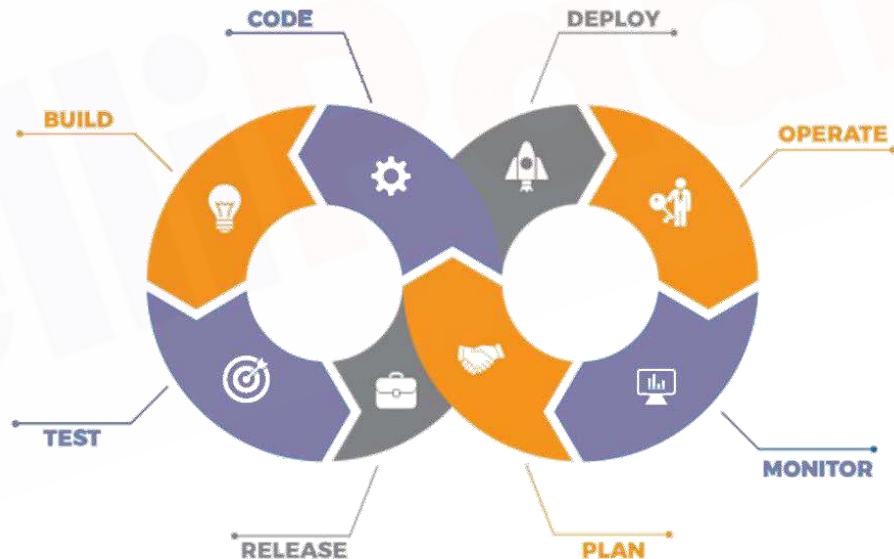


Introduction to DevOps



Agenda

01

WHAT IS
SOFTWARE
DEVELOPMENT?

02

WATERFALL
MODEL

03

AGILE MODEL

04

LEAN MODEL

05

WATERFALL VS
AGILE VS LEAN

06

WHY DEVOPS?

07

WHAT IS DEVOPS?

08

DEVOPS
LIFECYCLE

09

DEVOPS TOOLS

What is Software Development?

What is Software Development?

Software Development is the process of transforming customer requirements into a complete software product.



Software Development Life Cycle

In broader terms, software development involves the following stages:



Requirements

Design

Implementation

Verification

Maintenance

Software Development Life Cycle

Requirements

Design

Implementation

Verification

Maintenance

This is the most important phase in the software development lifecycle. In this stage, the requirements are gathered from the customers and the requirements are then analysed to ensure their feasibility.



Software Development Life Cycle

Requirements

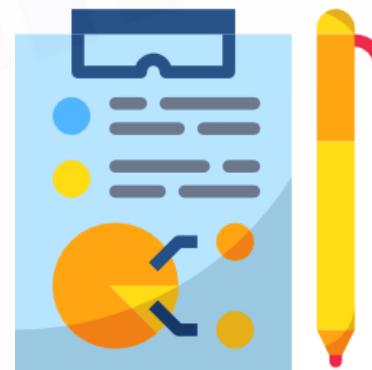
Design

Implementation

Verification

Maintenance

Once the requirements are received, the architect transforms these requirements into technical specifications and plan the software components which have to be designed



Software Development Life Cycle

Requirements

Design

Implementation

Verification

Maintenance

The specifications are then passed on to the developers which create the application based on these specifications



Software Development Life Cycle

Requirements

Design

Implementation

Verification

Maintenance

Once the development work is done on the application. It is verified by a group of testers to map the application's functionalities with the specification given by customers



Software Development Life Cycle

Requirements

Design

Implementation

Verification

Maintenance

Once the code is verified, it is pushed to production. Post this, the application is updated with any future enhancements or optimizations, if and when required.



Since the time software development started, various software development models have been curated which implement SDLC. Each of these models solve problems that existed before these models were invented.

Traditionally, there have been 3 major software development models that most companies follow:

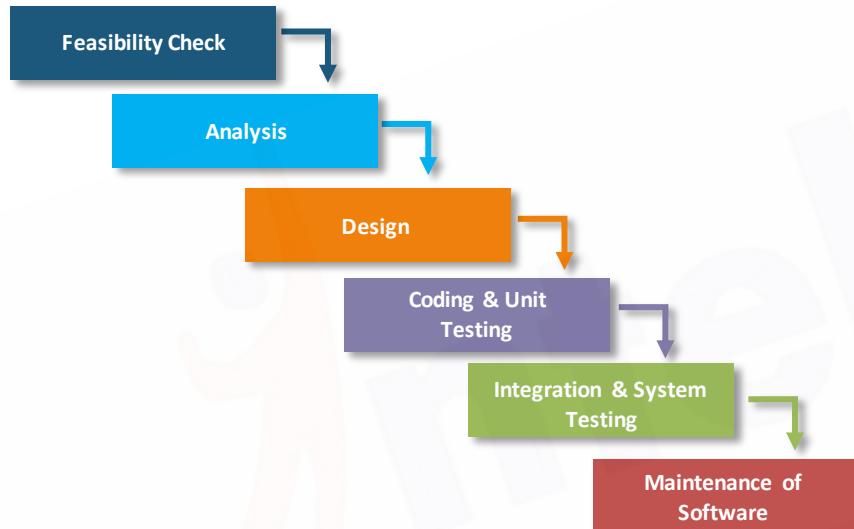
Waterfall Model

Agile Model

Lean Model

Waterfall Model

Waterfall Model



- ★ Waterfall Model was among the first development models which followed SDLC
- ★ The Waterfall model follows a linear sequential model of development i.e until the first stage is not finished, the next stage will not start

Advantages of Waterfall Model



- ✓ Clear Objectives
- ✓ Specific Deadlines
- ✓ No ambiguous requirements
- ✓ Well understood milestones
- ✓ Process and results are well documented

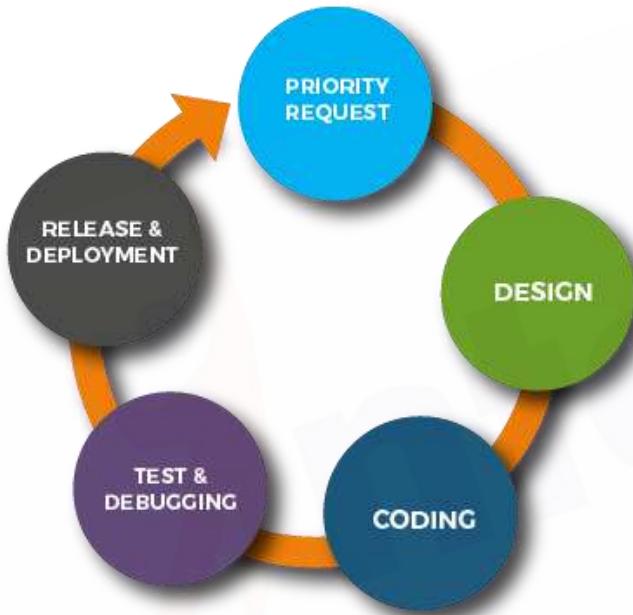
Disadvantages of Waterfall Model



- ✖ Working Product is not available until the later stage in lifecycle
- ✖ Poor model for large and complex projects
- ✖ Cannot accommodate changing requirements
- ✖ High risk and uncertainty

Agile Model

Agile Model



- ★ To overcome the challenges faced in the Waterfall Model, we came up with the Agile Methodology
- ★ Agile Method believes in creating shorter development lifecycles
- ★ Shorter Development Lifecycles are achieved by not releasing all the features at once by following an incremental model of development

Advantages of Agile Model



-  Customer Satisfaction is high
-  Less Planning Required
-  Requirements can be dynamic in nature
-  Functionality can be created and tested quickly

Disadvantages of Agile Model



- ✖ Not suitable for handling complex dependencies in projects
- ✖ Knowledge transfer to colleagues can be difficult since there is little documentation
- ✖ Success of the project depends heavily on customer interaction

Lean Model

7 Principles of Lean Methodology

- 🎯 Eliminate Waste
- 🎯 Amplify Learning
- 🎯 Decide as late as possible
- 🎯 Deliver as fast as possible
- 🎯 Empower the team
- 🎯 Build Integrity
- 🎯 See the whole

- ★ Lean development is a philosophy of increasing quality in software delivery by making use of agile methods
- ★ Ignore the clutter for later and focus on what is required now
- ★ Lean Methodology has it's primary focus on two things – Respect for frontline workers and Continuous Improvement

Advantages of Lean Model



- ✓ Carries the same advantages as Agile Methodology
- ✓ Creates a positive working environment
- ✓ Customer Feedback is given the utmost importance
- ✓ Limiting Wastes saves time and money

Disadvantages of Lean Model



- ✖ Largely dependent on the skill set of the team, therefore requires a strong team
- ✖ No room for error, a missed delivery can be bad for business
- ✖ Success of the project depends heavily on customer interaction

Waterfall vs Agile vs Lean

Waterfall vs Agile vs Lean

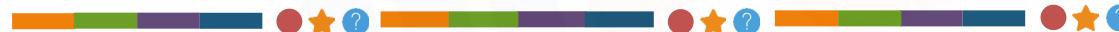
Waterfall Model



Agile Model



Lean Model



-  Requirements
-  Design
-  Implementation
-  Verification
-  Release
-  Customer Feedback
-  Eliminate Waste

Summarizing

Problem with Waterfall Model was, the development lifecycle took a lot of time to complete. Therefore, by the time finished product was delivered, the customer requirements were no longer the same.



Customers



Software Company

Summarizing

This problem was fixed by Lean and Agile methodologies. These methodologies strictly focussed on customer feedback and improving the software quality that too in a shorter development lifecycle



Customers



Software Company

Summarizing

This problem was fixed by Lean and Agile methodologies. These methodologies strictly focussed on customer feedback and improving the software quality that too in a shorter development lifecycle



Customers

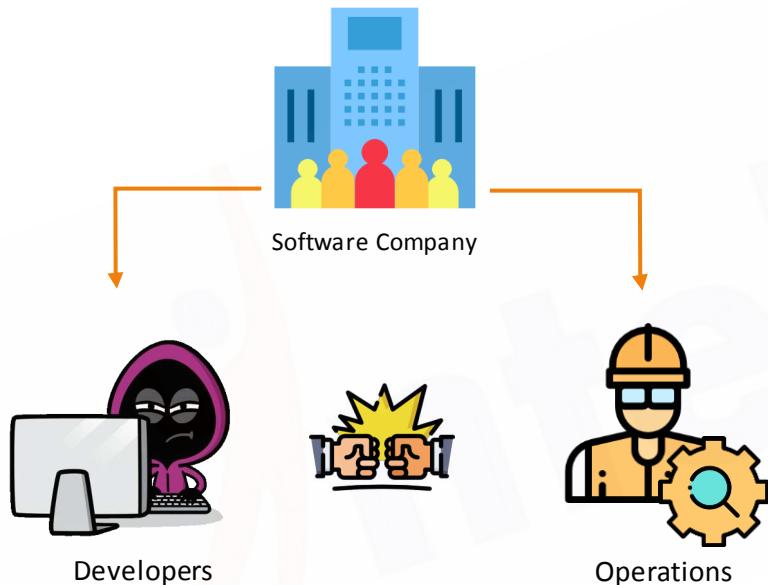


Software Company

Why do we need DevOps?

Why DevOps?

Why DevOps?



Although, the software quality was improved. We still had a lack of efficiency among the development team. A typical software development team consists of Developers and Operations employees. Let us understand their job roles

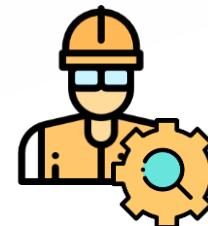
Why DevOps?

A developer's job is to develop applications and pass his code to the operations team



Developer

The operations team job is to test the code, and provide feedback to developers in case of bugs. If all goes well, the operations team uploads the code to the build servers



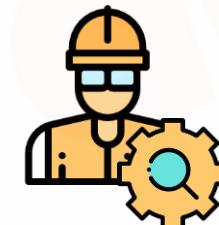
Operations

Why DevOps?



Developer

The developer used to run the code on his system, and then forward it to operations team.



Operations

The operations when tried to run the code on their system, it did not run!

Why DevOps?



Developer

But, the code runs fine on the developer's system and hence he says "It is not my fault!"



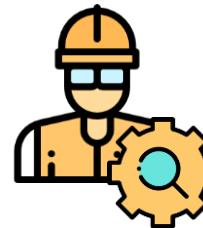
Operations

The operations then marked this code as faulty, and used to forward this feedback to the developer

Why DevOps?



Developer



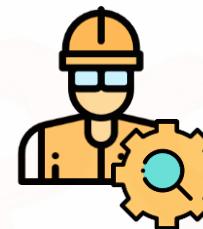
Operations

This led to a lot of back and forth between the developer and the operations team, hence impacted efficiency.

Why DevOps?



Developer



Operations

This problem was solved using Devops!

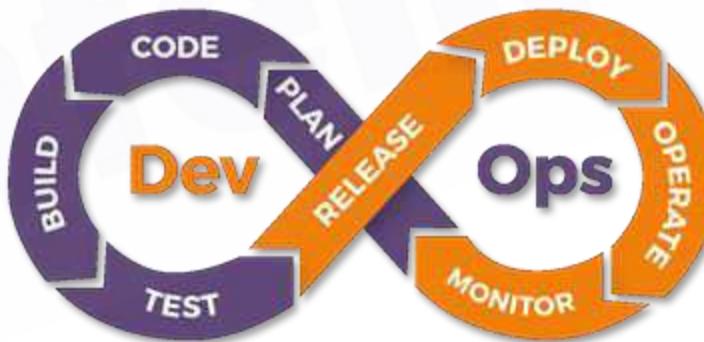
Traditional IT vs DevOps

Traditional IT	Devops
Less Productive	More Productive
Skill Centric Team	Team is divided into specialized silos
More Time invested in planning	Smaller and Frequent releases lead to easy scheduling and less time in planning
Difficult to achieve target or goal	Frequent releases, with continuous feedback makes achieving targets easy

What is Devops?

What is DevOps?

Devops is a software development methodology which improves the collaboration between developers and operations team using various automation tools. These automation tools are implemented using various stages which are a part of the Devops Lifecycle

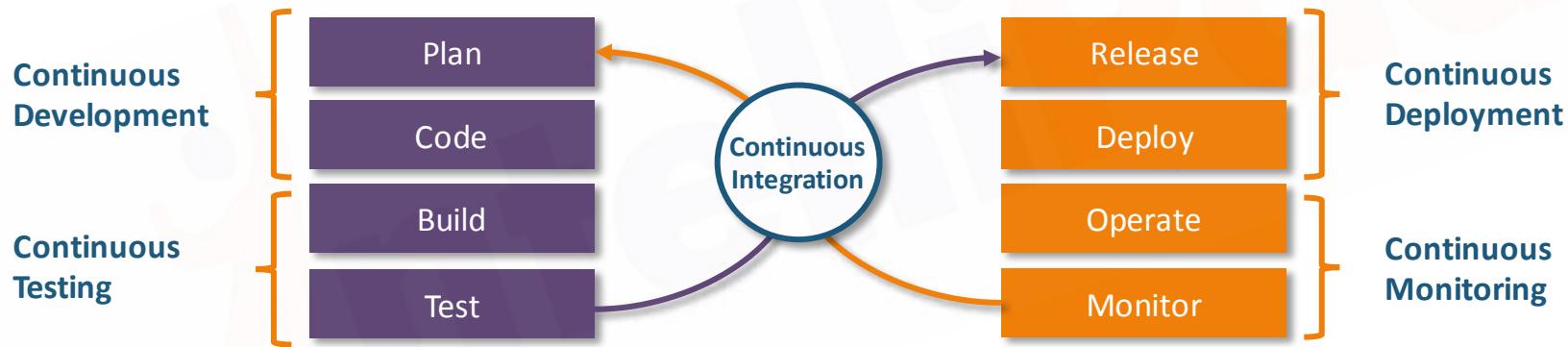


DevOps Lifecycle

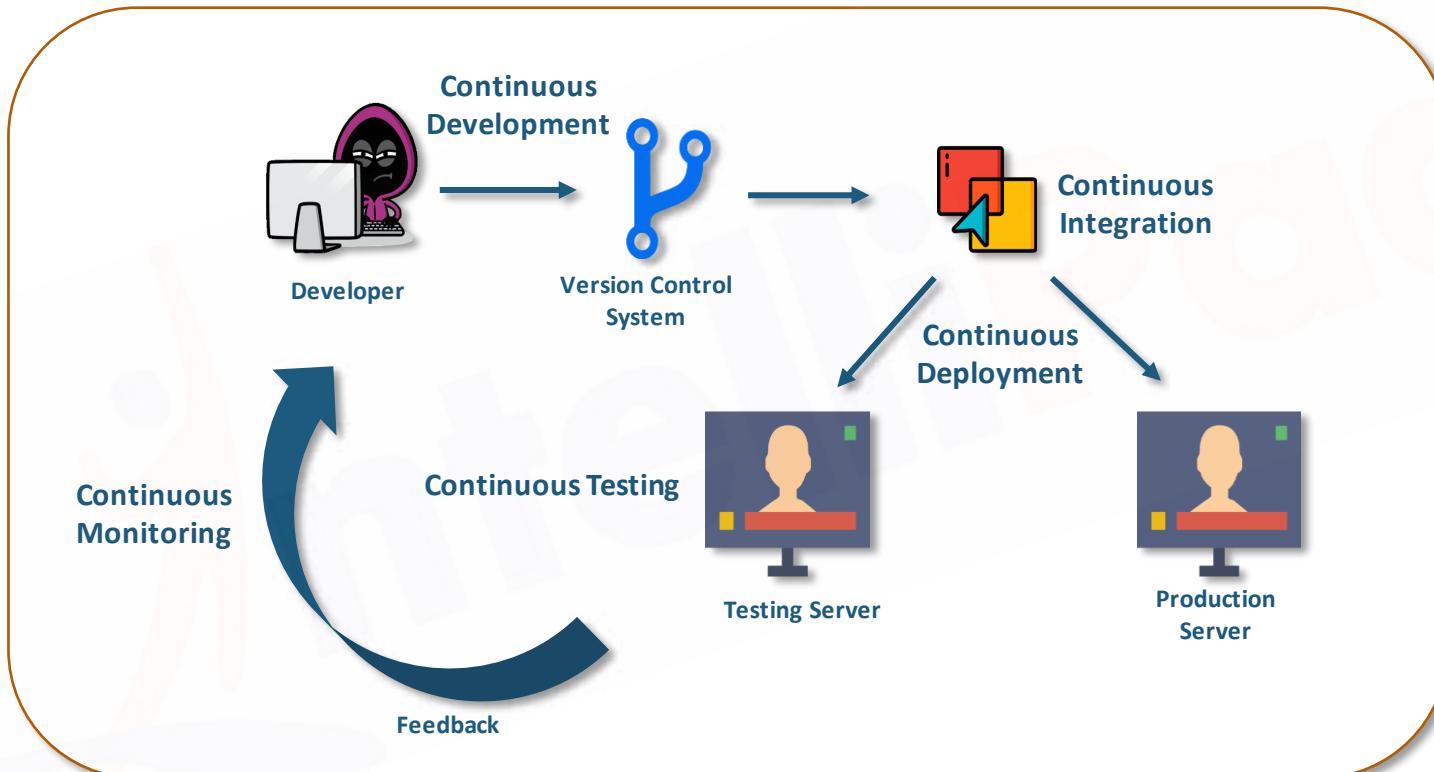


How DevOps Works?

The Devops Lifecycle divides the SDLC lifecycle into the following stages:



How DevOps Works?



Automated CI/CD Pipeline

How DevOps Works?

Continuous Development

Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

This stage involves committing code to version control tools such as **Git** or **SVN** for maintaining the different versions of the code, and tools like **Ant**, **Maven**, **Gradle** for building/ packaging the code into an executable file that can be forwarded to the QAs for testing.



How DevOps Works?

Continuous Development

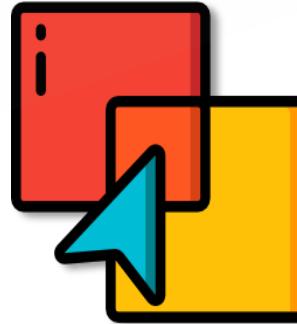
Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

The stage is a critical point in the whole Devops Lifecycle. It deals with integrating the different stages of the devops lifecycle, and is therefore the key in automating the whole Devops Process



How DevOps Works?

Continuous Development

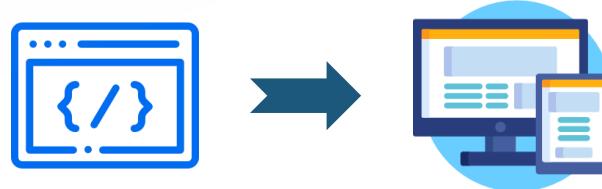
Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

In this stage the code is built, the environment or the application is containerized and is pushed on to the desired server. The key processes in this stage are Configuration Management, Virtualization and Containerization



How DevOps Works?

Continuous Development

Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

The stage deals with automated testing of the application pushed by the developer. If there is an error, the message is sent back to the integration tool, this tool in turn notifies the developer of the error. If the test was a success, the message is sent to Integration tool which pushes the build on the production server



How DevOps Works?

Continuous Development

Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

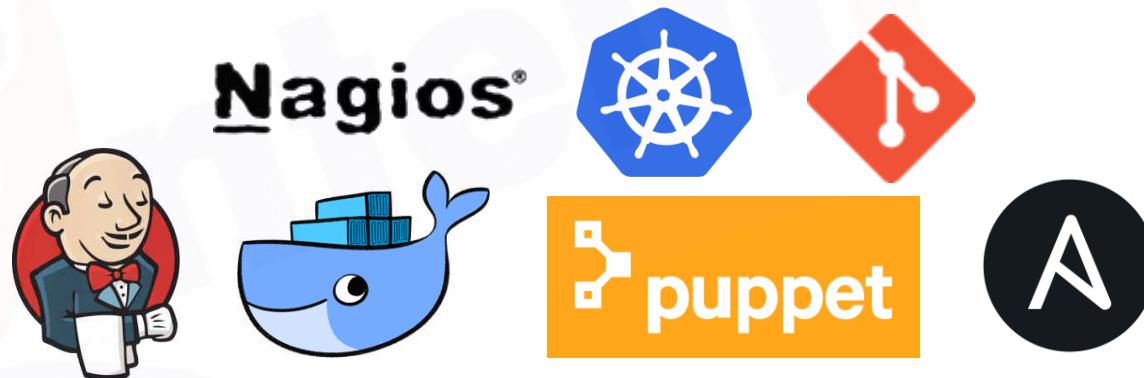
The stage continuously monitors the deployed application for bugs or crashes. It can also be setup to collect user feedback. The collected data is then sent to the developers to improve the application



Devops Tools

DevOps Tools

We have discussed the Devops Methodology, but this methodology cannot be put into action without it's corresponding tools. Let us discuss the devops tools with their respective lifecycle stages



DevOps Tools

Continuous Development

Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

Git is a distributed version-control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of files



DevOps Tools

Continuous Development

Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

Jenkins is an open source automation server written in Java.

Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery



DevOps Tools

Continuous Development

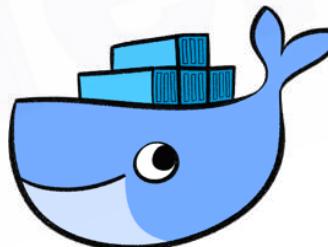
Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

Continuous Deployment



DevOps Tools

Continuous Development

Continuous Integration

Continuous Deployment

Continuous Testing

Continuous Monitoring

Selenium is a portable software-testing framework used for web applications. It is an open source tool which is used for automating the tests carried out on web browsers (Web applications are tested using any web browser).



DevOps Tools

Continuous Development

Continuous Integration

Continuous Deployment

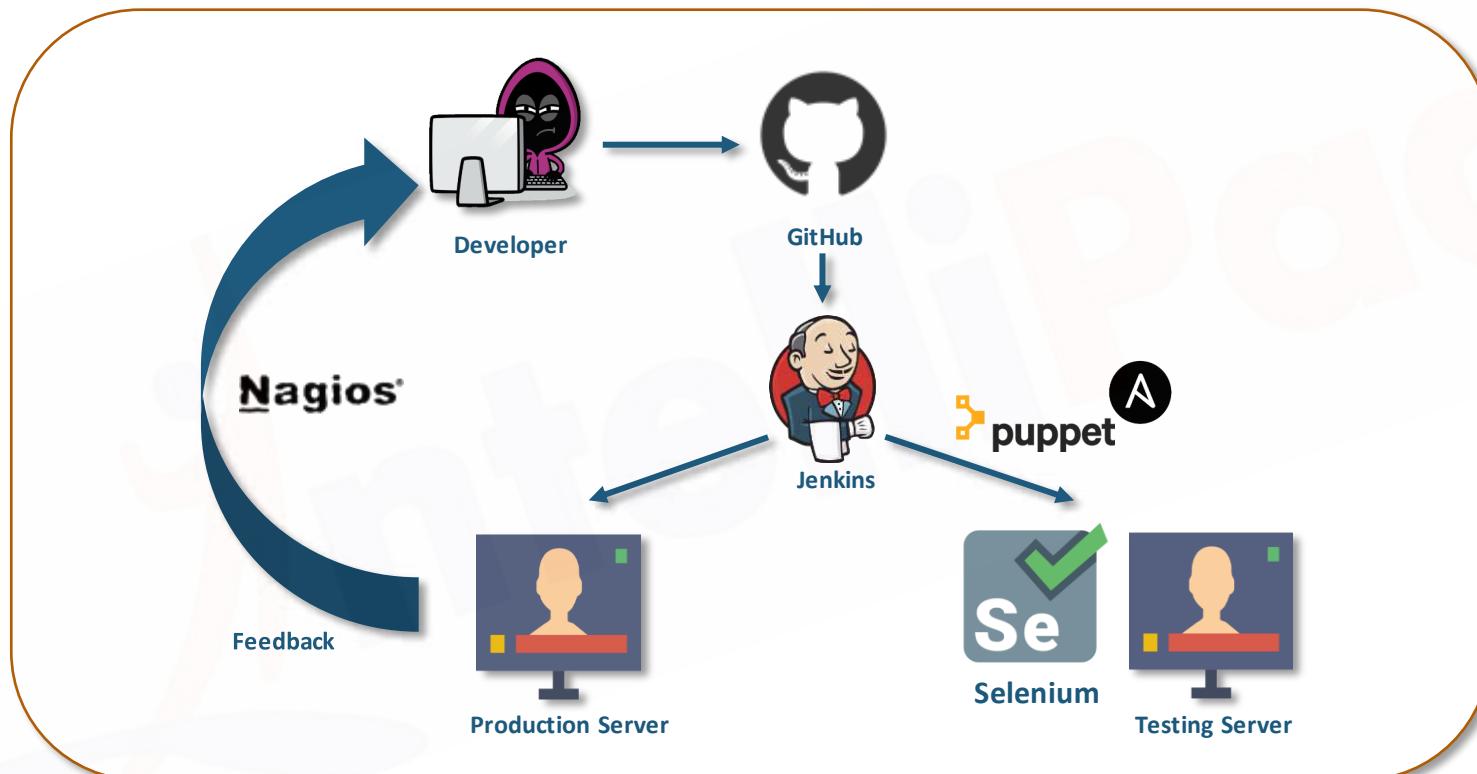
Continuous Testing

Continuous Monitoring

Nagios is an open-source devops tool which is used for monitoring systems, networks and infrastructure. It also offers monitoring and alerting services for any configurable event.

The Nagios logo consists of the word "Nagios" in a large, bold, black, sans-serif font. A registered trademark symbol (®) is positioned at the top right of the letter "o".

DevOps Tools



Quiz

1. Which of these Software Development Methodologies are not suitable for large and complex projects?

- A. Waterfall Model
- B. Devops
- C. Agile Methodology
- D. None of these

1. Which of these Software Development Methodologies are not suitable for large and complex projects?

A. Waterfall Model

B. Devops

C. Agile Methodology

D. None of these

2. Devops Methodology was focused on solving the problems between the customers and the software company.

A. True

B. False

Quiz

2. Devops Methodology was focused on solving the problems between the customers and the software company.

A. True

B. False

3. Which of these principles are NOT included in Agile Methodologies?

A. Frequent Release Cycles

B. Focus on Customer Feedback

C. Eliminating Waste

D. None of these

3. Which of these principles are NOT included in Agile Methodologies?

A. Frequent Release Cycles

B. Focus on Customer Feedback

C. Eliminating Waste

D. None of these

4. Which Lifecycle stage in Devops helps in Transition from one stage to another?

- A. Continuous Development
- B. Continuous Testing
- C. Continuous Monitoring
- D. Continuous Integration

4. Which Lifecycle stage in Devops helps in Transition from one stage to another?

- A. Continuous Development
- B. Continuous Testing
- C. Continuous Monitoring
- D. Continuous Integration**

5. Which tool among the following helps in containerization?

A. Jenkins

B. Git

C. Kubernetes

D. Docker

Quiz

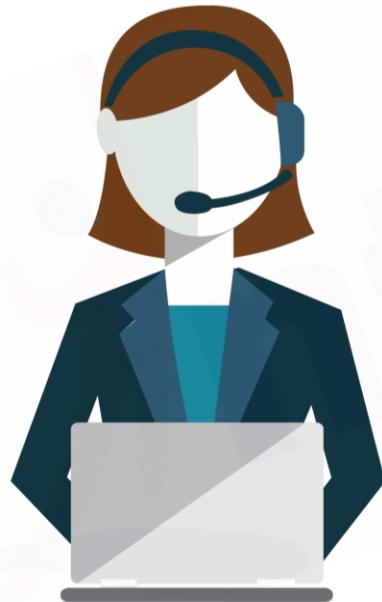
5. Which tool among the following helps in containerization?

A. Jenkins

B. Git

C. Kubernetes

D. Docker



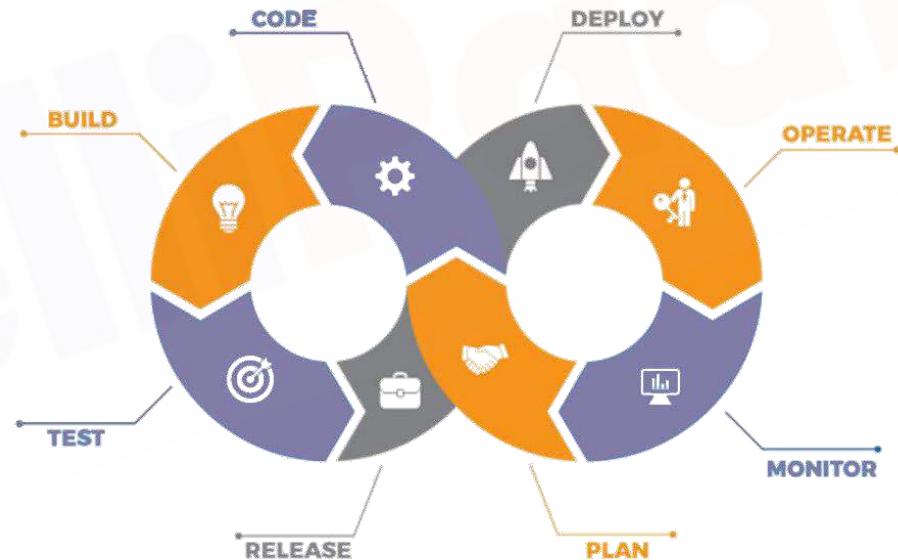
India : +91-7847955955

US : 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24X7 Chat with our Course Advisor

Version Control with GIT



Agenda

01

WHAT IS VERSION
CONTROL?

02

TYPES OF
VERSION
CONTROL SYSTEM

03

INTRODUCTION TO
GIT

04

GIT LIFECYCLE

05

COMMON GIT
COMMANDS

06

MERGING IN GIT

07

RESOLVING
MERGE
CONFLICTS

08

GIT WORKFLOW

What is Version Control?

What is Version Control?

Version control is a system that records/manages changes to documents, computer programs etc over time. It helps us tracking changes when multiple people work on the same project



Problems before Version Control

Imagine, Developer A creates a software, and starts a company with this software.



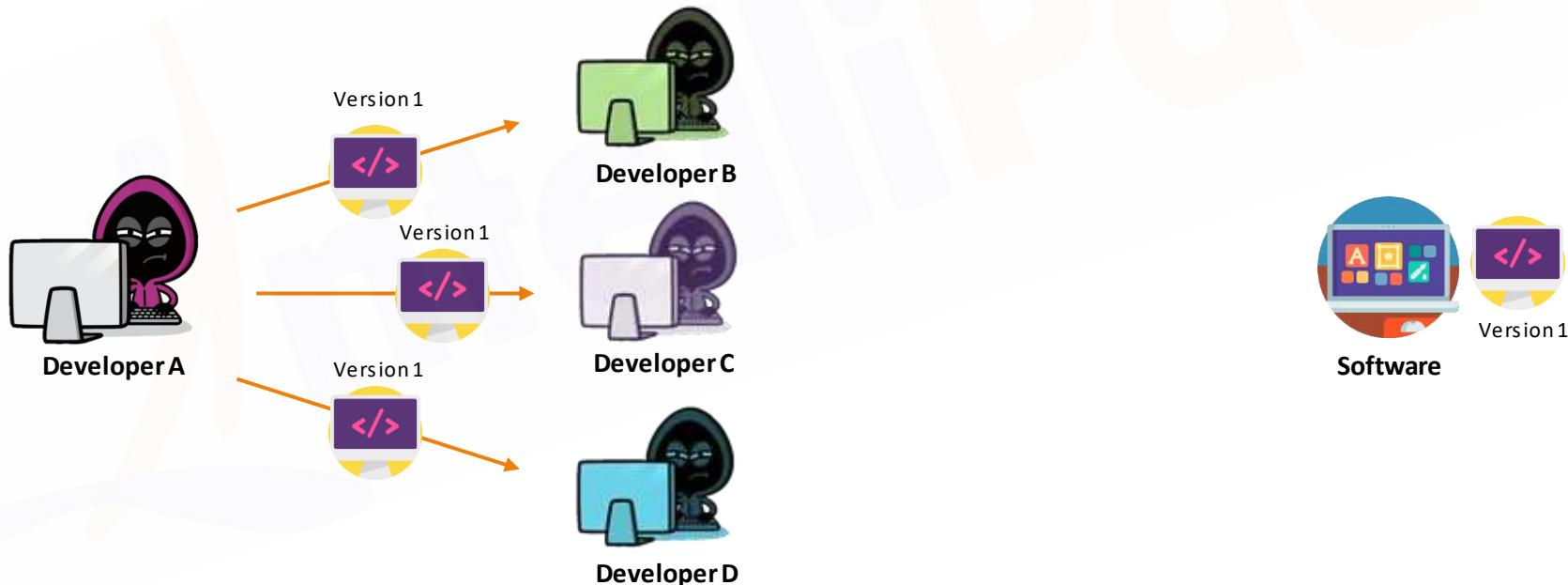
Developer A



Software

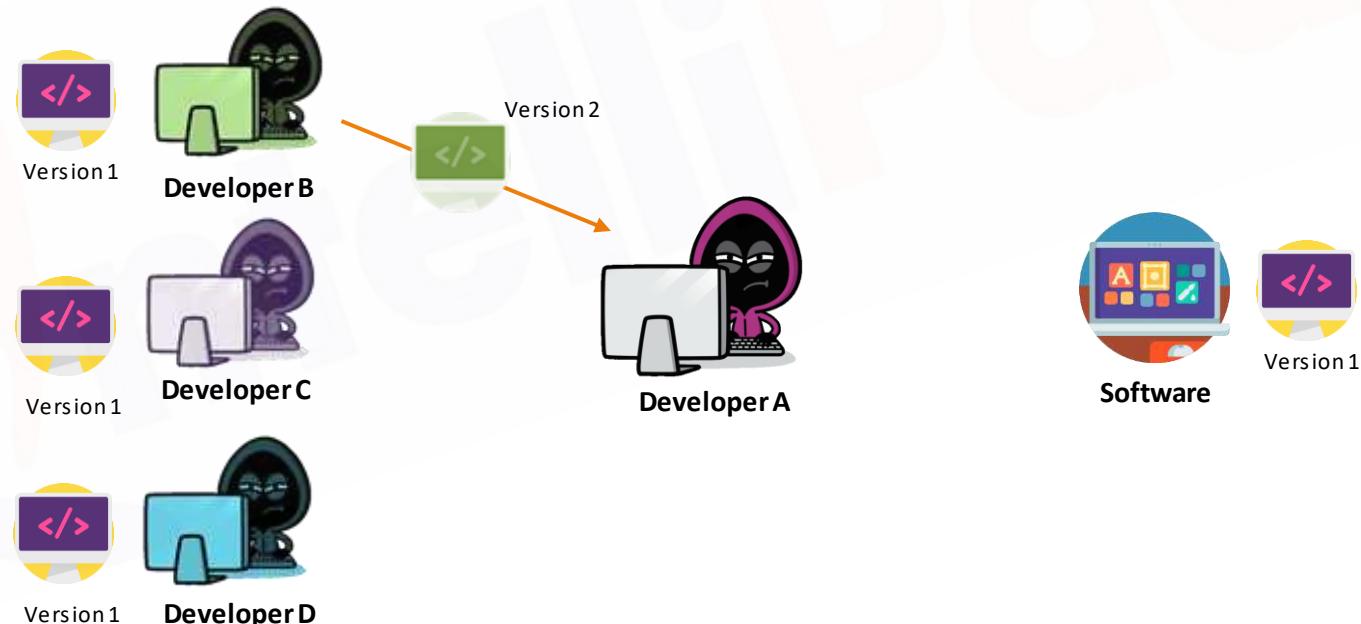
Problems before Version Control

As the company grows, Developer A hires more people to enhance the features of this software. Developer A shares the source code copy with each one of them to work on



Problems before Version Control

Developer B, enhances the software with a feature and submits it to Developer A



Problems before Version Control

Developer A, verifies the changes, and if all looks well, simply replaces the code of the main software



Version 1



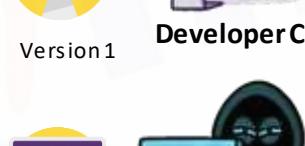
Developer B



Version 1



Developer C



Version 1



Developer D



Developer A



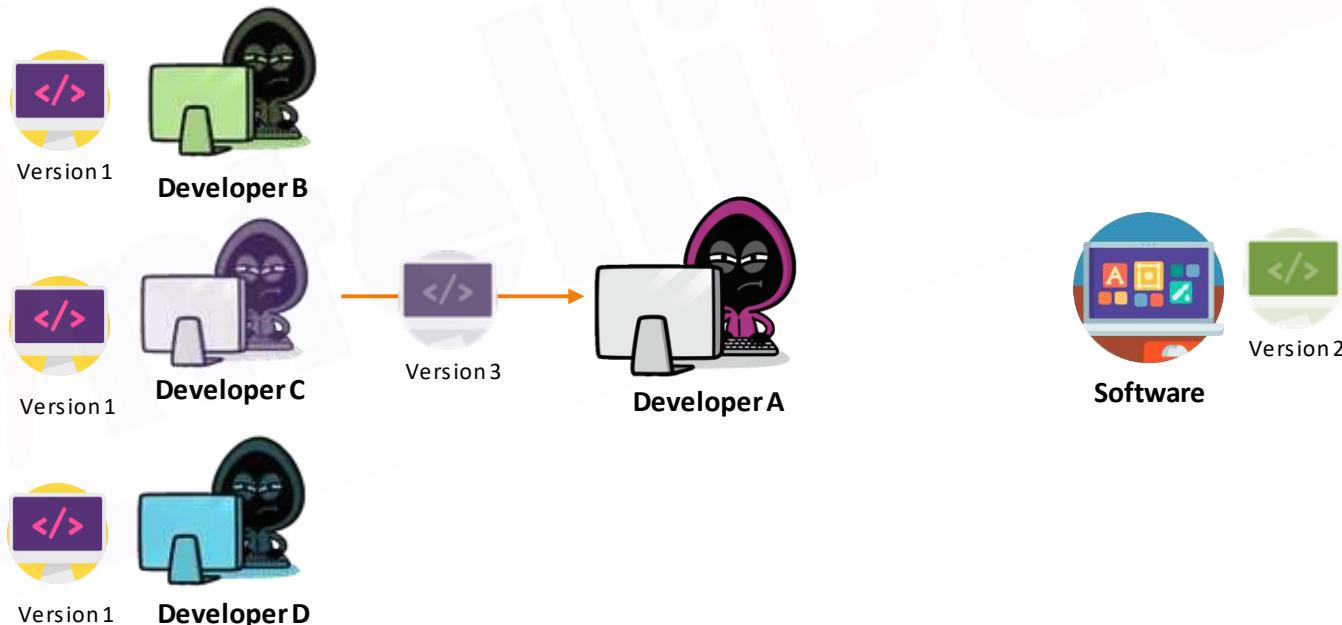
Software



Version 2

Problems before Version Control

Now, the problem starts here, Developer C also finished his work, and submits the changes to Developer A. But, Developer C worked on the code of Version 1.



Problems before Version Control

Developer A verifies the features, takes the code changes and manually integrates them with Version 2 code



Version 1



Developer B



Version 1



Developer C



Version 1



Developer D



Developer A



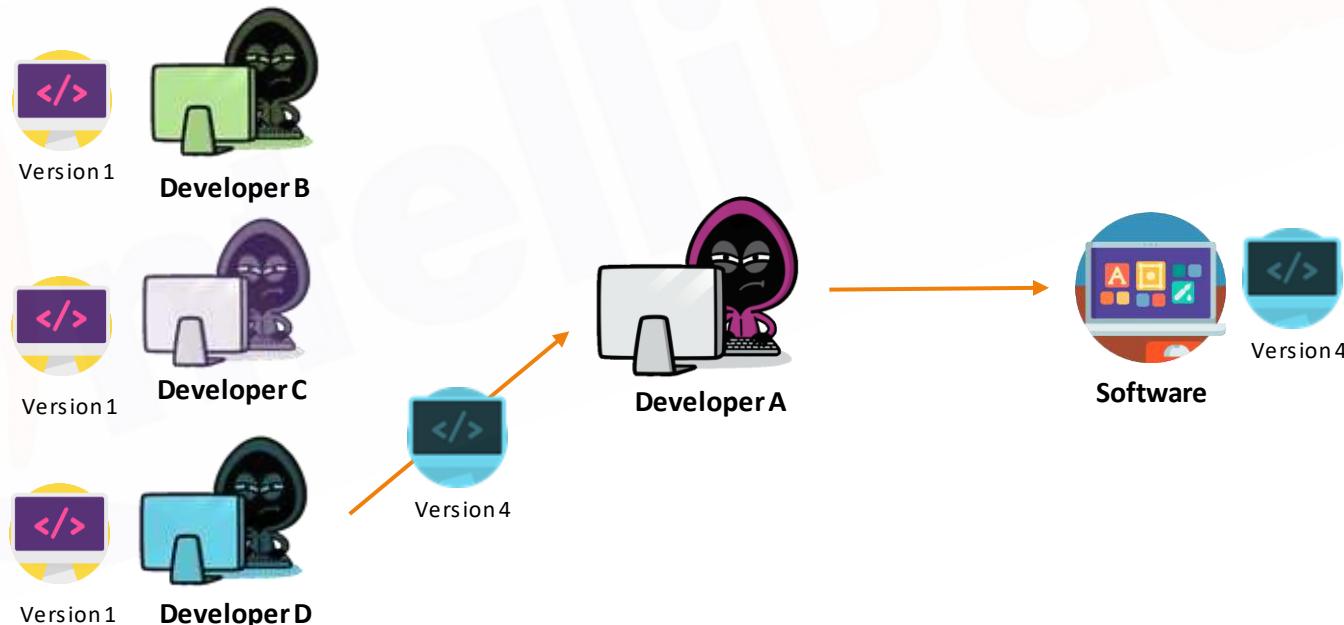
Software



Version 3

Problems before Version Control

Similarly when Developer C is done with his work, submits the work to Developer A. Developer A verifies it, manually integrates the changes with Version 3



Problems before Version Control



- ✖ Versioning was Manual
- ✖ Team Collaboration was a time consuming and hectic task
- ✖ No easy access to previous versions
- ✖ Multiple Version took a lot of space

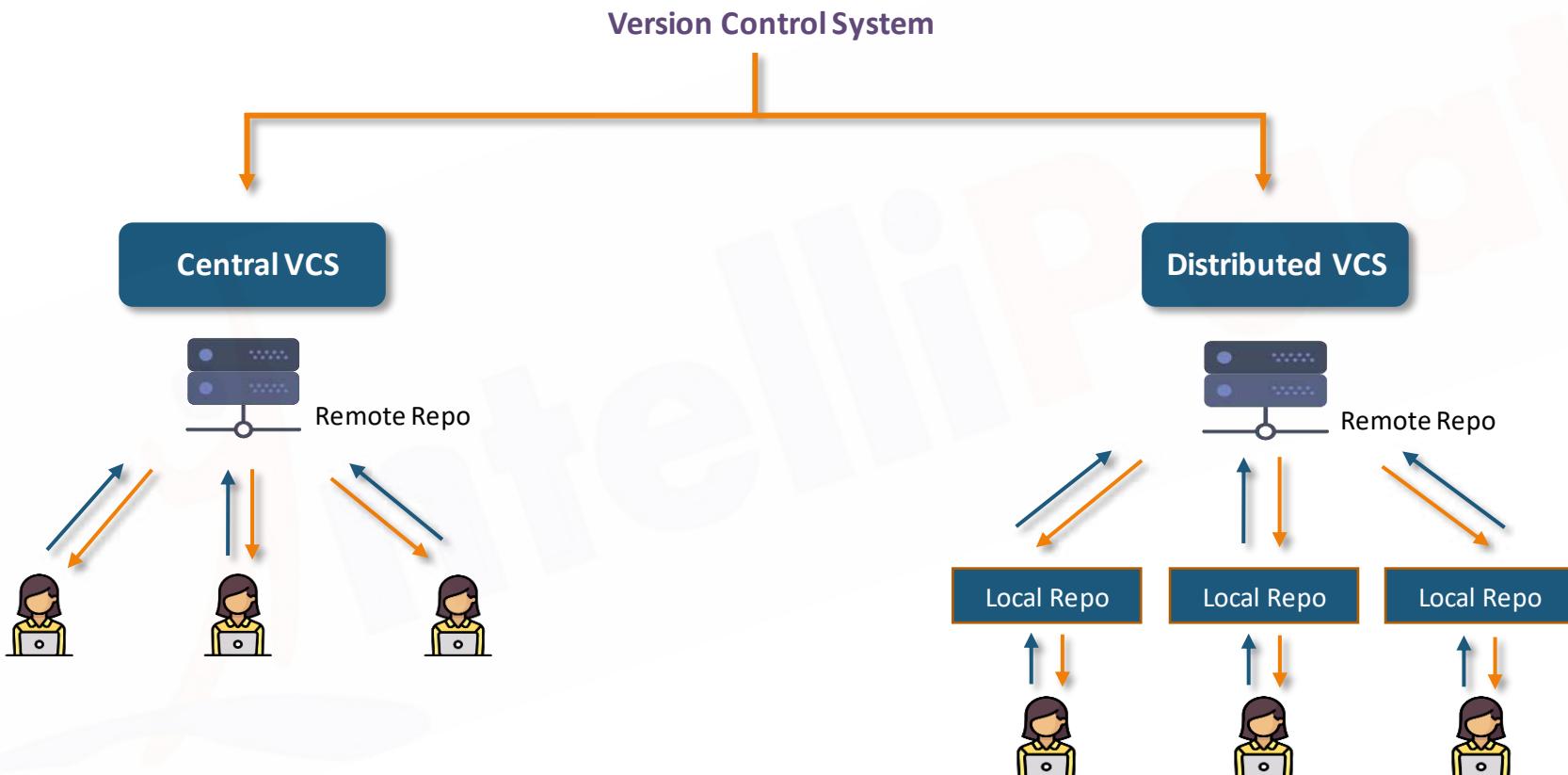
Advantages of Version Control



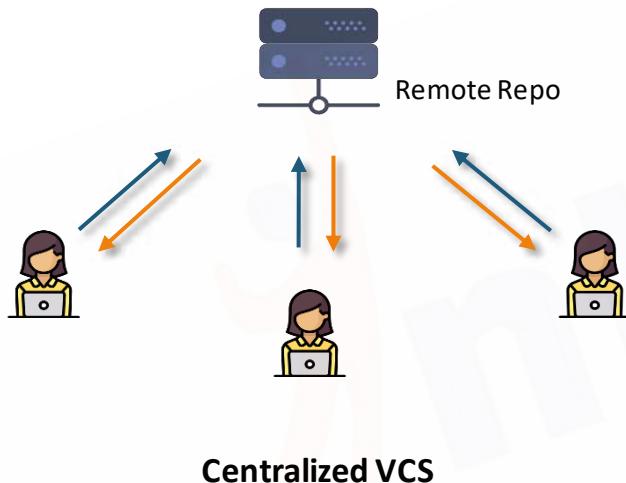
- ✓ Versioning is Automatic
- ✓ Team Collaboration is simple
- ✓ Easy Access to previous Versions
- ✓ Only modified code is stored across different versions, hence saves storage

Types of Version Control System

Types of Version Control System

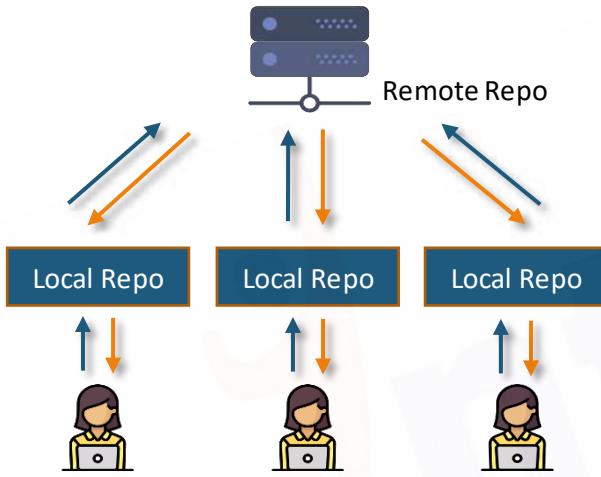


Centralized Version Control System



- ★ Centralized Version Control System has one single copy of code in the central server
- ★ Developers will have to “commit” their changes in the code to this central server
- ★ “Committing” a change simply means recording the change in the central system

Distributed Version Control System



Distributed VCS

- ★ In Distributed VCS, one does not necessarily rely on a central server to store all the versions of a project's file
- ★ Every developer “clones” a copy of the main repository on their local system
- ★ This also copies, all the past versions of the code on the local system too
- ★ Therefore, the developer need not be connected to the internet to work on the code

Difference between DVCS and CVCS

Distributed VCS

- ★ Everything except pushing and pulling can be done without Internet Connection
- ★ Every Developer has full version history on local hard drive
- ★ Committing and retrieving action is faster since data is on local drive
- ★ Not Good for storing large files which are binary in nature, this would increase the repo size at every commit
- ★ If a project has a lot of commits, downloading them may take a lot of time

Centralized VCS

- ★ Needs a dedicated internet connection for every operation
- ★ Developers just have the working copy and no version history on their local drive
- ★ Committing and retrieving action is slower since it happens on the internet
- ★ Good for storing large files, since version history is not downloaded
- ★ Not dependent on the number of commits

Examples of CVCS



Helix**Core**

What is SVN?

