# How To Configure CI/CD on MedStack


SiteRocket® Labs

SiteRocket® Labs

MEDSTACK®

# How to Configure CI/CD on MedStack

## What is CI/CD? Why use it?

**Continuous integration and continuous delivery** (CI/CD) is a set of tools and practices that automate the process of building, testing and deploying software updates on an ongoing basis.

In the early days of the internet, many developers would often just copy their latest code onto the server whenever they wanted to update a web application. This might have been easy and convenient – but pasting code into a live application is like performing brain surgery on a patient who's awake and conscious. (Meaning, it's not very safe!)

Many things can go wrong: Files can be corrupted in transit, redundant files might fail to be deleted from the server (posing security risks), the update might need to coincide with database changes, the developer might spill coffee on their keyboard while uploading their files, etc.

Enter CI/CD: An automated way of deploying frequent updates to live applications in a manner that's secure, reliable and poses minimal risk. While teams aren't required to set up an automated CI/CD pipeline, we do recommend it, and it's never been easier to do so using the MedStack API.

CI/CD pipelines also reinforce good software development practices such as protecting production secrets and controlling exactly which service container images run in production, two practices that are especially important for health and medical apps that handle sensitive data.

## Getting started

Before rolling out CI/CD in your organization, a number of software development best practices need to be in place. Here is a brief list.

### Culture of ongoing enhancements

In order for CI/CD to succeed, *everyone* involved in the evolution of your application – from your company's executives and product managers to the UI designers and developers – needs to buy into the CI/CD philosophy. In a nutshell, all stakeholders must be willing and able to continually translate major product goals into a series of small, ongoing enhancements that can be rolled out and supported incrementally. Needless to say, if your organization can't get away from performing one very large upgrade every few months, you may not realize the true benefits of CI/CD as it relates to deployment speed and autonomy.

### Staging environment

You should have a separate staging environment up and running where you can test the latest branches and versions of your applications before they make it into production. Many consulting companies also make use of an additional "UAT" (user acceptance testing) server on which their clients can perform their tests. Regardless, at least two environments (staging and production) will be needed.

### Well-defined version control and code branching strategy

It's critical to use code branching and merging strategies consistently across your organization. Most development teams use "feature flags" or "version control branching" techniques to define the parts of the code that need deployment. (For the purposes of this guide, we'll assume that your code resides in separate branches which mirror the code that's deployed to your staging, production, and sometimes UAT environments.)

**Security**

When dealing with protected health information (PHI), security best practices need to be well-ingrained into your operations.

For example, using a company-wide password manager is recommended – and it's good practice to have separate password vaults for production and staging credentials. (The production vault should be shared with highly trusted developers and devops staff only.)

Production deployments should only be handled by senior and trusted developers – and we will show you later how to configure this.

**Ability to deploy database schema changes**

When you need to launch a new update that requires a change to the database, you'll need to be able to automate this change. A common way to do so is to use a SQL script that can be run during the deployment and can perform the necessary changes. Many development frameworks include a mechanism to do this.

**Automated unit tests**

Automated tests are important because they provide an easy way to ensure that parts of the system work as expected.

While relatively few companies have full 100% test coverage for their code, it's good to automate at least some basic "sanity checks" to ensure that major parts of the system will work after being deployed. This way, deployments can automatically stop if a test has failed.

# How to Configure CI/CD

Now that we've covered the "soft skills" and practices that are prerequisites for a successful CI/CD implementation, we'll provide the technical details on how to configure CI/CD on MedStack Control. We'll also provide a sample CI (continuous integration) script that you can tailor to your needs.

## Overview

There are many ways to configure a CI/CD pipeline. In this example, our CI/CD pipeline works with our source control, Slack, and MedStack Control to perform a series of steps that control and inform our team on the release of new deployments.

The CI tool first springs into action when it detects a code merge to the /stage or /master branch in your code repository. It then performs the following tasks:

1. Notifies you via Slack that the process has been initiated
2. Creates a new Docker container for the release
3. Downloads the required applications (nginx, MariaDB, etc.) and installs them in the Docker container
4. Clones the code from the Git branch into the Docker container
5. Runs some automated unit tests if required
6. Sends the built Docker image to the registry
7. Instructs MedStack Control via its API to pull the Docker image (among other things)
8. Notifies you via Slack that the deployment has concluded

Just like magic!

