Automation of building selenium script docker image through maven in docker

Table of Contents

1 Introduction to Docker4

1.1 What is Docker?4

1.2 Why Docker?4

1.3 Things to know before starting with Docker5

2 Docker configuration in Infosys Network9

2.1 How to get docker work in Infosys network and to pull images from Infosys network9

3 Some commonly used Docker commands13

4 Introduction to Dockerfile15

4.1 What is Dockerfile?15

4.2 Some useful dockerfile commands15

4.3 Steps to create and build a dockerfile18

5 Introduction to Docker-compose4

5.1 What is docker-compose?5

5.2 Brief about docker-compose.yml file with example5

5.3 Some basic commands used in docker-compose file5

5.4 Steps to run and stop docker-compose5

6 Selenium Grid with Docker4

6.1 Selenium Grid and Selenium Hub 5

6.2 Selenium Grid setup through Selenium Hub5

6.2.1 Selenium Hub setup through Docker commands5

6.2.2 Selenium Hub setup through Docker-compose5

6.3 Selenium Grid setup through Zalenium5

6.3.1 Why Zalenium?5

6.3.2 To start with Zalenium5

7 Selenium scripts execution 4

8 Creation of selenium docker image4

8.1 Process of building selenium docker image5

8.1.2 Generate Runnable jar file using eclipse5

8.1.3 Create Dockerfile to build docker image5

9 Automation of above process through Apache Maven4

9.1 Generating runnable jar file through Maven plugins 5

9.2 What is docker-maven plugin?5

9.3 Use docker-maven plugin to build the docker image automatically5

10 Parallel Execution using Junit framework4

10.1 Using Maven surefire plugin5

10.2 Using ParallelComputer class of Junit5

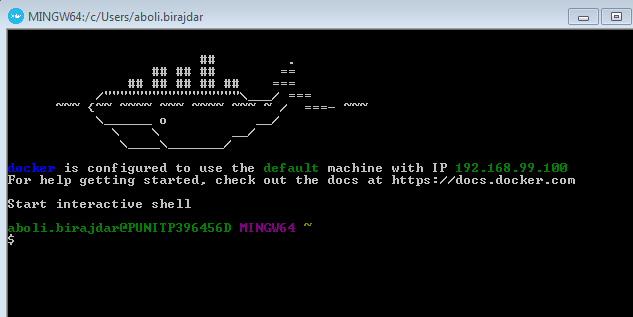
10.2.1 To make jar of the parallel execution script5

11 Running multiple selenium docker images 4

1. Introduction to Docker

1.1 What is Docker?

Docker is a container management service. The keywords of Docker are develop, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.



1.2 Why Docker?

* Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
* With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
* You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
* Since Docker containers are pretty lightweight, they are very easily scalable.

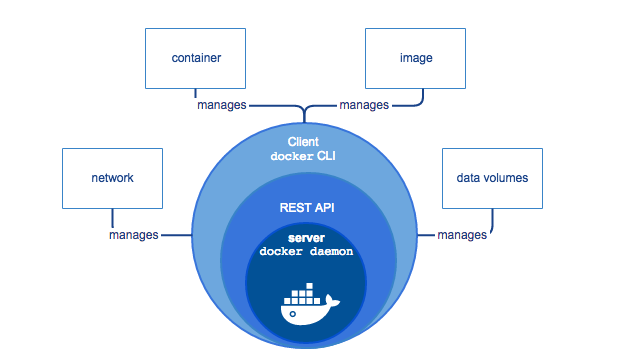
1.3 Things to know before starting with Docker

Before we start in detail, knowledge in basics of docker, maven, selenium, java is required. Additionally, docker must be installed on your system.

**Docker Engine:**

Docker Engine is a client-server application with these major components:

* A server which is a type of long-running program called a daemon process
* A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface (CLI) client.



**Docker Daemon:**

The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

**Docker Registries:**

A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker store allows you to buy and sell Docker images or distribute them for free. For instance, you can buy a Docker image containing an application or service from a software vendor and use the image to deploy the application into your testing, staging, and production environments. You can upgrade the application by pulling the new version of the image and redeploying the containers.

### Docker objects:

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects

**Docker Hub:**

Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub.

**Docker Compose:**

Compose is a tool for defining and running multi-container. Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration.

**Docker Files:**

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

Docker can build images automatically by reading the instructions from a Dockerfile.

**Docker Image:**

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

**Container:**

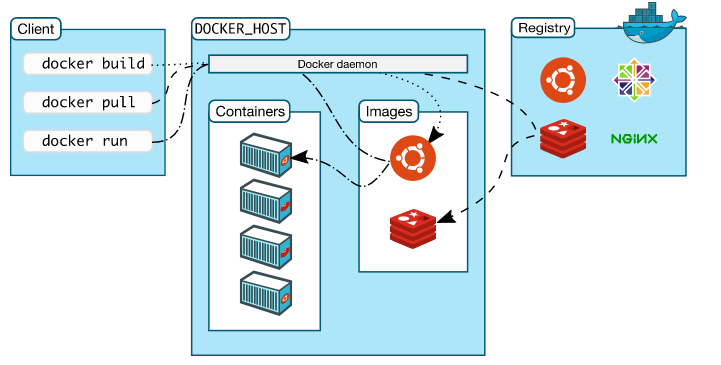
A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container’s network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

**Docker architecture:**

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

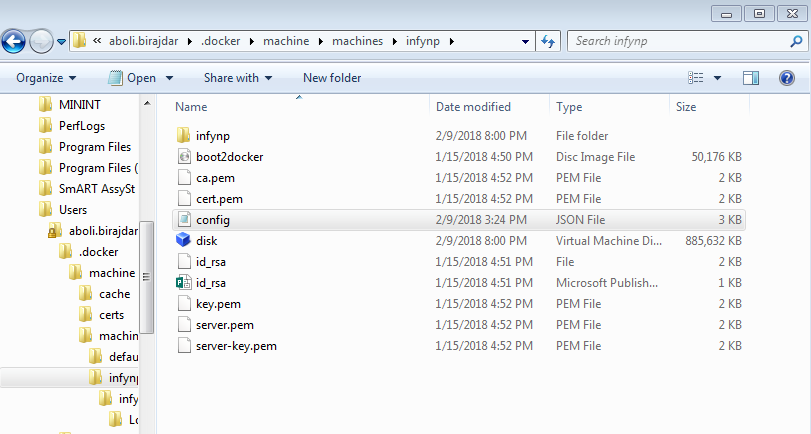


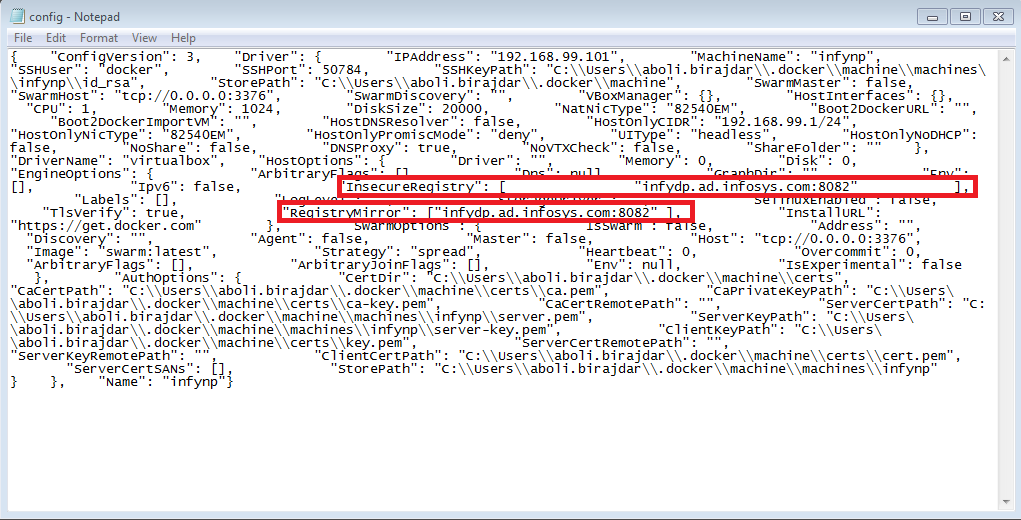
1. Docker configuration in Infosys Network

2.1 How to get docker work in Infosys network and to pull images from Infosys network?

Usually Docker creates a default VM inside which docker operations are performed but downloading an image from <https://hub.docker.com> is blocked by Infosys Firewall. To overcome this issue, we can use Nexus based Docker proxy and is accessible at *infydp.ad.infosys.com:8082.* Follow the below steps to configure Infosys Docker Proxy in case if you are using Docker toolbox:

* Navigate to C:\Users\<<User>>\.docker\machine\machines\default and edit the *config.json* file to add the *Insecure Registry* and *Registry Mirror* as shown below.



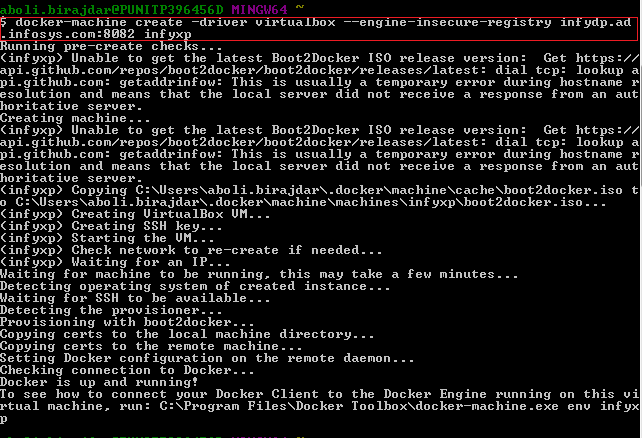


* Create a new docker-machine with insecure-registry option through the docker command:

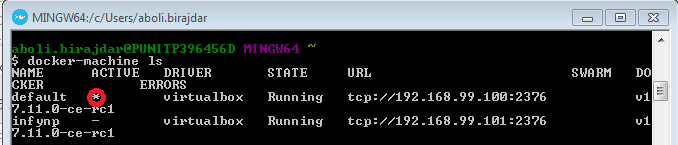
*docker-machine create --driver virtualbox --engine-insecure-registry infydp.ad.infosys.com:8082 infynp*

Where, Infynp is the VM’s name to be created.

Below is the screenshot of its usage with infyxp as its name.



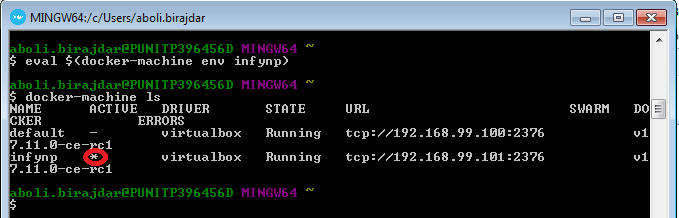
* After creating our own VM, one should check currently which VM is running because Docker already has it’s default container.



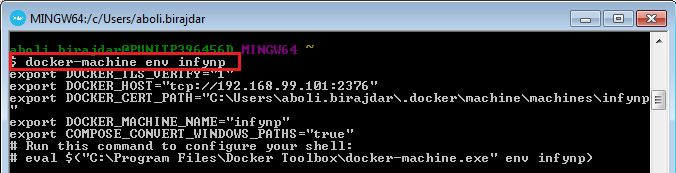
In the above screenshot the \* is indicating the VM in which container is running.

* It was running in Default VM. So one should have to change that into Infynp VM.

To switch the default VM to infynp VM, use the following command:

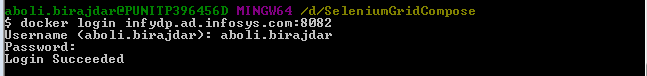


* To know more information about running VM, use the following docker command.