- ★ Apache Subversion is a software versioning and revision control system distributed as open source under the Apache License
- ★ It is based on Centralized Version Control Architecture
- ★ The development started in 2000, and this version finally became available in 2004
- ★ It is still constantly being developed by a small but active open source community



Disadvantages of SVN

- ✖ Constantly needs an Internet Connection for any operation
- ✖ Version History is not downloaded or maintained on the local system
- ✖ Slower than DVCS, since requires internet for every operation
- ✖ Conflicts have to be resolved manually



Examples of DVCS

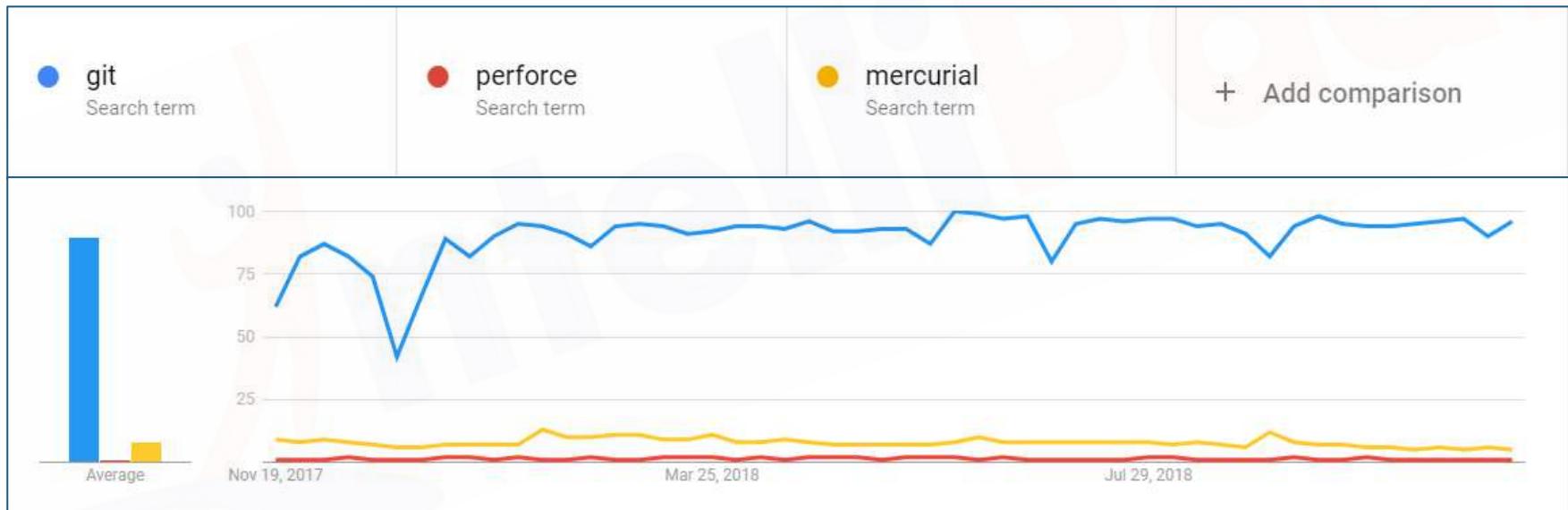
PERFORCE



Introduction to Git

Why Git?

Git is the most popular tool among all the DVCS tools.



What is Git?

Git is a version-control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of files.



Git Lifecycle

Git Lifecycle

Following are the lifecycle stages of files in Git

Working
Directory



Staging
Area



Commit



Git Lifecycle

Working Directory

Staging Area

Commit

- ★ The place where your project resides in your local disk
- ★ This project may or may not be tracked by git
- ★ In either case, the directory is called the working directory
- ★ The project can be tracked by git, by using the command *git init*
- ★ By doing *git init*, it automatically creates a hidden .git folder

Git Lifecycle

Working Directory

Staging Area

Commit

- ★ Once we are in the working directory, we have to specify which files are to be tracked by git
- ★ We do not specify all files to be tracked in git, because some files could be temporary data which is being generated while execution
- ★ To add files in the staging area, we use the command *git add*

Git Lifecycle

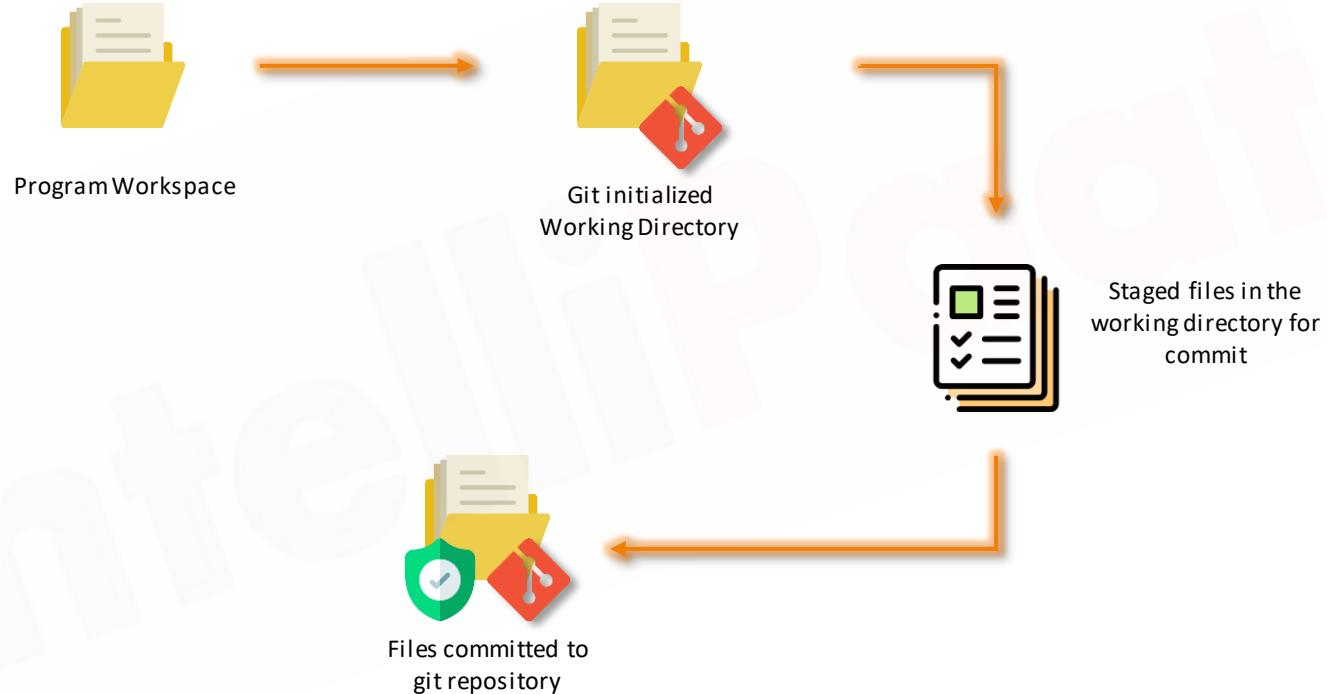
Working Directory

Staging Area

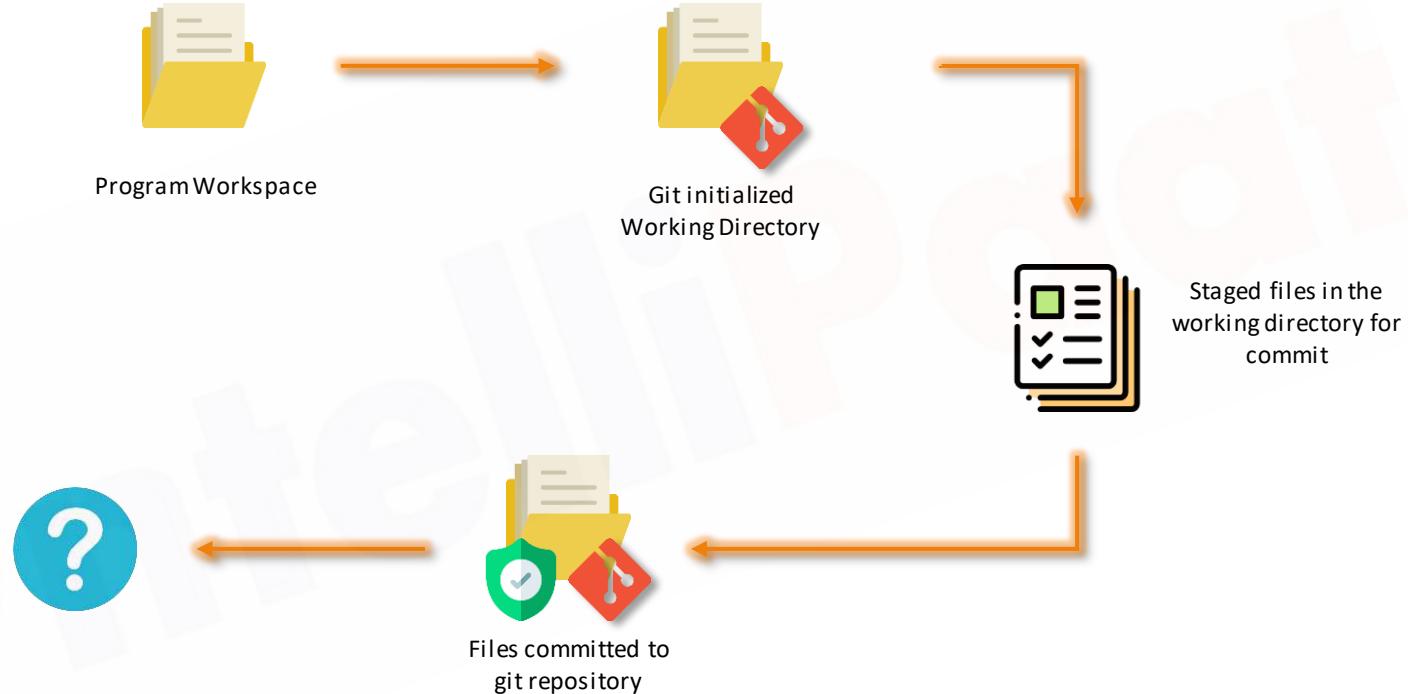
Commit

- ★ Once the files are selected and are ready in the staging area, they can now be saved in repository
- ★ Saving a file in the repository of git is known as doing a commit
- ★ When we commit a repository in git, the commit is identified by a commit id
- ★ The command for initializing this process is *git commit -m "message"*

Git Lifecycle

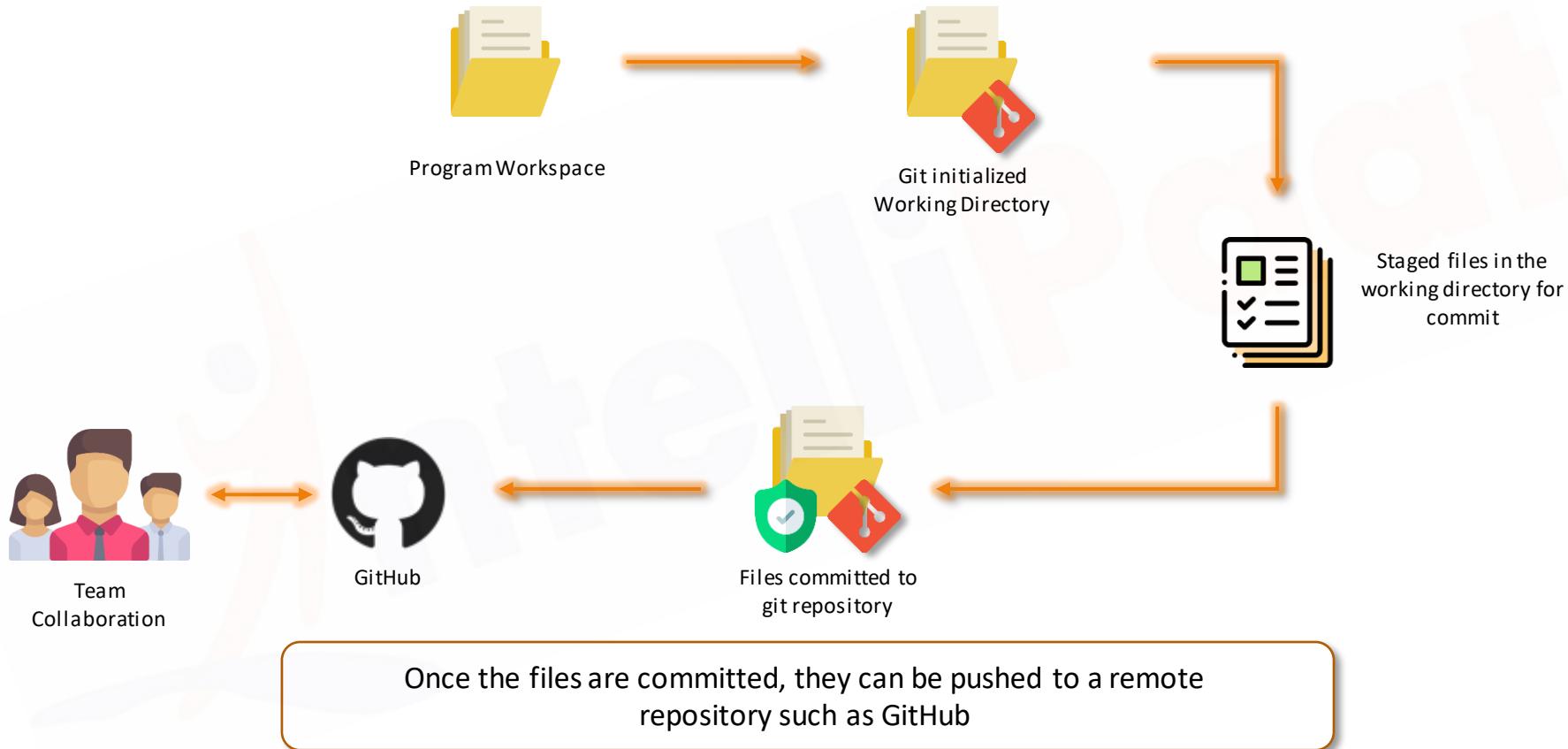


Git Lifecycle



How do we collaborate with the team?

Git Lifecycle



How does Git work?

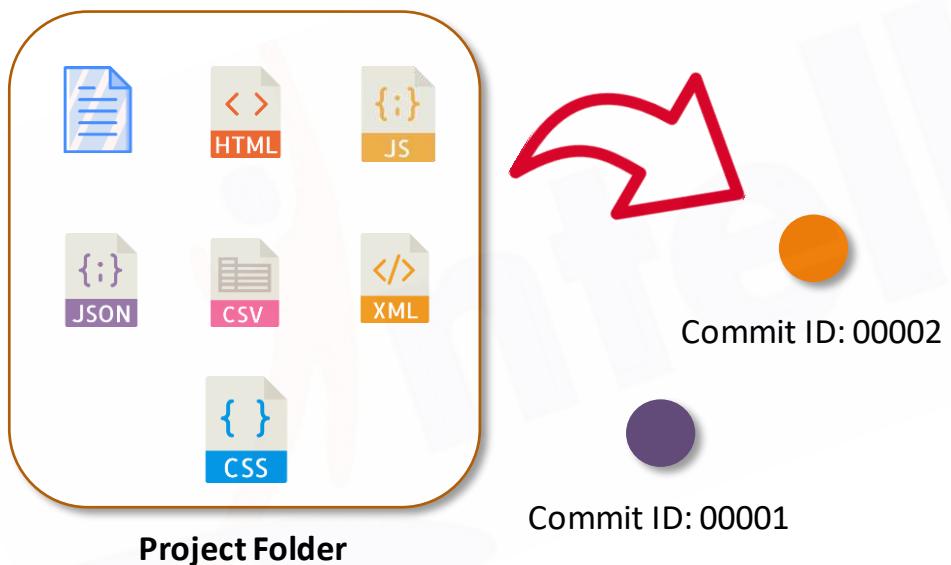
Any project which is saved on git, is saved using a commit. The commit is identified using a commit ID.



Project Folder

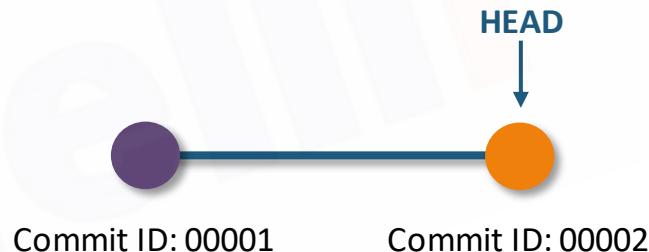
How does Git work?

When we edit the project or add any new functionality, the new code is again committed to git, a new commit ID is assigned to this modified project. The older code is stored by git, and will be accessible by it's assigned Commit ID



How does Git work?

All these commits are bound to a **branch**. Any new commits made will be added to this branch. A branch always points to the latest commit. The pointer to the latest commit is known as **HEAD**



Project Folder

How does Git work?

The default branch in a git repository is called the Master Branch



Project Folder



How does Git work?

The default branch in a git repository is called the Master Branch



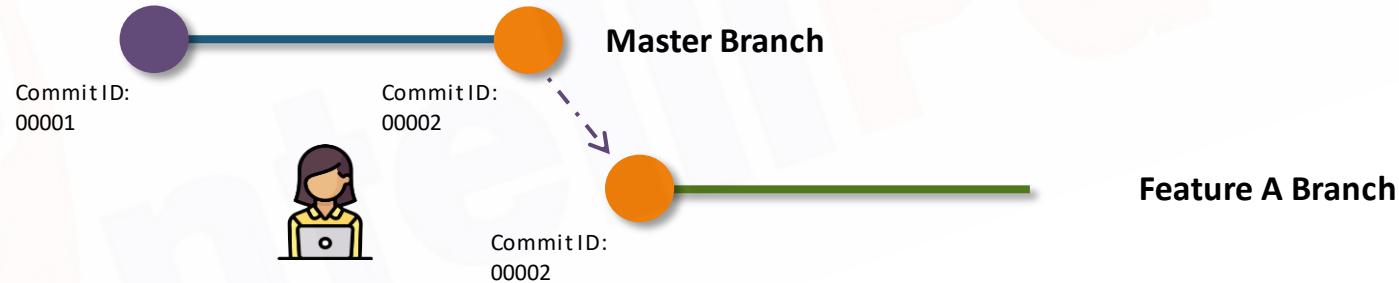
Project Folder



But, why do we need a branch?

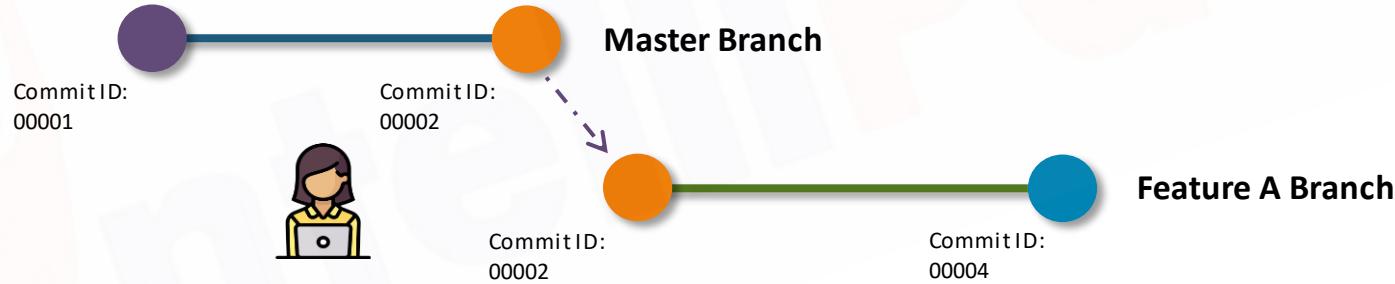
How does Git work?

Say, a developer has been assigned to enhance this code by adding Feature A. The code is assigned to this developer in a separate branch “Feature A”. This is done, so that master contains only the code which is finished, finalized and is on production



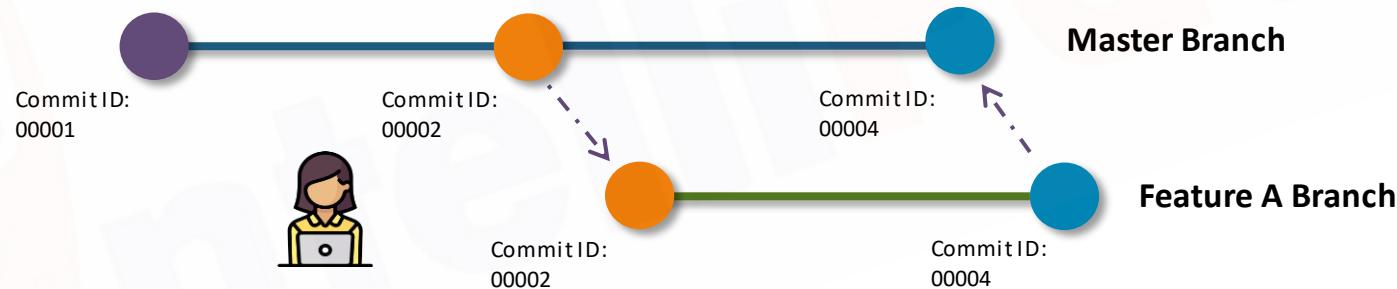
How does Git work?

Therefore, no matter how many commits are made by this developer on Feature A branch, it will not affect the Master Branch.



How does Git work?

Once the code is finished, tested and ready we can merge the Feature A branch, with the master branch and now the code is available on the production servers as well



Common Git Commands

Common Git Commands

You can do the following tasks, when working with git. Let us explore the commands related to each of these tasks



Creating Repository



Making Changes



Parallel Development



Syncing Repositories

Common Git Commands – git init



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

You can create a repository using the command `git init`. Navigate to your project folder and enter the command `git init` to initialize a git repository for your project on the local system

```
[ubuntu@ip-172-31-33-5:~/project$ ls  
1.txt 2.txt  
[ubuntu@ip-172-31-33-5:~/project$ git init  
Initialized empty Git repository in /home/ubuntu/project/.git/  
ubuntu@ip-172-31-33-5:~/project$ ]
```

Common Git Commands – git status



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Once the directory has been initialized you can check the status of the files, whether they are being tracked by git or not, using the command

git status

```
[ubuntu@ip-172-31-33-5:~/project$ ls  
1.txt 2.txt  
[ubuntu@ip-172-31-33-5:~/project$ git status  
On branch master  
  
No commits yet  
  
Untracked files:  
(use "git add <file>..." to include in what will be committed)  
  
    1.txt  
    2.txt  
  
nothing added to commit but untracked files present (use "git add" to track)  
ubuntu@ip-172-31-33-5:~/project$ ]
```

Common Git Commands – git add



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Since no files are being tracked right now, let us now stage these files. For that, enter the command **git add**. If we want to track all the files in the project folder, we can type the command,
git add .

```
[ubuntu@ip-172-31-33-5:~/project$ ls  
1.txt 2.txt  
[ubuntu@ip-172-31-33-5:~/project$ git add .  
[ubuntu@ip-172-31-33-5:~/project$ git status  
On branch master  
  
No commits yet  
  
Changes to be committed:  
(use "git rm --cached <file>..." to unstage)  
  
      new file:   1.txt  
      new file:   2.txt  
  
ubuntu@ip-172-31-33-5:~/project$ ]
```

Common Git Commands – git commit



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Once the files or changes have been staged, we are ready to commit them in our repository. We can commit the files using the command

git commit -m "custom message"

```
ubuntu@ip-172-31-33-5:~/project$ ls
1.txt 2.txt
ubuntu@ip-172-31-33-5:~/project$ git commit -m "First Commit"
2 files changed, 2 insertions(+)
create mode 100644 1.txt
create mode 100644 2.txt
```

Common Git Commands – git remote



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Once everything is ready on our local, we can start pushing our changes to the remote repository. Copy your repository link and paste it in the command

git remote add origin "<URL to repository>"

```
[ubuntu@ip-172-31-33-5:~/project$ git remote add origin "https://github.com/devops-intellipaat/devops.git"
ubuntu@ip-172-31-33-5:~/project$ ]
```

Common Git Commands – git push



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

To push the changes to your repository, enter the command `git push origin <branch-name>` and hit enter. In our case the branch is master, hence
git push origin master

This command will then prompt for username and password, enter the values and hit enter.

```
ubuntu@ip-172-31-33-5:~/project$ git push origin master
Username for 'https://github.com': devops-intellipaat
Password for 'https://devops-intellipaat@github.com':
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 292 bytes | 292.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:     https://github.com/devops-intellipaat/devops/pull/new/master
remote:
To https://github.com/devops-intellipaat/devops.git
 * [new branch]      master -> master
ubuntu@ip-172-31-33-5:~/project$
```

Common Git Commands – git push



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Your local repository is now synced with the remote repository on
github

1 commit 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

Ubuntu First Commit Latest commit 6f13532 33 minutes ago

1.txt First Commit 33 minutes ago

2.txt First Commit 33 minutes ago

Add a README

Help people interested in this repository understand your project by adding a README.

A screenshot of a GitHub repository page. At the top, it shows 1 commit, 1 branch, 0 releases, and 0 contributors. Below that, it says "Branch: master" and has a "New pull request" button. There are buttons for "Create new file", "Upload files", "Find file", and "Clone or download". The main area shows a commit from "Ubuntu" titled "First Commit" made 33 minutes ago. It lists two files: "1.txt" and "2.txt", both of which are "First Commit" and were made 33 minutes ago. At the bottom, there's a note to "Add a README" and a placeholder text "Help people interested in this repository understand your project by adding a README.".

Common Git Commands – git clone



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Similarly, if we want to download the remote repository to our local system, we can use the command:

git clone <URL>

This command will create a folder with the repository name, and download all the contents of the repository inside this folder. In our example, repository contents were downloaded into the "devops" folder.

```
[ubuntu@ip-172-31-33-5:~$ git clone https://github.com/devops-intellipaat/devops.git
Cloning into 'devops'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), done.
[ubuntu@ip-172-31-33-5:~$ ls
devops  project
ubuntu@ip-172-31-33-5:~$ ]
```

Common Git Commands – git pull



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

The git pull command is also used for pulling the latest changes from the repository, unlike git clone, this command can only work inside an initialized git repository. This command is used when you are already working in the cloned repository, and want to pull the latest changes, that others might have pushed to the remote repository

git pull <URL of link>

```
[ubuntu@ip-172-31-33-5:~/devops$ git pull https://github.com/devops-intellipaat/d
evops.git
From https://github.com/devops-intellipaat/devops
 * branch            HEAD      -> FETCH_HEAD
Already up to date.
ubuntu@ip-172-31-33-5:~/devops$ ]
```

Common Git Commands – git branch



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Until now, we saw how you can work on git. But now imagine, multiple developers working on the same project or repository. To handle the workspace of multiple developers, we use branches. To create a branch from an existing branch, we type

git branch <name-of-new-branch>

Similarly, to delete a branch use the command

git branch -D <branch name>

```
[ubuntu@ip-172-31-33-5:~$ cd devops  
[ubuntu@ip-172-31-33-5:~/devops$ git branch branch1  
ubuntu@ip-172-31-33-5:~/devops$ ]
```

Common Git Commands – git checkout



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

To switch to the new branch, we type the command

git checkout <branch-name>

```
[ubuntu@ip-172-31-33-5:~/devops$ git checkout branch1
Switched to branch 'branch1'
[ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt 2.txt
ubuntu@ip-172-31-33-5:~/devops$ ]
```

Common Git Commands – git log



Want to check the log for every commit detail in your repository?
You can accomplish that using the command

git log

```
[ubuntu@ip-172-31-33-5:~/devops$ git log
commit dd6974eda23d7644d9cb724a82ebd829c7717ac6 (HEAD -> branch1, master)
Author: Ubuntu <ubuntu@ip-172-31-33-5.us-east-2.compute.internal>
Date:   Fri Nov 23 06:21:41 2018 +0000

    adding test file

commit 6f135327baf101788b23e3053a75d828709f6bb7 (origin/master, origin/HEAD)
Author: Ubuntu <ubuntu@ip-172-31-33-5.us-east-2.compute.internal>
Date:   Fri Nov 23 05:00:03 2018 +0000

    First Commit
ubuntu@ip-172-31-33-5:~/devops$ ]
```

Common Git Commands – git stash



Creating Repository



Making Changes



Syncing Repositories



Parallel Development

Want to save your work without committing the code? Git has got you covered. This can be helpful when you want to switch branches, but do not want to save your work to your git repository. To stash your staged files without committing just type in **git stash**. If you want to stash your untracked files as well, type **git stash -u**.

Once you are back and want to retrieve working, type in **git stash pop**

```
ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt 2.txt 3.txt 4.txt
ubuntu@ip-172-31-33-5:~/devops$ git stash -u
Saved working directory and index state WIP on master: dd6974e adding test file
ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt 2.txt 3.txt
ubuntu@ip-172-31-33-5:~/devops$ git stash pop
Already up to date!
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    4.txt

nothing added to commit but untracked files present (use "git add" to track)
Dropped refs/stash@{0} {7f106523effac55075b2d03387245c487a3de84f}
ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt 2.txt 3.txt 4.txt
ubuntu@ip-172-31-33-5:~/devops$
```

Common Git Commands – git revert

This command helps you in reverting a commit, to a previous version

```
git revert <commit-id>
```

<commit-id> can be obtained from the output of `git log`

```
ubuntu@ip-172-31-33-5:~/devops$ git revert dd6974eda23d7644d9cb724a82ebd829c7717  
ac6  
[branch1 88c0d66] Revert "adding test file"  
Committer: Ubuntu <ubuntu@ip-172-31-33-5.us-east-2.compute.internal>  
Your name and email address were configured automatically based  
on your username and hostname. Please check that they are accurate.  
You can suppress this message by setting them explicitly. Run the  
following command and follow the instructions in your editor to edit  
your configuration file:  
  
git config --global --edit  
  
After doing this, you may fix the identity used for this commit with:  
  
git commit --amend --reset-author  
  
1 file changed, 1 deletion(-)  
delete mode 100644 3.txt
```

Common Git Commands – git diff

This command helps us in checking the differences between two versions of a file

git diff <commit-id of version x> <commit-id of version y>

<commit-id> can be obtained from the output of **git log**

```
ubuntu@ip-172-31-23-227:~/devopsIQ/devopsIQ$ git diff 4bdbc8b0d037553729e2e75e75  
48bc84dcf19564 55d4c573efcd1f1ab70c2f926cb41f4c61d29d20  
diff --git a/devopsIQ/index.html b/devopsIQ/index.html  
index 87f0103..e4404e7 100644  
--- a/devopsIQ/index.html  
+++ b/devopsIQ/index.html  
@@ -1,5 +1,5 @@  
<html>  
-<title>Jenkins Final Website2</title>  
+<title>Jenkins Final Website</title>^M  
<body background="images/1.jpg">  
</body>  
</html>
```

Merging Branches

Merging Branches

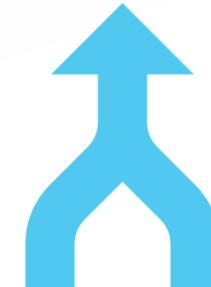
Once the developer has finished his code/feature on his branch, the code will have to be combined with the master branch. This can be done using two ways:



Git Merge



Git Rebase



Merging Branches – git merge



Git Merge



Git Rebase

- ★ If you want to apply changes from one branch to another branch, one can use merge command
- ★ Should be used on remote branches, since history does not change
- ★ Creates a new commit, which is a merger of the two branches
- ★ Syntax: `git merge <source-branch>`

Merging Branches – git merge

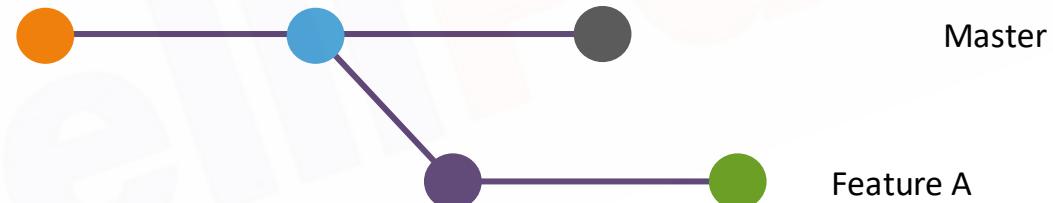


Git Merge



Git Rebase

Imagine, you have a Master branch and a Feature A branch.
The developer has finished his/her work in the feature A
branch and wants to merge his work in the master.



Merging Branches – git merge



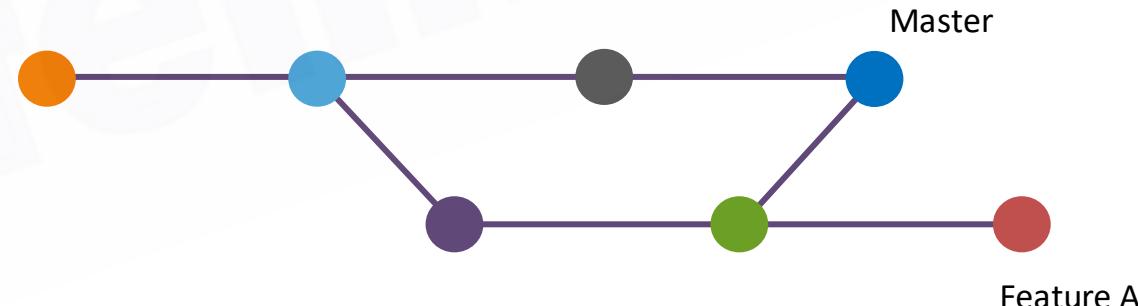
Git Merge



Git Rebase

If he is using **git merge**, a new commit will be created, which will have the changes of Feature A and Master branch combined.

Any new commits to the Feature branch will be isolated from the master branch



Merging Branches – git merge



Git Merge



Git Rebase

This command can be executed using the syntax

git merge <source-branch-name>

```
[ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt 2.txt 3.txt
[ubuntu@ip-172-31-33-5:~/devops$ git status
On branch branch1
nothing to commit, working tree clean
[ubuntu@ip-172-31-33-5:~/devops$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
[ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt 2.txt
[ubuntu@ip-172-31-33-5:~/devops$ git merge branch1
Updating 6f13532..dd6974e
Fast-forward
 3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 3.txt
[ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt 2.txt 3.txt
[ubuntu@ip-172-31-33-5:~/devops$ ]
```

Merging Branches – git merge



Git Merge



Git Rebase

The history of the branch will look something like this, if we are using
git merge

```
[ubuntu@ip-172-31-26-120:~/n$ git log --graph --pretty=oneline
*   d92f22eeb6bb7fef1706b397abe804dc557ec88 (HEAD -> master) Merge branch
 |
 |\
 | * aebc77927892bd1c74ffd9b3d9af7f3b763ee8da (test) 1st on test
 * | b62c11b6a12e4c0431bf4ae7f9fe90f744d485b7 second on master
 |
 * 071f9bd946e502d4643d2fc7e2dd7c26dea0eaf9 first commit in master
```

Merging Branches – git merge



Git Merge



Git Rebase

- ★ This is an alternative to git merge command
- ★ Should be used on local branches, since history does change and will be confusing for other team members
- ★ Does not create any new commit, and results in a cleaner history
- ★ The history is based on common commit of the two branches (base)
- ★ The destination's branch commit is pulled from its “base” and “rebased” on to the latest commit on the source branch

Merging Branches – git rebase

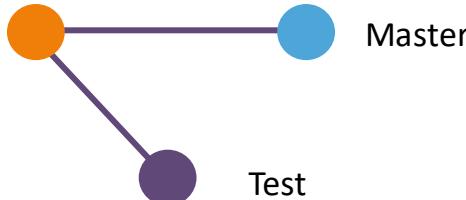


Git Merge



Git Rebase

- ★ Imagine, you have a Master branch and a test branch(local branch)
- ★ The developer has finished his/her work in the test branch
- ★ But the master moved forward, while the code was being developed
- ★ Code being developed is related to the new commit added in master



Merging Branches – git rebase



Git Merge



Git Rebase

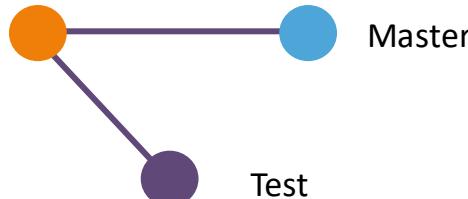


Therefore you want all the changes from master in feature.



Since, it is a local branch, you would want a cleaner or linear history, you decide to use git rebase

Syntax: **git rebase <source branch>**



Merging Branches – git rebase



Git Merge



Git Rebase



This is how the output looks like:

```
ubuntu@ip-172-31-26-120:~/n$ git checkout test
Switched to branch 'test'
ubuntu@ip-172-31-26-120:~/n$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: 1st in test
```

Merging Branches – git rebase



Git Merge



Git Rebase



This is how the commits look like, after a rebase. The commit was “rebased” from the first commit to the next commit



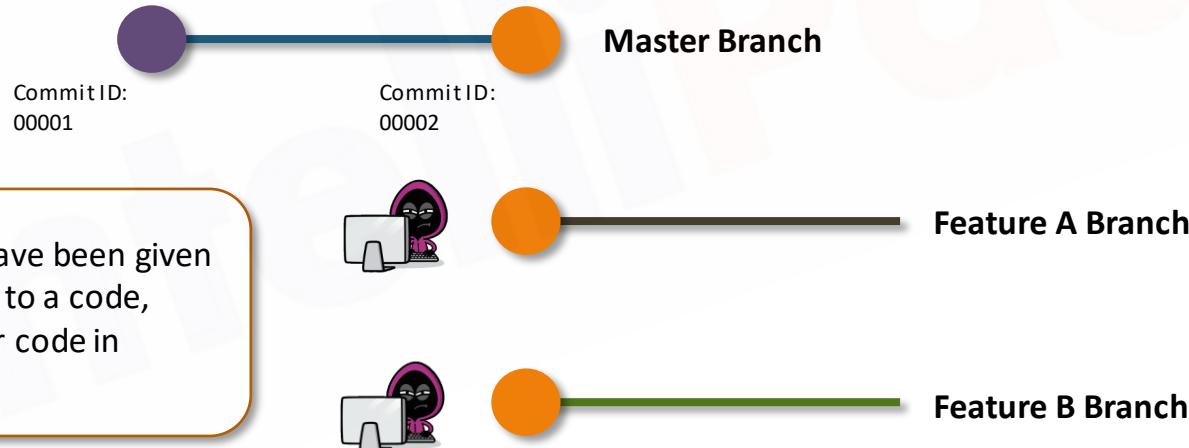
And looking at the history we can clearly see, it's a clean linear history, without any branches

```
[ubuntu@ip-172-31-26-120:~/n$ git log --graph --pretty=oneline
* 3885b20a7f8880acf4b7a785a638e95d1759dcf2 (HEAD -> test) 1st in test
* cce38fa142699171d08b08b27ed44f49052ac134 (master) 2nd in master
* 7d77f726ad1d0b64f6f20c2587560dc18123082d 1st in master
```

Merge Conflicts

Merge Conflicts

Merge conflicts occur when we try to merge two branches, which have the same file updated by two different developers. Let's understand it using a scenario:



Merge Conflicts

The functions.c file looks something like this as of now,

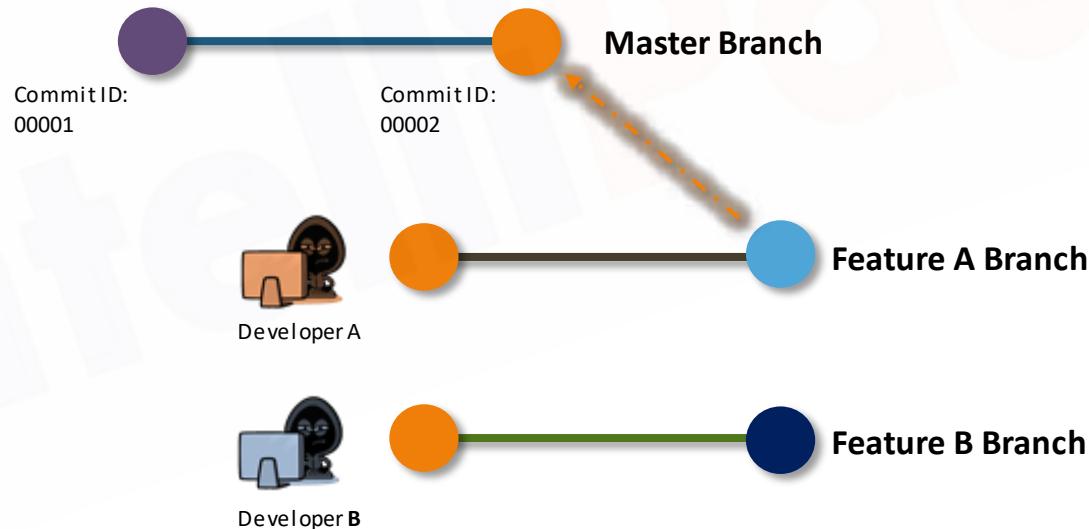
```
Main()
{
    Function1()
    {
        //InitialCode
    }

}
```

function.c

Merge Conflicts

Developer A finished his code, and pushes the changes to the master branch



Merge Conflicts

```
Main()
{
Function1()
{
    //Initial Code
}
Function2()
{
    //Developer A Code
}

}
```

function.c

After the **Developer A** changes his code and pushes it to master, the code on the **Master** branch looks something like this

```
Main()
{
Function1()
{
    //Initial Code
}
Function3()
{
    //Developer B Code
}

}
```

function.c

After the **Developer B** changes his code, the code on **Feature B** branch looks something like this

Merge Conflicts

Comparing the two code, we can see Feature A Branch is missing Developer A code. Therefore if we merge Feature A Branch with Master Branch, logically Developer A changes will disappear

Master Branch

```
Main()
{
Function1()
{
    //Initial Code
}
Function2()
{
    //Developer A Code
}

}
```

function.c

Feature A Branch

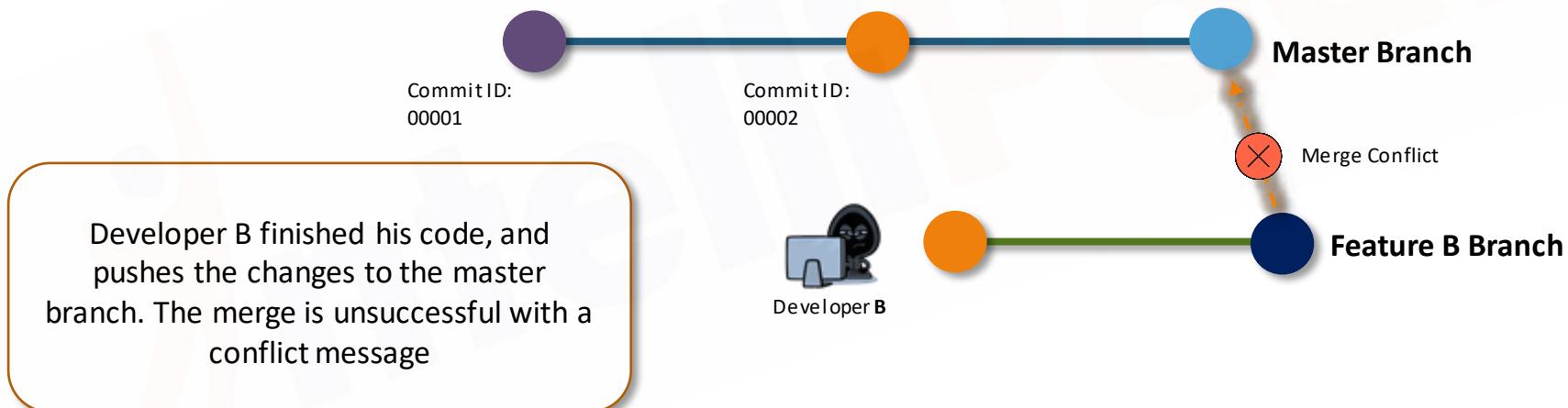
```
Main()
{
Function1()
{
    //Initial Code
}
Function3()
{
    //Developer B Code
}

}
```

function.c

Merge Conflicts

To solve this, git has a fail safe. If the Master branch has been moved forward in commits, compared to the branch which is being merged, it creates a conflict.



Hands-on – Simulating a Merge Conflict

Merge Conflicts

This is the message, you will get when you merge a branch, which has a conflicting file

```
[ubuntu@ip-172-31-26-120:~/dev1/devops$ git merge dev2
Auto-merging feature.c
CONFLICT (content): Merge conflict in feature.c
Automatic merge failed; fix conflicts and then commit the result.
```

Merge Conflict Message

How to resolve Merge Conflicts?

How to resolve Merge Conflicts?

Once we have identified, there is a merge conflict we should go ahead and use the command

git mergetool

```
ubuntu@ip-172-31-26-120:~/dev1/devops$ git mergetool

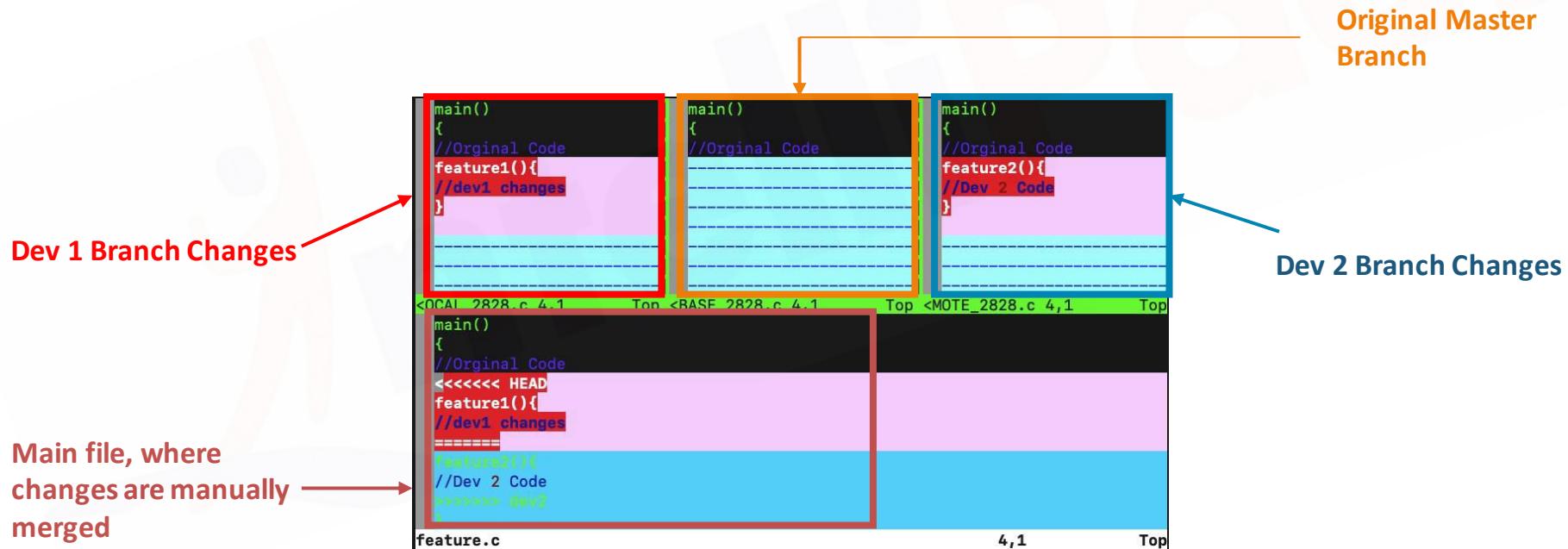
This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
tortoisemerge emerge vimdiff
Merging:
feature.c

Normal merge conflict for 'feature.c':
{local}: modified file
{remote}: modified file
Hit return to start merge resolution tool (vimdiff): █
```

Hit enter after this prompt, and then you should enter the merge tool

How to resolve Merge Conflicts?

The merge tool looks something like this, the top leftmost column is for Dev1 Branch changes, the centre column is for the original code i.e the master's code before any commits, and the right most column are the dev 2 branch changes. The area below these columns is where we make the changes, this is the place where we have the merged code.



Original Master Branch

Dev 1 Branch Changes

Dev 2 Branch Changes

Main file, where changes are manually merged

```

main()
{
    //Original Code
    feature1(){
        //dev1 changes
    }
}

<LOCAL_2828.c 4,1> Top <BASE_2828.c 4,1> Top <MOTE_2828.c 4,1> Top
main()
{
    //Original Code
    <<<<< HEAD
    feature1(){
        //dev1 changes
    }
    =====
    feature2(){
        //Dev 2 Code
    }
}

feature.c 4,1 Top

```

How to resolve Merge Conflicts?

Once you have resolved the changes, save the file using “:wq”, vim command for save and exit. Do the same for all the files



```
main()
{
//Orginal Code
feature1(){
//dev1 changes
}

<LOCAL_3163.c 3,14      Top <BASE_3163.c 3,14      Top <MOTE_3163.c 3,14      Top
main()
{
//Orginal Code
feature1(){
//dev1 changes
}
feature2(){
//Dev 2 Code
}

}

feature.c [+]           3,14          All
:wq
```

How to resolve Merge Conflicts?

After this step, see the status of your local repository you can see the file in conflict has been modified successfully and is merged with master. This modified file can now be committed to the master

```
ubuntu@ip-172-31-26-120:~/dev1/devops$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

  modified:   feature.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    feature.c.orig
    feature_BACKUP_2828.c
    feature_BASE_2828.c
    feature_LOCAL_2828.c
    feature_REMOTE_2828.c
```

There will be some other files which have been created, these files are a copy of the original files which have been changed, you can delete them, if not needed.

