

# Docker Training

## Table of Contents

1 Creating Docker Files to create Docker Images.....	2
example 1 : Hello World Without Arguments.....	2
Example 2 : Hello World With Arguments.....	3
Example 3: Simple Java Hellow World.....	4
Example 4 : Hello World Apache Webserver.....	5
Example 5: Java Spring Boot Microservice.....	6
2 Portainer.....	12
Install Portainer running these commands in a terminal:.....	12
3 Getting Familiar with docker commands.....	13
Listing docker Images.....	13
Listing All Docker Containers.....	13
Starting a listed container.....	13
Run and Delete a container afterwards.....	13
Deleting all Docker Containers.....	13
Pulling (down) Docker containers.....	14
Show Details of Docker containers.....	14
List all Containers.....	14
List all Images.....	14
Stopping a container.....	14
Deleting a container.....	14
Deleting an Image (This is permanent).....	14
Running containers :.....	15
Launch a container in the interactive mode.....	15
Attached Mode.....	15
Detached Mode.....	15
Running Containers in Attached mode:.....	15
Running Containers in Detached mode (as a service):.....	15
Run and then delete the container:.....	15
Running a webserver with internal port 80 and external port 4000 and 4001.....	15
Pruning Docker Containers Images and volumes.....	16
Pruning unused Docker Containers.....	16
Prune unused Docker Images.....	16
Limiting images pruned to only images created more than 24 hours ago:.....	16
Prune Volumes (this is non recoverable).....	16
Removing volumes which are not labelled with the keep label:.....	16
4 The Dockerfile Syntax.....	17
5 Docker Swarms.....	18
Difference between Docker and Docker Swarm.....	18
Worker Nodes.....	18
Create Swarm.....	18
Joining a swarm.....	18
Creating workers.....	18
Joining another node as a worker to this swarm.....	18

Please note this Tutorial will be run on Linux

# 1 Creating Docker Files to create Docker Images

## example 1 : Hello World Without Arguments

**Step 1 : Create a folder for your Docker Image and open this folder:**

```
hello0/
```

**Step 2: Inside this folder create a file called: Dockerfile with this content:**

```
FROM amytabb/docker_ubuntu16_essentials
COPY HelloWorld /HelloWorld
WORKDIR /HelloWorld/
RUN g++ -o HelloWorld helloworld.cpp
CMD ["/HelloWorld"]
```

**Step 3: Inside this folder create a folder called HelloWorld**

**step 4: Inside this folder create a C++ file : helloworld.cpp with this content:**

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world 0!" << endl;
    return 0;
}
```

**Step 6: Run the following commands to create the Docker Image:**

```
sudo docker pull amytabb/docker_ubuntu16_essentials
```

```
sudo docker build -t hello0 .
```

**Step 7: Launch the Docker Image using Docker:**

```
sudo docker run -it hello0
```

## Example 2 : Hello World With Arguments

**Step 1: Create a folder for your Docker Image and open this folder:**

hello1/

**Step 2: create a file called: Dockerfile with this content:**

```
FROM amytabb/docker_ubuntu16_essentials
ENV NAME VAR1
ENV NAME VAR2
ENV NAME VAR3
COPY run_hello1.sh /run_hello1.sh
COPY HelloWorld /HelloWorld
WORKDIR /HelloWorld/
RUN g++ -o HelloWorld1 helloworld1.cpp
WORKDIR /
CMD ["/bin/sh", "/run_hello1.sh"]
```

**Step 3: Inside this folder create a folder called HelloWorld**

**Step 4: Inside this folder create a C++ file : helloworld.cpp with this content:**

```
#include <iostream>
#include <string>
using namespace std;

int main(int argc, char **argv)
{
    cout << "Hello world 1, with arguments!" << endl;

    string val;
    for (int i = 1; i < argc; i++){
        val = argv[i];
        cout << "Argument " << i << " " << val << endl;
    }
    return 0;
}
```

**Step 5: Inside this folder create a shell script file: run\_hello1.sh with this content:**

