

FACULTY OF AUTOMATION AND COMPUTER SCIENCE COMPUTER SCIENCE DEPARTMENT

DISTRIBUTED SYSTEMS

CI/CD Tutorial and Deployment on cloud (Heroku Cloud)

Ioan Salomie

Tudor Cioara Claudia Antal Marcel Antal

2023-2024

Contents

1.	Overv	view	3
2.	Conti	nuous Integration (CI) - Backend	4
2	.1. S	etting the gitlab-ci file	4
	2.1.1.	Setting the Build phase	5
	2.1.2. 2.1.3.	Setting the Testing phase	6 7
2	.2. Т	est your solution	8
3.	Conti	nuous Deployment (CD) – Backend	9
3	.1. S	etting the Application on Heroku	9
3	.2. S	letting the Continuous Deployment phase on Gitlab 1	0
	3.2.1.	Test your solution 1	2
3	.3. S	etting the Database on Heroku (Postgres) 1	2
	3.3.1.	Test your solution: 1	4
4.	CI/CI	D – Frontend 1	5
4	.1. Т	Sest your solution:	5
Ref	erences	s 1	5

1. Overview

From this tutorial you will learn how to configure the CI/CD pipeline in Gitlab for a spring-boot application. You can use the source code provided in [1] and [2] and setup your own repository on Gitlab and following the instructions and the exercises from *"Test your solution"*. By the end of the laboratory you should have your own backend and frontend application configured to run both the CI and CD pipeline.



For setting your Gitlab repositories:

1. First setup a Gitlab Group by going to: *Groups* →*Your Groups* → *New Group* and create your *private* group for the DS project as:

DS2020_GroupNumber_LastName_FirstName (e.g. DS2020_30441_Popescu_Ioan)

2. Give access to your group for the DS lab assistants. On your Group page go to:

Members \rightarrow Invite Member \rightarrow

and offer Maintainer rights for the user: utcn.dsrl@gmail.com

- 3. Inside the group, you can create your own projects for different applications of the DS lab. Make sure you keep the naming conventions for the projects as well, considering one of the following formats:
 - DS2020_GroupNumber_LastName_FirstName_AssigNumber
 - DS2020_GroupNumber_LastName_FirstName_AssigNumber_Backend
 - DS2020_GroupNumber_LastName_FirstName_AssigNumber_Frontend

2. Continuous Integration (CI) - Backend

Continuous Integration (CI) refers to a pipeline of steps that are applied whenever your code is pushed on the code repository. It aims to validate that the code you developed does not affect the previously developed features and that the integration between your newly developed code and the previous code is done correctly. Specifically, this can be done in 3 steps:

- 1. Verify that the project builds correctly
- 2. Verify that the tests run and are successful
- 3. Verify that there are no major code style issues

In the following sections each of these verification steps will be addressed.

*) Additional information and the source code for this tutorial can be find at [1]

Exercise: You can download the source code from [1] and upload it in your own Gitlab repository or push your own Spring Boot source code online.

2.1. Setting the gitlab-ci file

In order to trigger the pipeline of continuous integration on Gitlab, it is necessary to add a .gitlabci.yml configuration file, where you need to specify all the pipeline steps together with the commands and configurations necessary to successfully run that step.

The .gitlab-ci.yml need to be registered in the root of the project as depicted in Figure 1.

Name	Last commit	Last update
🖨 src	jacoco and checkstyle enabled	2 days ago
♦ .gitignore	initial commit	3 days ago
₩ <u>.gitlab-ci.yml</u>	Update .gitlab-ci.yml	2 days ago
M+ README.md	Initial commit	4 days ago
b checkstyle.xml	configure jaccoco and checkstyle	3 days ago
b pom.xml	jacoco and checkstyle enabled	2 days ago

Figure 1 Project structure for setting CI on Gitlab

In order to validate that the CI/CD configuration is possible, make sure that you have the Runners enabled for your project.

For this go to: *Settings* \rightarrow *CI/CD* \rightarrow *Runners*



Figure 2 Check Runners are enabled

By default the runners are enabled, thus you do not have to do anything.

