

FACULTY OF AUTOMATION AND COMPUTER SCIENCE COMPUTER SCIENCE DEPARTMENT

DISTRIBUTED SYSTEMS

CI/CD Deployment using Docker on Azure

Rancea Alexandru

Marcel Antal

Claudia Antal

2023-2024

\sim				
Со	nt	· (-)	nı	ΓC.
0	110	. –		L J

 Overview Docker 	
 2.1. What is Docker? Why choosing Docker over VMs? 2.2. How to install Docker 2.3. Docker container lifecycle 	5
Docker REGISTRY Create Docker Image Create Docker Volume Configure Network Create Container Check Running container and logs Stop/Remove Container and Image Enter terminal of a container	6 7 7 8 8
 2.4. Docker Commands Summary 2.5. DOCKER deploy example 2.5.1. Deployment of Spring application 	8 9 9
2.5.2. Deployment of React application	15
4.1. Azure CI Automatic Build Docker Image.14.1.1. Configure Azure.14.1.2 Azure DevOps24.1.3 Azure Pipeline Agent.24.1.4 Azure CI24.1.5 Azure CD.2	17 20 22 25
5. Further development	

1. Overview

From this tutorial you will learn how to configure the CI/CD pipeline in Gitlab for a spring-boot application using Dockers. You can use the source code provided in [1] and [3] and setup your own repository on Gitlab and following the instructions and the exercises from *"Test your solution"*. The docker-configuration is found on the *docker-production* branch in the specified repositories. By the end of the laboratory you should have your own backend and frontend application configured to run both the CI and the CD pipeline using Dockers.

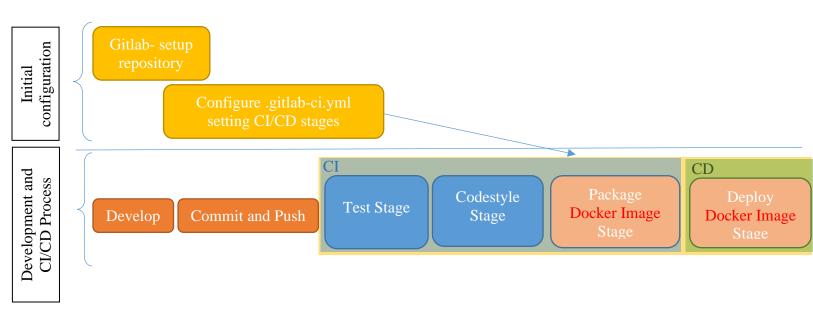


Figure 1. CI/CD Pipeline

With respect to the previous CI/CD setup established for the first assignment on the **production** branch of the repositories, the docker-based CI/CD replaces the Build Phase and Deploy Phase with docker instructions that will be exemplified in the following two chapters.

The Heroku cloud allows for the <u>free account a 60 second boot time and 512MB memory</u>. For this reason, some extra measures are considered when configuring the Docker images in order to improve the application startup resources consumption.

2. Docker

2.1. What is Docker? Why choosing Docker over VMs?

The Operating System divides the computer memory in several sections, where the **Kernel space** and the **User space** are most important, as shown in Figure 2. The **Kernel Space** is the portion of memory where privileged operating system kernel processes are executed, while the User Space contains unprivileged processes. The separation is performed using a set of privileges. Programs or processes are run in user mode and are sandboxed, meaning that they are isolated from other processes from the memory point of view and cannot have complete access to the computer's memory, disk storage, network hardware, and other resources.

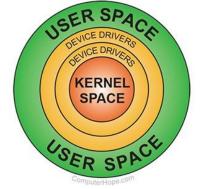


Figure 2. Computer Memory privileges separation (Source [8]).

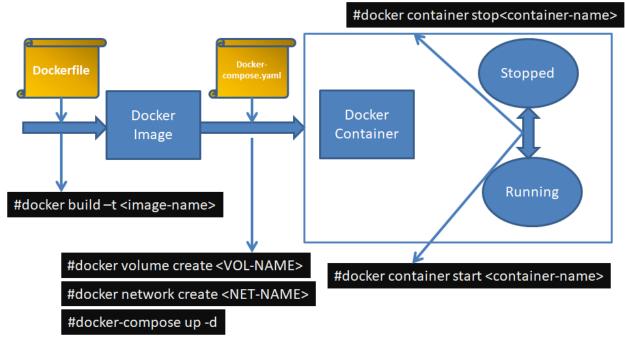
Starting with 1970, IBM and later other companies started developing special software called **Hypervisor**, used to create and run Virtual Machines (VMs) [9]. A VM is an emulation of a computer system, based on a computer architecture and providing the functionality of a physical computer [15]. Several VMs with different hardware requirements and guest OSes can be run on a host computer. Multiple instances of a variety of operating systems may share the virtualized hardware resources: for example, Linux, Windows, and macOS instances can all run on a single physical x86 machine. The hypervisor runs in **Kernel mode**, while the **guest OS** runs in **user mode**, thus theoretically being **sandboxed** from the host OS.

As opposed to virtualization, **Docker** is a container-based technology where containers are running as processes in the user space of the operating system. Docker originally used LinuX Containers (LXC), but later switched to runC (formerly known as libcontainer), which runs in the same operating system as its host. This allows it to share a lot of the host operating system resources. At the low level, a container is just a set of processes that are isolated from the rest of the system, running from a distinct image that provides all files necessary to support the processes. It is built for running applications. In Docker, the containers running share the host OS kernel.

Features	VM	Docker
Host OS	Any	Linux – based (if installed on
		Windows it installs a VM with
		Linux)
Guest OS	Any	Linux – based (it uses the
		kernel of the host operating
		system)
Sandboxing	Full isolation	Can access host through shared
		filesystem (such as docker-
		volume)
HW Resource requirements	High (similar to the physical	Low (many containers can run
	machine emulated)	on the same physical machine)

2.2. How to install Docker

Recommended to use Linux. Docker for Windows seems to create a VM with Linux on it, on which it runs Docker, inside which it then runs containers. So much indirection might lead to problems in the future (such as managing volumes). Furthermore, **docker** commands need **privileged** access, using **sudo** command. Follow the tutorial here and install Docker CE [7]: https://docs.docker.com/install/linux/docker-ce/ubuntu/



2.3. Docker container lifecycle

Figure 3. Docker lifecycle and basic commands

The first step would be to create an empty folder for each docker container you want to create. This folder should contain at least 2 files, with exactly these names (since only these are recognized by docker). The files will be detailed in the following sections.

- Dockerfile the description of the image
- Docker-compose.yaml the description of the container.

Docker REGISTRY

The Docker registry [13] is a server application that stores and distributes Docker images. Almost all custom images that will be build are based on an existing image that already exists in the Docker Registry [14]. Images can also build from scratch, without inheriting any parent image.

Create Docker Image

The Docker image is described by a *Dockerfile*. An image can be created either by inheriting a parent image, or by creating a base image from scratch [14]:

- **Inherit a parent image**: the new image will customize an existing image (parent image) from Docker Registry, by referencing it using the FROM directive at the beginning of the Dockerfile. All the other instructions in the docker file modify the parent image.
- **Create a base image**: a base image is created by using the FROM scratch directive at the beginning of the Dockerfile.

Thus, a docker image can either be used directly from an online repository (e.g. the Docker REGISTRY) or it can be customized starting from a parent image by describing it in a Dockerfile and issuing the following command in the terminal:

#docker build –t <image-name>

An example for a Dockerfile for Nodejs server (downloaded from Docker REGISTRY) customized for deploying an application is the following:

```
FROM node:8
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
CMD npm start
EXPOSE 3000
```

By issuing the docker build command, an image with the name <image-name> will be created. At this stage, the image is stored locally and can be viewed using the command:

#docker image ls

Create Docker Volume

To create a docker volume you just have to run:

#docker volume create <vol_name>.

The information from the volume will be stored at /var/lib/docker/volumes/<vol_name>/_data. You can copy any files of interest directly here and they will appear in the container in the specified folder (see Figure 4).

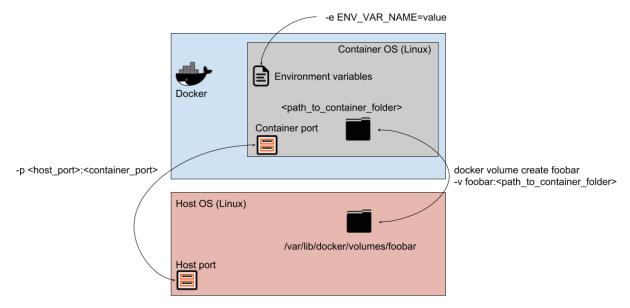


Figure 4. A docker container and communication with the host (port mapping and shared folders using volumes)

To see all available volumes, you can run docker volume ls. To remove a volume you can run *docker volume rm <vol_name>*.

