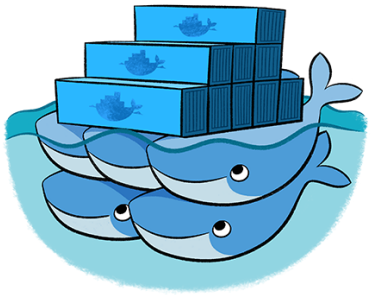
Usage

Configuration Example 1 (Global Mode)



> Before starting, you must have a swarm-cronjob instance up and running using docker.

Docker System Prune - Global Mode

This guide will help you set up a global Docker system prune task using Swarm Cronjob. This task will run on every node in your Docker Swarm cluster to remove unused data daily at 01:00 AM.

Step 1: Create a Docker Compose YAML

Create a Docker Compose file to define the system prune task. (You could also use the same Docker-Compose.yaml as in the BaseInstance)

```
version: "3.2"


services:
  prune-nodes:
    image: docker
    command: ["docker", "system", "prune", "-f"]
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      mode: global
      labels:
        - "swarm.cronjob.enable=true"
        - "swarm.cronjob.schedule=0 0 1 * * *"
        - "swarm.cronjob.skip-running=false"
      restart_policy:
        condition: none
```

Explanation of Configuration:

- `mode: global`: Ensures the task runs on every node in the cluster.
- `swarm.cronjob.schedule=0 0 1 * * *`: Schedules the task to run daily at 01:00 AM.
- `swarm.cronjob.skip-running=false`: Prevents skipping the task if it is already running.
- `restart_policy.condition=none`: Ensures the task does not restart automatically after completion.

> Add Docker labels to tell *swarm-cronjob* that your service is a cronjob.

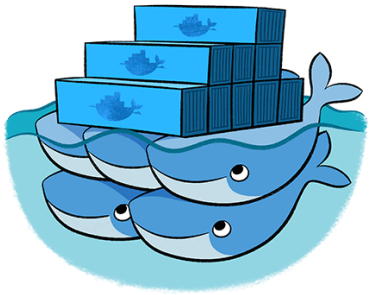
Step 2: Deploy the Stack

Deploy the global stack to your Docker Swarm cluster:

```
docker stack deploy -c prune-nodes.yml prune-nodes
```

Once deployed, the Docker system prune task will execute on every node in the swarm at the scheduled time.

Configuration Example 2 (Replicated Mode)



Running a Custom Cronjob with Swarm Cronjob

This guide demonstrates how to set up a cronjob using Swarm Cronjob. The example focuses on running a task periodically based on a defined schedule.

Prerequisites

- Swarm Cronjob instance is already up and running.

Step 1: Create a Docker Compose YAML

Create a Docker Compose file to define the cronjob:

```
version: "3.2"


services:
  test:
    image: busybox
    command: date
    deploy:
      mode: replicated
      replicas: 0
      labels:
        - "swarm.cronjob.enable=true"
        - "swarm.cronjob.schedule=* * * * *"
        - "swarm.cronjob.skip-running=false"
      restart_policy:
        condition: none
```

Step 2: Deploy the Stack

Deploy the stack to your Docker Swarm cluster:

```
docker stack deploy -c test.yml test
```

Explanation of Configuration

- `command`: Runs the task defined by the `command` field (in this case, the `date` command).
- `mode: replicated`: Ensures the task runs only when scheduled.

- `replicas: 0`: Prevents the task from running immediately upon deployment.
- `swarm.cronjob.schedule=* * * * *`: Schedules the task to run every minute.
- `restart_policy.condition=none`: Ensures the task does not restart automatically after completion.

Logs

Check the logs to verify task execution:

```
docker service logs swarm_cronjob_app
```

Example Logs:

```
swarm_cronjob_app.1.nvsjbhdhiagl@default    | Thu, 13 Dec 2018 20:04:37 UTC INF Starting swarm-cronjob v1.2.0
swarm_cronjob_app.1.nvsjbhdhiagl@default    | Thu, 13 Dec 2018 20:04:37 UTC INF Add cronjob with schedule * * * * * service=test
swarm_cronjob_app.1.nvsjbhdhiagl@default    | Thu, 13 Dec 2018 20:05:00 UTC INF Start job last_status=n/a service=test
swarm_cronjob_app.1.nvsjbhdhiagl@default    | Thu, 13 Dec 2018 20:06:00 UTC INF Start job last_status=n/a service=test
swarm_cronjob_app.1.nvsjbhdhiagl@default    | Thu, 13 Dec 2018 20:07:00 UTC INF Start job last_status=n/a service=test
swarm_cronjob_app.1.nvsjbhdhiagl@default    | Thu, 13 Dec 2018 20:08:00 UTC INF Start job last_status=n/a service=test
```

Once deployed, the cronjob will execute the defined task at the specified schedule.