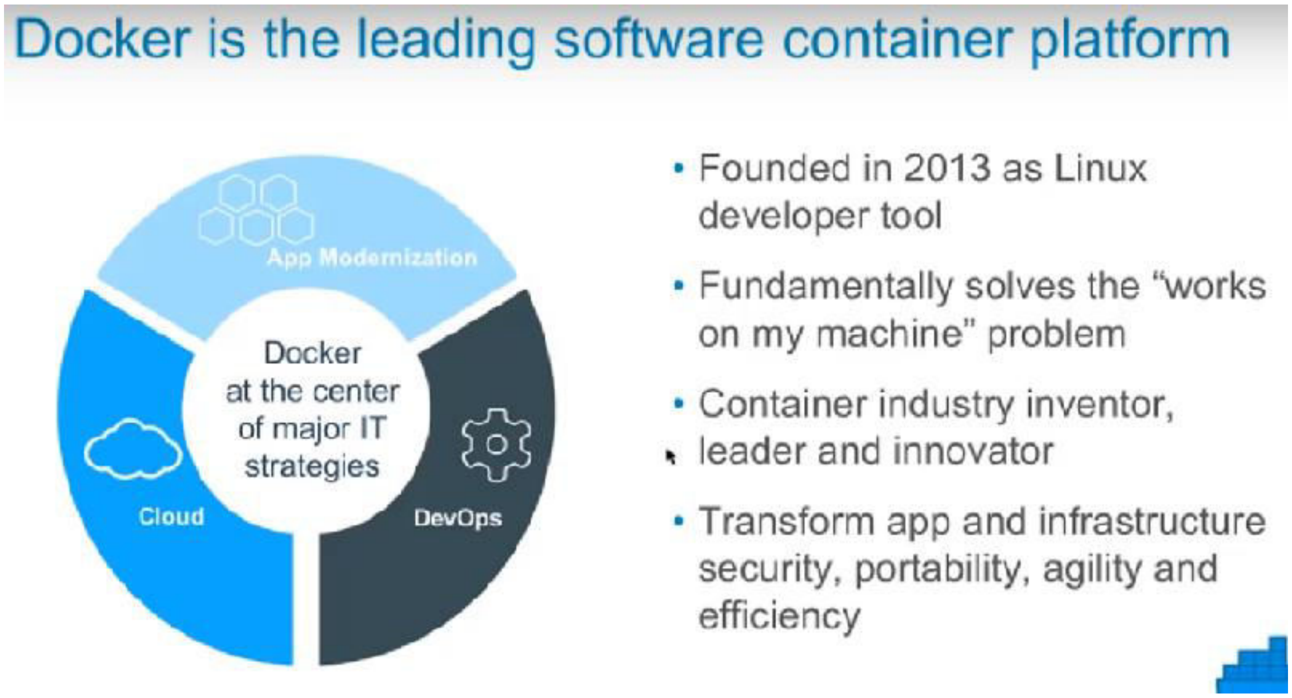
@ More About Docker

Docker is a free piece of S/W that allows users to develop and manage apps in an environment that takes **advantage of Linux-based software containers**. It's a great tool because it gives developers the ability to perform app management and development **without needing to use a virtual server or any extra hardware**.
This basically means that you can get **various IT infrastructure components to work together** with much less troubleshooting than other methods.

The **container technology** was originally developed with the aim of running several virtual operating systems in isolated environments on the same kernel. These are known as full-system containers.
The **Docker container** platform, on the other hand, focuses on so-called **application containers**, in which each application runs in its own virtual environment. While full-system containers are designed so that different processes can be executed in them, an application container always contains just one single process.
Extensive applications are therefore implemented as multi-container apps with Docker.

Pros : Efficient Tools, Affordable, Requires Limited System Resources, Open-Source

Cons: Large Learning Curve, Requires Perfect Syntax

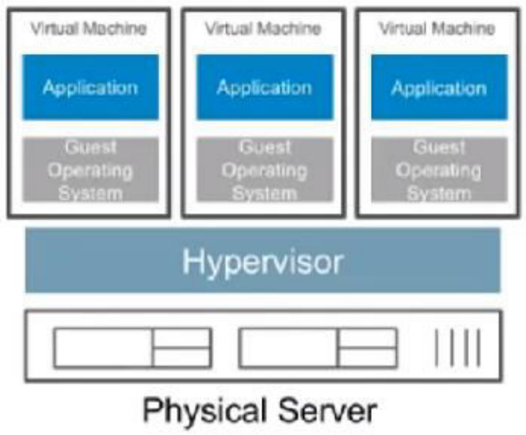


Docker is the leading software container platform

- Founded in 2013 as Linux developer tool
- Fundamentally solves the "works on my machine" problem
- Container industry inventor, leader and innovator
- Transform app and infrastructure security, portability, agility and efficiency