Configure Network

A virtual network can be created between the containers. Each network has a name and a IP address. To create a network, the following command is used:

#docker network create <NET-NAME>

Create Container

There are 3 basic things that we want to synchronize between a docker container and the host:

- ports: map a container's port to a host's port (to be able to access the application from outside)
- volumes: map a container's folder to a host's Docker volume (to be able to persist information and do backups)
- environment vars: set the container's environment variables that can be used by the application for various configurations

You can use docker-compose files to write the configurations you want for a docker container. A docker-compose file MUST be named docker-compose.yaml, and for the syntax you can check the existing ones or the internet.

The docker-compose files for the most used images are already created, and you can find them in Docker-compose Scripts folder at this location (mysql, cassandra, tomcat, gitlab).

In order to run/create a docker container you have to move this docker-compose.yaml file to the desired computer, cd to its location and run:

#docker-compose up –d

Make sure the appropriate docker volumes are created before running the above command (e.g. tomcat volume for tomcat image, check the used volumes in each docker-compose file in particular).

Containers created and started with this are started and stopped with the classic Docker commands (start, stop, restart).

Check Running container and logs

The containers that are running can be checked with

#docker ps The logs of a container can be accessed using

#docker logs –f <container-name>

Stop/Remove Container and Image

Containers and corresponding images can be stopped and removed using the following commands (in this order):

#docker container stop <container-name> #docker container rm <container-name> #docker image rm <image-name>

Enter terminal of a container

One can enter the terminal of a container using the following command:

#docker exec -- it < container-name > / bin/bash

Furthermore, using the instruction *docker exec –it <container-name>*, other commands to the container can be appended.

2.4. Docker Commands Summary

Interogate running containers: > docker ps Interogate existing volumes: > docker volume ls Stop and Remove running containers: > docker stop {name} > docker rm {name} Remove existing volume: > docker volume rm {volume-name} Use docker-compose.yaml files, if the container needs a volume create it first with: > docker volume create {volume-name} Then start the docker container using: > docker-compose up -d Volumes location on host: /var/lib/docker/volumes/ Access container bash: docker exec -it [container-id] /bin/bash

2.5. DOCKER deploy example

In this section we will exemplify the deployment on Docker of the Spring demo application and React application from tutorials for Assignment 1. The code can be downloaded from:

- Spring Application [1]: git clone https://gitlab.com/ds_20201/spring-demo.git
- React Application [3]: git clone https://gitlab.com/ds_20201/react-demo

We suppose the database connection to the PostgreSQL deployed on Heroku in the previous laboratory session.

2.5.1. Deployment of Spring application

In order to be able to build a custom docker image containing your application's executable code a Dockerfile must be configured in the root directory of the project.

A Dockerfile and docker-compose.yml files are already available in the Spring demo if you switch the branch to *docker_production*. Otherwise, create a new branch and create yourselves the files specified in the laboratory work.

#git fetch -a #git branch -a #git checkout docker_production

The Dockerfile used for building the image for our application is presented in Figure 6.

Starting with line 1, an intermediate maven image called builder is configured. This is used in order to build the executable of the application from the source code. Thus, firstly the source code is copied in the temporary image (src, pom.xml and checkstyle.xml). Normally, a Spring Boot application can be started using this .jar file. **However, due to the boot resources limitation, we**

have considered a more optimal approach, by using layered Jars. More details about the Layered Jars and the reasons for using them can be found at [2].

1. In order to specify that a layered jar is required, firstly you need to modify lines 93-97 from pom.xml file, and add the following configuration, specifying that layers are enabled.



Figure 5. Section from pom.xml of spring-demo

- 2. The *mvn package* is run (Figure 6 line 7) in order to obtain the application's layered .jar file from the source code.
- 3. The layers of the created jar file are listed using command at line 8
- 4. The layers of the created .jar file are extracted using command at line 9
- 5. Lines 14-19 from the Dockerfile presented in **Figure 6** contain the DB credentials from the PostgreSQL deployed in Heroku in the previous laboratory session. (aici trb sa discutam)

5. Lines 14-19 from the Dockerfile presented in **Figure 6** contain the DB credentials from the PostgreSQL that is found locally either by installing Postgres or by using the latest Postgres Docker Image (docker pull postgres)

🖹 Dockerfile 1022 Bytes 🔓

Edit Web IDE L

1	FROM maven:3.6.3-jdk-11 AS builder
2	
З	COPY ./src/ /root/src
4	COPY ./pom.xml /root/
5	COPY ./checkstyle.xml /root/
6	WORKDIR /root
7	RUN mvn package
8	RUN java -Djarmode=layertools -jar /root/target/ds-2020-0.0.1-SNAPSHOT.jar list
9	RUN java -Djarmode=layertools -jar /root/target/ds-2020-0.0.1-SNAPSHOT.jar extract
10	RUN ls -1 /root
11	
12	FROM openjdk:11.0.6-jre
13	
14	ENV TZ=UTC
15	ENV DB_IP=ec2-52-48-65-240.eu-west-1.compute.amazonaws.com
16	ENV DB_PORT=5432
17	ENV DB_USER=wlryktxyqpyomt
18	ENV DB_PASSWORD=bee98a2afc7f0c3bcdd7df60ee7278ec5fa5cb4fb06a4039b1ffb1107d5851fd
19	ENV DB_DBNAME=devidei2vqv0v4
20	
21	
22	COPYfrom=builder /root/dependencies/ ./
23	COPYfrom=builder /root/snapshot-dependencies/ ./
24	
25	RUN sleep 10
26	COPYfrom=builder /root/spring-boot-loader/ ./
27	COPYfrom=builder /root/application/ ./
28	ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher","-XX:+UseContainerSupport -XX:+UnlockExperimentalVMOptions

Figure 6 Dockerfile for Spring Boot Application

Starting with line 12, a JDK 11 image is used and the layers obtained in the previous temporary image are copied in this image (lines 22, 23, 26, 27). Furthermore, the details for the DB connections can be specified in the Dockerfile as environmental variables. In this way, the source code will not contain the connection details but will be able to read them from the Environmental Variables set in the Dockerfile. This is possible by having the right configuration in your springboot *application.properties* file. Each variable form the *application.properties* has a format of $\{ENV_VAR_NAME: default_value\}$ specifying that, if the ENV_VAR_NAME is found in the local environmental variables, then that value is considered, otherwise the default value is considered. On one hand, in the development mode (when the project is setup in your IDE) there are no environmental variables set, so the default values are considered. On the other hand, when the docker image is launched, the environmental variables are set (Figure 7 lines 15-19) so the specified values are considered, and the default ones are ignored.

1	*****
2	### DATABASE CONNECTIVITY CONFIGURATIONS ###
3	***************************************
4	<pre>database.ip = \${DB_IP:localhost}</pre>
5	<pre>database.port = \${DB_PORT:5432}</pre>
6	<pre>database.user = \${DB_USER:root}</pre>
7	<pre>database.password = \${DB_PASSWORD:root}</pre>
8	<pre>database.name = \${DB_DBNAME:city-db}</pre>
0	

```
Figure 7. Application Properties section from spring-demo
```

Line 28 from the Dockerfile from Figure 6, specifies the command with which the newly created image should be launched. As noticed, there are several options set in order to optimize the resources consumption during boot time.

Remember to set the spring.jpa.hibernate.ddl-auto property to create/validate/update according to your database structure and contents.

2.5.1.1. Test your solution

Before continuing your configuration on the Gitlab repository, make sure that your *Dockerfile* written by you is correct and that the obtained image can run successfully. For this follow the instructions:

1. Create a *docker-compose.yml* file in the root of your project, containing the following lines. The name of the image must correspond to the name given as argument to the build command from step 2 (i.e. "your image name")

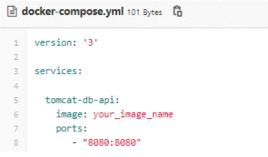


Figure 8. Docker-compose file of spring-demo image

- 2. Build your image using:
 - docker build -t your_image_name .
- 3. Start your image:
 - docker-compose up -d
- 4. Access your deployed application at *http://localhost:8080*. Furthermore, other endpoints of the application should be accessible, such as *http://localhost:8080/person*, returning the list of persons stored in the DB.

If everything is successful, you can push your newly created files on your repository (create a new branch like the example given *docker_production* branch) and proceed with the Gitlab configuration.

2.5.2. Deployment of React application

For the Frontend application, the same principles apply when setting the CI/CD pipeline. At [3] you can find a React application configured to be built and deployed on Heroku using dockers. The docker based CI/CD is configured on the **docker_production** branch.

Observations:

• Check the Dockerfile exposed on the **docker_production** branch:

CI/CD Docker



Figure 9. Dockerfile for docker production branch

Same approach is used as for the Maven project. In the first stage the application is built in an intermediate node image, while the built results are copied in the final nginx image. The Envsubst plugin is installed in order to make possible the parametrization of the nginx scripts with Environmental Variables. More details about this at [6].

