|  |  |  |  |
| --- | --- | --- | --- |
| Developed By | Date | Approved By | Course Type (New/Update) |
|  | 20XX-XX-XX | Wang Xiaomin | Update |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Course Code | Product | Product Version | Course Issue |
|  |  |  |  |

# Contents

[1. Lab Environment 1](#_Toc137639850)

[1.1 About This Course 1](#_Toc137639851)

[1.2 Lab Environment of Ansible Systems 2](#_Toc137639852)

[1.3 System Information 2](#_Toc137639853)

[2. Jenkins Installation 4](#_Toc137639854)

[2.1 Introduction 4](#_Toc137639857)

[2.1.1 About this lab 4](#_Toc137639864)

[2.1.2 Objectives 4](#_Toc137639865)

[2.3 Configuration Steps 4](#_Toc137639867)

[3. Version Control with GitHub 11](#_Toc137639868)

[3.1 Introduction 11](#_Toc137639870)

[3.1.1 About this lab 11](#_Toc137639873)

[3.1.2 Objectives 11](#_Toc137639874)

[3.2 Configuration Steps 11](#_Toc137639875)

[4. Manage Git Repository 20](#_Toc137639876)

[4.1 Introduction 20](#_Toc137639878)

[4.1.1 About this lab 20](#_Toc137639884)

[4.1.2 Objectives 20](#_Toc137639885)

[4.2 Configuration Steps 20](#_Toc137639886)

[5. Create your first Job and Enter some simple configuration 24](#_Toc137639887)

[5.1 Introduction 24](#_Toc137639889)

[5.1.1 About this lab 24](#_Toc137639896)

[5.1.2 Objectives 24](#_Toc137639897)

[5.2 Configuration Steps 24](#_Toc137639898)

[6. Configuring a free style project 25](#_Toc137639899)

[6.1 Introduction 25](#_Toc137639902)

[6.1.1 About this lab 25](#_Toc137639909)

[6.1.2 Objectives 25](#_Toc137639910)

[6.2 Prerequisites 25](#_Toc137639911)

[6.3 Configuration Steps 25](#_Toc137639925)

[7. Creating Builds with Build Step Configuration, Statuses and Storing Artifacts 27](#_Toc137639912)

[7.1 Introductions 27](#_Toc137639915)

[7.1.1 About this lab 27](#_Toc137639923)

[7.1.2 Objectives 27](#_Toc137639924)

[7.2 Configuration Steps 27](#_Toc137639925)

[8. Using Plugins in Jenkins 30](#_Toc137639926)

[8.1 Introduction 3](#_Toc137639928)0

[8.1.1 About this lab 3](#_Toc137639931)0

[8.1.2 Objectives 3](#_Toc137639932)0

[8.2 Configuration Steps 3](#_Toc137639937)0

[9. Install Docker and Run a Container 3](#_Toc137639938)2

[9.1 Introduction 3](#_Toc137639940)2

[9.1.1 About this lab 32](#_Toc137639941)

[9.1.2 Objectives 32](#_Toc137639942)

[9.2 Configuration Steps 32](#_Toc137639947)

[10. Register for Docker and Exercise the commands 34](#_Toc137639948)

[10.1 Introduction 34](#_Toc137639950)

[10.1.1 About this lab 34](#_Toc137639953)

[10.1.2 Objectives 34](#_Toc137639954)

[10.2 Configuration Steps 34](#_Toc137639955)

[11. Run a Simple Pipeline Job in Jenkins 3](#_Toc137639956)6

[11.1 Introduction](#_Toc137639958) 36

[11.1.1 About this lab 36](#_Toc137639961)

[11.1.2 Objectives 36](#_Toc137639962)

[11.2 Configuration Steps 36](#_Toc137639963)

[12. Run a CI Job in Jenkins](#_Toc137639966) 38

[12.1 Introduction 38](#_Toc137639968)

[12.1.1 About this lab 38](#_Toc137639971)

[12.1.2 Prerequisites 38](#_Toc137639972)

[12.1.3 Objectives 38](#_Toc137639972)

[12.2 Configuration Steps 38](#_Toc137639977)

[13. Run a CT Pipeline Job in Jenkins 4](#_Toc137639978)1

[13.1 Introduction 4](#_Toc137639977)1

[13.1.1 About this lab 41](#_Toc137639852)

[13.1.2 Prerequisites 41](#_Toc137639972)

[13.1.3 Objectives 41](#_Toc137639853)

[13.2 Configartion Steps 41](#_Toc137639853)

[14. Docker Compose 43](#_Toc137639854)

14.1 [Introduction 43](#_Toc137639857)

[14.1.1 About this lab 43](#_Toc137639864)

[14.1.2 Prerequisites 4](#_Toc137639972)3

[14.1.3 Objectives 43](#_Toc137639865)

[14.2 Configuration Steps 3](#_Toc137639867)

[15. Docker Compose Lab 2 46](#_Toc137639868)

[15.1 Introduction 46](#_Toc137639870)

[15.1.1 About this lab 46](#_Toc137639873)

[15.1.2 Prerequisites 46](#_Toc137639972)

[15.1.3 Objectives 46](#_Toc137639874)

[15.2 Configuration Steps 46](#_Toc137639875)

[16. Docker Swarm Initialization 49](#_Toc137639876)

[16.1 Introduction 49](#_Toc137639878)

[16.1.1 About this lab 49](#_Toc137639884)

[16.1.2 Prerequisites 49](#_Toc137639972)

[16.1.3 Objectives 49](#_Toc137639885)

[16.2 Configuration Steps 49](#_Toc137639886)

[17. Docker Swarm Node Management 51](#_Toc137639887)

[17.1 Introduction 51](#_Toc137639889)

[17.1.1 About this lab 51](#_Toc137639896)

[17.1.2 Prerequisites 51](#_Toc137639972)

[17.1.3 Objectives 51](#_Toc137639897)

[17.2 Configuration Steps 51](#_Toc137639898)

[18. Load Balancing with Docker Swarm 54](#_Toc137639899)

[18.1 Introduction 54](#_Toc137639902)

[18.1.1 About this lab 54](#_Toc137639909)

[18.1.2 Prerequisites 54](#_Toc137639972)

[18.1.3 Objectives 54](#_Toc137639910)

[18.2 Configuration Steps 54](#_Toc137639911)

[19. Kubernetes Orchestration 57](#_Toc137639912)

[19.1 Introductions 57](#_Toc137639915)

[19.1.1 About this lab 57](#_Toc137639923)

[19.1.2 Prerequisites 57](#_Toc137639972)

[19.1.3 Objectives 57](#_Toc137639924)

[19.2 Configuration Steps 57](#_Toc137639925)

[20. Cloud Formation 59](#_Toc137639926)

[20.1 Introduction 59](#_Toc137639928)

[20.1.1 About this lab 59](#_Toc137639931)

[20.1.2 Prerequisites 59](#_Toc137639972)

[20.1.3 Objectives 59](#_Toc137639932)

[20.2 Configuration Steps 59](#_Toc137639937)

[21. Terraform………….. 60](#_Toc137639938)

[31.1 Introduction 60](#_Toc137639940)

[21.1.1 About this lab 60](#_Toc137639941)

[21.1.2 Prerequisites 60](#_Toc137639972)

21[.1.3 Objectives 60](#_Toc137639942)

[21.2 Configuration Steps 60](#_Toc137639947)

[22. Chef…………………. 63](#_Toc137639948)

[22.1 Introduction 63](#_Toc137639950)

[22.1.1 About this lab 63](#_Toc137639953)

[22.1.2 Prerequisites 63](#_Toc137639972)

[22.1.3 Objectives 63](#_Toc137639954)

[22.2 Configuration Steps 63](#_Toc137639955)

[23. Ansible…………….. 6](#_Toc137639956)5

[11.1 Introduction 65](#_Toc137639958)

[11.1.1 About this lab 65](#_Toc137639961)

[11.1.2 Objectives 65](#_Toc137639962)

[11.2 Configuration Steps 65](#_Toc137639963)

[24. Creating an application using Jenkins, Docker and Puppet 67](#_Toc137639966)

[24.1 Introduction 67](#_Toc137639968)

[24.1.1 About this lab 67](#_Toc137639971)

[24.1.2 Prerequisites 67](#_Toc137639972)

[24.1.3 Objectives 67](#_Toc137639972)

[24.2 Configuration Steps 67](#_Toc137639963)

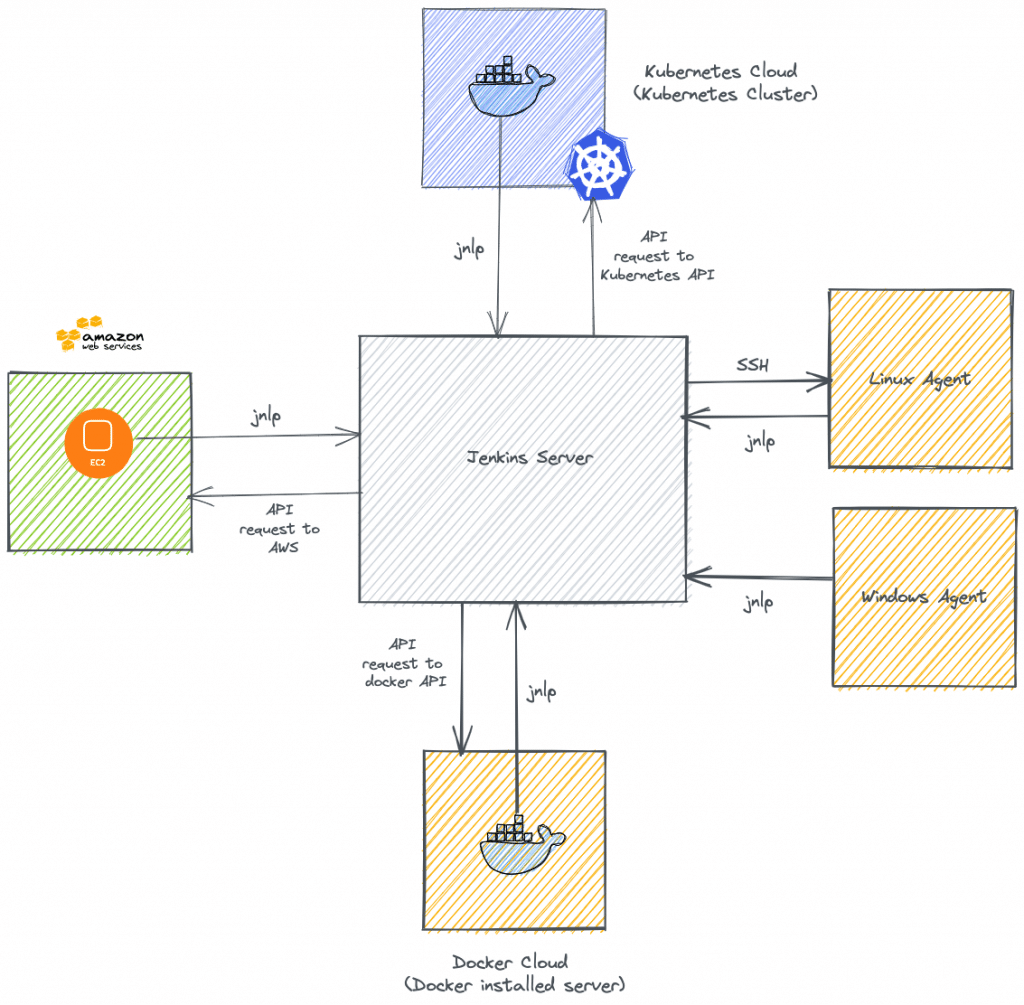
# Lab Environment

## About This Course

The course is focuses on the usage of DevOps in Software Development. We will cover the following topics:

Perform Continuous Development, Continuous Testing, Continuous Integration, Continuous Deployment and Continuous Monitoring using DevOps tools - Git, GitHub, Maven, Jenkins, Dockers, Kubernetes, CloudFormation, Terraform, Chef, Ansible, Nagios and ELK.

## Lab Envirnoment for DevOps Tools and Usage



## Prerequisites

* Learners must have basic knowledge of Virtualization, Cloud Computing, Linux, Using Remote Access tools, vi/vim editor tools
* Google Chrome to be installed on participant’s laptops
* All the other tools required for the program would be installed on a separate remote machine and given access which Learners would be using for labs

# Jenkins Installation

## Introduction

### About this lab

* Upon completion of this task, you will be able to perform:
  + Jenkin Installation.

