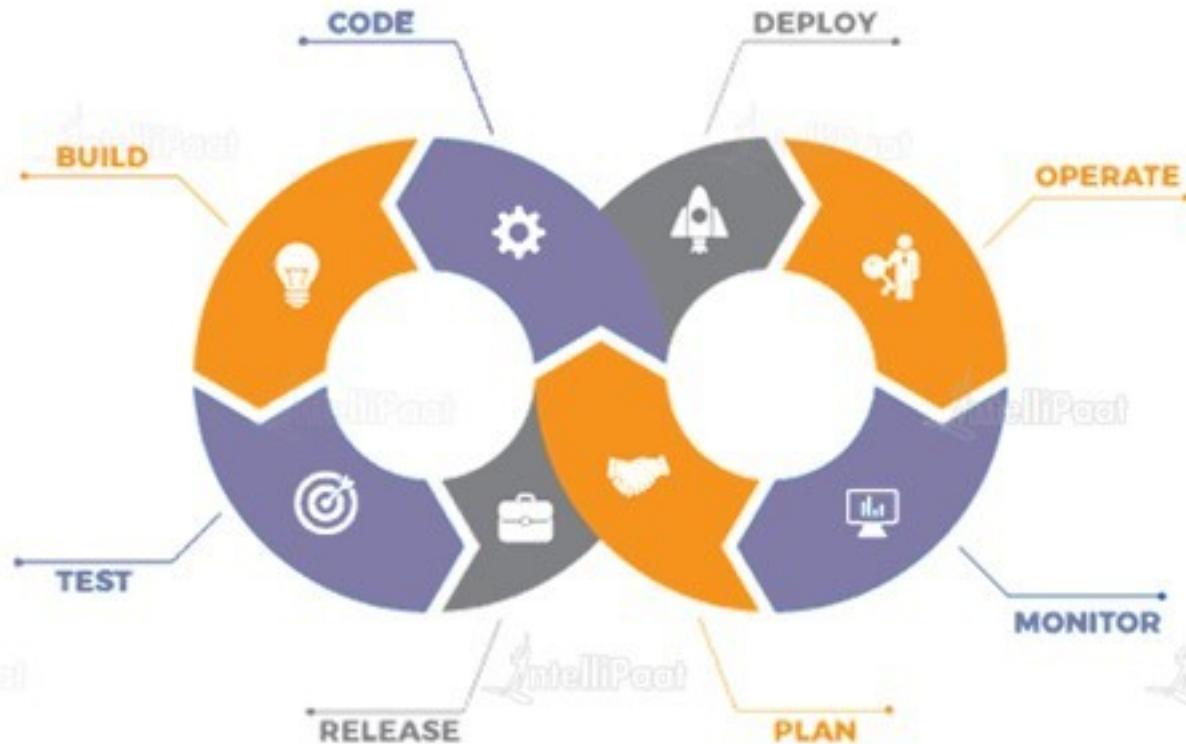




Introduction to DevOps



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.



SDLC Models

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:

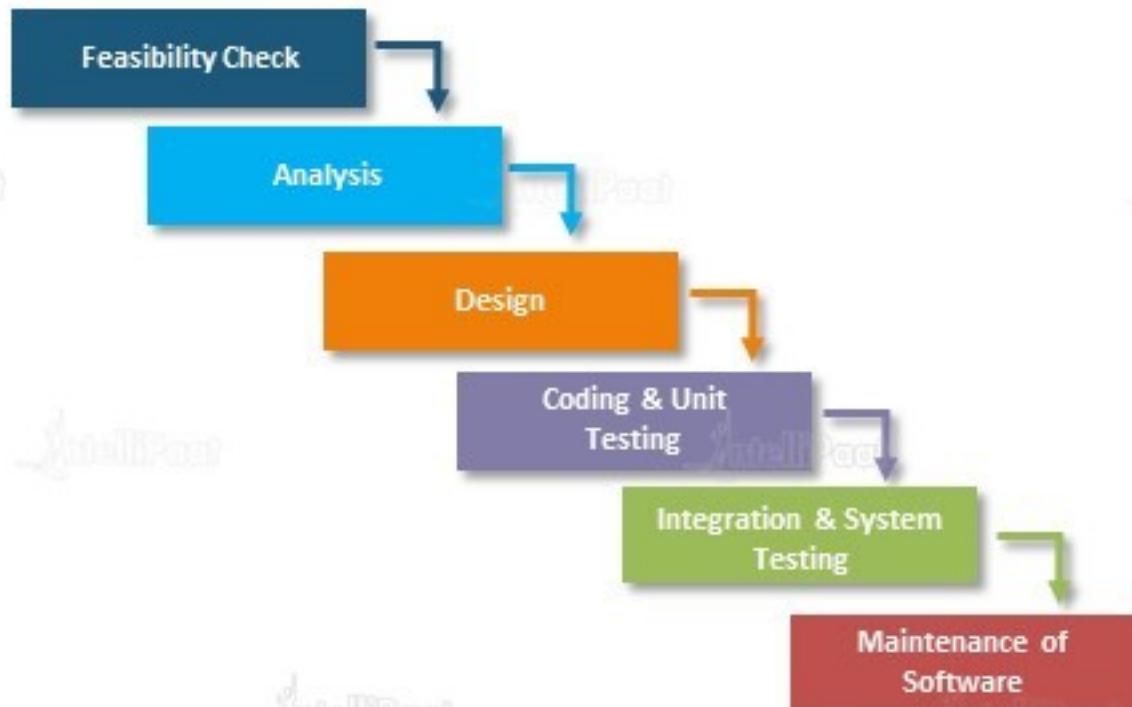
Waterfall Model

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

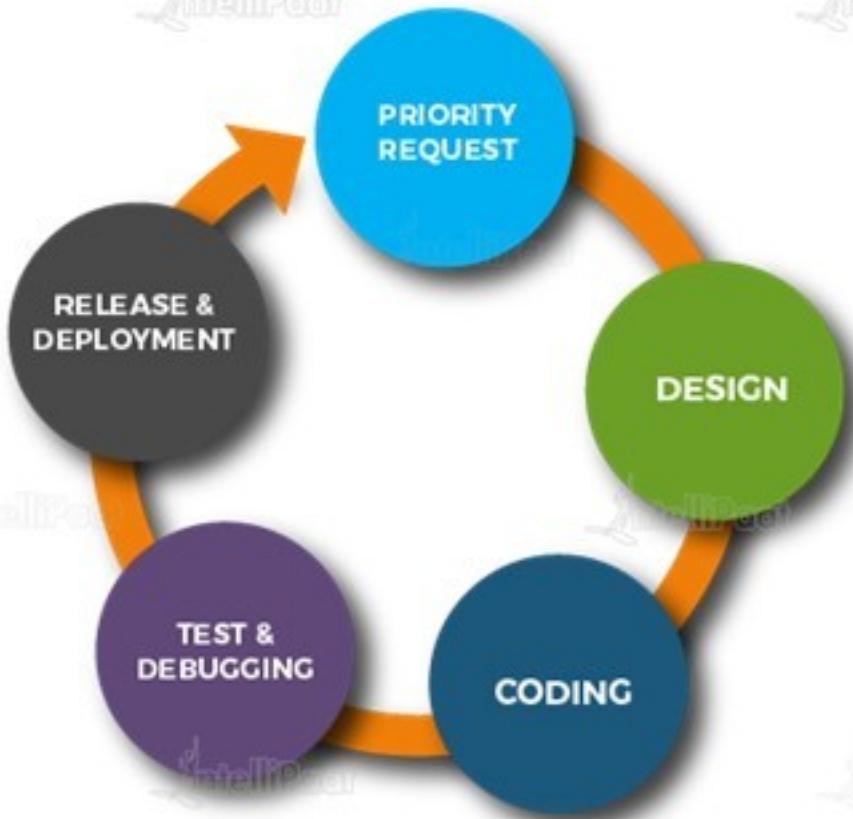
Disadvantages of Waterfall Model



- ✖ 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 its 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



- 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

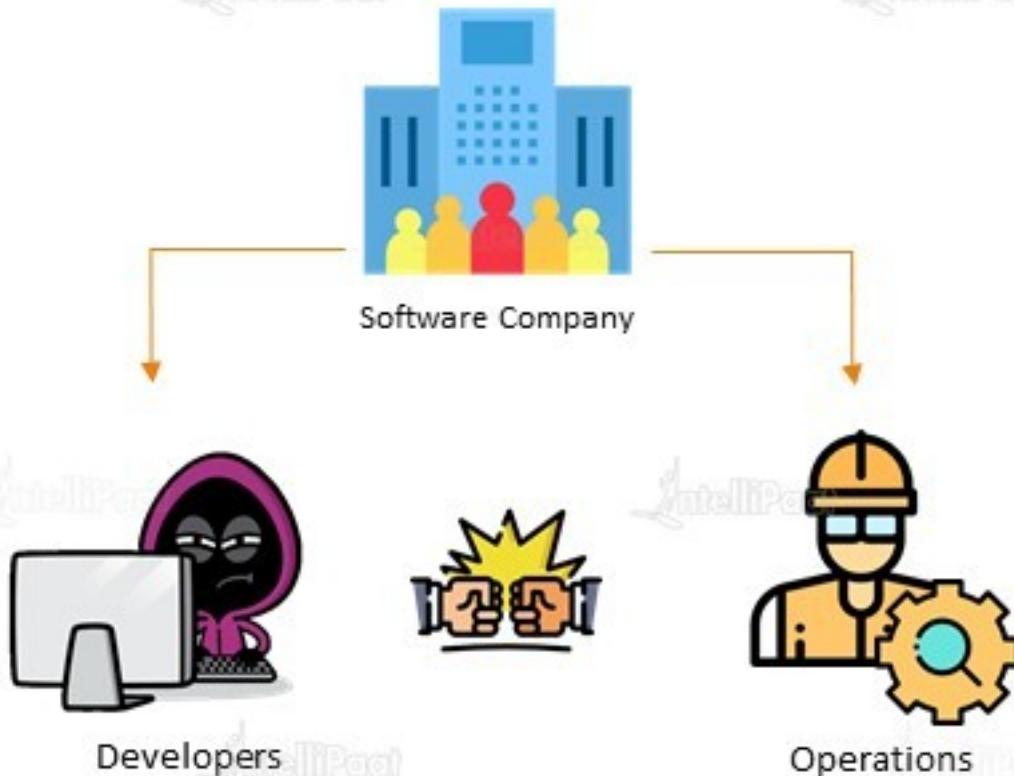


Software Company

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



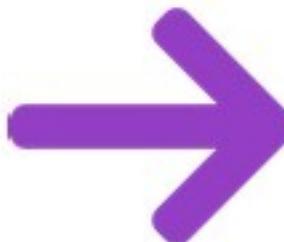
Operations

Why DevOps?



Developer

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



Operations

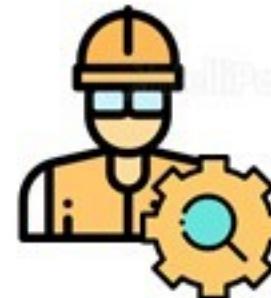
The operations when tried to run the code on their system, it did not run!

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

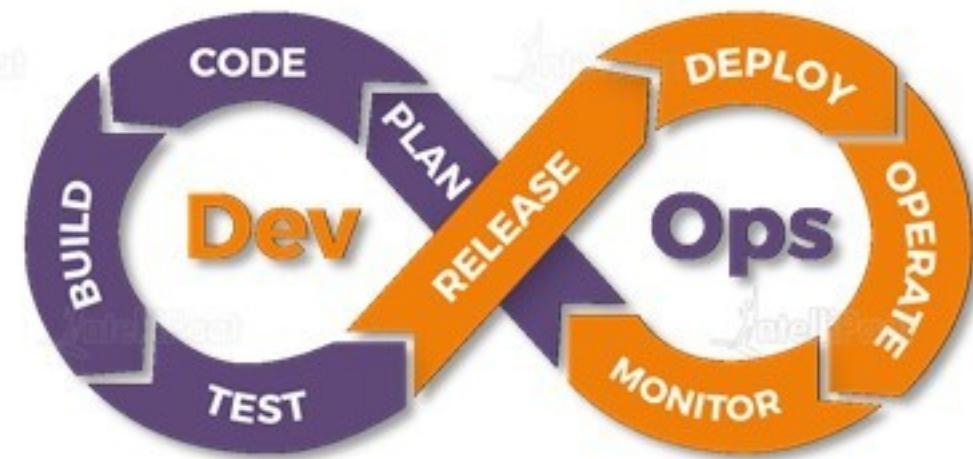
What is Devops?

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

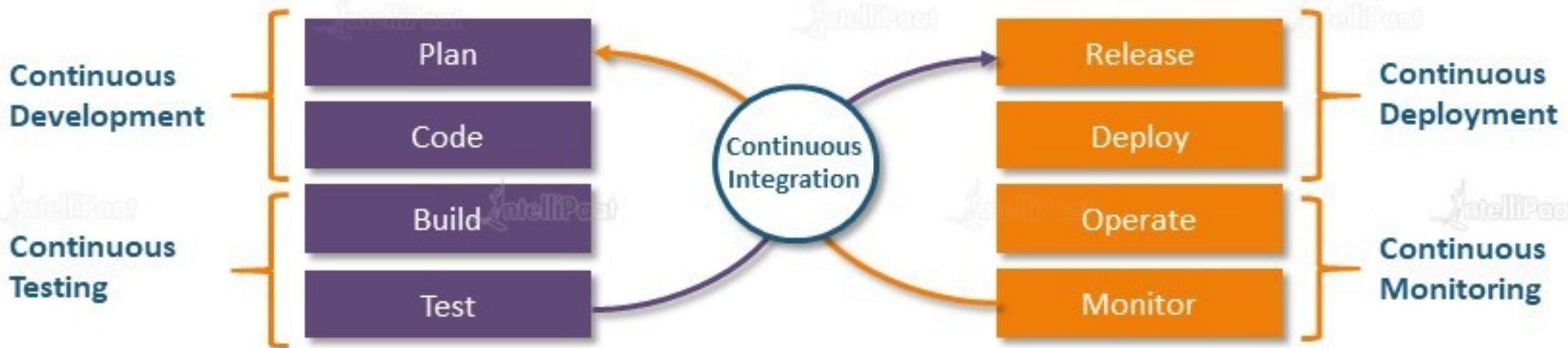


DevOps Lifecycle

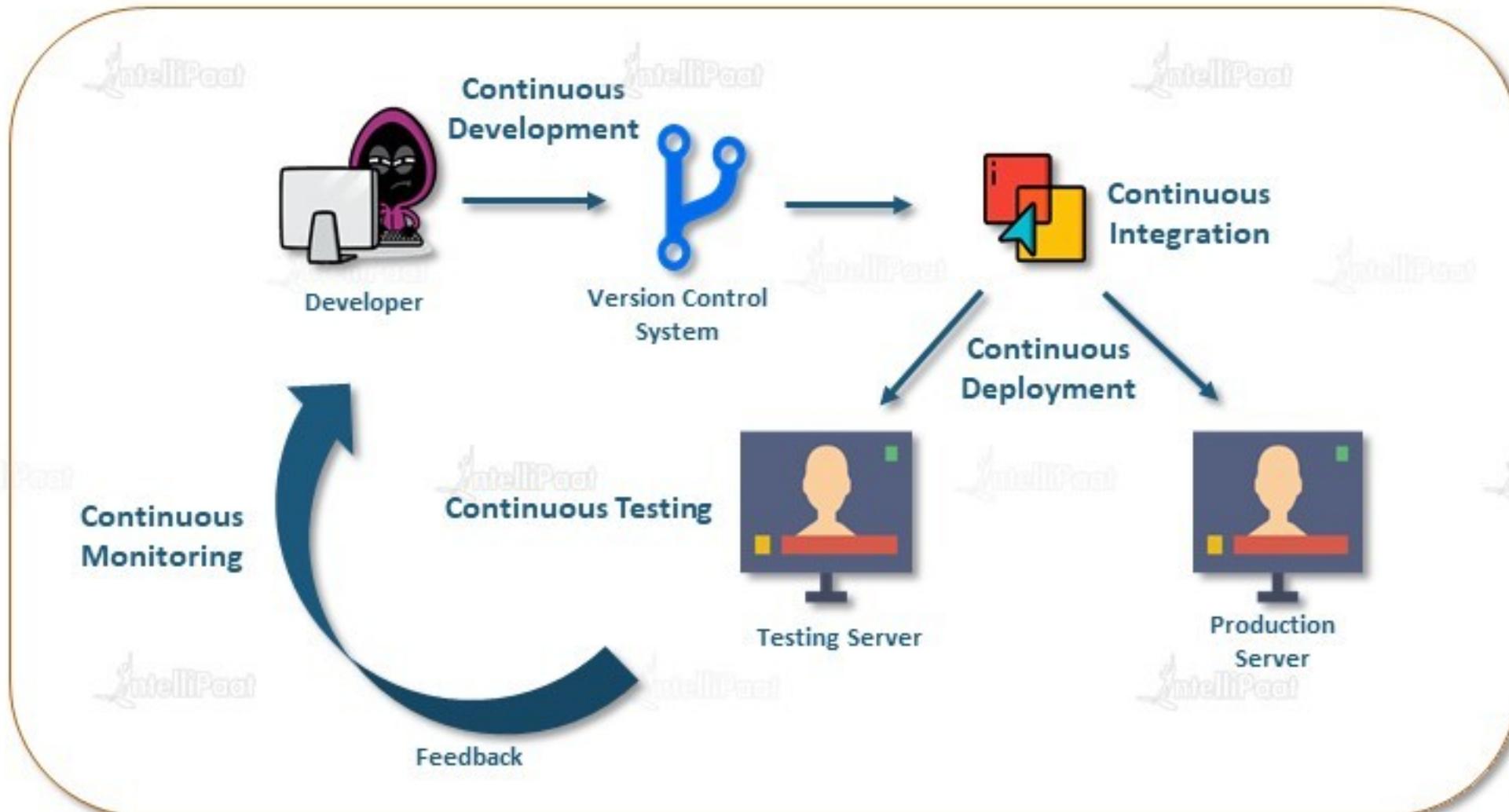


How DevOps Works?

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

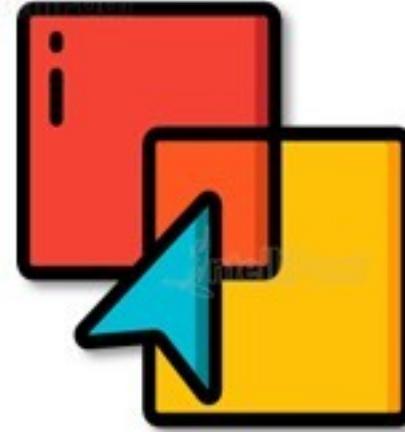
Continuous Integration

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

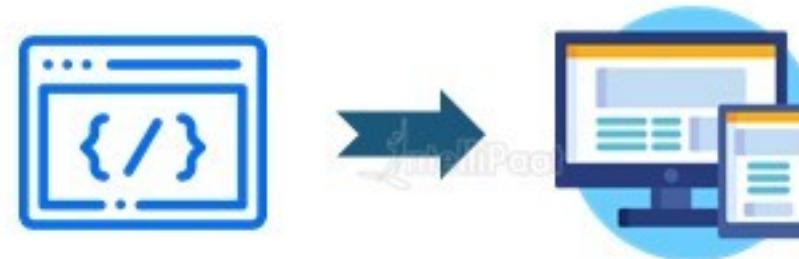
Continuous Integration

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

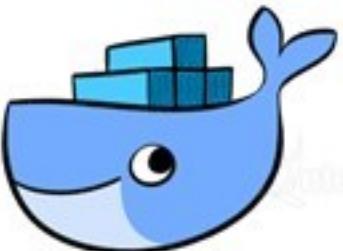


Devops Tools

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

Nagios®



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

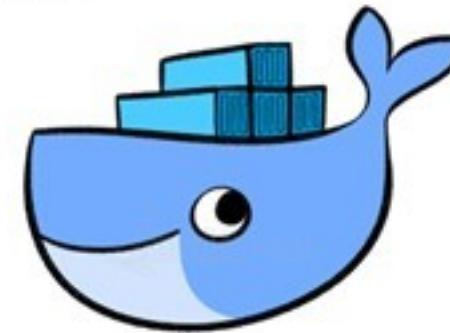
Continuous Deployment

Continuous Testing

Continuous Monitoring

Continuous Deployment

Virtualization &
Containerization



Configuration
Management



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

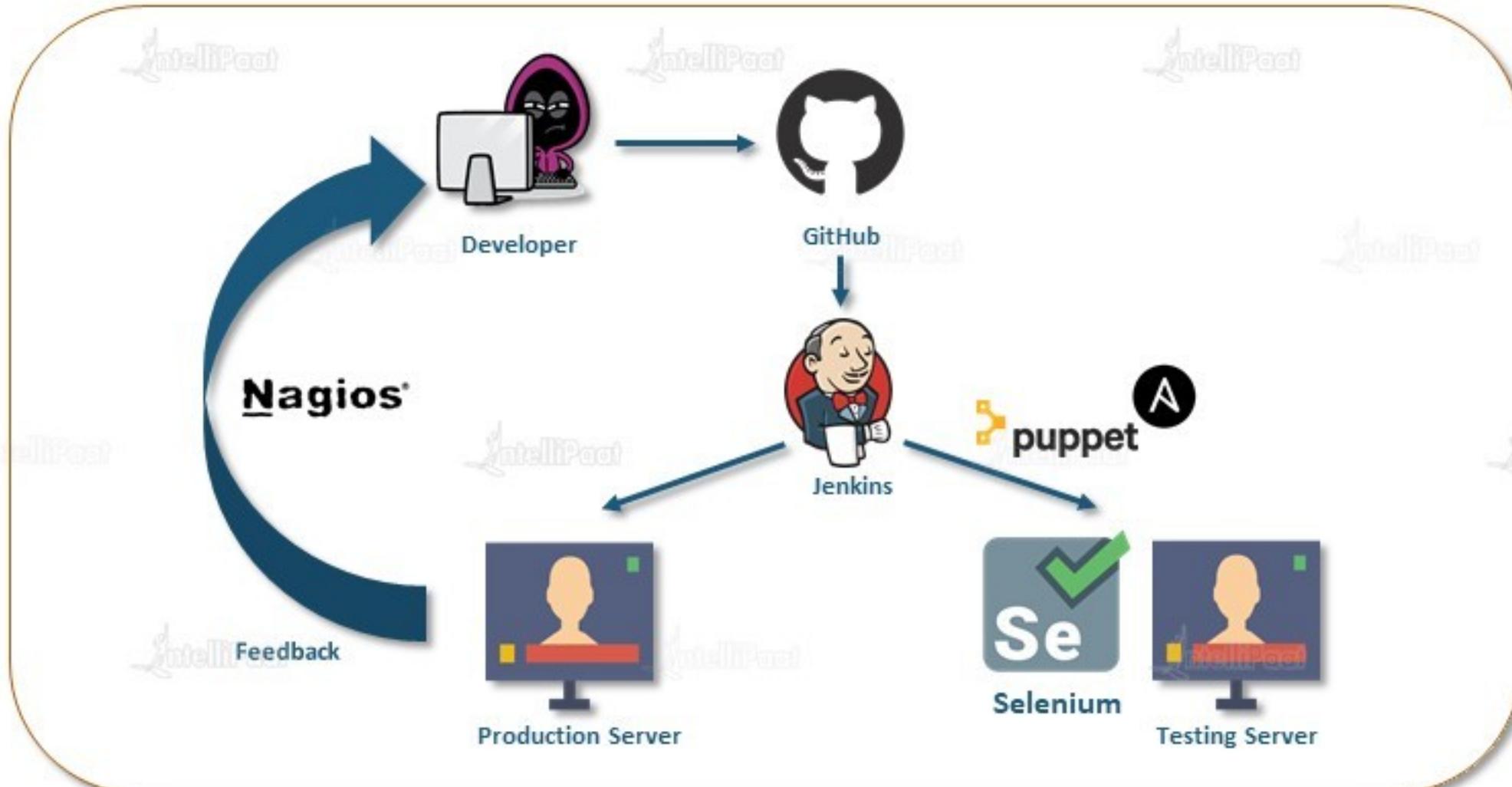
Continuous Testing

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.

Nagios®

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

Quiz

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

5. Which tool among the following helps in containerization?

A. Jenkins

B. Git

C. Kubernetes

D. Docker



India : +91-7847955955



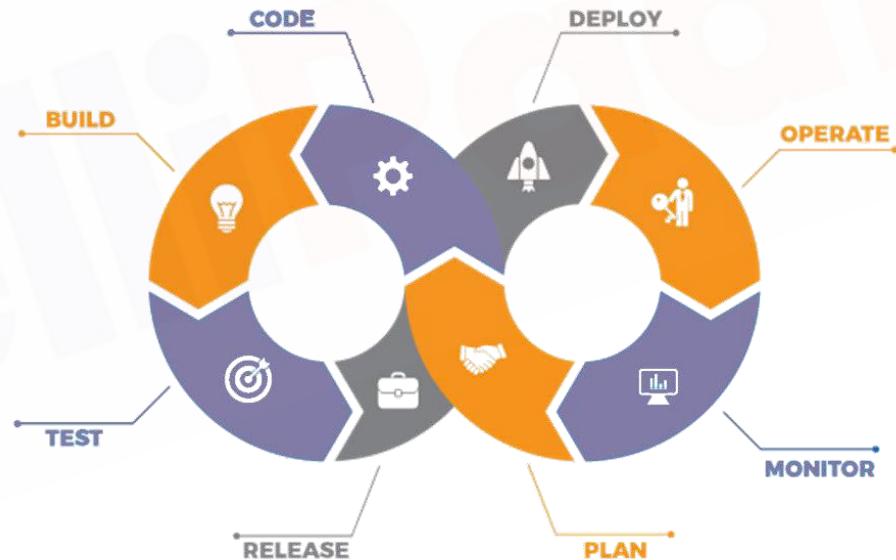
US : 1-800-216-8930 (TOLL FREE)



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.



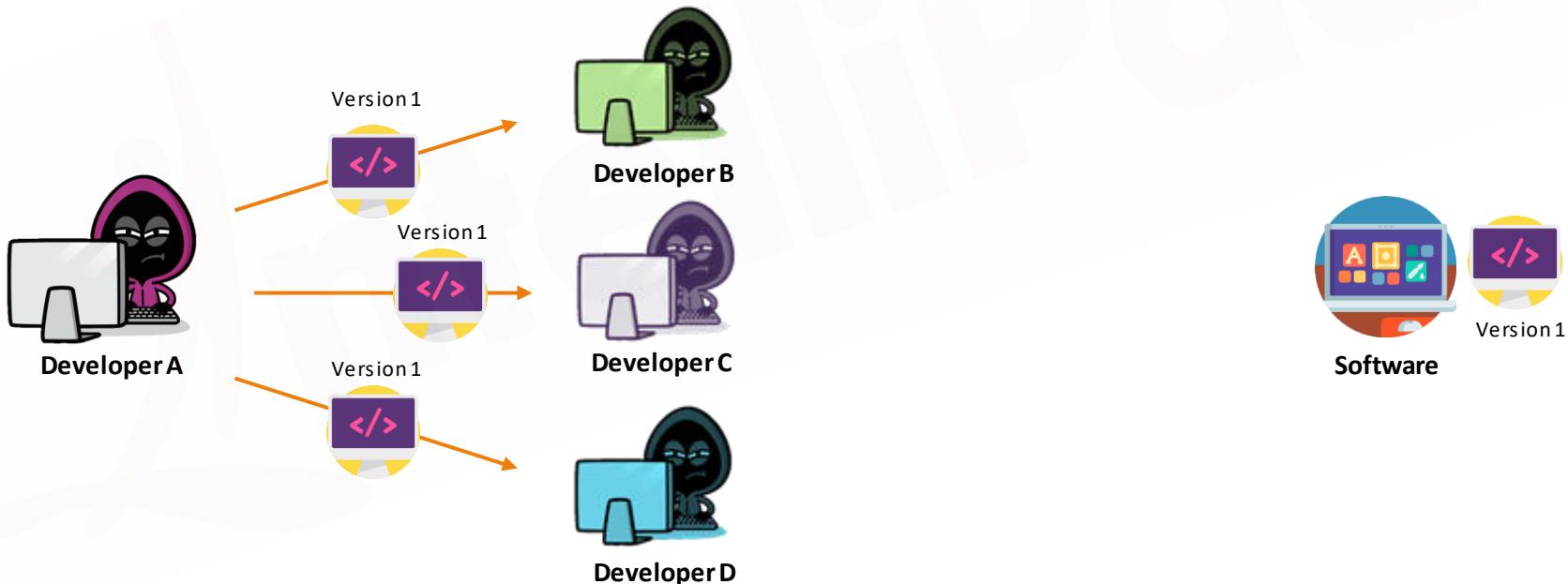
Developer A



Software

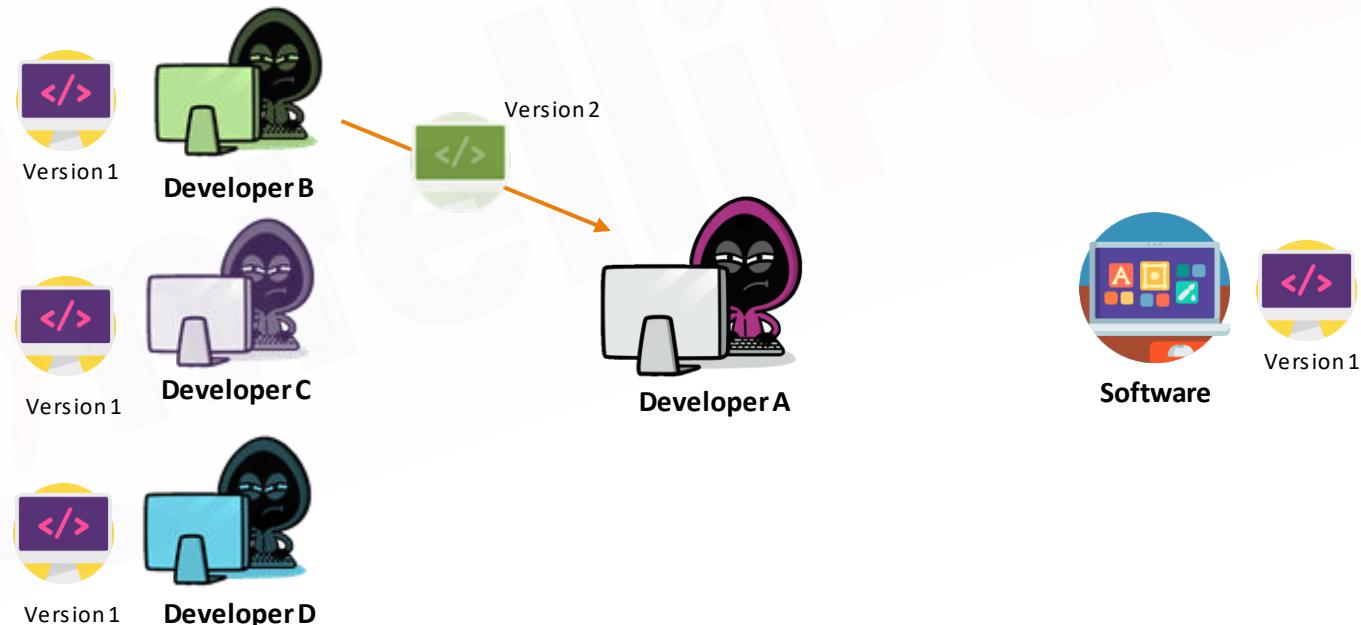
Problems before Version Control

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



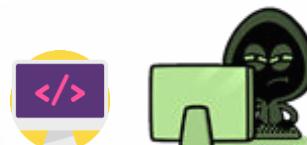
Problems before Version Control

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



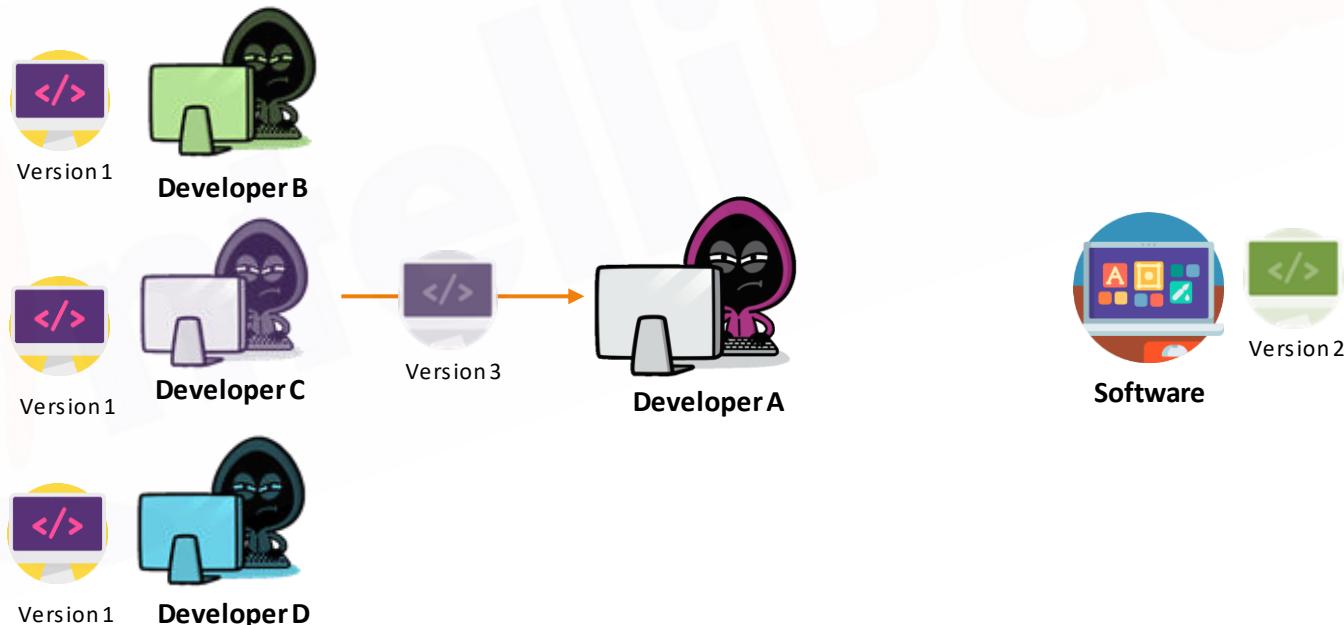
Software



Version 2

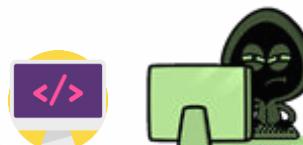
Problems before Version Control

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

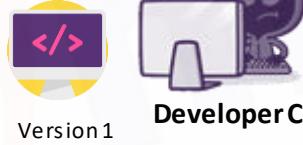
Developer A verifies the features, takes the code changes and manually integrates them with Version 2 code



Version 1



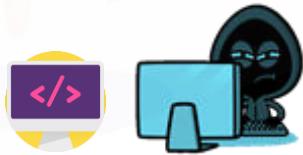
Developer B



Version 1



Developer C



Version 1



Developer D



Developer A



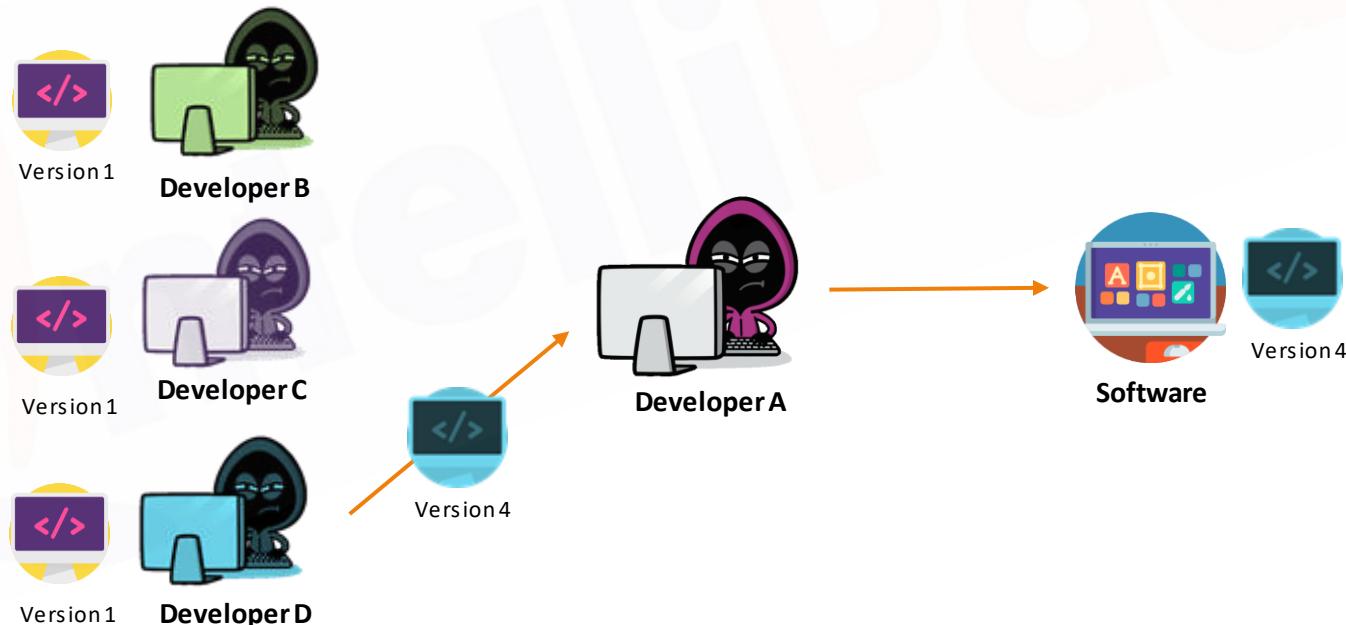
Software



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

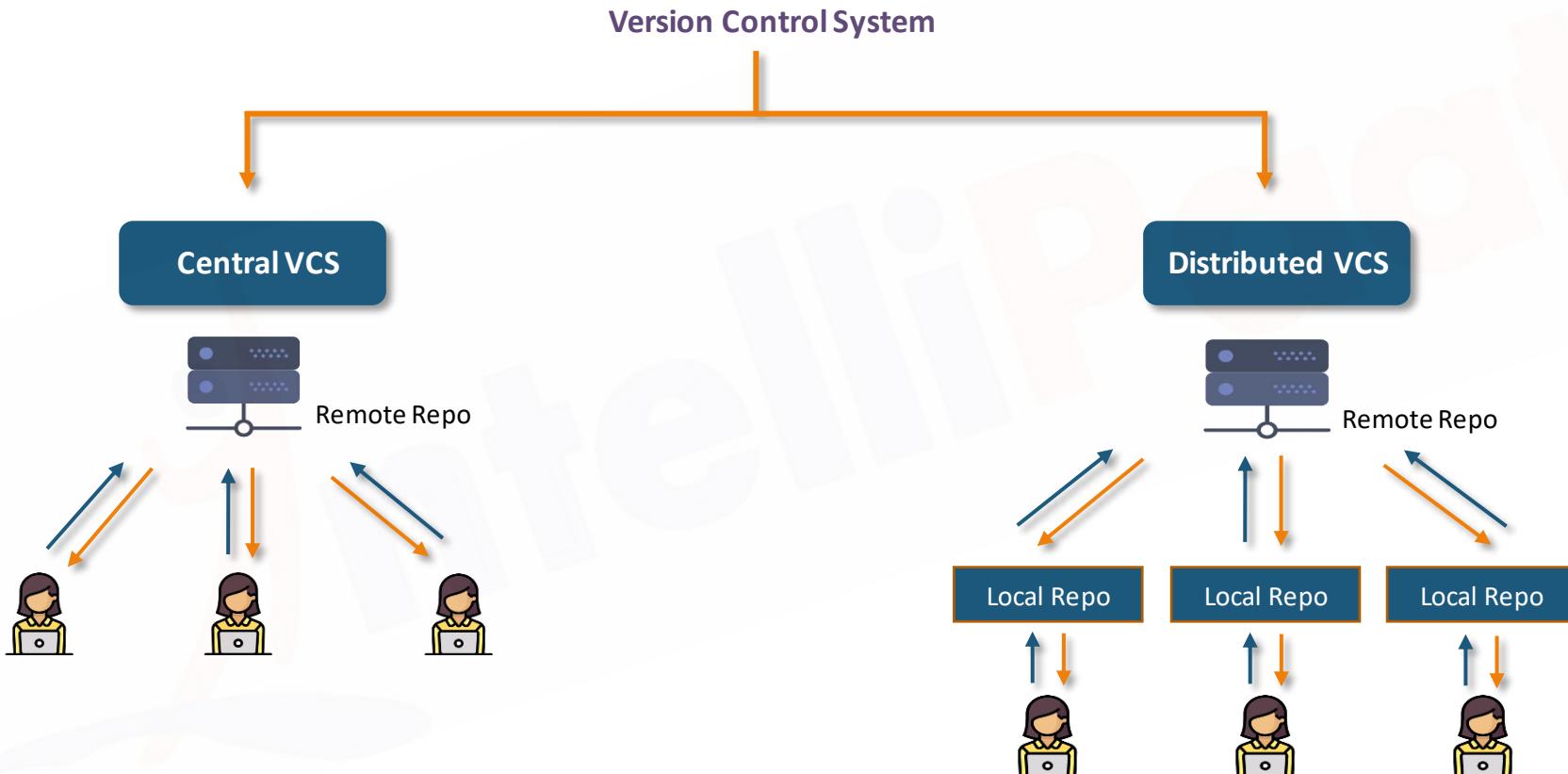
Advantages of Version Control



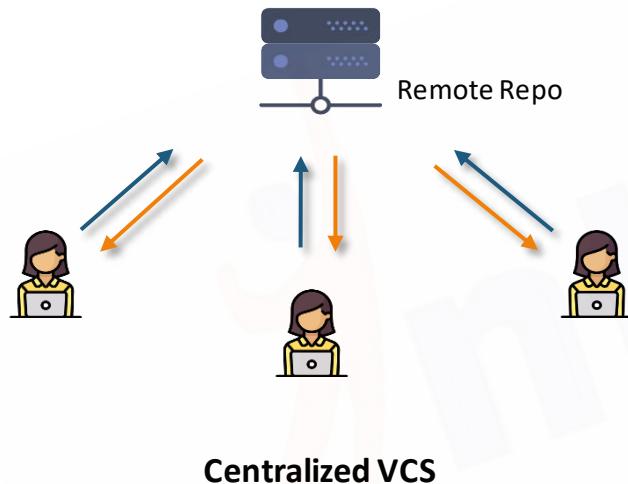
- ✓ Versioning is Automatic
- ✓ Team Collaboration is simple
- ✓ Easy Access to previous Versions
- ✓ Only modified code is stored across different versions, hence saves storage

Types of Version Control System

Types of Version Control System

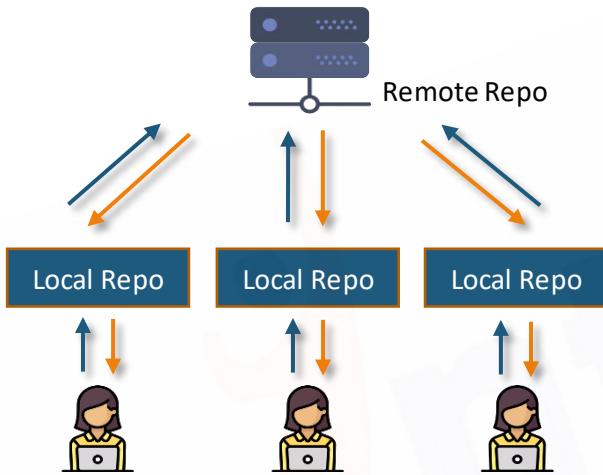


Centralized Version Control System



- ★ 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

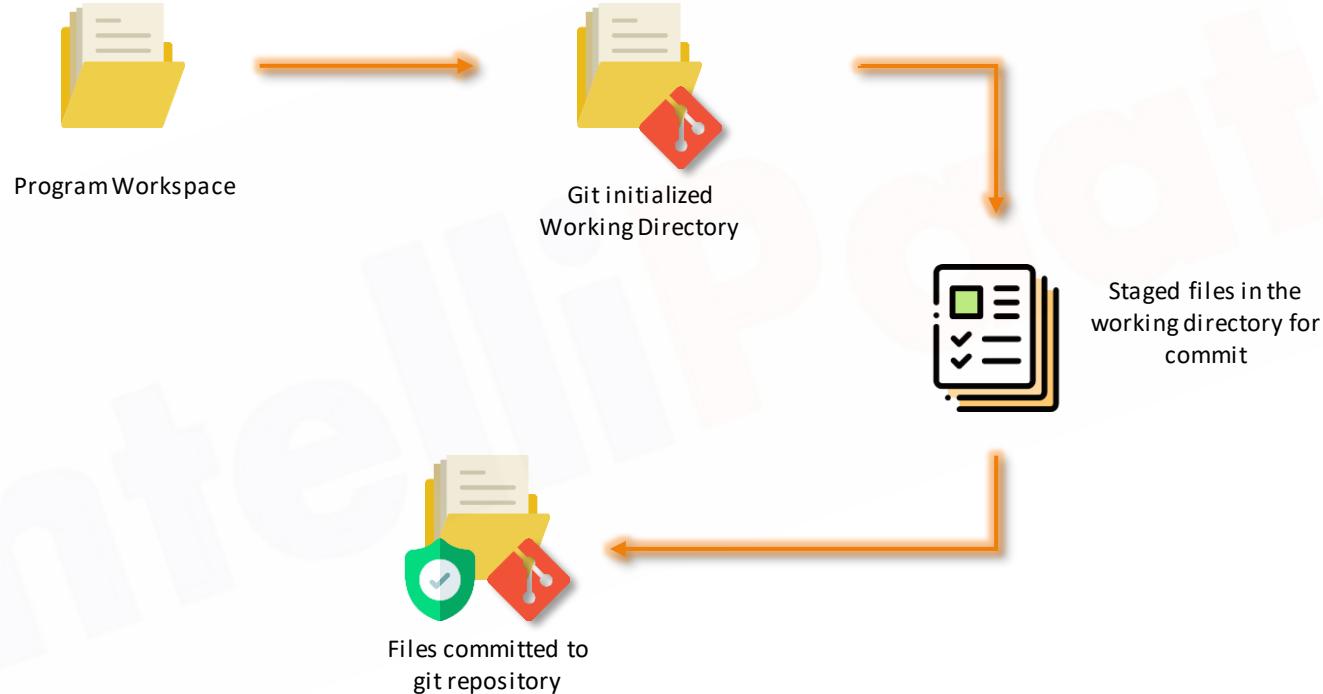
Working Directory

Staging Area

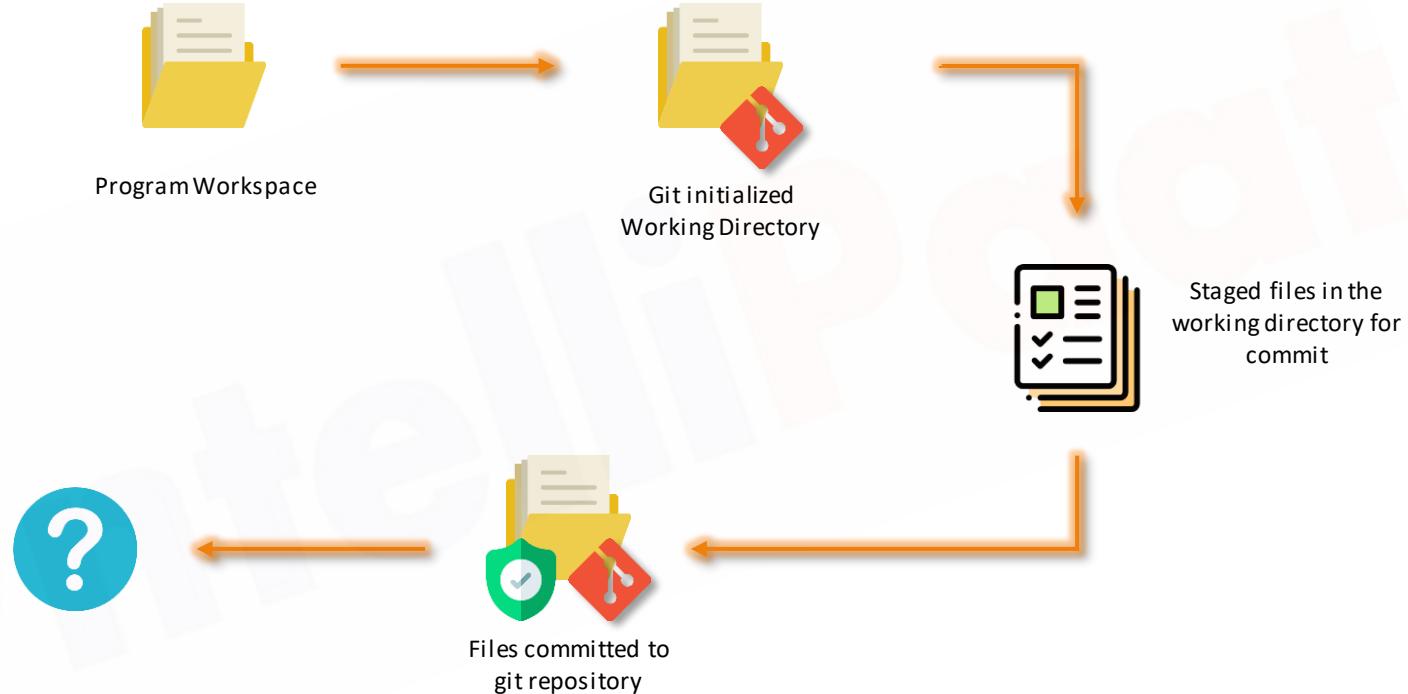
Commit

- ★ 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

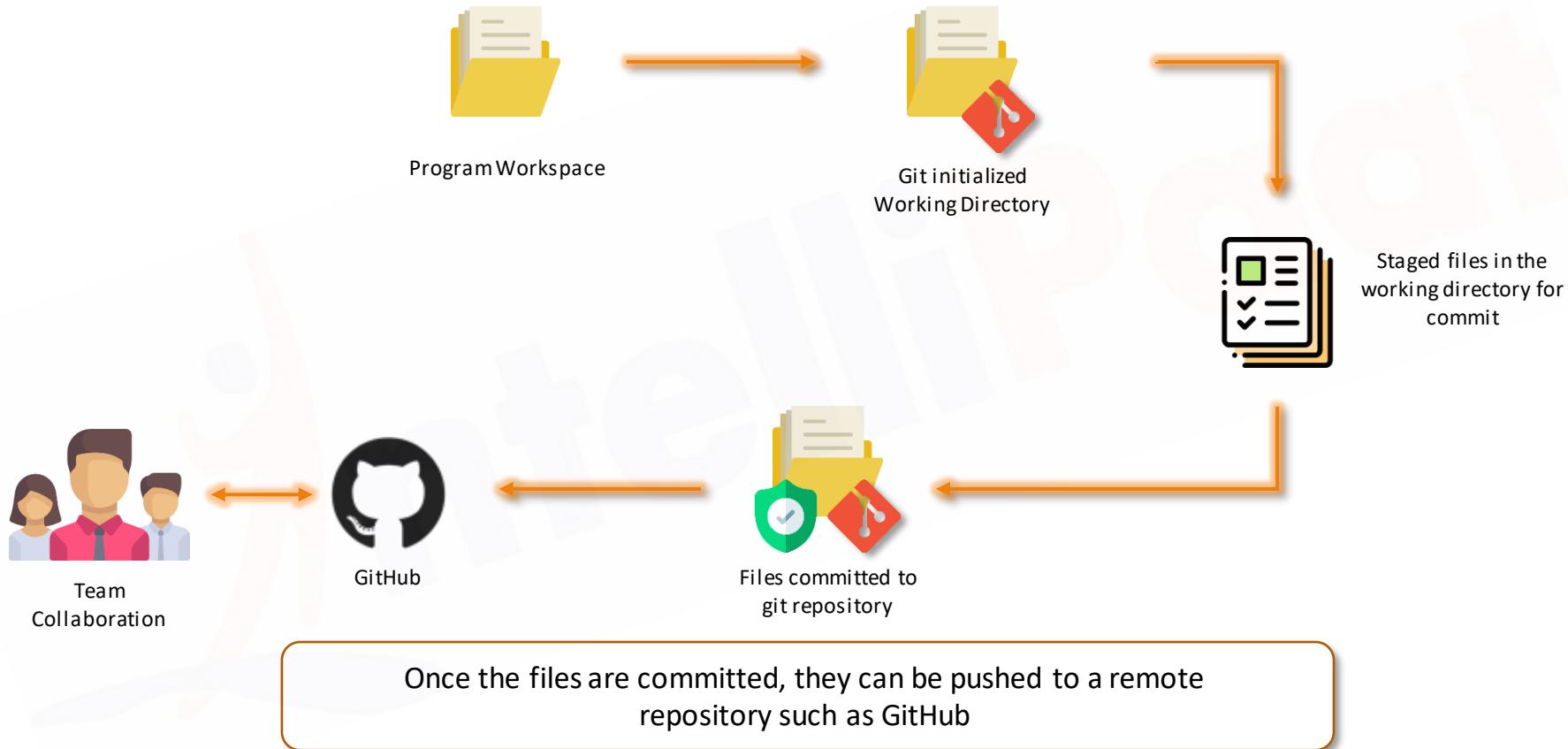


Git Lifecycle



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.