```
#!/bin/sh
./HelloWorld/HelloWorld1 $VAR1 $VAR2 $VAR3
```

**Step 6: Inside this folder run the following command:**

```
chmod 775 run_hello1.sh
```

**Step 7: Inside this folder run the following command:**

```
cd ../
```

**Step 8: Run the following commands to create the Docker Image:**

```
sudo docker build -t hello1 .
sudo docker run -it -e VAR1='23' hello1
```

## Example 3: Simple Java Hellow World

**Step 1: Create a folder for your Docker Image and open this folder:**

hellojava/

**Step 2: Inside this folder create a Java file : HelloWorld.java with this content:**

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World");  
    }  
}
```

**Step 3: Inside this folder run the following command to compile this Class:**

javac HelloWorld.java

**Step 4: Inside this folder run the following command to Pull the Alpine Docker Image from the Docker Repository**

sudo docker pull alpine:latest

**Step 5: Inside this folder create a file called: Dockerfile with this content:**

```
FROM alpine:latest  
ADD HelloWorld.class HelloWorld.class  
RUN apk --update add openjdk8-jre  
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "HelloWorld"]
```

**Step 6: Inside this folder run the following commands to create the Docker Image:**

```
sudo docker pull alpine:latest  
sudo docker build --tag "docker-hello-world:latest" .
```

**Step 7: Inside this folder run the following command to launch the Docker Image:**

sudo docker run docker-hello-world:latest

## Example 4 : Hello World Apache Webserver

**Step 1: create a folder for your Docker Image and open this folder:**

helloweb/

**Step 2: Inside this folder create a HTML file : Index.html with this content:**

```
<html>
<body>
<h1>Hellow World</h1>
</body>
</html>
```

**Step 3: Inside this folder create a file called: Dockerfile with this content:**

```
FROM centos:latest
MAINTAINER NewstarCorporation
RUN yum -y install httpd
COPY index.html /var/www/html/
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
EXPOSE 80
```

**Step 4: Inside this folder run the following commands to create the Docker Image:**

```
sudo docker pull centos:latest
sudo docker build -t webserver:v1 .
```

**Step 5: Inside this folder run the following commands to launch the Docker Image using :**

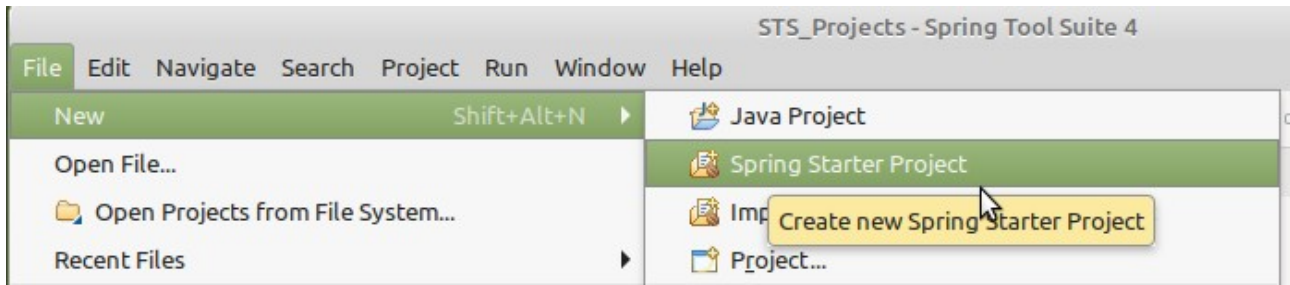
```
sudo docker run -dit -p 1234:80 webserver:v1
```