### Objectives

* Upon completion of this task, you will be able to:
  + Install and run Jenkins

## Configuration Steps

### Installing Jenkins

#### Docker (Any Platform)

To install Jenkins through Docker, you will, of course, need Docker installed! With Docker, you can use the following command to install Jenkins:

docker run -d -p 8080:8080 --name jenkins jenkins/jenkins:lts-alpine

#### Windows

An installer can be downloaded and run to get Jenkins running on Windows here: <https://jenkins.io/download/>

#### Linux

To install Jenkins on Linux, put the below script in a file on the machine you wish to install Jenkins on and execute it:

#!/bin/bash

if type apt > /dev/null; then

pkg\_mgr=apt

if [ $(uname -v) == \*Debian\* ]; then

java="default-jre"

else

java="openjdk-11-jre"

fi

elif type yum /dev/null; then

pkg\_mgr=yum

java="java"

fi

echo "updating and installing dependencies"

sudo ${pkg\_mgr} update

sudo ${pkg\_mgr} install -y ${java} wget git > /dev/null

echo "configuring jenkins user"

sudo useradd -m -s /bin/bash jenkins

echo "downloading latest jenkins WAR"

sudo su - jenkins -c "curl -L https://updates.jenkins-ci.org/latest/jenkins.war --output jenkins.war"

echo "setting up jenkins service"

sudo tee /etc/systemd/system/jenkins.service << EOF > /dev/null

[Unit]

Description=Jenkins Server

[Service]

User=jenkins

WorkingDirectory=/home/jenkins

ExecStart=/usr/bin/java -jar /home/jenkins/jenkins.war

[Install]

WantedBy=multi-user.target

EOF

sudo systemctl daemon-reload

sudo systemctl enable jenkins

sudo systemctl restart jenkins

sudo su - jenkins << EOF

until [ -f .jenkins/secrets/initialAdminPassword ]; do

sleep 1

echo "waiting for initial admin password"

done

until [[ -n "\$(cat .jenkins/secrets/initialAdminPassword)" ]]; do

sleep 1

echo "waiting for initial admin password"

done

echo "initial admin password: \$(cat .jenkins/secrets/initialAdminPassword)"

EOF

Please note, if you are using the below script, you will need *sudo* access on the machine you execute it on. Sudo will give you administrative access to your Linux machine, which in turn will allow you to install Jenkins.

### Unlocking Jenkins

To begin the setup process for Jenkins, you will need to go to port 8080 on your machine.

Jenkins setup requires an initial admin password to unlock the setup and installation. This is a simple verification process Jenkins includes to ensure an administrator is installing the tool.

The initial admin password is stored on the file system of the machine that Jenkins is running on. The page states where this file is located - you just need to copy the contents of it into the text field and click Continue.

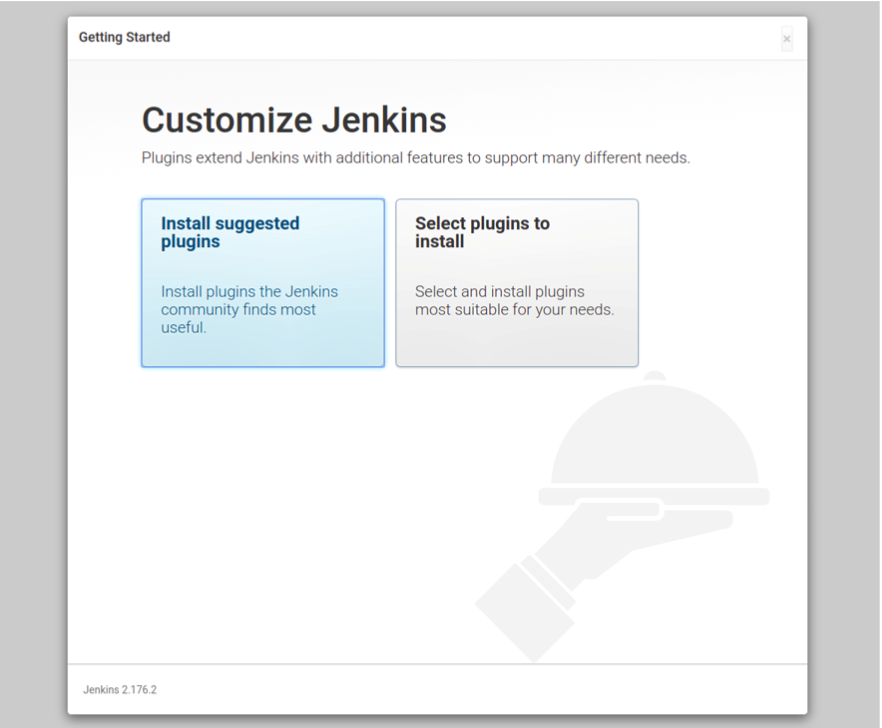
cat /var/jenkins\_home/secrets/initialAdminPassword

### Customize Jenkins

Jenkins is highly configurable due to the amount of plugins that you can install.

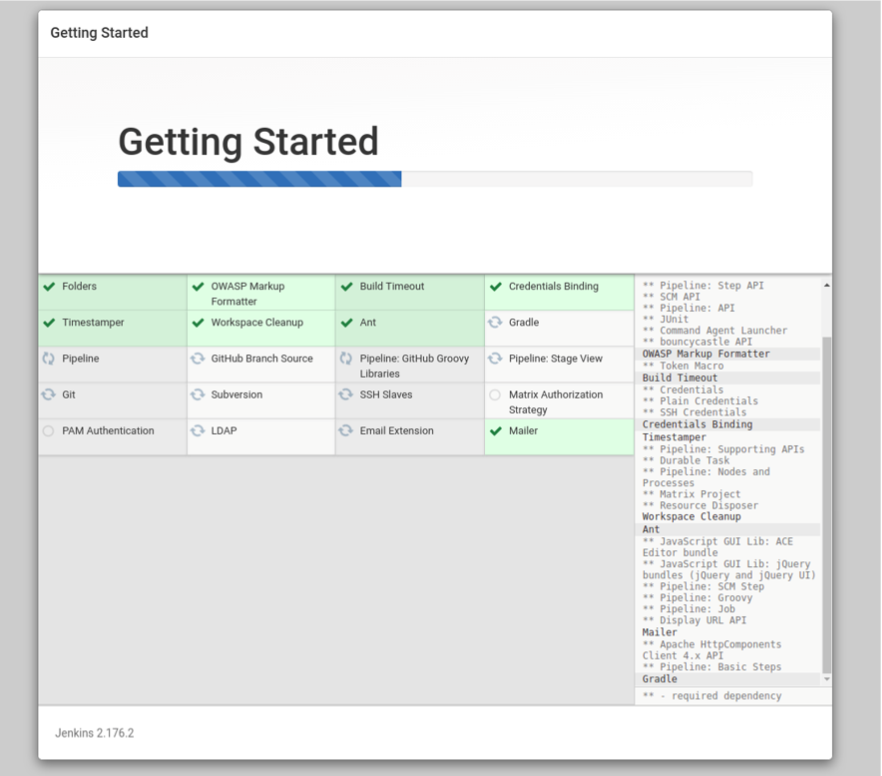
This is fantastic, but if you are new to Jenkins, you might not have much of an idea about what plugins you would want!

Fortunately, the setup gives you the option to install suggested plugins - select this option:



Jenkins will then begin to install its suggested plugins. For this part, there isn't much to do but wait!

If plugins are failing to install, make sure that you have the latest version on Jenkins installed.



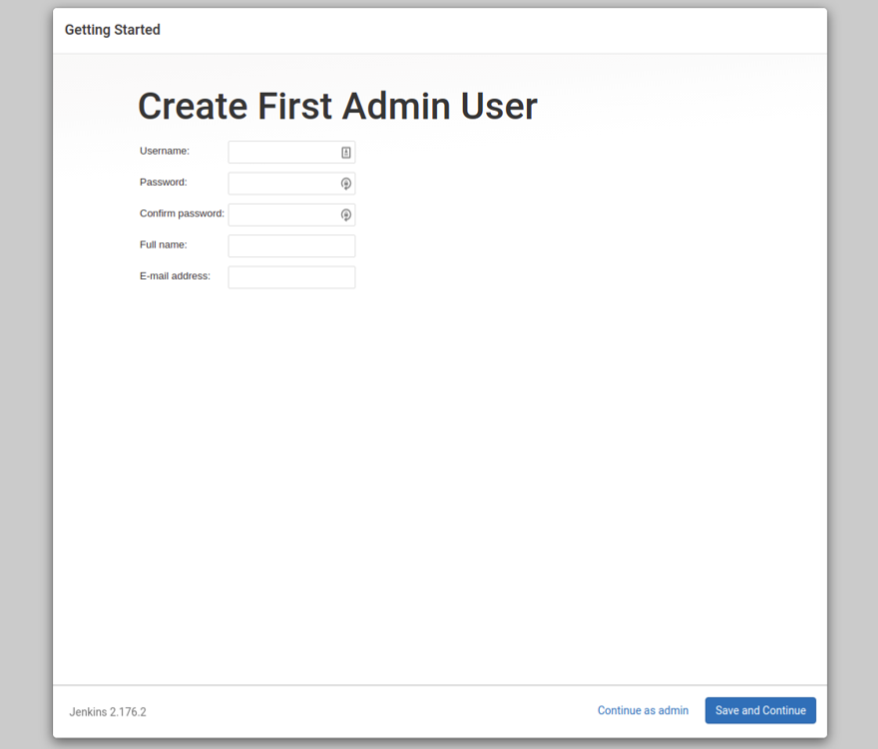
### Create the First Admin User

You will then be presented with a form to create the first admin user.

You can either fill in the form to have your details saved into the first admin account, or continue as admin.

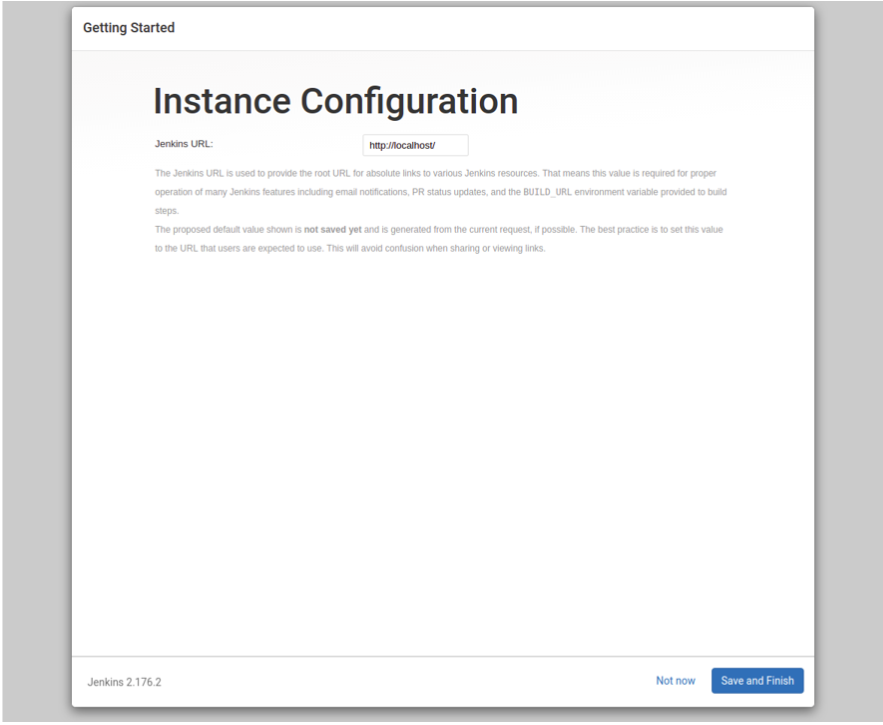
If you pick the second option, the admin user account name is admin and the password will remain initial admin password that you entered.

Be careful not to enter actual passwords and information here, especially if you are connected to a Jenkins instance over the internet with no TLS or SSL configured (HTTPS secure connection).



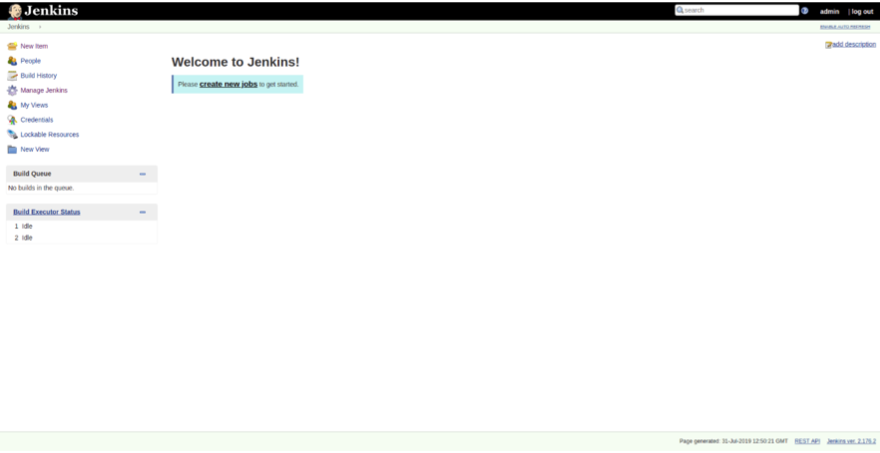
### Instance Configuration

All you will need to for this step is to select Save and Finish.



