# Circuit Breaker Pattern (Design Patterns for Microservices)


Hasitha Subhashana Follow 10 min read · Jun 12, 2021



Source: https://womenyoushouldknow.net/

In a distributed system we have no idea how other components would fail. Network issues could occur, components could fail or a router or a switch might be broken along the way. We don't know what could go wrong.
Therefore each component in a distributed system is responsible for staying alive. As a software engineer, you are responsible to keep them alive.

## What is a circuit breaker?

🌟 To learn about Circuit Breaker design pattern, you should know about circuit breakers first.

🌟 If your house is powered by electricity, I'm sure your house has a circuit breaker. When you get power from the main grid, it comes through a circuit breaker.

🌟 A circuit breaker is an electrical switch that operates automatically to

protect an electrical circuit from damage done by an overload (ex: lightning strike) or short circuit. Its main purpose is to stop current flow when a defect is identified and protect the electrical appliances at your home.

🌟 You want to know this because, how a circuit breaker works is more or less very equal to what a Circuit Breaker design pattern does.


Circuit breaker (Source: https://pixabay.com/users/diermaier-1497330/)

. . .

## Circuit Breaker Pattern

🌟 The Circuit Breaker design pattern is used to stop the request and response process if a service is not working, as the name suggests.

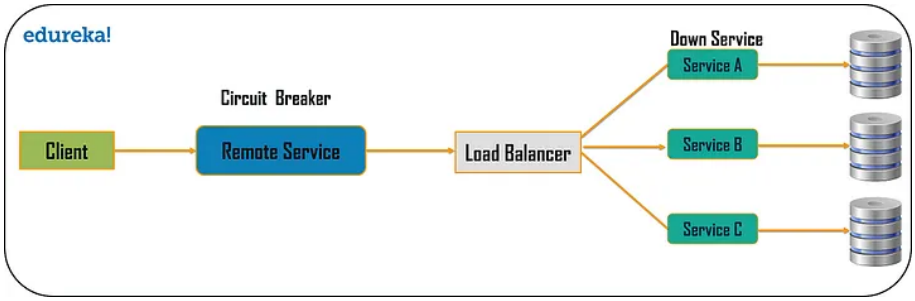



Figure 1: Circuit Breaker Pattern

🌟 As an example, assume a consumer sends a request to get data from multiple services. But, one of the services is unavailable due to technical issues. There are mainly two issues you will face now.

- First, because the consumer will be unaware that a particular service is unavailable (failed), so the requests will be sent to that service continuously.
- The second issue is that network resources will be exhausted with low

performance and user experience.

> *We'll go over thsese issues with use cases. Just keep reading* 😃

🌟 You can leverage the Circuit Breaker Design Pattern to avoid such issues. The consumer will use this pattern to invoke a remote service using a proxy. This proxy will behave as a circuit barrier.

🌟 When the number of failures reaches a certain threshold, the circuit breaker trips for a defined duration of time.

🌟 During this timeout period, any requests to the offline server will fail. When that time period is up, the circuit breaker will allow a limited number of tests to pass, and if those requests are successful, the circuit breaker will return to normal operation. If there is a failure, the time out period will start again.

> 🤔 *Still confused about how Circuit breaker pattern works. Im sure you will have a better idea at the end of this article.*

But before we dive into more details I want you to be familiarized with two things,

1. The main use case we are going to use for this article.
2. How availability could affect if our services failed.

. . .

## The main use case for Circuit Breaker

🌟 This will be the main example Im using to explain several use cases with Circuit Breaker pattern.

🌟 If you have gone through my previous Micorservioce blogs you are probably familiar with the example of Mercantile Finance.

🌟 Well, if you aren't familiar with my previous blogs, do not worry. Because you don't have to know the whole scenario. (But I hope you are familiar with the Aggregation pattern).

🌟 You have implemented employee management for Mercantile Finance with microservice architecture. This system has the below services.

- 🔩 Service 1: get personal information.
- 🔩 Service 2: get leave information.
- 🔩 Service 3: get employee performance information.
- 🔩 Service 4: get allocation information.

