1. How to connect application with two different DB:-

Step 1:- create two different configuration in application.xml file,

1st DB configuration:-

Database1.datasource.url=

Database1.datasource.username=

Database1.datasource.password=

2nd DB Configuration:-

Databse2.datasource.url=

Database2.datasource.username=

Database2.datasource.password=

Step 2:- Configure the Data source for both the Db connections:-

@Bean(name=”datasource1”)

@ConfigurationProperties(“database1.datasource”)

@Primary

Public DataSource dataSource(){

Return DataSourceBuilder().create().build();

}

@Bean(name=”datasource2”)

@ConfigurationProperties(“database2.datasource”)

Public DataSource dataSource2(){

Return DataSourceBuilder().craete().build();

}

Step 3:- how we can make the Db operations Atomic,

We need to do the configuration for TranasactionManager in the same class,

For 1st Db:-

@Bean(name=”t1”)

@autowired

Public DataSourceTransactionManager(@Qualifier(“datasource1”) DataSource dataSource){

}

For 2nd Db:-

@Bean(name=”t2”)

@autowired

Public DataSourceTransactionManager(@Qualafier(“dataSource”) DataSource dataSource){

}

If I will inject @Transaction annotation in this case by default DB1 will work.

@Transactional(“t2”) -🡪 Db2 will work.

If I want to use CURD operations with in the same method, I could use @ChainTransactionalManager

@Transactional(“chainTransaction”)

Db.update();

Db.insert();

Questions 2:- how we can enable application running if Db is down as well,

First we need to understand if database down is impact on the the client business if not much like offline outlook , means here we are not giving so much importance to connections related exceptions and application is up and running in offline mode as well.

If we have spring boot application in that case we could go with below configurations and mainly on

continueOnerror : true

use\_jdbc\_metadata\_defaults: false

ddl-auto: false

But if download Db is impacting on client business so in that case we could either go with staging server which will be available in case of down time.

We can change the spring boot application.properties file like below,

spring:

datasource:

driverClassName: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost:3306/schema

username: root

password: root

continueOnError: true

initialize: false

initialSize: 0

timeBetweenEvictionRunsMillis: 5000

minEvictableIdleTimeMillis: 5000

minIdle: 0

jpa:

show-sql: true

hibernate:

ddl-auto: none

naming\_strategy: org.hibernate.cfg.DefaultNamingStrategy

properties:

hibernate:

dialect: org.hibernate.dialect.MySQL5Dialect

hbm2ddl:

auto: none

temp:

use\_jdbc\_metadata\_defaults: false

Question 3:- what is load balancer?

Load balancer is the process to distribute the traffic among different instances of same application. To create the fault tolerant application it is common to run multiple instances of each application.

Whenever service need to communicate to each other its need to pick an instance to send the request.

Random selection

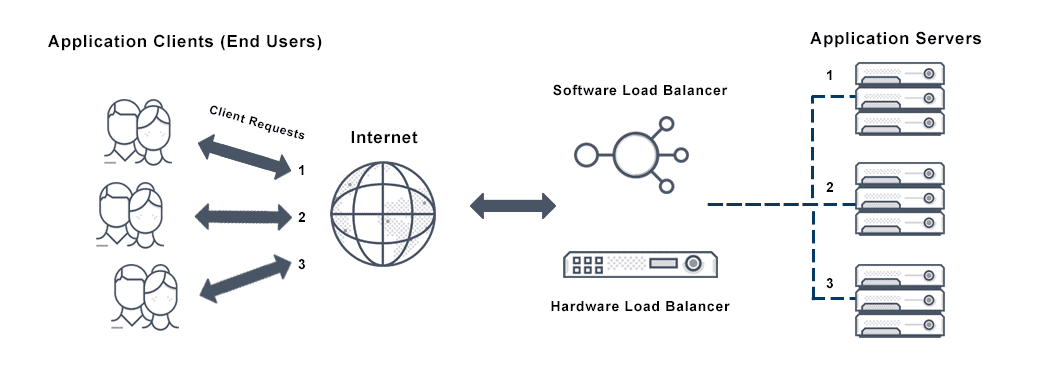
Round-Robin algorithm

Least Connection algorithm

IP Hash.

1. Random Selection:-
   1. Pick a random server and assign the request to random server.
   2. Its lightweight and no need to calculate the existing load on server and no need time management also.
2. Round Robin:-

**Round robin load balancing** is a simple way to distribute client requests across a group of servers. A client request is forwarded to each server in turn. The algorithm instructs the [load balancer](https://avinetworks.wpengine.com/software-load-balancer/) to go back to the top of the list and repeats again.



round robin network load balancing rotates connection requests among web servers in the order that requests are received. For a simplified example, assume that an enterprise has a cluster of three servers: Server A, Server B, and Server C.

• The first request is sent to Server A.  
• The second request is sent to Server B.  
• The third request is sent to Server C.

The load balancer continues passing requests to servers based on this order. This ensures that the server load is distributed evenly to handle high traffic.

The biggest drawback of using the round robin algorithm in load balancing is that the algorithm assumes that servers are similar enough to handle equivalent loads. If certain servers have more CPU, RAM, or other specifications, the algorithm has no way to distribute more requests to these servers. As a result, servers with less capacity may overload and fail more quickly while capacity on other servers lie idle.

The weighted round robin load balancing algorithm allows site administrators to assign weights to each server based on criteria like traffic-handling capacity. Servers with higher weights receive a higher proportion of client requests. For a simplified example, assume that an enterprise has a cluster of three servers:

• Server A can handle 15 requests per second, on average  
• Server B can handle 10 requests per second, on average  
• Server C can handle 5 requests per second, on average

Next, assume that the load balancer receives 6 requests.

• 3 requests are sent to Server A  
• 2 requests are sent to Server B  
• 1 request is sent to Server C.

In this manner, the weighted round robin algorithm distributes the load according to each server’s capacity.

1. Caching implementation:-
2. Caching refer the process to store the file at temporary storage location called cache which helps to access the data quickly and reducing the latency of application.
3. The cache can only store a limited amount of data. Due to this, it is important to determine cache update strategies that are best suited for the business requirements. Following are the various caching strategies available:

* **Cache-aside**: In this strategy, our application is responsible to write and read data from the storage. Cache interaction with the storage is not direct. Here, the application looks for an entry in the cache, if the result is not found, then the entry is fetched from the database and is added to the cache for further use. Memcached is an example of using this update strategy.

Cache-aside strategy is also known as lazy loading because only the requested entry will be cached thereby avoiding unnecessary caching of the data. Some of the disadvantages of this strategy are:

* In cases of a cache miss, there would be a noticeable delay as it results in fetching data from the database and then caching it.
* The chances of data being stale are more if it is updated in the database. This can be reduced by defining the time-to-live parameter which forces an update of the cache entry.
* When a cache node fails, it will be replaced by a new, empty node which results in increased latency.
* **Write-through**: In this strategy, the cache will be considered as the main data store by the system and the system reads and writes data into it. The cache then updates the database accordingly as shown in the database.
* The system adds or updates the entry in the cache.
* The cache synchronously writes entries to the database. This strategy is overall a slow operation because of the synchronous write operation. However, the subsequent reads of the recently written data will be very fast. This strategy also ensures that the cache is not stale. But, there are chances that the data written in the cache might never be read. This issue can be reduced by providing appropriate TTL.
* **Write-behind (write-back)**: In this strategy, the application does the following steps:
  + Add or update an entry in the cache
  + Write the entry into the data store asynchronously for improving the write performance. This is demonstrated in the image below:
* The main disadvantage of this method is that there are chances of data loss if the cache goes down before the contents of the cache are written into the database.

Examples:- L1,L2,L3 cache , CPU chache, Memcached, Hibernate Chache(first level and second level)

Code implementation:-

LRU :- least recently used.

For this we could use Queue along with HashMap .

* Create a class LRUCache with declare a list of type int, an unordered map of type <int, list<int>>, and a variable to store the maximum size of the cache
* In the refer function of LRUCache
  + If this value is not present in the queue then push this value in front of the queue and remove the last value if the queue is full
  + If the value is already present then remove it from the queue and push it in the front of the queue
* In the display function print, the LRUCache using the queue starting from the front

ACID properties:-

* + 1. Atomicity
    2. Consistency
    3. Isolation
    4. Durability
* Atomicity defines that the data remains atomic, it means if any operation performed it should not be partially , means either it should execute fully or should not be executed.
* Consistency means data should remain consistence before and after operations.
* The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained.
* Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives.

# [How can I change server in spring boot application?](https://stackoverflow.com/questions/63054114/how-can-i-change-server-in-spring-boot-application)

# You have to exclude tomcat from starter dependency:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

and now you need to include new server as a dependency i.e.:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

# Change the port for server:-

# Change in application.properties file like blow,

# Server.port=

# Spring Interceptor are used to intercept client requests and process them. Sometimes we want to intercept the HTTP Request and do some processing before handing it over to the controller handler methods.

# PreHandle();

# postHandle();

# afterCompletion();

### **What are Profiles in Spring Boot?**

Profiles in the Spring framework enables users to map components and beans to specific profiles, such as the Development (dev) profile, Production (prod) profile, or the Test profile.

In Spring Boot, the annotation @Profile is used to map components and beans to a certain profile.

Developers can also set up profiles using the SpringApplication, for instance, SpringApplication.setAdditionalProfiles("dev");

### **What differentiates Spring Data JPA and Hibernate?**

A Java Persistence API (JPA) implementation, Hibernate facilitates Object-Relational Mapping (ORM), allowing users to store, retrieve, map, and update application data to and from Java objects and relational databases. Hibernate maps the data types in Java to SQL (Structured Query Language) data types, and the classes in java to the database tables, relieving developers from scripting data persistence SQL programs.

A Spring Data sub-project, Spring Data JPA, on the other hand, gives abstraction over the DAL (Data Access Layer) applying JPA and Object–Relational Mapping implementations, such as Hibernate. Spring Data JPA facilitates the smooth implementation of JPA repositories, and it intends to improve the overall implementation of DAL to a great extent.

# Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom **init()** method and the **destroy()** method.

# To provide the facility to the created bean to invoke custom **init()** method on the startup of a spring container and to invoke the custom **destroy()** method on closing the container, we need annotate **init()** method by **@PostConstruct** annotation and **destroy()** method by **@PreDestroy** annotation. **Note:** To invoke the **destroy()** method we have to call the **close()** method of ConfigurableApplicationContext.

|  |  |
| --- | --- |
| **Sr.No.** | **Scope & Description** |
| 1 | **singleton**  This scopes the bean definition to a single instance per Spring IoC container (default). |
| 2 | **prototype**  This scopes a single bean definition to have any number of object instances. |
| 3 | **request**  This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext. |
| 4 | **session**  This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |
| 5 | **global-session**  This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |

# Bydefault scope of bean is singleton.

Can we inject **prototype** inside singleton if yes how?

Using @lookup annotation.--> method injection.spring override it dynamically.

By suing applicationContext.getBean() methos inside getMethod();-🡪 it is directly calling the bean from container but we are over the IOC, so should not use this .

By using Objectfactory as well -🡪 objectFactory.get().getmethod();

@Primary , @Qualifier, @Required:-

Interface Animal, having two Dog and Cat implementing classes,

@service

Class Dog implement Animal{

}

@service

Class Cat implement Animal{

}

Class Controller {

@autowired

Private Animal animal;

}

It will fail -🡪 because we are binding two beans

Solution:- use @qualifier

@qualifier(“cat”)

@autowired

Private Animal animal;

Another solution:-

@primary annotate any one class

Class Dog implements Animal{

}

* CrudRepository provides CRUD functions.
* PagingAndSortingRepository provides methods to do pagination and sort records.
* JpaRepository provides JPA related methods such as flushing the persistence context and delete records in a batch.

JpaRepository implements both the repo.

**Java 8 interview questions:-**

ou made a good summary of the practical differences in use and implementation but did not say anything about the difference in meaning.

An **interface** is a description of the behaviour an implementing class will have. The implementing class ensures, that it will have these methods that can be used on it. It is basically a contract or a promise the class has to make.

An **abstract class** is a basis for different subclasses that share behaviour which does not need to be repeatedly created. Subclasses must complete the behaviour and have the option to override predefined behaviour (as long as it is not defined as final or private).

You will find good examples in the java.util package which includes interfaces like List and abstract classes like AbstractList which already implements the interface. The [official documentation](http://docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html) describes the AbstractList as follows:

This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array).

**5,204**33 gold badges2323 silver badges3333 bronze badges

* 23

This should be the answer. Not a list of details, but the underlying concept that make the difference between an interface and an abstract class, not only in Java but in general.

This is really Good. Of course others answers are good too. But this tells you a major heads-up about abstract keyword, that is when a compiler sees this, they know, **the following information is *incomplete* and needs implementation**. Interfaces are always incomplete, but abstract classes are abstract because they had to have incomplete (abstract) methods.

Solid principle:-

S🡪 single responsibility principle

o🡪 open/close principle

l🡪 liskov substitute principle

i🡪 interface segregation principle

d🡪 dependency inversion principle

Single responsibility principle🡪 class should have single responsibility, means it should has only one resion to change the class.

Example:-

class Invoice{

public void print(){

}

public void counThePages(){

}

}

Advantages:-

East to maintain

Easy to understand

Open close principle:- open for extension but close for modification.

Class Account{

Public void paymentType(){

}

// and this class is active in production or it is live

//but tomorrow you want to add one more method in this like sattelment() , that should not allowed according to this principle

}

liskov substitute principle : - if class A have subclass B , in that case we should able to replace object of A with object of B with out changing the behavior of class.

interface segregation principle 🡪 client should not implement unnassary methods in there classes.

dependency inversion principle 🡪 we should dependent on inetrfaces instead of concrete classes.

Microservices deplohyment with docker:-

Commands:-

docker run -p 5000:5000 in28min/todo-rest-api-h2:1.0.RELEASE

if you want to always running put –d,

docker run -p 5000:5000 –d in28min/todo-rest-api-h2:1.0.RELEASE

image 🡪 static version, or could say a template

Container -> running version.

Bridge network-🡪 also called docker internal network no one can access it until unless you are not exposing it on to the host, where your container is running.

-p hostport: containerport

docker run -p 5000:5000 –d in28min/todo-rest-api-h2:1.0.RELEASE

Docker logs containerid

Docker container ls

Docker container ls –a

Docker container stop containerId

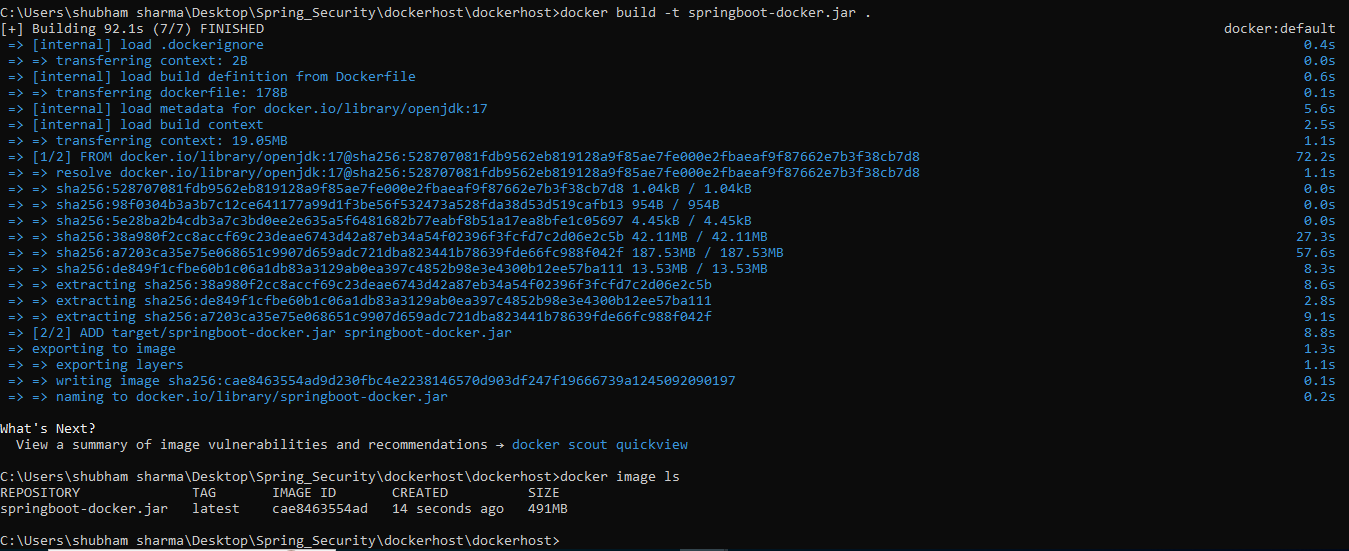
Docker images

Process to host on docker:-

1.Create an image

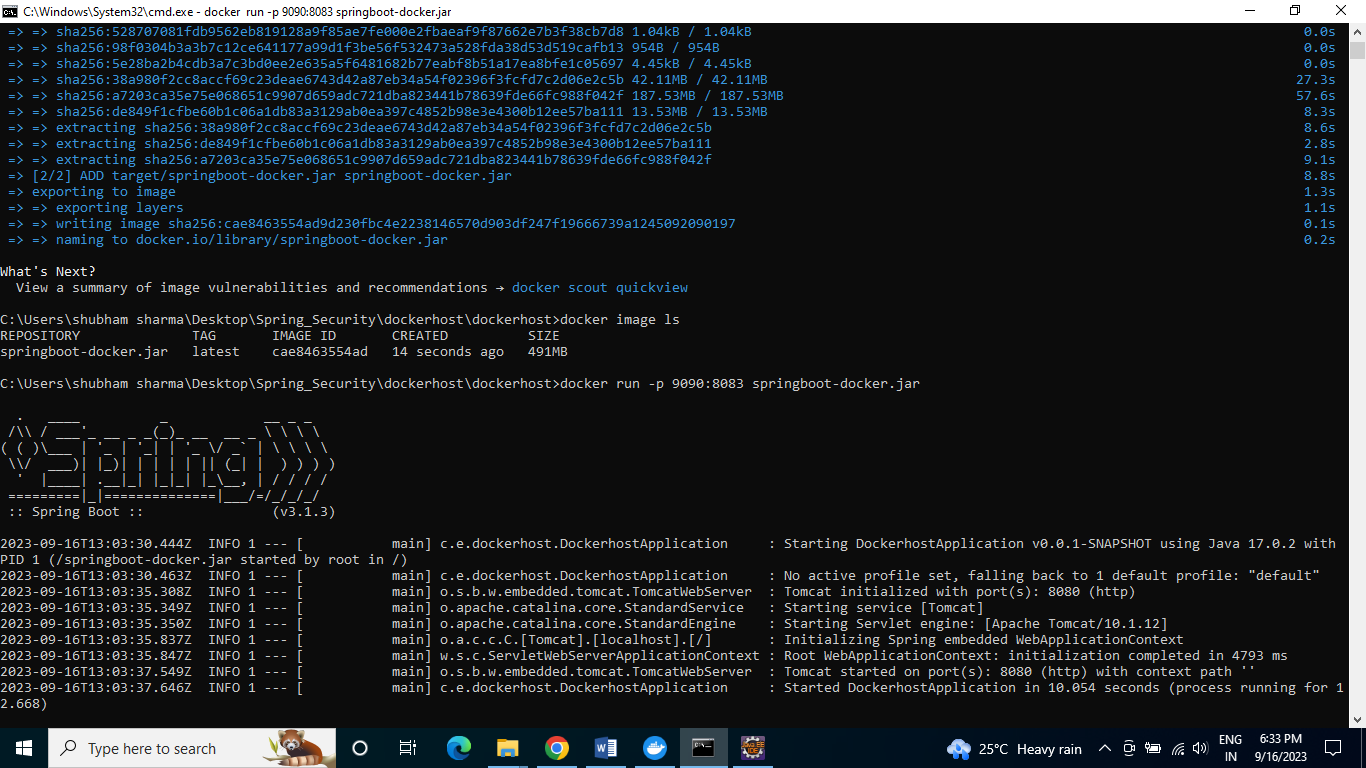
Build the image:-

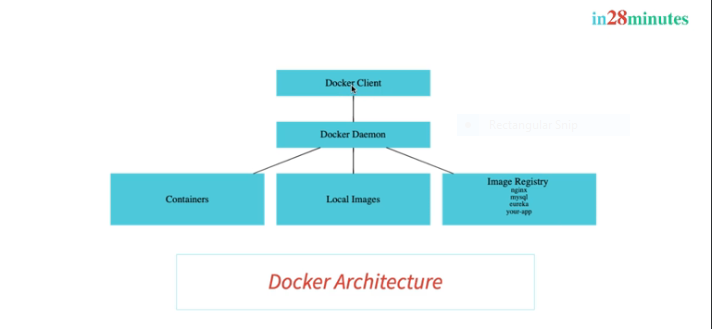
docker build -t springboot-docker.jar .



