**Circuit Breaker Pattern**

The Circuit Breaker pattern is a popular design pattern used in Microservices Architecture, that falls under the Sustainable Design Patterns category. In Microservices architecture, a service usually calls other services to retrieve data, and there is the chance that the downstream service may be down. It may be cause by slow network connection, timeouts, or temporal unavailability. Therefore, retrying calls can solve the issue. However, if there is a severe issue on a particular microservice, then it will be unavailable for a longer time. In such case, the request will be continuously sent to that service, since the client doesn’t have any knowledge about a particular service being down. As a result, the network resources will be exhausted with low performance and bad user experience. Also, the failure of one service might lead to **Cascading failures** throughout the application.

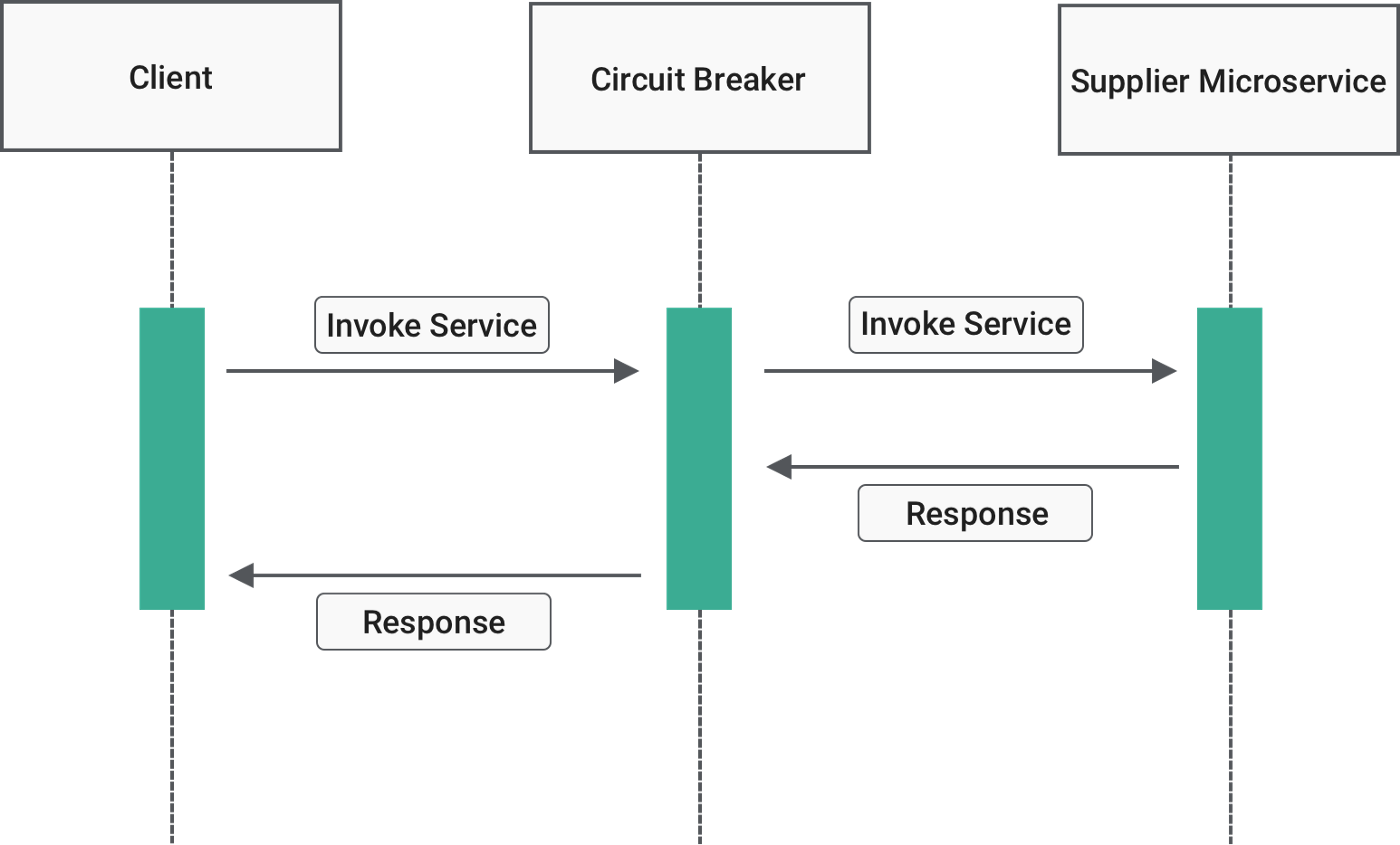
Therefore, you can use the Circuit Breaker Design Pattern to overcome this problem. With the help of this pattern, the client will invoke a remote service through a proxy. This proxy will basically behave as an electrical circuit breaker. So, when the number of failures crosses the threshold number, the circuit breaker trips for a particular time period. Then, all the attempts to invoke the remote service will fail within this timeout period. After the timeout expires, the circuit breaker allows a limited number of test requests to pass through it. If those requests succeed, the circuit breaker resumes back to the normal operation. Otherwise, if there is a failure, the timeout period begins again.