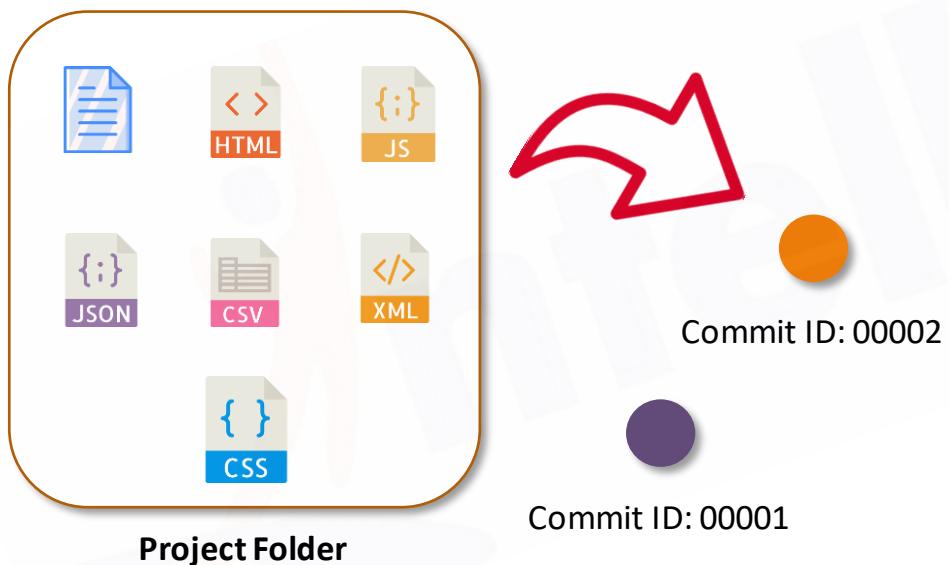
Project Folder



Commit ID: 00001

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

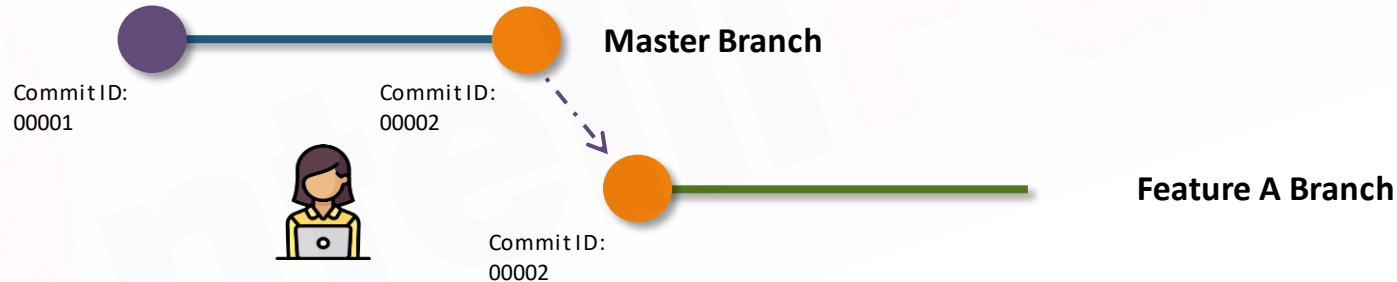


Master Branch

But, why do we need a branch?

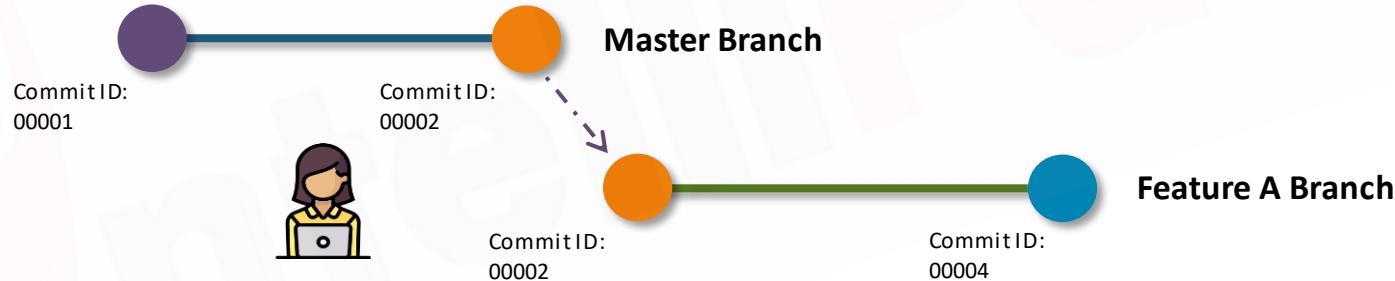
How does Git work?

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



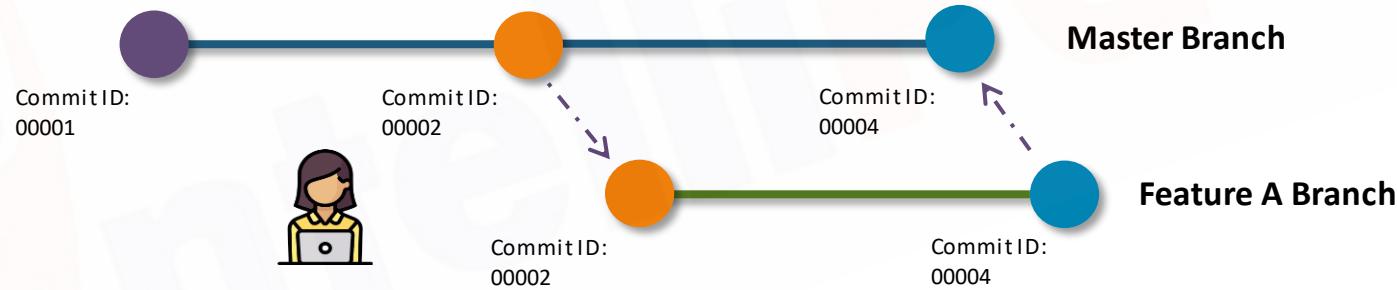
How does Git work?

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

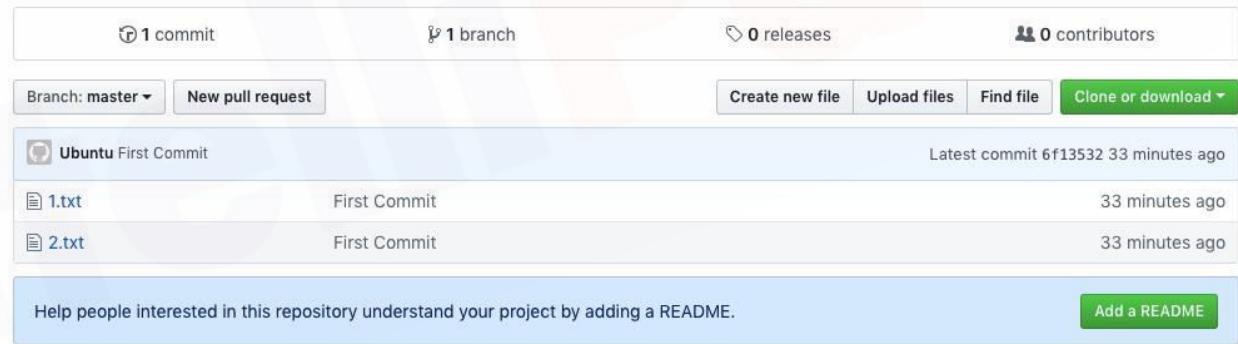


Syncing Repositories



Parallel Development

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



The screenshot shows a GitHub repository summary. At the top, it displays: 1 commit, 1 branch, 0 releases, and 0 contributors. Below this, 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. Two files, '1.txt' and '2.txt', are listed under this commit, both also made 33 minutes ago. At the bottom, there is a note: 'Help people interested in this repository understand your project by adding a README.' followed by a green 'Add a README' button.

File	Commit Message	Time Ago
1.txt	First Commit	33 minutes ago
2.txt	First Commit	33 minutes ago

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:~/devops]$ 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

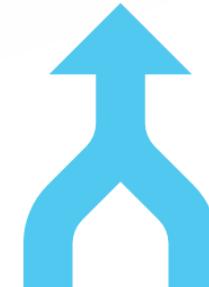
Once the developer has finished his code/feature on his branch, the code will have to be combined with the master branch. This can be done using two ways:



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

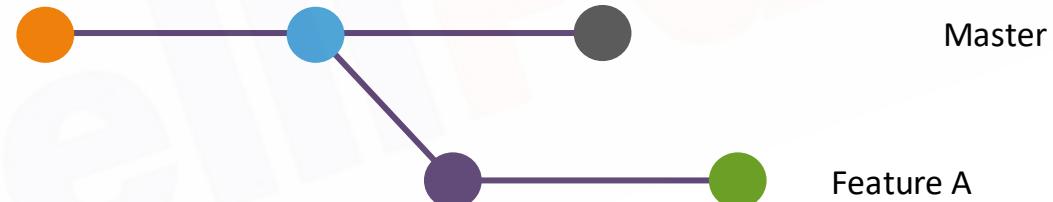


Git Merge



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



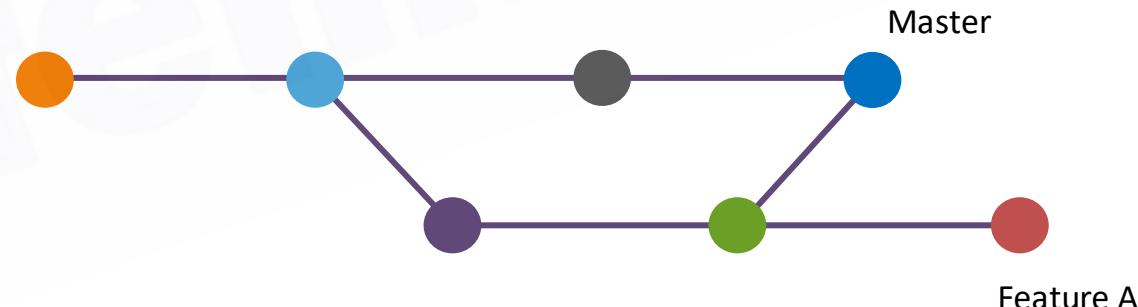
Git Merge



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
*   d92f22eeb6bb7fefd1706b397abe804dc557ec88 (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

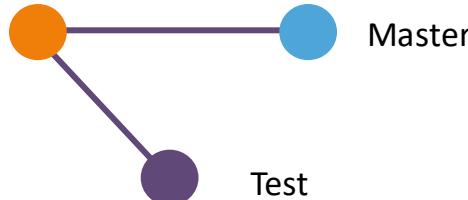


Git Merge



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

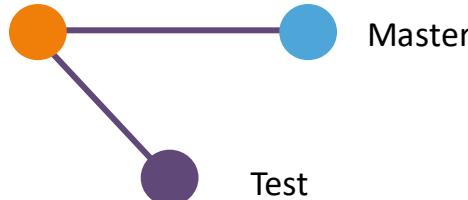


Therefore you want all the changes from master in feature.



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



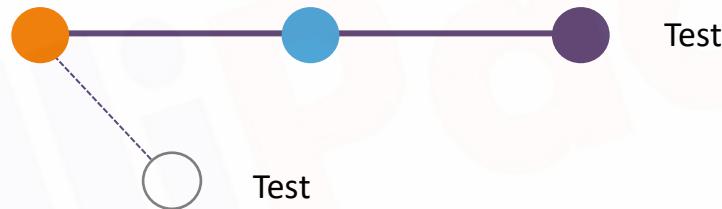
Git Merge



Git Rebase



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



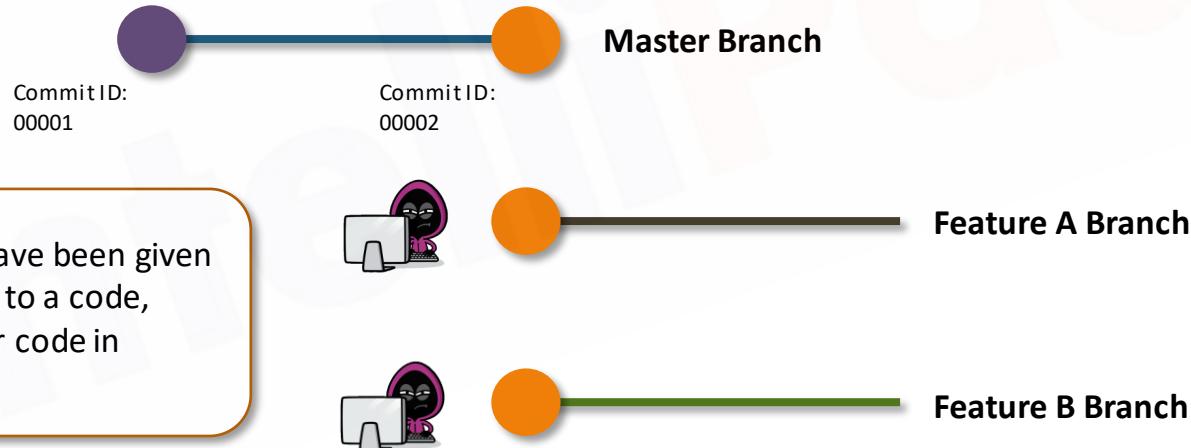
And looking at the history we can clearly see, it's a clean linear history, without any branches

```
[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,

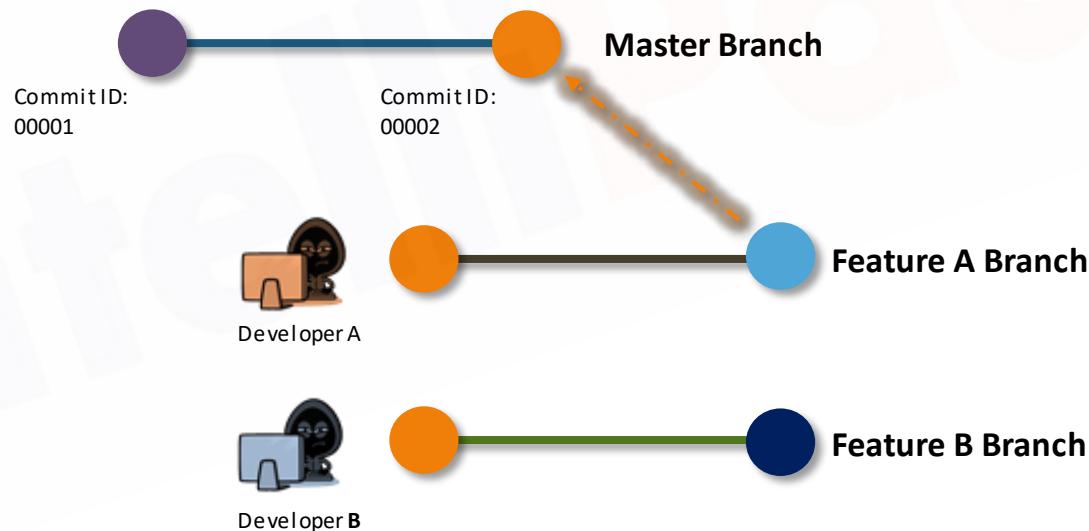
```
Main()
{
    Function1()
    {
        //InitialCode
    }

}
```

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

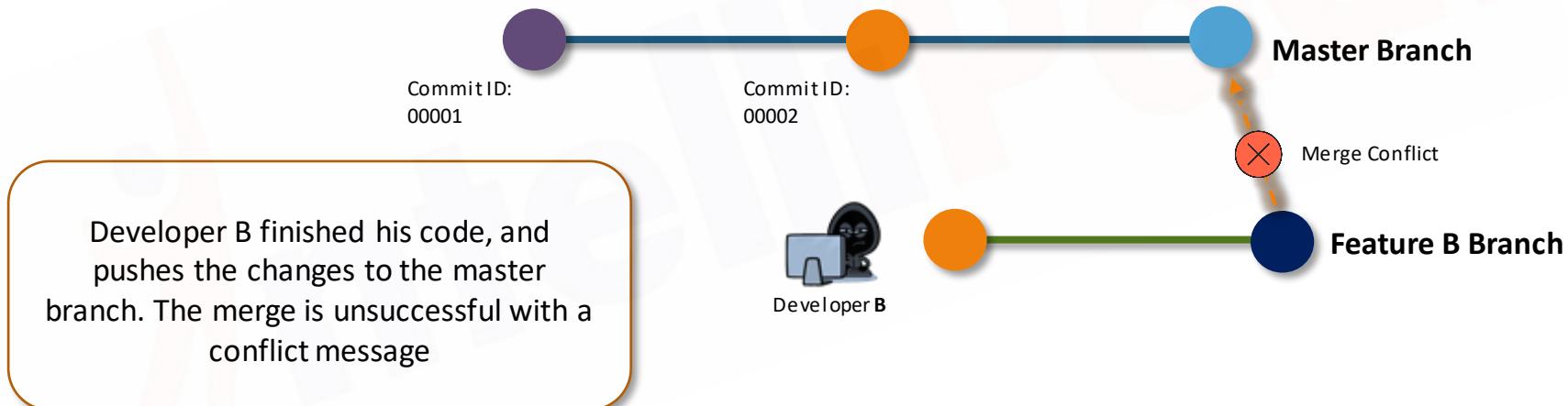
```
Main()
{
Function1()
{
    //Initial Code
}
Function3()
{
    //Developer B Code
}

}
```

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

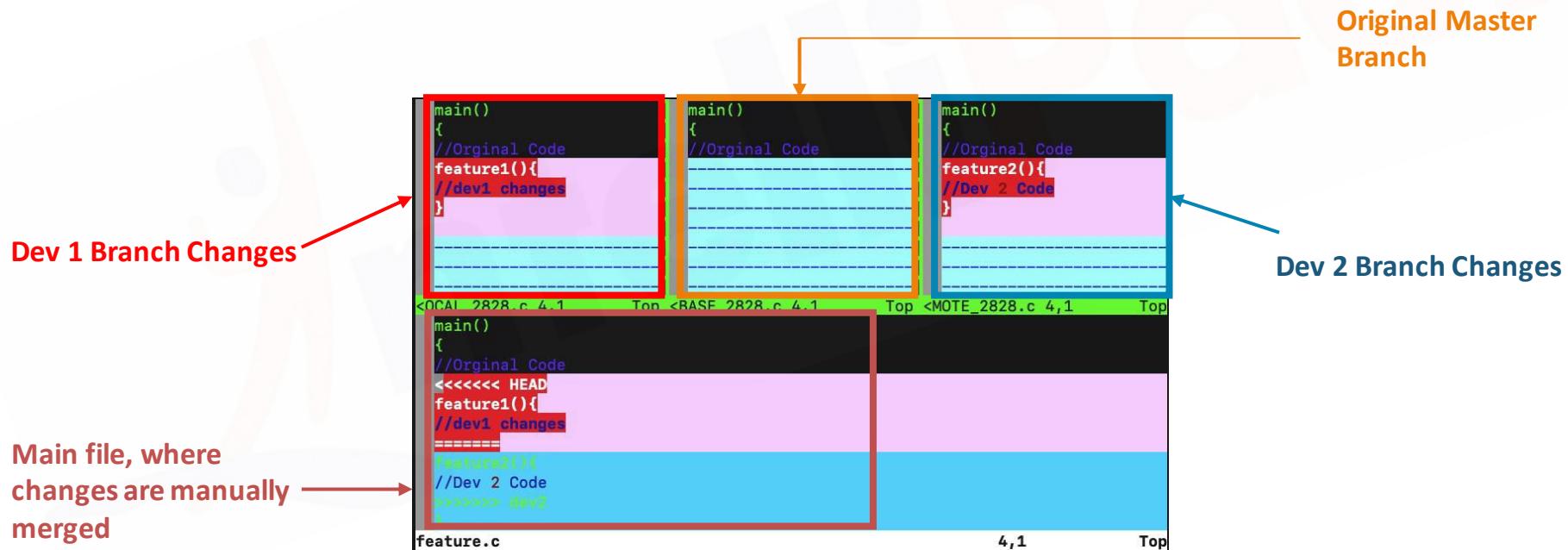
This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
tortoisemerge emerge vimdiff
Merging:
feature.c

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.



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

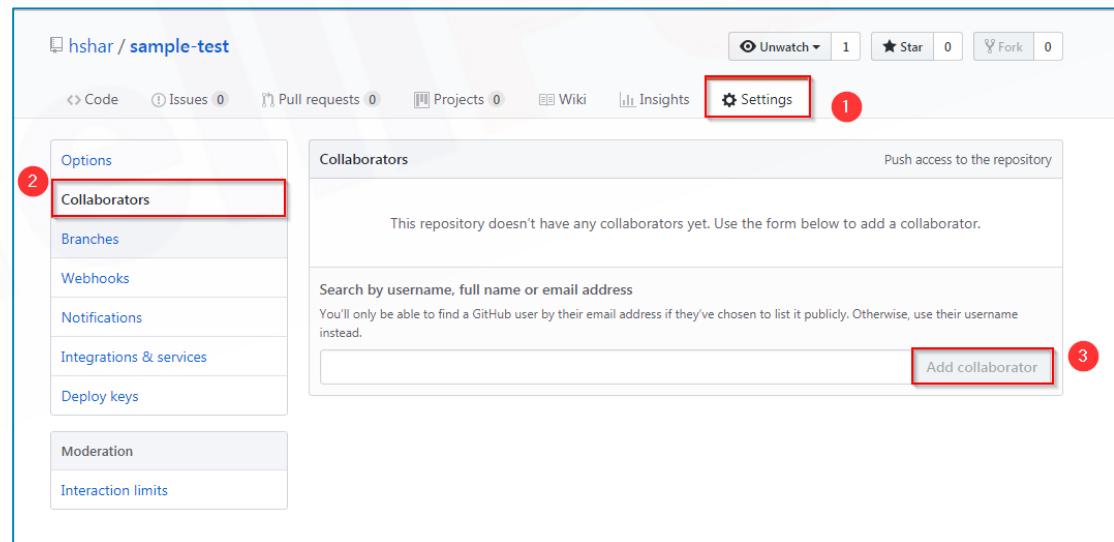


Collaborators



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:



hshar / sample-test

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings 1

Options 2 Collaborators

Branches

Webhooks

Notifications

Integrations & services

Deploy keys

Moderation

Interaction limits

Collaborators Push access to the repository

This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

Search by username, full name or email address

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.

Add collaborator 3

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



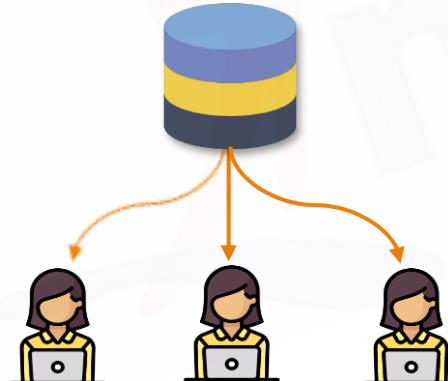
Git Workflow

Git Workflow

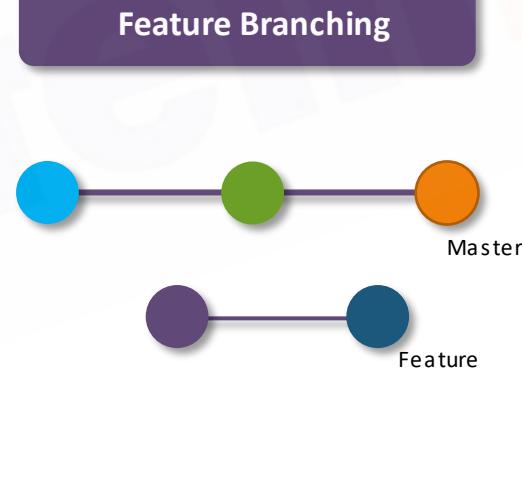
A Git Workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner. Git workflows encourage users to leverage Git effectively and consistently.

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

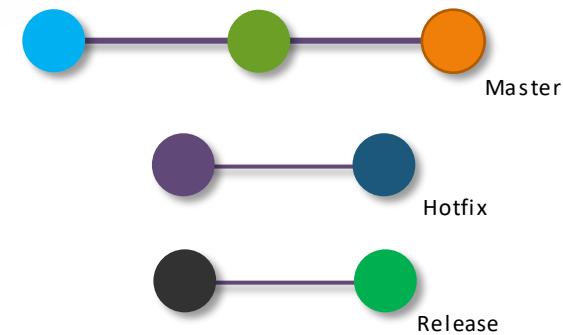
Centralized Workflow



Feature Branching



GitFlow Workflow



Git Workflow



Centralized Workflow

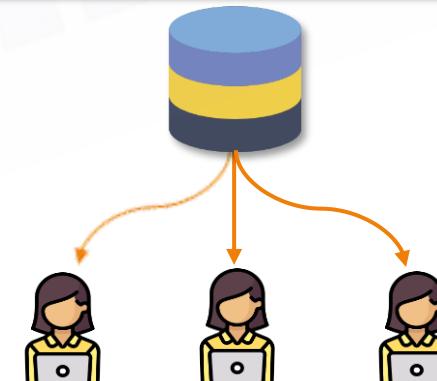
- ★ This workflow doesn't require any other branch other than master
- ★ All the changes are directly made on the master, and finally merged on the remote master, once work is finished
- ★ Before pushing changes, the master is rebased with the remote commits
- ★ Results in a clean, linear history



Feature Branching



GitFlow Workflow



Git Workflow



Centralized Workflow

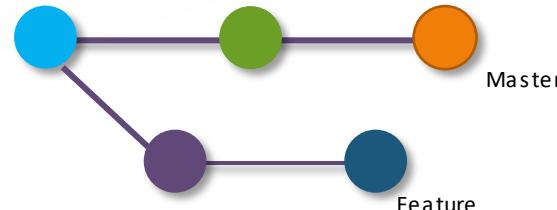


Feature Branching



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

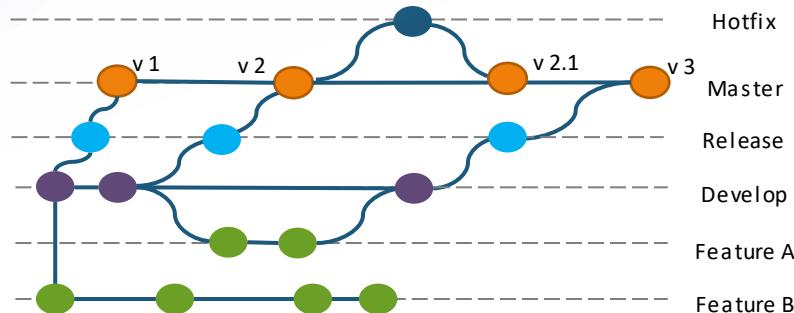


Feature Branching



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

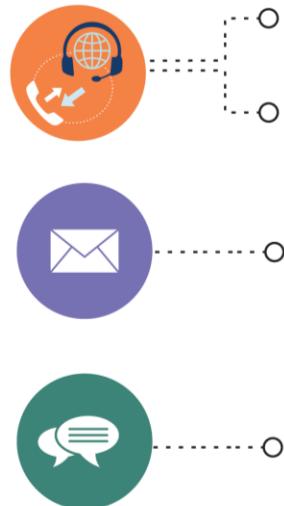
5. Merge Conflicts occur when,

- 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

Quiz

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)

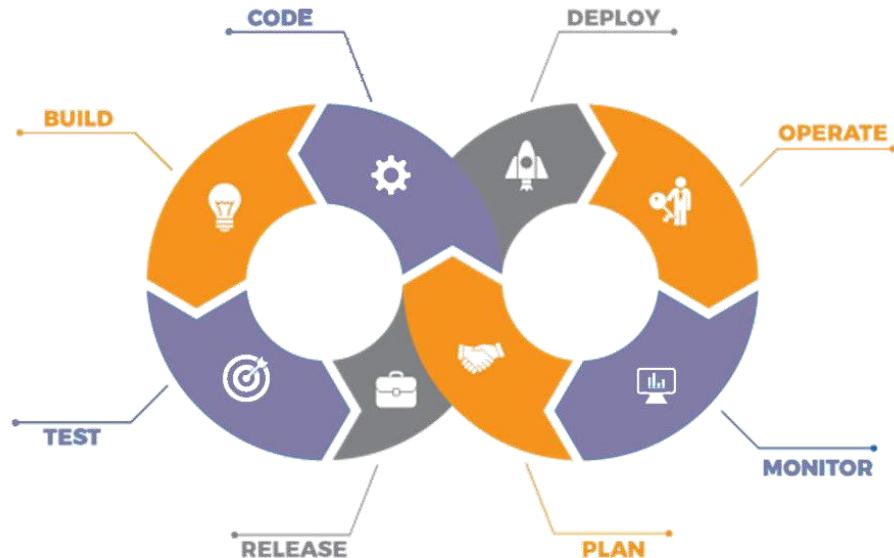
sales@intellipaat.com

24X7 Chat with our Course Advisor



Continuous Integration Using Jenkins

&
Maven™



Agenda

01 Why Maven?

02 What is Maven?

03 What does Maven do?

04 What is a POM file?

05 Maven Life Cycles

06 Introduction to Jenkins

07 Maven Installation

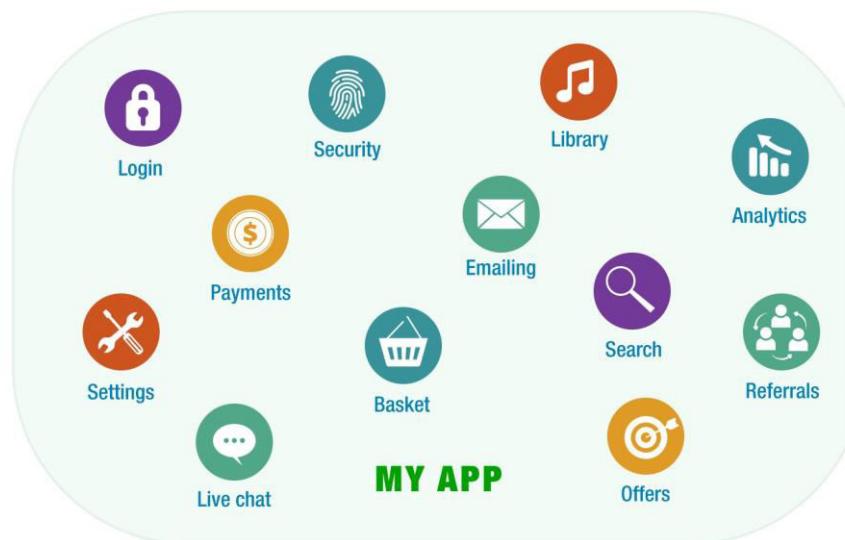
08 Maven Hands-on



Why Maven?

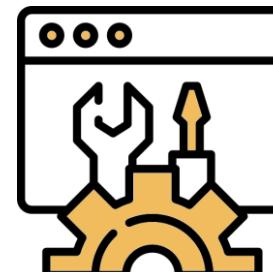
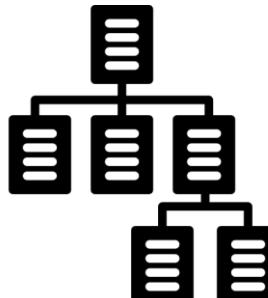
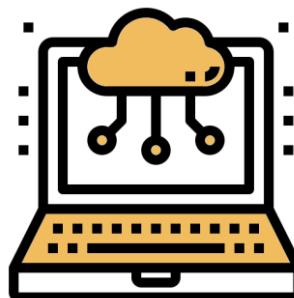
Why Maven?

The written code by developers has to be packaged and converted into a executable format, this is done with the help of Maven



Using Maven

Since Maven is a build tool, it can help in building an application that has several different modules.
It helps doing so by:



Downloading
dependencies

Enforcing a
directory structure

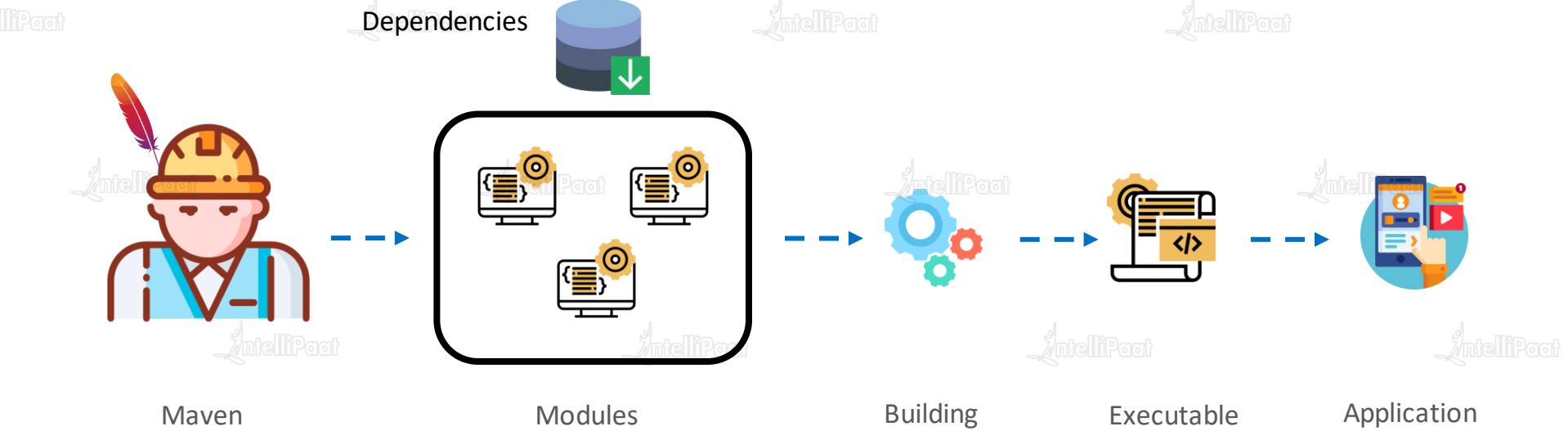
Building an executable with
all the dependencies

What is Maven?



What is Maven?

Maven is commonly referred to as a build tool that is used to manage the entire life cycle of a project, generate reports, and store documents with its POM repository



Maven

Modules

Building

Executable

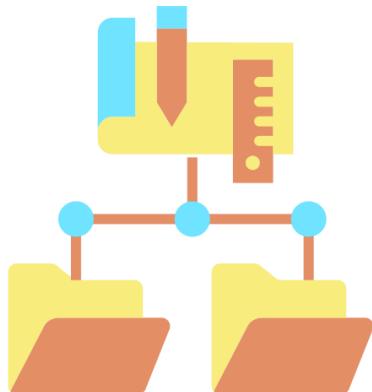
Application



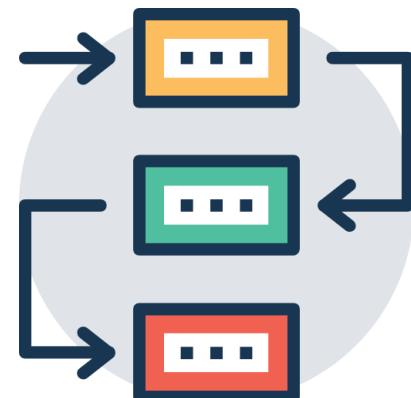
What does Maven do?

What does Maven do?

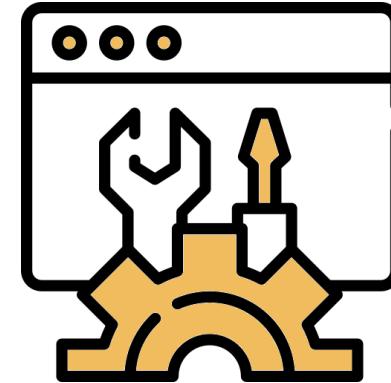
Enforce a Directory Structure

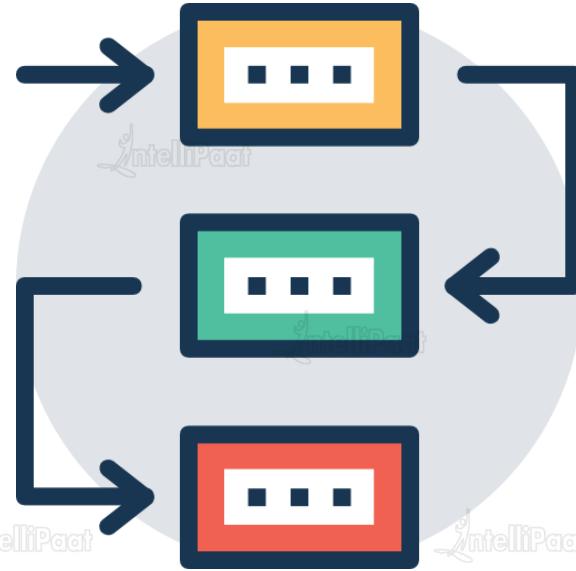


Manage and Download Project Dependencies



Build an executable for the code

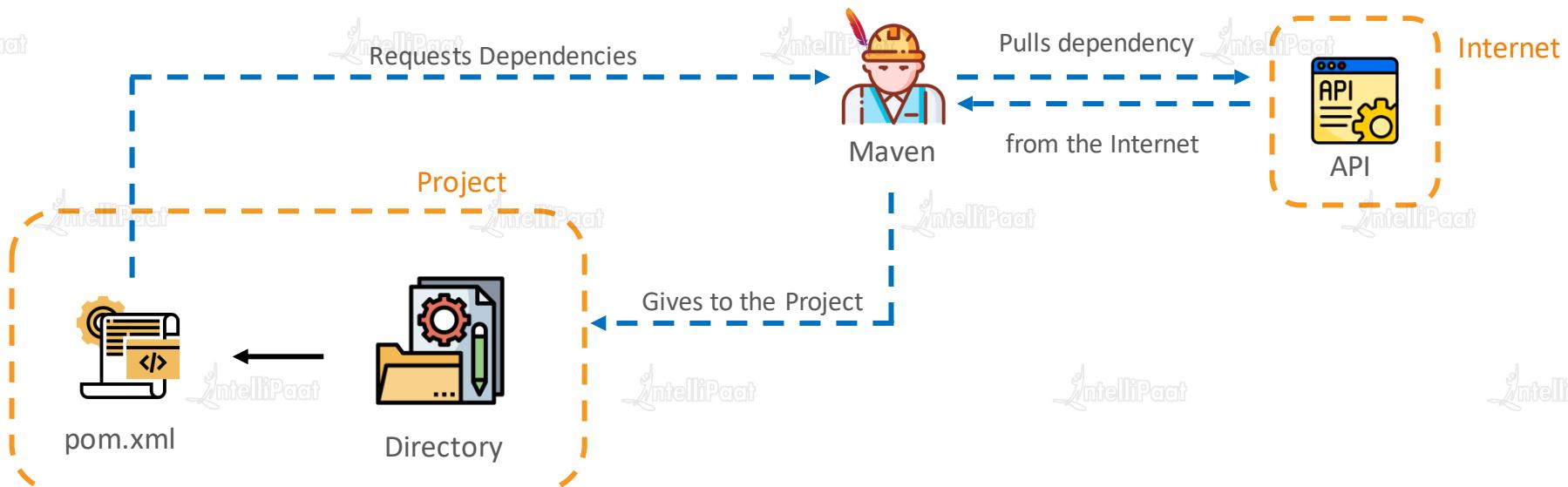




Managing and Downloading Project Dependencies

Project Dependencies

Projects may need Java APIs or frameworks that are packaged in their own JAR files. These JAR files are needed in the class path when a project code is compiled





Building POM Files

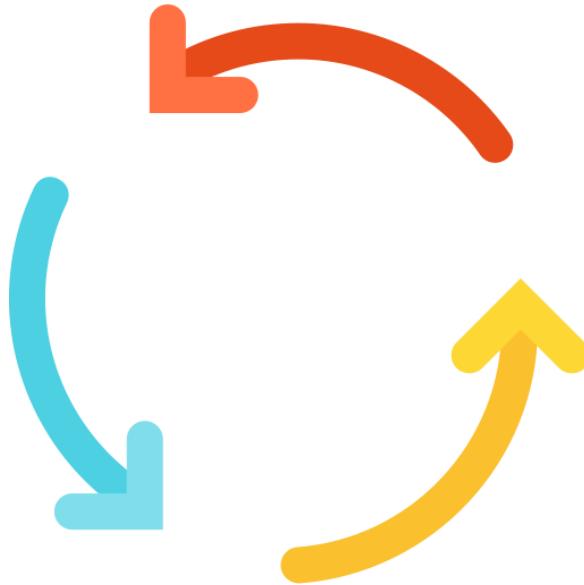
POM (Project Object Model) File



Every Maven project directory needs a **pom.xml** file. These files contain all details necessary for Maven to effectively execute a project. The POM file is stored in 'src' or in the root directory

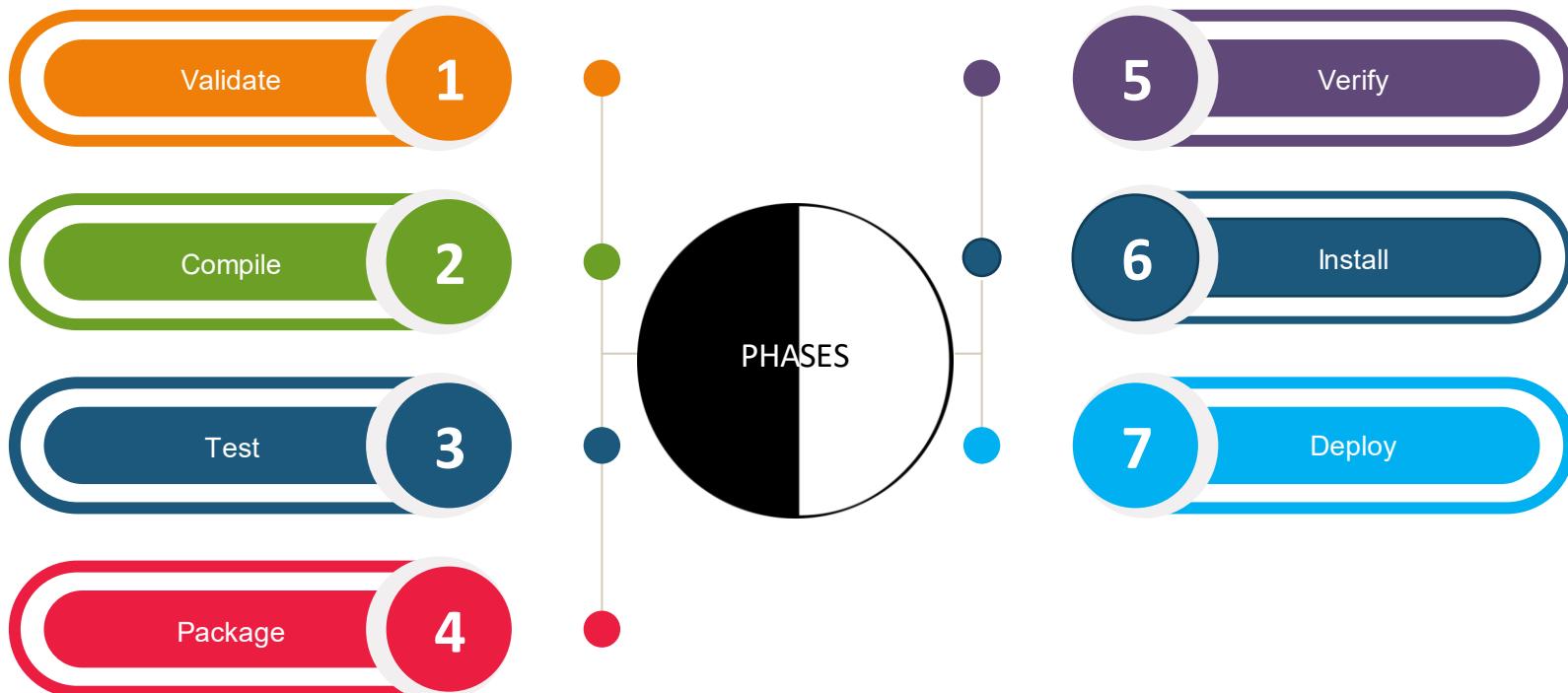
A screenshot of a code editor window titled "pom - Dependency example.xml". The window has a dark theme with light-colored text. The code is a snippet of XML defining a dependency for JUnit 4.8.2 with a scope of "testing".

