EP9: Retry patterns

ALEX XU
MAY 28, 2022

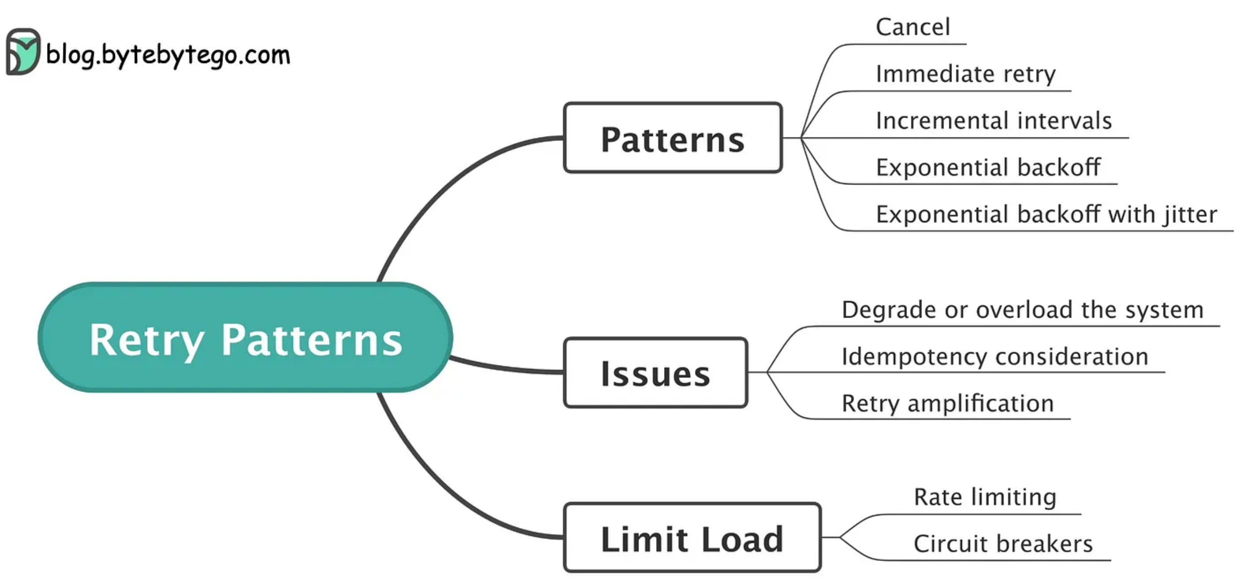
In this newsletter, we will talk about the following:

- Retry patterns
- Evolution of the Netflix API architecture
- Free ebook: Software Engineering at Google
- How does stop loss work?
- Chaos Engineering

Retry patterns

Some failures are transient, which makes it a good candidate to retry the request.

If an application detects a failure when it tries an operation, it can handle the failure using the following strategies:



- Cancel: the client can cancel the request.
- Immediate retry: client immediately resends a request.
- Fixed intervals: wait a fixed amount of time between the time of the failed payment and a new retry attempt.
- Incremental intervals: client waits for a short time for the first retry, and then incrementally increases the time for subsequent retries.
- Exponential backoff: double the waiting time between retries after each failed retry. For example, when a request fails for the first time, we retry after 1 second; if it fails a second time, we wait 2 seconds before the next retry; if it fails a third time, we wait 4 seconds before another retry.
- Exponential backoff with jitter. If all the failed calls back off at the same time, they cause contention or overload again when they retry. Jitter adds some amount of randomness to the backoff to spread the retries.

Issues

Retry is not perfect. It can cause issues such as overloading the system, executing the same operation multiple times, and amplifying a problem by creating a storm of requests.

