**DOCKER METADATA AND LABELS**

STEP-1: DOCKER CONTAINERS

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.

Notice in this example, because we're using the labels are for use with the CLI, and not an automated tool, we're not using the DNS notation format.

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

External file:

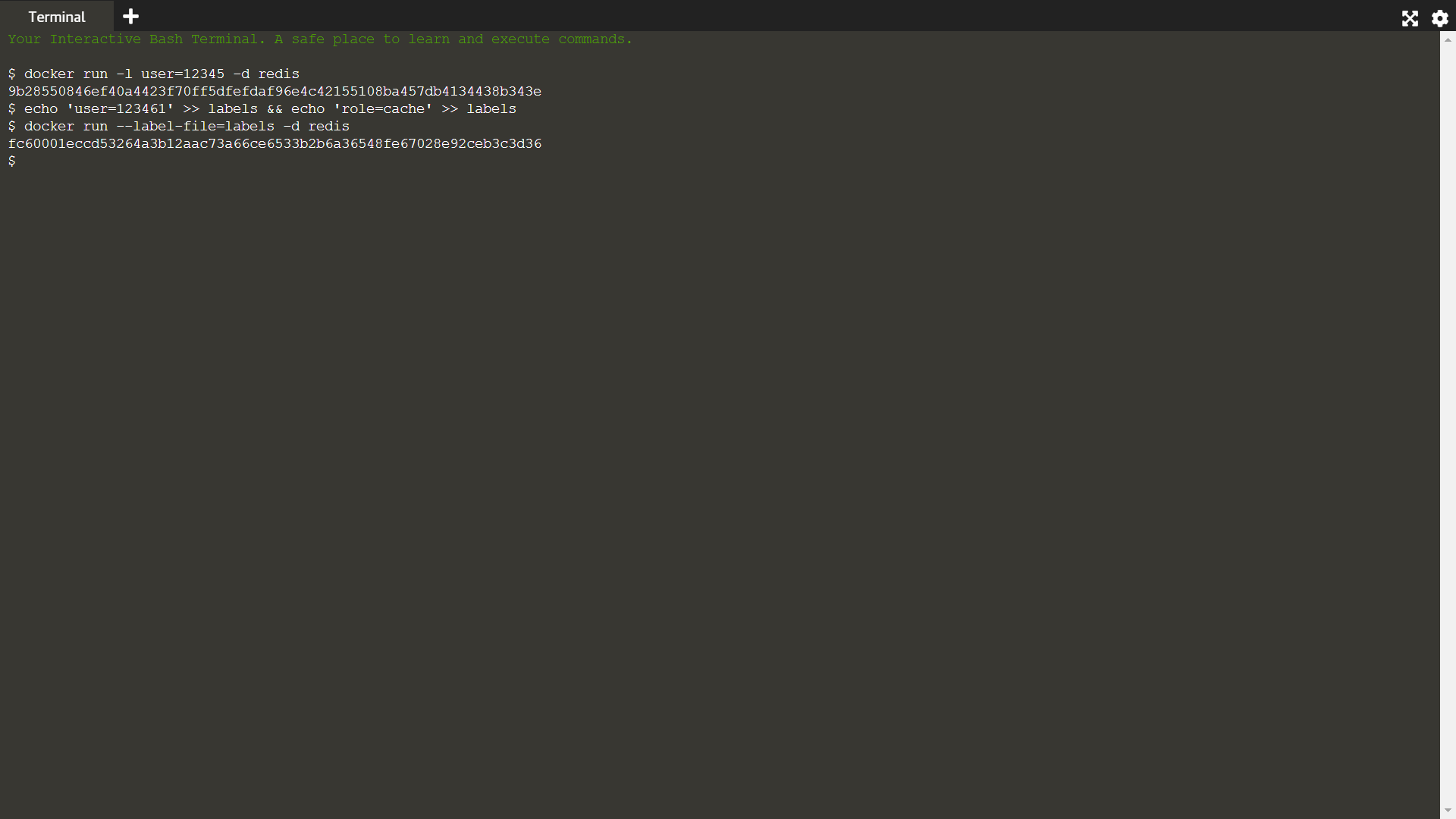
If you're adding multiple labels, then these can come from an external file. The file needs to have a label on each line, and then these will be attached to the running container.

This line creates two labels in the file, one for the user and the second assigning a role.

