and Operations



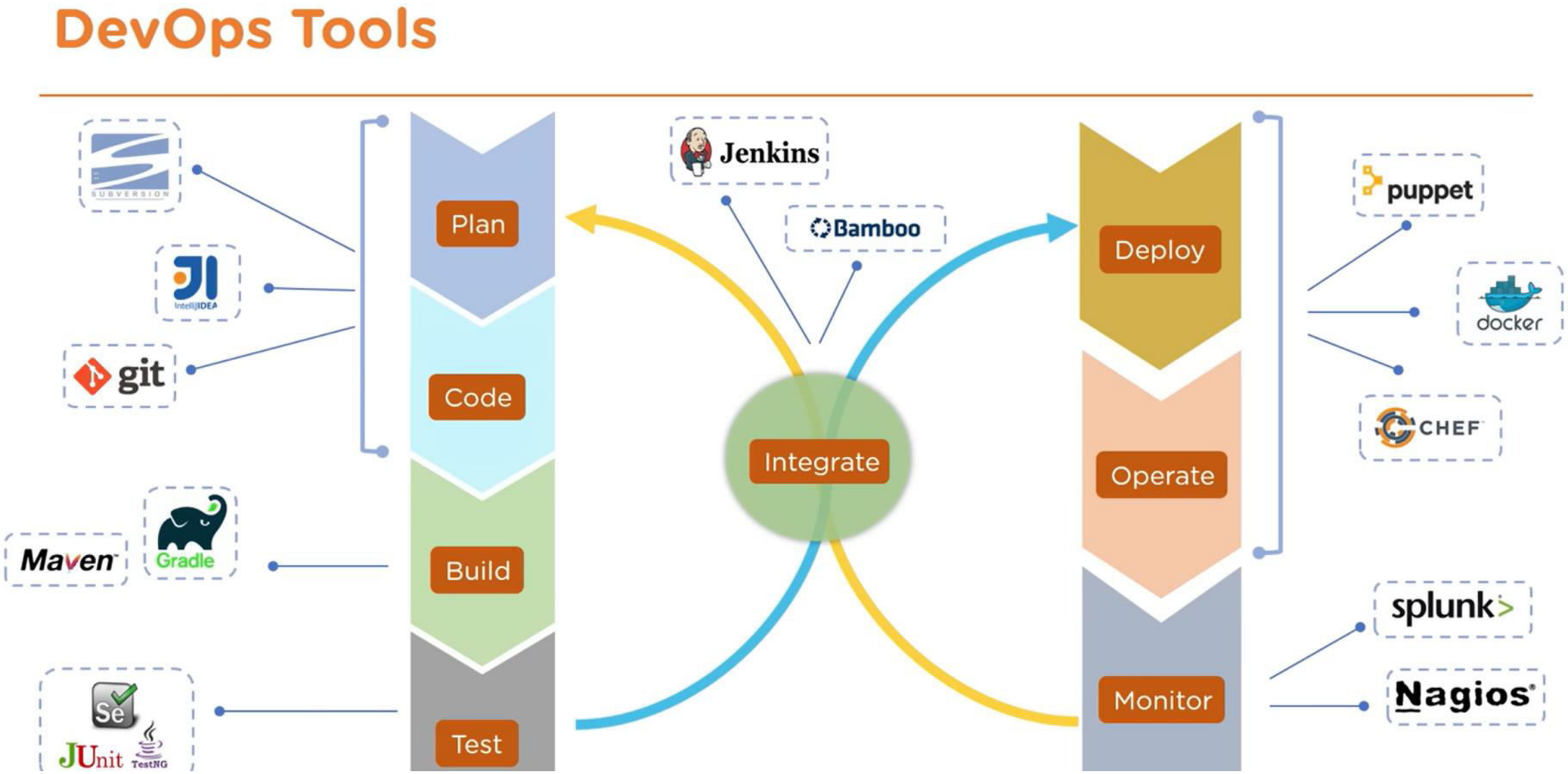
DevOps

DevOps Phases

According to DevOps practices, the workflow in software development and delivery is divided into 8 phases

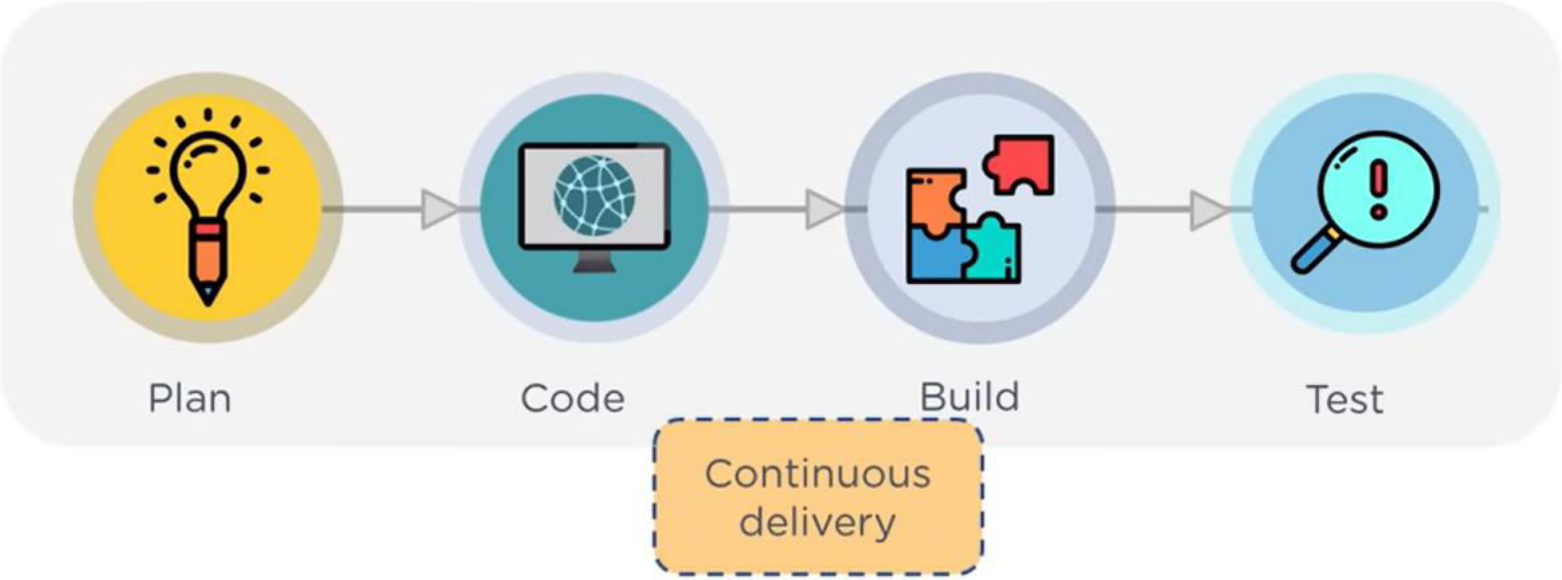


DevOps


DevOps Tools
Plan
Code
Build
Test
Integrate
Deploy
Operate
Monitor
Jenkins
Bamboo
puppet
docker
CHEF
splunk>
Nagios
git
Maven
Gradle
JUnit
TestNG

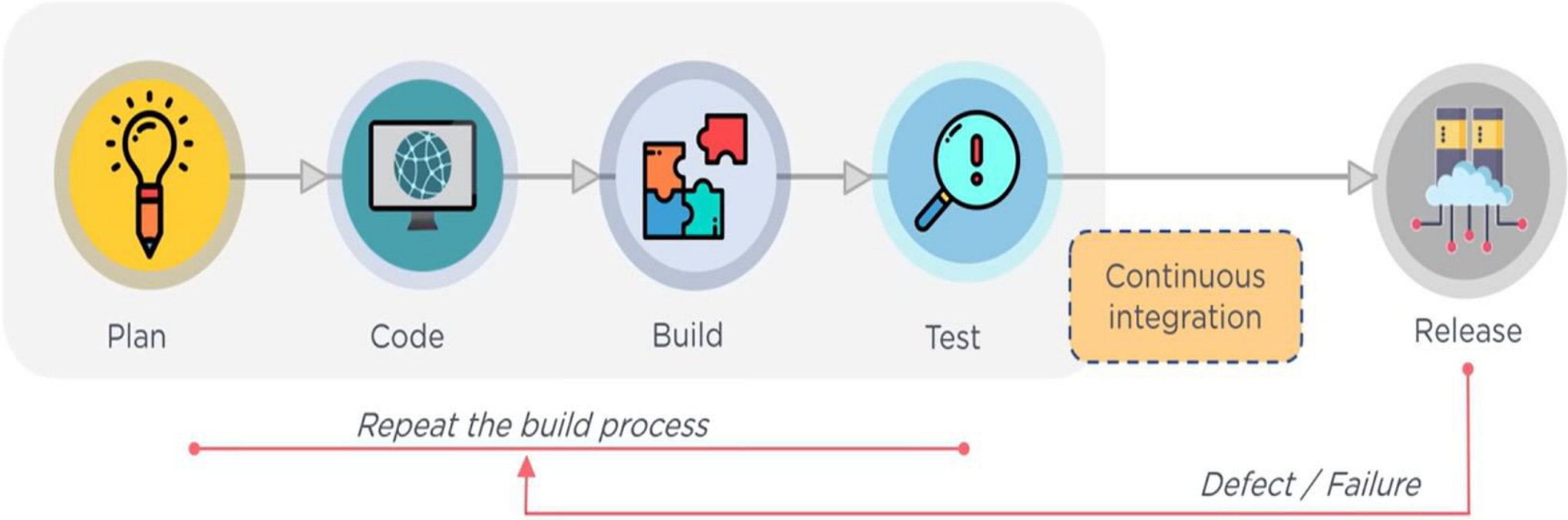
DevOps

Continuous Delivery

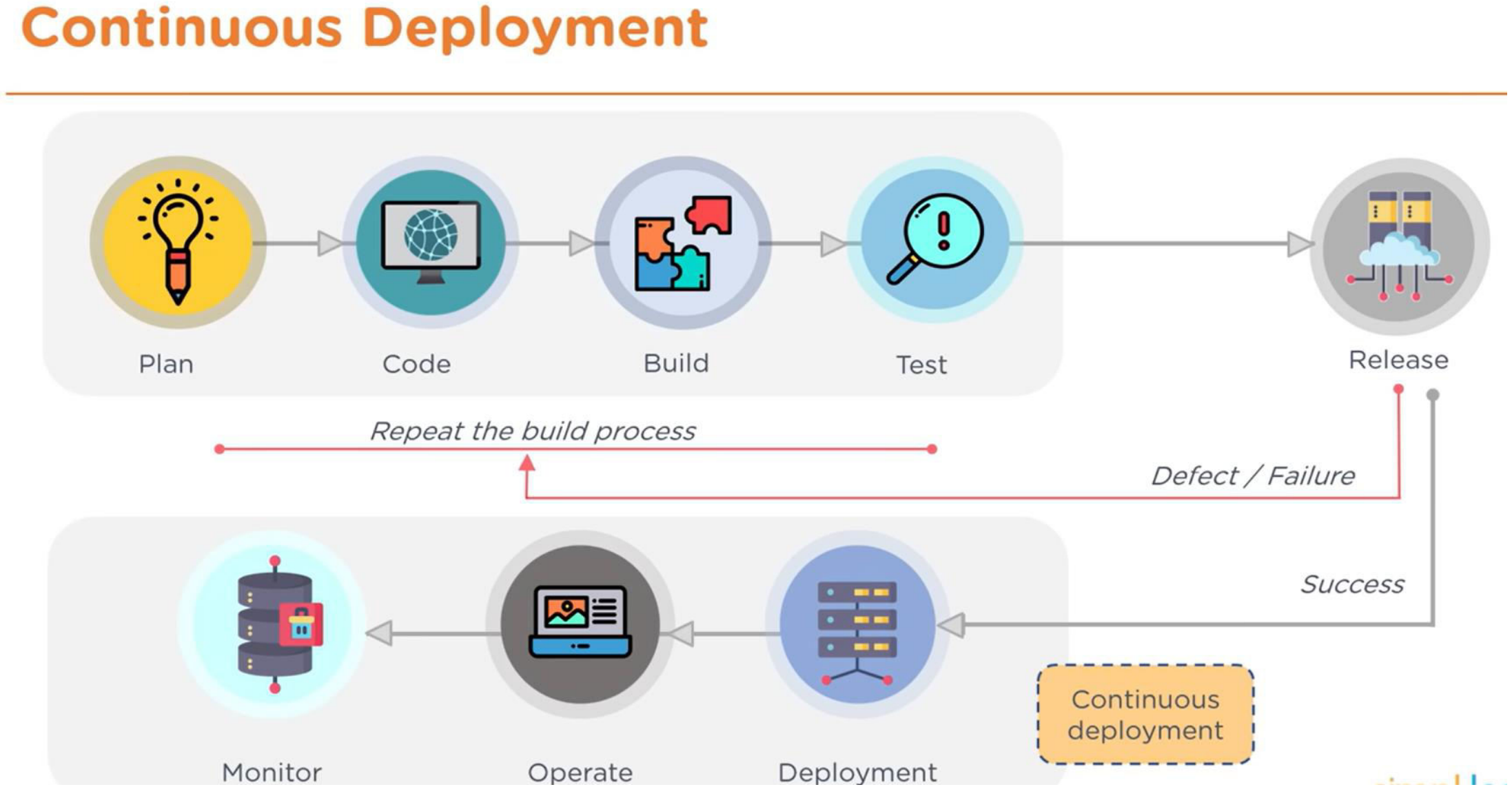


DevOps

Continuous Integration

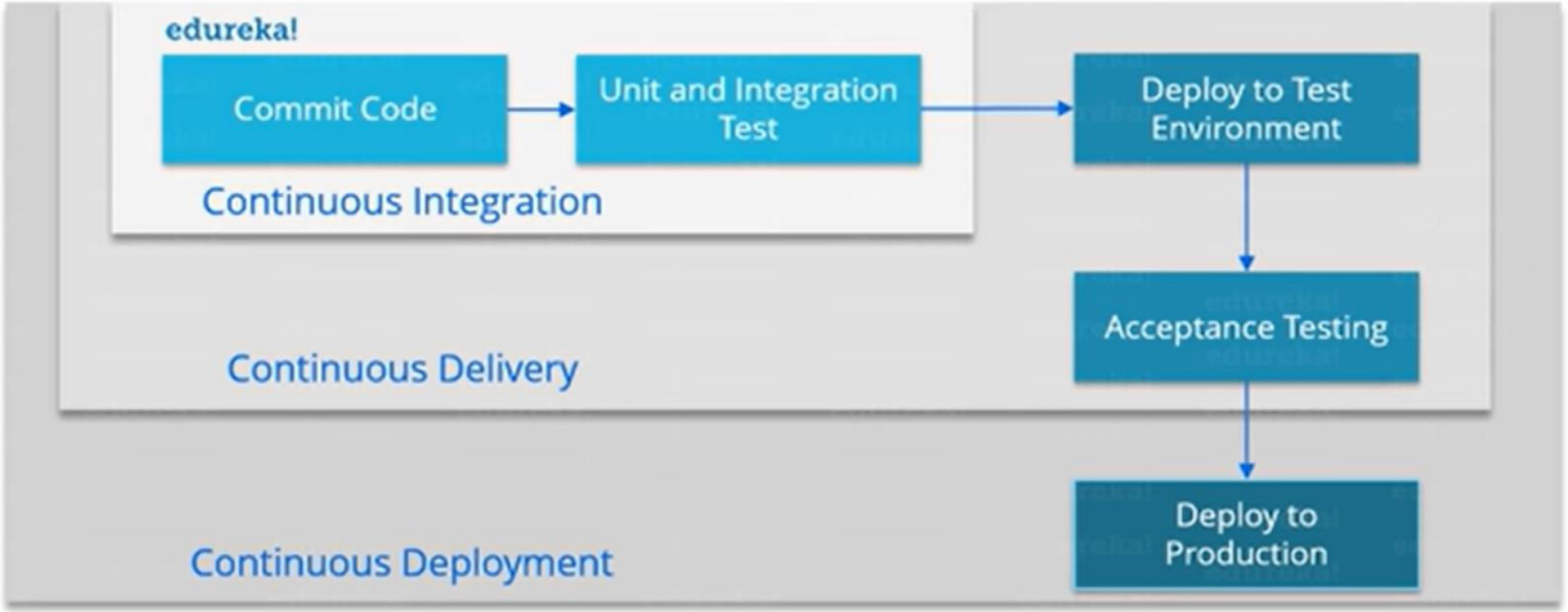


DevOps

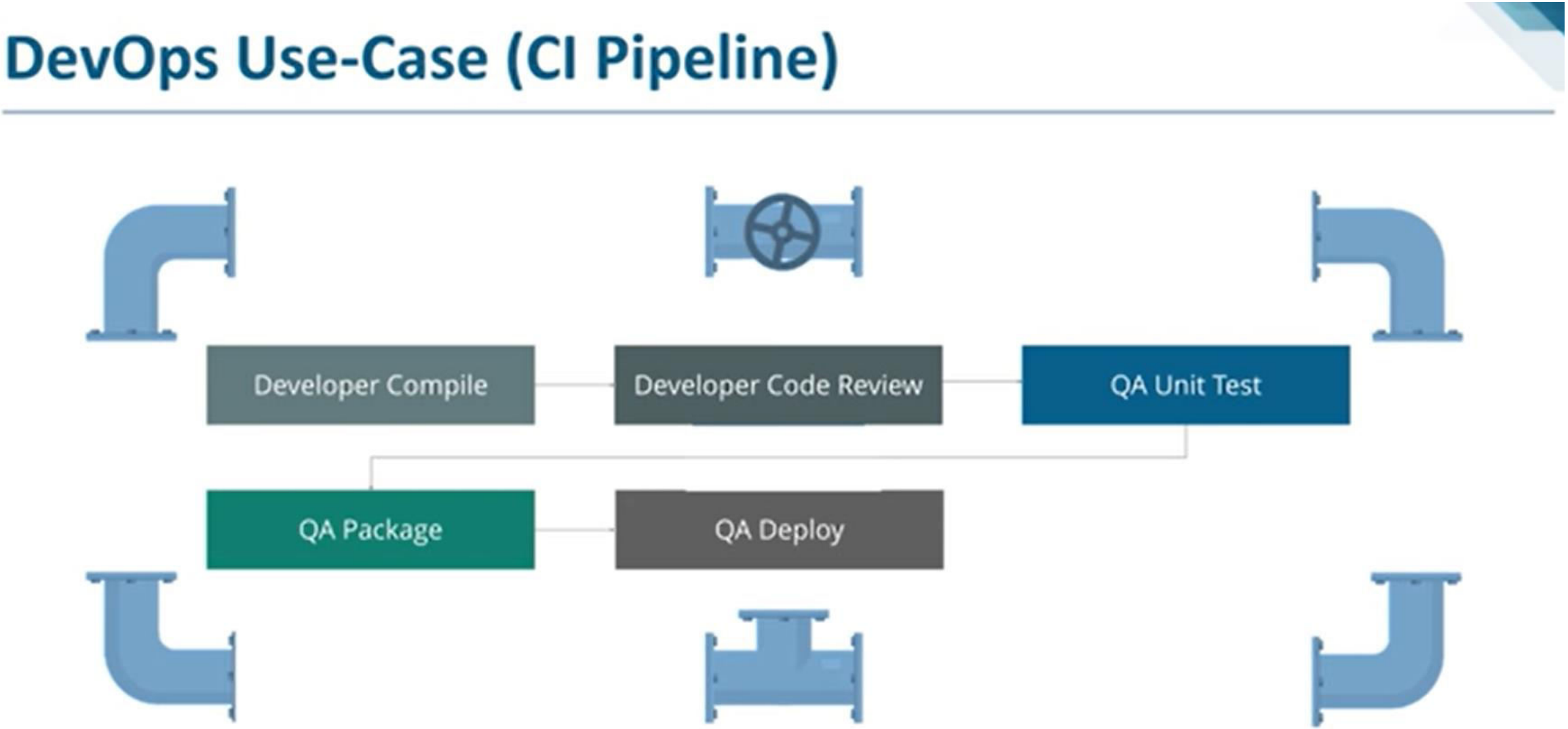

Continuous Deployment
Plan
Code
Build
Test
Release
Repeat the build process
Defect / Failure
Success
Monitor
Operate
Deployment
Continuous deployment

DevOps

CI/ CD


edureka!
Commit Code
Unit and Integration Test
Deploy to Test Environment
Continuous Integration
Acceptance Testing
Continuous Delivery
Deploy to Production
Continuous Deployment

DevOps


DevOps Use-Case (CI Pipeline)
Developer Compile
Developer Code Review
QA Unit Test
QA Package
QA Deploy

DevOps

DevOps Advantages

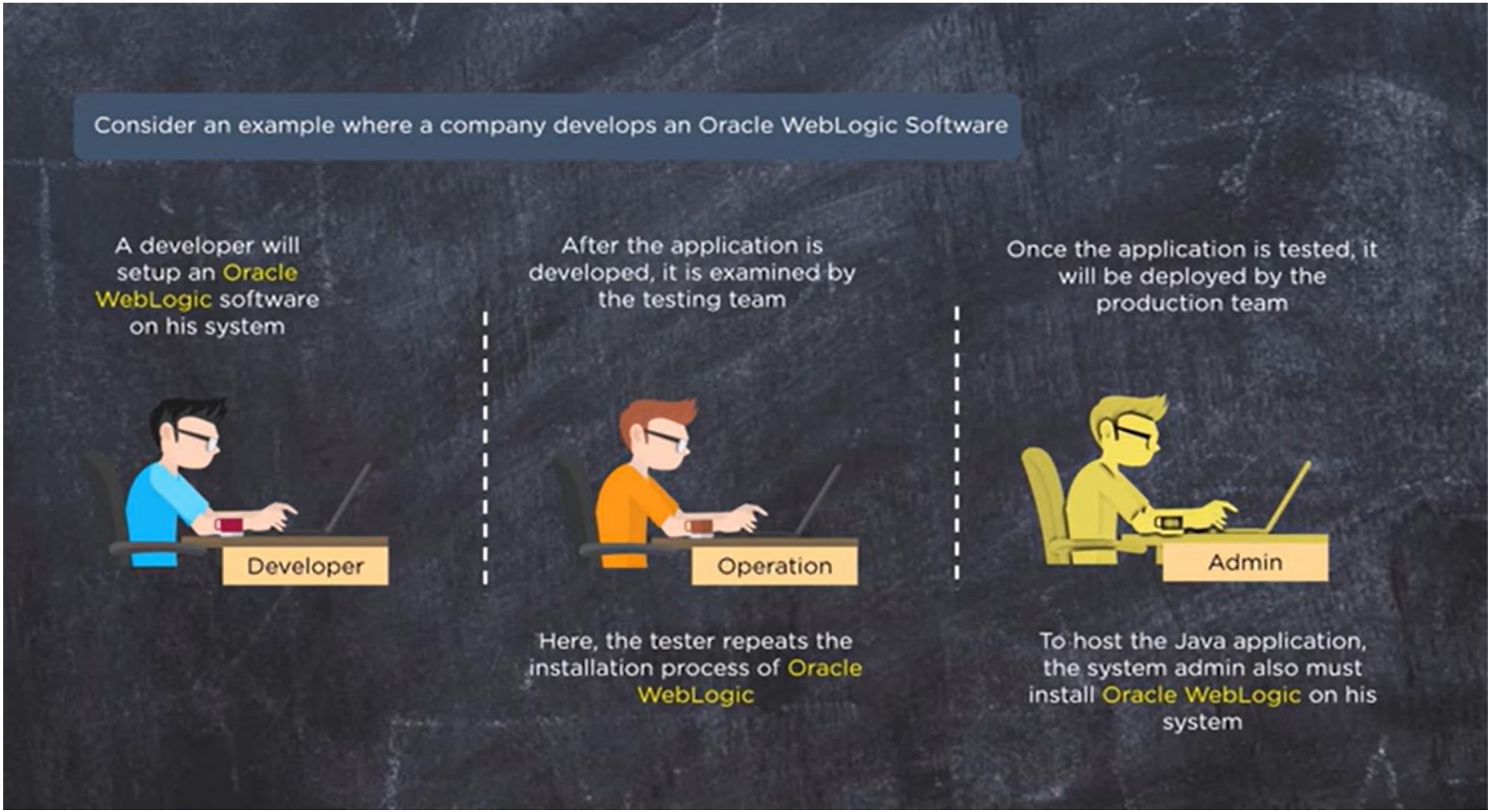
Time taken to create and deliver software is reduced

Complexity of maintaining an application is reduced

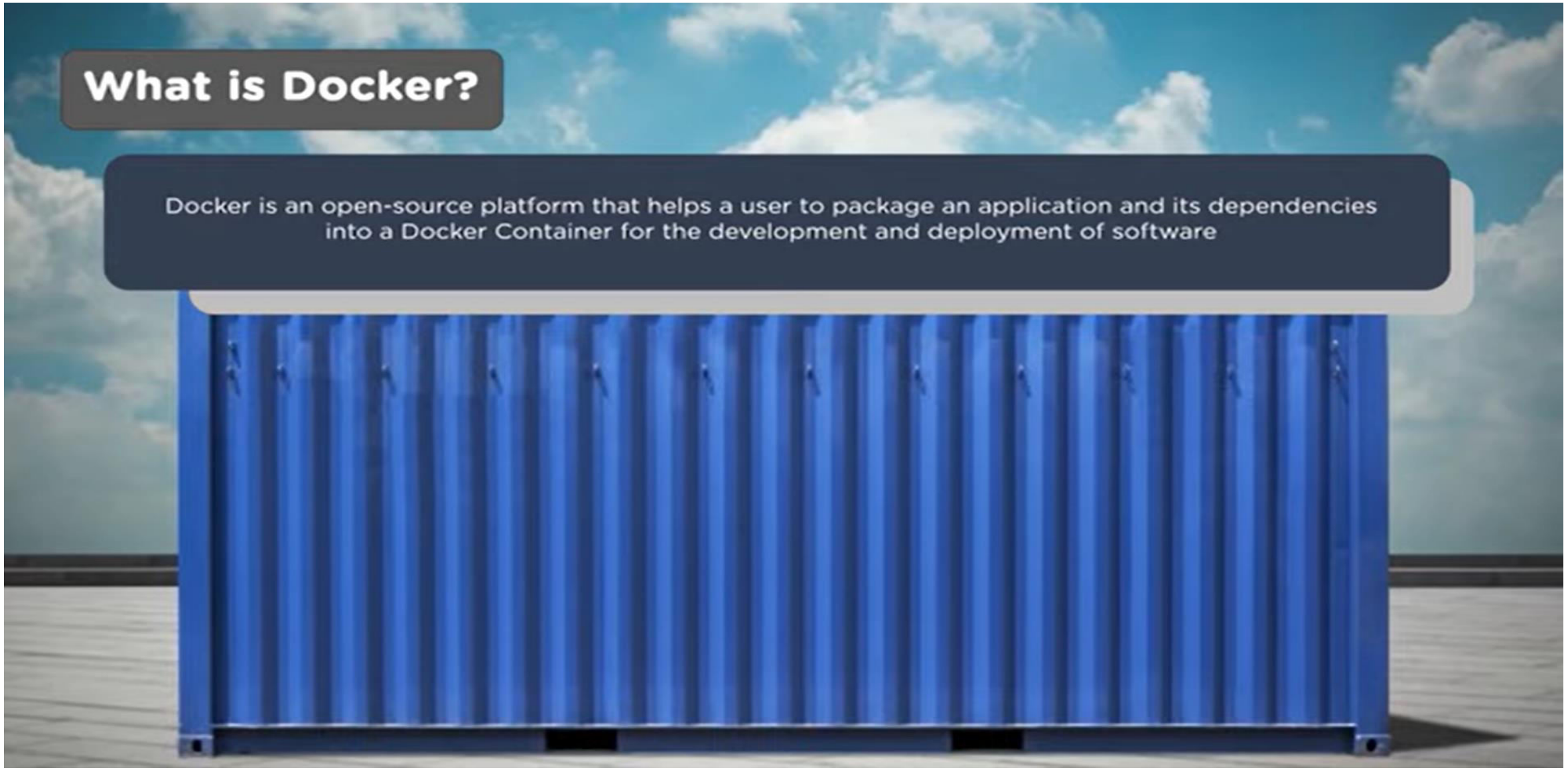
Improved collaboration between developers and operations team

Continuous integration and delivery ensure faster time to market

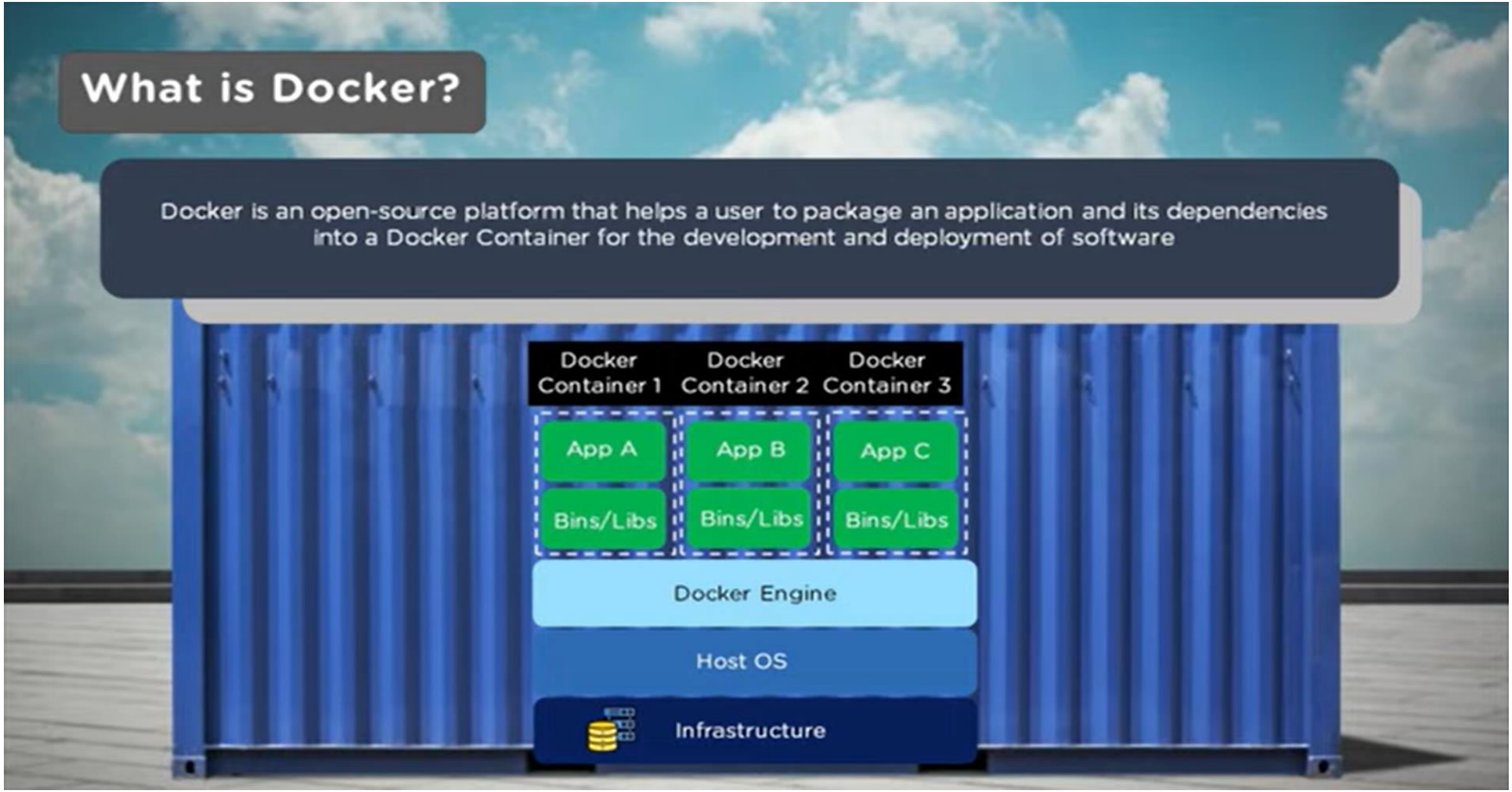
Docker


Consider an example where a company develops an Oracle WebLogic Software
A developer will setup an Oracle WebLogic software on his system
Developer
After the application is developed, it is examined by the testing team
Operation
Here, the tester repeats the installation process of Oracle WebLogic
Once the application is tested, it will be deployed by the production team
Admin
To host the Java application, the system admin also must install Oracle WebLogic on his system

