GitLab Runners: Docker-in-Docker Explained

By Haseeb Majid

What are GitLab Runners?

An application which works with GitLab CI to run jobs in a pipeline ^1.

Types

- SaaS (Shared): GitLab's own runners
  - Enabled by default
  - Limited by credits
- Self-Hosted: Runners we manage
  - On our own infrastructure
  - Need to register them with GitLab to use them

What are Executors?

Any system used to make sure the CI jobs are run [2].

Each job is run separately from each other.

Let's take a look at a few of them in more detail.

Shell Executor

- Simplest executor
- All the dependencies required for the job must be manually pre-installed

Docker Executor

- Uses Docker to build clean environments for each job
- All dependencies can be set up within the Docker container
- Can also be used to run dependent services like MySQL

Kubernetes Executor

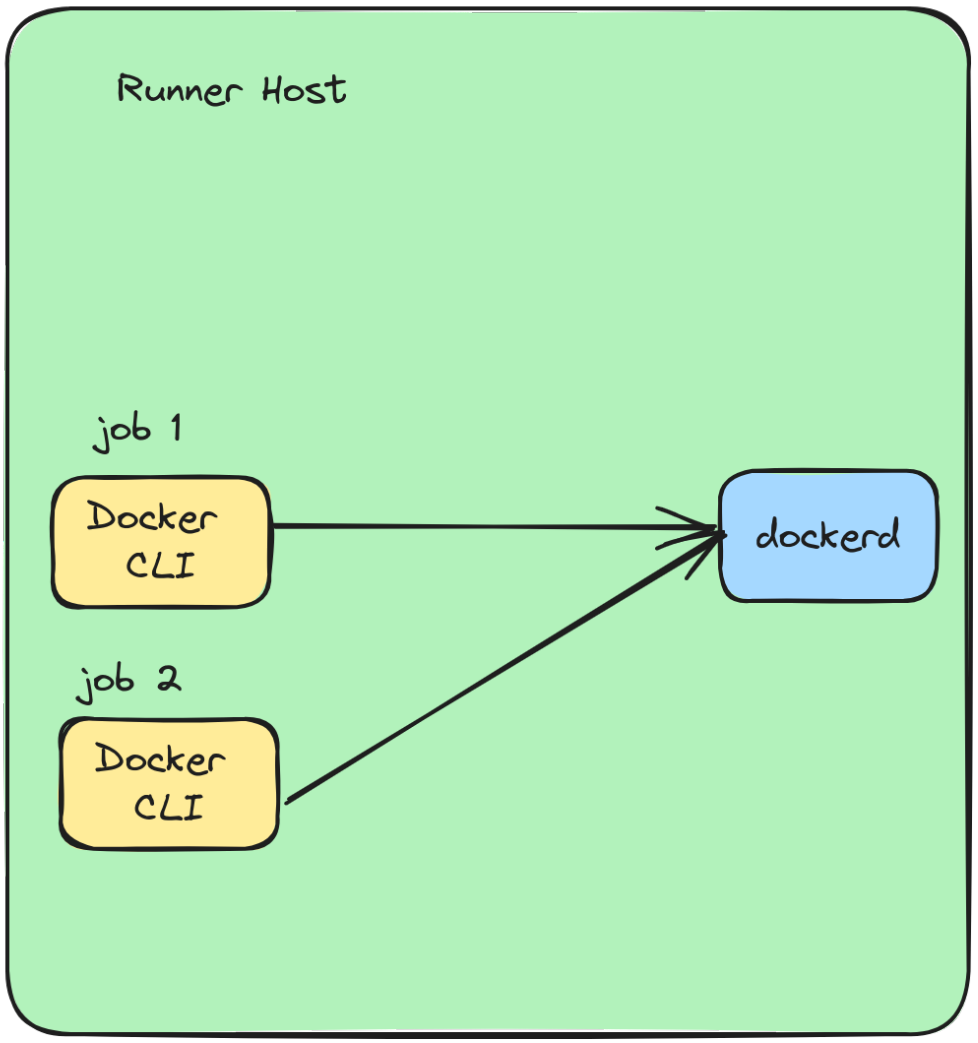
- Use k8s API to create a new pod for each job
  - In our cluster
- A pod can have multiple containers for a CI job

Docker in GitLab CI?

It depends on the executor we use.

