# **Practical - 1**

**Version Control System**

**Q1. What is a Version Control System? Define Git. How Git is used in the Version Control System**.

A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.

Developers can compare earlier versions of the code with an older version to fix the mistakes.

Git is an open-source distributed version control system. It is designed to handle minor to major projects with high speed and efficiency. It is developed to coordinate the work among the developers.

Git is the foundation of many services like GitHub and GitLab, but we can use Git without using any other Git services. Git can be used privately and publicly.

Git was created by Linus Torvalds in 2005 to develop the Linux Kernel. It is also used as an important distributed version-control tool for DevOps.

**Q2. Explain and implement all the staging and commit commands in Git.**

1) Git config command

This command configures the user. The Git config command is the first and necessary command used on the Git command line. This command sets the author name and email address to be used with your commits. Git config is also used in other scenarios.

Syntax

$ git config --global user.name "ImDwivedi1"

$ git config --global user.email "Himanshudubey481@gmail.com"

2) Git Init command

This command is used to create a local repository.

Syntax

$ git init Demo

The init command will initialize an empty repository. See the below screenshot.

3) Git clone command

This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL

Syntax

$ git clone URL

4) Git add command

This command is used to add one or more files to staging (Index) area.

Syntax

To add one file

$ git add Filename

To add more than one file

$ git add\*

5) Git commit command

This is used in two scenarios. They are as follows.

**Git commit -m**

This command changes the head. It records or snapshots the file permanently in the version history with a message.

Syntax

$ git commit -m " Commit Message"

**Git commit -a**

This command commits any files added in the repository with git add and also commits any files you've changed since then.

Syntax

$ git commit -a

6) Git status command

The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git. It does not show you any information about the committed project history. For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.

Syntax

$ git status

7) Git push Command

It is used to upload local repository content to a remote repository. Pushing is an act of transferring commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.

Git push command can be used as follows.

**Git push origin master**

This command sends the changes made on the master branch, to your remote repository.

Syntax

$ git push [variable name] master

**Git push -all**

This command pushes all the branches to the server repository.

Syntax

$ git push --all

8) Git pull command

This is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.

Syntax

$ git pull URL

9) Git Branch Command

This command lists all the branches available in the repository.

Syntax

$ git branch

10) Git Merge Command

This command is used to merge the specified branch history into the current branch.

Syntax

$ git merge BranchName

11) Git log Command

This command is used to check the commit history.

Syntax

$ git log

By default, if no argument passed, Git log shows the most recent commits first. We can limit the number of log entries displayed by passing a number as an option, such as -3 to show only the last three entries.

$ git log -3

12) Git remote Command

This is used to connect your local repository to the remote server. This command allows you to create, view, and delete connections to other repositories. These connections are more like bookmarks rather than direct links to other repositories. This command doesn't provide real-time access to repositories.

# **Practical - 2**

**Continuous Integration using Jenkins**

**Q1. Integrate GitHub with Jenkins by fetching the source code from GitHub and build it using Jenkins.**

Jenkins is a CI (Continuous Integration) server and this means that it needs to check out source code from a source code repository and build code. Jenkins has outstanding support for various source code management systems like Subversion, CVS etc.

Github is the fast becoming one of the most popular source code management systems. It is a web based repository of code which plays a major role in DevOps. GitHub provides a common platform for many developers working on the same code or project to upload and retrieve updated code, thereby facilitating continuous integration. Jenkins works with Git through the Git plugin.

Connecting a GitHub private repository to a private instance of Jenkins can be tricky. To do the GitHub setup, make sure that internet connectivity is present in the machine where Jenkins is installed.

**Q2. Install and setup Jenkins.**

Jenkins is an open-source automation tool written in Java programming language that allows continuous integration.

Jenkins builds and tests our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously deliver our software by integrating with a large number of testing and deployment technologies. Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

For example: If any organization is developing a project, then Jenkins will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

* Perform a software build using a build system like Gradle or Maven Apache
* Execute a shell script
* Archive a build result
* Running software tests

# **Practical - 3**

**Continuous Delivery using Jenkins**

In Jenkins, a pipeline is a collection of events or jobs that are interlinked with one another in a sequence.

