**Docker Tips: Rollout and Rollback In Docker Swarm**

[[](https://lucjuggery.medium.com/?source=post_page-----7f19e2fe2cd1--------------------------------)](https://lucjuggery.medium.com/?source=post_page-----7f19e2fe2cd1--------------------------------)

[[Better Programming](https://betterprogramming.pub/?source=post_page-----7f19e2fe2cd1--------------------------------)](https://betterprogramming.pub/?source=post_page-----7f19e2fe2cd1--------------------------------)

[Luc Juggery](https://lucjuggery.medium.com/?source=post_page-----7f19e2fe2cd1--------------------------------)

·

Follow

Published in

[Better Programming](https://betterprogramming.pub/?source=post_page-----7f19e2fe2cd1--------------------------------)

·

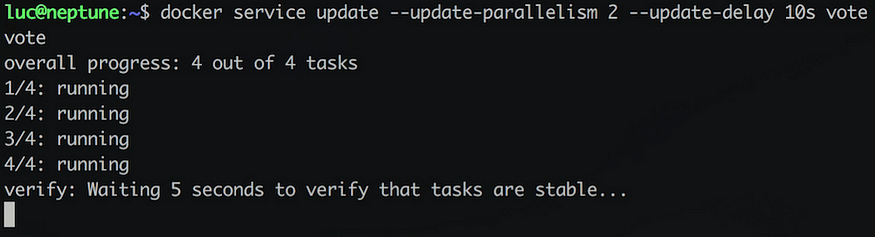
6 min read

·

Oct 1, 2018

516

1



**TL;DR**

Thanks to a couple of options, [Docker Swarm](https://docs.docker.com/engine/swarm/" \t "_blank) makes it really easy to perform a rolling update on a running service.

**Creating a Swarm**

The following command is the only thing needed to move the Docker daemon into Swarm mode.

$ docker swarm init

Note: if you have several private IPs, you need to use one of them and feed the --advertise-addr with it. The above command would then be:

$ docker swarm init --advertise-addr MY\_IP

Once the Swarm is created we could add additional nodes but this is not useful in this article — a single-node Swarm will be totally fine.

Disclaimer: do not run a single-node Swarm in production through.

**Deploying a service**

On a Swarm, a service can be either deployed manually or through a stack file. Let’s use the first option and create a service based on the instavote/vote image.