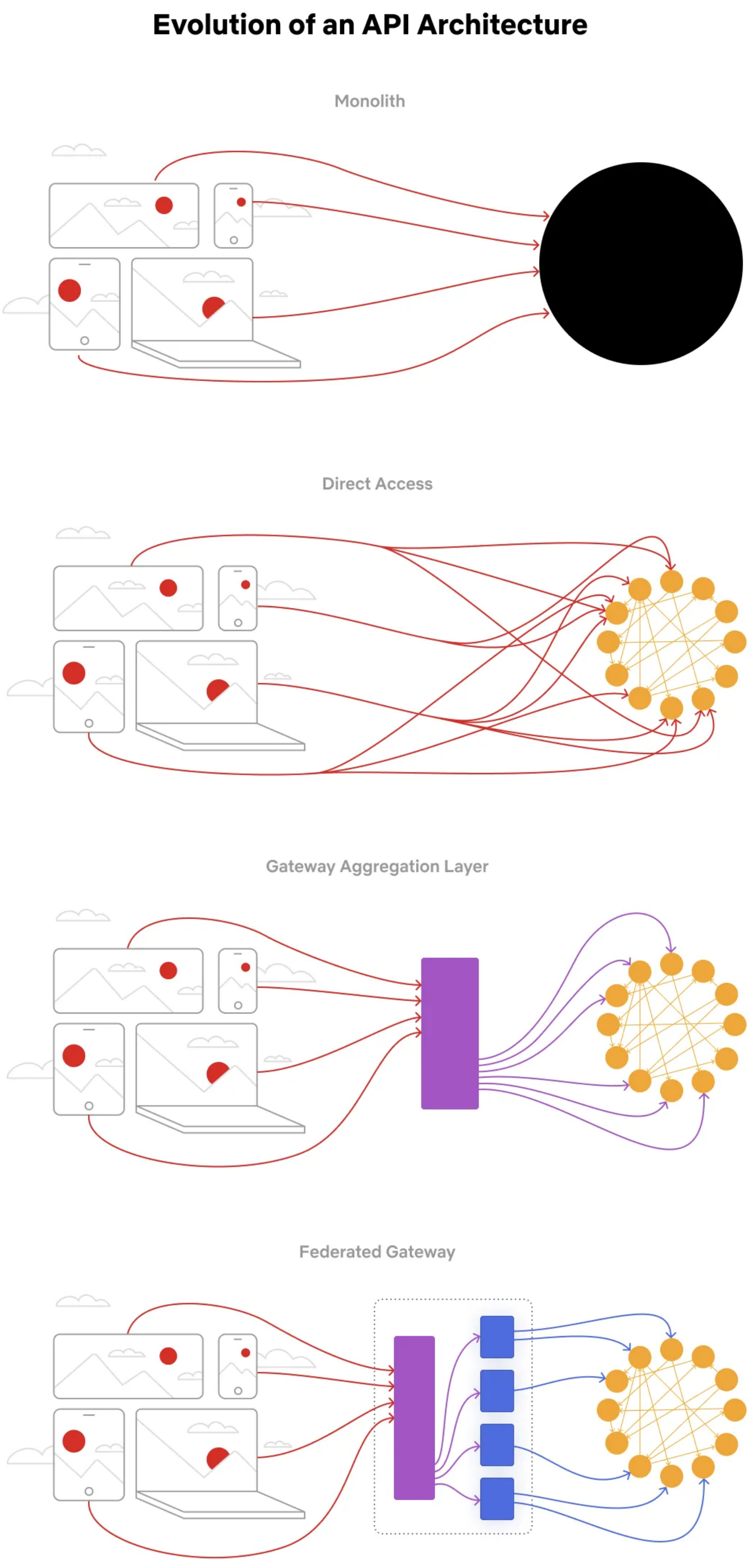
Rate limiting and circuit breakers patterns are commonly used to limit the load and avoid service overload.

This is just an overview of retry patterns. More to come later.

Evolution of the Netflix API architecture

Evolution of the Netflix API architecture.

The Netflix API architecture went through 4 main stages.


Evolution of an API Architecture
Monolith
Direct Access
Gateway Aggregation Layer
Federated Gateway

Monolith. The application is packaged and deployed as a monolith, such as a single Java WAR file, Rails app, etc. Most startups begin with a monolith architecture.

Direct access. In this architecture, a client app can make requests directly to the microservices. With hundreds or even thousands of microservices, exposing all of them to clients is not ideal.