• A nginx.conf file must be added in the root directory in order to specify the configuration for the nginx server. Here the port of the server is passed as an environmental variable by the Heroku cloud (line 2).

nginx.conf 175 Bytes					
1	server {				
2	listen \${PORT:80};				
З	server_name _;				
4					
5	<pre>root /usr/share/nginx/html;</pre>				
6	<pre>index index.html;</pre>				
7					
8	location / {				
9	<pre>try_files \$\$uri /index.html;</pre>				
10	}				
11	}				

Figure 10. nginx.conf configuration file

2.5.2.1. Test your solution

Before continuing your configuration on the Gitlab repository, make sure that the Dockerfile written by you is correct and that the obtained image can run successfully. For this follow the instructions:

1. Create a *docker-compose.yml* file in the root of your project, containing the following lines. The name of the image must correspond to the name given as argument to the build command from step 2 (i.e. "fe-image")

🖻 d	ocker-compose.yml 82 Bytes 🛱	
1	version: '3'	
2		
З	services:	
4		
5	react:	
6	<pre>image: fe-image</pre>	
7	ports:	
8	- "80:80"	

Figure 11. Docker-compose configuration file for React app

- 2. Build your image using:
 - docker build -t fe-image .
- 3. Start your image:
 - docker-compose up -d
- 4. Access your deployed application at *http://localhost*

If everything is successful, you can push your newly created files on your repository (create a new branch like the example given *docker_production* branch) and proceed with the Gitlab configuration.

3. Project Deployment

The goal of this tutorial is to deploy the software stack in Docker containers, to handle the heterogeneity of the platforms and to ease the migration to the Heroku platform.

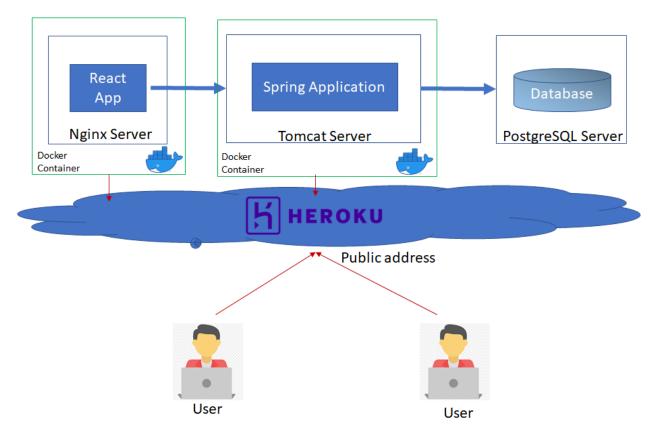


Figure 12. Project Deployment Diagram

Each module of the application, namely the database server and the backend server will be deployed in Docker containers that will be hosted on the Heroku cloud, thus eliminating the need of custom scripts used for application deployment in the previous laboratory tutorial "CI/CD Tutorial and Deployment on cloud (Heroku Cloud)". The database is deployed in an instance of PostgreSQL server as shown in the previous laboratory work.

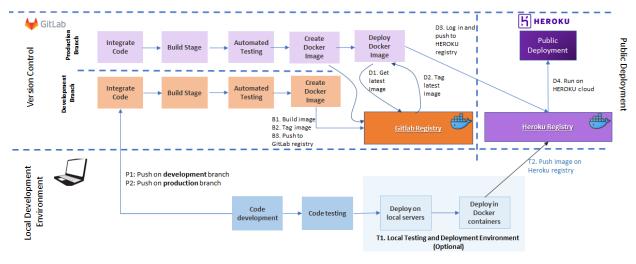


Figure 13. Project Deployment Pipeline

The steps that are involved in deploying the application on the Heroku cloud are depicted in Figure 13 and described in Table 2:

Step	Description	Detailed in section
P1	Develop code locally. Push on development branch	Previous laboratory
	while adding new features.	work: "CI/CD
P2	When a stable version is reached, merge the	Tutorial and
	development branch with the production branch and	Deployment on
	push the code.	cloud"
T1	Test that the solution runs without problems on a local	
(OPTIONAL)	test environment. Initially deploy on web servers, then	Section 2.5.
	deploy on local Docker containers.	
T2	Test the connection with the Azure Container	
(OPTIONAL)	Repository. Push the image to Azure Container	Section 4.2.1.
	Repository.	
B1	Build the image from the code pushed and built on one	
	of the branches from GitHub.	Section 4.1.1.
B2	Tag the built image	
B3	Push the image on Azure Container Repository	
D1	Login to GitLab registry and get the latest image from	
	the production branch (other images may exist from	
	building the development branch).	
D2	Tag the latest image	Section 4.2.2.
D3	Login and push the image to the Heroku registry	
D4	Trigger the automatic deploy of the image on Heroku	
	cloud	

Table 2.	Project	Deployment	Pipeline	Steps
----------	---------	------------	----------	-------

4. Azure CI/CD using Docker

This section covers the steps needed to deploy the Spring-demo app in Docker containers on the Azure:

- The first part of this section covers the setup for resource groups and the container registry.
- The second part of this section covers the setup to automate the process of creating images by the CI pipeline from GitHub and save them in Azure Container Repository.
- The third part of this section covers the setup to automate the process of deployment by the CD pipeline from Azure.

4.1. Azure Cl Automatic Build Docker Image

Consider the following setup on your *docker_production* branch. Set your branch as protected by going on GitHub to *Settings* \rightarrow *Branches* \rightarrow *Add rule.*

4.1.1. Configure Azure

Visit <u>https://azureforeducation.microsoft.com/devtools</u> \rightarrow *Sign in* and use your student email (@student.utcluj.ro) to complete the account creation.

Press the *Claim your Azure credit now* and complete with your own personal information.

After the log in, select from the top bar, the console/terminal icon. Select your preferred command line interpreter, and press Create storage.

To create a resource group, run the following command in the Azure terminal, created one step before: az group create --name nameLastNameGroupNumber --location germanywestcentral and change the name with your own information.

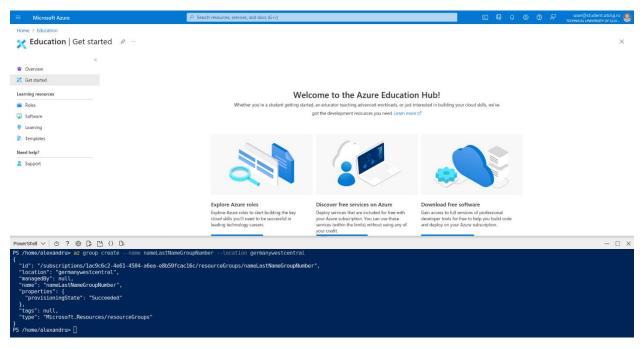


Figure 14. Azure Resource Group

After the resource group has been created, create a container registry by the following command: az acr create --resource-group nameLastNameGroupNumber --name

containerregistrynamelastnamegroupnumber --sku Basic and replace the --resource-group name with the one created above and --name with your own information (the container registry name should not contain capital letters).

Log out and log in again, if the Azure terminal says that there is no available subscription.

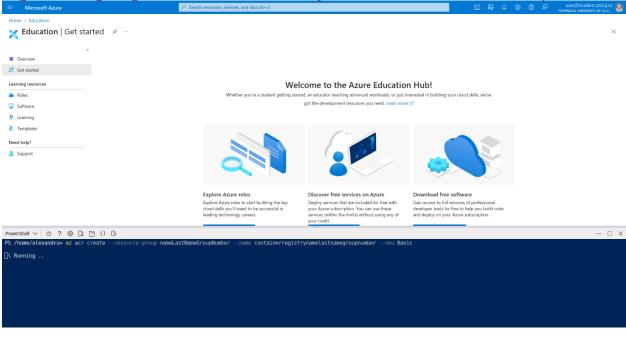


Figure 15. Azure Container Registry

To visualize the last two steps, select the *Menu bar* \rightarrow *Resource groups* \rightarrow *nameLastNameGroupNumber*. At this point, the only resource created in our group is the container registry.

