**Continuous integration and testing pipeline**

**Version 1.0**

**Prepared by:**

1. **Marcus Nogueira**

**Supervised by:**

**Dr. Mahbub Ahmed**

**July 2021**

Table of Contents

[a) Implement Source Control Using Git and Github 3](#_Toc77733216)

[1) Create a free GitHub account if you do not have one 3](#_Toc77733217)

[2) Create a simplistic application in your favourite programming language and push into GitHub. 3](#_Toc77733218)

[3) Create three branches from your repository named ‘feature-x’, ‘feature-y’, and ‘feature- z’. 4](#_Toc77733219)

[4) From each branch modify one of the source files so that they would lead to merge conflicts. 5](#_Toc77733220)

[5) Commit your changes in each branch and push each of the branches into GitHub 6](#_Toc77733221)

[6) In GitHub, create a pull request to merge ‘feature-x’ branch into the master branch and approve the pull request. 6](#_Toc77733222)

[7) From Git CLI, merge the updated master branch with ‘feature-y’ branch. Resolve the conflict and push it into GitHub 7](#_Toc77733223)

[8) In GitHub, create a pull request to merge ‘feature-z’ branch into the master branch. Resolve the conflict and complete the merge. 8](#_Toc77733224)

[9) Commit your changes and push the master branch into GitHub. 10](#_Toc77733225)

[b)  Implement Unit testing 10](#_Toc77733226)

[1) Create a branch named ‘unit-test’ and checkout the newly created branch. 10](#_Toc77733227)

[2) In your chosen Unit testing framework, write five test cases for your applications such that three test cases pass and two test cases fail. Modify your applications if required. 11](#_Toc77733228)

[3) Add unit test files to your repository, commit the change and push the branch into GitHub. 14](#_Toc77733229)

[4) In GitHub, create a pull request to merge ‘unit-test’ branch with the master branch and complete the merge. 14](#_Toc77733230)

[c)  Implement Build automation 16](#_Toc77733231)

[1) In your chosen build automation tools create a script to compile and test your application as well as making a deployable package or executable for your application. 16](#_Toc77733232)

[2) Create a ‘Readme.txt’ file with instructions on how to run the build script. 19](#_Toc77733233)

[3) Add the build script and the Readme.txt file to your master branch. Commit the changes and push the changes into GitHub. 20](#_Toc77733234)

[d)  Write a report on Infrastructure as Code (IaC) 21](#_Toc77733235)

[1) The report shall contain the link to your GitHub repository. 21](#_Toc77733236)

[2) Explain the concept of Infrastructure as Code (IaC) and its benefits in the context of DevOps. 21](#_Toc77733237)

[e)  References 22](#_Toc77733238)

# Implement Source Control Using Git and Github

## Graphical user interface, application, Teams Description automatically generatedGraphical user interface, text, application Description automatically generatedCreate a free GitHub account if you do not have one

("Hello World · GitHub Guides", 2021)

## 2) Create a simplistic application in your favourite programming language and push into GitHub.

To push a code to a Github repository we need to:

* 1. Create a repo – Repository Laravel\_app\_ass2\_dot created.

Graphical user interface, website

Description automatically generated

* 1. Push the code, I’ve used the sequence below:
* **git init** – To initialize the local directory as a Git Repository
* **git add** . – Adds the files in the local repository and stages them for commit.
* **git commit -m "first commit"** - Commits the tracked changes and prepares them to be pushed to a remote repository.
* **git branch main** - The git branch command can be used to create a new branch. In my case I’ve created the branch main
* **git remote add origin** [**git@github.com:mvsnogueira-dnx/laravel\_app\_ass2\_DOT.git**](mailto:git@github.com:mvsnogueira-dnx/laravel_app_ass2_DOT.git) - I’ve created a new connection to my remote repository on Github.
* **git push -u origin main** - Pushes the changes in your local repository up to the remote repository you specified as the origin.

**Follow the link of my repository:** https://github.com/mvsnogueira-gmail/laravel\_app\_ass2\_DOT

Graphical user interface, text

Description automatically generated

## 3) Create three branches from your repository named ‘feature-x’, ‘feature-y’, and ‘feature- z’.

I’ve used the sequence below:

* 1. Created branch feature-x:
     1. **git branch feature-x** – Create the branch feature-x
     2. **git push origin feature-x** – Push the code to feature-x
  2. Created branch feature-y:
     1. **git branch feature-x** – Create the branch feature-y
     2. **git push origin feature-x** – Push the code to feature-y
  3. Created branch feature-z:
     1. **git branch feature-x** – Create the branch feature-z
     2. **git push origin feature-x** – Push the code to feature-z

("Hello World · GitHub Guides", 2021)

Graphical user interface, application

Description automatically generated

## 4) From each branch modify one of the source files so that they would lead to merge conflicts.

1. **git branch "feature-x"** - I’ve connected to branch feature-x
2. I’ve updated the file **file-modified-feature-x** on my local machine with the message “file updated”

**Graphical user interface, text, application

Description automatically generated**

I’ve done the same steps on branches feature-y and feature-z.