Run the container:-

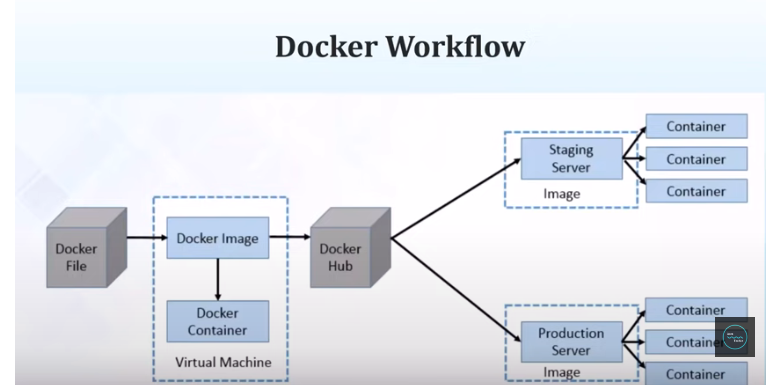
docker run -p 9090:8083 springboot-docker.jar





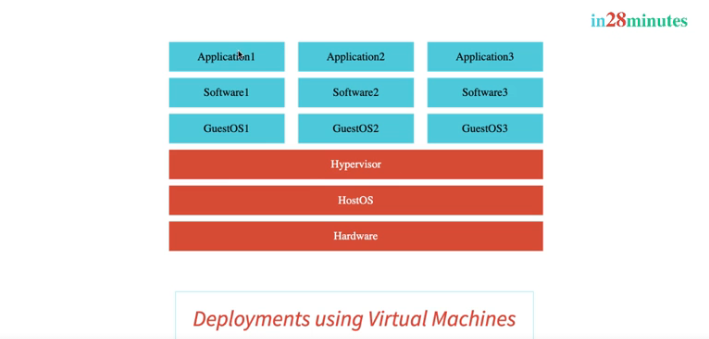
Run the commands in docker client -🡪 send to docker deomen , will check the image locally if not download from the ihub repo.

Docker Daemon🡪 creating the images , pulling the images from image registry as well.



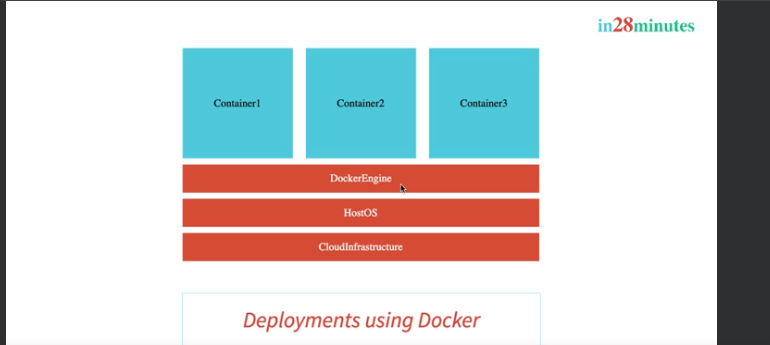
Why docker is very famous🡪 we can install docker on cloud and make it use.

Before Docker VM was very famous,



Problem with these VM are these are very heavy , we have 2 OS one is HostOS and another one GuestOS

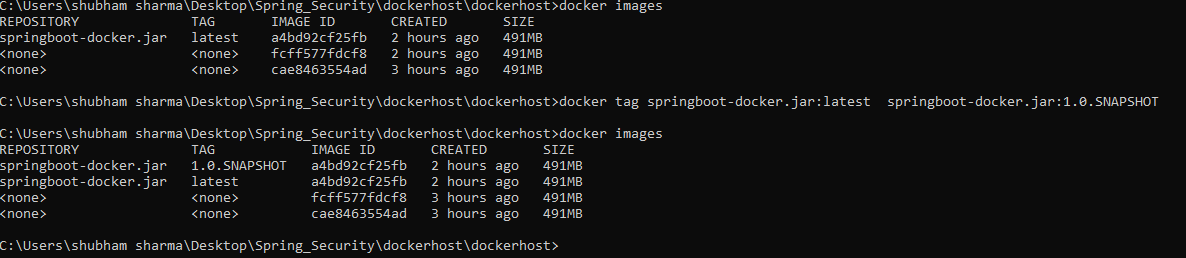
So here docker come in picture,



Commands for images :-

Want to change the tags,

docker tag springboot-docker.jar:latest springboot-docker.jar:1.0.SNAPSHOT



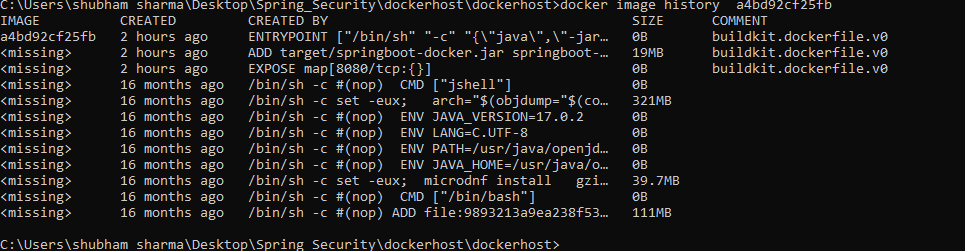
imageID will be same for both the image.

If we want to pull some images from ,

docker pull mysql

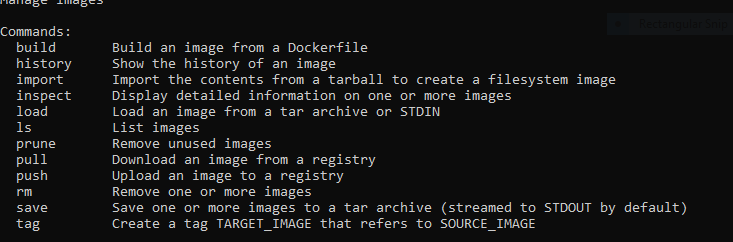
for search the image,

docker search mysql



docker image inspact a4bd92cf25fb

commands with images:-



Single image can have multiple tags.

Official version also we have for image.

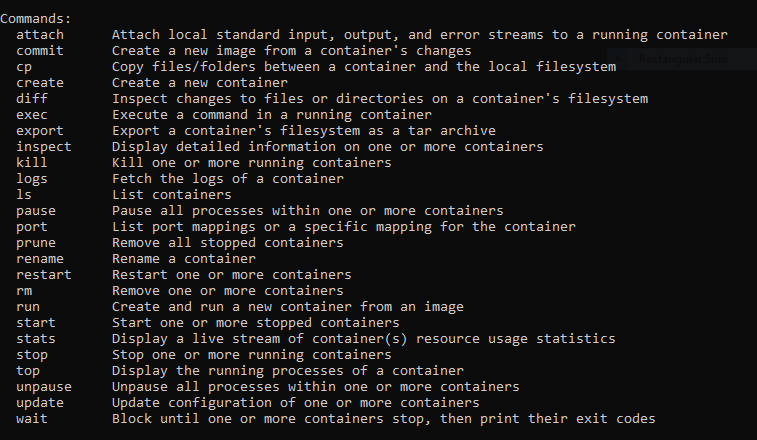
Docker image remove image\_id.

Containers commands:-

docker container pause 09b65243c9ef



docker container unpause 09b65243c9ef



docker container inspect 09b65243c9ef

to remove all the stop containers:-

docker container prune

docker container ls –a

docker container stop 09b65243c9ef

SIGDOWN

grace fully shutdown.

docker container kill 09b65243c9ef

will not shutdown gracefully.it will stop immidietly.

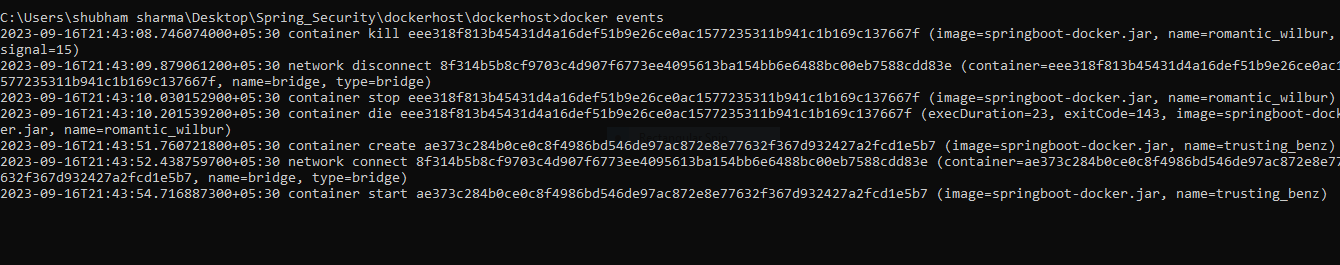
Restart policy:-

Docker run –p 5000:5000 –d –restart=always springboot.jar

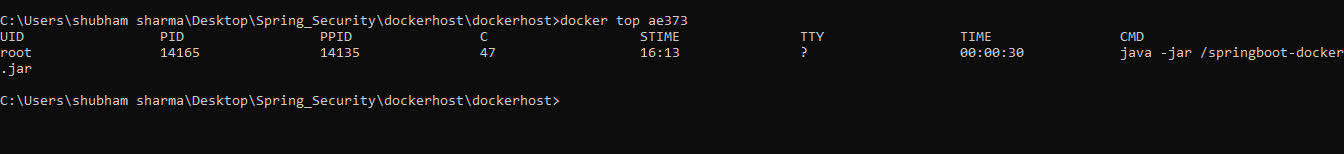
When ever you will restart your docker desktop it will automatically start running.