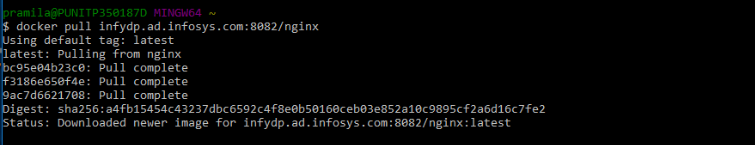


In this, 192.168.99.101 is Remote IP and 2376 is port. We can switch to any necessary port like to access selenium grid we can 192.168.99.101:4444 to access it as selenium grid usually hosts in port 4444 by default.

* After that Login with your Infosys ID and password to connect with Network and pull images as follows



* Then test the Pull & Run command for the required docker Image and make sure that the download is done.



For more information about Nexus based Docker proxy, you can refer the website <http://wiki/Docker>

1. Some commonly used Docker Commands

* To list all running containers -

$ docker ps

* To list all running and stopped containers -

$ docker ps –a

* To get help

$ docker help

* To list all Docker images -

$ docker images

* To pull a Docker image -

$ docker pull [image name]

* To push an image or a repository to a registry -

Syntax - $ docker push[options] name [: tag]

Example- $ docker tag rhel-httpd registry-host:5000/myadmin/rhel-httpd

//tag with hostname: port

docker push registry-host:5000/myadmin/rhel-httpd //push it to registry

* Run a Docker container-

Syntax - $ docker run –d –p [container port]:[host port][image or repository: tag][command][args]

Example- $ docker run-i-t-alpine/bin/sh

* To stop a docker containers –

Syntax - $ docker stop<container id>

* To remove one or more docker containers -

Syntax - $ docker rm[options] container [container…]

* To stop all containers -

$ docker rm $(docker ps –a -q

* To remove one or more Docker images -

Syntax - $ docker rmi[options]image [image...]

Example- $ docker rmi test

$ docker rmi –f fd48f19954f // deletes all the image's matching id

* To get Information of Docker –

$ docker info

* To search for images created by other people –

$ docker search learn

* To save the container with new image-

Syntax - $ docker commit [id] [new image]

* To create a tag Target image that refers to Source image-

Syntax - $ docker tag source image[:tag] target image [:tag]

Example- $ docker image tag 03e398c809c5 ourfiglet //tags this image id as ourfiglet

1. Introduction to Docker file

4.1 What is Docker file?

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

A Docker file is a text document that contains all the commands a user could call on the command line to assemble an image.

The Docker file is essentially the build instructions to build the image. The advantage of a Docker file over just storing the binary image (or a snapshot / template in other virtualization systems) is that the automatic builds will ensure you have the latest version available. This is a good thing from a security perspective, as you want to ensure you’re not installing any vulnerable software.

4.2 Some Useful Docker File commands

1. **ADD**

The ADD command gets two arguments: a source and a destination. It basically copies the files from the source on the host into the container's own file system at the set destination. If, however, the source is a URL (e.g. http://github.com/user/file/), then the contents of the URL are downloaded and placed at the destination.

Example**:**

# Usage: ADD [source directory or URL] [destination directory]

ADD /my\_app\_folder /my\_app\_folder

**2. CMD**

The command CMD, similarly to RUN, can be used for executing a specific command. However, unlike RUN it is not executed during build, but when a container is instantiated using the image being built. Therefore, it should be considered as an initial, default command that gets executed (i.e. run) with the creation of containers based on the image.

To clarify: an example for CMD would be running an application upon creation of a container which is already installed using RUN (e.g. RUN apt-get install …) inside the image. This default application execution command that is set with CMD becomes the default and replaces any command which is passed during the creation.

**Example:**

# Usage 1: CMD application "argument", "argument”, .

CMD "echo" "Hello Docker!"

**3. ENTRYPOINT**

ENTRYPOINT argument sets the concrete default application that is used every time a container is created using the image. For example, if you have installed a specific application inside an image and you will use this image to only run that application, you can state it with ENTRYPOINT and whenever a container is created from that image, your application will be the target.

If you couple ENTRYPOINT with CMD, you can remove "application" from CMD and just leave "arguments" which will be passed to the ENTRYPOINT.

**Example:**

# Usage: ENTRYPOINT application "argument", "argument", ..

# Remember: arguments are optional. They can be provided by CMD

# or during the creation of a container.

ENTRYPOINT echo

# Usage example with CMD:

# Arguments set with CMD can be overridden during \*run\*

CMD "Hello docker!"

ENTRYPOINT echo

**4. ENV**

The ENV command is used to set the environment variables (one or more). These variables consist of “key value” pairs which can be accessed within the container by scripts and applications alike. This functionality of Docker offers an enormous amount of flexibility for running programs.

**Example:**

# Usage: ENV key value

ENV SERVER\_WORKS 4

**5.EXPOSE**

The EXPOSE command is used to associate a specified port to enable networking between the running process inside the container and the outside world (i.e. the host).

**Example:**

# Usage: EXPOSE [port]

EXPOSE 8080

**6. FROM**

FROM directive is probably the most crucial amongst all others for Docker files. It defines the base image to use to start the build process. It can be any image, including the ones you have created previously. If a FROM image is not found on the host, Docker will try to find it (and download) from the Docker Hub or other container repository. It needs to be the first command declared inside a Docker file.

**Example:**

# Usage: FROM [image name]

FROM Ubuntu

**7. MAINTAINER**

One of the commands that can be set anywhere in the file - although it would be better if it was declared on top - is MAINTAINER. This non-executing command declares the author, hence setting the author field of the images. It should come nonetheless after FROM.

**Example:**

# Usage: MAINTAINER [name]

MAINTAINER authors\_name

**8. RUN**

The RUN command is the central executing directive for Docker files. It takes a command as its argument and runs it to form the image. Unlike CMD, it actually is used to build the image (forming another layer on top of the previous one which is committed).

**Example:**

# Usage: RUN [command]

RUN aptitude install -y riak

**9. USER**

The USER directive is used to set the UID (or username) which is to run the container based on the image being built.

**Example:**

# Usage: USER [UID]

USER 751

**10. VOLUME**

The VOLUME command is used to enable access from your container to a directory on the host machine (i.e. mounting it).

**Example:**

# Usage: VOLUME ["/dir\_1", "/dir\_2" ..]

VOLUME ["/my\_files"]

**11. WORKDIR**

The WORKDIR directive is used to set where the command defined with CMD is to be executed.

**Example:**

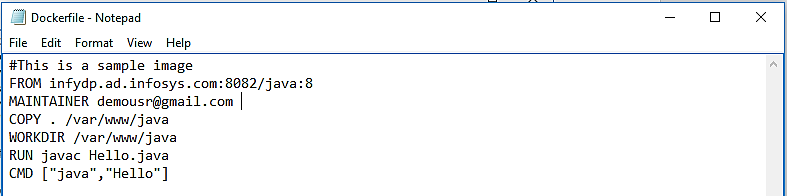
# Usage: WORKDIR /path

WORKDIR ~/

4.3 Steps to create and build a Dockerfile

**Step 1** - Create a file called Dockerfile inside a folder. Please note that the name of the file has to be "Dockerfile" with "D" as capital and double quotes.

**Step 2** – Open the dockerfile and add the below content.



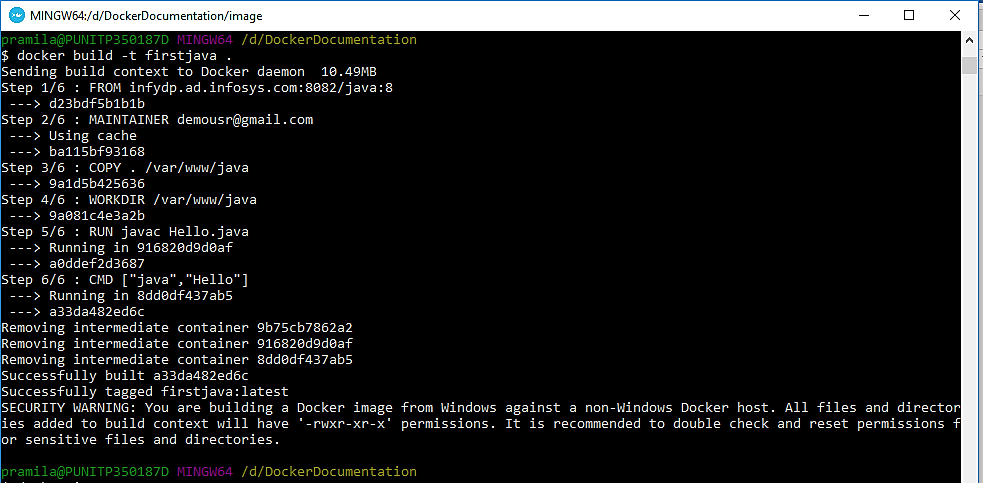
The following points need to be noted about the above file -

* The first line "#This is a sample Image" is a comment. You can add comments to the Docker File with the help of the #command
* The next line has to start with the FROM keyword. It tells Docker, from which base image you want to base your image from. In our example, we are creating our image from the Java docker image.
* The next command is the person who is going to maintain this image. Here you specify the MAINTAINER keyword and just mention the email ID.
* The RUN command is used to run instructions against the image. In our case, we are compiling the Hello.java file. *Note*: Hello.java is a simple Java class printing a message.
* Commands mentioned after CMD keyword will be executed when a container is launched from a Docker image. In our case, we are running the java class.

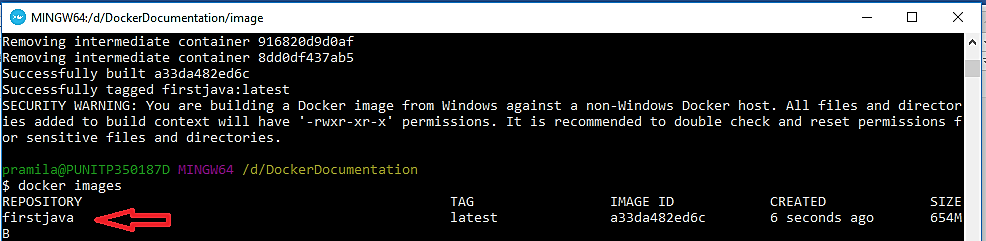
**Step 3** – Go to the docker file folder and run the beneath command, in the below command –t is used to set tag name of docker image, in my case I am setting Docker image name and tag as “firstjava”

Docker build –t firstjava .

Below is the screenshot of building the dockerfile.



After the image has built successfully, we can view the docker image by using docker command as follows



You can run the docker image as follows



1. Introduction to Docker-Compose

5.1 What is Docker Compose?

Docker-compose is a command line tool to define and configure ***multi-container Docker application***s. In other words, we can say Docker compose is used to link multiple containers and deploy application from a single command

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration.

Following are some features of Compose that make it effective:

* Multiple isolated environments on a single host
* Preserve volume data when containers are created
* Only recreate containers that have changed
* Variables and moving a composition between environments

5.2 Brief about docker-compose.yml file with example

A docker-compose.yml is a config file for Docker-compose. It allows to deploy, combine and configure multiple Docker-container at the same time.

Docker compose is tool which allows you to deploy multiple containers at the same time. File docke-compose.yml is file with instructions - how to do that. In this file you instruct Docker-compose where to take Docker file to build particular image, which ports you want to expose, how to link containers, which ports you want to bind to host machine etc. Docker-compose reads that file and executes commands. It is used instead all optional parameters when building and running single Docker container.

**Example**:

version: '2'

services:

nginx:

build: ./nginx

links:

- django:django

- angular:angular

ports:

- "80:80"

- "8000:8000"

- "443:443"

networks:

- my\_net

django:

build: ./django

expose:

- "8000"

networks:

- my\_net

angular:

build: ./angular2

links: - django:django

expose:

- "80"

networks:

- my\_net

networks:

my\_net:

external:

name: my\_net

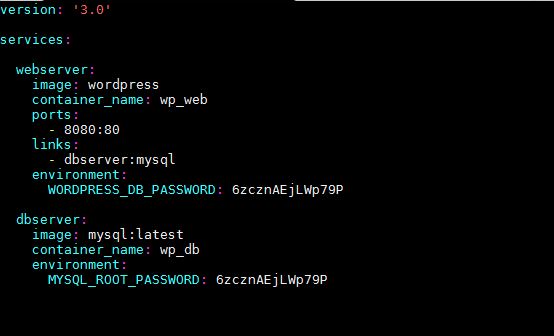
This example instructs Docker-compose to:

* build nginx from path./nginx
* links angular and django containers (so their IP in Docker network is resolved by name)
* binds ports 80,443,8000 to host machine
* add it to network my\_net(so all 3 containers are in the same network and therefore accessible from each other)
  1. Some Basic Commands Used in Docker-Compose

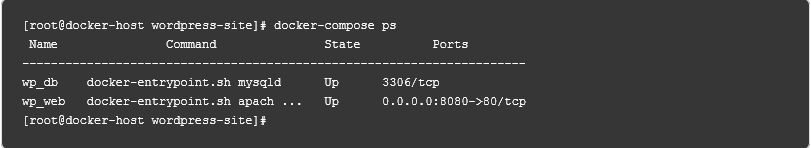
|  |  |
| --- | --- |
| * **Build** | Build or rebuild services |
| * **bundle** | Generate a Docker bundle from the Compose file |
| * **config** | Validate and view the compose file |
| * **create** | Create services |
| * **down** | Stop and remove containers, networks, images, and volumes |
| * **events** | Receive real time events from containers |
| * **exec** | Execute a command in a running container |
| * **help** | Get help on a command |
| * **images** | List images |
| * **kill** | Kill containers |
| * **logs** | View output from containers |
| * **pause** | Pause services |
| * **port** | Print the public port for a port binding |
| * **ps** | List containers |
| * **pull** | Pull service images |
| * **push** | Push service images |
| * **restart** | Restart services |
| * **rm** | Remove stopped containers |
| * **run** | Run a one-off command |
| * **scale** | Set number of containers for a service |
| * **start** | Start services |
| * **stop** | Stop services |
| * **top** | Display the running processes |
| * **unpause** | Unpause services |
| * **up** | Create and start containers |
| * **version** | Show the Docker-Compose version information |

5.4 Steps to Run Docker Compose

* To deploy Docker containers with Docker-compose command, first create a directory under that directory create a compose file with name ***docker-compose.yml*.** In the compose file we will define the services for our applications and container images.
* Then we need to pull the images we have otherwise Docker-compose will create its own image. Command used *docker pull imagename.*
* To List the images that we are using use command *docker image ls.*
* Create docker-compose.yml file with following content.

****

* In the above compose file
* We have defined two services with the name “webserver” and “dbserver”, for these services we have also specified container image “wordpress” and “mysql: latest” respectively.
* Apart from this we have specify environments by mentioning the mysql root password and wordpress db password.
* Use command *docker-compose up,*to deploy our application. This Command will deploy two containers with name “wp\_web” and “wp\_db”.
* Command *docker compose ps* is used to list the containers which are deployed for our application. Example:



From the above image we get to know that, it gives us the idea about the State of the container and its corresponding port number.

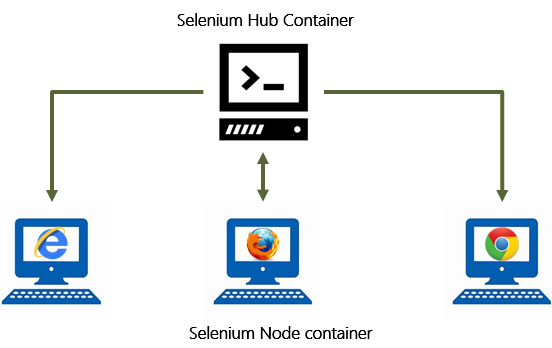
*docker compose down* command is used to stop and remove the containers.

1. Selenium grid with Docker

6.1 Selenium Grid and Selenium Hub

Selenium Grid is used to speed up the execution of a test pass by using multiple machines to run tests in parallel. It is also used to support running tests against multiple runtime environments, specifically, against different browsers at the same time. Selenium Hub works as a server, which routes the request given by clients to its Nodes. Nodes can be any physical, virtual or any remote machine.

Following is the Selenium Grid architecture:



Why should we use Selenium hub?

* no need to manage Selenium server jar – it’s done automatically inside the Selenium hub container;
* no need to download browsers drivers and configure classpath – for the same reason;
* registering nodes on the hub is done much more easily, Grid is easily extendable and manageable.
* The selenium Hub receives a test to be executed along with information on which browser and platform where the test should be run.
* The hub will use this information and delegate to a node that can service those needs.

6.2 Selenium grid setup through selenium hub

Following docker images should be pulled before starting with the steps:

1. selenium/hub
2. selenium/node-chrome
3. selenium/firefox

Selenium hub can either be configured through docker commands or docker-compose

* + 1. Selenium hub setup through docker commands
* Following Docker command is used to run selenium Hub with Docker

$ docker run -d –p 4444:4444 –name selenium-hub selenium-hub

Where-p is used for exposing a port (it will expose a port called 4444)

-d is used to run the container in background.

* We can optionally override default configuration settings using environment variables.

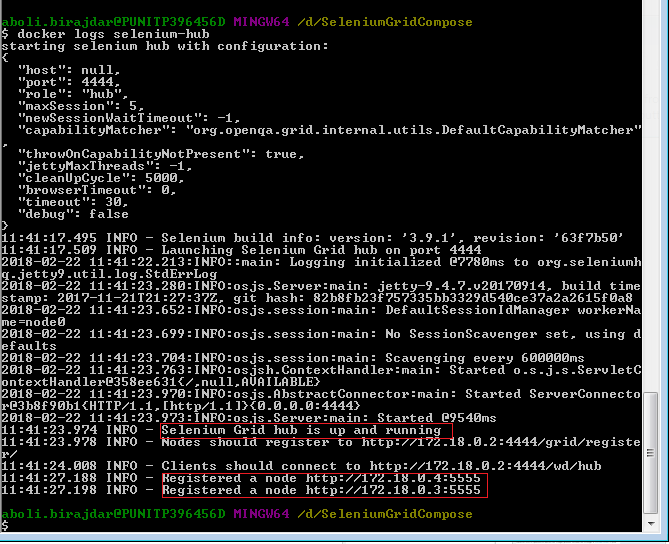
$ docker run -d -P --name selenium-hub -e GRID\_TIMEOUT=10 selenium/hub

* Once the hub is up and running will want to launch nodes that can run tests. You can run as many nodes as you wish.
* To link the nodes to the selenium hub following commands are used-

$ docker run -d --link selenium-hub: hub selenium/node-chrome

$ docker run -d –link selenium-hub: hub selenium/node-firebox

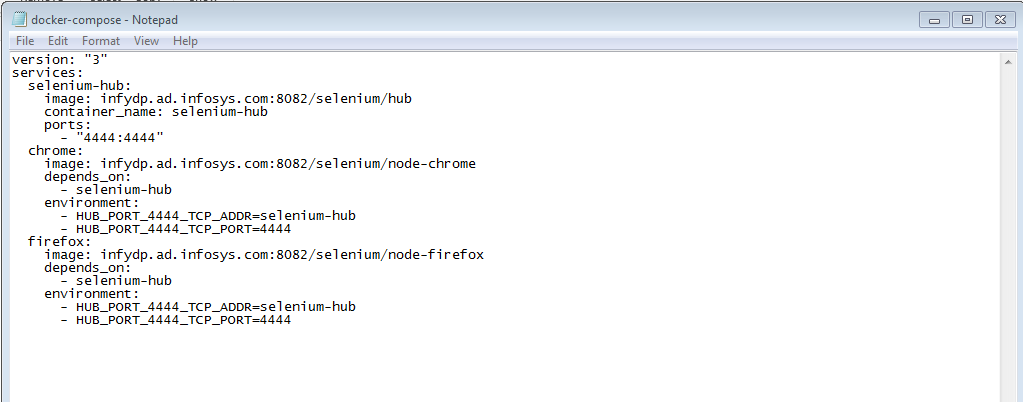
* To see the number of process running currently in docker $ docker ps –a
* To see what is happening with particular hub $ docker logs
* Now, Selenium hub is running succesfully and currently 2 nodes are registered to the selenium hub.



Your hub will be available through the <http://192.168.99.101:4444/grid/console> URL where http://192.168.99.101 docker VM’s remote IP.



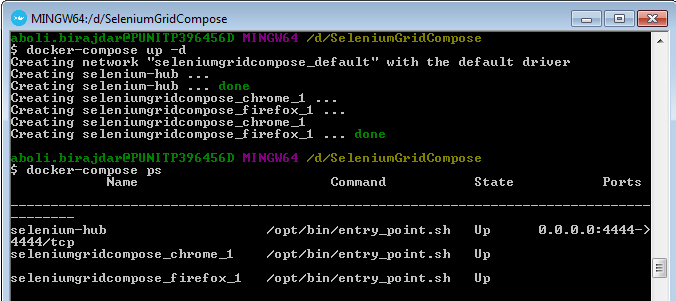
* + 1. Selenium hub setup through docker-compose
* Create a docker-compose.yml file. This file should be placed in a directory with the following contents



* Open a shell, navigate to the directory of the newly created YAML file, and use the following command:

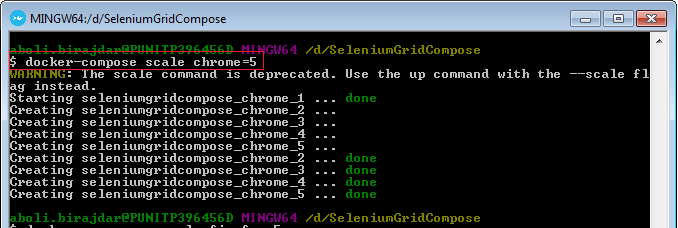
$ docker–compose up –d

* Open This will pull the official images for hub, node-chrome, and node-firefox from Docker if doesn’t exist in docker and start a grid with one Istance of each browser available.

