

# Flowchart of how slack decides to send a notification (Episode 3)



ALEX XU

APR 16, 2022



85



1



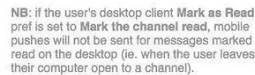
Share



In this newsletter, we will talk about the following:

- Flowchart of how slack decides to send a notification
- Orchestration and choreography
- How to design secure web API access for your website?
- How does Amazon build and operate software?

## Flowchart of how slack decides to send a notification



When we have a great design, users may not notice the complexity because it feels like the feature just working as intended.

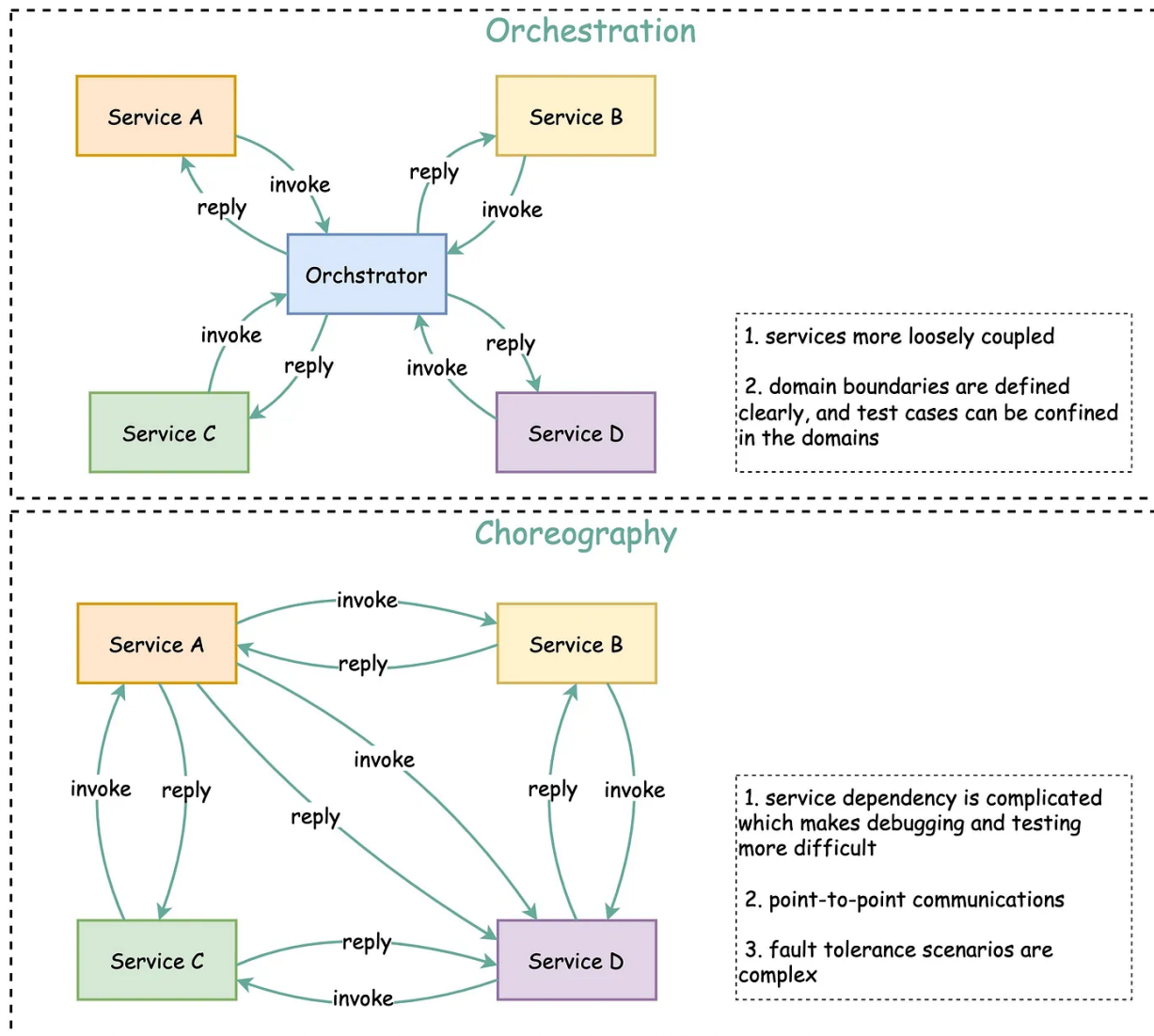
Image source: Slack Eng blog

## 2/10

There are two ways: orchestration and choreography

The diagram below illustrates the collaboration of microservices.

