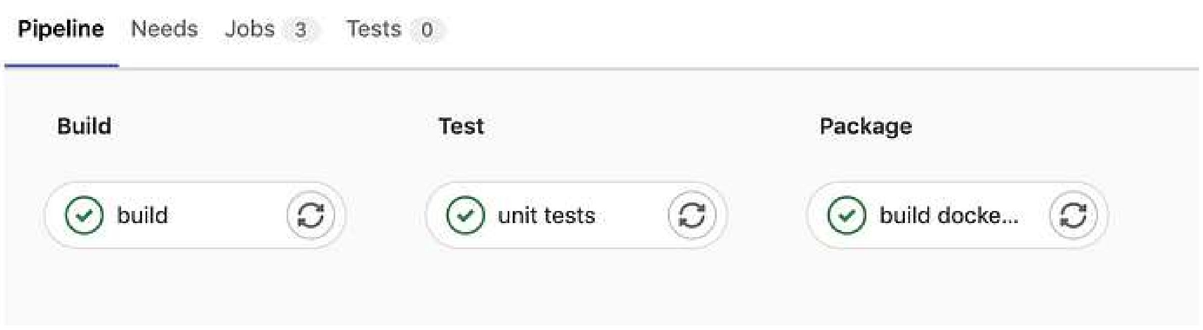
Freedium

< Go to the original



How to Start a Docker Container Inside your GitLab CI Pipeline

Valentin Despa

Follow

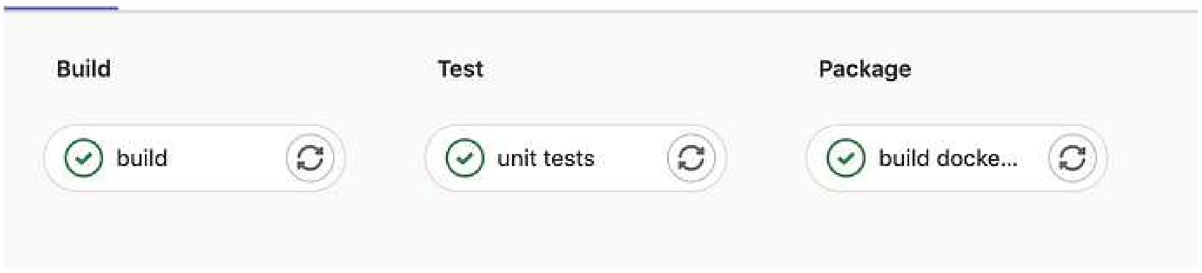
DevOps with Valentine · ~6 min read · November 2, 2021 (Updated: September 21, 2023) · Free: No

Quite often, we use GitLab CI to dockerize our applications. But how to start a Docker container from the GitLab Container Registry? Can we use Docker Compose? A lesser-known feature in GitLab CI is the services keyword which allows you to start one or more Docker images and link them to your job. Let's explore how this works.

Last update: September 2023

Background

Allow me to describe the following scenario for a pipeline. I have built a Node.js application that exposes an API.



So the current pipeline has the following stages:

- `build` where all dependencies are installed
- `test` where all unit tests and executed
- `package` where the application is dockerized, and the image is being pushed to the GitLab Container Registry.

For your reference, this is how the `.gitlab-ci.yml` looks like this at this stage:

Copy

```
stages:
  - build
  - test
  - package

build:
  stage: build
  image: node:14-alpine
  script:
    - npm ci --only=production
  artifacts:
    paths:
      - node_modules/
      - server.js

unit tests:
  stage: test
  image: node:14-alpine
  before_script:
    - npm install
```

```
build docker image:
  stage: package
  image: docker
  services:
    - docker:dind
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER $(
    - docker build -t $CI_REGISTRY_IMAGE .
    - docker push $CI_REGISTRY_IMAGE
```

> *This scenario uses the GitLab.com shared runner infrastructure where the the GitLab runners are using a Kubernetes executor.*

> *For some Docker executors, you may need to specify the DOCKER_HOST variable. You will know that this is the case if you get this error:*

> ***Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?***

> *This approach is **NOT** for shell executors.*

Copy

```
variables:
  DOCKER_HOST: tcp://docker:2375/
```

or

Copy

The Problem

The next logical step would be to do acceptance tests, run some cURL commands against the Docker container, or, even better use Postman/Newman to test the API.

