

microservices - Notepad
File Edit Format View Help

profiles in services
dev tets stage prod

Navigate to resources > application.properties and add the following line of code in it-

spring.profiles.active=dev

spring cloud config--

@EnableConfigServer to enable the config server functionality. Each service pulls its configuration from the central repository on startup. externalized configuration in a distributed system. With the Config Server you have a central place to manage external properties for applications across all environments.

syn comm==microservices communicate over HTTP, use restTemplate, mc exposes endpoint so that other mc can call synchronously. allows services to exchange data & request using get/post/put/delete

async comm==mc will send msg or event to msg broker or event bus. services can publish events/msg to broker and other mc can subscribe to and consume those events

service discovery will communicate each other

strategies to improve performance of microservice

-asynchronous communication--here it allows handle request in non-blocking manner, so performance increase

-caching mechanism--reduce no of network, DB call

-circuit breaker--if 1 service fail, no need whole system should stop working..use fallback method

-api gateway--to avoid issue of complexity, inconsistency, authentication

--spring batch to make processing faster

if you have yaml and properties file both and same key in both, then spring boot will look first in properties and if not found then will go to yaml.

devtools--

if its enabled, any changes in code/resources/config, will be automatically detected and trigger restart.

no need to manual stop/start app during dev

service discovery has 2 types--client and server side info

Type here to search

Ln 265, Col 1 100% Windows
28°C Sunny

microservices - Notepad

File Edit Format View Help

how to define 2 db in same microservice

```
--in app.pro define both DB properties like datasource nm, usernm, password, driver name  
spring.jpa.generate-ddl=true,spring.jpa.show-sql=true  
--create separate datasource config classes for each db. one of the config must be @primary  
@Primary  
@Bean(name="datasource")  
@ConfigurationProperties(prefix = "spring.admissions.datasource")  
public DataSource dataSource(){  
    return DataSourceBuilder.create().build();  
}
```

microservices--collection of small, independent services that communicate with each other over a network. Instead of building a monolithic application where all the functionality is tightly integrated into a single codebase,

Microservices:

These are the individual, self-contained services that encapsulate specific business capabilities. Each microservice focuses on a distinct function or feature.

API Gateway:

The API Gateway is a central entry point for external clients to interact with the microservices. It manages requests, handles authentication, and routes requests to the appropriate microservices.

Service Registry and Discovery:

This component keeps track of the locations and network addresses of all microservices in the system. Service discovery ensures that services can locate and communicate with each other dynamically.

Load Balancer:

Load balancers distribute incoming network traffic across multiple instances of microservices. This ensures that the workload is evenly distributed, optimizing resource utilization and preventing any single service from becoming a bottleneck.

Database per Microservice:

Each microservice typically has its own database, ensuring data autonomy. This allows services to independently manage and scale their data storage according to their specific requirements.

Fault Tolerance and Resilience Components:

Implementing components for fault tolerance, such as circuit breakers and retry mechanisms, ensures that the system can gracefully handle failures in microservices and recover without impacting overall functionality.

circuit breaker--if 3 services r dependant and 2nd service is down.wont response

Ln 265, Col 1

100% Windows (CRL)



microservices - Notepad

File Edit Format View Help

circuit breaker--if 3 services r dependant and 2nd service is down,wont response
saga--transaction management section

rest template and registry????
bean lifecycle??init process destroy
spring security????
map filter diff??

~~Hystrix is a library developed by Netflix that implements the Circuit Breaker pattern.~~

In a microservice architecture, it is common to have multiple layers of service calls, i.e one microservice can call multiple downstream microservices. A service failure in any one of the lower level services can cause cascading failure all the way up to the user.

Circuit Breaker pattern provides a fallback mechanism, which avoids cascading of failures up to the user.
use spring-cloud-starter-netflix-hystrix dependency and

@EnableCircuitBreaker annotation on the Spring Boot Application class.

@HystrixCommand annotation on the method for which fallback method has to be applied.

Eureka

Eureka is a Service Discovery Server and Client provided in the Netflix OSS platform. Service Discovery is one of the key tenets of a microservice-based cloud architecture.

Eureka Client=spring-cloud-starter-netflix-eureka-client dependency and @EnableEurekaClient annotation

Eureka server=spring-cloud-starter-eureka-server dependency and @EnableEurekaServer annotation

Eureka is a key component of microservice architecture because it makes it easy to scale and manage microservices. Without Eureka, each microservice would need to know the location of all the other microservices it needs to communicate with. This would make it difficult to scale microservices because you would need to update the configuration of each microservice whenever you added or removed a microservice.