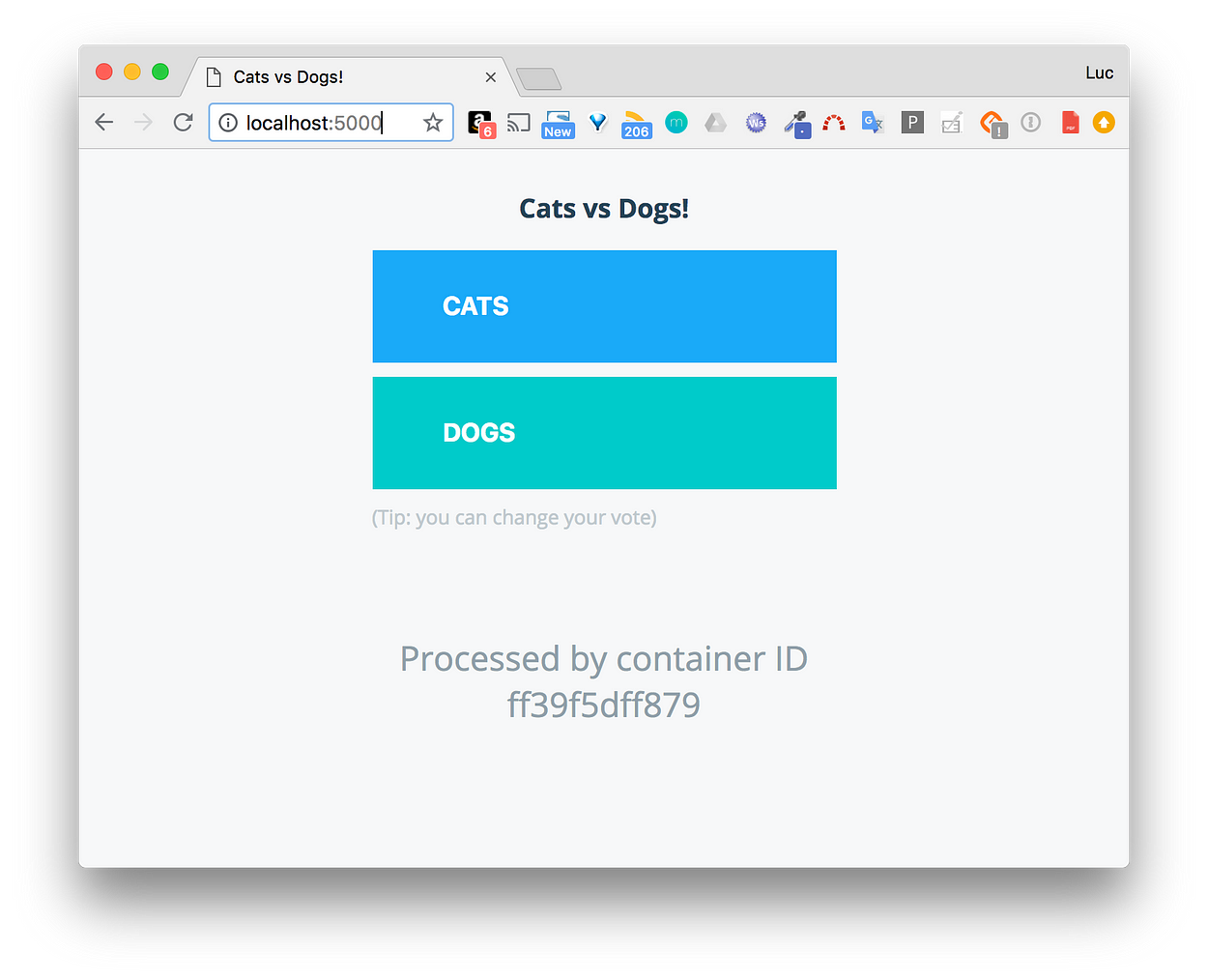


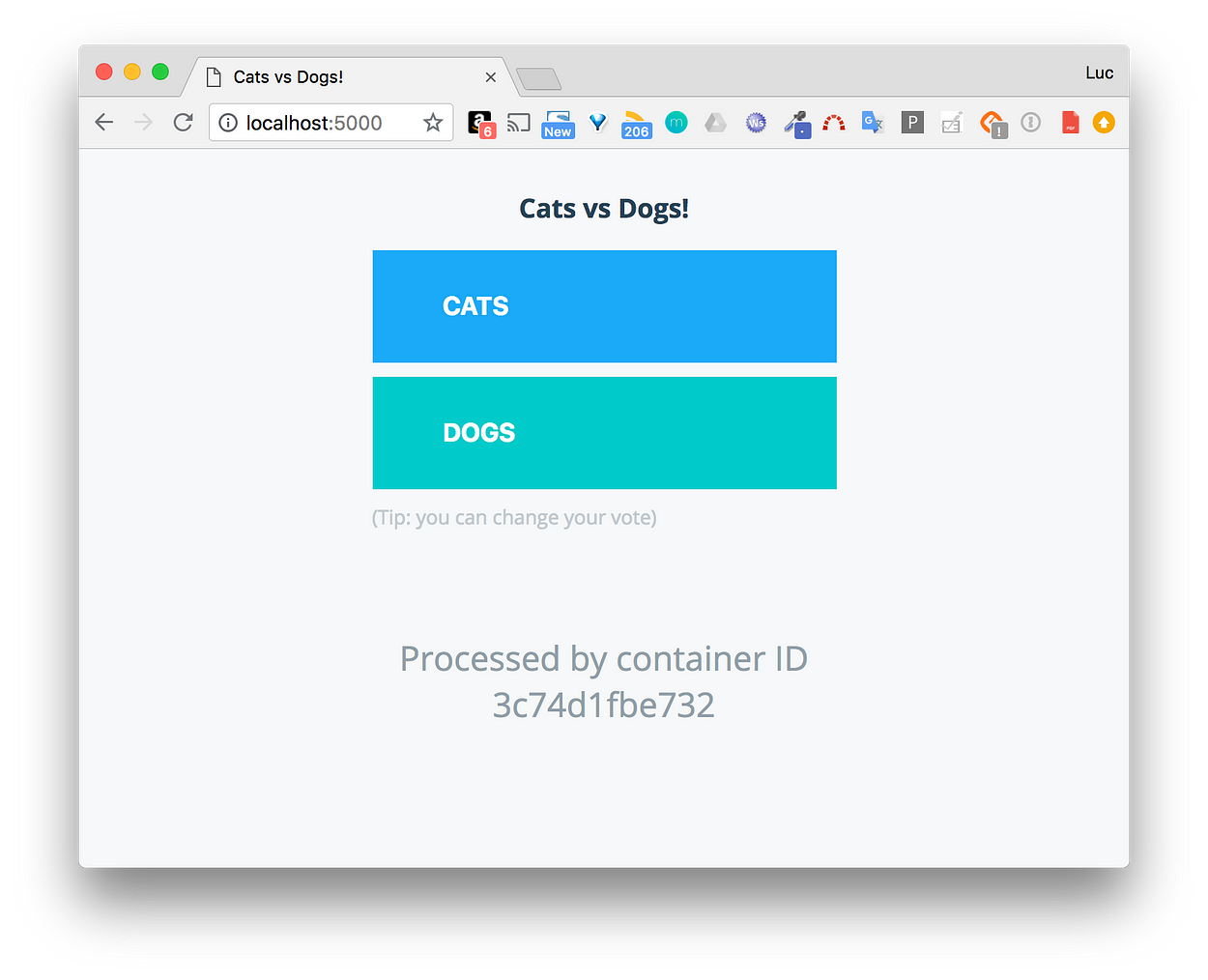
The command run above is the following one.