App Modernization
Docker at the center of major IT strategies
Cloud
DevOps

Lesson: Hypervisor-Based Virtualization



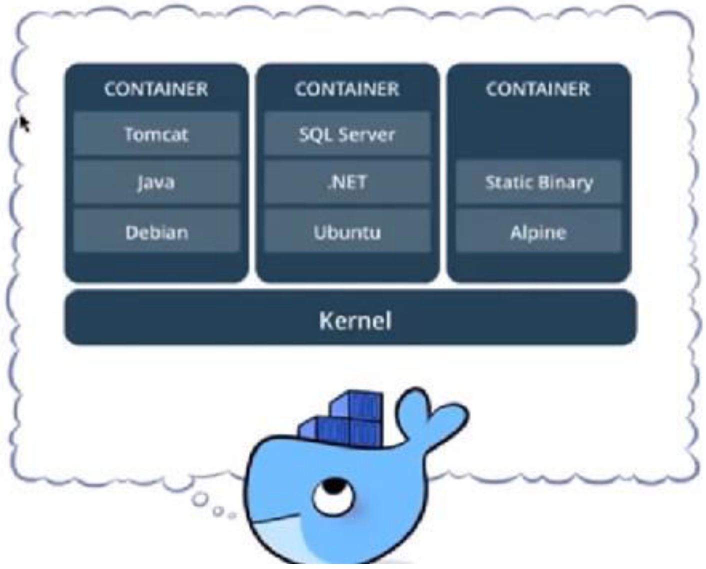
Benefits:

- Better resource pooling
  - One physical machine divided into multiple virtual machines
- Easier to scale
- VMs in the cloud
  - Rapid elasticity
  - Pay as you go model

Limitations:

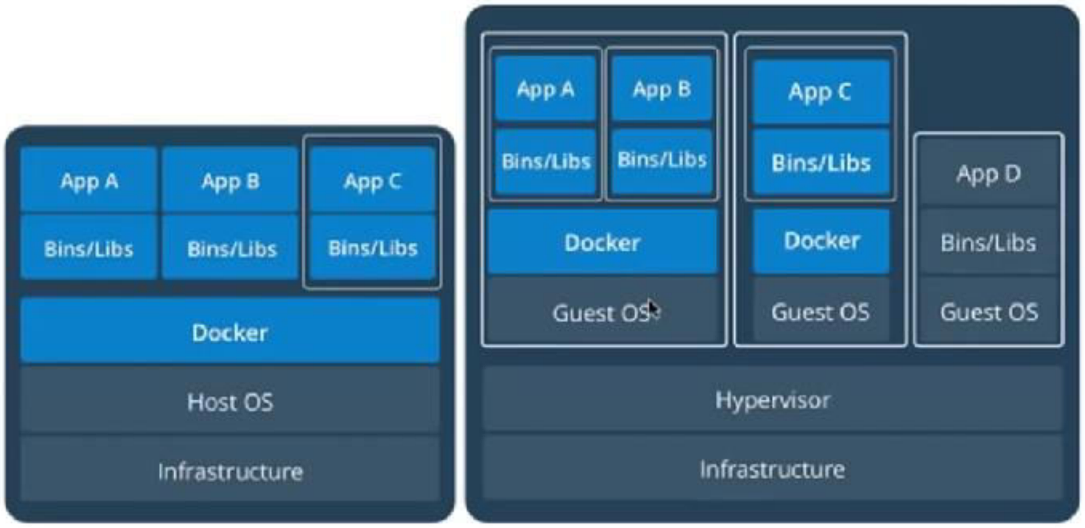
- Each VM stills requires:
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- Full guest OS means wasted resources
- Application portability not guaranteed

What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works with all major Linux and Windows Server

