

2023 EDITION

FREE

SYSTEM DESIGN

THE BIG ARCHIVE



Explaining 9 types of API testing	7
How is data sent over the internet? What does that have to do with the OSI model? How does TCP/IP fit into this?	10
Top 5 common ways to improve API performance	11
There are over 1,000 engineering blogs. Here are my top 9 favorites:	15
REST API Authentication Methods	16
Linux Boot Process Illustrated	18
Netflix's Tech Stack	22
What does ACID mean?	26
Oauth 2.0 Explained With Simple Terms	28
The Evolving Landscape of API Protocols in 2023	30
Linux boot Process Explained	32
Explaining 8 Popular Network Protocols in 1 Diagram.	34
Data Pipelines Overview	36
CAP, BASE, SOLID, KISS, What do these acronyms mean?	38
GET, POST, PUT... Common HTTP “verbs” in one figure	40
How Do C++, Java, Python Work?	42
Top 12 Tips for API Security	44
Our recommended materials to crack your next tech interview	45
A handy cheat sheet for the most popular cloud services (2023 edition)	49
Best ways to test system functionality	51
Explaining JSON Web Token (JWT) to a 10 year old Kid	53
How do companies ship code to production?	55
How does Docker Work? Is Docker still relevant?	57
Explaining 8 Popular Network Protocols in 1 Diagram	59
System Design Blueprint: The Ultimate Guide	61
Key Concepts to Understand Database Sharding	63
Top 5 Software Architectural Patterns	67
OAuth 2.0 Flows	69
How did AWS grow from just a few services in 2006 to over 200 fully-featured services?	71
HTTPS, SSL Handshake, and Data Encryption Explained to Kids	75
A nice cheat sheet of different databases in cloud services	77
CI/CD Pipeline Explained in Simple Terms	78
What does API gateway do?	80
The Code Review Pyramid	82
A picture is worth a thousand words: 9 best practices for developing microservices	83

What are the greenest programming languages?	85
An amazing illustration of how to build a resilient three-tier architecture on AWS	87
URL, URI, URN - Do you know the differences?	88
What branching strategies does your team use?	90
Linux file system explained	90
What are the data structures used in daily life?	95
18 Most-used Linux Commands You Should Know	99
Would it be nice if the code we wrote automatically turned into architecture diagrams?	101
Netflix Tech Stack - Part 1 (CI/CD Pipeline)	103
18 Key Design Patterns Every Developer Should Know	105
How many API architecture styles do you know?	107
Visualizing a SQL query	109
What distinguishes MVC, MVP, MVVM, MVVM-C, and VIPER architecture patterns from each other?	111
Almost every software engineer has used Git before, but only a handful know how it works :)	113
I read something unbelievable today: Levels. fyi scaled to millions of users using Google Sheets as a backend!	115
Best ways to test system functionality	117
Logging, tracing and metrics are 3 pillars of system observability	119
Internet Traffic Routing Policies	121
Subjects that should be mandatory in schools	123
Do you know all the components of a URL?	124
What are the differences between cookies and sessions?	125
How do DevOps, NoOps change the software development lifecycle (SDLC)?	127
Popular interview question: What is the difference between Process and Thread?	129
Top 6 Load Balancing Algorithms	131
Symmetric encryption vs asymmetric encryption	133
How does Redis persist data?	135
IBM MQ -> RabbitMQ -> Kafka ->Pulsar, How do message queue architectures evolve?	137
Top 4 Kubernetes Service Types in one diagram	139
Explaining 5 unique ID generators in distributed systems	141
How Do C++, Java, and Python Function?	143
How will you design the Stack Overflow website?	145
Explain the Top 6 Use Cases of Object Stores	147
API Vs SDK!	149
A picture is worth a thousand words: 9 best practices for developing microservices	151

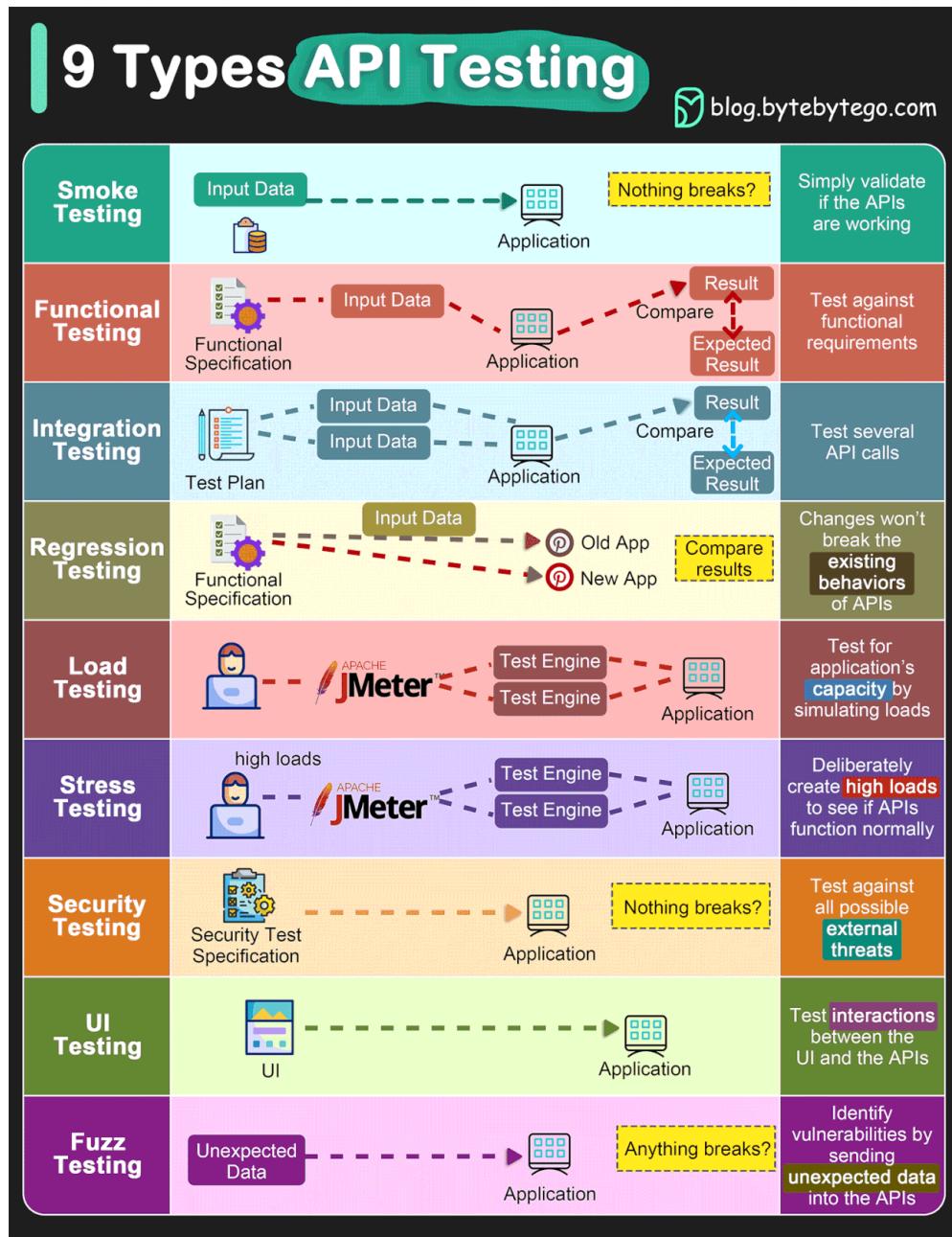
Proxy Vs reverse proxy	152
Git Vs Github	153
Which latency numbers should you know	154
Eight Data Structures That Power Your Databases. Which one should we pick?	156
How Git Commands Work	158
How to store passwords safely in the database and how to validate a password?	160
How does Docker Work? Is Docker still relevant?	164
Docker vs. Kubernetes. Which one should we use?	166
Writing Code that Runs on All Platforms	168
HTTP Status Code You Should Know	170
Docker 101: Streamlining App Deployment	172
Git Merge vs. Rebase vs. Squash Commit	174
Cloud Network Components Cheat Sheet	176
SOAP vs REST vs GraphQL vs RPC	178
10 Key Data Structures We Use Every Day	179
What does a typical microservice architecture look like?	181
My recommended materials for cracking your next technical interview	183
Uber Tech Stack	185
Top 5 Caching Strategies	187
How many message queues do you know?	189
Why is Kafka fast?	190
How slack decides to send a notification	192
Kubernetes Tools Ecosystem	193
Cloud Native Landscape	195
How does VISA work when we swipe a credit card at a merchant's shop?	196
A simple visual guide to help people understand the key considerations when designing or using caching systems	198
What tech stack is commonly used for microservices?	199
How do we transform a system to be Cloud Native?	201
Explaining Sessions, Tokens, JWT, SSO, and OAuth in One Diagram	203
Most Used Linux Commands Map	204
What is Event Sourcing? How is it different from normal CRUD design?	205
What is k8s (Kubernetes)?	207
How does Git Work?	209
How does Google Authenticator (or other types of 2-factor authenticators) work?	211
IaaS, PaaS, Cloud Native... How do we get here?	214

How does ChatGPT work?	215
Top Hidden Costs of Cloud Providers	217
Algorithms You Should Know Before You Take System Design Interviews	219
Understanding Database Types	221
How does gRPC work?	222
How does a Password Manager such as 1Password or Lastpass work? How does it keep our passwords safe?	224
Types of Software Engineers and Their Typically Required Skills	226
How does REST API work?	228
Session, cookie, JWT, token, SSO, and OAuth 2.0 - what are they?	229
Linux commands illustrated on one page!	232
The Payments Ecosystem	233
Algorithms You Should Know Before You Take System Design Interviews (updated list)	235
How is data transmitted between applications?	236
Cloud Native Anti Patterns	240
Uber Tech Stack - CI/CD	242
How Discord Stores Trillions Of Messages	244
How to diagnose a mysterious process that's taking too much CPU, memory, IO, etc?	246
How does Chrome work?	247
Differences in Event SOuring System Design	249
Firewall explained to Kids... and Adults	251
Paradigm Shift: How Developer to Tester Ratio Changed From 1:1 to 100:1	253
Why is PostgreSQL voted as the most loved database by developers?	255
8 Key OOP Concepts Every Developer Should Know	257
Top 6 most commonly used Server Types	259
DevOps vs. SRE vs. Platform Engineering. Do you know the differences?	261
5 important components of Linux	263
How to scale a website to support millions of users?	265
What is FedNow (instant payment)	267
5 ways of Inter-Process Communication	270
What is a webhook?	272
What tools does your team use to ship code to production and ensure code quality?	274
Stack Overflow's Architecture: A Very Interesting Case Study	276
Are you familiar with the Java Collection Framework?	277
Twitter 1.0 Tech Stack	279
Linux file permission illustrated	281

What are the differences between a data warehouse and a data lake?	282
10 principles for building resilient payment systems (by Shopify).	284
Kubernetes Periodic Table	286
Evolution of the Netflix API Architecture	287
Where do we cache data?	289
Top 7 Most-Used Distributed System Patterns ↓	291
How much storage could one purchase with the price of a Tesla Model S? ↓	292
How to choose between RPC and RESTful?	293
Netflix Tech Stack - Databases	294
The 10 Algorithms That Dominate Our World	296
What is the difference between “pull” and “push” payments?	298
ChatGPT - timeline	300
Why did Amazon Prime Video monitoring move from serverless to monolithic? How can it save 90% cost?	302
What is the journey of a Slack message?	303
How does GraphQL work in the real world?	305
Important Things About HTTP Headers You May Not Know!	307
Think you know everything about McDonald's? What about its event-driven architecture ?	308
How ChatGPT works technically	310
Choosing the right database is probably the most important technical decision a company will make.	311
How do you become a full-stack developer?	312
What's New in GPT-4	314
Backend Burger	315
How do we design effective and safe APIs?	316
Which SQL statements are most commonly used?	317
Two common data processing models: Batch v.s. Stream Processing. What are the differences?	
318	
Top 10 Architecture Characteristics / Non-Functional Requirements with Cheatsheet	320
Are serverless databases the future? How do serverless databases differ from traditional cloud databases?	321
Why do we need message brokers?	323
How does Twitter recommend “For You” Timeline in 1.5 seconds?	325
Popular interview question: what happens when you type “ssh hostname”?	327
Discover Amazon's innovative build system - Brazil.	329
Possible Experiment Platform Architecture	331
YouTube handles 500+ hours of video content uploads every minute on average. How does it	

<u>manage this?</u>	333
<u>A beginner's guide to CDN (Content Delivery Network)</u>	335
<u>What are the API architectural styles?</u>	337
<u>Cloud-native vs. Cloud computing</u>	339
<u>C, C++, Java, Javascript, Typescript, Golang, Rust...</u>	341
<u>The Linux Storage Stack Diagram shows the layout of the the Linux storage stack</u>	343
<u>Breaking down what's going on with the Silicon Valley Bank (SVB) collapse</u>	344

Explaining 9 types of API testing



- ◆ **Smoke Testing**

This is done after API development is complete. Simply validate if the APIs are working and nothing breaks.

- ◆ **Functional Testing**

This creates a test plan based on the functional requirements and compares the results with the expected results.

- ◆ **Integration Testing**

This test combines several API calls to perform end-to-end tests. The intra-service communications and data transmissions are tested.

- ◆ **Regression Testing**

This test ensures that bug fixes or new features shouldn't break the existing behaviors of APIs.

- ◆ **Load Testing**

This tests applications' performance by simulating different loads. Then we can calculate the capacity of the application.

- ◆ **Stress Testing**

We deliberately create high loads to the APIs and test if the APIs are able to function normally.

- ◆ **Security Testing**

This tests the APIs against all possible external threats.

- ◆ **UI Testing**

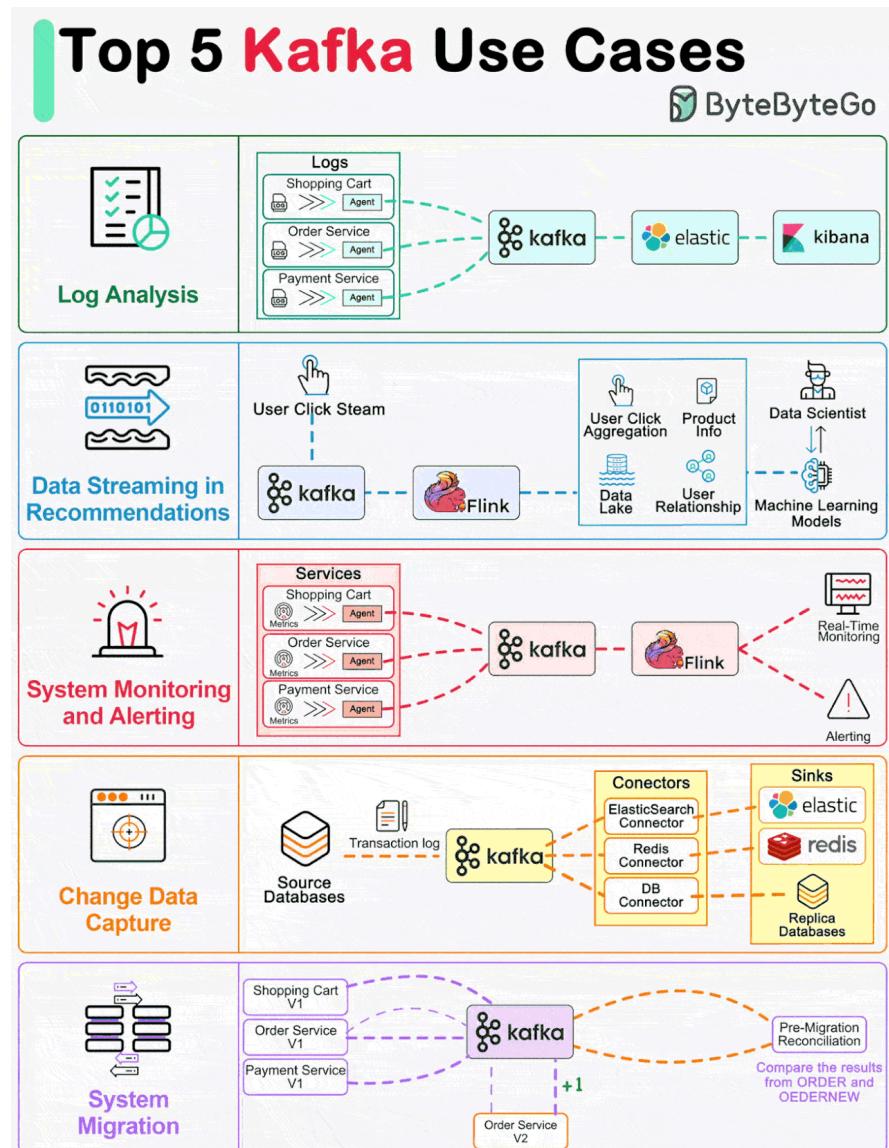
This tests the UI interactions with the APIs to make sure the data can be displayed properly.

- ◆ **Fuzz Testing**

This injects invalid or unexpected input data into the API and tries to crash the API. In this way, it identifies the API vulnerabilities.

Top 5 Kafka use cases

Kafka was originally built for massive log processing. It retains messages until expiration and lets consumers pull messages at their own pace.