### Dashboard

Now that you have Jenkins set up and installed, you will be presented with the Jenkins dashboard on your screen:



Congratulations, you have installed the Jenkins instance on a machine!

# Version Control with GitHub



## Introduction



### About this lab

* Upon completion of this task, you will be able to perform:
  + Version Control with GitHub.

### Objectives

* Upon completion of this task, you will be able to:

1. Create a branch
2. Commit a file
3. Open a pull request
4. Merge your pull request

## Configuration Steps

1. Please sign in to GitHub and create a repository from the link below and follow the steps:

https://github.com/new

**Step 1: Create a branch**

*Welcome to "Introduction to GitHub"! 👋*

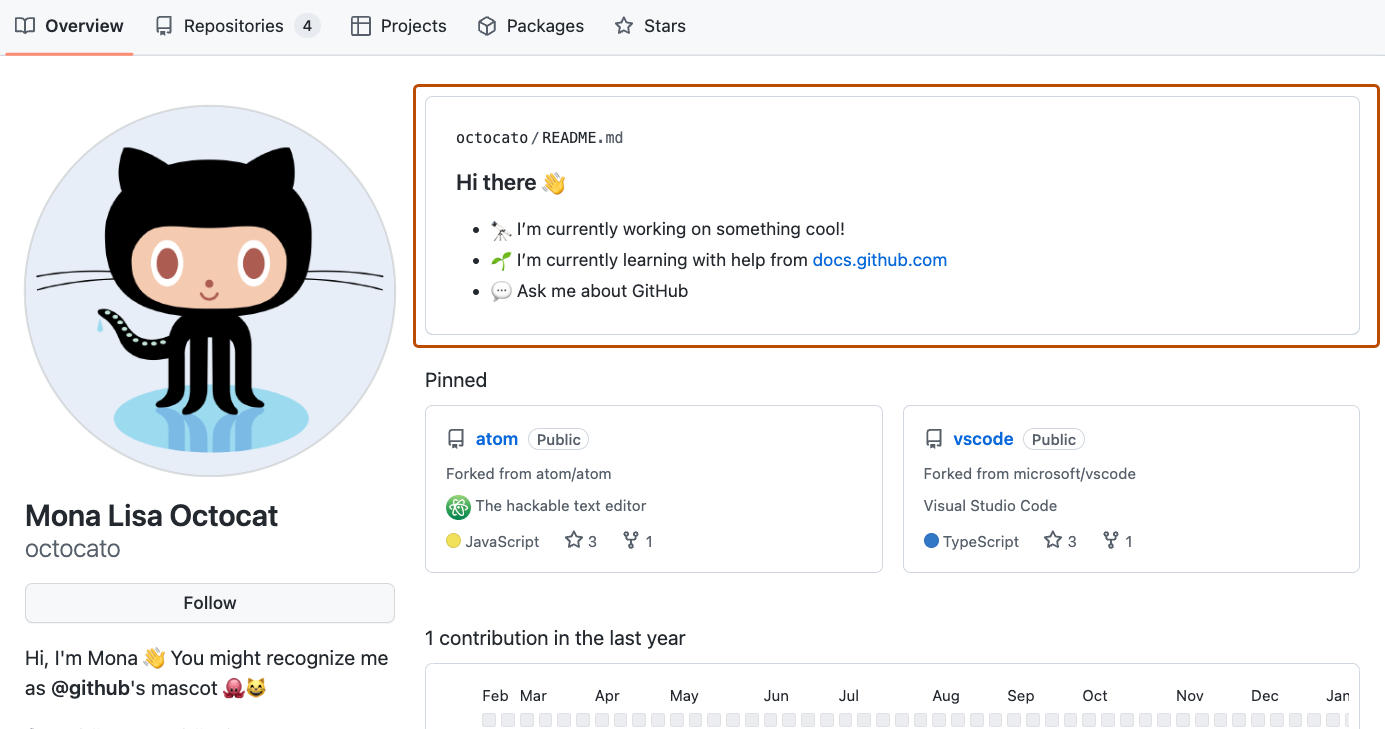
**What is GitHub?**: GitHub is a collaboration platform that uses [*Git*](https://docs.github.com/get-started/quickstart/github-glossary#git) for versioning. GitHub is a popular place to share and contribute to [open-source](https://docs.github.com/get-started/quickstart/github-glossary#open-source) software.   
📺 [Video: What is GitHub?](https://www.youtube.com/watch?v=pBy1zgt0XPc)

**What is a repository?**: A [*repository*](https://docs.github.com/get-started/quickstart/github-glossary#repository) is a project containing files and folders. A repository tracks versions of files and folders. For more information, see "[About repositories](https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories)" from GitHub Docs.

**What is a branch?**: A [*branch*](https://docs.github.com/en/get-started/quickstart/github-glossary#branch) is a parallel version of your repository. By default, your repository has one branch named main and it is considered to be the definitive branch. Creating additional branches allows you to copy the main branch of your repository and safely make any changes without disrupting the main project. Many people use branches to work on specific features without affecting any other parts of the project.

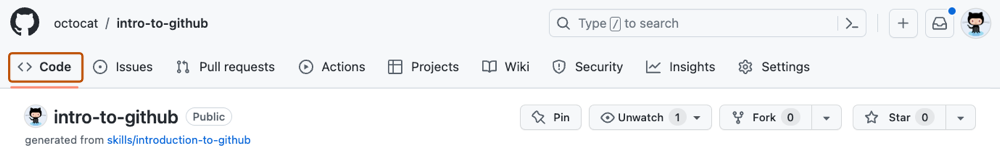
Branches allow you to separate your work from the main branch. In other words, everyone's work is safe while you contribute. For more information, see "[About branches](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches)".

**What is a profile README?**: A [*profile README*](https://docs.github.com/account-and-profile/setting-up-and-managing-your-github-profile/customizing-your-profile/managing-your-profile-readme) is essentially an "About me" section on your GitHub profile where you can share information about yourself with the community on GitHub.com. GitHub shows your profile README at the top of your profile page. For more information, see "[Managing your profile README](https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-profile/customizing-your-profile/managing-your-profile-readme)".

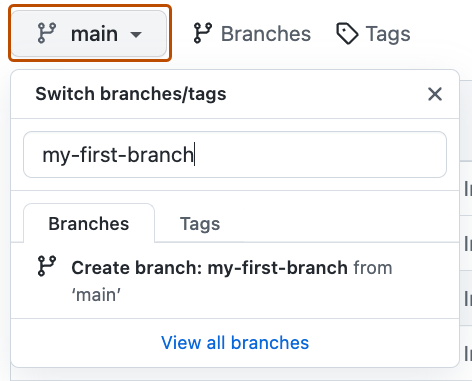
[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/main/images/profile-readme-example.png)

**⌨️ Activity: Your first branch**

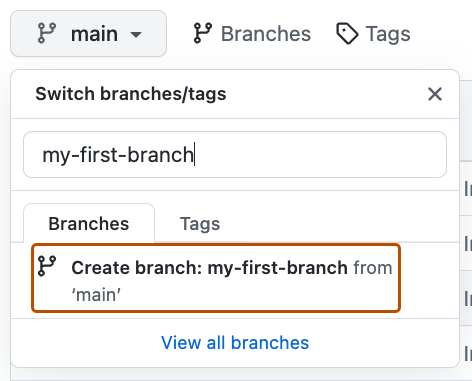
1. Open a new browser tab and navigate to your newly made repository. Then, work on the steps in your second tab while you read the instructions in this tab.
2. Navigate to the **< > Code** tab in the header menu of your repository.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/main/images/code-tab.png)

1. Click on the **main** branch drop-down.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/main/images/main-branch-dropdown.png)

1. In the field, name your branch my-first-branch. In this case, the name must be my-first-branch to trigger the course workflow.
2. Click **Create branch: my-first-branch** to create your branch.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/main/images/create-branch-button.png)

The branch will automatically switch to the one you have just created. The **main** branch drop-down bar will reflect your new branch and display the new branch name.

**Step 2: Commit a file**

*You created a branch! 🎉*

Creating a branch allows you to edit your project without changing the main branch. Now that you have a branch, it’s time to create a file and make your first commit!

**What is a commit?**: A [*commit*](https://docs.github.com/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits) is a set of changes to the files and folders in your project. A commit exists in a branch. For more information, see "[About commits](https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits)".

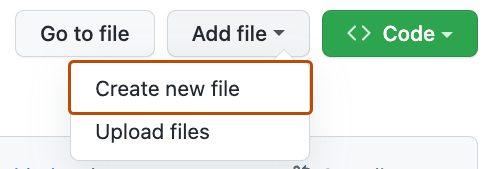
**⌨️ Activity: Your first commit**

The following steps will guide you through the process of committing a change on GitHub. A commit records changes in renaming, changing content within, creating a new file, and any other changes made to your project. For this exercise, committing a change requires first adding a new file to your new branch.

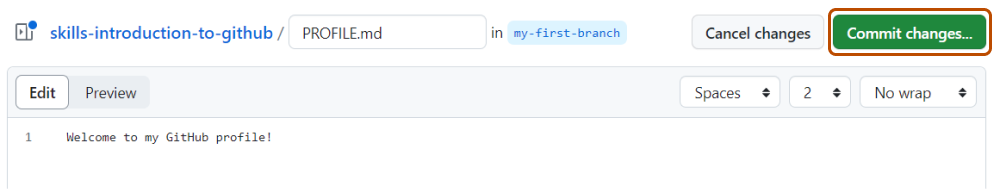
Note

.md is a file extension that creates a Markdown file. You can learn more about Markdown by visiting "[Basic writing and formatting syntax](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax)" in our docs or by taking the "[Communicating using Markdown](https://github.com/skills/communicate-using-markdown)" Skills course.

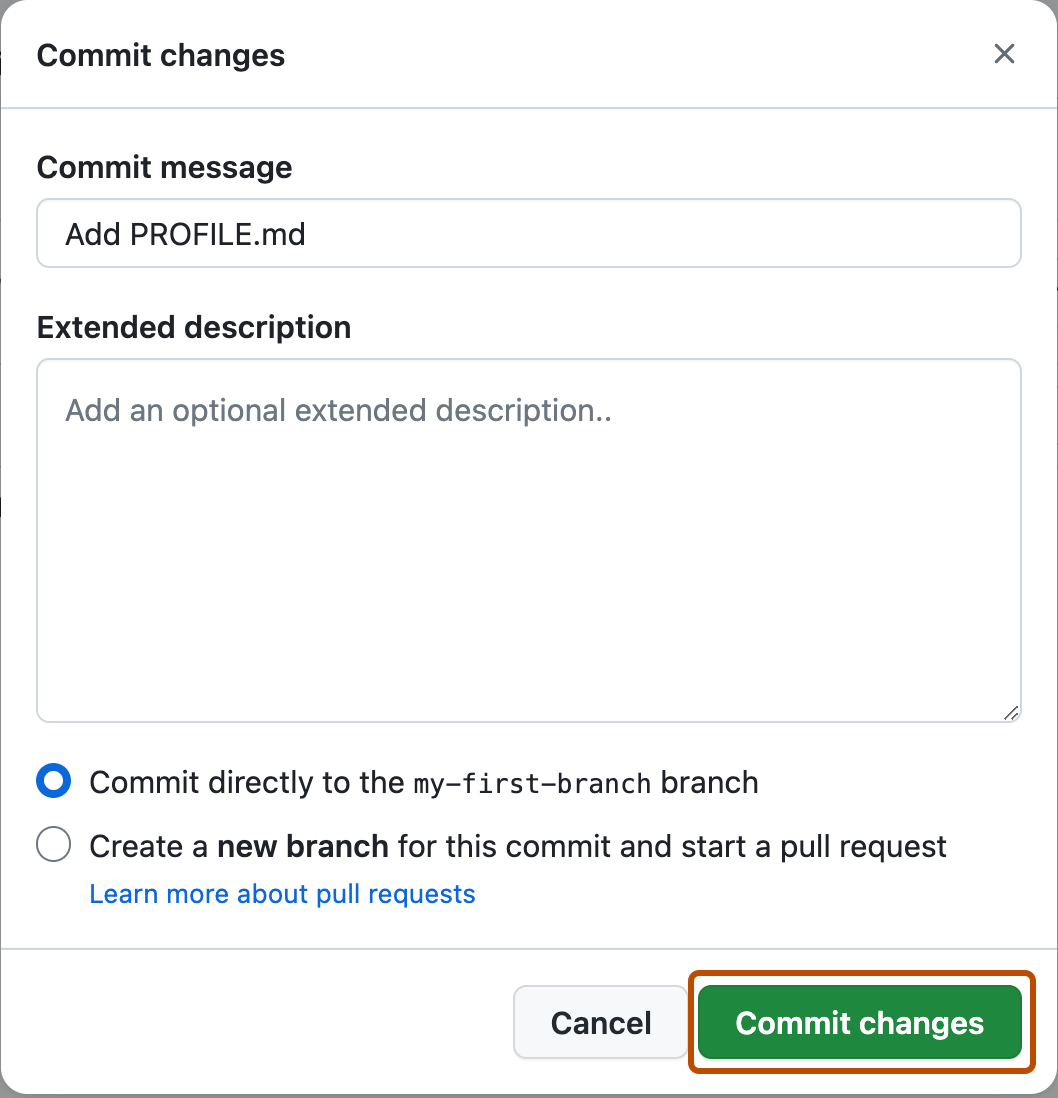
1. On the **< > Code** tab in the header menu of your repository, make sure you're on your new branch my-first-branch.
2. Select the **Add file** drop-down and click **Create new file**.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/my-first-branch/images/create-new-file.png)

1. In the **Name your file...** field, enter PROFILE.md.
2. In the **Enter file contents here** area, copy the following content to your file:
3. Welcome to my GitHub profile!

