Compose Deploy Specification

- Attributes
  - endpoint_mode
  - labels
  - mode
  - placement
    - constraints
    - preferences
  - replicas
  - resources
    - cpus
    - memory
    - pids
    - devices
      - capabilities
      - driver
      - count
      - device_ids
      - options
  - restart_policy
  - rollback_config
  - update_config

{{% include "compose/deploy.md" %}}

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 a service or job. Options include:

- `global`: Ensures exactly one task continuously runs per physical node until stopped.
- `replicated`: Continuously runs a specified number of tasks across nodes until stopped (default).
- `replicated-job`: Executes a defined number of tasks until a completion state (exits with code 0)'.
  - Total tasks are determined by `replicas`.
  - Concurrency can be limited using the `max-concurrent` option (CLI only).
- `global-job`: Executes one task per physical node with a completion state (exits with code 0).
  - Automatically runs on new nodes as they are added.

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

  batch-job:
    image: example/processor
    deploy:
      mode: replicated-job
      replicas: 5

  maintenance:
    image: example/updater
    deploy:
      mode: global-job
```

> - Job modes (`replicated-job` and `global-job`) are designed for tasks that complete and exit with code 0.
> - Completed tasks remain until explicitly removed.
> - Options like `max-concurrent` for controlling concurrency are supported only via the CLI and are not available in Compose.

For more detailed information about job options and behavior, see the Docker CLI documentation

`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. For a further example, see the CLI reference docs.

```
deploy:
  placement:
    constraints:
      - node.labels.disktype==ssd
```

`preferences`

`preferences` defines a strategy (currently `spread` is the only supported strategy) to spread tasks evenly over the values of the datacenter node label. For a further example, see the CLI reference docs

```
deploy:
  placement:
    preferences:
      - spread: node.labels.zone
```

`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 from allocating more resources.
- `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 container'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`: The maximum number of failed restart attempts allowed before giving up. (Default: unlimited retries.) A failed attempt only counts toward `max_attempts` if the container does not successfully restart within the time defined by `window`. For example, if `max_attempts` is set to `2` and the container fails to restart within the window on the first try, Compose continues retrying until two such failed attempts occur, even if that means trying more than twice.
- `window`: The amount of time to wait after a restart to determine whether it was successful, specified as a duration (default: the result is evaluated immediately after the restart).

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

`rollback_config`

`rollback_config` configures how the service should be rolled back 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
```

□□□ #2
□ □□□ □□□ 2026-10-31 19:42:46 UTC
□ □□□ □□□ 2026-10-31 19:42:47 UTC