Gateway aggregation layer. Some use cases may span multiple services, we need a gateway aggregation layer. Imagine the Netflix app needs 3 APIs (movie, production, talent) to render the frontend. The gateway aggregation layer makes it possible.

Federated gateway. As the number of developers grew and domain complexity increased, developing the API aggregation layer became increasingly harder. GraphQL federation allows Netflix to set up a single GraphQL gateway that fetches data from all the other APIs.

Over to you - why do you think Netflix uses GraphQL instead of RESTful?

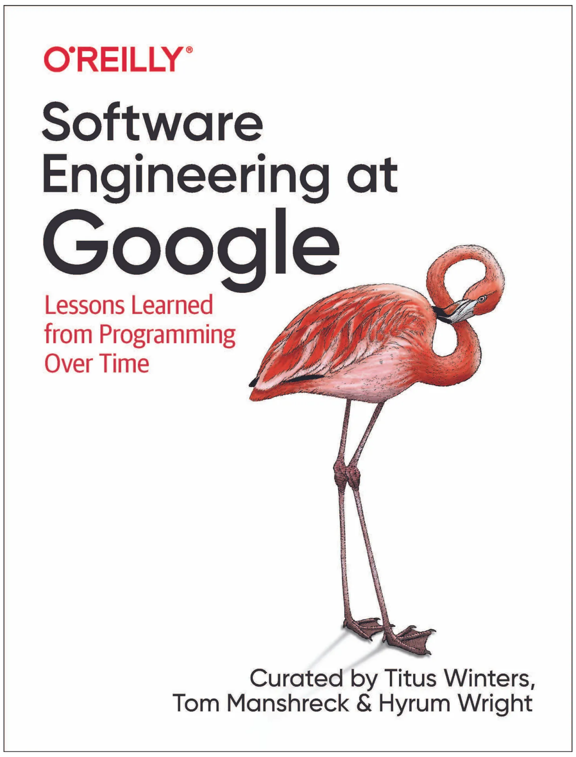
References:

[1] How Netflix Scales its API with GraphQL Federation (Part 1): https://netflixtechblog.com/how-netflix-scales-its-api-with-graphql-federation-part-1-ae3557c187e2 (image source)

[2] Why You Can't Talk About Microservices Without Mentioning Netflix: https://smartbear.com/blog/why-you-cant-talk-about-microservices-without-ment/

Free ebook: Software Engineering at Google

Free eBooks. The ebook “Software Engineering at Google: Lessons Learned from Programming Over Time” is free to read online now at: https://bit.ly/3wVhVgK


O'REILLY
Software Engineering at Google
Lessons Learned from Programming Over Time
Curated by Titus Winters, Tom Manshreck & Hyrum Wright