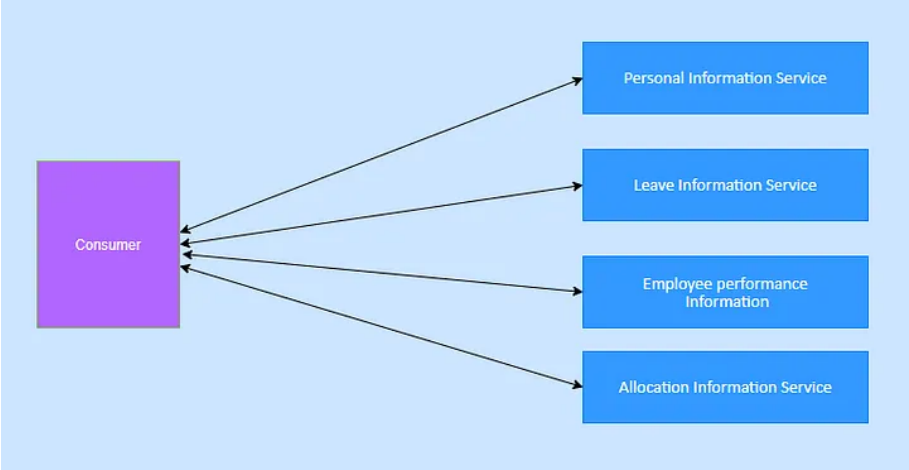


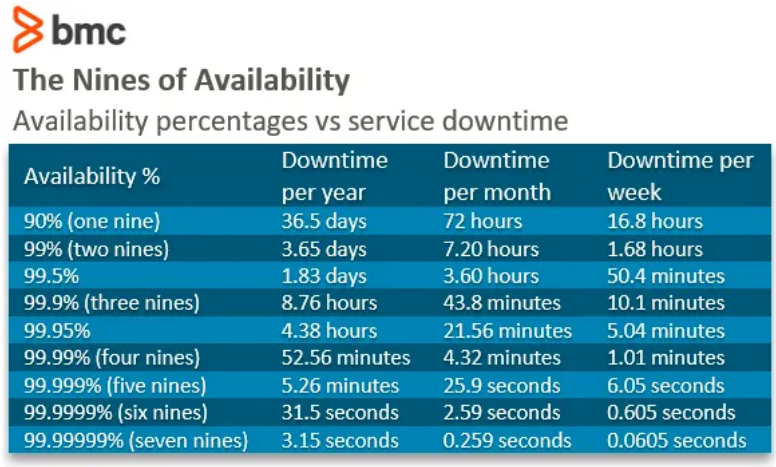

Figure 2

🌟 It's very much possible when there are multiple services, those services call (using aggregator pattern) for multiple backends(services).

🌟 So that all you want to know about the main use case scenario. Now let's see why availability is very critical for microservices.

. . .

## Why availability is critical in Microservices Architecture




| Availability % | Downtime per year | Downtime per month | Downtime per week |
|---|---|---|---|
| 90% (one nine) | 36.5 days | 72 hours | 16.8 hours |
| 99% (two nines) | 3.65 days | 7.20 hours | 1.68 hours |
| 99.5% | 1.83 days | 3.60 hours | 50.4 minutes |
| 99.9% (three nines) | 8.76 hours | 43.8 minutes | 10.1 minutes |
| 99.95% | 4.38 hours | 21.56 minutes | 5.04 minutes |
| 99.99% (four nines) | 52.56 minutes | 4.32 minutes | 1.01 minutes |
| 99.999% (five nines) | 5.26 minutes | 25.9 seconds | 6.05 seconds |
| 99.9999% (six nines) | 31.5 seconds | 2.59 seconds | 0.605 seconds |
| 99.99999% (seven nines) | 3.15 seconds | 0.259 seconds | 0.0605 seconds |

Figure 3: Levels of availability

🌟 So assume that, when your company implemented these services of the employee management system, they promised an uptime of `99.999%` (five nines) to Mercantile finance.

💻❓ How the maximum downtime is calculated for `99.999%`

`24` `hours` per day and `365` `days` per year = `8760` `hours` per year.

`8760` x `60` = `525600` `minutes` per year.

`99.999%` uptime means accepted failure change = `0.001%`

`525600` X `0.001%` = `5.256 minutes`

🌟 One service can be down `5.256 minute`s per year. A downtime of 5.25 minutes per year is totally fine for a monolithic system.

🌟 But unlike monolithic architecture, in a microservice architecture, there could be many services. Assume that there 100 services. This could cost **8.78 hours** of downtime per year. Now that's a critical number 😨 . This is why we need to pay attention to protect our services.

So, let's see what are the causes to break the service.

. . .

## What are the causes to break the services ?

### Use case 1

🌟 Assume that you have 5 different services and you have a webserver to call these services. Now when a request is received, the server allocates one

thread to call the service.

🌟 Now what happens is that this service is a little delayed (due to a failure) and this thread is waiting. One thread waiting would be fine.

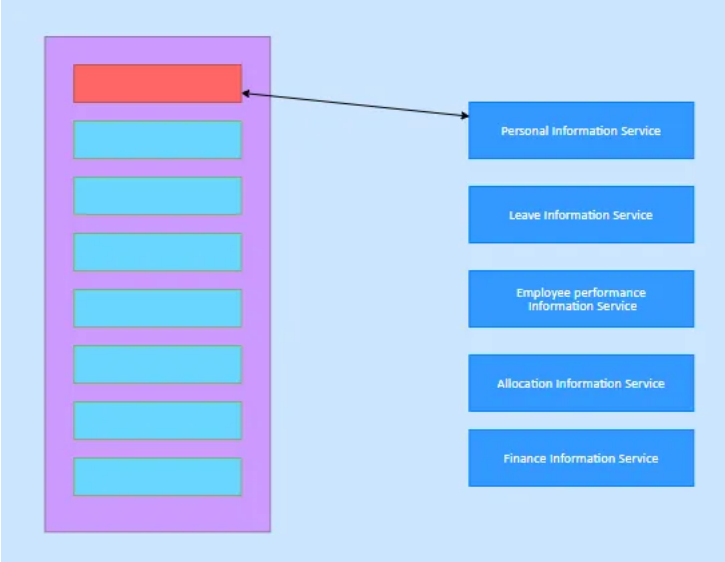



Figure 4

🌟 But if this service is a high demand service (gets more and more requests), more threads will be allocated for this service and all the threads allocated to call the service will have to wait.

