DevOps Training

- ❑ SDLC
- ❑ Intro DevOps and tools
- ❑ Git – Source Code Management
- ❑ Docker – Containerization
- ❑ Kubernetes – Orchestration
- ❑ Ansible – Infra as a Code (IaC)
- ❑ Jenkins – Continuous Integration
- ❑ Nagios – Continuous Monitoring

SDLC

Kapil Thakkar
DevOps Trainer

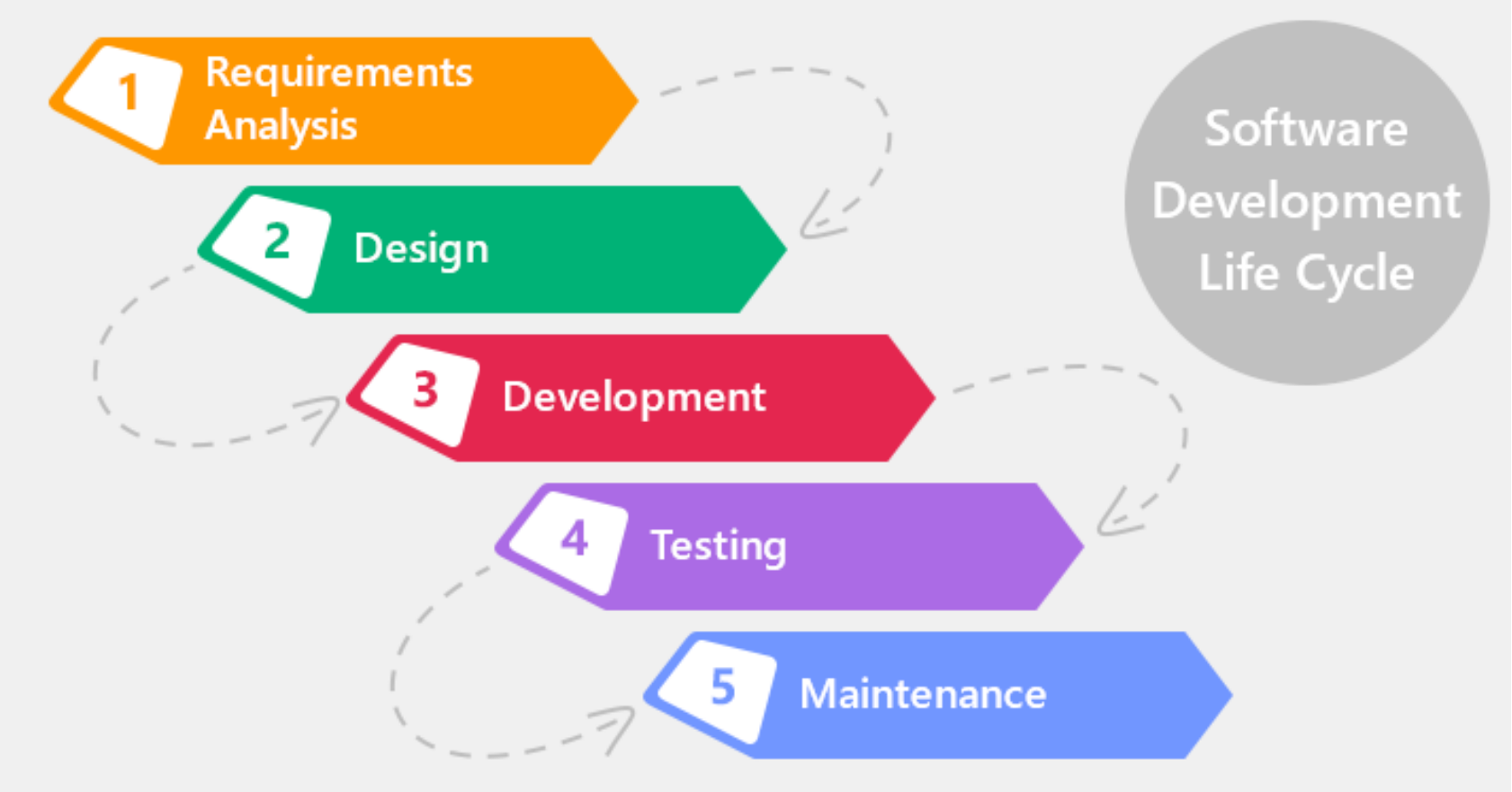
Introduction

- SDLC is the acronym of Software Development Life Cycle.
- SDLC, Software Development Life Cycle is a process used by software industry to design, develop and test high quality software's.
- The SDLC aims to produce a high quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
- It is also called as Software development process.
- The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process.

What is SDLC?

- SDLC is a process followed for a software project, within a software organization.
- It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software.
- The life cycle defines a methodology for improving the quality of software and the overall development process.

Software Development Life Cycle phases?



Requirement gathering and analysis:

- Business requirements are gathered in this phase.
- This phase is the main focus of the project managers and stake holders.
- Meetings with managers, stake holders and users are held in order to determine the requirements like; who is going to use the system.

Design

- In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase.
- System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- The system design specifications serve as input for the next phase of the model.

Development :

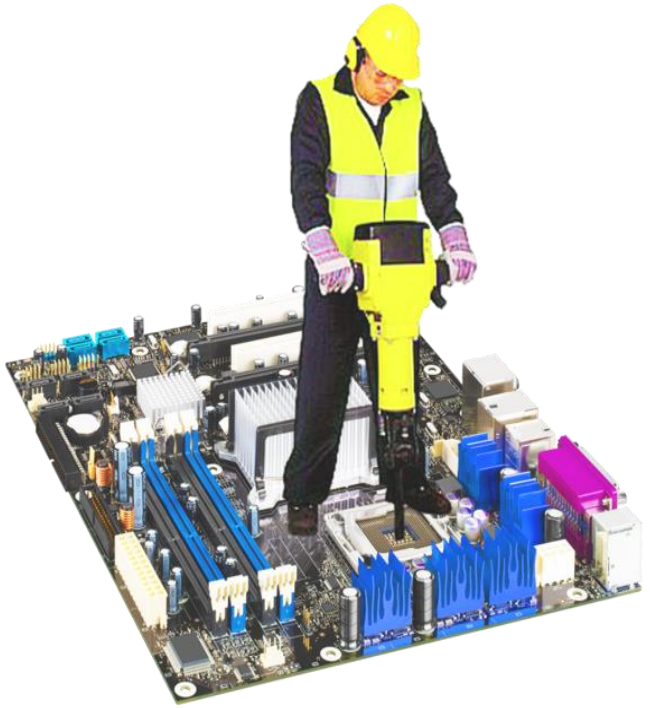
- On receiving system design documents, the work is divided in modules/units and actual coding is started.
- Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

Testing

- After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase.
- During this phase unit testing, integration testing, system testing, acceptance testing are done.

Maintenance

- Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time.
- This process where the care is taken for the developed product is known as maintenance.



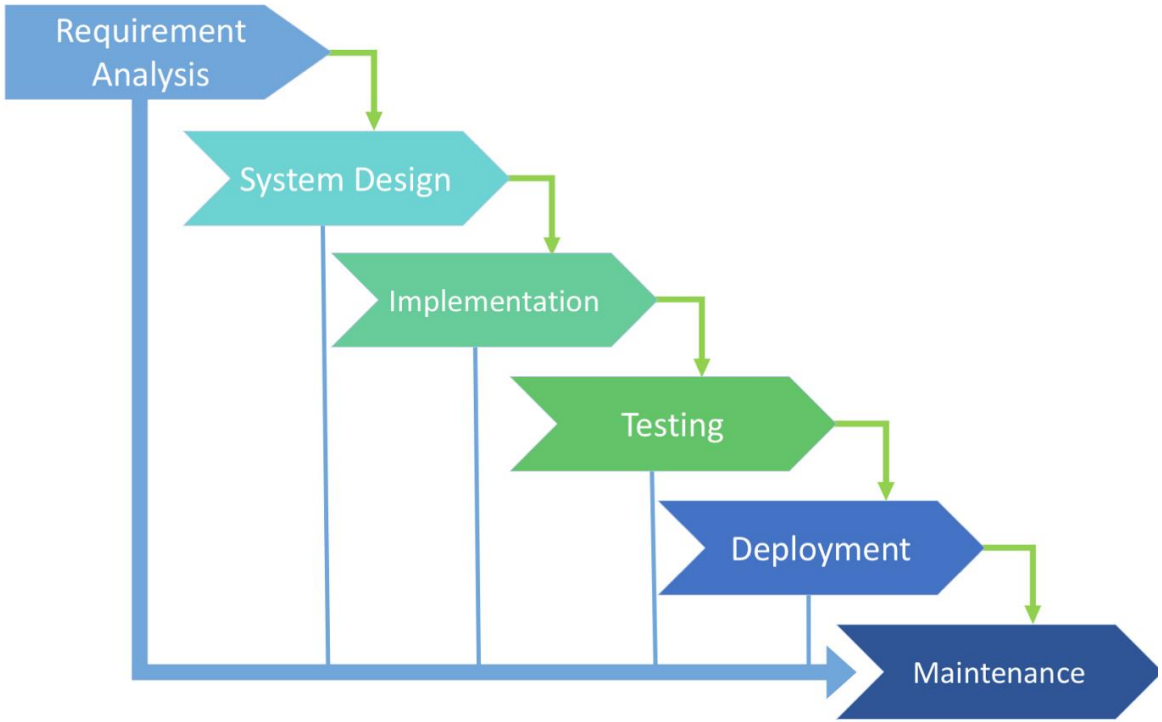
Model of SDLC

There are a number of software development life cycle models and those are implemented during the process of software development. Some of the popular SDLC models are as follows:

- Waterfall model
- Iterative model (skip)
- Spiral model (skip)
- V-model (skip)
- Agile SDLC Model

Waterfall

Waterfall — is a cascade SDLC model, in which development process looks like the flow, moving step by step through the phases of analysis, projecting, realization, testing, implementation, and support. This SDLC model includes gradual execution of every stage completely. This process is strictly documented and predefined with features expected to every phase of this software development life cycle model.



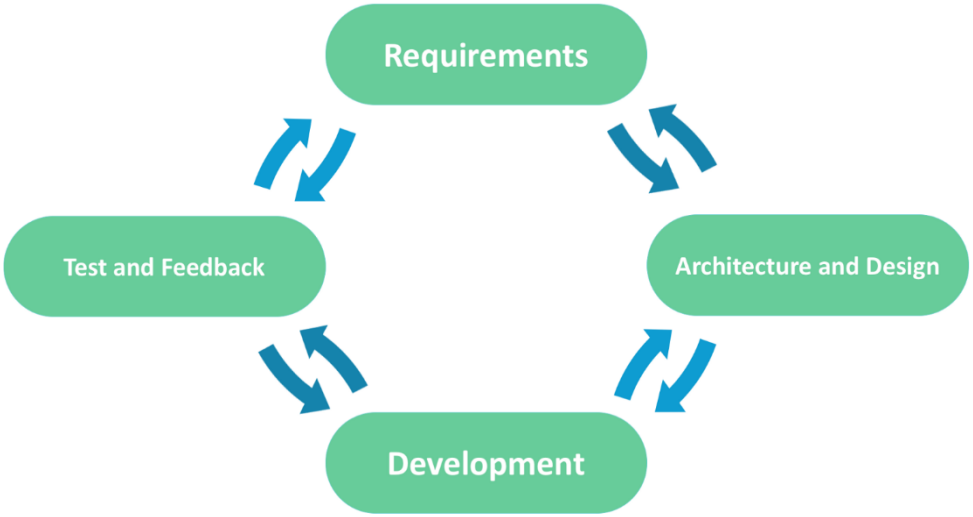
ADVANTAGES	DISADVANTAGES
Simple to use and understand	The software is ready only after the last stage is over
Management simplicity thanks to its rigidity: every phase has a defined result and process review	High risks and uncertainty
Development stages go one by one	Not the best choice for complex and object-oriented projects
Perfect for the small or mid-sized projects where requirements are clear and not equivocal	Inappropriate for the long-term projects
Easy to determine the key points in the development cycle	The progress of the stage is hard to measure while it is still in the development
Easy to classify and prioritize tasks	Integration is done at the very end, which does not give the option of identifying the problem in advance

Use cases for the Waterfall SDLC model:

- The requirements are precisely documented
- Product definition is stable
- The technologies stack is predefined which makes it not dynamic
- No ambiguous requirements
- The project is short

Agile Model

In the agile methodology after every development iteration, the customer is able to see the result and understand if he is satisfied with it or he is not. This is one of the advantages of the agile software development life cycle model. One of its disadvantages is that with the absence of defined requirements it is difficult to estimate the resources and development cost. Extreme programming is one of the practical use of the agile model. The basis of such model consists of short weekly meetings — Sprints which are the part of the Scrum approach.