## Configure MedStack Control and other tools

Here's how to configure your CI/CD tools for the pipeline described above.

- Assign permissions to your code branches in order to allow a limited number of trusted engineers to merge to specific branches such as /stage or /master, and deploy to your various servers. If you use GitHub, you can configure it as documented here.

- Generate a token in MedStack Control, which will be used to access the MedStack Control API, allowing you to control the Docker resources in your clusters. Ensure that you have created a cluster with all of the nodes/services that you need. MedStack Control runs Docker in a swarm mode: according to Docker's documentation, it's better to run at least a Manager node and a Worker node (both with service replicas) so you minimize service interruptions on deployments. We encourage you to test when using replicas as there might be some odd behaviors (like cron jobs being executed multiple times). If such a case occurs you should try to split tasks to two or more services to avoid that.

- You will need a place to store container images after they have been built by the CI tool and before they are deployed to Docker on MedStack Control. GitLab provides a significant amount of free storage space and might be a good fit for this. Docker Hub, on the other hand, provides free container image scanning with Snyk on their Pro and higher subscriptions. You can host your container images on whatever registry works best for your team's needs.

- If your application contains an "environment file" that stores database credentials and application parameters, we encourage you to store these parameters in Secrets and Environment Variables instead. Environment Variables are set at runtime with global scopes in a service instance (containers). Secrets are data in Docker that write to an encrypted file on disk that can be used by select services. Only containers that have been configured to use select Docker Secrets have the ability to decrypt its contents. All other containers will perceive the contents of Secrets files as encrypted.

- We advise against environment files because they tend to not be encrypted, and if they were, decryption would need to be handled in the application layer rather than in the orchestration layer which MedStack Control manages.

- Once you've completed the above steps, it's time to create the "CI script" which is what the CI tool will run when performing the continuous integration workflow. Create the script and store it in your code repository. If you use Travis CI, you must call it .travis.yml. Just make it blank for now; we'll give you a sample script in the next section.

**Create your CI script**

Now it's time to create the script that your CI tool will run. Popular CI tools include TravisCI, CircleCI and others. We have created a sample script that runs on Travis CI and works beautifully with MedStack Control. This script may be found here: https://github.com/siterocketlabs/MedStack_CICD.

The script makes use of the following four files, which you'll want to copy into your code repository and modify for your needs. We'll go through each of these files and explain what they do.

```
.travis.yml
Dockerfile
Api.sh
index.php
```

If you use a different CI tool (such as CircleCI) with MedStack Control and are willing to share your script with us, please get in touch with us at support@medstack.co. The more samples we can help our users with, the better. If we use your script, we'll even send you a free T-shirt!

## Build the Server Image

First it's time to build your server image. To do this, we'll start with .travis.yml and will go through this script section by section to explain how it works. (Note that the variables that start with $TRAVIS_ are predefined by Travis CI; some are OS environment variables, like $HOME; while others are custom defined by users like you.)

First, this script gets Travis CI to install a specific development language and database. In this case, we're specifying that this application uses PHP v8.0 and MySQL.

```
# File: .travis.yml
# programming language
language: php

# list any PHP versions you want to test against
php:
  - 8.0

# install mysql to run phpunit tests
services:
  - mysql
```

Next, we specify that Travis CI should run if you push into master or stage code branches. Of course, if you have a UAT branch, add it here too.

```
# build only on specific branches
branches:
  only:
    - master
    - stage
```

### Update Slack

If your company uses Slack, you might want Travis CI to update Slack when the deploy process is about to start. That's what the curl command does, below. (Be sure to replace $YOURSUBDOMAIN, $TOKEN and $CHANNEL with your Slack subdomain, Token and Channel. These appear further down in the script as well.) Disabling xdebug improves performance.

```
before_install:
  - curl -s --data "Starting build job on $TRAVIS_BRANCH branch."
'https://$YOURSUBDOMAIN.slack.com/services/hooks/slackbot?token=$TOKEN&channel=$CHANNEL'
  - export PATH=$HOME/.local/bin:$PATH
  - phpenv config-rm xdebug.ini
```