Microsoft Azure	✓ Search resources, services, and docs (G+/)		F 🖉 🍥 🖗 🖓	user@student.utcluj.r TECHNICAL UNIVERSITY OF CLUJ.
e > Resource groups >				
nameLastNameG	roupNumber 🖈 🛧 …			
	« + Create @ Manage view ∨ 📋 Delete resource group 💍 Refresh 🞍 Export to C	SV 😚 Open query 🖗 Assign tags → Move 🗸 🗎 Delete 🞍 Export template	Open in mobile	
() Overview	↑ Essentials			JSON V
 Activity log 	Subscription (move) : Azure for Students	Deployments : No deployments		
Access control (IAM)	Subscription ID : 1ac9c6c2-4e61-4504-a6ea-e8b59fcac16c	Location : Germany West Central		
Tags	Tags (edit) : de :			
A Resource visualizer				
🗲 Events	Resources Recommendations			
Settings	Filter for any field	filter		
Deployments	Showing 1 to 1 of 1 records. Show hidden types ①		No grouping V	☐ III List view ✓
Security				List view V
Policies	Name 🕆	Type ↑↓	Location ↑↓	
Properties	🔲 🗣 containerregistrynamelastnamegroupnumber	Container registry	Germany West Central	
🔒 Locks				
Cost Management				
 Cost analysis 				
Cost alerts (preview)				
Budgets				
Advisor recommendations				
Monitoring				
Insights (preview)				
Alerts				
iá Metrics				
Diagnostic settings	Burling B. Annu 14 Martin			
🧬 Logs	< Previous Page 1 v of 1 Next >			Give feedbar
· · · · · · · · · · · · · · · · · · ·				

Figure 16. Azure Resource Group and Container Registry

A mandatory example to do before moving forward is to push to latest Postgres image in our Azure Container Repository. This is done with the following steps:

1. Open a CLI (Command Line Interpreter) locally (on your personal computer), and login into the Azure Container Registry:

docker login containerregistrynamelastnamegroupnumber.azurecr.io -u containerregistrynamelastnamegroupnumber -p sTsnMgSQa0dx5LD=qWqEtcIv6zmJSmMP;

If you receive the following error: accepts at most 1 arg(s), received 3, this means you are on a Windows machine and need to run the following command:

docker login -u containerregistrynamelastnamegroupnumber –p sTsnMgSQa0dx5LD=qWqEtcIv6zmJSmMP containerregistrynamelastnamegroupnumber.azurecr.io;

The credentials can be found in: Home \rightarrow Resource Group \rightarrow nameLastNameGroupNumber \rightarrow containerregistrynamegroupnumber \rightarrow Access keys and by enabling the Admin user. Replace the Login server, Username and password with your own details. Use the first password.

		$\mathcal P$ Search resources, services, and docs (G+/)			Ð	₽ ⊚	0 F	user@student.utcluj.ro
Home > Resource groups > nameLas	stNameGroupNumber > containerregistry	ynamelastnamegroupnumber						
containerregistryn	amelastnamegroupnum	nber Access keys ☆ …						×
	Registry name	containerregistrynamelastnamegroupnumber	D					
Overview	Login server	containerregistrynamelastnamegroupnumber.azurecr.io	0					
Activity log	Admin user 🛈	Enabled						
Access control (IAM)								
🗳 Tags	Username	containerregistrynamelastnamegroupnumber	0					
Quick start	Name	Password	Regenerate					
🗲 Events	password	sTsnMgSQa0dx5LD=qWqEtclv6zmJSmMP	0					
Settings	password2	ykWVt4JlisBM92eubQKDYns3NZRx4y/O	0 0					
Access keys								
Encryption								
💲 Identity								
Networking								
Ø Microsoft Defender for Cloud								
🔒 Locks								
Services								
Repositories								
🖧 Webhooks								
Replications								
🚖 Tasks								
Connected registries (Preview)								
Repository permissions								
O Tokens (Preview)								
?: Scope maps (Preview)								
Policies								

Figure 17. Azure Container Registry Credentials

2. Pull the latest docker image for Postgres

docker pull postgres;

3. Create a tag for the Postgres image

docker tag postgres containerregistrynamelastnamegroupnumber.azurecr.io/db

Modify the containerregistrynamelastnamegroupnumber with your details.

4. Push the image to your Azure Container Registry

docker push containerregistrynamelastnamegroupnumber.azurecr.io/db

Modify the container registry namelast name group number with your details. 5. Go to Home->Resource Groups \rightarrow nameLastNameGroupNumber \rightarrow container registry name group number \rightarrow Repositories to check if the image is in your Azure Container Repository.

4.1.2 Azure DevOps

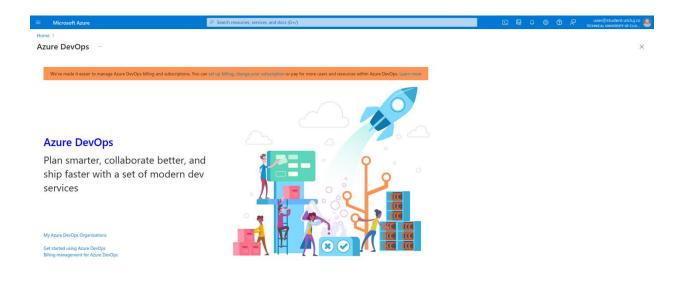


Figure 18. Azure DevOps

Before developing the CI/CD pipeline, go to My Azure DevOps Organziations \rightarrow Create new organization and follow the steps presented in Figure 19 and Figure 20.

	Azure DevOps user@student.utcluj.ro Almost done Name your Azure DevOps organization	
- <u>-</u>	Wallie your Acute Decision signification dev.azure.com/ nameLastNameGroupNumber West Europe West Europe Enter the characters you see New Audio	
	Continue	

Figure 19. Create Azure DevOps Organization

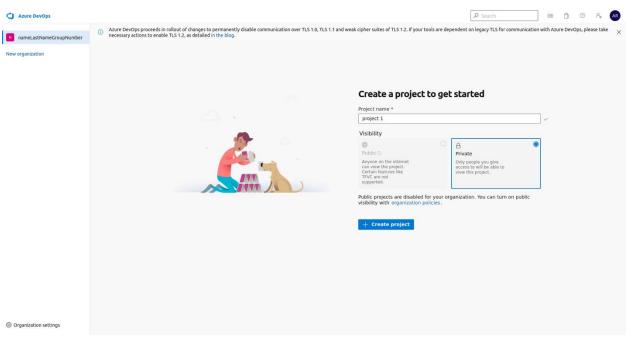


Figure 20. Create a project in Azure DevOps

4.1.3 Azure Pipeline Agent

To be able to run the CI/CD pipeline, we need an Agent that is responsible for all the instructions. Due to student account limitations, we must run the Agent locally. Select *Project settings* from the bottom-left of the screen \rightarrow *Agent pools* \rightarrow *Add pool* and select Self-hosted for Pool type and name it local.

¢	nameLastNameGroupNum / p	project 1 / Settings / Agent pools	Add agent pool
P +	Project Settings	Agent pools	Agent pools are shared across an organization.
	General	Name	New Existing Pool type:
	Overview	Azure Pipelines Azure Pipelines	Self-hosted V
<u> </u>	Teams	Default	A pool of agents that you set up and manage on your own to run
8	Permissions	Azure Pipelines	jobs. Learn more.
1	Notifications		Name:
	Service hooks		local
4	Dashboards		Description (optional):
а.	Boards		
	Project configuration		Markdown supported.
	Team configuration		
	GitHub connections		Pipeline permissions: Grant access permission to all pipelines
	Pipelines		Grant access permission to an pipennes
	Agent pools		
	Parallel jobs		
	Settings		
	Test management		
	Release retention		
	Service connections		
	XAML build services		
	Repos		
_	Repositories		
٢	Artifacts		Create
\gg	Storage		

Figure 21. Agent pool

The list of agent pools should refresh, and a new item named local should appear. The Azure Pipeline Agent is used to:

- 1. Build the image for the code from GitHub, push it on Docker Registry
- 2. Get the image from Docker Registry and deploy the container on Docker on Azure

Download the Ubuntu VM Image linked

https://drive.google.com/drive/folders/1A5_L8WHCgsUutVbfwtl7ZMQg4uPPufjY?usp=sharing Install Virtual Box (<u>https://www.virtualbox.org/wiki/Downloads</u>) and open the VM Image downloaded. It has the Azure Pipeline Agent preinstalled and all you have to do is to configure the access credentials.

Note:

If you use the agent from the VM, login intro the VM and use the Ubuntu commands listed below. If you use a Windows machine, use the corresponding commands.

Before running the config command, we need an access token. Select the *icon highlighted* in Figure $22 \rightarrow Personal access tokens \rightarrow New Token$ and configure the token as presented in Figure 23. Note: you can increase the expiration date to your needs.

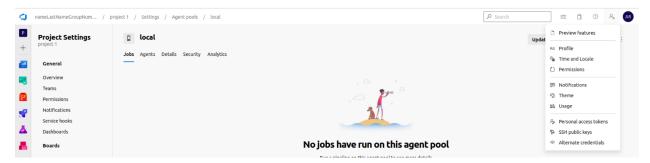


Figure 22. Personal Access Tokens *trebe highlight la imgt*

C Azure DevOps				Create a new personal access token		
User settings user	Personal Access Tokens These can be used instead of a password for applie + New Token ? Revoke ? Edit Revoke	cations like Git or can be passed in the authorization header to access REST APIs		Name agent-token		×
Account	Token name	Status	Organiz	Organization		~
RE Profile				nameLastNameGroupNumber		
So Time and Locale				Expiration (UTC) 30 days	10/14/2022	
O Permissions				30 days	10/14/2022	
Preferences INotifications Theme Bb Usage				Scopes Authorize the scope of access associated with this tok Scopes @ Full access Custom defined	en	
Security						
SSH public keys						
 Alternate credentials 						
 Authorizations 						
				Creats Cancel		

Figure 23. Personal Access Token Settings

Make sure you are saving the token displayed on screen, locally on a text file. If you forget the personal token, you need to regenerate the token or create a new one and change the value everywhere it was used.