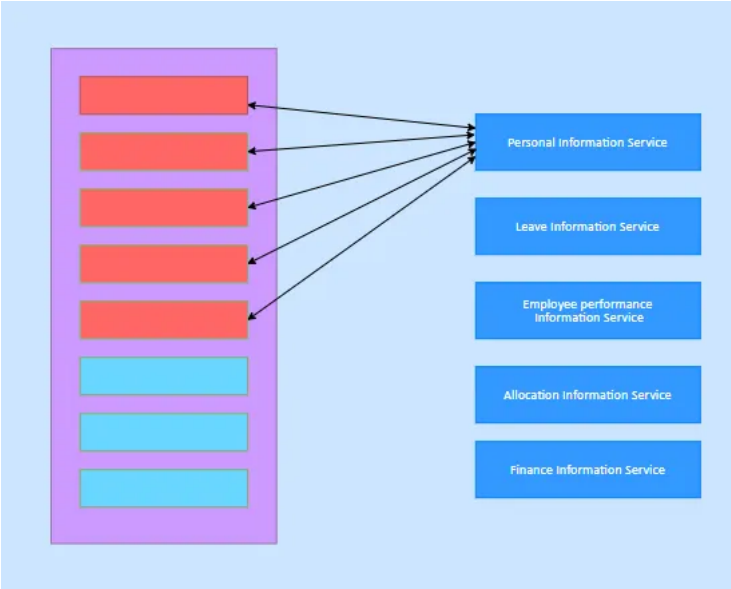



Figure 5

🌟 So, if you had 100 threads now 98 of them could be occupied and if the other two threads are occupied by another 2 services all the threads are now blocked.

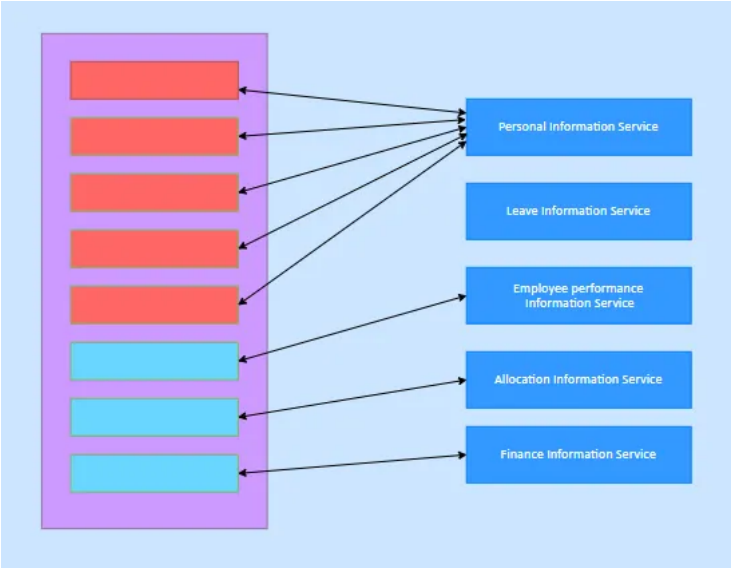



Figure 6

🌟 Now what happens is the remaining requests the reach your service will be queued (blocked). So meanwhile 50 more requests also came and all of them were queued because threads were feezed.

