upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

Introduction to DevOps

Presented By- Alok Tiwari

Contents

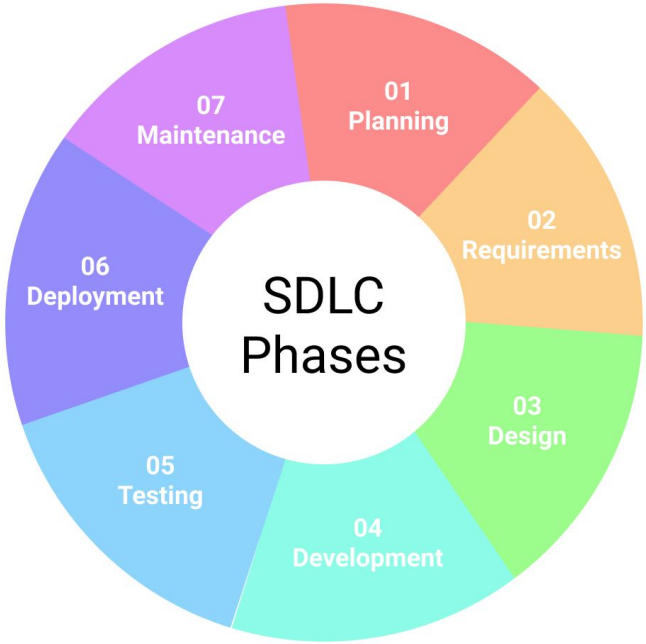
- Introduction to SDLC
- Agile
- DevOps
- DevOps Tools -
  - Software Configuration Management- Git and GitHub
  - Docker
  - CI/CD- Jenkins

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC)

The software development life cycle (SDLC) is a process which is used to develop software. SDLC is a step by step procedure need to be followed by the organization to design and develop a high quality product. The phases of software development life cycle are which describes that how to develop, maintain particular software. The life cycle aims to develop a good quality product/software. SDLC produces intermediate products that can be reviewed to check whether they work according to customer requirement.



- SDLC is also known as Software development process.
- SDLC is an approach creates considerable documentation where this documentation helpful to make sure that requirement can be traced back to stated business requirements.
- It is a framework which has a set of tasks performed at each phase in the software development process.

Phases of SDLC

Planning

The planning phase is the most vital step in determining if and how to build your software project. Some of the typical tasks involved in this phase include:

- Planning capacity and budget
- Scheduling timelines
- Allocating resources and more

Organizations enter this phase with the idea of building a software solution and exit with a series of concrete plans, schedules, and estimations.

Requirements Analysis

The next step after understanding your capacity and intent is to understand your idea better. In the requirements analysis phase, business and IT teams communicate to understand the business and technical requirements of the final product. Finally, you need to document what are the business processes that the software fulfills. Organizations enter this phase with a rough idea of what they want out of their software. The outcome from here is a detailed document that defines all types of requirements in detail.

Design

After laying out the requirements, software architects and developers develop designs for the final software. These designs explain how the application would look and how it would function behind the scenes. Design patterns define the application architecture. Developers utilize design patterns and algorithms to fulfill the application's requirements in the best way possible. Sometimes prototypes are also created to gain feedback and improve the designs. The input to this phase is the requirements for the software, and the output is a well-laid out plan to build it.

Development

This phase is where the actual magic happens. Development teams take up the design and a suitable SDLC model and produce the code required for the software application. The intent here is two-fold—high-quality code and low turn-up time. Many organizations choose to involve business stakeholders in this phase to keep the product's growth aligned to their expectations. The input to this phase is the design for the software, and the outcome is fully working code.

Testing

Testing teams conduct various tests on the built product to ensure that the software is of the highest quality. In most cases, each line of code is put to the test automatically to ensure that no bugs slip into the final product. The bugs reported are fixed by the development team and then tried again for assurance. Some teams choose to automate the test execution process to make it easy to find bugs using continuous integration tools like Appveyor or Travis CI. The input to this phase is a working version of the software, and the output is a thoroughly tested and reliable version of the same.

Deployment

Once you have the tested working version of the software that matches your requirements, the next step is to roll it out to your customers. Usually, this process is automated, and the development team can release stable versions as and when required. However, in complex cases, DevOps teams assist them in this process. Application Release Automation (ARA) tools are used to automate the deployment process and include manual reviews when necessary. The input to this phase is a thoroughly tested version of the software. The outcome is the release of the software to the users.

Maintenance

While it may seem that the software development process was complete when you rolled it out to your users, it usually does not end there. In most cases, software teams monitor the performance of a deployed version of the software and fix any anomalies that occur. If users encounter and report bugs, they are addressed and resolved in this phase. Most changes made here call for another bout of testing and deployment, and this process keeps on looping until the software is active in the market. There are no strict inputs or outputs for this phase. Instead, it focuses on improving the quality of the software while target users are using it.

Why Use the SDLC Model?

- **SDLC brings order to the process:** One of the most significant advantages of using an SDLC model is that you follow a predefined set of steps instead of an ad hoc approach. In addition, thanks to relying on established methods, there's little chance of a software project not turning out the way you intended due to a lack of experience or organization.

- **SDLC offers a common vocabulary:** With a well-defined set of rules, you and your team can be on the same page on how to go about building your next software project. As a result, you can expect your team to understand each step well and act accordingly, which reduces entropy and enables higher levels of collaboration.

- **SDLC enables better communication between development and business:** When you use one of the SDLC models, your stakeholders know where they fit in the process. They have a high-level overview of how you will be building their product and can be better involved.

- **SDLC couples each phase of the process well with the next one:** It is often tricky for cross-domain teams (like development and testing) to understand when they have completed their task and when they need to step out. SDLC breaks down the entire process into distinct steps and reduces overlap between them to help teams coordinate effectively.

- **SDLC phases can loop around easily:** Most SDLC models support looping back to any step to perfect the final deliverable. Looping back helps to dedicate extra time towards a particular phase if needed. This results in highly reliable and well-tested end products

- **SDLC eliminates common issues even before software development begins:** With the use of SDLC models, you clear a lot of clutter even before you start. SDLC also defines the roles and scopes of each team member clearly, which reduces overlap and conflicts later in the process. With dedicated time for each critical step such as planning, designing, etc., you don't miss out on any of these accidentally.

- **SDLC helps design projects better:** Since SDLC involves a deliberate designing phase, there is particular emphasis on solving problems in the design phase instead of the coding phase. This approach helps develop innovative strategies and optimized plans before writing code, resulting in a well-designed project and reduced coding time.

- **Personnel switch is more effortless when following SDLC models:** SDLC models offer plenty of documentation to help any new software developer learn and get started with a project very quickly. So If a member of your team drops out from an ongoing project, you do not need to worry about the orientation of the replacement member.

Examples of Models that Use the SDLC

Waterfall Model:
This SDLC model is the oldest and most straightforward. With this methodology, we finish one phase and then start the next. Each phase has its own mini-plan and each phase "waterfalls" into the next. The biggest drawback of this model is that small details left incomplete can hold up the entire process.

Agile Model:
The Agile SDLC model separates the product into cycles and delivers a working product very quickly. This methodology produces a succession of releases. Testing of each release feeds back info that's incorporated into the next version. According to Robert Half, the drawback of this model is that the heavy emphasis on customer interaction can lead the project in the wrong direction in some cases.

Iterative Model:
This SDLC model emphasizes repetition. Developers create a version very quickly and for relatively little cost, then test and improve it through rapid and successive versions. One big disadvantage here is that it can eat up resources fast if left unchecked.

V-Shaped Model:

An extension of the waterfall model, this SDLC methodology tests at each stage of development. As with waterfall, this process can run into roadblocks.

Big Bang Model:

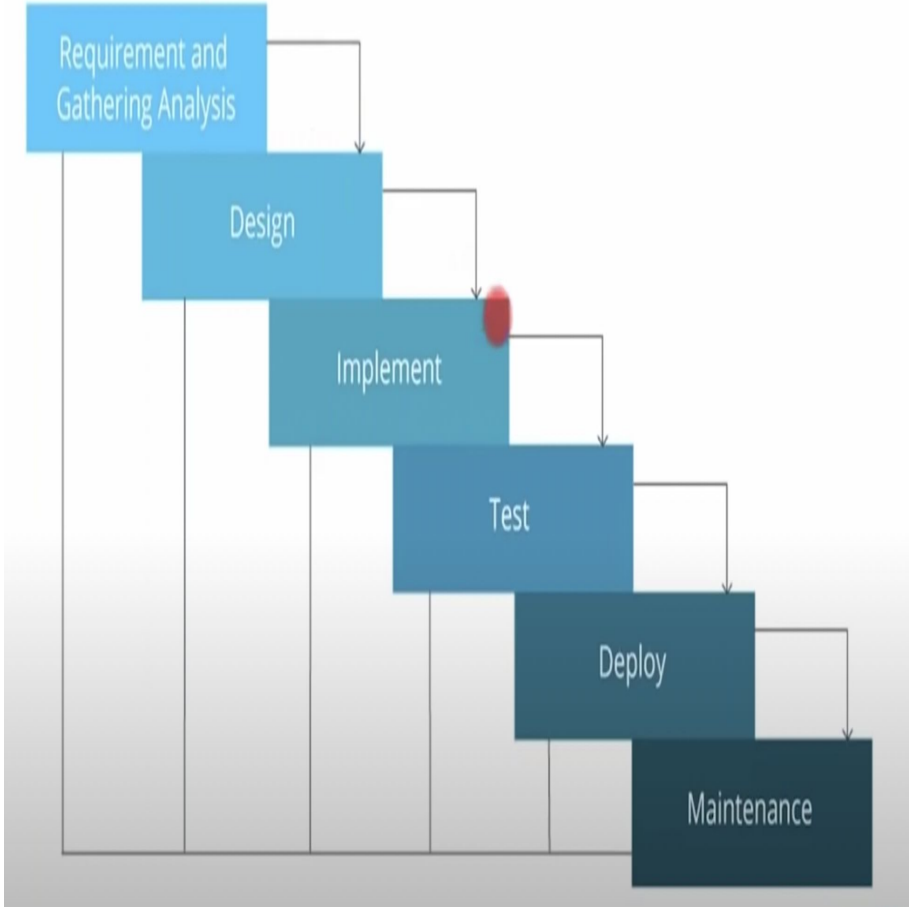
This high-risk SDLC model throws most of its resources at development and works best for small projects. It lacks the thorough requirements definition stage of the other methods.

Spiral Model:

The most flexible of the SDLC models, the spiral model is similar to the iterative model in its emphasis on repetition. The spiral model goes through the planning, design, build and test phases over and over, with gradual improvements at each pass.

Waterfall Model

- The Waterfall method is one of the oldest software development lifecycle models. It follows a rigid, predetermined path that consists of a set of steps. Like a waterfall, the multiple phases cascade into each other – with the end of each step initiating the subsequent one. The final product is ready at the end of the last step.
- The Waterfall model begins with a thorough planning and design phase. Since you can not go backward in this model, it is essential to plan and design the most optimal product as early in the process as possible. Then you need to proceed with the development and testing phases before deploying for the end-user.
- The rigid nature of the model deems it inappropriate for large or innovative projects. Furthermore, there is no support for feedback through the process; hence, you can not adapt to the market's evolving requirements over time. The lack of flexibility in the Waterfall model is one of the top reasons why many models came up for more iterative variants of SDLC.



WaterFall Model Pros and Cons

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

Pros	Cons
1. Very simple to understand 2. Has a straightforward management process 3. All phases come one after another in a fixed sequence 4. It is easy to determine critical points of the development cycle 5. It is easy to classify and prioritize the various stages of the process	1. The process is very rigid; you can not make changes on the go 2. There is a high risk involved, as you do not know the final product's quality beforehand 3. Not suited for long-term, complex projects 4. It is difficult to judge the progress of a phase individually

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

Introduction to Agile

Agile Model

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

- The Agile method is one of the most popular software development methodologies to date. The Agile Manifesto created in 2001 addressed significant problems in the traditional waterfall model that limited developers from producing the best quality software products.

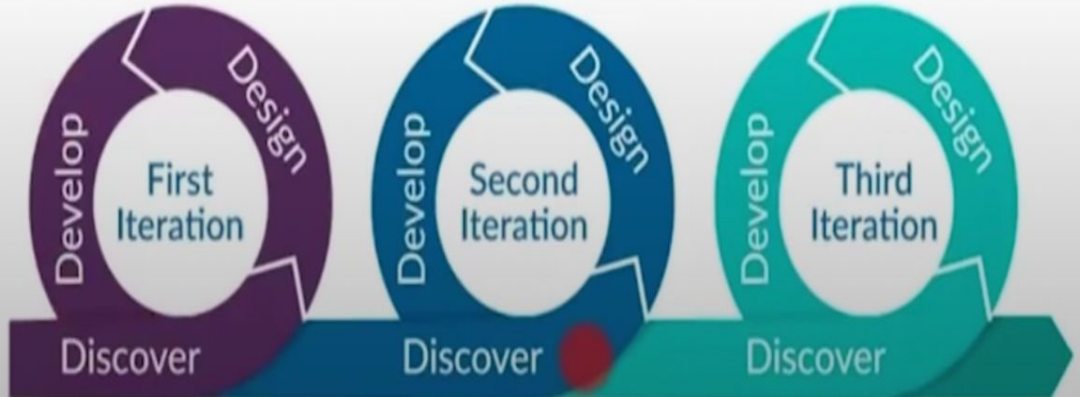
- This model believes in embracing the uncertainty in the software development process and utilizing it to come out with something new and better. This model develops a very primitive yet fully functional version of the app very early in the process. Then with the help of **iterative "sprints,"** more and more features are added to the app, and user feedback is taken into account to shape the app's evolution. Each sprint ends with a fully functional version of the app, which contains a subset of the features planned for the app.

- Over time, several variations of Agile have emerged. Scrum is one of them which defines roles and events (also called ceremonies) as part of its process. Kanban is another, which is much simpler with enhanced flexibility and fewer restrictions. Most modern teams combine these variations to create a customized Agile model for themselves.

In the Agile Methodology each project is broken up into several 'Iterations'

All Iterations should be of the same time duration (between 2 to 8 weeks)

