



Configure Laravel 8 for CI/CD with Jenkins and GitHub — Part 1



Daniel Correa · [Follow](#)

Published in FAUN—Developer Community · 11 min read · Sep 25, 2020



143



4




This month I started to read a book called “Continuous Delivery with Docker and Jenkins — Second edition” from Rafal LESZKO. I bought it on Amazon.

I'm a professor at the University EAFIT in Colombia, and I teach a software architecture course (I have a Ph.D. in computer science). I never used Jenkins and I'm relatively new to CI/CD topics. For my software architecture course, I use the Laravel framework. So, I thought it will be good, try to replicate the elements I learned from the “Continuous Delivery with Docker and Jenkins — Second edition” book, but applied to Laravel (to get some practical knowledge in these fields).

This story is about that. I will show you some results and some code, about how I applied the lessons I learned in that book (Chapters 1 to 5 / I plan to develop a second part of this story regarding Chapters 6 to 8) but applied to the Laravel framework (the book originally shows an example in Java with the Spring boot framework).

0. Concepts introduction

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.



Sign up to discover human stories that deepen your understanding of the world.

Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

[Sign up for free](#)

Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

[Try for \\$5/month](#)

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, scalable compute capacity in the cloud.

Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud.

Unit testing is a software testing method by which individual units of source

code — sets of one or more computer program modules.

Code coverage is a metric that can help you understand how much of your source is tested.

Static code analysis is a method of debugging by examining source code before a program is run.

Acceptance Testing is to assess whether the Product is working for the user, correctly for the usage.

1. Installing Docker on AWS EC2

I use Amazon AWS EC2 to create virtual machines / EC2 instances (VMs). I created a very simple EC2 instance with the next characteristics:

- Image: Amazon Linux AMI 2018
- Instance: t2.micro
- Security groups: I opened HTTP, HTTPS ports. And I opened TCP 50000, and TCP 49001.

I connected to my EC2 instance through SSH, and then I installed Docker in my EC2 instance:

```
sudo yum update -y
sudo yum install -y docker
sudo service docker start
sudo usermod -a -G docker ec2-user
```

2. Installing Jenkins through Docker

Then, I had to create a container to deploy Jenkins. At the first time, I created a simple container with the `docker run -d -p 49001:8080 -p 50000:50000 -v $HOME/jenkins_home:/var/jenkins_home --name jenkins jenkins/jenkins` command (DO NOT use this command, this is just an example). And I started to use Jenkins.

The book mentioned you need to have a Jenkins master and some Jenkins slaves. For simplicity, I decided to have only one Jenkins container, running as master and also as slave (running jobs). In order to do that, I had to create a Dockerfile with Jenkins, but also with PHP 7.4 (in order to run Laravel), Composer, and other packages. The final Dockerfile is presented next:

```
FROM jenkins/jenkins:2.257-centos7
USER root
RUN yum install epel-release -y
RUN rpm -Uvh http://rpms.famillecollet.com/enterprise/remi-release-7.rpm
RUN yum --enablerepo=remi-php74 install php php-mbstring php-xml php-pdo php-pdo_mysql php-xdebug -y
RUN yum update -y
RUN cd /tmp
RUN curl -sS https://getcomposer.org/installer | php
RUN mv composer.phar /usr/local/bin/composer
```

Then I created a new Docker image (I called it jenkins-php), and I run a docker container:

```
docker image build -t jenkins-php .

docker run -d -p 49001:8080 -p 50000:50000 \
-v /var/run/docker.sock:/var/run/docker.sock \
--name jenkins jenkins-php
```

In order to execute some CI tasks (such as acceptance tests), I had to install Docker in Docker. So I got inside the Jenkins container, and I installed Docker inside the Jenkins container, I used the next commands:

```
docker exec -it -u root jenkins bash

yum install -y yum-utils
yum-config-manager \
--add-repo \
https://download.docker.com/linux/centos/docker-ce.repo
yum install -y docker-ce docker-ce-cli containerd.io

docker --version
docker run hello-world

chmod 666 /var/run/docker.sock
exit
```

Then I used the next line, to catch the Jenkins initial password (in order to configure Jenkins from the browser)