ADVANTAGES	DISADVANTAGES
Corrections of functional requirements are implemented into the development process to provide the competitiveness	Difficulties with measuring the final cost because of permanent changes
Project is divided by short and transparent iterations	The team should be highly professional and client-oriented
Risks are minimized thanks to the flexible change process	New requirements may conflict with the existing architecture
Fast release of the first product version	With all the corrections and changes there is possibility that the project will exceed expected time

Use cases for the Agile model:

- The users' needs change dynamically
- Less price for the changes implemented because of the many iterations
- Unlike the Waterfall model, it requires only initial planning to start the project

Conclusion

- Software Development Life Cycle (SDLC) is the process of developing information systems through analysis, planning, design, implementation, integration maintenance and testing of software applications.
- Scope Restriction
- Progressive Enhancement
- Pre-defined Structure
- Incremental Planning at each of the stages

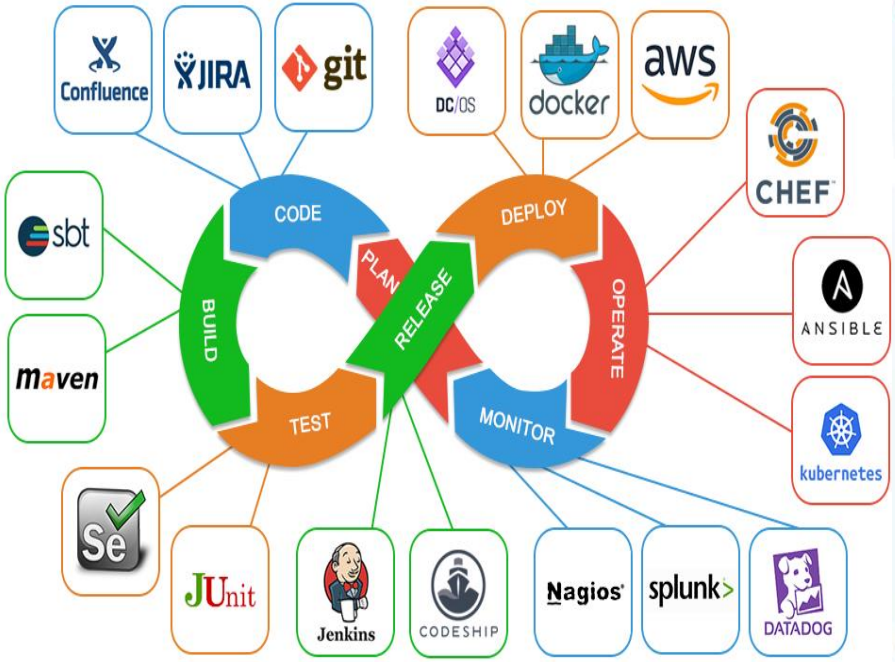
DevOps

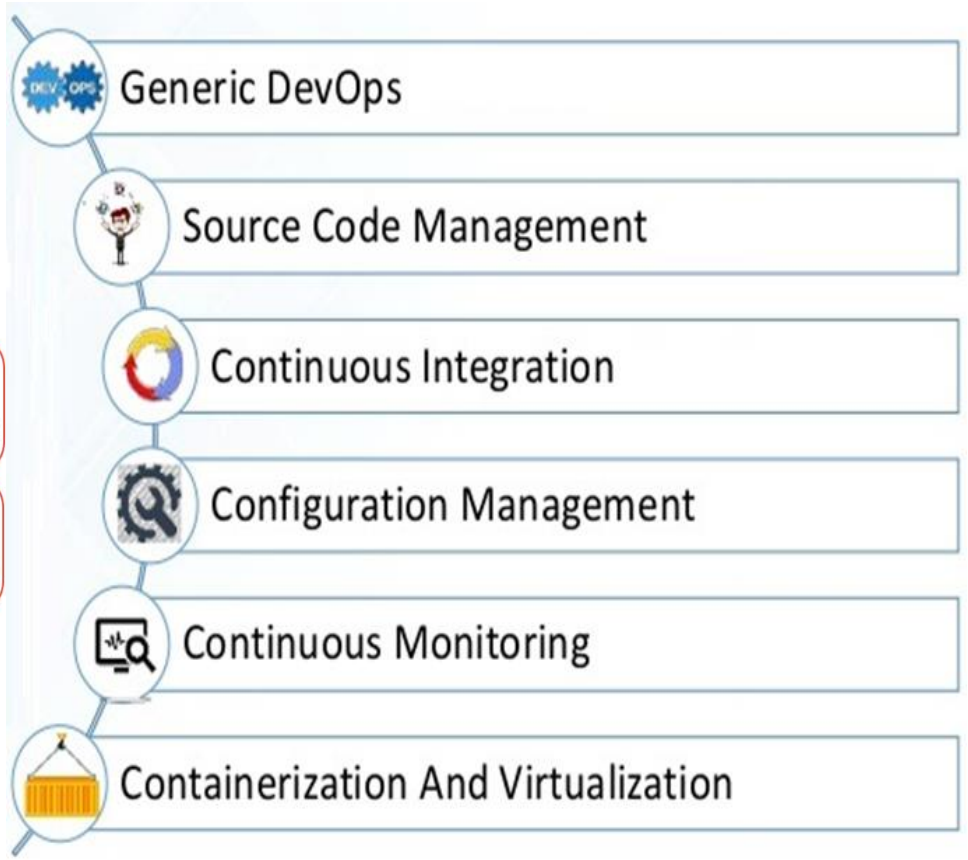
Kapil Thakkar
DevOps Trainer

What is DevOps?

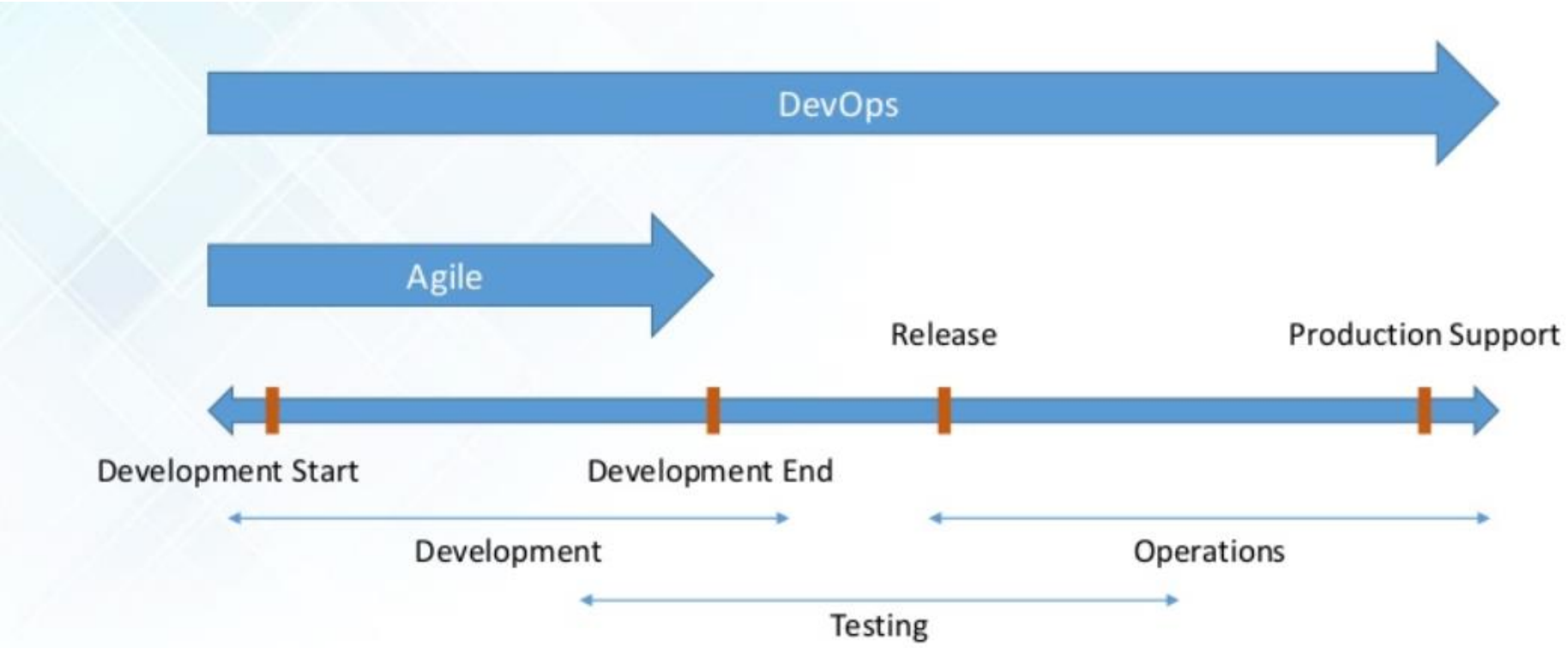
DevOps (development and operations) is an enterprise software development phrase used to **mean** a type of agile relationship between development and IT operations. The goal of **DevOps** is to change and improve the relationship by advocating better communication and collaboration between these two business units.

DevOps Classification





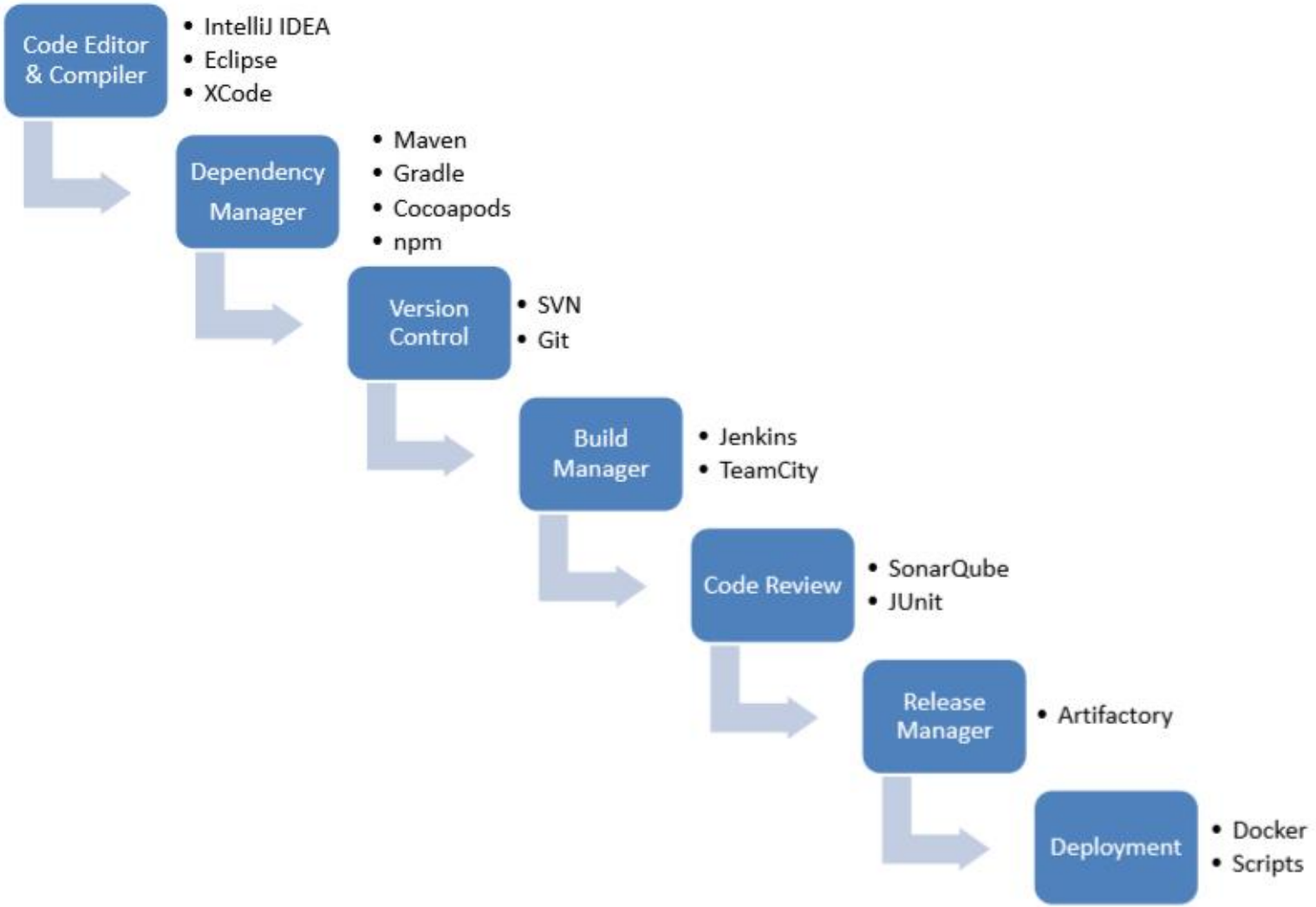
How DevOps is different from Agile



Why DevOps

- Increase deployment frequency
- Lower failure rate of new releases
- Shortened lead time between fixes
- Faster mean time to recovery in the event of new release failure

DevOps Toolchain - Lifecycle



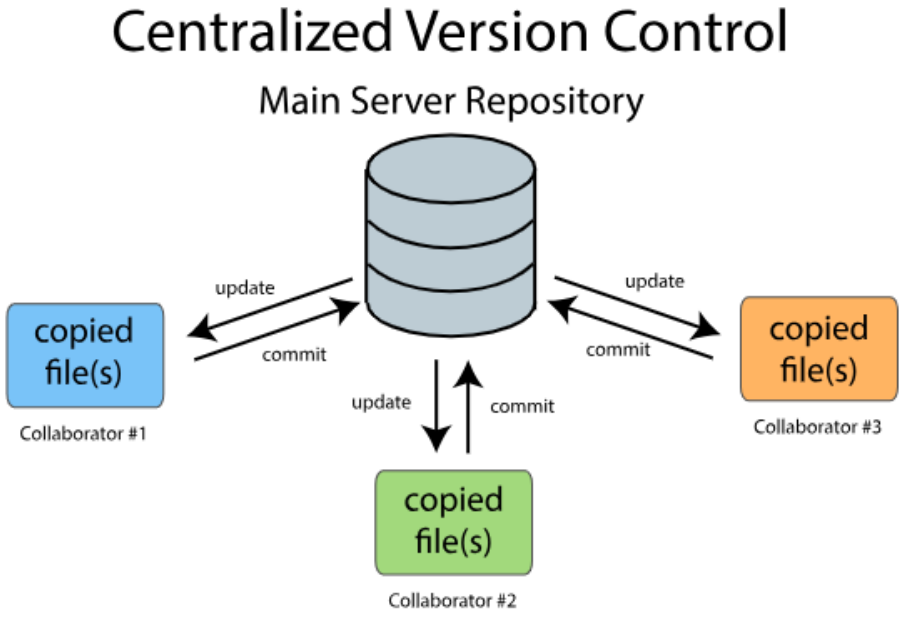
Git -
(VCS / SCM)

Kapil Thakkar
DevOps Trainer

VCS (Version Control System)

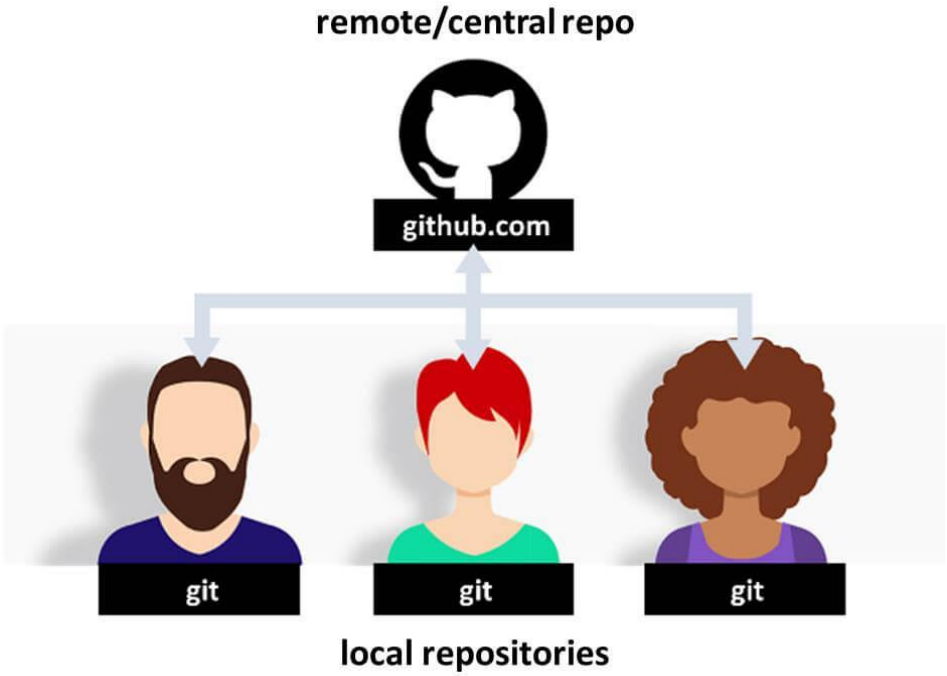
Version control is a **system** that records changes to a file or set of files over time so that you can recall specific **versions** later. For the examples in this book, you will use software **source** code as the files being **version** controlled, though in reality you can do this with nearly any type of file on a computer.