Docker run –p 5000:5000 –d –restart=no springboot.jar

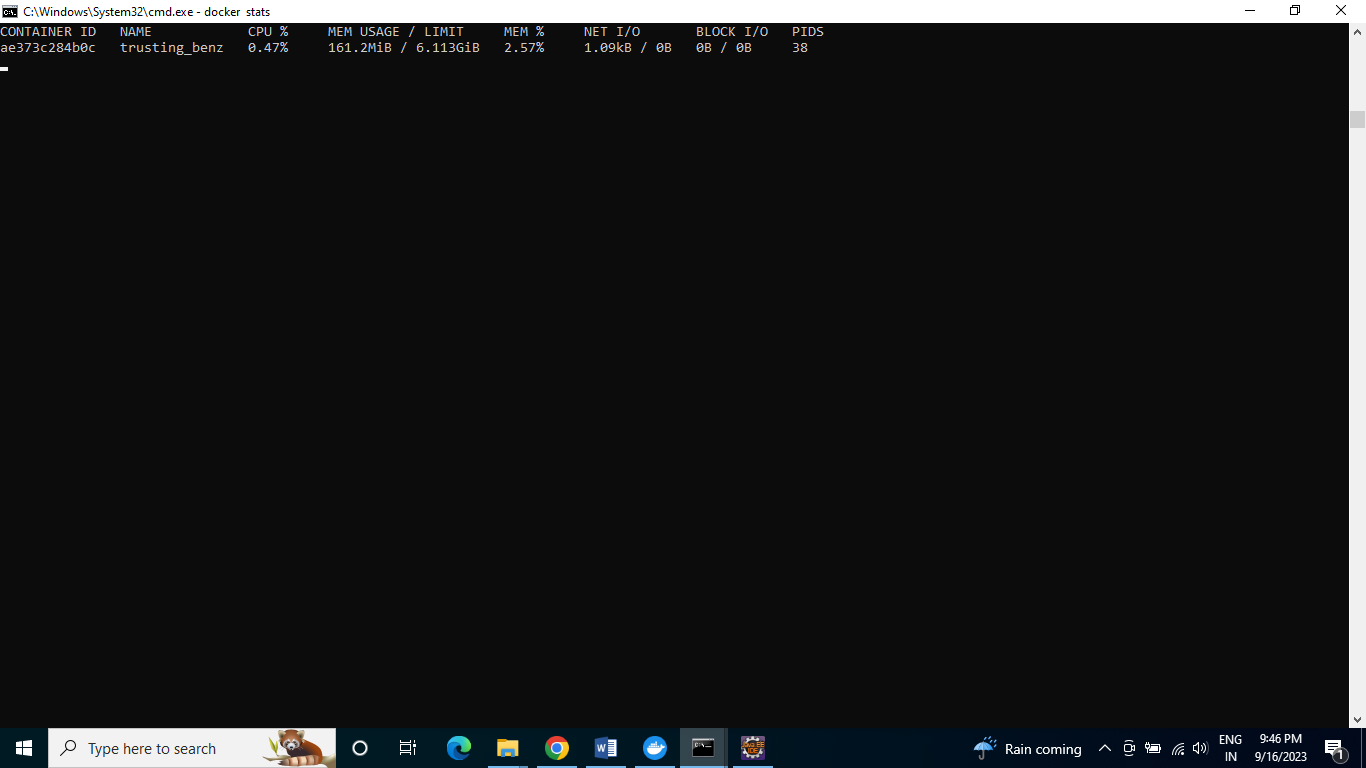
Docker events



Docker top command to see the process inside the container,



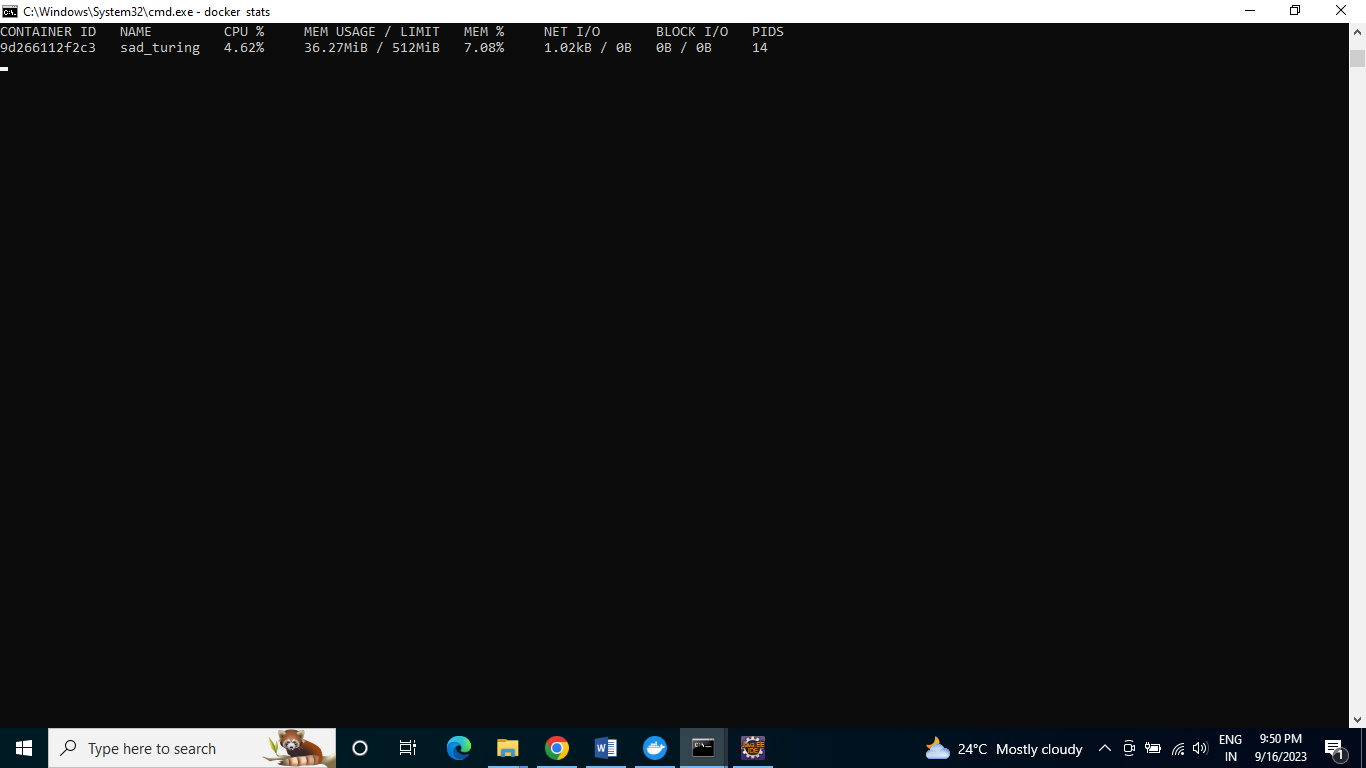
Docker stats



docker run -p 5000:8080 -m 512m --cpu-quota 5000 springboot-docker.jar

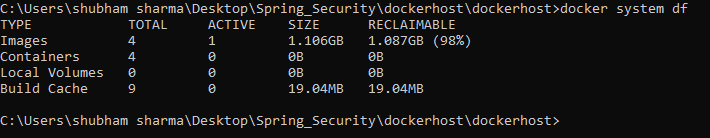
with memory and cpu quota.