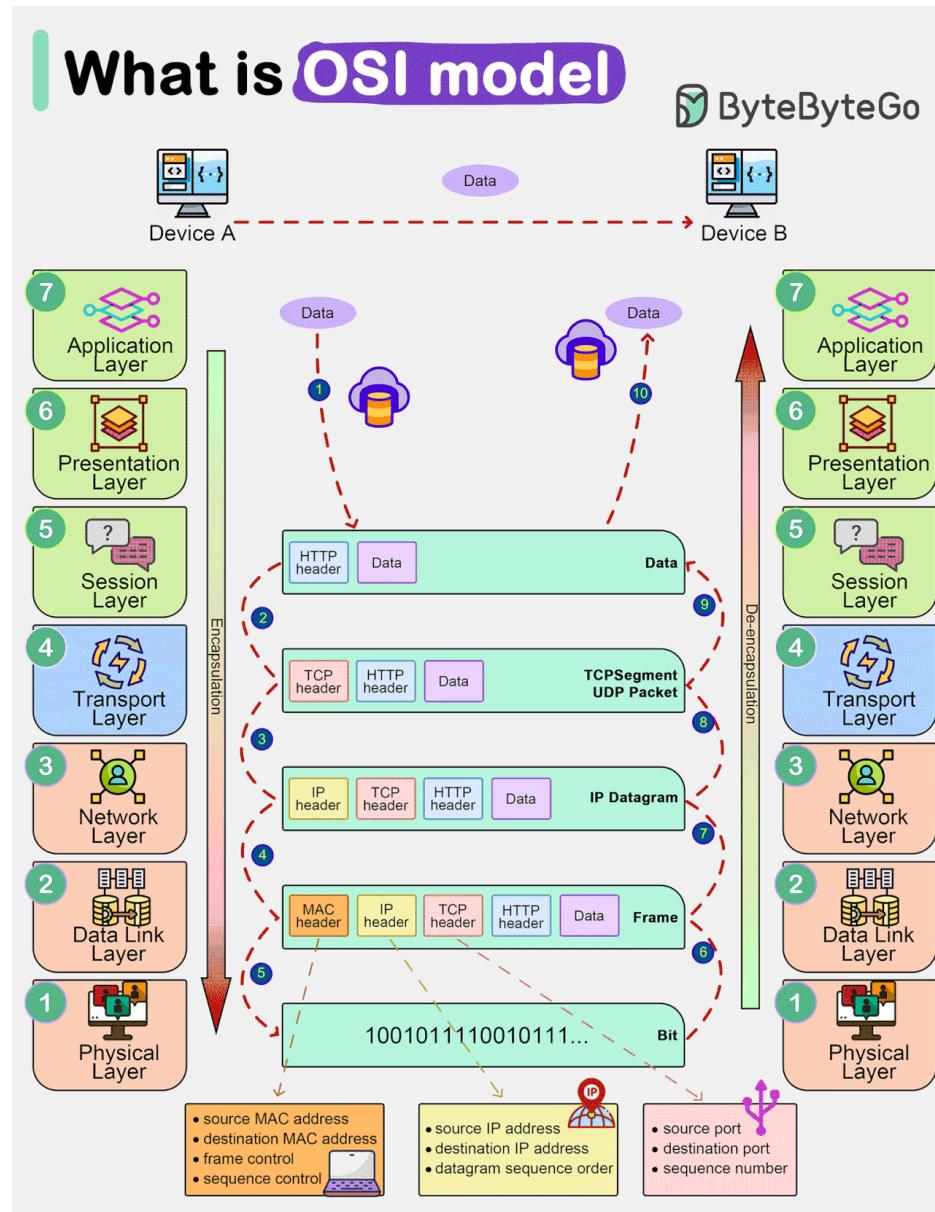


Let's review the popular Kafka use cases.

- Log processing and analysis
- Data streaming in recommendations
- System monitoring and alerting
- CDC (Change data capture)
- System migration

Over to you: Do you have any other Kafka use cases to share?

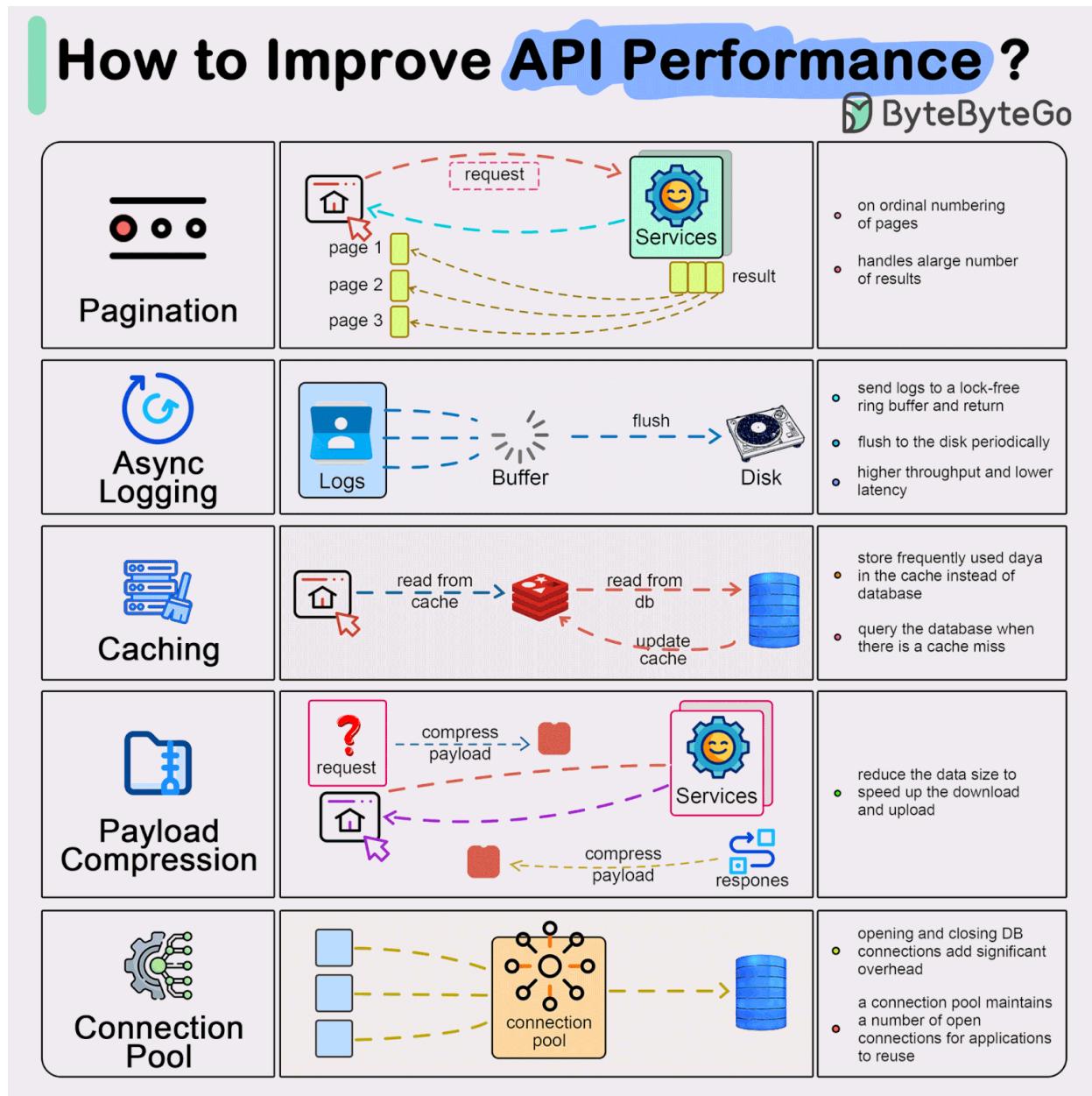
How is data sent over the internet? What does that have to do with the OSI model? How does TCP/IP fit into this?



7 Layers in the OSI model are:

1. Physical Layer
2. Data Link Layer
3. Network Layer
4. Transport Layer
5. Session Layer
6. Presentation Layer
7. Application Layer

Top 5 common ways to improve API performance



Result Pagination:

This method is used to optimize large result sets by streaming them back to the client, enhancing service responsiveness and user experience.

Asynchronous Logging:

This approach involves sending logs to a lock-free buffer and returning immediately, rather than dealing with the disk on every call. Logs are periodically flushed to the disk, significantly reducing I/O overhead.

Data Caching:

Frequently accessed data can be stored in a cache to speed up retrieval. Clients check the cache before querying the database, with data storage solutions like Redis offering faster access due to in-memory storage.

Payload Compression:

To reduce data transmission time, requests and responses can be compressed (e.g., using gzip), making the upload and download processes quicker.

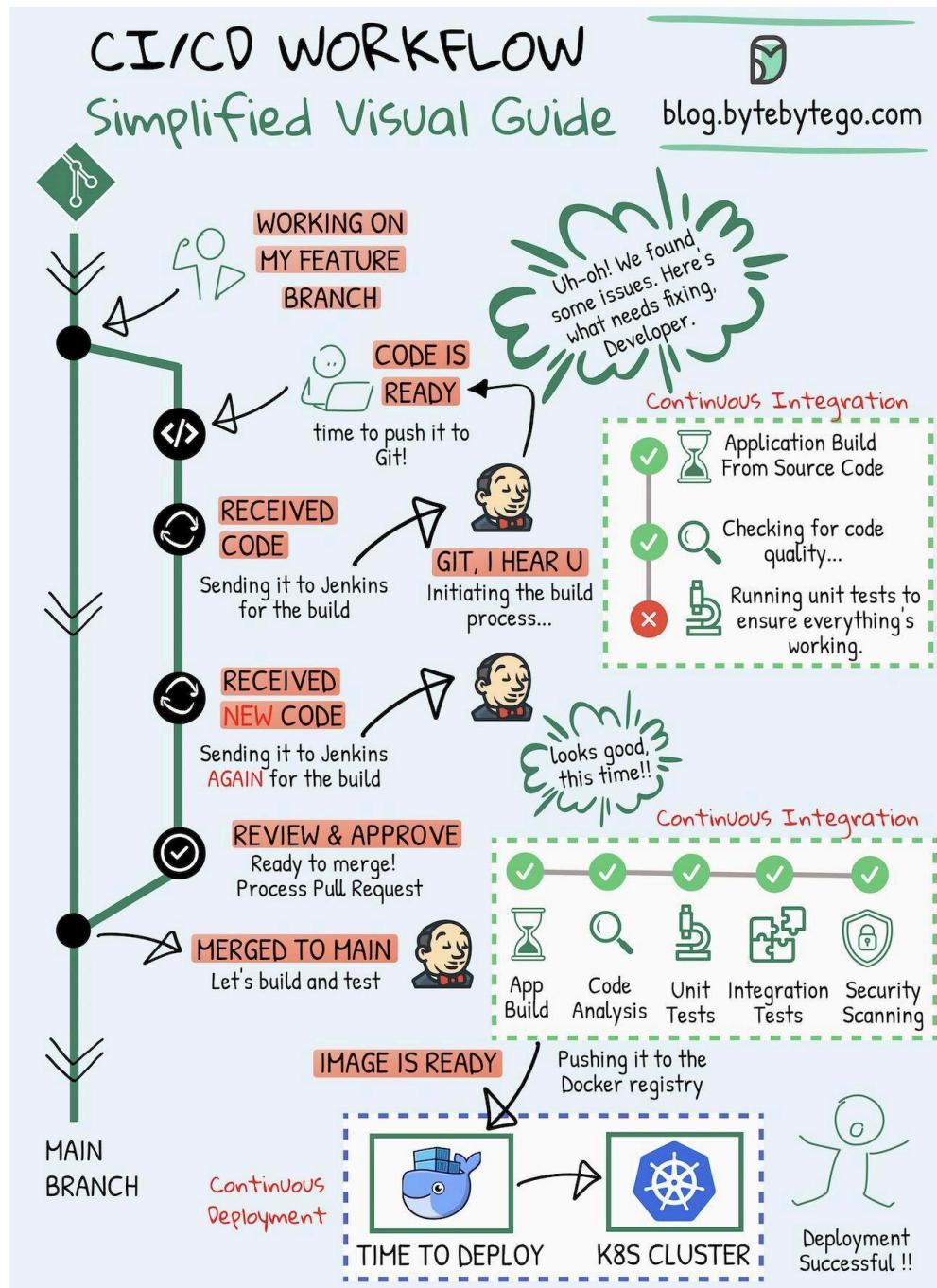
Connection Pooling:

This technique involves using a pool of open connections to manage database interaction, which reduces the overhead associated with opening and closing connections each time data needs to be loaded. The pool manages the lifecycle of connections for efficient resource use.

Over to you: What other ways do you use to improve API performance?

CI/CD Simplified Visual Guide

Whether you're a developer, a DevOps specialist, a tester, or involved in any modern IT role, CI/CD pipelines have become an integral part of the software development process.



Continuous Integration (CI) is a practice where code changes are frequently combined into a shared repository. This process includes automatic checks to ensure the new code works well.

with the existing code.

Continuous Deployment (CD) takes care of automatically putting these code changes into real-world use. It makes sure that the process of moving new code to production is smooth and reliable.

This visual guide is designed to help you grasp and enhance your methods for creating and delivering software more effectively.

Over to you: Which tools or strategies do you find most effective in implementing CI/CD in your projects?

There are over 1,000 engineering blogs. Here are my top 9 favorites:

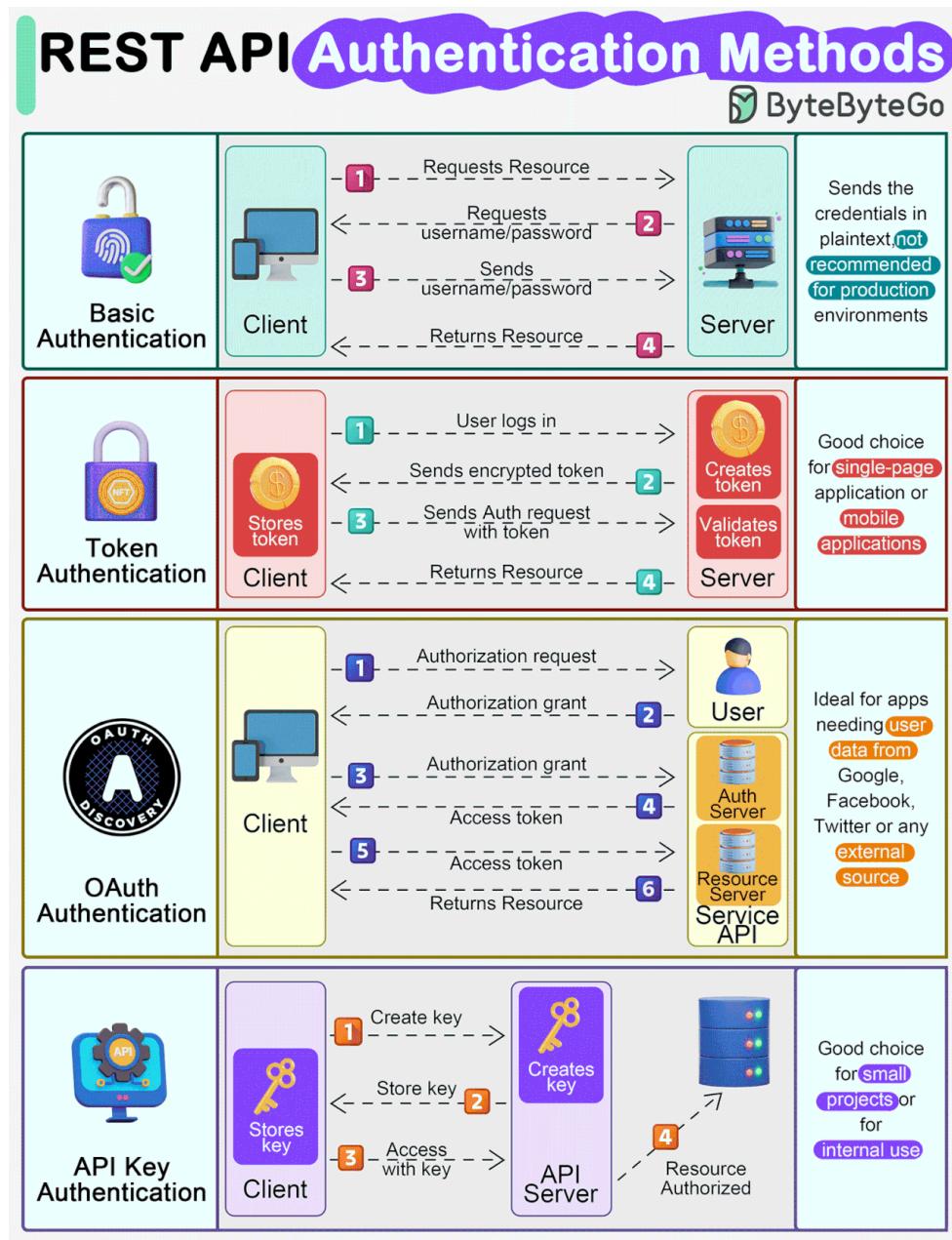


- Netflix TechBlog
- Uber Blog
- Cloudflare Blog
- Engineering at Meta
- LinkedIn Engineering
- Discord Blog
- AWS Architecture
- Slack Engineering
- Stripe Blog

Over to you - What are some of your favorite engineering blogs?

REST API Authentication Methods

Authentication in REST APIs acts as the crucial gateway, ensuring that solely authorized users or applications gain access to the API's resources.



Some popular authentication methods for REST APIs include:

1. Basic Authentication:

Involves sending a username and password with each request, but can be less secure without encryption.

When to use:

Suitable for simple applications where security and encryption aren't the primary concern or when used over secured connections.

2. Token Authentication:

Uses generated tokens, like JSON Web Tokens (JWT), exchanged between client and server, offering enhanced security without sending login credentials with each request.

When to use:

Ideal for more secure and scalable systems, especially when avoiding sending login credentials with each request is a priority.

3. OAuth Authentication:

Enables third-party limited access to user resources without revealing credentials by issuing access tokens after user authentication.

When to use:

Ideal for scenarios requiring controlled access to user resources by third-party applications or services.

4. API Key Authentication:

Assigns unique keys to users or applications, sent in headers or parameters; while simple, it might lack the security features of token-based or OAuth methods.

When to use:

Convenient for straightforward access control in less sensitive environments or for granting access to certain functionalities without the need for user-specific permissions.