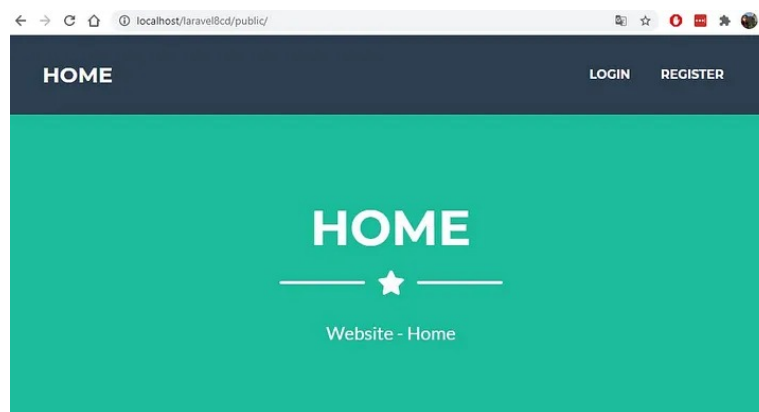
```
docker exec jenkins cat
/var/jenkins_home/secrets/initialAdminPassword
```

Then I opened my AWS EC2 instance from the browser (Accessing the 49001 port). Something like this: <http://EC2DNS.amazonaws.com:49001/>

I installed the recommended plugins and I installed another plugin called “GitHub integration”

3. A new Jenkins Item

Before creating a new Jenkins item, I designed a very basic Laravel 8 application, which can be found here: <https://github.com/danielgara/laravel8cd>



PORTFOLIO
Figure 1. Laravel8cd project

Then, I created a new item, I called it “laravel” and I selected the “Pipeline”

option.

I selected the “Pipeline script from SCM”, I choose “Git”, and I put my GitHub project address <https://github.com/danielgara/laravel8cd.git> — you can use that address if you want, or you can Fork my GitHub project.

That project already contains a Jenkins file with all the pipeline stages. In the next sections, I will explain most of those stages. Including what I did in order to: (i) run unit tests, (ii) run code coverage test, (iii) run static code tests, and (iv) run acceptance tests. The next figure shows what I implemented.

Pipeline laravel



Figure 2. Running a Job about a Laravel 8 project in Jenkins

4. Running unit tests

In order to automatically run unit/feature tests, I create a simple test in my Laravel project. That test can be found here: `/test/Feature/RouteTest.php`

```
<?php
namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithFaker;
use Tests\TestCase;

class RouteTest extends TestCase
{
    public function testHome()
    {
        $this->get('/')
            ->assertSee('Home');
    }

    public function testLogin()
    {
        $this->get('/login')
            ->assertSee('Remember Me');
    }
}
```

That test verifies that the “/” route displays a “Home” text. And the “/login” route displays a “Remember me” text. These are very simple tests; here you can apply some Class tests, model Tests, controllers, etc, and so on.

Then I developed an early version of the Jenkins file to run the unit tests:

```

pipeline {
  agent any
  stages {
    stage("Build") {
      steps {
        sh 'php --version'
        sh 'composer install'
        sh 'composer --version'
        sh 'cp .env.example .env'
        sh 'php artisan key:generate'
      }
    }
    stage("Unit test") {
      steps {
        sh 'php artisan test'
      }
    }
  }
}

```

The previous code builds the project and runs the “PHP artisan test” command which executes the unit tests (that command executes PHPUnit /a PHP framework for unit testing / which is already included in Laravel projects). More info here: <https://phpunit.de/>

I build the Laravel item (from Jenkins), and it shows me a message saying that it passed the “Declarative: Checkout SCM, Build, Unit test” stages. Job marked everything in green.