At the end of each iteration, a working product should be delivered

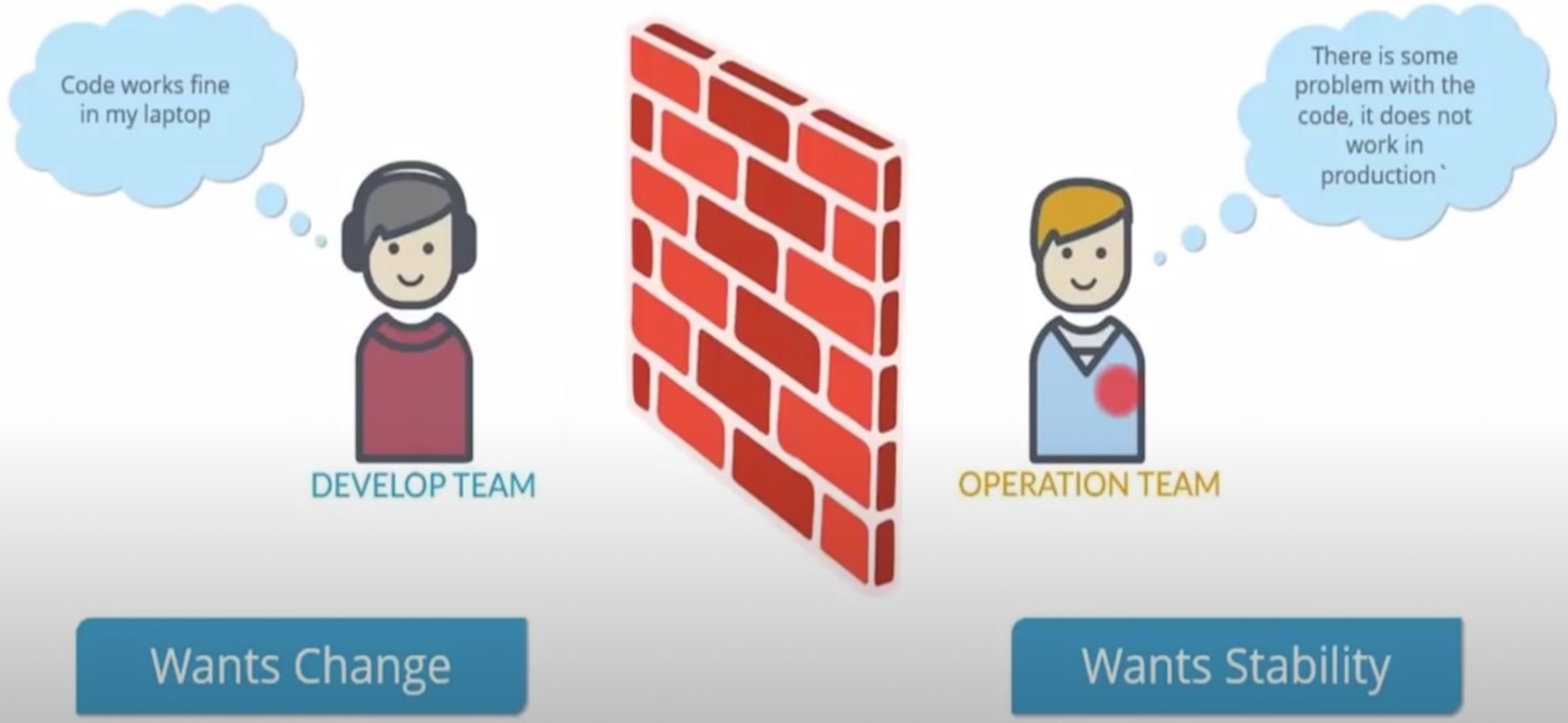


Agile Model Pros and Cons

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

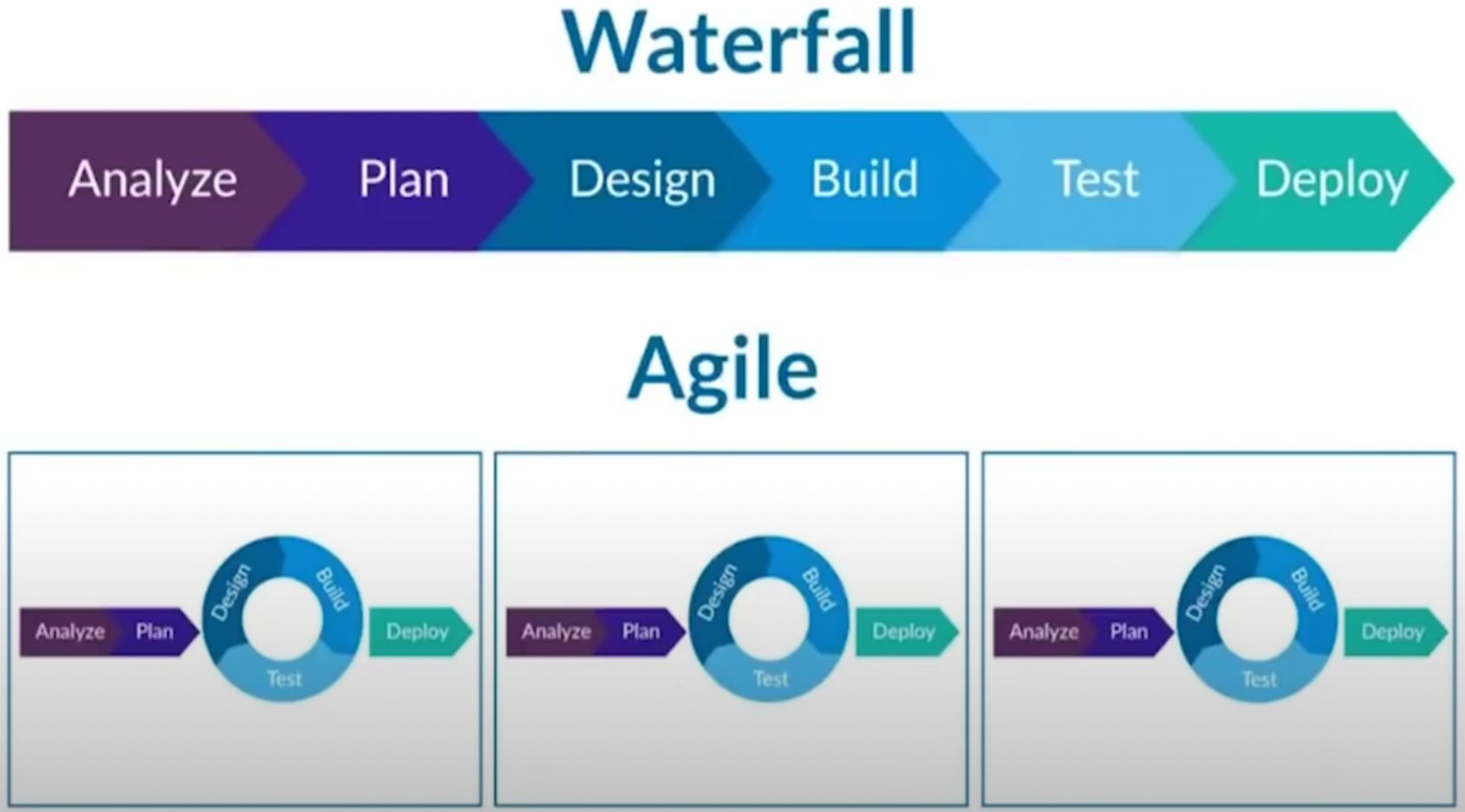
Pros	Cons
1. Supports changes and evolution of the product requirements from very early in the process 2. Short iterations help to keep the project light and active 3. Due to the flexibility involved, the risk is minimum 4. You can get a stable yet primitive version of the application at an early stage of your process	1. The final cost is difficult to measure due to the repetitive nature of the process 2. Requires a highly professional and user-oriented team for execution 3. The final product may evolve to become too different from the initial plan and might even conflict with the initial features 4. With the room for changes on the go, there is a high chance that the projects may overshoot their set durations

Limitations of Agile

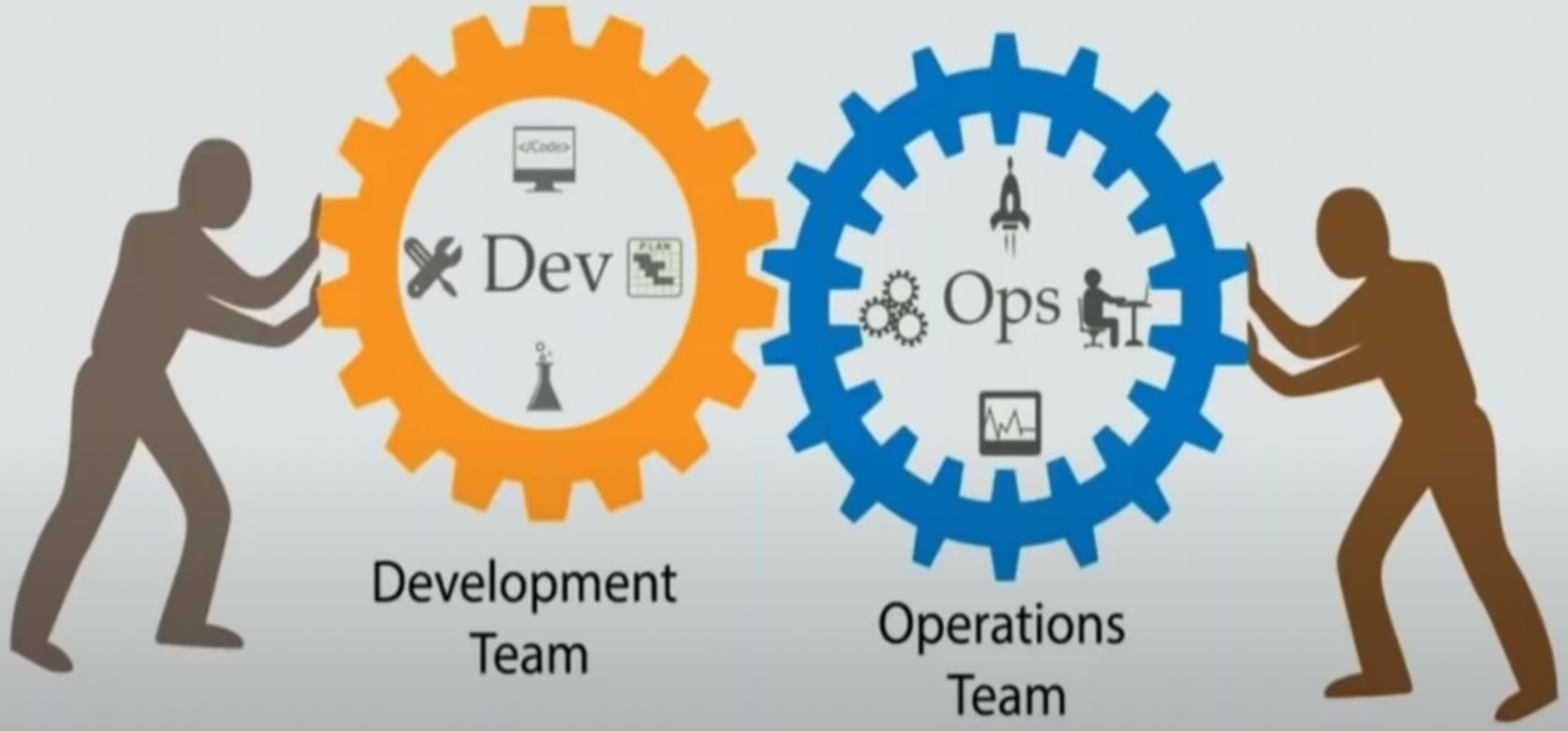


Waterfall vs Agile

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM



Solution is DevOps



upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

Development and operations <DevOps>

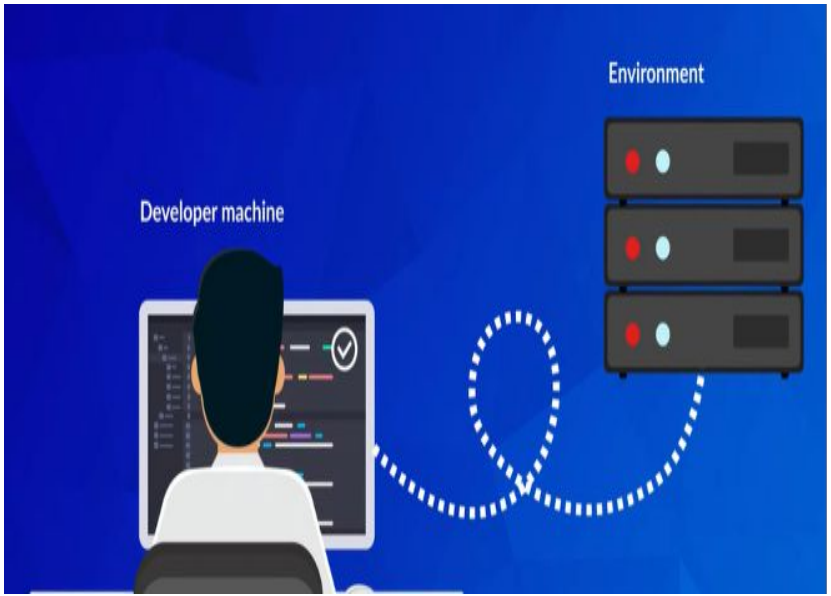
Motivation

"This code works perfectly fine on my machine but does not work in test or pre-prod or prod or other environments."

These situations could be frustrating but can be avoided by following some principles or using some concepts.

So questions comes to the minds are:

- ➔ How to make sure that code can be deployed in a repeatable fashion?
- ➔ How to make sure that we can have consistent results when deploying the same code in different environments?
- ➔ What is the best way to package my code?
- ➔ How do I promote my code or application from one environment to another?
- ➔ What is the best or good way to deploy an application?




DEVOPS

DevOps Dimensions

The best tools to address each of the 4 dimensions of DevOps Practices.

1 Plan & Track
Collaborative planning and tracking tools are a must if you want disparate groups to work together across the organization.

2 Dev & QA
Dev and QA teams need to work together so a standard set of tools that encourage a single source of truth is a must.

3 Release & Deploy
Frequent Release and Deploy requires automation of both infrastructure provisioning and software configuration and deployment.

4 Monitor & Optimize
You need to be able to react quickly and fix production issues so that the business can have trust in the DevOps process and you constantly learn and improve.

Agile

focuses on
processes

highlighting
change

while accelerating
delivery

CI/CD

focuses on
software-defined life cycles

highlighting
tools

that emphasize
automation

DevOps

focuses on
culture

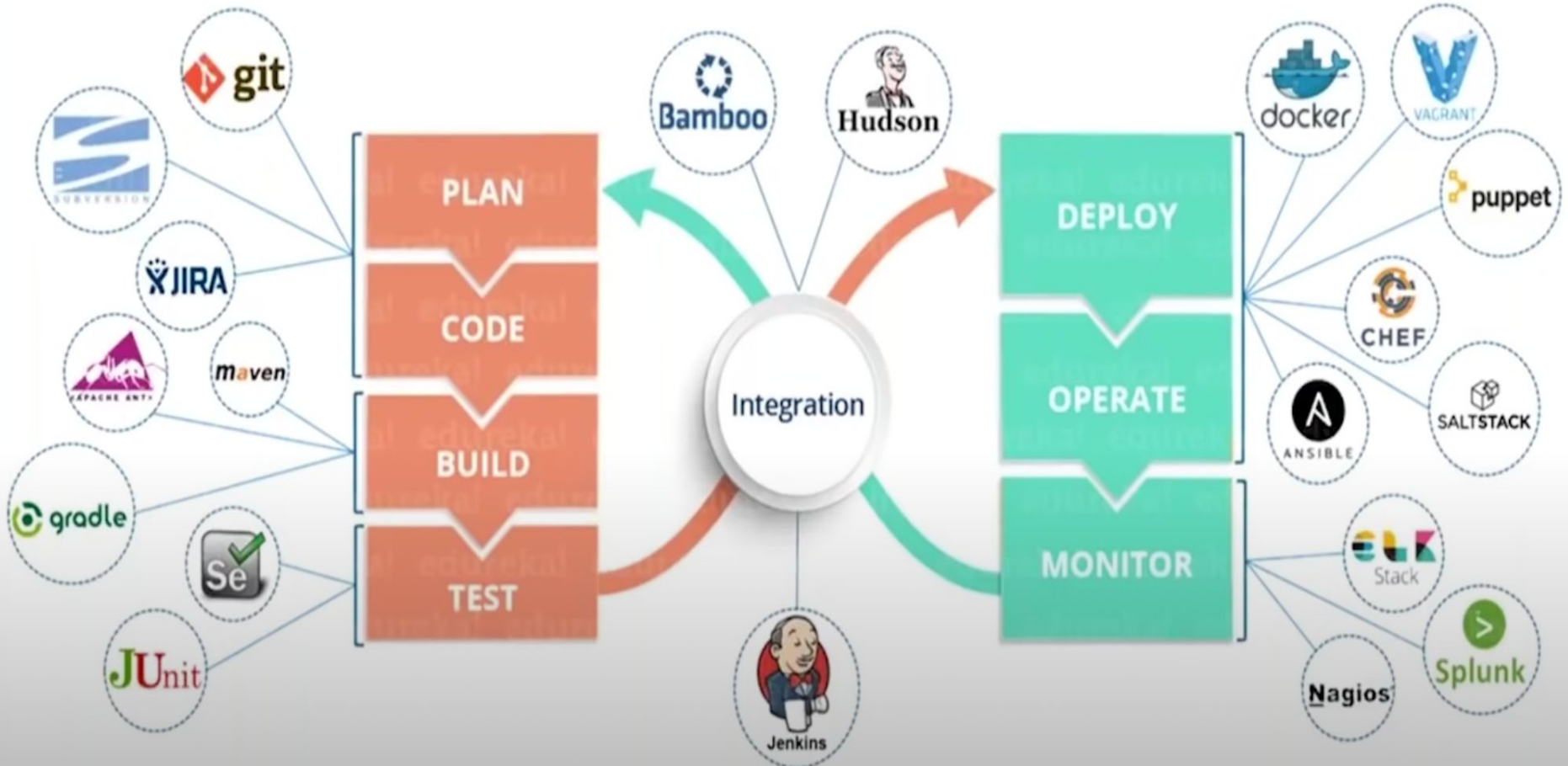
highlighting
roles

that emphasize
responsiveness

Dev Challenges	DevOps Solution
Waiting time for code deployment	Continuous integration ensures there is a quick development of code, faster testing, and a speedy feedback mechanism.
Pressure of work on old , pending and new code	There is no waiting time to deploy the code. Hence the developer focuses on building the current code.

Ops Challenges	DevOps Solution
Difficult to maintain uptime of the production environment	**Containerization/Virtualization** ensures there is a simulated environment created to run the software as containers offer great reliability for service uptime.
Tools to automate infrastructure management are not effective	**Configuration Management** helps you to organize and execute configuration plans, consistently provision the system & proactively manage their infrastructure.
No. of servers to be monitored increases	**Continuous Monitoring** Effective monitoring and feedbacks systems is established through Nagios. Thus effective administration is assured.
Difficult to diagnose and provide feedback on the product	

DevOps Tools



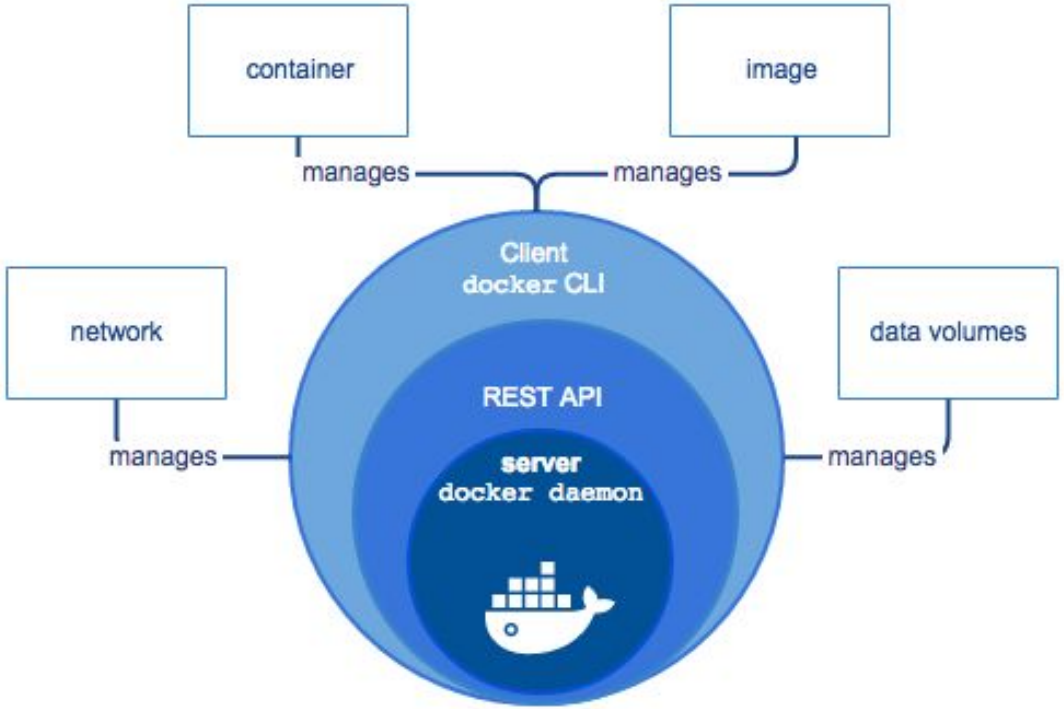
upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

