Docker Tutorials

Docker Metadata & Labels

Introduction

- When running containers in production, it can be useful to add additional metadata relating to the container to help their management.
- *This metadata* could be related to which version of the code is running, which applications or users own the container or define special criteria such as which servers they should run on.
- This additional data is managed via *Docker Labels*. Labels allow you to define custom metadata about a container or image which can later be inspected or used as part of a filter.
- This scenario guides you though creating and querying the metadata for containers and images.

Step 1 - Docker Containers❶

- Labels can be attached to containers when they are launched via *docker run*. A container can have multiple labels attached to them at any one time.

Single Label

- To *add a single label* you use the *l =<value>* option. The example below assigns a label called user with an ID to the container. This would allow us to query for all the containers running related to that particular user.
  - *docker run -l user=12345 -d redis*

Step 1 - Docker Containers❷

- If you're adding multiple labels, then these can come from an external file. The file needs to have a label on each line, and then these will be attached to the running container.
  - *echo 'user=123461' >> labels && echo 'role=cache' >> labels*

- The *--label-file=<filename>* option will create a label for each line in the file.
  - *docker run --label-file=labels -d redis*

ShenHengheng

Step 2 - Docker Images❶

- Labelling images work in the same way as containers but are set in the Dockerfile when the image is built. When a container has launched the labels of the image will be applied to the container instance.

Single Label

- Within a *Dockerfile* you can assign a label using the LABEL instruction. Below the label *vendor* is created with the name Scrapbook.
    - *LABEL vendor=Katacoda*

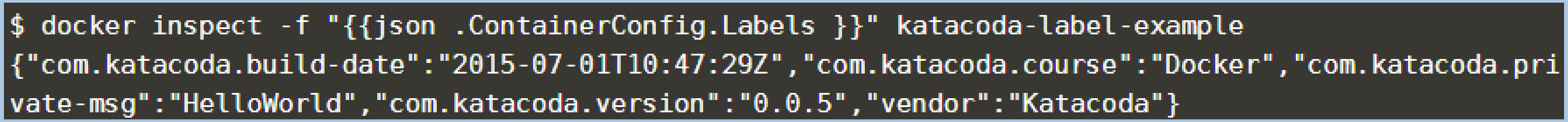
ShenHengheng

Step 2 - Docker Images❷

Multiple Labels

- If we want to assign multiple labels then, we can use the format below with a label on each line, joined using a back-slash ("\"). Notice we're using the DNS notation format for labels which are related to third party tooling.
  - *LABEL vendor=Katacoda \ com.katacoda.version=0.0.5 \ com.katacoda.build-date=2016-07-01T10:47:29Z \ com.katacoda.course=Docker*

Step 3 - Inspect

- Labels and Metadata are only useful if you can view/query them later. The first approach to viewing all the labels for a particular container or image is by using *docker inspect*.
  - *docker inspect rd*
- Using the *-f* option you can filter the JSON response to just the Labels section we're interested in.
  - *docker inspect -f "{{json .Config.Labels }}" rd*
  - 

```
$ docker inspect -f "{{json .ContainerConfig.Labels }}" katacoda-label-example
{"com.katacoda.build-date":"2015-07-01T10:47:29Z","com.katacoda.course":"Docker","com.katacoda.pri
vate-msg":"HelloWorld","com.katacoda.version":"0.0.5","vendor":"Katacoda"}
```

Step 4 - Query By Label

Filtering Containers

- *docker ps --filter "label=user=scrapbook“*

Filtering Images

- *docker images --filter "label=vendor=Katacoda"*

Step 5 - Daemon labels

- Labels are not only applied to images and containers but also the Docker Daemon itself. When you launch an instance of the daemon, you can assign it labels to help identify how it should be used, for example, if it's a development or production server or if it's more suited to particular roles such running databases.

  -
    ```
    docker -d \
    -H unix:///var/run/docker.sock \
    --label com.katacoda.environment="production" \
    --label com.katacoda.storage="ssd"
    ```

ShenHengheng

Reference

- https://www.katacoda.com/courses/docker/15