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

## 5) Commit your changes in each branch and push each of the branches into GitHub

I’ve used the sequence below:

* 1. **git branch "feature-x"** - I’ve connected to branch feature-x
  2. **git add** . - Adds the files in the local repository and stages them for commit
  3. **git commit -m "Modifying file file-modified-feature-x** to simulate merge conflict.
  4. **git push origin feature-x** – To Push the changes in my local repository up to the remote repository you specified as the origin.

I’ve done the same steps on branches feature-y and feature-z.

("Hello World · GitHub Guides", 2021)

## 6) In GitHub, create a pull request to merge ‘feature-x’ branch into the master branch and approve the pull request.

Graphical user interface, text, application, email

Description automatically generated

**To approve:**

1. Under your repository name, click Pull requests.
2. On the pull request, click Files changed.
3. Review the changes in the pull request, and optionally, comment on specific lines.
4. Above the changed code, click Review changes.
5. Type a comment “Update will be approved”
6. Select Approve to approve merging the changes proposed in the pull request.
7. Click Submit review

("Hello World · GitHub Guides", 2021)

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, application

Description automatically generated with medium confidence

## 7) From Git CLI, merge the updated master branch with ‘feature-y’ branch. Resolve the conflict and push it into GitHub

1. git branch main – To connect on main branch
2. git merge feature-y – To merge updates from feature-y to the main branch.

**Text

Description automatically generated**

1. I chose to accept the change made by feature-y

Graphical user interface, text

Description automatically generated

1. I chose to accept the change made by feature-y
2. git add .
3. git commit -m “Updating merge conflicts”
4. git push origin main

Text

Description automatically generated8)

## 8) In GitHub, create a pull request to merge ‘feature-z’ branch into the master branch. Resolve the conflict and complete the merge.

Graphical user interface, text, application, email

Description automatically generated

1. To fix click on Resolve conflicts
2. Fix the conflict

Graphical user interface, text, application, email

Description automatically generated

1. Click on Marked as Resolved

Graphical user interface, text, application, email

Description automatically generated

1. Click on Commit the merge

Graphical user interface, text, application, Teams

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## 9) Commit your changes and push the master branch into GitHub.

1. Click on Merge the Pull request

Graphical user interface, text, application, Teams

Description automatically generated

1. Confirm the merge

Graphical user interface, text, application, email

Description automatically generated

# b)  Implement Unit testing

## 1) Create a branch named ‘unit-test’ and checkout the newly created branch.

1. **git branch unit-test –** Create branch unit-test
2. **git checkout unit-test –** Switch to branch unit-test

## 2) In your chosen Unit testing framework, write five test cases for your applications such that three test cases pass and two test cases fail. Modify your applications if required.

For the application tests I've created two directories: Feature and Unit. Unit tests concentrate on a small, isolated section of your code. The tests in the "Unit" test directory do not start the Laravel application, so they can't access the database or other framework services.

Generally, most of your tests should be feature tests. Feature tests may test a more significant portion of your code, including how several objects interact with each other or even a complete HTTP request to a JSON endpoint. **These types of tests provide the most confidence that your system as a whole is functioning as intended.**

("Laravel testing tips & tricks", 2021)

1. **Simple HTTP response test - test the Laravel welcome screen.**

**Let’s check if its displayed correctly using a simple test case.**

Text

Description automatically generated

Run the following command in the Laravel root folder to execute this test case.

$ ./vendor/bin/phpunit ./tests/Feature/ExampleTest.php

Graphical user interface, text, application

Description automatically generated

1. **View test - Let’s check if its displayed correctly using a simple test case.**

With the previous test we can’t test whether the Welcome screen is displayed and what contents it has.

Modify the code as following,

Text

Description automatically generated

assertViewIs() to test what view was used in the method.

In addition we can use assertSeeUsing() method, which is to check if specified string is found in the welcome screen.

Text

Description automatically generated

("Creating & Running Tests with Laravel", 2021)

**3. Test login - From here we will test screens with different results depending on whether the user is logged in or not.**

Modify the code as following,

* Using the first user record from database.

