|  |
| --- |
|  |
| Explore Docker |
| Basics :: Installation: POC |
|  |
| **Vibranarayanan** |
| **5/16/2018** |

|  |
| --- |
|  |

Table of Contents

[Change Log 3](#_Toc514871032)

[Docker Terms 4](#_Toc514871033)

[Docker work flow 5](#_Toc514871034)

[Virtualization Vs Containerization 5](#_Toc514871035)

[Containers on VM 6](#_Toc514871036)

[Docker Client-server architecture 6](#_Toc514871037)

[Docker Benefits 6](#_Toc514871038)

[Portability 7](#_Toc514871039)

[Version Control 7](#_Toc514871040)

[Installing Docker Toolbox in Windows 7](#_Toc514871041)

[Pre requisites 7](#_Toc514871042)

[VM virtual box with Linux 674 bit 7](#_Toc514871043)

[Running Docker Linux Vs Window 8](#_Toc514871044)

[Current system Configuration 8](#_Toc514871045)

[Hyperv detection Tool 8](#_Toc514871046)

[Download & Install Docker Tool 9](#_Toc514871047)

[Install Docker Tool 9](#_Toc514871048)

[Step by Step Installation screen 9](#_Toc514871049)

[Verify Docker commands after installation 12](#_Toc514871050)

[Docker Hello-world 12](#_Toc514871051)

[Launching Simple Spring boot application in Docker 13](#_Toc514871052)

[Create Spring project structure 13](#_Toc514871053)

[Project structure in downloaded folder 13](#_Toc514871054)

[Open this project in eclipse 13](#_Toc514871055)

[Project Explorer 13](#_Toc514871056)

[Write a simple REST end point resource 14](#_Toc514871057)

[Build the project 14](#_Toc514871058)

[Create a Docker File 14](#_Toc514871059)

[Docker File Explained 14](#_Toc514871060)

[Build docker image 15](#_Toc514871061)

[Run created image (Create Container) 16](#_Toc514871062)

[Container output in browser 16](#_Toc514871063)

[Issues faced during this container creation 17](#_Toc514871064)

[Java not found error inside container 17](#_Toc514871065)

[Unable to access container via browser 17](#_Toc514871066)

[Port 8085 is already in use 17](#_Toc514871067)

[Build REST Client and server Containers 17](#_Toc514871068)

[Build REST API/SERVER 17](#_Toc514871069)

[Project Explorer Snippet 18](#_Toc514871070)

[Model Class 18](#_Toc514871071)

[Service Class 19](#_Toc514871072)

[Controller Class 19](#_Toc514871073)

[Changes in Main method 20](#_Toc514871074)

[Docker file changes 20](#_Toc514871075)

[Build Docker image 20](#_Toc514871076)

[Run Image/creating container 20](#_Toc514871077)

[Accession REST API/end points 20](#_Toc514871078)

[Build REST Client application 21](#_Toc514871079)

[Create a project 21](#_Toc514871080)

[Client class 21](#_Toc514871081)

[Changes in Main class 21](#_Toc514871082)

[Build this project 22](#_Toc514871083)

[Create a Docker file 22](#_Toc514871084)

[Build image 22](#_Toc514871085)

[Run Image/container 22](#_Toc514871086)

[REST Client Container-output 22](#_Toc514871087)

[Currently running containers 23](#_Toc514871088)

[Docker Compose 23](#_Toc514871089)

[What is Docker Compose 23](#_Toc514871090)

[Example multi container 23](#_Toc514871091)