**Step 6: Open your browser to :**

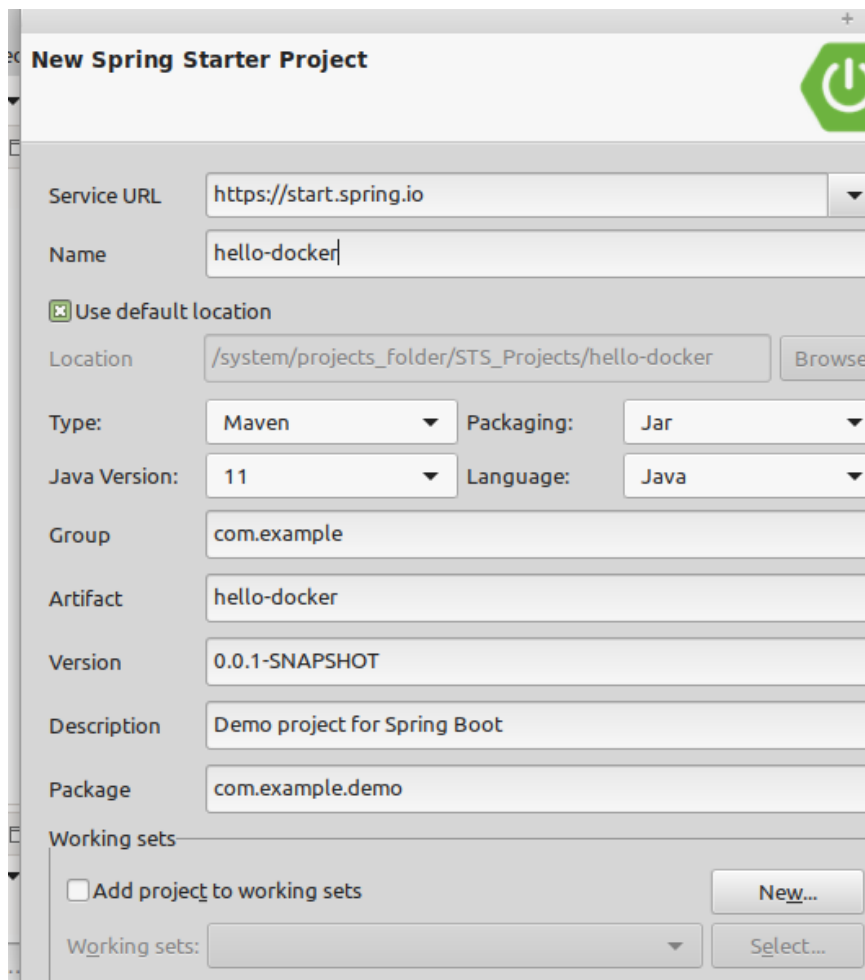
<http://localhost:1234>

## Example 5: Java Spring Boot Microservice

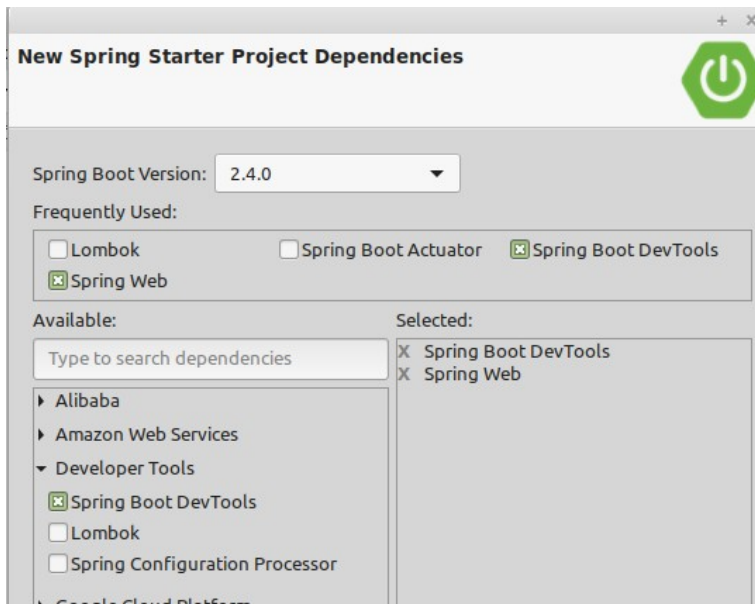
**Step 1 : create a Spring Boot project that will generate JAR file:  
hello-docker.jar**