How to resolve Merge Conflicts?

Finally commit your changes, to the branch and then push it to the remote repository

```
[ubuntu@ip-172-31-26-120:~/dev1/devops$ git commit -m "merged feature"
[master f0ecdbd] merged feature
Committer: Ubuntu <ubuntu@ip-172-31-26-120.us-east-2.compute.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 feature.c
```

Collaboration in GitHub

Collaboration in GitHub

Collaboration in GitHub is very important aspect of Software Development. It enables developers to parallelly work on the same project



Process of Collaboration in GitHub

1

Add collaborators to your repository

2

Protect the branches, which need restricted access



Process of Collaboration in GitHub

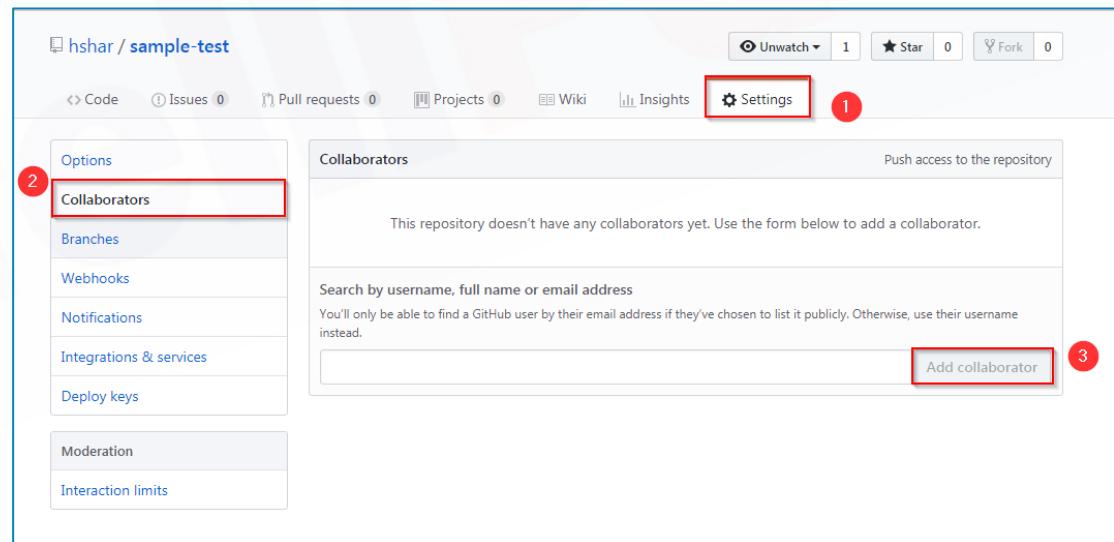


Collaborators



Protecting Branches

Collaborators are the people who will be working on your project. To add contributors to your repository, use the steps in the following image:



The screenshot shows the GitHub repository settings for 'hshar / sample-test'. The 'Settings' tab is selected, indicated by a red box and the number 1. On the left, a sidebar menu lists 'Options' (highlighted with a red box and the number 2), 'Collaborators', 'Branches', 'Webhooks', 'Notifications', 'Integrations & services', 'Deploy keys', 'Moderation', and 'Interaction limits'. The 'Collaborators' section displays a message: 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' It includes a search bar with placeholder text 'Search by username, full name or email address' and a note: 'You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.' A red box highlights the 'Add collaborator' button at the bottom right of the form, with the number 3 above it.

Process of Collaboration in GitHub



Collaborators



Protecting Branches

Using this feature, you can restrict access on how commits are made to the branches you specify



Hands-on: Collaborating in GitHub

Hands-on

1. Add a Collaborator in your GitHub repository, by creating an alternate account
2. Protect the Master branch, from getting changes directly pushed on it
3. Push changes to the repository, in a feature branch
4. Create a Pull Request from the feature branch to the Master Branch
5. From the owner's account approve the pull request



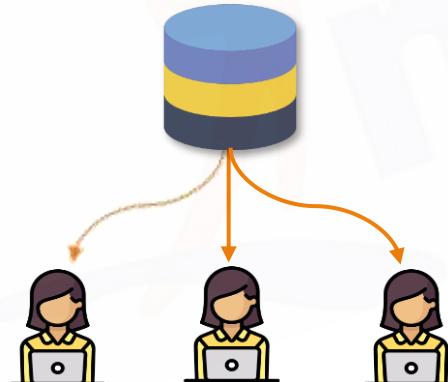
Git Workflow

Git Workflow

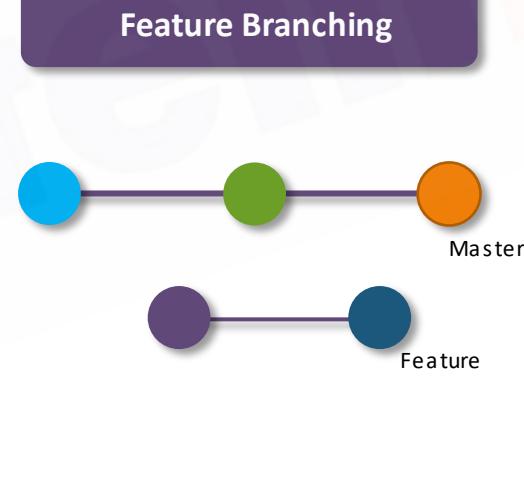
A Git Workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner. Git workflows encourage users to leverage Git effectively and consistently.

There are three popular workflows which are accepted and are followed by various tech companies:

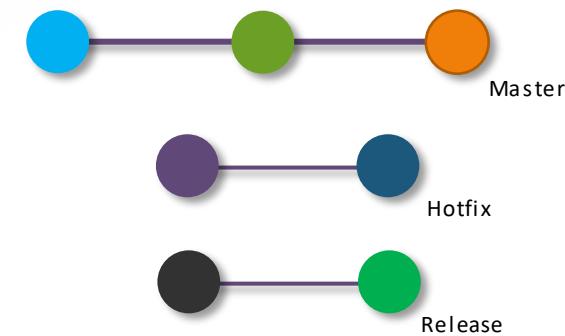
Centralized Workflow



Feature Branching



GitFlow Workflow



Git Workflow



Centralized Workflow

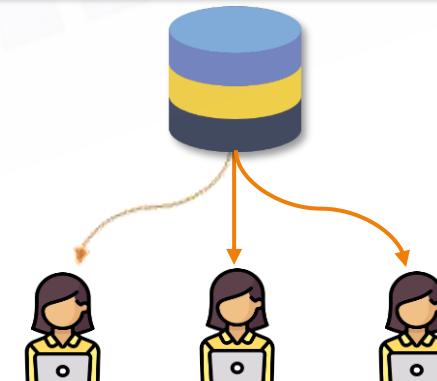


Feature Branching



GitFlow Workflow

- ★ This workflow doesn't require any other branch other than master
- ★ All the changes are directly made on the master, and finally merged on the remote master, once work is finished
- ★ Before pushing changes, the master is rebased with the remote commits
- ★ Results in a clean, linear history



Git Workflow



Centralized Workflow

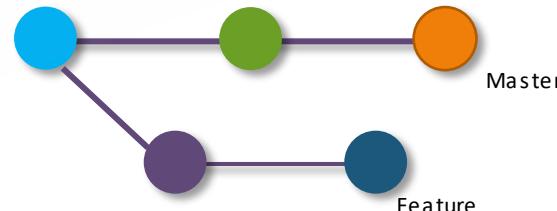


Feature Branching



GitFlow Workflow

- ★ Master only contains the production ready code
- ★ Any Development work, is converted into a feature branch
- ★ There can be numerous feature branches, depending on the application's development plan
- ★ Once the feature is complete, the feature branch is merged with the master



Git Workflow



Centralized Workflow

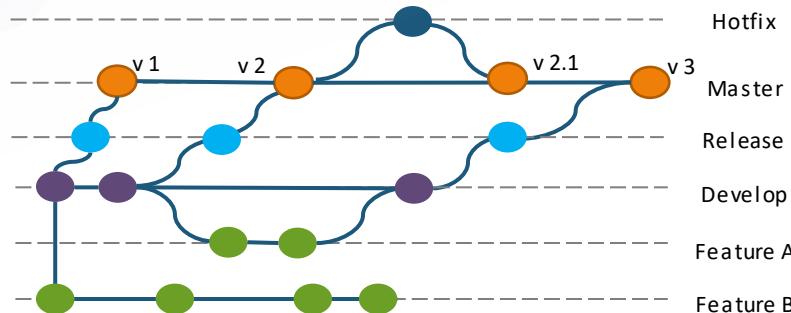


Feature Branching



GitFlow Workflow

- ★ The Feature branches are never merged directly with master
- ★ Once the features are ready, commit is merged with Develop
- ★ When there are enough commits on Develop, we merge the Develop branch with Release branch, only read me files or License files are added after this commit on Release
- ★ Any quick fixes which are required, are done on the Hotfix branch, this branch can directly be merged with Master



Forking in GitHub

What is Forking?

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either suggest changes to someone else's project or to use someone else's project as a starting point for your own idea.



Forking in GitHub

- ★ Forking is a GitHub concept, it has nothing to do with Git software
- ★ It is used to copy someone else's repository to your own repo in GitHub
- ★ The changes made to a forked repository are not reflected in the parent repository
- ★ If one wants to suggest any change to the parent repository from the forked repository



Quiz

1. Which of the following is a characteristics of a Centralized Version Control System?

- A. Local Operations are fast
- B. Version History is available on the local storage as well
- C. Version History is not available on the local storage as well
- D. None of these

1. Which of the following is a characteristics of a Centralized Version Control System?

- A. Local Operations are fast
- B. Version History is available on the local storage as well
- C. Version History is not available on the local storage as well**
- D. None of these

2. Which of the following should git rebase be used on?

- A. Master Branch
- B. Remote Branch
- C. Local Branch
- D. None of these

2. Which of the following should git rebase be used on?

- A. Master Branch
- B. Remote Branch
- C. Local Branch
- D. None of these

3. Which of the following workflow should be used, if we want to deploy multiple features at once?

- A. Centralized Workflow
- B. Feature Workflow
- C. GitFlow Workflow
- D. None of these

3. Which of the following workflow should be used, if we want to deploy multiple features at once?

A. Centralized Workflow

B. Feature Workflow

C. GitFlow Workflow

D. None of these

4. Forking helps us in _____

- A. Cloning the Repository
- B. Copying a Repository to your GitHub account
- C. Suggesting Changes to the present repository
- D. None of these

4. Forking helps us in _____

- A. Cloning the Repository
- B. Copying a Repository to your GitHub account**
- C. Suggesting Changes to the present repository
- D. None of these

5. Merge Conflicts occur when,

- A. Two developers simultaneously commit to a repository
- B. Two or more developers try to merge on local system
- C. Two or more developers try to merge to the remote repository
- D. Two developers who worked on the same file, try to merge on the branch

5. Merge Conflicts occur when,

- A. Two developers simultaneously commit to a repository
- B. Two or more developers try to merge on local system
- C. Two or more developers try to merge to the remote repository
- D. Two developers who worked on the same file, try to merge on the branch



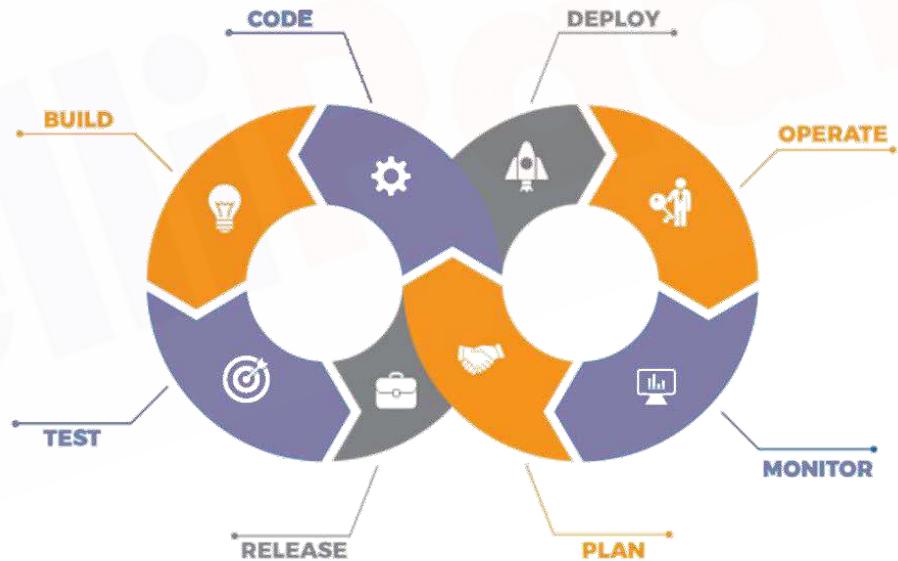
India : +91-7847955955

US : 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24X7 Chat with our Course Advisor

Containerization Using Docker - I



Agenda

01

What is
Virtualization?

02

What is
Containerization?

03

Containerization
Tools

04

Components of
Docker

05

Installing Docker

06

Common Docker
Commands

07

Creating a Docker Hub
Account

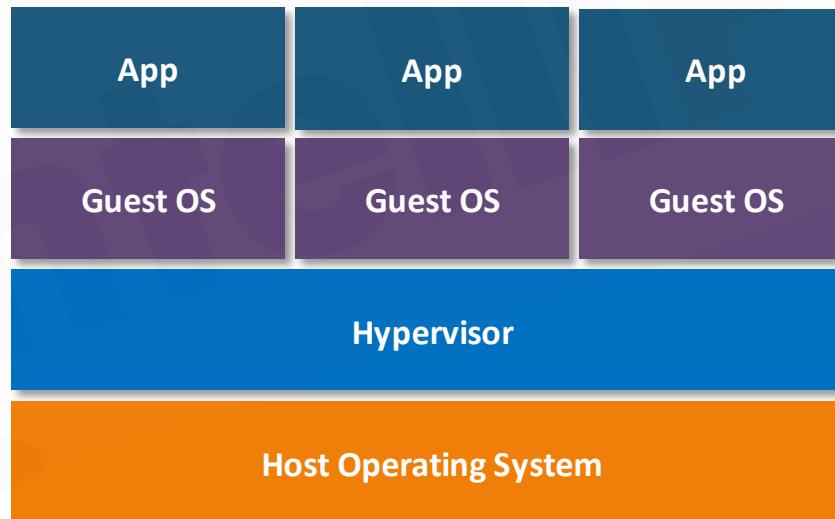
08

Introduction to
Dockerfile

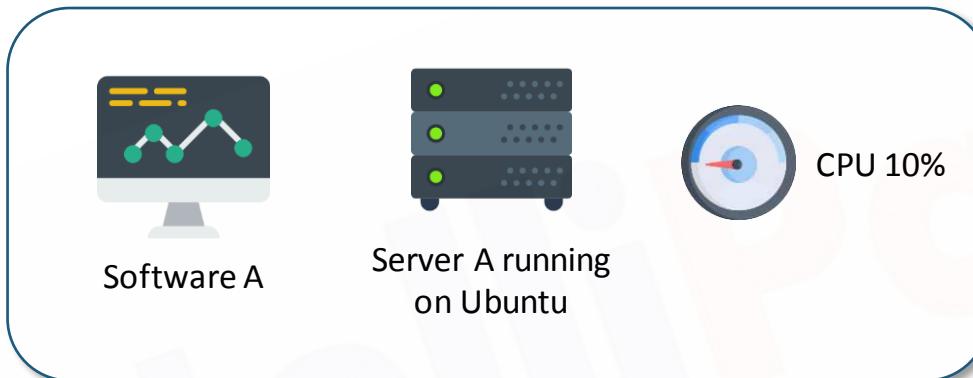
What is Virtualization?

What is Virtualization?

Virtualization is the process of running multiple virtual systems or resources on top of a single physical machine. These resources could be a storage device, network or even an operating system!



Problems before Virtualization



Imagine Software A running on Server A which has Ubuntu running on it. This software can only run in the Ubuntu environment.

Problems before Virtualization



Software A



Server A running
on Ubuntu



CPU 10%



Software B



Server B running
on Windows



CPU 10%

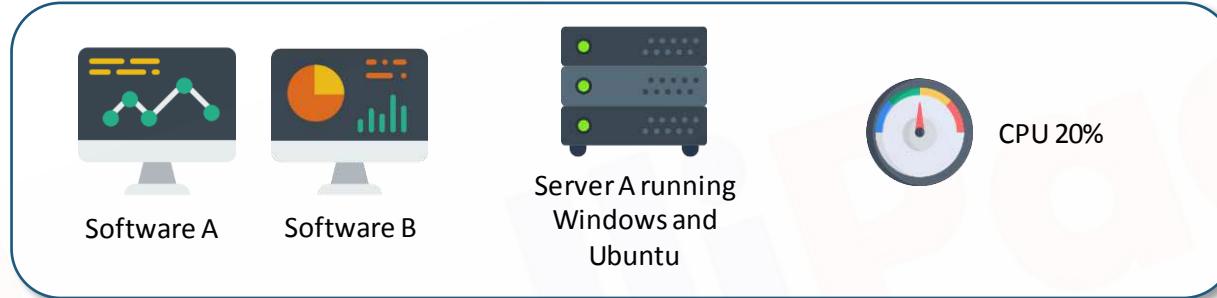
Some time later, we needed Software B which can only run on Windows. Therefore, we had to buy and run a Server B which had windows running on it. The software took only 10% of the CPU resources.

Problems before Virtualization



- ✖ Buying servers was expensive.
- ✖ Resources were not being utilized at their full potential.
- ✖ The process of getting any software up and running was time consuming.
- ✖ Disaster recovery was difficult.

After Virtualization



Windows and Ubuntu OS now are running on the same server in parallel using the Virtualization technology. This accounts for better CPU utilization and cost savings!

Advantages of Virtualization

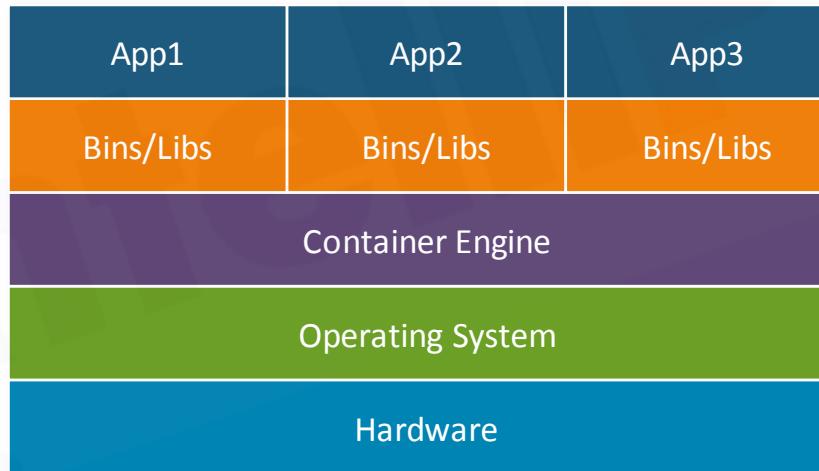


- ✓ It results in reduced spending.
- ✓ Resources are utilized more efficiently.
- ✓ Process of getting software up and running is shorter.
- ✓ Easier backup and disaster recovery is available.

What is Containerization?

What is Containerization?

Application **containerization** is an OS-level virtualization method used to deploy and run distributed applications without launching an entire virtual machine (VM) for each app.



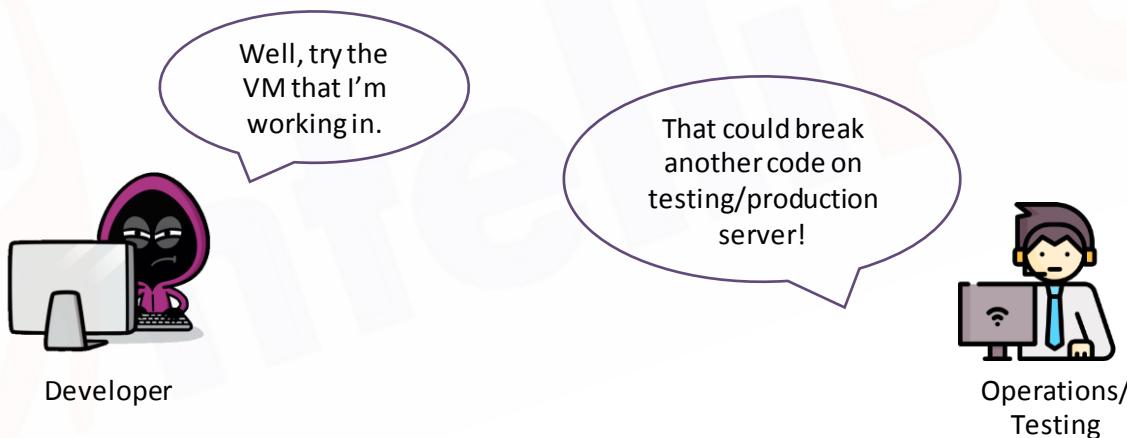
Problems before Containerization

Developers when run the code on their system, it would run perfectly. But the same code would not run on the operations team's system.



Problems before Containerization

The problem was with the environment the code was being run in. Well, a simple answer could be, why not give the same VM to the operations/testing team to run the code.



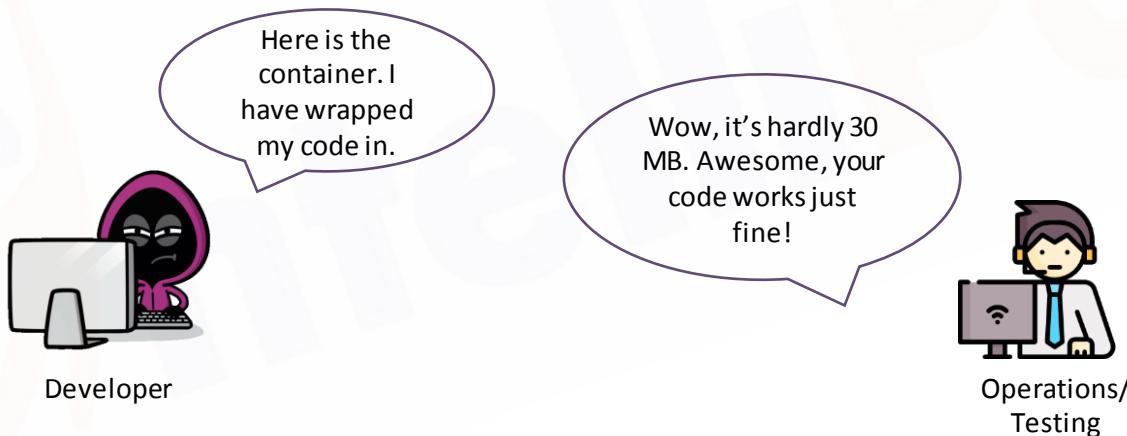
Problems before Containerization



- ✖ VMs took too many resources to run.
- ✖ VMs were too big in size to be portable.
- ✖ VMs were not developer friendly.

How did containers solve the problems?

With containers, all the environment issues were solved. The developer could easily wrap their code in a lightweight container and pass it on to the operations team.



Advantages of Containers



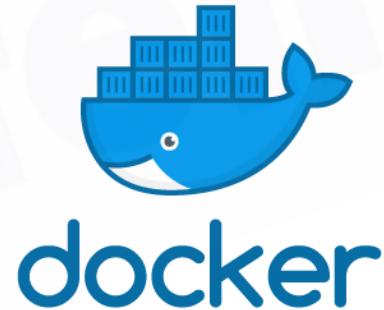
- ✓ Containers are not resource hungry.
- ✓ They are lightweight and hence portable.
- ✓ They are developer friendly and can be configured through the code.

Containerization Tools

Containerization Tools

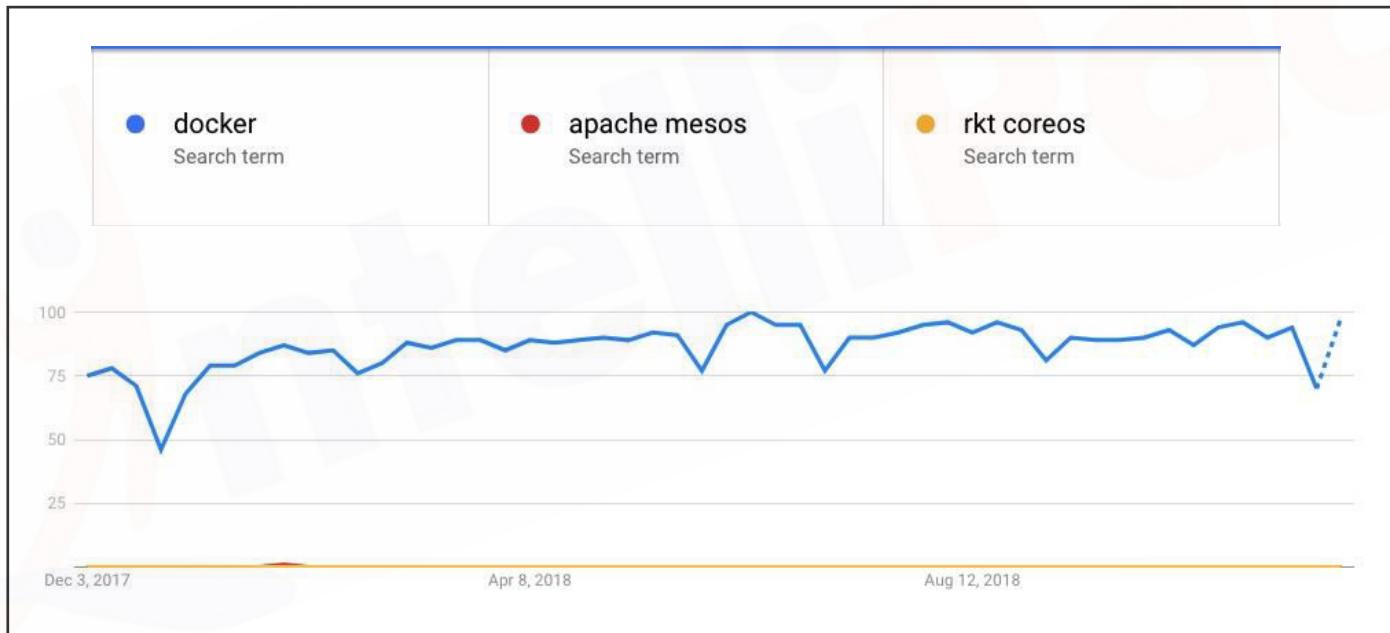


MESOS



Containerization Tools

Docker is clearly the most famous among them all!

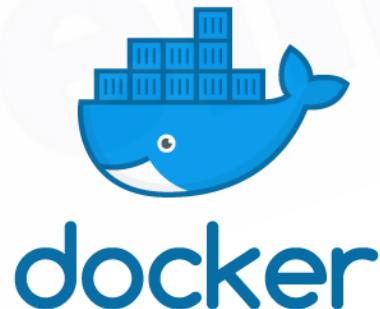


What is Docker?

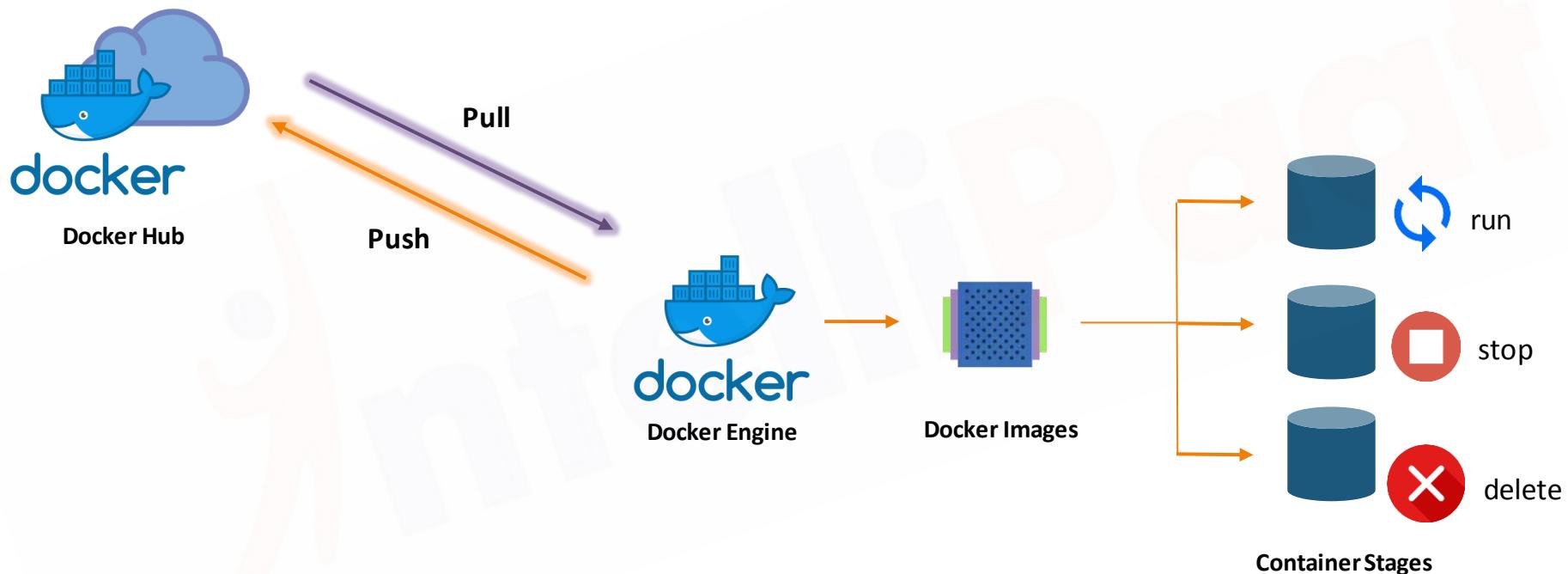
What is Docker?

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization". It was first released in 2013 and is developed by Docker, Inc.

Docker is used to run software packages called "containers".

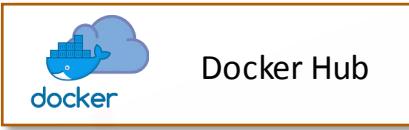


Docker Container Life Cycle



Components of Docker Ecosystem

Components of Docker Ecosystem



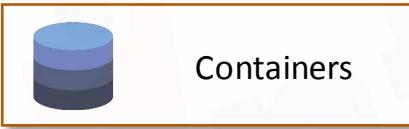
Docker Hub



Docker Engine



Docker Images



Containers

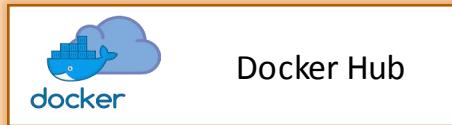


Docker Volumes



Docker File

Components of Docker Ecosystem



Docker Hub



Docker Engine



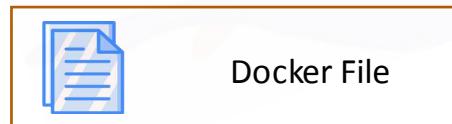
Docker Images



Containers

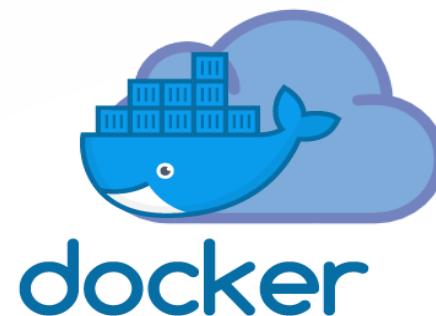


Docker Volumes

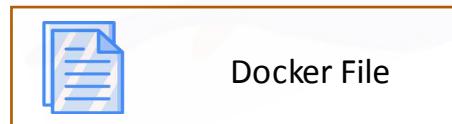
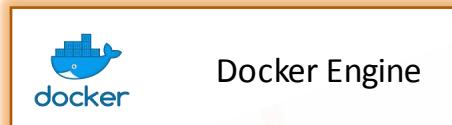
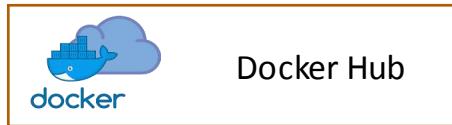


Docker File

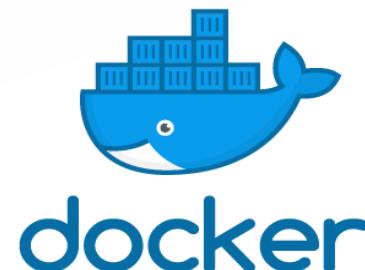
- ★ Docker Hub is a central public docker registry.
- ★ It can store custom docker images.
- ★ The service is free, but your images would be public.
- ★ It requires username/password.



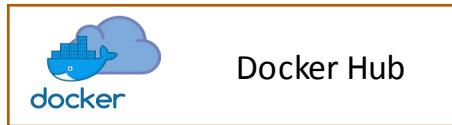
Components of Docker Ecosystem



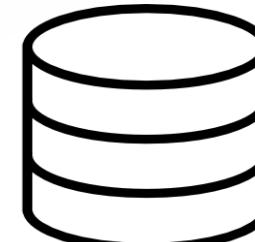
- ★ Docker Engine is the heart of the docker ecosystem.
- ★ It is responsible for managing your container runtimes.
- ★ It works on top of operating system level.
- ★ It utilizes the kernel of the underlying OS.



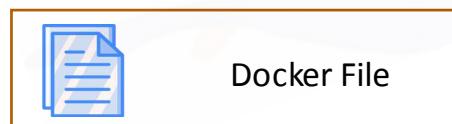
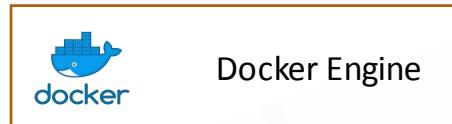
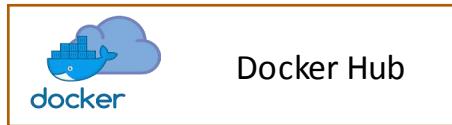
Components of Docker Ecosystem



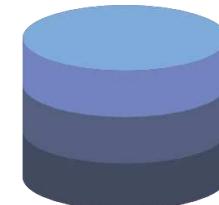
- ★ Docker Image is like the template of a container.
- ★ It is created in layers.
- ★ Any new changes in the image results in creating a new layer.
- ★ One can launch multiple containers from a single docker image.



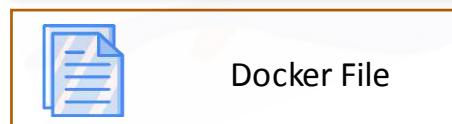
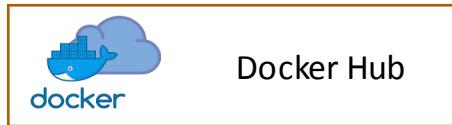
Components of Docker Ecosystem



- ★ A Docker Container is a lightweight software environment.
- ★ It works on top of the underlying OS kernel.
- ★ It is small in size and therefore is highly portable.
- ★ It is created using the docker image.



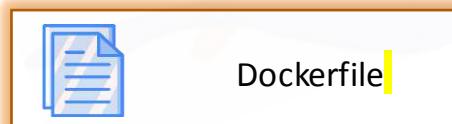
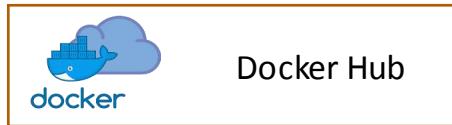
Components of Docker Ecosystem



- ★ Docker Containers cannot persist data.
- ★ To persist data in containers, we can use Docker Volume.
- ★ A Docker Volume can connect to multiple containers simultaneously.
- ★ If not created explicitly, a volume is automatically created when we create a container.



Components of Docker Ecosystem



- ★ Dockerfile is a YAML file, which is used to create custom containers
- ★ It can include commands that have to be run on the command line
- ★ This Dockerfile can be used to build custom container images



Installing Docker

Common Docker Commands

Common Docker Commands

```
docker --version
```

```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker --version
Docker version 18.06.1-ce, build e68fc7a
ubuntu@ip-172-31-26-120:~$
```

This command helps you know the installed version of the docker software on your system.

Common Docker Commands

```
docker pull <image-name>
```

```
ubuntu@ip-172-31-26-120:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
32802c0cfa4d: Pull complete
da1315cffa03: Pull complete
fa83472a3562: Pull complete
f85999a86bef: Pull complete
Digest: sha256:6d0e0c26489e33f5a6f0020edface2727db94897
Status: Downloaded newer image for ubuntu:latest
ubuntu@ip-172-31-26-120:~$ █
```

This command helps you pull images from the central docker repository.

Common Docker Commands

docker images

```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker images
REPOSITORY          TAG      IMAGE ID
SIZE
ubuntu              latest   93fd78260bd1
86.2MB
ubuntu@ip-172-31-26-120:~$
```

This command helps you in listing all the docker images downloaded on your system.

Common Docker Commands

```
docker run <image-name>
```

```
ubuntu@ip-172-31-26-120: ~  
ubuntu@ip-172-31-26-120:~$ docker run -it -d ubuntu  
233e926091f338a18d3ba915ad34a6b1bc868642d7f3eb120f91  
ubuntu@ip-172-31-26-120:~$ █
```

This command helps in running containers from their image name.

Common Docker Commands

```
docker ps
```

```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker ps
CONTAINER ID        IMAGE               COMMAND
STATUS              PORTS
233e926091f3        ubuntu              "/bin/bash"
Up About a minute
angry_jennings
ubuntu@ip-172-31-26-120:~$
```

This command helps in listing all the containers which are **running** in the system.

Common Docker Commands

```
docker ps -a
```

```
ubuntu@ip-172-31-26-120: ~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
STATUS              PORTS
NAMES
f0a5fa001b0e        ubuntu              "/bin/bash"
Exited (0) 5 seconds ago
relaxed_clark
233e926091f3        ubuntu              "/bin/bash"
Up 4 minutes
angry_jenning
ubuntu@ip-172-31-26-120:~$
```

If there are any stopped containers, they can be seen by adding the **-a** flag in this command.

Common Docker Commands

```
docker exec <container-id>
```

A terminal window showing the command "docker exec -it 233e926091f3 bash" being run and its output, which is a root prompt in a container. The terminal has a dark background with green text and a blue icon.

```
root@233e926091f3: /  
ubuntu@ip-172-31-26-120:~$ docker exec -it 233e926091f3 bash  
root@233e926091f3:/# █
```

For logging into/accessing the container, one can use the **exec** command.

Common Docker Commands

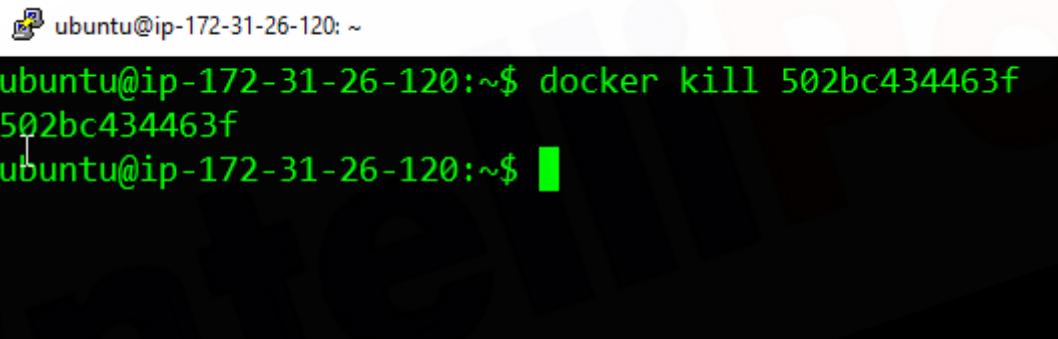
```
docker stop <container-id>
```

```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker stop 233e926091f3
233e926091f3
ubuntu@ip-172-31-26-120:~$ █
```

For stopping a running container, we use the **stop** command.

Common Docker Commands

```
docker kill <container-id>
```



A terminal window showing the command `docker kill 502bc434463f` being run. The output shows the container ID being killed. The terminal prompt is `ubuntu@ip-172-31-26-120:~$`.

```
ubuntu@ip-172-31-26-120:~$ docker kill 502bc434463f
502bc434463f
ubuntu@ip-172-31-26-120:~$
```

This command kills the container by stopping its execution immediately. The difference between **docker kill** and **docker stop**: ‘`docker stop`’ gives the container time to shutdown gracefully; whereas, in situations when it is taking too much time for getting the container to stop, one can opt to kill it.

Common Docker Commands

```
docker rm <container-id>
```

```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker rm 502bc434463f
502bc434463f
ubuntu@ip-172-31-26-120:~$
```

To remove a stopped container from the system, we use the **rm** command.

Common Docker Commands

```
docker rmi <image-id>
```

A small icon of a terminal window with a cursor, located to the left of the terminal prompt.

```
ubuntu@ip-172-31-26-120: ~$ docker rmi 93fd78260bd1
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:6d0e0c26489e33f5a6f0020edface27
71f23c49
Deleted: sha256:93fd78260bd1495afb484371928661f63e64be3
Deleted: sha256:1c8cd755b52d6656df927bc8716ee0905853fad
Deleted: sha256:9203aabb0b583c3cf927d2caf6ba5b11124b0a2
Deleted: sha256:32f84095aed5a2e947b12a3813f019fc69f159c
Deleted: sha256:bc7f4b25d0ae3524466891c41cefc7c6833c533
ubuntu@ip-172-31-26-120:~$ █
```

A terminal window showing the execution of the `docker rmi` command to remove an image. The command `docker rmi 93fd78260bd1` is run, followed by several lines of output showing the removal of multiple untagged and tagged versions of the image, each resulting in a `Deleted:` message. The terminal prompt `ubuntu@ip-172-31-26-120:~$` is visible at the bottom.

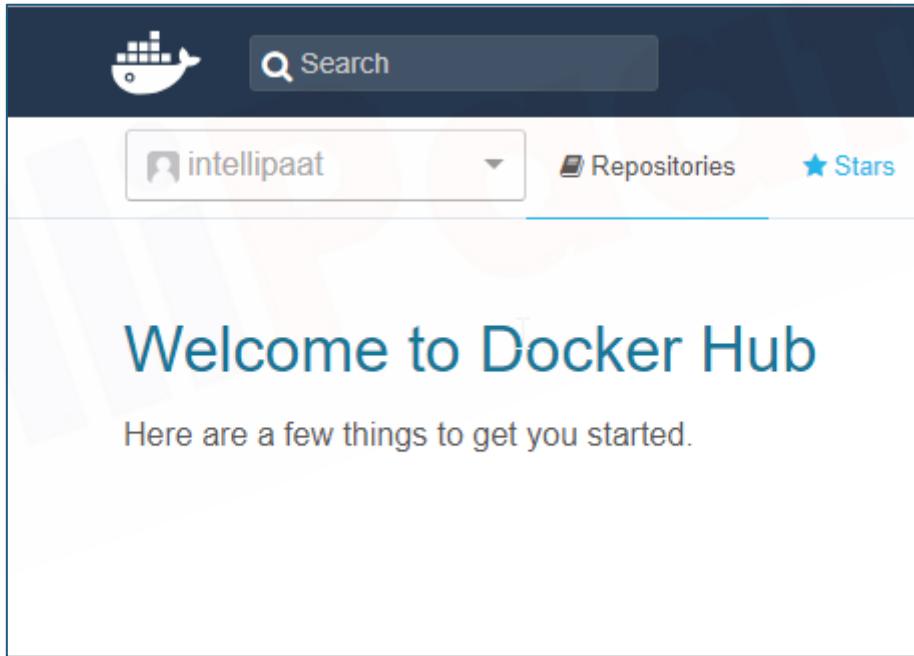
To remove an image from the system, we use the **rmi** command.

Creating a Docker Hub Account

Creating a Docker Hub Account



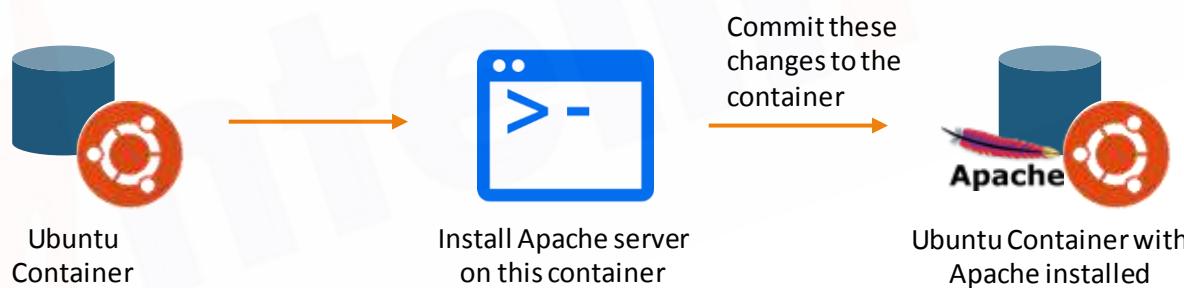
1. Navigate to <https://hub.docker.com>
2. Sign up on the website
3. Agree to the terms and conditions
4. Click on Sign up
5. Check your email, and verify your email by clicking on the link
6. Finally, login using the credentials you provided on the sign up page



Committing Changes to a Container

Committing Changes to a Docker Container

Let's try to accomplish the following example with a container and see how we can commit this container into an image.



Committing Changes to a Docker Container

1. Pull the Docker Container using the command:

```
docker pull ubuntu
```

```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
32802c0cfa4d: Pull complete
da1315cffa03: Pull complete
fa83472a3562: Pull complete
f85999a86bef: Pull complete
Digest: sha256:6d0e0c26489e33f5a6f0020edface2727db9489
Status: Downloaded newer image for ubuntu:latest
ubuntu@ip-172-31-26-120:~$ █
```

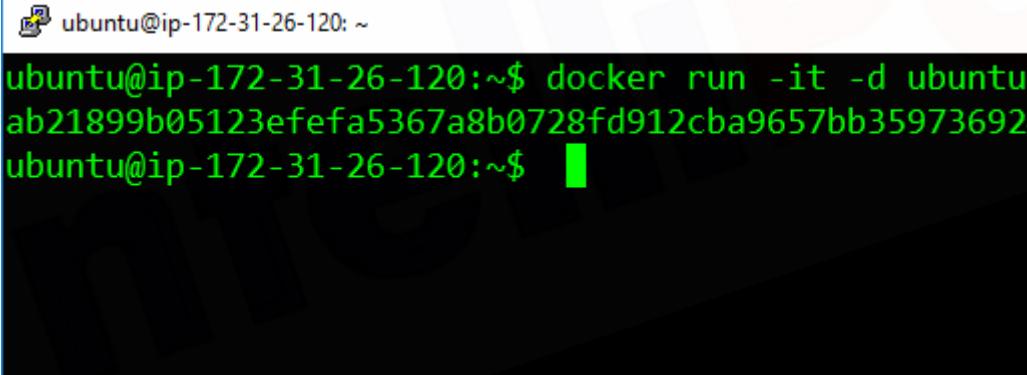
In our case, the image name is “ubuntu”.

Committing Changes to a Docker Container



2. Run the container using the command:

```
docker run -it -d ubuntu
```



A terminal window showing the command `docker run -it -d ubuntu` being run. The output shows a long container ID and a prompt for further commands.

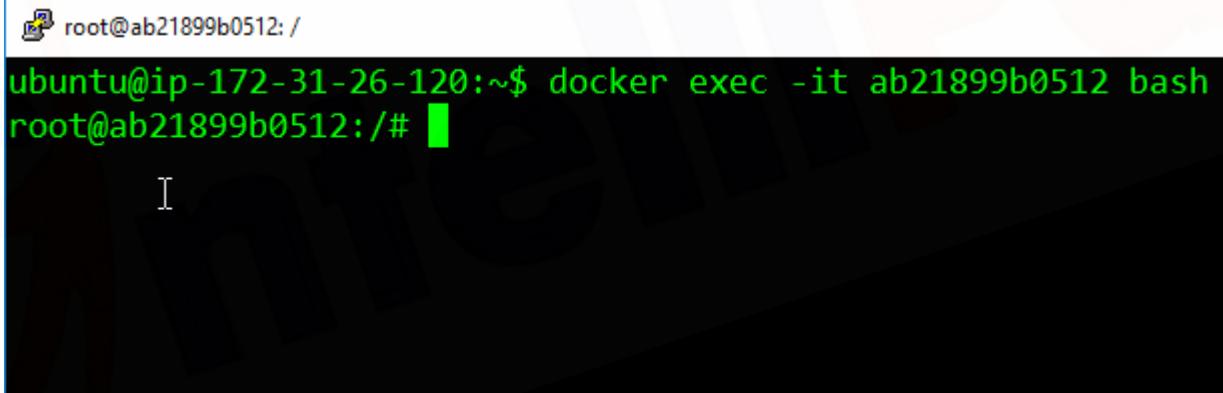
```
ubuntu@ip-172-31-26-120:~$ docker run -it -d ubuntu
ab21899b05123efefa5367a8b0728fd912cba9657bb35973692
ubuntu@ip-172-31-26-120:~$ █
```

Committing Changes to a Docker Container



3. Access the container using the command:

```
docker exec -it <container-id> bash
```



A terminal window showing the command `docker exec -it ab21899b0512 bash` being run. The output shows the user is now inside the container, with the prompt `root@ab21899b0512:/#`. A cursor is visible at the bottom of the terminal window.

```
root@ab21899b0512: /  
ubuntu@ip-172-31-26-120:~$ docker exec -it ab21899b0512 bash  
root@ab21899b0512:/# █  
[
```

Committing Changes to a Docker Container



4. Install Apache2 on this container using the following commands:

```
apt-get update  
apt-get install apache2
```

```
root@ab21899b0512:/# apt-get install apache2  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  apache2-bin apache2-data apache2-utils file libapr1  
  libaprutil1-dbd-sqlite3 libaprutil1-ldap libasn1-8-
```

Committing Changes to a Docker Container



5. Exit the container and save it using this command. The saved container will be converted into an image with the name specified.

```
docker commit <container-id> <username>/<container-name>
```

```
ubuntu@ip-172-31-26-120:~$ docker commit ab21899b0512 intellipaat/apache
sha256:c8446a9b3ca4a6436cbed6765744bbcfaa2e1629e1c25fc54ddfa1e34377326c
ubuntu@ip-172-31-26-120:~$ docker images
REPOSITORY          TAG           IMAGE ID            CREATED             SIZE
intellipaat/apache  latest        c8446a9b3ca4    21 seconds         206MB
```

The **username** has to match with the username you created on DockerHub.

