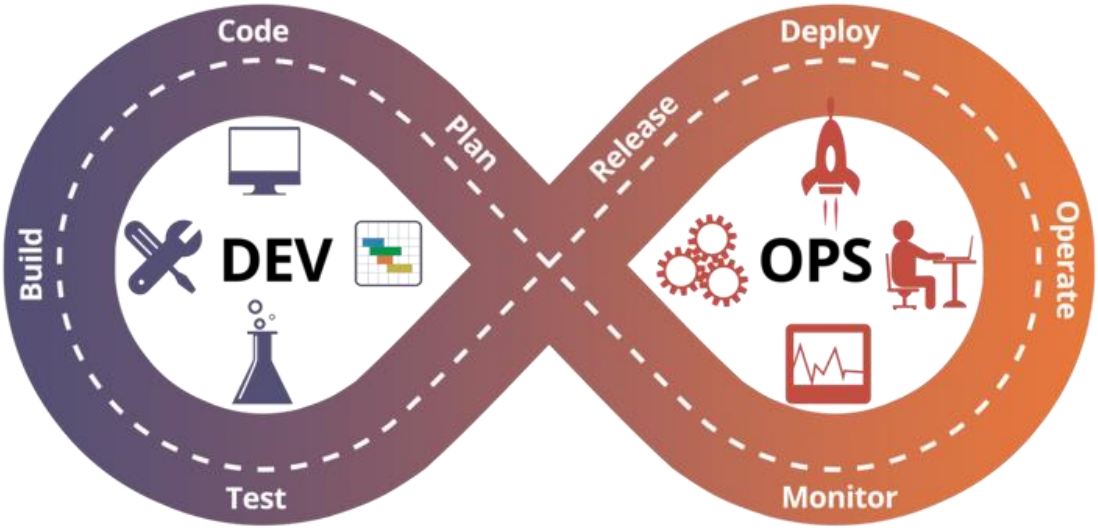
E-COMMERCE WEB APP

Demonstrates DevOps Pipeline with selected tools on E-Commerce Web App



APRIL 30, 2019

Institute of Technology, Tralee
B.Sc. (Hons) in Computing with Software Development

Table of Contents

Table of Figures

Section 1: DevOps Pipeline Flow Diagram

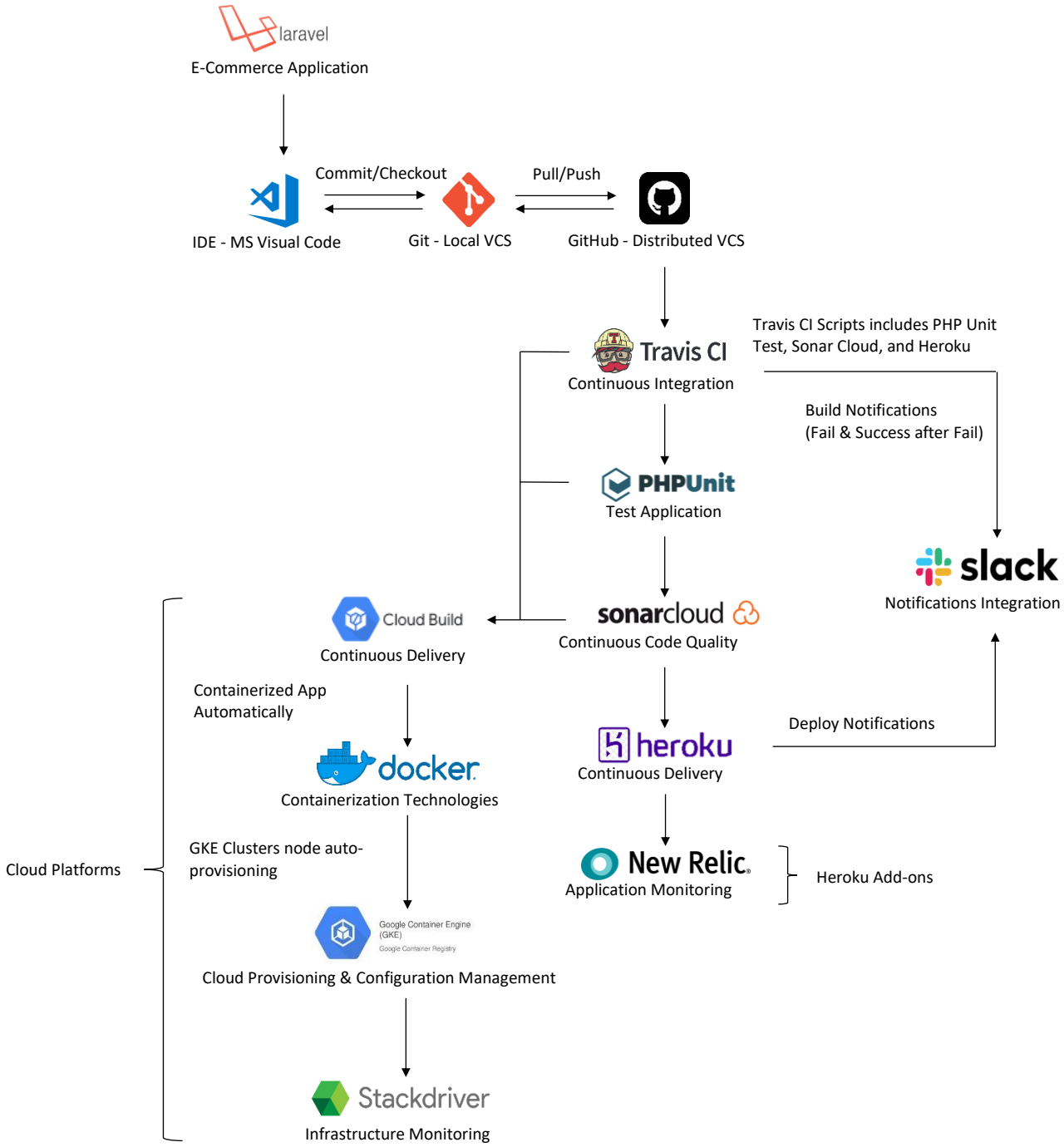


Figure 1: DevOps Tools Pipeline Flow Diagram.

Section 2: Introduction code and tools

The chosen application for development operations and application stores was an online commerce Laravel PHP and VueJS application. The primary author of this application is by Neo Ighodaro and were forked by Fisayo Afolayan for PHPUnit testing. The application was cloned from Fisayo Afolayan and is renamed to ecommerce-devops. Table below shows summary of the author that implemented the online commerce application.

Primary Author:	Neo Ighodaro https://github.com/neoighodaro-articles/e-commerce-laravel-vue
Author	Fisayo Afolayan
GitHub ID	fisayoafolayan
Location	Hamburg, Germany
GitHub Link	https://github.com/fisayoafolayan/e-commerce-laravel-vue
Laravel Version	5.8
Last update	2018
Tutorial Blog	https://blog.pusher.com/author/neo/ https://blog.pusher.com/tests-laravel-applications/

My GitHub	https://github.com/dejongyeong/ecommerce-devops
GitHub ID	dejongyeong
Author	De Jong Yeong

A blog was written on the implementation of e-commerce application using Laravel and VueJS and guides on how to write PHPUnit test. Links to the blog were mentioned in table above. The application contains several features like user login and signup feature and checkout integration et cetera. The application allows user to insert shopping products into shopping cart and proceeds to checkout for payment.

A screenshot of the landing page of e-commerce application is shown below.

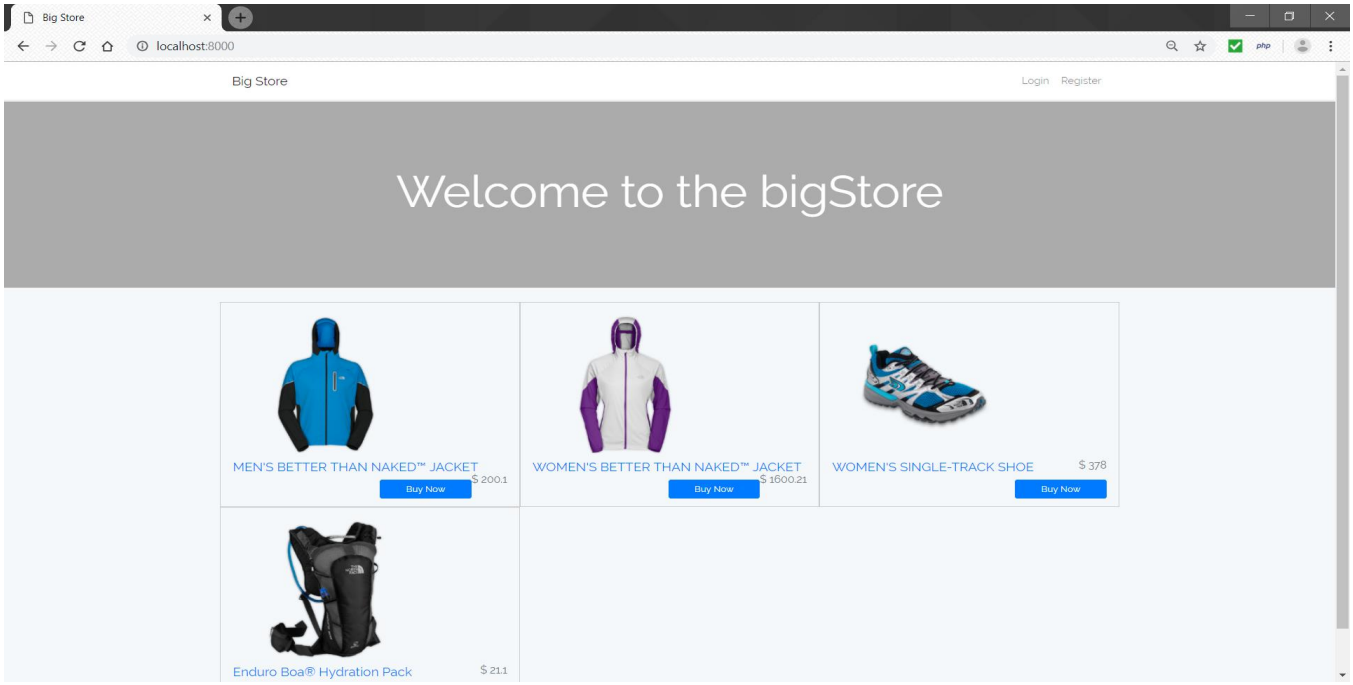


Figure 2: Landing page of Laravel VueJS e-commerce screenshot.

Listed below shows a summary of the e-commerce application:

- Create orders.
- Deliver orders.
- Update orders.
- Delete orders.
- Create products.
- Retrieve all products.
- Update products.
- Delete products.
- Upload product images.
- Login.
- Signup.

Microsoft Visual Studio Code is used for adding additional codes in cloned e-commerce application. Visual Studio Code is a source-code editor which includes support for debugging, syntax highlighting, intelligent code completion and most important an embedded Git control. Visual Studio Code is an open-source integrated development environment (IDE) for developers and can be downloaded from https://code.visualstudio.com/. Plugins can be installed in Visual Studio Code for Laravel PHP development as shown in screenshot below.

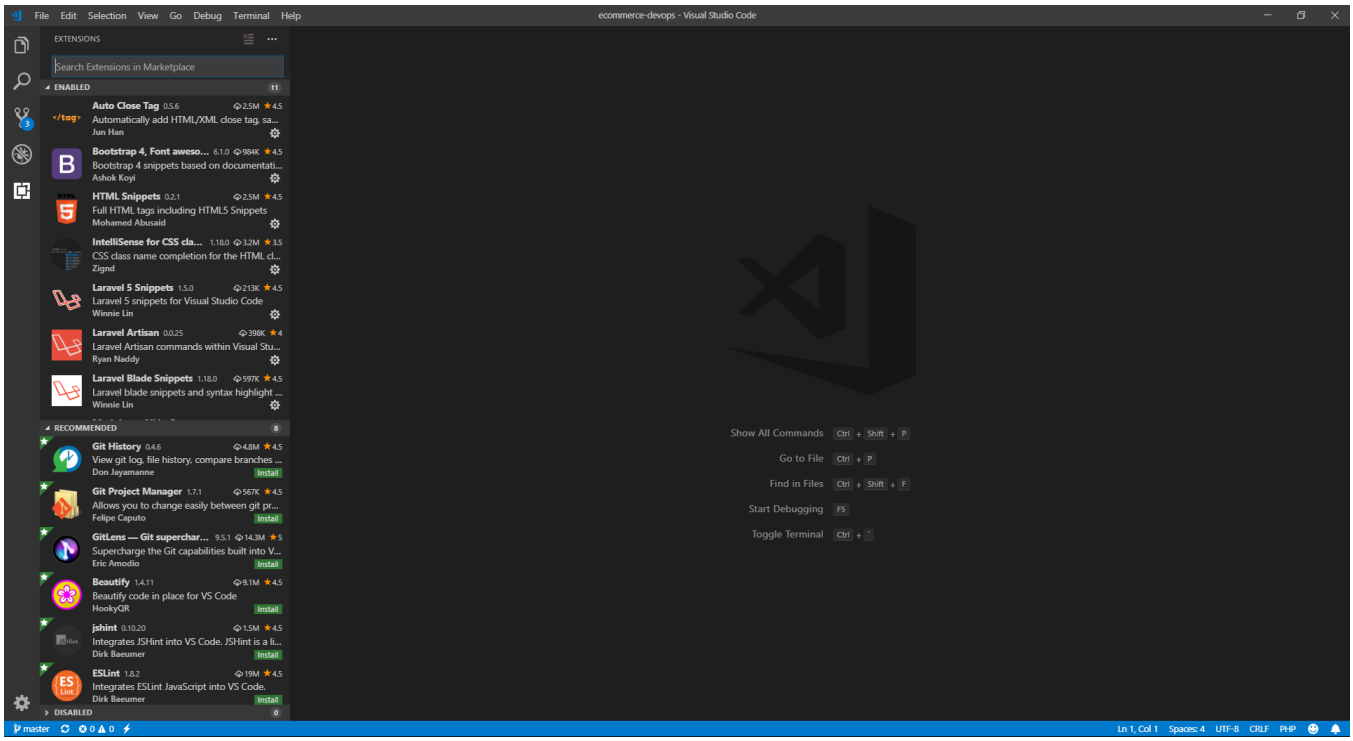


Figure 3: Microsoft Visual Studio Code extension page.

2.1 Laravel E-Commerce Installation Guide

Laravel application uses composer to manage dependencies. It is important to install composer on the system before Laravel is installed. The latest version 1.8.4 of composer can be downloaded and installed from https://getcomposer.org/download/. Path of the composer is added into PATH environment variable which allows user to call composer command from any directory. Tutorial on how to add path to environment variables is shown in table below.

Demo Link:
https://docs.telerik.com/teststudio/features/test-runners/add-path-environment-variables

The composer was already installed in the system and is required to update composer version 1.7.2 to the latest version of 1.8.4 as shown below.

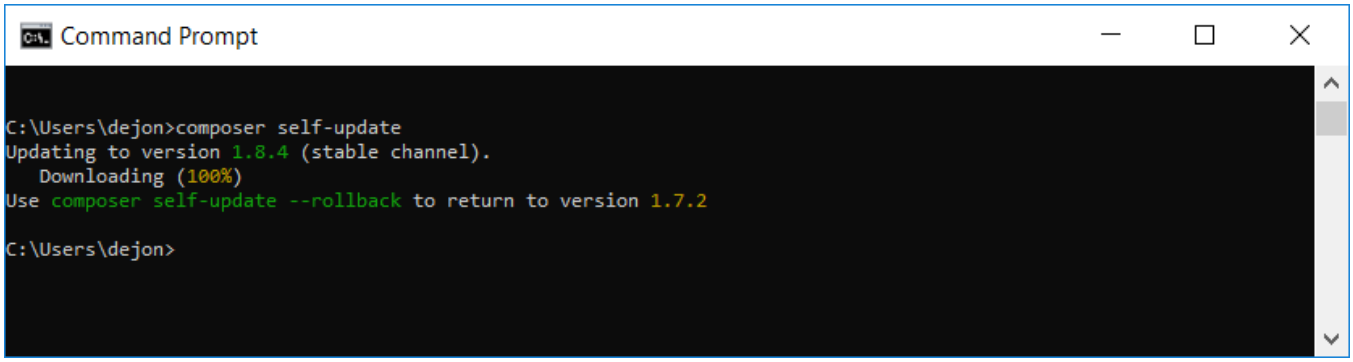


Figure 4: Update composer to the latest version 1.8.4.

An installation guide is documented for developers to run the application locally. The installation guide was copied from his GitHub repository and was included below.

Step 01: Clone the repository and change directory (cd) into it.

Step 02: Run composer install to install all dependencies of the application.

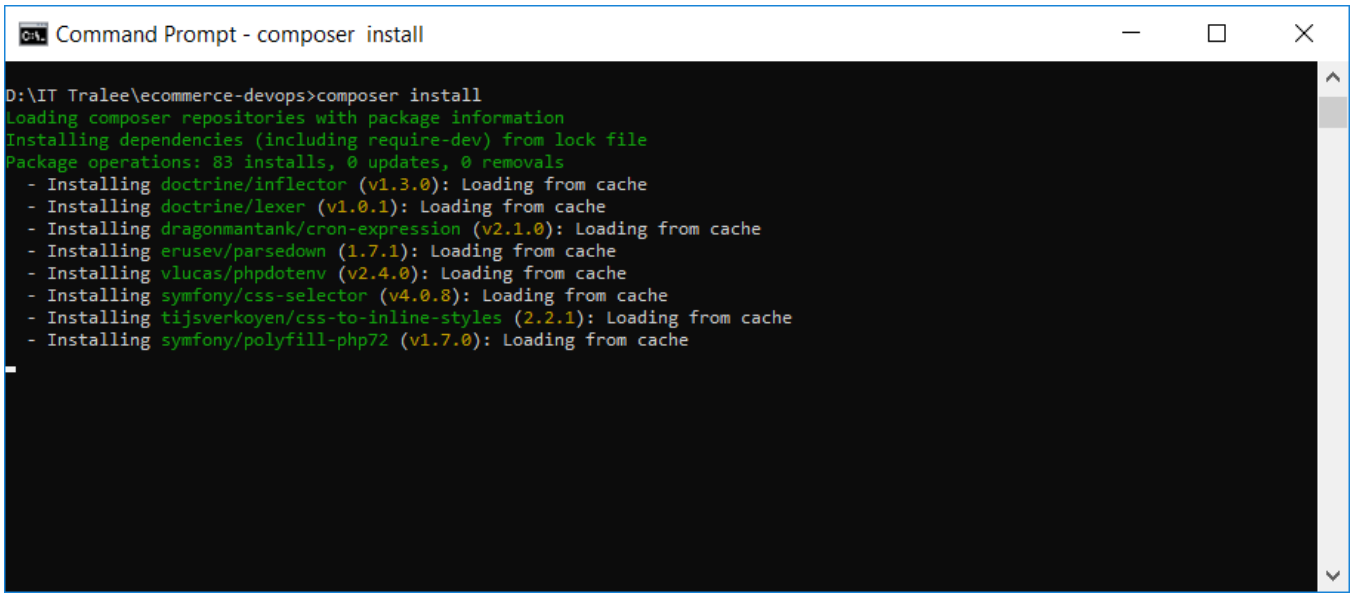


Figure 5: Run composer install command to install all dependencies.

Step 03: Copy and rename .env.example file to .env file. The .env environment file is ignored when changes is pushed to Git repository. It was used to manage configuration values based on application environment.

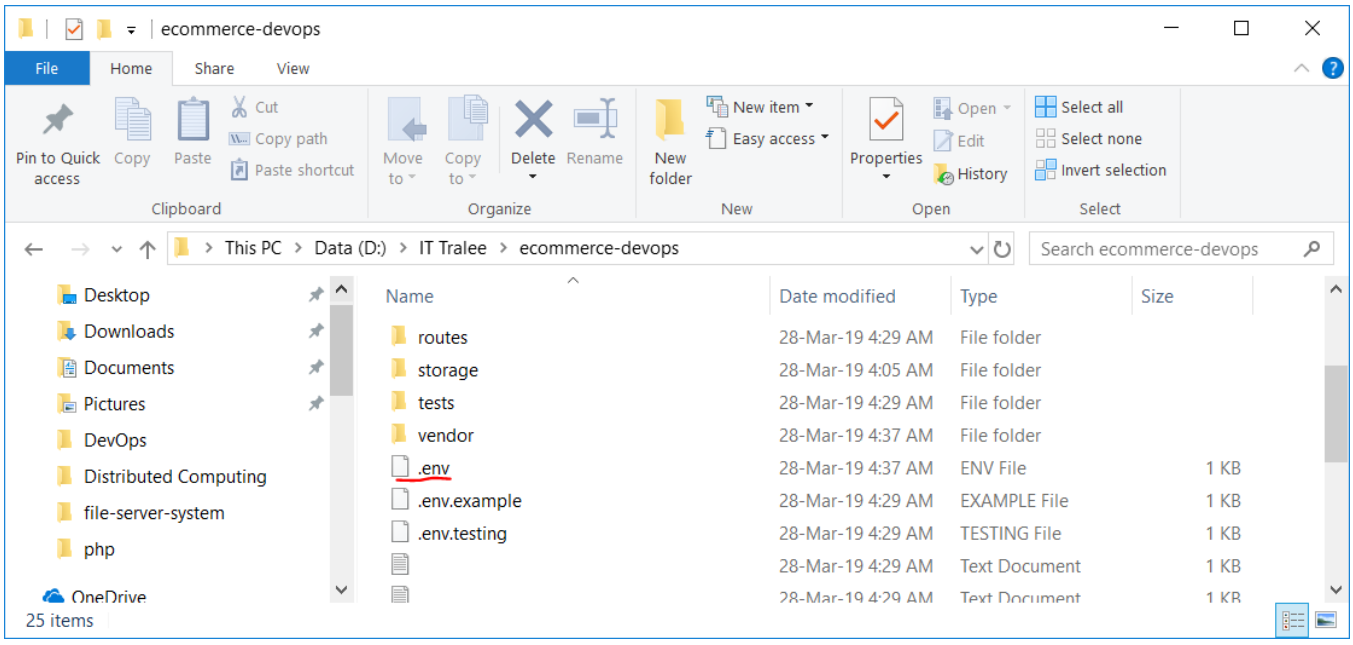


Figure 6: Copy and rename .env.example file to .env.

Step 04: Run php artisan key:generate to set generate a unique application key.

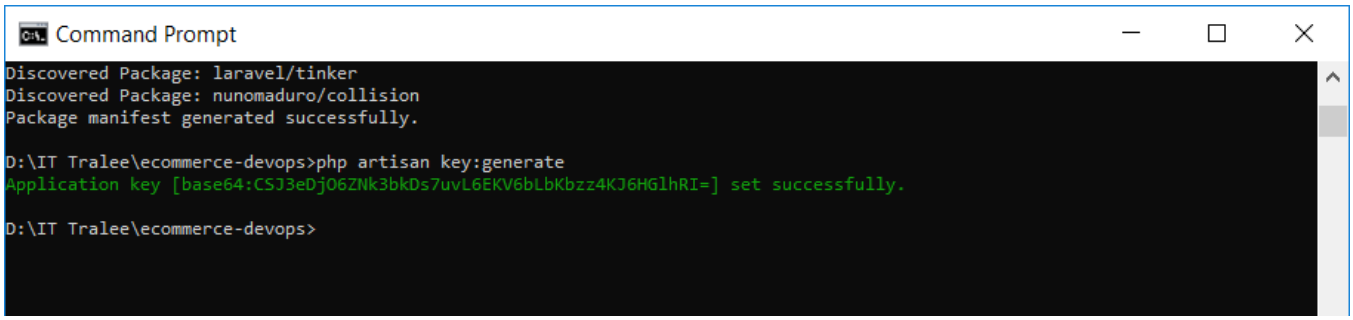


Figure 7: Run php artisan key:generate to create unique application key.

Step 05: Configure database credentials in the .env file. The database configured can be either in-memory database like SQLite or other databases like MySQL et cetera.

Please note that it is recommended to use local database for local development and in-memory database for unit testing. It is also recommended to use a cloud provisioning database in production. Tutorial for PostgreSQL database configuration is shown in Appendix A.

Step 06: Run php artisan migrate --seed to migrate database and seeds data.

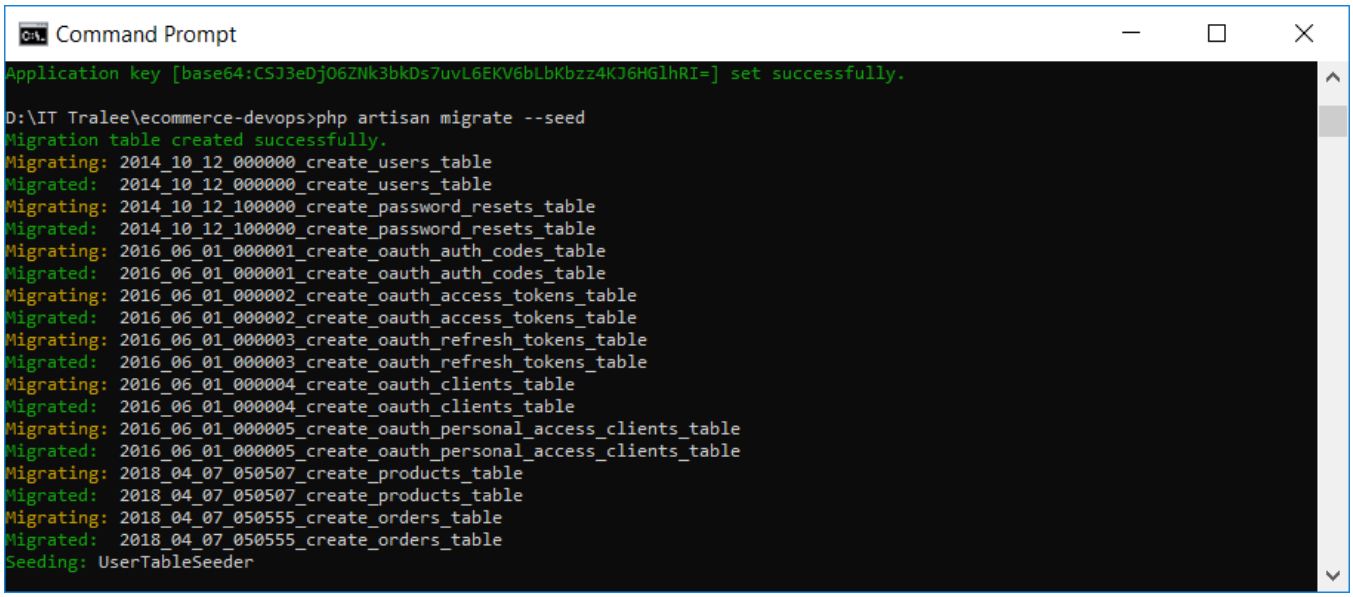


Figure 8: Run php artisan migrate --seed to migrate database and seeds data.

Step 07: Run php artisan passport:install to create encryption keys that are needed to generate secure access tokens.

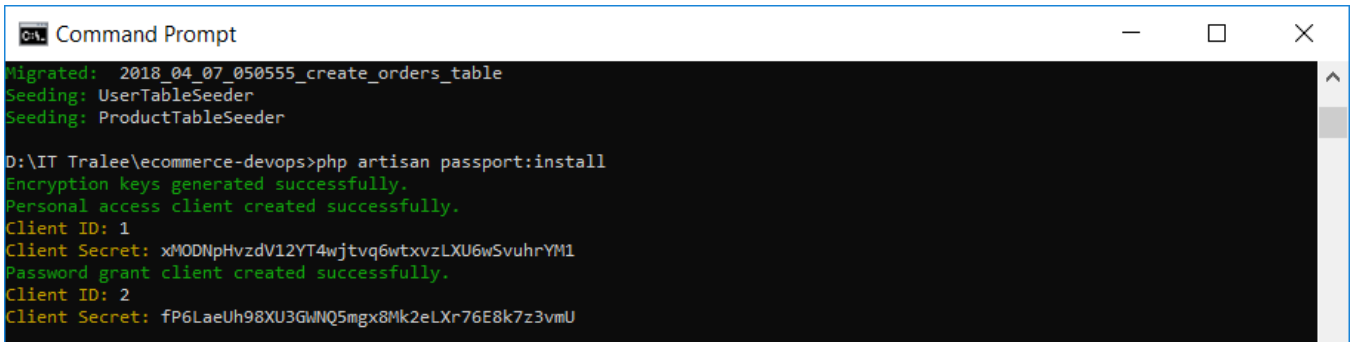


Figure 9: Command to create encryption keys to generate secure access tokens.

Step 08: Change the version of vue and vue-template-compile to 2.5.16 to resolve version mismatch issue as shown below.

Figure 10: Change version to resolve version mismatch issue.

Step 08: Run npm install to install all npm packages from command prompt.

Step 09: Run npm install popper.js --save to install popper for bootstrap.

Step 10: Run npm install vue-router --save to install vue router to resolve issues.

Step 11: Run npm run development to run all Mix tasks in development environment which is a configuration layer on top of webpack. A successful message will be display if the command runs successfully.

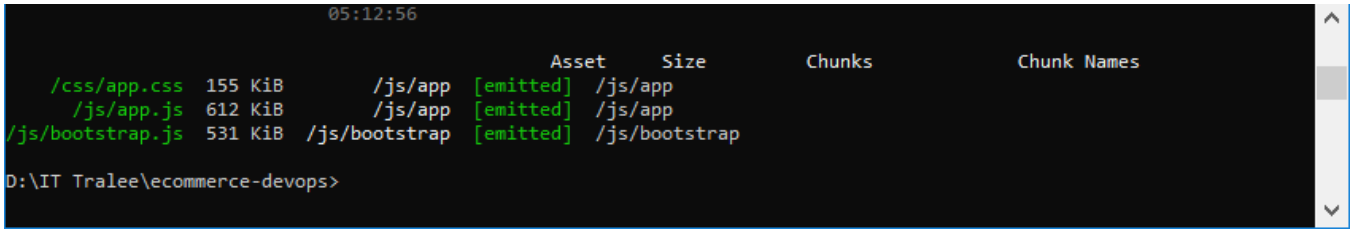


Figure 11: Successful message after running the npm run development command.

Step 12: Run php artisan serve to run application on localhost.

Step 13: Visit localhost:8000 in browser to check for the application as shown in screenshot below.

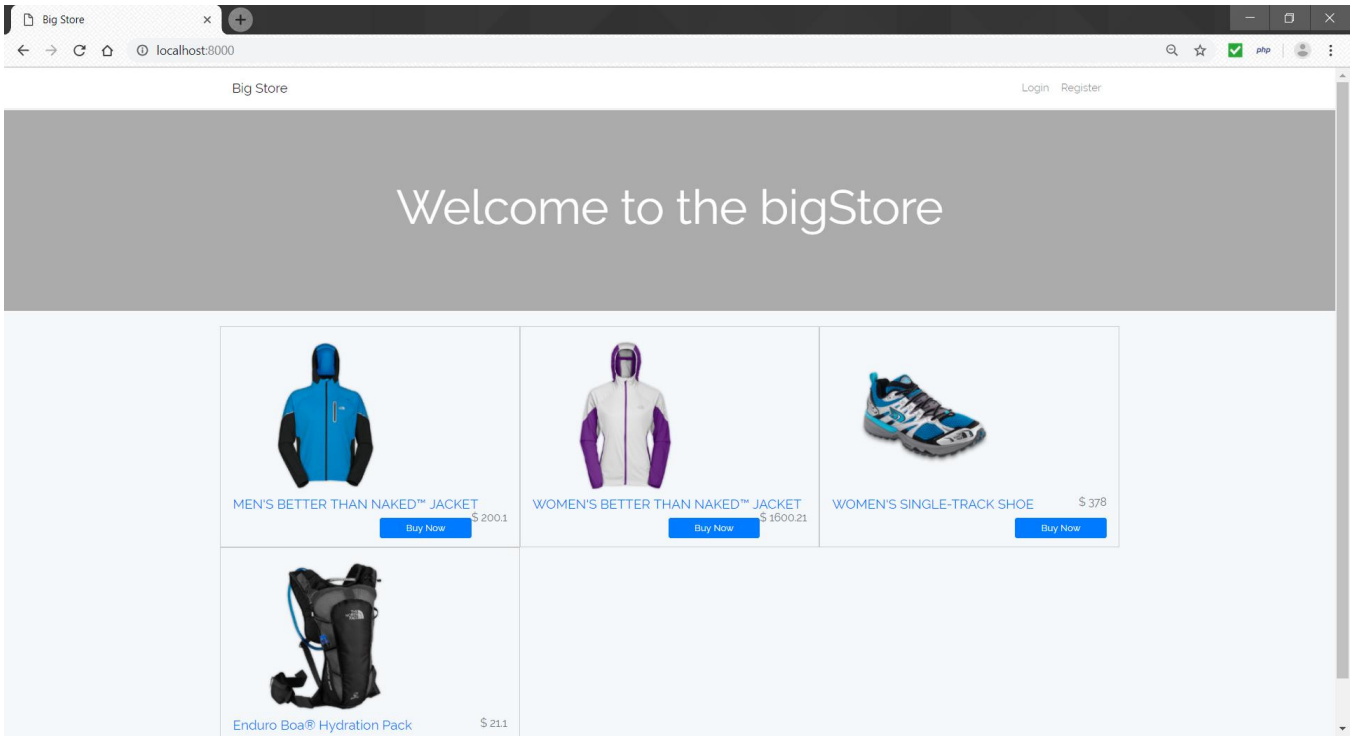


Figure 12: Laravel VueJS e-commerce application runs successfully in localhost.

2.2 Source Control Management

Git in conjunction with GitHub was chosen for source control system. It is a free and open source distributed version control system that are designed to manage projects with speed and efficiency. It comprises a full-fledged repository and version control tracking capabilities independent of a central server or network access. The advantages of source control management using tools like Git are as follow:

- Developers work simultaneously on the same code.
- Code can be recovered from central server if computer crashes.
- Code can be easily reverted to previous version if bug occurs (Soni, 2016).

GitHub is a web-based hosting server for version control using Git. It offers all distributed version control and source code management functionality of Git. GitHub provides access control and collaboration features like task management and bug tracking for projects. Tutorial link to install Git and setting up a GitHub account is shown below (techcrunch.com, 2012). The Git command line interface is installed automatically when Git is installed.

Demo Links
https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

Git contains approximately 40 commands for version control tracking capabilities. These includes create repositories, track local changes with remote changes et cetera. A PDF format of Git cheat summarized by Tobias Gunther in February 2017 is available in link below.

Demo Link:
https://www.git-tower.com/blog/git-cheat-sheet

Tutorial on Git version control:

Step 1: Change directory to Laravel e-commerce application. Please note that changes were made when installing the Laravel E-Commerce application.