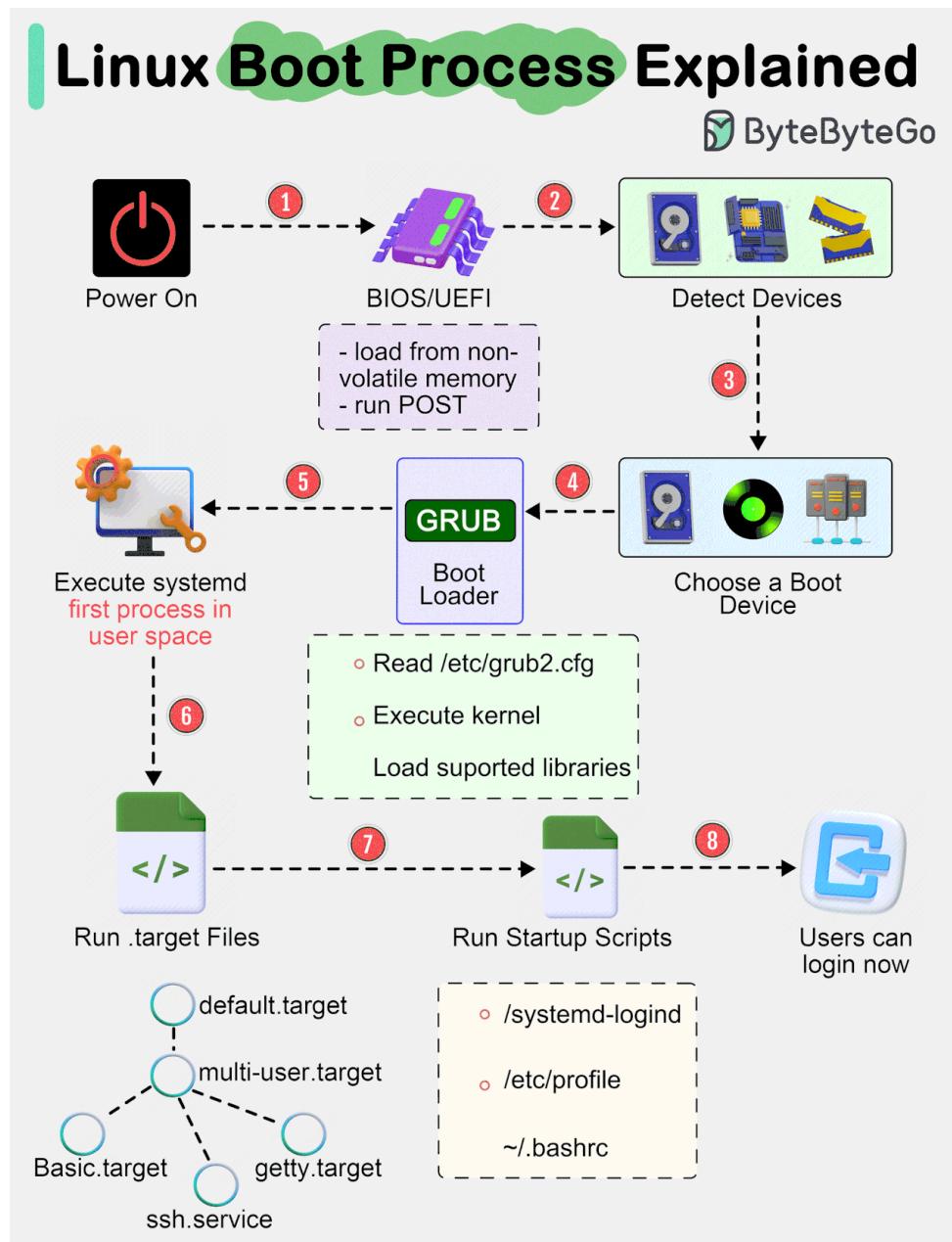
Over to you:

Which REST API authentication method do you find most effective in ensuring both security and usability for your applications?

Linux Boot Process Illustrated

We've made a video (YouTube Link at the end).

The diagram below shows the steps.



Step 1 - When we turn on the power, BIOS (Basic Input/Output System) or UEFI (Unified Extensible Firmware Interface) firmware is loaded from non-volatile memory, and executes POST (Power On Self Test).

Step 2 - BIOS/UEFI detects the devices connected to the system, including CPU, RAM, and storage.

Step 3 - Choose a booting device to boot the OS from. This can be the hard drive, the network server, or CD ROM.

Step 4 - BIOS/UEFI runs the boot loader (GRUB), which provides a menu to choose the OS or the kernel functions.

Step 5 - After the kernel is ready, we now switch to the user space. The kernel starts up systemd as the first user-space process, which manages the processes and services, probes all remaining hardware, mounts filesystems, and runs a desktop environment.

Step 6 - systemd activates the default target unit by default when the system boots. Other analysis units are executed as well.

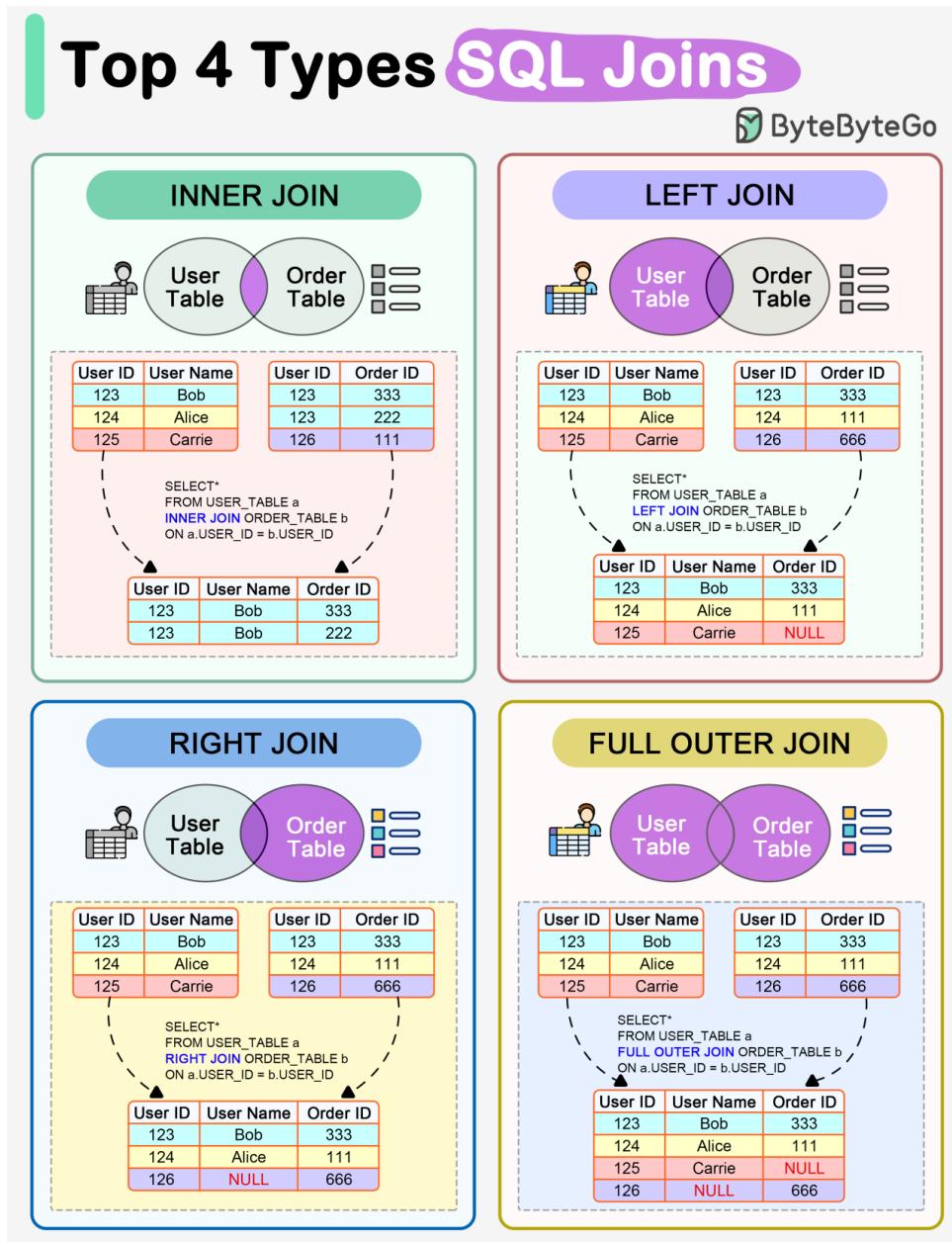
Step 7 - The system runs a set of startup scripts and configures the environment.

Step 8 - The users are presented with a login window. The system is now ready.

Watch and subscribe here: <https://lnkd.in/ezkZb5Wq>

How do SQL Joins Work?

The diagram below shows how 4 types of SQL joins work in detail.



◆ INNER JOIN

Returns matching rows in both tables.

◆ LEFT JOIN

Returns all records from the left table, and the matching records from the right table.

- ◆ **RIGHT JOIN**

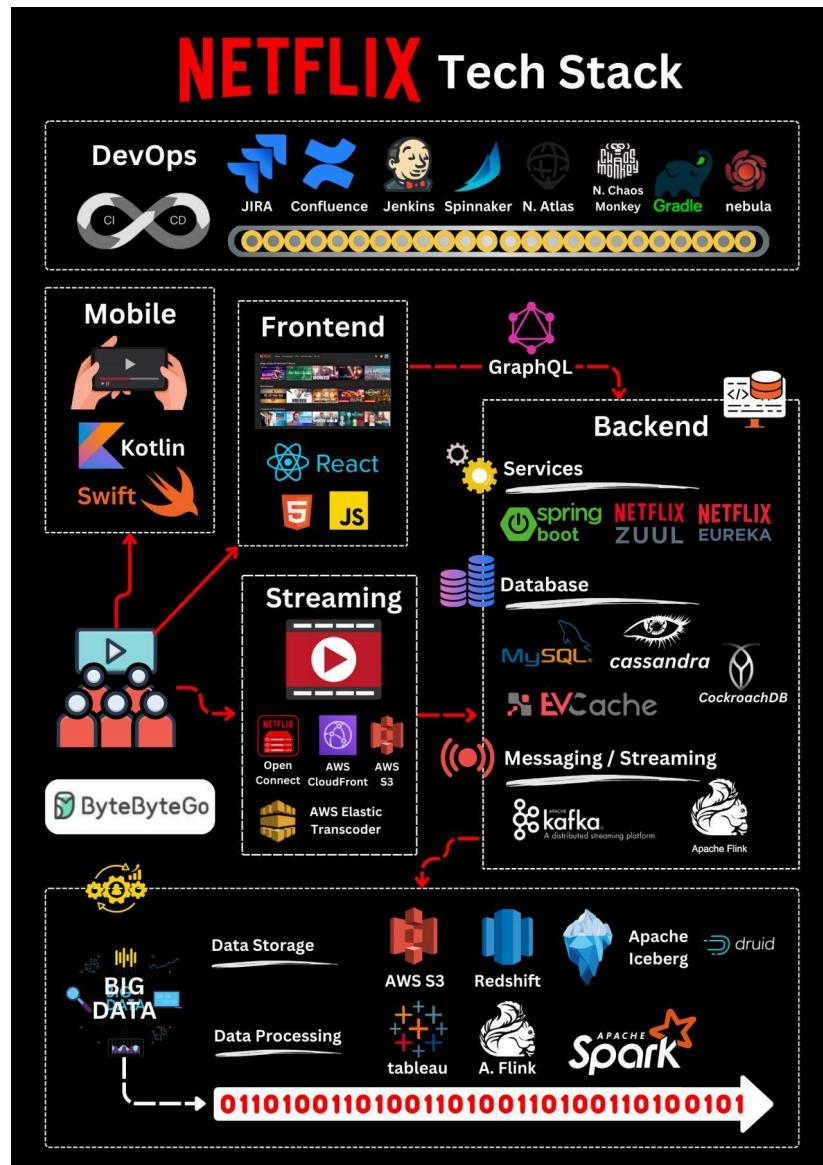
Returns all records from the right table, and the matching records from the left table.

- ◆ **FULL OUTER JOIN**

Returns all records where there is a match in either the left or right table.

Netflix's Tech Stack

This post is based on research from many Netflix engineering blogs and open-source projects. If you come across any inaccuracies, please feel free to inform us.



Mobile and web: Netflix has adopted Swift and Kotlin to build native mobile apps. For its web application, it uses React.

Frontend/server communication: GraphQL.

Backend services: Netflix relies on ZUUL, Eureka, the Spring Boot framework, and other technologies.

Databases: Netflix utilizes EV cache, Cassandra, CockroachDB, and other databases.

Messaging/streaming: Netflix employs Apache Kafka and Flink for messaging and streaming purposes.

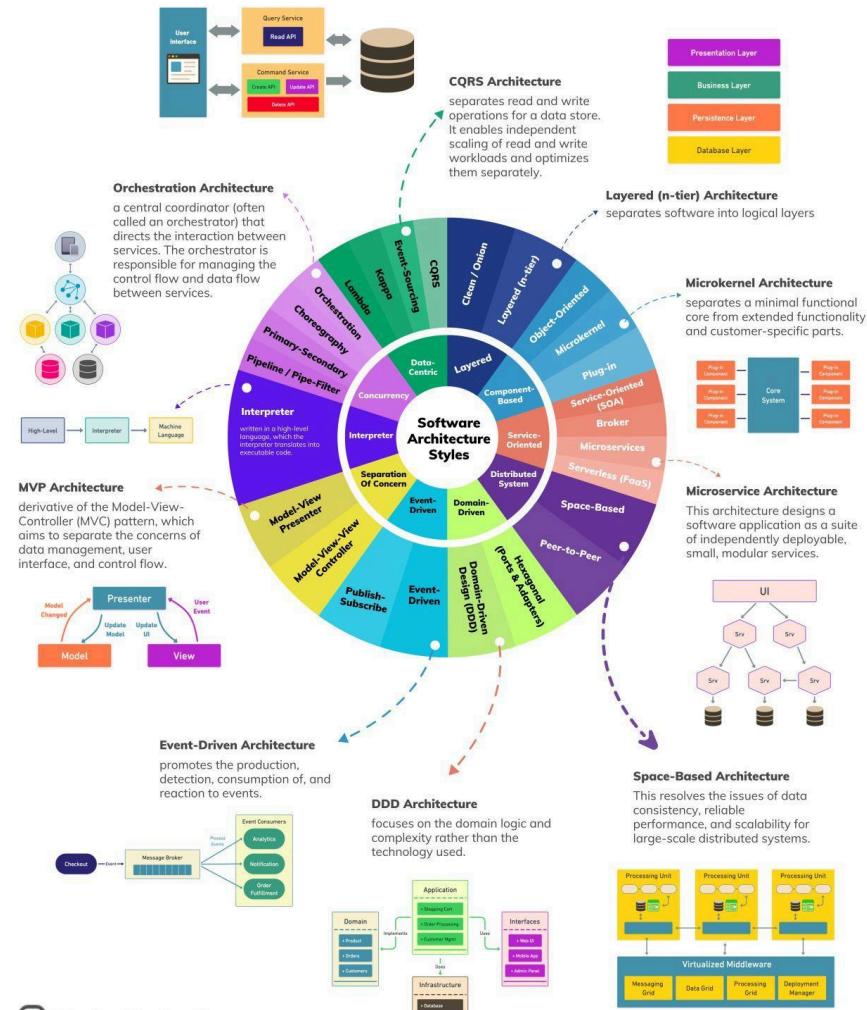
Video storage: Netflix uses S3 and Open Connect for video storage.

Data processing: Netflix utilizes Flink and Spark for data processing, which is then visualized using Tableau. Redshift is used for processing structured data warehouse information.

CI/CD: Netflix employs various tools such as JIRA, Confluence, PagerDuty, Jenkins, Gradle, Chaos Monkey, Spinnaker, Altas, and more for CI/CD processes.

Top Architectural Styles

Software Architecture Styles



 ByteByteGo

In software development, architecture plays a crucial role in shaping the structure and behavior of software systems. It provides a blueprint for system design, detailing how components interact with each other to deliver specific functionality. They also offer solutions to common problems, saving time and effort and leading to more robust and maintainable systems.

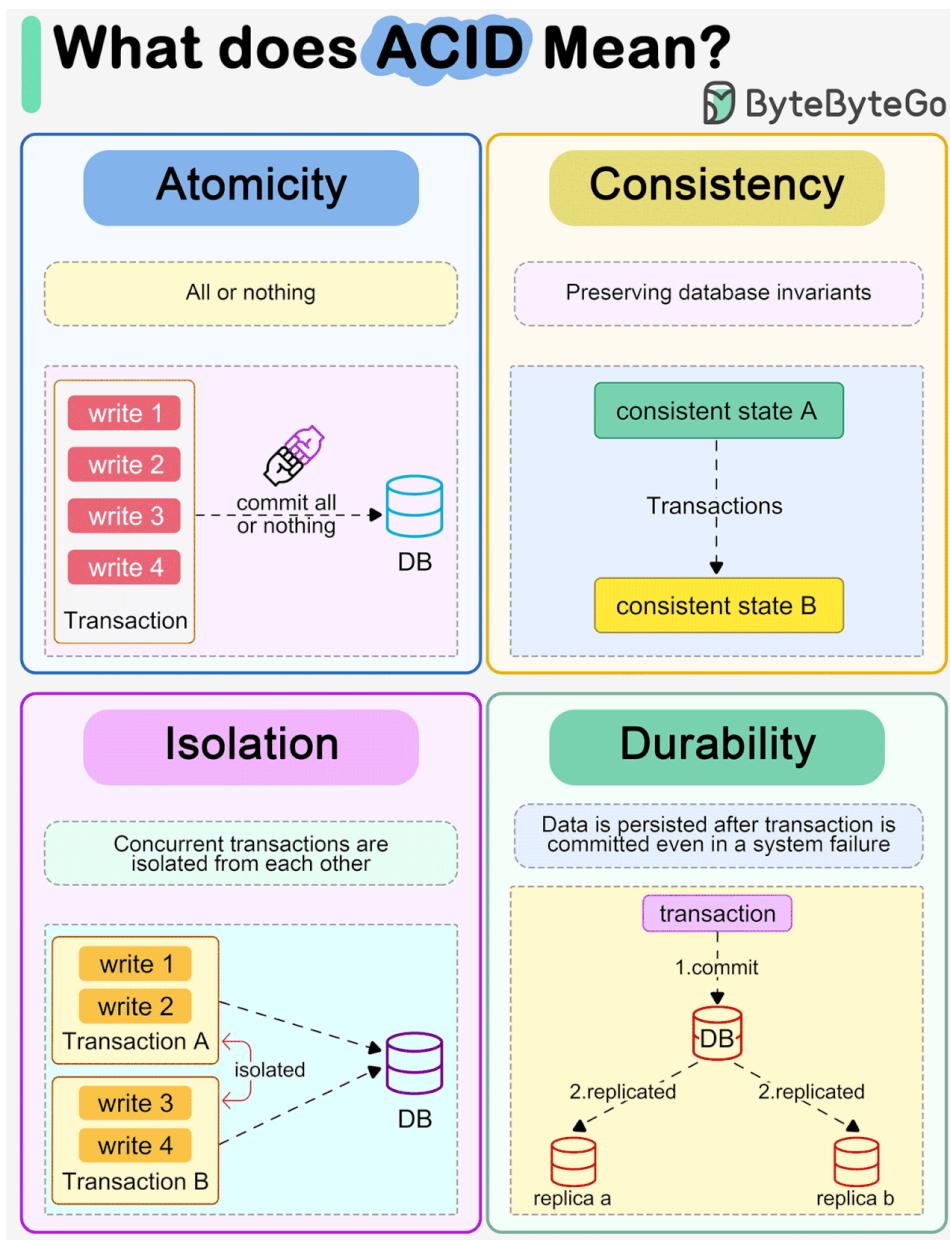
However, with the vast array of architectural styles and patterns available, it can take time to discern which approach best suits a particular project or system. Aims to shed light on these

concepts, helping you make informed decisions in your architectural endeavors.