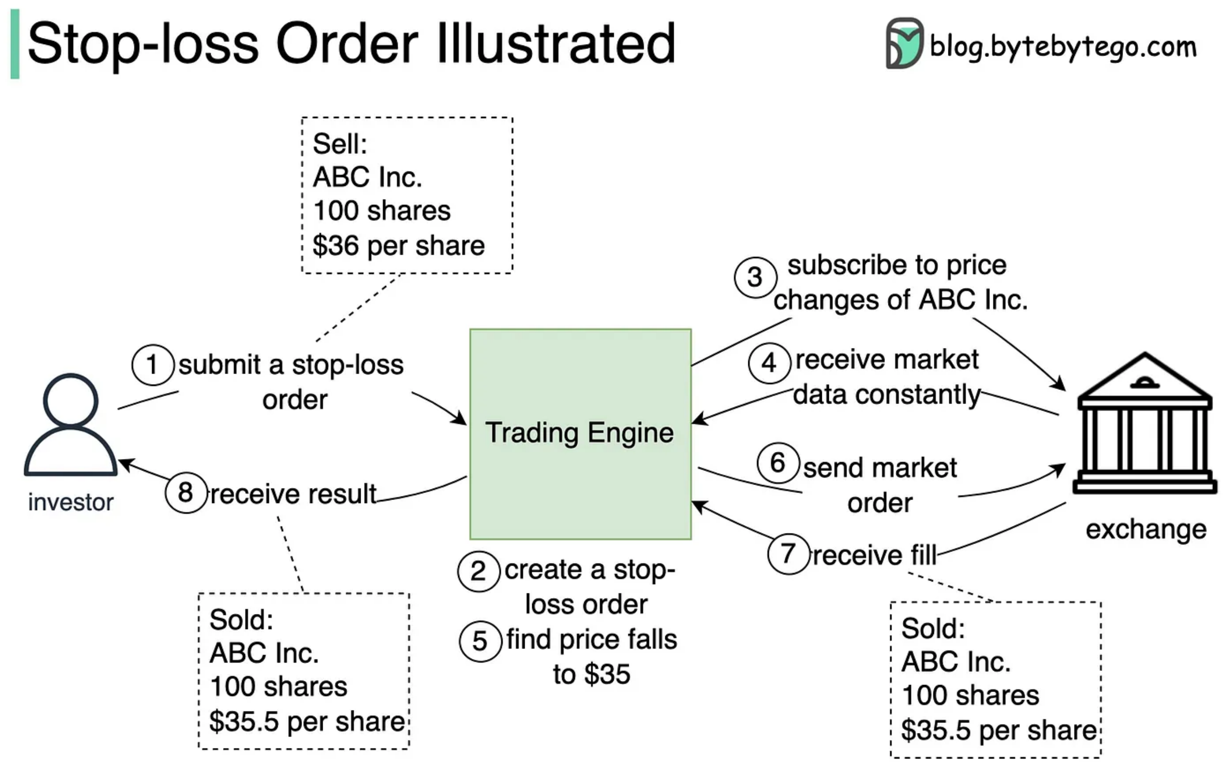
It mainly talks about:

- How to make your code resilient over time
- How software practices are affected by scale
- What trade-offs do engineers need to make during design and development

How does stop loss work?

What is a stop-loss order and how does it work?

A stop-loss order allows us to set a price called the ‘stop-loss price’ of a stock or a share. This is a value the investor chooses, at which they will sell it to minimize their loss on the investment.


Stop-loss Order Illustrated
blog.bytebytego.com
Sell: ABC Inc. 100 shares $36 per share
1 submit a stop-loss order
investor
8 receive result
Sold: ABC Inc. 100 shares $35.5 per share
Trading Engine
2 create a stop-loss order
5 find price falls to $35
3 subscribe to price changes of ABC Inc.
4 receive market data constantly
6 send market order
7 receive fill
exchange
Sold: ABC Inc. 100 shares $35.5 per share

When the price of the stock hits the stop-loss point, the stop-loss order is triggered and it turns into a market order to sell at the current market price.

For example, let's say an investor has 100 shares in ABC Inc., and the current price is $40 per share. The investor wants to sell the stock if the market price falls to or below $36, in order to limit their loss.

The diagram below illustrates how a stop-loss order is executed by a trading system.

1. The investor submits a stop-loss order to the trading system with 100 shares, to sell for $36.

2. Upon receiving the order request, the trading engine creates the stop-loss order.

3-4. The trading engine subscribes to the market data of ABC Inc. from the exchange and monitors its real-time market price.

5-6. If the trading engine detects that the market price of ABC Inc. falls to, say, $35, it immediately creates a market order and then submits it to the exchange to sell the 100 shares for the current best market price.

7. The order is filled (i.e. matched to the best buy orders in the market,) usually instantaneously. Then the trading engine receives from the exchange a ‘fill report’ stating the shares have been sold for, say, $35.5 per share.