Step 2: Run git status command to list untracked changes since the last commit and is denoted in red and will changed to green when files are tracked and staged.

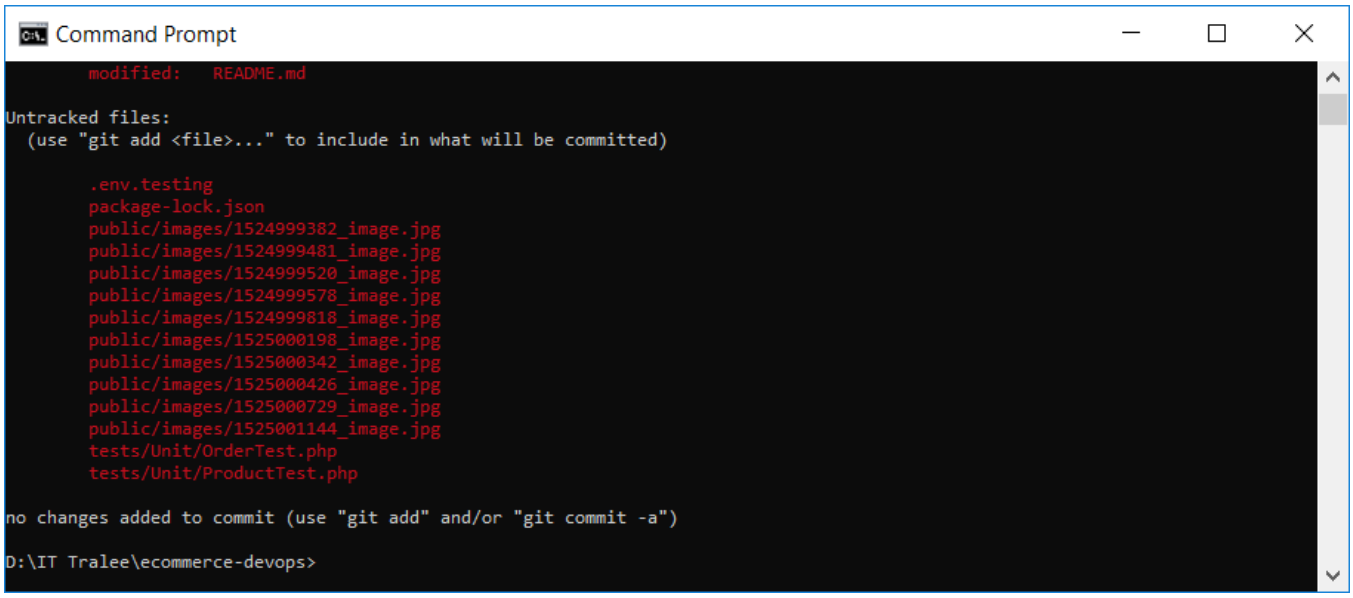


Figure 13: Git command to list untracked changes since the last commit.

Step 3: Run git add . to stage new and modified files which will be denoted in green.

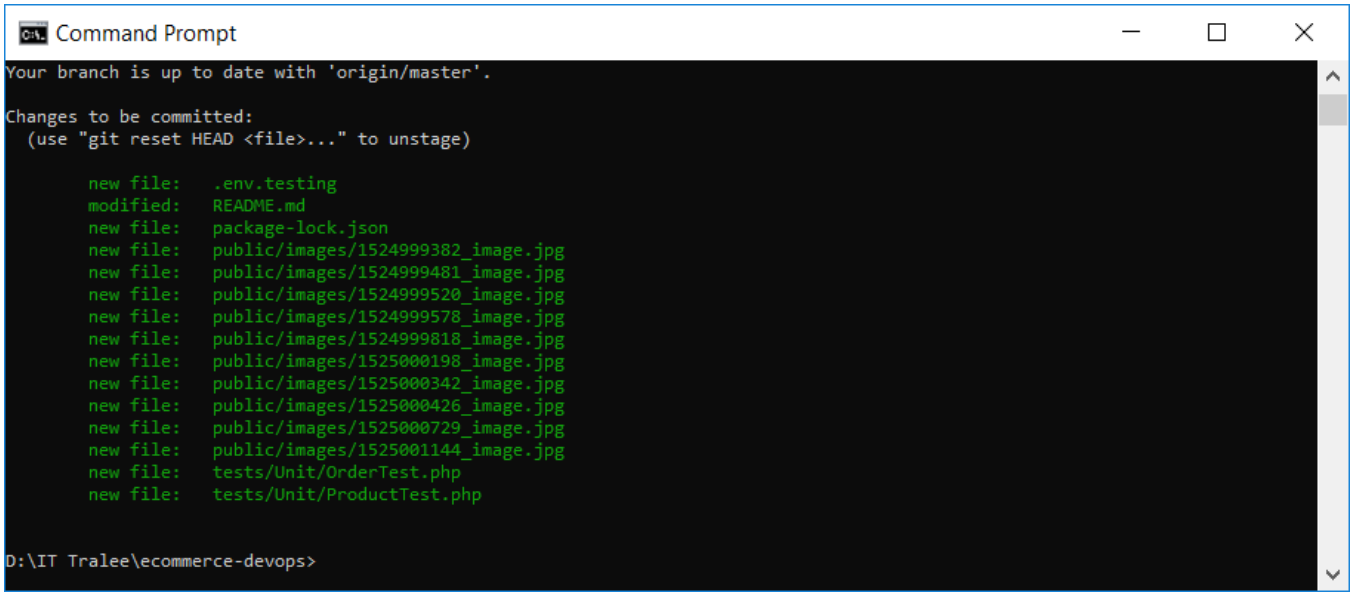


Figure 14: Git command to stage new and modified files.

Step 4: Run git commit -m “message” to write a small description on the committed files and is now tracked as a revision in source control with a unique ID.

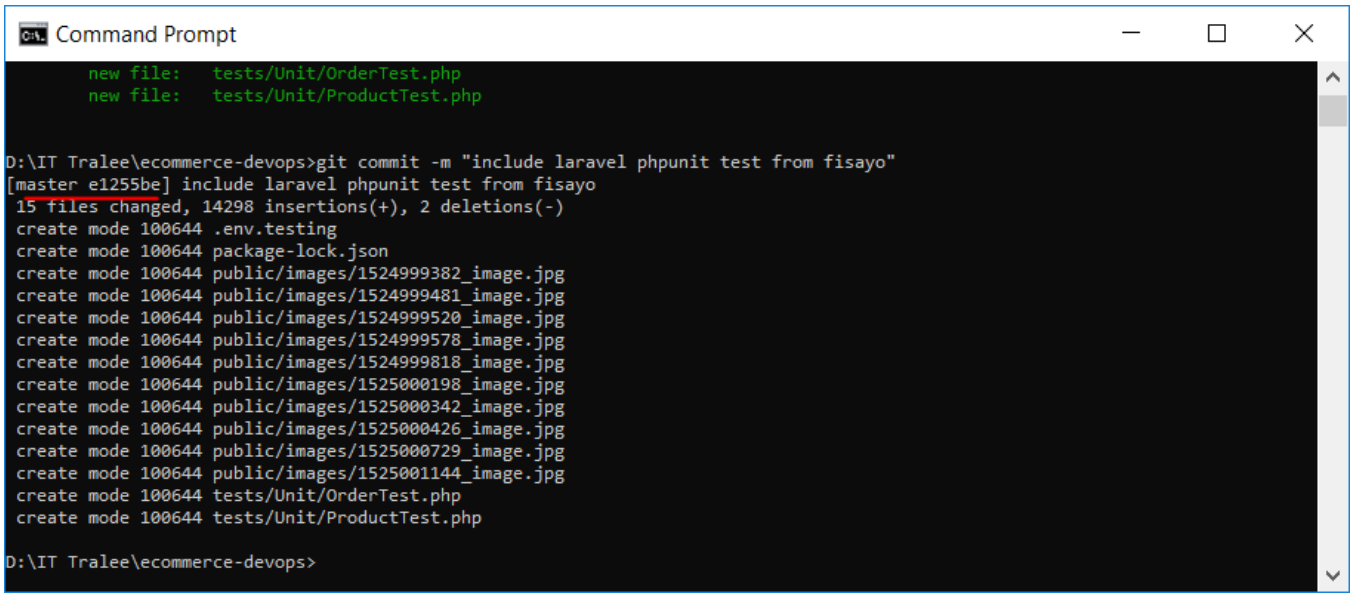


Figure 15: Git command to write small description on the commit and tracked as a revision.

Step 5: Run git push origin to push local changes to hosted Git repository on GitHub.

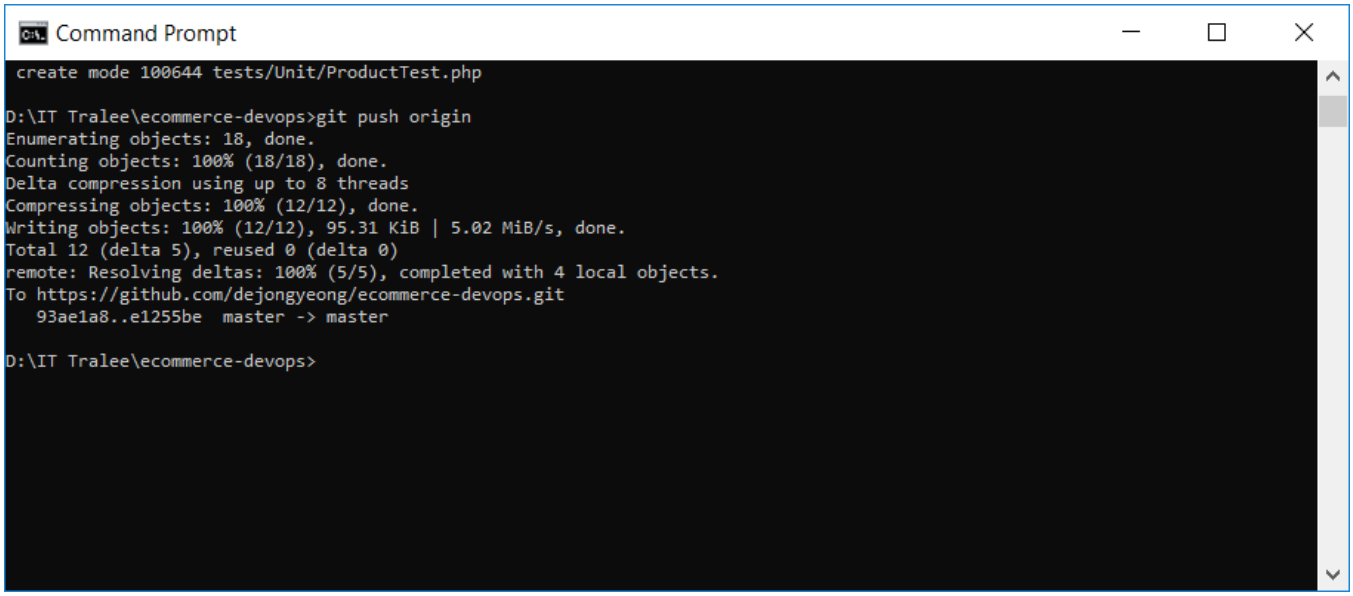


Figure 16: Git command to push local changes to remote server on GitHub.

Tutorial Link:

https://git-scm.com/docs/gittutorial

2.3 PHP Unit Test

Technologies: PHPUnit and Laravel Dusk

Laravel is the most popular PHP framework for application development which provides highly adequate testing features from the testing perspective. Laravel offers two way to test an application:

- Unit testing to test for classes models, controllers et cetera.
- Feature testing to test for code base (cloudways.com, 2018).

Laravel automatically set the configuration environment to test when tests are ran via phpunit. This is because the environment variables were defined in the phpunit.xml file. The session and cache in Laravel were also automatically configured to the array driver which results in no session or cache data will be persisted while testing (laravel.com, 2019).

The Laravel documentation also mentioned that the testing environment variables may be configured in the phpunit.xml file or create a .env.testing file in the root project. It overrides the .env file when running PHPUnit tests.

Tutorial Link:
https://laravel.com/docs/5.8/testing

Several test cases were written by Fisayo in the e-commerce application. Test cases were written and stored in test folder. It contains two unit-test classes in the tests/Unit folder called OrderTest.php and ProductTest.php which tests for order and product CRUD operation. CRUD operation refers to:

- Create operation.
- Read operation.
- Update operation.
- Delete operation.

The application uses Vue Axios library to consume REST API. Axios is a great HTTP client library which uses promises by default and runs on both client and the server. This makes it appropriate for fetching data during server-side rendering. A tutorial link on the fundamentals of Axios is shown below.

Axios Tutorial Link:

https://alligator.io/vuejs/rest-api-axios/

Example tests cases that were written to test the e-commerce application are shown below.

```php
public function testCreateProduct()
{
    $data = [
                'name' => "New Product",
                'description' => "This is a product",
                'units' => 20,
                'price' => 10,
                'image' => "https://images.pexels.com/photos/1000084/pexels-photo-1000084.jpeg?auto=compress&cs=tinysrgb&dpr=2&h=650&w=940"
            ];
    $user = factory(\App\User::class)->create();
    $response = $this->actingAs($user, 'api')->json('POST', '/api/products',$data);
    $response->assertStatus(200);
    $response->assertJson(['status' => true]);
    $response->assertJson(['message' => "Product Created!"]);
    $response->assertJson(['data' => $data]);
}
```

Figure 17: Method to test for product creation with dummy user data.

Test case above in test/Unit ProductTest.php tests for create operation of a product. It uses the factory method to create dummy user data and uses the dummy user data to test for create product using REST API calls. It returns a JSON response which contain details of a product and checks if message status is equals to 200. It also checks if JSON response contains the status variable, a message of "product created" and is the expected data is equals to the actual data.

```php
public function testDeleteOrder()
{
    $user     = factory(\App\User::class)->create();
    $response = $this->actingAs($user, 'api')->json('GET', '/api/orders');
    $response->assertStatus(200);

    $order    = $response->getData()[0];

    $update   = $this->actingAs($user, 'api')->json('DELETE', '/api/orders/'.$order->id);
    $update->assertStatus(200);
    $update->assertJson(['message' => "Order Deleted!"]);
}
```

Figure 18: Method to test for create product in ProductTest.php.

Test case above in test/Unit OrderTest.php tests for delete operation of an order. It uses the factory method to create a dummy user data and uses that data to test for delete order using REST API calls. The method first retrieves the first data returned and calls to DELETE function to delete the selected order. It is expected to return a message code of 200 and should contains a message variable with a value of “Order Deleted!”.

Please note that more unit tests are available in GitHub repository mentioned in Section 2.

Please note that it is recommended to use SQL databases other than SQLite for migration and SQLite in-memory database for testing. Steps to write test cases to test the e-commerce application with PHPUnit is shown below.

Tutorial to test application with PHPUnit is available below:

Tutorial blog on PHPUnit:
https://blog.pusher.com/tests-laravel-applications/

Please note that steps to run PHPUnit test in ecommerce application is shown in Appendix B. Developers might need to clear the application cache and configuration cache during development phases. Please note that commands to clear application cache and configuration cache is detailed in Appendix C.

2.4 Notification Integration

Slack is chosen for notification integration across the development operations and application stores lifecycle. It is an American cloud-based set of proprietary team collaboration tools and services. Slack is a collaboration hub for work and is a place where conversations happen, and decisions are made. It offers an easy-to-use interface for teams to discuss projects and organize projects et cetera. It integrates with various applications such as GitHub, Trello, Microsoft OneDrive et cetera (slack.com, 2019).

Slack and GitHub integration installation guide is shown in Appendix D. Slack conversation offers slash command feature which allow users to deal with issues and pull requests. The best practice of Slack and GitHub integration is to send notification on project implementation issues and pull requests. (Klopp, 2018). Tutorial below outlines steps to start receiving updates about the specified project after signing in to Slack workspace.

Guidelines:
https://it-tralee.slack.com/apps/A8GBNUWU8-github

Step 1: Run /github subscribe [owner/repo] command in Slack to start receiving updates about the project. Click on the **Connect GitHub account** and completes the installation step. A successful message is displayed when Slack is successfully connected to GitHub.

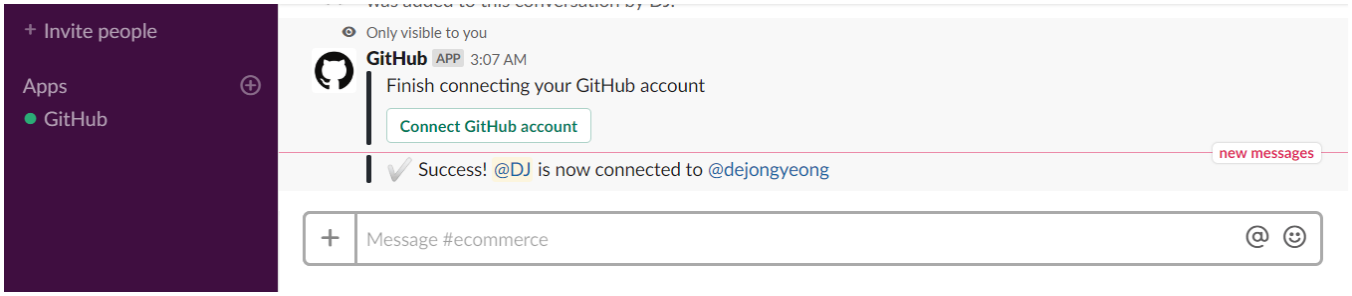


Figure 19: Connect Slack to specified GitHub repository.

Step 2: Run /github [action] [resource] commands to close or reopen and issue or pull request or open a new issue using Slack diagram if necessary.

The link mentioned in table above provides a detailed documentation on what GitHub notifications will be sent to user when issues are closed or opened et cetera. Below shows a screenshot of GitHub notification on Slack ecommerce channel.

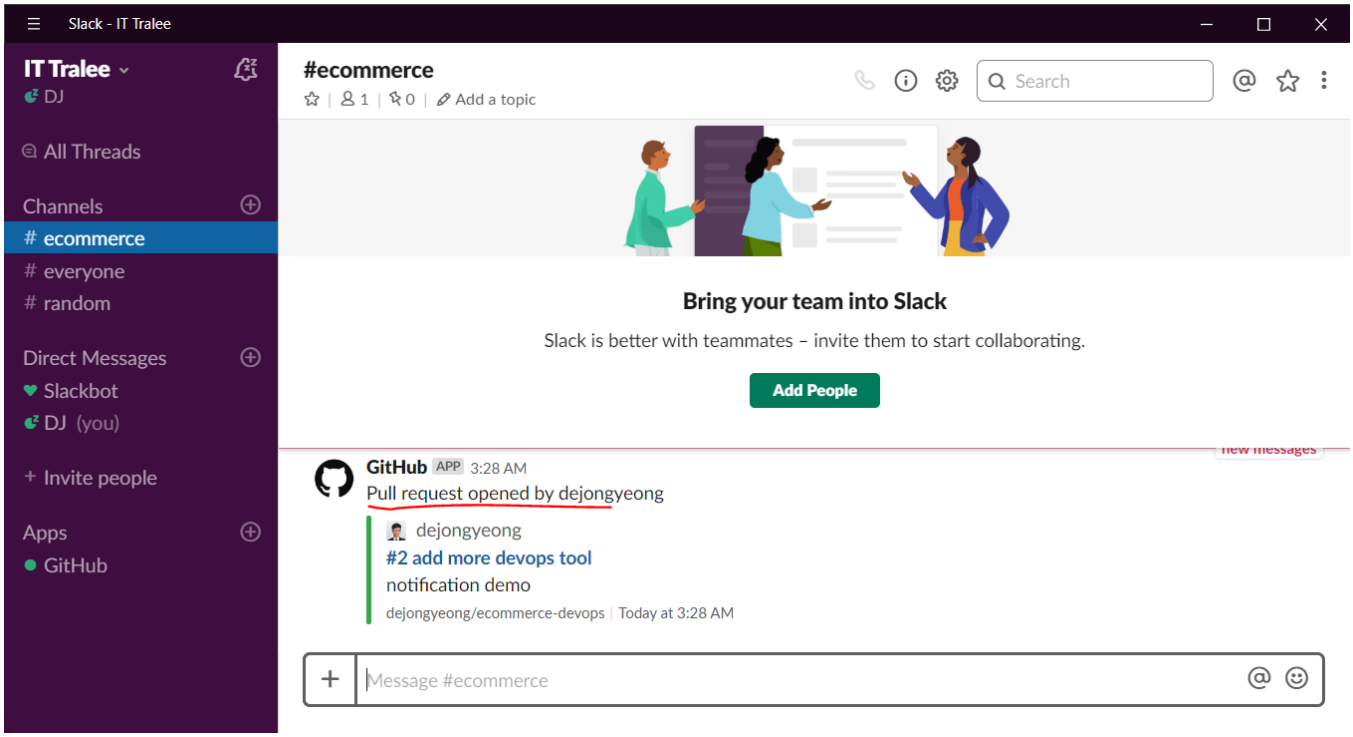
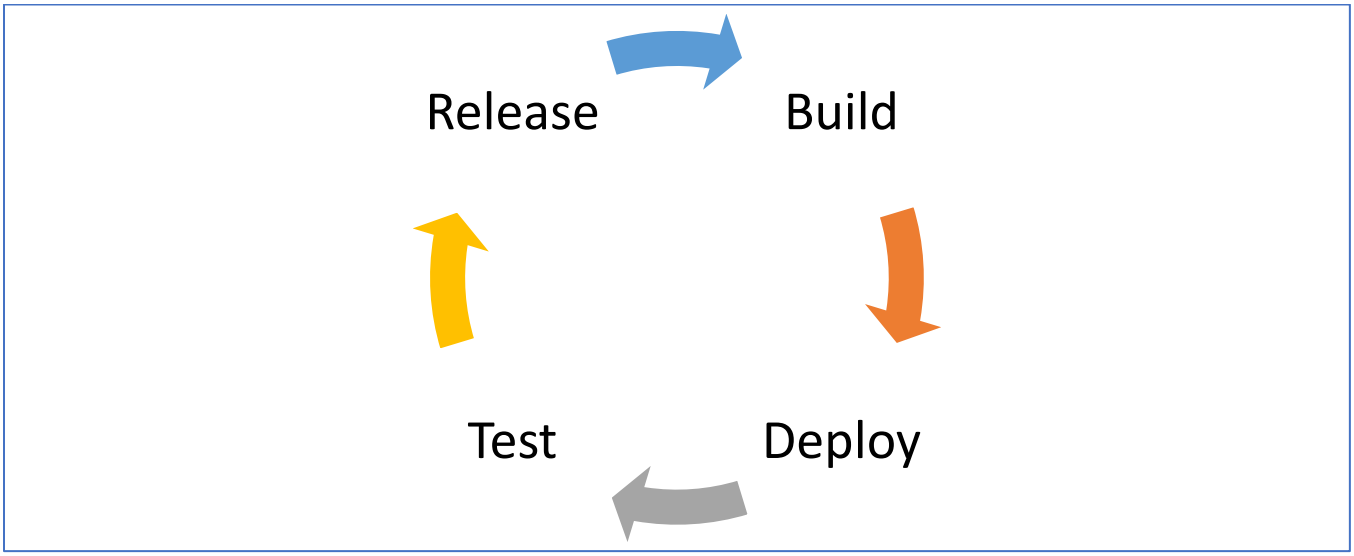


Figure 20: GitHub pull request notification in Slack.

Section 3: Continuous Integration

Continuous Integration (CI) is a software development practice which based on a frequent integration of the code in a shared repository and is verified by an automated build for each check-in or commit. Goal of CI is to identify problems that may occur during the development that helps to reduce time spent on debugging phase. CI offers an option to set up code style inspection and checks for cyclomatic complexity metrics et cetera. This helps to improve code quality and minimizes the effort required during code review process (Stars, 2017).



Travis CI was chosen for continuous service implementation. It is a hosted, distributed continuous integration service that is used to build, and test software projects hosted at GitHub. It offers the supports of Docker to run tests and supports more languages compare to Circle CI including PHP, JavaScript, Visual Basic, R et cetera. Travis CI configuration is saved in a file named .travis.yml, a YAML format text file stored in the root directory of the project. Travis CI supports of build matrix which is a tool that gives an opportunity to run tests with different versions of language and packages. For example, fails of some environments can trigger notifications but does not fails all the build (Stars, 2017).

Travis CI has pre-installed database services like MySQL ClearDB database and Heroku Postgres database and auto deploys application on passing builds. It supports pull request on GitHub and supports multiple operating system platform like Mac, Linux and iOS (travis-ci.org, 2019).

Branch build flow on Travis CI:

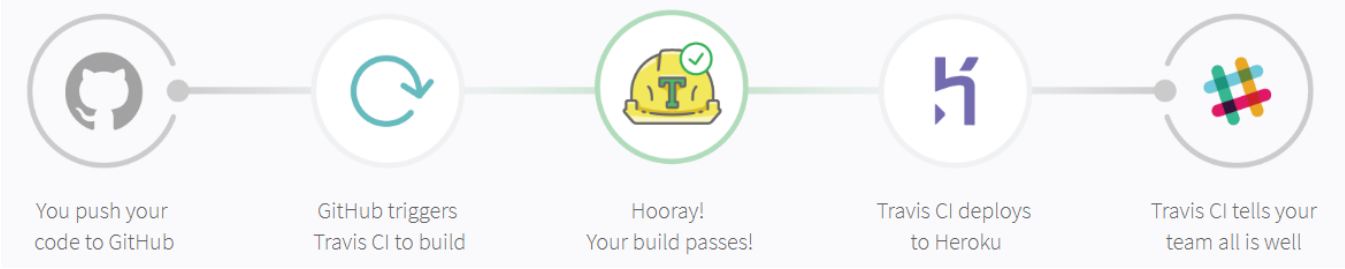


Figure 21: Branch build flow on Travis CI.

GitHub pull request build flow on Travis CI:

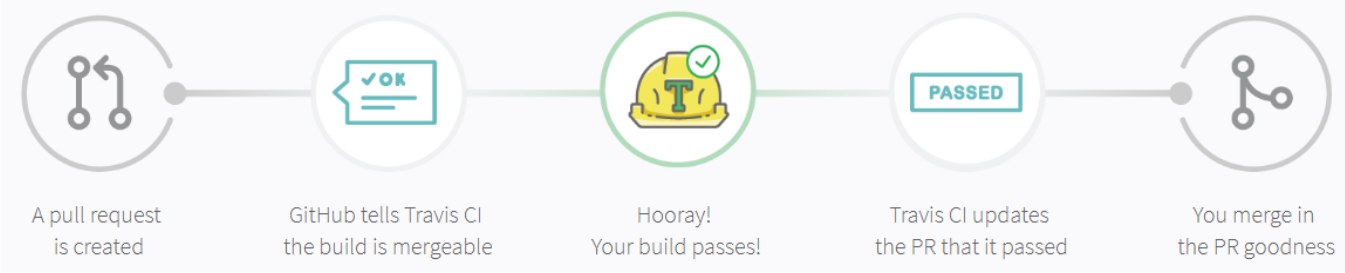


Figure 22: Pull request build flow on Travis CI.

3.1 Travis CI Installation Guidelines

Step 1: Visit https://travis-ci.org/ and click on the “Sign in with GitHub” button to sign in to user GitHub account.

Step 2: It redirects user to login page. User enters GitHub username and password and click on the sign-in button as shown below.

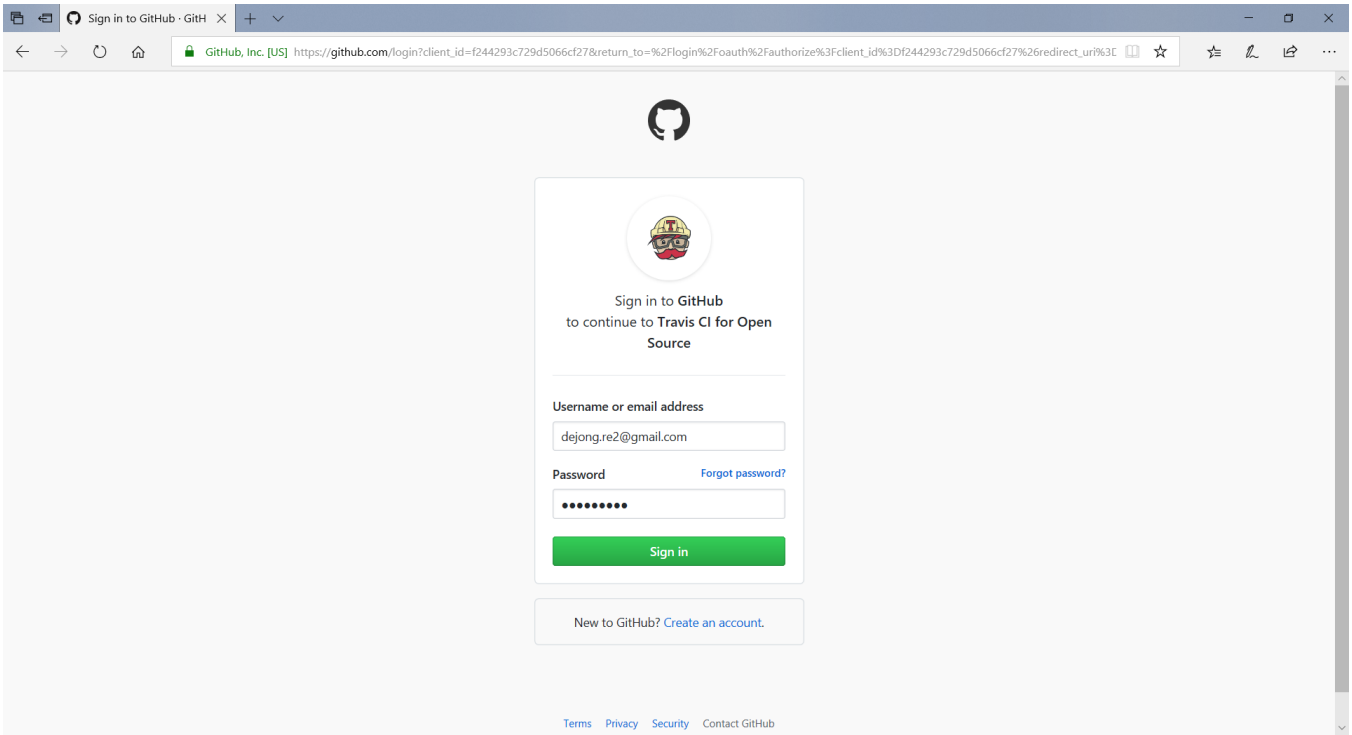


Figure 23: TravisCI Continuous Integration server dashboard.

Step 3: Follows the getting started guide outlined in https://travis-ci.org/getting_started after successfully logged in.

Step 4: Go to user profile and synchronize GitHub account to TravisCI and a list of public repositories will be displayed. Search and connect the selected public repository for Travis continuous integration process.

Figure 24: Connect select public repository for Travis CI process.

3.2 Automated Test Configuration in TravisCI

Guideline below outlines the steps required to configure automated unit test in TravisCI server. TravisCI will build and test the project automatically when a commit is pushed to GitHub.

Step 1: Create a .travis.yml file in project root directory.

Step 2: Copy the following code to file created in step 1 as show below. Tutorial link for configuration script is attached.

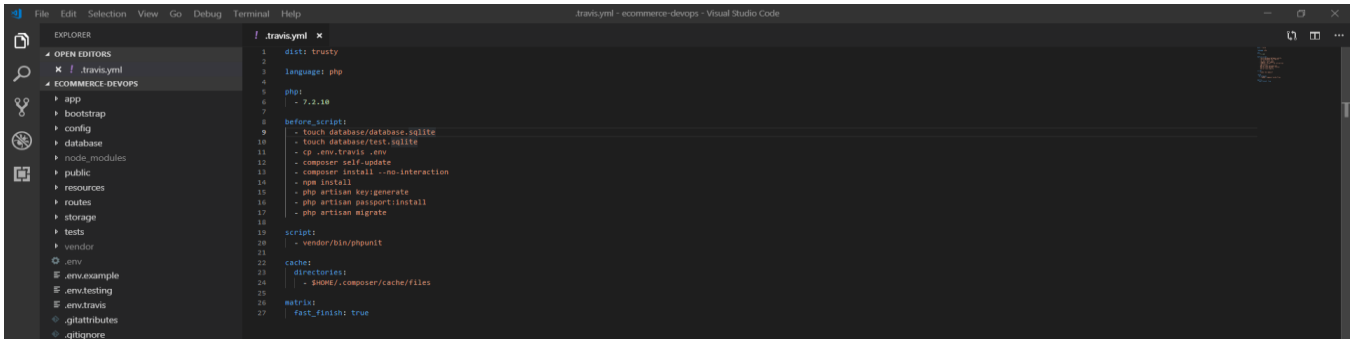


Figure 25: TravisCI .travis.yml configuration file.

Below shows the configuration code for TravisCI automated test and build process.

```
dist: trusty
language: php
php:
  - 7.2.10
before_script:
  - cp .env.travis .env
  - composer self-update
  - composer install --no-interaction
  - npm install
  - php artisan key:generate
  - php artisan passport:install
  - php artisan migrate
script:
  - vendor/bin/phpunit
cache:
  directories:
    - $HOME/.composer/cache/files
matrix:
  fast_finish: true
```

Step 3: Create a .env.travis environment file in project root folder and copy the following codes into the created file.

```
APP_ENV=testing
APP_KEY=yourappkey

DB_CONNECTION=sqlite
DB_TEST_USERNAME=root
DB_TEST_PASSWORD=

CACHE_DRIVER=array
SESSION_DRIVER=array
QUEUE_DRIVER=sync
```

Step 4: Create a new test.sqlite file for SQLite test database in database folder.

Step 5: Run command php artisan migrate:refresh --seed --env=testing --force to migrate tables and seed data for test database.

Step 6: Remove *.sqlite code from .gitignore file in database folder if code appears in the .gitignore file (empty file).

Step 7: Commit local changes and push to remote GitHub server to trigger automated test and build process in TravisCI. It is expected to have a successful build as shown below.

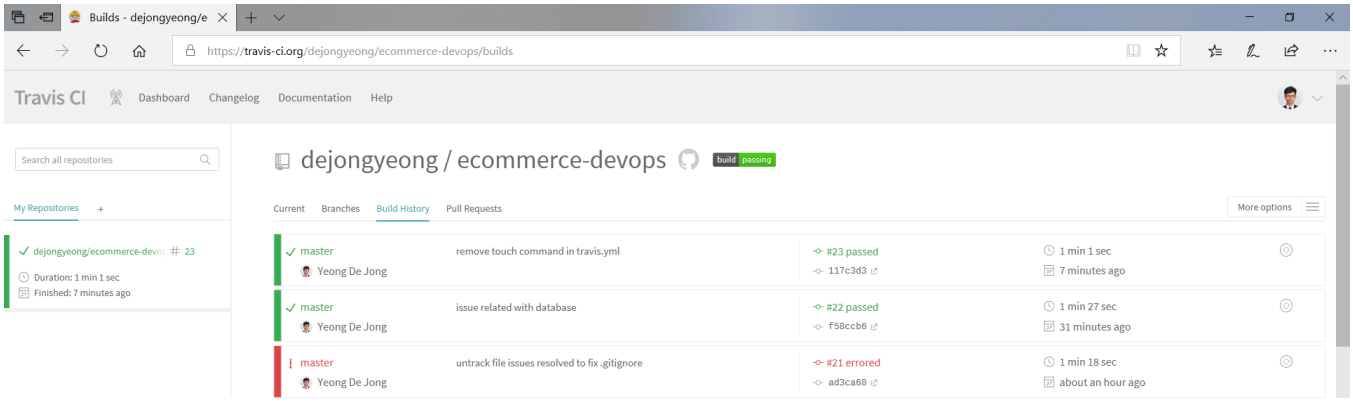


Figure 26: Successful build for automated test in TravisCI.