Let's take a look at how we can do per executor 3

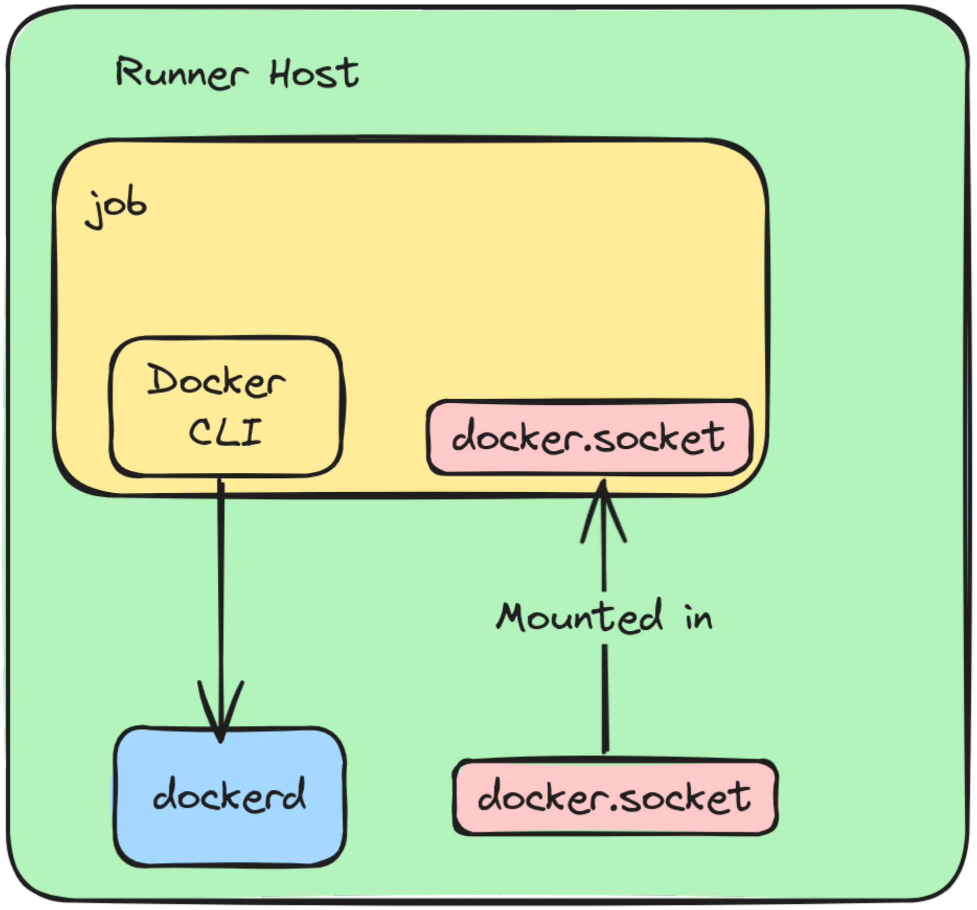
Docker with the Shell Executor


Runner Host
job 1
Docker CLI
job 2
Docker CLI
dockerd

- Runner needs to be in the `docker` group
- Have root level permissions
- Can easily take over the host machine