```
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.8.2</version>
<scope>testing</scope>
</dependency>
</dependencies>
```



Maven Build Life Cycle

Maven Build Life Cycle



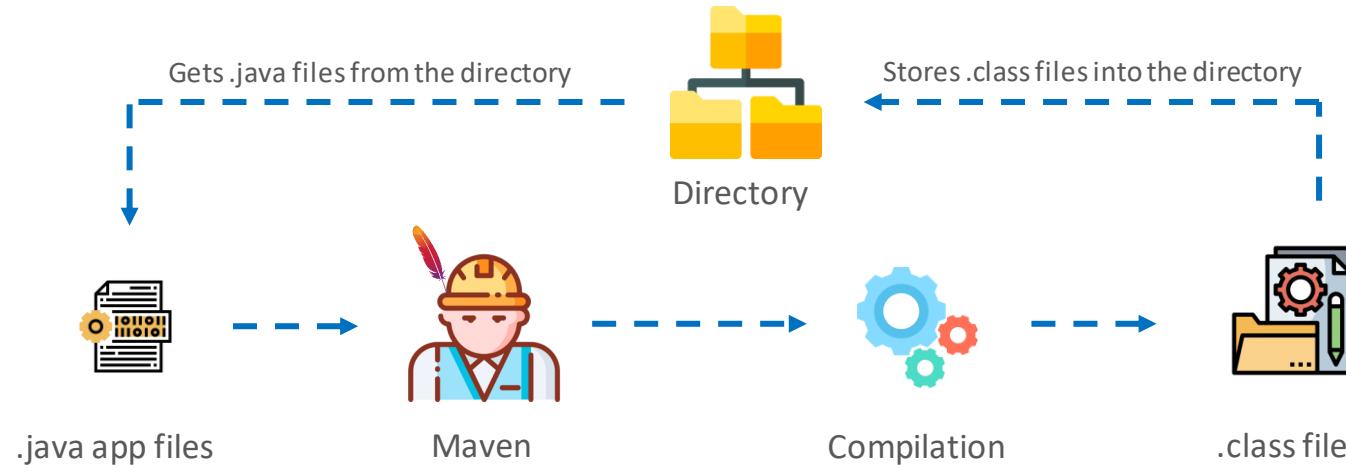
1. Validate Phase

validate the project is correct and all necessary information is available



2. Compile Phase

During the compile phase, Maven will compile all **.java** files present in the main directory into **.class** files and put them back into the main directory in the dedicated folder



3. Test Phase



During the test phase, Maven will execute the specified test cases and create a summary log



Test Cases

Maven

Testing

Results

4. Package Phase

During the package phase, Maven will package all **.class** files and resources into one file. This file will be formatted into one of the three types given below:

JAR Archive

WAR Archive

EAR Archive



.class files



.class files



Maven



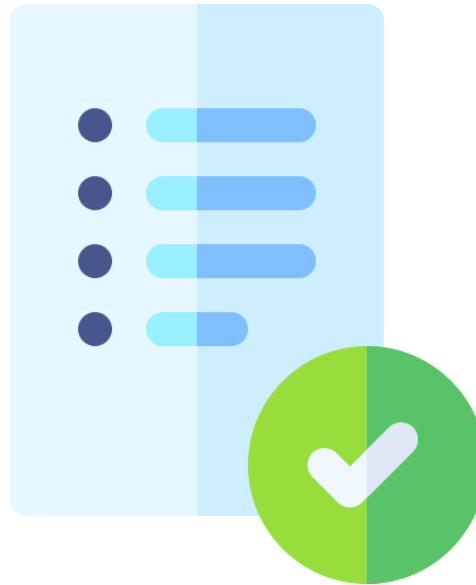
Packaging



Archive files

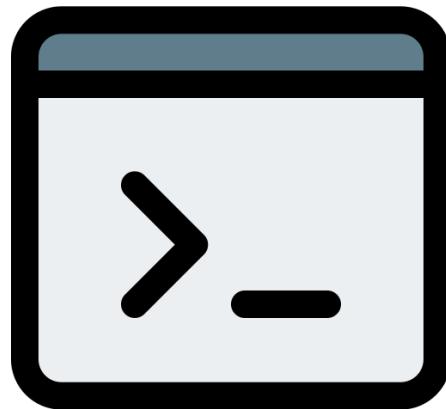
5. Verify Phase

It runs any checks on results of integration tests to ensure quality criteria are met, at the same time
all the previous lifecycle phases are also executed



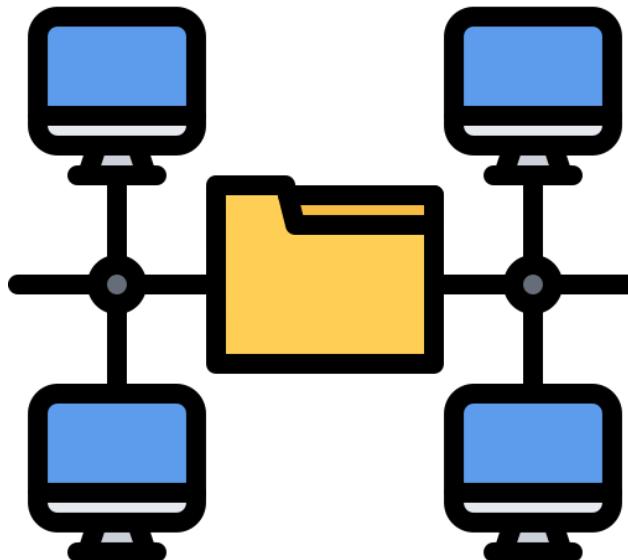
6. Install Phase

install the package into the local repository, for use as a dependency in other projects locally, all previous lifecycle phases are also executed



7. Deploy Phase

It copies the final package to the remote repository for sharing with other developers and projects.

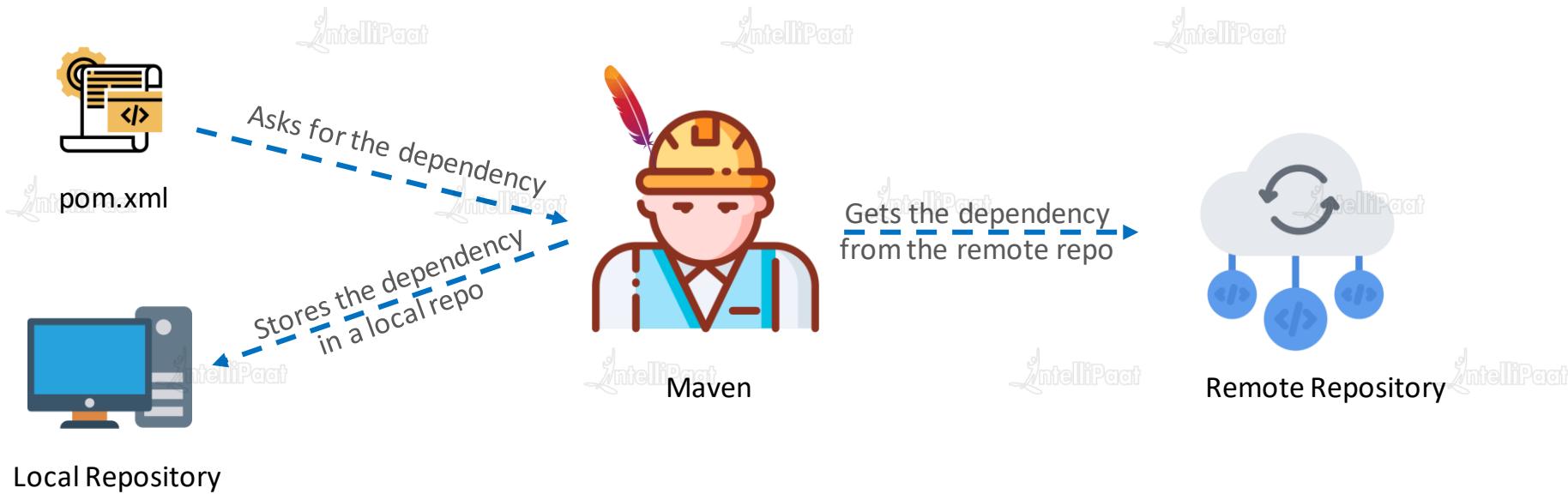




Maven Repositories

Maven Repositories

After Maven understands which all dependencies are needed from the pom.xml file, it will download those dependencies from remote repositories and then store them in the local repository for current or future use



Installing Maven

Installing Maven

1. Download and Install Maven on Ubuntu 18.04 on AWS
2. Create a sample project using the maven generate command
3. Verify the directory structure by installing tree

Building a Project using Maven

Installing Maven

1. Create a sample Dynamic Website Project in Jenkins
2. Push the code to github
3. On the AWS Instance clone the code and Build the code
4. Deploy this code on tomcat to verify, if everything looks good

What is Continuous Integration?

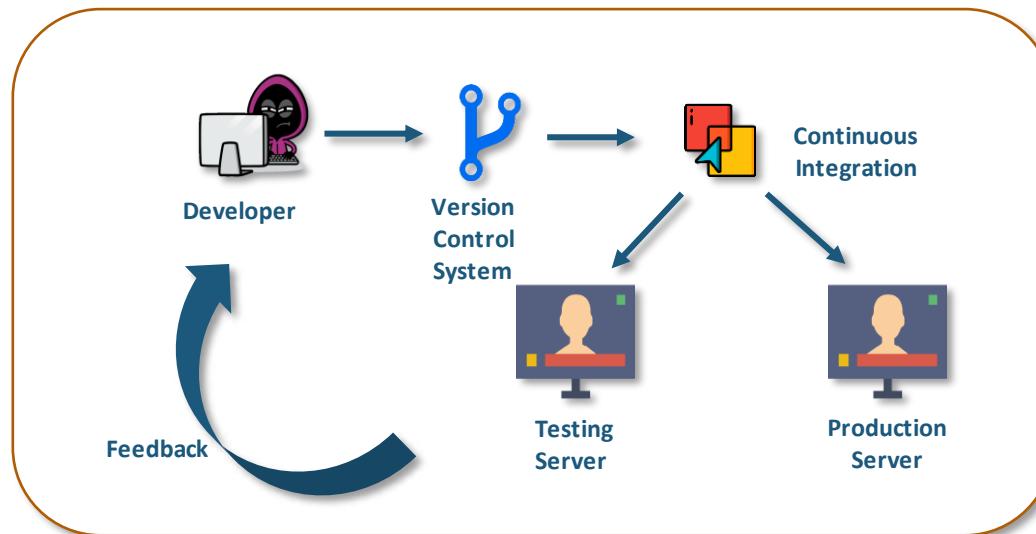
Problems before Continuous Integration



- ✖ 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?

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



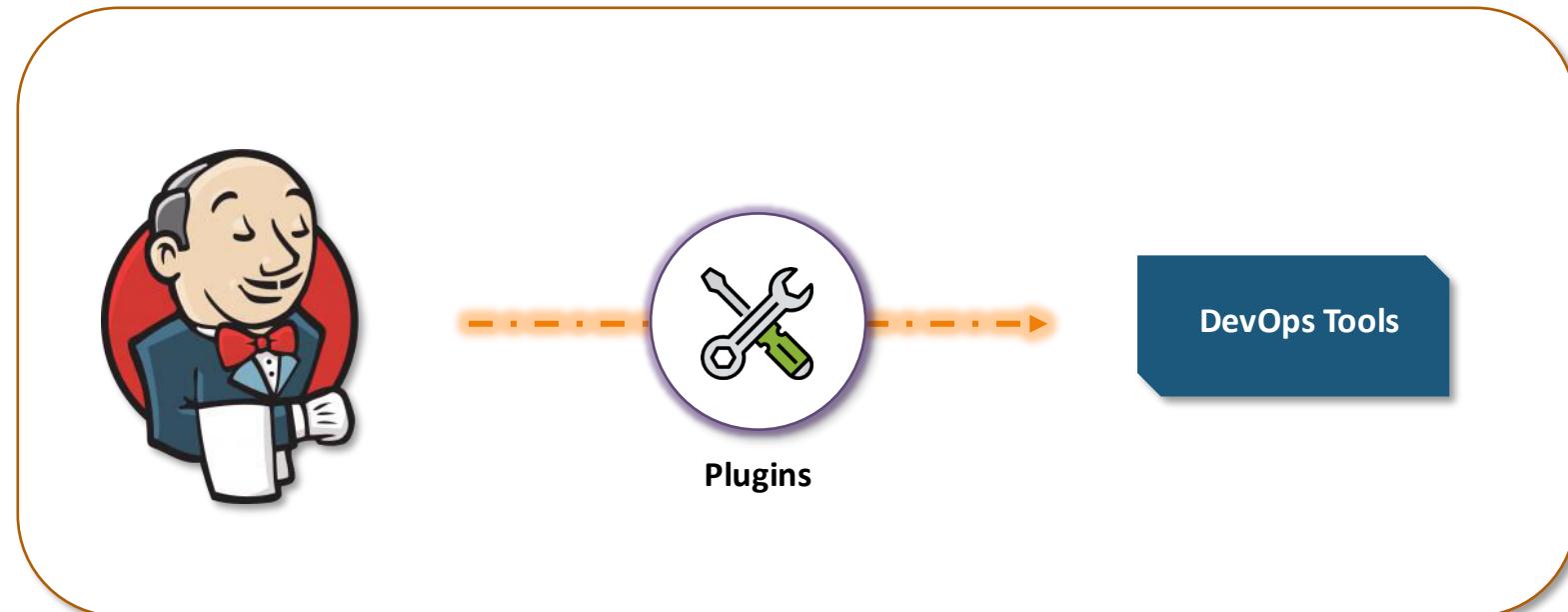
1. Launch an AWS Instance
2. Connect through SSH
3. Execute the following commands:

Jenkins Installation:

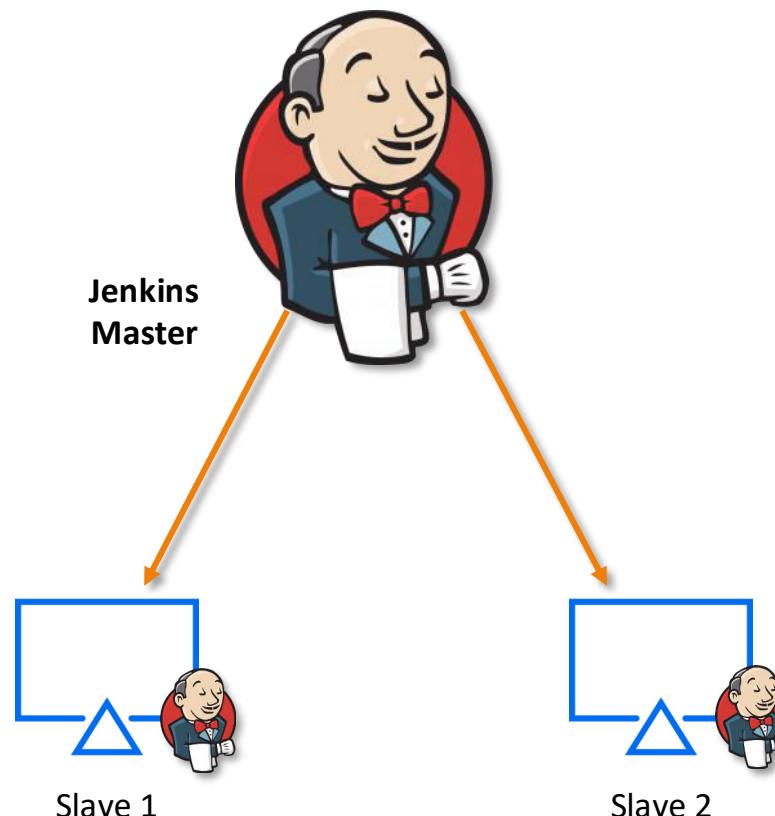
```
$> sudo apt-get update  
$> sudo apt install openjdk-8-jdk  
$> wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -  
$> sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'  
$> sudo apt update  
$> sudo apt install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

Jenkins Architecture

Jenkins Architecture



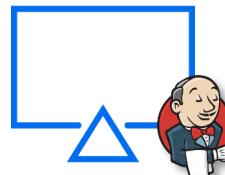
Jenkins Architecture



Managing Nodes on Jenkins

Managing Nodes on Jenkins

Add a slave node to Jenkins using JNLP connection



Managing users on Jenkins

Managing Users on Jenkins

We can provide access to multiple users in Jenkins and can provide them with roles based on project-based access.



Let's start by adding some users in Jenkins

Hands-on: Adding users in Jenkins

Managing Users on Jenkins

Now obviously, you would not want each user to be an administrator in Jenkins,
hence you can also control what permissions each user gets



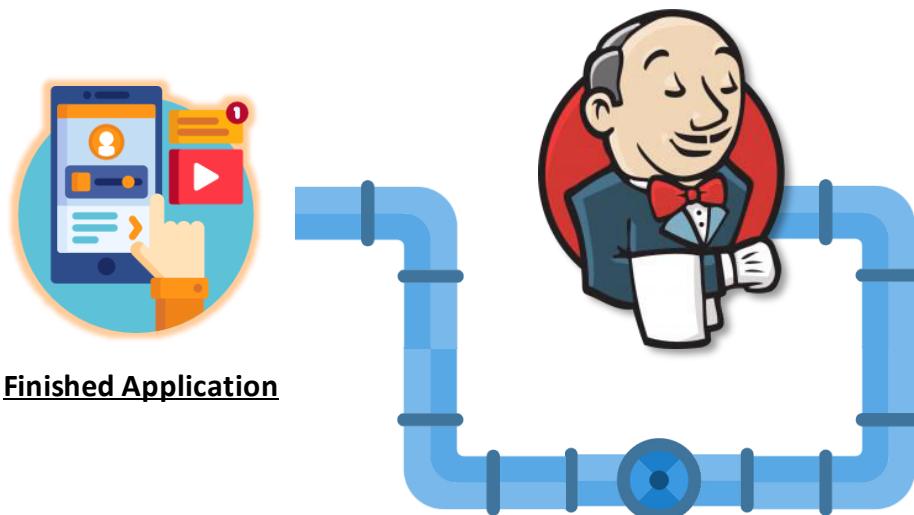
Let's see, how we can manage permissions to a user using Matrix Based Authorization

Hands-on: Giving restricted access to users

Understanding CI/CD Pipelines in Jenkins

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.

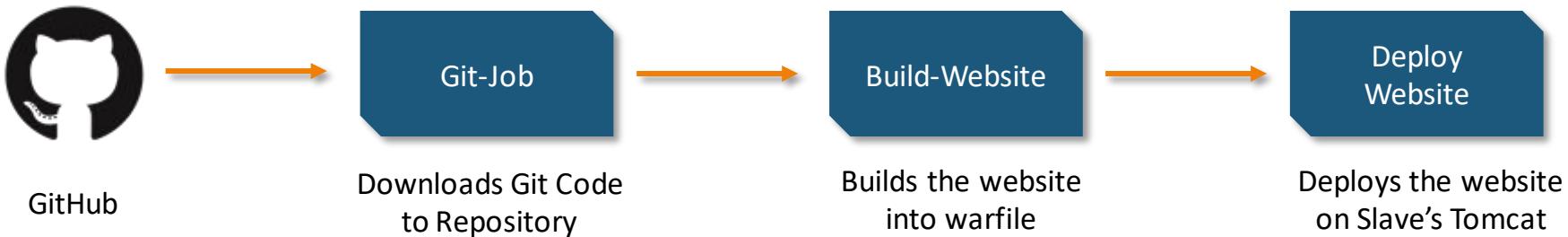


Finished Application

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 in Jenkins

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 → Deploy-Website
3. Install the plugin for the pipeline view
4. Make changes to the website and commit the job to see the changes





India: +91-7847955955



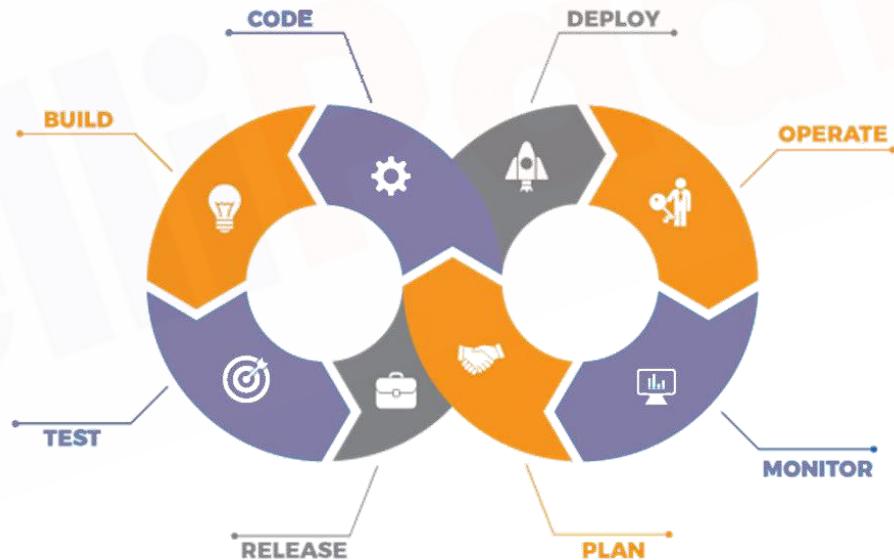
US: 1-800-216-8930 (TOLL FREE)



sales@intellipaat.com

24/7 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

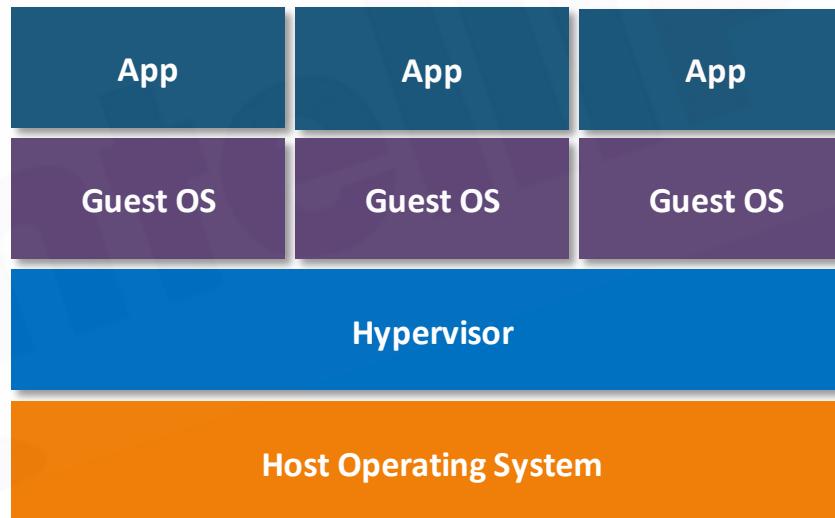
08

Introduction to
Dockerfile

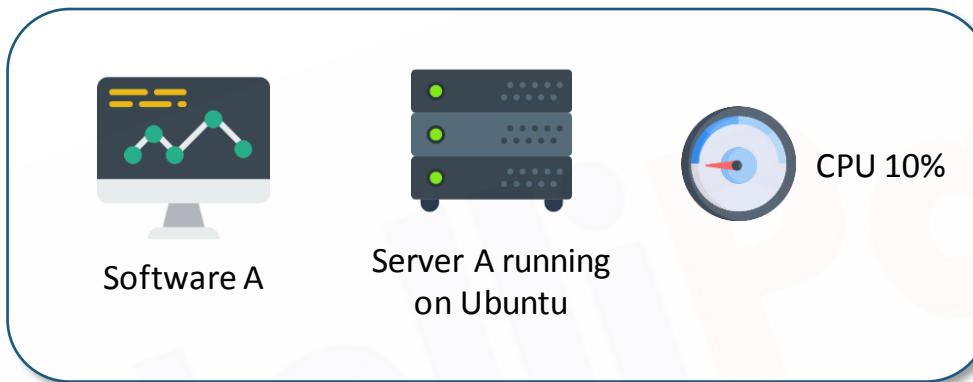
What is Virtualization?

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%

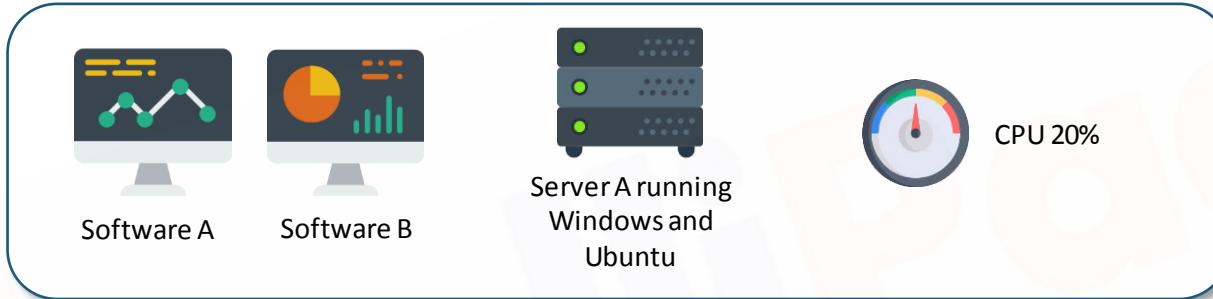
Some time later, we needed Software B which can only run on Windows. Therefore, we had to buy and run a Server B which had windows running on it. The software took only 10% of the CPU resources.

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

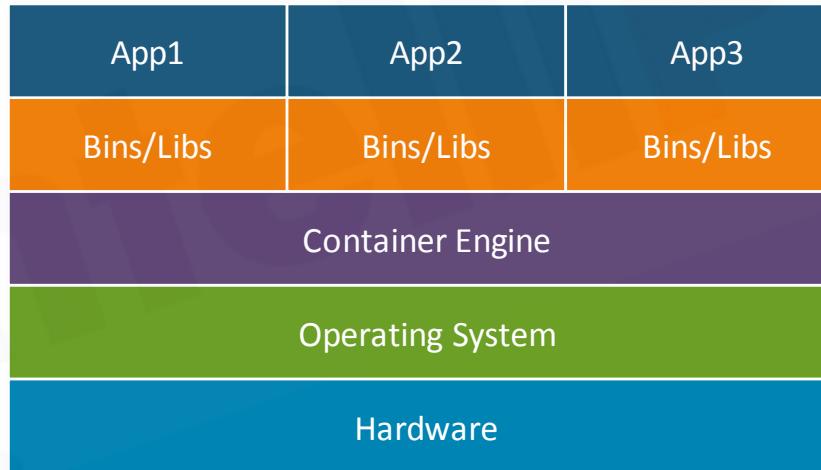


- ✓ It results in reduced spending.
- ✓ Resources are utilized more efficiently.
- ✓ Process of getting software up and running is shorter.
- ✓ Easier backup and disaster recovery is available.

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.



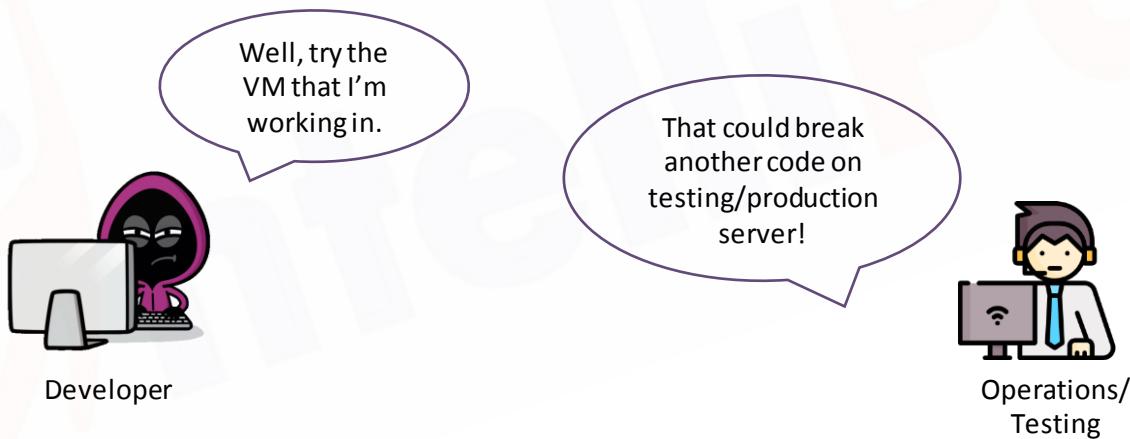
Problems before Containerization

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.



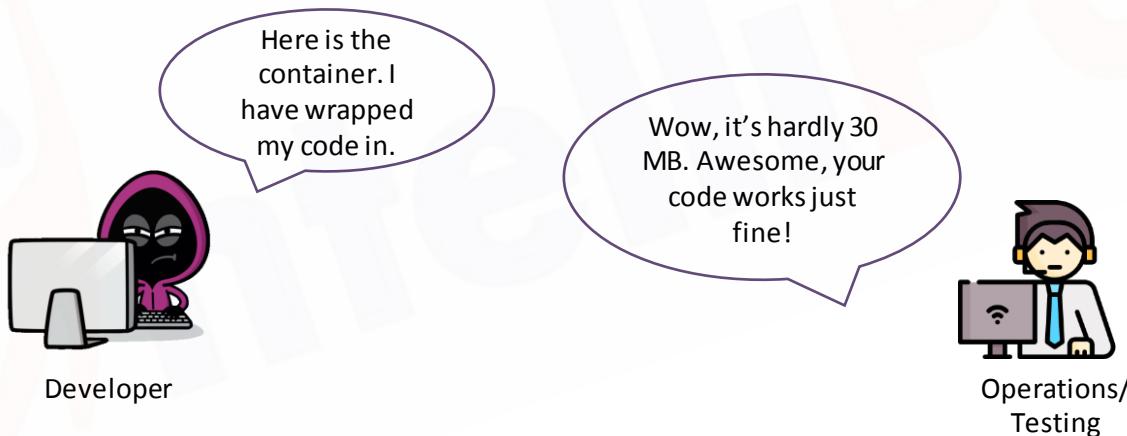
Problems before Containerization



- ✖ 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



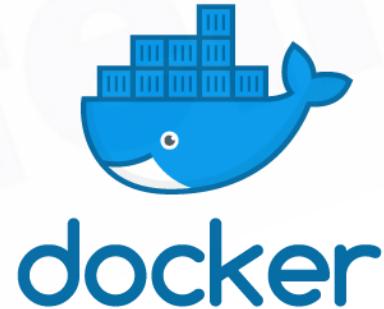
- ✓ Containers are not resource hungry.
- ✓ They are lightweight and hence portable.
- ✓ They are developer friendly and can be configured through the code.

Containerization Tools

Containerization Tools

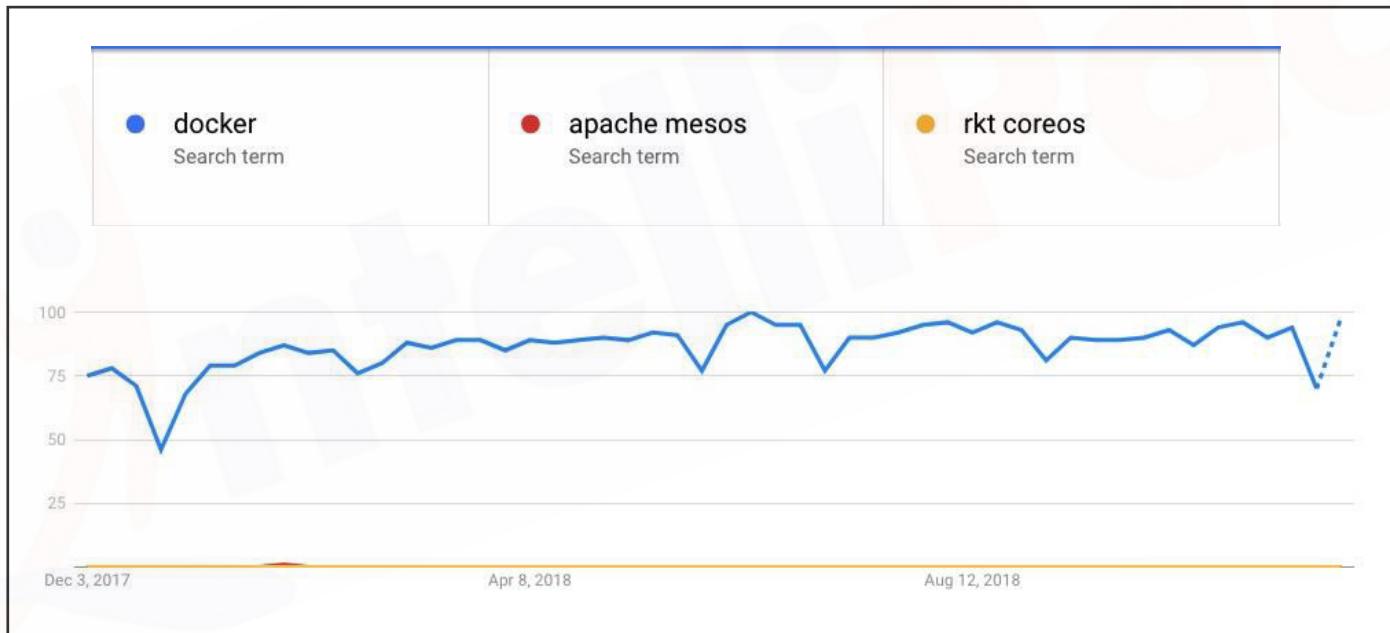


MESOS



Containerization Tools

Docker is clearly the most famous among them all!

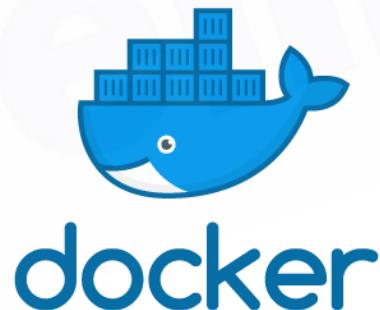


What is Docker?

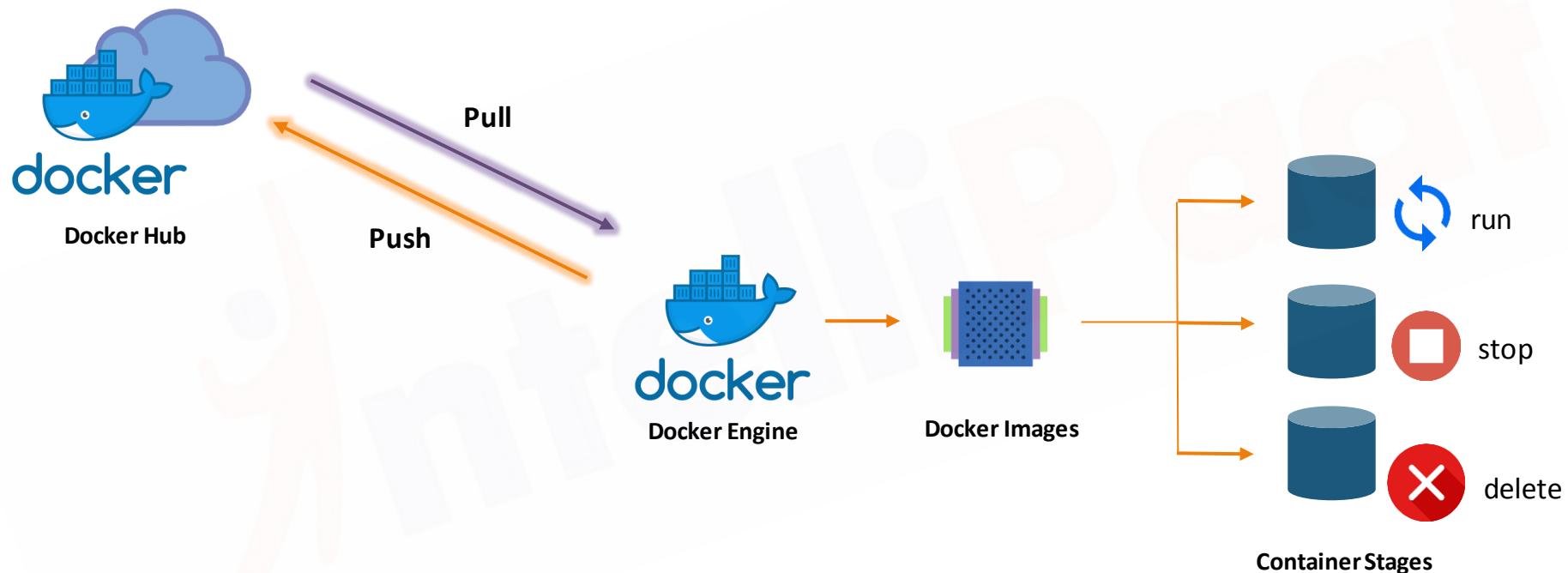
What is Docker?

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".

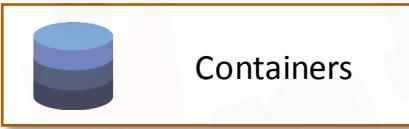


Docker Container Life Cycle

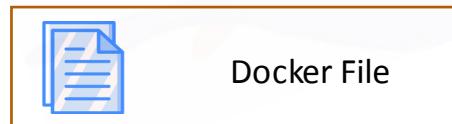
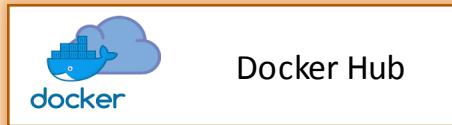


Components of Docker Ecosystem

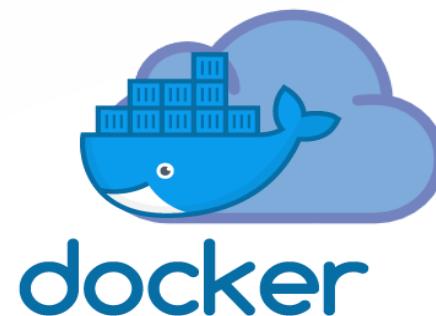
Components of Docker Ecosystem



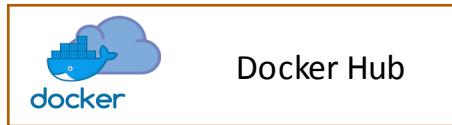
Components of Docker Ecosystem



- ★ 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 Hub



Docker Engine



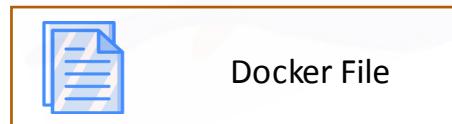
Docker Images



Containers

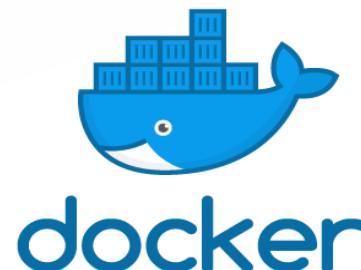


Docker Volumes

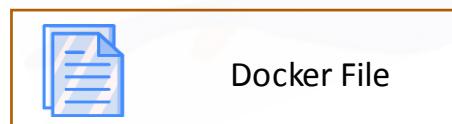
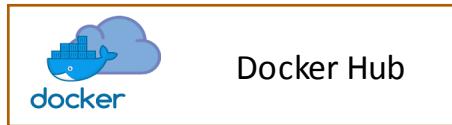


Docker File

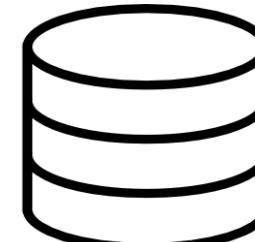
- ★ Docker Engine is the heart of the docker ecosystem.
- ★ It is responsible for managing your container runtimes.
- ★ It works on top of operating system level.
- ★ It utilizes the kernel of the underlying OS.



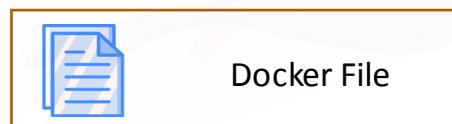
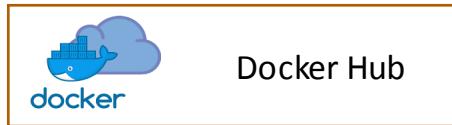
Components of Docker Ecosystem



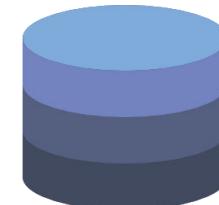
- ★ Docker Image is like the template of a container.
- ★ It is created in layers.
- ★ Any new changes in the image results in creating a new layer.
- ★ One can launch multiple containers from a single docker image.



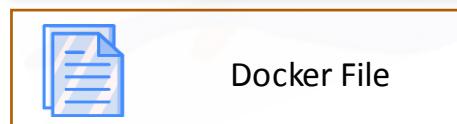
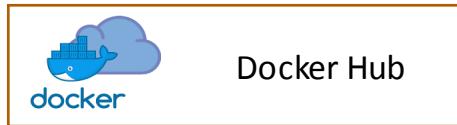
Components of Docker Ecosystem



- ★ 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.



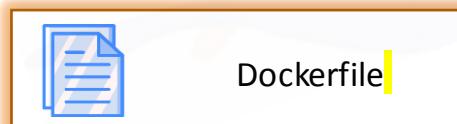
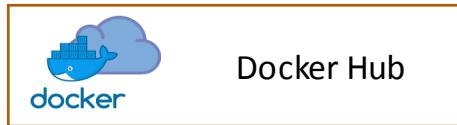
Components of Docker Ecosystem



- ★ 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

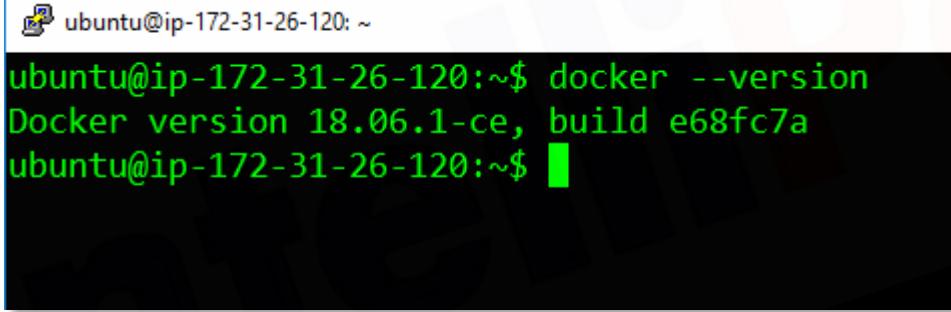


Installing Docker

Common Docker Commands

Common Docker Commands

```
docker --version
```



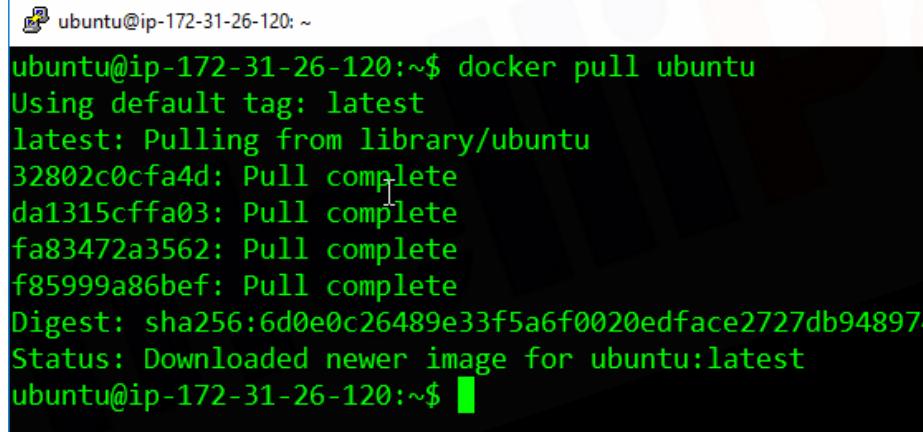
A terminal window on an Ubuntu system (indicated by the icon in the top-left corner) showing the output of the `docker --version` command. The output displays the Docker version as 18.06.1-ce and the build hash as e68fc7a.

```
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>
```



A terminal window showing the execution of the `docker pull ubuntu` command. The output indicates that the latest version of the Ubuntu image is being pulled, with each layer being downloaded and completed. The final status message shows that a newer image was downloaded for the `ubuntu:latest` tag.

```
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:~$ 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

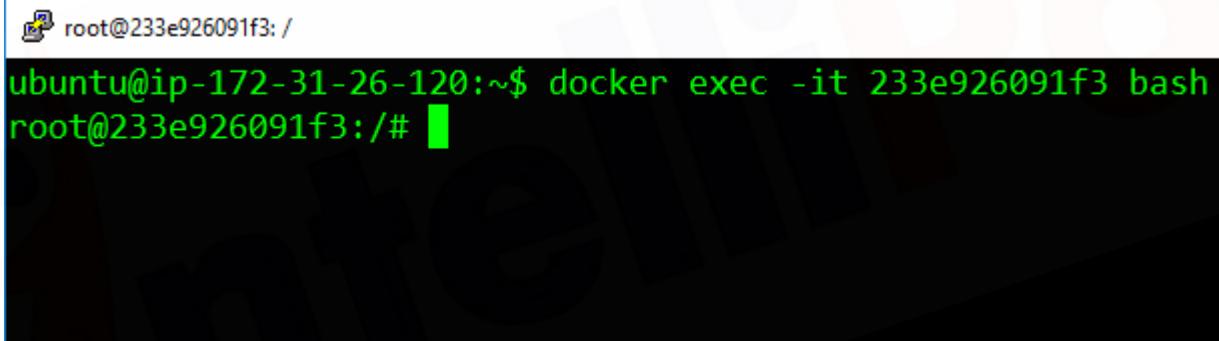
```
docker ps -a
```

```
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 execution of a Docker container. The prompt is root@233e926091f3:/. The user runs the command docker exec -it 233e926091f3 bash, which brings them into the container's bash shell. The shell prompt changes to root@233e926091f3:/#.