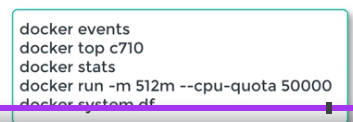
Docker stats

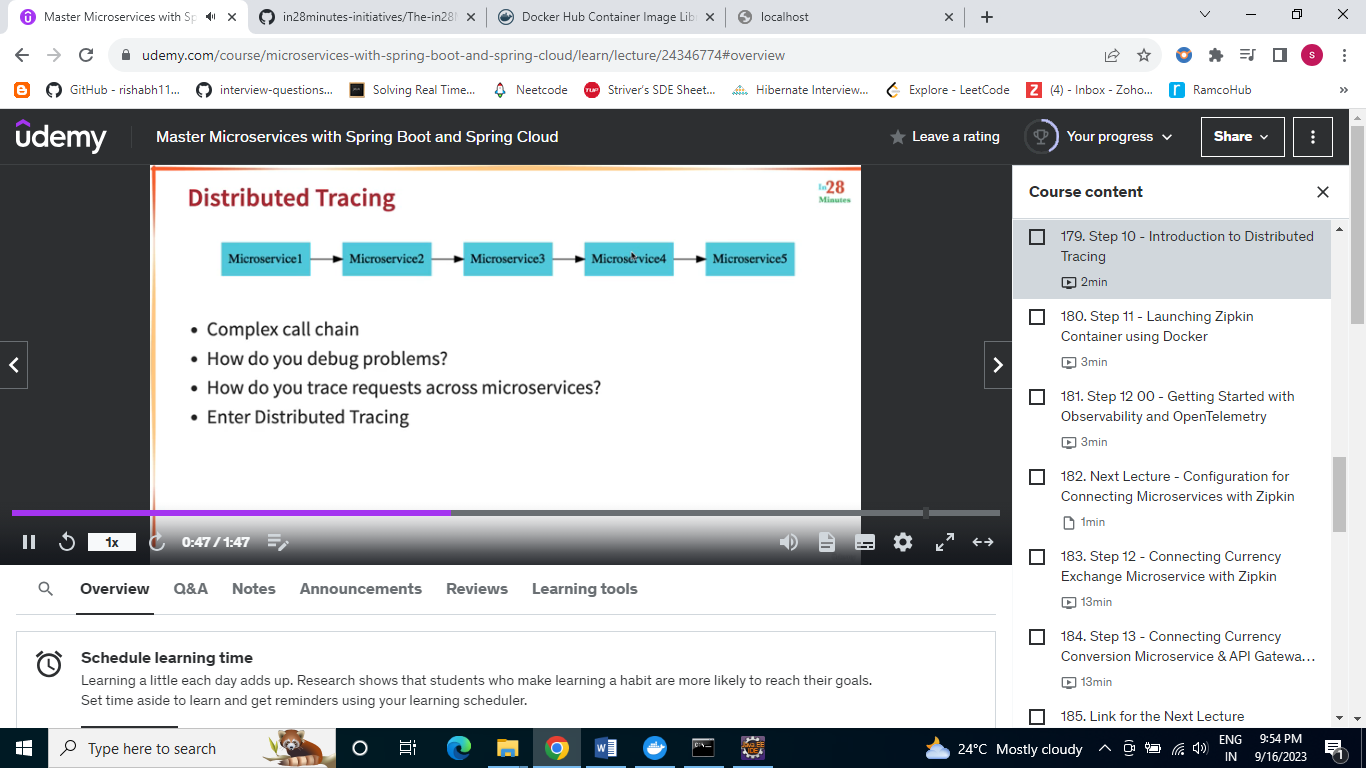


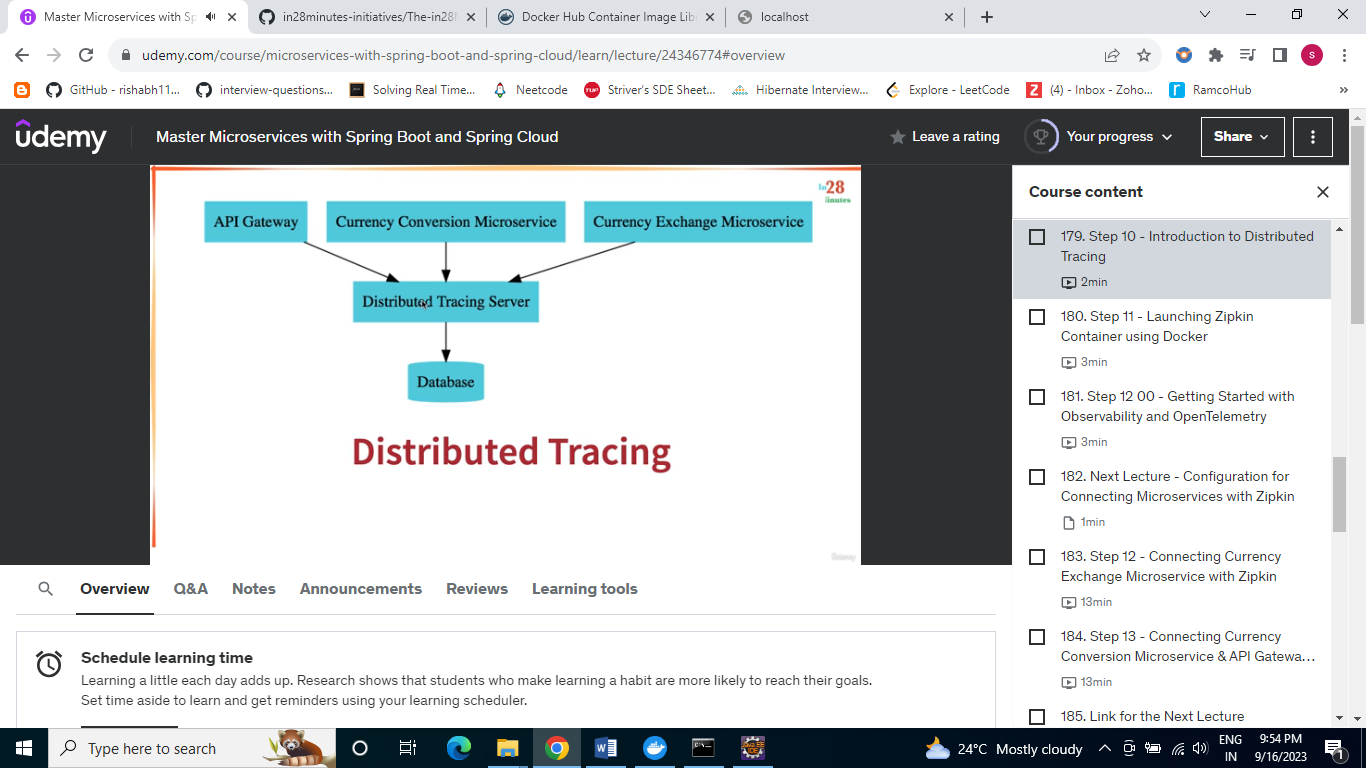
Docker system df

The work done by daemon managing:-

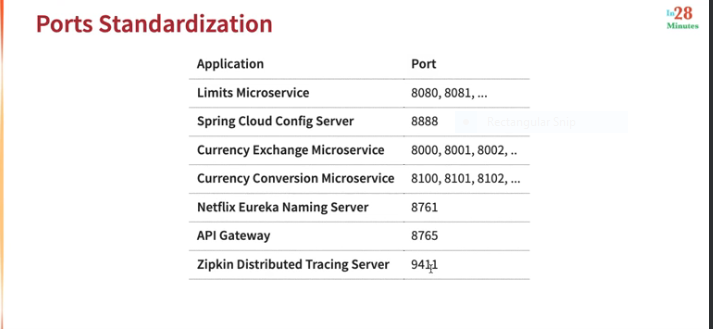








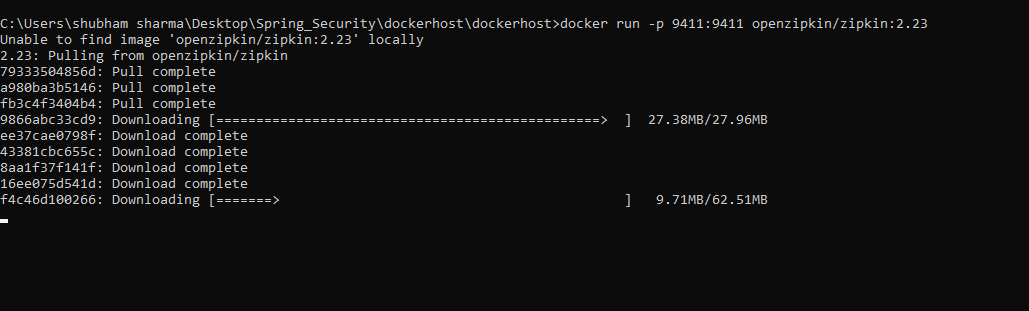
Distributed tracing server will provide the information for every request is coming --🡪 zipkin is an example of Distributed tracing server.

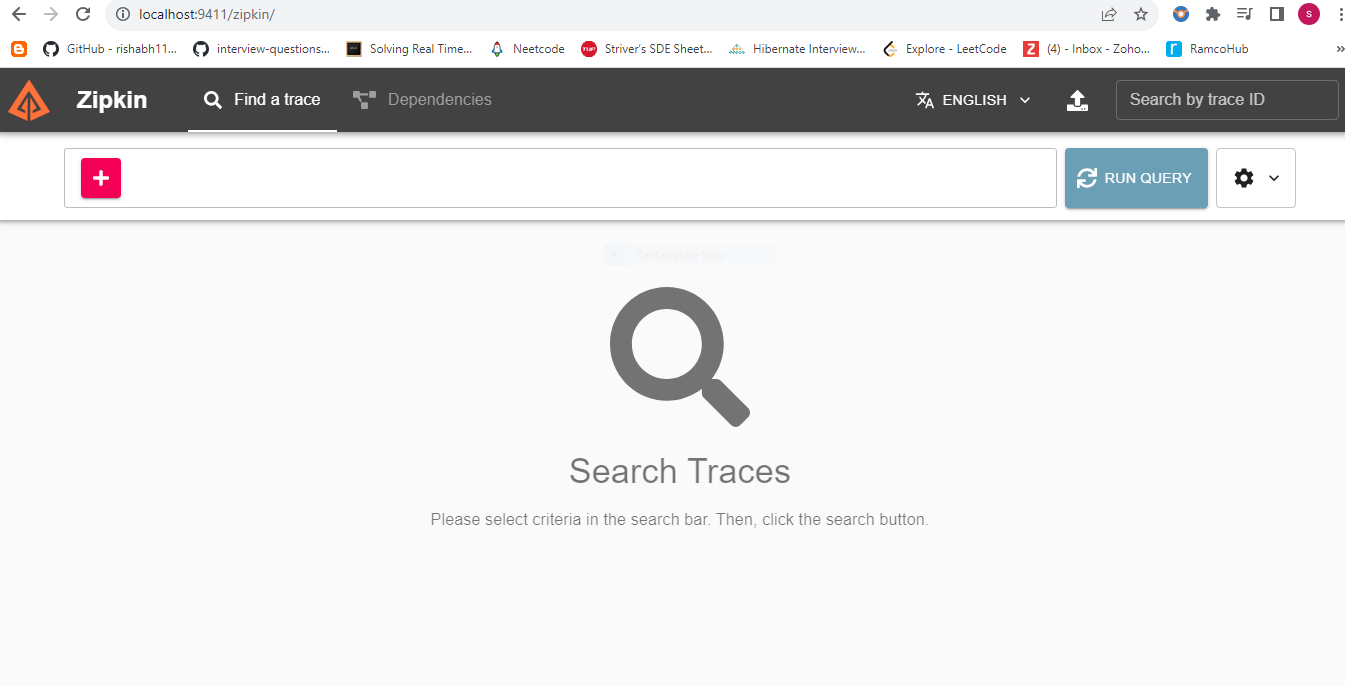


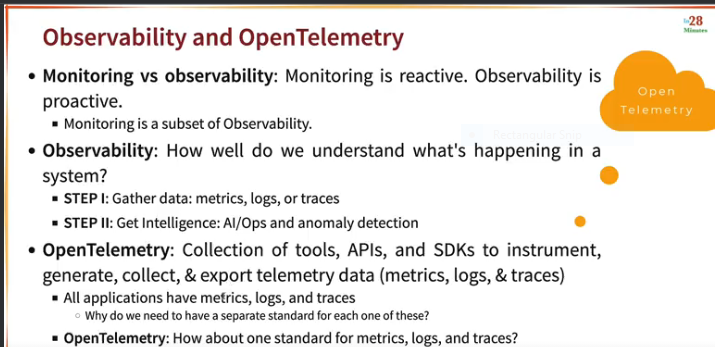
Launch zipkin on docker follow below commands,

Launching zipkin docker container

docker run -p 9411:9411 openzipkin/zipkin:2.23







In spring boot 2-🡪

Sleuth(Tracing configuration) -🡪 Brave(Tracer Lib) -🡪 zipkin

Sleuth can only handel tracers

In Spring boot 3🡪

Micrometer(logs,metrics and traces)-🡪 openTelemetry(metrics,logs & traces)-🡪 zipkin

<!-- SB3 : Micrometer

> OpenTelemetry

> Zipkin

-->

<!-- Micrometer - Vendor-neutral application observability facade.

