

# Working with GitLab and Terraform

# TABLE OF CONTENTS

1. Git and GitLab Introduction: 3
2. GitLab Flow: 21
3. Branching: 88
4. Configuring Git: 105
5. Terraform Basics: 110
6. Infrastructure Deployment using Terraform: 143
7. GitLab Workflow Management - Merging: 155
8. GitLab Workflow Management - Resolving Merge Conflicts: 179
9. GitLab Workflow Management - Remote Repositories: 186
10. Reviewing the Commit History: 209
11. Continuous Integration / Continuous Delivery (CI/CD): 226
12. Advanced Infrastructure Management with Terraform: 291

# 1. Git and GitLab

## Introduction

# Introducing GitLab

In this lesson, we'll explore the following topics:

- An overview of version control
- The main features of GitLab
- Self-managed versus SaaS
- Free versus paid
- A brief history of GitLab

# Version control systems and Git

- Let's say you write code, or work on a course, or even just want to collect and update a set of text-based documents.
- You need some method of keeping track of changes, of being able to revert mistakes in the work, or branch in new directions; and you'll probably want some way of remotely backing up your work in case of fire, theft, or acts of a misbehaving computer.

# GitLab and Git

- GitLab is built on top of git so that users who are contributing work (editing code, writing lessons, and so on) to a project will have a copy of the project downloaded/checked out/cloned on their local computer.
- It provides a web interface for handling many of git's more advanced workflows, and recommends a workflow for interacting with git for the best in productivity, efficiency, and ease of use.

# Features

- There is the classic file browser that lets you explore the files in your repository:

📁 shared	Add GitLab Pages	1 year ago
📁 spec	Merge remote-tracking branch 'dev/master'	10 hours ago
📁 symbol	Resolve "Better SVG Usage in the Frontend"	10 months ago
📁 tmp	Move Prometheus presentation logic to Prometh...	1 year ago
📁 vendor	Specify Jupyter Image to use with JupyterHub In...	6 days ago
📄 .babelrc	only apply rewire plugin when running karma tests	3 months ago
📄 .codeclimate.yml	Removed API endpoint and specs	1 month ago
📄 .csscomb.json	Remove SCSS rules for short hex chars.	1 year ago
📄 .eslintignore	update eslintignore for node scripts	4 months ago
📄 .eslintrc.yml	Enable "prefer-destructuring" in JS files	1 month ago
📄 .flayignore	Backport from EE !5954	3 weeks ago
📄 .foreman	complete hooks for post receive	6 years ago
📄 .gitattributes	Start to use Danger for automating MR reviews	2 weeks ago
📄 .gitignore	Exclude Geo DB Yaml on CE too	1 month ago

# Features

- There's also a branch viewer, which lets you see variations of your work under active development, as well as branches that are considered stale and no longer developed:

The screenshot shows the 'Branches' page in the GitLab Community Edition. The top navigation bar includes links to GitLab.org, GitLab Community Edition, Repository, and Branches. Below the navigation is a filter bar labeled 'Filter by branch name'. A tabs menu at the top left allows switching between Overview, Active, Stale, and All branches. The main content area is titled 'Active branches' and lists five branches:

- 48773-gitlab-project-import-should-use-object-storage**: Last updated 5 minutes ago by commit [a8c87544](#). It has 56 merges and 8 pull requests. Buttons for 'Compare' and a dropdown menu are available.
- bugs-and-regressions-process**: Last updated 13 minutes ago by commit [46a54123](#). It has 2 merges and 10 pull requests. Buttons for 'Compare' and a dropdown menu are available.
- artifact-format-v2-with-parser**: Last updated 40 minutes ago by commit [fda5e9bd](#). It has 2 merges and 203 pull requests. Buttons for 'Compare' and a dropdown menu are available.
- master**: Labeled as default and protected. Last updated 51 minutes ago by merge commit [835bacc2](#). It has 0 merges and 0 pull requests. Buttons for 'Compare' and a dropdown menu are available.
- 48419-charts-with-long-label-appear-oversized**: Last updated 51 minutes ago by merge commit [835bacc2](#). It has 0 merges and 0 pull requests. Buttons for 'Compare' and a dropdown menu are available.

At the bottom of the list is a link 'Show more active branches'.

# Features

- Alongside this is a tag viewer that lets you explore specific releases of your work:

Tags give the ability to mark specific points in history as being important

---

 **v10.8.7** protected Version v10.8.7

-o [eb600b0b](#) · Update VERSION to 10.8.7 · 1 day ago

---

 **v11.0.5** protected Version v11.0.5

-o [a4583c3b](#) · Update VERSION to 11.0.5 · 1 day ago

---

 **v11.1.2** protected Version v11.1.2

-o [35936b0b](#) · Update VERSION to 11.1.2 · 1 day ago

---

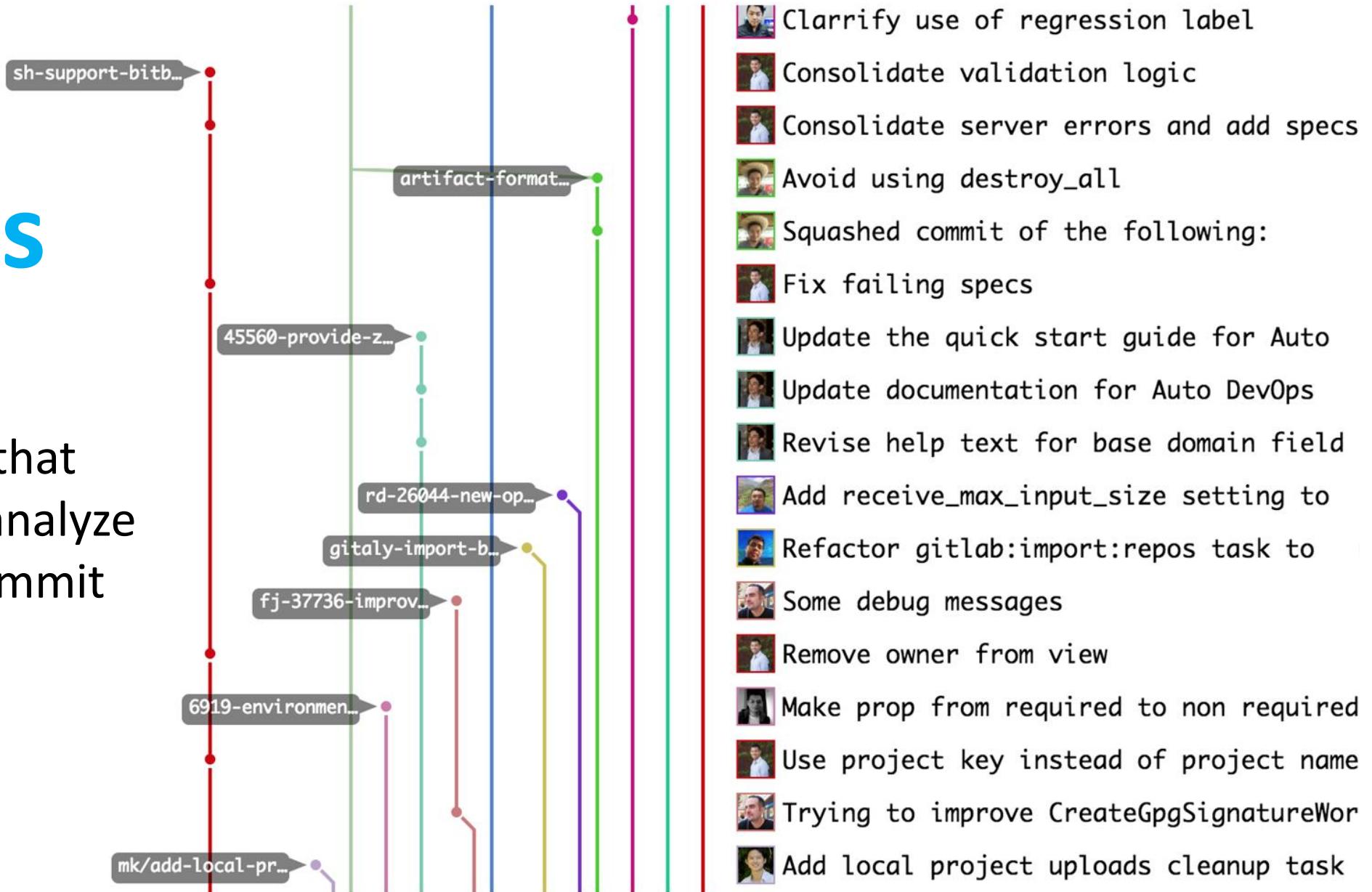
 **v11.1.1** protected Version v11.1.1

-o [94b93230](#) · Update VERSION to 11.1.1 · 3 days ago

---

# Features

- There are tools that can be used to analyze and view the commit graph:

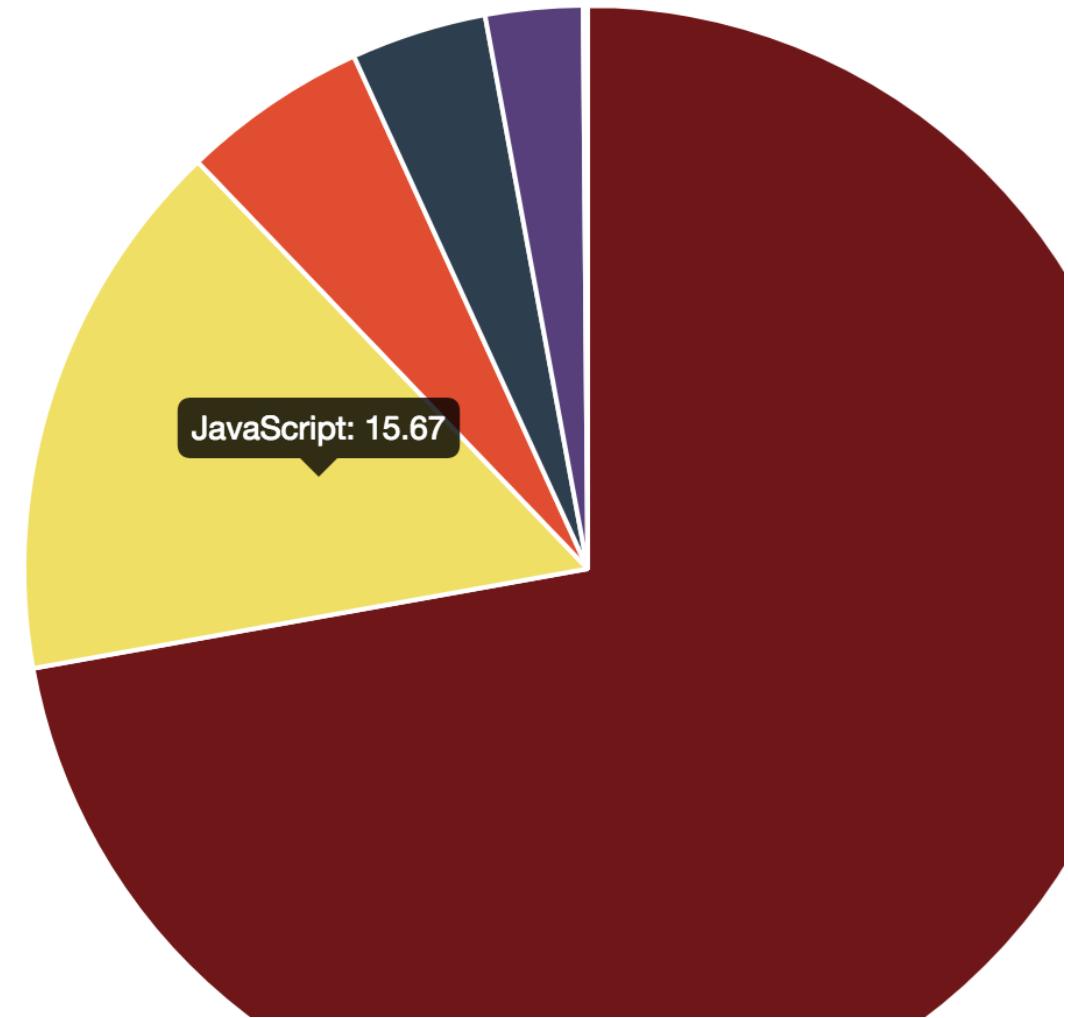


# Features

- Among these tools are charting tools, which are used to get a better understanding of the composition of the repository:

Programming languages used in this repository

● Ruby	72.17 %
● JavaScript	15.67 %
● HTML	5.35 %
● Vue	3.89 %
● CSS	2.79 %
● Shell	0.13 %
● Clojure	0.0 %



## Commit statistics for master Jun 01 - Jul 26

- Total: **2000 commits**
- Average per day: **35 commits**
- Authors: **204**

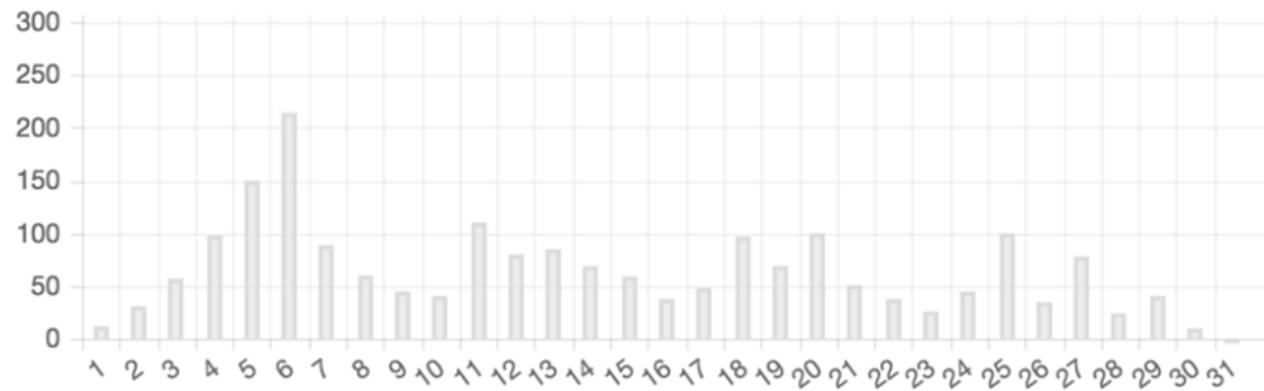
# Features

- Alongside this is a breakdown of the frequencies of commits and activity:

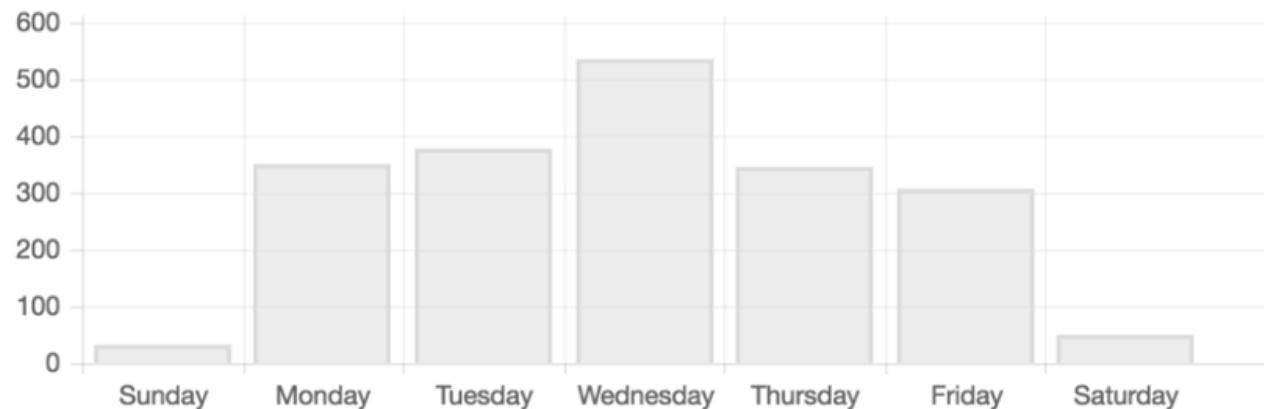
master

gitlab-ce

Commits per day of month



Commits per weekday



# Features

- GitLab also provides a web interface where you can make changes to code and commit it straight from the browser:



The screenshot shows a GitLab interface for editing files. On the left, there's a sidebar with 'Edit' and a '+' button, followed by a 'Files' section containing a 'Docs' folder. Inside 'Docs', several RTF files are listed: 11.rtf, 11\_notes.rtf, 11\_synopsis.txt, 12.rtf (which is selected), 29.rtf, 30.rtf, 35.rtf, 38.rtf, 39.png, 4.rtf, 42.rtf, 45.rtf, 46.rtf, 47.rtf, 48.rtf, 49.rtf, 50.rtf, and 51.rtf. Below the file list is a 'Commit...' button. At the bottom, it says '0 unstaged and 0 staged changes'. The main area displays the content of the selected file, 12.rtf, which contains RTF code. The code includes various document structures like lists, tables, and sections, with some lines numbered 1 through 28.

```
1 {\rtf1\ansi\ansicpg1252\cocoartf1347\cocoasubrtf570
2 {\fonttbl\f0\fnil\fcharset0 Baskerville;\f1\fnil\fcharset0 CenturySchoolbook;\f2\fswiss\fcharset0 Helvetica;
3 \f3\fnil\fcharset0 Cochin;}
4 {\colortbl;\red255\green255\blue255;}
5 {\*\listtable{\list\listtemplateid1\listhybrid
6 {\listlevel\levelnfc23\levelfcn23\leveljc0\leveljcn0\levelfollow0\levelstartat1\levelspace360\levelindent0{\*\levelmarker \
7 {disc}}{\leveltext\leveltemplateid1\'01\uc0\u8226 ;}{\levelnumbers;}\fi-360\li720\lin720 }{\listname ;}\listid1}
8 {\list\listtemplateid2\listhybrid
9 {\listlevel\levelnfc23\levelfcn23\leveljc0\leveljcn0\levelfollow0\levelstartat1\levelspace360\levelindent0{\*\levelmarker \
10 {disc}}{\leveltext\leveltemplateid101\'01\uc0\u8226 ;}{\levelnumbers;}\fi-360\li720\lin720 }{\listname ;}\listid2}
11 {\list\listtemplateid3\listhybrid
12 {\listlevel\levelnfc23\levelfcn23\leveljc0\leveljcn0\levelfollow0\levelstartat1\levelspace360\levelindent0{\*\levelmarker \
13 {disc}}{\leveltext\leveltemplateid201\'01\uc0\u8226 ;}{\levelnumbers;}\fi-360\li720\lin720 }{\listname ;}\listid3}
14 {\*\listoverridetable{\listoverride\listid1\listoverridecount0\ls1}{\listoverride\listid2\listoverridecount0\ls2}
15 {\listoverride\listid3\listoverridecount0\ls3}}
16 \pard\tx560\tx1120\tx1680\tx2240\tx2800\tx3360\tx3920\tx4480\tx5040\tx5600\tx6160\tx6720\sl288\slmult1\sb160\pardirnatural\qj
17 \f0\fs42 \cf0 N
18 \fs31\fsmilli15750 ON
19 \fs42 -F
20 \fs31\fsmilli15750 ICTION
21 \fs42
22 \fs31\fsmilli15750 WITH
23 \fs42 S
24 \fs31\fsmilli15750 UB
25 \fs42 -H
26 \fs31\fsmilli15750 EADS
27 \f1\fs28 \
28 \pard\tx560\tx1120\tx1680\tx2240\tx2800\tx3360\tx3920\tx4480\tx5040\tx5600\tx6160\tx6720\sl360\slmult1\pardirnatural\qj
29 \f2\fs24 \cf0 \
30 
31 \fs36 About This Template
32 \fs24 \
33 When compiled (File > Compile), this project will generate a document in standard manuscript format with sub-headings inside the
34 chapters.\
```

# Features

- With things like epics, milestones, and cycle analytics, GitLab can help measure the effectiveness of your development process.

▼ Backlog 5552

- Remove trigger.owner #36794  
CI/CD breaking change technical debt
- GitLab pages support for subgroups #30548  
CI/CD customer devops:release  
feature proposal pages subgroups
- Support new issue creation by email without subaddressing #29951  
P3 Plan S3 customer customer+ emails feature proposal issues reply by email
- Collect usage data on issue boards #25288  
Plan boards
- Filter Merge Requests by target branch #33831 4  
Community Contribution Create P3 S3 UX ready auto updated customer devops:create feature proposal merge requests potential proposal repository

regression 92

- Reinstate option to force lowercase project/repository names on creation #1989  
feature proposal regression
- PostgreSQL DB Migration Error When Upgrading to 9.2.0 #32721  
CI/CD auto updated awaiting feedback customer database regression
- Error 500 loading merge requests due to `nil` parameter in Repository#merge\_base\_commit #32812  
Next Patch Release Platform backend bug regression repository reproduced on GitLab.com
- Rspec feature tests have useless javascript stack traces #34012  
backstage blocked regression test webpack
- New issue from failed job not always workin #35874

direction 151

- JUnit XML Test Summary In MR widget #45318  
CI/CD Deliverable In dev Product Vision 2018 UX ready backend customer devops:verify direction feature proposal frontend merge requests
- Pipeline view of environments #28698 9  
CI/CD Deliverable deploy devops:release direction feature proposal frontend meta missed-deliverable
- Method to update managed apps in Kubernetes #42686  
CI/CD UX backend devops:configure direction feature proposal frontend kubernetes
- Track label add/remove events for issues/mrs/epics #47993 5  
Deliverable In dev Plan backend direction epics feature proposal issues labels merge requests

# Features

- It also includes the necessary tools for code review prior to merging branches to ensure that all work is up to scratch:

Showing 2 changed files ▾ with 6 additions and 0 deletions

Hide whitespace changes Inline Side-by-side

▼  app/assets/stylesheets/pages/builds.scss 

 Edit View file @ cff585df

...	...	@@ -270,6 +270,7 @@
270	270	
271	271	.block {
272	272	width: 100%;
	273	+ word-break: break-word;
273	274	
274	275	&:last-child {
275	276	border-bottom: 1px solid \$border-gray-normal;
...	...	

# Features

- Automated testing tools and pipelines are also included to help make sure that code is working perfectly before it's merged back in or released:

passed Pipeline #26392237 triggered 1 hour ago by Winnie Hellmann

### Enable no-console ESLint rule for tests

⌚ 93 jobs from [winh-lint-console-tests](#) in 62 minutes 18 seconds (queued for 9 seconds)

- 7a870e40 ⋮ ⌂

❗ SAST detected [59 vulnerabilities](#)

❗ Dependency scanning detected [133 vulnerabilities](#)

Pipeline Jobs 93 Security report 192

Build

Prepare

Test

Post-test

⌚ package-and-qa

⌚ review-docs-d...

✓ compile-assets

✓ retrieve-tests-...

✓ setup-test-env

✓ codequality

✓ danger-review

✓ db:check-sche...

✓ db:migrate:res...

✓ db:migrate:res...

✓ coverage

✓ flaky-examples...

✓ lint:javascript:r...

# **Self-managed versus Software as a Service (SaaS)**

- GitLab can be used in one of two ways: either self-managed, where you host your own instance of GitLab Community Edition/Enterprise Edition
- Using the online platform GitLab.com, which comes as a paid or free Software as a Service (SaaS) model.

# Free versus paid

- Lastly, there are multiple tiers of GitLab for both the self-managed and SaaS versions.
- Please note that both versions can be used for free and provide all of the main features that you'd expect (git hosting, code review, issue management, testing, and deployment).
- The added tiers provide extra features that are available at different levels of pricing on a per-user, per-month basis.

# Summary

- So far, we've discovered what version control is: a method of tracking revisions of work, of creating alternate test branches, and working collaboratively.
- We know that git is a form of version control system that specializes in working in a distributed network and that GitLab is a platform that is based on git but with a lot of powerful features.

## 2. GitLab Flow

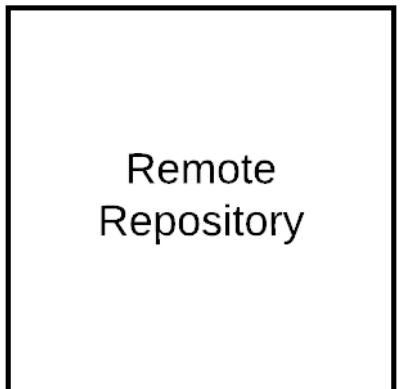
# GitLab Flow

In this lesson, we'll explore the following topics:

- How to use Git
- The GitFlow for branching and merging
- The GitLab flow (recommended by GitLab)
- The GitLab flow in comparison to the GitHub flow

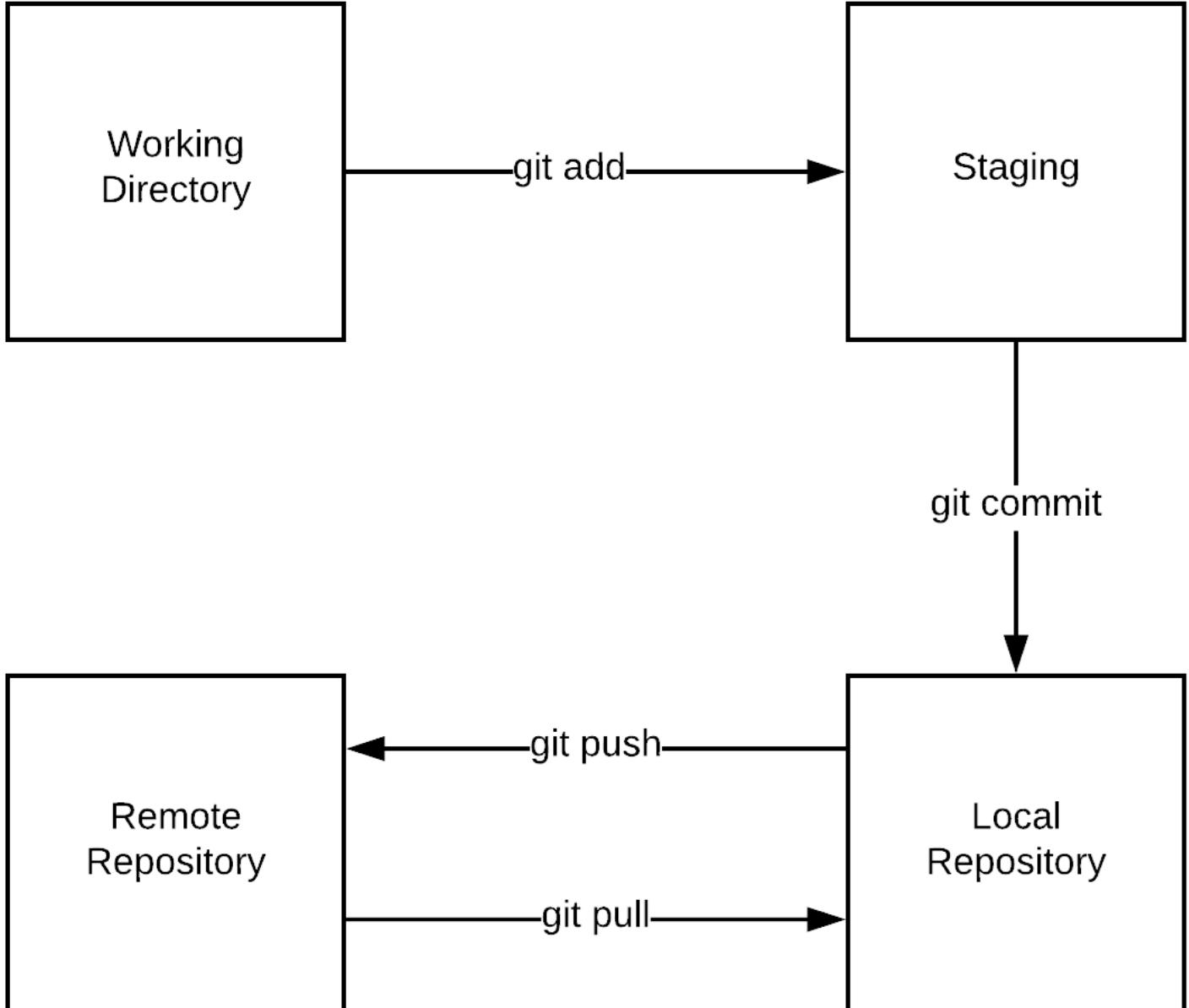
# Using Git

- One thing to recognize is the life cycle of Git work and the stages it can go through.
- There are four separate parts: your working directory, the index or staging area, committing to the local repository, and pushing to remote repositories:



# Using Git

- Overall, it seems like a pretty complicated workflow, which is why I've created this handy graph to illustrate the basic operations that you can perform and how they relate to these stages:

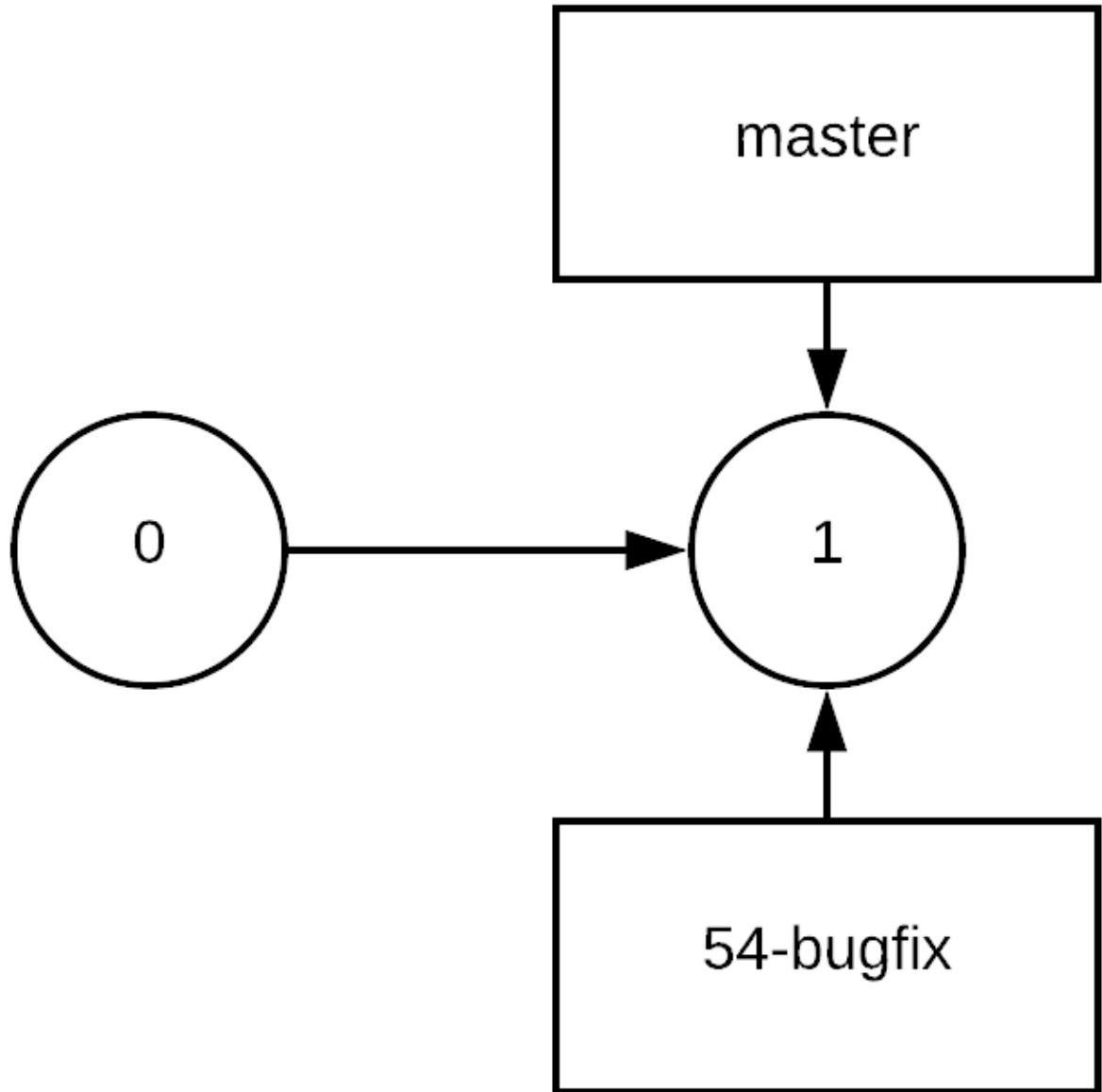


# Git commands

- I've introduced a few Git commands while discussing how Gitworks, but haven't actually explained properly how to use them, so let's do a very brief introduction to the basic usage of Git.
- The first command you'll need when starting a new project is `git init`. This initializes a new Git repository, and you'll notice that it creates a `.git` folder so that you can start your project (this is hidden by default on Unix-based and Windows operating systems).