VCS are sometimes known **as** SCM (**Source** Code **Management**) tools or RCS (**Revision Control** System). One of the most popular VCS tools in use today is called **Git**. **Git** is a Distributed VCS, a category known **as** DVCS, more on that later. **Like** many of the most popular VCS systems available today, **Git** is free and open **source**.

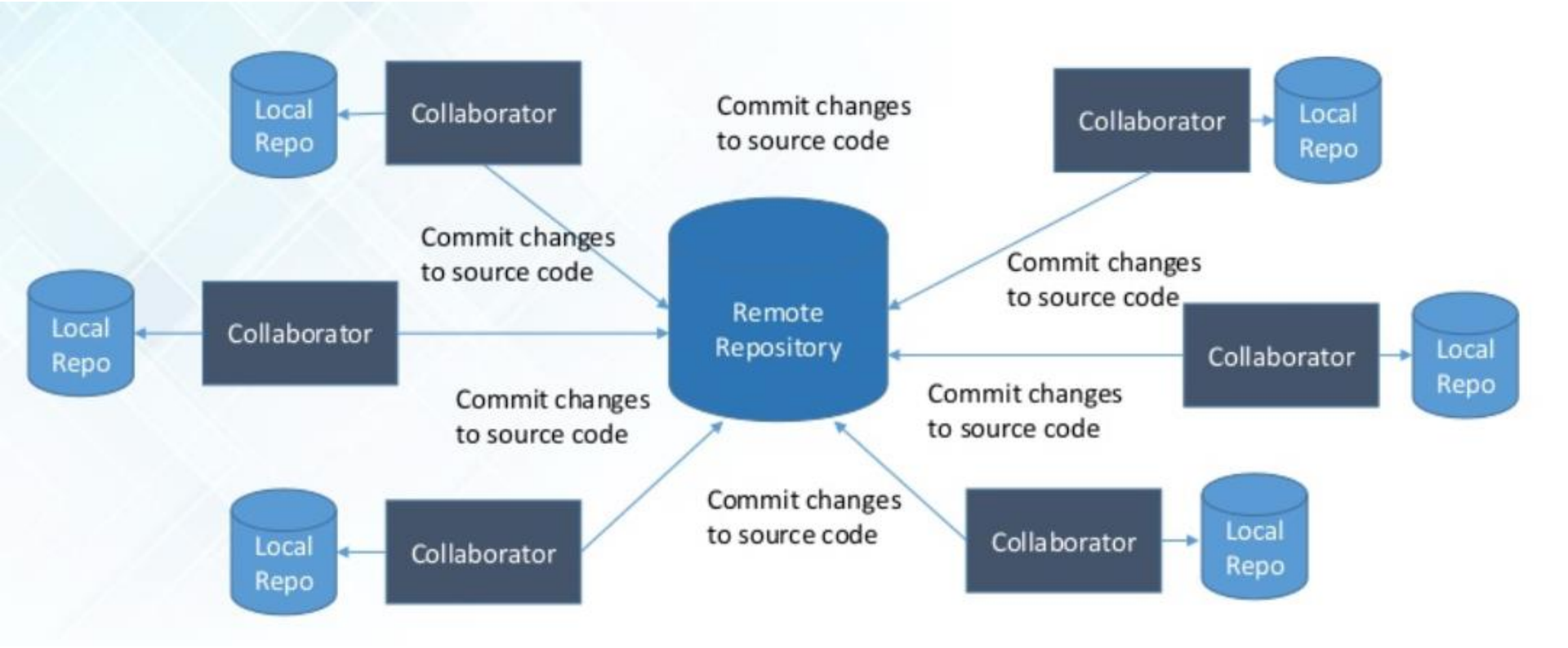


Git and GitHub

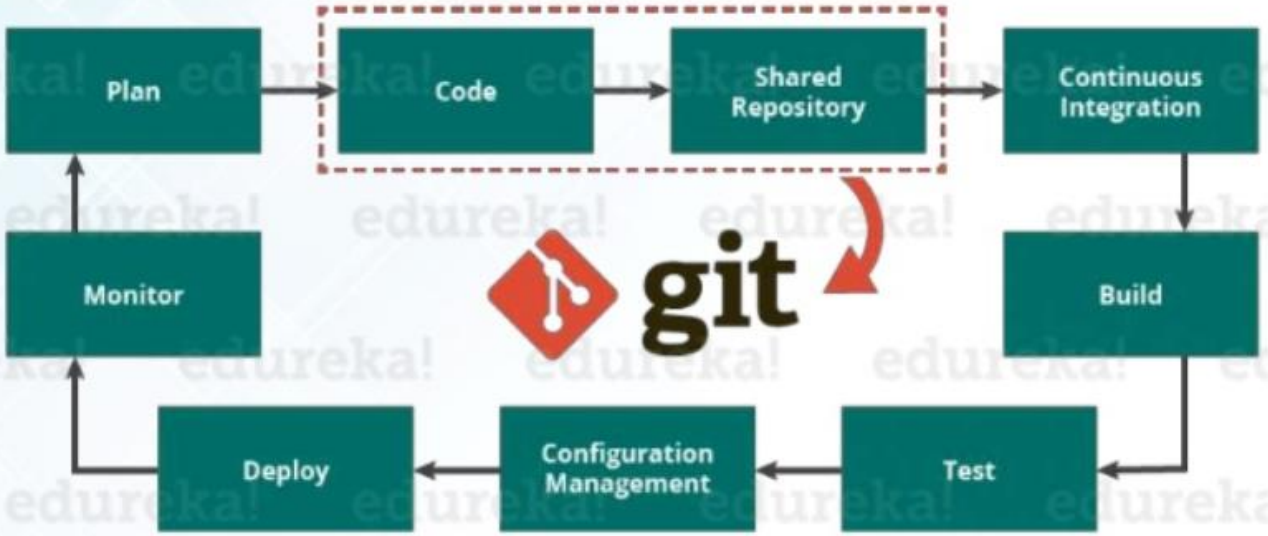
- **Git** is a revision control system, a tool to manage your source code history. **GitHub** is a hosting service for **Git** repositories. So they are not the same thing: **Git** is the tool, **GitHub** is the service for projects that use **Git**.



Git Distributed Architecture



How git performs in DevOps



Every time a commit is made in the Git repository, Continuous Integration server pulls it and compiles it and also deploys it on the test server for testing

Revert commit in git

Remove or fix the bad file in a new commit and push it to the remote repository. This is the most natural way to fix an error. Once you have made necessary changes to the file, commit it to the remote repository for that I will use

git commit -m "commit message"

Create a new commit that undoes all changes that were made in the bad commit.to do this you can use a command

git revert <name of bad commit>

Git - Find list of files in changed in particular commit

To get a list files that has changed in a particular commit use the below command:

git diff-tree -r {hash}

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names only.

The output will also include some extra information, which can be easily suppressed by including two flags:

git diff-tree –no-commit-id –name-only -r {hash}

Here –no-commit-id will suppress the commit hashes from appearing in the output, and –name-only will only print the file names, instead of their paths.

Git Commands

Create a Repository

From scratch -- Create a new local repository
$ git init [project name]

Download from an existing repository
$ git clone my_url

Observe your Repository

List new or modified files not yet committed
$ git status

Show the changes to files not yet staged
$ git diff

Show the changes to staged files
$ git diff --cached

Show all staged and unstaged file changes
$ git diff HEAD

Show the changes between two commit ids
$ git diff commit1 commit2

List the change dates and authors for a file
$ git blame [file]

Show the file changes for a commit id and/or file
$ git show [commit]:[file]

Show full change history
$ git log

Show change history for file/directory including diffs
$ git log -p [file/directory]

Working with Branches

List all local branches
$ git branch

List all branches, local and remote
$ git branch -av

Switch to a branch, my_branch, and update working directory
$ git checkout my_branch

Create a new branch called new_branch
$ git branch new_branch

Delete the branch called my_branch
$ git branch -d my_branch

Merge branch_a into branch_b
$ git checkout branch_b
$ git merge branch_a

Tag the current commit
$ git tag my_tag

Make a change

Stages the file, ready for commit
$ git add [file]

Stage all changed files, ready for commit
$ git add .

Commit all staged files to versioned history
$ git commit -m "commit message"

Commit all your tracked files to versioned history
$ git commit -am "commit message"

Unstages file, keeping the file changes
$ git reset [file]

Revert everything to the last commit
$ git reset --hard

Git Commands

Synchronize

Get the latest changes from origin (no merge)
$ git fetch

Fetch the latest changes from origin and merge
$ git pull

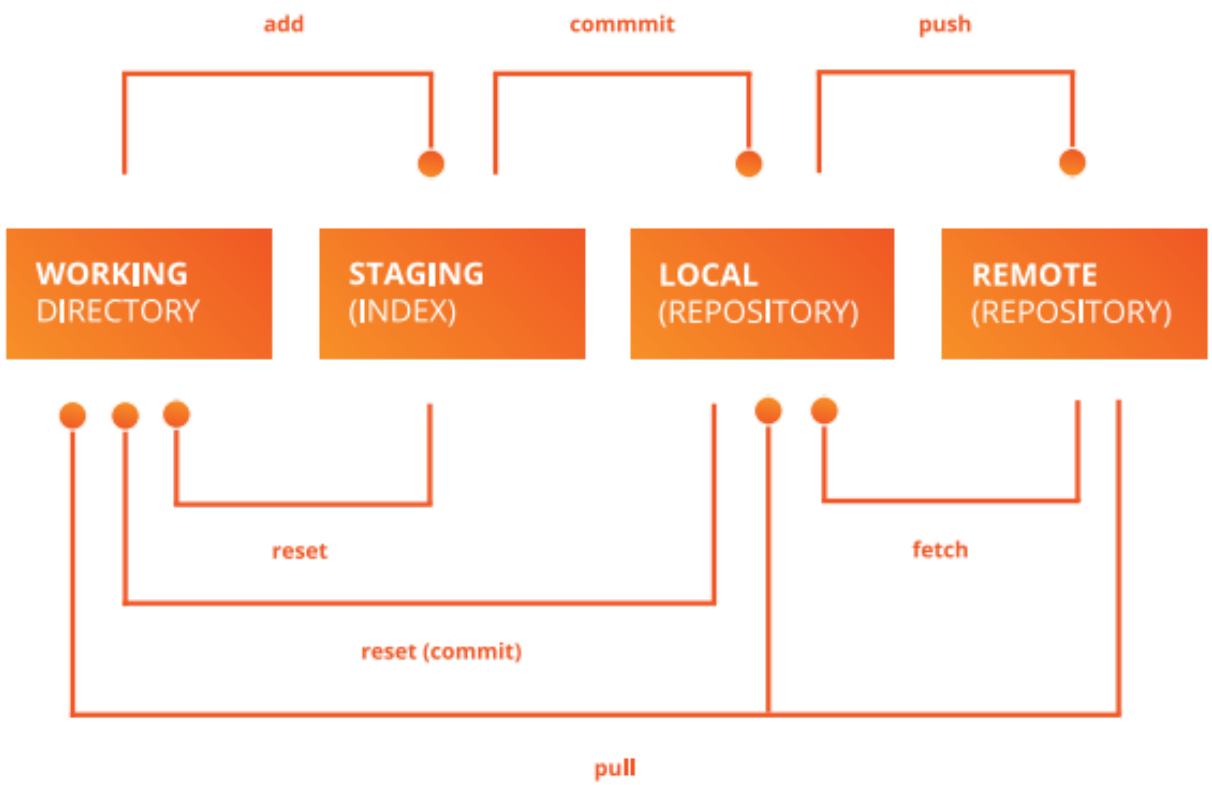
Fetch the latest changes from origin and rebase
$ git pull --rebase

Push local changes to the origin
$ git push

Finally!

When in doubt, use git help
$ git command --help

Or visit **https://training.github.com/** for official GitHub training.