DevOps Tools <Docker>

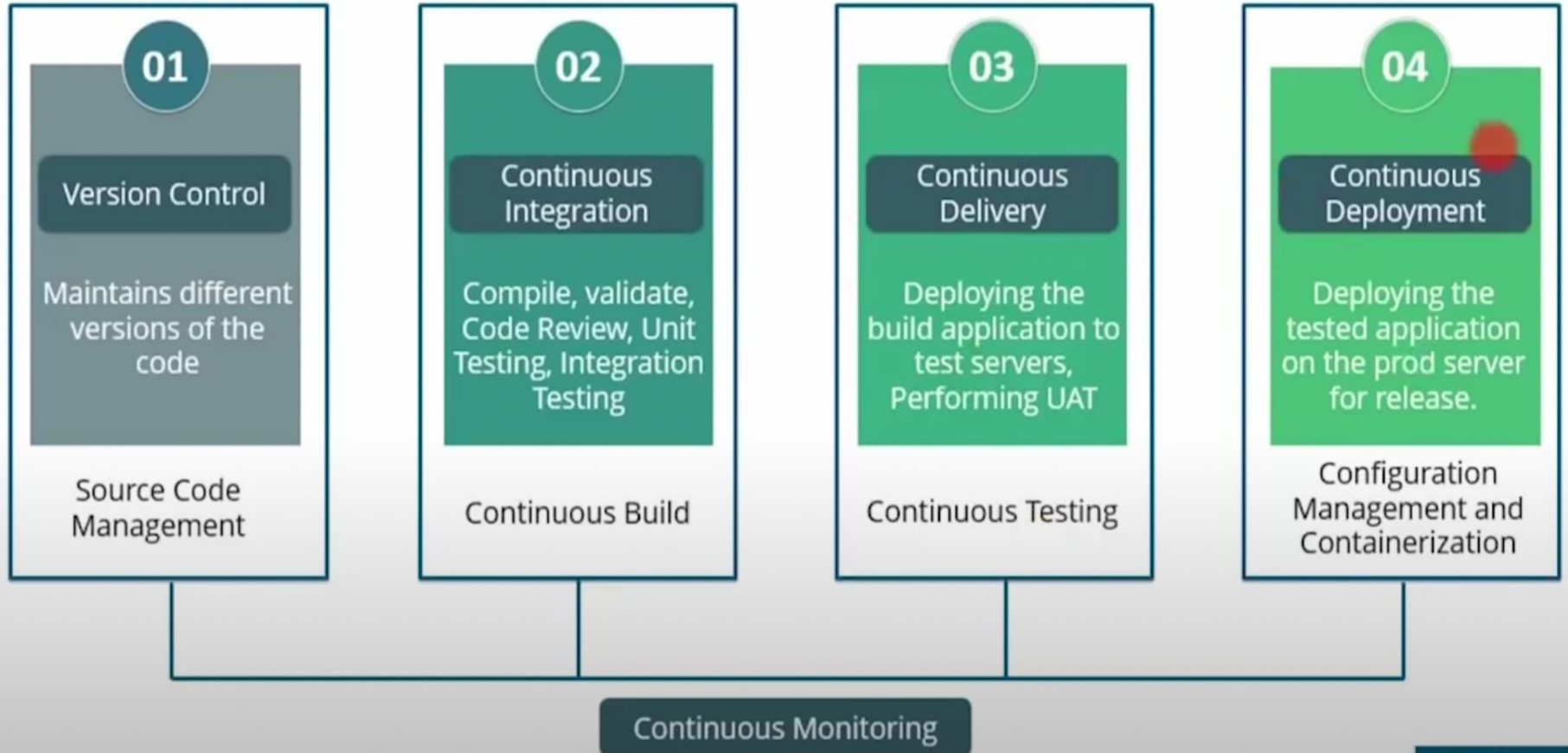
Docker Engine

Docker Engine is a client-server application with these major components:

- A Server
- A REST API
- A Command Line interface (CLI)



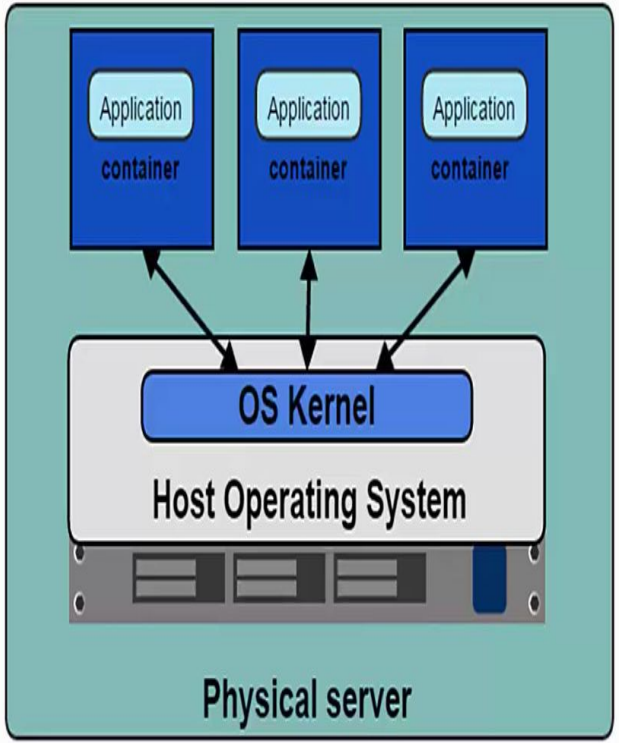
Docker Stages



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



The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

QA server

Public Cloud

Production Cluster

Disaster recovery

Customer Data Center

Production Servers

Contributor's laptop

Can I migrate smoothly and quickly?

The Matrix from Hell

	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	?	?	?	?	?	?	?

Cargo Transport Pre-1960

Multiplicity of Goods







Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing







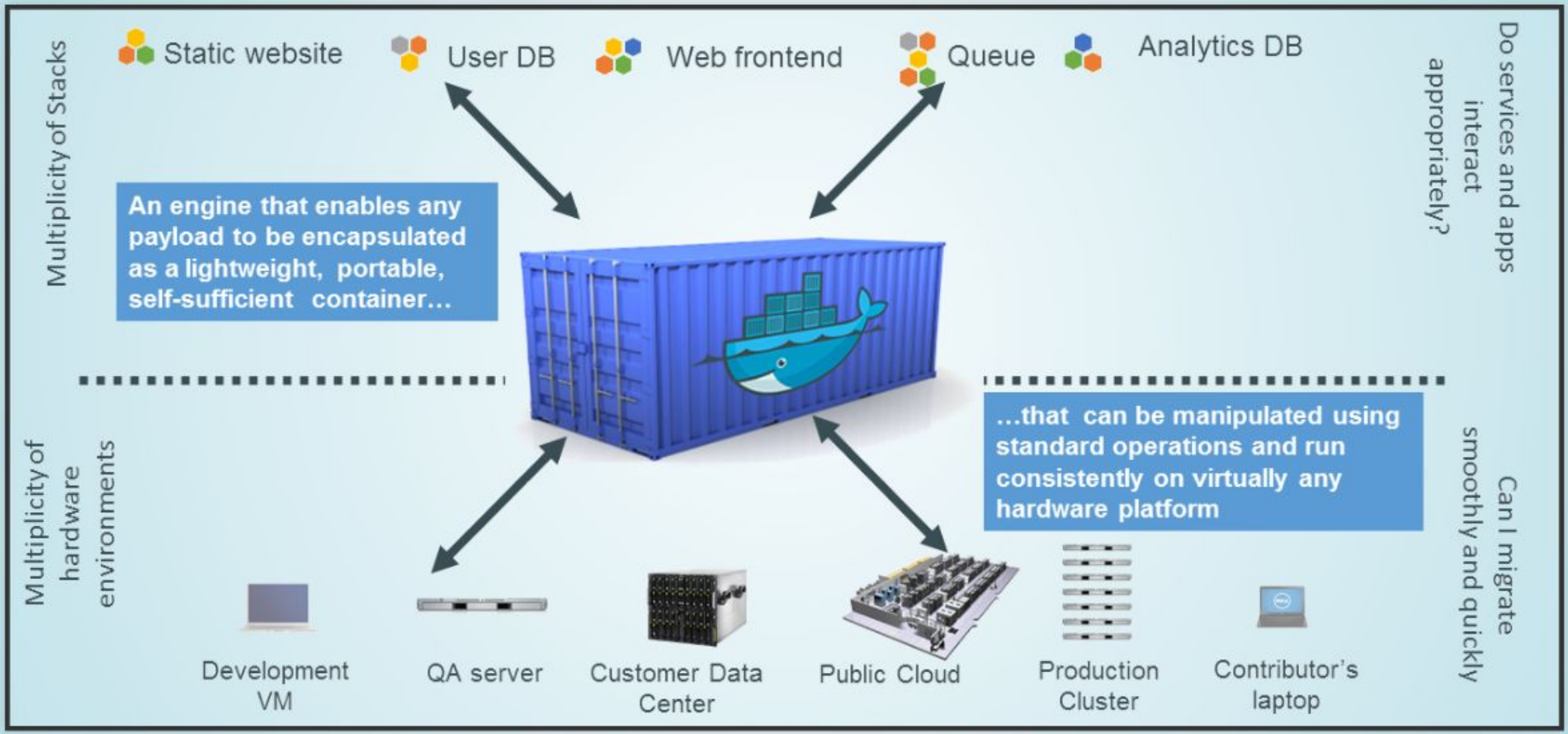
Can I transport quickly and smoothly (e.g. from boat to train to truck)

Also a Matrix from Hell

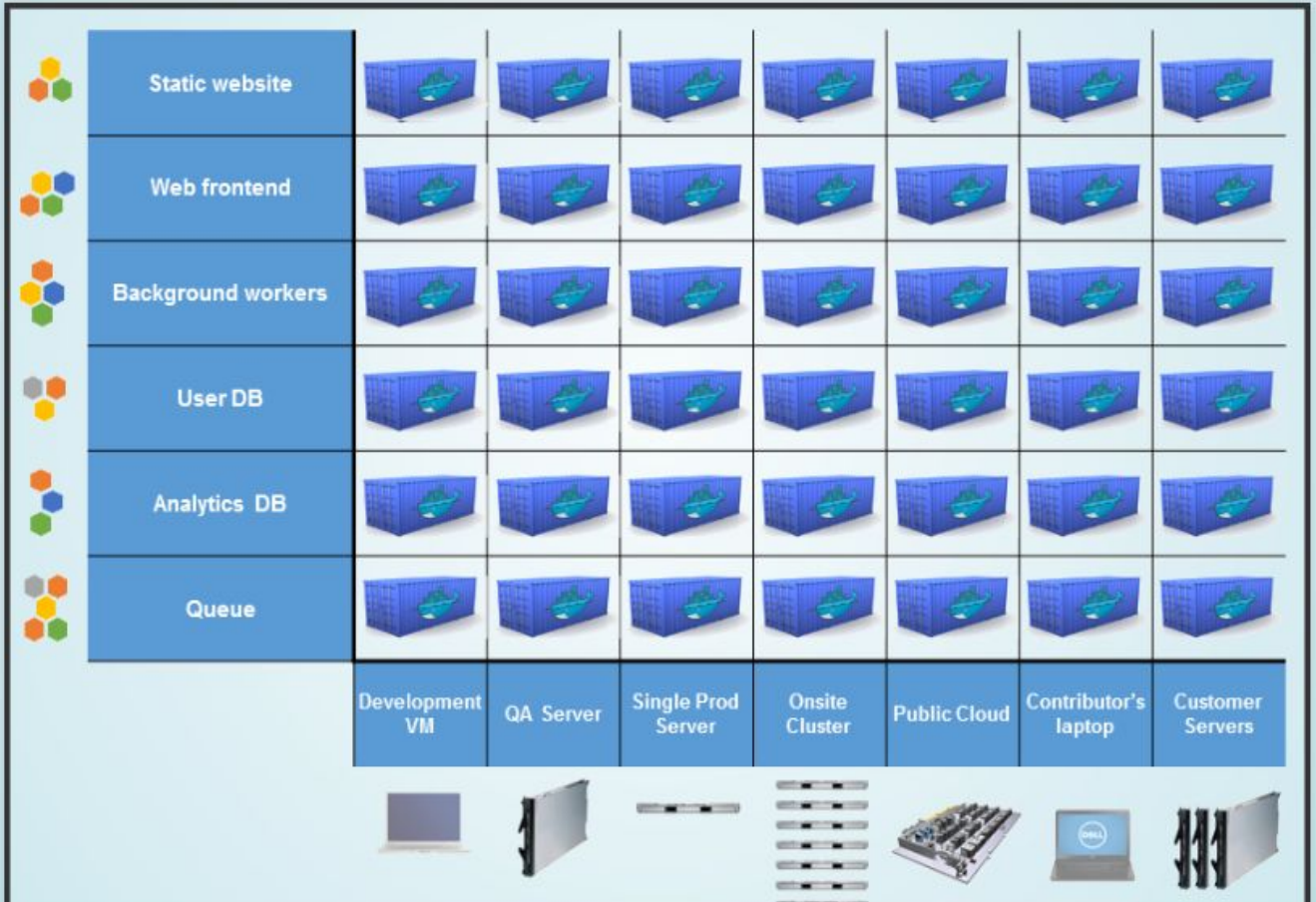
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container

Docker is a Container System for Code



Docker Eliminates the Matrix from Hell



Why Developers Care

Build once... (finally) run *anywhere**

- A clean, safe, hygienic, portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying.
- Automate testing, integration, packaging...anything you can script.

** Where "anywhere" means an x86 server running a modern Linux kernel (3.2+ generally or 2.6.32+ for RHEL 6.5+, Fedora, & related)*

Docker

Dev: It works fine in my system!

Tester: It doesn't work in my system

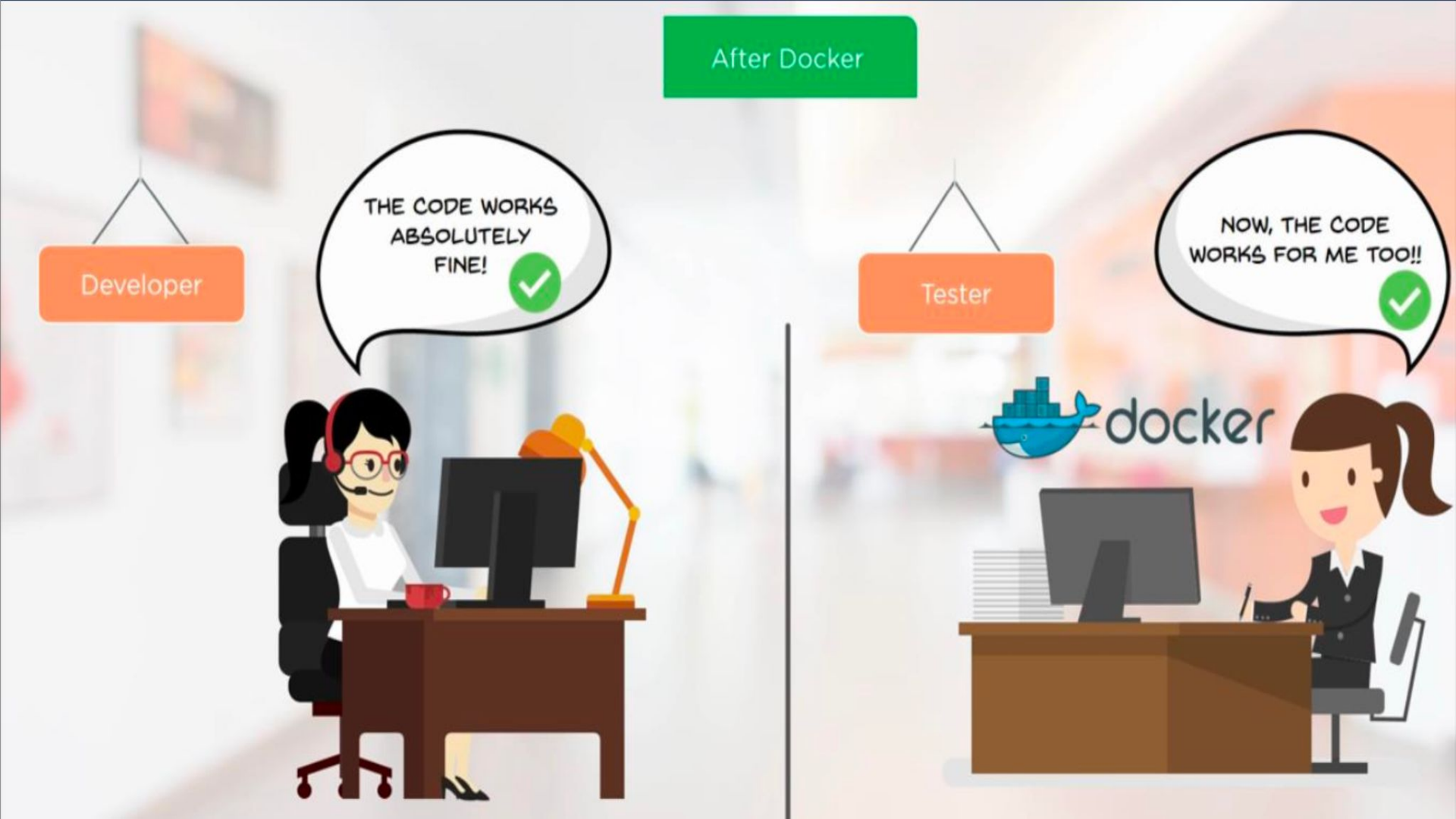
Before Docker

A developer sends code to a tester but it doesn't run on the tester's system due to various dependency issues, however it works fine on the developer's end.

After Docker

As the tester and developer now have the same system running on Docker container, they both are able to run the application in the Docker environment without having to face differences in dependencies issue as before.

Before Docker

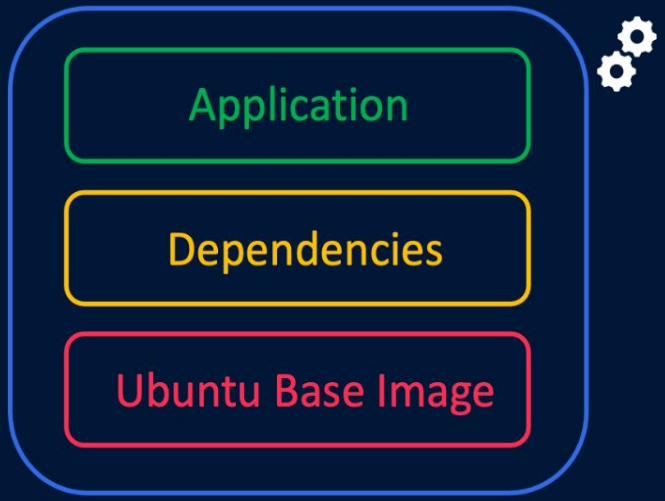

After Docker
Developer
THE CODE WORKS ABSOLUTELY FINE!
Tester
NOW, THE CODE WORKS FOR ME TOO!!
docker

Docker

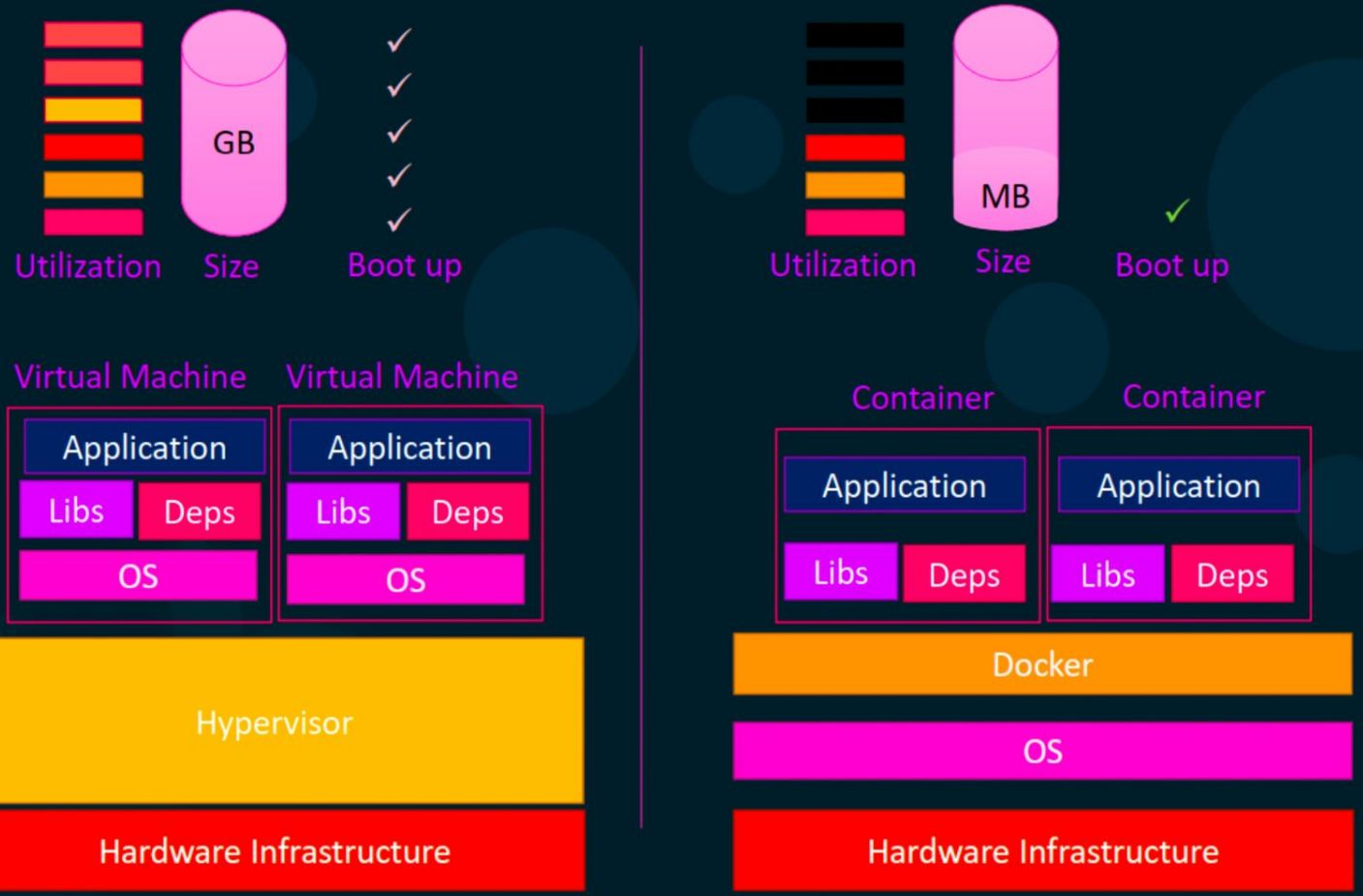
Docker is a software development tool and a virtualization technology that makes it easy to develop, deploy, and manage applications by using containers.