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

Common Docker Commands

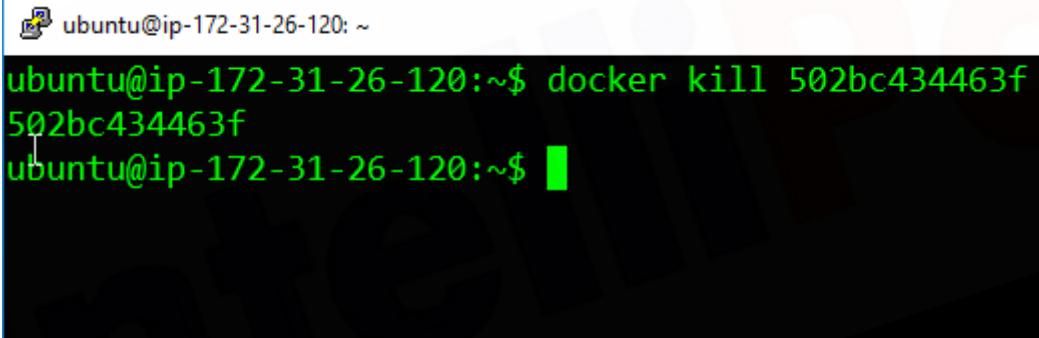
```
docker stop <container-id>
```

```
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 execution of the `docker kill` command. The prompt is `ubuntu@ip-172-31-26-120: ~`. The user types `docker kill 502bc434463f`, followed by the container ID `502bc434463f`. After pressing enter, the command is completed, and the terminal returns to the prompt.

```
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>
```



A terminal window showing the execution of the `docker rm` command. The command `docker rm 502bc434463f` is entered and executed, resulting in the removal of the container with ID `502bc434463f`. The terminal prompt shows the user is back at the root directory (~).

```
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:~$ █
```

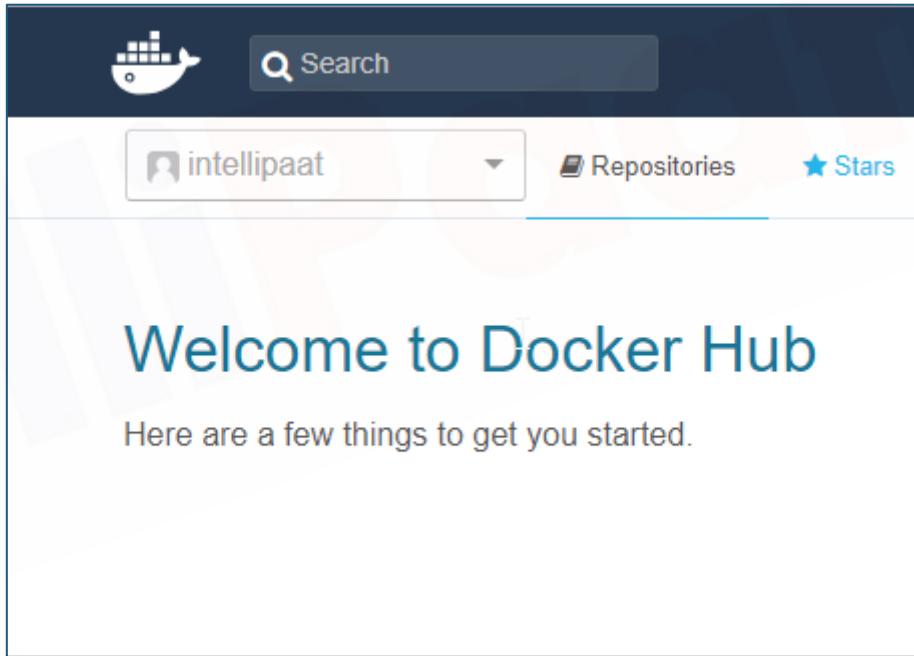
A black rectangular box containing a terminal session. The session starts with a prompt 'ubuntu@ip-172-31-26-120: ~\$', followed by the command 'docker rmi 93fd78260bd1'. The output shows the removal of the 'ubuntu' image and its specific SHA256 digest. Finally, the prompt 'ubuntu@ip-172-31-26-120:~\$' is shown again with a terminal cursor at the end.

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

Creating a Docker Hub Account

Creating a Docker Hub Account

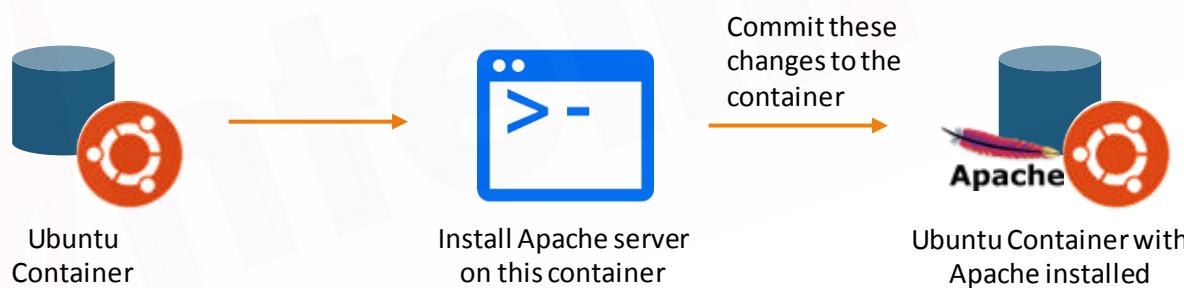
1. Navigate to <https://hub.docker.com>
2. Sign up on the website
3. Agree to the terms and conditions
4. Click on Sign up
5. Check your email, and verify your email by clicking on the link
6. Finally, login using the credentials you provided on the sign up page



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
```

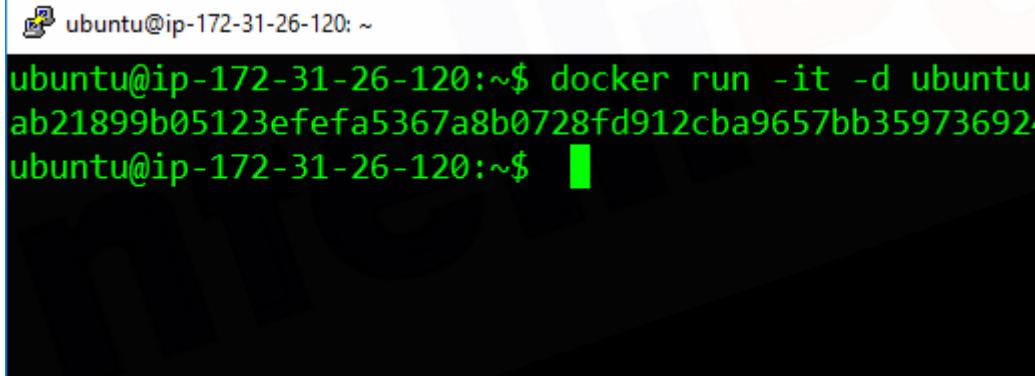
```
ubuntu@ip-172-31-26-120: ~
ubuntu@ip-172-31-26-120:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
32802c0cfa4d: Pull complete
da1315cffa03: Pull complete
fa83472a3562: Pull complete
f85999a86bef: Pull complete
Digest: sha256:6d0e0c26489e33f5a6f0020edface2727db9489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



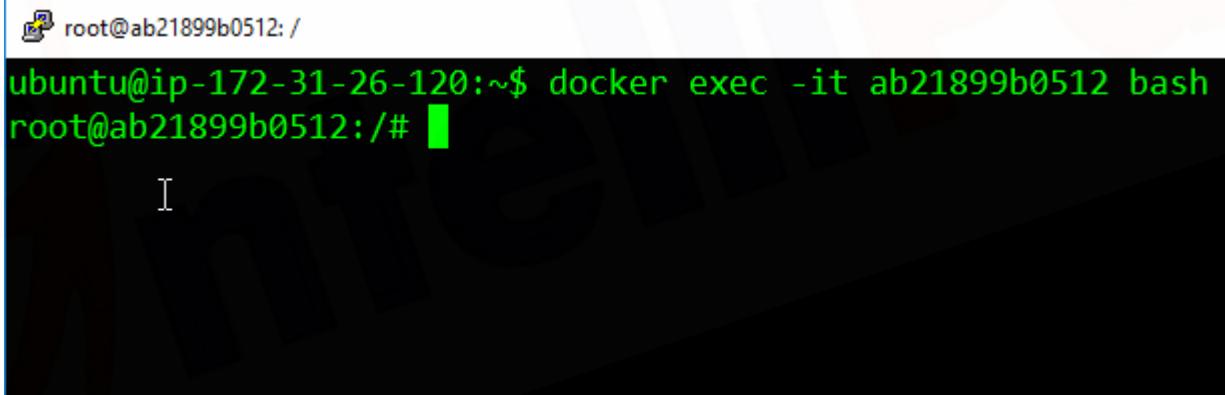
A terminal window showing the command `docker run -it -d ubuntu` being run. The output shows a long container ID and a prompt for further commands.

```
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 being executed. The prompt is root@ab21899b0512: / . The command entered is ubuntu@ip-172-31-26-120:~\$ docker exec -it ab21899b0512 bash . The response shows the user is now in a bash shell within the container, with the prompt root@ab21899b0512:/# . A cursor is visible at the end of the prompt.

```
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: ~
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:

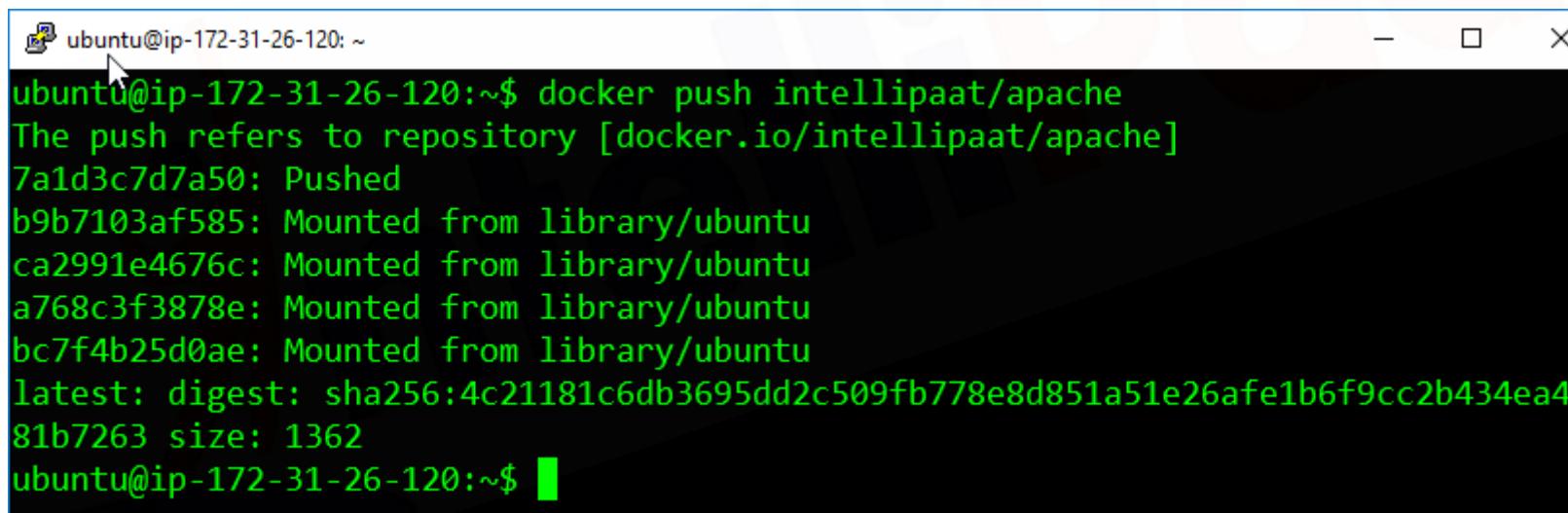
docker login

```
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 pushes a container to a repository named "intellipaat/apache". The output shows the repository details and the SHA-256 digest of the image.

```
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 dropdown menu showing 'intellipaat'. Below the header, there are tabs for 'Repositories', 'Stars', and 'Contributed'. The main area is titled 'Repositories' and contains a search bar with the placeholder 'Type to filter repositories by name'. Below the search bar, there's a list of repositories. The first repository shown is 'intellipaat/apache', which is described as 'public'. It has a small profile picture icon next to its name.

Repositories

Type to filter repositories by name

intellipaat/apache
public

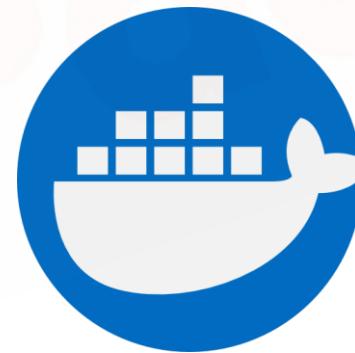
Now anyone, who wants to download this container, can simply pass the following command:

```
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'

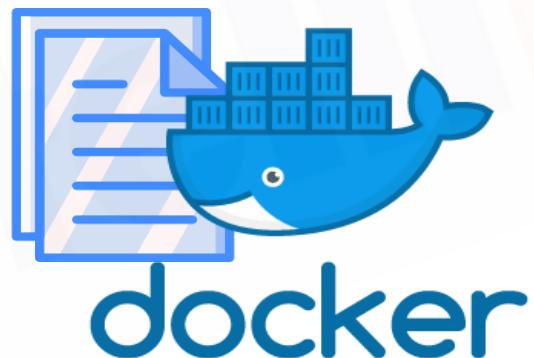


Hands-on: Creating a Private Registry in Docker

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.

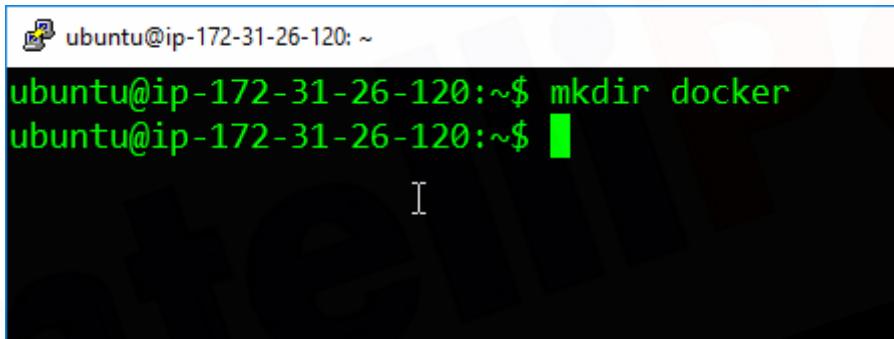
Example

```
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.

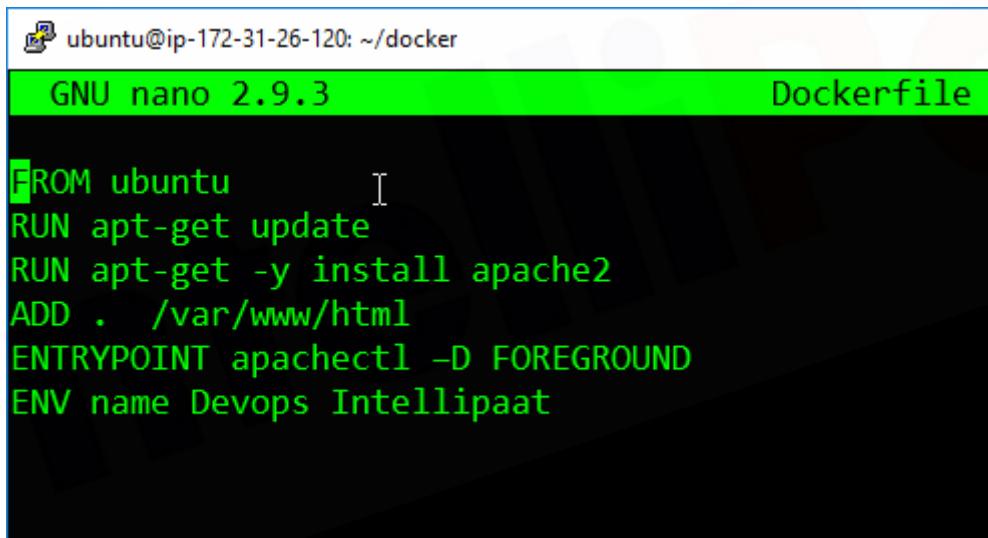


A screenshot of a terminal window on an Ubuntu system. The window title bar shows the user's name and IP address: "ubuntu@ip-172-31-26-120: ~". The terminal prompt is "ubuntu@ip-172-31-26-120:~\$". Below the prompt, the command "mkdir docker" is typed and has just been executed. A cursor is visible at the end of the command line. The background of the terminal window is dark, and the text is white or light-colored.

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

Running the Sample Dockerfile

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

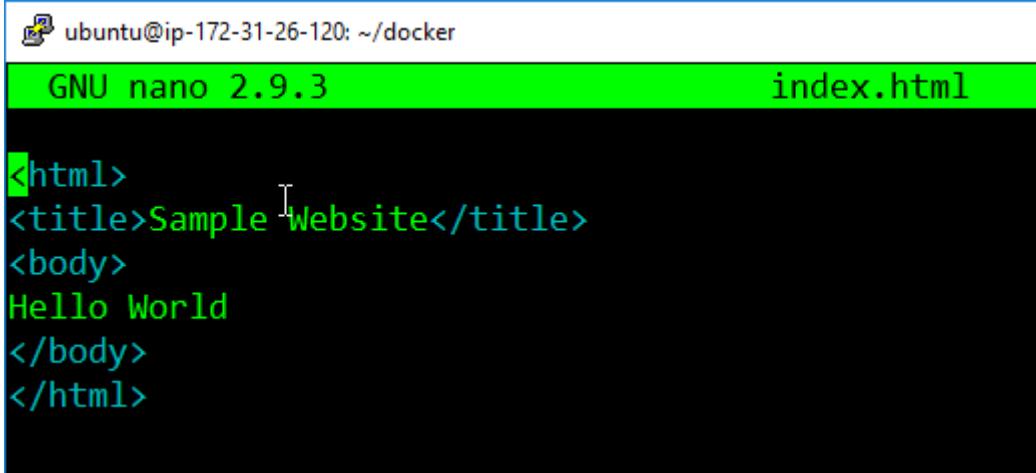


A screenshot of a terminal window titled "Dockerfile". The title bar is green and displays "ubuntu@ip-172-31-26-120: ~/docker" and "GNU nano 2.9.3". The main area of the terminal shows a Dockerfile with the following content:

```
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.



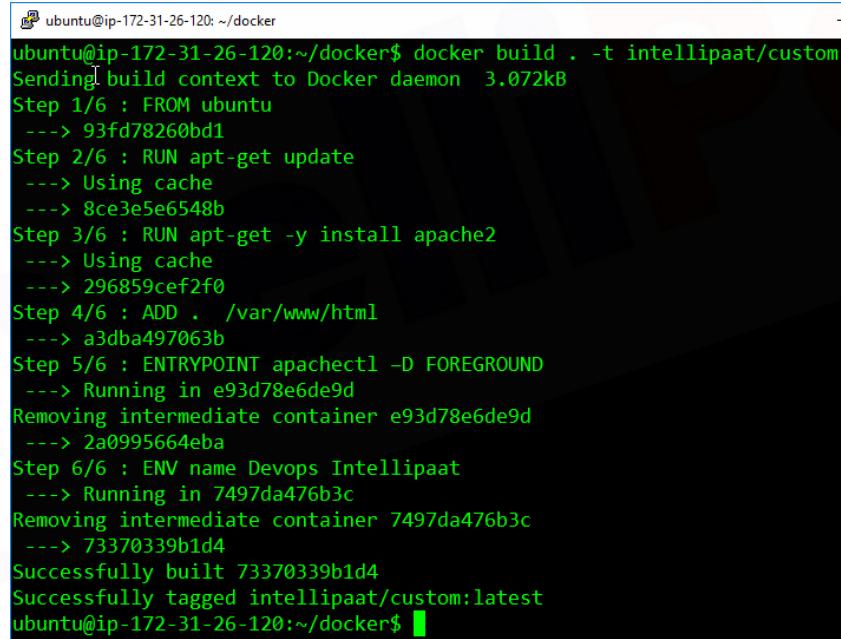
A screenshot of a terminal window titled "ubuntu@ip-172-31-26-120: ~/docker". The title bar is green, and the main area shows the content of an "index.html" file being edited with the "GNU nano 2.9.3" text editor. The file contains the following HTML code:

```
<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>
```



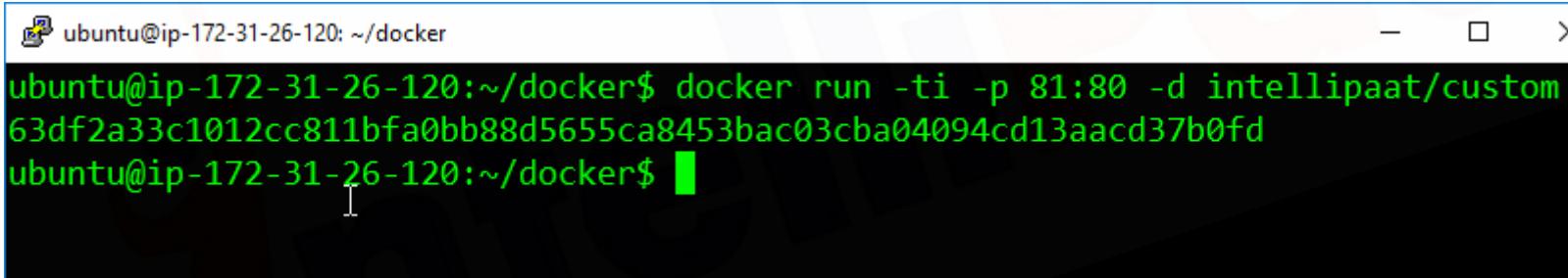
A terminal window showing the execution of a Docker build command. The command is `docker build . -t intellipaat/custom`. The output shows the build process, which includes sending the build context to the Docker daemon, executing several steps (Step 1/6 to Step 6/6), adding files, setting an entrypoint, and finally successfully building and tagging the image as `intellipaat/custom:latest`.

```
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:

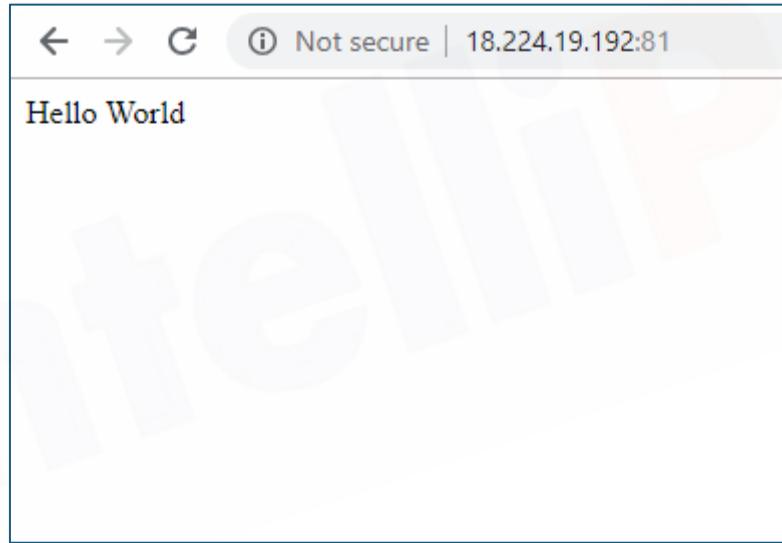
```
docker run -it -p 81:80 -d intellipaat/custom
```

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 generated container ID "63df2a33c1012cc811bfa0bb88d5655ca8453bac03cba04094cd13aacd37b0fd". The prompt "ubuntu@ip-172-31-26-120:~/docker\$" is visible at the bottom, with a cursor indicating an active session.

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

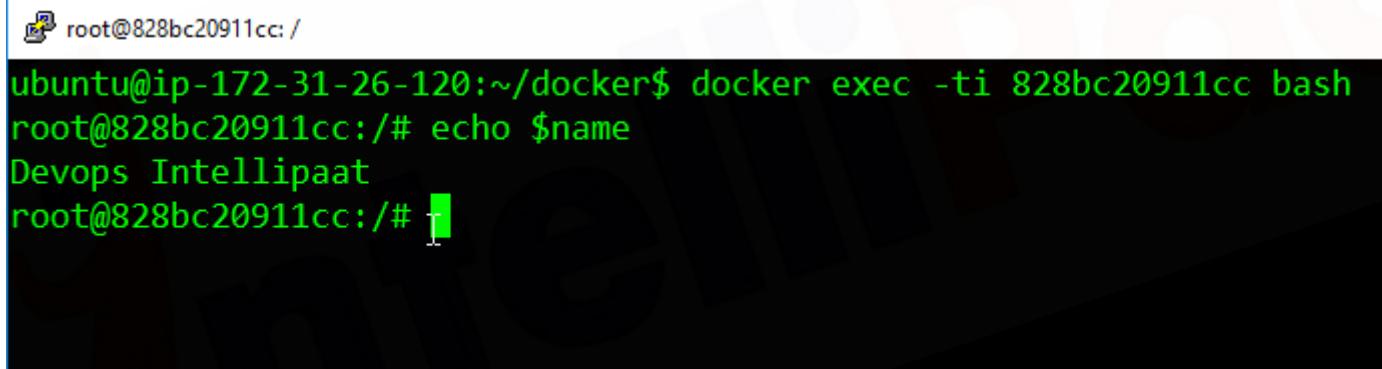
Running the Sample Dockerfile

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:/# [
```

A screenshot of a terminal window with a black background and white text. The window title bar shows a small icon of a laptop and the text "root@828bc20911cc: /". The terminal content starts with "ubuntu@ip-172-31-26-120:~/docker\$ docker exec -ti 828bc20911cc bash", followed by "root@828bc20911cc:/# echo \$name", then "Devops Intellipaat", and finally "root@828bc20911cc:/# [". A green cursor is visible at the end of the line.

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

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.

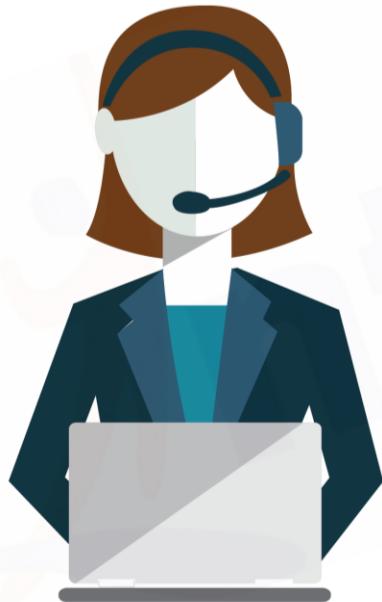
A. True

B. False

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

A. True

B. False



India: +91-7847955955



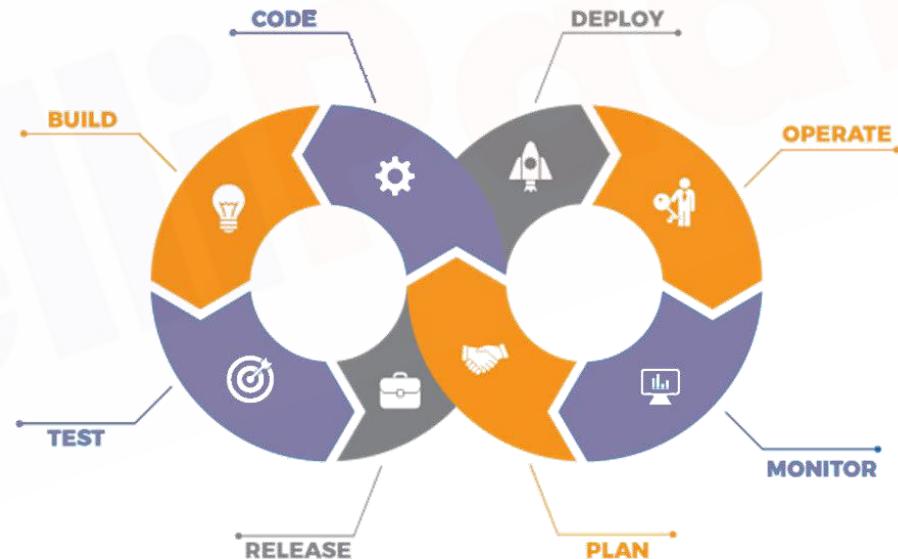
US: 1-800-216-8930 (TOLL FREE)



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

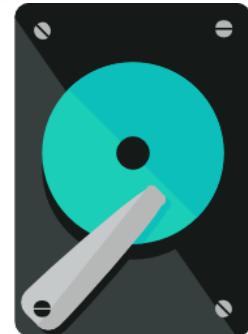
Docker
Networks

Introduction to Docker Storage

Introduction to Docker Storage

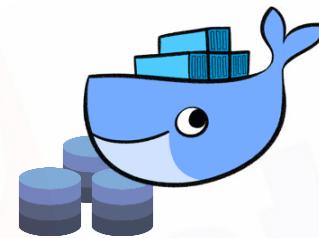
By default, all data of a container is stored on a writable container layer. This layer has the following properties:

- ★ 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.

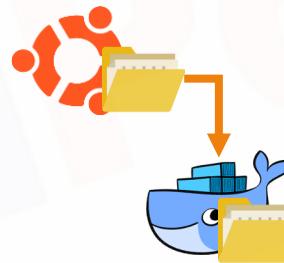


Introduction to Docker Storage

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

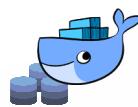
A **Docker Volume** is a mountable entity which can be used to store data in the docker filesystem.

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

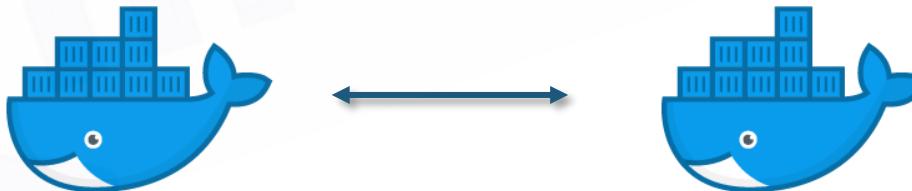
```
docker run -it -v <source-directory>:<destination-directory>  
-d <container-name>
```

```
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

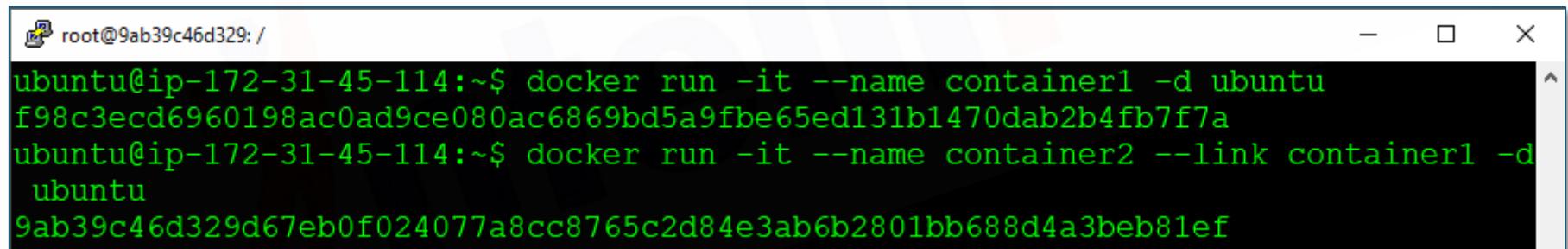
- ★ Linking is a legacy feature of Docker, which is used to connect multiple containers.
- ★ With linking, containers can communicate among each other.
- ★ Name of containers is an important aspect while linking containers.
- ★ Once you link containers, they can reach out to others using their names.



Linking Docker Containers

Syntax

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



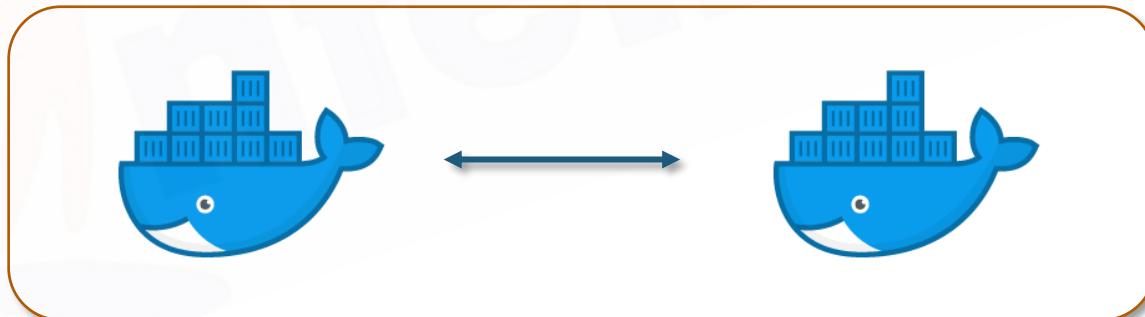
A screenshot of a terminal window titled 'root@9ab39c46d329: /'. The window shows two Docker commands being run:

```
root@9ab39c46d329: /  
ubuntu@ip-172-31-45-114:~$ docker run -it --name container1 -d ubuntu  
f98c3ecd6960198ac0ad9ce080ac6869bd5a9fbe65ed131b1470dab2b4fb7f7a  
ubuntu@ip-172-31-45-114:~$ docker run -it --name container2 --link container1 -d  
ubuntu  
9ab39c46d329d67eb0f024077a8cc8765c2d84e3ab6b2801bb688d4a3beb81ef
```

Hands-on: Linking Docker Containers

Hands-on: Linking Docker Containers

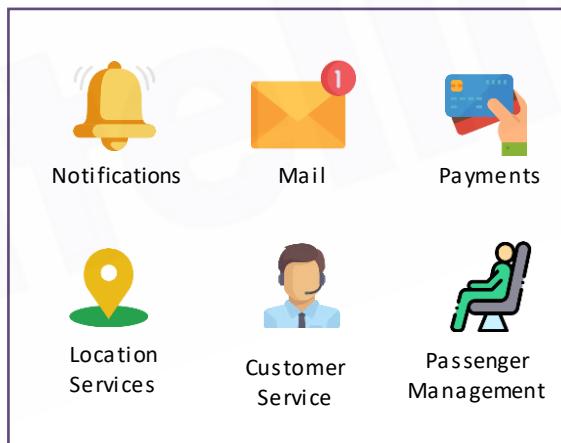
1. Create two containers of Ubuntu with names as follows: container1 and container2
2. Link container2 to container1
3. Try pinging from container2 to container1 by just using the command “ping container1”



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.

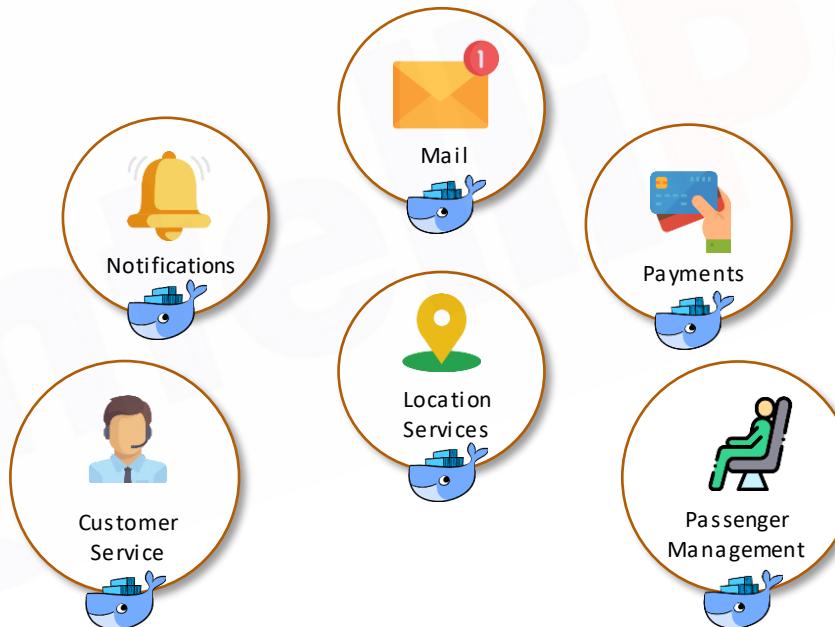
What are Microservices?

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

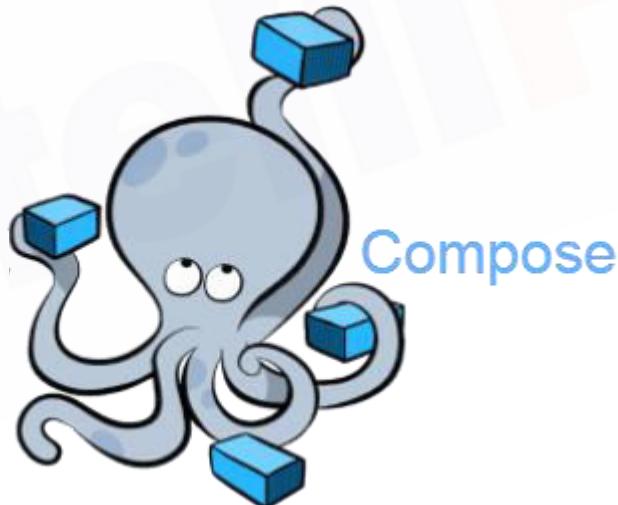


- ✓ Application is distributed, hence easy to understand.
- ✓ The code of only the Microservice which is supposed to be updated is changed.
- ✓ Bug, in one service, does not affect other services.
- ✓ There is no barrier to any specific technology.

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:

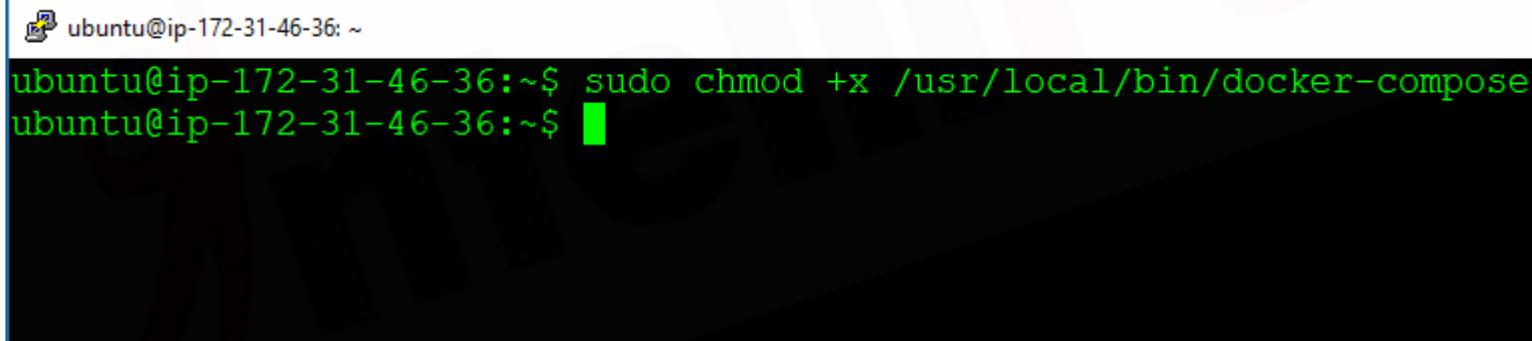
```
sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-  
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
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 title bar shows the session name "ubuntu@ip-172-31-46-36: ~". The terminal itself contains the command "sudo chmod +x /usr/local/bin/docker-compose" in green text, indicating it has been successfully run. The background of the terminal window is dark, and the overall interface is typical of a Linux desktop environment.

```
ubuntu@ip-172-31-46-36:~$ sudo chmod +x /usr/local/bin/docker-compose
```

Installing Docker Compose

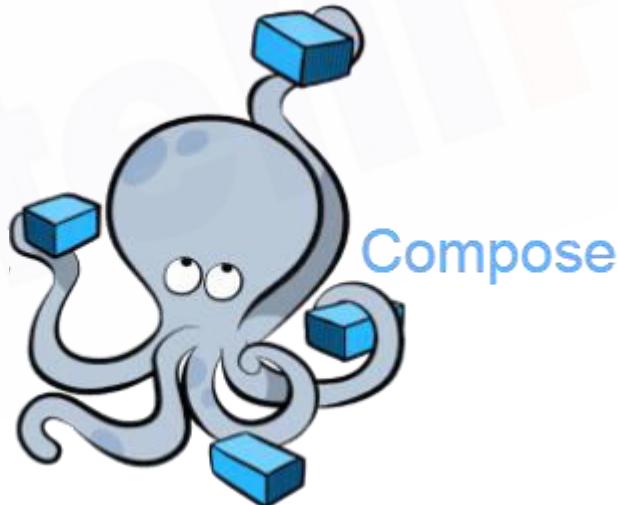
3. Finally, verify your installation using the following command:

```
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:

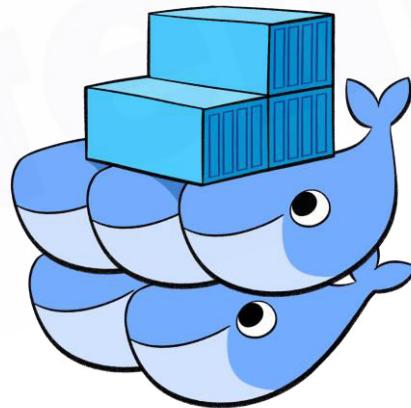
```
docker-compose up -d
```

4. Ensure that all your containers are running

What is Container Orchestration?

What is Container Orchestration?

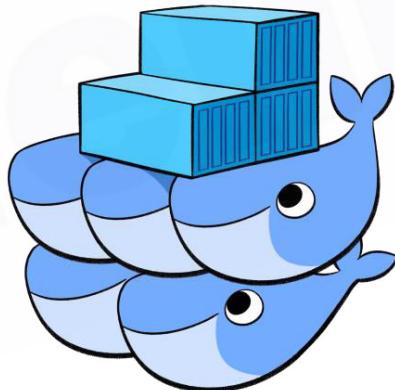
Applications are typically made up of individually containerized components (often called microservices) that must be organized at the networking level in order for the application to run as **intended**. The process of organizing multiple **containers** in this manner is known as **container orchestration**.



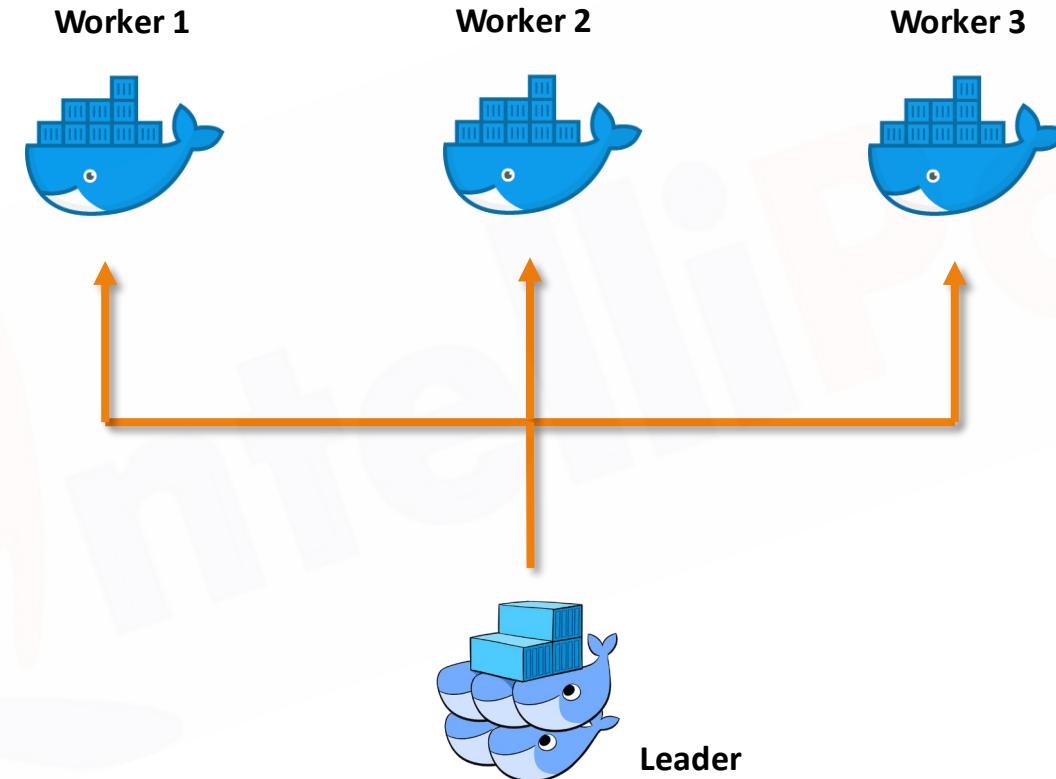
Introduction to Docker Swarm

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$ █
```

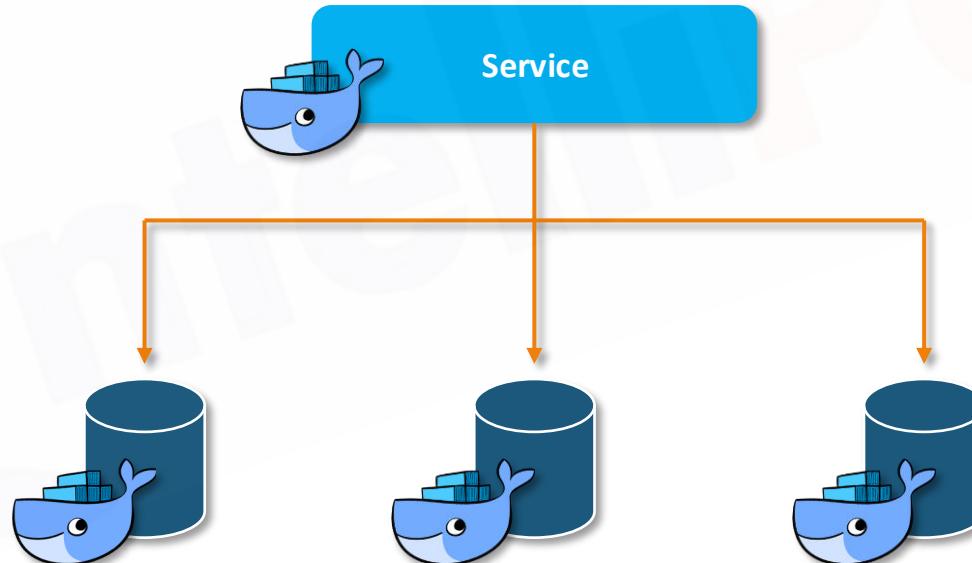
This command should be passed on to the worker node to join the docker swarm cluster.



