**Lab Environment Documentation summary**

**Table of Contents**

**1. Proxmox Virtual Environment (VE)**

- Overview

- Access Details

- Version and Release Date

- Service Nodes

**2. Lab Environment Details**

- Jenkins Cluster

- Services and IP Addresses

- Kubernetes Cluster

- Nodes and IP Addresses

- Virtual Machine Templates

**3. Comprehensive Lab Environment Configuration Guide**

- Overview

- Version Control

- Git

**Build and Development Tools**

- Maven

- Java

- Node.js and npm

- .NET

- Python3

- boto3

**Continuous Integration and Delivery (CI/CD)**

- Jenkins

- Plugins Installed

**Containerization and Orchestration**

- Docker

- Docker Compose

- Kubernetes

- k9s

**Infrastructure as Code (IaC)**

- Terraform

**Monitoring and Visualization**

- Prometheus

- Grafana

- EFK Stack

**Additional Tools**

- Helm

- Argo CD

- SonarQube

**Environment Configuration**

- Operating System

- Security Measures

**Additional Note**

- Automation Scripts

**4. Application Repositories for Practice**

- Gradle Basic Application

- Repository and Description

- Python Flask Application

- Repository and Description

- Recipe Management System

- Repository and Description

- Additional Repositories

- Node Port Application

- Kompose + StatefulSet Application

- Ingress Application

**5. Lab DevOps Tools**

1. Maven

- Definition and Lab Installation

2. Docker

- Definition and Lab Installation

3. Docker Compose

- Definition and Lab Installation

4. Kubernetes

- Definition and Lab Installation

- Cluster Configuration

Additional Information

- K9s Integration

- Application Monitoring

- Resource Management

- Namespace Management

- Ingress and Networking

- Persistent Storage

5. Helm

- Definition and Lab Installation

6. Jenkins

- Definition and Introduction

- Jenkins Cluster Configuration

- URL, Username, and Password

- Service IP Addresses

- Lab Jenkins Environment

- Virtualization Platform

- Network Configuration

- Storage

- Security

- Backup and Recovery

7. Argo CD

- Definition and Introduction

- Lab Installation

- Access Details

8.Grafana

- Definition and Introduction

- Lab Installation

- Access Details

9. EFK Stack (Elasticsearch, Fluentd, Kibana)

- Definition and Introduction

- Lab Installation

- Additional Information

10. **Updates**

* Updates will be added below this

**Pre-requisites**  
This lab manual is only a companion on your DevOps learning journey. We assume you are using this manual and lab environment to try out things you are learning from a course or some other learning material. Platform Engineering team recommends Udemy course [**DevOps Master Class 2024**](https://www.udemy.com/course/devops-training/?couponCode=NVDPRODIN35)**.** It has got wide and deep content and seems to be the best DevOps training available on Udemy at the time of writing this manual. Over 45K students have taken this course and it has 4.6 rating based on 5K ratings. We assume you have:

* Good understanding of Git/GitHub, including setting up a repository, check out code, etc.
* Good understanding of Jenkins, setting up pipelines with integration to automated tests, code quality scanning tools, etc.
* Understanding of Docker and Kubernetes concepts, creating docker containers for applications, configuring K8s cluster and so on
* Understanding of Git Ops using Argo CD
* Familiarity with Code quality scanning tools, like SonarQube.  
    
  **Access Request:**  
  Please raise a request for access to the following URLs by using ManageEngine.
* ManageEngine URL: <https://optitcloud.com/>
* Site: Platform Engineering

**Use the Below Template:**

* Training date:
* Username:( Use Optit email):
* Once the access is provided you will receive an email and ticket will be closed.

**Proxmox Virtual Environment (VE)**

Proxmox VE is an open-source server virtualization management platform that combines Kernel-based Virtual Machine (KVM) and Linux Containers (LXC). It provides a web-based interface for managing virtual machines, containers, storage, and networking. Proxmox VE supports high availability clusters, backup and restore, and live migration. It is widely used for its powerful features and ease of integration into existing IT infrastructure.

**Proxmox VE Access:**

|  |  |
| --- | --- |
| **URL:** | [https://10.10.30.20:8006/v1:0:=qemu%2F111:4:::::21::](https://10.10.30.20:8006/#v1:0:=qemu%2F111:4:::::21::) |
| **Username:** | (Your email address) |
| **Password:** | Optit@123 |

**Lab Environment Details.**

**Proxmox VE:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lab Environment Details.**   |  |  | | --- | --- | | Version: | Version: 7.4-3 | | Release Date | April 19, 2023 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **prerequisites.**   |  |  | | --- | --- | | Proxmax Cluster Access | VM (OS: Ubuntu,RAM) | | Proxmax Cluster | URL:10.10.30.20:8006 | | min 4Gb, CPU min 2 cores | Disk: min 10gb | |

**Installation of VM in Proxmox**

Step 1: Access the Proxmox Web Interface

1. Open your web browser and navigate to the Proxmox URL: `http://10.10.30.20:8006`

2. Log in with your Proxmox credentials.

Step 2: Create a New Virtual Machine

1. In the Proxmox interface, click on the `Create VM` button at the top right corner.

2. In the `General` tab, enter a name for your VM.

3. Select the appropriate `Node` where you want the VM to be created.

4. Click `Next` to proceed.

Step 3: Configure the OS Installation

1. In the `OS` tab, select the location of your installation media. If you have uploaded an ISO image, select it from the dropdown list.

2. Click `Next`.

Step 4: Configure the System

1. In the `System` tab, configure the following settings:

- BIOS: Leave as `Default` unless you have specific requirements.

- Machine: Leave as `Default` unless you have specific requirements.

- SCSI Controller: Leave as `Default`.

2. Click `Next`.

Step 5: Configure the Hard Disk

1. In the `Hard Disk` tab, set the following:

- Bus/Device: Leave as `Default`.

- Storage: Select the storage where you want the VM's disk to be stored.

- Disk Size: Set the disk size (minimum 10GB).

2. Click `Next`.

Step 6: Configure the CPU

1. In the `CPU` tab, configure the following:

- Sockets: 1

- Cores: Minimum 2 cores.

- Type: Leave as `Default`.

2. Click `Next`.

Step 7: Configure the Memory

1. In the `Memory` tab, set the memory size to a minimum of 4GB (4096MB).

2. Click `Next`.

Step 8: Configure the Network

1. In the `Network` tab, configure the following:

- Bridge: Select the network bridge (e.g., `vmbr0`).

- Model: Leave as `Default`.

2. Click `Next`.

Step 9: Confirm and Create the VM

1. In the `Confirm` tab, review all the configurations.

2. Click `Finish` to create the VM.

Step 10: Start and Install the OS

1. Once the VM is created, it will appear in the Proxmox interface under the selected node.

2. Select the VM and click on the `Start` button.

3. Click on the `Console` tab to open the VM console.

4. Follow the installation instructions for your chosen operating system (e.g., Ubuntu) to complete the OS installation.

Step 11: Post-Installation Steps

1. Once the OS is installed, log in to the VM.

2. Update the package list and upgrade the packages:

```

sudo apt update

sudo apt upgrade -y

```

3. Install any additional packages or perform any necessary configuration based on your requirements.  
  
**Installation of SSH Client**

Install the SSH package on Ubuntu by typing:

|  |
| --- |
| sudo apt install openssh-client |
| sudo apt install openssh-server |

Once the installation is done, use it by typing from the CLIENT:

|  |
| --- |
| ssh user\_name@server\_private\_ip |

**Current Running Service Nodes.**

**Jenkin Cluster**

|  |  |
| --- | --- |
| Service | IP Address |
| Jenkins | 10.10.30.77 |
| Jenkins Slave Node | 10.10.30.70 |
| PE | 10.10.30.78 |

**Kubernetes Cluster**

|  |  |
| --- | --- |
| Node | IP Address |
| K8s Master Node | 10.10.30.87 |
| K8s Node 1 | 10.10.30.88 |
| K8s Node 2 | 10.10.30.89 |

**Virtual Machine Templates**

|  |  |  |
| --- | --- | --- |
| Virtual Machine | ID | Description |
| Ubuntu Template | 117 | Ubuntu 20.04 Template |
| Ubuntu 22.04 Template | 999 | Ubuntu 22.04 Template |
| Platform Engineering | 111 | Platform Engineering VM |

Additional Information  
-

**Comprehensive List of Installed Tools and Technologies in Lab Environment**  
  
**Overview**  
The lab environment is configured with various tools and technologies to support development, continuous integration, and deployment. The setup includes version control systems, build tools, CI/CD tools, containerization, monitoring, and more.

**1. Version Control**

- Git: 2.35.1

Purpose: Version control for source code management.

**Installation Steps:**

```bash

sudo apt update

sudo apt install git

git --version

```

**2. Build and Development Tools**

**- Maven: 3.8.5**

Purpose: Build automation tool for Java projects.

**Installation Steps:**

```bash

sudo apt update

sudo apt install maven

mvn -version

```

**- Java: 11 (with Java 17 also installed)**

Purpose: Java Development Kit (JDK) for building and running Java applications.

**Installation Steps:**

**To install the OpenJDK JRE:**

```bash

sudo apt install default-jre

```

**Check if OpenJDK JRE was properly installed by running:**

```bash

java -version

```

**Create a directory to install JRE:**

```bash

sudo mkdir /usr/local/java

sudo mv jre-8u291-linux-x64.tar.gz /usr/local/java

cd /usr/local/java

sudo tar zxvf jre-8u291-linux-x64.tar.gz

```

**Post-installation steps:**

```bash

sudo rm jre-8u291-linux-x64.tar.gz

sudo update-alternatives --install "/usr/bin/java" "java" "/usr/local/java/jre1.8.0\_291/bin/java" 1

```

**Verify the installation by running:**

```bash

java -version

```

**- Node.js and npm**

Purpose: JavaScript runtime and package manager for front-end and server-side development.