**A screenshot of a computer

Description automatically generated with medium confidence**

**4. Test login -** Creating a new user and using that for testing.

**Text

Description automatically generated**

**5. Using faker -** Faker is a useful library to generate fake data quickly and easily. For example say you would like a real sounding name, address, company name you can simply say faker->name and it will pick one at random.

**Text

Description automatically generated**

("HTTP response status codes - HTTP | MDN", 2021)

## 3) Add unit test files to your repository, commit the change and push the branch into GitHub.

1. **git branch "unit-test"** - I’ve connected to branch unit-test
2. **git add** . - Adds the files in the local repository and stages them for commit
3. **git commit -m "Creating test cases**" - Committing the test cases.
4. **git push origin unit-test** – To Push the changes in my local repository up to the remote repository you specified as the origin.

**Text

Description automatically generated**

## 4) In GitHub, create a pull request to merge ‘unit-test’ branch with the master branch and complete the merge.

**1)** Create a Pull Request

Graphical user interface, text, application, email

Description automatically generated

2) Click on Create Pull request

Graphical user interface, text, application, email

Description automatically generated

3) Merge the Pull Request Clicking on Merge Pull request and confirm the merge

Graphical user interface, text, application, email

Description automatically generated

# c)  Implement Build automation

## In your chosen build automation tools create a script to compile and test your application as well as making a deployable package or executable for your application.

Let’s set up GitHub Actions for the repository. Click on the **Actions** tab on the repository navigational menu

Graphical user interface

Description automatically generated with medium confidence

GitHub is smart enough to know that the repository contains PHP (Laravel) code, so we are presented with starter workflows related to those.

Graphical user interface, text, application, chat or text message

Description automatically generated

We’ll be going with the one on Laravel then commit the file. This will create a new .github/workflows/laravel.yml file in the repository.

Click on the Actions tab again, and you should see the list of all workflows similar to the image below:

Graphical user interface, text, application

Description automatically generated

Laravel pipeline will have three jobs:

1. **Laravel-Tests:** To be confident with the code, we need to write tests. For the purpose of this tutorial, we’ll be making use of the default tests that come with a fresh Laravel application.
2. **Build** – This job will build the application, checkout the code, install composer dependencies, install NPM dependencies and compile assets.
3. **Deploy** - Finally, let’s add a job to deploy our application

Graphical user interface, application

Description automatically generated

**The code:**

name: Laravel

on:

push:

jobs:

laravel-tests:

runs-on: ubuntu-latest

steps:

- uses: shivammathur/setup-php@15c43e89cdef867065b0213be354c2841860869e

with:

php-version: '8.0'

- uses: actions/checkout@v2

- name: Copy .env

run: php -r "file\_exists('.env') || copy('.env.example', '.env');"

- name: Install Dependencies

run: composer install --ignore-platform-reqs

- name: Generate key

run: php artisan key:generate

- name: Directory Permissions

run: chmod -R 777 storage bootstrap/cache

- name: Create Database

run: |

mkdir -p database

touch database/database.sqlite

- name: Execute tests (Unit and Feature tests) via PHPUnit

env:

DB\_CONNECTION: sqlite

DB\_DATABASE: database/database.sqlite

run: vendor/bin/phpunit

Build:

needs: [laravel-tests]

runs-on: ubuntu-latest

name: Build

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Cache dependencies

uses: actions/cache@v1

with:

path: ~/.composer/cache/files

key: dependencies-composer-${{ hashFiles('composer.json') }}

- name: Setup PHP

uses: shivammathur/setup-php@v2

with:

php-version: 7.3

extensions: dom, curl, libxml, mbstring, zip, pcntl, pdo, sqlite, pdo\_sqlite

coverage: none

- name: Install Composer dependencies

run: composer install --prefer-dist --no-interaction --no-suggest

- name: Install NPM dependencies

run: npm install

- name: Compile assets

run: npm run production

deploy:

needs: [laravel-tests,Build]

runs-on: ubuntu-latest

name: Deploy

steps:

- name: Checkout

uses: actions/checkout@v2

- name: Deployment

uses: appleboy/ssh-action@master

with:

host: ${{ secrets.SSH\_HOST }}

key: ${{ secrets.SSH\_PRIVATE\_KEY }}

username: ${{ secrets.SSH\_USERNAME }}

script: |

cd /var/www/html/

- name: Deploy Laravel

uses: shivammathur/setup-php@v2

with:

host: ${{ secrets.SSH\_HOST }}

key: ${{ secrets.SSH\_PRIVATE\_KEY }}

username: ${{ secrets.SSH\_USERNAME }}

script: |

echo "Application Deployed"

echo "Access website https://laravel.labs2.dnx.host/"

("Features • GitHub Actions", 2021)

**Output of this pipeline:** The application will be accessible via link: <https://laravel.labs2.dnx.host/>

Graphical user interface, application

Description automatically generated

## 2) Create a ‘Readme.txt’ file with instructions on how to run the build script.

Readme created and can be accessed via <https://github.com/mvsnogueira-gmail/laravel_app_ass2_DOT/blob/main/README.md>

Graphical user interface, application, Teams

Description automatically generated

## 3) Add the build script and the Readme.txt file to your master branch. Commit the changes and push the changes into GitHub.

