

DevOps, Docker and Gitlab-CI

Part 5: Project

Version 1.2.0 (2023-03-20)



-- Cyril zarak Duval, root CRI/ACU 2020

About the project

Assignments ...



IDVOC project

- Will be started together
- Will make you write Dockerfiles for existing python applications
 - ◆ Learn more about Dockerfile
 - Learn about the syntax
 - Learn about the directives
 - ◆ Interact with docker (images, containers, ...)
 - ◆ Learn to convert a project to docker
 - ◆ A must known for the future



IDVOC project

- Will make you write a docker-compose.yml
 - ◆ Using already existing docker image
 - ◆ Using your newly built images
 - ◆ Learn about docker-compose
 - The docker-compose.yml file
 - The CLI
- Will make you interact with containers



IDVOC project

- Will make you write a `.gitlab-ci.yml` file
- Create a basic CI with multiple jobs
- Work with Gitlab

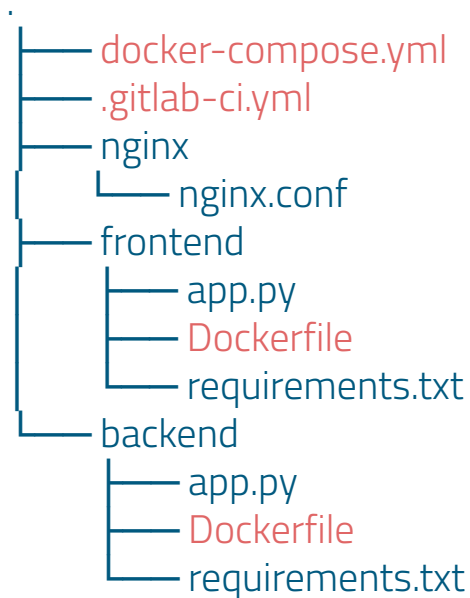


IDVOC project

- Project must be created on gitlab.cri.epita.fr
- Individual (must be private, otherwise considered as cheating)
- Project must be called IDVOC-2023
 - ◆ If it's not called IDVOC-2023, you won't get a grade
- I must be added as a Maintainer of the project
 - ◆ [@cyril](#) on gitlab.cri.epita.fr
 - ◆ You can do it on
https://gitlab.cri.epita.fr/<your_login>/IDVOC-2023/-/project_members
- The deadline will be for the **1st of june, 23h59**

IDVOC project

→ Expected repo architecture:



IDVOC project

- 3 steps
 - ◆ And advanced levels for steps
- Step 2 needs step 1, but step 3 is standalone
- Steps 1, 2 and 3 will grant you the most points
- Steps 1.5, 2.5 and 3.5 will grant you the rest of the points
- Perfect steps 1, 2 and 3 will give you a decent-ish grade
- Adding perfect steps 1.5, 2.5 and 3.5 will give you more than 20/20. Pick some elements in those steps to implement
 - ◆ Not every elements in steps 1.5, 2.5 and 3.5 are the same difficulty and length. Be wise !

Before getting started

Pay close attention to these details



IDVOC project – docker hub limits

- Docker hub has limits
- 300+ students downloading on the hub from the EPITA IP address will trigger this limit and will ratelimit EPITA
- To avoid this, we will be using another
- Each image that doesn't specify explicitly a registry **MUST** use this proxy
- This proxy is simply a proxy+cache on docker hub
- The proxy is registry.zarak.fr/library/

IDVOC project – docker hub limits

- Each image that doesn't specify explicitly a registry **MUST** use this proxy
- The proxy is `registry.zarak.fr/library/`
- Examples:
 - ◆ ~~`docker run busybox ->`~~
`docker run registry.zarak.fr/library/busybox`
 - ◆ ~~`docker run grafana/loki:main ->`~~
`docker run registry.zarak.fr/grafana/loki:main`
 - ◆ ~~`FROM busybox:glibc ->`~~
`FROM registry.zarak.fr/library/busybox:glibc`

IDVOC evaluation

- Your project will be evaluated in our environment, by picking the files in **red** from slide 7
- You can create other files in your repo, but they will be ignored
- Dementors and results will be provided as an issue on your project



Step 1

The basics



IDVOC project – Step 1

- The first step of the project is to write 2 Dockerfiles for our application
- The application is made of 2 app, 2 services: frontend & backend
- The 2 apps are available on <https://gitlab.cri.epita.fr/cyril/IDVOC-public>
- Those are python3 applications
 - ◆ The needed libs are in requirements.txt
 - ◆ You can install them with `pip install -r requirements.txt`
- A docker run <newly built image> shall start the application