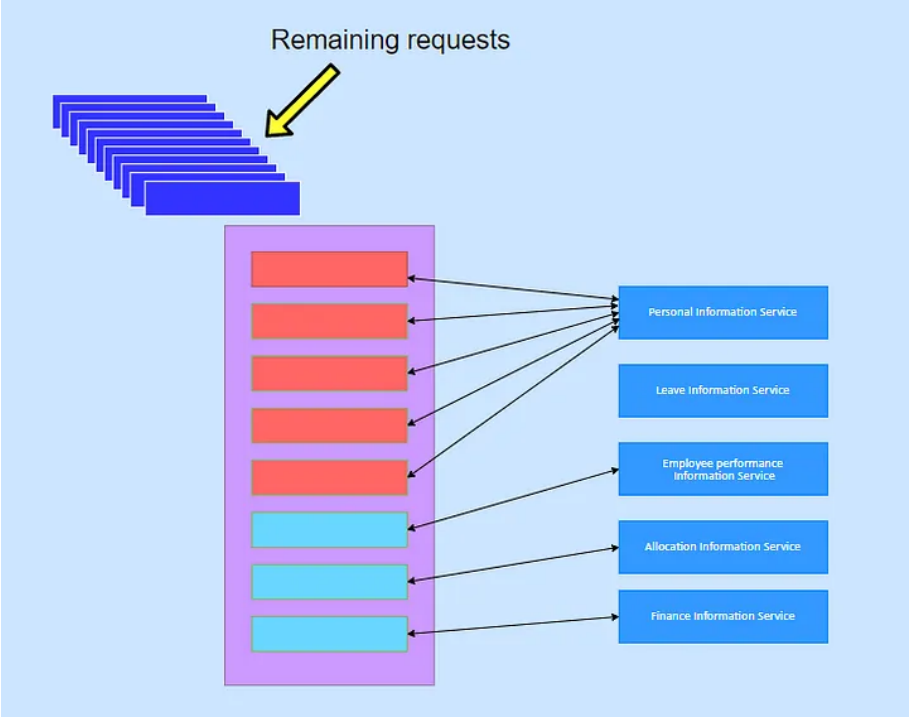



Figure 7

🌟 Then a few seconds later the failed service recovers back. Now the webserver tries to process all the requests in the queue. As a result, the webserver (or the proxy) may never recover. The reason is when the webserver processes queue request more and more requests reach continuously. So, this type of scenario will kill your service.

Now let's go to the next scenario

. . .

## Use case 2

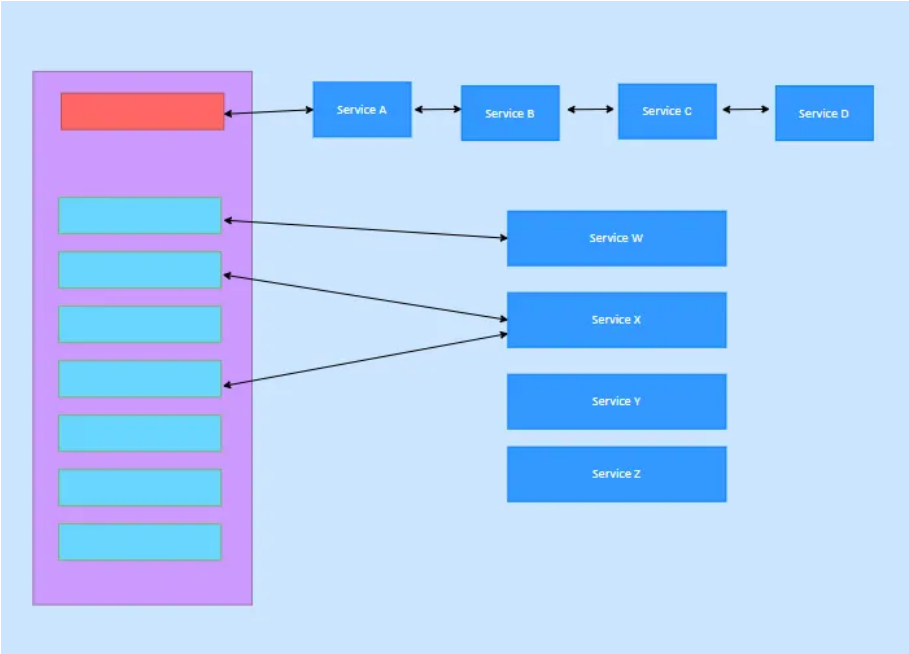



Figure 8

🌟 In this scenario, **service A** calls **B**, **service B** calls **C**, **service C** calls **D**. Meanwhile there are also other services **W**, **X**, **Y** and **Z**. But what would happen if a service could not respond on time ? 💥

🌟 Assume **service D** failed to respond on time.

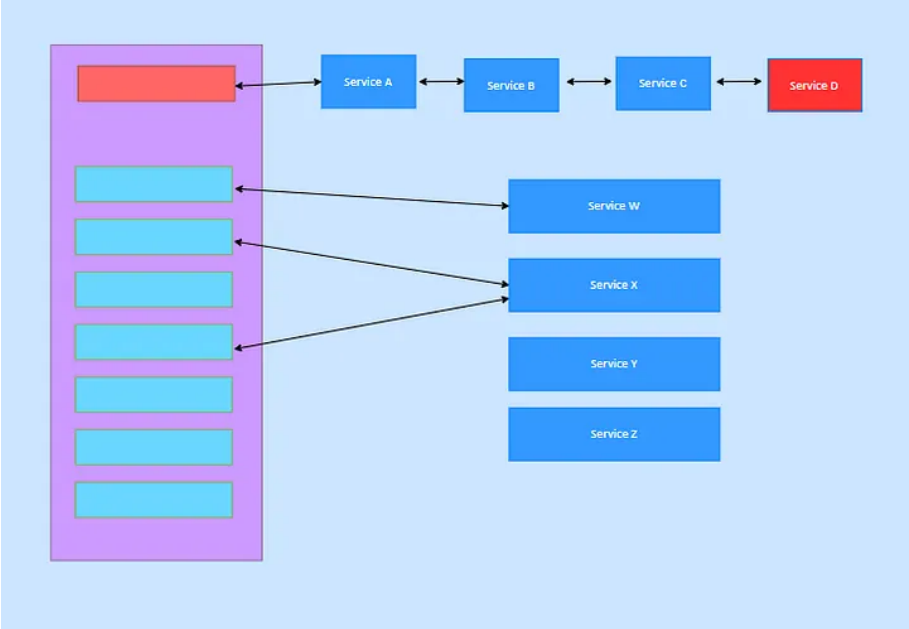



Figure 9 : Service D failed

🌟 Now the **service C** will have to wait.