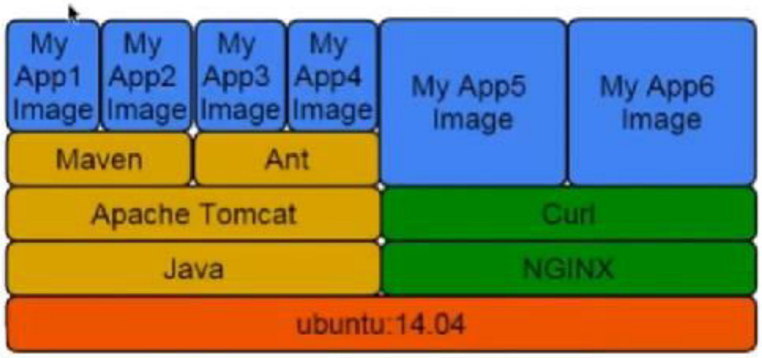
Containers and VMs Together

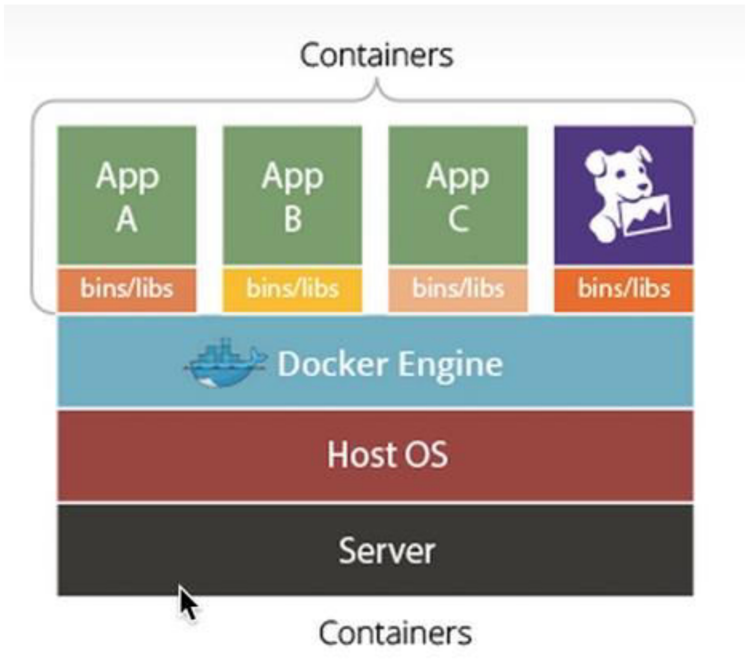


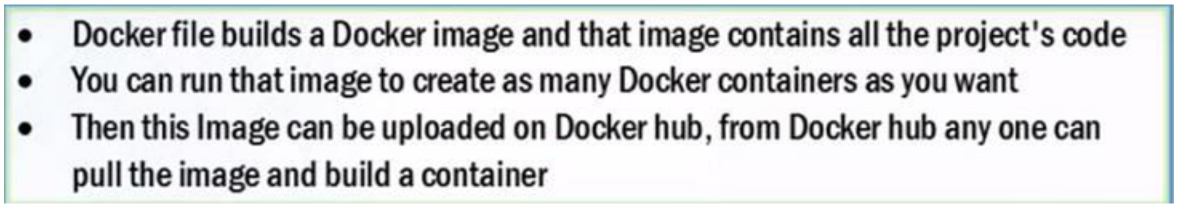
Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

Sharing Layers

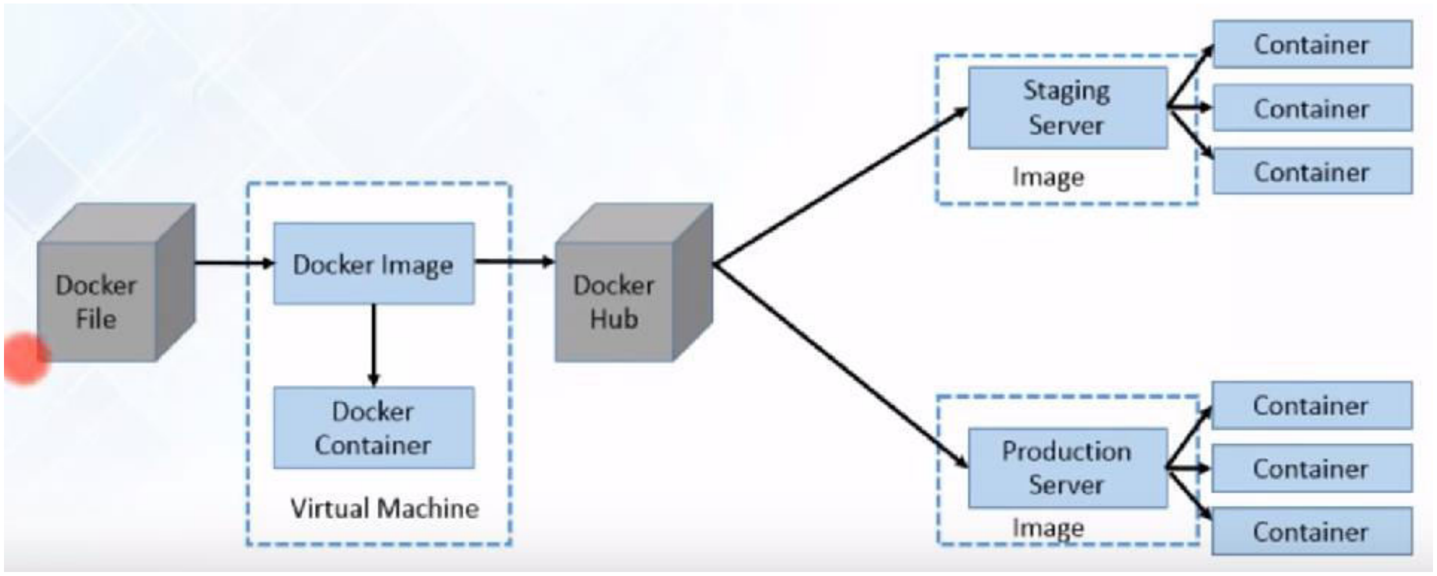
- Images can share layers in order to speed up transfer times and optimize disk and memory usage
- Parent images that already exists on the host do not have to be downloaded

