Theory:

Version Control System.

Version Control System is a method to record changes to a file over time so that you can recall specific versions later. In the case of software, you will use your software source code as the files being version controlled, though in reality, you can do this with nearly any type of file on a computer.

Types of Version Control System:

- Local VCS.
- Centralised VCS
- Distributed VCS.

Importance of Version Control in DevOps:

1) Avoiding Dependency issues in modern applications.
2) Providing better performance
3) Improving application Reliability.

The systems used to track changes and different code versions. Just like with anything else, each VCS has its unique features and comes with its own set of advantages & disadvantages.

Source Code Management

Used to track modifications to a source code repository. SCM tracks a running history of changes to a code base and helps to resolve conflicts when merging updates from multiple contributors. SCM is also synonymous with Version control.

Git

It is a distributed version control system (DVCS) for tracking changes to files. But what does that mean? Git is an open source VCS, which is not file based, unlike other systems. Rather, it stores information as snapshots.

Git Terminologies:

1. Repository.
2. Branch
3. Fork.
4. Remote.
5. HEAD
6. Master
7. Merge
8. Origin
9. Pull requests.
10. Push
11. Stash

Theory:

Git commands are used like utilities to interact with our version control system. After installing git, one can get the list of all available commands by typing 'git --help' in the CLI.

List of frequently used commands are to cheat sheet:

1. git add
2. git branch
3. git checkout
4. git clean
5. git clone
6. git commit
7. git commit --amend
8. git config
9. git fetch

10. git init
11. git log
12. git merge
13. git pull
14. git push
15. git rebase
16. git rebase -i
17. git reflog
18. git remote
20. git reset
21. git revert
22. git status.

Some of these are implemented in the below attached snapshots.

Theory:

CONTINUOUS INTEGRATION (CI)

Development practice where the development teams make small, frequent changes to the code. An automated build verifies the code each time developers check their changes into the version control repository.

Advantages:

- Reduced integration risk.
- Higher code quality.
- The code in version control works.
- Reduced friction between team members.
- quality of life improvement for testers.
- less time deploying projects.

There are various tools used in the industry to implement CI/CD in a ~~manag~~ organized manner. Some of the most famous ones are:
Jenkins, Gradle, etc.

JENKINS.

Jenkins is an open-source server that is written entirely in Java. It lets you execute a series of actions to achieve the continuous integration process, that too in an automated fashion.

The CI server runs in servlet containers such as Apache Tomcat. Jenkins facilitates continuous integration and continuous delivery in software projects by automating parts related to build, test and deployment. This makes it easy for developers to continuously work on the betterment of the project by integrating changes to the product.

Jenkins automates the software builds in a continuous manner and lets the developers know about the errors at an early stage. A strong Jenkins community is one of the prime reasons for its popularity.

Jenkins is not only extensible but also has a thriving plugin system.

Theory:

APACHE MAVEN

Maven is a build automation tool primarily used for Java people/projects. It provides a standard way to build, package and deploy Java applications, as well as manage their dependencies.

It uses a project object model (POM) to manage the build process. The POM is an XML file that describes the project's configuration, dependencies, build profiles, and other details.

Maven uses the POM to download dependencies, compile source code, create executable JAR files, run tests and deploy the application to a server.

It is widely used in the Java community and is supported by popular integrated development environments (IDEs) such as Eclipse and IntelliJ IDEA. It is a valuable tool for managing complex Java projects and ensuring that builds are consistent and reproducible across different environments.

A pipeline is an important concept in software development because it allows for the automation of the entire software delivery process, from development through testing and deployment. By breaking down the software delivery process into smaller, more manageable stages, a pipeline can help teams to increase efficiency, improve quality & reduce errors.

Reasons why a pipeline is important:

1) Automation: reduces risk of human error, increases speed, makes it easier & consistent.

2) Collaboration: promotes better communication.

3) Quality Assurance: easier to test and validate, hence assuring quality.

4) Continuous delivery: allows teams to respond more quickly to customer needs & feedback.

5) Scalability: a pipeline can be easily scaled to handle larger or more complex projects.

Theory:

Jenkins is an open source automation server that is widely used for continuos integration and continuos delivery of structure of software applications.

One of the key feature of the Jenkins is its ability to distribute workloads across multiple machines using master-slave architecture.

In the Jenkins master-slave architecture, the master node is responsible for managing and scheduling Jobs.

While the slave nodes are responsible for executing actual jobs. The master node communicates with the slave nodes through a designated protocol such as SSH or JNLP.

When a job is submitted to Jenkins, the master node determines which slave node is the best suited for the job based on factors such as availability, workload and capabilities.

The master node then sends the job to selected slave node, which executes the job and sends results back to the master node

Using a master-slave architecture offers several benefits for jenkins users. such as:

i) Efficient workload.
ii) Distributed work.
iii) Increased Reliability
iv) Scalability
v) Increased or decreased no. of nodes.