To help you navigate the vast landscape of architectural styles and patterns, there is a cheat sheet that encapsulates all. This cheat sheet is a handy reference guide that you can use to quickly recall the main characteristics of each architectural style and pattern.

What does ACID mean?

The diagram below explains what ACID means in the context of a database transaction.



◆ Atomicity

The writes in a transaction are executed all at once and cannot be broken into smaller parts. If there are faults when executing the transaction, the writes in the transaction are rolled back.

So atomicity means “all or nothing”.

- ◆ Consistency

Unlike “consistency” in CAP theorem, which means every read receives the most recent write or an error, here consistency means preserving database invariants. Any data written by a transaction must be valid according to all defined rules and maintain the database in a good state.

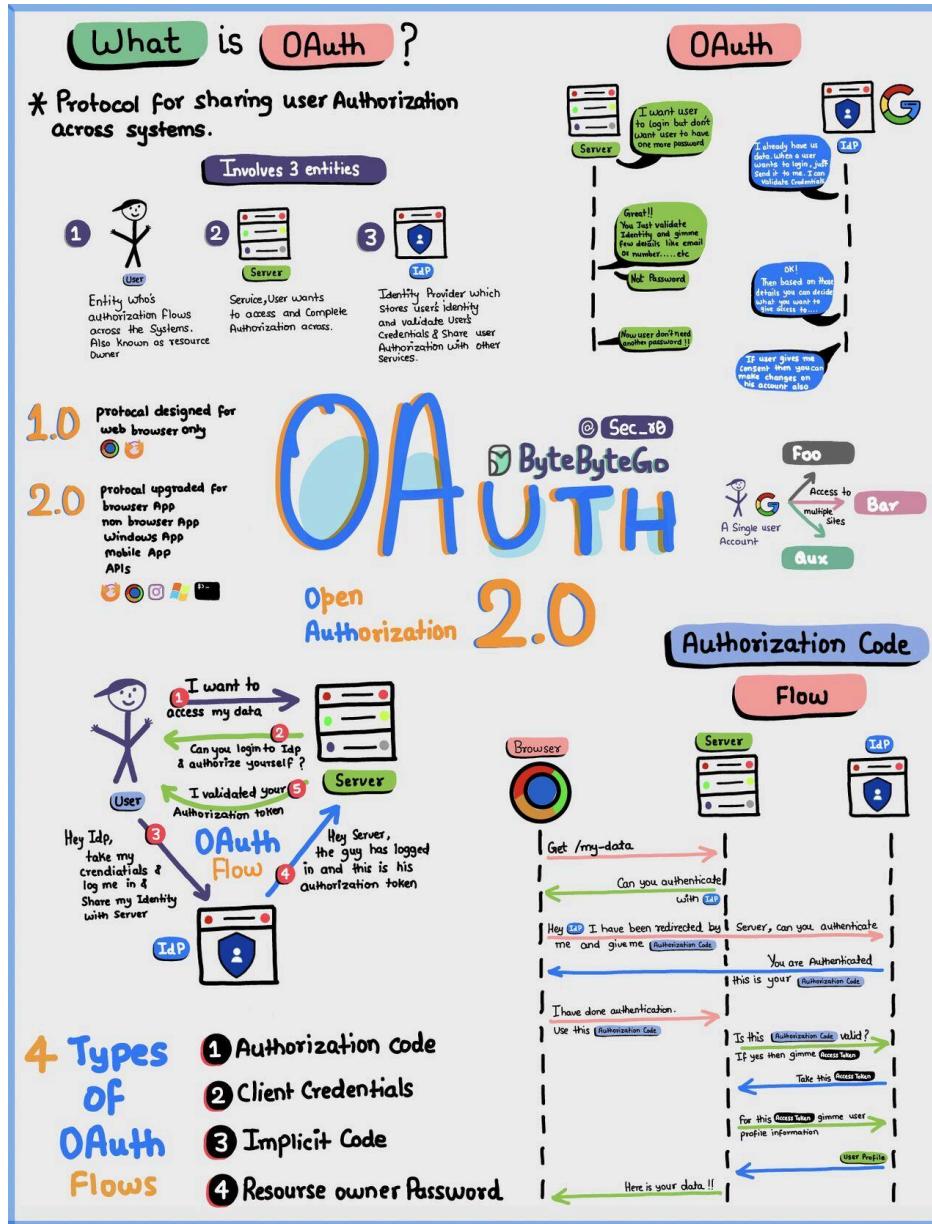
- ◆ Isolation

When there are concurrent writes from two different transactions, the two transactions are isolated from each other. The most strict isolation is “serializability”, where each transaction acts like it is the only transaction running in the database. However, this is hard to implement in reality, so we often adopt a lesser isolation level.

- ◆ Durability

Data is persisted after a transaction is committed even in a system failure. In a distributed system, this means the data is replicated to some other nodes.

Oauth 2.0 Explained With Simple Terms



OAuth 2.0 is a powerful and secure framework that allows different applications to securely interact with each other on behalf of users without sharing sensitive credentials.

The entities involved in OAuth are the User, the Server, and the Identity Provider (IDP).

What Can an OAuth Token Do?

When you use OAuth, you get an OAuth token that represents your identity and permissions. This token can do a few important things:

Single Sign-On (SSO): With an OAuth token, you can log into multiple services or apps using just one login, making life easier and safer.

Authorization Across Systems: The OAuth token allows you to share your authorization or access rights across various systems, so you don't have to log in separately everywhere.

Accessing User Profile: Apps with an OAuth token can access certain parts of your user profile that you allow, but they won't see everything.

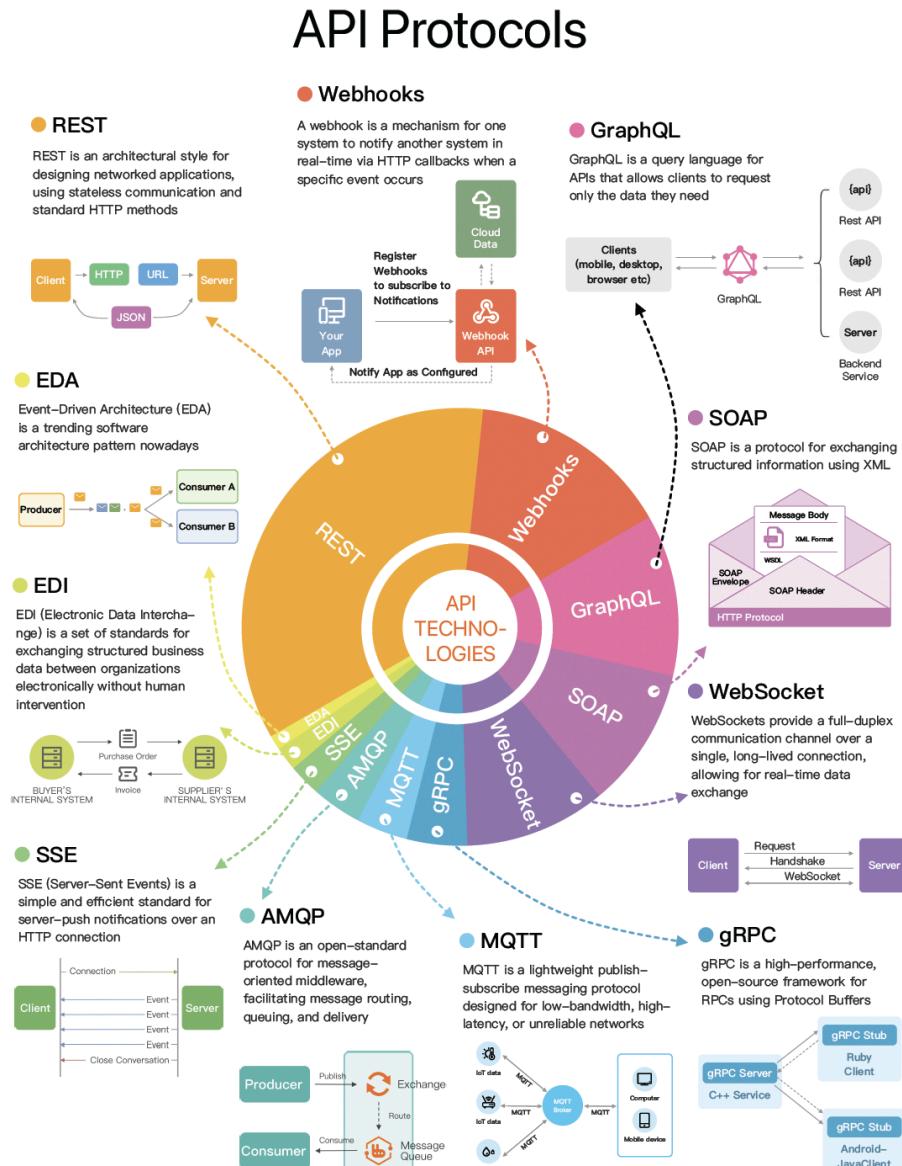
Remember, OAuth 2.0 is all about keeping you and your data safe while making your online experiences seamless and hassle-free across different applications and services.

Over to you: Imagine you have a magical power to grant one wish to OAuth 2.0. What would that be? Maybe your suggestions actually lead to OAuth 3.

The Evolving Landscape of API Protocols in 2023

This is a brief summary of the blog post I wrote for Postman.

In this blog post, I cover the six most popular API protocols: REST, Webhooks, GraphQL, SOAP, WebSocket, and gRPC. The discussion includes the benefits and challenges associated with each protocol.

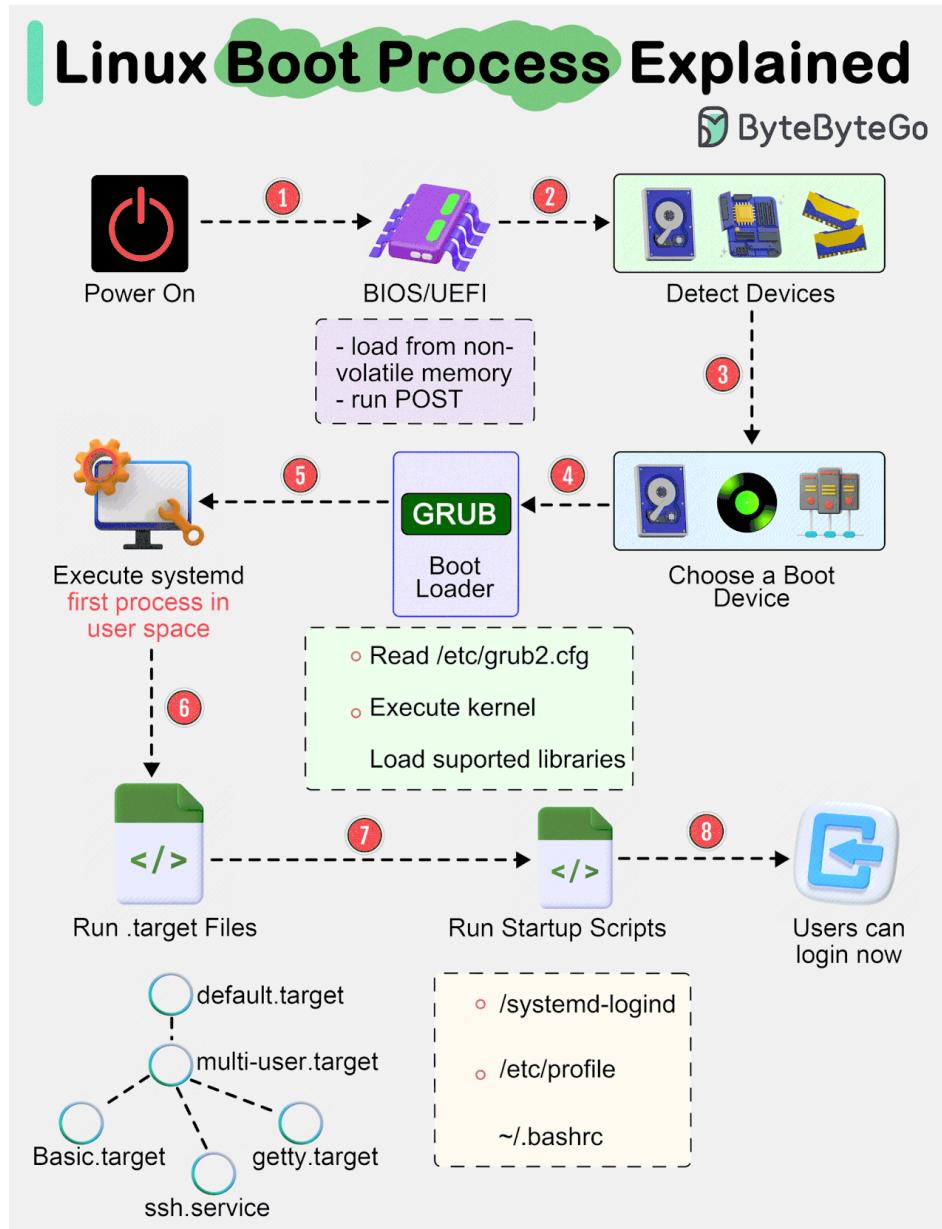


You can read the full blog post here: <https://blog.postman.com/api-protocols-in-2023/>

Linux boot Process Explained

Almost every software engineer has used Linux before, but only a handful know how its Boot Process works :) Let's dive in.

The diagram below shows the steps.



Step 1 - When we turn on the power, BIOS (Basic Input/Output System) or UEFI (Unified Extensible Firmware Interface) firmware is loaded from non-volatile memory, and executes POST (Power On Self Test).

Step 2 - BIOS/UEFI detects the devices connected to the system, including CPU, RAM, and storage.

Step 3 - Choose a booting device to boot the OS from. This can be the hard drive, the network server, or CD ROM.

Step 4 - BIOS/UEFI runs the boot loader (GRUB), which provides a menu to choose the OS or the kernel functions.

Step 5 - After the kernel is ready, we now switch to the user space. The kernel starts up systemd as the first user-space process, which manages the processes and services, probes all remaining hardware, mounts filesystems, and runs a desktop environment.

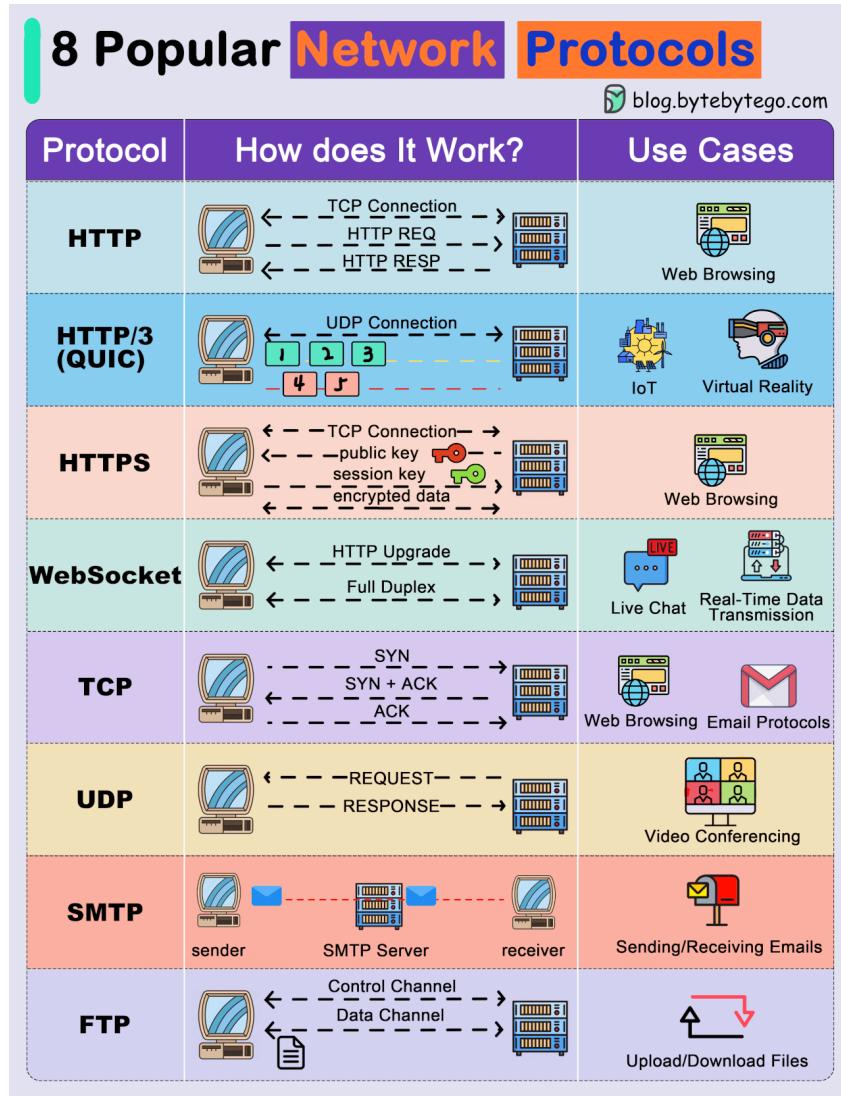
Step 6 - systemd activates the default target unit by default when the system boots. Other analysis units are executed as well.