Container refers to a lightweight, stand-alone, executable package of a piece of software that contains all the libraries, configuration files, dependencies, and other necessary parts to operate the application.

Ex: Ubuntu + Python + Dependencies

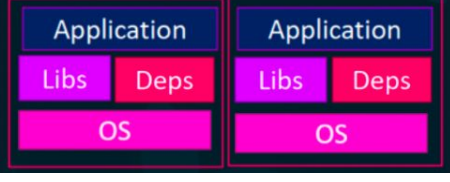


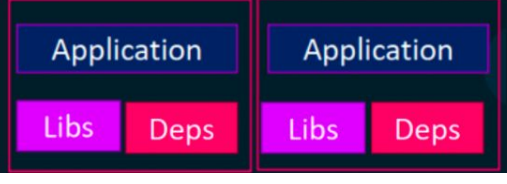
VMs vs Docker Containers



VMs vs Docker Containers

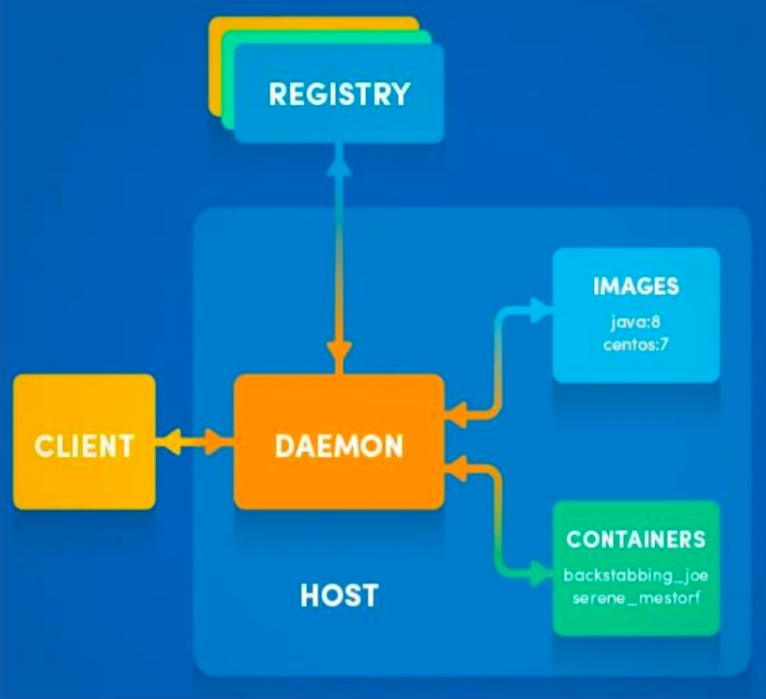
Virtual Machine	Docker Container
Hardware-level process isolation	OS level process isolation
Each VM has a separate OS	Each container can share OS
Boots in minutes	Boots in seconds
VMs are of few GBs	Containers are lightweight (KBs/MBs)
Ready-made VMs are difficult to find	Pre-built docker containers are easily available
VMs can move to new host easily	Containers are destroyed and re-created rather than moving
Creating VM takes a relatively longer time	Containers can be created in seconds
More resource usage	Less resource usage



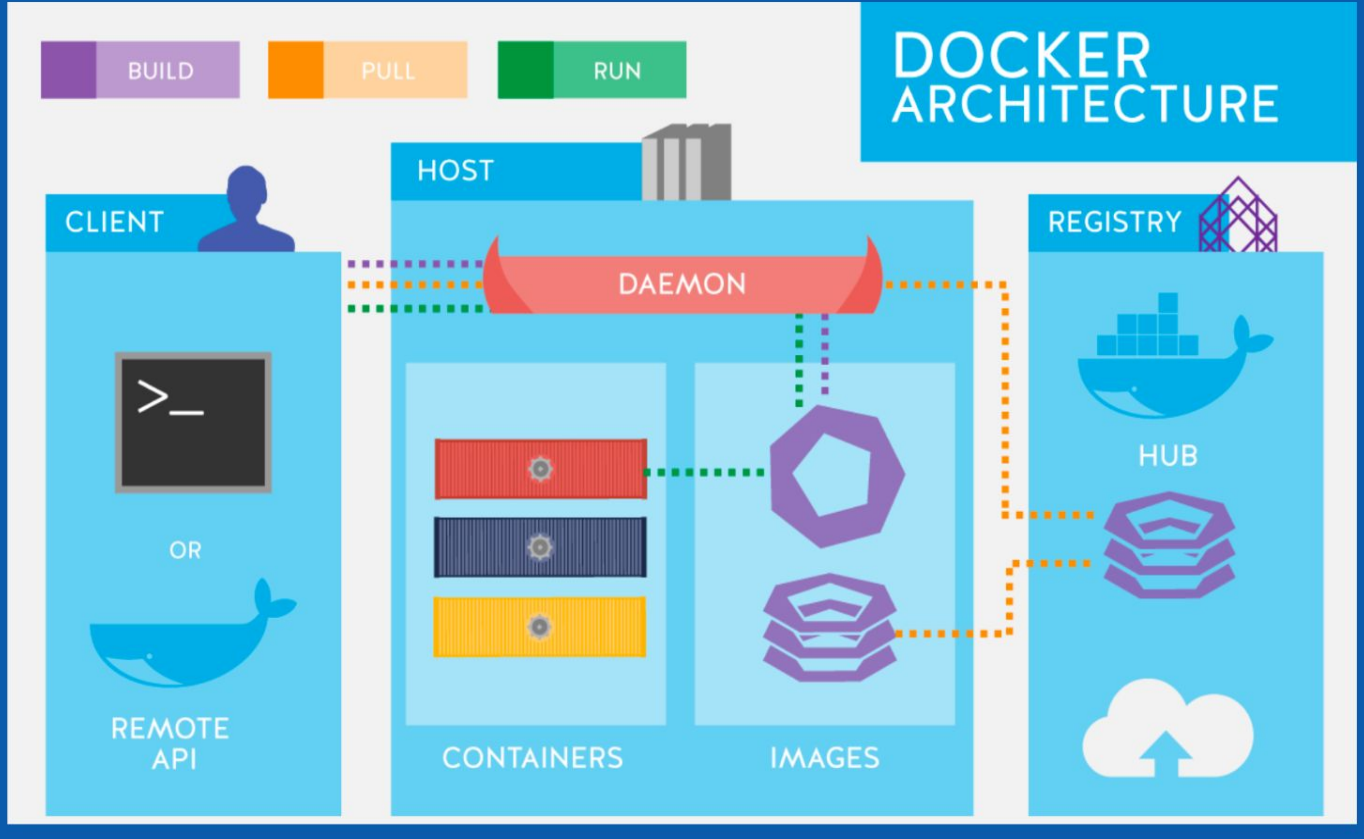


Docker Architecture

- Docker uses a client-server architecture.
- Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.
- Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.
- For a virtual communication between CLI client and Docker daemon, a REST API is used



Docker Architecture



Docker VS Container

What is Docker Image

What Is Docker?

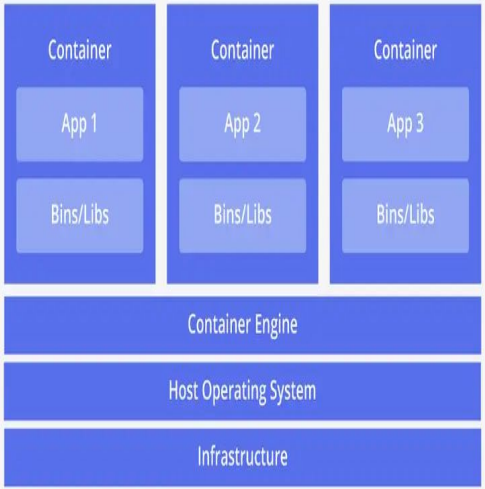
- Docker is similar to a mammoth cargo ship that has the ability to hold big boxes (containers) having their distinct objectives and id.
- These boxes contain items that are unique and are required to make that particular box useful for the company/person who has asked for it to be shipped.
- These items are first manufactured in factories that have templates to reproduce such items. These items (images) are packaged into the boxes (dockerized/containerized) in such a manner that they are useful for someone who has asked them to be shipped.
- If you are finding this analogy hard to digest, let me define it by keeping its technical aesthetics in place
- Docker is open-source, virtualization software created to make developer's life easy. It is a kind of PaaS (platform-as-a-service) product whose core objective is to isolate virtual environments to deploy, build, and test applications that are usually incompatible or not meant to work with the current OS.

Docker Images

- It is a kind of ready-to-use software read-only template crafted with source codes, libraries, external dependencies, tools, and other miscellaneous files that are needed for any software application to run successfully on any platform or OS.
- The developer community also likes to call it Snapshots, representing the app and its virtual environment at a specific point in time.
- This docker snapshot is a perfect recipe for developers to build test and deploy the desired app swiftly

What is Docker Container?

- Now that you have got hold of the docker image concept it will be apt now to describe the docker container.
- As we discussed, docker images are nothing but a read-only template that can't be executed by themselves and cannot run or start. If that is the limitation how can one make a real value out of it?
- Well, the answer lies in the concept of the Docker container.
- A container is nothing but a box that has the ability to run the docker image templates. The moment you create a container using those immutable images you essentially end up creating a read-write copy of that filesystem (docker image) inside the given container. This adds a container layer which helps you to modify the entire copy of the given Docker image.
- A container can also be considered as a cohesive software unit that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.



Why Docker Containers Are Useful?

All the docker images become docker containers when they run the on Docker Engine and these containers are popular amongst developers and organizations of all shapes and sizes because it is-

Standardized: Docker created the industry standard for containers, so they could be portable anywhere.

Lightweight: Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs.

Secure: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry.

Now that we have covered the fundamentals of docker images and containers, we have got the right set of foundations to compare them wisely

How do Docker Images Differ from Containers?

To be true if you see the docker ecosystem in its entirety you will agree that both of these docker objects are an inclusive part of it. Without docker images, you can imagine docker containers at all and without docker containers, docker images are like completely meaningless and orphans.

So is it wise to compare them, I may say no but still to clarify the concepts we can do so only on the basis of their roles and responsibilities

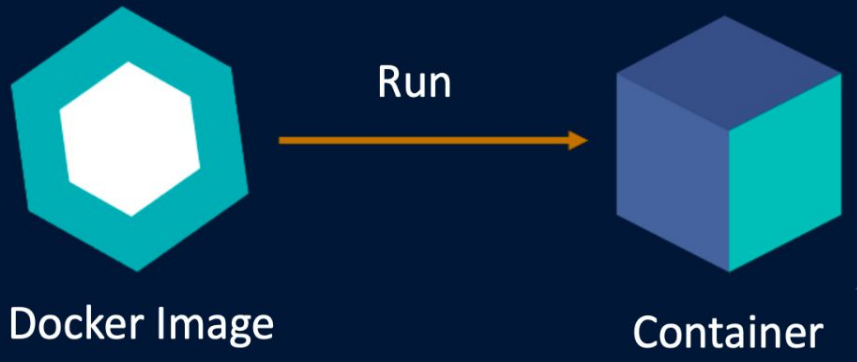
Docker Image vs Containers

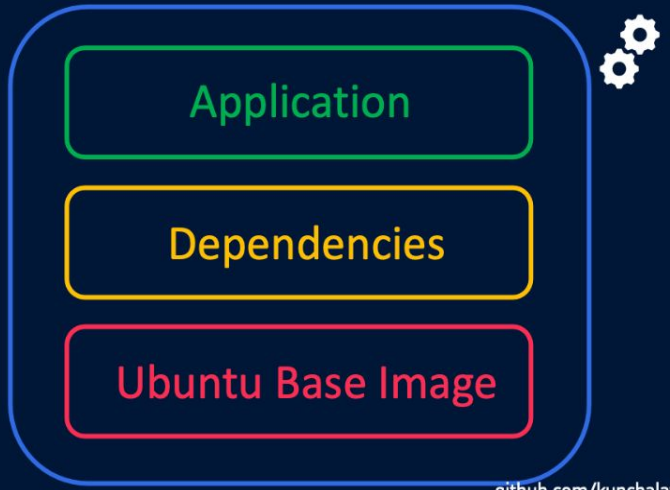
- The key difference between a Docker image Vs a container is that a Docker image is a read-only immutable template that defines how a container will be realized. A Docker container is a runtime instance of a Docker image that gets created when the $ docker run command is implemented.
- Before the docker container can even exist docker templates/images are built using $ docker build CLI.
- Docker image templates can exist in isolation but containers can't exist without images.
- So docker image is an integral part of containers that differs only because of their objectives which we have already covered.
- Docker images can’t be paused or started but a Docker container is a run time instance that can be started or paused.

Docker Images & Containers

- **Docker Image:** Docker image can be compared to a template that is used to create Docker containers. These are read-only templates that contains application binaries and dependencies. Docker images are stored in the Docker Registry.

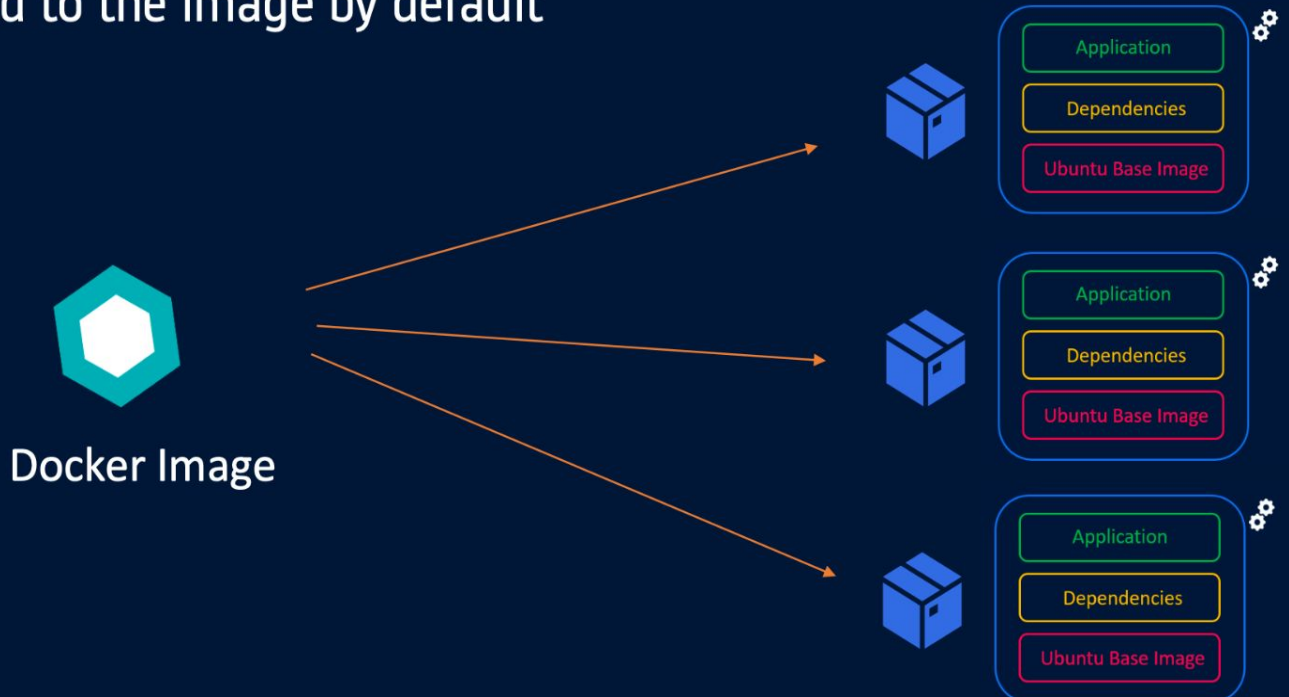
- **Docker Container:** Docker container is a running instance of a Docker image as they hold the entire package needed to run the application.





