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.

Recommended Guidelines
There are recommended guidelines to follow when working with labels to ensure they're consistent and manageable as defined by Docker.

Firstly, all tools should prefix their keys with the reverse DNS notation of a domain controlled by the author. For example, com.katacoda.some-label.

Secondly, if you're creating labels for CLI use, then they should follow the DNS notation making it easier for users to type.

Finally, keys should only consist of lower-cased alphanumeric characters, dots and dashes (for example, [a-z0-9-.])

**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

```
$ docker run -l user-12345 -d redis
719013536bf94d9dfb3edbef299ba5fb53571b5410da1d671148e8f70ef27d9e
$
```

**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

```
CMD ["sh", "-c", "/cmd.sh"]
$ echo 'user=123461' >> labels && echo 'role=cache' >> labels
$ cat labels
user=123461
role=cache
user=123461
role=cache
$ docker run --label-file=labels -d redis
ef1571601da032d486cdadd4d479767844f061a79e164b0b321ea1538ff910ed
$
```

**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.

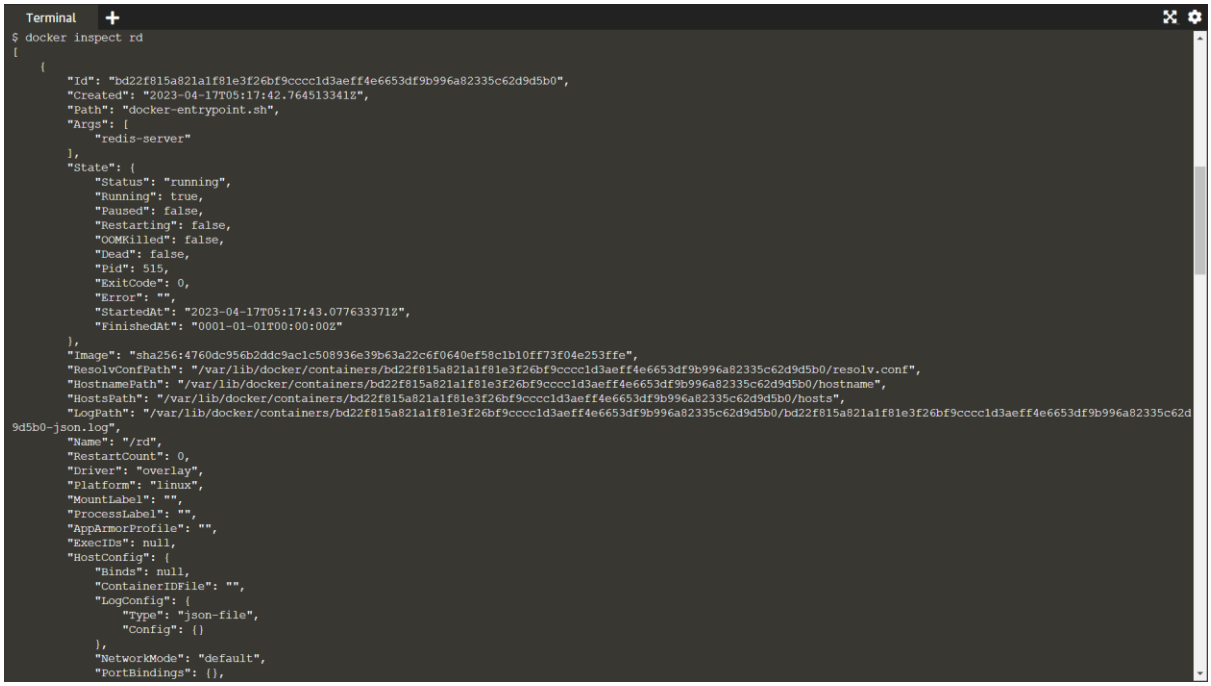
LABEL vendor=Katacoda \ com.katacoda.version=0.0.5 \ com.katacoda.build-date=2016-07-01T10:47:29Z \ com.katacoda.course=Docker

```
ef1571601da032d486cdadd4d479767844f061a79e164b0b321ea15
$ cat Dockerfile
FROM ubuntu
COPY cmd.sh /cmd.sh
LABEL vendor=Katacoda
LABEL com.katacoda.version=0.0.5
LABEL com.katacoda.build-date=2015-07-01T10:47:29Z
LABEL com.katacoda.private-msg=HelloWorld
LABEL com.katacoda.course=Docker
CMD ["sh", "-c", "/cmd.sh"]
```