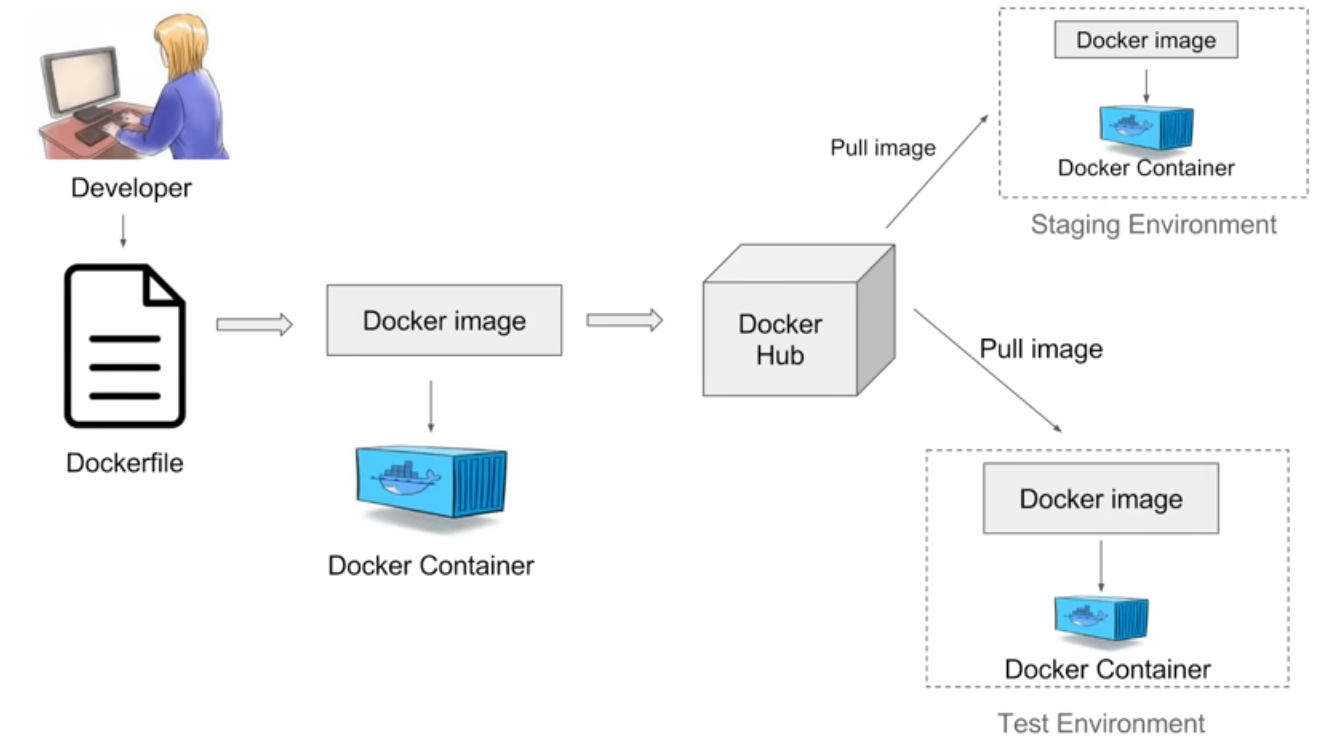
## Change Log

|  |  |
| --- | --- |
| Date | Details |
| 05/16/2018 | * Docker terms * Docker work flow * Virtualization vs Containerization * Client server architecture * Benefits |
| 05/17/2018 | * Docker Tool installation on windows * verify Docker commands after installation * Running hello world Docker image * Learn how to create Docker file * Running Docker file with existing repo image.   Pages (6 to 10) |
| 05/21/2018 | * Setting-up local instance * Creating simple spring boot application * Enable rest service * Create Docker file with this spring boot app and create image * Launch this image into container and access via browser.   Pages (11 - 16) |
| 05/22/2018 | * 05/21issue analysis and fixes * Build JAVA REST Server application in a container * Build JAVA REST client application in a container and consume REST API running in another container.   Pages (17 to 23) |
| 05/23/2018 | * Docker Compose basics   Page (23) |

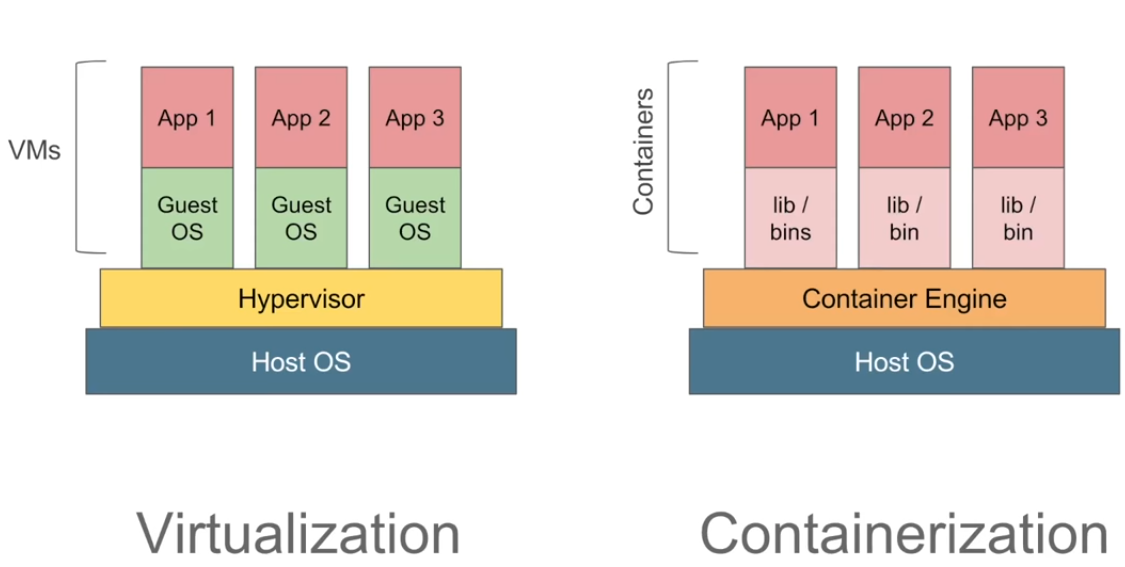
## Docker Terms

|  |  |
| --- | --- |
| Term | Explanation |
| Docker file | * Describes steps to create Docker image |
| Docker image | * Created using Docker file * An **image** is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files. dependencies |
| Docker container | * Created by running Docker image. * Run time instances of Docker image. * A **container** runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight. |
| Docker Hub | * Online repository, images can be stored in this repository |
| Docker Client | * Command line inter face to interact with Docker server. |
| Docker Server/Daemon | * Will have all the containers |
| Docker Engine | * Combination of Docker Client and server component. |

## Docker work flow



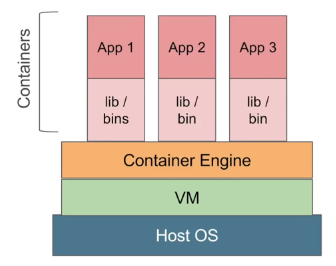
## Virtualization Vs Containerization



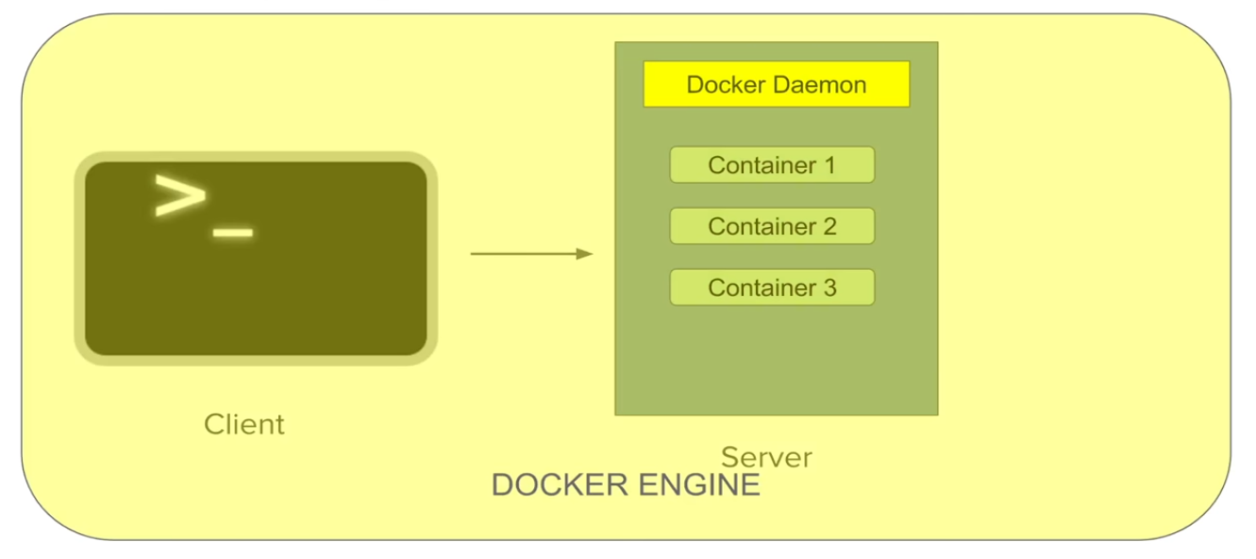
|  |  |
| --- | --- |
| **Virtualization** | **Containerization** |
| * Create Multiple VMs using Hypervisor in Host OS * VMs have their own OS and does not use host OS. * Overhead on Host platform * Each VM fixed memory need to be allocated. this leads to wastage of memory. | * Containers are light weight * Container engine will manage/use Host OS configurations/memory based on applications in containers. * No memory/space overhead. |