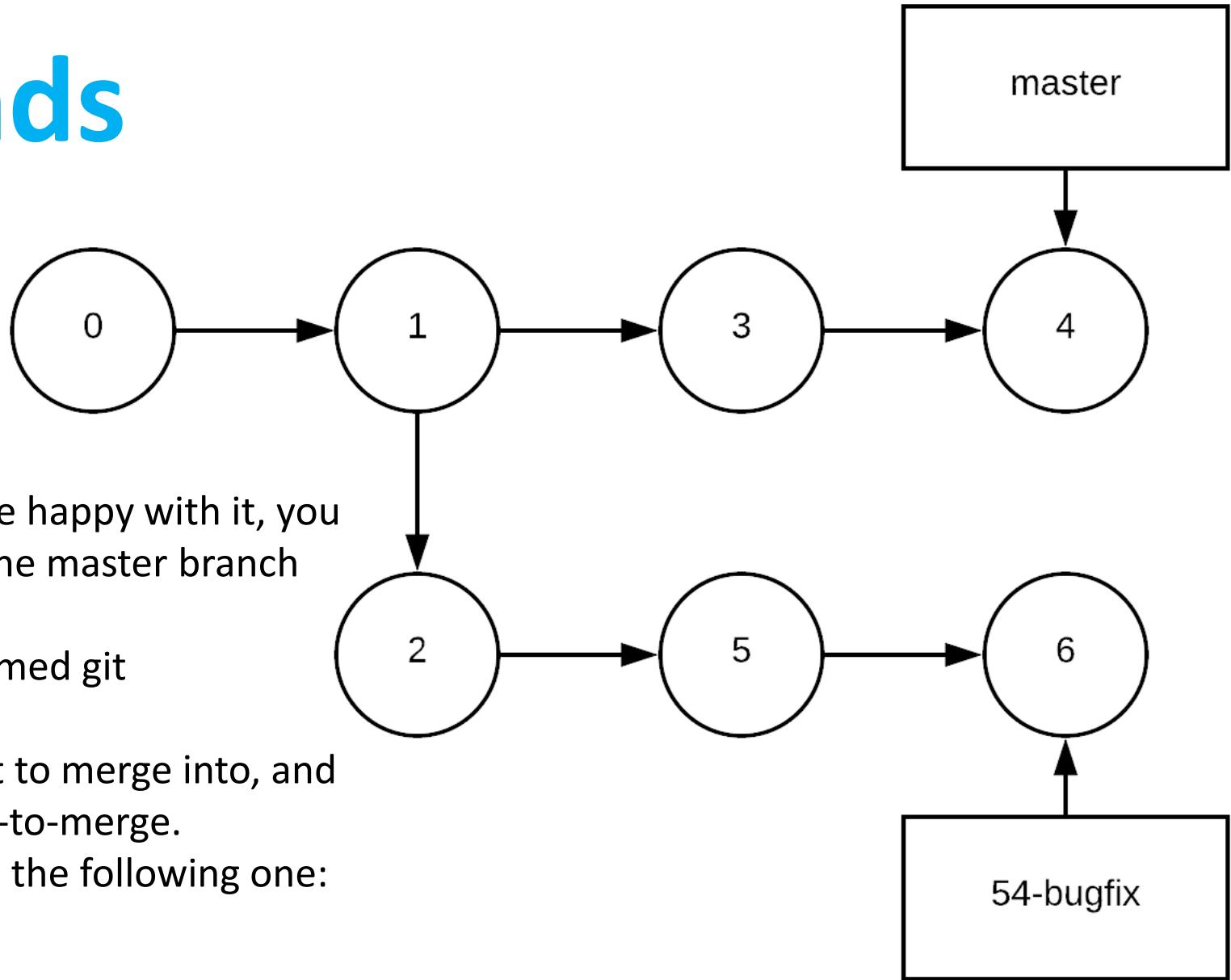
# Git commands

- Behind the scenes, a Git branch is a reference to a particular commit, and adding new commits moves the branch pointer forward, but forks off from the main (master) branch.
- In this way, they are lightweight and still connected to the entire history of the local repository:



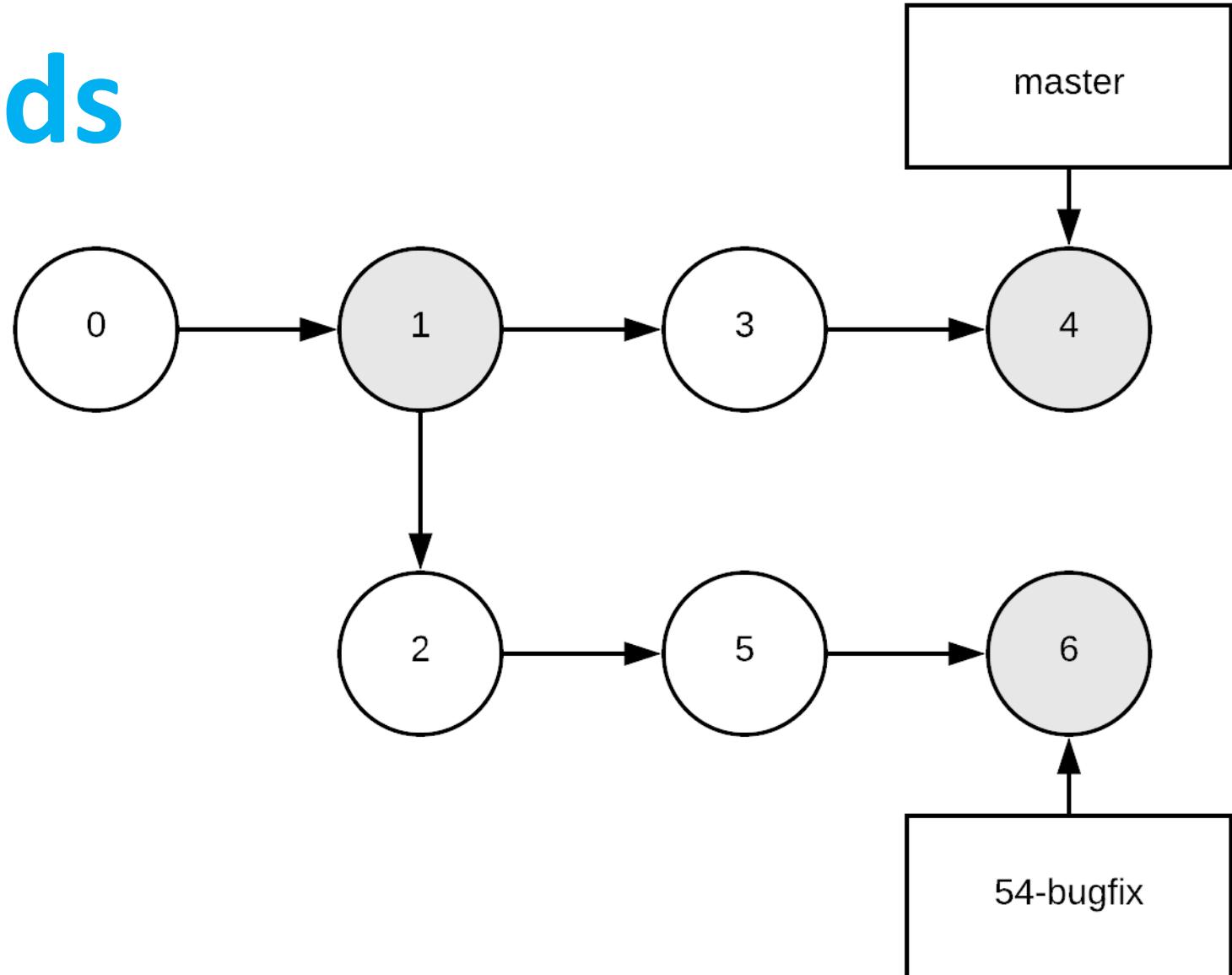
# Git commands

- If you've done some work and are happy with it, you may want to merge it back into the master branch of the code.
- You can do this with the aptly named git merge command.
- You swap to the branch you want to merge into, and then run git merge branch-name-to-merge.
- Let's say you have a situation like the following one:



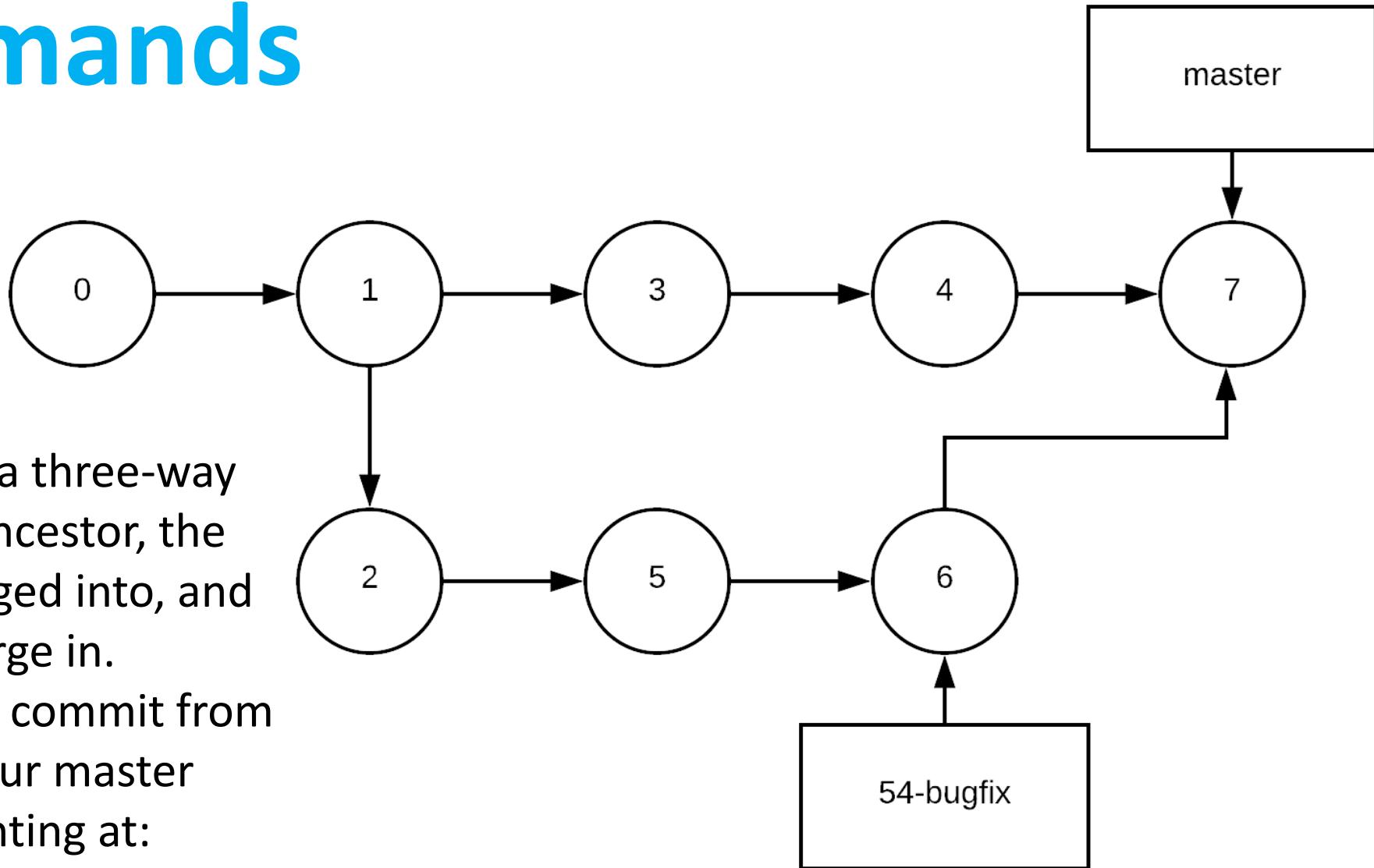
# Git commands

- When you're on the master branch and run git merge 54-bugfix, behind the scenes, Git grabs those two commits and then finds their closest common ancestor:



# Git commands

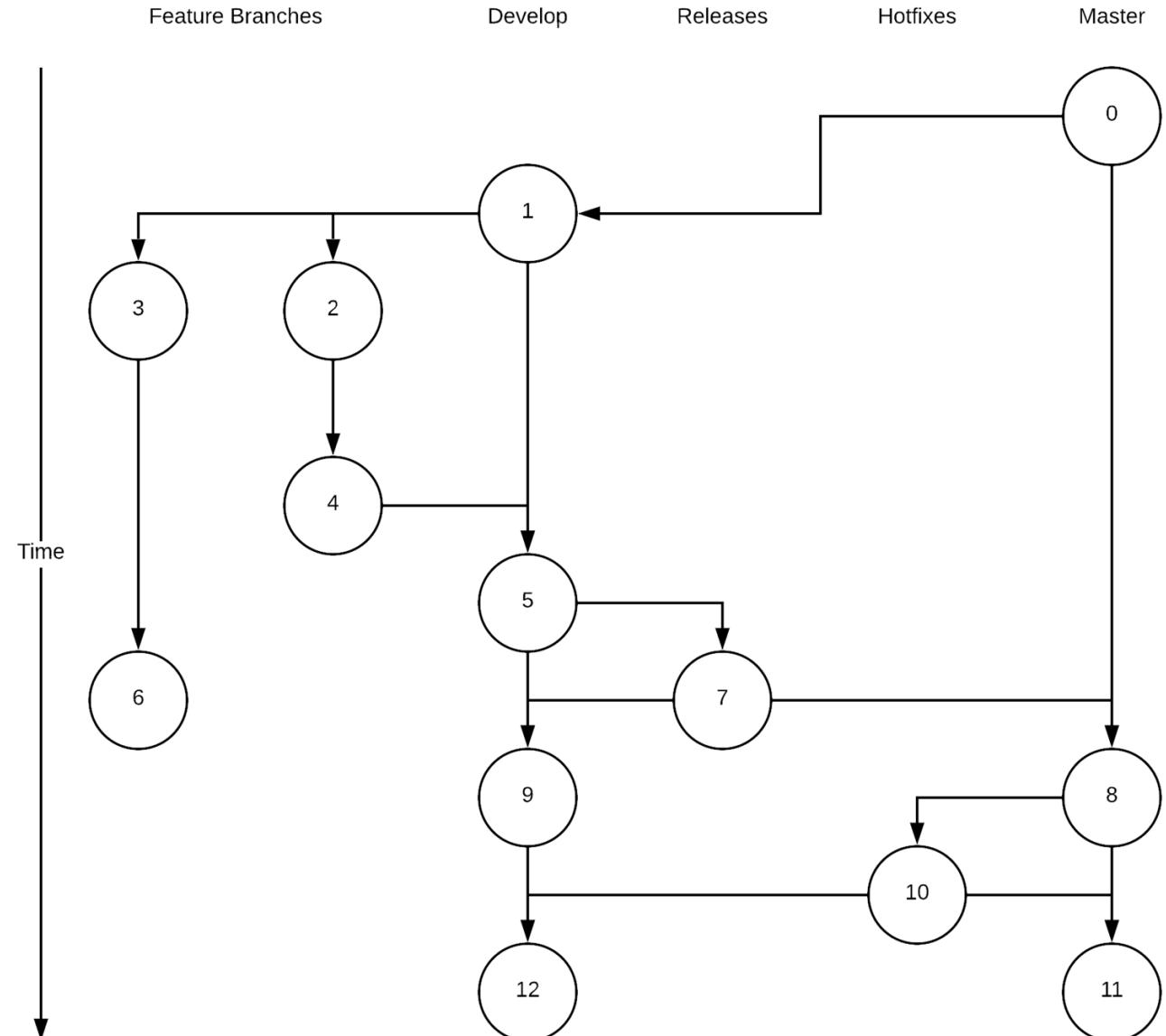
- Git then attempts a three-way merge using the ancestor, the commit to be merged into, and the commit to merge in.
- This creates a new commit from all three, which your master branch is now pointing at:



# GitFlow

- Git flow is a semi-standardized workflow for dealing with projects in Git.
- It can be daunting, but the rules involve creating a stringent practice for when branches should be committed to or merged in order to prevent deploying buggy code or releases.

A diagram of it is as follows:



# Master

The master branch is the original branch where all code is branched from.

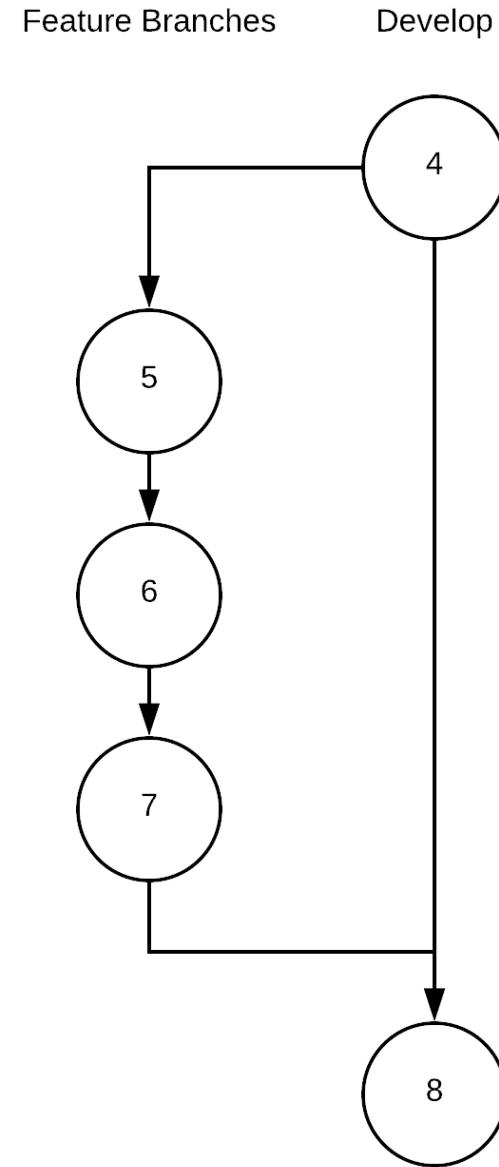
- However, it is never committed to directly; it is only ever merged into.
- The master branch is then used as the deployed branch, or the released branch if you have a product that is shipped. You can tag each merge with a version number to help identify these releases or deployed versions.

# Develop

- The develop branch is the other continuously existing branch in the Git flow model, along with the master branch.
- All other branches are considered temporary and can be deleted after work is finished on them and they've been merged.

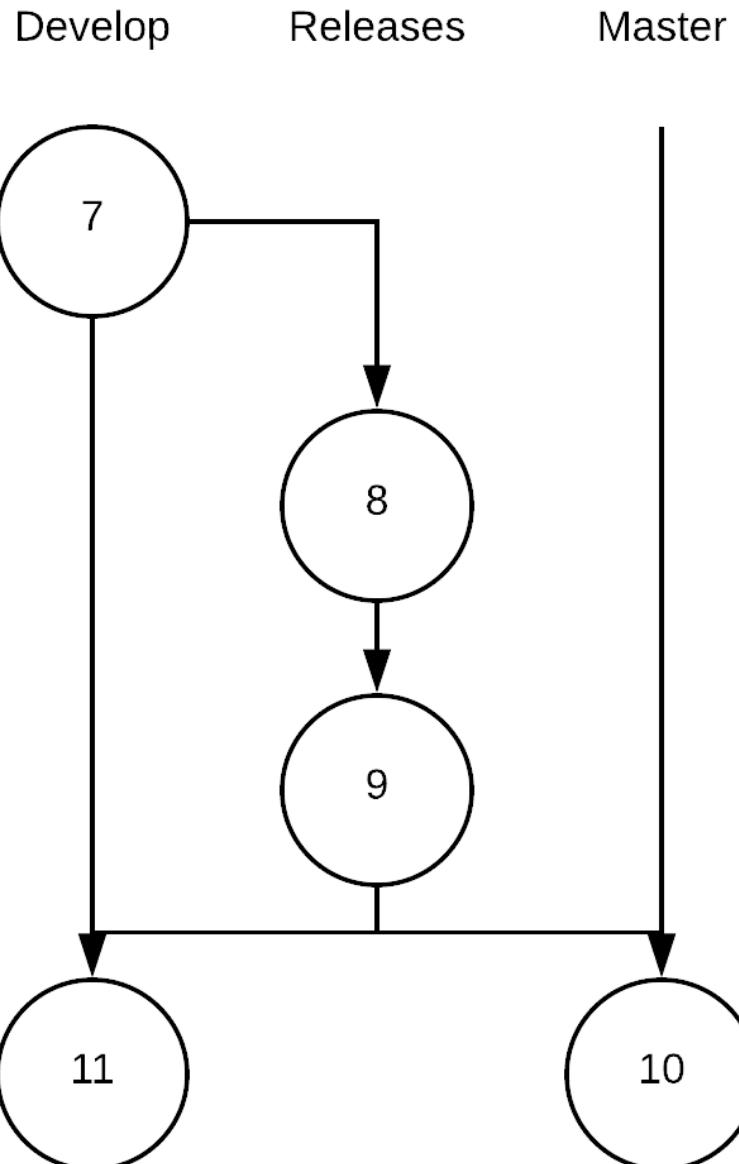
# Features

- Unlike the develop or master branches, feature branches only need to last until the feature is complete and merged into develop, after which you can delete the branch since it should no longer be needed:



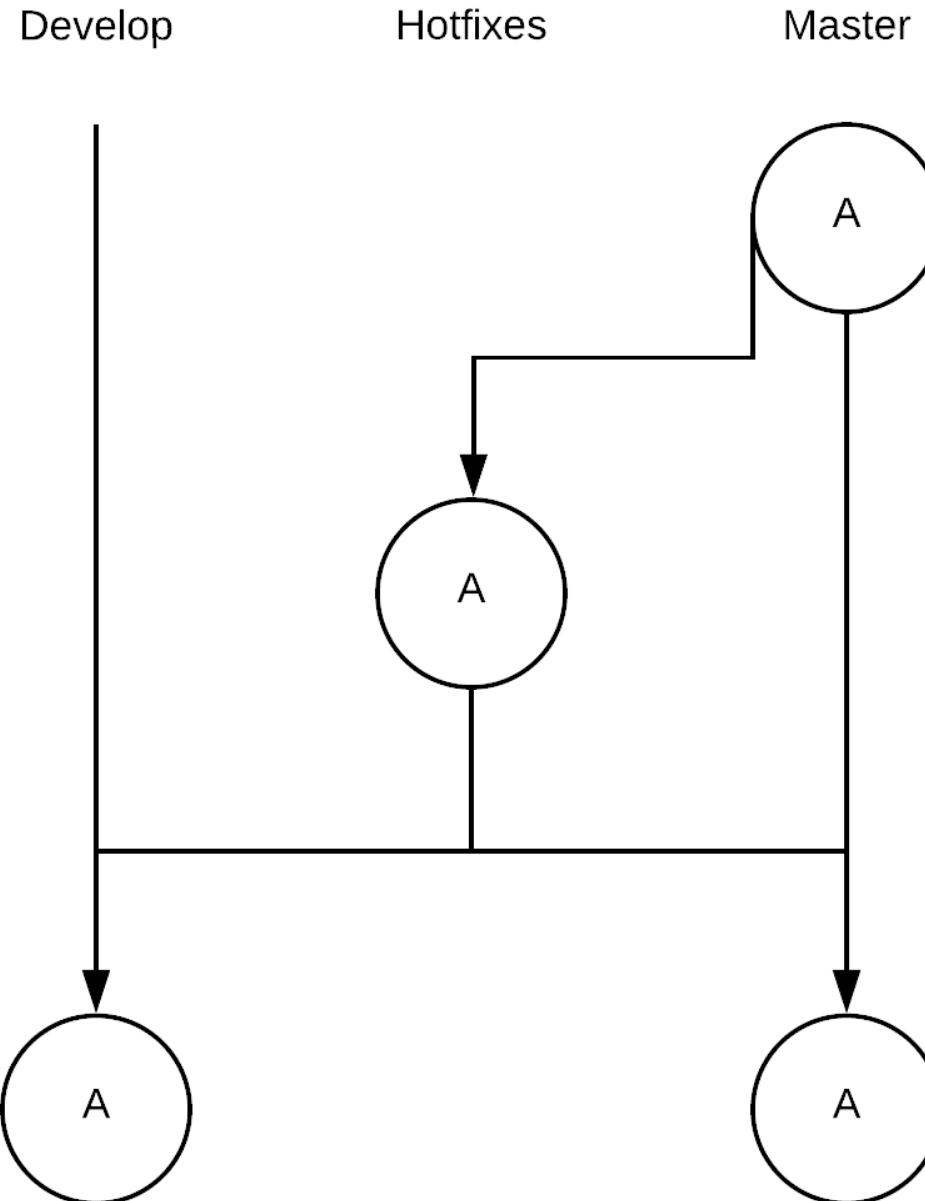
# Releases

- If everything is ready, the branch can be merged directly into the master and tagged appropriately as a new release (with either a major or minor version number, depending on your scheme).
- Once this is done, the branch should be remerged into develop once more to ensure that all of the changes are back into the main development stream.
- You can then safely delete this branch since it is no longer required:



# Hotfixes

- After you've merged into master, you also need to merge this into develop so that develop has the latest patches and fixes and you don't conflict in future releases.
- The hotfix can then be safely deleted as it is no longer required:

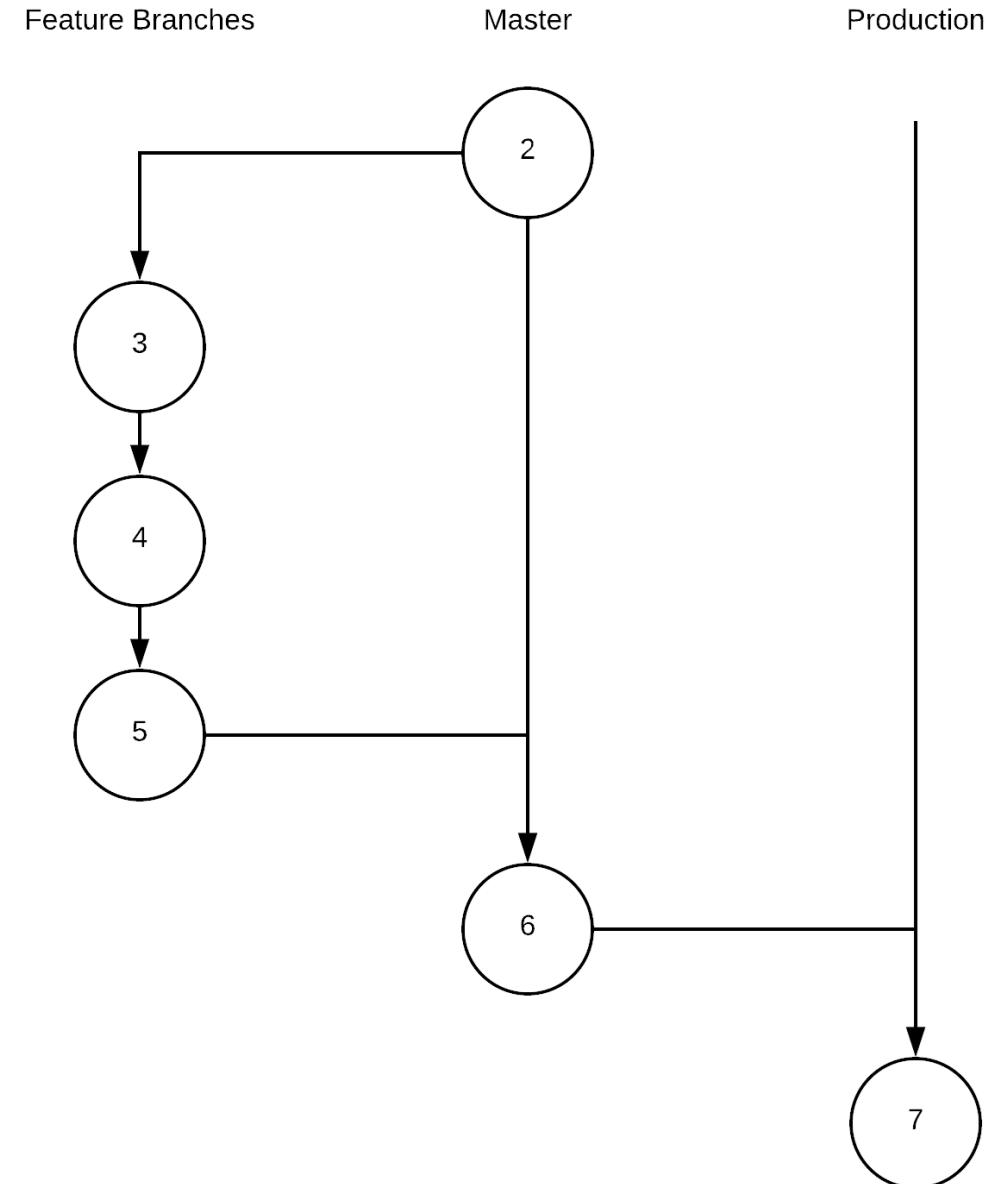


# GitLab Flow

- So far, we've explored GitFlow for collaborative project management, but it can be a complex one that doesn't suit all needs.
- There are alternatives, though, and one of these is posited by GitLab and thus known as GitLab Flow.
- GitLab Flow is actually a collection of different branching strategies that can be used depending on your environments and needs.

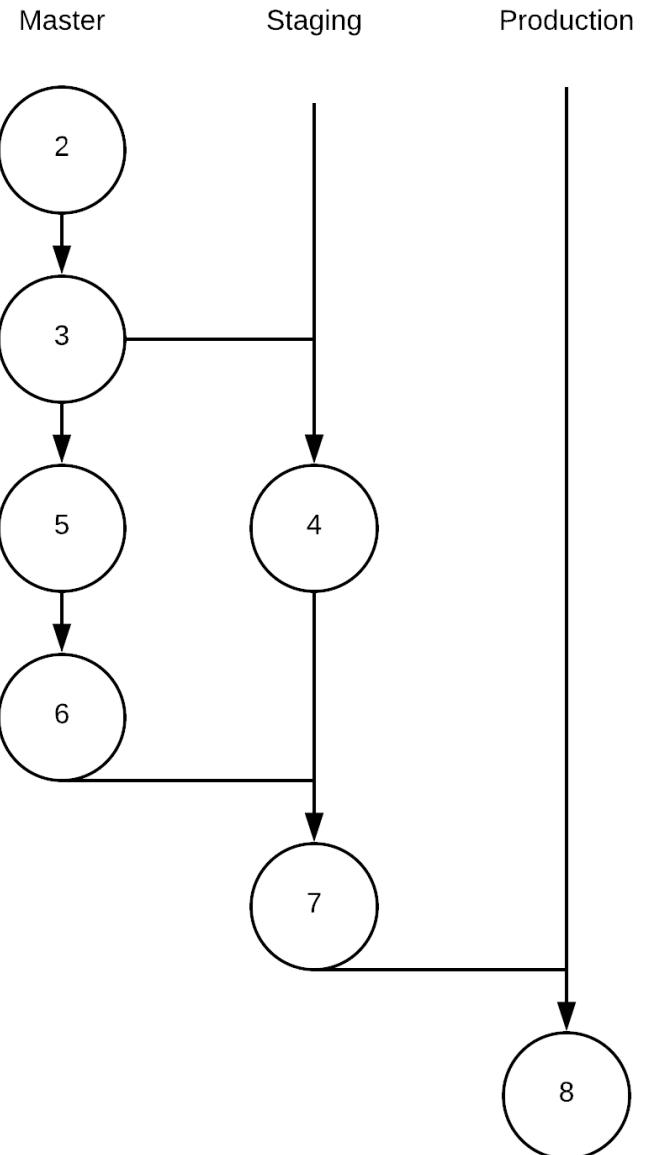
# Production branch

- While some development environments can be deployed as soon as commits are merged in, some projects aren't set up like that.
- A good example is apps that need to be approved by the app store and thus aren't on the same agile, continuous release schedule as the code in the master branch, or platforms that can only be deployed during certain late night/early morning windows so as not to affect the users of the systems:



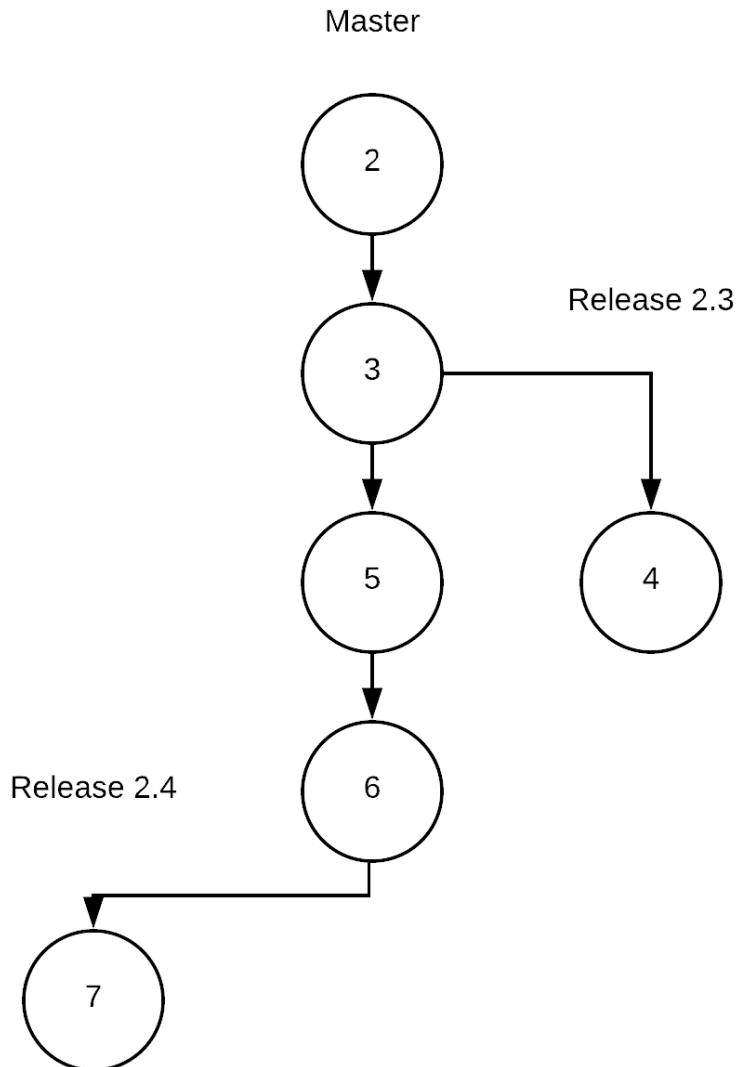
# Staging branch

- The production branch workflow is good, but sometimes you have multiple environments and your QA team might want to test stuff a bit further before it gets released.
- For example, you might have a staging or pre-production environment that you want to deploy to and ensure that everything is working before you deploy to production.
- In that case, you can use a workflow like the following:



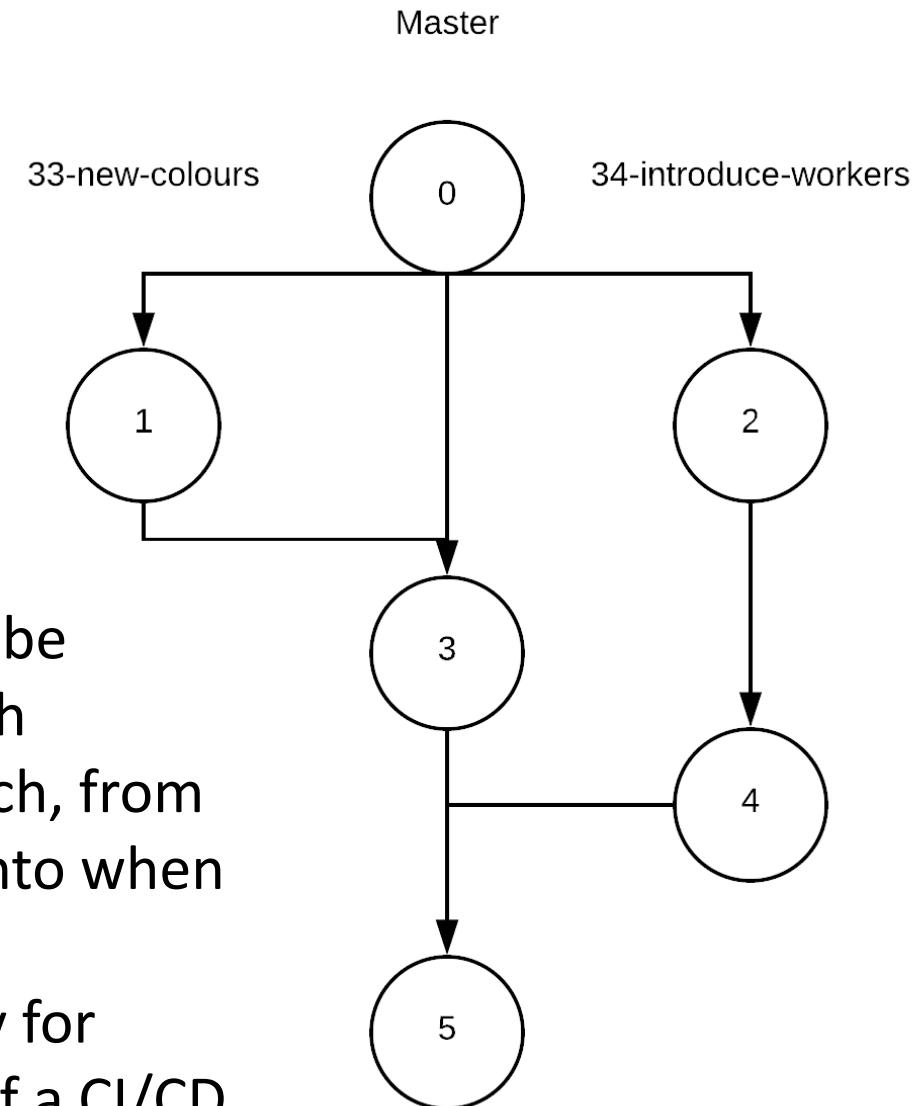
# Release branch

- The last flow suggested by the GitLab team is the release branch flow.
- A handy diagram of it is included as follows:



# Differences from GitHub flow

- If you've previously worked with GitHub, you might be familiar with the GitHub flow, a rigid workflow which recommends only having a continuous master branch, from which features are branched off and then merged into when created.
- Any code in the master branch is theoretically ready for deployment and continuous release with the help of a CI/CD platform:



# Summary

- In this lesson, we've looked at how to apply Git to your software development process and we've covered most of the basic commands, including add, commit, push, pull, branch, and merge.
- We also looked at GitFlow, a recommended workflow for Git software projects, and how to handle branching and merging correctly.

# Introducing the GitLab Architecture

In this lesson, we will be covering the following topics:

- The origins of GitLab
- GitLab CE or EE
- The core components of GitLab
- GitLab CI
- GitLab Runners
- Cloud native

# The origins of GitLab

- The story began in 2011, when Dimitri Zaporozhets, a web programmer from Ukraine, was faced with a common problem.
- He wanted to switch to Git for version management and GitHub to collaborate, but that was not allowed in his company.
- He needed a tool that did not hinder him in developing code and was easy to use.

# The origins of GitLab

- After this initiative, the project grew enormously:

Date	Fact
2011	Sytze Sybrandij, the future CEO of GitLab, is impressed by the GitLab project and code, and offers Zaporozhets the opportunity to try to commercialize it via <a href="https://about.gitlab.com/">https://about.gitlab.com/</a> .
2012	GitLab was announced to a broader audience via Hacker News ( <a href="https://news.ycombinator.com/item?id=4428278">https://news.ycombinator.com/item?id=4428278</a> ).
2013	Dimitri Zaporozhets decides to work full-time on GitLab and joins the company.
2015	GitLab becomes part of the Y Combinator class and received VC funding that year.
2018	GitLab receives another \$100 million of VC funding and is valued at \$1 billion.
2019	The GitLab company employs over 600 employees.

