docker service create

Estimated reading time: 33 minutes

Description

Create a new service

API 1.24+ (https://docs.docker.com/engine/api/v1.24/) The client and daemon API must both be at least 1.24 (https://docs.docker.com/engine/api/v1.24/) to use this command. Use the `docker version` command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

Usage

```
docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Options

Name, shorthand	Default	Description
`--config`		API 1.30+ (https://docs.docker.com/engine/api/v1.30/) Specify configurations to expose to the service
`--constraint`		Placement constraints
`--container-label`		Container labels
`--credential-spec`		API 1.29+ (https://docs.docker.com/engine/api/v1.29/) Credential spec for managed service account (Windows only)
`--detach , -d`		API 1.29+ (https://docs.docker.com/engine/api/v1.29/) Exit immediately instead of waiting for the service to converge

Name, shorthand	Default	Description
`--dns`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Set custom DNS servers
`--dns-option`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Set DNS options
`--dns-search`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Set custom DNS search domains
`--endpoint-mode`	`vip`	Endpoint mode (vip or dnsrr)
`--entrypoint`		Overwrite the default ENTRYPOINT of the image
`--env` , `-e`		Set environment variables
`--env-file`		Read in a file of environment variables
`--generic-resource`		User defined resources
`--group`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Set one or more supplementary user groups for the container
`--health-cmd`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Command to run to check health
`--health-interval`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Time between running the check (ms|s|m|h)
`--health-retries`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Consecutive failures needed to report unhealthy
`--health-start-period`		API 1.29+ (https://docs.docker.com/engine/api/v1.29/) Start period for the container to initialize before counting retries towards unstable (ms|s|m|h)

Name, shorthand	Default	Description
`--health-timeout`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Maximum time to allow one check to run (ms|s|m|h)
`--host`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Set one or more custom host-to-IP mappings (host:ip)
`--hostname`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Container hostname
`--init`		API 1.37+ (https://docs.docker.com/engine/api/v1.37/) Use an init inside each service container to forward signals and reap processes
`--isolation`		API 1.35+ (https://docs.docker.com/engine/api/v1.35/) Service container isolation mode
`--label` , `-l`		Service labels
`--limit-cpu`		Limit CPUs
`--limit-memory`		Limit Memory
`--log-driver`		Logging driver for service
`--log-opt`		Logging driver options
`--mode`	`replicated`	Service mode (replicated or global)
`--mount`		Attach a filesystem mount to the service
`--name`		Service name
`--network`		Network attachments
`--no-healthcheck`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Disable any container-specified HEALTHCHECK

Name, shorthand	Default	Description
`--no-resolve-image`		API 1.30+ (https://docs.docker.com/engine/api/v1.30/) Do not query the registry to resolve image digest and supported platforms
`--placement-pref`		API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Add a placement preference
`--publish` , `-p`		Publish a port as a node port
`--quiet` , `-q`		Suppress progress output
`--read-only`		API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Mount the container’s root filesystem as read only
`--replicas`		Number of tasks
`--reserve-cpu`		Reserve CPUs
`--reserve-memory`		Reserve Memory
`--restart-condition`		Restart when condition is met (“none”|”on-failure”|”any”) (default “any”)
`--restart-delay`		Delay between restart attempts (ns|us|ms|s|m|h) (default 5s)
`--restart-max-attempts`		Maximum number of restarts before giving up
`--restart-window`		Window used to evaluate the restart policy (ns|us|ms|s|m|h)
`--rollback-delay`		API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Delay between task rollbacks (ns|us|ms|s|m|h) (default 0s)
`--rollback-failure-action`		API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Action on rollback failure (“pause”|”continue”) (default “pause”)

Name, shorthand	Default	Description
`--rollback-max-failure-ratio`		API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Failure rate to tolerate during a rollback (default 0)
`--rollback-monitor`		API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Duration after each task rollback to monitor for failure (ns|us|ms|s|m|h) (default 5s)
`--rollback-order`		API 1.29+ (https://docs.docker.com/engine/api/v1.29/) Rollback order (“start-first”|”stop-first”) (default “stop-first”)
`--rollback-parallelism`	`1`	API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Maximum number of tasks rolled back simultaneously (0 to roll back all at once)
`--secret`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Specify secrets to expose to the service
`--stop-grace-period`		Time to wait before force killing a container (ns|us|ms|s|m|h) (default 10s)
`--stop-signal`		API 1.28+ (https://docs.docker.com/engine/api/v1.28/) Signal to stop the container
`--tty` , `-t`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Allocate a pseudo-TTY
`--update-delay`		Delay between updates (ns|us|ms|s|m|h) (default 0s)
`--update-failure-action`		Action on update failure (“pause”|”continue”|”rollback”) (default “pause”)
`--update-max-failure-ratio`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Failure rate to tolerate during an update (default 0)

Name, shorthand	Default	Description
`--update-monitor`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Duration after each task update to monitor for failure (ns|us|ms|s|m|h) (default 5s)
`--update-order`		API 1.29+ (https://docs.docker.com/engine/api/v1.29/) Update order ("start-first"|"stop-first") (default "stop-first")
`--update-parallelism`	`1`	Maximum number of tasks updated simultaneously (0 to update all at once)
`--user , -u`		Username or UID (format: <name|uid>[:<group|gid>])
`--with-registry-auth`		Send registry authentication details to swarm agents
`--workdir , -w`		Working directory inside the container

Parent command

Command	**Description**
docker service (https://docs.docker.com/engine/reference/commandline/service)	Manage services

Related commands

Command	Description
docker service create (https://docs.docker.com/engine/reference/commandline/service_create/)	Create a new service
docker service inspect (https://docs.docker.com/engine/reference/commandline/service_inspect/)	Display detailed information on one or more services
docker service logs (https://docs.docker.com/engine/reference/commandline/service_logs/)	Fetch the logs of a service or task
docker service ls (https://docs.docker.com/engine/reference/commandline/service_ls/)	List services

Command	Description
docker service ps (https://docs.docker.com/engine/reference/commandline/service_ps/)	List the tasks of one or more services
docker service rm (https://docs.docker.com/engine/reference/commandline/service_rm/)	Remove one or more services
docker service rollback (https://docs.docker.com/engine/reference/commandline/service_rollback/)	Revert changes to a service's configuration
docker service scale (https://docs.docker.com/engine/reference/commandline/service_scale/)	Scale one or multiple replicated services
docker service update (https://docs.docker.com/engine/reference/commandline/service_update/)	Update a service

Extended description

Creates a service as described by the specified parameters. You must run this command on a manager node.

Examples

Create a service