Reference Links

 # Installations:
https://git-scm.com/downloads
https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

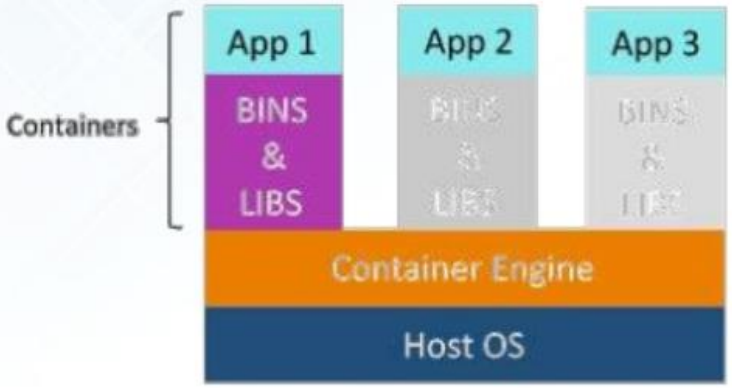
 # Example:
https://github.com/kapilsthakkar25

Containerization

Kapil Thakkar
DevOps Trainer

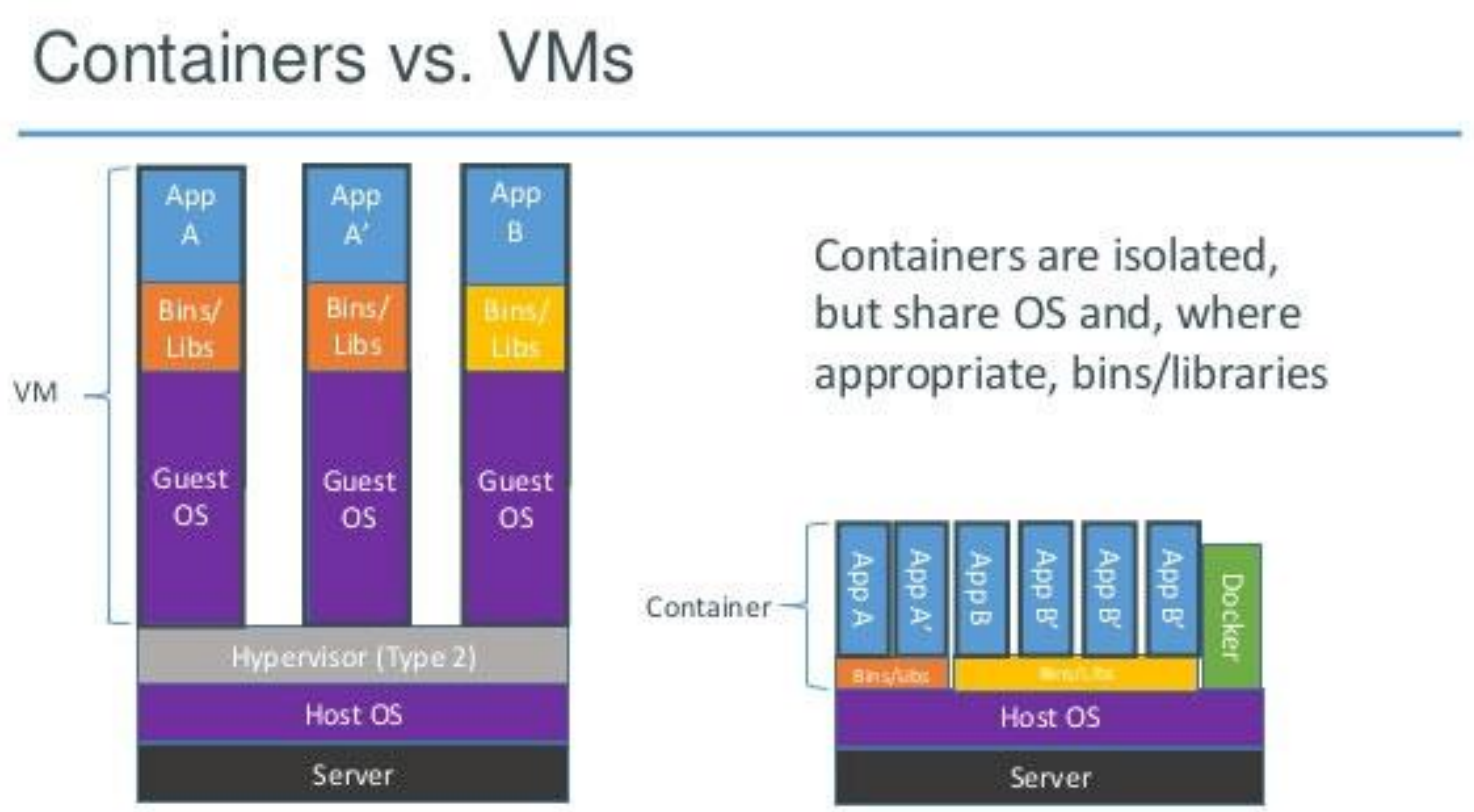
What are Containers?

A container consists of an entire runtime environment: an application, plus all its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package. Containerizing the application platform and its dependencies removes the differences in OS distributions and underlying infrastructure.



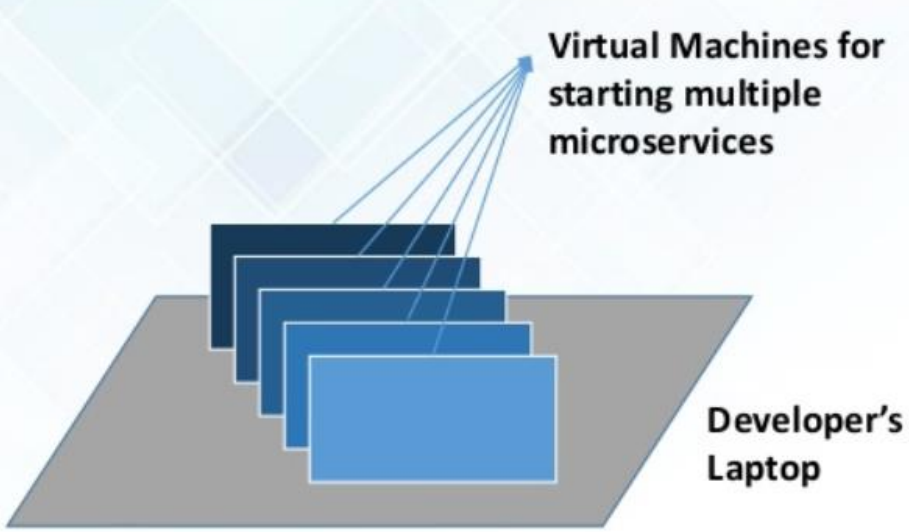
Docker Containers vs VMs

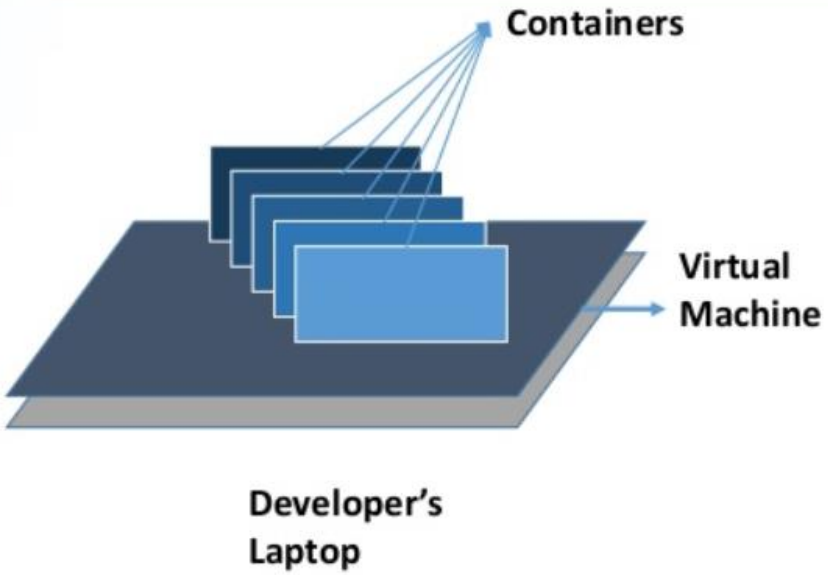
Docker is an open platform for developers and system admins to build, ship, and run distributed applications, whether on laptops, data canter VMs, or the cloud



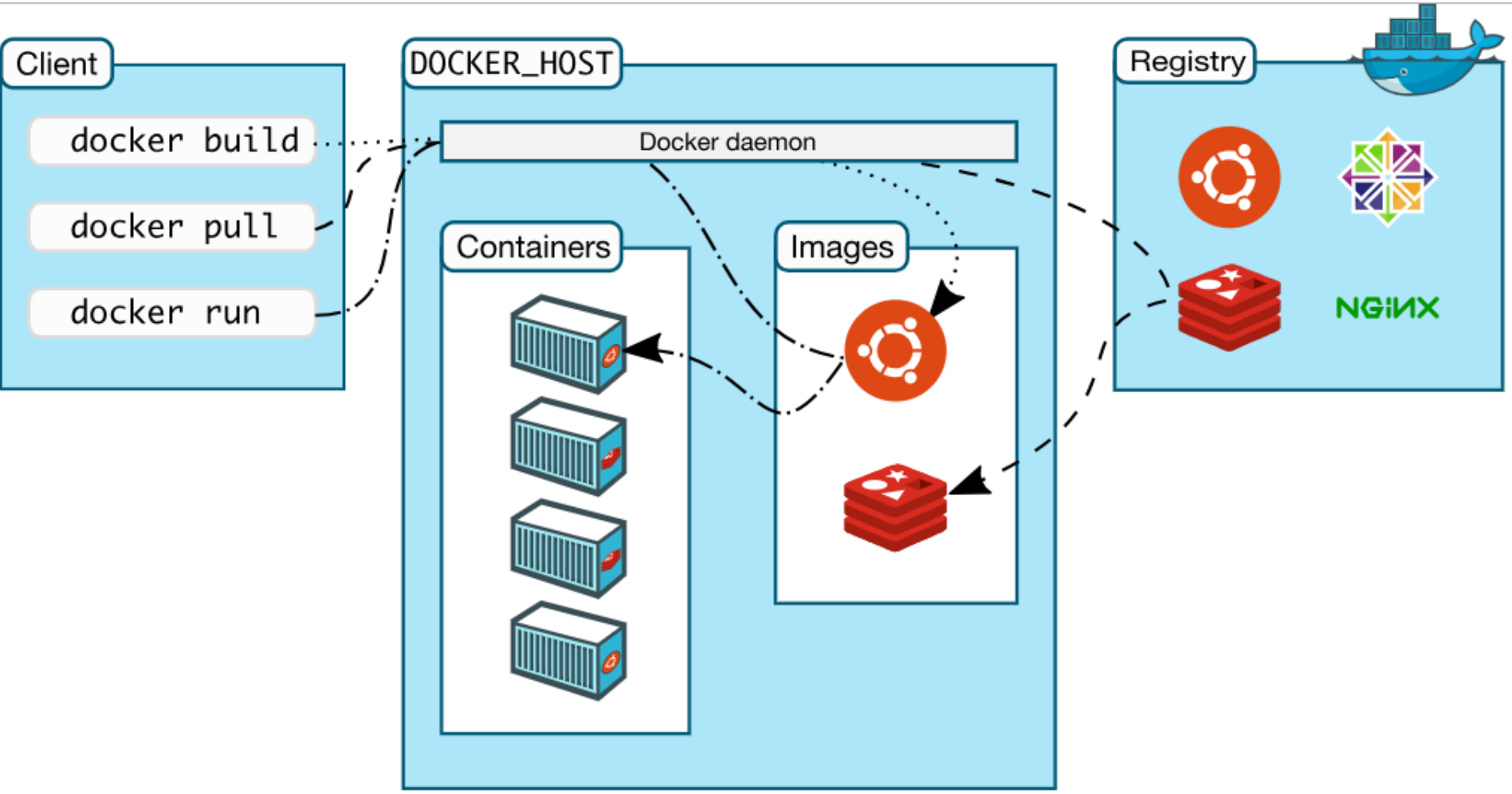
Imagine a scenario when a large application is broken into smaller composable pieces and each of those pieces have their own set of dependencies, let us call these pieces as Microservices. In order to run each of these Microservices what will you use

1. Containers
2. Virtual Machines





Docker Architecture



Basic Docker Commands

```
$ docker pull      # Download an image from a registry
$ docker images    # List all images on a Docker host
$ docker build     # Build an image from a Dockerfile
$ docker run       # Run an image
$ docker commit    # Create an image from a container
$ docker ps        # List all running and stopped instances
$ docker stop      # Stop a running instances
$ docker rm        # Remove an instance
$ docker rmi       # Remove an image
```

Glossary

Layer - a set of read-only files to provision the system

Image - a read-only layer that is the base of your container. Might have a parent image

Container - a runnable instance of the image

Registry / Hub - central place where images live

Docker machine - a VM to run Docker containers (Linux does this natively)

Docker compose - a utility to run multiple containers as a system

Useful one-liners

Download an image

```
docker pull image_name
```

Start and stop the container

```
docker [start|stop] container_name
```

Create and start container, run command

```
docker run -ti --name container_name
    image_name command
```

Create and start container, run command, destroy container

```
docker run --rm -ti image_name command
```

Example filesystem and port mappings

```
docker run -it --rm -p 8080:8080 -v
/path/to/agent.jar:/agent.jar -e
JAVA_OPTS="-javaagent:/agent.jar"
tomcat:8.0.29-jre8
```

Docker cleanup commands

Kill all running containers

```
docker kill $(docker ps -q)
```

Delete dangling images

```
docker rmi $(docker images -q -f
    dangling=true)
```

Remove all stopped containers

```
docker rm $(docker ps -a -q)
```

Docker machine commands

Use docker-machine to run the containers

Start a machine

```
docker-machine start machine_name
```

Configure docker to use a specific machine

```
eval "$(docker-machine env machine_name)"
```

Docker compose syntax

docker-compose.yml file example

```
version: "2"
services:
web:
  container_name: "web"
  image: java:8 # image name
  # command to run
  command: java -jar /app/app.jar
  ports: # map ports to the host
    - "4567:4567"
  volumes: # map filesystem to the host
    - ./myapp.jar:/app/app.jar
mongo: # container name
  image: mongo # image name
```

Create and start containers

```
docker-compose up
```

Interacting with a container

Run a command in the container

```
docker exec -ti container_name command.sh
```

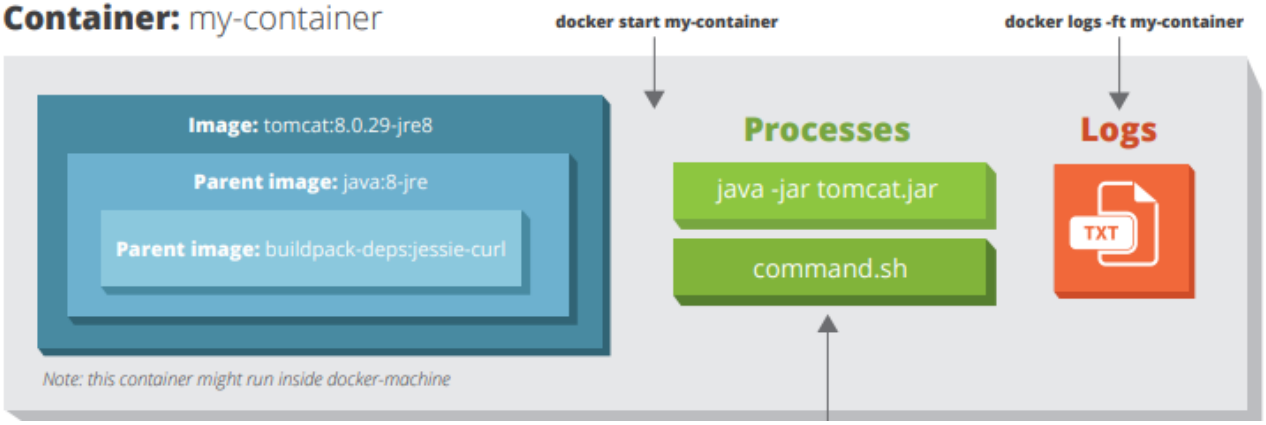
Follow the container logs

```
docker logs -ft container_name
```