The **container-name** can be anything.

Pushing the Container on DockerHub

Pushing the Container on DockerHub



1. The first step is to login. It can be done using the following command:

`docker login`

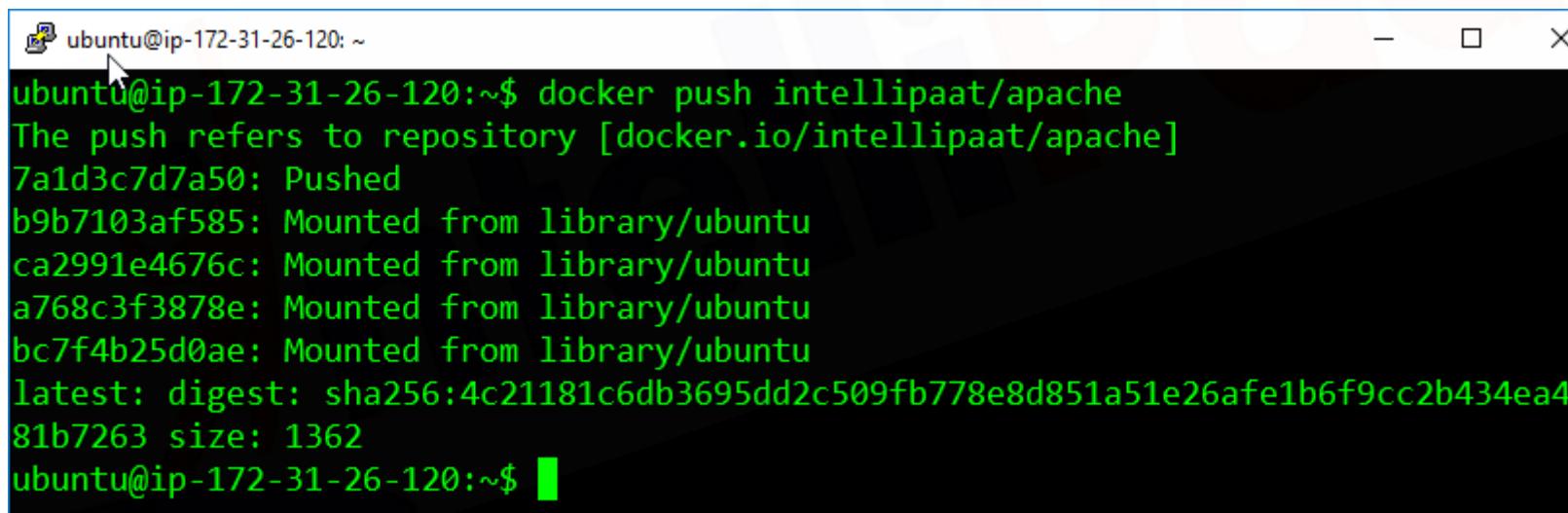
```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker login
Login with your Docker ID to push and pull images fro
have a Docker ID, head over to https://hub.docker.com
Username: intellipaat
Password:
WARNING! Your password will be stored unencrypted in
.json.
Configure a credential helper to remove this warning.
https://docs.docker.com/engine/reference/commandline/
Login Succeeded
ubuntu@ip-172-31-26-120:~$
```

Pushing the Container on DockerHub



2. For pushing your container on DockerHub, use the following command:

```
docker push <username>/<container-id>
```



A terminal window titled "ubuntu@ip-172-31-26-120: ~" showing the output of a docker push command. The command is "docker push intellipaat/apache". The output shows the repository [docker.io/intellipaat/apache] and several layers being pushed, ending with the latest digest.

```
ubuntu@ip-172-31-26-120:~$ docker push intellipaat/apache
The push refers to repository [docker.io/intellipaat/apache]
7a1d3c7d7a50: Pushed
b9b7103af585: Mounted from library/ubuntu
ca2991e4676c: Mounted from library/ubuntu
a768c3f3878e: Mounted from library/ubuntu
bc7f4b25d0ae: Mounted from library/ubuntu
latest: digest: sha256:4c21181c6db3695dd2c509fb778e8d851a51e26afe1b6f9cc2b434ea4
81b7263 size: 1362
ubuntu@ip-172-31-26-120:~$
```

Pushing the Container on DockerHub



3. You can verify the push on DockerHub.

A screenshot of the DockerHub website. At the top, there's a dark header with a white whale icon, a search bar with the placeholder 'Search', and a user dropdown menu showing 'intellipaat'. Below the header, there are navigation links for 'Repositories', 'Stars', and 'Contributed'. The main area is titled 'Repositories' and features a search bar with the placeholder 'Type to filter repositories by name'. A single repository card is visible, belonging to the user 'intellipaat' with the name 'apache' and the status 'public'. The repository card has a small profile picture icon.

Now anyone, who wants to download this container, can simply pass the following command:

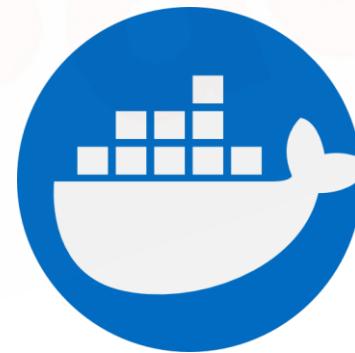
```
docker pull intellipaat/apache
```



Private Registry for Docker

Private Registry for Docker

- ✓ DockerHub is a publicly available Docker Registry
- ✓ You may want to create a Private Registry for your company or personal use
- ✓ The registry is available on DockerHub, as a container named 'registry'

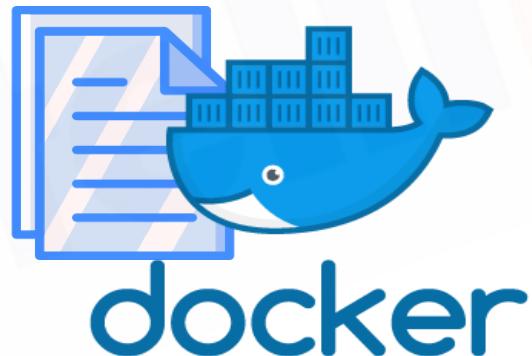


Hands-on: Creating a Private Registry in Docker

Introduction to Dockerfile

Introduction to Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image. Using the **docker** build, users can create an automated build that executes several command-line instructions in succession.



Various Commands in Dockerfile

FROM

ADD

RUN

CMD

ENTRYPOINT

ENV

The **FROM** keyword is used to define the base image, on which we will be building.

Example

FROM ubuntu

Dockerfile

Various Commands in Dockerfile

FROM

ADD

RUN

CMD

ENTRYPOINT

ENV

The **ADD** keyword is used to add files to the container being built. The syntax used is:

ADD <source> <destination in container>

Example

```
FROM ubuntu  
ADD . /var/www/html
```

Dockerfile

Various Commands in Dockerfile

FROM

ADD

RUN

CMD

ENTRYPOINT

ENV

The **RUN** keyword is used to add layers to the base image, by installing components. Each RUN statement adds a new layer to the docker image.

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD . /var/www/html
```

Dockerfile

Various Commands in Dockerfile

FROM

ADD

RUN

CMD

ENTRYPOINT

ENV

The **CMD** keyword is used to run commands on the start of the container. These commands run only when there is no argument specified while running the container.

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD . /var/www/html
CMD apachectl -D FOREGROUND
```

Dockerfile

Various Commands in Dockerfile

FROM

ADD

RUN

CMD

ENTRYPOINT

ENV

The **ENTRYPOINT** keyword is used strictly to run commands the moment the container initializes. The difference between CMD and ENTRYPOINT: ENTRYPOINT will run irrespective of the fact whether the argument is specified or not.

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD . /var/www/html
ENTRYPOINT apachectl -D FOREGROUND
```

Dockerfile

Various Commands in Dockerfile

FROM

ADD

RUN

CMD

ENTRYPOINT

ENV

The **ENV** keyword is used to define environment variables in the container runtime.

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD . /var/www/html
ENTRYPOINT apachectl -D FOREGROUND
ENV name Devops Intellipaat
```

Dockerfile

Running the Sample Dockerfile

Running the Sample Dockerfile



Let's see how we can run this sample Dockerfile now.

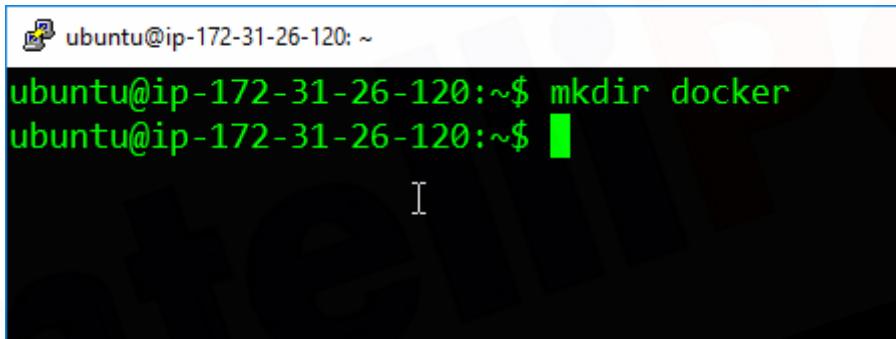
Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD . /var/www/html
ENTRYPOINT apachectl -D FOREGROUND
ENV name Devops Intellipaat
```

Dockerfile

Running the Sample Dockerfile

1. First, create a folder docker in the home directory.

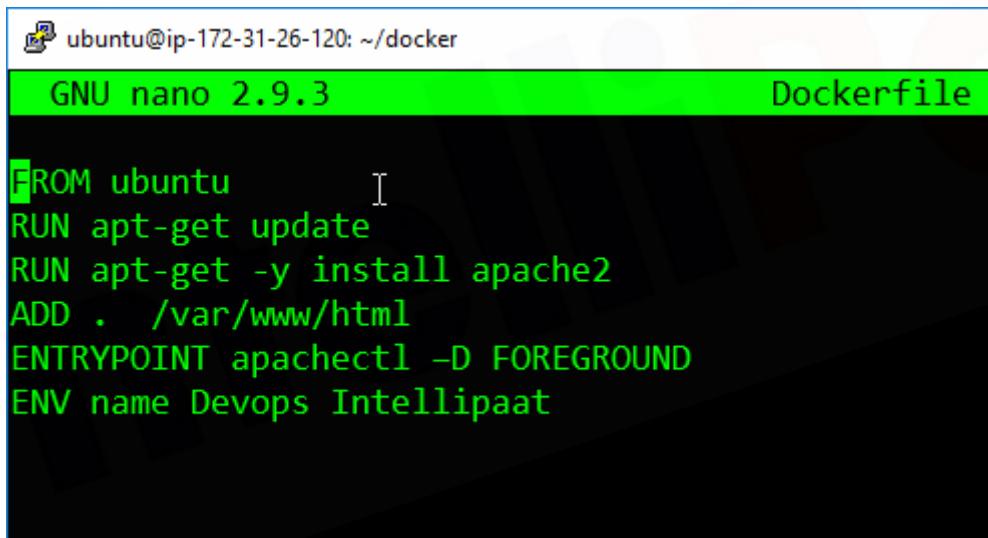


```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ mkdir docker
ubuntu@ip-172-31-26-120:~$ █
```

A screenshot of a terminal window on an Ubuntu system. The prompt shows the user is in their home directory (~). The command 'mkdir docker' is entered and executed. The terminal then displays a new line starting with a green cursor character '█'. The background of the terminal window has a subtle geometric pattern.

Running the Sample Dockerfile

2. Enter into this directory and create a file called 'Dockerfile', with the same contents as the sample Dockerfile.

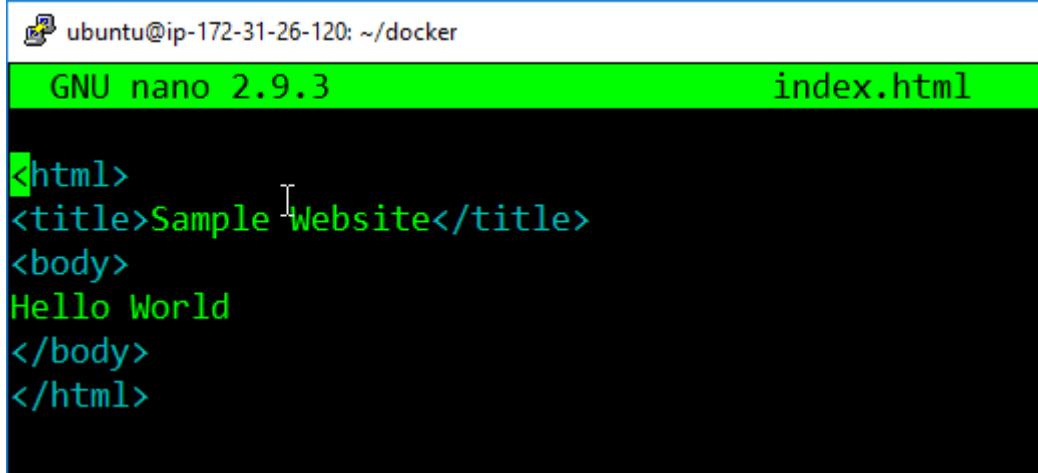


A screenshot of a terminal window titled "Dockerfile". The title bar is green and displays "ubuntu@ip-172-31-26-120: ~/docker" and "GNU nano 2.9.3". The main area of the terminal shows the following Dockerfile code:

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD . /var/www/html
ENTRYPOINT apachectl -D FOREGROUND
ENV name Devops Intellipaat
```

Running the Sample Dockerfile

3. Create one more file called 'index.html' with the following contents.



A screenshot of a terminal window titled "ubuntu@ip-172-31-26-120: ~/docker". The title bar is green, and the main area shows the content of an "index.html" file being edited with the "GNU nano 2.9.3" text editor. The file contains the following HTML code:

```
<html>
<title>Sample Website</title>
<body>
Hello World
</body>
</html>
```

Running the Sample Dockerfile

4. Now, pass the following command:

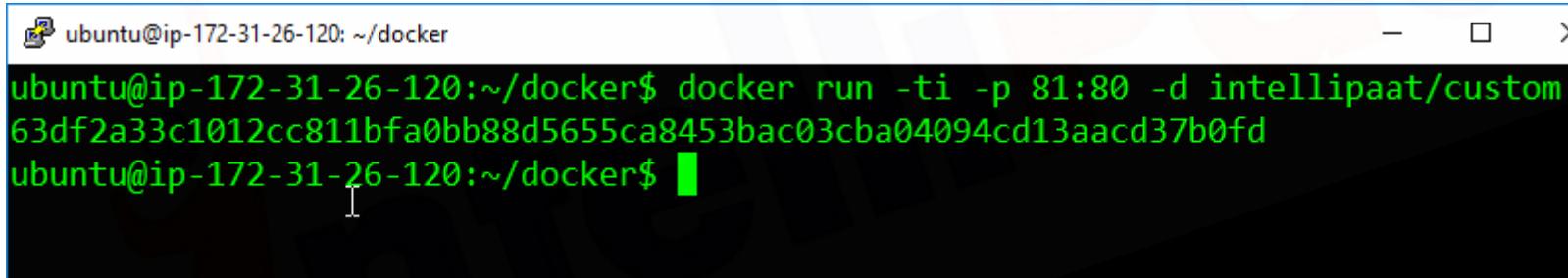
```
docker build <directory-of-dockerfile> -t <name of container>
```

```
ubuntu@ip-172-31-26-120:~/docker$ docker build . -t intellipaat/custom
Sending build context to Docker daemon 3.072kB
Step 1/6 : FROM ubuntu
--> 93fd78260bd1
Step 2/6 : RUN apt-get update
--> Using cache
--> 8ce3e5e6548b
Step 3/6 : RUN apt-get -y install apache2
--> Using cache
--> 296859cef2f0
Step 4/6 : ADD . /var/www/html
--> a3dba497063b
Step 5/6 : ENTRYPOINT apachectl -D FOREGROUND
--> Running in e93d78e6de9d
Removing intermediate container e93d78e6de9d
--> 2a0995664eba
Step 6/6 : ENV name Devops Intellipaat
--> Running in 7497da476b3c
Removing intermediate container 7497da476b3c
--> 73370339b1d4
Successfully built 73370339b1d4
Successfully tagged intellipaat/custom:latest
ubuntu@ip-172-31-26-120:~/docker$ █
```

Running the Sample Dockerfile

5. Finally, run this built image, using the following command:

```
docker run -it -p 81:80 -d intellipaat/custom
```

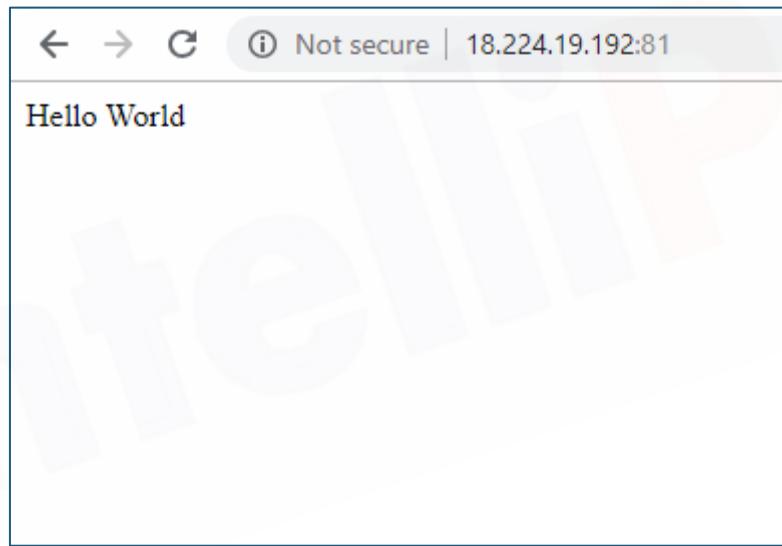


A screenshot of a terminal window titled "ubuntu@ip-172-31-26-120: ~/docker". The window shows the command "docker run -ti -p 81:80 -d intellipaat/custom" being run, followed by the resulting container ID "63df2a33c1012cc811bfa0bb88d5655ca8453bac03cba04094cd13aacd37b0fd". The terminal prompt "ubuntu@ip-172-31-26-120:~/docker\$" is visible at the bottom.

```
ubuntu@ip-172-31-26-120:~/docker$ docker run -ti -p 81:80 -d intellipaat/custom
63df2a33c1012cc811bfa0bb88d5655ca8453bac03cba04094cd13aacd37b0fd
ubuntu@ip-172-31-26-120:~/docker$
```

Running the Sample Dockerfile

6. Now, navigate to the server IP address on port 81.



Running the Sample Dockerfile



7. Finally, login into the container and check the variable \$name. It will have the same value as given in the Dockerfile.

```
root@828bc20911cc: /  
ubuntu@ip-172-31-26-120:~/docker$ docker exec -ti 828bc20911cc bash  
root@828bc20911cc:/# echo $name  
Devops Intellipaat  
root@828bc20911cc:/# [
```

Quiz

Quiz

1. Docker Containers include the kernel of OS as well.

A. True

B. False

1. Docker Containers include the kernel of OS as well.

A. True

B. False

2. How to save an Image of Docker on the disk?

A. Docker save

B. Docker commit

C. Docker push

D. None of these

2. How to save an Image of Docker on the disk?

- A. Docker save
- B. Docker commit**
- C. Docker push
- D. None of these

3. _____ is a service from Docker which provides registry capabilities for public and private Docker Images.

A. Docker Cloud

B. Docker Community

C. Docker Hub

D. None of these

3. _____ is a service from Docker which provides registry capabilities for public and private Docker Images.

- A. Docker Cloud
- B. Docker Community
- C. Docker Hub
- D. None of these

Quiz

4. Virtual Machines include the kernel of the OS.

A. True

B. False

4. Virtual Machines include the kernel of the OS.

A. True

B. False

5. Containers, running on the same machine, share the underlying kernel of the host OS.

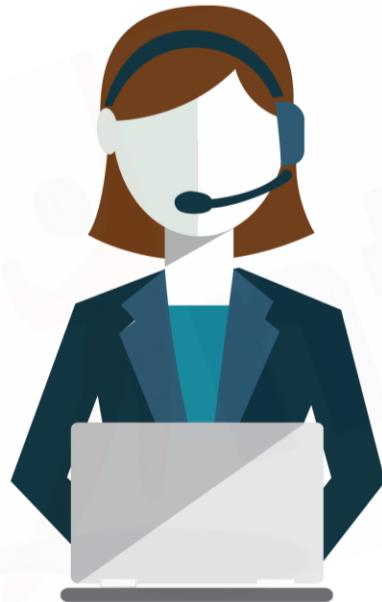
A. True

B. False

5. Containers, running on the same machine, share the underlying kernel of the host OS.

A. True

B. False



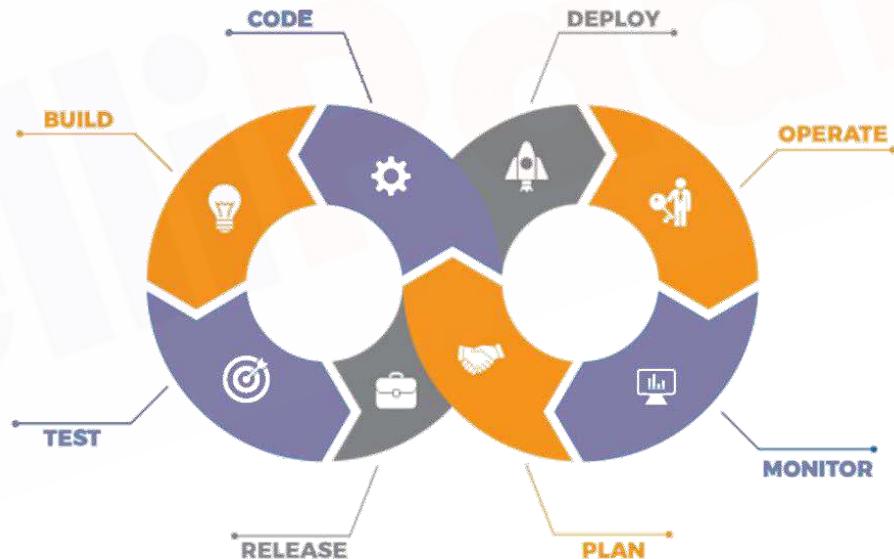
India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor

Containerization Using Docker - II



Agenda

01

Introduction to
Docker Storage

02

Understanding
Microservices

03

Introduction to
Docker Compose

04

What are YAML
files?

05

Introduction to
Docker Swarm

06

Docker
Networks

Introduction to Docker Storage

Introduction to Docker Storage



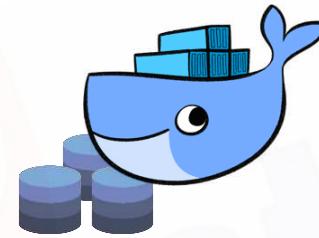
By default, all data of a container is stored on a writable container layer. This layer has the following properties:

- ★ Data only exists while the container is active. If the container no longer exists, the data is also deleted along with the container.
- ★ The writable container layer is tightly coupled with the host machine; hence, it is not portable.
- ★ Data on the writable layer in the container is written using a storage driver.

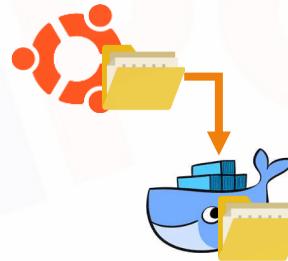


Introduction to Docker Storage

To persist data inside the container, even after it is deleted, we have two options:



Docker
Volumes



Bind
Mounts

Types of Docker Storage



Docker Volumes



Bind Mounts

A **Docker Volume** is a mountable entity which can be used to store data in the docker filesystem.

Syntax

```
docker volume create my-vol
```

```
ubuntu@ip-172-31-45-114: ~
ubuntu@ip-172-31-45-114:~$ docker volume create my-vol
my-vol
ubuntu@ip-172-31-45-114:~$ █
```

Types of Docker Storage



Docker Volumes

A **Docker Volume** is a mountable entity which can be used to store data in the docker filesystem.

Syntax

```
docker run -it --mount  
source=<source=folder>,destination=<destination-folder> -d  
<container-name>
```



Bind Mounts

```
ubuntu@ip-172-31-45-114: ~$ docker run -it -d --mount source=my-vol,destination=/  
app ubuntu  
592f59807209a6881b7fd5fa7de0db2c6a6c97bc48ec0905af3832a0642a4ace  
ubuntu@ip-172-31-45-114: ~$ █
```

Types of Docker Storage



Docker Volumes



Bind Mounts

Bind Mounts mount a directory of the host machine to the docker container.

Syntax

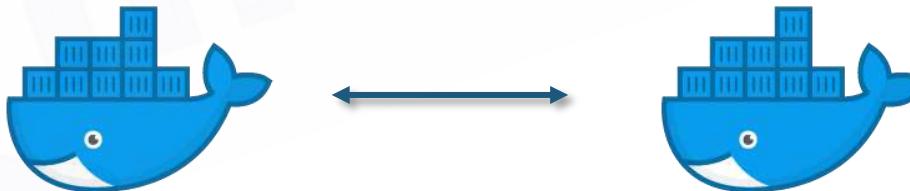
```
docker run -it -v <source-directory>:<destination-directory>
-d <container-name>
```

```
ubuntu@ip-172-31-17-56:~$ docker run -it -v /home/ubuntu/hello:/app -d ubuntu
979e6a564f141f38e9c18bb6d36c569185ea77b3f8d77aece2111c7af2396aa3
ubuntu@ip-172-31-17-56:~$
```

Linking Docker Containers

Linking Docker Containers

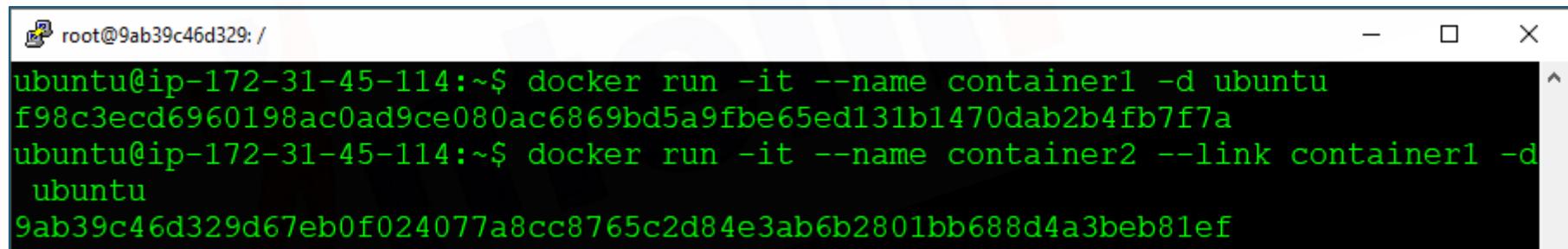
- ★ Linking is a legacy feature of Docker, which is used to connect multiple containers.
- ★ With linking, containers can communicate among each other.
- ★ Name of containers is an important aspect while linking containers.
- ★ Once you link containers, they can reach out to others using their names.



Linking Docker Containers

Syntax

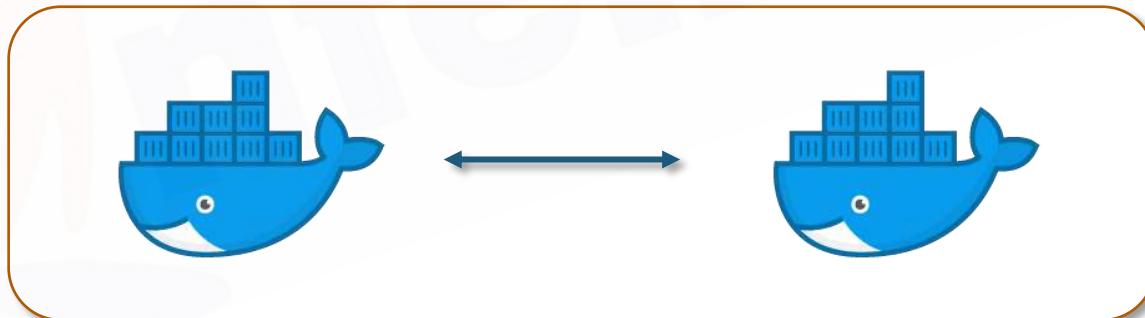
```
docker run -it - -link <name-of-container> -d <image-name>
```

A screenshot of a terminal window titled 'root@9ab39c46d329: /'. The window shows two commands being run: 'docker run -it --name container1 -d ubuntu' followed by 'f98c3ecd6960198ac0ad9ce080ac6869bd5a9fbe65ed131b1470dab2b4fb7f7a'. The second command is partially cut off. Below these, another command is shown: 'ubuntu@ip-172-31-45-114:~\$ docker run -it --name container2 --link container1 -d ubuntu'. The output for this command is '9ab39c46d329d67eb0f024077a8cc8765c2d84e3ab6b2801bb688d4a3beb81ef'. The terminal has a standard Linux-style header with icons for file, terminal, and close buttons.

Hands-on: Linking Docker Containers

Hands-on: Linking Docker Containers

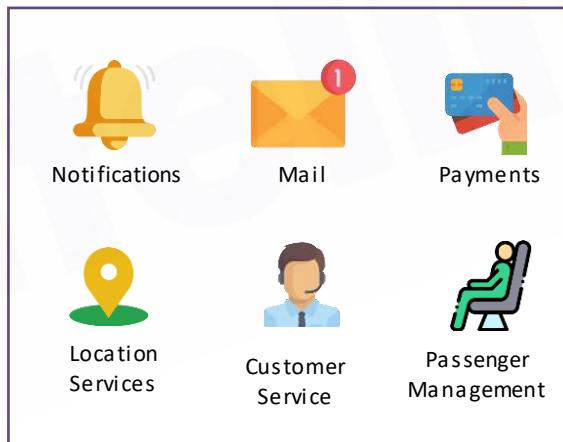
1. Create two containers of Ubuntu with names as follows: container1 and container2
2. Link container2 to container1
3. Try pinging from container2 to container1 by just using the command “ping container1”



Understanding Microservices

What is a Monolithic Application?

A **Monolithic** application is a single-tiered software application in which different components are combined into a single program which resides in a single platform.



Disadvantages of a Monolithic Application



- ✖ Application is large and complex to understand.
- ✖ Entire application has to be re-deployed on an application update.
- ✖ Bug, in any module, can bring down the entire application.
- ✖ It has a barrier to adopting new technologies.

What are Microservices?

Microservices are a software development architectural style that structures an application as a collection of loosely coupled services.



What are Microservices?

Microservices are a software development architectural style that structures an application as a collection of loosely coupled services.



Advantages of Microservices

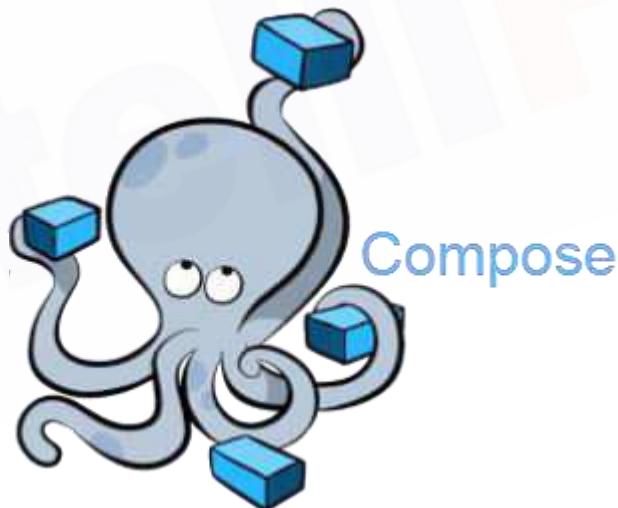


- ✓ Application is distributed, hence easy to understand.
- ✓ The code of only the Microservice which is supposed to be updated is changed.
- ✓ Bug, in one service, does not affect other services.
- ✓ There is no barrier to any specific technology.

Introduction to Docker Compose

What is Docker Compose?

Compose is a tool for defining and running multi-container **Docker** applications. With **Compose**, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. Run **docker-compose up** and **compose** starts and runs your entire app.



Installing Docker Compose

Installing Docker Compose

1. First, download the Docker Compose file using the following command:

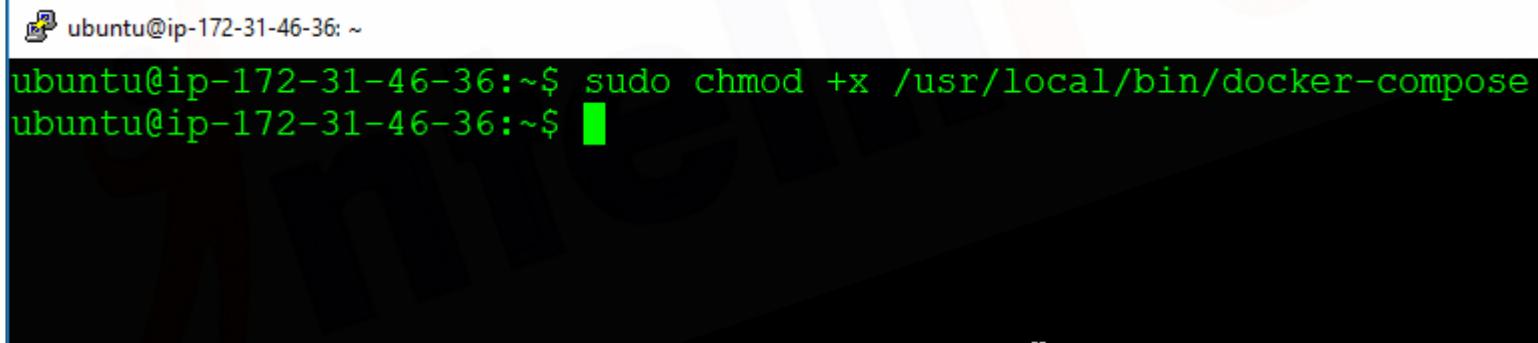
```
sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-  
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
ubuntu@ip-172-31-46-36: ~  
ubuntu@ip-172-31-46-36:~$ sudo curl -L "https://github.com/docker/compo  
es/download/1.23.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/loc  
ker-compose  
% Total    % Received % Xferd[ Average Speed   Time     Time     Time  
                                         Dload  Upload   Total  Spent  Left  
100  617     0  617     0      0  4226      0 --:--:-- --:--:-- --:--:  
100 11.1M  100 11.1M     0      0 22.6M      0 --:--:-- --:--:-- --:--:  
ubuntu@ip-172-31-46-36:~$ █
```

Installing Docker Compose

2. Now, give the required permission to the Docker Compose file to make it executable:

```
sudo chmod +x /usr/local/bin/docker-compose
```



A screenshot of a terminal window on an Ubuntu system. The window has a light gray header bar with the text "ubuntu@ip-172-31-46-36: ~". The main area of the terminal is black with white text. It shows the command "sudo chmod +x /usr/local/bin/docker-compose" being typed in, with the cursor at the end of the command. The command has been partially executed, as indicated by the green text "ubuntu@ip-172-31-46-36:~\$".

Installing Docker Compose

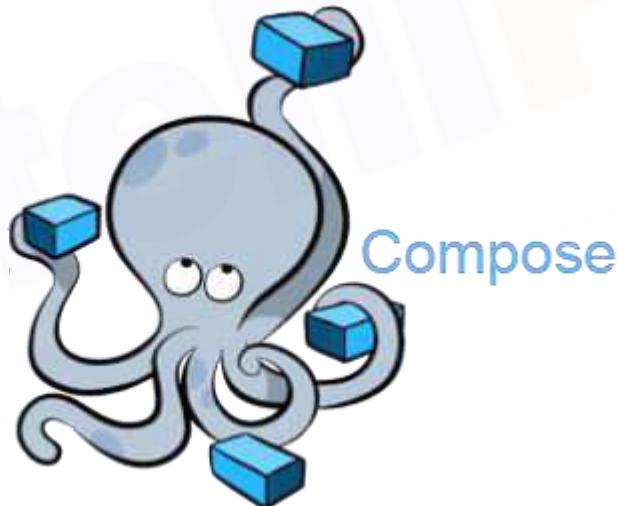
3. Finally, verify your installation using the following command:

```
docker-compose --version
```

```
ubuntu@ip-172-31-46-36: ~
ubuntu@ip-172-31-46-36:~$ docker-compose --version
docker-compose version 1.23.1, build b02f1306
ubuntu@ip-172-31-46-36:~$ █
```

What is Docker Compose?

For deploying containers using Docker Compose, we use YAML files.



What are YAML files?

What are YAML files?

YAML is a superset of a JSON file. There are only two types of structures in YAML which you need to know to get started:



Maps



Lists



What are YAML files?

Maps

Lists

When we map a **key** to a **value** in YAML files,
they are termed as Maps.

<key> : <value>

For example:

Name: Intellipaat
Course: Devops

What are YAML files?

Maps

Lists

YAML Lists are a sequence of objects.

Args

- arg 1
- arg 2
- arg 3

For example:

```
args
  [
    sleep
    - "1000"
    message
    - "Bring back Firefly!"
```

Writing a Docker Compose File

Writing a Docker Compose File



```
version: '3'  
services:  
  sample1:  
    image: httpd  
    ports:  
      - "80:80"  
  sample2:  
    image: nginx
```

Sample Docker Compose File

Hands-on: Running a Sample Docker Compose File

Hands-on: Sample Docker Compose File



1. Create a folder called “docker”
2. Write the sample YAML file in “docker-compose.yml” file
3. To build this docker-compose file, the syntax is as follows:

```
docker-compose up -d
```

4. Ensure that all your containers are running

Hands-on: Deploying WordPress

Hands-on: Deploying WordPress



```
version: '3.3'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
  volumes:
    db_data:
```

docker-compose.yaml

Hands-on: Deploying WordPress

1. Create a folder called “docker-wordpress”
2. Write the sample YAML file in “docker-compose.yml” file
3. To build this docker-compose file, the syntax is as follows:

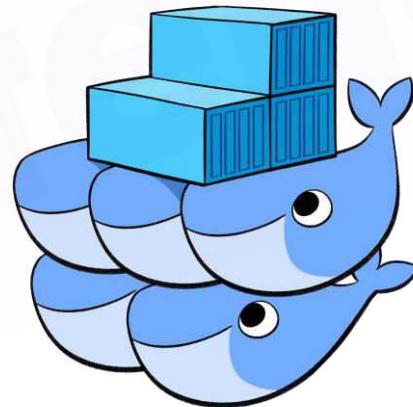
```
docker-compose up -d
```

4. Ensure that all your containers are running

What is Container Orchestration?

What is Container Orchestration?

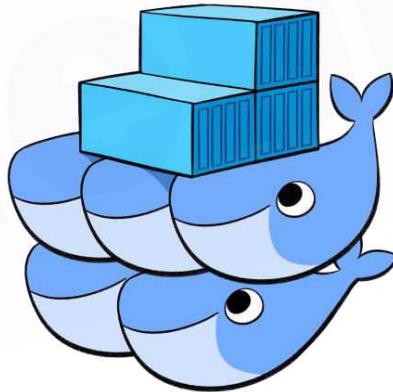
Applications are typically made up of individually containerized components (often called microservices) that must be organized at the networking level in order for the application to run as **intended**. The process of organizing multiple **containers** in this manner is known as **container orchestration**.



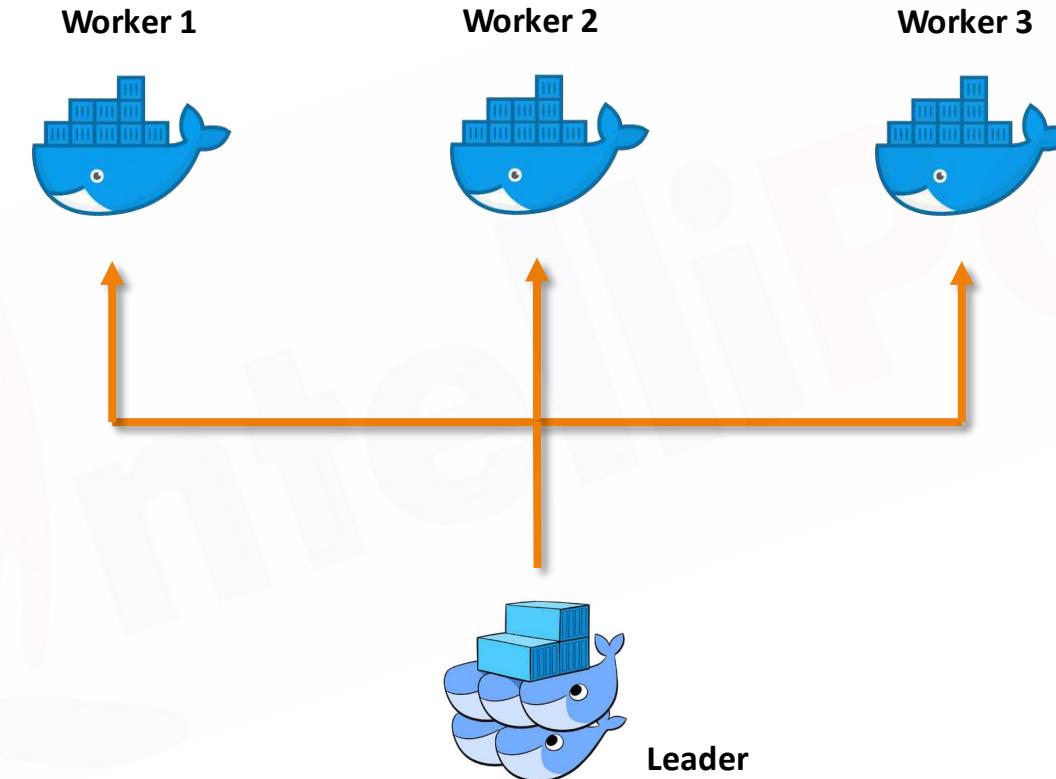
Introduction to Docker Swarm

What is Docker Swarm?

Docker Swarm is a clustering and scheduling tool for **Docker** containers. With **Swarm**, IT administrators and developers can establish and manage a cluster of **Docker** nodes as a single virtual system.



What is Docker Swarm?



Creating a Docker Swarm Cluster

Creating a Docker Swarm Cluster

```
docker swarm init --advertise-addr=<ip-address-of-leader>
```

```
ubuntu@ip-172-31-26-120:~/wordpress$ docker swarm init --advertise-addr=172.31.25.120:2377  
Swarm initialized: current node (ptde8fg2vbxp8py931vrxdpp) is now a manager.
```

```
To add a worker to this swarm, run the following command:
```

```
    docker swarm join --token SWMTKN-1-2m8bntbbysh354anwigivubiqwf21kq6xkww4kjnq  
.26.120:2377
```

```
To add a manager to this swarm, run 'docker swarm join-token manager' and follow
```

```
ubuntu@ip-172-31-26-120:~/wordpress$ █
```

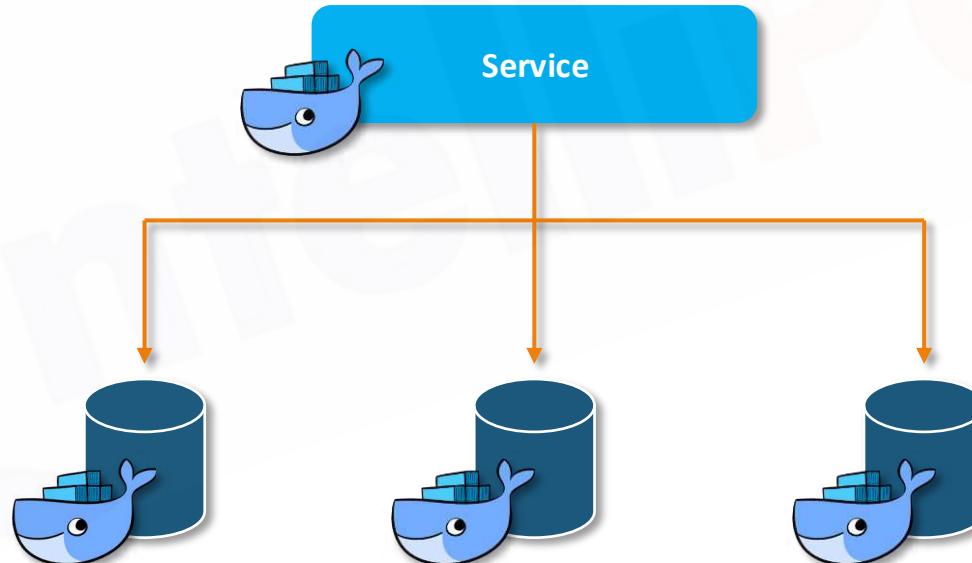
This command should be passed on to the worker node to join the docker swarm cluster.



Introduction to Services

What is a Service?

Containers on the cluster are deployed using **services** on Docker Swarm. A **service** is a long-running **Docker** container that can be deployed to any node worker.



Creating a Service

```
docker service create --name <name-of-service> --replicas <number-of-replicas> <image-name>
```

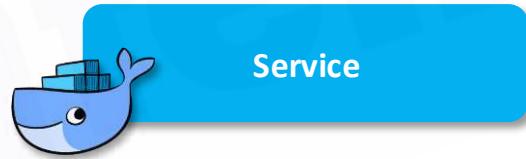
```
ubuntu@ip-172-31-26-120:~$ docker service create --name apache --replicas 3 -p 80:80 hshar/webapp
osftoz95rma0dkbsganqk0f3o
overall progress: [=====>] 3 out of 3 tasks
1/3: running [=====>]
2/3: running [=====>]
3/3: running [=====>]
verify: Service converged
ubuntu@ip-172-31-26-120:~$ █
```

Hands-on: Creating a Service in Docker Swarm

Hands-on: Creating a Service in Docker Swarm



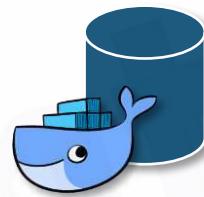
1. Create a Service for nginx webserver
2. There should be 3 replicas of this service running on the swarm cluster
3. Try accessing the service from Master IP and Slave IP



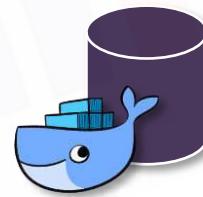
Docker Networks

Why Docker Networks?

Let's take an example. Say, there are two containers which we deploy in the docker ecosystem.



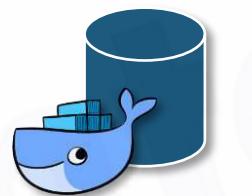
Website Container



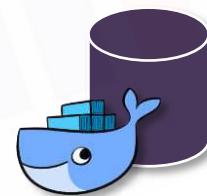
Database Container

Why Docker Networks?

By default, these containers cannot communicate with each other.



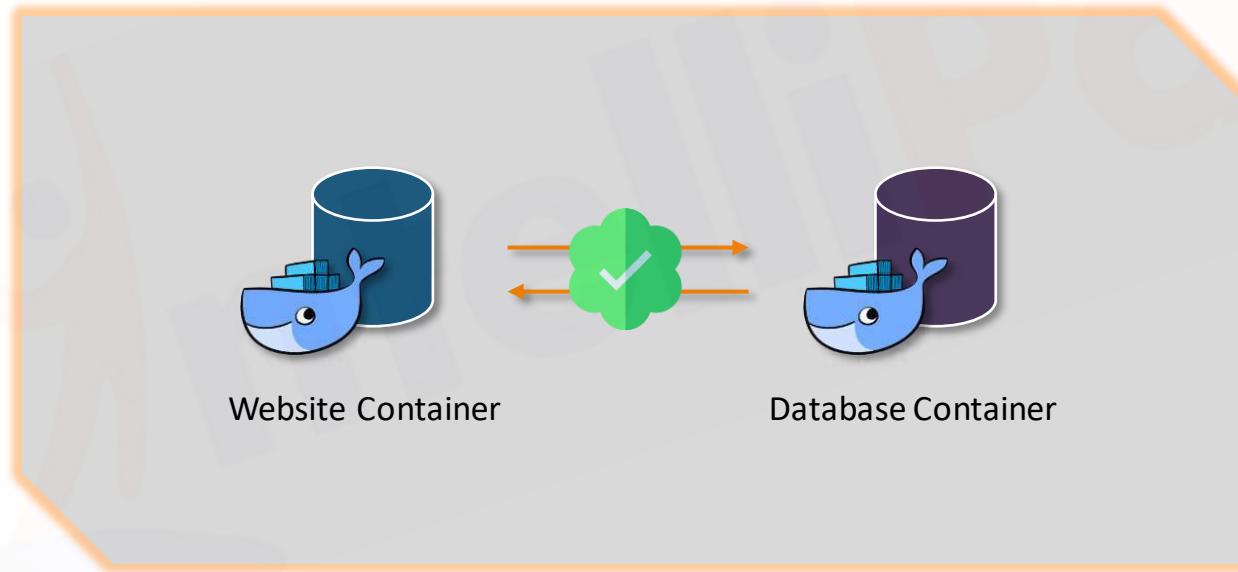
Website Container



Database Container

Why Docker Networks?

Therefore, in-order to have interactions between Docker Containers, we need Docker Networks.



Docker Network

What are Docker Networks?

One of the reasons Docker containers and services are so powerful is that you can connect them together or connect them to non-Docker workloads. And, this can be accomplished using Docker Networks.



Docker Network Types

Docker Networks are of the following types:

bridge

host

overlay

macvlan

none

Docker Network Types

bridge

host

overlay

macvlan

none

Bridge Networks

The default network driver. If you don't specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.

Docker Network Types

bridge

host

overlay

macvlan

none

Host Networks

For standalone containers, remove network isolation between the container and the Docker host and use the host's networking directly. Host is only available for swarm services on Docker 17.06 and higher.

Docker Network Types

bridge

host

overlay

macvlan

none

Overlay Networks

Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container or between two standalone containers on different Docker daemons.

Docker Network Types

bridge

host

overlay

macvlan

none

Macvlan Networks

Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses.

Docker Network Types

bridge

host

overlay

macvlan

none

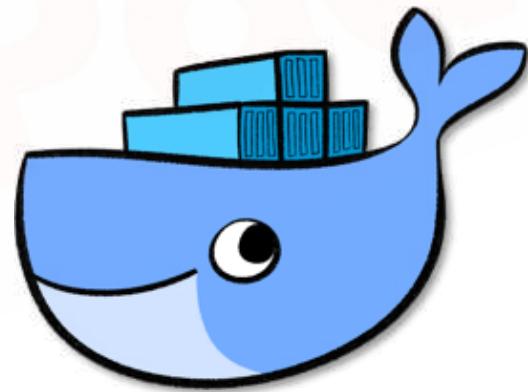
None

For this container, disable all networking. This is usually used in conjunction with a custom network driver. And, none is not available for swarm services.