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

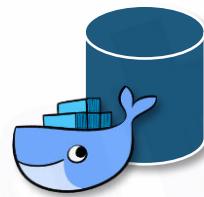
1. Create a Service for nginx webserver
2. There should be 3 replicas of this service running on the swarm cluster
3. Try accessing the service from Master IP and Slave IP



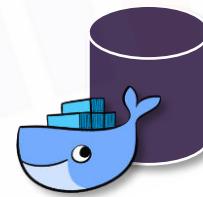
Docker Networks

Why Docker Networks?

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



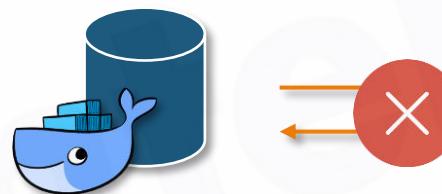
Website Container



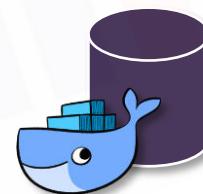
Database Container

Why Docker Networks?

By default, these containers cannot communicate with each other.



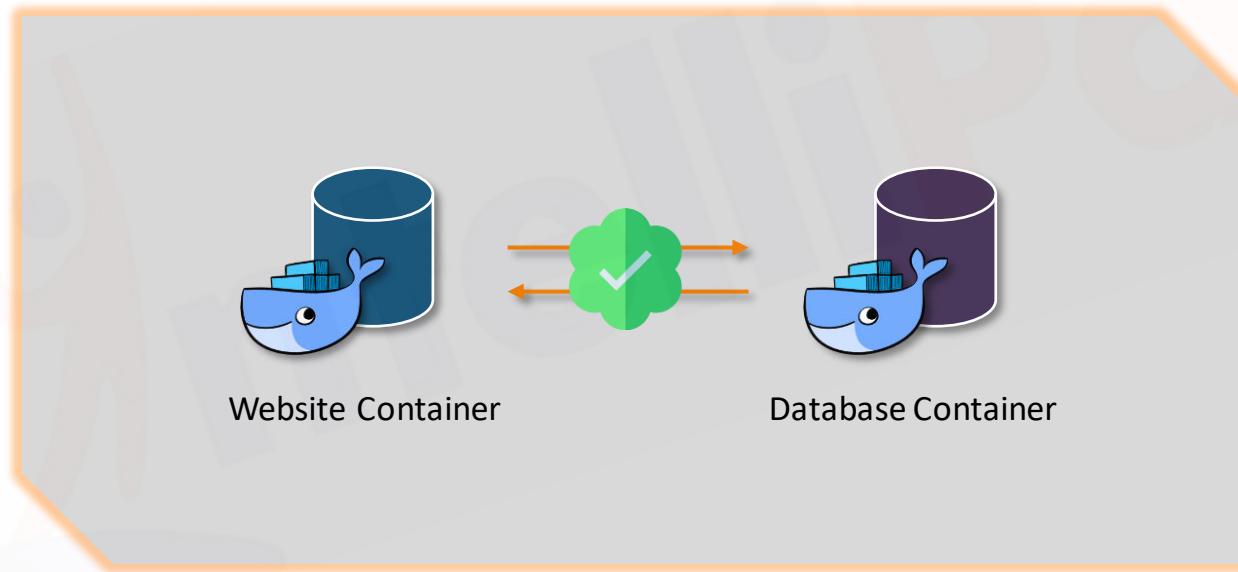
Website Container



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

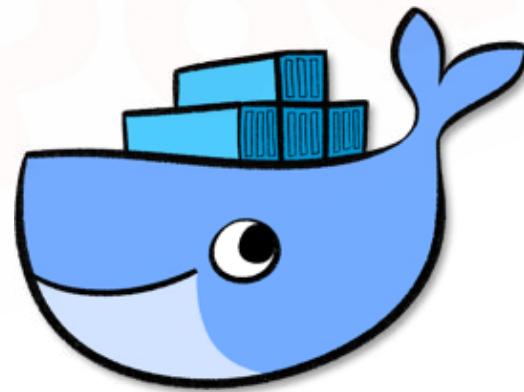
None

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

Quiz

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

A. True

B. False



India: +91-7847955955



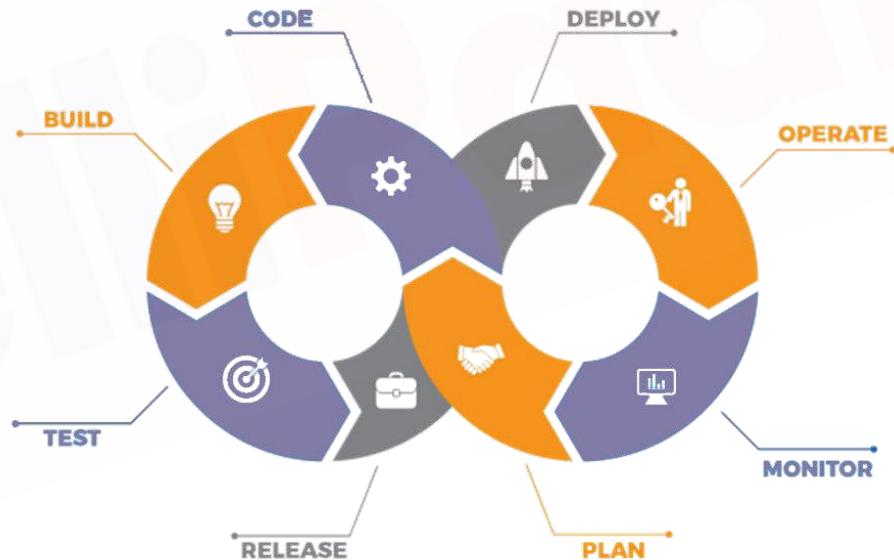
US: 1-800-216-8930 (TOLL FREE)



support@intellipaat.com

24/7 Chat with Our Course Advisor

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

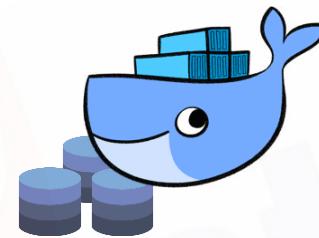
By default, all data of a container is stored on a writable container layer. This layer has the following properties:

- ★ 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.

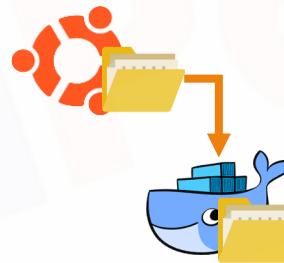


Introduction to Docker Storage

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

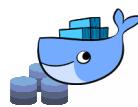
A **Docker Volume** is a mountable entity which can be used to store data in the docker filesystem.

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

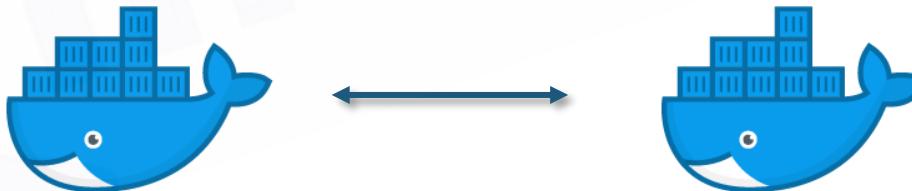
```
docker run -it -v <source-directory>:<destination-directory>  
-d <container-name>
```

```
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

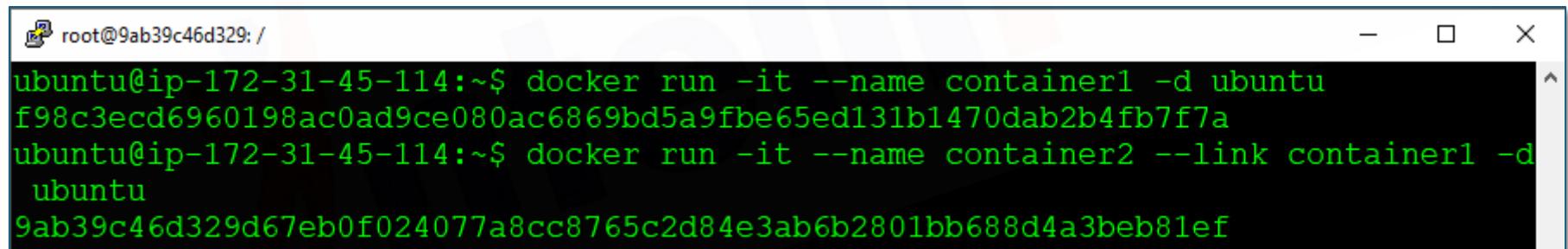
- ★ Linking is a legacy feature of Docker, which is used to connect multiple containers.
- ★ With linking, containers can communicate among each other.
- ★ Name of containers is an important aspect while linking containers.
- ★ Once you link containers, they can reach out to others using their names.



Linking Docker Containers

Syntax

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



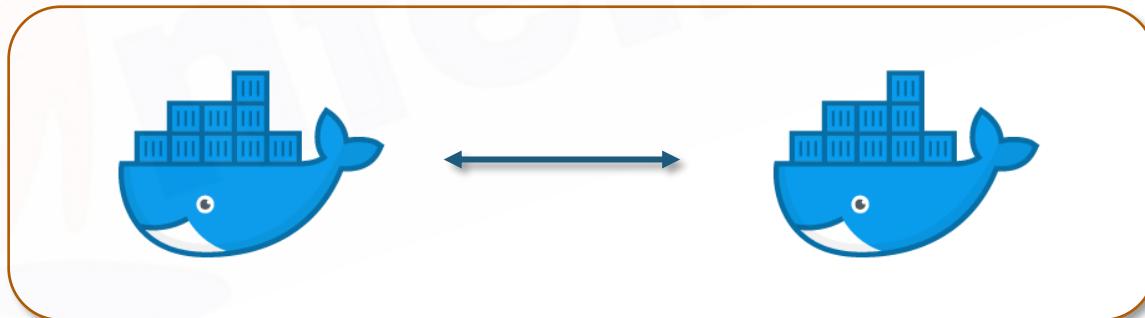
A screenshot of a terminal window titled 'root@9ab39c46d329: /'. The window shows two Docker commands being run:

```
root@9ab39c46d329: /  
ubuntu@ip-172-31-45-114:~$ docker run -it --name container1 -d ubuntu  
f98c3ecd6960198ac0ad9ce080ac6869bd5a9fbe65ed131b1470dab2b4fb7f7a  
ubuntu@ip-172-31-45-114:~$ docker run -it --name container2 --link container1 -d  
ubuntu  
9ab39c46d329d67eb0f024077a8cc8765c2d84e3ab6b2801bb688d4a3beb81ef
```

Hands-on: Linking Docker Containers

Hands-on: Linking Docker Containers

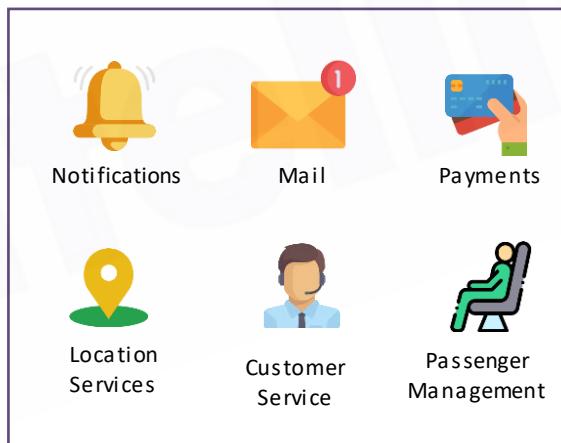
1. Create two containers of Ubuntu with names as follows: container1 and container2
2. Link container2 to container1
3. Try pinging from container2 to container1 by just using the command “ping container1”



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

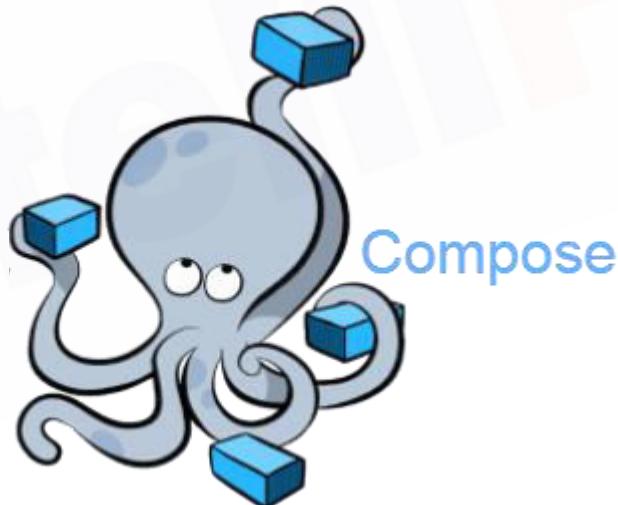


- ✓ Application is distributed, hence easy to understand.
- ✓ The code of only the Microservice which is supposed to be updated is changed.
- ✓ Bug, in one service, does not affect other services.
- ✓ There is no barrier to any specific technology.

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
```



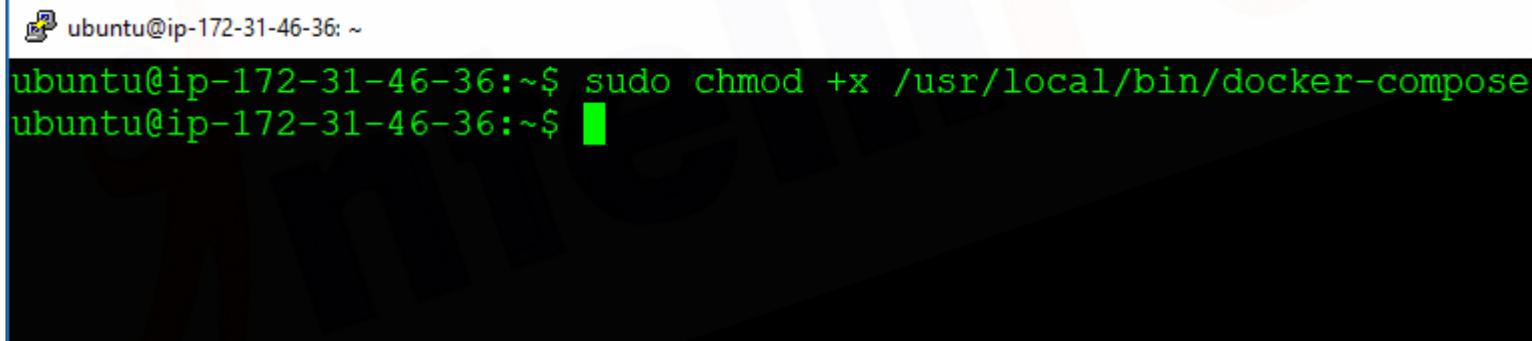
ubuntu@ip-172-31-46-36: ~

```
ubuntu@ip-172-31-46-36:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-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 title bar shows the session name "ubuntu@ip-172-31-46-36: ~". The terminal itself contains the command "sudo chmod +x /usr/local/bin/docker-compose" in green text, which has been entered by the user. The cursor is visible at the end of the command line.

```
ubuntu@ip-172-31-46-36:~$ sudo chmod +x /usr/local/bin/docker-compose
```

Installing Docker Compose

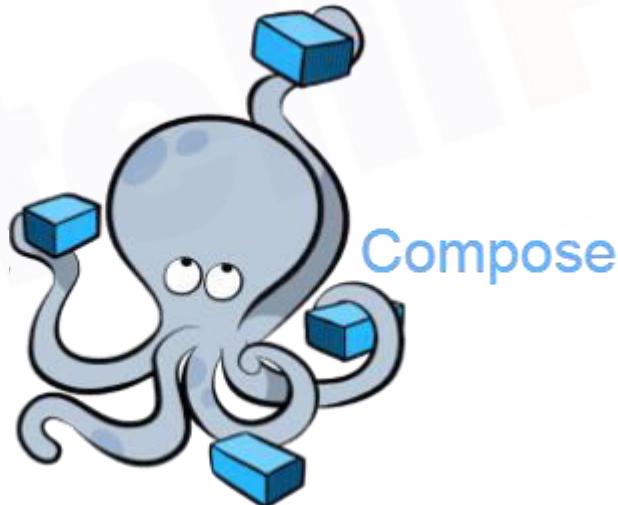
3. Finally, verify your installation using the following command:

```
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:

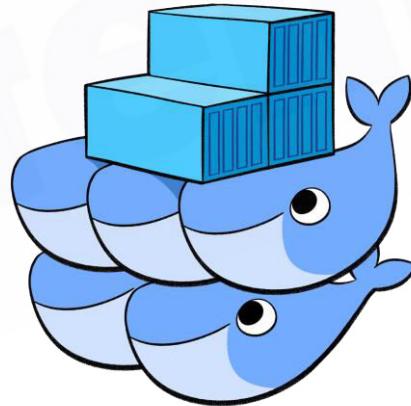
```
docker-compose up -d
```

4. Ensure that all your containers are running

What is Container Orchestration?

What is Container Orchestration?

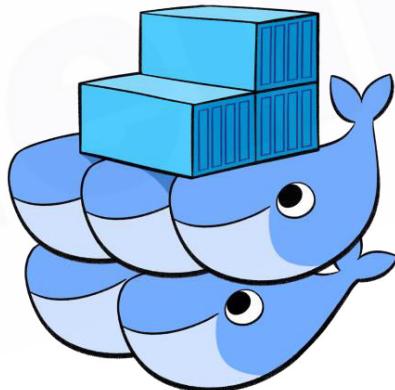
Applications are typically made up of individually containerized components (often called microservices) that must be organized at the networking level in order for the application to run as **intended**. The process of organizing multiple **containers** in this manner is known as **container orchestration**.



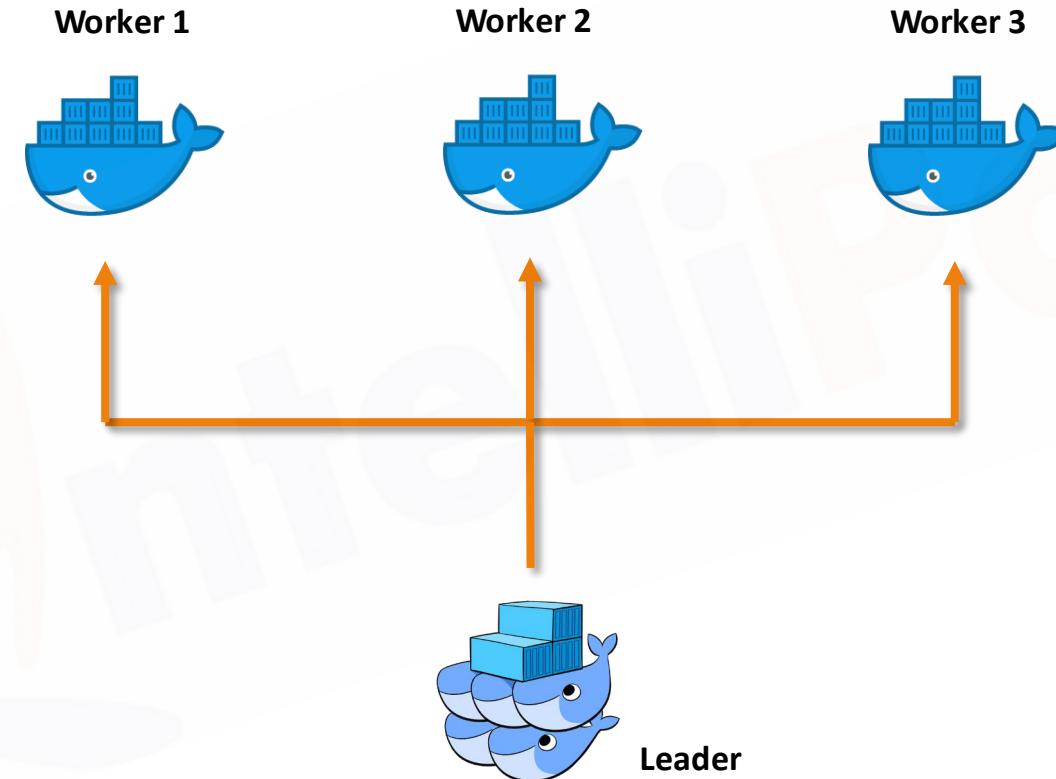
Introduction to Docker Swarm

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$ █
```

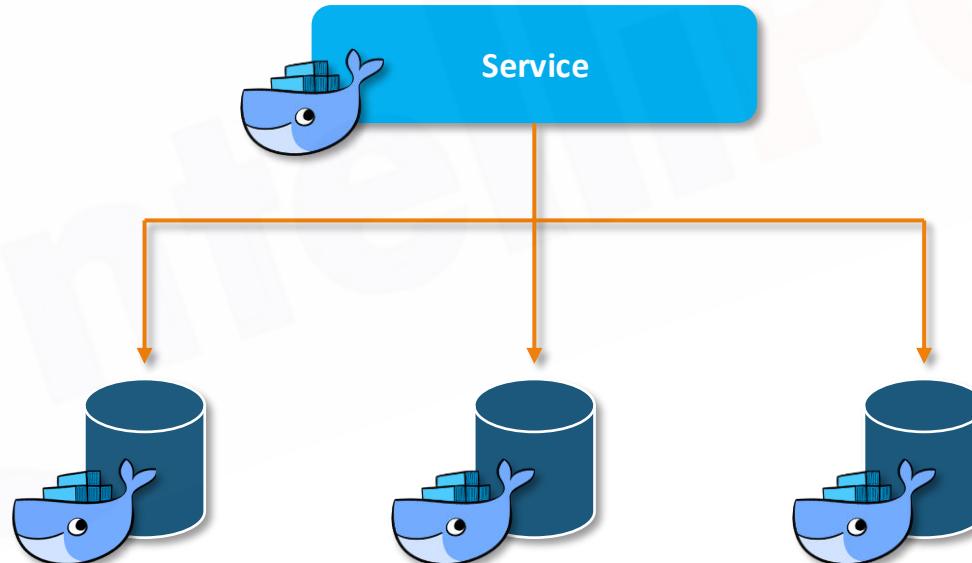
This command should be passed on to the worker node to join the docker swarm cluster.



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

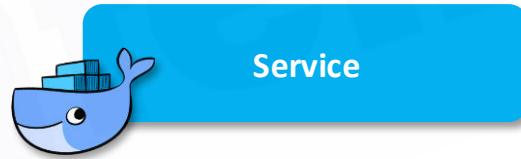
```
docker service create --name <name-of-service> --replicas <number-of-replicas> <image-name>
```

```
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

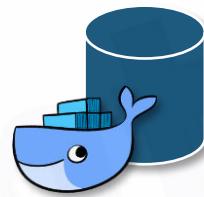
1. Create a Service for nginx webserver
2. There should be 3 replicas of this service running on the swarm cluster
3. Try accessing the service from Master IP and Slave IP



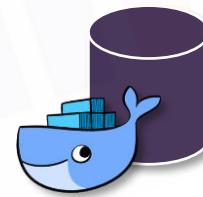
Docker Networks

Why Docker Networks?

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



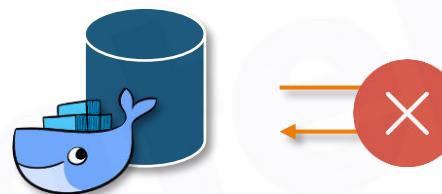
Website Container



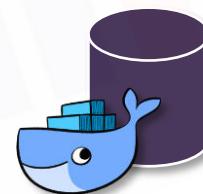
Database Container

Why Docker Networks?

By default, these containers cannot communicate with each other.



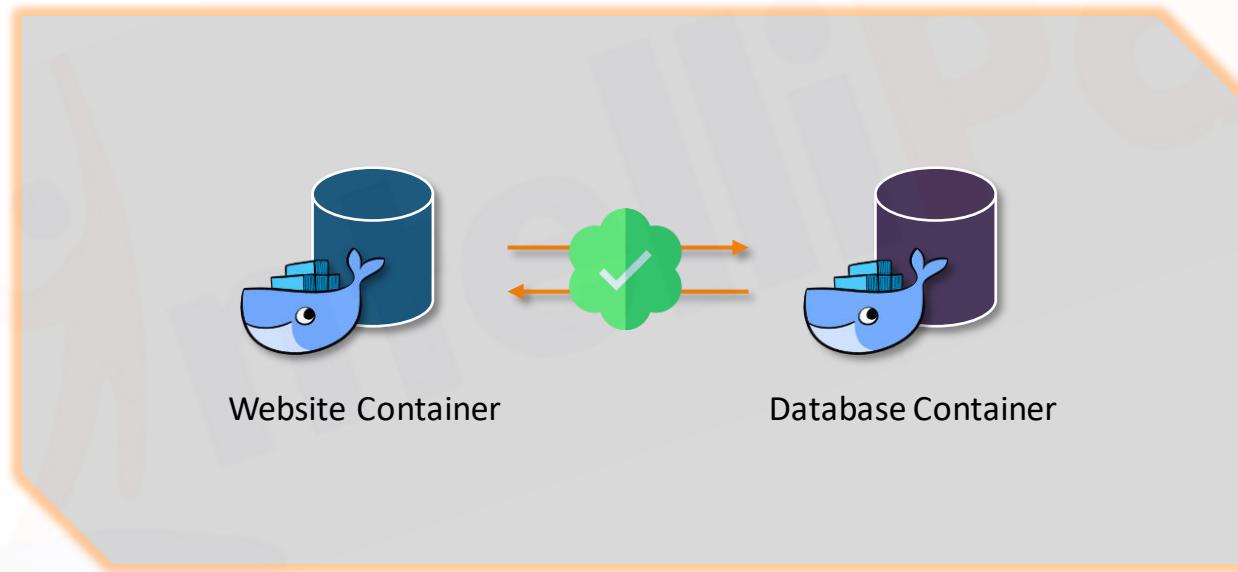
Website Container



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

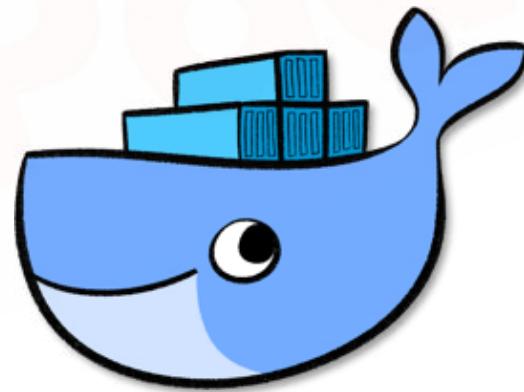
None

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

Quiz

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

A. True

B. False



India: +91-7847955955



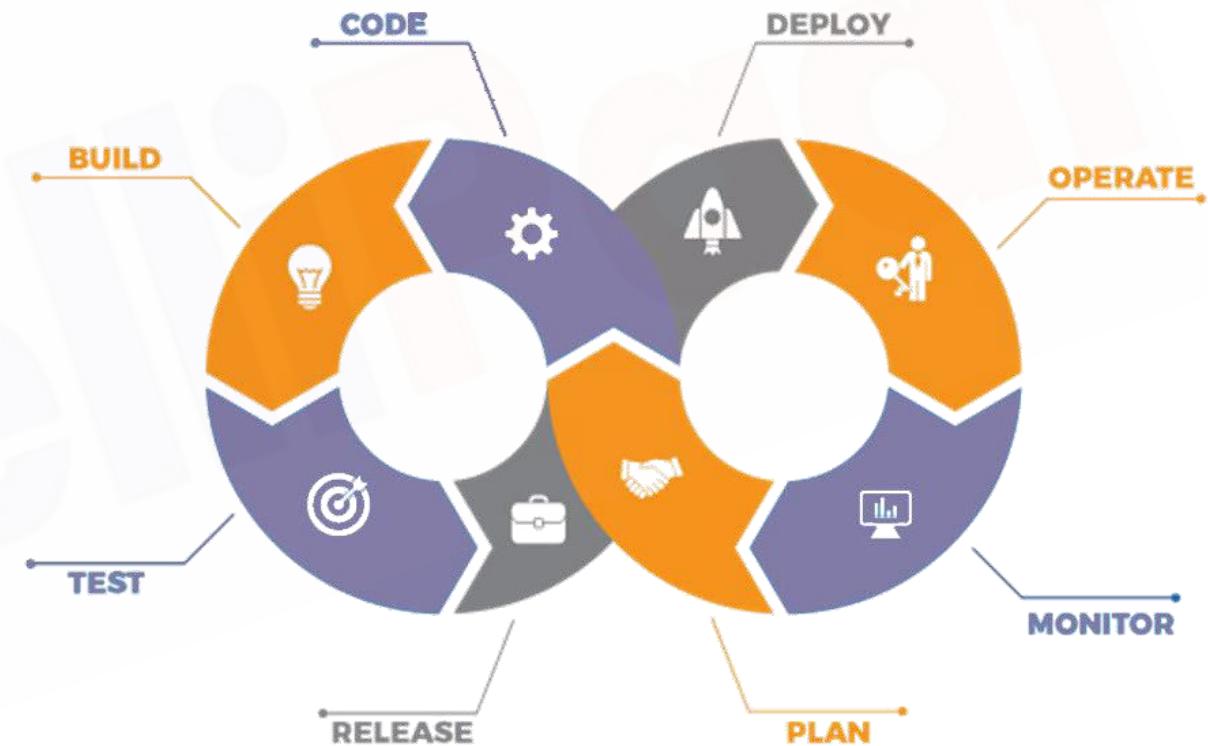
US: 1-800-216-8930 (TOLL FREE)



support@intellipaat.com

24/7 Chat with Our Course Advisor

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

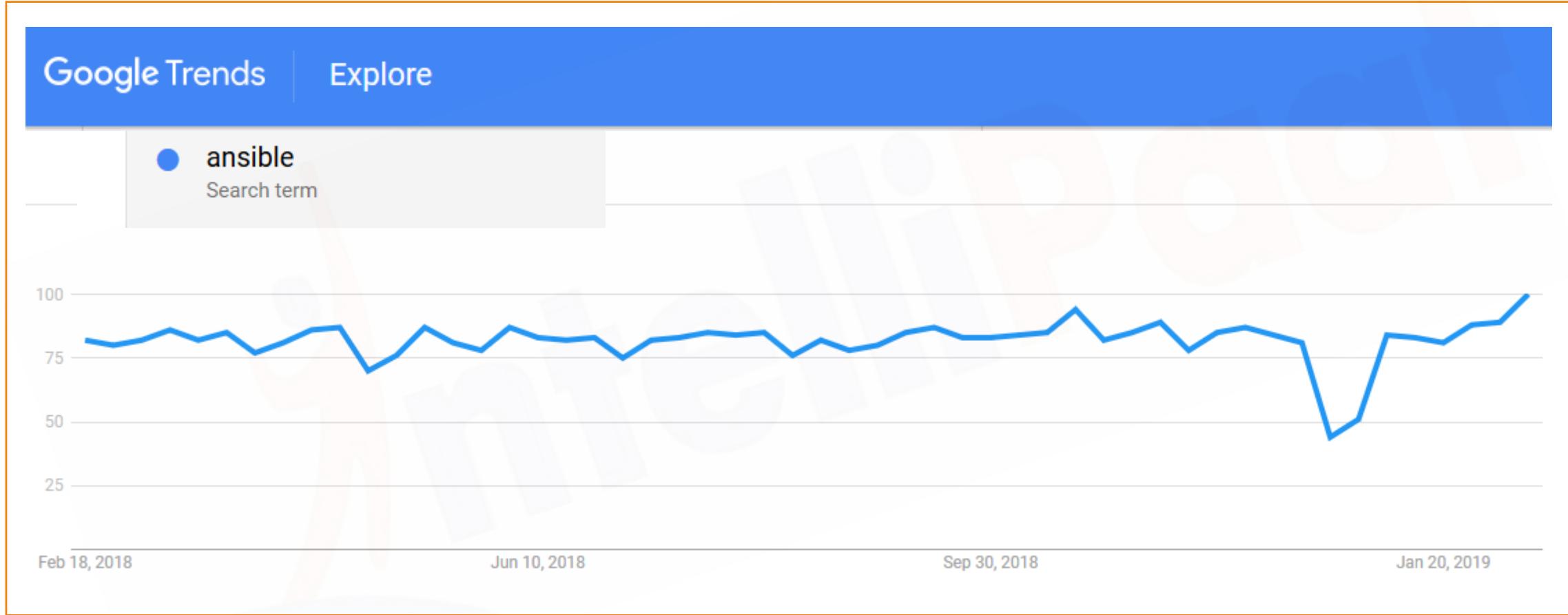


ANSIBLE



Why Ansible?

Why Ansible?



Google Trends Results for Ansible

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®

Advantage of Ansible

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

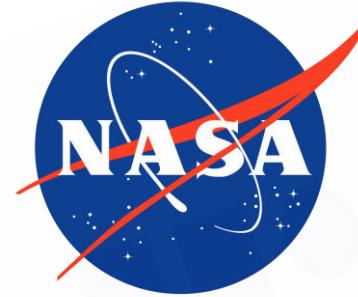


ANSIBLE

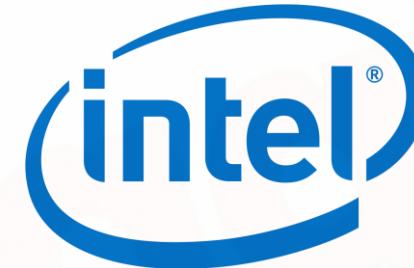
Popularity of Ansible



Apple



NASA



Intel



Percussion



Cisco



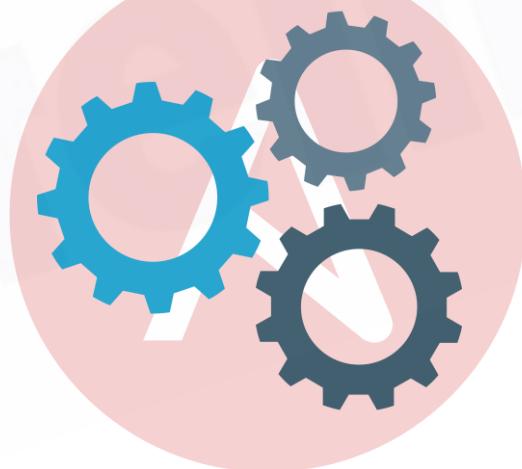
Twitter

How does Ansible work?

How does Ansible work?

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

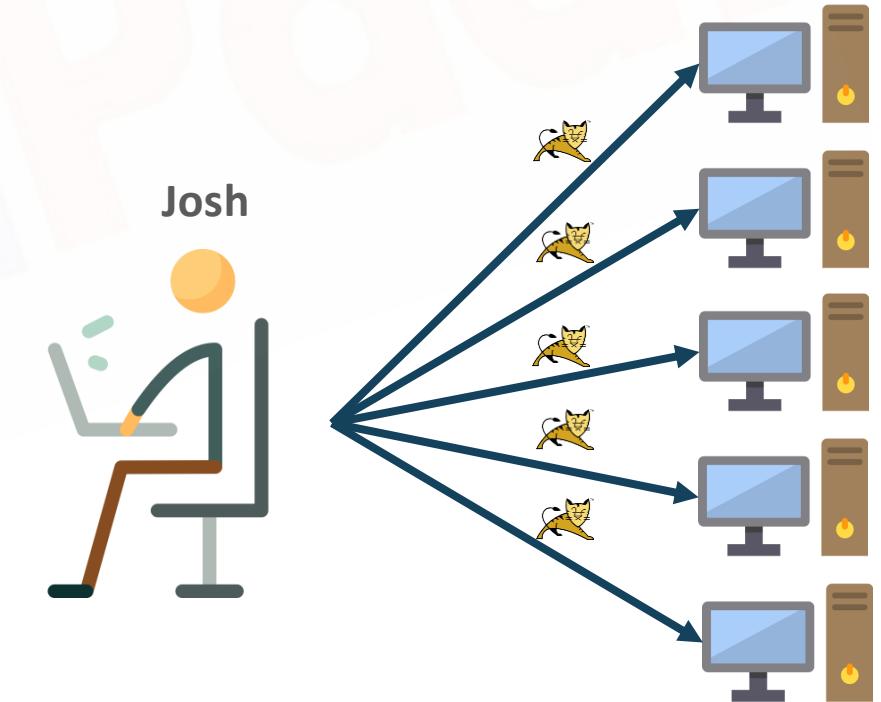
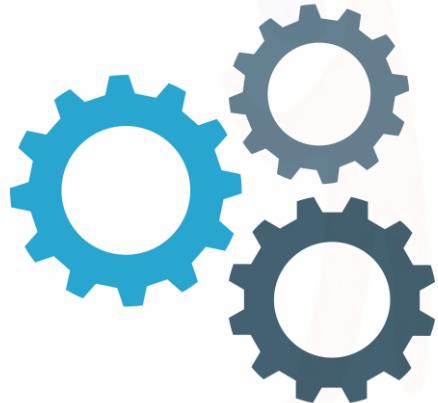
Configuration Management



Problem Statement

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

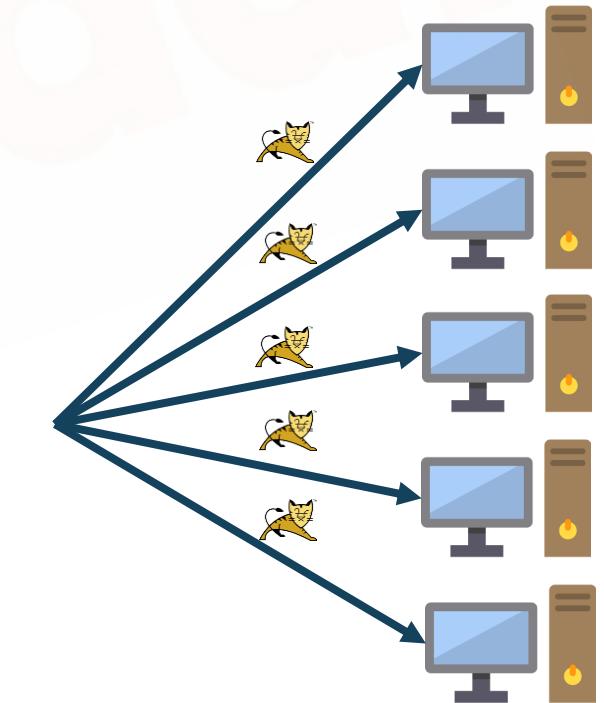
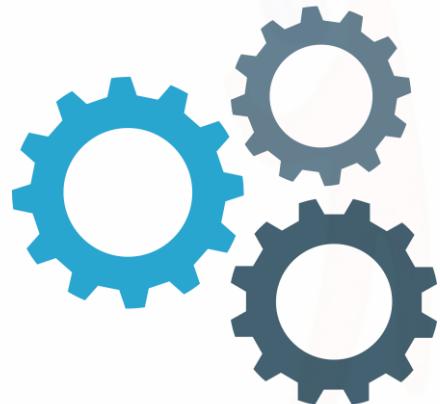
Configuration Management



Problem Statement-Solution with Ansible

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

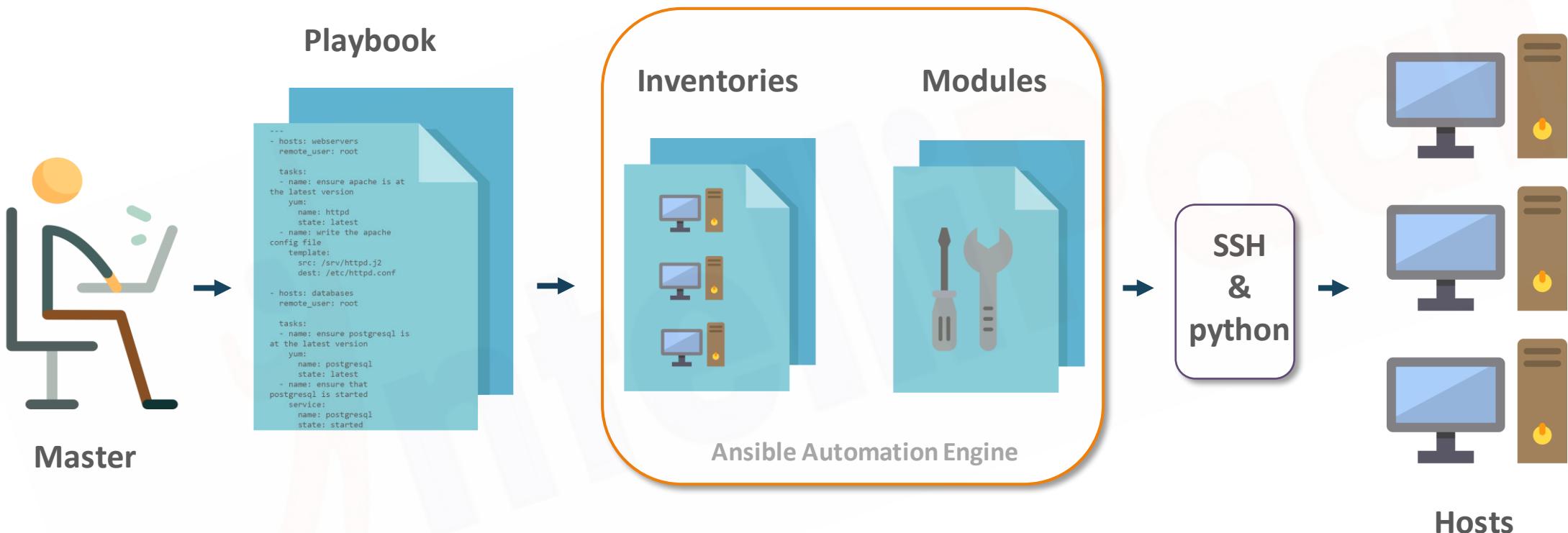
Configuration Management





Ansible Architecture

Ansible Architecture

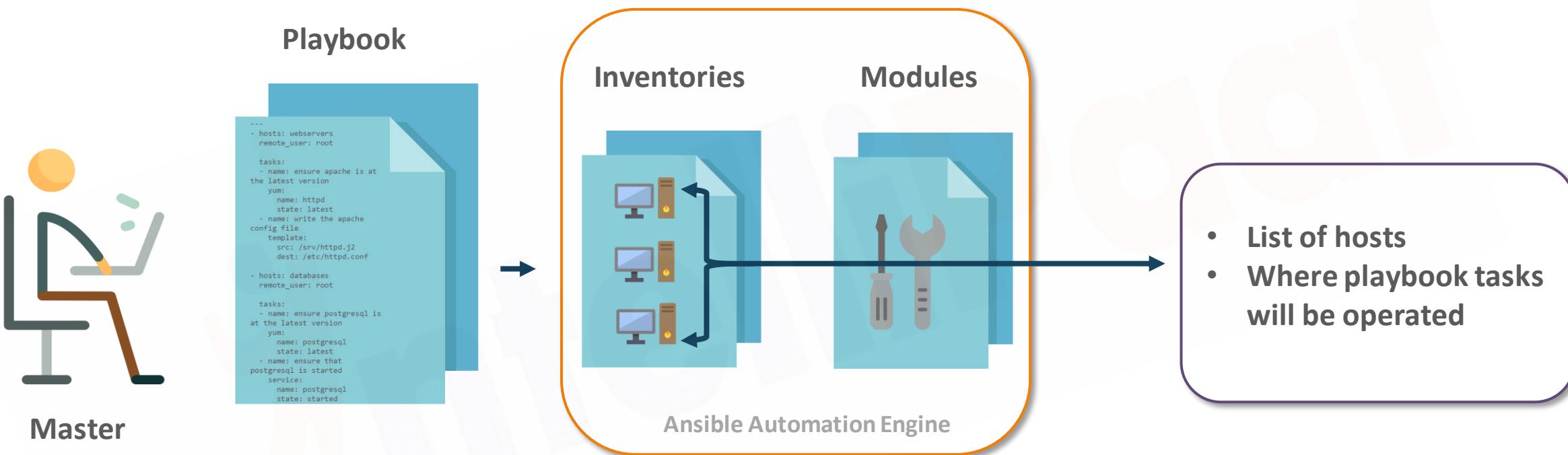


Basic Ansible Architecture

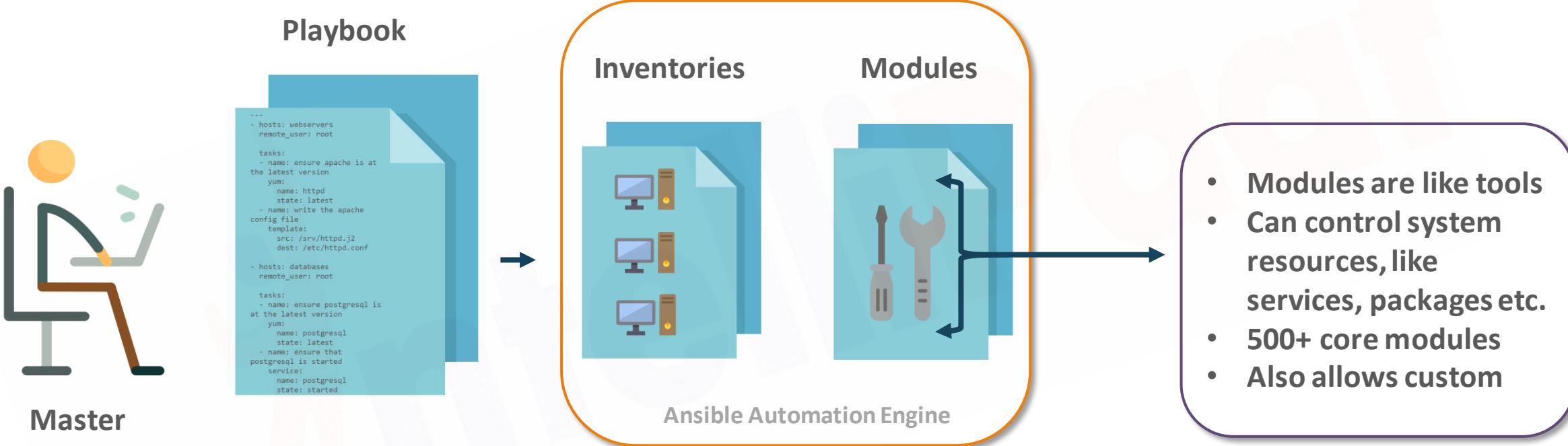
Ansible Architecture- Master



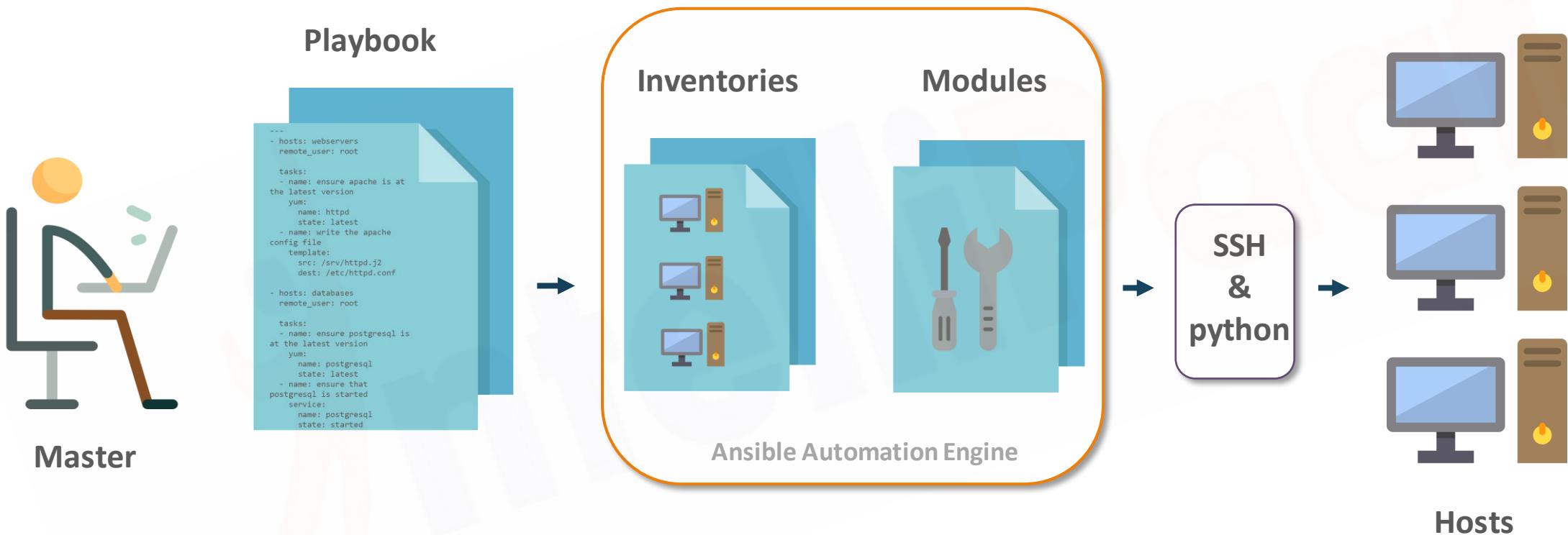
Ansible Architecture- Inventories



Ansible Architecture- Modules



Ansible Architecture- Hosts





Case Study: Ansible being used in NASA

Case Study- Business Challenge

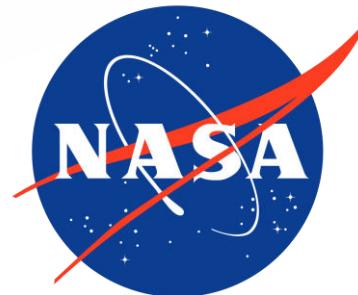
NASA needed to move roughly 65+ applications from a Traditional Hardware Based Data Center to Cloud Based Environment for better agility and cost saving



Traditional Hardware Based Data Center



Cloud Based Environment



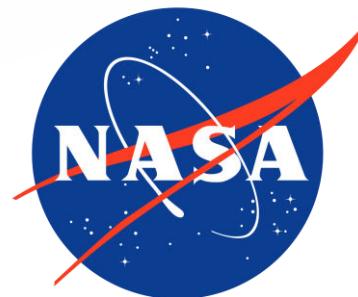
Case Study- Solution

NASA used Ansible to manage and schedule the cloud environment



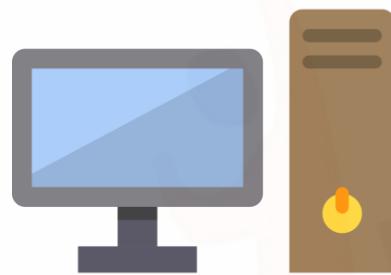
Traditional Hardware Based Data Center

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



ANSIBLE



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?

