This tutorials showcases the [config](https://github.com/moby/moby/pull/32336) swarm feature that allow config objects to be attached to services. Config files can be mounted inside services’ containers, avoiding the need to bake configuration into images.

Configuration files are similar to secrets, and in fact the CLI and API show few differences between the two. The principal differences so far are:

* Secrets are always redacted at the API level, so the payload cannot be obtained through an API call after they are created.
* Secrets are restricted to the /run/secrets directory inside the container, as a design choice. Config files can be mounted anywhere. Start securing your swarm services using the latest compose reference that allows to specify secrets in your application stack

Requirements

Docker 17.06+

Getting started

Initialize your swarm:

docker swarm init --advertise-addr $(hostname -i)

Let’s peak the config options:

docker config --help

As you can see the API is very similar to the [docker secrets](https://egitim.acikkaynakfikirler.com/swarm-config/2017-01-23-swarm-compose-secrets.markdown). Let’s create our first config object

echo "this is some crazy config stuff" | docker config create my\_config -

As stated before, unlike secrets, you can actually see the content of the config objects directly from the CLI. Let’s check this:

docker config inspect my\_config

Wait, what?, where’s my config?. Docker hides the config information by default to prevent unnecessary large outputs; in order to display the config value the --pretty flag needs to added

docker config inspect --pretty my\_config

Finally, let’s deploy a service using our recently created config

docker service create --name test\_cfg --config my\_config alpine cat /my\_config

You can check your service logs to see your configuration.

docker service logs test\_cfg

As you can see, as we didn’t specify any destination mountpoint, by default configs will be located at the root path. However, with configs you can place them wherever you need.

docker service create --name test\_cfg\_mount --config source=my\_config,target=/tmp/cfg alpine cat /tmp/cfg

Same as before, check your service logs to see the expected configuration:

docker service logs test\_cfg\_mount