$ docker service create \   
--name vote \  
--replicas 4 \  
--publish 5000:80 \  
instavote/vote

It specifies 4 replicas for the service. Behind the hood, this means four tasks (a task runs a container) of the vote service are now running on the Swarm. When sending a request to the vote service, on port 5000, this one is load-balanced (by default in a round robin way) towards one of the four tasks.

The screenshots below illustrates 2 successive calls of the vote service — each call is handled by a different container.





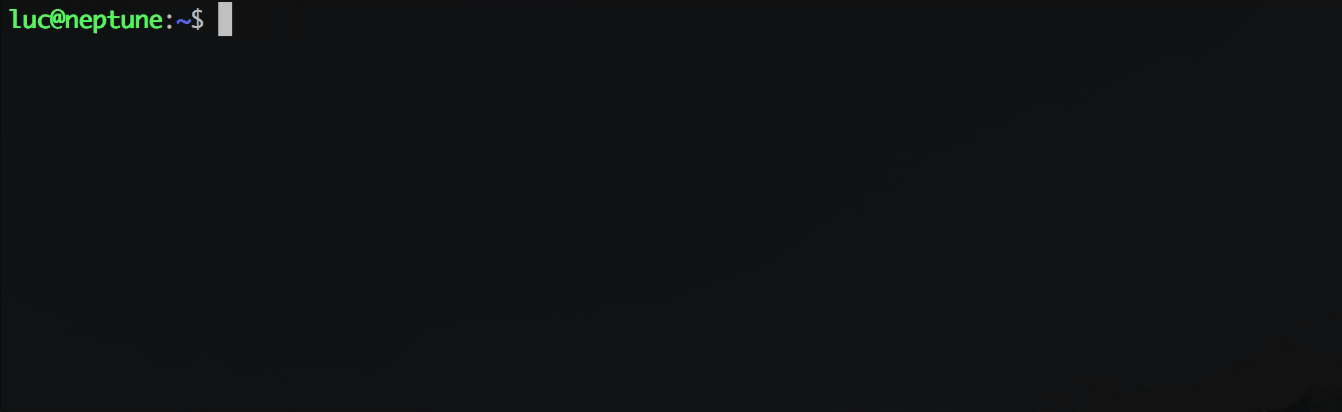
Note: this service just provides the front end of the [Voting App](https://github.com/dockersamples/example-voting-app), and does not allow us to vote, but it’s just fine to illustrate the rolling update as we will see below.

**Rolling update**

Let’s say we now need to update the service and change the original image with the instavote/vote:indent one. Updating a service is as simple as:

$ docker service update \  
--image instavote/vote:indent \  
vote

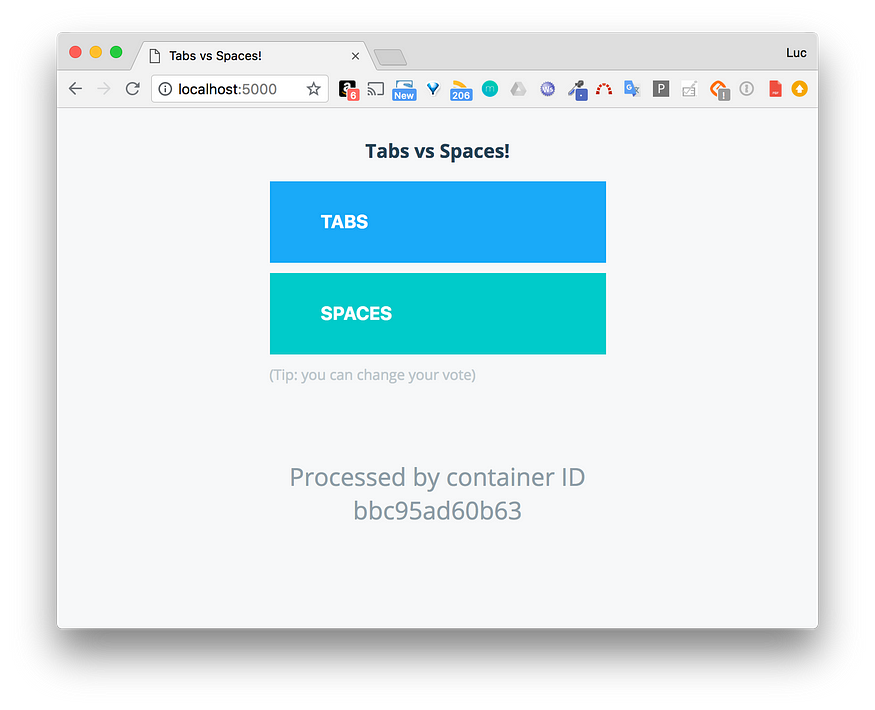
Let’s see it live:



From the above video, we notice that in order to update the service, the tasks are updated sequentially. Each one goes through the following states:

* Preparing
* Ready
* Starting
* Running

Once a task is updated, then it’s the next one’s turn and so on. This is the default behavior. When sending a request to the service, we can now vote between Tabs and Spaces.



Using the following options in the service configuration, we can customize the way the update is done:

* --update-parallelis: number of tasks to update at the same time.
* --update-delay: time to wait before updating the next batch of tasks.

Note: the [documentation](https://docs.docker.com/engine/reference/commandline/service_create/) provides a list of all the options that can be used for the service creation/update.

Let’s update the service once again, specifying that we want the tasks to be taken two by two, and that each batch should be updated 10 seconds after the previous one is done. The following command allows us to do so:

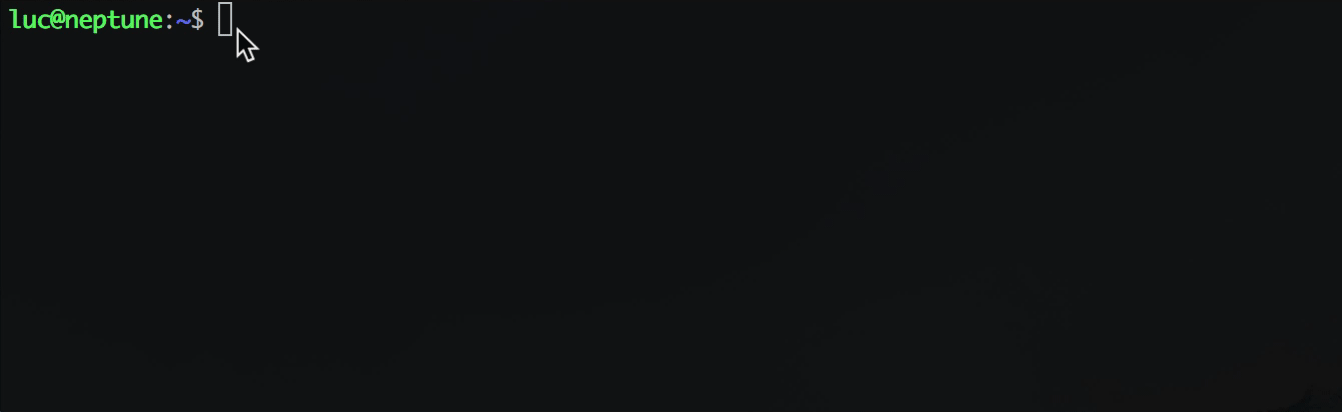
$ docker service update \  
--update-parallelism 2 \  
--update-delay 10s \  
vote



