3/02/2023 to 06/02/2023

**Container images:**

* Navigate container registries to find and manage container images.
* Custom container images: Build custom container images to containerize applications.
* Persisting data: Build persistent databases.
* Container networking: Describe basic container networking and how to access containerized services.
* Troubleshooting containers: Analyze container logs and configure a remote debugger

[root@podman ~]# podman login registry.redhat.io

Username: gummaharish66

Password:

Login Succeeded!

[root@podman ~]# podman pull registry.redhat.io/ubi8/ubi:8.6

Trying to pull registry.redhat.io/ubi8/ubi:8.6...

Getting image source signatures

Checking if image destination supports signatures

Copying blob 8abc86061280 done

Copying config 4d15d5a0ea done

Writing manifest to image destination

Storing signatures

4d15d5a0eaed6365743c13e4818a1471239c2c63832dbbbdea78a73c8e2e74b8

Podman searches first in the Red Hat Registry. If the image is not found in the

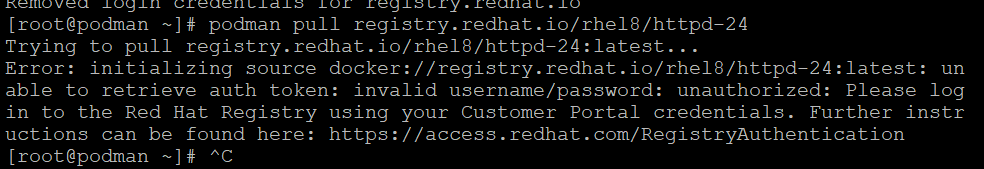
Red Hat Registry, then Podman searches in the Docker Hub.

**Configuration file for the redhat registry**

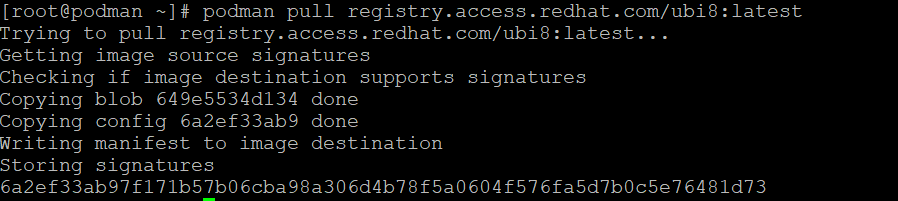
**[root@podman ~]#** cat /etc/containers/registries.conf

**Manage Registry Credentials with.**

Some registries require users to authenticate

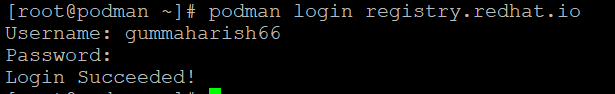


You might choose a different image that does not require authentication, such as the UBI 8 image

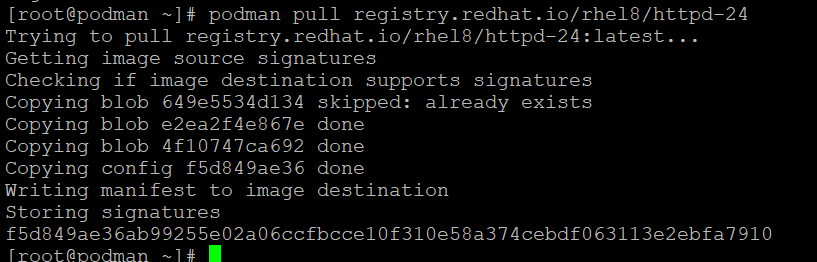
****

Alternatively, you must execute the podman-login command for the registry before you can pull

the RHEL 8 image.