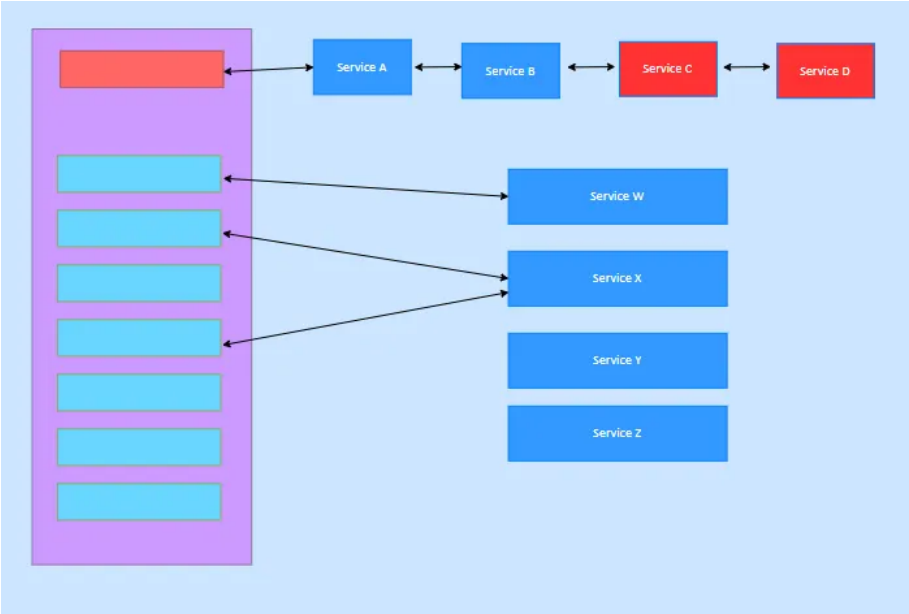



Figure 10: Service C is waiting

🌟 Since **service C** is waiting, **service B** also will have to wait. So accordingly, **service A** will have to be waiting. This can cause **cascade failure**.