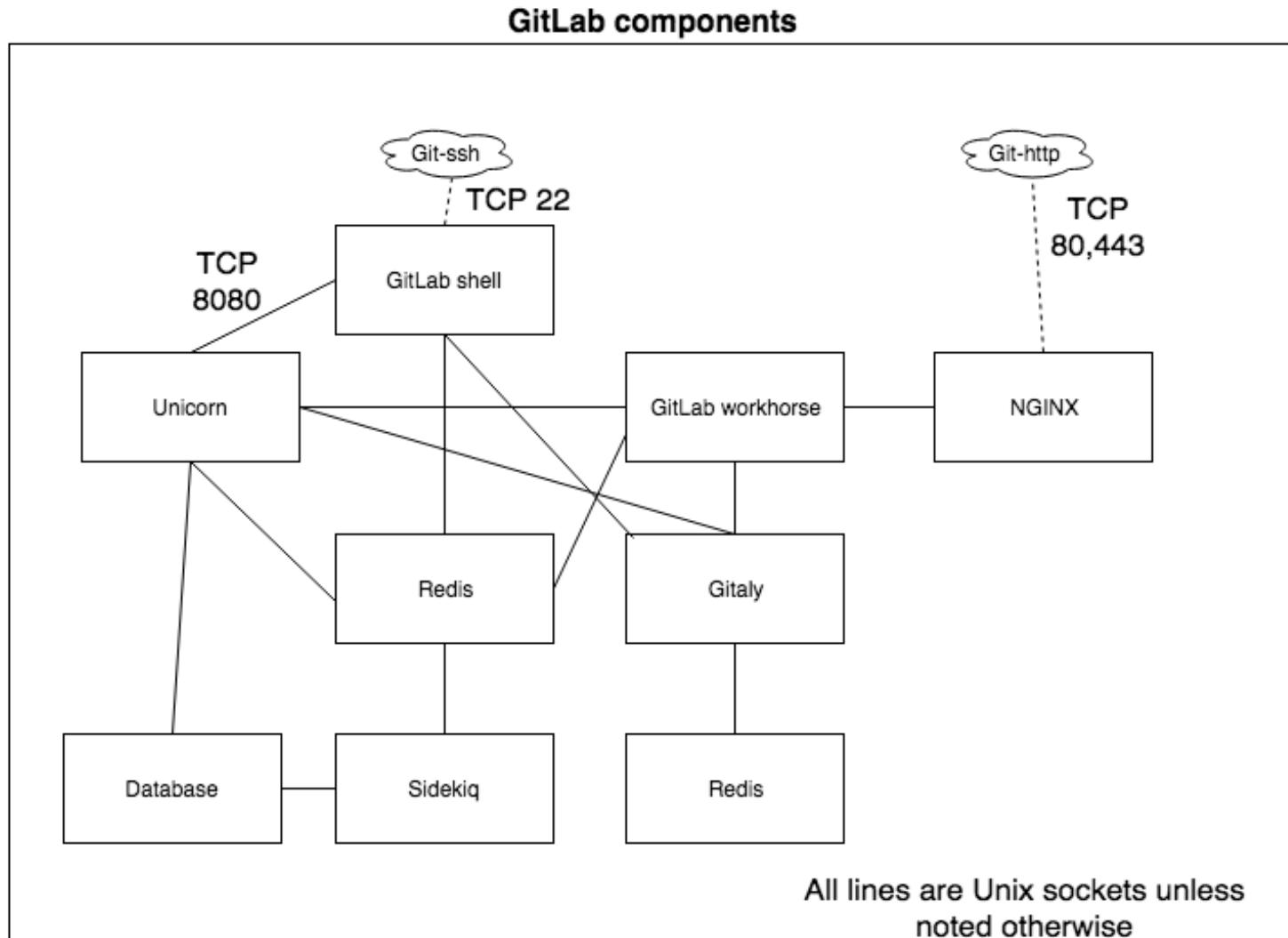
# Exploring GitLab editions – CE and EE

- The core of the GitLab software is called the CE.
- It is distributed under the MIT license, which is a permissive free software license created at the Massachusetts Institute of Technology.
- You are allowed to modify the software and use it in your creations.

# Exploring GitLab editions – CE and EE

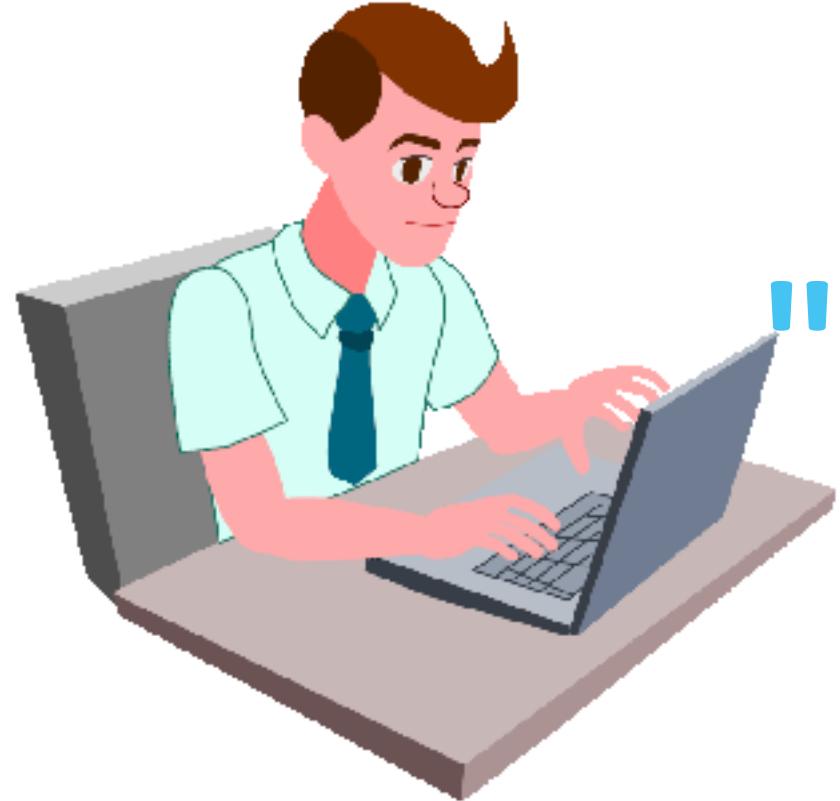
Version	Features (short list)
Starter	<p>Everything on core GitLab CE:</p> <ul style="list-style-type: none"><li>•CI/CD</li><li>•Project Issue Board</li><li>•Mattermost integrations</li><li>•Time tracking</li><li>•GitLab pages</li></ul>
Premium	<p>More enterprise features such as the following:</p> <ul style="list-style-type: none"><li>•Maven and NPM repository functionality</li><li>•Protected environments</li><li>•Burndown charts</li><li>•Multiple LDAP servers and Active Directory support</li></ul>
Ultimate	<p>All options, including the following:</p> <ul style="list-style-type: none"><li>•All security scanning tools</li><li>•Epics</li><li>•Free guest users</li><li>•Web terminal for the web IDE</li></ul>

# The core system components of GitLab



# Summary

- In this lesson, we have learned about the people and the organization behind GitLab.
- Starting from the beginning, we have shown you how the project has developed over the years.
- We went through the core components of GitLab.



# "Complete Lab 1&2"

# Configuring GitLab

## Using the Web UI

The following topics will be covered in this lesson:

- Configuring GitLab settings at the instance level
- Configuring GitLab settings at the group level
- Configuring GitLab settings at the project level

# Configuring GitLab settings at the instance level

- When you log on to GitLab as an administrator, you will notice a tool icon in the top right of the menu:



# Configuring GitLab settings at the instance level

The screenshot shows the GitLab Admin Area Dashboard. The left sidebar contains a navigation menu with the following items:

- Admin Area
- Overview
- Dashboard** (selected)
- Projects
- Users
- Groups
- Jobs
- Runners
- Cohorts
- ConvDev Index
- Monitoring
- Messages
- System Hooks
- Applications
- Abuse Reports (0)
- Deploy Keys
- Service Templates
- Labels
- Appearance
- Settings

The main dashboard area displays several key statistics and management sections:

- Projects: 1**: Includes a "New project" button.
- Users: 1**: Includes a "New user" button.
- Groups: 0**: Includes a "New group" button.
- Statistics**: Shows counts for Forks (0), Issues (0), Merge Requests (0), Notes (0), Snippets (0), SSH Keys (1), Milestones (0), and Active Users (1).
- Features**: Shows status for Sign up (green), LDAP (yellow), Gravatar (green), OmniAuth (yellow), Reply by email (yellow), Container Registry (yellow), GitLab Pages (yellow), and Shared Runners (green).
- Components**: Shows component names and versions, with "GitLab" having an "update asap" button:

Component	Version
GitLab	11.1.4 (63daf37)
GitLab Shell	7.1.4
GitLab Workhorse	v5.0.0
GitLab API	v4
Ruby	2.4.4p296
Rails	4.2.10
postgresql	9.6.8
Gitaly Servers	
- Latest projects**: Shows a recent project by Administrator: "mastering-gitlab..." created 2 months ago.
- Latest users**: Shows a recent user: Administrator created 2 months ago.
- Latest groups**: No data shown.

# Messages

- Your GitLab instance has a facility where you can send messages to all of your users.
- These broadcasts can come in handy if you want to inform your users about system-wide events, such as upgrades and scheduled downtime.
- The following is the Admin page, which you can find in the side menu:

The screenshot shows the 'Broadcast Messages' section of the GitLab Admin Area. At the top, there's a note: 'Broadcast messages are displayed for every user.' followed by a pencil icon. Below this is a red input field labeled 'Your message here'. Underneath, there are fields for 'Message' (empty), 'Starts at (UTC)' (set to 2018-11-04 14:57:00 UTC), and 'Ends at (UTC)' (set to 2018-11-04 14:58:00 UTC). A 'Customize colors' link is also present. A button labeled 'Add broadcast message' is located below the scheduling fields. At the bottom, a message states: 'After you've scheduled a new message, it can be reused later as well:' followed by a table showing a single message entry: Status: Pending, Preview: 'This is serious message!', Starts: 2018-11-04 14:57:00 UTC, Ends: 2018-11-04 14:58:00 UTC, with a checkbox and a delete button.

Status	Preview	Starts	Ends
Pending	This is serious message!	2018-11-04 14:57:00 UTC	2018-11-04 14:58:00 UTC

# System hooks

- A standard event is raised when you're creating a new project or user.
- Additionally, it can send other types of events as well. Just add a destination URL and (optionally) a secret token:

**Edit System Hook**

System hooks can be used for binding events when GitLab creates a User or Project.

---

**URL**

`http://docker.for.mac.localhost:8081`

**Secret Token**

`r`

Use this token to validate received payloads

**Trigger**

System hook will be triggered on set of events like creating project or adding ssh key. But you can also enable extra triggers like Push events.

**Repository update events**  
This URL will be triggered when repository is updated

**Push events**  
This URL will be triggered for each branch updated to the repository

**Tag push events**  
This URL will be triggered when a new tag is pushed to the repository

**Merge request events**  
This URL will be triggered when a merge request is created/updated/merged

**SSL verification**

**Enable SSL verification**

---

**Save changes** **Test ▾** **Remove**

# Plugins

- On this page, you also have the option to configure installed plugins.
- This basically fires a locally installed program instead of calling a URL with parameters.
- It requires you to place the plugin code in `/opt/gitlab/embedded/service/gitlab-rails/plugins`, and it has to be written in a certain way.

# Applications

- In this section of the administration page, you have the option to register third-party applications in order to use GitLab as an OAuth authorization provider.
- Open Authorization (OAuth) is an open standard for authorization.
- Users can give a program or website access to their private data that's kept on another website without revealing their username and password.

# Abuse reports



joustie

@joustie · Member since November 13, 2018

**Overview**

Activity

Groups

Contributed projects

Personal projects

Snippets

## Personal projects



Report abuse

# Abuse reports

## Report abuse to GitLab

Please use this form to report users to GitLab who create spam issues, comments or behave inappropriately.

A member of GitLab's abuse team will review your report as soon as possible.

User

joustie (@joustie)



Message

<https://gitlab.joustie.nl:8443/admin/users/joustie>  
This is an abuse report!

Explain the problem. If appropriate, provide a link to the relevant issue or comment.

Send report

# Abuse reports

Admin Area > Abuse Reports

## Abuse Reports

User	Reported by	Message	Action
joustie Joined 1 month ago	Administrator 23 seconds ago	<a href="https://gitlab.joustie.nl:8443/admin/users/joustie">https://gitlab.joustie.nl:8443/admin/users/joustie</a> This is an abuse report!	<button>Remove user &amp; report</button> <button>Block user</button> <button>Remove report</button>

# Push rules

- In this section, you can define all kinds of rules that will allow or disallow Git pushes:

Admin Area > Push Rules

---

## Pre-defined push rules.

Rules that define what git pushes are accepted for a project. All newly created projects will use this settings.

---

### **Committer restriction**

Users can only push commits to this repository that were committed with one of their own verified emails. This

### **Reject unsigned commits**

Only signed commits can be pushed to this repository. This setting will be applied to all projects unless overrid

### **Do not allow users to remove git tags with `git push`**

Tags can still be deleted through the web UI.

### **Check whether author is a GitLab user**

Restrict commits by author (email) to existing GitLab users

### **Prevent committing secrets to Git**

GitLab will reject any files that are likely to contain secrets. The list of file names we reject is available in the [do](#)

# Deploy Keys

- In this section, you can register SSH keys, which are known as Global Shared Deploy keys.
- They allow read-only or read-write (if enabled) access to be configured on any repository in the entire GitLab installation.
- When the administrator has registered them here, you can assign them in your project, as shown in the following screenshot:

## Deploy Keys

Collapse

Deploy keys allow read-only or read-write (if enabled) access to your repository. Deploy keys can be used for CI, staging or production servers. You can create a deploy key or add an existing one.

Create a new deploy key for this project

Title

Key

Paste a machine public key here. Read more about how to generate it [here](#)

Write access allowed

Allow this key to push to repository as well? (Default only allows pull access.)

Add key

Enabled deploy keys 1 Privately accessible deploy keys 0 Publicly accessible deploy keys 0

Deploy key	Project usage	Created
------------	---------------	---------

<b>Test</b> f4:cc:ec:76:d0:1c:86:1d:45:6d:a7:6e:b3:df:32:7c	Current project	⌚ 1 minute ago
--	-----------------	----------------



# Appearance

- You can define some cosmetic aspects of your GitLab instance on the Appearance settings page:

Admin Area > Appearance

## Appearance settings

You can modify the look and feel of GitLab here

### Navigation bar:

Header logo

no file selected

Maximum file size is 1MB. Pages are optimized for a 28px tall header logo

### Favicon:

Favicon

no file selected

Maximum file size is 1MB. Image size must be 32x32px. Allowed image formats are '.png' and '.ico'.  
Images with incorrect dimensions are not resized automatically, and may result in unexpected behavior.

### Sign in/Sign up pages:

Title

Description

Description parsed with *GitLab Flavored Markdown*.

Logo

no file selected

Maximum file size is 1MB. Pages are optimized for a 640x360 px logo.

### New project pages:

New project guidelines

Guidelines parsed with *GitLab Flavored Markdown*.

Save

# Appearance

- After you have done this, log out. You will be redirected to the front page:

## GitLab Community Edition



### Open source software to collaborate on code

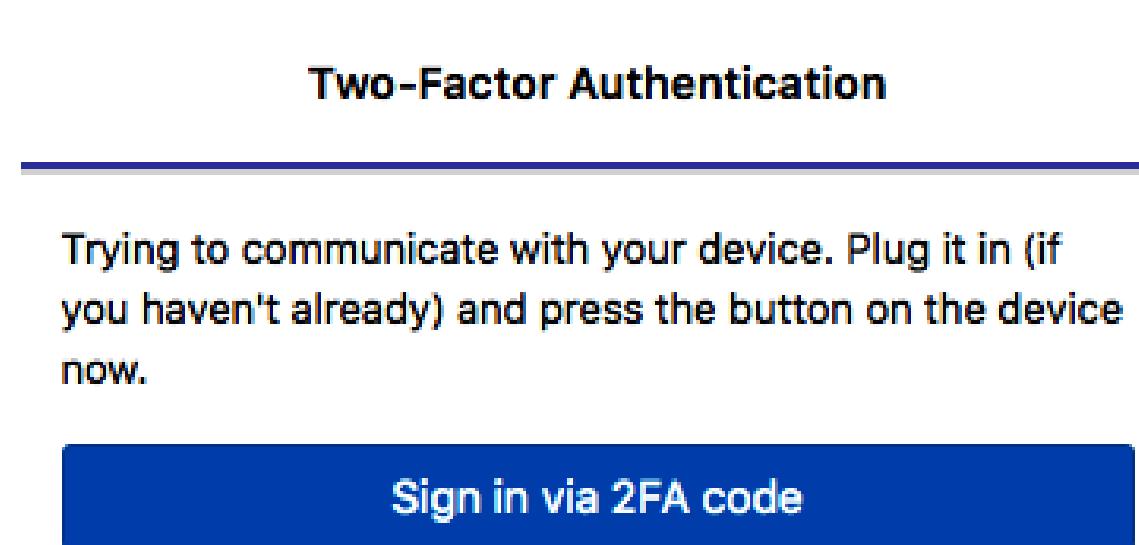
Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

<a href="#">Sign in</a>	<a href="#">Register</a>
Username or email	
<input type="text"/>	
Password	
<input type="password"/>	
<input type="checkbox"/> Remember me	<a href="#">Forgot your password?</a>
<a href="#">Sign in</a>	

Didn't receive a confirmation email? [Request a new one.](#)

# Sign-in restrictions

- It is possible to use a hardware token device as a second factor, as you can see in the following screenshot (this only works in Chrome):



# Sign-in restrictions

- You can also choose to use a code generator app such as Google Authentication:

**Two-Factor Authentication**

---

Two-factor authentication code

*Enter the code from the two-factor app on your mobile device. If you've lost your device, you may enter one of your recovery codes.*

**Verify code**

# Elasticsearch

- Elasticsearch indexing
- Use the [new repository indexer \(beta\)](#)
- Search with Elasticsearch enabled

## URL

`http://localhost:9200`

The url to use for connecting to Elasticsearch. Use a comma-separated list to support clustering (e.g., "http://localhost:9200, http://localhost:9201").

## Number of Elasticsearch shards

5



How many shards to split the Elasticsearch index over. Changes won't take place until the index is [recreated](#).

## Number of Elasticsearch replicas

1



How many replicas each Elasticsearch shard has. Changes won't take place until the index is [recreated](#).

# Elasticsearch

- You can also limit what will be indexed:

## Elasticsearch indexing restrictions

- Limit namespaces and projects that can be indexed

# Elasticsearch

- Another option is to connect to an Elasticsearch instance that you are running in the Amazon cloud.
- You can specify connection settings here as well if you have this set up:

**Elasticsearch AWS IAM credentials**

Using AWS hosted Elasticsearch with IAM credentials

**AWS region**

us-east-1

Region that elasticsearch is configured

**AWS Access Key**

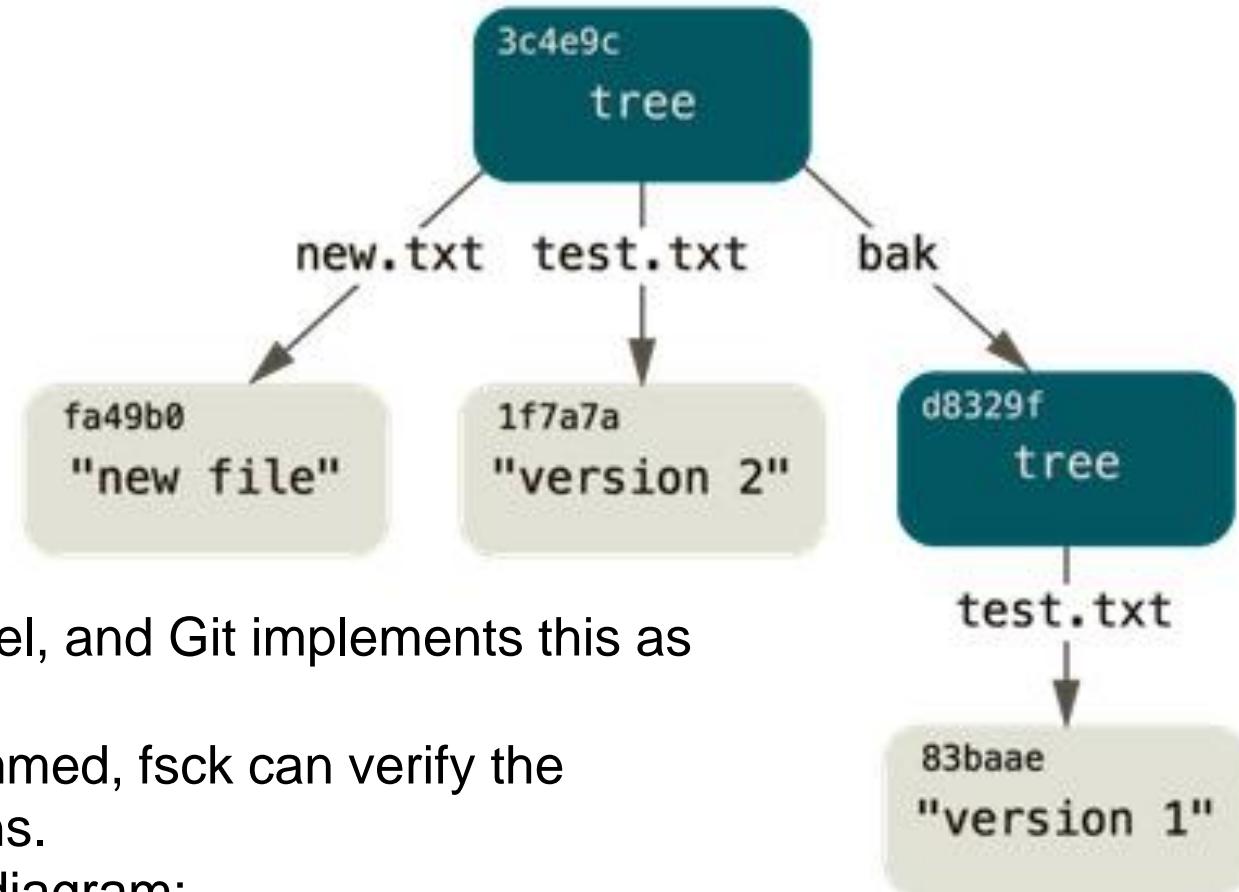
AWS Access Key. Only required if not using role instance credentials

**AWS Secret Access Key**



AWS Secret Access Key. Only required if not using role instance credentials

# Repository maintenance



- A filesystem is classed as a graph model, and Git implements this as tree and blob objects.
- Because all of the items are check-summed, fsck can verify the integrity of the objects and their relations.
- This graph is depicted in the following diagram:

# Auto DevOps settings

## Continuous Integration and Deployment

Auto DevOps, runners and job artifacts

- Default to Auto DevOps pipeline for all projects

The Auto DevOps pipeline will run if no alternative CI configuration file is found. [More information](#)

### Auto devops domain

domain.com

Specify a domain to use by default for every project's Auto Review Apps and Auto Deploy stages.

[Collapse](#)

# Help page

- You can also customize the way the Help page for GitLab is presented.
- There's the option to provide some custom text, which will be displayed on top of the Help page:

## Help page

Help page text and support page url.

### Help page text

This is a help page.

Markdown enabled

Hide marketing-related entries from help

### Support page URL

<http://blog.joustie.nl>

Alternate support URL for help page

[Save changes](#)

# Help page

- The following is a screenshot of the standard Help page:

Help > Help

## GitLab Community Edition 11.4.5

update asap

GitLab is open source software to collaborate on code.

Manage git repositories with fine-grained access controls that keep your code secure.

Perform code reviews and enhance collaboration with merge requests.

Each project can also have an issue tracker and a wiki.

Used by more than 100,000 organizations, GitLab is the most popular solution to manage git repositories on-premises.

Read more about GitLab at [about.gitlab.com](http://about.gitlab.com).

[Check the current instance configuration](#)

## GitLab Documentation

Welcome to [GitLab](#), a Git-based fully featured platform for software development!

GitLab offers the most scalable Git-based fully integrated platform for software development, with flexible products and subscriptions. To understand what features you have access to, check the [GitLab subscriptions](#) below.

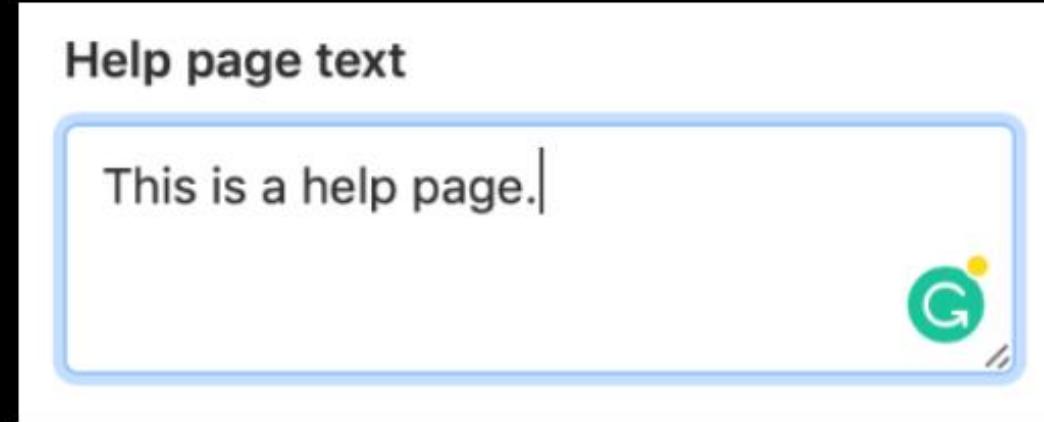
Quick help

[See our website for getting help](#)

[Use the search bar on the top of this page](#)

# Help page

Let's make some changes by adding **This is a help page.**:



The result will be as follows:

A screenshot of the GitLab Community Edition 11.4.5 interface. At the top left, there is a breadcrumb navigation "Help > Help" and the text "This is a help page.". Below this, the title "GitLab Community Edition 11.4.5" is followed by a red button labeled "update asap". A horizontal line separates this from the "GitLab Documentation" section. Under "GitLab Documentation", there is a welcome message: "Welcome to GitLab, a Git-based fully featured platform for software development! GitLab offers the most scalable Git-based fully integrated platform for software development, with flexible products and subscriptions. To understand what features you have access to, check the GitLab subscriptions below." Another horizontal line follows, with the heading "Shortcuts to GitLab's most visited docs:". Below this, there are two buttons: "General documentation" and "GitLab CI/CD docs". On the right side of the page, there is a sidebar with the title "Quick help" containing four items: "See our website for getting help", "Use the search bar on the top of this page", and "Use shortcuts".

# Pages

- If you use the GitLab Pages feature, you can specify the maximum size of pages.
- You can set it to zero if you want the size to be unlimited.
- You can also allow users to prove that they own a domain before you serve a page for it:

## Pages

[Collapse](#)

Size and domain settings for static websites

### Maximum size of pages (MB)

100



0 for unlimited

#### Require users to prove ownership of custom domains

Domain verification is an essential security measure for public GitLab sites. Users are required to demonstrate they control a domain before it is enabled [?](#)

# Localization

- This is a big section of the settings since there are many localization settings for software products.
- The only one that is exposed in this screen is Default first day of the week:

## Localization

Various localization settings.



Collapse

### Default first day of the week

Monday

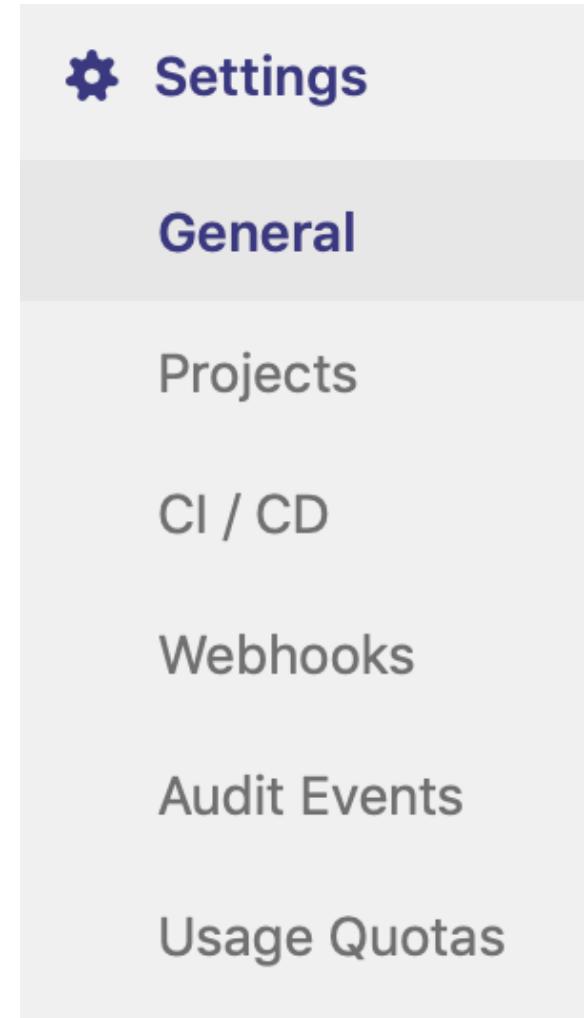


Default first day of the week in calendars and date pickers.

**Save changes**

# Configuring GitLab settings at the group level

- You will see a submenu with the items that you can configure.
- It looks a lot like the UI in the admin area but is, of course, scoped to the group:



# Configuring GitLab settings at the group level

## Restrict access by IP address

192.168.1.0/24

This group, including all subgroups, projects and git repositories, will only be reachable from the specified IP address range.

Example: [192.168.0.0/24](#). [Read more.](#)

## Large File Storage

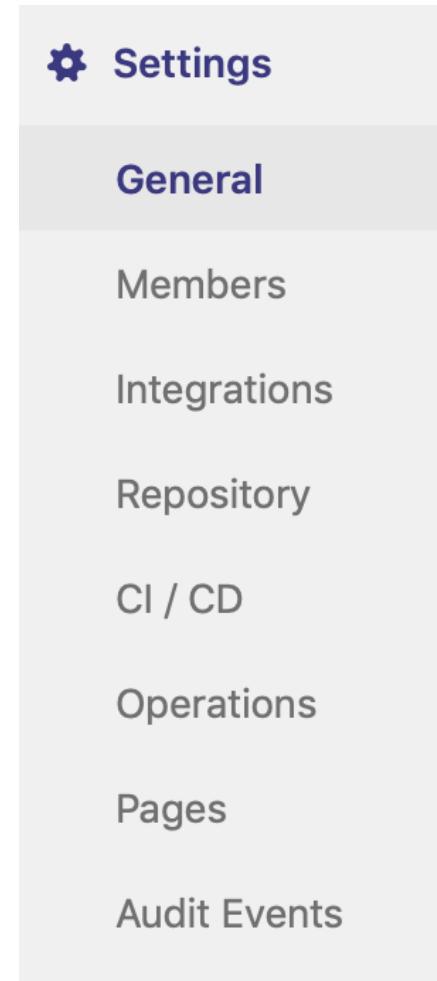
Check the [documentation](#).

- Allow projects within this group to use Git LFS  
This setting can be overridden in each project.

- In the following screenshot, you can see that 192.168.1.0/24 is the only IP range that's allowed to see the group content:

# Configuring GitLab settings at the project level

- There are also options for setting an individual project.
- If you browse to one of your projects, you will see a Settings menu on the left:



# Naming, topics, avatar

- Under the General settings, you can find the fields:

## Naming, topics, avatar

Update your project name, topics, description and avatar.

**Project name**

eventmanager

**Project ID**

3

**Topics**

Separate topics with commas.

**Project description (optional)**

**Repository size limit (MB)**

The total size of this project's repository including files in LFS will be limited to this size. 0 for unlimited. Leave empty to inherit the group/global value.

**Project avatar**



No file chosen

The maximum file size allowed is 200KB.

# Visibility, project features, permissions

## Visibility, project features, permissions

Choose visibility level, enable/disable project features (issues, repository, wiki, snippets) and set permissions.

**Project visibility** 

Private 

The project is accessible only by members of the project. Access must be granted explicitly to each user.

**Issues**

Lightweight issue tracking system for this project

 Only Project Members 

**Repository**

View and edit files in this project

 Only Project Members 

**Merge requests**

Submit changes to be merged upstream

 Only Project Members 

# Merge requests

- For every executed merge request, there is a Git session on the server running the same Git binary that you have on your workstation.
- For instance, you can specify that the server side never does a merge commit:

## Merge requests

Collapse

Choose your merge method, options, checks, and set up a default merge request description template.

### Merge method

This will dictate the commit history when you merge a merge request

Merge commit

Every merge creates a merge commit

Merge commit with semi-linear history

Every merge creates a merge commit

Fast-forward merges only

When conflicts arise the user is given the option to rebase

Fast-forward merge

No merge commits are created

Fast-forward merges only

When conflicts arise the user is given the option to rebase

### Merge options

Additional merge request capabilities that influence how and when merges will be performed

Merge pipelines will try to validate the post-merge result prior to merging

Pipelines need to be configured to enable this feature. [?](#)

Allow merge trains

Automatically resolve merge request diff discussions when they become outdated

Show link to create/view merge request when pushing from the command line

### Merge checks

These checks must pass before merge requests can be merged

Pipelines must succeed

Pipelines need to be configured to enable this feature. [?](#)

All discussions must be resolved

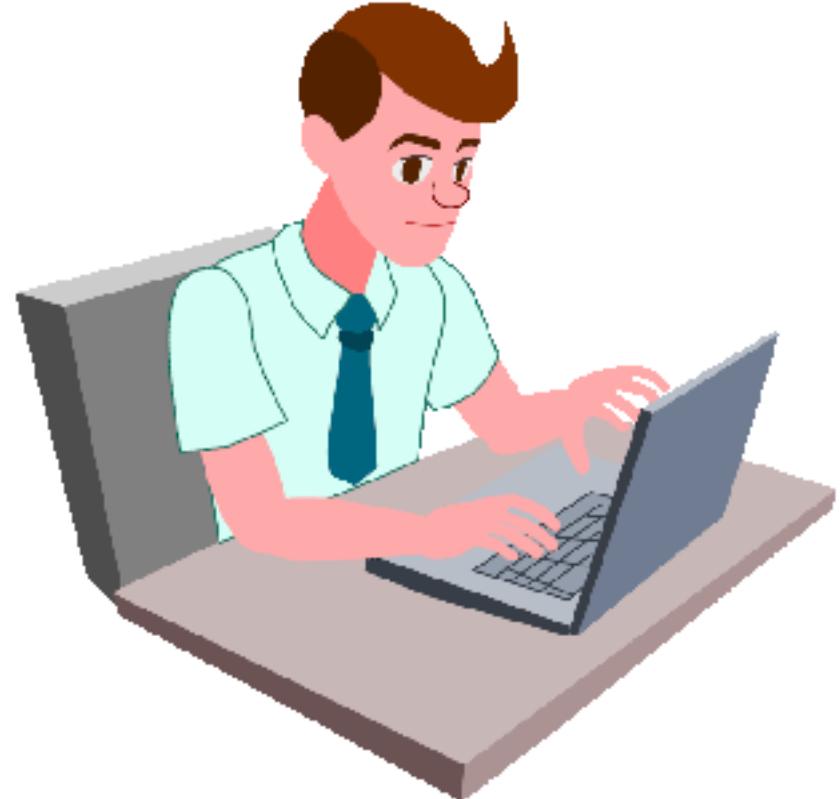
### Default description template for merge requests [?](#)

Description parsed with [GitLab Flavored Markdown](#)

Save changes

# Summary

- In this lesson, we discussed how to configure an existing GitLab application instance via the web interface.
- The administration pages of GitLab give you a lot of control over your instance.
- After going through those pages, we explained the various items that can be managed.