Eureka also makes it easy to failover to a different microservice if one fails. When a microservice registers with Eureka, it provides a list of other microservices that it depends on. If the microservice fails, Eureka will remove it from the registry, and other microservices will be notified. The other microservices can then failover to a different microservice that provides the same functionality.

To call microservices from the Eureka server, you can use the Eureka client library. The Eureka client library will query the Eureka registry to find the location of the microservice you want to call. Once it has the location of the microservice, it will make a call to the microservice.

Ln 265, Col 1

100% Windows (CRLF)

UTF

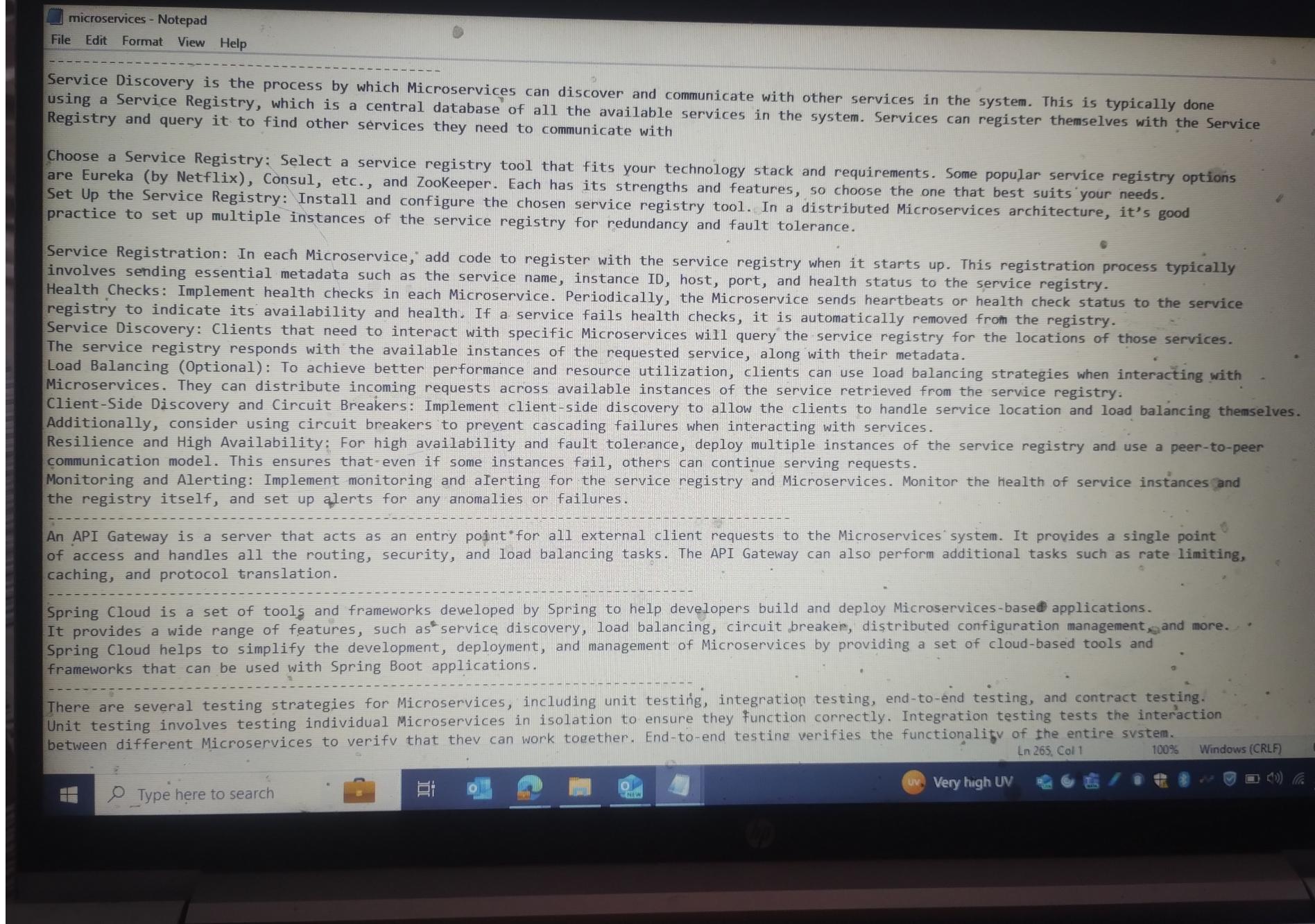


Type here to search



Very high UV





microservices - Notepad

File Edit Format View Help

means that issues with one Microservice will not affect the entire system.