It's a good idea to configure your CI/CD pipeline to run automated unit tests; this way the deployment will stop if any test fails. Even if you don't have full test coverage of your code, it's a good idea to add at least a few automated "sanity tests" to be safe. A good place to initiate these unit tests is in the script section of the file, which is shown below. You might also need to add other scripts here too, to do things like copy or rename files, for example.

```
# add script(s) here, such as unit tests
script:
# [insert desired script(s) here]
```

### Push Image to the Registry

Now it's time for Travis CI to build and push the server image to your registry of choice (GitLab in this case). In the next section, we'll instruct the MedStack Control API to pull and run these images. You'll need to add a registry (username/password to log into your registry of choice to pull images) so that MedStack can use them on every pull.

To do this, we need to set five Travis Secrets, and these are:

- URL with the URL of MedStack's API
- AUTH with your MedStack API token
- SECRET with the secret string you want to create
- DOCKER_USERNAME and DOCKER_PASSWORD (registry credentials)

You'll notice that our script calls **Dockerfile** which is one of the files in our code repository. This file provides instructions to build the Docker image – but we'll ignore that for now. (We will describe it later in this document.)

```
# prepare the build for deployment
before_deploy:
  - docker build -f Dockerfile --build-arg URL=$URL --build-arg AUTH=$AUTH --build-arg
SECRET=$SECRET -t registry.gitlab.com/siterocket/medstacktest/apachephp:$TRAVIS_BRANCH .
  - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
registry.gitlab.com
  - docker push registry.gitlab.com/siterocket/medstacktest/apachephp:$TRAVIS_BRANCH
```

### Deploy the Image to MedStack Control

Now we call the MedStack Control API to instruct it to fetch and deploy the new image; this is the first curl statement below. Then we notify your desired Slack channel that the deployment is about to begin; hence the second curl. Finally, we add skip_cleanup in order to prevent Travis CI from resetting your working directory and deleting all changes made during the build. (The parameter all_branches is needed because the branch name is not known ahead of time.)

```
# deploy
deploy:
  - provider: script
    script:
      curl -X POST -H "Accept:application/json" -H "Authorization:Basic $AUTH"
"$URL/$(docker run --rm registry.gitlab.com/siterocket/medstacktest/apachephp:$TRAVIS_BRANCH
echo \$company_id/clusters/\$cluster_id/services/\$service_id)/refresh_image"
      curl -s --data "Deploying APP for $TRAVIS_BRANCH branch."
'https://$YOURSUBDOMAIN.slack.com/services/hooks/slackbot?token=$TOKEN&channel=$CHANNEL'
    skip_cleanup: true
    on:
      all_branches: true
```

Now let's look at the file named **Dockerfile**, which lets us build the Docker container with all the necessary services in it. This file contains all the commands necessary to install the packages that we require (like PHP 8.0 and Apache) in order to deploy an index.php file with a congratulatory "hello world"-type message in it.

```
# File: Dockerfile
# we use ubuntu 20.04
FROM ubuntu:20.04

# update packages, add PHP 8 repository and install jq and curl
RUN apt update && apt -y install ca-certificates apt-transport-https
software-properties-common && add-apt-repository ppa:ondrej/php && apt -y install jq curl
php8.0 libapache2-mod-php8.0

# copy files
COPY ./index.php /var/www/html/index.php
COPY ./api.sh /root/api.sh

# make api.sh executable and rm Apache's default index.html
RUN chmod +x /root/api.sh
RUN rm -f /var/www/html/index.html

# assign Travis secrets to variables so we can pass them as parameters
ARG URL=${URL}
ARG AUTH=${AUTH}
ARG SECRET=${SECRET}

# call MedStack's API and create a secret (and pass it 3 parameters)
RUN /root/api.sh $URL $AUTH $SECRET

# set the secret on env on the build
ENV THESECRET=${SECRET}
```

```
# start apache
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

You'll notice that the above script references api.sh which is the third file in our code repository. This file contains all the necessary MedStack Control API calls to do the following things:

1. Get your company ID, cluster ID and service ID in MedStack Control. The script will request changes to the specific cluster's service.
2. Create a Secret in that cluster
3. Attach the Secret to the service

As you can see above, the Dockerfile passes three parameters to api.sh, which it uses when calling the API. The api.sh file refers to these parameters as $1, $2 and $3.

Below are the contents of api.sh. (This file uses the jq utility, which is a command-line JSON processor that assists it in querying the API and parsing its results.)