The Circuit Breaker Design pattern have three states:

1. Closed
2. Open
3. Half-Open

**Closed state**

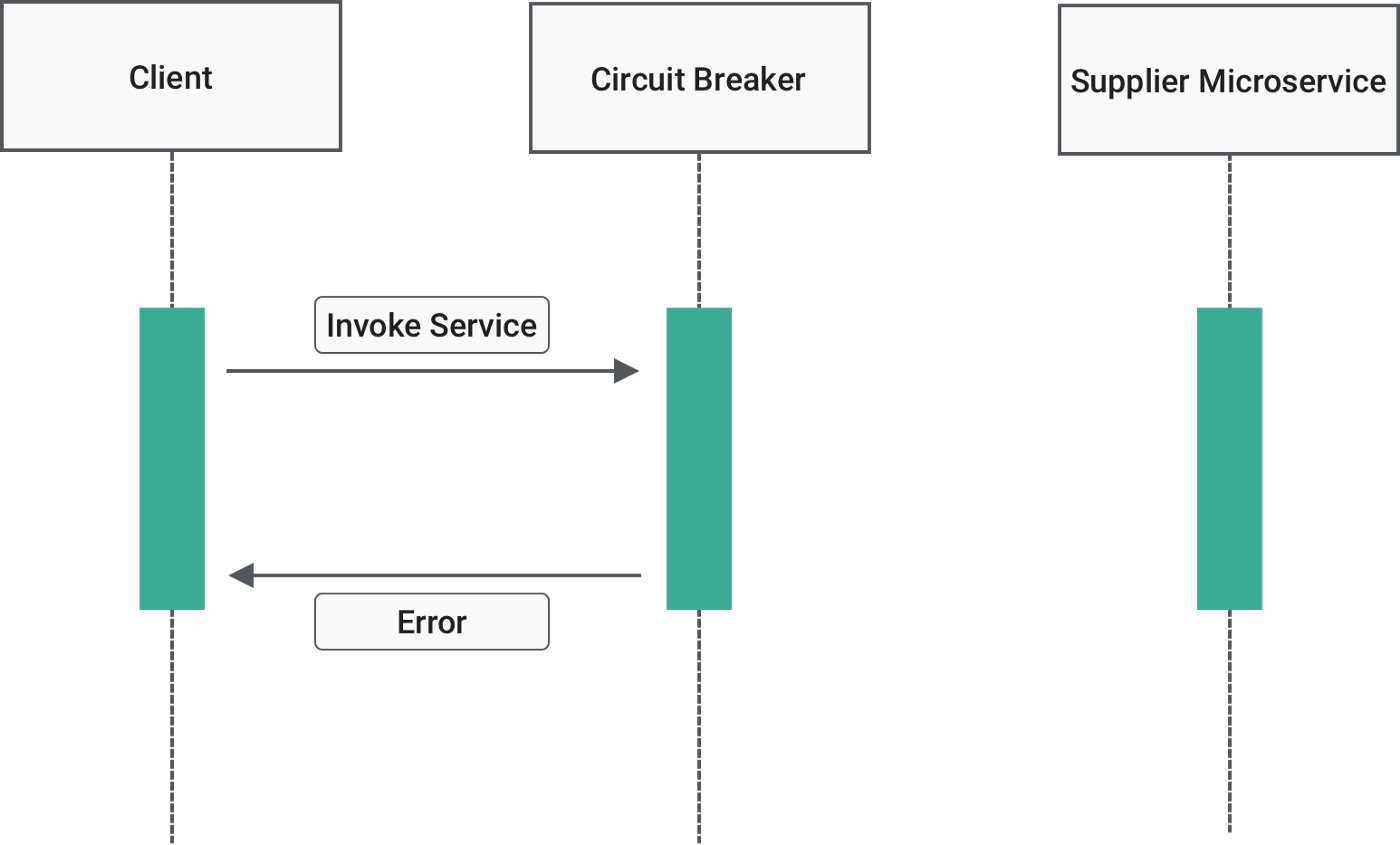
In this state, the Circuit Breaker routs the requests to the Microservice and counts the number of failures in each period of time. That means it work without any failures. But if the number of failures in a certain period of time exceeds a threshold, the circuit will trip and will move to an “Open” state.