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



STEP-2: DOCKER IMAGES:

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

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

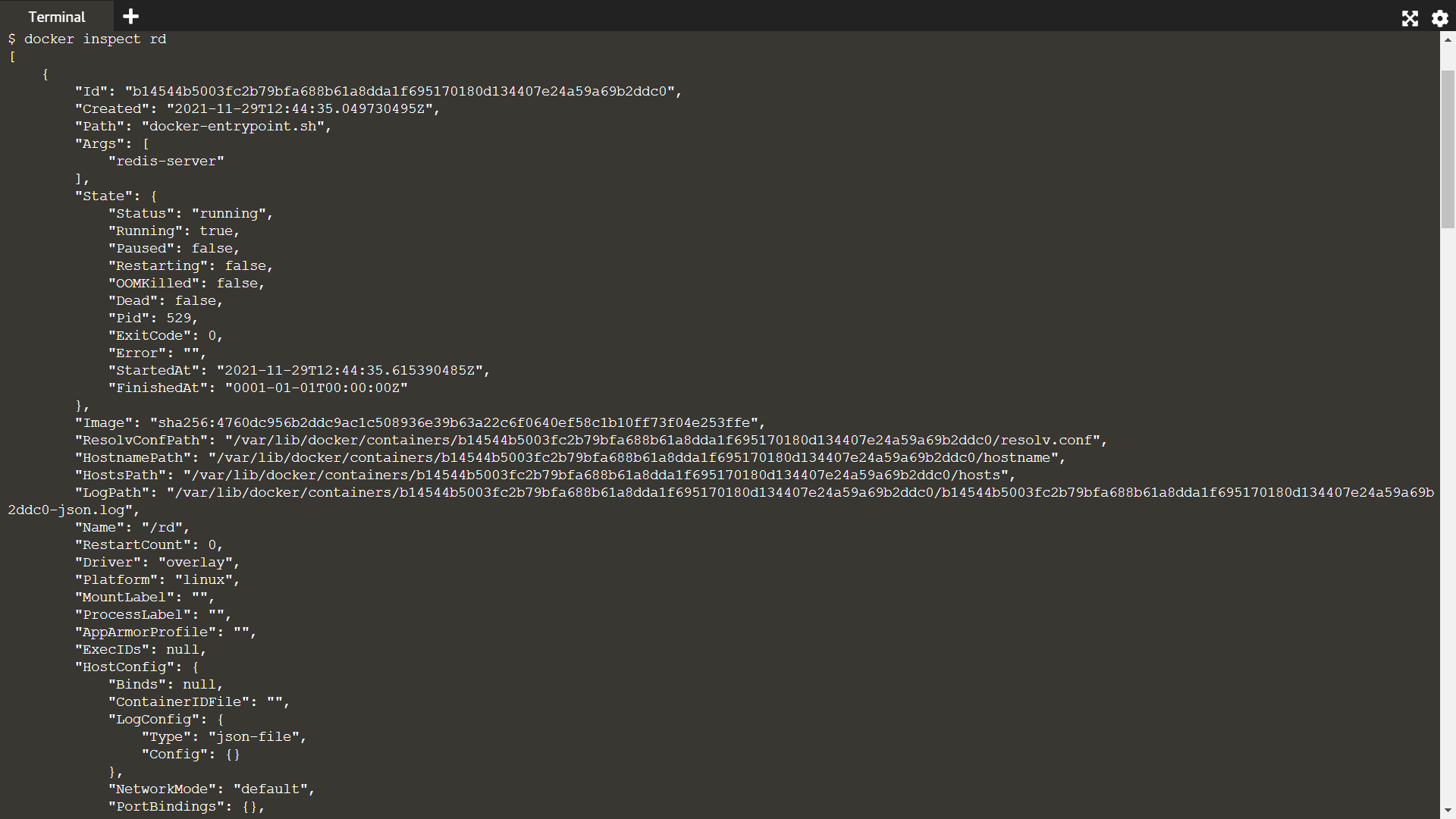
