



# **LFS261: DevOps and SRE Fundamentals - Implementing Continuous Delivery**

**- V7.11.2025 -**

---

# Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Lab 1. Setting up the Learning Environment</b>	<b>5</b>
Create a Google Cloud Platform Account	5
Creating a Project to Contain Your VM	7
Navigating to VM Instances on Google Cloud	9
Create a VM Instance	11
Connecting to Your Instance	15
Opening the Firewall for Our Google Cloud VM	16
Creating a Firewall Rule	17
Adding the Firewall Rule to Our VM	19
Verify the Rule Was Applied	22
Installing and Verifying Docker on Your Linux VM	22
Verifying and Installing Git	24
Summary	24
<b>Lab 2. Getting Started with Docker</b>	<b>25</b>
Validate the Setup	25
Launching Your First Container	25
Listing Containers	26
Learning About Images	27
Default Run Options	27
Launching and Connecting to Web Applications	28
Troubleshooting Containers	28
Exercise: Stop, Remove and Clean Up	29
Setting Up Nextcloud	30
Setting up Portainer with Advanced run Options	33
Cleanup	34
Summary	34
<b>Lab 3. Revision Control with Git</b>	<b>35</b>
Configuring Git Client	35
Creating a Sample Application	36
Basics of Revision Control	38
Using Docker in Development	39
Branching and Merging	41
Adding a New Feature by Branching Out	42
Merging with Commit History	42
Working with Remotes	44
Resolving Conflicts	45
Creating a Conflict	46

---

Manually Resolving a Conflict	47
Learning to Undo	48
Revert vs Reset	49
Working with GitHub	51
Summary	51
<b>Lab 4. Setting Up Continuous Integration with Jenkins</b>	<b>52</b>
Install Docker Compose	52
Set Up Jenkins with Docker	52
Fork the Voting App	60
Configuring a Maven Build Job	61
Adding Unit Test and Packaging Jobs	68
Configuring Build Triggers	72
Creating a Job Pipeline	77
Set Up the Pipeline View	79
Exercise: Create a Pipeline for the Node.js Application	82
Summary	83
<b>Lab 4S. Creating A Pipeline for the NodeJS Result Application (Solution)</b>	<b>84</b>
<b>Lab 4.1. Setting Up Continuous Integration with Jenkins (Additional Topics)</b>	<b>92</b>
Optional: Integrating GitHub with Jenkins	92
Setting Up Build Triggers Using GitHub Webhooks	92
Optional: Adding Jenkins Status Badges to GitHub	96
Optional: Configuring Job Status with Commit Messages	97
<b>Lab 5. Enforcing Workflows, Pipeline as Code</b>	<b>100</b>
Enforcing Workflow with Branch Policies	100
Testing Your New Branch Protection Rules	104
Code Reviews with Pull Requests	104
Writing Pipeline as Code	108
Jenkinsfile for a Java Application	109
Multi-Branch Pipeline Project	110
Configuring Conditional Execution of Stages	122
Exercise: Writing a Jenkinsfile for the Result NodeJS Application	125
Summary	125
<b>Lab 5S. Solution to the Jenkinsfile Exercise</b>	<b>126</b>
Writing A Jenkinsfile for a NodeJS Application	126
Common Issues	128
<b>Lab 5.1. Pipeline as Code (Additional Topics)</b>	<b>129</b>
Integrating Slack with Jenkins	129
Sending Notifications from a Pipeline Job	134
<b>Lab 6. Using Docker with Jenkins Pipelines</b>	<b>137</b>
Install Docker Plugins	137
Refactoring the Worker Pipeline to Build with Docker Agent	138

---

Validation	140
Summary	140
<b>Lab 7. Packaging with Docker</b>	<b>141</b>
Registering with Docker Hub	141
Test Building Docker Image for the worker Application	142
Automated Image Build with Dockerfile	145
Adding Docker Build and Publish Stages to Jenkinsfile	146
Adding Docker Hub Credentials	148
Jenkinsfile Per Stage Agent Configurations	150
Exercise: Vote and Result Application Dockerfiles	153
Result application - inside example-vote-app/result	154
Vote application - inside example-vote-app/vote	156
<b>Lab 8. Deploy to Dev with Docker Compose</b>	<b>160</b>
Create a Simple Docker Compose Spec	160
Add Networks and Volumes	162
Fixing Postgres DB Issue	164
Additional Docker Compose Commands to Explore	165
Finishing Up	166
Integrate Docker Compose with Dockerfiles	166
Add New Port Mapping	168
Exercise: Create a Consolidated Pipeline for Instavote Stack	169
Add "Deploy to Dev" with Docker Compose	170
<b>Lab 9. Continuous Testing</b>	<b>172</b>
Code Coverage with Jacoco	172
Adding Static Code Analysis with SonarQube	179
Set Up Sonar Cloud as Sonarqube Server	179
Integrate SonarQube with Jenkins	188
Scanning Code and Reporting to SonarQube	194
Configuring a Quality Gate	196
Adding Sonarqube Scanner Stage to Jenkinsfile	203
Adding Integration Tests	208
E2E Tests	210
The .env File	210
Setting Up E2E	211
Integrating E2E Tests with Jenkins	212
Summary	217
<b>Lab 10. Running Containers at Scale with Kubernetes</b>	<b>218</b>
Set Up Kubernetes Cluster with GCP	218
Connecting to Your Cluster	225
Connect to Your Cluster Via RUN IN CLOUD SHELL	225
Manual Connection to Your Cluster	227

<b>Deploy Frontend Vote App with Kubernetes</b>	<b>230</b>
Pod Operations	234
Scalability	235
Scaling Manually	235
Autoscaling	236
Availability	238
<b>Publishing and Load Balancing with Services</b>	<b>238</b>
Opening Firewall Rules	241
Service Discovery	244
Summary	248
<b>Lab 11. Continuous Deployment with Argo CD</b>	<b>248</b>
Connect to the Cluster	249
Deploy ArgoCD to the Cluster	250
Setting Up Our Git Repository	256
Deploying Our Vote Front End	258
Set Argo CD to Continuous Deployment	269
Creating the deployment Pipeline	270
Triggering Deployment Pipeline	274
Setting Argo CD to Continuously Deploy	274
Auto Deploy Changes	278
Lab Summary	279



## Lab 1. Setting up the Learning Environment

By the end of this lab exercise, you should be able to:

- Create a Google Cloud Platform (GCP) account
- Configure a VM on GCP
- Connect to a VM on GCP
- Install Docker on Linux VM
- Verify that git is installed

First, sign up for Docker Hub at <https://hub.docker.com/>.

### Create a Google Cloud Platform Account

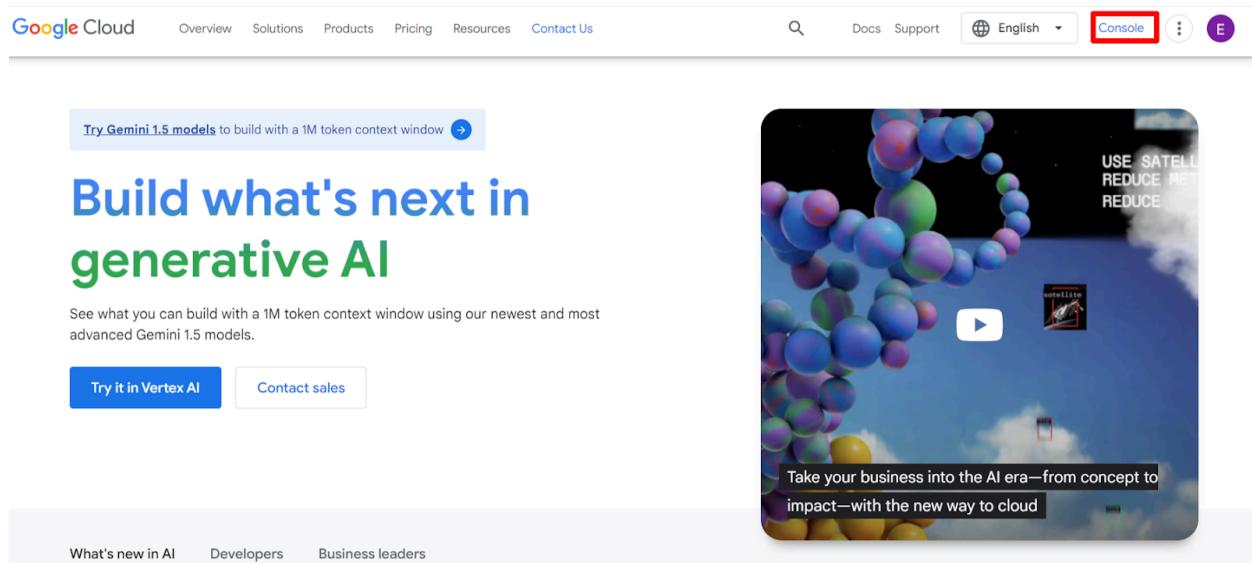
The only prerequisite for this task is a Google account. If you do not have one, go to [google.com](https://google.com), select **Sign in** and follow the directions.

To get started, visit [cloud.google.com](https://cloud.google.com) and follow the simple steps to create a Google Cloud account. You can begin with a free account and sign up for free credits, which are valid for a limited time.

While creating an account, choose the account type **individual** and provide your address and payment details to complete the signup.

Once you create a Google Cloud Platform account, you will get Console access. You can revisit your account by going to [cloud.google.com](https://cloud.google.com) at any time.

Once logged in, you should see your Google Cloud Console, as pictured below:

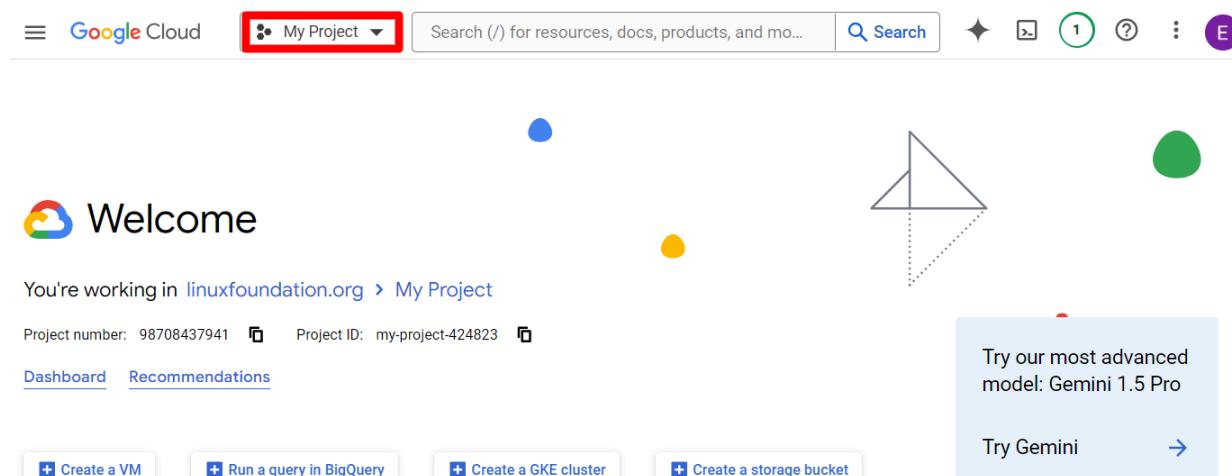


Alternatively, you can visit the console directly with this link: <https://console.cloud.google.com>.

## Creating a Project to Contain Your VM

According to [Google Cloud documentation](#), a "project in Google Cloud Platform (GCP) organizes all of your Google Cloud resources, including Cloud Storage data, Compute instances, monitoring and logging data, and App Engine instances. Projects also include associated permissions for these resources".

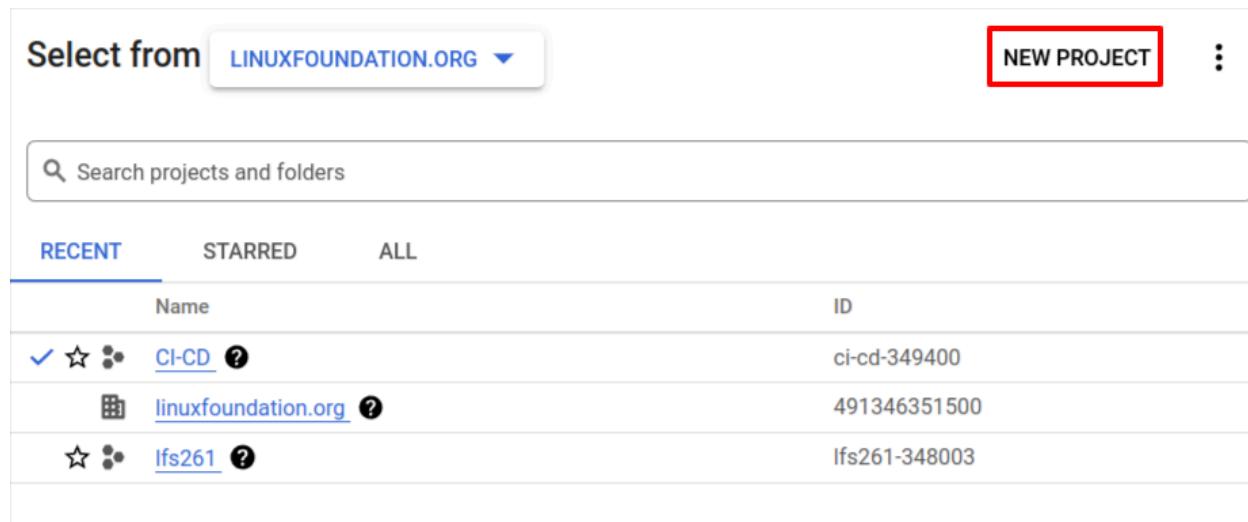
A project provides a namespace to isolate resources for that project. To set up a new project, select the project dropdown menu. If it is your first time, you will see **My First Project** in the box. Otherwise, you will see the last project that you were in. To create a new project, click inside the box as illustrated below.



The screenshot shows the Google Cloud Welcome screen. At the top, there's a navigation bar with the Google Cloud logo, a dropdown menu for 'My Project' (highlighted with a red box), a search bar, and various icons. Below the header, the word 'Welcome' is displayed next to a colorful cloud icon. A message says 'You're working in linuxfoundation.org > My Project'. It shows project details: 'Project number: 98708437941' and 'Project ID: my-project-424823'. Navigation links for 'Dashboard' and 'Recommendations' are present. A 'Quick access' section includes buttons for 'Create a VM', 'Run a query in BigQuery', 'Create a GKE cluster', and 'Create a storage bucket'. A callout box in the top right corner promotes 'Gemini 1.5 Pro' with the text 'Try our most advanced model: Gemini 1.5 Pro' and a 'Try Gemini' button.

Quick access

Select **NEW PROJECT**.



The screenshot shows a 'Select from' dropdown menu. It has a dropdown arrow pointing down and the text 'LINUXFOUNDATION.ORG'. To the right of the dropdown is a 'NEW PROJECT' button, which is highlighted with a red box. Below the dropdown is a search bar with the placeholder 'Search projects and folders'. Underneath the search bar are three tabs: 'RECENT', 'STARRED', and 'ALL'. The 'RECENT' tab is selected. A table follows, showing three recent projects:

Name	ID
CI-CD ?	ci-cd-349400
linuxfoundation.org ?	491346351500
Ifs261 ?	Ifs261-348003

Name the project **CI-CD** and click **CREATE**.

## New Project

**⚠ You have 8 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)**

[MANAGE QUOTAS](#)

**Project name \*** CI-CD

**Project ID \*** psychic-nuance-349400 [C](#)

Project ID can have lowercase letters, digits, or hyphens. It must start with a lowercase letter and end with a letter or number.

**Location \*** [No organization](#) [BROWSE](#)

Parent organization or folder

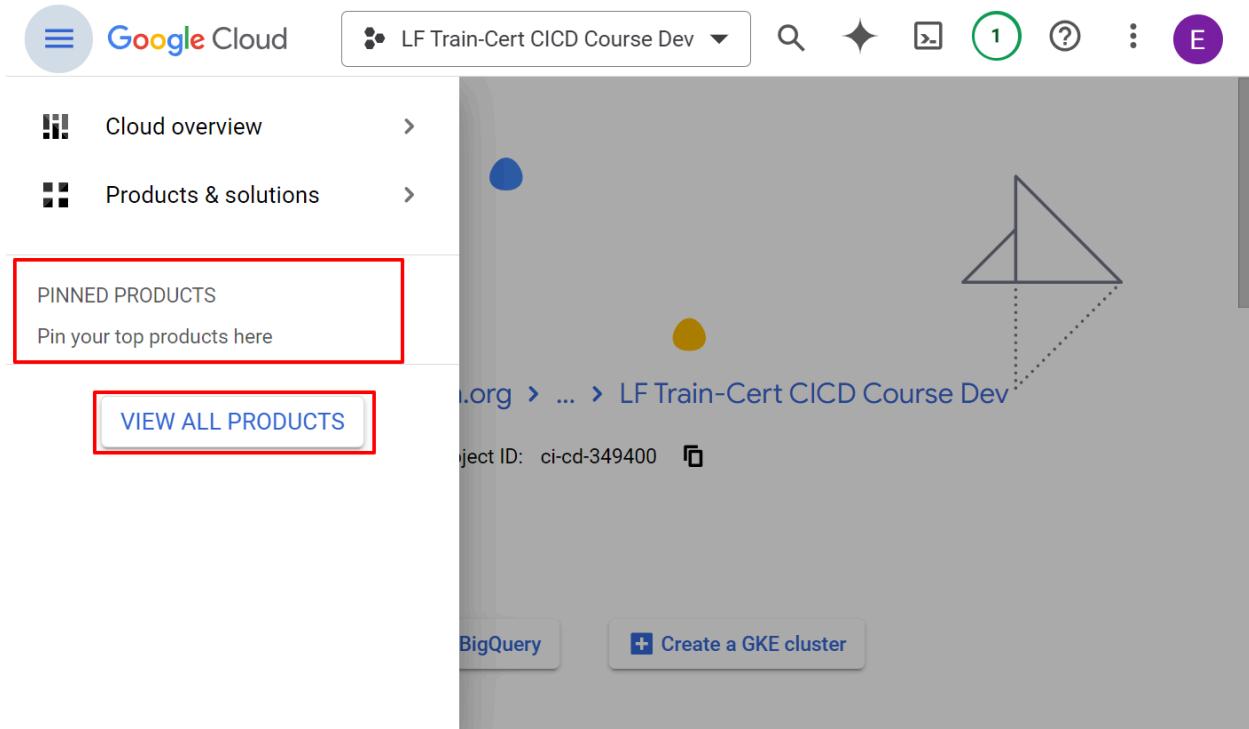
[CREATE](#) [CANCEL](#)

## Navigating to VM Instances on Google Cloud

Open the menu in your Google Cloud account by clicking the menu in the top left:

The screenshot shows the Google Cloud interface. At the top, there's a blue header bar with a menu icon (three horizontal lines), the text "Google Cloud", a dropdown for "ci-cd", and a search bar with the placeholder "Search Products, resources, docs (/)". Below the header is a decorative background with abstract shapes like triangles and circles. In the center, the word "Welcome" is displayed next to the Google Cloud logo. Below "Welcome", it says "You're working in linuxfoundation.org > CI-CD". It also shows "Project number: 349571583085" and "Project ID: ci-cd-349400". At the bottom of the main area, there are links for "Dashboard" and "Recommendations". Along the bottom edge, there are four buttons: "Create a VM", "Run a query in BigQuery", "Create a GKE cluster", and "Create a storage bucket".

The flyout menu has two sections. Look for **Compute Engine** in the **PINNED** or **VIEW ALL PRODUCTS** sections. If **Compute Engine** is not in the **PINNED** section, click **VIEW ALL PRODUCTS**:



Find **Compute Engine** and then click the pin icon.

The screenshot shows the Google Cloud Platform dashboard. On the left, there's a sidebar with 'Products & solutions' and 'Categories' sections. Under 'Compute', the 'Compute' section is highlighted with a red box. It contains a brief description: 'Run scalable virtual machines and containers'. Below this, a table lists various compute engines:

Name	Description
<a href="#">Compute Engine</a>	VMs, GPUs, TPUs, disks
<a href="#">Kubernetes Engine</a>	Managed Kubernetes / containers
<a href="#">VMware Engine</a>	VMware as a service
<a href="#">Anthos</a>	Enterprise hybrid multi-cloud platform
<a href="#">Batch</a>	Jobs as a service

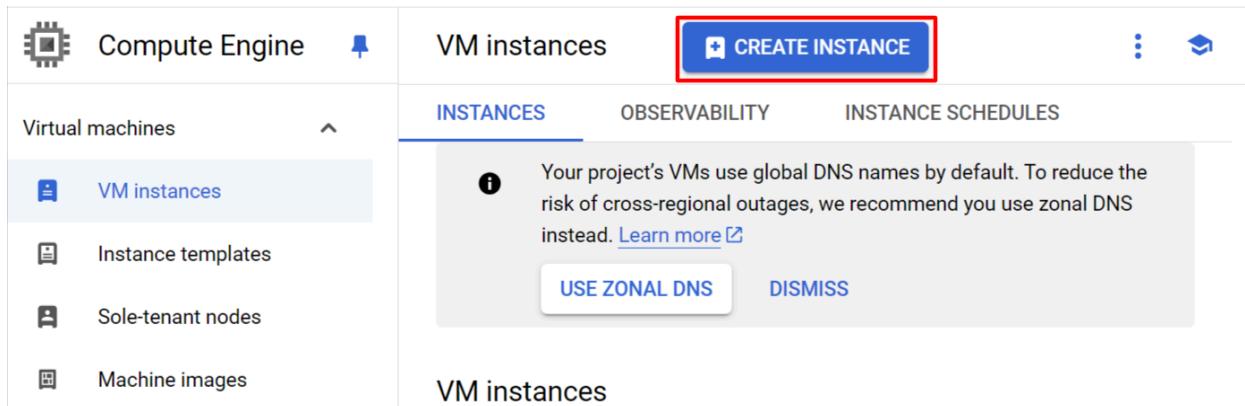
**Navigate to Menu > Compute Engine > VM instances:**

This screenshot shows the navigation path to the VM instances page. The 'Compute Engine' menu item is highlighted with a red box. The 'VM instances' option under the 'Compute Engine' menu is also highlighted with a red box. The right side of the screen shows the 'VM instances' page with a table of existing VMs and various management options.

This will bring you to the VM instances page where you can create your VMs and see any VMs that have already been created.

## Creating a VM Instance

To bring up the UI for creating a new VM instance, select **CREATE INSTANCE** at the top of the page:



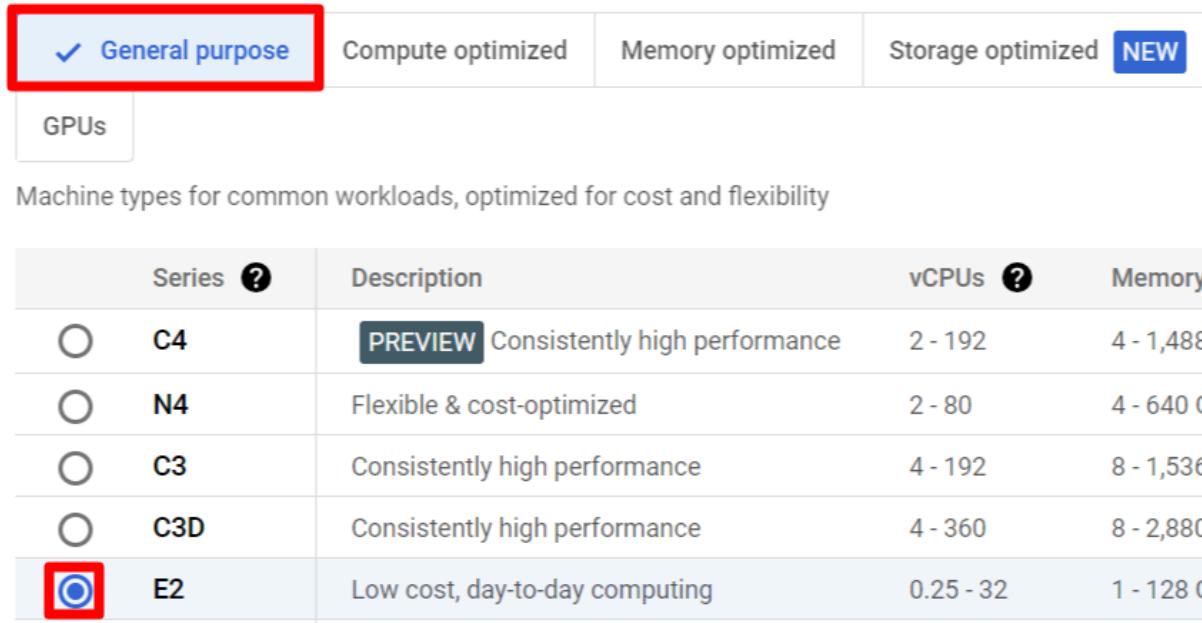
You can give the instance a name that is meaningful to you. In this example, the VM instance is named **ci**.

Choose the **Region** that is closest to you. In this example, we have chosen **us-west1 (Oregon)**.

This screenshot shows the 'Create VM instance' form. The 'Name' field is filled with 'ci' and has a red box around it. The 'Labels' section has a '+ ADD LABELS' button. The 'Region' dropdown is set to 'us-west1 (Oregon)' with a red box around it, and a note below says 'Region is permanent'. The 'Zone' dropdown is set to 'us-west1-b' with a red box around it, and a note below says 'Zone is permanent'.

Scroll to **Machine Configuration**. Select **E2** under the **General purpose** tab.

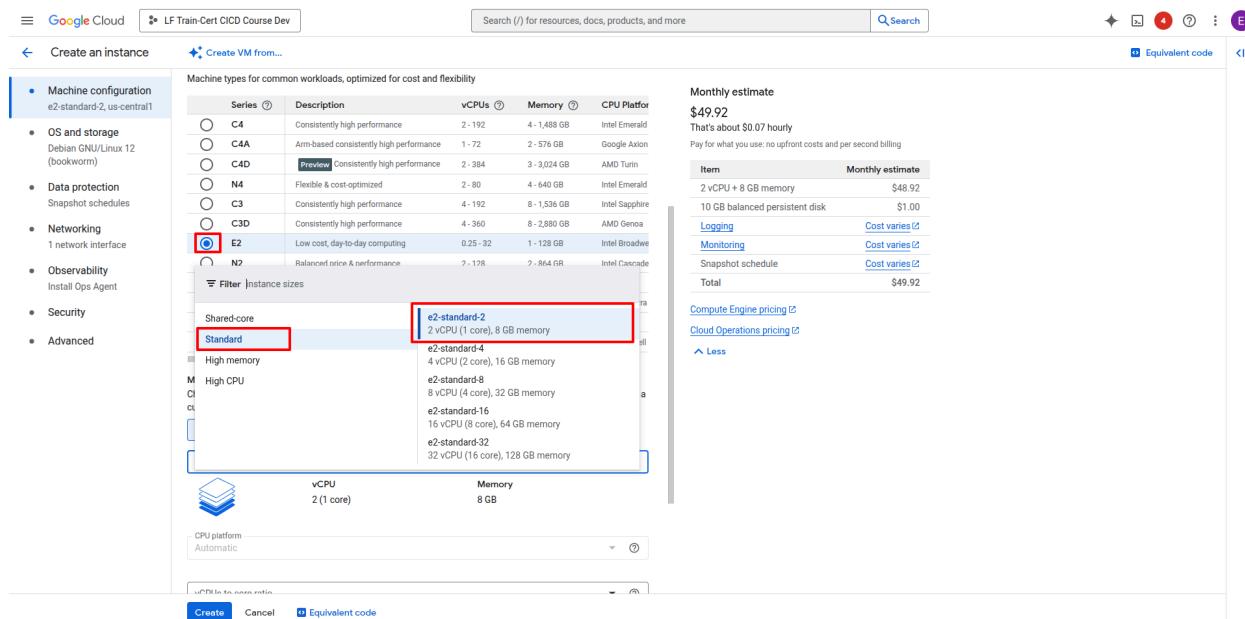
## Machine configuration



The screenshot shows the Google Cloud Machine Configuration interface. The 'General purpose' tab is highlighted with a red box. Below it, there's a section for GPUs. A list of machine types is shown:

Series	Description	vCPUs	Memory
C4	Consistently high performance	2 - 192	4 - 1,488 GB
N4	Flexible & cost-optimized	2 - 80	4 - 640 GB
C3	Consistently high performance	4 - 192	8 - 1,536 GB
C3D	Consistently high performance	4 - 360	8 - 2,880 GB
<b>E2</b>	Low cost, day-to-day computing	0.25 - 32	1 - 128 GB

Scroll to **Machine type**. For machine type, choose **e2-standard-2 (2 vCPU, 8 GB memory)**.



The screenshot shows the Google Cloud 'Create an instance' page. On the left, a sidebar lists categories like Machine configuration, OS and storage, Data protection, Networking, Observability, Security, and Advanced. Under Machine configuration, 'e2-standard-2, us-central1' is selected. The main area shows the 'Machine types for common workloads, optimized for cost and flexibility' table. The 'E2' row is highlighted with a red box. In the 'Shared-core' section, the 'Standard' button is also highlighted with a red box. The 'e2-standard-2' row is also highlighted with a red box. To the right, a 'Monthly estimate' table shows costs for various services. At the bottom, there are fields for 'vCPU' (2), 'Memory' (8 GB), and 'CPU platform' (Automatic). Buttons for 'Create' and 'Cancel' are at the bottom.

Click **OS and storage** on the right.

Scroll to **Boot disk** and select **CHANGE**:

This will bring up a form that allows you to choose the operating system you want to run. As we decided to use Ubuntu 24.04 for lab exercises, choose **Operating system > Ubuntu**. Choose the **x86** version of **Ubuntu 24.04**. Enter **30** for the **Size**. Then, click **Select**.

Boot disk X

Select an image or snapshot to create a boot disk; or attach an existing disk. Can't find what you're looking for? Explore hundreds of VM solutions in [Marketplace](#)

[Public images](#) [Custom images](#) [Snapshots](#) [Archive Snapshots](#) [Existing Disks](#)

Operating system [Ubuntu](#)

Version \* [Ubuntu 24.04 LTS](#)

x86/64, amd64 noble image built on 2025-06-24

Boot disk type \* [Balanced persistent disk](#)

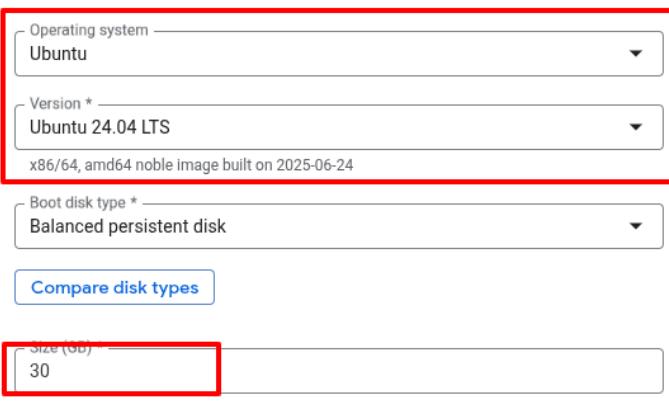
[Compare disk types](#)

Size (GB) [30](#)

Provision between 10 and 65536 GB

[Show advanced configuration](#)

[Select](#) [Cancel](#)



Scroll to the bottom of the VM creation form and click **CREATE**.

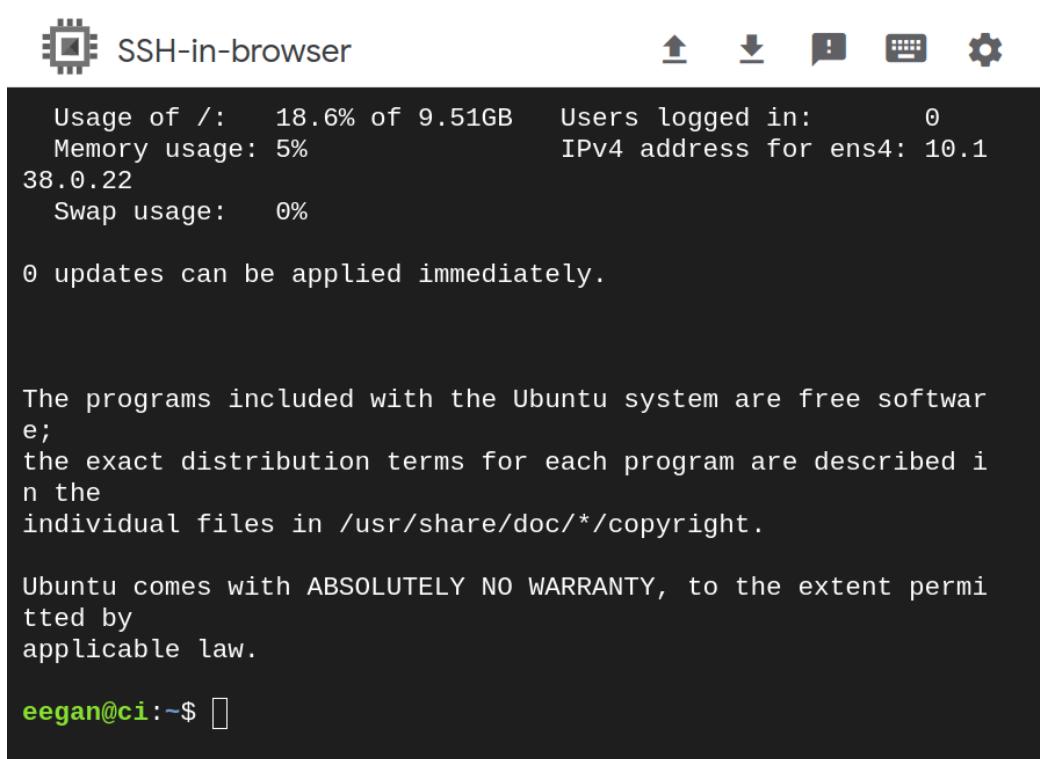
The screenshot shows the Google Cloud VM creation interface. On the left, a sidebar lists configuration sections: Machine configuration, OS and storage (selected), Data protection, Networking, Observability, Security, and Advanced. The main panel displays the 'Operating system and storage' configuration. It includes fields for Name (ci), Type (New balanced persistent disk), Size (20 GB), Snapshot schedule (default-schedule-1), License type (Free), and Image (Ubuntu 24.04 LTS). Below this are sections for Additional disks, Container (with a Deploy container button), and a monthly estimate table. The 'Create' button at the bottom left is highlighted with a red box.

## Connecting to Your VM Instance

You will be taken back to the **VM instances** page where you will see your newly created VM. To connect to your VM, click **SSH**.

<input type="checkbox"/>	Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input checked="" type="checkbox"/>		ci-01	us-west1-b			10.138.0.2 (nic0)	35.197.102.152 (nic0)	<b>SSH</b>

A terminal window will pop up.



The screenshot shows a terminal window titled "SSH-in-browser". At the top, there are icons for file operations (upload, download, copy, paste, cut, settings) and a gear icon. The terminal displays the following text:

```
Usage of /: 18.6% of 9.51GB  Users logged in: 0
Memory usage: 5%          IPv4 address for ens4: 10.1
38.0.22
Swap usage: 0%

0 updates can be applied immediately.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

eegan@ci:~$ 
```

You are inside a Linux Ubuntu machine. This is a terminal that you can use to interact with your VM on GCP.

If you would like to connect to a Google Cloud VM instance from a terminal on your own computer, you will need to follow the directions here:

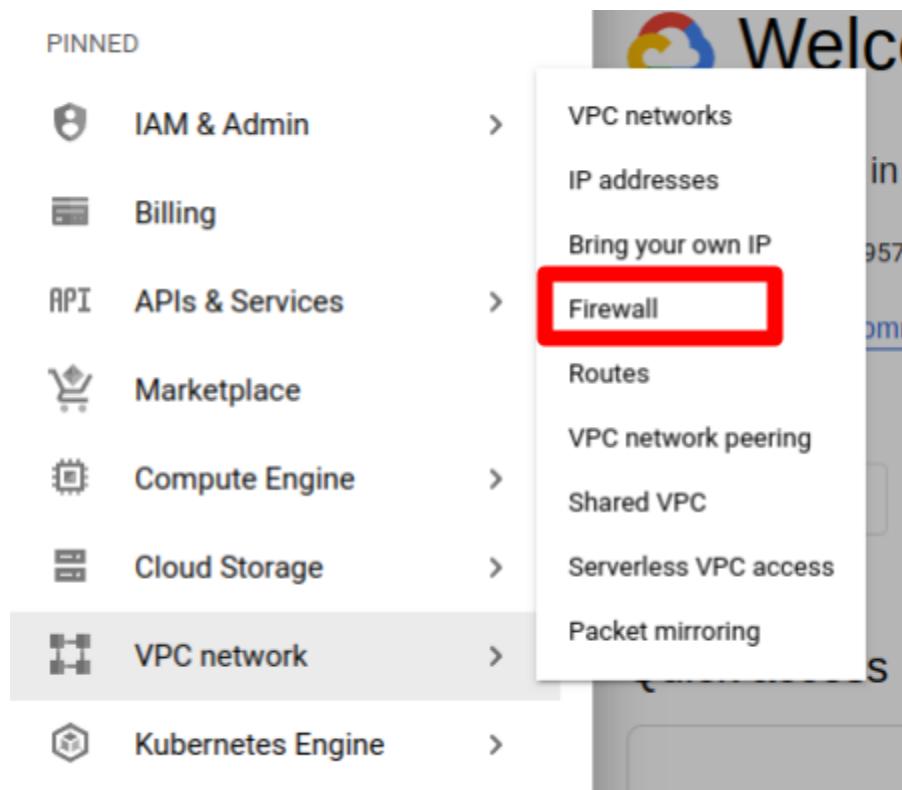
<https://cloud.google.com/compute/docs/instances/connecting-advanced#before-you-begin>

## Opening the Firewall for Our Google Cloud VM

When learning new concepts, it is helpful to configure our learning environments in a way that allows us to focus on the topic at hand. In our case, we don't want to worry about whether the issues we are running into are network-related or due to the tools we are learning. This is not a networking course. As such, we will open our firewall completely to rule out any firewall issues. This is only for learning purposes. **This is NOT something you would do in production, as it is NOT secure.** However, it will help you troubleshoot a CI/CD pipeline.

## Creating a Firewall Rule

Under the **VPC network** go to **Firewall**.



In networking, it is common to create **rules** that are then **applied** to specific computers or VMs. We will create a **Firewall rule** and then apply it to our VM instance.

A screenshot of the 'Firewall' rules list page. At the top, there are tabs for 'Firewall' (selected) and 'REFRESH'. Below the tabs is a large red button labeled '+ CREATE FIREWALL RULE'. To the left of the main content area is a decorative graphic of overlapping shapes in red, blue, and white. The main content area features a heading 'Get real-time analytics with Network Intelligence Center' and a list of benefits: 'Use Network Intelligence Center for comprehensive visibility and control', '✓ Visualize your network resources', '✓ Diagnose and prevent connectivity issues', '✓ View packet loss and latency metrics', and '✓ Keep your firewall rules strict and efficient'. At the bottom of the page is a blue link 'GO TO NETWORK INTELLIGENCE CENTER'.

Refer to the following screenshot:

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. [Learn more](#)

The screenshot shows the configuration page for creating a new firewall rule. Key fields and their values are as follows:

- Name \***: open
- Description**: Only use this firewall rule for learning. Do not use this firewall rule for production environments.
- Logs**: Off
- Network \***: default
- Priority \***: 1000
- Direction of traffic**: Ingress
- Action on match**: Allow
- Targets**: Specified target tags
- Target tags \***: open
- Source filter**: IPv4 ranges
- Source IPv4 ranges \***: 0.0.0.0/0
- Second source filter**: None
- Protocols and ports**: Allow all

At the bottom, there is a **CREATE** button highlighted with a red box, and a **CANCEL** button.

Name the firewall rule **open** and give it a description. In our case, we have reminded ourselves NOT to use this rule in production.

Then, under **Target tag**, enter **open**. This is how you will add this rule to your VM instance. You need to remember this or look it up to add it to your VM. To make it easy, give this firewall rule the same tag as the name of the firewall rule.

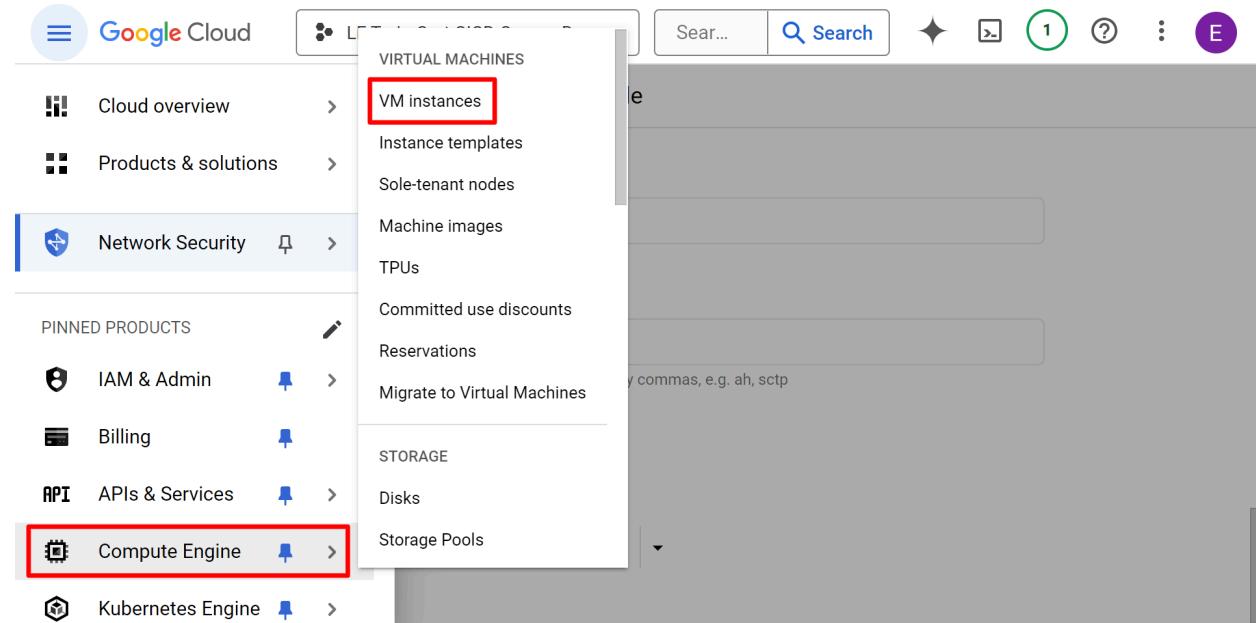
Under **Source IPv4 Ranges** we will enter **0.0.0.0/0**. **0.0.0.0/0** is networking for *all IP addresses*. This means that your VM can be connected to from *any computer*. As noted, this is highly insecure, but it is good for learning.

Under **Protocols and ports** select **Allow all** and then click **CREATE**.

### Adding the Firewall Rule to Our VM

Now we have a firewall rule that will allow *all* traffic through. A rule is of no use unless it is applied to something. In this case, we want to apply it to our VM.

First go to **Compute Engine > VM instances**.



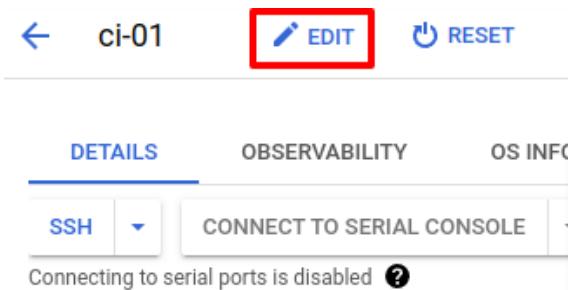
---

Click on your VM's name.



	Status	Name ↑	Zone
<input type="checkbox"/>	✓	ci-01	us-west1-b

Click **EDIT** at the top of the screen.



← ci-01 EDIT RESET

DETAILS OBSERVABILITY OS INFO

SSH ▾ CONNECT TO SERIAL CONSOLE

Connecting to serial ports is disabled ?

Scroll down to **Networking** if it isn't visible.

## Networking

### Network performance configuration

Network interface card is permanent  
Network Interface card

—

▼

#### Network bandwidth

You must stop the VM instance to edit Network bandwidth.

Increase total egress bandwidth

Maximum outbound network bandwidth: 2Gbps

### Network interfaces ?

Network interface is permanent

default default (10.138.0.0/20)

▼

ADD NETWORK INTERFACE

#### Firewalls

Allow HTTP traffic

Allow HTTPS traffic

#### Network tags

Network tags

?

CANCEL

Enter **open** in the **Network tags** box. Remember, this is what we tagged our firewall rule earlier.

## Verify the Rule Was Applied

To verify that the instance is associated with the **open** firewall rule, return to **Google Cloud > VPC Network > Firewall**. Click on the **open** firewall rule.

<input type="checkbox"/>	Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs
<input type="checkbox"/>	<a href="#">default-allow-http</a>	Ingress	http-server	IP ranges: 0.0.0.0/0	all	Allow	1000	<a href="#">default</a>	Off
<input type="checkbox"/>	<a href="#">default-allow-https</a>	Ingress	https-server	IP ranges: 0.0.0.0/0	tcp:443	Allow	1000	<a href="#">default</a>	Off
<input type="checkbox"/>	<a href="#">open</a>	Ingress	open	IP ranges: 0.0.0.0/0	all	Allow	1000	<a href="#">default</a>	Off
<input type="checkbox"/>	<a href="#">open-all</a>	Ingress	open-all-tag	IP ranges: 0.0.0.0/0	all	Allow	1000	<a href="#">default</a>	Off

Then scroll down to **Applicable to Instances**.

Applicable to instances			
<i>The following table shows only the VM instances that you have permission to view.</i>			
<input type="checkbox"/> Filter Filter by instance name, project or subnetwork			
Name	Subnetwork	Internal IP ranges	External IP ranges
<a href="#">ci</a>	<a href="#">default</a>	10.138.0.22	34.168.137.243

You should see your VM listed.

## Installing and Verifying Docker on Your Linux VM

The following directions have been pulled directly from Docker's documentation. Run each command in the order in which they appear. The first command ensures that any existing Docker versions are removed, preventing conflicts with the new version we will install. If Docker is not installed, the output will inform you that Docker was not found. Either outcome is fine.

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

---

Now that you have removed or ensured that no Docker is present on the machine, we can proceed with the install.

```
$ sudo apt-get update

$ sudo apt-get install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /etc/apt/keyrings/docker.gpg

$ echo \

  "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list \
> /dev/null

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

Type **Y** when prompted.

The directions are found here: <https://docs.docker.com/engine/install/ubuntu/>.

After installation, validate Docker by running the following commands:

```
$ sudo docker version

Client:           Docker Engine - Community
Version:          28.0.4
...
Server: Docker Engine - Community
Engine:
Version:          28.0.4
...
```

Do a smoke test with:

```
$ sudo docker run hello-world
...
Hello from Docker!
```

---

```
This message shows that your installation appears to be working  
correctly.
```

....

This validates that Docker has successfully been installed.

Add your user to the `docker` group. This will make it more convenient to run commands, as you won't need to type `sudo` every time you run Docker.

**WARNING:** In a production environment, this has significant security implications that must be carefully weighed.

There is no one-size-fits-all answer as to whether you should add users to the `docker` group.

```
$ sudo usermod -aG docker $USER  
$ newgrp docker
```

You should now be able to run `docker` commands without typing `sudo` first.

## Verifying and Installing Git

Git comes pre-installed on most Linux distributions. To verify, run:

```
$ git version
```

If it is not installed, reference the official download and installation guides below to set up Git:

- [Git - Downloads](#)
- [Installing Git](#)

## Summary

You are now set up to get started with the hands-on lab exercises in the rest of the course.



## Lab 2. Getting Started with Docker

By the end of this lab exercise, you should be able to:

- Launch containers with Docker
- List common options used with the `docker container run` command
- Run web applications and access them with port mapping
- Manage container lifecycle and learn how to debug them
- Clean up

### Validate the Setup

Begin by checking the Docker version you have installed using the following command:

`docker version`

Validate further by running a smoke test:

`docker run hello-world`

This should launch a container successfully and show you a `hello world` message.

This smoke test validates the following:

- You have Docker client installed
- Docker daemon is up and running
- Docker client is correctly configured and authorized to talk to Docker daemon
- You have non-blocking internet connectivity
- You are able to pull an image from Docker registry and run a container with it

### Launching Your First Container

Before launching your first container, open a new terminal and run the following command to

---

analyze the events of the Docker daemon:

`docker system events`

When you run the above command, you may not see any output. Keep this window open, and it will start streaming events as you proceed to use the Docker CLI in another window.

Now, launch your first container with the following command:

`docker container run alpine ps`

where,

- `docker` is the command line client
- The `container run` command is used to launch a container. You can alternatively just use the `run` command here
- `alpine` is the image
- `ps` is the actual command which is run inside this container

Create a few more containers with the same image but with different commands as follows:

`docker container run alpine uptime`

`docker container run alpine uname -a`

`docker container run alpine free`

You can check the events in the window where you started running the `docker system events` command earlier. It shows what is happening on the Docker daemon side.

## Listing Containers

Check your last run container using the following commands:

`docker ps -l`

`docker ps -n 2`

`docker ps -a`

where,

- `-l`: last run container
- `-n xx`: last xx number of containers
- `-a`: all containers (even if they are in a ‘stopped’ state)

## Learning About Images

Containers are launched using images which are pulled from the registry. Observe the following command:

```
docker container run alpine ps
```

Here, the image name is `alpine`, which can actually be expanded to:

```
registry.docker.io/docker/alpine:latest
```

where the following are the fields:

- registry: `registry.docker.io`
- namespace: `docker`
- repo: `alpine`
- tag: `latest`

Use the following commands to list your images from your local machine and pull the `nginx` image from DockerHub. Examine the layers of an image with the `history` command:

```
docker image ls
```

```
docker image pull nginx
```

```
docker image history nginx
```

## Default Run Options

You have learned how to launch a container. However, this container is ephemeral and exits immediately after the command is run. To make the container persistent and to be able to interact with it, let's try adding a couple of new options:

```
docker run -it alpine bash
```

where,

- `-i`: (interactive) provides standard in to the process running inside the container
- `-t`: provides a pseudo-terminal in order to interact

You could further add the `--rm` options to ensure the container is deleted when stopped:

```
docker run --rm -it alpine sh
```

**Note: Use `^d` to come out of the shell opened within the container.**

Another useful option is `-d`, which will launch the process in a contained environment and

---

detach from it. This allows the container to keep running in the background:

```
docker container run -idt --name redis redis:alpine
```

## Launching and Connecting to Web Applications

So far, you have launched a few simple containers with one-off commands. It's time to learn how to launch a web application and connect to it.

You could launch a container with the `nginx` web server as follows. Observe the new `-p` option:

```
docker container run -idt -p 8080:80 nginx
```

where,

- `-p` allows you to define a port mapping
- 8080 is the host-side port
- 80 is the container-side port, the one on which the application is listening

Once you configure the port mapping, access that application on your browser by visiting `http://IPADDRESS:8080`.

**NOTE:** Replace `IPADDRESS` with the \*actual IP\* in case your Docker host is remote, or with \*localhost\* if using Docker Desktop.

With the command above, using the `-p` option, you explicitly define the host-side port. You can also have Docker pick the port automatically with the `-P` option:

```
docker container run -idt -P nginx
```

where,

- The host port is automatically chosen and incremented starting with 32768.
- The container port is automatically read from the image.

As usual, use `docker ps` commands to observe the port mapping.

## Troubleshooting Containers

To debug issues with the containers, which are nothing but processes running in an isolated environment, the following could be two important tasks:

- Checking process logs
- Being able to establish a connection inside the container (similar to ssh)

---

You will learn about both in this subsection.

To start examining logs, find your container ID or container name using the following command:

`docker ps`

You can use the command below to find logs for a container named `redis`:

`docker logs redis`

You have an option of replacing the `redis` name with the actual container ID (the first column in the output of the `docker ps` command).

You can follow the logs using the `-f` option. `docker exec` allows you to run a command inside a container:

`docker logs -f redis`

**Note: Use ^c to exit**

The `exec` command allows you to connect to the container and launch a shell inside it, similar to an ssh connection. It also allows running one-off commands:

`docker exec redis ps`

`docker exec -it redis sh`

With the `logs` and `exec` commands, you should be able to get started with essential debugging.

## Exercise: Stop, Remove and Clean Up

With the knowledge gained thus far, set up the following two apps with Docker:

- [Nextcloud](#)
- [Portainer](#)

Try launching it on your own before proceeding with the solutions below.

Here, you will learn how to stop, remove, and clean up the containers that you have created.

Find your container ID and stop the container before removing it. You can stop containers using the name or the ID of the container. You can also stop multiple containers at once, as follows:

`docker stop redis`

`docker stop b584 84e6 e4da`

---

where,

- `ac69 b584 84e6 e4da` are container IDs (as per the above example)

These could vary on your system. Use the ones you see when you run the `docker ps` command. You can start a container that is in a stopped state with:

`docker start redis`

To remove a container (destructive process) that is stopped, use:

`docker stop redis`

`docker rm redis`

If the container is in a running state, it cannot be stopped unless a `force` option is used:

`docker container run -idtP --name web nginx`

`docker rm web`

`docker rm -f web`

Images are independent of containers. You can remove images using the following commands:

`docker image rm 4206`

`docker image rm 4f1 4d9 4c1 9f3`

`docker image prune`

where `4206` and `4f1 4d9 4c1 9f3` are the beginning characters of the image IDs on our host. You can get all your Docker system information by using `docker system info`.

## Setting Up Nextcloud

You should now have some insight into running applications using containers. Next, you will set up the Nextcloud application using the official image, creating a volume and mounting it inside the container. Because of the [documentation](#), we know the container directory we must mount our volume to is `/var/www/html`. You may look at it for your information, but it is not necessary.

Follow these steps to set up Nextcloud:

`docker container run -idt -P -v nextcloud:/var/www/html nextcloud`

You will see Docker download the image and finally print the container ID.

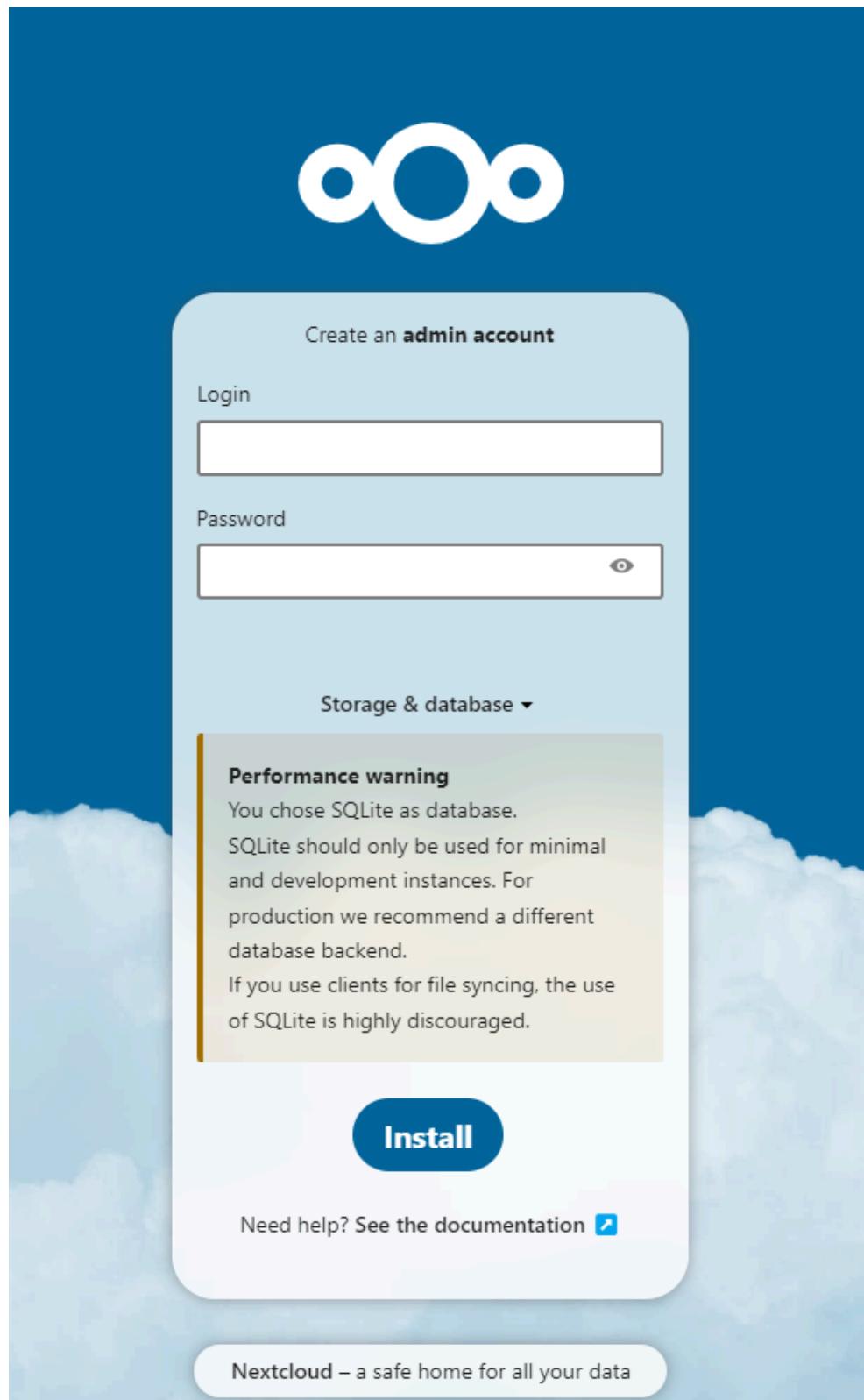
Use the following commands to get the running container port and logs:

`docker ps`

`docker logs ac69`

`docker logs -f ac69`

Example: You can visit the application by using `localhost:32770` on your browser.



## Setting up Portainer with Advanced `run` Options

Here, you will launch and practice more advanced `run` options using the Portainer application.

In this case, you need to create a volume before running the container and validate the volume by using the following commands:

```
docker volume create portainer_data
```

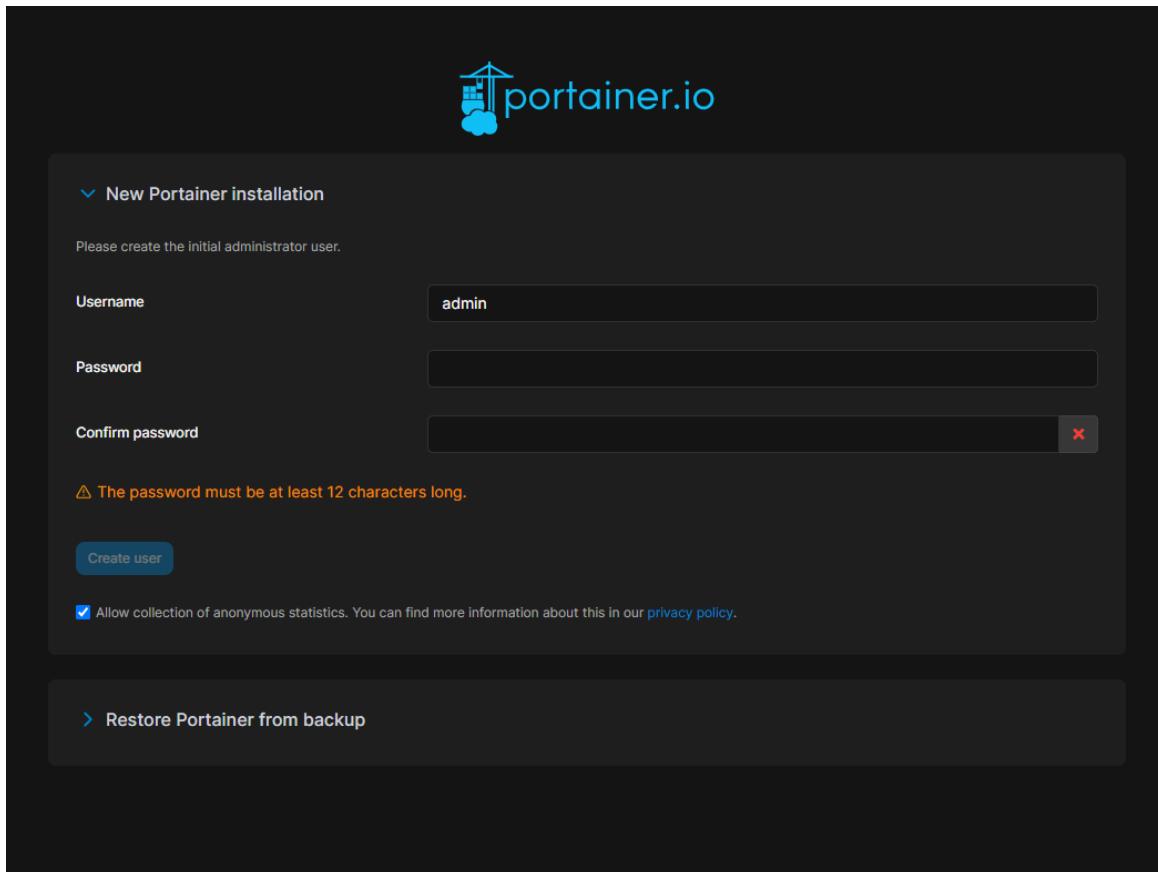
```
docker volume ls
```

```
docker volume inspect portainer_data
```

After creating the volume, run the `docker container run` command along with the volume which we have created:

```
docker run -d -p 9000:9000 -v  
/var/run/docker.sock:/var/run/docker.sock -v  
portainer_data:/data/portainer/portainer
```

Access the Portainer application using port 9000 in your browser. To do this, enter `localhost:9000` or `IPADDRESS:9000` in your browser's address bar.



## Cleanup

First, we need to stop any running containers. To get all running containers and their IDs, run:

```
docker ps
```

Keep the first set of unique characters of each container ID handy. These are all you need to enter after the command. Run:

```
docker stop containerID1 containerID2 containerID3...
```

Don't type the '...'. That is simply to indicate that you may have more containers.

Now that all the containers have been stopped, we can run:

```
docker container prune
```

Type 'y' to continue.

Finally, run:

```
docker system prune
```

Type 'y' to continue.

Get your images with:

```
docker image ls
```

Take note of the image IDs. Then run:

```
docker image rm imageID1 imageID2 imageID3...
```

Run:

```
docker system info
```

This should verify that you have 0 containers running, paused, and stopped.

## Summary

In this lab, we spun up multiple Docker containers with different options. These options allowed us to interact with these containers through their shells and via networking. Finally, we cleaned up our Docker environment so that we could move on to other concepts with a clean slate.



## Lab 3. Revision Control with Git

By the end of this lab exercise, you should be able to:

- Configure a Git client
- Create and work with repositories
- Understand the basics of revision control operations
- Understand basic branching and merging
- Work with remotes
- Resolve conflicts
- Leverage Docker for local development

### Configuring Git Client

Before you start revision control, you need to configure Git. There are three different places you can configure Git, as explained here.

Git configs:

Type	Scope	Config Path
LOCAL	Repo	.git/config
GLOBAL	User	/home/user/.gitconfig
SYSTEM	System	/etc/gitconfig

You will create configurations with user scope, valid for all repositories created by the current user:

```
git config --global user.name "username"
git config --global user.email "user@gmail.com"
git config -l
```

---

```
cat ~/.gitconfig
```

You can also create additional configurations, e.g. to switch the default editor to a terminal text editor called *Vim*:

```
git config --global core.editor vim
```

To see the change, run the following command:

```
git config -l
```

To edit the configuration file, use the `-e` flag as follows:

```
git config -e --global
```

This will open the file in Vim. To exit Vim, type the following:

```
:q
```

This will exit Vim and bring you back to the terminal.

Vim will now be the default editor for Git. *Nano* and *Emacs* are two other command line text editors that can be used. Pick your favorite. Nano is the most intuitive.

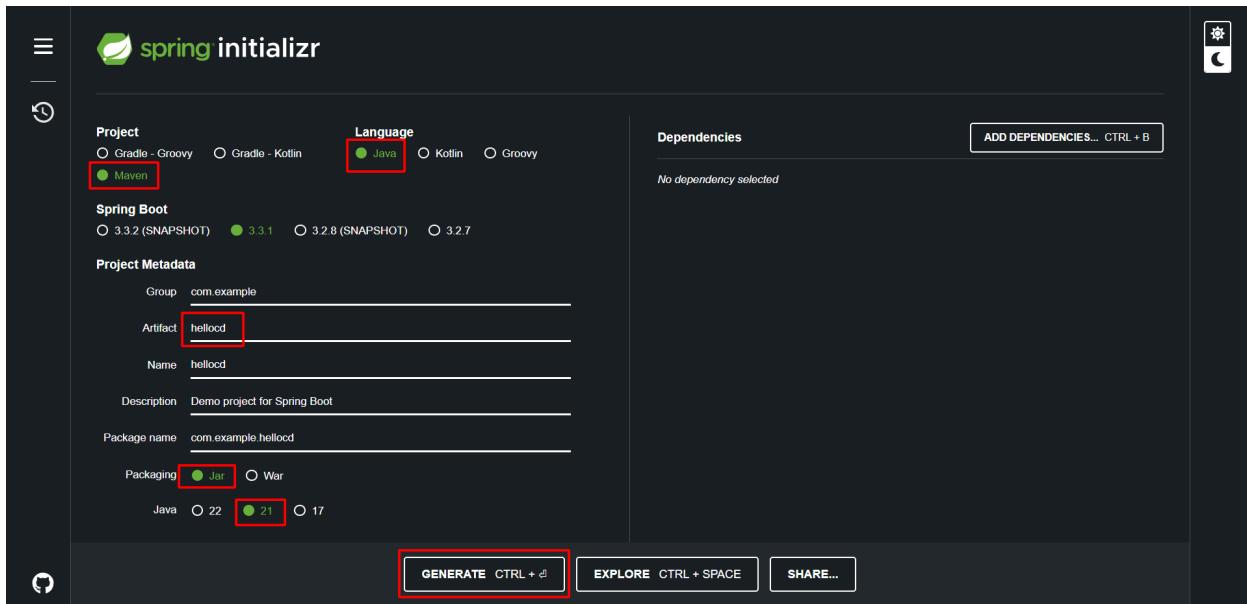
## Creating a Sample Application

We need an application to use version control on.

We will guide you through creating a simple `hello-world` application using the Java-based `spring-boot` framework.

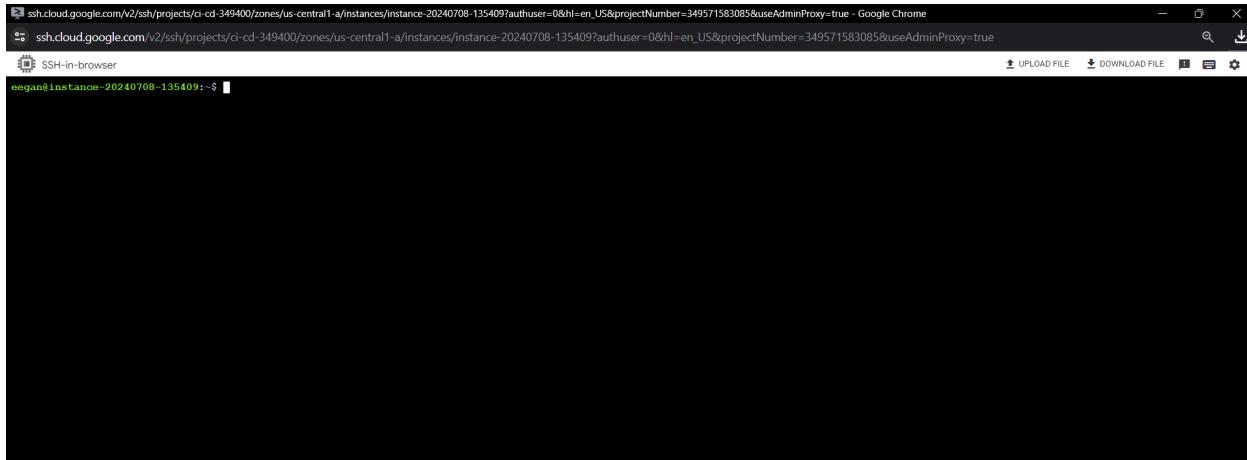
To create the application, visit <https://start.spring.io/>.

Be sure to select the **Maven Project**. Change your artifact name to `hellocd`. This can be anything you want, but we will change the artifact name to `hellocd`. Select **Java 21**. Click **Generate** to download your application.



After downloading the application code, move the `.zip` file to your local working directory and unzip that application.

If you are working on a Google Cloud VM, upload the file via the **UPLOAD FILE** link in the top right of the terminal.



Choose the file to upload and it will upload to the current working directory. Install the unzip program:

```
sudo apt install unzip
```

Unzip `hellocd.zip`:

```
unzip hellocd.zip
```

You will use this application for revision control operations.

## Revision Control Basics

Now that you have a `hellocd` application downloaded and unzipped, you will use it to learn revision control operations. Enter the `hellocd/` directory.

```
cd hellocd
```

We will tell Git to make this directory a repository by *initializing* it. We do this with the following command from inside `hellocd/`:

```
git init
```

Now run:

```
git status
```

`git status` shows the status of all the files and directories in the repository. Untracked files and directories appear in red. You need to add these files so that Git starts tracking them. Use the following commands to start tracking the files in the `hellocd` project with Git:

```
git add pom.xml
```

```
git status
```

```
git commit
```

```
git status
```

```
git add src
```

```
git status
```

```
git add *
```

```
git commit mvnw
```

```
git status
```

```
git commit -am "adding files created with spring boot generator"
```

```
git log
```

`git log --branches` shows you the commit history and where each branch is at, while `git status` shows untracked files in your directory. Create one more file in your working directory

---

called `README.md`.

**Note:** the following command will start with whatever your chosen command line text editor was, nano, emacs, or vim:

```
vim README.md
```

Check your git status after that commit using `reset`:

```
git add README.md
```

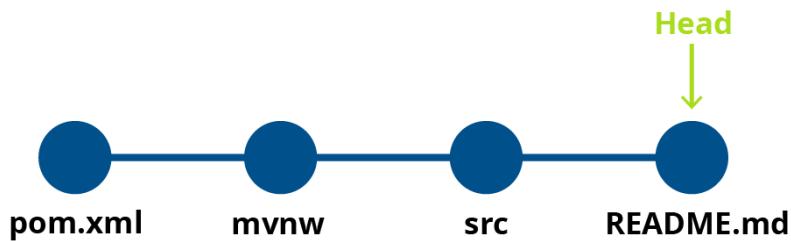
```
git status
```

```
git reset README.md
```

```
git add README.md
```

```
git commit -am "adding README"
```

You have added and committed the files one by one, and this is the current state of your repository with each commit represented by a circle on the line:



## Using Docker in Development

After creating a Java project based on Maven, we need to build and test it before deploying it into an environment. To do this, you will need a build environment. Instead of setting up and managing a build environment with Maven, Java, etc., you can leverage Docker's disposability feature.

You need to get the absolute path to the `hellocd` directory. From inside `hellocd` run:

```
pwd
```

The output will be the absolute path to `hellocd`. Be sure to use it in the following command in place of `/absolute/path/to/hellocd`.

---

Use the following command to run the container:

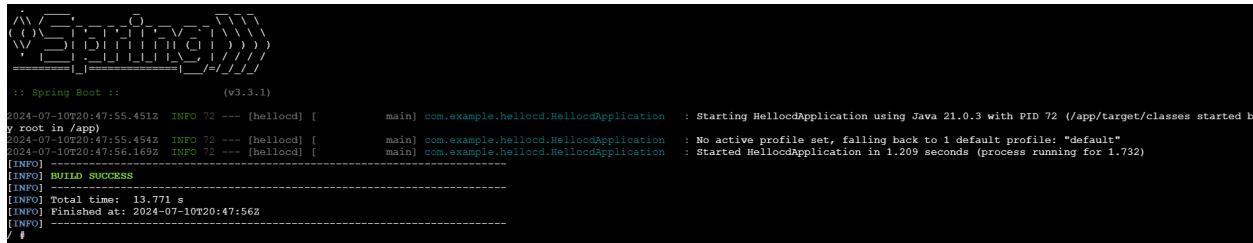
```
docker container run --rm -it -v /absolute/path/to/hellocd:/app
maven:3.9.8-eclipse-temurin-21-alpine sh
```

Running the above command gets you:

- A disposable container environment with Docker
- Created with the image `maven:3.9.8-eclipse-temurin-21-alpine`
- This container has build tools necessary to run a Java application, including Maven, Java, etc.
- Exiting this container will delete it, as we are using the `--rm` option.

You will see that your terminal prompt has switched to a # which indicates that you have a terminal inside the container. While inside the container, run the project with:

```
# mvn spring-boot:run -f app/pom.xml
```



```
^[[?12l
:: Spring Boot ::      (v3.3.1)
2024-07-10T20:47:55.451Z  INFO 72 --- [hellocd] [           main] com.example.hellocd.HellocdApplication : Starting HellocdApplication using Java 21.0.3 with PID 72 (/app/target/classes started by root in /app)
2024-07-10T20:47:55.452Z  INFO 72 --- [hellocd] [           main] com.example.hellocd.HellocdApplication : No active profile set, falling back to 1 default profile: "default"
2024-07-10T20:47:56.169Z  INFO 72 --- [hellocd] [           main] com.example.hellocd.HellocdApplication : Started HellocdApplication in 1.209 seconds (process running for 1.732)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  13.771 s
[INFO] Finished at: 2024-07-10T20:47:56Z
[INFO] -----
/ #
```

You can see your libraries are stored in the `/root/.m2/` directory.

```
# ls /root/.m2/repository
ch  com  commons-io  commons-io  io  jakarta  javax  net  org
```

Type `exit` to return to your machine's terminal:

```
# exit
```

The container is immediately destroyed. You can reduce the time to download libraries by creating a volume which can then be mounted at this path.

```
docker volume create m2
m2
```

Run the following command:

```
docker container run --rm -it -v m2:/root/.m2 -v /path/to/hellocd:/app
maven:3.9.8-eclipse-temurin-21-alpine mvn spring-boot:run -f
```

---

### /app/pom.xml

Run the same command again. You should notice the build is faster due to the Maven package being cached with the newly mounted volume.

## Branching and Merging

Create a branch using the `git branch` command and switch to that branch using `git checkout`. The default branch is master. Whenever we need to work on another branch, we use the `git checkout` command. Let's run the following commands to see our branches and create and move to the `webapp` branch:

```
git branch --list  
git log  
git branch webapp  
git branch --list  
git checkout webapp  
git branch --list
```

Instead of using a two-step process, you could have created and switched to the `webapp` branch with the following single command:

```
git checkout -b webapp
```

You can delete your branch using `git branch -d`. Before deleting the branch, you should switch to another:

```
git checkout master  
git branch -d webapp
```

Now `git branch` will show that you only have the `master` branch. Since we want the `webapp` branch, we will create it and switch to it one more time:

```
git checkout -b webapp
```

## Adding a New Feature by Branching Out

Now you are going to add new features to your application:

- For source code changes, visit the [Ift 261 devops-repo](#) and go to the `hellocd` directory. Copy `HellocdApplication.java.v1` code and replace the code in `HellocdApplication.java` which you will find in your `src/main/java/com/example/hellocd` directory.
- Copy `pom.xml.snippet1` from <https://github.com/lftraining/devops-repo/blob/master/hellocd/pom.xml.snippet1> and paste it into the `pom.xml` file under the dependencies section between `<dependencies>` and `</dependencies>` before or after the dependencies you see present there.

Now you can check your git status. The result will be that those files you have changed are unstaged. We will add them, but first you can review the changes made using the following:

`git diff`

Continuously press `Enter` or hold `Enter` down to scroll to the bottom to see all changes. Press `q` to exit.

You can commit after `git add` or else directly commit by using `git commit -a`. The `-a` option, if you recall, means '*add all files with changes to the staging area*'. We will also use the `-m` option to add a message from the command line:

```
git commit -am "committing changes to pom.xml and
HellocdApplication.java"
```

`git status`

Now you can see the changes are in the new branch. The master still does not have any of those incorporated yet. This is the safer way of bringing your code to master. The following subsection will teach you how to bring these changes into the master.

## Merging with Commit History

Previously you made some changes and committed to a new branch called `webapp`. It's now time to test the application.

Again, run the same `spring-boot` container command with the port mapped to 8080 on host:

```
docker container run --rm -it -v m2:/root/.m2 -v /path/to/hellocd:/app
-p 8080:8080 maven:3.9.8-eclipse-temurin-21-alpine mvn spring-boot:run
-f /app/pom.xml
```

Once the container is running, you can visit the application on `http://IPADDRESS:8080` or `http://localhost:8080`. When you visit your `webapp`, right now you will get an error due to some missing code in `HellocdApplication.java`. To fix this issue, copy the code inside `HellocdApplication.java.v2` and use that code to replace *all* the code inside your own `hellocd/src/main/java/com/example/hellocd/HellocdApplication.java` file. After replacing the snippet, run the following command to see your changes:

```
git diff
```

You will see that Git has made the added lines green. Continuously press `Enter` or hold `Enter` down to scroll to the bottom to see all changes. Press `q` to exit.

```
git commit -am "added missing annotations"
```

Again, run your `spring-boot` container and visit `localhost:8080`. You should now see `Hello World!` in your browser.

Change “Hello World!” to “Hello Continuous Delivery!” in `HellocdApplication.java`. Commit the change, then run your container. You will see `Hello Continuous Delivery!` in the browser.

```
1 package com.example.hellocd;
2
3 import org.springframework.boot.*;
4 import org.springframework.boot.autoconfigure.*;
5 import org.springframework.web.bind.annotation.*;
6 import org.springframework.boot.SpringApplication;
7 import org.springframework.boot.autoconfigure.SpringBootApplication;
8
9 @SpringBootApplication
10 @RestController
11 @EnableAutoConfiguration
12 public class HellocdApplication {
13
14
15     @RequestMapping("/")
16     String home() {
17         return "Hello Continuous Delivery!";
18     }
19
20     public static void main(String[] args) {
21         SpringApplication.run(HellocdApplication.class, args);
22     }
23
24 }
```

Now that the feature is added, it's time to incorporate these changes into the main branch by using `git merge`. Before you do so, make sure you switch to the main branch first:

```
git checkout master
```

```
git merge webapp
```

```
git log --branches
```

We can now delete our `webapp` branch using `git branch -d`:

```
git branch -d webapp
```

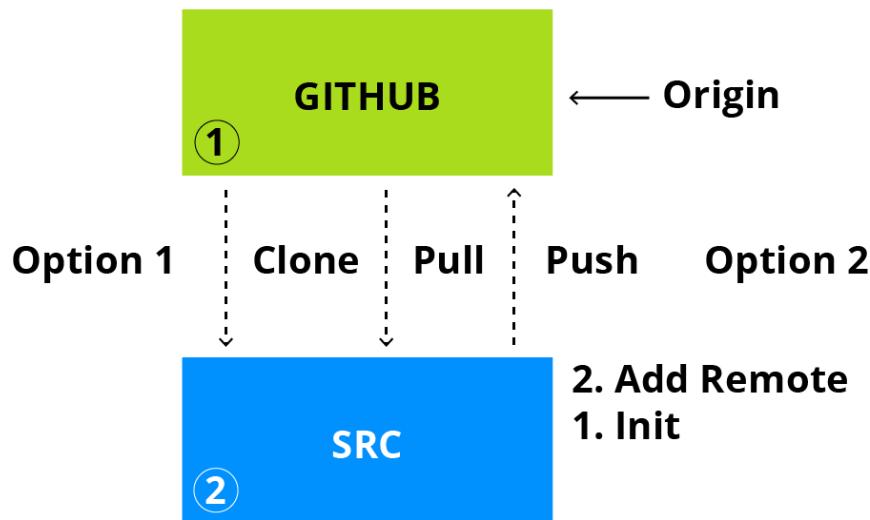


## Working with Remotes

You can collaborate and work with different branches, users, or central repositories by using *remotes*. Here, your central repository is GitHub.

There are two ways to go about this:

1. Create the repository on GitHub first.
  - a. Clone the repository on your local machine.
2. Create the repository on your machine first.
  - a. Create the repository on GitHub
  - b. Tell your local repository to make the remote repository your GitHub repository



Before you start, you need to create an account on GitHub. Visit [GitHub](#) to create your account. While creating your account, use the free account option.

Once you create an account, create a new `hellocd` public repository. You have two ways to work with a repository. You could use an initialized repository or clone the repository from GitHub and make changes and push to your repository.

Here you have already initialized a repository. Now you need to add the newly created repository as origin by using the following command and push your application to that origin master:

```
git remote add origin https://github.com/xxxxx/hellocd.git
```

[Replace `xxxxx` with your GitHub user/org name, as per your repository URI]

```
git remote show
git remote show origin
git push origin master
```

Once you push to the origin master, you can see your code in the repository with the commit that you have changed.

## Resolving Conflicts

When you are collaborating with others, conflicts can happen. Git provides you with various options to resolve them.

---

Add the line `server.port=80` in `src/main/resource/application.properties` while you are on the main branch, commit and push:

```
git diff  
git commit -am "added server port 80"  
git push origin master
```

## Creating a Conflict

To resolve a conflict, we first have to create one. To do that, you will create two branches and update the same file with different content in each branch. The following sequence of commands will help you do that.

You have the option to work in pairs. The commands are differentiated between `user1` and `user2`. Make sure you are working with the same repository in case you are working in pairs. If you cannot find a partner, go ahead and run both blocks of commands from the same workspace.

`user1`: Use the following code:

```
git checkout -b user1
```

Set the value of `server.port` to 8080 in `src/main/resource/application.properties`:

```
git diff  
git commit -am "this should run on port 8080"  
git push origin user1
```

`user2`: Use the following code:

```
git checkout -b user2
```

Set the value of `server.port` to 8081 in `src/main/resource/application.properties`:

```
git diff  
git commit -am "this should run on port 8081"  
git push origin user2
```

Once you do the above, switch to the main branch and try to merge the branches you just updated, simultaneously. Each will run into a conflict.

```
user1:  
git checkout master  
git pull origin user2  
git log --branches  
git push origin master
```

user2:

```
git checkout master  
git pull origin user1  
git log --branches  
git push origin master
```

You should get into a conflict here. Use `git log` to check the details.

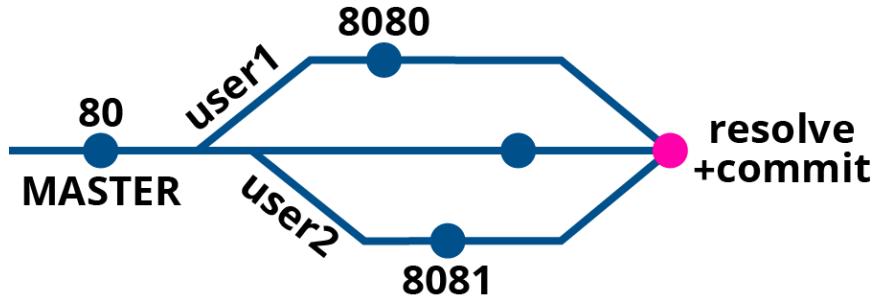
## Manually Resolving a Conflict

You can resolve a conflict manually by editing the file in conflict. In this example, you can pick one value for `server.port` and keep that line in the file. Remove everything else including the lines starting with <<<<<< and >>>>>>. Once you do so, commit the changes:

```
git commit -am "resolved merge conflict"  
git push origin master
```

Validate by examining the commit history:

```
git log --branches
```



Once you complete this, delete your user1 and user2 branches by using the following commands:

```
git branch -d user1
git branch -d user2
git push origin --delete user1
git push origin --delete user2
```

## Learning to Undo

It's common to add changes to the commit history and later wish you could have taken those back. At times, mistakes or bugs you detect in your feature branches do not need to be reflected in the commit history. There are times when you may wish to unstage a file. Git offers the **reset** command with its options to achieve this.

Let's add a change that we will undo later:

```
git checkout -b config
```

```
git branch
```

Update **src/main/resource/application.properties** with **server.port=9000**:

```
git status
```

```
git add src/main/resource/application.properties
```

```
git status
```

---

The file has now been staged. To unstage the same:

```
git reset src/main/resource/application.properties
```

```
git status
```

```
git diff
```

```
git commit -am "changed port to 9000"
```

```
git log
```

```
git push origin config
```

You have not only added it to a commit history, but also pushed the change to the remote.

Let's say that you have realized that 9000 should really be 8080.

Try using the following command:

```
git reset HEAD~1
```

Note: that is HEAD *tilda* 1. Don't mistake the tilda for a dash, or it won't work. This should revert the last commit and point the **HEAD** (the tip of the Git repository) to the previous commit. You could use **HEAD~2**, **HEAD~3** to shift the **HEAD** back by two/three commits, respectively. You could optionally use the commit ID to shift **HEAD** to a specific commit.

What you did above was a mixed reset. This means the undo was done for the commit history only. You would still see the files in the staging area, and the current working tree being unchanged.

You can undo it all the way (this is a destructive process) by doing a hard reset:

```
git reset --hard HEAD~1
```

Once you reset, if you try to push the changes to remote, it should give you an error. To undo changes to remote, use the **force** option:

```
git push origin config -f
```

## Revert vs Reset

**reset** is useful when working on feature/personal branches. However, when you have a collaborative workflow with common branches such as trunk/mainline/master, resetting changes on common branches can leave everyone else confused and in an inconsistent state.

This is because `reset` wipes out the commits to undo changes. A cleaner solution in such cases is to use `git revert` instead, which undoes the changes by creating a new commit that is then part of the commit history, leaving everyone in sync.

To learn how `revert` works, let's first update a file and commit it.

Use the `config2` branch and file and commit the changes in the feature branch, bring those changes to the main branch using the following commands:

```
git checkout -b config2
```

Update `src/main/resource/application.properties` with `service.port=7000`:

```
git diff
```

```
git commit -am "run on port 7000"
```

```
git push origin config2
```

```
git checkout master
```

```
git pull origin config2
```

```
git push origin master
```

Now that you have incorporated these changes to the main branch, it's time to revert:

```
git log
```

```
git revert HEAD
```

or

```
git revert 5080dcc9d13ac920b5867891166a
```

Where `5080dcc9d13ac920b5867891166a` is the previous commit ID that you would want to revert.

Once you revert, you need to sync the changes in the main branch with remote by using the following command:

```
git log
```

```
git push origin master
```

Once you sync the changes, it will show the changes to your co-developer and delete your `config2` branch after `git revert`:

Go ahead and delete the branch:

```
git branch -d config2
```

```
git push origin :config2
```

## Working with GitHub

Next, create a new repository to keep a journal. On GitHub, create a repository and add a description and `init` with a `README.md` file with the license for your public repository.

Once you create the repository, clone it locally by using the following command and add your skills in a file, then push to your origin. Create a file inside the repository called `skills.md`:

```
git clone https://github.com/yourUsername/skills-journal.git
```

Create a file inside the `skills.md` repository then commit it:

```
git commit -am "added skills"  
git remote show  
git remote show origin  
git push origin master
```

If you like, you can create more files, commit, and push them to your remote repository on GitHub.

## Summary

In summary, we initialized a local Git repository on our machine, created a Spring Boot application within that repository, made changes, and tracked those changes using Git. We ran a Java and Maven container so that we would not have to load Maven on our local machine. This container ran our SpringBoot application. We connected our local git repository to a remote git repository on GitHub. Finally, we learned how to handle conflicts when merging branches both locally and remotely.



## Lab 4. Setting Up Continuous Integration with Jenkins

By the end of this lab exercise, you should be able to:

- Set up Jenkins using Docker
- Take a walkthrough of the Jenkins web interface and essential configurations and start creating freestyle and maven jobs
- Integrate with tools such as Node.js and Maven
- Integrate with GitHub and set up build triggers
- Set up pipelines which run automated builds and unit tests

### Install Docker Compose

Follow the directions at <https://docs.docker.com/compose/install/> to install Docker Compose on your machine.

### Set Up Jenkins with Docker

You will learn how to set up Jenkins using Docker. Docker must be installed before this lab.

You will be running a Jenkins container on your Docker host by using the Jenkins image with version `jenkins/jenkins:2.492.3-1ts-jdk21`. Use the following commands to clone the `devops-repo` directory and launch the Jenkins container:

```
git clone https://github.com/lftraining/devops-repo.git  
cd devops-repo/setup
```

Inside the `devops-repo/setup` directory is the `docker-compose.yml` and the Dockerfile

---

that we will be working with.

```
docker compose build
```

```
docker compose up -d
```

Access the Jenkins UI by browsing to `http://IPADDRESS:8080`. In this case, it will most likely be `http://localhost:8080` if you are on your own machine. If you are in the cloud, you will need to find the public IP of your cloud machine.

---

#### Getting Started

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

**Administrator password**



[Continue](#)

---

To fetch the `initialAdminPassword` use the following command:

```
docker exec -it setup-docker-1 cat /var/jenkins_home/secrets/initialAdminPassword
```

The output will be the initial Jenkins password. Paste it into the Jenkins UI to unlock and click **Continue**.

Getting Started

# Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

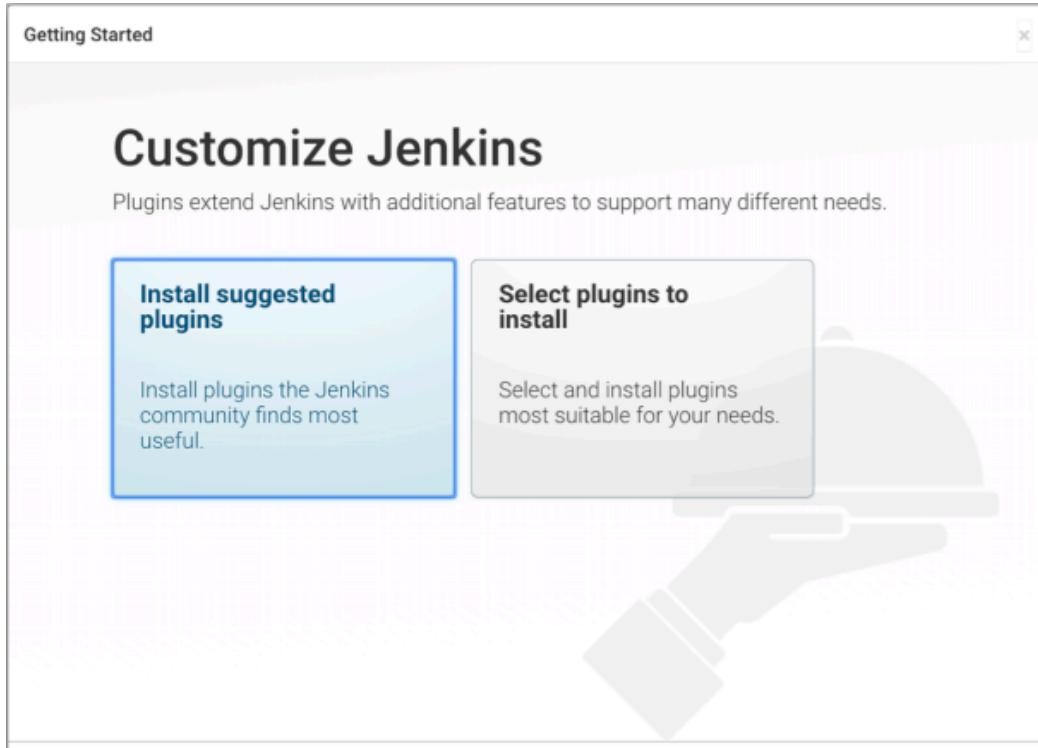
Administrator password



**Continue**

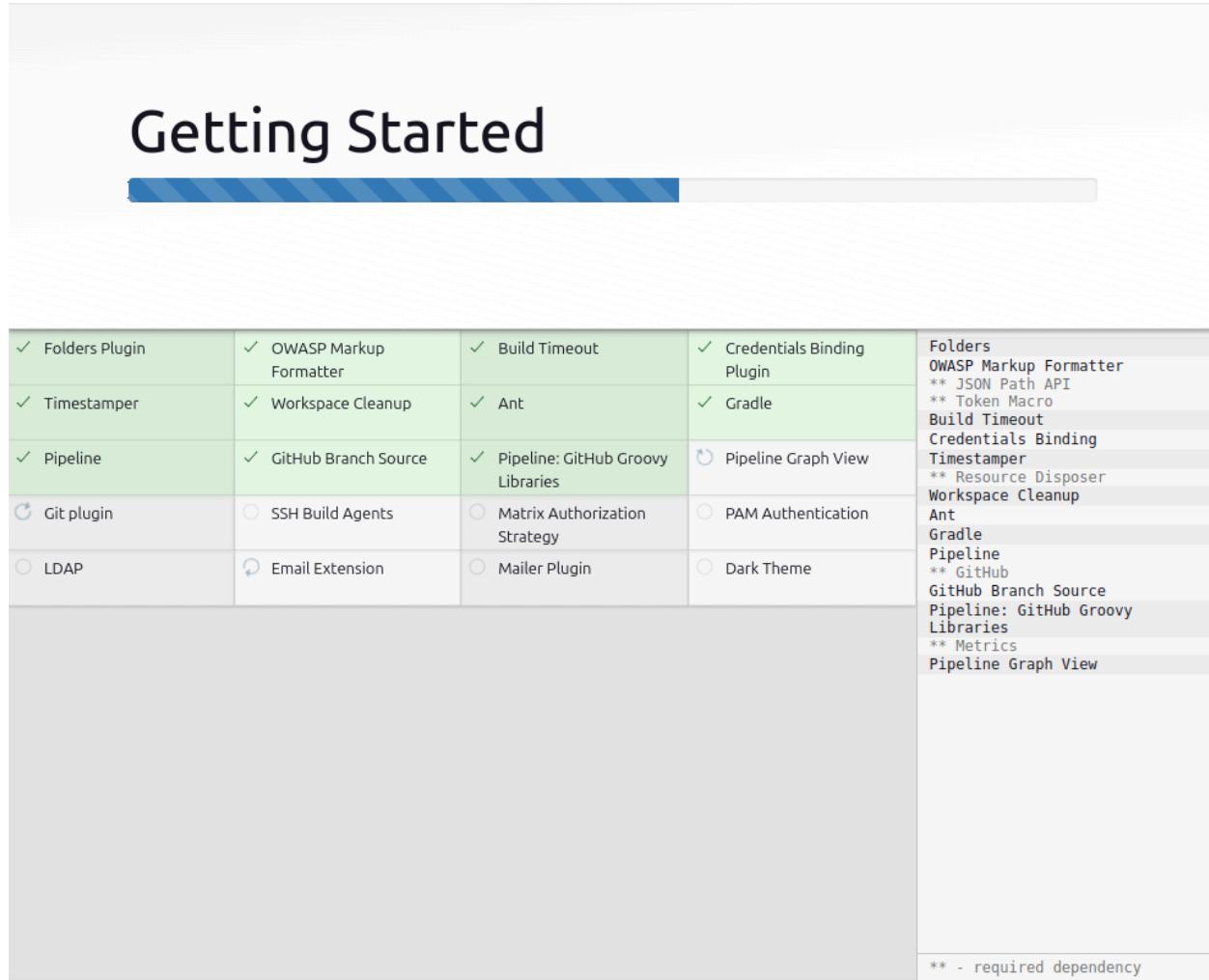
---

In the next step, choose **Install suggested plugins** to configure the default plugins automatically.



You will be able to observe the progress of the plugin installation process as follows:

### Getting Started



The screenshot shows the Jenkins 'Getting Started' page. At the top, there's a large 'Getting Started' heading with a blue striped bar below it. Below this, there's a table with two columns of plugin options. The left column contains checked items: 'Folders Plugin', 'Timestamper', 'Pipeline', 'Git plugin', and 'LDAP'. The right column contains checked items: 'OWASP Markup Formatter', 'Workspace Cleanup', 'GitHub Branch Source', 'SSH Build Agents', 'Email Extension', 'Build Timeout', 'Ant', 'Pipeline: GitHub Groovy Libraries', 'Matrix Authorization Strategy', 'Mailer Plugin', 'Credentials Binding Plugin', 'Gradle', 'Pipeline Graph View', 'PAM Authentication', and 'Dark Theme'. To the right of the table, a vertical sidebar lists all the checked items from both columns. At the bottom of the sidebar, it says '\*\* - required dependency'. A note at the bottom of the page states '\*\* - required dependency'.

✓ Folders Plugin	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding Plugin	Folders OWASP Markup Formatter ** JSON Path API ** Token Macro Build Timeout Credentials Binding Timestamper ** Resource Disposer Workspace Cleanup
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle	Ant Gradle
✓ Pipeline	✓ GitHub Branch Source	✓ Pipeline: GitHub Groovy Libraries	✗ Pipeline Graph View	Pipeline ** GitHub GitHub Branch Source Pipeline: GitHub Groovy Libraries ** Metrics Pipeline Graph View
✗ Git plugin	✗ SSH Build Agents	✗ Matrix Authorization Strategy	✗ PAM Authentication	
✗ LDAP	✗ Email Extension	✗ Mailer Plugin	✗ Dark Theme	

\*\* - required dependency

Once the plugins are installed, create the admin user using the form presented. Fill out the page and click **Save and Continue**. You can use a username and password of your choice

---

## Getting Started

---

# Create First Admin User

Username

Password

Confirm password

Full name

E-mail address



---

On the **Instance Configuration** page, the URL will depend on whether you are on a local or cloud machine. The default should be fine for our purposes. Click **Save and Finish**.

---

## Getting Started

---

# Instance Configuration

Jenkins URL:

`http://localhost:8080/`

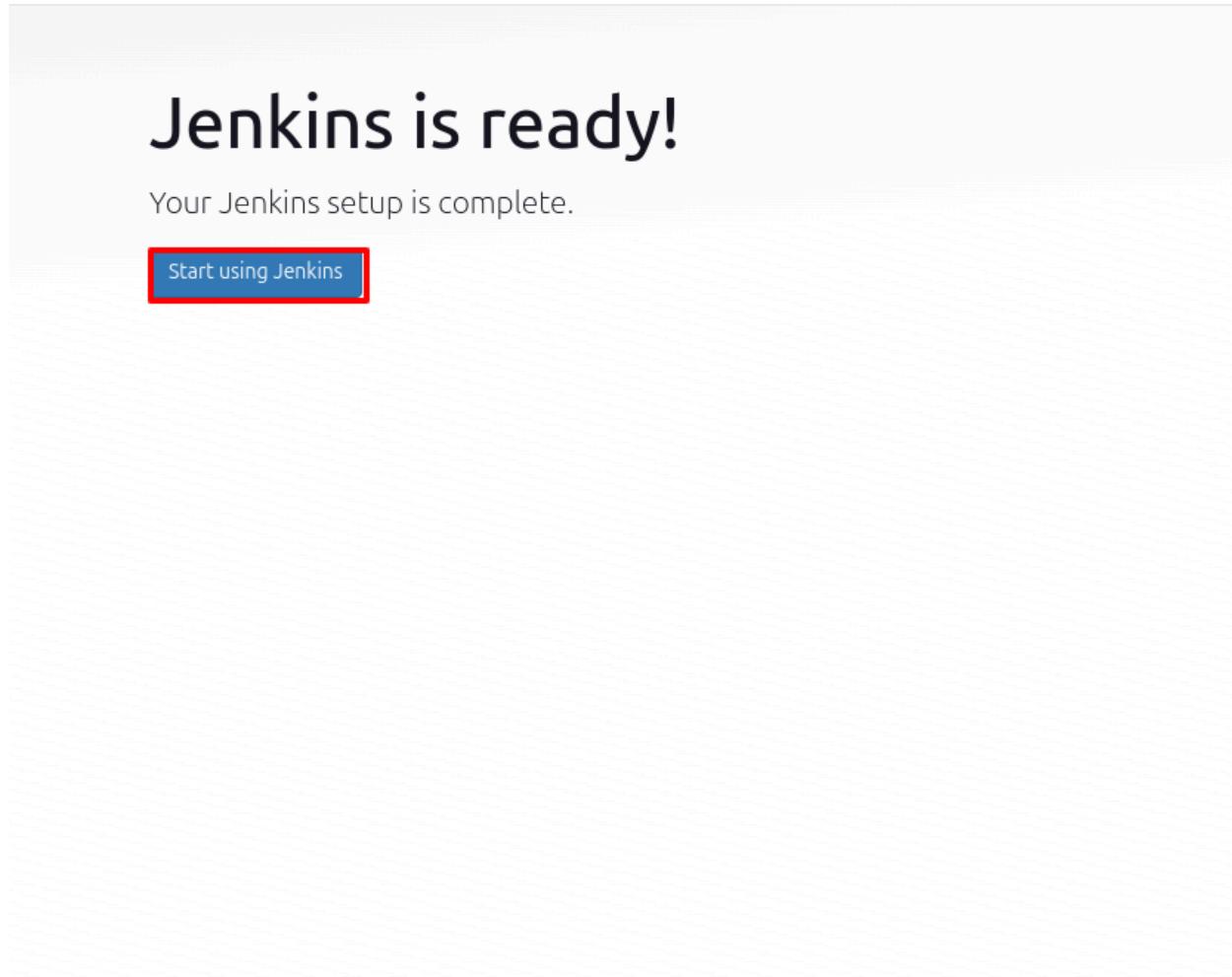
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

---

You should see a confirmation page. Click **Start using Jenkins** or **Restart**.

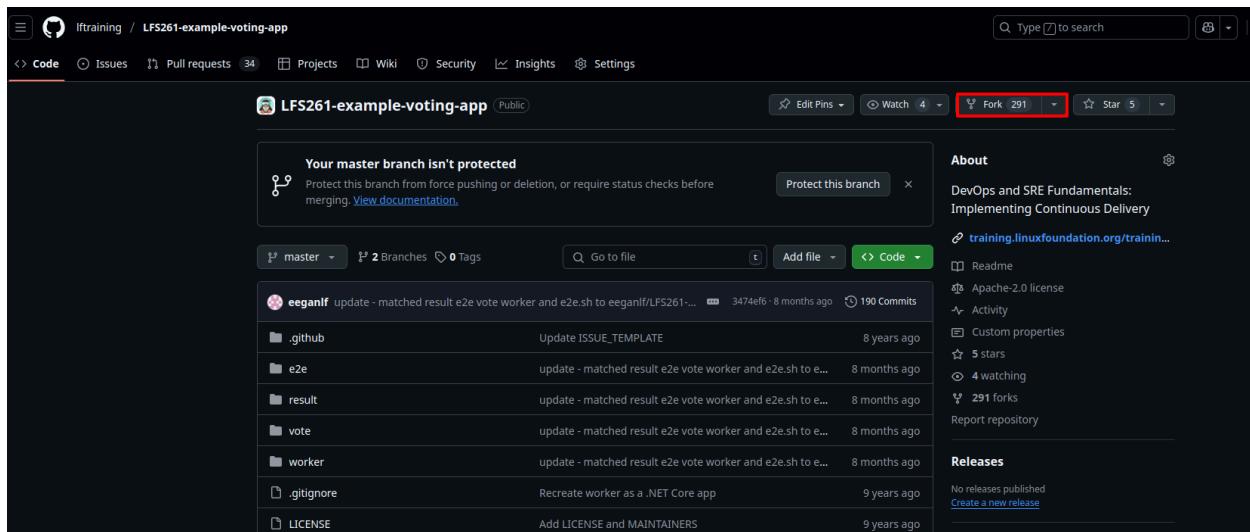
## Getting Started



The screenshot shows the Jenkins dashboard. At the top, there's a navigation bar with a Jenkins logo, a search bar containing 'Search (CTRL+K)', and user status indicators for 'admin' and 'log out'. Below the header, the main content area has a title 'Welcome to Jenkins!' and a sub-section 'Start building your software project'. On the left, there are two collapsed sections: 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (showing '1 Idle' and '2 Idle'). On the right, there are three buttons: 'Create a job' (with a '+' icon), 'Set up a distributed build' (with a server icon), 'Set up an agent' (with a monitor icon), 'Configure a cloud' (with a cloud icon), and 'Learn more about distributed builds' (with a help icon). A note at the bottom of the main section says: 'This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.'

## Fork the Voting App

Visit [example-voting-app on GitHub](#) and fork the repository onto your Git account.



Clone the repository onto your own machine.

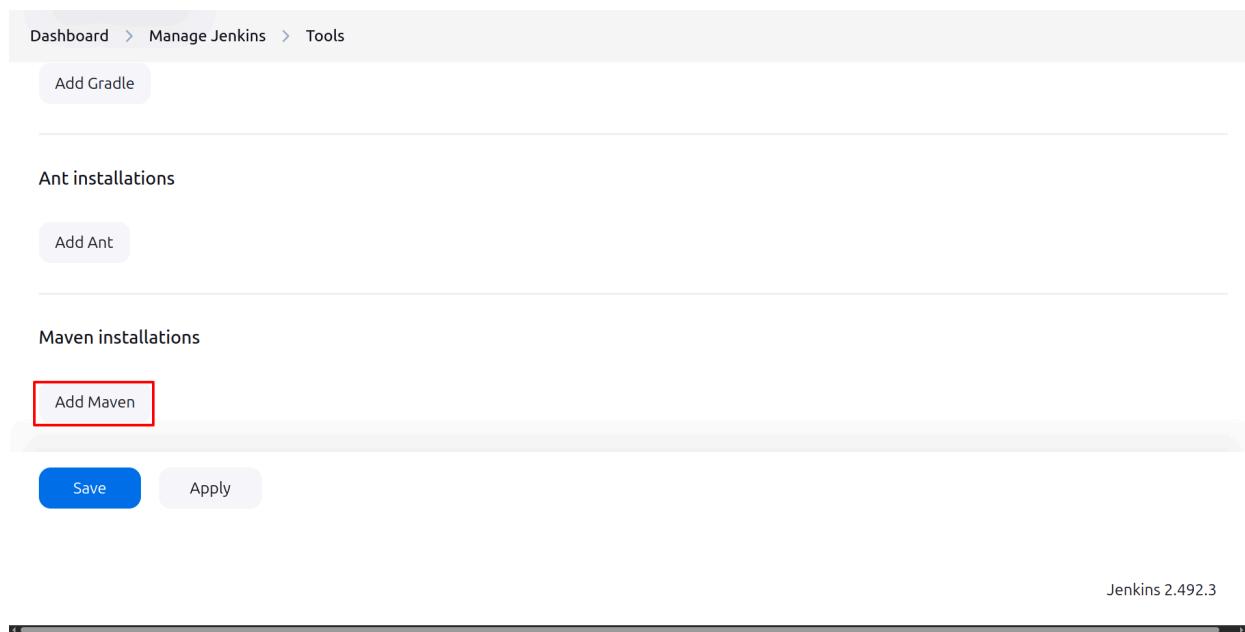
## Configuring a Maven Build Job

We went over a simple `job-01` in Chapter 6. Now you will build the `worker` application as part of the `example-voting-app` project. This is a Java application that uses Maven as a build tool.

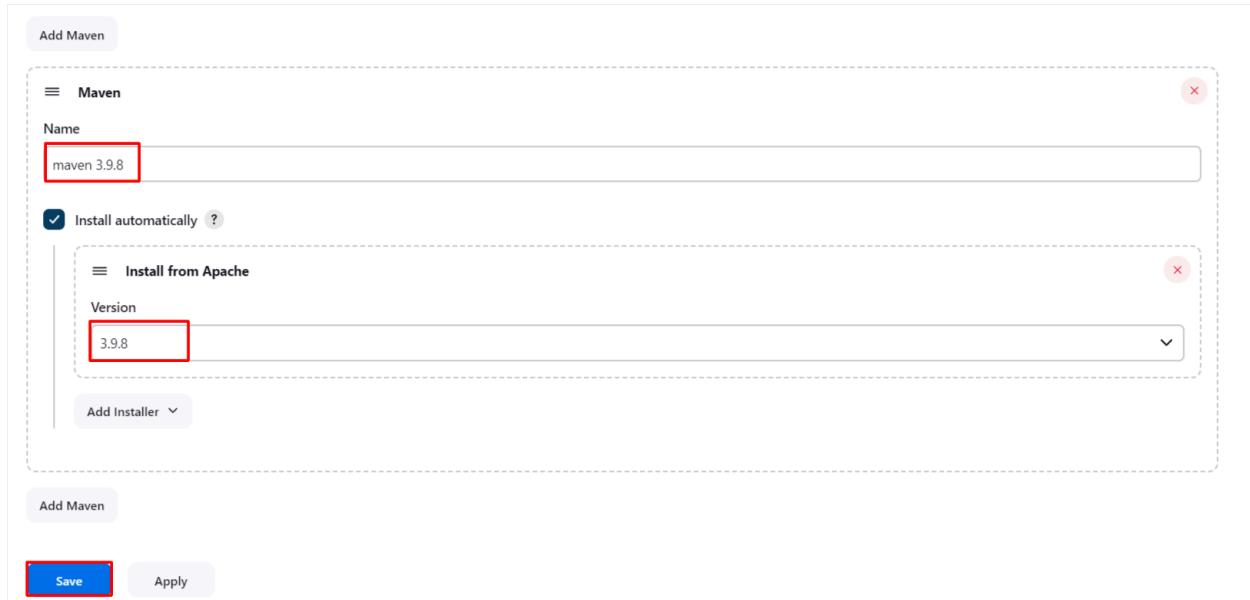
First we will configure a Maven build job.

---

Go to **Manage Jenkins > Tools** and, under the **Maven** section, click **Add Maven**.



Paste `maven 3.9.8` into the name box and select maven version **3.9.8** from the **Version** dropdown menu under **Install from Apache**. Save the changes:



**NOTE:** Remember to use the exact name in this screenshot. This is the name you are going to use to reference the Maven installation in your Jenkinsfile later. If you provide a different name, be aware of it in future labs.

Install the Maven integration plugin. Go to **Manage Jenkins > Plugins > Available plugins**, search for **Maven Integration** and **Pipeline: Stage View** plugin and install them:

The screenshot shows the Jenkins 'Available plugins' page. A search bar at the top contains the text 'pipeline'. Below the search bar, there is a table with columns for 'Install', 'Name', and 'Released'. Two specific plugins are highlighted with red boxes around their checkboxes:

- Maven Integration**: Version 3.23, released 1 yr 0 mo ago. Description: This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTs as well as the automated configuration of various Jenkins publishers such as Junit.
- Pipeline: Stage View**: Version 2.34, released 9 mo 16 days ago. Description: Provides a REST API to access pipeline and pipeline run data.

At the bottom right of the table, there is a large blue 'Install' button with a download icon.

The screenshot shows the Jenkins 'Available plugins' page again, but this time the search bar contains the text 'maven'. The same two plugins are listed, but only the first one, 'Maven Integration', has its checkbox checked. The 'Install' button at the bottom right is also highlighted with a red box.

Click **Dashboard** in the top left of the page to go back to the home page.

The screenshot shows the Jenkins 'Plugins' page. On the left, there's a sidebar with links: 'Updates', 'Available plugins', 'Installed plugins', 'Advanced settings', and 'Download progress' (which is selected). The main content area is titled 'Download progress' and includes a 'Preparation' section with three items: 'Checking internet connectivity', 'Checking update center connectivity', and 'Success'. Below this are sections for 'Javadoc', 'JSch dependency', 'Maven Integration', and 'Loading plugin extensions', each with a green checkmark and the word 'Success'. At the bottom, there are two buttons: one pointing to 'Go back to the top page' and another with a checkbox labeled 'Restart Jenkins when installation is complete and no jobs are running'. The top navigation bar includes a search bar, a help icon, a shield icon, the user 'admin', and a 'log out' button.

Create an **instavote** folder for your project. To do this, click **New item**:

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links: '+ New item' (which is highlighted with a red box), 'Build History', 'Manage Jenkins', 'My Views', and 'Open Blue Ocean'. Below this is a 'Build Queue' section with a dropdown menu showing 'No builds in the queue.' and a 'Build Executor Status' section showing '0/2'. The main content area is titled 'Welcome to Jenkins!' and includes a message: 'This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.' It also features a 'Start building your software project' button and several other buttons for 'Create a job', 'Set up a distributed build', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'.

On the item creation page, enter the name as “instavote” and select the type as **Folder**. Click **OK**:

New Item

Enter an item name

instavote

Select an item type

- Freestyle project  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Maven project  
Build a maven project. Jenkins takes advantage of your POM Files and drastically reduces the configuration.
- Pipeline  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- Multibranch Pipeline  
Creates a set of Pipeline projects according to detected branches in one SCM repository.
- Organization Folder  
Creates a set of multibranch project subfolders by scanning for repositories.

OK

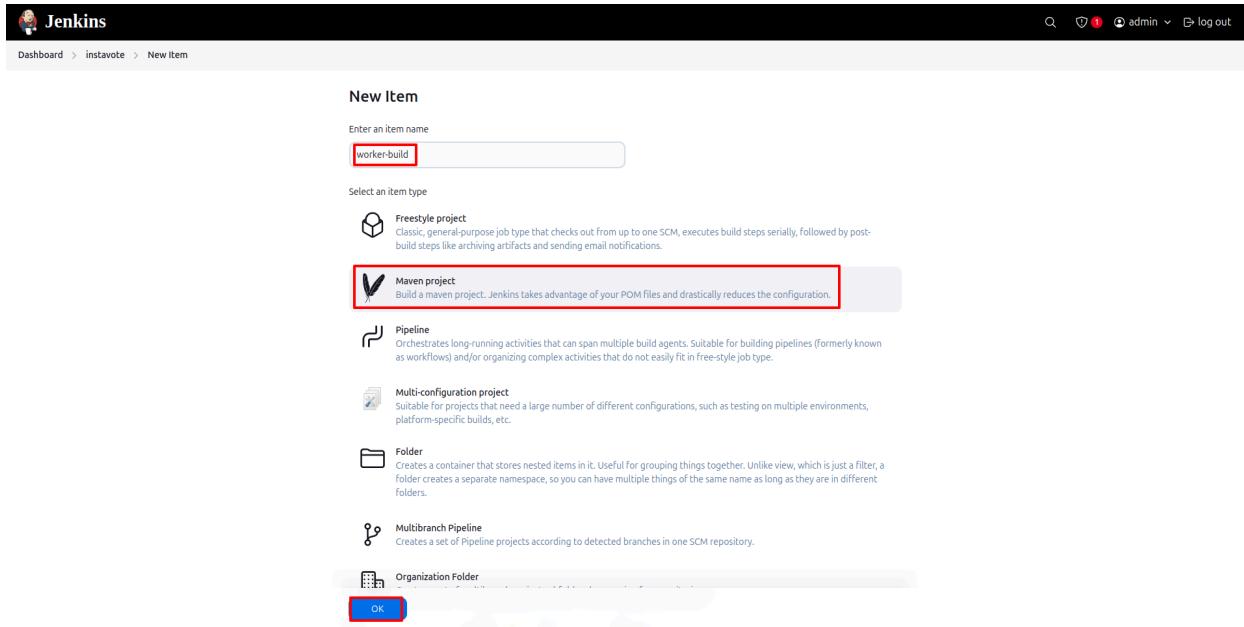
On the configuration page that appears, fill out the required details as follows and **Save**:

The screenshot shows the Jenkins configuration interface for the 'instavote' item. The 'General' tab is active. In the 'Display Name' field, the value 'instavote' is entered. In the 'Description' field, the value 'instavote folder' is entered. At the bottom of the configuration page, there are two buttons: 'Save' and 'Apply'. The 'Save' button is highlighted with a red box.

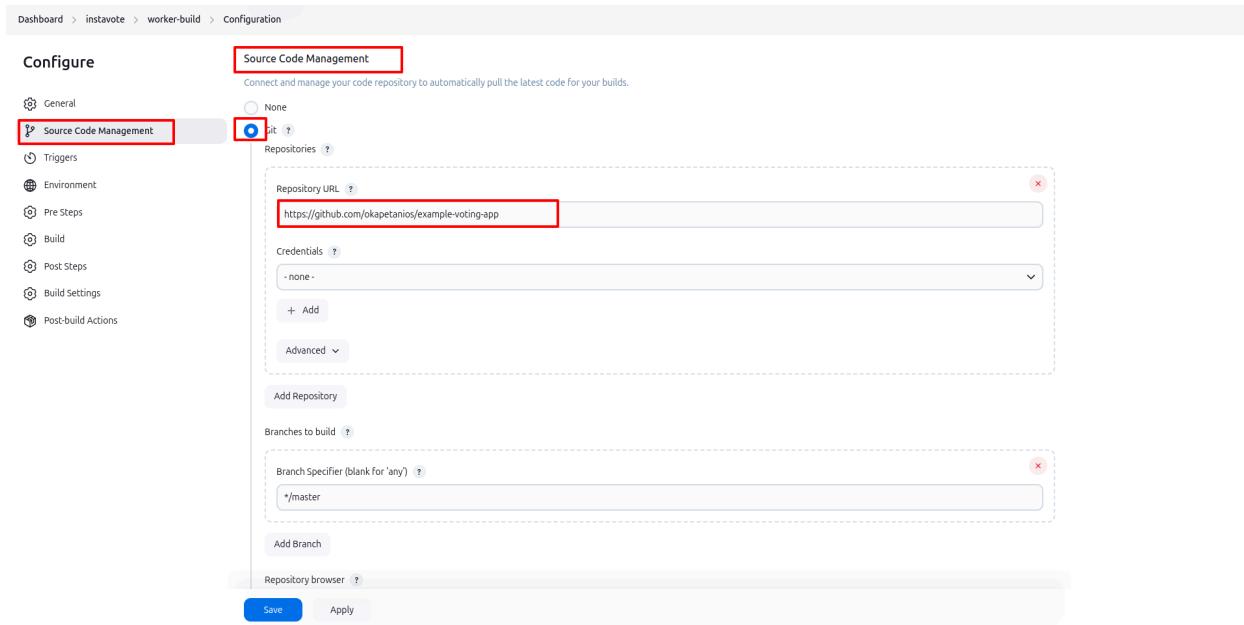
Inside the **instavote** folder, create a new item:

The screenshot shows the Jenkins instavote folder page. On the left, there is a sidebar with various options: Status, Configure, + New item (highlighted with a red box), Delete Folder, Build History, Favorite, Open Blue Ocean, Rename, and Credentials. The main area shows a folder icon labeled 'instavote' and a sub-folder labeled 'instavote folder'. Below the folder list, it says 'This folder is empty'. There is a 'Create a job' button with a '+' sign. At the bottom, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (0/2). The top right corner shows the Jenkins logo, user 'admin', and a 'log out' link.

Select **Maven project** as the project type. Name the job **worker-build** and click **OK**:



The **Configuration** page will appear. Scroll to the **Source Code Management** section and check the **Git** option. Provide the URL to your **example-voting-app** repository that you forked earlier.



---

Go to the **Build** section. Provide the path to **pom.xml**, which is in the **worker** subdirectory of the repo, i.e. **worker/pom.xml**. Type or paste **compile** into the **Goals and options** box.

The screenshot shows the 'Build' section of a Jenkins job configuration. It includes fields for 'Root POM' containing 'worker/pom.xml' and 'Goals and options' containing 'compile'. A red box highlights the 'Build' tab at the top left. Another red box highlights the 'worker/pom.xml' entry in the 'Root POM' field. A third red box highlights the 'compile' entry in the 'Goals and options' field.

Build

Root POM ?  
worker/pom.xml

Goals and options ?  
compile

Advanced ▾

Save the job and build.

Dashboard > instavote > worker-build >

 Status

 Changes

 Workspace

 Build Now

 Configure

 Delete Maven project

 Modules

 Move

 Favorite

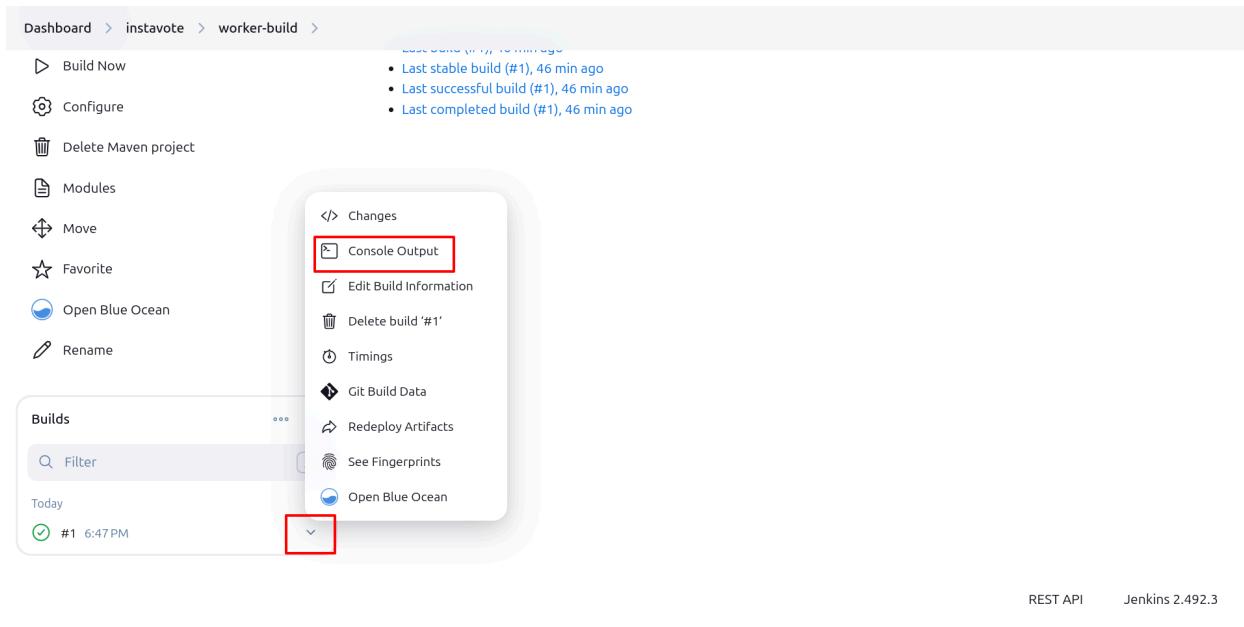
 Open Blue Ocean

 Rename

## worker-build

### Permalinks

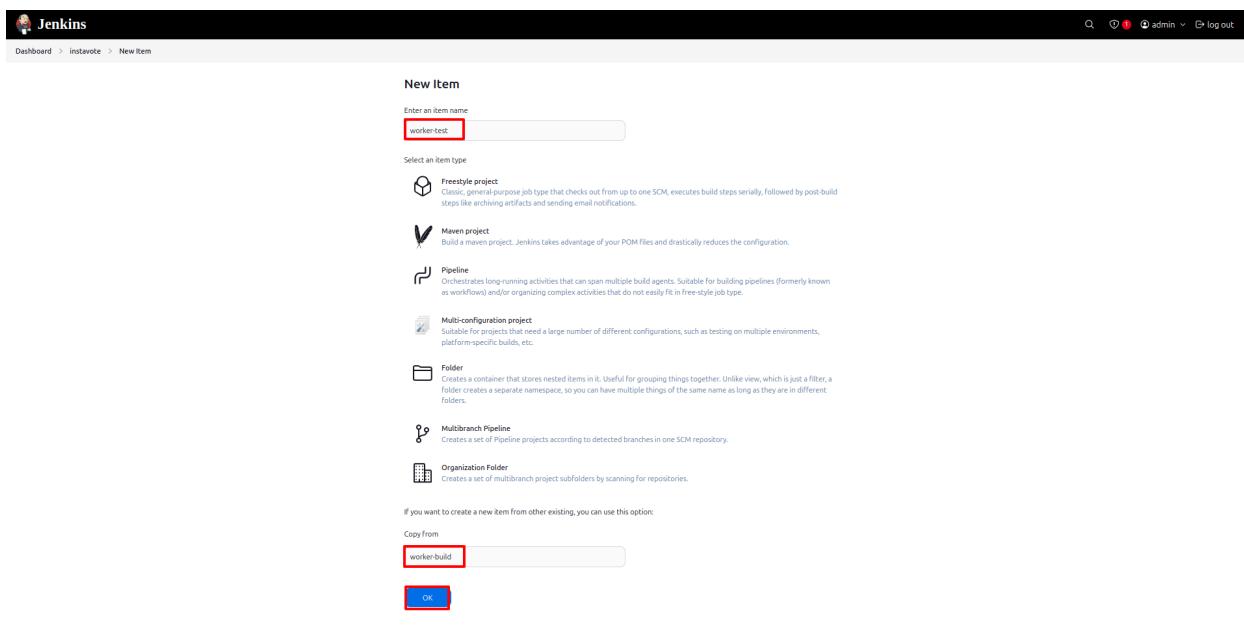
Observe the job status, console output, etc.



## Adding Unit Test and Packaging Jobs

You will now add test and package jobs for the worker application.

You need to create one more job in the **instavote** folder and name it “worker-test”. You can copy the **worker-build** job.

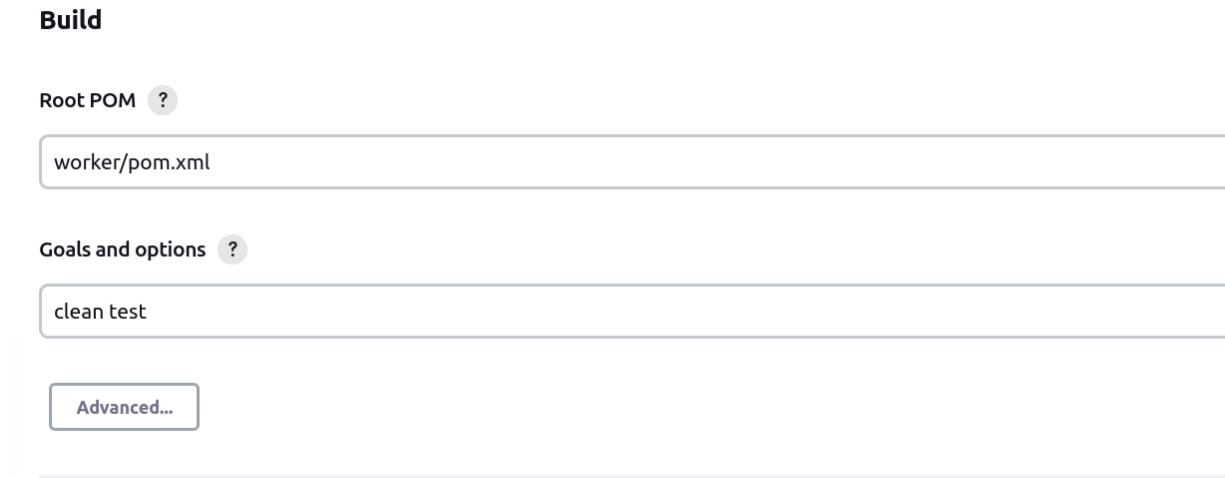


Follow these steps to complete the configuration.

---

In the **worker-test** job, change your description to “test worker java app”. The source code management repository is the same.

Under the **Build** section, change the **Goals and options** field to “clean test” and leave the rest. Save the job and build.

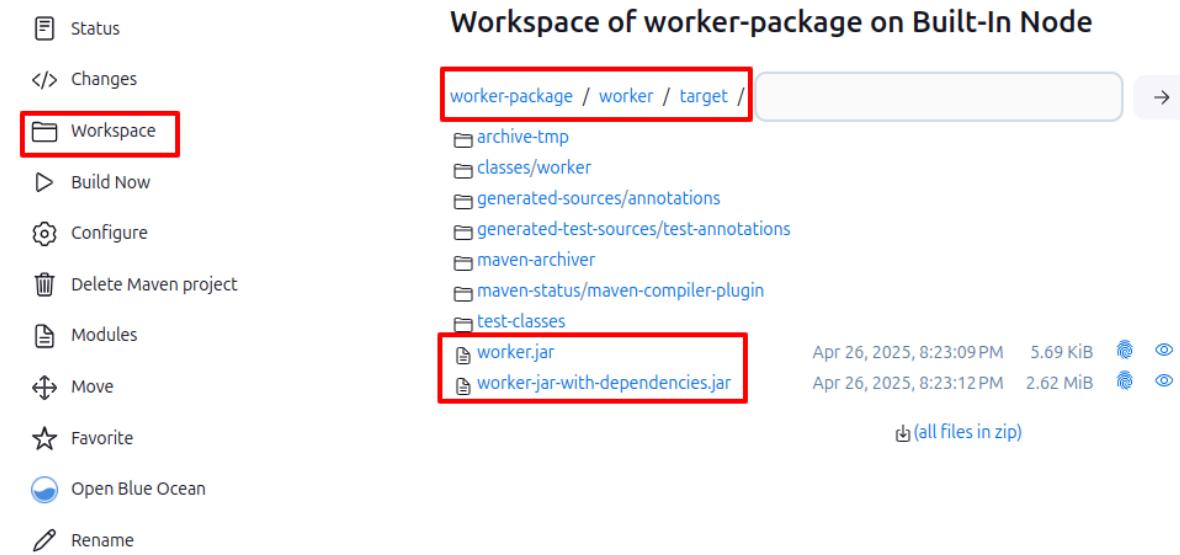


The next job will be **worker-package**. This will compile the application and then generate the **.jar** file. Create a job in the same folder named “worker-package”, copying the **worker-test** or the **worker-build** job.

Update the description to “package worker java app, create jar”. The only change in the configuration is in the Build step.

In the **Build** section for the package job, change the goal to “package -DskipTests”. Save the changes and build.

After the build is successful, you can see the `.jar` files created in the workspace under the `worker/target/` directory. You can verify your workspace by comparing it to the screenshot below.



Click the **Configure** option in the left panel.

From the post-build actions, choose **Archive the artifacts** and provide the path **\*\*/target/\*.jar** to store the .jar file.

### Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

The screenshot shows the Jenkins interface for configuring post-build actions. A red box highlights the 'Archive the artifacts' option under the 'Actions' section. Another red box highlights the 'Files to archive' input field, which contains the pattern '\*\*/target/\*.jar'. Below these, there is an 'Advanced' dropdown and an 'Add post-build action' button. At the bottom are 'Save' and 'Apply' buttons.

Save the changes and build the job.

Once the build is successful, check the project page to find your artifacts there.

Dashboard > instavote > worker-package >

**worker-package**

Changes package worker java app, create jar

Workspace

Build Now

Configure

Delete Maven project

Modules

Move

Favorite

Open Blue Ocean

Rename

Last Successful Artifacts

[worker-jar-with-dependencies.jar](#) 2.62 MiB [view](#)

[workerjar](#) 5.69 KiB [view](#)

Permalinks

- [Last build \(#4\), 7 min 5 sec ago](#)
- [Last stable build \(#4\), 7 min 5 sec ago](#)
- [Last successful build \(#4\), 7 min 5 sec ago](#)
- [Last completed build \(#4\), 7 min 5 sec ago](#)

## Configuring Build Triggers

You can use anything under **Build Triggers** to trigger automatic builds, but for now you are going to use **Poll SCM**.

Go to the `worker-build` job configuration page > **Triggers** and choose **Poll SCMs**. To periodically poll the Git repository, type or paste the following in the **Poll SCM** schedule:

```
H/2 * * * *
```

Once you have made changes, save the job, and go to your job page, where you will find the **Git polling log** in the left panel. Check your polling logs by clicking **Git Polling log**. It may take a minute to run.

The screenshot shows a sidebar with various build-related options: Status, Changes, Workspace, Build Now, Configure, Delete Maven project, Modules, Favorite, and Git Polling Log. The Git Polling Log option is highlighted with a light gray background. The main area is titled "Git Polling Log" and displays the following log output:

```
Started on Nov 12, 2022, 3:40:00 PM
Using strategy: Default
[poll] Last Built Revision: Revision 4f256b896204526d787c804600380f192815f6cc
(refs/remotes/origin/master)
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
> git --version # timeout=10
> git --version # 'git version 2.30.2'
> git ls-remote -h -- https://github.com/eeganlf/LFS261-example-voting-app.git #
timeout=10
Found 2 remote heads on https://github.com/eeganlf/LFS261-example-voting-app.git
[poll] Latest remote head revision on refs/heads/master is:
4f256b896204526d787c804600380f192815f6cc - already built by 2
Done. Took 0.52 sec
No changes
```

Now you will trigger builds remotely. Navigate back to **worker-build > Configuration** and scroll to the **Triggers** section. Select **Trigger builds remotely**. You can type any characters randomly into the token box, but be sure to remember what you put in as you will use it as the token to trigger a build.

A trigger URL is displayed just below where you defined the token. Your custom trigger URL will be similar to the following with your Jenkins instance IP address and port number.

TOKEN\_NAME should be replaced with the text you entered into the **Authentication Token** text box:

```
http://IPADDRESS:8080/job/instavote/job/worker-build/build?token=TOKEN_NAME
```

**Save** the configuration. Enter your URL in the browser and press **Enter**; you *may* see a page asking you to proceed, but you may see no indication that anything has occurred. You can verify that the URL ran the build by checking **Build History** at the bottom of the left pane in your Maven **worker-build** project page and seeing that one more build has occurred:

The screenshot shows the Jenkins interface for the 'worker-build' job under the 'instavote' project. The top navigation bar shows 'Dashboard > instavote > worker-build'. A red box highlights this path. On the left, a sidebar lists various Jenkins management options like Status, Changes, Workspace, Build Now, Configure, Delete Maven project, Modules, Move, Favorite, Open Blue Ocean, and Rename. The main content area is titled 'worker-build' with a green checkmark icon. Below it is a 'Permalinks' section with a list of recent builds. A red box highlights the first item in this list: 'Last build (#2), 1 hr 37 min ago'. Underneath is a 'Builds' section with a 'Filter' input field and a 'Today' dropdown. A red box highlights the first build entry in the 'Today' list: '#3 3:29AM'. Below it are entries for '#2 1:50AM' and '#1 12:19AM'.

Click **Proceed** if the UI appeared and, since you are already authenticated, you should see the job launched automatically. Verify this from the project page.

You can't use this URL by itself outside of a browser session you are already logged into Jenkins with. You will need an API token tied to a Jenkins account for authentication and authorization.

You can trigger the build using a command line interface or programmatically from external code

by providing an API token associated with a Jenkins user.

To create an API token, browse to **Jenkins -> UserAccount** as shown in the following image:

The screenshot shows the Jenkins User Account page for the 'admin' user. The top navigation bar includes a search icon, a shield icon with a red notification dot (containing the number '1'), the 'admin' user icon, and a 'log out' link. The left sidebar contains links for Status, Builds, Favorites, My Views, Account, Appearance, Preferences, Security, Experiments, and Credentials. The main content area is titled 'admin' and shows the Jenkins User ID: admin. A button labeled 'Add description' is visible. The 'Security' link in the sidebar is highlighted with a red box.

Browse to **Security** and **Add new Token**:

The screenshot shows the Jenkins Security page. The top navigation bar includes a search icon, a shield icon with a red notification dot (containing the number '1'), the 'admin' user icon, and a 'log out' link. The left sidebar contains links for Status, Builds, Favorites, My Views, Account, Appearance, Preferences, **Security** (which is highlighted with a red box), Experiments, and Credentials. The main content area is titled 'Security' and has a section for 'API Token'. It displays the message 'There are no registered tokens for this user.' and features a prominent 'Add new Token' button, which is also highlighted with a red box. Below this, there are fields for 'Password' and 'Confirm Password', both containing placeholder dots. At the bottom are 'Save' and 'Apply' buttons.

Name the token “cli-token” and click **Generate**:

**API Token**

Current token(s) ?

There are no registered tokens for this user.

cli-token  X

**API Token**

Current token(s) ?

cli-token	11276bd2394ea3719da6d36b2e4dc78a	<input type="button" value="Copy"/>	<span style="color: red;">Delete</span>
-----------	----------------------------------	-------------------------------------	---

⚠ Copy this token now, because it cannot be recovered in the future.

Copy the API token and put together a remote trigger URL similar to the following where **API\_TOKEN** is the token you just created and **BUILD\_TOKEN** is the token you entered in the **Triggers** step:

```
http://admin:API_TOKEN@IPADDRESS:8080/job/instavote/job/worker-build/build?token=BUILD_TOKEN
```

Test trigger the build using a **curl** command or equivalent:

```
curl
http://admin:API_TOKEN@IPADDRESS:8080/job/instavote/job/worker-build/build?token=BUILD_TOKEN
```

The above instructions demonstrate how to trigger builds remotely.

## Creating a Job Pipeline

Now you will link jobs by defining upstreams and downstreams. You will also create a pipeline view using a plugin.

Follow these steps to set up upstream and downstream jobs.

From the `worker-build` job configuration page, scroll all the way to **Post-build Actions**, click **Add post-build action** and select **Build other projects**.

The screenshot shows the Jenkins configuration interface for a job named 'worker-build'. The left sidebar lists various configuration sections: General, Source Code Management, Triggers, Environment, Pre Steps, Build, Post Steps (which is selected and highlighted with a red box), Build Settings, and Post-build Actions. A modal window is open over the 'Post Steps' section, titled 'Add post-build action'. It contains a list of actions, with 'Build other projects' being the first item and highlighted with a red box. Other listed actions include Aggregate downstream test results, Archive the artifacts, Deploy artifacts to Maven repository, Publish HTML reports, Record fingerprints of files to track usage, Git Publisher, Editable Email Notification, Set GitHub commit status (universal), Set build status on GitHub commit [deprecated], and Delete workspace when build is done. At the bottom of the modal are 'Save' and 'Apply' buttons. The footer of the page shows 'REST API' and 'Jenkins 2.492.3'.

This is where you will define the downstream job. Provide `worker-test` as the project to build and save.

### Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

Build other projects ? X

Projects to build

worker-test,

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Add post-build action ▾

Save Apply

Now you are going to set up the upstream for `worker-package`. Go to the `worker-package` configuration page. From **Triggers**, check the box for **Build after other projects are built**. Put `worker-test` in the **Build after other projects are built** input box.

The screenshot shows the Jenkins configuration interface for the 'worker-package' job. The left sidebar lists various configuration sections: General, Source Code Management, Triggers (which is selected and highlighted with a red box), Environment, Pre Steps, Build, Post Steps, Build Settings, and Post-build Actions. The main panel is titled 'Triggers' and describes setting up automated actions based on specific events. It includes options like 'Build whenever a SNAPSHOT dependency is built', 'Schedule build when some upstream has no successful builds', 'Trigger builds remotely (e.g., from scripts)', and 'Build after other projects are built'. The 'Build after other projects are built' section is expanded, showing 'Projects to watch' with 'worker-test' selected. Below this, there are four radio button options: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', 'Trigger even if the build fails', and 'Always trigger, even if the build is aborted'. At the bottom of the panel are 'Save' and 'Apply' buttons, with 'Save' being highlighted with a red box.

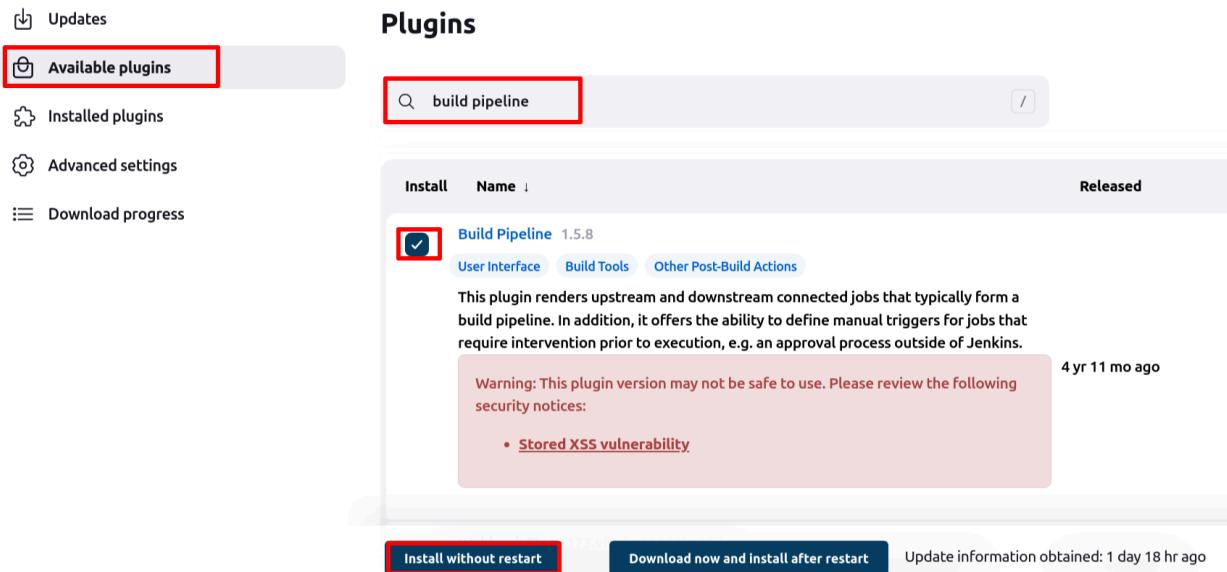
This defines the upstream for **worker-package**.

Once upstreams and downstreams are defined, run **worker-build** and it will automatically run **worker-test** and **worker-package**.

## Set Up the Pipeline View

Next, you are going to set up a pipeline view for this build job.

Begin by installing the Build Pipeline plugin from the **Manage Jenkins > Manage Plugins** page. Type *build pipeline* in the search box, select the **Build Pipeline** plugin and click **Install without restart**.



**NOTE:** You may see a vulnerability warning while installing the Build Pipeline plugin. It's not recommended that you install this plugin in a production environment. You are using this plugin only during this lab to aid your understanding of creating pipelines. Starting with the next chapter, you will use the `Jenkinsfile`, which does not need this plugin to provide you with a pipeline view. `Jenkinsfiles` are used in a real production environment.

Click on the **instavote** folder and you will now notice a + sign under the title. Click on it:

S	W	Name ↓	Last Success	Last Failure
		worker-build	21 hr #1	N/A

Select **Build Pipeline View** and provide a name for it, e.g. “worker-pipe-view”, then click **Create**:

Dashboard > instavote >

- Status
- Configure
- + New Item
- Delete Folder
- People
- Build History
- Project Relationship

View name

Type

Build Pipeline View

Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

From the configuration page, select the first job in the pipeline, **worker-build**, and select the number of displayed builds as 5, then save it.

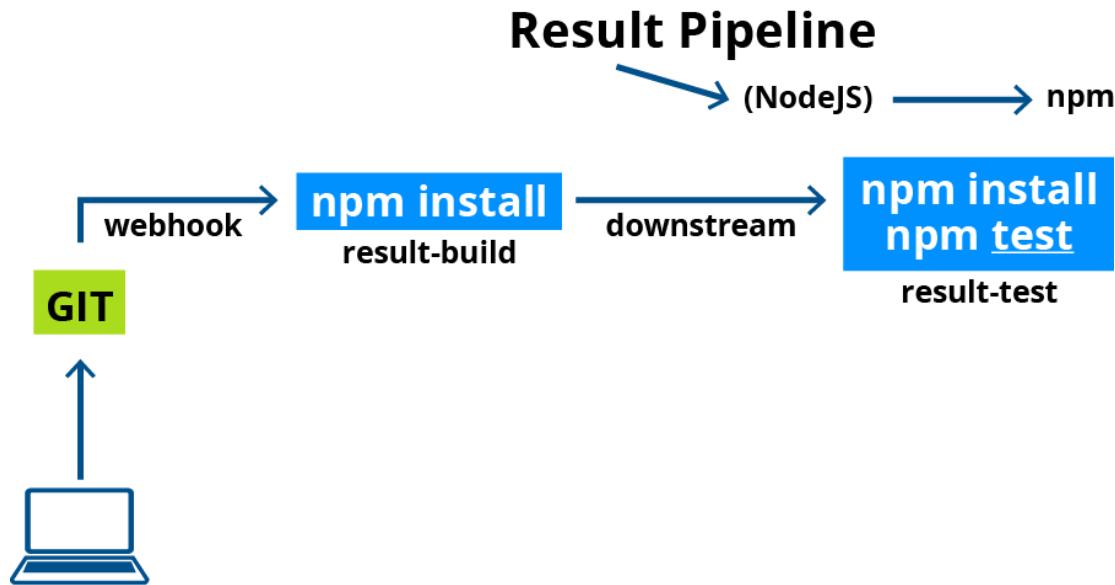
The screenshot shows the Jenkins Pipeline Configuration page for a view named 'worker-pipe-view'. The 'Upstream / downstream config' section has 'Instavote > worker-build' selected. In the 'Display Options' section, the 'No Of Displayed Builds' input field is set to '3'. Both the '3' input field and the 'Save' button are highlighted with red boxes.

Once you complete, you will see the pipeline of your job. Green is successful, red is failed, blue is to do, yellow is in progress.

You have just created a pipeline for a Java project.

## Exercise: Create a Pipeline for the Node.js Application

Now that you are familiar with creating CI pipelines, try creating one for the Node.js `result` application with `npm`.



### Steps:

- Install **NodeJS** plugin for Jenkins.
- Configure **Global Tools** with a NodeJS installation with version 22.4.0.
- Create two jobs `result-build` and `result-test`, this time as freestyle projects.
- Define the Git repository you have forked in the **Source Code Management** section.
- Define the build trigger as **PollSCM** with an interval of 2 mins.
- In the build environment, choose to add NodeJS configurations with the version you have selected in **Global Tools**.
- Select the **Execute Shell** option from build and provide the commands to build the Node application from the `result` subdirectory. For example:

```

cd result
npm install
npm test
    
```

## Summary

In this lab, we set up Jenkins using Docker Compose. We forked the `instavote` app so that we could work with our version of the app. We then used Jenkins to set up an integration pipeline for our `instavote` app on GitHub. Finally, we added a visual representation of the build pipeline.



## Lab 4S. Creating A Pipeline for the NodeJS Result Application (Solution)

Install the **NodeJS** plugin for Jenkins.

A screenshot of the Jenkins 'Manage Jenkins &gt; Plugins' page. The search bar at the top contains 'nodejs'. In the main list, the 'NodeJS 1.6.4' plugin is shown as installed. The 'Available plugins' tab is highlighted with a red box. The 'Install' button for the NodeJS plugin is also highlighted with a red box.

From **Jenkins > Manage Jenkins > Tools > NodeJS Installations**, add a NodeJS installation with version 22.4.0.

A screenshot of the Jenkins 'Tools > NodeJS Installations' configuration page. A red box highlights the 'NodeJS installations' section. Below it, a red box highlights the 'Add NodeJS' button. At the bottom, there are 'Save' and 'Apply' buttons.

Name the installation exactly **NodeJS 22.4.0** as the name will be referred to later in the pipeline.

**Name**

NodeJS 22.4.0

Install automatically

**Install from nodejs.org**

**Version**

NodeJS 22.4.0

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail

Force 32bit architecture

**Global npm packages to install**

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax 'packageName@version'

Create a new job inside the **instavote** folder called **result-build**, this time as a freestyle project.

Dashboard > instavote > New Item

### New Item

Enter an item name

result-build

Select an item type

**Freestyle project**  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

**Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Paste in the **example-voting-app** Git repository that you forked in the **Source Code Management** section.

The screenshot shows the Jenkins configuration interface for a build job named 'instavote'. The 'Source Code Management' section is selected, indicated by a red box around the 'Source Code Management' tab. Under the 'Git' tab, the 'Repository URL' field contains the value `https://github.com/okapetanios/example-voting-app`, also highlighted with a red box. The 'Branches to build' field has the value `*/master`. At the bottom of the screen, there are 'Save' and 'Apply' buttons.

Select **Poll SCM** with an interval of 2 mins by pasting `H/2 * * * *` into the **Schedule** input box.

The screenshot shows the Jenkins configuration interface for a build job named 'result-build'. In the 'Triggers' section, the 'Poll SCM' option is selected, and the 'Schedule' field is set to 'H/2 \*\*\* \*'. This schedule means the build will run every two hours. The 'Ignore post-commit hooks' checkbox is not checked. Other trigger options like 'Trigger builds remotely' and 'Build after other projects are built' are available but not selected.

In the **Environment** choose to provide NodeJS configurations with the version you have selected in the **Global Tool Configuration**.

The screenshot shows the Jenkins configuration interface for the same build job. In the 'Environment' section, the 'Provide Node & npm bin/ Folder to PATH' checkbox is checked. Below it, the 'NodeJS Installation' dropdown is set to 'NodeJS 22.4.0'. Other environment settings like 'npmrc file' (set to 'use system default') and 'Cache location' (set to 'Default (-.npm or %APP\_DATA%\npm-cache)') are also visible. The 'Build Steps' section at the bottom is currently empty.

Select the **Execute shell** option from **Build steps > Add build step** and paste or type the following into the box:

```
cd result
```

```
npm install
```

```
npm ls
```

The screenshot shows the Jenkins configuration page for a project named 'result-build'. The 'Build Steps' section is highlighted with a red box around the 'Execute shell' step. Inside this step, the command `cd result` followed by `npm install` and `npm ls` is entered. The 'Configure' sidebar on the left lists General, Source Code Management, Triggers, Environment, Build Steps (which is selected and highlighted), and Post-build Actions.

Save the project and build it.

Create another project with the name **result-test**, this time copying from **result-build**.

Change the build trigger to run after `result-build` is run.

Dashboard > instavote > result-test > Configuration

**Configure**

**Triggers**

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- Trigger builds remotely (e.g., from scripts) ?
- Build after other projects are built ?

Projects to watch

result-build,

- Trigger only if build is stable
- Trigger even if the build is unstable
- Trigger even if the build fails
- Always trigger, even if the build is aborted

- Build periodically ?
- GitHub hook trigger for GITScm polling ?
- Poll SCM ?

Schedule ?

H/2 \* \* \* \*

Would last have run at Saturday, May 3, 2025, 8:38:30PM Coordinated Universal Time; would next run at Saturday, May 3, 2025, 8:40:30PM Coordinated Universal Time.

Ignore post-commit hooks ?

**Save** **Apply**

Update the command to run `npm test` instead of `npm ls`.

### Build Steps

≡ Execute shell ? ×

Command

See [the list of available environment variables](#)

```
cd result
npm install
npm test
```

**Advanced...**

**Add build step ▾**

Create a new pipeline view, this time by choosing **result-build** as the first job to run.

After **result-build** runs, wait a few moments and refresh the page; you should see both jobs turn green in the pipeline as follows:



This completes the CI pipeline for the **result** app created with NodeJS.



## Lab 4.1. Setting Up Continuous Integration with Jenkins (Additional Topics)

### Optional: Integrating GitHub with Jenkins

***WARNING:*** This will only work if your Jenkins server is available publicly and can be accessed from GitHub. If you are running Jenkins on your local machine (e.g. laptop) with a private IP space, this will **not** work unless you use some method to expose your local machine to the public internet. This is beyond the scope of this course, but you can look into [ngrok](#), [boring proxy](#), or [localtunnel](#) as possible solutions. Be aware that this will have security implications.

By the end of this exercise, you should be able to:

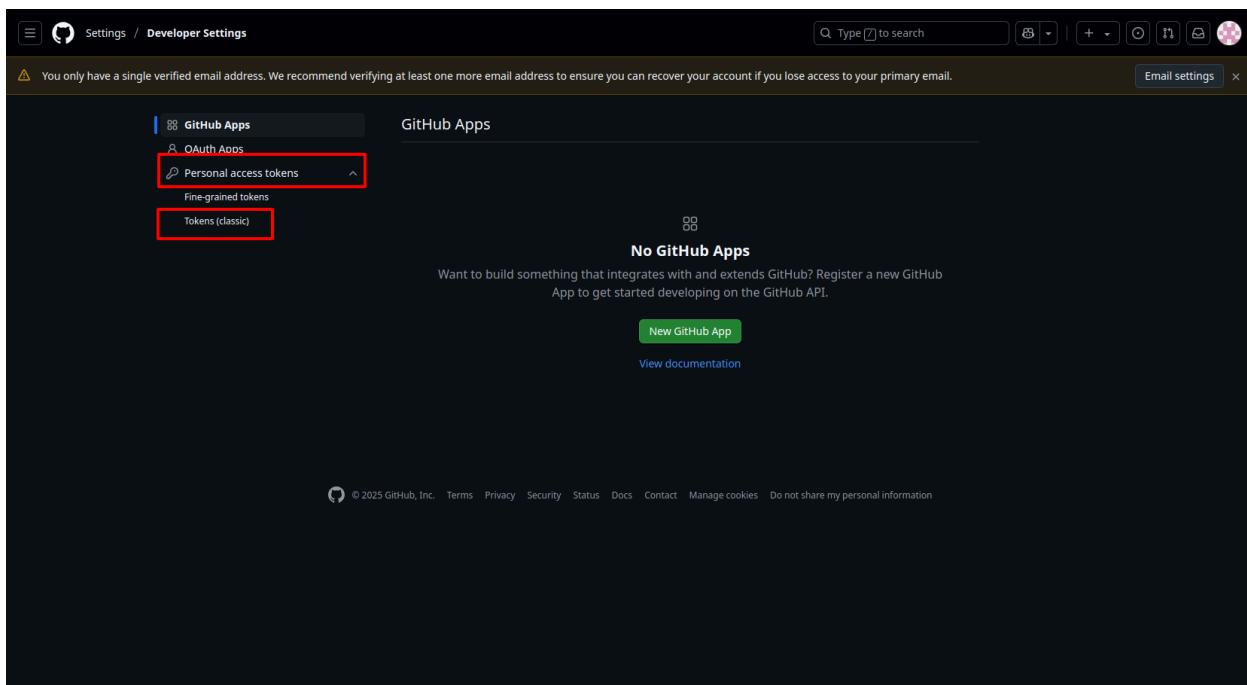
- Integrate GitHub with Jenkins
- Trigger builds automatically using GitHub webhooks

### Setting Up Build Triggers Using GitHub Webhooks

You can trigger your job when there is a commit change in the master branch on GitHub. Webhooks will trigger your job from the GitHub side and you can send the status back to GitHub, so you don't need polling from Jenkins.

Follow these steps to set up the Webhook.

To set up a GitHub token, go to the user page and select **Settings > Developer settings > Personal access tokens > Tokens (classic)**.



In the **Personal access token** page select **Generate new token > Generate new token (classic)**, add a note, check *all* boxes, and generate the token. Once you generate your token, save it somewhere immediately because you can't view it again.

[Settings](#) / Developer settings

[GitHub Apps](#)

[OAuth Apps](#)

[Personal access tokens](#)

### New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

**Note**

What's this token for?

**Expiration \***

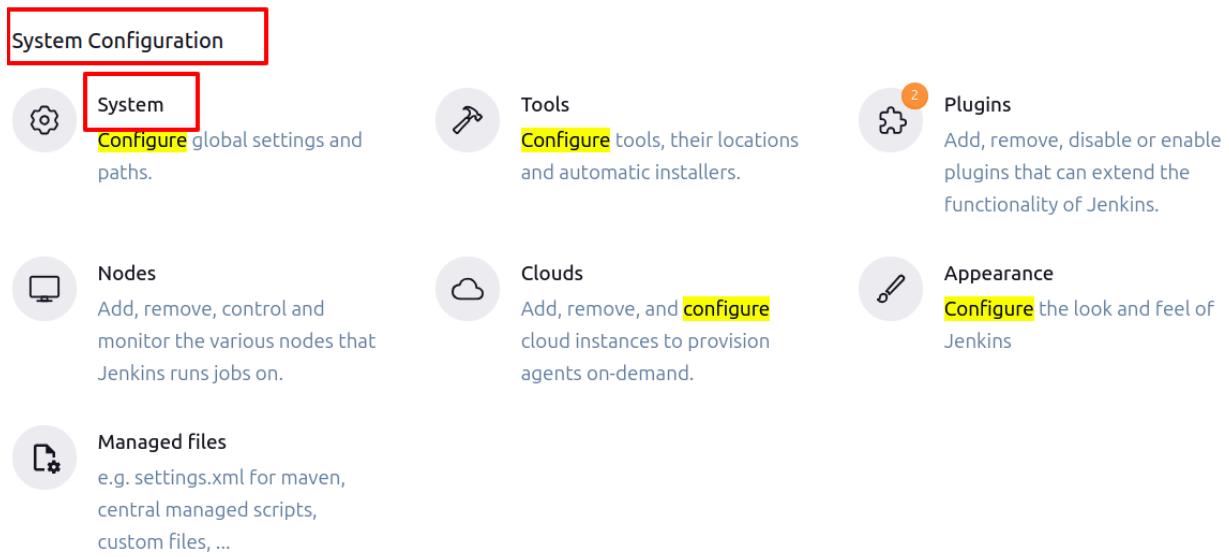
 The token will expire on Fri, Aug 5 2022

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input type="checkbox"/> <b>repo:status</b>	Access commit status
<input type="checkbox"/> <b>repo_deployment</b>	Access deployment status
<input type="checkbox"/> <b>public_repo</b>	Access public repositories
<input type="checkbox"/> <b>repo:invite</b>	Access repository invitations
<input type="checkbox"/> <b>security_events</b>	Read and write security events
<input checked="" type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows

Now go to **Jenkins page > Manage Jenkins > System Configuration > System**.



Scroll to **GitHub** and click **Add GitHub Server**. Provide any name for the server. This does not matter. Click **Add** under **Credentials** which brings up the following:

The screenshot shows the Jenkins Credentials creation form. The fields are as follows:

- Kind**: Secret text (highlighted with a red box)
- Scope**: Global (Jenkins, nodes, items, all child items, etc.)
- Secret**: (redacted)
- ID**: github-secret-token (highlighted with a red box)
- Description**: secret token for github (highlighted with a red box)
- Buttons**: Add (highlighted with a red box) and Cancel

The secret is your GitHub access token that you generated. Provide an ID description. Save it. Choose “secret token for github” under credentials and check the **Manage hooks** box.

Dashboard > Manage Jenkins > System >

### GitHub

GitHub Servers ?

**GitHub Server ?**

Name ? GitHub Server 1

API URL ? https://api.github.com

Credentials ? Secret token for Github

+ Add

Manage hooks

Advanced ▾

Test connection

Add GitHub Server ▾

**Save** **Apply**

Dashboard > Manage Jenkins > System >

### Manage hooks

Advanced ▾

Add GitHub Server ▾

Advanced ▾ Edited

Override Hook URL

Specify another hook URL for GitHub configuration

http://34.9.158.130:8080/github-webhook/  

Re-register hooks for all jobs

Shared secrets

Add shared secret

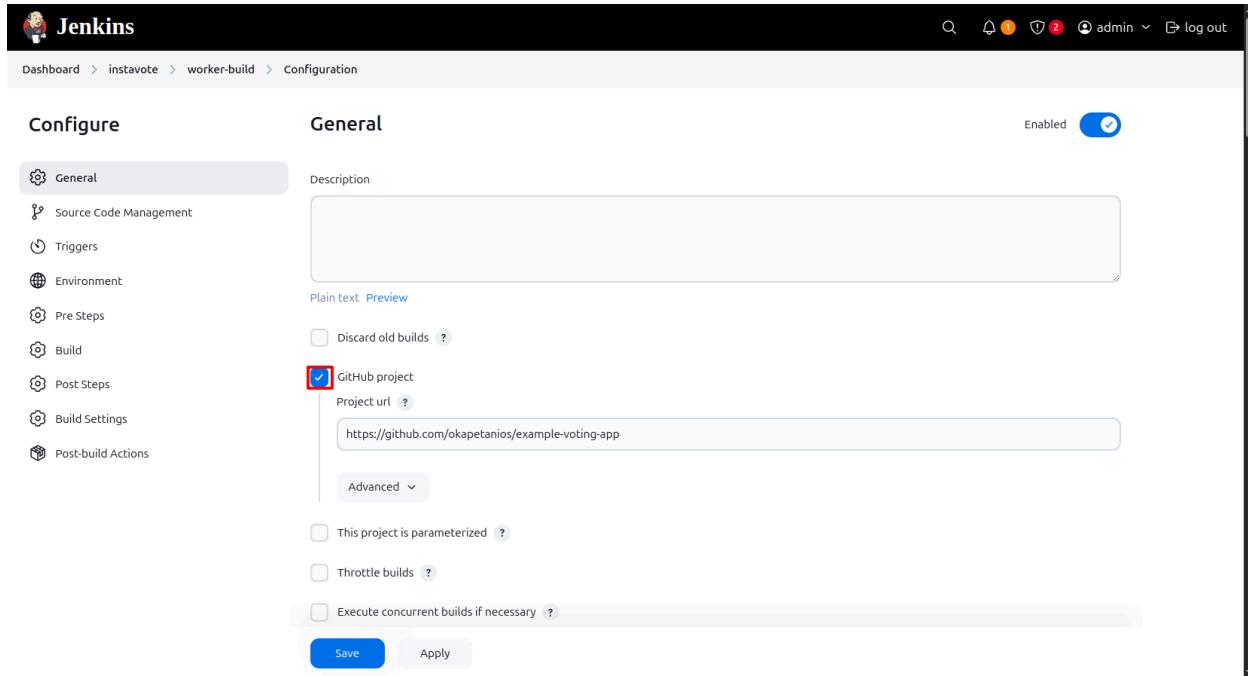
Additional actions ?

Manage additional GitHub actions ▾

**Save** **Apply**

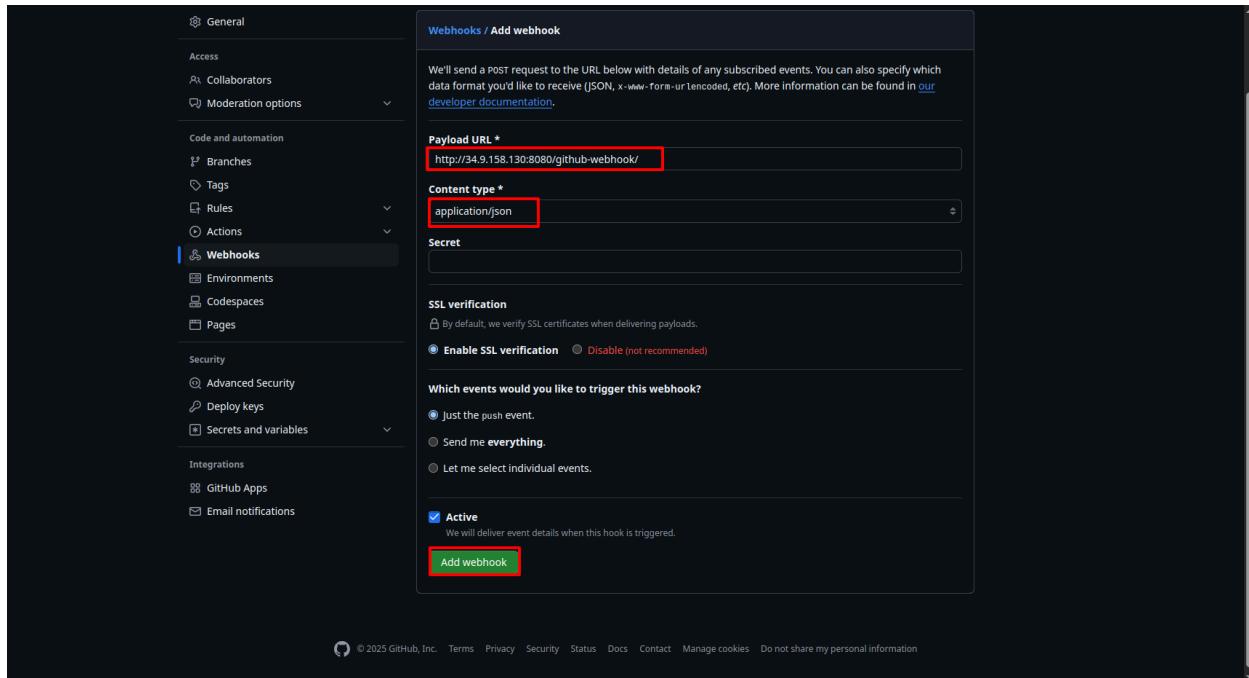
Copy the web hook address and save it for later. We will need to give this to GitHub in a few steps.

Go to your **worker-build** configuration page, choose “github project” and provide the **example-voting-app** repository URL.



Remove **Poll SCM** and choose **GitHub hook trigger** for Git SCM polling, save the changes.

Go to the GitHub repository and choose **Settings > Webhooks**. Add “Payload URL” and “Content type” as “application/json”. Save it (refer to the example image given below).



Next, go to the Git repository, add some file in your repository and commit the changes.

Once you make the commit in your repository, it will automatically trigger your build. You can see it by using the build pipeline you created earlier.

This is how you integrate GitHub with Jenkins.

## Optional: Adding Jenkins Status Badges to GitHub

**WARNING:** This only works if the Jenkins server is accessible from GitHub.

By the end of this exercise, you should be able to:

- Set up a two-way communication between Jenkins and GitHub
- Send the status of the build from Jenkins to GitHub and display it as a badge

First, allow anonymous read access so GitHub can access the necessary information.

**Dashboard > Manage Jenkins > Security > Authentication > Allow anonymous read access:**

The screenshot shows the Jenkins 'Security' configuration page. At the top, the navigation path 'Dashboard > Manage Jenkins > Security' is highlighted with a red box. The page is divided into several sections:

- Authentication**: Includes an option to 'Disable "Keep me signed in"' with a checkbox.
- Security Realm**: Set to 'Jenkins' own user database'. Below it is a checkbox for 'Allow users to sign up'.
- Authorization**: Set to 'Logged-in users can do anything'. Below it is a checkbox for 'Allow anonymous read access' which is checked.
- Markup Formatter**: Set to 'Plain text'. A note below says: 'Shows descriptions mostly as written. HTML unsafe characters like < and & are escaped to their respective character entities, and line breaks are converted to their HTML'.

At the bottom are 'Save' and 'Apply' buttons.

You need to install the **Embeddable Build Status** plugin from *Manage Jenkins > Plugins > Available plugins*. Once the installation is done, you can see **Embeddable build status** on every job page.

Follow the next steps to add the Jenkins status badge to GitHub.

Go to the `worker-build` job page, select **Embeddable Build Status** from the menu on the left. On the **Embeddable Build Status** page, scroll to **Links** and under **Markdown** copy the **unprotected** link. Paste it in your GitHub repository `README.md` file which you have created in the previous lab and commit the changes.

Once you commit the changes, you should see the build status being listed as passing on your `README.md` file.



---

Go to the `worker-build` configuration page. Under the **Build** section change the root file to `pom.xml` instead of `worker/pom.xml` and save the changes.

Build your `worker-build` job; it will fail your build. Now go to the GitHub repository page. There you can see the build's failed status on the `README.md`.

Correct the `worker-build` by switching `pom.xml` back to `worker/pom.xml`.

If you wish, you can add the embeddable build status unprotected markdown link of `worker-test` and `worker-package` to your GitHub repository `README.md` file as well.

This is how you add the Jenkins status badge to GitHub.

## Optional: Configuring Job Status with Commit Messages

By the end of this exercise, you should be able to:

- Add build and test job statuses to GitHub commit messages

Follow these steps to configure the job status.

Go to your `worker-build` job configuration page, under **Post-build Actions**, **Add post-build action** and choose “Set GitHub commit status (universal)” and set the **What > Status Result** to *One of default messages and statuses*. Save the changes.

**What:**

Commit context:

From GitHub property with fallback to job name



Status result:

One of default messages and statuses



Status backref:

Backref to the build



**Save**

**Apply**

Go to the `worker-test` configuration page and paste the GitHub project URL into the **Source Code Management** section. Add the post-build action “Set GitHub commit status (universal)” in the same way that you did for the `worker-build` job. Be sure **What > Status result** is set to *One of default messages and statuses*.

Now go to your `worker-test` job page. There, select the **Embeddable Build Status**. In the **Embeddable status** page, copy the markdown unprotected link under **Links** and paste it on your GitHub repository `README.md` file. Add `subject=UnitTest` to your `worker-test` link as follows:

The unprotected markdown link will look similar to the following:

```
[![Build
Status] (http://IPADDRESS:8080/buildStatus/icon?job=instavote%2Fworker-test) ] (h
tp://IPADDRESS:8080/job/instavote/job/worker-test/)
```

You need to add `subject=UnitTest` like so:

```
[![Build
Status] (http://IPADDRESS:8080/buildStatus/icon?job=instavote%2Fworker-
test&subject=UnitTest) ] (http://IPADDRESS:8080/job/instavote/job/worker-
```

-test/)

Once you commit the changes, it will automatically build the triggers. You can see the builds in the pipeline view and the status will be updated on GitHub.

If you check your commit messages, it shows them as check-marked. You can see all the current build statuses with build number on the git commit message. If you click **Details** in the commit message, it will take you to the Jenkins page.

This is how you configure the job status and `git commit` messages.



## Lab 5. Enforcing Workflows, Pipeline as Code

By the end of this lab exercise, you should be able to:

- Define branch protection rules on GitHub
- Enforce code reviews via pull requests
- Write a pipeline as code
- Learn the syntax to write declarative Jenkins pipelines
- Create Jenkinsfiles for Java applications
- Launch multi-branch pipelines upon commits to a remote repository

### Enforcing Workflow with Branch Policies

#### Resources:

- [Web: Guide to Trunk Based Development](#)
- [Video: Trunk Based Development Explained](#)

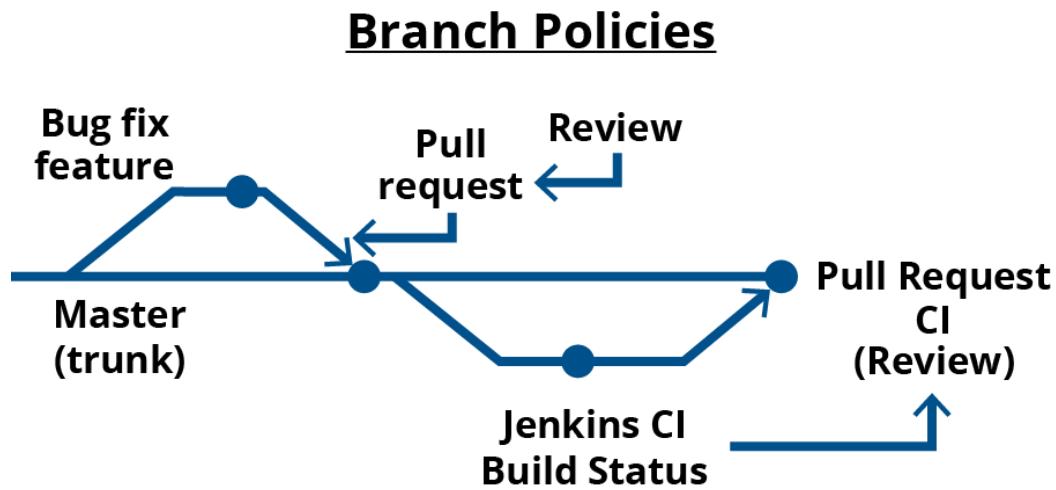
Let's create branch policies based on the trunk-based development workflow. With trunk-based development:

- The master branch is the trunk/main line of code, which is locked (no direct check-ins to master are allowed).
- Every bug fix gets its own branch.
- Every feature gets its own branch too.
- Feature branches are typically short-lived.
- Merges to master require pull requests.
- Further policies could be added to enforce code review, as well as integrate with Jenkins CI to run per branch pipeline.

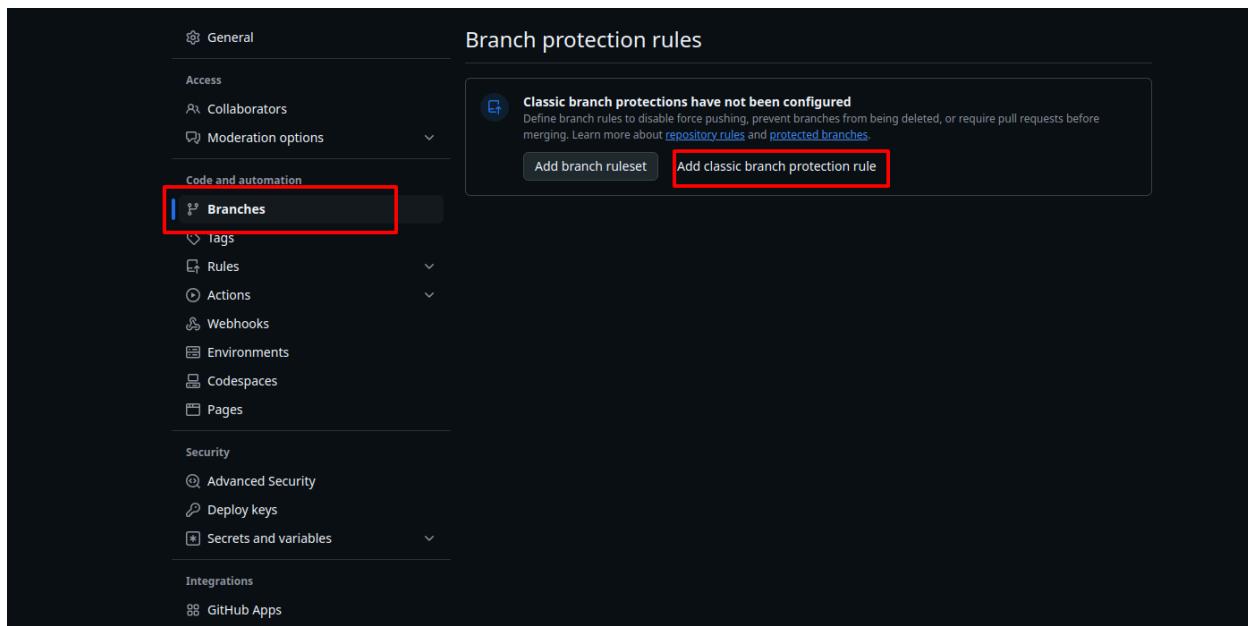
Make sure you have cloned your own fork of the `example-voting-app` repository locally before you proceed. Make sure you are currently inside the `example-voting-app` directory.

**NOTE:** If you do not have a collaborator, create another account on GitHub, add that account as a collaborator on the [example-voting-app](#) repository, and add it as a reviewer or skip to the [Writing Pipeline as Code](#) section.

This is how the process looks:



To define the branch policies, go to **Settings** for your [example-voting-app](#) repository on GitHub and select the **Branches** option.



From the **Branch Protection Rules** section, click on the **Add branch protection rule** button. Add the following set of rules:

- Require pull request reviews before merging
- Dismiss stale pull request approvals when new commits are pushed
- Require status checks to pass before merging
- Do not allow bypassing the above settings

The following screenshots depict the branch protection rules:

Make the name the same as the branch, in this case **master**, that you want to add rules to.

**Branch name pattern \***

**Applies to 1 branch**

**Protect matching branches**

**Require a pull request before merging**  
When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

**Require approvals**  
When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.  
Required number of approvals before merging: 1 ▾

**Dismiss stale pull request approvals when new commits are pushed**  
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

**Require review from Code Owners**  
Require an approved review in pull requests including files with a designated code owner.

**Restrict who can dismiss pull request reviews**  
Specify people, teams, or apps allowed to dismiss pull request reviews.

**Allow specified actors to bypass required pull requests**  
Specify people, teams, or apps who are allowed to bypass required pull requests.

**Require approval of the most recent reviewable push**  
Whether the most recent reviewable push must be approved by someone other than the person who pushed it.

**Require status checks to pass before merging**  
Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

**Require conversation resolution before merging**  
When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule.  
[Learn more](#).

**Require signed commits**  
Commits pushed to matching branches must have verified signatures.

**Require linear history**  
Prevent merge commits from being pushed to matching branches.

**Require deployments to succeed before merging**  
Choose which environments must be successfully deployed to before branches can be merged into a branch that matches this rule.

**Lock branch**  
Branch is read-only. Users cannot push to the branch.

**Do not allow bypassing the above settings**  
The above settings will apply to administrators and custom roles with the "bypass branch protections" permission.

**Restrict who can push to matching branches**  
Specify people, teams, or apps allowed to push to matching branches. Required status checks will still prevent these people, teams, and apps from merging if the checks fail.

**Rules applied to everyone including administrators**

**Allow force pushes**  
Permit force pushes for all users with push access.

**Allow deletions**  
Allow users with push access to delete matching branches.

**Save changes**

## Testing Your New Branch Protection Rules

To test the branch protection rules, from the local repository, make some changes to any file, e.g. `README.md`. Try adding it to Git and commit those changes:

```
git diff  
git commit -am "test"  
git push origin master
```

When you try to push the changes to a remote, it throws an error “*master protected branch hook declined*”. That’s because you can’t directly push to master anymore.

You can undo the changes by resetting the commit:

```
git log  
git reset --hard HEAD~1
```

Once you do a hard reset, it will roll back the changes you have made to the commit history and also to the local workspace.

## Code Reviews with Pull Requests

Next, you will learn how to make changes in the master branch by using code reviews and pull requests. Update the existing `README.md` file for the project and incorporate those changes to the trunk i.e. master branch.

Create a branch `readme` using the following command:

```
git checkout -b readme
```

Running this command will create the `readme` branch, and then switch to it. Update the `README.md` file by adding a description of the application.

For example,

```
filename: README.md
```

```
Example Voting App  
=====
```

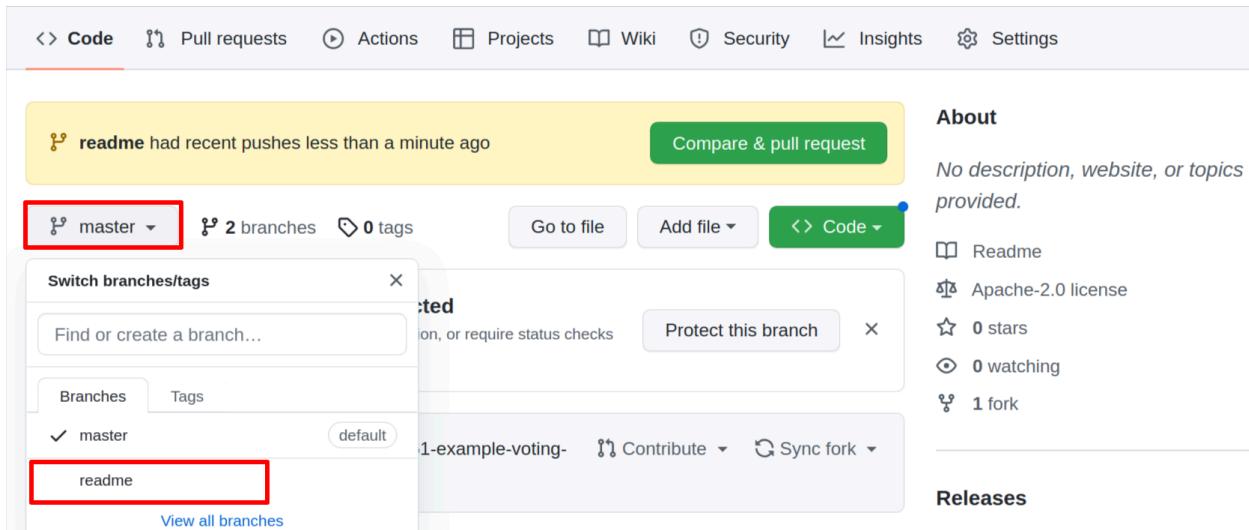
```
This is a sample voting app.
```

Save and commit the changes you have made:

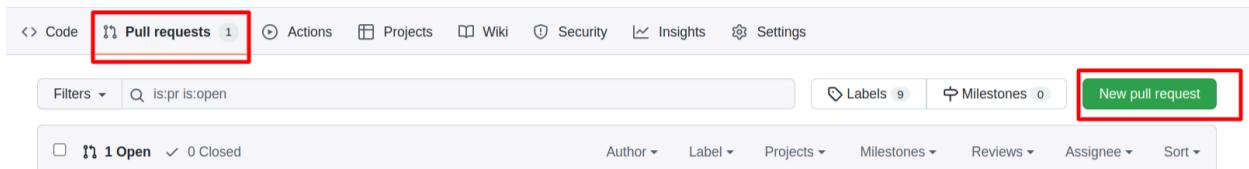
```
git status
git commit -am "added application info"
git push origin readme
```

This pushes the changes to the `readme` branch of the repo on GitHub.

Go to the GitHub repository and switch to the branch `readme`. You can do this by clicking the dropdown menu that has **master** currently selected:



Select the **Pull requests** tab next to the **Code** tab across the top of the repository:



You should see **New pull request** to the right. The pull request is to allow you to merge the changes from the feature branch on GitHub to the master branch on GitHub.

Select **New pull request** and choose your repository for the **base repository** and choose **master** for the **base**. Select your repository from the **head repository** dropdown and select the `readme` branch in the **Compare** dropdown. Click **Create pull request**.

## Comparing changes

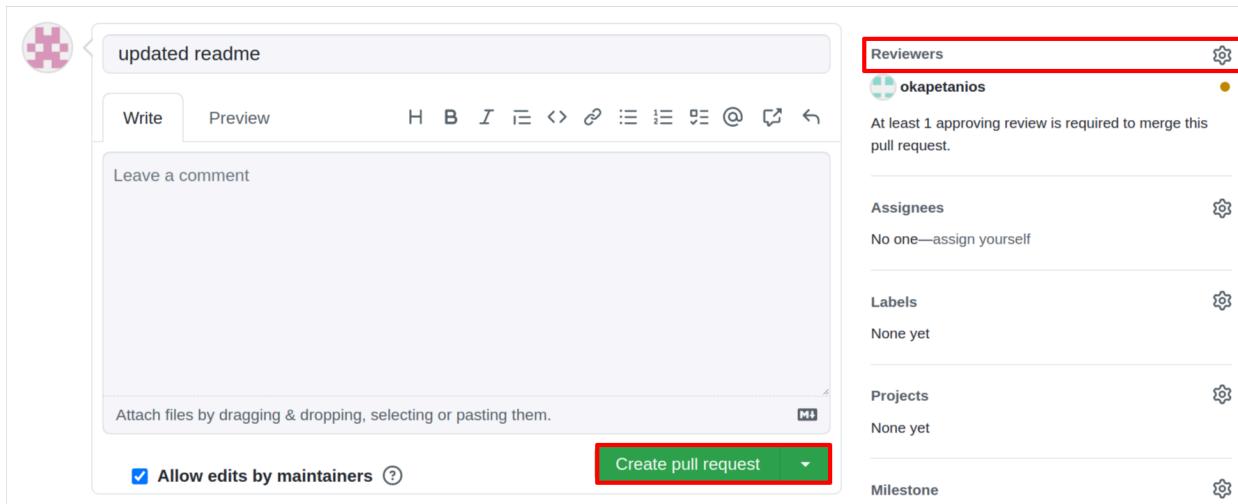
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

The screenshot shows the GitHub interface for comparing branches. At the top, there are dropdown menus for 'base: master' and 'compare: readme'. A green checkmark indicates that the branches are 'Able to merge'. Below this, a message encourages users to 'Discuss and review the changes in this comparison with others' and provides a link to 'Learn about pull requests'. A prominent green button at the bottom right is labeled 'Create pull request', which is also highlighted with a red box.

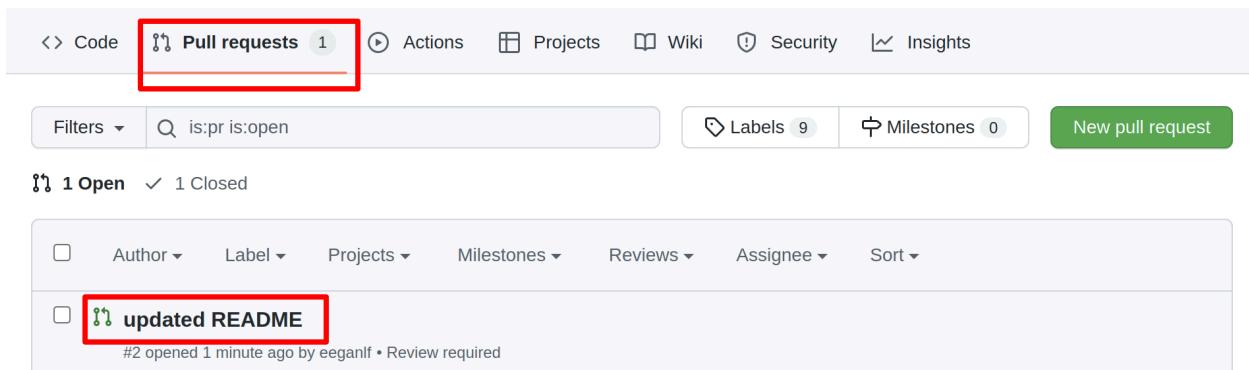
Enter the title and leave a comment if you'd like, then click **Create pull request**:

The screenshot shows the GitHub interface for creating a pull request. It features a 'Pull' button in the top left, a 'Write' tab selected, and a 'Leave a comment' text area. The 'Leave a comment' area is highlighted with a large red box. Below it is a note: 'Attach files by dragging & dropping, selecting or pasting them.' At the bottom, there is a checkbox for 'Allow edits by maintainers' and a green 'Create pull request' button, which is also highlighted with a red box.

Add a reviewer: Creating a pull request does not allow you to merge immediately. It will require a code reviewer. This is mandated as part of the branch protection rules you created. Add at least one of your collaborators as a reviewer with the **Reviewers** gear icon:



Approve a pull request as the reviewer. With the reviewer's account, visit the repository page, go to pull requests, and click on the pull request:



Review the changes, leave a comment if you wish, and approve the request by checking **Approve** and clicking **Submit Review**:

updated README #2

[Edit](#) [Code ▾](#)

[Open](#) eeganlf wants to merge 1 commit into [master](#) from [readme](#) [diff](#)

Conversation 0 Commits 1 Checks 0 Files changed 2 +45 -0 [View changes](#)

Changes from all commits ▾ File filter ▾ Conversations ▾ [Review changes ▾](#)

Filter changed files [X](#)

README.md [diff](#)

worker [diff](#)

Jenkinsfile [diff](#)

57 57  
58 58  
59 59  
60 60  
61 61  
62 62  
63 63  
64 64

41 41

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Comment  
Submit general feedback without explicit approval.

Approve  
Submit feedback and approve merging these changes.

Request changes  
Submit feedback that must be addressed before merging.

[Submit review](#)

```
4 +      image 'maven:3.6.1-jdk-8-slim'  
5 +      args '-v $HOME/.m2:/root/.m2'
```

After approval, merge the pull request to master either from the reviewer's account or the requester's account. Go to **Pull requests > Conversation** and click **Merge pull request** as follows:

The screenshot shows a GitHub pull request conversation for a file named "updated README #2". The pull request is from the user "eeganlf" into the "master" branch. The conversation tab is selected, indicated by a red box. The message from "eeganlf" states: "No description provided." Below it, a commit from "eeganlf" adds blank lines to the README. A review from "okapetanios" is shown, with a green checkmark indicating it is approved. The right sidebar provides merge options: "Changes approved" (highlighted with a red box), "Merge pull request", and "Unsubscribe". Other settings like Milestone, Notifications, and Participants are also visible.

Delete the `readme` branch after the merge.

## Writing Pipeline as Code

### Resources:

- [Declarative Pipeline Syntax](#)
- [Declarative Pipeline Steps](#)

Some of the important instructions are:

- 
- Pipeline
  - Agent
  - Tools
  - Stages, Stage, Steps
  - Post

## Jenkinsfile for a Java Application

In this section, you will define pipeline as code for a Java worker app which uses Maven as a build tool.

Create a new `feature/workerpipe` branch by using the following commands:

```
git checkout -b feature/workerpipe
```

```
git branch
```

Now you are going to write a Jenkinsfile for the `worker` application.

Create a Jenkinsfile inside the `worker` directory of your code. Before you start writing the code, check your Maven version in your Jenkins: **Manage Jenkins > System Configuration > Tools > Maven Installations** and note down the exact name in the configuration (e.g. Maven 3.9.8).

```
pipeline {
    agent any

    tools{
        maven 'maven 3.9.8'

    }

    stages{
        stage("build"){
            steps{
                echo 'Compiling worker app'
                dir('worker'){
                    sh 'mvn compile'
                }
            }
        }
        stage("test"){
            steps{
                echo 'Running Unit Tests on worker app'
            }
        }
    }
}
```

---

```

stage("package") {
    steps{
        echo 'Packaging worker app'
    }
}

post{
    always{
        echo 'Building multibranch pipeline for worker is completed..'
    }
}
}

```

Once you create the Jenkinsfile, commit the file and push the changes to **feature/workerpipe**:

```

git status

git add worker/Jenkinsfile

git commit -am "added Jenkinsfile for worker with build job"

git push origin feature/workerpipe

```

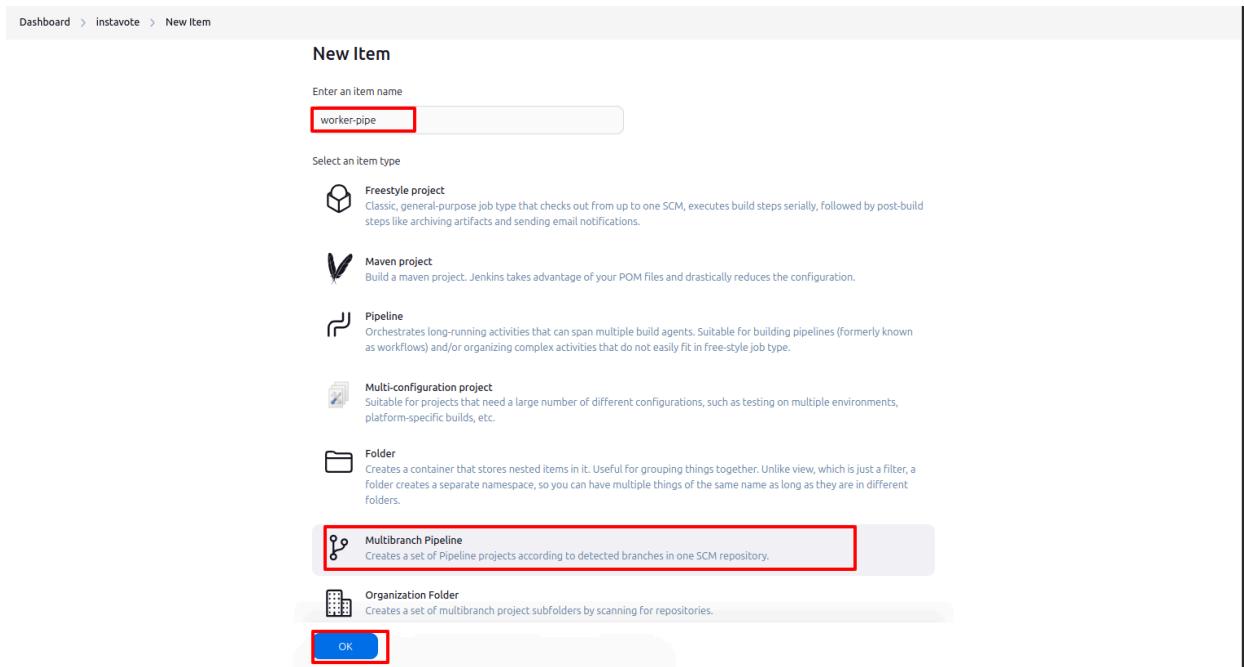
## Multi-Branch Pipeline Project

In this section, you are going to create and execute a multi-branch pipeline which:

- Scans your repository automatically for a Jenkinsfile
- Sets up the pipeline jobs for the branches which contain the above pipeline script
- Starts executing the CI jobs

Refer to the following steps to create a multi-branch pipeline.

Create a new job `worker-pipe` in your `instavote` directory; select “Multibranch Pipeline” as the project type.



In the job configuration page, under the branch sources choose “GitHub”, add your account credentials, and choose the project repository. Refer to the images below for exact steps.

In a separate browser, navigate to **GitHub > Settings** and select **Developer Settings**.

The screenshot shows a GitHub repository page for 'eeganlf / LFS261-example-voting-app'. The user 'eeganlf' is signed in, and their profile picture is highlighted with a red box. The repository is public and has been forked from 'lftesting/LFS261-example-voting-app'. The 'Code' tab is selected. A modal message 'Your master branch isn't protected' is displayed, with 'Protect this branch' and 'Dismiss' buttons. The right sidebar shows navigation links like 'Your profile', 'Your repositories', and 'Settings', with 'Settings' also highlighted with a red box. Below the sidebar, the 'Developer settings' link is highlighted with a red box.

Signed in as **eeganlf**

Search or jump to... Pulls Issues Codespaces Marketplace Explore

**eeganlf / LFS261-example-voting-app** Public

forked from [lftesting/LFS261-example-voting-app](#)

Pin Watch Fork 1

Code Pull requests 1 Actions Projects Wiki Security Insights

master Go to file Add file Code About

Your master branch isn't protected

Protect this branch Dismiss

No description provided.

Upgrade Try Enterprise Feature preview Help Settings

Readme Apache 0 stars 0 watches

This branch is up to date with lftesting/LFS261...

Contribute Sync fork

**Security**

Code security and analysis

**Integrations**

Applications Scheduled reminders

**Archives**

Security log Sponsorship log

**<> Developer settings**

Generate a *personal access token* with access to **repo** enabled.

The screenshot shows the GitHub 'Personal access tokens (classic)' page. On the left, there's a sidebar with 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens' (which is expanded). Under 'Personal access tokens', there are 'Fine-grained tokens' (Beta) and 'Tokens (classic)' (highlighted with a red box). On the right, the main area shows 'Tokens you have generated' with one entry: 'tokenlf261 — admin:enterprise'. Below it is a section for generating a new token, with 'Generate new token (classic)' highlighted by a red box. Other buttons include 'Generate new token' (Beta) and 'Revoke all'.

The screenshot shows the 'New personal access token (classic)' creation form. It includes a 'Note' field containing 'jenkins-github', an 'Expiration' field set to '90 days' (highlighted with a red box), and a 'Select scopes' section. In the 'repo' scope group, the 'repo' checkbox is checked (highlighted with a red box), and other options like 'repo:status', 'repo\_deployment', 'public\_repo', 'repo:invite', and 'security\_events' are listed. A note at the bottom says 'Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)'

Click **Generate token** at the bottom of the page.

Copy the token:

The screenshot shows the 'Personal access tokens (classic)' page again. It lists a single token: 'ghp\_ESF0fGISwHGZCX6dVYcp8TKmR7hSZ82yXkNt' (highlighted with a red box) with a copy icon. A message above the token says 'Make sure to copy your personal access token now. You won't be able to see it again!' There are also 'Generate new token' and 'Revoke all' buttons at the top right.

Return to the **worker-pipe** configuration page in Jenkins and configure the credentials. Be sure to use “Username and Password” and provide the token as the password, not the password you use to login to GitHub.

The screenshot shows the Jenkins configuration interface for the 'worker-pipe' item. The left sidebar lists several configuration sections: General, **Branch Sources**, Build Configuration, Scan Multibranch Pipeline Triggers, Orphaned Item Strategy, Appearance, Health metrics, and Properties. The 'Branch Sources' section is currently selected and highlighted with a red box. On the right, under 'Branch Sources', there is a dropdown menu titled 'Add source ▾' which is also highlighted with a red box. The menu contains options: Bitbucket, Git, and GitHub, with 'GitHub' being the selected option and also highlighted with a red box. Below the menu, it says 'Single repository & branch'. A 'Mode' dropdown is set to 'by Jenkinsfile'. Under 'Script Path', the value 'worker/Jenkinsfile' is entered. At the bottom of the 'Branch Sources' section, there are 'Save' and 'Apply' buttons.

This screenshot shows the 'Branch Sources' configuration for GitHub. It displays a 'GitHub Credentials' dialog box. Inside the dialog, there is a dropdown menu labeled '- none -' with a red box around it. Below it is a '+ Add ▾' button, which is also highlighted with a red box. A dropdown menu for selecting a Jenkins Credentials Provider is open, showing 'instavote » worker-pipe' and 'Jenkins', with 'Jenkins' being the selected option and highlighted with a red box. Below the provider selection is a 'Repository HTTPS URL' input field. At the bottom of the dialog, there is a 'Validate' button. At the very bottom of the configuration page, there are 'Save' and 'Apply' buttons.

Domain  
Global credentials (unrestricted)

Kind  
Username with password

Scope ?  
Global (Jenkins, nodes, items, all child items, etc)

Username ?  
eeganlf

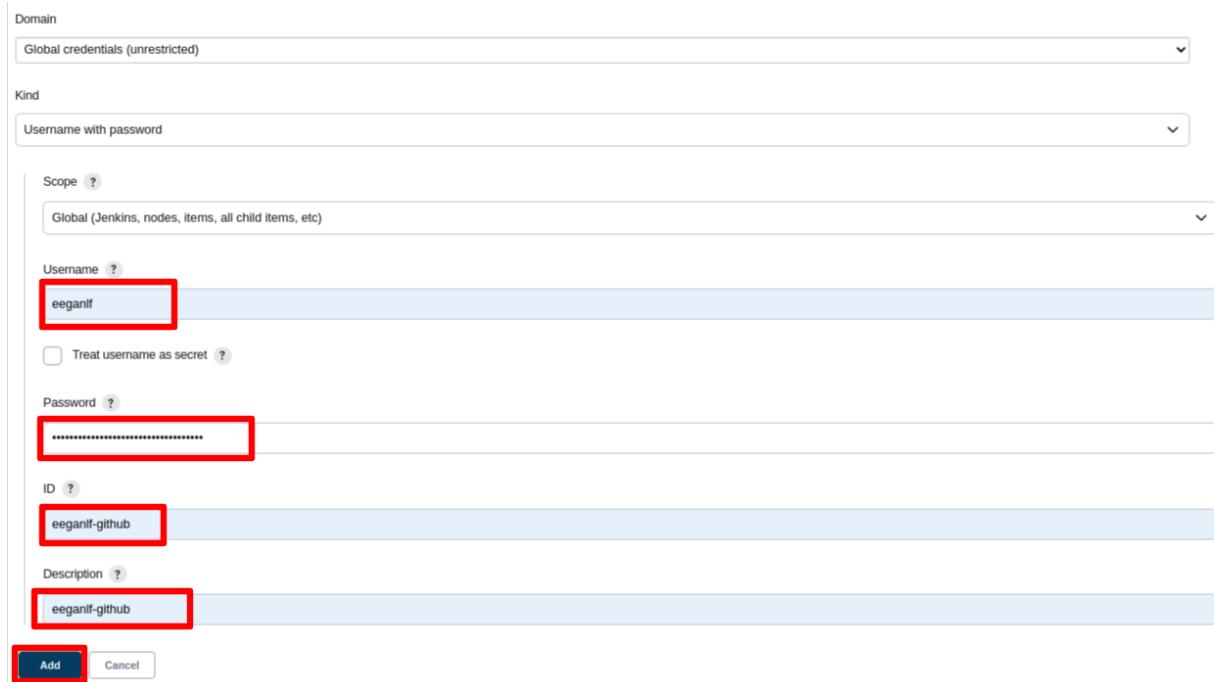
Treat username as secret ?

Password ?  
\*\*\*\*\*

ID ?  
eeganlf-github

Description ?  
eeganlf-github

**Add** **Cancel**



Under **Branch Sources** click **Add source** and select **GitHub**.

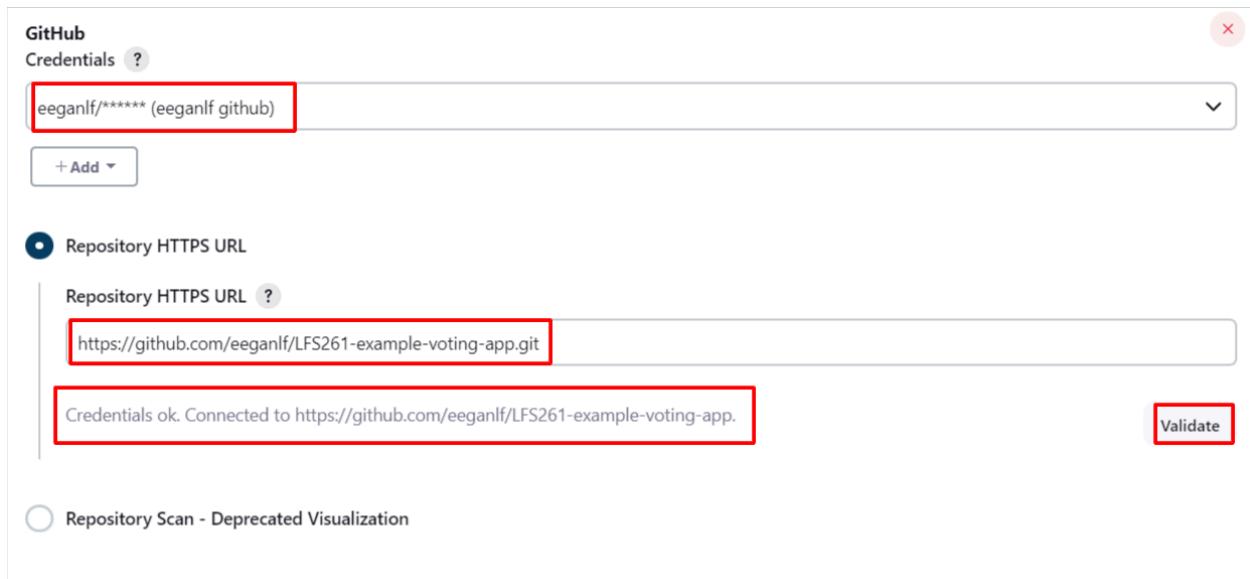
**GitHub**  
Credentials ?  
eeganlf/\*\*\*\*\* (eeganlf github)

+ Add ▾

Repository HTTPS URL  
Repository HTTPS URL ?  
https://github.com/eeganlf/LFS261-example-voting-app.git

Credentials ok. Connected to https://github.com/eeganlf/LFS261-example-voting-app.  
**Validate**

Repository Scan - Deprecated Visualization



Under **Build Configuration**, input your Jenkinsfile path as `worker/Jenkinsfile` inside the **Script Path** input box.

The screenshot shows the Jenkins configuration interface. On the left, there is a sidebar with the following options:

- General
- Branch Sources
- Build Configuration** (highlighted with a red box)
- Scan Multibranch Pipeline Triggers
- Orphaned Item Strategy
- Appearance
- Health metrics

The main panel is titled "Build Configuration". It has a "Mode" dropdown set to "by Jenkinsfile". Below it is a "Script Path" input field containing "worker/Jenkinsfile", which is also highlighted with a red box.

Select “1 minute” from the **Interval** drop down under **Scan Multibranch Pipeline Triggers** and **Save** the configuration.

The screenshot shows the Jenkins configuration interface. On the left, there is a sidebar with the following options:

- General
- Branch Sources
- Build Configuration
- Scan Multibranch Pipeline Triggers** (highlighted with a red box)

The main panel is titled "Scan Multibranch Pipeline Triggers". It contains a checkbox labeled "Periodically if not otherwise run" with a question mark icon, followed by an "Interval" dropdown menu. The "Interval" dropdown is open, showing the option "1 minute", which is also highlighted with a red box.

**Scan Repository Log**

```

Started
[Sun Nov 13 22:29:20 UTC 2022] Starting branch indexing...
22:29:20 Connecting to https://api.github.com using eeganlf/******** (auth token for github)
Examining eeganlf/LFS261-example-voting-app

Checking branches...

Getting remote branches...

Checking branch master

Getting remote pull requests...
'worker/Jenkinsfile' not found
Does not meet criteria

Checking branch readme

2 branches were processed

Checking pull-requests...

Checking pull request #2
'worker/Jenkinsfile' found
Met criteria
Scheduled build for branch: PR-2

1 pull requests were processed

Finished examining eeganlf/LFS261-example-voting-app

[Sun Nov 13 22:29:22 UTC 2022] Finished branch indexing. Indexing took 1.9 sec
Finished: SUCCESS

```

**Build Queue**

No builds in the queue.

Dashboard > instavote > worker-pipe >

**worker-pipe**

S	W	Name	Last Success	Last Failure	Last Duration	Fav
✓	☀️	master	51 sec #1	N/A	39 sec	▶ ⭐
✓	☀️	readme	51 sec #1	N/A	39 sec	▶ ⭐

Icon: S M L      Icon legend      Atom feed for all      Atom feed for failures      Atom feed for just latest builds

**Status**

**Configure**

**Scan Repository Now**

**Scan Repository Log**

**Multibranch Pipeline Events**

**Delete Multibranch Pipeline**

**People**

**Build History**

**Project Relationship**

**Check File Fingerprint**

**Move**

**Open Blue Ocean**

**Rename**

**Config Files**

**Pipeline Syntax**

**Credentials**

**Disable Multibranch Pipeline**

The screenshot shows the Jenkins Branch Readme view for the project 'instavote/worker-pipe/readme'. The top navigation bar includes links for Status, Changes, Build Now, View Configuration, Full Stage View, Open Blue Ocean, GitHub, Pipeline Syntax, and Build History. The Build History section shows a single build (#1) from Nov 13, 2022, at 10:32 PM, with 'No Changes'. The Stage View section displays five stages: Declarative: Checkout SCM (1s), build (11s), test (7s), package (10s), and Declarative: Post Actions (226ms). Below the stage view is a timeline showing the duration of each stage. The Permalinks section lists four links: Last build (#1), Last stable build (#1), Last successful build (#1), and Last completed build (#1), all from 1 min 34 sec ago.

It will automatically scan your repository. Whenever it detects a new branch under the repository, it runs the pipeline build automatically.

Once the pipeline run is successful, add two more stages in the same Jenkinsfile that contain two more commands: `mvn clean test` and `mvn package`:

```
pipeline {
    agent any

    tools{
        maven 'maven 3.9.8'
    }

    stages{
        stage("build") {
            steps{
                echo 'Compiling worker app..'
                dir('worker'){
                    sh 'mvn compile'
                }
            }
        }
        stage("test") {
            steps{
                echo 'Running Unit Tests on worker app..'
                dir('worker'){

```

---

```

        sh 'mvn clean test'
    }
}
stage("package") {
    steps{
        echo 'Packaging worker app'
        dir('worker'){
            sh 'mvn package'
        }
    }
}
post{
    always{
        echo 'Building multibranch pipeline for worker is completed..'
    }
}
}

```

After making changes in Jenkinsfile, commit the file from your `feature/workerpipe` and push the changes into the same branch:

```

git status
git add worker/Jenkinsfile
git commit -am "added Test and package job for worker pipeline"
git push origin feature/workerpipe

```

Once you push the changes to the `feature/workerpipe` branch, the scan will automatically trigger the `feature/workerpipe` branch build in the `instavote` multibranch pipeline.

After running the package, it will create a jar/war file. You need to archive that artifact.

```

pipeline {
    agent any

    tools{
        maven 'maven 3.9.8'
    }

    stages{
        stage("build"){

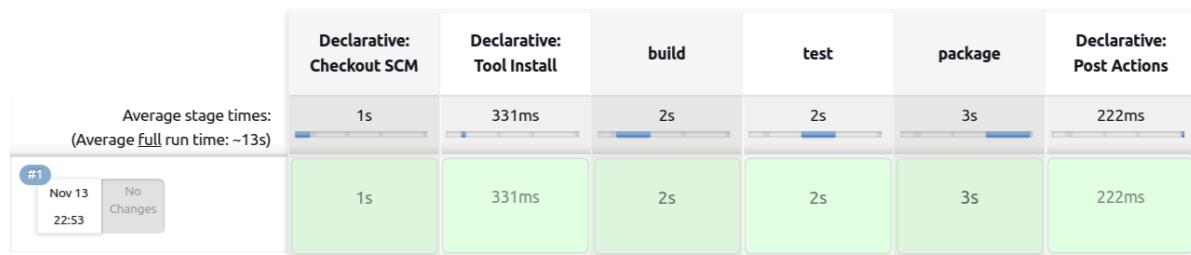
```

```
steps{
    echo 'Compiling worker app..'
    dir('worker'){
        sh 'mvn compile'
    }
}
stage("test"){
    steps{
        echo 'Running Unit Tests on worker app..'
        dir('worker'){
            sh 'mvn clean test'
        }
    }
}
stage("package"){
    steps{
        echo 'Packaging worker app'
        dir('worker'){
            sh 'mvn package -DskipTests'
        }
    }
}
post{
    always{
        archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
        echo 'Building multibranch pipeline for worker is completed..'
    }
}
```

## Branch feature/workerpipe

Full project name: instavote/worker-pipe/feature%2Fworkerpipe

### Stage View



### Permalinks

- [Last build \(#1\), 4 min 37 sec ago](#)
- [Last stable build \(#1\), 4 min 37 sec ago](#)
- [Last successful build \(#1\), 4 min 37 sec ago](#)
- [Last completed build \(#1\), 4 min 37 sec ago](#)

Commit the changes to `feature/workerpipe` and push to the branch:

```
git status
git add worker/Jenkinsfile
git commit -am "archive artifacts, skip test and package"
git push origin feature/workerpipe
```

You have set up a multibranch pipeline so that the CI pipelines are run automatically for every branch that you create in future. This is extremely useful to get the feedback from the CI system, before you merge the code into the trunk. This helps ensure that the master branch is never broken.

## Configuring Conditional Execution of Stages

With the current configuration, the pipeline you have created will launch for any and every change committed to the repository. However, it makes sense to restrict this behavior to changes made for the `worker` application. This is a desirable feature with mono repositories that contain multiple subprojects.

You may also want to restrict certain jobs to run only on specific branches. For example, integration and acceptance tests should run only for changes to the master branch, and

---

packaging applications to create and distribute artifacts should only run for the master branch.

This is achieved by setting up conditional execution. You are going to define the following conditions for the `worker` pipeline:

- Run packaging jobs only on master
- Run the jobs in this pipeline only if the code in the `worker` subdirectory is updated

Read the description of the `when` directive in the [Jenkinsfile documentation](#) for reference, then refactor the code as follows:

```
pipeline {
    agent any

    tools{
        maven 'maven 3.9.8'

    }
    stages{
        stage("build"){
            when{
                changeset "**/worker/**"
            }

            steps{
                echo 'Compiling worker app..'
                dir('worker'){
                    sh 'mvn compile'

                }
            }
        }
        stage("test"){
            when{
                changeset "**/worker/**"
            }

            steps{
                echo 'Running Unit Test on worker app..'
                dir('worker'){
                    sh 'mvn clean test'
                }
            }
        }
        stage("package"){
            when{
                branch 'master'
```

---

```

        changeset "**/worker/**"
    }
    steps{
        echo 'Packaging worker app'
        dir('worker'){
            sh 'mvn package -DskipTests'
            archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true  }

    }
}

post{
    always{
        echo 'Building multibranch pipeline for worker is completed..'
    }
}

}

```

Once you make changes, commit the changes to `feature/workerpipe` and push to GitHub:

```

git status

git add worker/Jenkinsfile

git commit -am "run package step only on master, run stages only when
worker changes"

git push origin feature/workerpipe

```

After making these changes, you will notice that the build will run only for the first two stages and skip the package stage.

**WARNING:** Even though `Jenkinsfile` is part of the `worker` subpath, it's not considered to be part of the changeset to trigger the builds. You will need to make changes to other files in order to see the jobs being triggered.

You can make some changes in the `README.md` file in the root directory, commit those changes, and push to `feature/workerpipe`. The build will be triggered, but it will skip all three stages.

Use `git log` and `reset` to go back to your previous commit:

```
git log
```

---

```
git reset --hard yourlastcommitid
git push origin feature/workerpipe -f
```

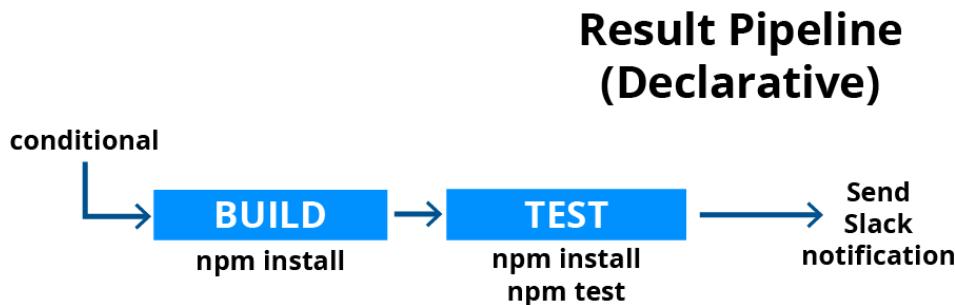
You have learned how to use conditional pipeline stages.

Once you are done with the work, don't forget to incorporate these changes into the master branch via a **pull request** and delete the feature branch.

## Exercise: Writing a Jenkinsfile for the Result NodeJS Application

You have been tasked to write a declarative pipeline for the `result` application:

- Two stages, build and test
- Stages should run conditionally only when there is a change to the `result` subdirectory
- You can refer to the following workflow:



The solution to this exercise will be provided as a separate document. Refer to it to validate once you have attempted it yourself.

## Summary

We learned how to write a pipeline using declarative code in this lab. We also learned how to enforce workflows through GitHub, create multi-branch pipelines, and add conditional stages.



## Lab 5S. Solution to the Jenkinsfile Exercise

With this exercise, you will create a `result` declarative pipeline for a NodeJS application and a multi-branch Jenkins project to build and test it.

### Writing A Jenkinsfile for a NodeJS Application

Create a branch `feature/resultpipe` using the `git checkout` command:

```
git checkout -b feature/resultpipe
```

Create a Jenkinsfile inside the `result` directory or copy the previous `Jenkinsfile` file from `worker` into `result`.

In the `result/Jenkinsfile`, add Node.js tools with the exact version that is configured in **Manage Jenkins > System Configuration > Tools > Nodejs Installations**.

Paste the following code into your Jenkinsfile:

```
pipeline {
    agent any

    tools{
        nodejs 'NodeJS 22.4.0'

    }

    stages{
        stage(build) {
            when{
                changeset "***/result/**"
            }

            steps{
                echo 'Compiling result app..'
            }
        }
    }
}
```

```
        dir('result'){
            sh 'npm install'
        }
    }
}

stage(test){
    when{
        changeset "**/result/**"
    }
    steps{
        echo 'Running Unit Tests on result app..'
        dir('result'){
            sh 'npm install'
            sh 'npm test'
        }
    }
}
}
```

Now you have a Jenkinsfile that installs and tests npm, and it will also send a Slack notification if you have completed that lab. Commit and push the changes into `feature/resultpipe` on GitHub using the following commands. This will automatically run the pipeline:

```
git status  
git add result/Jenkinsfile  
git commit -am "adding Jenkinsfile for result app"  
git push origin feature/resultpipe
```

On Jenkins, create a new multibranch pipeline called **result-pipe** and copy from **worker-pipe**. Go to your **result-pipe** configuration page, name it **result-pipe**, add the description **instavote result multi branch pipeline** and, under **Build Configuration**, change the script path to **result/Jenkinsfile**. **Save** the configuration page to create the project. Observe it as it scans your repository and starts the build.

Go to `result/test/mock.test.js` file and add one more mock test by copying and pasting an existing test block. Commit the changes to the `feature/resultpipe` branch using the following commands:

---

```
git status  
git add result/Jenkinsfile  
git commit -am "adding mock test"  
git push origin feature/resultpipe
```

Create a pull request on your GitHub account and merge it with the master branch. Once merged, Jenkins will run pipeline jobs for master. Disable the jobs `worker-build` and `result-build` via their respective configuration pages. These do not need to run anymore.

In this exercise, you learned how to create a Jenkinsfile for a Node.js application.

## Common Issues

If you see the following error as part of the test job run on Jenkins:

```
+ npm test  
  
> result@1.1.0 test /var/jenkins_home/workspace/e_result  
pipe_feature_resultpipe/result  
> mocha  
  
sh: 1: mocha: not found  
npm ERR! Test failed. See above for more details.
```

Ensure that you have added `npm install` as part of the test stage:

```
stage(test){  
    when{  
        changeset "**/result/**"  
    }  
    steps{  
        echo 'Running Unit Tests on result app..'  
        dir('result'){  
            sh 'npm install'  
            sh 'npm test'  
        }  
    }  
}
```



## Lab 5.1. Pipeline as Code (Additional Topics)

After creating a Jenkinsfile and a multi-branch project to build the Java application, you can additionally configure integration with Slack and send notifications related to builds right from the Jenkinsfile.

### Integrating Slack with Jenkins

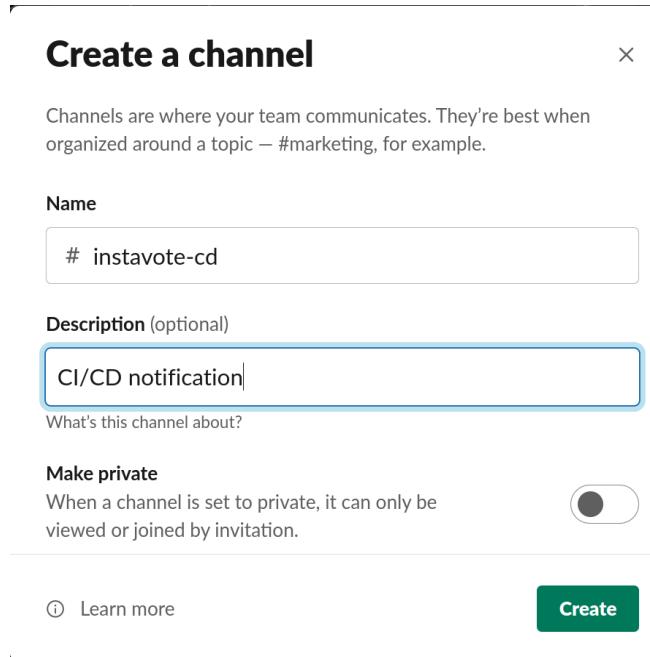
**NOTE:** *This is an optional section. Refer to it only if you wish to integrate with Slack for notifications.*

You will learn how to integrate Slack with Jenkins. This will be helpful to send notifications related to your build from Jenkins to a Slack channel.

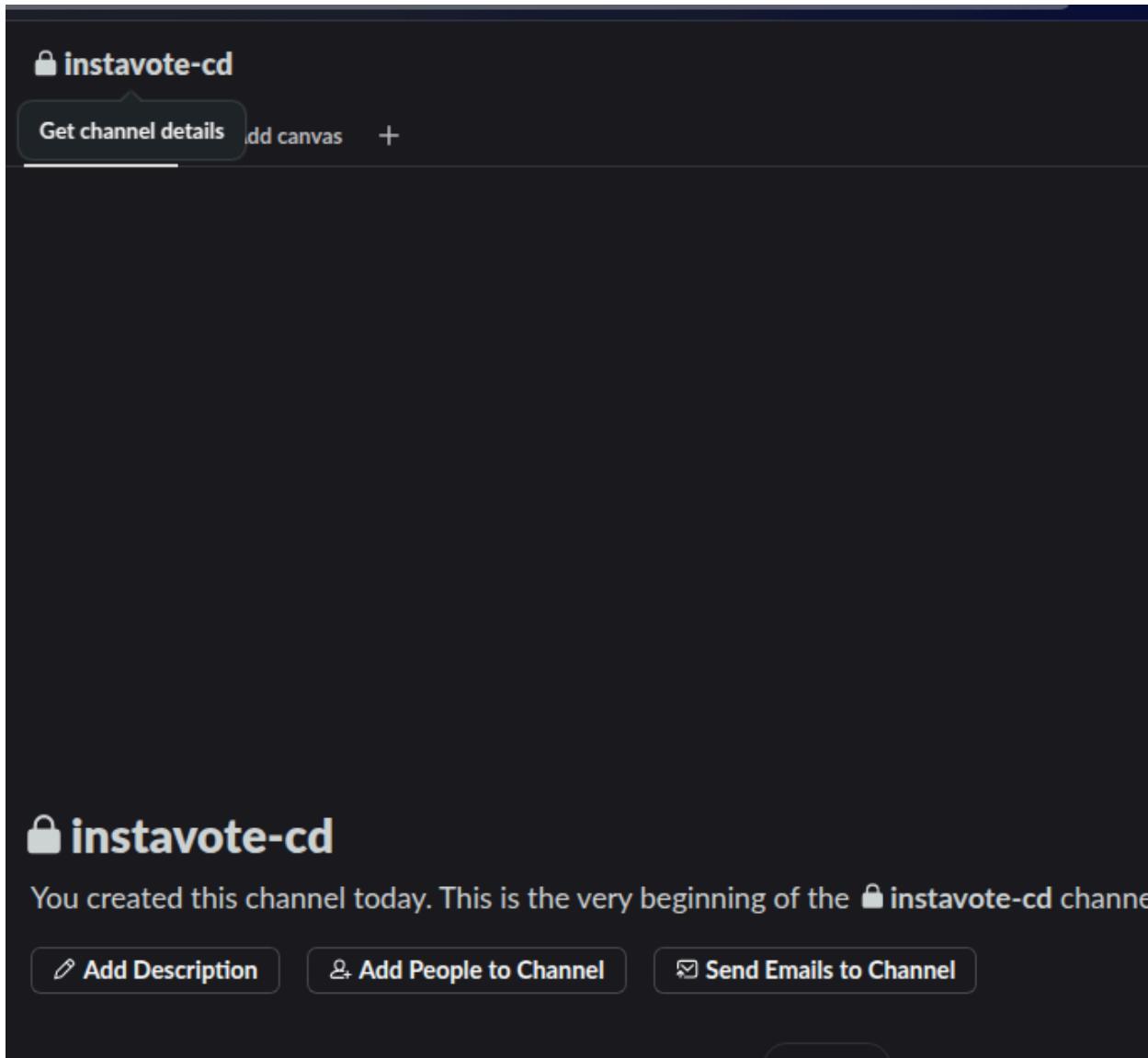
As a prerequisite for this, you need a Slack account. Additionally, you should be the Slack channel administrator.

Follow these steps to complete this setup.

Create a Slack account and create a channel named `#instavote-cd`. You can find Slack documentation on creating a channel [here](#).



After creating the channel, go to the channel details by clicking the name of the channel.



Find the **Integrations** tab and click **Add an App**.

instavote-cd

instavote-cd

X

★ Get Notifications for @ Mentions Huddle

About Members 1 Tabs Integrations Settings

Supercharge your channel

There's a few automations that will make your life easier. Set them up in a flash.

Add Automation Learn More

Apps

Bring the tools you need into this channel to pull reports, start calls, file tickets and more.

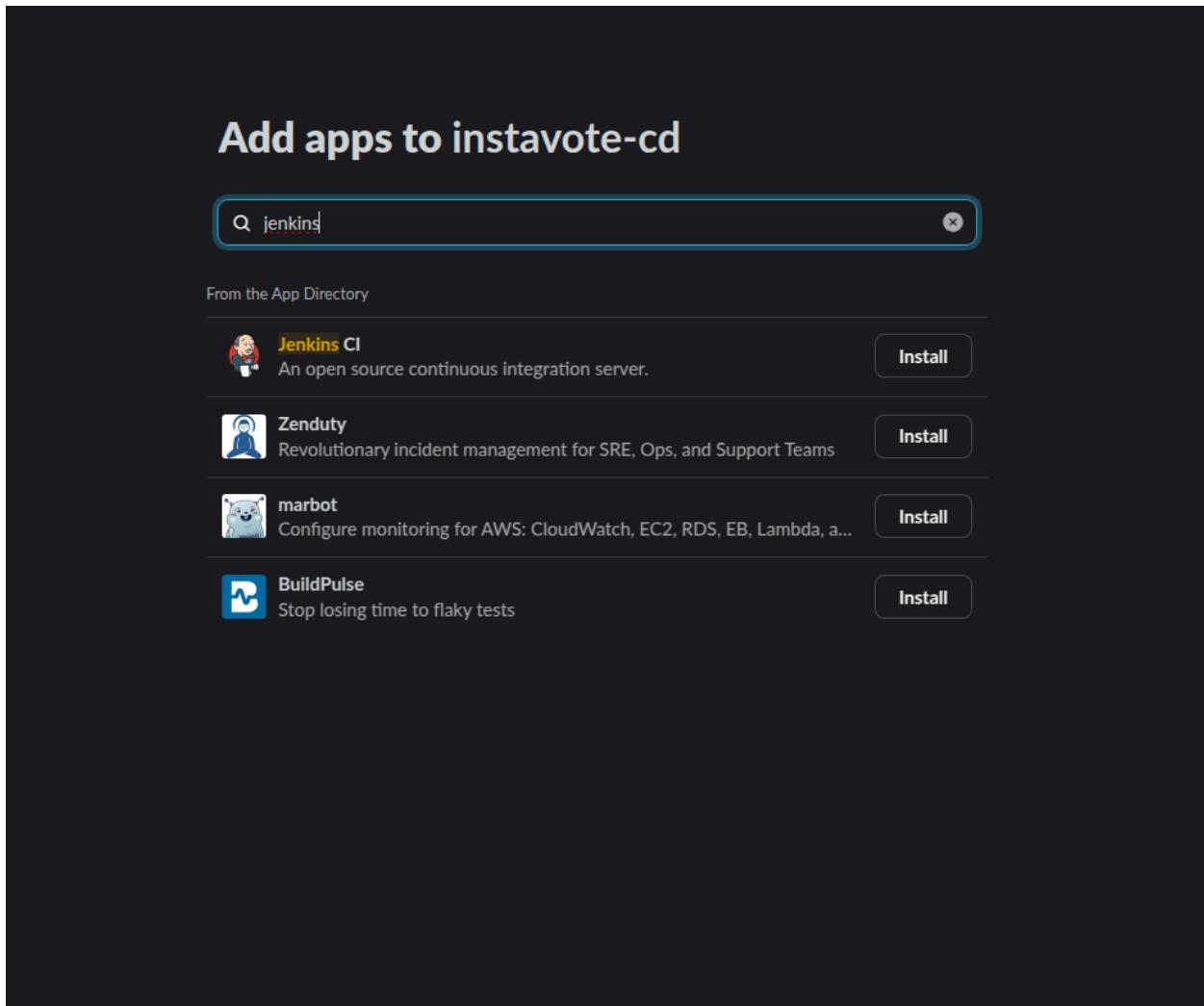
Add an App

Send emails to this channel

Get an email address that posts incoming emails in this channel.

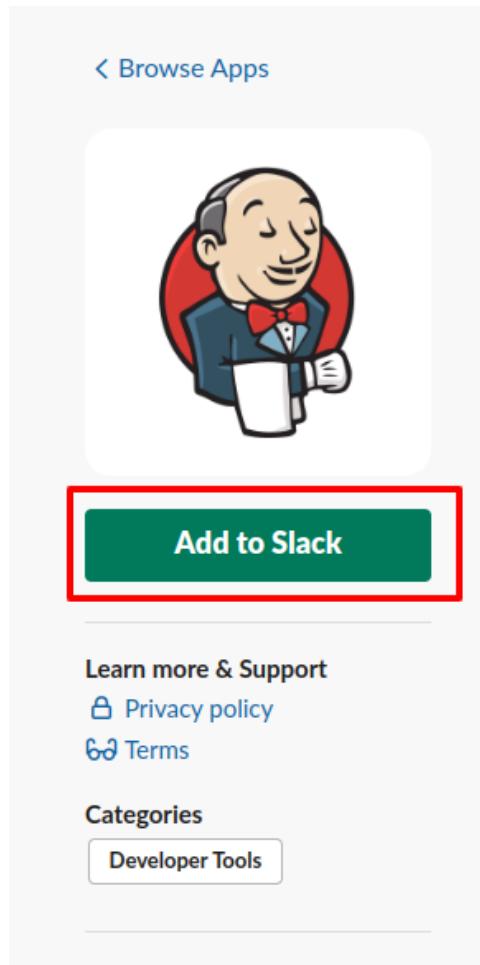
The screenshot shows the Slack interface for the 'instavote-cd' channel. At the top, there are three buttons: a star icon, a bell icon labeled 'Get Notifications for @ Mentions', and a 'Huddle' button. Below these are navigation links: 'About', 'Members 1', 'Tabs', 'Integrations' (which is underlined), and 'Settings'. A large yellow callout box titled 'Supercharge your channel' contains text about automations and two buttons: 'Add Automation' and 'Learn More'. To the right of this box is a diagram showing two automation flows: one from a lightning bolt icon to a document icon, and another from a checkmark icon to a hash tag icon. Below this is a section titled 'Apps' with a sub-section for adding tools. A hand is shown interacting with a laptop screen displaying various app icons. At the bottom left is an envelope icon with the text 'Send emails to this channel' and 'Get an email address that posts incoming emails in this channel'.

Search for “Jenkins CI” and select it.

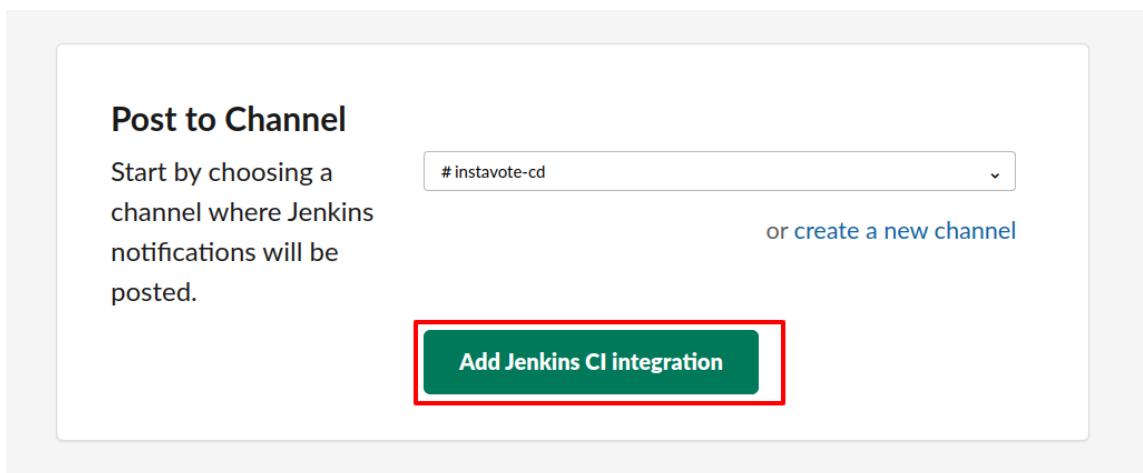


This will open a browser window.

Click **Add to Slack**.



Choose the channel and click **Add Jenkins CI integration**.



Follow the **Setup Instructions** on this page:

**Jenkins CI**

Added by eegan on July 7th, 2022

Jenkins CI is a customizable continuous integration server with over 600 plugins, allowing you to configure it to meet your needs.

This integration will post build notifications to a channel in Slack.

## Setup Instructions

Here are the steps necessary to add the Jenkins CI integration.

**Note:** These instructions are for v2.8. To install an older version, go down to [Previous Setup Instructions](#).

You will be gathering the **Team Subdomain** and **Integration Token Credential ID** from this page, so leave it open.

**Step 3** After it's installed, click on **Manage Jenkins** again in the left navigation, and then go to **Configure System**. Find the **Global Slack Notifier Settings** section and add the following values:

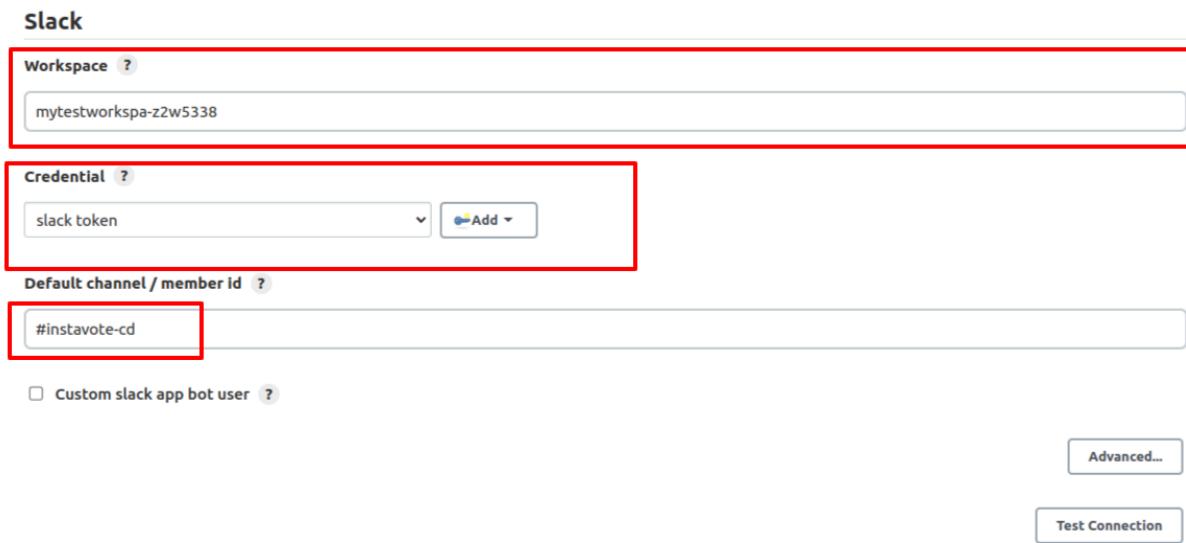
- **Team Subdomain:** `mytestworkspa-z2w5338`
- **Integration Token Credential ID:** Create a secret text credential using `sxEdKd0tMdvyx0jq7gG9VgUZ` as the value

Go to your Jenkins, **Manage Jenkins > Manage Plugins > Available Plugins**, and search for the “Slack notification” plugin and install it without a restart.

Go to **Manage Jenkins > System**. At the bottom, you will find the **Slack** section. Paste your **Team Subdomain** into the **Workspaces** input box.

Click **Add** under **Credential**. Add the credential as secret text using the integration token credential ID given in the Slack-Jenkins setup instructions in previous steps.

Add your **Default channel / member id** as `#instavote-cd` (or whatever you named your channel in Slack) and save the configuration.



From the Slack configuration page, click to save the configuration.

From your Jenkins account, choose any job and go to the configuration page. Add the Slack notification as the post-build action. Save and build the job. Now your notifications should show up in your Slack channel.

## Sending Notifications from a Pipeline Job

You can add Slack notifications to any pipeline job with the `slackSend` directive. You will have to set up the integration as described above before you attempt this.

Below is a code snippet which demonstrates how to achieve this. The entire Jenkinsfile can be found here:

[https://raw.githubusercontent.com/lftraining/LFS261-example-voting-app/refs/heads/master/gists/lab5.1\\_worker\\_jenkinsfile\\_slack\\_integration](https://raw.githubusercontent.com/lftraining/LFS261-example-voting-app/refs/heads/master/gists/lab5.1_worker_jenkinsfile_slack_integration).

```
file: worker/Jenkinsfile

pipeline {
    agent any
```

```
tools{
    maven 'maven 3.9.8'
}

stages{
    stage(build) {
        when{
            changeset "**/worker/**"
        }

        steps{
            echo 'Compiling worker app..'
            dir('worker'){
                sh 'mvn compile'
            }
        }
    }
    stage(test) {
        when{
            changeset "**/worker/**"
        }
        steps{
            echo 'Running Unit Tests on worker app..'
            dir('worker'){
                sh 'mvn clean test'
            }
        }
    }
    stage(package) {
        when{
            branch 'master'
            changeset "**/worker/**"
        }
        steps{
            echo 'Packaging worker app'
            dir('worker'){
                sh 'mvn package -DskipTests'
                archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true
            }
        }
    }
}

post{
```

```
always{
    echo 'Building multibranch pipeline for worker is completed..'
}
failure{
    slackSend (channel: "instavote-cd", message: "Build Failed - ${env.JOB_NAME} ${env.BUILD_NUMBER} (<${env.BUILD_URL}|Open>)" )
}

success{
    slackSend (channel: "instavote-cd", message: "Build Succeeded - ${env.JOB_NAME} ${env.BUILD_NUMBER} (<${env.BUILD_URL}|Open>)" )
}

}
```

Once you make changes in the Jenkinsfile, commit and push to the `feature/workerpipe` branch:

```
git status
git add worker/Jenkinsfile
git commit -am "add slack notifications"
git push origin feature/workerpipe
```

After pushing the changes to the branch, the pipeline will build automatically and send a notification to Slack. From Slack, you will get a link to visit the build status of your job.

Since this feature is complete, it is time to merge it all into the master. Once merged, don't forget to delete the branch from the remote repo, as well as from your local workspace.



## Lab 6. Using Docker with Jenkins Pipelines

By the end of this lab exercise, you should be able to:

- Prepare the Jenkins environment to build with a Docker agent
- Refactor a Jenkinsfile with a Docker-based agent

### Steps:

- Read "[Using Docker with Pipeline](#)"
- Refactor the Jenkinsfile for the `worker` application

## Install Docker Plugins

Install the following plugins:

- Docker Plugin
- Docker Pipeline

 A screenshot of the Jenkins 'Manage Jenkins > Plugins' page. The search bar at the top contains the text 'docker'. On the left sidebar, 'Available plugins' is selected. In the main pane, three Docker-related plugins are listed:
 

- Docker 1.6.2**: This plugin integrates Jenkins with Docker. It has a checked checkbox next to its name. Release date: 1 mo 19 days ago.
- Docker Commons**: Provides the common shared functionality for various Docker-related plugins. It has an unchecked checkbox next to its name. Release date: 1 yr 0 mo ago.
- Docker Pipeline**: Build and use Docker containers from pipelines. It has a checked checkbox next to its name. Release date: 2 mo 2 days ago.

 A red box highlights the checked checkboxes for the Docker and Docker Pipeline plugins.

---

Refer to the screenshot above to choose the correct plugins and proceed with the installation.

## Refactoring the Worker Pipeline to Build with Docker Agent

Create a new branch to make changes.

```
git checkout -b feature/dockerbuild
```

Now update the Jenkinsfile for the `worker` app:

- Remove the `tools` section
- Change the agent configuration from `any` to `docker`
- Provide the Docker agent configurations with
  - `image: maven:3.9.8-sapmachine-21`
  - `args: '-v $HOME/.m2:/root/.m2'`

Your `worker/Jenkinsfile` should contain the following:

```
pipeline {

    agent{
        docker{
            image 'maven:3.9.8-sapmachine-21'
            args '-v $HOME/.m2:/root/.m2'
        }
    }

    stages{
        stage('build'){
            steps{
                echo 'building worker app'
                dir('worker'){
                    sh 'mvn compile'
                }
            }
        }
        stage('test'){
            steps{
                echo 'running unit tests on worker app'
                dir('worker'){
                    sh 'mvn clean test'
                }
            }
        }
    }
}
```

```

stage('package') {
    steps{
        echo 'packaging worker app into a jarfile'
        dir('worker'){
            sh 'mvn package -DskipTests'
            archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true
        }
    }
}
post{
    always{
        echo 'the job is complete'
    }
}

}

```

Once you have modified the file, commit the changes:

```

git commit -am "add docker based agent"
git push origin feature/dockerbuild

```

Jenkins launches the pipeline, this time with Docker as an agent.

## Validation

While the job is running, log into the Docker DIND Host and watch for the container which gets launched when the pipeline starts and stays till the end of the build. Switch into `devops-repo/setup`:

```

$ docker compose exec docker sh
$ watch docker ps

...
...
[observe the container running while the pipeline is in progress]
...
...
[^c followed by ^d to stop the watch command and exit from the
container]

```

You should see containers being launched periodically by Jenkins. You can also observe the console logs for the pipeline to see a container being launched, then stopped, and finally removed as part of the build process. You can always view the pipeline console output by clicking the build number and clicking **Console Output** on the menu on the right.

Once you've done this, remember to merge the changes into the master branch via a pull request and then delete the feature branch from remote (GitHub) and locally.

## Summary

In this lab, we built a simple Docker pipeline to ensure that Docker was working on Jenkins. We refactored our **worker** application Jenkinsfile so that the pipeline would run on Docker.



## Lab 7. Packaging with Docker

In addition to helping with the process of building and testing software, Docker can also provide a standard packaging format with Docker images. This packaging format comes with the distribution mechanism, i.e., Docker image registries.

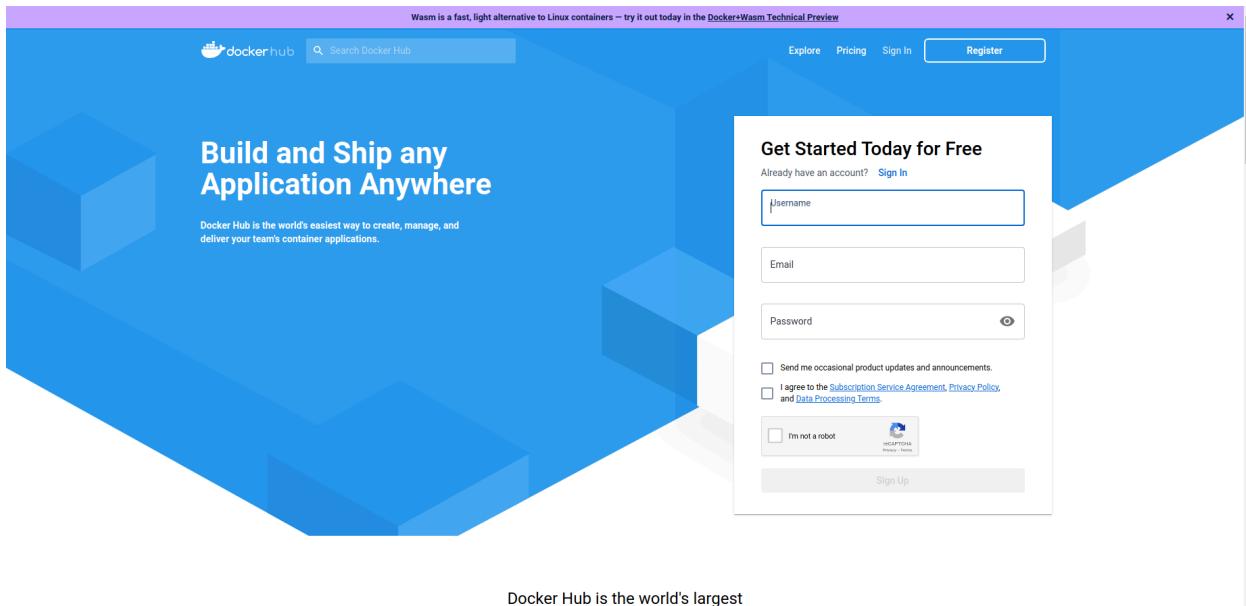
By the end of this lab exercise, you should be able to:

- Discuss Docker's standard image format
- Test and build Docker images manually
- Write Dockerfiles
- Understand the automated, iterative Docker image process
- Add the Docker packaging stage to the Jenkins pipeline
- Create per-stage agent configurations in Jenkinsfile

### Registering with Docker Hub

As you prepare to work with the registry, it's important to have your own account. This account will enable you to build and push images to the registry. For the purpose of this tutorial, we'll be using a hosted registry called Docker Hub.

If you do not already have an account, visit the following link: <https://hub.docker.com/> and sign up using your email.



Docker Hub is the world's largest

Check your email inbox and check the activation email sent by Docker.

After clicking on the activation link, you will be redirected to a login page. Enter your credentials and log in.

You will be taken to the Docker Hub main page. The registration process is complete and you have an account with Docker Hub.

## Test Building Docker Image for the `worker` Application

Before you start building automated images, you will manually create a Docker image. You have already used the pre-built image from the registry in the last section. In this subsection, you will create an image with the `worker` application installed. Since `worker` is a Java-based application that needs Maven to build, you will base your work on Maven's existing official image.

Enter the `example-voting-app/worker` directory.

Create a container named `build` with the `maven:3.9.8-sapmachine-21` image:

```
docker run -idt --name build maven:3.9.8-sapmachine-21 sh
```

Copy the source code. You must be inside the `example-voting-app/worker` directory for this to work:

```
docker container cp . build:/app
```

---

Connect to the container to compile and package the code:

```
docker exec -it build sh  
cd app  
mvn package
```

Verify the `.jar` file has been built:

```
ls target/
```

Run the `.jar` file with the following `java` command:

```
java -jar target/worker-jar-with-dependencies.jar
```

Sample output:

```
Waiting for redis  
ctrl+c
```

\*Use `Ctrl+c` to exit.

The above is the expected output. The `worker` app is waiting for `redis`.

Move the artifact:

```
mv target/worker-jar-with-dependencies.jar /run/worker.jar
```

Remove the source code:

```
rm -rf /app/*
```

Exit the container shell to return to your machine's shell:

```
exit
```

Commit the container to an image.

```
docker container commit build <xxxxxx>/worker:v1
```

List your containers:

```
docker ps
```

Look for the container named `build` and take note of the container **ID**.

---

Commit the container into an image.

**NOTE:** Be sure to replace <xxxxx> with your own Docker Hub Account ID, that is your username.

Test before pushing by launching the container with the packaged app:

```
docker run --rm -it <xxxxx>/worker:v1 java -jar /run/worker.jar
...
...
...
```

\*Use **Ctrl + C** to exit.

Push the image to the registry.

**NOTE:** Before you push the image, you need to be logged in to the registry, with the Docker Hub ID created earlier.

Login using the following command:

```
docker login
```

To push the image, first list it:

```
docker image ls
```

Sample output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
eeganlf/worker	v2	90cbeb6539df	18 minutes ago	194MB
eeganlf/worker	v1	c0199f782489	34 minutes ago	189MB

To push the image, use:

```
docker push <xxxxx>/worker:v1
```

## Automated Image Build with Dockerfile

Now, let's build the same image, this time using a Dockerfile.

To do this, create a file and name it **Dockerfile** in the root of the source code for the **worker** app. It should have the path **example-voting-app/worker/Dockerfile**:

```
FROM maven:3.9.8-sapmachine-21
WORKDIR /app
```

---

```
COPY . .
RUN mvn package && \
    mv target/worker-jar-with-dependencies.jar /run/worker.jar && rm -rf /app/*
CMD ["java", "-jar", "/run/worker.jar"]
```

Let's build the image. Make sure you are in the `example-voting-app/worker` directory:

```
docker image build -t <xxxxx>/worker:v2 .
docker image ls
```

Try building again:

```
docker image build -t <xxxxx>/worker:v2 .
```

This time it runs much faster because Docker uses the cache.

Test the image using:

```
docker container run --rm -it <xxxxx>/worker:v2
```

You will see "waiting for redis" repeating in the console if it works.

\*Use `Ctrl + c` to exit.

Tag the image as `latest`:

```
docker image tag <xxxxx>/worker:v2 <xxxxx>/worker:latest
docker image ls
```

Finally, publish images to the registry:

```
docker image push <xxxxx>/worker:latest
docker image push <xxxxx>/worker:v2
```

You must commit the Dockerfile into your Git repo before you can automate this process with Jenkins:

```
git add worker/Dockerfile
git commit -am "adding Dockerfile for worker"
git push origin feature/dockerfiles
```

## Adding Docker Build and Publish Stages to Jenkinsfile

To have Jenkins build and publish a Docker image, you must add a pipeline script block. Copy and paste the following code inside the stages section between the `package` and `post` stages

---

of your **worker/Jenkinsfile**:

```
stage('docker-package') {
    agent any
    steps{
        echo 'Packaging worker app with docker'
        script{
            docker.withRegistry('https://index.docker.io/v1/',
'dockerlogin') { def workerImage =
docker.build("xxxx/worker:v${env.BUILD_ID}", "./worker")
            workerImage.push()
            workerImage.push("latest")
        }
    }
}
}
```

The entire **worker/Jenkinsfile** pipeline should appear as follows, BUT notice the red text which should be replaced with your Docker Hub ID:

```
pipeline{
    agent{
        docker{
            image 'maven:3.9.8-sapmachine-21'
            args '-v $HOME/.m2:/root/.m2'
        }
    }

    stages{
        stage('build'){
            steps{
                echo 'building worker app'
                dir('worker'){
                    sh 'mvn compile'
                }
            }
        }
        stage('test'){
            steps{
                echo 'running unit tests on worker app'
                dir('worker'){
                    sh 'mvn clean test'
                }
            }
        }
    }
}
```

```

        }
    stage('package') {
        steps{
            echo 'packaging worker app into a jar file'
            dir('worker'){
                sh 'mvn package -DskipTests'
                archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true
            }
        }
    }
    stage('docker-package') {
        agent any
        steps{
            echo 'Packaging worker app with docker'
            script{
                docker.withRegistry('https://index.docker.io/v1/',
'dockerlogin') {
                    def workerImage = docker.build("xxxxxx/worker:v$"
{env.BUILD_ID}", "./worker")
                    workerImage.push()
                    workerImage.push("latest")
                }
            }
        }
    }
}

post{
    always{
        echo 'the job is complete'
    }
}

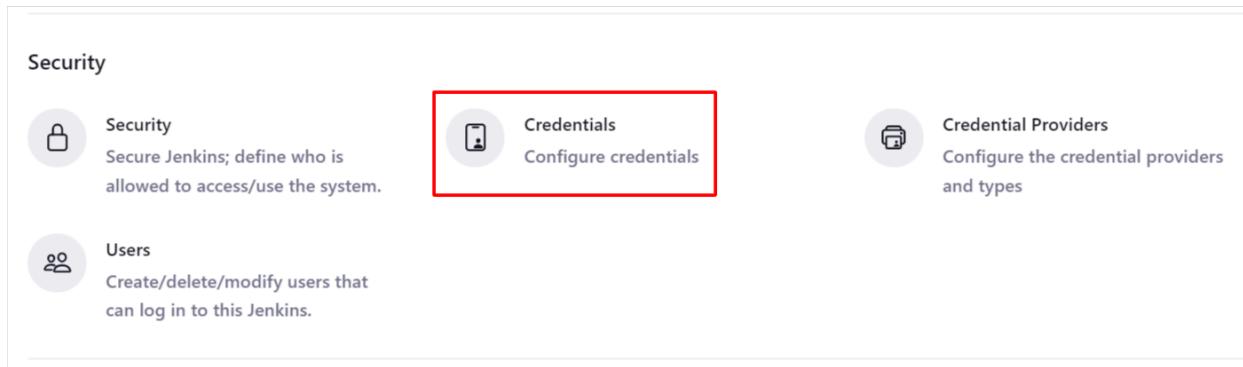
}

```

## Adding Docker Hub Credentials

Add the username/password credentials to Jenkins with name `dockerlogin`. To do so, browse to **Jenkins Dashboard > Manage Jenkins > Credentials > Jenkins > Global Credentials > Add Credentials > Username and password**. Ensure you add the Docker Hub login and password with the ID as `dockerlogin`.

Reference the screenshots below to follow along.



The screenshot shows the Jenkins Security page. It includes three main sections: 'Security' (with a lock icon), 'Credentials' (with a key icon, highlighted by a red box), and 'Users' (with a person icon). Below these are sections for 'Credential Providers' and 'Stores scoped to Jenkins'.

**Credentials**

T	P	Store	Domain	ID	Name
key	System	(global)		0d47ee8b-691f-4e9e-8b9e-02049e396a56	Secret text
key	System	(global)		jenkins-github-access-token	jenkins-github-access-token
key	System	(global)		github-auth-token	eeganlf/******** (auth token for github)
key	System	(global)		sonar-maven-examples	sonar-maven-examples

**Stores scoped to Jenkins**

P	Store	Domains
System	(global)	

**Global credentials (unrestricted)**

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
0d47ee8b-691f-4e9e-8b9e-02049e396a56	Secret text	Secret text	
jenkins-github-access-token	jenkins-github-access-token	Secret text	
github-auth-token	eeganlf/******** (auth token for github)	Username with password	auth token for github
sonar-maven-examples	sonar-maven-examples	Secret text	sonar-maven-examples

**New credentials**

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: eeganlf

Treat username as secret:

Password:

ID: dockerlogin

Description: dockerlogin

**Create**

Commit the changes you have made to the Jenkinsfile so that a new pipeline run is triggered. You will see the pipeline fail with an error similar to the one in the screenshot below.

```
$ docker exec --env ***** --env ***** --env ***** --env ***** --env ***** --env ***** --env ****
env ***** --env ****
env ***** --env ****
env ***** --env ****
env ***** --env ****
env ***** --env ***** 30fdaceaf998f05ff55e943dc14873b7eca2f9f72c7018f6753627274a28edbe docker login
-u eeganlf -p ***** https://index.docker.io/v1/
OCI runtime exec failed: exec failed: unable to start container process: exec: "docker": executable file
not found in $PATH: unknown
```

In order to fix this error, you will have to add *Per Stage Agent* configurations.

## Jenkinsfile Per Stage Agent Configurations

The following code contains the complete Jenkinsfile along with:

- Per Stage Agent configuration
- Conditional execution of the Docker **package** stage on the master
- Publishing the image on Docker Hub with the branch name as the tag

```
pipeline {
    agent none

    stages{
        stage("build") {
            when{
                changeset "**/worker/**"
            }

            agent{
                docker{
                    image 'maven:3.9.8-sapmachine-21'
                    args '-v $HOME/.m2:/root/.m2'
                }
            }

            steps{
                echo 'Compiling worker app..'
                dir('worker'){
                    sh 'mvn compile'
                }
            }
        }
        stage("test") {
            when{
                changeset "**/worker/**"
            }
            agent{
                docker{
                    image 'maven:3.9.8-sapmachine-21'
                    args '-v $HOME/.m2:/root/.m2'
                }
            }

            steps{
                echo 'Running Unit Tests on worker app..'
                dir('worker'){

```

```

        sh 'mvn clean test'
    }
}

}
stage("package") {
    when{
        branch 'master'
        changeset "**/worker/**"
    }
    agent{
        docker{
            image 'maven:3.9.8-sapmachine-21'
            args '-v $HOME/.m2:/root/.m2'
        }
    }
    steps{
        echo 'Packaging worker app'
        dir('worker') {
            sh 'mvn package -DskipTests'
            archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true
        }
    }
}

stage('docker-package') {
    agent any
    when{
        changeset "**/worker/**"
        branch 'master'
    }
    steps{
        echo 'Packaging worker app with docker'
        script{
            docker.withRegistry('https://index.docker.io/v1/',
'dockerlogin') {
                def workerImage =
docker.build("xxxxxx/worker:v${env.BUILD_ID}", "./worker")
                workerImage.push()
                workerImage.push("${env.BRANCH_NAME}")
                workerImage.push("latest")
            }
        }
    }
}

```

```

        }

    }

post{
    always{
        echo 'Building multibranch pipeline for worker is completed..'
    }
}
}
}

```

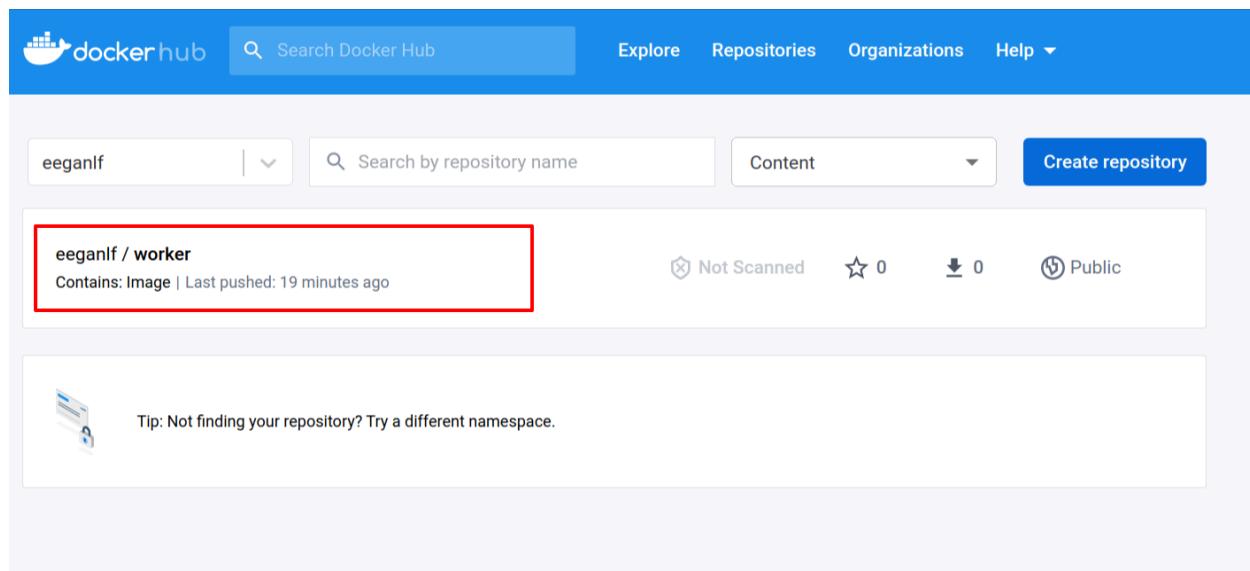
You can access the `worker/Jenkinsfile` with the stage agents as well as conditional execution code [here](#).

Once you edit your Jenkinsfile to this effect, commit the changes:

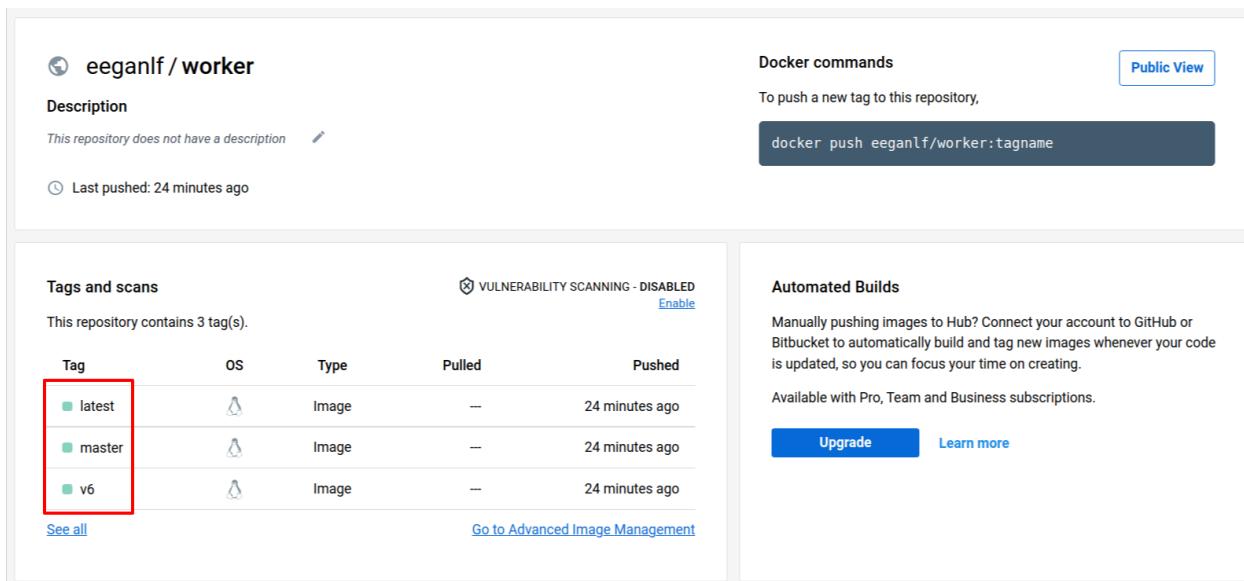
```
git commit -am "per stage agents, conditional execution"
git push origin feature/dockerfiles
```

Merge the code into the master via a pull request. Remember not to delete the branch yet, as you will iterate the Jenkinsfile for `vote` and `result` applications with similar configurations.

Head over to the Docker Hub repository to validate that Jenkins has built and published images.



Click on the image.



## Exercise: Vote and Result Application Dockerfiles

Follow the same process for `vote` and `result` applications:

- Check if Dockerfiles exist for `vote` and `result` applications. If not, create and check into the Git repository.
- Update Jenkinsfiles for `vote` and `result` applications with Docker Build and Publish stages.
- Make sure you refactor all Jenkinsfiles with Per Stage Docker Agent configurations.
- Validate the pipeline runs and images are visible on Docker Hub.

## Result application - inside `example-vote-app/result`

The manual image sans Dockerfile creation process for the `result` application is as follows:

```
docker container run -idt --name nodebuild -p 80:80 node:22.4.0-slim
docker cp . nodebuild:/app -this copies everything in . into
nodebuild container in the /app dir
docker exec -it nodebuild sh
which node
```

---

```

which npm

cd app

npm install -this will install modules from package.json

npm audit fix -this will fix known vulnerabilities in the modules
installed by npm

npm ls - lists all modules downloaded by npm

npm test - runs unit tests to verify your code is ok

npm start

waiting for db - will be repeatedly displayed

ctrl + c

exit

docker container commit nodebuild xxxxx/result:v1 - this will create an image
and tag it.

```

**docker image history xxxxx/result:v1** - this will show a list of commands entered inside containers run with the image. These will help build a Dockerfile for the image to automate the creation of containers based on the image.

**docker image push xxxxx/result:v1**

The automated build, using Dockerfile, is as follows. Create the **result/Dockerfile** file with the following code:

```

FROM node:22.4.0-slim

# add curl for healthcheck
RUN apt-get update && \
    apt-get install -y --no-install-recommends curl tini && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /usr/local/app

# have nodemon available for local dev use (file watching)
RUN npm install -g nodemon

COPY package*.json .

RUN npm ci && \
    npm cache clean --force && \
    mv /usr/local/app/node_modules /node_modules

```

```
COPY . .

ENV PORT 80
EXPOSE 80

ENTRYPOINT ["/usr/bin/tini", "--"]
CMD ["node", "server.js"]

docker image build -t xxxxx/result:v2 . - DO NOT FORGET THE PERIOD
because it tells Docker the path to the Dockerfile that will be used to build the image.

docker image tag xxxxx/result:v2 xxxxx/result:latest

docker image push xxxxx/result
```

## Vote application - inside `example-vote-app/vote`

Create a `Dockerfile` inside the `vote/` directory with the following code:

```
# Define a base stage that uses the official python runtime base image
FROM python:3.11-slim AS base

# Add curl for healthcheck
RUN apt-get update && \
    apt-get install -y --no-install-recommends curl && \
    rm -rf /var/lib/apt/lists/*

# Set the application directory
WORKDIR /usr/local/app

# Install our requirements.txt
COPY requirements.txt ./requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Define a stage specifically for development, where it'll watch for
# filesystem changes
FROM base AS dev
RUN pip install watchdog
ENV FLASK_ENV=development
CMD ["python", "app.py"]

# Define the final stage that will bundle the application for
# production
FROM base AS final

# Copy our code from the current folder to the working directory
# inside the container
```

---

COPY . .

```
# Make port 80 available for links and/or publish
EXPOSE 80

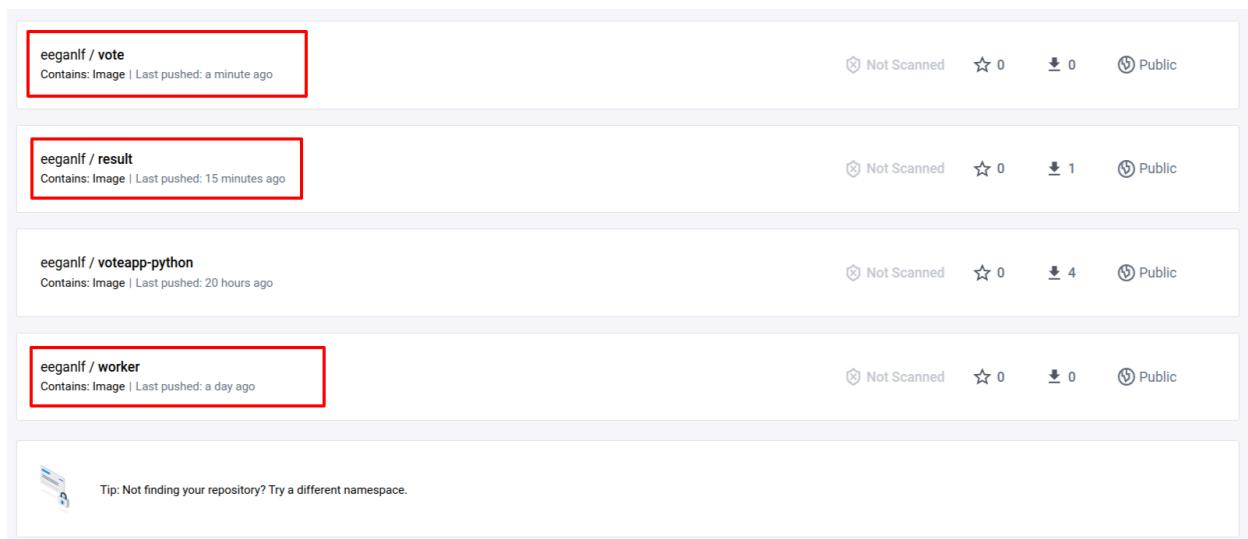
# Define our command to be run when launching the container
CMD ["gunicorn", "app:app", "-b", "0.0.0.0:80", "--log-file", "-",
"--access-logfile", "-", "--workers", "4", "--keep-alive", "0"]
```

**docker image build -t xxxxx/vote:v1 .** -DO NOT FORGET THE PERIOD because it tells Docker the path to the Dockerfile that will be used to build the image.

**docker image tag xxxxx/vote:v1 xxxxx/vote:latest**

**docker image push xxxxx/vote**

You should now see your images on your Docker Hub repository page with any other images you may have pushed in the past:



These images will be required for the next lab.



## Lab 8. Deploy to Dev with Docker Compose

By the end of this lab exercise, you should be able to:

- Explain how Docker Compose can launch a monitoring stack
- Add services to Docker Compose
- Create a consolidated Jenkins pipeline
- Start deploying to an integrated dev environment with Docker Compose

Before you begin, create a new branch and switch to it:

```
git checkout -b feature/monopipe
```

Also, enter the **devops-repo/setup** directory and shut Jenkins down while we explore the next steps:

```
cd devops-repo/setup
```

```
docker compose down
```

We will restart Jenkins after exploring the example voting application. If we do not shut Jenkins down, there will be port conflicts.

### Create a Simple Docker Compose Spec

**NOTE:** Be sure to replace **xxxxx** with your Docker Hub ID.

```
cd example-voting-app
```

Create the **docker-compose.yaml** with the following contents.

---

```
file: example-voting-app/docker-compose.yaml

services:
  vote:
    image: xxxxx/vote:latest
    ports:
      - 5000:80

  result:
    image: xxxxx/result:latest
    ports:
      - 5001:80

  worker:
    image: xxxxx/worker:latest
```

You can access the file [here](#).

**NOTE:** Be sure to replace **xxxxx** with your Docker Hub ID.

Launch the stack with:

`docker compose up -d`

`docker compose ps`

At this time, if you try submitting the vote from the UI, you will see an error. That is because you have not provisioned the backend service, i.e. **redis**.

Add **redis** along with **db** in `docker-compose.yaml`:

```
file: example-voting-app/docker-compose.yaml

services:
  vote:
    image: xxxxx/vote:latest
    ports:
      - 5000:80

  redis:
    image: redis:alpine

  db:
    image: postgres:15-alpine

  result:
    image: xxxxx/result:latest
    ports:
```

---

```

- 5001:80

worker:
  image: xxxxx/worker:latest

```

You can access this Docker Compose yaml file [here](#).

**NOTE:** Be sure to replace `xxxxx` with your Docker Hub ID.

Launch the new services using the same command as earlier:

```

docker compose up -d
docker compose ps

```

At this time, if you attempt to submit the vote using the UI, it should go through. This validates the service discovery (connecting `vote` with `redis`).

## Add Networks and Volumes

To add networks and volumes to the example voting application, copy the following into your `example-voting-app/docker-compose.yaml`:

```

volumes:
  db-data:

networks:
  instavote:
    driver: bridge

services:
  vote:
    image: xxxxx/vote:latest
    ports:
      - 5000:80
    depends_on:
      - redis
    networks:
      - instavote

  redis:
    image: redis:alpine
    networks:
      - instavote

db:
  image: postgres:15-alpine

```

---

```

environment:
  POSTGRES_USER: "postgres"
  POSTGRES_PASSWORD: "postgres"
volumes:
  - "db-data:/var/lib/postgresql/data"
  - "./healthchecks:/healthchecks"
healthcheck:
  test: /healthchecks/postgres.sh
  interval: "5s"
networks:
  - instavote

result:
  image: xxxxx/result:latest
  ports:
    - 5001:80
depends_on:
  - db
networks:
  - instavote

worker:
  image: xxxxx/worker:latest
  depends_on:
    - redis
    - db
networks:
  - instavote

```

**NOTE:** Be sure to replace **xxxxx** with your Docker Hub ID.

As usual, launch it again with `docker compose`:

`docker compose up -d`

`docker compose ps`

You can find the complete Docker Compose file [here](#).

## Additional Docker Compose Commands to Explore

This command will list containers and the images used to create them.

`docker compose images`

---

The following commands will list logs of running services. The first will list all the logs of all the services. The second command demonstrates that you can specify a service from the `docker-compose.yaml` as an option to only show the logs for that service, in this example `worker`:

```
docker compose logs
```

```
docker compose logs worker
```

The following command pulls an image associated with a service defined in a `docker-compose.yml` or `docker-stack.yml` file, but does not start containers based on those images.

```
docker compose pull
```

The following command will execute a command inside a service. The following example specifically runs the `ps` command inside the `vote` service:

```
docker compose exec vote ps
```

The following command stops running containers without removing them. They can be started again with `docker compose start`:

```
docker compose stop
```

The following command stops and removes containers and networks:

```
docker compose down
```

If you run `docker compose logs` after running `docker compose down`, nothing will be displayed because it removes everything. You can get started again with `docker compose up -d`.

## Finishing Up

Push the changes you have made so far to the repository:

```
git add docker-compose.yaml  
git commit -am "adding docker compose spec"  
git push origin feature/monopipe
```

## Integrate Docker Compose with Dockerfiles

So far, you have been using pre-baked images to deploy to development. You can also have Docker Compose build the images by calling the respective Dockerfiles. Below is the code which will allow you to do this. Replace `xxxxx` with your DockerHub user ID.

```
volumes:
  db-data:

networks:
  instavote:
    driver: bridge

services:
  vote:
    image: xxxxx/vote:latest
    build: ./vote
    ports:
      - 5000:80
    depends_on:
      - redis
    networks:
      - instavote

  redis:
    image: redis:alpine
    networks:
      - instavote

  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
    networks:
      - instavote

  result:
    image: xxxxx/result:latest
    build: ./result
```

---

```

ports:
  - 5001:80
depends_on:
  - db
networks:
  - instavote

worker:
  image: xxxxx/worker:latest
  build: ./worker
  depends_on:
    - redis
    - db
  networks:
    - instavote

```

You can access the Docker Compose spec with Dockerfile build integration [here](#).

You can build and deploy using the following commands:

```

docker compose config
docker compose build
docker compose build (observe caching)
docker compose up -d
docker compose ps

```

Finally, be sure to run:

```
docker compose down
```

If you forget to do this, it will cause issues when you restart Jenkins and the DinD containers. Restart Jenkins and DinD by running the following from inside the **devops-repo/setup/** directory:

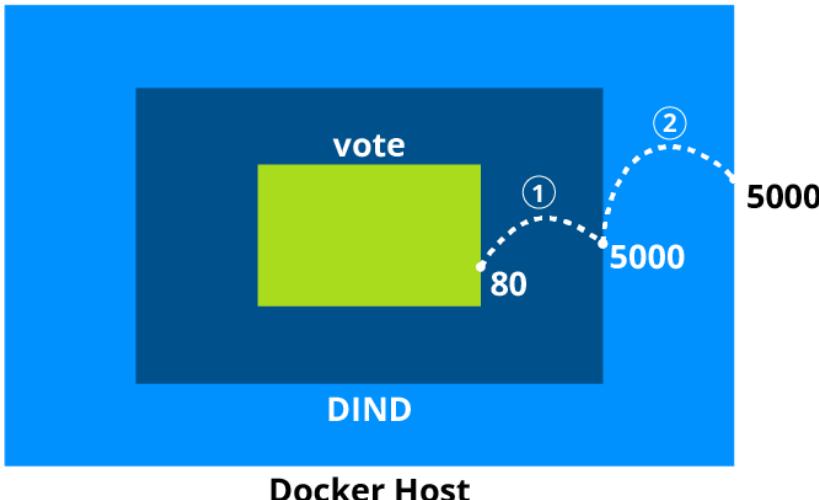
```
docker compose up -d
```

You are now ready to move on to the next section.

## Add New Port Mapping

You will have to define an additional port mapping from inside **devops-repo/setup/docker-compose.yaml** and redeploy it. The following diagram depicts the two level port mappings:

### 1. example-vote-app/docker-compose.yaml



Docker Host

### 2. devops-repo/setup/docker-compose.yaml

Edit the `compose` spec used to set up Jenkins and DIND server by adding new port mappings. Be sure to remove any previous containers consuming these ports before you do so. Make sure the following is in `devops-repo/setup/docker-compose.yaml`:

```
docker:
  image: docker:dind
  ports:
    - 2376:2376
    - 5000:5000
    - 5001:5001
  environment:
    - DOCKER_TLS_CERTDIR=/certs
```

You can access the `docker-compose.yaml` file [here](#).

After making this change, redeploy the setup using `docker compose up -d` from the setup directory. It will not disturb or recreate Jenkins, but it will relaunch the Docker DIND environment with new port mappings. Take care of any port conflicts that may exist while you bring up this environment.

## Exercise: Create a Consolidated Pipeline for Instavote Stack

You have created one `Jenkinsfile` per application, matching `vote`, `worker`, and `result` apps. Consolidating all those stages and building one single pipeline for the `instavote` app is called

---

a **Mono Pipe**. As an exercise, use the following steps as a loose reference while attempting to complete this exercise on your own:

- Copy over `worker/Jenkinsfile` to the top level directory, i.e. `example-voting-app/Jenkinsfile`.
- Rename stages to identify with the application name. i.e. `worker`.
- Bring in just the stages from `result/Jenkinsfile` and `vote/Jenkinsfile`.
- Commit the changes to the repository branch created earlier, i.e. `feature/monopipe`.
- Create a new Multi Branch Pipeline job.

Once you validate the pipeline, add the “deploy to dev” stage.

## Add “Deploy to Dev” with Docker Compose

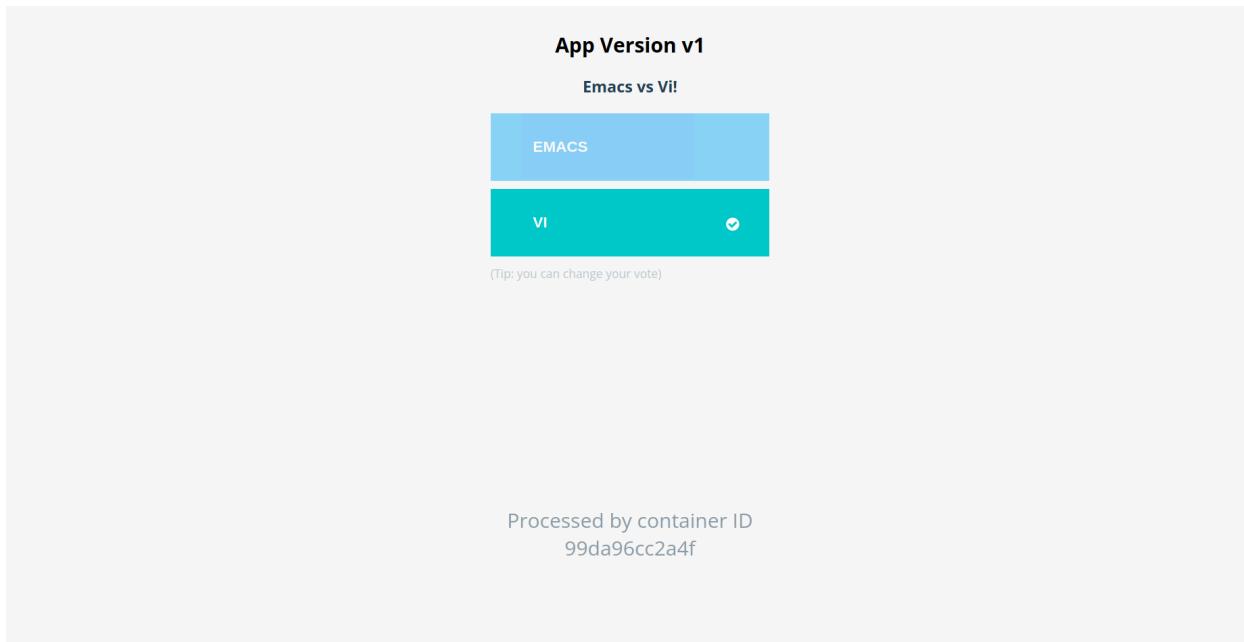
Add a “deploy to dev” stage to Jenkinsfile using the following stage snippet:

```
stage('deploy to dev') {
    agent any
    when{
        branch 'master'
    }
    steps{
        echo 'Deploy instavote app with docker compose'
        sh 'docker compose up -d'
    }
}
```

You can access the “deploy to dev” stage snippet [here](#).

The entire Jenkinsfile for the Mono Pipe can be found [here](#).

Be sure to push your changes to Github. After Jenkins runs the Mono Pipe job, you should be able to visit `IPADDRESS:5000`, for example `localhost:5000`, to see the vote application is up and running:



In this lab, we learned how to draft a `docker-compose.yaml` document and implemented a Mono Pipe with Jenkins. We also instructed Jenkins to automatically deploy our application. This concludes Lab 8.



## Lab 9. Continuous Testing

By the end of this lab exercise, you should be able to:

- Analyze code coverage with JaCoCo
- Improve the code coverage
- Set up SonarQube to automatically scan your repository
- Create a project gating system to automatically decide whether the code is safe to deploy
- Add integration and acceptance tests with Docker Compose
- Incorporate all of the steps into the Jenkins pipeline

### Code Coverage with Jacoco

Create a new Jenkins project which will download the following sample repository and run unit tests with a Jacoco plugin for Java and display code coverage reports.

Fork [the maven-examples repository](#) to your GitHub account. Be sure to *uncheck* the **Copy the master branch only** option, as follows:

## Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks](#).

**Owner \***      **Repository name \***

 eeganlf / maven-examples 

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

**Description (optional)**

**Copy the master branch only**  
Contribute back to Iftraining/maven-examples by adding your own branch. [Learn more](#).

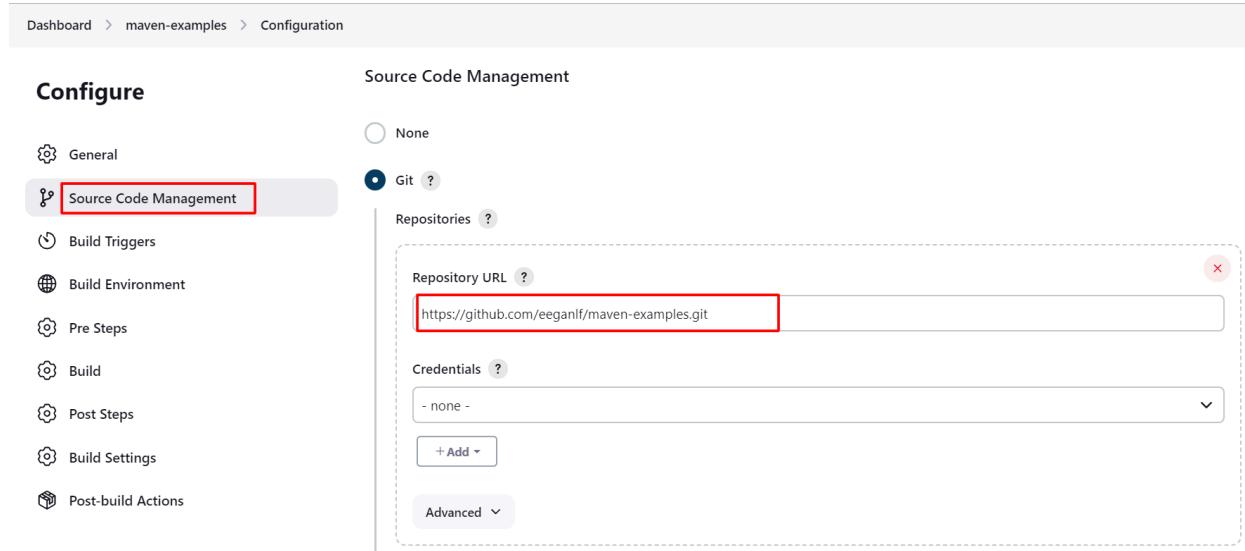
ⓘ You are creating a fork in your personal account.

**Create fork**

## Install the **Coverage** Plugin for Jenkins.

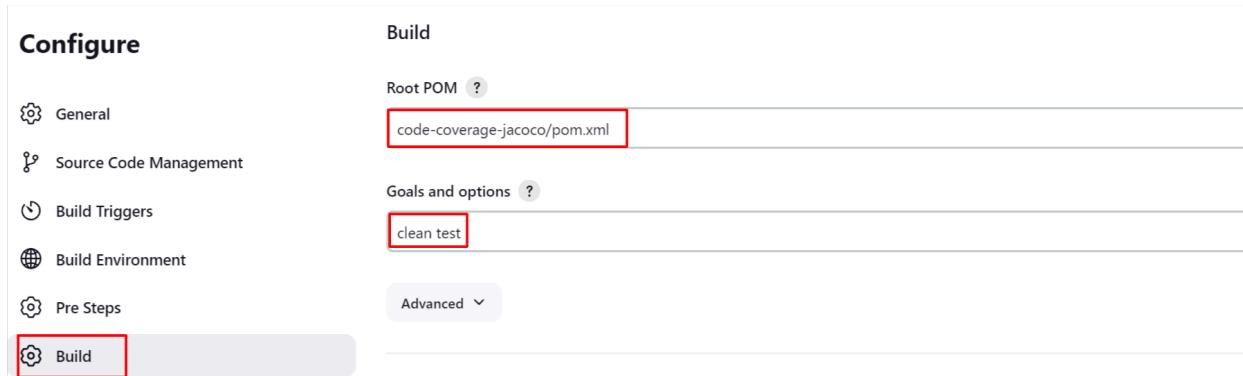
Create a Maven project named **maven-examples**, provide the repository you forked as source, add Maven goals to run tests on it. Use “clean test” as the Maven goal.

Under the **Source Code Management** section, select **Git** and paste the repository URL of the **maven-examples** repository that you forked.

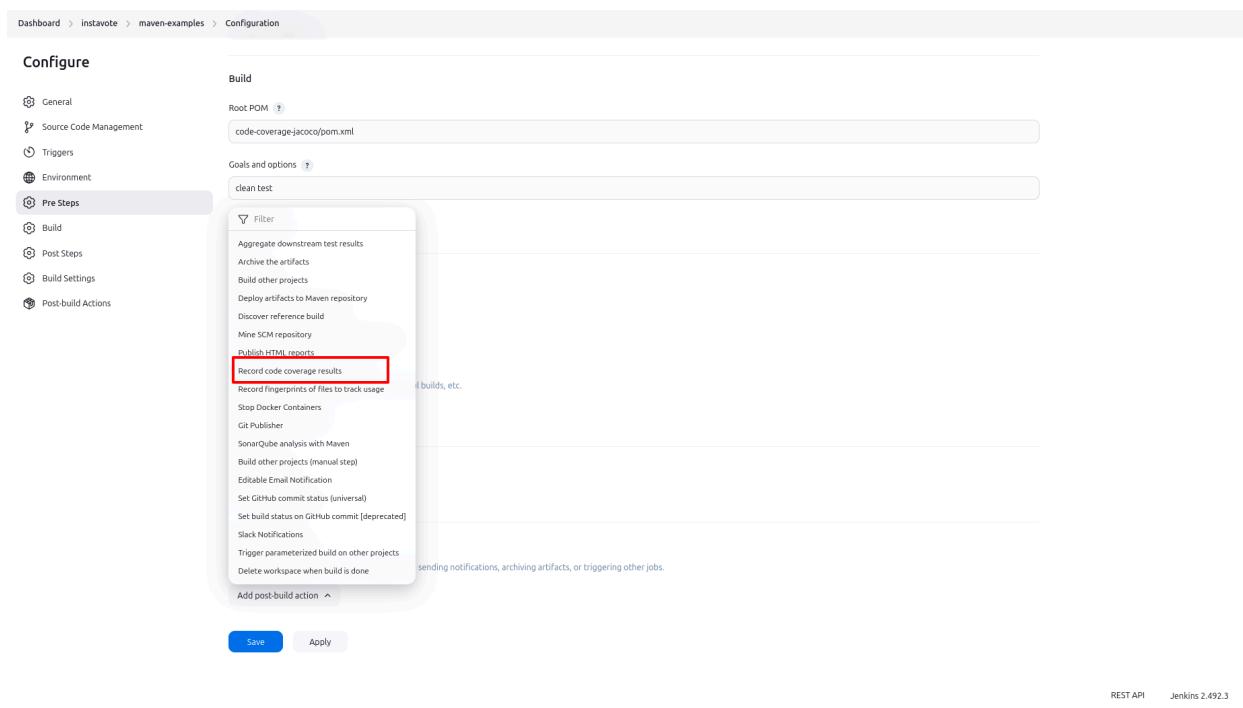


The screenshot shows the Jenkins configuration page for a job named "maven-examples". The left sidebar lists configuration sections: General, Source Code Management (highlighted with a red box), Build Triggers, Build Environment, Pre Steps, Build, Post Steps, Build Settings, and Post-build Actions. The "Source Code Management" section is expanded, showing a "Repositories" panel. The "Repository URL" field contains the value "https://github.com/eeganlf/maven-examples.git", which is also highlighted with a red box. The "Credentials" dropdown is set to "- none -". There is an "Advanced" button at the bottom of the panel.

Under the **Build** section, enter the path to the **pom.xml** as **code-coverage-jacoco/pom.xml**, and type **clean test** into the **Goals** text box.



From post-build actions, choose **Record code coverage results**.



Select **JaCoCo Coverage Reports** as the **Coverage Parser** and **Save**.

Build the project twice and refresh the page after the build is complete:

The screenshot shows the Jenkins interface for the 'maven-examples' project. On the left, there's a sidebar with options like Status, Changes, Workspace, Build Now, Configure, Delete Maven project, Modules, Favorite, Open Blue Ocean, Rename, and Coverage Report. The main area displays the project name 'maven-examples' with a green checkmark icon. Below it is a 'Permalinks' section listing four builds. To the right is a 'Code Coverage Trend' chart comparing Line Coverage and Branch Coverage between build #1 and #2. The chart shows a horizontal axis from 0 to 100 with two points: #1 at approximately 10 and #2 at approximately 10. The legend indicates that green dots represent Line Coverage and dark green dots represent Branch Coverage. At the bottom of the dashboard, there are links for REST API and Jenkins 2.492.3.

You should see a Code Coverage Trend between the two builds. It won't show much at the moment because you have only built the project once and there are no tests. You will add tests to the master branch next.

Create and merge a pull request from the `ut1` branch in your repository onto your `master` branch.

First, navigate to **Pull requests**:

The screenshot shows a GitHub repository page for **eeganlf / maven-examples**. The top navigation bar includes links for **Pull requests**, **Issues**, **Codespaces**, **Marketplace**, and **Explore**. Below the repository name, it says "forked from [lfraining/maven-examples](#)". The main navigation tabs are **Code** and **Pull requests**, with **Pull requests** highlighted by a red box. Other tabs include **Actions**, **Projects**, **Wiki**, **Security**, **Insights**, and **Settings**. Below the tabs, there are buttons for **master**, **1 branch**, and **0 tags**, along with **Go to file**, **Add file**, and **Code** dropdowns. A summary box states: "This branch is up to date with lfraining/maven-examples:master." It includes a **Contribute** button and a **Sync fork** button. The main content area displays four pull request entries:

- eeganlf removed test ...** (8eb1394, 2 hours ago, 63 commits)
- assembly-plugin** Move license to the root directory (7 years ago)
- code-coverage-jacoco** removed test (2 hours ago)
- findbugs-cookbook** First version of FindBugs Cookbook (9 years ago)

On the right side, there's an **About** section with the message: "No description, website, or topics provided." It also lists **Readme**, **View license**, **0 stars**, **0 watching**, and **669 forks**.

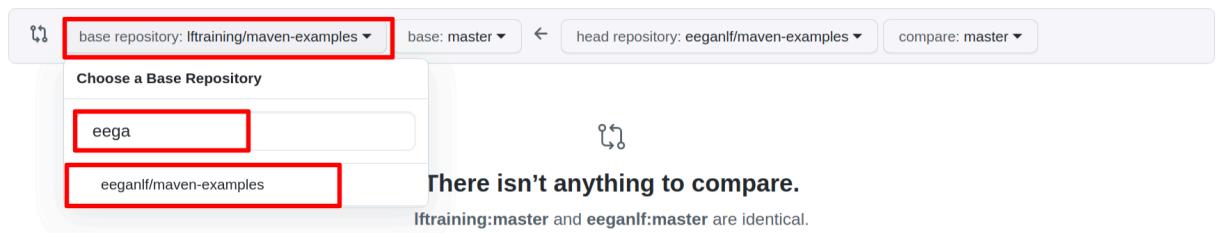
Click **New pull request**:

The screenshot shows the same GitHub repository page for **eeganlf / maven-examples**. The **Pull requests** tab is selected. The top navigation bar and repository information are identical to the previous screenshot. Below the tabs, there are filters for **Filters** (dropdown), **is:pr is:open** (search bar), **Labels** (9), **Milestones** (0), and a prominent **New pull request** button highlighted with a red box. The main content area features a large "Welcome to pull requests!" message with a small icon above it. Below the message, a paragraph explains: "Pull requests help you collaborate on code with other people. As pull requests are created, they'll appear here in a searchable and filterable list. To get started, you should [create a pull request](#)."

Click on the **base repository** dropdown and search for your GitHub username. When you find your repository, select it as the base repository:

### Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).



Select the **ut1** branch as the branch to **compare** and click **Create pull request**:

**eeganlf / maven-examples** (Public)  
forked from [lftutorial/maven-examples](#)

Code Pull requests Actions Projects Wiki Security Insights Settings

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base: master ← compare: ut1 ✓ Able to merge. These branches can be automatically merged.

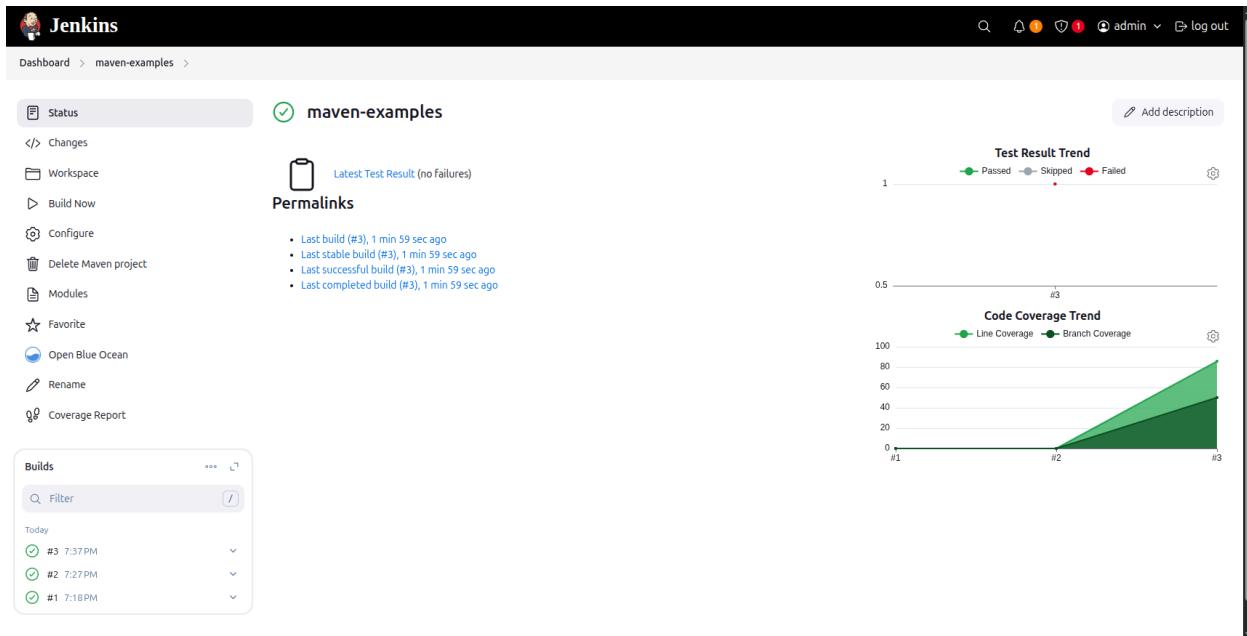
Discuss and review the changes in this comparison with others. [Learn about pull requests](#)

Create pull request

Approve the request as normal.

This will add pre-written tests to your master branch so that Jacoco will show some coverage in Jenkins.

Go back to the Maven project page you created earlier and build the project; refresh the page when the build is finished and you should see a code coverage report get published on the project page.



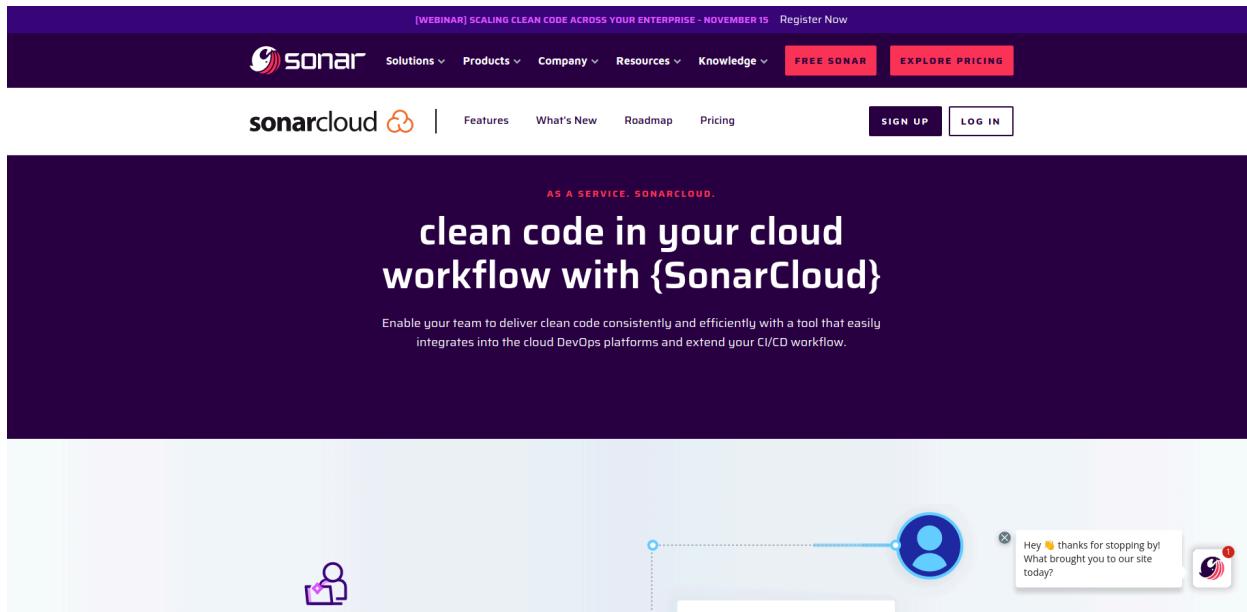
## Adding Static Code Analysis with SonarQube

### Steps:

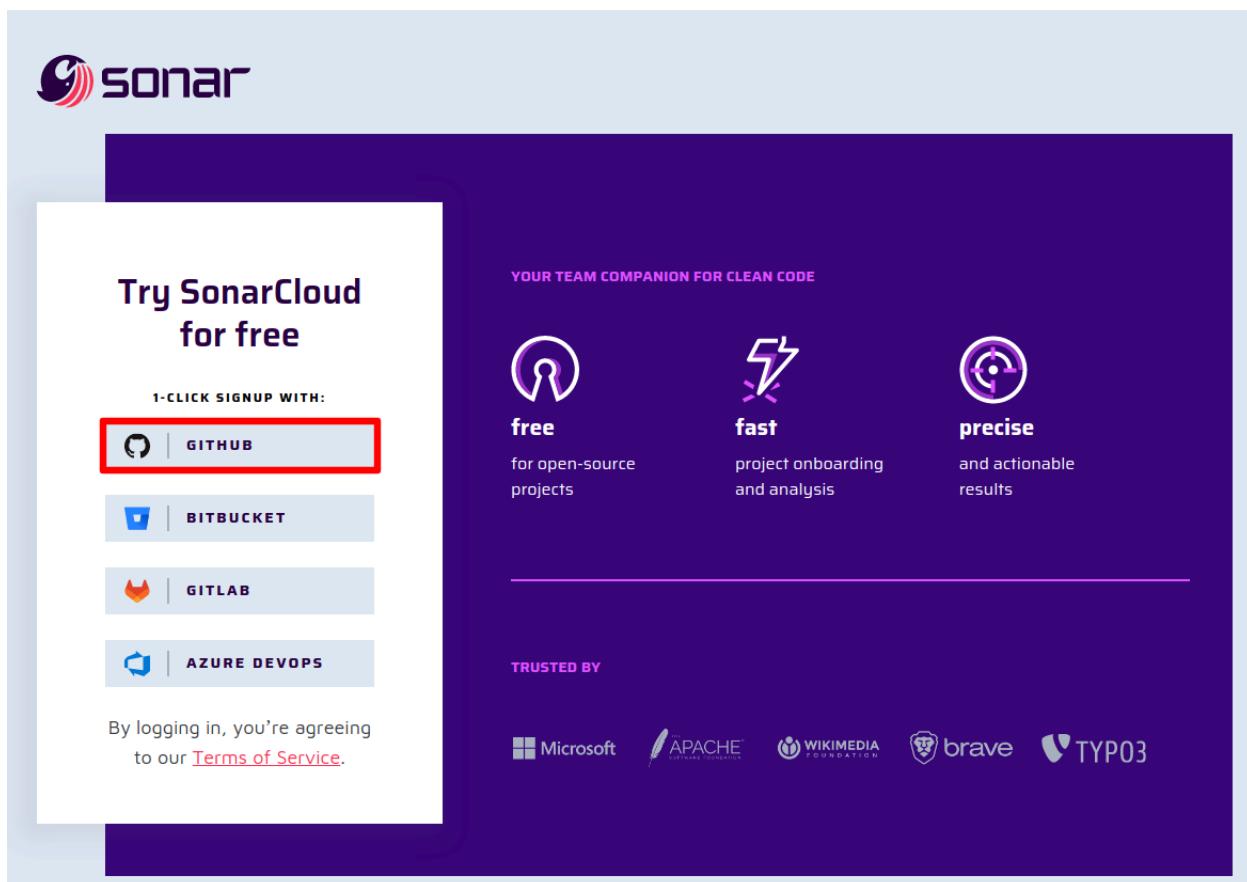
- Set up Sonar Cloud.
- Install SonarQube Scanner plugin for Jenkins.
- Configure SonarQube Servers from Jenkins Systems Configuration.

### Set Up Sonar Cloud as SonarQube Server

Head over to [SonarCloud](#) and sign up using your GitHub account.



The screenshot shows the SonarCloud homepage. At the top, there's a purple header bar with the Sonar logo and navigation links for Solutions, Products, Company, Resources, Knowledge, a 'FREE SONAR' button, and an 'EXPLORE PRICING' button. Below the header, the main content area has a dark background. It features the 'sonarcloud' logo with a cloud icon, followed by a horizontal line and navigation links for Features, What's New, Roadmap, and Pricing. To the right are 'SIGN UP' and 'LOG IN' buttons. A central heading reads 'AS A SERVICE, SONARCLOUD.' followed by 'clean code in your cloud workflow with {SonarCloud}'. Below this, a subtext says 'Enable your team to deliver clean code consistently and efficiently with a tool that easily integrates into the cloud DevOps platforms and extend your CI/CD workflow.' On the left side of the main content area, there's a small icon of a person with a plus sign. On the right, there's a blue circular profile icon with a message bubble that says 'Hey 😊 thanks for stopping by! What brought you to our site today?'. In the bottom left corner of the main content area, there's a white call-to-action box with a purple border. It contains the text 'Try SonarCloud for free' and '1-CLICK SIGNUP WITH:' followed by four buttons for GitHub, Bitbucket, GitLab, and Azure DevOps.



This screenshot shows the SonarCloud landing page specifically for GitHub integration. The main title 'Try SonarCloud for free' is at the top, followed by the '1-CLICK SIGNUP WITH:' section with the GitHub button highlighted with a red border. Below this are buttons for Bitbucket, GitLab, and Azure DevOps. To the right, there's a section titled 'YOUR TEAM COMPANION FOR CLEAN CODE' with three icons: a purple gear for 'free' (for open-source projects), a lightning bolt for 'fast' (project onboarding and analysis), and a target for 'precise' (and actionable results). At the bottom, there's a 'TRUSTED BY' section with logos for Microsoft, Apache, Wikimedia Foundation, brave, and TYPO3.

Ensure that you are logged into the GitHub account from the same browser in order to provide

authorization. Follow the prompts to authorize SonarCloud with GitHub.

From the welcome page, click **Import an organization from GitHub**.

The screenshot shows the SonarCloud homepage. At the top, there's a navigation bar with links for 'My Projects', 'My Issues', 'Explore', and a search icon. To the right of the search are icons for notifications, help, and account management. The main area features a large orange cloud icon. Below it, the text 'Welcome to SonarCloud' is displayed in bold, followed by a subtitle 'Let us help you get started in your journey to code quality'. There are two main sections: 'Analyze your first projects' with a 'Import an organization from GitHub' button (which is highlighted with a red box), and 'Join an organization' with a 'Learn More' link.

Choose your GitHub organization.

A screenshot of the SonarCloud 'Install' screen. At the top center is a large orange circular icon containing a white cloud-like shape. Below the icon, the text 'Install SonarCloud' is displayed in a large, bold, dark font. Underneath that, the question 'Where do you want to install SonarCloud?' is shown in a smaller, gray font. At the bottom left, there is a user profile box with a small purple and pink checkered icon next to the text 'eeganlf'. A red rectangular border highlights this user profile area.

Provide access to all your repositories:

## Install SonarCloud

Install on your personal account eeganlf 

**All repositories**  
This applies to all current *and* future repositories owned by the resource owner.  
Also includes public repositories (read-only).

**Only select repositories**  
Select at least one repository.  
Also includes public repositories (read-only).

with these permissions:

✓ **Read** access to code and metadata

✓ **Read and write** access to checks, commit statuses, pull requests, and security events

User permissions  
SonarCloud can also request users' permission to the following resources. These permissions will be requested and authorized on an individual-user basis.

✓ **Read** access to email addresses

**Install** **Cancel**

Next: you'll be directed to the GitHub App's site to complete setup.

You will be redirected back to SonarCloud.

Provide an organization name to be created on SonarCloud. This is important, as you are going

to use it as a reference later when you integrate it with Jenkins. Your GitHub username should be auto-populated. If not, enter it into the **Key** box. Click **Continue**.

**Create an organization**

An organization is a space where a team or a whole company can collaborate across many projects.

1 Import organization details

Import eeganlf into a SonarCloud organization X

Key \* eeganlf

Organization key must start with a lowercase letter or number, followed by lowercase letters, numbers or hyphens, and must end with a letter or number. Maximum length: 255 characters.

Add additional info ▼

All members from your GitHub organization eeganlf will be added to your SonarCloud organization. As they connect to SonarCloud with their GitHub account, members will automatically have access to your SonarCloud organization and its projects. [See all members on GitHub](#)

**Continue**

Since your repository is public, you can proceed with the free plan.

2 Choose a plan

Free	Team	Enterprise
For individual developers, students and open-source projects	Essential capabilities for small teams and business	Mission critical flexibility, scalability, and performance.
\$0	Starting at <del>\$64</del> \$32/month - 50% off	To get started <a href="#">Talk to sales</a>
<b>Select Free</b>	<b>Start Free Trial</b>	<b>Contact sales</b>
<b>What's included</b>	<b>What's included</b>	<b>What's included</b>
<ul style="list-style-type: none"> <li>Up to 50K private lines of code</li> <li>Up to 5 members</li> <li>Unlimited public lines of code</li> </ul>	<ul style="list-style-type: none"> <li>Up to 1.9M private lines of code</li> <li>Unlimited members</li> <li>Unlimited public lines of code</li> </ul>	<ul style="list-style-type: none"> <li>From 5M private lines of code</li> <li>Unlimited members</li> <li>Unlimited public lines of code</li> </ul>
<b>Key features</b>	<b>Everything in Free and</b>	<b>Everything in Team and</b>
<ul style="list-style-type: none"> <li>Open-source and private projects</li> <li>30 languages supported</li> <li>Issue detection and SAST</li> <li>Analyze the main branch &amp; pull requests</li> <li>DevOps platform integration</li> </ul>	<ul style="list-style-type: none"> <li>14-day free trial, cancel any time</li> <li>Open-source and private projects</li> <li>Analyze feature branches, maintenance branches, &amp; pull requests</li> <li>Define the quality standard for your team</li> <li>Advance issue detection</li> <li>Deeper SAST</li> <li>Synchronized user management</li> </ul>	<ul style="list-style-type: none"> <li>Enterprise-Level Hierarchy</li> <li>Secure SAML single sign-on (SSO)</li> <li>Management reporting</li> <li>Centralized enterprise billing</li> <li>Enterprise-grade security</li> <li>AWS Marketplace transaction option</li> <li>6 additional enterprise languages: ABAP, APEX, COBOL, JCL, PL/I, RPG</li> </ul>
<b>Create Organization</b>	<b>Available packs</b>	<b>Premium Support</b>

Choose the repositories that you want to analyze with SonarQube. Select **maven-examples** and the **example-voting-app**. Click **Set Up**:

Analyze projects - Select repositories

Organization\*

eeganlf eeganlf [Import another organization](#)

Select all available repositories  Search for repositories...

- argocd-example-apps
- armory-minnaker
- devops-repo
- helm-charts-lfs269-original
- json-server
- LFS261-example-voting-app
- LFS269-helm-charts
- localtunnel
- maven-examples

**2 repositories selected**  
2 repositories will be created as public projects on SonarCloud [Set Up](#)

If you are not redirected to your **My Projects** page in SonarCloud, navigate there now via the **My Projects** link.

sonarcloud [My Projects](#) My Issues Explore [New](#)

**Set up 2 projects for Clean as You Code**

The new code definition sets which part of your code will be considered new code.

This helps you focus attention on the most recent changes to your project, enabling you to follow the Clean as You Code methodology.

Learn more: [New Code Definition](#)

Set a new code definition for your organisation to use it by default for all new projects

This can help you use the Clean as You Code methodology consistently across projects.  
[eeganlf - Administration - New Code](#)

This is the page that lists all of your projects.

The screenshot shows the SonarCloud web interface with the 'My Projects' tab selected. On the left, there's a sidebar with 'Filters' and sections for 'Quality Gate', 'Reliability', and 'Security'. The main area displays two projects: 'eeganlf / LFS261-example-voting-app' and 'eeganlf / maven-examples'. The 'maven-examples' project is highlighted with a red box. It shows a 'NEW' status and 'PUBLIC'. Below it, there's a note about extra configuration steps required and a link to 'Configure analysis'. The project was last analyzed on 11/14/2022 at 8:14 PM. The analysis results show 5 bugs (D), 0 vulnerabilities (A), 100% hotspots reviewed, 92 code smells (A), and 0.0% duplications. A 'Not computed' badge is present next to the analysis results.

Click on the **maven-examples** project:

The screenshot shows the SonarCloud project details page for 'eeganlf / maven-examples'. The project is marked as 'NEW' and 'PUBLIC'. It was last analyzed on 11/14/2022 at 8:14 PM, covering 3.3k lines of code in XML and JavaScript. The analysis results are displayed in a grid: 5 bugs (D), 0 vulnerabilities (A), 100% hotspots reviewed, 92 code smells (A), and 0.0% duplications. A 'Not computed' badge is visible next to the analysis results.

You will be taken to the **maven-examples** project page. You must disable automatic analysis as you want Jenkins to tell SonarCloud when to analyze your code.

You must also configure your project. To accomplish this, click navigate to **Administration > Analysis Method**.

The screenshot shows the SonarCloud interface for the 'maven-examples' project. On the left, the navigation bar has a red box around the 'Administration' option. A dropdown menu is open under 'Analysis Method', with its title also highlighted by a red box. The main content area displays project statistics and a status message: 'Not computed'.

Uncheck **Automatic Analysis** and click **Set up analysis via other methods > Manually**:

The screenshot shows the 'Analysis Method' configuration page. The 'Automatic Analysis' toggle switch is highlighted with a red box. Below it, the 'Set up analysis via other methods' section is shown, with the 'Manually' option highlighted by a red box. The page includes a note about integrating with CI tools like GitHub Actions, Travis CI, CircleCI, and Amazon CodeCatalyst.

Under **What option best describes your build?** select **Other**. Select **Linux** for the operating system.

Analyze from your local sources

Run analysis on your project

What option best describes your build?

Maven   Gradle   .NET   C, C++ or ObjC   **Other (for JS, TS, Go, Python, PHP, ...)**

Which OS do you run your build on?

**Linux**   Windows   macOS

Download and unzip the SonarScanner for Linux  
And add the `bin` directory to the `PATH` environment variable  
[Download](#)

Configure the `SONAR_TOKEN` environment variable

- 1 Name of the environment variable: `SONAR_TOKEN`
- 2 Value of the environment variable: `f21b498221705bb506ac8b21aee7f6c75acce1ef`

Execute the SonarScanner from your computer  
Run the following command in your project's folder.

```
sonar-scanner \
  -Dsonar.organization=eeganlf \
  -Dsonar.projectKey=eeganlf_maven-examples \
  -Dsonar.sources= \
  -Dsonar.host.url=https://sonarcloud.io
```

Please visit the [SonarScanner documentation](#) for more details.

Notice that a command is generated. This command contains configuration properties that we will use to integrate SonarQube with Jenkins.

## Integrate SonarQube with Jenkins

Install the SonarQube Scanner plugin for Jenkins.

Install	Name ↓	Released
<input checked="" type="checkbox"/>	<a href="#">SonarQube Scanner</a> 2.17.2	5 mo 13 days ago
<a href="#">External Site/Tool Integrations</a> <a href="#">Build Reports</a>		
This plugin allows an easy integration of <a href="#">SonarQube</a> , the open source platform for Continuous Inspection of code quality.		

From Jenkins > Manage Jenkins > Tools > SonarQube Scanner click Add SonarQube Scanner:

## SonarQube Scanner

### SonarQube Scanner installations

List of SonarQube Scanner installations on this system

[Add SonarQube Scanner](#)

Name the SonarQube Scanner **SonarScanner** which will be referred to later when you write the pipeline code. Also, choose whichever is the latest version, which should be automatically selected from the dropdown menu:

The screenshot shows the Jenkins management interface for tools. Under 'SonarQube Scanner installations', a new entry is being created. The 'Name' field contains 'SonarScanner'. The 'Version' dropdown is set to 'SonarQube Scanner 7.1.0.4889'. The 'Install automatically' checkbox is checked. Below the form, there are sections for 'Ant installations' and 'Maven installations'.

From the Jenkins home, navigate to **Manage Jenkins > Security > Credentials**:

The screenshot shows the Jenkins 'Credentials' page under 'Manage Jenkins > Security > Credentials'. There are three entries listed:

T	P	Store ↓	Domain	ID	Name
🔑	👤	System	(global)	0d47ee8b-691f-4e9e-8b9e-02049e396a56	Secret text
🔑	👤	System	(global)	jenkins-github-access-token	jenkins-github-access-token
📄	👤	System	(global)	github-auth-token	eeganlf***** (auth token for github)

### Stores scoped to Jenkins

The screenshot shows the 'Domains' section under 'Stores scoped to Jenkins'. It lists a single domain entry:

P	Store ↓	Domains
👤	System	(global)

Click **Global credentials (unrestricted)**:

The screenshot shows the 'System' global credentials page. It displays one entry:

Domain ↓	Description
👑 Global credentials (unrestricted)	Credentials that should be available irrespective of domain specification to requirements matching.

Icon: S M L

Click **Add Credentials**:

**Global credentials (unrestricted)**

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
0d47ee8b-691f-4e9e-8b9e-02049e396a56	Secret text	Secret text	
jenkins-github-access-token	jenkins-github-access-token	Secret text	
github-auth-token	eeganlf***** (auth token for github)	Username with password	auth token for github

Icon: S M L

Add new credentials with:

- Kind “Secret text”.
- Copy the value for the **SONAR\_TOKEN** found on the configuration page on SonarCloud for the **maven-examples** project:

ifl > maven-examples > Configure

1 Name of the environment variable: SONAR\_TOKEN

2 Value of the environment variable: f21b498221705bb506ac8b21aee7f6c75acce1ef

Execute the SonarScanner from your computer

Run the following command in your project's folder.

```
sonar-scanner \
-Dsonar.organization=eeganlf \
-Dsonar.projectKey=eeganlf_maven-examples \
-Dsonar.sources=. \
-Dsonar.host.url=https://sonarcloud.io
```

Copy

Please visit the [SonarScanner documentation](#) for more details.

- Paste it in the “Secret” field on Jenkins.
- Add ID and Description as **sonar-maven-examples**.

Refer to the following screenshot while adding credentials:

The screenshot shows the Jenkins 'New credentials' configuration page. The 'Kind' dropdown is set to 'Secret text'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Secret' field contains redacted text. The 'ID' field is labeled 'sonar-maven-examples'. The 'Description' field is also labeled 'sonar-maven-examples'. A red box highlights the 'create' button at the bottom.

Now navigate to **Jenkins > Manage Jenkins > System > SonarQube Servers** and add a new entry.

The screenshot shows the Jenkins 'SonarQube servers' configuration page. It includes a section for 'Environment variables' with an unchecked checkbox. Below this is a 'SonarQube installations' section with a 'List of SonarQube installations' link and a red-highlighted 'Add SonarQube' button.

Provide the configuration as shown below:

- Check the **Environment Variables** box to enable injection of configurations.
- Name this instance **sonar-maven-examples** as it will be referred to later.
- Use the [SonarQube URL](#) but do **NOT** include a trailing slash or it will not work.
- Select the credentials you added in the previous step from the dropdown, i.e. **sonar-maven-examples**.

- Save the configuration.

SonarQube installations

List of SonarQube installations

Name	<input type="text" value="sonar-maven-examples"/>	<span style="color: red;">X</span>
Server URL	<input type="text" value="Default is http://localhost:9000"/> <input type="text" value="https://sonarcloud.io"/>	
Server authentication token	<input type="text" value="sonar-maven-examples"/>	<span style="color: red;">▼</span>
<a href="#">+ Add ▾</a>		
<a href="#">Advanced ▾</a>		

## Scanning Code and Reporting to SonarQube

In Jenkins, go to the configuration page for the **maven-examples** job that you created to use code coverage and add a post build step. Select **Execute Sonarqube Scanner**.

Dashboard > maven-examples > Configuration

**Configure**

**Post Steps**

Run only if build succeeds

Run only if build succeeds or is unstable

Run regardless of build result

Should the post-build steps run only for successful builds, etc.

[Add post-build step ▾](#)

**Filter**

- Add a new template to all docker clouds
- Build / Publish Docker Image
- Execute Node.js script
- Execute SonarQube Scanner
- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Provide Configuration files
- Run with timeout

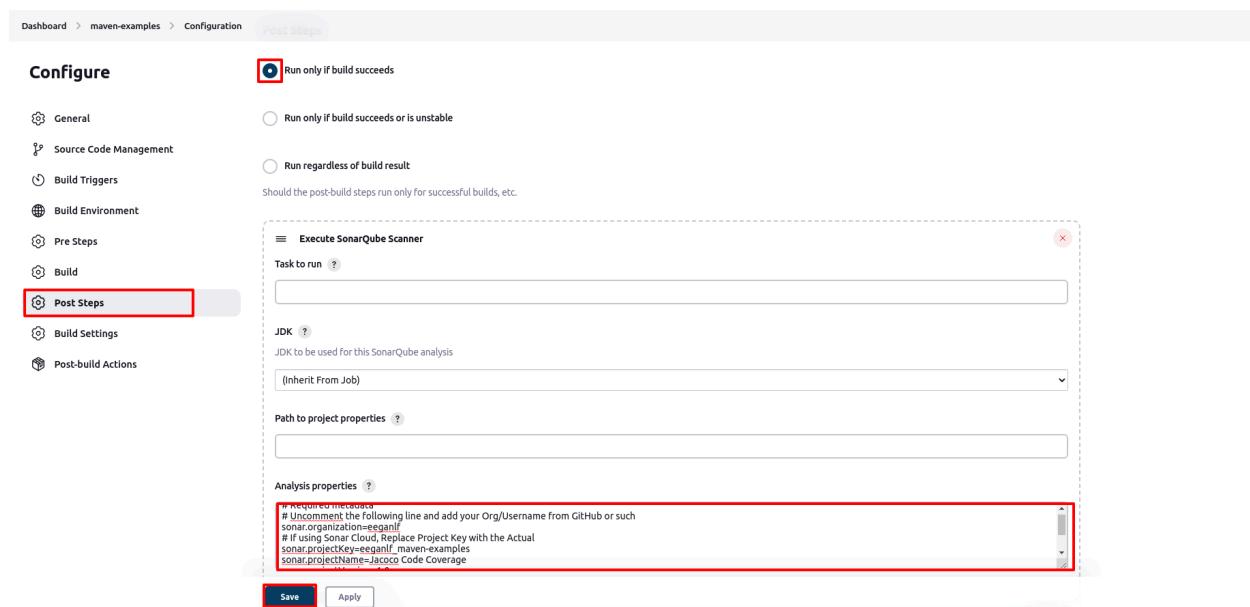
Put the following [snippet](#) into the **Analysis properties** text box:

```

# Required metadata
# Uncomment the following line and add your Org/Username from GitHub
# or such
sonar.organization=YourGitHubOrg
# If using Sonar Cloud, Replace Project Key with the Actual

sonar.projectKey=yourGitHubOrg_ProjectName
sonar.projectName=Jacoco Code Coverage
sonar.projectVersion=1.0
# Comma-separated paths to directories with sources (required)
sonar.sources=code-coverage-jacoco/src/main
# Encoding of the source files
sonar.sourceEncoding=UTF-8
sonar.java.binaries=
sonar.coverage.jacoco.xmlReportPaths=code-coverage-jacoco/target/site/
jacoco-ut/jacoco.xml

```



Ensure you replace the properties for `sonar.organization` and `sonar.projectKey` with what you see in the **Sonar Cloud Analysis Configuration** page:

Execute the SonarScanner from your computer

Run the following command in your project's folder.

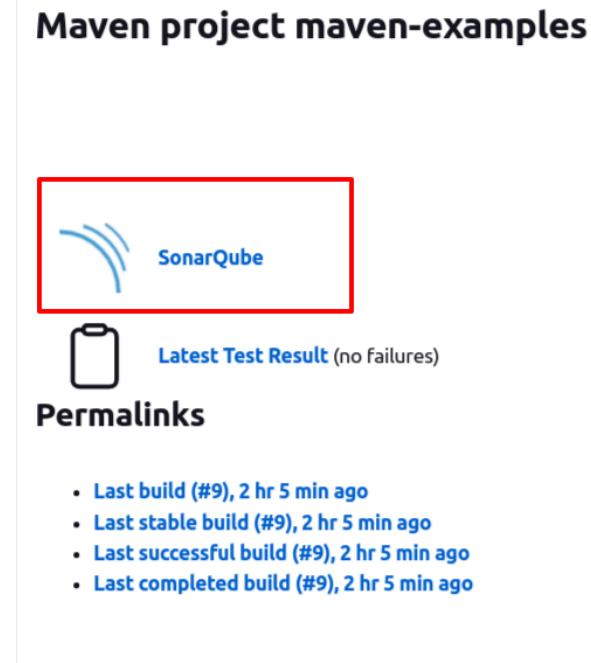
```
sonar-scanner \
-Dsonar.organization=eeganlf \
-Dsonar.projectKey=eeganlf_maven-examples \
-Dsonar.sources=. \
-Dsonar.host.url=https://sonarcloud.io
```

 Copy

Please visit the [SonarScanner documentation](#) for more details.

Back in Jenkins, build the project. Click the SonarQube link to navigate to SonarQube:

**Maven project maven-examples**



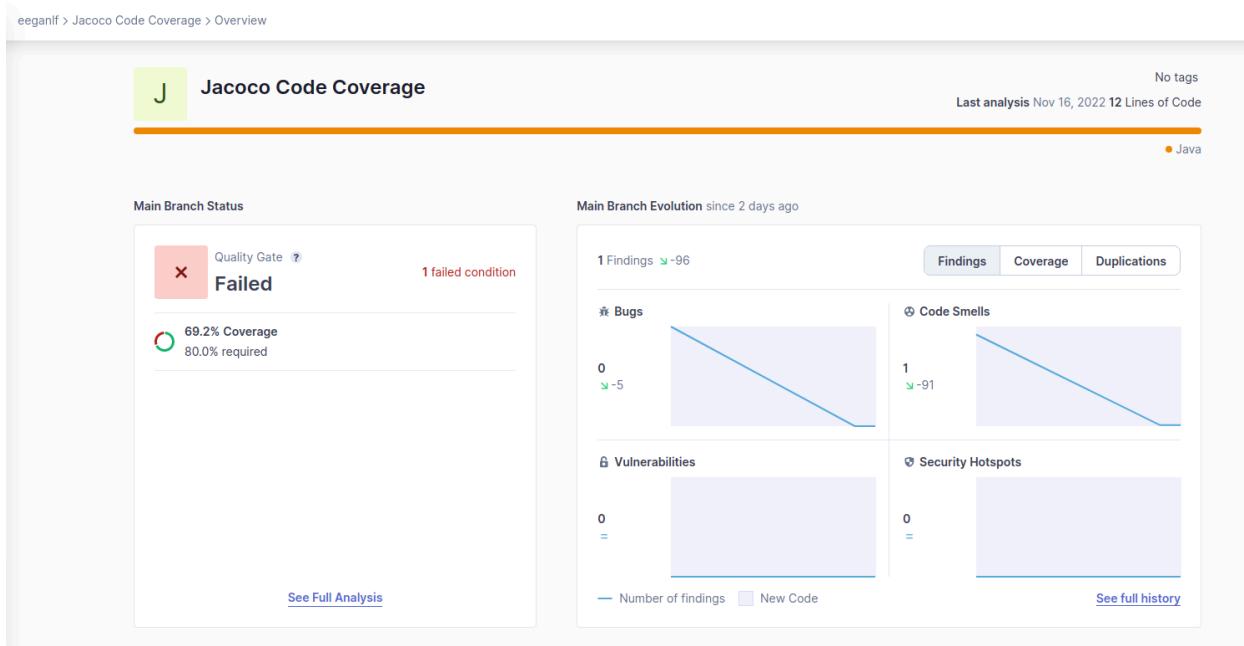
SonarQube

Latest Test Result (no failures)

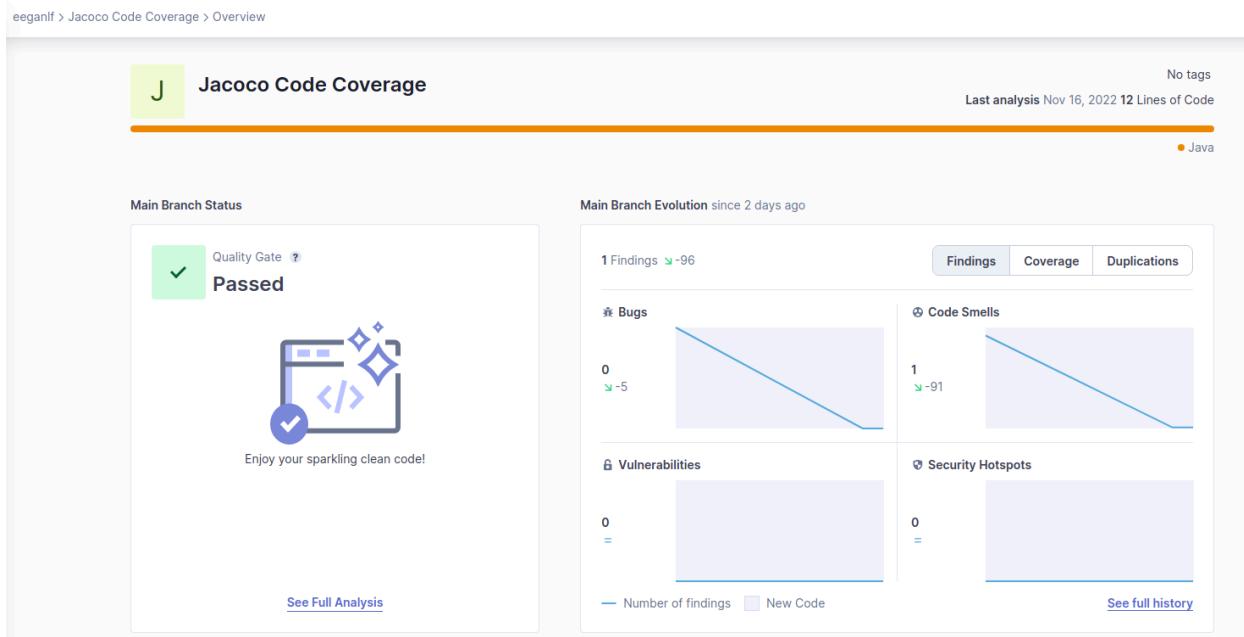
Permalinks

- Last build (#9), 2 hr 5 min ago
- Last stable build (#9), 2 hr 5 min ago
- Last successful build (#9), 2 hr 5 min ago
- Last completed build (#9), 2 hr 5 min ago

You will see that sonarcloud.io shows that the project failed and displays the reason why it failed. In this case, it failed because we have less than 80% test coverage.



Back on the GitHub repository for **maven-examples**, merge the **ut2** branch into the master and run **Build Now** in Jenkins on the **maven-examples** job. After it is complete, follow the **SonarQube** link. The job should now be displayed as passed:



Note that this does not keep Jenkins from successfully building the job. It is an illustration of

how quality gates can be configured to output a pass or fail result for a particular job. We do, however, want to enable SonarQube to fail builds before they get to production. We can and should do this in the Jenkinsfile. We will accomplish this in the next section for the `example-voting-app`.

## Adding SonarQube Scanner Stage to Jenkinsfile

You want SonarQube to fail the pipeline before the project is deployed if the code does not pass analysis. Add the following code snippet to the consolidated Jenkinsfile created earlier while setting up a monopipe. Place the code above the `stage ('deploy to dev')` block of code.

```
file: example-voting-app/Jenkinsfile

stage('Sonarqube') {
    agent any
    when{
        branch 'master'
    }
    environment{
        sonarpath = tool 'SonarScanner'
    }
    steps {
        echo 'Running Sonarqube Analysis..'
        withSonarQubeEnv('sonar-instavote') {
            sh "${sonarpath}/bin/sonar-scanner
-Dproject.settings=sonar-project.properties
-Dorg.jenkinsci.plugins.durabletask.BourneShellScript.HEARTBEAT_CHECK_
INTERVAL=86400"
        }
    }
}

stage("Quality Gate") {
    steps {
        timeout(time: 1, unit: 'HOURS') {
            // Parameter indicates whether to set pipeline to
UNSTABLE if Quality Gate fails
            // true = set pipeline to UNSTABLE, false = don't
waitForQualityGate abortPipeline: true
        }
    }
}
```

You can access this snippet [here](#).

Also update `sonar-project.properties` file at the root of `example-voting-app` repository with *your* organization and project key as per configured on SonarCloud.

**File: example-voting-app/sonar-project.properties**

```
# Uncomment and update Org matching your configurations on Sonarcloud
sonar.organization=your-org
sonar.projectKey=your-org_example-voting-app
sonar.projectName=LFS261 example voting app
sonar.projectVersion=1.0

# Comma-separated paths to directories with sources (required)
sonar.sources=worker

# Encoding of the source files
sonar.sourceEncoding=UTF-8

sonar.java.binaries=worker

sonar.coverage.jacoco.xmlReportPaths=worker/target
```

You also need to add Credentials to connect to the project added to SonarCloud. Navigate to **Jenkins > Manage Jenkins > Credentials > System > Global credentials (unrestricted)** and click **Add Credentials**:

The screenshot shows the Jenkins Global credentials (unrestricted) page. The URL in the browser bar is `Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted)`. A red box highlights the "Add Credentials" button. The table lists five credentials:

ID	Name	Kind	Description
Od47ee8b-691f-4e9e-8b9e-02049e396a56	Secret text	Secret text	
jenkins-github-access-token	jenkins-github-access-token	Secret text	
github-auth-token	eeganlf***** (auth token for github)	Username with password	auth token for github
sonar-maven-examples	sonar-maven-examples	Secret text	sonar-maven-examples
dockerlogin	eeganlf***** (dockerlogin)	Username with password	dockerlogin

Find the **SONAR\_TOKEN** on the **LFS261-example-voting-app** configuration page and copy it:

The screenshot shows the Jenkins configuration page for the 'LFS261-example-voting-app' job. The section title is 'Configure the SONAR\_TOKEN environment variable'. There are two numbered steps: 1. Name of the environment variable: SONAR\_TOKEN (with a copy icon) and 2. Value of the environment variable: ecfbe23ecedcc9dcd86279426a54dc03c4960ce2 (with a copy icon and a edit icon). The value field is highlighted with a red box.

Paste this token into the **Secret** text box while configuring the credentials:

The screenshot shows the Jenkins 'New credentials' creation page under 'Manage Jenkins > Credentials > System > Global credentials (unrestricted)'. The 'Kind' dropdown is set to 'Secret text' (highlighted with a red box). The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Secret' field contains a redacted value (highlighted with a red box). The 'ID' field is set to 'sonar-instavote' (highlighted with a red box). The 'Description' field is set to 'sonar-instavote'. At the bottom is a 'Create' button (highlighted with a red box).

Create a new SonarQube Server configuration from **Jenkins > Manage Jenkins > System > SonarQube Servers**.

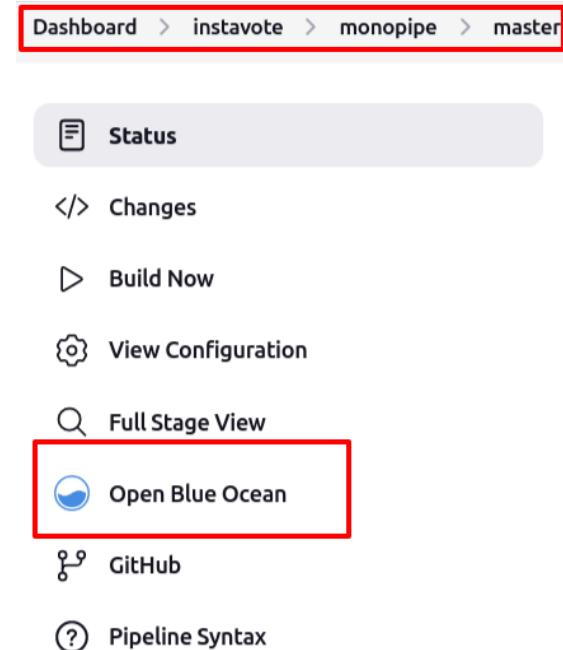
The screenshot shows the Jenkins configuration interface for adding a new SonarQube server. The 'Name' field is set to 'sonar-instavote'. The 'Server URL' field is set to 'https://sonarcloud.io'. The 'Server authentication token' dropdown also contains 'sonar-instavote'. The 'Save' button is highlighted with a red box.

Remember to make sure that the `https://sonarcloud.io` does **NOT** have a trailing slash.

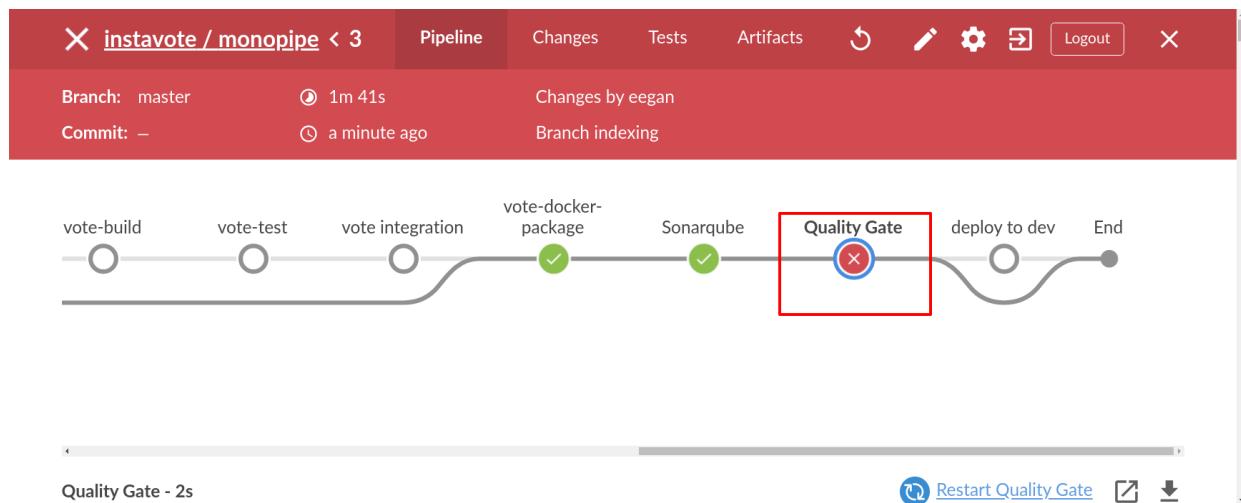
The free quality gate provided by SonarCloud only performs analysis on 20 lines or more of new code. Replace the entirety of the

`example-voting-app/worker/src/main/java/worker/Worker.java` file with this [code](#).

Commit to the master branch and observe the latest jobs being run with the next instance of the pipeline run. To view the build in the alternate BlueOcean interface, select the interface from the side menu of the master branch:

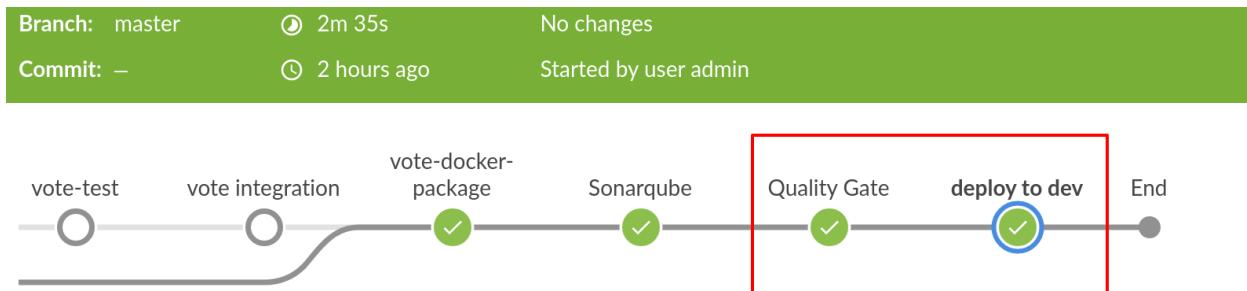


After the master branch builds in the monopipe job, you will notice that it fails at the **Quality Gate** step:



To get our worker app to pass the quality gate, we need to provide test coverage greater than 80%. Replace

[LFS261-example-voting-app/worker/src/test/java/worker/UnitWorker.java](#) with this [code](#).



## Adding Integration Tests

Integration tests for the `vote` Python app are already added to the `vote/` subdirectory of the repository. To add integration tests to the pipeline, follow the steps below.

Examine the script at `vote/integration_test.sh` in the `example-voting-app` repository. Reading the script and the docker compose setup it uses should give you a good understanding of how integration tests are set up. Add a stage to the Jenkinsfile to run the integration tests as follows. Place the below code block just after the stage `vote-test` stage which runs unit tests for the `vote` app:

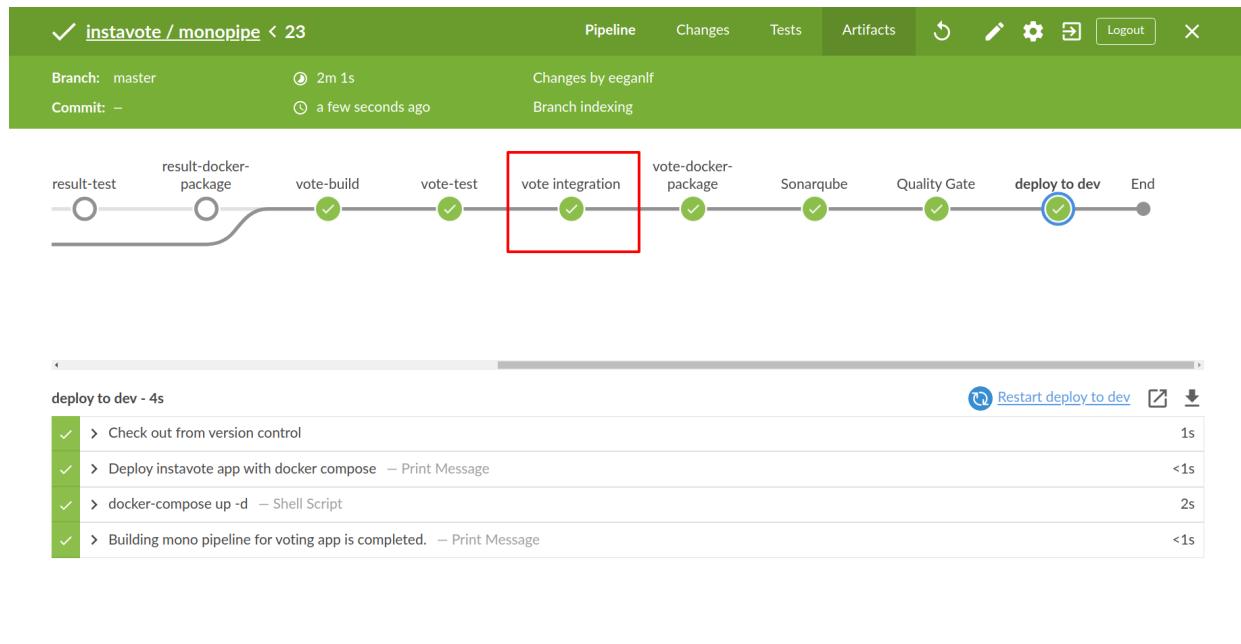
```

file: example-voting-app/Jenkinsfile

stage('vote integration') {
    agent any
    when{
        changeset "**/vote/**"
        branch 'master'
    }
    steps{
        echo 'Running Integration Tests on vote app'
        dir('vote'){
            sh 'sh integration_test.sh'
        }
    }
}
  
```

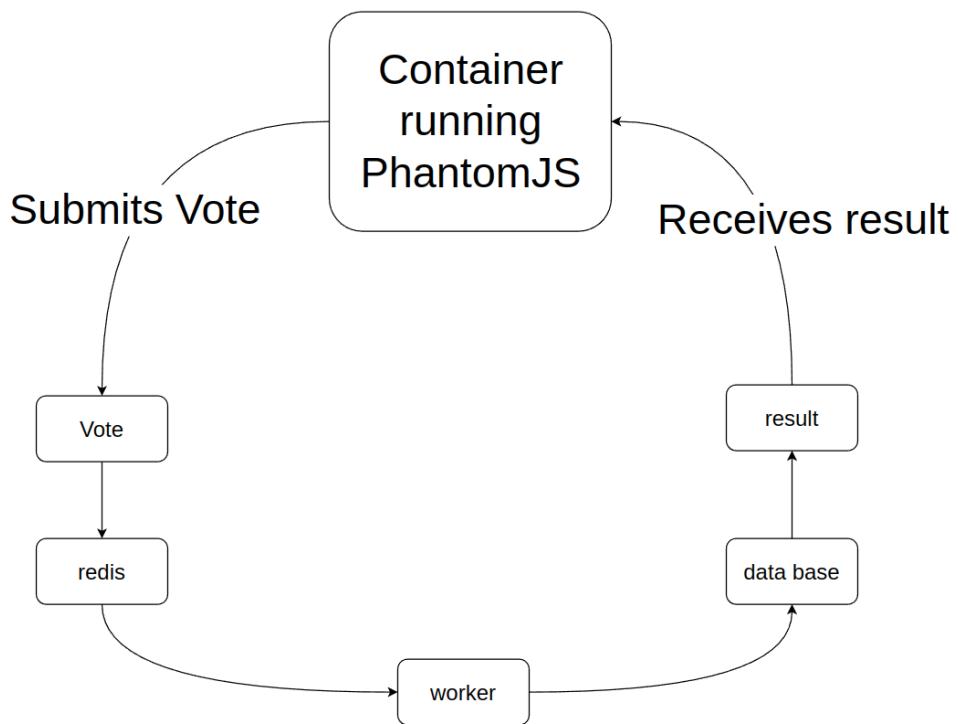
You should see a similar output in the Blue Ocean interface as the screenshot below. This

indicates a successful execution of integration tests for the `vote` app:



## E2E Tests

Examine the script `example-voting-app/e2e.sh` in the `example-voting-app` repository as well as the `example-voting-app/e2e/docker-compose.yml`. You do not need to understand all of the code, but reading the script and the Docker Compose setup it uses will give you a better understanding of how e2e tests are set up.



## The .env File

`example-voting-app/e2e/docker-compose.yml` uses environment variables to decide which images to use to spin up the containers for the e2e tests.

Inside the `example-voting-app/e2e/.env` file you will find the following:

```
VOTE_IMAGE=xxxx/vote:latest
WORKER_IMAGE=xxxx/worker:latest
RESULT_IMAGE=xxxx/result:latest
```

Replace `xxxx` with your own Docker Hub user ID.

## Setting Up E2E

Once the `.env` file is edited, run the following command from inside the `example-voting-app/e2e` which contains the `docker-compose.yml` file:

```
docker compose build
```

---

This will build the images necessary for the e2e test. It also ensures that any changes made to testing files are picked up by Docker Compose. After everything is built, run:

```
docker compose up -d
```

```
[+] Running 8/8
  # Network e2e_back-tier    Created          0.1s
  # Network e2e_front-tier   Created          0.1s
  # Container e2e-db-1      Started          1.8s
  # Container e2e-redis-1   Started          1.9s
  # Container e2e-worker-1  Started          2.8s
  # Container e2e-vote-1    Started          3.1s
  # Container e2e-result-1  Started          3.4s
  # Container e2e-e2e-1     Started          3.9s
```

To list your running containers to find their associated ports, run:

```
sudo docker ps
```

IMAGE	PORTS
xxxxxxxx/result:latest	0.0.0.0:49156->80/tcp, :::49156->80/tcp
xxxxxxxx/vote:latest	0.0.0.0:49155->80/tcp, :::49155->80/tcp

Find the containers associated with the `result:latest` image and the `vote:latest` image. Visit `IPADDRESS:VOTECONTAINERPORT` and `IPADDRESS:RESULTCONTAINERPORT`. Submit a vote. This is manual e2e testing.

To automate the process, we will now spin up the e2e service, which will automatically submit a vote for you. The e2e service is listed in the

`example-voting-app/e2e/docker-compose.yml`. Run:

```
docker compose run --rm e2e
```

```
-----
Current Votes Count: 13
-----
```

```
I: Submitting one more vote...
```

```
-----
New Votes Count: 14
-----
```

```
I: Checking if votes tally.....
```

-----  
**Tests passed**  
-----

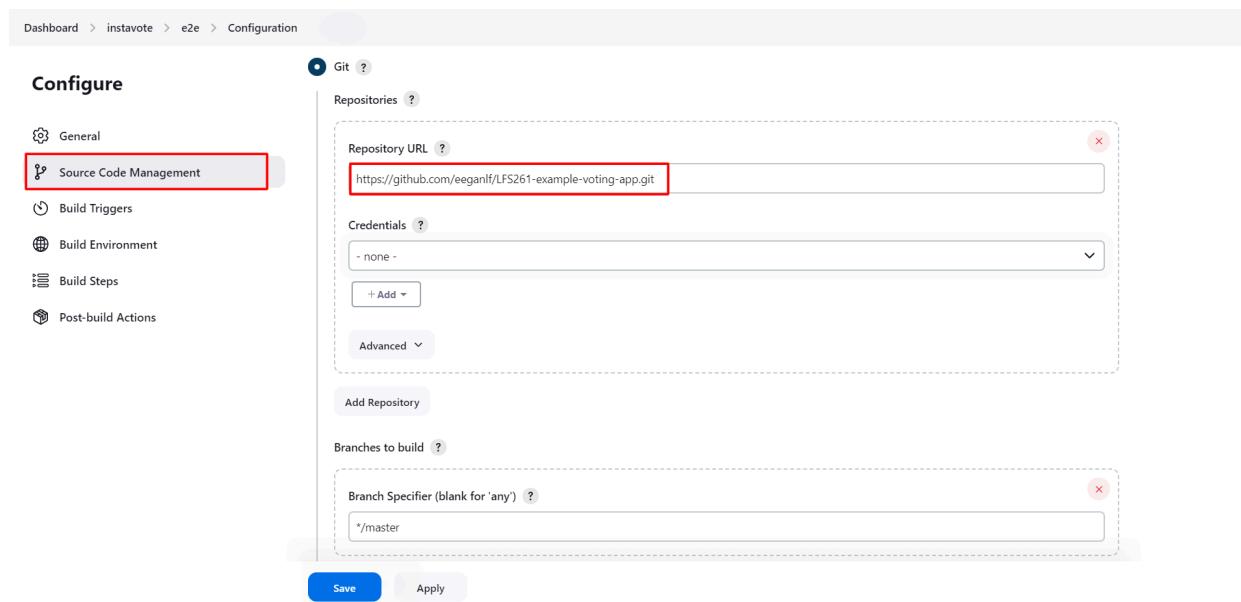
You should see something similar to the output above, though your counts will differ. If it fails, try running it again, and it should pass. This means that the application passed e2e testing. Shut the application down with the following command from inside `example-voting-app/e2e`:

```
docker compose down
```

Using a container to test our application is the first building block to integrating the e2e test into our CI process, which we will do next.

## Integrating E2E Tests with Jenkins

First, create a freestyle project called e2e in Jenkins. Add the Git repository:



Inside the **Build Steps > Add build step > Execute shell > Command** input box paste:  
`./e2e.sh`



Save the configuration.

Run the job and click the build:

The screenshot shows the Jenkins Project e2e dashboard. At the top left is the Jenkins logo and the project name 'Project e2e'. To the right are links for 'Search (CTRL+K)', 'admin', and 'log out'. Below the project name are several buttons: 'Status' (highlighted), 'Changes', 'Workspace', 'Build Now' (highlighted with a red box), 'Configure', 'Delete Project', 'Favorite', 'Move', 'Open Blue Ocean', and 'Rename'. To the right of these buttons is a 'Permalinks' section and a 'Disable Project' button. Below these are sections for 'Build History' (with a 'trend' dropdown) and 'Atom feeds' (links for 'Atom feed for all' and 'Atom feed for failures').

Click **Console output** to see the console output:

The screenshot shows a Jenkins build page for a project named 'instavote' under the 'e2e' job. The build number is #1, and it was started 41 seconds ago. A progress bar indicates the build is still executing. A red box highlights the 'Console Output' link in the left sidebar.

**Build #1 (Nov 23, 2022, 11:47:27 PM)**

Progress: Keep this build forever

Started 41 sec ago  
Build has been executing for 41 sec

Add description

**Console Output**

Status: No changes.

Changes: No changes.

Build Information: Edit Build Information

Git Build Data: Git

Open Blue Ocean

Revision: a997b50c8a53cd971e634826df19fc72f43cebb8  
Repository: <https://github.com/eeganlf/LFS261-example-voting-app.git>

refs/remotes/origin/master

If the tests fail, try running the job twice more. You should ultimately see something like the following:

```
Current Votes Count: 7
-----
I: Submitting one more vote...
-----
New Votes Count: 8
-----
I: Checking if votes tally.....
-----
[92mTests passed
-----
Stopping e2e_vote_1 ...
Stopping e2e_worker_1 ...
Stopping e2e_result_1 ...
Stopping e2e_redis_1 ...
Stopping e2e_db_1 ...
[[2B][1A][2K
Stopping e2e_db_1 ... [[32mdone][0m
[[1B]Removing e2e_e2e_1 ...
Removing e2e_vote_1 ...
Removing e2e_worker_1 ...
Removing e2e_result_1 ...
Removing e2e_redis_1 ...
Removing e2e_db_1 ...
[[1A][2K
Removing e2e_db_1 ... [[32mdone][0m
[[1B][2A][2K
Removing e2e_redis_1 ... [[32mdone][0m
[[2B][5A][2K
Removing e2e_vote_1 ... [[32mdone][0m
[[5B][6A][2K
Removing e2e_e2e_1 ... [[32mdone][0m
[[6B][3A][2K
Removing e2e_result_1 ... [[32mdone][0m
[[3B][4A][2K
Removing e2e_worker_1 ... [[32mdone][0m
[[4B]Removing network e2e_back-tier
[REMOVING NETWORK E2E_BACK-TIER
Finished: SUCCESS
```

You should see that the tests passed and then you should see the containers being cleaned up with a **SUCCESS** message at the bottom of the output. You have successfully run e2e tests with Jenkins.

## Summary

In this lab, we integrated Jacoco into Jenkins to obtain automatic testing coverage analysis. We then integrated SonarQube into Jenkins for automatic code analysis. Finally, we added integration and e2e tests into Jenkins.



## Lab 10. Running Containers at Scale with Kubernetes

By the end of this lab exercise, you should be able to:

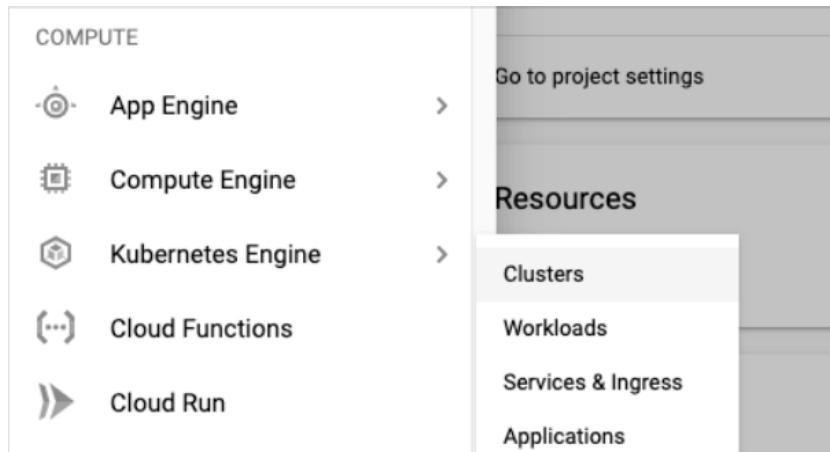
- Set up a managed Kubernetes environment with Google Cloud
- Set up a `kubectl` client, connect and work with the Kubernetes cluster
- Deploy applications to the Kubernetes cluster and expose them as services

### Set Up Kubernetes Cluster with GCP

Set up a Google Cloud account by browsing to <https://cloud.google.com/free>.

Once setup, login to your account and browse to the cloud console by visiting <https://console.cloud.google.com/>.

From **Navigation** select **View All Products > Compute > Kubernetes Engine > Clusters**.



If it asks you to enable Kubernetes Engine API, do so by clicking on the **Enable** button.

The screenshot shows the Google Cloud console interface. At the top, there's a blue header bar with the Google Cloud logo and a CI-CD dropdown menu. Below the header, a back arrow is visible. The main content area has a blue hexagonal icon representing Kubernetes. To its right, the text "Kubernetes Engine API" is displayed in large bold letters, with "Google Enterprise API" in smaller text below it. A descriptive paragraph follows: "Builds and manages container-based applications, powered by the open source Kubernetes technology." At the bottom of this section are two buttons: a blue "ENABLE" button with a red border, and a white "TRY THIS API" button with blue text and a small icon.

You may also be asked to set up a Billing Account, which you should set up.

## Billing required

Kubernetes Engine API requires a project with a billing account.

CANCEL **ENABLE BILLING**

## Set the billing account for project “CI-CD”

Please select a Google Cloud Platform billing account to support this project. Some billing accounts may not be available. [Learn more](#)

Billing account \* –  
My Billing Account



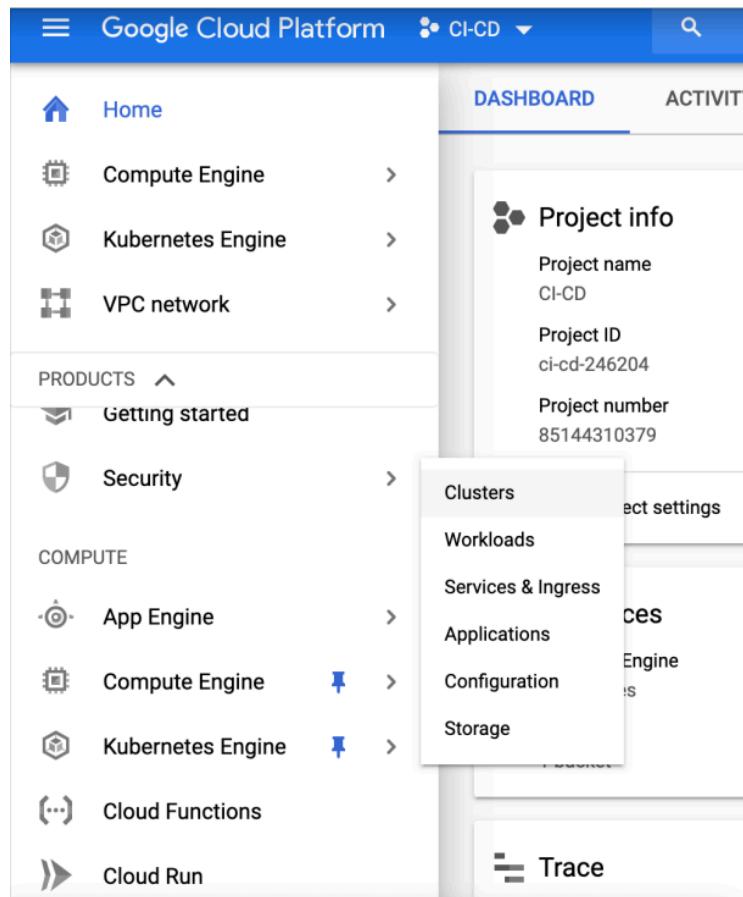
Any charges for this project will be billed to the account you select here.

CANCEL

**SET ACCOUNT**

Be aware that if you are signing up for a Free Trial with some free credits, enabling a billing account does not automatically charge you. When you run out of free credits, you will be given some time to allow Google Cloud to charge your credit card. Your resources may be deleted if you run out of free credits and do not allow Google Cloud to charge your credit card.

Once you have completed these steps, go to **Google Cloud Console**, select **Kubernetes Engine > Clusters** and you will arrive at the following page, ready to launch a cluster.



Kubernetes Engine

## Kubernetes clusters

Containers package an application so it can easily be deployed to run in its own isolated environment. Containers are run on Kubernetes clusters. [Learn more](#)

**CREATE**

DEPLOY CONTAINER

TAKE THE QUICKSTART

Click on **Create Cluster**, choose **Standard**.

The screenshot shows the 'Create an Autopilot cluster' page in the Google Cloud Platform. On the left, there's a sidebar with a list of steps: Cluster basics, Fleet registration, Networking, Advanced settings, and Review and create. The 'Cluster basics' step is currently selected. The main content area is titled 'Cluster basics' and contains instructions: 'Create an Autopilot cluster by specifying a name and region. After the cluster is created, you can deploy your workload through Kubernetes and we'll take care of the rest, including:'. It lists four features with green checkmarks: Nodes, Networking, Security, and Telemetry. Below this, there's a 'Name' input field containing 'autopilot-cluster-1'. A note below the input says: 'Cluster names must start with a lowercase letter followed by up to 39 lowercase letters, numbers, or hyphens. They can't end with a hyphen. You cannot change the cluster's name once it's created.' At the bottom, there are 'CREATE' and 'CANCEL' buttons, along with links for 'Equivalent REST or COMMAND LINE'.

This opens up a cluster configuration page similar to below. Provide the name for the cluster, e.g. **cd**, and select **Regular Channel** for control plane version. Do note, the actual version that you see may differ from what is shown in the screenshots.

The screenshot shows the 'Create a Kubernetes cluster' page in the Google Cloud Platform. On the left, there's a sidebar with a list of steps: Cluster basics, Fleet registration, NODE POOLS, default-pool, CLUSTER, Automation, Networking, Security, Backup plan, Metadata, and Features. The 'Cluster basics' step is currently selected. The main content area is titled 'Cluster basics' and contains instructions: 'The new cluster will be created with the name, version, and in the location you specify here. After the cluster is created, name and location can't be changed.' It has fields for 'Name' (containing 'cd'), 'Location type' (set to 'Zonal'), 'Zone' (set to 'us-west1-c'), and a checkbox for 'Specify default node locations'. Below this, there's a 'Release channel' section with a dropdown set to 'Regular (recommended)' and a 'Version' dropdown set to '1.32+gke.1353003 (default)'. A note at the bottom says: 'Versions in the Regular channel have been qualified over a longer period. They offer a balance of feature availability and release stability. We recommend the Regular channel for most users. For known issues and workarounds, review release notes.' At the bottom, there are 'Create', 'Cancel', and 'Code equivalent' buttons.

Click on **Create** and wait for the cluster to be ready.

## Connecting to Your Cluster

There are two ways to connect to your cluster. Either way will end up with you interacting with your cluster via a terminal in which you will run the commands given.

The first and easier way to connect to your cluster is via the **RUN IN CLOUD SHELL** option. The second is to manually connect to your cluster via your own machine.

We will go over both but it is suggested you use the easier method.

### Connect to Your Cluster Via **RUN IN CLOUD SHELL**

Navigate to **Kubernetes Engine > Clusters > click on your cluster name**, as follows:

In the horizontal menu across the top of the screen you will see the **CONNECT** option. Click it:

Cluster basics	
Name	cd
Location type	Zonal
Control plane zone	us-central1-c
Default node zones	us-central1-c

A pop up window will appear. Click **RUN IN CLOUD SHELL**:

## Connect to the cluster

You can connect to your cluster via command-line or using a dashboard.

### Command-line access

Configure [kubectl](#) command line access by running the following command:

```
$ gcloud container clusters get-credentials cd --zone us-central1-c --project ci-cd-349400
```

[RUN IN CLOUD SHELL](#)

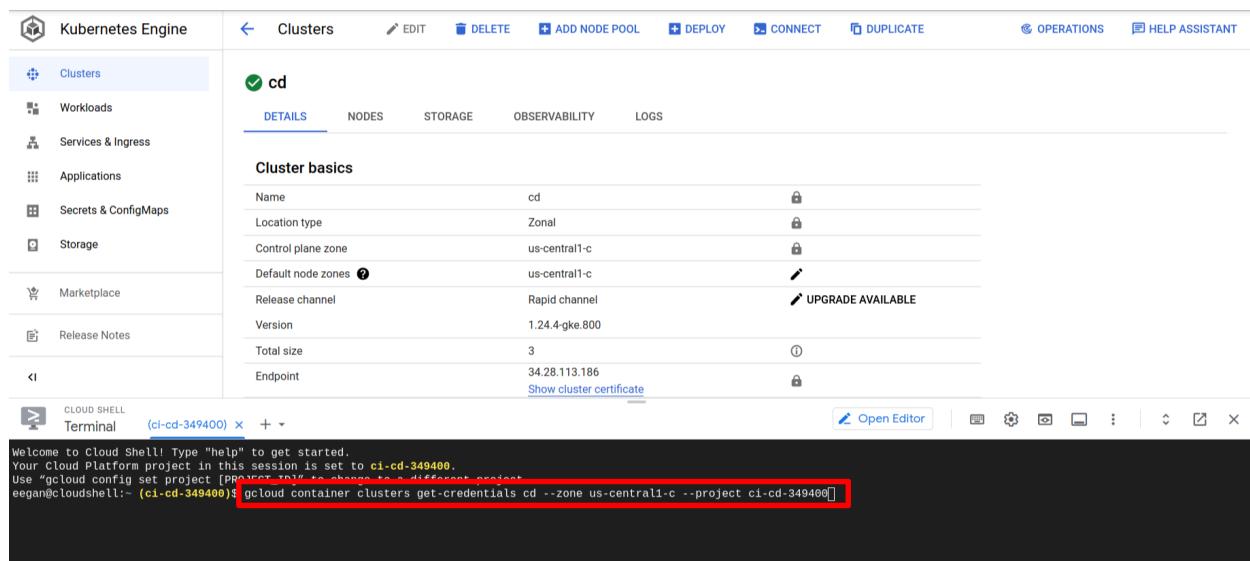
### Cloud Console dashboard

You can view the workloads running in your cluster in the Cloud Console [Workloads dashboard](#).

[OPEN WORKLOADS DASHBOARD](#)

OK

A terminal will appear with the **gcloud** command to connect to your cluster pre-populated.



Press **Enter** and you will be connected to your cluster via a terminal.

## Manual Connection to Your Cluster

To connect to your cluster via your own machine, use the following:

- Read the [Google SDK installation guide](#).
- Install Google SDK with [interactive option](#).
- Install [kubectl](#) by referring to the documentation relevant to your OS.

Navigate to **Kubernetes Engine > Clusters** and click on the name of your cluster.

	Status	Name ↑	Location	Number of nodes	Total vCPUs
<input type="checkbox"/>	<input checked="" type="checkbox"/> cd	cd	us-central1-c	3	6

Click on **CONNECT**. If you don't see it, you are likely zoomed in too far in your browser settings and it will be under the vertical three-dot menu on the right.

DETAILS	NODES	STORAGE	OBSERVABILITY	LOGS																											
<b>Cluster basics</b> <table border="1"> <tbody> <tr> <td>Name</td> <td>cd</td> <td>🔒</td> </tr> <tr> <td>Location type</td> <td>Zonal</td> <td>🔒</td> </tr> <tr> <td>Control plane zone</td> <td>us-central1-c</td> <td>🔒</td> </tr> <tr> <td>Default node zones <small>?</small></td> <td>us-central1-c</td> <td>✍</td> </tr> <tr> <td>Release channel</td> <td>Rapid channel</td> <td>✍ UPGRADE AVAILABLE</td> </tr> <tr> <td>Version</td> <td>1.24.4-gke.800</td> <td></td> </tr> <tr> <td>Total size</td> <td>3</td> <td> ⓘ</td> </tr> <tr> <td>Endpoint</td> <td>34.28.113.186</td> <td>🔒</td> </tr> <tr> <td></td> <td><a href="#">Show cluster certificate</a></td> <td></td> </tr> </tbody> </table>					Name	cd	🔒	Location type	Zonal	🔒	Control plane zone	us-central1-c	🔒	Default node zones <small>?</small>	us-central1-c	✍	Release channel	Rapid channel	✍ UPGRADE AVAILABLE	Version	1.24.4-gke.800		Total size	3	ⓘ	Endpoint	34.28.113.186	🔒		<a href="#">Show cluster certificate</a>	
Name	cd	🔒																													
Location type	Zonal	🔒																													
Control plane zone	us-central1-c	🔒																													
Default node zones <small>?</small>	us-central1-c	✍																													
Release channel	Rapid channel	✍ UPGRADE AVAILABLE																													
Version	1.24.4-gke.800																														
Total size	3	ⓘ																													
Endpoint	34.28.113.186	🔒																													
	<a href="#">Show cluster certificate</a>																														

A popup window should appear. Copy the `Connect to the cluster` command and run it on the host where you have installed `kubectl`.

## Connect to the cluster

You can connect to your cluster via command-line or using a dashboard.

### Command-line access

Configure `kubectl` command line access by running the following command:

```
$ gcloud container clusters get-credentials cd --zone us-central1-c --project ci-cd-3494
```

[RUN IN CLOUD SHELL](#)

Verify `kubectl` configuration using the following commands:

```
kubectl version -o yaml
```

```
clientVersion:
  buildDate: "2025-05-15T20:09:34Z"
  compiler: gc
  gitCommit: 4cb5f0764b8d5f425645c6f433394fede34a8a26
  gitTreeState: clean
  gitVersion: v1.32.4-dispatcher
  goVersion: go1.23.8
  major: "1"
  minor: 32+
  platform: linux/amd64
kustomizeVersion: v5.5.0
serverVersion:
  buildDate: "2025-05-19T04:19:46Z"
  compiler: gc
  gitCommit: d05b573072d7732178c3b9a376809c1d945f669f
  gitTreeState: clean
  gitVersion: v1.32.4-gke.1353003
  goVersion: go1.23.8 X:boringcrypto
  major: "1"
  minor: "32"
  platform: linux/amd64
```

```
kubectl get nodes
```

When you run `kubectl get nodes` you should see three nodes listed with **Ready** status.

Sample output:

NAME	STATUS	ROLES	AGE	VERSION
gke-cd-default-pool-4c6b87d8-39cg	Ready	<none>	26m	v1.32.4-gke.1353003
gke-cd-default-pool-4c6b87d8-9rbn	Ready	<none>	26m	v1.32.4-gke.1353003
gke-cd-default-pool-4c6b87d8-pwg0	Ready	<none>	26m	v1.32.4-gke.1353003

This validates that the cluster is ready and operational.

## Deploy the Frontend Vote App with Kubernetes

Navigate to Google Kubernetes Engine's **Workloads**:

The screenshot shows the Google Cloud Platform dashboard. On the left, there is a sidebar with various services: Cloud overview, Solutions, IAM & Admin, Billing, APIs & Services, Compute Engine, and Kubernetes Engine. The Kubernetes Engine item is highlighted with a red box. In the main pane, under 'All Fleets', it says 'No fleets found'. Below this, there is a 'Workloads' section with a red box around it, showing a table of workloads. The table has columns for NAME, STATUS, ROLES, AGE, and VERSION. It lists three workloads: 'cd' (Ready, <none>, 24m, v1.29.6-gke.1326000), 'Zonal' (Ready, <none>, 24m, v1.29.6-gke.1326000), and 'us-west1-a' (Ready, <none>, 24m, v1.29.6-gke.1326000). There is also a note 'UPGRADE AVAILABLE' next to the 'Regular channel' row. At the bottom of the table, there is a tooltip for 'cd' showing its details: NAME: cd, STATUS: Ready, ROLES: <none>, AGE: 24m, VERSION: v1.29.6-gke.1326000.

Click **Create Deployment**.

The screenshot shows the Google Cloud Kubernetes Engine interface. On the left, there's a sidebar with icons for Clusters, Workloads (which is selected and highlighted with a red box), Services & Ingress, Applications, Secrets & ConfigMaps, Storage, and Object Browser. At the top right, there are Refresh, Delete, Help Assistant, and a notification icon (7). A modal window titled "Deploy a containerized application" is centered over the main content. It contains the text "Kubernetes Engine Deploy a containerized application Deploy, manage, and scale containers on Kubernetes, powered by Google Cloud." Below this is a "Learn more" link. At the bottom of the modal are two buttons: "DEPLOY" (highlighted with a red box) and "SHOW SYSTEM WORKLOADS".

Provide the configurations to deploy the frontend `vote` app as follows and click **Next** to access the container details page.

The screenshot shows the "Create deployment" configuration page. On the left, a sidebar lists steps: 1. Deployment configuration (selected), 2. Container details, 3. Expose (optional). The main area is titled "Deployment configuration". It includes a description of what a deployment is, a "Deployment name" field containing "vote" (highlighted with a red box), and a "Labels" section with a key-value pair "app: vote" (highlighted with a red box). Below these are sections for "Cluster" (set to "cd (us-west1-c)" from "Kubernetes Cluster") and "Create new cluster". At the bottom is a "Next: Container details" button (highlighted with a red box). A success message at the bottom says "The cluster was created successfully".

We will use our existing container image: `DockerHubUserID/vote`. NOTE: Replace the image with your actual image tag.

The screenshot shows the 'Create deployment' wizard in Google Cloud. The 'Container details' step is active. In the 'Image path' field, the URL 'eeganlf/vote:latest' is entered. A red box highlights both the input field and the 'Select' button below it. At the bottom of the screen, the 'Deploy' button is also highlighted with a red box.

Click **Deploy**.

The screenshot shows the deployment progress for the 'vote' application. It indicates the use of the 'us-central1-a/ci-cd' cluster. The current step is 'Creating a Deployment'. The status bar at the top says 'Using an existing cluster: us-central1-a/ci-cd'. At the bottom left, there is a blue link labeled '▲ HIDE ALL STEPS'.

**vote**

To let others access your deployment, expose it to create a service

**OVERVIEW** DETAILS REVISION HISTORY EVENTS LOGS YAML

CPU

Memory

Cluster	<a href="#">cd</a>
Namespace	default
Labels	app: vote
Logs	<a href="#">Container logs</a> , <a href="#">Audit logs</a>
Replicas	3 updated, 3 ready, 3 available, 0 unavailable
Pod specification	Revision 1, containers: <a href="#">vote-1</a>
Horizontal Pod Autoscaler	⚠️ Unable to read all metrics

**Active revisions**

Revision	Name	Status	Summary	Created on	Pods running/Pods total
1	<a href="#">vote-6f9c8b4586</a>	OK	vote-1: okapetanios/vote:latest	Nov 6, 2022, 6:55:33 AM	3/3

**Managed pods**

Revision	Name	Status	Restarts	Created on
1	<a href="#">vote-6f9c8b4586-7hqw7</a>	Running	0	Nov 6, 2022, 6:55:34 AM
1	<a href="#">vote-6f9c8b4586-6rvrr</a>	Running	0	Nov 6, 2022, 6:55:34 AM
1	<a href="#">vote-6f9c8b4586-kj2r2</a>	Running	0	Nov 6, 2022, 6:55:34 AM

Your app is now deployed on the cluster.

## Pod Operations

From the host where `kubectl` is configured, you can explore pod operations by running the following commands:

```
kubectl get pods
```

```
kubectl get pods -o wide
```

**NOTE:** Replace `vote-xxxx` with the actual pod names displayed under `kubectl get pods`.

```
kubectl describe pods vote-xxxx
```

```
kubectl logs vote-xxxx
```

```
kubectl exec vote-xxxx -- ps
```

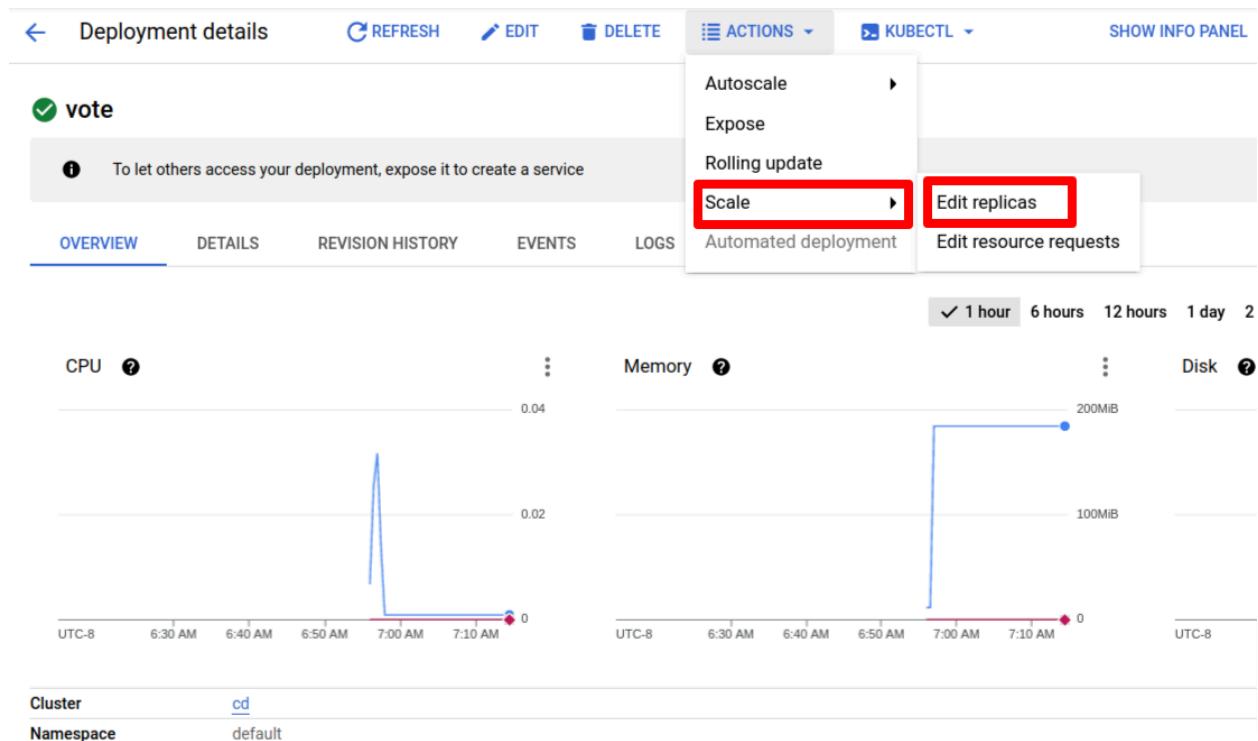
```
kubectl exec -it vote-xxxx -- sh
```

Use **CTRL + d** to exit from the container's shell.

## Scalability

### Scaling Manually

Click the **ACTIONS** menu and select **Scale > Edit replicas**.



Scale a workload to a new size.

Replicas \*

5

\* Indicates required field

CANCEL SCALE

Under the **Managed pods** section, you will now see that 5 pods have been deployed.

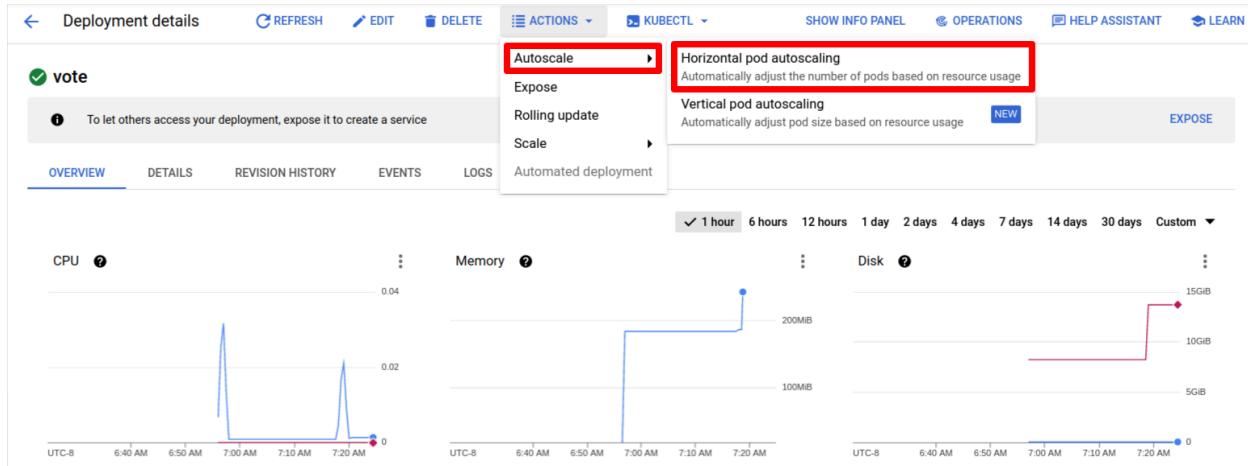
## Managed pods

Revision	Name	Status	Restarts	Created on	↑
1	<a href="#">vote-6f9c8b4586-7hqw7</a>	Running	0	Nov 6, 2022, 6:55:34 AM	
1	<a href="#">vote-6f9c8b4586-6rvvr</a>	Running	0	Nov 6, 2022, 6:55:34 AM	
1	<a href="#">vote-6f9c8b4586-kj2r2</a>	Running	0	Nov 6, 2022, 6:55:34 AM	
1	<a href="#">vote-6f9c8b4586-bklf8</a>	Running	0	Nov 6, 2022, 7:17:48 AM	
1	<a href="#">vote-6f9c8b4586-8trk5</a>	Running	0	Nov 6, 2022, 7:17:48 AM	

You have manually scaled your application using Kubernetes with Google's Kubernetes Engine.

## Autoscaling

You will now configure Kubernetes to handle the scaling for your cluster automatically. Once again, navigate to **ACTIONS**. This time select **Autoscale > Horizontal pod autoscaling** from the dropdown, as follows:



Give your deployment a minimum of 2 replicas and a maximum of 8, as follows:

## Configure Horizontal Pod Autoscaler

Horizontal Pod autoscaling increases and decreases the number of replicated pods to maintain performance and minimize cost. [Horizontal Pod Autoscaling](#)

Minimum number of replicas  Maximum number of replicas \*

### Autoscaling metrics

Use metrics to determine when to autoscale the deployment

CPU (80%)

[ADD METRIC](#)

\* Indicates required field

[CANCEL](#) [DELETE](#) [SAVE](#)

You will now test that your autoscaling works by attempting to delete pods.

## Availability

Try deleting a few pods after listing them and observe what happens. You should see new pods created automatically.

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
vote-799cf486bf-ctvww	1/1	Running	0	4s
vote-799cf486bf-jmr5b	1/1	Running	0	8s
vote-799cf486bf-jnwbs	1/1	Running	0	8s
vote-799cf486bf-p5wzc	1/1	Running	0	8s

*NOTE:* Replace the following `vote-xxxx-xxxx` pod names with your own.

```
$ kubectl delete pods vote-799cf486bf-jnwbs vote-799cf486bf-p5wzc
pod "vote-799cf486bf-jnwbs" deleted
pod "vote-799cf486bf-p5wzc" deleted
```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
vote-799cf486bf-bw4bf	1/1	Running	0	14s
vote-799cf486bf-ctvww	1/1	Running	0	34s
vote-799cf486bf-d7dzq	1/1	Running	0	14s
vote-799cf486bf-jmr5b	1/1	Running	0	38s

Kubernetes has created new pods to replace the ones you deleted. Kubernetes always maintains the desired scale that you provide and constantly monitors and takes action if the desired count changes. This is an example of availability, fault tolerance, and resilience.

## Publishing and Load Balancing with Services

Browse to **Workloads > vote > Actions** and then select **Expose**.

The screenshot shows the Kubernetes UI for a deployment named "vote". The "Actions" menu is open, with the "Expose" option highlighted by a red box. The CPU and Memory usage charts show spikes at approximately 7:00 AM UTC-8.

Time	CPU Usage
6:55 AM	0.01
7:00 AM	0.04
7:05 AM	0.01
7:10 AM	0.01
7:15 AM	0.01
7:20 AM	0.01
7:25 AM	0.01
7:30 AM	0.01

Time	Memory Usage
6:55 AM	100MB
7:00 AM	200MB
7:05 AM	200MB
7:10 AM	200MB
7:15 AM	200MB
7:20 AM	200MB
7:25 AM	200MB
7:30 AM	200MB

Provide values for “Port” and “Target Port” as 80 and then choose **NodePort** from the **Service type** dropdown. This is an important step in order to access this application from outside the cluster.

## Expose

Expose a resource's Pods using a Kubernetes Service.

### Port mapping

Port 1 80	Target port 1 80	Protocol 1 TCP
<a href="#">+ ADD PORT MAPPING</a>		
Service type Node port		

\* Indicates required field

CANCEL [EXPOSE](#)

Once created, browse to **Kubernetes Engine > Gateways, Services & Ingress > Services > vote-xxxx**. Scroll to the **Ports** section and find the NodePort associated with it.

### Serving pods

Name	Status	Endpoints	Restarts	Created on ↑
<a href="#">vote-6f9c8b4586-7hqw7</a>	<span>✓ Running</span>	10.32.0.7	0	Nov 6, 2022, 6:55:34 AM
<a href="#">vote-6f9c8b4586-6rvvr</a>	<span>✓ Running</span>	10.32.1.6	0	Nov 6, 2022, 6:55:34 AM
<a href="#">vote-6f9c8b4586-kj2r2</a>	<span>✓ Running</span>	10.32.2.4	0	Nov 6, 2022, 6:55:34 AM
<a href="#">vote-6f9c8b4586-bklf8</a>	<span>✓ Running</span>	10.32.2.5	0	Nov 6, 2022, 7:17:48 AM
<a href="#">vote-6f9c8b4586-8trk5</a>	<span>✓ Running</span>	10.32.0.8	0	Nov 6, 2022, 7:17:48 AM

### Ports

Port	Node Port	Target Port	Protocol	
80	32245	80	TCP	<a href="#">PORT FORWARDING</a>

You will need to know this later.

## Opening Firewall Rules

You may have already performed the following steps at the time of creating the cluster. If not, open the firewall rules, as follows, in order for you to be able to access the services from outside the cluster (e.g. using a browser on your workstation, which is outside the GKE cluster).

Navigate to the main cloud console menu and **VPC network > Firewall**:

Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs	Hit count
<a href="#">gke-cd-3050e07c-ingubebel</a>	Ingress	gke-cd-3050e07c-ingubebel	IP ranges: 10	tcp:10255	Allow	999	default	Off	-
<a href="#">gke-cd-3050e07c-exkubebel</a>	Ingress	gke-cd-3050e07c-exkubebel	IP ranges: 0..	tcp:10255	Deny	1000	default	Off	-
<a href="#">default-allow-http</a>	Ingress	http-server	IP ranges: 0..	all	Allow	1000	default	Off	-
<a href="#">default-allow-https</a>	Ingress	https-server	IP ranges: 0..	tcp:443	Allow	1000	default	Off	-
<a href="#">gke-cd-3050e07c-all</a>	Ingress	gke-cd-3050e07c-all	IP ranges: 10	tcp udp sctp;icmp;esp;ah	Allow	1000	default	Off	-
<a href="#">gke-cd-3050e07c-vms</a>	Ingress	gke-cd-3050e07c-vms	IP ranges: 10	tcp:1-65535 udp:1-65535 icmp	Allow	1000	default	Off	-

Scroll to the rule that ends in **-all** and click on it:

<input type="checkbox"/> Filter Enter property name or value										
	Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs	Hit count
<input type="checkbox"/>	<a href="#">gke-cd-3050e07c-ingubebel</a>	Ingress	gke-cd-3050e07c-ingubebel	IP ranges: 10	tcp:10255	Allow	999	default	Off	-
<input type="checkbox"/>	<a href="#">gke-cd-3050e07c-exkubebel</a>	Ingress	gke-cd-3050e07c-exkubebel	IP ranges: 0..	tcp:10255	Deny	1000	default	Off	-
<input type="checkbox"/>	<a href="#">default-allow-http</a>	Ingress	http-server	IP ranges: 0..	all	Allow	1000	default	Off	-
<input type="checkbox"/>	<a href="#">default-allow-https</a>	Ingress	https-server	IP ranges: 0..	tcp:443	Allow	1000	default	Off	-
<input checked="" type="checkbox"/>	<a href="#">gke-cd-3050e07c-all</a>	Ingress	gke-cd-3050e07c-all	IP ranges: 10	tcp udp sctp;icmp;esp;ah	Allow	1000	default	Off	-
<input type="checkbox"/>	<a href="#">gke-cd-3050e07c-vms</a>	Ingress	gke-cd-3050e07c-vms	IP ranges: 10	tcp:1-65535 udp:1-65535 icmp	Allow	1000	default	Off	-

Click **EDIT**:

[Firewall rule details](#)  [EDIT](#)  [DELETE](#)

---

gke-cd-3050e07c-all

**Logs** Off  
[view in Logs Explorer](#)

**Network**  
default

**Priority**  
1000

**Direction**  
Ingress

**Action on match**  
Allow

Scroll to **Source IPv4 ranges** and enter `0.0.0.0/0`:

Direction  
Ingress

Action on match  
Allow

Targets  
Specified target tags

Target tags \*  
gke-cd-3050e07c-node ×

Source filter  
IPv4 ranges

Source IPv4 ranges \*  
10.32.0.0/14 × 0.0.0.0/0 × for example, 0.0.0.0/0, 192.168.2.0/24 ?

Second source filter  
None

Click **Save** at the bottom of the screen.

Find the external IP address of the nodes participating in the Kubernetes cluster.

<input type="checkbox"/>	<input checked="" type="checkbox"/>	gke-cd-default-pool-8fd7c0bf-g5kg	us-central1-c	gke-cd-default-pool-8fd7c0bf-g5kg	10.128.0.22 (nic0)	35.238.107.163 (nic0)	SSH <span style="color: red;">⋮</span>
<input type="checkbox"/>	<input checked="" type="checkbox"/>	gke-cd-default-pool-8fd7c0bf-q43h	us-central1-c	gke-cd-default-pool-8fd7c0bf-q43h	10.128.0.21 (nic0)	34.71.167.239 (nic0)	SSH <span style="color: red;">⋮</span>
<input type="checkbox"/>	<input checked="" type="checkbox"/>	gke-cd-default-pool-8fd7c0bf-r8hn	us-central1-c	gke-cd-default-pool-8fd7c0bf-r8hn	10.128.0.20 (nic0)	104.154.246.28 (nic0)	SSH <span style="color: red;">⋮</span>

Alternately, you can find the external IPs of the nodes by using the following command:

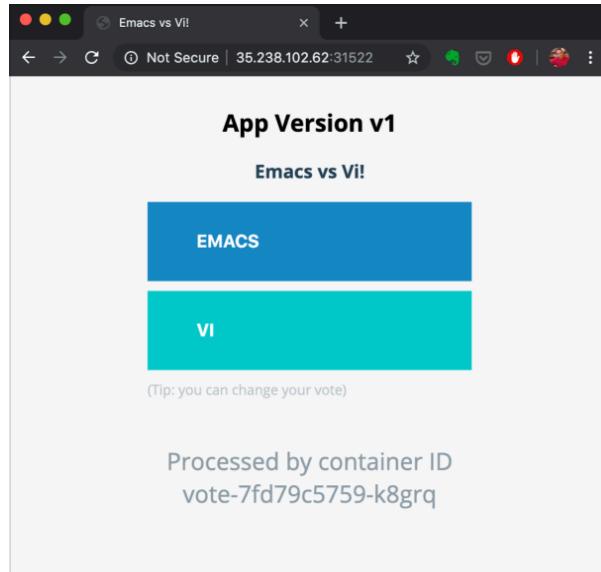
```
kubectl get nodes -o wide
```

Sample output:

NAME	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	STATUS	ROLES	AGE	VERSION
KERNEL-VERSION	CONTAINER-RUNTIME						
gke-cd-default-pool-4c6b87d8-39cg	v1.32.4-gke.1353003	10.138.0.38	Container-Optimized OS from Google	Ready	<none>	19h	
	6.6.72+						containerd://1.7.24
gke-cd-default-pool-4c6b87d8-9rbn	v1.32.4-gke.1353003	10.138.0.37	Container-Optimized OS	Ready	<none>	19h	
							34.168.221.251

```
from Google  6.6.72+           containerd://1.7.24
gke-cd-default-pool-4c6b87d8-pwg0  Ready     <none>   19h
v1.32.4-gke.1353003  10.138.0.39  35.185.198.94  Container-Optimized OS
from Google  6.6.72+           containerd://1.7.24
```

Use any of the external IPs and load the application by using NodePort as `http://IPADDRESS:NODEPORT`. For example:



Try reloading the URL a few times and observe the container ID. Do you see the container IDs changing? If yes, the load balancing is in place.

Try clicking an option. You will notice that the app crashes. This is because the `redis` backend is not set up. This is what you will do in the next section.

## Service Discovery

To launch the `redis` backend microservice go to **Kubernetes Engine > Workloads** and select the **Deploy** option.

The screenshot shows the Kubernetes Workloads interface. At the top, there are buttons for REFRESH, DEPLOY (which is highlighted with a red box), and DELETE. Below these are dropdown menus for Cluster and Namespace, and buttons for RESET and SAVE. A descriptive text block states: "Workloads are deployable units of computing that can be created and managed in a cluster." Below this, there are two tabs: OVERVIEW (which is selected) and COST OPTIMIZATION. A filter bar shows "Is system object : False" and a "Filter workloads" input field. A table lists a single workload named "vote" with status "OK", type "Deployment", 5/5 pods, namespace "default", and cluster "cd".

Enter the **Application name** and **Value 1** as `redis`.

## ② Configuration

A deployment is a configuration which defines how Kubernetes deploys, manages, and scales your container image. Kubernetes will ensure your system matches this configuration.

The form for creating a deployment. It has two main sections: "Application name \*" containing "redis" and "Namespace \*" containing "default". Both sections are highlighted with a red box.

## Labels

Use Kubernetes labels to control how workloads are scheduled to your nodes. Labels are applied to all nodes in this node pool and cannot be changed once the cluster is created.

The form for adding labels. It has two fields: "Key 1 \*" containing "app" and "Value 1" containing "redis". The "Value 1" field is highlighted with a red box.

Provide the image `redis:alpine` and click on **Deploy**.

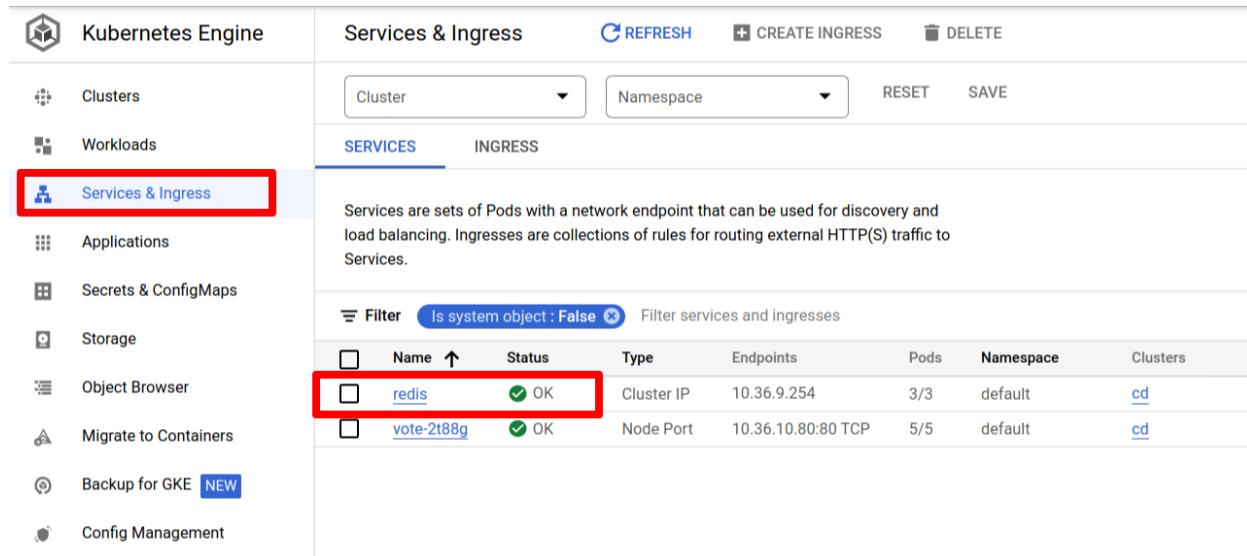
The screenshot shows the Google Cloud Platform interface for creating a Kubernetes deployment. The top navigation bar includes 'Google Cloud', 'Search', and various icons. The main page title is 'Kubernetes Engine / Create deployment'. On the left, a sidebar lists steps: 'Deployment configuration' (selected), 'Container details' (current step), and 'Expose (optional)'. The 'Container details' section shows a 'New container' configuration with 'Existing container image' selected and 'redis:alpine' entered in the 'Image path' field. Below it, an 'Initial command' field contains 'redis-server'. At the bottom of the form are 'Back', 'Next: Expose (optional)', 'Deploy', 'Cancel', and 'View YAML' buttons.

Wait for the `redis` deployment to be ready.

Enter the following command into your terminal to expose the `redis` application as a service:

```
$ kubectl expose deployment redis --port=6379 --name=redis
service/redis exposed
```

Navigate to **Kubernetes Engine > Services and Ingress**. You will notice the `redis` service is listed:



The screenshot shows the Kubernetes Engine interface with the 'Services & Ingress' tab selected. On the left, there's a sidebar with various options like Clusters, Workloads, Applications, etc., with 'Services & Ingress' highlighted. The main area displays a table of services. The first row of the table is highlighted with a red box, showing the service named 'redis' with status 'OK'. The table columns include Name, Status, Type, Endpoints, Pods, Namespace, and Clusters.

	Name	Status	Type	Endpoints	Pods	Namespace	Clusters
<input type="checkbox"/>	redis	OK	Cluster IP	10.36.9.254	3/3	default	<a href="#">cd</a>
<input type="checkbox"/>	<a href="#">vote-2t88g</a>	OK	Node Port	10.36.10.80:80 TCP	5/5	default	<a href="#">cd</a>

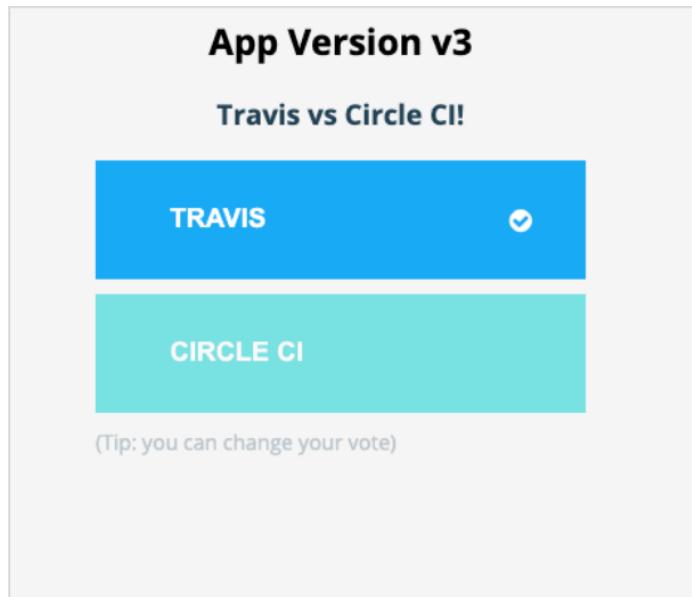
This means you can submit a vote on the vote application and it won't crash.

Run:

```
$ kubectl get node -o wide
```

The output will show your nodes. Under the `EXTERNAL-IP` column you will find the public IP's associated with each of your nodes. You can navigate to any of these IP's with the frontend port, which is found at **Kubernetes Engine > Services & Ingress > vote-xxxx > Ports > Node Port**. Enter the following into your web browser to see the frontend and click to vote:

`NODEIPADDRESS : NODEPORT`



As soon as you submit the vote, you should see a tick, which validates the communication between the frontend vote and the backend redis applications.

## Summary

In this lab, we set up an account on Google Cloud. We installed the `kubectl` command line utility and set it up to interact with our cluster on Google Cloud. We then deployed the voting application to a Kubernetes Cluster on Google Cloud so that we can visit the app from any browser and submit a vote.



## Lab 11. Continuous Deployment with Argo CD

By the end of this lab exercise, you should be able to:

- Implement ArgoCD on a cluster in the cloud
- Set up an automated deployment pipeline with Jenkins and Argo CD

### Connect to the Cluster

In order to install ArgoCD onto the cluster, we need to connect to our cluster with the **CLOUD SHELL** option.

#### Connect to the cluster

You can connect to your cluster via command-line or using a dashboard.

##### Command-line access

Configure [kubectl](#) command line access by running the following command:

```
$ gcloud container clusters get-credentials cd --zone us-west1-a --project ci-ci-349400
```

[RUN IN CLOUD SHELL](#)

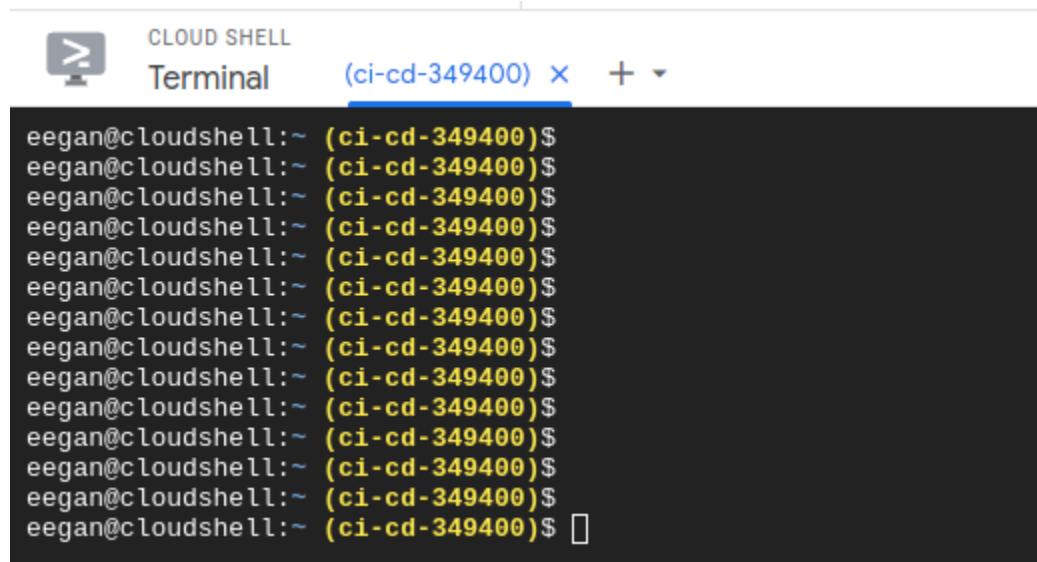
##### Cloud Console dashboard

You can view the workloads running in your cluster in the Cloud Console [Workloads dashboard](#).

[OPEN WORKLOADS DASHBOARD](#)

OK

This will open a terminal at the bottom of your browser:



```
eegan@cloudshell:~ (ci-cd-349400)$  
eegan@cloudshell:~ (ci-cd-349400)$
```

You will type the commands to deploy ArgoCD into this terminal.

## Deploy ArgoCD to the Cluster

First, create a namespace for `argocd` to separate it from the rest of the services running in the cluster:

```
$ kubectl create namespace argocd  
namespace/argocd created
```

```
kubectl get namespace  
NAME      STATUS   AGE  
argocd    Active   7s  
default   Active   13d  
kube-node-lease  Active   13d  
kube-public   Active   13d  
kube-system   Active   13d
```

Then apply the installation yaml file from Argo Project's GitHub repository:

```
$ kubectl apply -n argocd -f  
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml  
  
$ kubectl get svc -n argocd
```

Note that the ports that **argocd-server** is using are 443 and 80. We need to expose these ports on a service. Under **Workloads > argocd-server** click **Actions** and choose **Expose**.

ArgoCD is served on port 8080, so enter 8080 into **Port 1** and select **Node port** from the **Service type** dropdown menu.

## Expose

Expose a resource's Pods using a Kubernetes Service.

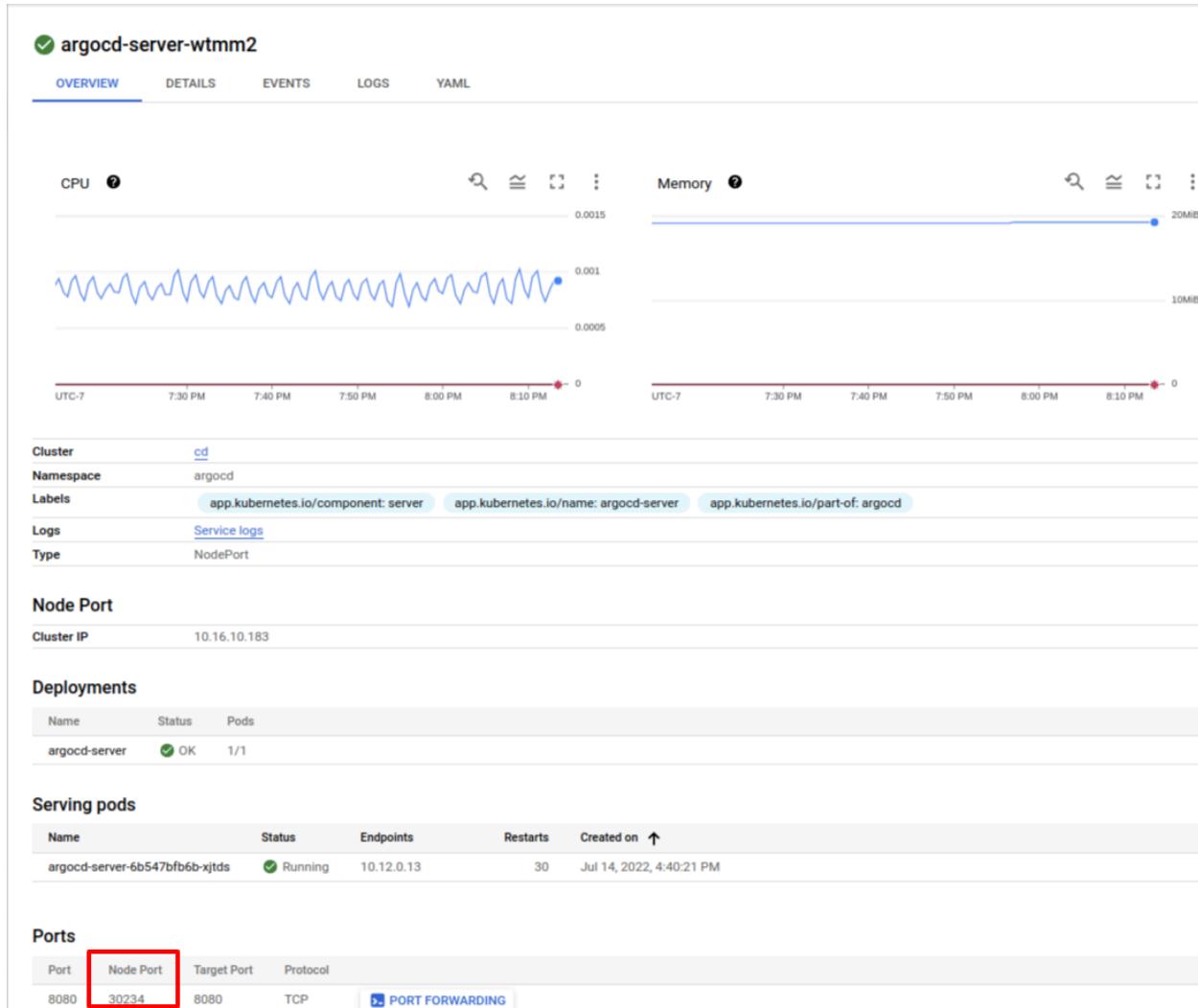
### Port mapping

Port 1 8080	Target port 1	Protocol 1 TCP
<b>+ ADD PORT MAPPING</b>		
Service type Node port		
<small>* Indicates required field</small>		
CANCEL		<b>EXPOSE</b>

Finally, click **EXPOSE**.

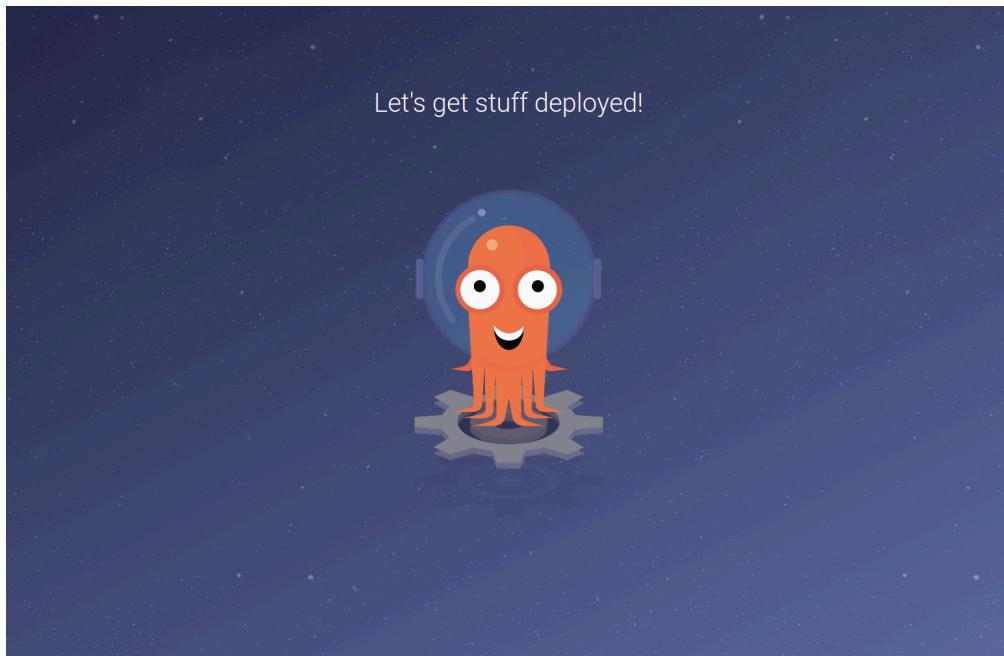
Exposing a workload on a Node Port automatically generates a service. The service will be named **argo-cd-server-xxxx**. **xxxx** is a randomly generated ID that will be different for everyone.

To get Argo CD's Node Port, navigate to **Kubernetes Engine > Gateways, Services and Ingress > argo-cd-server-xxxxx**.



Visit one of the nodes' IP address with the Node Port like so: **NODEIPADDRESS:NODEPORT**

You will get a warning. Click **Advanced** and click to proceed to the URL. You should now see ArgoCD's home page.



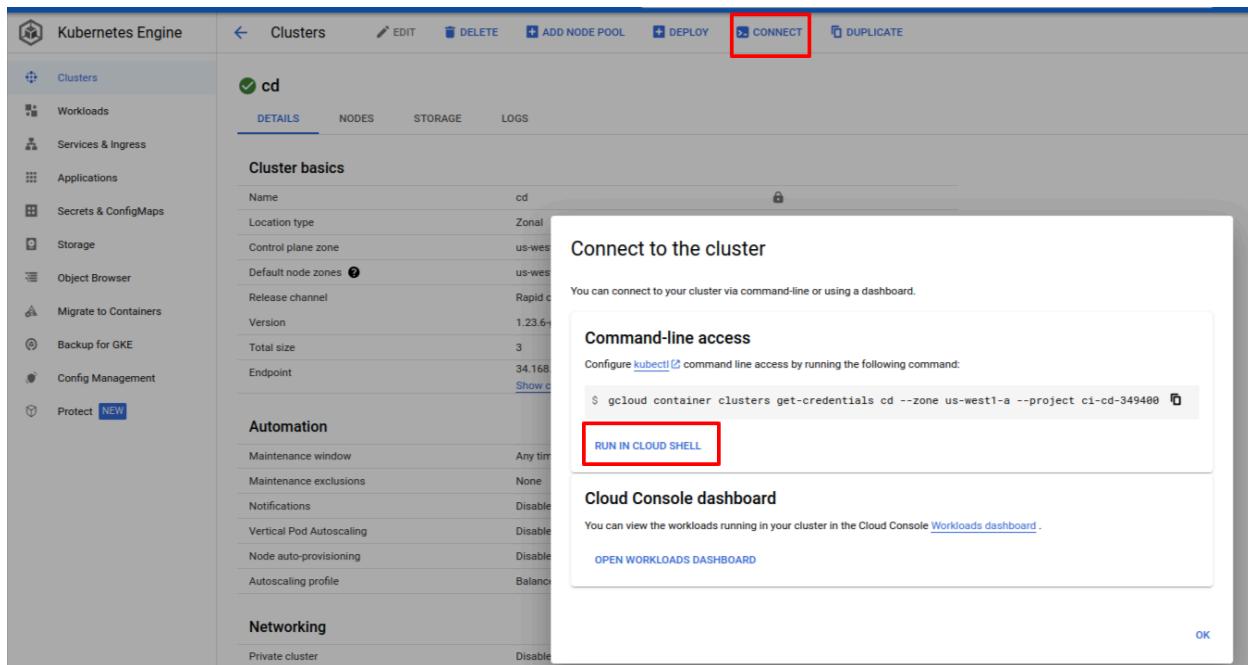
argo

The username is **admin**. We need to get the password that was automatically generated during install. To get the password, connect to your cluster:

A screenshot of the Kubernetes Engine UI. The left sidebar has a "Clusters" item highlighted with a red box. The main area shows a table of clusters. The first column is "Status" (checkbox), the second is "Name" (checkbox with a green checkmark and the value "cd"), the third is "Location" (us-west1-a), and the fourth is "Number of nodes" (3). The table header includes "OVERVIEW", "MONITORING", "PREVIEW" (which is active and highlighted in dark blue), and "COST OPTIMIZATION".

Status	Name	Location	Number of nodes
<input type="checkbox"/>	<input checked="" type="checkbox"/> cd	us-west1-a	3

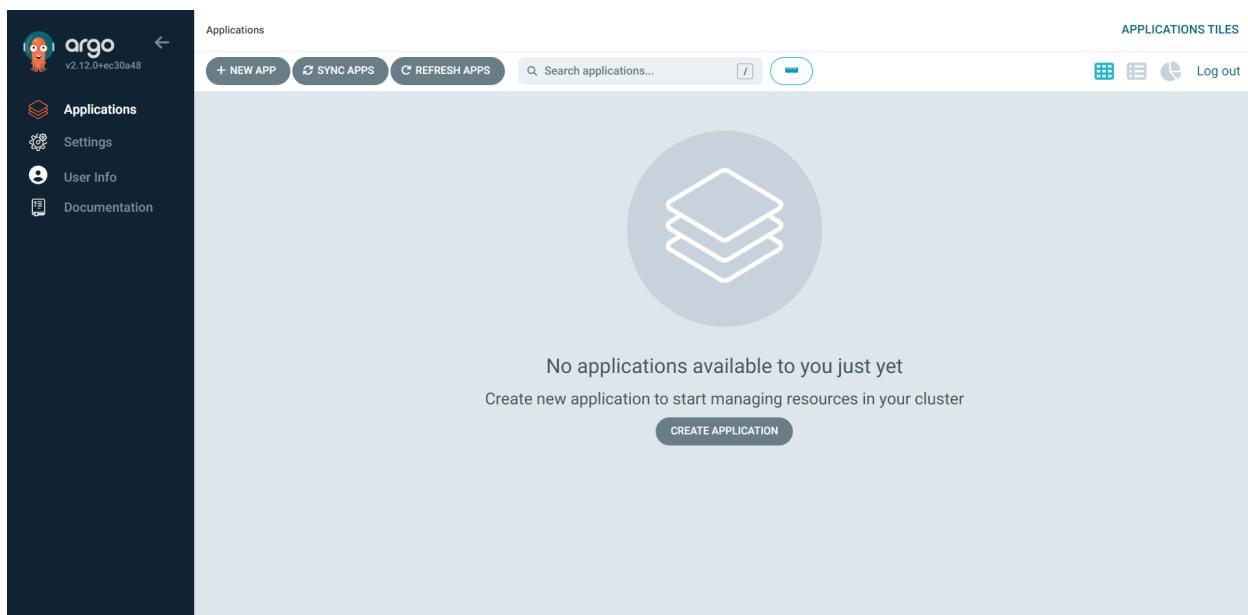
Run the cloud shell:



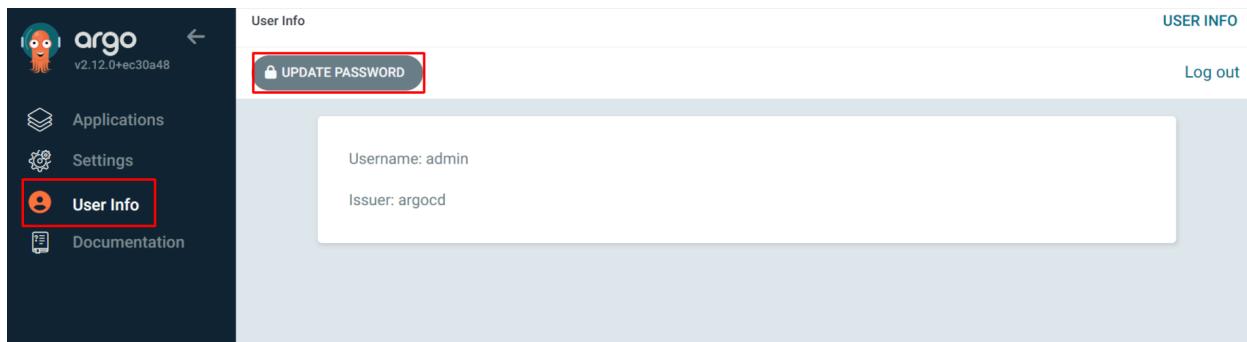
Run the following command inside the terminal:

```
$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath=".data.password" | base64 -d; echo
```

The output is the password for ArgoCD. Enter **admin** as your username and the output of the above command as the password.



In order to change the password, click the user icon and then password:



The password must be at least eight characters long.

## Setting Up Our Git Repository

Create a new repository on GitHub named `vote-deploy`. This could be named anything you want.

Add a file in the root directory of the repository called `vote-ui-deployment.yaml` and paste the following into it:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote-ui
spec:
  replicas: 1
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: vote-ui
  template:
    metadata:
      labels:
        app: vote-ui
    spec:
      containers:
        - image: docker.io/xxxx/vote
          name: vote-ui
          ports:
            - containerPort: 80
```

---

The code can be found here:

<https://raw.githubusercontent.com/lftraining/LFS261-example-voting-app/refs/heads/master/gists/vote-ui-deployment.yaml>

Be sure to replace **xxxx** with your own Docker Hub ID.

Add another file in the root directory of the repository called **vote-ui-svc.yaml** and paste the following into it:

```
apiVersion: v1
kind: Service
metadata:
  name: vote-ui
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: vote-ui
```

The code can be found here:

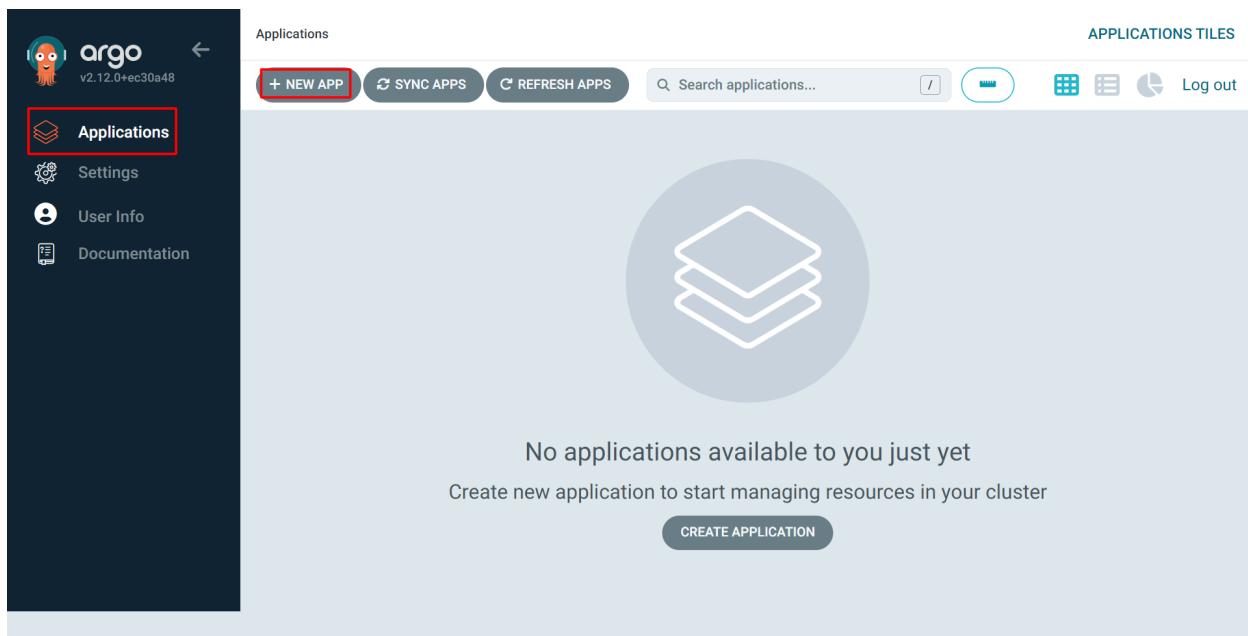
<https://raw.githubusercontent.com/lftraining/LFS261-example-voting-app/refs/heads/master/gists/vote-ui-svc.yaml>

If you did this in a local repository, make sure it is pushed to GitHub so that Argo CD can access the repository.

## Deploying Our Vote Front End

Now we will set up our vote application in Argo CD.

First, click **+ NEW APP**:



Enter **vote-ui** as the **Application Name**. Make sure **Project Name** is set to **default**. Check **AUTO-CREATE NAMESPACE**. Paste the URL to the **vote-deploy** GitHub repository. Make sure **HEAD** is chosen as the **Revision**. **Path** needs to be a period like so: `'.'`. This tells Argo CD that our yaml files are inside the root directory of our repository. Choose <https://kubernetes.default.svc> from the **Cluster URL** drop down and enter **vote-app** as the **Namespace**. Match the following:

Finally, click **CREATE**.

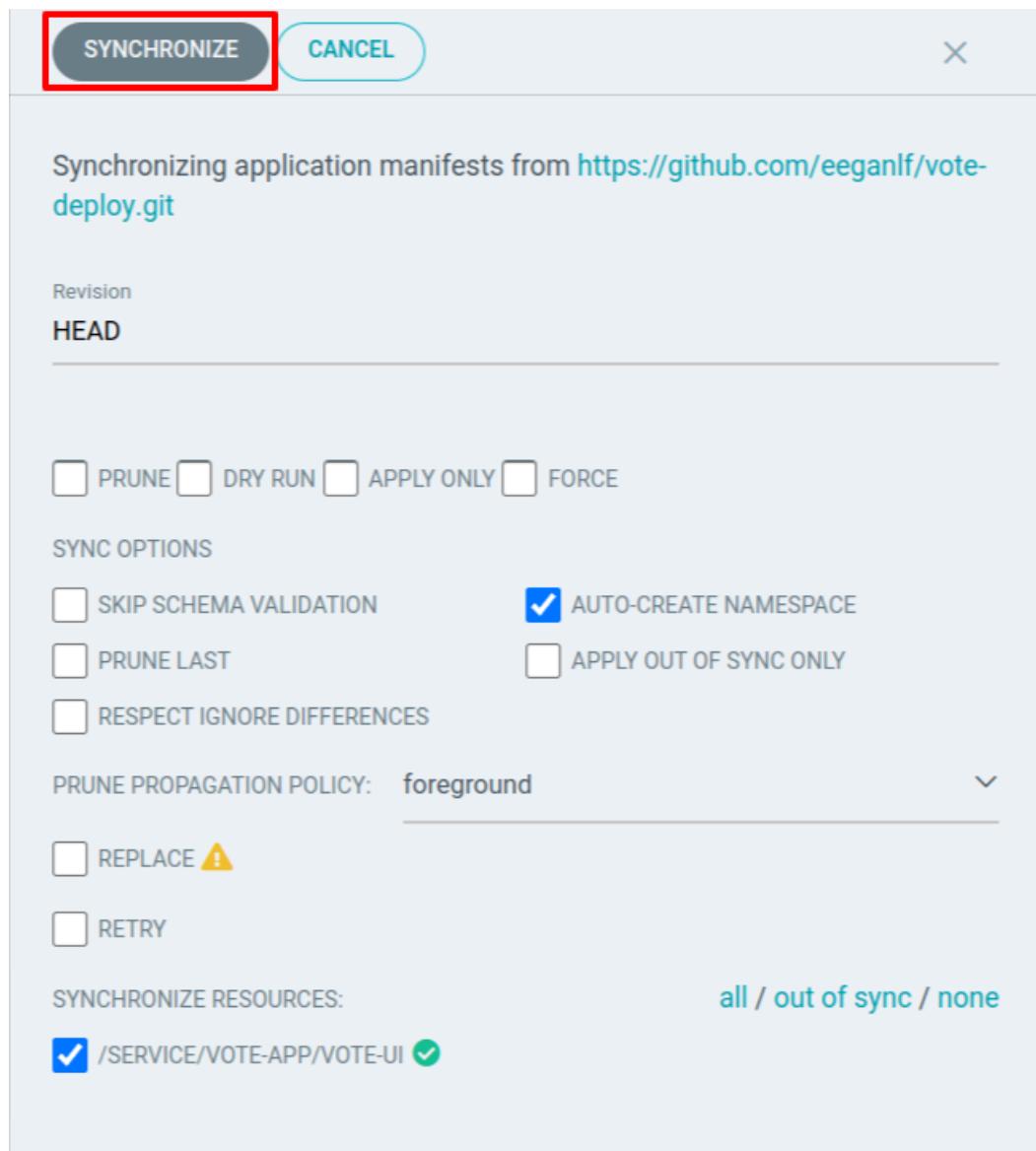
The application should show up automatically in the Argo UI:

A screenshot of the Argo UI interface. At the top, there's a header with a logo and the text "vote-ui". To the right is a star icon. Below the header, there's a table-like structure with the following data:

Project:	default
Labels:	
Status:	Missing OutOfSync
Repository:	<a href="https://github.com/eeganlf/vote-deploy.git">https://github.com/eeganlf/vote-deploy.git</a>
Target Revi...	HEAD
Path:	.
Destination:	in-cluster
Namespace:	vote-app
Created At:	08/09/2024 11:55:19 (a few seconds ago)

At the bottom, there are three buttons: a red-bordered "SYNC" button, a "C" button, and an "X" button.

Click **SYNCHRONIZE**. A fly-out will appear:



Click on the card:

**vote-ui**

Project: default

Labels:

Status: Healthy Synced

Repository: <https://github.com/eeganlf/vote-deploy.git>

Target Revi...: HEAD

Path: .

Destination: in-cluster

Namespace: vote-app

Created At: 08/09/2024 11:55:19 (11 minutes ago)

Last Sync: 08/09/2024 12:05:02 (a minute ago)

SYNC C X

We can see that Argo CD has deployed our **vote-ui** application:

**Applications / Q vote-ui**

**APPLICATION DETAILS TREE**

**vote-ui**

**APP HEALTH**: Healthy

**SYNC STATUS**: Synced to HEAD (87cbfc0)

**LAST SYNC**: Sync OK to 87cbfc0

**NAME**

**KINDS**

**SYNC STATUS**

- Synced: 2
- OutOfSync: 0

**HEALTH STATUS**

- Healthy: 4
- Progressing: 0
- Degraded: 0
- Suspended: 0
- Missing: 0
- Unknown: 0

To visit your **vote-ui**, get its Node Port by navigating to **Google Cloud > Kubernetes Engine > Gateways, Services & Ingress > vote-ui**:

The screenshot shows the Google Cloud Platform interface with the 'Compute Engine' menu selected. In the main pane, the 'Gateways, Services & Ingress' section is highlighted with a red box. Below it, a table lists various services and their details.

ID	EXTERNAL-IP	PORT(S)	AGE
9.106	<none>	7000/TCP, 8080/TCP	2m35s
6.212	<none>	5556/TCP, 5557/TCP, 5558/TCP	2m35s
7.185	<none>	8082/TCP	2m35s
5.171	<none>	9001/TCP	2m35s
8.231	<none>	6379/TCP	2m35s
9.227	<none>	8081/TCP, 8084/TCP	2m35s
6.205	<none>	80/TCP, 443/TCP	2m35s
5.9	<none>	8083/TCP	2m35s

The screenshot shows the Google Cloud Platform interface with the 'Kubernetes Engine' menu selected. The 'Services' tab is highlighted with a red box. Below it, a table lists various services and their details.

Name	Status	Type	Endpoints	Pods	Namespace	Clusters
argocd-applicationset-controller	OK	Cluster IP	34.118.229.106	1/1	argocd	cd
argocd-dex-server	OK	Cluster IP	34.118.226.212	1/1	argocd	cd
argocd-metrics	OK	Cluster IP	34.118.237.185	1/1	argocd	cd
argocd-notifications-controller-metrics	OK	Cluster IP	34.118.235.171	1/1	argocd	cd
argocd-redis	OK	Cluster IP	34.118.238.231	1/1	argocd	cd
argocd-repo-server	OK	Cluster IP	34.118.239.227	1/1	argocd	cd
argocd-server	OK	Cluster IP	34.118.226.205	1/1	argocd	cd
argocd-server-26t5w	OK	Node Port	34.118.229.34:8080 TCP	1/1	argocd	cd
argocd-server-metrics	OK	Cluster IP	34.118.235.9	1/1	argocd	cd
vote-5c7g4	OK	Node Port	34.118.231.23:80 TCP	5/5	default	cd
vote-ui	OK	Node Port	34.118.226.88:80 TCP	1/1	vote-app	cd

Scroll to the bottom and take note of the NodePort. Yours will likely be different than anyone else's:

The screenshot shows the 'Service details' page for a Kubernetes service. The left sidebar lists various options: Clusters, Workloads, Services & Ingress (which is selected), Applications, Secrets & ConfigMaps, Storage, Object Browser, Migrate to Containers, Backup for GKE, Config Management, and Protect (with a NEW button). The main content area has tabs for 'Logs' (selected) and 'Metrics'. The 'Logs' tab shows a message: 'No data is available for the selected time frame.' Below this are log entries with fields for Cluster (cd), Namespace (vote-app), Labels (app.kubernetes.io/instance: vote), Logs (Service logs), and Type (NodePort). The 'Metrics' tab shows a graph with a single data point at 10.16.11.83. The 'Node Port' section shows the Cluster IP as 10.16.11.83. The 'Serving pods' section indicates 'No matching pods'. The 'Ports' section shows a table with one row for 'http' port 80, where the 'Node Port' value 31323 is highlighted with a red box.

Name	Status	Endpoints	Restarts
No matching pods			

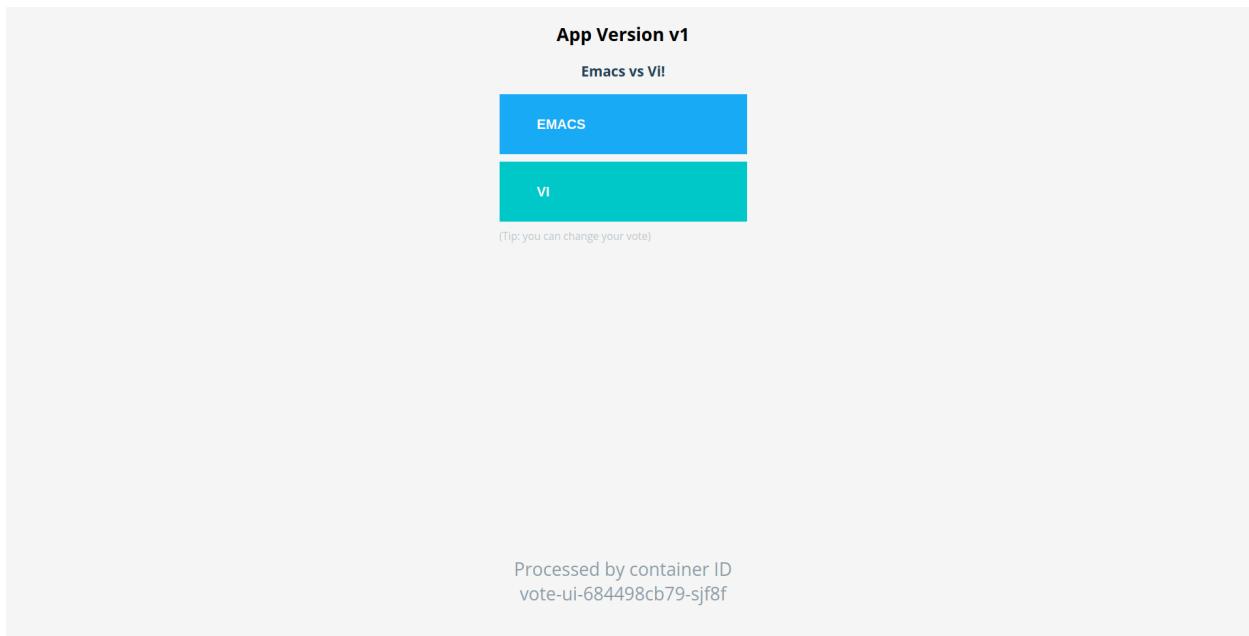
Name	Port	Node Port	Target Port	Protocol
http	80	31323	80	TCP

Obtain any one of your node's IP addresses by going to **Google Cloud > Compute Engine > VM instances:**

Status	Type	Endpoints	Pods	Namespace	Clusters
OK	Cluster IP	34.118.229.106	1/1	argood	cd
OK	Cluster IP	34.118.226.212	1/1	argood	cd
OK	Cluster IP	34.118.237.185	1/1	argood	cd
OK	Cluster IP	34.118.235.171	1/1	argood	cd
OK	Cluster IP	34.118.238.231	1/1	argood	cd
OK	Cluster IP	34.118.239.227	1/1	argood	cd
OK	Cluster IP	34.118.226.205	1/1	argood	cd
OK	Node Port	34.118.229.34:8080 TCP	1/1	argood	cd
OK	Cluster IP	34.118.235.9	1/1	argood	cd
OK	Node Port	34.118.231.23:80 TCP	5/5	default	cd
OK	Node Port	34.118.226.88:80 TCP	1/1	vote-app	cd

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
OK	ci-01	us-west1-b			10.138.0.2 (nic0)		SSH
OK	ci-2	us-west1-b			10.138.0.10 (nic0)	34.168.51.111 (nic0)	SSH
OK	gke-cd-default-pool-3deb5c5d-5nbt	us-west1-a	gke-cd-default-pool-3deb5c5d-5nbt		10.138.0.11 (nic0)	34.168.51.111 (nic0)	SSH
OK	gke-cd-default-pool-3deb5c5d-914d	us-west1-a	gke-cd-default-pool-3deb5c5d-914d		10.138.0.12 (nic0)	35.168.51.111 (nic0)	SSH
OK	gke-cd-default-pool-3deb5c5d-hn2s	us-west1-a	gke-cd-default-pool-3deb5c5d-hn2s		10.138.0.13 (nic0)	34.168.51.111 (nic0)	SSH
OK	minnaker	us-west4-b			10.182.0.2 (nic0)		SSH

Visit one of your node IPs with the NodePort of your `vote-ui` service and you should see the vote application UI:



## Set Argo CD to Continuous Deployment

In the previous section, you had to manually instruct Argo CD to synchronize the cluster with changes made in the `vote-deploy` repository. To implement Continuous Deployment, you must configure the Argo CD application to auto-sync. To do this, the next section will guide you through the steps to set up a Jenkins deployment pipeline and then configure Argo CD to automatically deploy our application.

## Creating the **deployment** Pipeline

First, you will need the description of your GitHub credentials. Refer to the following and take note of the description in the **Description** box:

The screenshot shows the Jenkins 'Update credentials' page for a GitHub credential named 'eeganlf/\*\*\*\*\* (eeganlf github)'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc.)'. The 'Username' is 'eeganlf', and the 'ID' is 'eeganlf-github'. The 'Description' is 'eeganlf github'. The 'Password' field is concealed. A red box highlights the 'Save' button at the bottom.

Add a **Jenkinsfile** to your **vote-deploy** Git repository that you created earlier.

Add the following code and be sure to replace the red text with your own information and please

**NOTE:** `JenkinsGithubCredsDescription` needs to be replaced with the **DESCRIPTION** found in the Description text box of your GitHub credential that you just took note of. It is NOT the text found in the ID text box:

```
node {
    def app

    stage('Clone repository') {
        checkout scm
    }

    stage('Update GIT') {
        script {
            catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {
                withCredentials([usernamePassword(credentialsId: 'JenkinsGithubCredsDescription', passwordVariable: 'GIT_PASSWORD', usernameVariable: 'GIT_USERNAME')]) {
                    sh "git config user.email yourGitHubEmail@email.com"
                    sh "git config user.name yourGitHubUsername"
                    sh "cat vote-ui-deployment.yaml"
                }
            }
        }
    }
}
```

```
        sh "sed -i
's+${dockerHubUserName}/vote.*+${dockerHubUsername}/vote:${DOCKERTAG}+g'
vote-ui-deployment.yaml"
        sh "cat vote-ui-deployment.yaml"
        sh "git add ."
        sh "git commit -m 'Done by Jenkins Job
deployment: ${env.BUILD_NUMBER}'"
        sh "git push
https://${GIT_USERNAME}:${GIT_PASSWORD}@github.com/${GIT_USERNAME}/vot
e-deploy.git HEAD:master"
    }
}
}
}
```

The code can be found here:

<https://raw.githubusercontent.com/lftraining/LFS261-example-voting-app/refs/heads/master/gists/LFS261ArgoDeployJenkinsfile>

If you did this in a local repository, make sure it is pushed to GitHub so that Jenkins can access the file.

Your `vote-deploy` repository should now contain three files:

```
Jenkinsfile  
vote-ui-deployment.yaml  
vote-ui-svc.yaml
```

In Jenkins, you will now create a **deployment** pipeline that will update the **vote-deploy** repository.

Create the **deployment** pipeline by clicking **New Item** on the Jenkins homepage. The job must be called **deployment**. Then choose **Pipeline**:

The screenshot shows the Jenkins interface for creating a new job. The title bar says "Enter an item name". A red box highlights the input field containing "deployment". Below it, a message says "» A job already exists with the name 'deployment'". The "Pipeline" option is also highlighted with a red box.

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

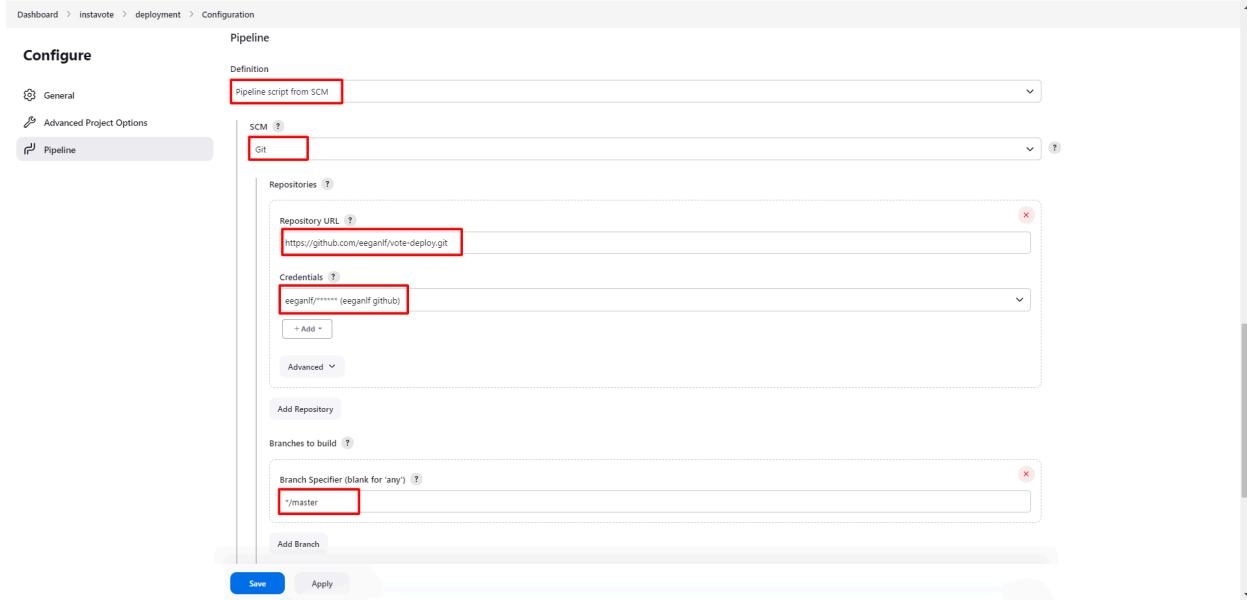
**Multibranch Pipeline**  
Creates a set of Pipeline projects according to detected branches in one SCM repository.

You must mark this pipeline as *parameterized* and tell the job which variable is being passed to it. Under the **General** tab, select **This project is parameterized**. Click the **Add Parameter** dropdown menu and select **String Parameter**. Give the parameter the name **DOCKERTAG** and the default as **latest** as follows:

The screenshot shows the Jenkins General configuration page. The "Configure" section is open, and the "General" tab is selected. Under "Advanced Project Options", the "Pipeline" section is expanded. A "String Parameter" is added with the following settings:

- Name: DOCKERTAG (highlighted by a red box)
- Default Value: latest (highlighted by a red box)
- Description: (empty)
- Plain text: Preview
- Trim the string: (unchecked)

Under the **Pipeline** section, fill out as follows, using your forked repository of the deployment repository:



Click **Save**.

## Triggering Deployment Pipeline

To achieve continuous deployment we need Jenkins to automatically update our deployment repository. Argo CD is watching this repository for changes and will automatically deploy our application to Kubernetes when it notices a change.

To do this, we need to add the following stage as the *final stage* in the `stages` section to our [example-voting-app/Jenkinsfile](#):

```
stage('Trigger deployment') {
    agent any
    environment{
        def GIT_COMMIT = "${env.GIT_COMMIT}"
    }
    steps{
        echo "${GIT_COMMIT}"
        echo "triggering deployment"
        // passing variables to job deployment run by vote-deploy
        repository Jenkinsfile
        build job: 'deployment', parameters: [string(name: 'DOCKERTAG',
value: GIT_COMMIT)]
    }
}
```

Find the code at the following link:

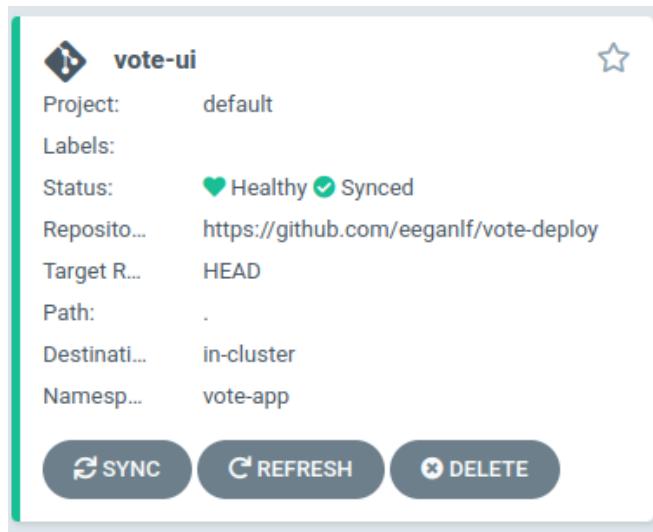
<https://raw.githubusercontent.com/lftraining/LFS261-example-voting-app/refs/heads/master/gists/Continuous-Deploy-Stage-Jenkinsfile>

Recall that our CI pipeline is already configured to look for this Jenkinsfile.

Commit all changes.

## Setting Argo CD to Continuously Deploy

In Argo CD, click on the vote-ui card:



Then click on **DETAILS**:

Applications / Q vote-ui

**DETAILS** DIFF SYNC SYNC STATUS HISTORY AND ROLLBACK DELETE REFRESH

APPLICATION DETAILS TREE

APP HEALTH: Healthy

SYNC STATUS: Synced to HEAD (87cbfc0)

LAST SYNC: Sync OK to 87cbfc0

Auto sync is not enabled.

Author: eeganlf <101604439+eeganlf@users.noreply.github.com>

Comment: Update vote-ui-deployment.yaml removed commit ha...

Sync OK to 87cbfc0

Succeeded 7 minutes ago (Fri Aug 09 2024 12:05:02 GMT-0700)

Author: eeganlf <101604439+eeganlf@users.noreply.github.com>

Comment: Update vote-ui-deployment.yaml removed commit ha...

```

graph LR
    voteUi[vote-ui] --> svc[vote-ui svc]
    svc --> ep[vote-ui ep]
    svc --> es[vote-ui endpointslice]
    svc --> deploy[vote-ui deploy]
    deploy --> rs[vote-ui-5ff48ccb5d rs]
    rs --> pod[pod vote-ui-5ff48ccb5d-x2mjj]
  
```

vote-ui (17 minutes)

vote-ui svc (9 minutes)

vote-ui ep (9 minutes)

vote-ui endpointslice (9 minutes)

vote-ui deploy (7 minutes rev:1)

vote-ui-5ff48ccb5d rs (7 minutes rev:1)

vote-ui-5ff48ccb5d-x2mjj pod (7 minutes running 1/1)

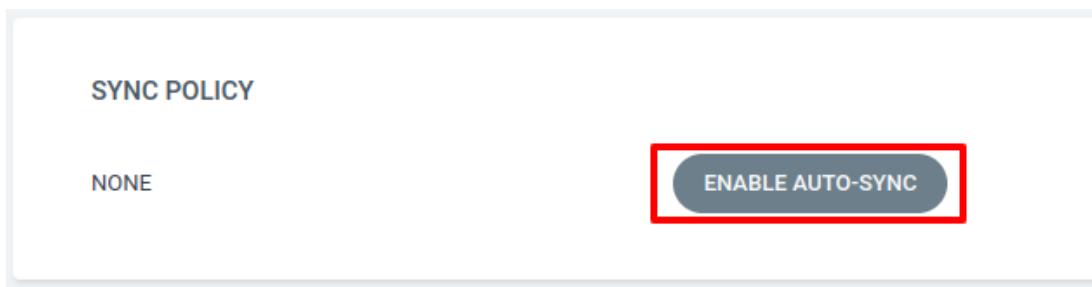
Then click **EDIT**:

VOTE-UI

**EDIT**

PROJECT	default
LABELS	
ANNOTATIONS	
CLUSTER	in-cluster ( <a href="https://kubernetes.default.svc">https://kubernetes.default.svc</a> )
NAMESPACE	vote-app
CREATED_AT	08/15/2022 16:14:21
REPO_URL	<a href="https://github.com/eeganlf/vote-deploy">https://github.com/eeganlf/vote-deploy</a>
TARGET_REVISION	HEAD
PATH	.
REVISION HISTORY LIMIT	
SYNC OPTIONS	<input checked="" type="checkbox"/> CreateNamespace
RETRY OPTIONS	Retry disabled
STATUS	Synced To HEAD (f0dff3a)
HEALTH	Healthy

Scroll to **SYNC POLICY** and select **ENABLE AUTO-SYNC**:



Scroll back to the top and click **SAVE**:

VOTE-UI	<b>SAVE</b>	CANCEL
PROJECT	default	
LABELS	No items	
ANNOTATIONS	No items	
CLUSTER	https://kubernetes.default.svc	URL ▾
NAMESPACE	vote-app	
CREATED_AT	08/15/2022 16:14:21	
REPO_URL	https://github.com/eeganlf/vote-deploy	
TARGET REVISION	HEAD	Branches ▾
PATH	.	
REVISION HISTORY LIMIT	10	
SYNC OPTIONS	<input type="checkbox"/> Skip Schema Validation <input type="checkbox"/> Prune Last <input type="checkbox"/> Respect Ignore Differences Prune Propagation Policy: foreground <input type="checkbox"/> Replace ▲ <input checked="" type="checkbox"/> Auto-Create Namespace <input type="checkbox"/> Apply Out of Sync Only	
RETRY OPTIONS	<input type="checkbox"/> Retry	

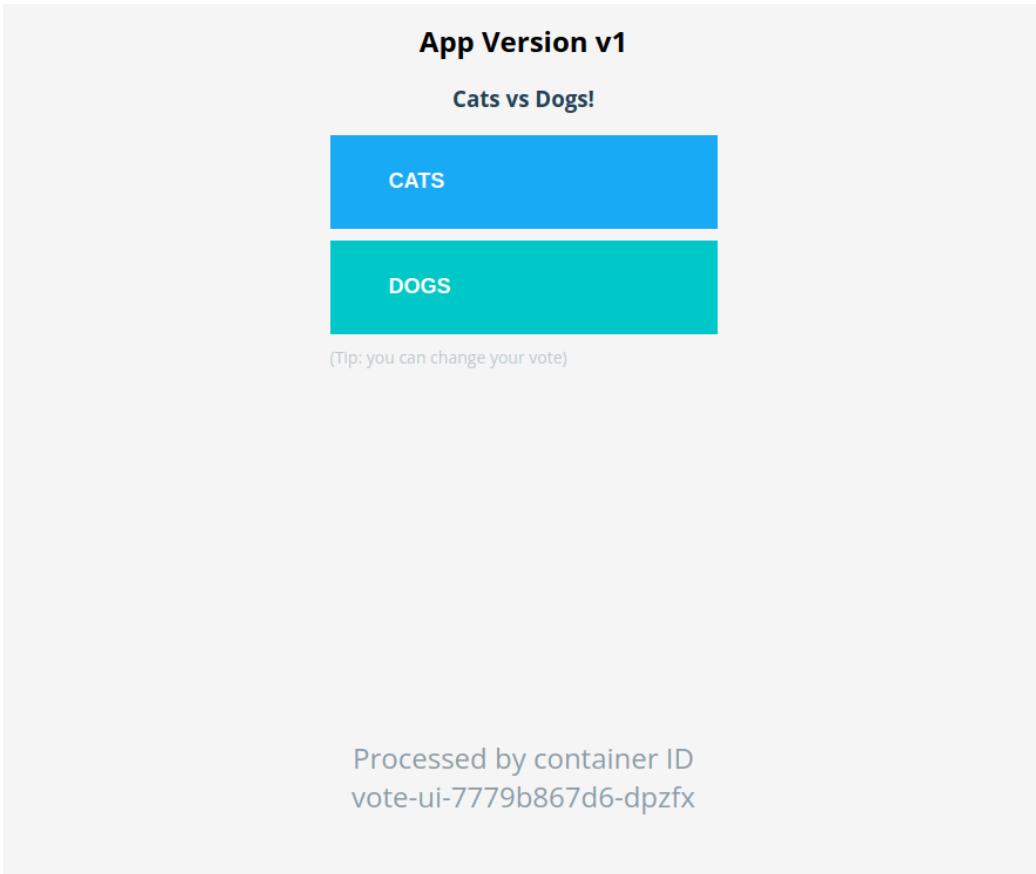
## Auto Deploy Changes

Change line 8 and 9 of `example-voting-app/vote/app.py` from **emacs** and **Vim** to **Cats** and **Dogs** as follows:

```
48 lines (38 sloc) | 1.14 KB

1 from flask import Flask, render_template, request, make_response, g
2 from redis import Redis
3 import os
4 import socket
5 import random
6 import json
7
8 option_a = os.getenv('OPTION_A', "Cats")
9 option_b = os.getenv('OPTION_B', "Dogs") option_a = os.getenv('OPTION_A', "Cats")
10 hostname = socket.gethostname()
11 version = 'v1'
12
13 app = Flask(__name__)
14
```

Commit your changes. After the `deployment` pipeline is complete, wait a few minutes for Argo CD to see the changes in the `vote-deploy` repository. Visit the public IP of one of your instances with the NODE PORT of the `vote-ui` as before and you should see the changes:



Congratulations! You have achieved CI/CD and Continuous Deployment with Jenkins, Kubernetes, and Argo CD.

## Lab Summary

In this lab, we configured Argo CD to deploy the frontend of our example voting application and set up Continuous Deployment.