Hands-on: Deploying a Multi-tier App in Docker Swarm

Hands-on: Multi-tier App in Docker Swarm

1. Create an overlay network named “my-overlay”
2. Deploy a website container in the overlay network
 - Image: hshar/webapp
3. Deploy a database container in the overlay network
 - image: **hshar/mysql:5.6**
 - username: **root** password: **intelli**
4. Make changes in the website code to point to MySQL service
5. Test the configuration by entering values in the website





Quiz

1. _____ is a document used to deploy multiple containers at once.

- A. Docker File
- B. Docker Compose File
- C. Docker Network
- D. None of these

1. _____ is a document used to deploy multiple containers at once.

- A. Docker File
- B. Docker Compose File**
- C. Docker Networks
- D. None of these

2. Which of these is used to mount a directory from the hard disk.

- A. Docker Volumes
- B. Bind Mounts
- C. Docker Network
- D. None of these

2. Which of these is used to mount a directory from the hard disk.

- A. Docker Volumes
- B. Bind Mounts**
- C. Docker Network
- D. None of these

3. For Building a Microservices Architecture, which of the following should you choose?

- A. Docker Compose
- B. Docker Volumes
- C. Docker Swarm
- D. None of these

3. For Building a Microservices Architecture, which of the following should you choose?

- A. Docker Compose
- B. Docker Volumes
- C. Docker Swarm
- D. None of these

4. The Docker Volume of type local is available throughout the swarm cluster.

A. True

B. False

4. The Docker Volume of type local is available throughout the swarm cluster.

A. True

B. False

5. A Docker Swarm service can have more than one containers.

A. True

B. False

5. A Docker Swarm service can have more than one containers.

A. True

B. False



India: +91-7847955955



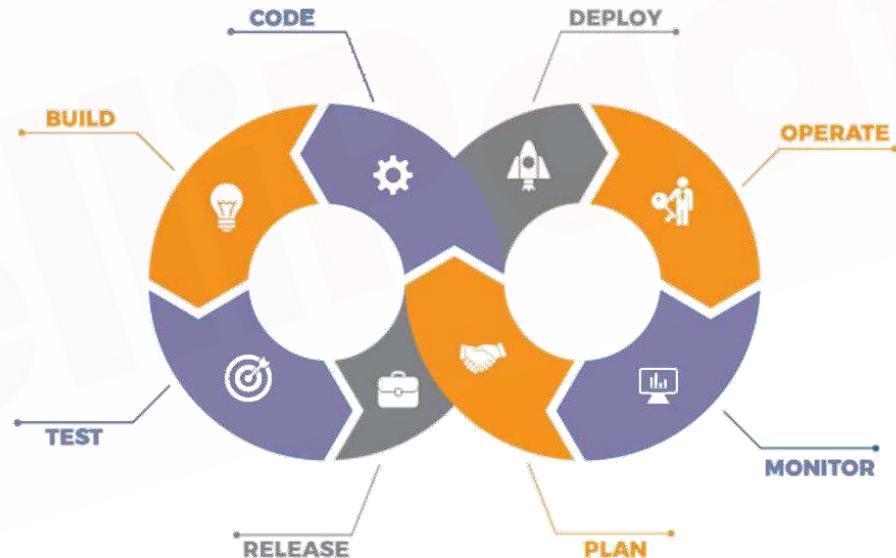
US: 1-800-216-8930 (TOLL FREE)



support@intellipaat.com

24/7 Chat with Our Course Advisor

Configuration Management Using Puppet



Agenda

01 Why Configuration Management?

02 What is Configuration Management?

03 Configuration Management Tools

04 What is Puppet?

05 Puppet Architecture

06 Puppet Master-Slave Setup

07 Puppet Code Basics

08 Applying Configuration Using Classes

Why Configuration Management?

Why Configuration Management?

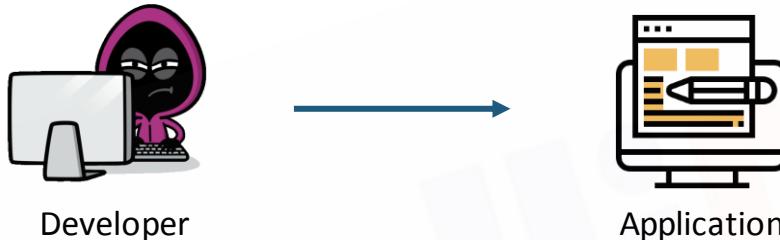


Developer

Why Configuration Management?



Why Configuration Management?

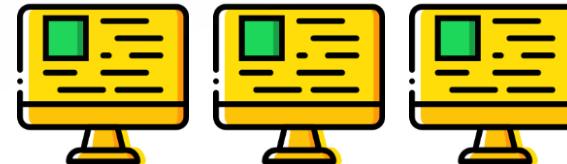


PHP 5.7
MySQL 4.7

Application



PHP Servers



Database Servers

Why Configuration Management?



Developer

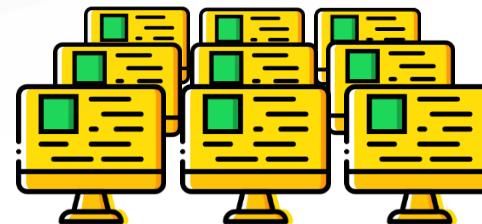


Application

PHP 5.7
MySQL 4.7



PHP 5.7 Servers



Database 4.7 Servers

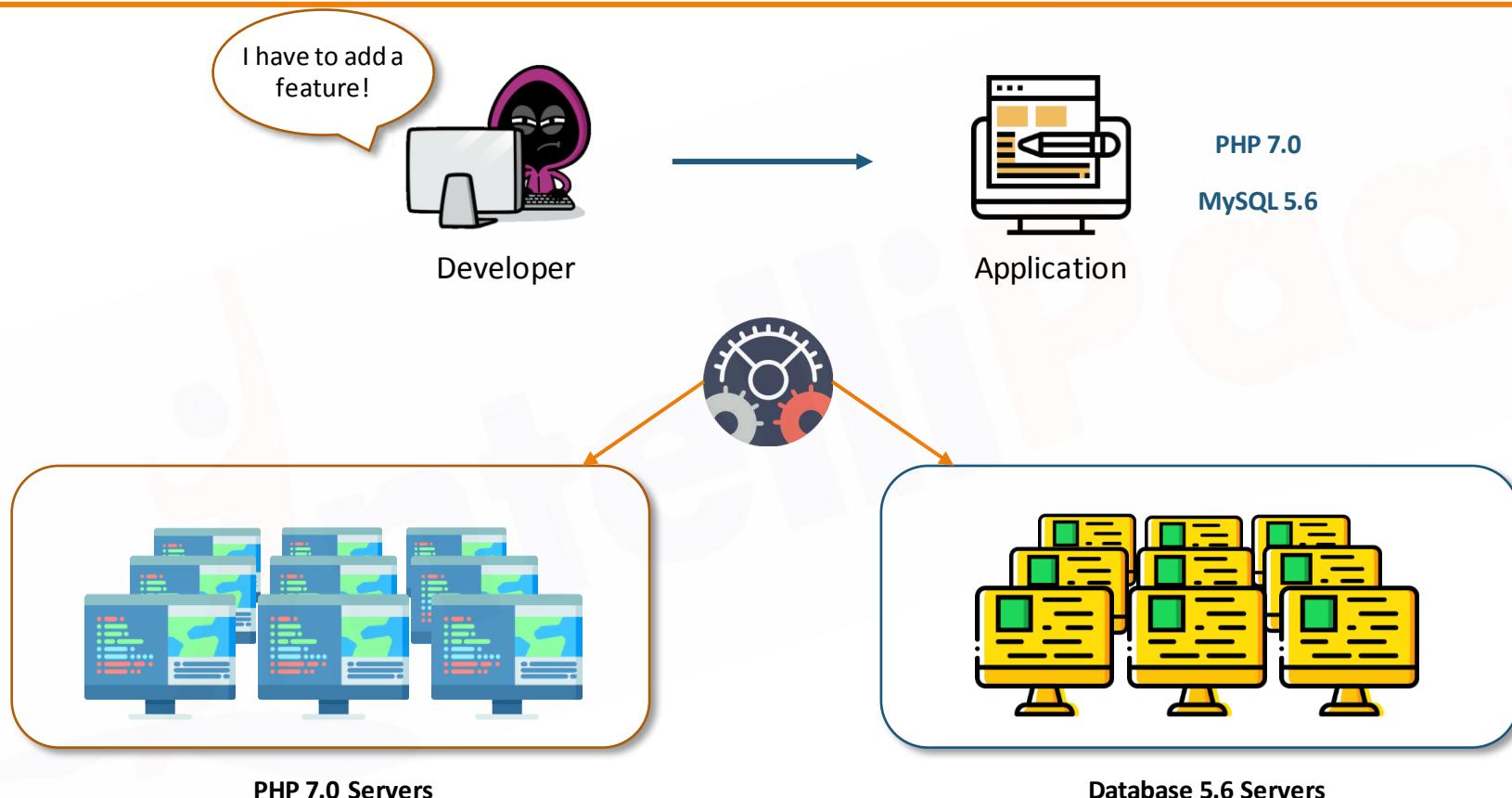
What is Configuration Management?

What is Configuration Management?

Configuration management is a systems engineering process for establishing and maintaining consistency of a product's performance, functional and physical attributes with its requirements, design and operational information throughout its life.



What is Configuration Management?



Configuration Management Features

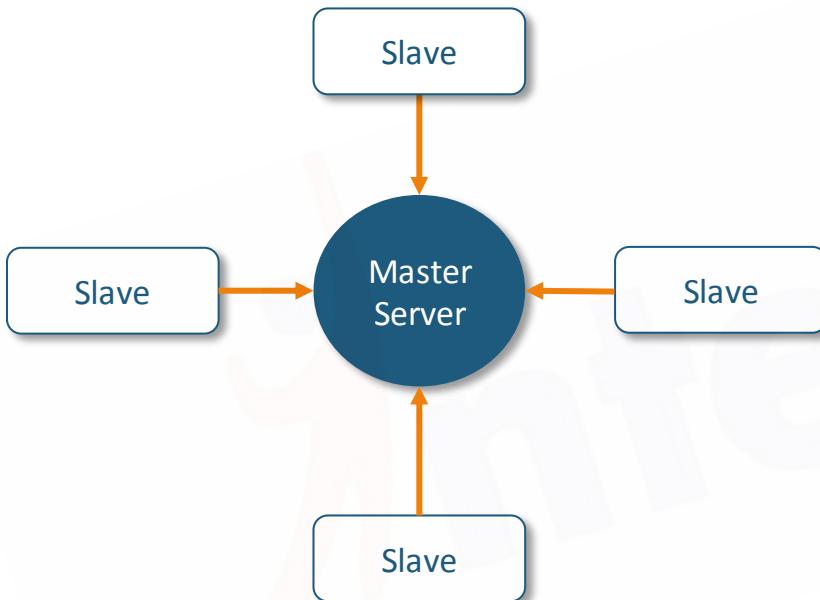
- ★ Automation
- ★ Consistency
- ★ Software Updates
- ★ Software Rollback



Configuration Management Tools

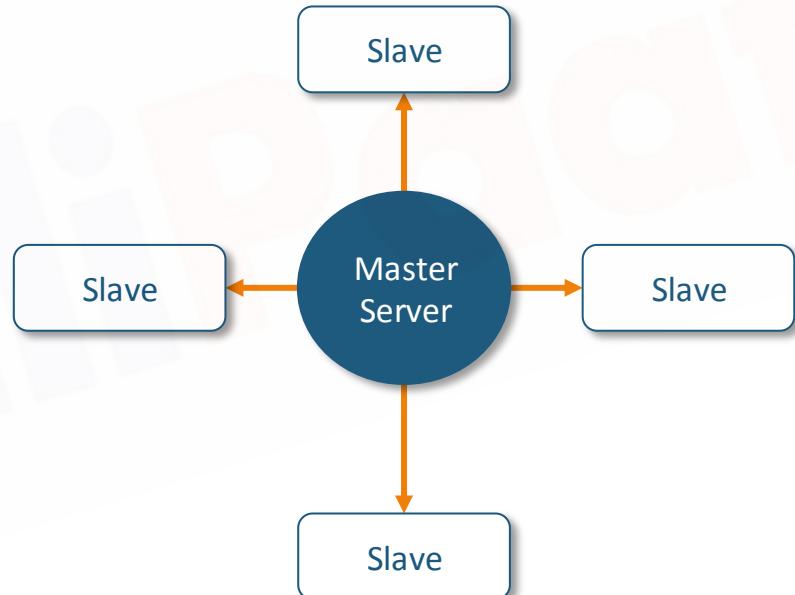
Types of Configuration Management Tools

Pull Configuration



Changes are pulled

Push Configuration



Changes are pushed

Types of Configuration Management Tools



Pull Configuration



Push Configuration



What is Puppet?

What is Puppet?

Puppet is an open-source software configuration management tool. It runs on many Unix-like systems, as well as on Microsoft Windows, and includes its own declarative language to describe the system configuration.



Key Features of Puppet

- ★ Large User Base
- ★ Big Open-source Community
- ★ Documentation
- ★ Platform Support

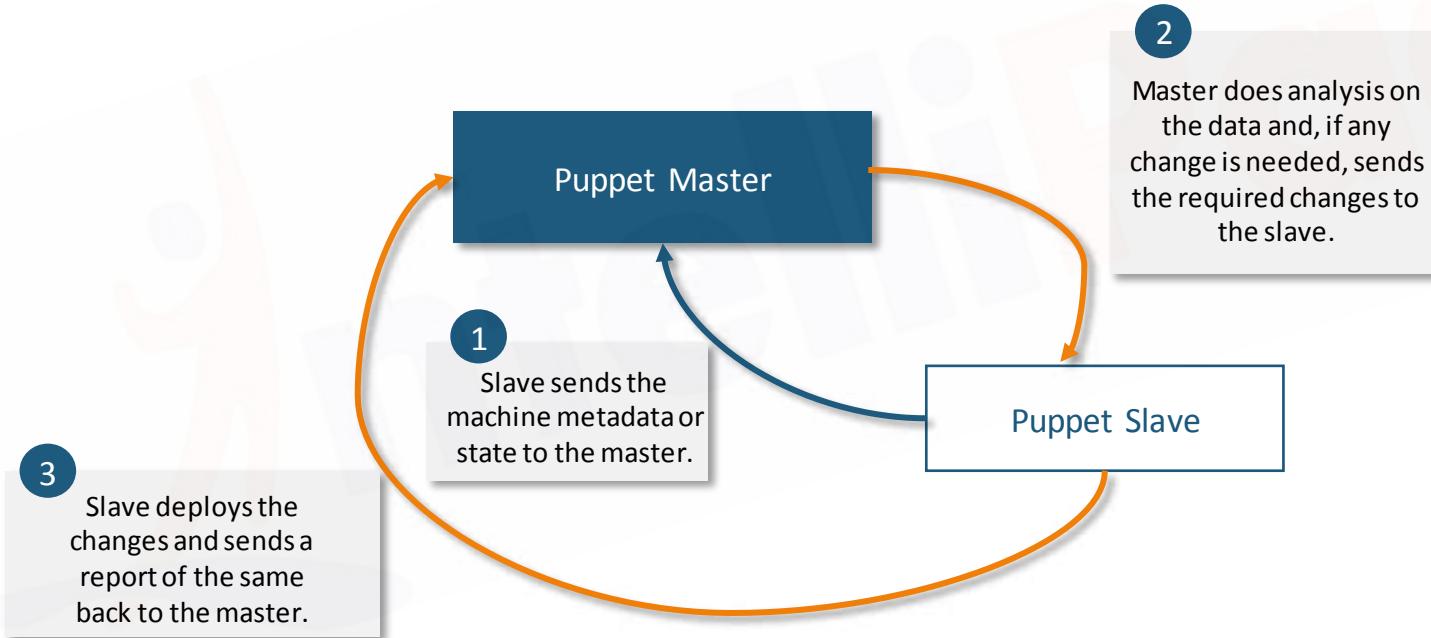




Puppet Architecture

Puppet Architecture

Puppet follows a Master–Slave architecture, the working of which has been explained in the below diagram.



Puppet Architecture: SSL Connection

Because Puppet nodes have to interact with the master, all the information which is communicated between the master node and slave nodes are encrypted using SSL certificates.
The certificate signing process is as follows:



Setting up Puppet Master–Slave on AWS

Code Basics for Puppet

Code Basics for Puppet

The most basic component of Puppet Code is a **resource**. A resource describes something about the state of the system, such as if a certain user or file should exist, or a package should be installed, etc.

Syntax

```
resource_type { 'resource_name':  
    attribute => value,  
    ...  
}
```

Code Basics for Puppet: Resource Example

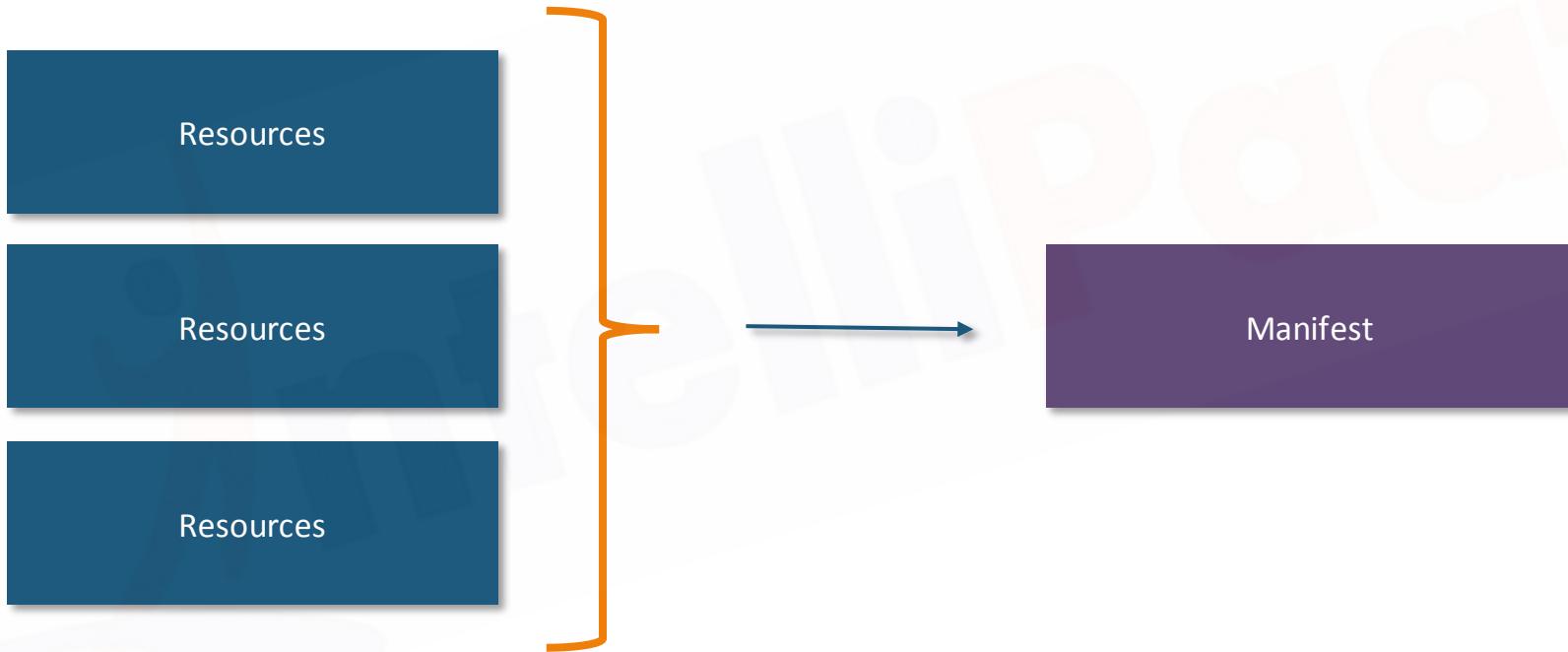


Example

```
package { 'nginx':  
    ensure => 'installed',  
}
```

Sample nginx package

Code Basics for Puppet: Manifest



Code Basics for Puppet: Manifest



Manifests are basically a collection of resource declarations, using the extension **.pp**.

Example

```
package { 'nginx':
  ensure => 'installed',
}

file {'/tmp/hello.txt':
  ensure => present,
  content => 'hello world',
  mode => '0644',
}
```

Sample Manifest File

Code Basics for Puppet: Manifest



Variables

Variables can be defined at any point in a manifest. The most common types of variables are strings and arrays of strings, but other types are also supported, such as Booleans and hashes.

Loops

Conditions

Example

```
$text = "hello world"

file {'/tmp/hello.txt':
  ensure => present,
  content => 'hello world',
  mode => '0644',
}
```

Code Basics for Puppet: Manifest



Variables

Loops

Conditions

Loops are typically used to repeat a task using different input values. For instance, instead of creating 10 tasks for installing 10 different packages, you can create a single task and use a loop to repeat the task with all different packages you want to install.

Example

```
$packages = ['nginx','mysql-server']

package { $packages:
  ensure => installed,
}
```

Code Basics for Puppet: Manifest



Variables

Loops

Conditions

Conditions can be used to dynamically decide whether or not a block of code should be executed, based on a variable or an output from a command, for instance.

Example

```
exec { "Test":  
  command => '/bin/echo apache2 is installed > /tmp/status.txt',  
  onlyif => '/bin/which apache2',  
}
```

Code Basics for Puppet: Manifest



Variables

Loops

Conditions

Conditions can be used to dynamically decide whether or not a block of code should be executed, based on a variable or an output from a command, for instance.

Example

```
exec { "Test":  
  command => '/bin/echo apache2 is not installed > /tmp/status.txt',  
  unless => '/bin/which apache2',  
}
```

Applying Configuration Using Modules

What are Modules?

A collection of manifests and other related files organized in a predefined way to facilitate sharing and reusing parts of a provisioning

1

`sudo puppet module generate <name>`

2

Edit the `init.pp` with a class, and build the module

3

Finally, install the module

What are Classes?

Just like with regular programming languages, classes are used in Puppet to better organize the provisioning and make it easier to reuse portions of the code.

Example

```
Class hello{
```

```
    exec { "Test":  
        command => '/bin/echo apache2 is installed > /tmp/status.txt',  
        unless => '/bin/which apache2',  
    }
```

```
}
```

Hands-on: Applying Configuration Using Modules

Hands-on: Invoking Module's Classes Based on Node Names



Quiz

1. Which of these can be re-used in a Puppet program?

- A. Resource
- B. Manifest
- C. Class
- D. None of these

1. Which of these can be re-used in a Puppet program?

A. Resource

B. Manifest

C. Class

D. None of these

2. What is the mode of communication between the Puppet Master and Slaves?

- A. SSH
- B. SSL Certificates
- C. RDP
- D. None of these

2. What is the mode of communication between the Puppet Master and Slaves?

- A. SSH
- B. SSL Certificates
- C. RDP
- D. None of these

3. Can we create Modules manually rather than using the utility?

A. Yes

B. No

3. Can we create Modules manually rather than using the utility?

A. Yes

B. No

4. Which of these is the main manifest file?

A. init.pp

B. site.pp

C. main.pp

D. None of these

4. Which of these is the main manifest file?

A. init.pp

B. site.pp

C. main.pp

D. None of these

5. Which Loop statement allows the command to execute if the condition is true?

A. unless

B. onlyif

5. Which Loop statement allows the command to execute if the condition is true?

A. unless

B. onlyif



India: +91-7847955955



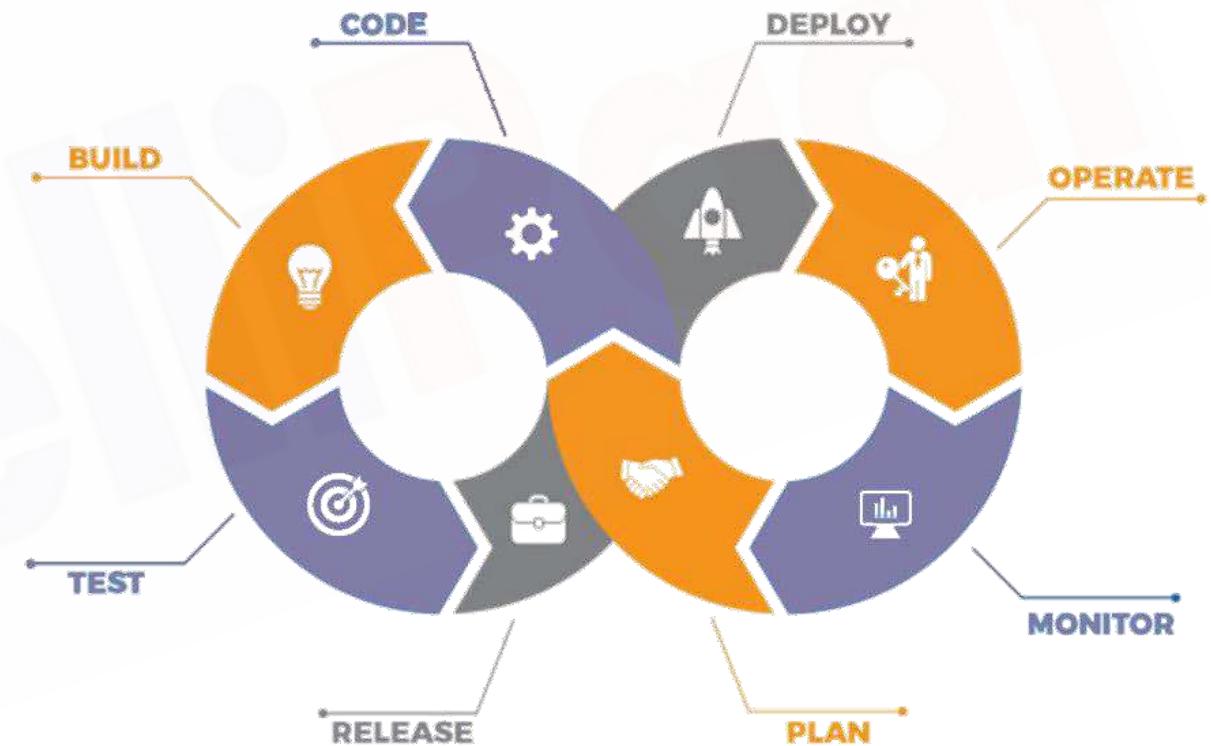
US: 1-800-216-8930 (TOLL FREE)



sales@intellipaat.com

24/7 Chat with Our Course Advisor

Introduction to Ansible



Agenda

01 WHAT IS ANSIBLE?

02 WHY ANSIBLE?

03 HOW DOES
ANSIBLE WORK?

04 CASE STUDY:
NASA

05 SETTING UP
MASTER SLAVE

06 ANSIBLE
PLAYBOOKS

07 ANSIBLE ROLES

08 USING ROLES IN
PLAYBOOK



What is Ansible?

What is Ansible?

- ★ Ansible is an open-source configuration management tool
- ★ Used for configuration management
- ★ Can solve wide range of automation challenges
- ★ Written by Michael DeHaan
- ★ Named after a fictional communication device, first used by Ursula K. LeGuin in her novel Rocannon's World in 1966
- ★ In 2015 Red Hat acquired Ansible

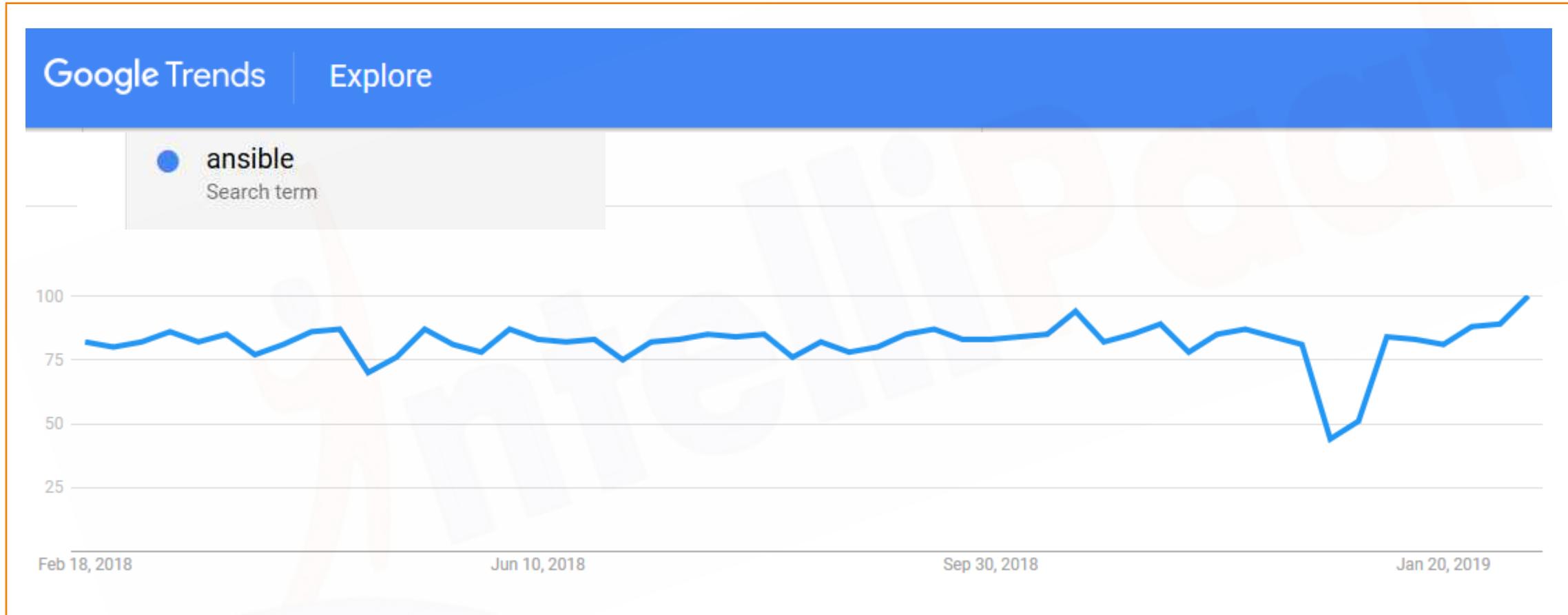


ANSIBLE



Why Ansible?

Why Ansible?



Google Trends Results for Ansible

Career Opportunities of Ansible



DevOps Engineer

BlackBuck Logistics 3 reviews - Bengaluru, Karnataka

₹15,00,000 - ₹17,00,000 a year

Responsibilities and Duties

- 3 - 8 years of experience
- Hands-on experience with any flavour of Linux and can perform basic administrative tasks
- Hands-on experience working with AWS (EC2, VPC, S3, EBS, RDS, IAM, etc)
- Familiarity with a CI/CD system (e.g. Jenkins, Ansible, Puppet)
- Familiarity with a monitoring & alerting system (e.g. Nagios, NewRelic, etc)
- Has an understanding of web architecture, distributed systems, single points of failures, etc.
- Hands-on with a scripting language (preferably Python)
- Good Networking Fundamentals - understands SSH, DNS, DHCP, Load Balancing, Firewalls, etc.
- Basic knowledge of Security good practices e.g. firewalls, etc.
- Worked in an Indian Startup before



Software Engineer, Sr. Principal

Epsilon India  4 reviews - Bengaluru, Karnataka

Must Have:

- Strong knowledge of configuration management process using software such as Ansible, Puppet or Chef.
- Experience with monitoring tools like Nagios, Munin, Zenoss, etc.
- Experience with Release Engineering and Continuous Integration using tools like Maven, Jenkins, etc.
- Configuring, setting up and tuning of JBOSS, Tomcat, WebSphere, WebLogic, Apache, HAProxy servers or equivalent.
- Experience with using tools like Git, SVN etc and knowledge of SCM concepts.



EPSILON®

The Epsilon logo consists of the word "EPSILON" in a bold, sans-serif font, with a registered trademark symbol (®) at the top right. The entire logo is enclosed within a black rectangular border.

Advantage of Ansible

-  Easy to learn
-  Written in Python
-  Easy installation and configuration steps
-  No need to install ansible on slave
-  Highly scalable

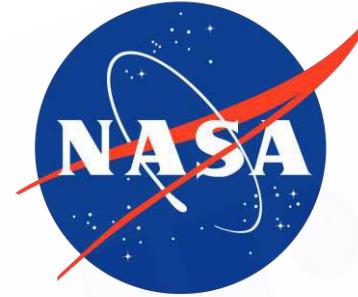


ANSIBLE

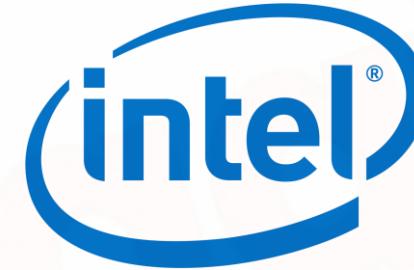
Popularity of Ansible



Apple



NASA



Intel



Percussion



Cisco



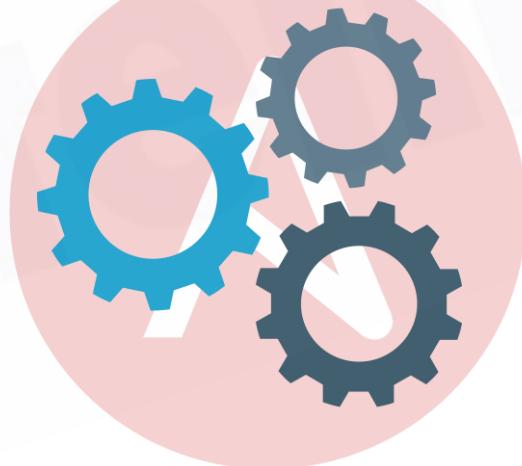
Twitter

How does Ansible work?

How does Ansible work?

With the help of **Ansible Playbooks**,
which are written in a very simple language, **YAML**

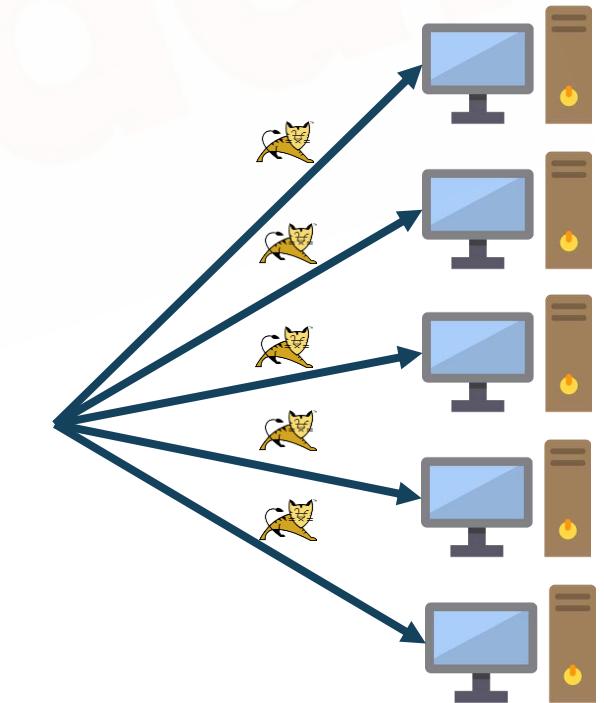
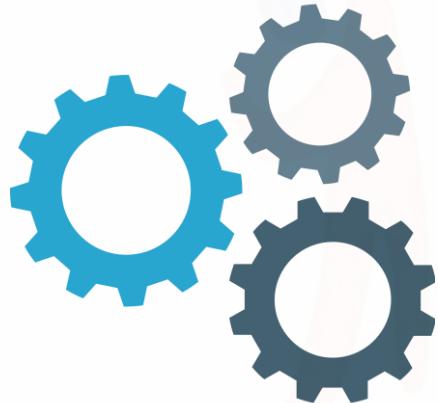
Configuration Management



Problem Statement

Say, Josh runs an enterprise, wants to install a new version of Apache Tomcat in all the systems

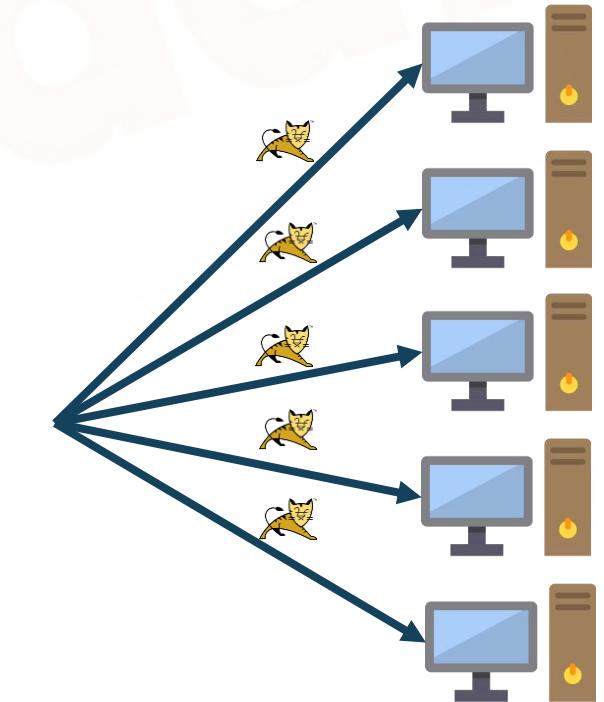
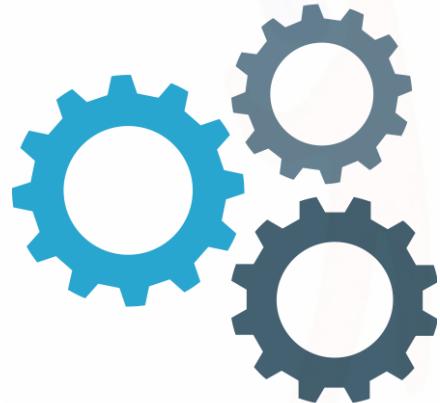
Configuration Management



Problem Statement-Solution with Ansible

Instead of going to each system, manually updating, Josh can use Ansible to automate the installation using Ansible Playbooks

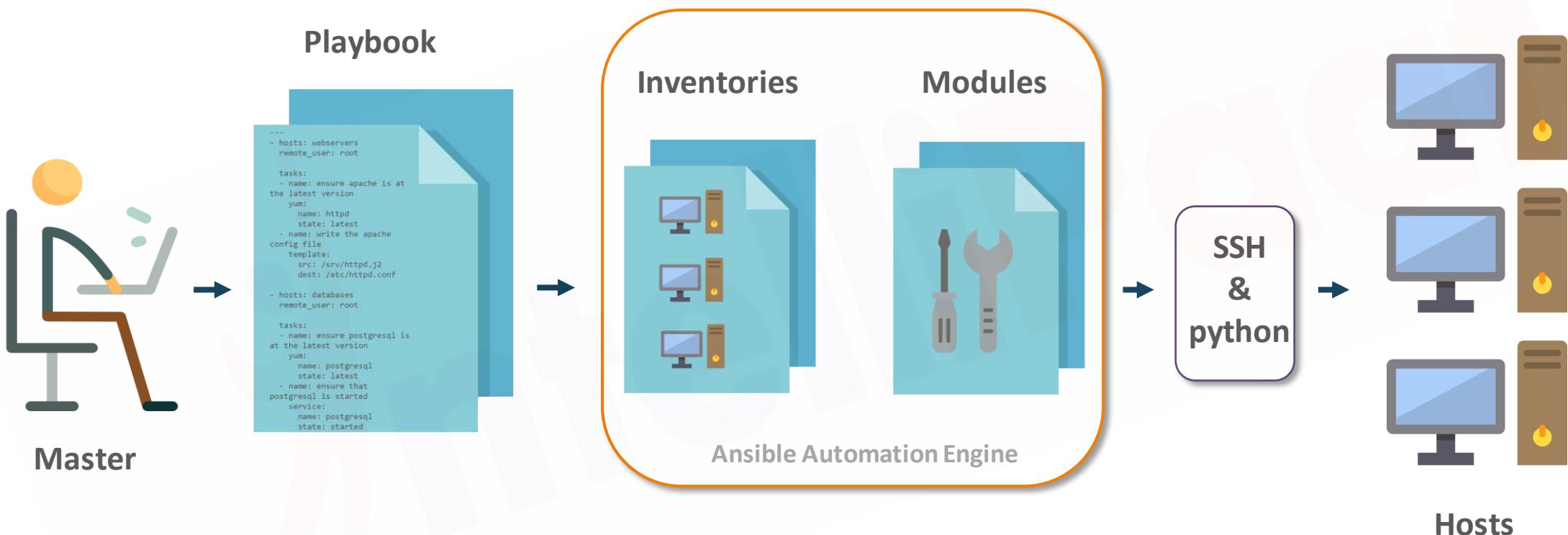
Configuration Management





Ansible Architecture

Ansible Architecture

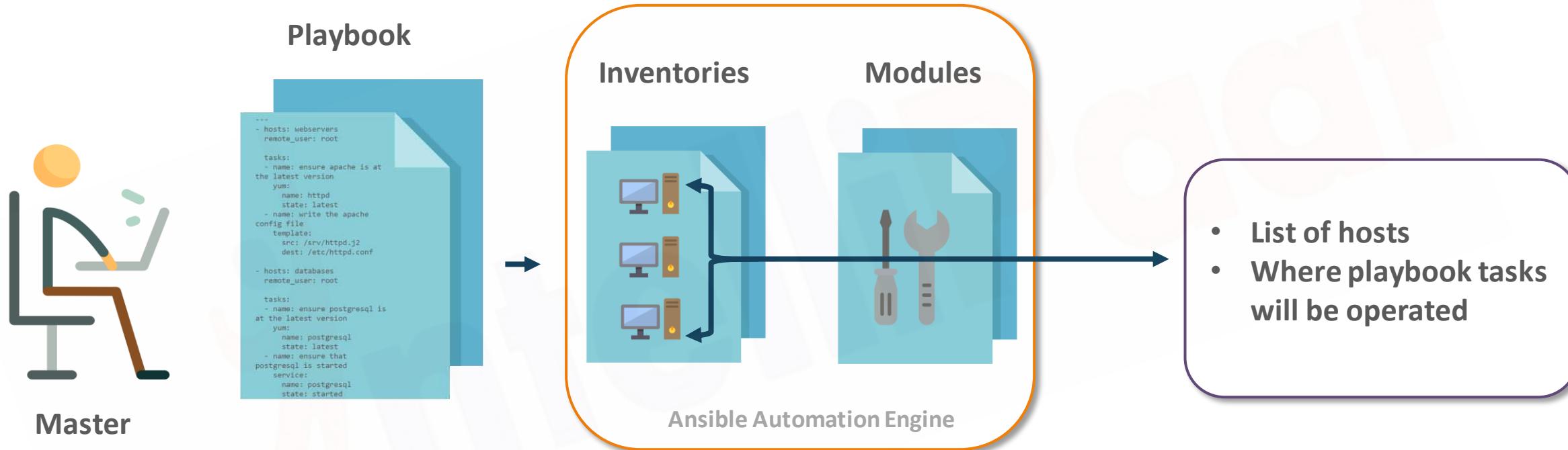


Basic Ansible Architecture

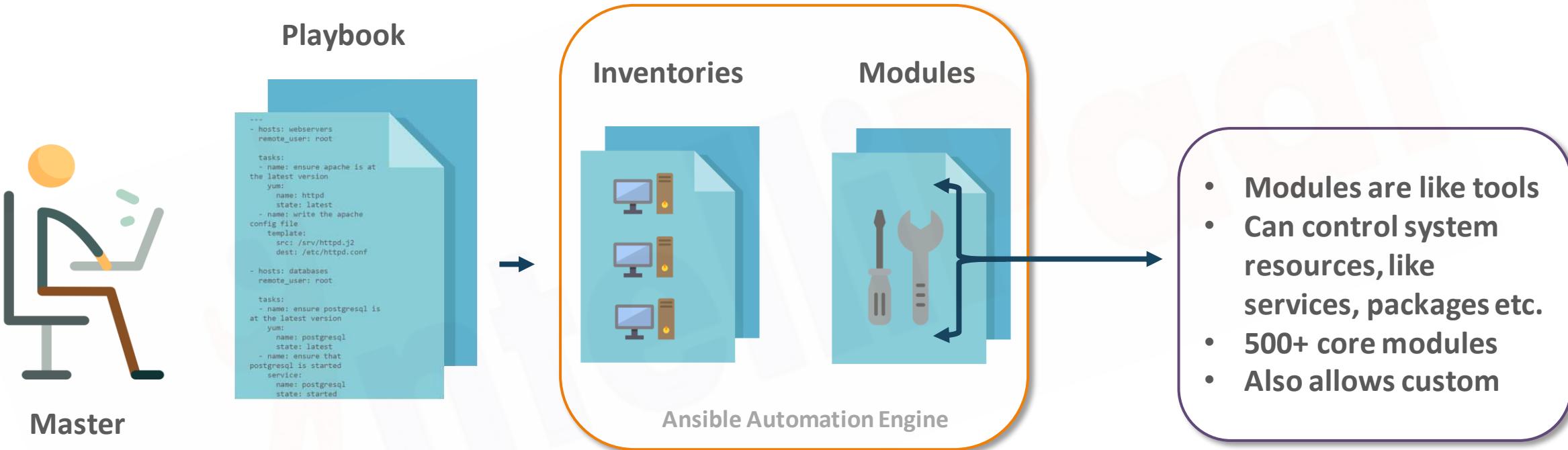
Ansible Architecture- Master



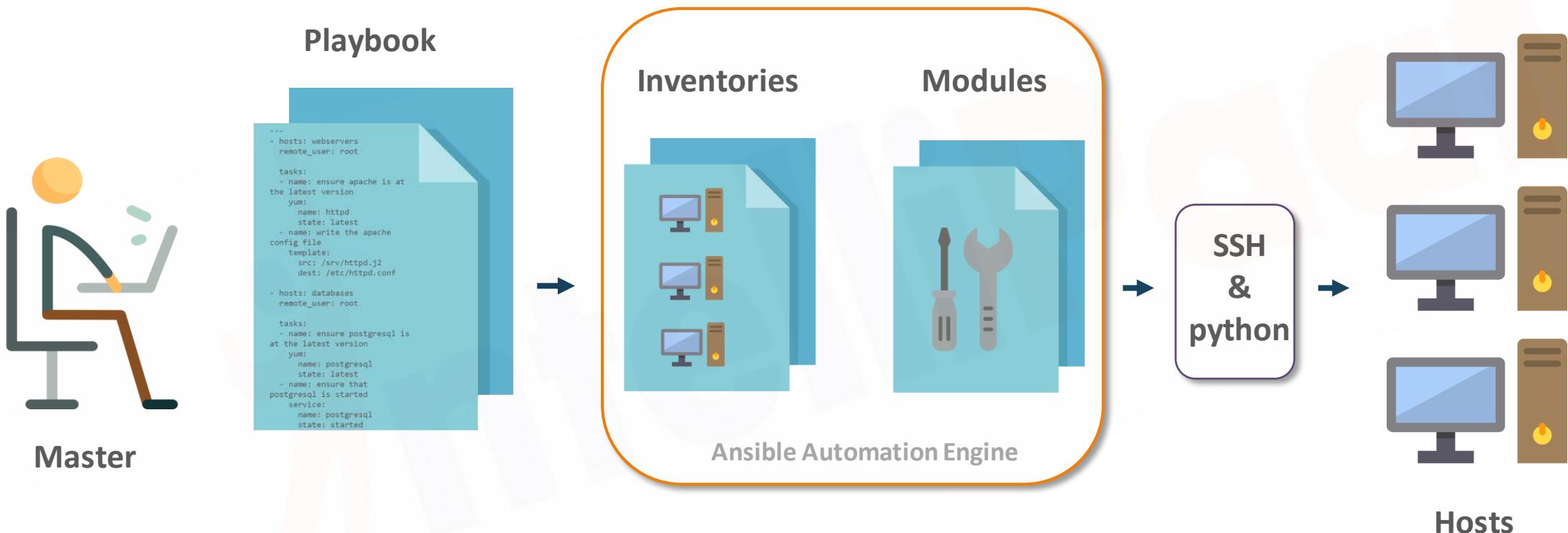
Ansible Architecture- Inventories



Ansible Architecture- Modules



Ansible Architecture- Hosts



Case Study: Ansible being used in NASA

Case Study- Business Challenge

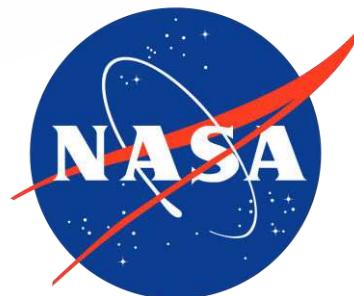
NASA needed to move roughly 65+ applications from a Traditional Hardware Based Data Center to Cloud Based Environment for better agility and cost saving



Traditional Hardware Based Data Center



Cloud Based Environment



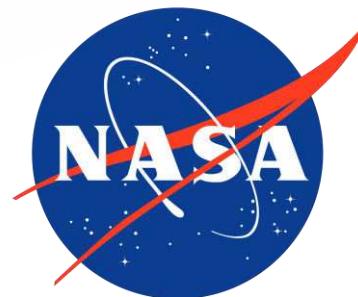
Case Study- Solution

NASA used Ansible to manage and schedule the cloud environment



Traditional Hardware Based Data Center

Cloud Based Environment



Case Study- Results

- ✓ Could provide better operations and security to its clients
- ✓ Increased team efficiency
- ✓ Patching updates went from a multi-day process to 45 minutes



Traditional Hardware Based Data Center



Cloud Based Environment





Installing Ansible on AWS

Installing Ansible on AWS



1

Install Ansible on Master

2

Configure SSH access to Ansible Host

3

Setting up Ansible Host and testing connection



Creating Ansible Playbooks

What is Ansible Playbook?

