Some Context on Docker Contexts

Jacob Howard

Docker Captain
Founder @ Mutagen

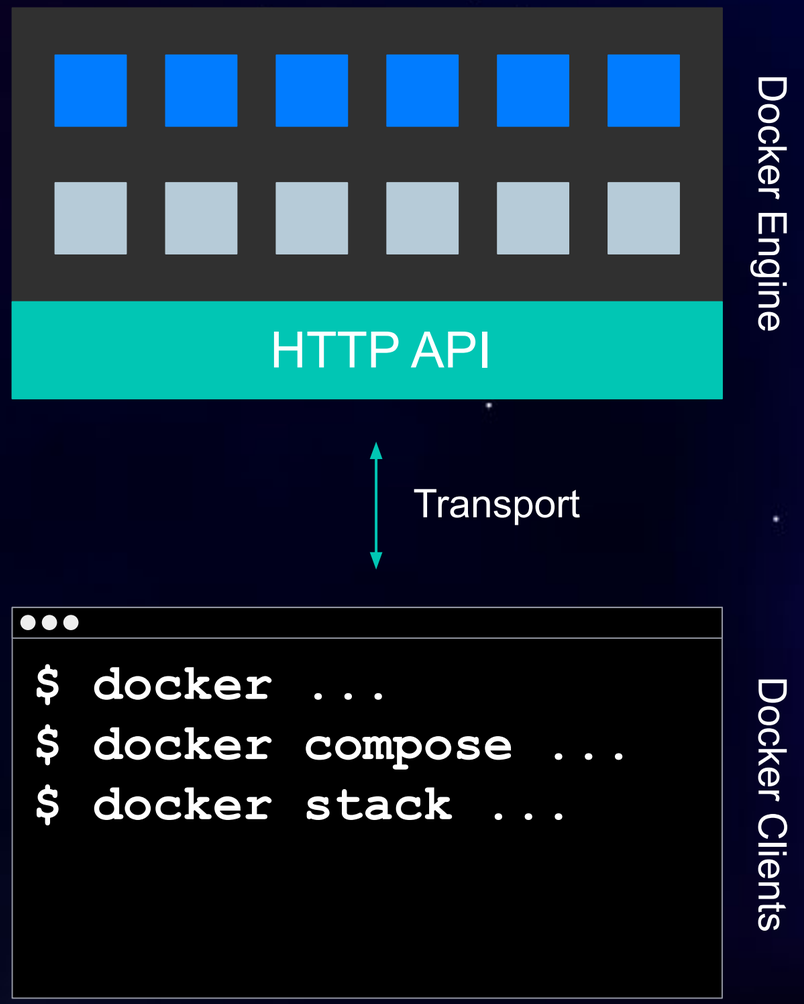
@xenoscopic

docker

The Docker Engine Architecture

A client/server architecture built on HTTP

- Not a monolithic component
- Uses a **RESTful HTTP API**
- **Many different clients**, including:
  - Docker CLI
  - Docker Compose
  - Docker Desktop Dashboard
- **Multiple transports**, including Unix Domain Sockets, Windows Named Pipes, TCP, TCP+TLS, and SSH
- Offers easy access to **remote engines**



docker

Targeting Docker Engines

- Docker clients typically default to **`/var/run/docker.sock`**
- You can override this with command line flags
- **`DOCKER_HOST`** is the traditional override mechanism, e.g.
  - **`export DOCKER_HOST=unix:///some/other/docker.sock`**
- But you might also need...
  - **`DOCKER_TLS`**
  - **`DOCKER_TLS_VERIFY`**
  - **`DOCKER_CERT_PATH`**
  - **`DOCKER_API_VERSION`**
- **This can make switching engines tedious and error-prone**

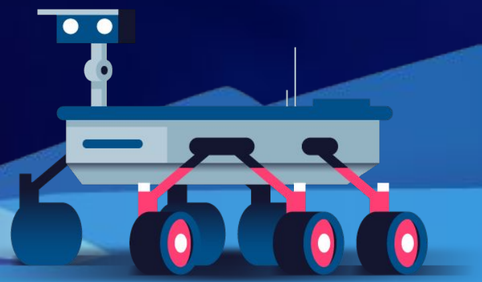
Why Multiple Docker Engines?

Building, Testing, and Deployment

- Better emulate your production environment
- Use a different machine for builds
- Develop and test in the cloud
- Manage deployment

Docker Contexts

An easy way to store connection information and work with multiple Docker Engines (and other container platforms).



Using Docker Contexts

TERMINAL

```
# Create a Docker Context
$ docker context create <name> --docker "host=<url>"

# Set a Docker Context as active
$ docker context use <name>

# Manually specify a Docker Context to use
$ docker --context <name> compose ...

# Switch back to the Docker Desktop context
$ docker context use desktop-linux

# Switch back to DOCKER_HOST-based functionality
$ docker context use default
```

`default` vs Active Context

The `default` Context

A built-in context that reverts to **`DOCKER_HOST`**-based engine targeting and configuration.

The Active Context

The currently selected context (set via **`docker context use`**) that's being used when no context is specified on the command line.

Down the Rabbit Hole...

Accessing other container platforms

- Docker Contexts provide a general abstraction layer for container platforms
- You can create contexts for:
  - Docker engines
  - Docker swarms
  - Kubernetes clusters
  - Azure Container Instances (ACI)
  - Amazon Elastic Container Service (ECS)
- You can use **`docker stack`** and **`docker compose`** to deploy projects to these platforms

So, basically, Docker Contexts are:

- A **robust** way to encapsulate configuration
- An **easy** way to switch between Docker Engines
- The only way to access **cloud integrations**

docker

Thanks for listening!

Send me your questions,
feedback, and
Docker Context life hacks!

@xenoscopic