## Containers on VM

There may be a scenarios like container need to be run on VM. In that case Containers can be managed in below stacks.



## Docker Client-server architecture



* Docker server/Docker Daemon receives command from client in this form of CLI or rest API.
* All the Docker Client and server together form Docker engine.
* Docker client or Docker Daemon can be present in same host or in different host.

## Docker Benefits

* An Application inside a container runs on any system that Docker installed.
* Build application only once and no need to configure multiple times.
* Docker images can be maintained in any repository , later images can be pulled for usage.
* Test your application inside your container and ship it inside the container, This means Environment in which you test is identical with the app in production.
* *Isolation* is the Key, with Docker every application works in isolation in its own container. Does not interferes with other application running on the same system.
* Removal of an application is easy by deleting a container.
* Developer can package software with all its dependency and Docker will take care if running those application in different platform.
* *Productivity* is an another key. Docker allows faster and efficient deployment without worrying about, application running on different planform.

### Portability



* Docker containers can run on any platform. This can run on local system and Amazon ec2 or Google Cloud, etc..
* Container running on AWS can easily be ported to Virtual machine.

### Version Control

* Like Git Docker has in-build version control.
* Docker containers work just like a GIT repo, allow you to commit your changes into Docker images and version control them.
* Docker images can be tracked in other version control system like GIT.

## Installing Docker Toolbox in Windows

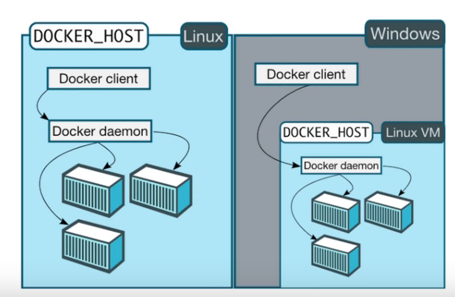
### Pre requisites

review system requirement and install in your windows system. your OS should be in 64 bit architecture. get installation information from this [Link](https://docs.docker.com/toolbox/toolbox_install_windows/#step-1-check-your-version)

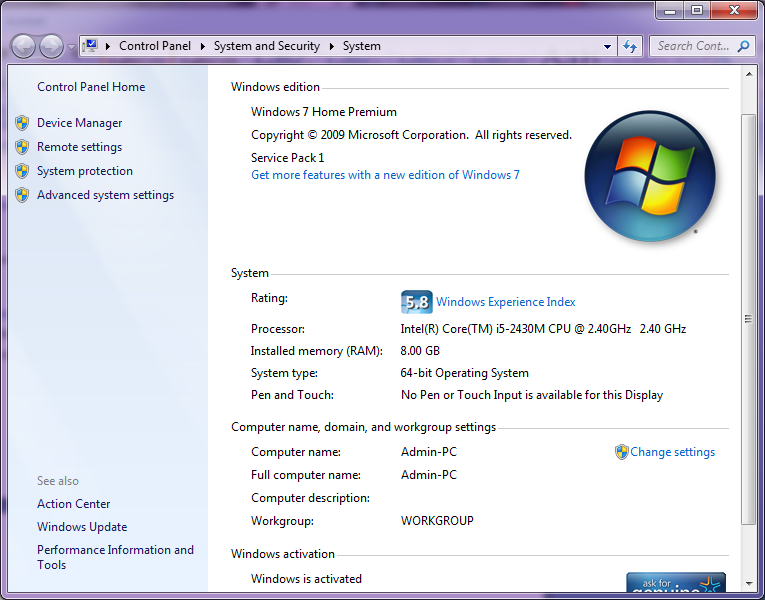
### VM virtual box with Linux 674 bit

from higher version of windows 7, by enabling Virtualization at BIOS level and disabling Hyper-vT in windows feature. you can start installing 64 bit version linux flavour. for more details refer this [link](http://www.fixedbyvonnie.com/2014/11/virtualbox-showing-32-bit-guest-versions-64-bit-host-os/#.Wv0vLsIQDIU)

### Running Docker Linux Vs Window

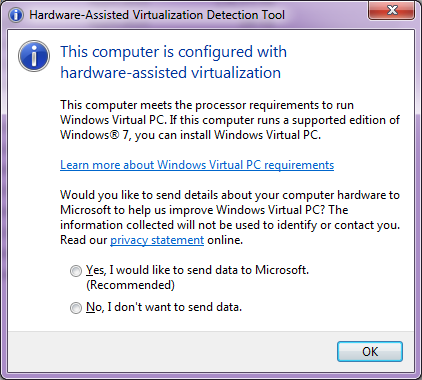


### Current system Configuration



### Hyperv detection Tool

As mention in this [link](https://docs.docker.com/toolbox/toolbox_install_windows/#step-2-install-docker-toolbox) under Window section downloaded havdetection tool and verified, i am good with run windows virtual PC. [Download URL](http://www.microsoft.com/en-us/download/details.aspx?id=592)