Step 7 - The system runs a set of startup scripts and configure the environment.

Step 8 - The users are presented with a login window. The system is now ready.

Explaining 8 Popular Network Protocols in 1 Diagram.

You can find the link to watch a detailed video explanation at the end of the post.



Network protocols are standard methods of transferring data between two computers in a network.

1. HTTP (HyperText Transfer Protocol)

HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol.

2. HTTP/3

HTTP/3 is the next major revision of the HTTP. It runs on QUIC, a new transport protocol designed for mobile-heavy internet usage. It relies on UDP instead of TCP, which enables faster

web page responsiveness. VR applications demand more bandwidth to render intricate details of a virtual scene and will likely benefit from migrating to HTTP/3 powered by QUIC.

3. HTTPS (HyperText Transfer Protocol Secure)

HTTPS extends HTTP and uses encryption for secure communications.

4. WebSocket

WebSocket is a protocol that provides full-duplex communications over TCP. Clients establish WebSockets to receive real-time updates from the back-end services. Unlike REST, which always “pulls” data, WebSocket enables data to be “pushed”. Applications, like online gaming, stock trading, and messaging apps leverage WebSocket for real-time communication.

5. TCP (Transmission Control Protocol)

TCP is designed to send packets across the internet and ensure the successful delivery of data and messages over networks. Many application-layer protocols are built on top of TCP.

6. UDP (User Datagram Protocol)

UDP sends packets directly to a target computer, without establishing a connection first. UDP is commonly used in time-sensitive communications where occasionally dropping packets is better than waiting. Voice and video traffic are often sent using this protocol.

7. SMTP (Simple Mail Transfer Protocol)

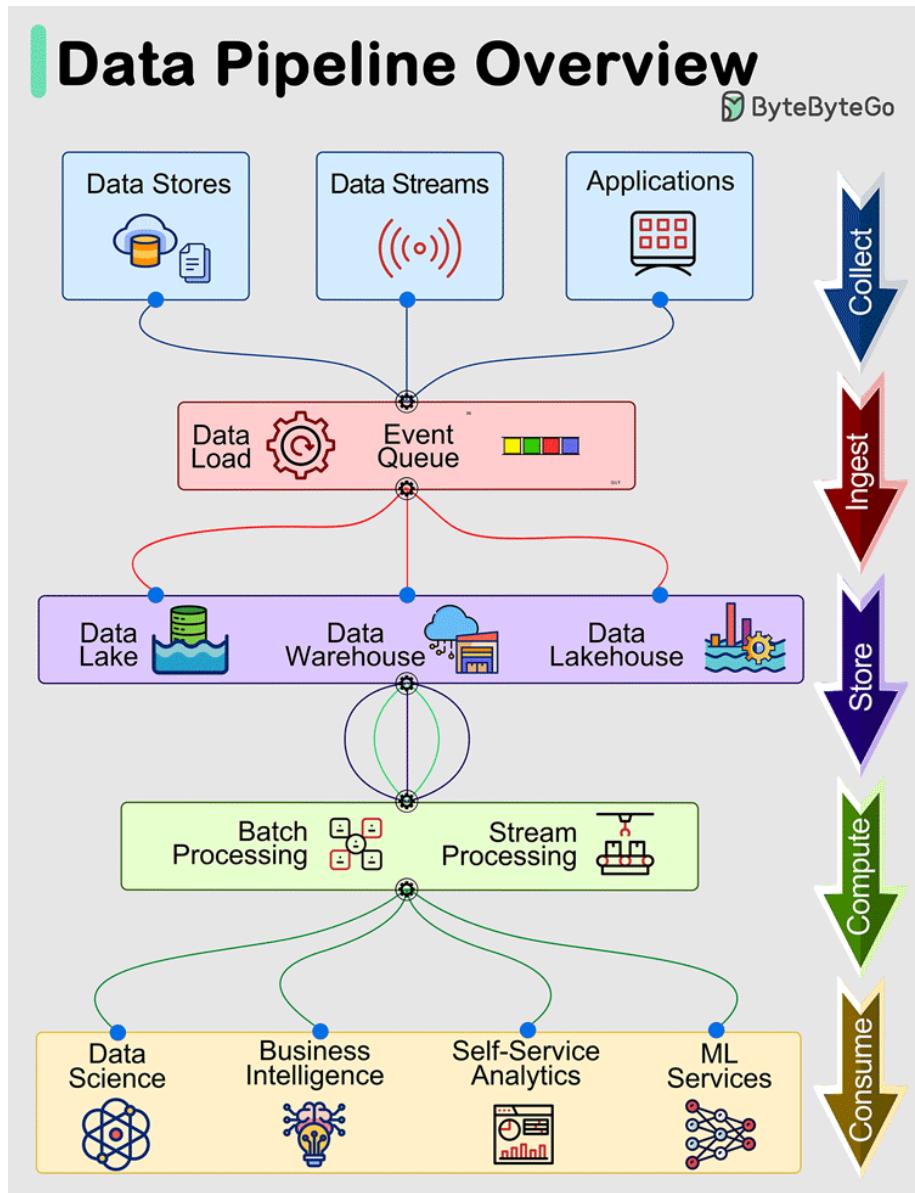
SMTP is a standard protocol to transfer electronic mail from one user to another.

8. FTP (File Transfer Protocol)

FTP is used to transfer computer files between client and server. It has separate connections for the control channel and data channel.

Data Pipelines Overview

Data pipelines are a fundamental component of managing and processing data efficiently within modern systems. These pipelines typically encompass 5 predominant phases: Collect, Ingest, Store, Compute, and Consume.



1. Collect:

Data is acquired from data stores, data streams, and applications, sourced remotely from devices, applications, or business systems.

2. Ingest:

During the ingestion process, data is loaded into systems and organized within event queues.

3. Store:

Post ingestion, organized data is stored in data warehouses, data lakes, and data lakehouses, along with various systems like databases, ensuring post-ingestion storage.

4. Compute:

Data undergoes aggregation, cleansing, and manipulation to conform to company standards, including tasks such as format conversion, data compression, and partitioning. This phase employs both batch and stream processing techniques.

5. Consume:

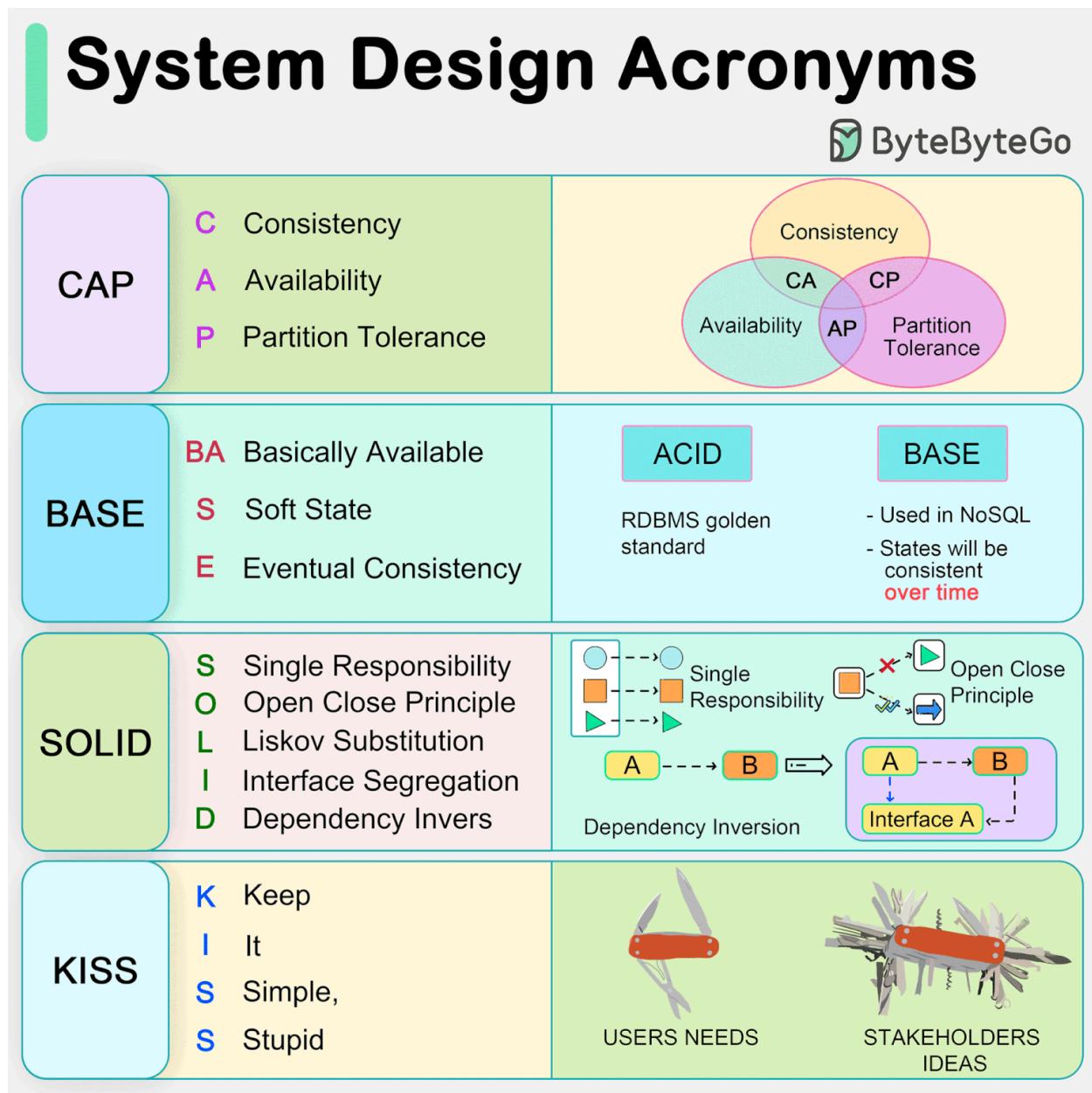
Processed data is made available for consumption through analytics and visualization tools, operational data stores, decision engines, user-facing applications, dashboards, data science, machine learning services, business intelligence, and self-service analytics.

The efficiency and effectiveness of each phase contribute to the overall success of data-driven operations within an organization.

Over to you: What's your story with data-driven pipelines? How have they influenced your data management game?

CAP, BASE, SOLID, KISS, What do these acronyms mean?

The diagram below explains the common acronyms in system designs.



◆ CAP

CAP theorem states that any distributed data store can only provide two of the following three guarantees:

1. Consistency - Every read receives the most recent write or an error.
2. Availability - Every request receives a response.
3. Partition tolerance - The system continues to operate in network faults.

However, this theorem was criticized for being too narrow for distributed systems, and we shouldn't use it to categorize the databases. Network faults are guaranteed to happen in distributed systems, and we must deal with this in any distributed systems.

You can read more on this in “Please stop calling databases CP or AP” by Martin Kleppmann.

◆ **BASE**

The ACID (Atomicity-Consistency-Isolation-Durability) model used in relational databases is too strict for NoSQL databases. The BASE principle offers more flexibility, choosing availability over consistency. It states that the states will eventually be consistent.

◆ **SOLID**

SOLID principle is quite famous in OOP. There are 5 components to it.

1. SRP (Single Responsibility Principle)

Each unit of code should have one responsibility.

2. OCP (Open Close Principle)

Units of code should be open for extension but closed for modification.

3. LSP (Liskov Substitution Principle)

A subclass should be able to be substituted by its base class.

4. ISP (Interface Segregation Principle)

Expose multiple interfaces with specific responsibilities.

5. DIP (Dependency Inversion Principle)

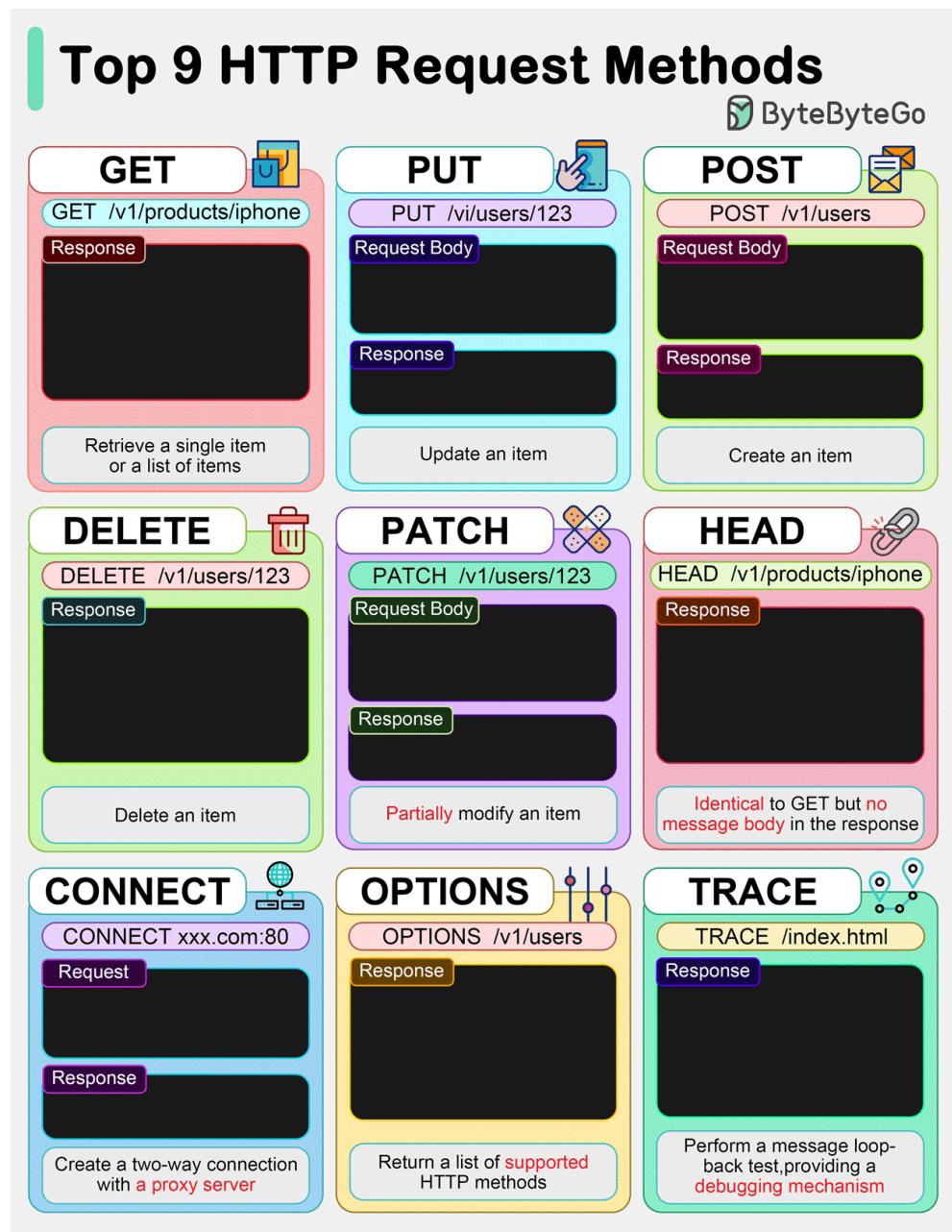
Use abstractions to decouple dependencies in the system.

◆ **KISS**

"Keep it simple, stupid!" is a design principle first noted by the U.S. Navy in 1960. It states that most systems work best if they are kept simple.

Over to you: Have you invented any acronyms in your career?

GET, POST, PUT... Common HTTP “verbs” in one figure



1. HTTP GET

This retrieves a resource from the server. It is idempotent. Multiple identical requests return the same result.

2. HTTP PUT

This updates or Creates a resource. It is idempotent. Multiple identical requests will

update the same resource.

3. HTTP POST

This is used to create new resources. It is not idempotent, making two identical POST will duplicate the resource creation.

4. HTTP DELETE

This is used to delete a resource. It is idempotent. Multiple identical requests will delete the same resource.

5. HTTP PATCH

The PATCH method applies partial modifications to a resource.

6. HTTP HEAD

The HEAD method asks for a response identical to a GET request but without the response body.

7. HTTP CONNECT

The CONNECT method establishes a tunnel to the server identified by the target resource.

8. HTTP OPTIONS

This describes the communication options for the target resource.