It is a combination of plugins that support the integration and implementation of continuous delivery pipelines using Jenkins.

In other words, a Jenkins Pipeline is a collection of jobs or events that brings the software from version control into the hands of the end users by using automation tools. It is used to incorporate continuous delivery in our software development workflow.

A pipeline has an extensible automation server for creating simple or even complex delivery pipelines "as code", via DSL (Domain-specific language).

**DevOps**

# **Practical - 4**

**Containerization using Docker**

Docker is a centralised platform for packaging, deploying, and running applications. Before Docker, many users faced the problem that a particular code is running in the developer's system but not in the user's system. So, the main reason to develop docker is to help developers to develop applications easily, ship them into containers and can be deployed anywhere.

Docker is an open-source centralised platform designed to create, deploy, and run applications. Docker uses a container on the host's operating system to run applications. It allows applications to use the same Linux kernel as a system on the host computer, rather than creating a whole virtual operating system. Containers ensure that our application works in any environment like development, test, or production.

Docker includes components such as Docker client, Docker server, Docker machine, Docker hub, Docker compose, etc.

**Docker Containers**

Docker containers are lightweight alternatives to the virtual machine. It allows developers to package up the application with all its libraries and dependencies, and ship it as a single package. The advantage of using a docker container is that you don't need to allocate any RAM and disk space for the applications. It automatically generates storage and space according to the application requirement.

**Docker Engine**

* It is a client-server application that contains the following major components.
* A server is a type of long-running program called a daemon process.
* The REST API is used to specify interfaces that programs can use to talk to the daemon and instruct it on what to do.
* A command line interface client.

**DevOps**

# **Practical - 5**

**Configuration Management using Puppet**

Configuration management occurs when a configuration platform is used to automate, monitor, design, and manage otherwise manual configuration processes. System-wide changes take place across servers and networks, storage, applications, and other managed systems.

An important function of configuration management is defining the state of each system. By orchestrating these processes with a platform, organizations can ensure consistency across integrated systems and increase efficiency. The result is that businesses can scale more readily without hiring additional IT management staff. Companies that otherwise wouldn’t have the resources can grow by deploying a DevOps approach.

Configuration management is closely associated with change management, and as a result, the two terms are sometimes confused. Configuration management is most readily described as the automation, management, and maintenance of configurations at each state, while change management is the process by which configurations are redefined and changed to meet the conditions of new needs and dynamic circumstances.

Several tools are available for those seeking to implement configuration management in their organizations. Puppet has carried the torch in pioneering configuration management, but other companies like Chef and Red Hat also offer intriguing suites of products to enhance configuration management processes. Proper configuration management is at the core of continuous testing and delivery, two key benefits of DevOps.

Based on what we’ve discussed, you may have already learned that configuration management takes on the primary responsibility for three broad categories required for DevOps transformation: identification, control, and audit processes.

**Identification:** The process of finding and cataloging system-wide configuration needs.

**Control:**During configuration control, we see the importance of change management at work. It’s highly likely that configuration needs will change over time, and configuration control allows this to happen in a controlled way so as to not destabilize integrations and existing infrastructure.

**Audit:** Like most audit processes, a configuration audit is a review of the existing systems to ensure that it stands up to compliance regulations and validations.

Like DevOps, configuration management is spread across both operational and development buckets within an organization. This is by design. There are primary components that go into the comprehensive configuration management required for DevOps:

● Artifact repository

● Source code repository

● Configuration management data architecture

**What is Puppet?**

* Puppet is a **DevOps configuration management tool**. This is developed by Puppet Labs and is available for both open-source and enterprise versions. It is used to centralize and automate the procedure of configuration management.
* This tool is developed using Ruby DSL (domain-specific language), which allows you to change a complete infrastructure in code format and can be easily managed and configured.
* Puppet tool deploys, configures, and manages the servers. This is used particularly for the automation of hybrid infrastructure delivery and management.
* With the help of automation, Puppet enables system administrators to operate easier and faster.
* Puppet can also be used as a deployment tool as it can deploy software on the system automatically. Puppet implements infrastructure as a code, which means that you can test the environment for accurate deployment.
* Puppet supports many platforms such as Microsoft Windows, Debian/Ubuntu, Red Hat/CentOS/Fedora, MacOS X, etc.
* Puppet uses the client-server paradigm, where one system in any cluster works as the server, called the puppet master, and other works as a client on nodes called a slave.