Instrument your JVM-based application code without vendor lock-in.

Observation (Metrics & Logs) + Tracing.

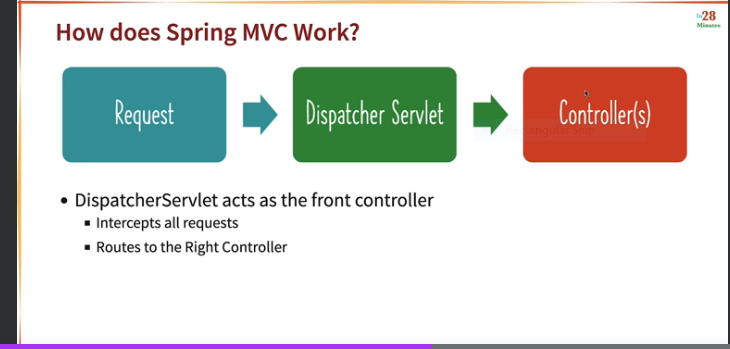
-->

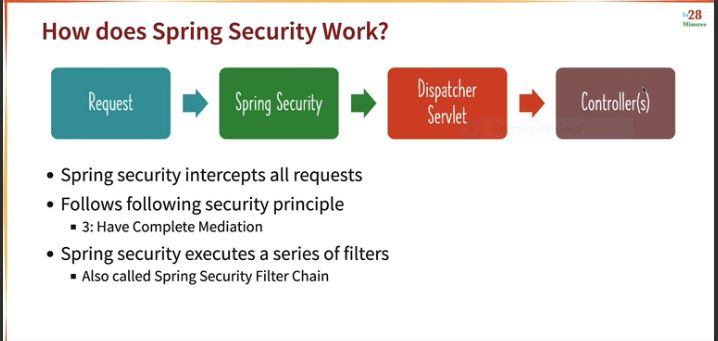
We have concept like sampling , because you want to trace all the information related to request it may impact to the application performance.

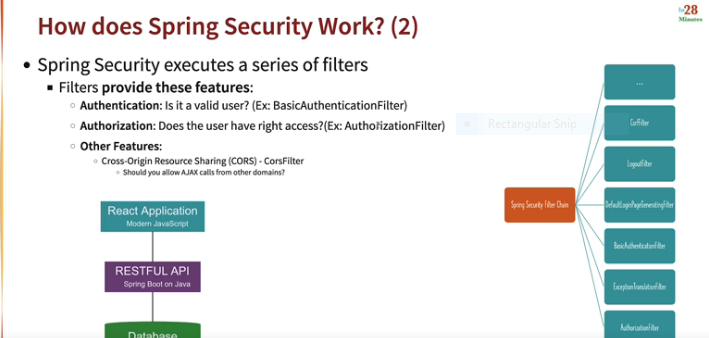
So we can make use of sampling which will tell like how much % you want to trace.

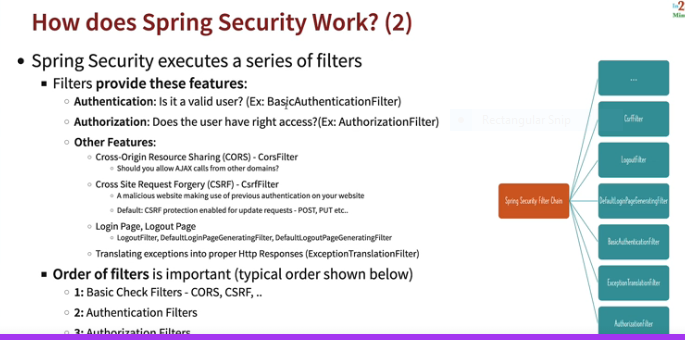
Spring Security:-

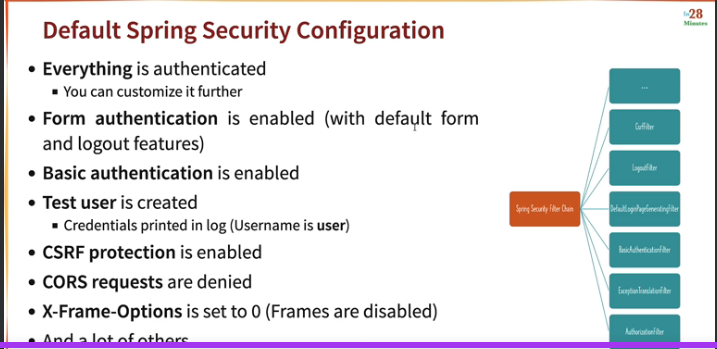
Spring Security :-

* + 1. Filter Chain.
    2. Authentication manager.
    3. Authentication Provider.
* 

