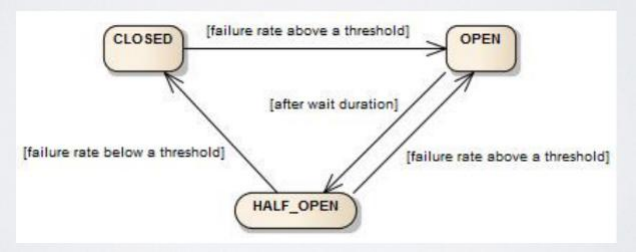
Beaconfire Inc, Home Work, Week7 Day29.
By (Ping) Nalongsone Danddank.
ndanddank@gmail.com
wechatID: ndanddank
Short Answer:

1. Explain cascading failure in microservice and how to prevent it.

-> the cascading failure in Microservice is the situation like when there are multiple caller making calls to unresponsive supplier, so it is possible to run out memory while waiting, which may lead to cascade failure in the application. We can prevent it by implement Fault Tolerance, a system's ability to continue operating uninterrupted despite the failure of one or more of its component. Example: using Circuit Breaker, Bulkhead, Rate Limiter, and Retry.

2. Explain CircuitBreaker and its states.

-> CircuitBreaker pattern is the solution for cascading failure in microservice.
It's idea are wrap a protected function call in a circuit breaker object, which monitors for failures. And Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error, without the protected call being made at all. We have a module in spring cloud project names Spring Cloud Netfix Hystrix which implement the Circuit Breaker pattern. • It is a fault-tolerance library, which is describing a strategy against failure cascading at different levels in an application
• The principle is analogous to electronics: Hystrix is watching methods for failing calls to related services.
• If there is such a failure, it will open the circuit and forward the call to a fallback method.
• The library will tolerate failures up to a threshold. Beyond that, it leaves the circuit open.
RESILENCE4J - CIRCUIT BREAKER : A finite state machine with three normal states: CLOSED, OPEN and HALF_OPEN



3. Explain Spring Cloud Configuration.

-> • Spring Cloud Config provides server-side and client-side support for externalized configuration in a distributed system
• With the Config Server, you have a central place to manage external properties for applications across all environments.
• The concepts on both client and server map identically to the Spring Environment and PropertySource abstractions, so they fit very well with Spring applications but can be used with any application running in any language.
• As an application moves through the deployment pipeline from dev to test and into production, you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate
• The default implementation of the server storage backend uses git, so it easily supports labelled versions of configuration environments as well as being accessible to a wide range of tooling for managing the content.

4. Describe the CAP Principal.

-> CAP stands for Consistency, Availability and Partition Tolerance.

• Consistency (C): All nodes see the same data at the same time. What you write you get to read.

• Availability (A): A guarantee that every request receives a response about whether it was successful or failed. Whether you want to read or write you will get some response back.

• Partition tolerance (P): The system continues to operate despite arbitrary message loss or failure of part of the system. Irrespective of communication cut down among the nodes, system still works.

• We can only achieve either PC or PA — Microservices are distributed/partitioned by nature!