# "Complete Lab 3"

# 3. Branching

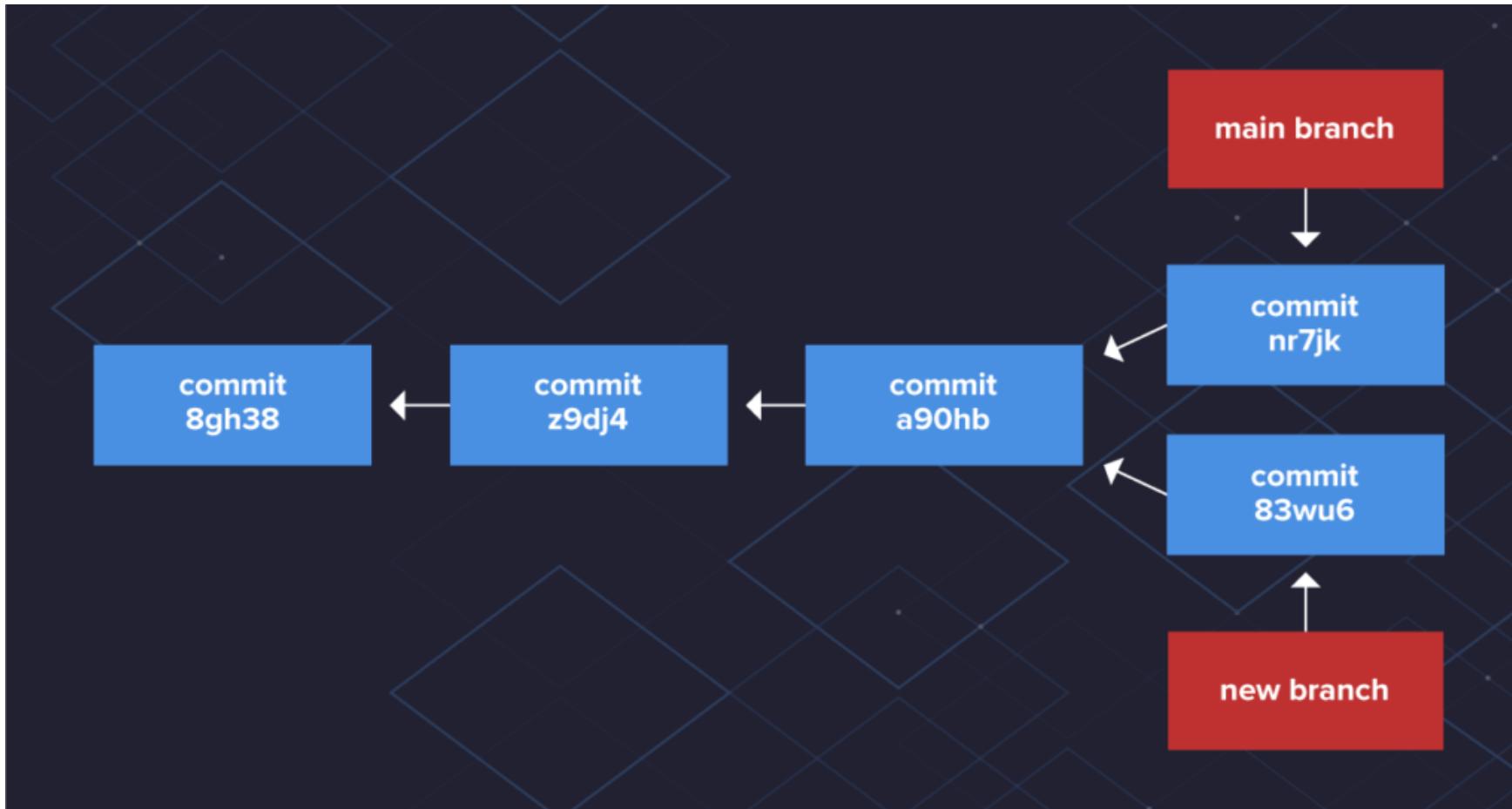
# Git branching

- Branches are not just exclusive to Git. However, in this course we focus on Git due to the many advantages this model of branching offers.
- Consequently, before we delve into the various branching strategies out there, including Git branching strategies.
- Put simply, Git and other version control tools allow developers to track, manage and organize their code.

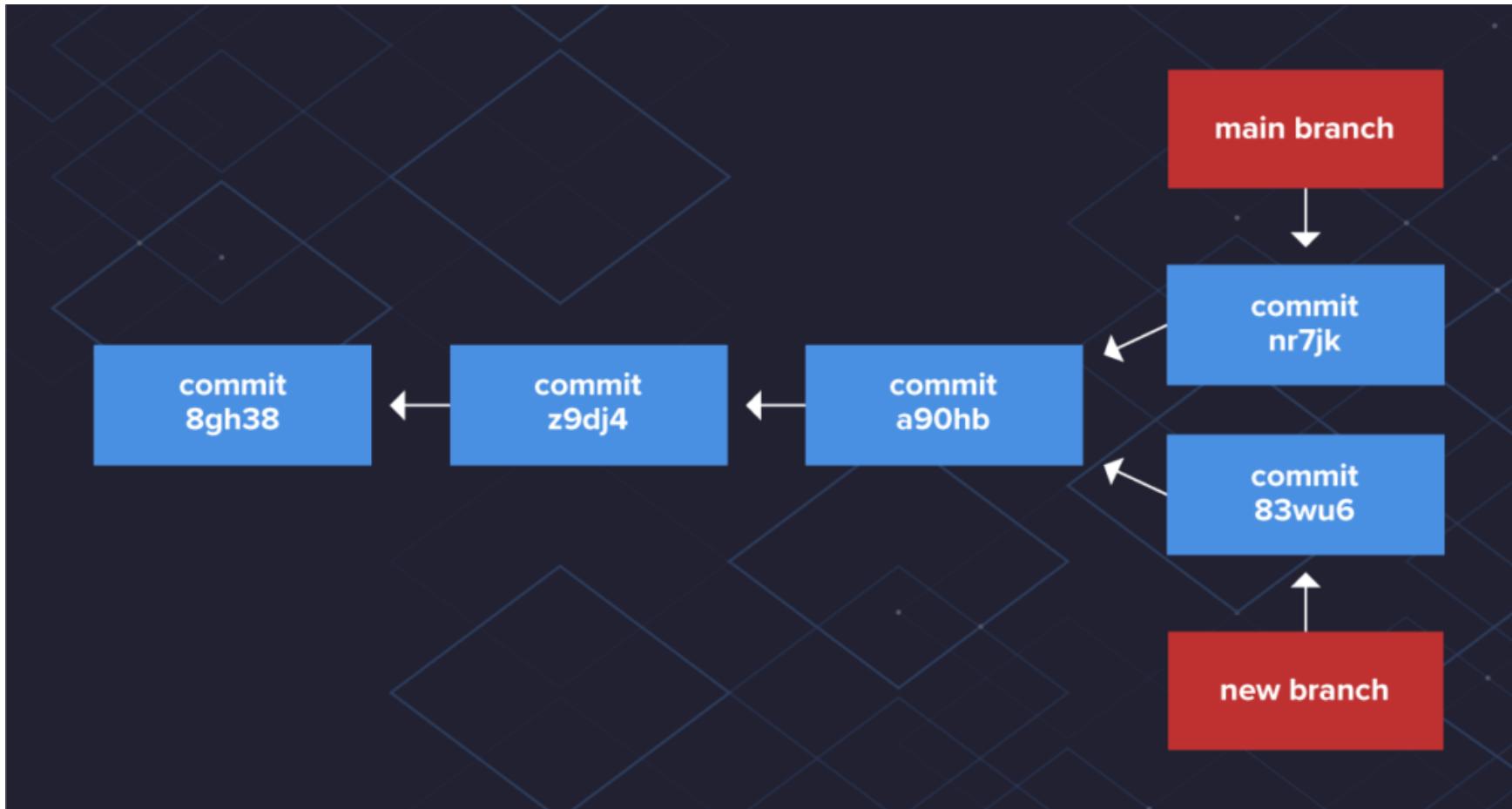
# Git Branching



# Git Branching



# Git Branching



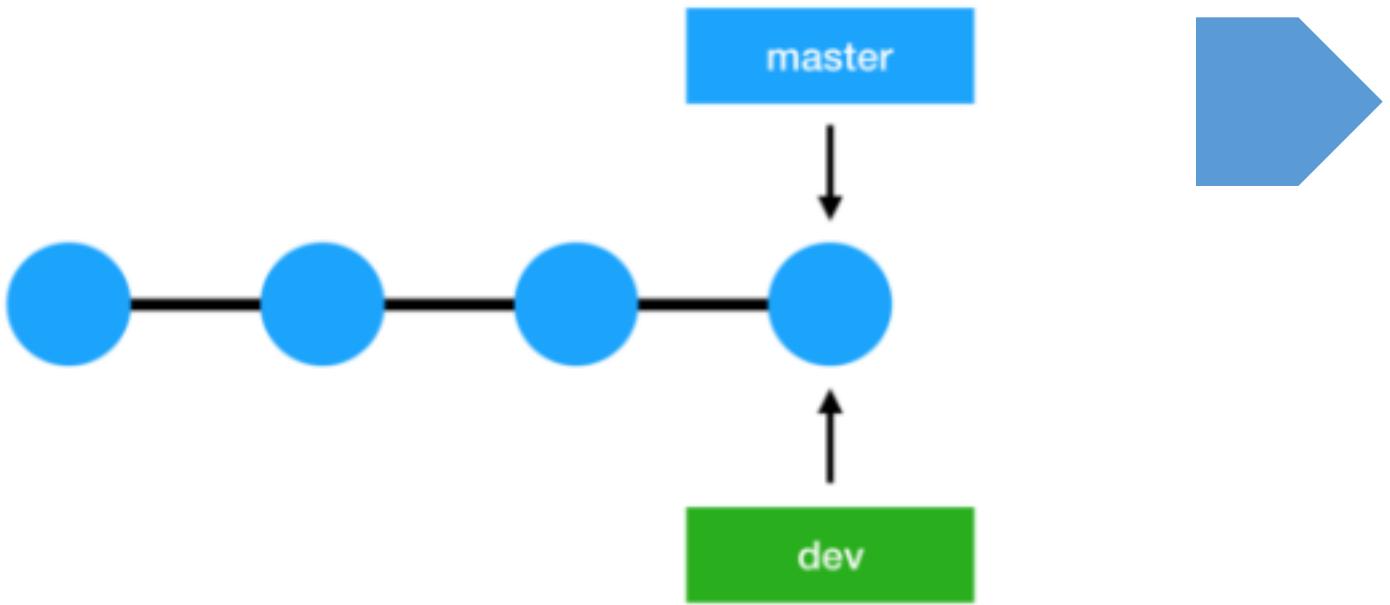
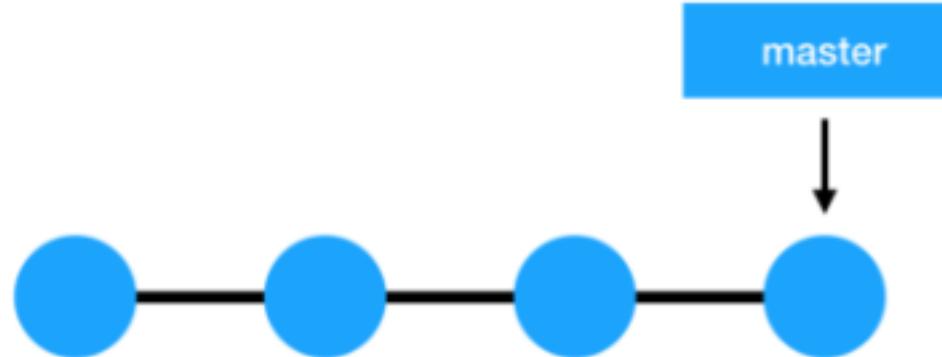
# Branch Naming Strategies

- Branch names can be anything you'd like.
- Your organization or project may have standards outlined for branch naming.
- For example, naming the branch based on the person responsible for working on the branch and a description or work item:
  - username/description
  - username/workitem

# Branch Naming Strategies

- You can name a branch to indicate the branch's function, like a feature, bug fix, or hotfix:
  - bugfix/description
  - feature/feature-name
  - hotfix/description

# Introduction



# Introduction

- If you are developing a small application in a big corporation as a developer.
- Or you are trying to wrap your head around an open source project from GitHub, you have already been using branches with Git.

# Managing your local branches

- Suppose you are just having your local Git repository, and you have no intentions at the moment to share the code with others.
- However, you can easily share this knowledge while working with a repository with one or more remotes.

# Getting ready

- Use the following command to clone the jgit repository to match:

```
$ git clone https://git.eclipse.org/r/jgit/jgit  
$ cd jgit
```

# How to do it...

- Whenever you start working on a bug fix or a new feature in your project, you should create a branch.
- You can do so using the following code:

```
$ git branch newBugFix
```

```
$ git branch
```

```
* master
```

```
newBugFix
```

# How to do it...

- The newBugFix branch points to the current HEAD I was on at the time of the creation.
- You can see the HEAD with git log -1:

```
$ git log -1 newBugFix --format=format:%H  
25fe20b2dbb20cac8aa43c5ad64494ef8ea64ffc
```

- If you want to add a description to the branch, you can do it with the --edit-description option for the git branch command:

```
$ git branch --edit-description newBugFix
```

# How to do it...

- The previous command will open an editor where you can type in a description:

Refactoring the Hydro controller

The hydro controller code is currently horrible needs to be refactored.

- Close the editor and the message will be saved.

# How it works...

- Git stores the information in the local git config file; this also means that you cannot push this information to a remote repository.
- To retrieve the description for the branch, you can use the --get flag for the git config command:

```
$ git config --get branch.newBugFix.description
```

Refactoring the Hydro controller

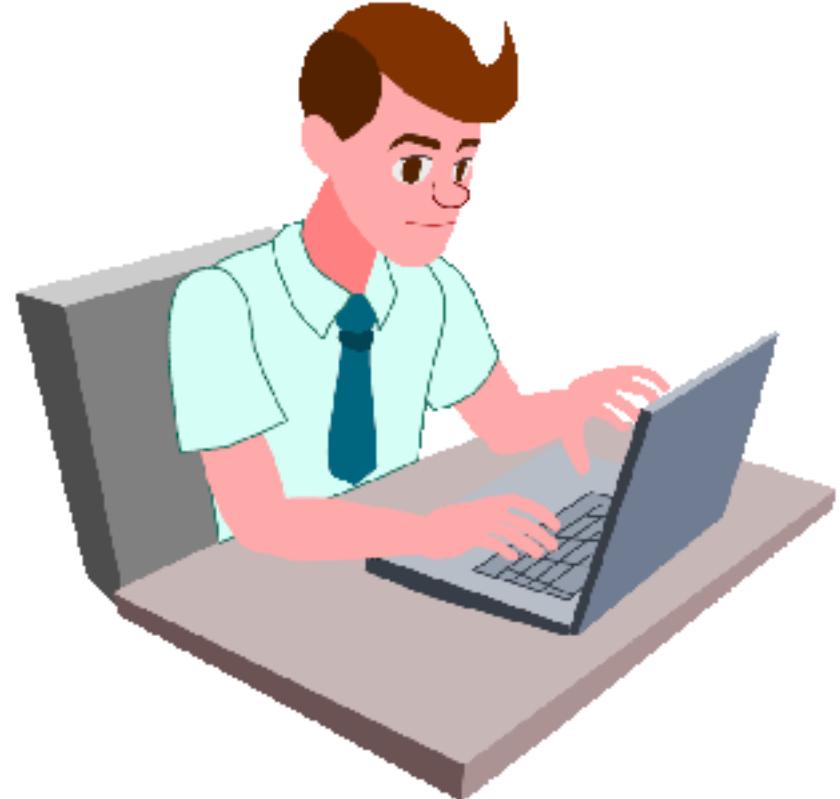
The hydro controller code is currently horrible needs to be refactored.

# How it works...

- The branch information is stored as a file in .git/refs/heads/newBugFix:

```
$ cat .git/refs/heads/newBugFix  
25fe20b2dbb20cac8aa43c5ad64494ef8ea64ffc
```

- Note that it is the same commit hash we retrieved with the git log command

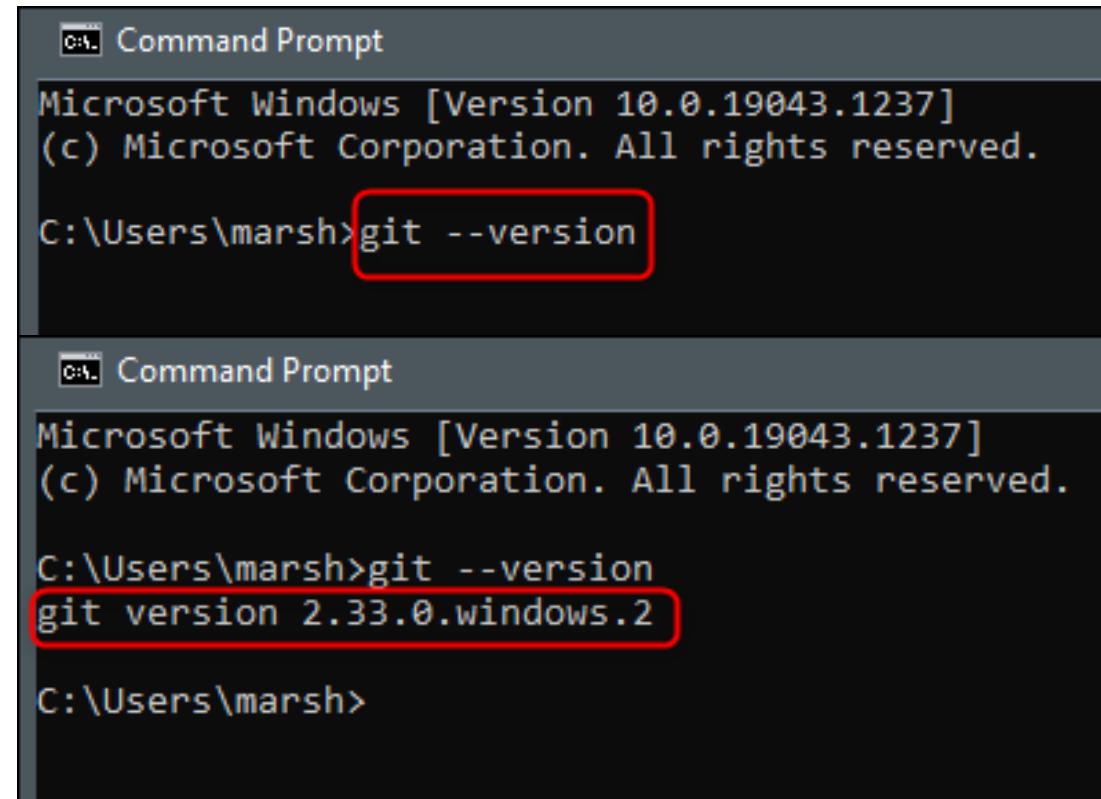


# "Complete Lab 4"

# 4. Configuring Git

# Checking your Git version

- Once open, run this command:  
`git --version`



```
Command Prompt
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\marsh>git --version

Command Prompt
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\marsh>git --version
git version 2.33.0.windows.2

C:\Users\marsh>
```

The image shows two separate Command Prompt windows. Both windows display the same system information at the top: "Microsoft Windows [Version 10.0.19043.1237]" and "(c) Microsoft Corporation. All rights reserved.". In the first window, the command "git --version" is entered at the prompt, and its output "git version 2.33.0.windows.2" is shown below it. The entire command line "git --version" is highlighted with a red rectangular box. In the second window, the command is also entered at the prompt, but its output has been removed or obscured by a large black redaction box.

# Git Configuration Levels

- In Git different layers that can be configured. The layers are:

**SYSTEM:** This layer is system-wide and found in `/etc/gitconfig`

**GLOBAL:** This layer is global for the user and found in `~/.gitconfig`

**LOCAL:** This layer is local to the current repository and found in `.git/config`

# Viewing your configurations

- You can use:

`git config --list`

- or look at your `~/.gitconfig` file. The local configuration will be in your repository's `.git/config` file.

Use:

`git config --list --show-origin`

to see where that setting is defined (global, user, repo, etc...)

# Configuring your username and email

```
# Create a project specific config, you have to execute this  
under the project's directory.  
$ git config user.name "John Doe"
```