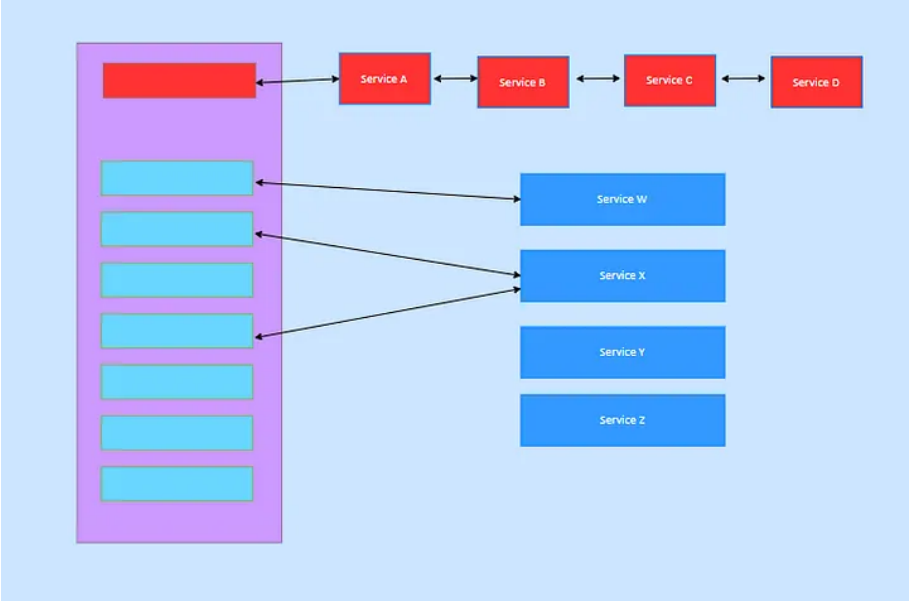



Figure 11 : Service A,B and C waiting for D

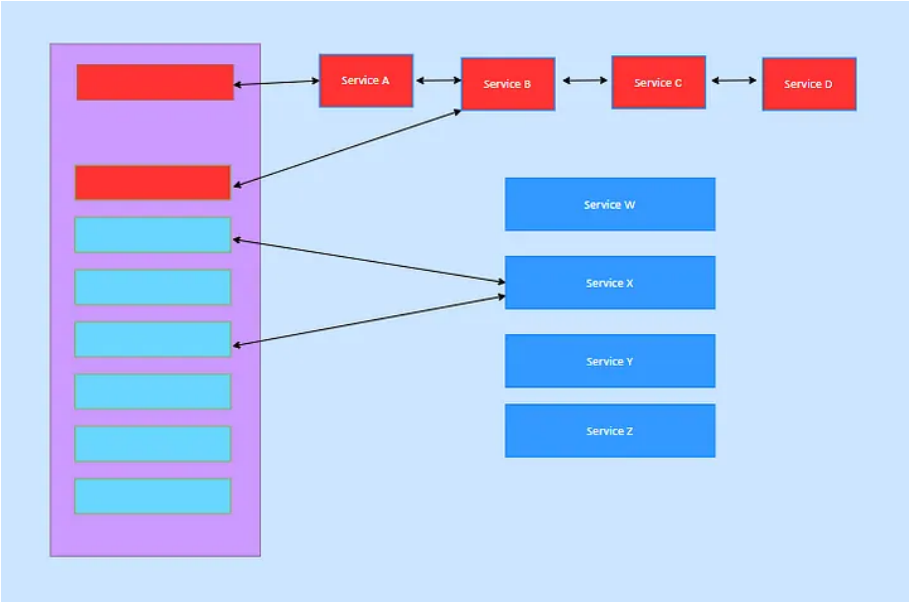



Figure 12 : Casacade faliure

🌟 No matter how your service failed but when the service failed it will go offline. So what can we do to keep these services up and running ? 🤔

. . .

## Solution with Circuit Breaker pattern

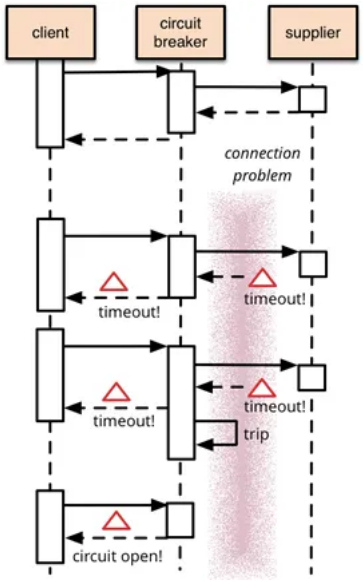



Figure 13: Circuit breaker pattern

🌟 So, as I mentioned above (in “Main use case for Circuit Breaker”) there are 04 services and a proxy (an aggregator pattern to call those services) in the system.

🌟 After going through the use cause 1 and 2, I’m sure now you know that system failures could happen when an individual service failed (**supplier become unresponsive**)— Figure 13.

🌟 Then it could lead the system to run out of critical resources (ex: remember how threads were occupied in use case 1).

🌟 The circuit breaker's basic concept is to wrap a protected function call in a circuit breaker object that monitors for failures. When the number of failures reaches a certain threshold, the circuit breaker trips, and all requests to the circuit breaker return with an error.

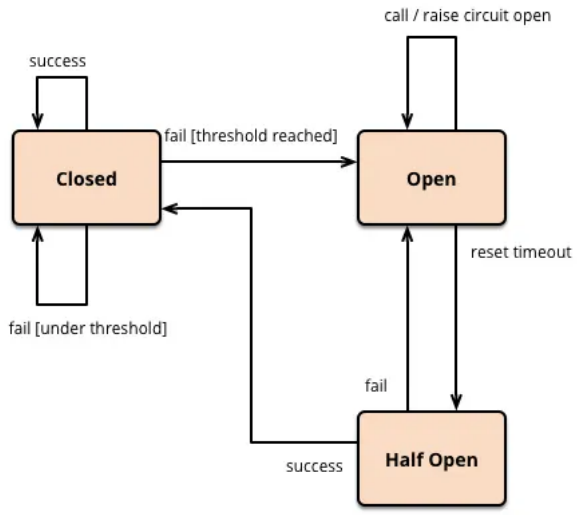



Figure 14: States of Circuit Breaker

The Circuit Breaker pattern has 3 states.

1. **Open.**
2. **Closed.**
3. **Half-Open.**

🌟 The circuit breaker works (pass requests through the service) normally when it is in the "**closed**" state. But when the failures exceed the threshold limit, the circuit breaker trips. As seen in the above diagram, this "**opens**" the circuit (state switches to "open").

🌟 When the circuit is "**open**" incoming requests will return with error without any attempt to execute the real operation

🌟 After a duration of time, the circuit breaker goes into the "**half-open**" state. In this state, the circuit breaker will allow a limited number of test requests to pass through and if the requests succeed, the circuit breaker resets and returns to the "**closed**" state and the traffic will go through as usual. If this request fails, the circuit breaker returns to the open state until another timeout.

> *Ex: Service A should respond within 200ms.*
>
> *0ms -100ms : expected delay interval.*

> *100ms -200ms : risky.*
>
> *If response time is more thsn 200ms then cut off the service.*
>
> *If you create a monitoring dashboard you can monitor the requests and their response times. Based on that you can decide threshold level.*

1

🌟 So how the circuit breaker pattern works is, if the number of requests (ex: 75% of requests) is reaching the upper threshold (150–200ms) that means the service is failing slowly.