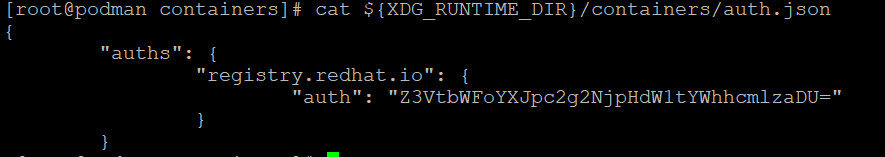
****

Now we can pull the image from the registry.redhat.io



Podman stores the credentials in this path

where the ${XDG\_RUNTIME\_DIR} refers to a directory specific to the current user



If you want encrypte that generated token by using this command

**[root@podman containers]** echo -n <token> | base64 -d

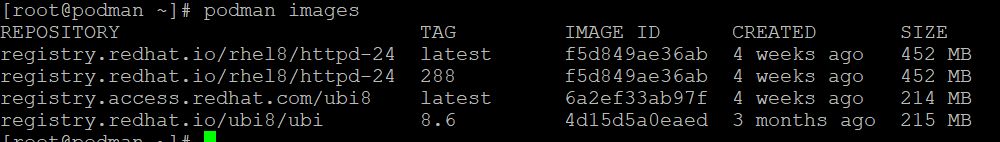


**Image Versioning and Tags**

To create additional tags for local images, use the podman image tag command.



Now can see the newly tag image



**Custom container images**

**Creating Images with Containerfiles**

A *Containerfile* lists a set of instructions to construct a container image. Once built, the image is

used to create any number of containers.

**Containerfile Instructions**

Containerfiles use a small domain-specific language (DSL) consisting of basic instructions for

crafting container images. The following are the most common instructions.

FROM

Takes the name of the base image as an argument.

WORKDIR

Sets the current working directory within the container. Later instructions run within this

directory.

COPY

Copies files from the build host into the file system of the resulting container image. Relative

paths respect the host’s current working directory, known as the build context, and the

working directory within the container as defined by WORKDIR. It is not possible to copy a

remote file by using its URL with this Containerfile instruction.

ADD

Copies files or folders from a local or remote source and adds them to the container’s file

system. If used to copy local files, those must be in the working directory. The ADD instruction

also unpacks local .tar archive files to the destination image directory.

RUN

Runs a command in the container and commits the resulting state of the container to a new

layer within the image.

ENTRYPOINT

Sets the executable to run when the container is started.

CMD

Runs a command when the container is started. This command is passed to the executable

defined by ENTRYPOINT. Base images define a default ENTRYPOINT, which is usually a shell

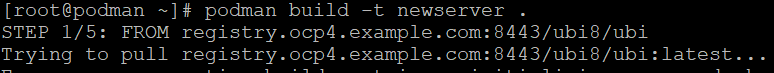
executable, such as Bash.

Create a new file called Containerfile with the following contents:



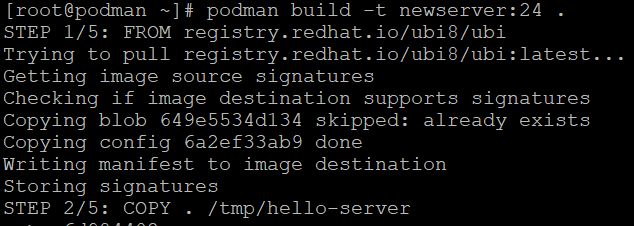
After creating the container file we need to build it.

[root@podman ~]# podman build -t <name of image> <pathof the created file>



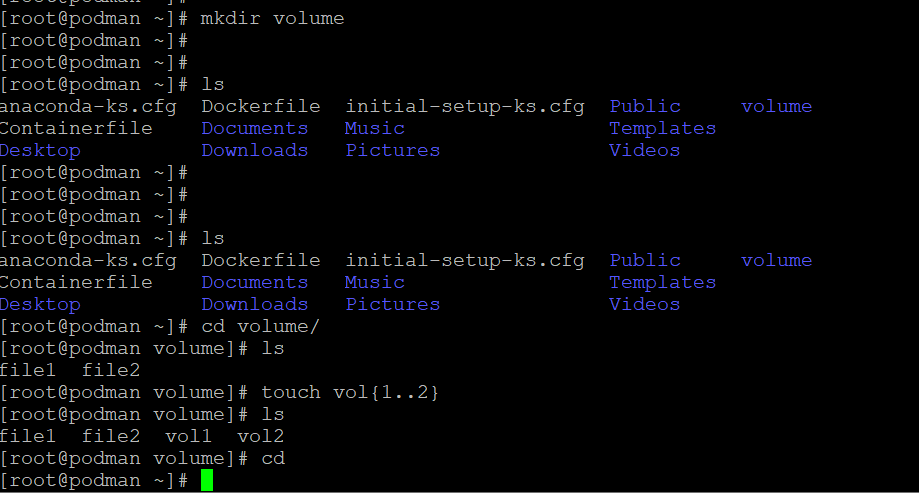
Build a container image tagged newserver:24 by using the Containerfile you

created.