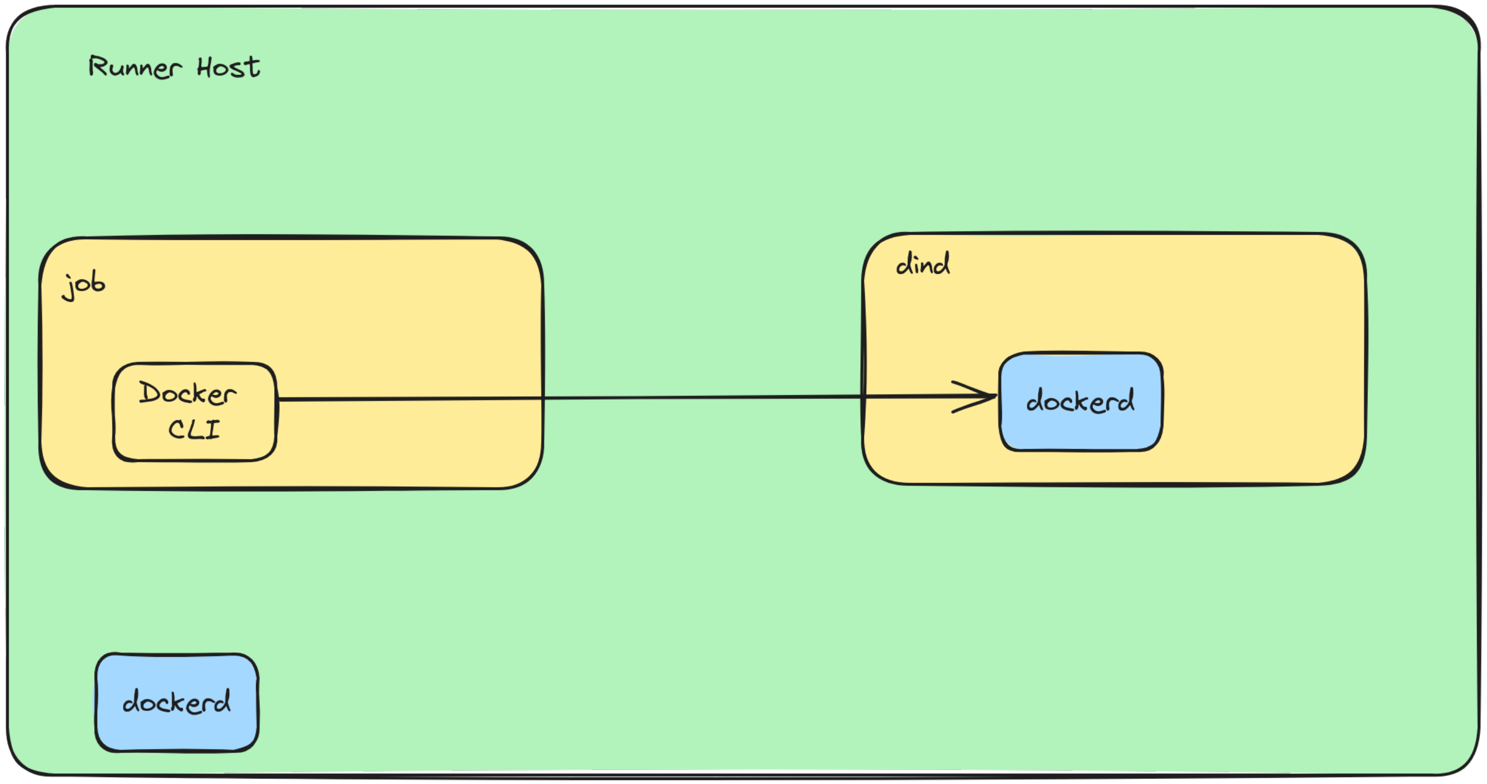
Docker With Docker Executor

DooD


Runner Host
job
Docker CLI
docker.socket
Mounted in
dockerd
docker.socket

- Naive first approach
- Mount Unix socket into container from host
  - Docker-out-of-Docker (DooD)
- Not secure
  - Kill all containers on the host machine
- Collisions between jobs
  - Two containers with same name

DinD



- Spins up a Docker engine service just for this job
  - Linked to our job
- Need to use the privileged flag

What is the privileged flag?

- Privileged mode gives a Docker container more permissions
  - Including running a Docker daemon inside of it
    - DinD
- Container can access all devices on the host machine
  - Can be insecure

Shared Runners

- Uses `docker+machine` executor
  - Adds auto-scaling support to runner
- For each job
  - A new VM is provisioned
  - VM only exists for duration of the job and is deleted after wards
  - Job has sudo access without a password

GitLab CI Services

```yaml
services-example:
  image: docker:24.0.7
  services:
    - docker:dind
    - nginx
  script:
    - sleep 5
    - wget -O - http://nginx:80
    - docker ps -a
```

GitLab CI Services

```yaml
services-example:
  image: docker:24.0.7
  services:
    - docker:dind
    - nginx
  script:
    - sleep 5
    - wget -O - http://nginx:80
    - docker ps -a
```

Why do they work?

- A container created for our job
- Uses deprecated Docker links
- Won't see in `docker ps`

```
# Output
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAM
```

Docker Compose

Let's look an example:

- We use docker-compose to spin up containers
- The "tests" will run within the job container

docker-compose.yml

```yaml
services:
  nginx:
    image: nginx
    ports:
      - 8080:80
```

.gitlab-ci.yml

```yaml
docker-compose-example:
  image: docker:24.0.7
  services:
    - docker:dind
  before_script:
    - docker-compose up --detach
  script:
    - sleep 5
    - wget -O - http://docker:8080
    - docker ps -a
  after_script:
    - docker-compose down
```