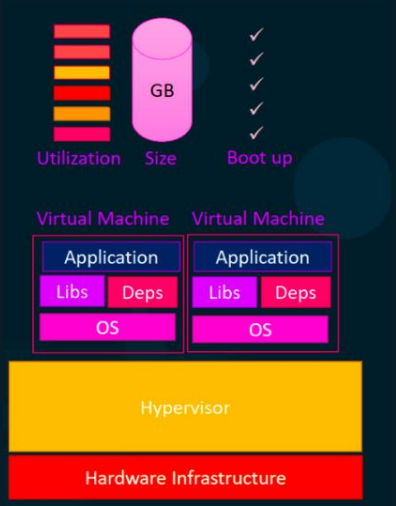
Docker Images & Containers

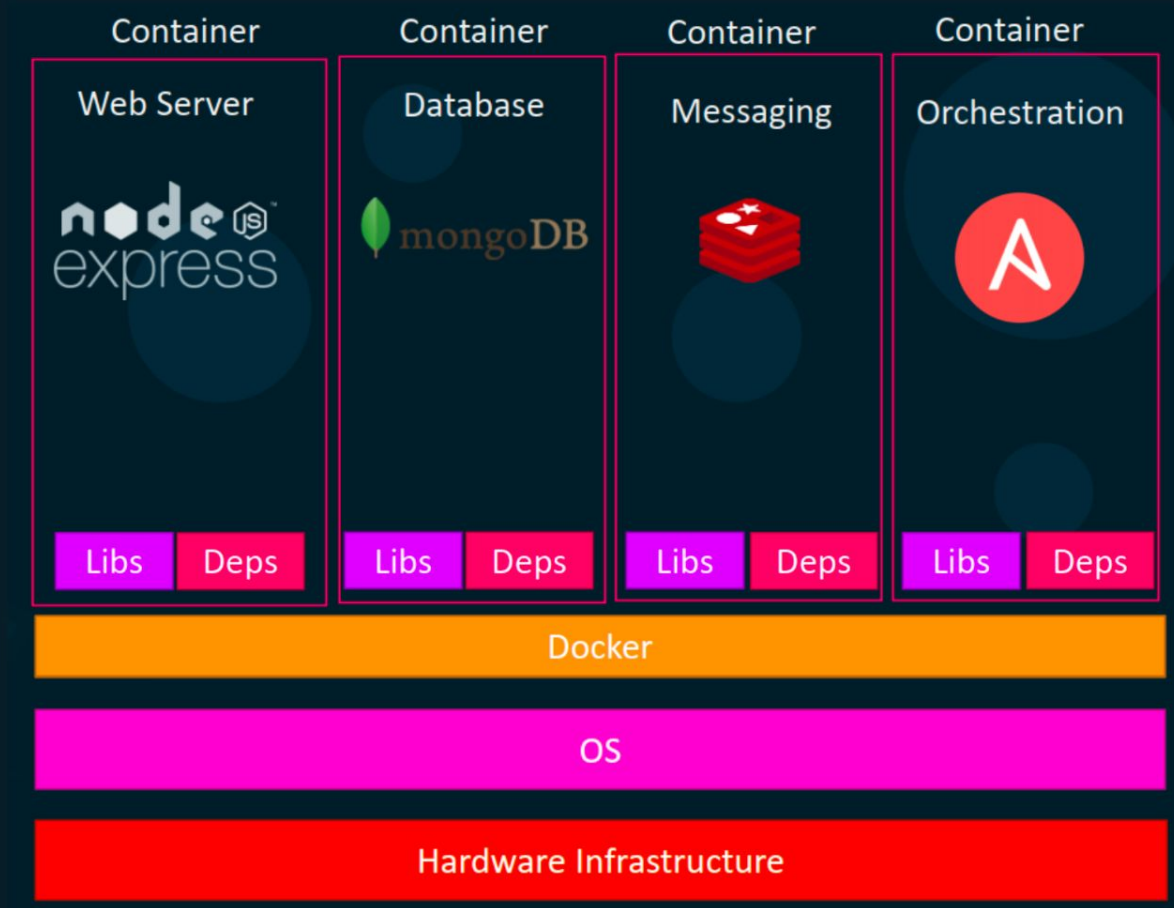
- We can run any number of containers based out of an image and Docker makes sure that each container created has a unique name in the namespace.
- Docker image is a read-only template. Changes made in containers won't be saved to the image by default



Docker Images & Containers

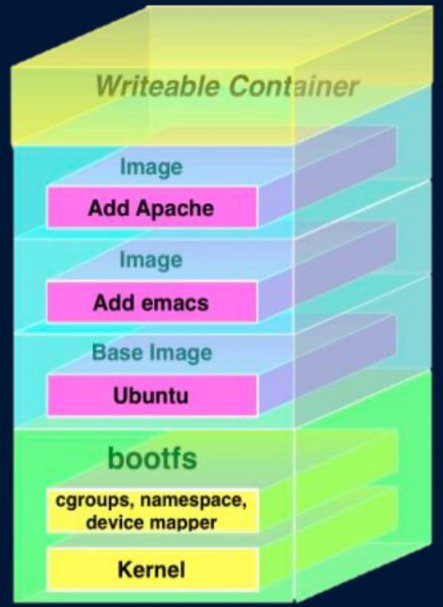
Containers run each service with its own dependencies in separate containers





Docker Volumes

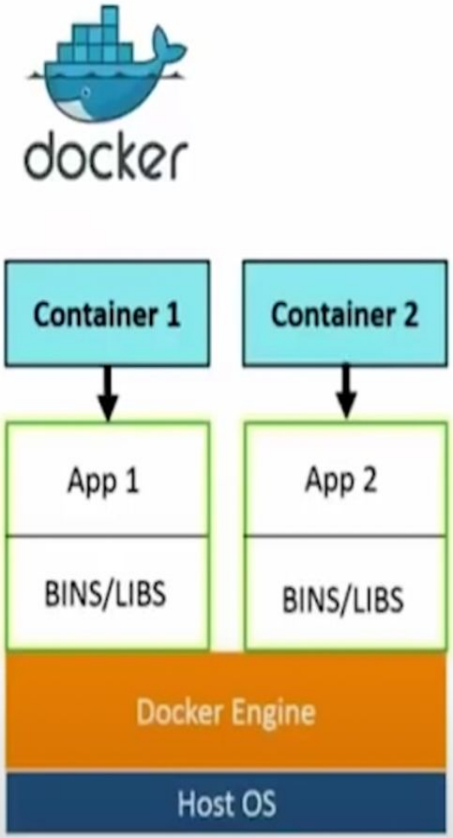
- Images are a series of read-only layers
- A container is merely an instantiation of those read-only layers with a single read-write layer on top.
- Any file changes that are made within a container are reflected as a copy of modified data from the read-only layer.
- The version in the read-write layer hides the underlying file but does not remove it.
- When deleting a container, the read-write layer containing the changes are destroyed and gone forever!
- In order to persist these changes we use docker volumes

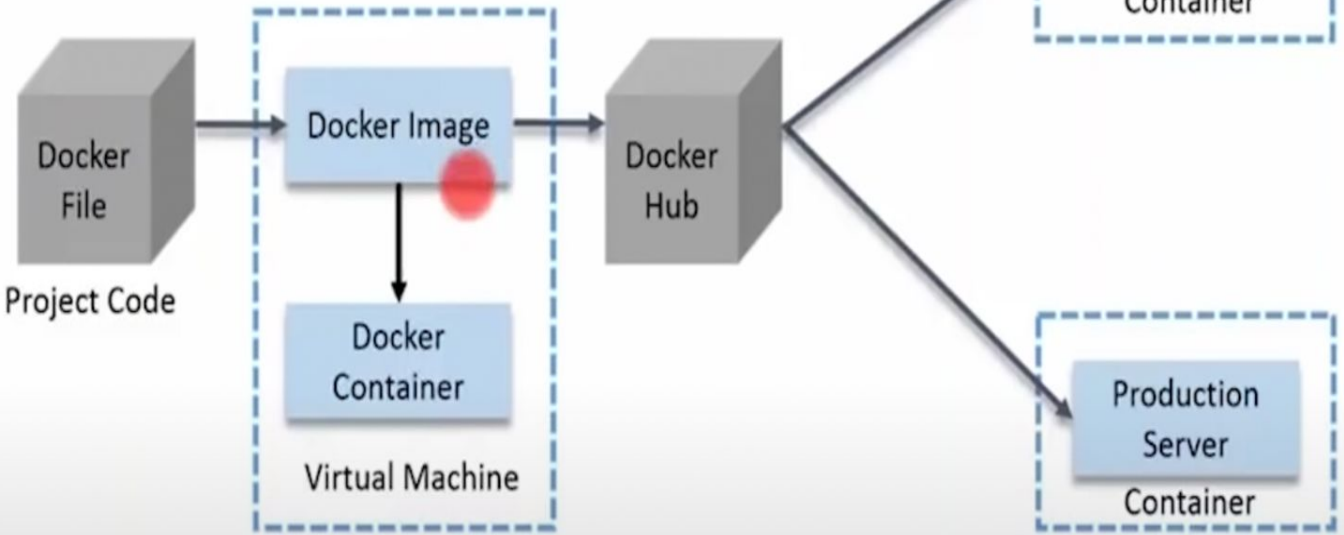


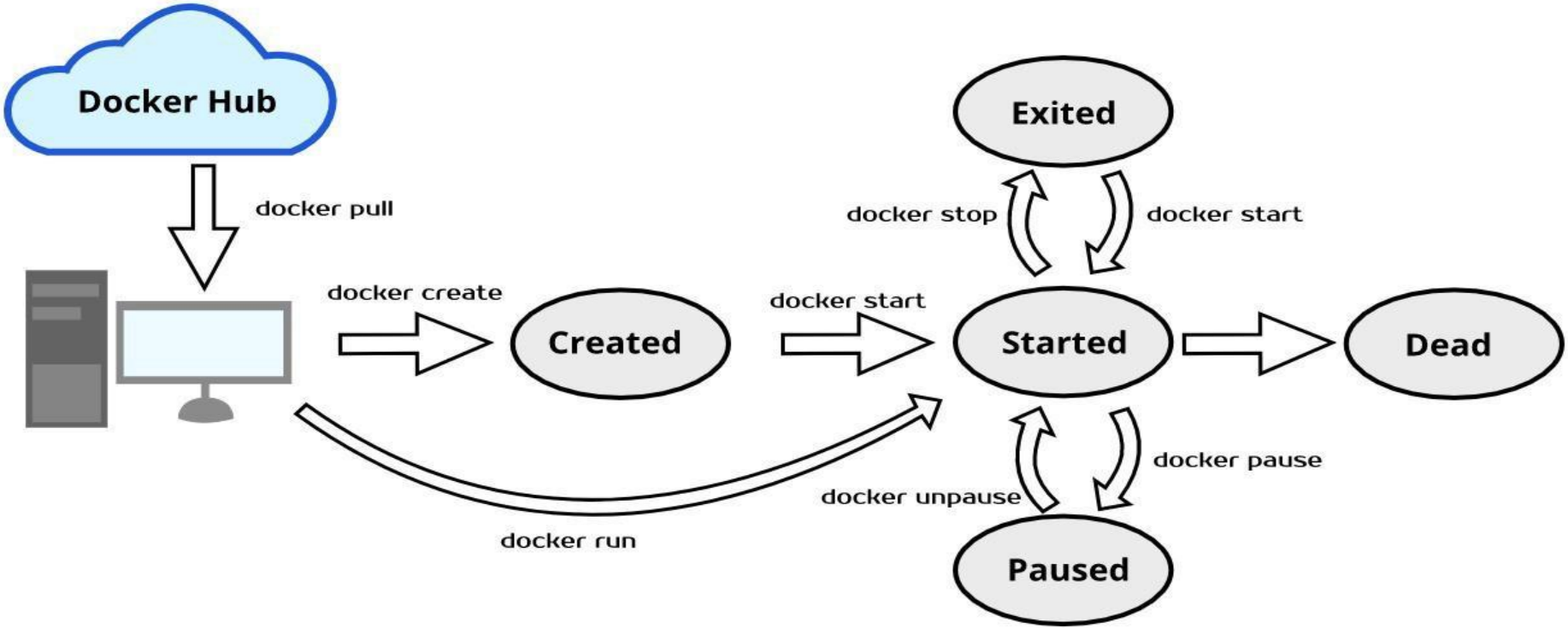
Advantages:

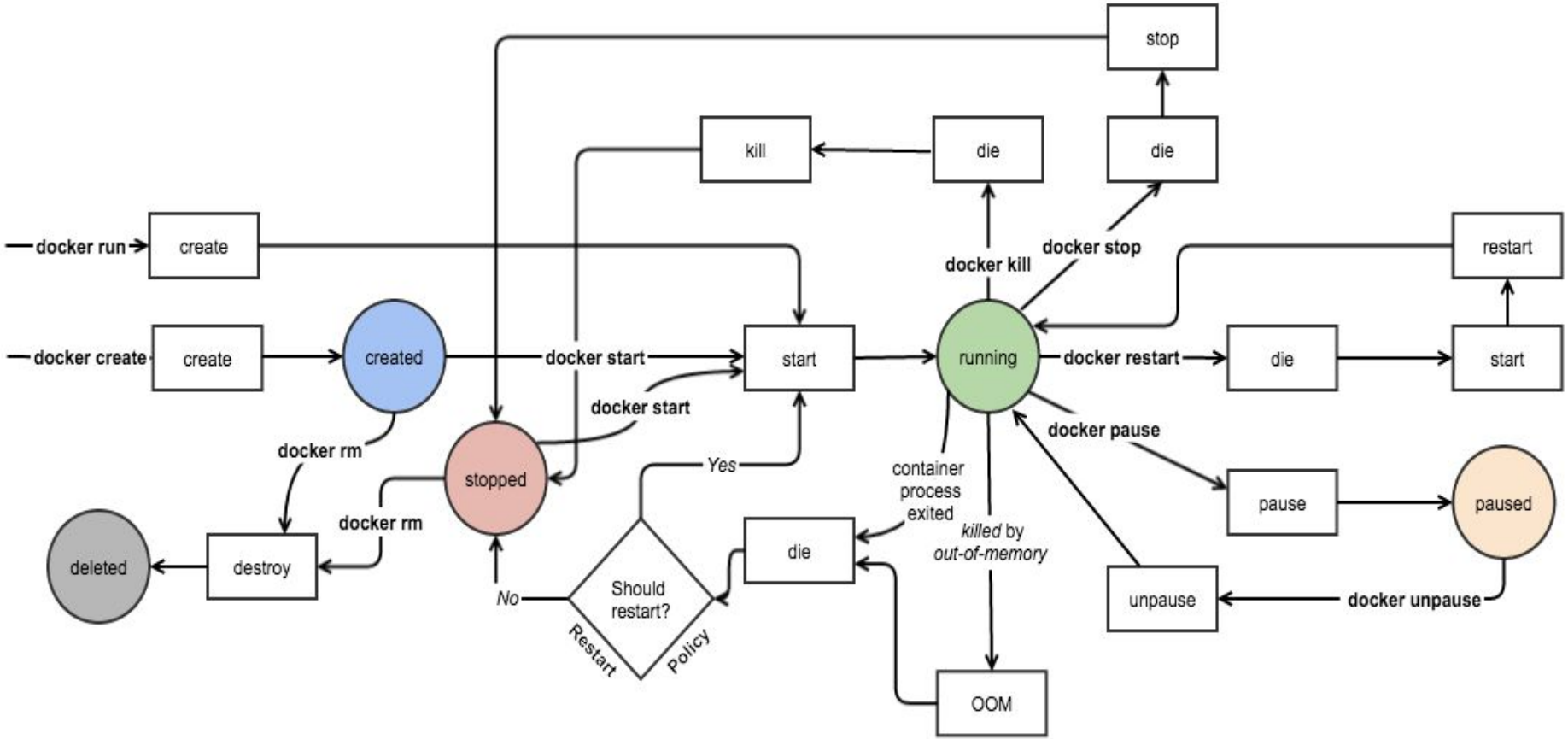
1. To keep data around when a container is removed
2. To share data between the host filesystem and the Docker container

Containerization






Docker Hub
docker pull
docker create
Created
docker start
Started
Exited
docker stop
docker start
Dead
docker pause
docker unpause
Paused
docker run



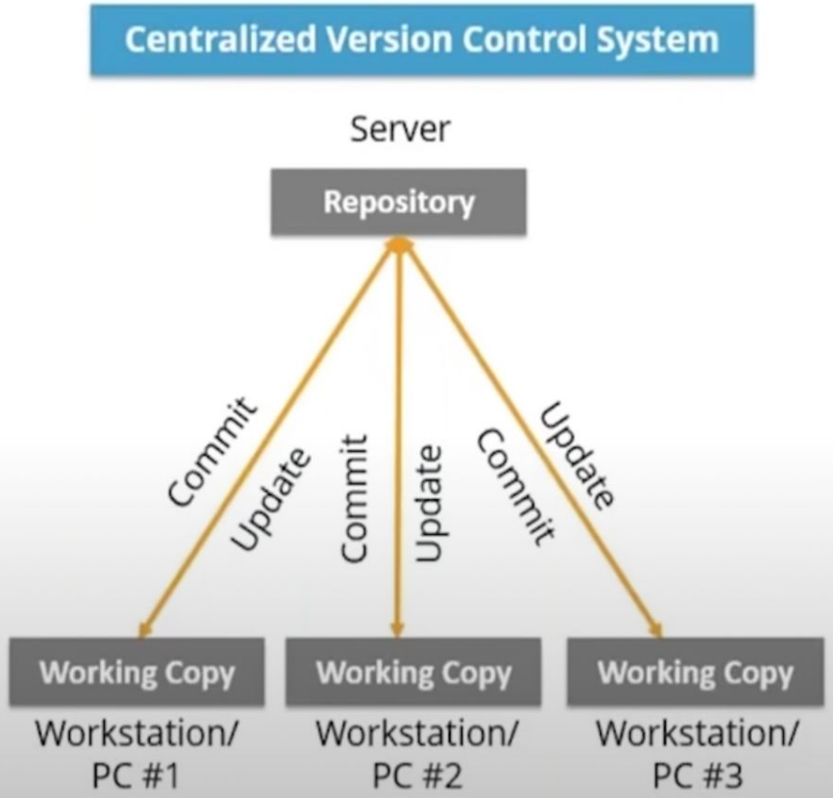
upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