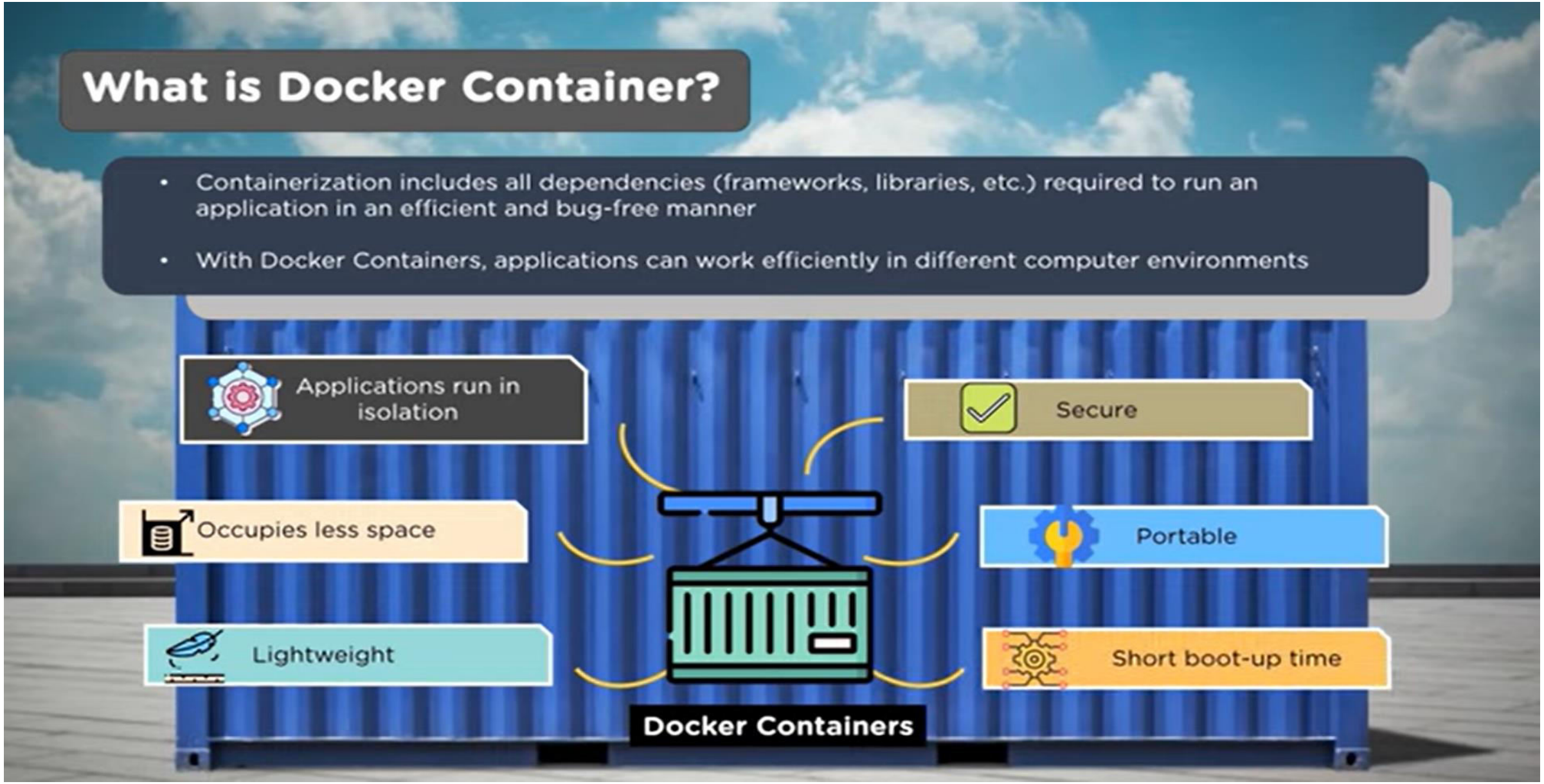
Docker


What is Docker?
Docker is an open-source platform that helps a user to package an application and its dependencies into a Docker Container for the development and deployment of software

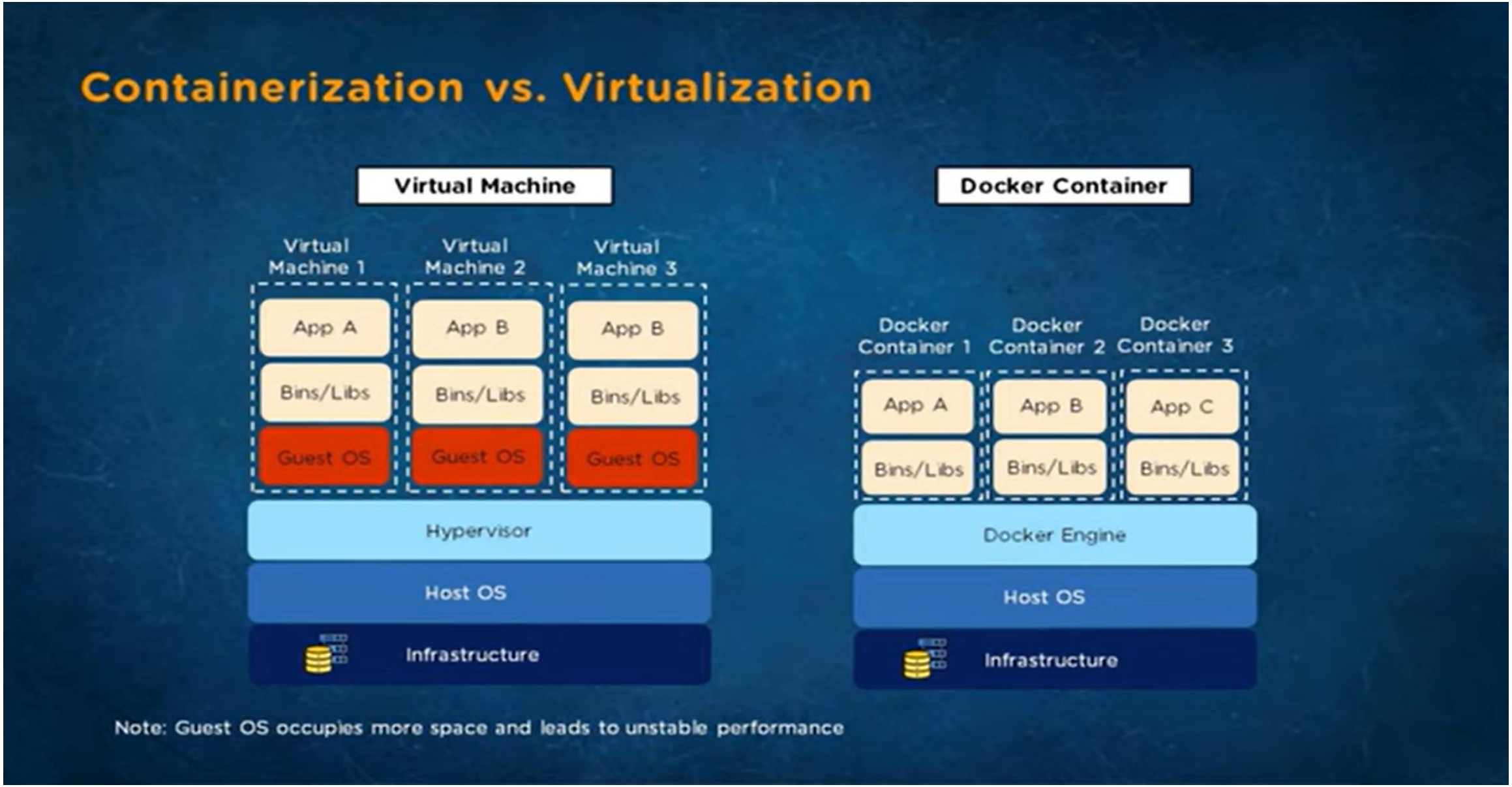
Docker


What is Docker?
Docker is an open-source platform that helps a user to package an application and its dependencies into a Docker Container for the development and deployment of software
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Infrastructure

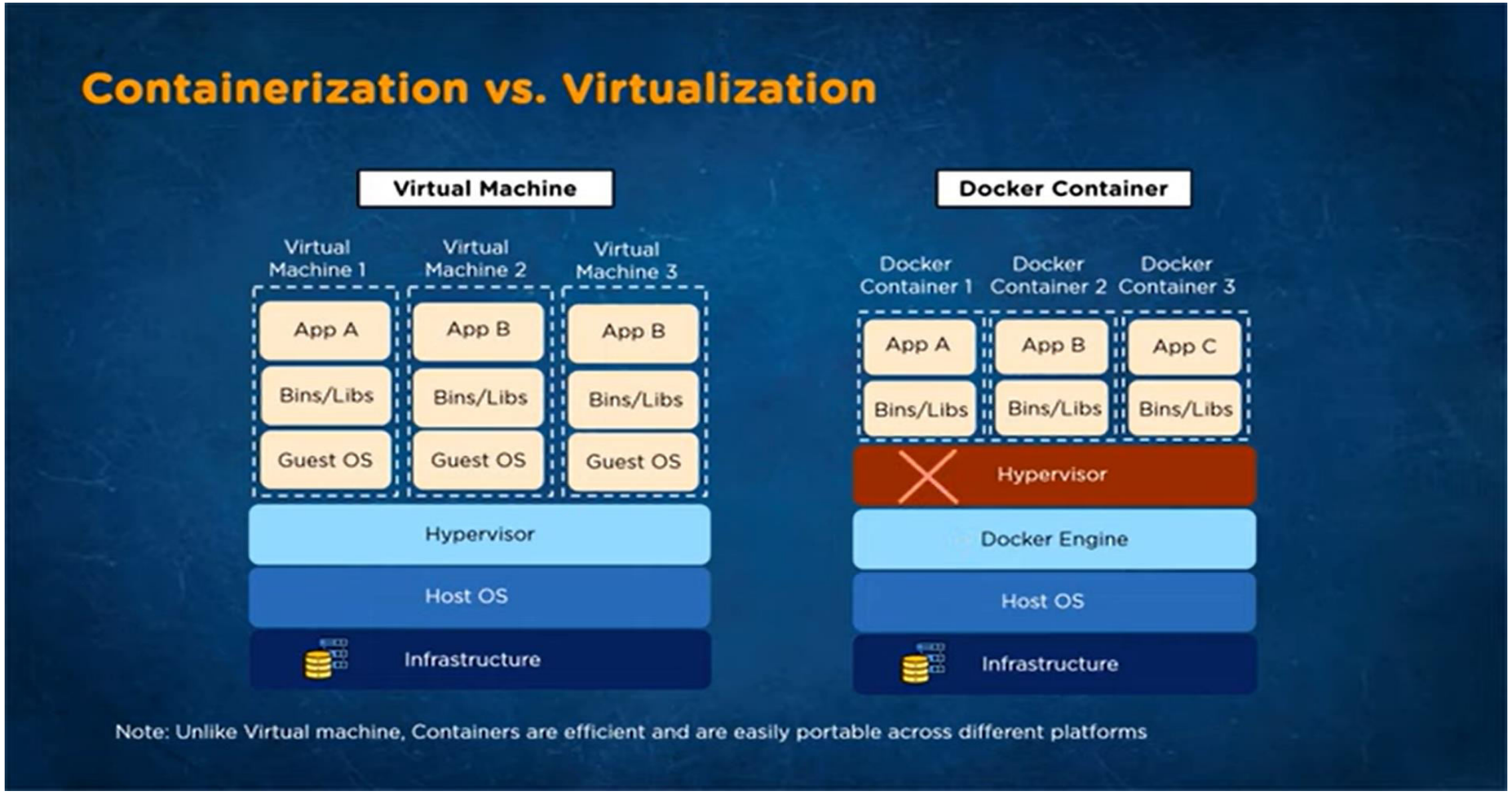
Docker


What is Docker Container?
Containerization includes all dependencies (frameworks, libraries, etc.) required to run an application in an efficient and bug-free manner
With Docker Containers, applications can work efficiently in different computer environments
Applications run in isolation
Secure
Occupies less space
Portable
Lightweight
Short boot-up time
Docker Containers

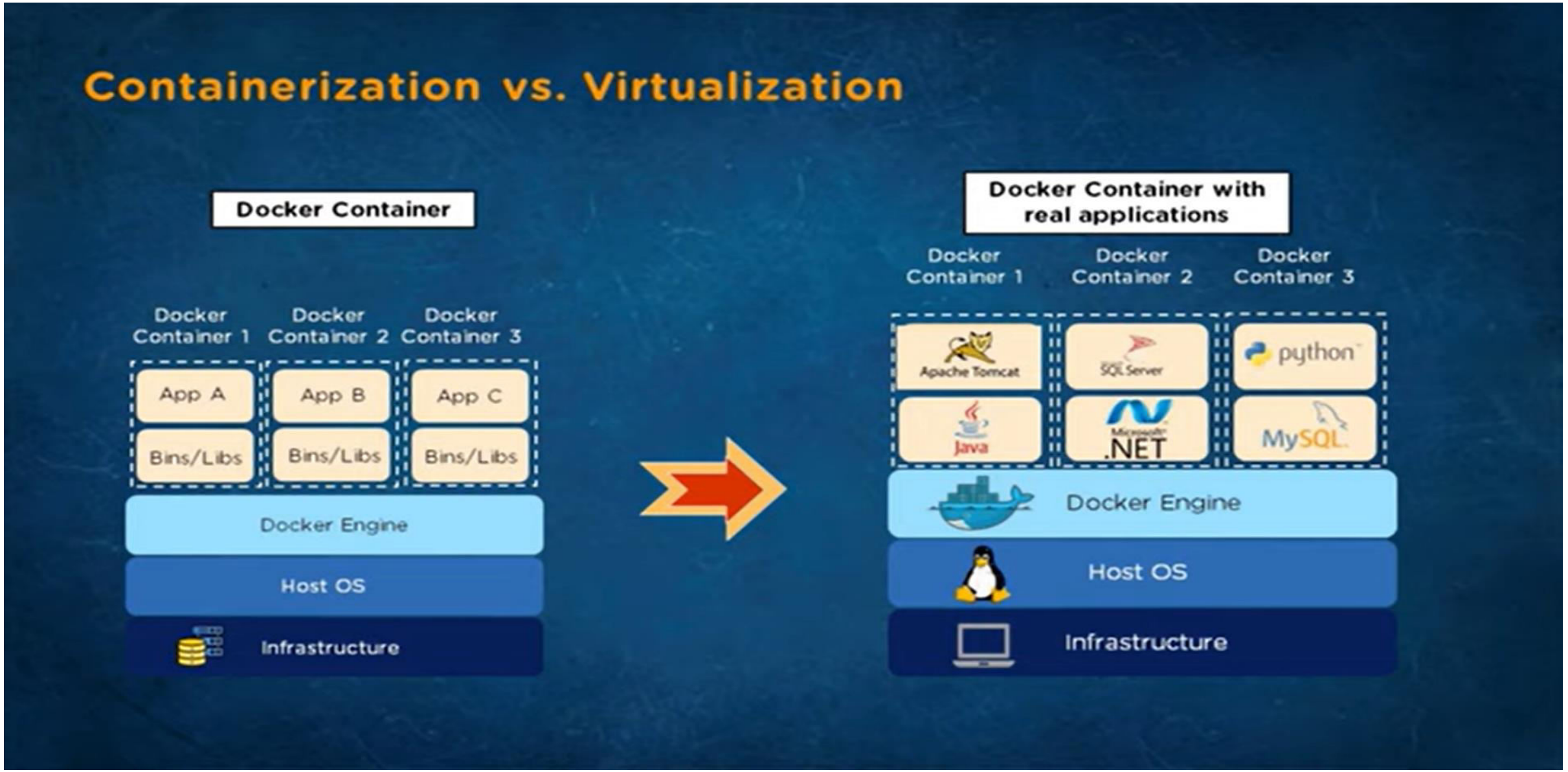
Docker


Containerization vs. Virtualization
Virtual Machine
Virtual Machine 1
Virtual Machine 2
Virtual Machine 3
App A
App B
App B
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Infrastructure
Docker Container
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Infrastructure
Note: Guest OS occupies more space and leads to unstable performance

Docker


Containerization vs. Virtualization
Virtual Machine
Virtual Machine 1
Virtual Machine 2
Virtual Machine 3
App A
App B
App B
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Infrastructure
Docker Container
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Hypervisor
Docker Engine
Host OS
Infrastructure
Note: Unlike Virtual machine, Containers are efficient and are easily portable across different platforms

Docker


Containerization vs. Virtualization
Docker Container
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Infrastructure
Docker Container with real applications
Docker Container 1
Docker Container 2
Docker Container 3
Apache Tomcat
SQL Server
python
Java
.NET
MySQL
Docker Engine
Host OS
Infrastructure

Docker

Containerization vs. Virtualization

Virtual machine

Example: Bungalow

- Amenities (binary and library) cannot be shared with neighbours (applications)
- Cannot have multiple tenants (application)

Containers

Example: Apartment

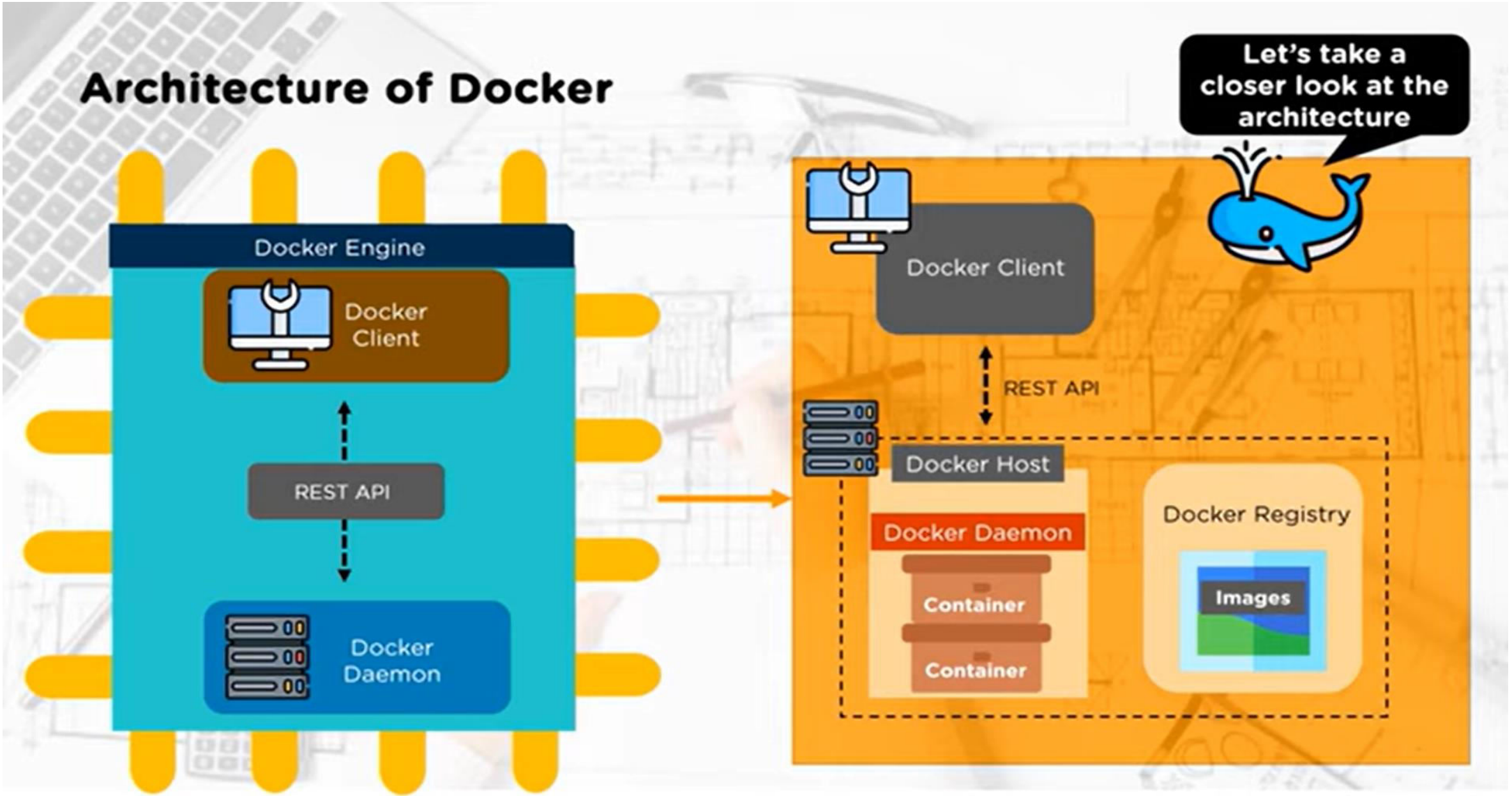
- Most amenities (binary and library) are shared with neighbours (applications)
- Can have multiple tenants (Applications)

Docker

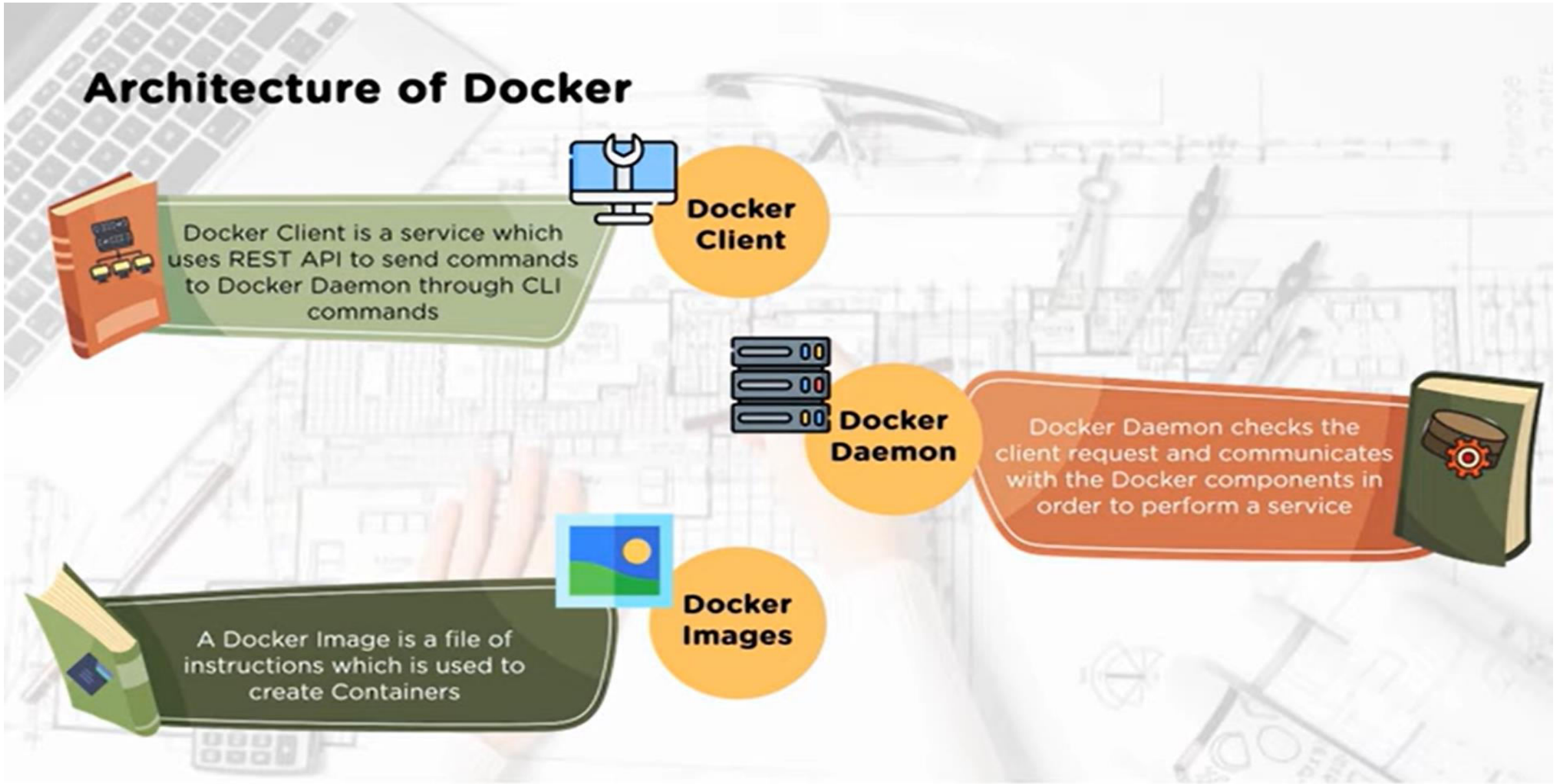
Containerization vs. Virtualization

	DOCKER CONTAINERS	VIRTUAL MACHINE
Space allocation	Data volumes can be shared and reused among multiple containers	Data volumes cannot be shared
Performance	They have a better performance as they are hosted in a single Docker engine	Running multiple virtual machines leads to unstable performance
Efficiency	High efficiency	Low efficiency
Capability	It can run multiple containers on a system	It can run only a limited number of VMs on a system
Portability	Easily portable across different platforms	Compatibility issues while porting across different platforms

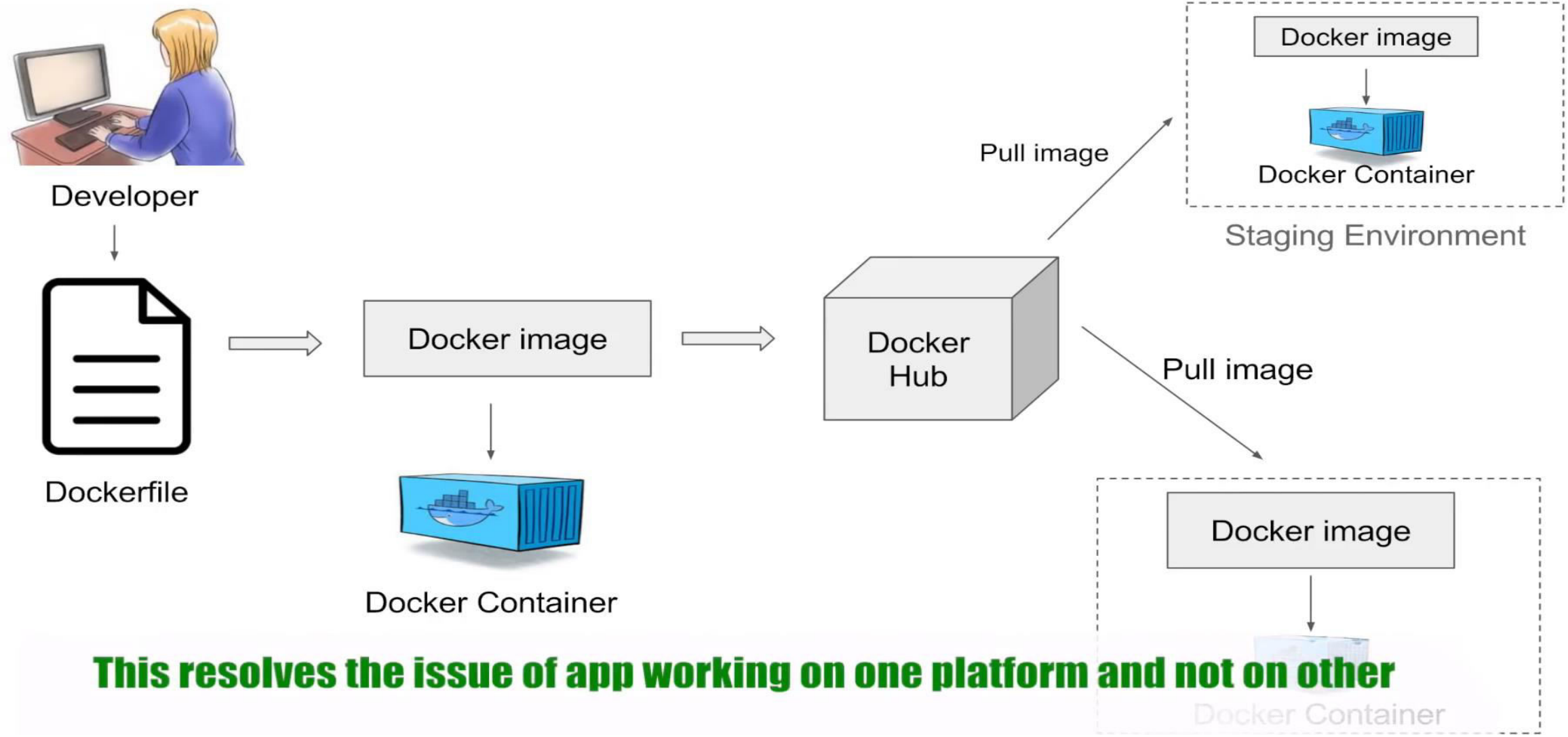
Docker


Architecture of Docker
Docker Engine
Docker Client
REST API
Docker Daemon
Let's take a closer look at the architecture
Docker Client
REST API
Docker Host
Docker Daemon
Container
Container
Docker Registry
Images

Docker


Architecture of Docker
Docker Client
Docker Client is a service which uses REST API to send commands to Docker Daemon through CLI commands
Docker Daemon
Docker Daemon checks the client request and communicates with the Docker components in order to perform a service
Docker Images
A Docker Image is a file of instructions which is used to create Containers

Docker


Developer
Dockerfile
Docker image
Docker Container
Docker Hub
Pull image
Docker image
Docker Container
Staging Environment
Pull image
Docker image
Docker Container
This resolves the issue of app working on one platform and not on other

Docker

What is Docker?

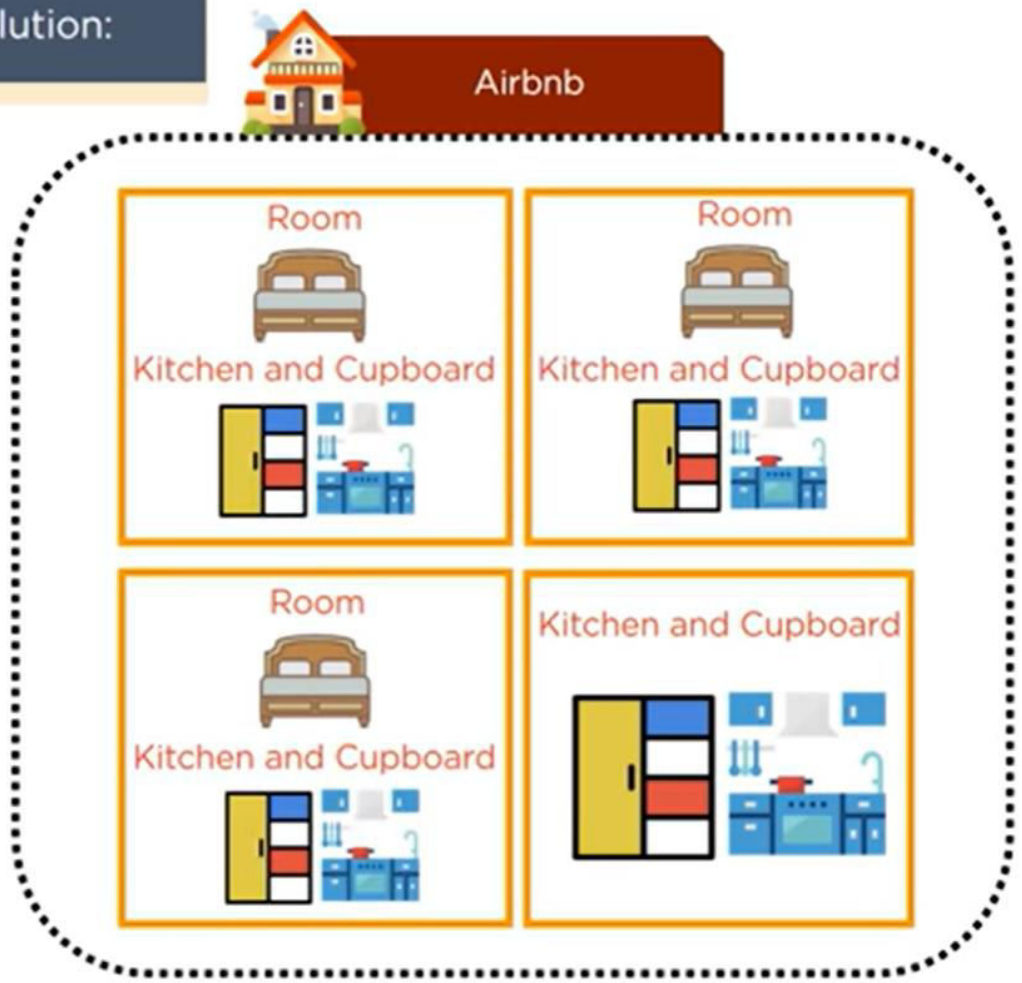
Let's take an example where you plan to rent a house in Airbnb

Because every individual has different preferences when it comes to the cupboard and the kitchen usage

Docker

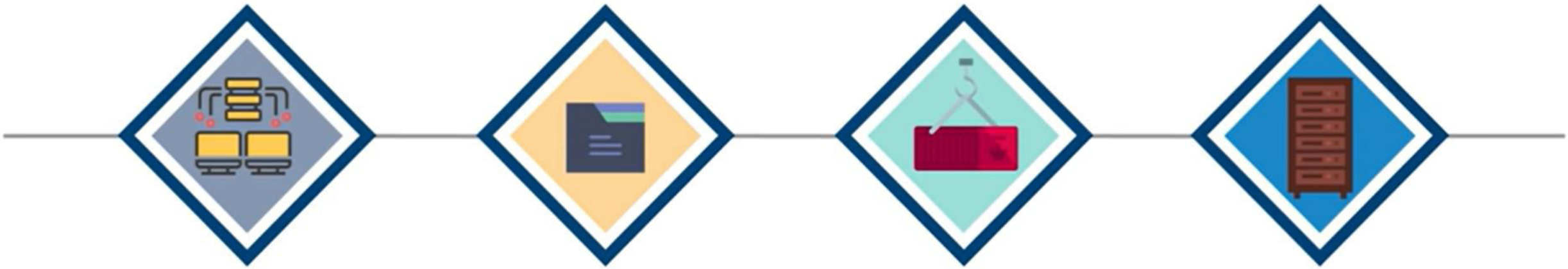
What is Docker?

