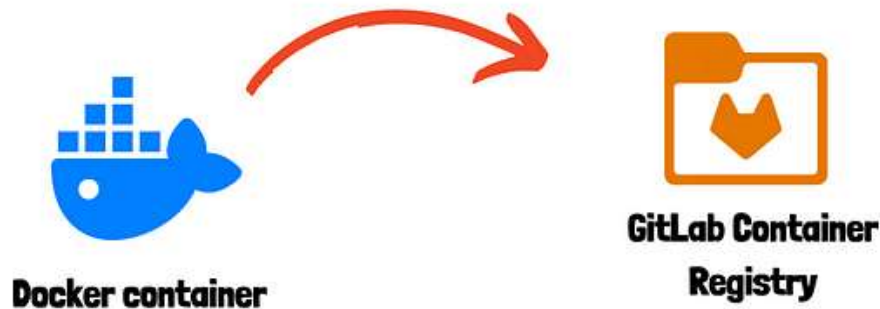


[< Go to the original](#)

How to Build a Docker Image and Push it to the GitLab Container Registry from a GitLab CI pipeline



Valentin Despa

Follow



DevOps with Valentine · ~7 min read ·

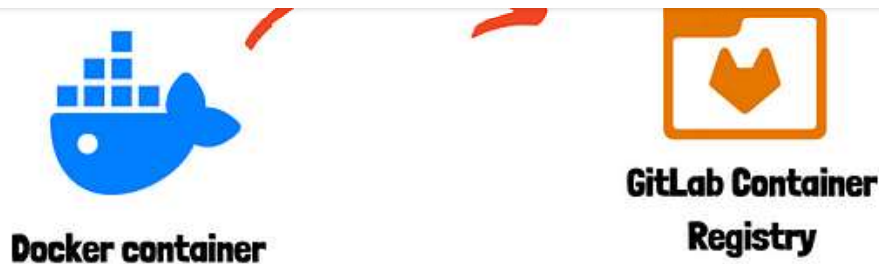
October 27, 2021 (Updated: January 4, 2022) · Free: No

While Dockerhub is a public registry that is essentially used by everyone, you don't always need to use it. You can build your own Docker images and publish them in the GitLab Container Registry, which can act as a private registry.

Maybe you are wondering if there is a way to store Docker images at GitLab and use them in pipelines. There are many reasons why you may want to use the GitLab Container Registry. There are two typical use-cases:

- to store your dockerized applications
- to use your own Docker image inside the GitLab CI pipeline

Freedium



Here is what this article will cover:

- How to **build and push a Docker image** to the GitLab Container Registry.
- How to figure out the **size of the Docker images** you are using.
- How to **use a Docker image from the GitLab Container Registry** in a GitLab CI pipeline.

The problem

Quite often, when creating a GitLab CI pipeline, you start with a base Docker image from Dockerhub and you add any missing dependencies. Here is a very simplified version where `pytest` is installed as a dependency:

Copy

```
some test job:
  image: python
  script:
    - python --version
    - pip --version
    - pip install pytest
    - pytest --version
    - echo "Doing complex stuff"
```

If you want to learn how to build pipelines in Gitlab CI, I have created an online course that starts with the basics of Gitlab CI and

Freedium

At one point, you may end up with a lot of boilerplate scripts that you carry over from project to project. **If there is no public Docker image that has everything you need, just build one!**

The first step is to create a `Dockerfile` . For the example above, the file would look as follows:

Copy

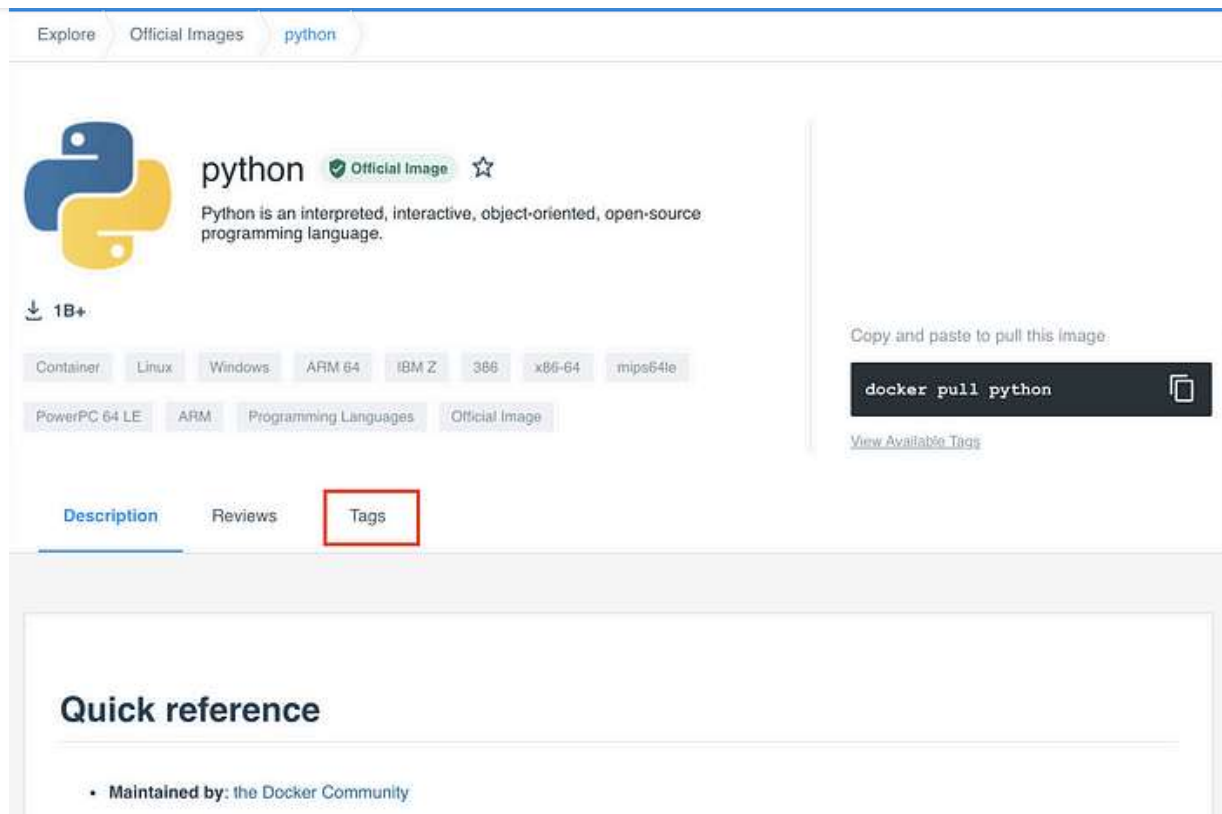
```
FROM python:3.10
RUN pip install pytest
```

What is the size of the Docker images I am using?

Regardless if you are building a custom Docker image or if you are using an image from the public Dockerhub registry, **knowing the size of your images is key in optimizing the performance of your pipeline.**

Let's take a Docker image containing Python for example. From the [official Python page on Dockerhub](#), click on *Tags*.

Freedium



This will display all the different versions available. As you can notice, the latest version has over 300MB!

TAG		
latest		
Last pushed 3 hours ago by doijanky		
docker pull python:latest		
DIGEST	OS/ARCH	COMPRESSED SIZE
fba7d2d8371f	linux/386	338.75 MB
1283a8de5b5c	windows/amd64	5.9 GB
694c80fd5c48	windows/amd64	2.55 GB
+8 more...		

This is a full version of the Docker image with many Linux packages. But for every job that uses this image, you need to download 300MB before you can do anything.