12 factor app

Codebase

One codebase, multiple deploys. This means that we should only have one codebase for different versions of a microservices. Branches are ok, but different repositories are not.

Dependencies

Explicitly declare and isolate dependencies. The manifesto advises against relying on software or libraries on the host machine. Every dependency should be put into pom.xml or build.gradle file.

Config

Config

Store config in the environment. Do never commit your environment-specific configuration (most importantly: password) in the source code repo. Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system. Using Spring Cloud Config Server you have a central place to manage external properties for applications across all environments.

Backing services

Treat backing services as attached resources. A microservice should treat external services equally, regardless of whether you manage them or some other team. For example, never hard code the absolute url for dependent service in your application code, even if the dependent microservice is developed by your own team. For example, instead of hard coding url for another service in your RestTemplate, use Ribbon (with or without Eureka) to define the url:

Release & Run

Strictly separate build and run stages. In other words, you should be able to build or compile the code, then combine that with specific configuration information to create a specific release; then deliberately run that release. It should be impossible to make code changes at runtime, for e.g. changing the class files in tomcat directly. There should always be a unique id for each version of release, mostly a timestamp. Release information should be immutable, any changes should lead to a new release.

Processes

Execute the app as one or more stateless processes. This means that our microservices should be stateless in nature, and should not rely on any state being present in memory or in the filesystem. Indeed the state does not belong in the code. So no sticky sessions, no in-memory cache, no local filesystem storage, etc. Distributed cache like memcache, ehcache or Redis should be used instead.

Port Binding

Export services via port binding. This is about having your application as standalone, instead of relying on a running instance of an application server, where you deploy. Spring boot provides a mechanism to create a self-executable uber jar that contains all dependencies

Ln 265, Col 1

100%

Windows (CRLF)

UTF-8

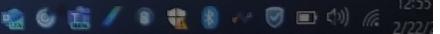
12:55

2/22/

Type here to search



Very high UV



12:55 2/22/

Windows (CRLF) UTF-8

12:55 2

microservices - Notepad

File Edit Format View Help

Concurrency

Scale-out via the process model. In the twelve-factor app, processes are a first-class citizen. This does not exclude individual processes from handling their own internal multiplexing, via threads inside the runtime VM, or the async/evented model found in tools such as EventMachine, Twisted, or Node.js. But an individual VM can only grow so large (vertical scale), so the application must also be able to span multiple processes running on multiple physical machines. Twelve-factor app processes should never write PID files, rather it should rely on operating system process manager such as systemd - a distributed process manager on a cloud platform.

Disposability

The twelve-factor app's processes are disposable, meaning they can be started or stopped at a moment's notice. This facilitates fast elastic scaling, rapid deployment of code or config changes, and robustness of production deploys. Processes should strive to minimize startup time. Ideally, a process takes a few seconds from the time the launch command is executed until the process is up and ready to receive requests or jobs. Short startup time provides more agility for the release process and scaling up; and it aids robustness because the process manager can more easily move processes to new physical machines when warranted.

Dev/Prod parity

Keep development, staging, and production as similar as possible. Your development environment should almost identical to a production one (for example, to avoid some "works on my machine" issues). That doesn't mean your OS has to be the OS running in production, though. Docker can be used for creating logical separation for your microservices.

Logs

Treat logs as event streams, sending all logs only to stdout. Most Java Developers would not agree to this advice, though.

Admin processes

Run admin/management tasks as one-off processes. For example, a database migration should be run using a separate process altogether.

12 factors of microservices

- codebase should b unique like git, sinle repo we shoudl use
- we should always explicitly declare all dependancies in pom
- we should externalize all configuration whose value will change as per environment like image role, username, password
- .when we ll add such things in property file no need of deployment just change property file values
- Backing services are services that the application depends on for operation. For instance a database or a message broker.