IDVOC project – Step 1

- As it happens often, you're not the one who wrote the app
- You still have to dockerize it without knowing how it works
 - ◆ Or how python3 works
 - Or flask
 - Or python dependencies
- It's part of your job (and assignment) to figure it out
 - ◆ (some help tho)
 - ◆ (it's just some guidance)

IDVOC project – Step 1

- Frontend should be listening on port 5000
- Backend should be listening on port 6000
- It is part of the assignment to figure out how the app work and how to test if your image works



Step 1.5

Advanced, you may want to jump to step 2



IDVOC project – Step 1.5

- All docker images aren't good docker images
- Let's make yours a good one
- frontend and backend docker images are similar: find a way to reuse most layers
- Find a way to not redownload the dependencies if the app changes
- Don't run the app as root ! Find a way to run it as another unprivileged user
- Knowing who the author/maintainer of an image is great. Find a way to expose this information



IDVOC project – Step 1.5

- Find the most suited base image for this image
 - ◆ It shall be small, up-to-date, well known (and maintained) and suits the project
- Install bash in the image, for debugging purposes
 - ◆ But limit the number of layers
- Indicate the port exposed by default
- Figure out the best syntax for the CMD/ENTRYPOINT directive

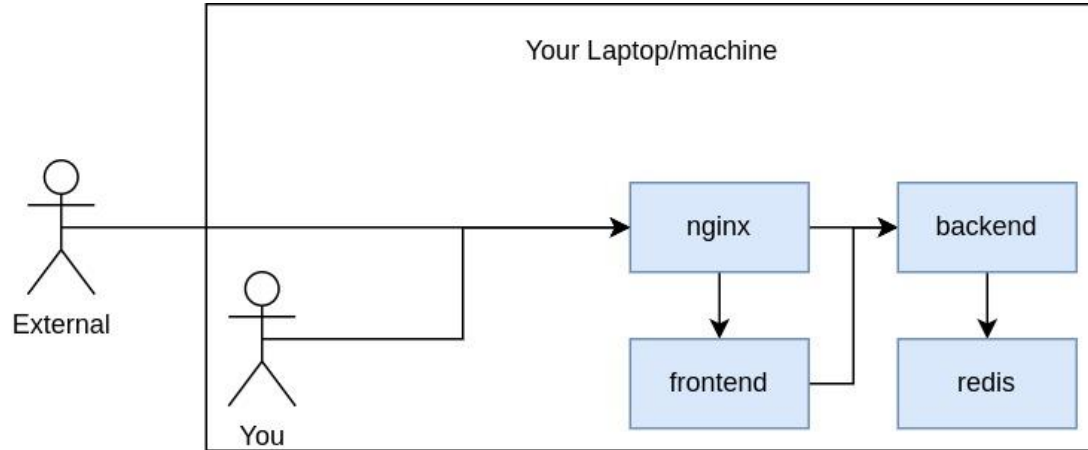
Step 2

You need step 1



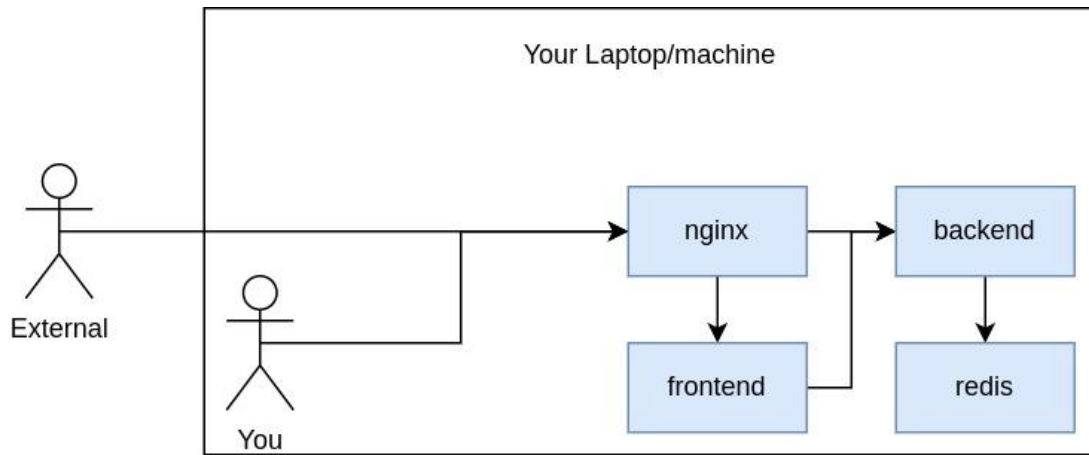
IDVOC project – Step 2

- The next step is to build this application architecture with docker-compose
- The webapp architecture is the following