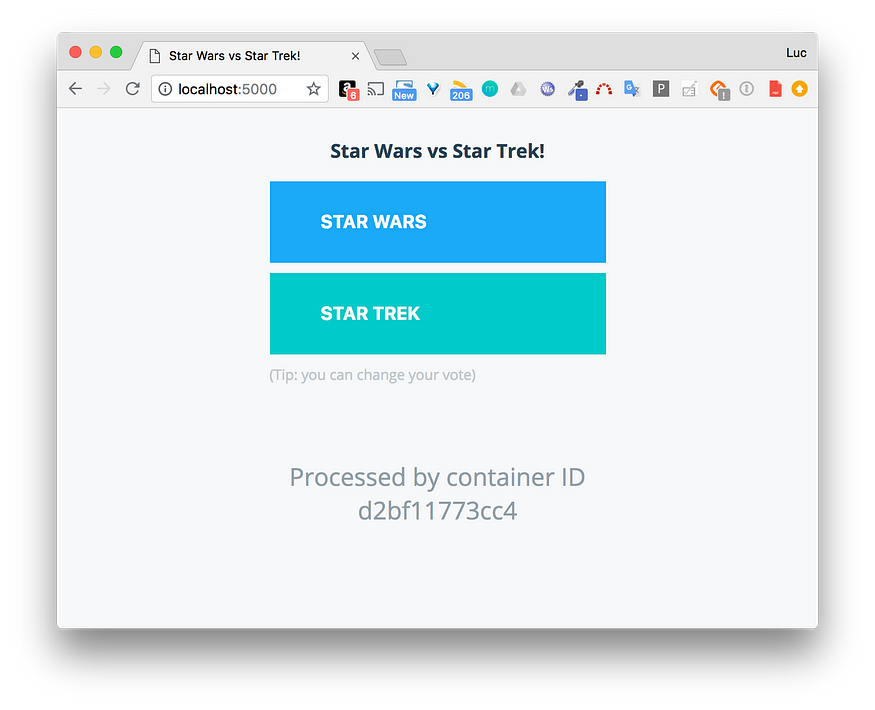
We can see the tasks are not stopped during the process; they remain in running state, as only the service parameters are updated. No need for a restart here.

If we now update the service with a new image, the rolling update will act differently. Let’s update the service with the instavote/vote:movies:

$ docker service update \  
--image instavote/vote:movies \  
vote



As expected, two tasks are updated. Ten seconds later the other two are done. We now have a choice of Sci-Fi movies.



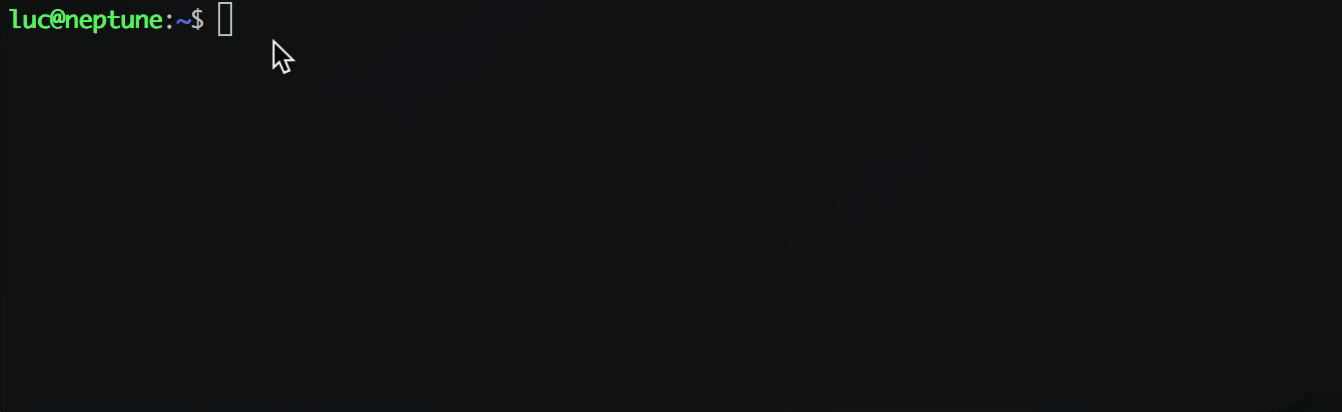
**Service inspection**

If we inspect the service, we can see the Spec and PreviousSpec keys. While the Spec key refers to the current specification of the service (the one with the movies image tag), the PreviousSpec refers to the image with the indent tag. Swarm keeps track of two levels in the service’s history.