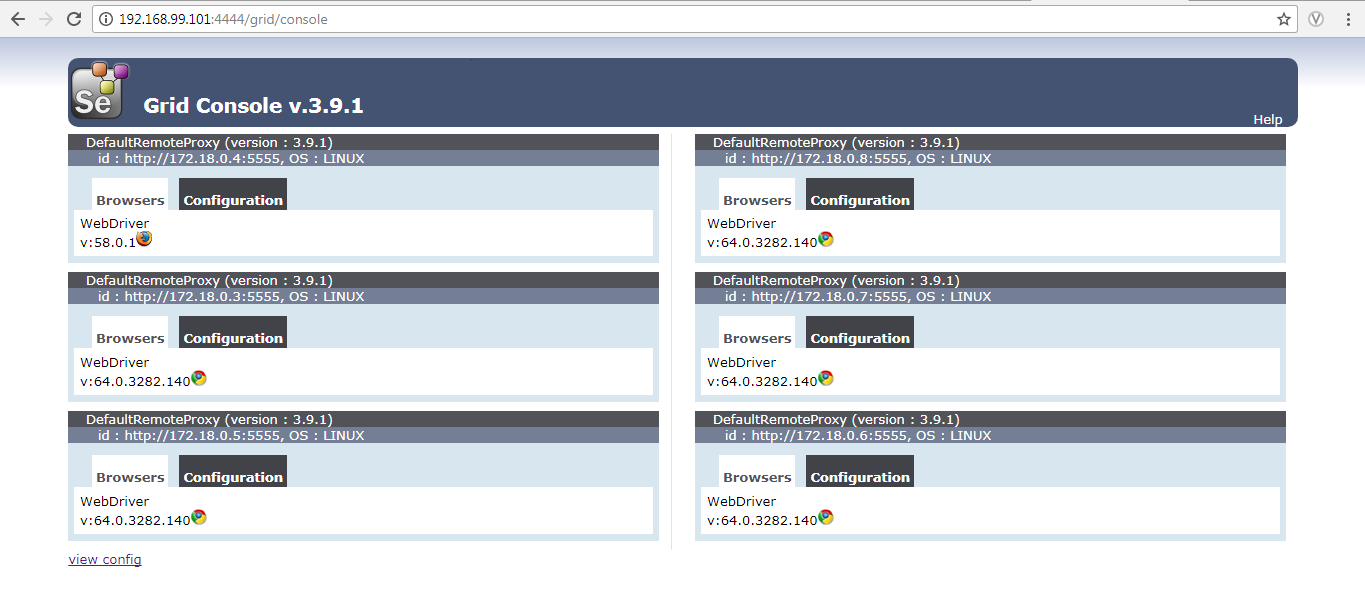


* We can increase the number of nodes by setting the number of containers to run for a service though the following command.

$ docker–compose scale chrome=5

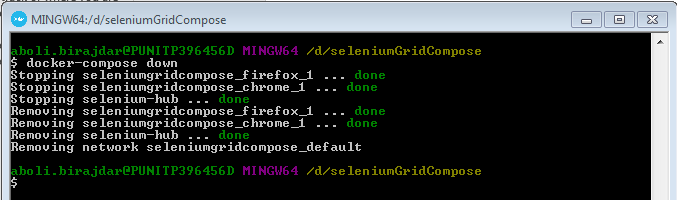


* Here 5 Nodes were created after executing the above command.



* The services and stops by using docker-compose down command
* It Stops containers and removes containers, networks, volumes, and images created by up. Use the following command

$ docker-compose down



* By default, the only things removed are:
  + Containers for services defined in the Compose file
  + Networks defined in the networks section of the Compose file
  + The default network, if one is used

Networks and volumes defined as external are never removed.

6.3 Selenium grid setup through Zalenium

Zalenium is result of mixing Zalando and Selenium. It A Selenium Grid extension to scale up and down your local grid dynamically with docker containers. It uses [docker-selenium](https://github.com/elgalu/docker-selenium) to run your tests in Firefox and Chrome locally.

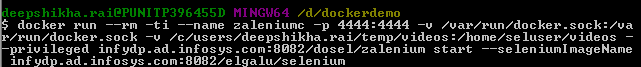
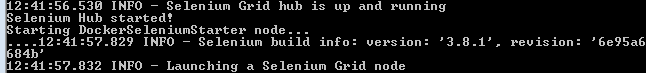
6.3.1 Why Zalenium?  
We know how complicated it is to: -

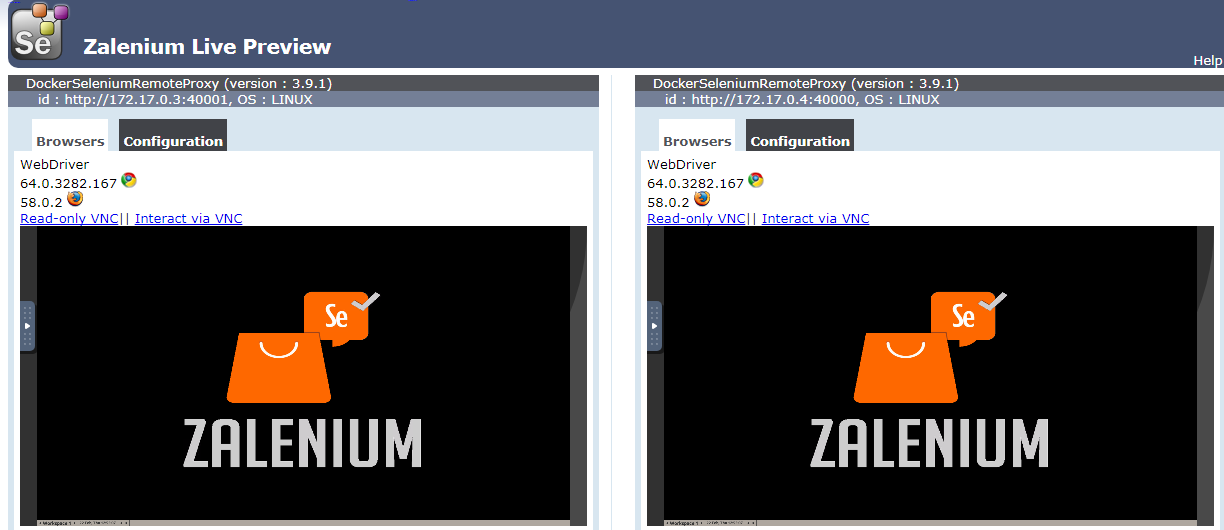
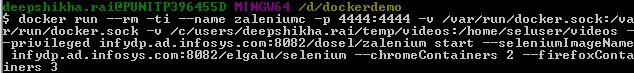
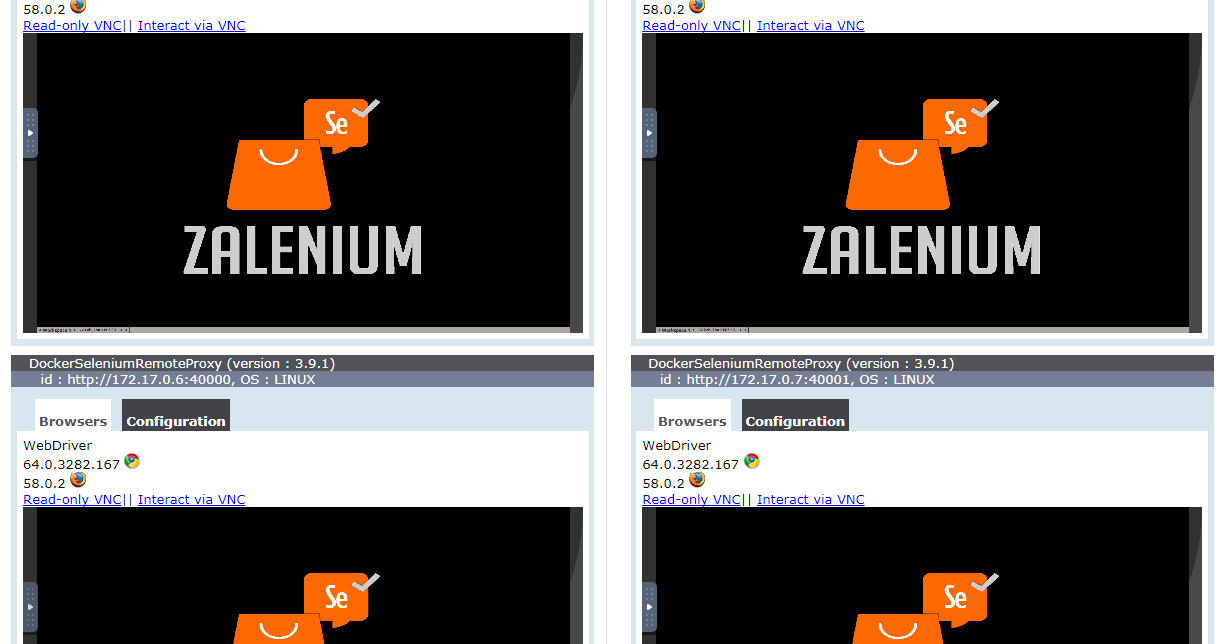
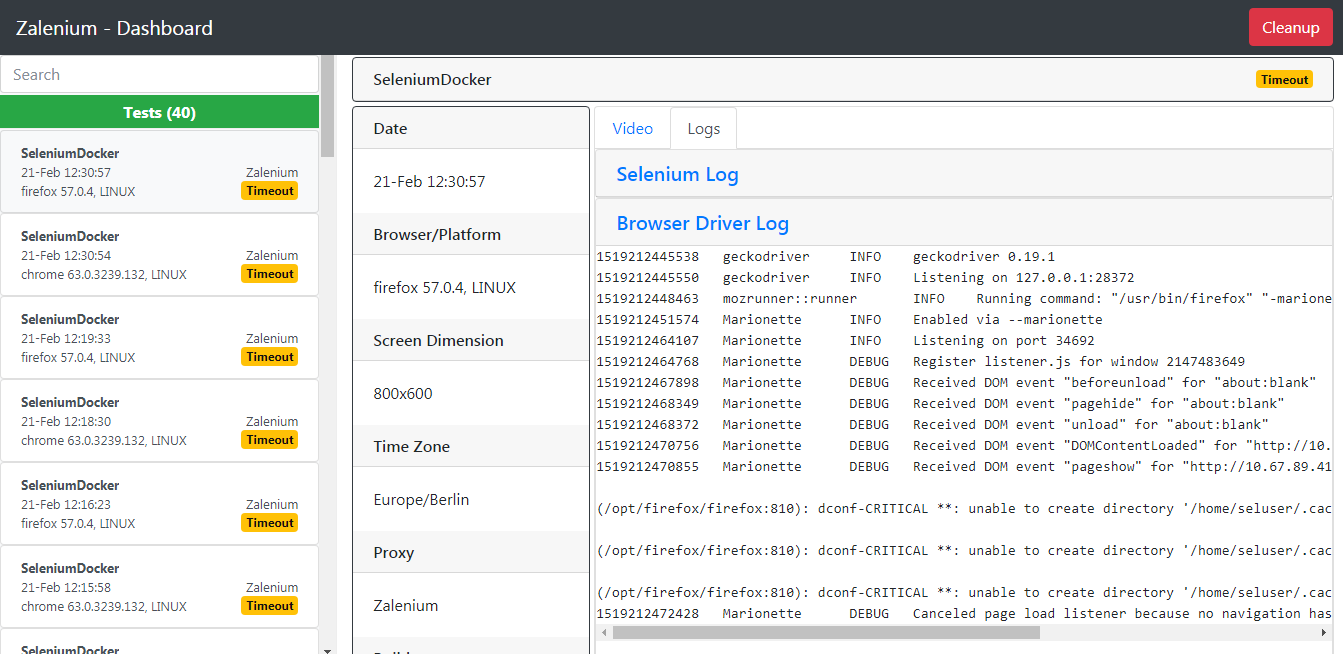
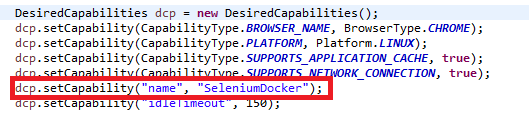
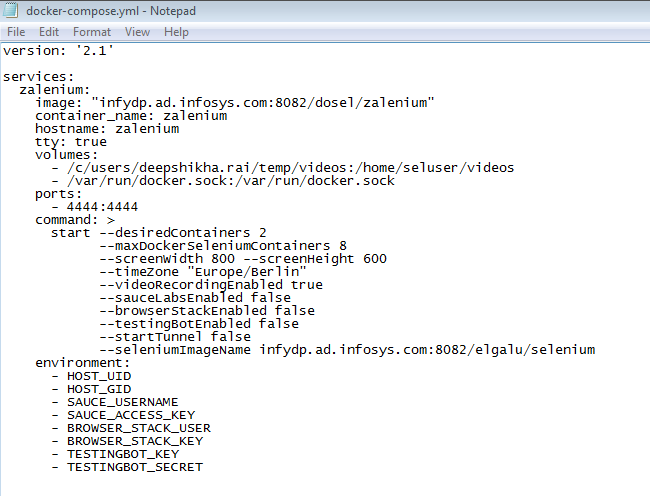
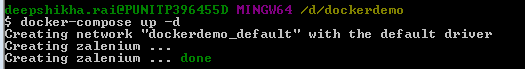
* Have a stable grid to run UI tests with Selenium
* Maintain it over time (keep up with new browser, Selenium and driver’s versions)
* Provide capabilities to cover all browsers and platforms

That is why we took this approach where docker-selenium nodes are created on demand. Our UI tests run faster in Firefox and Chrome because they are running in our own local network, on a node created from scratch and disposed after the test completes.  
*The main goal of zalenium is to allow anyone have a disposable and flexible Selenium Grid infrastructure.*

Pre-requisites:Docker should be installed and you must be familiar with basic docker commands.