 [](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/my-first-branch/images/my-profile-file.png)

 Click **Commit changes...** in the upper right corner above the contents box. For commits, you can enter a short commit message that describes what changes you made. This message helps others know what's included in your commit. GitHub offers a simple default message, but let's change it slightly for practice. First, enter Add PROFILE.md in the first text-entry field titled "Commit message".

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/my-first-branch/images/commit-full-screen.png)

 In this lesson, we'll ignore the other fields and click **Commit changes**.

**Step 3: Open a pull request**

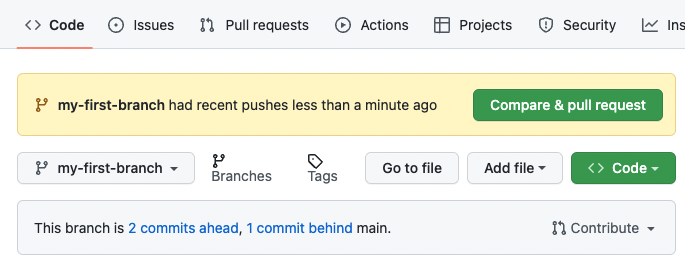
*Nice work making that commit! ✨*

Now that you have made a change to the project and created a commit, it’s time to share your proposed change through a pull request!

**What is a pull request?**: Collaboration happens on a [*pull request*](https://docs.github.com/en/get-started/quickstart/github-glossary#pull-request). The pull request shows the changes in your branch to other people and allows people to accept, reject, or suggest additional changes to your branch. In a side by side comparison, this pull request is going to keep the changes you just made on your branch and propose applying them to the main project branch. For more information about pull requests, see "[About pull requests](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests)".

**⌨️ Activity: Create a pull request**

You may have noticed after your commit that a message displayed indicating your recent push to your branch and providing a button that says **Compare & pull request**.

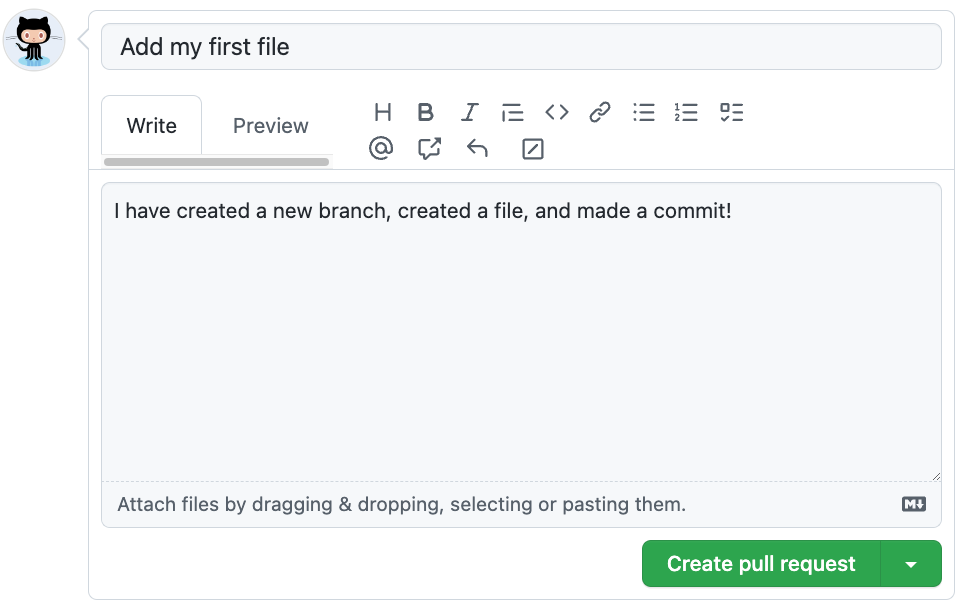
[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/my-first-branch/images/compare-and-pull-request.png)

To create a pull request automatically, click **Compare & pull request**, and then skip to step 6 below. If you don't click the button, the instructions below walk you through manually setting up the pull request.

1. Click on the **Pull requests** tab in the header menu of your repository.
2. Click **New pull request**.
3. In the **base:** dropdown, make sure **main** is selected.
4. Select the **compare:** dropdown, and click my-first-branch.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/my-first-branch/images/pull-request-branches.png)

1. Click **Create pull request**.
2. Enter a title for your pull request. By default, the title will automatically be the name of your branch. For this exercise, let's edit the field to say Add my first file.
3. The next field helps you provide a description of the changes you made. Here, you can add a description of what you’ve accomplished so far. As a reminder, you have: created a new branch, created a file, and made a commit.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/my-first-branch/images/Pull-request-description.png)

1. Click **Create pull request**. You will automatically be navigated to your new pull request.

Note

You may see evidence of GitHub Actions running on the tab with the pull request opened! The image below shows a line you might see on your pull request after the Action finishes running.

[screenshot of an example of an actions line](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/my-first-branch/images/Actions-to-step-4.png)

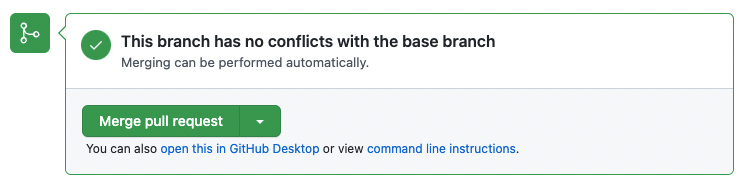
**Step 4: Merge your pull request**

*Nicely done! 😎*

You successfully created a pull request. You can now merge your pull request.

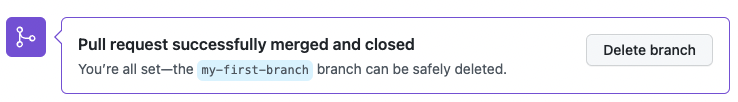
**What is a merge?**: A [*merge*](https://docs.github.com/en/get-started/quickstart/github-glossary#merge) adds the changes in your pull request and branch into the main branch. For more information about merges, see "[Merging a pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request)."

As noted in the previous step, you may have seen evidence of GitHub Actions running which automatically progresses your instructions to the next step. You'll have to wait for it to finish before you can merge your pull request. It will be ready when the merge pull request button is green.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/main/images/Green-merge-pull-request.png)

**⌨️ Activity: Merge the pull request**

1. Click **Merge pull request**.
2. Click **Confirm merge**.
3. Once your branch has been merged, you don't need it anymore. To delete this branch, click **Delete branch**.

[](https://github.com/deepanshuMeteor/skills-introduction-to-github/blob/main/images/delete-branch.png)

# Manage Git Repository Lab



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  + Manage Git Repository.

### Objectives

* Upon completion of this task, you will be able to:
  1. Open a pull request
  2. Assign yourself
  3. Leave a review
  4. Suggest changes
  5. Apply changes
  6. Merge your pull request

## Configuration Steps

1. Please sign in to GitHub and create a repository from the link below and follow the steps:

https://github.com/new

**Step 1: Open Pull Request**

Welcome to "Review pull requests"! 👋

Let's get started by opening a pull request.

**What is a pull request?**: Collaboration happens on a pull request. The pull request shows the changes in your branch to other people. This pull request is going to keep the changes you just made on your branch and propose applying them to the main branch.

**⌨️ Activity: Create a pull request**

* + Click on the **Pull requests** tab in your repository.
  + Click **New pull request**.
  + In the **base:** dropdown, make sure **main** is selected.
  + Select the **compare:** dropdown, and click update-game.
  + Click **Create pull request**.
  + Enter a title for your pull request: Update the game over message.
  + Enter a description for your pull request: Update the game over message so people know how to play again!
  + Click **Create pull request**.

**Step 2: Assign yourself**

Great job opening that pull request! 👋

**What is a pull request review?**: Reviewing a pull request is an opportunity to examine another contributor's changes and give them feedback. It's an awesome opportunity to learn more about how the project works and how others solve problems.

The best way to get a review is to ask for one. On GitHub, you can ask someone to review a pull request by assigning them as a reviewer or assignee. If you are not ready for review, consider [creating a draft pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request) instead.

⌨️ Activity: Assign yourself

* + Open the pull request you just created.
  + Under **Assignees** on the right side of the screen, add yourself.

Because you created the pull request, you can't assign yourself as a reviewer, but feel free to assign a friend as a reviewer instead to see how it works 😄

**Step 3: Leave a Review**

You assigned yourself! 🎉

Pull request reviews ensure quality and maintain momentum of changes to your project.

When reviewing a pull request:

1. Review the title and body of the pull request, and possibly any associated issue, to understand the intended change.
2. Review the [diff](https://docs.github.com/en/get-started/quickstart/github-glossary#diff), the comparison of the proposed code, in the context of the whole project.
3. For most things, try out the proposed change. Check if the actual change matches the intention. Find the repository's [contributing guide](https://docs.github.com/en/communities/setting-up-your-project-for-healthy-contributions/setting-guidelines-for-repository-contributors) to find out how to review the changes.

In your review comments:

* + Identify potential issues, risks, and limitations.
  + Suggest changes and improvements.
  + Share awareness of upcoming changes that the pull request doesn't account for.
  + Ask questions to verify shared understanding.
  + Highlight what the author did well and should keep doing.
  + Prioritize the most important feedback.
  + Be concise and provide meaningful detail.
  + Treat the pull request author with kindness and empathy.

When an approval or request for changes is not yet needed, consider using **comments**. An **approval** lets the author know you believe the pull request is safe to merge. **Requesting changes** lets the author know you believe the pull request is not ready to merge.

**⌨️ Activity: Leave a review**

* + On the pull request, click **Files changed**.
  + Click **Review changes**.
  + Add a comment with your initial thoughts on the pull request.
  + Select comment. You won't be able to approve or request changes to your own pull request.
  + Click **Submit review**.

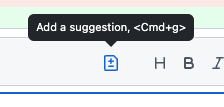
**Step 4: Suggest Changes**

Nice work reviewing that pull request ✨

Now that you have explored the different ways you can review a pull request it is time to learn how to use suggest changes.

**What is suggest changes?**: This feature enables you to recommend a change to a pull request that the author can commit with the push of a button.

⌨️ Activity: Suggest changes

* + On the pull request, click **Files changed**.
  + Find the index.html changes.
  + Hover your cursor next to the line numbers on the left side of the page.
  + Click the blue plus icon.
  + After the comment form appears, click the **Add a suggestion** button.   
    [](https://user-images.githubusercontent.com/97056108/184449714-61e8ee51-824a-48c1-9436-2dfd67f2c070.png)
  + Edit the suggestion.
  + Click **Add a single comment**.

**Step 5: Apply Suggested Changes**

Nicely done suggesting changes! 🥳

Now let's see how easy it is to [apply your suggestion](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/incorporating-feedback-in-your-pull-request).

**⌨️ Activity: Apply suggested changes**

* + Click **Commit suggestion**.
  + Type a commit message.
  + Click **Commit changes**.

**Step 6: Merge Pull Request**

You can now [merge](https://docs.github.com/en/get-started/quickstart/github-glossary#merge) your pull request!

**⌨️ Activity: Merge your pull request**

* + Click **Merge pull request**.
  + Delete the branch update-game (optional).

# Create your first job and enter some simple configuration



## 5.1Introduction



### About this lab

* Upon completion of this task, you will be able to:

Create and execute a job in Jenkins

### Objectives

* Create a Jenkins Job



## 5.2 Configuration Steps

* Choose any project from [GitHub](https://github.com)
* Enter some information about the project into the Project url field, for the GitHub project option
* Set a String Parameter for the job:
  + Set the NAME field to be VERSION
  + Set the Default Value field to be 0.1.0
  + Set the Description field to be Version of the application to build
* Select the Save button (you will be redirected to the job's page)
* Select Build with Parameters. You should then see the parameter that you configured
* Select Delete Project and confirm the deletion

# Configuring a Freestyle Project



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  1. Configure a Freestyle Project

### Objectives

* Create and Execute a Freestyle project in Jenkins

## Prerequisites

### GitHub Account

### Jenkins account

## Configuration Steps

### Create a New GitHub Repository

GitHub is, of course, where our code will be! Therefore, we will need to allow Jenkins to download the code frmo that repository and access it in the Job.