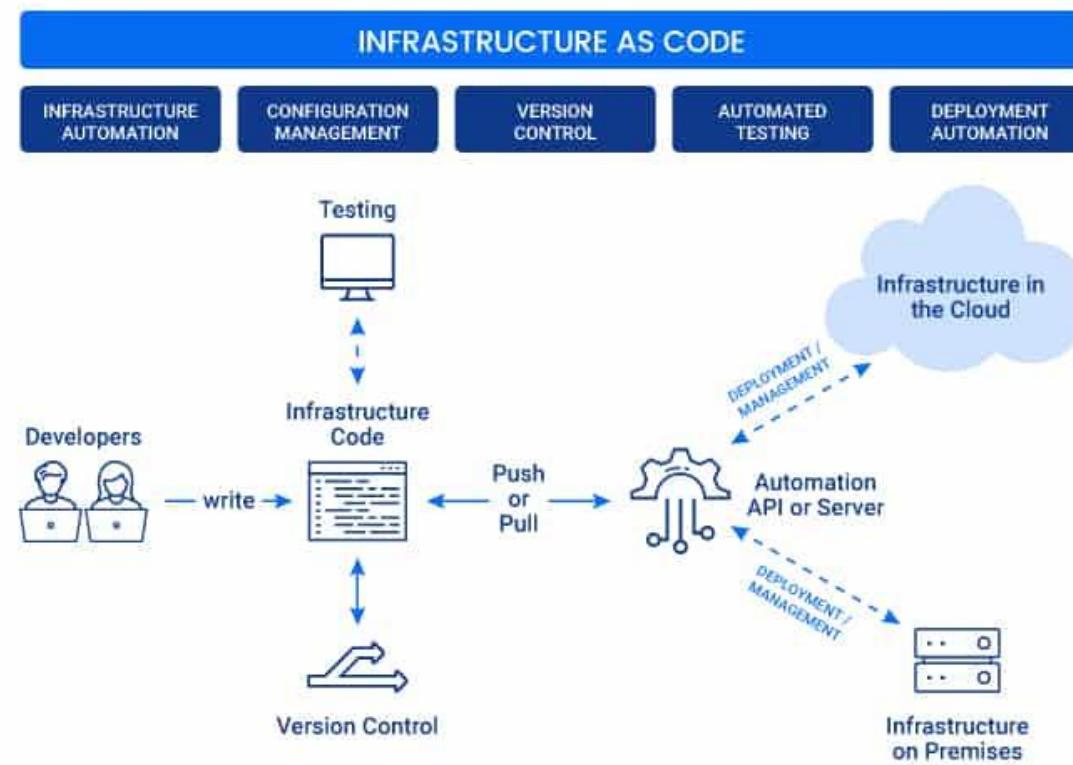
```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

# 5. Terraform Basics

# Terraform Beginner's Guide: Everything You Should Know

## What Is Infrastructure as Code (IaC)?

- Infrastructure as Code (IaC) is a widespread terminology among DevOps professionals and a key DevOps practice in the industry. It is the process of managing and provisioning the complete IT infrastructure (comprises both physical and virtual machines) using machine-readable definition files.
- It helps in automating the complete data center by using programming scripts.



## Popular IaC Tools:

- Terraform An open-source declarative tool that offers pre-written modules to build and manage an infrastructure.
- Chef: A configuration management tool that uses cookbooks and recipes to deploy the desired environment. Best used for Deploying and configuring applications using a pull-based approach.
- Puppet: Popular tool for configuration management that follows a Client-Server Model. Puppet needs agents to be deployed on the target machines before the puppet can start managing them.
- Ansible: Ansible is used for building infrastructure as well as deploying and configuring applications on top of them. Best used for Ad hoc analysis.
- Packer: Unique tool that generates VM images (not running VMs) based on steps you provide. Best used for Baking compute images.
- Vagrant: Builds VMs using a workflow. Best used for Creating pre-configured developer VMs within VirtualBox.

## **What Is Terraform?**

Terraform is one of the most popular Infrastructure-as-code (IaC) tool, used by DevOps teams to automate infrastructure tasks.

It is used to automate the provisioning of your cloud resources.

Terraform is an open-source, cloud-agnostic provisioning tool developed by HashiCorp and written in GO language.

## **Benefits of using Terraform:**

Does orchestration, not just configuration management

Supports multiple providers such as AWS, Azure, Oracle, GCP, and many more

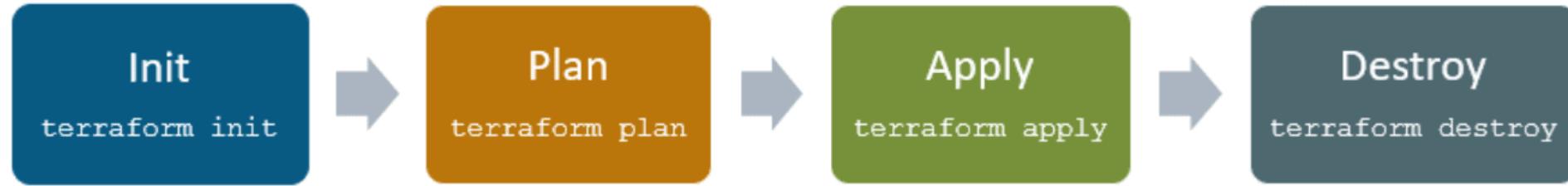
Provide immutable infrastructure where configuration changes smoothly

Uses easy to understand language, HCL (HashiCorp configuration language)

Easily portable to any other provider

## Terraform Lifecycle

Terraform lifecycle consists of – init, plan, apply, and destroy.



- Terraform init initializes the (local) Terraform environment. Usually executed only once per session.
- Terraform plan compares the Terraform state with the as-is state in the cloud, build and display an execution plan. This does not change the deployment (read-only).
- Terraform apply executes the plan. This potentially changes the deployment.
- Terraform destroy deletes all resources that are governed by this specific terraform environment.

## Terraform Core Concepts

- **Variables:** Terraform has input and output variables, it is a key-value pair. Input variables are used as parameters to input values at run time to customize our deployments. Output variables are return values of a terraform module that can be used by other configurations.
- **Provider:** Terraform users provision their infrastructure on the major cloud providers such as AWS, Azure, OCI, and others. A provider is a plugin that interacts with the various APIs required to create, update, and delete various resources.
- **Module:** Any set of Terraform configuration files in a folder is a module. Every Terraform configuration has at least one module, known as its root module.
- **State:** Terraform records information about what infrastructure is created in a Terraform state file. With the state file, Terraform is able to find the resources it created previously, supposed to manage and update them accordingly.

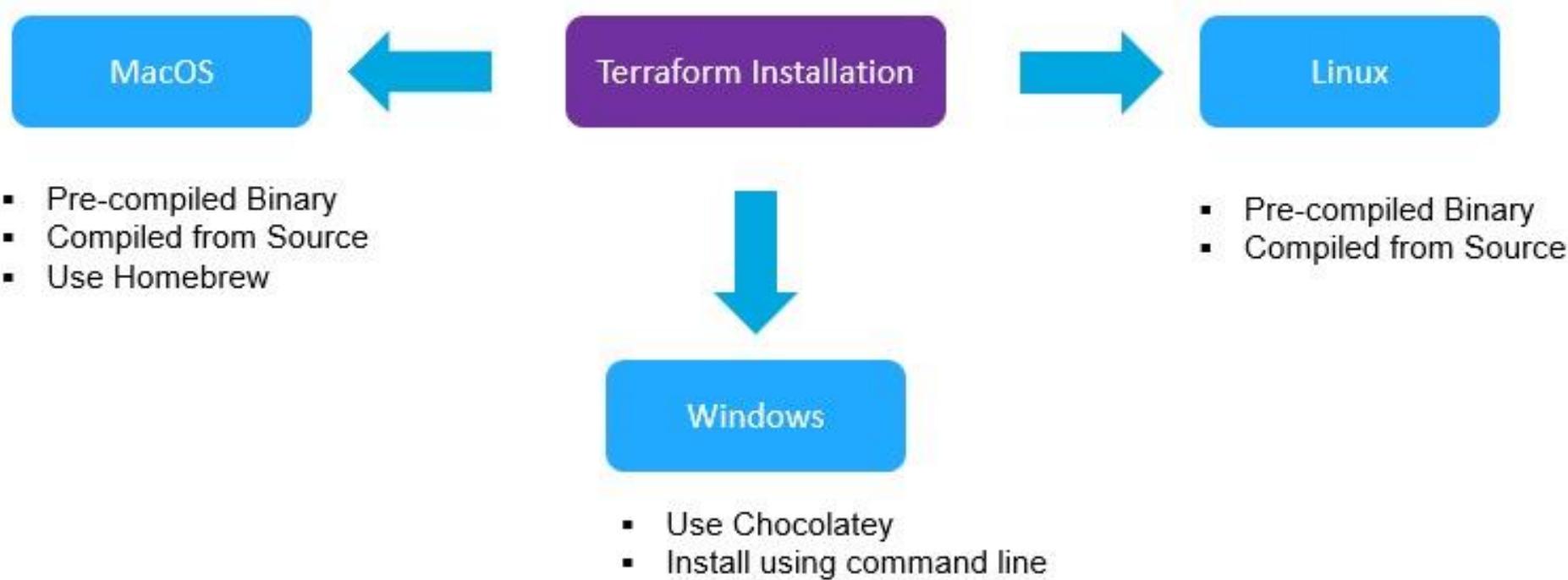
- **Resources:** Cloud Providers provides various services in their offerings, they are referenced as Resources in Terraform. Terraform resources can be anything from compute instances, virtual networks to higher-level components such as DNS records. Each resource has its own attributes to define that resource.
- **Data Source:** Data source performs a read-only operation. It allows data to be fetched or computed from resources/entities that are not defined or managed by Terraform or the current Terraform configuration.
- **Plan:** It is one of the stages in the Terraform lifecycle where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.
- **Apply:** It is one of the stages in the Terraform lifecycle where it applies the changes real/current state of the infrastructure in order to achieve the desired state.

## Terraform Installation

Before you start working, make sure you have Terraform installed on your machine, it can be installed on any OS, say Windows, macOS, Linux, or others.

Terraform installation is an easy process and can be done in a few minutes.

Read our blog to know how to install Terraform in Linux, Mac, Windows

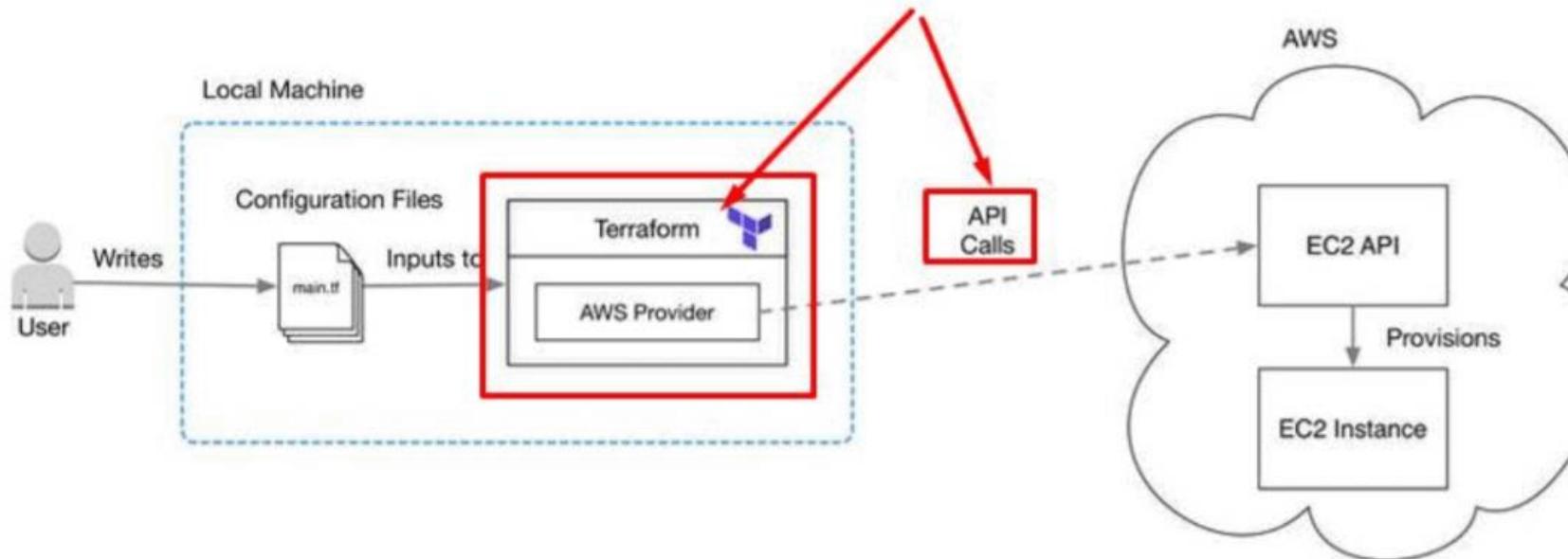


## Terraform Providers

A provider is responsible for understanding API interactions and exposing resources.

It is an executable plug-in that contains code necessary to interact with the API of the service.

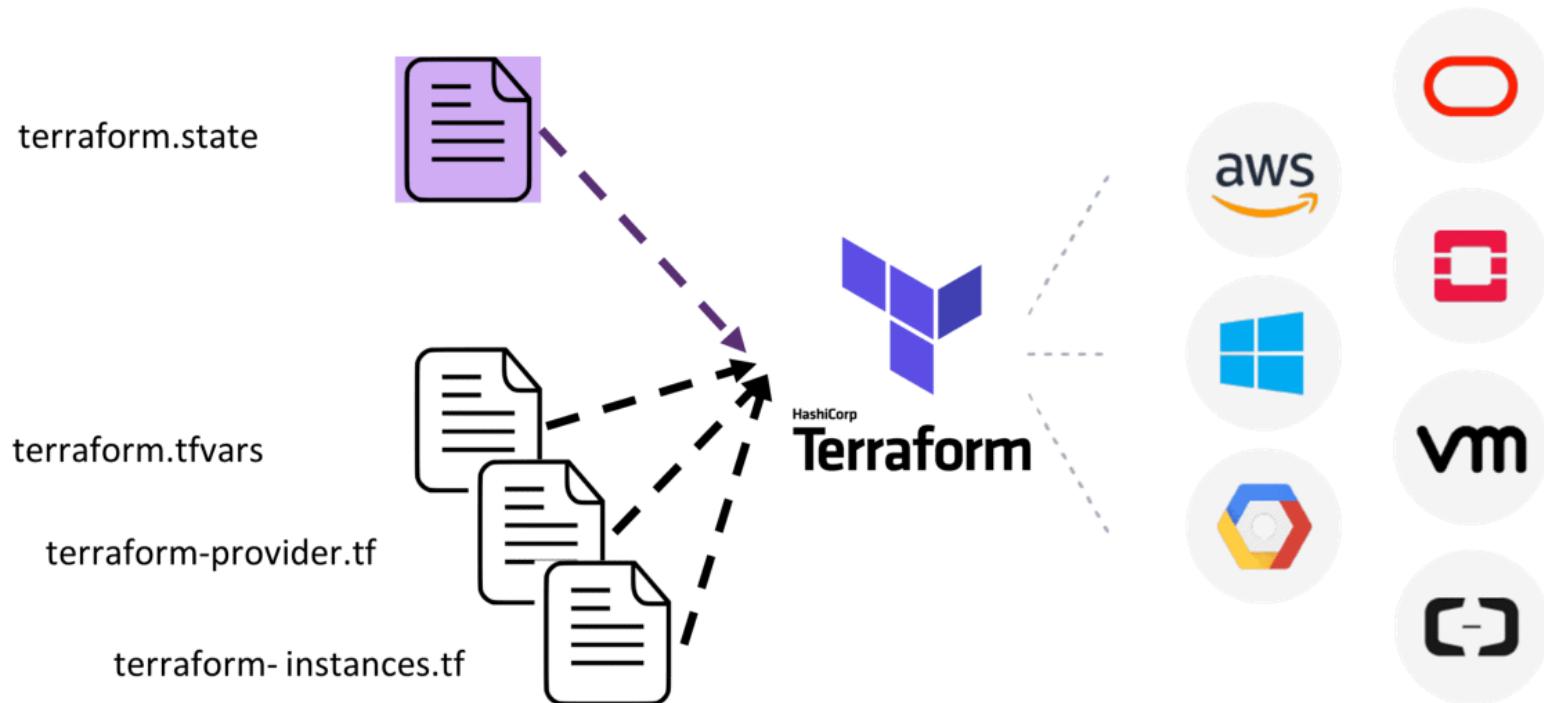
Terraform configurations must declare which providers they require so that Terraform can install and use them.



## Terraform Configuration Files

Configuration files are a set of files used to describe infrastructure in Terraform and have the file extensions .tf and .tf.json. Terraform uses a declarative model for defining infrastructure.

Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.



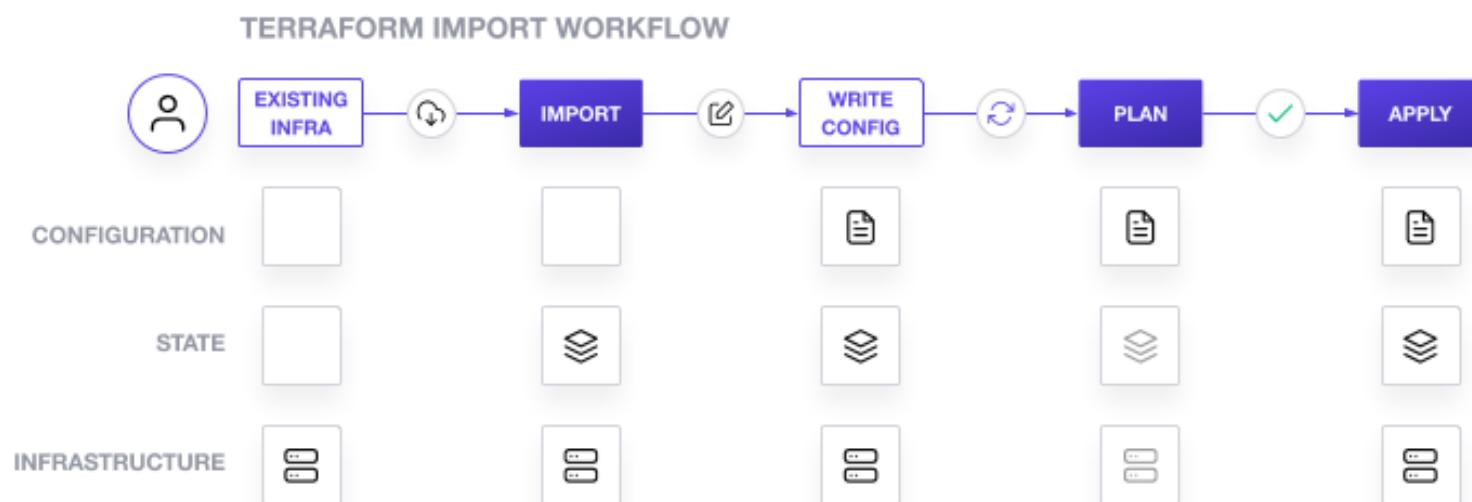
## Getting started using Terraform

To get started building infrastructure resources using Terraform, there are few things that you should take care of. The general steps to deploy a resource(s) in the cloud are:

- Set up a Cloud Account on any cloud provider (AWS, Azure, OCI)
- Install Terraform
- Add a provider – AWS, Azure, OCI, GCP, or others
- Write configuration files
- Initialize Terraform Providers
- PLAN (DRY RUN) using `terraform plan`
- APPLY (Create a Resource) using `terraform apply`
- DESTROY (Delete a Resource) using `terraform destroy`

## Import Existing Infrastructure

- Terraform is one of the great IaC tools with which, you can deploy all your infrastructure's resources.
- In addition to that, you can manage infrastructures from different cloud providers, such as AWS, Google Cloud, etc. But what if you have already created your infrastructure manually?
- Terraform has a really nice feature for importing existing resources, which makes the migration of existing infrastructure into Terraform a lot easier.



# AWS Virtual Machine Using AWS Console | Terraform AWS Deployment Overview

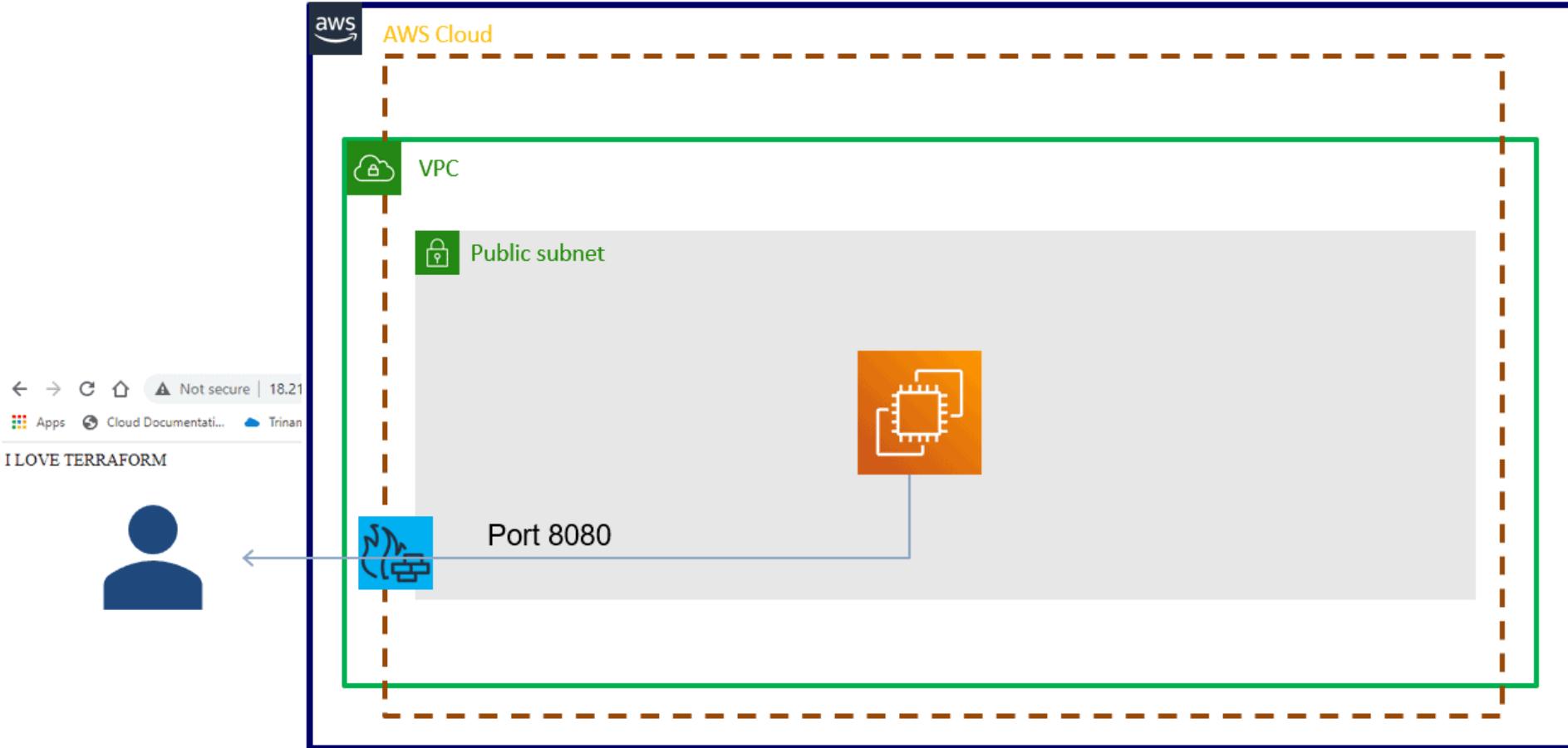
In this blog, we will see how to create an AWS Virtual machine in two different ways, one using the AWS console and the other using terraform.

Pre-requisites:

- AWS Free Tier or Paid account. Check how to create an AWS free tier account
- Terraform should be installed. Check how to install terraform

## Architecture

To give a brief about what demo we will be performing, please refer to the below Architecture

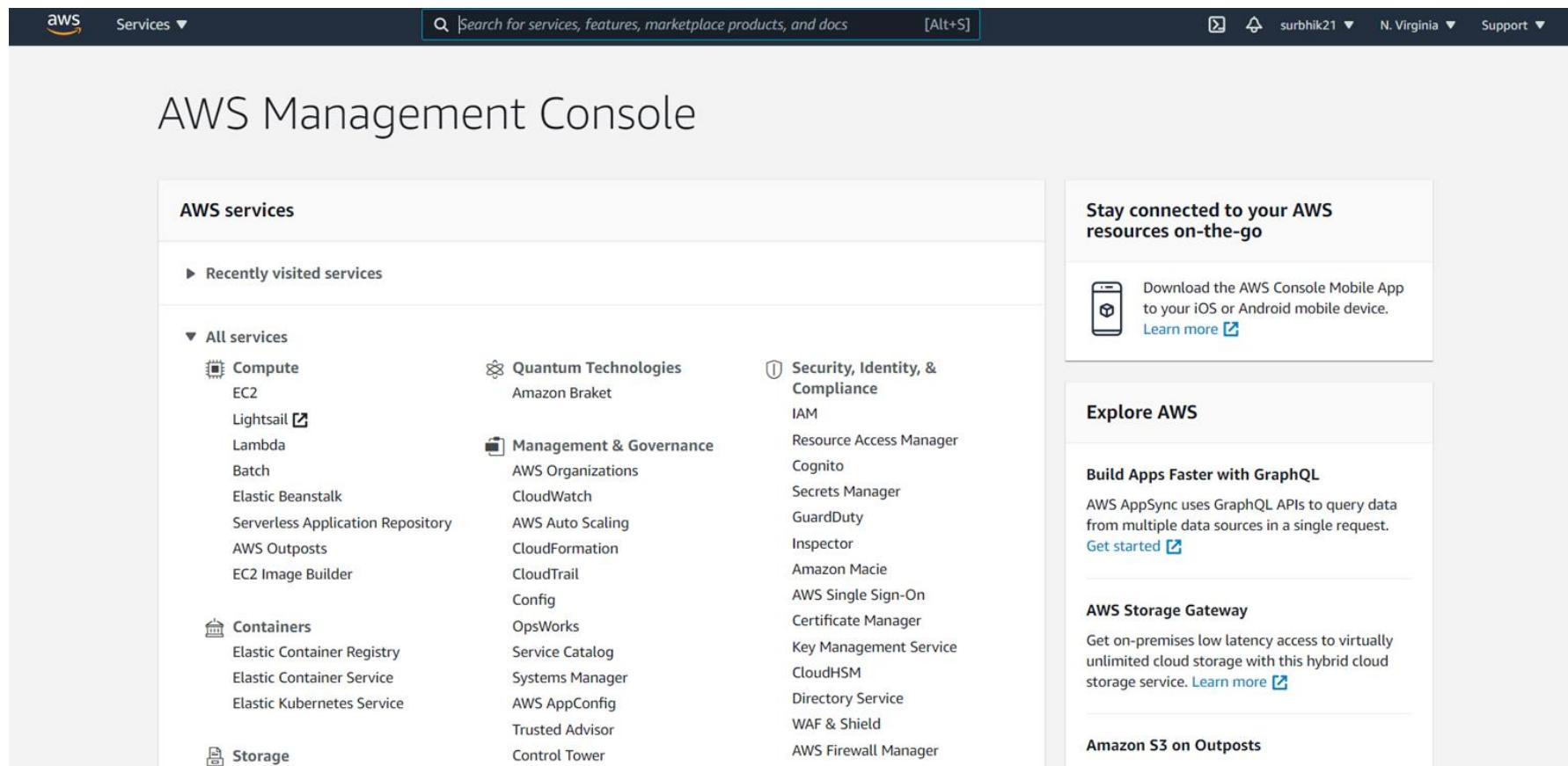


## Create VM in AWS Using Terraform

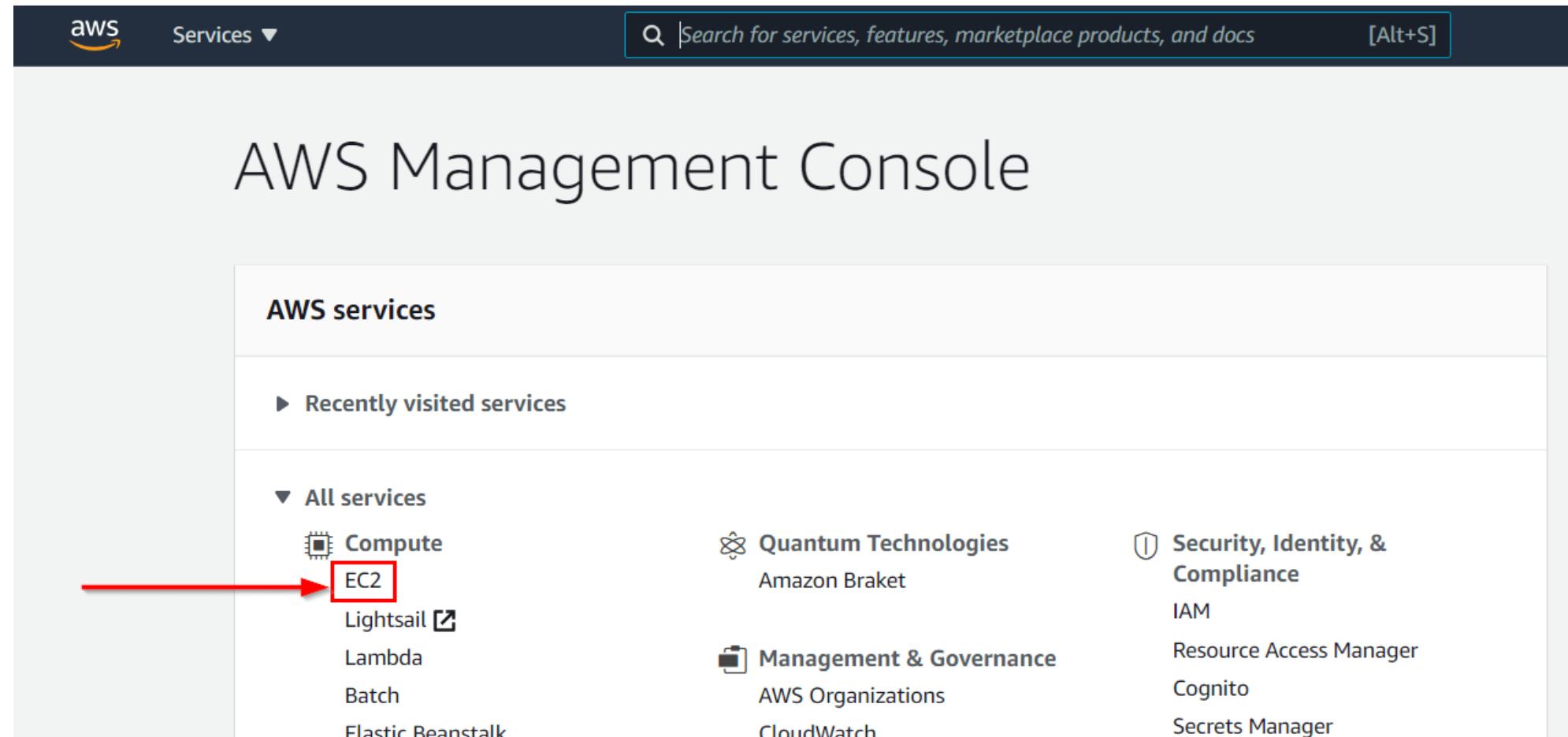
There are a series of steps that you need to follow to create an AWS Virtual Machine and deploy a webpage on top of it.

Now, below are the steps that you can follow to create the above-shown architecture:

- Log in to your AWS account.



- Go to EC2 from the navigation menu present under Compute service.



- Go to Instances from the navigation menu on the left and click on Launch instances.

The screenshot shows the AWS EC2 Instances page. On the left sidebar, under the 'Instances' section, the 'Instances New' link is highlighted with a red box and a circled '1'. At the top right of the main content area, the 'Actions' dropdown is highlighted with a red box and a circled '2', and the 'Launch instances' button is also highlighted with a red box. A red arrow points from the circled '1' to the 'Instances New' link, and another red arrow points from the circled '2' to the 'Launch instances' button.

Instance ID	Name	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
i-084bb0def65d50ced	-	Stopped	t2.micro	-	No alarms +	us-east-1a	-
i-023bb3454f8050b50	-	Stopped	t2.micro	-	No alarms +	us-east-1e	-
i-0f3917eefa7a99e23	-	Running	t2.micro	2/2 checks ...	No alarms +	us-east-1b	ec2-18-206-145-
Cloned_EC2	i-0fb459c58869f7fc	Stopped	t2.micro	-	No alarms +	us-east-1b	-
-	i-0df4f22e3f288ca3d	Stopped	t2.micro	-	No alarms +	us-east-1b	-
Webserver	i-04983c78840fa0bc5	Stopped	t2.micro	-	No alarms +	us-east-1b	-
ProdInstance	i-06a2207a0fec9215b	Stopped	t2.micro	-	No alarms +	us-east-1b	-

- Search for Ubuntu and select Ubuntu Server 20.04 LTS (HVM).

Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

1. Choose AMI    2. Choose Instance Type    3. Configure Instance    4. Add Storage    5. Add Tags    6. Configure Security Group    7. Review

**Quick Start (8)**

My AMIs (0)  
AWS Marketplace (559)  
Community AMIs (35992)

Free tier only ⓘ

Search for services, features, marketplace products, and docs [Alt+S]

Cancel and Exit

Search by Systems Manager parameter

1 to 8 of 8 AMIs

**Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-0885b1f6bd170450c (64-bit x86) / ami-054e49cb26c2fd312 (64-bit Arm)**

Ubuntu Server 20.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).  
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

**Select**

64-bit (x86)  
 64-bit (Arm)

**Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-00ddb0e5626798373 (64-bit x86) / ami-074db80f0dc9b5f40 (64-bit Arm)**

Ubuntu Server 18.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).  
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

**Select**

64-bit (x86)  
 64-bit (Arm)

**Deep Learning AMI (Ubuntu 18.04) Version 39.0 - ami-03e0fdb8c9d235984**

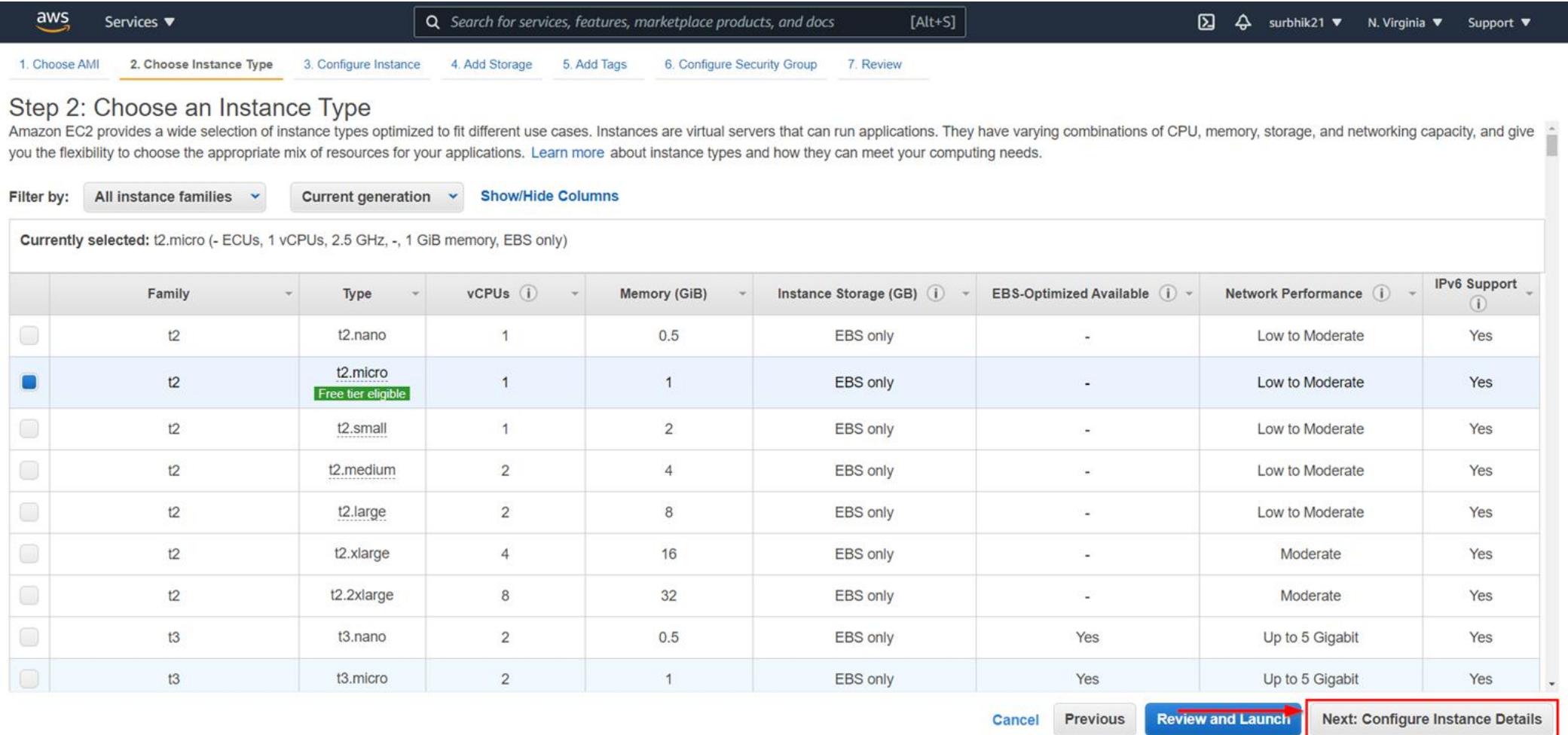
MXNet-1.8.0 & 1.7.0, TensorFlow-2.3.1, 2.1.2 & 1.15.4, PyTorch-1.4.0 & 1.7.1, Neuron, & others. NVIDIA CUDA, cuDNN, NCCL, Intel MKL-DNN, Docker, NVIDIA-Docker & EFA support. For fully managed experience, check: <https://aws.amazon.com/sagemaker>

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

**Select**

64-bit (x86)

- Select Instance type as t2.micro and click on Next: Configure Instance Details.



The screenshot shows the AWS EC2 instance creation wizard at Step 2: Choose an Instance Type. The user has selected the t2.micro instance type, which is highlighted with a green border and labeled "Free tier eligible". The table displays various instance types across different families (t2, t3) with their respective specifications like vCPUs, Memory, and Network Performance.

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	t2	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.xlarge	4	16	EBS only	-	Moderate	Yes
<input type="checkbox"/>	t2	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
<input type="checkbox"/>	t3	t3.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit	Yes
<input type="checkbox"/>	t3	t3.micro	2	1	EBS only	Yes	Up to 5 Gigabit	Yes

At the bottom right, the "Review and Launch" button is highlighted with a red box, indicating the next step in the process.

- On Configure Instance Details page, scroll down and add the following code in User data to provision the web page after the instance is provisioned. Click on 6. Configure Security Group to skip Storage and Tags.

Step 3: Configure Instance Details

Tenancy: Shared - Run a shared hardware instance  
Additional charges will apply for dedicated tenancy.

Elastic Inference: Add an Elastic Inference accelerator  
Additional charges apply.

Credit specification: Unlimited  
Additional charges may apply

File systems: Add file system, Create new file system

**Advanced Details**

Enclave: Enable

Metadata accessible: Enabled

Metadata version: V1 and V2 (token optional)

Metadata token response hop limit: 1

User data:

```
#!/bin/bash
echo "I LOVE TERRAFORM" > index.html
nohup busybox httpd -f -p 8080 &
```

Cancel Previous Review and Launch Next: Add Storage

- Add Custom TCP protocol to open 8080 port accessible from the internet and click on Review and Launch.

**Step 6: Configure Security Group**

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group:  Create a new security group  Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source	Description
Custom TCP F	TCP	8080	Custom <input type="button" value="0.0.0.0/0"/>	e.g. SSH for Admin Desktop <input type="button" value="X"/>

**Add Rule**

**Warning**  
You will not be able to connect to this instance as the AMI requires port(s) 22 to be open in order to have access. Your current security group doesn't have port(s) 22 open.

**Warning**  
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

**Review and Launch**

- Review all the information and click on Launch.

**Step 7: Review Instance Launch**

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

**AMI Details**

**Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-0885b1f6bd170450c**  
Free tier eligible Ubuntu Server 20.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).  
Root Device Type: ebs Virtualization type: hvm

**Instance Type**

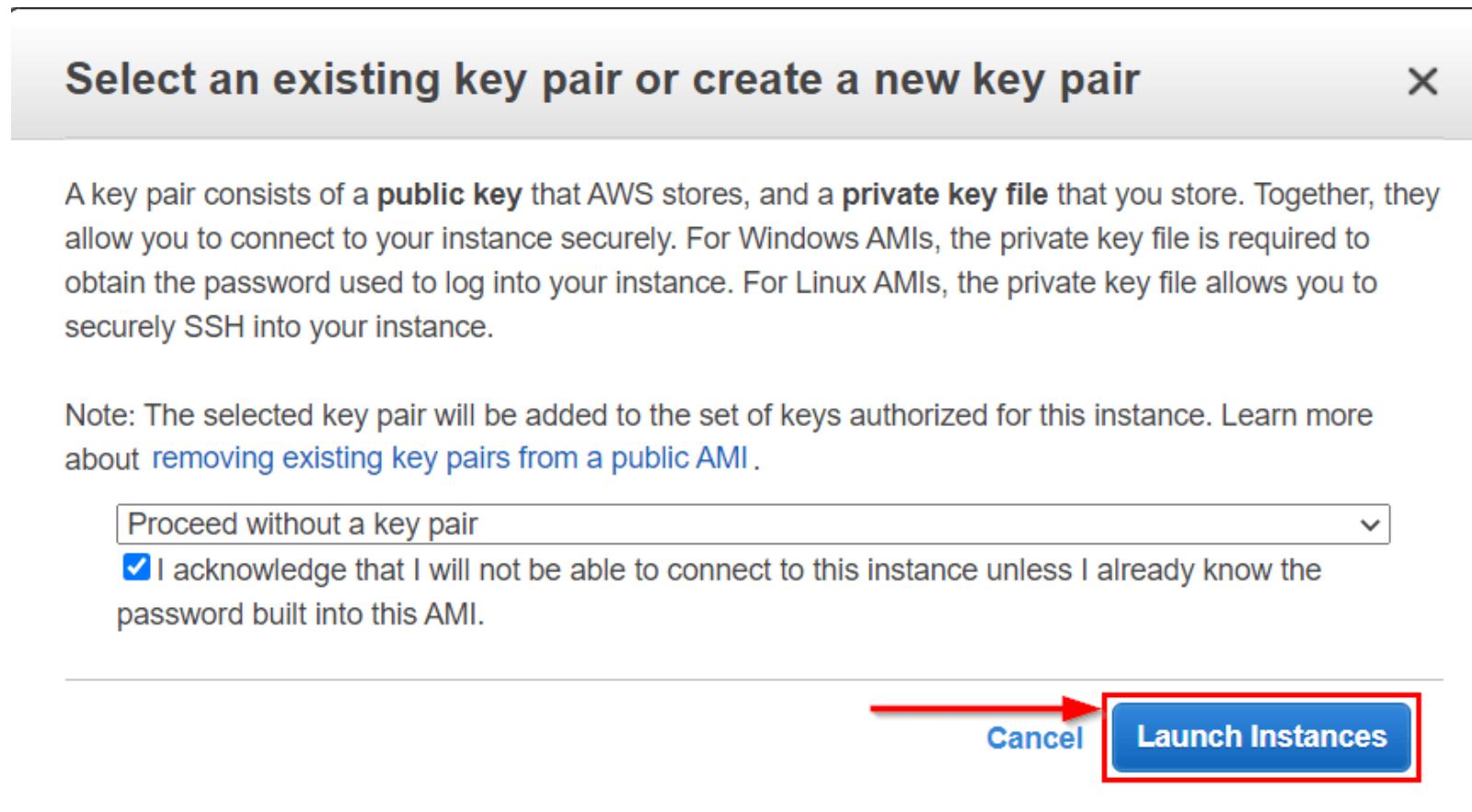
Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

**Security Groups**

Security group name: launch-wizard-36  
Description: launch-wizard-36 created 2021-01-29T17:05:56.078+05:30

**Launch** Previous

- When asked to select a key pair, select Proceed without a key pair and click on Launch Instances.



- Click on View Instance to check the status of the Instance.

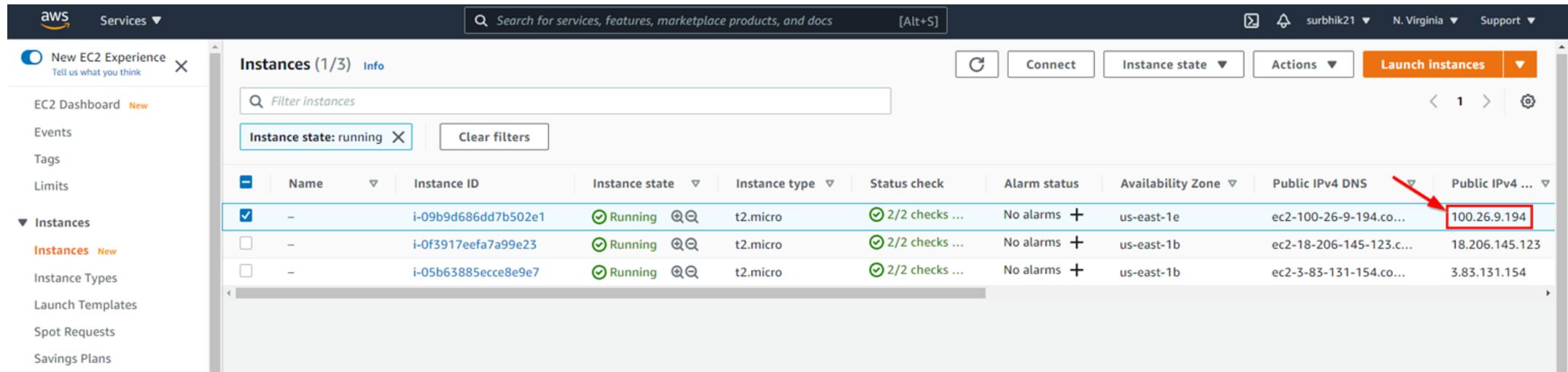
Screenshot of the AWS CloudWatch Launch Status page for an instance launch.

The page shows:

- A success message: "Your instances are now launching" with a link to "View launch log".
- An information message: "Get notified of estimated charges" with a link to "Create billing alerts".
- A section titled "How to connect to your instances" with instructions and links to "View Instances" and "Find out how to connect to your instances".
- A "Helpful resources" section with links to "How to connect to your Linux instance", "Amazon EC2: User Guide", "Learn about AWS Free Usage Tier", and "Amazon EC2: Discussion Forum".
- A list of additional actions you can take while your instances are launching.

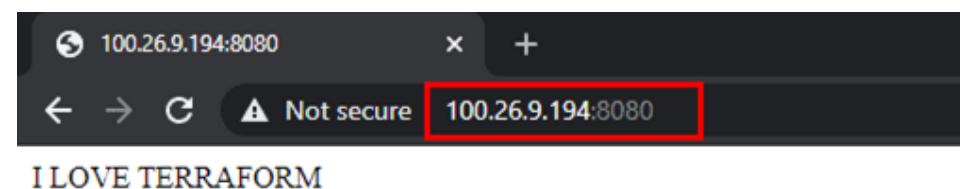
A red arrow points to the "View Instances" button at the bottom right of the page.

- Once the instance is up and running, copy the Public IP.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
<input checked="" type="checkbox"/>	i-09b9d686dd7b502e1	<span>Running</span>	t2.micro	<span>2/2 checks ...</span>	No alarms	+ us-east-1e	ec2-100-26-9-194.co...	100.26.9.194
<input type="checkbox"/>	i-0f3917eefa7a99e23	<span>Running</span>	t2.micro	<span>2/2 checks ...</span>	No alarms	+ us-east-1b	ec2-18-206-145-123.c...	18.206.145.123
<input type="checkbox"/>	i-05b63885ecce8e9e7	<span>Running</span>	t2.micro	<span>2/2 checks ...</span>	No alarms	+ us-east-1b	ec2-3-83-131-154.co...	3.83.131.154

- Paste the IP in your browser followed by port 8080 to access the created webpage.



## Terraform AWS Deployment

We will use the below code to provision AWS Virtual Machine in our account. You can download the code by clicking [here](#).

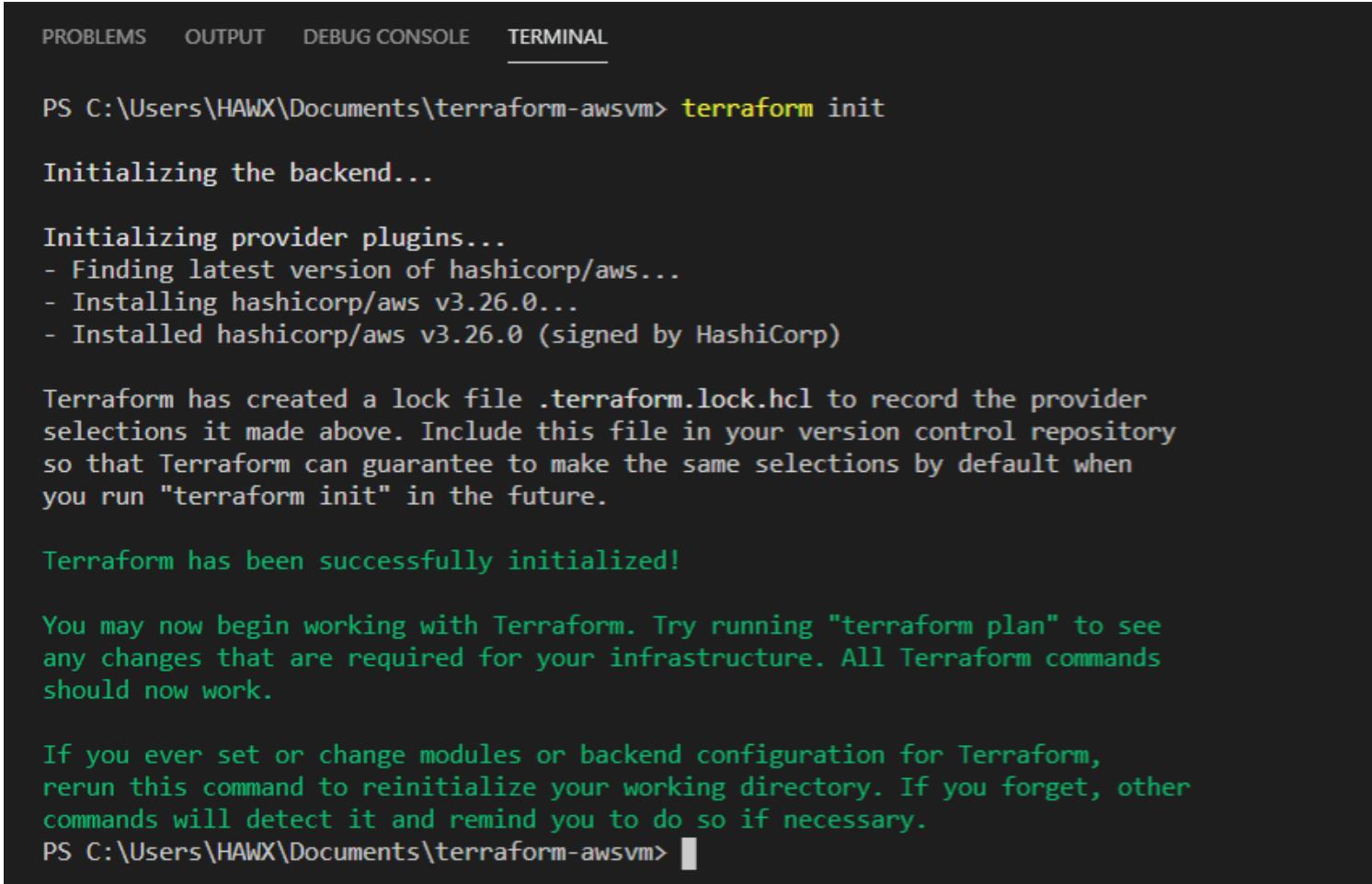
```
provider "aws" {  
    region = "us-east-1"  
    access_key = "<YOUR ACCESS KEY HERE>"  
    secret_key = "<YOUR SECRET KEY HERE>"  
}  
  
resource "aws_instance" "tfvm" {  
    ami = "ami-0885b1f6bd170450c"  
    instance_type = "t2.micro"  
    vpc_security_group_ids = [ aws_security_group.websg.id ]  
    user_data = <<-EOF  
#!/bin/bash  
    echo "I LOVE TERRAFORM" > index.html  
    nohup busybox httpd -f -p 8080 &  
    EOF
```

```
tags = {
    Name = "WEB-demo"
}
}

resource "aws_security_group" "websg" {
    name = "web-sg01"
    ingress {
        protocol = "tcp"
        from_port = 8080
        to_port = 8080
        cidr_blocks = [ "0.0.0.0/0" ]
    }
}
output "instance_ips" {
    value = aws_instance.tfvm.public_ip
}
```

## Steps to create the VM from the code:

- Use terraform init command in terminal to initialize terraform and download the configuration files



The screenshot shows a terminal window with the following text output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\HAWX\Documents\terraform-awsvm> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.26.0...
- Installed hashicorp/aws v3.26.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

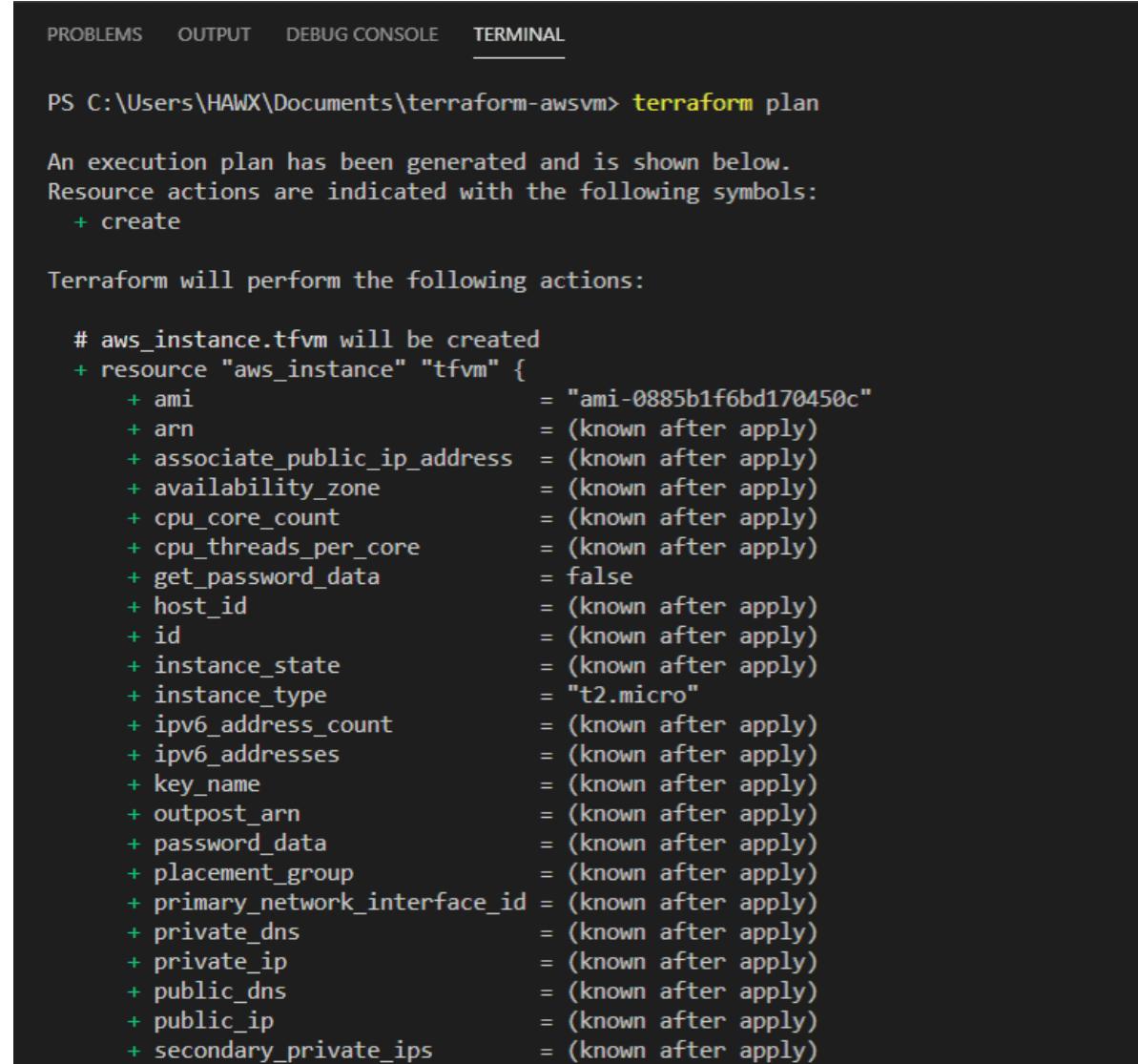
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

PS C:\Users\HAWX\Documents\terraform-awsvm>
```

- Now let's check for any human error in our script by using terraform plan command.