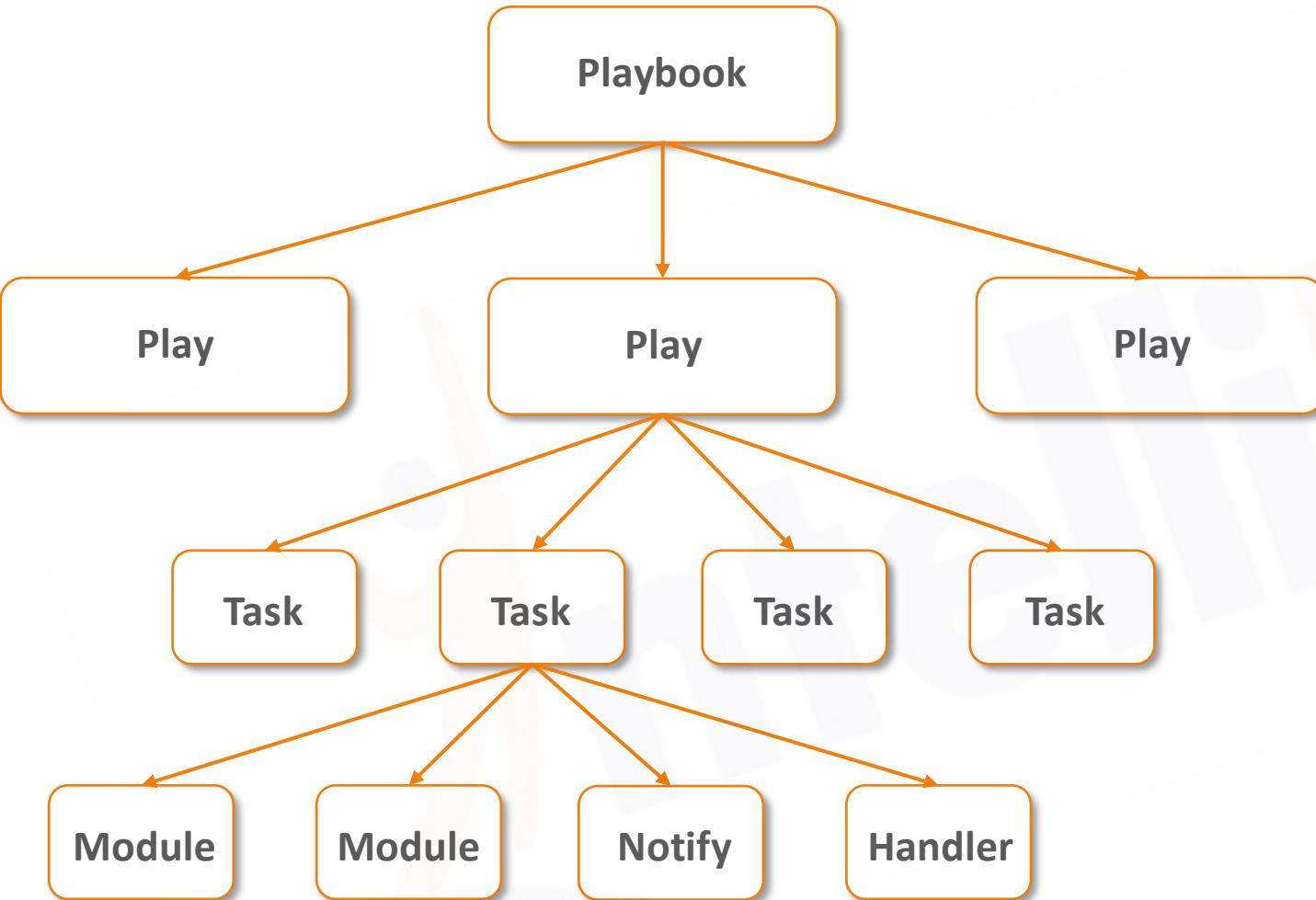
An organized unit of scripts
Defines work for a server configuration
Written in **YAML**

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

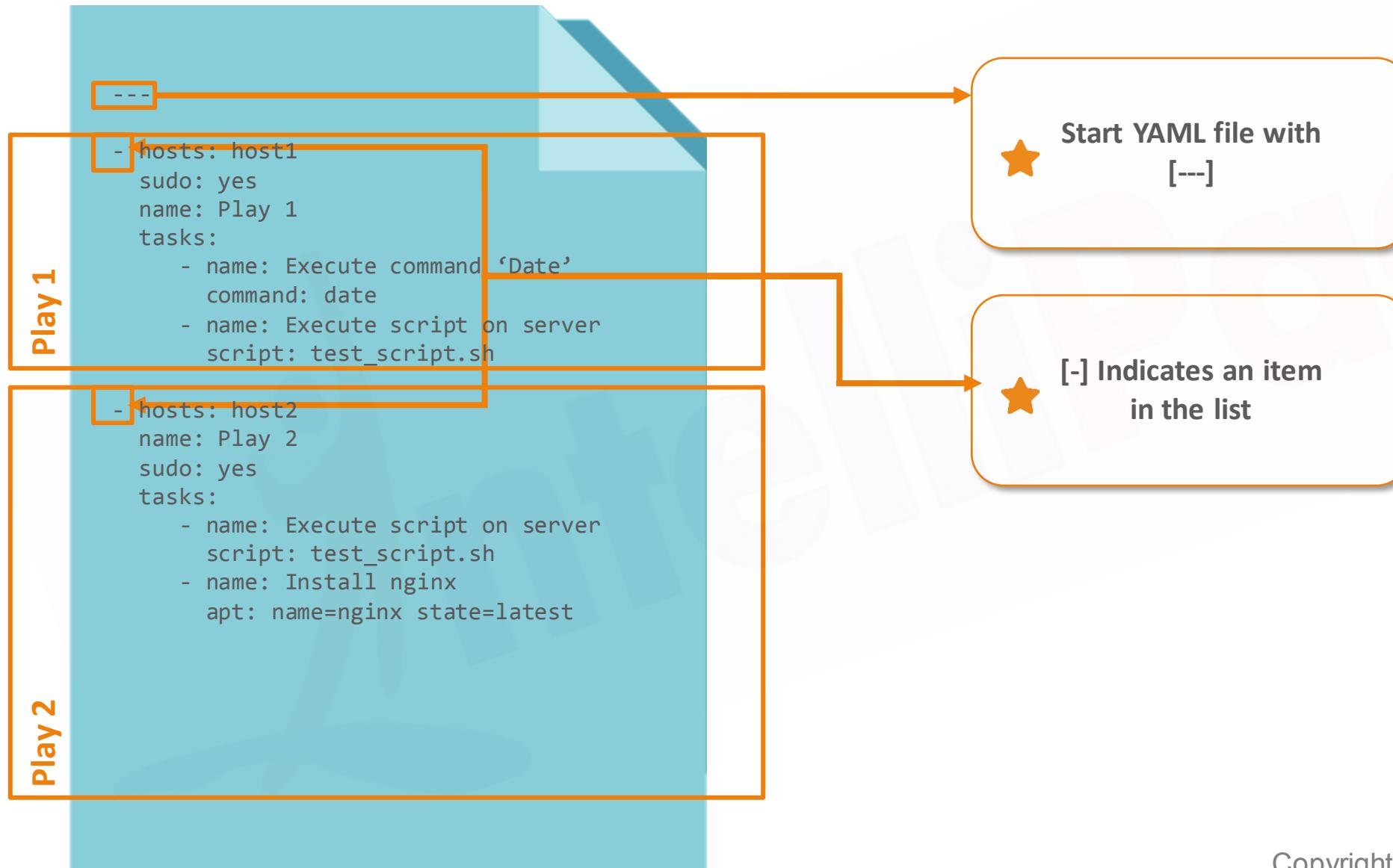
4 Install nginx in host2

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



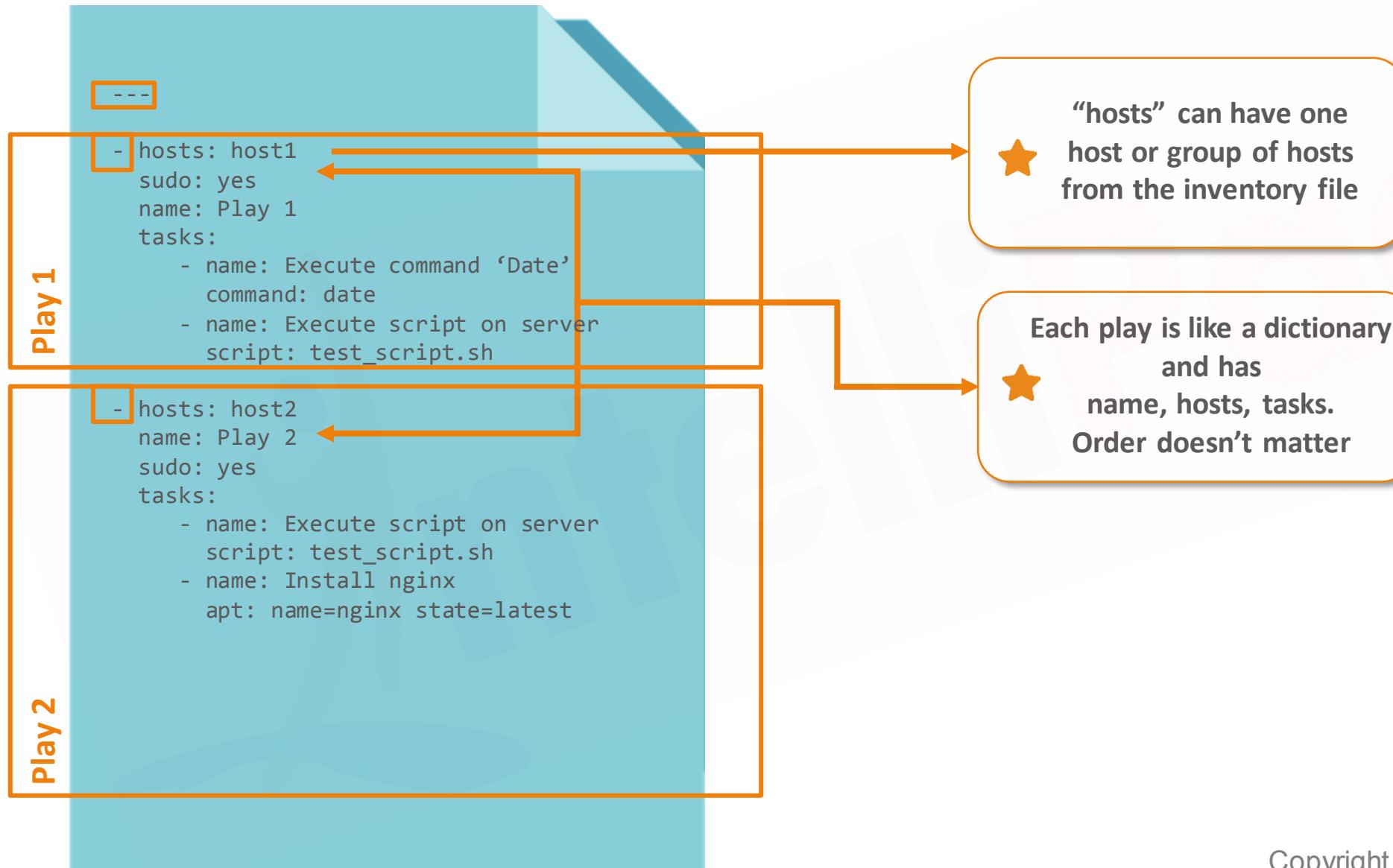
Creating Ansible Playbook-Example

```
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
```

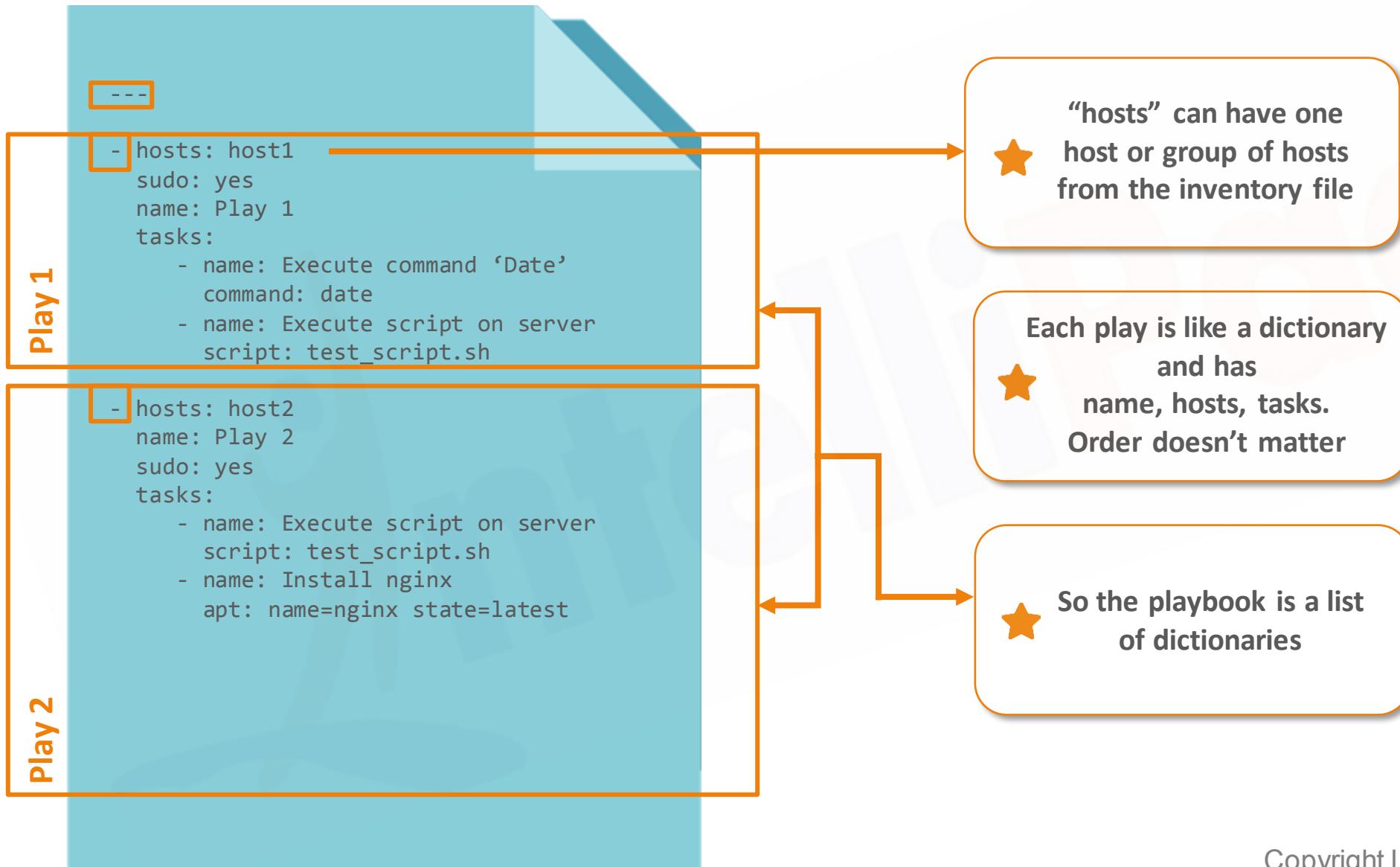
★ “hosts” can have one host or group of hosts from the inventory file /etc/ansible/hosts

```
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
```

Creating Ansible Playbook-Example



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]

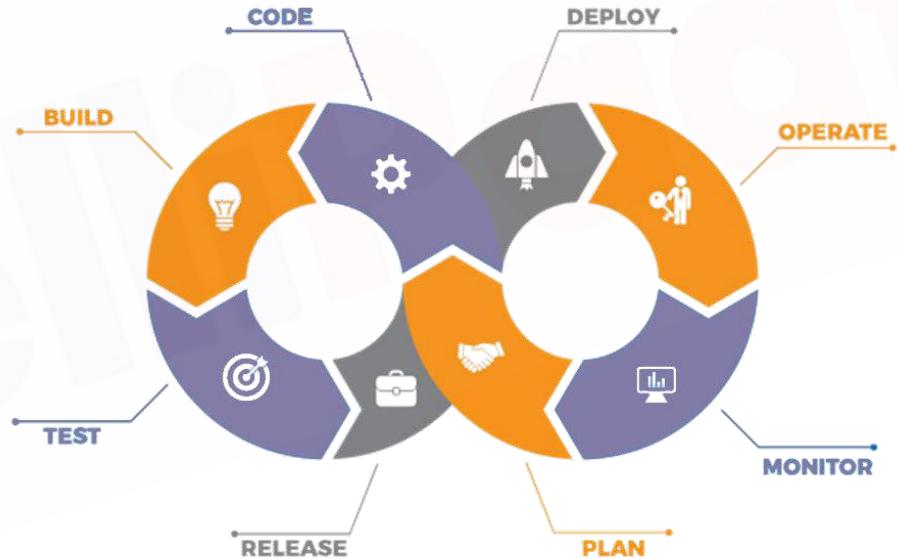
TASK [apache : send copy.html file] ****
ok: [host1]
ok: [host2]

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



Hands-on: Configuring Multiple Nodes using Ansible

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?

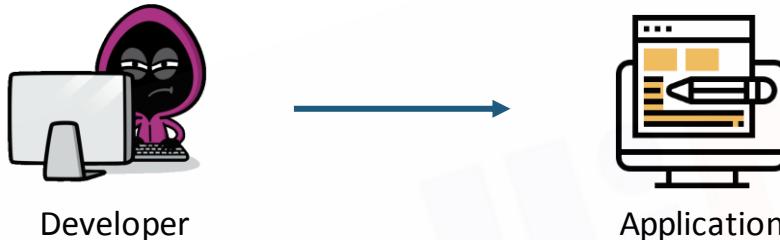


Developer

Why Configuration Management?



Why Configuration Management?

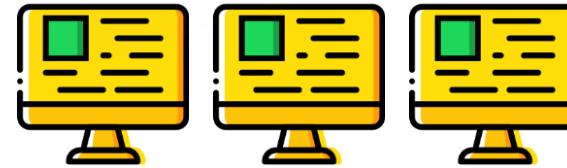


PHP 5.7
MySQL 4.7

Application



PHP Servers



Database Servers

Why Configuration Management?



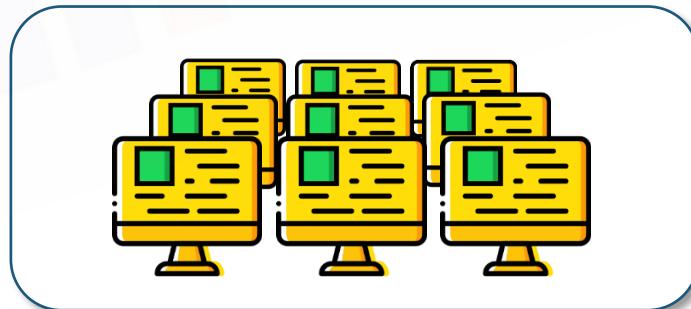
Developer

Application

PHP 5.7
MySQL 4.7



PHP 5.7 Servers



Database 4.7 Servers

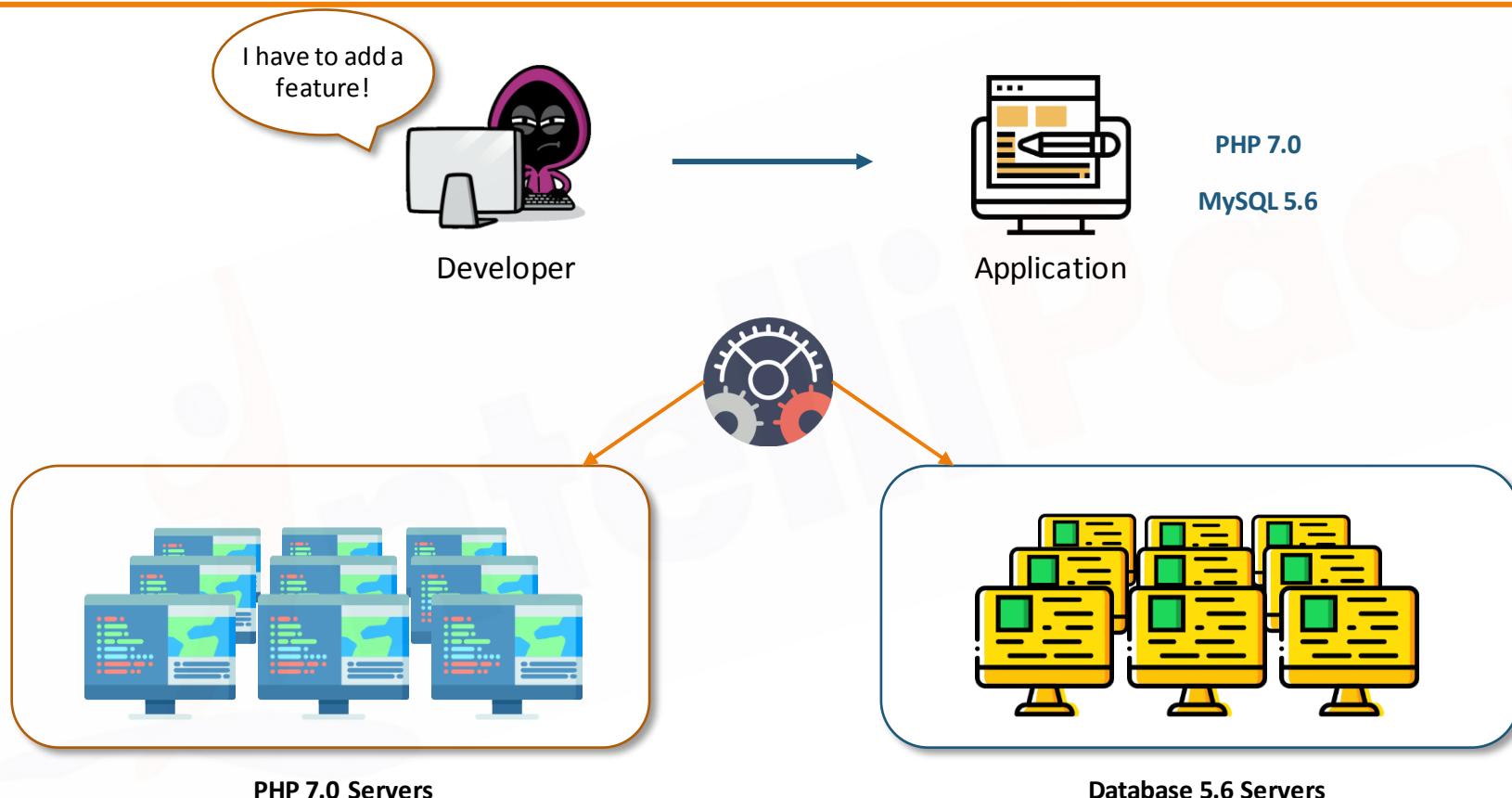
What is Configuration Management?

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

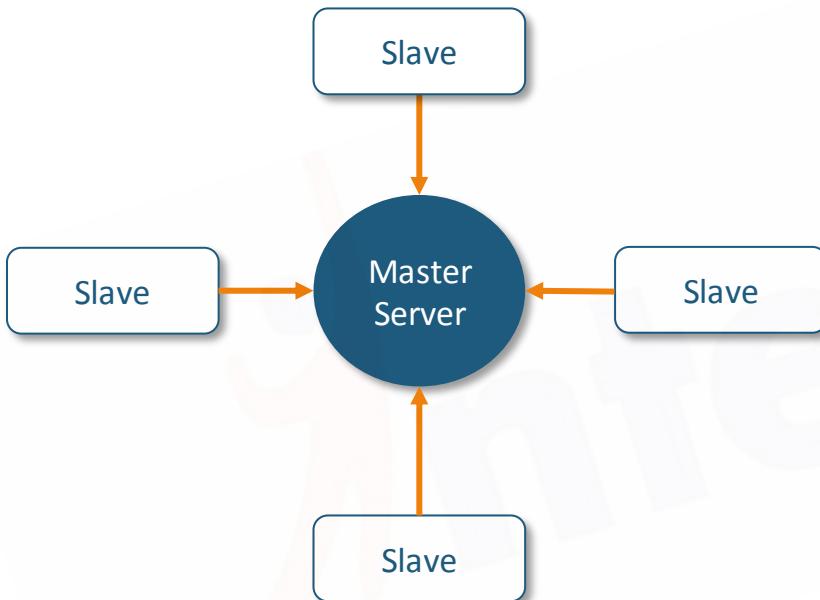
- ★ Automation
- ★ Consistency
- ★ Software Updates
- ★ Software Rollback



Configuration Management Tools

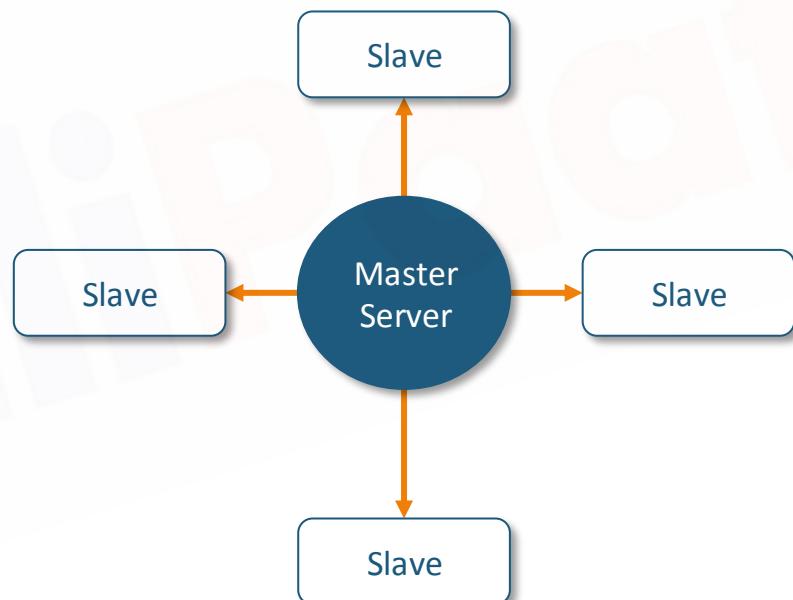
Types of Configuration Management Tools

Pull Configuration



Changes are pulled

Push Configuration



Changes are pushed

Types of Configuration Management Tools

Pull Configuration



Push Configuration



ANSIBLE



SALTSTACK

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

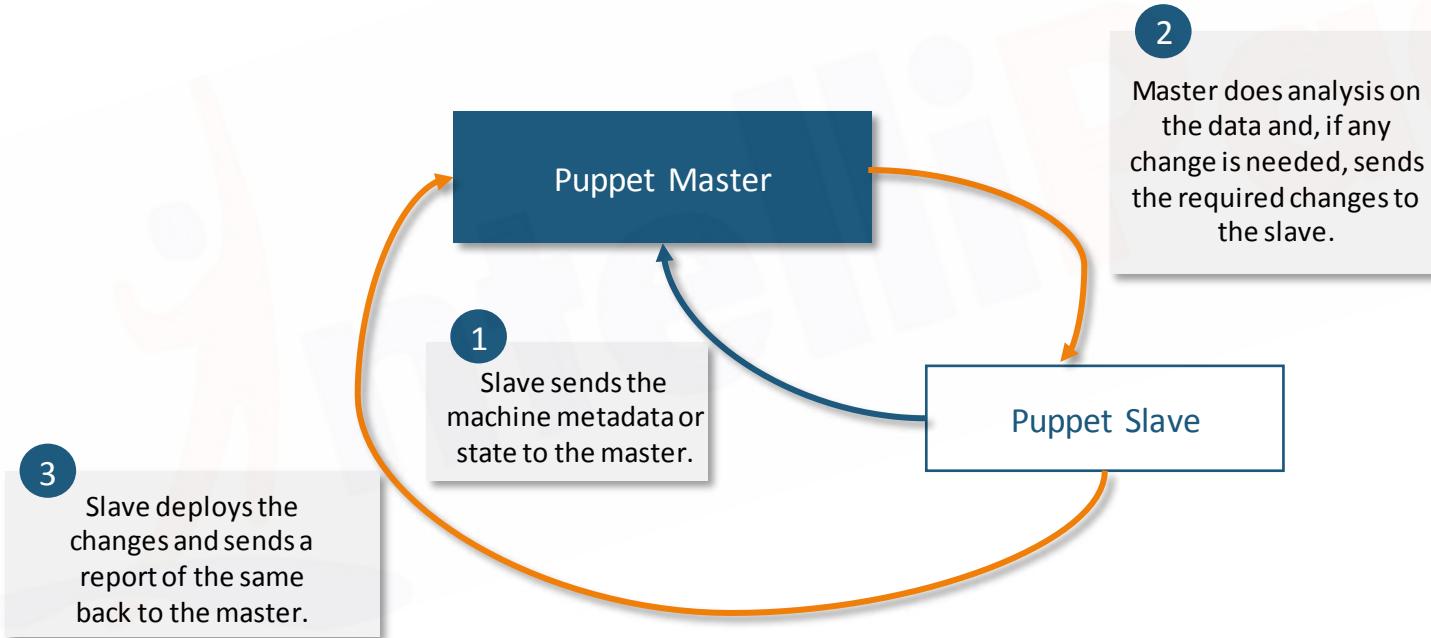
- ★ Large User Base
- ★ Big Open-source Community
- ★ Documentation
- ★ Platform Support