**Puppet Blocks**

Puppet provides the flexibility to integrate Reports with third-party tools using Puppet APIs.

Four types of Puppet building blocks are

1. Resources
2. Classes
3. Manifest
4. Modules

**Puppet Resources:**

Puppet Resources are the building blocks of Puppet.

Resources are the inbuilt functions that run at the back end to perform the required operations in Puppet.

**Puppet Classes:**

A combination of different resources can be grouped together into a single unit called a class.

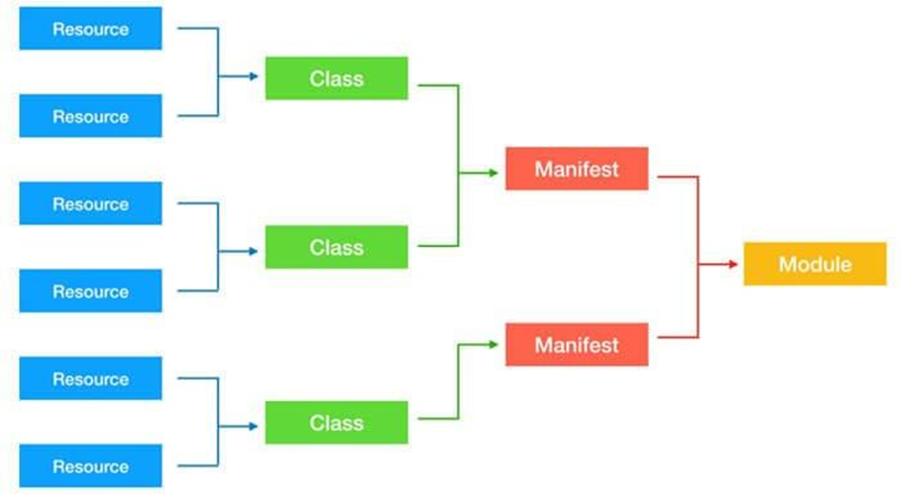
**Puppet Manifest:**

Manifest is a directory containing puppet DSL files. Those files have a .pp extension. The .pp extension stands for puppet program. The puppet code consists of definitions or declarations of Puppet Classes.

**Puppet Modules:**

Modules are a collection of files and directories such as Manifests and class definitions. They are the reusable and shareable units in Puppet.

For example, the MySQL module to install and configure MySQL or the Jenkins module to manage Jenkins, etc.



Windows nodes cannot serve as puppet master servers.

● If your Windows nodes will be fetching configurations from a puppet master, you will need a \*nix server to run as a puppet master at your site.

● If your Windows nodes will be compiling and applying configurations locally with Puppet apply, you should disable the Puppet agent service on them after installing Puppet.

# **Practical - 6**

**Kubernetes**

Kubernetes is also known as 'k8s'. This word comes from the Greek language, which means a pilot or helmsman.

Kubernetes is an extensible, portable, and open-source platform designed by Google in 2014. It is mainly used to automate the deployment, scaling, and operations of container-based applications across the cluster of nodes. It is also designed for managing the services of containerized apps using different methods that provide scalability, predictability, and high availability.

It is actually an enhanced version of 'Borg' for managing long-running processes and batch jobs. Nowadays, many cloud services offer a Kubernetes-based infrastructure on which it can be deployed as a platform-providing service. This technique or concept works with many container tools, like docker, and follows the client-server architecture.