In the .gitlab-ci.yml file we configure the 3 stages (Figure 3) required to be run whenever new code is pushed in the repository:

1	stages:	
2	- build	
3	- test	
4	- checkstyle	

Figure 3 CI Pipeline stages

2.1.1. Setting the Build phase

During the build phase, the build command is issued on the source code that has been updated:

build:					
stage:	build				
image:	maven:3.3.9-jdk-8				
script:					
- mvn	clean package				

Figure 4 Build Stage

Having a Spring Boot application configured using Maven, in order to build our code we need to specify that this stage needs to be executed on an image preconfigured with Maven. Furthermore, using the *script* tag, the command needs to be given that will be executed for this stage.

Exercise: You may change the command and specify mvn clean install

2.1.2. Setting the Testing phase

During the test stage, all the tests are expected to be run in order to detect any possible problems that may have raised as a result of updating the code.

est:
stage: test
<pre>image: maven:3.3.9-jdk-8</pre>
script:
- echo "Test DEMO app"
- mvn test && mvn jacoco:report
- cat target/site/jacoco/index.html grep -o '.*'
<pre>coverage: "/Total.*?([0-9]{1,3})%/"</pre>

Figure 5 Test Stage

Similarly to the build stage, we start from a maven image, however, the script tag contains the maven command for executing the tests.

Additionally, the jacoco plugin is used (check the project's pom.xml lines 94-151) in order to generate the test reports. This is useful in order to extract useful information about the testing phase, such as the test coverage (the percentage of lines of code covered by tests).

As a result the test coverage is depicted in the test stage details. In order to see the test details:

Select $CI/CD \rightarrow Pipelines \rightarrow$ Select and Click the Test phase from the Pipeline \rightarrow Test
Coverage is depicted on the right side of the screen (Figure 7)

Project overview	I	DS_2020 → spring-de	emo > Pipelines		
Repository		All 21 Finished	d Branches Tags		
D Issues 0		Filter pipeline	!S		
ំំំង Merge Requests 0		Status	Pipeline	Triggoror	Commit
🥠 CI / CD (1)		Status	ripeline	inggerer	Commit
Pipelines		@passed	#189661785 latest		γ master ↔ 390a6a47 🏘 Update.gitlab-ci.yml
Jobs Schedules		(Q passed)	#189659518	A	γ master -0-7b7e663c
		(passed			🍓 Update .gitlab-ci.yml

Figure 6 Test Stage Selection

test	etry
Duration: 1 minute 49 seconds	
Timeout: 1h (from project)	0
Runner: shared-runners-manager- 4.gitlab.com (#44949)	
Coverage: 53%	
Job artifacts These artifacts are the latest. They wi not be deleted (even if expired) until newer artifacts are available.	II
Commit 7b7e663c	
 Pipeline #189659518 for master test 	~

Figure 7 Test Stage Report

2.1.3. Setting the Check style phase

During the check style phase (or the linting phase) the code is evaluated in order to detect possible coding style errors or warnings. These errors can be signs of suspicious constructs that can lead to future programming errors.

checkstyle:
<pre>stage: checkstyle</pre>
<pre>image: maven:3.3.9-jdk-8</pre>
script:
- echo "Checkstyle DEMO app"
- mvn checkstyle:check

Figure 8 Checkstyle Stage

For the current pipeline the checkstyle plugin is used (see the project's pom.xml [1]). The plugin relies on a configuration file (checkstyle.xml – see the project structure [1]) that specifies which coding errors / warning to be investigated when the linter is applied. For official configurations, check the 'Google Java Style' xml at [3].

Whenever the check phase encounters an error in your coding style, the check style phase will fail.

2.2. Test your solution

By this point your .gitlab-ci.yml should look like this:

```
📄 .gitlab-ci.yml 485 Bytes 🛛 🔓
  1 stages:
       - build
       - test
  4
       - checkstyle
 5
 6 build:
       stage: build
      image: maven:3.3.9-jdk-8
 8
 9
      script:
 10
         - mvn clean package
 12 test:
        stage: test
        image: maven:3.3.9-jdk-8
 14
 15
        script:
         - echo "Test DEMO app"
 16
        - mvn test && mvn jacoco:report
     - cat target/site/jacoco/index.html | grep -o '.*'
coverage: "/Total.*?([0-9]{1,3})%/"
 19
 20
 22 checkstyle:
       stage: checkstyle
 24
        image: maven:3.3.9-idk-8
        script:
- echo "Checkstyle DEMO app"
 25
26

    mvn checkstyle:check
```

Figure 9 CI setup

Make a modification on your code, locally and then push it to your Gitlab account.