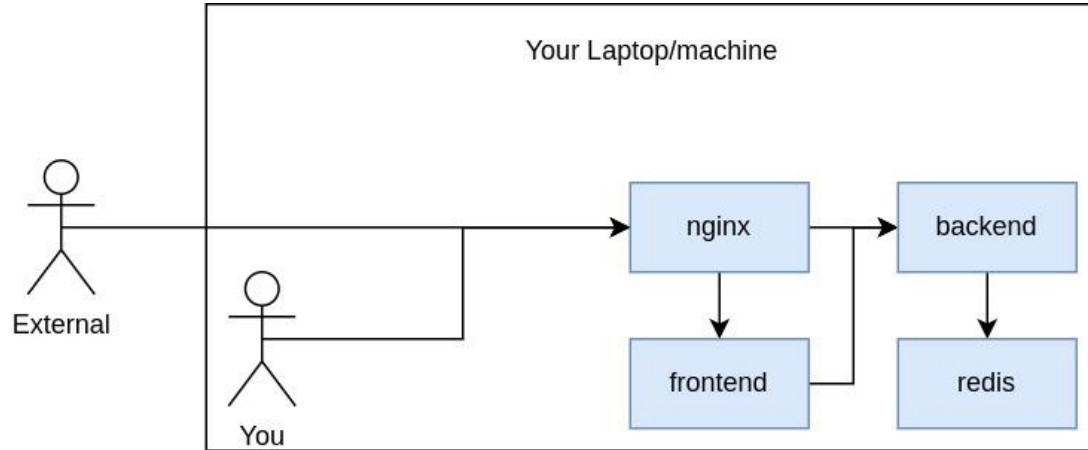
IDVOC project – Step 2

- Redis must not be directly reachable from the host machine
- Frontend and backend
- The containers must be named as shown in the schema



IDVOC project – Step 2

- Figure out what docker image to use for nginx and redis
- Frontend and backend are obviously your 2 images built in step 1



IDVOC project – Step 2

- Configuration for nginx is provided
 - ◆ It may be overridden by the testsuite
- Figure out how to provide nginx this configuration file
- Redis doesn't need configuration
- As always, find the best images for nginx and redis
 - ◆ It's better if it's official, maintained, up to date
- Find the best tag for nginx and redis
 - ◆ Avoid latest, we want the webapp to be reproducible



IDVOC project – Step 2

- Expose ports for nginx, to be able to reach it on HTTP and HTTPS ports
 - ◆ Don't worry about HTTPS on your local setup, test only if you can connect on HTTP
- Frontend and backend must not be reachable externally directly: all connections must go through nginx
 - ◆ That's also how you will allow the "client" to connect to your webapp

Step 2.5

Advanced, you may want to jump to step 3



IDVOC project – Step 2.5

- Let's make our docker-compose better
- Use networks to isolate nginx, frontend and redis
- Figure a way to make the containers crash resilient
 - ◆ And make them start on host machine startup also
 - No need to look outside of docker-compose.yml for this
- Write good YAML
- Nginx config file shall not be edited by nginx container. Find a way to enforce this rule