**Installation Steps:**

```bash

sudo apt update

sudo apt install nodejs npm

node -v

npm -v

```

**- .NET**

Purpose: Framework for building and running .NET applications.

**Installation Steps:**

```bash

wget https://packages.microsoft.com/config/ubuntu/20.04/prod.list

sudo mv prod.list /etc/apt/sources.list.d/

wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | sudo apt-key add -

sudo apt update

sudo apt install dotnet-sdk-6.0

dotnet --version

```

**- Python3**

Purpose: Programming language for scripting and automation tasks.

**Installation Steps:**

```bash

sudo apt update

sudo apt install python3 python3-pip

python3 --version

pip3 --version

```

**- boto3**

Purpose: AWS SDK for Python, allowing Python applications to interact with AWS services.

**Installation Steps:**

```bash

pip3 install boto3

```

**3. Continuous Integration and Delivery (CI/CD)**

**- Jenkins: 2.331**

Purpose: Continuous integration and delivery (CI/CD) server.

**Installation Steps:**

```bash

sudo apt update

sudo apt install openjdk-11-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

sudo systemctl start jenkins

sudo systemctl status jenkins

```

**4. Containerization and Orchestration**

**- Docker**

Version: latest (or specify a particular version if needed)

Purpose: Containerization platform for building, running, and managing containers.

**Installation Steps:**

```bash

sudo apt update

sudo apt install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

sudo apt update

sudo apt install docker-ce

sudo systemctl status docker

```

**- Docker Compose**

Purpose: Tool for defining and running multi-container Docker applications.

**Installation Steps:**

```bash

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

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

docker-compose --version

```

**- Kubernetes**

Purpose: Container orchestration platform for managing containerized applications.

**Installation Steps:**

```bash

curl -LO "https://dl.k8s.io/release/v1.27.0/bin/linux/amd64/kubectl"

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

kubectl version --client

```

**- k9s**

Purpose: Terminal-based Kubernetes UI for managing and monitoring Kubernetes clusters.

**Installation Steps:**

```bash

curl -sSLO https://github.com/derailed/k9s/releases/download/v0.27.1/k9s\_Linux\_x86\_64.tar.gz

tar zxvf k9s\_Linux\_x86\_64.tar.gz

sudo mv k9s /usr/local/bin/

k9s version

```

**5. Infrastructure as Code (IaC)**

**- Terraform: 1.1.0**

Purpose: Tool for provisioning and managing cloud resources.

**Installation Steps:**

```bash

wget https://releases.hashicorp.com/terraform/1.1.0/terraform\_1.1.0\_linux\_amd64.zip

unzip terraform\_1.1.0\_linux\_amd64.zip

sudo mv terraform /usr/local/bin/

terraform version

```

**6. Monitoring and Visualization**

**- Prometheus**

Purpose: Monitoring and alerting toolkit for containerized applications and infrastructure.

**Installation Steps:**

```bash

wget https://github.com/prometheus/prometheus/releases/download/v2.40.0/prometheus-2.40.0.linux-amd64.tar.gz

tar xvf prometheus-2.40.0.linux-amd64.tar.gz

cd prometheus-2.40.0.linux-amd64

./prometheus --version

```

**- Grafana**

Purpose: Visualization and analytics platform for monitoring data.

**Installation Steps:**

```bash

wget https://dl.grafana.com/enterprise/release/grafana-enterprise\_9.5.0\_amd64.deb

sudo dpkg -i grafana-enterprise\_9.5.0\_amd64.deb

sudo systemctl start grafana-server

sudo systemctl status grafana-server

```

**- EFK Stack**

**Components:**

- Elasticsearch: Search and analytics engine.

- Fluentd: Log collector and forwarder.

- Kibana: Visualization tool for Elasticsearch data.

**Installation Steps:**

**- Elasticsearch:**

```bash

wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-8.5.0-amd64.deb

sudo dpkg -i elasticsearch-8.5.0-amd64.deb

sudo systemctl start elasticsearch

sudo systemctl status elasticsearch

```

**- Fluentd:**

```bash

curl -L https://toolbelt.treasuredata.com/sh/install-debian-buster-td-agent3.sh | sh

sudo systemctl start td-agent

sudo systemctl status td-agent

```

**- Kibana:**

```bash

wget https://artifacts.elastic.co/downloads/kibana/kibana-8.5.0-amd64.deb

sudo dpkg -i kibana-8.5.0-amd64.deb

sudo systemctl start kibana

sudo systemctl status kibana

```

**7. Additional Tools**

**- Helm**

Purpose: Kubernetes package manager for managing Helm charts.

**Installation Steps:**

```bash

curl -L https://get.helm.sh/helm-v3.11.1-linux-amd64.tar.gz | tar xz

sudo mv linux-amd64/helm /usr/local/bin/helm

helm version

```

**- Argo CD**

Purpose: Continuous delivery tool for Kubernetes, providing GitOps capabilities.

**Installation Steps:**

```bash

curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-cd/releases/download/v2.7.3/argocd-linux-amd64

chmod +x argocd-linux-amd64

sudo mv argocd-linux-amd64 /usr/local/bin/argocd

argocd version

```

**- SonarQube: 8.9.4-community**

Purpose: Code quality and security analysis.

**Installation Steps:**

```bash

wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.9.4.51709.zip

unzip sonarqube-8.9.4.51709.zip

sudo mv sonarqube-8.9.4.51709 /opt/sonarqube

sudo useradd --system --home /opt/sonarqube sonarqube

sudo chown -R sonarqube:sonarqube /opt/sonarqube

sudo -u sonarqube /opt/sonarqube/bin/linux-x86-64/sonar.sh start

```

**Lab Environment**  
In our lab environment, we've installed and configured various tools and platforms to provide a playground for experimentation and learning. This setup allows users to explore different technologies and tools, enhancing their understanding of modern development and deployment processes.

The Opt-IT Platform Engineering team has created this lab for engineers to gain hands-on experience with CI/CD. It complements training courses on CI/CD by offering practical applications. This manual provides all necessary information to maximize the lab environment's potential.

In this lab, you'll learn to set up and configure a CI/CD pipeline using Git/GitHub for Source Code Management (SCM), Jenkins for Continuous Integration (CI), and ArgoCD, Docker, and Kubernetes for Continuous Deployment (CD). You'll understand the workflow of Jenkins and ArgoCD pipeline stages and the integration between CI and CD pipelines.

**Git Repo Url S**  
 **Sample Python Microservice Applications:** <https://github.com/optit-cloud-team/optit-lab-python-microservice-example.git>LAB EXERCISE:  
Clone existing Jenkins pipeline jobs for Java, ReactJS, and Python, modify their configurations, and test them without affecting the lab assignment jobs. Jenkins Pipeline job for Java ,Python, Reactjs that includes cloning a Git repository, building a application, and then building and pushing a Docker image, Kubernetes deployment by using Argo cd

**Java:**

Pipeline job with clone the Java from git repository, Gradle build, sonar-qube scan, Docker build, docker push

**Python:**

Pipeline job with clone the git repository, Python build, Docker build, docker push

**React js:**

Pipeline job with clone the git repository, CraftsGuild, Docker build, docker push

**Identify Existing Jobs:**

* + Log in to your Jenkins server and navigate to the dashboard where your existing pipeline jobs are listed.
  + Identify the Java, ReactJS, and Python pipeline jobs that you want to clone.

**Modify Configuration:**

* + Access the configuration page of each cloned pipeline job.
  + Update the configuration as needed, such as adjusting build triggers, modifying build steps, or changing environment variables. Ensuring they don't impact the server's operation.
  + Ensure that you use the same Git repositories and Docker registry as specified in the original jobs. You may need to update repository URLs or credentials if necessary.
  + Verify that the cloned jobs do not interfere with the existing lab assignment jobs. You can achieve this by reviewing the configurations and ensuring they use different job names, triggers, and resources.

**Test Cloned Jobs:**

* + Run the cloned pipeline jobs to test the modified configurations.
  + Monitor the build logs and verify that the jobs execute successfully without errors.
  + Test the integration with Git repositories and Docker registry to ensure that the pipelines.

**Application Repositories for Practice**  
  
**Java:**

**Description** : A simple hello world application integrated with html,css,js. Pipeline job with clone the Java from git repository, Gradle build, sonar-qube scan, Docker build, docker push   
**Repository**:  
  
**Note: practice deployment by removing existing pipeline, Kubernetes, docker files.**

**Python:**

**Description** : A simple hello world application integrated with html,css,js. clone the git repository with Pipeline job, Python build, Docker build, docker push   
**Repository**:  
**Note: practice deployment by removing existing pipeline, Kubernetes, docker files.**

**React js:**

**Description** :An simple hello world application integrated with html,css,js .Pipeline job with clone the git repository, CraftsGuild, Docker build, docker push

**Repository**:  
**Note: practice deployment by removing existing pipeline, Kubernetes, docker files.**

**Gradle basic Application**

* **Description**: This simple service, built using Gradle, was used for testing Kubernetes, Docker, Argo CD, Jenkins (including shared libraries), and Grafana with Ingress using Helm. It offers an opportunity to experiment with different automation and deployment strategies.
* **Repository**: [Gradle Sample Service](https://github.com/optit-cloud-team/sample-service)

**Note: Readme contains detailed information and documentation**  
  
**Python Flask Application**

* **Description**: This project is built using Python and Flask, a lightweight web framework. It implements microservice communication with RabbitMQ, allowing each service to operate independently. The microservices architecture enhances modularity and maintainability.
* **Python Application Repository**:[Python Microservice Example](https://github.com/optit-cloud-team/optit-lab-python-microservice-example.git)

**Note: Readme contains detailed information and documentation**

**Recipe Management System**

* **Description**: The Recipe Management System is an MVC web application designed for managing food recipes. It supports creating, viewing, editing, and deleting recipes, with features for ingredient lists, cooking instructions, and difficulty levels. The application integrates with a MySQL database and includes detailed deployment instructions.
* **Spring Application Repositories:** [Spring MVC Example](https://github.com/optit-cloud-team/optit-lab-springmvc-example)

**Note: Readme contains detailed information and documentation**

**Additional Repositories for Experimentation:**

**Node Port Application:**

**Description**: This repository contains a Spring application configured to use Node Port for exposing services.

**Repository**:

**Note: Readme contains detailed information and documentation**

**Kompose + StatefulSet Application:**

**Description**: This repository showcases a Python application using Kompose to convert Docker Compose files into Kubernetes manifests, along with StatefulSets for managing stateful applications.