Login into the VM and using your favorite command line interpreter (Bash/PowerShell), you must be within the agent's directory before trying to run the config command - Ubuntu, .\config.cmd - Windows).

The configuration is the following:

- 1. your organization server url → https://dev.azure.com/nameLastNameGroupNumber
- 2. enter authentication type (press enter for PAT) press enter
- 3. personal access token \rightarrow the personal access token generated in Figure 23
- 4. agent pool \rightarrow local
- 5. agent name \rightarrow agent
- 6. press enter

If you use a Windows machine only enter Y for run agent as Service. Y again, Enter, Enter



Figure 24. Agent Configuration

Prerequisites

1. Add user to docker group:

sudo usermod -aG docker \$USER

newgrp docker

2. Login Azure

docker login azure

Login with your azure credentials Verify if *dockerAccessToken.json exists in /home/\$USER/.azure*

To start the agent, execute the <u>\run.cmd</u> (Only if you haven't set the agent as a Service) -Windows or <u>./run.sh</u> - Ubuntu command. To install the service (Ubuntu only), run the following commands: sudo <u>./svc.sh install</u> and <u>sudo ./svc.sh start</u>. To visualize the agent's tasks run <u>sudo</u> <u>./svc.sh status</u>.

6
mes:/snoolbox/scripts
ĺ

Figure 25. Agent Status

To verify that all the configurations are right and the connection with the Azure DevOps platform is done, go to, *Project Settings* \rightarrow *Agent pools* \rightarrow *local* \rightarrow *Agents*. Under the name of the agent you can see his status.

nameLastNameGroupNum /	project 1 / Settings / Agent pools /	local		P Search	i≣ @ %
Project Settings	Jobs Agents Details Security	Analytics			Update all agents New agent
General Overview	Name	Last run	Current status	Agent version	Enabled
Teams	agent		Idle		On
Permissions	Online		Idle	2.210.0	On On
Notifications					
Service hooks					
Dashboards					
Boards					
Project configuration					
Team configuration					
GitHub connections					

Figure 26. Local Agent Status Visualization

4.1.4 Azure Cl

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

To create a pipeline, you must select *Pipelines* \rightarrow *New pipeline*. Because our code is present in a GitHub repository, we will continue by selecting *GitHub*.

CI/CD Docker

Azure DevOps nameLastName	GroupNur / project 1 / Pipelines 🛛 🗇 🗞 👧
P project 1 +	Azure DevOps proceeds in rollout of changes to permanently disable communication over TLS 1.0, TLS 1.1 and weak cipher suites of TLS 1.2. If your tools are dependent on legacy TLS for communication with Azure DevOps, please take necessary actions to enable TLS 1.2, as detailed in the blog.
Overview	Connect Select Configure Review
n Boards	New pipeline Where is your code?
😢 Repos	where is your code?
Pipelines	Azure Repos Git YAML Free private Git repositories, pull requests, and code search
i Pipelines	Btbucket Cloud VAML VAML Hotod by Atlassian
Environments Releases	GitHub VAML Home to the world's largest community of developers
Library Task groups	CILHub Enterprise Server WAMEL The self-adoted version of CilHub Enterprise
T Deployment groups	Other Git Any generic Git repository
📥 Test Plans	Subversion Centralized version control by Apache
Artifacts	

Figure 27. Pipeline Creation – GitHub

After you log in with your GitHub credentials, you should select your repository.

Azure DevOps nameLastNameC	iroupNum / project 1 / Pipelines 🗵 🗇 🕫 🙉
P project 1 +	Azure DevOps proceeds in rollout of changes to permanently disable communication over TLS 1.0, TLS 1.1 and weak cipher suites of TLS 1.2. If your tools are dependent on legacy TLS for communication with Azure DevOps, please take excessary actions to enable TLS 1.2, as detailed in the blog.
Cverview	✓ Connect Select Configure Review
n Boards	New pipeline
😢 Repos	Select a repository
Y Pipelines	
Pipelines	user/nameLastNameGroupNumberBackend private
A Environments	5m ago
🖉 Releases	O Showing the most recently used repositories where you are a collaborator. If you can't find a repository make sure you provide access.
II\ Library	You may also select a specific connection.

Figure 28. GitHub Available Repositories

You will be redirected to GitHub, scroll down to Repository access section, select Only selected repositories and press Approve and Install.

	Read and write access to checks, code, commit statuses, deployments, issues, and pull requests
Archives 22 Security log 23 Sponsorship log	Repository access
C Developer settings	Azure Pipelnes suggested installation on the following repositories. • All repositories This applies to all current and huma repositories. • Only select repositories • Select repositories • Select repositories • Selected 1 repositor
	Approve and install Cancel

Figure 29. GitHub Repository Access Section

To configure the pipeline, select Docker (Build and push an image to Azure Container Registry) and configure the container registry name with the one you created in the first step of this tutorial.

CI/CD Docker

DISTRIBUTED SYSTEMS

Azure DevOps nameLastName	eGroupNum / project 1 / Pipelines	
P project 1 +	✓ Connect ✓ Select Configure Review	Docker Build and push an image to Azure Container Registry
Cverview	New pipeline	Container registry containerregistrynamelastnamegroupnumber
n Boards	Configure your pipeline	Image Name
😢 Repos	Docker sour Build a Docker image	namelastnamegroupnumberbackend
Pipelines	Docker Source Build and push an image to Azure Container Registry	Dockerfile \$(8uild.SourcesDirectory)/Dockerfile
Pipelines Environments	Bulid and push image to Azure Kubernetes Service Bulid and push image to Azure Container Registry; Deploy to Azure Kubernetes Service	
∯ Releases	Beploy to Kubernetes - Review app with Azure DevSpaces Build and push image to Azure Container Registry; Deploy to Azure Kuberentes Services and setup Review App with Azure DevSpaces	
Library Task groups	Starter pipeline Start with a minimal pipeline that you can customize to build and deploy your code.	
T Deployment groups	Existing Azure Pipelines YAML file Select an Azure Pipelines YAML file in any branch of the repository.	
Lest Plans	INET Core Function App to Windows on Azure Build a NET Core function app and deploy it to Azure as a Windows function App.	
Artifacts	Build and run tests for .NET Desktop or Windows classic desktop solutions.	
	Padroid Build your Android project with Gradie.	
	Not Build your Java projects and run tests with Apache Ant.	
	Build and test ASPNET projects.	
	Build and test ASPNET Core projects targeting. NET Core.	
Project settings	ASP.NET Core (NET Framework) Build and test ASP.NET Core projects targeting the full .NET Framework.	Back Validate and conf

Figure 30. Configure your pipeline - Docker

You must replace the code after the stages tag with the X proprieties found on GitHub. Select save and run.

Azure DevOps nameLastNameG	roupNum / project 1 / Pipelines	Save and run
P project 1 +	✓ Connect ✓ Select ✓ Configure Review	Save and run Saving will commit azure-pipelines.yml to the repository.
	· connect · select · connigore	Commit message
Cverview	New pipeline	Set up CI with Azure Pipelines
🔣 Boards	Review your pipeline YAML	Optional extended description
😢 Repos	O user/nameLastNameGroupNumberBackend / azure-pipelinesyml * =#	Add an optional description
Y Pipelines	1 # Docker 2 # Build and push an image to Azure Container Registry	
🖬 Pipelines	<pre>3 # https://docs.microsoft.com/azure/devops/pipelines/languages/docker</pre>	
Environments	5 trigger: 6 - main	 Commit directly to the main branch Create a new branch for this commit
🔊 Releases	7 7 resources:	
III\ Library	9 - repo: self	
📟 Task groups	11 variables: 12 # Container registry service connection established during pipeline creation	
T Deployment groups	13 dockerRegistryServiceConnection: '' #uuid - should be autocompleted 14 imageRepository: 'namelastnamegroupnumberbackend'	
👗 Test Plans	15 containerRegistry: 'containerregistrynamelastnamegroupnumber.azurecr.io' 16 dockerfilePath: '\$(Build.SourceSDirectory)/Dockerfile' 17 - 'cag: '\$(Build.Buildd)'	
Artifacts	18	
Artifacts	19 # Agent VM image name 20 vmImageName: 'ubuntu-latest'	
	21	
	22 stages: 23 - stage: Build	
	24 displayMane: Build stage	
	25 jobs:	
	26 job: CodeCoverage	
	27 ····displayName: Code Coverage	
	28pool: 29name:local	
	30 ·····steps:	
	Settings	
	31 task: Maven@3	
	32inputs:	
	33 ·····mavenPomFile:·'pom.xml'	
	34 ·······publishJUnitResults: true	Save and run
Ø Project settings	<pre>35 control testResultsFiles: '*'/surefire-reports/TEST-*.xml' 36 control codeCoverageToolOption: 'JaCoCo'</pre>	

Figure 31. Pipeline Stages Setup

After the job is finished, there are 3 main tabs available. On the summary tab (Figure 32), we can verify that the project builds correctly. On the Tests tab (Figure 33) we can check that the tests run and how many are successful and on the Code Coverage tab, we can see the code style issues (Figure 34).