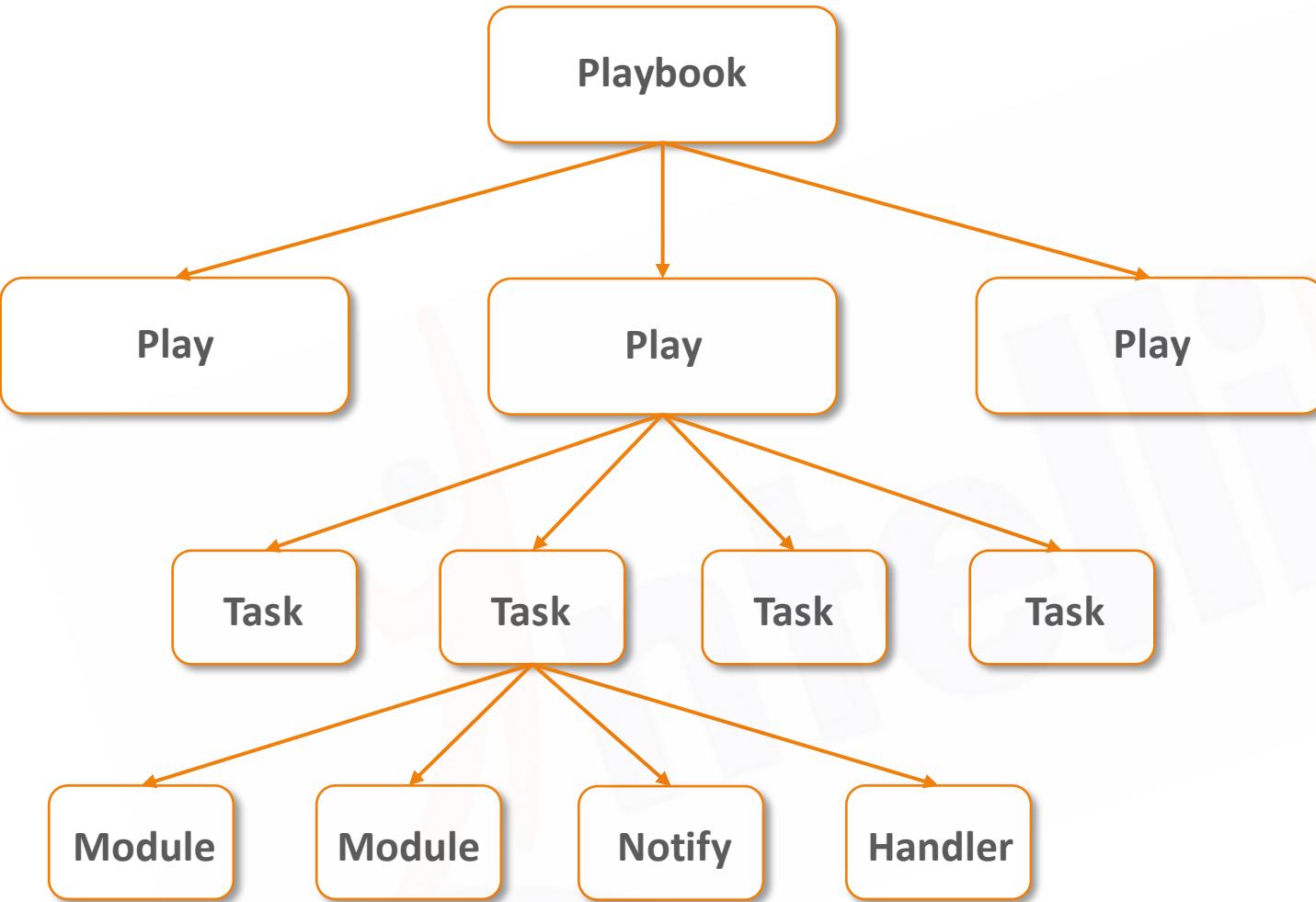
An organized unit of scripts
Defines work for a server configuration
Written in **YAML**

Ansible Playbook



YAML Ain't Markup Language

Ansible Playbook Structure



- ★ Playbook have number of **plays**
- ★ Play contains **tasks**
- ★ Tasks calls core or custom **modules**
- ★ Handler gets triggered from **notify** and executed at the end only once.



Creating Ansible Playbook-Example

Say, we want to create a playbook with two plays with following tasks

1 Execute a command in host1

2 Execute a script in host1

3 Execute a script in host2

4 Install nginx in host2

Play1

Play2

Creating Ansible Playbook-Example



```
---  
- hosts: host1  
  sudo: yes  
  name: Play 1  
  tasks:  
    - name: Execute command 'Date'  
      command: date  
    - name: Execute script on server  
      script: test_script.sh  
  
- hosts: host2  
  name: Play 2  
  sudo: yes  
  tasks:  
    - name: Execute script on server  
      script: test_script.sh  
    - name: Install nginx  
      apt: name=nginx state=latest
```

Say we want to create a playbook with two plays with following tasks

1 Execute a command in host1

2 Execute a script in host1

3 Execute a script in host2

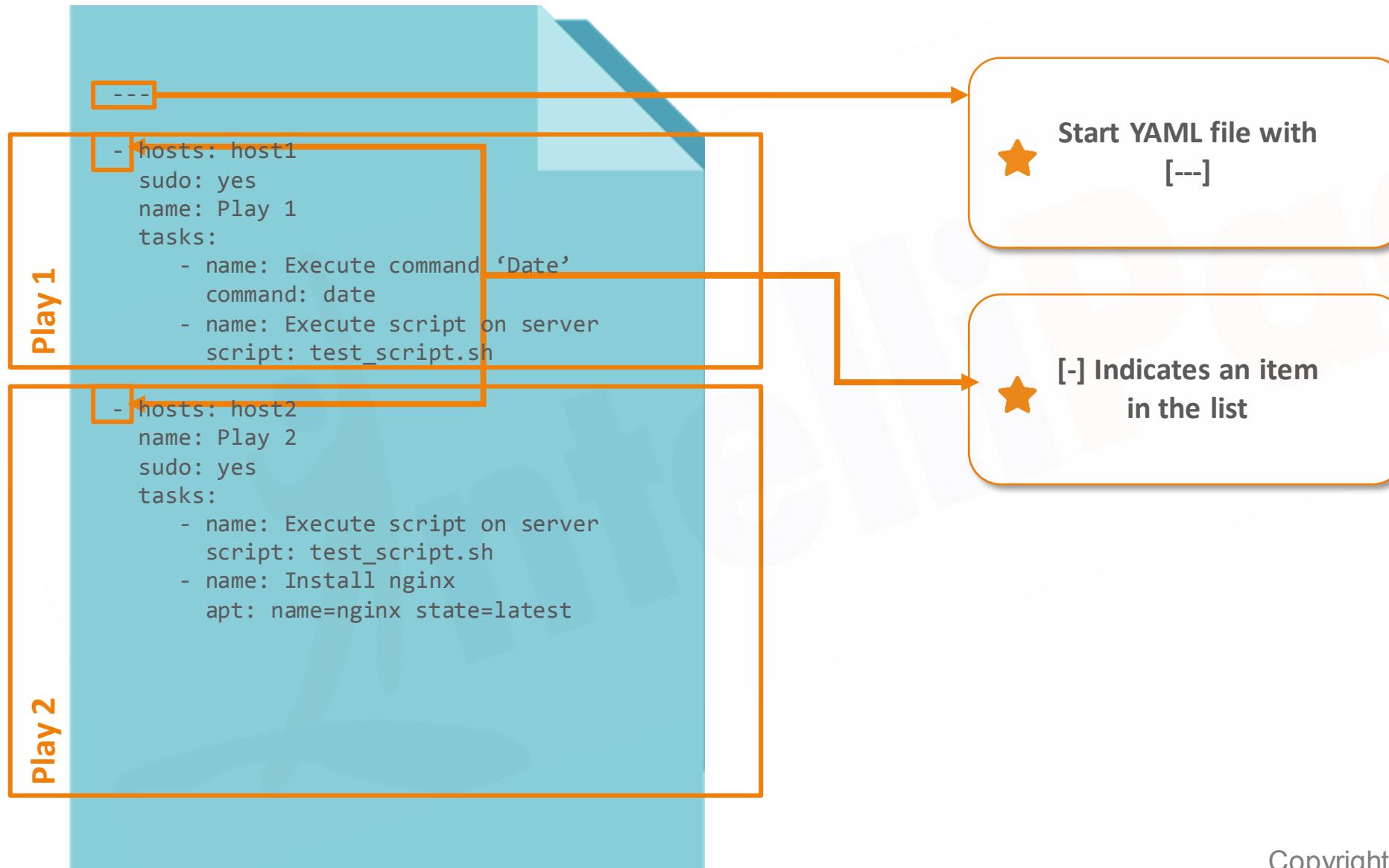
4 Install nginx in host2

Creating Ansible Playbook-Example

```
Play 1
---  
- hosts: host1  
  sudo: yes  
  name: Play 1  
  tasks:  
    - name: Execute command 'Date'  
      command: date  
    - name: Execute script on server  
      script: test_script.sh  
  
Play 2
---  
- hosts: host2  
  name: Play 2  
  sudo: yes  
  tasks:  
    - name: Execute script on server  
      script: test_script.sh  
    - name: Install nginx  
      apt: name=nginx state=latest
```

★ Start YAML file with
[---]

Creating Ansible Playbook-Example



Creating Ansible Playbook-Example

```
Play 1
---  
- hosts: host1  
  sudo: yes  
  name: Play 1  
  tasks:  
    - name: Execute command 'Date'  
      command: date  
    - name: Execute script on server  
      script: test_script.sh  
  
Play 2
- hosts: host2  
  name: Play 2  
  sudo: yes  
  tasks:  
    - name: Execute script on server  
      script: test_script.sh  
    - name: Install nginx  
      apt: name=nginx state=latest
```

★ “hosts” can have one host or group of hosts from the inventory file /etc/ansible/hosts

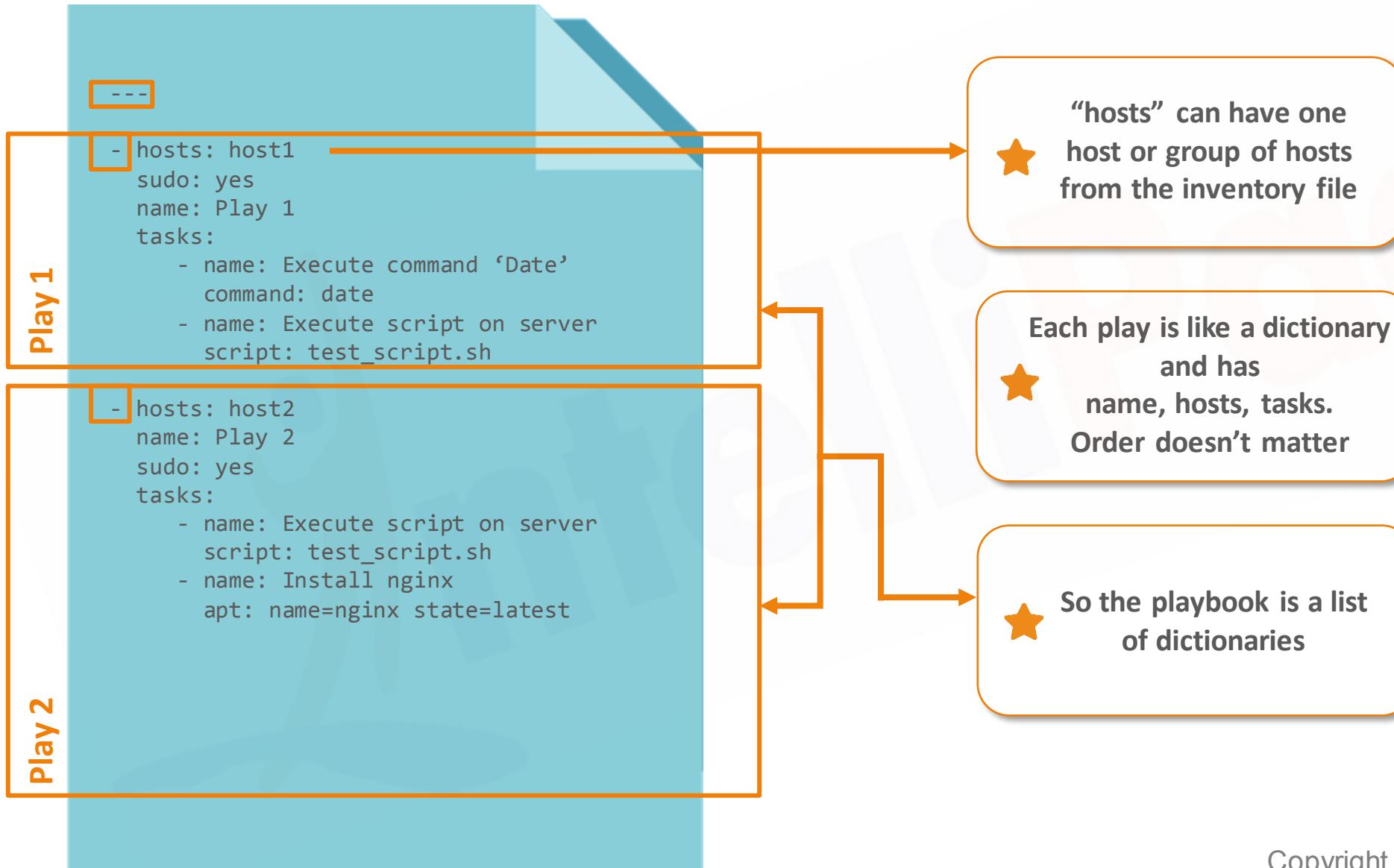
Creating Ansible Playbook-Example

```
Play 1
---  
- hosts: host1  
  sudo: yes  
  name: Play 1  
  tasks:  
    - name: Execute command 'Date'  
      command: date  
    - name: Execute script on server  
      script: test_script.sh  
  
Play 2
---  
- hosts: host2  
  name: Play 2  
  sudo: yes  
  tasks:  
    - name: Execute script on server  
      script: test_script.sh  
    - name: Install nginx  
      apt: name=nginx state=latest
```

★ “hosts” can have one host or group of hosts from the inventory file

★ Each play is like a dictionary and has name, hosts, tasks. Order doesn't matter

Creating Ansible Playbook-Example



Creating Ansible Playbook-Example

```
Play 1
---  
- hosts: host1  
  sudo: yes  
  name: Play 1  
  tasks:  
    - name: Execute command 'Date'  
    - command: date  
    - name: Execute script on server  
      script: test_script.sh

Play 2
- hosts: host2  
  name: Play 2  
  sudo: yes  
  tasks:  
    - name: Execute script on server  
      script: test_script.sh  
    - name: Install nginx  
      apt: name=nginx state=latest
```

Similarly tasks are nothing but lists Denoted by [-]

For tasks ordered collection.
Position of entry matters

First entry gets performed first

Creating Ansible Playbook-Example

Create first_playbook.yml using
sudo nano <playbookname>

```
ubuntu@ip-172-31-40-83: ~
ubuntu@ip-172-31-40-83:~$ sudo nano first_playbook.yml
```

```
ubuntu@ip-172-31-40-83: ~
GNU nano 2.9.3                                     first playbook.yml

--


- hosts: host1
  sudo: yes
  name: Play 1
  tasks:
    - name: Execute command 'Date'
      command: date
    - name: Execute script on server
      script: test_script.sh


- hosts: host2
  name: Play 2
  sudo: yes
  tasks:
    - name: Execute script on server
      script: test_script.sh
    - name: ensure nginx is at the latest version
      apt: name=nginx state=latest
```

Creating Ansible Playbook-Example

Create test_script.sh using
sudo nano <file_name>

```
ubuntu@ip-172-31-40-83: ~
ubuntu@ip-172-31-40-83:~$ sudo nano test_script.sh
```

```
ubuntu@ip-172-31-40-83: ~
GNU nano 2.9.3                                     test_script.sh

#!/bin/sh
# This is a comment!
echo Hello World      # This is a comment, too!
```

Creating Ansible Playbook-Example

Syntax-check and execute ansible playbook using
ansible-playbook <playbook> --syntax-check and
ansible-playbook <playbook>

```
ubuntu@ip-172-31-40-83:~$ ansible-playbook first_playbook.yml --syntax-check
playbook: first_playbook.yml
```

```
ubuntu@ip-172-31-40-83:~$ sudo ansible-playbook first_playbook.yml
PLAY [Play 1] ****
TASK [Gathering Facts] ****
ok: [host1]

TASK [Execute command 'Date'] ****
changed: [host1]

TASK [Execute script on server] ****
changed: [host1]

PLAY [Play 2] ****
TASK [Gathering Facts] [****]
ok: [host1]
```



Ansible Roles

What is Ansible Roles?

An ansible role is group of tasks, files, and handlers stored in a standardized file structure.

Roles are small functionalities which can be used independently used but only within playbook

Ansible Playbook

Ansible playbook organizes tasks

Ansible Roles

Ansible roles organizes playbooks

Why do we need Ansible Roles?



- ★ Roles simplifies writing complex playbooks
- ★ Roles allows you to reuse common configuration steps between different types of servers
- ★ Roles are flexible and can be easily modified

Structure of Ansible Role

```
new_role
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Structure of an Ansible Role

Structure of an ansible role consists of below given components

Defaults: Store data about the role, also store default variables.

Files: Store files that needs to be pushed to the remote machine.

Handlers: Tasks that get triggered from some actions.

Meta: Information about author, supported platforms and dependencies.

Structure of Ansible Role

```
new_role
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Structure of an Ansible Role

Structure of an ansible role consists of below given components

Tasks: Contains the main list of tasks to be executed by the role.

Templates: Contains templates which can be deployed via this role.

Handlers: Tasks that get triggered from some actions.

Vars: Stores variables with higher priority than default variables.
Difficult to override.

Creating an Ansible Role

1

Use the *ansible-galaxy init <role name> --offline* command to create one Ansible role



Remember that Ansible roles should be written inside */etc/ansible/roles/*

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles
ubuntu@ip-172-31-40-83:~$ cd /etc/ansible/roles/
ubuntu@ip-172-31-40-83:/etc/ansible/roles$ ansible-galaxy init apache --offline
```

Creating an Ansible Role

2

Install tree package using *sudo apt install tree*. Use *tree* command to view structure of the role



Use *tree <role name>* to see the role structure

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles
ubuntu@ip-172-31-40-83:/etc/ansible/roles$ sudo apt install tree
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 154 not upgraded.
```

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles
ubuntu@ip-172-31-40-83:/etc/ansible/roles$ tree apache
apache
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   └── inventory
│       └── test.yml
└── vars
    └── main.yml
```

Creating an Ansible Role

3

Go inside task folder inside apache directory. Edit **main.yml** using *sudo nano main.yml*. Make changes as shown. Save and then exit.



Keeping install, configure and service files separately helps us reduce complexity.

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
ubuntu@ip-172-31-40-83:/etc/ansible/roles/apache/tasks$ sudo nano main.yml
```

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
GNU nano 2.9.3                                     main.yml

---
# tasks file for apache
- include: install.yml
- include: configure.yml
- include: service.yml
```

Creating an Ansible Role

4

Create `install.yml`, `configure.yml` and `service.yml` to include in the `main.yml`



To install apache2 in the remote machine

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
ubuntu@ip-172-31-40-83:/etc/ansible/roles/apache/tasks$ sudo nano install.yml
```

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
GNU nano 2.9.3                                     install.yml

---
- name: install apache2
  apt: name=apache2 update_cache=yes state=latest
```

Creating an Ansible Role

4

Create **install.yml**, **configure.yml** and **service.yml** to include in the **main.yml**



To configure the apache2.conf file and to send copy.html file to the remote machine. Add notify too, based on which handlers will get triggered

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
ubuntu@ip-172-31-40-83:/etc/ansible/roles/apache/tasks$ sudo nano configure.yml
```

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
GNU nano 2.9.3                                     configure.yml

---
#configure apache2.conf and send copy.html file
- name: apache2.conf file
  copy: src=apache2.conf dest=/etc/apache2/
  notify:
    - restart apache2 service

- name: send copy.html file
  copy: src=copy.html dest=/home/ubuntu/
```

Creating an Ansible Role

4

Create `install.yml`, `configure.yml` and `service.yml` to include in the `main.yml`



To start apache2 service in the remote machine

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
ubuntu@ip-172-31-40-83:/etc/ansible/roles/apache/tasks$ sudo nano service.yml
```

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
GNU nano 2.9.3                                     service.yml

---
- name: starting apache2 service
  service: name=apache2 state=started
```

Creating an Ansible Role

5

Now go inside files. Store the files that needs to be pushed to the remote machine



Copy the apache2.conf file and create one html file

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/files
ubuntu@ip-172-31-40-83:/etc/ansible/roles/apache/files$ ls
apache2.conf  copy.html
```

Creating an Ansible Role

6

Go inside handlers and add the action that needs to be performed after notify from configure.yml is executed.



Once the notify gets executed restart the apache2 service

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/handlers
ubuntu@ip-172-31-40-83:/etc/ansible/roles/apache/handlers$ sudo nano main.yml

ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/handlers
GNU nano 2.9.3                                     main.yml

---
# handlers file for apache
- name: restart apache2 service
  service: name=apache2 state=restarted
```

Creating an Ansible Role



Remember that notify name and handler name should match.

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/tasks
GNU nano 2.9.3                                         configure.yml

---
#configure apache2.conf and send copy.html file
- name: apache2.conf file
  copy: src=apache2.conf dest=/etc/apache2/
  notify:
    - restart apache2 service

- name: send copy.html file
  copy: src=copy.html dest=/home/ubuntu/
```

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/handlers
GNU nano 2.9.3                                         main.yml

---
# handlers file for apache
- name: restart apache2 service
  service: name=apache2 state=restarted
```

IMPORTANT

Creating an Ansible Role

7

Go inside meta and add information related to the role



Add author information, role descriptions, company information etc.

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/meta
ubuntu@ip-172-31-40-83:/etc/ansible/roles/apache/meta$ sudo nano main.yml
```

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles/apache/meta
GNU nano 2.9.3                                     main.yml

galaxy_info:
author: Intellipaat
description: Simple apache role
company: Intellipaat

# If the issue tracker for your role is not on github, uncomment the
# next line and provide a value
# issue_tracker_url: http://example.com/issue/tracker
```

Creating an Ansible Role



Structure of the role after adding all the required files

```
ubuntu@ip-172-31-40-83:/etc/ansible/roles$ tree apache
apache
├── README.md
├── defaults
│   └── main.yml
├── files
│   ├── apache2.conf
│   └── copy.html
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   ├── configure.yml
│   ├── install.yml
│   ├── main.yml
│   └── service.yml
├── templates
└── tests
    └── inventory
        └── test.yml
vars
└── main.yml
```

Creating an Ansible Role

8

Go to the `/etc/ansible/` and create one top level file where we can add hosts and roles to be executed



Execute *apache role* on the hosts that is under the group name *servers*, added in the inventory file `/etc/ansible/hosts`

```
ubuntu@ip-172-31-40-83: /etc/ansible/roles
ubuntu@ip-172-31-40-83:/etc/ansible$ sudo nano site.yml
```

```
ubuntu@ip-172-31-40-83: /etc/ansible
GNU nano 2.9.3                                         site.yml
---
- hosts: servers
  roles:
    - apache
```

Creating an Ansible Role

9

Before we execute our top level yml file we will check for syntax errors.



Use ansible-playbook <filename.yml>--syntax-check

```
ubuntu@ip-172-31-40-83: /etc/ansible
ubuntu@ip-172-31-40-83:/etc/ansible$ ansible-playbook site.yml --syntax-check
playbook: site.yml
```

Creating an Ansible Role

10

Execute the top level yml file



Use ansible-playbook<filename.yml>

```
ubuntu@ip-172-31-40-83: /etc/ansible
ubuntu@ip-172-31-40-83:/etc/ansible$ ansible-playbook site.yml
```

```
PLAY [servers] *****
TASK [Gathering Facts] *****
ok: [host1]
ok: [host2]

TASK [apache : install apache2] *****
ok: [host1]
ok: [host2]

TASK [apache : apache2.conf file] *****
ok: [host1]
ok: [host2]

TASK [apache : send copy.html file] *****
ok: [host1]
ok: [host2]

TASK [apache : starting apache2 service] *****
ok: [host1]
ok: [host2]

PLAY RECAP *****
host1                  : ok=5      changed=0      unreachable=0      failed=0
host2                  : ok=5      changed=0      unreachable=0      failed=0
```



Using Roles in Playbook

Using Roles in Playbook



To use ansible roles along with other tasks in playbook
Use *import_role* and *include_role*.



Here we have created one playbook called *playbookrole.yml* to execute on servers along with two *debug* tasks before and after *apache* role.

```
ubuntu@ip-172-31-40-83: /etc/ansible
ubuntu@ip-172-31-40-83:/etc/ansible$ sudo nano playbookrole.yml

ubuntu@ip-172-31-40-83: /etc/ansible
GNU nano 2.9.3                                     playbookrole.yml

---
- hosts: servers
  sudo: yes
  tasks:
    - debug:
        msg: "before we run our role"
    - import_role:
        name: apache
    - include_role:
        name: apache
    - debug:
        msg: "after we ran our role"
```

Using Roles in Playbook



Check for syntax error and execute the playbook with roles.

```
ubuntu@ip-172-31-40-83: /etc/ansible
ubuntu@ip-172-31-40-83:/etc/ansible$ ansible-playbook playbookrole.yml --syntax-check
playbook: playbookrole.yml
```

```
ubuntu@ip-172-31-40-83: /etc/ansible
ubuntu@ip-172-31-40-83:/etc/ansible$ ansible-playbook playbookrole.yml
PLAY [servers] ****
TASK [Gathering Facts] ****
ok: [host1]
ok: [host2]

TASK [debug] ****
ok: [host1] => {
    "msg": "before we run our role"
}
ok: [host2] => {
    "msg": "before we run our role"
}

TASK [apache : install apache2] ****
ok: [host1]
ok: [host2]

TASK [apache : apache2.conf file] ****
ok: [host1]
ok: [host2]

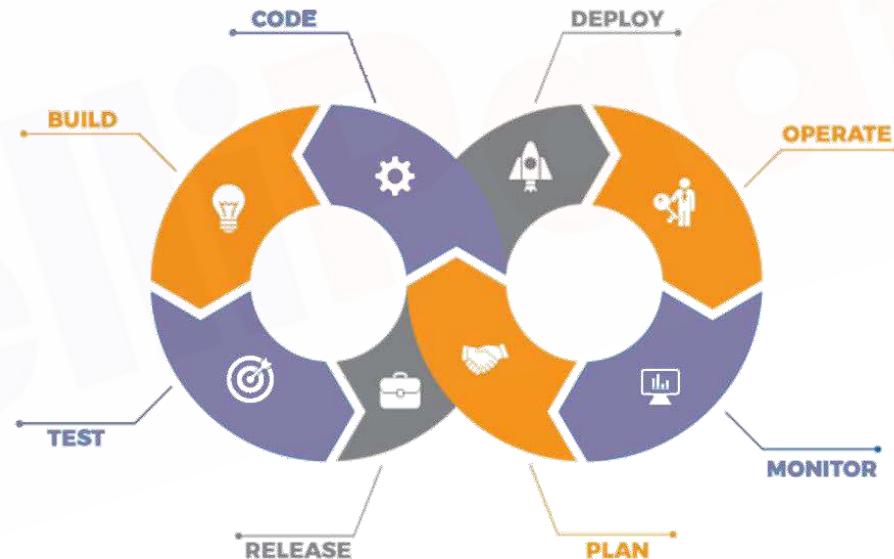
TASK [apache : send copy.html file] ****
ok: [host1]
ok: [host2]

TASK [apache : starting apache2 service] ****
ok: [host1]
ok: [host2]
```



Hands-on: Configuring Multiple Nodes using Ansible

Introduction to Selenium



Agenda

01

Introduction to
Software Testing

02

Introduction to
Selenium

03

What is
Maven?

04

Creating
Automated Tests

05

Introduction to
TestNG

06

Introduction to
Continuous Testing

Introduction to Software Testing

Introduction to Software Testing

Software testing is defined as an activity to check whether the actual results match with the expected results and to ensure that the software system is defect free.



Types of Software Testing

Types of Software Testing

Automated Testing

- ★ Test cases are executed automatically
- ★ More accurate
- ★ More suitable when test cases are run repeatedly
- ★ Suitable for scenarios when testing is functionality based

Manual Testing

- ★ Tests cases are executed manually
- ★ Less accurate
- ★ More suitable when test cases are supposed to run only once or twice
- ★ More suitable when testing is for user experience

Automated Testing Tools

Types of Testing



Selenium



Appium



Cucumber



Telerik
Test Studio

Test Studio

Introduction to Selenium

What is Selenium?

Selenium is a portable framework for testing web applications. Selenium provides a playback tool for authoring functional tests without the need to learn a new test scripting language.



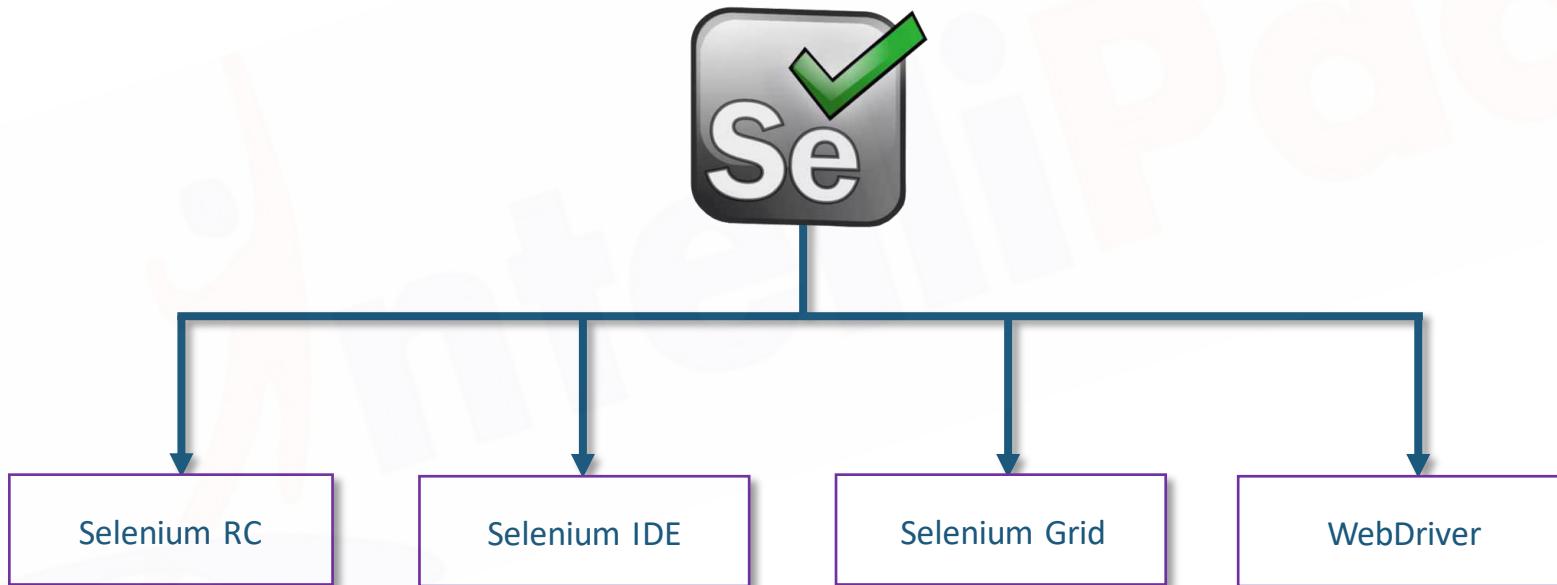
What is Selenium?

- Selenium was first created by **Jason Huggins** in **2004**.
- He was working on a web application which had to be frequently tested.
- Since manual testing was taking a lot of time, he wrote a JavaScript program.
- This program could automatically control the browser's actions.
- He named it as ***JavaScriptTestRunner*** and donated it to the open-source community.
- It was later named as **Selenium Core**.



Components of Selenium

But only the Selenium Core could not suffice all the use cases of testing. Hence, a suite of Selenium components was developed for different purposes.



Components of Selenium

Selenium RC

- Due to same origin policy, testers had to install Selenium Core and the web server on their local system.
- This was done to keep the domain same for the Selenium Core and the web application to be tested.
- Selenium RC is a web server, which acts as an HTTP proxy.
- It tricks the OS into believing both Selenium Core and the website to be tested are on the same domain.
- This system was also known as Selenium 1.

Selenium Grid

WebDriver



Components of Selenium

Selenium RC

Selenium IDE

Selenium Grid

WebDriver

- Selenium IDE was originally created as a Firefox extension.
- It could automate tests using the record and playback feature.
- The intention behind creating this component was to increase the speed of creating test cases in Selenium.
- It was created by Shinya Kasatani from Japan.



Components of Selenium

Selenium RC

Selenium IDE

Selenium Grid

WebDriver

- Selenium Grid enables parallel testing of applications on multiple machines.
- It was primarily created to minimize the time taken in executing test cases.
- It can be used across multiple browsers and OS.
- It can also be used to break down a huge test suite among many computers testing the same application.



Components of Selenium

Selenium RC

Selenium IDE

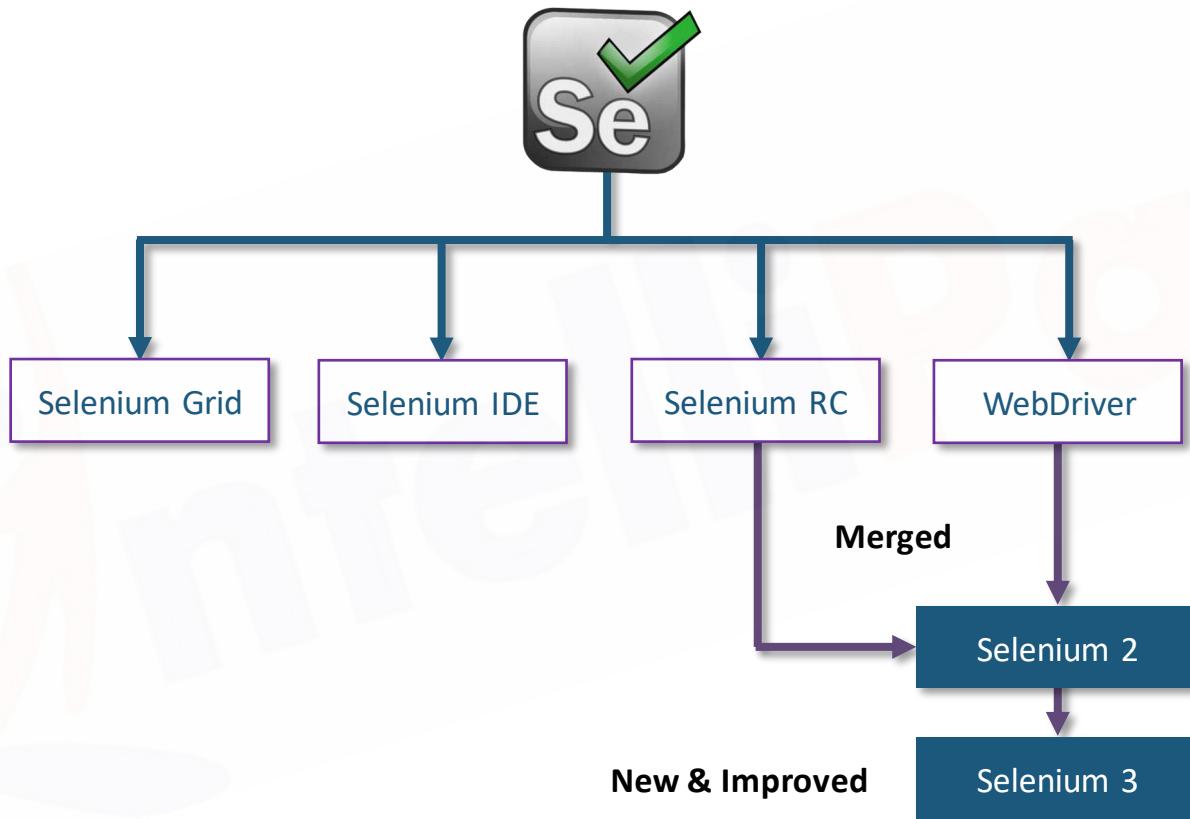
Selenium Grid

WebDriver

- Selenium WebDriver was the first cross-platform testing framework that could control the browser from the OS level.
- It was developed in 2006, when web applications and browsers were becoming more powerful and restrictive with JavaScript programs like Selenium Core.
- It was better than Selenium IDE and RC.
- It controls the browser by directly communicating with it.



History of Selenium



What is Maven?

What is Maven?

Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: first, it describes how a software is built and, second, it describes its dependencies.



Why do we need Maven?

- ★ Maven is used to download dependencies for a software program.
- ★ Dependencies to be downloaded are included inside a POM file.
- ★ Once the dependencies are added in the POM file, simply save the project and all the dependencies will automatically be downloaded.

The official Maven logo, where the letter 'a' is replaced by a colorful feather or leaf, with a trademark symbol at the end.



Hands-on: Setting up Selenium with Maven

Setting up Selenium with Maven



- Install Java
- Download and Install Eclipse for Developers
- Create a Maven Project
- Acquire Maven Dependencies for Selenium



Creating Automated Tests

Creating Automated Tests

There are three steps to executing a web test:

- Find the element on the web browser
- Perform an action on the found element(s)
- Test and create a test report with the results





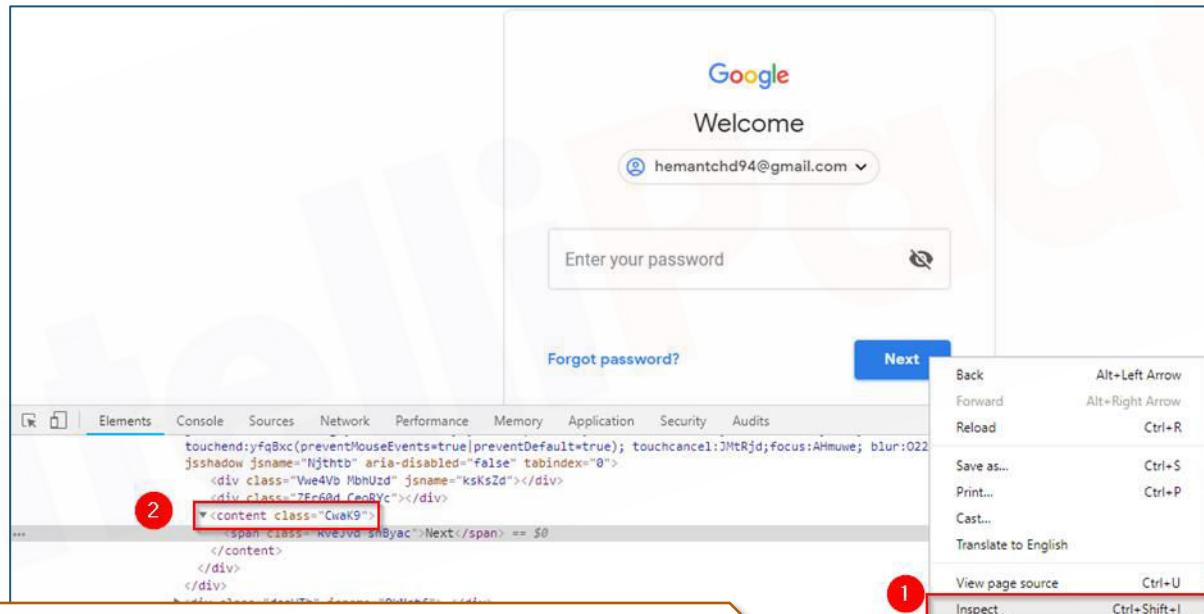
Finding Elements in Selenium

An element can be found on a web page using the following selectors:

- ID
- Name
- Class Name
- Tag Name
- Link Text
- Partial Link Text
- XPATH

Syntax

```
WebElement button = driver.findElement(By.class("CwaK9"));
```





Performing Action on Elements

The next step is taking an action. For that, one can try the following options:

- **Click():** Used to click on the link and wait for page load to complete before proceeding to the next command
- **sendKeys():** Used to enter values onto text boxes
- **Clear():** Used to clear text boxes of its current value
- **Submit():** WebDriver will automatically trigger the submit function of the form where that element belongs to

Syntax