**To start with Zalenium:**

C:\Users\deepshikha.rai\Desktop\commands_to_pull_zalenium.PNG   
  
First we need to pull Zalenium images to proceed. The above command is used to pull the image of Zalenium from Docker-Hub, but when we are working in Infosys network we need to add the Infosys registry as done below.  
  
C:\Users\deepshikha.rai\Desktop\zael_image_commamd.PNG  
  
Once we have pulled the images, now to start zalenium in windows we need to use the below command in our docker terminal.  
  
  
  
here –rm tag is used so that it automatically removes the container if it already existed. –name tag is used to provide name to the container, --privileged is used to provide extra privilege to this container. Now when we execute the command we will see the following results.  
  
  
Here we can see that selenium hub is up and running and nodes are to be registered in the hub.  
  
  
Now we can see that the nodes are also successfully registered.

**Zalenium live preview:**  
One of the cool features of zalenium is the live preview of the dockerized containers. We can access this using URL <http://192.168.99.101:4444/grid/admin/live> . Below is the screenshot for the same.  
  
  
By default, our Selenium grid will have 1 Chrome and 1 Firefox container. If you need more, say 2 chrome containers, 3 Firefox then use below arguments  
--chromeContainers 2, --firefoxContainers 3 as shown in screenshot below  
  
  
  
Now the nodes in our zalenium preview will be increased respectively. As seen below.  
  
  
**Dashboard:**   
  
Zalenium also has  a nice dashboard to show the list of recorded tests it has captured. The URL for the same is [http://192.168.99.101:4444/dashboard/#](http://192.168.99.101:4444/dashboard/). Below is the screenshot of the dashboard.  
  
  
As we can see it shows us the test that we have done in the past and also provide the related details. It also stores video. Which we help us to find the errors etc. so that it become easy to work with. Here my test name is SeleniumDokcer but if you want to change you can do that. See the below code for the same.  
  
  
Now if you want to stop recording videos, for your test then you just need to add the below code.  
  
C:\Users\deepshikha.rai\Desktop\stopvideorecording.PNG  
  
This was done by manually typing the command in docker terminal, but if we continuously want to work with zalenium. Then we have to always give the command in docker terminal which become tedious. But instead of this if we create a docker-compose yml file to start zalenium. To know how to create docker-compose file refer *docker-compose* section of the document. Below is the sample of docker-compose yml file to set the up the zalenium.  
  
after this when we run the *docker-compose up* command in docker-terminal our zalenium is started and selenium hub is up. As shown below.  
  
  
Here it is visible that zalenium is up, we have used the –d tag so that the process can be done in background. We can view selenium grid, *zalenium live and dashboard* on their respective *URL* as given previously.

In this way, we have to just save this docker-compose once. From next time whenever we want to use it we have to just give docker-compose up command.

NOTE: We cannot work with Internet explorer in docker, as docker have no image of Internet explorer.

1. Selenium scripts execution

Selenium scripts execution can be done either by using selenium hub or zelanium. We learnt how to setup selenium grid through docker but what modifications should we do to run our selenium scripts in selenium hub?

It can be done by using DesiredCapabilities class to tell the webdriver, which environment we are going to use in our test script.

Following are the functions we have used:

* The setCapability() method of the Desired Capabilities class can be used to set the device name, platform version, platform name, absolute path of the app under test.
* RemoteWebDriver function is an implementation class of the WebDriver interface that a test script developer can use to execute their test scripts
* The RemoteWebDriver server is a component that listens on a port for various requests from a RemoteWebDriver client. Once it receives the requests, it forwards them to any of the following: Firefox Driver, IE Driver, or Chrome Driver, whichever is asked.



Above is the code snippet we have included.

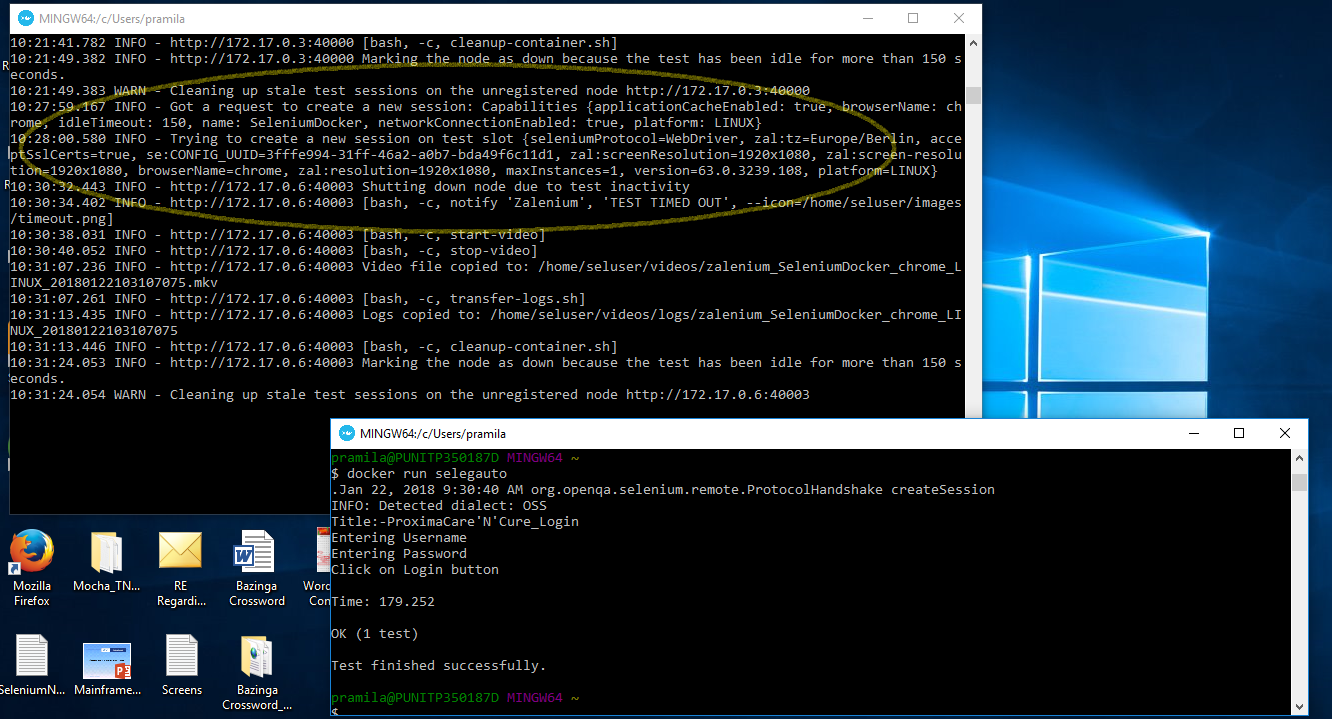
Code Explanation:

* In line 26, we have initialized the DesiredCapabilities class.
* In line 27, we have specified on which browser the tests should execute.
* In line 28, platform is specified.
* Capabilities in line 31 and 32 is set for Zalenium.
* In line 33, we have provided the URL in which our selenium hub is setup with the capabilities to the RemoteWebDriver function and assigned it to Webdriver variable.

You can also use other capabilities as per your needs.

When you run your script with the desired capabilities inclusion, request with the specified information goes to the selenium grid and accordingly selenium hub will decide to which browser it should be executed.

Following is screenshot of the request selenium hub got when docker selenium image is run.



8. Creation of selenium docker image

First, we will understand what's the logic behind creating docker image of selenium project.

Idea is to build image of the selenium scripts through dockerfile.

So, next question comes in our mind is what command to use in the dockerfile

As Docker image should run on any platform, the selenium project should be independent to run in any machine.

To make the selenium project independent, we have to package all the dependencies within the docker image by creating a runnable jar file of the project using java as the base image.

Then, by running selenium project's jar file, docker image can be built. Hence, the command we will use is java -jar jarfilename.jar.

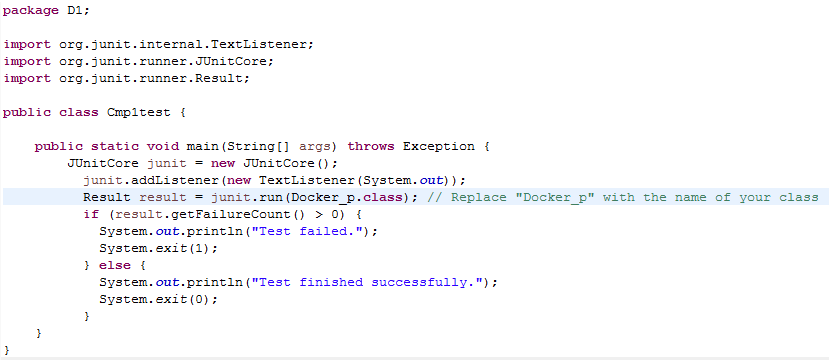
But a runnable jar needs a main method which selenium scripts don't have.

Hence, we have to first create java main class of the project which would run the selenium tests.

8.1. Process of building selenium docker image:

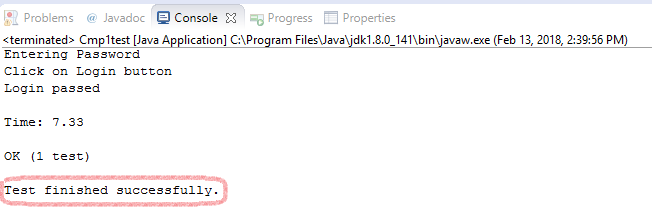
8.1.1 Creating Java main class:

Create a separate java file or add the below code at the beginning of the junit testcase file in your project.



Now run the file as java application (Run As -> Java application).

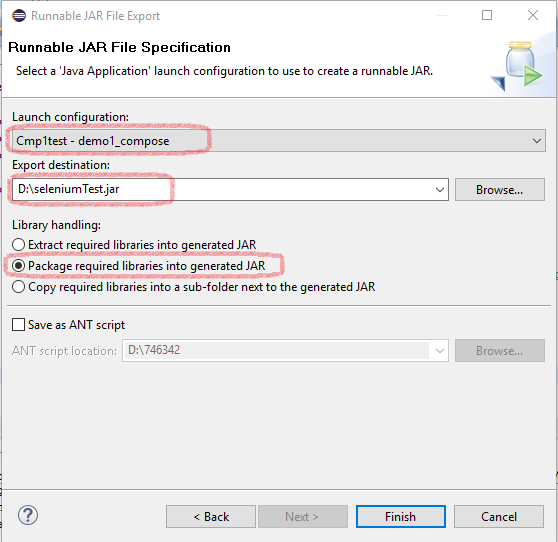
After the execution of java file is finished, you will get a similar output showing test finished successfully if your code is correct.