Solution:



Docker

Components of Docker



Docker Client and Server

Docker Images

Docker Containers

Docker Registry

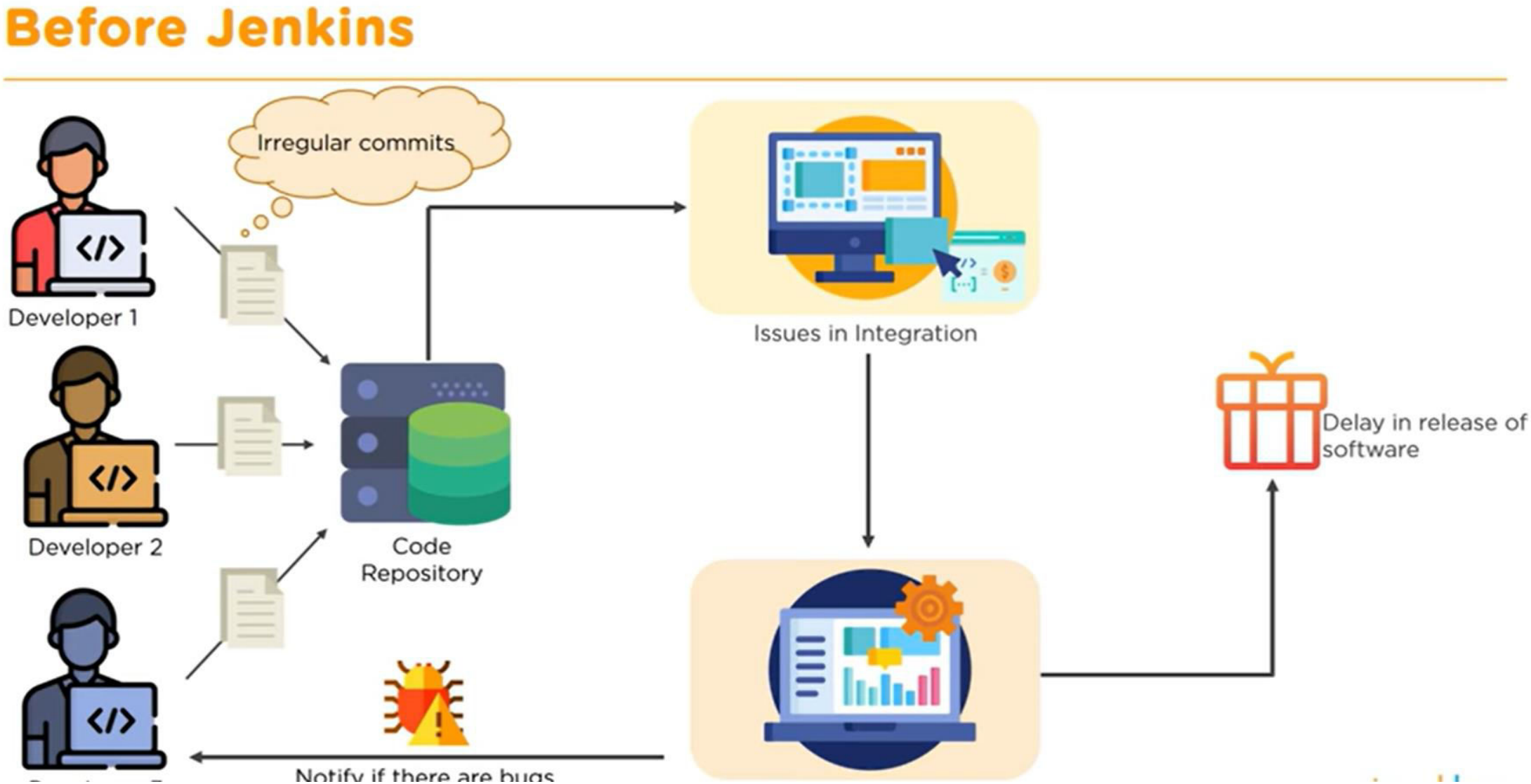
Jenkins

What's in it for you?

- Before Jenkins
- Issues before Jenkins
- What is Jenkins?
- What is Continuous Integration?

- Continuous Integration Tools
- Features of Jenkins
- Jenkins Architecture
- Jenkins Case Study

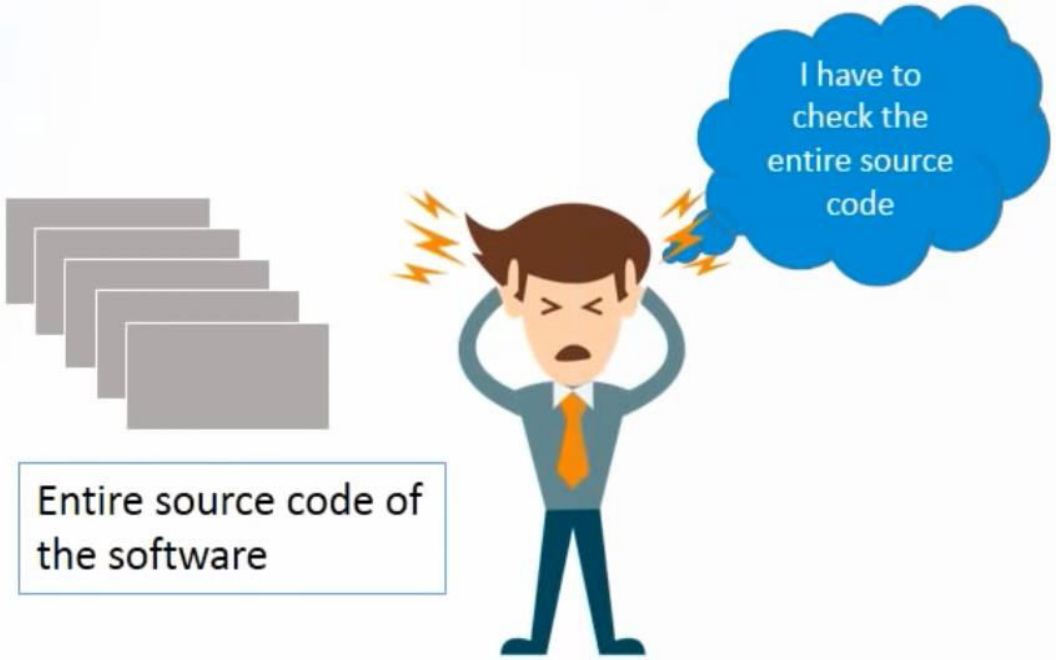
Jenkins


Before Jenkins
Irregular commits
Developer 1
Developer 2
Code Repository
Issues in Integration
Delay in release of software
Notify if there are bugs

Jenkins

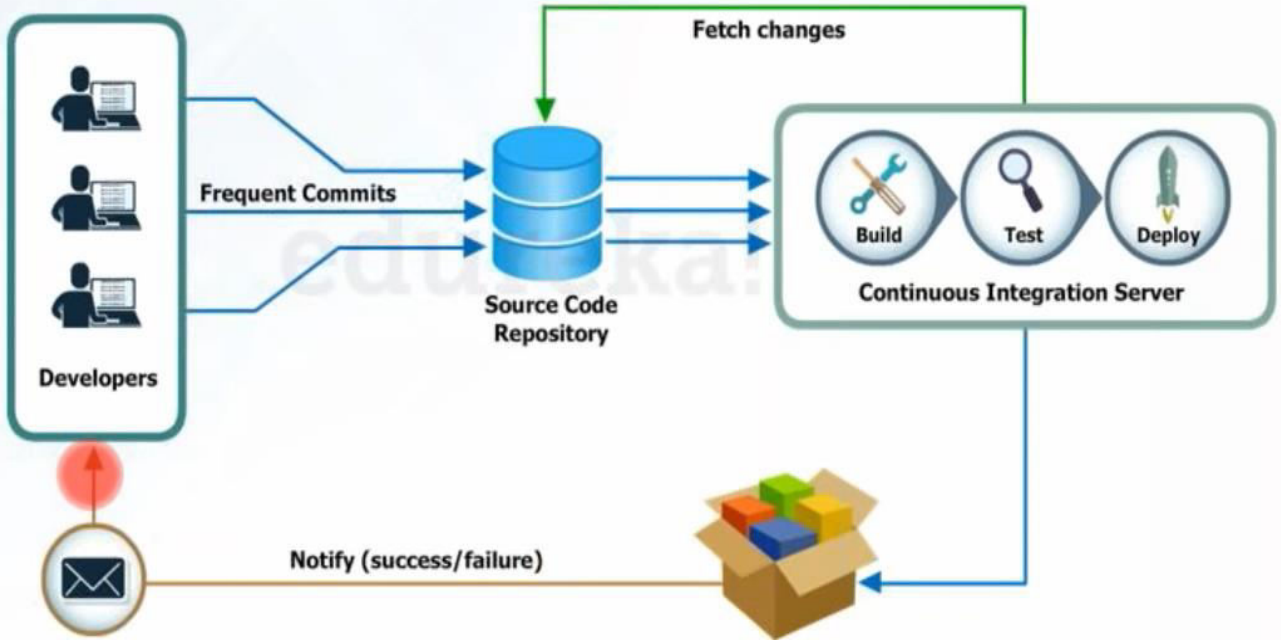
Developers have to wait till the complete software is developed for the test results.

If the test fails then locating and fixing bugs is very difficult. Developers have to check the entire source code of the software.



Jenkins

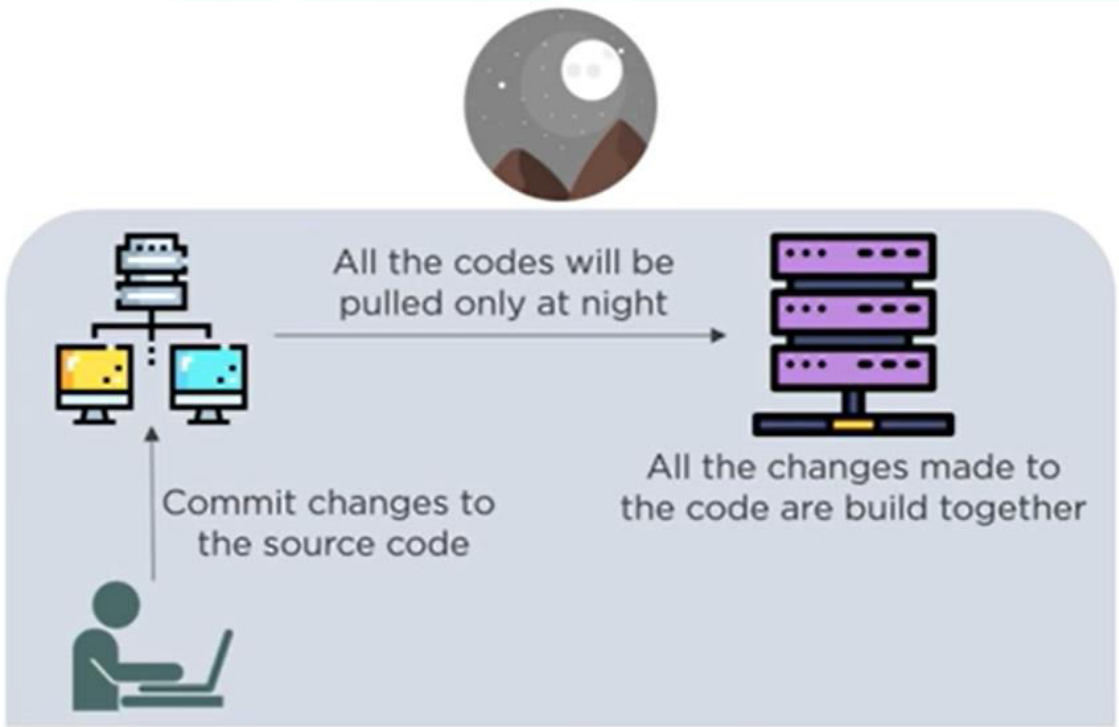
- Since after every commit to the source code an auto build is triggered and then it is automatically deployed on the test server
- If the test results shows that there is a bug in the code then the developers only have to check the last commit made to the source code
- This also increases the frequency of new software releases
- The concerned teams are always provided with the relevant feedback



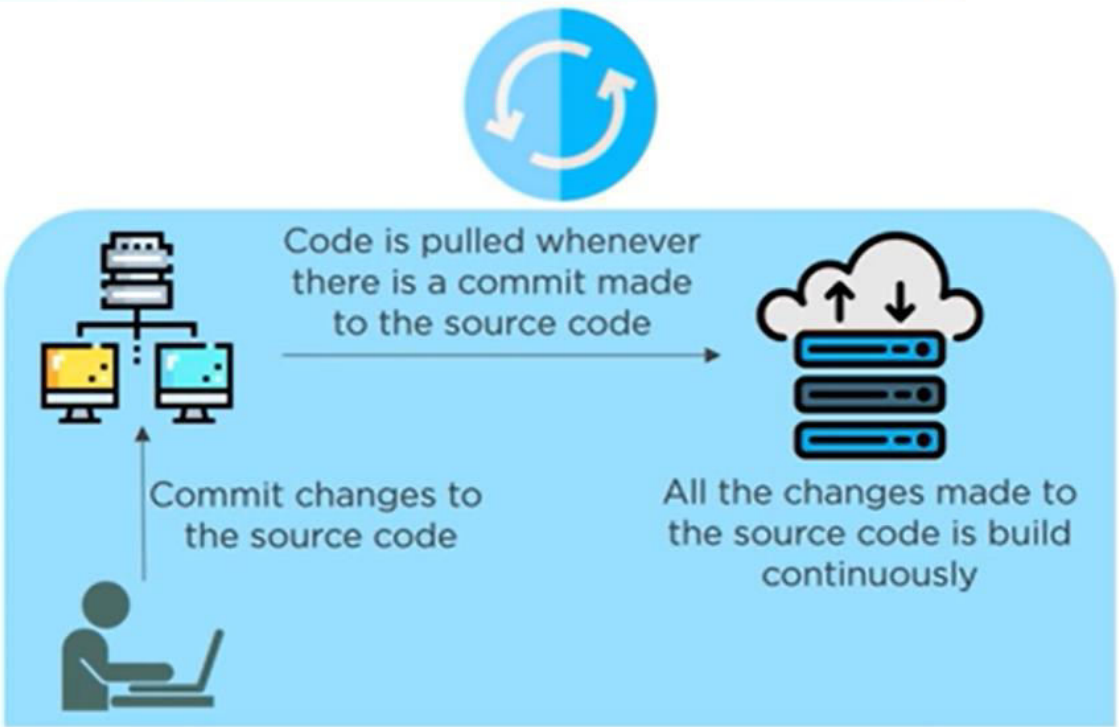
Jenkins

What is Jenkins?

Jenkins is a Continuous Integration tool that allows continuous development, test and deployment of newly created codes



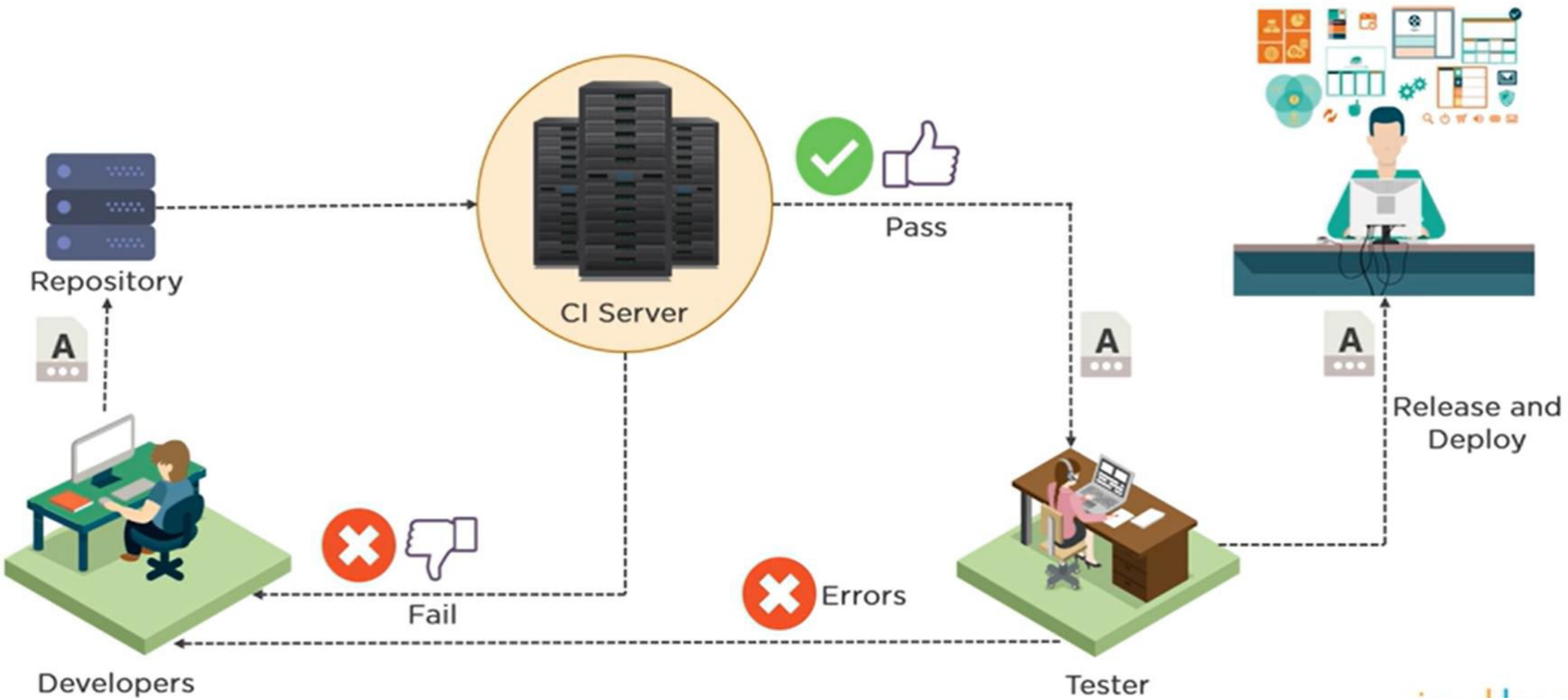
Nightly build and integration



Continuous build and Integration

Jenkins

What is Continuous Integration?

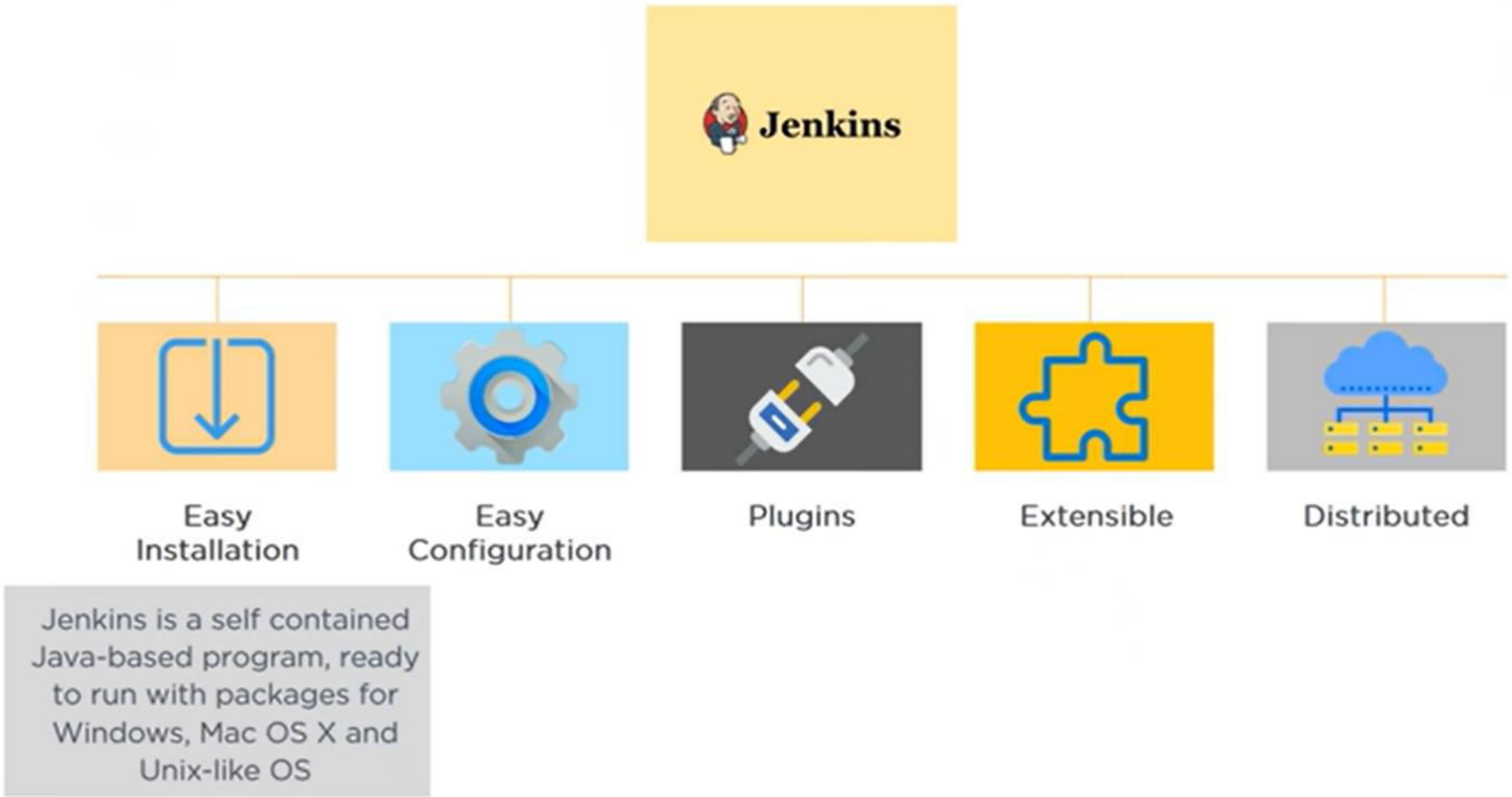


Jenkins

Continuous Integration Tools

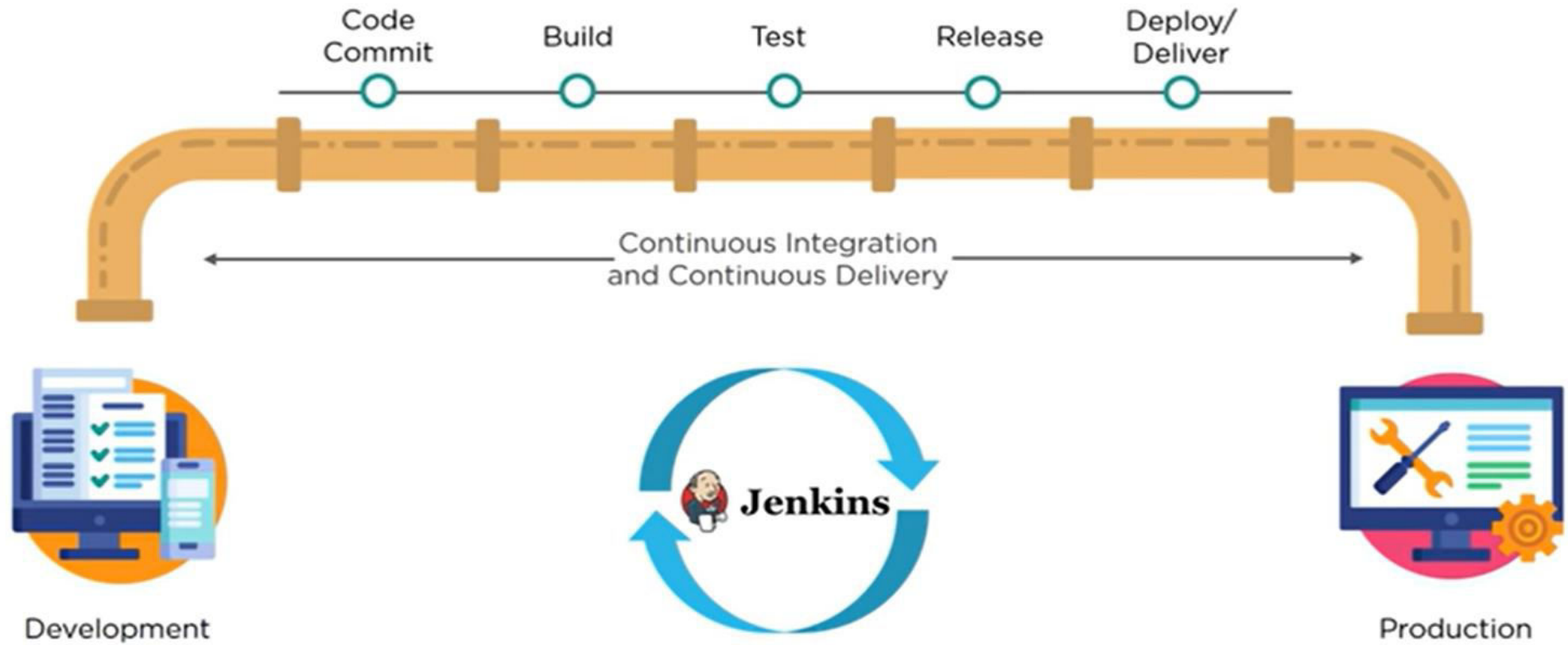
Jenkins

Features of Jenkins



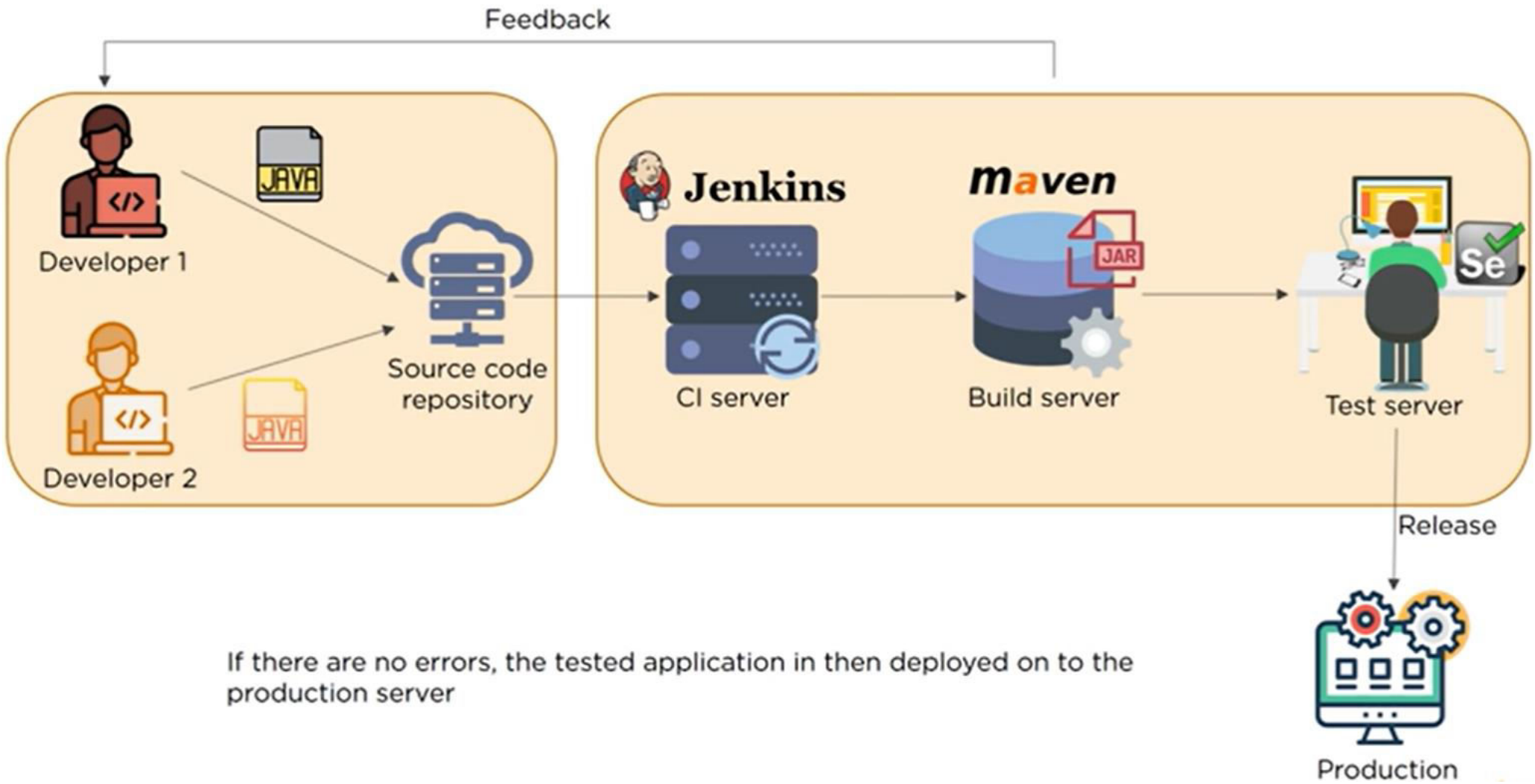
Jenkins

Jenkins Pipeline



Jenkins

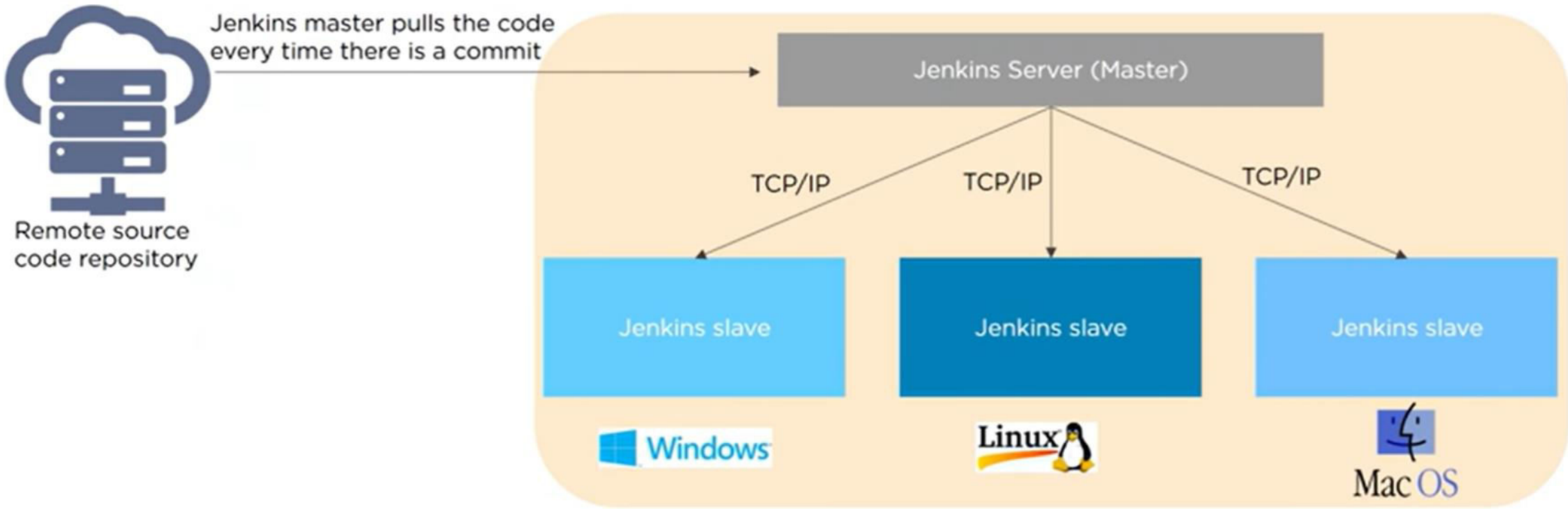
Jenkins Architecture



If there are no errors, the tested application in then deployed on to the production server

Jenkins

Jenkins Master-Slave Architecture



- Jenkins master distributes its workload to all the slaves
- On request from Jenkins master, the slaves carry out builds and tests and produce test reports

Jenkins

Jenkins Case Study

Automotive systems are becoming more sophisticated and complex, so the focus for automotive manufacturers is shifting from hardware to software

Many vehicle features, capabilities and performance enhancements are being implemented in software

Jenkins

Jenkins Case Study

BOSCH

BOSCH found a growing need to help its software engineers produce and deliver higher quality software faster

CHALLANGE

Manage and streamline the development of increasingly complex automotive software by adopting CI and CD practices to shorten the entire development and delivery process

Jenkins

Jenkins Case Study

BOSCH

CloudBees
The Enterprise Jenkins Company

RESULTS

- 3 day build process reduced to less than 3 hours
- Large scale deployment kept on track by expert support
- Visibility and transparency improved with Jenkins Operations support

YAML

YAML is a human-readable data serialization language that is often used for writing configuration files. Depending on whom you ask, YAML stands for yet another markup language or YAML ain't markup language (a recursive acronym), which emphasizes that YAML is for data, not documents.