Set up a repository so we can configure a Jenkins job to access and use it in later steps:

1. Create a public GitHub repository for this exercise, you can call it jenkins-freestyle-project.
2. Add a script to the repository called run.sh, with the contents:

echo 'Hello from run.sh!'

### Create a Jenkins Job

The Jenkins job is going to be able to:

* download the repository that we created
* run the run.sh script

1. Create a new Freestyle Project on Jenkins. You can call this job whatever you like!
2. Configure the Job to download the repository:
   * Under Source Code Management, select Git
   * Enter https://github.com/[YOUR\_USERNAME]/jenkins-freestyle-project, replacing [YOUR\_USERNAME] with your GitHub username.
   * If your Git Repository has a main branch, then you will need to change the branch specifier from \*/master to \*/main.
3. In the Build section, create a build step to Execute shell and enter the following:

sh run.sh

### Run the Job

Now that everything is set up, Save the changes that were made and the Build the job.

Check the console output of the build to see that the job has executed. The end of the output will show that the script on the repository has run correctly:

+ sh run.sh

+ Hello from run.sh!

+ Finished: SUCCESS

### Clean Up

Feel free to now delete the created resources:

* Jenkins job
* GitHub Repository

# Creating builds with build step configuration, statuses and storing artifacts



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  1. Create builds with build step configuration
  2. Manage status
  3. Store artifacts

### Objectives

* Create a Jenkins job to start a workflow which will run the following steps:
  1. Creating builds with build step configuration
  2. Manage status
  3. Store artifacts

## Configuration Steps

### Create a Job

First, we will need a new Jenkins job to work on. Create a new job and name it whatever you like!

### Add a Build Step

Let's add a build step that we know will succeed. Select the Add build step button and add an Execute shell build step.

Add the following script into the Command field:

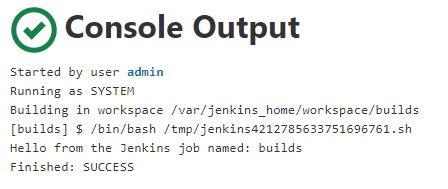
#!/bin/bash

echo "Hello from the Jenkins job named: ${JOB\_NAME}"

### Run the Job

Save the job and then build it. You should then have one successful build in your history for that job.

Once you navigate to the console output, you should see an output like this:



### Make the Build Fail

Now that the last build succeeded, let's see what a failed build looks like!

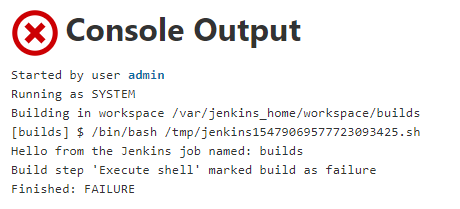
To do this, we will make the script we added to the command box \*"fail"\*.

Jenkins will treat any script or application exiting with a non-zero status as a failure. So, to create a failed build, add exit 1 to the script box, which will make the script exit with a code of 1:

#!/bin/bash

echo "Hello from the Jenkins job named: ${JOB\_NAME}"

exit 1



### Abort a build

You can remove the exit 1 from the build step to fix it.

Next, uyou are going to make the build run for 10 minutes, which will give you plenty of time to abort it.

Here is the code that should be in your script box now:

#!/bin/bash

echo "Hello from the Jenkins job named: ${JOB\_NAME}"

sleep 600

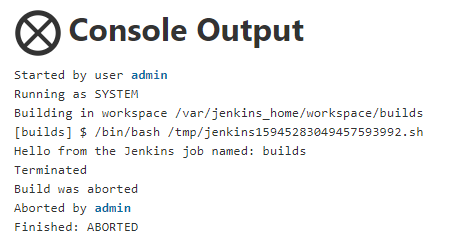
Click Save and Build Now to begin the build.

To abort the build, click the red x on the in-progress build.

Note: While the build is in progress, it will have the status of the previous build, but with a spinning icon.

Jenkins Running Build

This will result in an aborted build, the console output will look like this:



### Fix the Build and Create Artifacts

Removing the sleep 600 from the build step will ensure the build doesn't take 10 minutes to complete!

Next, change the script to create several files, and then put them in a zip archive called archive.zip:

#!/bin/bash

echo "Hello from the Jenkins job named: ${JOB\_NAME}"

touch 1.txt 2.txt 3.txt 4.txt 5.txt

zip archive.zip \*.txt

A Post-build Action must be configured to archive the zip files. Configure a simple post-build action to archive the created .zip file - refer to the [Artifacts](https://qa-community.co.uk/~/_/learning/jenkins-introduction/jenkins--builds#artifacts) section for guidance on how to configure this!

### Finish Up

Finish up by running the job. You should see artifacts on the project dashboard!

If this isn't present, try refreshing the page.

# Using plugins in Jenkins



## Introduction



### About this lab

In this tutorial, you will install a plugin and see what affect that has on our Jenkins instance.

### Objectives

* Upon completion of this task, you will be able to:
  1. Install a plugin with Jenkins Job

## Configuration Steps

**Before starting this tutorial, make sure you have a Jenkins instance up and running.**

1. Go to Manage Jenkins and click on Manage Plugins:
2. Click on the Available Tab and search "maven". Check Maven Integration and select Install without restart:
3. In Manage Jenkins > Global Tool Configuration, you will now see an option to configure the path to the Maven file:
4. If you create a New item, you can now see the Maven project option:
5. Inside this project, there are some options available that make integrating with Maven much easier:

The Maven plugin we installed allows much easier integration with Maven, and makes it simple to configure; this is the power of plugins!

To uninstall a plugin, you simply go back to Manage Plugins and go to the Installed tab:

# Install Docker and Run a container



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  1. Install Docker
  2. Run a container

### Objectives

* Install Docker
* Run a Container

## Configuration Steps

First, ensure that Docker is installed and your current user is in the Docker group.

Now, give *Docker* a go by running a **hello world container** using the docker run command:

# --rm flag removes the container when it exits or when the daemon exits, whichever happens first

docker run --rm hello-world

You will see *Docker* download the image and run it.  
The script in the *hello-world container* will create an output which describes what just happened, creating an output something like this:

Hello from Docker!

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

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

(amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

# Register for Docker and exercise the commands



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  1. register for Docker
  2. Run commands to search and register an Image

### Objectives

Register a Docker Image with linux commands.

## Configuration Steps

First, register an account with Docker [here](https://hub.docker.com/)

Once created, authenticate the Docker CLI to Dockerhub with the login command.

docker login

Next, let's download the official Java image.

docker search java

The search command returns a table of images relevant to the search term with their description and whether they are an official image.

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

node Node.js is a JavaScript-based platform for s… 9149 [OK]

tomcat Apache Tomcat is an open source implementati… 2814 [OK]

openjdk OpenJDK is an open-source implementation of … 2393 [OK]

java Java is a concurrent, class-based, and objec… 1976 [OK]

ghost Ghost is a free and open source blogging pla… 1240 [OK]

couchdb CouchDB is a database that uses JSON for doc… 364 [OK]

jetty Jetty provides a Web server and javax.servle… 343 [OK]

groovy Apache Groovy is a multi-faceted language fo… 100 [OK]

We see the official java image on line 4. Use the docker pull command to download the java image.

docker pull java

REPOSITORY TAG IMAGE ID CREATED SIZE

java latest d23bdf5b1b1b 3 years ago 643MB

**Rename the Java 8 Docker image so that it is prefixed with your Docker hub name**

Replace <your\_docker\_username> with your Docker username and run the following command:

docker tag java <your\_docker\_username>/java:8

Now, when we run docker images, we can see two images. We have created a new image with a different name. Now we can push the newly tagged image to our repository by tagging it in the format [USERNAME]/[IMAGE]:[TAG].

docker push <your\_docker\_username>/java:8

Navigate to [Dockerhub](https://hub.docker.com/), and you should see a new repository.

Now that the image has been uploaded, we can delete the images we have locally and free up space.

**Delete the java image**

docker rmi java

The output displays the different layers being deleted. The last three lines:

Deleted: sha256:dbf7b16cf5d32dfec3058391a92361a09745421deb2491545964f8ba99b37fc2

Deleted: sha256:4cbc0ad7007fe8c2dfcf2cdc82fdb04f35070f0e2a04d5fa35093977a3cc1693

Deleted: sha256:a2ae92ffcd29f7ededa0320f4a4fd709a723beae9a4e681696874932db7aee2c

Repeat for the renamed image

docker rmi <your\_docker\_username>/java:8

Now, when we run docker images we should see an empty table.

# Run a simple Pipeline job in Jenkins



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  1. Run a pipeline in Jenkins

### Objectives

### Run a Pipeline job in Jenkins

## Configuration Steps

1. Install and set up Jenkins, using this script:

#!/bin/bash

if type apt > /dev/null; then

pkg\_mgr=apt

if [ $(uname -v) == \*Debian\* ]; then

java="default-jre"

else

java="openjdk-11-jre"

fi

elif type yum /dev/null; then

pkg\_mgr=yum

java="java"

fi

echo "updating and installing dependencies"

sudo ${pkg\_mgr} update

sudo ${pkg\_mgr} install -y ${java} wget git > /dev/null

echo "configuring jenkins user"

sudo useradd -m -s /bin/bash jenkins

echo "downloading latest jenkins WAR"

sudo su - jenkins -c "curl -L https://updates.jenkins-ci.org/latest/jenkins.war --output jenkins.war"

echo "setting up jenkins service"

sudo tee /etc/systemd/system/jenkins.service << EOF > /dev/null

[Unit]

Description=Jenkins Server

[Service]

User=jenkins

WorkingDirectory=/home/jenkins

ExecStart=/usr/bin/java -jar /home/jenkins/jenkins.war

[Install]

WantedBy=multi-user.target

EOF

sudo systemctl daemon-reload

sudo systemctl enable jenkins

sudo systemctl restart jenkins

sudo su - jenkins << EOF

until [ -f .jenkins/secrets/initialAdminPassword ]; do

sleep 1

echo "waiting for initial admin password"

done

until [[ -n "\$(cat .jenkins/secrets/initialAdminPassword)" ]]; do

sleep 1

echo "waiting for initial admin password"

done

echo "initial admin password: \$(cat .jenkins/secrets/initialAdminPassword)"

EOF

1. Go to Jenkins and click on New Item, on the left hand side, and then choose Pipeline. Give the job a name, such as "jenkins-tutorial".
2. Create a new Repository on your Version Control Provider - in this tutorial, we will be using GitHub - and call it "jenkins-tutorial".
3. Create a file in this repository, called Jenkinsfile.
4. In the Jenkinsfile, enter the following:

pipeline{

agent any

stages{

stage('Make Directory'){

steps{

sh "mkdir ~/jenkins-tutorial-test"

}

}

stage('Make Files'){

steps{

sh "touch ~/jenkins-tutorial-test/file1 ~/jenkins-tutorial-test/file2"

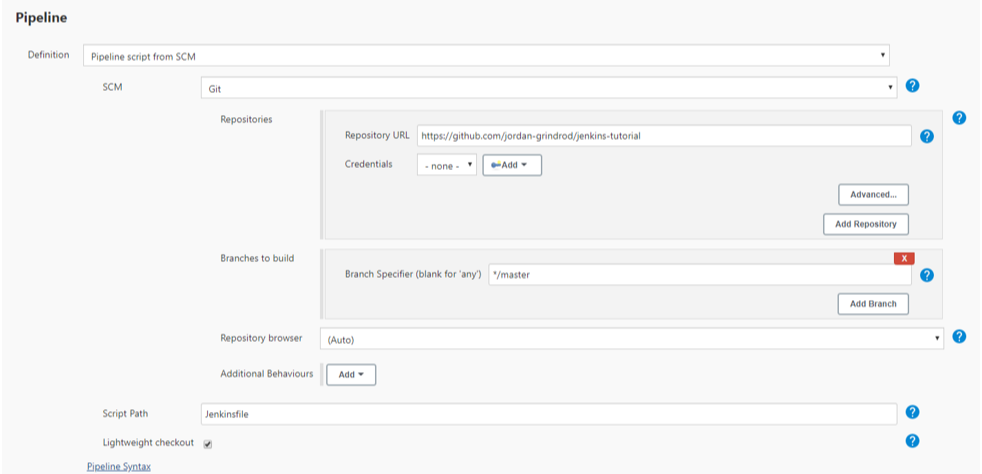
}

}

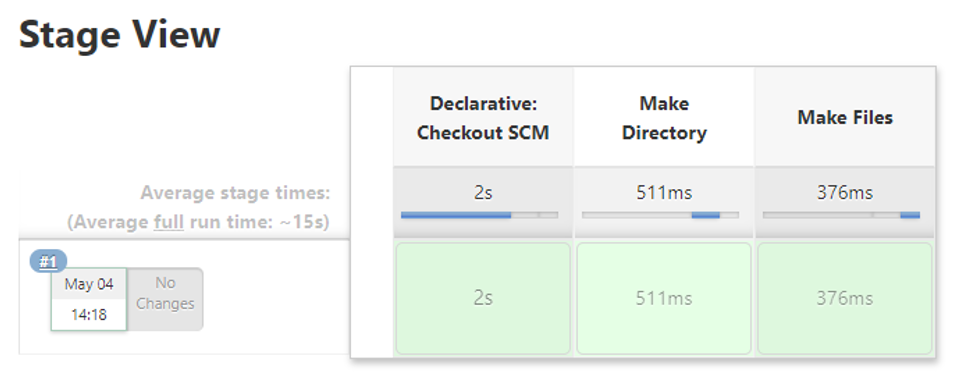
}

}

1. Go to your Jenkins instance, and change the Pipeline section to the below, making sure to use the URL for the repository you made in step 2 for Repository URL:



1. Click Save
2. Click Build Now, which can be found on the left hand side
3. The job should run, showing the status at each stage:



1. Go to your Jenkins machine and run the following command:

sudo su - jenkins

ls -al jenkins-tutorial-test

You should see file1 and file2, which have been created by the Jenkins Pipeline job!