**and name it hello-docker and press the Next button**



Select the Spring Boot version required  
Expand Web and select SpringWeb and expand  
DeveloperTools and select Spring Boot devTools  
**and press the Next button and then the Finish Button**



## Step 2 : Adding the Java code

Create a package : `com.example.demo.controllers`

In this package create a class `HelloDockerController` and use the following Java Code:

```
package com.example.demo.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloDockerController {

    @RequestMapping("/greet")
    public String getGreeting() {
        return "HelloDocker";
    }
}
```



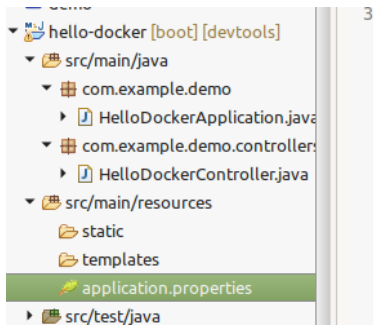
### Step 3 : Make the following changes to the POM file:

```
<dependencies>
  <!-- needed to fix issue with missing/invalid enum values -->
  <dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>2.2.4</version>
  </dependency>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <!-- <version>3.3.2</version> -->
  </dependency>
  <dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.11</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <finalName>${project.artifactId}</finalName>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

#### Step 4: In the resources folder edit the applications.properties file

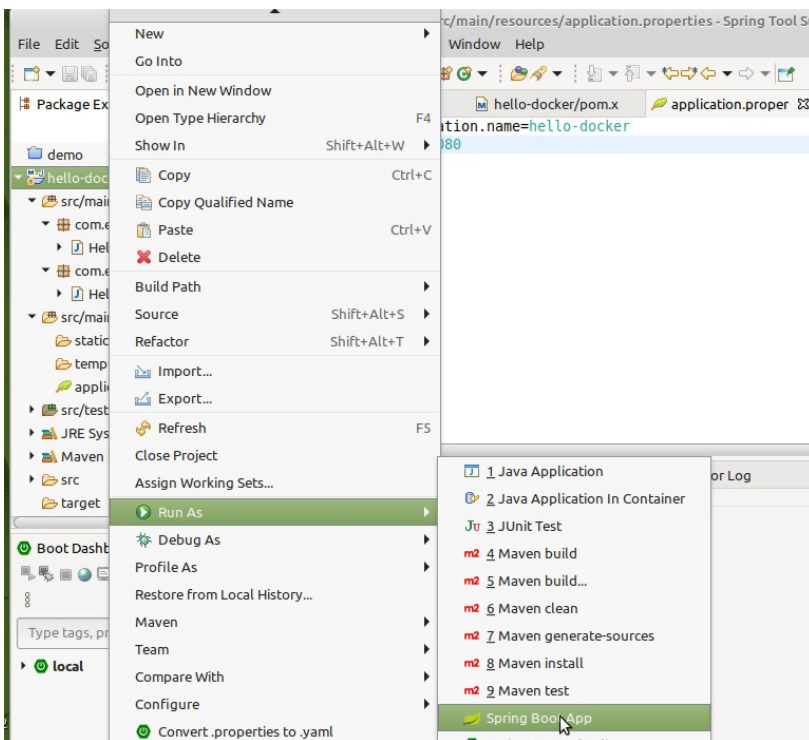


Now add this code there:

```
spring.application.name=hello-docker  
server.port=8080
```

#### Step 5: Test this project

Right click the project and select Run As and select Spring Boot App



Open the browser to this URL: <http://localhost:8080/greet>

You should see : HelloDocker

Stop the Server running in the IDE or shut down the IDE

#### Step 6 Open the folder where your IDE saved this project

Now run the following maven command in a terminal in that folder:

```
nicm@nm-laptop:/system/projects_folder/STS_Projects/hello-docker$ mvn clean package
```

You should find the following file in the target folder : hello-docker.jar

You will need to copy it in the Docker folder in the next steps

Test this jar file : `java -jar hello-docker.jar`

Open the Browser here : <http://localhost:8080/greet>

**Step 7: Building the Docker Project**

**Create a folder called:**

Hellodocker/

**Step 8: Inside this folder copy the JAR file from the previous step**

**Step 9: Inside this folder create a file called: Dockerfile with this content:**

```
FROM alpine:latest
WORKDIR /
VOLUME /tmp
ADD hello-docker.jar hello-docker.jar
RUN apk --update add openjdk11-jre
EXPOSE 8080
CMD java -jar hello-docker.jar
```

**Step 10: Inside this folder run the following commands to create the Docker Image:**

```
sudo docker pull alpine:latest
sudo docker build -t hello-docker:v1 .
```

**Step 11: Inside this folder run the following commands to launch the Docker Image using :**

```
sudo docker run -it -p 1234:8080 hello-docker:v1
```

**Step 12: Open your browser to :**

<http://localhost:1234>

You should now see :

HelloDocker

## 2 Portainer

Now we will look at docker container management commands and manage then manage them in Portainer

### **Install Portainer running these commands in a terminal:**

```
sudo docker volume create portainer_data
```

```
sudo docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce
```

**Open the browser with thhis URL:**

<http://localhost:9000>

**Use the Following Credentials:**

user:admin

password:adminadmin

## 3 Getting Familiar with docker commands

### Running an Ubuntu Container and accessing it from the terminal

`sudo docker run -it ubuntu bash`

### Now run these commands from the terminal

`ls -l`

`pwd`

`whoami`

`more /etc/os-release`

`lscpu`

`top`

`uname -a`

## Listing docker Images

`sudo docker images`

## Listing All Docker Containers

`sudo docker container ls -a`

## Starting a listed container

(assuming a container was listed as: 0a0631eeea21)

`sudo docker start -i 0a0631eeea21`

## Run and Delete a container afterwards

`sudo docker run --rm -t ubuntu bash`

## Deleting all Docker Containers

(its dangerous but it give you back all the disk space used)

`sudo docker container prune`

## **Pulling (down) Docker containers**

```
sudo docker pull ubuntu
sudo docker pull sonarqube
sudo docker pull postgres
sudo docker pull nginx
sudo docker pull ubuntu
sudo docker pull mariadb
sudo docker pull httpd
sudo docker pull centos
```

## **Show Details of Docker containers**

```
sudo docker container ls
```

## **List all Containers**

```
sudo docker container ls -l
sudo docker container ls -a
sudo docker container ls --all
```

## **List all Images**

```
sudo docker images
```

## **Stopping a container**

```
sudo docker stop ubuntu
```

## **Deleting a container**

```
sudo docker stop hello-world
sudo docker rm hello-world
sudo docker stop 724a4081c966
sudo docker rm 724a4081c966
```

## **Deleting an Image (This is permanent)**

```
sudo docker images
sudo docker rmi hello-world
```

## Running containers :

```
sudo docker run hello-world  
sudo docker run -it ubuntu bash
```

## Launch a container in the interactive mode

```
sudo docker run -i -t ubuntu:latest /bin/bash
```

## Attached Mode

In this mode, the console you are using to execute docker run will be attached to standard input, output and error.

That means your console is attached to the container's process.

## Detached Mode

In detached mode, you can follow the standard output of your docker container with docker logs -f <container\_ID>

## Running Containers in Attached mode:

```
docker run -rm mongo          ..... Mongo listens to port 27017  
Ctrl C to stop it
```

## Running Containers in Detached mode (as a service):

```
sudo docker run -d -p 80:80 gvelrajan/hello-world:v1.0  
sudo docker run -d -p 8080:80 gvelrajan/hello-world:v1.0  
sudo docker ps
```

## Run and then delete the container:

```
sudo docker run --rm -t ubuntu bash
```

## Running a webserver with internal port 80 and external port 4000 and 4001