```
WebElement button = driver.findElement(By.class("CwaK9")).click();
```





Testing & Reporting in Selenium Using TestNG

What is TestNG?

TestNG is an open-source automated testing framework, where **NG** means "Next Generation". It is a testing framework inspired from JUnit and NUnit, but introducing some new functionalities makes it more powerful and easier to use.

Test**NG**

Features of TestNG

- TestNG annotations are easy to create test cases.
- Test cases can be grouped and prioritized more easily.
- It supports parameterization.
- It supports data-driven testing using DataProviders.
- It generates HTML reports.
- Parallel test execution is possible.
- It readily supports integration with other tools and plug-ins like Eclipse IDE, build tools Ant, Maven etc.

Test**NG**

Hands-on: Setting up TestNG

Annotations in TestNG

Annotations in TestNG

Annotations in TestNG are used to decide the flow of the program. There are a lot of annotations in TestNG, we will focus on the most used ones and following are the same:

@BeforeSuite: The annotated method will be run only once before all tests in this suite have run.

@BeforeTest: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.

@BeforeClass: The annotated method will be run only once before the first test method in the current class is invoked.

@BeforeMethod: The annotated method will be run before each test method.

@Test: This marks a class or a method as a part of the test.

Hands-on: Working with Annotations in TestNG

Hands-on: Creating Automated Test with TestNG

Hands-on: Creating Our First Test Case



- Open the Intellipaat website, by visiting www.intellipaat.com
- Enter “DevOps” term and click search
- On the search page, check for “DevOps Certification Course”. If it exists, click on the course
- On the course page, verify if in the page “DevOps Certification Training” is present as the header

Hands-on: Running a Headless Test in Selenium

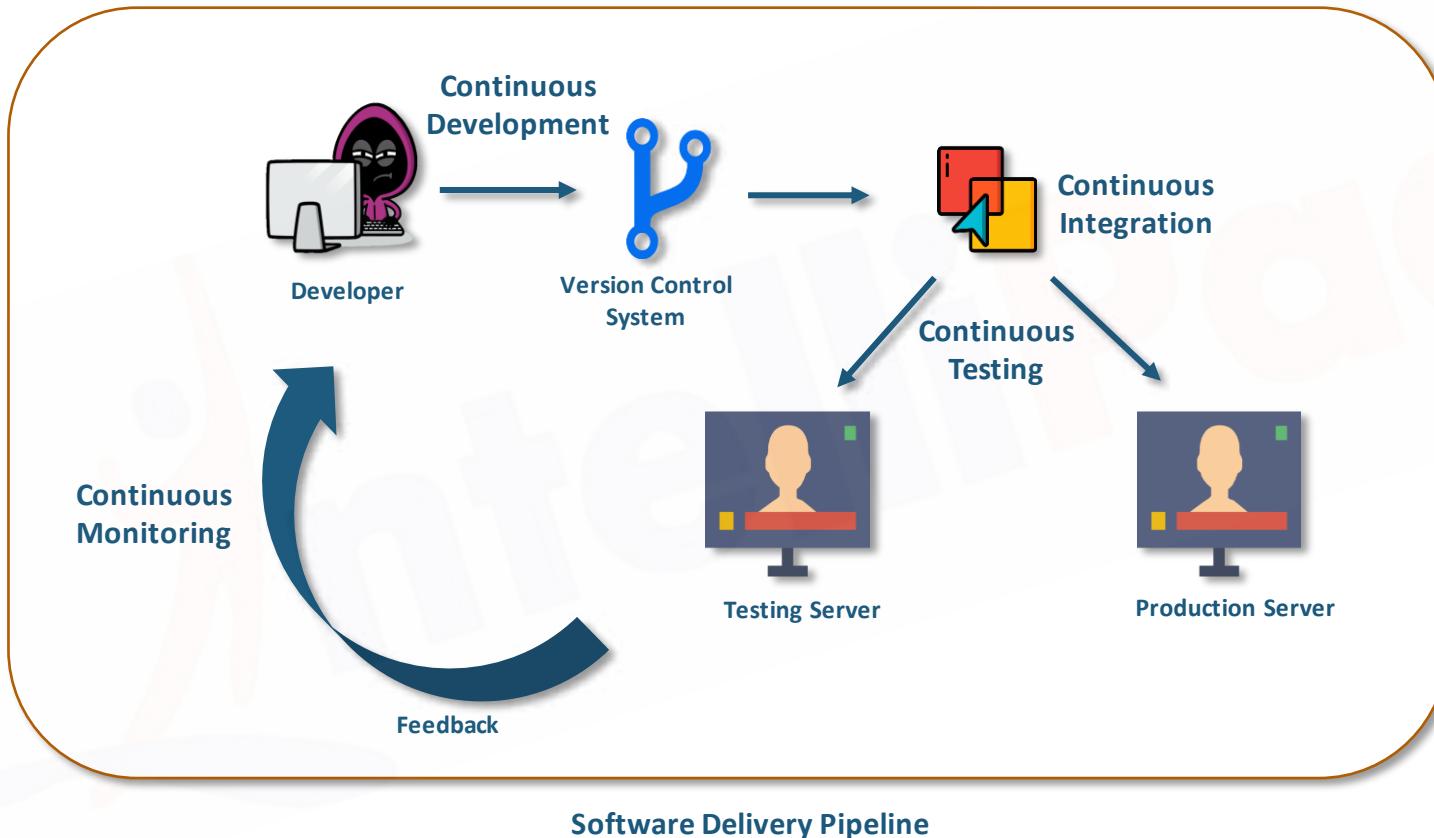
Introduction to Continuous Testing

Introduction to Continuous Testing

Continuous Testing is the process of executing **automated tests** as part of the software delivery pipeline in order to obtain feedback on the business risks associated with a software release candidate as rapidly as possible.



Introduction to Continuous Testing



Introduction to Continuous Testing



Products are built with their respective test suites while the features are being developed.



Production Server



Testing Server

Product Features

Feature A

Test Suite

Test suite for Feature A

Introduction to Continuous Testing



Products are built with their respective test suites while the features are being developed.



Production Server



Testing Server

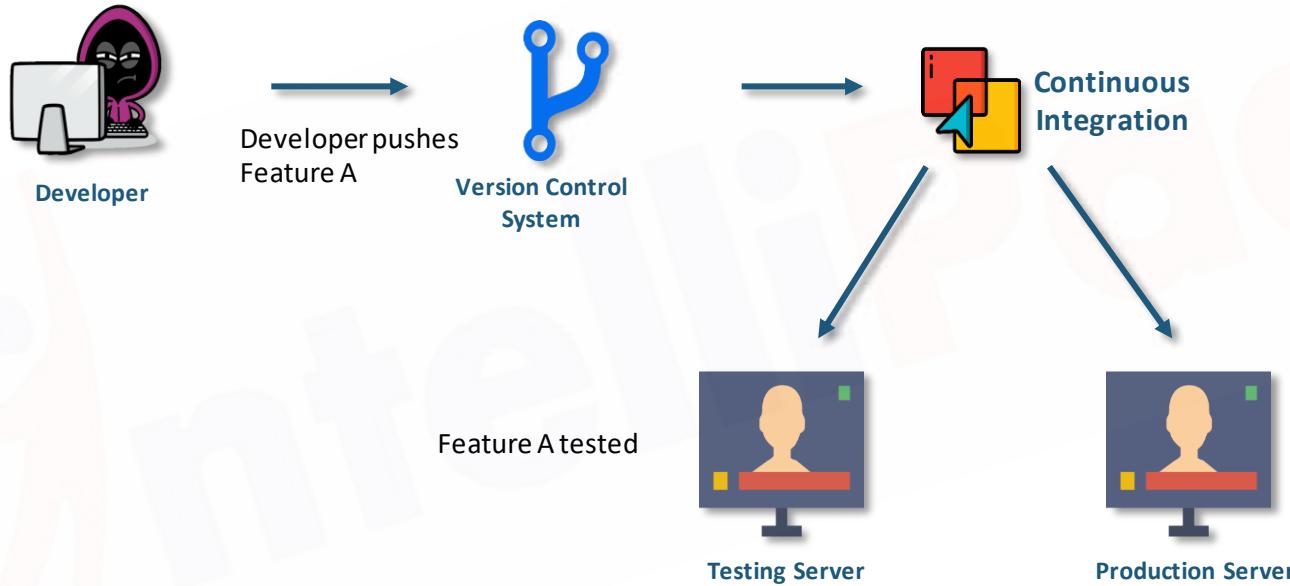
Product Features

Feature A
Feature B

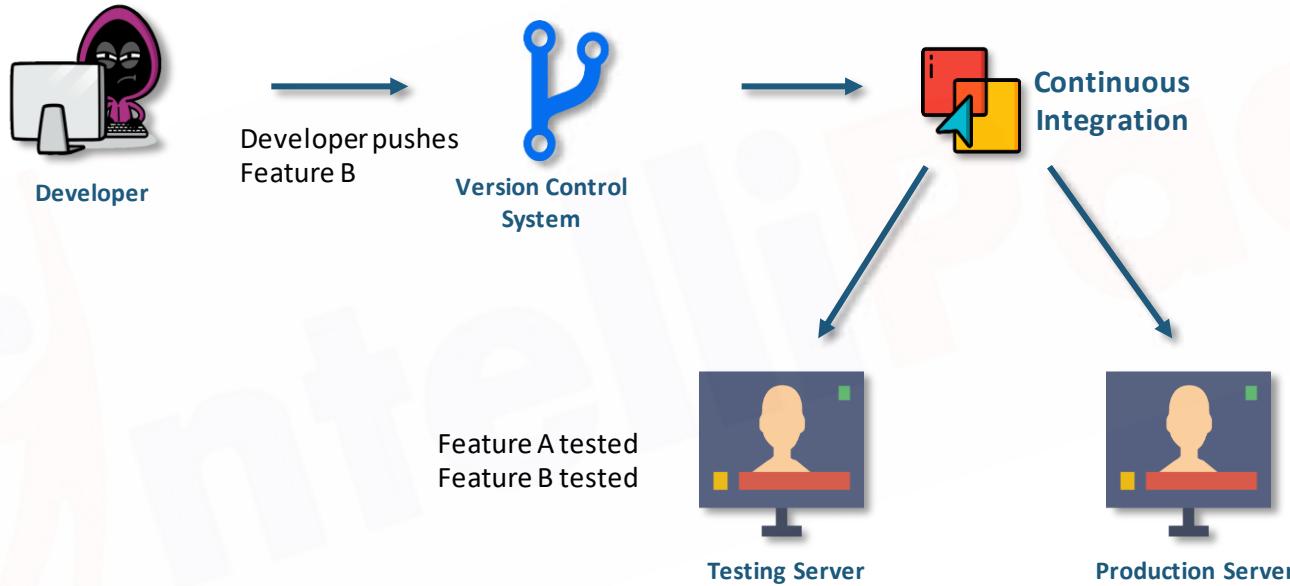
Test Suite

Test suite for Feature A
Test suite for Feature B

Introduction to Continuous Testing



Introduction to Continuous Testing



Quiz

1. Can Selenium automate Desktop Applications?

- A. No
- B. Yes
- C. Only Microsoft Applications
- D. None of these

Quiz

1. Can Selenium automate Desktop Applications?

A. No

B. Yes

C. Only Microsoft Applications

D. None of these

2. What does Maven do?

- A. Creates Test Reports
- B. Dependency Management
- C. Helps in Testing Applications
- D. None of these

2. What does Maven do?

A. Creates Test Reports

B. Dependency Management

C. Helps in Testing Applications

D. None of these

3. Which component in Selenium helps in parallel test execution?

- A. Selenium RC
- B. Selenium Grid
- C. Selenium WebDriver
- D. None of these

3. Which component in Selenium helps in parallel test execution?

A. Selenium RC

B. Selenium Grid

C. Selenium WebDriver

D. None of these

4. Annotations in Selenium help us _____.

- A. Follow a sequence in Selenium code
- B. Comment code
- C. Create reports
- D. None of these

4. Annotations in Selenium help us _____.

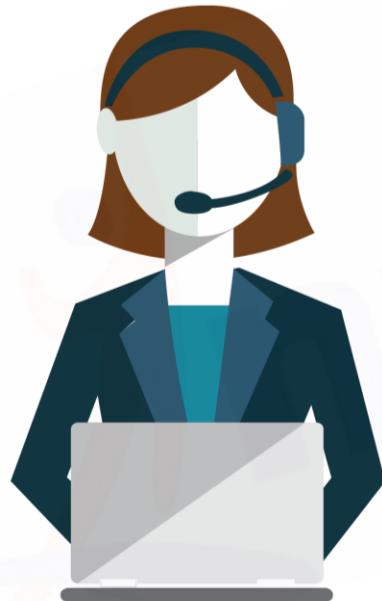
- A. Follow a sequence in Selenium code
- B. Comment code
- C. Create reports
- D. None of these

5. What is Headless Mode?

- A. Runs Selenium tests in Headless Mode
- B. Runs Selenium tests in foreground
- C. Runs Selenium tests in background
- D. None of these

5. What is Headless Mode?

- A. Runs Selenium tests in Headless Mode
- B. Runs Selenium tests in foreground
- C. Runs Selenium tests in background**
- D. None of these



India: +91-7847955955



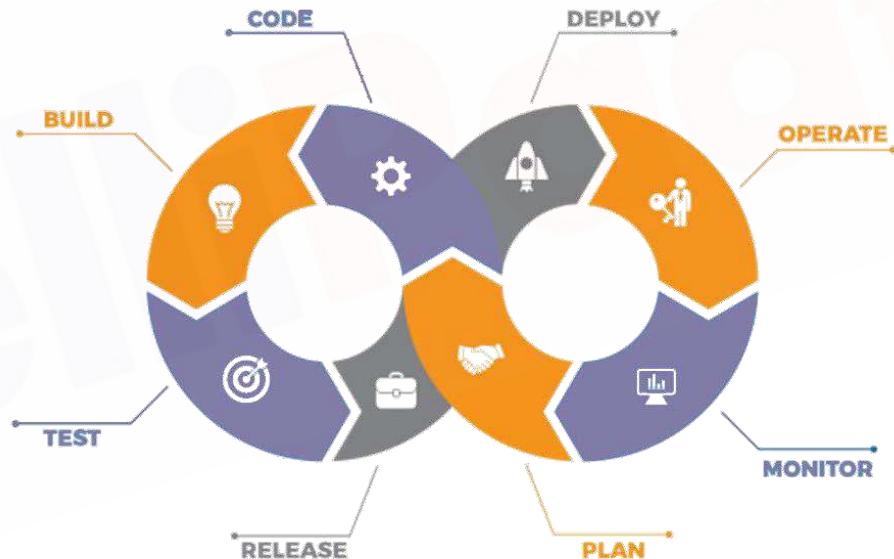
US: 1-800-216-8930 (TOLL FREE)



support@intellipaat.com

24/7 Chat with Our Course Advisor

Continuous Integration Using Jenkins



Agenda

01

Introduction to
Continuous Integration

02

What is
Jenkins?

03

Installing Jenkins
on AWS

04

Jenkins
Architecture

05

Managing Nodes
on Jenkins

06

Jenkins Integration
with DevOps Tools

07

Understanding
CI/CD Pipelines

08

Creating an End-to-end
Automated Pipeline in AWS

Why Continuous Integration?

Before Continuous Integration



Version 1



Developer 1



Version 1



Developer 2



Source Code
Management



Version 1

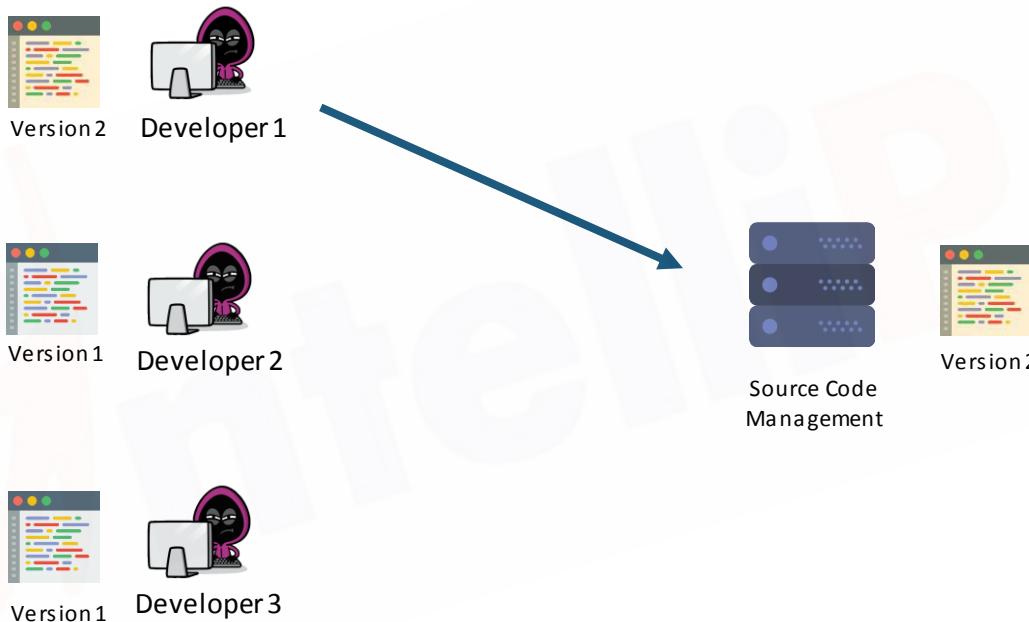


Version 1

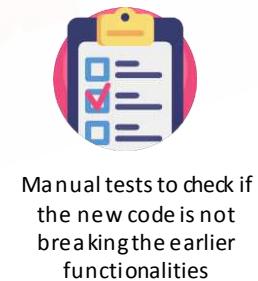


Developer 3

Before Continuous Integration



Before Continuous Integration



Problems before Continuous Integration

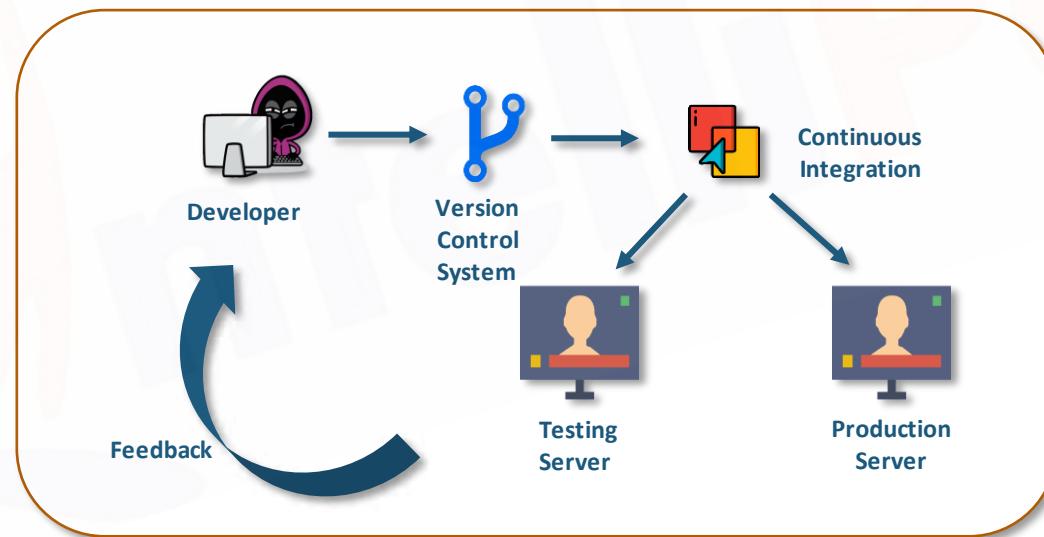


- ✖ Infrequent Commits led to bigger releases in one go leading to complex integration.
- ✖ Manual testing took a lot of time.
- ✖ Feedback took a lot of time to reach the developer.
- ✖ It involved high risk and uncertainty.

What is Continuous Integration?

What is Continuous Integration?

The process of having shorter release cycles (sometimes, several times a day), i.e., creating small features and integrating them to the source code and employing automated build and test processes for quicker feedback is called Continuous Integration.



Advantages of Continuous Integration



- ✓ Frequent Commits, hence small feature release
- ✓ Automated Build and Testing
- ✓ Instant feedback to the developer
- ✓ Low risk and faster delivery

What is Jenkins?

What is Jenkins?

Jenkins is an open-source automation server written in Java. Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery.



Features of Jenkins



Adoption: Jenkins is extremely popular among the open-source community; hence, there are more than 147,000 active installations throughout the world and 1 million people are using it.



Plugins Support: With an extremely active open-source community, Jenkins has around 1000 plugins that allow it to integrate with most of the development, testing and deployment tools.



Advantages of Jenkins

Before Jenkins

- ★ Locating and fixing bugs in the event of build and test failure was difficult and time consuming.
- ★ Tests were triggered manually.
- ★ No central place for triggering jobs on remote systems.

After Jenkins

- ★ Smaller and automated continuous build and testing make the task accurate and faster.
- ★ Developers have to just commit the code to the remote repository, build, test and deployment happen automatically.
- ★ All builds or tests on multiple remote systems can be controlled from one place.



Installing Jenkins on AWS

Installing Jenkins on AWS

1. Launch an AWS Instance
2. Connect through SSH
3. Execute the following commands:

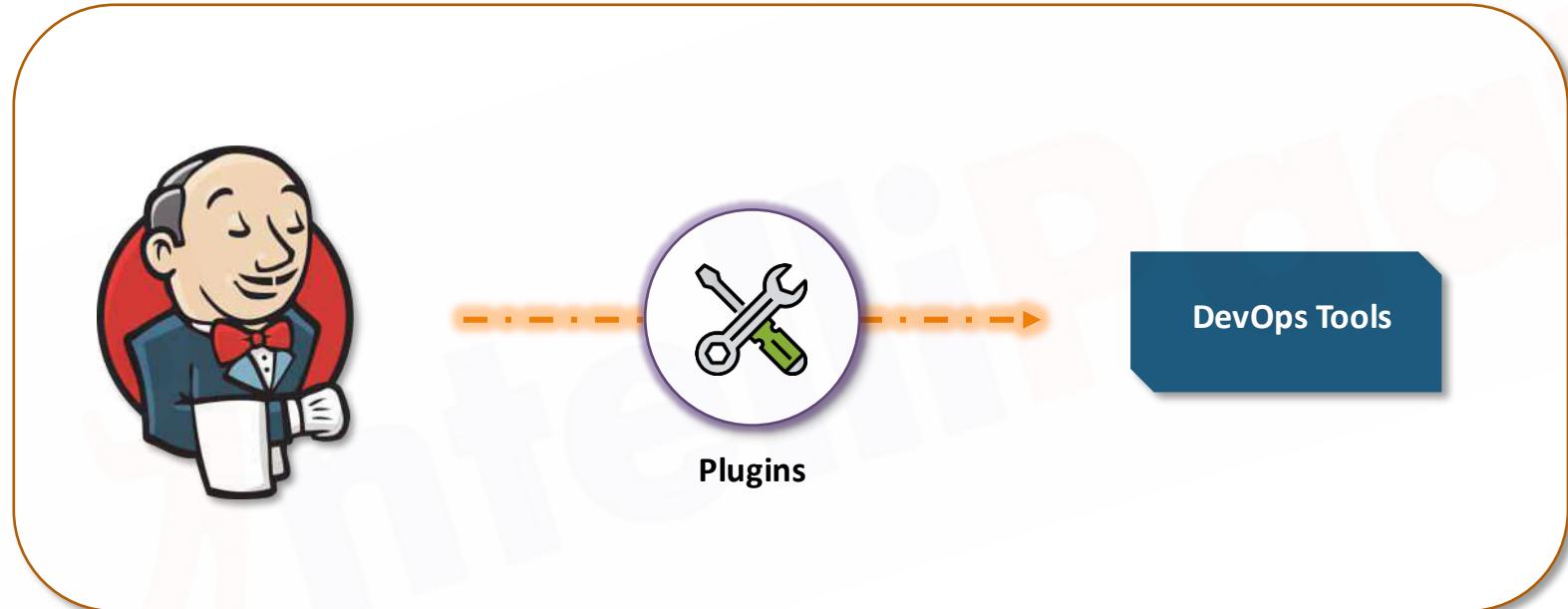
Jenkins Installation:

```
$> sudo apt-get update  
$> sudo apt install openjdk-8-jdk  
$> wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -  
$> sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'  
$> sudo apt update  
$> sudo apt install jenkins
```

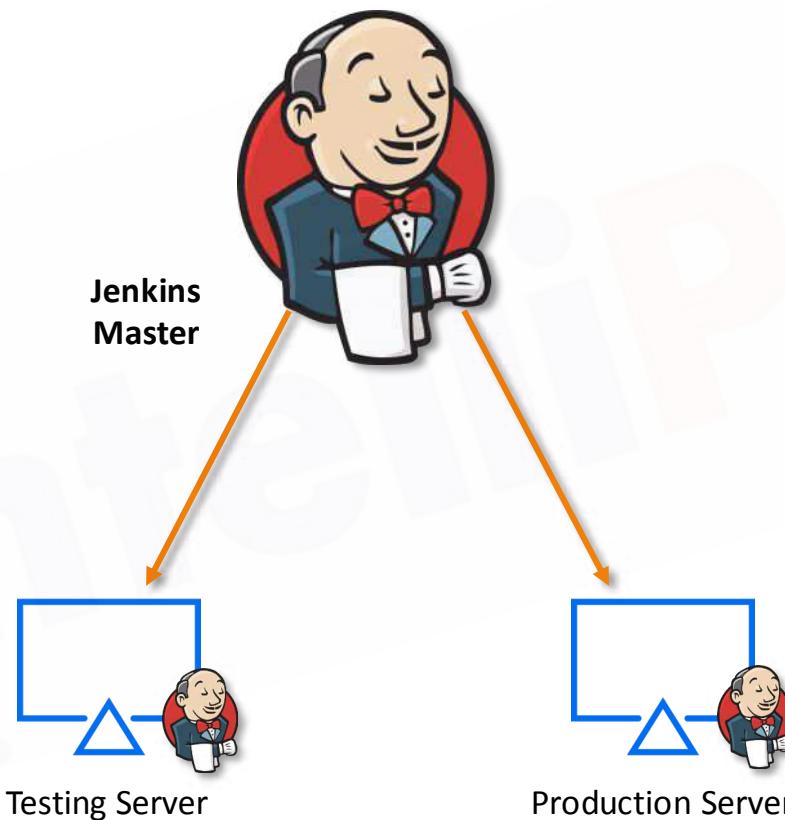


Jenkins Architecture

Jenkins Architecture



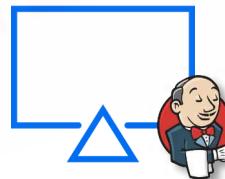
Jenkins Architecture



Managing Nodes on Jenkins

Managing Nodes on Jenkins

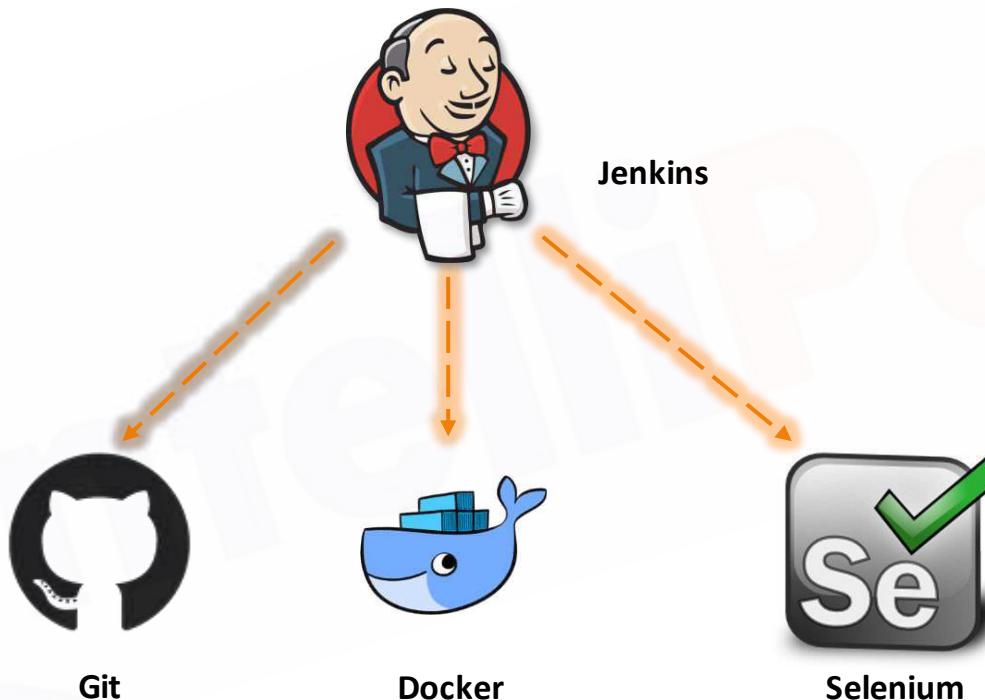
Add a slave node to Jenkins using JNLP connection





Jenkins Integration with DevOps Tools

Jenkins Integration with DevOps Tools



Jenkins Integration with DevOps Tools



Git



Docker



Selenium

Copy a Git repository to the slave's filesystem from Jenkins master



Jenkins Integration with DevOps Tools



Git

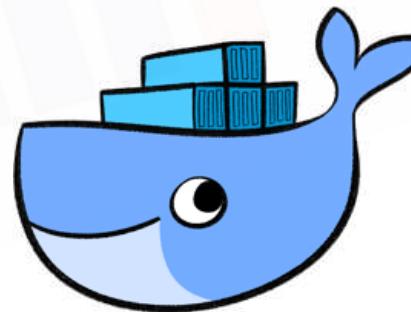
Containerize the website in the previous step to a Docker Container using Jenkins



Docker



Selenium



Jenkins Integration with DevOps Tools



Git



Docker



Selenium

Create a test case for the website in the previous step and execute the test on the slave using Jenkins

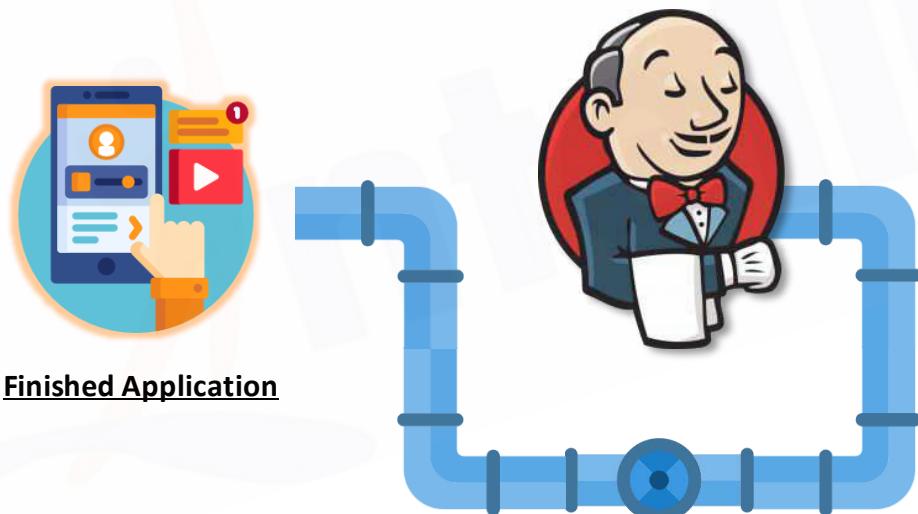




Understanding CI/CD Pipelines

What are CI/CD Pipelines?

CI/CD Pipelines, i.e., Continuous Integration, Continuous Delivery and Deployment pipelines, are a way of running Jenkins jobs in a sequence, which resembles a pipeline view.



Finished Application

What are CI/CD Pipelines?

CI/CD Pipelines, i.e., Continuous Integration, Continuous Delivery and Deployment pipelines, are a way of running Jenkins jobs in a sequence, which resembles a pipeline view.

For Example:



Creating an Automated CI/CD Pipeline

Creating an Automated CI/CD Pipeline



1. Initiate a Git Webhook for the Jenkins' git-job repository
2. Trigger the jobs after the completion of previous jobs with the following map: Git-Job → Build-Website → Website-Test
3. Install the plugin for the pipeline view
4. Make changes to the website and commit the job to see the changes



Quiz

1. Can Jenkins execute jobs without slaves?

- A. Yes
- B. No
- C. Minimum one slave is required
- D. None of these

1. Can Jenkins execute jobs without slaves?

- A. Yes
- B. No
- C. Minimum one slave is required
- D. None of these

2. Which plugin in Jenkins helps us see the pipeline view?

- A. Build Pipeline
- B. Create Pipeline
- C. Pipeline View
- D. None of these

2. Which plugin in Jenkins helps us see the pipeline view?

A. Build Pipeline

B. Create Pipeline

C. Pipeline View

D. None of these

3. Which Protocol in Jenkins is used to connect to Jenkins Slave?

- A. SSL
- B. SSH
- C. JNLP
- D. None of these

3. Which Protocol in Jenkins is used to connect to Jenkins Slave?

- A. SSL
- B. SSH
- C. JNLP
- D. None of these

4. Which of these will help you in triggering jobs from Git automatically?

A. Git Commit

B. Git Rebase

C. Git Webhook

D. None of these

4. Which of these will help you in triggering jobs from Git automatically?

A. Git Commit

B. Git Rebase

C. Git Webhook

D. None of these

5. Do the slaves need Jenkins installed on them?

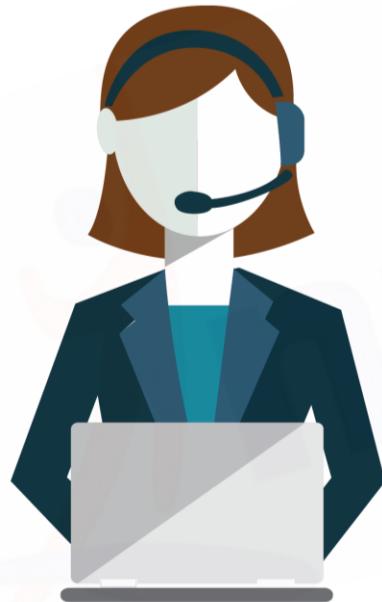
A. Yes

B. No

5. Do the slaves need Jenkins installed on them?

A. Yes

B. No



India: +91-7847955955



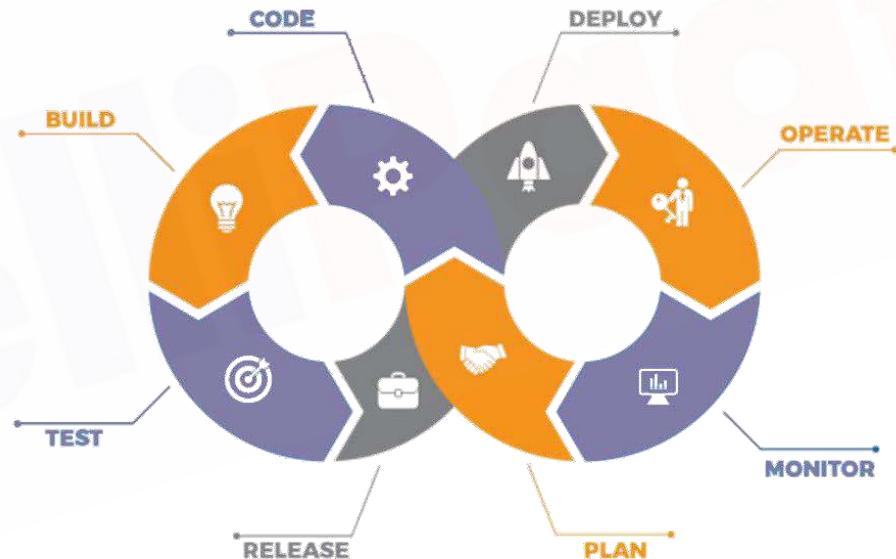
US: 1-800-216-8930 (TOLL FREE)



support@intellipaat.com

24/7 Chat with Our Course Advisor

Introduction to Kubernetes



Agenda

01

Introduction to
Kubernetes

02

Docker Swarm Vs.
Kubernetes

03

Kubernetes
Architecture

04

Kubernetes
Installation

05

Working of
Kubernetes

06

Deployments in
Kubernetes

07

Services in
Kubernetes

08

Ingress in
Kubernetes

09

Kubernetes
Dashboard

Introduction to Kubernetes

Introduction to Kubernetes



- ★ Kubernetes is an open-source container orchestration software.
- ★ It was originally developed by Google.
- ★ It was first released on July 21, 2015.
- ★ It is the ninth most active repository on GitHub in terms of number of commits.

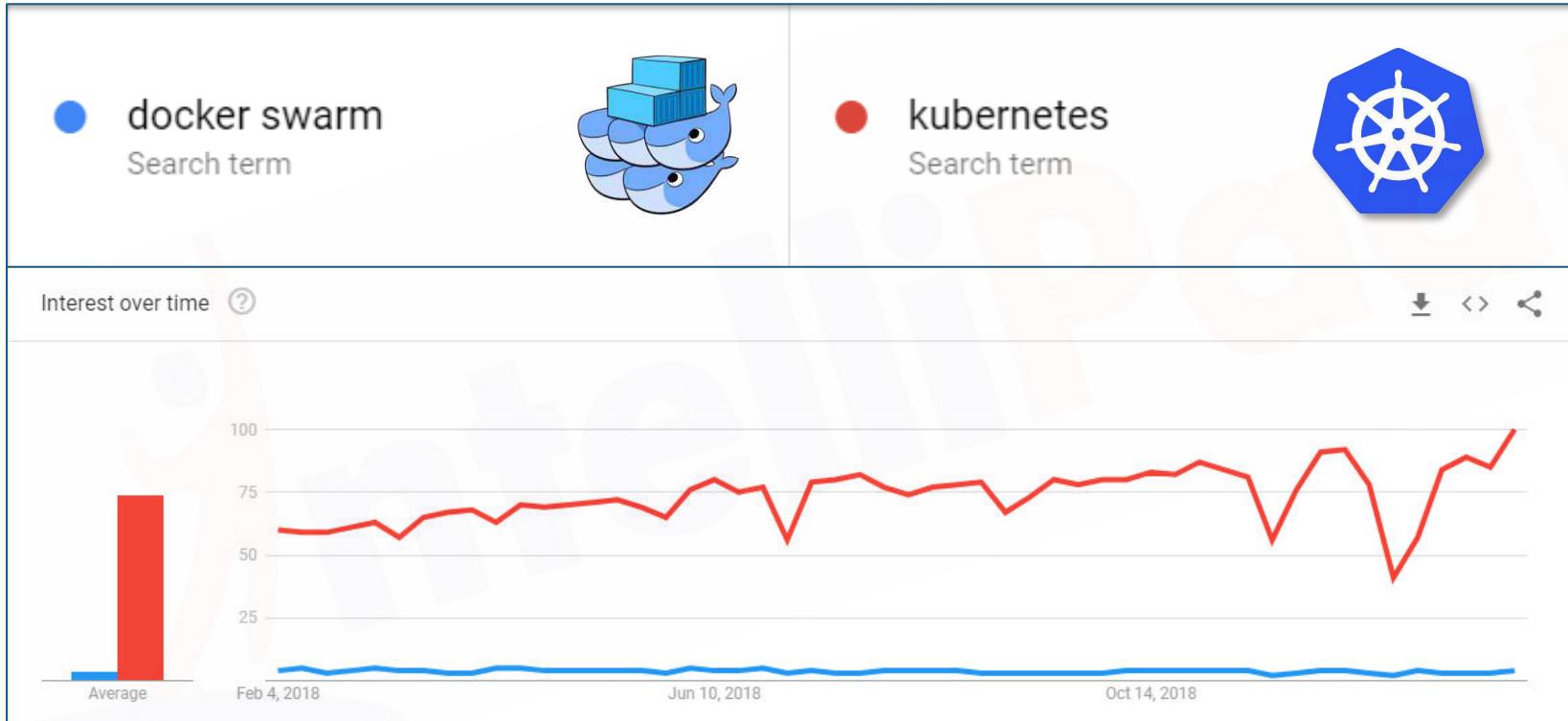
Features of Kubernetes

- ★ Pods
- ★ Service Discovery
- ★ Replication Controller
- ★ Networking
- ★ Storage Management
- ★ Secret Management
- ★ Resource Monitoring
- ★ Rolling Updates
- ★ Health Checks



Docker Swarm Vs. Kubernetes

Docker Swarm Vs. Kubernetes



Source: trends.google.com

Docker Swarm Vs. Kubernetes

Docker Swarm



- ★ Easy to install and initialize
- ★ Faster when compared to Kubernetes
- ★ Not reliable and has less features

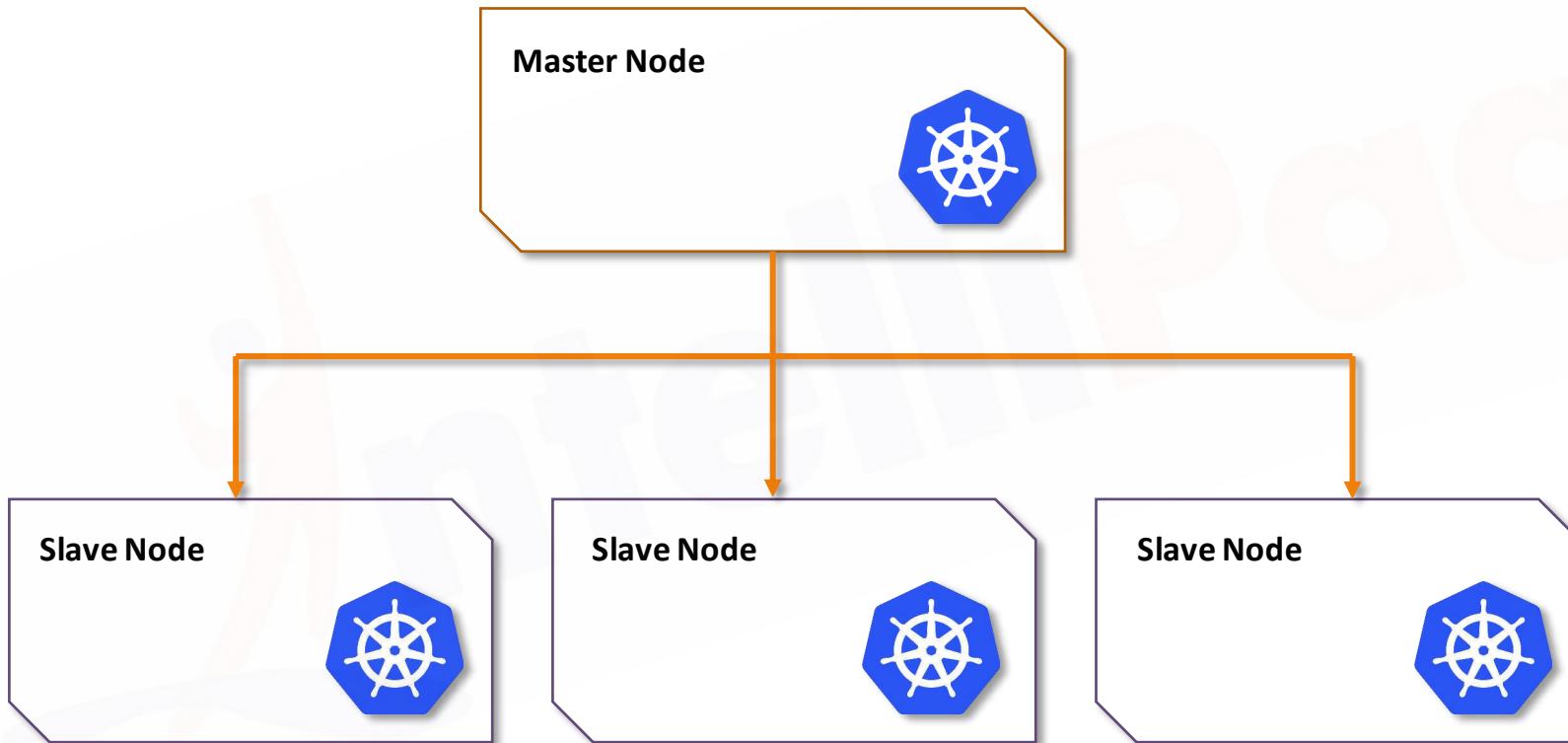
Kubernetes



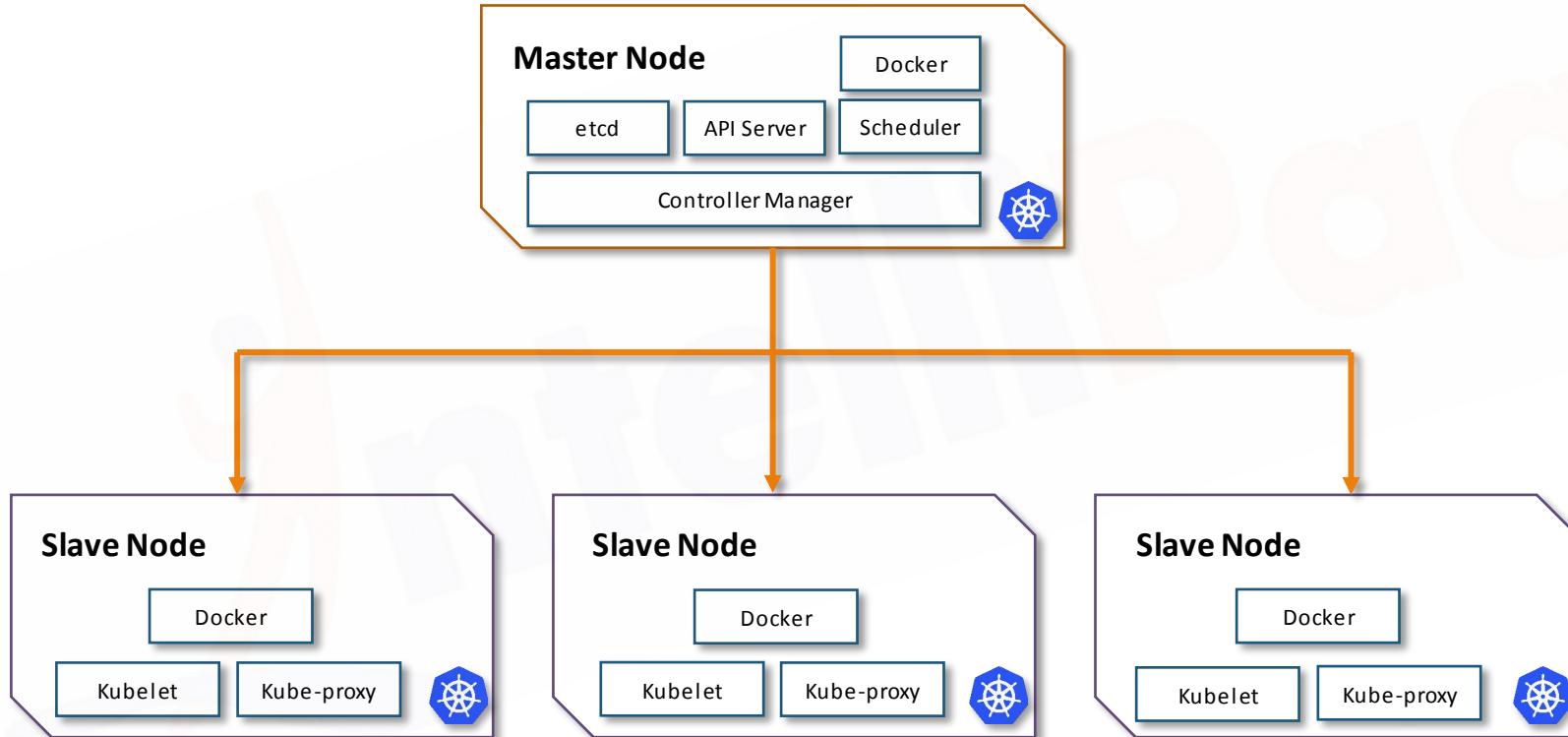
- ★ Complex procedure to install
- ★ Slower when compared to Docker Swarm
- ★ More reliable and has more features

Kubernetes Architecture

Kubernetes Architecture



Kubernetes Architecture



Kubernetes Architecture: Master Components

Kubernetes Architecture: Master Components



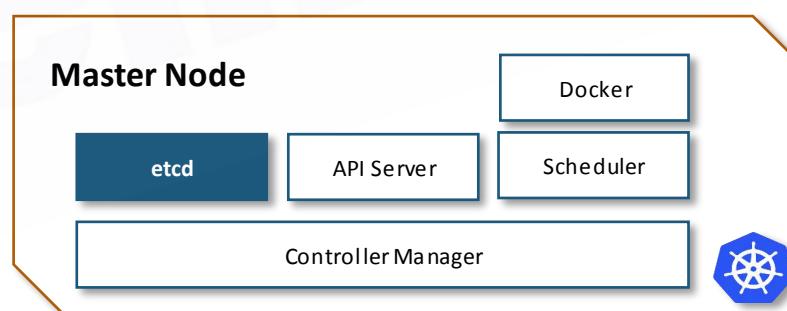
etcd

API Server

Scheduler

Controller Manager

It is a highly available distributed key–value store, which is used to store cluster wide secrets. It is only accessible by the Kubernetes API server, as it has sensitive information.



Kubernetes Architecture: Master Components



etcd

API Server

Scheduler

Controller Manager

It exposes Kubernetes API. Kubernetes API is the front-end for the Kubernetes Control Plane and is used to deploy and execute all operations in Kubernetes.

Master Node

etcd

API Server

Docker

Scheduler

Controller Manager



Kubernetes Architecture: Master Components



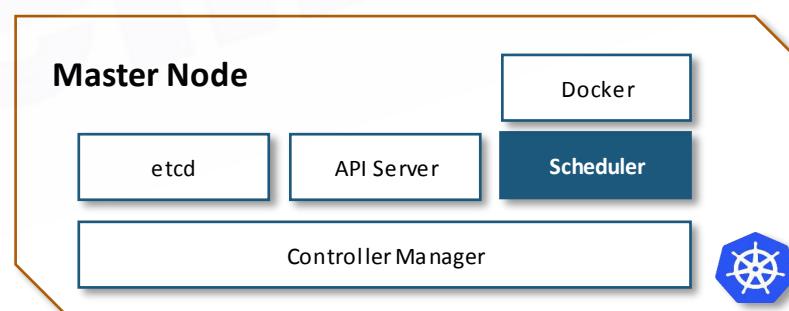
etcd

API Server

Scheduler

Controller Manager

The scheduler takes care of scheduling of all processes and the dynamic resource management and manages present and future events on the cluster.



Kubernetes Architecture: Master Components

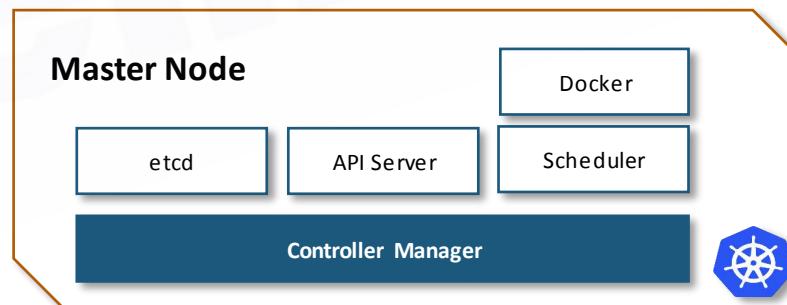
etcd

API Server

Scheduler

Controller Manager

The controller manager runs all controllers on the Kubernetes cluster. Although each controller is a separate process, to reduce complexity, all controllers are compiled into a single process. They are as follows: **Node Controller, Replication Controller, Endpoints Controller, Service Accounts and Token Controllers.**



Kubernetes Architecture: Slave Components

Kubernetes Architecture: Slave Components



Kubelet

Kube-proxy

Kubelet takes the specification from the API server and ensures that the application is running according to the specifications which were mentioned. Each node has its own kubelet service.

Slave Node

Docker

Kubelet

Kube-proxy



Kubernetes Architecture: Slave Components



Kubelet

Kube-proxy

This proxy service runs on each node and helps in making services available to the external host. It helps in connection forwarding to the correct resources. It is also capable of doing primitive load balancing.

Slave Node

Docker

Kubelet

Kube-proxy



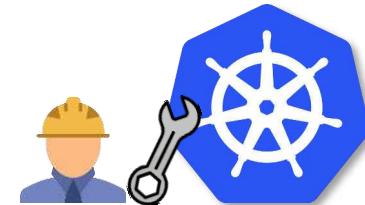


Kubernetes Installation

Kubernetes Installation

There are numerous ways to install Kubernetes. Following are some of the popular ways:

- **Kubeadm**: Bare Metal Installation
- **Minikube**: Virtualized Environment for Kubernetes
- **Kops**: Kubernetes on AWS
- **Kubernetes on GCP**: Kubernetes running on Google Cloud Platform



Hands-on: Installing Kubernetes Using Kubeadm

Working of Kubernetes

Working of Kubernetes



Pods can have one or more containers coupled together. They are the basic unit of Kubernetes. To increase high availability, we always prefer pods to be in replicas.



Pod – Replica 1



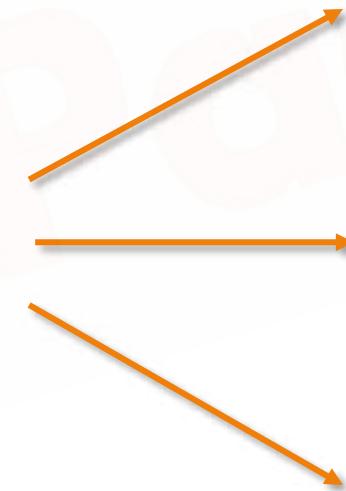
Pod – Replica 2



Pod – Replica 3

Working of Kubernetes

Services are used to load balance the traffic among the pods. It follows round-robin distribution among the healthy pods.



Pod – Replica 1

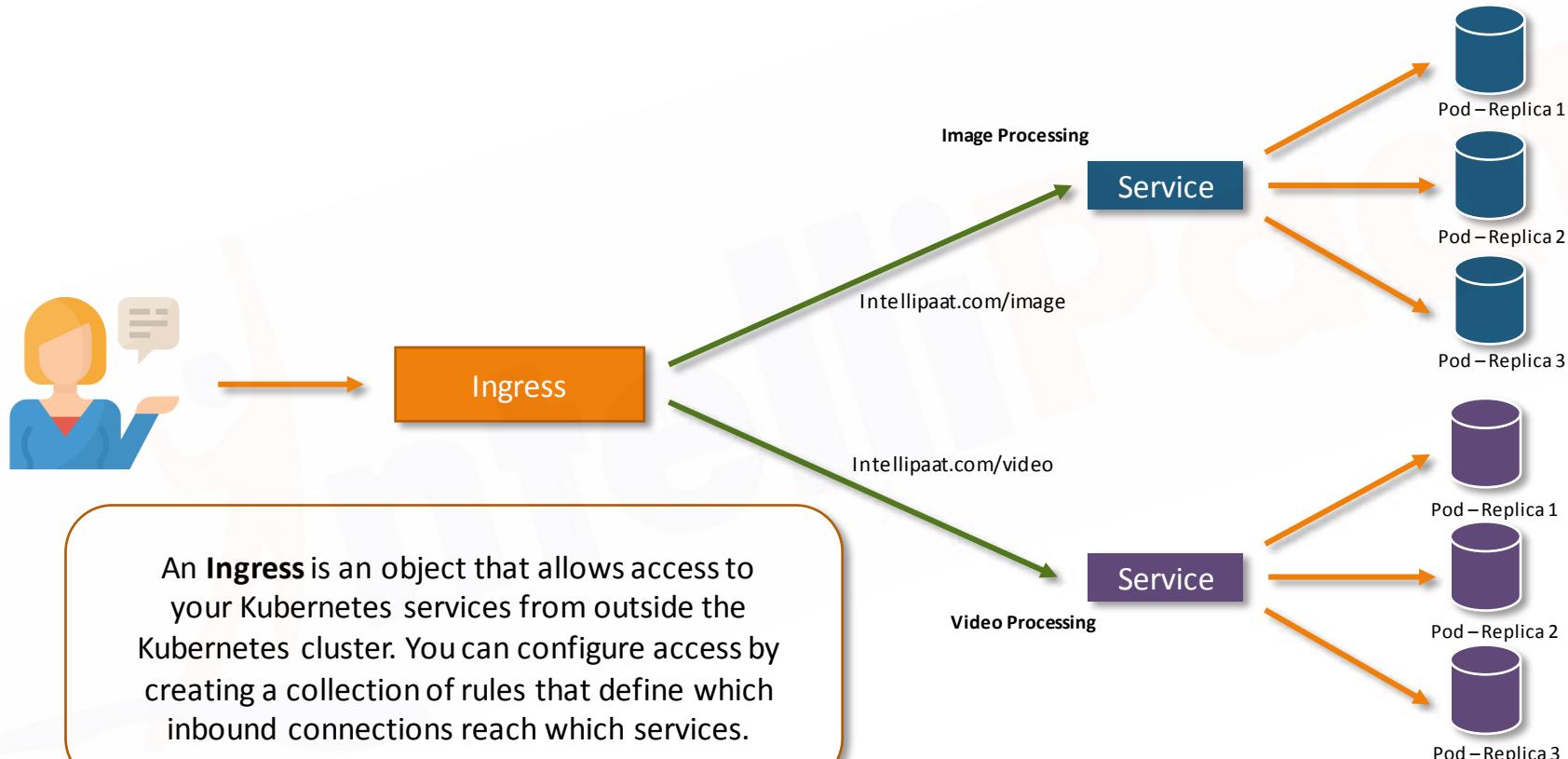


Pod – Replica 2



Pod – Replica 3

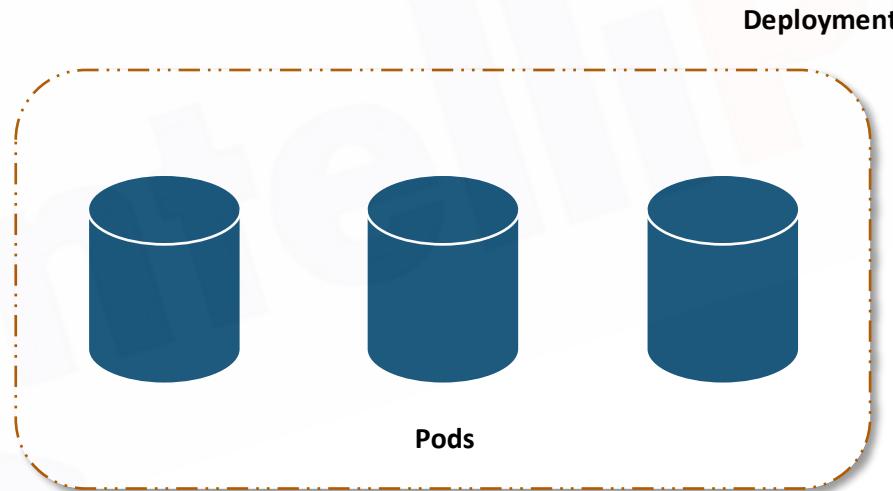
Working of Kubernetes



Deployments in Kubernetes

Deployments in Kubernetes

Deployment in Kubernetes is a controller which helps your applications reach the desired state; the desired state is defined inside the deployment file.



YAML Syntax for Deployments



This YAML file will deploy 3 pods for nginx and will maintain the desired state, which is 3 pods, until this deployment is deleted.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Creating a Deployment

Once the file is created, to deploy this deployment use the following syntax:

Syntax

```
kubectl create -f nginx.yaml
```

```
ubuntu@ip-172-31-39-244: ~
ubuntu@ip-172-31-39-244:~$ kubectl create -f nginx.yaml
deployment.apps/nginx-deployment created
ubuntu@ip-172-31-39-244:~$ █
```

Listing the Pods

To view the pods, type the following command:

Syntax