Please note that TravisCI builds will fail without execution of Step 6 due to database issue.

3.3 SonarCloud - Travis CI Integration

SonarCloud is a cloud-based continuous code quality which is totally free for all open-source projects. It supports major programming languages including Java, C#, JavaScript et cetera. SonarCloud creates a detailed report on bugs and vulnerabilities which helps to improve quality code in a project easily. SonarCloud offers free analyzing for public GitHub repository only. It provides end-to-end integrations for development teams in the following platform:

- GitHub.
- BitBucket Cloud.
- Azure DevOps (sonarcloud.io, 2019).

A full tutorial on SonarCloud-GitHub integration is available in table below:

Tutorial Link:
https://sonarcloud.io/documentation/integrations/github/

Step 1: Click on **GitHub** button to login to SonarCloud with GitHub in https://sonarcloud.io. It will redirect user to GitHub authorization page.

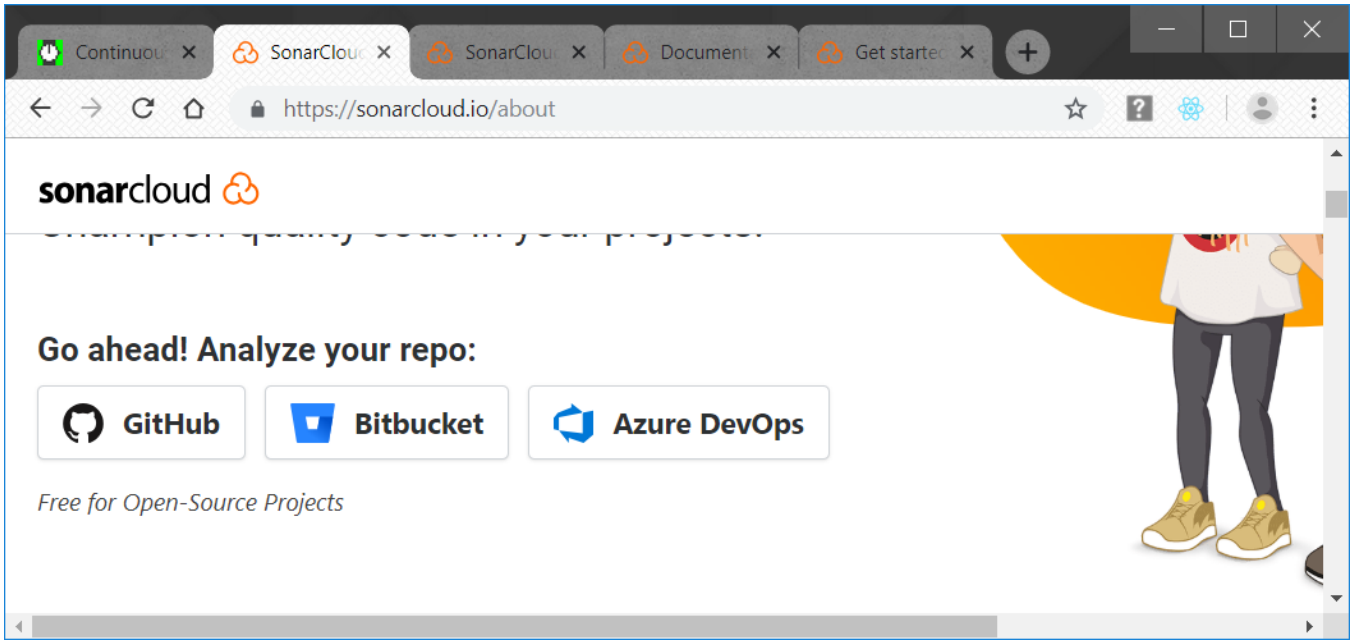


Figure 27: SonarCloud login with GitHub.

Step 2: Authorize SonarCloud in GitHub and user are redirect to SonarCloud dashboard after successful authorization.

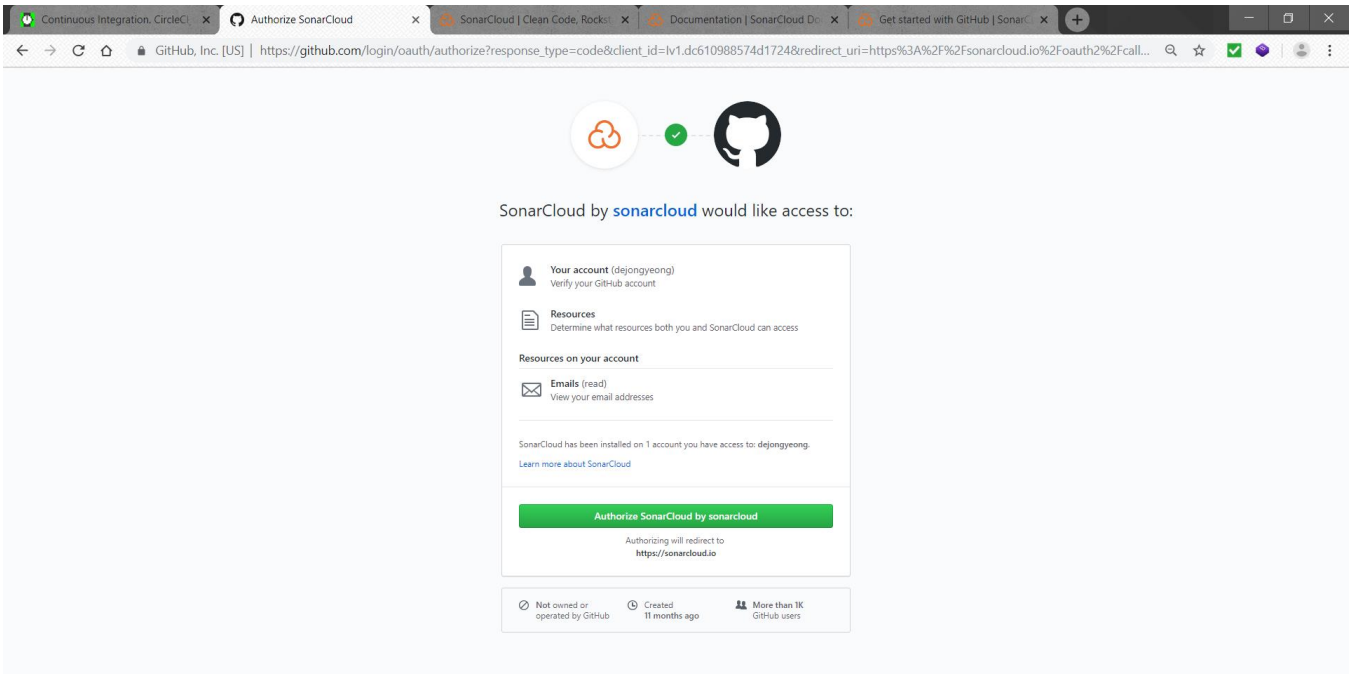


Figure 28: Authorize SonarCloud in GitHub.

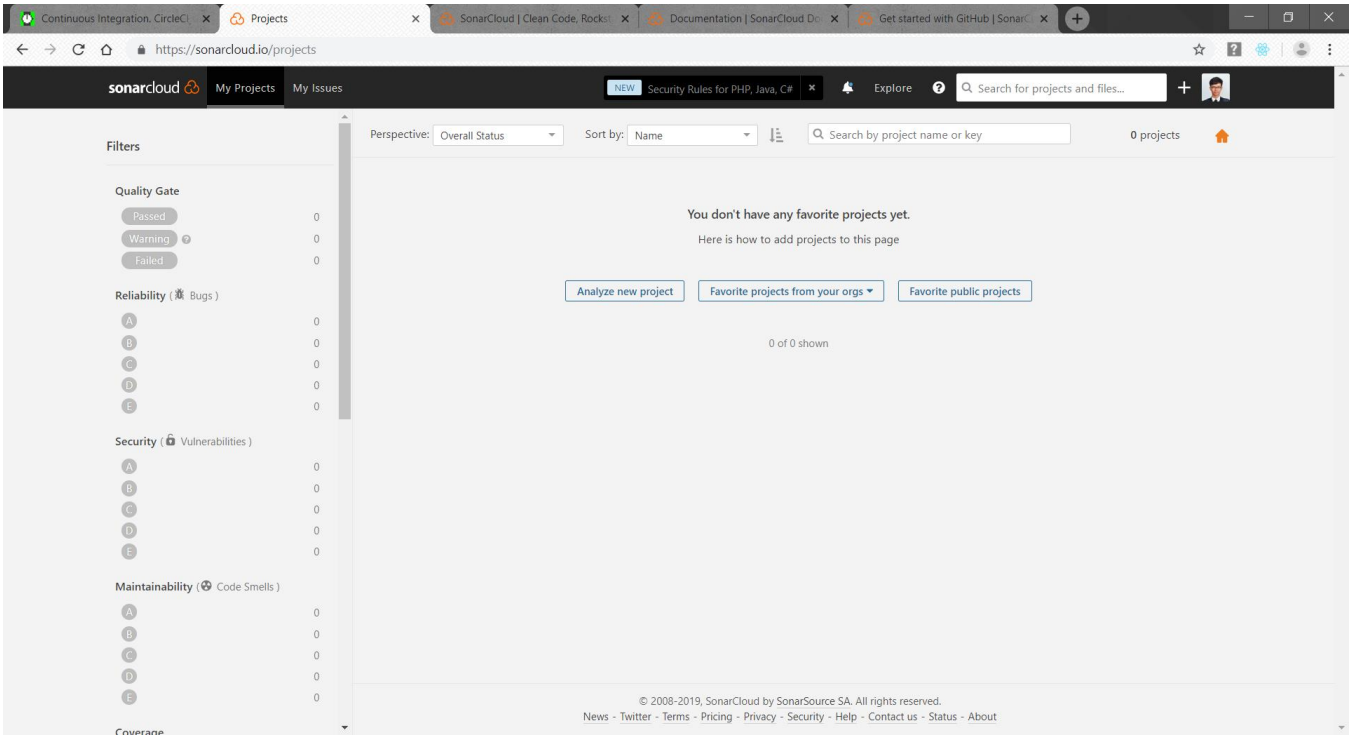


Figure 29: SonarCloud dashboard after successful authorized.

Step 3: Click on the analyze project button in SonarCloud dashboard shown above.

Step 4: Click on the import another organization link if SonarCloud contains a linked GitHub repository or clicks on create organization link to create a new organization.

Step 5: Click on **choose an organization on GitHub** on the Import from GitHub tab.

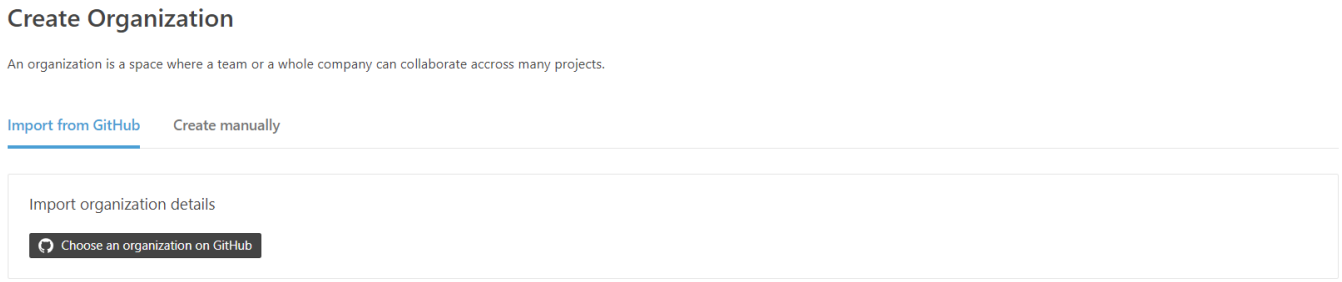


Figure 30: Create organization to analyze projects on GitHub public repository.

Step 6: Choose your repository if several options were provided. The author GitHub account is selected as shown below.

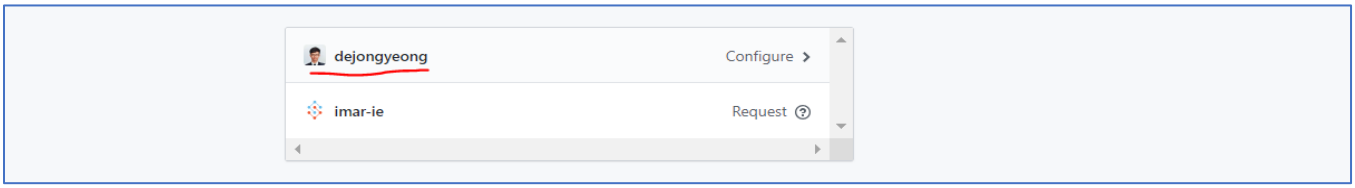


Figure 31: Choose an organization if multiple options provided.

Step 7: Click all repositories option to analyze code quality in all current and future repositories or select one repository to analyze code quality and click **save** button as shown below.

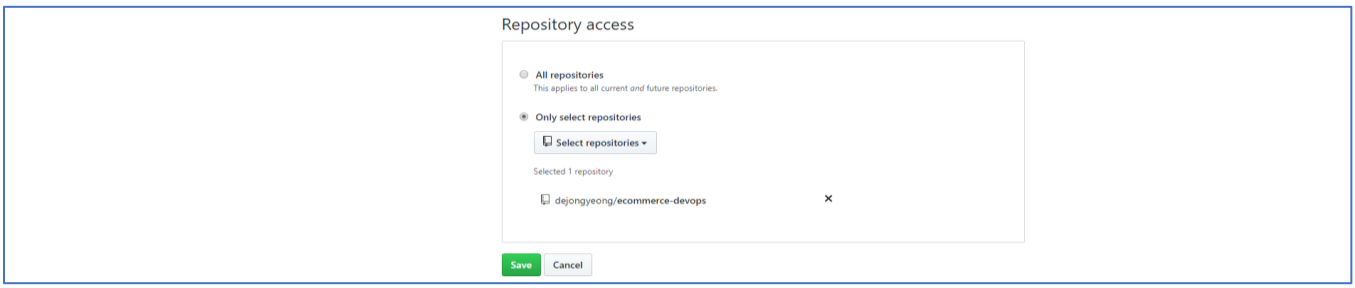


Figure 32: GitHub repository access configuration for SonarCloud.

Please note that SonarCloud offers free code quality analyzing for public repositories only and a paid plan from €10 for private repositories.

Step 8: GitHub redirects user back to SonarCloud to import and amend personal organization details including SonarCloud plan of either free or paid plan and click on create organization button.

Step 9: Provide a token name named "ecommere-devops" and click on generate button to generate token. The token is used as an identification when an analysis is performed.

Figure 33: Generate token and configure SonarCloud to run analysis.

Please note that SonarCloud offers two ways to trigger code analysis (choose either one):

AutoScan to trigger code analysis. SonarCloud autonomously pull the code and scan the default branch and pull requests. This is a beta version which only works with limitations. Detailed documentation on auto-scanning is available in SonarCloud documentation.

Step 1: Create an empty .sonarcloud.properties file in project root directory of default branch.

Step 2: Commit repository and push changes to remote server and results will be visible in SonarCloud project dashboard.

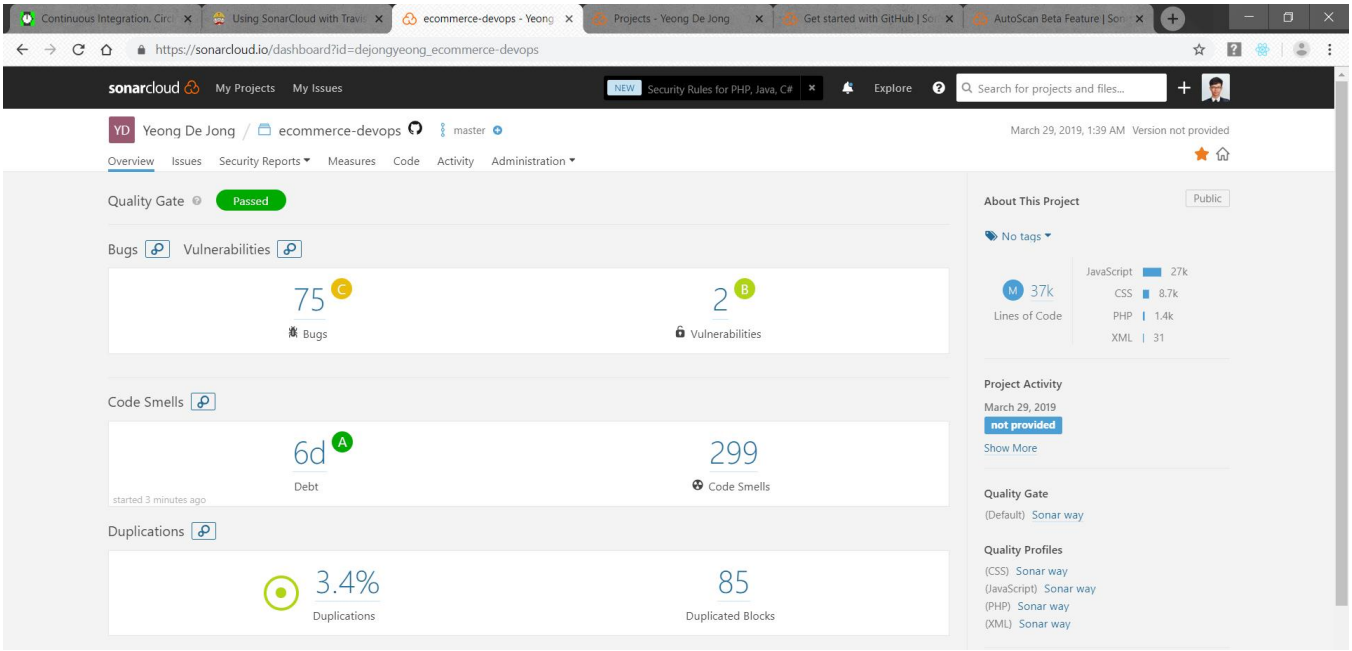


Figure 34: SonarCloud code quality report on project dashboard.

Continuous Integration service to trigger code analysis. It requires developer to write script that triggers the code quality analyzer. This needs to be configured if the auto-scan beta feature of SonarCloud does not work as expected. The chosen continuous integration service for this project was Travis CI. It makes it easier to activate the SonarCloud analysis with add-ons configured in Travis configuration file. Guide on Travis CI - SonarCloud integration is shown in table below.

Tutorial Link:
https://sonarcloud.io/documentation/integrations/github/
https://docs.travis-ci.com/user/sonarcloud/
https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner

Guideline below outlines several steps required to activate SonarCloud analyzer with TravisCI configuration file.

Step 1: Visit https://sonarcloud.io/projects and clicks on the selected project to retrieve project key and organization key shown on the bottom-right of the page.

Step 2: Visit https://sonarcloud.io/account/security to generate SonarCloud user token**.**

Step 3: Run travis encrypt <token> command to encrypt user token generated in Step 2.

Step 4: Create a sonar-project.properties file in project root directory and copy the following code into the file as shown below.

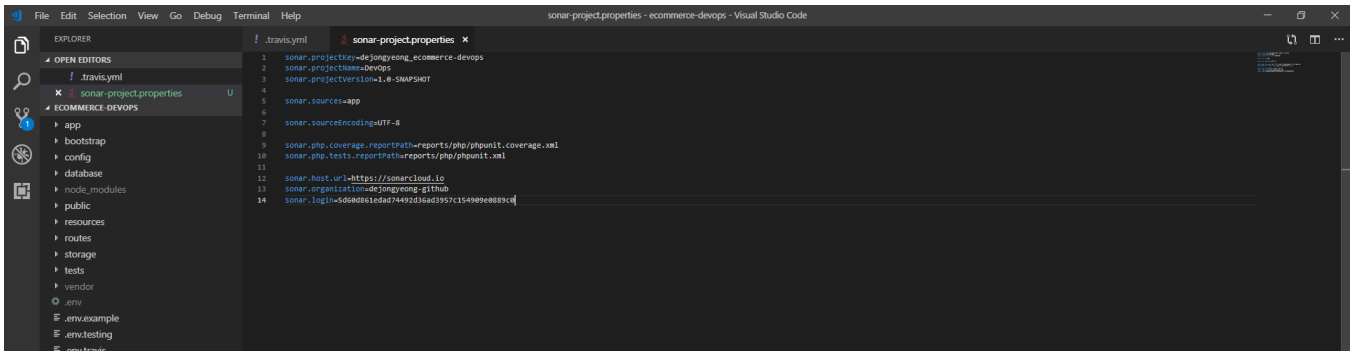


Figure 35: Configuration file for SonarCloud Travis CI integration.

```
sonar.projectKey=yourprojectkey
sonar.projectName=DevOps
sonar.projectVersion=1.0-SNAPSHOT

sonar.sources=app

sonar.sourceEncoding=UTF-8

sonar.php.coverage.reportPath=reports/php/phpunit.coverage.xml
sonar.php.tests.reportPath=reports/php/phpunit.xml

sonar.host.url=https://sonarcloud.io
sonar.organization=dejongyeong-github
sonar.login=yoursonarloginencrypt
```

Step 5: Append SonarCloud configuration to .travis.yml file with the following code shown in table below.

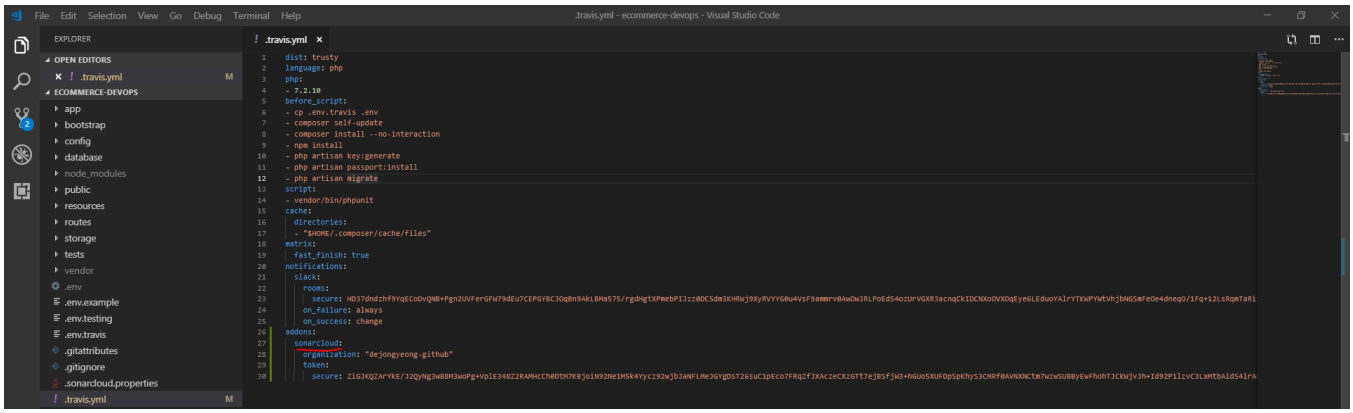


Figure 36: SonarCloud credentials in Travis CI configuration file.

```
addons:
  sonarcloud:
    organization: "dejongyeong-github"
    token:
      secure: yourencryptedtoken
```

Step 6: Commit local changes and push to remote GitHub server. It triggers SonarCloud analyzer to analyze code quality of ecommerce application.

3.4 Slack Notification - Travis CI Integration

TravisCI integrates with Slack for build results notification. TravisCI sends email notifications (default) to committer and the commit author. Notifications are sent when a build was broken or is still broken on a given branch or previously broken build was fixed. Guideline below outlines the step to configure Slack notifications for TravisCI build. Detailed documentation is detailed in table below:

Tutorial Link:
https://docs.travis-ci.com/user/notifications/
https://docs.travis-ci.com/user/notifications/#configuring-slack-notifications

It is recommended to encrypt Slack account and token with Travis command line interface. Detailed tutorial on how to encrypt credentials is outlined in TravisCI encryption keys documentation. Prerequisite is to install "gem" command from https://rubyinstaller.org/.

Assumed that the Travis command line interface is installed.

Step 1: Browse Travis CI integration application in Slack and click Install to install Travis CI.

Step 2: It redirects user to Slack website and click on the Install button shown on website.

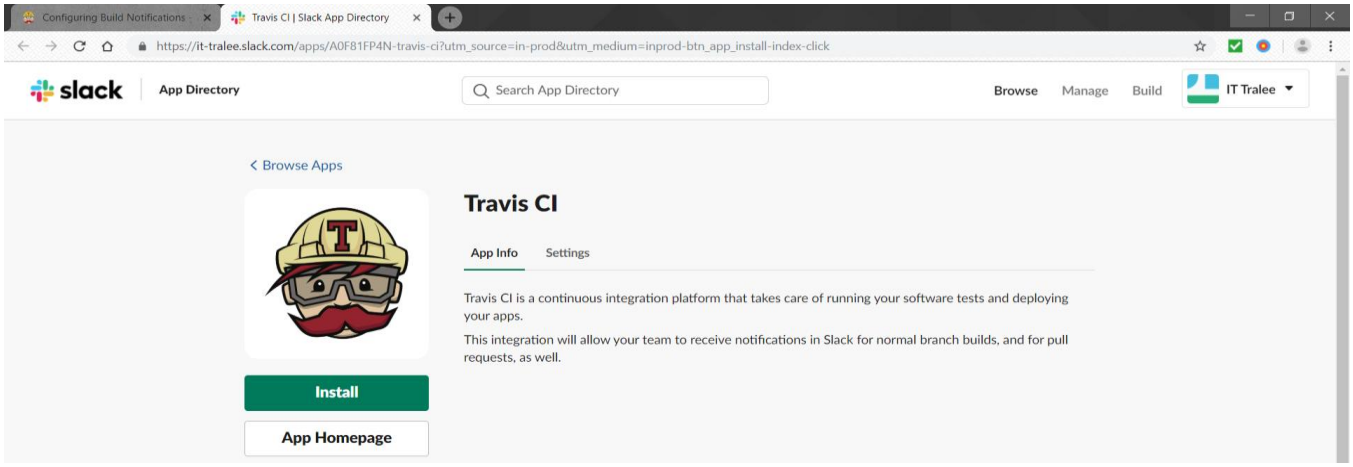


Figure 37: Slack installation of Travis CI integration page.

Step 3: Selects or creates a new channel where Travis build notification will be posted and click on the **Add Travis CI Integration** as shown in screenshot below.

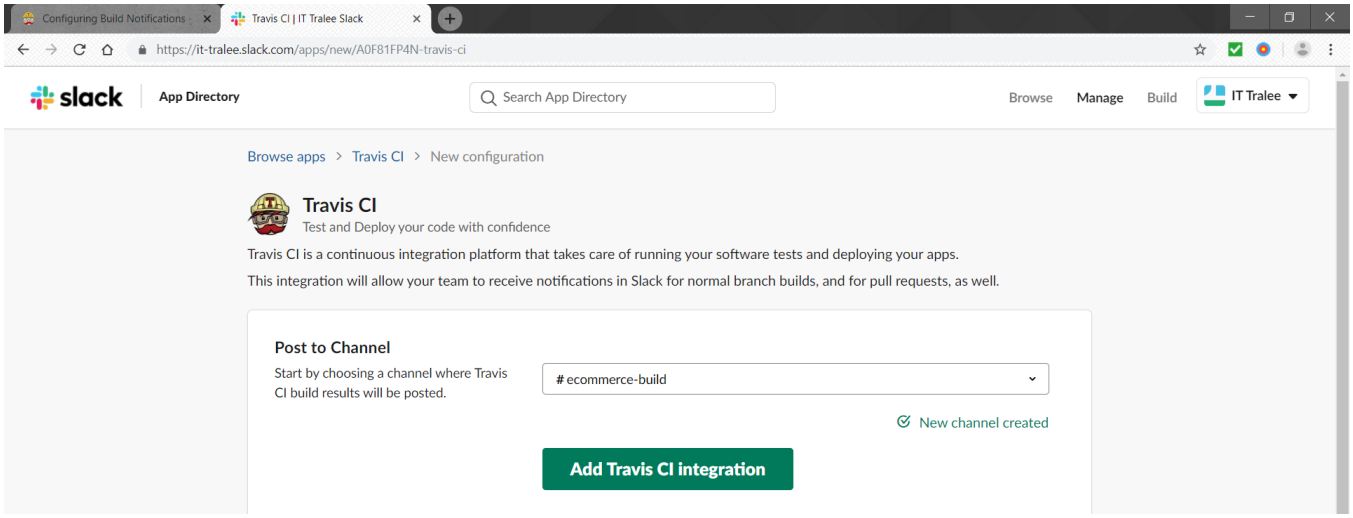


Figure 38: Slack install Travis CI integration.

Step 4: Follows the setup instructions shown in Slack website.

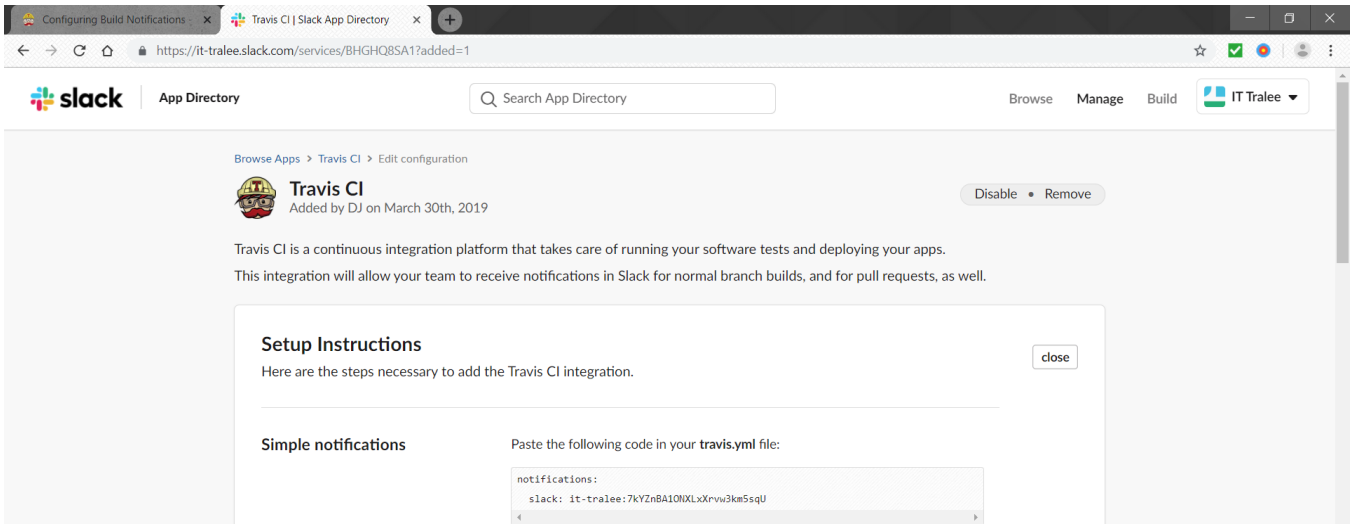


Figure 39: Travis CI integration setup instructions for Slack.

Please note that it is recommended to encrypt credentials when setting up Travis CI build notifications. Run travis encrypt SOMEVAR=”secret value” command in Git local repository to encrypt data and add the output string into .travis.yml file as shown below.

The best practice for notification integration is to send only changes or failure builds to Slack.

Figure 40: Insert Slack configuration with secure value in .travis.yml file.

Step 5: Commit local changes and push to remote GitHub server. A notification on the build result will only display in Slack #ecommerce-build channel when Travis build fails.

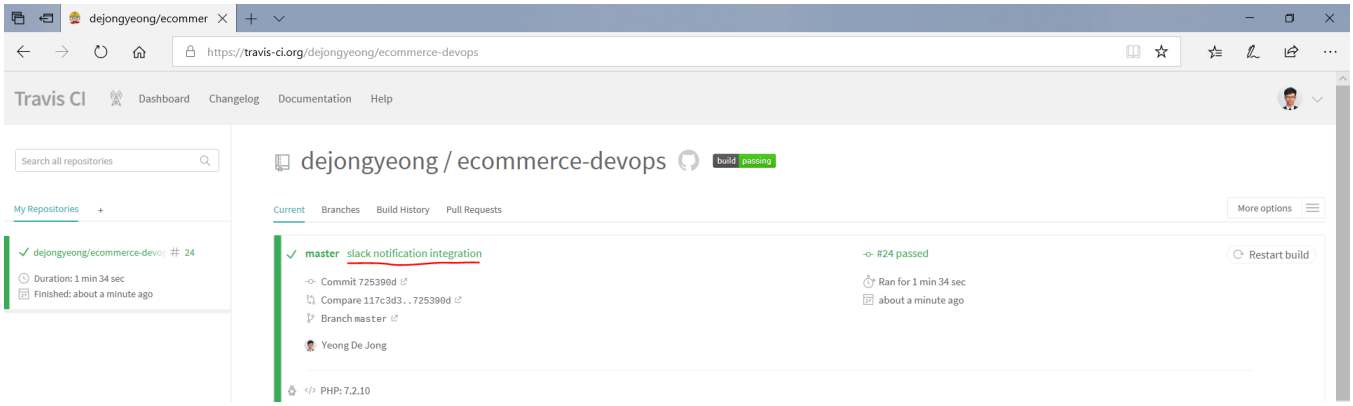


Figure 41: Slack - Travis CI integration build successful.

Section 4: Building Code and Configure Build Pipeline

A continuous integration/continuous delivery (CI/CD) pipeline is the backbone of modern development operations environment. It automates the build phase, test phase and deployment phase of applications which bridges the gap between development and operations teams (Tuli, 2018). CI is the base on which CD setup is built and it is not possible to start CD before configures a CI environment.

CD is defined as the processes and practices through which applications are made available to be deployed to production at any given time. Builds can be deployed to a user acceptance testing (UAT) environment which allows stakeholders to test out the application and makes the decision for application deployment. CD ensures code is available for deployment to production in a reliable manner and aims to speed up the release process. It enables teams to search and fix bugs earlier in the development cycle and encourages strong collaboration between developers (smartbear.com, 2019).

4.1 Heroku Installation and Configuration

Heroku was chosen for build pipeline configuration. It is a platform as a service (PaaS) that enables developers to build, run and operation the entire applications in the cloud. Flexibility of Heroku allows developer to build applications using language or framework that developer familiar with. Heroku is an open-source cloud platform for public repositories and costs a total of $7 per month for hobby plan and a range of $25 - $500 for professional plan (heroku.com, 2019). Tutorial outlines the steps to create build pipelines in Heroku.

Tutorial Link:
https://devcenter.heroku.com/articles/getting-started-with-php
https://devcenter.heroku.com/articles/pipelines

Step 1: Visit https://www.heroku.com/ and click on “Sign Up” button to create a Heroku account with user information or the “Login” button with an existing Heroku account. It redirects user to Heroku dashboard as shown below.

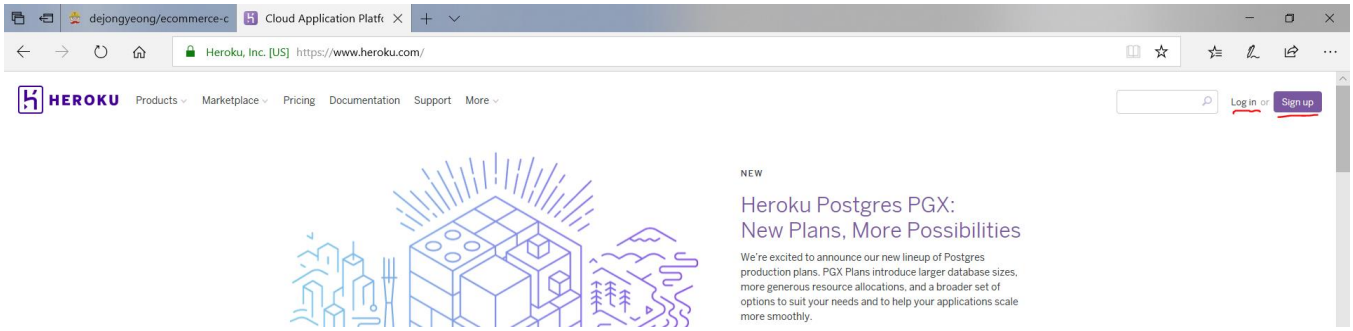


Figure 42: Heroku landing page to sign-up or login.

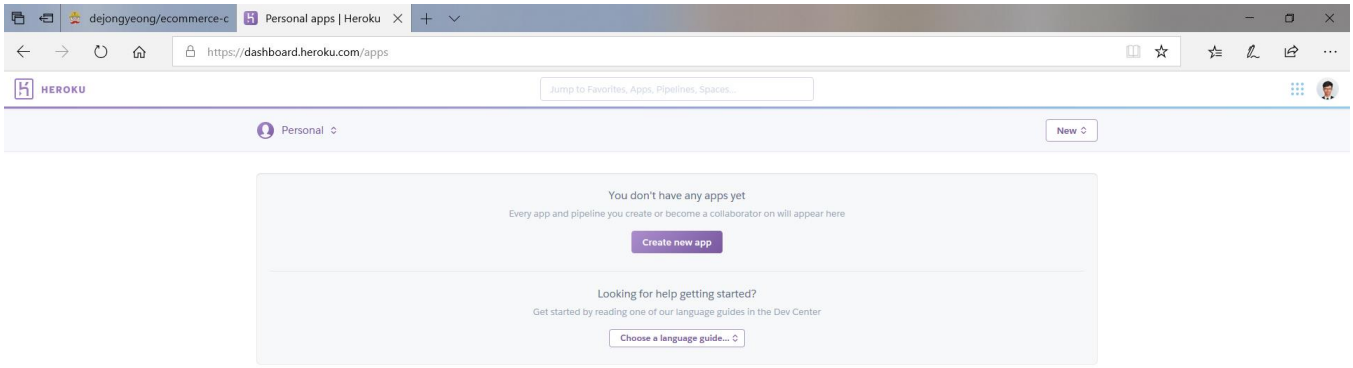


Figure 43: Heroku user dashboard.

Step 2: Download and run Windows installer to install Heroku command line interface shown in the tutorial link above. Heroku CLI is used to manage and scale applications, provision add-ons, view application logs and run application locally.

Please note that user can run the heroku login command to log in to the Heroku CLI after Heroku CLI is installed globally.

Step 3: Click the **new** button in the top right of the app list and select **create new app** from Heroku dashboard.

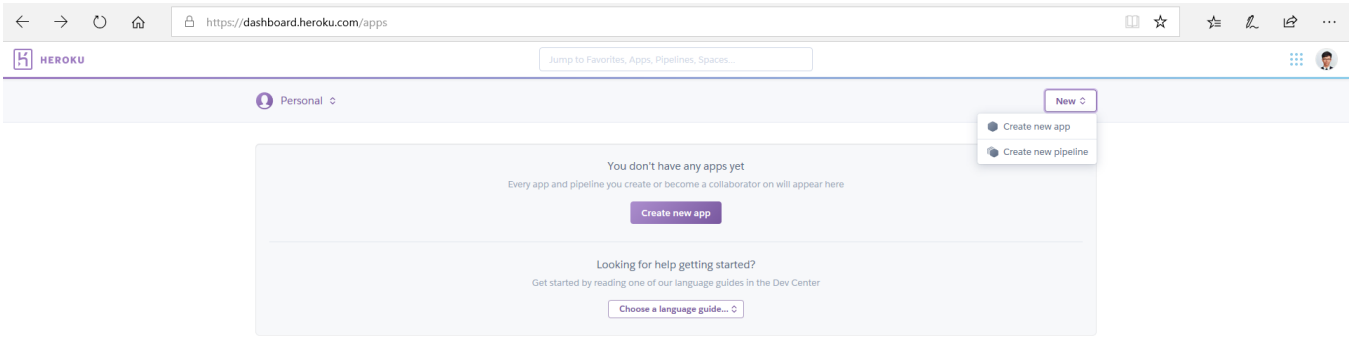


Figure 44: Heroku create new app from dashboard.

Step 4: Enter the application name with Europe region and click on the Create app button.

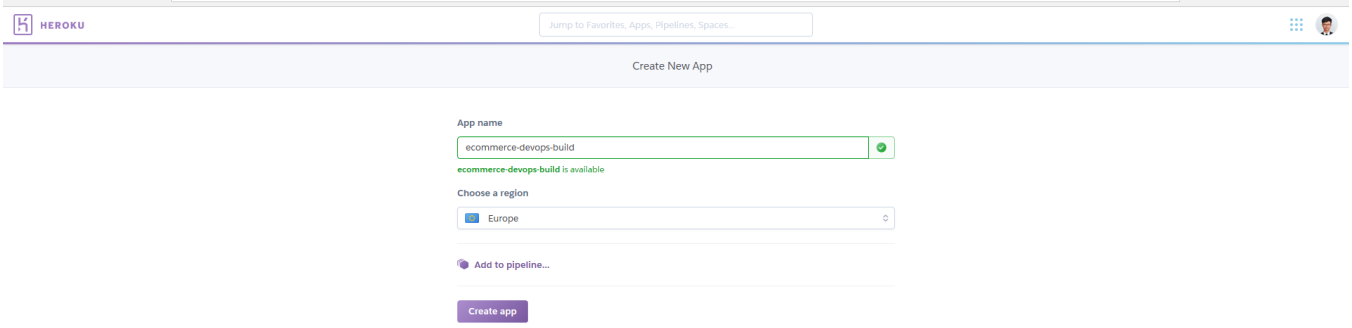


Figure 45: Application Name and Region configuration in Heroku.

Step 5: Visit Deploy tab from Heroku dashboard and click on the **create new pipeline** option under **choose a pipeline** dropdown list in add this app to pipeline section to include that app.

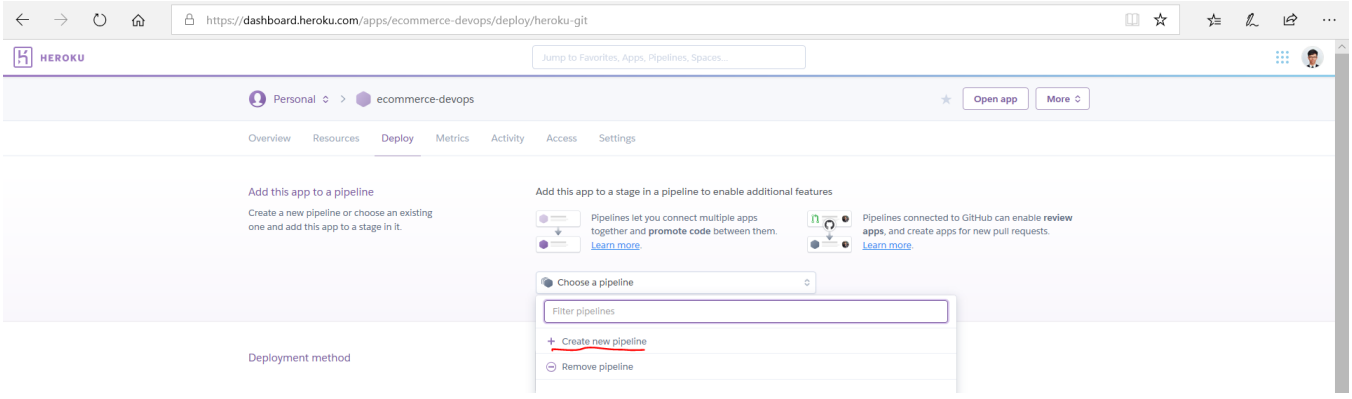


Figure 46: Create a new pipeline from Heroku dashboard.

Step 6: Enter a name for the pipeline with the staging option and clicks on the create pipeline button shown below. It will redirect user to Heroku dashboard and prompts user to connect the created pipeline to GitHub to enable additional features.

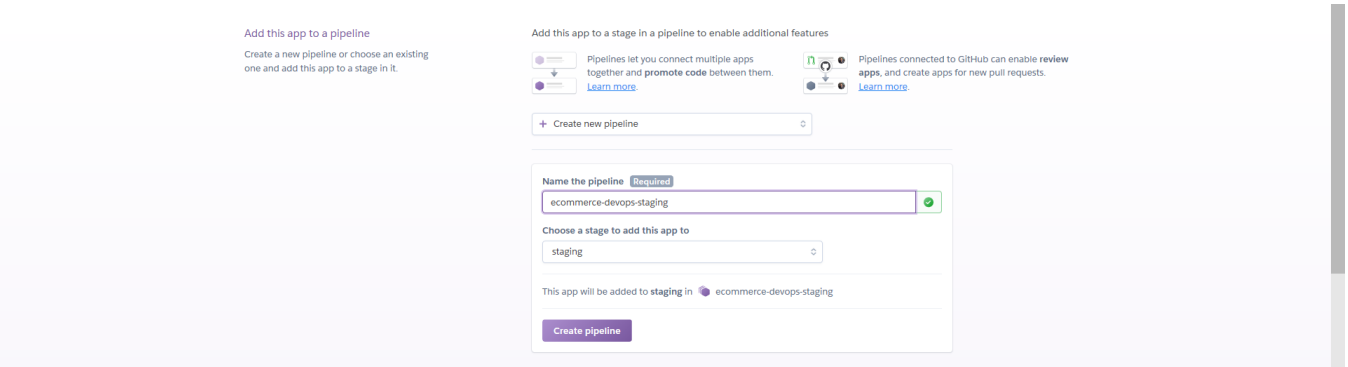


Figure 47: Configure pipeline with a name and the staging option.

Step 7: Click on **connect to GitHub** button to connect created pipeline to GitHub as shown. It redirects user to build pipeline setting page and requires GitHub authorization before connect build pipeline to GitHub.

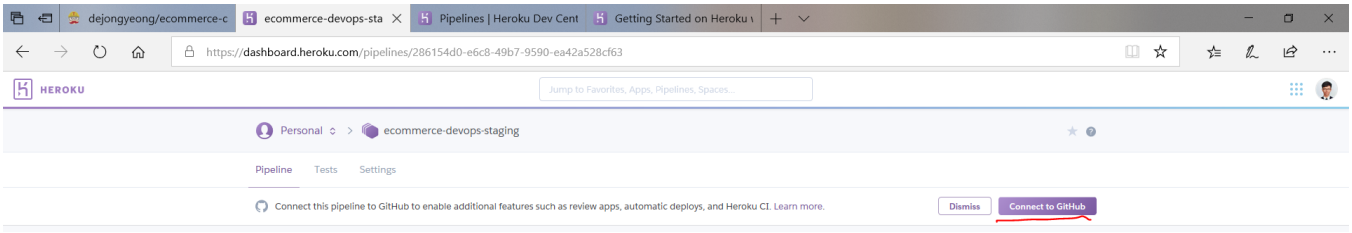


Figure 48: Connect build pipeline to GitHub to enable additional features.

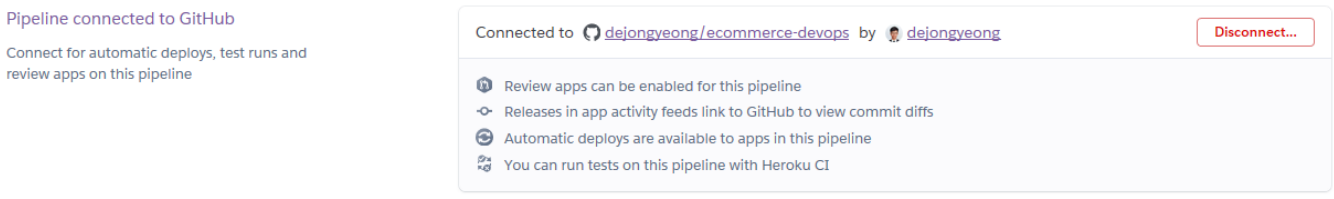


Figure 49: Build pipeline connected to GitHub.

The build pipeline is shown in Heroku dashboard after the pipeline created in step 6 is successfully connected to GitHub as shown below. The ecommerce application is now in the staging phase of the build pipeline.

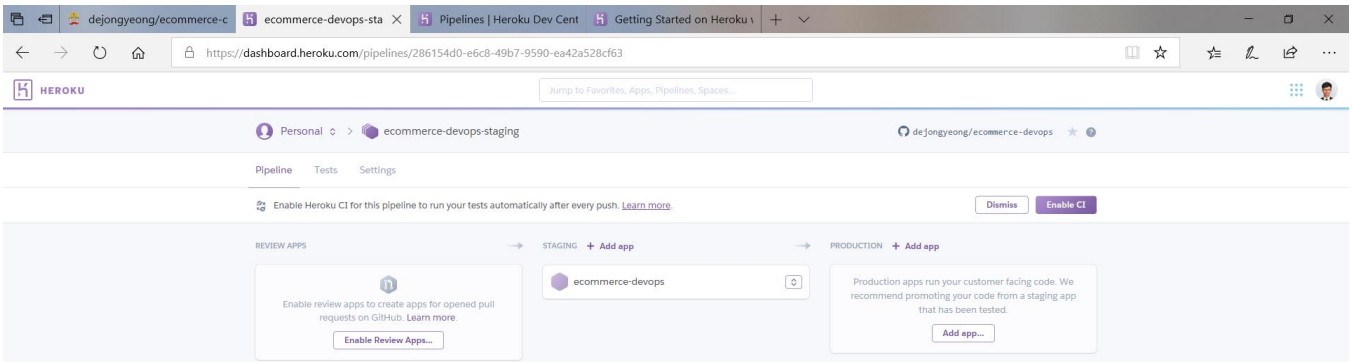


Figure 50: Heroku build pipeline.

Step 8: Click on the **ecommerce-devops pipeline** and enables automatic deploys application feature from GitHub in the Deploy tab. Check the “wait for CI to pass before deploy” option and click on the enable automatic deploys button as shown below.

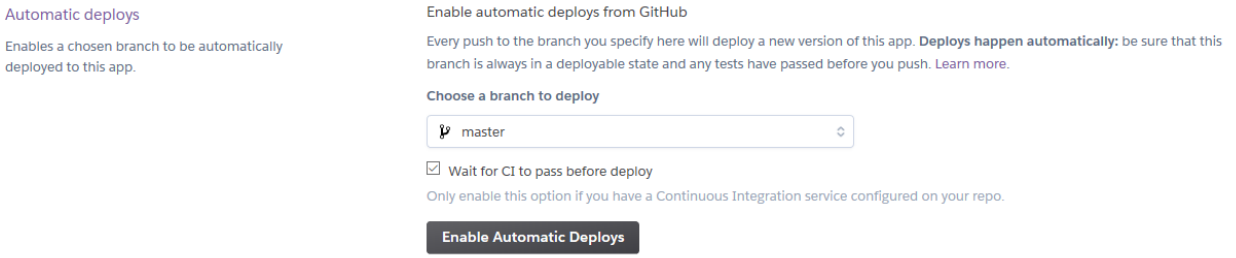


Figure 51: Enable automatic deploys feature to automatic deploy application.

Step 9: Create a **Procfile** file in project root directory and copy the following line in to that file.

```
web: vendor/bin/heroku-php-apache2 public/
```

4.2 Heroku Automated Build (Staging Stage)

Please note that SQLite won't work with Heroku and requires to use a SQL database of either Postgres or MySQL (ClearDB). This is because the "php artisan migrate" command runs on a one-off dyno and the application starts on the other isolated dyno where the database does not exist (Zulke, 2017). Tutorial on Heroku Postgres installation is shown in Appendix E.

It is recommended that review apps should inherit their configuration from a development or staging version of the application only. This is because inheriting configuration from production could inadvertently leak production data to a testing environment. Guideline below outlines the steps required to create a Heroku remote git repository and a Laravel buildpack and Travis CI configuration for automated deployment after successful build.

Step 1: Run heroku git:remote --app [appname] command to add to Heroku git remote.

```
PS D:\IT Tralee\ecommerce-devops> heroku git:remote --app ecommerce-devops
set git remote heroku to https://git.heroku.com/ecommerce-devops.git
PS D:\IT Tralee\ecommerce-devops>
```

Figure 52: Add Heroku git remote.

Step 2: Visit Heroku application setting page and add php buildpack configuration which will be use when the application is deployed as shown below.

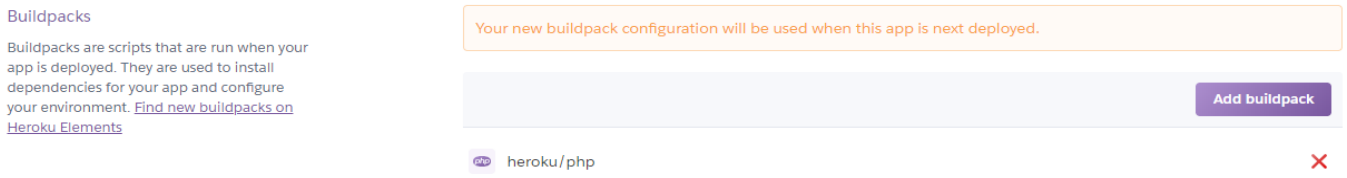


Figure 53: Configure PHP buildpack for application deployment process.

Step 3: Visit https://dashboard.heroku.com/account and look for Heroku API key (at bottom of the page) as shown below.

Figure 54: Heroku API key for Travis CI configuration.

Step 4: Run travis encrypt "<token>" command to encrypt Heroku API key.

Step 5: Append Travis CI automated deployment configuration in .travis.yml file with the following codes as shown below:

```
deploy:
  provider: heroku
  api_key:
    secure: heroku_encypted_key
  app:
    master: ecommerce-devops
  on:
    repo: dejongyeong/ecommerce-devops
```

Step 6: Run the follow commands to configure Heroku variables as shown in table below. Configuration variables changed the behavior of the application.

```
heroku config:set APP_DEBUG=true
heroku config:set APP_ENV=development
heroku config:set APP_KEY=app_key_from_env_file
heroku config:set APP_LOG=errorlog
```

Step 7: Commit local changes and push to remote GitHub repository which will then triggers the automated deployment process after a successful build.

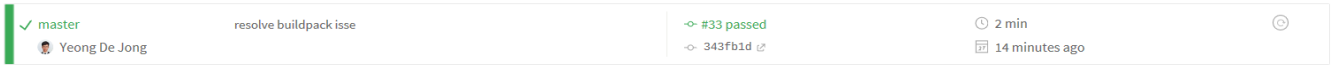


Figure 55: Build successful and will triggers automated deployment.

Visit https://ecommerce-devops.herokuapp.com/ to check for the deployed ecommerce application. Mixed content error occurs due to HTTP content is served over HTTPS. Guidelines outlined below shows additional procedure that needs to be executed to resolve the mixed content error.

Step 1: Copy the following code and paste it in routes/api.php and routes/web.php file.

```
if (\App::environment('development') ||
    \App::environment('production')) {
    \URL::forceScheme('https');
    \URL::forceRootUrl('https://ecommerce-devops.herokuapp.com/');
}
```

Step 2: Remove trailing slash in each resource view files on the RESTful route as shown below. All files required to be modified is in resources/assets/js folder.

Folder	Filename	Line Number
components/admin	Main.vue	26, 34, 42
	Orders.vue	37
	Products.vue	43, 79
	Users.vue	33
views	Checkout.vue	74
	Home.vue	33

Step 3: Amend the secure variable (line 167) to true and http_only variable (line 180) to false.

Step 4: Commit local changes and push to remote GitHub server. It will triggers automated deploy application process after a successful build.

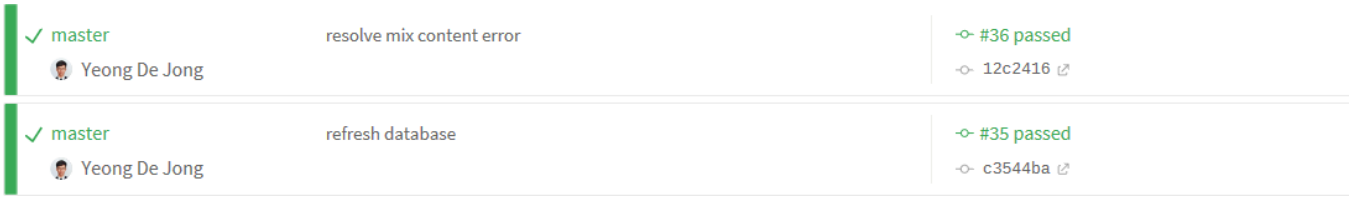


Figure 56: Build successful triggers automated deployment.

4.3 Review Apps Stage

Review Applications create application for opened pull request on GitHub which requires user to create an app.json file to set config vars, create add-ons and run post-deploy scripts when new review apps are created. Guideline below outlines the steps required to enable review applications.

Step 1: Visit Heroku pipeline dashboard and click on “Enable Review Apps” button to enable review apps feature for opened pull requests on GitHub.

Step 2: Click on **create an app.json file** on the pop-up dialog. An app.json file needs to be created to enable review apps.

Step 3: Heroku detects the default configuration of the application to create the app.json file. Click on the **Commit to Repo** button at the bottom of the page which will then commit a file named app.json to the default branch of ecommer-devops in GitHub.

Step 4: Enable the **create new review apps for new pull requests automatically** option to create an application for every new pull request.

Step 5: Enable the **destroy stale review apps after 5 days without any deploys** option to destroy stale review application after 5 days without any deploys and click Enable.

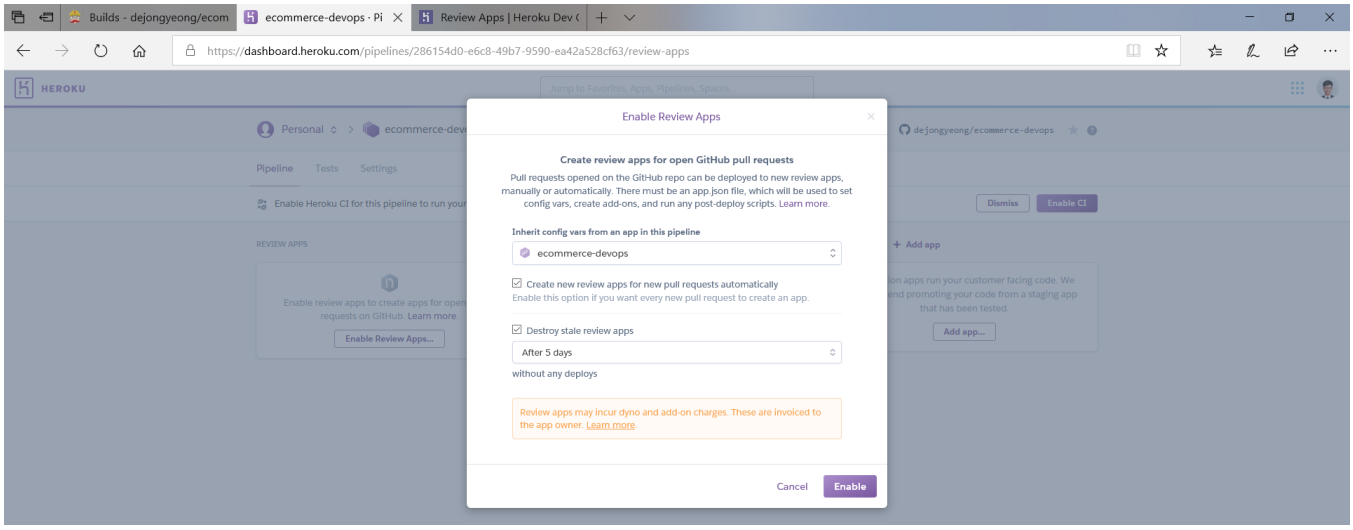


Figure 57: Enable review apps configuration.

Let’s create a pull request on the ecommerce application to test for the review apps feature in Heroku.

Step 1: Append information in readme.md file.

Step 2: Run git checkout -b <branchname> command to create and change to that branch.

Step 3: Add and commit local changes.

Step 4: Run git push --set-upstream origin <branchname> command to push the current branch and set the remote as upstream.

Step 5: Login to GitHub account and click on the **pull requests** section in ecommerce-devops repository.

Step 6: Click on the **New pull requests** button to open a new pull request as shown below.

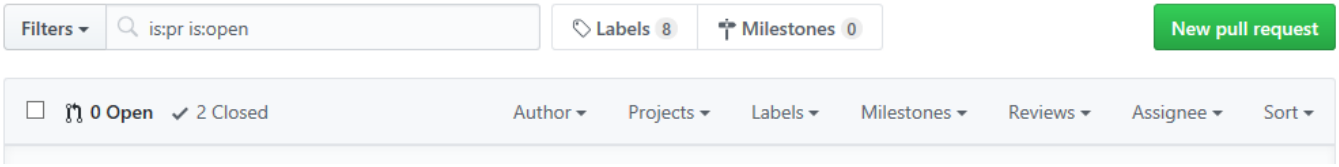


Figure 58: GitHub new pull requests.

Step 7: Choose two branches to see what's changed or to start a new pull request. Select master branch as the base branch and the review-apps-test branch as the compare branch and click on the "create pull request" button to create and open a new pull request as shown below.

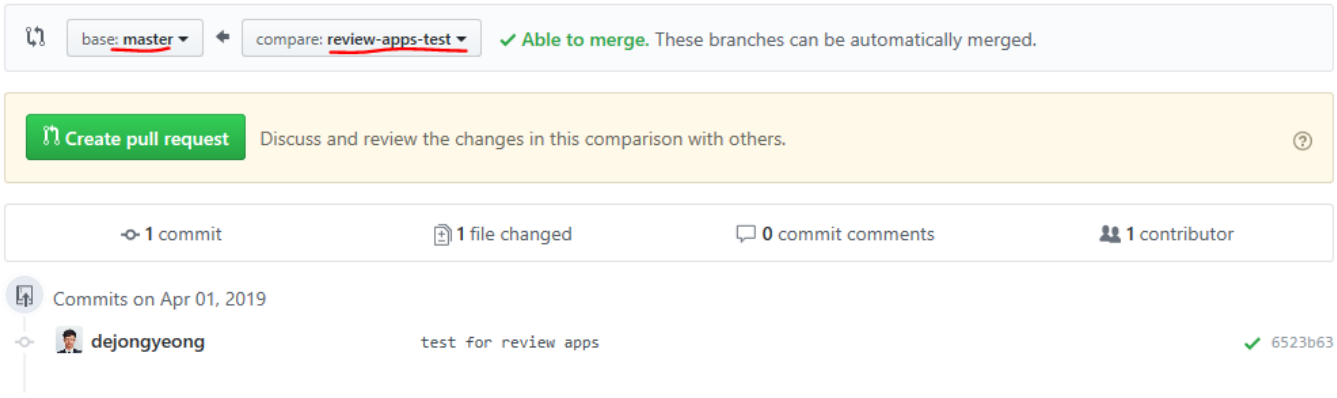


Figure 59: Create and open a new pull request.

The review apps stage in Heroku build pipeline created a review apps as shown below after a new pull request is opened.

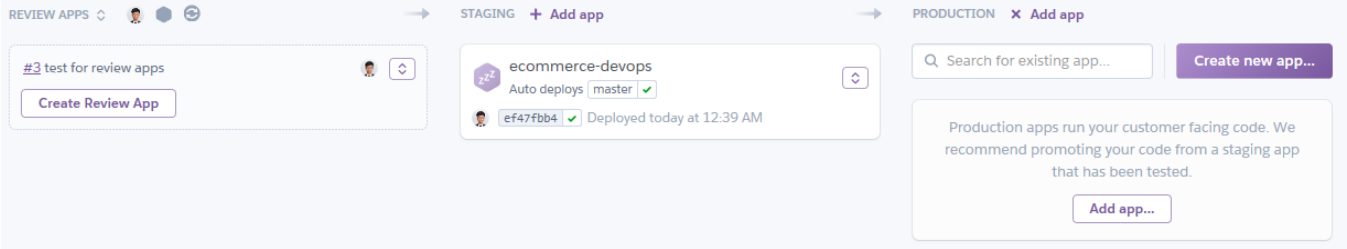


Figure 60: The created new pull request is now in review apps stage.

Step 8: Merge pull request to merge commit of review-apps-test branch into the master branch. All checks should pass and has no conflict with the base branch before merging the pull request into master branch.

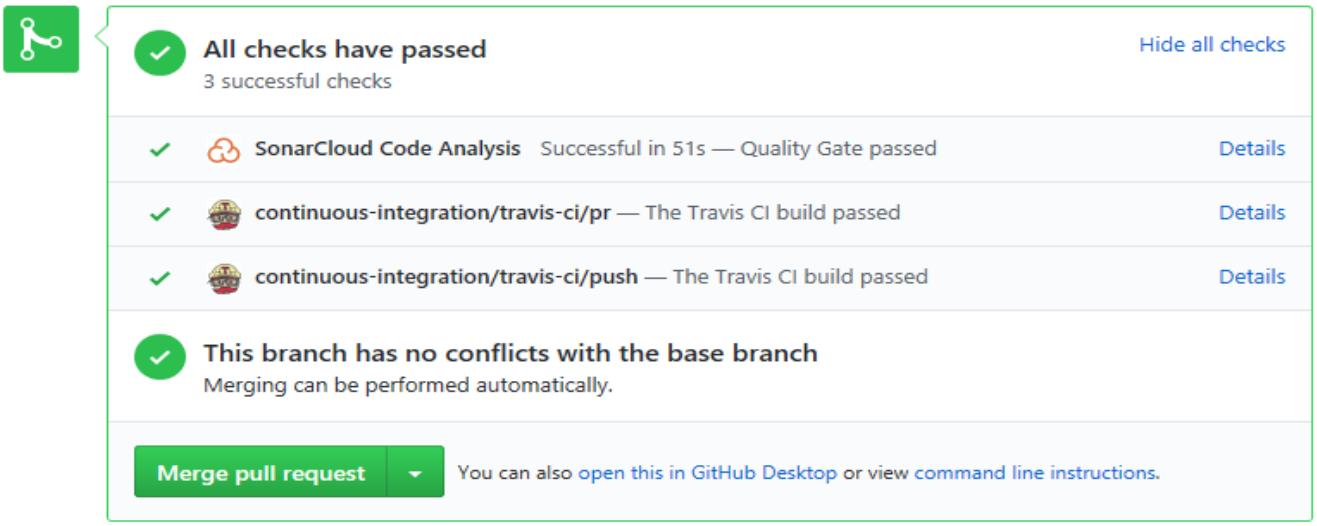


Figure 61: Merge pull request of the created branch to master branch.

The pull request shown in the Heroku build pipeline is now removed after merging the pull request into master branch as shown below.

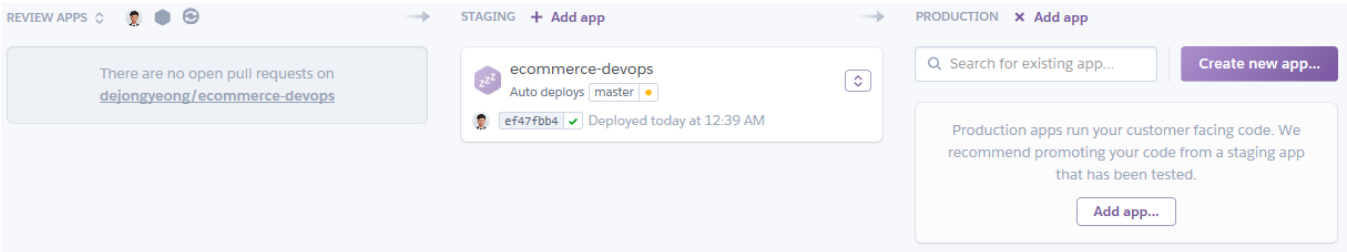


Figure 62: No open pull request on review apps stage after a pull request is merged.

4.4 Production Stage

Heroku documentation outlines detailed step on creating pipelines from staging stage to production stage. Heroku build pipelines can be configured from the Heroku dashboard or from the Heroku command line interface.

Tutorial Link:
https://devcenter.heroku.com/articles/pipelines#creating-pipelines

Pipeline let developers to defined how deployed code flows from one environment to the next. This ensures production stage contains the exact same code that are being tested in the staging stage and is much faster than rebuilding the slug. Staging is downstream of development and production is downstream of staging. It is recommended to click on the “Promote to production” button to promote the ecommerce-devops application in staging stage to production stage.

Step 1: Click **add app** button on the production with a name of ecommerce-devops-prod and click on the **promote to** button to promote application into production stage.

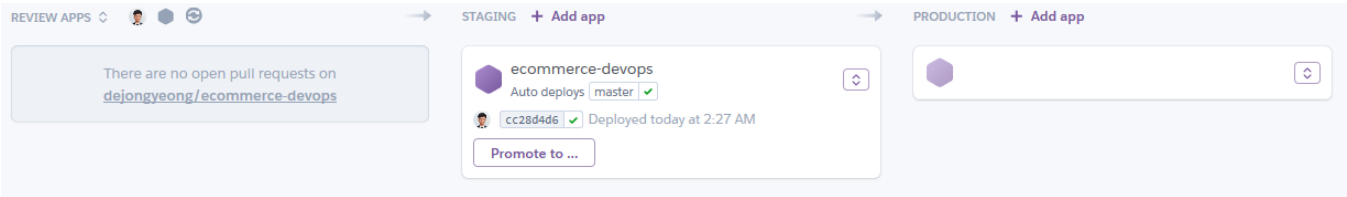


Figure 63: Promote from staging to production.

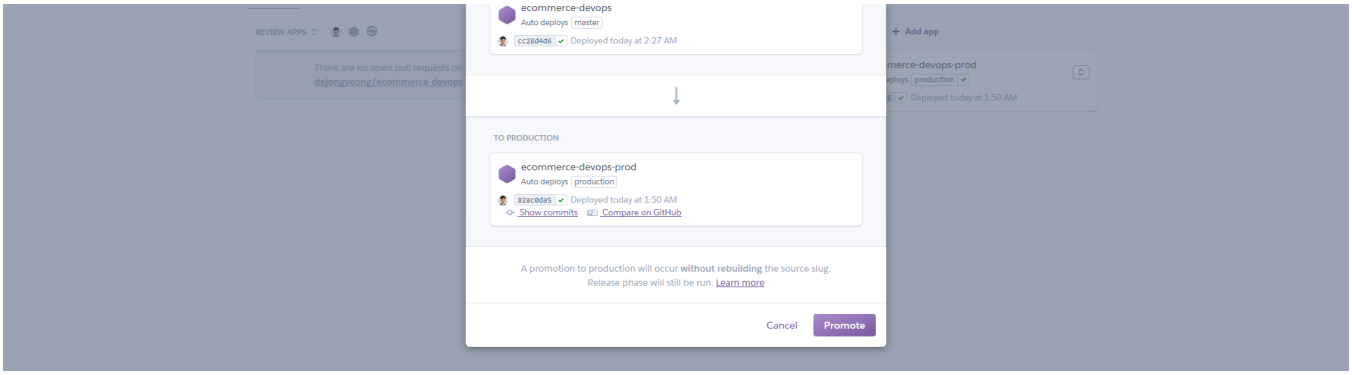


Figure 64: Promote prompt dialog in Heroku dashboard.

Step 2: Click on the **ecommerce-devops-prod** in the build pipeline and configure application variables (same as ecommerce-devops configuration variable) in Heroku dashboard.

Figure 65: Heroku dashboard configuration variables in production stage.

Step 3: Configure a Heroku Postgres add-ons under overview section in the production pipeline dashboard.

Step 4: Run git remote add production git@heroku.com:<production-appname>.git command to add heroku remote production as shown below.

```
PS D:\IT Tralee\ecommerce-devops> git remote -v
heroku  https://git.heroku.com/ecommerce-devops.git (fetch)
heroku  https://git.heroku.com/ecommerce-devops.git (push)
origin  https://github.com/dejongyeong/ecommerce-devops.git (fetch)
origin  https://github.com/dejongyeong/ecommerce-devops.git (push)
production      https://git.heroku.com/ecommerce-devops-prod.git (fetch)
production      https://git.heroku.com/ecommerce-devops-prod.git (push)
```

Figure 66: Git remote for Heroku production stage.

Step 5: Run heroku run php artisan migrate:refresh --seed --force --remote=production command to migrate tables and seed data in production environment.

Step 6: Visit the domain found in domains and certificates section under settings tab in the ecommerce-devops-prod pipeline. The ecommerce application should works as expected when configuring the staging stage of the application.

4.5 Heroku Pipeline Overview

A pipeline is a group of Heroku apps that shares the same codebase and each app in a pipeline represents development, review, staging or production in a continuous delivery workflow. It is extremely useful for managing multiple environments of an application. A common Heroku pipeline workflow contains the following steps:

1. Developer creates a pull request to make a change to the codebase.
2. Heroku automatically creates a **review app** for the pull request which allow developers to test the change.
3. It merged into codebase's master branch when the change is ready.
4. The master branch is **automatically deployed** to the pipeline's staging app for further testing.
5. A developer promotes the staging app to **production** when the change is ready, making it available to the application end users.

Summary of Heroku ecommerce-devops build pipelines:

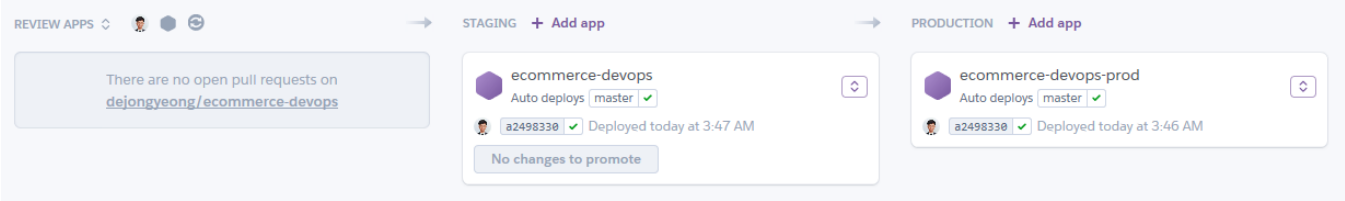


Figure 67: Overall Heroku build pipeline of ecommerce application.

Note: Heroku CI for automated testing costs $10 per month (not using this feature). PHPUnit test is written in Travis CI scripts.

4.6 Heroku Slack Integration

Heroku ChatOps is built around the pipelines and installation of Heroku ChatOps requires Slack permissions to add and approve apps. Assume that you have already logged into Slack.

Step 1: Visit https://devcenter.heroku.com/articles/chatops and click on the **ADD TO SLACK** button to start authorizing Heroku ChatOps in Slack under getting started section.

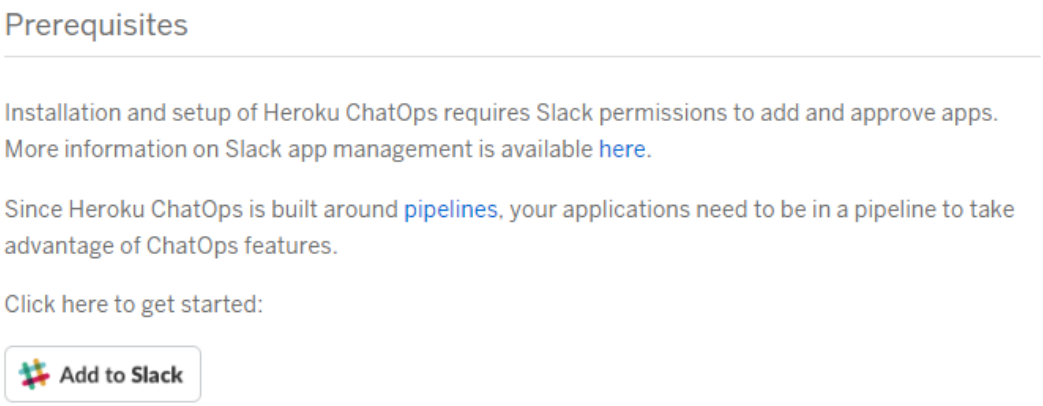


Figure 68: Heroku Slack Integration.

Step 2: Click on the **authorize** button to authorize Heroku ChatOps.

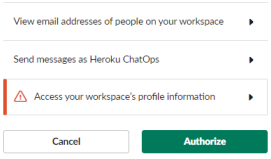


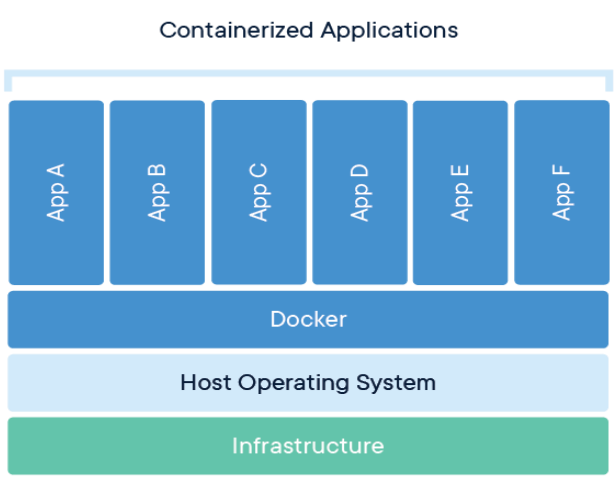
Figure 69: Authorize Heroku ChatOps for Slack integration.

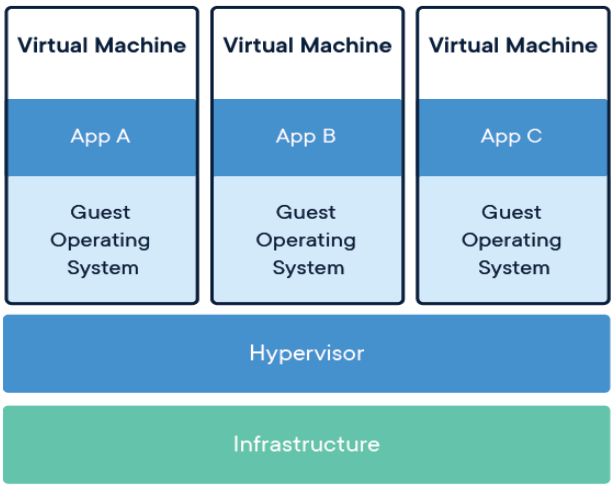
Step 3: Push changes to GitHub to show results of Heroku Deployment notifications.

Figure 70: Heroku deployment Slack notifications.

Section 5: Containerization Technologies

Containerization is an operating system level virtualization method that is used to encapsulate an application in a container with its operating system. Components that are required to run the application are packaged as a single image which can be re-used. Image is executed in an isolated environment that central processing unit (CPU), memory or disk of the host operating system are not shared. This ensures that the execution process inside the container are isolated from process outside the container (Aneja, 2017). Containerization tools like Docker contains the application files, dependencies and libraries of an application to run on an operating system. Containerization technologies remove cross-service dependencies and conflicts to enhance the productivity of developers. It also supports multi-cloud platform where containers can be executed on multiple cloud platform such as Amazon Web Services (AWS), Google Cloud Platform (GCP) et cetera. However, containers are difficult to be monitored when several containers are running on the same server (K, 2018).





Docker is a computer program that performs operating system level virtualization and was first released in 2013. Docker was written in Go programming language and was developed by Docker, Inc. Docker is a containerization tool that is used to run containers. Docker containers are the fastest growing cloud-enabling technology that drives to a new era of computing and application architecture with its lightweight architecture to isolate bundled applications and dependencies, yet highly portable application packages (docker.com, 2019).

5.1 Docker Configuration

Docker is a computer program that performs operating-system level virtualization. Docker was first released in 2013 and was originally developed by Solomon Hykes. It is used to run software packages known as containers. Containers are isolated from each other and each application, tools, libraries and configuration files are bundled together in each container. Containers are run in a single operating-system-kernel which is more lightweight than virtual machines. Containers are created from images that specifies detailed contents. It is often created by combining and modifying standard images from public repositories (docker.com, 2019).

Docker offers a high-level tool with several functionalities as listed below:

- Docker containers are portable across all machines. Docker defines a format of bundling an application and all dependencies into a single object called container which can be transfer to any Docker-enable machine. It can be executed with the guarantee that the execution environment exposed to the application is the same in the development, testing and production phase.
- Docker includes a tool for developers that automatically assemble a container from source code with full control over application dependencies, build tools, packaging et cetera.
- Docker includes version control capabilities like Git for tracking successive versions of a container, inspecting difference between version et cetera (docker.com, 2019).

More functionalities that Docker offers is available in Docker documentation. Please note that the Docker install guide is available in Appendix F.

Guideline below outlined the steps required to create a Docker image of the ecommerce application. It was based on the link below:

Tutorial Link:

https://docs.travis-ci.com/user/docker/#building-a-docker-image-from-a-dockerfile

https://buddy.works/guides/laravel-in-docker

Step 1: Run Docker terminal and change current working path to the root path of ecommerce application.

```
dejong@MSI MINGW64 /d
$ cd IT\ Tralee/ecommerce-devops/

dejong@MSI MINGW64 /d/IT Tralee/ecommerce-devops (master)
$ ls
app/          composer.json  node_modules/      Procfile    routes/                  tests/
app.json      composer.lock  package.json       public/     server.php               vendor/
artisan*      config/        package-lock.json  readme.md   sonar-project.properties  webpack.mix.js
bootstrap/    database/      phpunit.xml        resources/  storage/                 yarn.lock
```

Figure 71: Change path to root path of ecommerce application.

Step 2: Create a **Dockerfile** file in project root directory and paste the following code into the created Dockerfile.

```
FROM php:7.2.0

RUN apt-get update -y && apt-get install -y openssl zip unzip git
RUN curl -sS https://getcomposer.org/installer | php -- --install-
dir=/usr/local/bin --filename=composer
RUN docker-php-ext-install pdo

WORKDIR /app
COPY . /app

CMD composer install --no-interaction
CMD vendor/bin/phpunit
CMD php artisan serve --host=0.0.0.0 --port=8181

EXPOSE 8181
```

Step 3: Run docker build -t ecommerce-devops-image . command on project root directory to create the Docker image. The Docker build process may take some time to run all the process and should receive a successful docker build message as shown below.

```
Step 11/11 : EXPOSE 8181
 ---> Running in cec1c581b5da
Removing intermediate container cec1c581b5da
 ---> 4623fc1c95c8
Successfully built 4623fc1c95c8
Successfully tagged ecommerce-app-image:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories adde
ies.
```

Figure 72: Docker successful build ecommer-devops-image.

Step 4: Run docker-machine ip default command on project root directory to retrieve the default ip address of docker machine.

Step 5: Run docker run -p 81:8181 ecommerce-devops-image command to launch docker containers. The port 8181 is exposed outside the launched container.

Step 6: Nagivate <docker-machine ip default>:81 on the ecommerce application is displayed on the browser as shown below. Please note to rename or remove config.php file in boostrap/cache if file_get_contents error occurs.

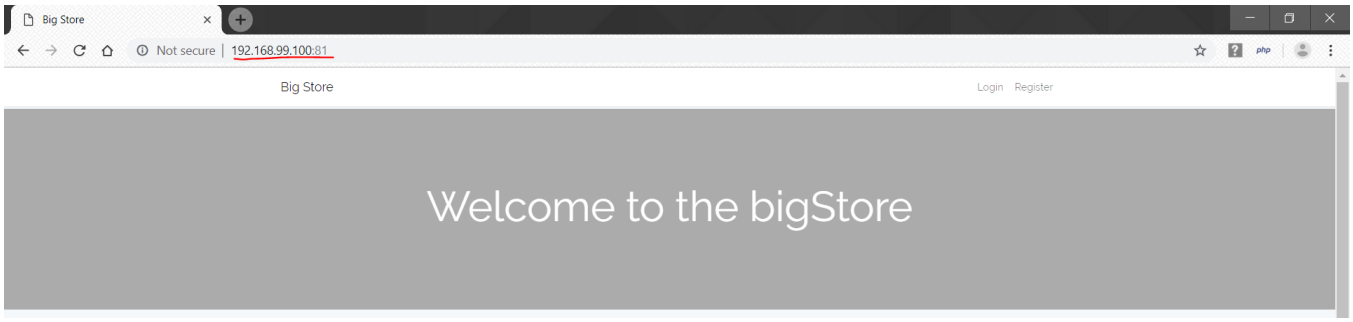


Figure 73: Docker runs application and displayed on browser.

Additional docker commands that can be useful for troubleshooting or to clear cache:

List of Docker Image: docker image ls

```
D:\IT Tralee\ecommerce-devops>docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
ecommerce-devops-image      latest              06df69247631        15 minutes ago      536MB
<none>                      <none>              55f6b1038520        41 minutes ago      536MB
ecommerce-app-image         latest              4623fc1c95c8        About an hour ago   538MB
<none>                      <none>              e556ffed93be        2 hours ago         536MB
php                         7.2.0               6a8b55cfa0a1        15 months ago       351MB
```

Figure 74: List of Docker images.

Remove unused data: docker system prune

```
D:\IT Tralee\ecommerce-devops>docker system prune
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
```

Figure 75: Docker remove unused data.

5.2 Share Images to Docker Hub

Docker images is pushed to a Docker registry after the Docker image is created to share with other users. Developer has a choice to create a private registry and serve images from the server or with a cloud-hosted solution like Docker Hub, Google Container Registry, Amazon EC2 Container Registry or Azure Container Registry (buddy.works, 2019).

Tutorial below shows how a Docker image can be added to **Docker Hub** for sharing purposes:

Step 1: Create a profile at Docker Hub.

Step 2: Run docker build -t [USERNAME]/ecommerce-devops-image . from project root directory to build a Docker image and tag it with Docker Hub username. Please note that it may take some time to complete the process and should receive a successful message as shown below.

```
Removing intermediate container a04cf84daa19
 ---> 7e9265aafa17
Successfully built 7e9265aafa17
Successfully tagged dejong/ecommerce-devops-image:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and
l have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and
```

Figure 76: Docker image build successful.

Step 3: Run docker login command with the created username and password to login to Docker Hub.

```
D:\IT Tralee\ecommerce-devops>docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker
Username: dejong
Password:
Login Succeeded
```

Figure 77: Docker login command login to Docker Hub.

Step 4: Run docker push [USERNAME]/ecommerce-devops-image to push Docker image to Docker Hub as shown below.

```
D:\IT Tralee\ecommerce-devops>docker push dejong/ecommerce-devops-image
The push refers to repository [docker.io/dejong/ecommerce-devops-image]
3f66c3e39f59: Pushing [==>                                                ]  5.909MB/129.7MB
bd58a57e0781: Pushing  2.048kB
683a141c283f: Pushing [==================================================>]  174.1kB
b499db62e4f4: Pushing [==================================================>]  1.917MB
75a7c8152a1d: Pushing [===>                                               ]  3.708MB/51.17MB
de4f26cfafda: Waiting
a3eab6e2a18c: Waiting
a354098c21d7: Waiting
c8c7074012b3: Waiting
d45e3781d4b2: Waiting
8d8f451305fc: Waiting
e416a4904ee4: Waiting
2ec5c0a4cb57: Waiting
```

Figure 78: Push Docker image to Docker Hub.

The docker image is now available in Docker Hub as shown below. Run docker run [USERNAME]/ecommerce-devops-image command to pull and run the image on any machine with Docker installed. Please note that the Docker image can be viewed under the **Repositories** tab after logged in to Docker Hub.

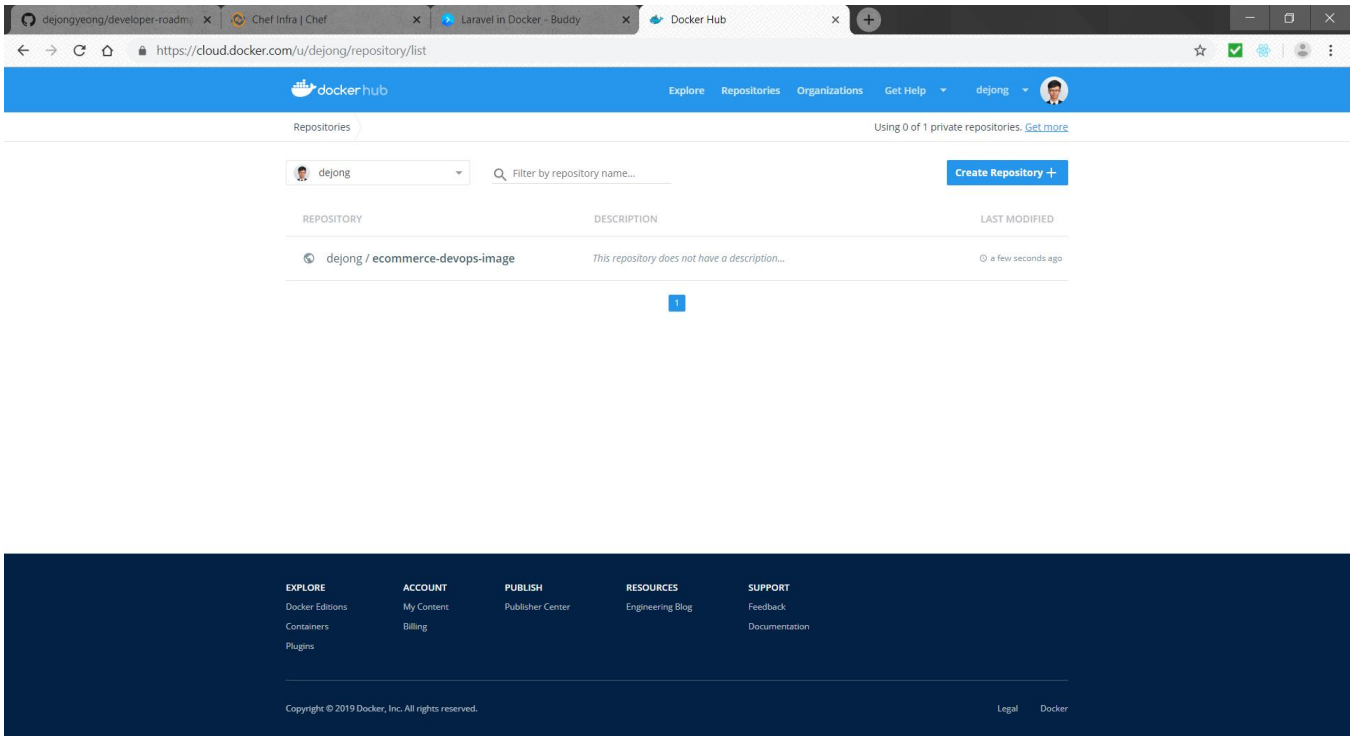


Figure 79: Docker image is now available on Docker Hub.

Section 6: Container Orchestration & Deployment

Container orchestration describes the process of deploying containers on a computer cluster which consists of multiple nodes. Different services that create the application are containerized into separate containers and are deployed across a cluster of physical or virtual machines. This requires the need of container orchestration tool like Kubernetes or Docker Swarm to automates the deployment, management, scaling, networking and the availability of container-based applications (Yegulalp, 2019).

Kubernetes is an open source container orchestration tool that allows developer to deploy and manage multi-container applications at scale. It can be utilized in several architecture deployment due to its modularized production-ready, enterprise-grade and self-healing platform. Kubernetes distributes load balancer amongst containers which aims to relieve the tools and components from issues due to running applications in private and public codes. Issues were resolved by placing containers into groups and naming containers as logical units (Ravindra, 2018).

Docker Swarm or Swarm is an open source container orchestration platform where applications can be deployed as micro-services or services in a swarm cluster. It is used to cluster and schedule Docker containers. Docker Swarm can be used to establish and mange cluster of docker nodes as a single virtual system. It uses scheduling capabilities to ensure enough resources for distributed containers. Containers are assigned to the underlying nodes and resources are optimized by automating the process of container workloads scheduling to run on the most appropriate host. Clusters in Docker Swarm provides IT administrators and developers with the ability to add or remove container iterations as computing demands change (Rouse & Casey, 2019).

Kubernetes was chosen as container orchestration tool. Let's deployed a containerized web application to Google Cloud. Prerequisite is to install Google Cloud CLI as shown in Appendix G and to enable billing for the created Google Cloud project created in Appendix G. Please note that Google Cloud Free-Tier offers $300 free credit for 12 months. Tutorial on how to enable billing is shown in table below:

Tutorial on Google Cloud Billing and GCP Kubernetes:
https://cloud.google.com/billing/docs/how-to/modify-project#enable_billing_for_a_project
https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app#step_1_build_the_container_image

6.1 Push Docker to GCE Container Registry

Option A: Windows CLI

Step 1: Run gcloud auth configure-docker command to configure Docker to use Google Cloud as a credential helper.

Step 2: Determine the container registry name. The container registry name contains information about the hostname of either gcr.io, us.gcr.io, eu.gcr.io or asia.gcr.io; image name; and GCP project ID which can be found in project info as shown below.

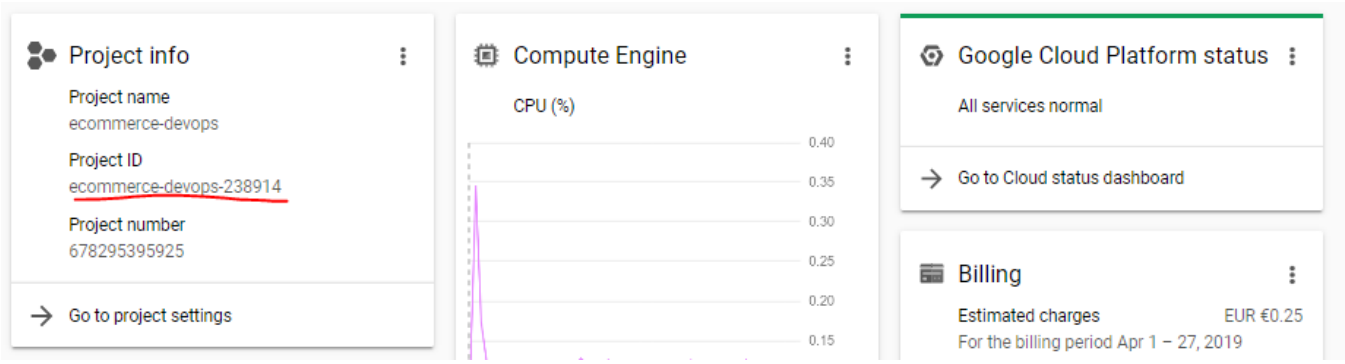


Figure 80: Google Cloud Platform Project Info.

Step 3: Run docker tag [source image] [hostname]/[project id]/[image]:[tag] command to tag local image with container register and run docker images command to verify the tagged image.

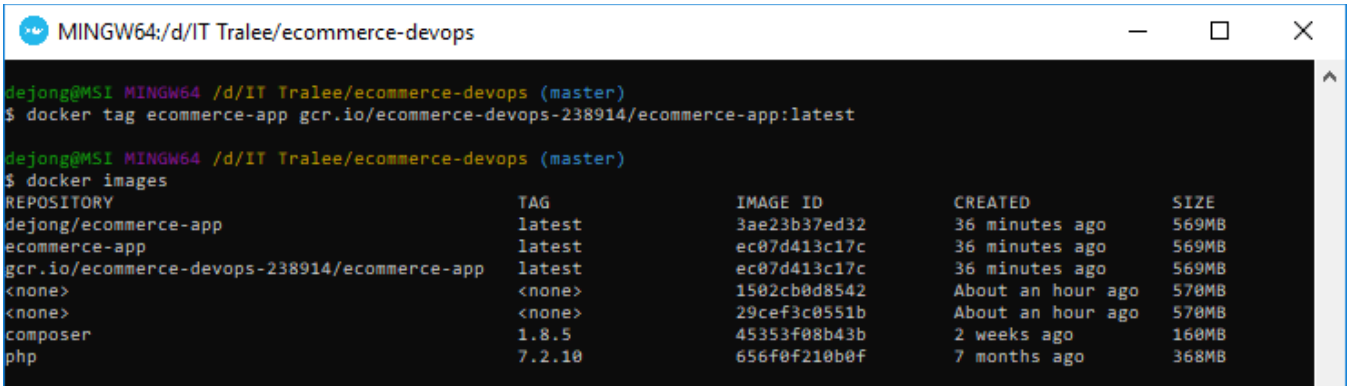


Figure 81: Tag local image with registry hostname and GCP project info.

Step 4: Run docker push [hostname]/[project id]/[image]:[tag] command to push the created tagged image to Google Cloud Container Register and a **SUCCESS** status will be displayed after successful execution.

Figure 82: Push tagged image to GCP Container Registry.

Step 5: Visit [Google Cloud Platform - Container Registry] to verify that the Docker image was pushed successfully to GCP.

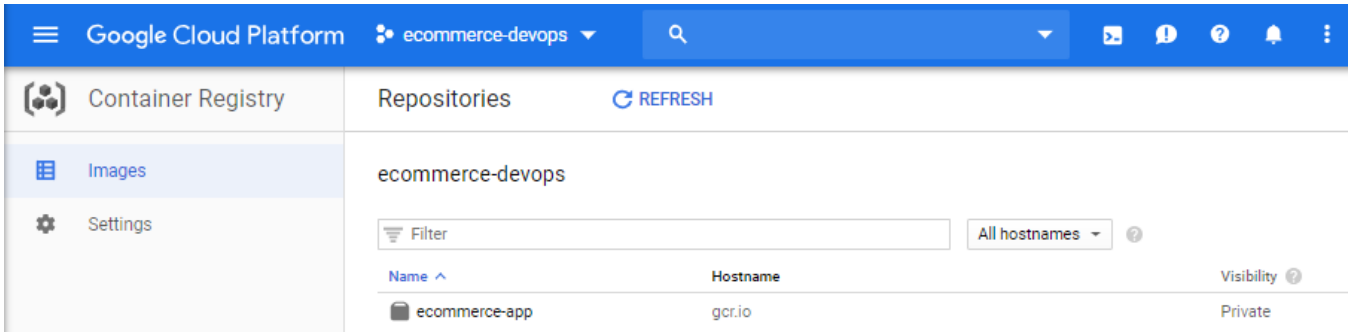


Figure 83: Verify container registry in GCP dashboard.

Option B: Automated Build and Containerize Application (GCE Console)

The GCE Build triggers automatically build containers based on source code or tag changes in a repository. Let's create a build trigger to automatically build containers and push images to GCE Container Registry.

Step 1: Visit GCE Cloud Build webpage and click on **add trigger** button to create build triggers.

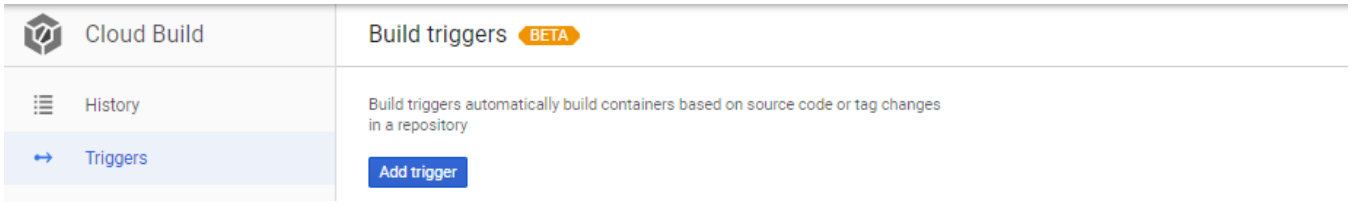


Figure 84: GCE Cloud Build Triggers.

Step 2: Selects a repository hosting option and click on the **continue** button. Complete the GitHub - GCP authorization process if prompted.

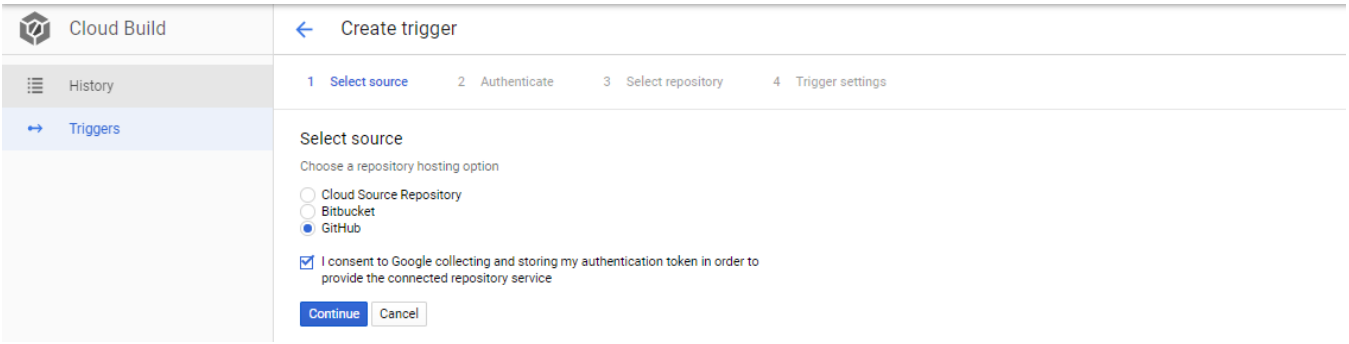


Figure 85: Select repository hosting option in GCE Cloud Build.

Step 3: Selects the GitHub repository that will be used for GCE Cloud Builds from the listed repository and click on the **continue** button.

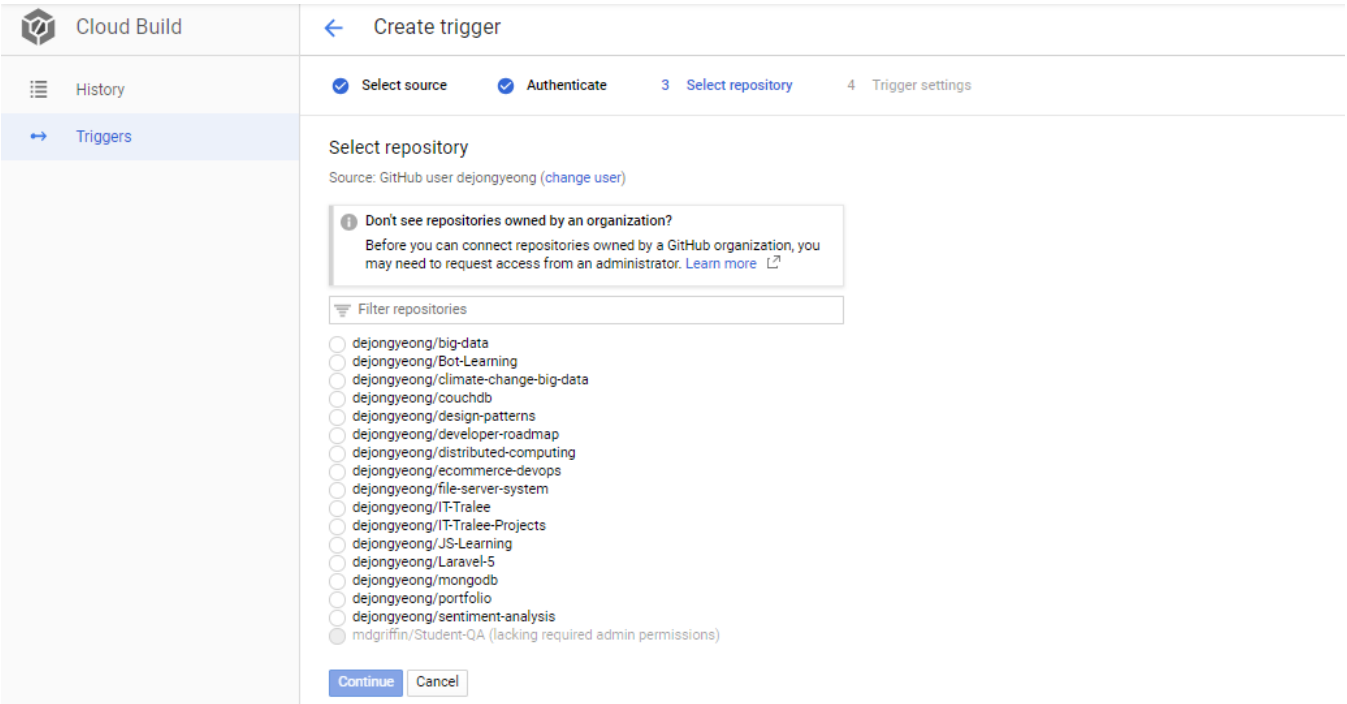


Figure 86: Select GitHub repository for GCE Cloud Build.

Step 4: Enter a name for the GCE Build trigger and click on the **CREATE TRIGGER** button to crate the build trigger.

Image name can be amended in the format of [HOSTNAME]/[PROJECT_ID]/[IMAGE_NAME]:[TAG]

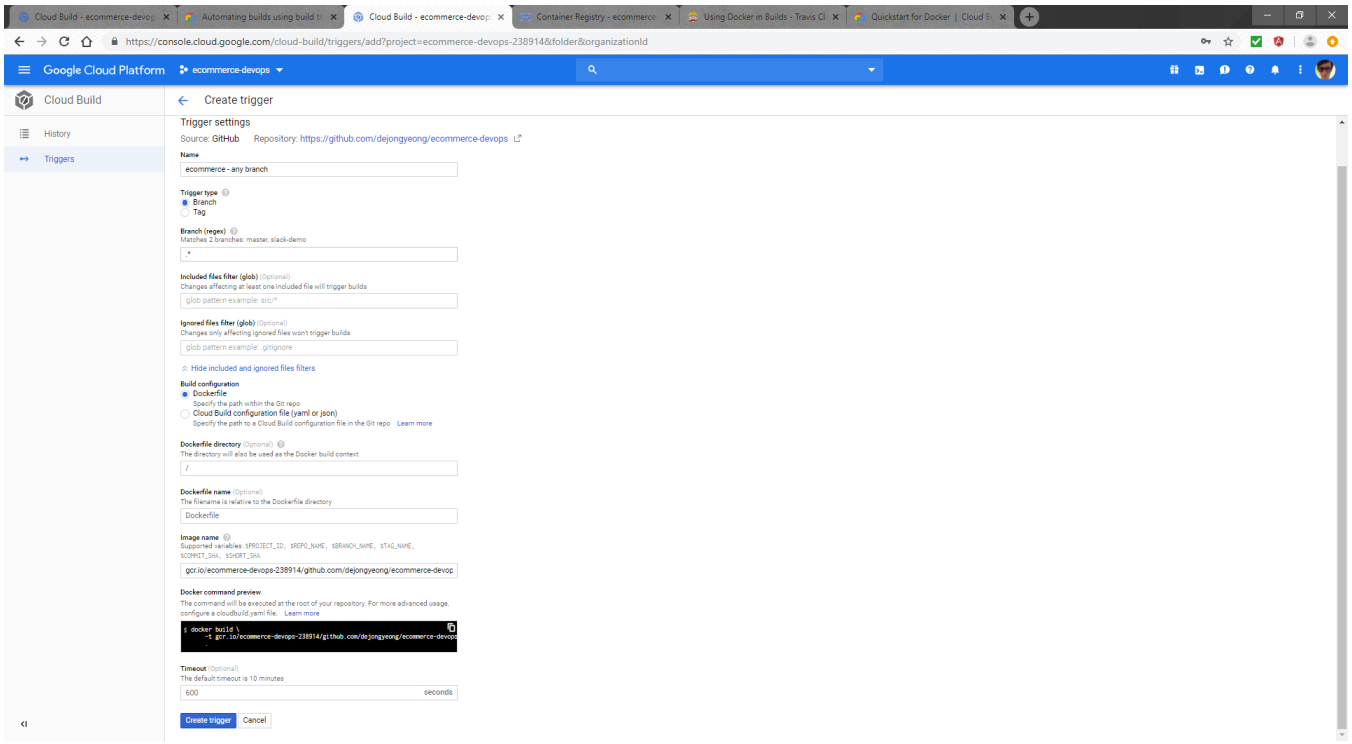


Figure 87: Create GCE Cloud Build triggers.

Step 5: Push changes to GitHub, e.g. git commit --allow-empty to enable automated GCE Cloud Build trigger and push the containerized application to GCE Container Registry. History of GCE Cloud Build can be seen in the history section.

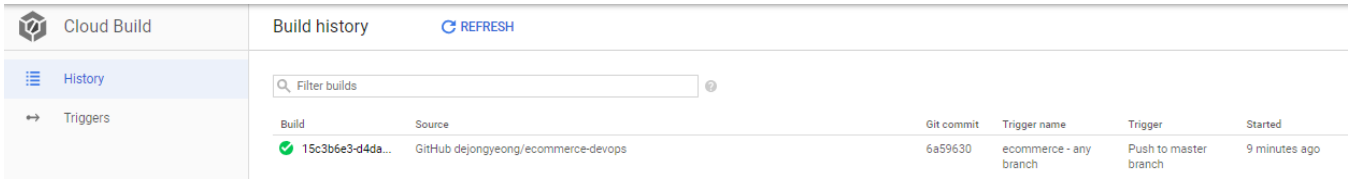


Figure 88: GCE automated build history.

Visit GCE Container Registry to look for the created

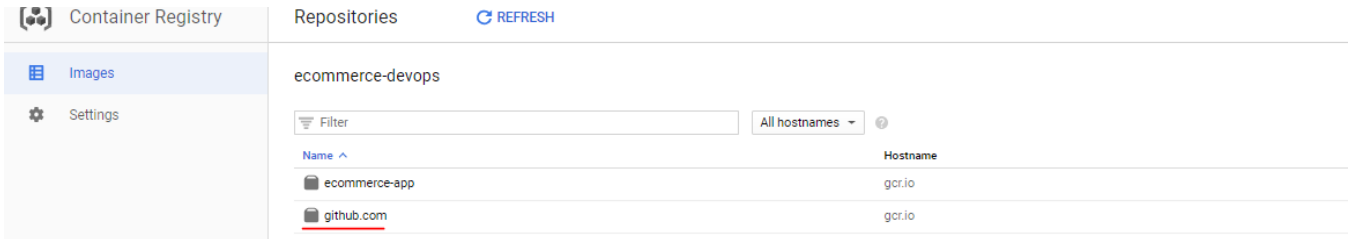


Figure 89: Automatically push image to Container Registry

6.2 GKE Container Cluster

Container cluster is required to be created to run the container image after Docker image is stored in a registry. A cluster consists of a pool of Compute Engine VM instances running Kubernetes. Let's create container cluster to run container image.

Option A: Google Cloud Platform Console

Step 1: Select Kubernetes Engine > Clusters from the navigation menu in GCP online dashboard.

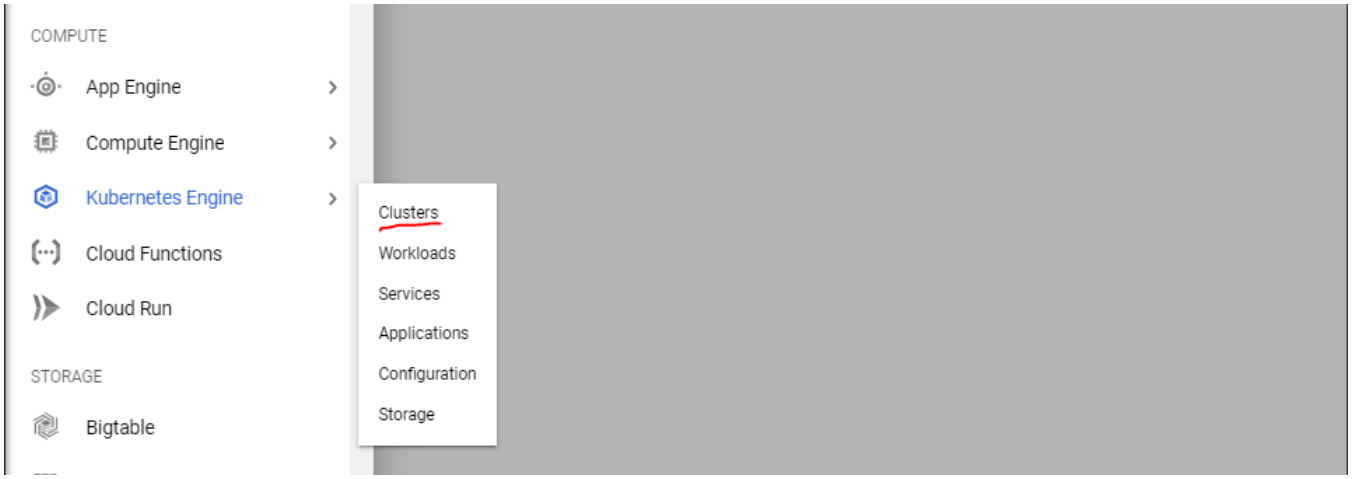


Figure 90: GCP Console create clusters.

Step 2: Click on **create cluster** button to create cluster of size 2 and a name of ecommerce-cluster. The green tick in the *name* column indicates that the cluster is running.

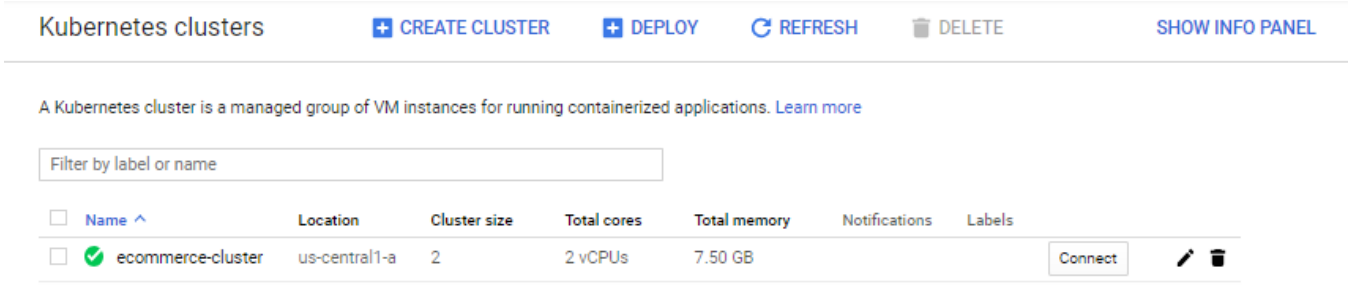


Figure 91: Create GKE cluster of size 2 and a name of ecommerce-cluster.

Step 3: Click on the **connect** button shown above and run the command in windows command line interface to connect to the cluster.

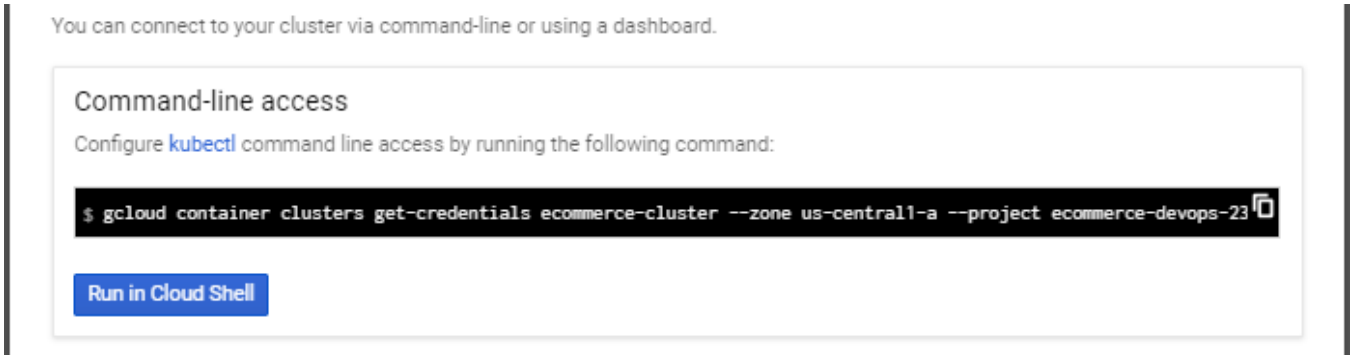


Figure 92: Command to connect to the created cluster.

Step 4: Run gcloud container clusters get-credentials ecommerce-cluster command in windows command line interface if an existing GKE cluster is used or cluster is created through Google Cloud Platform console.

Option B: Windows CLI

Step 1: Run gcloud container clusters create ecommerce-cluster --num-nodes=2 command to create a two-node cluster named ecommerce-cluster. Please note that cluster may take several minutes to be created.

Step 2: Run gcloud compute instances list command to verify the status of the created cluster. It is expected to have a status of **RUNNING**.

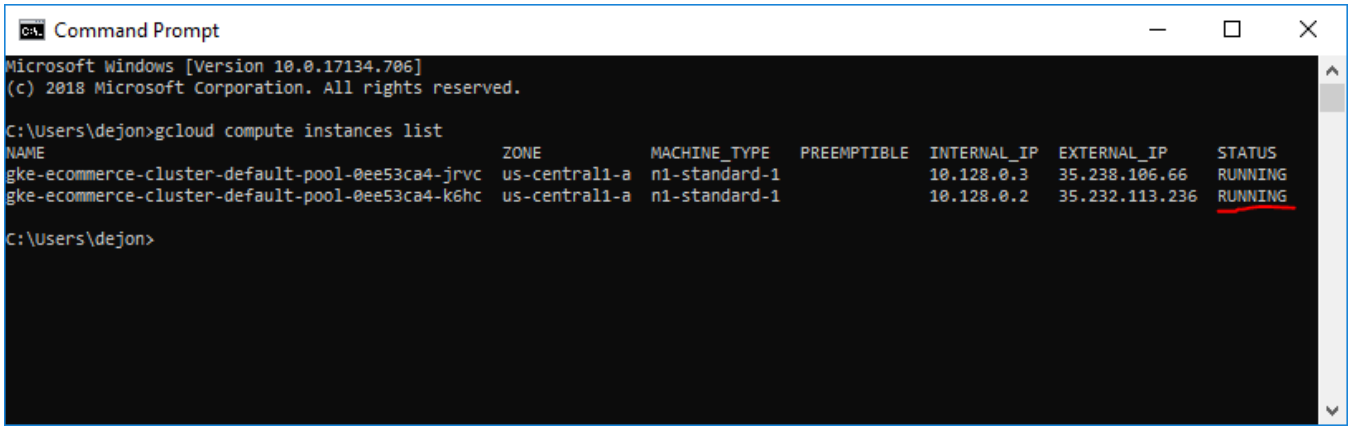


Figure 93: Command to verify the status of created GKE cluster.

6.3 Deploy Containerized Web Application

The Kubernetes command-line tool is used to communicate with the Kubernetes cluster management system to manage and deploy applications on a GKE cluster. Kubernetes is a configuration management tool that represents applications as pods which are units that represent a container or a group of tightly-coupled containers.

Let's create a deployment named ecommerce-web on the created cluster from Windows CLI with the image created manually in Section 6.1 - Option A.

Step 1: Run kubectl run ecommerce-web --image=gcr.io/[project id]/[image]:[tag] --port 8181 command to create a deployment on the created cluster, listening on port 8181. A created message is displayed back to user after successful deployment as shown below.

```
C:\Users\dejon>kubectl run ecommerce-web --image=gcr.io/ecommerce-devops-238914/ecommerce-app:latest --port 8181
deployment.apps/ecommerce-web created
```

Figure 94: Deploy successful message of GKE cluster with Kubernetes CLI.

Step 2: Run kubectl get pods command to verify the created pods. Please note that pods may take several minutes to display.

```
C:\Users\dejon>kubectl get pods
NAME                             READY     STATUS    RESTARTS   AGE
ecommerce-web-7ff48d8768-t8kdc   1/1       Running   0          1h
```

Figure 95: Command to verify the created Kubernetes pods.

Deployment of containerized web application to GCP Kubernetes Engine, known as workload is shown in GCP online console.

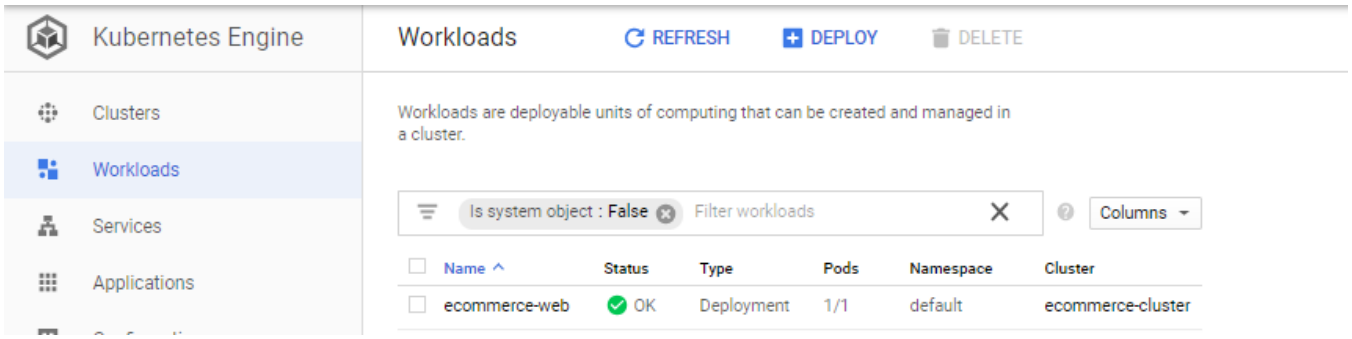


Figure 96: Workloads in GCP Kubernetes Engine.

6.4 Expose Containerized Web Application to Internet

Step 1: Run command kubectl expose deployment ecommerce-web --type=LoadBalancer --port 80 --target-port 8181 to explicitly expose the application to traffic from the Internet. The application is assigned with an external IP address which can be assessible from the Internet. An exposed message is displayed back to user after successful deployment as shown below.

```
C:\Users\dejon>kubectl expose deployment ecommerce-web --type=LoadBalancer --port 80 --target-port 8181
service/ecommerce-web exposed
```

Figure 97: Expose successful message of container with Kubernetes CLI.

GKE creates a service resource which provides networking and IP support to pods of the application. Please note that Load Balance is subjected to billing. The exposed containerized web, known as service is shown in GCP online console.

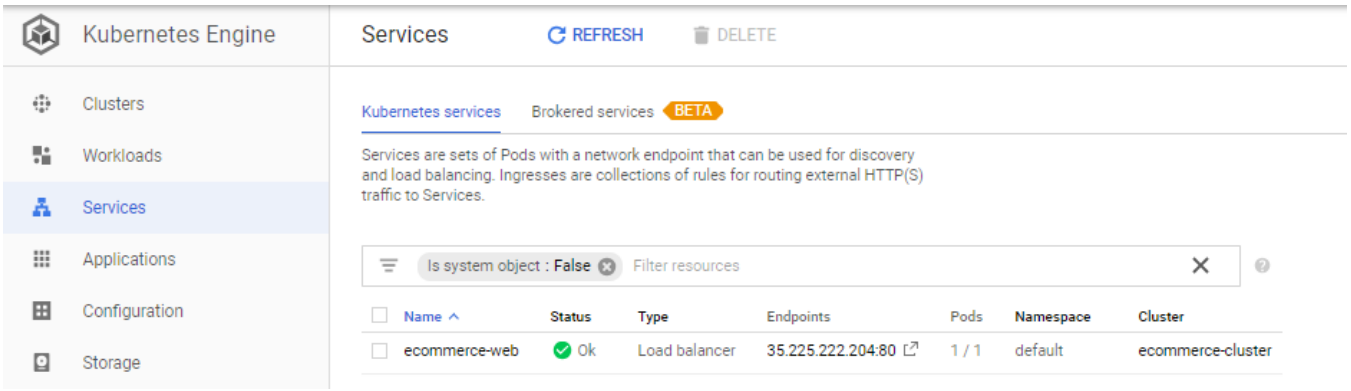


Figure 98: Service in GCP Kubernetes Engine.

Step 2: Run kubectl get service command to retrieve a list of service resource which contains information of external IP address. Visit the ecommerce web application with the external IP address in the browser.

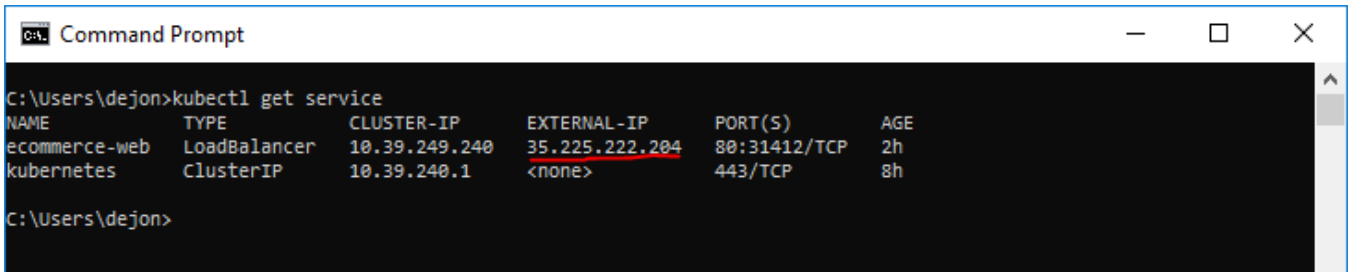


Figure 99: Command to retrieve the external IP of exposed applications.

6.5 Scale Up Web Application

Scale Up, also known as vertical scaling is the process of adding additional resources to an existing system to reach a desired state of performance. Additional resources like compute, memory, storage or network can be added to the system to keep the performance at desired levels. Applications often moved to a more powerful instances and may migrate to a different host when scaling up applications were performed in the cloud. It is important that the execution of vertical scaling process of an application is transparent to client (Schoeb, 2019).

Let's scale up the ecommerce web application of GCP with additional replicas.

Step 1: Run kubectl scale deployment ecommerce-web --replicas=3 command on Windows CLI to create additional replicas to the ecommerce-web deployment resources.

```
C:\Users\dejon>kubectl scale deployment ecommerce-web --replicas=3
deployment.extensions/ecommerce-web scaled
```

Figure 100: Command to scale up the web application.

Step 2: Run kubectl get deployment ecommerce-web command to verify that the new replicas are running on the cluster.

```
C:\Users\dejon>kubectl get deployment ecommerce-web
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
ecommerce-web   3         3         3            3           11h
```

Figure 101: Command to verify the created replicas.

6.6 Cloud Provisioning GKE Clusters

Cloud provisioning is the allocation of resources and services to a customer. It is a key feature of cloud computing model that relates how a customer obtains cloud services and resources from a cloud provider. Cloud provisioning process can be orchestrated with either advanced provisioning model, dynamic provisioning model or user self-provisioning model (Rouse, 2019):

Advanced Provisioning Model	• Customer signs format contract of service with cloud provider. • Cloud provider prepares and distributed the agreed-upon resources or servicers to customer. • Flat fee or billed monthly.
Dynamic Provisioning Model	• Customer purchases cloud resources or services based on average consumptions need. • Cloud provider deploys and adjusts resources to match customer needs. • Pay-per-use billing method.
User Self-Provisioning Model	• Customer purchases resources or services from cloud provider via web interface. • Cloud provider makes resources or services available shortly after payment. • Credit card billing method.

Google Cloud Platform offers a free trial services with $300 free trial credit. A Beta release of GKE cluster node auto-provisioning feature is released and is provided free of charge in GKE release 1.11.2. It is a mechanism of cluster auto-scaler which manages a list of node pools automatically. The cluster auto-scaler considers new nodes only from the set of the chosen node pools without node auto-provisioning feature. However, with auto-provisioning feature, new node pools can be created or deleted automatically (cloud.google.com, 2019). Further information on GKE Cluster auto-scaler is available on GKE documentation. Let's enable node auto-provisioning for GKE ecommerce-cluster.

Tutorial:

https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-provisioning

Step 1: Visit GCP Kubernetes Engine website to retrieve cluster name or use the cluster name created in Section 6.2.

Step 2: Run gcloud beta container clusters update [CLUSTER_NAME] --enable-autoprovisioning --max-cpu 10 --max-memory 64 --zone=[ZONE_NAME] command on Windows CLI or Google Cloud shell to enable node auto-provisioning feature.

```
C:\Users\dejon>gcloud beta container clusters update ecommerce-cluster --enable-autoprovisioning --max-cpu 10 --max-memory 64 --zone=us-central1-a
Updating ecommerce-cluster...done.
Updated [https://container.googleapis.com/v1beta1/projects/ecommerce-devops-238914/zones/us-central1-a/clusters/ecommerce-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-central1-a/ecommerce-cluster?project=ecommerce-devops-238914
```

Figure 102: Command to enable node auto-provisioning feature in GKE.

Note: Enter **Y** to install Google Cloud Beta commands on Windows CLI if prompted and rerun the command.

Where:

- The --enable-autoprovisioning parameter indicates that node auto-provisioning is enable.
- The --max-cpu parameter specifies the maximum number of cores in the cluster.
- The --max-memory parameter specifies the maximum number of GB of memory in the cluster.

Section 7: Monitoring Infrastructures and Applications

Infrastructure monitoring is a virtual mechanism for communication that gives an insight of the data to infrastructure manage. It allows better understanding of the infrastructure and the ability to quantify progress towards the objectives of an organization. In other words, infrastructure monitoring is a process of continual collection and review of data that ensures effective infrastructure management. Information collected from infrastructures monitoring provides infrastructure manager the ability to manage other processes such as service level management, security et cetera (Roush, 2017). There are three cloud service models in cloud computing, which are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Continuous Monitoring of infrastructures and applications ensure collaboration between development and operations teams to optimize user experience.

Tools like Nagios, Azure Web Apps Monitoring, AWS Elastic Beanstalk, Prometheus or Stackdriver et cetera is an open-source tools that is used to monitor infrastructures. StackDriver is chosen to monitor infrastructure of Google Kubernetes cluster of the ecommerce web application in GKE. IaaS vendors like Google offers GCP StackDriver which enables operations for consuming usage and performance information through its Application Programming Interface (API). StackDriver was developed by Google in 2014 which works with both Amazon and Google IaaS vendors (blazemeter.com, 2016).

Tools like AppDynamics and New Relics et cetera is an open-source tools that is used to monitor applications. New Relic APM is chosen to monitor the ecommerce web application in Heroku. It delivers a real-time and trending data about the performance of web application and the satisfaction level of end users experience. New Relic APM is a Software as a Service (SaaS) model that provides an insight with visualizations on web applications. It provides comprehensive monitoring of applications written in Java, Node.js, Python, Go, .NET, PHP and Ruby. A native C/C++ agent is available to instrument and monitor applications written in C/C++ or other languages that are not supported (newrelic.com, 2019).

7.1 Infrastructures Monitoring

Let's monitor the uptime of GCE VM Instance and GKE Container in Google Stack Driver. Uptime checks provide a solution to test for the availability of the public services. Assume that you have already logged in to GCP console.

Step 1: Click on the Navigation Menu and selects StackDriver Monitoring option. It will redirect user to Google SlackDriver webpage.

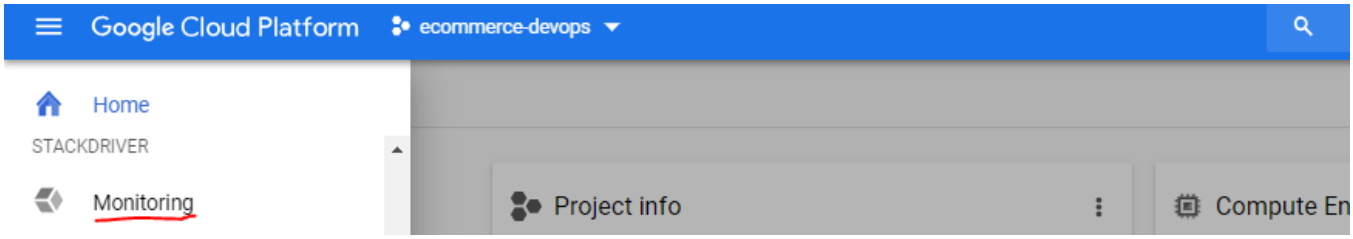


Figure 103: GCP Stack Driver for continuous monitoring.

Step 2: Choose GCP project to be monitored and click on **create workspace** button to create workspace for infrastructure monitoring.

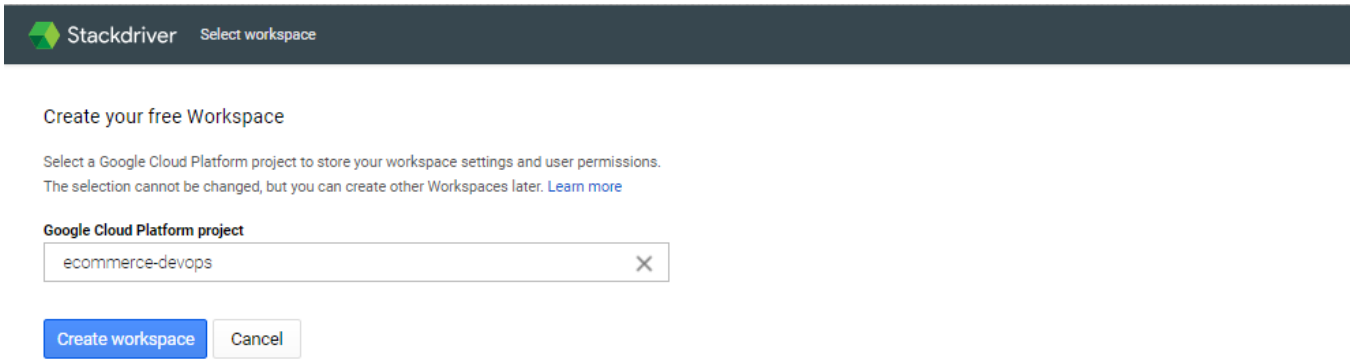


Figure 104: Create workspace to monitor GCP project.

Google StackDriver can also be used to monitor infrastructures hosted in Amazon Web Services. Click on the SKIP AWS SETUP button if no AWS accounts to be monitored.

Step 3: *(Optional)* It is recommended to install the StackDriver agents on each VM instances. Agents collect additional information from VM instances including metrics and logs from third party applications.

Visit [StackDriver Monitoring Agent installation guide] on how to install monitoring agents on each VM instances. Code to install StackDriver monitoring agent and logging agent on each agent is show below.

1. Switch to the terminal connected to your VM instance, or create a new one.
2. Install the Stackdriver agents by running the following commands on your instance:

```
$ # To install the Stackdriver monitoring agent:
$ curl -sSO https://dl.google.com/cloudagents/install-monitoring-agent.sh
$ sudo bash install-monitoring-agent.sh
$
$ # To install the Stackdriver logging agent:
$ curl -sSO https://dl.google.com/cloudagents/install-logging-agent.sh
$ sudo bash install-logging-agent.sh
```

Figure 105: Install StackDriver Monitoring and Logging Agent (optional).

Step 4: Select the frequency of reports that would like to receive and click on the **continue** button.

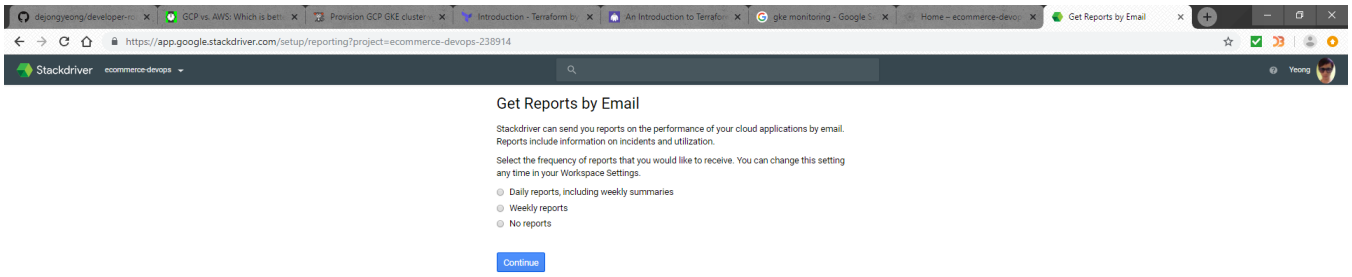


Figure 106: Google StackDriver frequency of reports.

Step 5: Click on the **launch monitoring** button and start monitoring the GCP infrastructure.

Step 6: Click on the **create an uptime check** button to create GKE Container and GCE VM Instance uptime check with a name of *ecommerce web app* and a hostname (external IP address) of GKE services created in [Section 6.4].

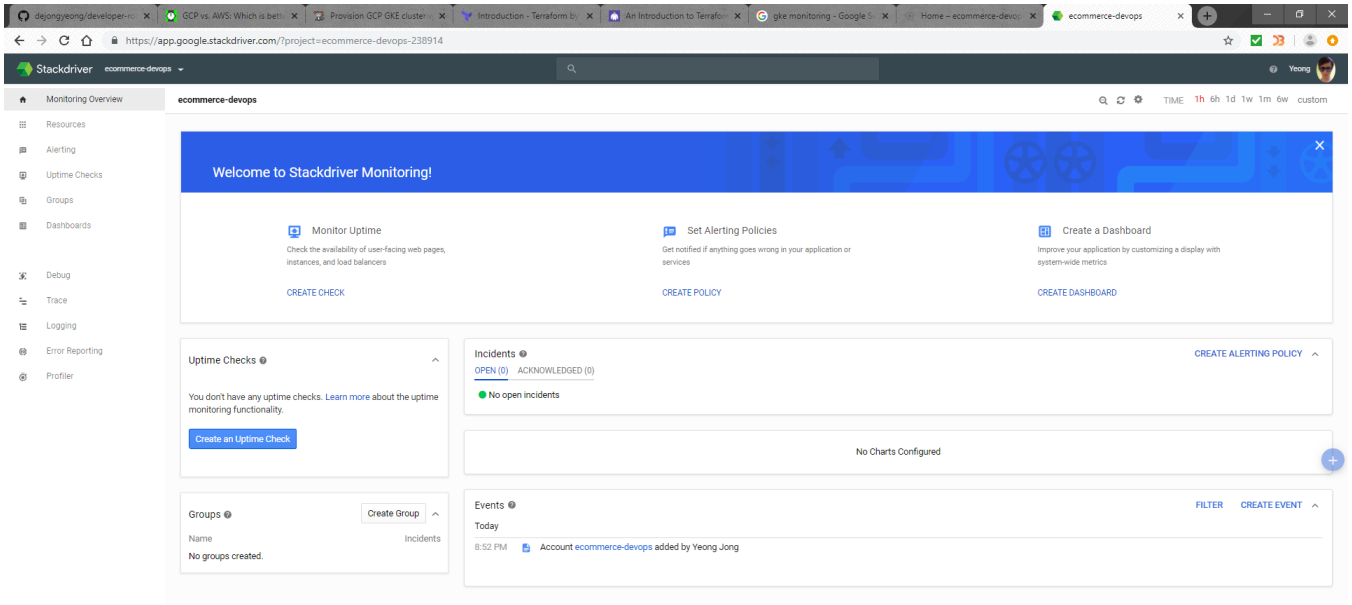


Figure 107: Create Uptime Check for GKE container and GCE VM Instance.