### Download & Install Docker Tool

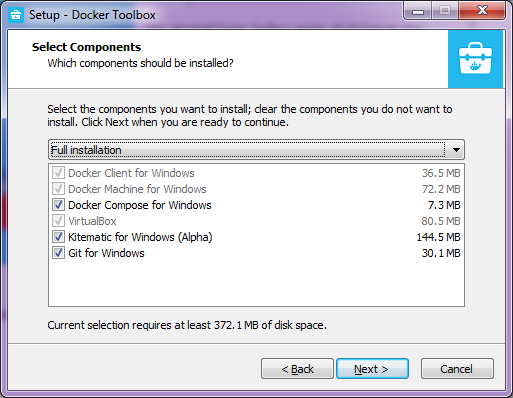
Down load Docker tool from this [URL](https://docs.docker.com/toolbox/toolbox_install_windows/)

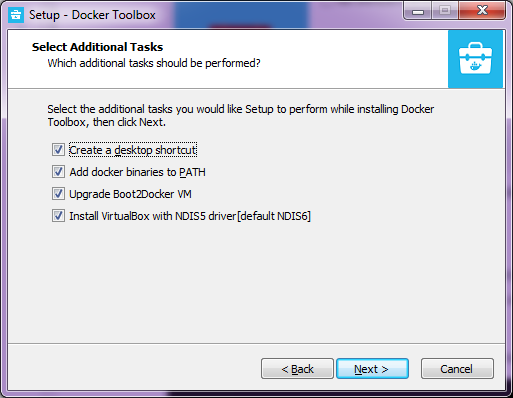
### Install Docker Tool

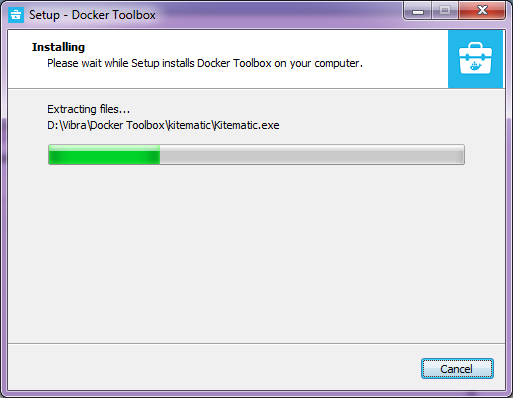
Install Down loaded docker tool.

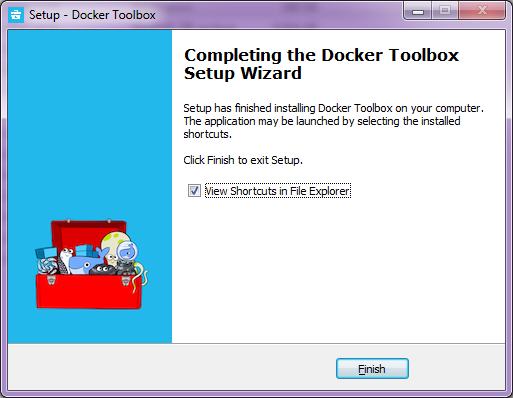
### Step by Step Installation screen

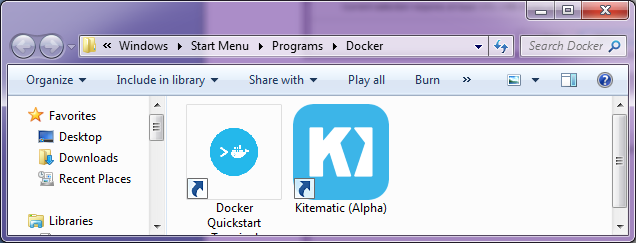




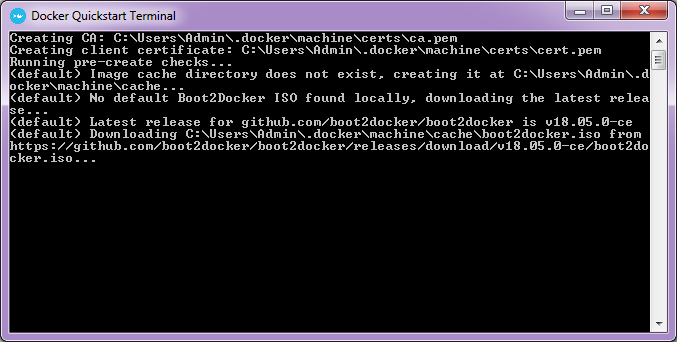




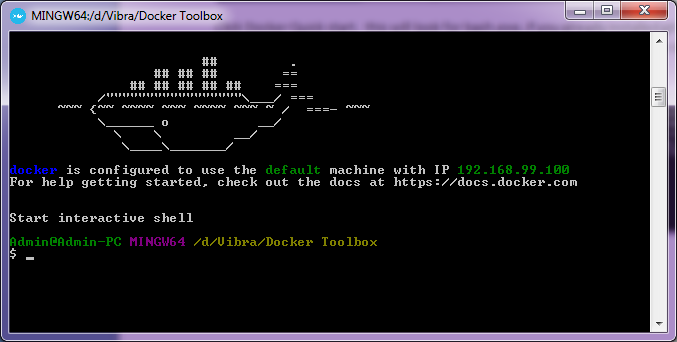




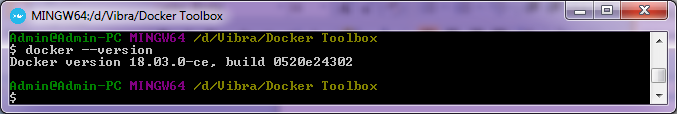
Click Docker Quick start. this will look for bash.exe, if you already installed git this will try to find that bash.exe location. if you sure about location you can point that location by browsing option provided.