Save a running container as an image

```
docker commit -m "commit message" -a "author"
    container_name username/image_name:tag
```

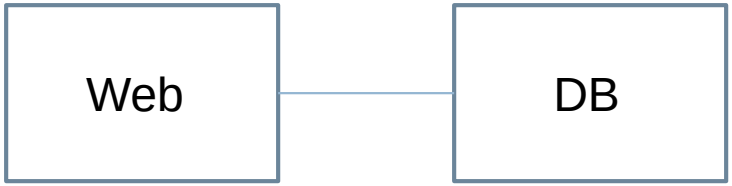
Container: my-container



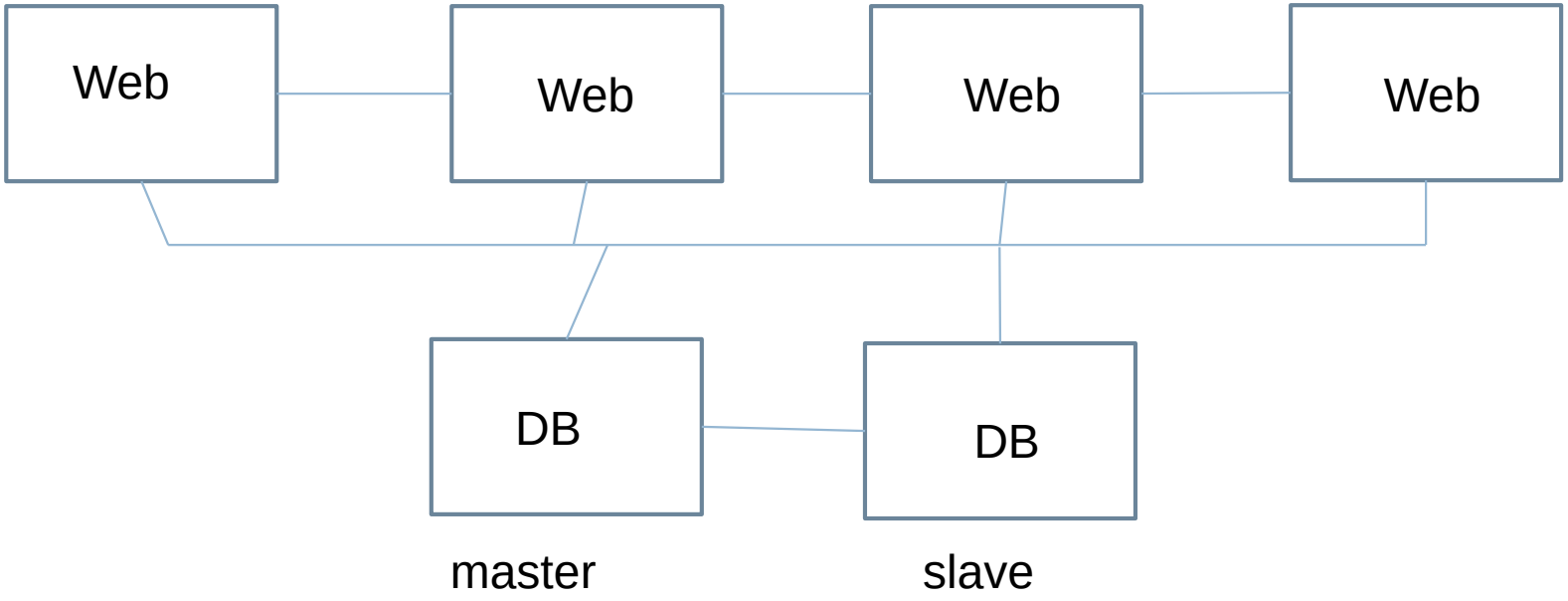
Dockerfile for MongoDB

```
FROM ubuntu                                    #Set the base Image to Ubuntu
MAINTAINER Example McAuthor                    #File author / mantainer
RUN apt-get update                             #Update the repository sources list
#Add the package verification key
RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
#Add MongoDB to the repository sources list
RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | tee /etc/apt/sources.list.d/mongodb.list
#Update the repository sources list once more
RUN apt-get update
#Install MongoDB package
RUN apt-get install -y mongodb-10gen
#Create the default data directory
RUN mkdir -p /data/db
#Expose the default port
EXPOSE 27017
#Default port to execute the entrypoint (MongoDB)
CMD ["--port 27017"]
#Set default container command
ENTRYPOINT usr/bin/mongod
```

Dev / Staging

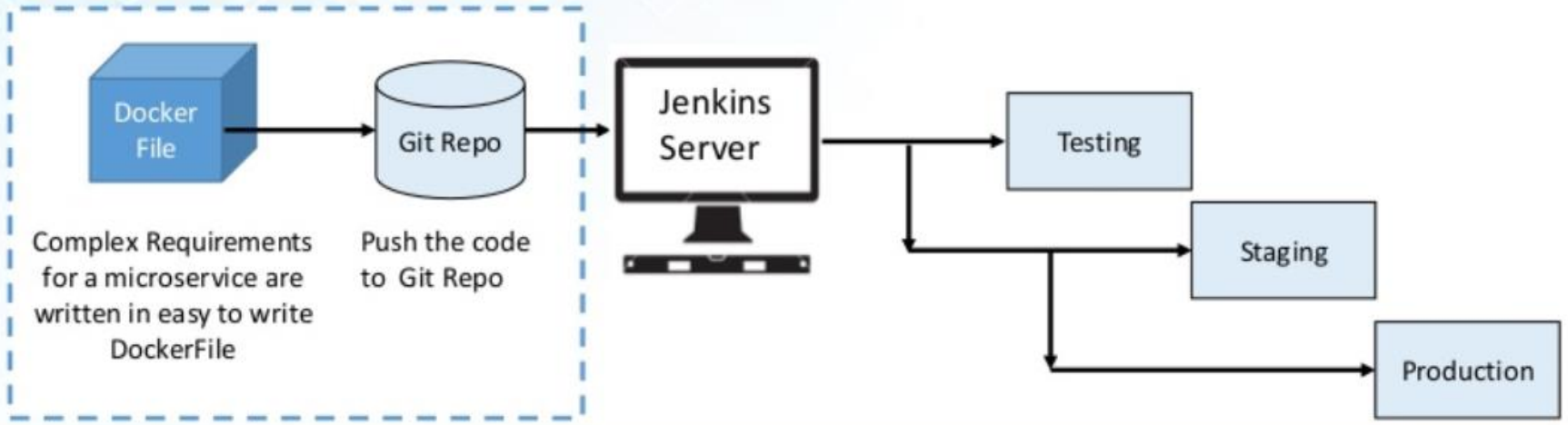


Production



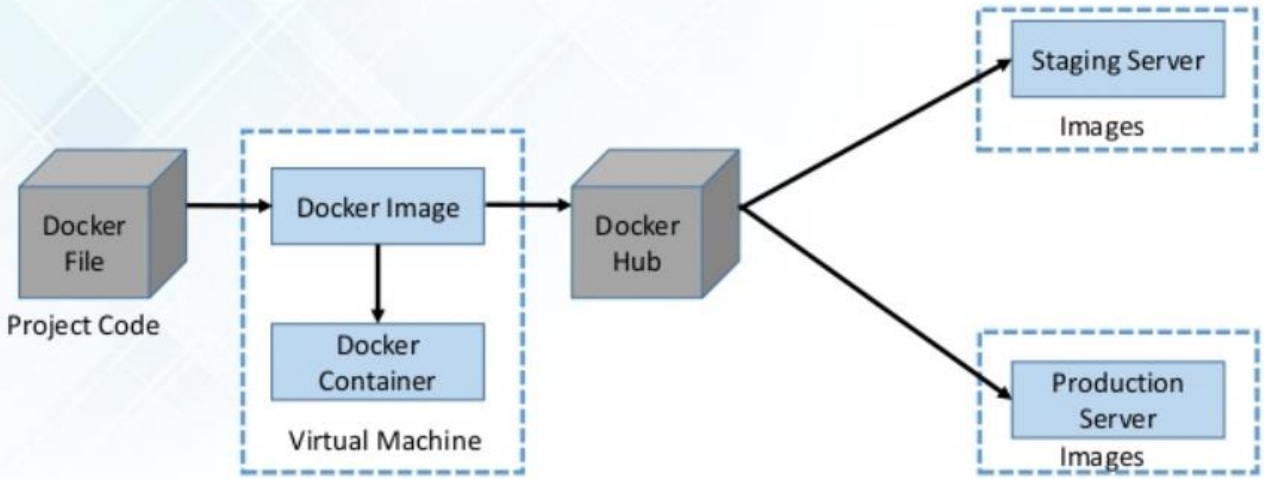
How docker is use in SDLC?

- 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



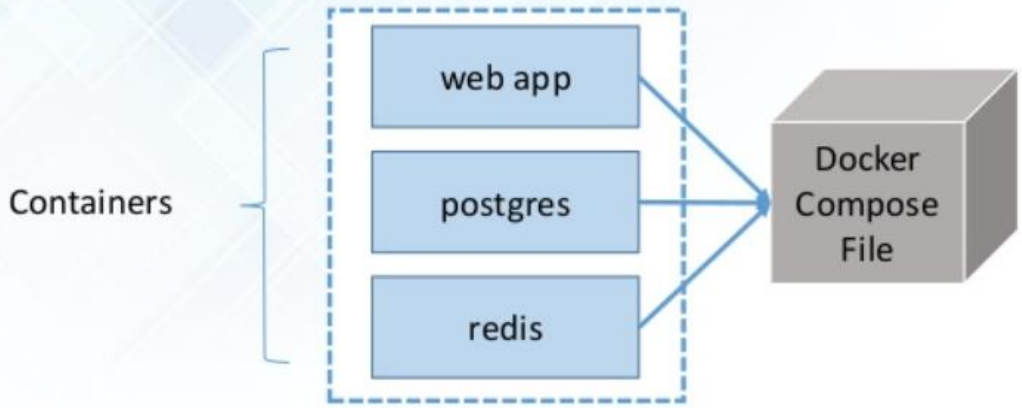
Docker provides consistent computing environment throughout the SDLC

- 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



What is Docker compose

Docker Compose makes it easier to configure and run applications made up of multiple containers. For the example: imagine being able to define three containers—one running a web app, another running postgres, and a third running redis—all in one YAML file and then running those three connected containers with a single command.



You can run these three containers with a single command

Reference Links

 # Installations:

https://docs.docker.com/install/linux/docker-ce/centos/
https://docs.docker.com/compose/install/
https://www.docker.com/products/docker-desktop
https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
https://www.tutorialspoint.com/docker/docker_compose.htm

 # Docker Hub

https://hub.docker.com/
https://cloud.docker.com/u/kapilsthakkar25/repository/docker/kapilsthakkar25/spaceshooter

 # Git Example:

https://github.com/kapilsthakkar25/webapp-repo
https://github.com/kapilsthakkar25/docker-mysql-php-example
https://github.com/kapilsthakkar25/docker-wordpress
https://github.com/kapilsthakkar25/space-shooter
https://github.com/kapilsthakkar25/docker-nginx-php-mysql
https://github.com/kapilsthakkar25/docker-repo

Kubernetes

Kapil Thakkar
DevOps Trainer

What is Kubernetes?

Kubernetes(K8S) is an open source tool for managing containerised workloads. It operated at the container(not hardware) level to automate the deployment, scaling and management of applications. ▹ K8S works alongside a containerization tool, like Docker. So if containers are the 'Ingredients' of an application, then K8S would be the 'Chef or Ansible'

As well as managing individual containers, K8S can also manage clusters:

- A cluster is a series of servers connected to run containers.
- K8S can scale upto 5000 server and 150,000 pods in a single cluster.
- A pod is a group of containers that share resources, a network and can communicate with one another

Why use k8s?

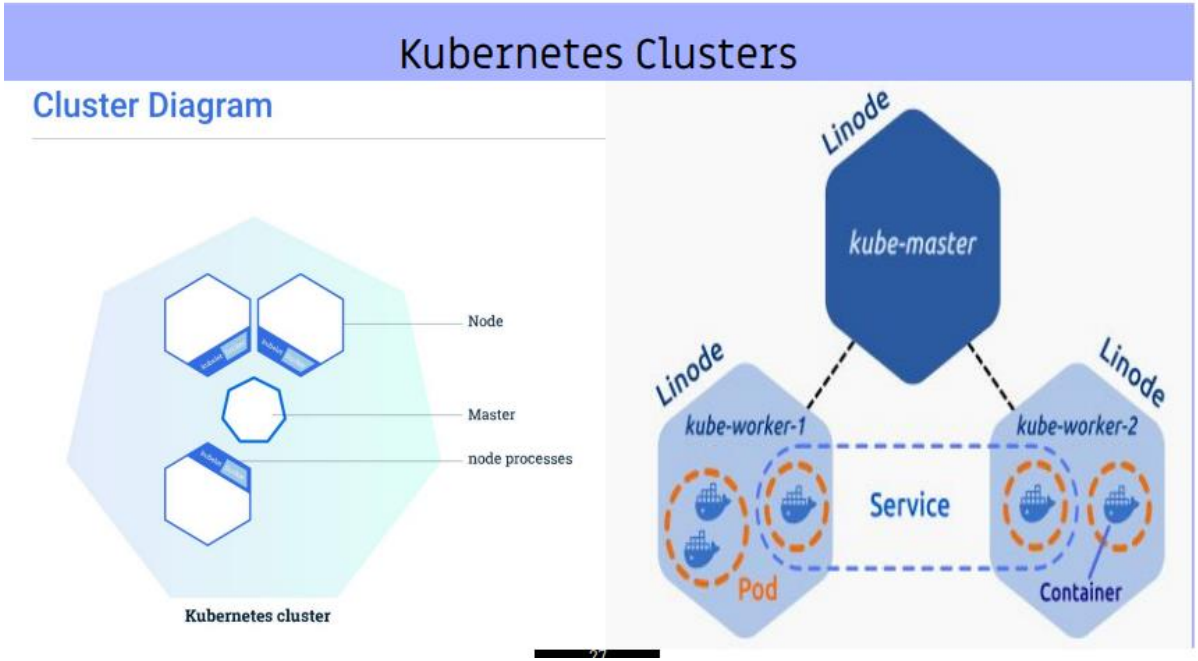
As an orchestration platform, K8S provides features to make the management, maintenance and life-cycle of containers easier than using a container-engine alone.