```
kubectl get po
```

```
ubuntu@ip-172-31-39-244: ~$ kubectl get po
NAME                      READY   STATUS    RESTARTS   AGE
nginx-deployment-76bf4969df-24vp1   1/1     Running   0          4m38s
nginx-deployment-76bf4969df-frz7j   1/1     Running   0          4m38s
nginx-deployment-76bf4969df-grnmc   1/1     Running   0          4m38s
ubuntu@ip-172-31-39-244: ~$
```

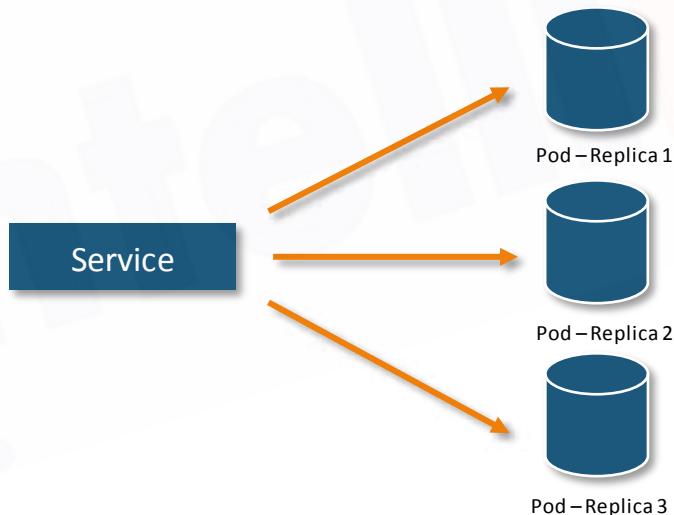
As you can see, the number of pods are matching with the number of replicas specified in the deployment file.



Creating a Service

Creating a Service

A Service is basically a round-robin load balancer for all pods, which matches with its name or selector. It constantly monitors the pods; in case a pod gets unhealthy, the service will start deploying the traffic to other healthy pods.



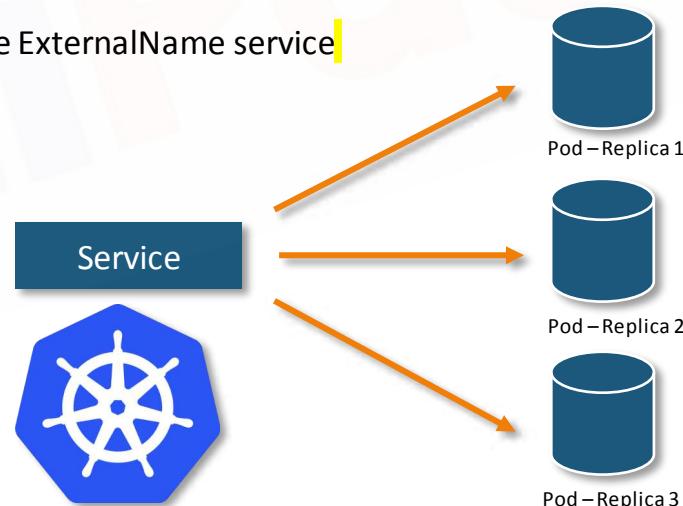
Service Types

ClusterIP: Exposes the service on cluster-internal IP

NodePort: Exposes the service on each Node's IP at a static port

LoadBalancer: Exposes the service externally using a cloud provider's load balancer

ExternalName: Maps the service to the DNS Name mentioned with the ExternalName service



Creating a NodePort Service

We can create a NodePort service using the following syntax:

Syntax

```
kubectl create service nodeport <name-of-service> --tcp=<port-of-service>:<port-of-container>
```

```
ubuntu@ip-172-31-39-244:~$ kubectl create service nodeport nginx --tcp=80:80
service/nginx created
ubuntu@ip-172-31-39-244:~$ █
```

Creating a NodePort Service

To know the port, on which the service is being exposed, type the following command:

Syntax

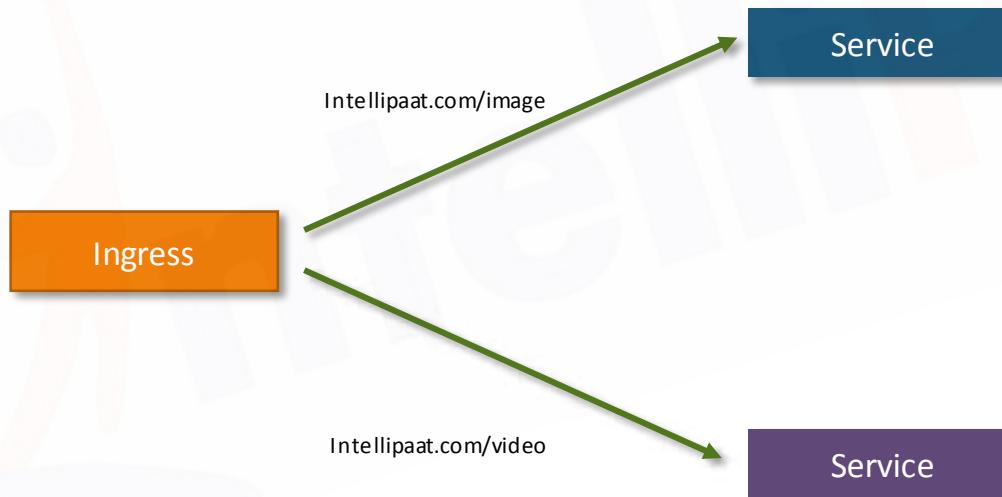
```
kubectl get svc nginx
```

```
ubuntu@ip-172-31-39-244:~$ kubectl get svc nginx
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx    NodePort    10.103.235.81    <none>        80:32043/TCP   114s
ubuntu@ip-172-31-39-244:~$
```

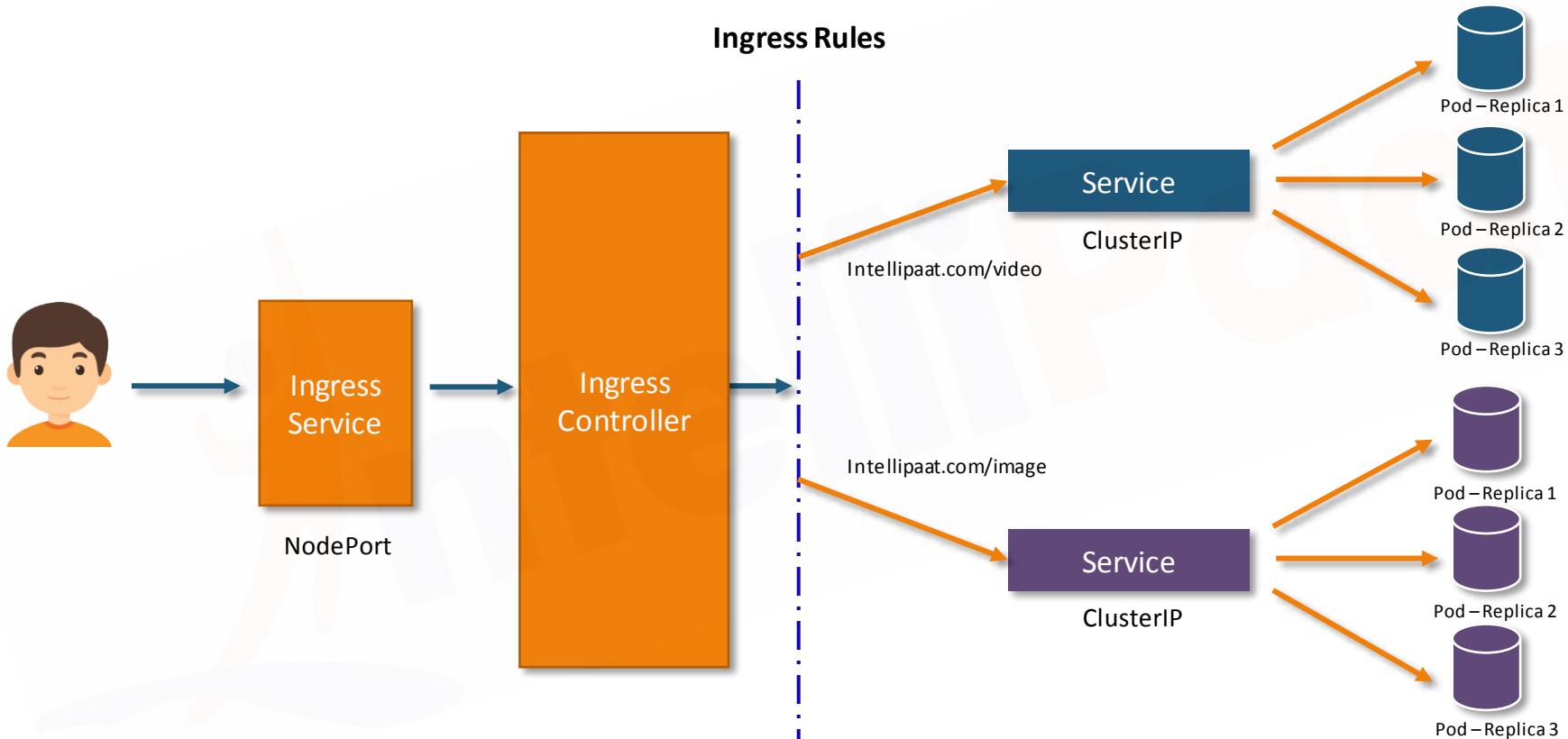
Creating an Ingress

What is an Ingress?

Kubernetes ingress is a collection of routing rules that govern how external users access services running in a Kubernetes cluster.



What is an Ingress?



Installing Ingress Controller



We will be using the nginx ingress controller for our demo. We can download it from the following link:

Link

<https://github.com/kubernetes/ingress-nginx/blob/master/docs/deploy/index.md>

A large, bold, green "NGINX" logo centered on the page.

Defining Ingress Rules

The following rule, will redirect traffic which asks for /foo to nginx service. All other requests will be redirected to ingress controller's default page.

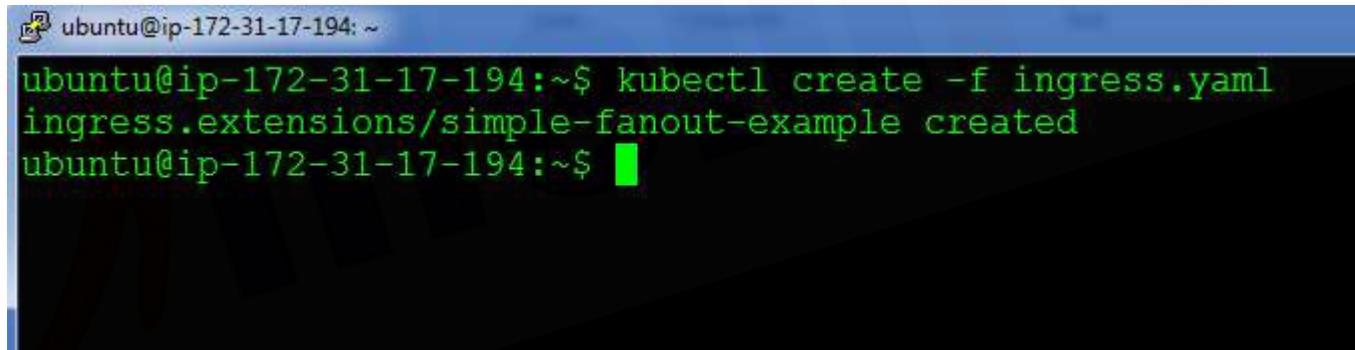
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /foo
        backend:
          serviceName: nginx
          servicePort: 80
```

Deploying Ingress Rules

To deploy ingress rules, we use the following syntax:

Syntax

```
kubectl create -f ingress.yaml
```



A screenshot of a terminal window on an Ubuntu system. The terminal prompt is "ubuntu@ip-172-31-17-194:~\$". The user has run the command "kubectl create -f ingress.yaml" and the output shows "ingress.extensions/simple-fanout-example created". The terminal has a blue header bar and a black body.

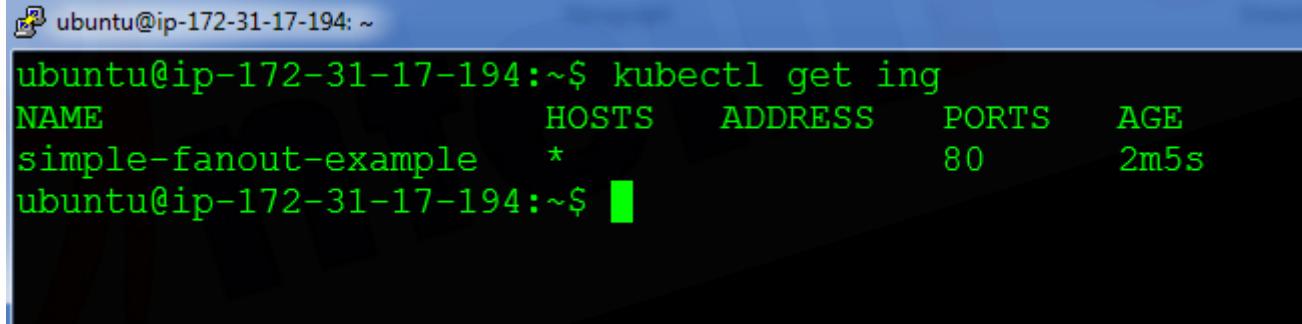
```
ubuntu@ip-172-31-17-194:~$ kubectl create -f ingress.yaml
ingress.extensions/simple-fanout-example created
ubuntu@ip-172-31-17-194:~$ █
```

Viewing Ingress Rules

To list the ingress rules we use the following syntax:

Syntax

kubectl get ing



A terminal window showing the command `kubectl get ing` being run. The output displays a table with columns: NAME, HOSTS, ADDRESS, PORTS, and AGE. A single ingress rule named `simple-fanout-example` is listed, which routes all traffic (*) to port 80. The entry was created 2m5s ago.

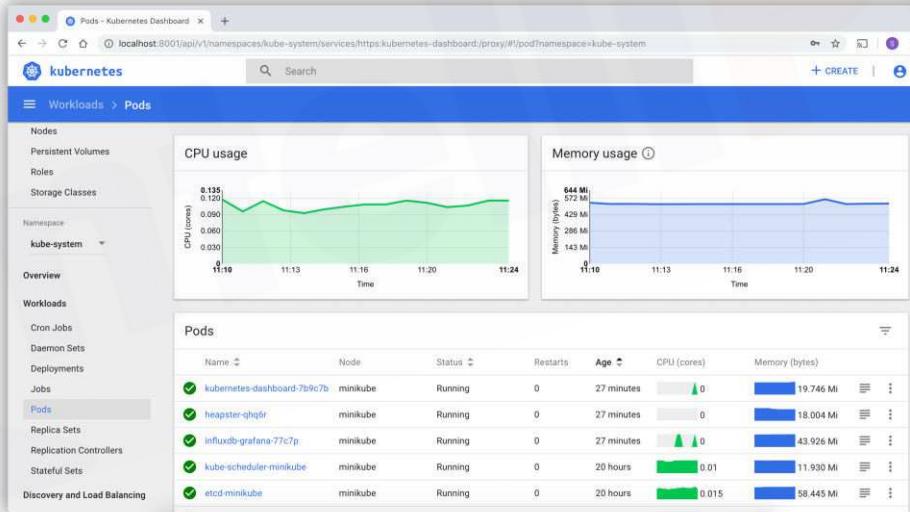
NAME	HOSTS	ADDRESS	PORTS	AGE
simple-fanout-example	*		80	2m5s



Kubernetes Dashboard

Kubernetes Dashboard

Dashboard is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application and manage cluster resources.



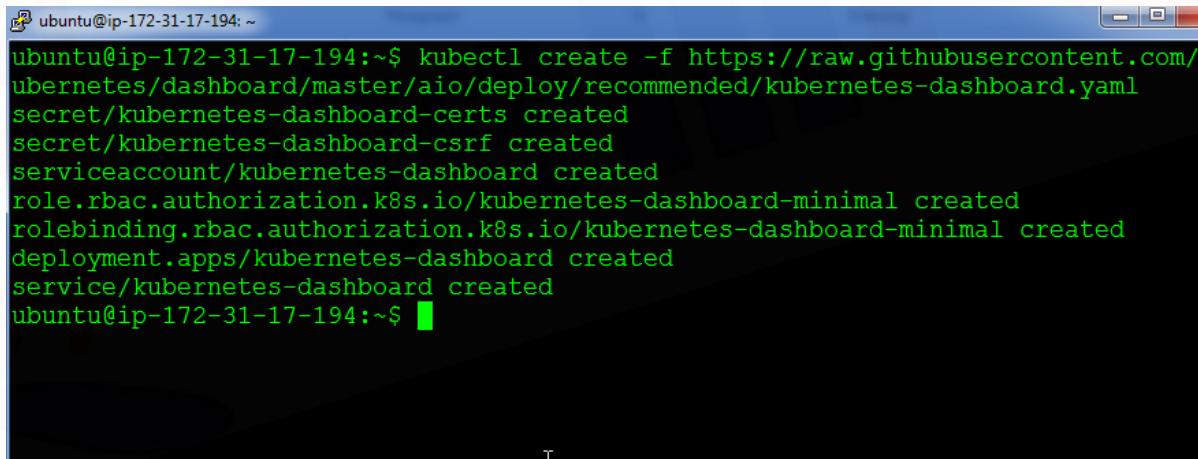
Installing Kubernetes Dashboard

To install Kubernetes Dashboard, execute the following command:

Syntax

```
kubectl create -f
```

```
https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended/kubernetes-dashboard.yaml
```

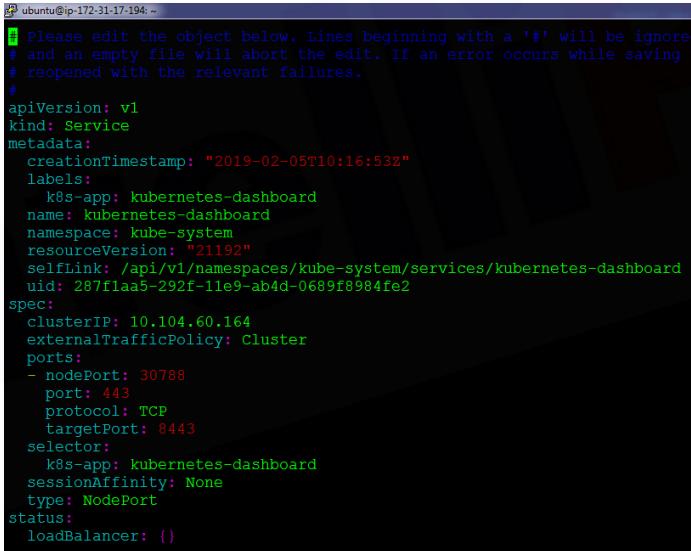
A screenshot of a terminal window titled "ubuntu@ip-172-31-17-194: ~". The window shows the command "kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended/kubernetes-dashboard.yaml" being run and its output. The output details the creation of various Kubernetes resources: a secret for certificates, a service account, a role and role binding for RBAC, and a deployment for the dashboard. The terminal has a dark theme with light-colored text and standard window controls at the top.

Accessing Kubernetes Dashboard

Change the service type for kubernetes-dashboard to NodePort

Syntax

```
kubectl -n kube-system edit service kubernetes-dashboard
```



```
ubuntu@ip-172-31-17-194: ~
$ kubectl -n kube-system edit service kubernetes-dashboard
# Please edit the object below. Lines beginning with a '#' will be ignored
# and an empty file will abort the edit. If an error occurs while saving t
# reopened with the relevant failures.
#
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2019-02-05T10:16:53Z"
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
  resourceVersion: "21192"
  selfLink: /api/v1/namespaces/kube-system/services/kubernetes-dashboard
  uid: 287f1aa5-292f-11e9-ab4d-0689f8984fe2
spec:
  clusterIP: 10.104.60.164
  externalTrafficPolicy: Cluster
  ports:
  - nodePort: 30788
    port: 443
    protocol: TCP
    targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

Logging into Kubernetes Dashboard

1. Check the NodePort from the kubernetes-dashboard service
2. Browse to your cluster on the Internet browser, and enter the IP address
3. Click on Token, which will ask you for the token entry
4. Generate a token using the following command:

```
$ kubectl create serviceaccount cluster-admin-dashboard-sa
$ kubectl create clusterrolebinding cluster-admin-dashboard-sa \
--clusterrole=cluster-admin \
--serviceaccount=default:cluster-admin-dashboard-sa

$ TOKEN=$(kubectl describe secret $(kubectl -n kube-system get secret | awk '/^cluster-admin-dashboard-sa-token-/ {print $1}') | awk '$1=="token:" {print $2}')

$ echo $TOKEN
```

5. Finally, enter the token and login to your dashboard

Hands-on: Deploying an App Using Dashboard



Quiz

1. Which of these is an installation method of Kubernetes cluster?

- A. Kubeadm
- B. Kops
- C. Both A and B
- D. None of these

1. Which of these is an installation method of Kubernetes cluster?

A. Kubeadm

B. Kops

C. Both A and B

D. None of these

2. Which of these components is a distributed key–value store?

- A. Kubelet
- B. Scheduler
- C. etcd
- D. None of these

2. Which of these components is a distributed key-value store?

- A. Kubelet
- B. Scheduler
- C. etcd
- D. None of these

3. Which type of service is used to expose application without using Cloud Native Support?

A. Load Balancer

B. NodePort

C. ExternalName

D. None of these

3. Which type of service is used to expose application without using Cloud Native Support?

A. Load Balancer

B. NodePort

C. ExternalName

D. None of these

4. Which of these is not a component of Kubernetes Slave?

- A. Kubelet
- B. Docker
- C. Scheduler
- D. None of these

4. Which of these is not a component of Kubernetes Slave?

- A. Kubelet
- B. Docker
- C. Scheduler
- D. None of these

5. Which of these components helps us to route traffic based on the user request?

A. Deployment

B. Service

C. Ingress

D. None of these

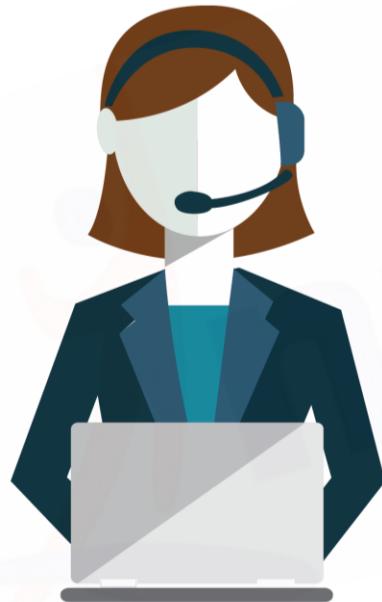
5. Which of these components helps us to route traffic based on the user request?

A. Deployment

B. Service

C. Ingress

D. None of these



India: +91-7847955955



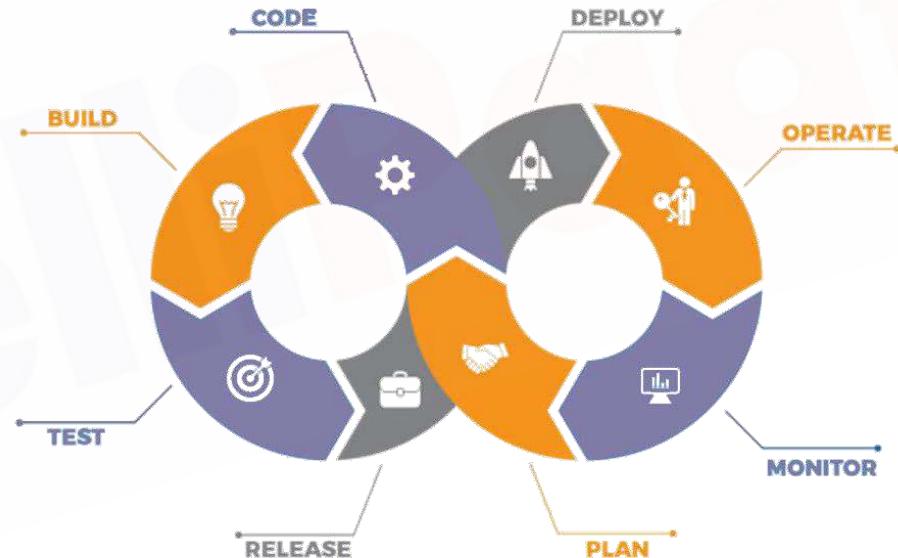
US: 1-800-216-8930 (TOLL FREE)



support@intellipaat.com

24/7 Chat with Our Course Advisor

Continuous Monitoring Using Nagios



Agenda

01

What is Continuous Monitoring?

02

Introduction to Nagios

03

Nagios Architecture

04

Installing Nagios on AWS

05

Nagios Components

06

What are Plugins?

07

Adding a Host in Nagios Using NRPE Plugin

08

Monitoring Service on Host Using NRPE

What is Continuous Monitoring?

What is Continuous Monitoring?

Continuous monitoring is the process and technology used to detect compliance and risk issues associated with an organization's financial and operational environment. The financial and operational environment consists of people, processes and systems working together to support efficient and effective operations.



Why Continuous Monitoring?

It detects any network or server problems.

It determines the root cause of any issues.

It maintains the security and availability of the service.

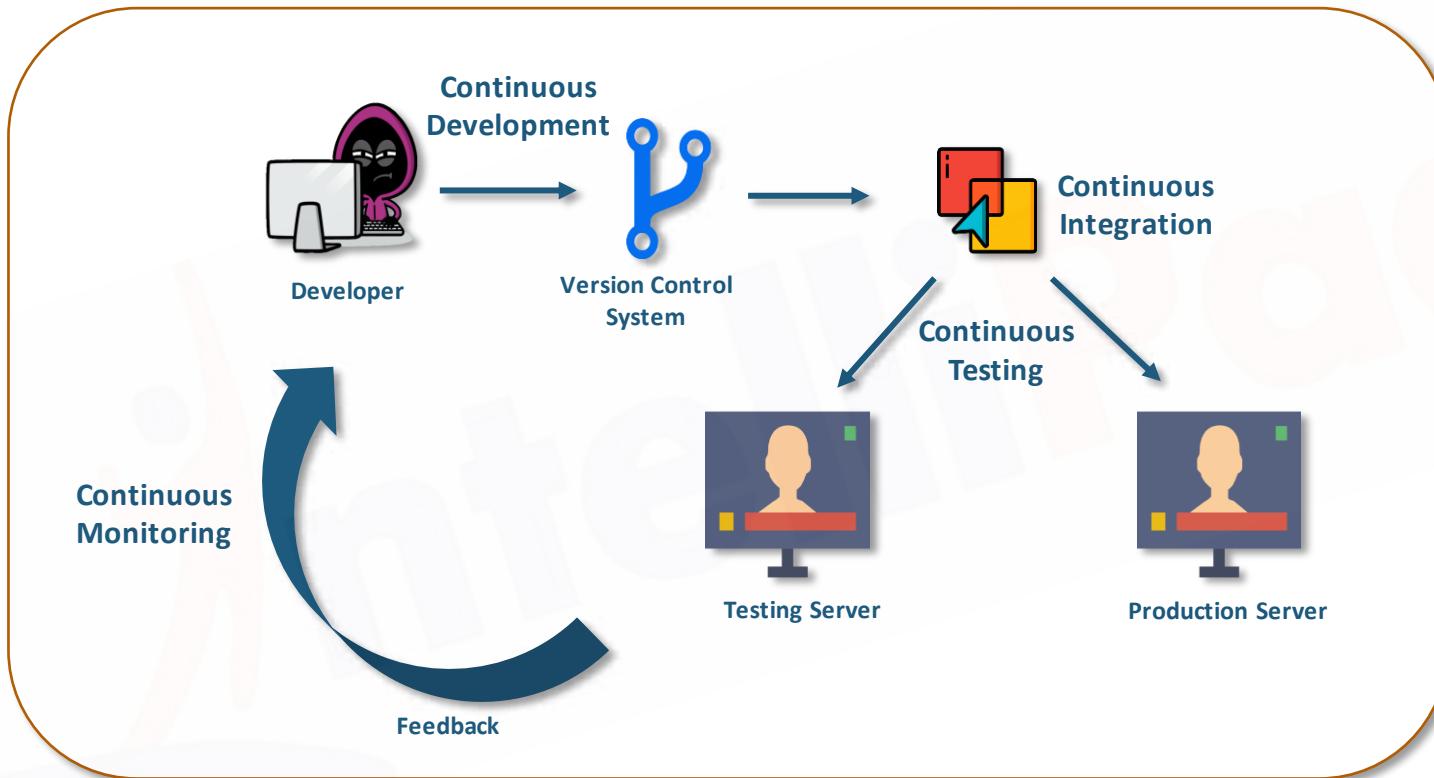
It monitors and troubleshoots server performance issues.

It can respond to issues at the first sign of a problem.

It monitors your entire infrastructure and business processes.



What is Continuous Monitoring?



Continuous Monitoring Tools

Continuous Monitoring Tools



Nagios®

splunk®>

Introduction to Nagios

Introduction to Nagios

Nagios, now known as Nagios Core, is a free and open-source computer-software application that monitors systems, networks and infrastructure. Nagios offers monitoring and alerting services for servers, switches, applications and services.

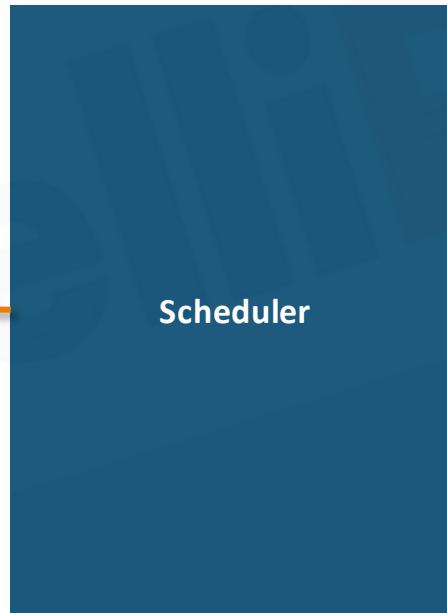
Nagios[®]

Nagios Architecture

Nagios Architecture



Nagios GUI



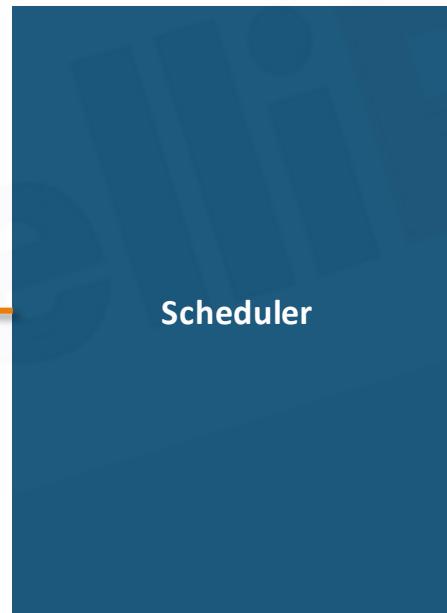
NRPE Plugins

Nagios Architecture



Nagios GUI

This is a web UI, which can be used to check the status of the host or the services.

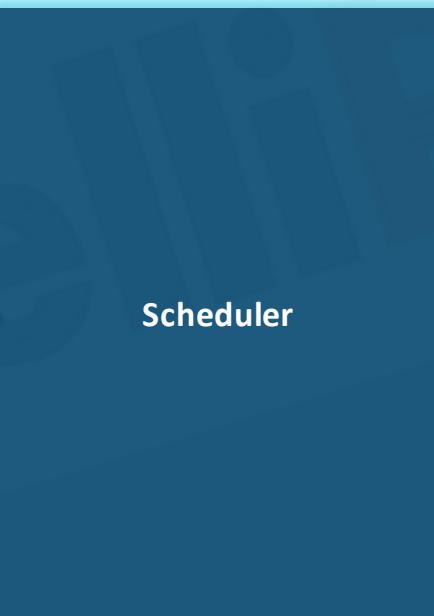


Nagios Architecture



Nagios GUI

The scheduler does the job of scheduling checks, i.e., what to check and when to check it.

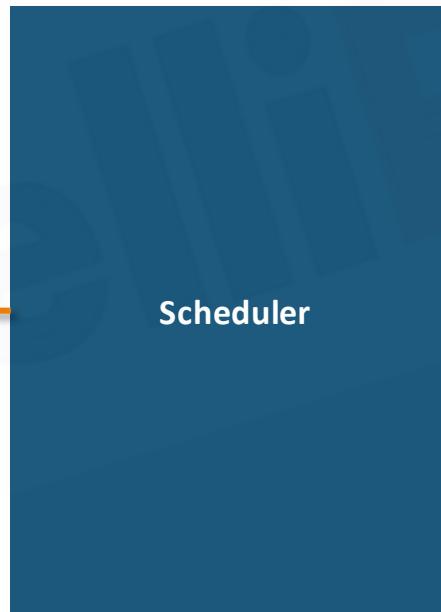


NRPE Plugins

Nagios Architecture



Nagios GUI



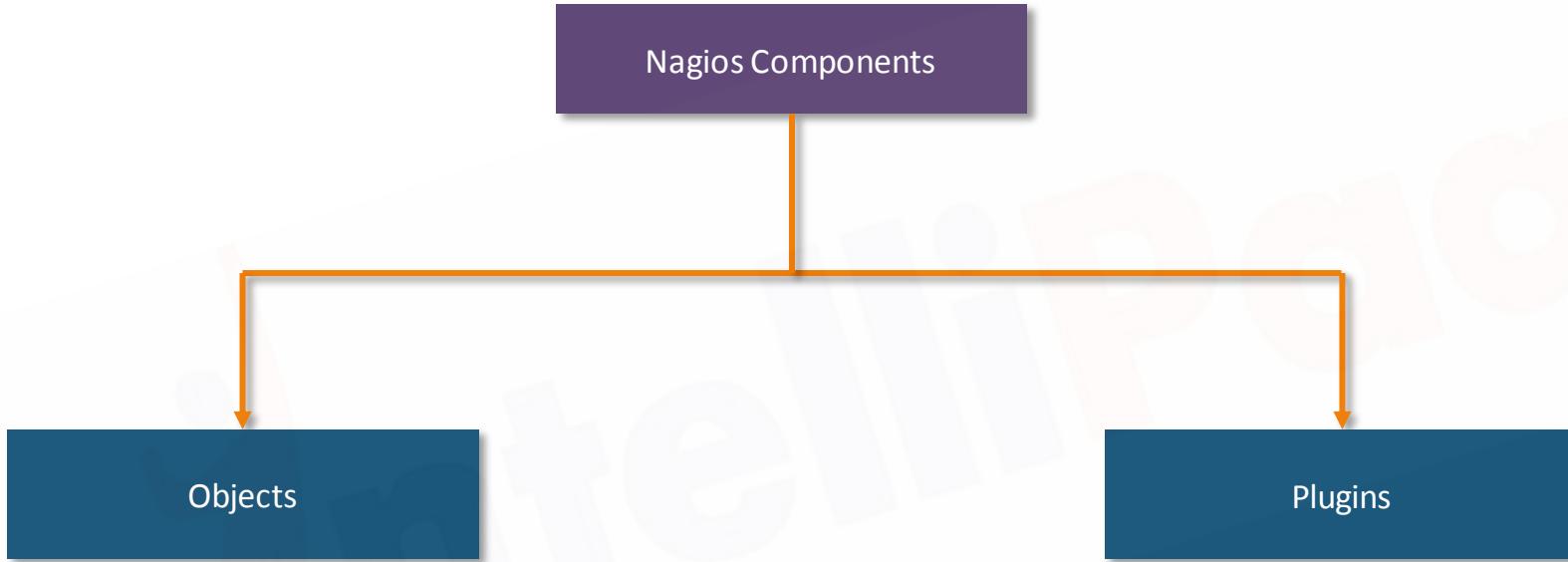
NRPE Plugins

The clients do not have Nagios installed on them. Hence, the log data of clients are sent through plugins.

Installing Nagios on AWS

Nagios Components

Nagios Components

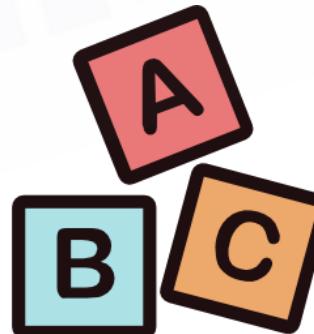


Nagios Components

Objects

Objects are all the elements that are involved in the monitoring and notification logic. When Nagios starts, restarts or reloads, it reads all the “.cfg” files within the object directories.

Plugins



Nagios Components

Objects

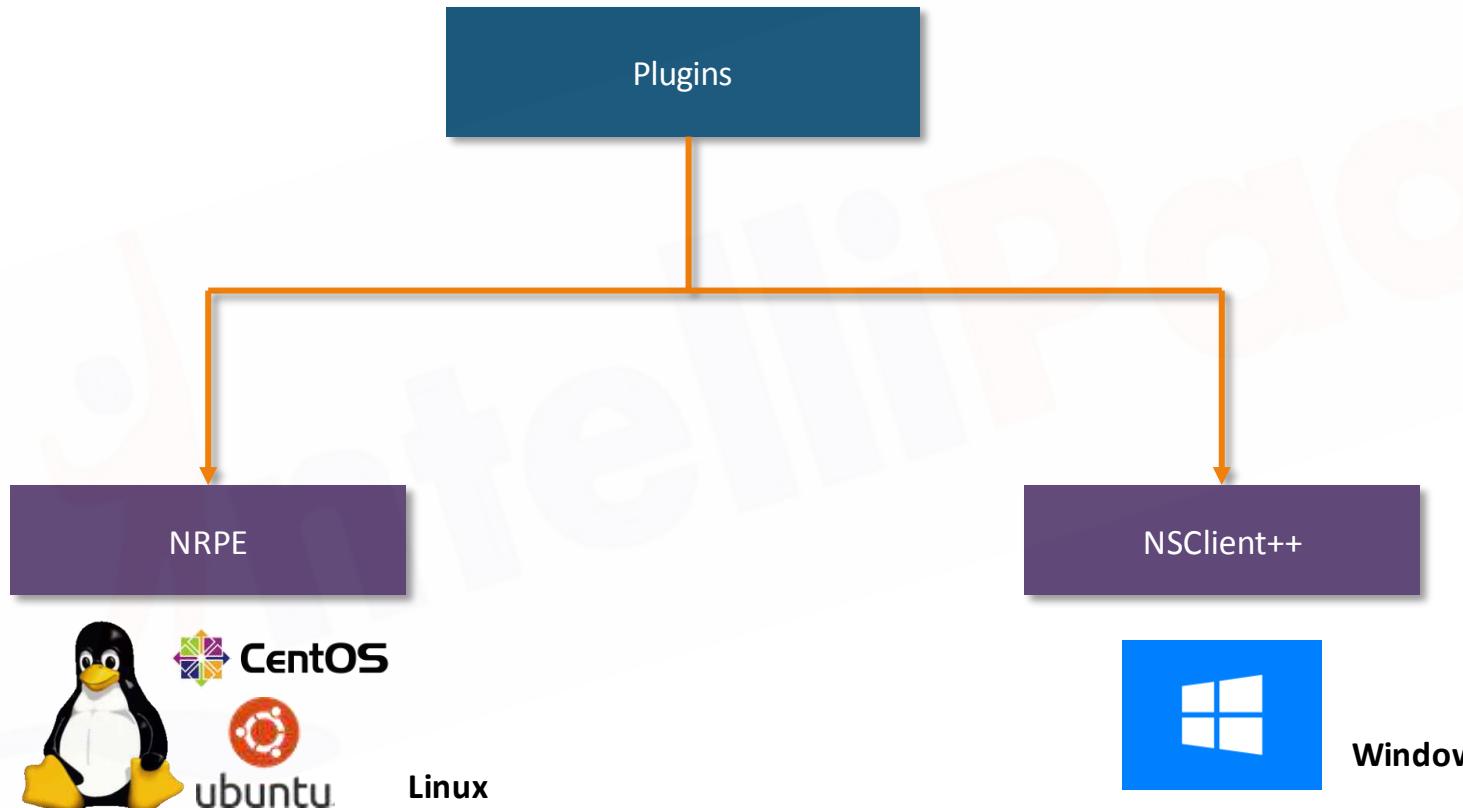
Plugins

Plugins are the piece of software that one installs on the Nagios slave/host, using which the host can interact with the Nagios Server and send the application logs.



Type of Plugins in Nagios

Types of Plugins in Nagios



Creating a Host in Nagios Using NRPE Plugin

Creating a Host in Nagios Using NRPE



1. Launch an Ubuntu box on AWS
2. Follow the commands for Installing NRPE Plugin
3. Add the Host in Nagios Server in the objects folder
4. Change the Main Nagios Configuration file for reading this file
5. Finally, restart the Nagios Server

Creating a Monitoring Service in Nagios for Remote NRPE Client

Quiz

1. Which configuration file will have all the sample syntax defined?

- A. templates.cfg
- B. servers.cfg
- C. printer.cfg
- D. switches.cfg

1. Which configuration file will have all the sample syntax defined?

- A. templates.cfg
- B. servers.cfg
- C. printer.cfg
- D. switches.cfg

2. You defined a new host in Nagios. After defining when you went to the Nagios GUI, you did not find the Nagios host listed. What will be the first thing that you will check?

- A. Whether Nagios Host is running or not
- B. Whether Nagios Core was restarted
- C. Nagios Host's definition
- D. None of these

2. You defined a new host in Nagios. After defining when you went to the Nagios GUI, you did not find the Nagios host listed. What will be the first thing that you will check?

- A. Whether Nagios Host is running or not
- B. Whether Nagios Core was restarted**
- C. Nagios Host's definition
- D. None of these

3. Which Plugin is used to connect to Windows Hosts?

A. NSClient++

B. NRPE

C. Build Pipeline

D. None of these

3. Which Plugin is used to connect to Windows Hosts?

A. NSClient++

B. NRPE

C. Build Pipeline

D. None of these

4. Can we disable Notifications for host in Nagios?

A. Yes

B. No

4. Can we disable Notifications for host in Nagios?

A. Yes

B. No

Quiz

5. On which port does Nagios GUI work?

- A. 8000
- B. 80
- C. 8001
- D. None of these

Quiz

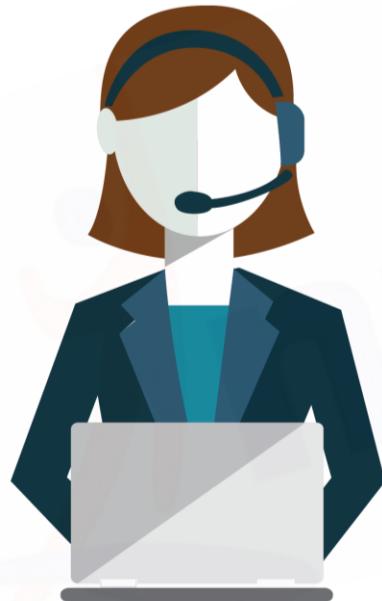
5. On which port does Nagios GUI work?

A. 8000

B. 80

C. 8001

D. None of these



India: +91-7847955955



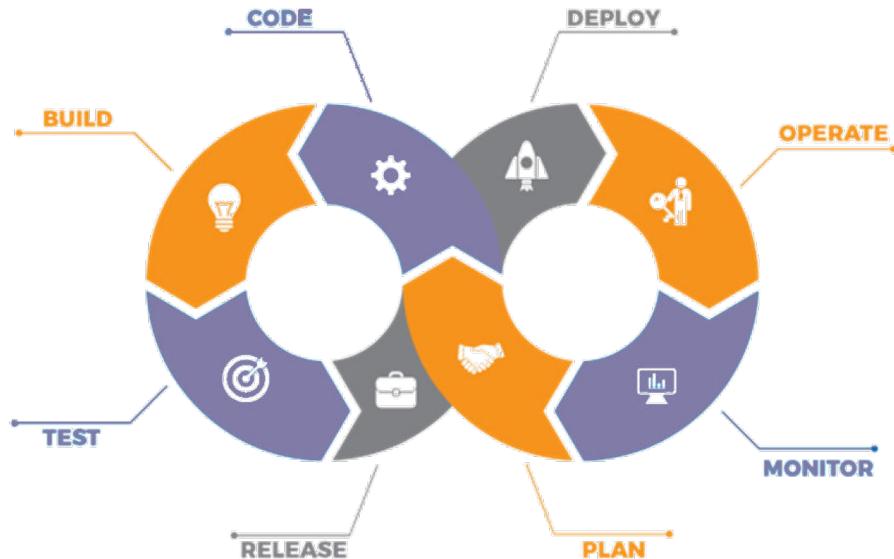
US: 1-800-216-8930 (TOLL FREE)



sales@intellipaat.com

24/7 Chat with Our Course Advisor

Introduction to Terraform



Agenda

01

What is Infrastructure
as a Code?

02

Infrastructure as a Code vs
Configuration Management

03

Introduction to
Terraform

04

Installing Terraform
on AWS EC2

05

Basic Terraform
Operations

06

Terraform Code
Basics

07

Deploying an end to end
Architecture using
Terraform

What is Infrastructure as a Code?

What is Infrastructure as Code (IaC)?

Infrastructure as Code (IaC) is the management of **infrastructure** (networks, virtual machines, load balancers, and connection topology) in a descriptive model, using the same versioning as **DevOps** team uses for source **code**. **Infrastructure as Code** evolved to solve the problem of environment drift in the release pipeline.



Infrastructure as Code vs Configuration Management

IaC vs Configuration Management

Infrastructure as a Code

1. Can create / destroy hardware architectures
2. Can install software while bootstrapping servers
3. Should not be used as a replacement to CM tools

Configuration Management

1. Works only with softwares
2. Cannot work on hardware level, but can install any software
3. Cannot be used as a replacement to IaC tools

Introduction to Terraform

What is Infrastructure as Code (IaC)?

Terraform is an open-source infrastructure as code software tool created by HashiCorp. It enables users to define and provision a datacenter infrastructure using a high-level configuration language known as Hashicorp Configuration Language, or optionally JSON.

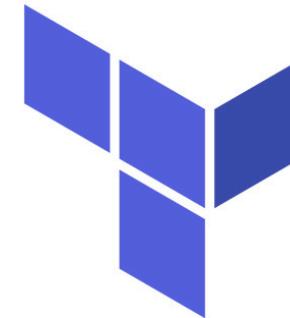


Installing Terraform for AWS

Installing Terraform for AWS



1. Install Terraform on local/EC2 Machine
2. Create a user, and use this user to authenticate terraform to AWS
3. Initialize Terraform Directory

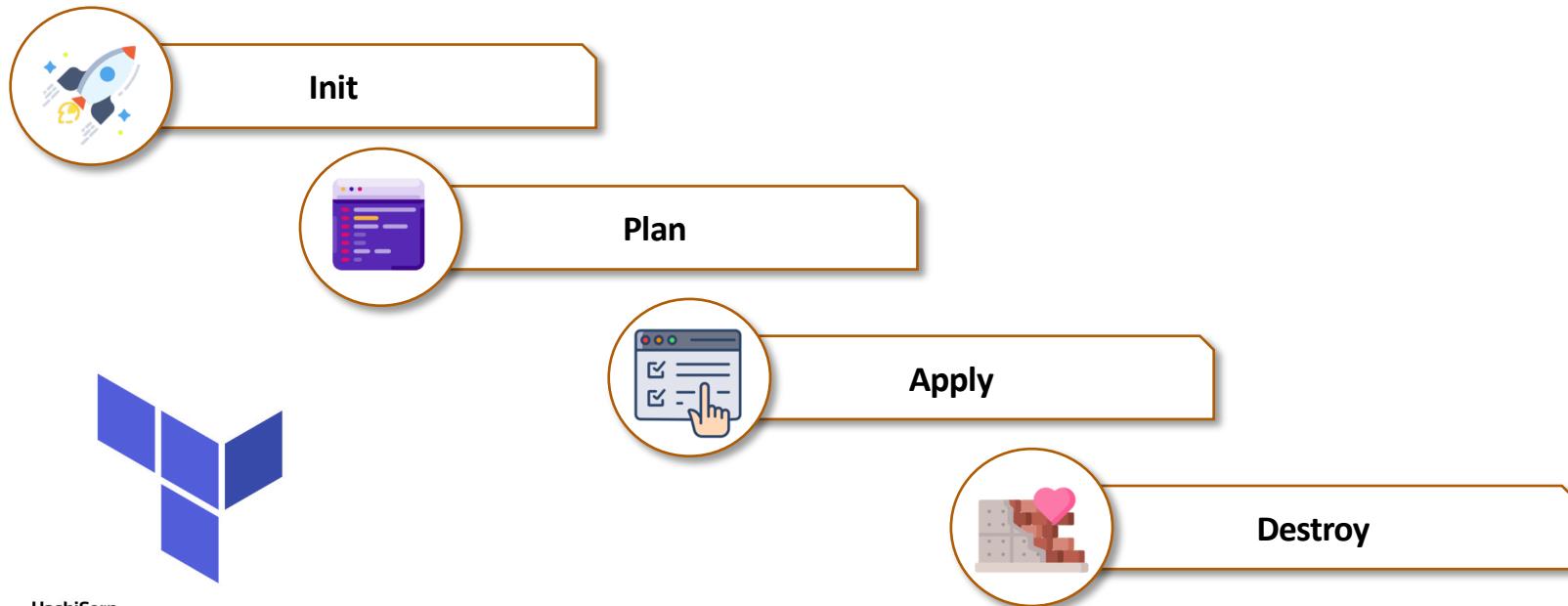


HashiCorp
Terraform

Basic Terraform Operations

Basic Terraform Operations

There are four kinds of operations in Terraform:



Terraform Code Basics

Terraform Code Basics

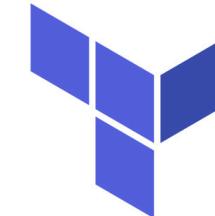
Following is a snippet of terraform file. Terraform files are saved with the extension of tf (*.tf). It has two main components, **block type** and **Key Value Pairs**

Block Type

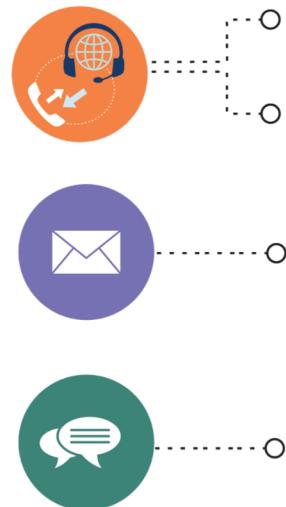
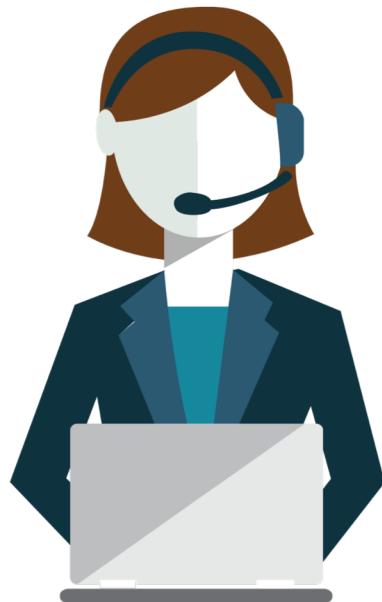
```
1 provider "aws" {  
2  
3     region = "ap-south-1"  
4     access_key = "AKIA2ESXR0HTZZNFXDKI"  
5     secret_key = "frQzLIIH0DoRISZ6wqtI0Z7Uoe8IuFIwxVXIjk1B"  
6  
7 }  
8
```



Key Value Pairs



Deploying end to end Infrastructure using Terraform



India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor