

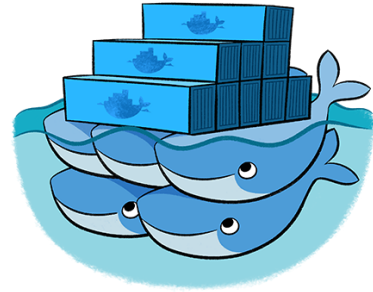
Swarm Cronjob

Setup Cronjobs for your Swarm Cluster!

- Overview
 - What are Swarm Cronjobs?
- Getting Started
 - Step by Step Setup Guide for Swarm Cronjobs
- Usage
 - Configuration Example 1 (Global Mode)
 - Configuration Example 2 (Replicated Mode)

Overview

What are Swarm Cronjobs?



--

--



Introduction

Swarm Cronjob is a tool designed to create and manage time-based scheduled tasks within a Docker Swarm environment. It operates by dynamically configuring services based on labels and interacting with the Docker API. This enables the execution of cron-like jobs in a distributed and failover-capable manner, ensuring high availability and flexibility.

Features

- ☐ Continuously updates its configuration without requiring a restart.
 - ☐ Implements cron scheduling using Go routines for efficiency.
 - ☐ Skips jobs if the associated service is already running, avoiding conflicts.
 - ☐ Supports timezone customization for accurate scheduling.
 - ☐ Automatic and dynamic configuration via Docker service labels.
 - ⚙ Distributed job execution across Docker Swarm nodes for failover and scalability.
-

Key Concepts

1. **Distributed Scheduling:** Swarm Cronjob operates within a Docker Swarm cluster, allowing jobs to run on any available node, ensuring failover and scalability.
 2. **Dynamic Configuration:** Services are configured automatically and dynamically via labels, removing the need for manual updates.
 3. **Flexible Scheduling:** With timezone support and Go-based cron implementation, scheduling is both precise and adaptable.
 4. **Node Independence:** By leveraging GlusterFS for shared storage, Swarm Cronjob ensures that job scripts like backup processes are not bound to specific nodes, enabling seamless failover.
-

Why Use Swarm Cronjob?

Swarm Cronjob is an excellent choice for teams or individuals looking to implement scheduled tasks in a Docker Swarm environment. It provides high availability by ensuring that tasks can execute on any node in the cluster, even in the event of node failure. When combined with a distributed file system like GlusterFS, it eliminates dependency on specific nodes for job execution, further enhancing reliability and flexibility. This makes it particularly suited for critical tasks like backups, data processing, and periodic maintenance in highly available environments.

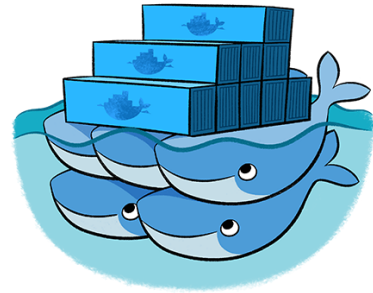
Conclusion

Swarm Cronjob bridges the gap between traditional cron functionality and the dynamic, distributed nature of Docker Swarm. By automating job scheduling and ensuring failover through its integration with distributed storage systems, it simplifies task management in containerized environments. Whether for backups, data synchronization, or other scheduled tasks, Swarm Cronjob delivers a robust and efficient solution.

Getting Started

Getting Started

Step by Step Setup Guide for Swarm Cronjobs



Introduction

This wiki article will guide you through the setup of Docker Swarm Cronjobs. Follow the steps below to deploy and configure cron-like scheduled tasks in your Docker Swarm environment efficiently.

Setup Guide

Step 1: Create a Docker Compose YAML

Create a Docker Compose file with the latest image:

```
version: "3.2"

services:
  swarm-cronjob:
    image: crazymax/swarm-cronjob:latest
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    environment:
      - "TZ=Europe/Paris"
      - "LOG_LEVEL=info"
      - "LOG_JSON=false"
    deploy:
      placement:
        constraints:
          - node.role == manager
```

Change the `TZ` environment variable to match your timezone.

Step 2: Deploy the Stack

Run the stack using one of the following methods:

- **Command Line:** Use the command:

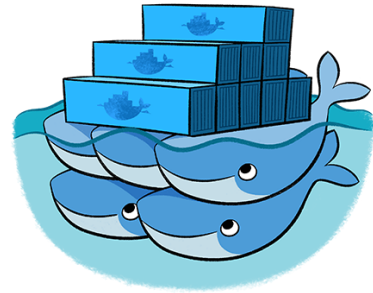
```
docker stack deploy -c docker-compose.yaml swarm-cronjob
```

- **Portainer GUI:** Follow the steps in the [Portainer Introduction](#) guide to deploy the stack.

If you are interested in configuring a System Prune job, check out the [System Prune Configuration](#).

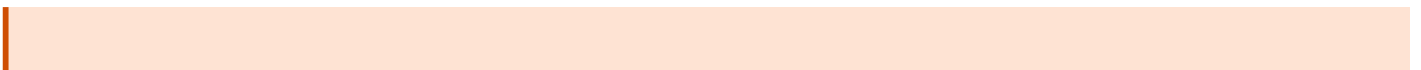
Usage

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.

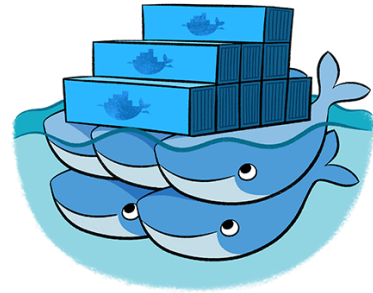
Step 2: Deploy the Stack

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.