## Orchestration vs Choreography of Microservices



Choreography is like having a choreographer set all the rules. Then the dancers on stage (the microservices) interact according to them. Service choreography describes this exchange of messages and the rules by which the microservices interact.

Orchestration is different. The orchestrator acts as a center of authority. It is responsible for invoking and combining the services. It describes the interactions between all the participating services. It is just like a conductor leading the

musicians in a musical symphony. The orchestration pattern also includes the transaction management among different services.

The benefits of orchestration:

1. Reliability - orchestration has built-in transaction management and error handling, while choreography is point-to-point communications and the fault tolerance scenarios are much more complicated.
2. Scalability - when adding a new service into orchestration, only the orchestrator needs to modify the interaction rules, while in choreography all the interacting services need to be modified.

Some limitations of orchestration:

1. Performance - all the services talk via a centralized orchestrator, so latency is higher than it is with choreography. Also, the throughput is bound to the capacity of the orchestrator.
2. Single point of failure - if the orchestrator goes down, no services can talk to each other. To mitigate this, the orchestrator must be highly available.

Real-world use case: Netflix Conductor is a microservice orchestrator and you can read more details on the orchestrator design.

Question - Have you used orchestrator products in production? What are their pros & cons?

## **How to design secure web API access for your website?**

When we open web API access to users, we need to make sure each API call is authenticated. This means the user must be who they claim to be.

In this post, we explore two common ways:

1. Token based authentication
2. HMAC (Hash-based Message Authentication Code) authentication

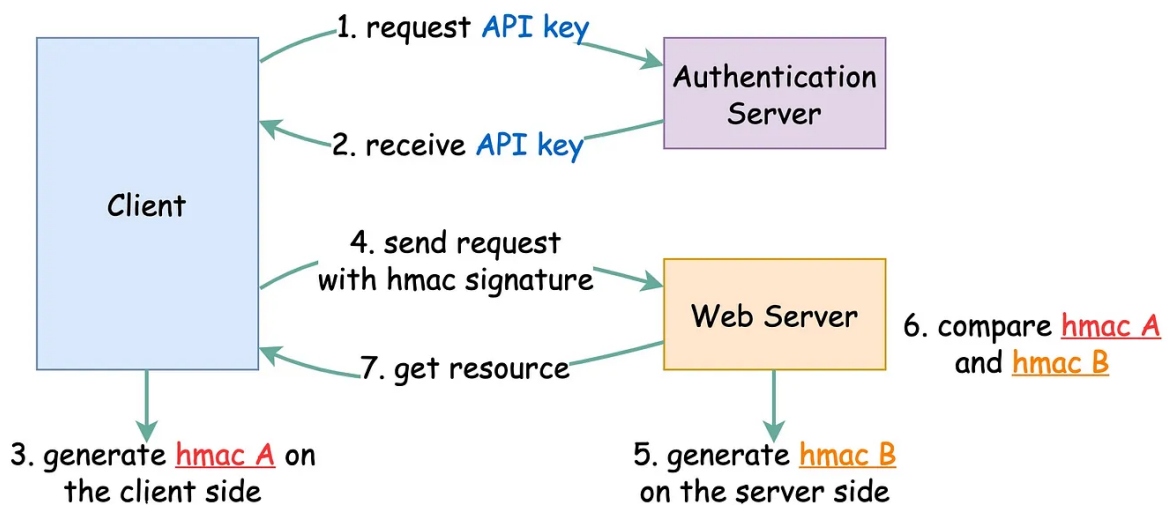
The diagram below illustrates how they work.