5. Improved version with Jenkins credentials

The previous pipeline didn’t change the .env variables (database username, password, host). So I used the Jenkins credentials (from the Jenkins admin panel), and I created four credentials. More info here: <https://www.jenkins.io/doc/book/using/using-credentials/>

Notice: for simplicity, I implemented the project database in Amazon RDS. More info here:

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html

The update Jenkins pipeline is presented:

```

pipeline {
  agent any
  stages {
    stage("Build") {
      environment {
        DB_HOST = credentials("laravel-host")
        DB_DATABASE = credentials("laravel-database")
        DB_USERNAME = credentials("laravel-user")
        DB_PASSWORD = credentials("laravel-password")
      }
      steps {
        sh 'php --version'
        sh 'composer install'
        sh 'composer --version'
        sh 'cp .env.example .env'
        sh 'echo DB_HOST=${DB_HOST} >> .env'
        sh 'echo DB_USERNAME=${DB_USERNAME} >> .env'
        sh 'echo DB_DATABASE=${DB_DATABASE} >> .env'
        sh 'echo DB_PASSWORD=${DB_PASSWORD} >> .env'
        sh 'php artisan key:generate'
        sh 'cp .env .env.testing'
        sh 'php artisan migrate'
      }
    }
    stage("Unit test") {
      steps {
        sh 'php artisan test'
      }
    }
  }
}

```

6. Running code coverage test

Running the code coverage test was very simple, PHPUnit already supports code coverage test. I updated the Jenkins pipeline and I included a new stage:

```
stage("Code coverage") {
    steps {
        sh "vendor/bin/phpunit --coverage-html
'reports/coverage'"
    }
}
```

I build the Laravel item (from Jenkins), and it shows me a message saying that it passed the “Declarative: Checkout SCM, Build, Unit test, Code coverage” stages. Job marked everything in green.

The code coverage report was found in a route like this:

<http://EC2DNS.compute-1.amazonaws.com:49001/job/Laravel/21/execution/node/3/ws/reports/coverage/dashboard.html> (you can navigate from your job workspace). That report showed some classes with no tests (0%), and other with (80%, 50% covered). I didn't implement a minimum of code coverage restriction, but it is a good idea to define a minimum.

7. Static code analysis

In order to run static code analysis I installed two packages in my Laravel project:

- LaraStan <https://github.com/nunomaduro/larastan>
- PHP CodeSniffer https://github.com/squizlabs/PHP_CodeSniffer

For LaraStan I created a phpstan.neon file in the project root folder with the next content

```
includes:
- ./vendor/nunomaduro/larastan/extension.neon

parameters:

paths:
- app/Http/Controllers/

level: 6

checkMissingIterableValueType: false
```

Basically, it checks all my controller files with some predefined rules. More info here: <https://github.com/nunomaduro/larastan> — those rules included to check valid comments before each method, some brackets spaces, and many more.

I also installed PHP CodeSniffer, and I modified the phpcs.xml file in the project root folder with the next content:

```
<?xml version="1.0"?>
<ruleset name="PSR2">
<description>The PSR2 coding standard.</description>
<rule ref="PSR2"/>
<file>app/Http/Controllers/</file>
<exclude-pattern>vendor</exclude-pattern>
<exclude-pattern>resources</exclude-pattern>
<exclude-pattern>database</exclude-pattern>
<exclude-pattern>storage</exclude-pattern>
<exclude-pattern>node_modules</exclude-pattern>
</ruleset>
```

I only wanted to check the controllers (but obviously you can apply it to models, and many other files). It uses a PSR2 coding standard that check camelCase in the method names, some spaces, and many more. More info here: <https://www.php-fig.org/psr/psr-2/>

Finally, I update the Jenkins pipeline and I included two new stages:

```
stage("Static code analysis larastan") {
    steps {
        sh "vendor/bin/phpstan analyse --memory-limit=2G"
    }
}
stage("Static code analysis phpcs") {
    steps {
        sh "vendor/bin/phpcs"
    }
}
```

I build the Laravel item (from Jenkins), and it shows me a message saying that it passed the “Declarative: Checkout SCM, Build, Unit test, Code coverage, Static code analysis larastan, Static code analysis phpcs” stages. Job marked everything in green.

8. Docker inside Docker (running a container of the Laravel project)

Docker in Docker was challenging, I faced many issues but in the end I did it. After the static code tests, I had to implement the “Acceptance tests”. The acceptance test requires a staging version of the application running. And then, you run some tests over that running version.

It means, you need to execute some commands, to build a new docker image (inside the Jenkins container) based on the GitHub current code, and run a new container, and test that container.

