Docker Swarm

- Docker Swarm service discovery architecture
- Setting up Prometheus
- Monitoring Docker daemons
- Monitoring Containers
- Discovered labels
  - Scraping metrics via a certain network only
  - Scraping global tasks only
  - Adding a docker_node label to the targets
- Connecting to the Docker Swarm
- Conclusion

Prometheus can discover targets in a Docker Swarm cluster, as of v2.20.0. This guide demonstrates how to use that service discovery mechanism.

Docker Swarm service discovery architecture

The Docker Swarm service discovery contains 3 different roles: nodes, services, and tasks.

The first role, **nodes**, represents the hosts that are part of the Swarm. It can be used to automatically monitor the Docker daemons or the Node Exporters who run on the Swarm hosts.

The second role, **tasks**, represents any individual container deployed in the swarm. Each task gets its associated service labels. One service can be backed by one or multiple tasks.

The third one, **services**, will discover the services deployed in the swarm. It will discover the ports exposed by the services. Usually you will want to use the tasks role instead of this one.

Prometheus will only discover tasks and service that expose ports.

NOTE: The rest of this post assumes that you have a Swarm running.

Setting up Prometheus

For this guide, you need to setup Prometheus. We will assume that Prometheus runs on a Docker Swarm manager node and has access to the Docker socket at `/var/run/docker.sock`.

Monitoring Docker daemons

Let's dive into the service discovery itself.

Docker itself, as a daemon, exposes metrics that can be ingested by a Prometheus server.

You can enable them by editing `/etc/docker/daemon.json` and setting the following properties:

```
{
  "metrics-addr" : "0.0.0.0:9323",
  "experimental" : true
}
```

Instead of `0.0.0.0`, you can set the IP of the Docker Swarm node.

A restart of the daemon is required to take the new configuration into account.

The Docker documentation contains more info about this.

Then, you can configure Prometheus to scrape the Docker daemon, by providing the following `prometheus.yml` file:

```
scrape_configs:
  # Make Prometheus scrape itself for metrics.
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']

  # Create a job for Docker daemons.
  - job_name: 'docker'
    dockerswarm_sd_configs:
```

```
      - host: unix:///var/run/docker.sock
        role: nodes
    relabel_configs:
      # Fetch metrics on port 9323.
      - source_labels: [__meta_dockerswarm_node_address]
        target_label: __address__
        replacement: $1:9323
      # Set hostname as instance label
      - source_labels: [__meta_dockerswarm_node_hostname]
        target_label: instance
```

For the nodes role, you can also use the `port` parameter of `dockerswarm_sd_configs`. However, using `relabel_configs` is recommended as it enables Prometheus to reuse the same API calls across identical Docker Swarm configurations.

Monitoring Containers

Let's now deploy a service in our Swarm. We will deploy cadvisor, which exposes container resources metrics:

```
docker service create --name cadvisor -l prometheus-job=cadvisor \
    --mode=global --publish target=8080,mode=host \
    --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock,ro \
    --mount type=bind,src=/,dst=/rootfs,ro \
    --mount type=bind,src=/var/run,dst=/var/run \
    --mount type=bind,src=/sys,dst=/sys,ro \
    --mount type=bind,src=/var/lib/docker,dst=/var/lib/docker,ro \
    google/cadvisor -docker_only
```

This is a minimal `prometheus.yml` file to monitor it:

```
scrape_configs:
  # Make Prometheus scrape itself for metrics.
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']

  # Create a job for Docker Swarm containers.
  - job_name: 'dockerswarm'
```

```
    dockerswarm_sd_configs:
      - host: unix:///var/run/docker.sock
        role: tasks
    relabel_configs:
      # Only keep containers that should be running.
      - source_labels: [__meta_dockerswarm_task_desired_state]
        regex: running
        action: keep
      # Only keep containers that have a `prometheus-job` label.
      - source_labels: [__meta_dockerswarm_service_label_prometheus_job]
        regex: .+
        action: keep
      # Use the prometheus-job Swarm label as Prometheus job label.
      - source_labels: [__meta_dockerswarm_service_label_prometheus_job]
        target_label: job
```

Let's analyze each part of the relabel configuration.

```
- source_labels: [__meta_dockerswarm_task_desired_state]
  regex: running
  action: keep
```

Docker Swarm exposes the desired state of the tasks over the API. In our example, we only **keep** the targets that should be running. It prevents monitoring tasks that should be shut down.

```
- source_labels: [__meta_dockerswarm_service_label_prometheus_job]
  regex: .+
  action: keep
```

When we deployed our cadvisor, we have added a label `prometheus-job=cadvisor`. As Prometheus fetches the tasks labels, we can instruct it to **only** keep the targets which have a `prometheus-job` label.

```
- source_labels: [__meta_dockerswarm_service_label_prometheus_job]
  target_label: job
```

That last part takes the label `prometheus-job` of the task and turns it into a target label, overwriting the default `dockerswarm` job label that comes from the scrape config.

Discovered labels

The Prometheus Documentation contains the full list of labels, but here are other relabel configs that you might find useful.

Scraping metrics via a certain network only

```
- source_labels: [__meta_dockerswarm_network_name]
  regex: ingress
  action: keep
```

Scraping global tasks only

Global tasks run on every daemon.

```
- source_labels: [__meta_dockerswarm_service_mode]
  regex: global
  action: keep
- source_labels: [__meta_dockerswarm_task_port_publish_mode]
  regex: host
  action: keep
```

Adding a docker_node label to the targets

```
- source_labels: [__meta_dockerswarm_node_hostname]
  target_label: docker_node
```

Connecting to the Docker Swarm

The above `dockerswarm_sd_configs` entries have a field host:

```
host: unix:///var/run/docker.sock
```

That is using the Docker socket. Prometheus offers additional configuration options to connect to Swarm using HTTP and HTTPS, if you prefer that over the unix socket.

Conclusion

There are many discovery labels you can play with to better determine which targets to monitor and how, for the tasks, there is more than 25 labels available. Don't hesitate to look at the "Service Discovery" page of your Prometheus server (under the "Status" menu) to see all the discovered labels.

The service discovery makes no assumptions about your Swarm stack, in such a way that given proper configuration, this should be pluggable to any existing stack.

□□□ #2
□ □□□ □□□ 2026-10-31 13:19:06 UTC
□ □□□ □□□ 2026-10-31 13:19:07 UTC