Many developers consider it easier to learn than JSON because it's written using natural language. YAML is a superset of JSON. It was developed around the same time to handle more kinds of data and offer a more complex but still readable syntax.

YAML is a popular programming language because it is designed to be easy to read and understand. It can also be used in conjunction with other programming languages. Because of its flexibility and accessibility

YAML

Differences

XML

```xml
<People>
    <Person>
        <name>Bob</name>
        <age>21</age>
        <address>Boca Raton, Florida(FL), 33432</address>
    </Person>
</People>
```

JSON

```json
{
    "People": [
        {
            "name": "Bob",
            "age": 21,
            "address": "Boca Raton, Florida(FL), 33432"
        }
    ]
}
```

YAML

```yaml
People:
  - name: Bob
    age: 21
    address: Boca Raton, Florida(FL), 33432
```

YAML

Key Value Pairs

Key Value Pair

```
Animal: Dog
Fruit: Apple
Meal: "Vegetable Lasagna"
```

YAML

Arrays / Lists

Arrays / Lists

```yaml
Drinks:
  - Smoothie
  - Water
Meals:
  - "Vegetable Lasagna"
  - "Tomato Soup"
  - Burger
```

YAML

Directories / Maps

Directories / Maps

```yaml
Burger:
  Calories: 200
  Price: 15
Smoothie:
  Calories: 80
  Price: 5
```

YAML

Combining everything together

Arrays with Directories

```yaml
Drinks:
  - Smoothie:
      Calories: 80
      Price: 5
Meals:
  - Burger:
      Calories: 200
      Price: 15
```

YAML

Spacing

Key Value Pair

```yaml
Animal: Dog
Fruit: Apple
Meal: "Vegetable Lasagna"
```

Directories / Maps

```yaml
Burger:
  Calories: 200
  Price: 15
Smoothie:
  Calories: 80
  Price: 5
```

Arrays / Lists

```yaml
Drinks:
  - Smoothie
  - Water
Meals:
  - "Vegetable Lasagna"
  - "Tomato Soup"
  - Burger
```

Lab

YAML

Comments

Arrays with Directories

```yaml
# This is single line comment.
Drinks:
  - Smoothie:
      Calories: 80
      Price: 5
Meals:
  - Burger:
      Calories: 200
      Price: 15
```

Kubernetes

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management. Originally designed by Google, the project is now maintained by the Cloud Native Computing Foundation. The name Kubernetes originates from Greek, meaning 'helmsman' or 'pilot'.

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications

Kubernetes automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, monitoring your applications, and more—making it easier to manage applications.

Kubernetes is the most popular container orchestration platform and has become an essential tool for DevOps teams. Application teams can now deploy containerized applications to Kubernetes clusters, which can run either on-premises or in a cloud environment.

Kubernetes creates and manages containers on the cloud-based server systems. Kubernetes helps DevOps teams to reduce the burden of infrastructure by letting containers operate on different machines/environments without breakdowns

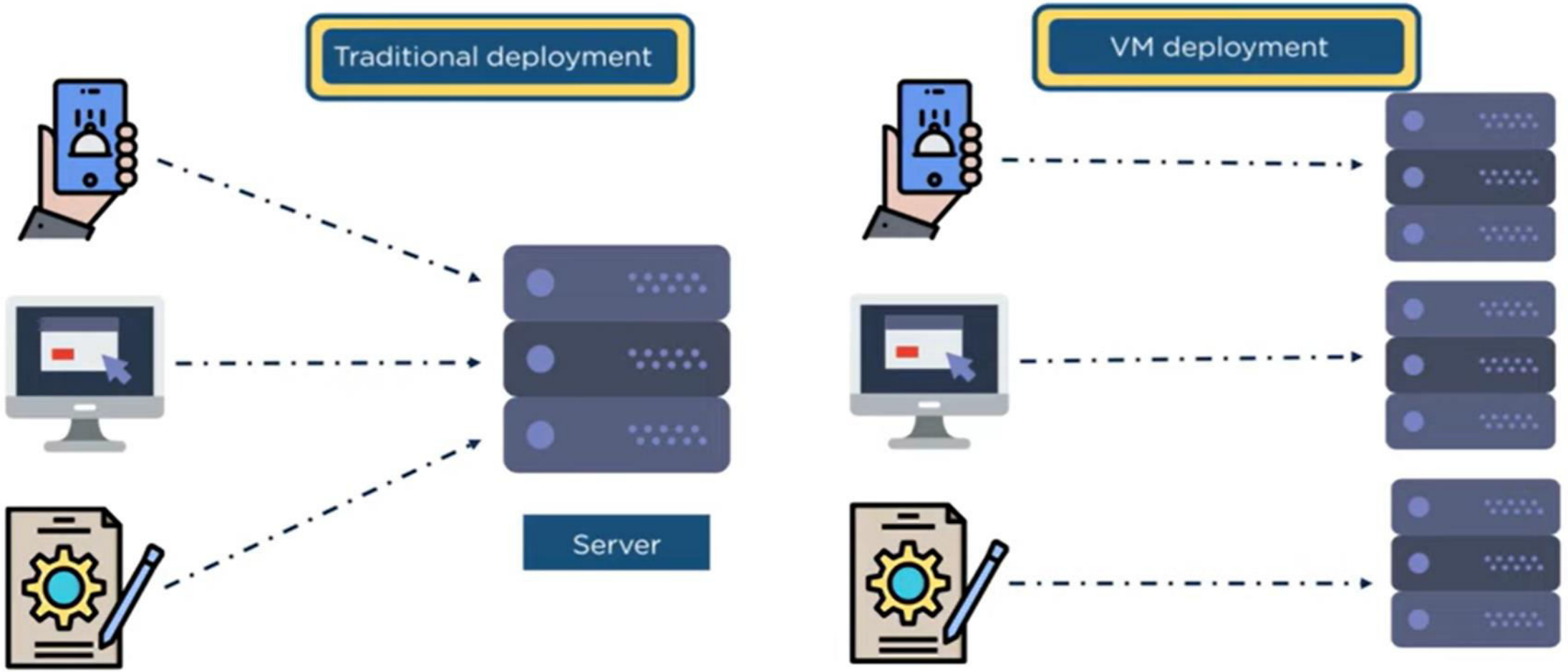
Kubernetes

What's in it for you?

- Before Kubernetes
- What is Kubernetes?
- Benefits of Kubernetes
- Kubernetes Architecture and Working

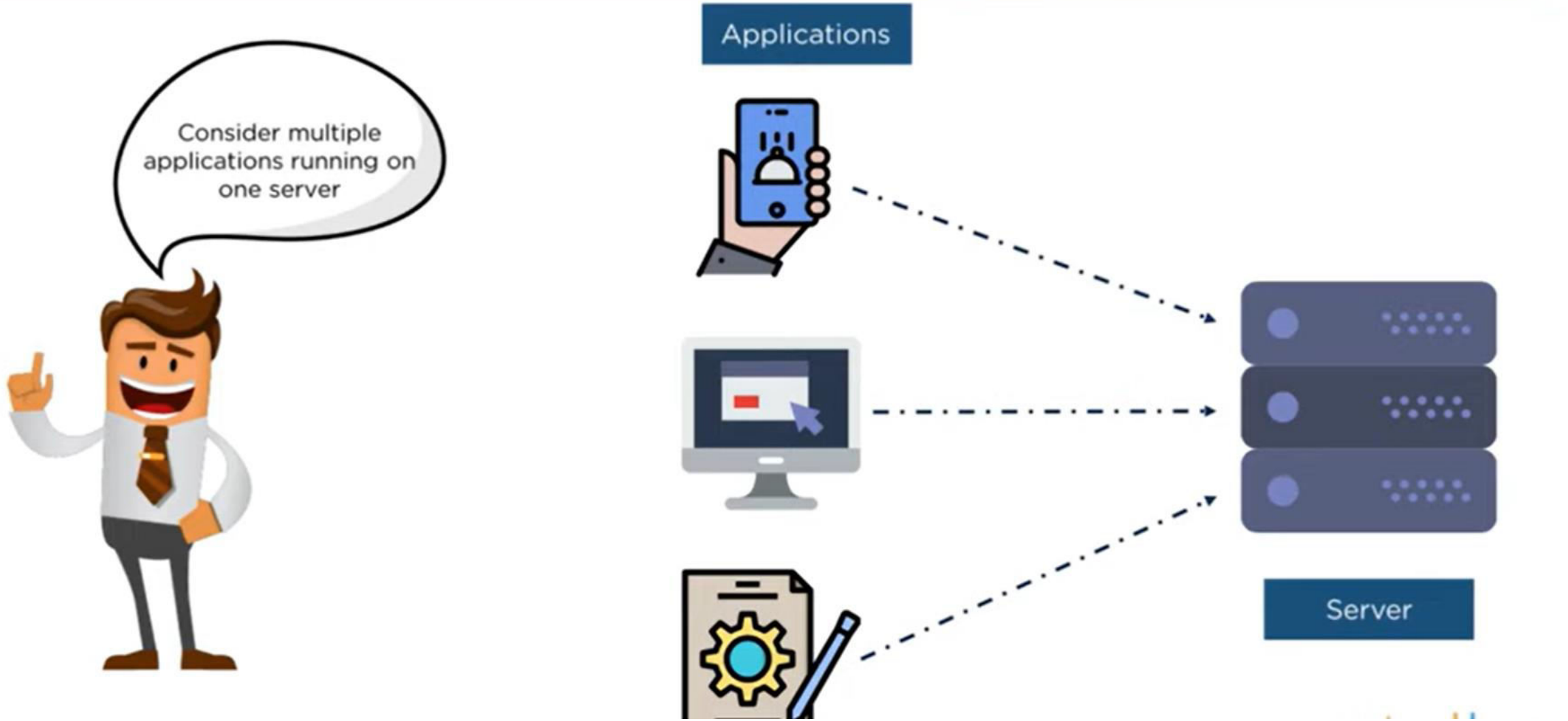
Kubernetes

Before Kubernetes



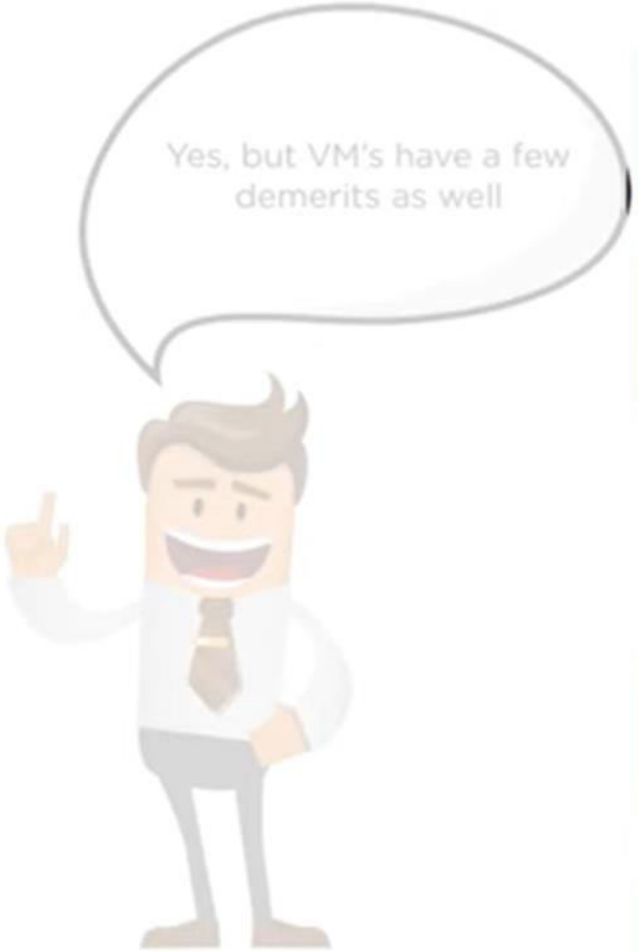
Kubernetes

Before Kubernetes – Traditional Deployment



Kubernetes

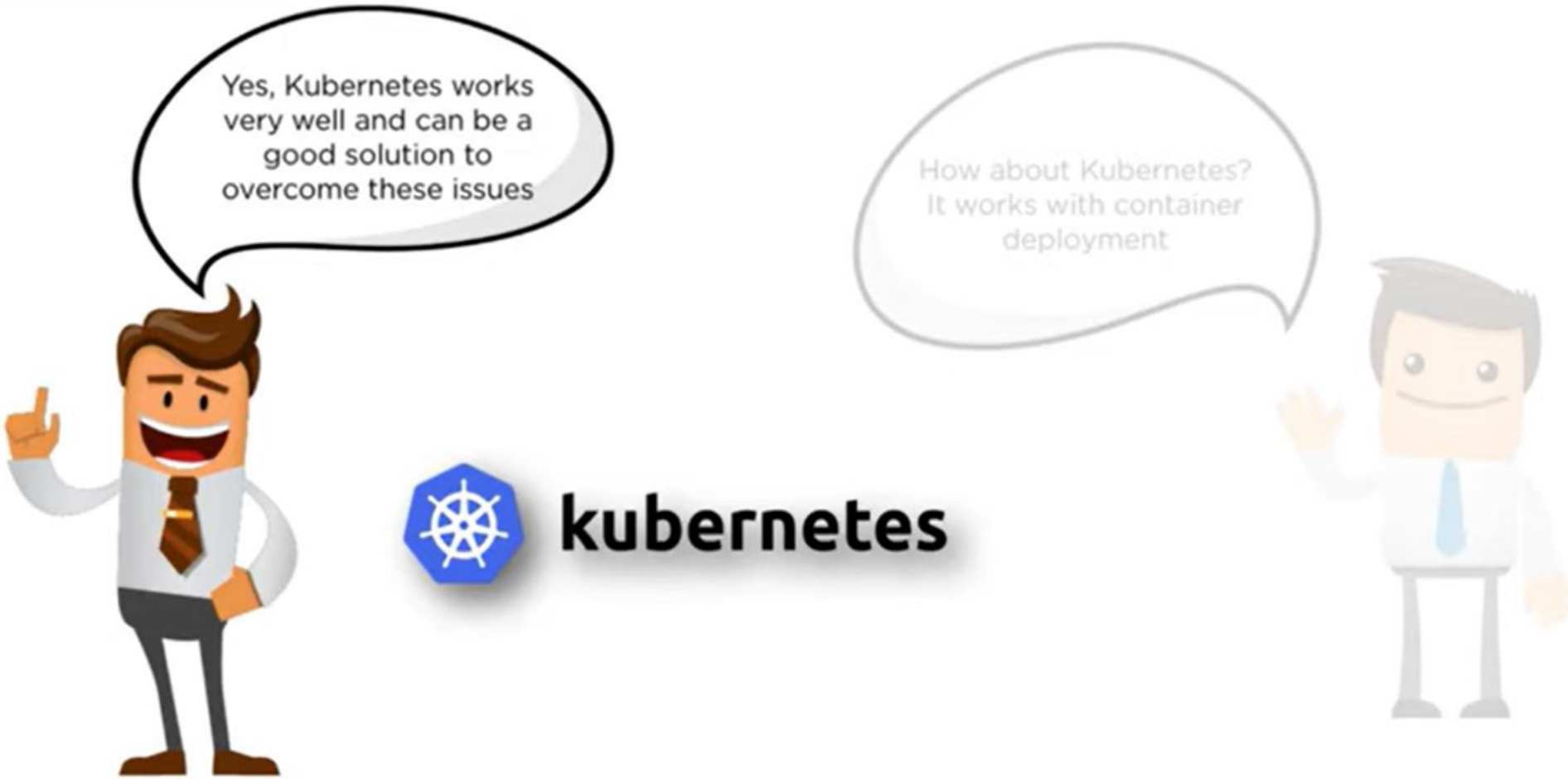
Before Kubernetes – Virtualization Deployment



High cost of implementation	Hardware and software components are required by the VM's that increases the cost of implementation
Security risk	The chances of data breach while using virtualization is common
Availability issue	Assets are unavailable most of the time because the resources are being used by other applications
Limitations and restrictions	Not every application or server can work in a virtualization environment. Some may require hybrid machines to work with
Time-consuming	It is a very time-consuming process. This is because extra steps are followed to generate the desired results

Kubernetes

After Kubernetes



Kubernetes

Virtual Machine vs Kubernetes

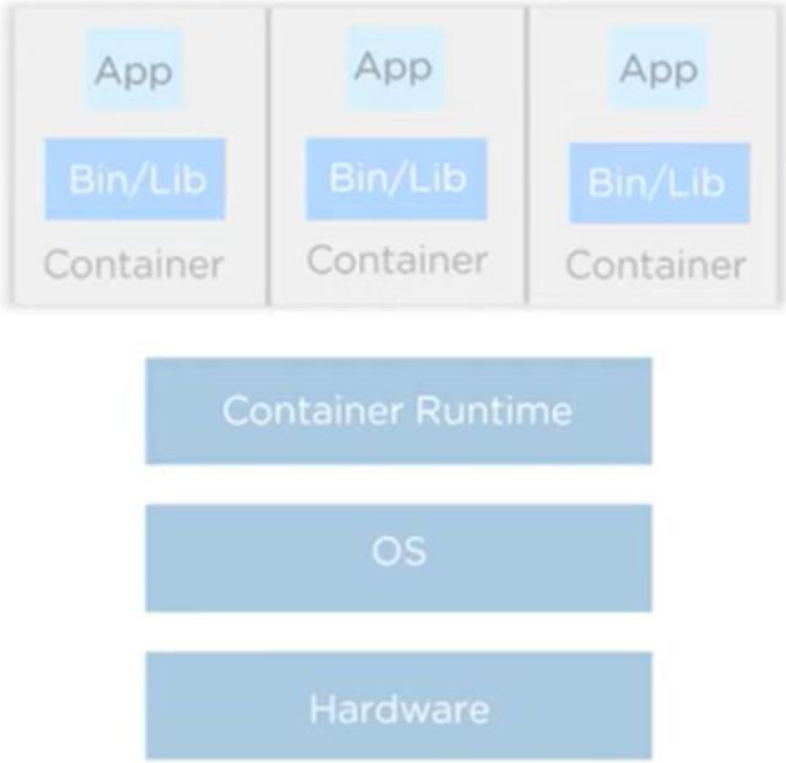
Major differences are:

Kubernetes

Virtual Machine vs Kubernetes

VM – They have less isolation when compared to Kubernetes, and hence, security risks are higher

Kubernetes - They have better isolation properties to share the OS among various applications

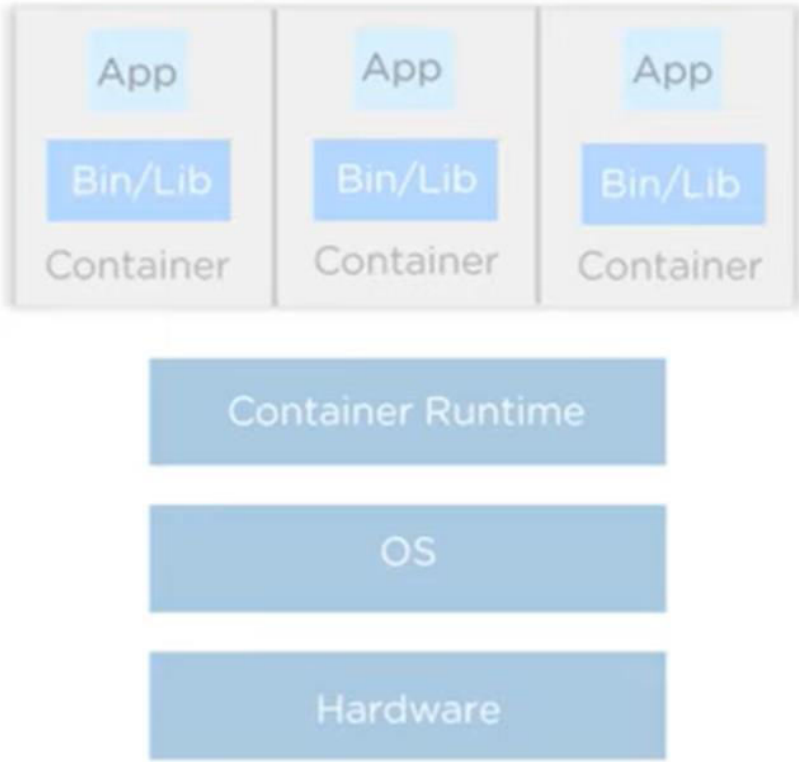


Kubernetes

Virtual Machine vs Kubernetes

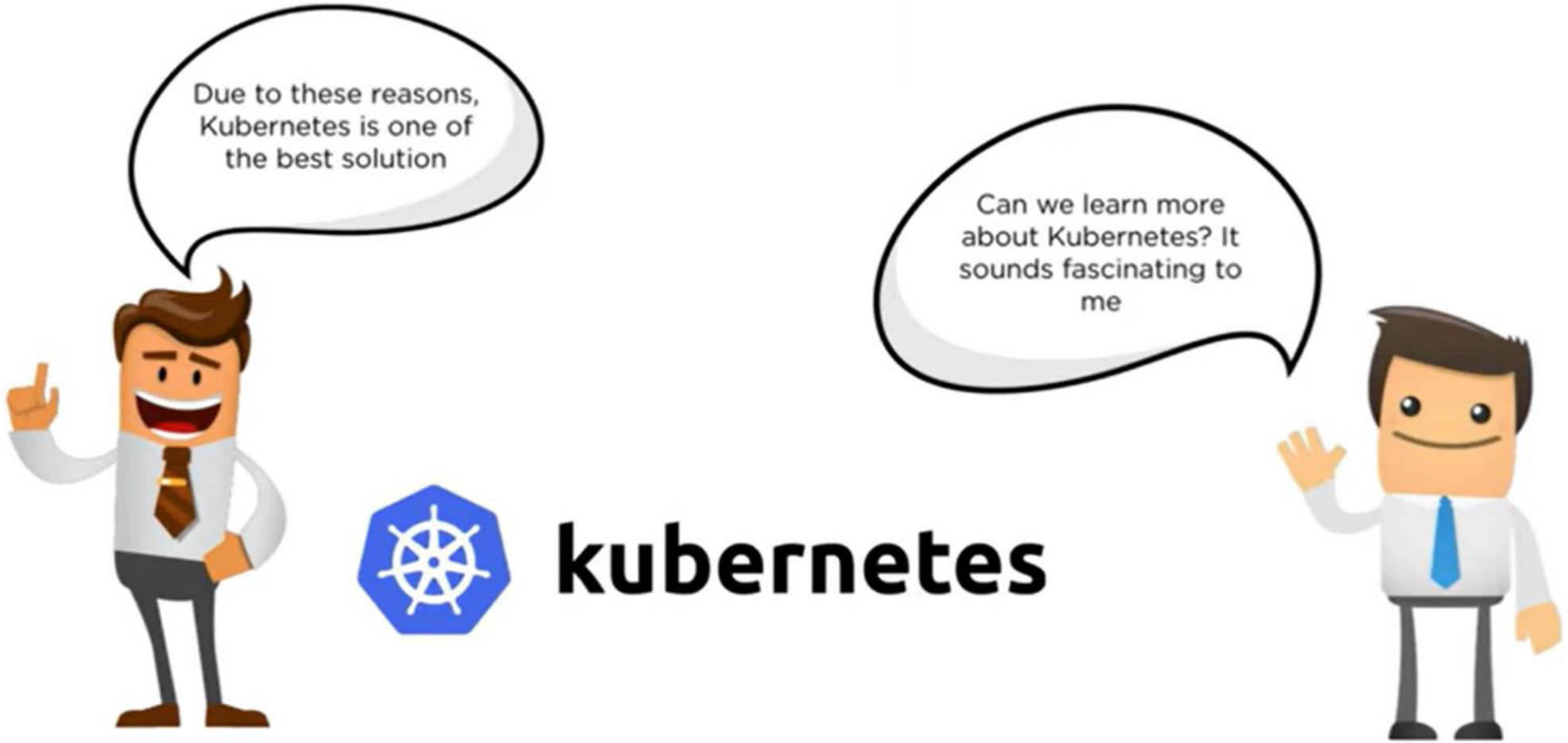
VM – They are portable but not as much as Kubernetes

Kubernetes – They are easily portable and could work on any platform or environment



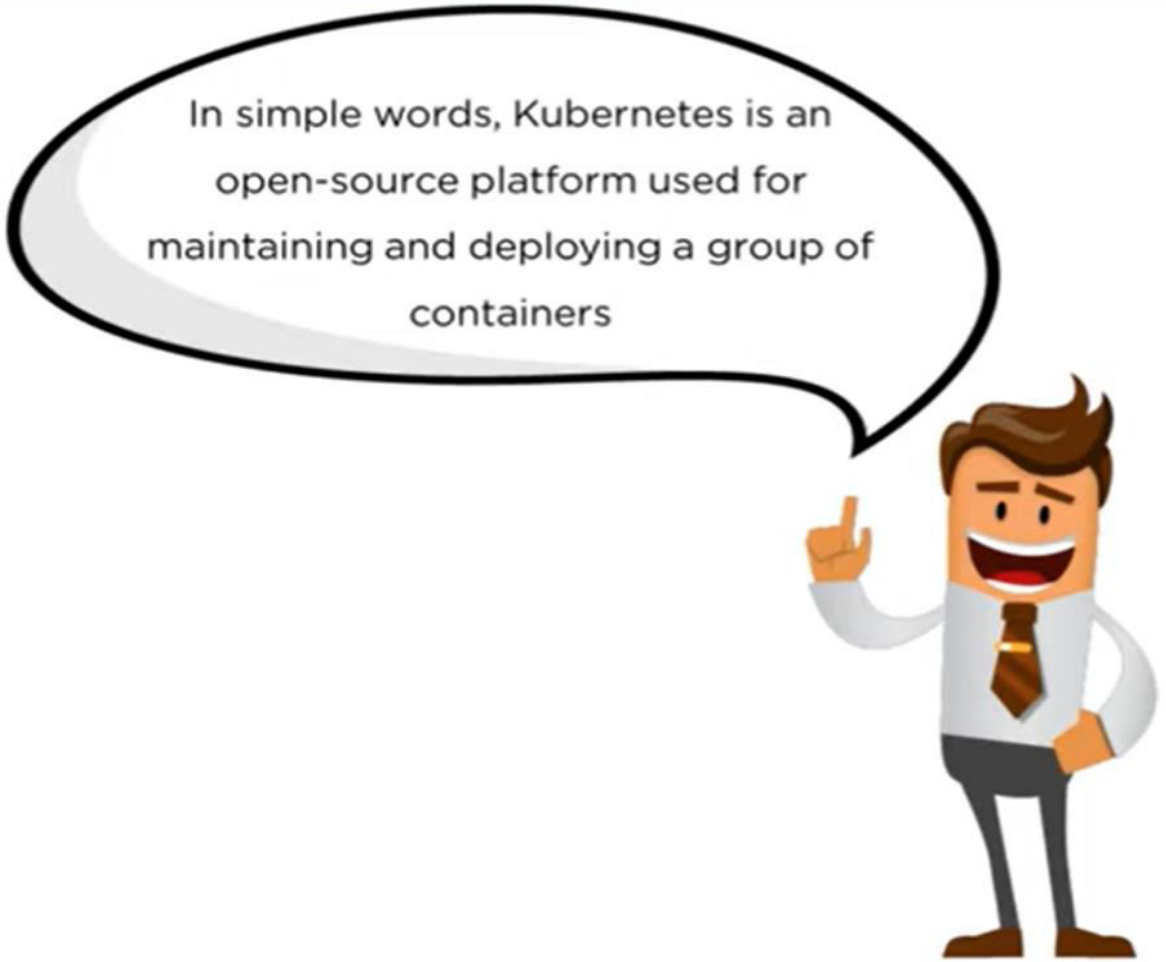
Kubernetes

Kubernetes Era



Kubernetes

What is Kubernetes?



kubernetes

Kubernetes

What is Kubernetes?

In practice, Kubernetes is most commonly used alongside Docker for better control and implementation of containerized applications

Note: Containerized Applications means bundling an application together with all its files, libraries, and packages required for it to run reliably and efficiently on different platforms

Kubernetes

Did you know?