**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.

```
CMD ["sh", "-c", "/cmd.sh"]
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
ef1571601da0        redis               "docker-entrypoint.s…"   3 minutes ago       Up 3 minutes        6379/tcp            frosty_wing
719013536bf9        redis               "docker-entrypoint.s…"   6 minutes ago       Up 6 minutes        6379/tcp            friendly_kapitsa
bd22f815a821        redis               "docker-entrypoint.s…"   10 minutes ago      Up 10 minutes       6379/tcp            rd
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
katacoda-label-example  latest              6d2d7592ddc4        10 minutes ago      112MB
redis                   latest              4760dc956b2d        5 years ago         107MB
ubuntu                  latest              f975c5035748        5 years ago         112MB
alpine                  latest              3fd9065eaf02        5 years ago         4.14MB
$
```

Container
By providing 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

```
$ docker inspect -f "{{ .Config.Labels }}" rd
map[com.katacoda.created:automatically com.katacoda.private-msg:magic user:scrapbook]
$ docker inspect -f "{{ json .Config.Labels }}" rd
{"com.katacoda.created":"automatically","com.katacoda.private-msg":"magic","user":"scrapbook"}
$
```

**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

These labels will remain even if the image has been untagged. When an image is untagged, it will have the name <none>.

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

**Step 4 - Query By Label**
While inspecting individual containers and images provides you with more context, on a production running potentially thousands of containers, it's useful to limit the responses to the containers you're interested in.

**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"

```
$ docker ps --filter "label=user=scrapbook"
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
bd22f815a821        redis               "docker-entrypoint.s…"   25 minutes ago      Up 25 minutes       6379/tcp            rd
$
```

**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"

```
$ docker images --filter "label=vendor=Katacoda"
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
katacoda-label-example  latest              6d2d7592ddc4        30 minutes ago      112MB
$
```

Note
When querying both the label key name and value are case sensitive. This is why it's important to follow a consistent pattern and the recommended guidelines.

**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.

**In the context of Docker, the "daemon"** refers to the background process that runs on the host machine and manages Docker objects such as images, containers, networks, and volumes.

The Docker daemon is responsible for receiving and executing commands from the Docker client, which is usually a command-line interface or a graphical user interface. When a user issues a Docker command, such as "docker run" or "docker build", the client sends the command to the daemon, which then performs the requested operation.

The Docker daemon runs as a system service or process and can be managed using system tools such as systemd or Upstart, depending on the operating system.

The Docker daemon listens for incoming requests on a Unix socket or a network port, which can be configured using the Docker Engine API. By default, the daemon listens on a Unix socket at /var/run/docker.sock.

The Docker daemon also manages security features such as user namespaces, container isolation, and image verification to ensure that containers are executed in a safe and secure environment.

Overall, the Docker daemon is a critical component of the Docker platform, and understanding its role and operation is essential for using Docker effectively.

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"
```

In this scenario we explored how you can use labels to assign metadata to your Docker containers and images. This metadata can provide additional context around individual items or as use with a filter to find the information you're looking for.

**Recommended Guidelines**
As mentioned at the beginning, there are a number of recommended guidelines to follow when working with labels to ensure they're consistent and manageable as defined by Docker.

Firstly, all tools should prefix their keys with the reverse DNS notation of a domain controlled by the author. For example, com.joinscrapbook.some-label.

Secondly, if you're creating the labels for CLI use then it they should not follow the DNS notation making it easier for users to type.

Finally, keys should only consist of lower-cased alphanumeric characters, dots and dashes (for example, [a-z0-9-.])