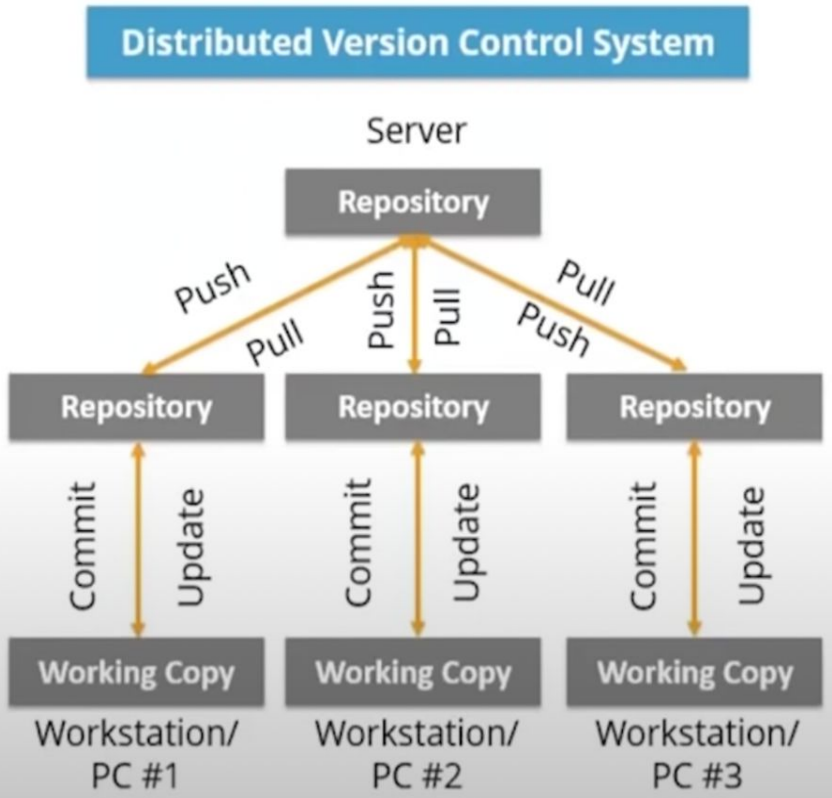
DevOps Tools
<Git and Github>

Software Configuration Management

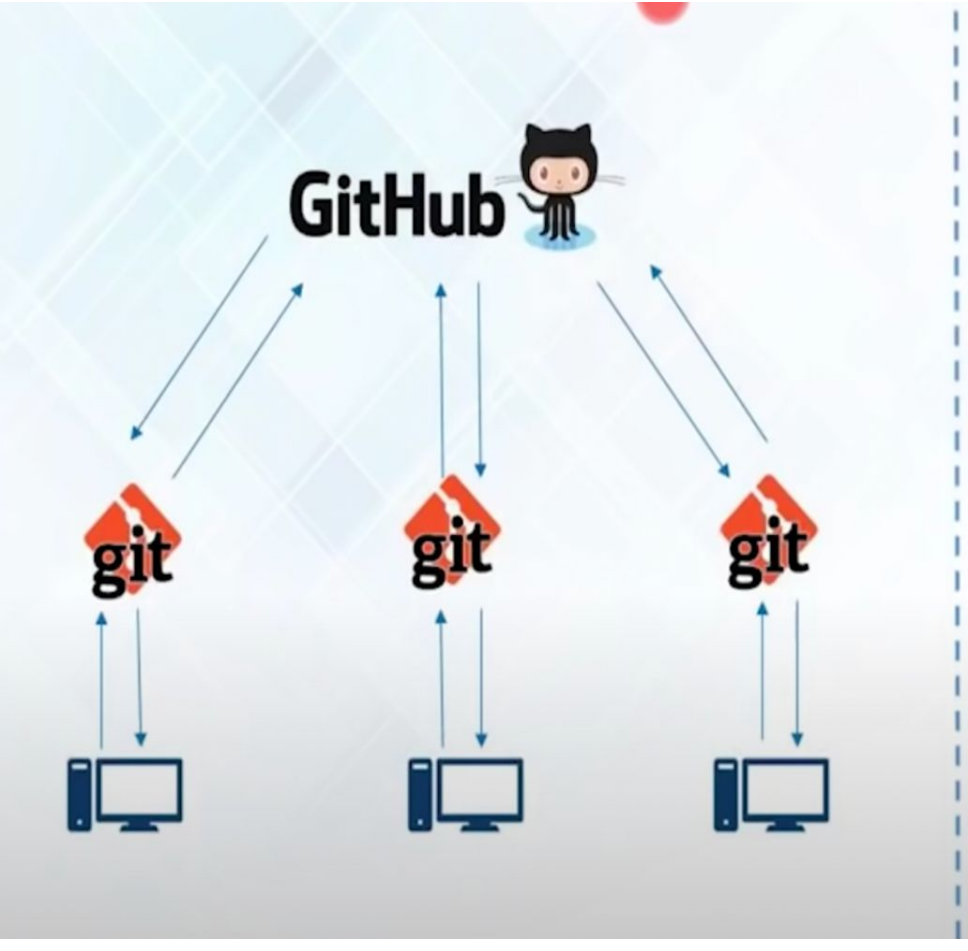
upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

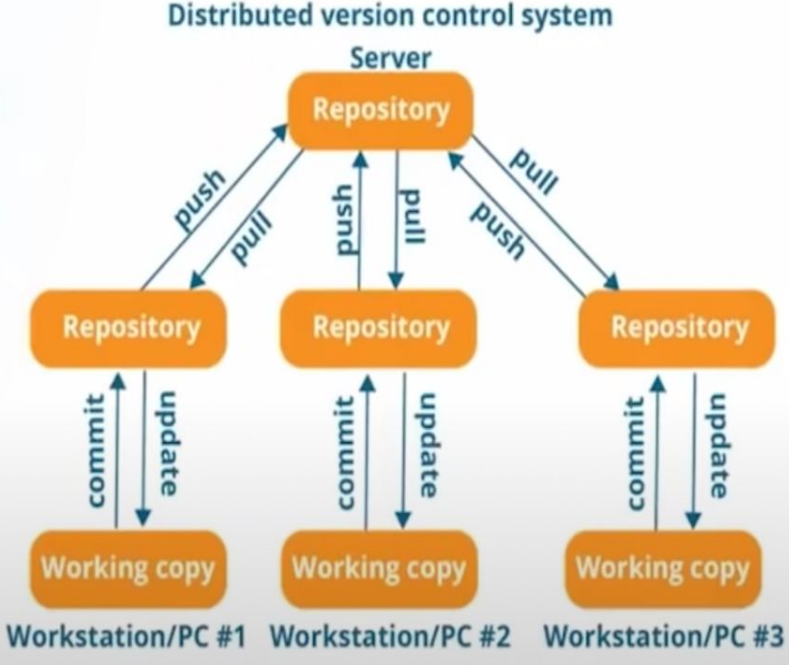
The management of changes to documents, computer programs, large websites and other collection of information





Git and GitHub





Software Configuration Management

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

- Created by Linus Torvalds, creator of Linux, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel

- Goals of Git:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

  - *(A "git" is a cranky old man. Linus meant himself.)*

Software Configuration Management

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

Git

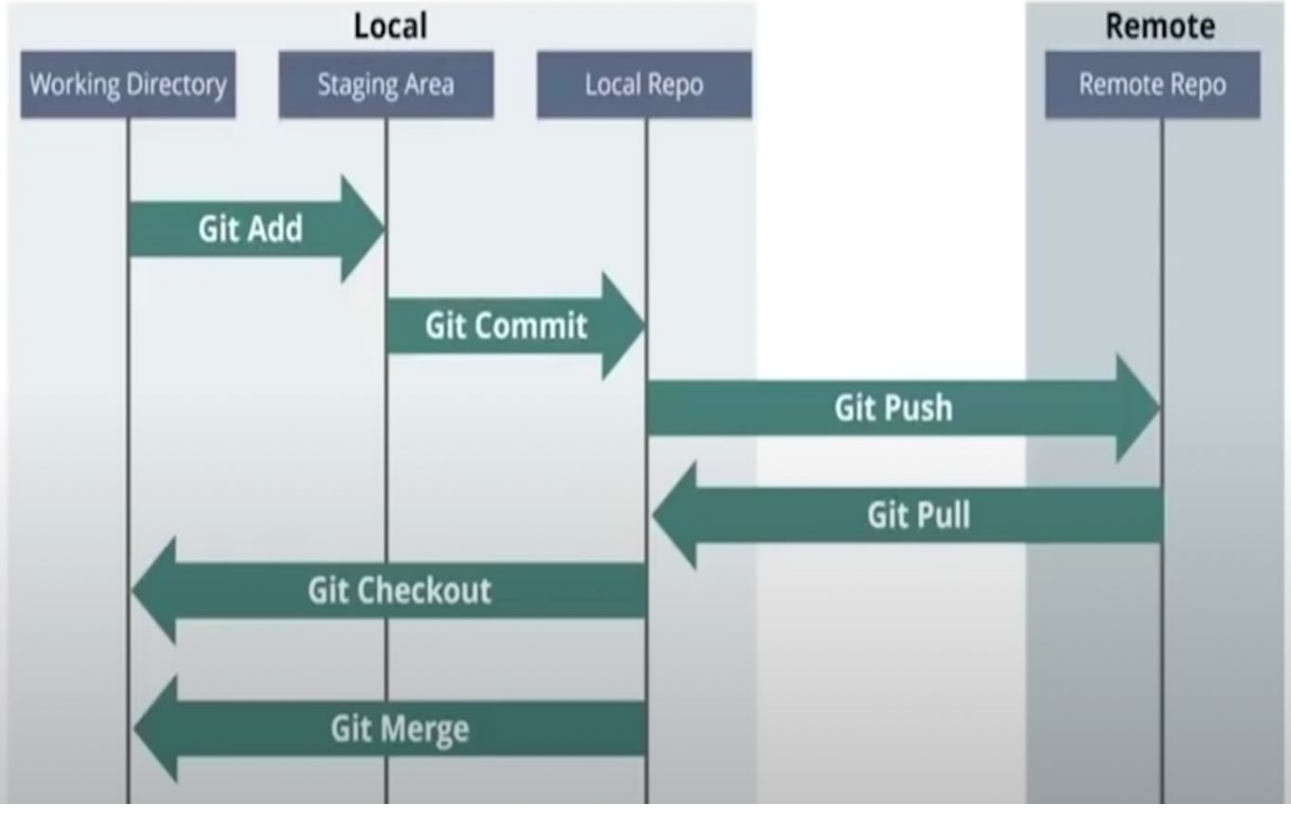
- Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

- **Git** is the most commonly **used** version control system. **Git** tracks the changes done to files, so the record of what has been done is available, and also once can revert to specific versions should one ever need to. **Git** also makes collaboration easier, allowing changes by multiple people to all be merged into one source

Your Daily Tasks

- **Create** things
- **Save** things
- **Edit** things
- Save the thing **again**

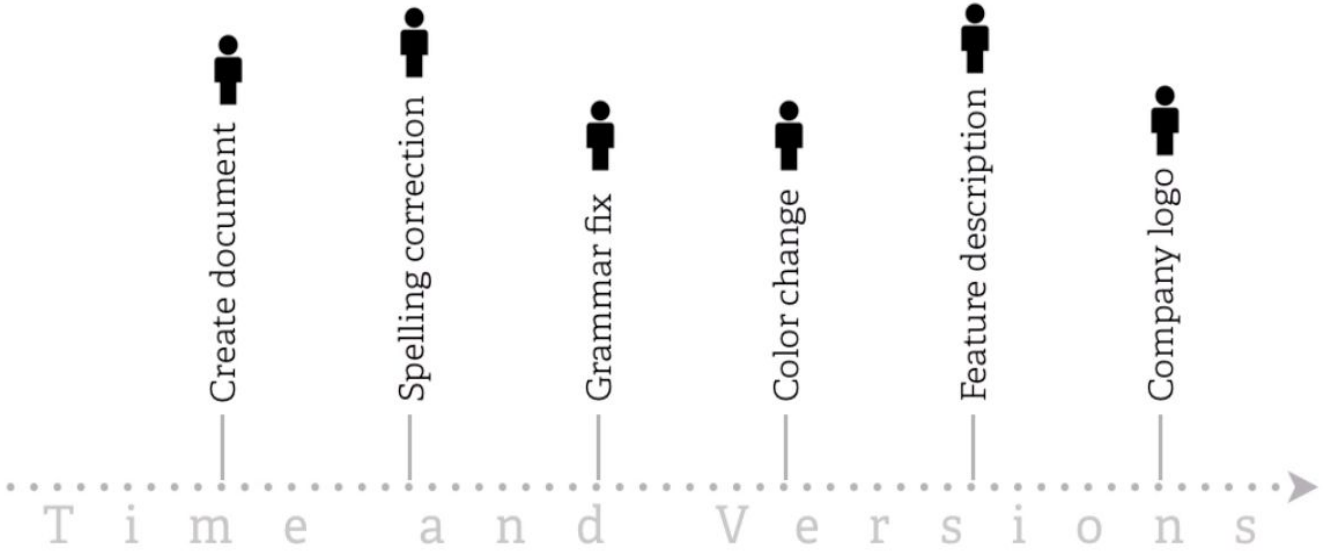
Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software



Software Configuration Management

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

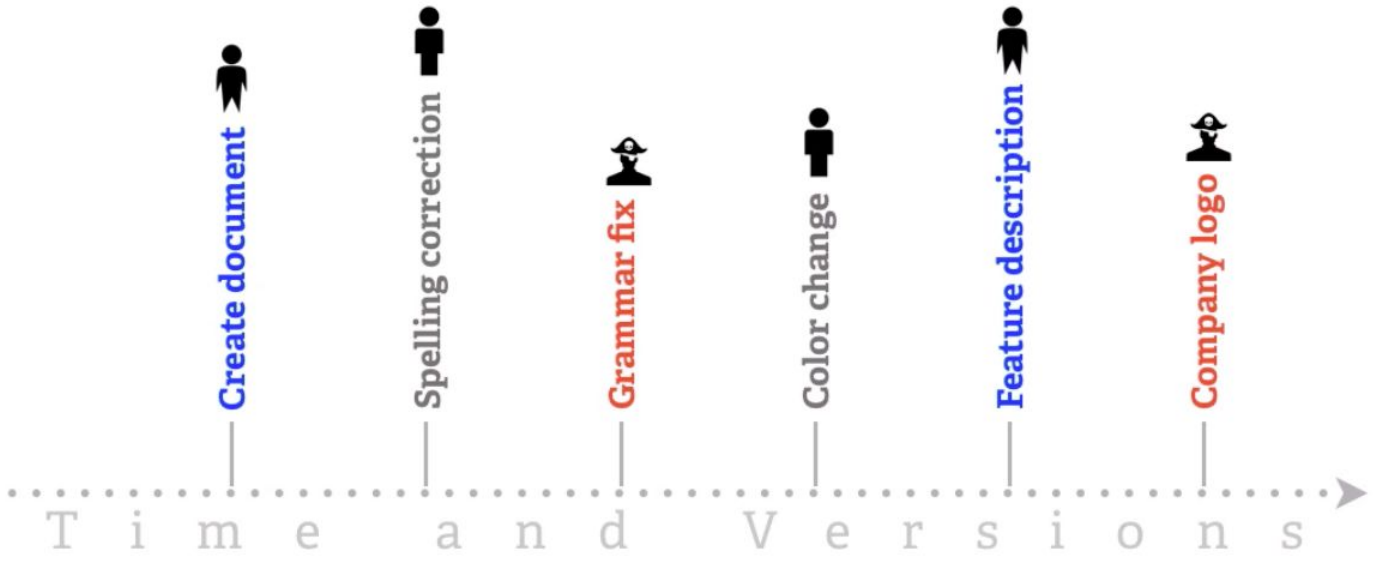
History Tracking



Software Configuration Management

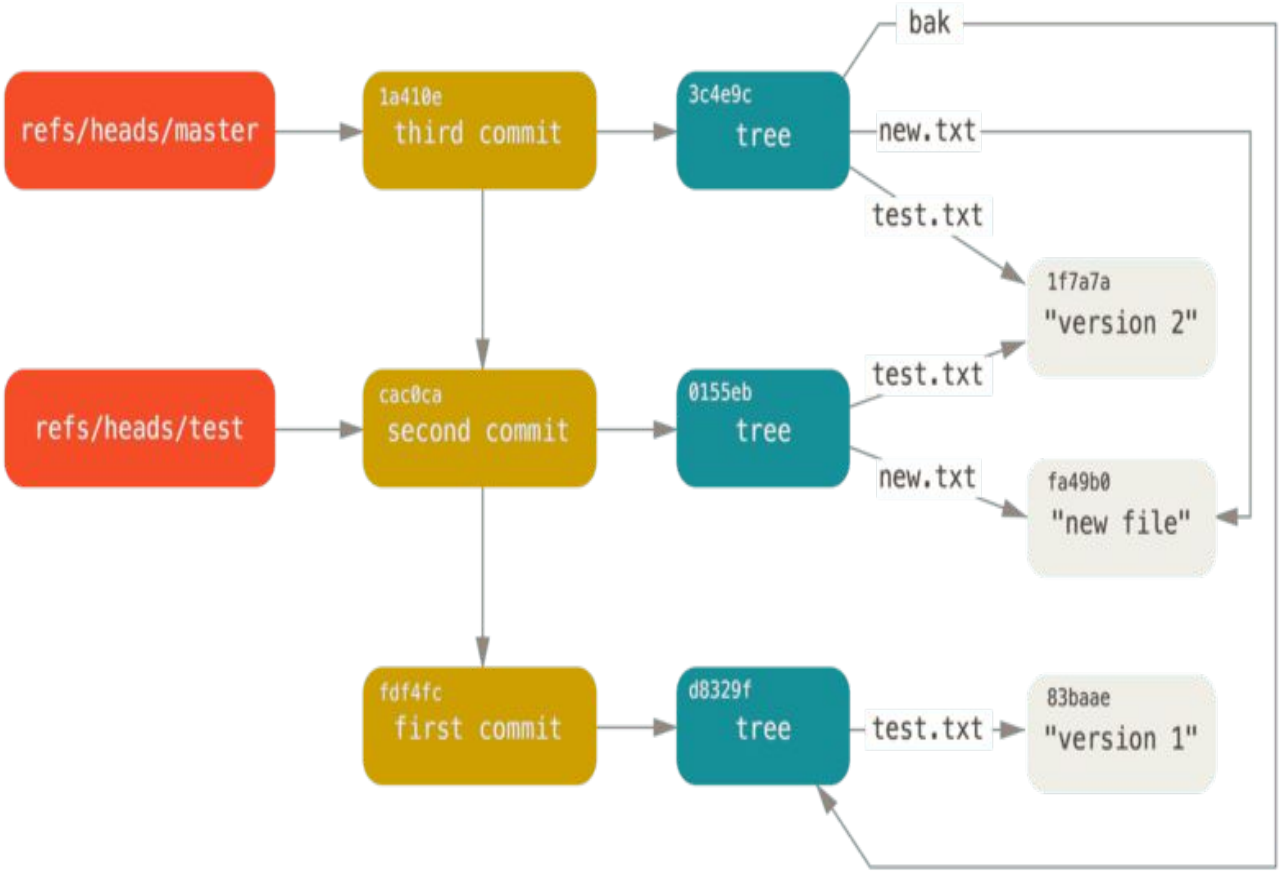
upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

Collaborative History Tracking



Software Configuration Management

- **Git** is a fast and modern implementation of version control
- **Git** provides a history of content changes.
- Git facilitates **collaborative changes** to files
- Git is easy to use for any type of **knowledge worker.**