Kubernetes was introduced as a project at Google, and it was a successor of Google Borg

It was started in 2014 because architecture of Kubernetes made it convenient for applications to run on the cloud

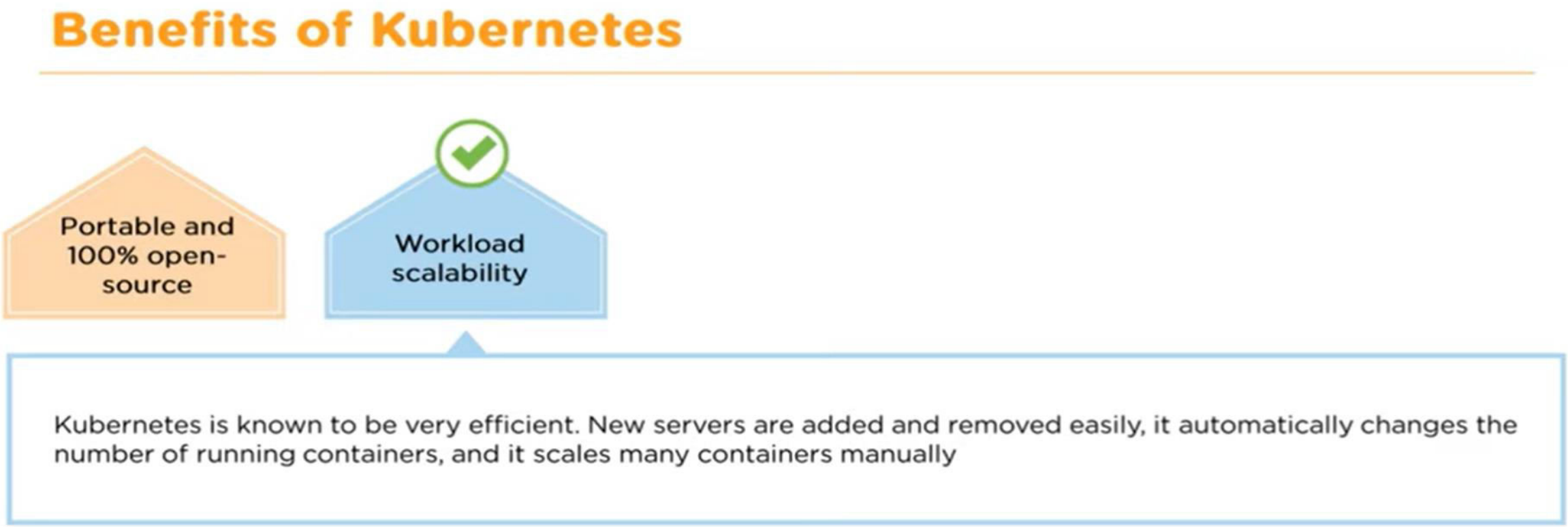
Cloud-Native Computing Foundation has currently maintained Kubernetes

Kubernetes

Benefits of Kubernetes

Kubernetes is compatible across several platforms. It is 100% open source project and thus, provides greater flexibility

Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
Kubernetes is known to be very efficient. New servers are added and removed easily, it automatically changes the number of running containers, and it scales many containers manually

Kubernetes

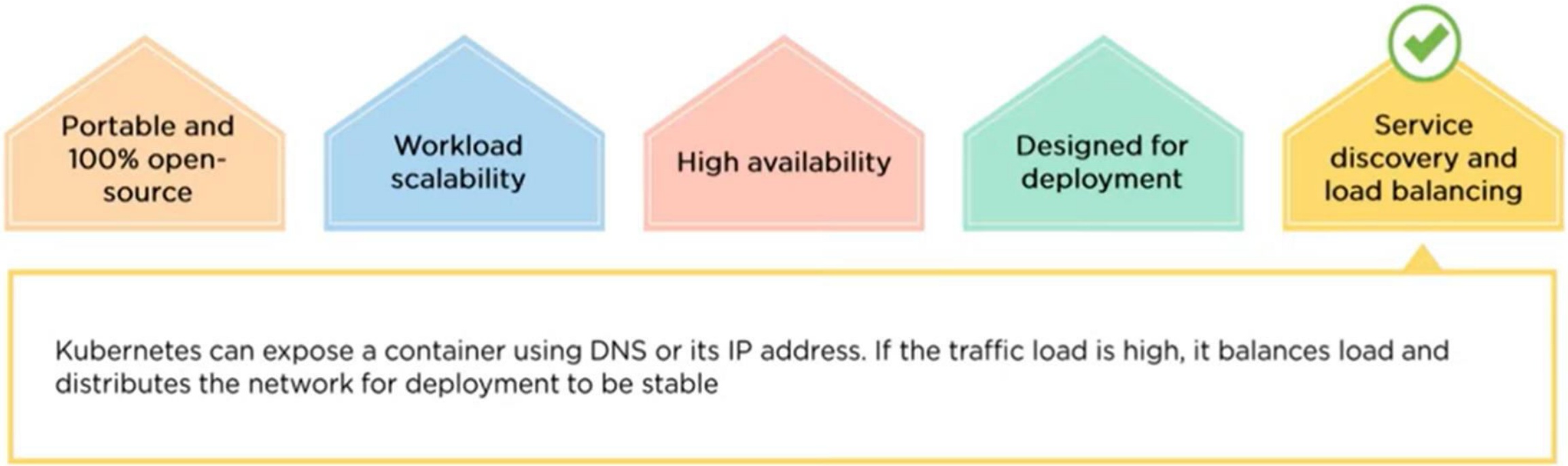
Benefits of Kubernetes

Kubernetes

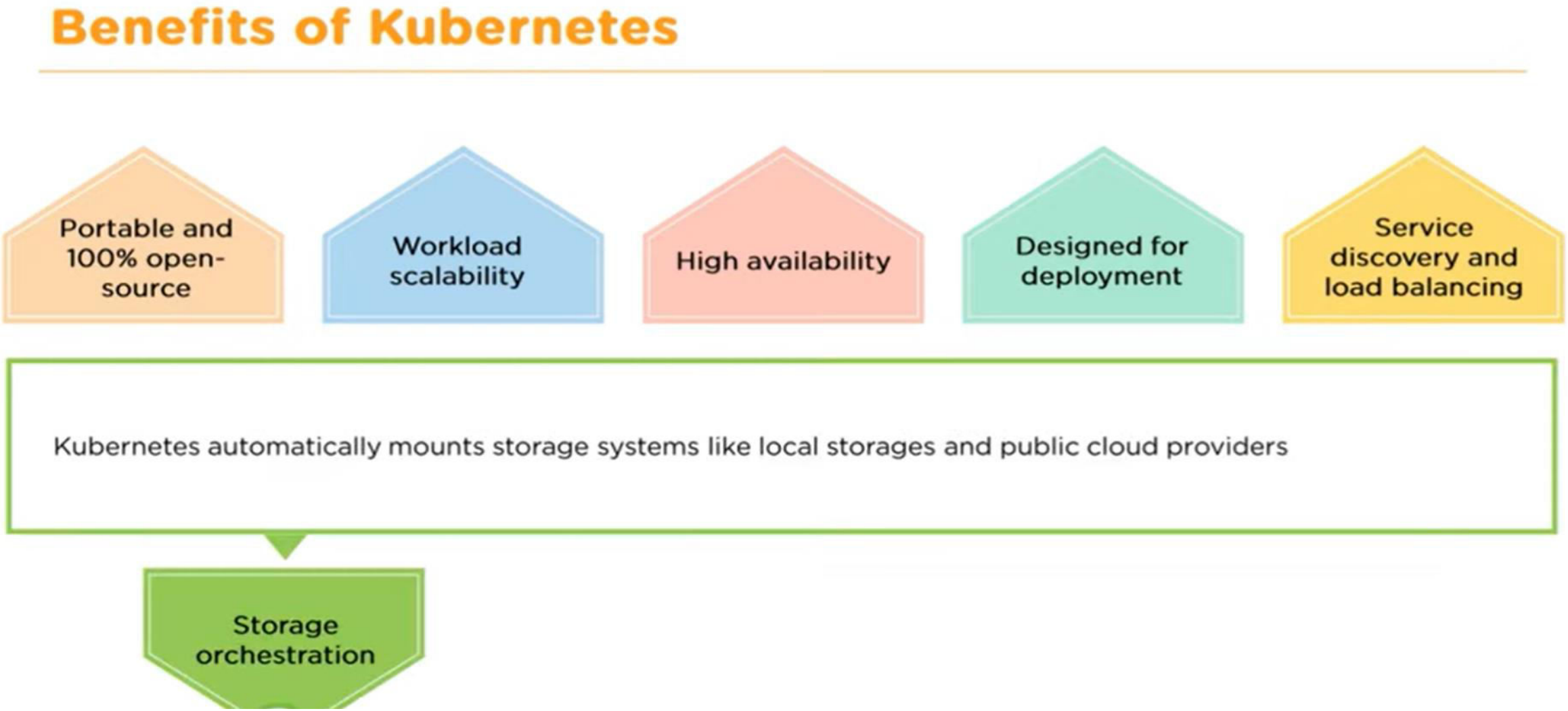
Benefits of Kubernetes

Kubernetes

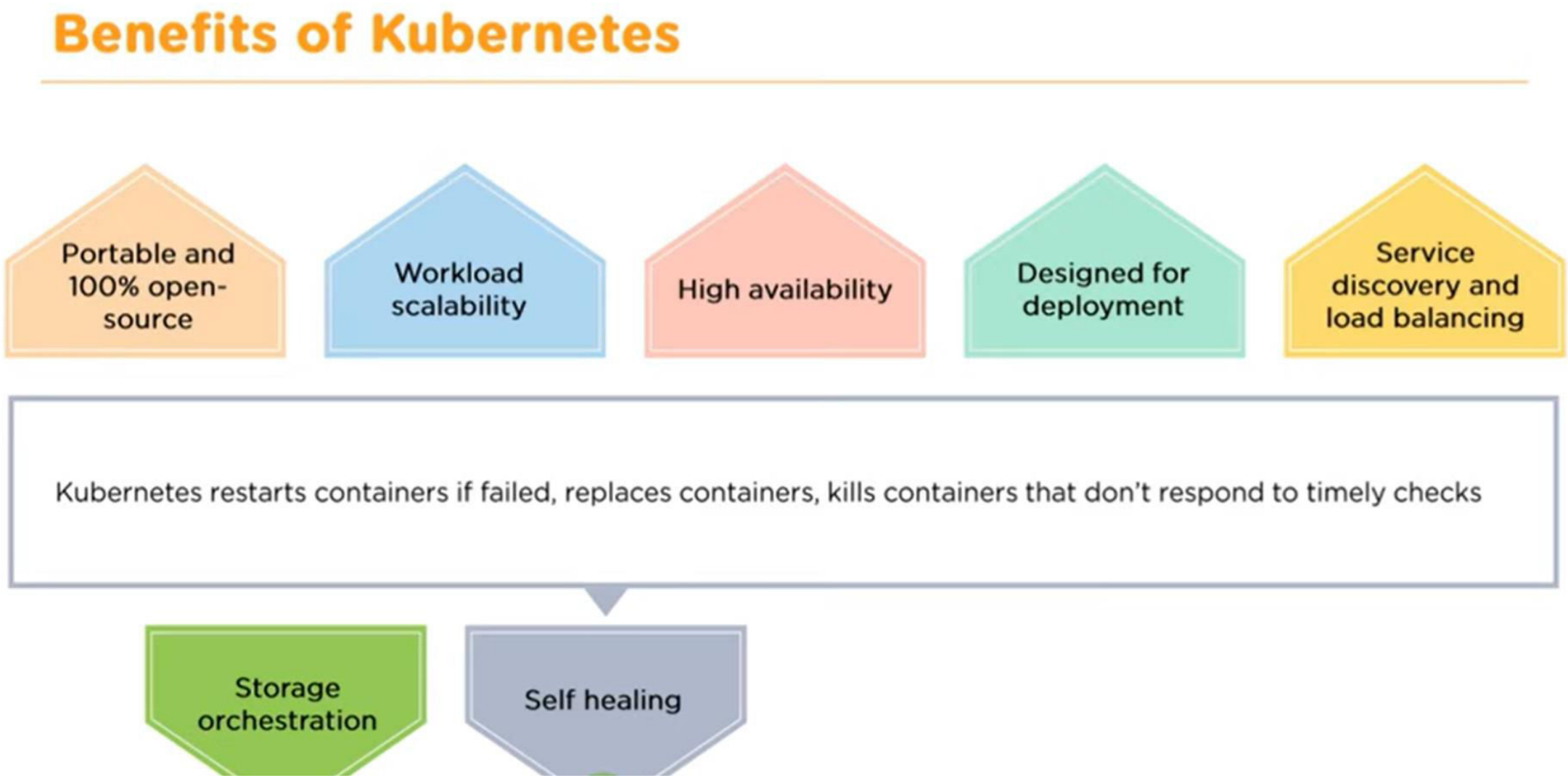
Benefits of Kubernetes



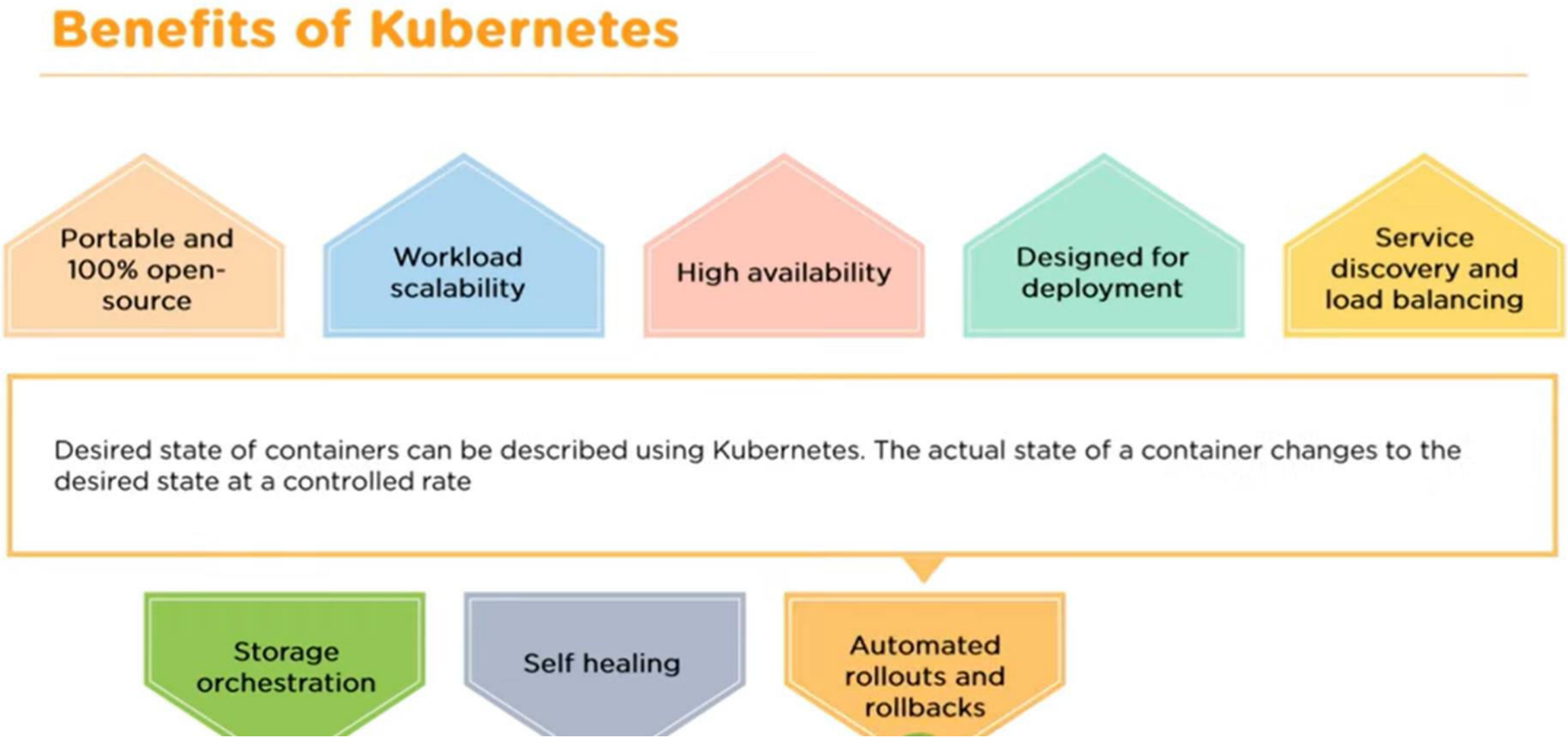
Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Kubernetes automatically mounts storage systems like local storages and public cloud providers
Storage orchestration

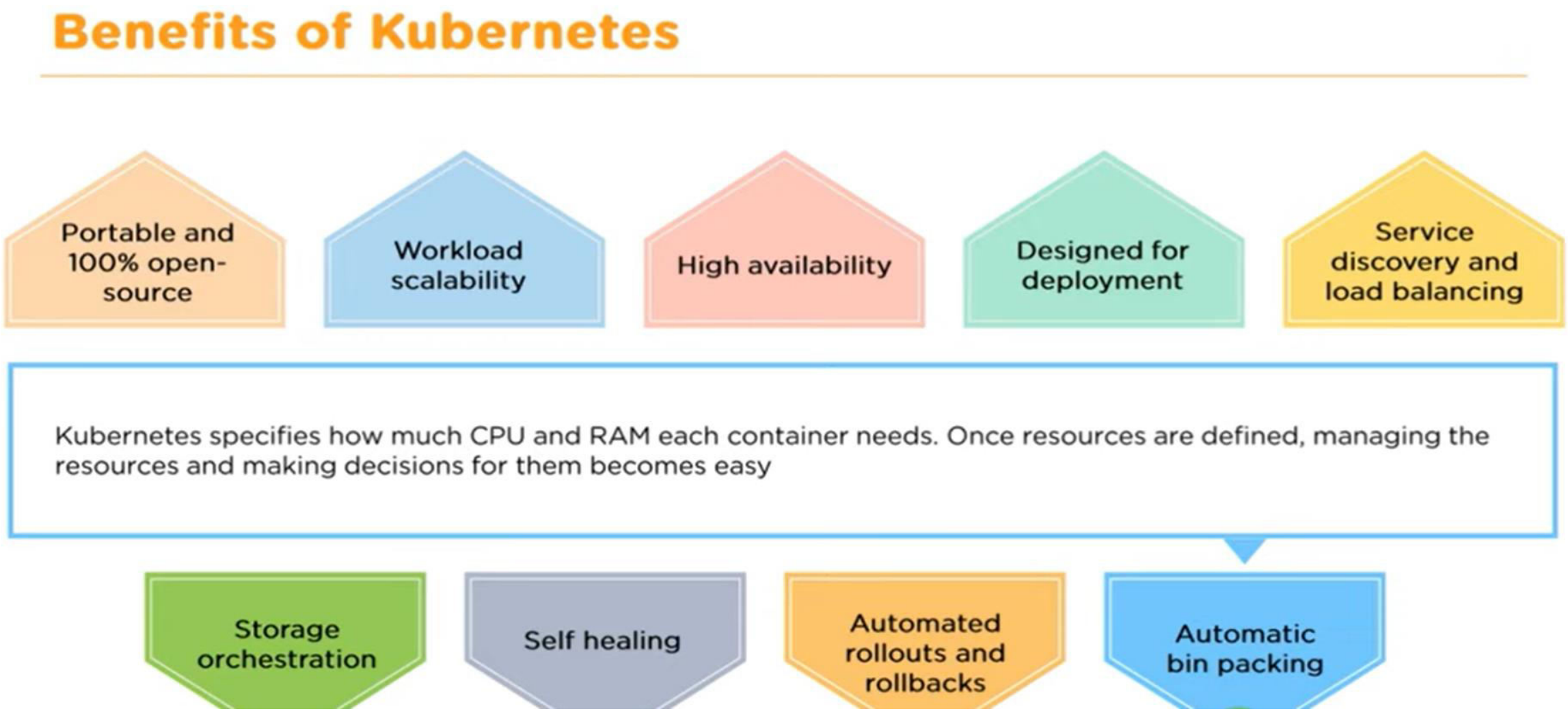
Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Kubernetes restarts containers if failed, replaces containers, kills containers that don't respond to timely checks
Storage orchestration
Self healing

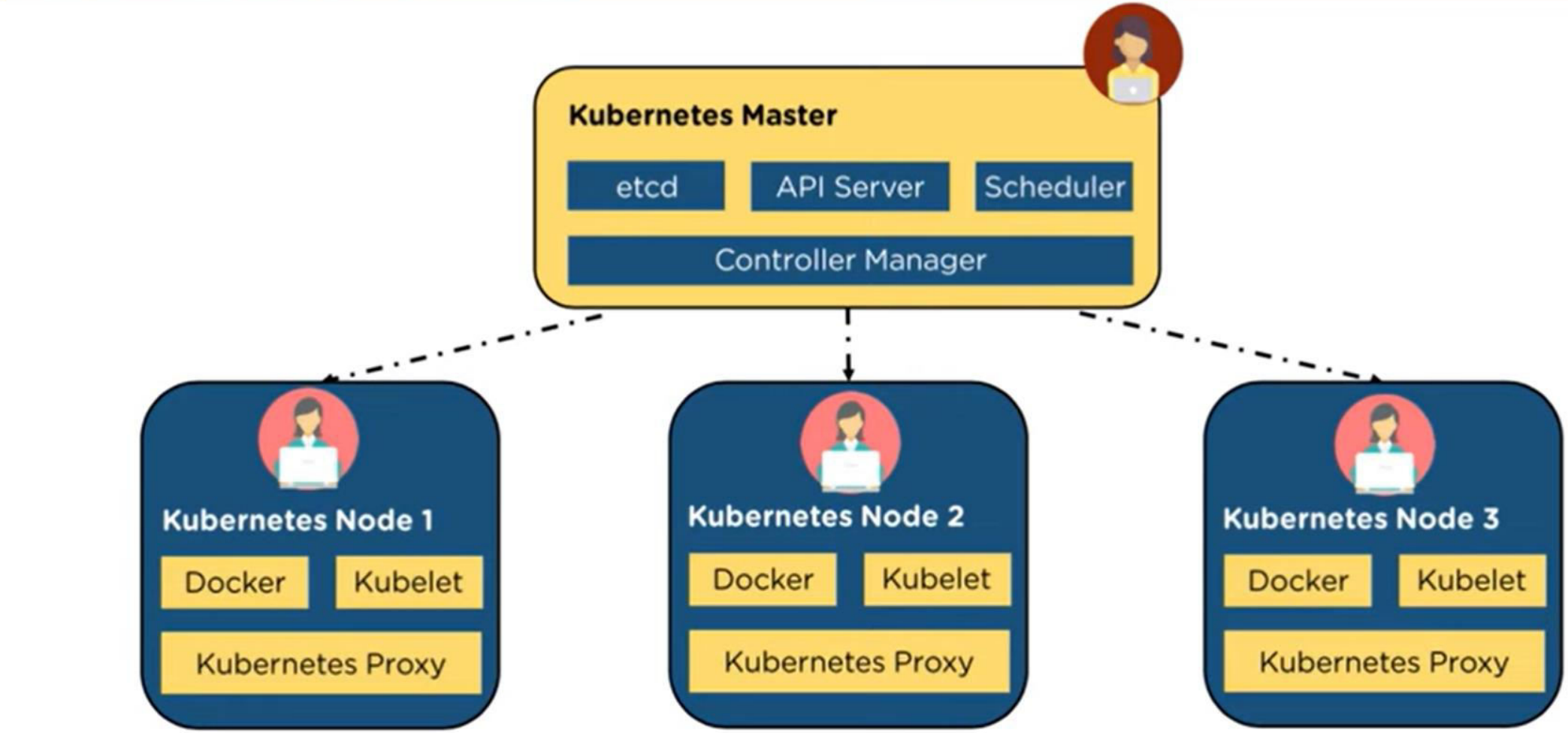
Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Desired state of containers can be described using Kubernetes. The actual state of a container changes to the desired state at a controlled rate
Storage orchestration
Self healing
Automated rollouts and rollbacks

Kubernetes

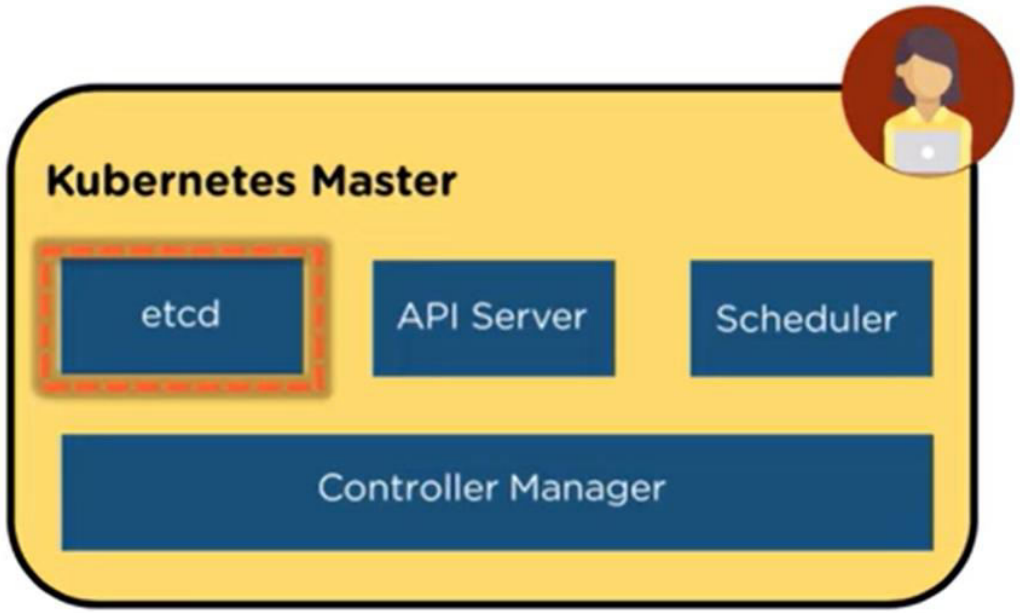

Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Kubernetes specifies how much CPU and RAM each container needs. Once resources are defined, managing the resources and making decisions for them becomes easy
Storage orchestration
Self healing
Automated rollouts and rollbacks
Automatic bin packing

Kubernetes


Kubernetes Cluster Architecture
Kubernetes Master
etcd
API Server
Scheduler
Controller Manager
Kubernetes Node 1
Docker
Kubelet
Kubernetes Proxy
Kubernetes Node 2
Docker
Kubelet
Kubernetes Proxy
Kubernetes Node 3
Docker
Kubelet
Kubernetes Proxy

Kubernetes

Kubernetes Master Components



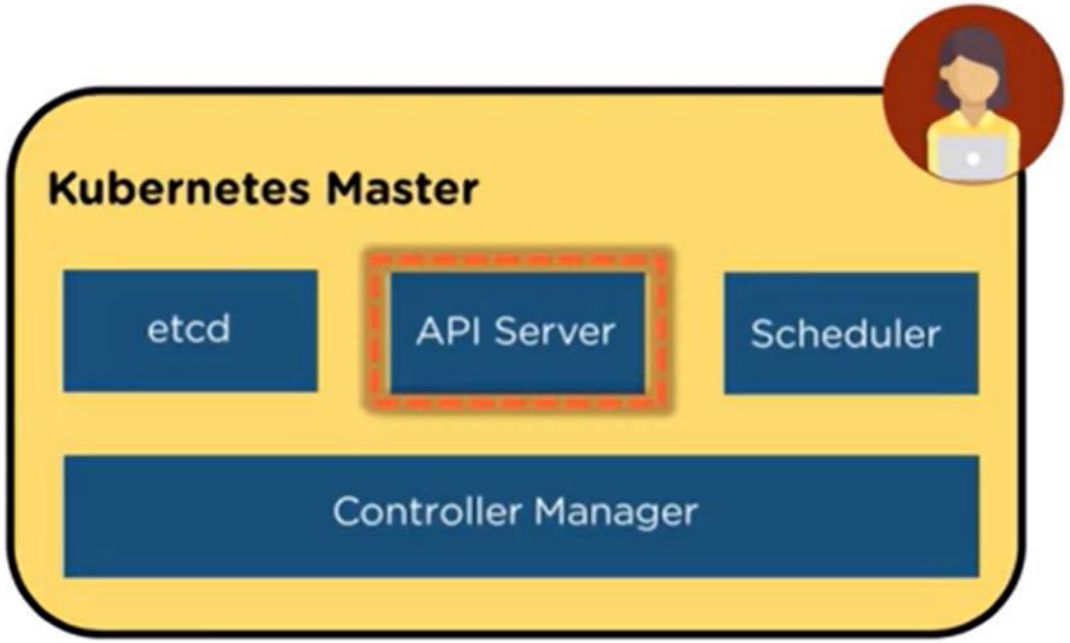
etcd

- Stores the configuration information required by nodes in the cluster
- Key-value is available and distributed across multiple nodes
- Accessible only by API Server as it may contain some sensitive information

Note - A node is a working machine in Kubernetes; it can be a VM, cloud, and so on

Kubernetes

Kubernetes Master Components

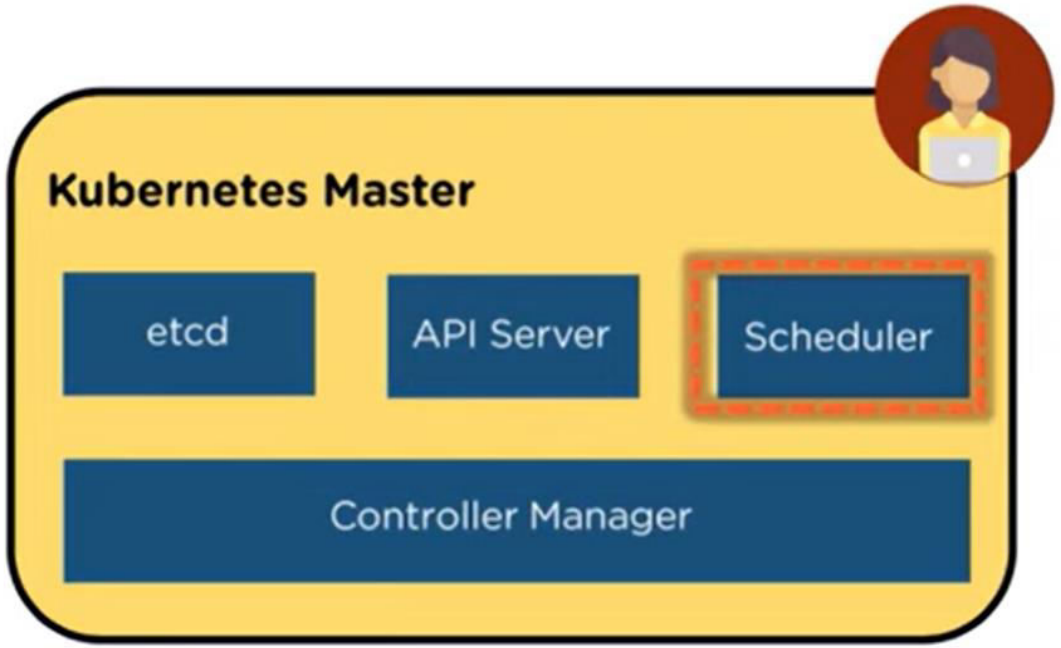


API Server

- It is a means to provide operations on the cluster
- It implements an interface so that different tools and libraries can communicate effectively
- This component interacts with the worker nodes and gives them the require information

Kubernetes

Kubernetes Master Components

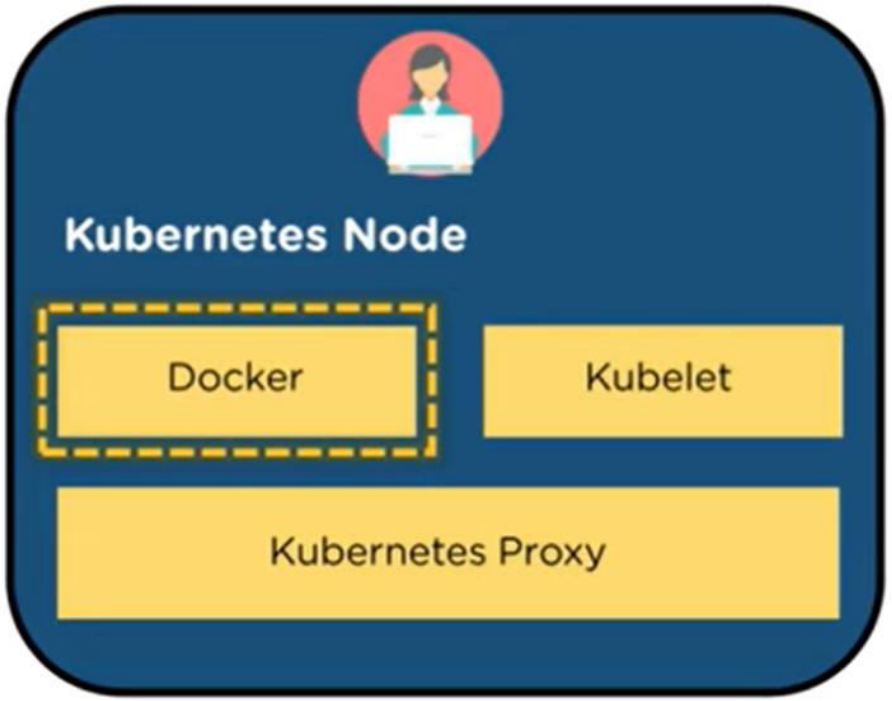


Scheduler

- It is a key component of Kubernetes Master
- It is a service that is responsible for the dispersing workload in the master node
- It also tracks the utilization of workload on cluster nodes and then places the workload onto the resources that are available