#### Final screen after installation

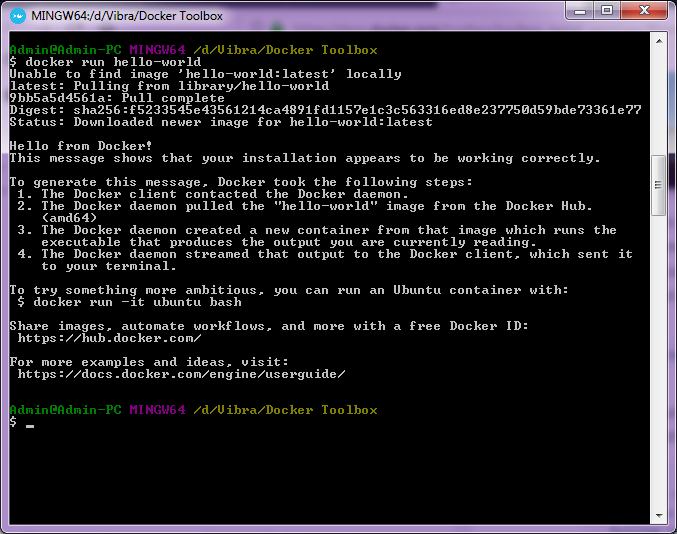


### Verify Docker commands after installation



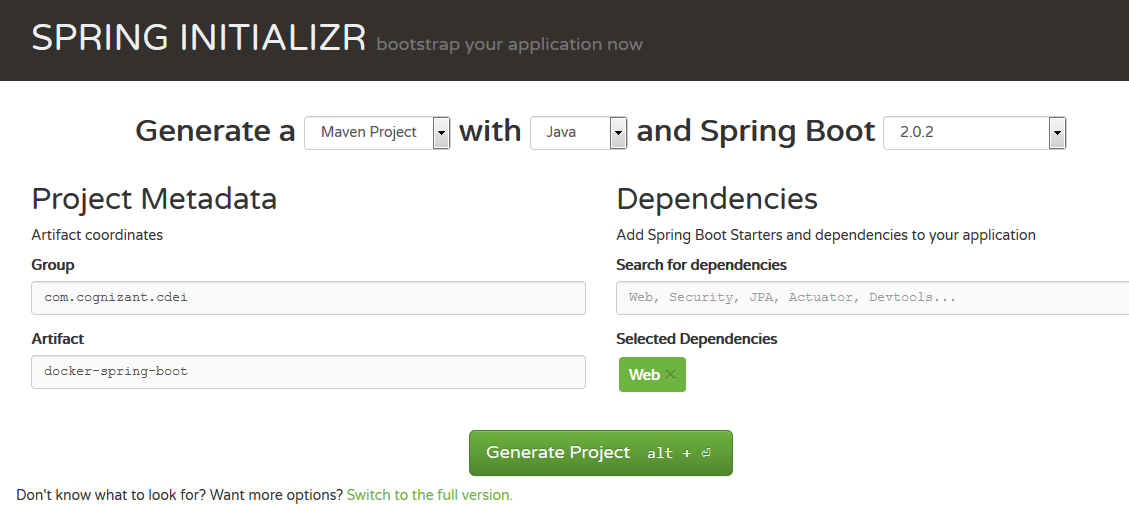
### Docker Hello-world

In command line type below to check the installation   
$docker run hellow-world  
This will look for local image, and if it doesn't find it will get it from common repro

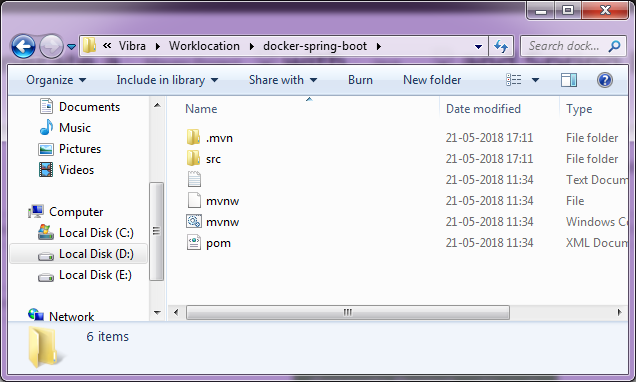


## Launching Simple Spring boot application in Docker

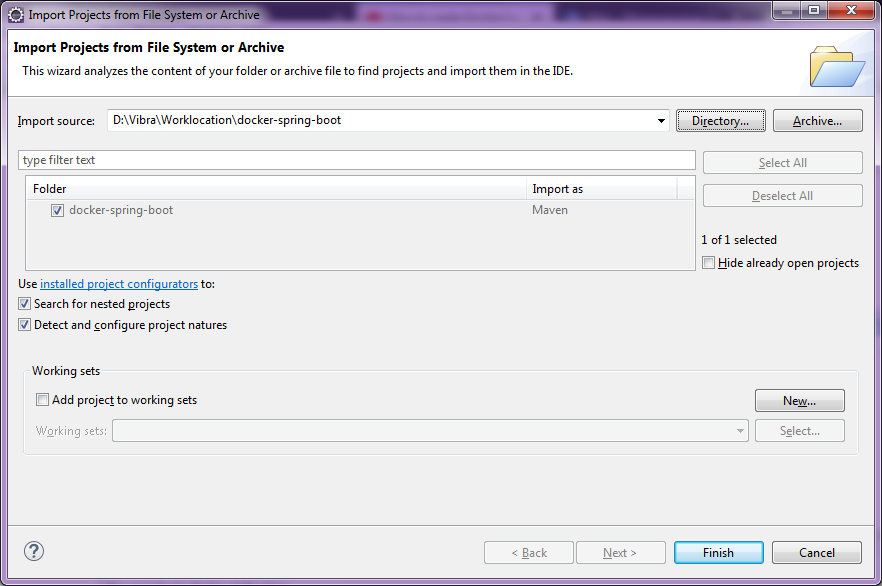
### Create Spring project structure

For easiest way go to http://start.spring.io/ , provide required details and click generate project. 