But here is the thing: you may not need all the dependencies that this full image has. Quite often, Docker images are available for

Freedium

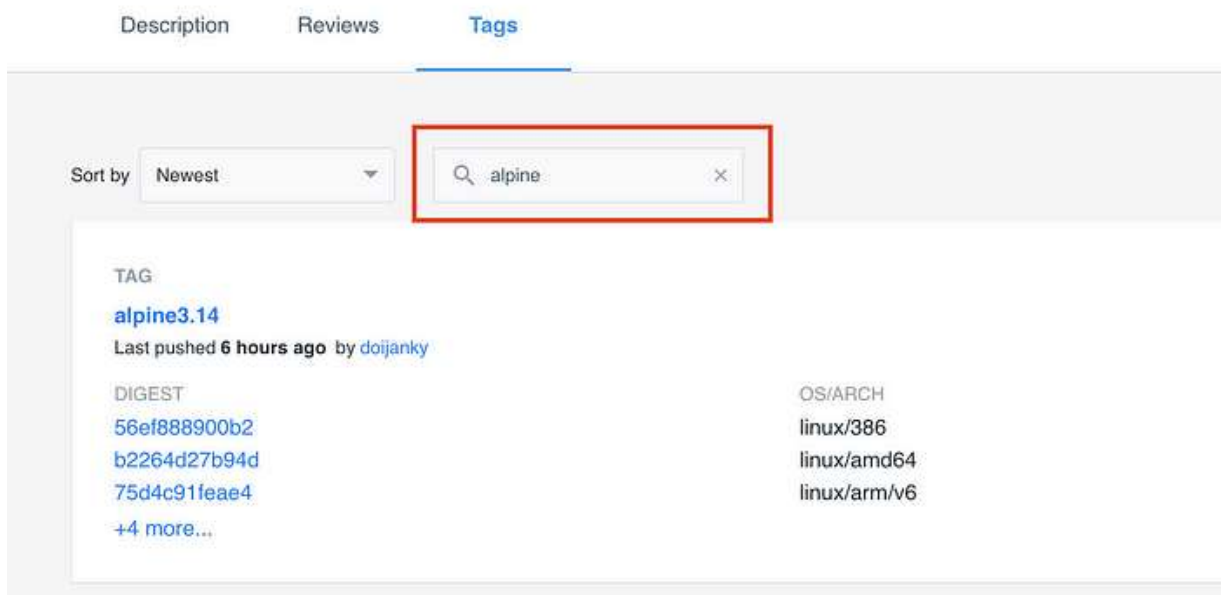


TAG		
alpine3.14		
Last pushed 5 hours ago by doijanky		
docker pull python:alpine3.14		
DIGEST	OS/ARCH	COMPRESSED SIZE ⓘ
56ef888900b2	linux/386	16.94 MB
b2264d27b94d	linux/amd64	16.72 MB
75d4c91feae4	linux/arm/v6	16.19 MB
+4 more...		

The difference comes from the fact that Alpine Linux is really lightweight and many packages are missing, including cURL.

There is also an alternative tag to Alpine Linux: `slim`. This is a type of a Docker image that has been minimized with DockerSlim.

So I highly recommend giving smaller packages a try. Use the search field to locate the version of the image you wish to use.



Description Reviews **Tags**

Sort by Newest

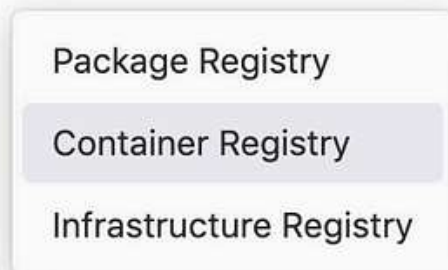
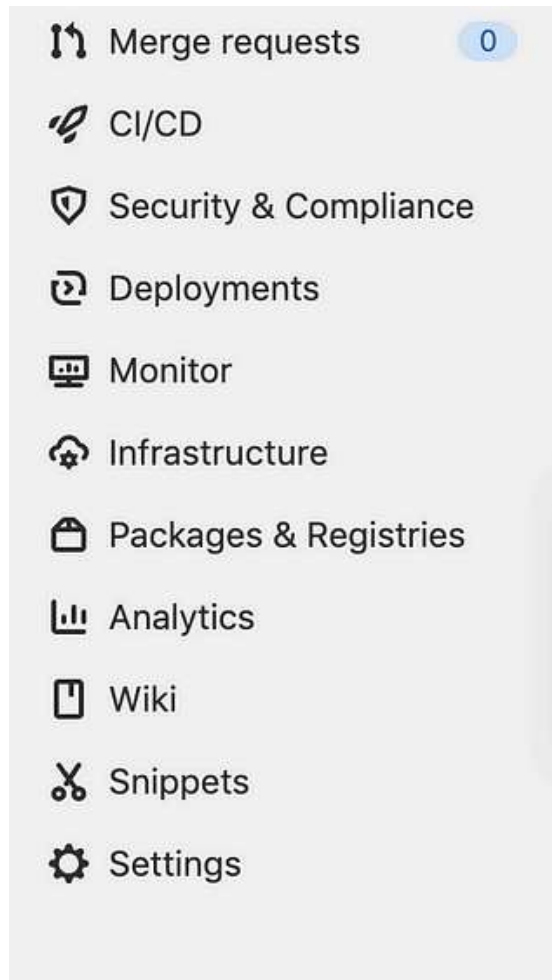
Search: alpine