**Repository**:

**Note: Readme contains detailed information and documentation**

**Ingress Application:**

**Description**: This repository provides a setup for a Recipe Management System with Ingress for managing external access to the application.

**Repository**:

**Note: Readme contains detailed information and documentation**

**LAB DevOps TOOLS:**

**0. Git**

**Definition and Introduction:**

Git is a distributed version control system that helps developers track changes to source code over time. It enables multiple developers to collaborate on a project by maintaining a history of changes, branches, and mergers. Git is widely used for managing source code and coordinating development efforts in both small and large projects.

**Lab Installation:**

Git is installed and configured in the lab environment, enabling users to track code changes, collaborate on projects, and manage different versions of their codebase effectively. Users can create repositories, commit changes, branch, merge, and use other Git functionalities as part of their development workflow.

**Installation of Git in lab steps**

**Prerequisites**

You will need an Ubuntu 22.04 server with a non-root superuser account.

To set this up, you can follow our Initial Server Setup Guide for Ubuntu 22.04.

With your server and user set up, you are ready to begin.  
  
**Installing Git with Default Packages**

‘’’

sudo apt update

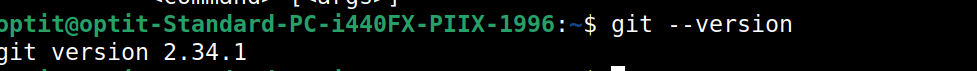
With the update complete, you can install Git:

sudo apt install git

‘’’

You can confirm that you have installed Git correctly by running the following command and checking that you receive relevant output.

git –version



With Git successfully installed, you can now move on to the Setting Up Git section of this tutorial to complete your setup.

**1. Maven**

**Definition and Introduction:**

* **Maven** is a build automation and project management tool primarily used for Java projects. It uses a Project Object Model (POM) file to manage project dependencies, build lifecycle, and plugins.

**Lab Installation:**

* Maven is installed and configured to manage dependencies and build processes for Java-based applications in the lab environment.

**Installation Steps:**

```bash

sudo apt update

sudo apt install maven

mvn -version

```

**2. Docker**

**Definition and Introduction:**

* **Docker** is a platform that allows developers to automate the deployment of applications inside lightweight, portable containers. Containers include everything needed to run the application, ensuring consistent environments across different stages of development and deployment.

**Lab Installation:**

* Docker is set up in lab environment, allowing users to containerize applications, experiment with container creation, and learn how to manage containerized environments.

**Docker Installation in lab step:**  
  
**Prerequisites**

To follow this tutorial, you will need the following

One Ubuntu 20.04 server set up by following the Ubuntu 20.04 initial server setup guide, including a sudo non-root user and a firewall

An account on Docker Hub if you wish to create your own images and push them to Docker Hub, as shown in Steps 7 and 8

Step 1 Installing Docker

The Docker installation package available in the official Ubuntu repository may not be the latest version. To ensure we get the latest version, we’ll install Docker from the official Docker repository to do that, we’ll add a new package source, add the GPG key from Docker to ensure the downloads are valid, and then install the package

First update your existing list of packages

‘’’

Sudo apt update

‘’’

Next, install a few prerequisite packages which let apt use packages over HTTPS

‘’’

Sudo apt install apt-transport-https ca-certificates curl software-properties-common

‘’’

Then add the GPG key for the official Docker repository to your system

‘’’

curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo apt-key add -

‘’’

Add the Docker repository to APT sources

‘’’

Sudo add-apt-repository "deb [arch=amd64] <https://download.docker.com/linux/ubuntu> focal stable"

‘’’

This will also update our package database with the Docker packages from the newly added repo.

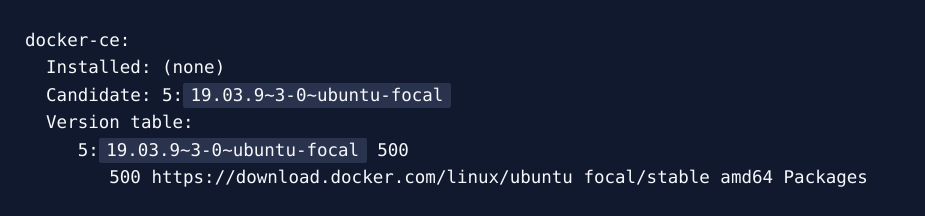
Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

‘’’

apt-cache policy docker-ce

‘’’

You’ll see output like this, although the version number for Docker may be different:



Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 20.04 (focal).

Finally, install Docker:

‘’’

sudo apt install docker-ce

‘’’

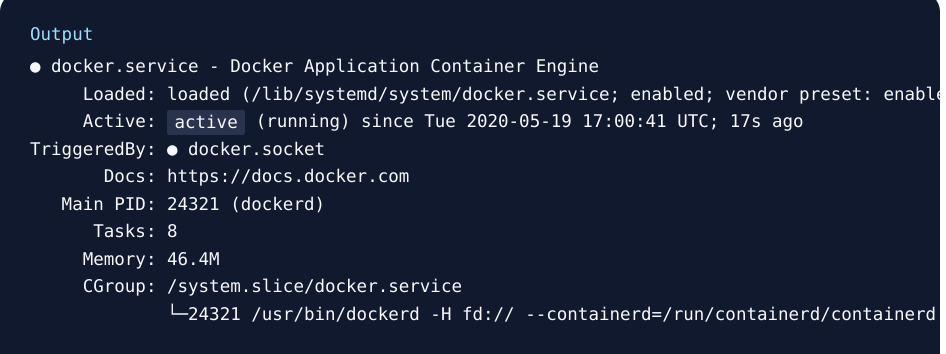
Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it’s running:

‘’’

sudo systemctl status docker

‘’’

The output should be like the following, showing that the service is active and running:



**3. Docker Compose**

**Definition and Introduction:**

* **Docker Compose** is a tool for defining and running multi-container Docker applications. With a single YAML file, you can configure the application's services, networks, and volumes.

**Lab Installation:**

* Docker Compose is installed and available for use in lab environment. Users can experiment with defining and managing multi-container applications, learning how different services interact within a containerized setup.

**install Docker Compose** **lab step**

Step 1: Install Docker Engine

Before installing Docker Compose, ensure Docker Engine is installed on your system. You can follow the installation instructions for Docker Engine [here](https://docs.docker.com/engine/install/).

Step 2: Download Docker Compose

1. Download the Docker Compose binary:

You can download the Docker Compose binary using `curl` or `wget`. Replace `1.29.2` with the desired version. To find the latest version, check the [Docker Compose releases page] (https://github.com/docker/compose/releases).

Using `curl`:

```bash

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

```

Using `wget`:

```bash

sudo wget "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -O /usr/local/bin/docker-compose

```

2. Apply executable permissions to the binary:

```bash

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

```

Step 3: Verify the Installation

Check that Docker Compose is installed correctly by running:

```bash

docker-compose --version

```

This should output the version of Docker Compose you installed.

Step 4: (Optional) Set Up Command Autocompletion

To enable command-line autocompletion, you can set it up by adding the following to your shell configuration file (e.g., `.bashrc` or `.zshrc`):

```bash

Add autocompletion for Docker Compose

if [ -f /etc/bash\_completion.d/docker-compose ]; then

. /etc/bash\_completion.d/docker-compose

fi

```

After adding this, reload your shell configuration with `source ~/.bashrc` or `source ~/.zshrc`.

Step 5: (Optional) Install Docker Compose as a Plugin

Docker Compose can also be installed as a Docker plugin:

1. Install the Docker Compose plugin:

```bash

docker plugin install docker/compose:latest

```

2. Verify the installation:

```bash

docker compose version

```

Notes:

- The installation commands may vary depending on your operating system (Linux, macOS, or Windows).

- For Windows and macOS, you can also install Docker Compose using Docker Desktop, which includes Docker Compose by default.

**Kubernetes**

**Definition and Introduction:**

* **Kubernetes** (K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of containers, ensuring high availability and scalability.

**Lab Installation:**

* Kubernetes is deployed in lab environment, providing a robust platform for users to experiment with orchestrating containerized applications, scaling deployments, and managing clusters. Kubernetes setup consists of a master node and two worker nodes:

**Kubernetes Cluster**

|  |  |
| --- | --- |
| Node | IP Address |
| K8s Master Node | 10.10.30.87 |
| K8s Node 1 | 10.10.30.88 |
| K8s Node 2 | 10.10.30.89 |

**Installation of SSH Client**

Install the SSH package on Ubuntu by typing:

|  |
| --- |
| sudo apt install openssh-client |
| sudo apt install openssh-server |

Once the installation is done, use it by typing from the CLIENT:

ssh user\_name@server\_private\_ip

**Kubernetes Installation and Setup Guide**

**Prerequisites:**

To follow this tutorial, you will need the following:

- One Ubuntu 20.04 server set up with a sudo non-root user and a firewall.