STEP-3: INSPECT

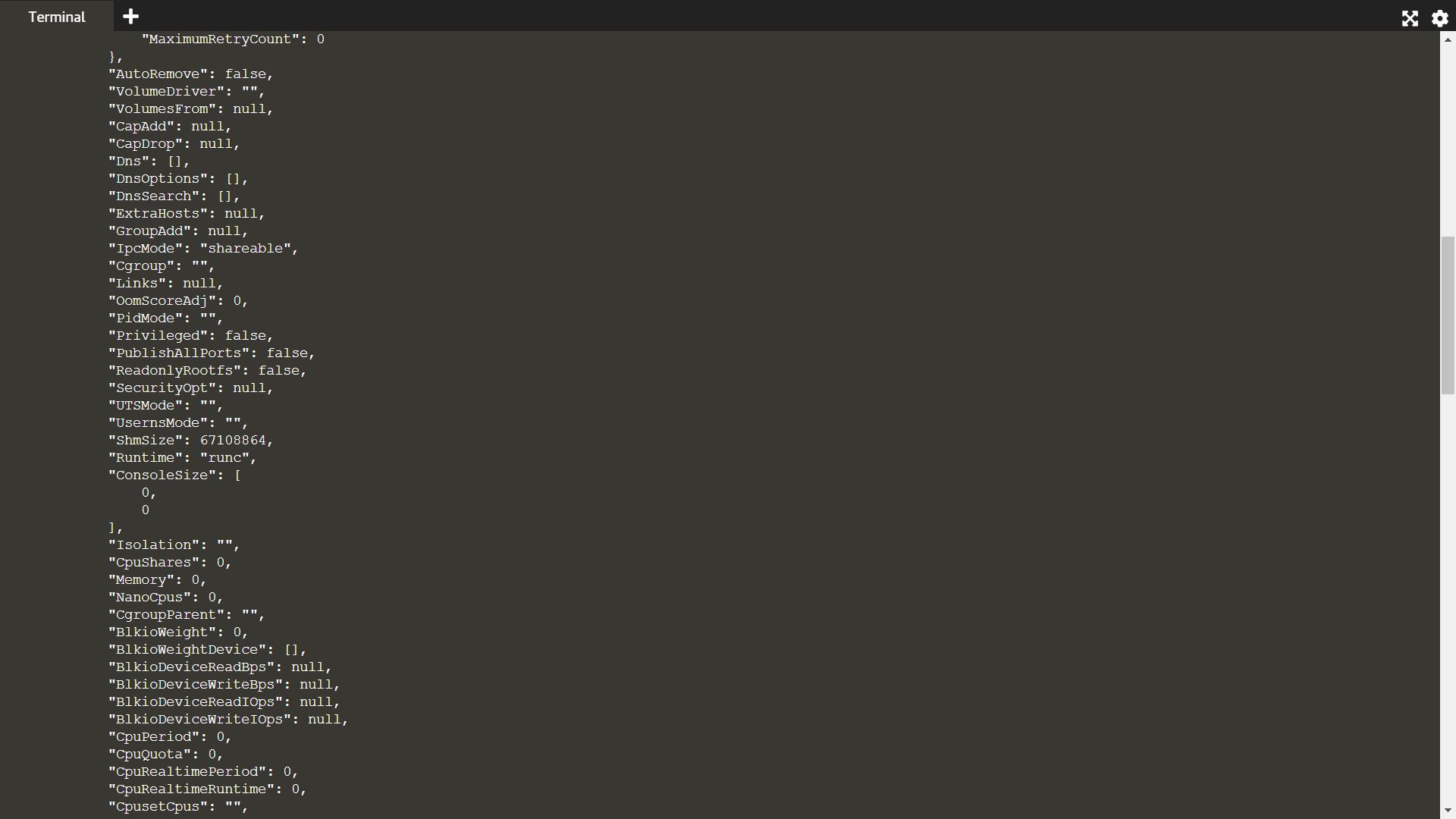
Labels and Metadata are only useful if you can view/query them later. The first approach to viewing all the labels for a particular container or image is by using docker inspect