```
sudo docker run --rm -it -p 4000:80 simple-webserver  
sudo docker run --rm -it -p 4001:80 simple-webserver
```

**You access the following webserver instances from the browser at:**

```
http://localhost:4000  
http://localhost:4001
```

# Pruning Docker Containers Images and volumes

## Pruning unused Docker Containers

**The following removes all containers**

```
sudo docker container prune
```

**The following removes containers created more than 5 minutes ago:**

```
sudo docker container prune --force --filter "until=5m"
```

## Prune unused Docker Images

```
sudo docker image prune
```

**Limiting images pruned to only images created more than 24 hours ago:**

```
sudo docker image prune -a --filter "until=24h"
```

## Prune Volumes (this is non recoverable)

```
sudo docker volume prune
```

**Removing volumes which are not labelled with the keep label:**

```
sudo docker volume prune --filter "label!=keep"
```



## 4 The Dockerfile Syntax

FROM centos // you are specify "Super - Container" usually the O/S web server or Database  
MAINTAINER <author's detail>  
COPY <src> ... <dst> // to copy files into the Image  
ADD <src> ... <dst> // similar to add but handles tar commands  
ENV <key> <value>  
ENV DEBUG\_LVL 3  
ENV APACHE\_LOG\_DIR /var/log/apache  
USER <UID>|<UName> // <UID> numerical user ID <UName> :valid user Name  
WORKDIR <dirpath>  
VOLUME <mountpoint>  
EXPOSE <port>[/<proto>] [<port>/<proto>]...]  
EXPOSE 7373/udp 8080  
RUN <command> // to run a command in the Image  
CMD <command> // execute a shell command during launch of the container  
ENTRYPOINT <command> // shell command executed during the launch of the container.  
ONBUILD <INSTRUCTION>  
ONBUILD ADD config /etc/appconfig

### And Example of a Dockerfile :

```
#####  
# Dockerfile to build an Apache2 image  
#####  
# Base image is Ubuntu  
FROM ubuntu:14.04  
# Author: James Bond  
MAINTAINER James Bond <james.bond007@gmail.com>  
# Install apache2 package  
RUN apt-get update && \  
apt-get install -y apache2 && \  
apt-get clean
```

### Running Containers with a Hard coded port

```
sudo docker container run --name mongo mongo -p 8081:271017 mongo  
sudo docker container rm -f mongo .....manually deletes the container
```

### Running Containers with Dynamically allocated port

```
sudo docker container run -d --name mongo -p 271017 mongo
```

### Pausing/Unpausing a Docker Container

```
sudo docker run -i -t ubuntu:latest /bin/bash
```

## 5 Docker Swarms

### Difference between Docker and Docker Swarm

As a platform, Docker has revolutionized the manner software was packaged.

Docker Swarm or simply Swarm is an open-source container orchestration platform and is the native clustering engine for and by Docker.

Swarm turns a pool of Docker hosts into a virtual, single host.

### Worker Nodes

Worker nodes are also instances of Docker Engine whose sole purpose is to execute containers.

### Create Swarm

`docker swarm init` ....do this on the first node to manage the swarm

### Joining a swarm

```
docker swarm join --token SWMTKN-1-
4zfe8c7vezompmxgnvr59z9fzj4dop9lggde6qk4qs71nytr97-03cjgmeh8yiun807y7adz6k79
10.154.2.105:2377
```

### Creating workers

`docker swarm join-token worker`

.. it will display :

To add a worker to this swarm, run the following command:

```
sudo docker swarm join --token SWMTKN-1-
4zfe8c7vezompmxgnvr59z9fzj4dop9lggde6qk4qs71nytr97-2srtr4qxcnvc9bqmr9nn6s76z
10.154.2.105:2377
```

### Joining another node as a worker to this swarm

```
sudo docker swarm join --token SWMTKN-1-
4zfe8c7vezompmxgnvr59z9fzj4dop9lggde6qk4qs71nytr97-2srtr4qxcnvc9bqmr9nn6s76z
10.154.2.105:2377
```