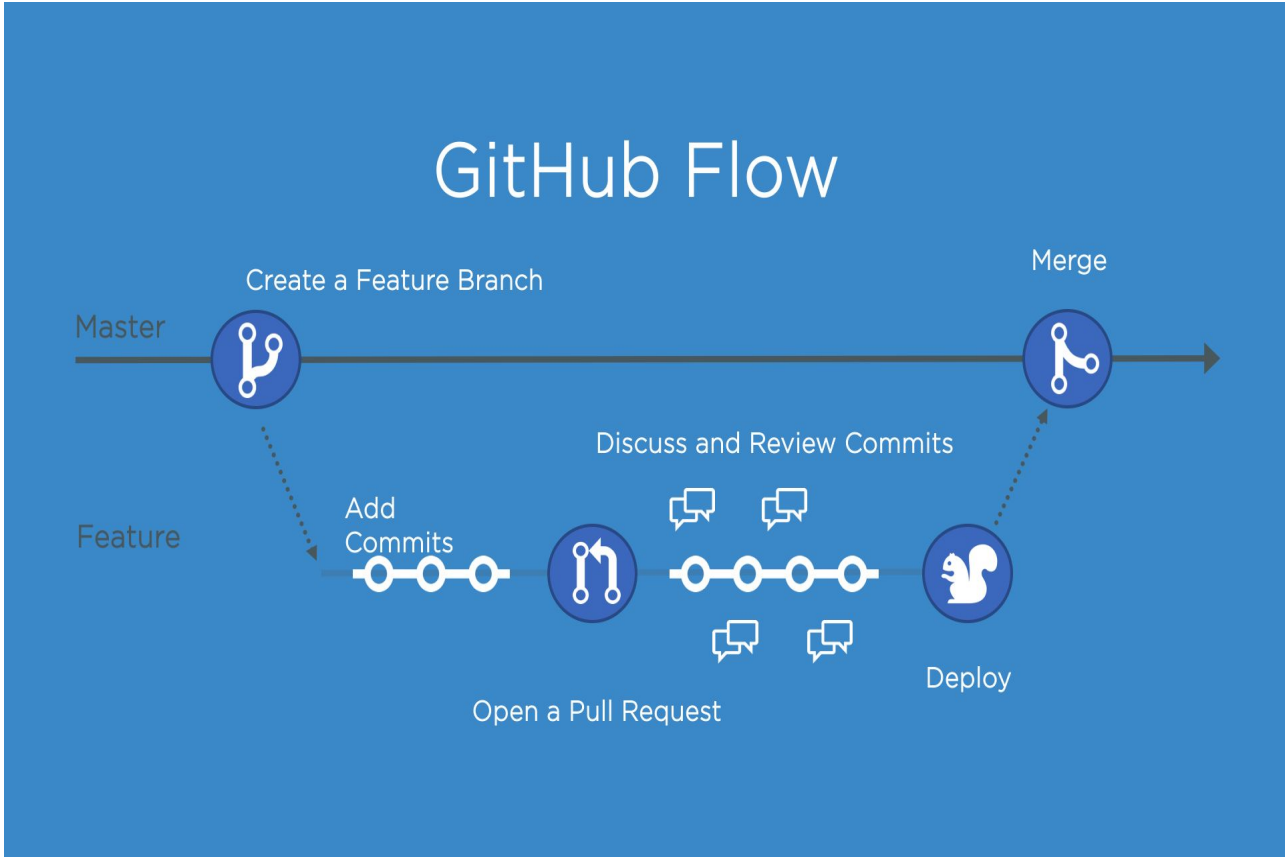


Software Configuration Management

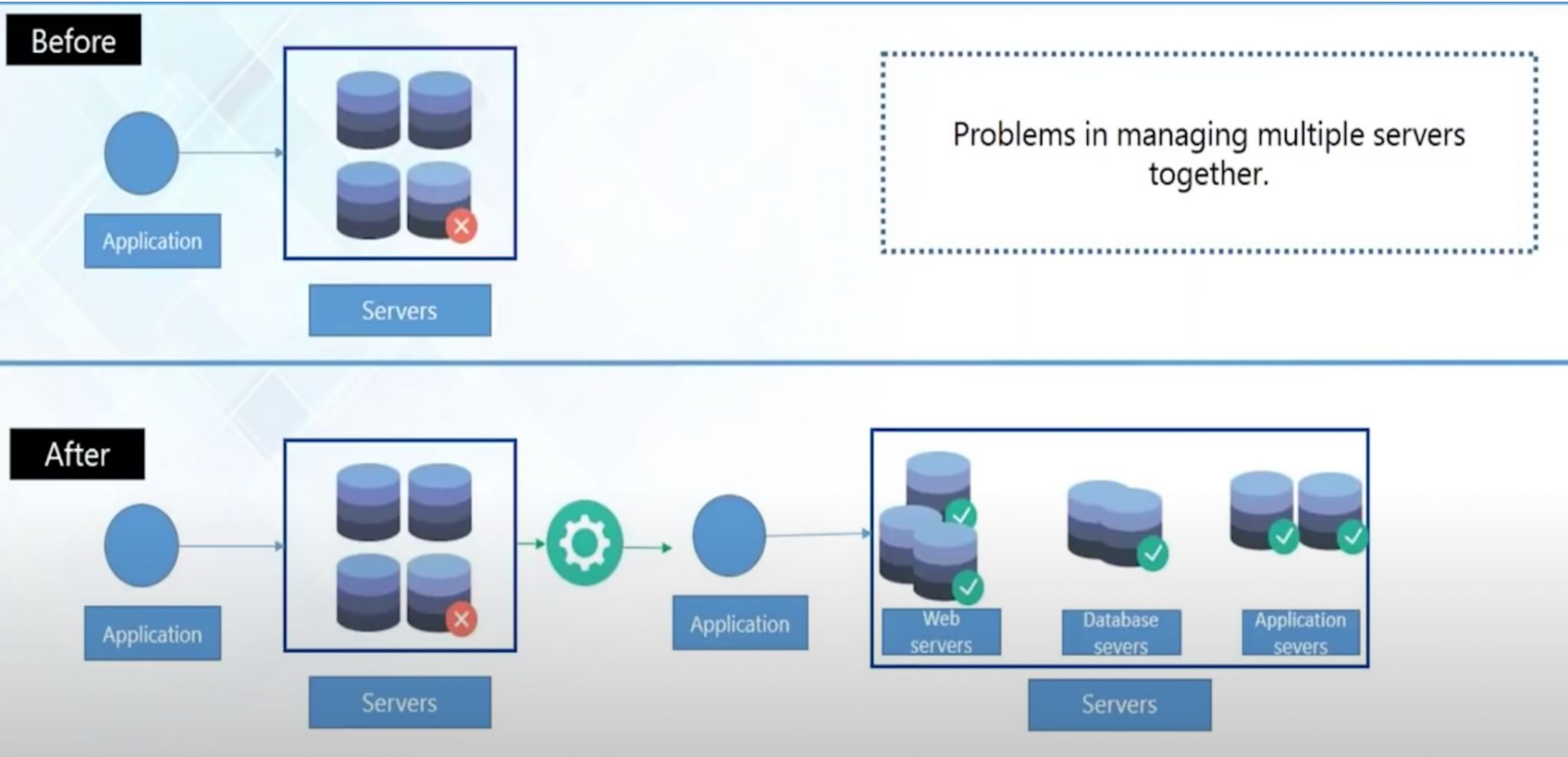
upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

GitHub

- GitHub, Inc. is a provider of Internet hosting for software development and version control using Git.
- It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features.
- It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project



Before and after configuration Management

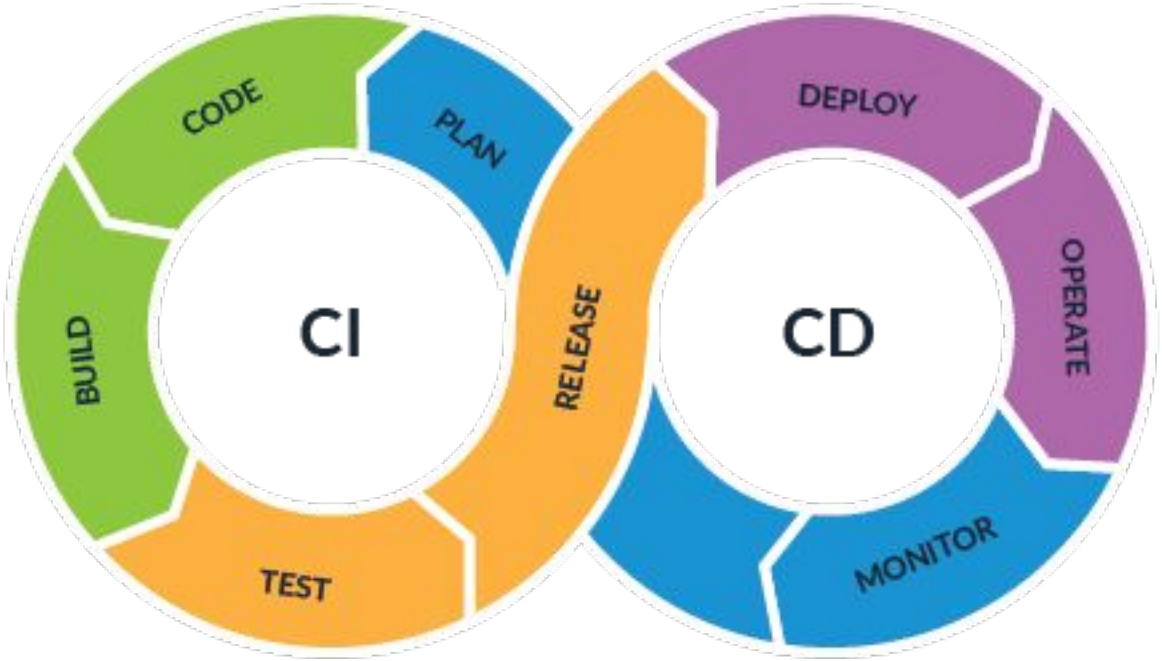


upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

DevOps Tools
< CI/CD>

Continuous Integration and Continuous Deployment

In software engineering, CI/CD or CICD is the combined practices of continuous integration and either continuous delivery or continuous deployment. CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications. - Wikipedia



What is CI/CD

What is CI/CD?

CI/CD is short for continuous integration/continuous deployment. CI and CD are related but have different definitions so let's take a look at each separately.

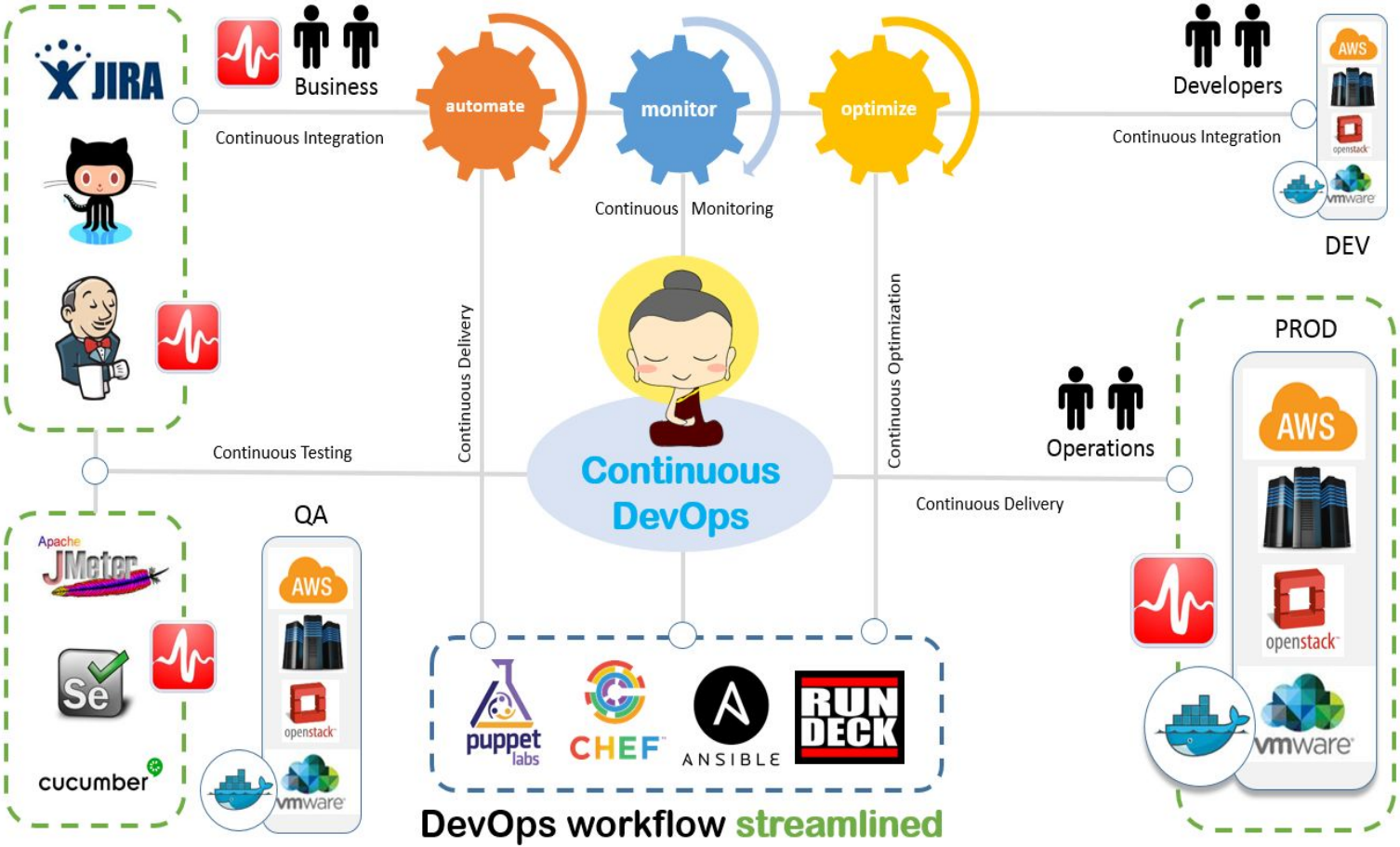
Continuous integration

Continuous integration is a development practice that involves developers on a team checking-in (integrating) their work frequently (usually multiple times a day). Once the work is checked-in, an automated build and testing process kicks off to see if it conflicts with existing code (these conflicts are called integration errors).

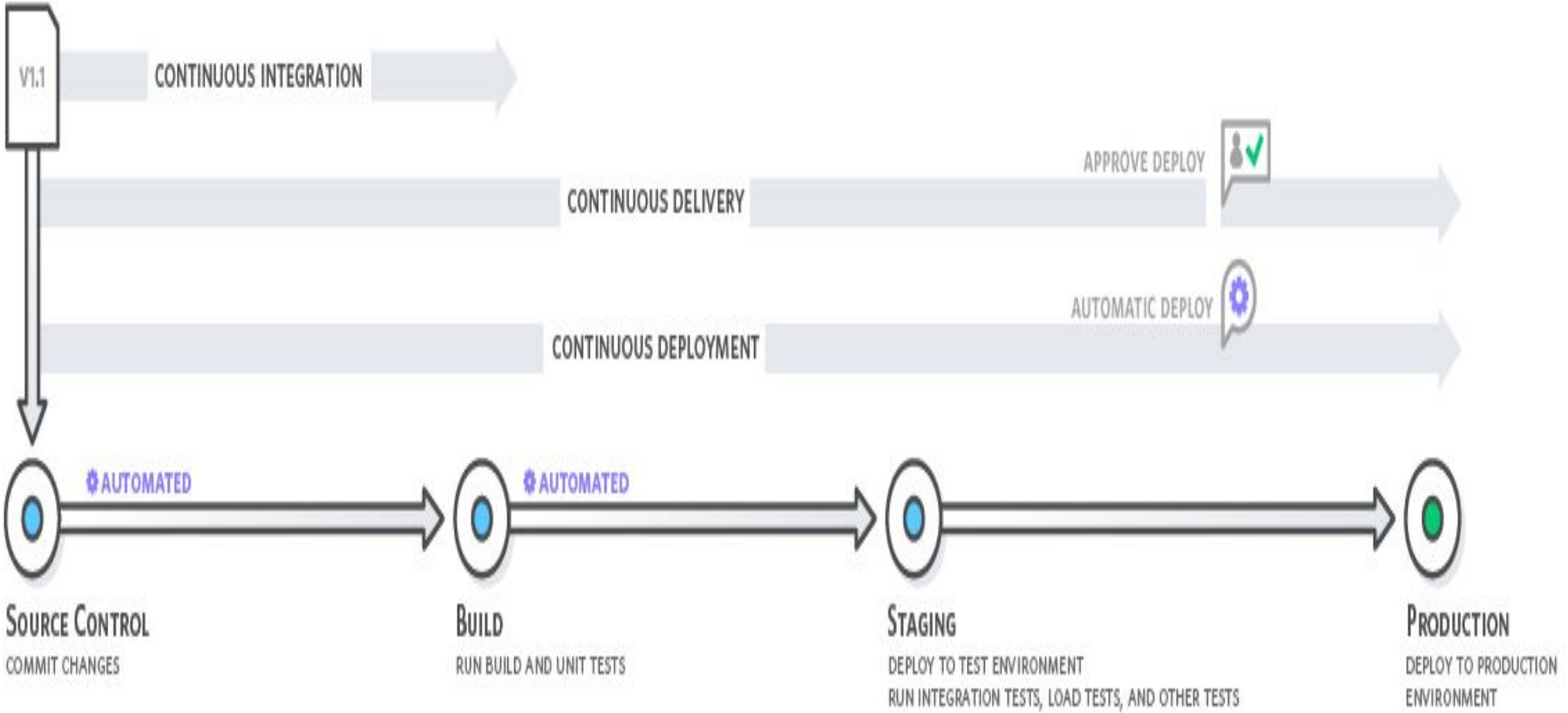
The CI process allows teams to quickly detect and correct integration problems and keep everyone's development code relatively up to date while they are working. Without CI and frequent check-ins, builds, and tests, code different developers are working on could get so out of sync, that productivity and code quality is negatively impacted. Why? Because with fewer check-ins and tests, a developer may work on code for days before they attempt to merge it with the main branch. If they do and there are conflicts, they now have to work through days – as opposed to hours – of their own and their teammates' code to resolve integration issues.

CI – What does it really mean?