```
# File: api.sh
# get company
company_id=$(curl -X GET -H "Accept: application/json" -H "Authorization: Basic $2"
"$1/current" | jq -r '.id')

# get cluster
cluster_id=$(curl -X GET -H "Accept: application/json" -H "Authorization: Basic $2"
"$1/$company_id/clusters" | jq -r '.data[0].id')

# get services list (we have traefik and apachephp; we select the second one)
service_id=$(curl -X GET -H "Accept: application/json" -H "Authorization: Basic $2"
"$1/$company_id/clusters/$cluster_id/services" | jq -r '. | .data[1].id')

# create a $SECRET on Medstack's cluster
curl -X POST -s -H "Accept: application/json" -H "Authorization: Basic $2" -H "Content-Type:
application/json" "$1/$company_id/clusters/$cluster_id/secrets" --data
"{\"name\":\"mysecret\",\"data\":\"$3\"}" | jq -r '.id'

# update the service with the new secret
curl -X PATCH -s -H "Accept: application/json" -H "Authorization: Basic $2" -H
"Content-Type: application/json" "$1/$company_id/clusters/$cluster_id/services/$service_id"
--data
"{\"secrets\":[{\"name\":\"mysecret\",\"file_name\":\"mysecret\",\"uid\":\"0\",\"gid\":\"0\"
,\"mode\":\"420\"}]}"
```

Lastly, **index.php** contains this script:

```
<!-- File: index.php -->
<html>
<body>

<h4>Congratulations, this demo app worked!</h4>
<p>Your MedStack Secret was set to <?php echo getenv('THESECRET'); ?></p>
<p><img src="https://www.siterocket.com/cicddemo/image.jpg"></p>

</body>
</html>
```

As you can see, we added a Secret in this demo. Our “hello world” uses the PHP getenv command to read the secret and display it.

If you access https://IP, where IP is the IP that Medstack provisioned to your node, you will see the index.php file successfully deployed and the Secret displayed.

## Get your Team on Board

Now your script is up and running, and you can run deployments simply by merging into specific branches of your code repos.

Although the technical work is now behind you, you’ll need to ensure that your team is up to speed on how to work with your new setup. It’s usually best to review the CI/CD tools with your team, and ensure that everyone is on the same page with your code branching strategy and release schedule. You might also want to schedule UAT (user acceptance testing) releases and beef up the test coverage of your unit tests.

Making the most of your CI/CD implementation will lead to safer and more reliable deploys, a better software development lifecycle, and a more productive team.

Happy building!

SiteRocket® Labs

MEDSTACK®

# Acknowledgements

SiteRocket® Labs

MedStack would like to thank the awesome team at SiteRocket Labs, who put in the time and effort to help us create this guide. Putting together a guide of this length and detail is not easy, especially while managing your daily company tasks, and we would not have been able to do it without them.

Thank you.

## About SiteRocket Labs

SiteRocket Labs designs, develops and maintains health and medical applications that solve unique and complex challenges. Our clients range from early stage startups to large companies – and we work closely with them to ensure that their digital health applications are reliable, secure, and compliant with privacy regulations such as HIPAA and PHIPA.

info@siterocket.com siterocket.com

## About MedStack

MedStack is a cloud automation technology company that builds and manages healthcare privacy and security compliance into cloud hosting tools to help bring apps to market faster, more easily and more affordably. MedStack has powered hundreds of digital health solution vendors and actively manages compliance policies in several countries around the world. The company is proudly based in Toronto, Canada.

info@medstack.co medstack.co