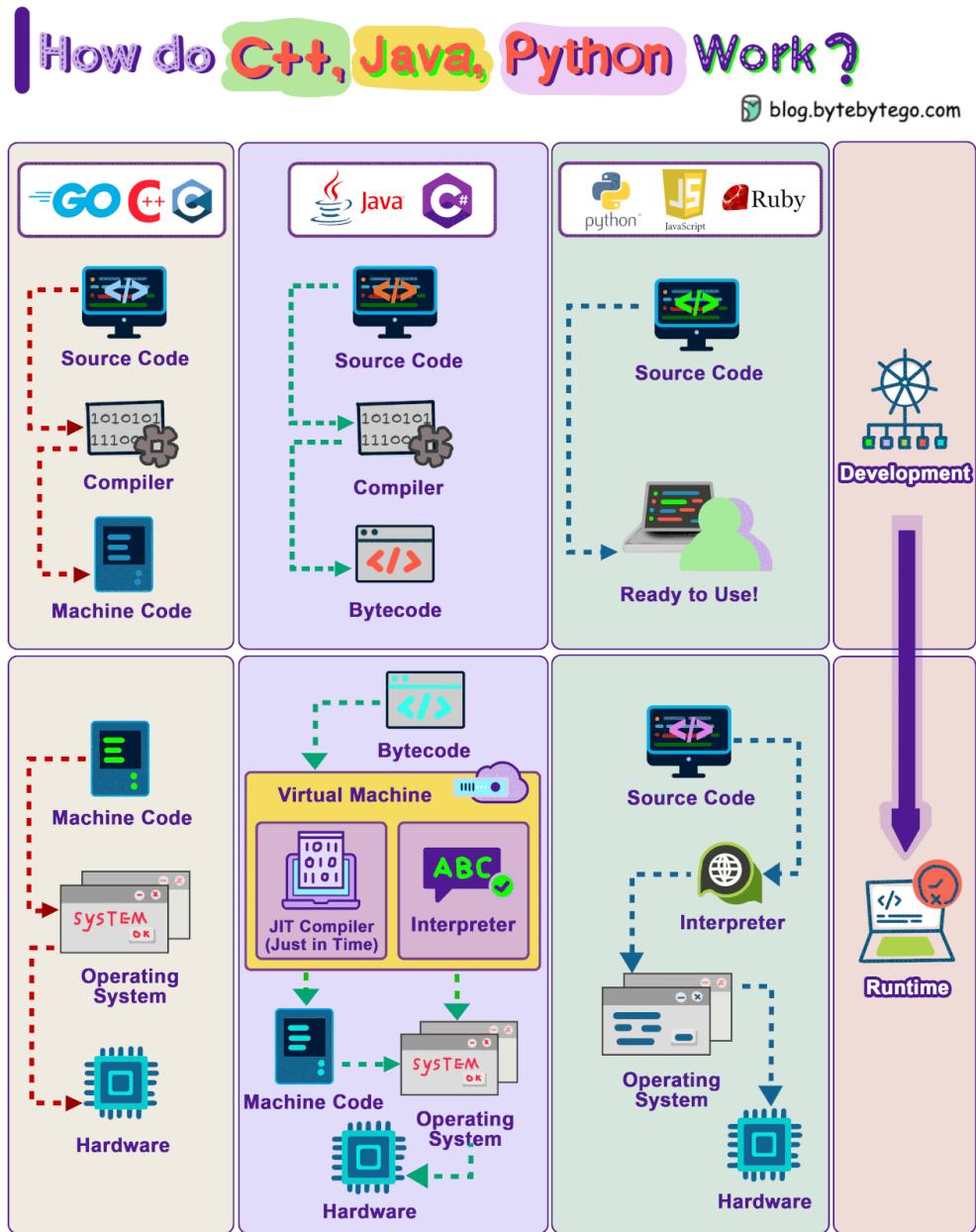
9. HTTP TRACE

This performs a message loop-back test along the path to the target resource.

Over to you: What other HTTP verbs have you used?

How Do C++, Java, Python Work?

The diagram shows how the compilation and execution work.



Compiled languages are compiled into machine code by the compiler. The machine code can later be executed directly by the CPU. Examples: C, C++, Go.

A bytecode language like Java, compiles the source code into bytecode first, then the JVM executes the program. Sometimes JIT (Just-In-Time) compiler compiles the source code into

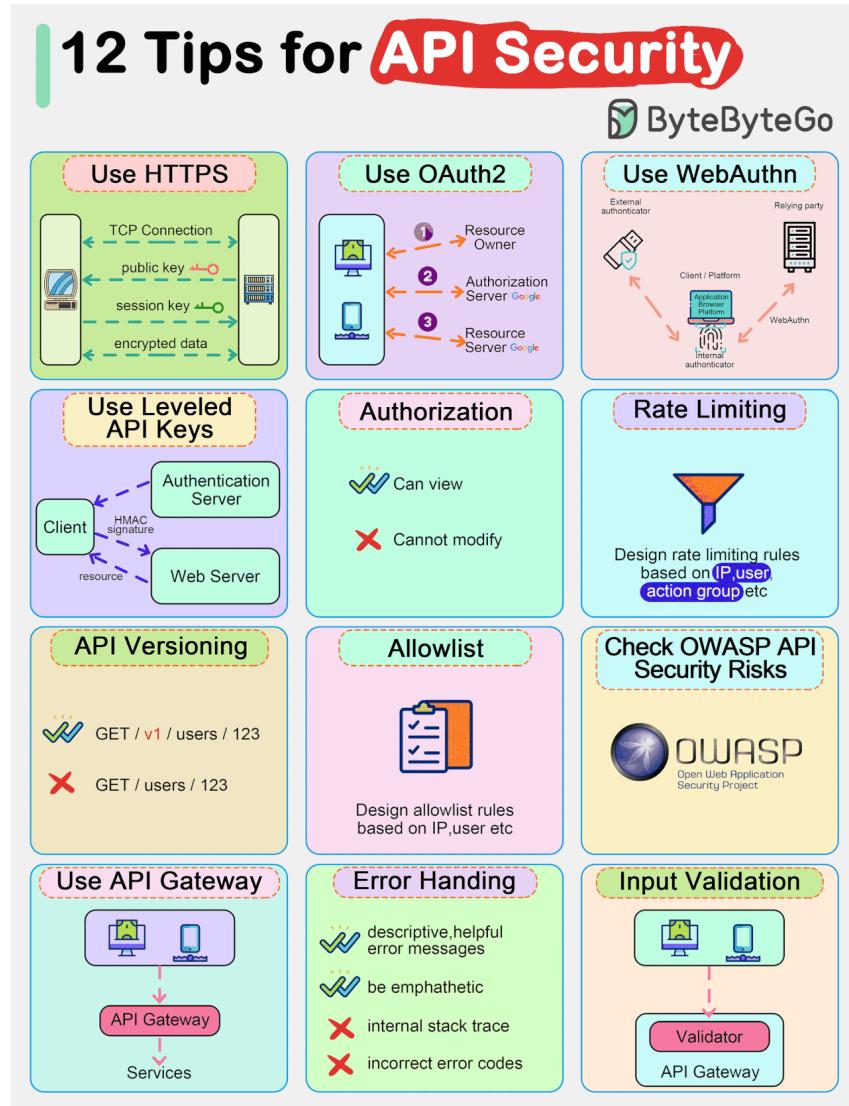
machine code to speed up the execution. Examples: Java, C#

Interpreted languages are not compiled. They are interpreted by the interpreter during runtime. Examples: Python, Javascript, Ruby

Compiled languages in general run faster than interpreted languages.

Over to you: which type of language do you prefer?

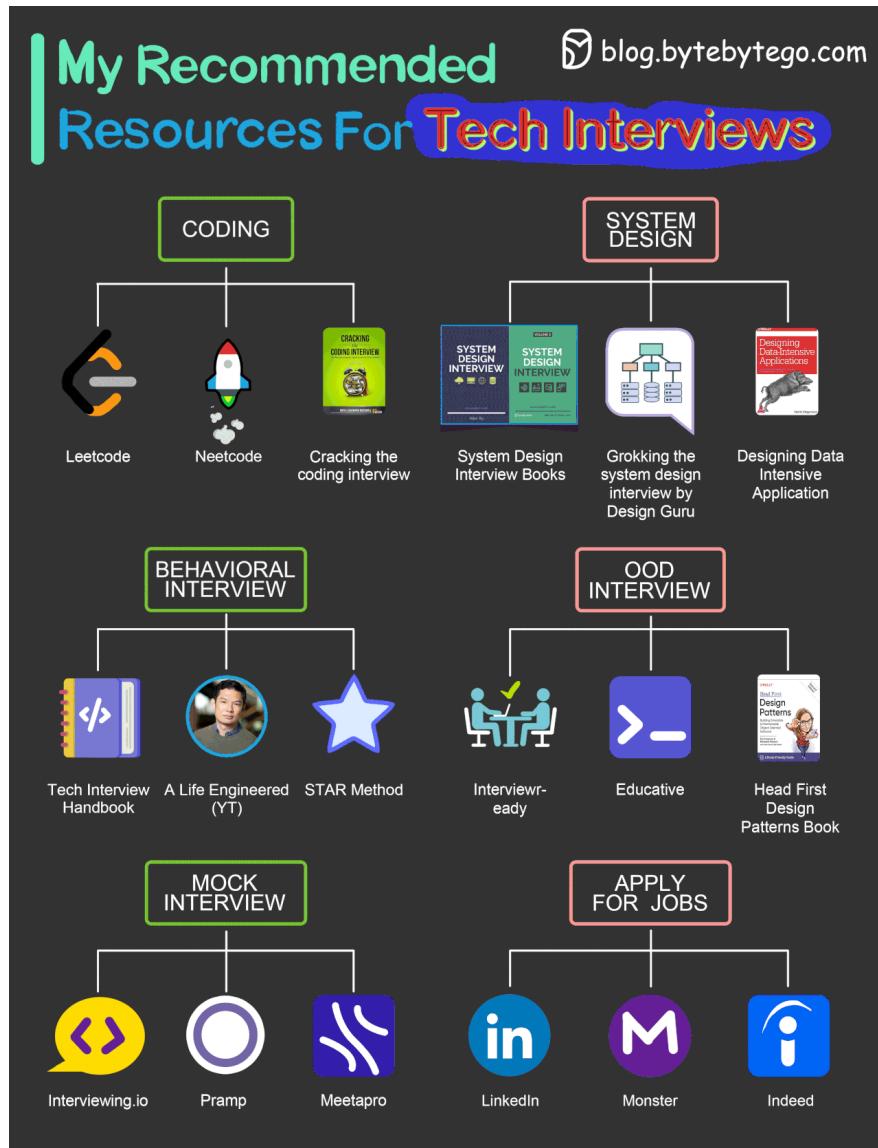
Top 12 Tips for API Security



- Use HTTPS
- Use OAuth2
- Use WebAuthn
- Use Leveled API Keys
- Authorization
- Rate Limiting
- API Versioning
- Whitelisting
- Check OWASP API Security Risks
- Use API Gateway
- Error Handling
- Input Validation

Our recommended materials to crack your next tech interview

You can find the link to watch a detailed video explanation at the end of the post.



Coding

- Leetcode
- Cracking the coding interview book
- Neetcode

System Design Interview

- System Design Interview book 1, 2 by Alex Xu
- Grokking the system design by Design Guru
- Design Data-intensive Application book

Behavioral interview

- Tech Interview Handbook (Github repo)
- A Life Engineered (YT)
- STAR method (general method)

OOD Interview

- Interviewready
- OOD by educative
- Head First Design Patterns Book

Mock interviews

- Interviewingio
- Pramp
- Meetapro

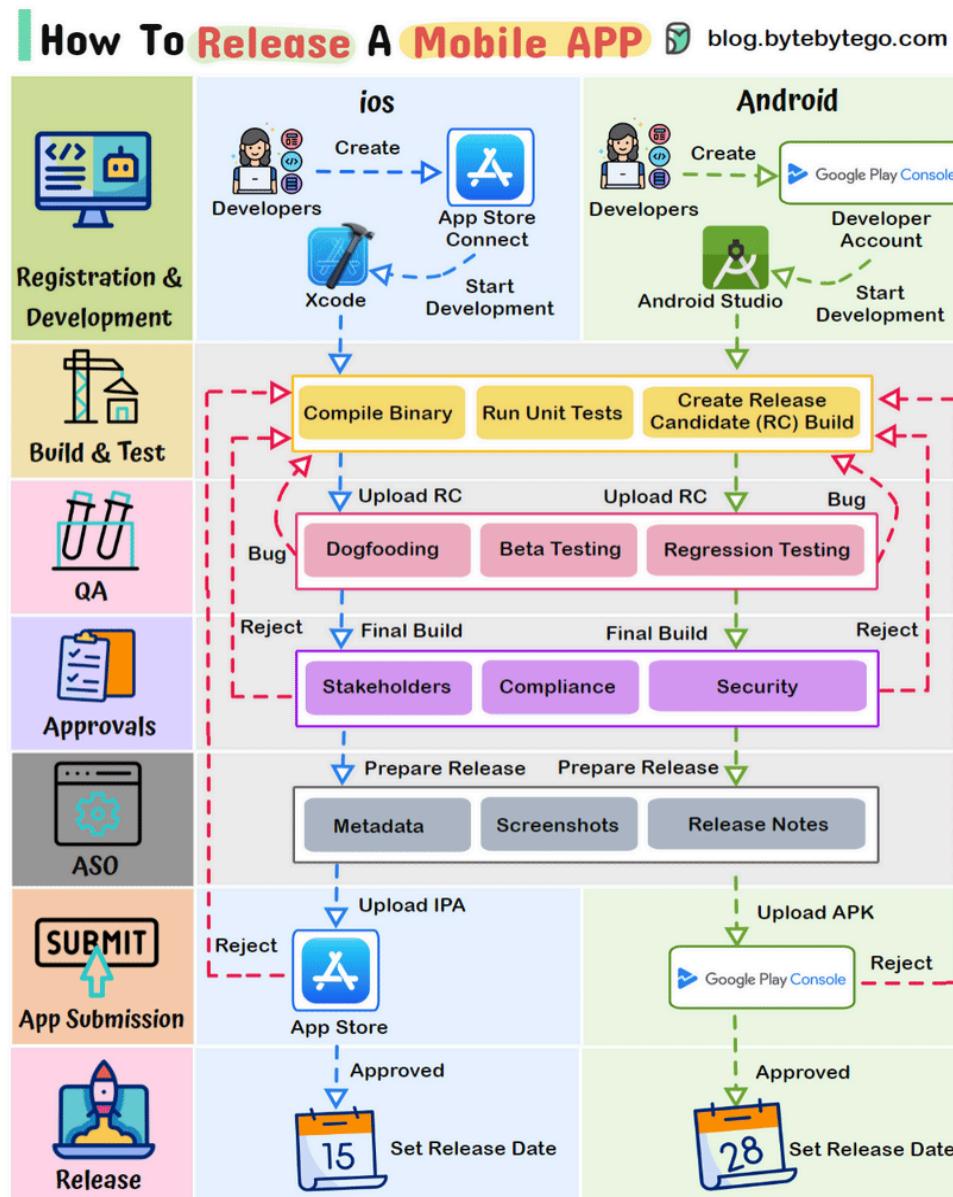
Apply for Jobs

- Linkedin
- Monster
- Indeed

Over to you: What is your favorite interview prep material?

How To Release A Mobile App

The mobile app release process differs from conventional methods. This illustration simplifies the journey to help you understand.



Typical Stages in a Mobile App Release Process:

1. Registration & Development (iOS & Android):

- Enroll in Apple's Developer Program and Google Play Console as iOS and Android developer
- Code using platform-specific tools: Swift/Obj-C for iOS, and Java/Kotlin for Android

2. Build & Test (iOS & Android):

Compile the app's binary, run extensive tests on both platforms to ensure functionality and performance. Create a release candidate build.

3. QA:

- Internally test the app for issue identification (dogfooding)
- Beta test with external users to collect feedback
- Conduct regression testing to maintain feature stability

4. Internal Approvals:

- Obtain approval from stakeholders and key team members.
- Comply with app store guidelines and industry regulations
- Obtain security approvals to safeguard user data and privacy

5. App Store Optimization (ASO):

- Optimize metadata, including titles, descriptions, and keywords, for better search visibility
- Design captivating screenshots and icons to entice users
- Prepare engaging release notes to inform users about new features and updates

6. App Submission To Store:

- Submit the iOS app via App Store Connect following Apple's guidelines
- Submit the Android app via Google Play Console, adhering to Google's policies
- Both platforms may request issues resolution for approval

7. Release:

- Upon approval, set a release date to coordinate the launch on both iOS and Android platforms

Over to you:

What's the most challenging phase you've encountered in the mobile app release process?

A handy cheat sheet for the most popular cloud services (2023 edition)

Cloud Comparison Cheat Sheet				 blog.bytebytego.com
 <ul style="list-style-type: none"> Elastic Compute Cloud (EC2) Elastic Kubernetes Service (EKS) Lambda Simple Storage Service (S3) Elastic Block Store Elastic File System Virtual Private Cloud Route 53 Elastic Load Balancing Web Application Firewall RDS DynamoDB Redshift Elastic MapReduce Kinesis SageMaker Glue EventBridge Simple Queuing Service Simple Notification Service CloudWatch CloudFormation IAM KMS 	 <ul style="list-style-type: none"> Virtual Machine Azure Kubernetes Service (AKS) Azure Functions Blob Storage Managed Disk File Storage Virtual Network DNS Load Balancer Web Application Firewall SQL Database Cosmos DB Synapse Analytics HDInsight Streaming Analytics Machine Learning Data Factory Event Grid Storage Queues Service Bus Monitor Resource Manager Active Directory Key Vault 	 <ul style="list-style-type: none"> Compute Engine Google Kubernetes Engine (GKE) Cloud Functions Cloud Storage Persistent Disk File Store Virtual Private Cloud Cloud DNS Cloud Load Balancing Cloud Armor Cloud SQL Firebase Realtime Database BigQuery Dataproc Dataflow Vertex AI Data Fusion Eventarc Pub/Sub Firebase Cloud Messaging Cloud Monitoring Deployment Manager Cloud Identity Cloud KMS 	 <ul style="list-style-type: none"> Virtual Machine Instance Oracle Container Engine OCI Functions Object Storage Persistent Volume File Storage Virtual Cloud Network DNS Load Balancer Web Application Firewall ATP NoSQL Database Autonomous Data Warehouse Big Data Streaming Data Science Data Integration Events Streaming Notifications Monitoring Resource Manager IAM Vault 	 <ul style="list-style-type: none"> Elastic Compute Service Alibaba Cloud Kubernetes Service Function Compute Object Storage Service Block Storage Network Attached Storage Virtual Private Cloud DNS Server Load Balancer Web Application Firewall ApsaraDB RDS Table Store AnalyticDB Elastic MapReduce DataHub Platform for AI DataWorks Eventbridge Message Queue Message Service CloudMonitor Resource Orchestration Resource Access Management KMS

What's included?

- AWS, Azure, Google Cloud, Oracle Cloud, Alibaba Cloud
- Cloud servers
- Databases

- Message queues and streaming platforms
- Load balancing, DNS routing software
- Security
- Monitoring

Over to you - which company is the best at naming things?

Best ways to test system functionality

Testing system functionality is a crucial step in software development and engineering processes.

Process	Illustration	Tools
Unit Testing		pytest JUnit nunit MOCHA
Integration Testing		POSTMAN cucumber SoapUI Selenium
System Testing		Selenium ROBOT FRAMEWORK appium APACHE JMeter™
Load Testing		APACHE JMeter™ Gatling LOCUST LOAD RUNNER
Error Testing		Gremlin
Test Automation		Jenkins Travis CI circleci GitHub Actions

It ensures that a system or software application performs as expected, meets user requirements, and operates reliably.