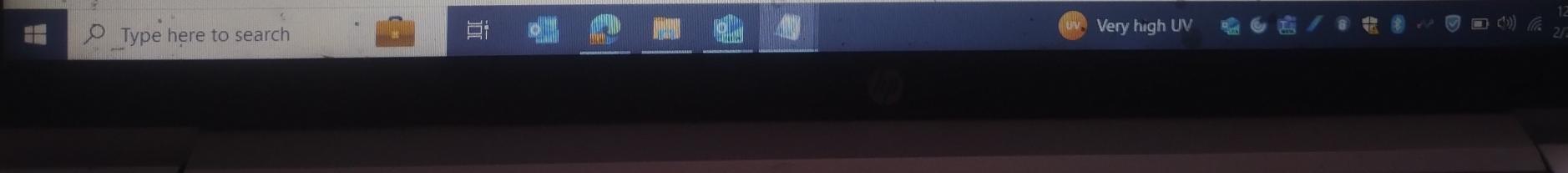
@Repository

```
public interface MovieRepository extends JpaRepository<Movie, Long> {  
}  
-process 1. build(take the code,perform static and dynamic checks,generate executable jar/war)  
 2. release (take war , add right config)  
3. run(here ll run app in target execution env)
```

Ln 265, Col 1

100% Windows (CRLF) UT

12
2/



microservices - Notepad

File Edit Format View Help

--process 1. build(take the code, perform static and dynamic checks, generate executable jar/war)

2. release (take war, add right config)

3. run(here ll run app in target execution env)

- process should b stateless

- expects no such runtime dependency. It's completely self-contained and only requires an execution runtime like Java.

- concurrency-Java offers Thread as a classical model to handle concurrency in an application. Threads are like lightweight processes and represent multiple paths of execution in a program. Threads are powerful but have limitations in terms of how much it can help an application scale.

The twelve-factor methodology suggests apps to rely on processes for scaling

- Dev/Prod Parity-means we develop all in local and deploy in prod..but server setup ll b diff in local and prod.so to keep minimum gap between env so that only few changes req

- logs are nothing but a time-ordered stream of events

The term monolithic applies to tightly integrated applications where it is hard to change one function without also recoding other parts of the application. It's highly tight coupled.to add any new feature, need complete a deployment and whole testing of other existing features also.

REST has below

Stateless: In stateless communication, the server does not store any information about past requests/responses. Each request and response contains all information needed to complete the interaction. Stateless communication reduces server load, saves memory, and improves performance. It also eliminates the possibility of a failed request caused by missing data.

Cacheable: Server responses indicate whether or not the resource is cacheable, so that clients can cache any resources to improve performance.

the microservice-based system, each feature can be divided into smaller subsystems like service to handle patient registration, service to handle database management, service to handle billing etc

if we r giving @produces(MediaType.APPLICATION_JSON) at class level and giving @produces(MediaType.

APPLICATION_XML) at function level

then it will take function level format i.e. XML not JSON

and if we r not giving any other format at function level then it will take class level format as JSON

post--created new object

put--if obj present updates it , if not present create it.to update need to send whole body and update updated param

patch--partial update..no need to send complete body..send only value wh need to update

What Is ACID Property Of A System?

ACID is a acronym which is commonly used to define the properties of a relational database system, it stand for following terms

Ln 265, Col 1

100% Windows (CRLF)

UTI



Type here to search



Very high UV



12:22

microservices - Notepad
File Edit Format View Help

patch--partial update..no need to send complete body..send only value wh need to update

What Is ACID Property Of A System?
ACID is a acronym which is commonly used to define the properties of a relational database system, it stand for following terms

Atomicity - This property guarantees that if one part of the transaction fails, the entire transaction will fail, and the database state will be left unchanged.

Consistency - This property ensures that any transaction will bring the database from one valid state to another.

Isolation - This property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially.

Durable - means that once a transaction has been committed, it will remain so, even in the event of power loss.

=====

JAX-RS Injection

```
@pathparam @matrixparam @queryparam @headerparam  
@cookieparam @beanparam @formparam @context  
=====
```

messageservice.java

```
public message getmessage(int id)  
{  
    return messages.get(id);  
}  
public message addmessage(message message){  
    message.setid(messages.size()+1);  
    messages.put(message.getid(), message);  
    return message;  
}  
public message deletemessage(int id){  
    return messages.remove(id);  
}  
public message updatemessage(message message){  
    messages.put(message.getid(), message);  
    return message;  
}  
public List<message> getallmessagespaginated(int start, int size){  
    return new ArrayList<message>(messages.values());  
}
```

Ln 265, Col 1 100% Wind