- Horizontal Scaling
- Self Healing
- Automated Rollouts
- Various other features like Service Discovery and load balancing etc.

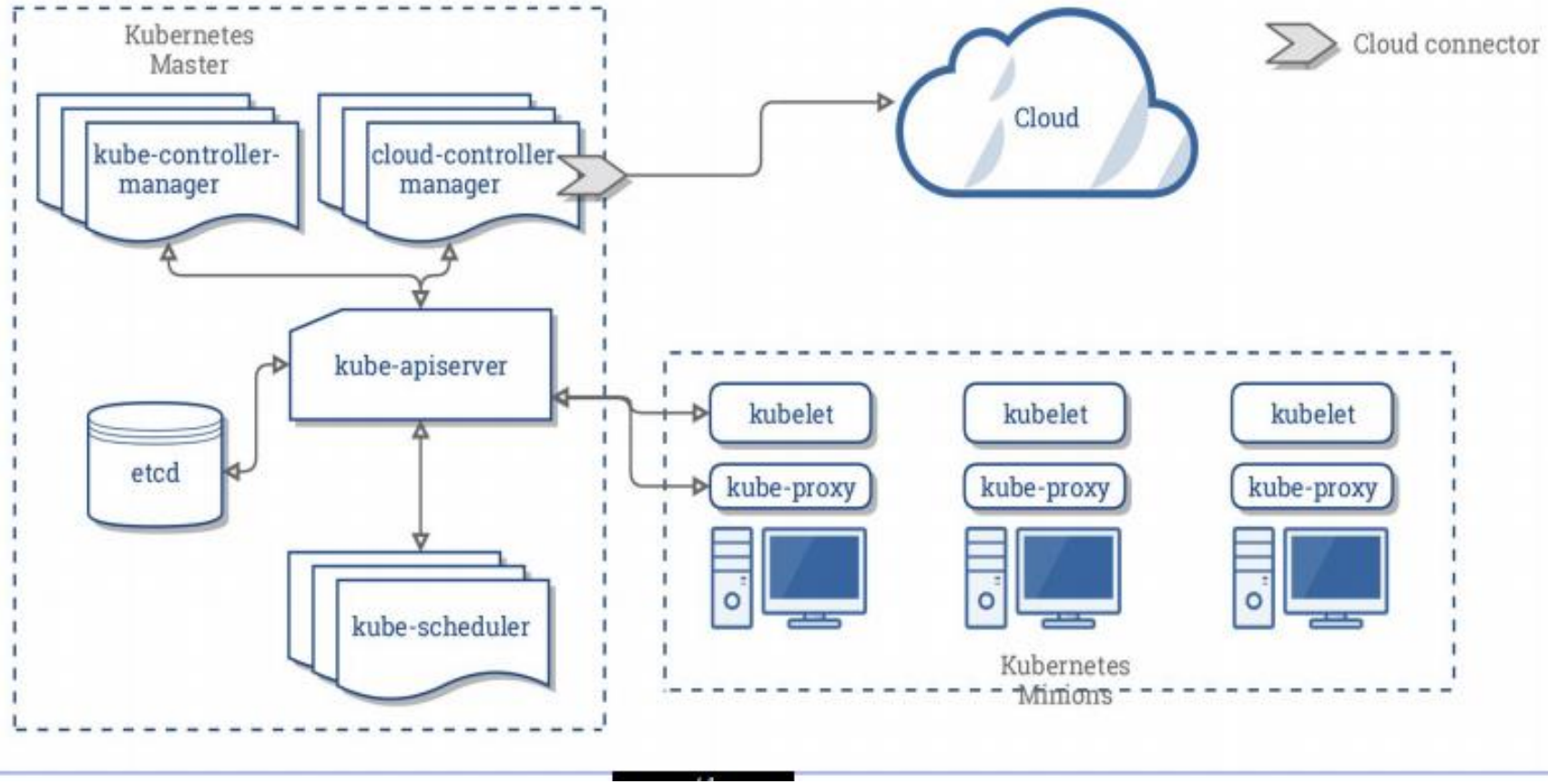
Kubernetes Cluster

Containerised applications are deployed with K8S into highly available clusters

- Clusters run over several computers called Worker Nodes, that are connected to work as a single unit.
- Containerised apps are automatically distributed among the Worker Nodes at deploy time.
- A Master Node manages the cluster - coordinating scheduling, scaling and rolling updates.



Kubernetes Architecture



Reference Links

 # Installations:

https://kubernetes.io/docs/setup/independent/install-kubeadm/
https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/
https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
https://kubernetes.io/docs/tasks/tools/install-kubectl/
https://github.com/kapilsthakkar25/kubernetes-demo/blob/master/basic-example4.txt

 # Interactive Tutorials and Tasks:

https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-interactive/
https://kubernetes.io/docs/tasks/

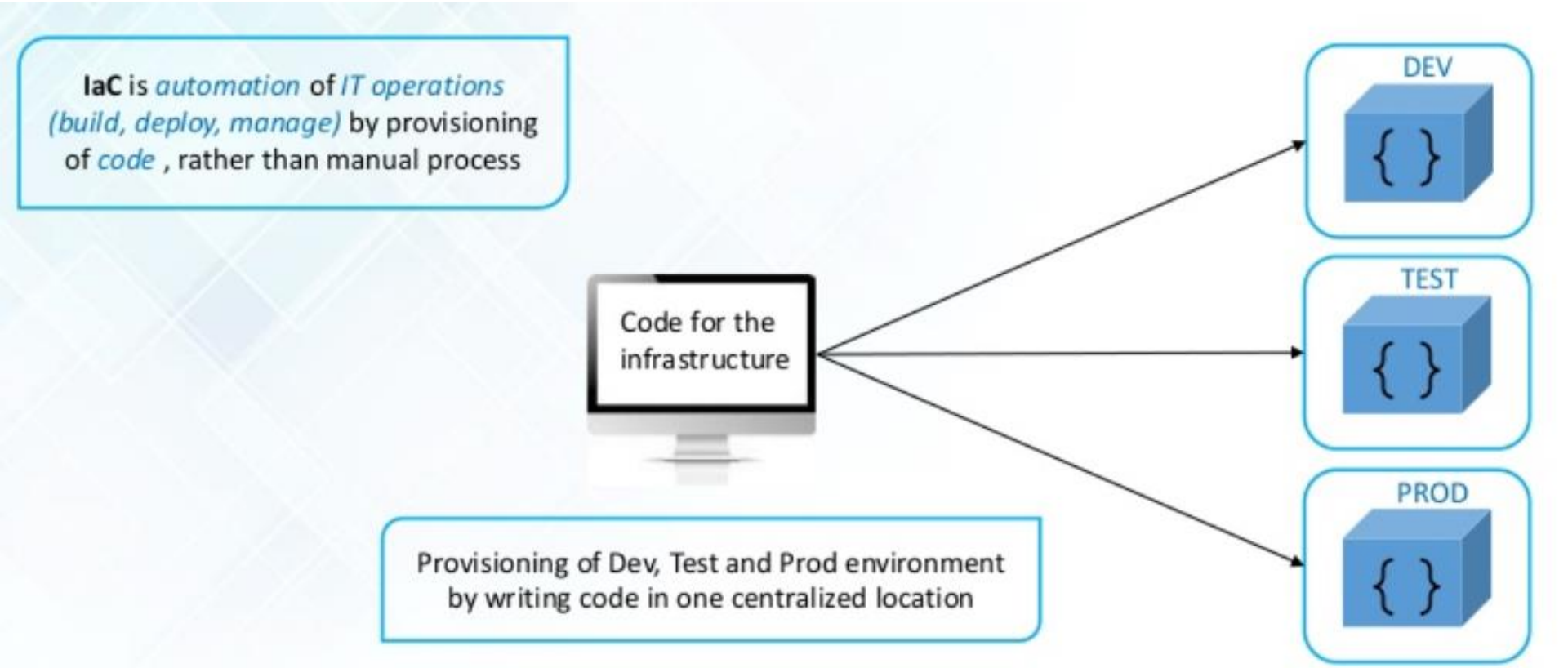
 # Git Example:

https://github.com/kapilsthakkar25/kubernetes-demo

IaC -
Infra as a Code

Kapil Thakkar
DevOps Trainer

Infra as a Code (IaC)



Puppet module and manifest

Puppet Module

A Puppet Module is a collection of Manifests and data (such as facts, files, and templates), and they have a specific directory structure. Modules are useful for organizing your Puppet code, because they allow you to split your code into multiple Manifests. Modules are self-contained bundles of code and data.

You can find pre-defined modules at forge.puppet.com/modules

Puppet Manifest

Every Slave has got its configuration details in Puppet Master, written in the native Puppet language. These details are written in the language which Puppet can understand and are termed as Manifests. They are composed of Puppet code and their filenames use the *.pp* extension. These are basically Puppet programs.

```
node 'host2' {
class { 'apache': } # use apache module
apache::vhost { 'example.com': # define vhost
resource port => '80',
docroot => '/var/www/html'
}
}
```

What is Ansible

Ansible is an **open-source IT automation engine**, which can remove drudgery from your work life, and will also dramatically **improve the scalability, consistency, and reliability of your IT environment**. We'll start to explore how to automate repetitive system administration tasks using Ansible, and if you want to learn more, you can go much deeper into how to use Ansible with Cloud Academy's new Introduction to Ansible learning path.

You can use Ansible to automate three types of tasks:

- **Provisioning**: Set up the various servers you need in your infrastructure.
- **Configuration management**: Change the configuration of an application, OS, or device; start and stop services; install or update applications; implement a security policy; or perform a wide variety of other configuration tasks.
- **Application deployment: Make DevOps easier by automating the deployment of internally developed applications to your production systems.**

Ansible can automate IT environments whether they are hosted on traditional bare metal servers, virtualization platforms, or in the cloud. It can also automate the configuration of a wide range of systems and devices such as databases, storage devices, networks, firewalls, and many others.

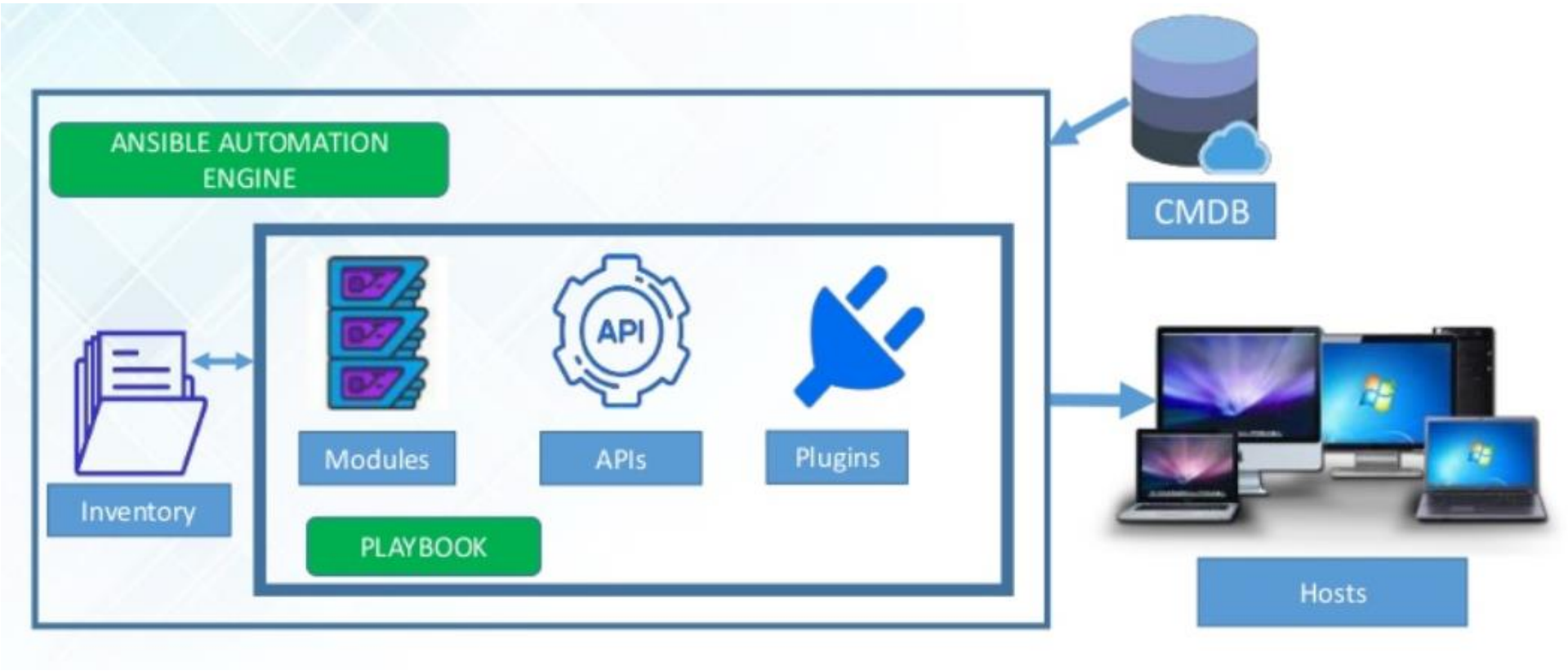
Why Ansible

There are many other IT automation tools available, including more mature ones like Puppet and Chef, so why would you choose Ansible? The main reason is simplicity. Michael DeHaan, the creator of Ansible, already had a lot of experience with other configuration management tools when he decided to develop a new one. He said that he wanted “a tool that you could not use for six months, come back to, and still remember.”