- Two additional Ubuntu 20.04 servers for worker nodes.

- Networking plugin (e.g., Calico, Flannel) for Kubernetes networking.

- A swap-free environment, as Kubernetes requires this.

- SSH access to all nodes in the cluster.

Step 1: Prepare the Servers

1. Update your system:

```bash

sudo apt update

sudo apt upgrade -y

```

2. Install required packages on all nodes:

```bash

sudo apt install -y apt-transport-https ca-certificates curl

```

3. Disable swap:

```bash

sudo swapoff -a

```

- To permanently disable swap, comment out the swap line in `/etc/fstab`.

Step 2: Install Docker Engine

1. Add Docker repository:

```bash

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"

```

2. Install Docker:

```bash

sudo apt update

sudo apt install -y docker-ce

```

3. Start and enable Docker:

```bash

sudo systemctl start docker

sudo systemctl enable docker

```

Step 3: Install Kubernetes Components

1. Add Kubernetes repository:

```bash

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

```

2. Install Kubernetes components:

```bash

sudo apt update

sudo apt install -y kubelet kubeadm kubectl

```

3. Hold the Kubernetes packages to prevent automatic updates:

```bash

sudo apt-mark hold kubelet kubeadm kubectl

```

Step 4: Initialize the Kubernetes Cluster (Master Node)

1. Initialize the master node:

```bash

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

```

- The `--pod-network-cidr` flag is required for some networking plugins.

2. Set up local kubeconfig:

```bash

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

```

3. Install a network plugin (e.g., Flannel):

```bash

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

```

4. Verify the cluster status:

```bash

kubectl get nodes

```

Step 5: Join Worker Nodes

1. On the master node, get the join command:

```bash

kubeadm token create --print-join-command

```

- Copy the command output.

2. On each worker node, run the join command:

```bash

sudo kubeadm join <master-node-ip>:<port> --token <token> --discovery-token-ca-cert-hash sha256:<hash>

```

3. Verify that the nodes have joined the cluster:

```bash

kubectl get nodes

```

**Additional Information:**

* **K9s Integration**: The lab environment includes K9s, a terminal-based UI for managing Kubernetes clusters. K9s provides an interactive and user-friendly way to view and manage Kubernetes resources, helping users navigate and operate their clusters more effectively.
* **Application Monitoring**: Users can view and interact with running applications such as Gradle-based Spring applications and Python microservices. This allows users to monitor application status, logs, and resource utilization directly from the Kubernetes dashboard and through K9s.
* **Resource Management**: Users can practice managing Kubernetes resources like pods, deployments, and services. This includes scaling applications, updating configurations, and troubleshooting issues within the cluster.
* **Namespace Management**: The lab setup supports the use of multiple namespaces, allowing users to segregate applications and resources for better organization and management.
* **Ingress and Networking**: Users can experiment with configuring Ingress controllers and network policies to manage external access to services and secure communication between services within the cluster.
* **Persistent Storage**: The lab environment includes configurations for persistent storage, enabling users to test StatefulSets and storage classes, and manage data persistence for stateful applications.

**Helm**

**Definition and Introduction:**

* **Helm** is a package manager for Kubernetes, which simplifies the deployment and management of applications on Kubernetes. Helm uses "charts," which are packages of pre-configured Kubernetes resources.

**Lab Installation:**

* Helm is installed in lab environment, allowing users to deploy and manage Kubernetes applications more efficiently using Helm charts, and experiment with package management and versioning.

**Helm Installation and Setup Guide**

**Helm Installation Steps:**

**Prerequisites:**

you will need:

- A Kubernetes cluster set up and running.

- `kubectl` configured to interact with your Kubernetes cluster.

- An Ubuntu 20.04 server (this guide can be adapted for other distributions).

Step 1: Install Helm

1. Download Helm:

```bash

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

```

2. Verify Helm installation:

```bash

helm version

```

- The output should display the version of Helm installed.

Step 2: Configure Helm

1. Add a Helm repository (e.g., stable):

```bash

helm repo add stable https://charts.helm.sh/stable

```

- This command adds the stable repository to Helm, which contains a variety of pre-configured Helm charts.

2. Update Helm repositories:

```bash

helm repo update

```

- This command updates the local cache of Helm charts.

Step 3: Deploying an Application using Helm

1. Search for a Helm chart:

```bash

helm search repo nginx

```

- This command searches for Helm charts related to `nginx` in the configured repositories.

2. Install a Helm chart (e.g., NGINX):

```bash

helm install my-nginx stable/nginx-ingress

```

- This command installs the `nginx-ingress` chart from the stable repository and names the release `my-nginx`.

3. Verify the installation:

```bash

helm ls

```

- This command lists all the Helm releases installed in the cluster.

Step 4: Managing Helm Releases

1. Upgrade a Helm release:

```bash

helm upgrade my-nginx stable/nginx-ingress

```

- This command upgrades the `my-nginx` release to the latest version of the `nginx-ingress` chart.

2. Rollback a Helm release:

```bash

helm rollback my-nginx 1

```

- This command rolls back the `my-nginx` release to revision 1.

3. Uninstall a Helm release:

```bash

helm uninstall my-nginx

```

- This command uninstalls the `my-nginx` release from the cluster.

Step 5: Creating Your Own Helm Chart

1. Create a new Helm chart:

```bash

helm create my-chart

```

- This command creates a new Helm chart named `my-chart` with a default directory structure.

2. Modify the Helm chart:

- Edit the `values.yaml` file and templates in the `my-chart` directory to customize your application.

3. Deploy your custom Helm chart:

```bash

helm install my-app ./my-chart

```

- This command installs your custom Helm chart named `my-chart` and names the release `my-app`.

Step 6: Helm in the Lab Environment

- Experimentation: Users can use Helm to deploy various applications and experiment with different configurations and versions.

- Package Management: Helm simplifies managing complex applications and their dependencies through Helm charts.

- Versioning: Helm provides easy rollbacks and upgrades, making it ideal for managing application versions in a development environment.

**Jenkins**  
Jenkins is an open-source automation server that facilitates continuous integration and continuous delivery (CI/CD) in software development. It automates tasks related to building, testing, and deploying code, allowing developers to integrate changes more frequently and efficiently. Jenkins supports a wide range of plugins, enabling it to integrate with various tools and technologies. It provides a user-friendly web interface and can be extended to suit diverse project requirements, making it a popular choice for DevOps teams.

**Jenkin Cluster**

|  |  |
| --- | --- |
| **Jenkins URL** | http://10.10.30.77:9090 |
| **Username** | your name |
| **Password** | provided separately to each user |

The Jenkins cluster is set up to ensure efficient handling of CI/CD tasks by distributing the load across multiple nodes. Here are the details of the services and their corresponding IP addresses within the cluster:

|  |  |
| --- | --- |
| **Service** | **IP Address** |
| Jenkins | 10.10.30.77 |
| Jenkins Slave Node | 10.10.30.70 |
| PE | 10.10.30.78 |

Jenkins CI/CD Pipeline Documentation link:

**Lab Jenkins Environment:**

The lab environment is designed to provide a realistic and isolated setting for learning and experimenting with Jenkins and CI/CD pipelines. Below are the components and configurations of the lab environment:

**1. Virtualization Platform:**

- Platform: Proxmox VE

- Version: 7.4-3

- Virtual machines are configured to simulate a production-like environment.

**2. Network Configuration:**

- Subnet: \_10.10.30.0/24\_

- Gateway: \_10.10.30.1

- DNS Server: \_10.10.30.2

- Each service has a static IP address assigned within this subnet.

**3. Storage:**

- Persistent storage is configured for Jenkins and slave nodes.

- Type: Local storage (provided by company server)

- Capacity: 192GB

**4. Security:**

- Firewall rules are set up to restrict access to the Jenkins server and slave nodes.

- SSH access is configured for secure management of nodes.

- SSH Key: configured as per [SSH Configuration]

**5. Backup and Recovery:**

- Regular backups of Jenkins configurations and job data are scheduled.

- Backup location: /var/lib/jenkins/jenkins\_backup/

- Recovery procedures are documented and tested.

**Installation of Jenkins:**

Jenkins Installation Guide

Overview:

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. To ensure you have the latest fixes and features, use the project-maintained packages to install Jenkins.

Prerequisites:

To follow this tutorial, you will need:

- An Ubuntu 20.04 server set up with a sudo non-root user and a firewall.

Step 1: Add the Jenkins Repository

1. Add the repository key to your system:

```bash

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

```

- This command downloads the Jenkins GPG key and saves it to your system.

2. Append the Debian package repository address to the server’s sources list:

```bash

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

```

- This command adds the Jenkins repository to your system's package sources list.

3. Update the package list to use the new repository:

```bash

sudo apt-get update

```

- This command updates the package list to include packages from the newly added Jenkins repository.

Step 2: Install Jenkins

1. Install Jenkins and its dependencies:

```bash

sudo apt-get install jenkins

```

- This command installs Jenkins along with any necessary dependencies.

Step 3: Start Jenkins

1. Start Jenkins using systemctl:

```bash

sudo systemctl start jenkins.service

```

- This command starts the Jenkins service.