Representing Closed state (blogs.halodoc.io)

**Open state**

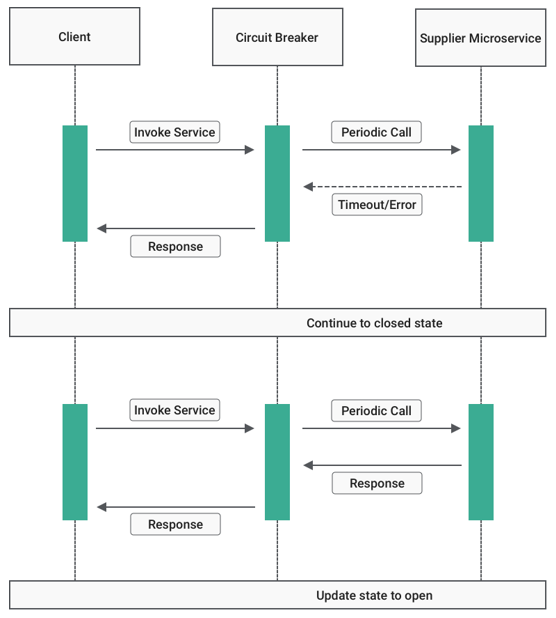
When Circuit breaker moves to the “Open” state, requests from the Microservices will fail immediately, and an exception will be returned. However, after a timeout, the Circuit Breaker will go to the “Half-Open” state.



Representing Open state (blogs.halodoc.io)

**Half-Open state**

In this state, the Circuit Breaker allows only a limited number of requests from the Microservice, to pass through and invoke the operation. If these requests are successful, the Circuit Breaker will go back to the “Closed” state. However, if any request fails again, it goes back to the “Open” state.



Representing Half-open state (blogs.halodoc.io)

**Why Availability Matters in Microservices Architecture**

Usually when we develop a system or application using whatever architecture (Microservices/ Monolithic), we guaranteed our clients, that the application availability will be 99.999%. That means there is only 0.001% chance for service downtime. At first, you may think that we don’t even have to care about this 0.001%. In that case let’s do some math.

**24 hrs \* 365 days = 8760 hours per year8760 hrs \* 60 mins = 525600 minutes per year525600 mins X 0.001% = 5.256 minutes of downtime**

As you can see, 0.001% of non-availability cost total 5.256 minutes of downtime per year. If you still think this is okay since its just 5 min, you are still wrong. Because when it comes to microservices, we have multiple services. As an example, assume there are 100 microservices in your application.

**(5.256 \* 100)/ 60 = 8.78 hours of downtime**

So, now you can see how critical it is. If you have 100 microservices, it cost total 8.78 hours of downtime in a year. Obviously, it’s not acceptable at all. Therefore, we need to pay more attention in order to protect our services.

**Use Case of Circuit Breaker Pattern**

Let’s take an example to understand, where we can apply Circuit Breaker Pattern in Microservices architecture.

**Scenario:**

Assume there are 5 different services in a Microservices application. Whenever it receives requests, the server will allocate one thread to call the particular service. But, due to some failure, the service is little delayed, and the thread is waiting. However it’s okay, if only one thread is waiting for that service. But, if the service is a high demanding service that gets many requests, it is not good to hold. Because more threads will be allocated for this service within some time, and these threads will have to wait.

As a result, the remaining requests that comes to your service will be blocked or queued. Even though, the service is recovered back, the webserver is still trying to process the requests that in the queue. Because the webserver will never recover, since it receives requests continuously. Eventually this might lead to Cascading failures throughout the application. Therefore, this kind of scenarios will lead to crash your services and even the application.

**Solution:**

The above scenario is a perfect example to apply Circuit Breaker Pattern. Assume you have defined threshold for a particular service, as it should respond within 200ms. As I mentioned above, that service is a high demand service, that continuously receive requests. In case, if 75% of those requests are reaching the upper threshold (150ms — 200ms) means that service is going to fail soon. However, if several requests exceed the maximum threshold (200ms) means that service not responding anymore. As a result, it will fail back to the consumer and inform this particular service is not available. So, if you remember the above-mentioned states of this pattern, now we are moving to “Open” state from the “Closed” state.

As a result, all those requests that comes to the particular service, not going to wait anymore. However, after a timeout, the Circuit Breaker sends ping requests to that service in the background. That means now we are in the “Half-Open” state of the Circuit Breaker pattern. If these requests are successful, the Circuit Breaker will allow to send requests for that service again.

So, you can use the Circuit Breaker Pattern to improve the fault-tolerance and resilience of the Microservice Architecture and also to prevent the cascading of failure to other microservices.

The Bulkhead pattern is **a type of application design that is tolerant of failure**. In a bulkhead architecture, elements of an application are isolated into pools so that if one fails, the others will continue to function

**Rate Limiter Pattern:**

In Microservice architecture, when there are multiple services (A, B, C & D), one service (A) might depend on the other service (B) which in turn might depend on C and so on. Let’s consider this example. We have 2 services A and B.  Service A depends on Service B. The Service B has to do a lot of ***CPU/IO intensive work*** for the requests it receives. So it usually takes time to respond because of the nature of its work. Service B has a limit on max number of requests it can handle within the given time window

In a microservices architecture, the retry pattern is **a common pattern for recovering from transient errors**. An application lost connectivity for a short period of time. A component is unavailable for a short period of time. This usually happens during maintenance or automatic recovery from a crash.