**microservices:**

Its an **architecture style** that structures an application as a collection of services that are
--independently deployable+development
--independent service
--easy to maintain & develop
--technology freedom
--business capabilities
--loosely coupled

**Microservice Characteristics:**
--smart endpoints—HTTP—REST(Representational State Transfer)
--products not projects
--decentralized data management
--infrastructure automation

**Principles of microservices:**
--single responsibility per service
--micorservices are autonomous

**@service**— used to mark the class as service provider..it's used with classes that provides some business functionalities.
**@repository**— used to indicate the class that provides mechanism for storage,retrieval,search,update n delete operation on objects
**@restcontroller**— allows the class to handle the requests made by the client.
**@requestmapping**— used to map http requests to handler methods of rest controllers.
**@entity**— specifies the class is an entity and is mapped to a database table.
**@table**—specifies the name of the database table to be used for mapping.
**@id—** specifies the primary key of an entity
**@generatedvalue**—provides for the specification of generation strategies for the values of primary keys.
**@data**—bundles all the getter/setters/constructors together
**@postmapping**(creating a resource)—used for mapping http post requests onto specific handler methods.
**@getmapping**(getting a resource)— used for mapping http get requests onto specific handler methods.
**@putmapping**(updating a resource)-- used for mapping http put requests onto specific handler methods.
**@deletemapping**(deleting a resource)-- -- used for mapping http delete requests onto specific handler methods.
**@NoArgsConstructors**—generate a constructor with no parameters.
**@AllArgsConstructors**—generate a constructor with one parameters for every field in the class.
**@Autowired**—used for automatic dependency injection
**@Qualifier**—to specify which exact bean will be wired.
**@Override—**denotes that child class method overrides the base class method

Spring cloud: provides tools for developers to quickly build some of the common patterns(eg: config,circuit breakers,micro proxy,service to service calls, etc)

- **Spring cloud config**-- stores properties in a version controlled repository such as git…
- Config port: 8888
- Add @Enableconfigserver—makes spring boot app acts as a configuration server

- Actuator/refresh endpoint---it just refreshes the client to which the request is made.
- Rabbitmq—common messaging that allows applications to connect and communicate.

**Eureka:**

- Eureka—used for self-registration, dynamic discovery and load balancing.
- Eureka—8761
- @EnableEurekaServer—is used to make spring boot app acts as a eureka server.
- @EnableEurekClient--- to register the spring boot app into eureka server.

**Feign:**

- It simplifies API calls.
- Feign makes it easy to invoke other microservices.
- Purpose of feign client – to communicate microservices with each other
- To use feign, create an interface and annotate it.
- @Feign-client: used to consume REST API endpoints which are exposed by third party or microservices.
- Create proxy interface
- Annotate with @FeignClient("service-name")
- Autowire the proxy in service/controller
- Add @EnabaleFeignClients in the main class

**Resilience4j:**

- It's a lightweight fault tolerance library designed for functional programming.
- It provides method reference with a circuit breaker, retry.,rate limiter,bulkhead.
- Resilience4j has 6 core modules—retry,circuitbreaker,ratelimiter,timelimiter,bulkhead,cache.

**Circuit Breaker: used to find the state transition in /actuator/circuitbreakerevents**
**Circuit Breaker: (how does it works):**

- @circuitbreaker(name=" " , fallbackmethod=" ")
- /actuator/circuitbreakers—gives the list of circuitbreakers(shows service name)
- /actuator/circuitbreakerevents- gives list of all events of circuitbreakers(shows service name,status,timeout,etc)
- Status success means, it is in closed state.
- Status error means, it is in open state or half open state.
- Definition: It temporarily blocks possible failures.
- It automatically reconnects back to the primary service when the service is back to normal.
- It has 3 states: closed, open, half-open.
- Closed: when everything is normal in the beginning, it will be in closed state.
- Open: if failures exceeds the threshold values,it will go into open state.
- Half-open: first call will not fail, and all other calls will fail just as in open state.

**Circuit breker configuration properties:**

- Management.health.circuitbreakers.enabled—we can enable the circuitbreaker health indicators via this configuration.
- Slidingwindowtype—count based—used to record the outcome of calls when circuitbreaker is closed.
- MinimumNumberofCalls—100 – circuitbreaker can calculate the error rate using this configuration.
- Eventconsumerbuffersize—100 – configures the size of buffers having events emitted by the circuit breaker,retry.
- failureRateThreshold—50—configures the failure rate threshold in percentage.

- automaticTransitionFromOpenToHalfOpenEnabled—false—if it set to false, the transition to HALF_OPEN only happen if a call is made. The advantage here is no thread monitors the state of all circuitbreakers.
- waitDurationOpenState—60000—the time that the circuitbreaker should wait before transitioning from open to half-open.

**Retry:**

- Retry decorator will help u to retry the failed decoration.

**Retry: (how does it works):**

- It repeats failed executions.
- Many faults are transient and may self-correct after a short delay.

**Retry config properties:**

-- maxAttempts – 3 – tells how many times to retry.
-- waitDuration—500 ms—a fixed wait duration btw retry attempts.

**Zipkin:**

- It's a very efficient tool for distributed tracing in the microservices ecosystem.
- If we troubleshooting latency problems or errors in the microservice means, we can filter,sort all traces,length of trace,timestamp of the application in zipkin.
- Internally it has 4 modules: collector, storage, search, web ui.

**Sleuth:**

- It's another tool from the spring cloud family. It is used to generate the trace id/span id and add the information to the service calls in the headers. So that it can be used by tools like zipkin.

**Trace ID:**

- When a new flow starts, a new trace id is generated.

**Span ID:**

- Is the unit of work.
- For example, for every new http request that your microservice will send, a new span id will be created.
- Trace id will be same but span id will be new for every http requests.