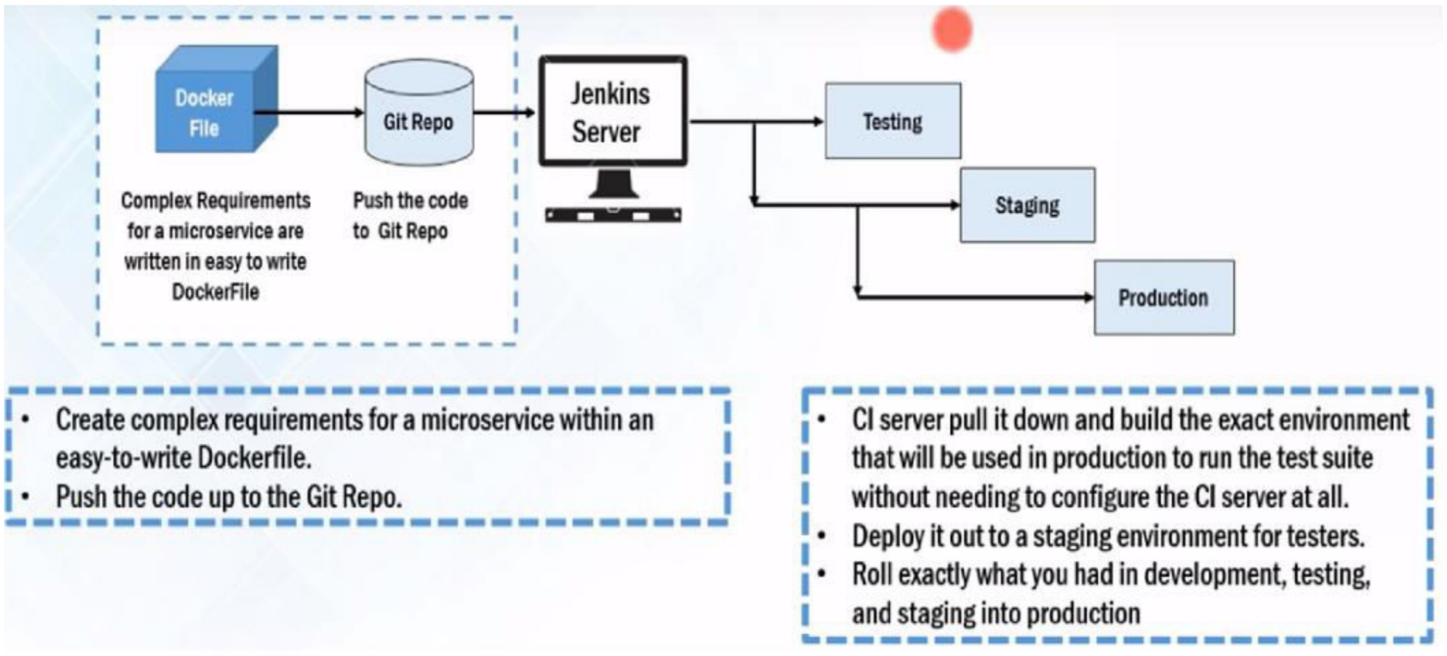






- Docker file builds a Docker image and that image contains all the project's code
- You can run that image to create as many Docker containers as you want
- Then this Image can be uploaded on Docker hub, from Docker hub any one can pull the image and build a container