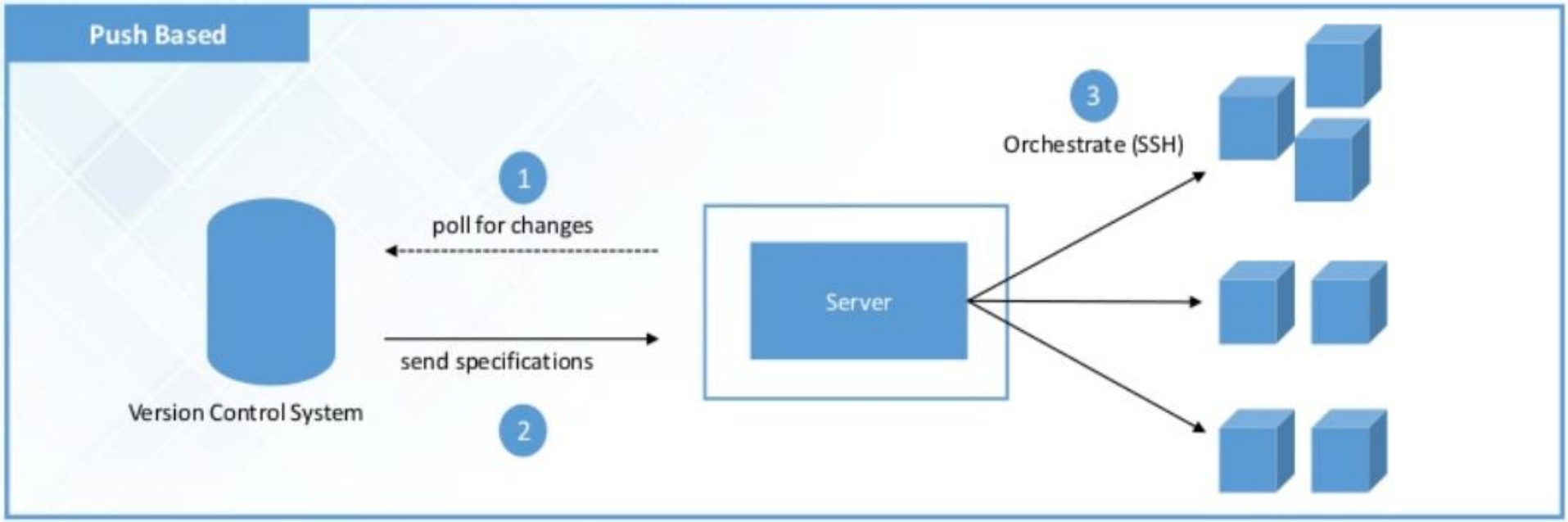
DeHaan accomplished this by using YAML, a simple configuration language. Puppet and Chef, on the other hand, use Ruby, which is more difficult to learn. This makes Ansible especially appealing to system administrators.

DeHaan also simplified Ansible deployment by making it agentless. That is, instead of having to install an agent on every system you want to manage (as you have to do with Puppet and Chef), Ansible just requires that systems have Python (on Linux servers) or PowerShell (on Windows servers) and SSH.

Ansible Architecture



Ansible is Push based CM tool



Directory layout

- group_vars
- host_vars
- roles

```
ansible.cfg                 # parameters that affect running ansible
inventory/                  # an inventory defines an environment
    hosts                   # defines the hosts in an inventory
    group_vars/             # here we assign variables to particular groups
        all                 # global variables for all groups
        dbservers/          # directory for dbservers group
            secrets         # -- encrypted variables for dbservers group
            vars            # -- plaintext variables for dbservers group
        group2              # plaintext variables for group2
    host_vars/              # here we assign variables to particular hosts
        hostname1           # if systems need specific variables, put them here
        hostname2           # ""
site.yml                    # master playbook
webservers.yml              # playbook for webserver tier
dbservers.yml               # playbooke for database tier
galaxy_roles/               # roles imported from galaxy
roles/                      # in-house roles
    common/                 # this hierarchy represents a "role"
        tasks/              # 'tasks' contains the actions that implement role
            main.yml        # -- main.yml could include other files if warranted
        handlers/           # 'handlers' can be notified by tasks on change
            main.yml        # -- handlers file often defines service actions
        templates/          # files for use with the template module
            hosts.j2        # -- Jinja templates, should end in .j2
        files/              # 'files' is the start for relative paths
```

Reference Links

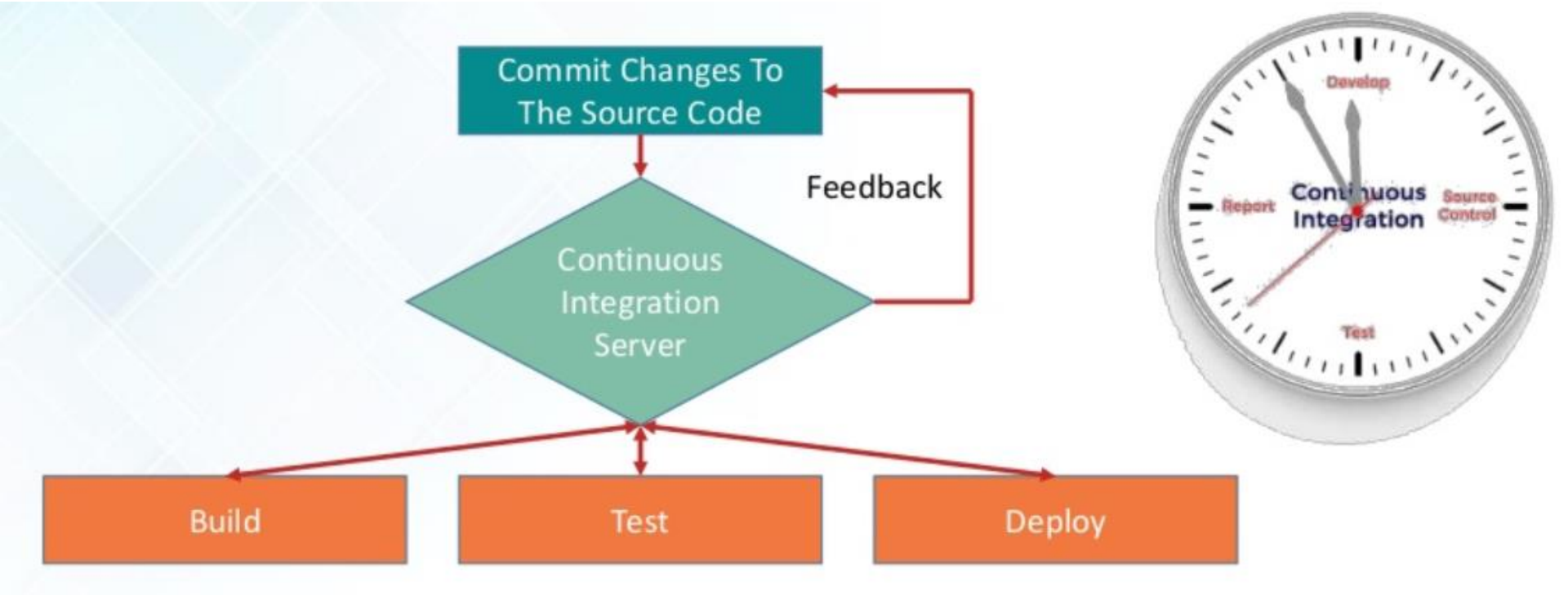
 # Installations:
https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html
https://docs.ansible.com/ansible/latest/user_guide/quickstart.html

 # Git Example:
https://github.com/kapilsthakkar25/ansible-project

CI -
Continuous Integrations

Kapil Thakkar
DevOps Trainer

What is Continuous Integration



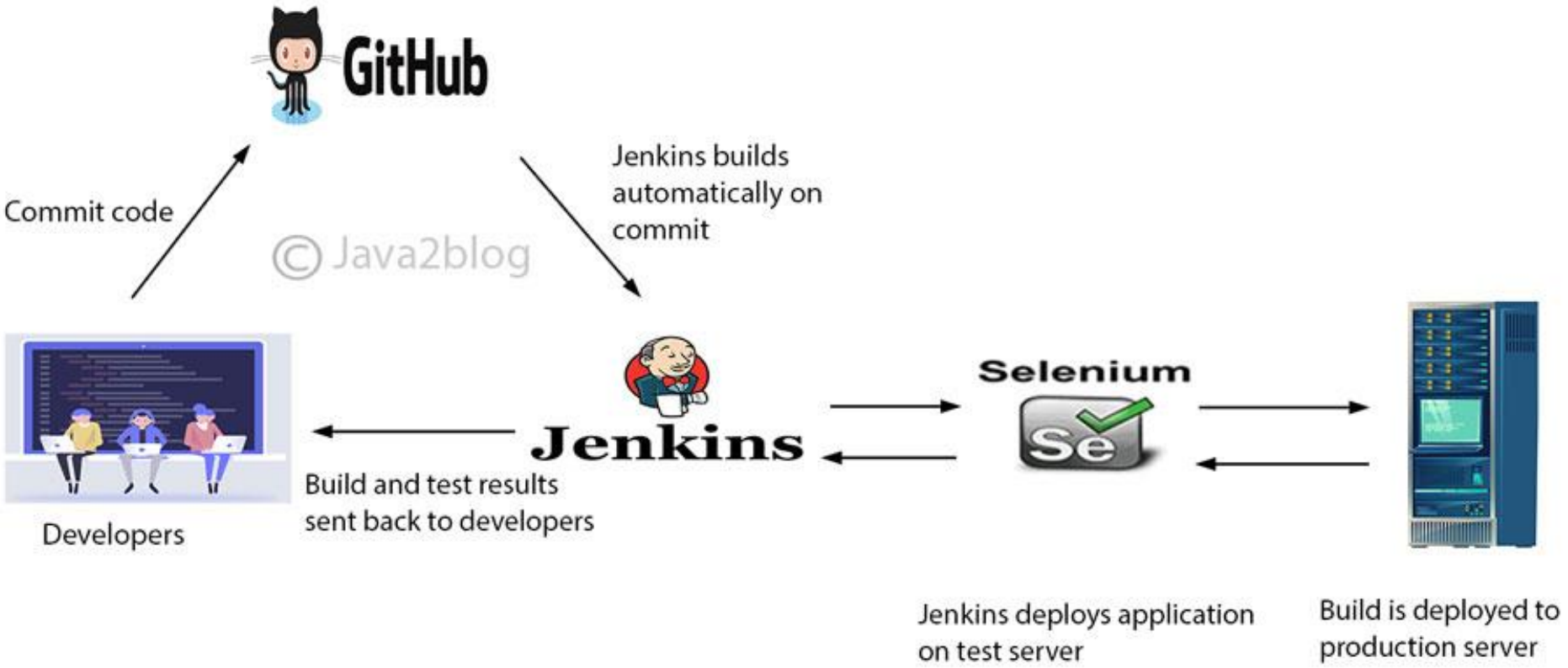
Jenkins Intro

Jenkins is an open source implementation of a CI server written in Java that can be used as a self-hosted option automating the build cycle for any project. It works with any programming language and for multiple platforms including Windows, Linux and macOS.
One of the main benefits of Jenkins is that it is a well-known tool with lots of community support, there are many plugins available (including well-known names like Slack, GitHub, Docker, Build Pipeline + more), and the project is well-maintained by a large community of developers.

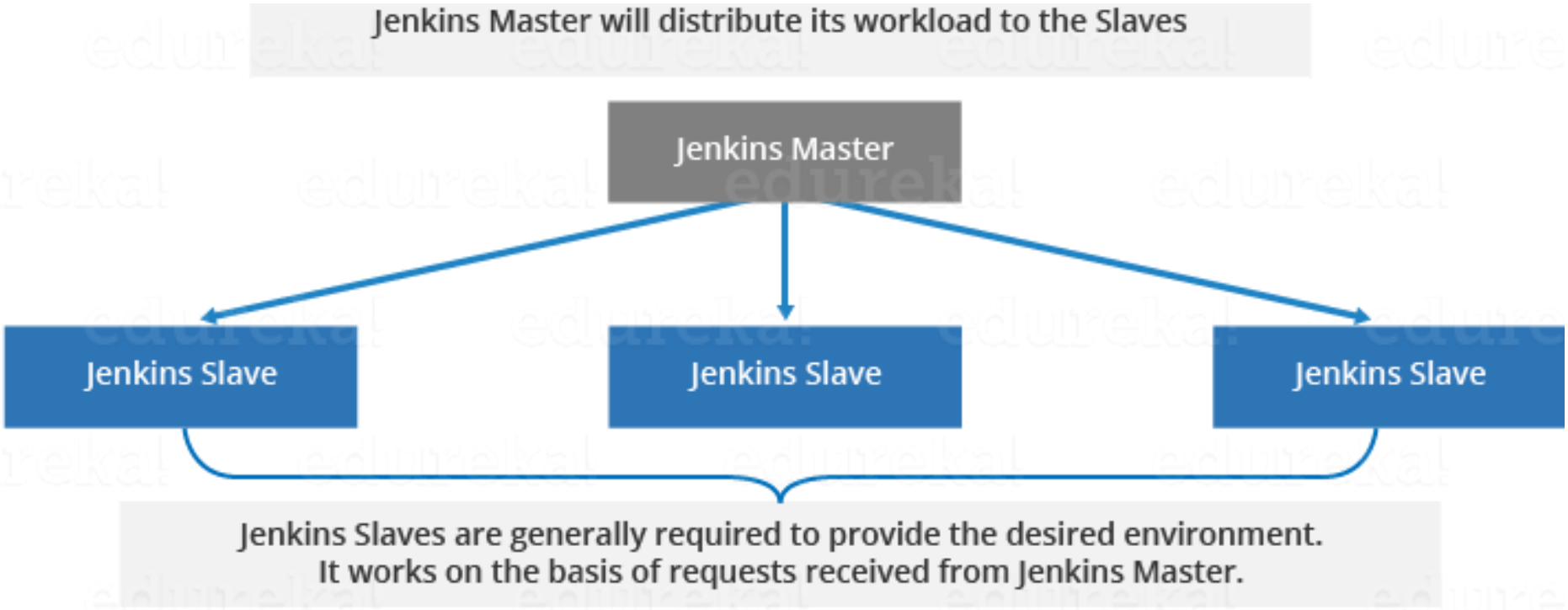
Common Jenkins Plugins

Plugin	Feature
Git Plugin	Git plugins are often used in projects to know whether the codes written are stable or not
SSH Plugin	These plugin is used to run shell commands through SSH on a remote machine as they are derived from SCP plugin
Build Pipeline Plugin	This plugin provides a Build Pipeline View of upstream and downstream connected jobs that typically form a build pipeline
Email-ext Plugin	This plugin allows you to configure every aspect of email notifications. You can customize when an email is sent, who should receive it, and what the email says
HTML Publisher Plugin	This plugin publishes HTML reports
Multi-slave config Plugin	This plugin allows administrators to configure, add and delete several dumb slaves at the same time
Parameterized Trigger Plugin	This plugin lets you trigger new builds when your build has completed, with various ways of specifying parameters for the new build