TAG	
alpine3.14	
Last pushed 6 hours ago by doijanky	
DIGEST	OS/ARCH
56ef888900b2	linux/386
b2264d27b94d	linux/amd64
75d4c91feae4	linux/arm/v6
+4 more...	

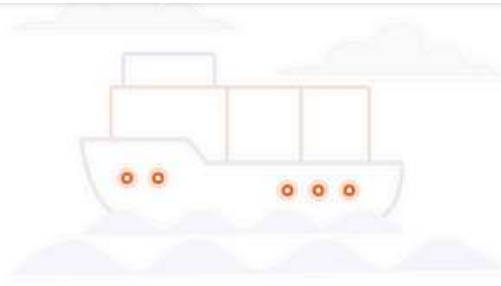
Viewing the GitLab Container Registry

Freedium

Container Registry.



Freedium



There are no container images stored for this project

With the Container Registry, every project can have its own space to store its Docker images. [More Information](#)

CLI Commands

If you are not already logged in, you need to authenticate to the Container Registry by using your GitLab username and password.

If you have [Two-Factor Authentication](#) enabled, use a [Personal Access Token](#) instead of a password.

```
docker login registry.gitlab.com
```



You can add an image to this registry with the following commands:

```
docker build -t registry.gitlab.com/user-que:
```



```
docker push registry.gitlab.com/user-questio
```



At this point, the registry will be empty as we have not pushed any images yet.

If you are using GitLab.com with a private account that is not part of an organization, this should be enabled by default. If this is not the case, you may need to ask your administrator for the necessary permissions.

Freedium

Build and push a Docker image to the GitLab Container Registry

Building and pushing a Docker image requires Docker. So we need to use the Docker image but also to start the Docker daemon using Docker-in-Docker (dind) as a service.

Copy

```
build image:
  image: docker
  services:
    - docker:dind
```

It is considered bad practice to use the latest version of a Docker image as this will always change and could break your pipeline. The version you are using changes all the time if you:

- don't specify a version (e.g. `docker`)
- use the latest tag (e.g. `docker:latest`)
- use the stable tag (e.g. `docker:stable`)
- use a major version tag(e.g. `docker:20`)

Consider the following:

- **Use a specific version**, like `docker:20.10.10` & `docker:20.10.10-dind` (recommended).
- Log the version with `--version` (this can help you figure out which was the version that worked).

So look up a version of Docker that works for you and try using that. I will use the version `20.10.10` .

Copy

Freedium

SERVICES:

- docker:20.10.10-dind

*If you want to **learn how to build pipelines in Gitlab CI**, I have created an online course that starts with the basics of Gitlab CI and YAML and helps you understand the fundamentals of CI/CD. [Learn more about the course.](#)*

So now that we know the size of the Docker images we are using, let's go ahead and adapt the `Dockerfile` by specifying a smaller base image that uses Alpine Linux:

Copy

```
FROM python:3.10-alpine
RUN pip install pytest
```

We could build this image from our computer and push it to the GitLab Container Registry, but I prefer to use automation. So let's define a new job in our pipeline. This can be part of the same pipeline that needs the image:

The first step is **login in to the GitLab Container Registry** using `docker login`. Since our plan is to use this command from a GitLab CI pipeline, we don't need to know or generate any credentials. We can simply reference the `CI_REGISTRY_USER`, `CI_REGISTRY_PASSWORD` and `CI_REGISTRY` environment variables.