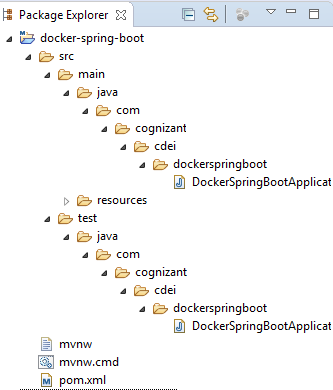
### Project structure in downloaded folder



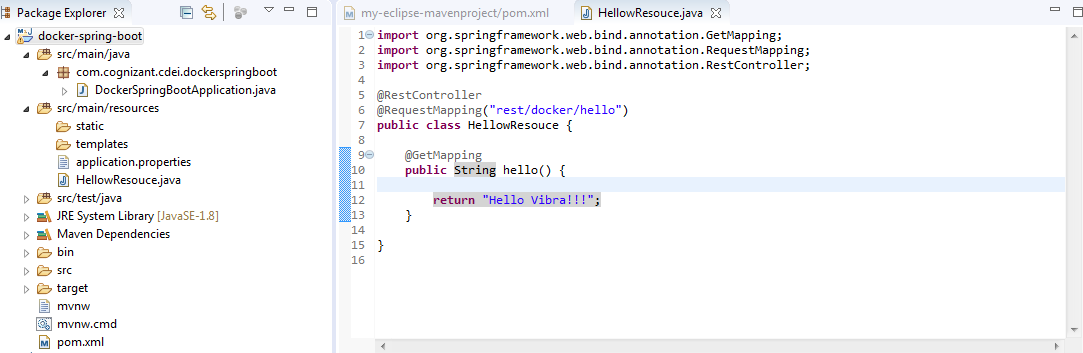
### Open this project in eclipse



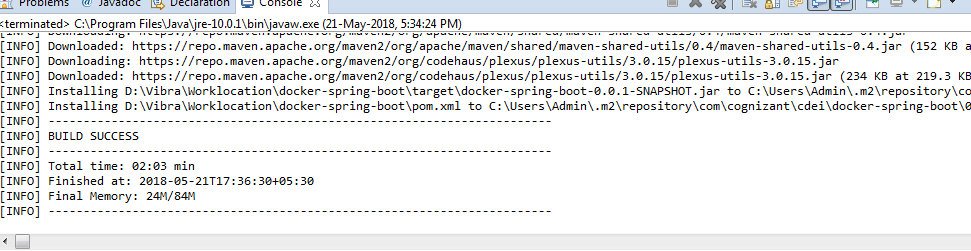
### Project Explorer



### Write a simple REST end point resource

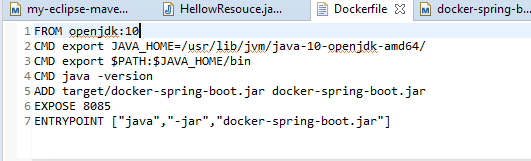
Create below class fine under resources package

### Build the project



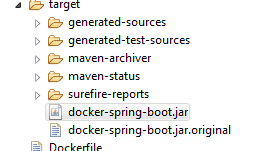
### Create a Docker File

Create a docker file with below commands to run created application in docker container.



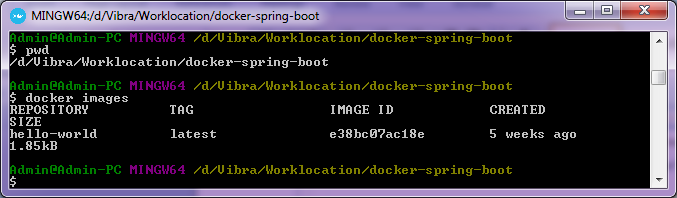
### Docker File Explained

* FROM openjdk:10 : This line will pull open JDK version 10 from docker hub. since my local environment created in using JSK 10.
* ADD : this line copy created jar file from "/target/docker-spring-boot.jar" to containers root directory with same name.

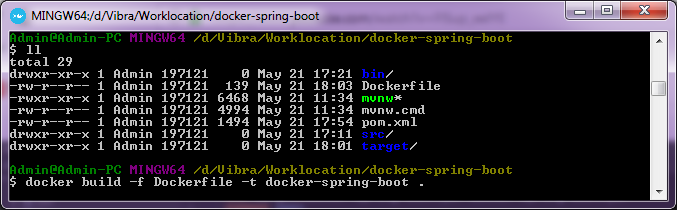


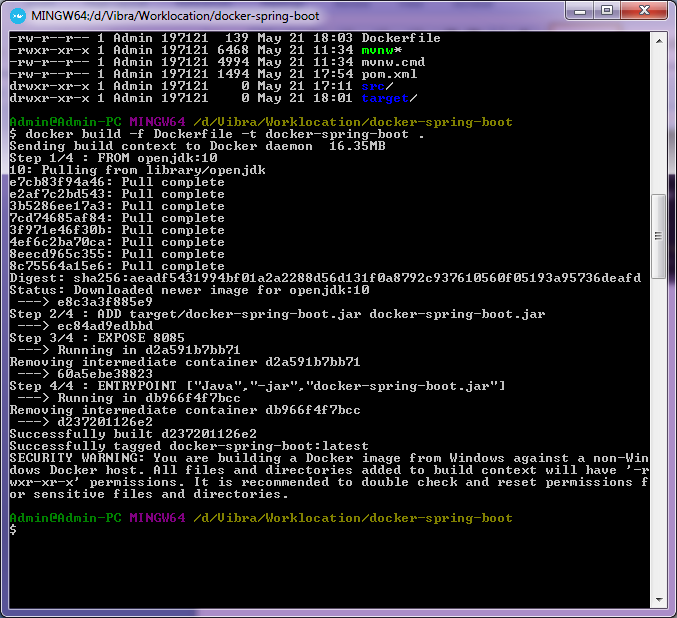
* EXPOSE: this line will expose port 8085to use.
* ENTEYPOINT: This the final command to execute our created jar file inside the container.

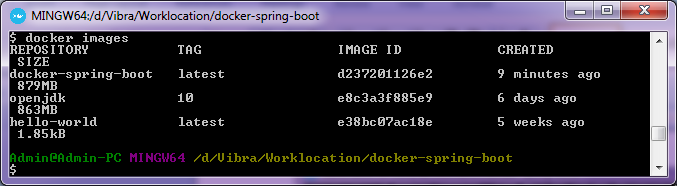
### Build docker image

Navigate to a directory where we developed our spring boot application. and start build the image.  
above screen will list the current directory and currently running images.