So how to start a Docker container within the GitLab CI pipeline?

Starting a container with docker run

Locally, I have built and tested the container using docker build & docker run. The same should be possible within the pipeline, right?

The `build docker image` job already provides a very good template for using Docker within GitLab. So I will log in to the GitLab Container Registry and start the image I have built previously.

> *To simplify the pipeline, I have NOT specified any versions for the* `docker` *and* `docker:dind` *images nor have I created any tags for the image I have built. In a real scenario, I would do both.*

Copy

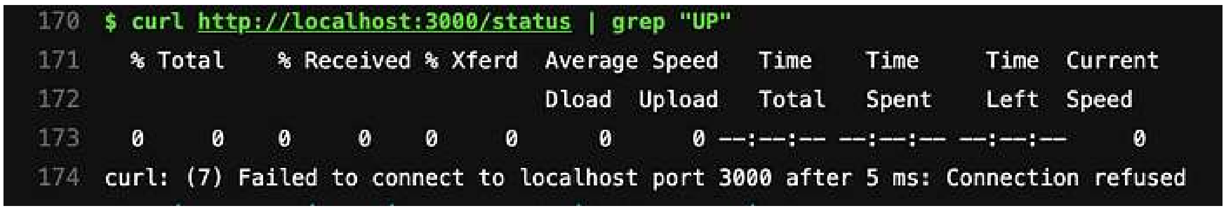
```
stages:
  - build
  - test
  - package
  - acceptance

...

curl api testing:
    - docker:dind
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER $(
    - docker run -d -p 3000:3000 $CI_REGISTRY_IMAGE
    - apk add curl
    - curl http://localhost:3000/status | grep "UP"
```

> *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.*

Unfortunately, the following setup will fail with the following error:

Copy

```
curl: (7) Failed to connect to localhost port 3000 after 5 ms: Connecti
```



```
$ curl http://localhost:3000/status | grep "UP"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
curl: (7) Failed to connect to localhost port 3000 after 5 ms: Connection refused
```

But why? Locally it was working just fine. To explain why, allow me to introduce you to GitLab CI services.

What are GitLab CI services?

As you have seen, the job building the pipeline has used the keyword `services` to specify the `docker:dind` image.

have specified in the `image` keyword).

One of the most typical use cases is when you need a database, an API, or some other service that you can call over the network.

Here are some key aspects that you need to remember:

- The image or images specified under services can only be reached over a network connection.
- The image or images specified under services must expose a service under a given port. Otherwise, they are useless in this context.
- You can't connect or run shell commands on the containers you start as services.
- using services is NOT like defining a `docker-compose.yaml` file

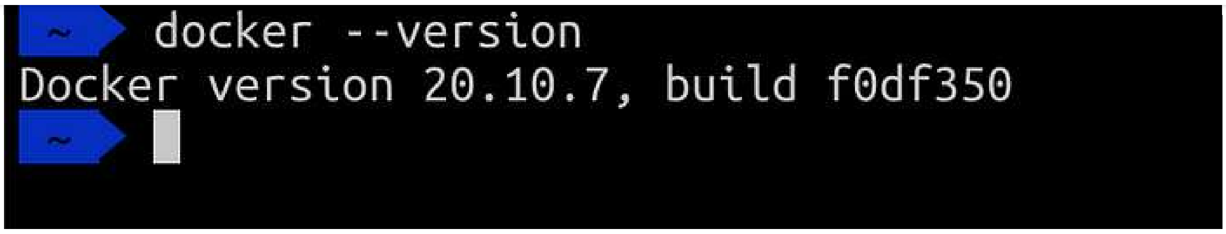
Why does `docker` need `docker:dind` as a service?

If you have Docker installed locally, you can open a terminal window and type a command like:

Copy

```
docker --version
```

The output will be something like



```
~ docker --version
Docker version 20.10.7, build f0df350
~
```

However, if I try to run a command like `docker pull`, `docker build` or `docker run`, I will get an error similar to this one:

Copy

```
Error response from daemon: dial unix docker.raw.sock: connect: connect
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is
```

At this point, allow me to quote the Docker documentation:

> Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Locally, you don't really think about this, as both the client and the daemon are installed on the same machine. But in GitLab, things are a bit different.