The following are the key objects that exist in Kubernetes:

**Pod**

It is the smallest and simplest basic unit of the Kubernetes application. This object indicates the processes which are running in the cluster.

**Node**

A **node** is nothing but a single host, which is used to run the virtual or physical machines. A node in the Kubernetes cluster is also known as a minion.

**Service**

A **service** in Kubernetes is a logical set of pods, which works together. With the help of services, users can easily manage load-balancing configurations.

**ReplicaSet**

A **ReplicaSet** in the Kubernetes is used to identify the particular number of pod replicas are running at a given time. It replaces the replication controller because it is more powerful and allows a user to use the "set-based" label selector.

**Namespace**

**Kubernetes** supports various virtual clusters, which are known as namespaces. It is a way of dividing the cluster resources between two or more users.

# **Practical - 7**

**JIRA**

JIRA is a software testing tool developed by the Australian Company Atlassian. It is a bug-tracking tool that reports all the issues related to your software or mobile apps. The word JIRA comes from the Japanese word, i.e., "Gojira" which means Godzilla.

JIRA is based on the Agile methodology and the current version of the Jira is 6.

The following are the useful aspects provided by the Jira:

* Projects: It is used to manage defects very effectively.
* Issue: It is used to track and manage defects/issues.
* Workflow: Processes the Issue/Defect life cycle. Suppose we have a business requirement, we create the technical design and from the technical design, we create the test cases. After creating the test cases, coding is done, and then testing is performed on the project. This design workflow is possible by using Jira.
* Search: Find with ease. Suppose we have done with a project at the beginning of December and its version is 1.0. Now, we move to version 1.1 and complete at the end of December. What we are doing is that we are adding new versions. Through Jira, we can get to know what happened in the earlier versions, how many defects occurred in the earlier projects, and the learning we achieved from the earlier projects.
* Dashboards: A dashboard is a display that you see when you log in to the Jira. You can create multiple dashboards for multiple projects. You can create the personal dashboard and can add the gadgets in a dashboard so that you can keep track of the assignments and issues that you are working on.
* Plan, Track and Work Faster  
  JIRA is a bug-tracking tool mainly used to track, organize, and prioritize the bugs, newly added features, improvements for certain software releases. Projects are subdivided into issues and issues can be of multiple types such as bug, new feature, improvement, and documentation tasks.  
  When the release date of software comes near, then software developers need to focus on the remaining issues which are to be fixed before the specified date. It also becomes difficult for the QA to maintain the status of the documentation, i.e., sometimes it becomes hard to keep track of everything.  
  JIRA is a good choice for handling the above issues. It enables software developers to track issues and improvements. It manages the projects as well as maintain the technical documentation.
* The main source of information  
  JIRA is the primary source of information for the next software release. On JIRA, the whole team of the software developers can plan for the new features which are to be added and bugs to be fixed in the next release.  
  It also helps the QA team in writing the technical documentation. Through JIRA, the QA team can check the status of each feature that is newly added by the software developers, and according to that, they can plan how to document for the new version.
* Organize the documentation tasks  
  JIRA tool is used to organize the documentation tasks. It is useful in grouping the multiple tasks by using the component functionality, and even you can create your own documentation. In this way, you can create a structured way of documentation.
* Track the progress of our documentation  
  It is a very useful tool in tracking the progress of our documentation. JIRA tool provides a very important feature, i.e., pie chart macro. In the pie chart macro, you can view tasks such as Open tasks, Closed tasks, Resolved tasks.
* Helps to meet the deadlines of a documentation release.  
  You can define the specific due date or deadline for the release of documentation, and even you can configure the JIRA tool with the notifications so that you can finish your documentation in time.
* Measures the time spent on documentation  
  JIRA tool does not have the default functionality for measuring the time spent on documentation. JIRA tool is bundled with the Tempo Timesheets, which measures how much time has been spent on the documentation.
* Provides feedback faster  
  JIRA tool provides the Confluence pages where you can connect to the issues in just a few clicks. If something needs to be updated, then you can create the issues directly from the Confluence page.