The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\HAWX\Documents\terraform-awsvm> terraform plan

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_instance.tfvm will be created
+ resource "aws_instance" "tfvm" {
    + ami                               = "ami-0885b1f6bd170450c"
    + arn                             = (known after apply)
    + associate_public_ip_address     = (known after apply)
    + availability_zone                = (known after apply)
    + cpu_core_count                  = (known after apply)
    + cpu_threads_per_core           = (known after apply)
    + get_password_data              = false
    + host_id                         = (known after apply)
    + id                             = (known after apply)
    + instance_state                 = (known after apply)
    + instance_type                  = "t2.micro"
    + ipv6_address_count             = (known after apply)
    + ipv6_addresses                 = (known after apply)
    + key_name                       = (known after apply)
    + outpost_arn                    = (known after apply)
    + password_data                  = (known after apply)
    + placement_group                = (known after apply)
    + primary_network_interface_id   = (known after apply)
    + private_dns                     = (known after apply)
    + private_ip                      = (known after apply)
    + public_dns                      = (known after apply)
    + public_ip                       = (known after apply)
    + secondary_private_ips          = (known after apply)
```

In the end, it will also show us the number of resources that will be added to our AWS account.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

+ prefix_list_ids  = []
+ protocol        = "tcp"
+ security_groups = []
+ self             = false
+ to_port          = 8080
},
]
+ name            = "web-sg01"
+ owner_id        = (known after apply)
+ revoke_rules_on_delete = false
+ vpc_id          = (known after apply)
}

Plan: 2 to add, 0 to change, 0 to destroy.

-----
Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.

PS C:\Users\HAWX\Documents\terraform-awsvm>
```

- Now we will apply our code using terraform apply -auto-approve command. It will provide us the IP of our created VM.

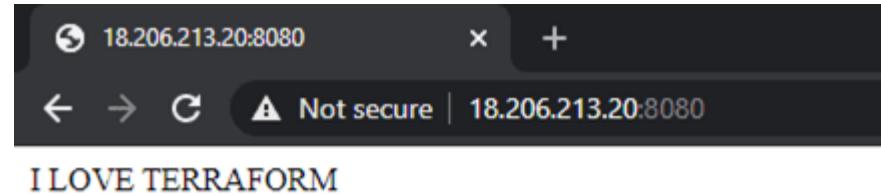
```
PS C:\Users\HAWX\Documents\terraform-awsvm> terraform apply -auto-approve
aws_security_group.websg: Creating...
aws_security_group.websg: Creation complete after 8s [id=sg-019465038b95617b1]
aws_instance.tfvm: Creating...
aws_instance.tfvm: Still creating... [10s elapsed]
aws_instance.tfvm: Still creating... [20s elapsed]
aws_instance.tfvm: Still creating... [30s elapsed]
aws_instance.tfvm: Still creating... [40s elapsed]
aws_instance.tfvm: Creation complete after 42s [id=i-04eec7b6688fa5a04]
```

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Outputs:

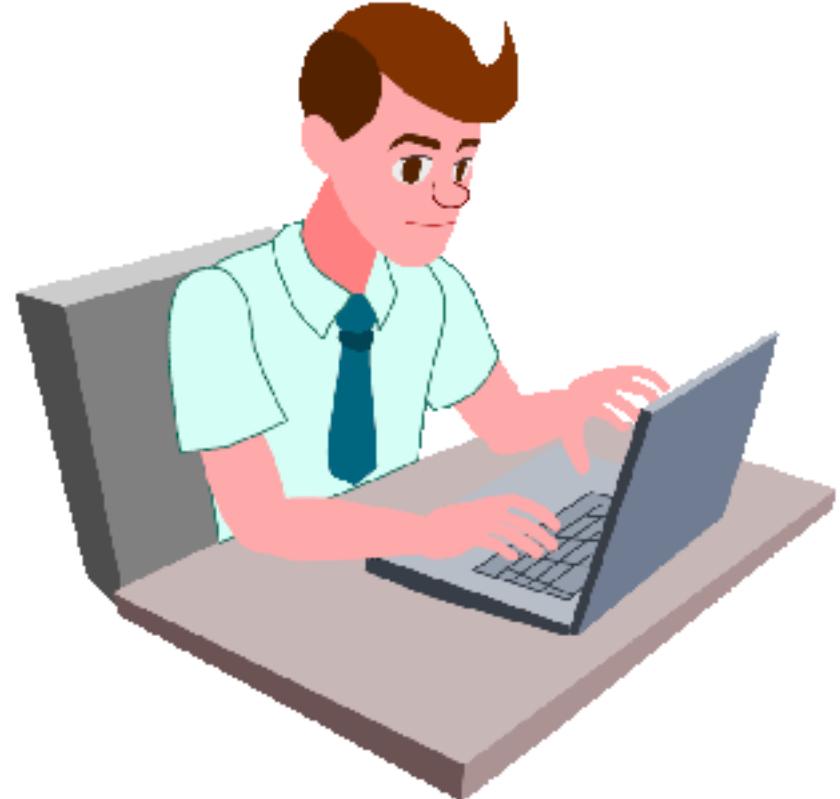
```
instance_ips = "18.206.213.20"
```

Now, we can check if the Webpage has been created or not. Just copy the IP and check on 8080 port regarding the same.



You can also check if the EC2 instance is created in the AWS by going to the EC2 dashboard.

A screenshot of the AWS EC2 Instances dashboard. The left sidebar shows navigation options like New EC2 Experience, EC2 Dashboard, Events, Tags, Limits, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, and Reserved Instances. The main content area shows a table titled "Instances (2)". The table has columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. Two instances are listed: one with Instance ID i-0f3917eefa7a99e23 and another with Instance ID i-04eec7b6688fa5a04, which is highlighted with a red box and a red arrow pointing to it. Both instances are in the "Running" state, t2.micro type, and have 2/2 checks passing. They are located in us-east-1b and us-east-1c respectively, with public IP addresses ec2-18-206-145-12 and ec2-18-206-213-20.



# "Complete Lab 5"

## 6. Infrastructure Deployment using Terraform

# Provision a CockroachDB Cloud Cluster with Terraform

- Terraform is an infrastructure-as-code provisioning tool that uses configuration files to define application and network resources.
- You can provision CockroachDB Cloud clusters and cluster resources by using the CockroachDB Cloud Terraform provider in your Terraform configuration files.
- This shows you how to provision a CockroachDB Cloud cluster using the CockroachDB Cloud Terraform provider.

## CockroachDB Serverless

### Before you begin

#### Before you start this tutorial, you must

- Install Terraform.
- Install the wget command line utility.
- Create a service account and API key in the CockroachDB Cloud Console, and assign admin privilege or Cluster Creator / Cluster Admin role at the organization scope. Refer to: Service Accounts

## Create the Terraform configuration files

- In a terminal create a new directory and use wget to download the CockroachDB Serverless main.tf example file:

```
wget https://raw.githubusercontent.com/cockroachdb/terraform-provider-cockroach/main/examples/workflows/cockroach\_serverless\_cluster/main.tf
```

- In a text editor create a new file terraform.tfvars with the following settings:

```
cluster_name = "{cluster name}"
```

```
sql_user_name = "{SQL user name}"
```

```
sql_user_password = "{SQL user password}"
```

- Create an environment variable named COCKROACH\_API\_KEY. Copy the API key from the CockroachDB Cloud console and create the COCKROACH\_API\_KEY environment variable:

```
export COCKROACH_API_KEY={API key}
```

## Provision the cluster

- Initialize the provider:  
`terraform init --upgrade`
- Create the Terraform plan. This shows the actions the provider will take, but won't perform them:  
`terraform plan`
- Create the cluster:  
`terraform apply`

## Delete the cluster

- If you want to delete the cluster, run the following command:  
`terraform destroy`
- Enter yes when prompted to delete the cluster.

# Provision a CockroachDB Cloud Cluster with Terraform

- Terraform is an infrastructure-as-code provisioning tool that uses configuration files to define application and network resources.
- You can provision CockroachDB Cloud clusters and cluster resources by using the CockroachDB Cloud Terraform provider in your Terraform configuration files.
- This tutorial shows you how to provision a CockroachDB Cloud cluster using the CockroachDB Cloud Terraform provider.

## CockroachDB Dedicated

### Before you begin

Before you start this tutorial, you must

- Install Terraform.
- Install the wget command line utility.
- Create a service account and API key in the CockroachDB Cloud Console, and assign admin privilege or Cluster Creator / Cluster Admin role at the organization scope. Refer to: [Service Accounts](#)

## Create the Terraform configuration files

- In a terminal create a new directory and use wget to download the CockroachDB Serverless main.tf example file:

```
wget https://raw.githubusercontent.com/cockroachdb/terraform-provider-cockroach/main/examples/workflows/cockroach\_dedicated\_cluster/main.tf
```

- In a text editor create a new file terraform.tfvars with the following settings:

```
cluster_name = "{cluster name}"
sql_user_name = "{SQL user name}"
sql_user_password = "{SQL user password}"
cloud_provider = "{cloud provider}"
cloud_provider_regions = ["{cloud provider region}"]
cluster_node_count = {number of nodes}
storage_gib = {storage in GiB}
machine_type = "{cloud provider machine type}"
allow_list_name = "{allow list name}"
cidr_ip = "{allow list CIDR IP}"
cidr_mask = {allow list CIDR mask}
```

- Create an environment variable named COCKROACH\_API\_KEY.
- Copy the API key from the CockroachDB Cloud console and create the COCKROACH\_API\_KEY environment variable:

```
export COCKROACH_API_KEY={API key}
```

## Provision the cluster

- **Initialize the provider:**

`terraform init -upgrade`

This reads the `main.tf` configuration file and uses the `terraform.tfvars` file for settings specific to your cluster. The `-upgrade` flag ensures you are using the latest version of the provider.

- **Create the Terraform plan. This shows the actions the provider will take, but won't perform them:**

`terraform plan`

- **Create the cluster:**

`terraform apply`

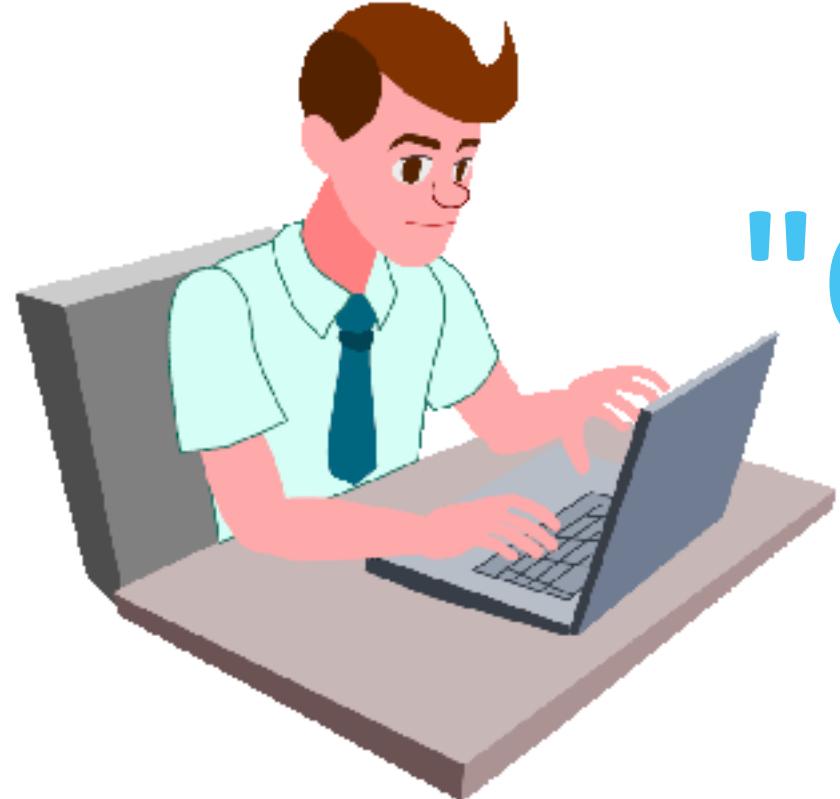
Enter yes when prompted to apply the plan and create the cluster.

## Get information about your cluster

- The terraform show command shows detailed information of your cluster resources.
- `terraform show`

## Delete the cluster

- If you want to delete the cluster, run the following command:
- `terraform destroy`



# "Complete Lab 6"

# 7. Merging

# Merging branches

In Git, merging two (or more!) branches is the act of making their personal history meet each other. When they meet, two things can happen:

- Files in their tip commit are different, so some conflict will rise
- Files do not conflict
- Commits of the target branch are directly behind commits of the branch we are merging, so a fast-forward will happen

# Merging branches

- Let's give it a try.
- We can try to merge the melons branch into the master one; to do so, you have to check out the target branch, master in this case, and then fire a git merge <branch name> command; as I'm already on the master branch, I go straight with the merge command:

```
[1] ~/grocery (master)
$ git merge melons
Auto-merging shoppingList.txt
CONFLICT (content): Merge conflict in shoppingList.txt
Automatic merge failed; fix conflicts and then commit the result.
```

# Merging branches

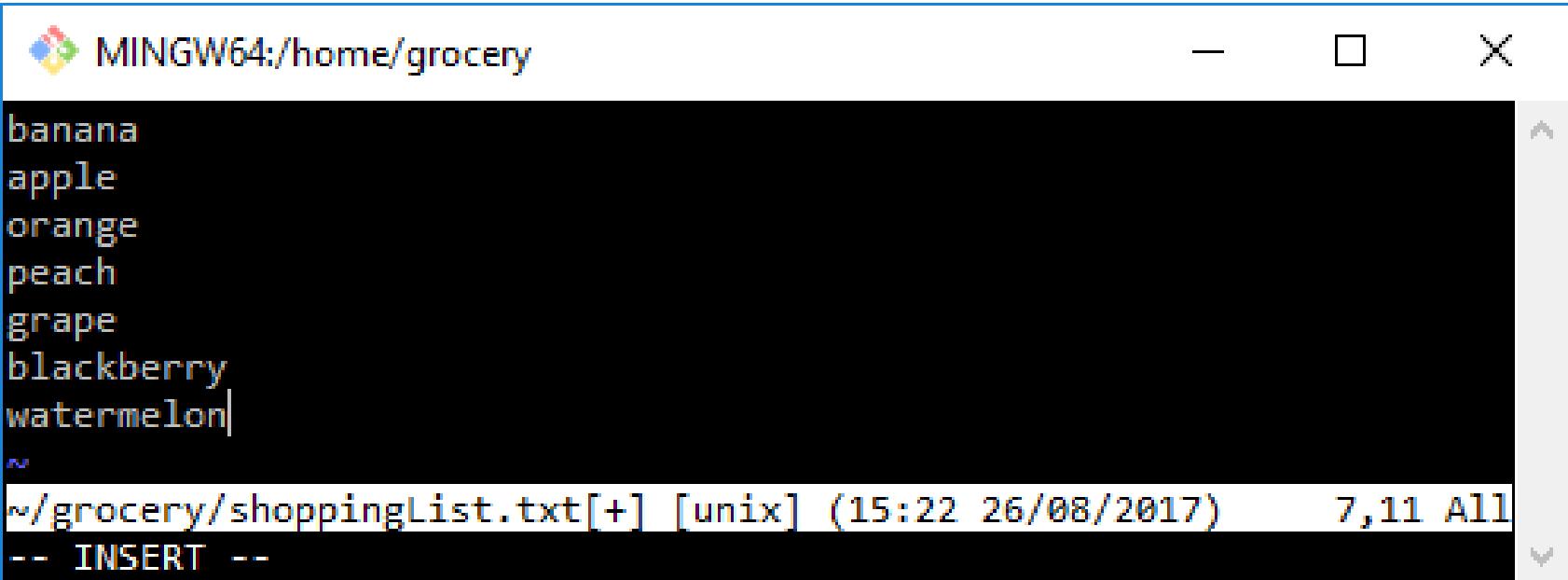
- See the conflict with git diff:

```
[2] ~/grocery (master|MERGING)
$ git diff
diff --cc shoppingList.txt
index 862debc,7786024..0000000
--- a/shoppingList.txt
+++ b/shoppingList.txt
@@@ -1,5 -1,5 +1,10 @@@
    banana
    apple
    orange
+===== HEAD
    +peach
    - grape

    ++grape
+=====
    + blackberry
    + watermelon
+=====
    >>>>> melons
```

# Merging branches

- I will edit the file enqueueing blackberry and watermelon after peach and grape, as per the following screenshot:



The screenshot shows a terminal window titled "MINGW64:/home/grocery". The window contains a list of items in a file named "shoppingList.txt". The items listed are banana, apple, orange, peach, grape, blackberry, and watermelon. The cursor is positioned at the end of the word "watermelon". Below the terminal window, the file path is shown as "/grocery/shoppingList.txt[+]" and the status bar indicates the file is in unix mode, was modified at 15:22 on 26/08/2017, and has 7,11 lines.

```
banana
apple
orange
peach
grape
blackberry
watermelon|
~
```

~/grocery/shoppingList.txt[+] [unix] (15:22 26/08/2017) 7,11 All

-- INSERT --

# Merging branches

- After saving the file, add it to the staging area and then commit:

[3] ~/grocery (master|MERGING)

\$ git add shoppingList.txt

[4] ~/grocery (master|MERGING)

\$ git commit -m "Merged melons branch into master"

[master e18a921] Merged melons branch into master

# Merging branches

- Now take a look at the log:

```
[5] ~/grocery (master)
$ git log --oneline --graph --decorate --all
*   e18a921 (HEAD -> master) Merged melons branch into master
|\ \
| * a8c6219 (melons) Add a watermelon
| * ef6c382 (berries) Add a blackberry
* | 6409527 Add a grape
* | 603b9d1 Add a peach
| /
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Merging branches

- Suggestion: look at the merge commit with git cat-file -p:

```
[6] ~/grocery (master)
$ git cat-file -p HEAD
tree 2916dd995ee356351c9b49a5071051575c070e5f
parent 6409527a1f06d0bbe680d461666ef8b137ac7135
parent a8c62190fb1c54d1034db78a87562733a6e3629c
author Ferdinando Santacroce <ferdinando.santacroce@gmail.com> 1503754221 +0200
committer Ferdinando Santacroce <ferdinando.santacroce@gmail.com> 1503754221 +0200

Merged melons branch into master
```

# Merging branches

## Fast forwarding

- A merge not always generates a new commit; to test this case, try to merge the melons branch into a berries one:

```
[7] ~/grocery (master)
$ git checkout berries
Switched to branch 'berries'

[8] ~/grocery (berries)
$ git merge melons
Updating ef6c382..a8c6219
Fast-forward
  shoppingList.txt | 1 +
  1 file changed, 1 insertion(+)

[9] ~/grocery (berries)
$ git log --oneline --graph --decorate --all
*   e18a921 (master) Merged melons branch into master
|\ 
| * a8c6219 (HEAD -> berries, melons) Add a watermelon
| * ef6c382 Add a blackberry
* | 6409527 Add a grape
* | 603b9d1 Add a peach
|/
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Merging branches

- Move back the berries branch where it was using git reset:

```
[10] ~/grocery (berries)
$ git reset --hard HEAD^
HEAD is now at ef6c382 Add a blackberry

[11] ~/grocery (berries)
$ git log --oneline --graph --decorate --all
*   e18a921 (master) Merged melons branch into master
|\ \
| * a8c6219 (melons) Add a watermelon
| * ef6c382 (HEAD -> berries) Add a blackberry
* | 6409527 Add a grape
* | 603b9d1 Add a peach
| /
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Merging branches

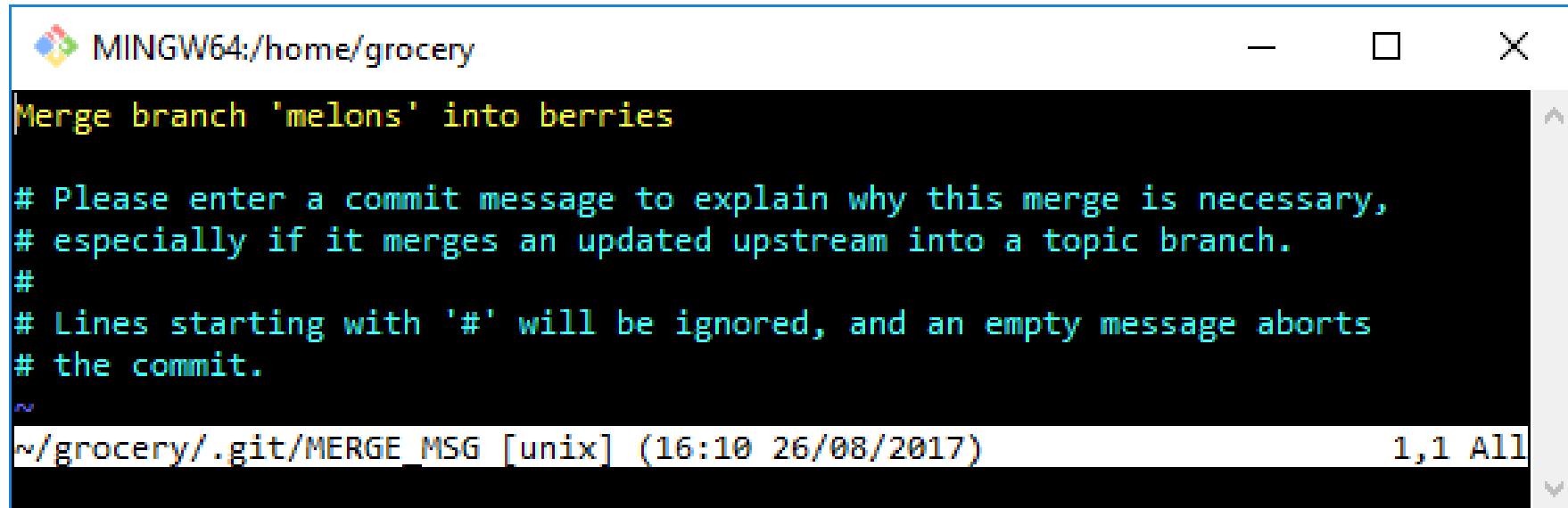
- We have just undone a merge, did you realize it?
- Okay, now do the merge again with the --no-ff option:

```
[12] ~/grocery (berries)  
$ git merge --no-ff melons
```



# Merging branches

- Git will now open your default editor to allow you to specify a commit message, as shown in the following screenshot:



The screenshot shows a terminal window titled "MINGW64:/home/grocery". The window contains the following text:

```
Merge branch 'melons' into berries

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~

~/grocery/.git/MERGE_MSG [unix] (16:10 26/08/2017) 1,1 All
```

# Merging branches

- Accept the default message, save and exit:

[13] ~/grocery (berries)

Merge made by the 'recursive' strategy.--all  
shoppingList.txt | 1 +  
1 file changed, 1 insertion(+)

# Merging branches

- Now a git log:

```
[14] ~/grocery (berries)
$ git log --oneline --graph --decorate --all
*   cb912b2 (HEAD -> berries) Merge branch 'melons' into berries
|\ \
| | *   e18a921 (master) Merged melons branch into master
| | |
| | /|
| | /|
| * | a8c6219 (melons) Add a watermelon
| // 
* | ef6c382 Add a blackberry
| * 6409527 Add a grape
| * 603b9d1 Add a peach
| /
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Merging branches

- We are done with these experiments; anyway, I want to undo this merge, because I want to keep the repository as simple as possible to allow you to better understand the exercise we do together; go with a git reset --hard HEAD^:

```
[15] ~/grocery (berries)
$ git reset --hard HEAD^
HEAD is now at ef6c382 Add a blackberry

[16] ~/grocery (berries)
$ git log --oneline --graph --decorate --all
*   e18a921 (master) Merged melons branch into master
|\ \
| * a8c6219 (melons) Add a watermelon
| * ef6c382 (HEAD -> berries) Add a blackberry
* | 6409527 Add a grape
* | 603b9d1 Add a peach
|/
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Merging branches

Okay, now undo even the past merge we did on the master branch:

```
[17] ~/grocery (master)
$ git reset --hard HEAD^
HEAD is now at 6409527 Add a grape

[18] ~/grocery (master)
$ git log --oneline --graph --decorate --all
* 6409527 (HEAD -> master) Add a grape
* 603b9d1 Add a peach
| * a8c6219 (melons) Add a watermelon
| * ef6c382 (berries) Add a blackberry
| /
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Cherry picking

- Let's play with it a little bit.
- Assume you want to pick the blackberry from the berries branch, and then apply it into the master branch; this is the way:

```
[1] ~/grocery (master)
$ git cherry-pick ef6c382
error: could not apply ef6c382... Add a blackberry
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
```

# Cherry picking

- Okay, the cherry pick raised a conflict, of course:

```
[2] ~/grocery (master|CHERRY-PICKING)
$ git diff
diff --cc shoppingList.txt
index 862debc,b05b25f..0000000
--- a/shoppingList.txt
+++ b/shoppingList.txt
@@@ -1,5 -1,4 +1,9 @@@
banana
apple
orange
+===== HEAD
+peach
-grape
++grape
+=====
+ blackberry
+>>>>> ef6c382... Add a blackberry
```

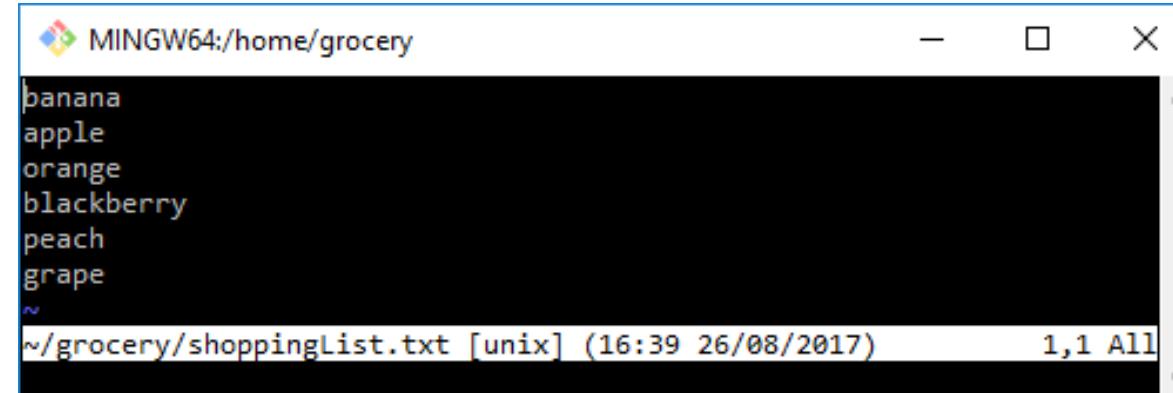
# Cherry picking

- The fourth line of both the shoppingList.txt file versions has been modified with different fruits.
- Resolve the conflict and then add a commit:

```
[3] ~/grocery (master|CHERRY-PICKING)
```

```
$ vi shoppingList.txt
```

- The following is a screenshot of my Vim console, and the files are arranged as I like:



A screenshot of a terminal window titled "MINGW64:/home/grocery". The window contains a list of fruit names: banana, apple, orange, blackberry, peach, and grape. The cursor is positioned at the end of the list, indicated by a tilde (~). At the bottom of the window, the status bar shows the path "~/grocery/shoppingList.txt [unix] (16:39 26/08/2017)" and the text "1,1 All".

# Cherry picking

```
[4] ~/grocery (master|CHERRY-PICKING)
$ git add shoppingList.txt

[5] ~/grocery (master|CHERRY-PICKING)
$ git status
On branch master
You are currently cherry-picking commit ef6c382.
(all conflicts fixed: run "git cherry-pick --continue")
(use "git cherry-pick --abort" to cancel the cherry-pick operation)

Changes to be committed:

modified:   shoppingList.txt
```

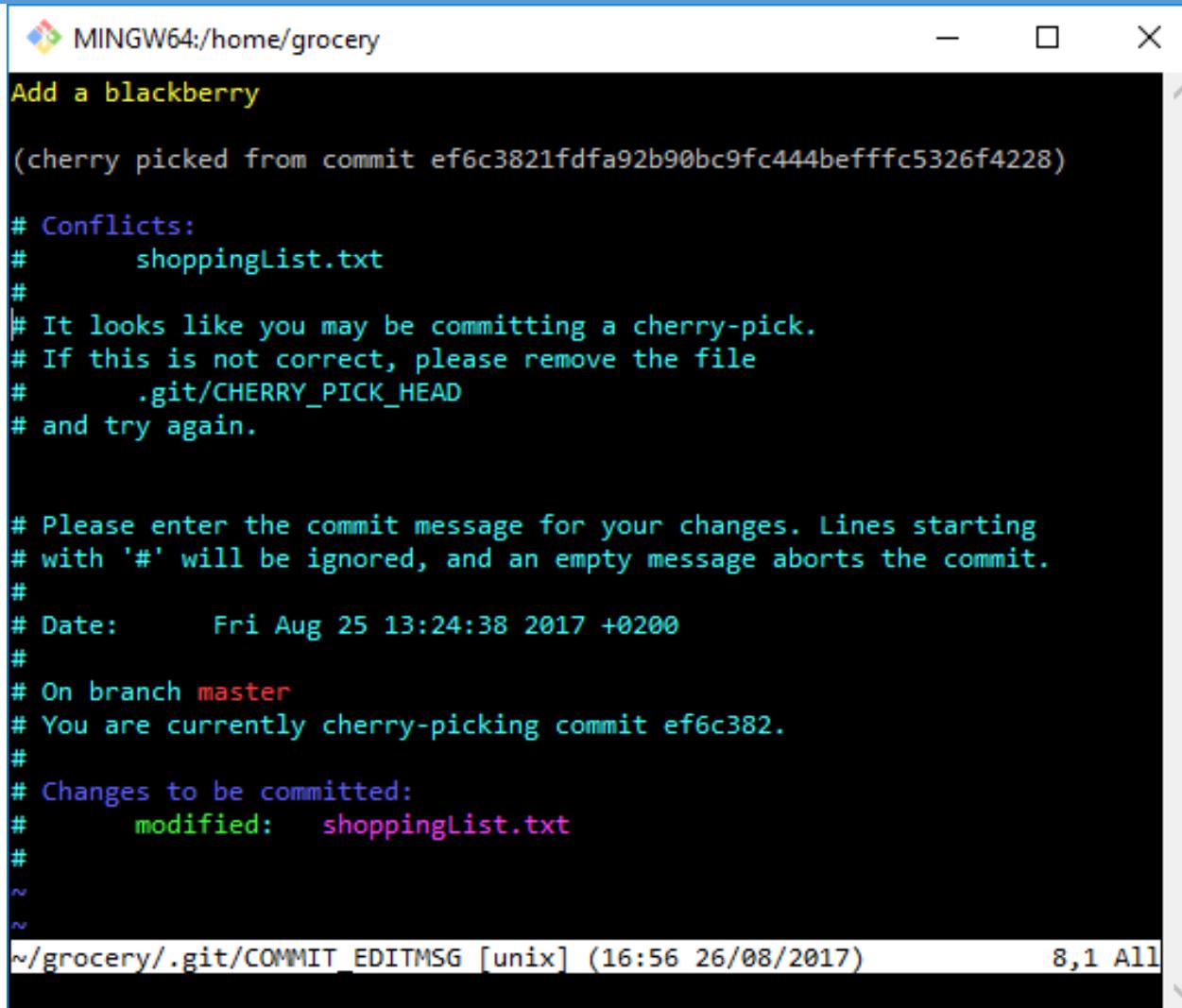
# Cherry picking

- Now go on and commit:

```
[6] ~/grocery (master)
$ git commit -m "Add a cherry-picked blackberry"
On branch master
nothing to commit, working tree clean

[7] ~/grocery (master)
$ git log --oneline --graph --decorate --all
* 99dd471 (HEAD -> master) Add a cherry-picked blackberry
* 6409527 Add a grape
* 603b9d1 Add a peach
| * a8c6219 (melons) Add a watermelon
| * ef6c382 (berries) Add a blackberry
|/
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Cherry picking



MINGW64:/home/grocery

```
Add a blackberry

(cherry picked from commit ef6c3821fd9a92b90bc9fc444befffc5326f4228)

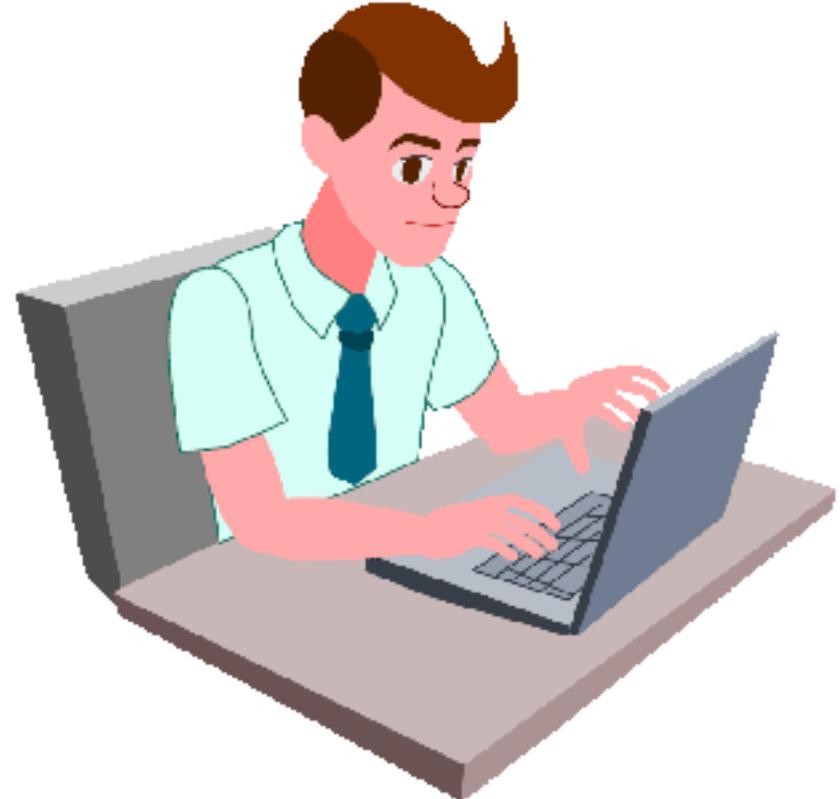
# Conflicts:
#       shoppingList.txt
#
# It looks like you may be committing a cherry-pick.
# If this is not correct, please remove the file
#       .git/CHERRY_PICK_HEAD
# and try again.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri Aug 25 13:24:38 2017 +0200
#
# On branch master
# You are currently cherry-picking commit ef6c382.
#
# Changes to be committed:
#       modified:   shoppingList.txt
#
~
```

~/grocery/.git/COMMIT\_EDITMSG [unix] (16:56 26/08/2017) 8,1 All

# Summary

- This has been a very long lesson, I know.
- But now I think you know all you need to work proficiently with Git, at least in your own local repository.
- You know about working tree, staging area, and HEAD commit; you know about references as branches and HEAD.
- You know how to merge rebase, and cherry pick; and finally, you know how Git works under the hood, and this will help you from here on out.



# "Complete Lab 7"

# 7. Resolving Merge Conflicts

# Local Merge Conflicts

- Merge conflicts occur when competing changes are made to the same line of a file, or when one person edits a file and another person deletes the same file.
- To resolve a merge conflict caused by competing line changes, you must choose which changes to incorporate from the different branches in a new commit.

# Local Merge Conflicts

- Open Git Bash.
- Navigate into the local Git repository that has the merge conflict.