Kubernetes

Kubernetes Worker Components

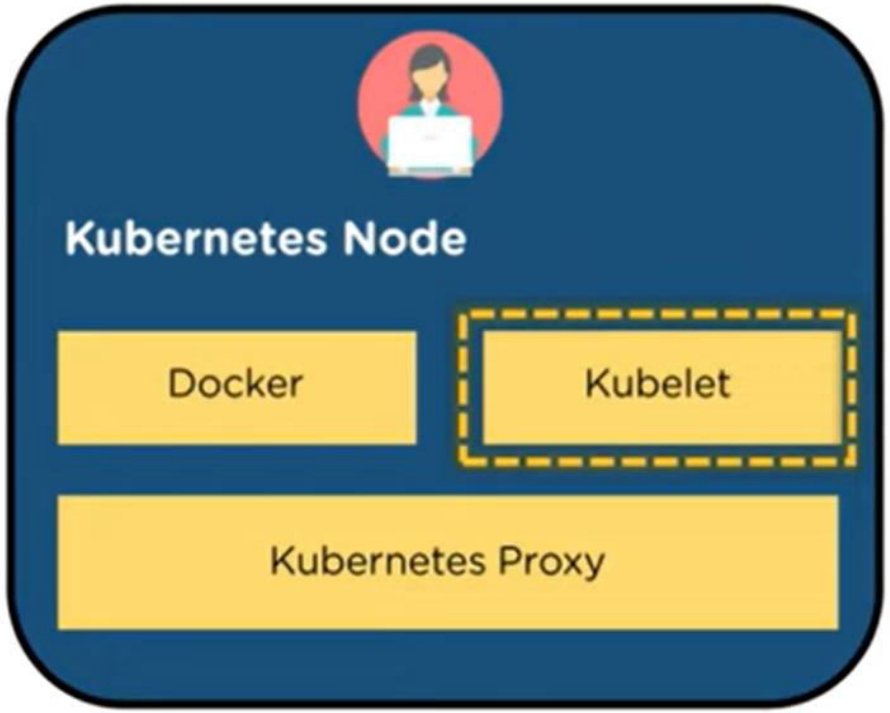


Docker

- It helps in running the applications in an isolated but lightweight operating environment
- Docker is referred to as container runtime and container runtime is used to run and manage a containers lifecycle

Kubernetes

Kubernetes Worker Components

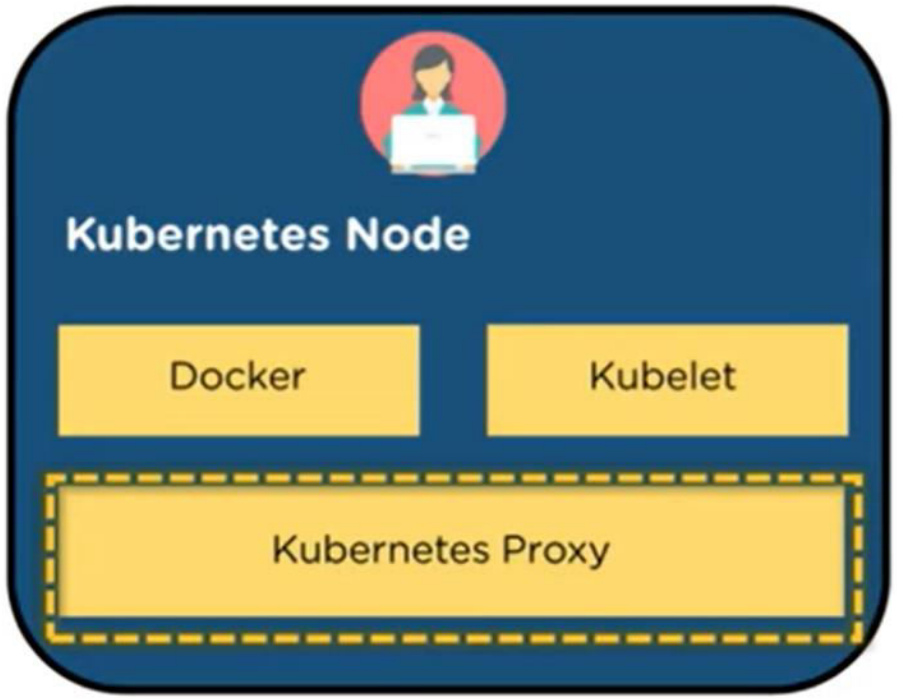


Kubelet

- It is a service responsible for conveying information to and fro to the control plane service (API Server)
- It interacts with etcd to read the configuration details. It communicates with the master node to get commands and work efficiently

Kubernetes

Kubernetes Worker Components

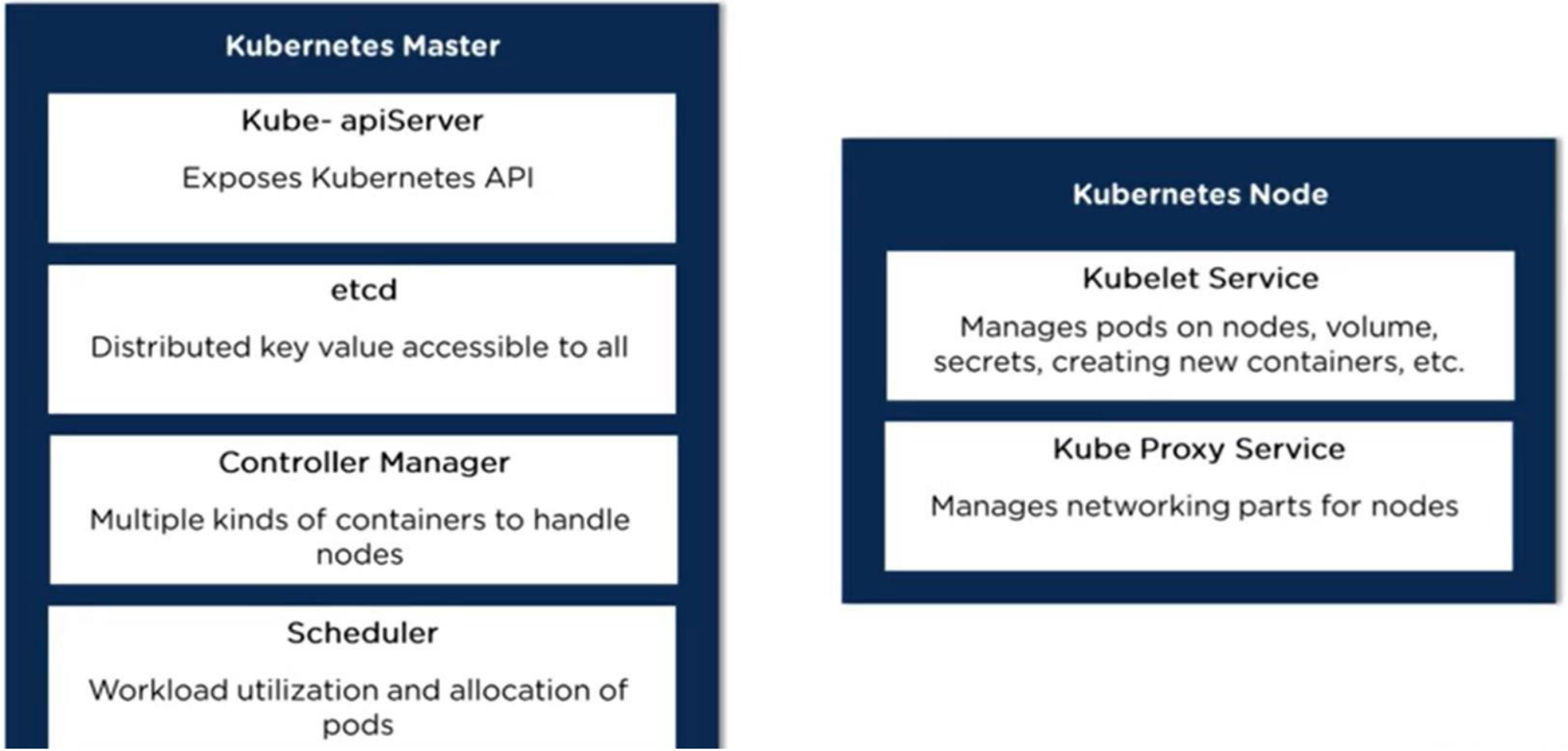


Kubernetes Proxy

- It is a proxy service that runs on every node and helps in making services available to the external host. It helps in forwarding request to assigned containers
- It performs primitive load balancing. It manages pods on nodes, volumes, secrets, creation of new containers, and health check-ups

Kubernetes

Kubernetes Cluster Structure (RECAP)



Microservices

Microservice is a service-based application development methodology. In this methodology, big applications will be divided into smallest independent service units. Microservice is the process of implementing Service-oriented Architecture (SOA) by dividing the entire application as a collection of interconnected services, where each service will serve only one business need.

Microservice is a small, loosely coupled distributed service. Microservice architecture evolved as a solution to the scalability, independently deployable, and innovation challenges with Monolithic architecture
It allows you to take a large application and decompose or break it into easily manageable small components with narrowly defined responsibilities. It is considered the building block of modern applications. Microservices can be written in a variety of programming languages, and frameworks, and each service act as a mini-application on its own. Microservice can be considered as the subset of SOA(Service Oriented Architecture).

A microservice typically implements a set of distinct features or functionality. Each microservice is a mini-application that has its own architecture and business logic. For example, some microservices expose an API that's consumed by other microservices or by the application's clients, such as third-party integrations with payment gateways and logistics.

Microservices

Deployment Patterns - There are a few patterns available for deploying microservices. These include the following:
Service Instance per Host
Service Instance per Container
Service instance per Virtual Machine.
Multiple service instances per host

Benefits of Microservices

Small Modules – The application is broken into smaller modules that are easy for developers to code and maintain.
Easier Process Adaption – By using microservices, new Technology & Process Adaption becomes easier. You can try new technologies with the newer microservices that we use.
Independent scaling – Each microservice can scale independently via X-axis scaling (cloning with more CPU or memory) and Z-axis scaling (sharding), and Y-axis scaling (functional decomposition) based on their needs.
Removes dependency – Microservice eliminates long-term commitment to any single technology stack.
Unaffected – Large applications remain largely unaffected by the failure of a single module.
DURS – Each service can be independently DURS (deployed, updated, replaced, and scaled).
Increased Security: –Microservices enable data separation. Each service has its own database, making it harder for hackers to compromise your application.
Open Standards: –APIs enable developers to build their microservices using the programming language and technology they prefer.

Microservices

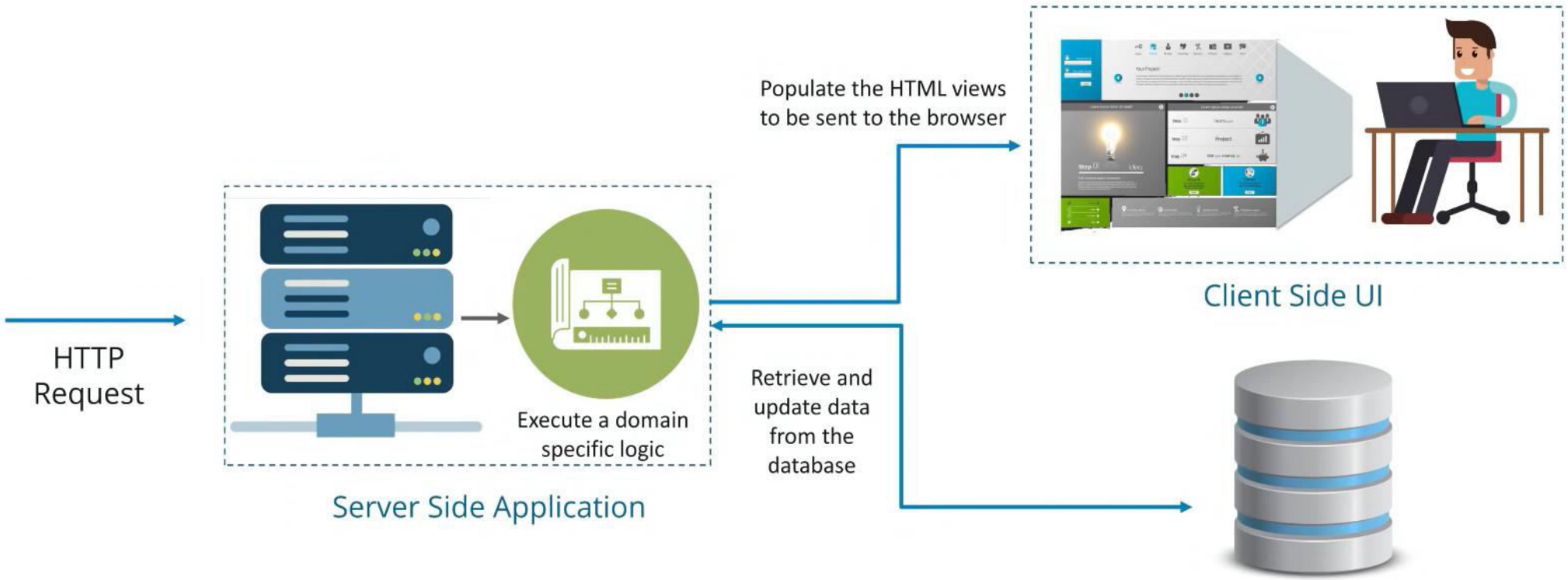
Limitations of Microservices

Configuration Management – As it becomes granular the headache comes with configuring the services and monitoring those. You need to maintain configurations for hundreds of components across environments.
Debugging – Tracking down the service failure is a painstaking job. You might need to look into multiple services across different components. Centralized Logging and Dashboards are essential to make it easy to debug problems.
Automation – Because there are a number of smaller components instead of a monolith, you need to automate everything – Builds, Deployment, Monitoring, etc.
Testing – Needs a greater effort for end-to-end testing as it needs all the dependent services to be up and running.
Coordination – While handling requests across multiple independent services there is a requirement for proper workflow management

Microservices

Before Microservices – Monolithic Architecture

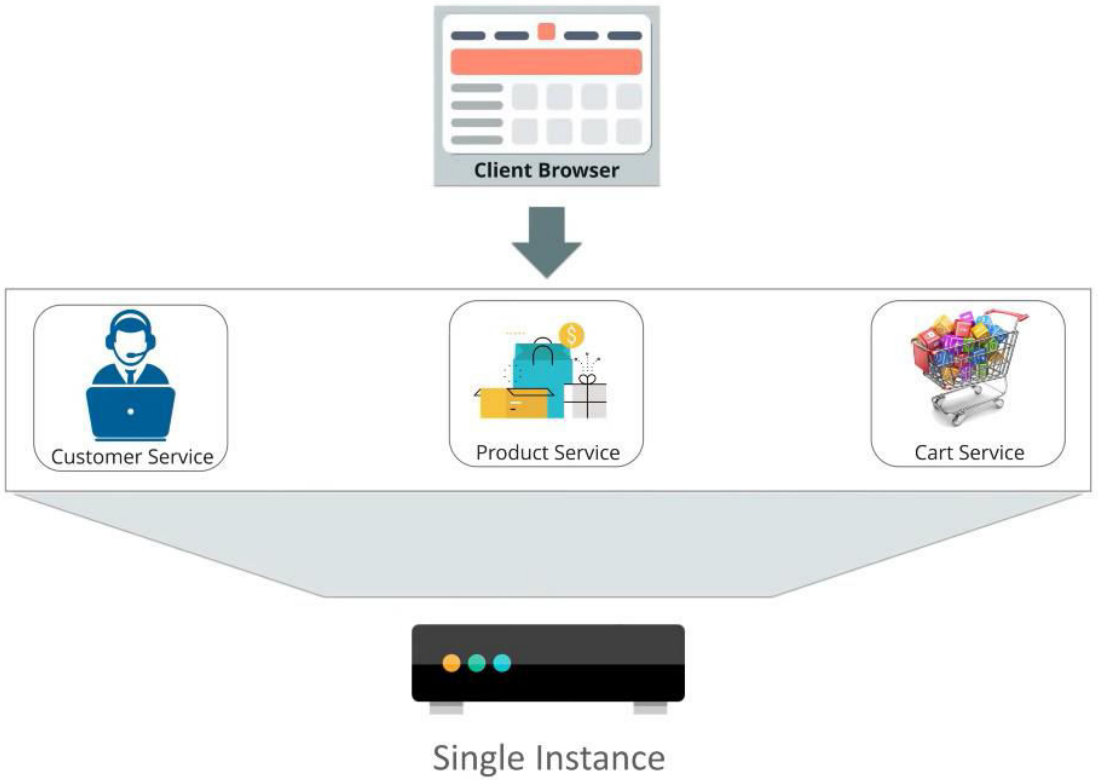
Monolithic Architecture is like a big container wherein all the software components of an application are assembled together and tightly packaged



Microservices

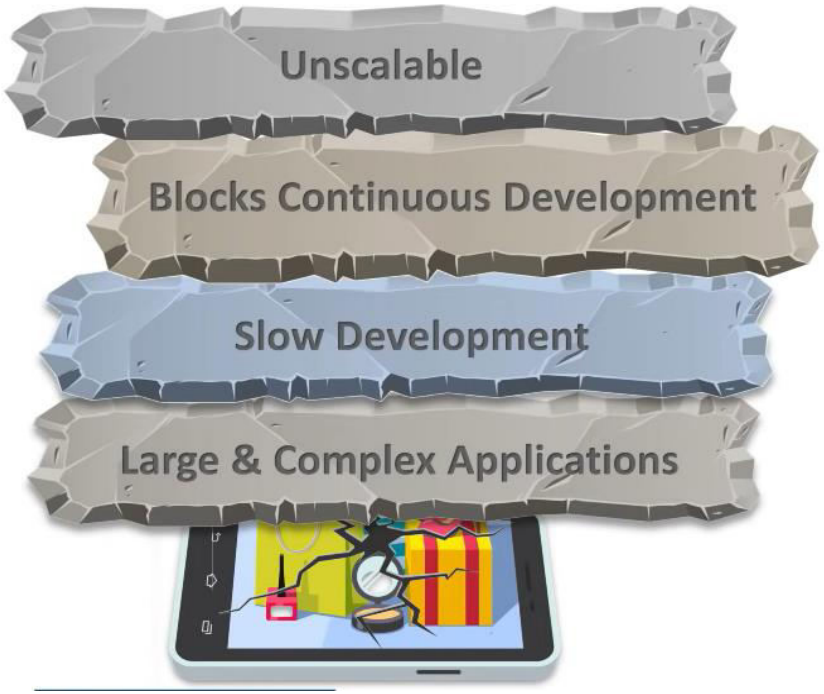
Monolithic Architecture - Example

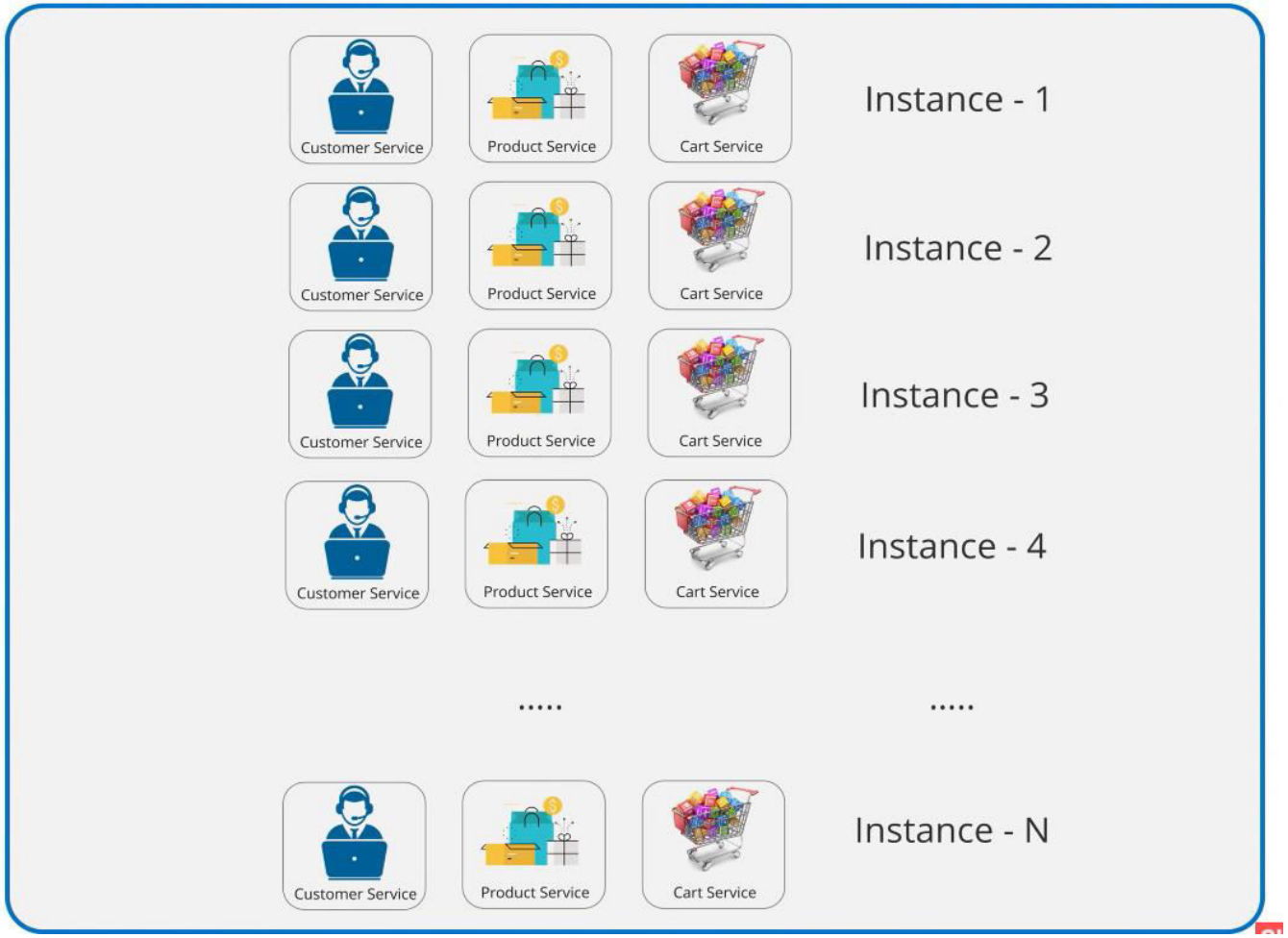
Let’s take a classic use case of an E-Commerce Application



Microservices

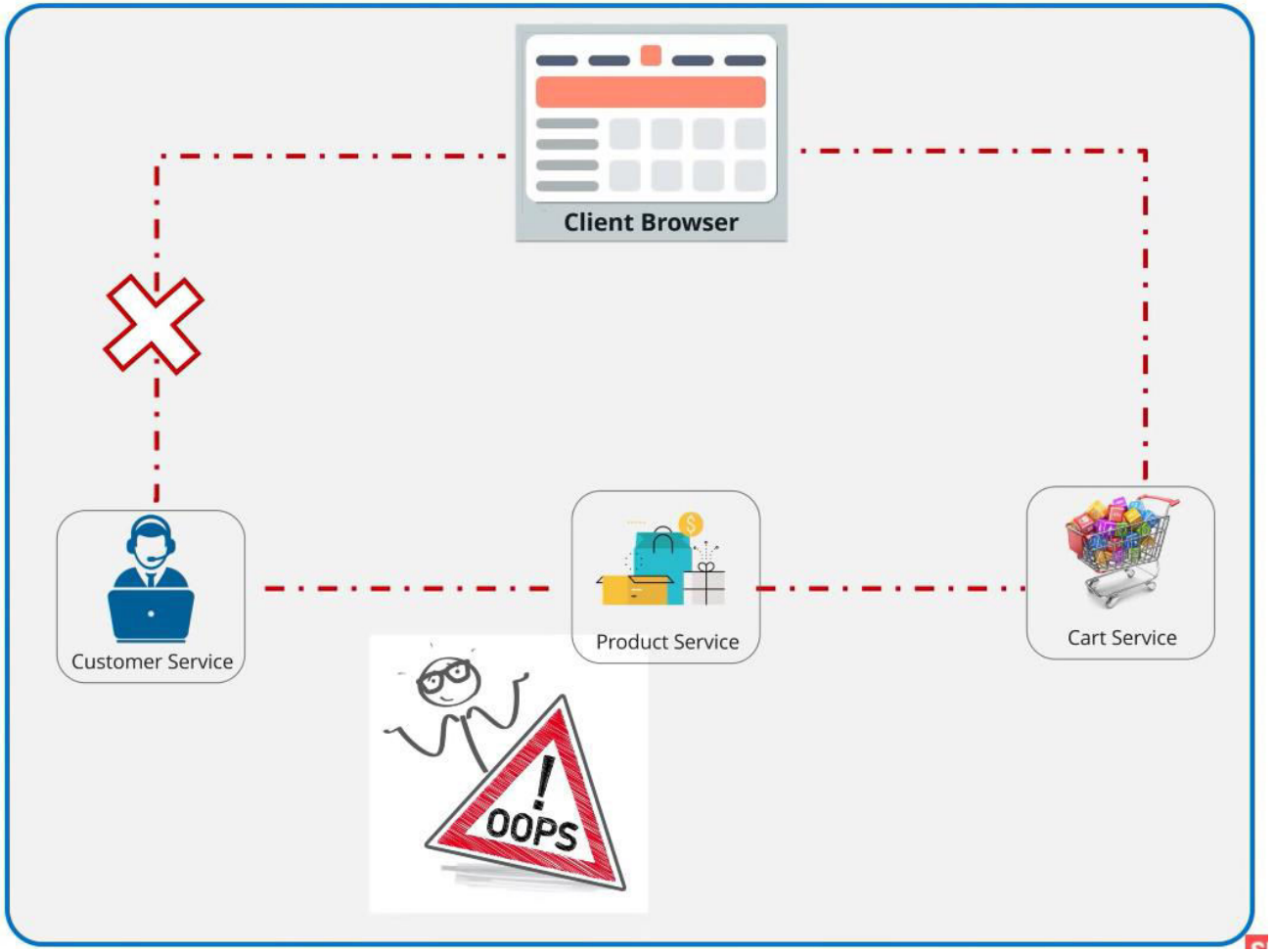
Monolithic Architecture - Challenges





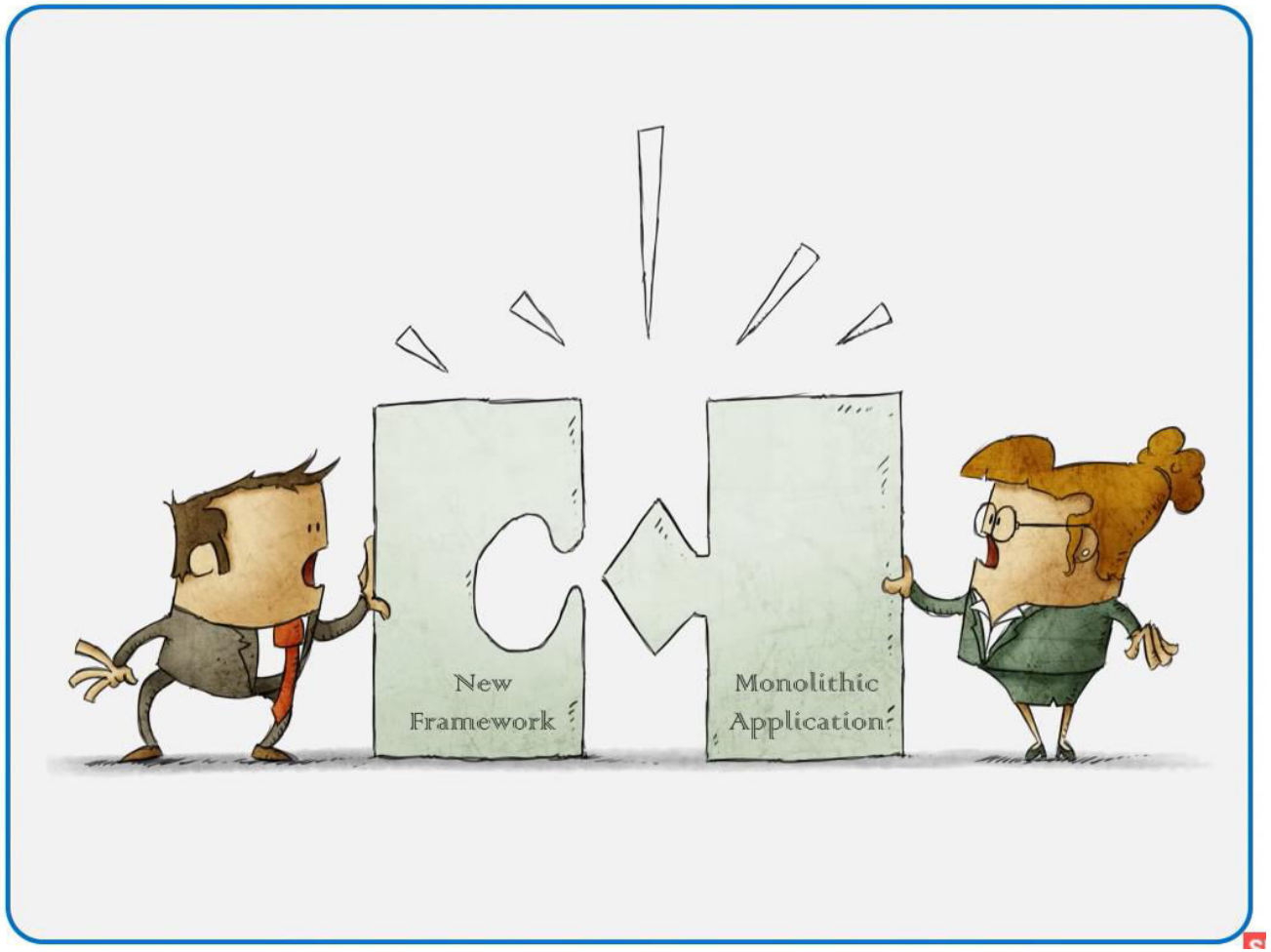
Microservices

Monolithic Architecture - Challenges



Microservices

Monolithic Architecture - Challenges



Microservices

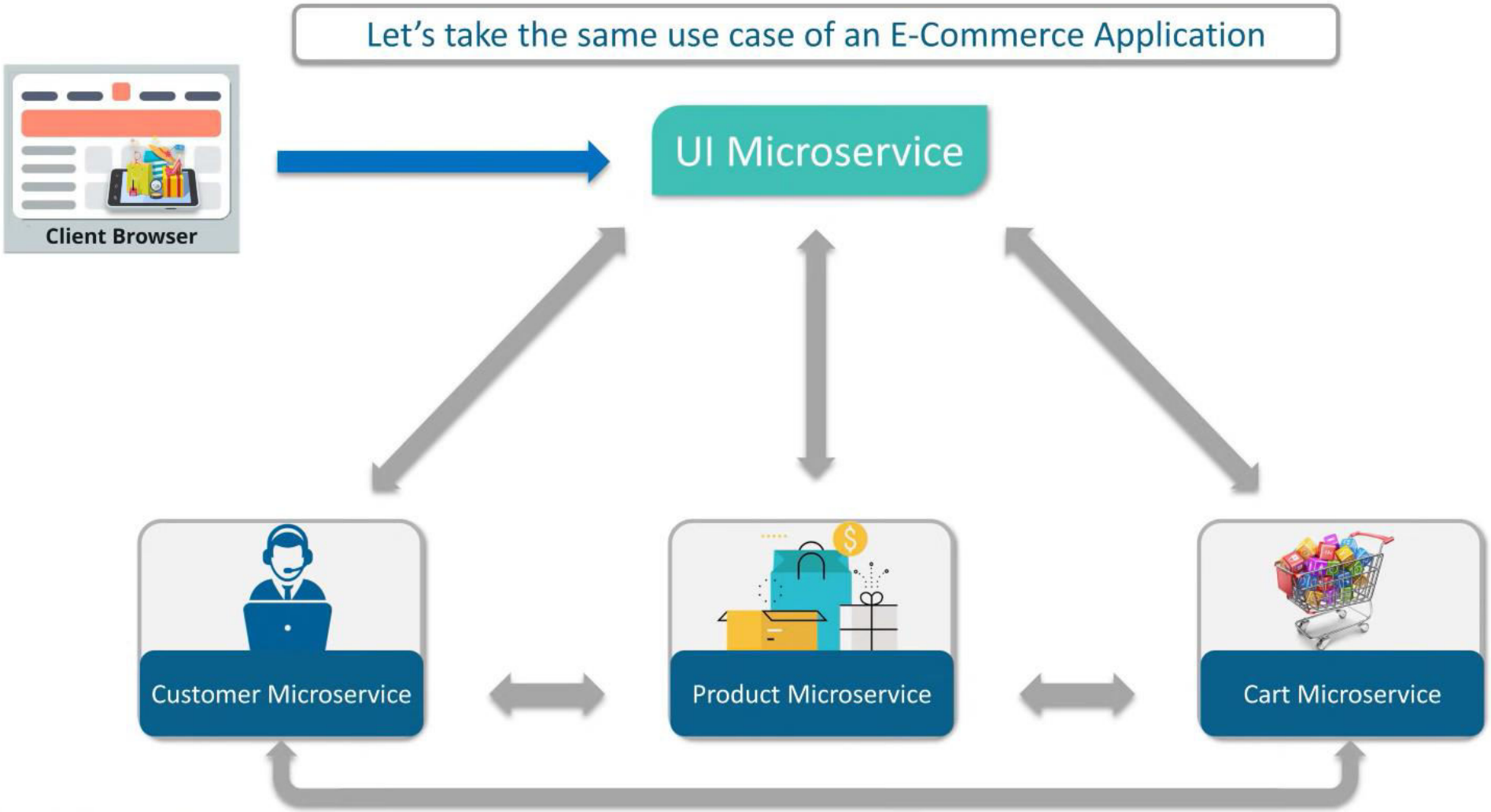
What Is Microservice Architecture?