Here we delve into the best ways:

1. Unit Testing: Ensures individual code components work correctly in isolation.
2. Integration Testing: Verifies that different system parts function seamlessly together.
3. System Testing: Assesses the entire system's compliance with user requirements and performance.
4. Load Testing: Tests a system's ability to handle high workloads and identifies performance issues.
5. Error Testing: Evaluates how the software handles invalid inputs and error conditions.
6. Test Automation: Automates test case execution for efficiency, repeatability, and error reduction.

Over to you:

- How do you approach testing system functionality in your software development or engineering projects?
- What's your company's release process look like?

Explaining JSON Web Token (JWT) to a 10 year old Kid

JWT {JSON WEB TOKEN}

With ❤️ By @Sec_zB

1 {"What": "JSON"}

* A file format to store data in key:value format

Key has to be string
can be string
or list of String or Json
Nested Json, [Json, Json]
Just a data structure :)

2 JWT Structure 3 parts

Header | data | Signature
↓ Base64 encode | ↓ Base64 encode | ↓ Base64 encode
X • Y • Z
Dots are just used for concatenation

3 Working?

1 Login (username, password) → 2 Validate credentials → 3 Create & sign JWT with secret → 4 Store JWT locally → 5 Authorization: Bearer JWT → 6 Validates signature → OK

4 Signing Alg

1 Public Key: Sign JWT with private key → Signed JWT + Public Key → Validate with public key
* RS256
* ES256 etc

2 Symmetric Key: Sign JWT with Shared Key → Signed JWT → Validate with Shared Key
* HMAC
* HS256

SecurityZines.com In Collaboration with ByteByteGo

Imagine you have a special box called a JWT. Inside this box, there are three parts: a header, a payload, and a signature.

The header is like the label on the outside of the box. It tells us what type of box it is and how it's secured. It's usually written in a format called JSON, which is just a way to organize information using curly braces {} and colons : .

The payload is like the actual message or information you want to send. It could be your name,

age, or any other data you want to share. It's also written in JSON format, so it's easy to understand and work with.

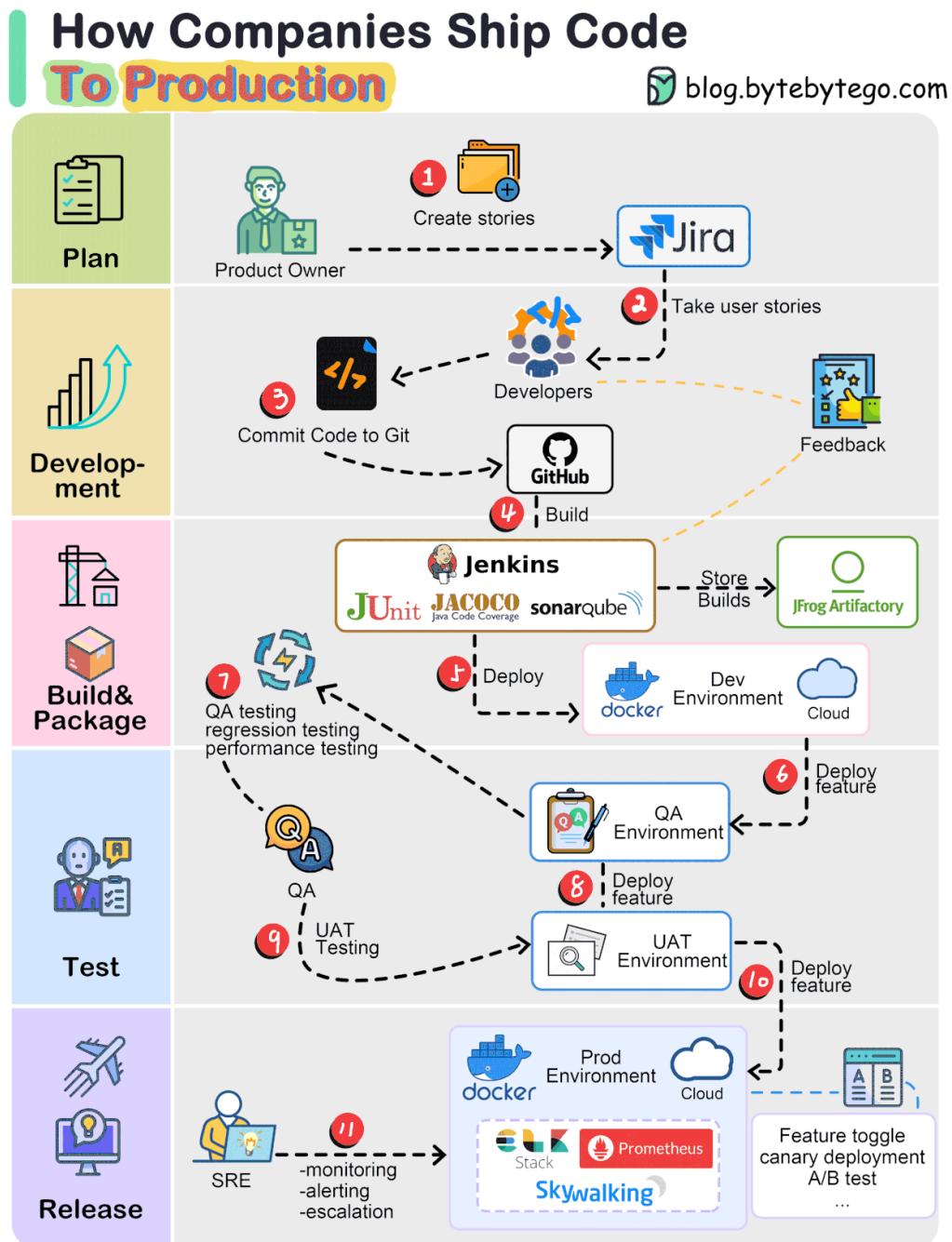
Now, the signature is what makes the JWT secure. It's like a special seal that only the sender knows how to create. The signature is created using a secret code, kind of like a password. This signature ensures that nobody can tamper with the contents of the JWT without the sender knowing about it.

When you want to send the JWT to a server, you put the header, payload, and signature inside the box. Then you send it over to the server. The server can easily read the header and payload to understand who you are and what you want to do.

Over to you: When should we use JWT for authentication? What are some other authentication methods?

How do companies ship code to production?

The diagram below illustrates the typical workflow.



Step 1: The process starts with a product owner creating user stories based on requirements.

Step 2: The dev team picks up the user stories from the backlog and puts them into a sprint for

a two-week dev cycle.

Step 3: The developers commit source code into the code repository Git.

Step 4: A build is triggered in Jenkins. The source code must pass unit tests, code coverage threshold, and gates in SonarQube.

Step 5: Once the build is successful, the build is stored in artifactory. Then the build is deployed into the dev environment.

Step 6: There might be multiple dev teams working on different features. The features need to be tested independently, so they are deployed to QA1 and QA2.

Step 7: The QA team picks up the new QA environments and performs QA testing, regression testing, and performance testing.

Step 8: Once the QA builds pass the QA team's verification, they are deployed to the UAT environment.

Step 9: If the UAT testing is successful, the builds become release candidates and will be deployed to the production environment on schedule.

Step 10: SRE (Site Reliability Engineering) team is responsible for prod monitoring.

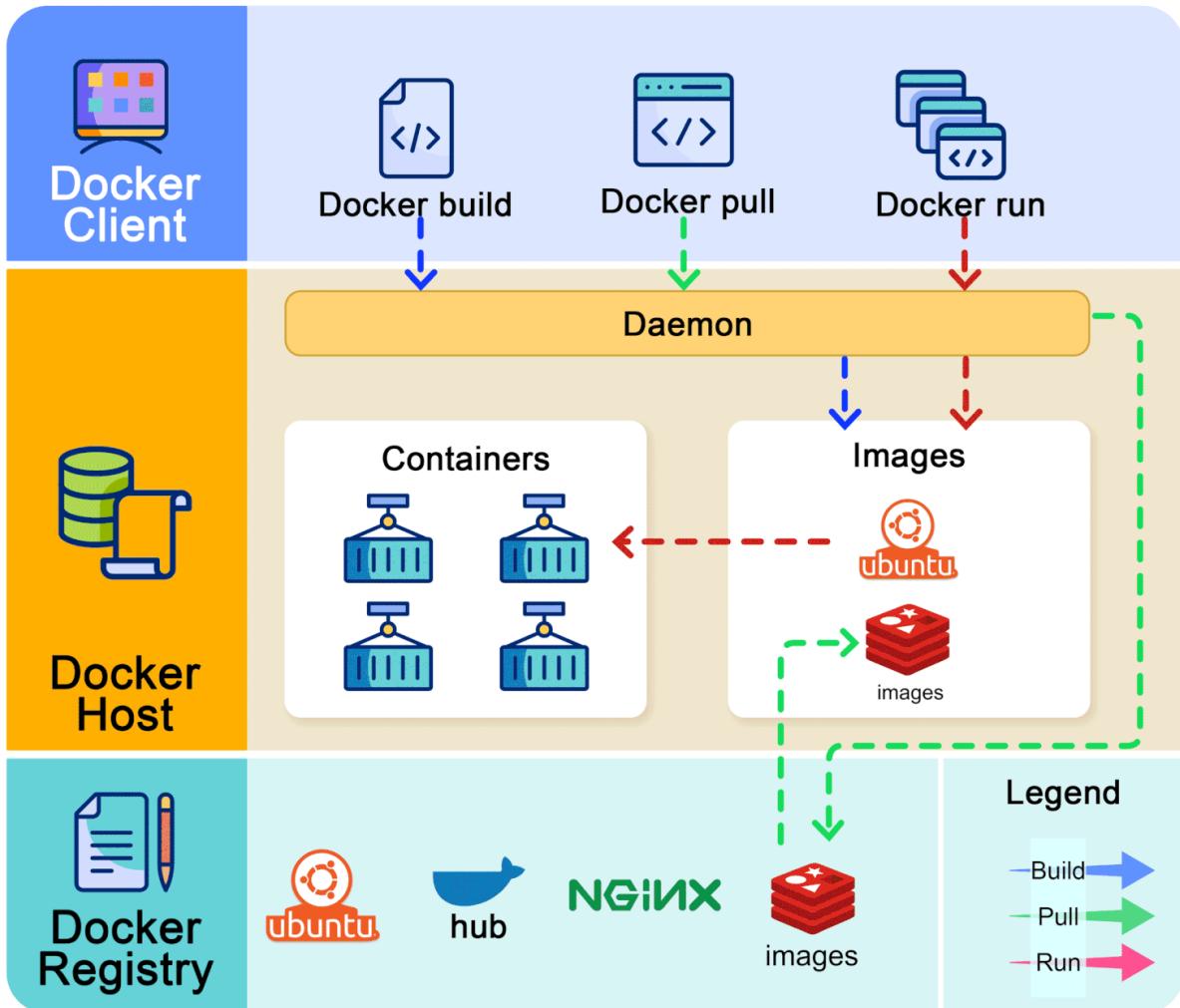
Over to you: what's your company's release process look like?

How does Docker Work? Is Docker still relevant?

We just made a video on this topic.

How does Docker Work ?

 blog.bytebytego.com



Docker's architecture comprises three main components:

- ◆ **Docker Client**

This is the interface through which users interact. It communicates with the Docker daemon.

- ◆ Docker Host

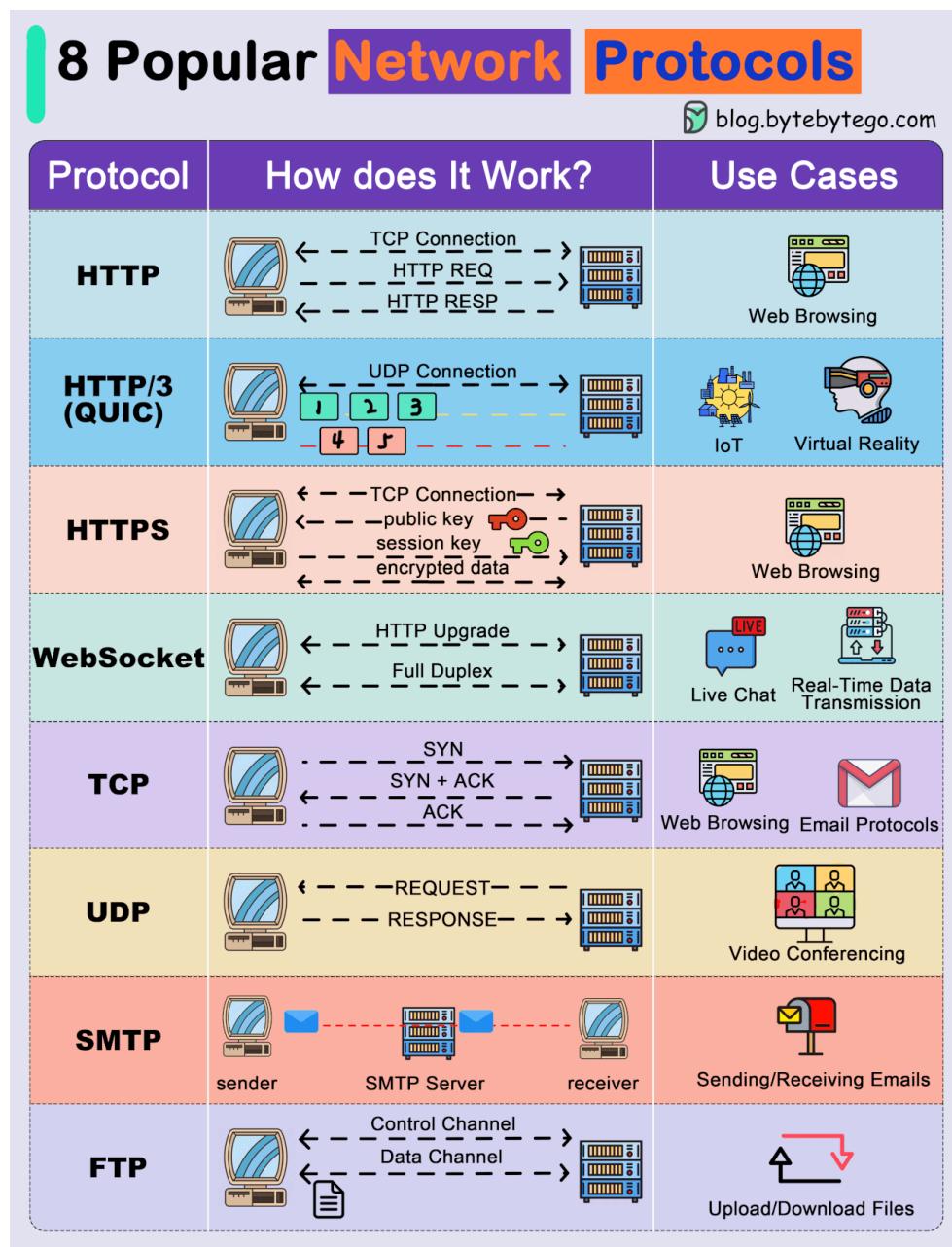
Here, the Docker daemon listens for Docker API requests and manages various Docker objects, including images, containers, networks, and volumes.

- ◆ Docker Registry

This is where Docker images are stored. Docker Hub, for instance, is a widely-used public registry.

Explaining 8 Popular Network Protocols in 1 Diagram

Network protocols are standard methods of transferring data between two computers in a network.



1. HTTP (HyperText Transfer Protocol)

HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol.

2. **HTTP/3**

HTTP/3 is the next major revision of the HTTP. It runs on QUIC, a new transport protocol designed for mobile-heavy internet usage. It relies on UDP instead of TCP, which enables faster web page responsiveness. VR applications demand more bandwidth to render intricate details of a virtual scene and will likely benefit from migrating to HTTP/3 powered by QUIC.

3. **HTTPS (HyperText Transfer Protocol Secure)**

HTTPS extends HTTP and uses encryption for secure communications.

4. **WebSocket**

WebSocket is a protocol that provides full-duplex communications over TCP. Clients establish WebSockets to receive real-time updates from the back-end services. Unlike REST, which always “pulls” data, WebSocket enables data to be “pushed”. Applications, like online gaming, stock trading, and messaging apps leverage WebSocket for real-time communication.

5. **TCP (Transmission Control Protocol)**

TCP is designed to send packets across the internet and ensure the successful delivery of data and messages over networks. Many application-layer protocols build on top of TCP.

6. **UDP (User Datagram Protocol)**

UDP sends packets directly to a target computer, without establishing a connection first. UDP is commonly used in time-sensitive communications where occasionally dropping packets is better than waiting. Voice and video traffic are often sent using this protocol.