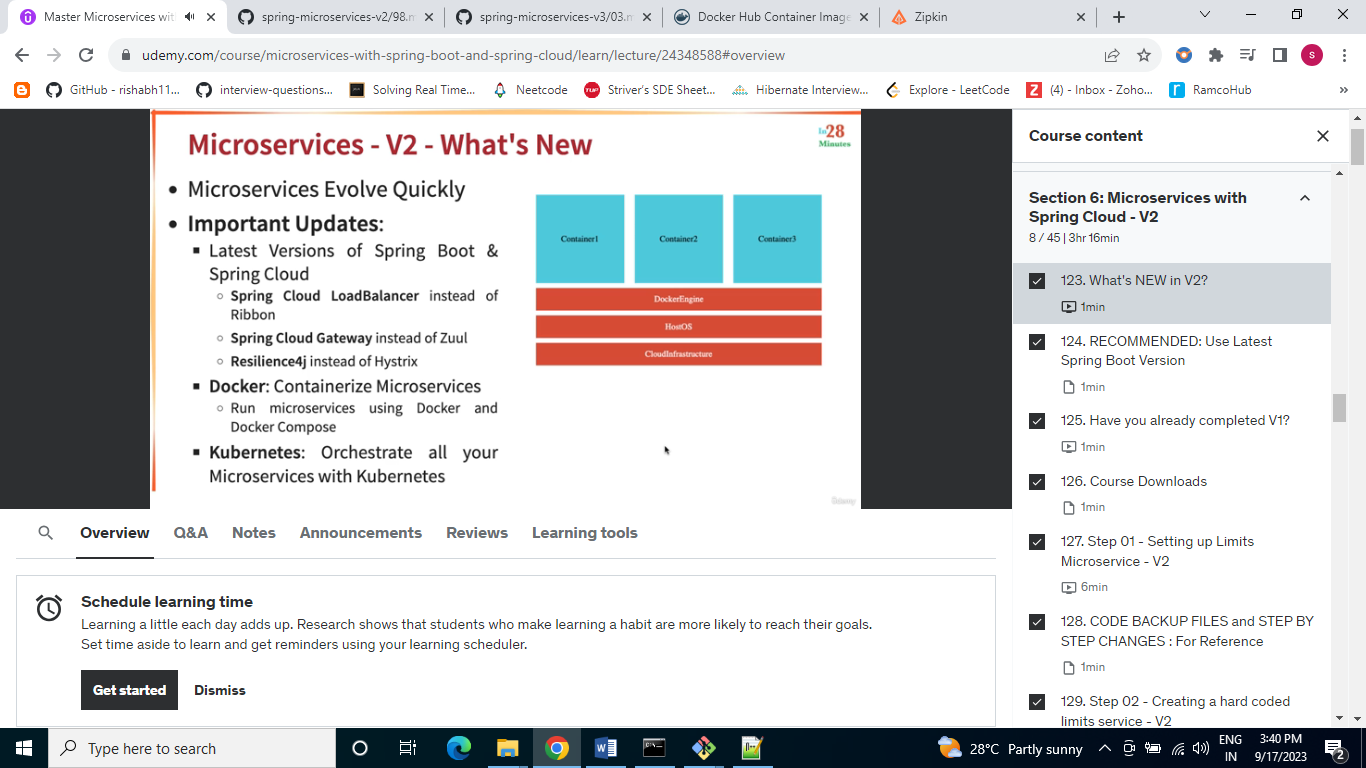


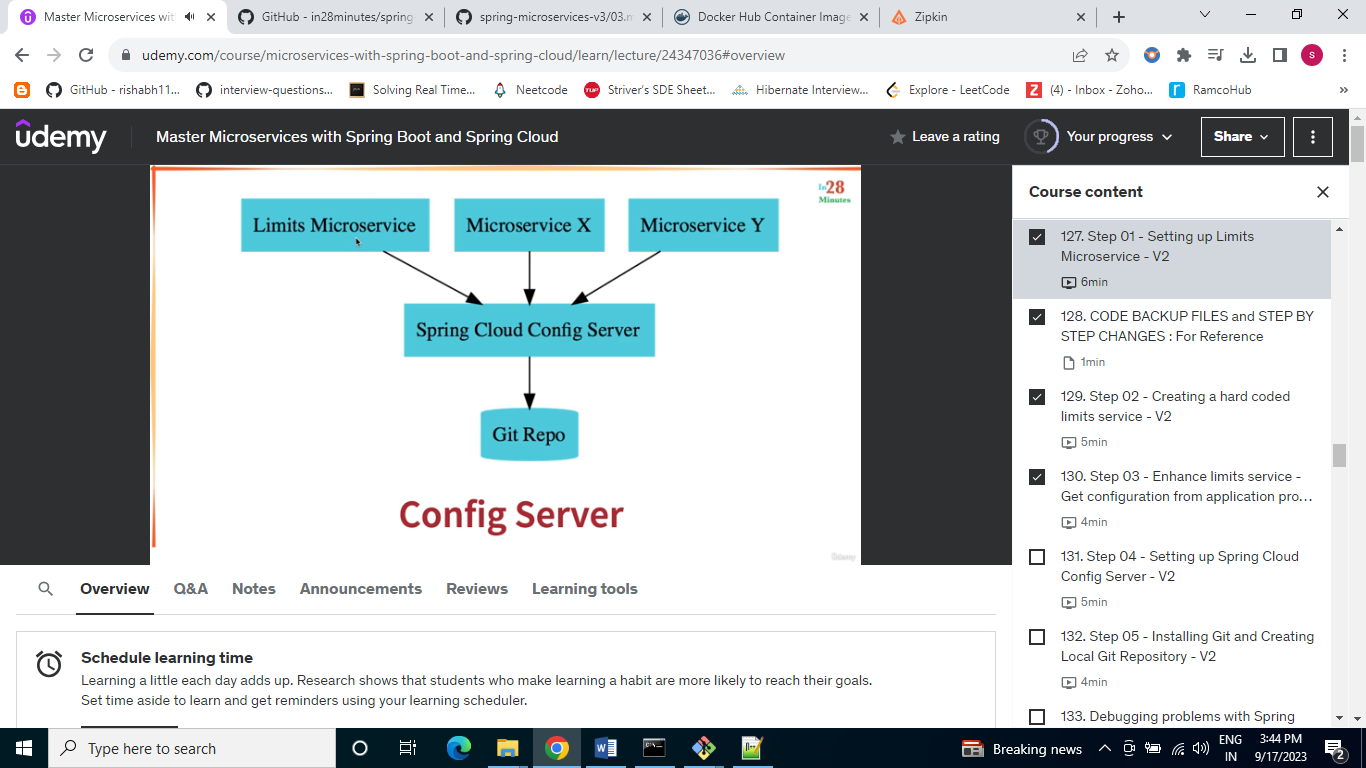






Microservices:-

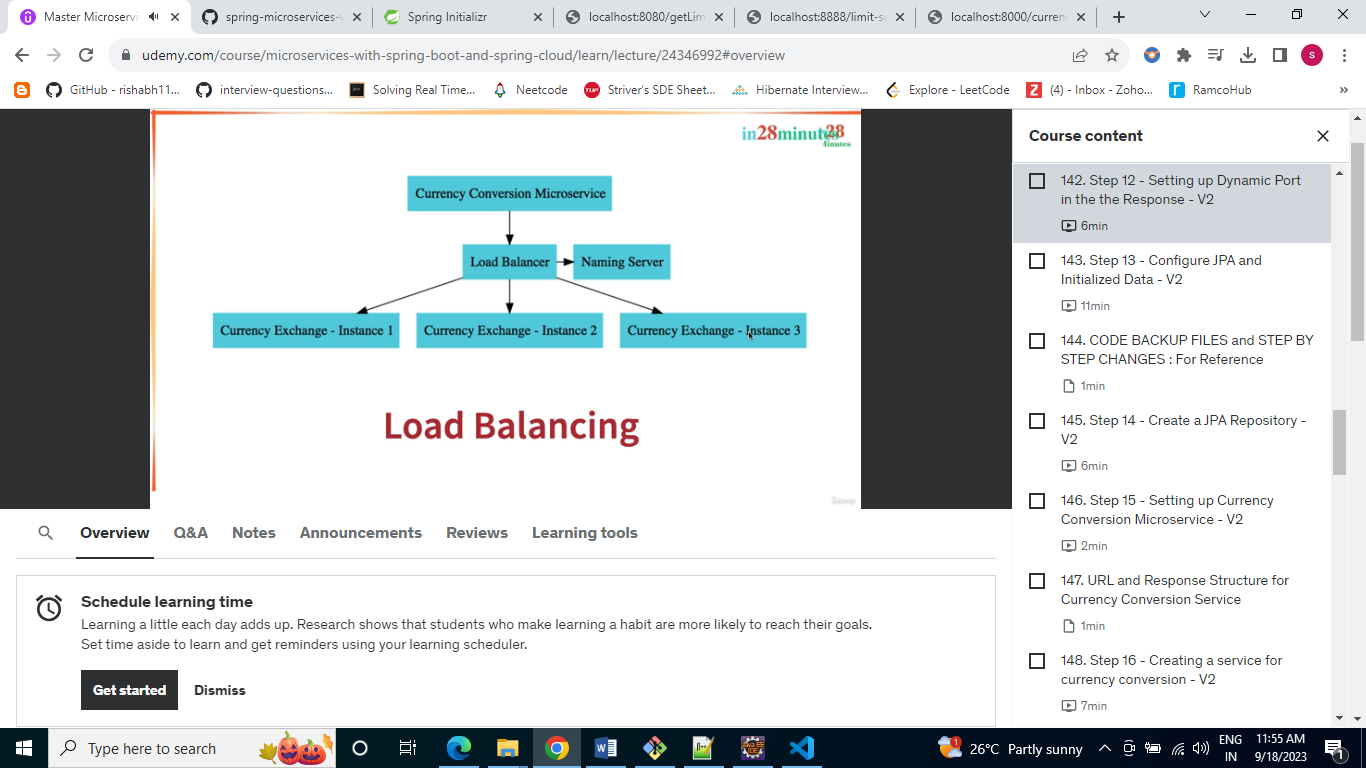




If we are connecting with spring cloud config add one dependency in pom.xml and also add in property file as well.

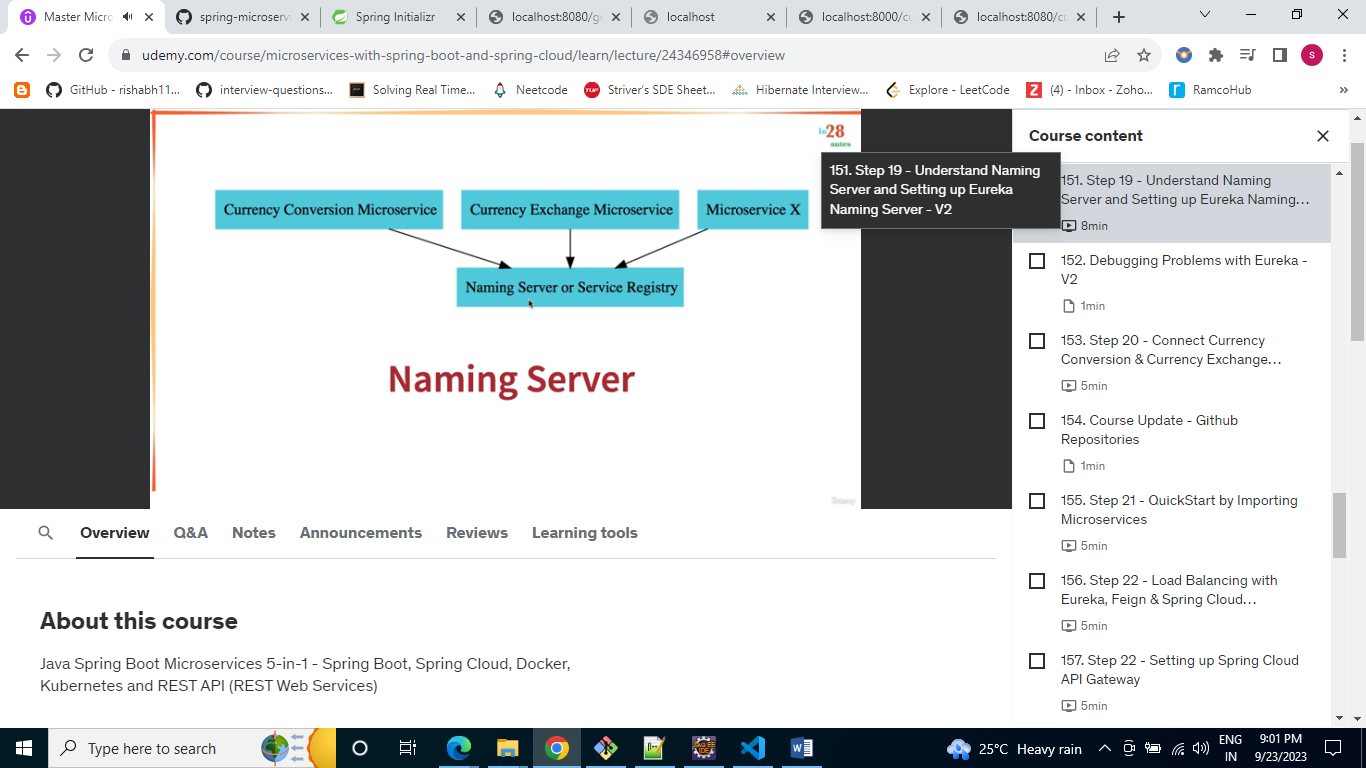
How we are connecting to the config server.