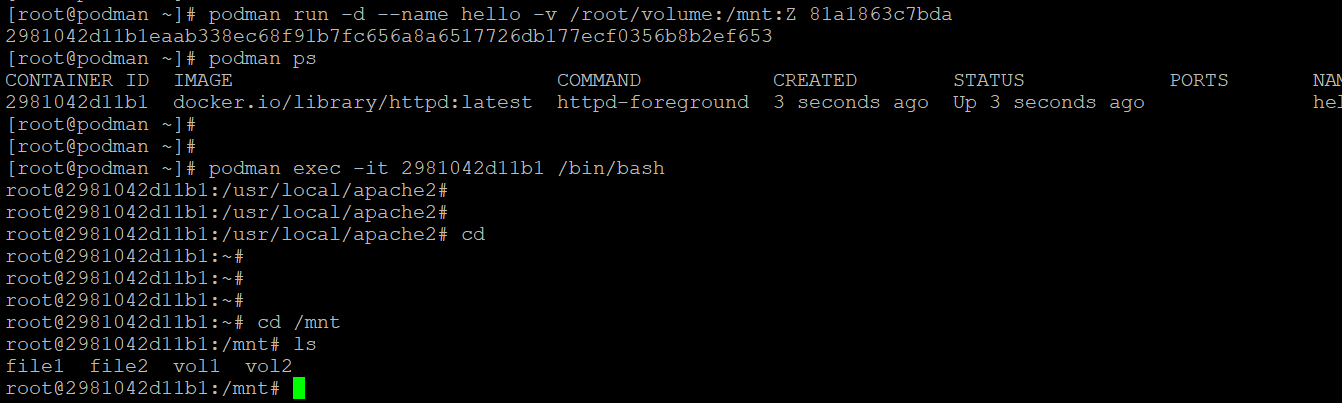
**Persisting data**

We need To create a directory

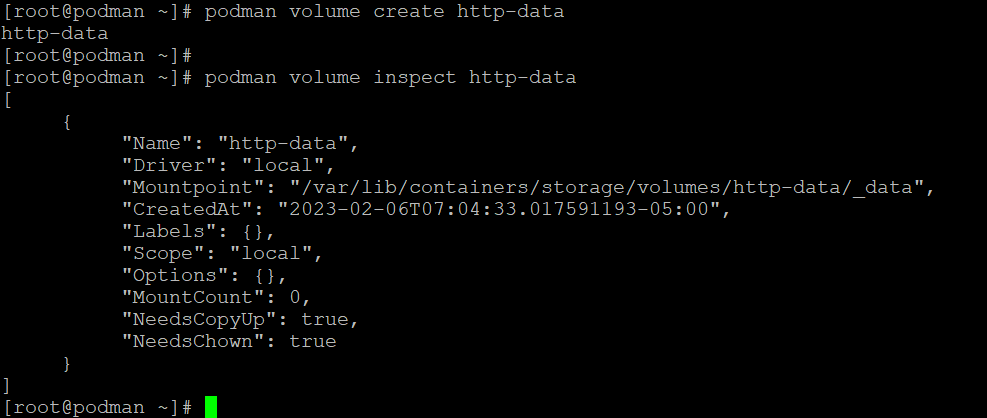


When we are running a container we need to attach the volume what we created

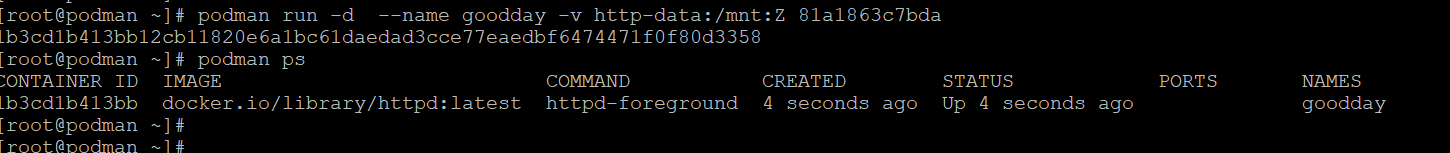
What ever we created in that directory that files should be reflected in the attached directory in container