Execute build command: docker build -f Dockerfile -t docker-spring-boot .

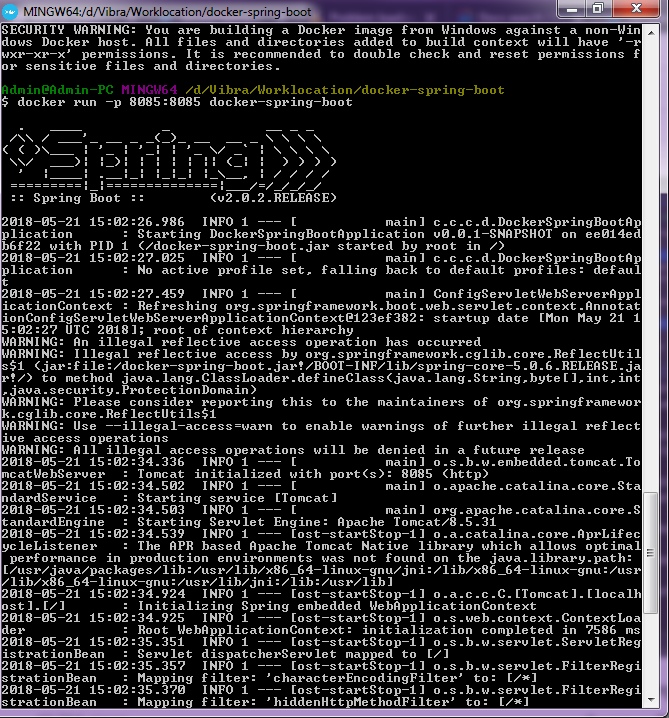
Docker build command will take -f argument "Dockerfile" for creating docker image. -t will be the tag option for this image. image will be created in this name. "." will be the current location to build the images.

Final screen after building image from docker image.In above screen, we can see each step we mention in Dockerfile executed one by one.

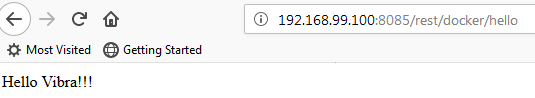
Openjdk and docker-spring-boot image available after build our image from Dockerfile

### Run created image (Create Container)

execute command "docker run -p 8085:8085 docker-spring-boot" this will run created image in port 8085. -p option will publish this application in 8085. first one implies local applications port and 2nd one mapping this to containers 8085 port.



### Container output in browser



## Issues faced during this container creation

### Java not found error inside container

#### Error:

$ docker run -p 8085:8085 docker-spring-boot   
D:\Vibra\Docker Toolbox\docker.exe: Error response from daemon: OCI runtime create failed: container\_linux.go:348: starting container process caused "exec: \"Java\": executable file not found in $PATH": unknown.

#### Resolution:

Verified Java version and Java path inside container and exported JAVA\_HOME and PATH variable with JRE path.  
Steps taken  
1) inspected container and verified java path. command docker inspect <imagename>. found java home and path variable empty.  
2) located Java path. by executing cd and ls command via docker file and located java executabe. CMD command used in docker file. (CMD cd <location> CMD ls -ltr  
3) Exported those path in Docker file before executing Entrypoint step.  
4) Application executed successfully.

### Unable to access container via browser

#### Error:

Tried with http://localhost:8085. Got result as page not found and not available.

#### Resolution:

suspected there could be a IP issue. found that docker-ip command will show the container IP. and access application using that IP and it worked.http://192.168.99.100:8085/, however it showed error page. added REST URL in this URL and got the expected result. http://192.168.99.100:8085/rest/docker/hello

### Port 8085 is already in use

During issue fix, ran build and run command multiple time and when i execute run command second time i got error message as port 8085 is already is in use. To overcome this, stopped the container before running run command.

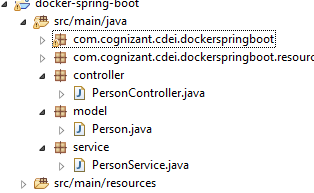
## Build REST Client and server Containers

## Build REST API/SERVER

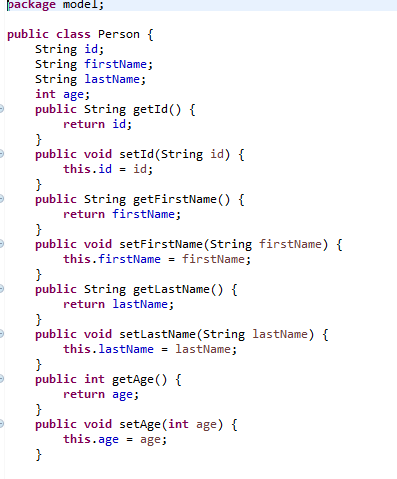
in above example we created a application to display text output. in this POC we will alert this Application to return some data in JSON format.  
Create packages  
To follow the standards created packages Model, Service and controller. like below.

* Person Model .java - This is a Java class for person this class contains attributes for a person.
* PersonService.java - This class will add details to Person model and return by id or all.
* PersonControler.java - This will map the request and get the info from service and publish it.

### Project Explorer Snippet



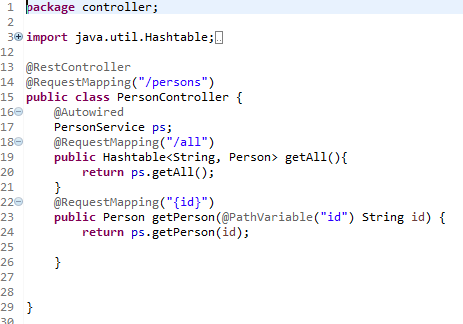
### Model Class



### Service Class

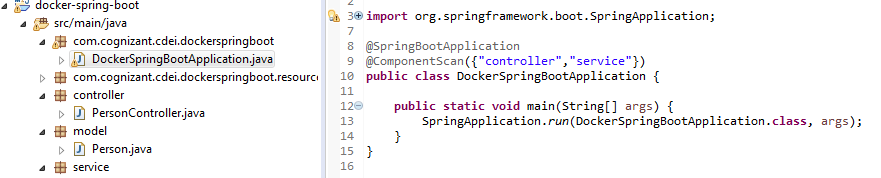


### Controller Class



### Changes in Main method

Annotation component scan added to check for controller and service.