Azure DevOps nameLastNameGro	pupNum / project 1 / Pipelines / user.nameLastNameGroup	/ 20220914.1			■ ⁽¹⁾ (2) (2) (1)
P project 1 +	#20220914.1 • Set up CI with Azure Pipelines				Runnew
Cverview	user.nameLastivameGroupivumoerbackend				
n Boards	 This run is being retained as one of 3 recent runs by pipeline. 				View retention leases
😢 Repos	Summary Tests Code Coverage Extensions				
Pipelines					
🕍 Pipelines	Triggered by user				View change
L Environments	Repository and version O user/nameLastNameGroupNumberBackend	Time started and elapsed	Related	Tests and coverage O 100% passed	
∯ Releases	P main	© 1m 34s	2 published	■ 52.94% covered	
II\ Library					
Task groups Deployment groups	Errors 1 Warnings 1				
Test Plans	 unknown shorthand flag: 'f' in -f Build - Build and push an image to container registry 				
Artifacts					
	Jobs		Status Di	uration	
	Code Coverage) 50s	
	 Build) 285	
Project settings					

Figure 32. CI Pipeline Summary

Azure DevOps nameLastNameGro	pupNum / project 1 / Pipelines / user.nameLastNameGroup / 20220914.1	P Search i≡ □ ⑦ P₀ AR
P project 1 +	#20220914.1 • Set up CI with Azure Pipelines dusenameLastNameGroupNumberBackend	Runnew
Cverview		
誤 Boards	① This run is being retained as one of 3 recent runs by pipeline.	View retention leases
😢 Repos	Summary Tests Code Coverage Extensions	
Pipelines	_	
👑 Pipelines	Summary 3 Run(s) Completed (3 Passed, 0 Failed)	^
Environments	3 kun(s) Completed (3 Passed, 0 Palled)	
🔊 Releases	24 Passed 100% 16s 199ms 0	
II\ Library	Total tests 0 ● Failed Pass percentage Run duration ① Tests not reported +24 0 ● Others ↑ 100% ↑ +16s 199ms ►	
E Task groups		
Deployment groups	🗅 Bug 🗸 👒 Link	🕾 Test run 🗸 🧷 Column Options 🛛 🕇
Lest Plans	√ Filter by test or run name	Tags $$ Test file $$ Owner $$ Aborted (+1) $$
Artifacts		

Figure 33. CI Pipeline Tests

CI/CD Docker

Azure DevOps nameLastNameGro	SupNum / project 1 / Pipelines / user.nameLastNameGroup / 20220914.1	i≡ 🖞 ⑦ % 🗛
P project 1 +	#20220914.1 • Set up CI with Azure Pipelines #usernameLastNameGroupNumberBackend	Runnew
Cverview		
i Boards	O This run is being retained as one of 3 recent runs by pipeline.	View retention leases
P Repos	Summary Tests Code Coverage Extensions	
Pipelines	Sessionsds-2020	
uiu Pipelines		
L Environments	ds-2020	
🖉 Releases	Element Missed Instructions Cov. Missed Branches Cov. Missed Cxty Missed Lines Missed Methods Missed Classes	
II\ Library	ro.tuc.ds2020.controllers.handlers.exceptions.model 33% n/a 19 27 34 50 19 27 5 6 ro.tuc.ds2020.controllers.handlers 53% 100%2 5 16 30 2 4 0 1	
📟 Task groups	ro.tuc.ds2020.dtos 58% 50% 14 26 18 47 9 21 0 2 ro.tuc.ds2020.controllers 42% n/a 3 6 5 11 3 6 0 2	
Deployment groups	ro.tuc.ds2020.services 73% 50% 1 6 2 17 0 5 0 1 ro.tuc.ds2020.entities 55% n/a 5 10 9 19 5 10 0 1	
Lans Test Plans	ro.tuc.ds2020 Image: 15% n/a 2 3 4 5 2 3 0 1 ro.tuc.ds2020.dtos.builders Image: 100% n/a 2 0 4 0 2 0 1 ro.tuc.ds2020.dtos.builders Image: 100% 100% 4 0 2 0 1	
Artifacts	Total 296 of 630 53% 6 of 16 62% 46 89 88 187 40 81 5 16 Created with <u>JaCoCo</u> 0.8.7.202105040129	
Project settings		

Figure 34. CI Pipeline Code Coverage

4.1.5 Azure CD

The continuous deployment aims at delivering as fast as possible the new features added through your code to the deployment servers. If all the previous steps (build, test, and check style) are successful, it will proceed with the deployment of your application on a server. 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.

To create a pipeline, you must select *Pipelines* \rightarrow *Releases* \rightarrow *New pipeline*. Change the stage name to Development.

CI/CD Docker

Azure DevOps nameLastNameG	roupNum / project 1 / Pipelines / Releases	Search III 🗇 🥱 🗛
P project 1 +	All pipelines > 7 New release pipeline	🔚 Save 🚀 Create release \cdots
Overview	Pipeline Tasks - Variables Retention Options History	
n Boards		Stage
😢 Repos	Artifacts + Add Stages + Add \	Development
Pipelines		Properties Name and owners of the stage
utar Pipelines	+ Add an & Development R 1 job.0 task R	Stage name Development
L Environments	A 1 job. 0 task	Stage owner
🖉 Releases	Schedule	user X
II\ Library	Schedule not set	
 Task groups Deployment groups 		
_		
Test Plans		
Artifacts		
Project settings		

Figure 35. Create a CD Pipeline

Before starting with the tasks within the pipeline, a connection between the CI and CD must be set up. This is done by pressing on *Add an artifact* \rightarrow select Source type as Build, select your current Azure DevOps project, the build pipeline and the default version.

Azure DevOps nameLastNameG	GroupNum / project 1 / Pipelines / Releases	×
P project 1 +	All pipelines > 7 New release pipeline	Add an artifact
Cverview	Pipeline Tasks V Variables Retention Options History	Source type
noards		
😢 Repos	Artifacts + Add Stages + Add \vee	✓ Build Azure Rep GitHub TFVC
Pipelines		5 more artifact types 🗸 Project * 🕜
👑 Pipelines	Add an & Development R Ljob. 0 task	project 1
Environments	artifact	Source (build pipeline) *
\$₽ Releases		ranceaaa.nameLastNameGroupNumberBackend ~
II\ Library	Schedule not set	Default version *
📟 Task groups		Latest \checkmark
T Deployment groups		Source alias * ()
👗 Test Plans		_ranceaaa.nameLastNameGroupNumberBackend
Artifacts		① The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of uest.nameLatManeGroupMumbeTBackend published the following artifacts: Code Analysis Results, Code Coverage Report_4.
		Add
Project settings		

Figure 36. Add an Artifact - Build

After the artifact has been added, press the lightning icon to creates a release every time a new built is available.

CI/CD Docker

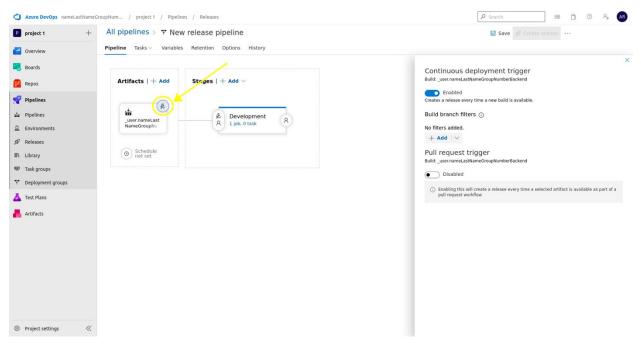


Figure 37. Trigger the CD when a Build was successfully added

Before the CD pipeline starts, the Docker Compose file needs to be available. We will do this by adding another artifact. Select Source type as GitHub, and select the Source (repository) to your repository where the Docker Compose file is found. Select the Default branch and the Default Version to Latest from the default branch.

