**DOCKER Automation**

**What is Docker?**

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

**6 Docker Basics You Should know before getting started.**

* **Containers**
* **Images**
* **Docker files**
* **Volumes**
* **Port Forwarding**
* **Docker Compose**

**Container :**

Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers. Let’s discuss how to work with containers.

**Images:**

Docker, everything is based on Images. An image is a combination of a file system and parameters.

**Docker Files:**

Docker also gives you the capability to create your own Docker images, and it can be done with the help of **Docker Files**. A Docker File is a simple text file with instructions on how to build your images.

**Volumes:**

Docker volumes are file systems mounted on Docker containers to preserve data generated by the running container.

The volumes are stored on the host, independent of the container life cycle. This allows users to back up data and share file systems between containers easily.

**Port Forwarding:**

In Docker, the containers themselves can have applications running on ports. When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host.

**Docker Compose:**

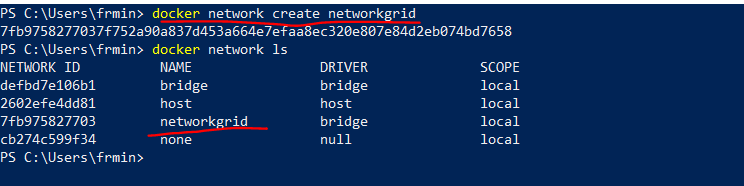
In Docker, the containers themselves can have applications running on ports. When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host.

**Docker Network**

**Create a Docker Network**

Use **docker network create <NETWORK\_NAME>** command to create a network so that the containers can communicate with each other.

List networks with **docker network ls**



**Create a Docker Hub**

Use docker run to create a hub.

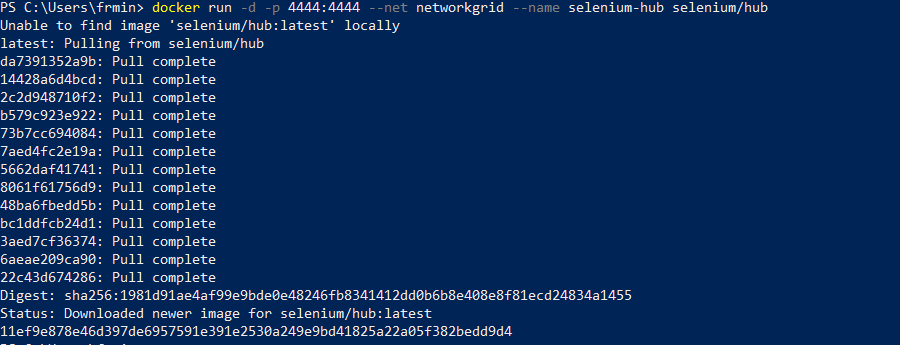
|  |  |
| --- | --- |
|  | $ docker run -d -p 4444:4444 --net networkgrid --name selenium-hub selenium/hub |

d: detached mode. Container starts in the background with this command. You don’t see any output from container console.

p: publish port (we bind the port 4444 of the container to 4444 of the docker host)

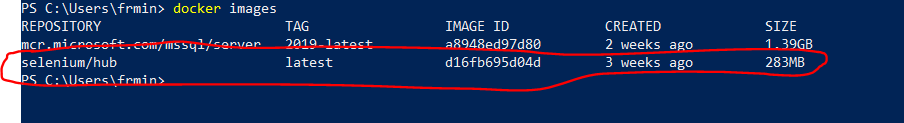
net: specify which network we add the container

name: specify a name of the container and the last parameter is the image name used when creating the container.



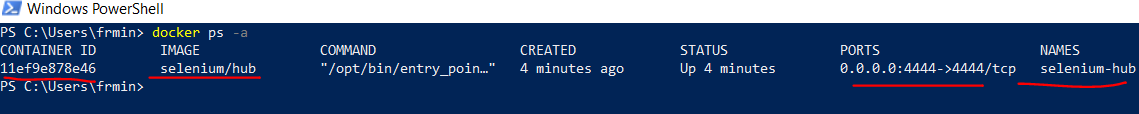
**To view the Downloaded images:**

**Command :** docker images

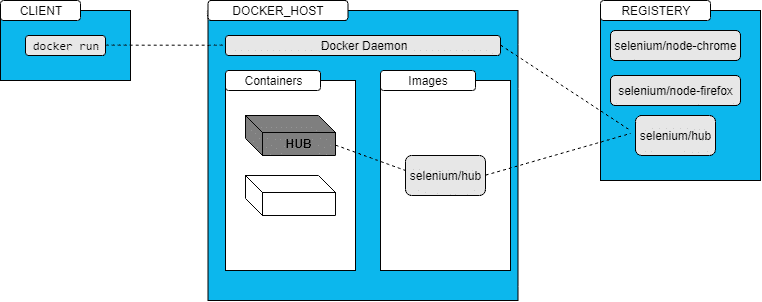


**To view the created container:**

**Command:** docker ps -a



Flow of the operation is as below:



**Step-1:** The Docker client contacted the Docker daemon.

**Step-2:** The Docker daemon pulled the “hub” image from the Docker Hub.

**Step-3:** The Docker daemon created a new container from that image.

If you want to see the logs of the container, you can use below command.