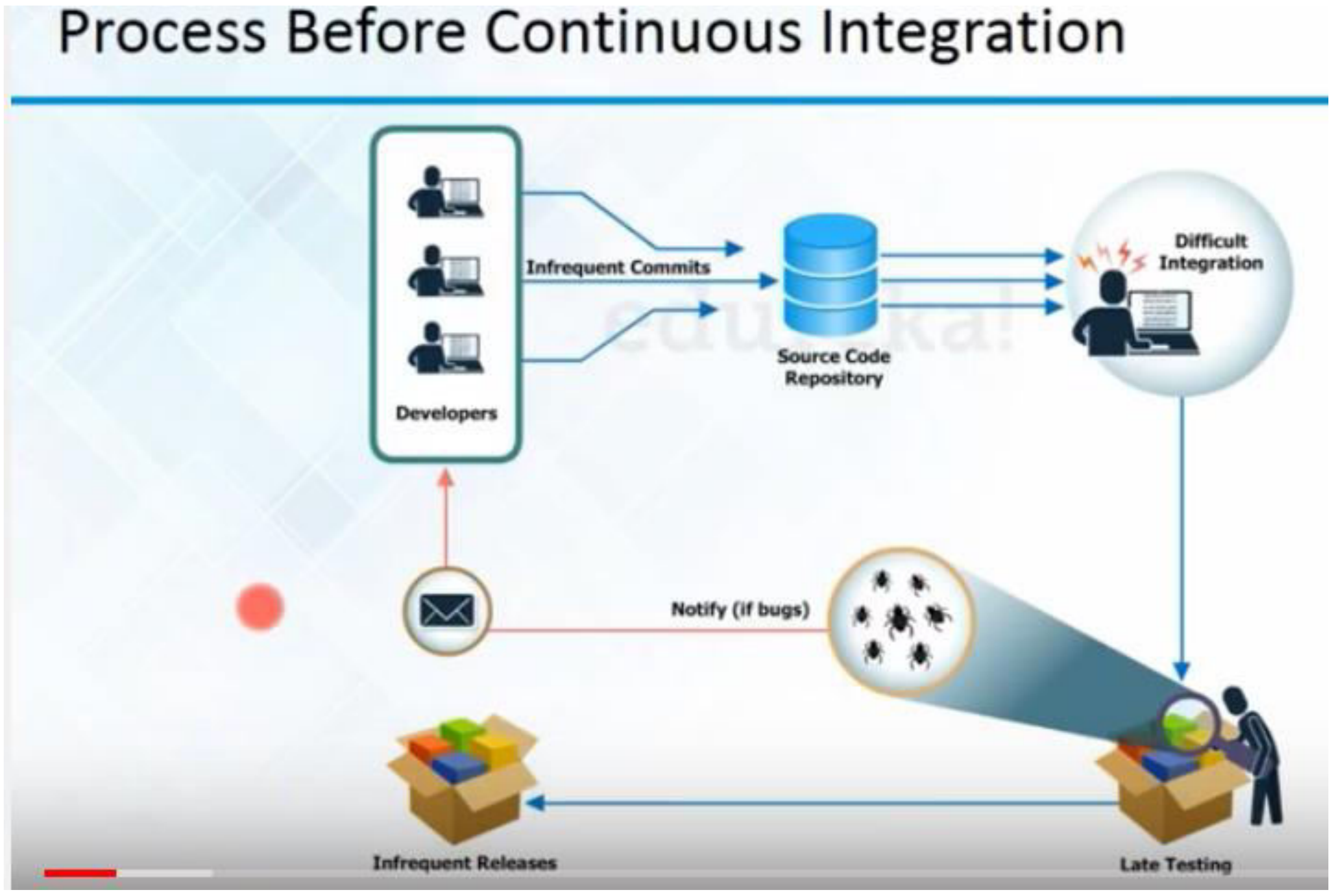


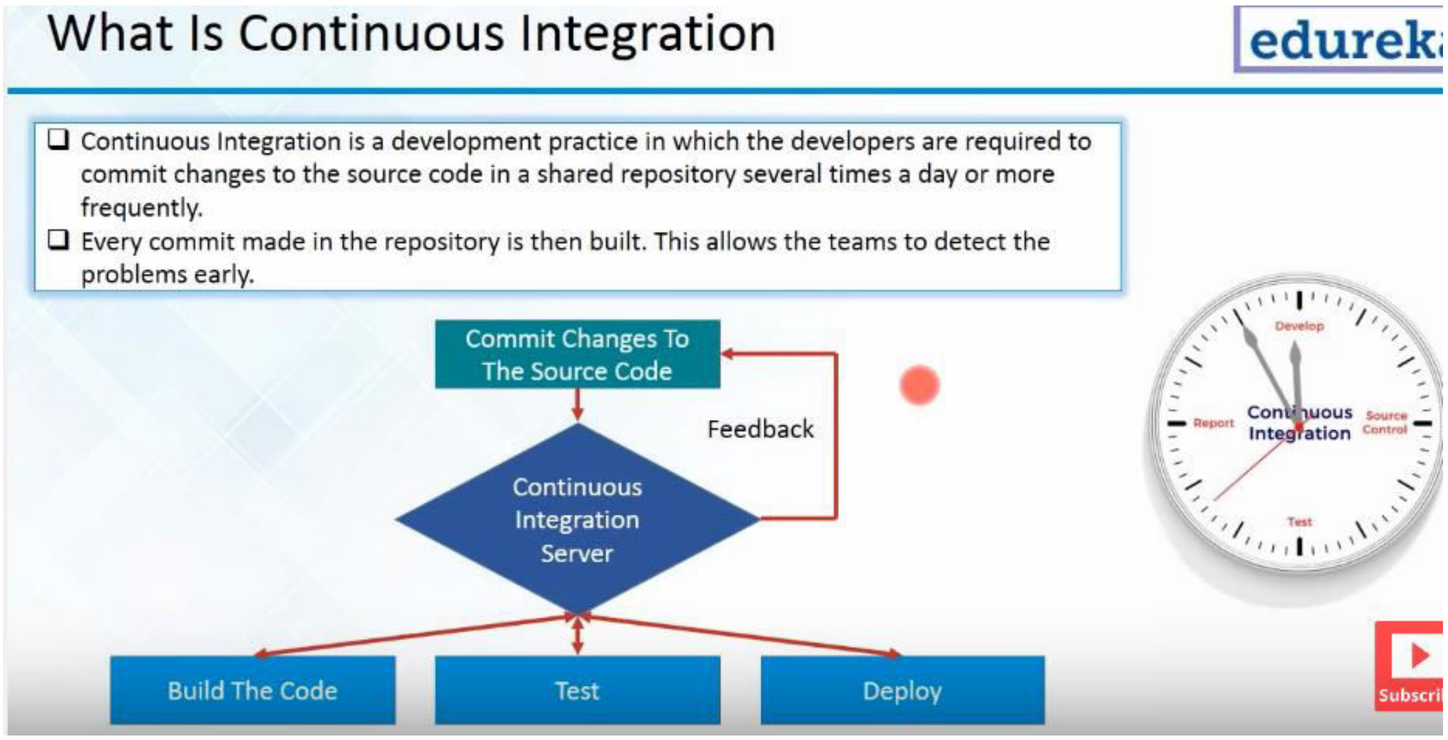

Docker File
Git Repo
Jenkins Server
Testing
Staging
Production
Complex Requirements for a microservice are written in easy to write DockerFile
Push the code to Git Repo

- Create complex requirements for a microservice within an easy-to-write Dockerfile.
- Push the code up to the Git Repo.