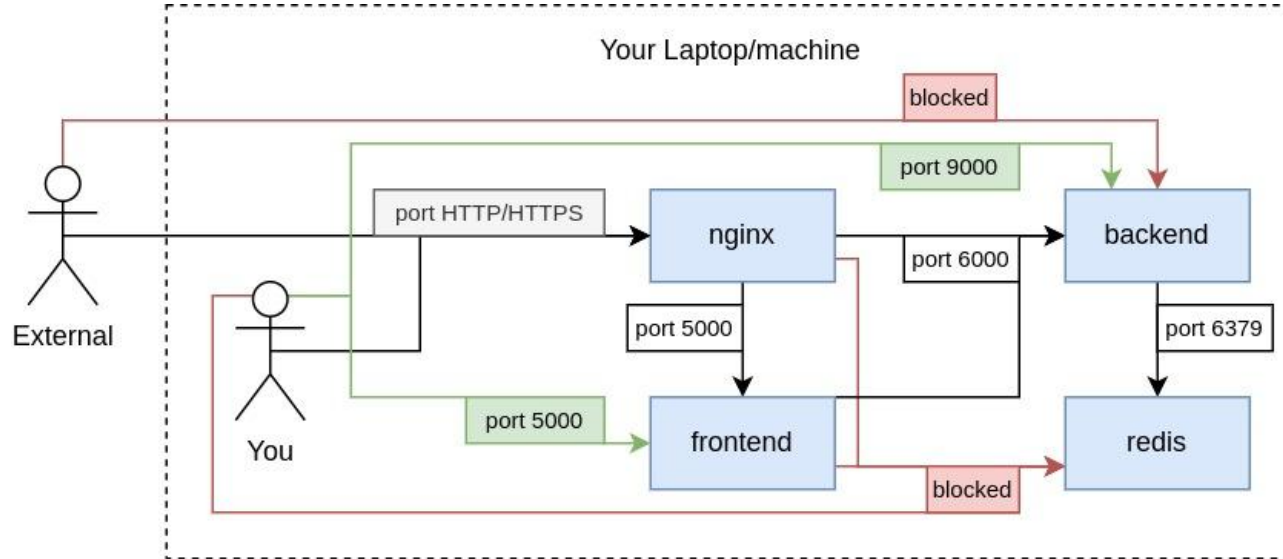
IDVOC project – Step 2.5

- Protect our host runner
 - ◆ Put some RAM limit for each container to 150 MiB
 - ◆ Limit the CPU to 2 for frontend and backend each
- Hostname looks better if it's not randomly generated
 - ◆ Give each container a hostname matching container name
- Give redis a volume for persistent data
 - ◆ Check that the visit counter isn't reseted on redis re-creation
- Expose frontend and backend directly, but on local machine only
 - ◆ They will still be reachable from outside the host machine via nginx



IDVOC project – Step 2.5

- Frontend & backend reachable only locally, on port 5000 and 9000



IDVOC project – Step 2.5

- Use some YAML anchors to avoid repeating yourself too much
- Help backend with performances by providing storage for cache
 - ◆ Mount /cache from the container to /tmp/backend on the host
 - You may have permission issues. `chmod 777` will be assumed on /tmp/backend for the testsuite
 - ◆ Check that the Lorem Ipsum on frontend doesn't change when backend is re-created

Step 3

Independent, interact with gitlab.cri.epita.fr



IDVOC project – Step 3

- You have a project, it's nice, but it needs some CI
- Write a `.gitlab-ci.yml` file to create a CI
- The CI shall be basic:
 - ◆ Two stages called lint and display-lint
 - ◆ In lint stage, 2 jobs
 - Each job will do the same thing, but one for frontend and one for backend
 - ◆ In display-lint, 1 job



IDVOC project – Step 3

- The lint jobs shall run python ruff on its project, to check code quality
 - ◆ They will be quality problems
- They need also to write a report of the code quality and provide it as an artifact
- The display-lint job needs to read these reports and print them
- Output of display-lint is up to you
 - ◆ But it must get its content from the previous jobs



Step 3.5

Last steps of the project



IDVOC project – Step 3.5

- Since both lint job are kinda the same, find a way to not write the image part twice
 - ◆ Nor the stage within the job
- Allow the CI to be run only on commit push, and if the commit message doesn't contain “no-ci”
- Printing the report must also be done even if any of the 2 first jobs failed

Advices

One will have to read it carefully



IDVOC project – Advices

- The project will take some time, as usual, start early
- The project is voluntarily vague on some points. The point is to provide a context that may be a bit similar to the one you may have in enterprise later on
 - ◆ However, don't hesitate to ask me any question
- Start by the easy stuff before getting to the hard one
- You know the drill about cheating by now
 - ◆ And its consequences
 - ◆ And how it's checked

IDVOC project – Advices

- Correction will be mostly automatic. Don't miss a typo
- Trust me, while being minimalistic and a bit dumb, this project is really realistic in terms of what can be asked and expected for most of you. Take it seriously and try to learn



Dementors

- Some dementors will be run and made available to you
- The goal of those dementors is not for you to do some moulinette-driven-development
- It's about making sure you don't fail your submission because of a typo
- The dementor will be provided as a gitlab issue on your project. If by the end of the dementor period your friends got an issue but you didn't, double check the submission instructions.
 - ◆ If it appears to be correct but still didn't receive, reach me out
 - ◆ If any question on the dementor, reply to the issue with your question

Slides available on zarak.fr/

Contact: cyril@cri.epita.fr

[zarak production#5492](#)