Azure DevOps nameLastNameG	roupNum / project 1 / Pipelines / Releases	×
P project 1 +	All pipelines > * New release pipeline	Add an artifact
Overview	Pipeline Tasks V Variables Retention Options History	Source type
Boards Repos	Artifacts + Add Stages + Add ~	Build Azure Rep
Pipelines		5 more artifact types \sim
Pipeunes	(k)	Service * Manage 🗅
🛍 Pipelines	usernameLast & Development	github.com_ranceaaa \checkmark 🔿 + New
🚊 Environments	NameGroupNu A 1 job. 0 task	Source (repository) * ()
"\$∕7 Releases		ranceaaa/cd-azure
II\ Library	0	Default branch * 🕡
🐨 Task groups	مع Add an artifact	main ~
T Deployment groups		Default version *
👗 Test Plans	Schedule not set	Latest from the default branch \sim
Artifacts	on not set	Checkout submodules 🛈
		Checkout files from LFS ①
		Shallow fetch depth
		Source alias * 🕥
		_ranceaaa_cd-azure
		Add
		_
Project settings		

Figure 38. Add an Artifact with the Docker Compose file

To configure the pipeline to work with our local agent. Tasks \rightarrow Agent job \rightarrow Agent pool \rightarrow local.

CI/CD Docker

Azure DevOps nameLastNameG	roupNum / project 1 / Pipelines / Releases				Ő ()	₽₀ A R
P project 1 +	All pipelines > T New release pipeline			层 Save 🚀 Create release		
Cverview	Pipeline Tasks Variables Retention Options History					
n Boards	Development Deployment process		Agent job 🕤			Remove
😢 Repos	Agent job	+	Display name *			
字 Pipelines	龗 Run on agent		Agent job			
📩 Pipelines			Agent selection A			
Environments			Agent pool 🕥 Pool information Manage 🛤			
🔊 Releases			Hosted Windows 2019 with VS2019			~ O
III\ Library			Hosted Azure Pipelines			
📟 Task groups			A Hosted Windows 2019 with VS2019			
T Deployment groups			Private			
👗 Test Plans			P Default (No agents)			
Artifacts			d ^g local			
Artifacts			Parallelism ()			
			None Multi-configuration Multi-agent			
			Timeout * 🕕			
			0			
			Job cancel timeout * 🕡			
			1			
			Artifact download A			
			🛔 _ranceaaa.nameLastNameGroupNumberBackend Latest	~	Selected a	all artifacts
$^{\odot}$ Project settings \ll			O_ranceaaa_cd-azure Specify at the time of	f release creation \sim	Selected a	all artifacts

Figure 39. Agent Job Configuration

To add a new task, press the + *button* next to Agent job, and search for Docker Compose. Add the Docker Compose task, and select you Azure subscription.

The next configuration must to be done, to work with the Docker Compose file we are pulling from our GitHub repository:

1. Change the Docker Compose file path to → \$(System.DefaultWorkingDirectory)/_usernameGithub_cd-azure/docker-compose.yml

2. Complete on Action field \rightarrow Run a Docker Compose command

3. Command \rightarrow up –d

4. On Advanced Options add the Working Directory → \$(System.DefaultWorkingDirectory)/_usernameGithub_cd-azure/

5. Press Save

CI/CD Docker

Azure DevOps nameLastNameC	roupNum / project 1 / Pipelines / Releases	P Search I≣ @ P₀ AP
P project 1 +	All pipelines > 7 New release pipeline	🗟 Save 🖉 Create release \cdots
Cverview	Pipeline Tasks	Run a Docker Compose command
n Boards	Development	Container Registry Type * 🕜
😢 Repos	Agent job	Azure Container Registry
Pipelines	Run a Docker Compose command	Azure subscription 🕥 Manage 🗈
👑 Pipelines	Docker Compose	Azure for Students (1ac9c6c2-4e61-4504-a6ea-e8b59fcac16c) V O Scoped to subscription 'Azure for Students'
🚊 Environments		Azure Container Registry
🔊 Releases		containerregistrynamelastnamegroupnumber V 🕐
II\ Library		Docker Compose File * 🕠
📟 Task groups		\$(System. <u>DefaultWorkingDirectory</u>)/_user_ <u>cd-azure</u> //docker-compose. <u>yml</u>
Deployment groups		Additional Docker Compose Files
📥 Test Plans		
Artifacts		Environment Variables
		Project Name 🕥
		\$(Build.Repository.Name)
		Qualify Image Names ①
		Action * 🕜 Run a Docker Compose command
		Command • ①
Project settings		up -d

Figure 40. Agent job - Docker Compose Configuration 1

🗘 Azure DevOps nameLastNameGroupNum / project 1 / Pipelines / Releases					_ i≡	9 Po	AR
P project 1 +	All pipelines > 🕆 New release pipeline			层 Save 🚀 Create	release ····		
Overview	Pipeline Tasks - Variables Retention Options History						
	Development Deployment process						
😢 Repos	Agent job I Run on sgent	+	Project Name				
Pipelines	Run a Docker Compose command	⊘	\$(Build.Repository.Name)				
📩 Pipelines	Docxer Compose		Qualify Image Names ()				
Environments			Action * ①				
\$₽ Releases			Run a Docker Compose command				\sim
IIN Library			Command * 🕡				
🐨 Task groups			up -d				
T Deployment groups			Arguments 🕥				
👗 Test Plans							
Artifacts			Advanced Options A				
Pictures			Docker Host Service Connection () Manage				
					~	О + N	lew
			No-op if no Docker Compose File ()				
			Require Additional Docker Compose Files Working Directory				
			\$(System. <u>DefaultWorkingDirectory</u>)/_user_ <u>cd-azure</u> /				
			Docker Compose executable Path				
Project settings			Control Options V				
127 Project settings ((Output Variables 🗸				

Figure 41. Agent job - Docker Compose Configuration 2

The second task that must be added is Bash. This contains CLI commands responsible for deploying the applications.

CI/CD Docker

DISTRIBUTED SYSTEMS

Azure Devops nameLastivam	eGroupNum / project 1 / Pipelines / Releases		P Search I≣ Ĉì ⑦ P₀ AR	
P project 1 +	All pipelines > 🐨 New release pipeline	🗟 Save 💋 Create release		
Overview	Pipeline Tasks Variables Retention Options History			
😓 Boards	Development Deployment process	Add tasks 🛛 💍 Refresh	✓ bash ×	
😢 Repos	Agent job	+		
Pipelines	Run a Docker Compose command	Bash Run a Bash script on macOS, Linux, or Windows		
🛓 Pipelines	Docker Compose	Command line		
Environments		Run a command line script using Bash on Linux and macO Windows	JS and cmd.exe on	
🕼 Releases		Marketplace 🔨		
II\ Library		Script Retryer tool		
Task groups		That task can be used to retry several times with some dela Inline script supported. Supported languages: bash, powe	ay specified script. ershell, python. It	
 Deployment groups 		works fine on Unix and Windows agents.		
👗 Test Plans		Applitools Eyes Integration Displays the Applitools Dashboard in the build summary page	ge.	
Artifacts		Use a service connection with scripts instead of past variables!	iting secrets into	
Project settings				

Figure 42. Bash Script

Select the Type to Inline and add the following script:

1. Move to the _usernameGithub_cd-azure folder.

cd usernameGithub_cd;

2. After the first task is finished (docker compose up), the running containers should be stopped.

docker compose down; docker-compose down;

3. Login into the Azure Container Registry:

docker login containerregistrynamelastnamegroupnumber.azurecr.io -u containerregistrynamelastnamegroupnumber -p sTsnMgSQa0dx5LD=qWqEtcIv6zmJSmMP;

The credentials can be found in: Home \rightarrow Resource Groups \rightarrow nameLastNameGroupNumber \rightarrow containerregistrynamegroupnumber \rightarrow Access keys and by enabling the Admin user. Replace the Login server, Username and password with your own details.

4. Create a Docker aci context on your resource group

docker context create aci acicontext --resource-group nameLastNameGroupNumber;

5. Use the context created above.

docker context use acicontext;

6. Start the containers

docker compose up;

All pipelines > ** New release pipeline		🗟 Save 💋 Create release 🗮 View releases \cdots
Pipeline Tasks V Variables Retention Options History		
Development Deployment process		Bash 🔘 🔯 View YAML 🗓 Remove
Agent job ﷺ Run on agent	+	Task version 3. ~
Run a Docker Compose command Docker Compose		Display name * Bash Script
#1 Bash Script Bash	❷ 🛔	Type ① File Path
		Script * cd_user_cd-azure; docker_compose down; docker_context registrynamelastnamegroupnumb-p sfsmMgSGaddx5LD-qWgEtchVSrnJSm; docker context registrynamelastnamegroupnumber; docker context registrynamelastnamegroupnumb-p; docker context use acicontext; docker ops; Z Advanced \rightarrow Environment Variables \rightarrow Output Variables \rightarrow

Figure 43. Bash Inline Script

After the Development Release is finished and Saved, press Create release and press Deploy.