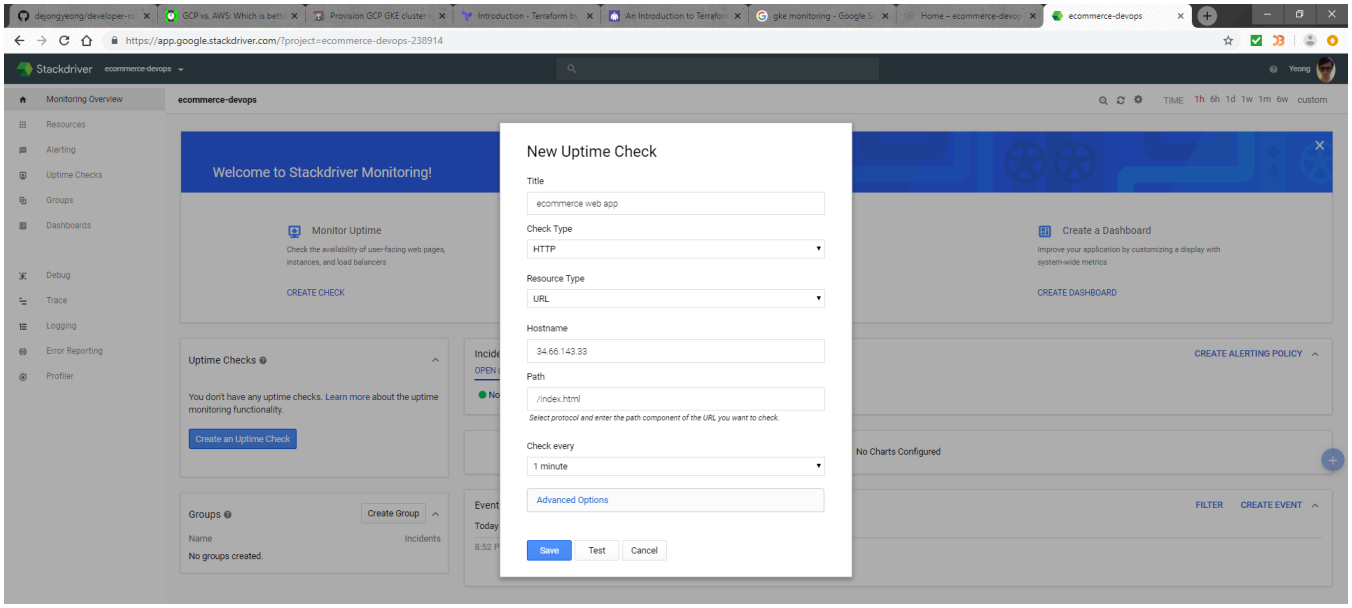


Figure 108: Configure the time of hostname for uptime check.

Please note that it may take an estimate of 10 minutes to show the results of uptime checks on dashboard after configuring the uptime check. A green tick indicates a successful uptime checks as shown below.

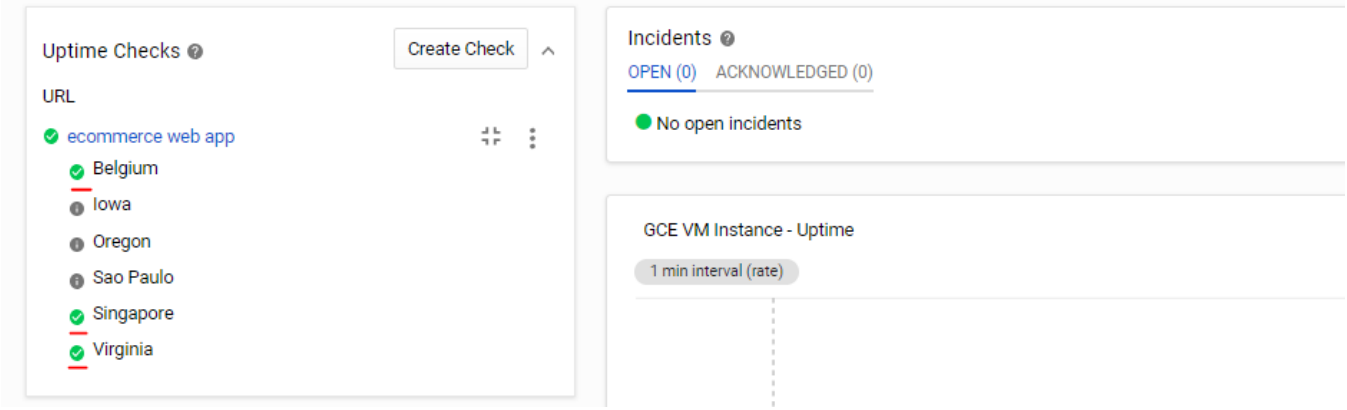


Figure 109: Uptime checks result on Dashboard.

Step 7: Click on the '**+**' sign button on the middle-right of the browser to add results of infrastructure monitoring on dashboard as graph.

Selects a resource type of GKE Container with an uptime metric and click on **save** button to add monitoring results to dashboard. Other metrics like *page fault* can also be added in the chart.

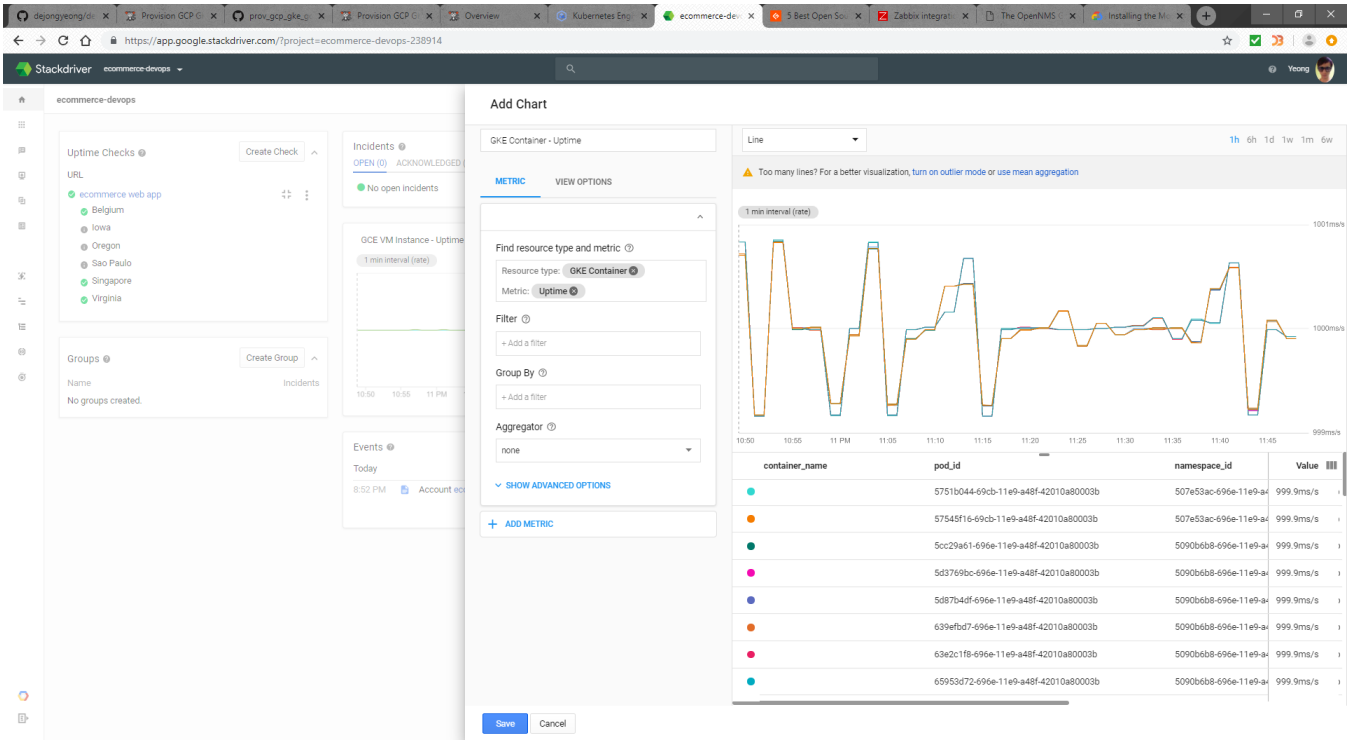


Figure 110: Add monitoring results (Chart) to StackDriver dashboard.

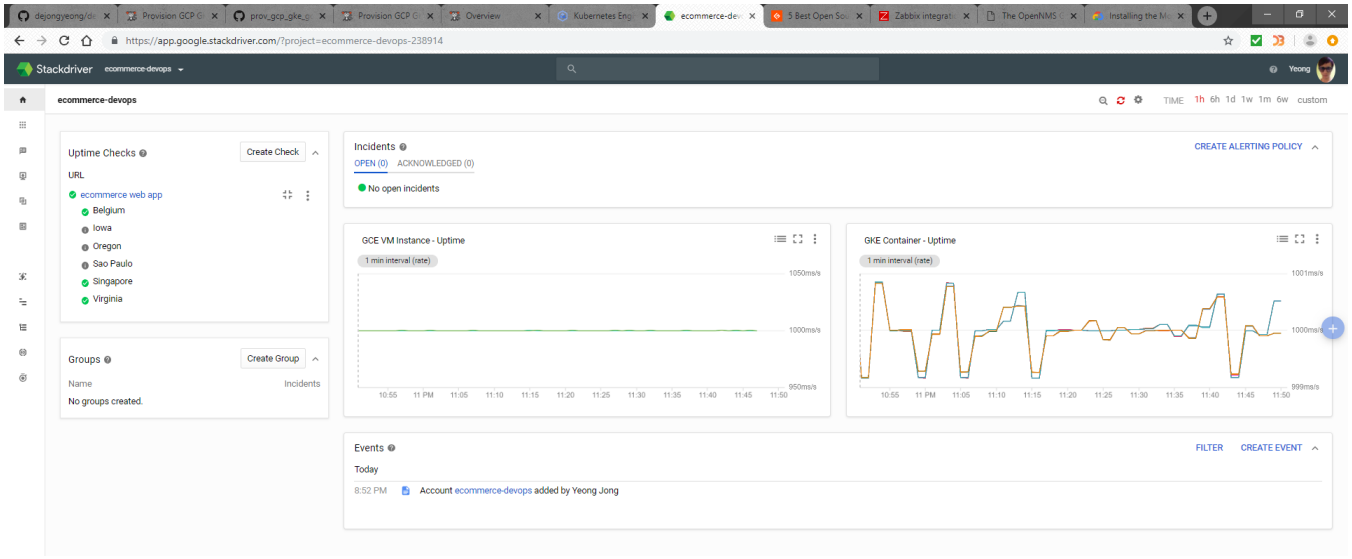


Figure 111: Monitoring results were shown in dashboard.

7.2 Monitoring Applications

Let's monitor the production of ecommerce web application in Heroku. Assume that you have logged in to Heroku with a username and password. Please make sure to verify Heroku account with credit card information (can be removed if not used).

Step 1: Visit Heroku-Addons and click on the monitoring add-ons categories.

Step 2: Select New Relic APM monitoring tools and click on **Install New Relic APM** button to install New Relic APM Heroku add-ons.

Figure 112: New Relic APM Heroku add-ons installation.

Step 3: Select the Heroku apps to be monitored and click on **Provision Add-On** button. It will redirect user to the Heroku New Relic APM monitor resources page.

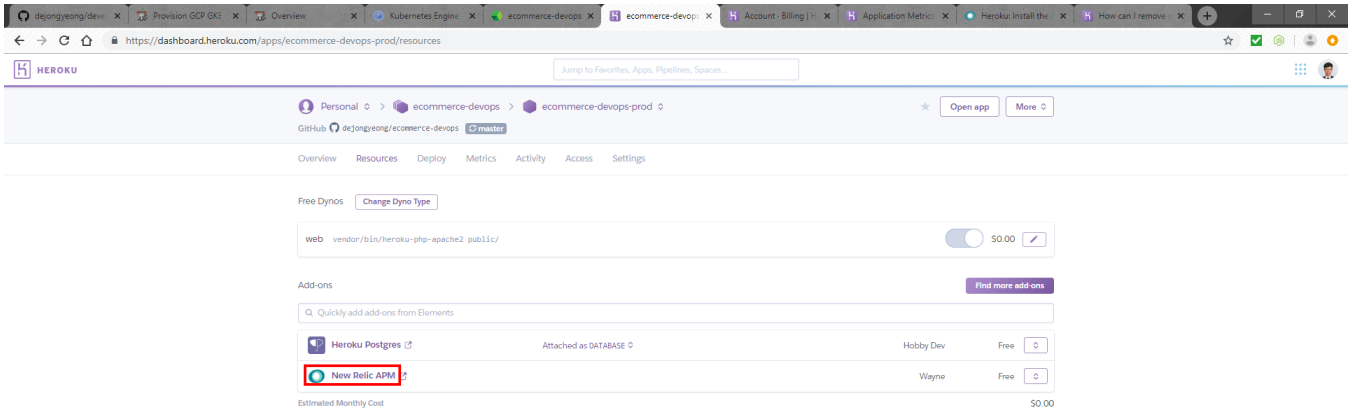


Figure 113: Heroku New Relic APM monitor resources page.

Step 4: Click on the New Relic APM link as shown above and will be redirect to New Relic APM page. Click on **I Agree** button if prompted.

Step 5: Selects the New Relic APM options to start monitoring ecommerce web application and selects the PHP web agent to install.

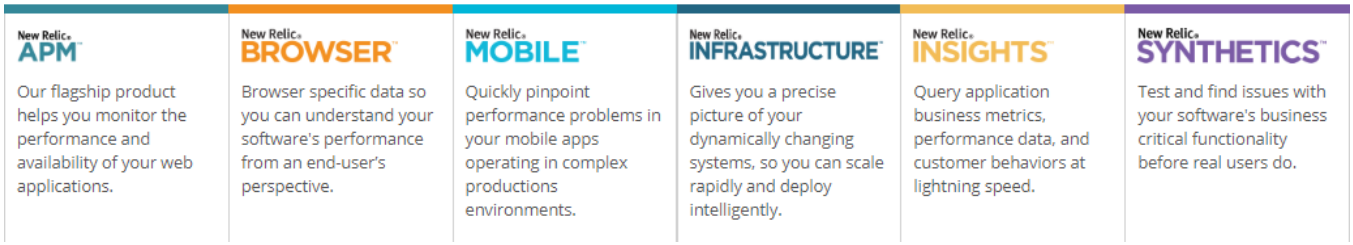


Figure 114: New Relic Monitoring Options.

Tutorial on PHP agent and Heroku is available on:

PHP agent and Heroku Tutorial:
https://docs.newrelic.com/docs/agents/php-agent/advanced-installation/php-agent-heroku

Step 6: Run heroku config:set NEW_RELIC_APP_NAME='YOUR_APP_NAME' --app='APP_NAME' command in project root directory to give a descriptive name of the application.

```
D:\IT Tralee\ecommerce-devops>heroku config:set NEW_RELIC_APP_NAME='ecommerce-webapp' --app=ecommerce-devops-prod
 »   Warning: heroku update available from 7.21.0 to 7.24.1
Setting NEW_RELIC_APP_NAME and restarting ⬚ ecommerce-devops-prod... done, v36
NEW_RELIC_APP_NAME: 'ecommerce-webapp'
```

Figure 115: Configure Heroku New Relic environment variable.

Step 7: Push changes to Heroku, e.g. git commit --allow-empty to enable PHP extension during build, in other words, application is automatically monitored after a successful build.

Step 8: Visit domain found in settings section of the application to generate traffics to the application.

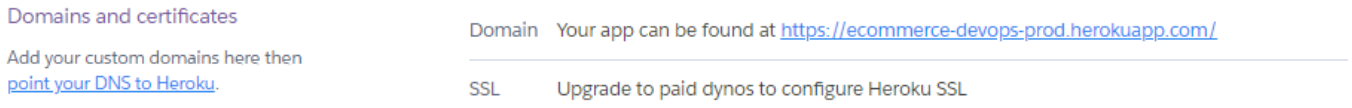


Figure 116: Domain of the monitored application.

Step 9: Visit New Relic APM overview page to view application monitoring results. It shows an overview of the monitoring results which includes web transaction time and throughput. It also shows the Heroku Postgres performance of the ecommerce web application.

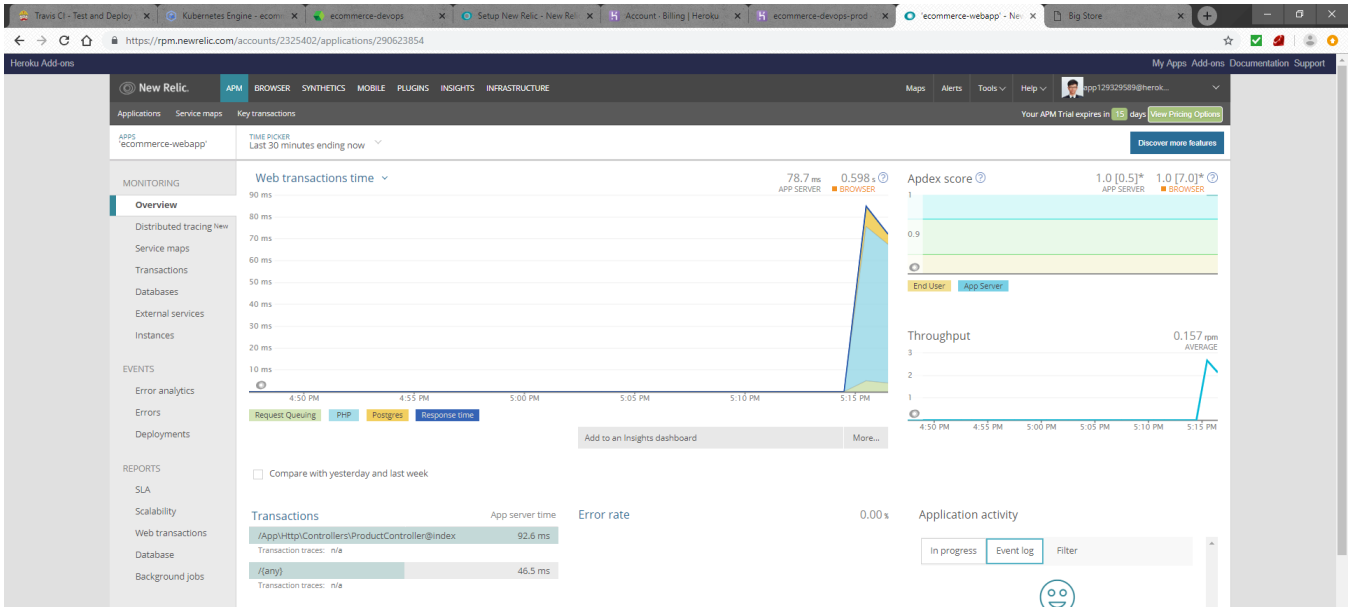


Figure 117: Overview of monitoring results.

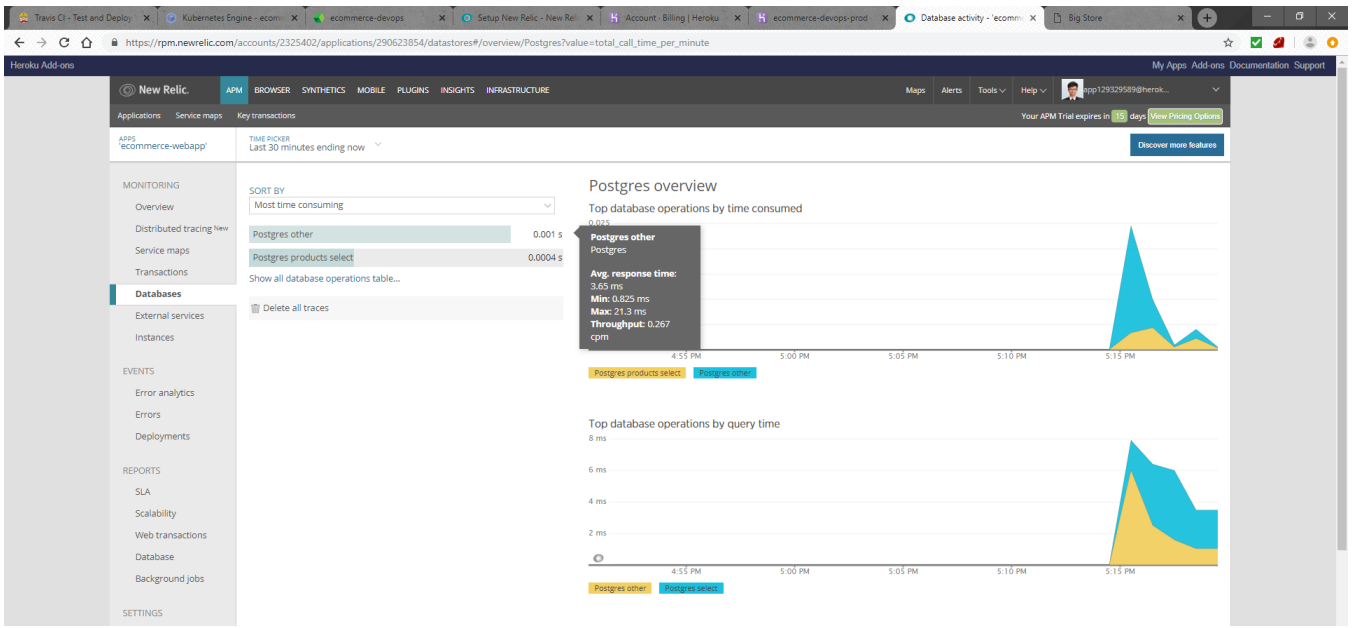


Figure 118: Monitoring of Heroku Postgres performance.

ISSUE: The APM trial expires in 15 days and requires upgrading to a paid plan.

Section 8: Conclusion

In conclusion, the practical application of E-Commerce Laravel and VueJS found in GitHub for automated deployment pipeline assignment is successfully build. This helps to further understanding the workflow of development operations and tools that can be utilized in each phase. Some issues occur when trying to configure the traditional configuration management tool, Ansible. However, an alternative way to resolve this issue is the use of features in Google Cloud Platform like GKE.

Table below shows a summary of tools and the required software that were utilized for the implementation of DevOps lifecycle.

Laravel PHP Framework	Laravel PHP ecommerce application from GitHub - Fisayo Afolayan.
Visual Studio Code	Integrated development environment.
GitHub	Source control management.
PHP Unit Test	Product and order tests with pre-written test cases - Fisayo Afolayan.
Slack	Build and Deploy notifications.
Travis CI	Continuous integration.
SonarCloud	Cloud-based continuous code quality management.
Heroku / GCP Cloud Build	Continuous deployment.
Docker	Containerized ecommerce application.
Docker Hub	Artefacts management.
Google Kubernetes Engine	Configuration management of containers and provisions of clusters.
Google StackDriver	Infrastructures monitoring.
New Relic APM	Applications monitoring - Heroku add-on.

Table below shows a summary of links or external IP address to visit the ecommerce application.

Heroku	https://ecommerce-devops-prod.herokuapp.com/
GKE web services	http://34.66.143.33/

References

Aneja, J., 2017. *Container Technologies Overview.* [Online]
Available at: https://dzone.com/articles/container-technologies-overview
[Accessed 26 April 2019].

blazemeter.com, 2016. *Top 14 Monitoring Tools that Every DevOps Needs.* [Online]
Available at: https://www.blazemeter.com/blog/top-10-monitoring-tools-every-devops-needs
[Accessed 28 April 2019].

buddy.works, 2019. *Laravel in Docker.* [Online]
Available at: https://buddy.works/guides/laravel-in-docker
[Accessed 2 April 2019].

cloud.google.com, 2019. *Using node auto-provisioning.* [Online]
Available at: https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-provisioning
[Accessed 29 April 2019].

cloudways.com, 2018. *Use Laravel Unit Testing to Avoid Project-Wrecking Mistakes.* [Online]
Available at: https://www.cloudways.com/blog/laravel-unit-testing/
[Accessed 27 March 2019].

docker.com, 2019. *Docker Containerization Unlocks the Potential for Dev and Ops.* [Online]
Available at: https://www.docker.com/why-docker
[Accessed 26 April 2019].

docker.com, 2019. *Docker frequently asked questions (FAQ).* [Online]
Available at: https://docs.docker.com/engine/faq/#what-does-docker-technology-add-to-just-plain-lxc
[Accessed 2 April 2019].

heroku.com, 2019. *Heroku.* [Online]
Available at: https://www.heroku.com/home
[Accessed 27 March 2019].

K, A., 2018. *WHAT IS CONTAINERIZATION IN DEVOPS?.* [Online]
Available at: https://www.linuxnix.com/what-is-containerization-in-devops/
[Accessed 26 April 2019].

Klopp, W., 2018. *New improvements to the Slack and GitHub integration.* [Online]
Available at: https://github.blog/2018-05-17-new-improvements-to-slack-and-github-integration/
[Accessed 28 March 2019].

laravel.com, 2019. *Testing: Getting Started.* [Online]
Available at: https://laravel.com/docs/5.8/testing
[Accessed 27 March 2019].

newrelic.com, 2019. *10 Key Capabilities of New Relic APM.* [Online]
Available at: https://newrelic.com/resource/key-capabilities-new-relic-apm
[Accessed 27 April 2019].

Ravindra, S., 2018. *Kubernetes vs. Docker Swarm: What's the Difference?.* [Online]
Available at: https://thenewstack.io/kubernetes-vs-docker-swarm-whats-the-difference/
[Accessed 25 April 2019].

Rouse, M., 2019. *cloud provisioning.* [Online]
Available at: https://searchitchannel.techtarget.com/definition/cloud-provisioning
[Accessed 29 April 2019].

Rouse, M. & Casey, K., 2019. *Docker Swarm.* [Online]
Available at: https://searchitoperations.techtarget.com/definition/Docker-Swarm
[Accessed 25 April 2019].

Roush, J., 2017. *Infrastructure Monitoring vs Management: What's The Difference.* [Online]
Available at: https://www.bmc.com/blogs/infrastructure-monitoring-vs-management-whats-difference/
[Accessed 27 April 2019].

Schoeb, L., 2019. *CLOUD SCALABILITY: SCALE UP VS SCALE OUT.* [Online]
Available at: https://blog.turbonomic.com/blog/on-technology/cloud-scalability-scale-vs-scale
[Accessed 27 April 2019].

slack.com, 2019. *Slack.* [Online]
Available at: https://slack.com/
[Accessed 28 March 2019].

smartbear.com, 2019. *The Key to an Effective CI/CD Pipeline: Automated Testing.* [Online]
Available at: https://smartbear.com/learn/automated-testing/the-continuous-development-pipeline/
[Accessed 27 March 2019].

sonarcloud.io, 2019. *Documentation.* [Online]
Available at: https://sonarcloud.io/documentation/
[Accessed 27 March 2019].

Soni, M., 2016. *DevOps for Web Development.* 1st ed. Birmingham, UK: Packt Publishing.

Stars, D., 2017. *Continuous Integration. CircleCI vs Travis CI vs Jenkins.* [Online]
Available at: https://hackernoon.com/continuous-integration-circleci-vs-travis-ci-vs-jenkins-41a1c2bd95f5
[Accessed 29 March 2019].

techcrunch.com, 2012. *GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz.* [Online]
Available at: https://techcrunch.com/2012/07/09/github-pours-energies-into-enterprise-raises-100-million-from-power-vc-andreesen-horowitz/?guccounter=1&guce_referrer_us=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_cs=HplslROqrA9BvZbGGStt7A
[Accessed 27 March 2019].

travis-ci.org, 2019. *Test and Deploy with Confidence.* [Online]
Available at: https://travis-ci.org/
[Accessed 31 March 2019].

Tuli, S., 2018. *Learn How to Set Up a CI/CD Pipeline From Scratch.* [Online]
Available at: https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch
[Accessed 29 March 2019].

Virendra, S., 2018. *Comparing top DevOps tools: Docker vs Kubernetes vs Puppet vs Chef vs Ansible.* [Online]
Available at: https://www.znetlive.com/blog/compare-top-devops-tools-docker-kubernetes-puppet-chef-ansible/
[Accessed 23 April 2019].

Yegulalp, S., 2019. *What is Kubernetes? Container orchestration explained.* [Online]
Available at: https://www.infoworld.com/article/3268073/what-is-kubernetes-container-orchestration-explained.html
[Accessed 23 April 2019].

Zulke, D., 2017. *Review Apps is failed to create new instance when `env` is set in `app.json` file?.* [Online]
Available at: https://github.com/heroku/heroku-buildpack-php/issues/227
[Accessed 28 March 2019].

Appendices

Appendix A: Tutorial on SQLite database configuration for Laravel migration.

Step 1: Amend the sqlite database variable in config/database.php as shown below.

```
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' => database_path().'/database.sqlite',
        'prefix' => '',
    ],
```

Figure 119: Modify SQLite database variable in config/database.php file.

Step 2: Create database.sqlite file and remove *.sqlite from .gitignore if contains the *.sqlite code in database folder.

Please note that the sqlite extension in php.ini needs to be uncomment if SQLite fails to run in localhost due to the could not find driver error. Right-click on php.ini and open the configuration file with notepad and search for ;extension=sqlite. Remove the semi-colon as shown below.

```
;extension=pdo_oci
;extension=pdo_odbc
extension=pdo_pgsql
extension=pdo_sqlite
extension=pgsql
;extension=shmop
```

Figure 120: Uncomment sqlite extension in php.ini file.

Appendix B: Tutorial on SQLite database configuration for PHPUnit.

Step 1: Create .env.testing environment file in project root folder and copy the following codes into the created file as shown below.

```
APP_NAME=Laravel
APP_ENV=testing
APP_KEY=base64:PSj6QMiE6efh3OB9XL0UsuZ6+TwmErY88mbVZl/Myn8=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=sqlite
DB_DATABASE=database/test.sqlite

BROADCAST_DRIVER=log
CACHE_DRIVER=array
SESSION_DRIVER=array
SESSION_LIFETIME=120
QUEUE_DRIVER=sync

MAIL_DRIVER=array
```

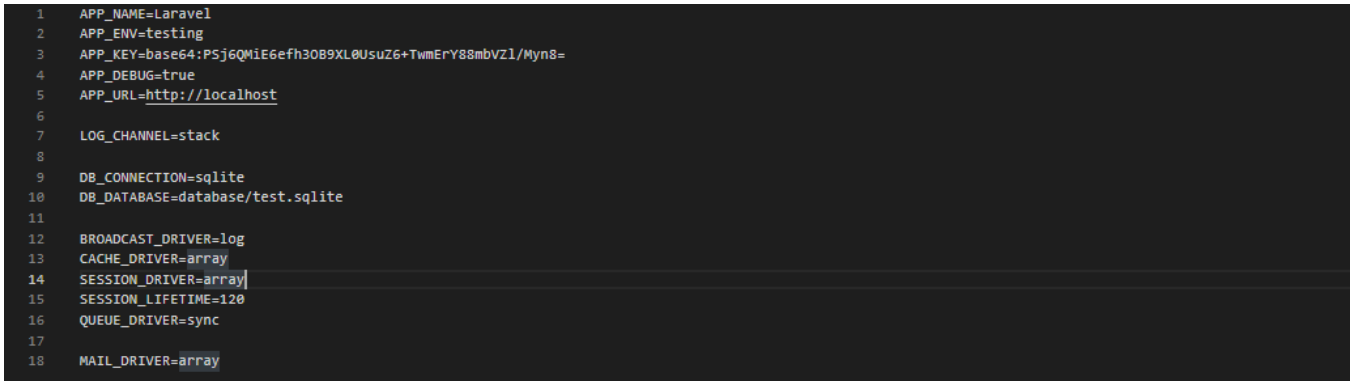


Figure 121: SQLite testing environment file for PHPUnit.

Step 2: Create a test.sqlite file in database folder in root project.

Step 3: Run php artisan migrate:refresh --seed --env=testing --force command to migrate tables and seed data for testing environment.

Step 4: Configure environment name within the php tag in phpunit.xml file as shown below.

```
<php>
    <env name="APP_ENV" value="testing"/>
    <env name="DB_CONNECTION" value="sqlite"/>
    <env name="DB_DATABASE" value=":memory:"/>
    <env name="CACHE_DRIVER" value="array"/>
    <env name="SESSION_DRIVER" value="array"/>
    <env name="QUEUE_DRIVER" value="sync"/>
</php>
```

Figure 122: Testing environment variable for PHPUnit test.

Step 5: Comment out line 86 in ProductTest.php in tests/Unit folder to run all tests successfully on localhost. Error expected status code was 201 but received 500 occurs if line 86 is not commented out. Please note to uncomment line 86 when connected to a continuous integration server for automated testing.

```
PHPUnit 7.5.8 by Sebastian Bergmann and contributors.

.........F.                                                       13 / 13 (100%)

Time: 1 second, Memory: 16.00 MB

There was 1 failure:

1) Tests\Unit\ProductTest::testUploadImage
Expected status code 201 but received 500.
Failed asserting that false is true.

D:\IT Tralee\ecommerce-devops\vendor\laravel\framework\src\Illuminate\Foundation\Testing\TestResponse.php:124
D:\IT Tralee\ecommerce-devops\tests\Unit\ProductTest.php:86

FAILURES!
Tests: 13, Assertions: 56, Failures: 1.
```

Figure 123: Comment line 86 in ProductTest.php to execute tests on localhost.

Step 6: Run ./vendor/bin/phpunit command to run all tests in ecommerce application. All tests should be executed successfully without errors.

```
PS D:\IT Tralee\ecommerce-devops> ./vendor/bin/phpunit
PHPUnit 7.5.8 by Sebastian Bergmann and contributors.

.............                                                     13 / 13 (100%)

Time: 1.06 seconds, Memory: 16.00 MB

OK (13 tests, 56 assertions)
```

Figure 124: Command to execute tests with PHPUnit.

Step 6: Rerun step 3 to roll back testing database.

Step 7: Run php artisan migrate:refresh --seed --force to roll back primary database (in case).

Appendix C: Clear Laravel Application and Configuration Cache

Step 1: Run php artisan cache:clear to clear application cache.