- CI server pull it down and build the exact environment that will be used in production to run the test suite without needing to configure the CI server at all.
- Deploy it out to a staging environment for testers.
- Roll exactly what you had in development, testing, and staging into production

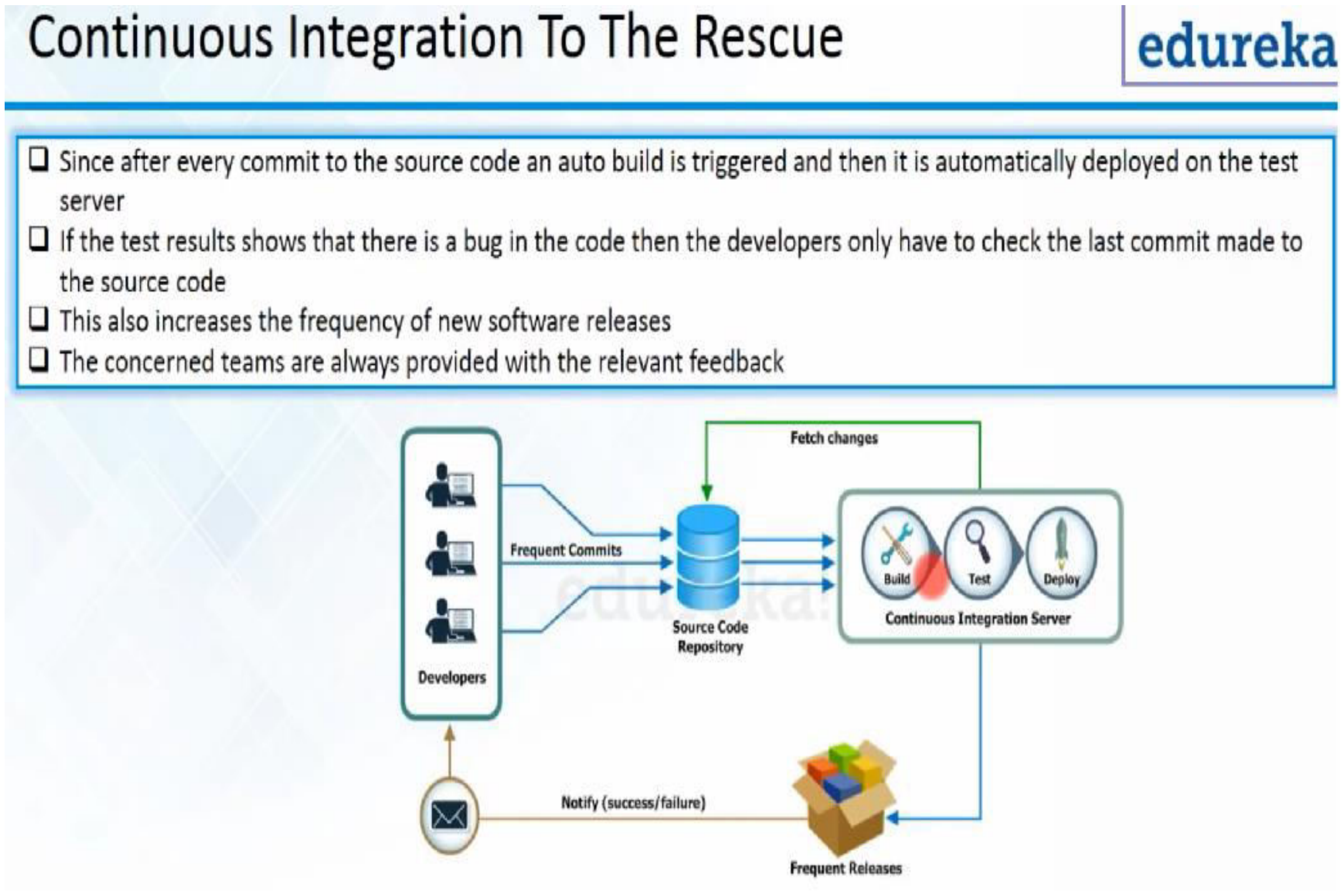