**Storing Data with Volumes**



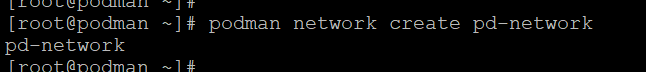
Now we are attaching the created volume



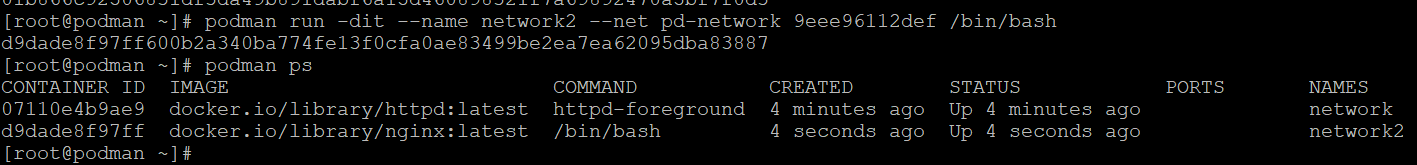
**Container Networking Basics**

Podman comes with a network called podman. By default, containers are attached to this network

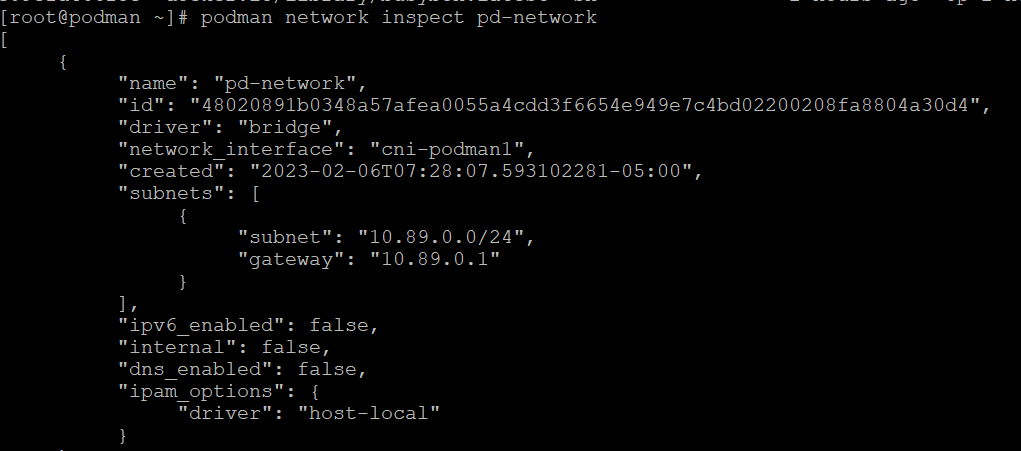
and can use it to communicate with one another.