Copy

```
echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER $CI_R
```

Freedium

The next step is building and the image. I will only use the latest version, but feel free to also tag it if you wish. Again, we can keep our pipeline configuration-free by using the predefined GitLab CI variable `CI_REGISTRY_IMAGE` .

***Heads-up!** If you are unsure which value this variables have, simply use `echo $VARNAME` to inspect their value. It is better to know exactly what you are doing instead of assuming.*

Copy

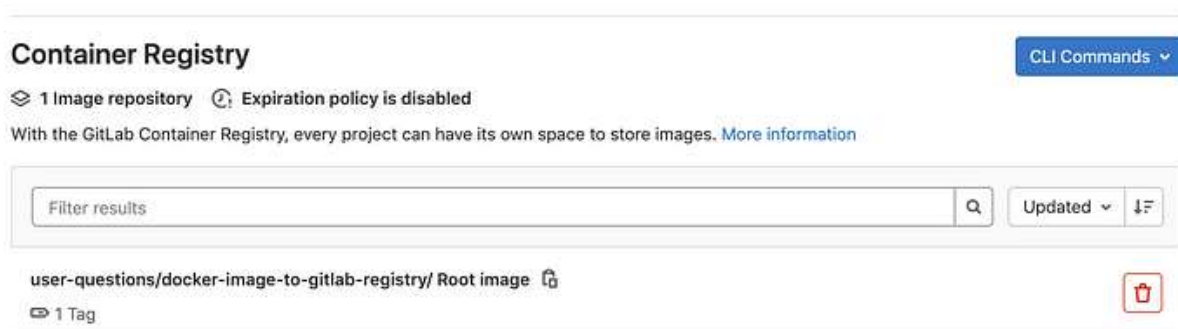
```
- docker build -t $CI_REGISTRY_IMAGE .
```

Finally, let's push the image:

Copy

```
docker push $CI_REGISTRY_IMAGE
```

If all commands succeeded, you should shortly see the image in your GitLab Container Registry.



Freedium



As I don't wish to build this image with every commit, I will setup a pipeline schedule and define a rule that will ensure this job only runs according to the schedule I define.

Copy

```
build image:
  image: docker:20.10.10
  services:
    - docker:20.10.10-dind
  rules:
    - if: $CI_PIPELINE_SOURCE == "schedule"
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER
    - docker build -t $CI_REGISTRY_IMAGE .
    - docker push $CI_REGISTRY_IMAGE
```

Creating well-researched and to-the-point content requires a lot of time and energy. If this was helpful and you wish to support me, consider [subscribing to Medium](#).

Using a Docker image from the GitLab Container Registry in a GitLab CI pipeline

This is actually easier than you think. All you need to know is the exact name of the image.

Go to your container registry and copy the full name.

Freedium



You can use the image in the same project or in other projects. A job requiring the Docker image would look like this:

Copy

```
run tests:
  image: registry.gitlab.com/somegroup/some-image-name
  rules:
    - if: $CI_PIPELINE_SOURCE != "schedule"
  script:
    - python --version
    - pip --version
    - pytest --version
```

Conclusion

I hope this tutorial helped you get started with building your own Docker images, publishing them in the GitLab Registry and using them in your pipelines. Leave a comment in the section below if you have any questions. I would love to hear from you!

Thank you for sticking with this article until the end. If you enjoyed it, please leave a comment, share, and press that 🙌 a few times (up to 50 times). It will help others discover this information and maybe it will help someone else as well.

Follow me on Medium and [YouTube](#) if you're interested in more tutorials like this one.

Freedium