****

8.1.2. Generate Runnable jar file using eclipse:

Now you can export runnable jar file of your project once your code runs successfully. Procedure is as follows:

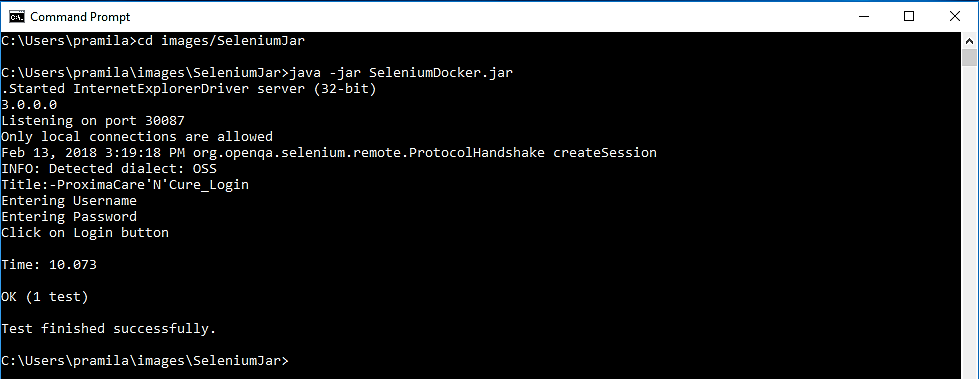
* Right-click on your project and select Export.
* Select Java ->Runnable JAR file as the export destination and click next
* On the next page, specify the name and path of the JAR file to create and select the Launch configuration that includes the project name and the name of the test class.
* Also, be sure to select Package required libraries into generated JAR to embed Selenium libraries into your JAR file. JARs created.
* Click finish and the JAR file is created.



Now, you can find the generated jar file in the specified location.

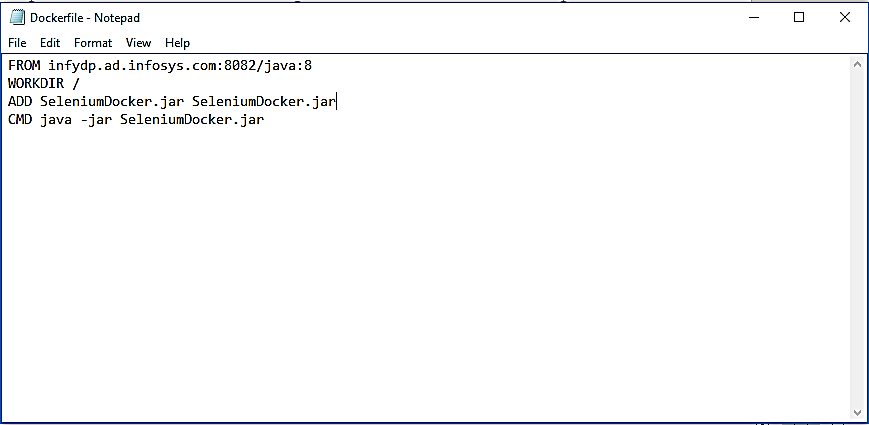
To confirm that the JAR file has been properly packaged, open the command prompt and run this command:

java -jar <path>\filename.jar



8.1.3. Create dockerfile to build docker image

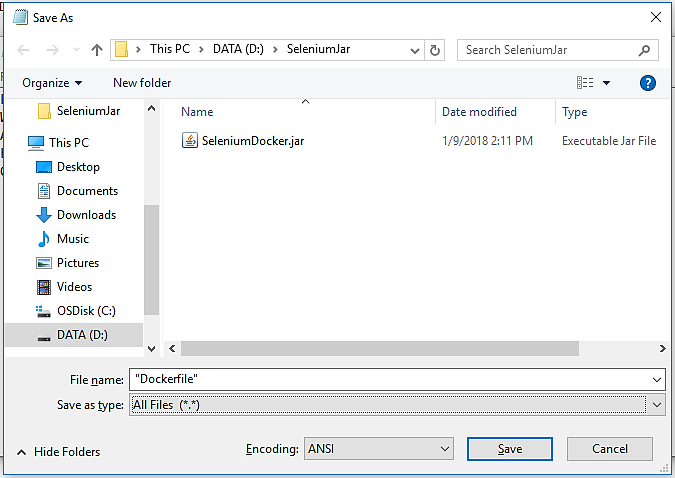
* Now create a docker file with content as follows:



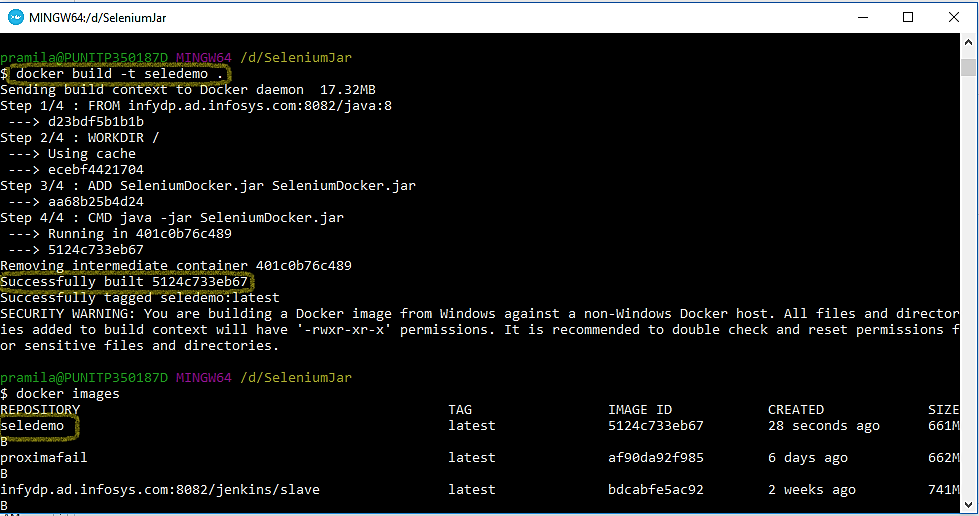
About each line in the Dockerfile:

* First line indicates that our base image is Java.
* Second line creates working directory for the image.
* Third line adds the JAR file from the current directory into the working dir.
* Fourth line executes the JAR file using the java command.

Note: - Name the Dockerfile as “dockerfile” including double quotes and don't give any extension by saving it as all files because by default it saves it as a .txt file. If it is a text file then docker will not recognize it as a dockerfile, hence you need to specify the file name with extension in the docker build command.

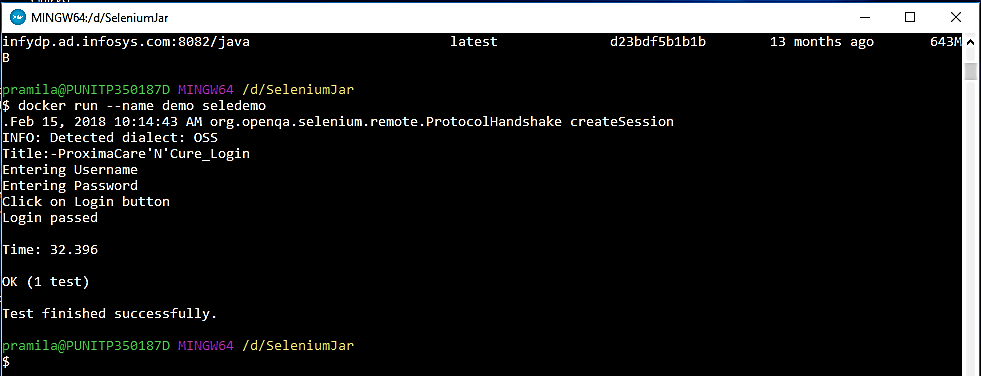


* Create a folder and save the dockerfile and jar file in the same folder.
* Now build the docker file through docker build command as follows



* After building, you can view the docker image built by using the command $docker images.
* Then run the docker image as $docker run image\_name

Docker image will run successfully if every step is followed properly and selenium hub is setup.



Note: - Configure selenium hub before running the selenium docker image with the same port as specified in your test case.

9. How to automate the above process through Apache Maven

We are using Maven as it is an automation tool which provides various plugins to help us automate the previous process.

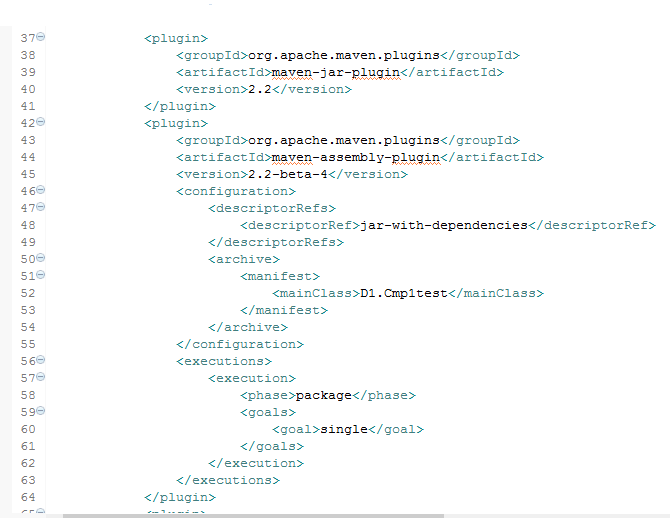
It will help us package our project's dependencies rather than adding external jar to the project manually which will make our project independent and also running our tests in selenium hub rather than selenium web drivers.

Now, convert your selenium project into a maven selenium project without any errors by adding the dependencies in the pom.xml file.

9.1. Generate Runnable jar file through Maven plugins

We will use Maven jar and assembly plugin to generate runnable jar file of the java main class.

Add the below plugins in your pom.xml file probably after maven compiler plugin.



Description of this code:

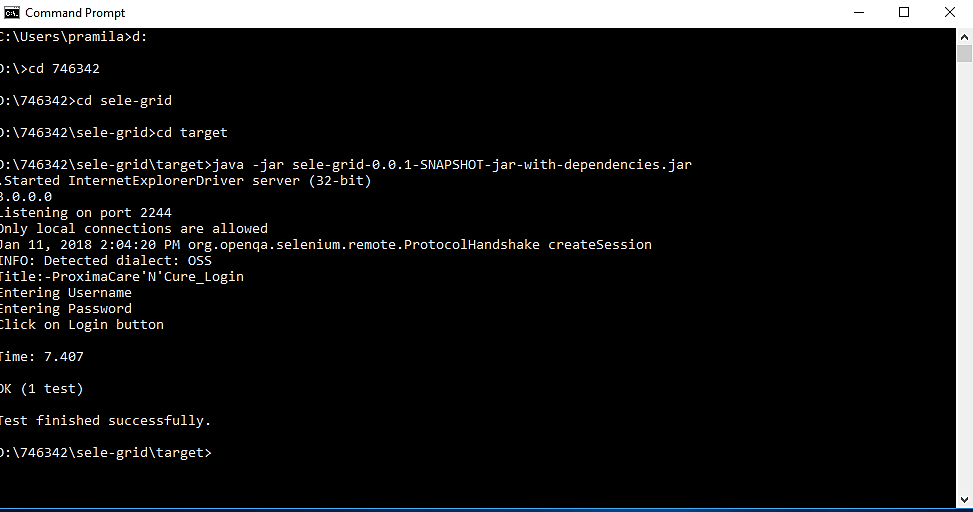
* Maven Jar is used to create the default jar of the project.
* Maven assembly plugin is used to package the jar file with its dependencies by specifying it in the 48th line.
* We mention the java main class name to be included in 52nd line.
* We are instructing the assembly plugin to be executed in package phase lifecycle of maven project in 58th line.
* Finally, we are specifying its goal as single in 60th line.