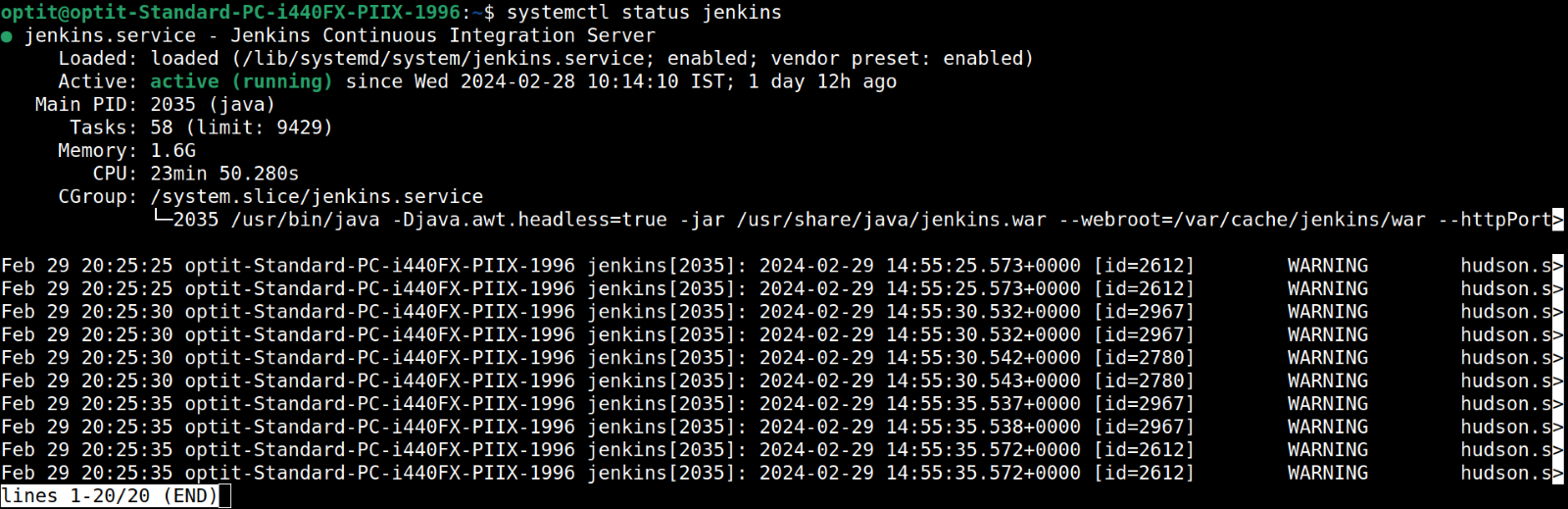
2. Verify that Jenkins started successfully:

```bash

sudo systemctl status jenkins

```

- This command checks the status of the Jenkins service. The output should indicate that the service is active and configured to start at boot.

  
 Step 3: Opening the Firewall

To ensure Jenkins can be accessed, you need to configure your firewall settings.

1. Open port 8080 for Jenkins:

```bash

sudo ufw allow 8080

```

2. If the firewall is inactive, enable it and allow OpenSSH:

```bash

sudo ufw allow OpenSSH

sudo ufw enable

```

3. Verify the firewall status and confirm the new rules:

```bash

sudo ufw status

```

Step 4: Setting Up Jenkins

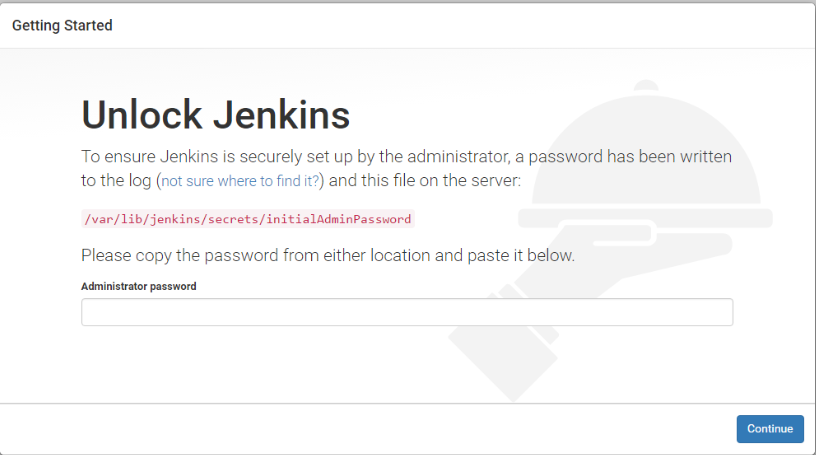
To complete the Jenkins setup, follow these steps:

1. Access Jenkins using your server's domain name or IP address on port 8080:

```plaintext

http://10.10.30.65:8080

```

2. You should see the "Unlock Jenkins" screen, which provides the location of the initial admin password. Use the password from the specified file to proceed with the setup.  


In the terminal window, use the cat command to display the password:

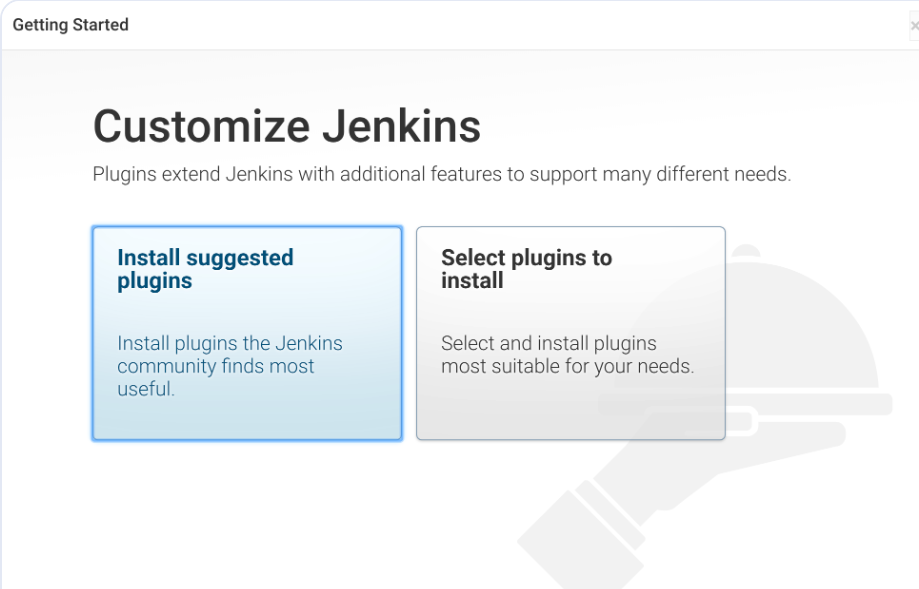
```

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

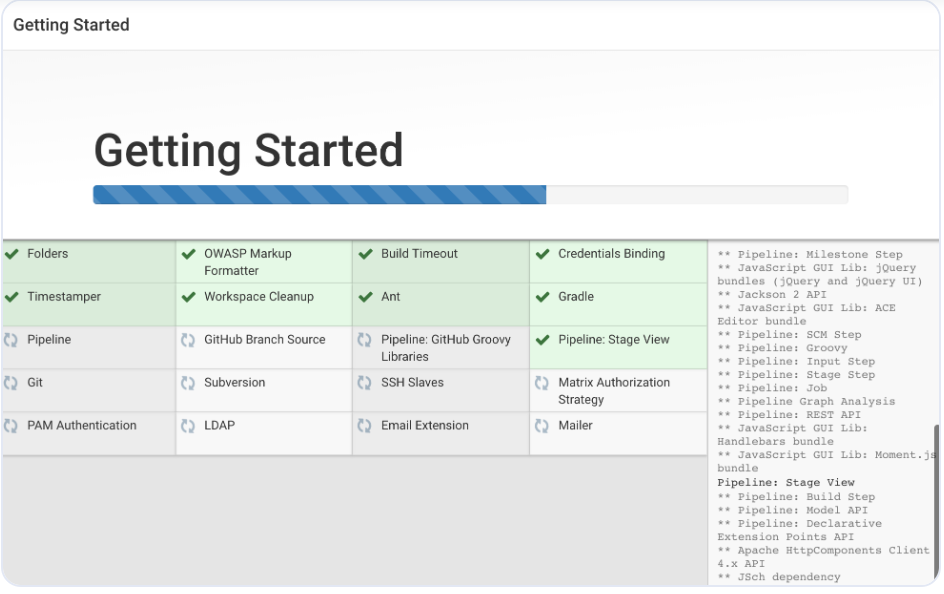
```

Copy the 32-character alphanumeric password from the terminal and paste it into the Administrator password field, then click Continue.

The next screen presents the option of installing suggested plugins or selecting specific plugins:

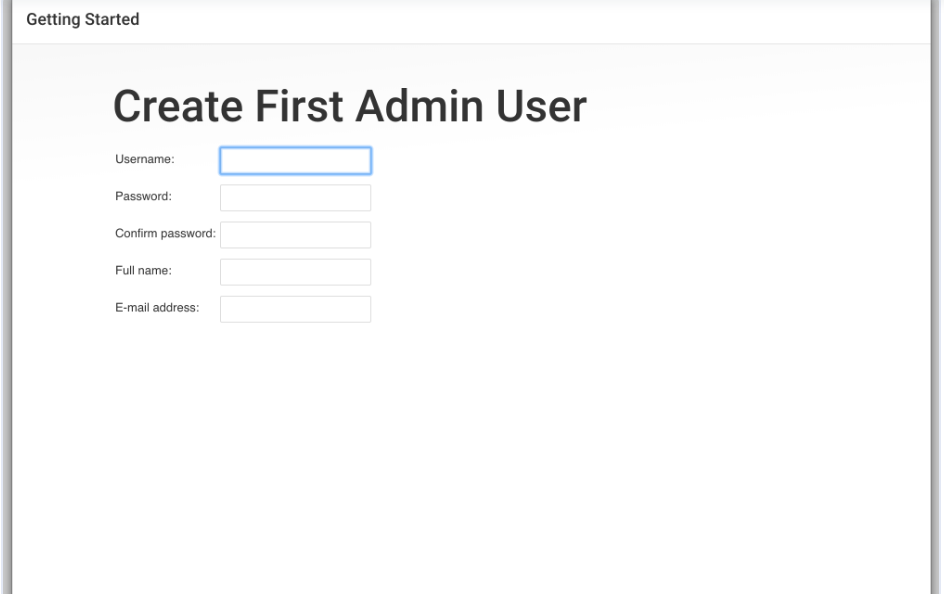


We’ll click the Install suggested plugins option which will immediately begin the installation process