.gitlab-ci.yml

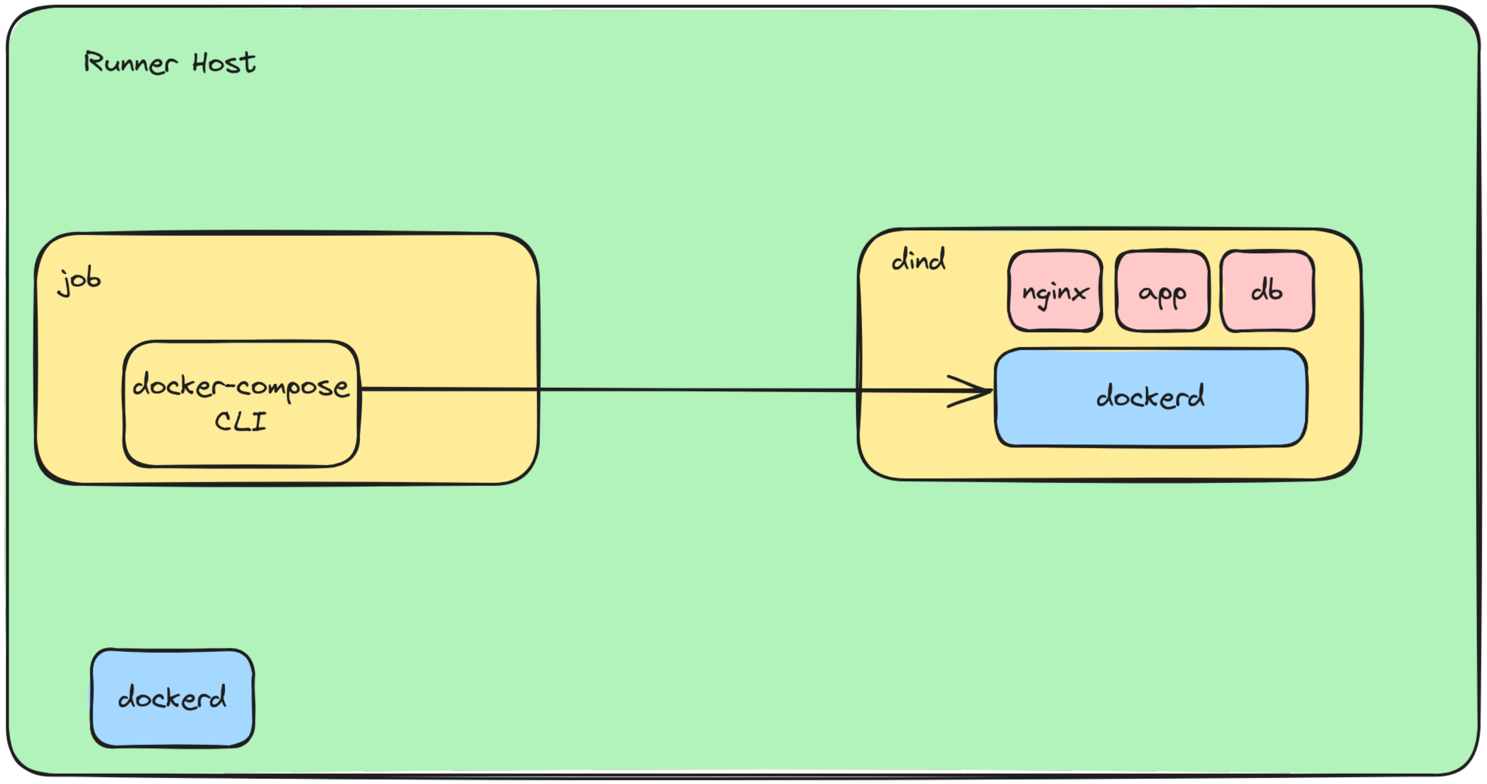
```yaml
docker-compose-example:
  image: docker:24.0.7
  services:
    - docker:dind
  before_script:
    - docker-compose up --detach
  script:
    - sleep 5
    - wget -O - http://docker:8080
    - docker ps -a
  after_script:
    - docker-compose down
```

.gitlab-ci.yml

```yaml
docker-compose-example:
  image: docker:24.0.7
  services:
    - docker:dind
  before_script:
    - docker-compose up --detach
  script:
    - sleep 5
    - wget -O - http://docker:8080
    - docker ps -a
  after_script:
    - docker-compose down
```

```
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         ST
94e16c6ec4f9   nginx     "/docker-entrypoint.…"   7 seconds ago   Up
```

What happens?



Deeper Dive

```
# View all docker neworks
docker network ls

NETWORK ID     NAME                              DRIVER    SCOPE
a5d2becba851   bridge                            bridge    local
8b593743d091   ci-dind-docker-compose_default    bridge    local
77382cfc2d50   host                              host      local
9ad5bc56591c   none                              null      local
```

```
#  Inspect the docker compose network
docker network inspect ci-dind-docker-compose_default
```

```
[
    {
        "Name": "ci-dind-docker-compose_default",
        "Id": "773e4ec2e9c032cbbd6fa903512089147a946327d46ae7264c
        "Created": "2023-11-28T14:48:29.714351003Z",
        "Scope": "local",
        "Driver": "bridge",
        "ConfigOnly": false,
        "Containers": {
            "c0f3c94f601f68296d204f982de792ad70b7fe8c1563d07e0fca
                "Name": "ci-dind-docker-compose-nginx-1",
                "EndpointID": "459198d8260f3474fbf8d426abc5fd4a38
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
```

Solutions?

- Attach to docker-compose network ^4
- Use `docker` instead of `localhost` or hostname
- Use GitLab Services ^5
  - With host network

Appendix

- Example Repo