Compose Deploy Specification

Table of contents

Deploy is an optional part of the Compose Specification. It is used to configure how services are deployed and managed in a Docker Swarm mode. Essentially, it provides a set of deployment specifications for managing the behavior of containers across different environments.

Attributes

endpoint_mode

`endpoint_mode` specifies a service discovery method for external clients connecting to a service. The Compose Deploy Specification defines two canonical values:

`endpoint_mode: vip`: Assigns the service a virtual IP (VIP) that acts as the front end for clients to reach the service on a network. Platform routes requests between the client and nodes running the service, without client knowledge of how many nodes are participating in the service or their IP addresses or ports.

`endpoint_mode: dnsrr`: Platform sets up DNS entries for the service such that a DNS query for the service name returns a list of IP addresses (DNS round-robin), and the client connects directly to one of these.

```
services:
 frontend:
   image: example/webapp
   ports:
     - "8080:80"
   deploy:
     mode: replicated
     replicas: 2
     endpoint_mode: vip
```

labels

`labels` specifies metadata for the service. These labels are only set on the service and not on any containers for the service. This assumes the platform has some native concept of "service" that can match the Compose application model.

```
services:
 frontend:
   image: example/webapp
   deploy:
     labels:
       com.example.description: "This label will appear on the web service"
```

mode

`mode` defines the replication model used to run the service on the platform. Either `global`, exactly one container per physical node, or `replicated`, a specified number of containers. The default is `replicated`.

```
services:
 frontend:
   image: example/webapp
   deploy:
     mode: global
```

placement

`placement` specifies constraints and preferences for the platform to select a physical node to run service containers.

constraints

`constraints` defines a required property the platform's node must fulfill to run the service container. It can be set either by a list or a map with string values.

```
deploy:
 placement:
   constraints:
     - disktype=ssd

deploy:
 placement:
   constraints:
     disktype: ssd
```

preferences

`preferences` defines a property the platform's node should fulfill to run service container. It can be set either by a list or a map with string values.

```
deploy:
 placement:
   preferences:
     - datacenter=us-east

deploy:
 placement:
   preferences:
     datacenter: us-east
```

replicas

If the service is `replicated` (which is the default), `replicas` specifies the number of containers that should be running at any given time.

```
services:
 frontend:
   image: example/webapp
   deploy:
     mode: replicated
     replicas: 6
```

resources

`resources` configures physical resource constraints for container to run on platform. Those constraints can be configured as:

- `limits`: The platform must prevent the container to allocate more.
- `reservations`: The platform must guarantee the container can allocate at least the configured amount.

```
services:
 frontend:
   image: example/webapp
   deploy:
     resources:
       limits:
         cpus: '0.50'
         memory: 50M
         pids: 1
       reservations:
         cpus: '0.25'
         memory: 20M
```

cpus

`cpus` configures a limit or reservation for how much of the available CPU resources, as number of cores, a container can use.

memory

`memory` configures a limit or reservation on the amount of memory a container can allocate, set as a string expressing a byte value.

pids

`pids` tunes a containerâ■■s PIDs limit, set as an integer.

devices

`devices` configures reservations of the devices a container can use. It contains a list of reservations, each set as an object with the following parameters: `capabilities`, `driver`, `count`, `device_ids` and `options`.

Devices are reserved using a list of capabilities, making `capabilities` the only required field. A device must satisfy all the requested capabilities for a successful reservation.

capabilities

`capabilities` are set as a list of strings, expressing both generic and driver specific capabilities. The following generic capabilities are recognized today:

- `gpu`: Graphics accelerator
- `tpu`: AI accelerator

To avoid name clashes, driver specific capabilities must be prefixed with the driver name. For example, reserving an nVidia CUDA-enabled accelerator might look like this:

```
deploy:
 resources:
   reservations:
     devices:
       - capabilities: ["nvidia-compute"]
```

driver

A different driver for the reserved device(s) can be requested using `driver` field. The value is specified as a string.

```
deploy:
 resources:
   reservations:
     devices:
       - capabilities: ["nvidia-compute"]
         driver: nvidia
```

count

If `count` is set to `all` or not specified, Compose reserves all devices that satisfy the requested capabilities. Otherwise, Compose reserves at least the number of devices specified. The value is specified as an integer.

```
deploy:
 resources:
   reservations:
     devices:
       - capabilities: ["tpu"]
         count: 2
```

`count` and `device_ids` fields are exclusive. Compose returns an error if both are specified.

device_ids

If `device_ids` is set, Compose reserves devices with the specified IDs provided they satisfy the requested capabilities. The value is specified as a list of strings.

```
deploy:
 resources:
   reservations:
     devices:
       - capabilities: ["gpu"]
         device_ids: ["GPU-f123d1c9-26bb-df9b-1c23-4a731f61d8c7"]
```

`count` and `device_ids` fields are exclusive. Compose returns an error if both are specified.

options

Driver specific options can be set with `options` as key-value pairs.

```
deploy:
 resources:
   reservations:
     devices:
       - capabilities: ["gpu"]
         driver: gpuvendor
         options:
```

```
          virtualization: false
```

restart_policy

`restart_policy` configures if and how to restart containers when they exit. If `restart_policy` is not set, Compose considers the `restart` field set by the service configuration.

`condition`. When set to:

- `none`, containers are not automatically restarted regardless of the exit status.
- `on-failure`, the container is restarted if it exits due to an error, which manifests as a non-zero exit code.
- `any` (default), containers are restarted regardless of the exit status.

- `delay`: How long to wait between restart attempts, specified as a duration. The default is 0, meaning restart attempts can occur immediately.
- `max_attempts`: How many times to attempt to restart a container before giving up (default: never give up). If the restart does not succeed within the configured `window`, this attempt doesn't count toward the configured `max_attempts` value. For example, if `max_attempts` is set to '2', and the restart fails on the first attempt, more than two restarts must be attempted.
- `window`: How long to wait before deciding if a restart has succeeded, specified as a duration (default: decide immediately).

```
deploy:
 restart_policy:
   condition: on-failure
   delay: 5s
   max_attempts: 3
   window: 120s
```

rollback_config

`rollback_config` configures how the service should be rollbacked in case of a failing update.

- `parallelism`: The number of containers to rollback at a time. If set to 0, all containers rollback simultaneously.
- `delay`: The time to wait between each container group's rollback (default 0s).
- `failure_action`: What to do if a rollback fails. One of `continue` or `pause` (default `pause`)
- `monitor`: Duration after each task update to monitor for failure `(ns|us|ms|s|m|h)` (default 0s).
- `max_failure_ratio`: Failure rate to tolerate during a rollback (default 0).
- `order`: Order of operations during rollbacks. One of `stop-first` (old task is stopped before starting new one), or `start-first` (new task is started first, and the running tasks briefly overlap) (default `stop-first`).

update_config

`update_config` configures how the service should be updated. Useful for configuring rolling updates.

- `parallelism`: The number of containers to update at a time.
- `delay`: The time to wait between updating a group of containers.
- `failure_action`: What to do if an update fails. One of `continue`, `rollback`, or `pause` (default: `pause`).
- `monitor`: Duration after each task update to monitor for failure `(ns|us|ms|s|m|h)` (default 0s).
- `max_failure_ratio`: Failure rate to tolerate during an update.
- `order`: Order of operations during updates. One of `stop-first` (old task is stopped before starting new one), or `start-first` (new task is started first, and the running tasks briefly overlap) (default `stop-first`).

```
deploy:
 update_config:
   parallelism: 2
   delay: 10s
   order: stop-first
```