- Check the CI/CD pipelines and make sure that the pipeline is triggered once your code is loaded on git.
- If the pipeline fails, make sure you make the necessary adjustments in your code/ configuration.
- Make sure you identify all the details regarding the configuration (e.g. the test coverage is displayed)
- Once your pipeline succeeds proceed to the next point regarding Continuous Deployment.

3. Continuous Deployment (CD) – Backend

The continuous deployment aims at delivering as fast as possible the new features added through your code to the deployment servers. This can be configured also as a stage in the .gitlab-ci.yml file, such that if all the previous steps (build, test, and checkstyle) are successful, it may proceed with the deployment of your application on a server. In order to avoid unnecessary deployments, it is recommended to configure the deployment stage such that to be run only for specific branches, when the feature you are working on is completed, and ready to be delivered to the end-user. More details about this will be covered in section 3.2.

3.1. Setting the Application on Heroku

In order to be able to deploy your application, a server instance or a cloud account is required. For this setup, we chose to use the Heroku cloud.

Sign up for experience He	free and eroku today
③ Free account Create apps, connect databases and add-on services, and collaborate on your apps, for free.	First name * First name Last name * Last name
Your app platform A platform for apps, with app management & instant scaling, for development and production.	Email address * Email address Company name Company name Role *
Deploy now Go from code to running app in minutes. Deploy, scale, and deliver your app to the world.	Role Country Country Country Primary development language Select a language

You can create your own account on Heroku for free: https://signup.heroku.com/

Figure 10 Heroku Free account

Upon registration you will be able to create a **New Application**. We named our application: "spring-demo-ds2020". For your application, make sure you provide a unique name, that does not already exist.

Then from your Profile, Account Settings \rightarrow API key \rightarrow click reveal and copy the content of your API key.

АРІ Кеу	Regenerate API Key	Reveal
	Figure 11 Heroku API key	

3.2. Setting the Continuous Deployment phase on Gitlab

The API key needs to be added in the Gitlab's project configuration, in order to be granted access when deploying the application.

For this go to: *Settings* \rightarrow *CI/CD* \rightarrow *Variables* \rightarrow *Add Variable*

S coving domo										
5 spring-demo		Gener	al pipelines							Expand
Project overview		Customiz	e your pipeline configu	iration, view y	our pipeline status and coverage	e report.				
Repository		Auto I	DevOps							Expand
D Issues 0		build, test, an t Auto DevOp:	d deploy applications based on s or use our quick start guide to	predefined contir get started right	nuous integration away.	and delivery				
11 Merge Requests 0										
🥠 CI/CD		Runne	ers							Expand
Security & Compliance		Runners a informati	are processes that pick on	up and execu	te jobs for GitLab. Here you can	register and see y	your Runners for	this project. More		
Operations			(1)							
	L	Variab	les ⑦ ⁽³⁾							Collaps
Packages & Registries		Environm	ent variables are applie	ed to environn	nents via the Runner. You can us	e environment va	ariables for passw	ords, secret keys,	etc.	
Lt Analytics		Make var	lables available to the n	unning applic	ation by prepending the variabl	e key with kas_si	ECRET YOU Can:	set variables to be:	:	
🖞 Wiki		• Pr • Ma	sked variables are d sked variables are hidd	oniy exposed den in job log:	to protected branches or tags. s (though they must match certs	iin regexp require	ments to do so).			
X Snippets		More info	ormation							
A Members			Enviror	nment variabl	es are configured by your admir	iistrator to be <mark>pro</mark>	tected by defaul	t		
🗘 Settings (1)		Туре	↑ Key		Value	Protected	Masked	Environments		
General		Variable	HEROKU_API_KEY		****	~	×	All (default)	Ø	
Integrations									(4)	
Webhooks							Reveal	values Add Va	ariable	
Repository										
CL/CD (2)		Group va	riables (inherited)							
		These var	iables are configured ir	n the parent g	roup settings, and will be active	in the current pro	oject in addition 1	to the		

Figure 12 Setup API key in Gitlab

Add the HEROKU_API_KEY variable and paste the API Key that you copied in section 2.1. from Heroku.