**docker logs <CONTAINER\_ID>**



**Images build for docker-selenium listed below and available in**[**Docker  Hub**](https://hub.docker.com/u/selenium/)**:**

base

hub

node-base

node-chrome

node-firefox

node-chrome-debug

node-firefox-debug

standalone-chrome

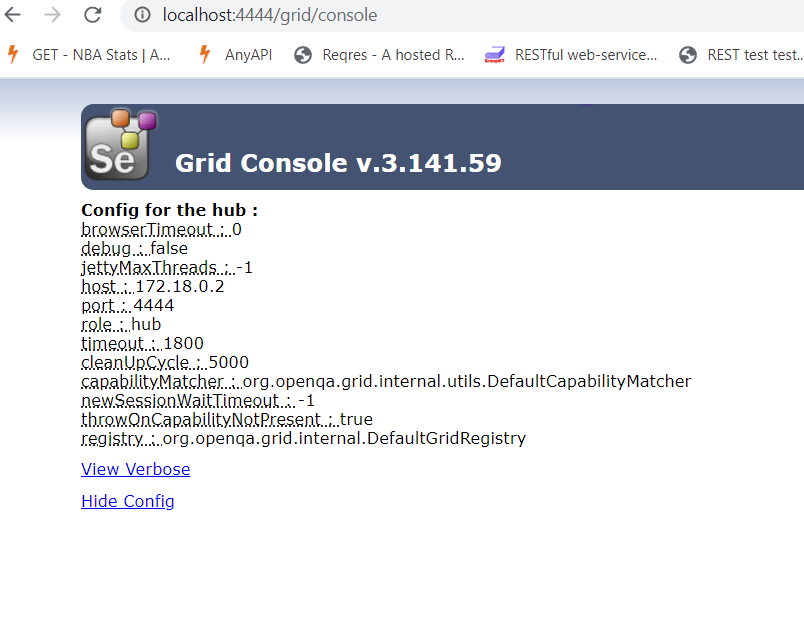
standalone-firefox

standalone-chrome-debug

standalone-firefox-debug

node-{Browser Name} and node-{Browser Name}-debug cannot be used alone; they must be connected to a hub. Debug versions include a VNC server to visually debug the browser during the test.

Check <http://localhost:4444/grid/console>



**Create a Docker Node:**

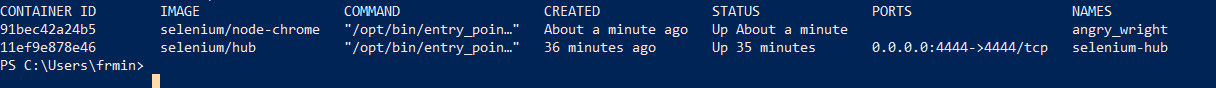
**Create a chrome node:**

|  |  |
| --- | --- |
|  | **$ docker run -d --net networkgrid -e HUB\_HOST=selenium-hub -v /dev/shm:/dev/shm selenium/standalone-chrome-debug** |

**e**: stands for environment variables.

**NODE\_MAX\_SESSION**, **NODE\_MAX\_INSTANCE** etc. are defined in this part. You can see other environment variables by using **docker inspect <IMAGE\_ID>** command.