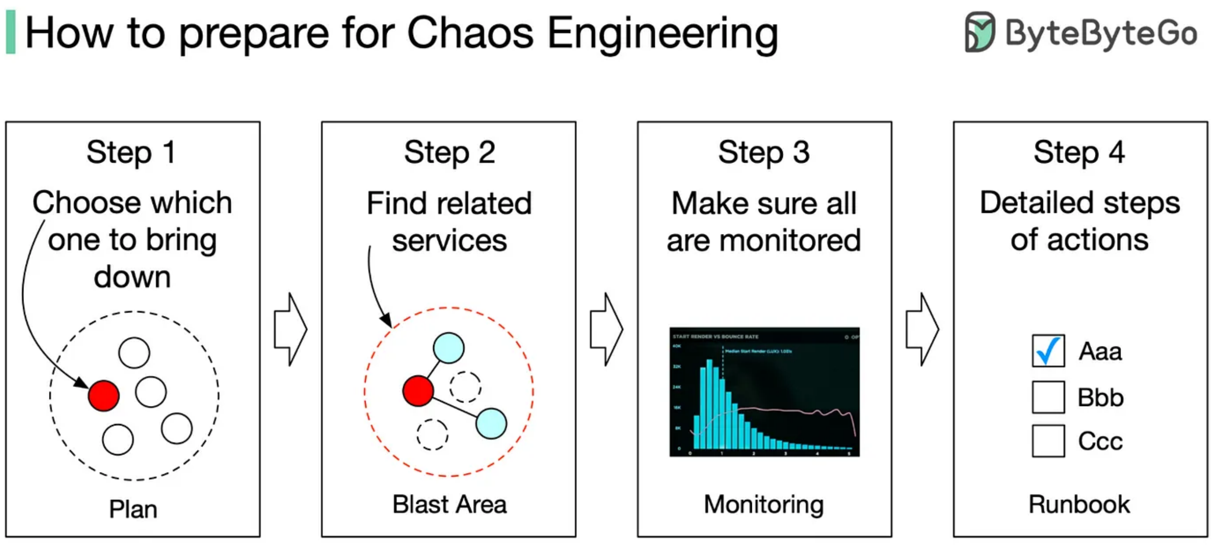
8. The trading system notifies the investor that the 100 shares have been sold for $35.5 per share.

Over to you: under extreme market conditions, where the stock price drops suddenly by 20%, what will happen to your stop-loss order?

Chaos Engineering

Why do Big Tech firms intentionally destroy their services in production?

The answer is they want to verify that their distributed system is as reliable as they have designed it to be. This methodology is called ‘Chaos Engineering,’ which was first used by Netflix about 12 years ago. How to try out chaos engineering in a system?


How to prepare for Chaos Engineering
ByteByteGo
Step 1
Choose which one to bring down
Plan
Step 2
Find related services
Blast Area
Step 3
Make sure all are monitored
Monitoring
Step 4
Detailed steps of actions
Aaa
Bbb
Ccc
Runbook

Obviously, some care must be taken when doing this. After all, we do not destroy our production services for fun, or to terrify our clients. We do chaos engineering to find pain points in the system; it is a service for our clients. To do this successfully, it’s helpful to prepare, like this:

1. Have a plan. In statistics terminology, we need to have a hypothesis for the behavior of the selected service that we want to bring down with chaos engineering.
2. Calculate blast radius. When a service is down, the failure may cascade to other services. So we need to have an idea of how wide the impact will be, known as the 'blast radius.'
3. Good monitoring. You need to double-check that the services within the blast radius have good monitoring, so we know how well the experiment proceeds, and whether the blast radius has widened.
4. Have a runbook. In this runbook, we document the steps we will take to bring down the service, the steps to bring it back up, and most importantly, the emergency plan to stop the experiment.

That's all set. You are good to go.

Over to you: some teams, such as QA and SRE, maybe against chaos engineering - sometimes for understandable reasons. How do you convince them that it's a valuable exercise?

Our Books:

152 Likes · 1 Restack

4 Comments

Write a comment...

Ishaan Sheikh May 28, 2022 Liked by Alex Xu

I am loving the newsletter so far. Great job

LIKE (4) REPLY SHARE

Jake Sonatore May 30, 2022 Liked by Alex Xu

Great to read about the evolution of Netflix's data aggregator layer.

LIKE (2) REPLY SHARE

2 more comments...