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

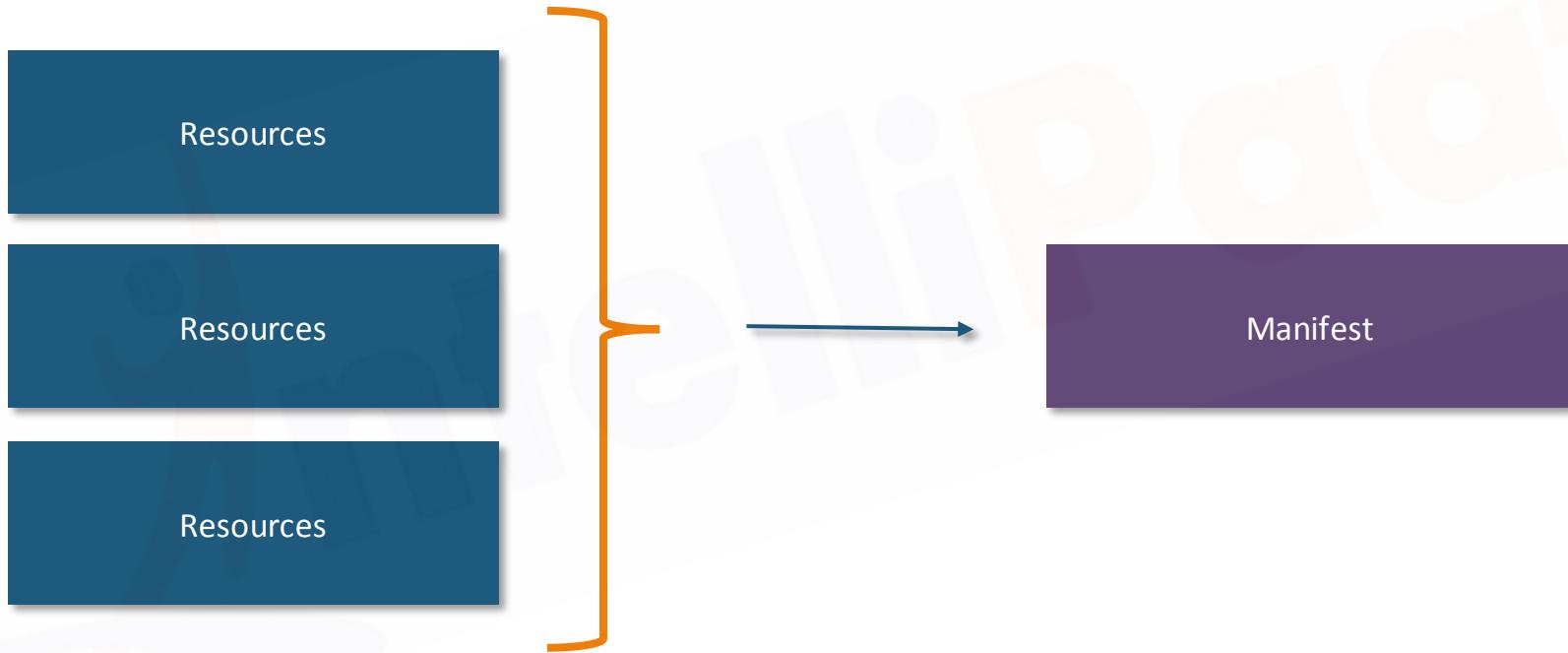


Example

```
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



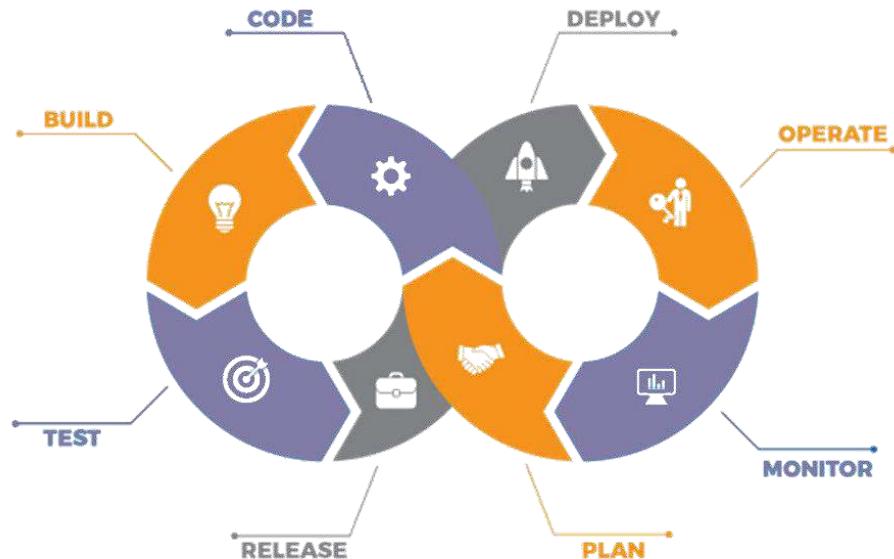
US: 1-800-216-8930 (TOLL FREE)



sales@intellipaat.com

24/7 Chat with Our Course Advisor

Introduction to 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 21st 2015
- ★ It is the ninth most active repository on GitHub in terms of number of commits

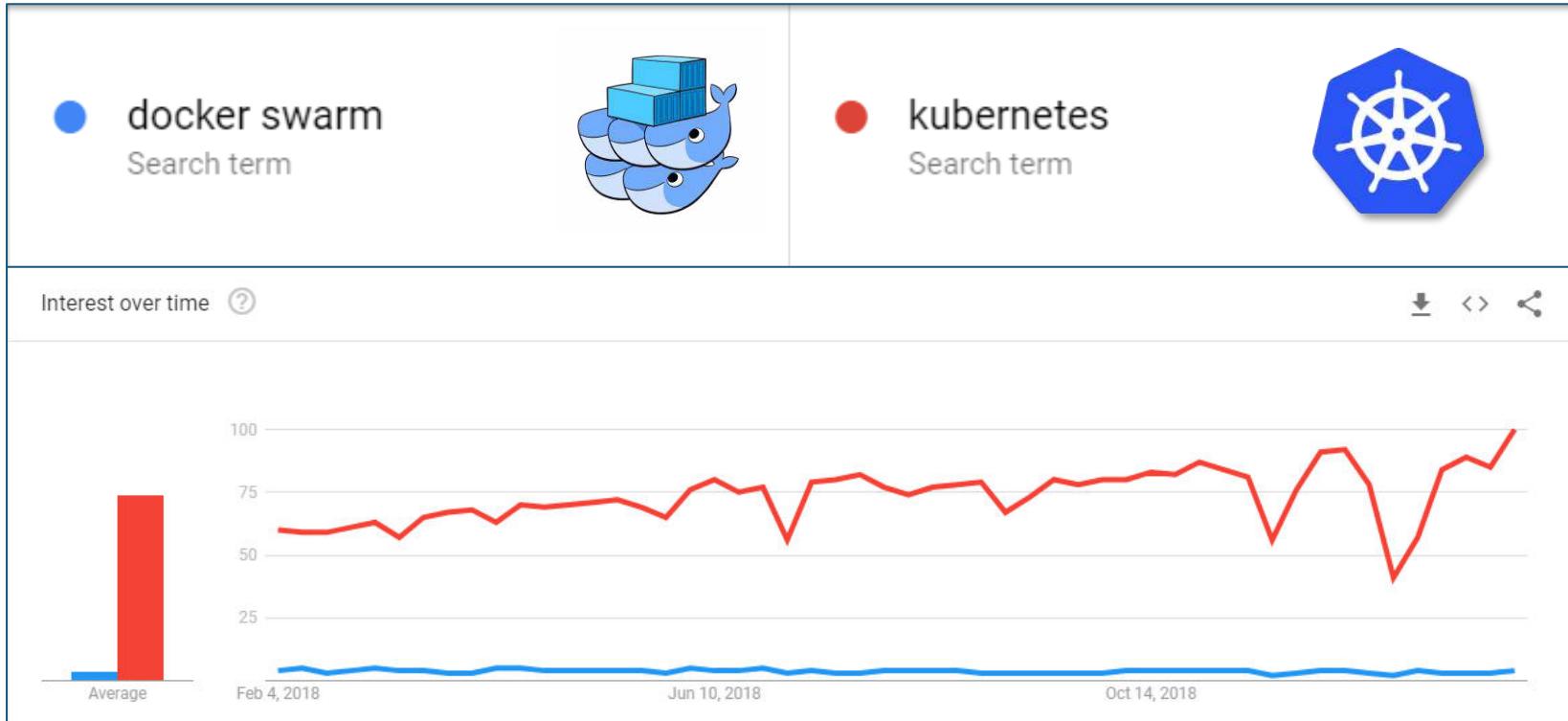
Features of Kubernetes

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



Docker Swarm vs Kubernetes

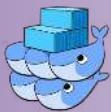
Docker Swarm vs Kubernetes



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

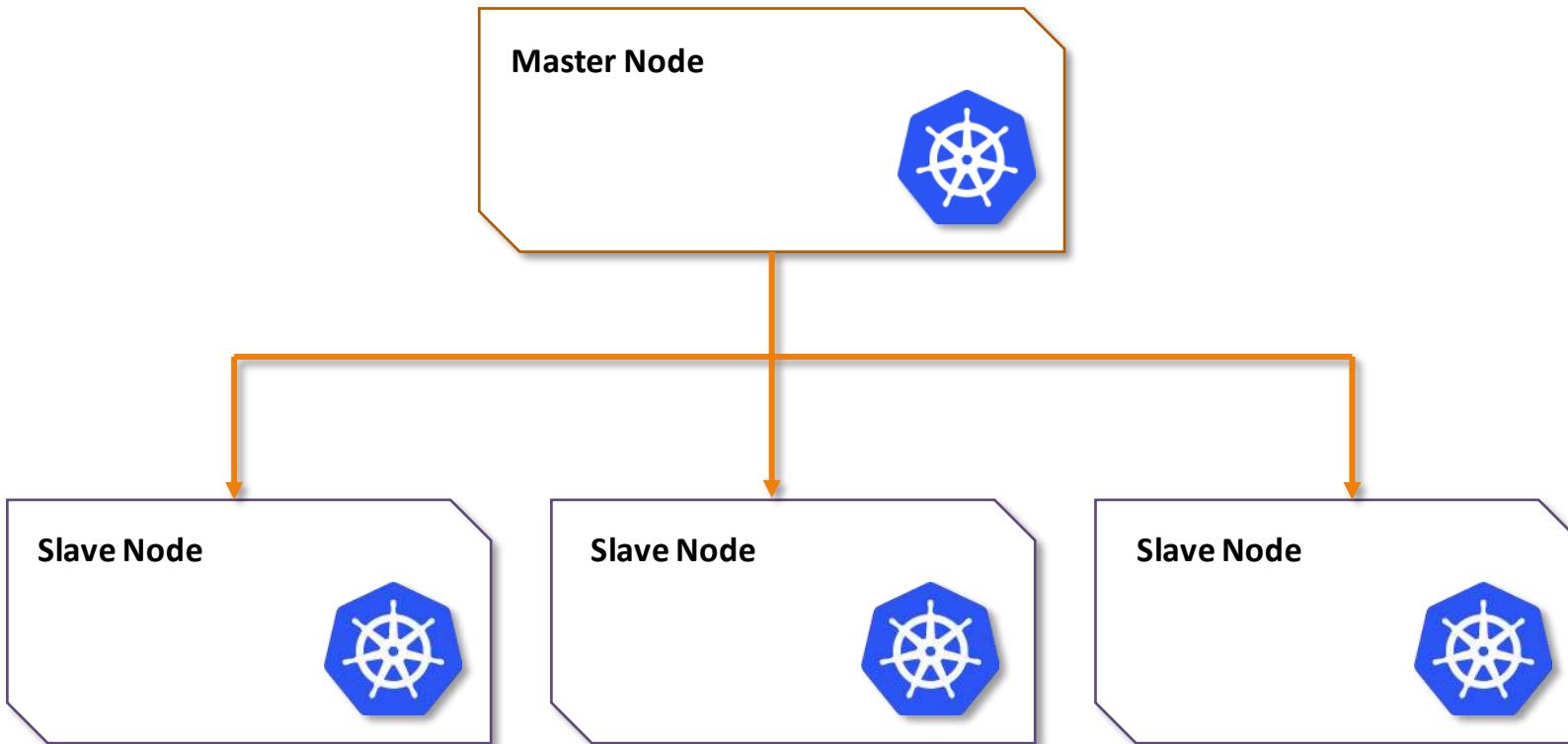
Kubernetes



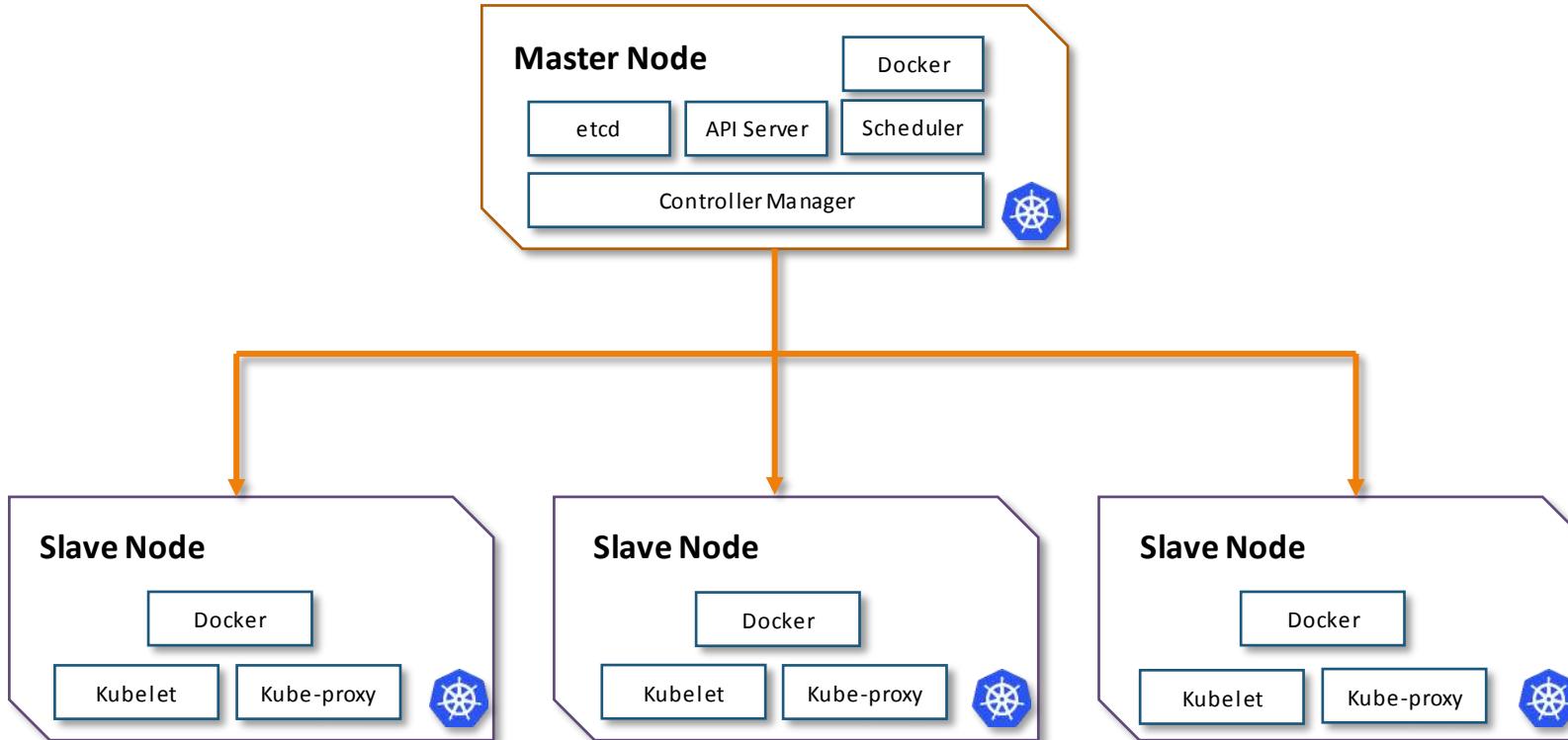
- ★ Complex Procedure to install Kubernetes
- ★ Slower when compared with Docker Swarm
- ★ More Reliable and comparatively has more features

Kubernetes Architecture

Kubernetes Architecture



Kubernetes Architecture



Kubernetes Architecture – Master Components

Kubernetes Architecture – Master Components



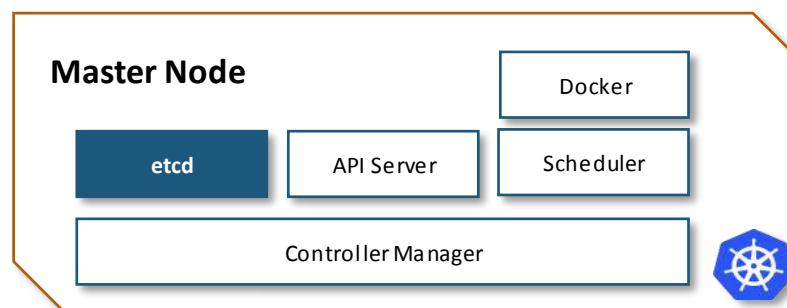
etcd

API Server

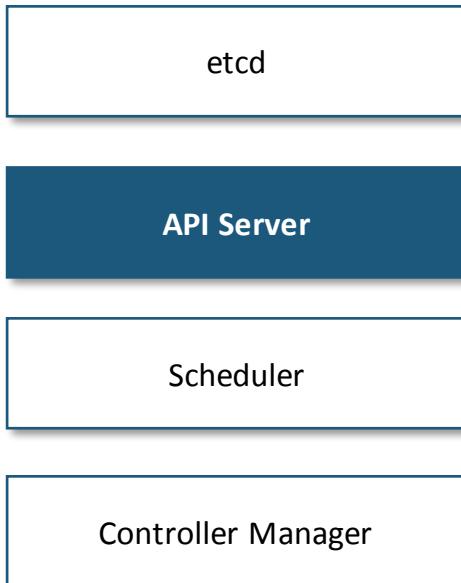
Scheduler

Controller Manager

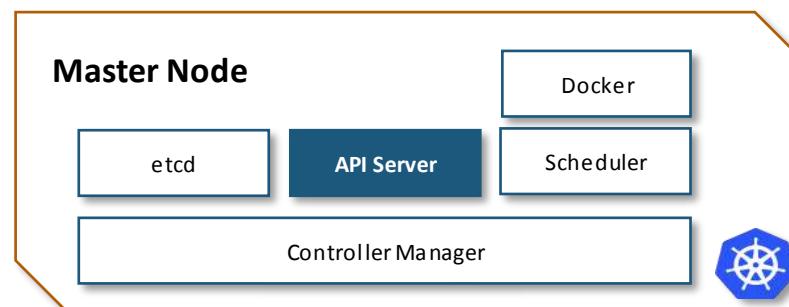
It is a highly available distributed key value store, which is used to store cluster wide secrets. It is only accessible by Kubernetes API server, as it has sensitive information.



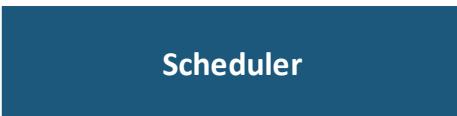
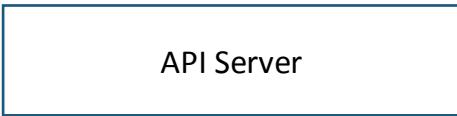
Kubernetes Architecture – Master Components



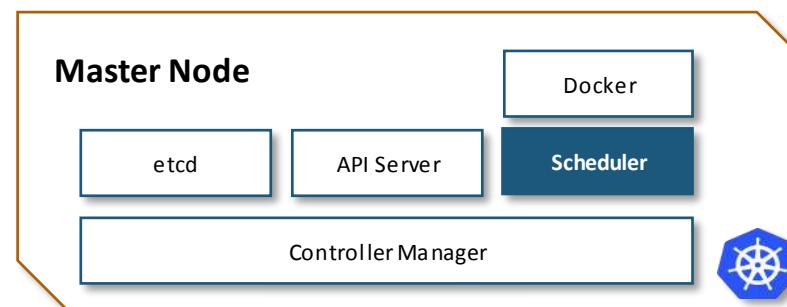
It exposes the Kubernetes API. The Kubernetes API is the front-end for Kubernetes Control Plane, and is used to deploy and execute all operations in Kubernetes



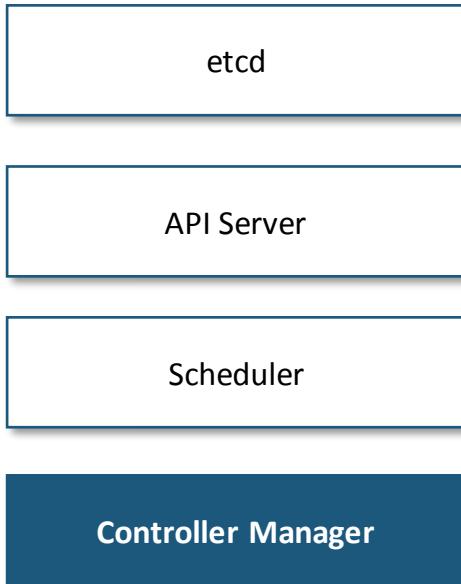
Kubernetes Architecture – Master Components



The scheduler takes care of scheduling of all the processes, Dynamic Resource Management and manages present and future events on the cluster

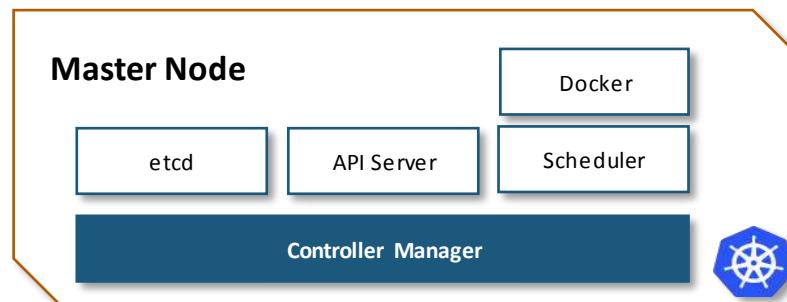


Kubernetes Architecture – Master Components



The controller manager, runs all the controllers on the Kubernetes Cluster. Although each controller, is a separate process, but to reduce complexity, all the 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 – Master Components



Kubelet

Kube-Proxy

Kubelet takes the specification from the API server, and ensures the application is running according to the specifications which were mentioned.
Each node has it's kubelet service

Slave Node

Docker

Kubelet

Kube-proxy



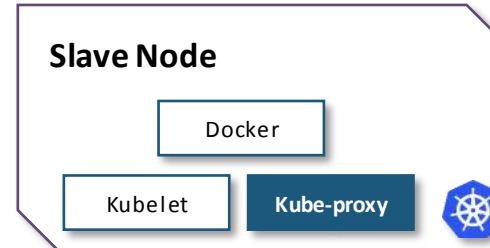
Kubernetes Architecture – Master Components



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



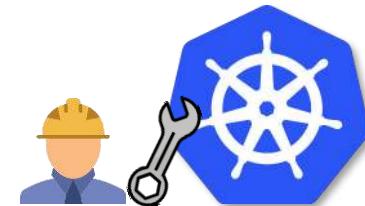


Kubernetes Installation

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



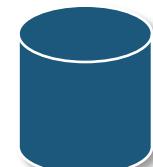
Hands-on: Installing Kubernetes using kubeadm

Working of Kubernetes

Working of Kubernetes



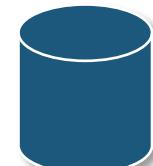
Pods can have one or more containers coupled together. They are the basic unit of Kubernetes. To increase High Availability, we always prefer pods to be in replicas



Pod – Replica 1



Pod – Replica 2

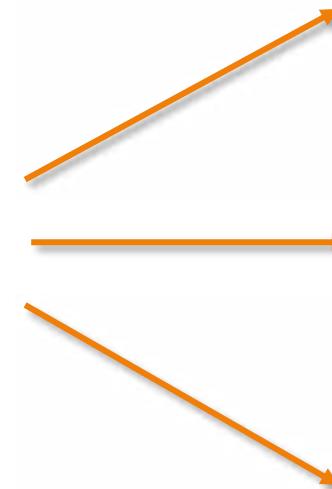


Pod – Replica 3

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

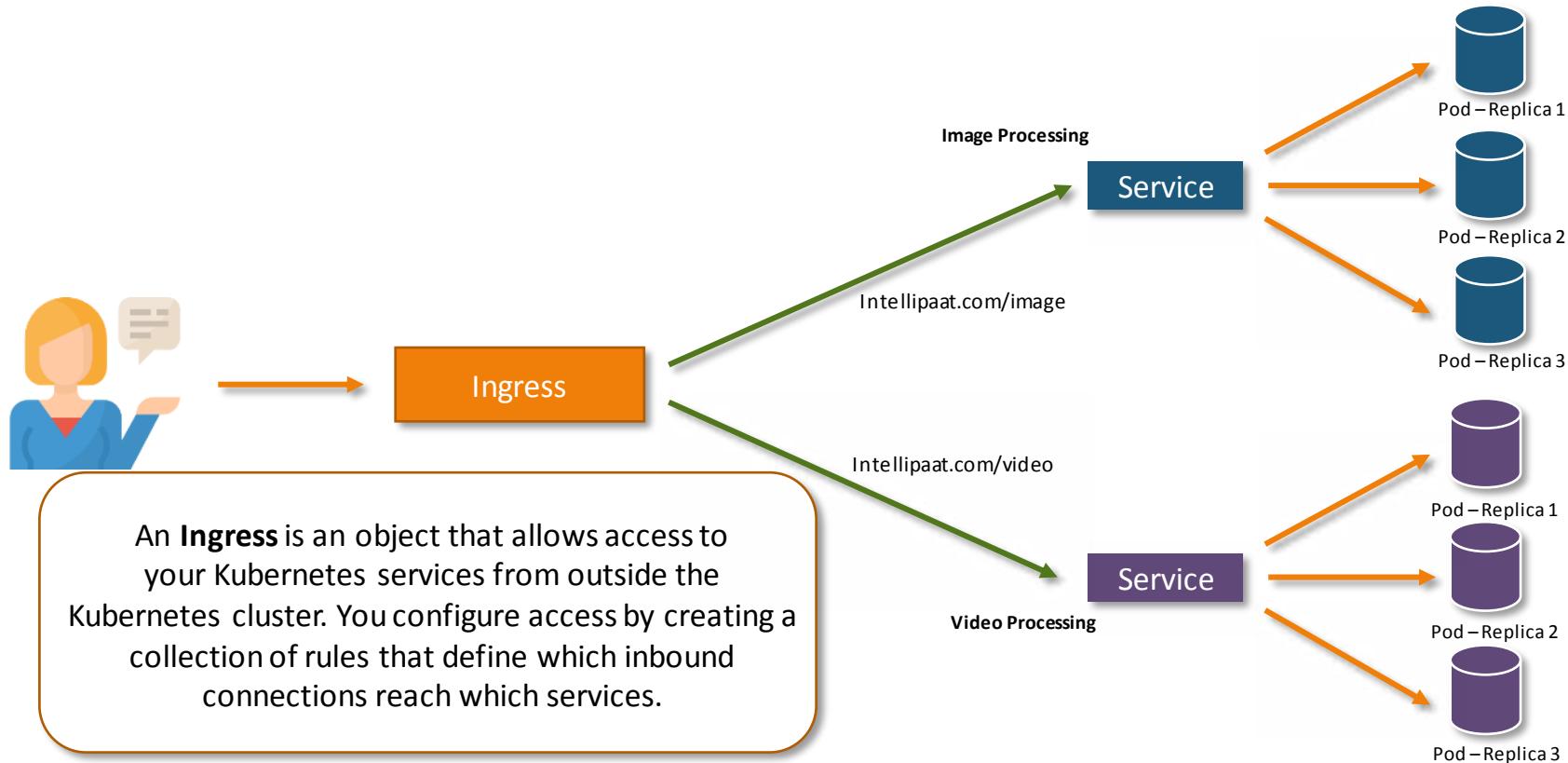


Pod – Replica 2



Pod – Replica 3

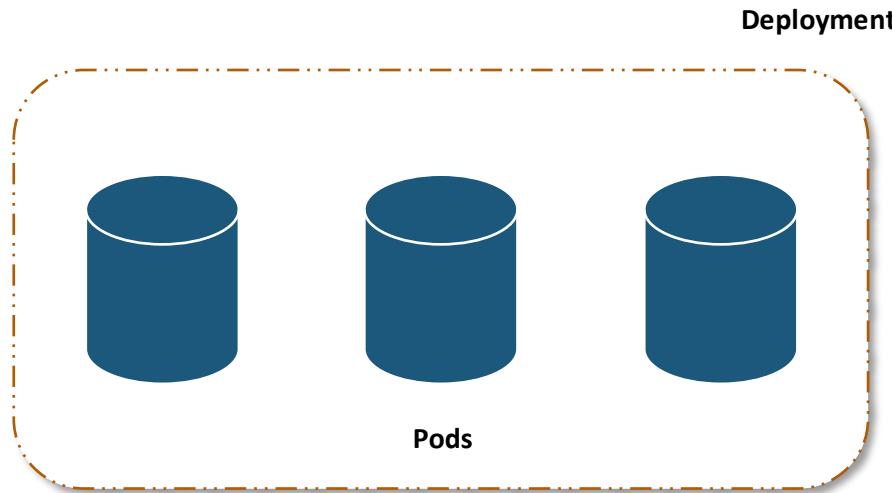
Working of Kubernetes



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 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:~$ █
```

List 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:~$
```

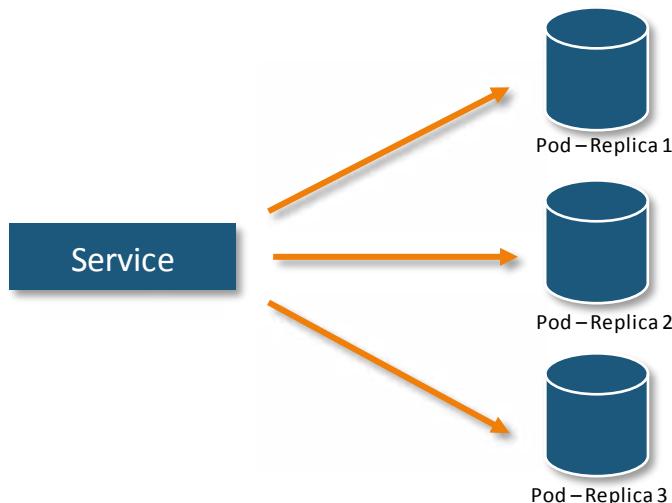
As you can see, the number of pods are matching with the number of replicas specified in the deployment file



Creating a Service

Creating a Service

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



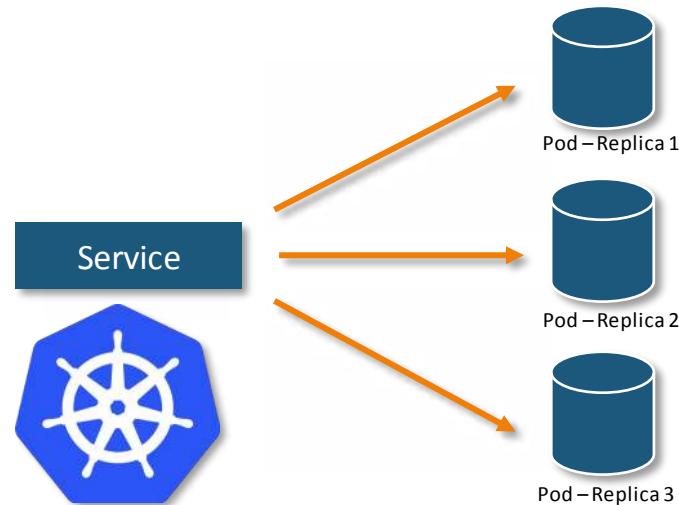
Service Types

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 contents of the ExternalName



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: ~
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 a NodePort Service

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

Syntax

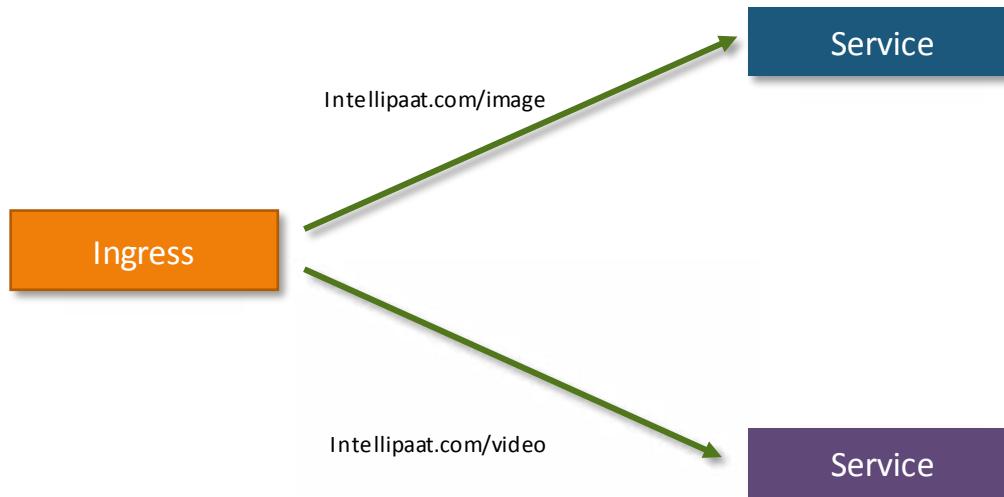
```
kubectl get svc nginx
```

```
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:~$
```

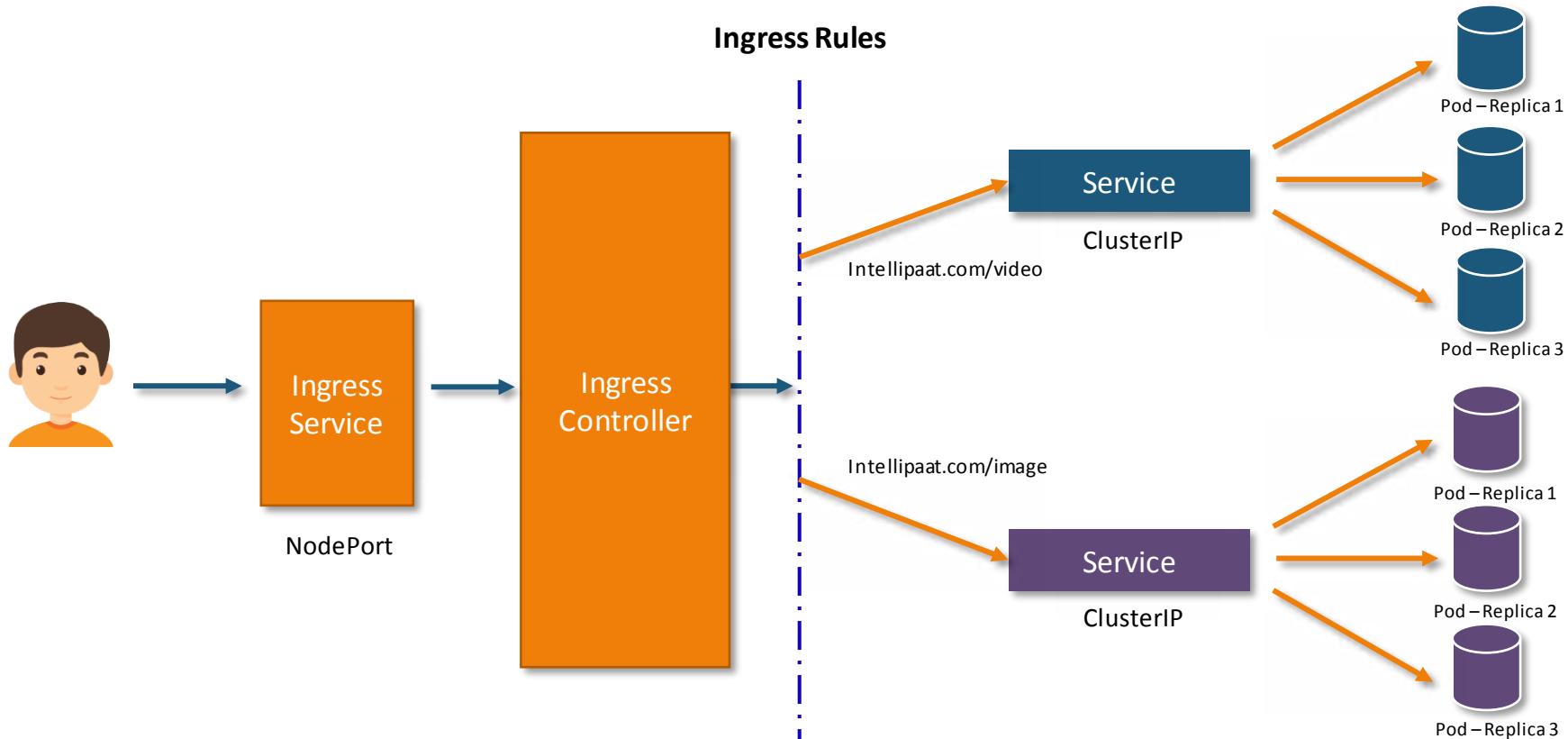
Creating an Ingress

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>



Define Ingress Rules

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

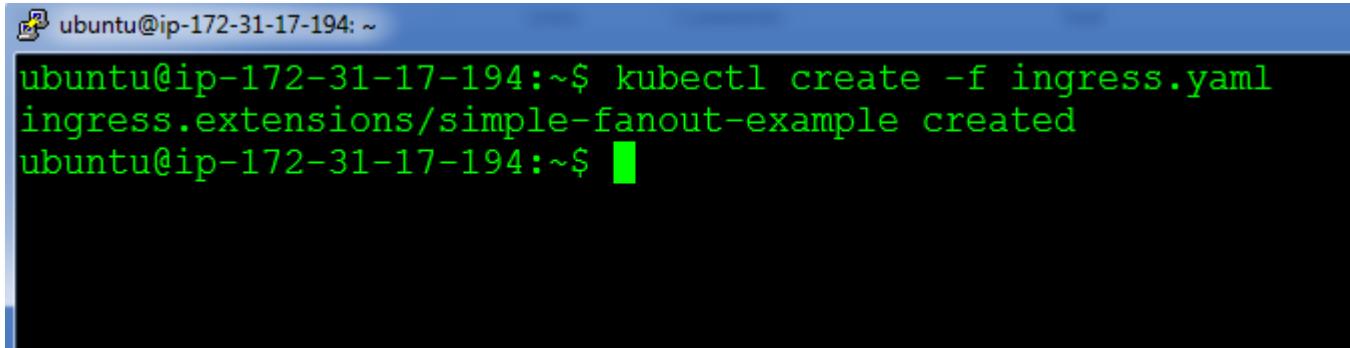
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /foo
        backend:
          serviceName: nginx
          servicePort: 80
```

Deploying Ingress Rules

To deploy the ingress rules, we use the following syntax:

Syntax

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

A screenshot of a terminal window on an Ubuntu system. The title bar shows the session is running on 'ubuntu@ip-172-31-17-194: ~'. The command 'kubectl create -f ingress.yaml' is entered, followed by the output: 'ingress.extensions/simple-fanout-example created'.

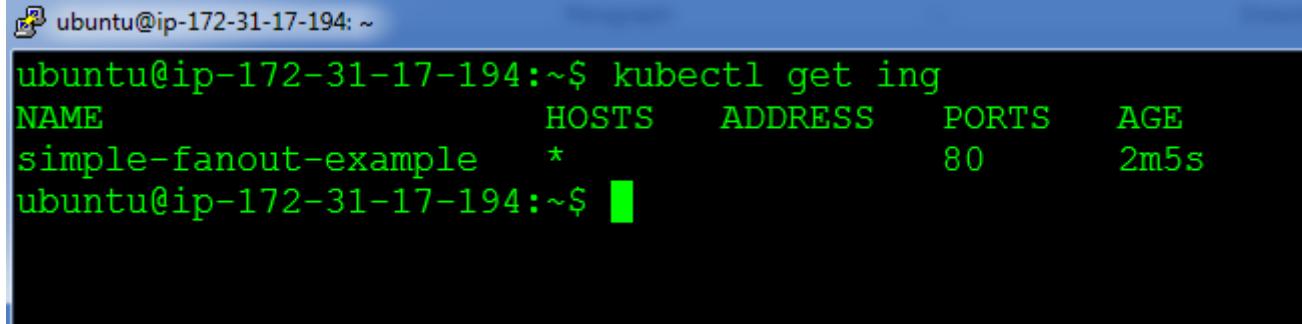
```
ubuntu@ip-172-31-17-194:~$ kubectl create -f ingress.yaml
ingress.extensions/simple-fanout-example created
ubuntu@ip-172-31-17-194:~$ █
```

Viewing Ingress Rules

To deploy the ingress rules, we use the following syntax:

Syntax

```
kubectl get ing
```



A terminal window with a blue header bar. The header bar contains a user icon and the text "ubuntu@ip-172-31-17-194: ~". The main area of the terminal shows the command "kubectl get ing" being run, followed by its output:

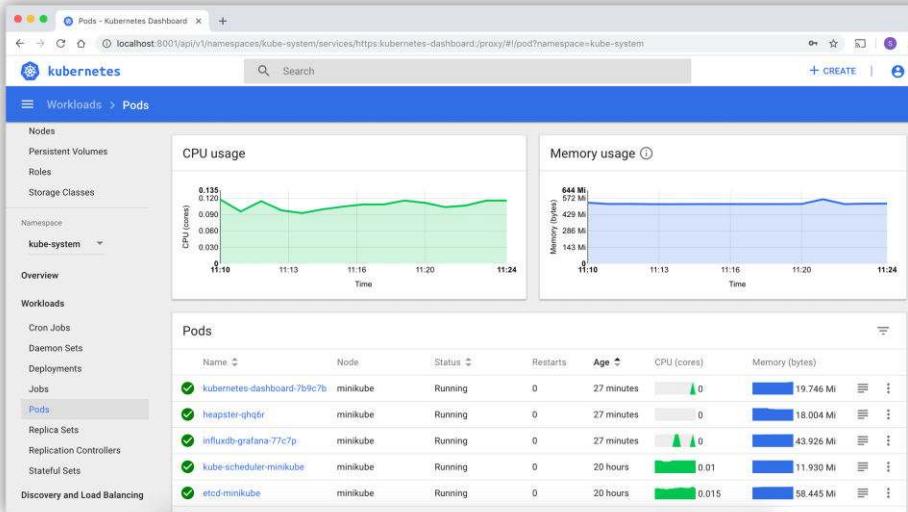
NAME	HOSTS	ADDRESS	PORTS	AGE
simple-fanout-example	*		80	2m5s



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 the cluster resources.



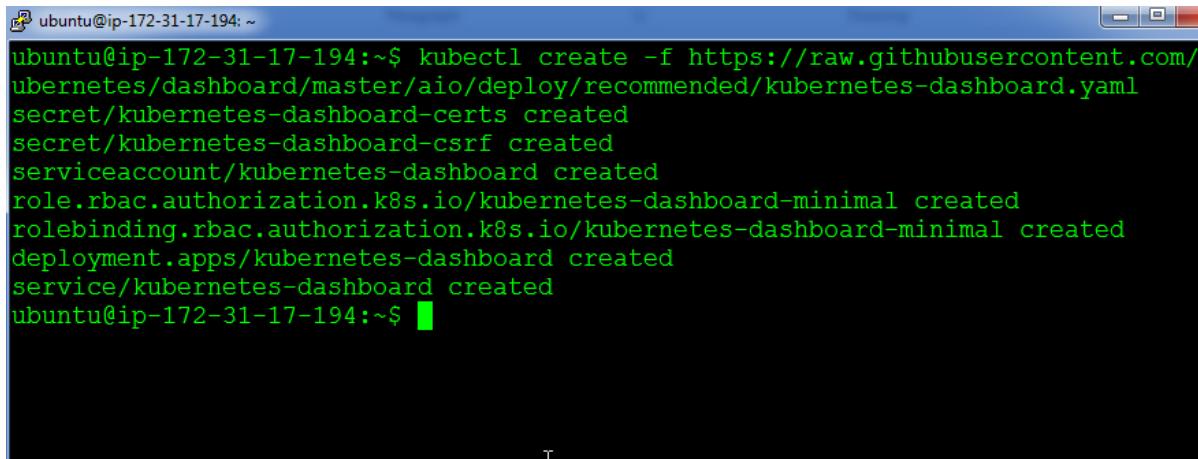
Installing Kubernetes Dashboard

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 terminal window titled "ubuntu@ip-172-31-17-194: ~" showing the output of a kubectl command. The command is to create resources from a YAML file at the provided URL. The output shows the creation of various Kubernetes objects: a secret for certificates, a service account, a role for RBAC, a role binding, a deployment for the dashboard, and a service. The terminal window has a standard Windows-style title bar and a black background for the code output.

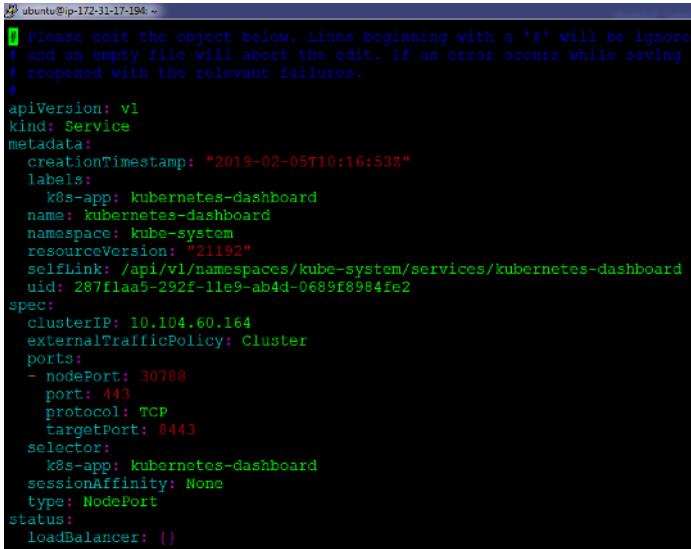
```
ubuntu@ip-172-31-17-194:~$ kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended/kubernetes-dashboard.yaml
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
serviceaccount/kubernetes-dashboard created
role.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
deployment.apps/kubernetes-dashboard created
service/kubernetes-dashboard created
ubuntu@ip-172-31-17-194:~$
```

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

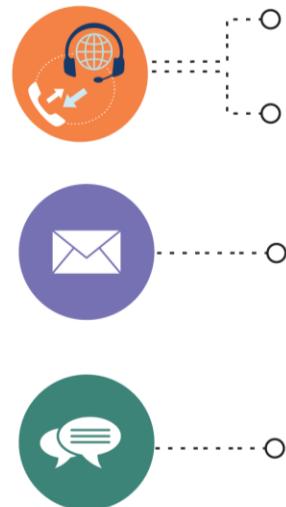
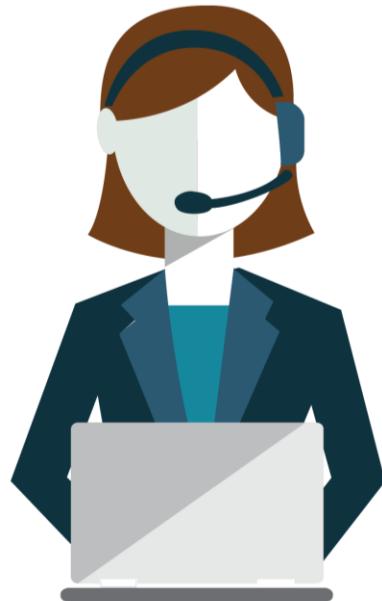
1. Check the NodePort from the kubernetes-dashboard service
2. Browse to your cluster on the internet browser, and enter the IP address
3. Click on Token, it will ask you for the token entry
4. Generate a token using the following command

```
$ 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



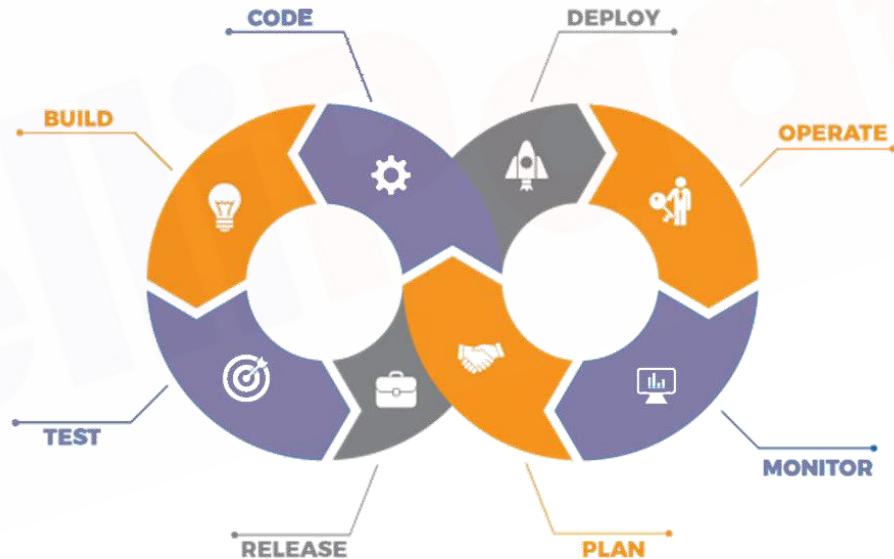
India : +91-7847955955

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

support@intellipaat.com

24X7 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.

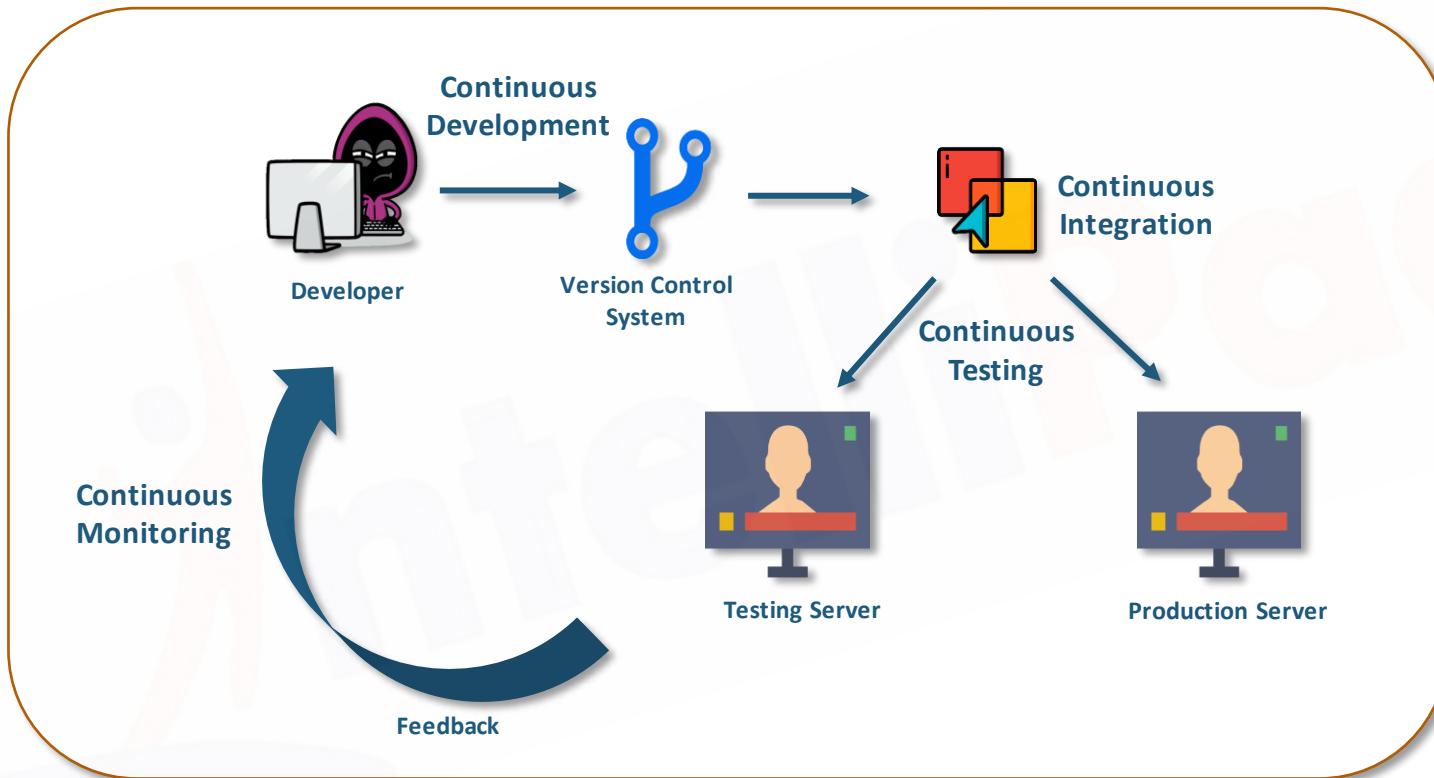
It monitors and troubleshoots server performance issues.

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.

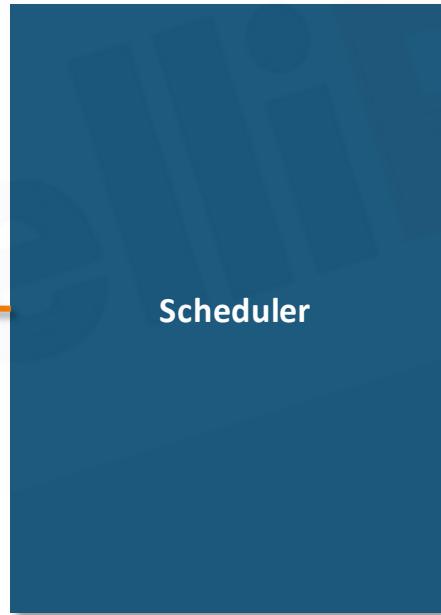
The word "Nagios" in a large, bold, black, sans-serif font. The letter "N" has a horizontal underline underneath it. A registered trademark symbol (®) is positioned at the top right of the "os".

Nagios Architecture

Nagios Architecture



Nagios GUI

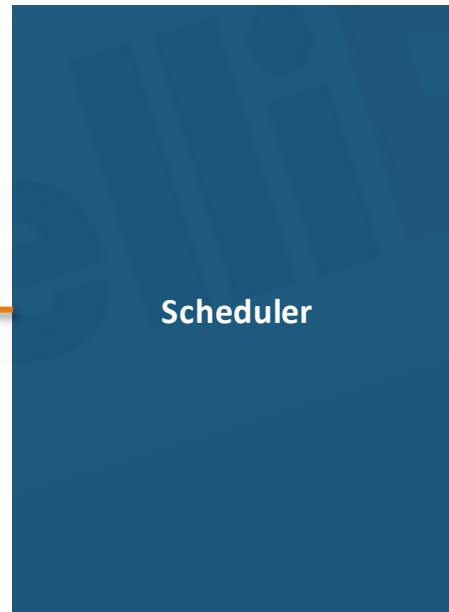


Nagios Architecture



Nagios GUI

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

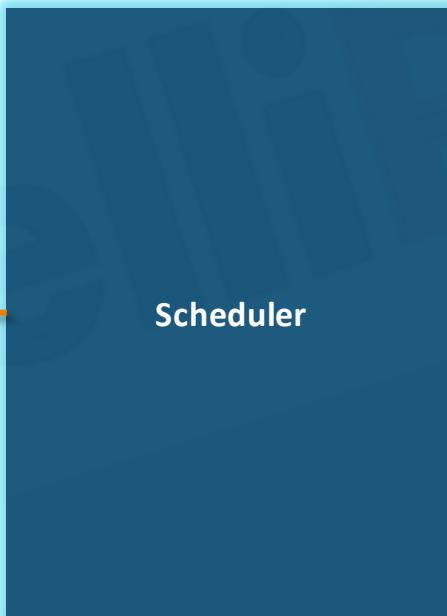


Nagios Architecture



Nagios GUI

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

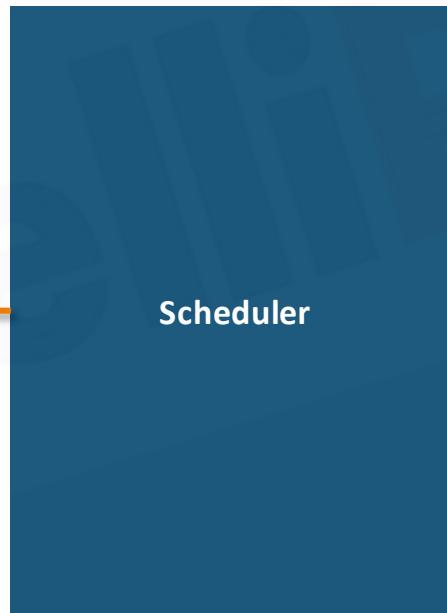


NRPE Plugins

Nagios Architecture



Nagios GUI



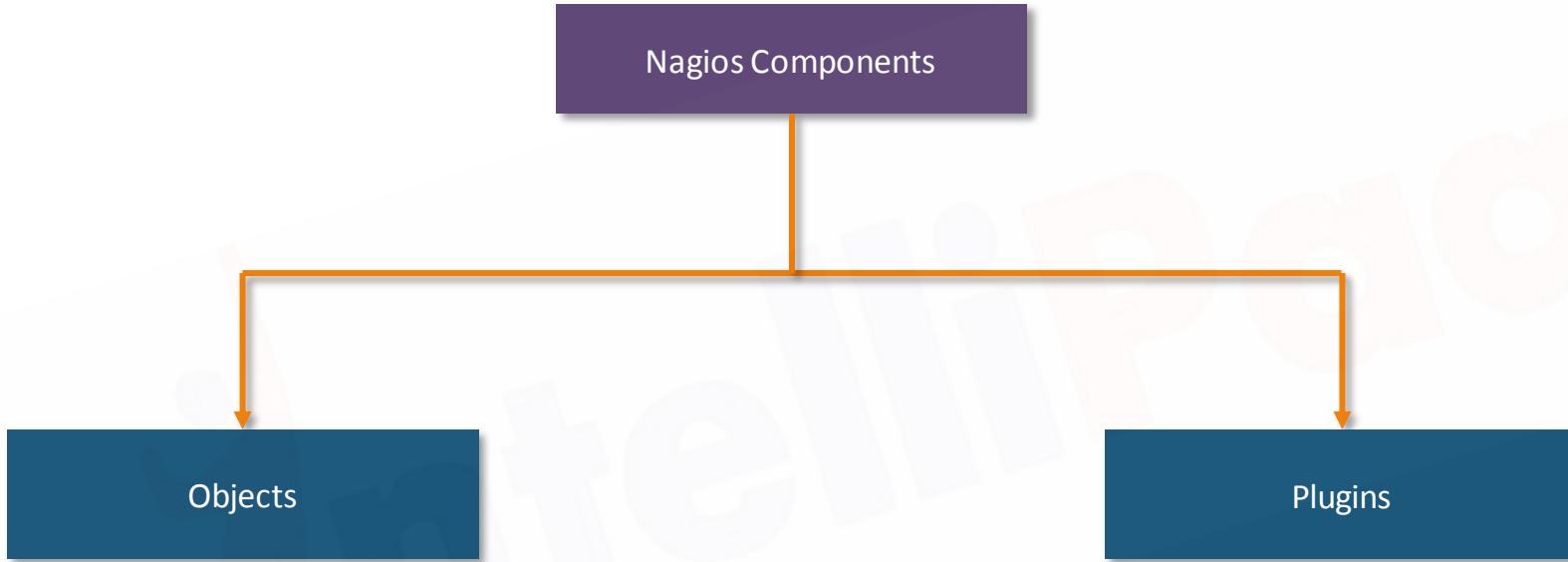
NRPE Plugins

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

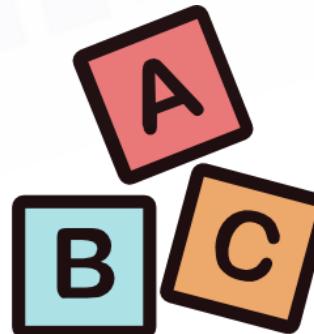


Nagios Components

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

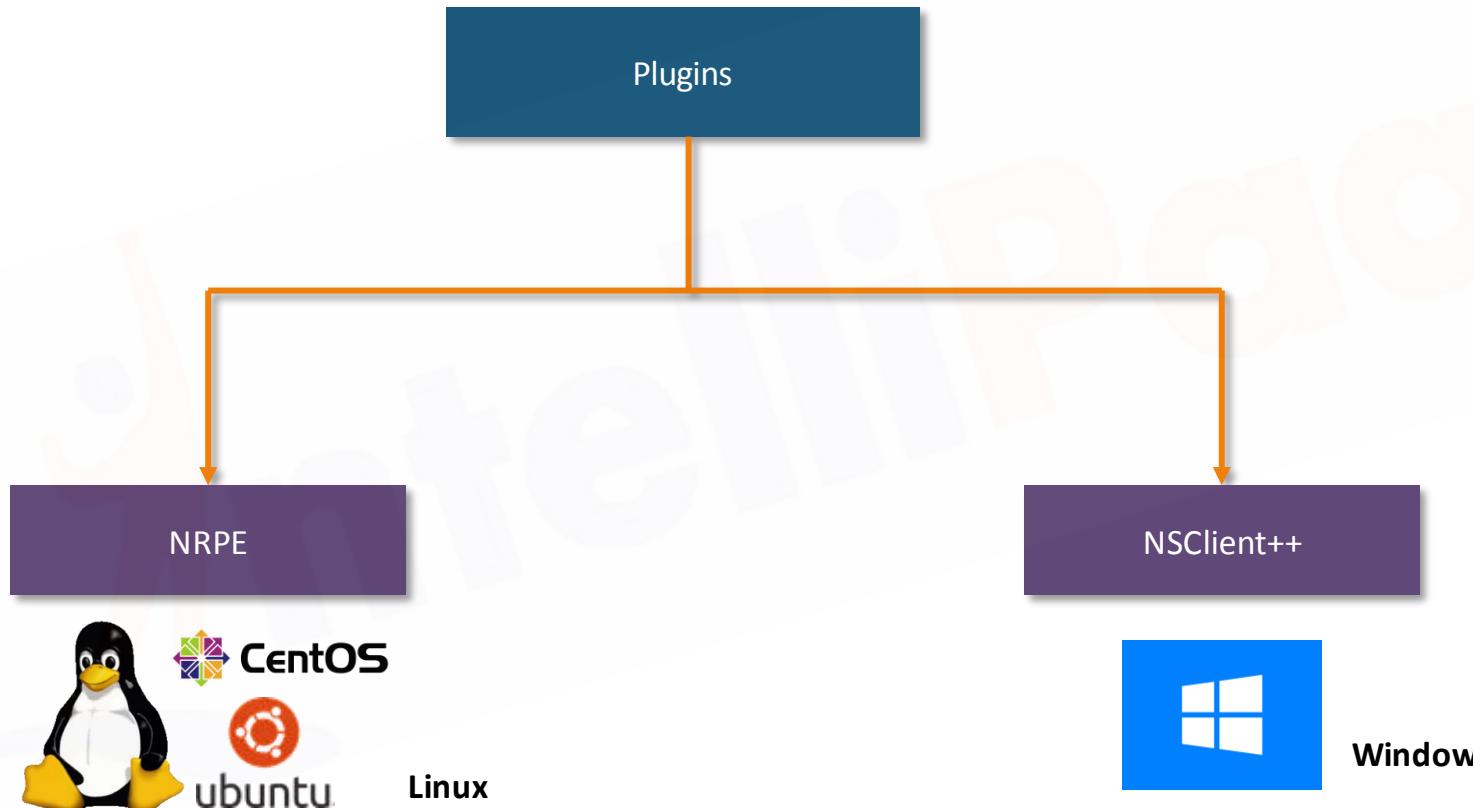
Plugins

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

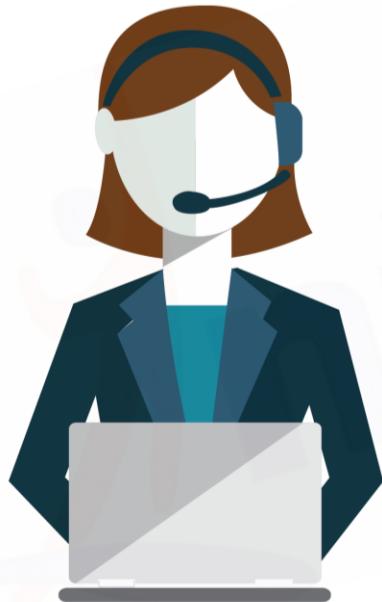
5. On which port does Nagios GUI work?

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



elastic stack

Agenda

01 What is ELK?

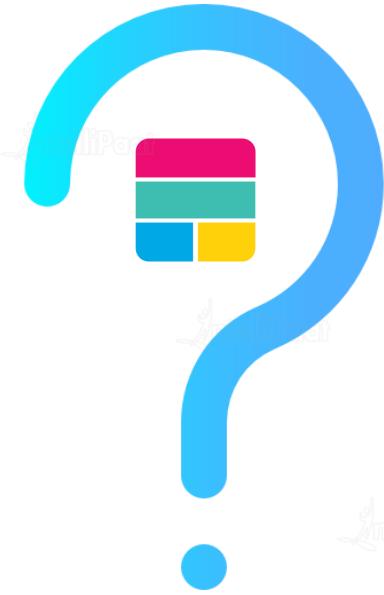
02 What are the components of ELK?

03 ELK Flow

04 Features of ELK

05 ELK Installation

06 ELK Hands-on



What is ELK?

What is ELK?



Elastic Stack (ELK) refers to a set of open-source products developed by Elastic to help its users collect data from different types of sources, analyze the collected data, and represent the analysis in an easy-to-understand and aesthetic visualization so that meaningful observations can be made



Sources of Data

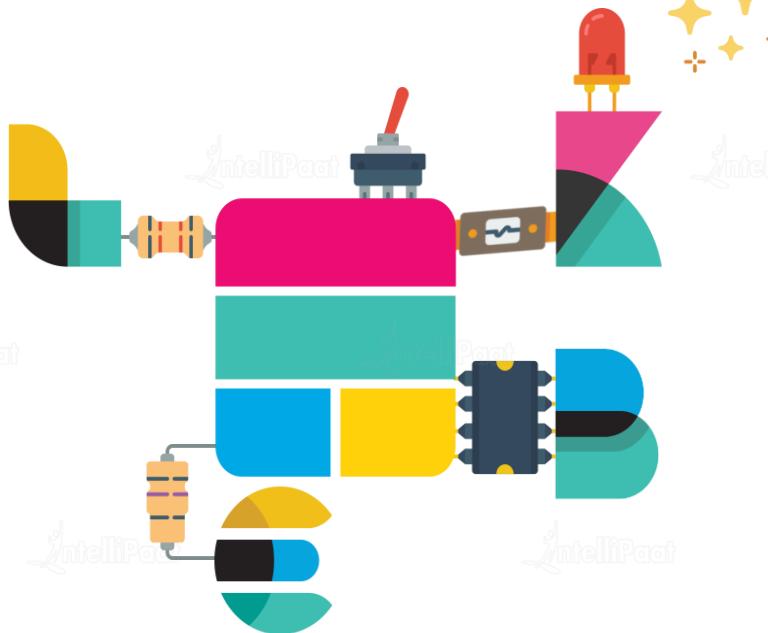


Elastic Stack



Visualizations

Learning how to use all components of Elastic Stack takes some time, but the payoff is great and grants a deeper understanding of the software's underlying structure



What are the components of ELK?

What are the components of ELK?

Elasticsearch



For storing and searching collected data

Logstash



For collecting and filtering the input data

Kibana



Provides a graphical user interface

Beats



Offers multiple light-weight data collectors

What are the components of ELK?

Elasticsearch

Logstash

Kibana

Beats

Elasticsearch is a NoSQL database that was developed based on Apache Lucene search engine. It can be used to index and store different types of documents and data. It provides a function to search for the data that is stored in real time as it's being fed.



What are the components of ELK?

Elasticsearch

Logstash

Kibana

Beats

Logstash is a collection agent used to collect both heterogenous/non-heterogenous data from various sources. It has the capability to screen, breakdown, and make string alterations in the data it collects. After collecting and filtering the data, it then sends it to Elasticsearch for storage



What are the components of ELK?

Elasticsearch

Logstash

Kibana

Beats

Kibana is a graphical user interface used to display the data that is collected and stored in Elasticsearch. It displays the data with appealing visuals so that the data could be easily understood and analyzed; it does so by using multiple types of visuals like bar chart, pie chart, world map, heat map, co-ordinate map, etc.



What are the components of ELK?

Elasticsearch

Logstash

Kibana

Beats

Features of Kibana

Discover your data by exploring it

Analyze your data by applying different metrics

Visualize the data by creating different types of charts

Apply Machine Learning on the data to get data anomaly

Manage users and roles

Offer a console to run Elasticsearch expressions

Play with time-series data using Timeline

Monitor your Elastic Stack using monitoring

What are the components of ELK?

Elasticsearch

Logstash

Kibana

Beats

Beats is similar to Logstash in the fact that they both collect data that will be later stored and analyzed, but Beats differs in the method of collection. Beats is a set of multiple small software installed on different servers from where they collect the data and send it to Elasticsearch





ELK Flow

ELK Flow



First, Beats are attached to remote servers from where these Beats collect information from various sources

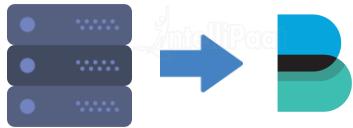
After collecting all the data needed, they either ship the data to Logstash for filtration or directly send it to Elasticsearch

The data is then stored in Elasticsearch. From here, it will not be directly sent to Kibana. Kibana first needs to find where Elastic is and then go and get the data by itself

ELK Flow



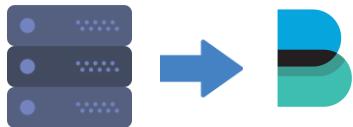
Data Collection



Server

Beat

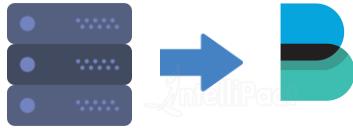
Data Collection



Server

Beat

Data Collection



Server

Beat



Data Aggregation
and Processing



Indexing and
Storing



Analysis and
Visualization



Notification

Slack

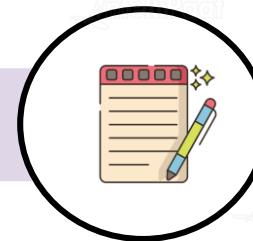


Features of ELK

Features of ELK



System Performance Monitoring



Log Management



Application Performance Monitoring

Features of ELK

Application Data Analysis

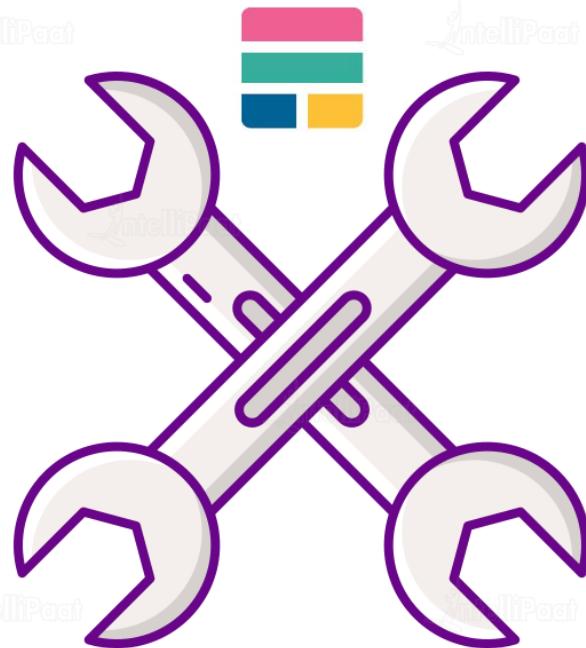


Security Monitoring and Alerting



Data Visualization



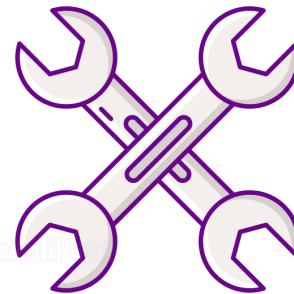


ELK Installation

ELK Installation

Prerequisites:

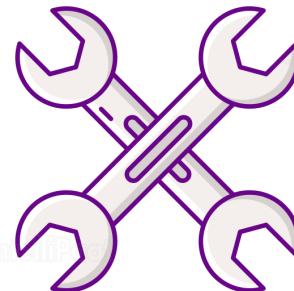
- At least 2 GB of RAM
- At least 20 GB storage
 - JAVA



ELK Installation

Installing JAVA on the instance:

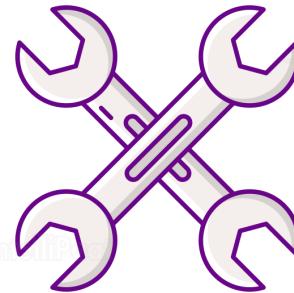
```
$ sudo apt-get update  
$ sudo apt-get install -y openjdk-8-jdk
```



ELK Installation

Installing nginx on the instance:

```
$ sudo apt-get update  
$ sudo apt-get -y install nginx  
$ sudo systemctl enable nginx
```

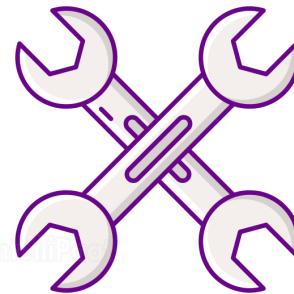


ELK Installation



Downloading and installing Elasticsearch:

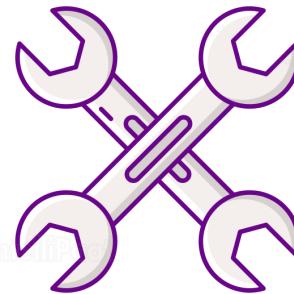
```
$ wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.2.0-amd64.deb  
$ sudo dpkg -i elasticsearch-7.2.0-amd64.deb
```



ELK Installation

Downloading and installing Kibana:

```
$ wget https://artifacts.elastic.co/downloads/kibana/kibana-7.2.0-amd64.deb  
$ sudo dpkg -i kibana-7.2.0-amd64.deb
```

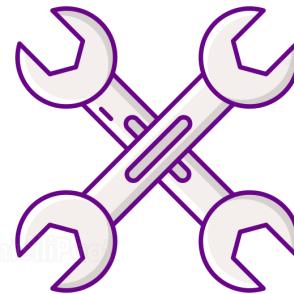


ELK Installation



Downloading and Installing Logstash:

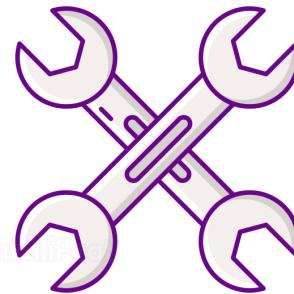
```
$ wget https://artifacts.elastic.co/downloads/logstash/logstash-7.2.0.deb  
$ sudo dpkg -i logstash-7.2.0.deb
```



ELK Installation

Installing a few dependencies:

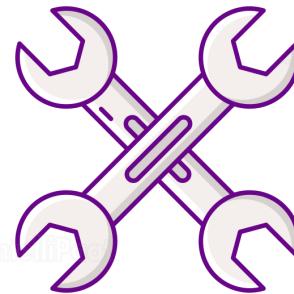
```
$ sudo apt-get install -y apt-transport-https
```



ELK Installation

Downloading and Installing Filebeat:

```
$ wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.2.0-amd64.deb  
$ sudo dpkg -i filebeat-7.2.0-amd64.deb
```





ELK Hands-on

ELK Hands-on



1.

Collect static Apache logs using Logstash and analyze them using Kibana

2.

Collect static '.CSV' using Logstash and analyze them using Kibana

3.

Collect and configure real-time web logs, inject them into Elasticsearch, and analyze them using Kibana



India: +91-7847955955



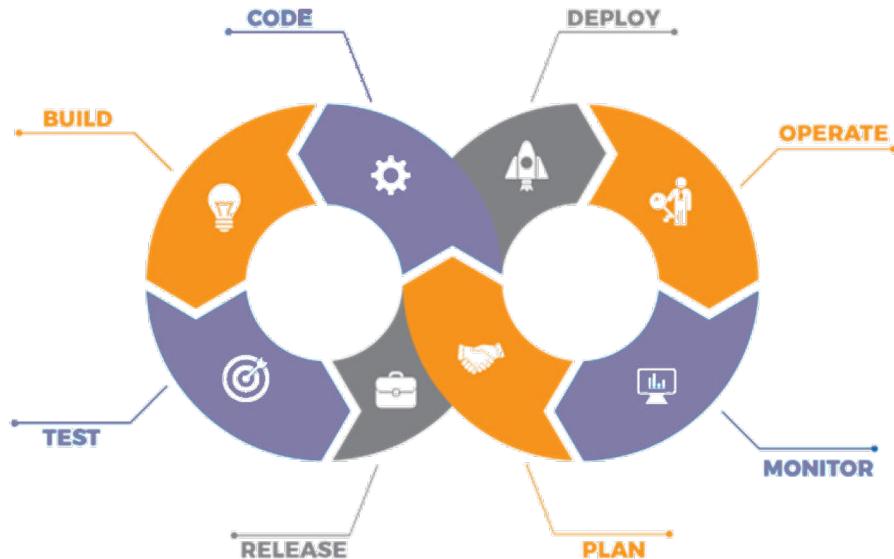
US: 1-800-216-8930 (TOLL FREE)



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.

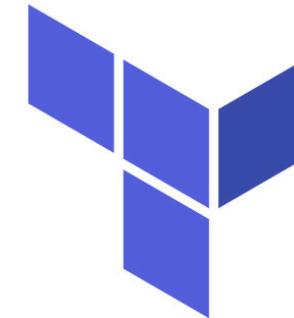


Installing Terraform for AWS

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

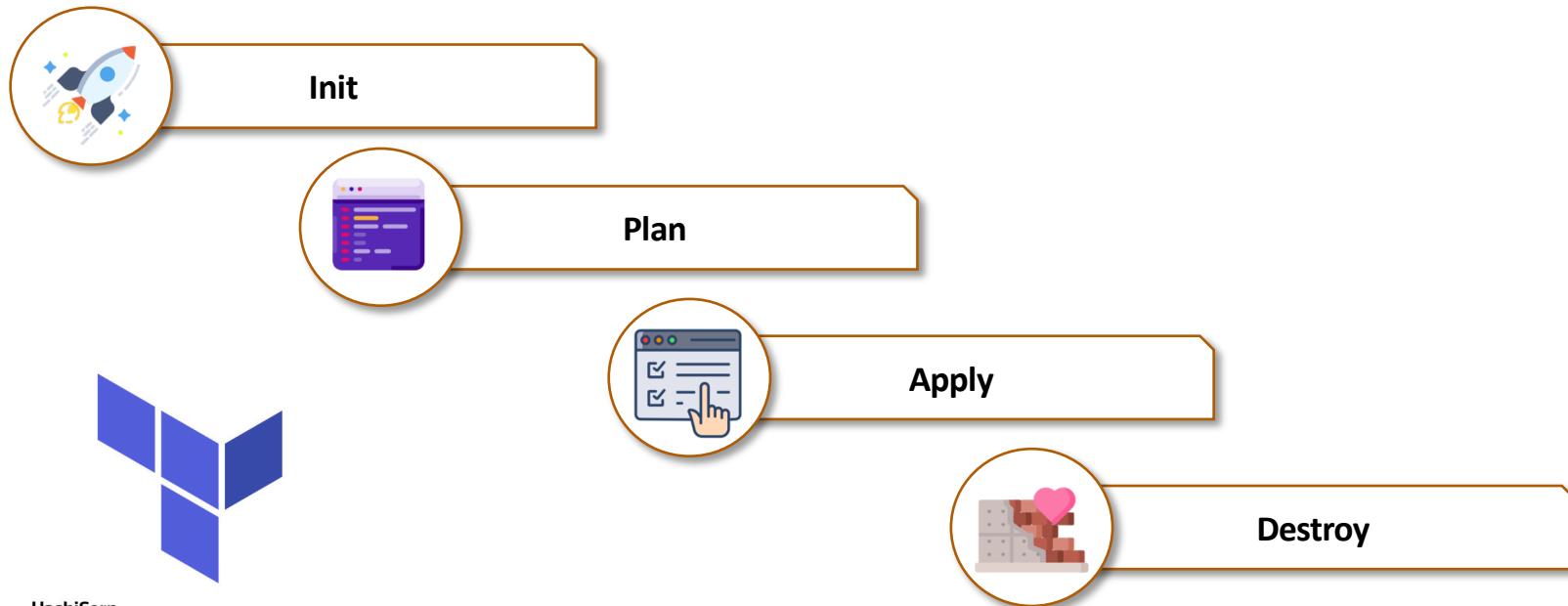


HashiCorp
Terraform

Basic Terraform Operations

Basic Terraform Operations

There are four kinds of operations in Terraform:



Terraform Code Basics

Terraform Code Basics

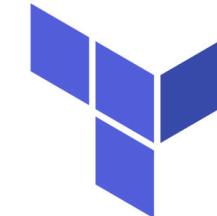
Following is a snippet of terraform file. Terraform files are saved with the extension of tf (*.tf). It has two main components, **block type** and **Key Value Pairs**

Block Type

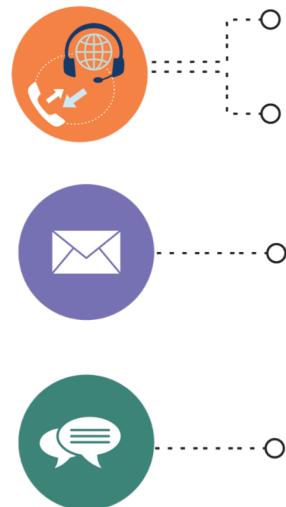
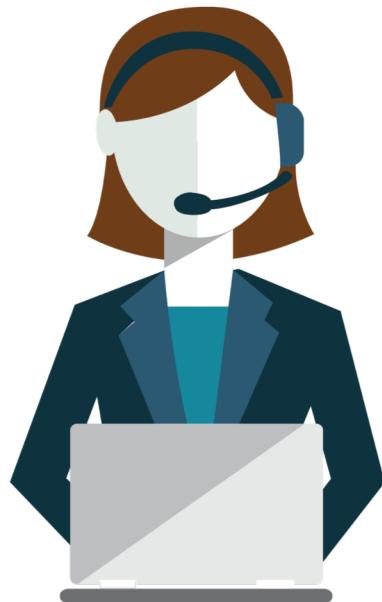
```
1 provider "aws" {  
2  
3     region = "ap-south-1"  
4     access_key = "AKIA2ESXR0HTZZNFXDKI"  
5     secret_key = "frQzLIIH0DoRISZ6wqtI0Z7Uoe8IuFIwxVXIjk1B"  
6  
7 }  
8
```



Key Value Pairs



Deploying end to end Infrastructure using Terraform



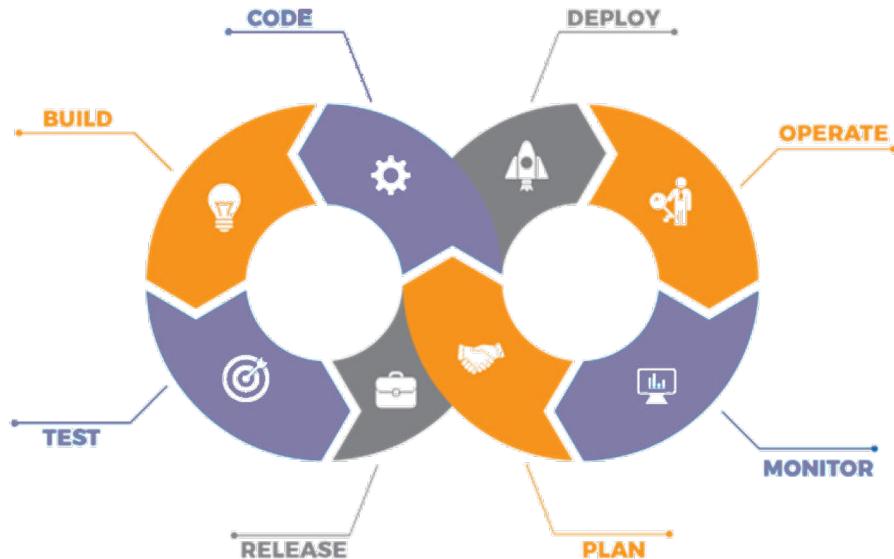
India: +91-7847955955

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

sales@intellipaat.com

24/7 Chat with Our Course Advisor

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

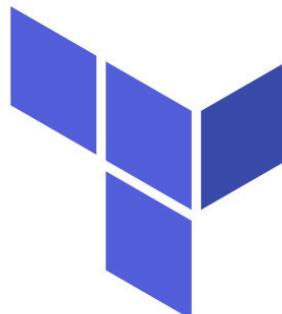
Configuration Management

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.



HashiCorp

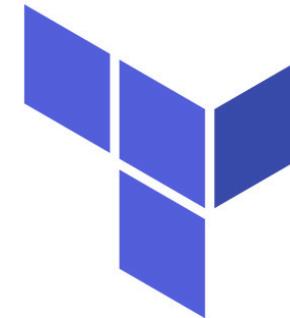
Terraform

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

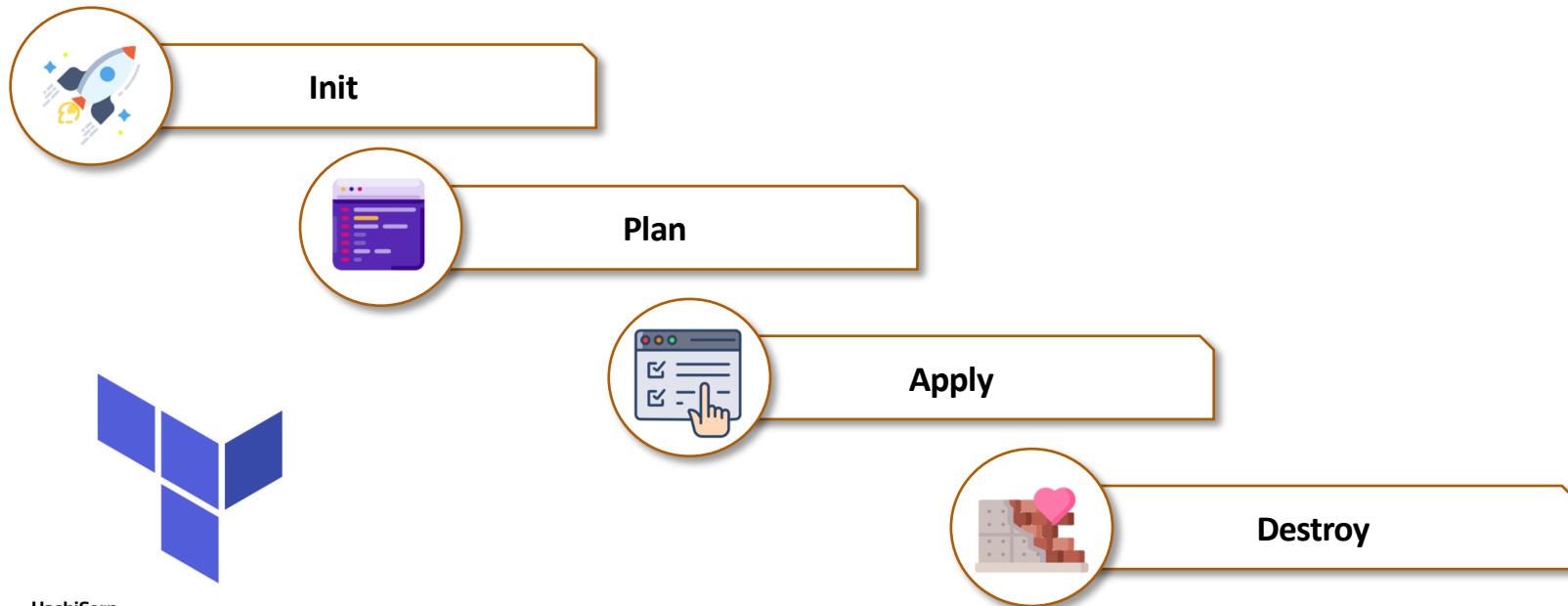


HashiCorp
Terraform

Basic Terraform Operations

Basic Terraform Operations

There are four kinds of operations in Terraform:



Terraform Code Basics

Terraform Code Basics

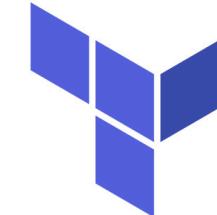
Following is a snippet of terraform file. Terraform files are saved with the extension of tf (*.tf). It has two main components, **block type** and **Key Value Pairs**

Block Type

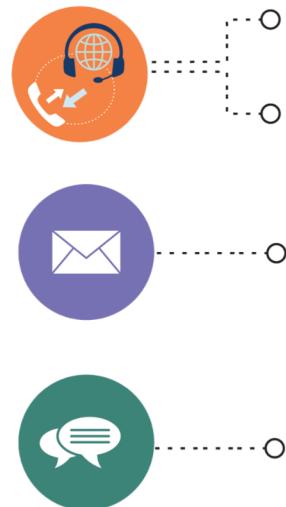
```
1 provider "aws" {  
2  
3     region = "ap-south-1"  
4     access_key = "AKIA2ESXR0HTZZNFXDKI"  
5     secret_key = "frQzLIIH0DoRISZ6wqtI0Z7Uoe8IuFIwxVXIjk1B"  
6  
7 }  
8
```



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

Module-6: Ansible Assignment - 4

You have been asked to:

- Use the previous deployment of ansible cluster
- Configure the files folder in “ansible-new” role with index.html which should be replaced with the original index.html
- Configure the handlers folder in “ansible-new” role for restarting the nginx service

All of the above should only happen on the slave which has nginx installed



Module-6: Ansible Assignment - 1

You have been asked to:

- Setup Ansible cluster with 3 nodes
- On slave1 install java
- On slave 2 install mysql-server

Do the above tasks using Ansible playbooks



Module-6: Ansible Assignment - 2

You have been asked to:

- Create a script which can add text “This text has been added by custom script” to /tmp.1.txt
 - Run this script using Ansible on all the hosts
-



Module-6: Ansible Assignment - 3

You have been asked to:

- Create Ansible Role called “ansible-new”
- Install apache2 on slave1 and nginx on slave2

Above should be implemented using Ansible Roles



Module-6: Ansible Assignment - 5

You have been asked to:

- Create a new deployment of ansible cluster of 5 nodes
- Label 2 nodes as test and other 2 as prod
- Install java on test nodes
- Install mysql-server on prod nodes

Use Ansible roles for the above, group the hosts under test and prod



ANSIBLE CASE STUDY

You are a Devops Engineer and the organization you are working on needs to set up two configuration management server groups. One for Apache another for Nginx. Being a Devops Engineer it is your task to deal with this configuration management issue.

Let us see the tasks that you need to perform using Ansible.

1. Create two Server Groups. One for Apache and another for Nginx.
2. Push two html files with their server information.

Make sure that you don't forget to start the services once the installation is done. Also send post installation messages for both the server groups.

Using Ansible Roles accomplish the above the tasks.

Also, once the Apache server configuration is done you need to install Java on that server group using ansible role in a playbook.



Module-3: Docker – I Assignment - 1

You have been asked to:

- Pull ubuntu container
 - Run this container, and map port 80 on the local
 - Install apache2 on this container
 - Check if you are able to access the apache page on your browser
-



Module-3: Docker – I Assignment - 2

You have been asked to:

- Save the image created in Assignment 1 as a Docker image
 - Launch container from this new image and map the port to 81
 - Go inside the container and start the apache2 service
 - Check if you are able to access it on the browser
-



Module-3: Docker – I Assignment - 3

You have been asked to:

- Use the saved image in the previous assignment
 - Upload this image on Dockerhub
 - On a separate machine pull this dockerhub image, and launch it on port 80
 - Start the apache2 service
 - Verify if you are able to see the apache2 service
-



Module-3: Docker - I Assignment - 4

You have been asked to:

Create a dockerfile with the following specs:

- Ubuntu container
- Apache2 installed
- Apache2 should automatically run once the container starts

Submit the dockerfile, for assignment completion



Module-3: Docker – I Assignment – 5

You have been asked to:

- Create a sample HTML file
 - Use the Dockerfile from the previous task
 - Replace this sample HTML file inside the docker container with the default page
-



Module-4: Docker - II: Assignment - 1

You have been asked to:

- Launch the Apache2 container created in previous module
- Create a Docker volume on /var/www/html

Please submit the commands, in order to complete this assignment



Module-4: Docker - II: Assignment - 2

You have been asked to:

- Use the apache2 container created in previous module
- Create a bind mount on /var/www/html to replace html files dynamically

Submit the commands, to complete the assignment



Module-4: Docker - II: Assignment - 3

You have been asked to:

- Create 5 custom container, with 5 different default pages
 - Using docker compose, deploy these 5 containers on port 81, 82, 83, 84 and 85 respectively
-



Module-4: Docker - II: Assignment - 4

You have been asked to:

- Create a Docker swarm cluster with 3 nodes
 - Deploy an apache container with 4 replicas
-



Module-4: Docker - II: Assignment - 5

You have been asked to:

- Use the previous assignment's deployment
 - Deploy any 2 containers in a overlay network
 - Try pinging each of the containers from within the containers
-



CASE STUDY - CONTAINERIZATION USING DOCKER I

Problem Statement:

You work as a Devops Engineer in a leading Software Company. You have been asked to Dockerize the applications on the production server. The company uses custom software, therefore there is no pre-built container which can be used.

Assume the following things:

1. Assume the software to be installed is apache
2. Use an Ubuntu container

The company wants the following things:

1. Push a container to DockerHub with the above config
 2. The Developers will not be working with Docker, hence from their side you will just get the code. Write a Dockerfile which could put the code in the custom image that you have built
-



CASE STUDY - CONTAINERIZATION USING DOCKER II

Problem Statements:

You have been hired as a Devops Engineer in GrapeVine Pvt. Ltd. You have been asked to improve the way the company is managing their Docker containers.

Following tasks have been assigned:

1. Deploy a sample HTML website in any apache container, and demonstrate how we can dynamically change content in the container by making changes on the host machine (Bind Mounts)

 2. Deploy apache and nginx containers using Docker Compose, Apache should be exposed on Port 91 and nginx on port 92

 3. Initialize a Docker Swarm Cluster, and deploy two ubuntu containers in a overlay network. Demonstrate they can communicate with each other, by pinging them
-



Module-2: GIT Assignment - 1

Based on what you have learnt in the class, do the following steps:

- Create a new folder
- Put the following files in the folder
 - Code.txt
 - Log.txt
 - Output.txt
- Stage the Code.txt and Output.txt files
- Commit them
- And finally push them to github

Please share the commands for the above points



Module-2: GIT Assignment - 2

Do the following tasks:

- Create a git working directory with feature1.txt and feature2.txt in the master branch
- Create 3 branches develop, feature1 and feature2
- In develop branch create develop.txt, do not stage or commit it
- Stash this file, and checkout to feature1 branch
- Create new.txt file in feature1 branch, stage and commit this file
- Checkout to develop, unstash this file and commit

Please submit all the git commands used to do the above steps



Module-2: GIT Assignment - 3

You have been asked to:

- Create a git working directory, with the following branches
 - Develop
 - F1
 - f2
 - In the master branch, commit main.txt file
 - Put develop.txt in develop branch, f1.txt and f2.txt in f1 and f2 respectively
 - Push all these branches to github
 - On local delete f2 branch
 - Delete the same branch on github as well
-



Module-2: GIT Assignment - 4

You have been asked to:

- Put master.txt on master branch, stage and commit
 - Create 3 branches: public1, public2 and private
 - Put public1.txt on public 1 branch, stage and commit
 - Merge public 1 on master branch
 - Merge public 2 on master branch
 - Edit master.txt on private branch, stage and commit
 - Now update branch public 1 and public 2 with new master code in private
 - Also update new master code on master
 - Finally update all the code on the private branch
-



Module-2: GIT Assignment - 5

You have been asked to:

- Create a gitflow workflow architecture on git
 - Create all the required branches
 - Starting from the feature branch, push the branch to the master, following the architecture
 - Push a urgent.txt on master using hotfix
-



CASE STUDY - GIT WORKFLOW

Problem Statement:

You work as a Devops Architect in Zendriix Softwares. The company has been struggling to manage their product releases. The releases should happen on 25th of every month. Suggest a Git Workflow Architecture for this requirement.

Simulate this workflow, by creating a pseudo code files and branches, and upload the same to your GitHub Account.

As a part of solution, share the link to your GitHub repository.



Module-8: Jenkins Assignment - 1

You have been asked to:

- Trigger a pipeline using Git when push on Develop branch
 - Pipeline should pull git content to a folder
-



Module-8: Jenkins Assignment - 2

You have been asked to:

- Add 2 nodes to Jenkins master
 - Create 2 jobs with the following jobs:
 - Push to test
 - Push to prod
 - Once a push is made to test branch copy git files to test server
 - Once a push is made to master branch copy git files to prod server
-



Module-8: Jenkins Assignment - 3

You have been asked to:

- Create a pipeline in jenkins
 - Once push is made to “develop” branch in git, trigger job “test”. This will copy git files to test node
 - If test job is successful, then prod job should be triggered
 - Prod jobs should copy files to prod node
-



Integration of Devops tools with Jenkins

You have been Hired as a Devops Engineer in xyz software company. They want to implement CI/CD pipeline in their company. You have been asked to implement this lifecycle as fast as possible. As this is a product-based company, their product is available on this GitHub link.

<https://github.com/hshar/website.git>

Following are the specifications of the Continuous integration:

1. Git Workflow has to be implemented
 2. Code Build should automatically be triggered once commit is made to master branch or develop branch.
If commit is made to master branch, build and publish website on port 82. If commit is made to develop branch, just build the product, do not publish.
 3. Create a pipeline for the above tasks.
 4. Create a container with Ubuntu and apache installed in it and use that container to build the code and the code should be on '/var/www/html'.
-

Module-9: Kubernetes Assignment - 1

You have been asked to:

- Deploy a Kubernetes Cluster for 3 nodes
 - Create a nginx deployment of 3 replicas
-



Module-9: Kubernetes Assignment - 2

You have been asked to:

- Use the previous deployment
 - Create a service of type NodePort for nginx deployment
 - Check the nodeport service on a browser to verify
-



Module-9: Kubernetes Assignment - 3

You have been asked to:

- Use the previous deployment
 - Change the replicas to 5 for the deployment
-



Module-9: Kubernetes Assignment - 4

You have been asked to:

- Use the previous deployment
 - Change the service type to clusterip
-



Module-9: Kubernetes Assignment - 5

You have been asked to:

- Use the previous deployment
 - Deploy an nginx deployment of 3 replicas
 - Create an nginx service of type clusterip
 - Create an ingress service /apache to apache service /nginx to nginx service
-



CASE STUDY - INTRODUCTION TO KUBERNETES

You have just joined a startup Ventura Software as a Devops Lead Engineer. The company relies on a Monolithic Architecture for its product. Recently, the senior management was hired. The new CTO insists on having a Microservice Architecture. The Development Team, is working on breaking the Monolith. Meanwhile, you have been asked to host a Test Application on Kubernetes, to understand how it works.

Following things have to be implemented:

1. Deploy an Apache2 deployment of 2 replicas
2. Sample code has been checked-in at the following Git-Hub repo:

<https://github.com/hshar/website.git>.

You have to containerize this code, and push it to Docker Hub. Once done, deploy it on Kubernetes with 2 replicas

3. Deploy Ingress with the following rules:
 - i) */apache* should point to the apache pods
 - ii) */custom* should point to the GitHub application
-

Kubernetes 1 - Assignment

Problem Statement:

You work for the xyz organization. Your organization uses Kubernetes for container orchestration. Your organization has recently created pods from which data was being lost. Now they require volume mounts which preserve data and save a password called “xyzIsthebest” and this has to be put on a particular node of your choice.

You have been asked to:

1. Create a persistent volume
2. create a persistent volume claim
3. create a secret “xyzIsthebest”
4. Taint one of the nodes of the cluster

Module-10: Nagios Assignment - 1

You have been asked to:

- Create a Nagios Master slave architecture and add 2 hosts
-



Module-10: Nagios Assignment - 2

You have been asked to:

- Create a monitoring service which will check whether apache2 service is up or not
-



Module-10: Nagios Assignment - 3

You have been asked to:

- Use the previous deployment cluster
 - Check whether the FTP service is up or not (Use the configuration files for this)
-



Module-10: Nagios Assignment - 4

You have been asked to:

- On both the slaves run a CPU checking service
 - Reduce the check interval on slave2 to 1 minute since it's a critical server
-



CASE STUDY - CONTINUOUS MONITORING USING NAGIOS

You've just started working as a Devops Engineer in a startup. They do not have any monitoring system in place for their applications. Help your company in monitoring their applications.

Following are the expectations:

1. The company has a Production system, where they have deployed a website. This website is critical for the business hence requires monitoring at all times. Create a Monitoring Service for the same.
2. The company also want to monitor the CPU usage of the Hosts
3. Since the application is critical, we need to reduce monitor_check_intervals to be 1 minute.

Assume the following things:

1. On the production server/Nagios Host deploy the hshar/webapp container. This will run a website on the server. This website has to be monitored.
 2. Deploy 2 servers on AWS, 1st Nagios Server, and the other would be the company's production server.
-

Module-5: Puppet Assignment - 1

You have been asked to:

- Setup puppet master-slave using 3 nodes
 - Installing apache on slaves using manifests
-



Module-5: Puppet Assignment - 2

You have been asked to:

- Add one more slave to the setup
- Add a condition, if apache is already installed on the slaves, install nginx

The result should be 3rd slave with nginx installed



Module-5: Puppet Assignment - 3

You have been asked to:

- Use the previous module's deployment
 - If apache is installed change index.html to "apache is installed"
 - If nginx is installed change index.html to "nginx is installed"
-



Module-5: Puppet Assignment - 4

You have been asked to:

- Use the previous module deployment
 - Use modules for this task
 - Ensure mysql and java is installed on nginx machine
-



Module-5: Puppet Assignment - 5

You have been asked to:

- Use the previous assignment's deployment
 - Define hostnames for slaves, as slave1, slave2 and slave3
 - On slave1 create a file in /temp, which says this is slave1
 - On slave2 create a file in /temp, which says this is slave2
 - On slave3 create a file in /temp, which says this is slave3
-



CASE STUDY - CONFIGURATION MANAGEMENT USING PUPPET

You work for Tech Giants Software. Recently the company got funding and they have decided to expand their server fleet. You have been designated the task for Configuration Management.

Assume the following infrastructure:

1. 1 Nginx Server
2. 1 Apache Server
3. 1 Puppet Master

On Apache Server, we have to install docker. And on the nginx server, we have to install Java. It sounds straightforward, but we do not know, which systems have what software installed on it.

Using conditional statements, accomplish the above task.

Also, for improving code readability, please make use of modules and classes.



Terraform Assignment - 1

You have been asked to:

- Create an EC2 service in the default subnet in the ohio region
-



Terraform Assignment - 2

You have been asked to:

- Destroy the previous deployment
 - Create a new EC2 instance with an Elastic IP
-



Terraform Assignment - 3

You have been asked to:

- Destroy the previous deployment
 - Create 2 EC2 instances in Ohio and N.Virginia respectively
 - Rename Ohio's instance to 'hello-ohio' and Virginia's instance to 'hello-virginia'
-



Terraform Assignment - 4

You have been asked to:

- Destroy the previous deployments
 - Create a VPC with the required components using Terraform
 - Deploy an EC2 instance inside the VPC
-



Terraform Assignment - 5

You have been asked to:

- Destroy the previous deployments
 - Create a script to install apache2
 - Run this script on a newly created EC2 instance
 - Print the IP address of the instance in a file on the local, once deployed
-



CASE STUDY -CREATING AN ARCHITECTURE USING TERRAFORM ON AWS

You work as a DevOps Engineer in leading Software Company. You have been asked to build an infrastructure safely and efficiently.

The company Requirements:

1. Use AWS cloud Provider and the software to be installed is Apache2
2. Use Ubuntu AMI

The company wants the Architecture to have the following services:

1. Create a template with a VPC, 2 subnets and 1 instance in each subnet
 2. Attach Security groups, internet gateway and network interface to the instance
-



CASE STUDY- RESOLVING MERGE CONFLICTS

Problem Statement:

You work for Zendrix Software & Co. You have been assigned the task of updating the Master branch of their Git repository with all the features from the feature branches.

Following is the GitHub account, <https://github.com/devops-intellipaat/merge-conflict.git>

Consider,

- Feature1 branch to be a public branch
- Feature2 branch to be a private branch

The company relies on a monolithic architecture, and for now all the code resides in one file "main.c".

The respective features have been added in the feature branches for main.c. Meanwhile, a security patch was made to the master branch, and now feature1 and feature2 branches are behind from master by 1 commit.

Following tasks have to be done:

1. Update Feature1 and Feature2 branch with the Security Patch
2. Apply changes of Feature1 and Feature2 branch on master
3. Finally push all the branches to GitHub

For Solving this, please fork the repository to your Github account and then work.

As a solution, please submit your GitHub's repository link.

CAPSTONE PROJECT - II

You are hired as a DevOps engineer for XYZ.pvt.co, The company is a product based company they are using Docker containers for their containerization inside the company, but in the meantime, the product got a lot of traffic, now they need to have a platform for automating deployment, scaling, and operations of application containers across clusters of hosts, As a DevOps engineer, you need to work on this and implement a DevOps life cycle, such that the Docker containers in the testing environment will not change.

As the company is a monolithic architecture with 2 developers and the product is present on <https://github.com/hshar/website.git>

Following are the specifications of life-cycle:

- Git workflow should be implemented, as the company is a monolithic architecture you need to take care of versions. The release should happen only on 25 of every month.
- Code build should be triggered once the commits are made in the master or hotfix branch.
- The code should be containerized with the help of the Docker file, The Dockerfile should be built every time if there is a push to git-hub. Use the container with Ubuntu and apache installed in it. After the build, this container should be pushed to the Docker hub.
- The above tasks should be done in a Jenkins pipeline with the following jobs.
 - Build website
 - Test website
 - Push to Docker hub
 - Push to production
- As per the requirement In the production server, you need to use the Kubernetes cluster and the containerized code from Docker hub should be deployed with 2 replicas. Use kubernetes dashboard for health checks of those containers using dashboard.
- Once the application is built on the production server you need to design a test case, Which will basically check whether the configurations are displaying on the website or not. The test should pass if the configurations are displayed for the product on both the production and testing server.
- For configuration management of the product, you need to deploy the configuration file in '/home/ubuntu' for the execution of the configuration in both testing and production server.
- The above task should be accomplished with the help of Ansible roles.
- Create a monitoring service for the website on the production server.
- Before that, as a Devops engineer test this monitoring service by stopping the apache service and check whether the email is sent to you or not.

Architectural advice:

Server1 jenkins master,Nagios master

Server2 jenkins slave,Testing server, nrpe plugin, PHP

Server3 jenkins slave, production server,nrpe plugin, kubernetes master,

PHP Server4 kubernetes node,nrpe plugin, host machine.





Capstone Project

DevOps Certification Training

support@intellipaat.com
+91-7022374614
US: 1-800-216-8930(Toll Free)

CAPSTONE PROJECT

You have been Hired Sr. Devops Engineer in Abode Software. They want to implement Devops Lifecycle in their company. You have been asked to implement this lifecycle as fast as possible. Abode Softwares is a product-based company, their product is available on this GitHub link.

<https://github.com/hshar/website.git>

Following are the specifications of the lifecycle:

1. Git Workflow has to be implemented
2. Code Build should automatically be triggered once commit is made to master branch or develop branch.

If commit is made to master branch, test and push to prod

If commit is made to develop branch, just test the product, do not push to prod

3. The Code should be containerized with the help of a Dockerfile. The Dockerfile should be built every time there is a push to Git-Hub. Use the following pre-built container for your application:

hshar/webapp

The code should reside in '/var/www/html'

4. The above tasks should be defined in a Jenkins Pipeline, with the following Jobs

Job 1 - Building Website

Job 2 - Testing Website

Job 3 - Push to Production

5. Since you are setting up the server for the first time, ensure the following file exists on both Test and Prod server in /home/ubuntu/config-management/status.txt. This file will be used by a third-party tool. This should basically have the info whether apache is installed on the system or not

The content of this file, should be based on whether git is installed or not.

If apache is installed => Apache is Installed on this System"

If apache is not installed => "Apache is not installed on this System"

Architectural Advice:

Create 3 servers on AWS "t2.micro"

Server 1 - should have Jenkins Master, Puppet Master and Nagios Installed

Server 2 - Testing Server, Jenkins Slave

Server 3 - Prod Server, Jenkins Slave