Add variable		×
Key		
HEROKU_API_KEY		
Value		
PASTE API KEY		^
		-
Туре	Environment scope	
Variable 🗘	All (default)	*
Flags ✓ Protect variable ● Export variable ● Mask variable ● Variable will be masked in iob loos. Requires values to	ches and tags only, meet regular expression requirements. More information	
	Cancel Add varia	ble

Figure 13 Gitlab Variable for Heroku API ley

Now you can add the deployment stage to your .gitlab-ci.yml file. So the final version of your .gitlab-ci.yml should look like Figure 14, considering for the --app flag, the name you have provided for the Heroku application.

🖻 .gitlab-ci.yml 968 Bytes 🕯

```
1
  stages:
     - build
     - test
З

    checkstyle

4
5
     - deploy
6
7 build:
    stage: build
8
9
    image: maven:3.3.9-jdk-8
    script:
10
       - mvn clean package
13 test:
14
      stage: test
      image: maven:3.3.9-jdk-8
      script:
16
       - echo "Test DEMO app"
       - mvn test && mvn jacoco:report
18
       - cat target/site/jacoco/index.html | grep -o '.*'
19
      coverage: "/Total.*?([0-9]{1,3})%/"
20
23 checkstyle:
24
      stage: checkstyle
      image: maven:3.3.9-jdk-8
      script:
26
       - echo "Checkstyle DEMO app"
      - mvn checkstyle:check
28
29
30 variables:
      DB IP: ec2-52-48-65-240.eu-west-1.compute.amazonaws.com
       DB PORT: 5432
      DB_USER: wlryktxyqpyomt
34
      DB_PASSWORD: bee98a2afc7f0c3bcdd7df60ee7278ec5fa5cb4fb06a4039b1ffb1107d5851fd
      DB_DBNAME: devidei2vqv0v4
36
37 deploy:
38
     stage: deploy
39
   image: ruby:2.4
40
   script:
41

    apt-get update -qy

42
       - apt-get install -y ruby-dev
43
       - gem install dpl
      - dpl --provider=heroku --app=spring-demo-ds2020 --api-key=$HEROKU_API_KEY
44
45
     only:
       - production
46
```

Figure 14 FINAL CI/CD configuration

For this setup, we used ruby gems to deploy our spring-boot application on Heroku. As marked in Figure 14, you need to specify the name of your application created in Heroku, and reference the API key variable previously added in Gitlab. Furthermore, you can configure your deployment stage to run only when modifications appear on the specified branches.

3.2.1. Test your solution

Make a modification on your code, locally and then push it to the Gitlab.

- Check the CI/CD pipelines and make sure that the Gitlab can connect to the Heroku application
- The deployment should be successful, although your spring-boot application will fail to start since it cannot connect yet to a database on the Heroku cloud.
- Make a protected *production* branch and make sure that it runs the deployment only when you merge your developed features in the production branch.

Configuring protected branches

- 1. Navigate to your project's Settings \rightarrow Repository.
- 2. Scroll to find the **Protected branches** section.
- 3. From the **Branch** dropdown menu, select the **branch** you want to **protect** and click **Protect**. ...
- 4. Once done, the protected branch will appear in the "Protected branches" list.

3.3. Setting the Database on Heroku (Postgres)

On your application page in Heroku go to Overview and then select Configure Add-ons.