Before Jenkins


Process Before Continuous Integration



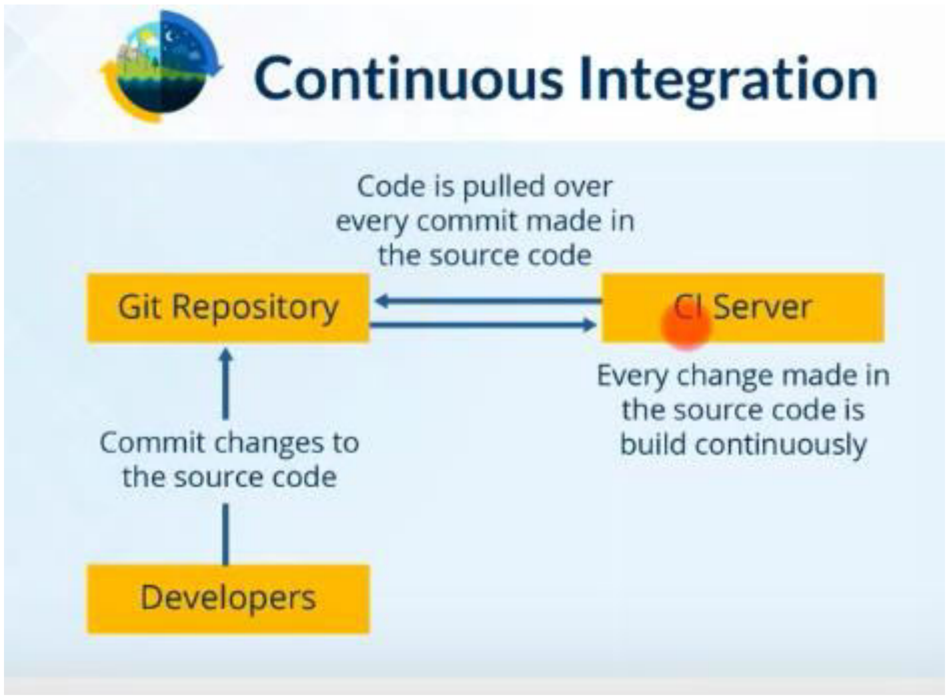
What Is Continuous Integration

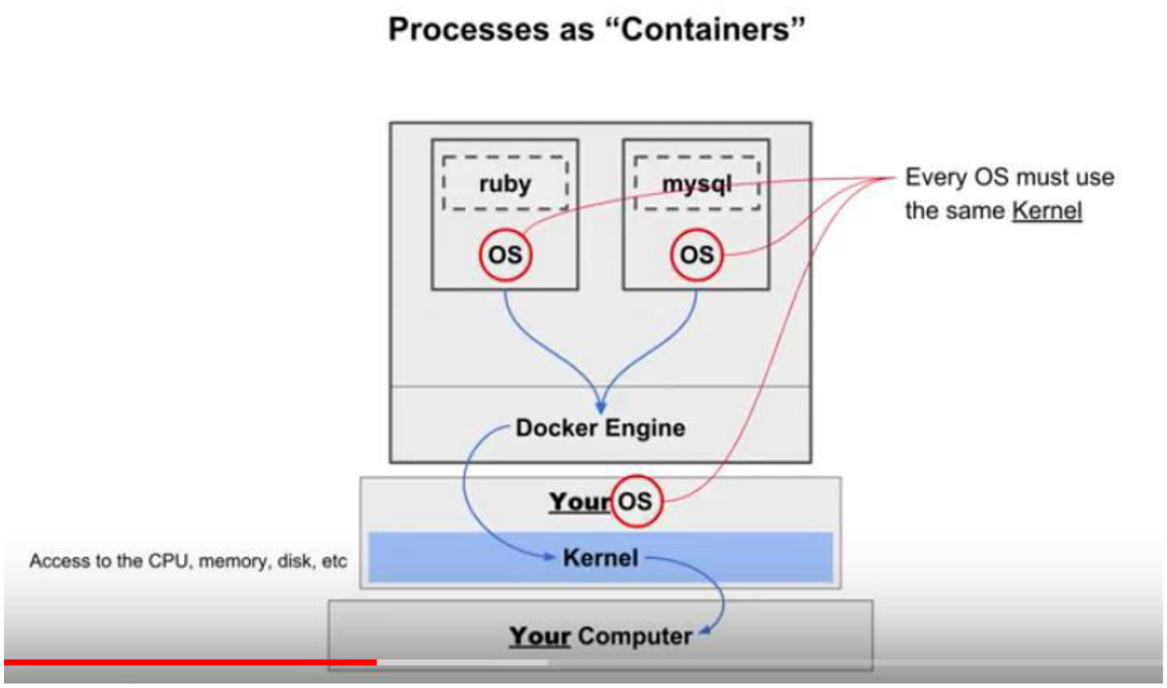
- ❑ Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently.
- ❑ Every commit made in the repository is then built. This allows the teams to detect the problems early.