In addition, the master slave architecture enables users to run jobs on different platforms and operating system, as well as take advantage of specialized hardware or software configuration on the slave node.

Overall jenkins master slave architecture is powerful feature that allows for efficient and scalable automation of software development and delivery workflows.

Theory:

Selenium is an open source automation testing tool that is widely used to automate web browsers, for testing web applications.
It supports a range of programming languages such as Java, Python, C# and Ruby.
It can also run on different OS like Windows, Linux and Mac.

Selenium provides a suite of tools that enables developers and testers to write automated test scripts, which can then be executed against a web application.

Selenium consists of various components:

i) SELENIUM IDE
A firefox add-on that records user action and generate test scripts.

ii) Selenium web driver.
A library that provides a programming interface to control the browser and perform operations such as clicking buttons, filling out forms and navigating between pages.

iii) Selenium add

A tool that allows users to run tests on multiple browsers & platforms simultaneously.

It is popular due to its features of

a) Versatility
b) Flexibility
c) Efficiency
d) Ease of use
e) Automation
f) Quality Assurance
g) Reliability

It is also used for

a) Functional testing
b) Regression testing
c) load testing
d) Test scripting
e) integration with testing framework

THEORY:

DOCKER

Linux based, open source, containerization platform that developers use to build, run & package applications for deployment using containers.

Unlike Virtual Machines, Docker containers offers:

- OS level abstraction
- Optimum resource utilization.
- Interoperability.
- Efficient building.
- Efficient testing.
- Faster application execution.
- Fundamental modularization.

Provides functionality into multiple components that allow deploying, testing or scaling them independently when needed.

Take for instance, a docker containerized database of an application. With such a framework you can scale or maintain the database independently from other modules/components of the application without impacting the workloads of other critical systems.

* Components of Docker Architecture

Docker comprises for following different components within its core architecture.

A) Images.

Images are like blueprints containing instructions for creating a docker container image. These define

- Application dependencies.
- process that should run when the application launches.

B) Containers

Live instances of images on which an application or its independent modules are run.

In an OOP analogy, an image is a class and container.

C) Registeries.

A docker registry is like a repository of Images. The default registry is the Docker Hub & public registry that stores public and official images.

For different languages and platforms by default, a request for an image from Docker is searched within the Docker Hub registry.

Theory:

Docker can build images automatically by reading the instructions from a Docker file. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

These instructions are not case-sensitive. However, the convention is for them to be UPPERCASE to distinguish them from arguements more easily.

Docker runs instructions in a Dockerfile in an order. It must begin from a FROM instruction. This may be after parser directives, comments and globally scoped ARG's.

The FROM instruction specifies the Parent image from which you are building. FROM may only be preceded by one or more ARG instructions, which declare arguments that are used in FROM lines in the Dockerfile.

Docker treats lines that begin with # as a command unless the line is a valid parser directive. A # marker anywhere else in a line is treated as an argument.

SEPM

Experiment 10

Aim:

To install & configure Pull based Software Configuration management and provisioning tools using Puppet.

Theory:

Puppet is a configuration management tool that is used for deploying, configuring and managing servers.

It performs the following functions:

1. Defining distinct configurations.
2. Scaling up and Scaling down dynamically
3. Centralized (master-slave or repobased) control

It uses a Master-Slave architecture in which the Master and Slave communicate through a secure encrypted channel with the help of SSL.

Key features of Puppet:

1. Large installed / user base.
2. Large developer base.
3. Long commercial track record.
4. Documentation.
5. Platform support.
6. Scalable
7. Consistent
8. Portable infrastructure
9. Flexible
10. Infrastructure insights.

CONCLUSION

Hence, we have successfully learned about Puppet, its features, importance and also implemented Pull based Software Configuration management.

SEPM

Experiment 11

Aim: To provide a LAMP/MEAN stack using Puppet Manifest

Theory:

A LAMP stack is a ~~but~~ bundle of four different software techniques that develops use to build websites and web applications.

LAMP is an acronym for the operating system : Linux;
web server : Apache;
database server : MySQL;
programming language : PHP.

All four of these techniques are open source, which means they are community maintained and freely available for anyone to use.

Developers use LAMP stacks to create, host and maintain web content. It is a popular solution that powers many of the websites you commonly use today.

LAMP provides the following features

- Cost-effective
- Efficiency
- Low-maintainence
- High-support
- Flexible.

LAMP stack working:

Web applications use LAMP stack to respond to requests from web browsers. When you open a web page in a browser, the stack goes through the process

1) Receiving requests
2) Processing requests
3) Sending responses.

PERL or Python is used as an alternative variation in LAMP stacks.

CONCLUSION

We have studied the features, importance, working of the LAMP stack and implemented it using Puppet Manifest.