```
cd REPOSITORY-NAME
```

# Local Merge Conflicts

```
$ git status
> # On branch branch-b
> # You have unmerged paths.
> # (fix conflicts and run "git commit")
> #
> # Unmerged paths:
> # (use "git add ..." to mark resolution)
> #
> # both modified:    styleguide.md
> #
> no changes added to commit (use "git add" and/or "git commit -a")
```

# Local Merge Conflicts

- In this example, one person wrote "open an issue" in the base or HEAD branch and another person wrote "ask your question in IRC" in the compare branch or branch-a.
- If you have questions, please

<<<<< HEAD

open an issue

=====

ask your question in IRC.

>>>>> branch-a

# Local Merge Conflicts

- Add or stage your changes.

```
$ git add .
```

- Commit your changes with a comment.

```
$ git commit -m "Resolved merge conflict by incorporating both  
suggestions."
```



# "Complete Lab 7"

# 8. Remote Repositories

# Git Fundamentals - Working Remotely

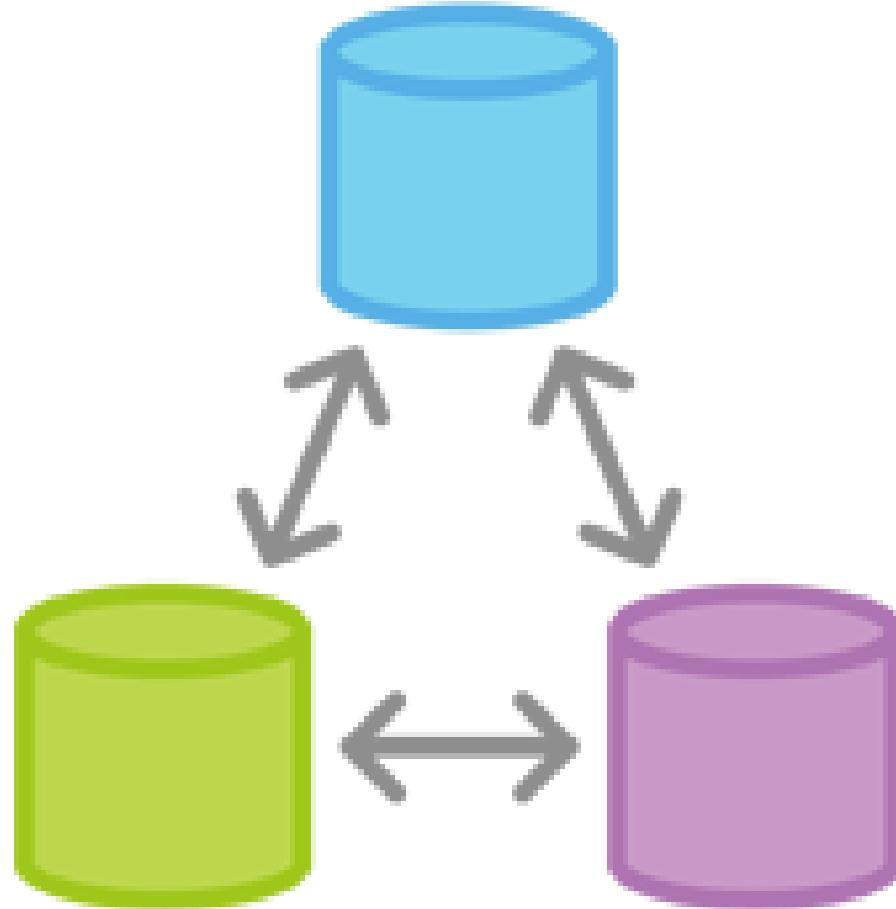
In this lesson, we finally start to work in a distributed manner, using remote servers as a contact point for different developers. These are the main topics we will focus on:

- Dealing with remotes
- Cloning a remote repository
- Working with online hosting services, such as GitHub

# Working with remotes

- Git is a tool for versioning files, as you know, but it has been built with collaboration in mind.
- In 2005, Linus Torvalds had the need for a light and efficient tool to share the Linux kernel code, allowing him and hundreds of other people to work on it without going crazy.

# Working with remotes



# Working with remotes

## Clone a local repository

- Create a new folder on your disk to clone our grocery repository:

```
[1] ~
```

```
$ mkdir grocery-cloned
```

- Then clone the grocery repository using the git clone command:

```
[2] ~ $ cd grocery-cloned [3] ~/grocery-cloned $ git clone ~/grocery . Cloning into '.'....
```

# Working with remotes

- Now, go directly to the point with a git log command:

```
[4] ~/grocery-cloned (master)
$ git log --oneline --graph --decorate --all
* 6409527 (HEAD -> master, origin/master, origin/HEAD) Add a grape
* 603b9d1 Add a peach
| * a8c6219 (origin/melons) Add a watermelon
| * ef6c382 (origin/berries) Add a blackberry
|/
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Working with remotes

- But don't worry: a local branch in which to work locally can be created by simply checking it out:

```
[5] ~/grocery-cloned (master)
```

```
$ git checkout berries
```

Branch berries set up to track remote branch berries from origin.

Switched to a new branch 'berries'

# Working with remotes

- Now, look at the log again:

```
[6] ~/grocery-cloned (berries)
$ git log --oneline --graph --decorate --all
* 6409527 (origin/master, origin/HEAD, master) Add a grape
* 603b9d1 Add a peach
| * a8c6219 (origin/melons) Add a watermelon
| * ef6c382 (HEAD -> berries, origin/berries) Add a blackberry
|/
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Working with remotes

- Let's try:

```
[7] ~/grocery-cloned (berries)
$ echo "blueberry" >> shoppingList.txt
```

```
[8] ~/grocery-cloned (berries)
$ git commit -am "Add a blueberry"
[berries ab9f231] Add a blueberry
Committer: Santacroce Ferdinando <san@intre.it>
```

# Working with remotes

- You can suppress this message by setting them explicitly:

```
git config --global user.name "Your Name"
```

```
git config --global user.email you@example.com
```

- After doing this, you may fix the identity used for this commit with the following code:

```
git commit --amend --reset-author
```

1 file changed, 1 insertion(+)

# Working with remotes

OK, let's see what happened:

```
[9] ~/grocery-cloned (berries)
$ git log --oneline --graph --decorate --all
* ab9f231 (HEAD -> berries) Add a blueberry
| * 6409527 (origin/master, origin/HEAD, master) Add a grape
| * 603b9d1 Add a peach
| | * a8c6219 (origin/melons) Add a watermelon
| |
|/
|/
* | ef6c382 (origin/berries) Add a blackberry
|/
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Working with remotes

- Now, we will try to push the modifications in the berries branch to the origin; the command is git push, followed by the name of the remote and the target branch:

```
[10] ~/grocery-cloned (berries)
$ git push origin berries
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 323 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To C:/Users/san/Google Drive/Packt/PortableGit/home/grocery
    ef6c382..ab9f231  berries -> berries
```

# Working with remotes

- Now, we obviously want to see if, in the remote repository, there is a new commit in the berries branch; so, open the grocery folder in a new console and do git log:

```
[11] ~
$ cd grocery

[12] ~/grocery (master)
$ git log --oneline --graph --decorate --all
* ab9f231 (berries) Add a blueberry
| * 6409527 (HEAD -> master) Add a grape
| * 603b9d1 Add a peach
| | * a8c6219 (melons) Add a watermelon
| |
|/
|/
* | ef6c382 Add a blackberry
|
* 0e8b5cf Add an orange
* e4a5e7b Add an apple
* a57d783 Add a banana to the shopping list
```

# Working with remotes

## Getting remote commits with git pull

- Now, it's time to experiment the inverse: retrieving updates from the remote repository and applying them to our local copy.
- So, make a new commit in the grocery repository, and then, we will download it into the grocery-cloned one:

```
[13] ~/grocery (master)
$ printf "\r\n" >> shoppingList.txt
```

# Working with remotes

- I firstly need to create a new line, because due to the previous grape rebase, we ended having the shoppinList.txt file with no new line at the end, as echo "" >> <file> usually does:

```
[14] ~/grocery (master)
$ echo "apricot" >> shoppingList.txt
```

```
[15] ~/grocery (master)
$ git commit -am "Add an apricot"
[master 741ed56] Add an apricot
 1 file changed, 2 insertions(+), 1 deletion(-)
```

# Working with remotes

- Let's try git pull for now, then we will try to use git fetch and git merge separately.
- Go back to the grocery-cloned repository, switch to the master branch, and do a git pull:

```
[16] ~/grocery-cloned (berries)
$ git checkout master
Your branch is up-to-date with 'origin/master'.
Switched to branch 'master'
```

# Working with remotes

- For now, go with git pull: the command wants you to specify the name of the remote you want to pull from, which is origin in this case, and then the branch you want to merge into your local one, which is master, of course:

```
[17] ~/grocery-cloned (master)
$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From C:/Users/san/Google Drive/Packt/PortableGit/home/grocery
 * branch      master    -> FETCH_HEAD
   6409527..741ed56  master    -> origin/master
Updating 6409527..741ed56
Fast-forward
  shoppingList.txt | 3 +++
  1 file changed, 2 insertions(+), 1 deletion(-)
```

# Working with remotes

- OK, now I want you to try doing these steps in a separate manner; create the umpteenth new commit in the grocery repository, the master branch:

```
[18] ~/grocery (master)
$ echo "plum" >> shoppingList.txt

[19] ~/grocery (master)
$ git commit -am "Add a plum"
[master 50851d2] Add a plum
1 file changed, 1 insertion(+)
```

# Working with remotes

- Now perform a git fetch on grocery-cloned repository:

```
[20] ~/grocery-cloned (master)
$ git fetch
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From C:/Users/san/Google Drive/Packt/PortableGit/home/grocery
  741ed56..50851d2  master      -> origin/master
```

# Working with remotes

- Do a git status now:

```
[21] ~/grocery-cloned (master)
$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
nothing to commit, working tree clean
```

# Working with remotes

- Now, let's sync with a git merge; to merge a remote branch, we have to specify, other than the branch name, even the remote one, as we did in the git pull command previously:

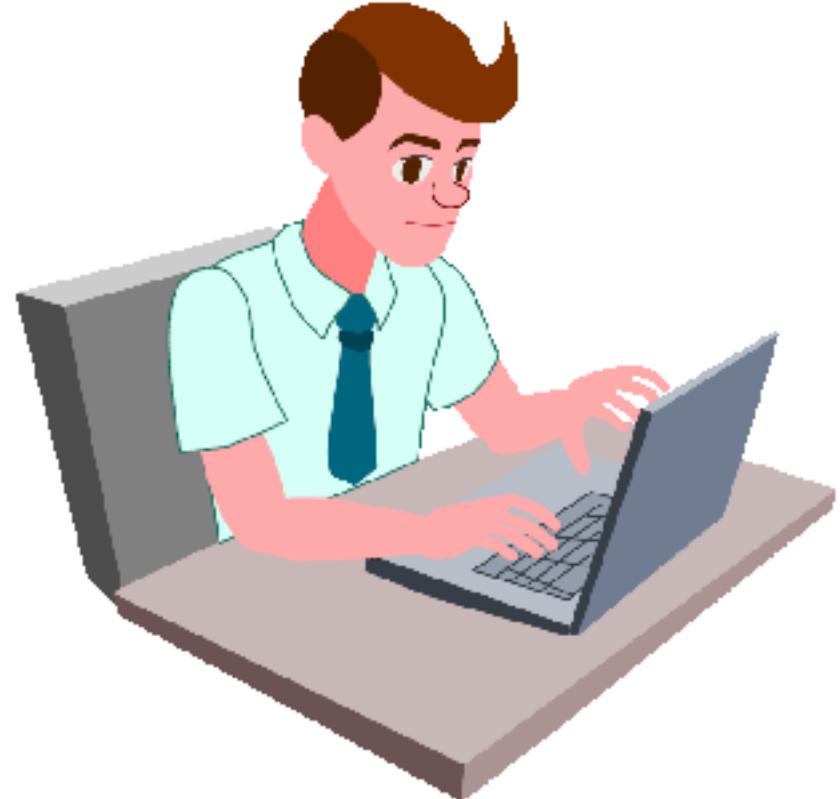
```
[22] ~/grocery-cloned (master)
$ git merge origin master
Updating 741ed56..50851d2
Fast-forward
  shoppingList.txt | 1 +
  1 file changed, 1 insertion(+)
```

# Working with remotes

## How Git keeps track of remotes

- Git stores remote branch labels in a similar way to how it stores the local branches ones; it uses a subfolder in refs for the scope, with the symbolic name we used for the remote, in this case origin, the default one:

```
[23] ~/grocery-cloned (master)
$ ll .git/refs/remotes/origin/
total 3
drwxr-xr-x 1 san 1049089 0 Aug 27 11:25 .
drwxr-xr-x 1 san 1049089 0 Aug 26 18:19 ../
-rw-r--r-- 1 san 1049089 41 Aug 26 18:56 berries
-rw-r--r-- 1 san 1049089 32 Aug 26 18:19 HEAD
-rw-r--r-- 1 san 1049089 41 Aug 27 11:25 master
```



# "Complete Lab 9"

# 9. Reviewing the Commit History

# Using Git Log

- In modern Git, you can trace the evolution of the line range within the file using `git log -L`, which is currently limited to walk starting from a single revision (zero or one positive revision arguments) and a single file.
- The range is given either by denoting the start and end of the range with `-L <start>,<end>:<file>` (where either `<start>` or `<end>` can be the line number or `/regexp/`), or a function to track with `-L <funcname regexp>:<file>`.

# Selecting and formatting the git log output

- Now that you know how to select revisions to examine and to limit which revisions are shown (selecting those that are interesting).
- There is a huge number and variety of options of the git log command available for this.

# Predefined and user defined output formats

```
$ git log --pretty="%h - %an, %ar : %s"  
50f84e3 - Junio C Hamano, 7 days ago : Update draft release notes  
0953113 - Junio C Hamano, 10 days ago : Second batch for 2.1  
afa53fe - Nick Alcock, 2 weeks ago : t5538: move http push tests out
```

# Using Git Log



Placeholder	Description of output
%H	Commit hash (full SHA-1 identifier of revision)
%h	Abbreviated commit hash
%an	Author name
%ae	Author e-mail
%ar	Author date, relative
%cn	Committer name
%ce	Committer email
%cr	Committer date, relative
%s	Subject (first line of a commit message, describing revision)
%%	A raw %

# Using Git Log

```
$ git log --graph --decorate --oneline origin/maint
* bce14aa (origin/maint) Sync with 1.9.4
  \
  | * 34d5217 (tag: v1.9.4) Git 1.9.4
  | * 12188a8 Merge branch 'rh/prompt' into maint
  |
  | \
  | * \ 64d8c31 Merge branch 'mw/symlinks' into maint
  |
  | \
  * ||| d717282 t5537: re-drop http tests
* ||| e156455 (tag: v2.0.0) Git 2.0
```



# 11. Continuous Integration / Continuous Delivery (CI/CD)

# CI/CD explained

- CI/CD falls under DevOps (the joining of development and operations teams) and combines the practices of continuous integration and continuous delivery.
- CI/CD automates much or all of the manual human intervention traditionally needed to get new code from a commit into production, encompassing the build, test (including integration tests, unit tests, and regression tests), and deploy phases, as well as infrastructure provisioning.

# What is continuous integration (CI)?

- Continuous integration is the practice of integrating all your code changes into the main branch of a shared source code repository early and often, automatically testing each change when you commit or merge them, and automatically kicking off a build.
- With continuous integration, errors and security issues can be identified and fixed more easily, and much earlier in the development process.
- By merging changes frequently and triggering automatic testing and validation processes, you minimize the possibility of code conflict, even with multiple developers working on the same application..

# What is continuous delivery (CD)?

- Continuous delivery is a software development practice that works in conjunction with CI to automate the infrastructure provisioning and application release process.
- Once code has been tested and built as part of the CI process, CD takes over during the final stages to ensure it's packaged with everything it needs to deploy to any environment at any time.
- CD can cover everything from provisioning the infrastructure to deploying the application to the testing or production environment.

# What is continuous deployment?

- Continuous deployment enables organizations to deploy their applications automatically, eliminating the need for human intervention.
- With continuous deployment, DevOps teams set the criteria for code releases ahead of time and when those criteria are met and validated, the code is deployed into the production environment.
- This allows organizations to be more nimble and get new features into the hands of users faster.

# What is continuous testing?

In continuous testing, various types of tests are performed within the CI/CD pipeline. These can include:

- Unit testing, which checks that individual units of code work as expected
- Integration testing, which verifies how different modules or services within an application work together
- Regression testing, which is performed after a bug is fixed to ensure that specific bug won't occur again

There are eight fundamental elements of CI/CD that help ensure maximum efficiency for your development lifecycle. They span development and deployment. Include these fundamentals in your pipeline to improve your DevOps workflow and software delivery:

- A single source repository
- Frequent check-ins to main branch
- Automated builds
- Self-testing builds
- Frequent iterations
- Stable testing environments
- Maximum visibility
- Predictable deployments anytime

## CI/CD fundamentals

# The benefits of CI/CD implementation

Companies and organizations that adopt CI/CD tend to notice a lot of positive changes. Here are some of the benefits you can look forward to as you implement CI/CD:

- Happier users and customers
- Accelerated time-to-value
- Less fire fighting
- Free up developers' time
- Less context switching
- Reduce burnout
- Recover faster

# Why GitLab CI/CD?

- In order to complete all the required fundamentals of full CI/CD, many CI platforms rely on integrations with other tools to fulfill those needs.
- Many organizations have to maintain costly and complicated toolchains in order to have full CI/CD capabilities.
- This often means maintaining a separate SCM like Bitbucket or GitHub, and connecting to a separate testing tool that connects to their CI tool, that connects to a deployment tool like Chef or Puppet, that also connects to various security and monitoring tools.

# GitLab CI

- GitLab CI is a feature that helps perform the Continuous Integration (CI) of software components.
- When several developers work together using a versioning system, problems can arise when changes made by one developer break the product as a whole.
- The best way to make sure this happens less often, or at least early in the process, is to use integration tests more frequently, hence the name continuous.

# GitLab CI

- Forrester classified GitLab as a leader in CI in The Forrester Wave: Continuous Integration Tools, Q3 2017.
- This is shown in the following diagram:



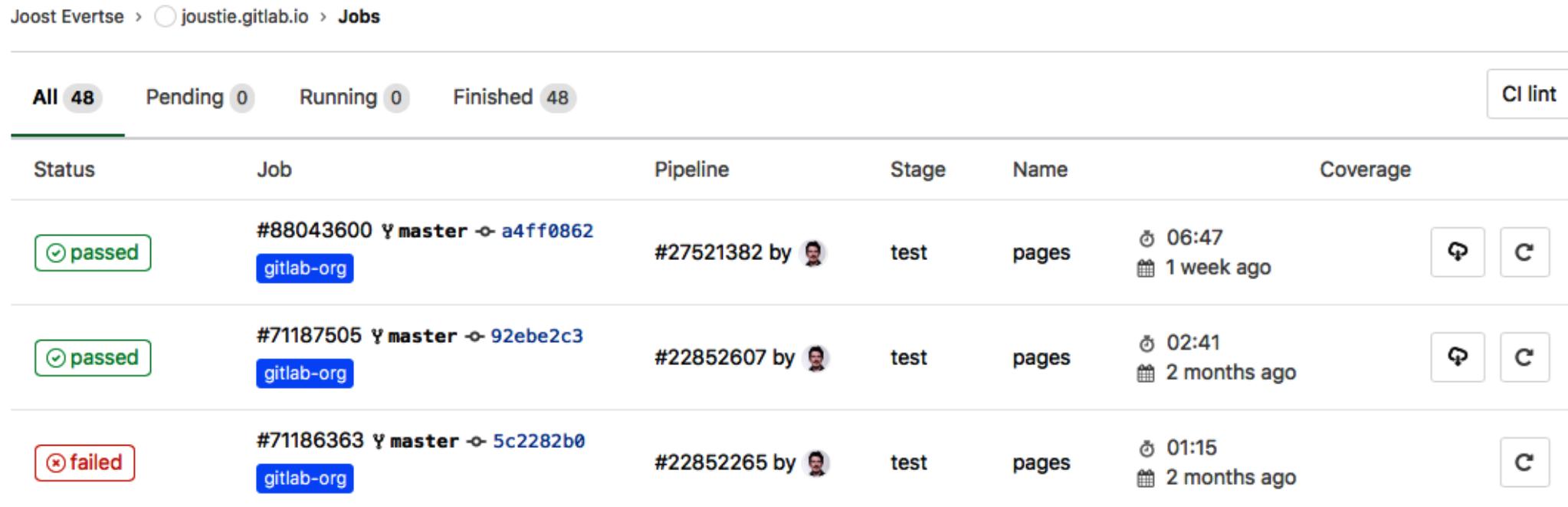
# Pipelines and jobs

Joost Evertse > [joustie.gitlab.io](#) > Pipelines

All 47	Pending 0	Running 0	Finished 47	Branches	Tags	Run Pipeline	Clear Runner Caches	CI Lint
Status	Pipeline		Commit	Stages				
<span>passed</span>	#27521382 by 	latest	 master -o a4ff0862  Translated minecraft post	 	⌚ 00:06:47	 		
<span>passed</span>	#22852607 by 		 master -o 92ebe2c3  Update .gitlab-ci.yml	 	⌚ 00:02:41	 		
<span>failed</span>	#22852265 by 		 master -o 5c2282b0  Update .gitlab-ci.yml		⌚ 00:01:15	 		
<span>failed</span>	#22852093 by 		 master -o 9f2c8edd  Update .gitlab-ci.yml		⌚ 00:01:28	 		

# Pipelines and jobs

- Alternatively, you can view all jobs, by going to the Pipelines' Jobs page, as shown in the following screenshot:

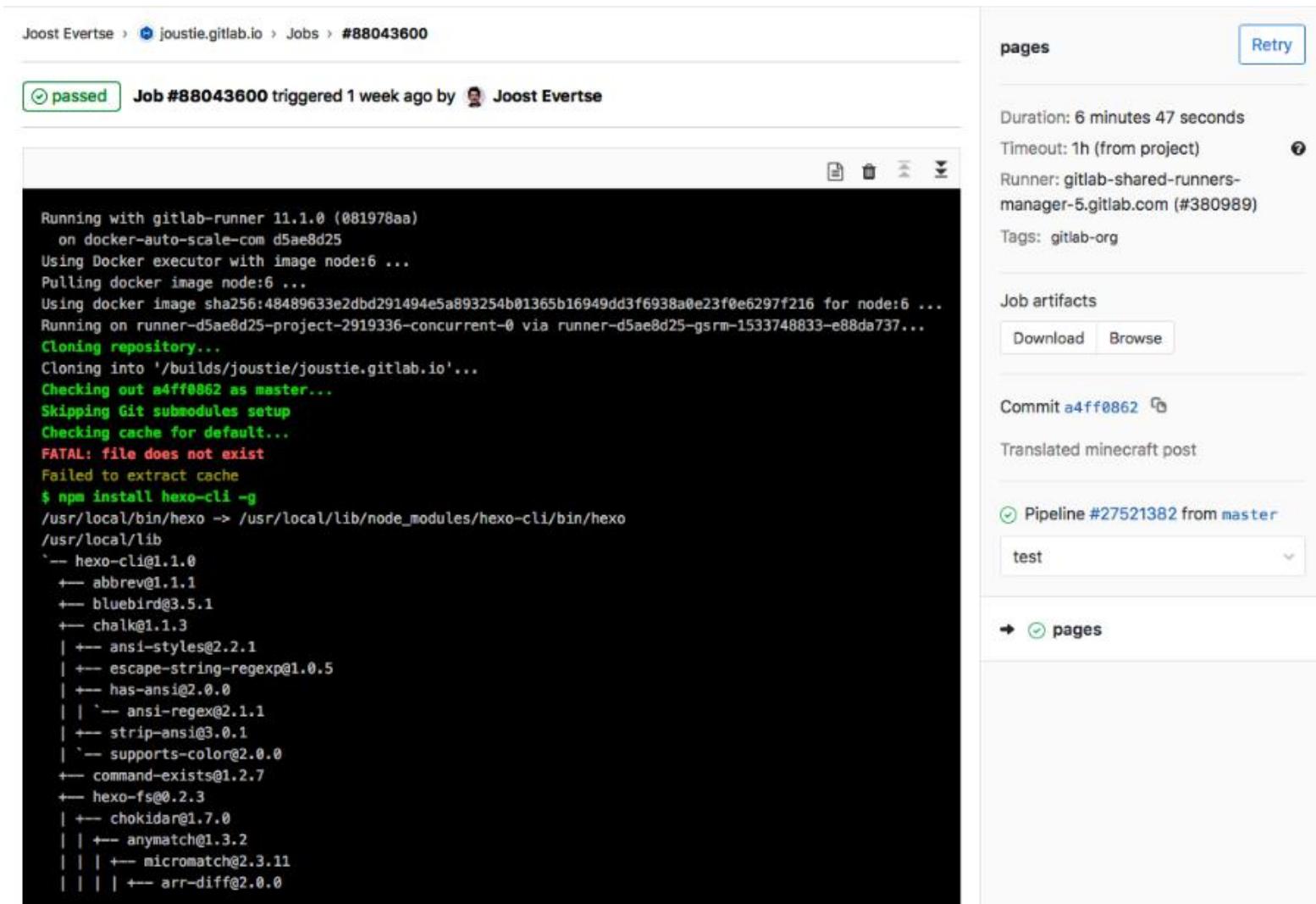


The screenshot shows a list of completed pipelines from the 'Jobs' page. The top navigation bar includes 'Joost Evertse > joustie.gitlab.io > Jobs'. Below the navigation, there are filters for 'All 48', 'Pending 0', 'Running 0', and 'Finished 48', along with a 'CI lint' button. The main table has columns for Status, Job, Pipeline, Stage, Name, and Coverage. Three rows are listed:

Status	Job	Pipeline	Stage	Name	Coverage
<span>passed</span>	#88043600 ⚡ master ➔ a4ff0862 gitlab-org	#27521382 by 🧑	test	pages	⌚ 06:47 📅 1 week ago
<span>passed</span>	#71187505 ⚡ master ➔ 92ebe2c3 gitlab-org	#22852607 by 🧑	test	pages	⌚ 02:41 📅 2 months ago
<span>failed</span>	#71186363 ⚡ master ➔ 5c2282b0 gitlab-org	#22852265 by 🧑	test	pages	⌚ 01:15 📅 2 months ago

# Pipelines and jobs

- You can check the log of a job by clicking on the status of the job (for example, failed or passed). You can debug why some jobs fail and see exactly what happened:



The screenshot shows a GitLab job log for job #88043600, triggered 1 week ago by Joost Evertse. The log output is as follows:

```
Running with gitlab-runner 11.1.0 (081978aa)
on docker-auto-scale-com d5ae8d25
Using Docker executor with image node:6 ...
Pulling docker image node:6 ...
Using docker image sha256:48489633e2dbd291494e5a893254b01365b16949dd3f6938a0e23f0e6297f216 for node:6 ...
Running on runner-d5ae8d25-project-2919336-concurrent-0 via runner-d5ae8d25-gsrm-1533748833-e88da737...
Cloning repository...
Cloning into '/builds/joustie/joustie.gitlab.io'...
Checking out a4ff0862 as master...
Skipping Git submodules setup
Checking cache for default...
FATAL: file does not exist
Failed to extract cache
$ npm install hexo-cli -g
/usr/local/bin/hexo => /usr/local/lib/node_modules/hexo-cli/bin/hexo
/usr/local/lib
`-- hexo-cli@1.1.0
  +-- abbrev@1.1.1
  +-- bluebird@3.5.1
  +-- chalk@1.1.3
    +-- ansi-styles@2.2.1
    +-- escape-string-regexp@1.0.5
    +-- has-ansi@2.0.0
    | +-- ansi-regex@2.1.1
    +-- strip-ansi@3.0.1
    +-- supports-color@2.0.0
    +-- command-exists@1.2.7
    +-- hexo-fs@0.2.3
    +-- chokidar@1.7.0
    | +-- anymatch@1.3.2
    | +-- micromatch@2.3.11
    | +-- arr-diff@2.0.0
```

The log indicates a fatal error: "FATAL: file does not exist" during the npm install process. The error occurs while trying to extract a cache file. The log also shows the dependency tree for the hexo-cli package.

# Issues with the old runner

- The main problem with the old runner is that it could only run one concurrent job at a time.
- If you wanted to run more, you could either set up a new server or create an additional user to build jobs.
- Secondly, it always ran projects on the server shell, This made it really hard to test projects using different versions of Ruby or any other dependencies.

# Switching to Go

- Go (or Golang) is a new language (less than 10 years old).
- It is already widely used by some impressive parties, such as Docker (<https://docker.com>), Google, Kubernetes (<https://kubernetes.io>), and Prometheus (<https://prometheus.io>).
- Go is a versatile tool that can help you to program at a low level, close to the operating system or at a high level in a language such as Java.
- It is perfectly suited to creating systems software.

# Switching to Go

- The project can be found at <https://gitlab.com/gitlab-org/gitlab-runner>:



A screenshot of a GitLab project page for "gitlab-runner". The page features a dark blue header with the GitLab logo and navigation links like "Dashboard", "Search", "Help Center", and "Logout". Below the header, there's a banner with the text "gitlab-runner" and a small orange fox icon. The main content area shows the project details: "GitLab Runner" and "Project ID: 250833". Below this, there are standard GitHub-style metrics: "Star" (1000), "Fork" (778), and "SSH" (with a dropdown menu). The repository URL is listed as "git@gitlab.com:gitlab-org/gitl". At the bottom, there are links for "Files", "Commits", "Branches", "Tags", "Readme", "Changelog", "LICENSE", "Contribution guide", and "CI/CD configuration".

# Build, test, deploy, and monitor your code from a single application

- We believe a single application that offers visibility across the entire
- SDLC is the best way to ensure that every development stage is included
- and optimized. When everything is under one roof, it's as easy to pinpoint
- workflow bottlenecks and evaluate the impact each element has on
- deployment speed. GitLab has CI/CD built right in, no plugins required.

# Breaking down .gitlab-ci.yml

- A basic .gitlab-ci.yml file might look something like this:

before\_script:

```
- apt-get update -qq && DEBIAN_FRONTEND=noninteractive apt-get install -y -  
qq ca-certificates git php php-xml  
- php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"  
- php composer-setup.php  
- php composer.phar install
```

phpunit:

script:

```
- vendor/bin/phpunit tests/ROT13FormatterTest
```

# The script parameter

- The most common parameter to a job, script is used to define any commands that should be run in the job. In our preceding example, we have the following:

phpunit:

script:

- vendor/bin/phpunit tests/ROT13FormatterTest

# The stage parameter

- Stages are defined at the top level of the YAML file and are used to define separate blocks of jobs, which can be executed in parallel.
- They also define the order in which stages are run. In each job, the stage parameter can be used to define which build stage a job is in, thus grouping together similar jobs and allowing for jobs to depend on other jobs having finished.
- The following is an example of defining and using stages:

```
stages:  
  - build  
  - test  
  - deploy  
  
job 1:  
  stage: build  
  script: npm run build dependencies  
  
job 2:  
  stage: build  
  script: npm run build artifacts  
  
job 3:  
  stage: test  
  script: npm run test  
  
job 4:  
  stage: deploy  
  script: npm run deploy
```

# The only and except parameters

- Please note that only and except don't have to be mutually exclusive either; you can use a combination of them to have more fine-grained control over when a job will be executed.
- An example of this is as follows:

job:

only:

- /`^iss-.*$/`

except:

- tags

# The tags parameter

- You can specify a tag or tags in this parameter, which will limit this job to only being executed on Runners that also have the same tag.
- Please note that the tags are additive, so if you specify multiple tags, the job will only be executed on a Runner that has all of those tags present:

job:

tags:

- php

- postgres

# The tags parameter

- One example use case of when is as follows:

stages:

- build
- cleanup\_build

build\_job:

stage: build

script:

- webpack

cleanup\_build\_job:

stage: cleanup\_build

script:

- rm /dist/\*

when: on\_failure

# cache – paths

- This is an array of paths to files and/or directories that should be cached.
- You can also use the asterisk wildcard character (\*). This is demonstrated in the following code snippet:

build:

  cache:

  paths:

- binaries/\*.apk
- .config

# cache – key

- This takes a string that can be used to create separate caches for different jobs or branches, like so:

test:

cache:

key: "\$CI\_COMMIT\_REF\_SLUG"

paths:

- binaries/

# cache – key

- In this example, we use an inbuilt default variable provided by GitLab CI – `CI_COMMIT_REF_SLUG` – that is equal to the branch/tag name of the commit.
- In this case, GitLab CI will maintain a separate cache for each branch. Next up, let's look at jobs with different paths cached:

```
stages:  
  - build  
  - test  
  
build_job:  
  stage: build  
  script: npm run build  
  cache:  
    key: build-key  
    paths:  
      - public/  
  
test_job:  
  stage: test  
  script: npm run test  
  cache:  
    key: test-key  
    paths:  
      - vendor/
```

# Breaking down .gitlab-ci.yml

- This lesson is a rapid and very high-level overview of Python; the following lessons provide more detail.
- You may find it valuable to return to this lesson and work through the appropriate examples as a review after you read about the features covered in subsequent lessons.
- If this lesson was mostly a review for you, or if you'd like to learn more about only a few features, feel free to jump around, using the index or table of contents.