# Run a CI Pipeline job in Jenkins



## Introduction



### About this lab

* We will learn how to run a CI Pipeline job in Jenkins

### Prerequisites

* + - 1. Jenkins installed and running.
      2. A GitHub account.
      3. Java and Maven installed on your Jenkins server.

### Objectives

* Upon completion of this task, you will be able to:
* Set up a CI pipeline in Jenkins that automates the build and test process for a sample Java project hosted on GitHub

## Configuration Steps

1. **Create a Sample Java Project**:
   * Create a simple Java project with a few classes and tests. You can initialize a new Maven project using the following command:

arduino

* + mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
  + Add some code and unit tests to the project. Ensure that the tests are written using a testing framework like JUnit.

1. **Push the Project to GitHub**:
   * Create a new repository on GitHub to host your sample Java project.
   * Push the project code to the GitHub repository.
2. **Set Up Jenkins Job**:
   * Log in to your Jenkins dashboard.
   * Click on "New Item" to create a new Jenkins job.
   * Enter a name for the job (e.g., "My Java CI Pipeline") and select "Freestyle project".
   * Click "OK" to create the job.
3. **Configure Source Code Management**:
   * In the job configuration page, scroll down to the "Source Code Management" section.
   * Choose "Git" and enter the URL of your GitHub repository.
   * Optionally, configure credentials if your repository requires authentication.
4. **Set Up Build Triggers**:
   * In the job configuration page, scroll down to the "Build Triggers" section.
   * Select "Poll SCM" and specify a schedule (e.g., \* \* \* \* \*) to poll for changes in your GitHub repository.
5. **Configure Build Steps**:
   * In the job configuration page, scroll down to the "Build" section.
   * Click on "Add build step" and choose "Invoke top-level Maven targets".
   * Enter the Maven goals to execute (e.g., clean test) and specify the path to the pom.xml file.
6. **Save Job Configuration**:
   * Save the job configuration by clicking "Apply" or "Save".
7. **Run the Pipeline**:
   * Trigger the job manually or wait for Jenkins to poll for changes in your GitHub repository.
   * Jenkins will clone the repository, build the project using Maven, and execute the tests.
8. **View Pipeline Results**:
   * Monitor the job's progress from the Jenkins dashboard.
   * If the build fails or any tests fail, Jenkins will provide detailed logs to help diagnose the issue.
9. **Optional: Add Post-Build Actions**:
   * Configure post-build actions to archive artifacts, generate reports, or trigger downstream jobs based on the build result.
10. **Iterate and Improve**:
    * Iterate on your Jenkins job configuration to add additional build steps, integrate with more tools (e.g., SonarQube for code quality analysis), and improve automation and error handling.

By completing this exercise, you'll have set up a basic CI pipeline in Jenkins that automates the build and test process for your sample Java project hosted on GitHub. You can further expand and customize this pipeline to suit your specific project requirements.

# Run a CT Pipeline job in Jenkins



## Introduction



### About this lab

* 1. In this lab we will learn how to set up a Jenkins pipeline to automate the execution of unit tests and integration tests for a sample Java project using Maven and JUnit.

### Prerequsites

* Jenkins installed and running.
* Your Java project hosted on a version control system (e.g., GitHub).
* Maven installed on your Jenkins server.
* JUnit testing framework for writing automated tests.

### Objectives

* Upon completion of this task, you will be able to:
  + Set up a Jenkins pipeline to automate the execution of unit tests and integration tests for a sample Java project using Maven and JUnit.

## Configuration Steps

1. **Parallel Testing**:
   * Objective: Set up a Jenkins pipeline to run tests in parallel across multiple stages.
   * Steps:
     + Define multiple test stages in your Jenkinsfile, each executing a subset of tests.
     + Use the parallel directive in Jenkins Pipeline to execute these stages concurrently.
     + Ensure proper synchronization and handling of test results across parallel stages.
   * Example: Divide your test suite into multiple categories (e.g., unit tests, API tests, UI tests) and execute each category of tests in parallel.
2. **Cross-Browser Testing**:
   * Objective: Extend your Jenkins pipeline to execute Selenium WebDriver tests on multiple browsers.
   * Steps:
     + Install and configure WebDriver for different browsers (e.g., ChromeDriver, GeckoDriver, SafariDriver) on your Jenkins server.
     + Modify your Selenium tests to specify the browser to be used dynamically.
     + Update your Jenkins pipeline script to execute tests on multiple browsers in parallel.
   * Example: Set up your Jenkins pipeline to run Selenium WebDriver tests on Chrome, Firefox, and Safari browsers concurrently.
3. **Parameterized Testing**:
   * Objective: Parameterize your tests to run with different configurations or data sets.
   * Steps:
     + Modify your test scripts to accept parameters or environment variables.
     + Configure your Jenkins pipeline to accept parameters (e.g., browser type, environment settings).
     + Dynamically pass these parameters to your test scripts during execution.
   * Example: Parameterize your Selenium WebDriver tests to run against different URLs, login credentials, or test data sets.
4. **Code Coverage Analysis**:
   * Objective: Integrate code coverage analysis into your Jenkins pipeline to measure the percentage of code covered by tests.
   * Steps:
     + Configure a code coverage tool (e.g., JaCoCo, Cobertura) in your Maven project.
     + Update your Maven build configuration to generate code coverage reports during the build process.
     + Integrate code coverage reporting into your Jenkins pipeline and display the results in the Jenkins dashboard.
   * Example: Set a code coverage threshold for your pipeline, and fail the build if the coverage falls below the specified threshold.
5. **Static Code Analysis**:
   * Objective: Incorporate static code analysis tools into your Jenkins pipeline to identify potential code quality issues.
   * Steps:
     + Choose a static code analysis tool (e.g., SonarQube, Checkstyle, FindBugs) compatible with your project's programming language.
     + Integrate the static code analysis tool into your Maven project configuration.
     + Configure your Jenkins pipeline to trigger static code analysis as part of the build process and display the analysis results.
   * Example: Set quality gates in your pipeline based on static code analysis results, and fail the build if code quality issues exceed predefined thresholds.

By completing these exercises, you'll gain hands-on experience in setting up more advanced Continuous Testing pipelines in Jenkins, covering a range of scenarios and techniques commonly used in software development projects.

# Docker Compose

## Introduction

### About this lab

* In this lab we will learn how to create a simple Docker Compose configuration which creates an NGINX container and scales the application by making multiple replicas of it.

### Prerequsites

* 1. Ensure Docker (and Docker Compose if using Linux) is installed.

### Objectives

* Upon completion of this task, you will be able to:
  + Create a simple Docker Compose configuration which creates an NGINX container and scales the application by making multiple replicas of it.

## Configuration Steps

**Docker Compose Configuration File**

Docker Compose is configured using YAML files, here we will configure one to create an NGINX container.  
The configuration shown below will create an NGINX container and publish port 80 to a random high port such as 35000.

Create a folder for this tutorial called docker-compose-nginx-tutorial and change to that directory:

mkdir docker-compose-nginx-tutorial && cd $\_

Now create a file called docker-compose.yaml and enter the following:

version: "3.8"

services:

nginx:

image: nginx:alpine

ports:

- target: 80

protocol: tcp

**Run Your Configuration**

You should now be able to run your first configuration using a docker-compose command:

docker-compose up -d

**View the Running Containers Using Compose**

The containers that are running can now be viewed:

docker-compose ps

The output should be something like this:

Name Command State Ports

-------------------------------------------------------------------

test\_nginx\_1 nginx -g daemon off; Up 0.0.0.0:32768->80/tcp

**Access the Application**

We can see that under the **Ports** column the high port that the container has been published on is listed.  
Try connect to the container in a browser or by using a curl command using that high port.

For this example we could use:

curl localhost:32768

Creating a response like this:

<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx!</title>

<style>

body {

width: 35em;

margin: 0 auto;

font-family: Tahoma, Verdana, Arial, sans-serif;

}

</style>

</head>

<body>

<h1>Welcome to nginx!</h1>

<p>If you see this page, the nginx web server is successfully installed and

working. Further configuration is required.</p>

<p>For online documentation and support please refer to

<a href="http://nginx.org/">nginx.org</a>.<br/>

Commercial support is available at

<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>

</body>

</html>

**Scale Your Application**

Compose will allow you to scale your application to an amount that you specify.  
You will be able to scale to different amounts depending on the machine that you are using and the resources that are being used by each of the containers.

In this instance we should be fine scaling to 3 NGINX containers:

docker-compose up -d --scale nginx=3

Now when you view the running containers there should be 3 instances of NGINX running:

docker-compose ps

Name Command State Ports

-------------------------------------------------------------------

test\_nginx\_1 nginx -g daemon off; Up 0.0.0.0:32768->80/tcp

test\_nginx\_2 nginx -g daemon off; Up 0.0.0.0:32769->80/tcp

test\_nginx\_3 nginx -g daemon off; Up 0.0.0.0:32770->80/tc

**Clean Up**

We can now stop and remove all the containers and images used by running this command:

docker-compose down --rmi all

# Docker Compose Lab 02

## Introduction

### About this lab

* In this lab we will learn how to use some basic commands with Docker Compose.

### Prerequsites

1 VM spun up with your cloud provider of choice

1. Running Ubuntu 18.04 LTS
2. Docker installed
3. Docker Compose installed
4. Allow incoming network traffic on port 5000

### Objectives

* Upon completion of this task, you will be able to:
  + Use some basic commands with Docker Compose .

## Configuration Steps

### Clone the Repository

docker-compose commands require you to be in the working directory of a docker-compose.yaml file. This is how Compose knows which configuration of containers it's meant to be managing. You should therefore only have one Compose file per directory.

We're going to use some Compose commands with an existing pair of containers. Clone down [this repository](https://gitlab.com/qacdevops/python-front-and-back) and change directory into it with the following commands:

git clone https://gitlab.com/qacdevops/python-front-and-back

cd python-front-and-back

This repository contains a simple pair of Python applications that communicate with one another. The exact functionality of this application is not important for this tutorial, but feel free to study the code.

### Running Containers

Run the following command to run the containers in our configuration:

docker-compose up

If the images don't already exist, docker-compose up will build them for you (assuming a build context is specified).

Once the images have been successfully built, you should see the following output:

Creating frontend ... done

Creating backend ... done

Attaching to backend, frontend

backend | \* Serving Flask app "app" (lazy loading)

backend | \* Environment: production

backend | WARNING: This is a development server. Do not use it in a production deployment.

backend | Use a production WSGI server instead.

backend | \* Debug mode: on

backend | \* Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)

backend | \* Restarting with stat

frontend | \* Serving Flask app "app" (lazy loading)

frontend | \* Environment: production

frontend | WARNING: This is a development server. Do not use it in a production deployment.

frontend | Use a production WSGI server instead.

frontend | \* Debug mode: on

frontend | \* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

frontend | \* Restarting with stat

backend | \* Debugger is active!

backend | \* Debugger PIN: 188-950-633

frontend | \* Debugger is active!

frontend | \* Debugger PIN: 149-867-548

Compose handily allows us to see the logs for multiple containers at once. This is particularly handy for troubleshooting applications with multiple services sending HTTP requests to one another.