```
$ docker service create --name redis redis:3.0.6

dmu1ept4cxcfe8k8lhtux3ro3

$ docker service create --mode global --name redis2 redis:3.0.6

a8q9dasaafudfs8q8w32udass

$ docker service ls

ID            NAME    MODE        REPLICAS  IMAGE
dmu1ept4cxcf  redis   replicated  1/1       redis:3.0.6
a8q9dasaafud  redis2  global      1/1       redis:3.0.6
```

CREATE A SERVICE USING AN IMAGE ON A PRIVATE REGISTRY

If your image is available on a private registry which requires login, use the `--with-registry-auth` flag with `docker service create`, after logging in. If your image is stored on `registry.example.com`, which is a private registry, use a command like the following:

```
$ docker login registry.example.com

$ docker service  create \
  --with-registry-auth \
  --name my_service \
  registry.example.com/acme/my_image:latest
```

This passes the login token from your local client to the swarm nodes where the service is deployed, using the encrypted WAL logs. With this information, the nodes are able to log into the registry and pull the image.

Create a service with 5 replica tasks (--replicas)

Use the `--replicas` flag to set the number of replica tasks for a replicated service. The following command creates a `redis` service with `5` replica tasks:

```
$ docker service create --name redis --replicas=5 redis:3.0.6

4cdgfyky7ozwh3htjfw0d12qv
```

The above command sets the *desired* number of tasks for the service. Even though the command returns immediately, actual scaling of the service may take some time. The `REPLICAS` column shows both the *actual* and *desired* number of replica tasks for the service.

In the following example the desired state is `5` replicas, but the current number of `RUNNING` tasks is `3` :

```
$ docker service ls

ID            NAME   MODE        REPLICAS  IMAGE
4cdgfyky7ozw  redis  replicated  3/5       redis:3.0.7
```

Once all the tasks are created and `RUNNING` , the actual number of tasks is equal to the desired number:

```
$ docker service ls

ID            NAME   MODE        REPLICAS  IMAGE
4cdgfyky7ozw  redis  replicated  5/5       redis:3.0.7
```

Create a service with secrets

Use the `--secret` flag to give a container access to a secret (https://docs.docker.com/engine/reference/commandline/secret_create/).

Create a service specifying a secret:

```
$ docker service create --name redis --secret secret.json redis:3.0.6

4cdgfyky7ozwh3htjfw0d12qv
```

Create a service specifying the secret, target, user/group ID, and mode:

```
$ docker service create --name redis \
    --secret source=ssh-key,target=ssh \
    --secret source=app-key,target=app,uid=1000,gid=1001,mode=0400 \
    redis:3.0.6

4cdgfyky7ozwh3htjfw0d12qv
```

To grant a service access to multiple secrets, use multiple `--secret` flags.

Secrets are located in `/run/secrets` in the container. If no target is specified, the name of the secret will be used as the in memory file in the container. If a target is specified, that will be the filename. In the example above, two files will be created: `/run/secrets/ssh` and `/run/secrets/app` for each of the secret targets specified.

Create a service with a rolling update policy

```
$ docker service create \
  --replicas 10 \
  --name redis \
  --update-delay 10s \
  --update-parallelism 2 \
  redis:3.0.6
```

When you run a service update (https://docs.docker.com/engine/reference/commandline/service_update/), the scheduler updates a maximum of 2 tasks at a time, with `10s` between updates. For more information, refer to the rolling updates tutorial (https://docs.docker.com/engine/swarm/swarm-tutorial/rolling-update/).

Set environment variables (-e, --env)

This sets an environmental variable for all tasks in a service. For example:

```
$ docker service create \
  --name redis_2 \
  --replicas 5 \
  --env MYVAR=foo \
  redis:3.0.6
```

To specify multiple environment variables, specify multiple `--env` flags, each with a separate key-value pair.

```
$ docker service create \
  --name redis_2 \
  --replicas 5 \
  --env MYVAR=foo \
  --env MYVAR2=bar \
  redis:3.0.6
```

Create a service with specific hostname (--hostname)

This option sets the docker service containers hostname to a specific string. For example:

```
$ docker service create --name redis --hostname myredis redis:3.0.6
```

Set metadata on a service (-l, --label)

A label is a `key=value` pair that applies metadata to a service. To label a service with two labels:

```
$ docker service create \
  --name redis_2 \
  --label com.example.foo="bar"
  --label bar=baz \
  redis:3.0.6
```

For more information about labels, refer to apply custom metadata (https://docs.docker.com/engine/userguide/labels-custom-metadata/).

Add bind mounts, volumes or memory filesystems

Docker supports three different kinds of mounts, which allow containers to read from or write to files or directories, either on the host operating system, or on memory filesystems. These types are *data volumes* (often referred to simply as volumes), *bind mounts*, and *tmpfs*.

A **bind mount** makes a file or directory on the host available to the container it is mounted within. A bind mount may be either read-only or read-write. For example, a container might share its host's DNS information by means of a bind mount of the host's `/etc/resolv.conf` or a container might write logs to its host's `/var/log/myContainerLogs` directory. If you use bind mounts and your host and containers have different notions of permissions, access controls, or other such details, you will run into portability issues.

A **named volume** is a mechanism for decoupling persistent data needed by your container from the image used to create the container and from the host machine. Named volumes are created and managed by Docker, and a named volume persists even when no container is currently using it. Data in named volumes can be shared between a container and the host machine, as well as between multiple containers. Docker uses a *volume driver* to create, manage, and mount volumes. You can back up or restore volumes using Docker commands.

A **tmpfs** mounts a tmpfs inside a container for volatile data.

Consider a situation where your image starts a lightweight web server. You could use that image as a base image, copy in your website's HTML files, and package that into another image. Each time your website changed, you'd need to update the new image and redeploy all of the containers serving your website. A better solution is to store the website in a named volume which is attached to each of your web server containers when they start. To update the website, you just update the named volume.

For more information about named volumes, see Data Volumes (https://docs.docker.com/engine/tutorials/dockervolumes/).

The following table describes options which apply to both bind mounts and named volumes in a service:

Option	Required	Description
types		The type of mount, can be either `volume`, `bind`, or `tmpfs`. Defaults to `volume` if no type is specified. • `volume`: mounts a managed volume (https://docs.docker.com/engine/reference/commandline/volume_create/) into the container. • `bind`: bind-mounts a directory or file from the host into the container. • `tmpfs`: mount a tmpfs in the container
src or **source**	for `type=bind` only>	• `type=volume`: `src` is an optional way to specify the name of the volume (for example, `src=my-volume`). If the named volume does not exist, it is automatically created. If no `src` is specified, the volume is assigned a random name which is guaranteed to be unique on the host, but may not be unique cluster-wide. A randomly-named volume has the same lifecycle as its container and is destroyed when the *container* is destroyed (which is upon `service update`, or when scaling or re-balancing the service) • `type=bind`: `src` is required, and specifies an absolute path to the file or directory to bind-mount (for example, `src=/path/on/host/`). An error is produced if the file or directory does not exist. • `type=tmpfs`: `src` is not supported.
dst or **destination** or **target**	yes	Mount path inside the container, for example `/some/path/in/container/`. If the path does not exist in the container's filesystem, the Engine creates a directory at the specified location before mounting the volume or bind mount.

readonly or **ro**	The Engine mounts binds and volumes `read-write` unless `readonly` option is given when mounting the bind or volume. • `true` or `1` or no value: Mounts the bind or volume read-only. • `false` or `0`: Mounts the bind or volume read-write.
consistency	The consistency requirements for the mount; one of • `default`: Equivalent to `consistent`. • `consistent`: Full consistency. The container runtime and the host maintain an identical view of the mount at all times. • `cached`: The host's view of the mount is authoritative. There may be delays before updates made on the host are visible within a container. • `delegated`: The container runtime's view of the mount is authoritative. There may be delays before updates made in a container are visible on the host.

BIND PROPAGATION

Bind propagation refers to whether or not mounts created within a given bind mount or named volume can be propagated to replicas of that mount. Consider a mount point `/mnt` , which is also mounted on `/tmp` . The propation settings control whether a mount on `/tmp/a` would also be available on `/mnt/a` . Each propagation setting has a recursive counterpoint. In the case of recursion, consider that `/tmp/a` is also mounted as `/foo` . The propagation settings control whether `/mnt/a` and/or `/tmp/a` would exist.

The `bind-propagation` option defaults to `rprivate` for both bind mounts and volume mounts, and is only configurable for bind mounts. In other words, named volumes do not support bind propagation.

- **`shared`** : Sub-mounts of the original mount are exposed to replica mounts, and sub-mounts of replica mounts are also propagated to the original mount.
- **`slave`** : similar to a shared mount, but only in one direction. If the original mount exposes a sub-mount, the replica mount can see it. However, if the replica mount exposes a sub-mount, the original mount cannot see it.
- **`private`** : The mount is private. Sub-mounts within it are not exposed to replica mounts, and sub-mounts of replica mounts are not exposed to the original mount.
- **`rshared`** : The same as shared, but the propagation also extends to and from mount points nested within any of the original or replica mount points.
- **`rslave`** : The same as `slave` , but the propagation also extends to and from mount points nested within any of the original or replica mount points.
- **`rprivate`** : The default. The same as `private` , meaning that no mount points anywhere within the original or replica mount points propagate in either direction.

For more information about bind propagation, see the Linux kernel documentation for shared subtree (https://www.kernel.org/doc/Documentation/filesystems/sharedsubtree.txt).

OPTIONS FOR NAMED VOLUMES

The following options can only be used for named volumes (`type=volume`):

Option	Description
volume-driver	Name of the volume-driver plugin to use for the volume. Defaults to `"local"`, to use the local volume driver to create the volume if the volume does not exist.
volume-label	One or more custom metadata ("labels") to apply to the volume upon creation. For example, `volume-label=mylabel=hello-world,my-other-label=hello-mars`. For more information about labels, refer to apply custom metadata (https://docs.docker.com/engine/userguide/labels-custom-metadata/).
volume-nocopy	By default, if you attach an empty volume to a container, and files or directories already existed at the mount-path in the container (`dst`), the Engine copies those files and directories into the volume, allowing the host to access them. Set `volume-nocopy` to disable copying files from the container's filesystem to the volume and mount the empty volume. A value is optional: • `true` or `1`: Default if you do not provide a value. Disables copying. • `false` or `0`: Enables copying.
volume-opt	Options specific to a given volume driver, which will be passed to the driver when creating the volume. Options are provided as a comma-separated list of key/value pairs, for example, `volume-opt=some-option=some-value,volume-opt=some-other-option=some-other-value`. For available options for a given driver, refer to that driver's documentation.

OPTIONS FOR TMPFS

The following options can only be used for tmpfs mounts (`type=tmpfs`);

Option	Description
tmpfs-size	Size of the tmpfs mount in bytes. Unlimited by default in Linux.
tmpfs-mode	File mode of the tmpfs in octal. (e.g. `"700"` or `"0700"`.) Defaults to `"1777"` in Linux.

DIFFERENCES BETWEEN “--MOUNT” AND “--VOLUME”

The `--mount` flag supports most options that are supported by the `-v` or `--volume` flag for `docker run` , with some important exceptions:

- The `--mount` flag allows you to specify a volume driver and volume driver options *per volume*, without creating the volumes in advance. In contrast, `docker run` allows you to specify a single volume driver which is shared by all volumes, using the `--volume-driver` flag.

- The `--mount` flag allows you to specify custom metadata ("labels") for a volume, before the volume is created.
- When you use `--mount` with `type=bind`, the host-path must refer to an *existing* path on the host. The path will not be created for you and the service will fail with an error if the path does not exist.
- The `--mount` flag does not allow you to relabel a volume with `Z` or `z` flags, which are used for `selinux` labeling.

CREATE A SERVICE USING A NAMED VOLUME

The following example creates a service that uses a named volume:

```
$ docker service create \
  --name my-service \
  --replicas 3 \
  --mount type=volume,source=my-volume,destination=/path/in/container,volume-label="color=
  nginx:alpine
```

For each replica of the service, the engine requests a volume named "my-volume" from the default ("local") volume driver where the task is deployed. If the volume does not exist, the engine creates a new volume and applies the "color" and "shape" labels.

When the task is started, the volume is mounted on `/path/in/container/` inside the container.

Be aware that the default ("local") volume is a locally scoped volume driver. This means that depending on where a task is deployed, either that task gets a *new* volume named "my-volume", or shares the same "my-volume" with other tasks of the same service. Multiple containers writing to a single shared volume can cause data corruption if the software running inside the container is not designed to handle concurrent processes writing to the same location. Also take into account that containers can be re-scheduled by the Swarm orchestrator and be deployed on a different node.

CREATE A SERVICE THAT USES AN ANONYMOUS VOLUME

The following command creates a service with three replicas with an anonymous volume on `/path/in/container`:

```
$ docker service create \
  --name my-service \
  --replicas 3 \
  --mount type=volume,destination=/path/in/container \
  nginx:alpine
```

In this example, no name (`source`) is specified for the volume, so a new volume is created for each task. This guarantees that each task gets its own volume, and volumes are not shared between tasks. Anonymous volumes are removed after the task using them is complete.

CREATE A SERVICE THAT USES A BIND-MOUNTED HOST DIRECTORY

The following example bind-mounts a host directory at `/path/in/container` in the containers backing the service:

```
$ docker service create \
  --name my-service \
  --mount type=bind,source=/path/on/host,destination=/path/in/container \
  nginx:alpine
```

Set service mode (--mode)

The service mode determines whether this is a *replicated* service or a *global* service. A replicated service runs as many tasks as specified, while a global service runs on each active node in the swarm.

The following command creates a global service:

```
$ docker service create \
 --name redis_2 \
 --mode global \
 redis:3.0.6
```

Specify service constraints (--constraint)

You can limit the set of nodes where a task can be scheduled by defining constraint expressions. Multiple constraints find nodes that satisfy every expression (AND match). Constraints can match node or Docker Engine labels as follows:

node attribute	matches	example
`node.id`	Node ID	`node.id==2ivku8v2gvtg4`
`node.hostname`	Node hostname	`node.hostname!=node-2`
`node.role`	Node role	`node.role==manager`
`node.labels`	user defined node labels	`node.labels.security==high`
`engine.labels`	Docker Engine's labels	`engine.labels.operatingsystem==ubuntu 14.04`

`engine.labels` apply to Docker Engine labels like operating system, drivers, etc. Swarm administrators add `node.labels` for operational purposes by using the `docker node update` (https://docs.docker.com/engine/reference/commandline/node_update/) command.

For example, the following limits tasks for the redis service to nodes where the node type label equals queue:

```
$ docker service create \
  --name redis_2 \
  --constraint 'node.labels.type == queue' \
  redis:3.0.6
```

Specify service placement preferences (--placement-pref)

You can set up the service to divide tasks evenly over different categories of nodes. One example of where this can be useful is to balance tasks over a set of datacenters or availability zones. The example below illustrates this:

```
$ docker service create \
  --replicas 9 \
  --name redis_2 \
  --placement-pref 'spread=node.labels.datacenter' \
  redis:3.0.6
```

This uses `--placement-pref` with a `spread` strategy (currently the only supported strategy) to spread tasks evenly over the values of the `datacenter` node label. In this example, we assume that every node has a `datacenter` node label attached to it. If there are three different values of this label among nodes in the swarm, one third of the tasks will be placed on the nodes associated with each value. This is true even if there are more nodes with one value than another. For example, consider the following set of nodes:

- Three nodes with `node.labels.datacenter=east`
- Two nodes with `node.labels.datacenter=south`
- One node with `node.labels.datacenter=west`

Since we are spreading over the values of the `datacenter` label and the service has 9 replicas, 3 replicas will end up in each datacenter. There are three nodes associated with the value `east`, so each one will get one of the three replicas reserved for this value. There are two nodes with the value `south`, and the three replicas for this value will be divided between them, with one receiving two replicas and another receiving just one. Finally, `west` has a single node that will get all three replicas reserved for `west`.

If the nodes in one category (for example, those with `node.labels.datacenter=south`) can’t handle their fair share of tasks due to constraints or resource limitations, the extra tasks will be assigned to other nodes instead, if possible.

Both engine labels and node labels are supported by placement preferences. The example above uses a node label, because the label is referenced with `node.labels.datacenter`. To spread over the values of an engine label, use `--placement-pref spread=engine.labels.<labelname>`.

It is possible to add multiple placement preferences to a service. This establishes a hierarchy of preferences, so that tasks are first divided over one category, and then further divided over additional categories. One example of where this may be useful is dividing tasks fairly between datacenters, and then splitting the tasks within each datacenter over a choice of racks. To add multiple placement preferences, specify the `--placement-pref` flag multiple times. The order is significant, and the placement preferences will be applied in the order given when making scheduling decisions.

The following example sets up a service with multiple placement preferences. Tasks are spread first over the various datacenters, and then over racks (as indicated by the respective labels):

```
$ docker service create \
  --replicas 9 \
  --name redis_2 \
  --placement-pref 'spread=node.labels.datacenter' \
  --placement-pref 'spread=node.labels.rack' \
  redis:3.0.6
```

When updating a service with `docker service update` , `--placement-pref-add` appends a new placement preference after all existing placement preferences. `--placement-pref-rm` removes an existing placement preference that matches the argument.

Attach a service to an existing network (--network)

You can use overlay networks to connect one or more services within the swarm.

First, create an overlay network on a manager node the docker network create command:

```
$ docker network create --driver overlay my-network

etjpu59cykrptrgw0z0hk5snf
```

After you create an overlay network in swarm mode, all manager nodes have access to the network.

When you create a service and pass the `--network` flag to attach the service to the overlay network:

```
$ docker service create \
  --replicas 3 \
  --network my-network \
  --name my-web \
  nginx

716thylsndqma81j6kkkb5aus
```

The swarm extends my-network to each node running the service.

Containers on the same network can access each other using service discovery (https://docs.docker.com/engine/swarm/networking/#use-swarm-mode-service-discovery).

Long form syntax of `--network` allows to specify list of aliases and driver options:
`--network name=my-network,alias=web1,driver-opt=field1=value1`

Publish service ports externally to the swarm (-p, --publish)

You can publish service ports to make them available externally to the swarm using the `--publish` flag. The `--publish` flag can take two different styles of arguments. The short version is positional, and allows you to specify the published port and target port separated by a colon.