# artifacts – paths

- Much like with caching, you specify an array of paths, which can include wildcards like so:

```
package:  
  artifacts:  
    paths:  
      - public/  
      - tests/*.html
```

# artifacts – name

- By default, any uploaded files will be stored in a .zip archive titled artifacts.zip.
- You can use the name parameter to change the default filename (it will still end with .zip), as demonstrated in the following code:

```
package:  
artifacts:  
  name: cool_project_name
```

# The variables parameter

- You can store variables per job or globally using the variables keyword.
- The value for variables should be an array of key/value pairs that can be represented by strings or integers (for both key and value), although typically the key will be an all-capitalized string for ease of recognition.
- Any variables used should be of a non-sensitive nature; they are not considered an appropriate method for storing secrets.
- Locally defined variables will override their globally defined counterparts, but to have a job run with no access to globally defined variables, you should redeclare it with an empty array like so:

```
cool_job_name:  
  variables: {}
```

# Adding .gitlab-ci.yml to our example project

before\_script:

- apt-get update -qq && DEBIAN\_FRONTEND=noninteractive apt-get install -y -qq ca-certificates git php php-xml
- php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
- php composer-setup.php
- php composer.phar install

phpunit:

script:

- vendor/bin/phpunit tests/ROT13FormatterTest

# Adding .gitlab-ci.yml to our example project

Run untagged jobs

Indicates whether this runner can pick jobs without tags

Now, when you go to Project | CI/CD | Pipelines, you should see your new pipeline there, ready to go:



# Adding .gitlab-ci.yml to our example project

passed Pipeline #6 triggered 16 minutes ago by  Administrator

---

adding git install to CI/CD startup so Composer works

- ① 1 job from [2-add-testing-and-ci](#) in 2 minutes 9 seconds (queued for 2 seconds)
  - o a8319230 [...](#) [📄](#)

Pipeline Jobs 1

---

Test

-  [phpunit](#) 

- You can click the status (passed, in the preceding screenshot) to receive a breakdown of the pipeline. Yours should look similar to this:

# Adding .gitlab-ci.yml to our example project

Adam O'Grady > monoROT13 > Jobs > #3



Job #3 triggered 1 minute ago by Adam O'Grady

```
Running with gitlab-runner 11.1.0 (081978aa)
on gitlab-runner-0 21e117ae
Using Docker executor with image ubuntu:latest ...
Pulling docker image ubuntu:latest ...
Using docker image sha256:cd6d8154f1e16e38493c3c2798977c5e142be5e5d41403ca89883840c6d51762 for ubuntu:latest ...
Running on runner-21e117ae-project-2-concurrent-0 via gitlab-runner-0...
Cloning repository...
Cloning into '/builds/judges119/monoROT13'...
Checking out a62246f2 as 2-add-testing-and-ci...
Skipping Git submodules setup
$ apt-get update -qq && apt-get install -y -qq php
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package perl-modules-5.26.
```

# Deconstructing an advanced .gitlab-ci.yml file

- The header of the .gitlab-ci.yml file defines an image to use from the Docker Registry (image: bitnami/laravel:latest), as well as defining a service (in this case, the postgres:9.6 image, also from the Docker registry).
- This service container will be attached directly to our Laravel image to provide a database without needing to set up a database as part of a before\_script each time a job is run:  
`image: bitnami/laravel:latest`

`services:`

`- postgres:9.6`

# Deconstructing an advanced .gitlab-ci.yml file

- The variables that are declared all relate to our database and will be used in the jobs so that we can connect to our database service image.
- Of particular note is the DB\_HOST parameter, which simply specifies postgres as the hostname.
- This is because with GitLab CI, any services that are connected are referred to by their image name, in this case, postgres:  
`variables:`  
    `POSTGRES_DATABASE: postgres`  
    `POSTGRES_PASSWORD: password`  
    `DB_HOST: postgres`  
    `DB_USERNAME: root`

# Deconstructing an advanced .gitlab-ci.yml file

- Next, we define three stages that we want: test, package, and deploy.
- These stages will be run sequentially (and any errors in an earlier stage will cause the whole pipeline to be canceled):

`stages:`

- `test`
- `package`
- `deploy`

# Deconstructing an advanced .gitlab-ci.yml file

- The big difference between the two is that php\_unit\_test also contains a cache section, which means that after the job is finished, the entire vendor/ directory will be zipped up and stored under the composer key for reuse in future jobs and pipelines.
- This helps reduce the time taken to build and execute subsequent runs of this job and any other jobs that use the same cache.
- Once both jobs are finished (and both are successful), the next stage can begin:

```
php_unit_test:  
  stage: test  
  script:  
    - cp .env.example .env  
    - composer install  
    - php artisan key:generate  
    - php artisan migrate  
    - vendor/bin/phpunit  
  cache:  
    key: composer  
    paths:  
      - vendor/  
  
js_unit_test:  
  stage: test  
  script:  
    - npm install  
    - npm run test
```

# Deconstructing an advanced .gitlab-ci.yml file

- For this example, we've also set the policy to pull to demonstrate it.
- This means that while it will download any available cache stored under the composer key, it will not re-upload any files once the job has finished.
- This task will also only operate on tagged commits, which is handy if you're doing versioned releases of software.
- Once all of the scripts have been executed, all of the files in the public/ directory will be packaged and uploaded as artifacts and stored for up to a week:

```
package_upload:  
  stage: package  
  script:  
    - composer install  
    - npm install  
    - webpack  
  cache:  
    key: composer  
    paths:  
      - vendor/  
  policy: pull  
artifacts:  
  paths:  
    - public/  
  expire_in: 1 week  
only:  
  - tags
```

# Deconstructing an advanced .gitlab-ci.yml file

```
deploy_production:
  stage: deploy
  script:
    - composer install
    - npm install
    - webpack
    - .composer/vendor/bin/envoy run deploy
  environment:
    name: production
    url: http://192.168.0.1
  when: manual
  only:
    - master
```

# GitLab CI/CD web UI

The screenshot shows the GitLab CI/CD web interface for the project `monoROT13`. The left sidebar has a navigation menu with items like Project, Repository, Issues, Merge Requests, CI / CD (which is currently selected), Pipelines, Jobs, Schedules, Charts, Operations, Wiki, Snippets, and Settings. The main area displays a table of pipelines. The table columns are Status, Pipeline, Commit, and Stages. The table shows the following data:

Status	Pipeline	Commit	Stages
skipped	#9 by latest	3-trial-env... -o c43d82bc testing the manual keyword	00:02:07 3 days ago
passed	#8 by latest	3-trial-env... -o 5908db68 #3 trialling environments i...	00:02:07 3 days ago
passed	#7 by latest	master -o 0b7cb642 Merge branch '2-add-testi...	00:02:06 3 days ago
passed	#6 by latest	2-add-testi... -o a8319230 adding git install to CI/CD ...	00:02:09 1 week ago
failed	#5 by latest	2-add-testi... -o 0406088e adding php-xml package t...	00:00:55 1 week ago
failed	#4 by latest	2-add-testi... -o d7a00f3f adding ca-certificates pac...	00:00:52 1 week ago

At the top right, there are buttons for "Run Pipeline", "Clear Runner Caches", and "CI Lint". The top navigation bar includes links for GitLab, Projects, Groups, Activity, Milestones, Snippets, and a search bar.

# GitLab CI/CD web UI

## Check your .gitlab-ci.yml

- The output of a linted .gitlab-ci.yml file looks like this:

Content of .gitlab-ci.yml

```
1 image: bitnami/laravel:latest
2
3 services:
4   - postgres:9.6
5
6 variables:
7   POSTGRES_DATABASE: postgres
8   POSTGRES_PASSWORD: password
9   DB_HOST: postgres
10  DB_USERNAME: root
11
12 stages:
13   - test
14   - package
15   - deploy
16
17 php_unit_test:
18   stage: test
19   script:
20     - cp .env.example .env
21     - composer install
22     - php artisan key:generate
23     - php artisan migrate
24     - vendor/bin/phpunit
25   cache:
```

Validate

Clear

Status: syntax is correct

# GitLab CI/CD web UI

- If the syntax passes its validation, you'll be shown the preceding message, as well as a table (visible in the following screenshot) that breaks down the keys and values provided in the script. You can use these to sanity check your work:

Parameter	Value
Test Job - php_unit_test	<p><b>Tag list:</b> <b>Only policy:</b> <b>Except policy:</b> <b>Environment:</b> <b>When:</b> on_success</p> <pre>cp .env.example .env composer install php artisan key:generate php artisan migrate vendor/bin/phpunit</pre>
Test Job - js_unit_test	<p><b>Tag list:</b> <b>Only policy:</b> <b>Except policy:</b> <b>Environment:</b> <b>When:</b> on_success</p> <pre>npm install npm run test</pre>

# GitLab CI/CD web UI

The screenshot shows the GitLab CI/CD web interface for the project `monoROT13`. The left sidebar has a navigation menu with options: Project, Repository, Issues (1), Merge Requests (0), CI / CD (selected), Pipelines, Jobs (selected), Schedules, Charts, Operations, Wiki, and Snippets. The main content area displays a table of CI jobs. The table columns are: Status, Job, Pipeline, Stage, Name, and Coverage. There are 11 total jobs, with 0 pending, 0 running, and 10 finished.

Status	Job	Pipeline	Stage	Name	Coverage
manual	#11 3-trial-env... -o c43d82bc allowed to fail manual	#9 by 🐛	test	phpunit	
passed	#10 3-trial-env... -o 5908db68	#8 by 🐛	test	phpunit	⌚ 02:07 📅 3 days ago
passed	#9 3-trial-env... -o 5908db68	#8 by 🐛	test	phpunit	⌚ 02:07 📅 3 days ago
passed	#8 master -o 0b7cb642	#7 by 🐛	test	phpunit	⌚ 02:06 📅 3 days ago
passed	#7 2-add-testi... -o a8319230	#6 by 🐛	test	phpunit	⌚ 02:09 📅 1 week ago
failed	#6 2-add-testi... -o 0406088e	#5 by 🐛	test	phpunit	⌚ 00:55 📅 1 week ago
failed	#5 2-add-testi... -o d7a00f3f	#4 by 🐛	test	phpunit	⌚ 00:52 📅 1 week ago

# GitLab CI/CD web UI

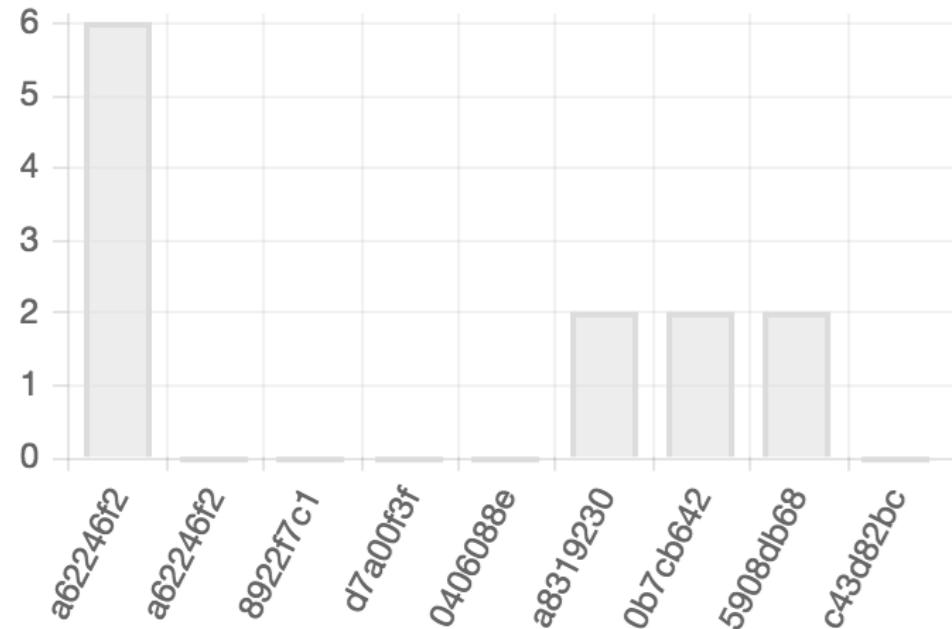
- From the menu on the left, under CI/CD, you can click Charts to get some graphical information about the CI/CD process attached to this project:

A collection of graphs regarding Continuous Integration

## Overall statistics

- Total: **9 pipelines**
- Successful: **3 pipelines**
- Failed: **4 pipelines**
- Success ratio: **42%**

Commit duration in minutes for last 30 commits



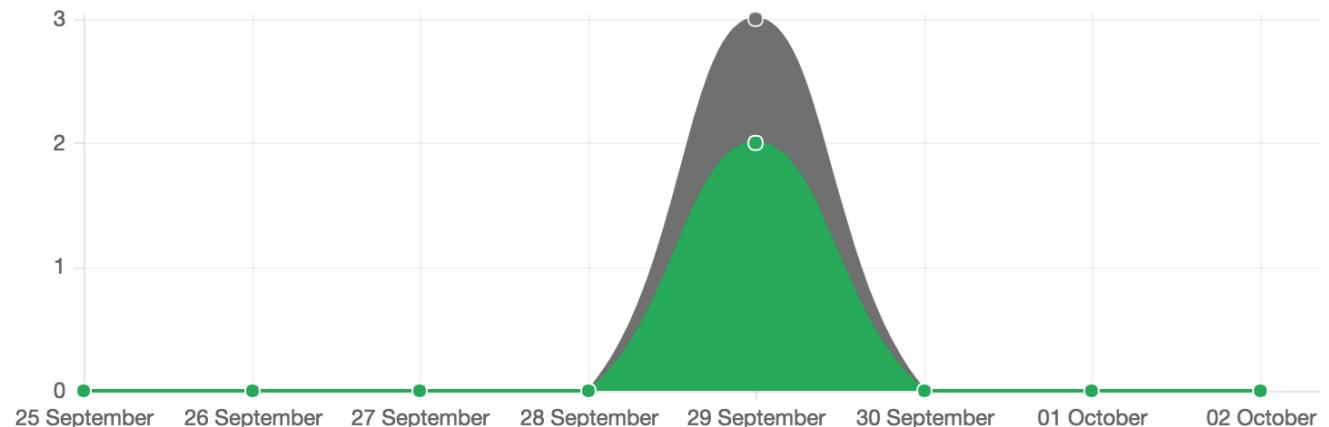
# GitLab CI/CD web UI

- Next up, we have charts showing the breakdown of pipelines over the past week, month, and year:

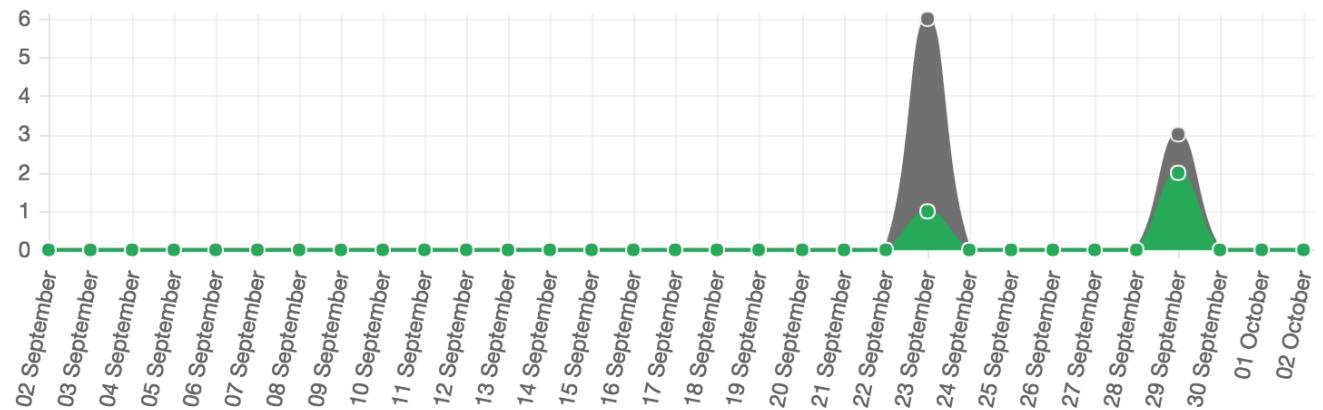
## Pipelines charts

● success ● all

Pipelines for last week (25 Sep - 2 Oct)



Pipelines for last month ( 2 Sep - 2 Oct)



# GitLab CI/CD web UI

The screenshot shows the GitLab CI/CD web UI for the project `monoROT13`. The left sidebar has a navigation bar with links: Project, Repository, Issues (1), Merge Requests (0), CI / CD, Operations (Metrics, Environments, Kubernetes). The **Environments** link is currently selected. The main area displays the environments for the project. It shows one available environment named `testing`, which was deployed by a job #3 (using `phpunit #12`) from a commit `e38afdea` (removing the manual keyw...). The deployment was updated 4 minutes ago. A green button labeled "New environment" is visible.

M monoROT13

Adam O'Grady > monoROT13 > Pipelines > Environments

Project Repository Issues (1) Merge Requests (0) CI / CD Operations Metrics Environments Kubernetes

Available 1 Stopped 0

New environment

Environment	Deployment	Job	Commit	Updated
testing	#3 by 🎨	phpunit #12	→ e38afdea 💀 removing the manual keyw...	4 minutes ago

Re-deploy

# Summary

- By now, you should have a firm grip on the basics of continuous integration and continuous delivery, as done by GitLab.
- We started out in this lesson by exploring the concept of CI/CD and giving you a crash course in it.
- Next up, we ran through the process of installing a GitLab Runner, Don't forget that the Runner is the platform where your CI/CD stages are actually executed; you can have many of them per project or per GitLab installation to help parallelize the work.
- We looked at their installation on Ubuntu and CentOS, and manually installing one via a binary.



# "Complete Lab 10 & 11"

# The scope of runners ALL TIERS

Runners are available based on who you want to have access:

- Shared runners are available to all groups and projects in a GitLab instance.
- Group runners are available to all projects and subgroups in a group.
- Project runners are associated with specific projects. Typically, project runners are used by one project at a time.

# Shared runners

- Shared runners are available to every project in a GitLab instance.
- Use shared runners when you have multiple jobs with similar requirements. Rather than having multiple runners idling for many projects, you can have a few runners that handle multiple projects.

If you are using a self-managed instance of GitLab:

- Your administrator can install and register shared runners by going to your project's Settings > CI/CD, expanding Runners, and selecting Show runner installation instructions. These instructions are also available in the documentation.

# Enable shared runners for a project

For existing projects, an administrator must install and register them.

To enable shared runners for a project:

- On the top bar, select Main menu > Projects and find your project.
- On the left sidebar, select Settings > CI/CD.
- Expand Runners.
- Turn on the Enable shared runners for this project toggle.

# Enable shared runners for a group

To enable shared runners for a group:

- On the top bar, select Main menu > Groups and find your group.
- On the left sidebar, select Settings > CI/CD.
- Expand Runners.
- Turn on the Enable shared runners for this group toggle.

# Disable shared runners for a project

To disable shared runners for a project:

- On the top bar, select Main menu > Projects and find your project.
- On the left sidebar, select Settings > CI/CD.
- Expand Runners.
- In the Shared runners area, turn off the Enable shared runners for this project toggle.

Shared runners are automatically disabled for a project:

- If the shared runners setting for the parent group is disabled, and
- If overriding this setting is not permitted at the project level.

# Disable shared runners for a group

To disable shared runners for a group:

- On the top bar, select Main menu > Groups and find your group.
- On the left sidebar, select Settings > CI/CD.
- Expand Runners.
- Turn off the Enable shared runners for this group toggle.
- Optional. To allow shared runners to be enabled for individual projects or subgroups, select Allow projects and subgroups to override the group setting.

# How shared runners pick jobs

- Shared runners process jobs by using a fair usage queue. This queue prevents projects from creating hundreds of jobs and using all available shared runner resources.
- The fair usage queue algorithm assigns jobs based on the projects that have the fewest number of jobs already running on shared runners.
- For example, if these jobs are in the queue:

Job 1 for Project 1

Job 2 for Project 1

Job 3 for Project 1

Job 4 for Project 2

Job 5 for Project 2

Job 6 for Project 3

# Group runners

- Use group runners when you want all projects in a group to have access to a set of runners.
- Group runners process jobs by using a first in, first out (FIFO) queue.

# Create a group runner

- You can create a group runner for your self-managed GitLab instance or for GitLab.com. You must have the Owner role for the group.

To create a group runner:

- Install GitLab Runner.
- On the top bar, select Main menu > Groups and find your group.
- On the left sidebar, select CI/CD > Runners.
- In the upper-right corner, select Register a group runner.
- Select Show runner installation and registration instructions. These instructions include the token, URL, and a command to register a runner.

# Project runners

Use project runners when you want to use runners for specific projects.

For example, when you have:

- Jobs with specific requirements, like a deploy job that requires credentials.
- Projects with a lot of CI activity that can benefit from being separate from other runners.

You can set up a project runner to be used by multiple projects. Project runners must be enabled for each project explicitly.

- Project runners process jobs by using a first in, first out (FIFO) queue.

# Create a project runner

To create a project runner:

- Install GitLab Runner.
- On the top bar, select Main menu > Projects and find the project where you want to use the runner.
- On the left sidebar, select Settings > CI/CD.
- Expand Runners.
- In the Project runners section, note the URL and token.
- Register the runner.
- The runner is now enabled for the project

# Enable a project runner for a different project

After a project runner is created, you can enable it for other projects.

- Prerequisites: You must have at least the Maintainer role for:
  - The project where the runner is already enabled.
  - The project where you want to enable the runner.
  - The project runner must not be locked

# Prevent a project runner from being enabled for other projects

To lock or unlock a project runner:

- On the top bar, select Main menu > Projects and find the project where you want to enable the runner.
- On the left sidebar, select Settings > CI/CD.
- Expand Runners.
- Find the project runner you want to lock or unlock. Make sure it's enabled. You cannot lock shared or group runners.
- Select Edit () .
- Select the Lock to current projects checkbox.
- Select Save changes.

# Executors ALL TIERS

To jump into the specific documentation for each executor, visit:

- SSH
- Shell
- Parallels
- VirtualBox
- Docker
- Docker Machine (auto-scaling)
- Kubernetes
- Custom

# Prerequisites for non-Docker executors

- Executors that do not rely on a helper image require a Git installation on the target machine and in the PATH.
- Always use the latest available version of Git.
- GitLab Runner uses the `git lfs` command if Git LFS is installed on the target machine.
- Ensure Git LFS is up to date on any systems where GitLab Runner uses these executors.

# Selecting the executor

Executor	SSH	Shell	VirtualBox	Parallels	Docker	Kubernetes	Custom
Clean build environment for every build	✗	✗	✓	✓	✓	✓	conditional (4)
Reuse previous clone if it exists	✓	✓	✗	✗	✓	✗	conditional (4)
Runner file system access protected (5)	✓	✗	✓	✓	✓	✓	conditional
Migrate runner machine	✗	✗	partial	partial	✓	✓	✓
Zero-configuration support for concurrent builds	✗	✗ (1)	✓	✓	✓	✓	conditional (4)
Complicated build environments	✗	✗ (2)	✓ (3)	✓ (3)	✓	✓	✓
Debugging build problems	easy	easy	hard	hard	medium	medium	medium

# Compatibility chart

- Supported features by different executors:

Executor	SSH	Shell	VirtualBox	Parallels	Docker	Kubernetes	Custom
Secure Variables	✓	✓	✓	✓	✓	✓	✓
GitLab Runner Exec command	✗	✓	✗	✗	✓	✓	✓
.gitlab-ci.yml: image	✗	✗	✓ (1)	✓ (1)	✓	✓	✓ (via \$CUSTOM_ENV_CI_JOB_IMA GE)
.gitlab-ci.yml: services	✗	✗	✗	✗	✓	✓	✓
.gitlab-ci.yml: cache	✓	✓	✓	✓	✓	✓	✓
.gitlab-ci.yml: artifacts	✓	✓	✓	✓	✓	✓	✓
Passing artifacts between stages	✓	✓	✓	✓	✓	✓	✓
Use GitLab Container Registry private images	n/a	n/a	n/a	n/a	✓	✓	n/a
Interactive Web terminal	✗	✓ (UNIX)	✗	✗	✓	✓	✗

# Compatibility chart

- Supported systems by different shells:

Shells	Bash	PowerShell Desktop	PowerShell Core	Windows Batch (deprecated)
Windows	✗ (4)	✓ (3)	✓	✓ (2)
Linux	✓ (1)	✗	✓	✗
macOS	✓ (1)	✗	✓	✗
FreeBSD	✓ (1)	✗	✗	✗

# Compatibility chart

- Supported systems for interactive web terminals by different shells:

Shells	Bash	PowerShell Desktop	PowerShell Core	Windows Batch (deprecated)
Windows	✗	✗	✗	✗
Linux	✓	✗	✗	✗
macOS	✓	✗	✗	✗
FreeBSD	✓	✗	✗	✗

# Plan and operate a fleet of shared runners

When you host a fleet of shared runners, you need a well-planned infrastructure that takes into consideration your:

- Computing capacity.
- Storage capacity.
- Network bandwidth and throughput.
- Type of jobs (including programming language, OS platform, and dependent libraries).
- Use this guide to develop a GitLab Runner deployment strategy based on your organization's requirements.

# Consider your workload and environment

Before you deploy runners, consider your workload and environment requirements.

- Create a list of the teams that you plan to onboard to GitLab.
- Catalog the programming languages, web frameworks, and libraries in use at your organization. For example, GoLang, C++, PHP, Java, Python, JavaScript, React, Node.js.
- Estimate the number of CI/CD jobs each team may execute per hour, per day.
- Validate if any team has build environment requirements that cannot be addressed by using containers.

# Workers, executors, and autoscaling capabilities

- The gitlab-runner executable runs your CI/CD jobs.
- Each runner is an isolated process that picks up requests for job executions and deals with them according to pre-defined configurations.
- As an isolated process, each runner can create “sub-processes” (also called “workers”) to run jobs.

# Basic configuration: one runner, one worker

- After the installation is complete, you execute the runner registration command just once and you select the shell executor.
- Then you edit the runner config.toml file to set concurrency to 1.

```
concurrent = 1
```

```
[[runners]]  
  name = "instance-level-runner-001"  
  url = ""  
  token = ""  
  executor = "shell"
```

# Intermediate configuration: one runner, multiple workers

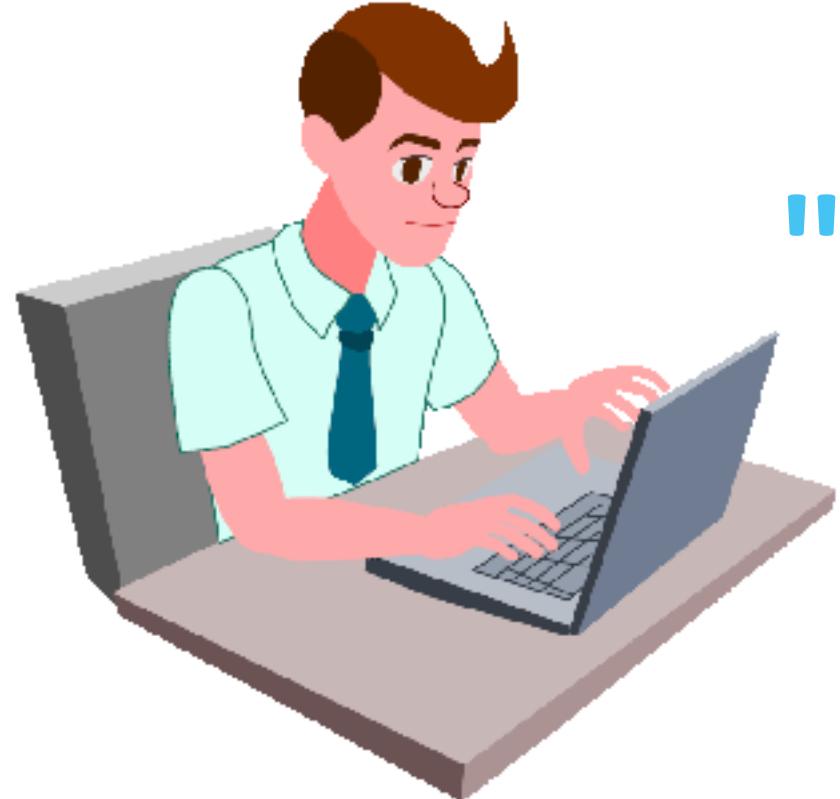
- You can also register multiple runner workers on the same machine.
- When you do this, the runner's config.toml file has multiple [[runners]] sections in it.
- If all of the additional runner workers are registered to use the shell executor, and you update the value of the global configuration option, concurrent, to 3, the upper limit of jobs that can run concurrently on this host is equal to three.

```
concurrent = 3

[[runners]]
  name = "instance_level_shell_001"
  url = ""
  token = ""
  executor = "shell"

[[runners]]
  name = "instance_level_shell_002"
  url = ""
  token = ""
  executor = "shell"

[[runners]]
  name = "instance_level_shell_003"
  url = ""
  token = ""
  executor = "shell"
```



# "Complete Lab 12 & 13"

## 12. Advanced Infrastructure Management with Terraform

# How to manage CockroachDB as Code with Terraform

## How to deploy CockroachDB Dedicated with Terraform

### List of required prerequisites:

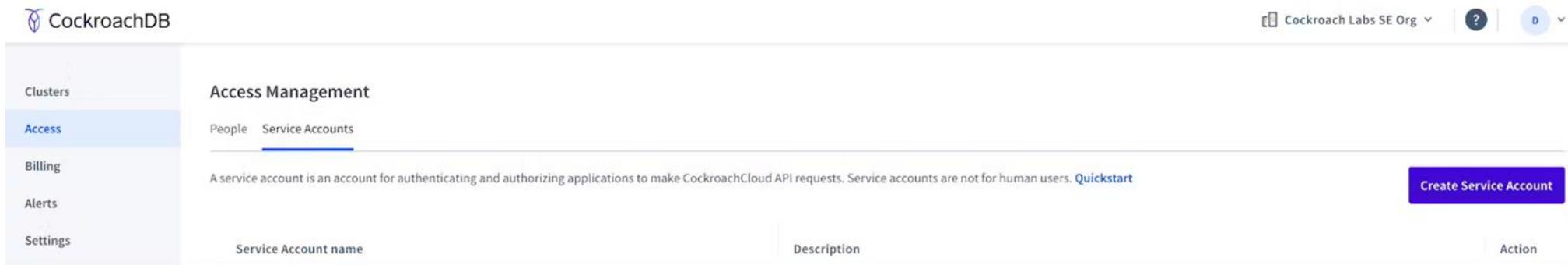
- **Git CLI**
- **Terraform (=>1.3.1)**
- **Cockroach Cloud API Key**

## Step 1: Clone the git repository

- git clone <https://github.com/cockroachlabs-field/cockroachdb-terraform-dedicated-july2023-example.git>

## Step 2: Grab your API key

Before we get into the Terraform code, and the resources it creates, we need to add the Cockroach API key as a local environment variable. You can create one of these in the Cockroach Cloud UI by adding a service account, which then allows you to generate a key associated with it.



The screenshot shows the CockroachDB Cloud UI interface. The top navigation bar includes the CockroachDB logo, a search bar, and user profile information. The main menu on the left has options: Clusters, Access (which is selected and highlighted in blue), Billing, Alerts, and Settings. The central content area is titled "Access Management" under the "Service Accounts" tab. It contains a sub-header "People" and a "Create Service Account" button. A descriptive text states: "A service account is an account for authenticating and authorizing applications to make CockroachCloud API requests. Service accounts are not for human users." Below this, there is a table with columns: "Service Account name", "Description", and "Action". There are no rows currently listed in the table.

Once a service account is created and an API key is generated we export that into our environment for Terraform to use.

- **export COCKROACH\_API\_KEY=<YOUR\_API\_KEY>**

## Step 3: Prepare your variables

- The new CockroachDB Cloud Terraform provider requires the presence of the environment variable to be able to create the required resources within the cloud platform. Now let's take a closer look at the Terraform code itself.

```
cluster_name = "dsmbcrdbtftexample2"
sql_user_name = "mbds"
sql_user_password = "ThisIsASuperSafePassword"
cloud_provider = "AWS"
cloud_provider_region = ["eu-west-2"]
cluster_nodes = "3"
storage_gib = "150"
machine_type = "m6i.xlarge"
region = "eu-west-2"
datadog_site = "US1"
datadog_api_key = "redacted"
```

## Step 4: Connection String and Certificate

- This is followed by the location of the certificate required to communicate with the cluster. There is an example below.

```
'postgresql://<user>@<cluster-name>-<short-id>.<region>.<host>:26257/<database>?sslmode=verify-full&sslrootcert='\$HOME/Library/CockroachCloud/certs/<cluster-name>-ca.crt'
```

- To retrieve this information with Terraform you need to create a data resource in your terraform code to retrieve it. Below is an example on how to do this.

```
data "cockroach_connection_string" "cockroach" {  
    id      = cockroach_cluster.rockroach.id  
    sql_user = cockroach_sql_user.rockroach.name  
    database = cockroach_cluster.rockroach.id  
}
```

- With this, you can then create an output in your output.tf file to display the connection string when the terraform runs, or output to a variable to use elsewhere. Below is an example of outputting the connection string

```
output "connection_string" {  
    value = module.rockroach-dedicated.connection_string  
}
```

- Alongside the connection string, in order to connect to your cluster, you'll also need to output the cluster certificate, you can output this as a file within the terraform with the code snippet below.

```
resource "local_file" "cluster_cert" {  
    filename = "${path.root}/cert.pem"  
    content = data.rockroach_cluster_cert.rockroach.cert  
}
```

- If required you can also output this as a variable or text output using the output snippet below:

```
output "cert" {  
    value = data.rockroach_cluster_cert.rockroach.cert  
}
```

## Step 5: Customer Managed Encryption Keys (CMEK)

Customer-Managed Encryption Keys (CMEK) allow you to protect data at rest in a CockroachDB Dedicated advanced private cluster using a cryptographic key that is entirely within your control, hosted in a supported cloud provider key-management system (KMS). This key is called the CMEK key.

You can manage your CMEK keys using one or more of the following services:

- Amazon Web Services (AWS) KMS
- Google Cloud Platform (GCP) KMS

## Step 6: Maintenance Windows

- Within Cockroach Cloud you can view and manage the patch upgrade window for your cluster.
- To help keep your clusters updated while minimizing disruption and downtime, set a window of time when your cluster is experiencing the lowest traffic.
- You are also able to configure this using Terraform.

```
resource "cockroach_maintenance_window" "example" {  
    id          = cockroach_cluster.cockroach.id  
    offset_duration = var.offset_duration  
    window_duration = var.window_duration  
}
```

## Step 7: Operations with Datadog

- Cockroach Cloud allows for the export of metrics to external monitoring tools like Datadog.
- By doing this you can collate metrics for other sources to give you an end to end view of the service you are trying to deliver.
- This is critical in maintaining your prescribed SLA to your customers, whoever that may be.
- This could be external users of the platform or an internal system where the customers are internal members of staff. To configure and manage metrics export for your CockroachDB Dedicated cluster, use the metricexport endpoint for Datadog.
- Access to the metricexport endpoints requires a valid CockroachDB Cloud service account with the appropriate permissions (admin privilege or Cluster Admin role). To configure this using Terraform use the following code example to help.

```
resource "cockroach_metric_export_datadog_config" "example" {  
    id      = cockroach_cluster.cockroach.id  
    site    = var.datadog_site  
    api_key = var.datadog_api_key  
}
```

## Step 8: Initialize your Terraform

- To prepare our directory containing the terraform code we have just cloned from the GitHub repository we must run `terraform init`.
- By running this command it prepares our directory with all the required components that are defined in our code.
- For example downloading any Terraform providers or modules that are externally hosted. In this blog that will be the CockroachDB Cloud provider.

`terraform init`

## Step 9: Check your planned outcome

Now we are able to move to the next stage which is to run a `terraform plan`. The `terraform plan` command creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure. When Terraform creates a plan it:

- Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.
- Compares the current configuration to the prior state and notes any differences.
- Proposes a set of change actions that should, if applied, make the remote objects match the configuration.

`terraform plan`

## Step 10: Deploy your infrastructure

- To build our cluster there is one final step to complete. Once we are happy with our terraform plan and the resources that it is going to create, then terraform apply can be executed.
- `terraform apply -auto-approve`
- This will create all the resources that are defined in the terraform code in the repository. It will now take a number of minutes as Terraform instructs the APIs of AWS and Cockroach Cloud.