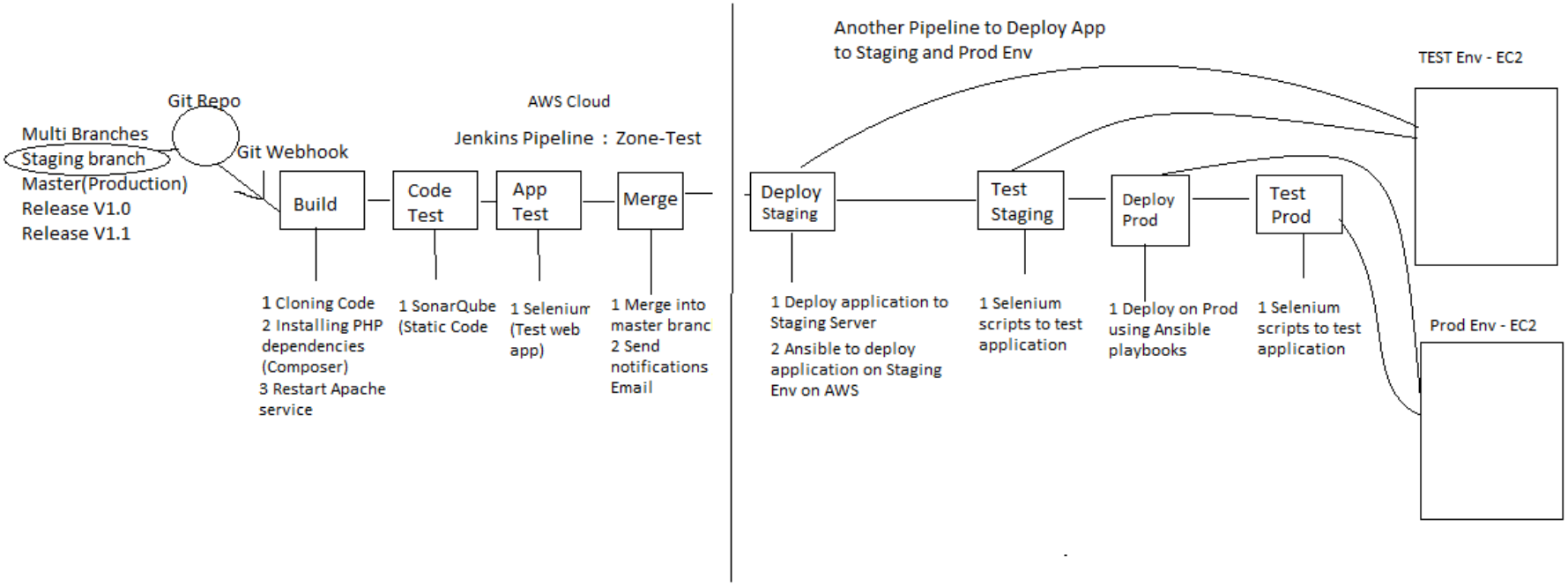
Jenkins with GitHub



Jenkins Master and Slave



Jenkins Pipeline Example



Reference Links

 # Installations:

https://linuxize.com/post/how-to-install-jenkins-on-centos-7/
https://hub.docker.com/_/jenkins

 # Git Example:

https://github.com/kapilsthakkar25/my-app
https://github.com/kapilsthakkar25/jenkins-lab-demo
https://github.com/kapilsthakkar25/pipeline-examples
https://github.com/kapilsthakkar25/simple-java-maven-app
https://github.com/kapilsthakkar25/spring-jenkins
https://github.com/kapilsthakkar25/maven-project

Continuous Monitoring

Kapil Thakkar
DevOps Trainer

Continuous Monitoring

Continuous Monitoring allows timely identification of problems or weaknesses in the software and quick corrective action that helps reduce expenses of an organization.
Continuous monitoring provides solution that addresses three operational disciplines known as:

Continuous Audit

Continuous Controls Monitoring

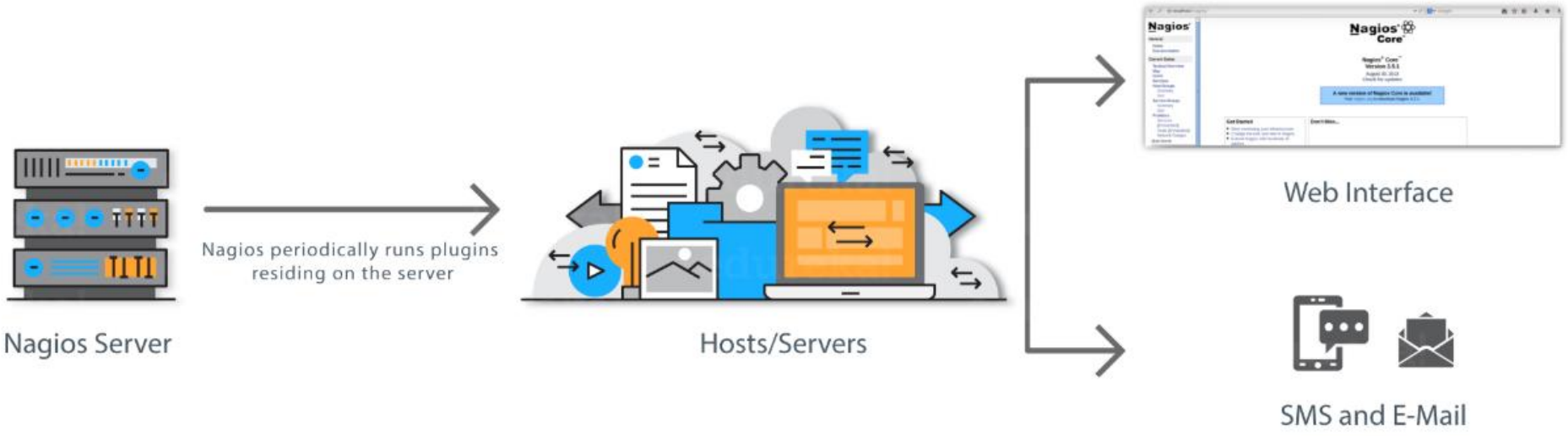
Continuous Transaction Inspection

What is Nagios?

- Nagios is used for Continuous monitoring of systems, applications, services, and business processes etc in a DevOps culture. In the event of a failure, Nagios can alert technical staff of the problem, allowing them to begin remediation processes before outages affect business processes, end-users, or customers. With Nagios, you don’t have to explain why an unseen infrastructure outage affect your organization’s bottom line.

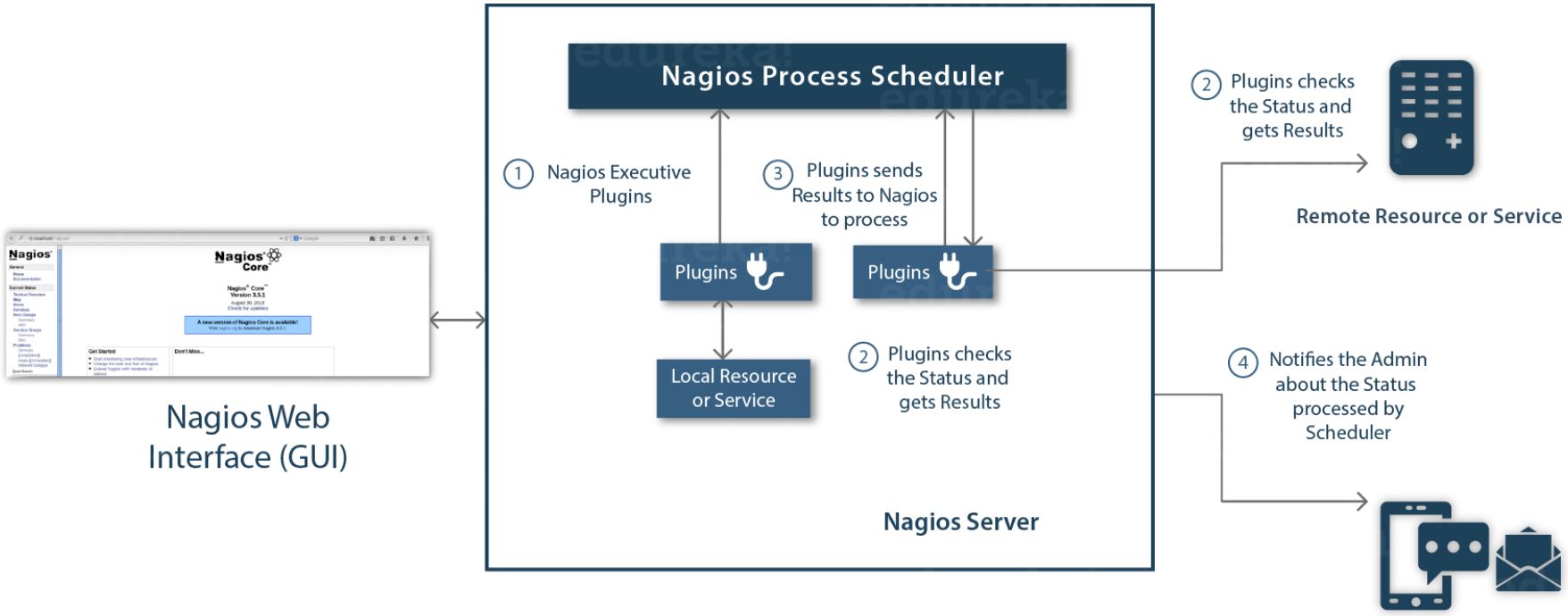
How Nagios works

- Nagios runs on a server, usually as a daemon or service. Nagios periodically runs plugins residing on the same server, they contact hosts or servers on your network or on the internet. One can view the status information using the web interface. You can also receive email or SMS notifications if something happens.
- The Nagios daemon behaves like a scheduler that runs certain scripts at certain moments. It stores the results of those scripts and will run other scripts if these results change.



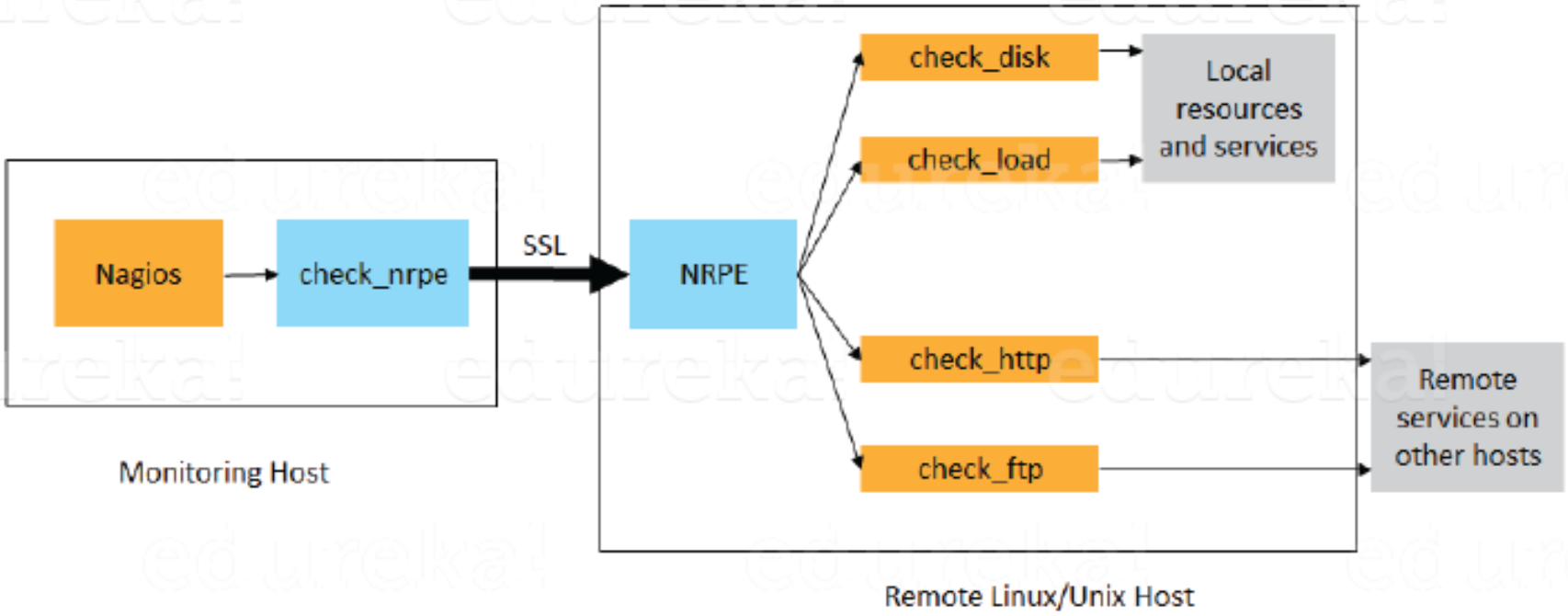
Nagios Architecture

- Nagios is built on a server/agents architecture.
- Usually, on a network, a Nagios server is running on a host, and Plugins interact with local and all the remote hosts that need to be monitored.
- These plugins will send information to the Scheduler, which displays that in a GUI.



What is NRPE

NRPE (Nagios Remote Plugin Executor)
The NRPE addon is designed to allow you to execute Nagios plugins on remote Linux/Unix machines. The main reason for doing this is to allow Nagios to monitor “local” resources (like CPU load, memory usage, etc.) on remote machines. Since these public resources are not usually exposed to external machines, an agent like NRPE must be installed on the remote Linux/Unix machines.



Active and Passive in Nagios

The major difference between Active and Passive checks is that Active checks are initiated and performed by Nagios, while passive checks are performed by external applications.

Passive checks are useful for monitoring services that are:

- ❑ Asynchronous in nature and cannot be monitored effectively by polling their status on a regularly scheduled basis.
- ❑ Located behind a firewall and cannot be checked actively from the monitoring host.

The main features of Actives checks are as follows:

- ❑ Active checks are initiated by the Nagios process.
- ❑ Active checks are run on a regularly scheduled basis.

Reference Links

 # Installations:

https://github.com/kapilsthakkar25/nagios-eample/blob/master/nagios-server-install.txt
https://github.com/kapilsthakkar25/nagios-eample/blob/master/Install-Nagios-Agent-on-client.txt
https://www.nagios.org/
https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/toc.html

 # Git Example:

https://github.com/kapilsthakkar25/nagios-eample

Thanks…!!!

Contact Details:

- Name: Kapil Thakkar (DevOps Trainer)
- Email ID: kapilsthakkar25@gmail.com
- GitHub Repo: https://github.com/kapilsthakkar25