****

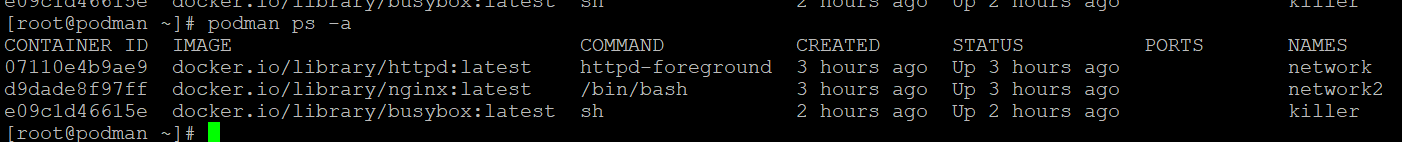
Attaching the network to the container

****

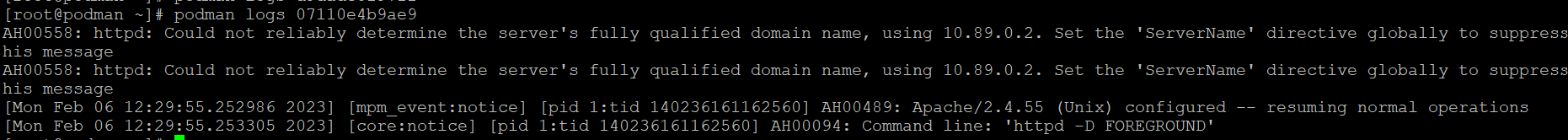
We can also inspect the created network



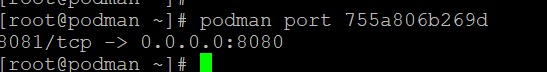
Troubleshoot Container Startup: A container might not be able to start successfully for different reasons, for example, due to missing configuration or a file access issue. Depending on the containerized application, the process executed by the container can either exit with an error status or keep running in an inconsistent state. You can list running and stopped containers by using the podman ps -a command.



If the container is in the Exited status then the problem might be in the start-up process.



Port Mapping Issues



To verify the application ports in use, list the open network ports in the running container. Use

Linux commands such as the socket statistics (ss) command to list open ports. A socket is the

combination of a port and an IP address. The ss command lists the open sockets in a system. You

can provide the ss command with options to filter and produce the desired output:

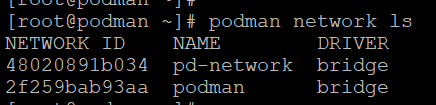
-p: display the process using the socket