I created a Docker Hub account, in order to put those images online. And avoid to create a Docker registry. In part 2 of this story, I already explained how to installed Docker in Docker. So, the next part was to add two new stages to the Jenkins pipeline:

```
stage("Docker build") {
    steps {
        sh "docker rmi danielgara/laravel8cdpart2"
        sh "docker build -t danielgara/laravel8cdpart2 --no-
cache ."
    }
}
stage("Docker push") {
    environment {
        DOCKER_USERNAME = credentials("docker-user")
        DOCKER_PASSWORD = credentials("docker-password")
    }
    steps {
        sh "docker login --username ${DOCKER_USERNAME} --
```

```
password ${DOCKER_PASSWORD}"
    sh "docker push danielgara/laravel8cdpart2"
  }
}
```

Those stages build a new image based on the GitHub current code, and connect with Docker Hub (through the use of two Jenkins credentials that I registered), and then, it pushes the new image to my Docker Hub account. You can find that image here:

<https://hub.docker.com/r/danielgara/laravel8cd>

Having that image online allows me to create new containers easily, and to apply the acceptance tests.

9. Acceptance tests

I run two types of acceptance tests. With CURL command, and with co-deception package.

CURL command was a first try to check if I was able to run a new container and apply some tests to that running container.

I create a very simple functionality in laravel to sum two numbers, check the next code (which was added in the `/app/Http/Controllers/HomeController.php`):

```
/**
 * Return sumatory
 *
 * @return int
 */
public function sum(int $num1, int $num2)
{
    return $num1+$num2;
}
```

If you open the browser and go to `/public/sum/2/4` it displays 6.

Then I create an `acceptance_test.sh` file to use the CURL command and to test the previous functionality. This file obtains the running container IP address and invoke the `/public/sum/4/2`. Then, it compares the result with the number "6". If both are equals, the test passed, otherwise it shows an error.

```
#!/bin/bash
res=$(curl -s -w "%{http_code}" $(docker inspect -f '{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
laravel8cd)/public/sum/4/2)
body=${res::-3}
if [ $body != "6" ]; then
    echo "Error"
fi
```

I updated the Jenkins pipeline with the next two stages:

```
stage("Deploy to staging") {
    steps {
        sh "docker run -d --rm -p 80:80 --name laravel8cd
```



```

danielgara/laravel8cd"
    }
    stage("Acceptance test curl") {
        steps {
            sleep 20
            sh "chmod +x acceptance_test.sh &&
            ./acceptance_test.sh"
        }
        post {
            always {
                sh "docker stop laravel8cd"
            }
        }
    }
}

```

It runs a container based on the update docker hub image, then it executes the acceptance_test.sh file (but before, it waits 20 seconds to allows docker to create the container successfully). At the end of the stage and always, it stops the container.

Finally, the **second acceptance test** was with the use of PHP package called PHP co-deception.

I installed PHP co-deception <https://codeception.com/for/laravel> — and I created a very simple acceptance test (which can be found on /tests/acceptance/RegisterCest.php):

```

<?php
class RegisterCest
{
    public function tryToTest(AcceptanceTester $I)
    {
        $I->amOnPage('/');
        $I->click('Register');
        $I->see('Confirm Password');
    }
}

```

This acceptance test checks that the “/” route contains a link called Register, and it clicks that link; therefore, it checks that the new loaded page contains a text called “Confirm password”.

I updated the Jenkins pipeline with the final stage:

```

stage("Acceptance test codeception") {
    steps {
        sh "vendor/bin/codecept run"
    }
    post {
        always {
            sh "docker stop laravel8cd"
        }
    }
}

```

That stage runs that co-deceptions tests, and at the end stops the container.

I build the Laravel item (from Jenkins), and it shows me a message saying that it passed the “Declarative: Checkout SCM, Build, Unit test, Code coverage, Static code analysis larastan, Static code analysis phpcs, Docker build, Docker push, Deploy to staging, Acceptance test curl, Acceptance test co-deception” stages. Job marked everything in green.

The final result is presented in Figure 2.

Summary

This story explains how to apply some CI/CD principles in Laravel projects (with the use of Jenkins and Docker). The example was very basic in order to introduce some principal concepts. But you have to consider that there are many risks with the presented example. Jenkins slaves must be separated. Not always is possible to have the database in Amazon RDS, some authors didn't recommend using Docker in Docker, and so on. However, for novice Laravel CI/CD developers (such as me), I think this is a good introduction.