# How to Design Secure Web API?

## Token based authentication



## HMAC authentication



### hmac signature generation algorithm

public app ID  
request URI  
request content  
HTTP method  
request timestamp  
nonce

+ API key  
(private key) → HMAC signature

## Token based

Step 1 - the user enters their password into the client, and the client sends the password to the Authentication Server.

Step 2 - the Authentication Server authenticates the credentials and generates a token with an expiry time.

Steps 3 and 4 - now the client can send requests to access server resources with the token in the HTTP header. This access is valid until the token expires.

## HMAC based

This mechanism generates a Message Authentication Code (signature) by using a hash function (SHA256 or MD5).

Steps 1 and 2 - the server generates two keys, one is Public APP ID (public key) and the other one is API Key (private key).

Step 3 - we now generate a HMAC signature on the client side (hmac A). This signature is generated with a set of attributes listed in the diagram.

Step 4 - the client sends requests to access server resources with hmac A in the HTTP header.

Step 5 - the server receives the request which contains the request data and the authentication header. It extracts the necessary attributes from the request and uses the API key that's stored on the server side to generate a signature (hmac B.)

Steps 6 and 7 - the server compares hmac A (generated on the client side) and hmac B (generated on the server side). If they are matched, the requested resource will be returned to the client.

Question - How does HMAC authentication ensure data integrity? Why do we include "request timestamp" in HMAC signature generation?

## How does Amazon build and operate software?


In 2019, Amazon released The Amazon Builders' Library. It contains architecture-based articles that describe how Amazon architects, releases, and operates technology.

As of today, it published 26 articles. It took me two weekends to go through all the articles. I've had great fun and learned a lot. Here are some of my favorites:

- ◆ Making retries safe with idempotent APIs
- ◆ Timeouts, retries, and backoff with jitter
- ◆ Beyond five 9s: Lessons from our highest available data planes
- ◆ Caching challenges and strategies
- ◆ Ensuring rollback safety during deployments
- ◆ Going faster with continuous delivery
- ◆ Challenges with distributed systems
- ◆ Amazon's approach to high-availability deployment



LEVEL 400





### Workload isolation using shuffle-sharding

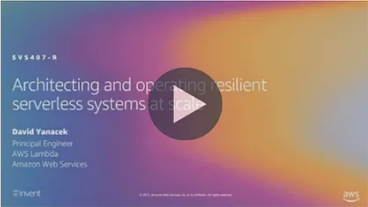
Author: Colm MacCarthaigh

Shuffle Sharding is one of our core techniques for drastically limiting the scope of impact of operational issues.

[PDF](#) | [Kindle](#)


LEVEL 400




### Architecting and operating resilient serverless...

Author: David Yanacek

In this video, we cover what AWS does to build reliable and resilient services, including avoiding modes and overload, performing bounded work, throttling at multiple layers, guarding concurrency,



LEVEL 300





### Caching challenges and strategies

Authors: Matt Brinkley, Jas Chhabra

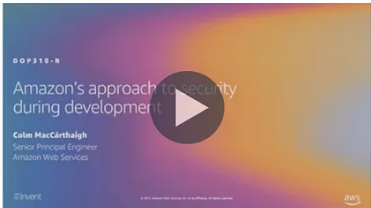
Improving latency and availability with caching while avoiding the modal behavior they can introduce.

[PDF](#) | [Kindle](#)

ARCHITECTURE



LEVEL 300



### Amazon's approach to security during...


Author: Colm MacCarthaigh

In this video, learn about how AWS teams both minimize security risks in

ARCHITECTURE


LEVEL 400



### Avoiding insurmountable queue backlogs


Author: David Yanacek

Prioritizing draining important



SOFTWARE DELIVERY AND OPERATIONS



LEVEL 400



### Implementing health checks

Author: David Yanacek


Automatically detecting and mitigating






85 Likes

1 Comment






Write a comment...






L30A Apr 27, 2022

Please where can finding the others (episode 2 and 1)

 LIKE  REPLY  SHARE



---

© 2023 ByteByteGo · [Privacy](#) · [Terms](#) · [Collection notice](#)  
[Substack](#) is the home for great writing