After placing this code on your pom.xml file, run the project using the command

mvn install (or) mvn clean install

After your project has successfully built, you can notice <projectname>-jar-with-dependencies.jar file inside “target” folder.

Setup selenium hub and run that jar file through command prompt to check whether it is built perfectly or not as follows: -



If you are able to see the “Test finished successfully” output line, then jar file of your selenium maven project is built independently through maven plugins.

Note: - You can use java –jar <path>\....jar as well instead of navigating to the jar file’s directory.

Now, we will take one step further by automating the process of docker creation and building the project docker image.

9.2. What is docker-maven plugin?

To automate the process, we have to build the images in docker, hence we need something to connect maven with docker for which docker-maven plugin is used.

Docker-maven plugin allows us to manage docker images and container from pom.xml file.

For now, we will use docker-maven plugin to generate docker file and build its docker image.

There are several plugins to create dockerfile but we are using the one by Spotify.

To know more about docker-maven plugin, you can refer its official github project <https://github.com/spotify/docker-maven-plugin> .

9.3. Use docker-maven plugin to build the docker image automatically

Add the below code in your pom.xml file after the maven assembly plugin code, as our idea is to create project jar first, and then build the docker image through dockerfile and jar.





Description of this code:

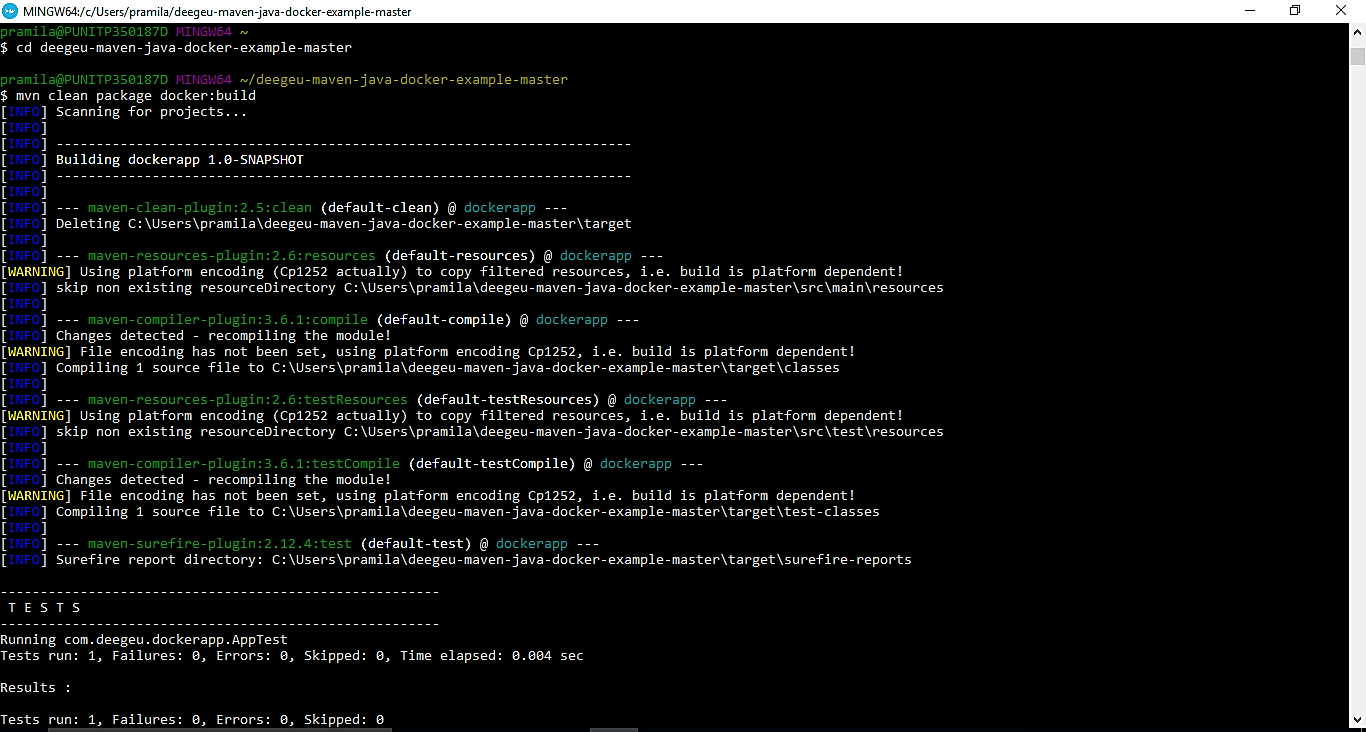
* Inside the execution tag, we are instructing to execute the build goal in package phase at 70th line, as we want to add dockerfile after we’re finished with compiling and testing.

At line 78, we are defining the image name to be built.

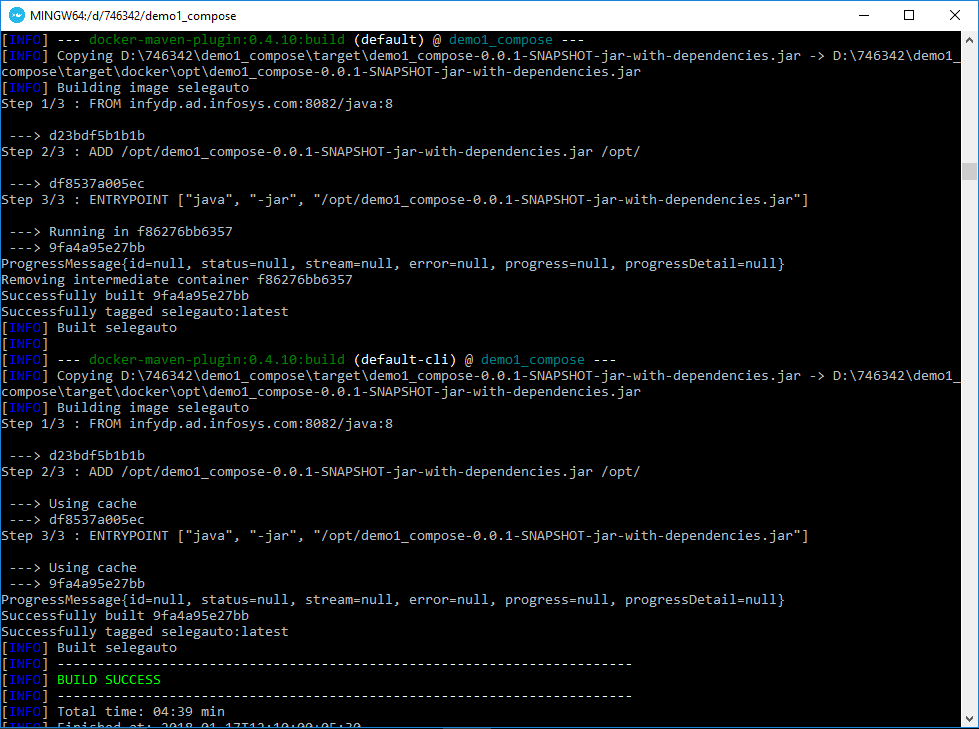
* We are specifying the base image name as Java with the infy registry prefixed in line 79.
* We have provided the main command to be executed in line 81 to run the jar file.
* Target path and directory name of the docker image resource is specified in line 90 & 91 respectively.
* We have instructed it to include only the generated jar file in line 97 excluding other files.

Note: - you can also specify your repository in which the image should be saved in <ImageName> tag by prefixing your repositoryName as deegu/image-name if deegu is your docker repository name.

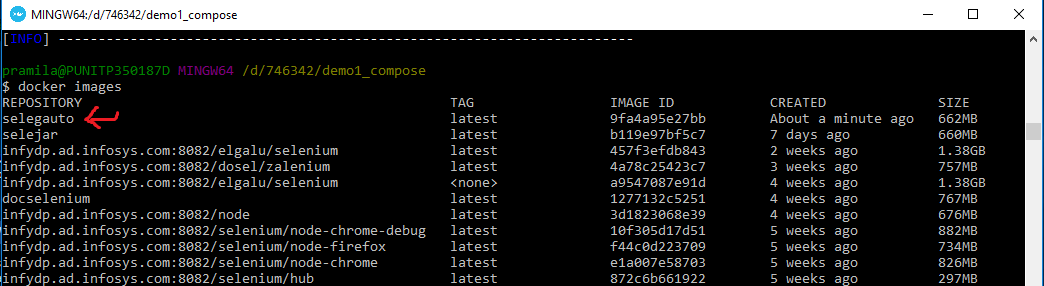
Now, run the maven project in docker terminal using the maven build command as follows: -



Note: - If maven is not able to clean the target folder and throws an error, you can eliminate clean command from the maven build and just type the command mvn package docker: build

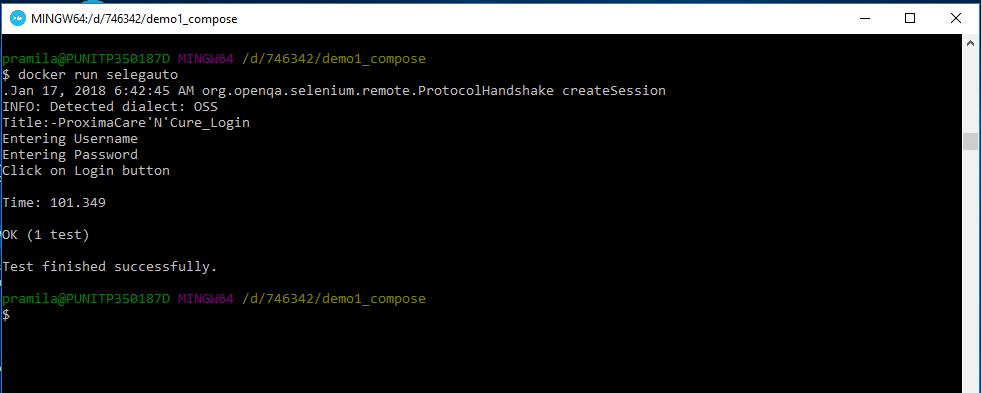


After the maven project is built successfully, you can view your docker image in the docker terminal by executing the following command: -



Then you can spin up the docker image container as follows whenever needed.

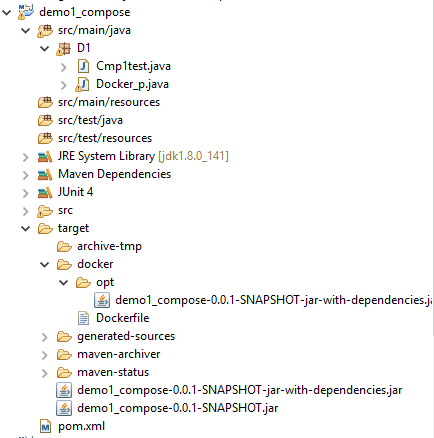
Note: - You can give the container’s name by using –name in the docker run command as “docker run –name demo selegauto”



So, we have successfully automated the process of generating dockerfile and building it through maven plugins.

Let us now understand how docker-maven plugin has worked by looking into our maven selenium project. Following is our maven project’s structure after building it.

In our project, Cmp1test.java is the java main class and Docker\_p.java is a simple selenium script in junit framework.