• -a: display listening and established connections

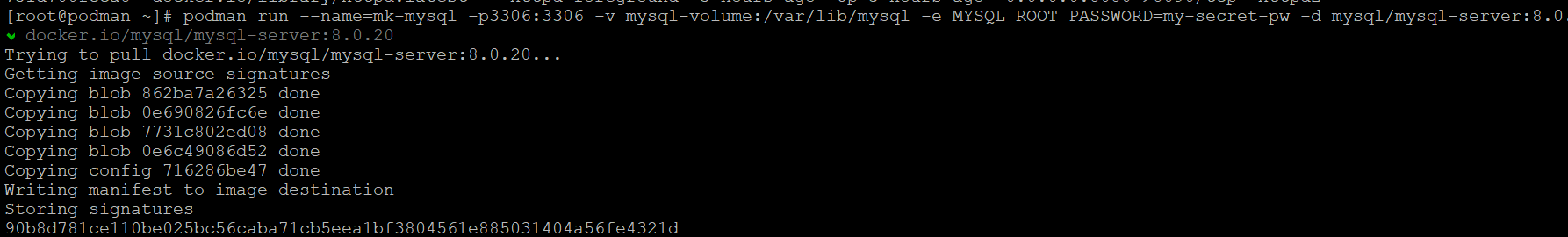
• -n: display IP addresses

• -t: display TCP sockets

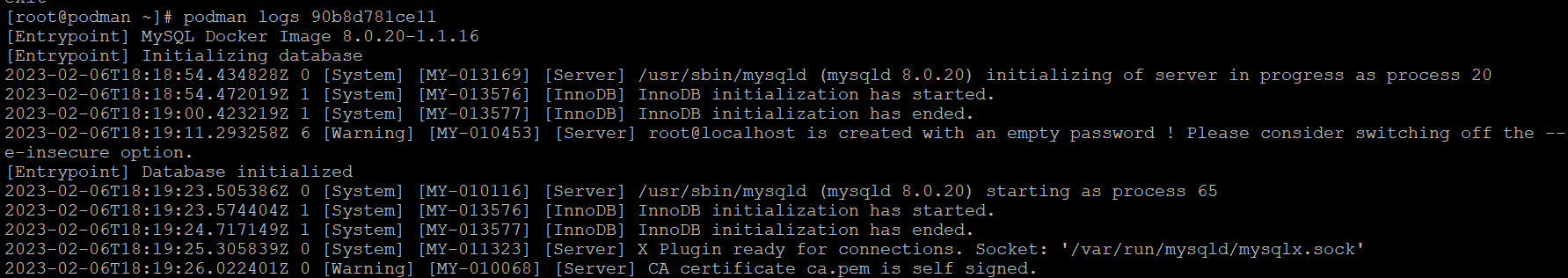
To list the networks



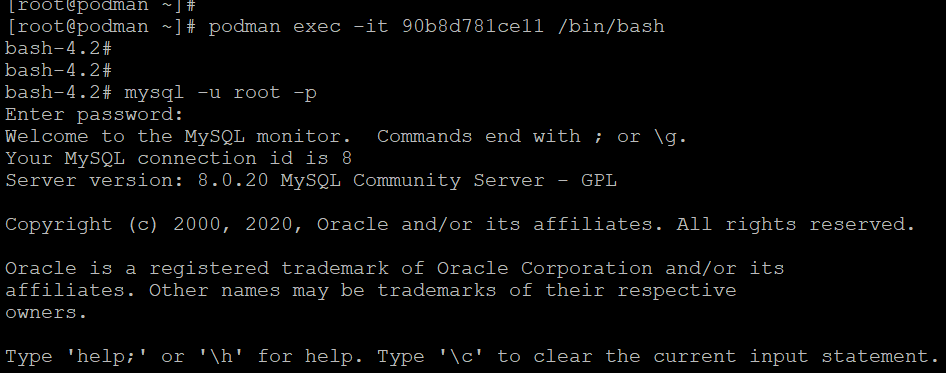
Creating the basic database



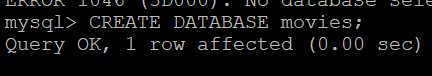
Logs of the container



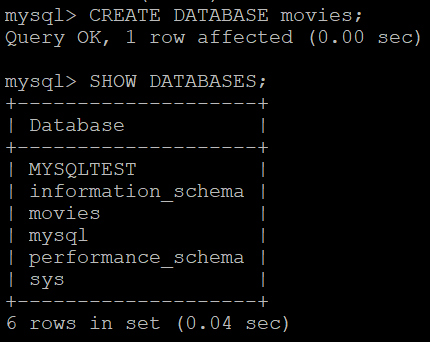
Switching to the mysql database



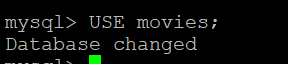
Creating the table



Listing the created tables

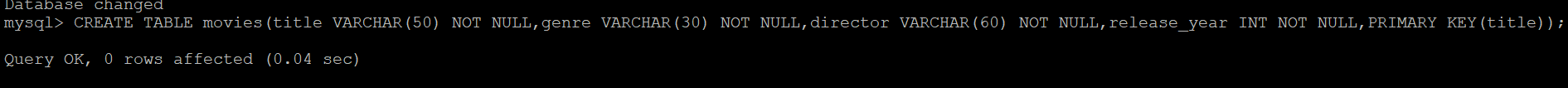


Changing the Database

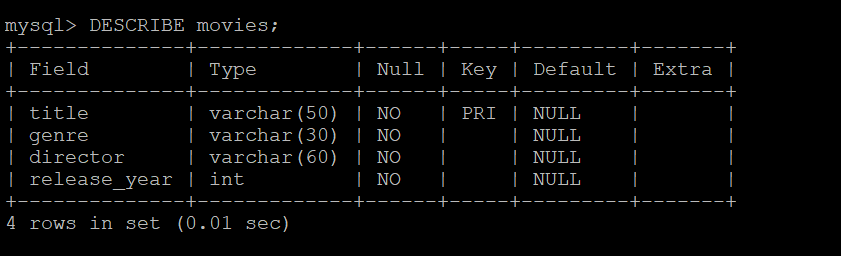


Creating the table:

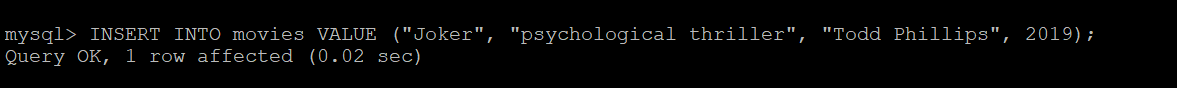
CREATE TABLE movies(title VARCHAR(50) NOT NULL,genre VARCHAR(30) NOT NULL,director VARCHAR(60) NOT NULL,release\_year INT NOT NULL,PRIMARY KEY(title));



Verify that the table is created using the DESCRIBE command:



Insert movie information in column order



Select a movie from the table