Microservices, aka *Microservice Architecture*, is an **architectural style** that structures an application as a **collection of small autonomous services**, modelled around a **Business Domain**

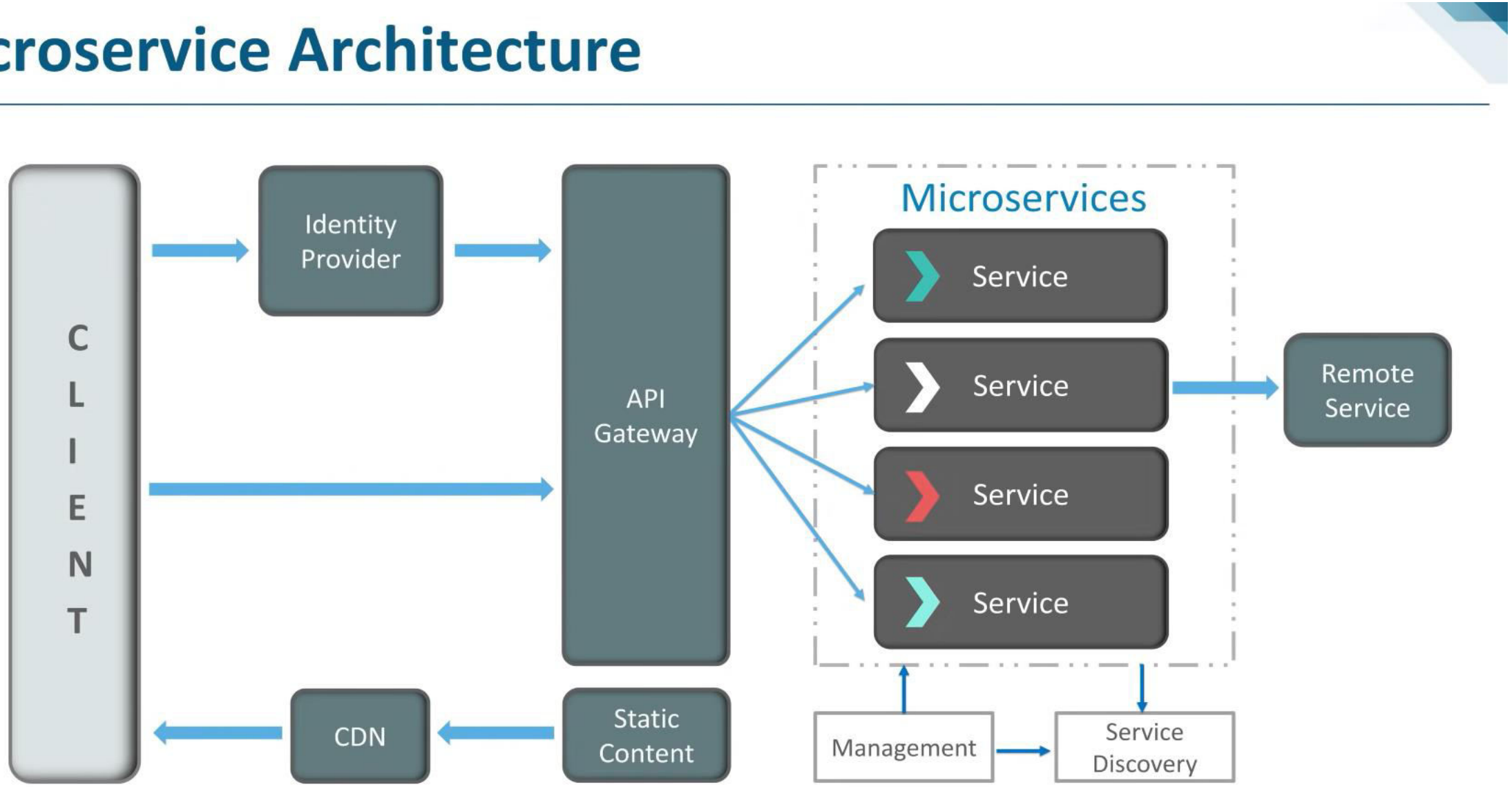
In Microservice Architecture, each service is **self-contained** and implements a **single Business capability**

Microservices

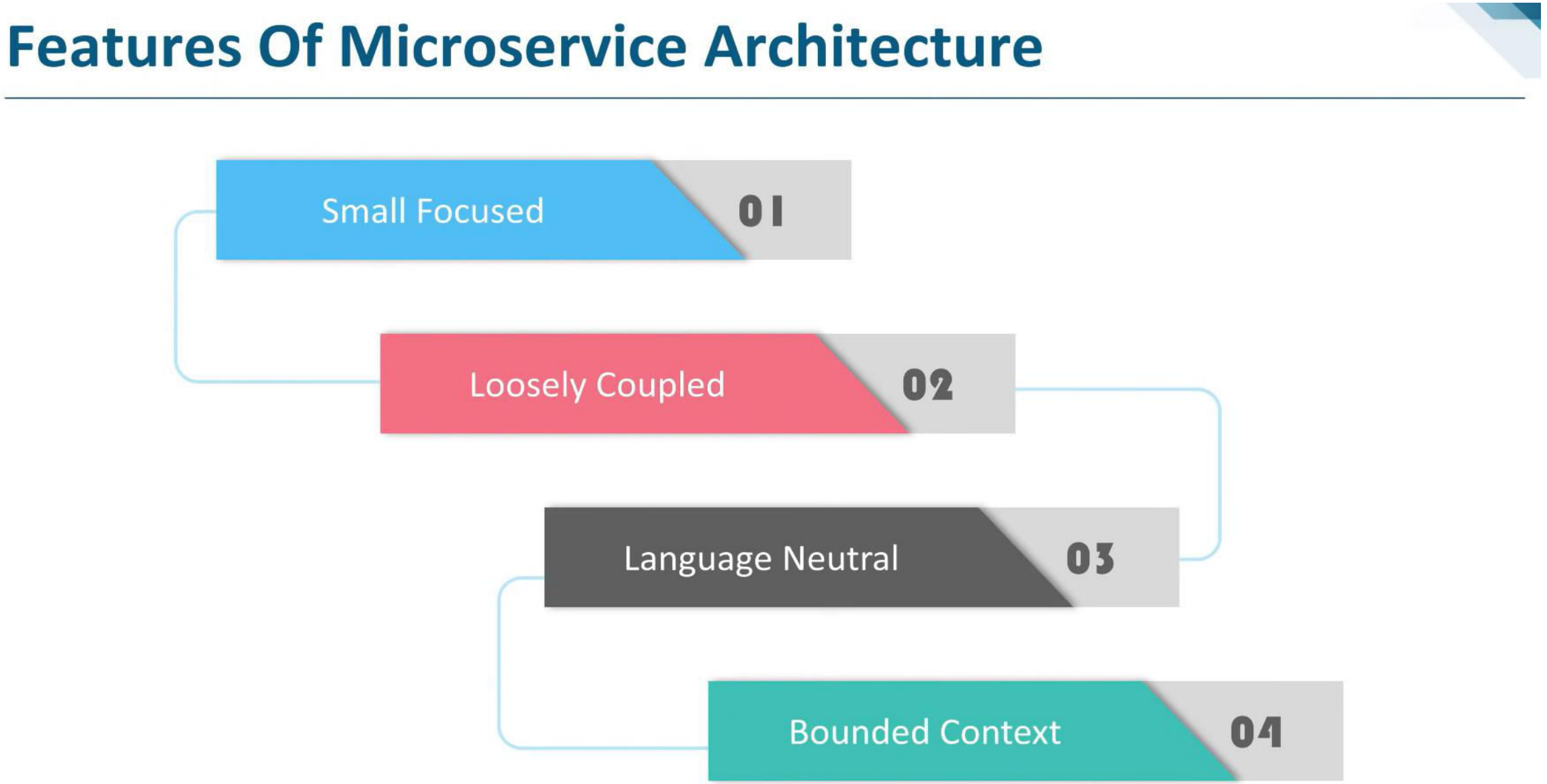
Microservice Architecture - Example



Microservices

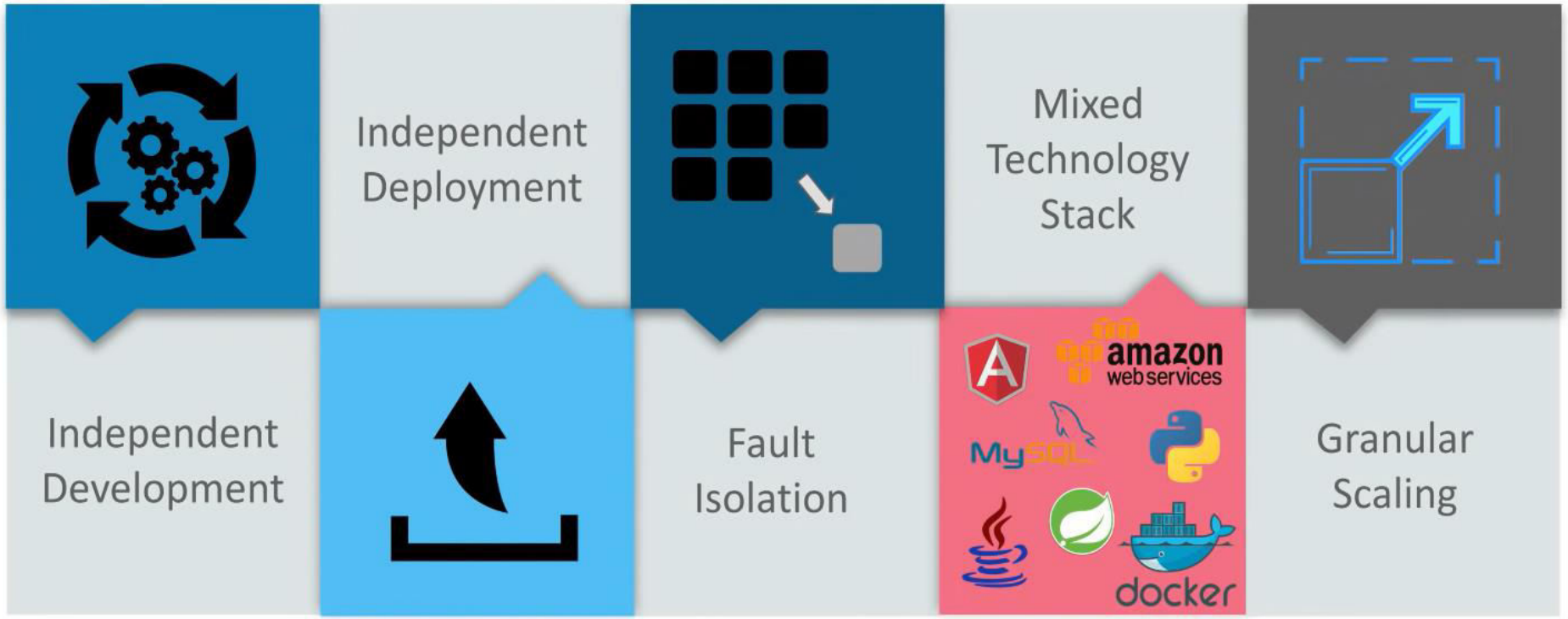


Microservices



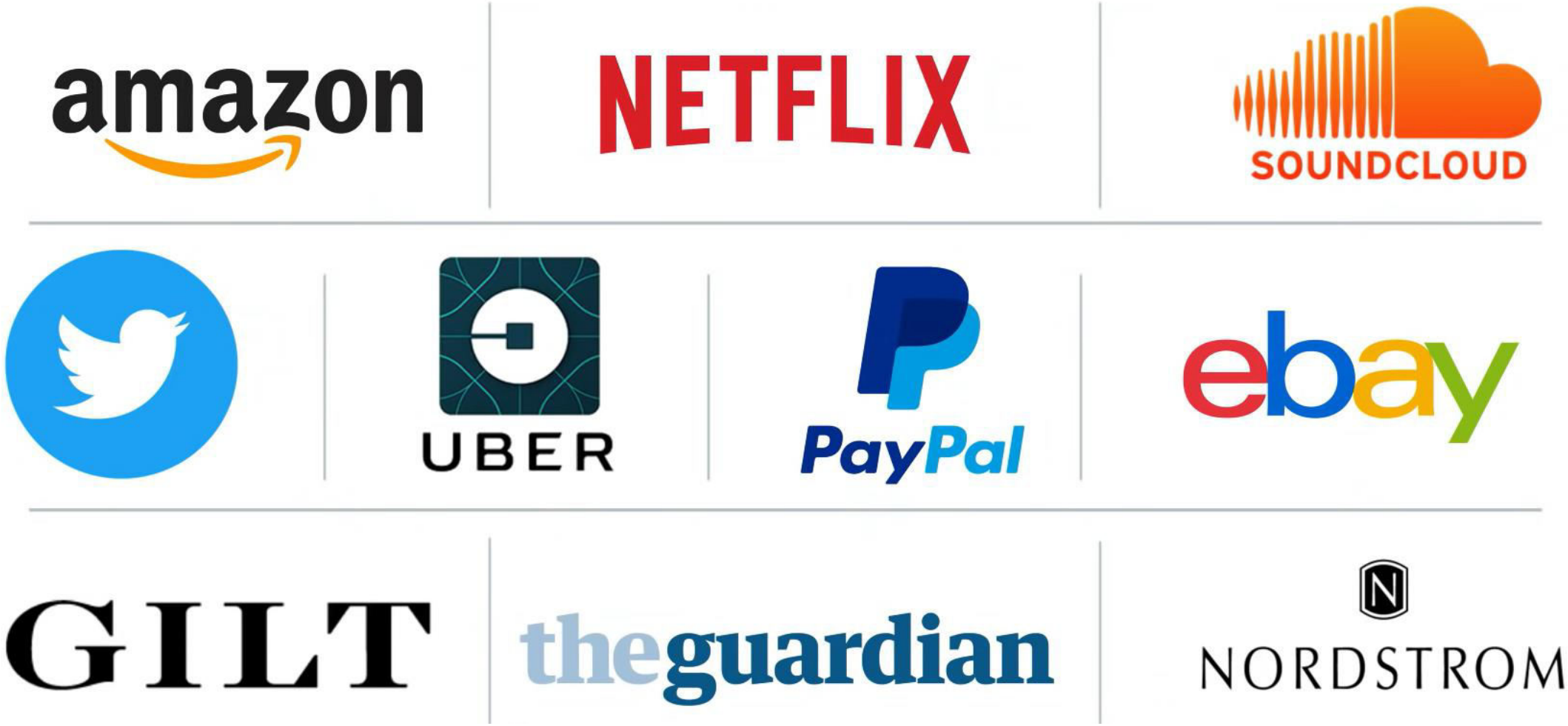
Microservices

Advantages Of Microservice Architecture

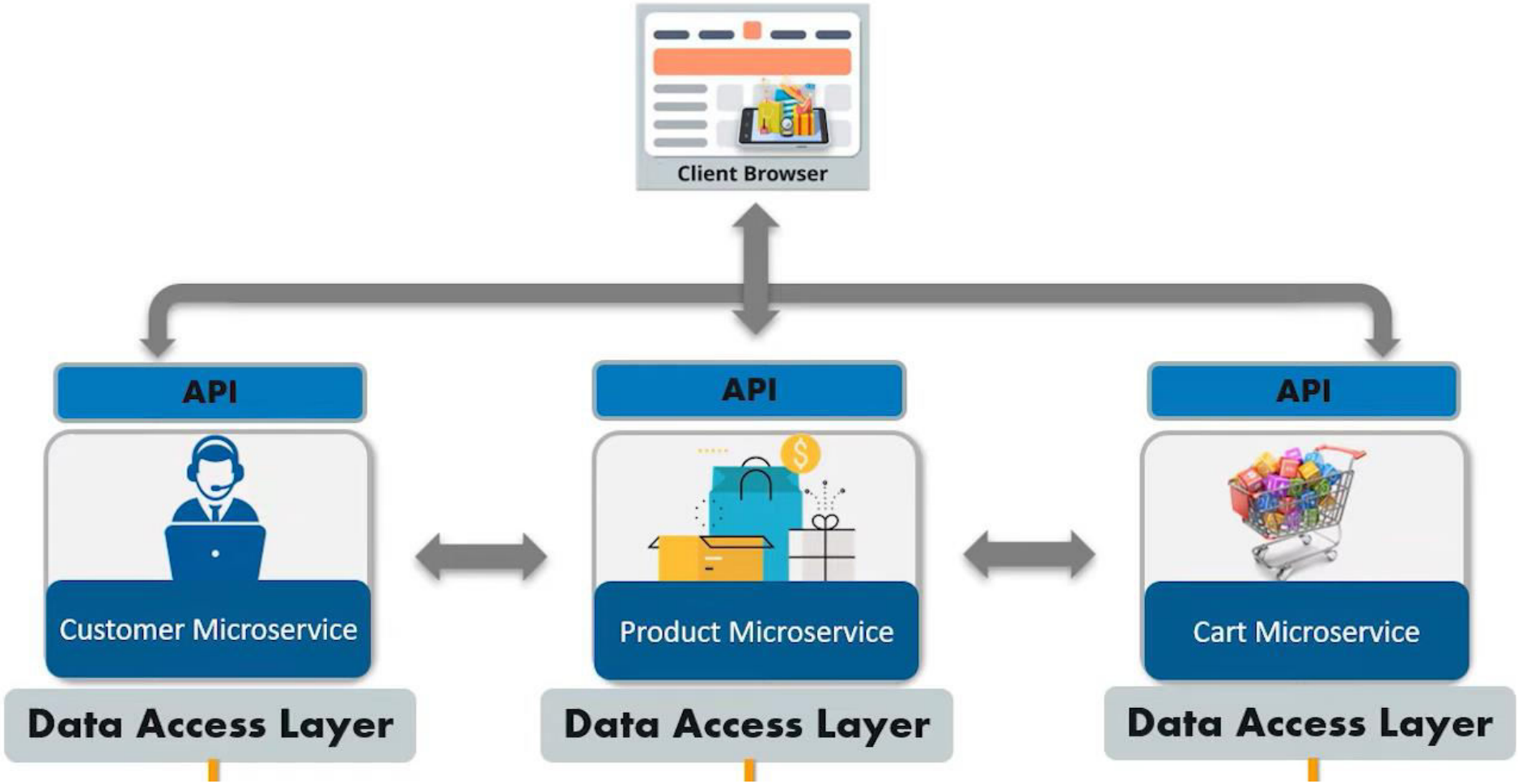


Microservices

Companies Using Microservices



Microservices



Microservices

MICROSERVICES

An architectural style through which, you can build applications in the form of small autonomous services.

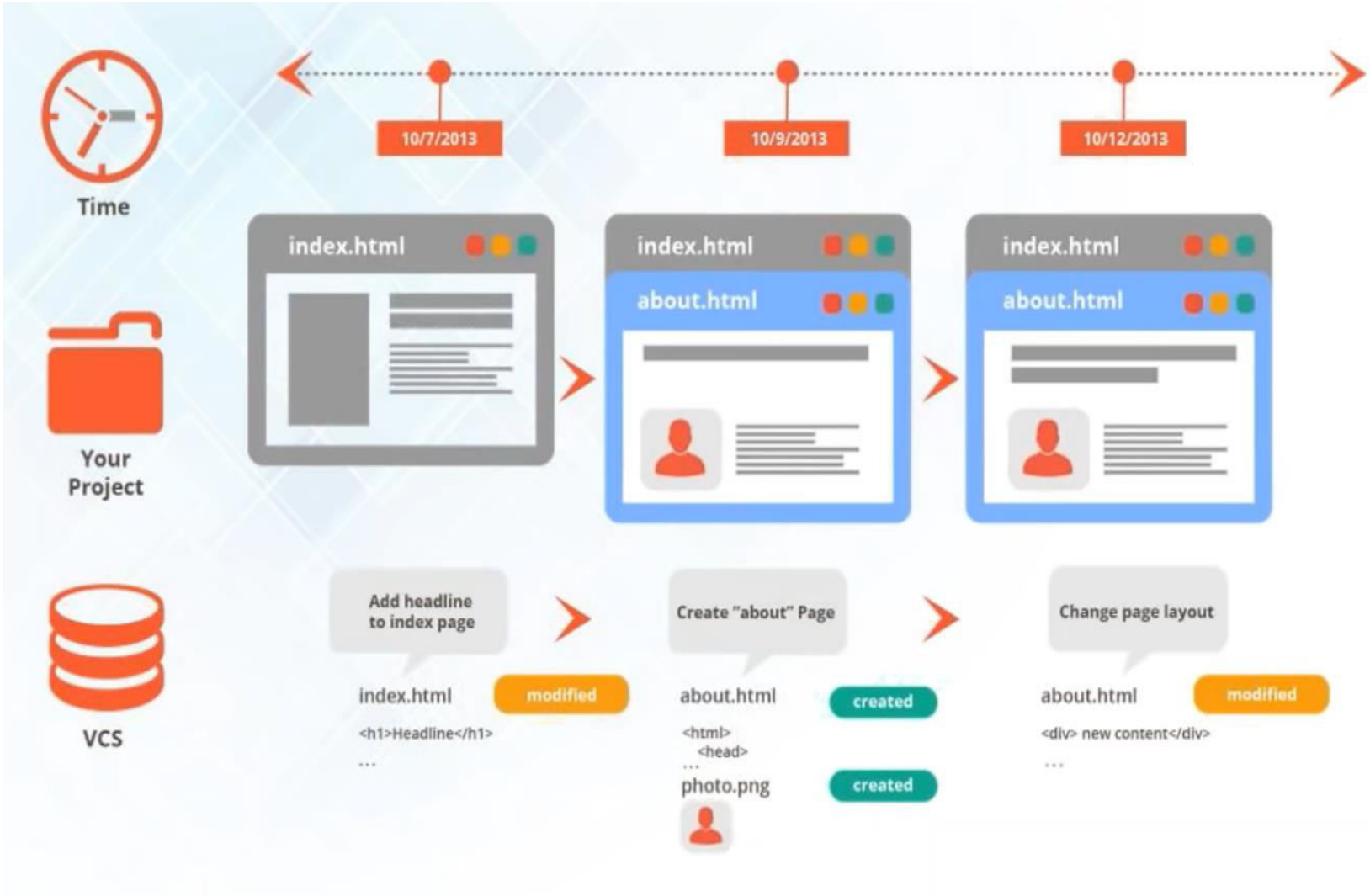
A set of procedures and functions which allow the consumer to use the underlying service of an application.

API

Git & GitHub

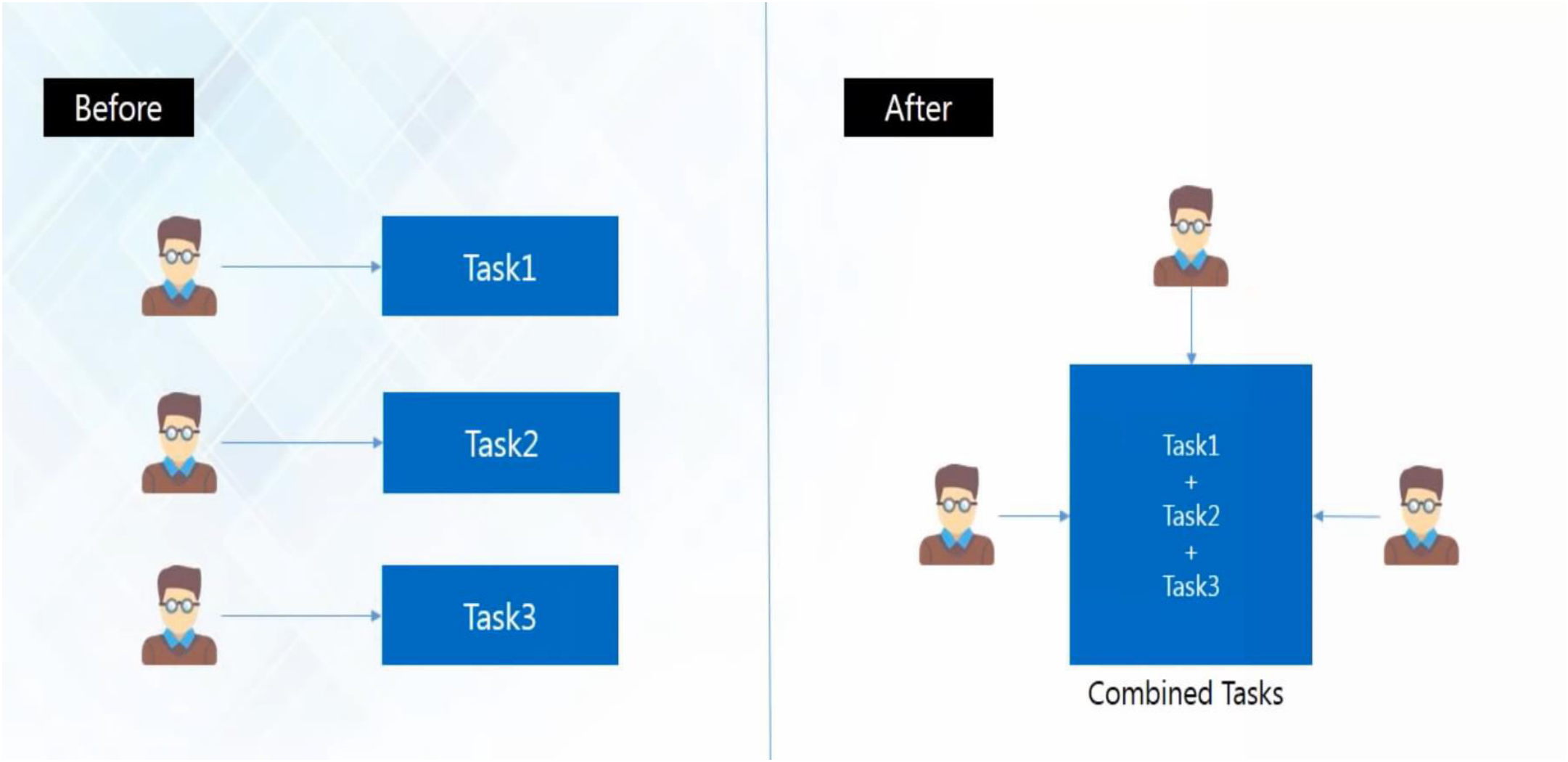
- Version Control – What & Why?
- Version Control Tools
- GitHub & Git
- Case Study: Dominion Enterprises
- Git Features
- Git Operations & Commands

Git & GitHub



- Version control is the management of changes to documents, computer programs, large web sites, and other collections of information.
- These changes are usually termed as "versions".

Git & GitHub

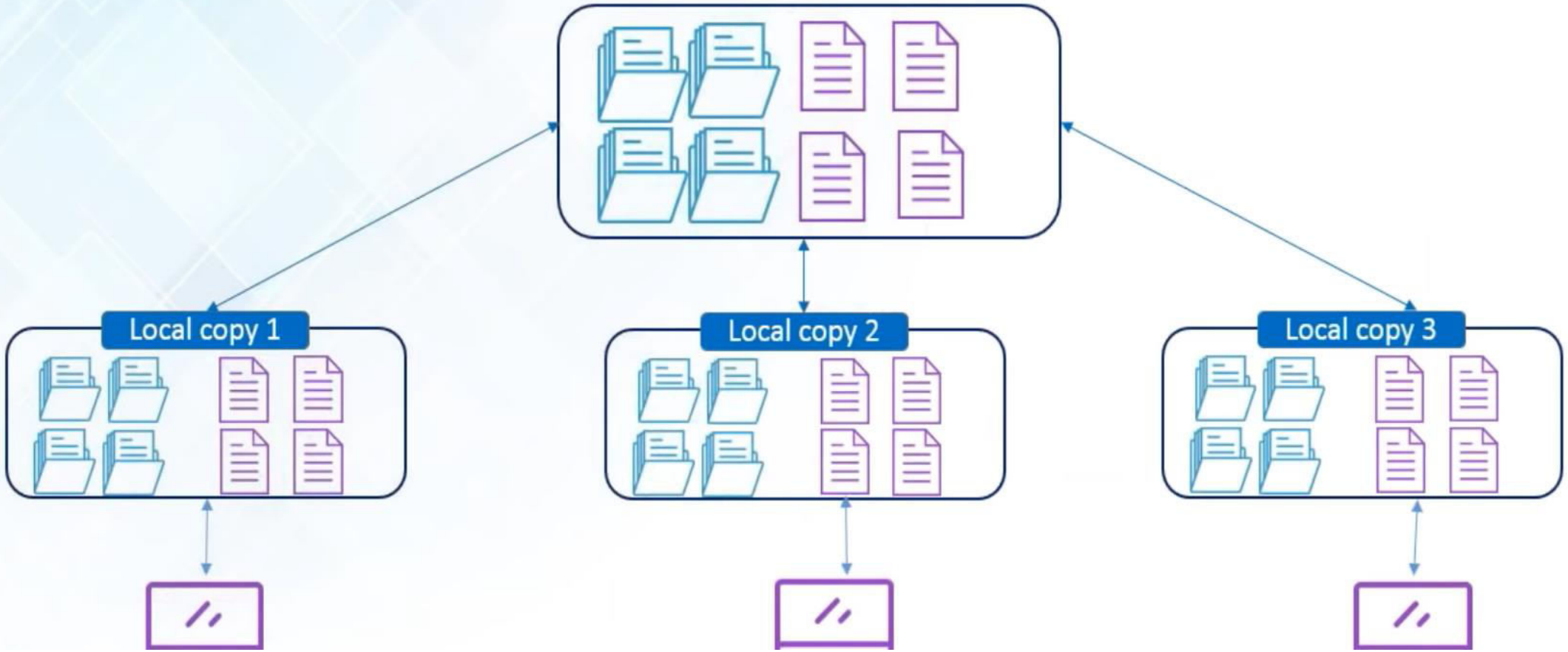

Before
Task1
Task2
Task3
After
Task1
+
Task2
+
Task3
Combined Tasks

Git & GitHub

- Snapshots of all versions are properly documented and stored.
- Versions are also named accurately.

Git & GitHub

In any case if your central server crashes, a backup is always available in your local servers.