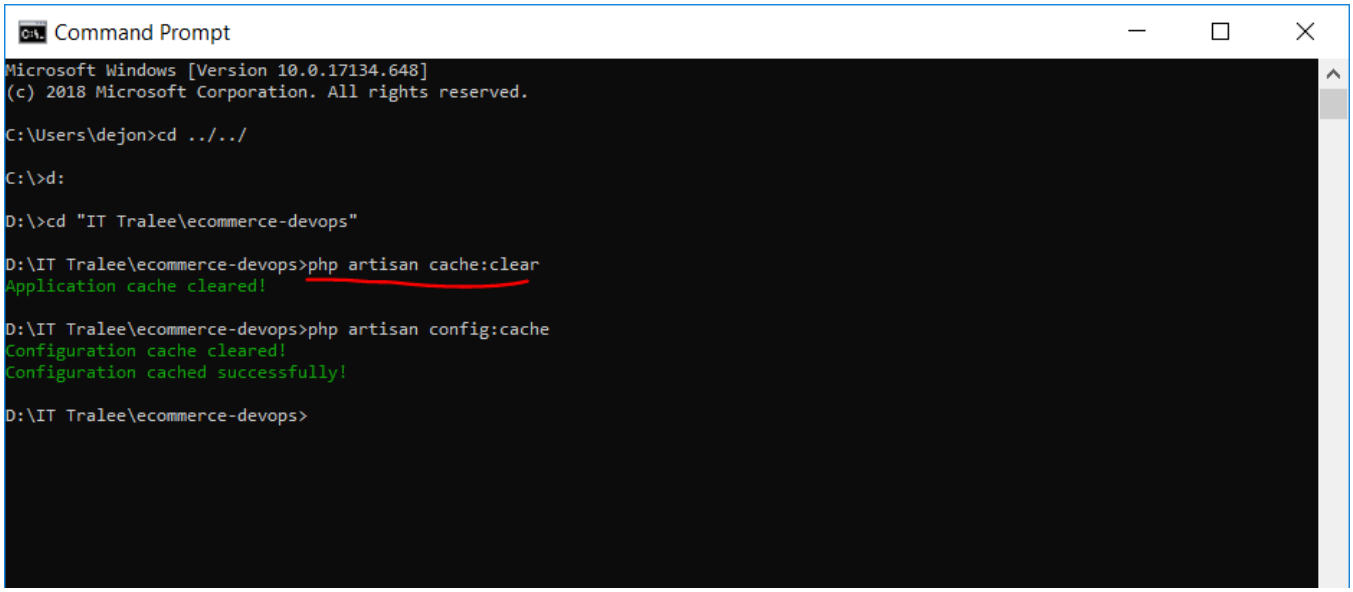


Figure 125: Command to clear application cache.

Step 2: Run php artisan config:cache to clear configuration cache and re-cache configuration of Laravel application.

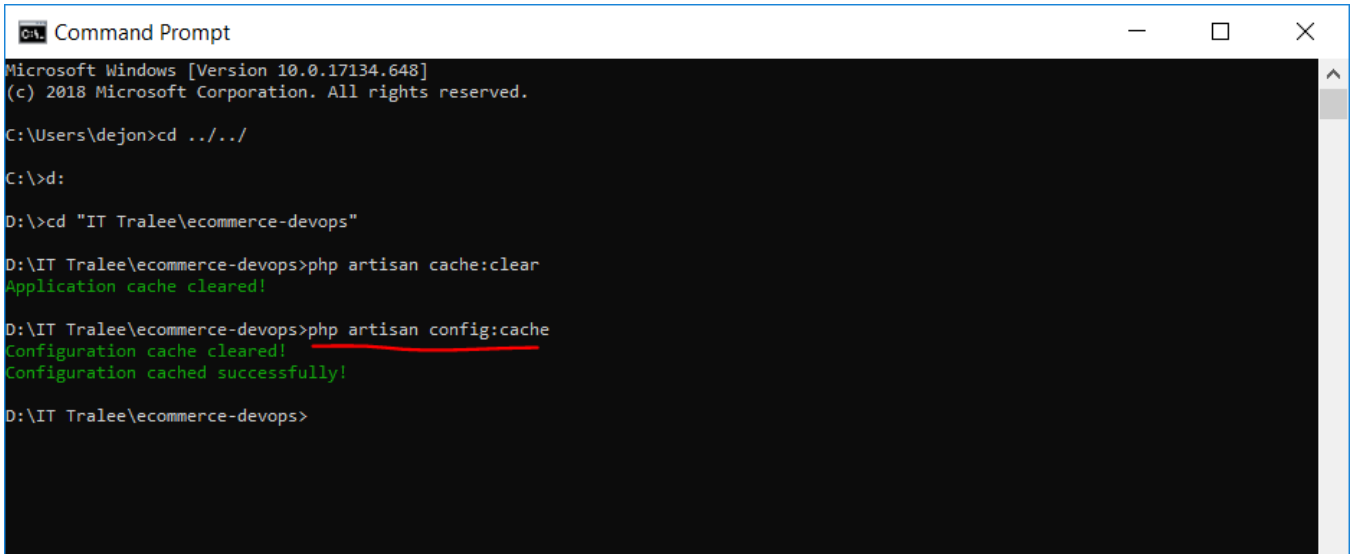


Figure 126: Command to clear configuration cache and re-cache.

Appendix D: Slack and GitHub Integration Installation Guide

Step 1: Install Slack desktop application from Windows Microsoft Store.

Step 2: Visit https://slack.github.com/ to add GitHub services to Slack and click on the "Add to Slack" button.

Step 3: Click on the "Create a new workspace" link to create a new workspace.

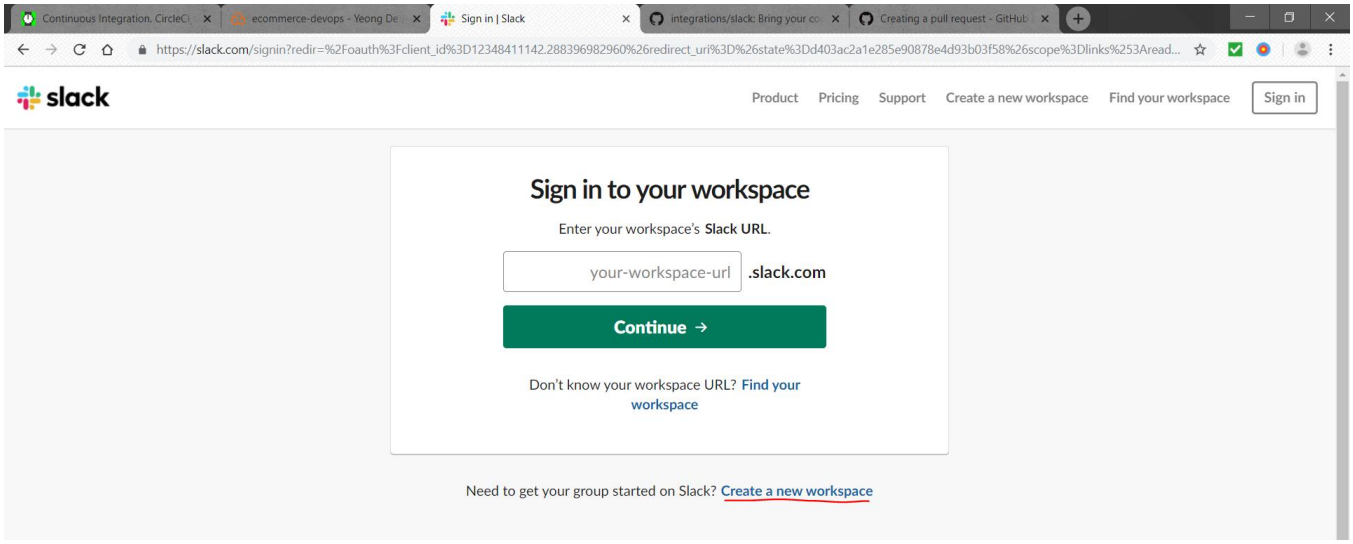


Figure 127: Create a new workspace on Slack.

Step 4: Enter your email address and click the "Next" button. A 6-digit confirmation code is sent to the entered email address for verification.

Step 5: Enter the name your company or team and click the "Next" button.

Step 6: Enter the name of the application that the team is working on and click "Next".

Step 7: Enter team member email address to add teams to the workspace or clicks on the "skip for now" link to skip this process.

Step 8: Visit https://it-tralee.slack.com/apps/A8GBNUWU8-github and click on the install button after signed-in to the created Slack workspace. It will redirect user to the authentication page for authentication.

Step 9: Select or specified selected channels for GitHub notifications and click install. It will redirect user to Slack desktop user. Slack user or invited project team members uses the Slack workspace URL to sign in to the slack channel.

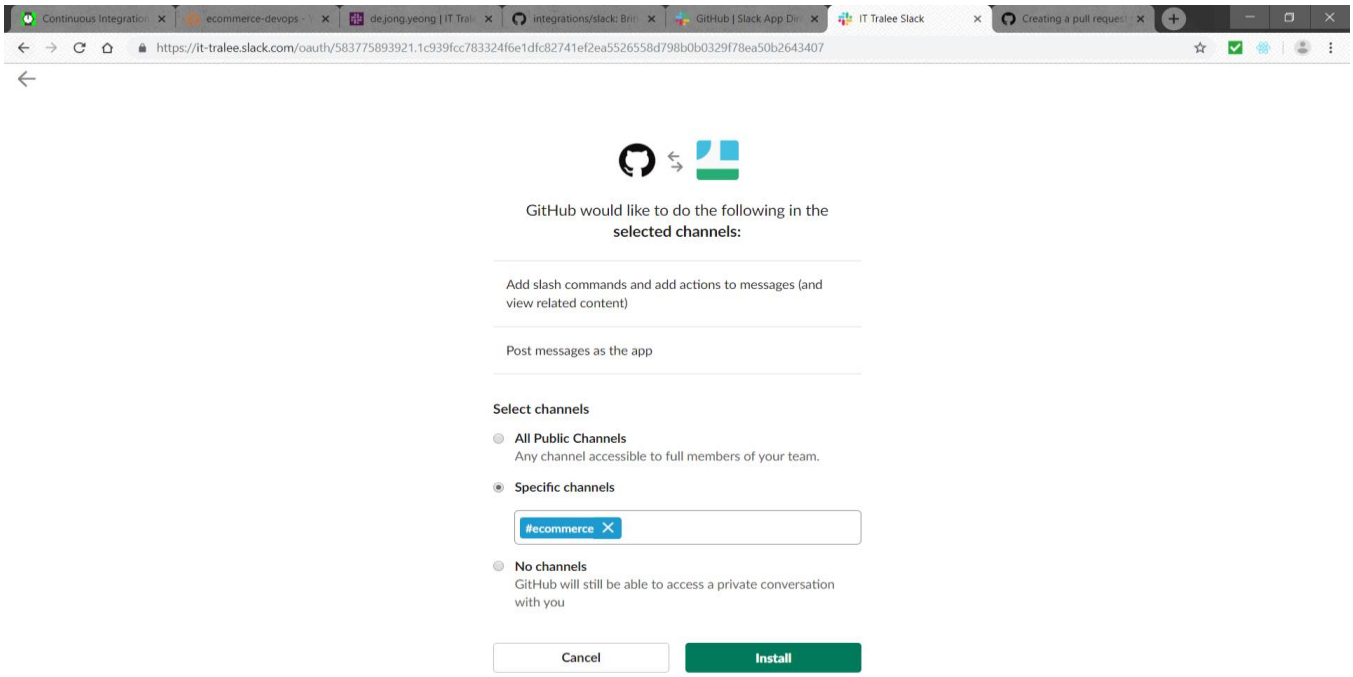


Figure 128: Select or specified selected channels for GitHub notifications.

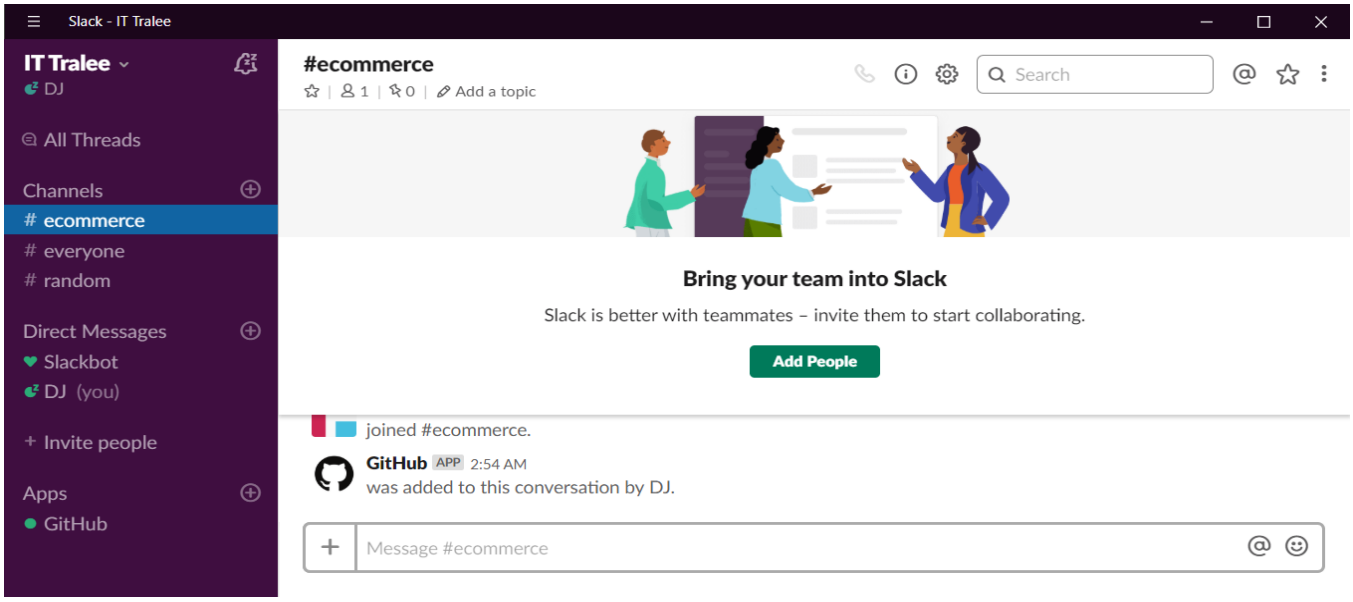


Figure 129: GitHub notification shown in ecommerce channel.

Appendix E: Tutorial on Heroku Postgres database configuration.

Heroku Postgres database with GUI.

Step 1: Create a Heroku account in https://signup.heroku.com/ and login to the verified account. The browser will redirect to user dashboard landing page.

Step 2: Clicks on the icon beside the avatar shown below and clicks on element. The browser will redirect to elements landing page as shown below. User can also browse the elements add-on page from https://elements.heroku.com.

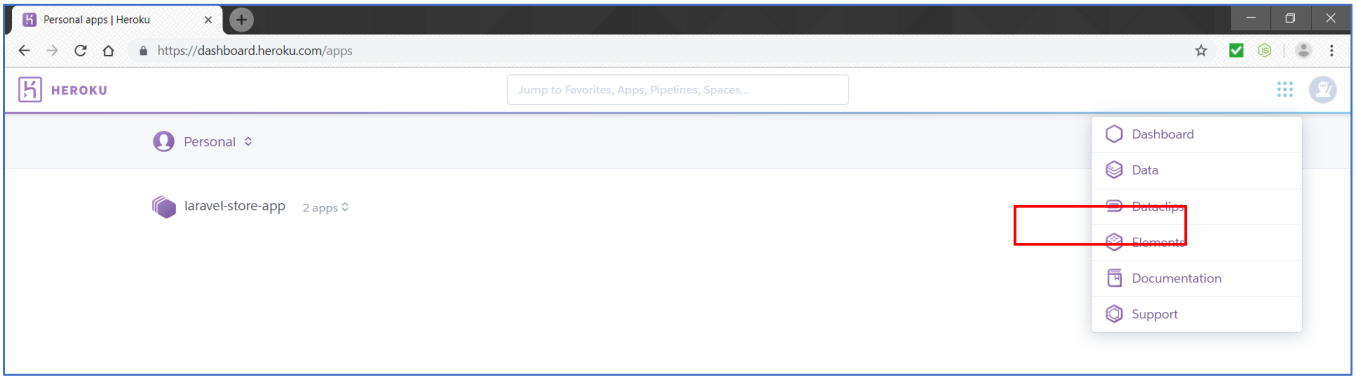


Figure 130: Heroku user dashboard to elements add-on page.

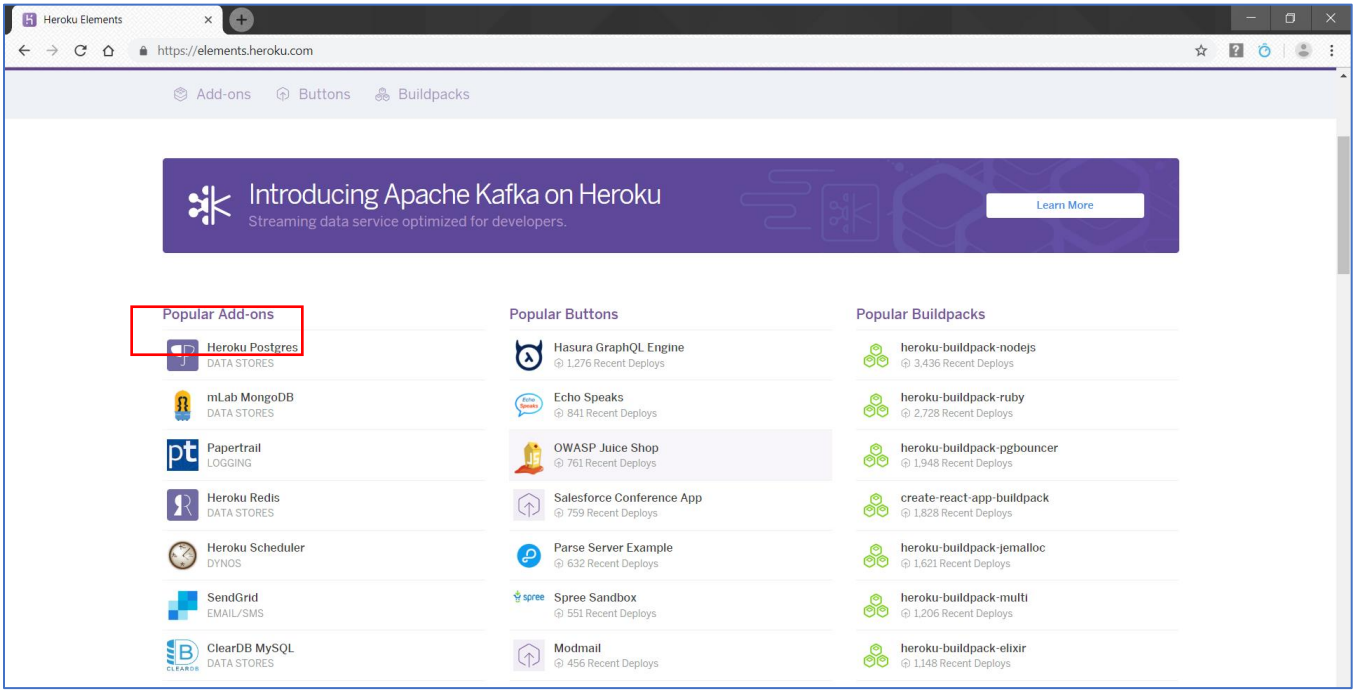


Figure 131: Heroku elements landing page.

Step 3: Clicks on Heroku Postgres add-on in Popular Add-ons section shown above in Heroku elements landing page. The browser will redirect to the element add-on page shown below.

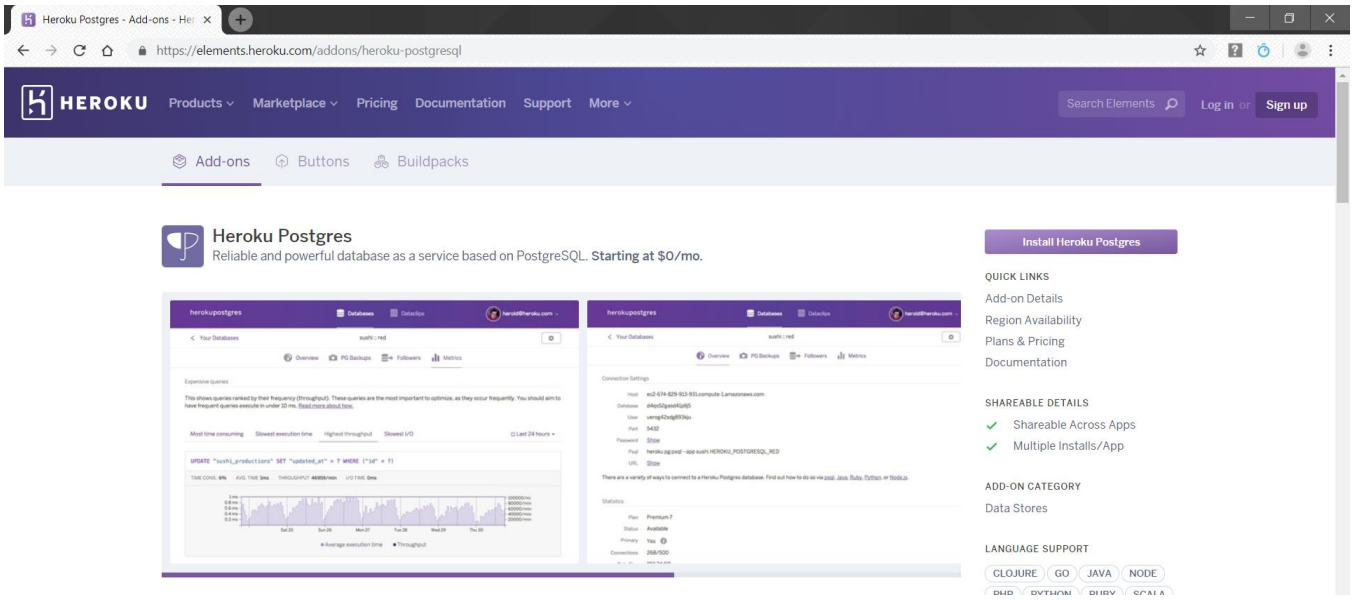


Figure 132: Heroku Postgres add-on page.

Step 4: Click on the "Install Heroku Postgres" button located on the right-hand side of the page shown in above screenshot. It will prompt for user credentials to login to Heroku if user is not logged in.

Step 5: Search ecommerce-devops application to provision to Heroku Postgres add-on and click on the "Provision add-on" button.

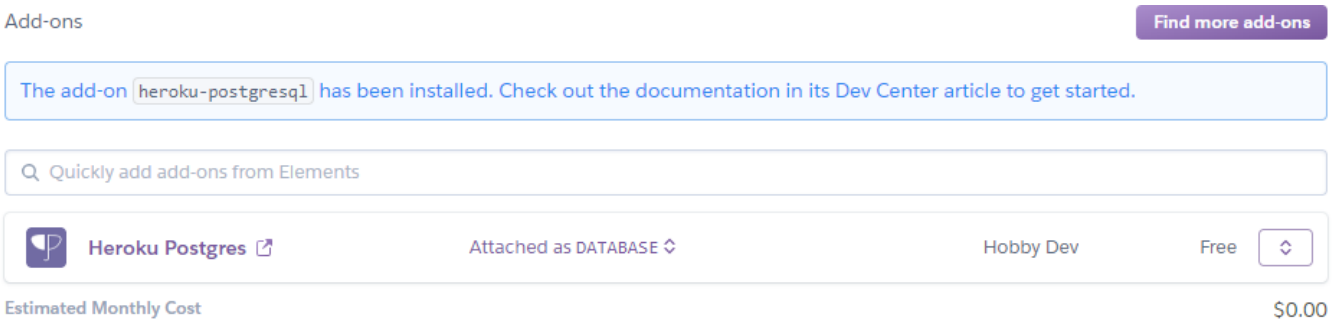


Figure 133: Add ecommerce application provision to Heroku Postgres add-on.

Step 6: Return to ecommerce-devops application Heroku dashboard landing page and click on the "Heroku Postgres" link on installed add-ons section under overview tab.

View Heroku Postgres database credentials for manual connections to this database under settings tab.

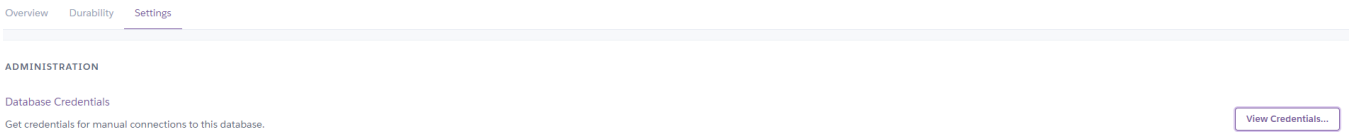


Figure 134: View Heroku Postgres database credentials.

Please note that the latest version of PostgreSQL 11.0 can be downloaded and installed from Enterprise DB. Tutorial on PostgreSQL local setup is shown in table below.

Tutorial Link:
https://devcenter.heroku.com/articles/heroku-postgresql#set-up-postgres-on-windows
https://elements.heroku.com/buildpacks/lifekent/heroku-buildpack-laravel
https://gist.github.com/calimaborges/5476c68369fafd259f2d

Step 7: Amend Laravel config/database.php file for Heroku Postgres database as shown below. Prerequisite is to add a DATABASE_URL with Heroku Postgres URL value found in Heroku environment variables under setting tab.

```php
<?php

$heroku_db_url = parse_url(env('DATABASE_URL', "postgres://forge:forge@localhost:5432/forge"));

return [
```

Figure 135: Heroku Postgres PHP variables.

```
'pg-heroku' => [
    'driver'   => 'pgsql',
    'host'     => $heroku_db_url['host'],
    'database' => substr($heroku_db_url['path'], 1),
    'username' => $heroku_db_url['user'],
    'password' => $heroku_db_url['pass'],
    'charset'  => 'utf8',
    'prefix'   => '',
    'schema'   => 'public',
],
```

Figure 136: Heroku Postgres database configuration in Laravel.

Step 8: Run heroku config:set DB_CONNECTION=pg-heroku command to configure Heroku to use the connect database.

Step 9: Run heroku run bash command to open up heroku bash command line and run php artisan migrate:refresh --seed --force to migrate tables. A full tutorial on how to host a Laravel application with database on Heroku is available on medium.com.

Please note that he /storage/*.key line in .gitignore to commit oauth public and private key to remote GitHub repository which is not recommended. The author tried several solutions to resolve non-readable private key in web server but still does not work, therefore, the author decided to remove the line from .gitignore.

Appendix F: Docker Toolbox Installation Guide

Docker Toolbox installation guide for Windows

Docker Toolbox is for older Mac and Windows system that does not meet the requirements of Docker Desktop for Mac and Docker Desktop for Windows. It is recommended to update Docker to the new application if possible. Please note that the operating system that the author used does not met the requirements of Docker Desktop for Windows, thus, Docker Toolbox is installed and configured for containerization technology.

Installation Guide:
https://docs.docker.com/toolbox/toolbox_install_windows/

Step 1: Check that the machine must have a 64-bits operating system running on Windows 7 or higher and make sure that the virtualization feature is enabled on the machine. Right-click the windows message and choose **System**.

Please do consider using Docker Desktop for Windows if machine is 64-bit Windows 10 Pro or with Enterprise and Education (1607 Anniversary update, Build 14393 or later).

Step 2: Choose **Start > Task Manager** and navigate to the **Performance** tab to look for virtualization under **CPU** section.

Please do follow the manufacturer's instructions to enable virtualization if virtualization is not enabled on the system.

Step 3: Verify Windows OS is 64-bit (x64). Further information on how to verify Windows OS is available on the article.

Step 4: Visit Docker Toolbox and click on the installer link to download the installer.

Step 5: Double-click the installer to install Docker Toolbox. The installer launches a dialog and clicks on Yes if windows security dialog prompts user to allow program to make a change.

Step 6: Click **Next** to accept all the defaults and then **Install**. It takes a few minutes to install all the Docker Toolbox components. The installer reports successful installation when the installation is complete.

Step 7: Look for **start** shell script on the **Application** folder and launch a pre-configured Docker Toolbox terminal. Click **Yes** if the system displays a **User Account Control** that prompts user to allow VirtualBox to make changes on the machine.

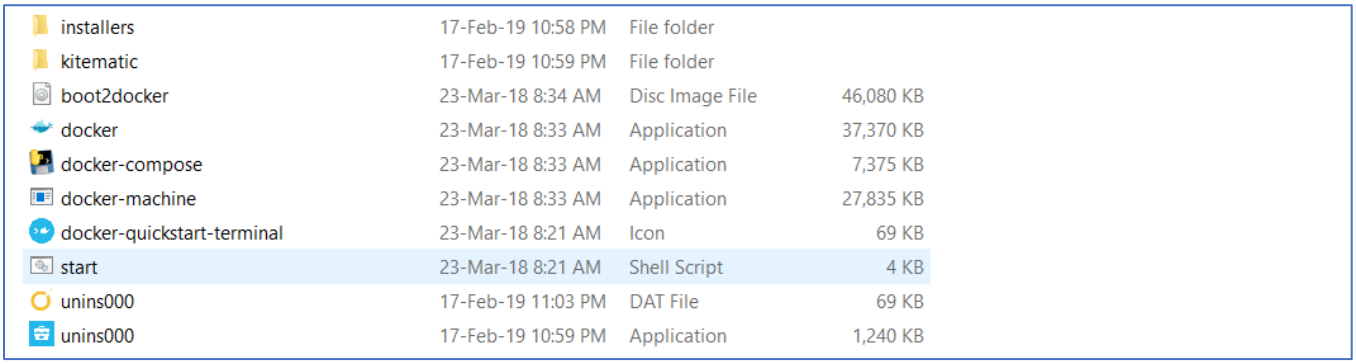


Figure 137: Docker QuickStart Terminal in Docker Toolbox Application folder.

Step 8: Docker terminal creates a public and private automatically and performs configuration on the machine. The terminal displays the **$** prompt when configuration is completed as shown below.

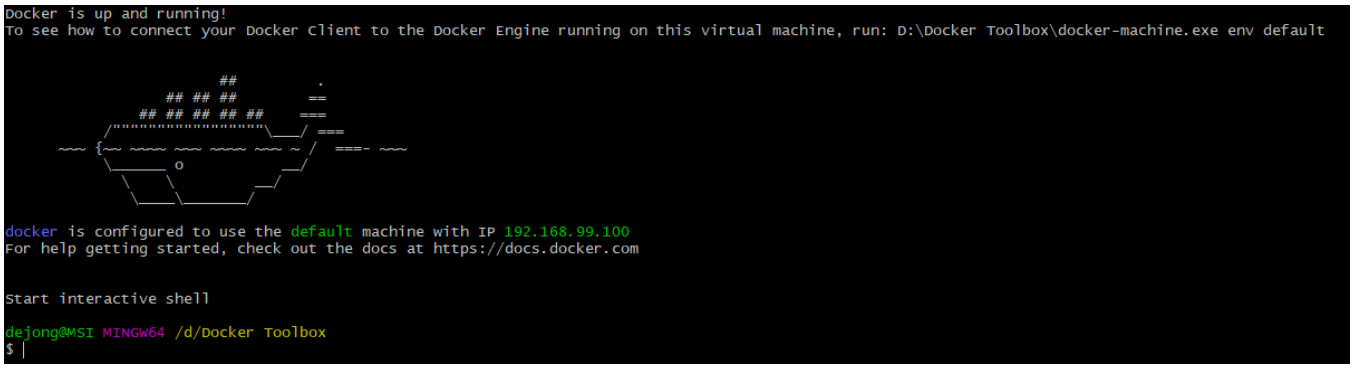


Figure 138: Docker terminal installed and configured successfully.

Appendix G: Google Cloud Command Line Installation Guide

Let's install the Google Cloud Command Line interface for containerized web applications deployment.

Installation Guide:
https://cloud.google.com/sdk/docs/quickstarts

Step 1: Visit Google Cloud SDK to install and initialize Cloud SDK on Windows.

Step 2: Click on the **Create Project** button in Google Cloud Platform to create a project in Google Cloud Platform. An ecommerce-devops project is created in Google Cloud Platform as shown in figure below.

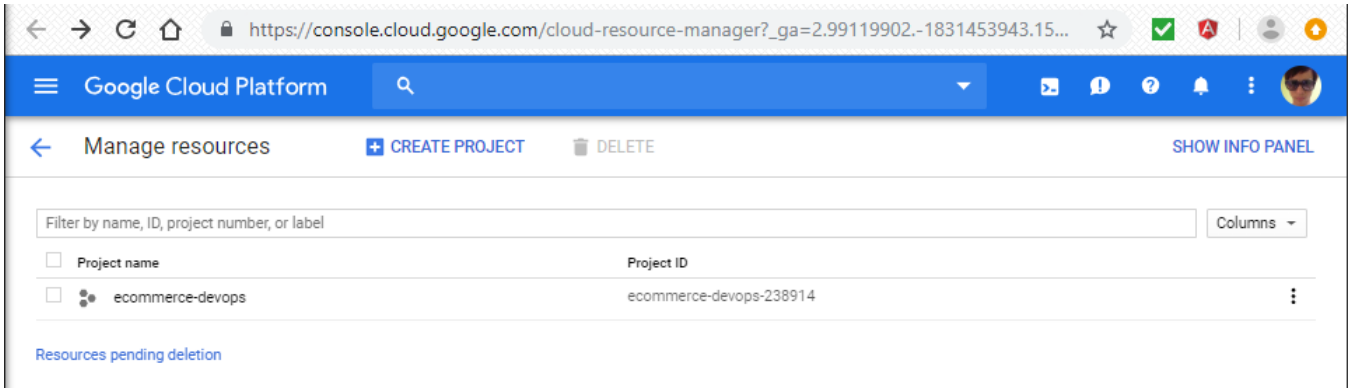


Figure 139: Create Google Cloud Platform Project.

Step 3: Download the Google Cloud SDK installer. Launch the installer and follow the prompts. Make sure that **Start Google Cloud SDK Shell** and **Run gcloud init** options are selected before the **Finish** button is clicked.

Step 4: Follow the prompts to perform several common SDK setup tasks after successful installation to initialize the SDK. Selects the created cloud project created in step 2 when prompted.

Step 5: Run gcloud components install kubectl to install Kubernetes command-line tool which is used to communicate with Kubernetes, a cluster orchestration tool of Google Kubernetes Engine clusters.

Appendix H: Comparison of Configuration Management Tool

Table below shows comparison of several development and operation tools (Virendra, 2018):

Docker **(chosen)**	• Containerization Technology. • Written in Go programming language. • Easier to manage, understand and isolate. • Delivers configuration for one process at a time. • Docker files are easier than bash script for process configuration.
Kubernetes **(chosen)**	• Configuration management tool. • Written in Go programming language. • Requires additional planning in terms of nodes and manual installation. • Suitable for modern applications development. • E.g. Google Kubernetes Engine.
Puppet	• Configuration management tool. • Written in Ruby-DSL. • Difficult for beginners. • Configure one or more process at a time and increase the complexity of the dependencies. • Puppet is more targeted towards operations, does not require development background.
Chef	• Configuration management tool. • Written in Ruby and Erlang. • Chef builds a pipeline of processes, similar to Puppet. • Complex from the development perspective.
Ansible	• Configuration management tool. • Written in Python programming language. • Easy for configuration management but is more appropriate for front-end developers. • Similar to Puppet in producing files.