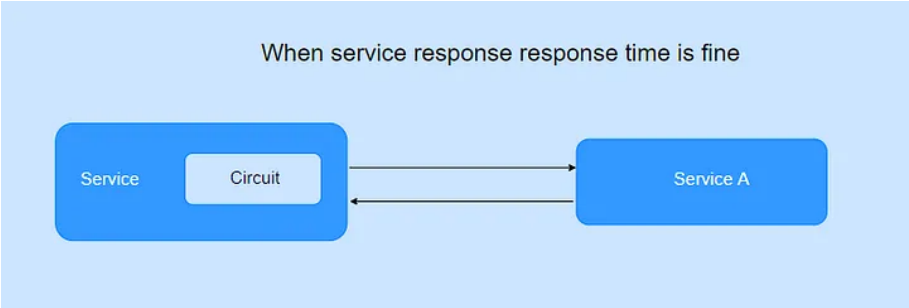



Figure 15

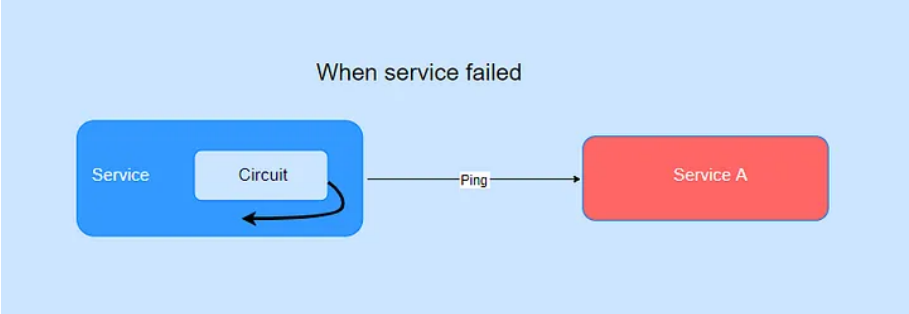



Figure 16

🌟 If the number of occurrences exceeds 200ms (the maximum threshold given for the service) the proxy will identify that this service is not responding anymore.

🌟 What it does next is the request which comes to access service A will failback. it breaks the connection between your proxy and service A.

🌟 Now your proxy will not go to service A. That means it will not wait.

Top highlight

. . .

**But why not directly go to the service and see. If it's failing the request can go back. Why is it necessary to have something in between? 🤔**

🌟 Assume you have a 30 seconds timeout. If each request is trying to reach service A, without considering it failing all the requests which come from the consumer will wait 30 seconds. End of the 30 seconds these requests will timeout.

🌟 During that 30 seconds, the remaining requests that come to consume A will try to reach service A and those are also going and wait in the queue.

🌟 So, what the circuit breaker pattern does is, if the service is failing more than the given threshold, it will not try to reach service A, instead it will fail back to the consumer saying service A is not available.

**So how is it going to connect back? 🤔**

🌟 In the background, it sends a ping request (Figure 16) or a default request to service A in a timely manner. So, when the response time comes back to the normal threshold it will turn on the request again.

🌟 So, the next requests that are reaching to consume service A will directly go to service A

> *During the failure time all the requests that come to consume service A are sent back with an error message. So now there is no queue. When the service back online it will be open for new traffic.*

. . .

**But didn't we fail to respond to some of the consumer requests? 🙄**

🌟 Yes, some requests were (were not queued) sent back to the consumer with an error message and they didn't get a response from service A.

🌟 But, what would have been the result if those requests try to reach service A (which failed) and those requests were queued?

🌟 The whole system could have failed due to the huge queue created behind the service. Because even if the service recovers back those queues will consume service A and eventually service A will fail.

🌟 But with this approach service, A might fail for some time duration and certain requests may not get a response from service A and will be returned with a error. But as soon as service A comes back online, the next coming traffic will be served. So a casacade faliure will not occur.

🌟 That's the principle behind Circuit Breaker Pattern.

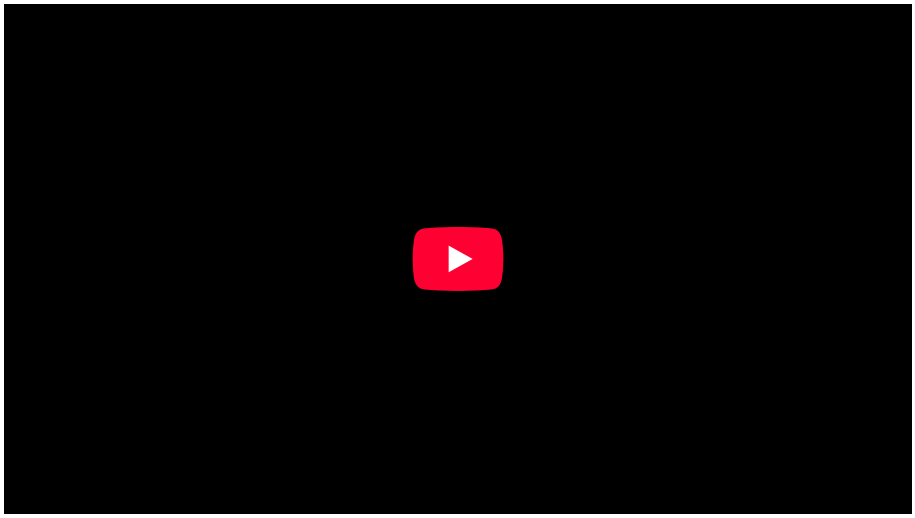
> *From the user's perspective, having to wait a long time for a response is not a good user experience. Rather than keeping the consumer waiting for a longer duration, it is better to respond quickly. It doesn't matter if it's a success or a failure; what counts is that the user isn't kept waiting.*