Azure DevOps nameLastName	GroupNum / project 1 / Pipelines / Releases / New release pipeline / Release-1	х
P project 1 +	*** New release pipeline > Release-1 > Development ~ O Not deployed	Development
Overview	← Pipeline Tasks Variables Logs Tests	Deploy release
E Boards	O Deployment of Release-1 to Development has not yet started.	Overview Commits Work Items
📔 Repos		To be deployed (Deploying for the first time) Release-1
Pipelines		Artifacts
👪 Pipelines		C ranceaaa_cd-azure / 72ec76cd4
🚊 Environments		ranceaaa.nameLastNameGro / 20220914.1
🔊 Releases		1 . House
IIN Library		Comment
🐨 Task groups		
*** Deployment groups		
👗 Test Plans		Deploy Cancel
Artifacts		

Figure 44. Create Release

When the next push occurs in your GitHub repository, the build pipeline will start, and if it is completed successfully, the CD pipeline starts. The output should be like Figure 45.

CI/CD Docker

DISTRIBUTED SYSTEMS

Azure DevOps nameLastName@	GroupNum / project 1 / Pipelines /	Releases / New release pipeline / Release-1	i≣ @ 0 % AR
P project 1 +	↑ New release pipeline > Release	se-1 > Development ~ 🕑 Succeeded	
Cverview	\leftarrow Pipeline Tasks Variables Logs	Tests 🔷 Deploy 🚫 Cancel 🕐 Refresh 🞍 Download all logs 🖌 Edit 🗸 \cdots	2
Noards	Deployment attempt #3 V	Agent Job Pool: local - Agent: agent	Started: 9/14/2022, 11:03:11 AM 1m 3s
😢 Repos	Agent job Succeeded	Initialize job - succeeded	<15
Pipelines		Download Artifacts - succeeded	15
Pipelines		Download artifactranceaaa.nameLastNameGroupNumberBackend - Code Analysis Results - succeeded	25
🚊 Environments		Download artifactranceaaa.nameLastNameGroupNumberBackend - Code Coverage Report_4 - succeeded	45
\$₽ Releases		Run a Docker Compose command - succeeded	1s
II\ Library		Bash Script · succeeded	525
Safe Task groups		Finalize Job - succeeded	<1s
 Deployment groups 			
📥 Test Plans			
Artifacts			
Project settings <			

Figure 45. Complete CD Pipeline Logs

If Bash Script log is selected, the IP of the server can be found, and tested by copying it in a browser.

project 1 +	*se New Bash Script			1 Previous task	\downarrow Next task \mid $ imes$
Overview	← Pipeline 1 2022-09-14708:03:22.63490927 ##[section]Starting: Bash Script 2 2022-09-14708:03:22.63490532				
Boards	Deploymer 2022-09-14T08:03:22.63495:22 Description Hum a Bash script Succeded 5 2022-09-14T08:03:22.6349729Z Version :3.201.1 2022-09-14T08:03:22.6349972Z Version :3.201.1 :3.201.1				li f Be
Repos		soft.com/azure/devops/pipelines/tasks/utility/bash			
Pipelines	9 2022/09-14108-03/22.74880402 weierlafting Script. 2022-09-14108:03:22.74880402 —	er-server/Desktop/myagent/_work/_temp/2afb0276-8e32-	4db9-8739-3d0796529e02.sh		
Pipelines	 2022-09-14T08:03:22.8433679Z See 'dockerhelp' 2022-09-14T08:03:22.934993Z WARNING' Usingpassword via the 2022-09-14T08:05:22.844283Z WARNING' Your password will be the 2022-09-14T08:05:22.84428XG Viour password will be the 	CLI is insecure. Usepassword-stdin.	ia ison		
Environments	16 2022-09-14T08:03:23.8043336Z Configure a credential helper to 17 2022-09-14T08:03:23.8044183Z https://docs.docker.com/engine/re	remove this warning. See	<u> </u>		
Releases	 2022-09-14708:03:23.80443752 2022-09-14708:03:23.80443792 2022-09-14708:03:24.49995472 Using only available subscription 		fcac16c)		
Library Task groups	21 2022-09-14T08:03:24.64489482 Successfully created aci context 22 2022-09-14T08:03:24.73740902 NAME TYPE 23 2022-09-14T08:03:24.73744302 acicontext aci		DOCKER ENDPOINT	KUBERNETES ENDPOINT	ORCHESTRAT
Deployment groups	24 2022-09-14T08:03:24.7379692Z acitest aci 25 2022-09-14T08:03:24.7380303Z acitest2 aci	b0d87868-acld-89c3-bdla-ebf85dld3dbf@eastus myapp-rg-alex@eastus			
Test Plans	26 2022-09-14768:03:24.73060352 acitest3 aci 27 2022-09-14768:03:24.73031932 acitest4 aci 28 2022-09-14768:03:24.73813132 default * moby 29 2022-09-14768:03:24.7382352 tkk aci	myapp-rg-alex@eastus myapp-rg-alex@eastus Current DOCKER_HOST based configuration myapp-rg-alex@eastus	unix:///var/run/docker.sock		swarm
Artifacts	2022-09-14T08:03:24.7863422 acicontext 31 2022-09-14T08:03:24.850790Z Group work Creating 2 2022-09-14T08:03:24.82710372 Group work Created	manh			
	 2022-09-14T08:03:28.2271798Z demo-db Creating 2022-09-14T08:03:28.2272138Z demo-be Creating 				
	35 2022-09-14768:04:14.13304792 deno-db Created 36 2022-09-14768:04:14.13339812 deno-be Created 37 2022-09-14768:04:14.74175302 CONTAINER ID IMAGE 38 2022-09-14768:04:14.74213552 vork deno-db mycontainerre	COMMAN	D STATUS Running	PORTS 20.79.196.104:5432-	\$\$432/tro
	39 2022-09-14T08:04:14.7430728Z work demo-be mycontainerre 40 2022-09-14T08:04:14.7524880Z ##[section]Finishing: Bash Script	gistryalextest123.azurecr.io/ranceaaasddemo	Running	20.79.196.104:8080-	
	41				
Project settings 🛛 🐇					

Figure 46. IP of the Deployed Containers

To visualize the deployed application and the containers, go to $Home \rightarrow Resource$ Groups \rightarrow nameLastNameGroupNumber \rightarrow work \rightarrow Containers.

CI/CD Docker

	P Search resources, serv	ices, and docs (G+/)	E & 0	© <i>© R</i>	user@student.utcluj.ro
Home > Resource groups > nameLastNameGroupNumber >					
work					×
	🕞 Start 🦿 Restart 🔲 Stop 📋 Delete 💍 Refresh				
🔮 Overview	↑ Essentials				JSON View
Activity log	Resource group (move) : nameLastNameGroupNumber		OS type : Linux		
Access control (IAM)	Status : Running		IP address (Public) : 20.218.208.131		
🔷 Tags	Location : Germany West Central		FQDN :		
Settings	Subscription (move) : Azure for Students		Container count : 3		
Containers	Subscription ID :				
% Identity	Tags (cdl) : docker-compose-application : docker-compose-application				
Properties					
C Locks	CPU 🔗	Memory 5	Network bytes received		
LOCKS			908		
Monitoring	100	120MB	816		
iii Metrics	80	100MB	708 MM		
Alerts	60	80MB	508		
Automation	40	60MB	408		
🚑 Tasks (preview)	20	40MB	208		
Export template		08	08		
	8:30 AM 8:45 AM 9 AM UTC+03:00	8:30 AM 8:45 AM 9 AM UTC+03:00	8:30 AM 8:45 AM 9 AM UTC+03:00		
Support + troubleshooting	CPU Usage (Avg) work	Memory Usage (Avg) work	Network Bytes Received Per Second (Avg) work		
R New Support Request	2.47	116.57 MB	67.13 s		
	Network bytes transmitted				

Figure 47. Deployed Application Dashboard

5. Further development

- > Apply the configuration from section 4 to deploy the React Application on Azure.
- Modify the host.js file by referencing the public IP address of the Spring App container.
- > Deploy the application resulted from Assignment 1 on Azure cloud.

References

[1] https://gitlab.com/ds_20201/spring-demo

- [2] https://www.baeldung.com/spring-boot-docker-images
- [3] https://gitlab.com/ds_20201/react-demo
- [4] https://docs.gitlab.com/ee/user/project/deploy_tokens/
- [5] https://developer.okta.com/blog/2020/06/24/heroku-docker-react
- [6] https://docs.docker.com/install/linux/docker-ce/ubuntu/
- [7] https://www.computerhope.com/jargon/u/user-space.htm
- [8] https://en.wikipedia.org/wiki/Hypervisor
- [9] https://devopscon.io/blog/docker/docker-vs-virtual-machine-where-are-the-differences/
- [10] https://www.docker.com/blog/vm-or-containers/
- [11] https://www.sciencedirect.com/topics/computer-science/hypervisors
- [12] https://docs.docker.com/registry/
- [13] https://docs.docker.com/develop/develop-images/baseimages/
- [14] https://en.wikipedia.org/wiki/Virtual_machine