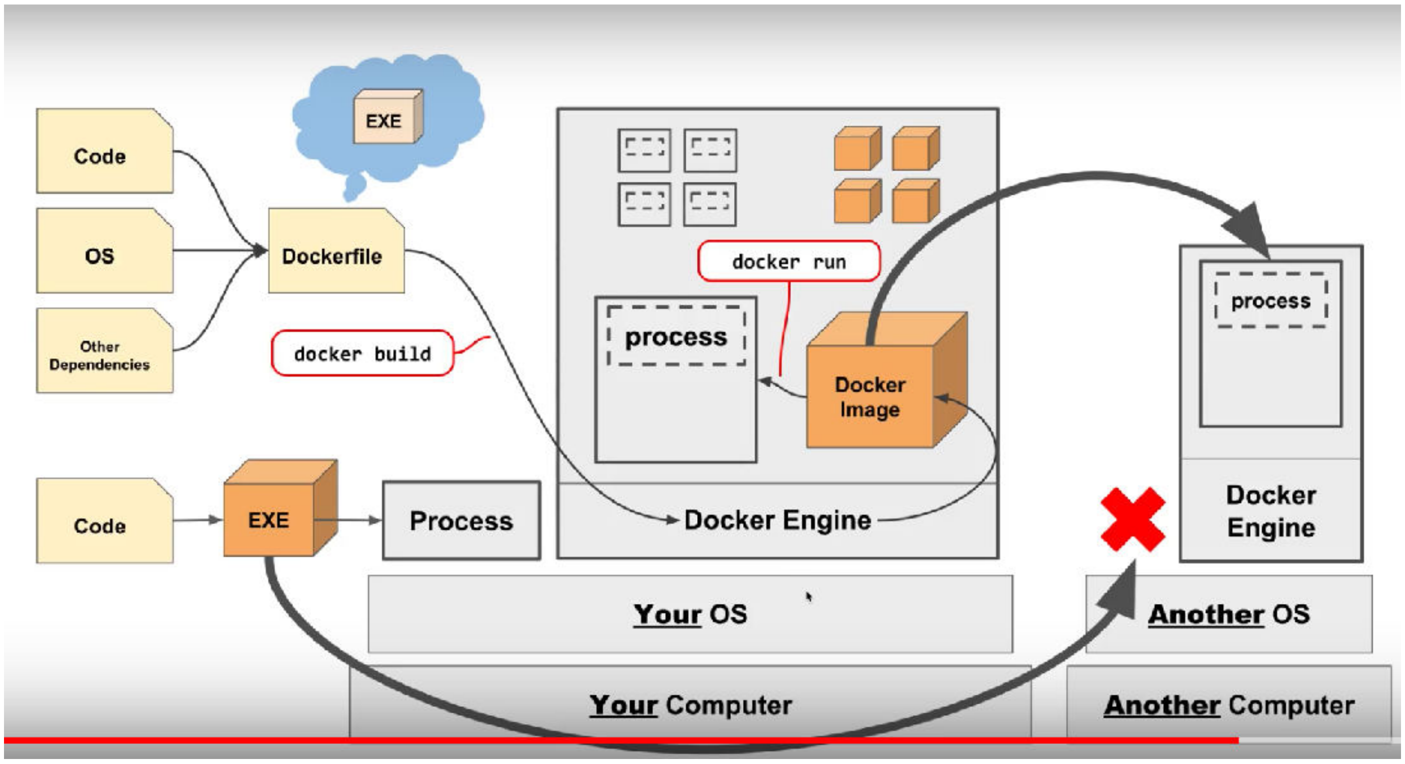


Continuous Integration To The Rescue

- ❑ 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


Continuous Integration
Code is pulled over every commit made in the source code
Git Repository
CI Server
Every change made in the source code is build continuously
Commit changes to the source code
Developers


Processes as "Containers"
ruby
mysql
OS
OS
Every OS must use the same Kernel
Docker Engine
Your OS
Kernel
Access to the CPU, memory, disk, etc
Your Computer



Jenkins,
is an open-source continuous integration software tool written in the Java programming language for testing and reporting
on isolated changes in a larger code base in real time.
The software enables developers to find and solve defects in a code base rapidly and to automate testing of their builds.

Apache vs. **Apache Tomcat**?
The Apache “http” server is often refered to as Apache and is a server that handles http requests.
Its job is to listen for requests and pass it on to the appropriate module to process.
There are modules to process C, Java, PERL, PHP, Python, Ruby etc.
Apache Tomcat is a Java Servlet container. It runs as an Apache http server module.
It constructs the HTML pages by executing Java Servlets and Java Server Pages and returns them to the http server.

JBoss vs. **Tomcat**: Choosing A Java Application Server
The big three are **Tomcat**, **Glassfish**, and **JBoss**(WildFly).

The Major Differences Between JBoss and Tomcat
Both JBoss and Tomcat are Java servlet application servers, but **JBoss** is a whole lot more.
The substantial difference between the two is that JBoss provides a full Java Enterprise Edition (JEE) stack, including Enterprise JavaBeans and many other technologies that are useful for developers working on enterprise Java applications. Tomcat is much more limited.

One way to think of it is that JBoss is a JEE stack that includes a servlet container and web server, whereas Tomcat, for the most part, is a servlet container and web server.

When To Choose JBoss?
JBoss is the best choice for applications where developers need full access to the functionality that the Java Enterprise
Edition provides and are happy with the default implementations of that functionality that ship with it.
If you don't need the full range of JEE features, then choosing JBoss will add a lot of complexity to deployment and
resource overhead that will go unused.
For example, the JBoss installation files are around an order of magnitude larger than Tomcat's.

When To Choose Tomcat?
Tomcat is a Java servlet container and web server, and, because it doesn't come with an implementation of the full JEE stack,
it is significantly lighter weight out of the box. For developers who don't need the full JEE stack that has two main advantages.
- Significantly less complexity and resource use.
- Modularity.

There are numerous providers of add-ons that work with Tomcat.
Developers can choose the specific implementations they want to use to add extra functionality.

For example, Tomcat can't natively host Enterprise JavaBeans.
However, if users need Enterprise JavaBeans (EJB) functionality like the persistence and transaction processing that the EJB container model provides, but want to avoid the problems inherent in the main implementation, there are many lightweight alternatives, including the Spring Framework and OpenEJB