Enter Ctrl+C to stop the containers. Let's run them in detached mode (i.e. in the background) so we can continue using our terminal session:

docker-compose up -d

### Logs

To view the logs of a Compose configuration, enter the following command:

docker-compose logs

The output should be identical to the logs you saw when you first spun the containers up in attached mode.

We can view the commands in realtime with the -f option. Run the following command:

docker-compose logs -f

While viewing these logs, navigate to the public IP address of your virtual machine on port 5000 (make sure your firewall rules have allowed incoming traffic on that port). Refresh the page a few times.

In realtime, you should see the following logs appear each time you refresh the page:

backend | 172.18.0.3 - - [09/Sep/2020 14:17:48] "GET /hostname HTTP/1.1" 200 -

backend | 172.18.0.3 - - [09/Sep/2020 14:17:48] "GET /random HTTP/1.1" 200 -

frontend | 87.75.101.48 - - [09/Sep/2020 14:17:48] "GET / HTTP/1.1" 200 -

The frontend container is receiving your HTTP requests on port 5000 and the backend is receiving two requests from the frontend. This is a great way to pinpoint where networking issues between your containers are occurring.

Enter Ctrl+C to stop viewing the logs.

### Building Images

Let's change the functionality of the frontend container. The repo has another version of the application on a separate branch called red-background. Change to that branch by entering:

git checkout red-background

Run the following command to rebuild our new image with our new functionality without bringing the current containers down:

docker-compose build

### Redeploying Containers

Compose can detect new versions of images and recreate running services with the new version without bringing the running versions down. Simply run the following command again:

docker-compose up -d

Refresh the page on your browser. The app should now display a red background!

### Cleaning Up

We can easily stop our containers with docker-compose down. If we want to bring the containers down and delete their associated images, we can run:

docker-compose down --rmi all

Do this now.

# Docker Swarm Initialization

## Introduction

### About this lab

* In this lab we will learn how to Initialize Docker Swarm.

### Prerequsites

* You will need two machines, ideally on the same network or subnet, which can connect to each other.
* Both the machines must have Docker installed on them.
* If you are using a cloud service, call one swarm-master and the other swarm-worker

### Objectives

* Upon completion of this task, you will be able to:
  + Initialize Docker Swarm and work with it.

## Configuration Steps

#### Initilising the Swarm

We need to create our first node in the Swarm which will serve as a Manager node.  
On the first machine you have, run the follow command:

docker swarm init

#### Adding a Worker Node

At this point the cluster is now setup and you may start creating services, however we will add a worker node first.

Upon initilising the Swarm, Docker would have likely prompted you with a command that allows you to join a worker node to the swarm.  
Once you have obtained the join command, you can simply run it on the second machine that you have.

The command will look similar to the one below, however the token and IP address will be different for yours, be sure to use the private address of the VM.

Log on to the **worker** node and run the join command similiar to below:

# docker swarm join --token [TOKEN] [IP\_ADDRESS]:[PORT]

docker swarm join --token SWMTKN-1-4y5lnvp7h7l7pt2qusvykiyk8p0hn0yywq3gk47ogau8pl64f9-15hstsq0lahr72cz6dppex0i6 swarm-master:2377

Also take a moment here to notice which port the workers are connecting from; 2377.  
This will be important to allow on locked down networks.

##### Retrieving the Join Command

If you can’t obtain the join command anymore, you can use the join-token command while on the **master** node to get it back:

docker swarm join-token worker

#### Create Your First Swarm Service

When managing Swarm services, ensure that you are first on the manager node.  
Here we will create a simple NGINX service and use the curl CLI tool to access it.

# docker service create --name [NAME] --publish [HOST\_PORT]:[CONTAINER\_PORT] [IMAGE]:[TAG]

docker service create --name nginx --publish 80:80 nginx:latest

Once you have created the service, it will be accessible on the private IP address of the machine, not the loopback interface (localhost).  
To access the NGINX service for this example, the curl command below can be used, the NGINX service will be available from both the worker and master node:

# curl http://[PRIVATE\_IP\_ADDRESS]

curl http://swarm-master

curl http://swarm-worker

#### Delete Your First Swarm Service

Let's remove the service that we created:

docker service rm nginx

# Docker Swarm Node Management

## Introduction

### About this lab

* In this lab we will learn how to manage Nodes using Docker Swarm

### Prerequsites

* 3 x Ubuntu VM's running version 18.04 LTS (Spun up with your choice of cloud provider).
* Docker installed on all VM's
* Port 80 open on your Manager VM

### Objectives

* Upon completion of this task, you will be able to:
  + Manage Nodes using Docker Swarm.

## Configuration Steps

### Create Cluster

Put together a cluster with 3 nodes, one manager and 2 workers.

1. On the machine you have chosen to be the Swarm Manager run the following command:

docker swarm init

This will initialise the Swarm Manager and Docker will output a Join Token. Note this down as you will need this command to add workers to your Swarm.

1. Log on to the Virtual Machine's that will be worker node's in the Swarm, run the join command that was generated earlier, the command should be similiar to the command below:

docker swarm join --token SWMTKN-1-14601cadnn15hf45gjqql23jta4vk4kxh8uy6zyn8vo0fxzu5z-31kwu4qd6up8ktez4kj62ppvr 192.168.61.1:2357

If you can’t obtain the join command anymore, run the join-token command below on the Swarm Manager VM:

docker swarm join-token worker

1. Log on to your second VM that will also be a worker node, run the same join-token command to join the Swarm as a worker.

Now you have 3 nodes in the Swarm! To check if the Swarm has been configured properly run the following command:

docker node ls

### Create a Service

Create a service that uses the bobcrutchley/python-http-server:latest image and is named python-http-server. Run the following command on your manager node to do this:

docker service create --name python-http-server bobcrutchley/python-http-server:latest

### Update the Service

Update the service so that there are 10 replicas and the port 9000 (inside the container) has been published to 80 (outside the container). Run the following Command to update the service:

docker service update --replicas 10 --publish-add 80:9000 python-http-server

The port numbers in the command above 80:9000 represent the Published Port which is 80 and the Container Port 9000.

### Access the Service

Use the curl CLI tool to view the info.json file served by the application, this file shows the name of the host that it is running on.

Run the following command:

curl http://[YOUR\_PRIVATE\_IP]/info.json

You can substitute the private IP address here with the private IP address of your manager node. Curl the file multiple times, what do you notice about the output?

### Remove the Worker Nodes

Update the amount of replicas to 2. Run the command Below on the manager VM to do this:

docker service update --replicas 2 python-http-server

Drain both of the worker nodes and remove them. Run the command below and replace the [NODE\_NAME] with your own.

docker node update --availability drain [NODE\_NAME]

You can get the node names by running docker node ls

**Clean up**

1. Remove Service (Run on Swarm Manager):

docker service rm python-http-server

1. On each worker node run the command below to shut it down:

docker swarm leave

1. Now remove the nodes using the command below on the Swarm Manager:

docker node rm [YOUR NODE NAME]

1. Shutdown and remove all VM resources on your cloud provider platform to avoid unexpected charges and fees.

# Load Balancing using Docker Swarm

## Introduction

### About this lab

* In this lab we will learn how to perform Load balancing using Docker Swarm.

### Prerequsites

For this task you will need:

* 3 x Ubuntu VM with Docker Installed with the following names:
  1. swarm-manager
  2. swarm-worker
  3. nginx
* Port 9000 on swarm-manager and swarm-worker should be open.
* Port 80 on the NGINX VM should also be open.

### Objectives

* Upon completion of this task, you will be able to:
  + Understand how Docker Swarm is used for Load Balancing.

## Configuration Steps

### Initialising the Swarm

SSH into your swarm-manager VM and enter the following command:

docker swarm init

This will now initialise the manager node and output a worker token similar to the token below:

docker swarm join --token SWMTKN-1-2tcnqv3awppiy848ho9cewqop1g3vvuxfk5mn8po27pjkpx304d-4l8vpniuvzzt22jd1k6q64t5p 192.168.12.2:2477

**Note this Command down as you will need to run it on the "worker node" VM to add a worker node to the swarm.**

### Join the Worker to the Swarm

On your swarm-worker VM, paste the join command that was generated by the manager node earlier.

If you forgot to note down the join token run this command on the Manager VM to generate another worker token:

docker swarm join-token worker

While on the manager node VM list all nodes in the swarm with the following command:

docker node ls

### Create a Service

On the Manager node VM run the following command to initialise the Python HTTP Server application:

docker service create --replicas 2 --publish 9000:9000 --name python-http-server bobcrutchley/python-http-server

This will create a HTTP server on port 9000. Navigate to your VM's public IP address on your browser on port 9000 to see this in action!

Run the command below to see if the service is running on the swarm:

docker service ls

### Creating our External Load Balancer

Now we will need to create an NGINX configuration file in our nginx VM to work as a reverse proxy/load balancer.

Create the file with the following command:

touch nginx.conf

Open the file in your text editor of choice and paste the following code into the file:

events{}

http {

upstream python-http-server {

server "[SWARM\_MANAGER\_PRIVATE\_IP]":9000;

server "[SWARM\_WORKER\_PRIVATE\_IP]":9000;

}

server {

location / {

proxy\_pass http://python-http-server;

}

}

}

Be sure to replace [SWARM\_MANAGER\_PRIVATE\_IP] and [SWARM\_WORKER\_PRIVATE\_IP] with the private IP addresses of your swarm-manager and swarm-worker VMs.

On the nginx VM, make sure your nginx.conf file is in your current working directory and run the following command to run an NGINX container with our coniguration:

docker run -d -p 80:80 --name nginx-swarm-ingress --mount type=bind,source=$(pwd)/nginx.conf,target=/etc/nginx/nginx.conf nginx:alpine

Navigate to your nginx VM's public IP address on port 80. You should see the web server hosted in your Swarm, confirming that our nginx VM is proxy passing to your Swarm cluster.

Refresh the page a few times. If the hostname is changing each time you refresh, NGINX is also successfully load balancing between the manager and worker node.

### Clean Up

Remove the service:

docker service rm python-http-server

In your swarm-worker VM, make the node leave the Swarm:

docker swarm leave

In your swarm-manager VM, stop the Swarm:

docker swarm leave --force

In your nginx VM, remove the NGINX container:

docker rm nginx-swarm-ingress.

# Kubernetes Orchestration

## Introduction

### About this lab

* In this lab we will learn how to set up a Kubernetes cluster with multiple worker nodes, deploying applications, scaling workloads, and managing resources using Kubernetes orchestration.

### Prerequsites

* Docker installed on each host.
* Kubernetes installed on a control plane node.
* Access to Kubernetes command-line tool (kubectl).

### Objectives

* Upon completion of this task, you will be able to:
  + Set up a Kubernetes cluster with multiple worker nodes and deploy a sample application across the cluster.

## Configuration Steps

1. **Provision Docker Hosts**:
   * Set up multiple Docker hosts (e.g., virtual machines) that will serve as worker nodes in your Kubernetes cluster.
   * Ensure that each host meets the minimum system requirements and has Docker installed and configured.
2. **Set Up Control Plane Node**:
   * Choose one of the Docker hosts to serve as the control plane node (master node) for your Kubernetes cluster.
   * Install Kubernetes components (kubelet, kube-proxy, container runtime) on the control plane node.
   * Initialize the Kubernetes cluster using kubeadm or another installation method.
3. **Join Worker Nodes**:
   * On each worker node, install Kubernetes components (kubelet, kube-proxy, container runtime).
   * Use kubeadm or kubectl to join the worker nodes to the Kubernetes cluster, connecting them to the control plane node.
4. **Configure Networking**:
   * Set up networking between nodes in the Kubernetes cluster to enable communication between pods and services.
   * Choose a networking solution compatible with Kubernetes (e.g., Calico, Flannel) and configure it to provide network connectivity and isolation.
5. **Deploy Sample Application**:
   * Package a sample application as a Docker container and push it to a container registry (e.g., Docker Hub).
   * Create Kubernetes manifests (YAML or JSON files) to define the deployment, service, and other resources required for the sample application.
   * Specify deployment configurations, such as the number of replicas, container image, resource requirements, and ports.
   * Apply the Kubernetes manifests using kubectl to deploy the sample application to the Kubernetes cluster.
6. **Scale Application**:
   * Experiment with scaling the sample application by adjusting the number of pod replicas in the deployment manifest.
   * Use kubectl commands to scale up or down the number of pod replicas and observe how Kubernetes automatically schedules and distributes pods across the cluster.
7. **Monitor Cluster**:
   * Monitor the health and performance of the Kubernetes cluster using built-in monitoring and logging tools or third-party solutions.
   * Use kubectl commands to retrieve cluster status, view pod logs, and inspect resource usage.
8. **Update Application**:
   * Perform a rolling update of the sample application to deploy a new version with updated features or bug fixes.
   * Update the container image tag in the deployment manifest to point to the new version.
   * Apply the updated manifest using kubectl to trigger a rolling update, allowing Kubernetes to gracefully replace existing pods with the new version.
9. **Clean Up**:
   * Once you've completed the exercise, clean up resources by deleting the sample application deployment and tearing down the Kubernetes cluster.

