**Table of Contents**

[Testing 2](#__RefHeading___Toc401_248654156)

[Running Unit Test 2](#__RefHeading___Toc403_248654156)

[Symfony Functional Tests 2](#__RefHeading___Toc405_248654156)

[Running All Tests 2](#__RefHeading___Toc407_248654156)

[Running Tests with Coverage 2](#__RefHeading___Toc409_248654156)

[Troubleshooting PhpUnit Testing 2](#__RefHeading___Toc447_248654156)

[Hosting 4](#__RefHeading___Toc592_248654156)

[Prerequisites 4](#__RefHeading___Toc413_248654156)

[Steps to Host Symfony Project 4](#__RefHeading___Toc415_248654156)

[Additional Considerations 6](#__RefHeading___Toc431_248654156)

[Jenkins CI CD 7](#__RefHeading___Toc811_248654156)

[Prerequisites 7](#__RefHeading___Toc435_248654156)

[Jenkins Pipeline Configuration 7](#__RefHeading___Toc437_248654156)

[Save and Run the Pipeline 9](#__RefHeading___Toc453_248654156)

[Jenkins Job Configuration: 9](#__RefHeading___Toc443_248654156)

# Testing

To run PHPUnit tests in Symfony, you use the php bin/phpunit command. Symfony includes a phpunit.xml.dist configuration file that PHPUnit uses to execute tests. Here's a step-by-step guide:

## Running Unit Test

Assuming your tests are in the tests/Entity/QuestionTest.php file, run the following command in your terminal:

$ php bin/phpunit tests/Entity/QuestionTest.php

This command executes the specified PHPUnit test file.

## Symfony Functional Tests

Assuming your functional tests are in the tests/Controller/QuestionControllerTest.php file, run the following command:

$ php bin/phpunit tests/Controller/QuestionControllerTest.php

This command executes the specified PHPUnit functional test file.

## Running All Tests

To run all tests in your Symfony application, simply run:

$ php bin/phpunit

This command executes all tests according to the configuration in the phpunit.xml.dist file.

## Running Tests with Coverage

If you want to generate a code coverage report, you can use the --coverage-html option. For example:

$ php bin/phpunit --coverage-html=var/coverage

This command generates a code coverage report in the var/coverage directory. Open the generated HTML file in your browser to view the coverage report.

$ composer require --dev phpunit/phpunit

These commands assume that you are in the root directory of your Symfony project. Adjust paths as needed based on your project structure.

## Troubleshooting PhpUnit Testing

* Database not found

$ php bin/console doctrine:database:create –env=test

$ php bin/console doctrine:schema:update --env=test –force

$ php bin/console doctrine:migrations:migrate –env=test

* APCu is not enabled

Error:

App\Tests\Controller\RegistrationControllerTest::testRegisterPageIsSuccessful

Symfony\Component\Cache\Exception\CacheException: APCu is not enabled.

Fix this error by running the command below:

$ sudo apt-get install php-apcu

# Hosting

Hosting a Symfony project on a server involves several steps, including configuring the server environment, deploying the Symfony application, and setting up a web server. Below is a basic guide to help you get started. Note that the specific steps may vary based on your server environment and requirements.

## Prerequisites

1. **Server**
   * You need a server where you can deploy your Symfony application. This could be a virtual private server (VPS), a cloud server (e.g., AWS, DigitalOcean), or a dedicated server.
2. **Web Server**
   * A web server (e.g., Apache, Nginx) should be installed and configured on the server.
3. **PHP**
   * PHP should be installed on the server. Ensure that the PHP version meets the Symfony project requirements.
4. **Database**
   * If your Symfony application uses a database, ensure that the required database server (e.g., MySQL, PostgreSQL) is installed and configured.

## Steps to Host Symfony Project

1. **Transfer Project Files**

Copy your Symfony project files to the server. This can be done using various methods, such as SCP, SFTP, Git, or by using deployment tools.

2. **Install Dependencies**

1. SSH into the server and navigate to the Symfony project directory.
2. Run the following command to install Composer dependencies:

$ composer install --no-dev –optimize-autoloader

3. **Set Permissions**

Set appropriate file permissions and ownership for Symfony cache and log directories:

$ chmod -R 775 var/cache var/log

$ chown -R www-data:www-data var/cache var/log

Replace www-data:www-data with the appropriate web server user and group.

4. **Configure Database**

If your Symfony project uses a database, update the database configuration in the parameters.yml or DATABASE\_URL environment variable.

5. **Configure Web Server**

Configure your web server (Apache or Nginx) to serve the Symfony application. Below are basic configurations:

**Apache**

<VirtualHost \*:80>

ServerName your-domain.com

DocumentRoot /path/to/symfony/public

<Directory /path/to/symfony/public>

AllowOverride None

Order Allow,Deny

Allow from All

FallbackResource /index.php

</Directory>

<FilesMatch \.php$>

SetHandler "proxy:unix:/var/run/php/php7.4-fpm.sock|fcgi://localhost/"

</FilesMatch>

</VirtualHost>

**Nginx**

server {

listen 80;

server\_name your-domain.com;

root /path/to/symfony/public;

location / {

try\_files $uri /index.php$is\_args$args;

}

location ~ ^/index\.php(/|$) {

include snippets/fastcgi-php.conf;

fastcgi\_pass unix:/var/run/php/php7.4-fpm.sock;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

include fastcgi\_params;

}

error\_log /var/log/nginx/symfony\_error.log;

access\_log /var/log/nginx/symfony\_access.log;

}

Adjust the configurations based on your Symfony project directory and PHP version.

**6. Restart Web Server**

Restart the web server to apply the changes.

**Apache**

$ service apache2 restart

**Nginx**

service nginx restart

**7. Test**

* Visit your domain in a web browser to test your Symfony application.

## Additional Considerations

* **SSL/TLS Configuration:** Consider configuring SSL/TLS for secure communication using a Let's Encrypt certificate or a commercial certificate.
* **Environment Configuration:** Ensure that Symfony is running in the correct environment (prod for production) by setting the APP\_ENV environment variable.
* **Logging and Monitoring:** Set up logging and monitoring to track errors and performance issues.
* **Backup and Recovery:** Implement regular backup strategies to protect your application data.

This guide provides a basic overview, and the actual steps may vary based on your server environment and specific project requirements.

# Jenkins CI CD

Setting up a Jenkins CI/CD pipeline for a Symfony project involves configuring Jenkins jobs to automate tasks such as building, testing, and deploying your application. Below is a general guide to help you set up a basic Jenkins pipeline for a Symfony project. Adjust the steps according to your specific project structure and requirements.

## Prerequisites

1. Jenkins server installed and running.
2. Git repository for your Symfony project.
3. Composer installed on the Jenkins server.

## Jenkins Pipeline Configuration

1. **Install Jenkins Plugins**

Make sure the following Jenkins plugins are installed:

* + Git Plugin
  + Pipeline
  + PHP Composer

1. **Create a New Jenkins Pipeline Job**
   * Open Jenkins and click on "New Item" to create a new pipeline job.
   * Enter a name for the job (e.g., SymfonyPipeline) and select "Pipeline" as the job type.
2. **Configure Source Code Management**
   * In the "Pipeline" section, choose "Pipeline script from SCM."
   * Select "Git" as the SCM.
   * Enter your Git repository URL.
3. **Building and Deploying**
   * **Define Jenkinsfile**

Create a Jenkinsfile (uses groovy) at the root of your Symfony project. This file defines the steps for your Jenkins pipeline. Below is a simple example:

pipeline {

agent any

environment {

// Define environment variables as needed

SSH\_KEY = credentials('your-ssh-key-credential-id')

SSH\_USER = 'your-ssh-username'

SSH\_HOST = 'your-ssh-host'

DEPLOY\_PATH = '/path/to/deploy'

}

stages {

stage('Checkout') {

steps {

checkout scm

}

}

stage('Install Dependencies') {

steps {

sh 'composer install --no-dev --optimize-autoloader'

}

}

stage('Run Tests') {

steps {

sh 'php bin/phpunit'

}

}

stage('Build and Deploy') {

steps {

script {

// Clear cache and warmup for production

sh 'php bin/console cache:clear --env=prod --no-warmup'

sh 'php bin/console cache:warmup --env=prod'

// Deploy to server via SSH

sshagent(['your-ssh-key-credential-id']) {

sh "scp -r -i ${SSH\_KEY} . ${SSH\_USER}@${SSH\_HOST}:${DEPLOY\_PATH}"

}

}

}

}

}

}

Explanation:

1. **Environment Variables**
   * SSH\_KEY: The ID of the Jenkins credential containing the SSH private key.
   * SSH\_USER: The SSH username for deployment.
   * SSH\_HOST: The SSH host for deployment.
   * DEPLOY\_PATH: The path on the server where the project will be deployed.
2. **Build and Deploy Stage**
   * The cache is cleared and warmed up for the production environment.
   * The project is deployed to the server via SSH using the scp command.

**Additional Considerations**

1. **Notification:** Consider adding notifications (email, Slack, etc.) to alert developers or teams about the build and deployment status.
2. **Security:** Ensure that your Jenkins server and pipeline are secure. Protect sensitive information, use credentials securely, and restrict access as needed.

## ****Save and Run the Pipeline****

1. Save the Jenkins job configuration.
2. Click on "Build Now" to run the pipeline manually.

This is a basic setup, and you may need to customize it based on your project's complexity and requirements. Consult the Jenkins documentation for more advanced features and options.

## Jenkins Job Configuration:

1. **SSH Key Credential:**
   * In Jenkins, go to "Credentials" > "System" > "Global credentials" and add a new SSH username with a private key credential.
   * Use the credential ID in the Jenkinsfile ('your-ssh-key-credential-id').
2. **Jenkins Job Configuration:**
   * Make sure the Jenkins job has the necessary permissions to access Git and the SSH key credential.
   * Set the Jenkinsfile path in the job configuration.
3. **Configure Deployment Server:**
   * Ensure that the deployment server is configured to receive the project via SSH and has the required dependencies (e.g., PHP, web server) installed.

This is a basic example, and you may need to adjust the deployment steps based on your server environment and deployment process. Make sure to secure sensitive information, such as SSH keys and credentials, and consider using more advanced deployment strategies if needed.