Overview	Resources	Deploy	Metrics	Activity	Access	Settings
Installed a	dd-ons	onth			C	
Installed a		onth			Cont	<u>igure Add-ons</u> 🕣
	You can add ad	There are r d-ons to this	no add-ons app and the	for this app y will show he	re. <u>Learn mo</u>	ore
Dyno form	ation \$0.00/mo	nth			<u>Co</u>	onfigure Dynos 🕥
	Add a Procfile to	This app ha	as no proce order to defin	ss types yet e its process t	ypes. <u>Learn r</u>	<u>more</u>
Collaborat	or activity 👔				Δ	Manage Access ④
	Tł	nere is no re	ecent activi	ty on this ap	р	
	Collaborator	activity will b	e shown whe	n there are re	cent deploy	5

Search the Heroku Postgres Add-on (there are other add-ons for databases as well, make sure you chose a free one) and click *Provision*.

DISTRIBUTED SYSTEMS

CI/CD

Personal 0 > Description description boot	☆ Open app More ≎
Overview Resources Deploy Metrics Activity Access Settings	
Dynos	
This app has no process types yet Add a Procfile to your app in order to define its process types. <u>Learn more</u>	
Add-ons	Find more add-ons
Q, heroku poj	
Heroku Postgres There are no add-ons for this app You can add add-ons to this app and they will show here <u>Learn more</u>	
Estimated Monthly Cost	\$0.00
Heroku Postgres $ ightarrow$ ds2020-spring-boo By choosing "Provision", this will add Heroku Postgres on yu personal ds2020-spring-boot application.	× ot our
Plan name	
Hobby Dev – Free	0
<u>View add-on details in Elements Marketplace</u> By provisioning this add-on, I agree to the <u>Terms of Servic</u>	<u>e</u> .
Provision	

Figure 15 Add database add-on

Under the Overview tag from your application, the Postgres add-on should appear.



Figure 16 Installed Add ons for application

By selecting your Postgres add-on you will be redirected to the Datastores page, where you can see the details about the Database credentials and other connection information.

DISTRIBUTED SYSTEMS

K DATA		9
SERVICE heroku-postgresql PLAN hobby-dev BILLING APP 🌒 spring-demo-ds2020		
Overview Durability Settings Dataclips		
ADMINISTRATION		
Database Credentials		_
Get credentials for manual connections to this database.	View Credentials	۶
Reset Database		
Reset the database to its originally-provisioned state, deleting all data inside it.	Reset Database	·
Destroy Database		
Destroys the database and all of the data inside it.	Destroy Database	·

Figure 17 Database information

OBSERVATION! Based on the Database credentials obtained from the Heroku Administration page, update the .gitlab-ci by modifying the marked Environmental Variables from Figure 18.

20	valiables.
31	DB_IP: ec2-52-48-65-240.eu-west-1.compute.amazonaws.com
32	DB_PORT: 5432
33	DB_USER: wlryktxyqpyomt
34	DB_PASSWORD: bee98a2afc7f0c3bcdd7df60ee7278ec5fa5cb4fb06a4039b1ffb1107d5851fd
35	DB_DBNAME: devidei2vqv0v4
36	
37	deploy:
38	stage: deploy
39	<pre>image: ruby:2.3</pre>
40	script:
41	- apt-get update -qy
42	- apt-get install -y ruby-dev
43	- gem install dpl
44	- dplprovider=herokuapp=spring-demo-ds2020api-key=\$HEROKU_API_KEY
45	only:
46	- production

Figure 18 DB Credentials

3.3.1. Test your solution:

- Using the connection details, you can update your *application.properties* file from your spring boot application to connect to the cloud database you have just created and test it from your IDE locally.
- Trigger the online pipeline on the production branch and check if successful
- From the Heroku page, click on the *Open App* button and you should be redirected to the index of your application deployed on cloud.

IMPORTANT! During the development phase use a local database, since the database created in the Heroku platform has a limited usage plan. Whenever you consider that a feature is completed, switch to your **production** branch.

4. CI/CD – Frontend

For the Frontend application, the same principles apply when setting the CI/CD pipeline. At [2] you can find a React application configured to be built and deployed on Heroku.

- 4.1. Test your solution:
- Configure your own frontend application and supply the necessary connection details (hostname in hosts.js) so that the frontend application can connect to your backend application.

References

- [1] https://gitlab.com/ds_20201/spring-demo
- [2] https://gitlab.com/ds_20201/react-demo
- [3] https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/google_checks.xml