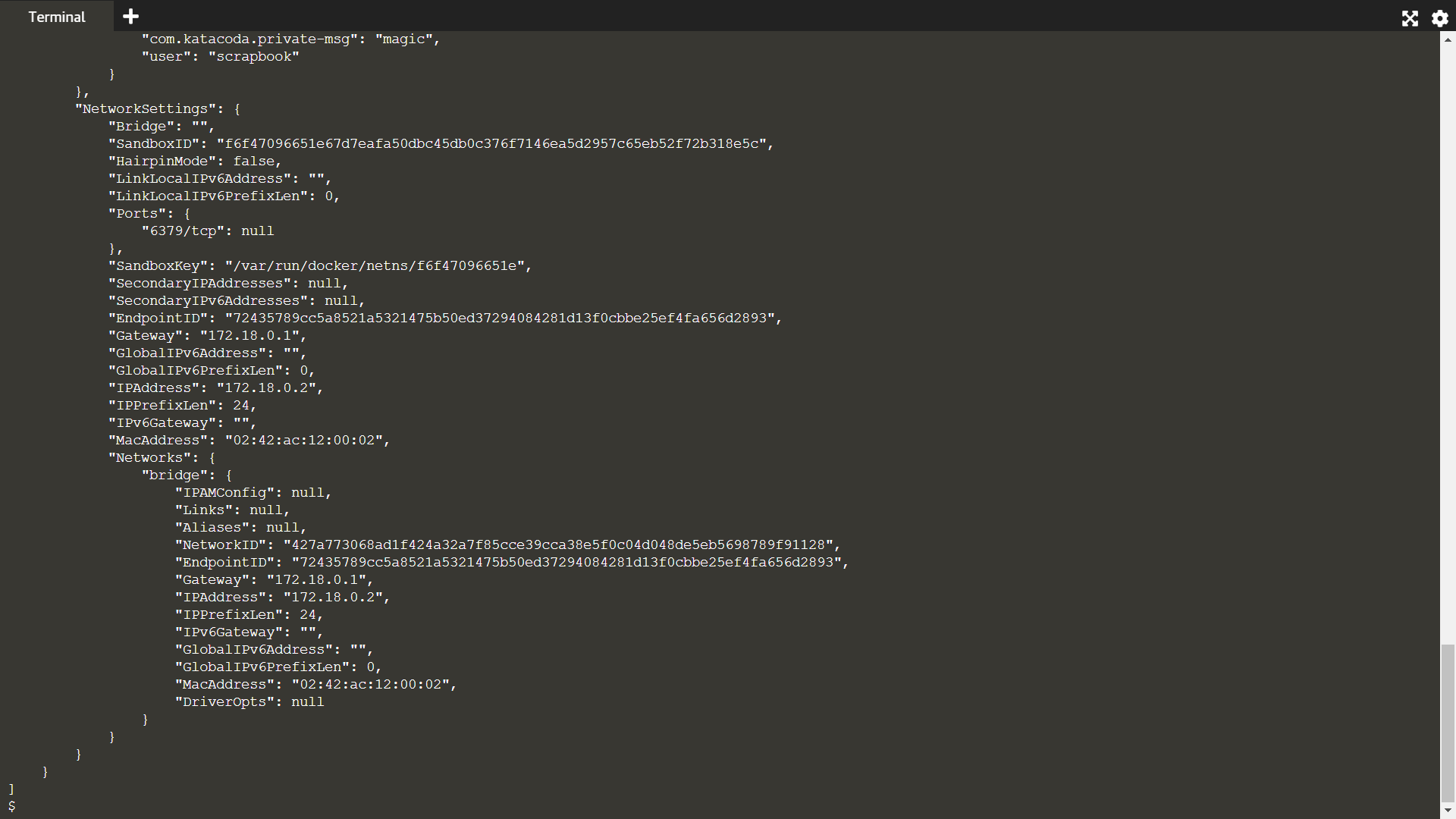
CONTAINER

:By the running container's friendly name or hash id, you can query all of it's metadata.

docker inspect rd







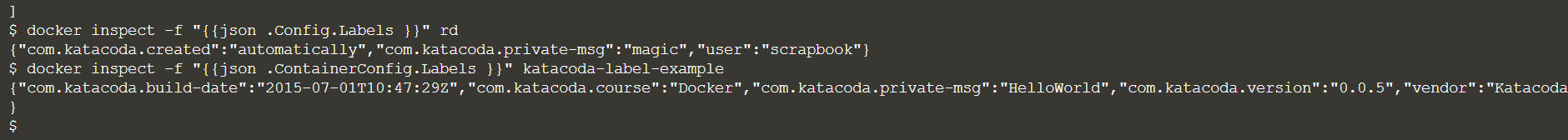
Using the -f option you can filter the JSON response to just the Labels section we're interested in.

docker inspect -f "{{json .Config.Labels }}" rd

IMAGE:

Inspecting images works in the same way however the JSON format is slightly different, naming it ContainerConfig instead of Config.

docker inspect -f "{{json .ContainerConfig.Labels }}" katacoda-label-example



STEP-4: QUERY BY LABEL:

FILTERING CONTAINERS:

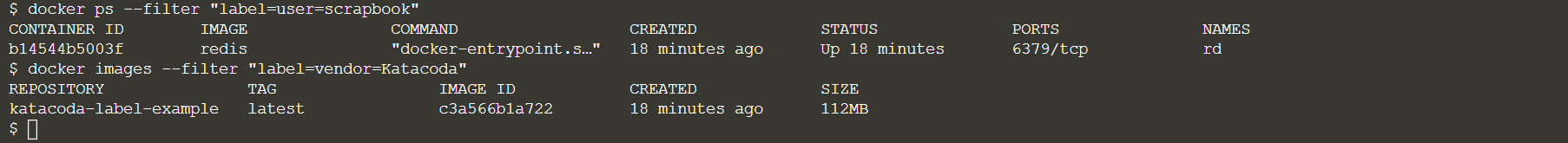
The docker ps command allows you to specify a filter based on a label name and value. For example, the query below will return all the containers which have a user label key with the value katacoda.

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

FILTERING IMAGES:

The same filter approach can be applied to images based on the labels used when the image was built.

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



STEP-5: DAEMON LABELS:

We'll explore more about customising Docker's configuration and how labels are used in future scenarios, but as a taster, the syntax is below.

docker -d \

-H unix:///var/run/docker.sock \

--label com.katacoda.environment="production" \

--label com.katacoda.storage="ssd"