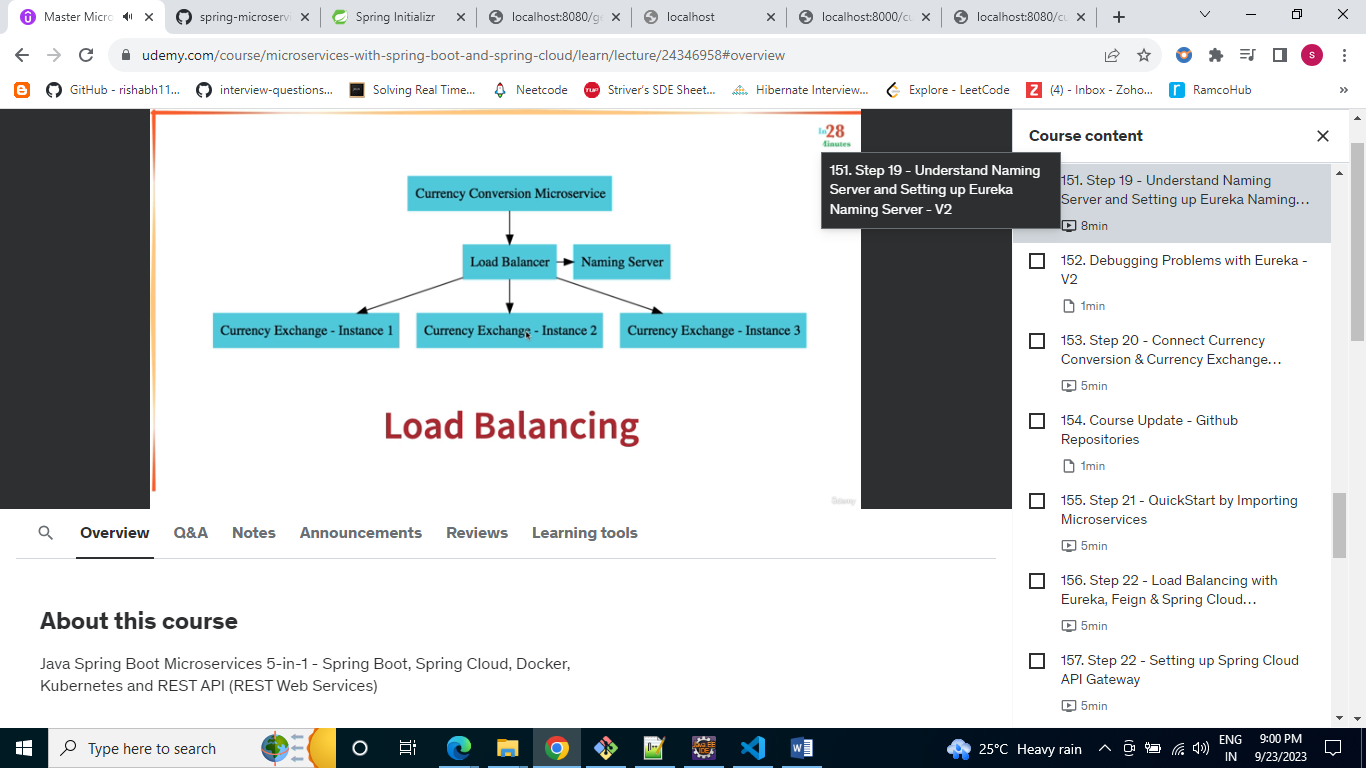
Spring.config.import=configserver:http://locathost:8888



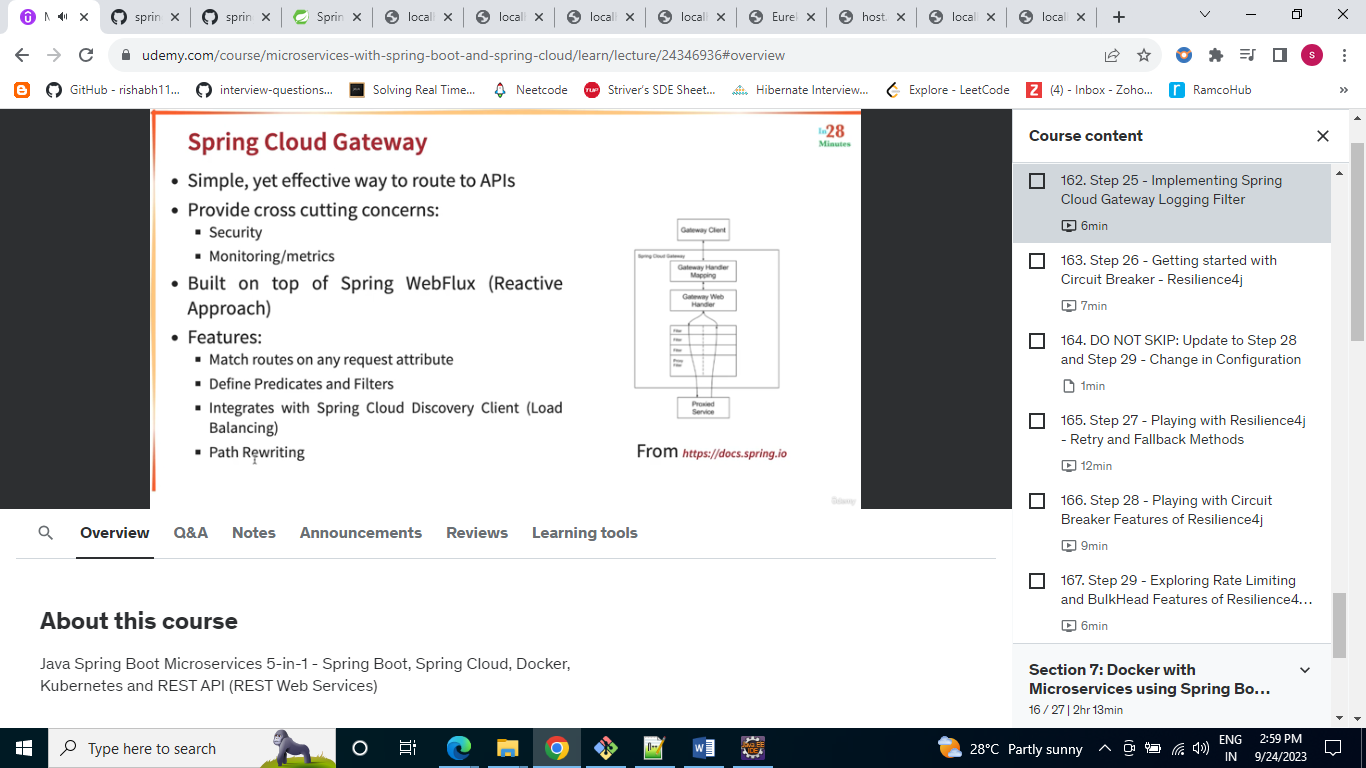
Feign framework:- used for mS communications

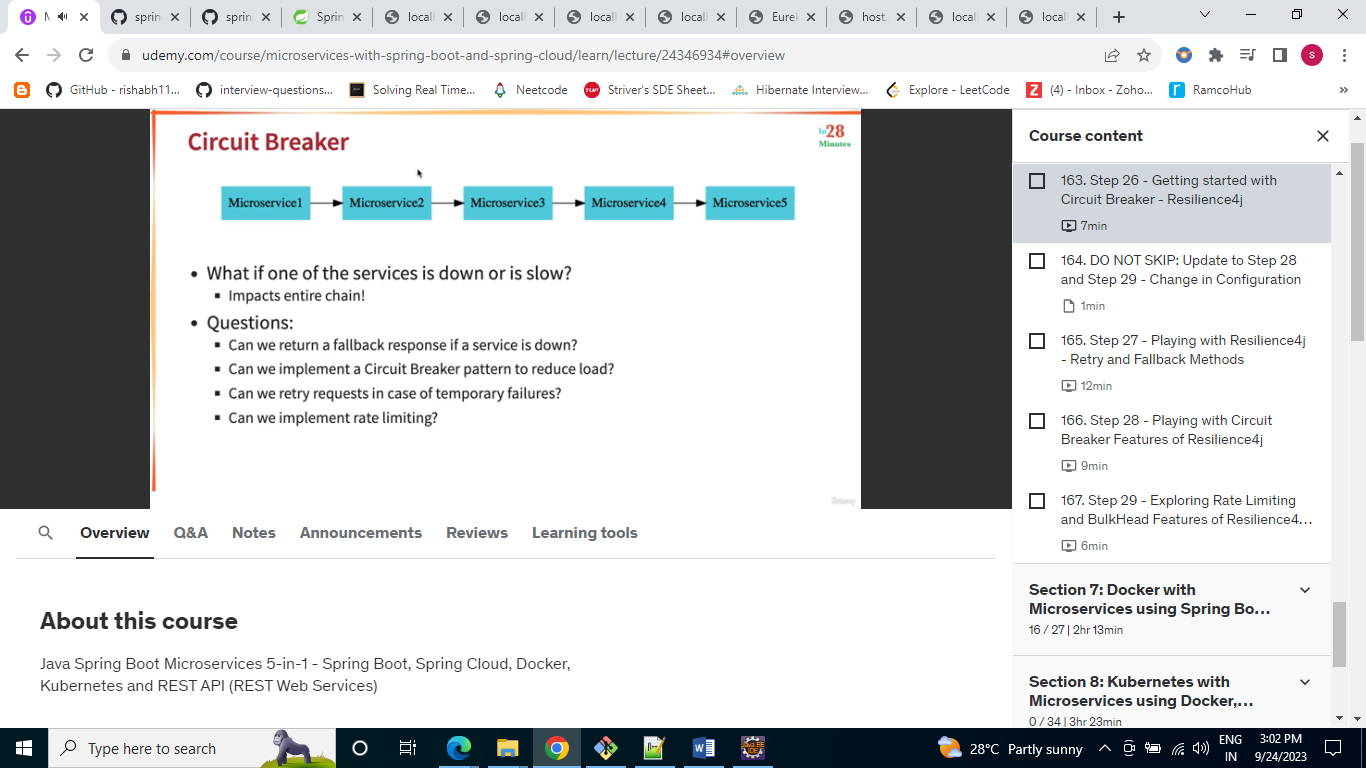
Naming server🡪 Edureka Naming Server.



we used Spring cloud load balancer which is used by feign client for load balancing.(Ribbon is old client side load balancer)

Api gateway🡪 will use spring cloud API gateway(before that we used zool(which is now old))





Resilience4j is inspire from Hystrix dashboard.