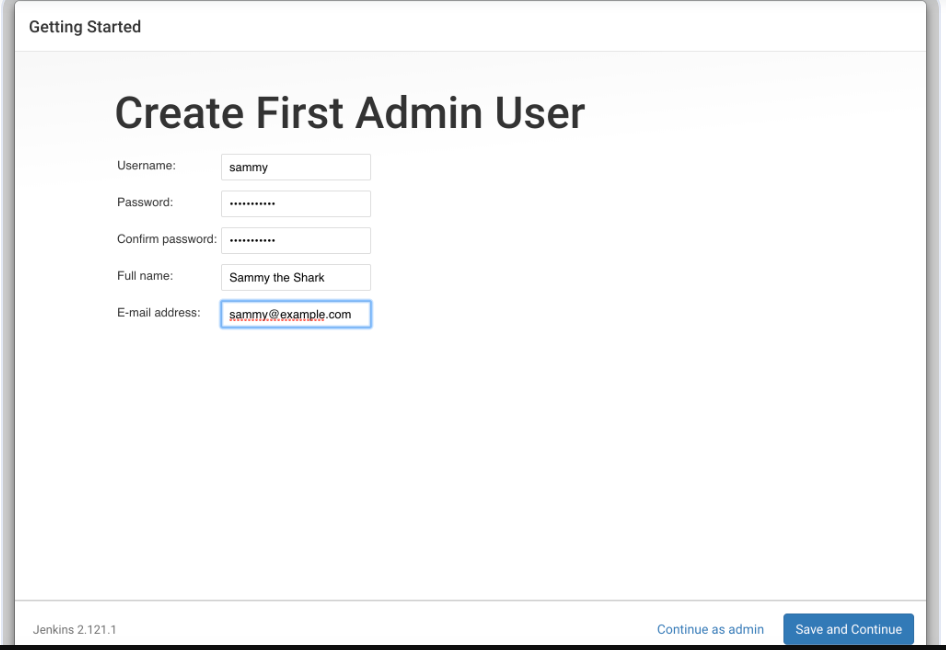


When the installation is complete, you’ll be prompted to set up the first administrative user. It’s possible to skip this step and continue as admin using the initial password from above, but we’ll take a moment to create the user

Note: The default Jenkins server is NOT encrypted, so the data submitted with this form is not protected. Refer to How to Configure Jenkins with SSL Using an Nginx Reverse Proxy on Ubuntu 22.04 to protect user credentials and information about builds that are transmitted via the web interface

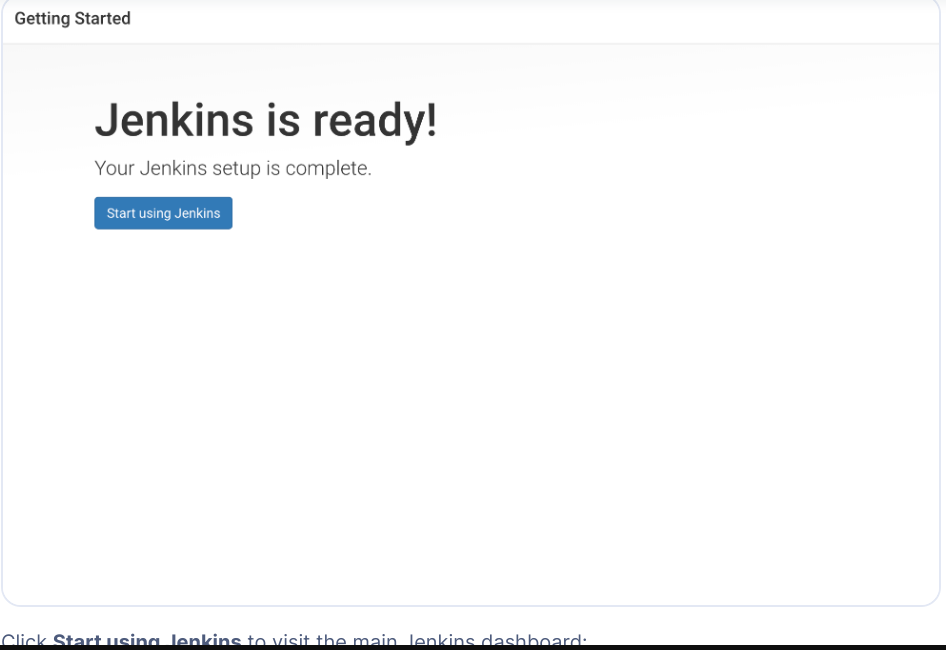


Enter the name and password for your user



You’ll receive an Instance Configuration page that will ask you to confirm the preferred URL for your Jenkins instance. Confirm either the domain name for your server or your server’s IP address

After confirming the appropriate information, click Save and Finish. You’ll receive a confirmation page confirming that “Jenkins is Ready!”



Click Start using Jenkins to visit the main Jenkins dashboard.

**7. Freestyle Project**

[Optitlab\_Assignment1\_Freestyleproject](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment1_Freestyleproject/)[Optitlab\_Assignment2\_Freestyleproject\_Parameter](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment2_Freestyleproject_Parameter/)[Optitlab\_Assignment3\_Freestyleproject\_gitclone\_java](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment3_Freestyleproject_gitclone_java/)

**Maven Project**

[Optitlab\_Assignment4\_Mavenproject\_gitclone\_java](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment4_Mavenproject_gitclone_java/)

**Pipeline**

[Optitlab\_Assignment5\_Declarativepipelinejob\_basic\_introduction](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment5_Declarativepipelinejob_basic_introduction/)

[Optitlab\_Assignment6\_scriptedpipelinejob\_basic\_introduction](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment6_scriptedpipelinejob_basic_introduction/)

**Multi branch pipeline**

[Optitlab\_Assignment14\_Multi\_branch\_pipeline\_introduction](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment14_Multi_branch_pipeline_introduction/)

**Folder**

[Optitlab\_Assignment13\_Folder\_basic\_introduction](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment13_Folder_basic_introduction/)

**Multi-Configuration Project**

[Optitlab\_Assignment12\_Multi\_Configuration\_project\_basic\_introduction](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment12_Multi_Configuration_project_basic_introduction/)

**Organization Folder**

**CI and CD Pipelines :**

[Optitlab\_Assignment15\_CI](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment15_CI/)

[Optitlab\_Assignment16\_CD](http://10.10.30.77:9090/job/OptitLabAssignments/job/Optitlab_Assignment16_CD/)

**Troubleshooting:**

If you are facing issue while accessing above URL’s, please contact Platform engineering team.

**Conclusion:**

Thanks for Using OptIT lab and once your job has been executed and tested using OptIT Lab, please ensure to delete it and refrain from creating unnecessary jobs, but only those that you have created.

**Additional Information**

-

**Argo CD**

**Definition and Introduction:**

* **Argo CD** is a declarative, GitOps continuous delivery tool for Kubernetes. It synchronizes application definitions from Git repositories with the current state of applications running in Kubernetes clusters. Argo CD provides a user-friendly web interface to manage and visualize application deployments, making it easier to maintain consistency and traceability across environments.

**Lab Installation:**

* Argo CD is configured in our lab environment to enable continuous delivery and automated deployment processes via GitOps. Users can experiment with automated deployment strategies and version control, gaining hands-on experience with modern CI/CD practices.

**Access Details:**

|  |  |
| --- | --- |
| **URL:** | http://argocd.10-10-30-87.sslip.io |
| **Username:** | admin |
| **Password:** | Optit@123 |

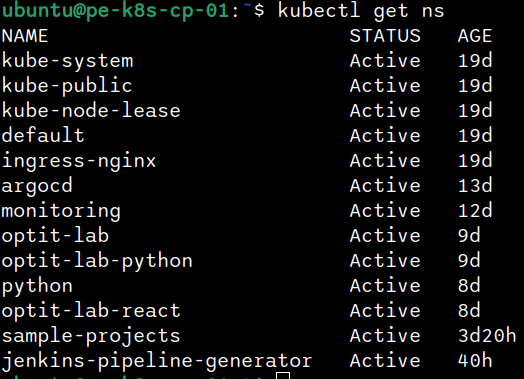
Users can log in to Argo CD using the provided credentials and explore various features, such as syncing applications, managing deployments, and monitoring application states. This setup allows users to learn and experiment with GitOps principles and continuous delivery workflows.