Use kubectl commands to delete Kubernetes resources (e.g., deployments, services, pods) and remove worker nodes from the cluster.

# Cloud Formation(Optional)

## Introduction

### About this lab

* Create a CloudFormation

### Objectives

* Upon completion of this task, you will be able to:
  + Create a CloudFormation template.

## Configuration Steps

1. **Create a YAML or JSON Template:** Decide whether you want to work with YAML or JSON for your CloudFormation template. YAML tends to be more human-readable.
2. **Define the Resources:** Define the resources you want to create. In this case, it's an S3 bucket with versioning enabled.
3. **Add Parameters (Optional):** If you want to make your template more flexible, you can add parameters. For example, you might want to allow users to specify the bucket name.
4. **Write the Template:** Here's an example YAML template to create an S3 bucket with versioning enabled:

yamlCopy code

AWSTemplateFormatVersion: '2010-09-09' Resources: MyS3Bucket: Type: AWS::S3::Bucket Properties: BucketName: my-bucket-name VersioningConfiguration: Status: Enabled

1. **Save the Template:** Save your template to a file with a **.yaml** or **.json** extension.
2. **Deploy the Stack:** Using the AWS Management Console, AWS CLI, or SDKs, deploy your CloudFormation stack using the template you created.

If you're using the AWS CLI, you can use the following command:

bashCopy code

aws cloudformation create-stack --stack-name my-s3-stack --template-body file://path/to/your/template.yaml

1. **Verify:** After the stack is created, verify that the S3 bucket was created with versioning enabled by checking the AWS Management Console or using AWS CLI commands like **aws s3api get-bucket-versioning**.
2. **Clean Up:** Once you're done experimenting, don't forget to delete the CloudFormation stack to avoid incurring any unnecessary costs.

That's it! This exercise should give you a good starting point for working with AWS CloudFormation. As you become more familiar with CloudFormation, you can explore more advanced features and templates.

# Terraform

## Introduction

### About this lab

* Upon completion of this task, you will be able to:
  + Have a basic understanding of how to use Terraform to provision infrastructure on Azure.

### Prerequsites

* AWS account with access keys configured.
* Terraform installed on your local machine.

### Objectives

* Upon completion of this task, you will be able to:
  + Provision a simple web server infrastructure on Azure using Terraform.

## Configuration Steps

 **Set Up Your Environment**:

* Install Terraform on your local machine. You can download it from the Terraform website or use a package manager.
* Install the Azure CLI on your local machine. You can download it from the Azure website or use a package manager. Log in to your Azure account using az login.

 **Write Terraform Configuration**:

* Create a new directory for your Terraform configuration files.
* Inside this directory, create a file named main.tf, which will contain your infrastructure configuration.
* Define the Azure provider and resources you want to provision. For example, let's create a simple Azure VM:

provider "azurerm" {

features {}

}

resource "azurerm\_resource\_group" "example" {

name = "terraform-example-rg"

location = "East US"

}

resource "azurerm\_virtual\_network" "example" {

name = "terraform-example-network"

address\_space = ["10.0.0.0/16"]

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

}

resource "azurerm\_subnet" "example" {

name = "subnet1"

resource\_group\_name = azurerm\_resource\_group.example.name

virtual\_network\_name = azurerm\_virtual\_network.example.name

address\_prefixes = ["10.0.1.0/24"]

}

resource "azurerm\_public\_ip" "example" {

name = "terraform-example-publicip"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

allocation\_method = "Dynamic"

}

resource "azurerm\_network\_interface" "example" {

name = "terraform-example-nic"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.example.id

private\_ip\_address\_allocation = "Dynamic"

public\_ip\_address\_id = azurerm\_public\_ip.example.id

}

}

resource "azurerm\_linux\_virtual\_machine" "example" {

name = "terraform-example-vm"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

size = "Standard\_DS1\_v2"

admin\_username = "adminuser"

network\_interface\_ids = [

azurerm\_network\_interface.example.id,

]

admin\_ssh\_key {

username = "adminuser"

public\_key = file("~/.ssh/id\_rsa.pub")

}

os\_disk {

caching = "ReadWrite"

storage\_account\_type = "Standard\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "UbuntuServer"

sku = "18.04-LTS"

version = "latest"

}

}

 **Initialize and Apply**:

* Open a terminal and navigate to your Terraform directory.
* Run terraform init to initialize Terraform and download any necessary plugins.
* Run terraform plan to see what Terraform will create. It will show you a preview of the changes.
* If the plan looks good, run terraform apply to create the resources.
* Terraform will prompt you to confirm the plan. Type yes and hit Enter to apply the changes.

 **Verify**:

* After Terraform finishes applying the changes, check the Azure portal to see if the VM has been provisioned.

 **Destroy**:

* Once you're done experimenting, you can destroy the resources created by Terraform to avoid incurring charges.
* Run terraform destroy in your Terraform directory.
* Terraform will prompt you to confirm the plan to destroy the resources. Type yes and hit Enter.

# Chef

## Introduction

### About this lab

* Upon completion of this task, you will be able to:
  + Have a basic understanding of how to use Chef to provision and configure a simple web server.

### Prerequsites

* Chef Workstation installed on your local machine.
* Access to a Chef Server (you can use a local Chef Server for testing purposes or a hosted one provided by Chef).
* A target node (either a virtual machine, container, or a cloud instance) where you'll deploy and configure the web server.

### Objectives

* Upon completion of this task, you will be able to:
  + Provision and configure a simple web server using Chef.

## Configuration Steps

1. **Setup Your Environment**:
   * Install Chef Workstation on your local machine.
   * Configure Knife to communicate with your Chef Server.
2. **Create a Cookbook**:
   * Use the **chef generate cookbook** command to create a new cookbook. Let's name it **webserver**.
   * Navigate into the **webserver** directory created by the command.
3. **Write a Recipe**:
   * Inside the **recipes** directory of your cookbook, create a file named **default.rb**.
   * Write a recipe to install and configure a basic web server (e.g., Nginx). Here's a simple example:

rubyCopy code

package 'nginx' do action :install end service 'nginx' do action [:enable, :start] end template '/var/www/html/index.html' do source 'index.html.erb' end

* + Create an **index.html.erb** file inside the **templates/default** directory and add some HTML content for your web page.

1. **Upload Cookbook to Chef Server**:
   * Use Knife to upload your cookbook to the Chef Server:

bashCopy code

knife cookbook upload webserver

1. **Bootstrap the Target Node**:
   * Bootstrap your target node to be managed by Chef:

bashCopy code

knife bootstrap <target\_node\_ip> --ssh-user <username> --sudo --node-name <node\_name> --run-list 'recipe[webserver]'

Replace **<target\_node\_ip>** with the IP address of your target node, **<username>** with your SSH username, and **<node\_name>** with a name for your node.

1. **Apply Configuration**:
   * SSH into your target node and run Chef Client to apply the configuration:

bashCopy code

sudo chef-client

This will install and configure the web server according to the recipe you defined.

1. **Verify**:
   * Open a web browser and navigate to the IP address of your target node. You should see the HTML content you defined in your recipe.
2. **Modify and Iterate**:
   * Modify the recipe or template files as needed, upload the updated cookbook, and re-run Chef Client on your target node to see the changes take effect.
3. **Cleanup**:
   * Once you're done experimenting, you can delete the node from the Chef Server and remove the web server configuration from the target node.

That's it! This exercise should give you a basic understanding of how to use Chef to provision and configure a simple web server. Feel free to explore more advanced features and configurations as you become more comfortable with Chef.

# Ansible

## Introduction

### About this lab

* In this lab we will learn how to work with Ansible and understand the basics of using Ansible playbooks to automate infrastructure configuration and management

### Prerequsites

* Control node with Ansible installed.
* Access to a target node (virtual machine, container, or server) where you'll configure the web server.
* SSH access to the target node from the control node.

### Objectives

* Upon completion of this task, you will be able to:
  1. Configure a simple web server on a target node using Ansible.

## Configuration Steps

1. **Set Up Your Environment**:
   * Install Ansible on your control node if you haven't already.
   * Ensure SSH access from the control node to the target node. You should be able to SSH into the target node without requiring a password.
2. **Create an Inventory File**:
   * Create an inventory file named **hosts** on your control node. Add the IP address or hostname of your target node to the file. For example:

csharpCopy code

[web\_servers] target\_node\_ip

1. **Write a Playbook**:
   * Create a playbook file named **web\_server.yml**. This playbook will define tasks to install and configure a simple web server (e.g., Nginx) on the target node.
   * Write tasks to install the Nginx package, start the Nginx service, and ensure that it's enabled to start on boot. Here's an example playbook:

yamlCopy code

--- - name: Configure Web Server hosts: web\_servers become: yes tasks: - name: Install Nginx apt: name: nginx state: present become: yes - name: Start Nginx Service service: name: nginx state: started enabled: yes

1. **Run the Playbook**:
   * Execute the playbook using the **ansible-playbook** command-line tool:

cssCopy code

ansible-playbook -i hosts web\_server.yml

1. **Verify**:
   * SSH into the target node and verify that Nginx is installed, running, and accessible.
   * You can use a web browser or tools like **curl** or **wget** to access the default Nginx welcome page.
2. **Cleanup**:
   * Once you're done experimenting, you can remove the Nginx package from the target node using Ansible.

**Notes**:

* Ensure that you have appropriate permissions and access rights to execute commands on the target node.
* Customize the playbook according to your requirements. You can add more tasks to configure additional settings or deploy applications.
* Explore Ansible's documentation and modules to learn more about its capabilities and features.

By completing this exercise, you'll gain hands-on experience with Ansible and understand the basics of using Ansible playbooks to automate infrastructure configuration and management.

# Creating an application using Jenkins, Docker and Puppet

## Introduction

### About this lab

* Upon completion of this task, you will be able to:
  + Gain hands-on experience in deploying an application using Jenkins, Docker and Puppet.

### Prerequsites

* Set up Jenkins to automate the CI/CD pipeline for the web application.
* Use Docker to containerize the Nginx web server and the web application.
* Use Puppet to manage the configuration of the Nginx server.
* Configure Jenkins to trigger the deployment process whenever changes are pushed to the Git repository

### Objectives

* Upon completion of this task, you will be able to:
  1. Automate the deployment of a basic web application using Jenkins, Docker, and Puppet. The web application consists of a static HTML page served by an Nginx web server.

## Configuration Steps

1. **Application Setup**:
   * Create a basic HTML file (index.html) containing some content (e.g., "Hello, World!").
   * Create a Dockerfile to build a Docker image for the Nginx web server, copying the index.html file into the server's document root.
2. **Jenkins Setup**:
   * Install Jenkins on a server or virtual machine.
   * Set up a Jenkins job to pull the application code from a Git repository.
   * Configure the Jenkins job to build the Docker image using the Dockerfile.
   * Add a post-build step to push the Docker image to a Docker registry (e.g., Docker Hub).
3. **Docker Setup**:
   * Install Docker on the Jenkins server or on a separate Docker host.
   * Write a Dockerfile to build the Nginx web server Docker image.
   * Include instructions to copy the index.html file into the Nginx document root (/usr/share/nginx/html).
4. **Puppet Setup**:
   * Set up a Puppet master server and Puppet agents on target servers where you want to deploy the Nginx web server.
   * Write a Puppet manifest to define the desired state of the Nginx server.
   * Use Puppet modules to manage Nginx configuration files, ensure the Nginx service is running, and enable necessary firewall rules.
5. **Integration**:
   * Configure the Jenkins job to trigger Puppet deployments after Docker image builds are successful.
   * Use Puppet to pull the Docker image from the registry and deploy it to the target servers.
   * Ensure Puppet applies the Nginx configuration to the servers, including any customizations or adjustments required for the web application.
6. **Testing**:
   * Set up automated tests within the Jenkins job to verify the functionality of the deployed web application.
   * Include tests to ensure the Nginx server is serving the index.html page correctly and responding to HTTP requests.
7. **Monitoring and Logging**:
   * Set up monitoring and logging for the Nginx server and the deployed web application.
   * Utilize tools like Prometheus for monitoring and the ELK stack (Elasticsearch, Logstash, Kibana) for centralized logging.
8. **Continuous Improvement**:
   * Monitor the CI/CD pipeline for efficiency and reliability.
   * Gather feedback from users and stakeholders to identify areas for improvement in the application and deployment process.
   * Iterate on the application code, Dockerfile, Puppet manifests, and Jenkins job configurations to address any issues and optimize performance.

**Conclusion**: By completing this exercise, you will have gained hands-on experience with integrating Jenkins, Docker, and Puppet to automate the deployment of a basic web application. This exercise demonstrates the power of CI/CD pipelines and infrastructure as code for automating software delivery and deployment processes.