. . .

🌟 Well, I hope now you know about the Circuit Breaker pattern and how it



## References



**Release It!: Design and Deploy Production-Ready Software**

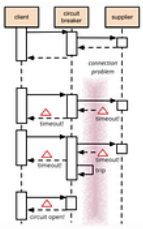
Release It!: Design and Deploy Production-Ready Software [Nygard, Michael T.] on Amazon.com. *FREE* shipping on...

www.amazon.com


**bliki: CircuitBreaker**

It's common for software systems to make remote calls to software running in different processes, probably on different...

martinfowler.com



**Microservices Design Patterns | Microservices Patterns | Edureka**

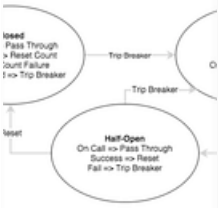
In today's market, Microservices have become the go-to solution, to build an application. They are known to solve...

www.edureka.co


**Circuit Breakers and Microservices Architecture | Constant Contact Tech Blog**

By now it's pretty well-known that a Microservices architecture has many advantages. These include low coupling...

techblog.constantcontact.com



Microservices | Design Patterns | Circuit Breaker | Microservices Pattern | Software Engineering

**Published in Geek Culture**

33K followers · Last published Sep 1, 2023

Follow

A new tech publication by Start it up (https://medium.com/swlh).

**Written by Hasitha Subhashana**

443 followers · 59 following

Follow

Software Engineer at CMS

## Responses (12)

I Ishu

What are your thoughts?

Stefan Nastic
Jun 19, 2021 (edited)

> The circuit breaker's basic concept is to wrap a protected function call in a circuit breaker object that monitors for failures. When the number of failures reaches a certain threshold,...

Thanks for a great article! I think it is also important to mention that when the circuit is tripped (open), a circuit breaker does not have to return an error. Instead, the circuit breaker can return a cached value or a default value when... more

18 Reply

Ross Bagley
Jun 14, 2021

So is the circuit breaker implemented in the client stub or in the load balancer or the network mesh provider or somewhere else entirely?

2 1 reply Reply

Dharmaraj Rathinavel
Feb 28

Very well explained. Thanks!

1 Reply

See all responses

## More from Hasitha Subhashana and Geek Culture

Hasitha Subhashana

### Understanding how Java Virtual Machine (JVM) works

Most of the developers learn JVM as a black box. You write the code. Then compile it and...

May 9, 2021 163 4

In Geek Culture by Shu Ishida

### Installing Linux (Ubuntu 20.04) on an external portable SSD and...

Carry your Ubuntu OS anywhere without being limited by hardware

Apr 22, 2022 379 5

In Geek Culture by Anshul Borawake

### React Native Generate APK—Debug and Release APK

Generate Debug and Release APK in React Native; Windows, iOS and Linux

Apr 3, 2021 1.8K 23

Hasitha Subhashana

### Introduction to Message Brokers

What is a message broker?

May 20, 2021 227 2

See all from Hasitha Subhashana | See all from Geek Culture

## Recommended from Medium

Priya Srivastava

### System Design Pattern : Scaling Writes: How Big Tech Handles...

Motivation :

6d ago 59

In ITNEXT by Animesh Gaitonde

### System Design: Building TikTok-Style Video Feed for 100 Million...

A deep dive into architecture, scalability, and real-time data delivery at scale

May 20 789 23

Sanyamdubey

### Why Choosing Golang for My Startup Was a Costly Lesson

When I founded my startup, I was thrilled about building a scalable, high-performance...

May 11 50

Indrajit

### WebFlux vs Virtual Threads: I Hit Both With 100K Requests—One...

100K requests against WebFlux and Virtual Threads. One dominated, one disappointed....

Aug 17 7 1

The Latency Gambler

### Scaling to 1 Million Users: The Architecture I Wish I Knew Sooner

When we launched, we were happy just having 100 daily users. But within months, w...

May 10 5.3K 114

Noah Byteforge

### How We Scaled from 10 to 10 Million Users with Zero Downtime...

When we crossed 10 million users without a single minute of downtime, people didn't...

Oct 13 35

See more recommendations