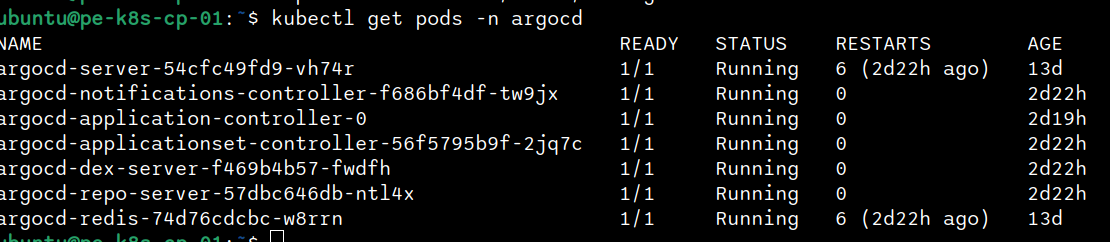
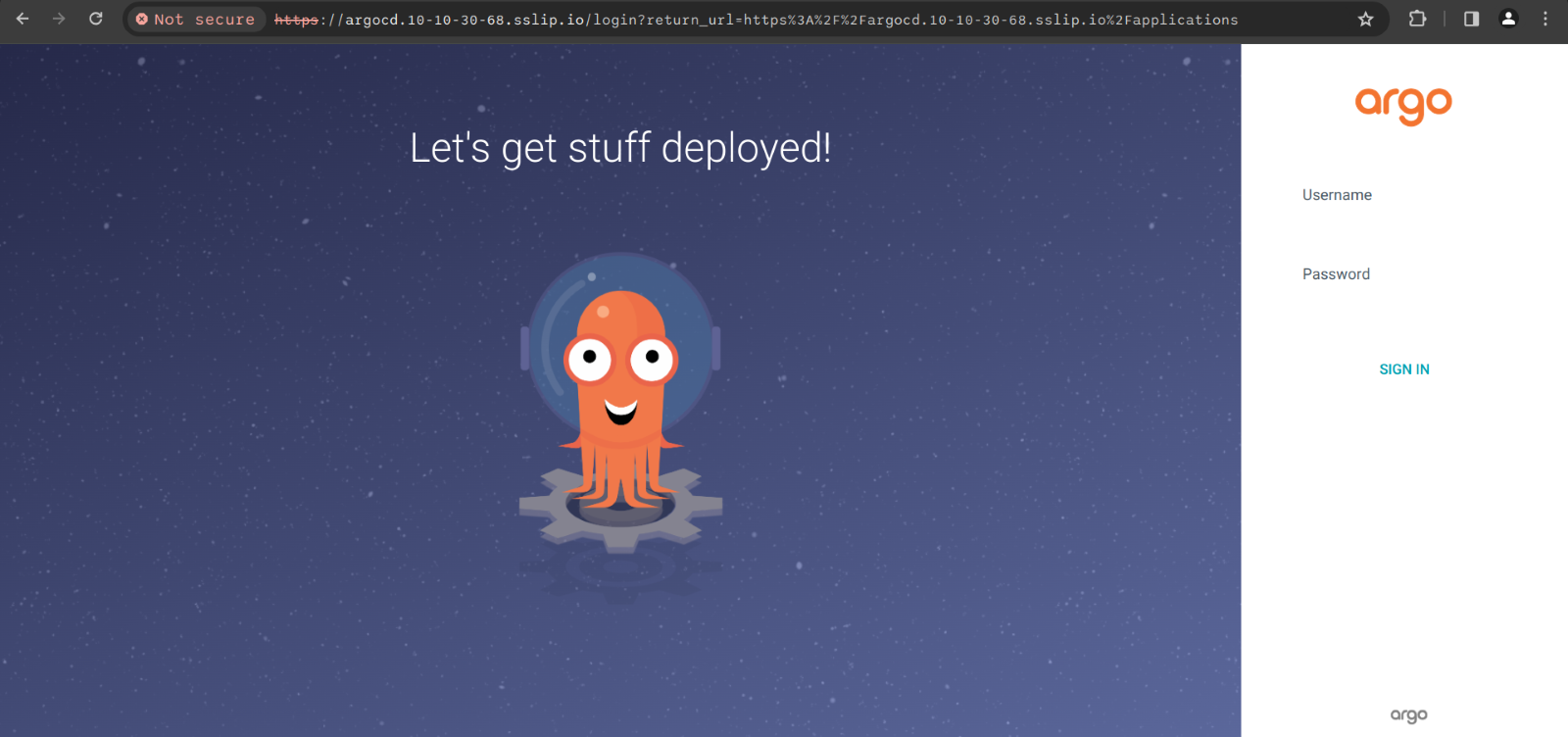
**Installation of Argocd through helm :**  
  
First we need to verify weather helm is installed our cluster

‘’’  
  
helm repo add argo <https://argoproj.github.io/argo-helm>

helm install argocd argo/argo-cd --version 6.6.0 --set "server.ingress.enabled=true" --set "server.ingress.ingressClassName=nginx"  
  
As our server don’t have public ip we are accessing through the ingerss nginx.

helm upgrade --install argocd argo/argo-cd --version 6.6.0 --set "server.ingress.enabled=true" --set "server.ingress.ingressClassName=nginx" --set "server.ingress.hostname=argocd.10-10-30-68.nip.io

After installing the above commands, it created namespace argocd   


under that name space it created its resource,  
  
  
  
As we don’t have public ip to our server we are trying to access through ingress nginx   
  
by this url <http://argocd.10-10-30-68.sslip.io/> you will be seeing the login page ,  
  
  
  
for credentials you can refer lab manual document.

**example Gradle application CI/CD**

**Tools Used:**

Jenkins

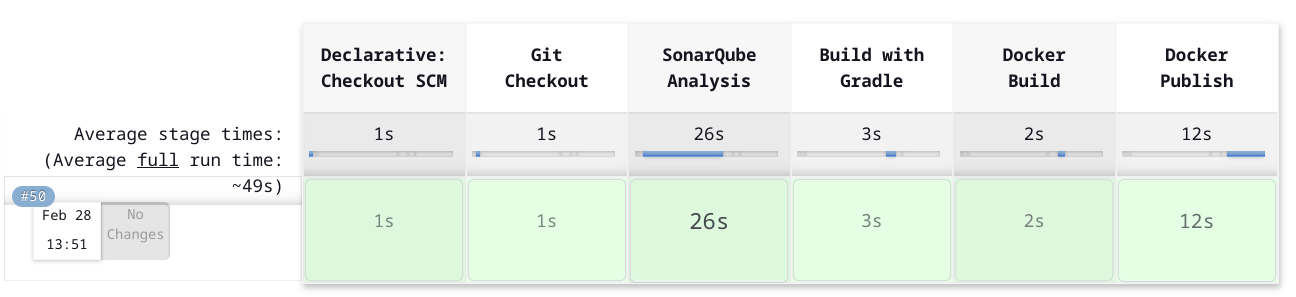
SonarQube

Docker Hub

Gradle

**Pipeline Overview:**

The Continuous Integration (CI) pipeline for the Gradle application automates the process of building, testing, analysing code quality, and pushing the Docker image to Docker Hub whenever changes are made to the source code repository.



Pipeline Steps:

Checkout: Jenkins pipeline checks out the latest version of the source code from the configured repository.

SonarQube Analysis: The source code is analyzed by SonarQube to identify code smells, bugs, security vulnerabilities, and maintainability issues.

Build: Gradle is used to build the application. Dependencies are resolved, and the application is compiled.

Test: Automated tests, including unit tests and integration tests, are executed to ensure the code quality and functionality.

Dockerization: The application is containerized using Docker. A Docker image is built with the necessary dependencies and configurations.

Push to Docker Hub: Once the Docker image is successfully built, it is pushed to Docker Hub, making it available for deployment.

**Continuous Delivery (CD) Pipeline:**

**Tools Used:**

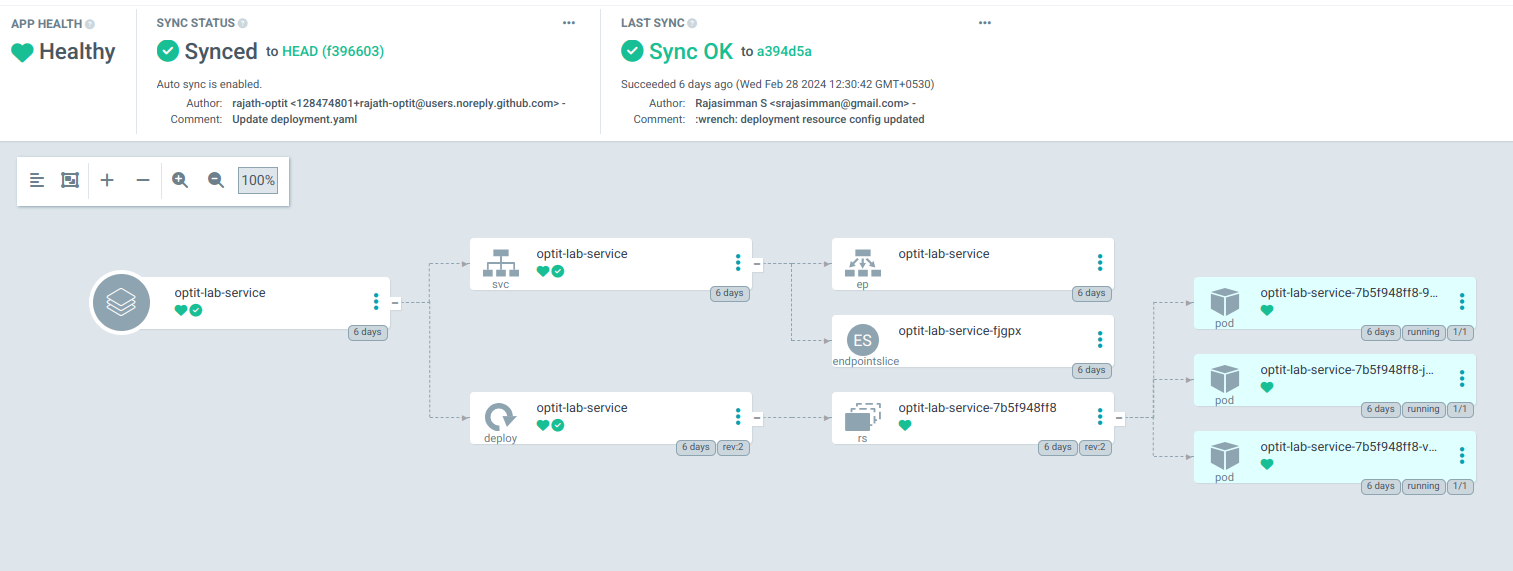
ArgoCD

Kubernetes

Kubernetes Manifest Files

**Pipeline Overview:**

The Continuous Delivery (CD) pipeline automates the deployment process of the Gradle application using Kubernetes and ArgoCD. It ensures that the latest version of the application is deployed seamlessly to the target environment.