### Docker file changes

No change required in Docker file.

### Build Docker image

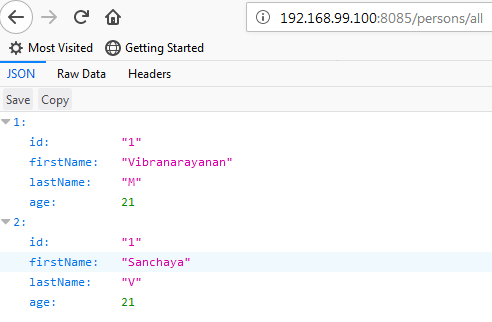
Execute below command and create a docker image.  
$ docker build -f Dockerfile -t docker-spring-boot

### Run Image/creating container

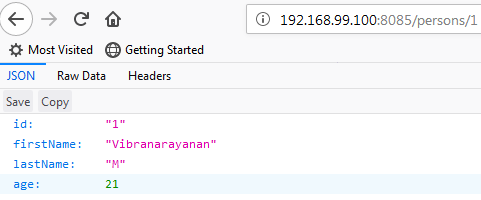
execute below command to create container and publish it in 8085 port.  
$docker run -p 8085:8085 docker-spring-boot

### Accession REST API/end points

API can be accessed by calling/accessing any one of the resources created.  
Persons/all

******

persons/{id}

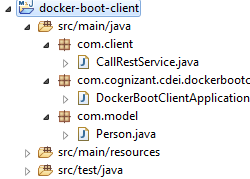
******

## Build REST Client application

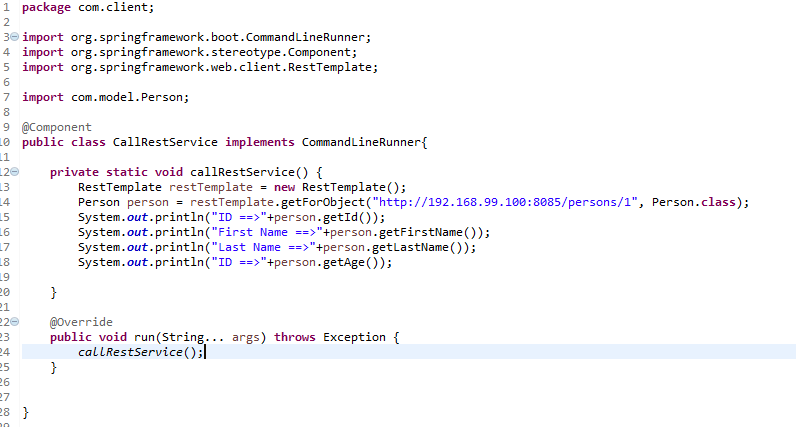
This application will call rest server/API and publish the responses from API call.

### Create a project

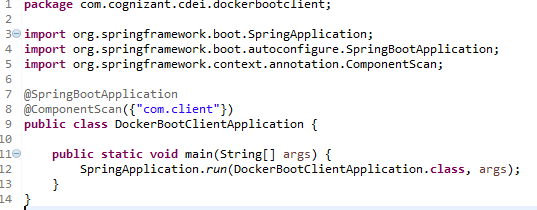
Create a project. you can follow the same steps mention in this [step](#_Launching_Simple_Spring) and create packages "com.client" and "com.model" model will be same as person.java we created in previous project you can use that class file.



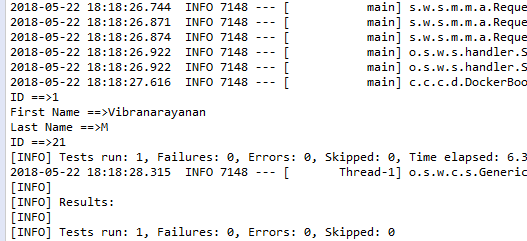
### Client class

This class will call the rest API with argument 1. which will result in showing results with 1

### Changes in Main class



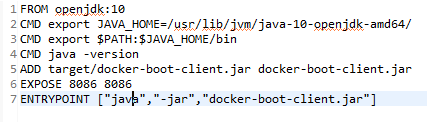
### Build this project



in above screen you can see ID, First, last Name and ID displayed. this is the response from another container.

### Create a Docker file

we can copy and paste the file which we created for rest API. Since we are going to run in same environment without UI.

  
you can note that ADD command with updated with new jar file created for this client and port exposed to 8086 with these two update you can build image and run the container.

### Build image

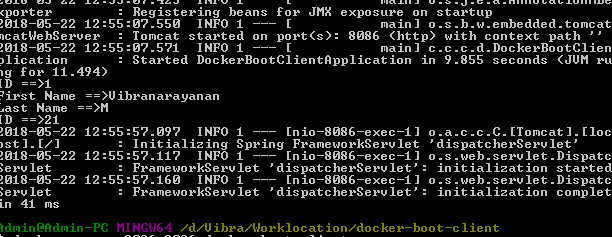
you can run below command to build a docker rest client.

$ docker build -f Dockerfile -t docker-boot-client .

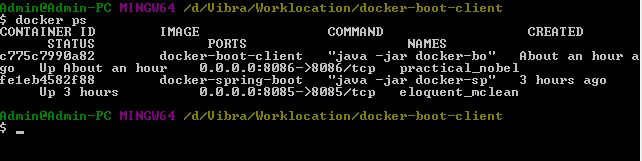
### Run Image/container

By executing below image you can run images in containers and see the results.  
$ docker run -p 8086:8086 docker-boot-client

### REST Client Container-output



### Currently running containers



## Docker Compose

## What is Docker Compose

Docker compose used to run multi-container applications. each container can run a standalone application and it can communicate with other containers present in same host.

### Example multi container