Note: I plan to develop a second part of this story regarding Chapters 6 to 8 of the "Continuous Delivery with Docker and Jenkins — Second edition" book.

UPDATE: part 2 available here: <https://medium.com/@danielgara/configure-laravel-8-for-ci-cd-with-jenkins-and-github-part-2-80d9c337f0e4>

👉 **Join FAUN today and receive similar stories each week in your inbox!** Get your weekly dose of the must-read tech stories, news, and tutorials.

Follow us on [Twitter](#) 🐦 and [Facebook](#) 🌐 and [Instagram](#) 📷 and join our [Facebook](#) and [LinkedIn](#) Groups 💬

If this post was helpful, please click the clap 👏 button below a few times to show your support for the author! ↓

Laravel

Jenkins

Continuous Integration

PHP

Docker



Published in FAUN—Developer Community 🍷🍷

19.98K Followers · Last published Dec 17, 2024

We help developers learn and grow by keeping them up with what matters. 📧
www.faun.dev

Follow



Written by Daniel Correa

74 Followers · 1 Following

Ph.D. in Computer Science. Author of 'Practical Laravel', 'Practical Nest.js', 'Beginning Django 4' and more books. Follow me on Twitter at @danielgarax

Follow

Responses (4)



What are your thoughts?

Respond



Daniel Correa Author
over 4 years ago



Now you can find "Part 2" of this story here: <https://medium.com/@danielgara/configure-laravel-8-for-ci-cd-with-jenkins-and-github-part-2-80d9c337f0e4>



Reply



Pooya Sabramooz
almost 4 years ago



Good job professor, can i test it using share hosting?



1



1 reply

Reply



Ahmed El Assimi
almost 4 years ago



thank you, it was very helpful



1

Reply

See all responses

More from Daniel Correa and FAUN—Developer Community 🌸



In FAUN—Developer Communit... by Daniel Corr...

Configure Laravel 8 for CI/CD with Jenkins and GitHub—Part 2

This is the second part of the "Configure Laravel 8 for CI/CD with Jenkins and GitHub..."

Oct 4, 2020



64



In FAUN—Developer Community 🌸 by Ali Hamza

5 Cloud Certifications That Can Skyrocket Your Career in 2025...

This guide will explain 5 cloud certifications that are highly valued in 2025 and provide...

Nov 25, 2024



425



13



In FAUN—Developer Communit... by Akanksha Bh...

12 Tools that will make Kubernetes management easier in 2024



Daniel Correa

Designing a knowledge tree for self-learning—learning fast in a...

Kubernetes, the revolutionary container orchestration platform, has empowered...

Sep 24, 2024 156 2



I'm a professor of a computer science program in Colombia. Everyone who works i...

Oct 20, 2020 15 2



See all from Daniel Correa

See all from FAUN—Developer Community

Recommended from Medium

Achyut Neupane

Are cPannels enough for your basic deployments?

You are looking for solutions for your deployment but cannot decide if cPanel is...

Dec 6, 2024 31



Adnan Turgay Aydin

Building a Real-Time CI/CD Pipeline: Step-by-Step Guide for...

This DevOps project is designed to enhance your resume and provide hands-on...

Aug 16, 2024 24



Lists

Coding & Development

11 stories · 959 saves

Natural Language Processing

1882 stories · 1520 saves

In Django Unleashed by Joel Wembo

Technical Guide: End-to-End CI/CD DevOps with Jenkins, Docker,...

Building an end-to-end CI/CD pipeline for Django applications using Jenkins, Docker,...

Apr 13, 2024 1K 21



Saqib Khan

NGINX - Zero To Hero: Your Ultimate Guide from Beginner to...

Introduction to Nginx

Aug 4, 2024 941 11



Anand Panchal

Harendra

Managing Multiple PHP Versions
Using Docker: A Simple Guide for...

Have you ever found yourself juggling multiple PHP projects, each requiring a...

★ Dec 24, 2024 👤 3 📖+

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

★ Oct 26, 2024 👤 8.5K 💬 134 📖+

See more recommendations