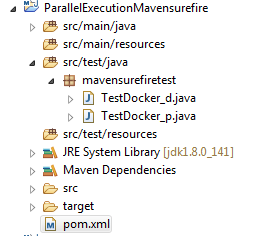
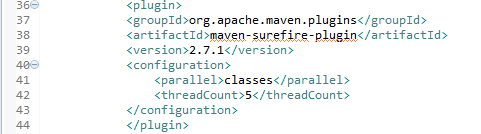
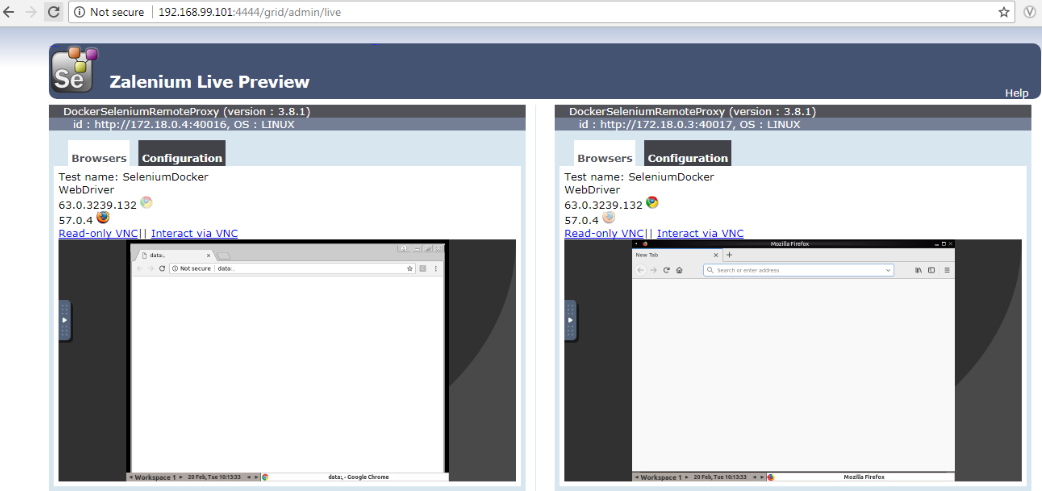
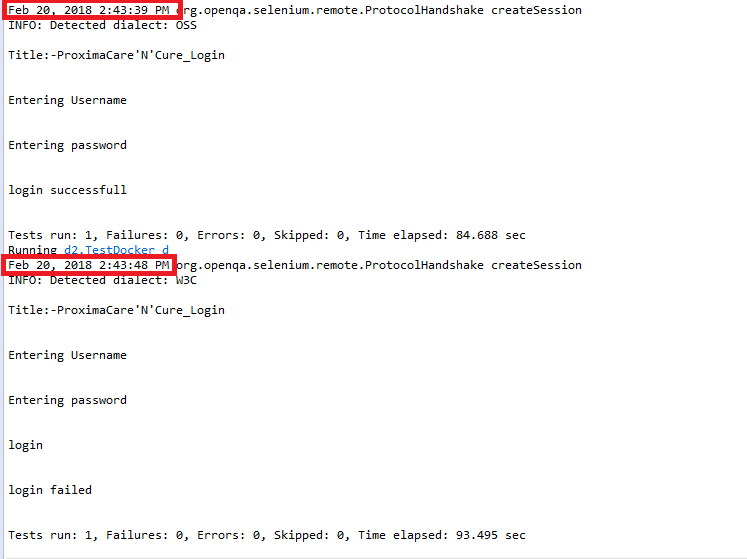
We have automated the following process after setting up selenium grid in docker through docker compose file and passing the maven command: -

* Jar file with dependencies of the selenium maven project is created through maven assembly and jar plugin.
* Dockerfile is created automatically through docker-maven plugin, in docker folder inside target folder of the project.
* jar built through plugin is added to the docker folder which is needed for the docker image to run.
* The same dockerfile is built to create docker image of the project.

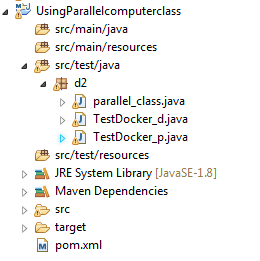
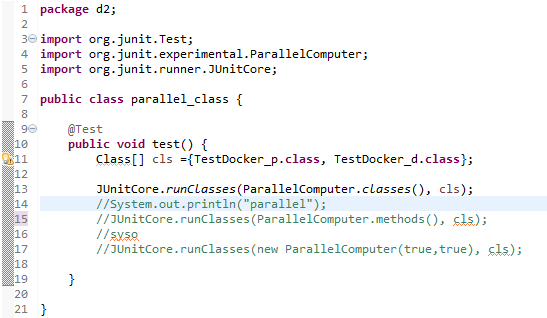
10. Parallel execution in selenium using Junit

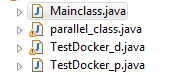
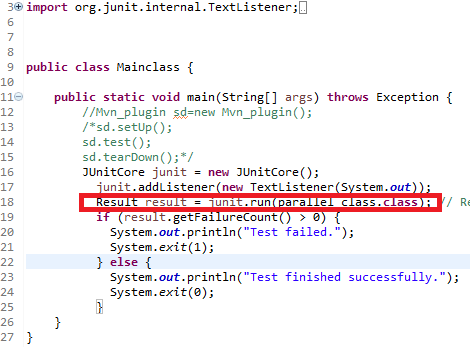
In parallel test execution, you take entire batch and split it up amongst multiple servers so that each server can run a single test at a time. To decrease the execution time of the test suite and this will fasten the feedback loop on which to act upon.

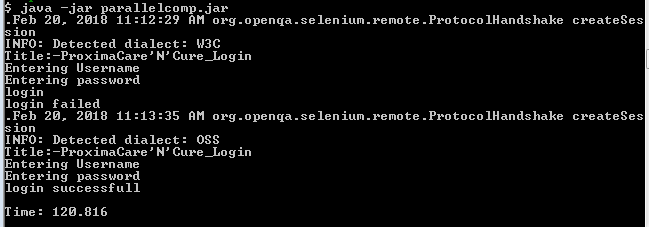
10.1 Using Maven surefire plugin

You must have installed selenium grid or zalenium  
  
  
  
In our project *ParallelExecutionMavensurefire*, we have two classes which we want to execute in parallel. In *TestDocker\_d* we are logging in an application with wrong credentials and in *TestDocker\_p* logging in same application with valid credentials.   
  
  
  
  
  
   
  
We have to add this maven surefire plugin to our POM file to do parallel execution in junit. Line 41 specifies that we want to run the classes in parallel while testing. Now if we want to run the methods in parallel, then we can replace the classes with methods. Line 42 specifies the number of threads to be generated while running our test cases. After implementing the necessary changes, run the POM file. We will see the following results.  
  
  
  
  
Here we can see that the 2 request for execution is passed to grid console.  
  
  
  
  
The above output is of zalenium which can be set using docker. In the above image we can see the two classes of the test execute in parallel on different nodes, one on **Chrome** and other on **Mozilla Firefox.**Above is the screenshot of console output. The execution time is same for both the classes.

10.2 Using ParallelComputer Class of Junit

Pre-requisite: You must have installed selenium grid or zalenium  
The project is same, we have same two classes *TestDocker\_d* and *TestDocker\_p* which is to be executed in parallel. But here instead of *maven surefire plugin*, we will be using *ParallelComputer* class, an in-built class of Junit.  
   
  
In the project we have added parallel\_class.java, which will use the ParallelComputer class of junit. The parallel\_class contain the following code.  
  
  
  
In line11 we have specified the classes, which is to be run in parallel. In line 13 we have use the JUnitcore command which will the classes in parallel, similarly line14 will methods in parallel of the specified class if required. Now if you want to run both class and methods in parallel then we need to use the command in line 17. After executing the script, we will get the same output as mentioned in **Using maven surefire plugin.**

10.2.1 To make jar of the parallel execution script  
  
  
  
In order to make jar file of our script we should have a main class, so we have added Mainclass.java in our project as shown above.  
  


In line 18 I have just called my parallel\_class so that the classes execute in parallel.  
**Now we can follow the steps given in Generate Runnable jar file through eclipse or Maven plugins**. Once the jar is created, we can see the below output after execution: -

After this you can create an image, refer **Process of building selenium docker image** for the sameand run in Docker.

11. Running multiple selenium docker images

We saw in the previous topic that how to run more than one testcases in parallel using maven and junit framework. We can run two docker selenium images at the same time in selenium hub using docker-compose. We can use this strategy for running multiple tests simultaneously.

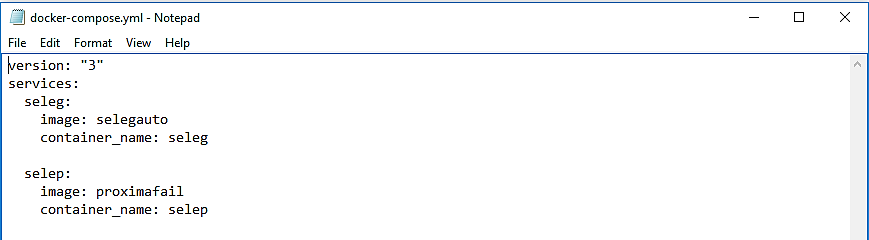
For example, we can create docker images of various independent modules in functional testing and run them in parallel through docker.

**Procedure**: -

Prerequisite: Build minimum 2 selenium docker images by referring the earlier topics and configure selenium grid through selenium hub or zalenium image.

To orchestrate docker images: -

* Create docker-compose.yml file in a directory with the following contents: -



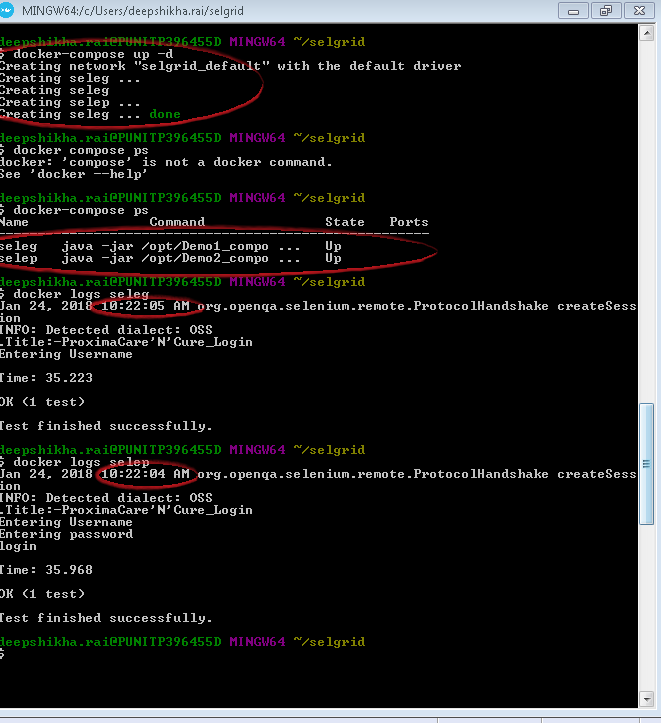
* Description about contents in docker-compose file: -
  + We have specified the service name for each selenium docker image.
  + We have specified respective docker image’s name to be run using “image:” attribute.
  + We have specified the container name to be given for the respective images using “container\_name” attribute.

So, you can modify this docker-compose file according to the selenium docker images you have built and going to use.

Here, we have used selegauto and proximafail selenium docker image which are build using maven in docker.

Note: - Indentation of contents in the docker-compose file should be proper as it may throw compilation error while running the docker-compose file.

* Start up the docker-images through docker-compose up –d command by navigating to its directory.
* Check the services startup by docker-compose using docker-compose ps command.
* You can see the logs of each services to see the output of the tests using the command docker logs <container\_name>.
* Refer below screenshot for the same.



By using docker-compose tool to run our selenium docker images, we can run the images at one go and also stop them by a single command in which the exited containers are removed. automatically

So, we need not run the docker images manually all the time.