So in GitLab, the `docker` image is simply the client. The `docker:dind` image is the Docker daemon, and it started as a service, offering network-accessible services. The client can communicate with the daemon through the network interface.

Accessing a Docker container started as a service

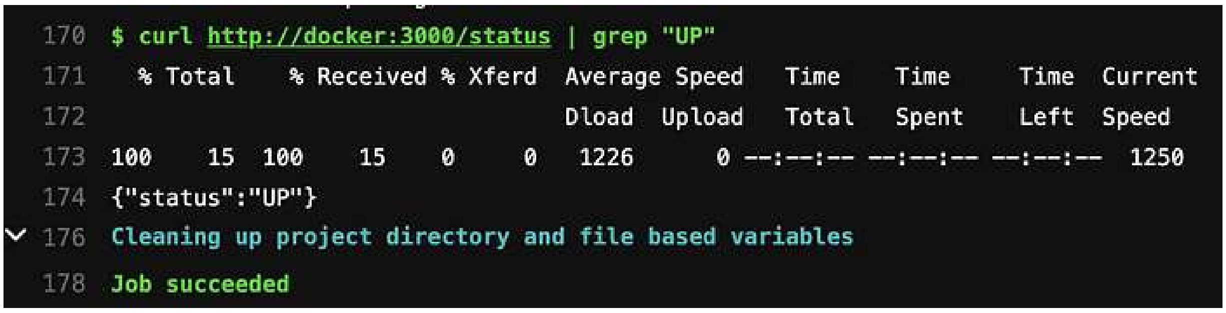
In the previous example, the Node.js application exposing an API on port 3000 has been started by the Docker Daemon within the `docker:dind` container.

So using localhost will not work, as the API is running on a different container. Fortunately, GitLab has auto-generated a hostname for the respective service. In this case, the hostname is `docker`.

So we can access the application with cURL by adapting the command to use `docker` instead of `localhost`.

Copy

```
curl http://docker:3000/status | grep "UP"
```



```
$ curl http://docker:3000/status | grep "UP"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    15  100    15    0     0   1226      0 --:--:-- --:--:-- --:--:--  1250
{"status":"UP"}
Cleaning up project directory and file based variables
Job succeeded
```

However, this approach is cumbersome, as we are now running Docker in Docker in Docker. The execution time for this job alone is 61 seconds. There must be a better way.

Starting a private Docker container with GitLab services

The services keyword allows us to specify both a public Docker image available on Dockerhub, but just as well we can use our private GitLab Container Registry.

```
curl api testing:
  stage: acceptance
  image: curlimages/curl
  services:
    - name: $CI_REGISTRY_IMAGE
      alias: banking-api
  script:
    - curl http://banking-api:3000/status | grep "UP"
```

I have extended the configuration by specifying an `alias`. I was not sure which hostname will GitLab generate, and I preferred to specify one by myself.

> *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,* ***please leave a comment, share, and press that 👏 a few times (up to 50 times). And consider [subscribing to Medium](). ***

Running Postman/Newman Acceptance Tests against a Docker image

Following the principles used with cURL, I can now add a new job that uses the public Newman Docker image to run an existing Postman collection.

Copy

```
postman api testing:
  stage: acceptance
  image:
    name: postman/newman
    entrypoint: [""]
  services:
    - name: $CI_REGISTRY_IMAGE
      alias: banking-api
```

Notice that I have injected the Postman environment variable baseUrl on runtime.

Troubleshooting

Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

For some Docker executors, you may need to specify the `DOCKER_HOST` variable at a pipeline or job level.

Copy

```
variables:
  DOCKER_HOST: tcp://docker:2375/
```

or

Copy

```
variables:
  DOCKER_HOST: tcp://docker:2376/
```

Conclusion

I hope this tutorial helped you access your dockerize application from your GitLab CI pipeline. Leave a comment in the section below if you have any questions. I would love to hear from you!

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.

References

- GitLab CI/CD Services
- Docker Docs — Docker overview
- GitLab Runner Issue tracker: Localhost not working like it does on my physical computers

#gitlab #docker #docker-run #dockerhub #gitlab-ci