```
$ docker service create --name my_web --replicas 3 --publish 8080:80 nginx
```

There is also a long format, which is easier to read and allows you to specify more options. The long format is preferred. You cannot specify the service's mode when using the short format. Here is an example of using the long format for the same service as above:

```
$ docker service create --name my_web --replicas 3 --publish published=8080,target=80 ngin
```

The options you can specify are:

Option	Short syntax	Long syntax	Description
published and target port	`--publish 8080:80`	`--publish published=8080,target=80`	The target port within the container and the port to map it to on the nodes, using the routing mesh (`ingress`) or host-level networking. More options are available, later in this table. The key-value syntax is preferred, because it is somewhat self-documenting.
mode	Not possible to set using short syntax.	`--publish published=8080,target=80,mode=host`	The mode to use for binding the port, either `ingress` or `host`. Defaults to `ingress` to use the routing mesh.

Option	Short syntax	Long syntax	Description
protocol	`--publish 8080:80/tcp`	`--publish published=8080,target=80,protocol=tcp`	The protocol to use, `tcp` , `udp`, or `sctp`. Defaults to `tcp`. To bind a port for both protocols, specify the `-p` or `--publish` flag twice.

When you publish a service port using `ingress` mode, the swarm routing mesh makes the service accessible at the published port on every node regardless if there is a task for the service running on the node. If you use `host` mode, the port is only bound on nodes where the service is running, and a given port on a node can only be bound once. You can only set the publication mode using the long syntax. For more information refer to Use swarm mode routing mesh (https://docs.docker.com/engine/swarm/ingress/).

Provide credential specs for managed service accounts (Windows only)

This option is only used for services using Windows containers. The `--credential-spec` must be in the format `file://<filename>` or `registry://<value-name>` .

When using the `file://<filename>` format, the referenced file must be present in the `CredentialSpecs` subdirectory in the docker data directory, which defaults to `C:\ProgramData\Docker\` on Windows. For example, specifying `file://spec.json` loads `C:\ProgramData\Docker\CredentialSpecs\spec.json` .

When using the `registry://<value-name>` format, the credential spec is read from the Windows registry on the daemon's host. The specified registry value must be located in:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Virtualization\Containers\CredentialSpec
```

Create services using templates

You can use templates for some flags of `service create` , using the syntax provided by the Go's text/template (http://golang.org/pkg/text/template/) package.

The supported flags are the following :

- `--hostname`
- `--mount`

- `--env`

Valid placeholders for the Go template are listed below:

Placeholder	Description
`.Service.ID`	Service ID
`.Service.Name`	Service name
`.Service.Labels`	Service labels
`.Node.ID`	Node ID
`.Node.Hostname`	Node Hostname
`.Task.ID`	Task ID
`.Task.Name`	Task name
`.Task.Slot`	Task slot

TEMPLATE EXAMPLE

In this example, we are going to set the template of the created containers based on the service's name, the node's ID and hostname where it sits.

```
$ docker service create --name hosttempl \
                        --hostname="{{.Node.Hostname}}-{{.Node.ID}}-{{.Service.Name}}"\
                         busybox top

va8ew30grofhjoychbr6iot8c

$ docker service ps va8ew30grofhjoychbr6iot8c

ID            NAME         IMAGE
wo41w8hg8qan  hosttempl.1  busybox:latest@sha256:29f5d56d12684887bdfa50dcd29fc31eea4aaf4ad

$ docker inspect --format="{{.Config.Hostname}}" 2e7a8a9c4da2-wo41w8hg8qanxwjwsg4kxpprj-ho

x3ti0erg11rjpg64m75kej2mz-hosttempl
```

Specify isolation mode (Windows)

By default, tasks scheduled on Windows nodes are run using the default isolation mode configured for this particular node. To force a specific isolation mode, you can use the `--isolation` flag:

```
$ docker service create --name myservice --isolation=process microsoft/nanoserver
```

Supported isolation modes on Windows are:

- `default` : use default settings specified on the node running the task
- `process` : use process isolation (Windows server only)
- `hyperv` : use Hyper-V isolation

Create services requesting Generic Resources

You can narrow the kind of nodes your task can land on through the using the `--generic-resource` flag (if the nodes advertise these resources):

```
$ docker service create --name cuda \
                        --generic-resource "NVIDIA-GPU=2" \
                        --generic-resource "SSD=1" \
                        nvidia/cuda
```