**Rollback**

Because Swarm knows the previous specification of the vote service, it’s possible to rollback to this one with the following command:

$ docker service rollback vote

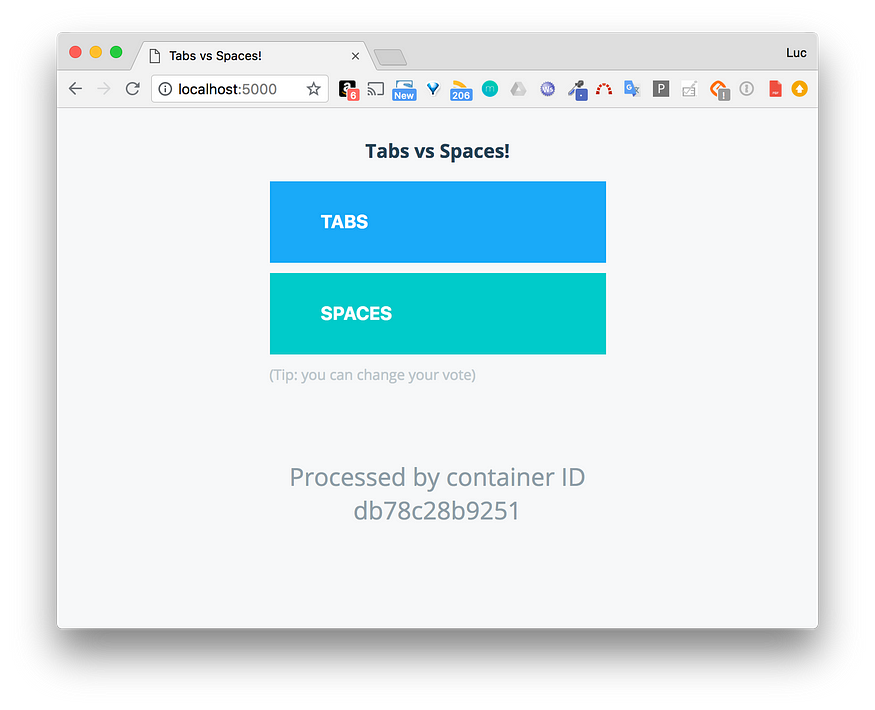


As we can see, the tasks rollback one after another. As we did not provide additional configuration, specifying the way the rollback should be done, it uses the default behavior (one task is updated after another and no delay between each update). As for the configuration of the rolling update, we could’ve used the following options in the service definition to specify the way we want the rollback to be done:

* --rollback-parallelism
* --rollback-delay

Note: the [documentation](https://docs.docker.com/engine/reference/commandline/service_create/) provides a list of all the options that can be used for the service creation/update.

After the rollback, the image instavote/vote:indent is the one used, once again, by the service.



**About automatic rollback**

In the example above, we’ve performed a manual rollback of the service. But, what if we want the rollback to be automated in case the new version is crappy? Let’s consider another example.

lucj/whoami:1.0 and lucj/whoami:2.0 are two versions of a simple API that only return the hostname of the container that handles a HTTP GET request received on the /whoami endpoint. Image with tag 1.0 runs fine, the one with tag 2.0 is buggy. The Dockerfile used to build those images is the following one:

An important thing to note here is the presence of the HEALTHCHECK instruction. It is used to verify the wellbeing of the service. Basically, it regularly checks, **from within the container**, if the /whoami endpoint is responding. Swarm uses the result of this HEALTHCHECK to perform additional actions we can configure at the service level.

Let’s create the service using the image with tag 1.0.

We specify the --update-failure-action flag and give it the rollback value so that an unsuccessful update will trigger a rollback automatically. Also, the --update-delay and --update-monitor are used to provide additional timing for the HEALTHCHECK.

Let’s update the service with the buggy lucj/whoami:2.0 image.

$docker service update --image lujs/whoami:2.0 whoami

image lujs/whoami:2.0 could not be accessed on a registry to record

its digest. Each node will access lujs/whoami:2.0 independently,

possibly leading to different nodes running different

versions of the image.

whoami

overall progress: rolling back update: 2 out of 2 tasks

1/2: running [==================================================>]

2/2: running [==================================================>]

rollback: update rolled back due to failure or early termination of task 0i0fkoktlmahdcm33pq1a5bn3

verify: Service converged



Because the first task cannot be updated (health check failed), a rollback is automatically done. In the process, the second task was not impacted, ensuring the service is still available. If we had specified only one replica, the service would be down during the update process, and obviously this is something we want to avoid.

**Summary**

I hope this article was useful in illustrating the rolling update capability of Docker Swarm. It’s considered best practice to specify the update configuration options, when defining a service in order to minimize the downtime.