**Pipeline Steps:**

ArgoCD Sync: ArgoCD monitors the Git repository containing Kubernetes manifest files for changes.

Manifest Update: When changes are detected in the repository, ArgoCD pulls the latest manifests.

Deployment: ArgoCD applies the updated manifests to the Kubernetes cluster, initiating the deployment process.

Rollout Strategy: Kubernetes orchestrates the deployment according to the defined rollout strategy, ensuring zero-downtime deployments and rollback capabilities if necessary.

Health Checks: Kubernetes monitors the health of the deployed application instances and automatically restarts or replaces any unhealthy instances.

Integration of CI and CD Pipelines:

The CI pipeline triggers the CD pipeline upon successful completion of the Docker image build and push process. This ensures that the latest version of the application is deployed to the target environment immediately after it passes all tests, analysis, and is packaged into a Docker image.

**Benefits:**

Automation: Both CI and CD pipelines automate the build, test, code analysis, and deployment processes, reducing manual errors and improving efficiency.

Consistency: By using pipelines, the build, test, code analysis, and deployment processes are consistent and repeatable, regardless of the environment.

Visibility: Jenkins, SonarQube, and ArgoCD provide visibility into the pipeline execution, allowing teams to track the progress of builds, analysis, and deployments.

Scalability: The pipelines are scalable and can accommodate future growth and changes in the application and infrastructure.

**Conclusion:**

The CI/CD pipeline for the Gradle application streamlines the software delivery process, from code changes to deployment, ensuring rapid and reliable delivery of new features and updates. By leveraging Jenkins, SonarQube, Docker, Kubernetes, and ArgoCD, teams can achieve continuous integration, delivery, and deployment with ease and confidence while maintaining high code quality.

**Additional Information**

-

**Grafana**

**Definition and Introduction:**

* **Grafana** is an open-source analytics and monitoring platform that allows users to visualize, query, and alert metrics and logs from multiple data sources. It provides customizable dashboards that help in understanding the performance and health of applications and infrastructure. Grafana supports a wide range of data sources, including Prometheus, InfluxDB, and Elasticsearch, among others. Its powerful visualization capabilities and extensibility make it a preferred tool for monitoring and observability in various environments.

**Lab Installation:**

* Grafana is installed and configured in our lab environment, providing a comprehensive platform for users to experiment with data visualization and monitoring. Users can create and customize dashboards to gain insights into application performance and infrastructure health.

**Access Details:**

|  |  |
| --- | --- |
| **URL:** | [http://grafana.10-10-30-87.sslip.io](http://grafana.10-10-30-87.sslip.io/) |
| **Username:** | (Your email address) |
| Password: | Optit@123 |

Users can log in to Grafana using the provided credentials and explore various features, such as creating dashboards, setting up alerts, and connecting different data sources to visualize metrics and logs.

Installation of Monitoring Stack through Helm

First, we need to create a namespace called `monitoring`:

```bash

kubectl create namespace monitoring

```

Next, add the Prometheus Community Helm repository:

```bash

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

```

Update the Helm repository to ensure we have the latest charts:

```bash

helm repo update prometheus-community

```

Before installing, we need to create and customize a `values.yml` file. Here is an example of the `values.yml` file:

```yaml

prometheus:

prometheusSpec:

podMonitorSelectorNilUsesHelmValues: false

serviceMonitorSelectorNilUsesHelmValues: false

grafana:

enabled: true

defaultDashboardsTimezone: Asia/Kolkata

env:

GF\_SERVER\_ROOT\_URL: http://grafana.10-10-30-68.sslip.io/

ingress:

enabled: true

hosts:

- grafana.10-10-30-68.sslip.io

windowsMonitoring:

enabled: false

kubeScheduler:

enabled: false

kubeControllerManager:

enabled: false

additionalScrapeConfigs:

- job\_name: 'optit-lab-service'

kubernetes\_sd\_configs:

- role: pod

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_pod\_container\_port\_name]

regex: 'http'

action: keep

```

Save this file as `values.yml`.

Now, install the Prometheus stack using Helm with the customized `values.yml` file:

```bash

helm install prometheus prometheus-community/kube-prometheus-stack -n monitoring -f values.yml

```

Since we do not have a public IP for the server, we will access Grafana through the domain `http://grafana.10-10-30-68.sslip.io/`. This should load the Grafana login page.

For credentials, please refer to the lab manual.

**Additional Information**

-

**EFK Stack (Elasticsearch, Fluentd, Kibana)**

**Definition and Introduction:**

* **EFK Stack** consists of Elasticsearch, Fluentd, and Kibana. It is used for logging and monitoring:
  + **Elasticsearch**: A search and analytics engine for storing and indexing logs.
  + **Fluentd**: A data collector that helps unify the logging layer.
  + **Kibana**: A visualization tool for Elasticsearch that provides a web interface for searching and visualizing logs.

**Lab Installation:**

* The EFK stack is deployed in our lab environment to enable centralized logging and monitoring for our applications. Users can experiment with log collection, indexing, and visualization, gaining insights into application performance and issues.

**Lab step of setting up EFK**  
step-by-step guide to set up the EFK Stack (Elasticsearch, Fluentd, Kibana) on your Kubernetes cluster. This guide uses `nip.io` for DNS-based access to services.

Prerequisites

- Kubernetes cluster running (Master: 10.10.30.87, Nodes: 10.10.30.88, 10.10.30.89)

- kubectl configured to interact with the cluster

- Helm installed on your machine

Step 1: Install Elasticsearch

Elasticsearch will store and index the logs. We'll use Helm to deploy Elasticsearch.

1. Add the Elastic Helm repository:

```sh

helm repo add elastic https://helm.elastic.co

helm repo update

```

2. Install Elasticsearch:

```sh

helm install elasticsearch elastic/elasticsearch --namespace kube-logging --create-namespace

```

3. Verify the installation:

```sh

kubectl get pods --namespace=kube-logging -l app=elasticsearch

```

Step 2: Install Fluentd

Fluentd collects logs from your applications and forwards them to Elasticsearch.

1. Create a Fluentd configuration file:

Save the following as `fluentd-config.yaml`:

```yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: fluentd-config

namespace: kube-logging

data:

fluent.conf: |

<source>

@type tail

path /var/log/containers/.log

pos\_file /var/log/es-containers.log.pos

time\_format %Y-%m-%dT%H:%M:%S.%N%:z

tag kubernetes.

format json

read\_from\_head true

</source>

<filter kubernetes.>

@type kubernetes\_metadata

</filter>

<match >

@type elasticsearch

host elasticsearch-master

port 9200

logstash\_format true

include\_tag\_key true

type\_name access\_log

flush\_interval 1s

</match>

```

2. Create the ConfigMap in your cluster:

```sh

kubectl apply -f fluentd-config.yaml

```

3. Deploy Fluentd using a DaemonSet:

Save the following as `fluentd-daemonset.yaml`:

```yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: fluentd-config

namespace: kube-logging

data:

fluent.conf: |

<source>

@type tail

path /var/log/containers/.log

pos\_file /var/log/es-containers.log.pos

time\_format %Y-%m-%dT%H:%M:%S.%N%:z

tag kubernetes.

format json

read\_from\_head true

</source>

<filter kubernetes.>

@type kubernetes\_metadata

</filter>

<match >

@type elasticsearch

host elasticsearch-master

port 9200

logstash\_format true

include\_tag\_key true

type\_name access\_log

flush\_interval 1s

</match>

```

4. Create the DaemonSet in your cluster:

```sh

kubectl apply -f fluentd-daemonset.yaml

```

5. Verify the installation:

```sh

kubectl get pods --namespace=kube-logging -l name=fluentd

```

Step 3: Install Kibana

Kibana provides the web interface for visualizing the logs stored in Elasticsearch.

1. Install Kibana:

```sh

helm install kibana elastic/kibana --namespace kube-logging

```

2. Verify the installation:

```sh

kubectl get pods --namespace=kube-logging -l app=kibana

```

3. Access Kibana:

- Create an Ingress resource to expose Kibana. Save the following as `kibana-ingress.yaml`:

```yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: kibana

namespace: kube-logging

spec:

rules:

- host: kibana.10-10-30-87.nip.io

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: kibana

port:

number: 5601

```

- Apply the Ingress resource:

```sh

kubectl apply -f kibana-ingress.yaml

```

4. Verify Kibana access:

- Check the Ingress setup:

```sh

kubectl get ingress --namespace=kube-logging

```

- Update your DNS configuration (if necessary):

Ensure that `kibana.10-10-30-87.nip.io` resolves to your Ingress controller’s IP.

Additional Step: Elasticsearch Ingress

1. Create an Ingress resource for Elasticsearch: Save the following as `elasticsearch-ingress.yaml`:

```yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: elasticsearch

namespace: kube-logging

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: elasticsearch.10-10-30-87.nip.io

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: elasticsearch-master

port:

number: 9200

```

2. Apply the Ingress resource:

```sh

kubectl apply -f elasticsearch-ingress.yaml

```

3. Verify Elasticsearch access:

- Check the Ingress setup:

```sh

kubectl get ingress --namespace=kube-logging

```

- Update your DNS configuration (if necessary):

Ensure that `elasticsearch.10-10-30-87.nip.io` resolves to your Ingress controller’s IP.

**Summary**

- **Elasticsearch** is installed to store and index logs.

- **Fluentd** is installed as a DaemonSet to collect logs from each node and forward them to Elasticsearch.

- **Kibana** is installed to provide a web interface for visualizing logs.

- **Ingress resources** are configured for Elasticsearch and Kibana using `nip.io` for DNS-based access.

**Additional Information**

**-**