Git & GitHub

When you change version -

- VCS provides you with proper description
- What exactly was changed
- When it was changed

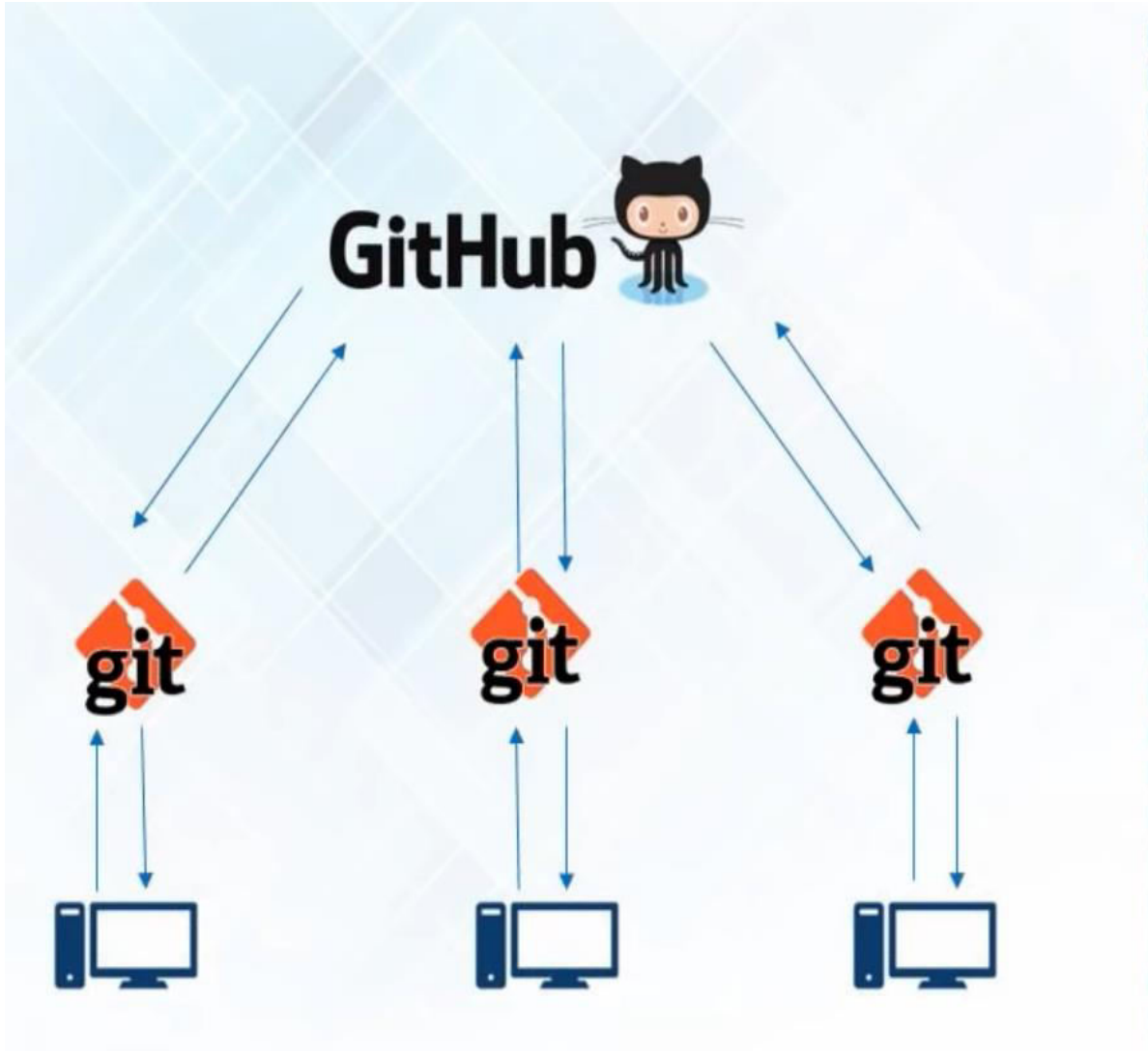
And hence, you can analyze how your project evolved between versions.

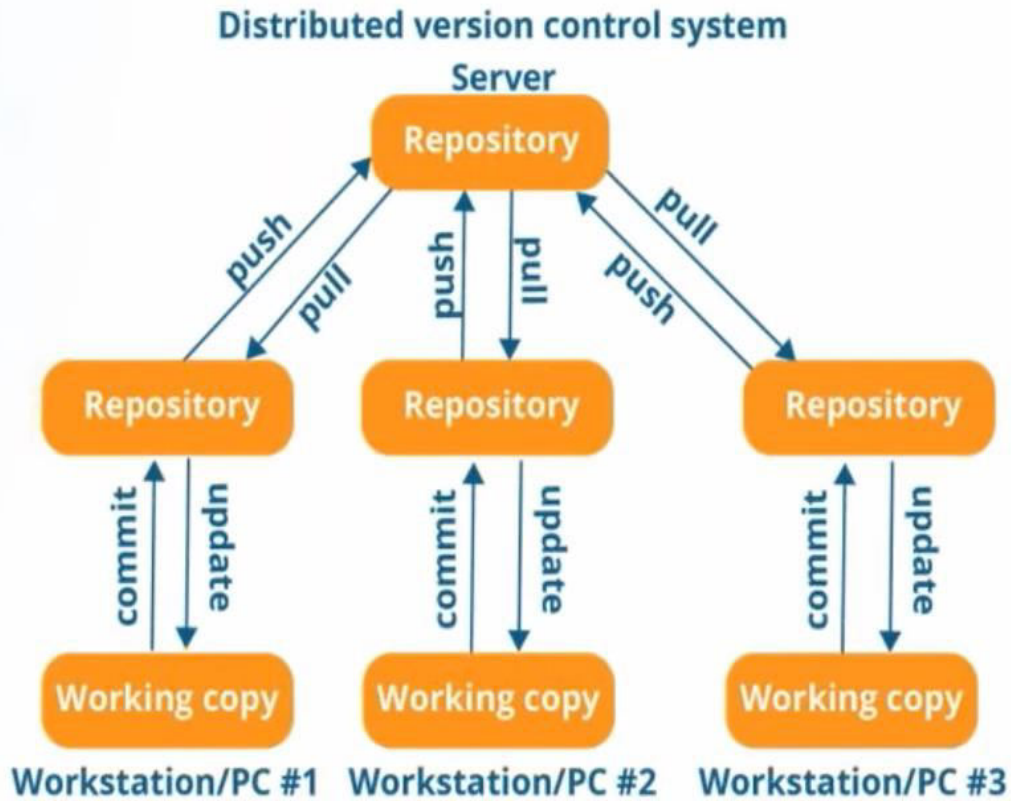
Git & GitHub

Git & GitHub


100
75
50
25
Average
Nov 6, 2011
Jul 21, 2013
Apr 5, 2015
Jan 16,2017
Git
SVN
Mercurial
CVS

Git & GitHub

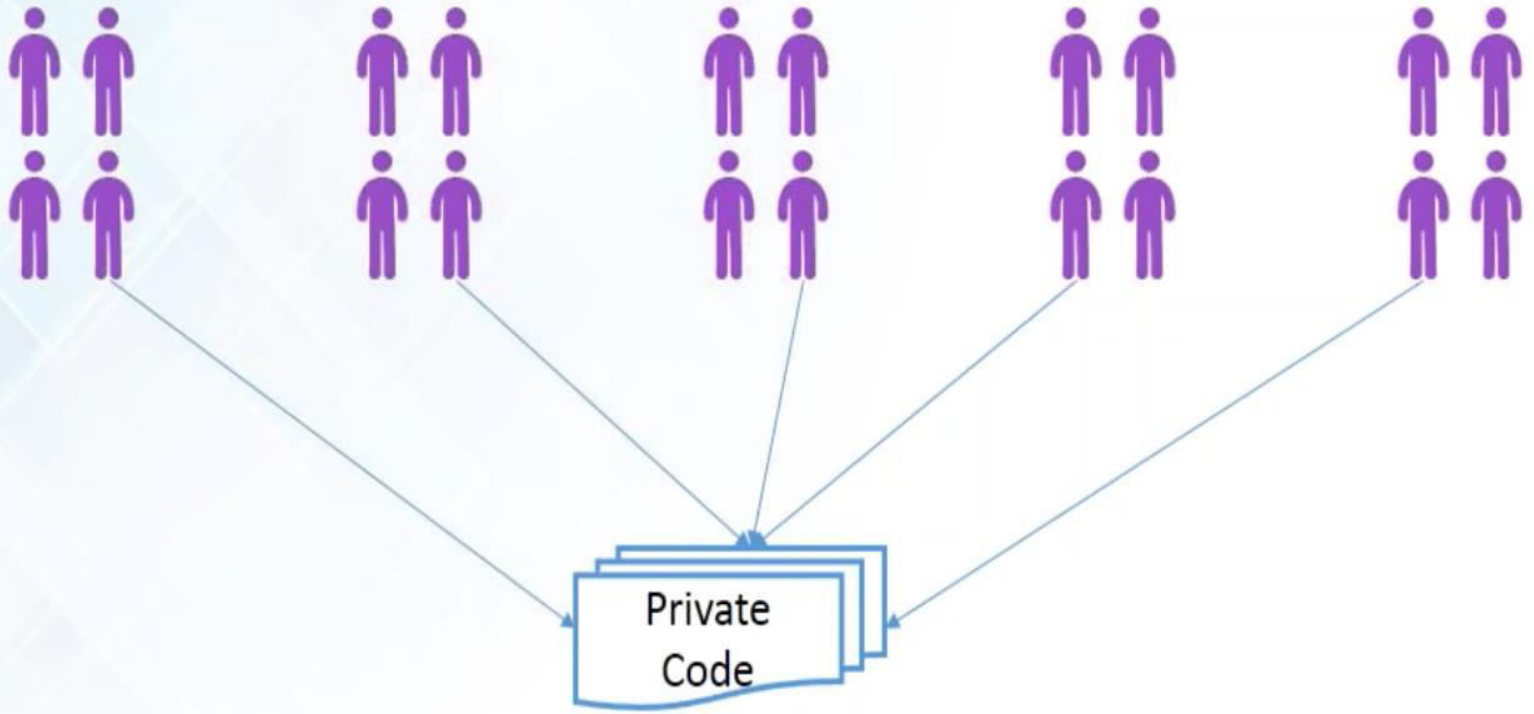

GitHub
git
git
git


Distributed version control system
Server
Repository
push
pull
push
pull
pull
push
Repository
Repository
Repository
commit
update
commit
update
commit
update
Working copy
Working copy
Working copy
Workstation/PC #1
Workstation/PC #2
Workstation/PC #3

Git & GitHub

Problem Statement:

Each team has its own goals, projects, and budgets and they also have Unique needs and workflows

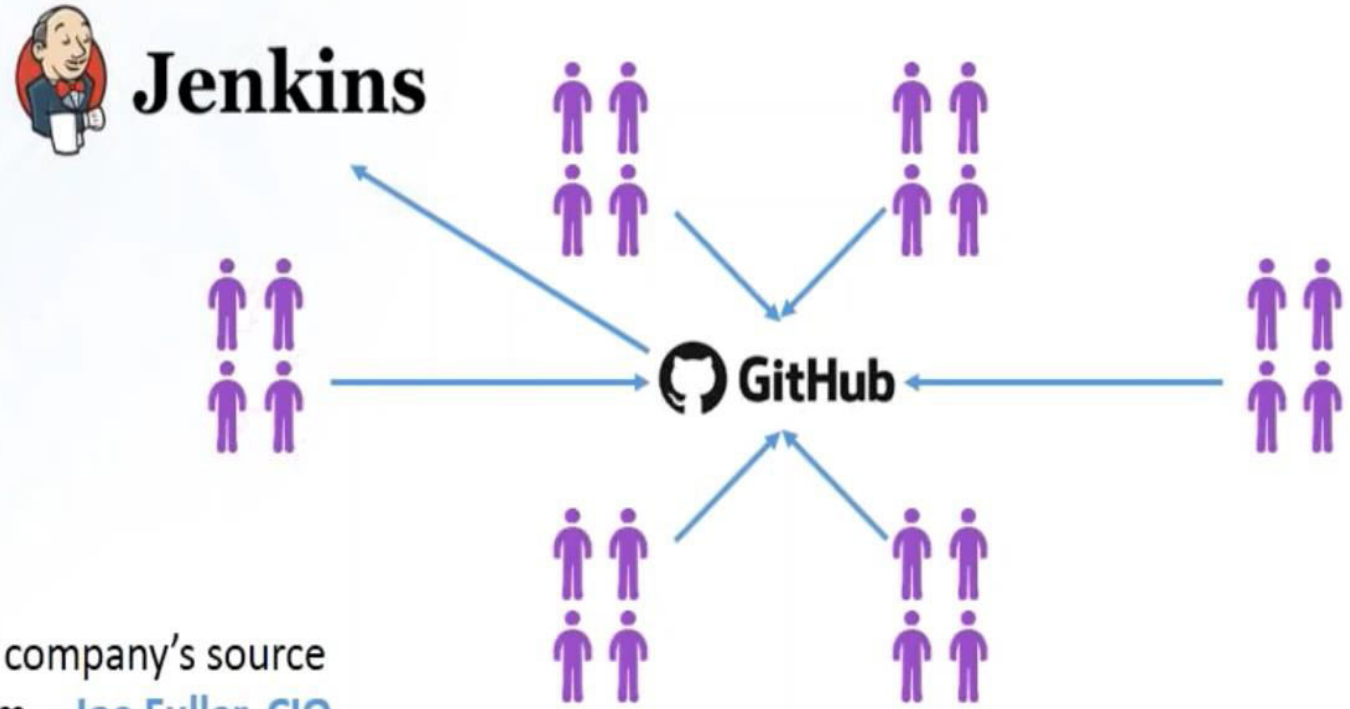


They wanted to make private code “publicly” to make their work more transparent across the company

Git & GitHub

Reason for using GitHub as the solution:

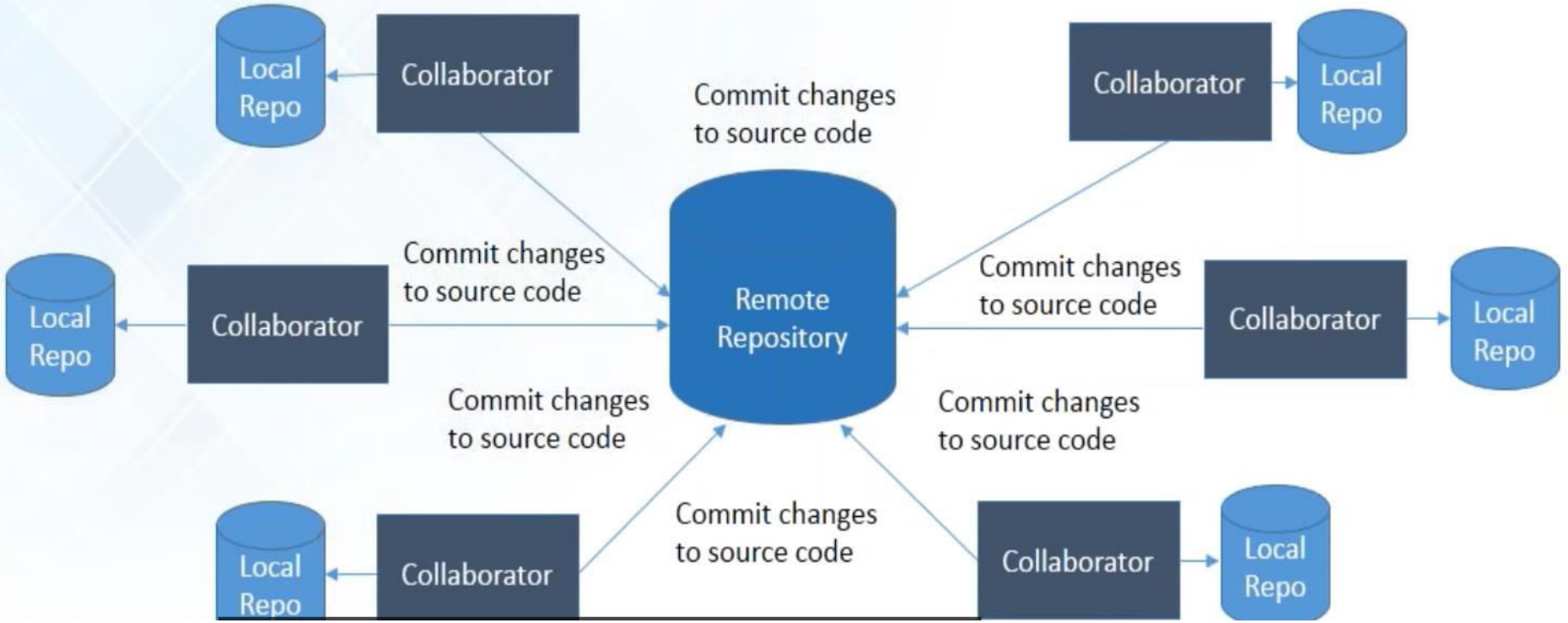
- ❑ They noticed that few of the teams were already using GitHub. Adopting a familiar platform has also made onboarding easier for new employees.
- ❑ Having all of their code in one place makes it easier for them to collaborate on projects.



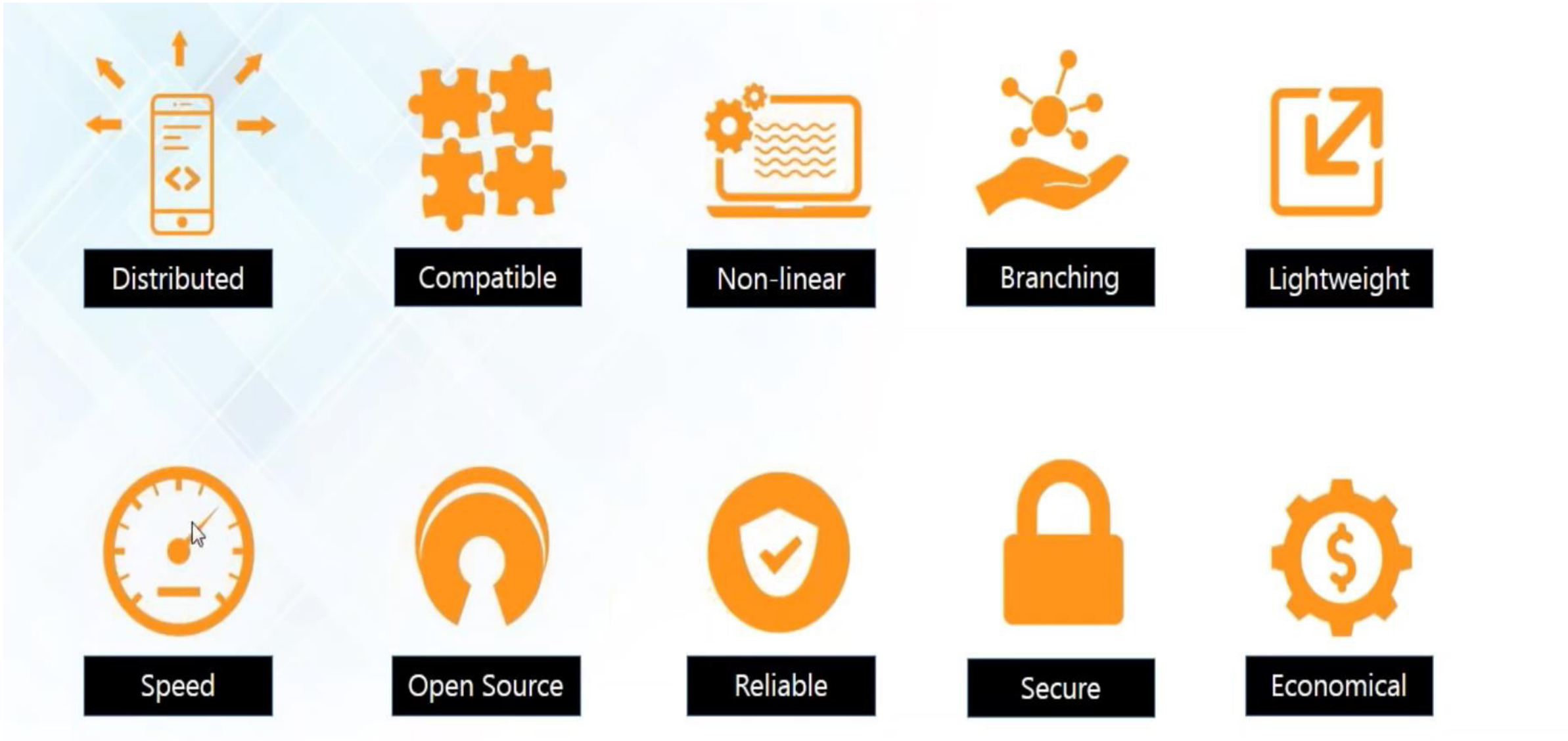
GitHub Enterprise has allowed us to store our company's source code in a central, corporately controlled system. - Joe Fuller, CIO

Git & GitHub

Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software.



Git & GitHub


Distributed
Compatible
Non-linear
Branching
Lightweight
Speed
Open Source
Reliable
Secure
Economical

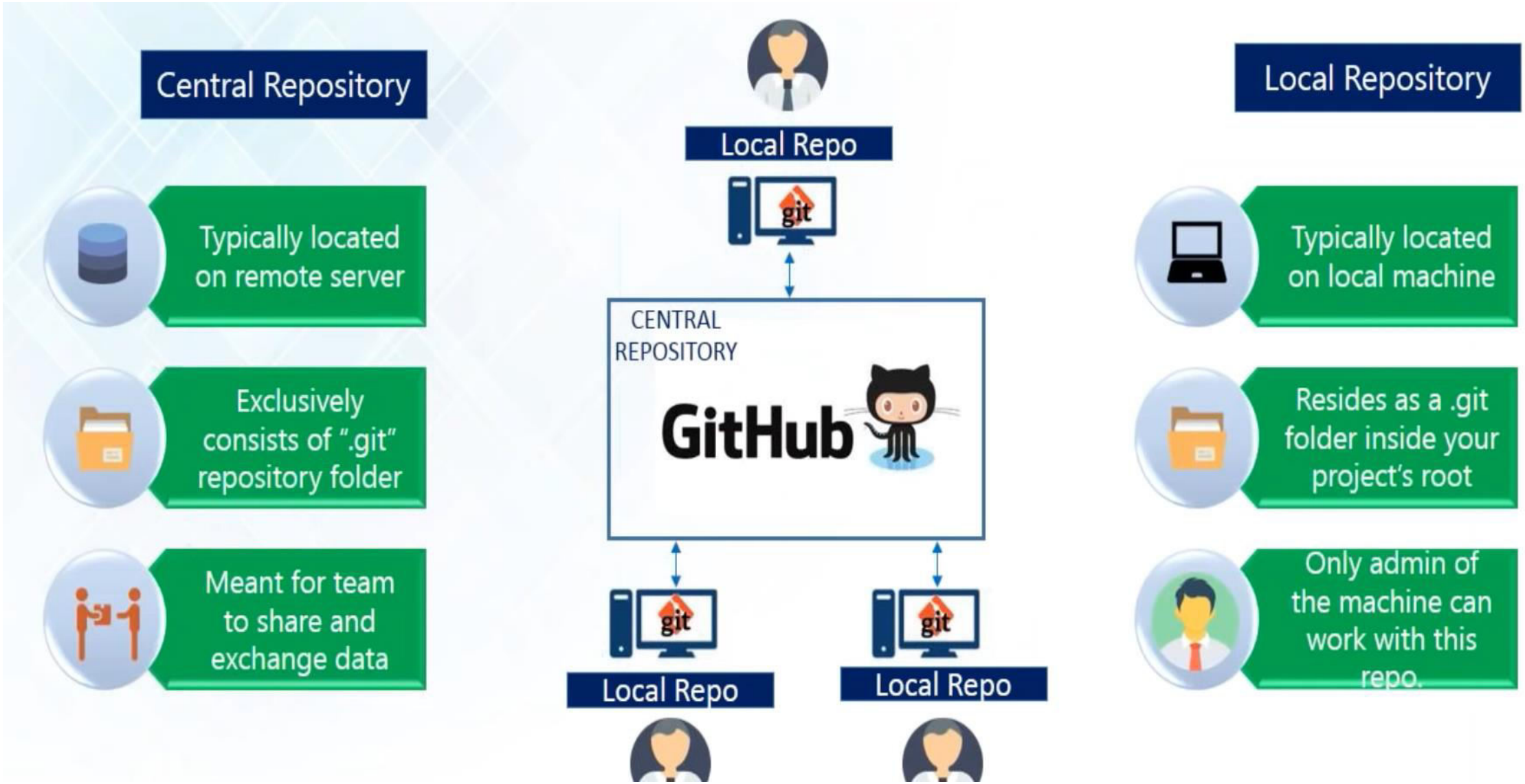
Git & GitHub

A directory or storage space where your projects can live. It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

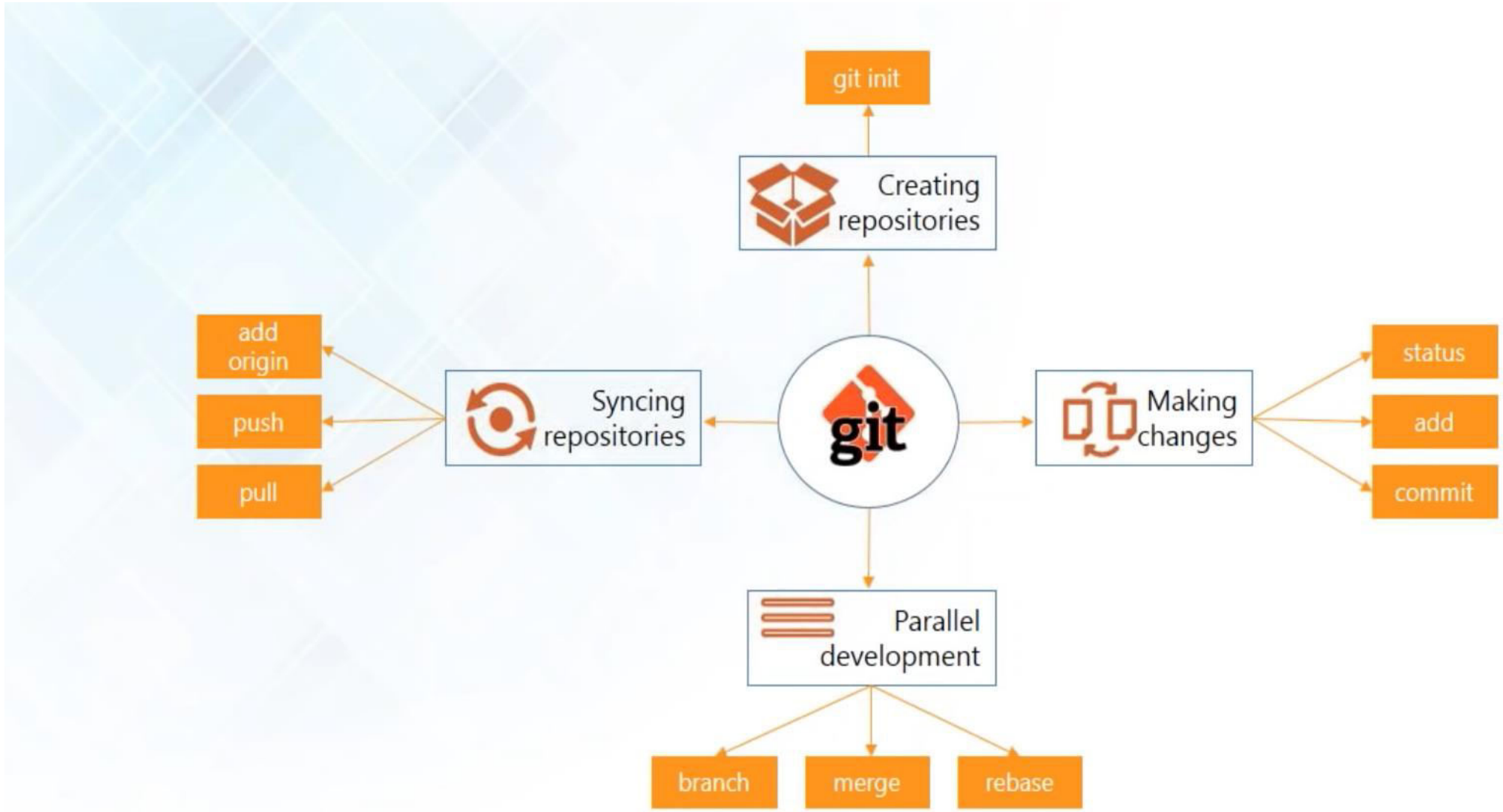
There are two types of repositories:

1. Central Repository
2. Local Repository

Git & GitHub


Central Repository
Typically located on remote server
Exclusively consists of ".git" repository folder
Meant for team to share and exchange data
Local Repo
git
CENTRAL REPOSITORY
GitHub
Local Repo
Local Repo
Local Repository
Typically located on local machine
Resides as a .git folder inside your project's root
Only admin of the machine can work with this repo.

Git & GitHub


git init
Creating repositories
add origin
push
pull
Syncing repositories
git
Making changes
status
add
commit
Parallel development
branch
merge
rebase

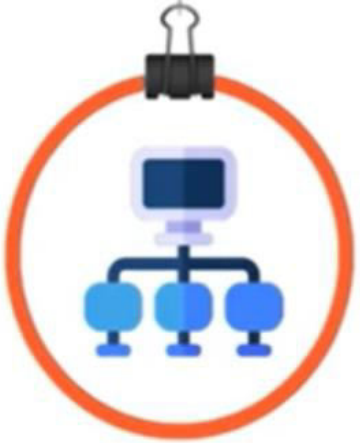
Git & GitHub

Git is a distributed version control tool which is used for managing source code

It's a software tool

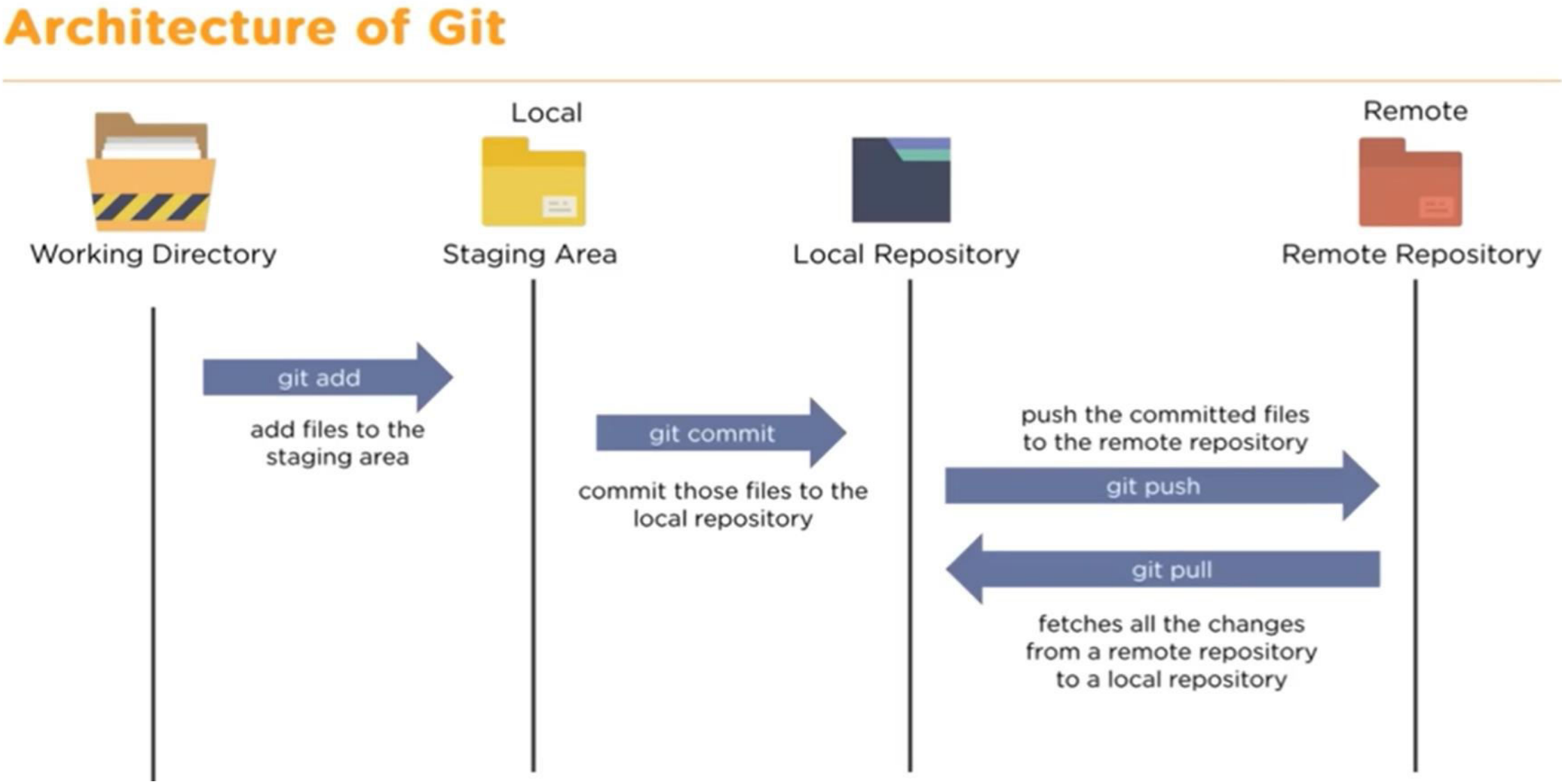
It is used to track changes in the source code

Multiple developers can work together



Supports non-linear development

Git & GitHub


Architecture of Git
Local
Remote
Working Directory
Staging Area
Local Repository
Remote Repository
git add
add files to the staging area
git commit
commit those files to the local repository
push the committed files to the remote repository
git push
git pull
fetches all the changes from a remote repository to a local repository