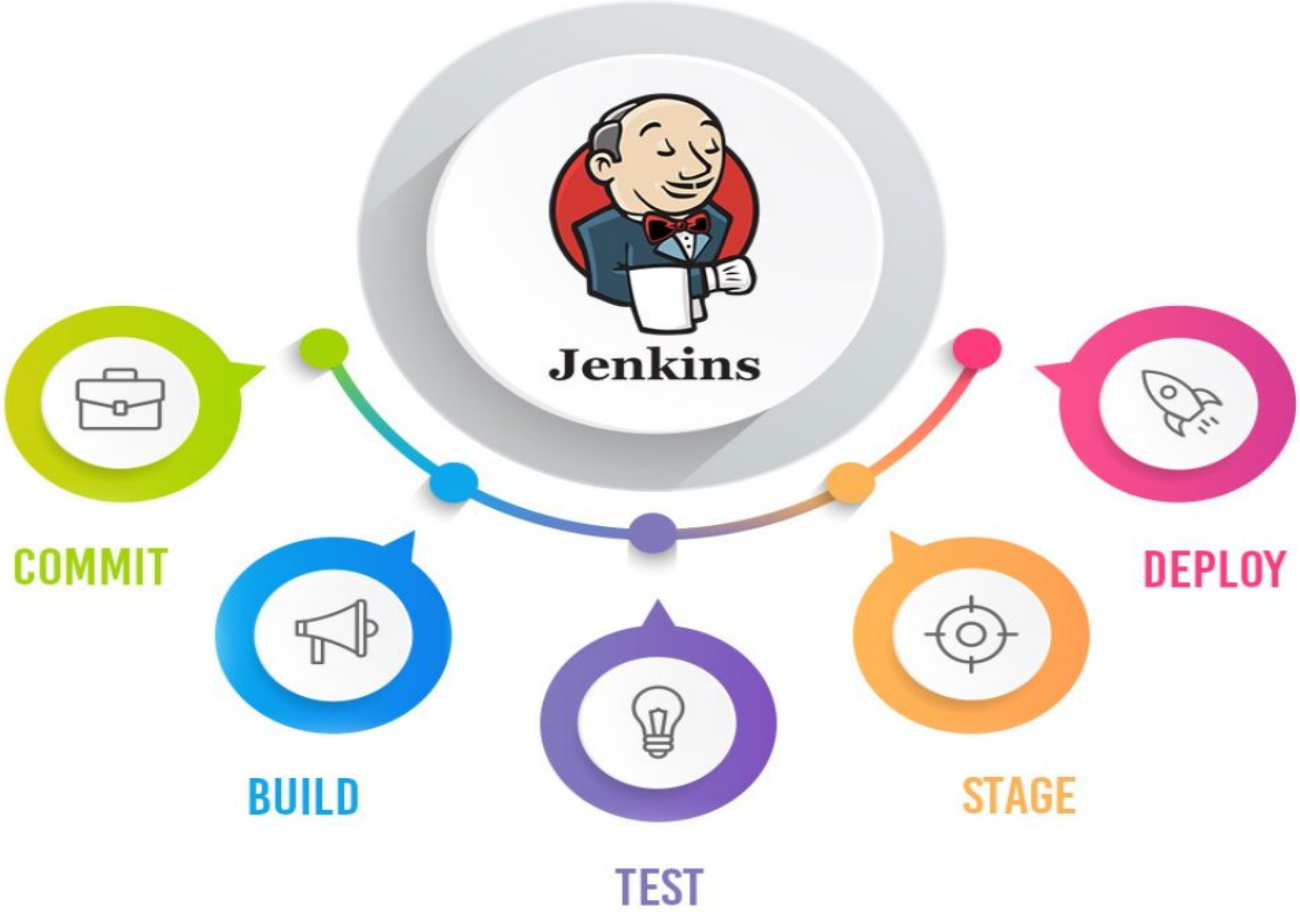
- At a regular frequency (ideally at every commit), the system is:
  - Integrated
    - All changes up until that point are combined into the project
  - Built
    - The code is compiled into an executable or package
  - Tested
    - Automated test suites are run
  - Archived
    - Versioned and stored so it can be distributed as is, if desired
  - Deployed
    - Loaded onto a system where the developers can interact with it


JIRA
Business
automate
monitor
optimize
Developers
AWS
openstack
vmware
DEV
Continuous Integration
Continuous Integration
Continuous Monitoring
Continuous Delivery
Continuous Optimization
Continuous Testing
Continuous DevOps
Operations
PROD
AWS
openstack
vmware
Continuous Delivery
QA
Apache JMeter
Se
cucumber
AWS
openstack
vmware
puppet labs
CHEF
ANSIBLE
RUN DECK
DevOps workflow streamlined

CI/CD/CD – Continuous Integration vs Delivery vs Deployment



CI/CD Jenkins



What is Jenkins?

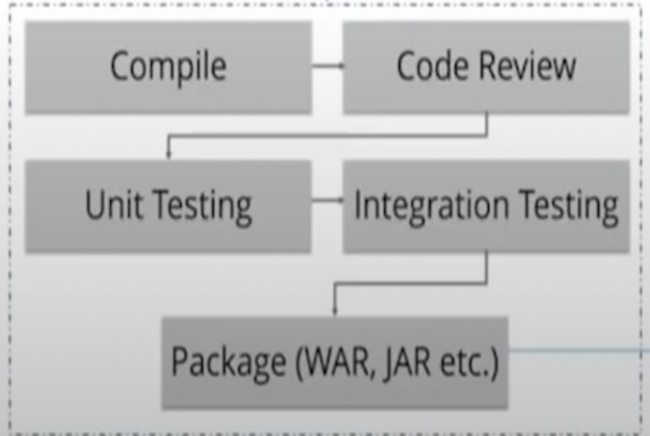
- Jenkins is an open source automation server written in Java.
- detects changes in Subversion/GIT..., performs tasks, repeatedly. (Build, Test, Deploy, Package, Integrate)
- A fork of the original Hudson project.
- Plug-in extensibility.
- Under development since 2005
- http://jenkins-ci.org/

Jenkins Server

Deploy the build application on the test server for UAT (User Acceptance Test)

Commit code to a shared repository

Build

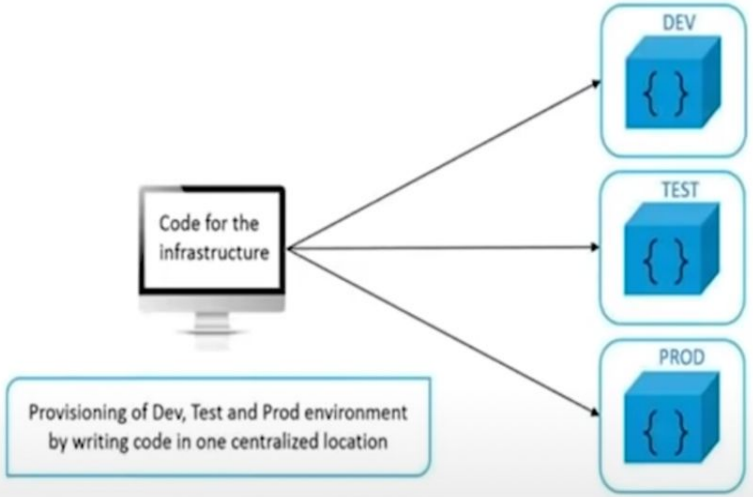


Deploy the build application on the prod server for release

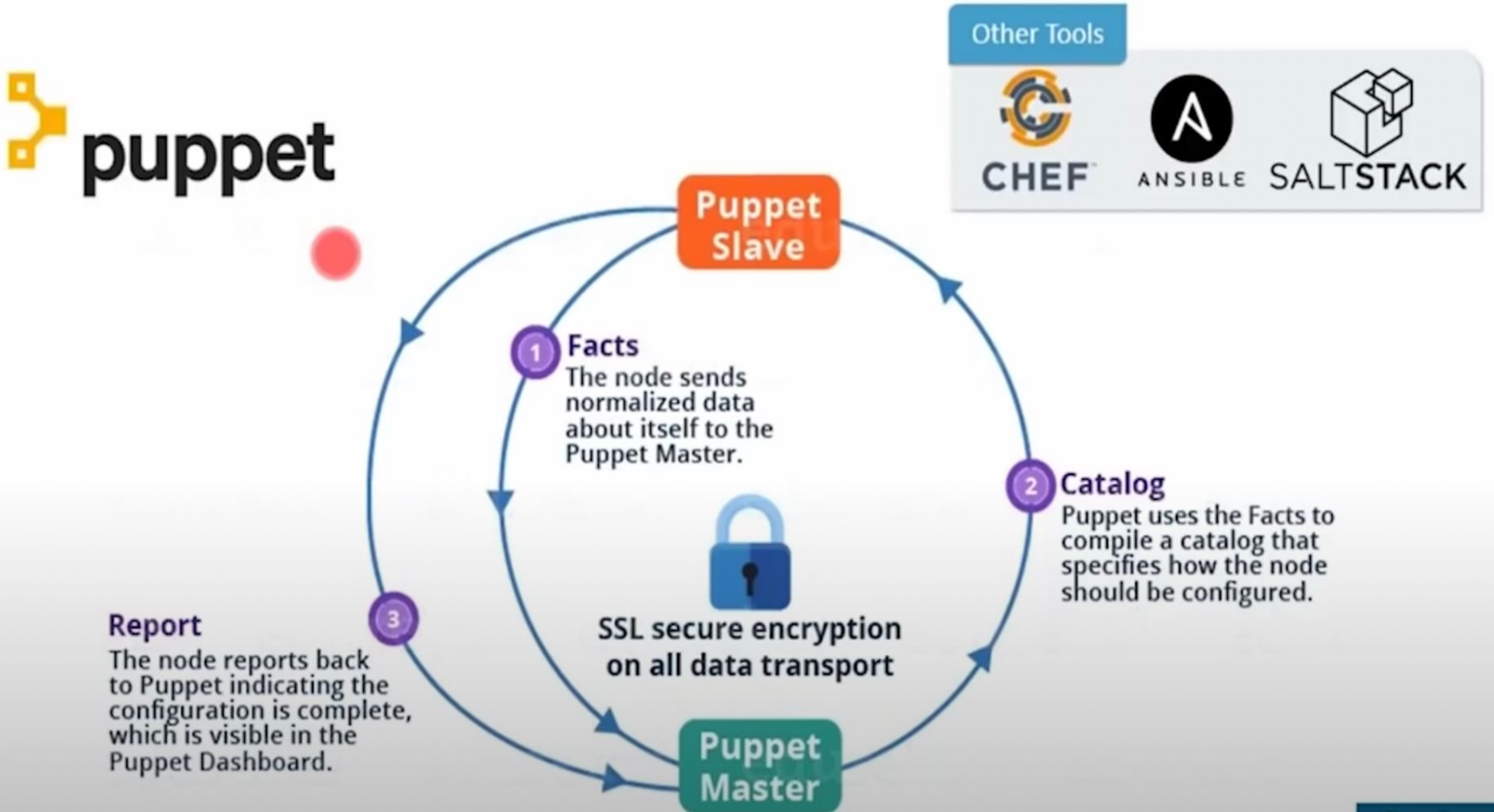
Configuration Management

upGrad INSOFE
INSPIRE • EDUCATE • TRANSFORM

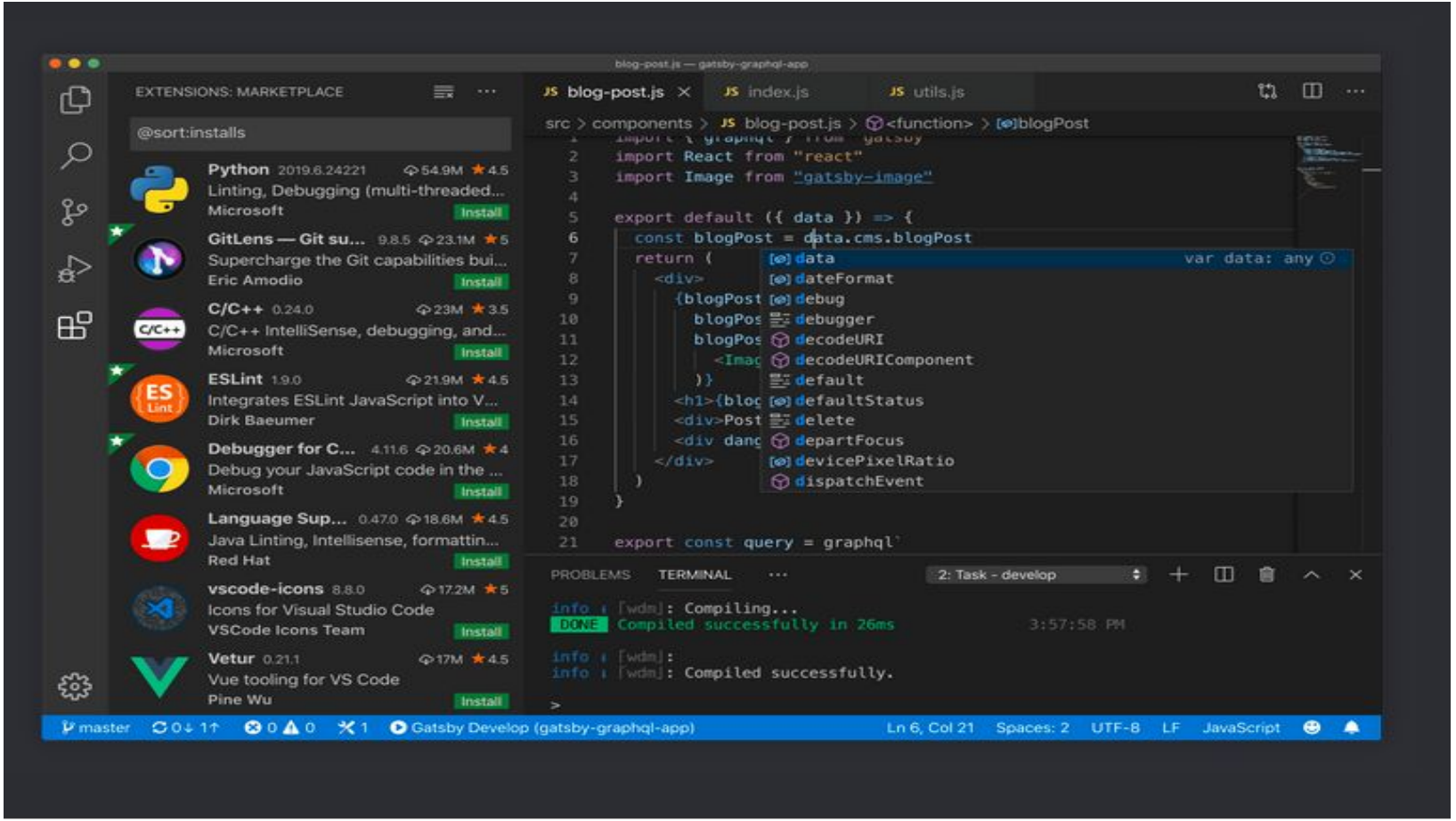
- Configuration Management is the practice of handling changes systematically so that a system maintains its integrity over time
- Configuration Management (CM) ensures that the current design and build state of the system is known, good & trusted
- It doesn't rely on the tacit knowledge of the development team



Configuration Management

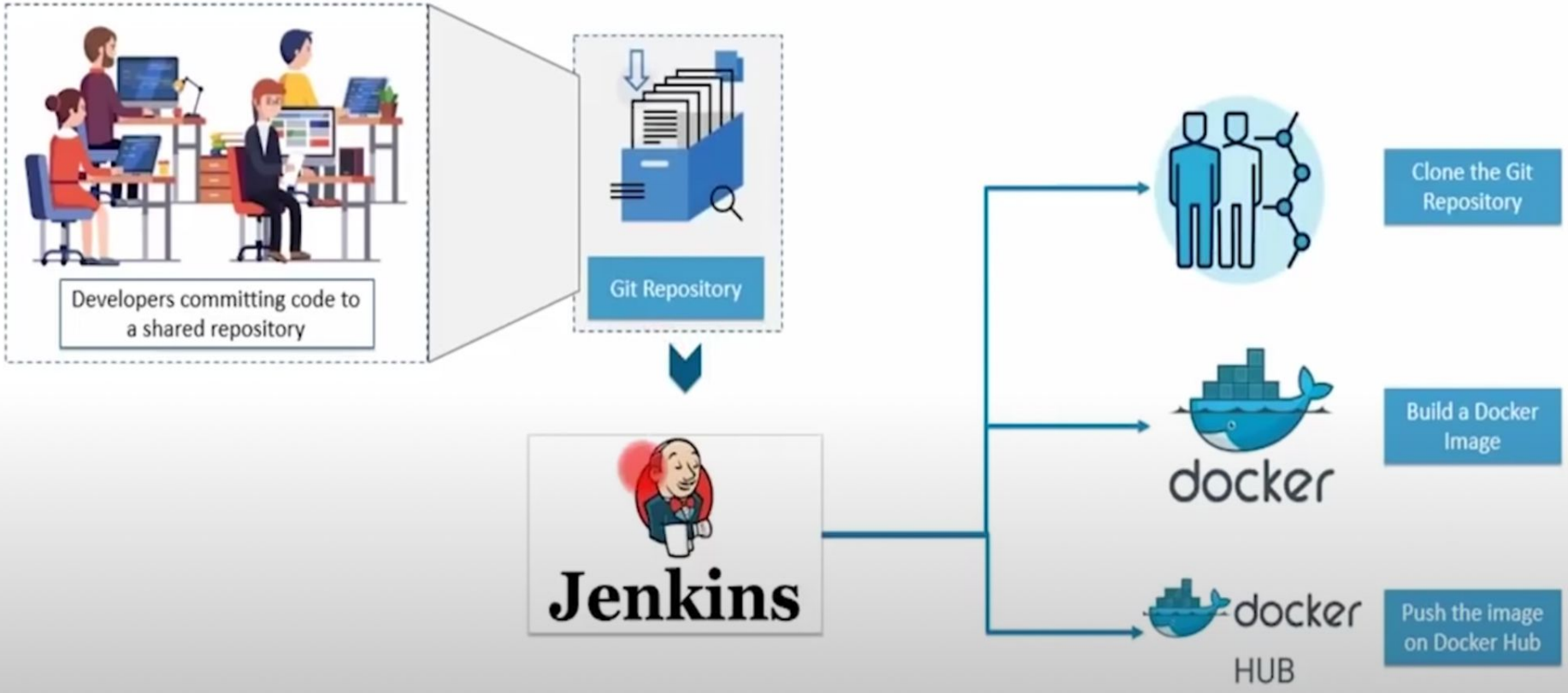


IDE/Editor - Visual Studio Code



Cloud Services Provider

DevOps Use Case



References

1. https://scoutapm.com/blog/sdlc
2. https://stackify.com/what-is-sdlc/
3. https://pointful.github.io/docker-intro/#/2
4. https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/
5. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01
6. https://github.com/mhausenblas/cinf
7. https://github.com/docker/cli
8. https://cri-o.io/
9. https://kubernetes.io/docs/setup/production-environment/container-runtimes/
10. https://computingforgeeks.com/docker-vs-cri-o-vs-containerd/
11. https://github.com/opencontainers/runtime-tools
12. https://cri-o.io/
13. https://opencontainers.org/community/overview/
14. https://www.cncf.io/case-studies/
15. https://github.com/cri-o/cri-o
16. https://containerd.io/docs/