**v**: to use host’s shared memory.



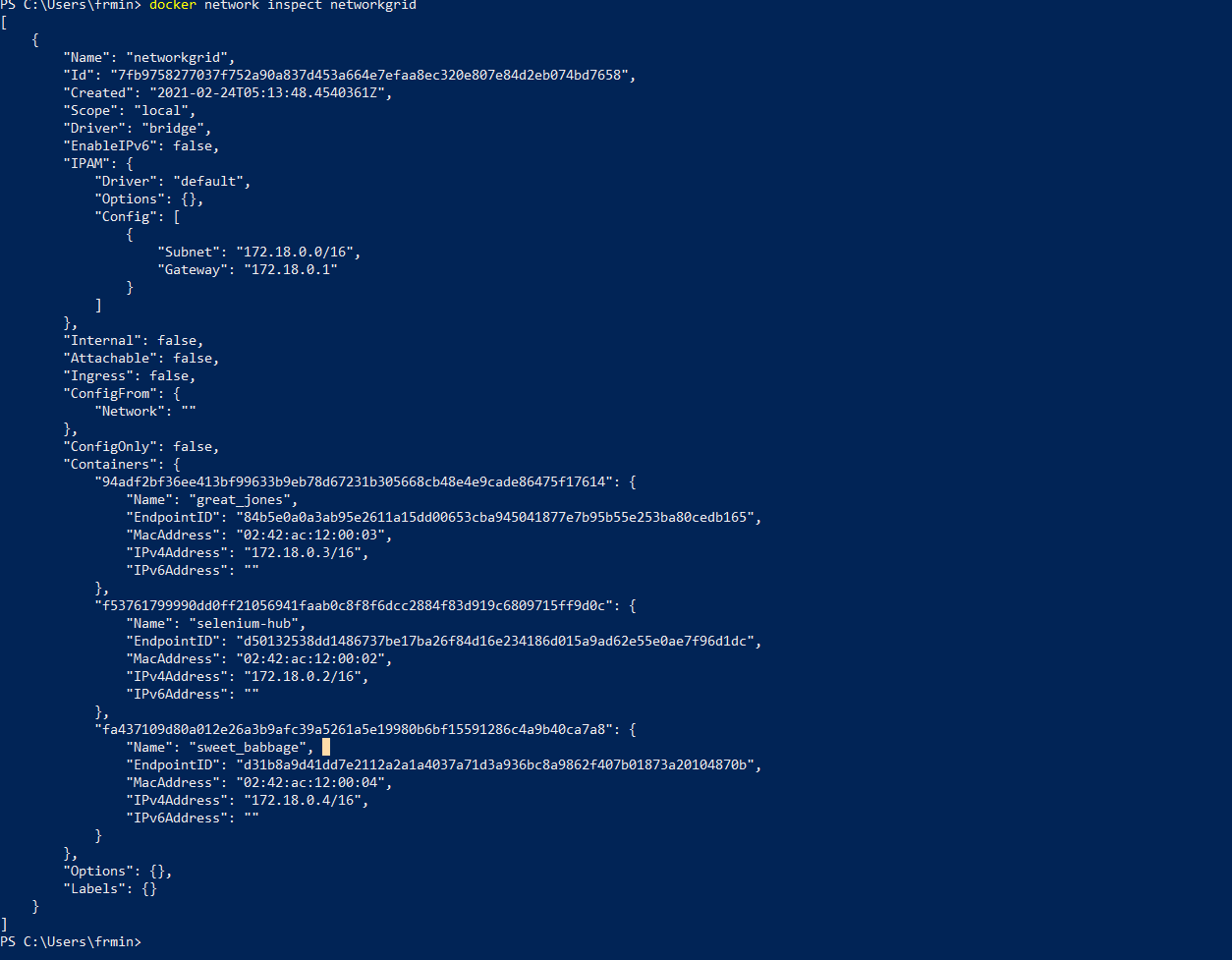
Similarly for Firefox standalone can be done:

**To see visually the running execution we need to port to vnc Player:**

**$ docker run -d -P -p 5903:5900 --net networkgrid -e HUB\_HOST=selenium-hub -v /dev/shm:/dev/shm selenium/standalone-chrome-debug**

**To view the attached nodes to selenium hub :**

**$ docker network inspect networkgrid**



**Docker Compose:**



Docker-compose up -d

docker-compose scale nodechrome=3 nodefirefox=3

**DockerFile:**

FROM maven:3-6-0-jdk-8-alphine  
#coping src of your framework  
COPY src /home/ParallelExecution/src  
#coping pom file of your framework  
COPY pom.xml /home/ParallelExecution  
#copy chromedriver  
COPY chromedriver.exe /home/ParallelExecution  
  
#building the package  
RUN mvn -f /home/ParallelExecution/pom.xml clean test -DskipTests=true

**To build the image:**

**Docker build -t parallelexecution .**

**Run the chrome node image:**

**$ docker run -d -p 4444:4444 --net networkgrid --name selenium-hub selenium/hub**

|  |  |
| --- | --- |
|  | **$ docker run -d --net networkgrid -e HUB\_HOST=selenium-hub -v /dev/shm:/dev/shm selenium/standalone-chrome-debug**  **$ docker run -d -P -p 5901:5900 --net networkgrid -e HUB\_HOST=selenium-hub -v /dev/shm:/dev/shm selenium/standalone-chrome-debug**  **To create a Docker Image:**  **$ docker run -d --net networkgrid parallelexecution mvn -f /home/ParallelExecution/pom.xml clean test -Dbrowser=”chrome”** |

**Stop and remove All running containers:**

docker stop $(docker ps -a -q)

docker rm $(docker ps -a -q)