1. **git add** . - Adds the files in the local repository and stages them for commit
2. **git status** – To show the files that have been modified
3. **git commit -m "** **Creating Readme and Github Action Pipeline**" - Committing the test cases.
4. **git push origin main** – To Push the changes in my local repository up to the remote repository you specified as the origin.

Text

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application

Description automatically generated

# d)  Write a report on Infrastructure as Code (IaC)

## 1) The report shall contain the link to your GitHub repository.

Link: <https://github.com/mvsnogueira-gmail/laravel_app_ass2_DOT>

Clone link: [git@github.com:mvsnogueira-gmail/laravel\_app\_ass2\_DOT.git](mailto:git@github.com:mvsnogueira-gmail/laravel_app_ass2_DOT.git)

## 2) Explain the concept of Infrastructure as Code (IaC) and its benefits in the context of DevOps.

Infrastructure as code, or IaC, is an IT practice that codifies and maintains the infrastructure that supports it. Instead of manually configuring separate hardware devices and operating systems, infrastructure as code allows developers and operations teams to manage, monitor, and provision resources automatically.

Higher-level or descriptive language is used to design more versatile and adaptive provisioning and deployment procedures in infrastructure as code.

For example, Ansible, an IT management and configuration tool, has infrastructure-as-code capabilities that can install MySQL servers, verify that MySQL is functioning correctly, create a user account and password, set up a new database, and remove excessive databases.

The infrastructure automation method based on code is quite similar to software design practices. Development teams closely monitor code versions, test iterations, and deploy software only when thoroughly tested and authorised for production.

Infrastructure as code has several advantages.

* **Efficiency and speed:** Provisioning and management that is automated are both faster and more efficient than manual operations.

IaC also has code that scales automatically (adds or shuts down environments and resources when they are no longer needed).

* **Consistency**: Instead of relying on system administrators, software developers can utilise code to supply and deploy servers and apps according to business processes and standards in a DevOps environment.

Before operations take over for live deployment in production, a developer might build a configuration file to provision and deploy a new application for quality assurance or experimental deployment.

* **Affiliation with DevOps:** Because the infrastructure is written in code, it can go through the same version control, automated testing, and other CI/CD processes that developers do for application code.

IT teams can use infrastructure as code to transfer a data centre's physical resources and services, as well as discrete configurations, into code to automatically provide and manage such resources.

* **Stability:** Changing the wrong configuration or deleting the wrong resource in the web console can cause problems.

Infrastructure as code (IaC) can help with this, especially when integrated with version control software like Git.

* **Scalability:** Infrastructure as code allows you to write it once and reuse it several times.

Despite its advantages, infrastructure as code has several drawbacks:

* There is a risk of configuration drift without additional change management tools if administrators modify server configurations outside of the specified infrastructure-as-code template.
* Infrastructure as code must be thoroughly integrated into systems administration, IT operations, DevOps techniques, and well-documented rules and procedures.
* Another disadvantage of infrastructure as code is that it puts additional pressure on developers to build efficient code that works in production conditions.

("What is Infrastructure as Code? - Azure DevOps", 2021)

# e)  References

Creating & Running Tests with Laravel. (2021). Retrieved 20 July 2021, from <https://medium.com/@mazraara/creating-running-tests-with-laravel-698c429a647b>

Features • GitHub Actions. (2021). Retrieved 20 July 2021, from <https://github.com/features/actions>

HTTP response status codes - HTTP | MDN. (2021). Retrieved 20 July 2021, from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Hello World · GitHub Guides. (2021). Retrieved 20 July 2021, from https://guides.github.com/activities/hello-world/

Laravel testing tips & tricks. (2021). Retrieved 20 July 2021, from <https://medium.com/star-gazers/laravel-testing-tips-tricks-fc69b1f02d5a>

Peck, N. (2021). Why use infrastructure as code?. Retrieved 20 July 2021, from <https://containersonaws.com/introduction/infrastructure-as-code/>

What is Infrastructure as Code? - Azure DevOps. (2021). Retrieved 20 July 2021, from https://docs.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code