7. **SMTP (Simple Mail Transfer Protocol)**

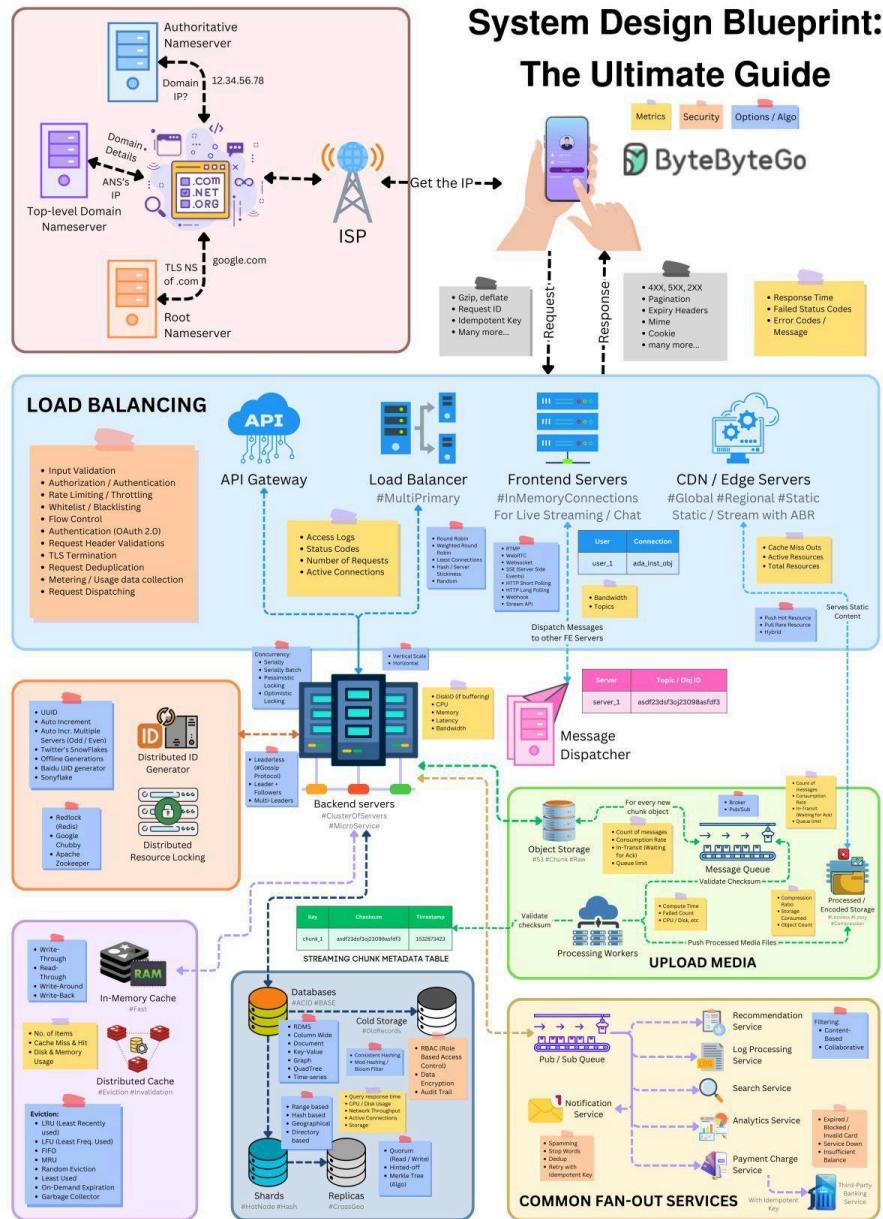
SMTP is a standard protocol to transfer electronic mail from one user to another.

8. **FTP (File Transfer Protocol)**

FTP is used to transfer computer files between client and server. It has separate connections for the control channel and data channel.

System Design Blueprint: The Ultimate Guide

We've created a template to tackle various system design problems in interviews.



Hope this checklist is useful to guide your discussions during the interview process.

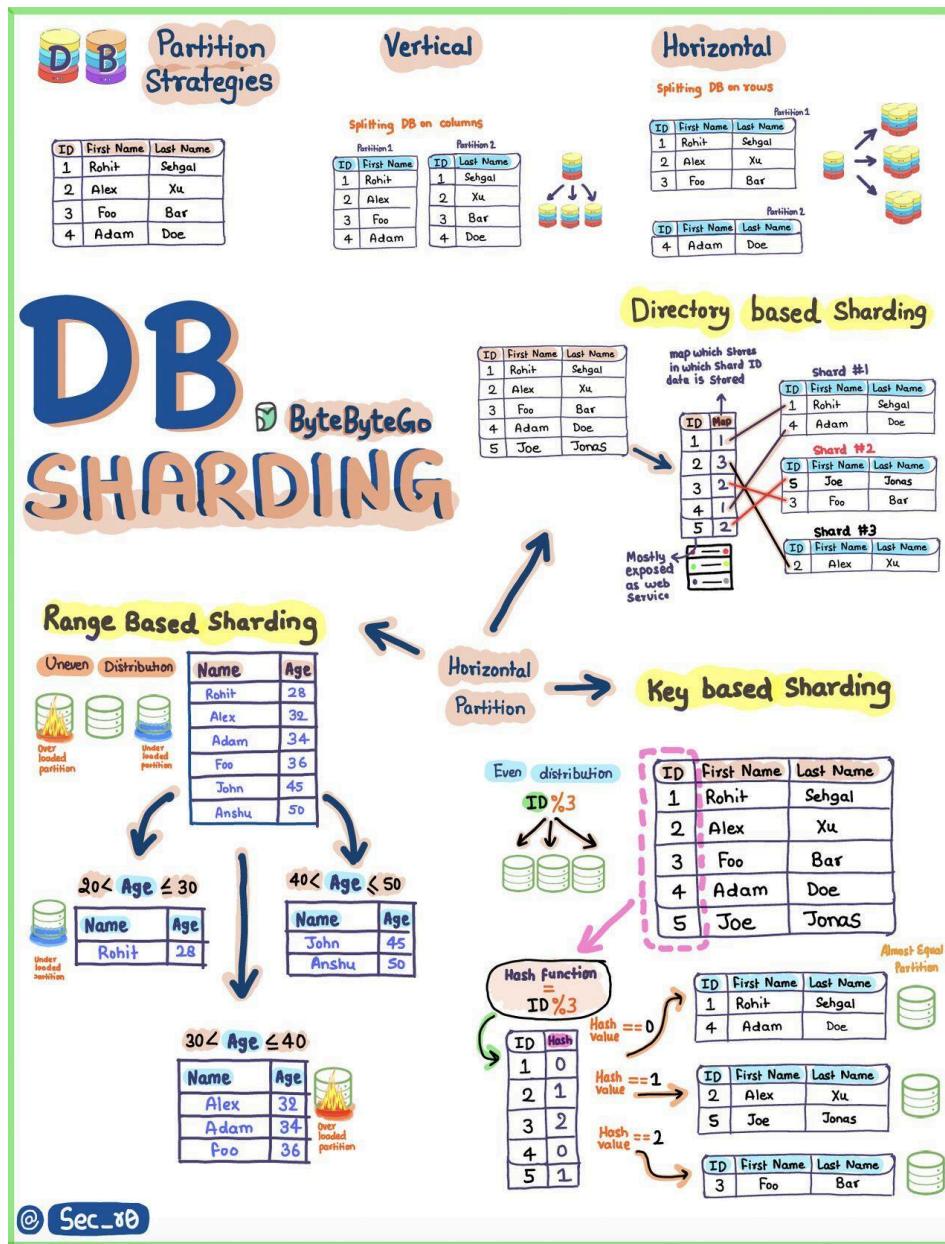
This briefly touches on the following discussion points:

- Load Balancing
- API Gateway

- Communication Protocols
- Content Delivery Network (CDN)
- Database
- Cache
- Message Queue
- Unique ID Generation
- Scalability
- Availability
- Performance
- Security
- Fault Tolerance and Resilience
- And more

Key Concepts to Understand Database Sharding

In this concise and visually engaging resource, we break down the key concepts of database partitioning, explaining both vertical and horizontal strategies.



1. Range-Based Sharding: Splitting your data into distinct ranges. Think of it as organizing your books by genre on separate shelves.

2. Key-Based Sharding (with a dash of %3 hash): Imagine each piece of data having a unique key, and we distribute them based on a specific rule. It's like sorting your playing cards by suit and number.
3. Directory-Based Sharding: A directory, like a phone book, helps you quickly find the information you need. Similarly, this technique uses a directory to route data efficiently.

Over to you: What are some other ways to scale a database?

A nice cheat sheet of different monitoring infrastructure in cloud services

This cheat sheet offers a concise yet comprehensive comparison of key monitoring elements across the three major cloud providers and open-source / 3rd party tools.

MONITORING CHEAT SHEET <small>blog.bytebytego.com</small>				
Element	aws	Google Cloud	Azure	Open Source / 3rd Party
Data Collection	Cloud Watch Cloud Watch Logs Cloud Trail Config Custom agents / Scripts	Cloud Monitoring Cloud Logging Cloud Audit Logs Custom agents / Scripts	Azure Monitor Azure Activity Log Azure Policy Security Center Custom agents / Scripts	ZABBIX Prometheus fluentd logstash splunk ELK telegraf Nagios Sensu
Data Storage	S3	Cloud Storage	Blob Storage	MINIO GLUSTER ceph
Data Analysis	CloudWatch Metrics Insights	Cloud Operations	Azure Monitor Metrics Explorer	Grafana kibana +ableau
Alerting	SNS	Cloud Monitoring Alerts	Azure Monitor Alerts	PagerDuty slack
Visualization	CloudWatch Dashboard QuickSight	Cloud Monitoring Dashboard Data Studio	Azure Monitor Dashboard Power BI	Grafana Superset Metabase tableau re-dash
Reporting and Compliance	Config Rules Trusted Advisor	Security Command Center	Policy Compliance Security Center Compliance	OpenSCAP CISOfy
Automation	Lambda Step Functions	Cloud Functions	Azure Functions Azure Automation	Jenkins ANSIBLE
Integration	CloudFormation CodePipeline	Cloud Deployment Manager Cloud Build	Azure Automation Azure DevOps	Pulumi Ansible Terraform GitLab Jenkins Travis CI
Feedback Loop	Well-Architected Tool	Well-Architected Framework	Well-Architected Framework	Scout APM Cloud Custodian

Let's delve into the essential monitoring aspects covered:

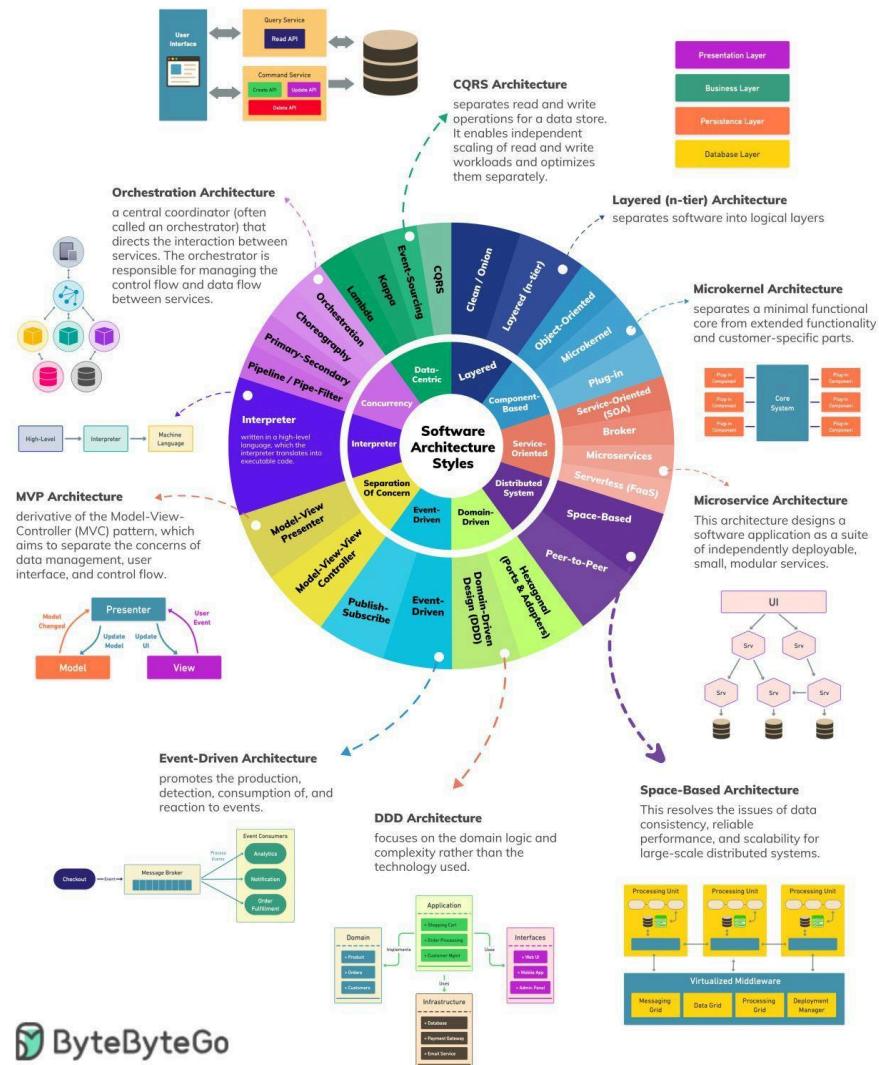
- Data Collection: Gather information from diverse sources to enhance decision-making.
- Data Storage: Safely store and manage data for future analysis and reference.
- Data Analysis: Extract valuable insights from data to drive informed actions.

- Alerting: Receive real-time notifications about critical events or anomalies.
- Visualization: Present data in a visually comprehensible format for better understanding.
- Reporting and Compliance: Generate reports and ensure adherence to regulatory standards.
- Automation: Streamline processes and tasks through automated workflows.
- Integration: Seamlessly connect and exchange data between different systems or tools.
- Feedback Loops: Continuously refine strategies based on feedback and performance analysis.

Over to you: How do you prioritize and leverage these essential monitoring aspects in your domain to achieve better outcomes and efficiency?

Top 5 Software Architectural Patterns

Software Architecture Styles

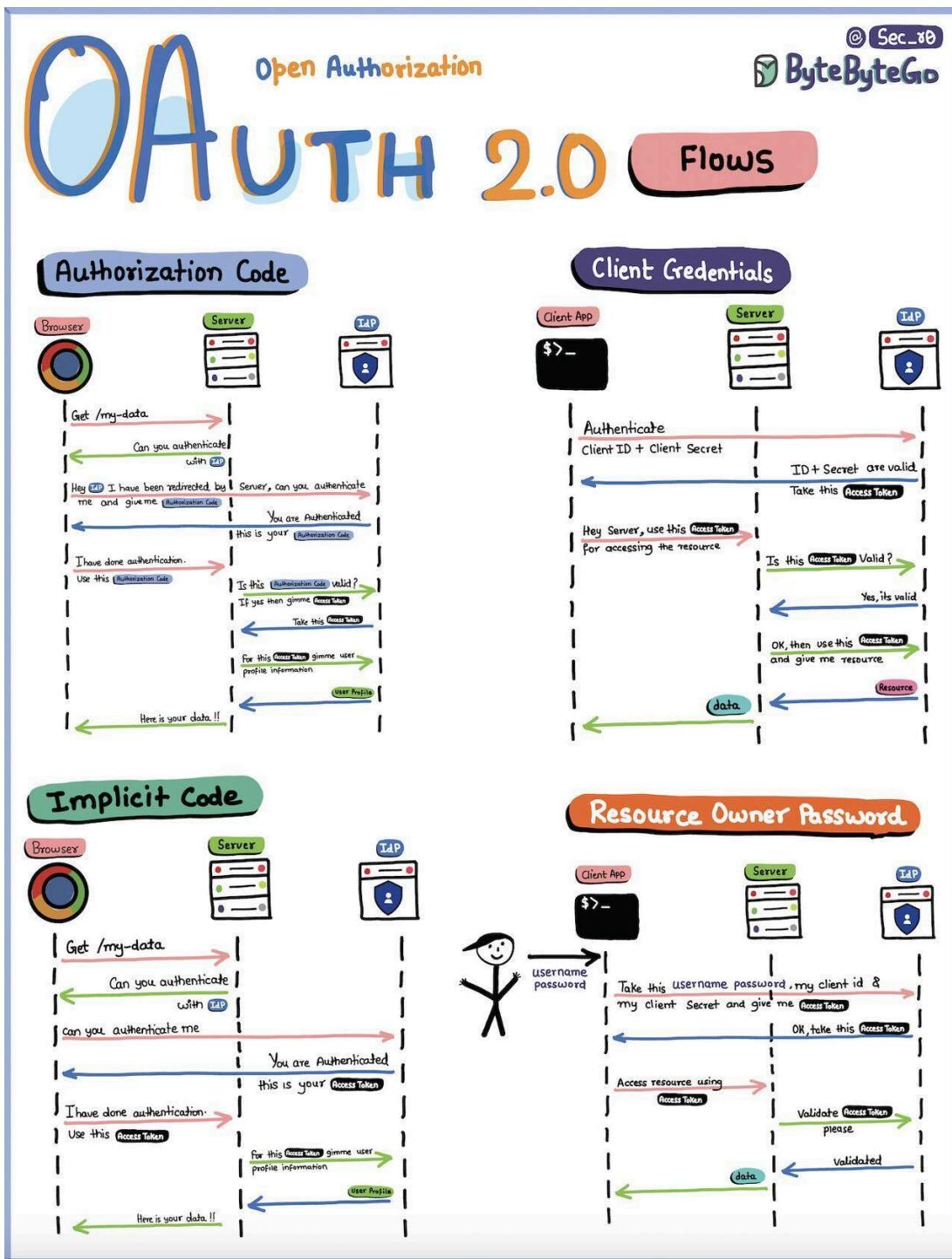


In software development, architecture plays a crucial role in shaping the structure and behavior of software systems. It provides a blueprint for system design, detailing how components interact with each other to deliver specific functionality. They also offer solutions to common problems, saving time and effort and leading to more robust and maintainable systems.

However, with the vast array of architectural styles and patterns available, it can take time to discern which approach best suits a particular project or system. Aims to shed light on these concepts, helping you make informed decisions in your architectural endeavors.

To help you navigate the vast landscape of architectural styles and patterns, there is a cheat sheet that encapsulates all. This cheat sheet is a handy reference guide that you can use to quickly recall the main characteristics of each architectural style and pattern.

OAuth 2.0 Flows



Authorization Code Flow: The most common OAuth flow. After user authentication, the client receives an authorization code and exchanges it for an access token and refresh token.

Client Credentials Flow: Designed for single-page applications. The access token is returned directly to the client without an intermediate authorization code.

Implicit Code Flow: Designed for single-page applications. The access token is returned directly to the client without an intermediate authorization code.

Resource Owner Password Grant Flow: Allows users to provide their username and password directly to the client, which then exchanges them for an access token.

Over to you - So which one do you think is something that you should use next in your application?

How did AWS grow from just a few services in 2006 to over 200 fully-featured services?

Let's take a look.

Since 2006, it has become a cloud computing leader, offering foundational infrastructure, platforms, and advanced capabilities like serverless computing and AI.



This expansion empowered innovation, allowing complex applications without extensive hardware management. AWS also explored edge and quantum computing, staying at tech's forefront.

This evolution mirrors cloud computing's shift from niche to essential, benefiting global businesses with efficiency and scalability

Happy to present the curated list of AWS services introduced over the years below.

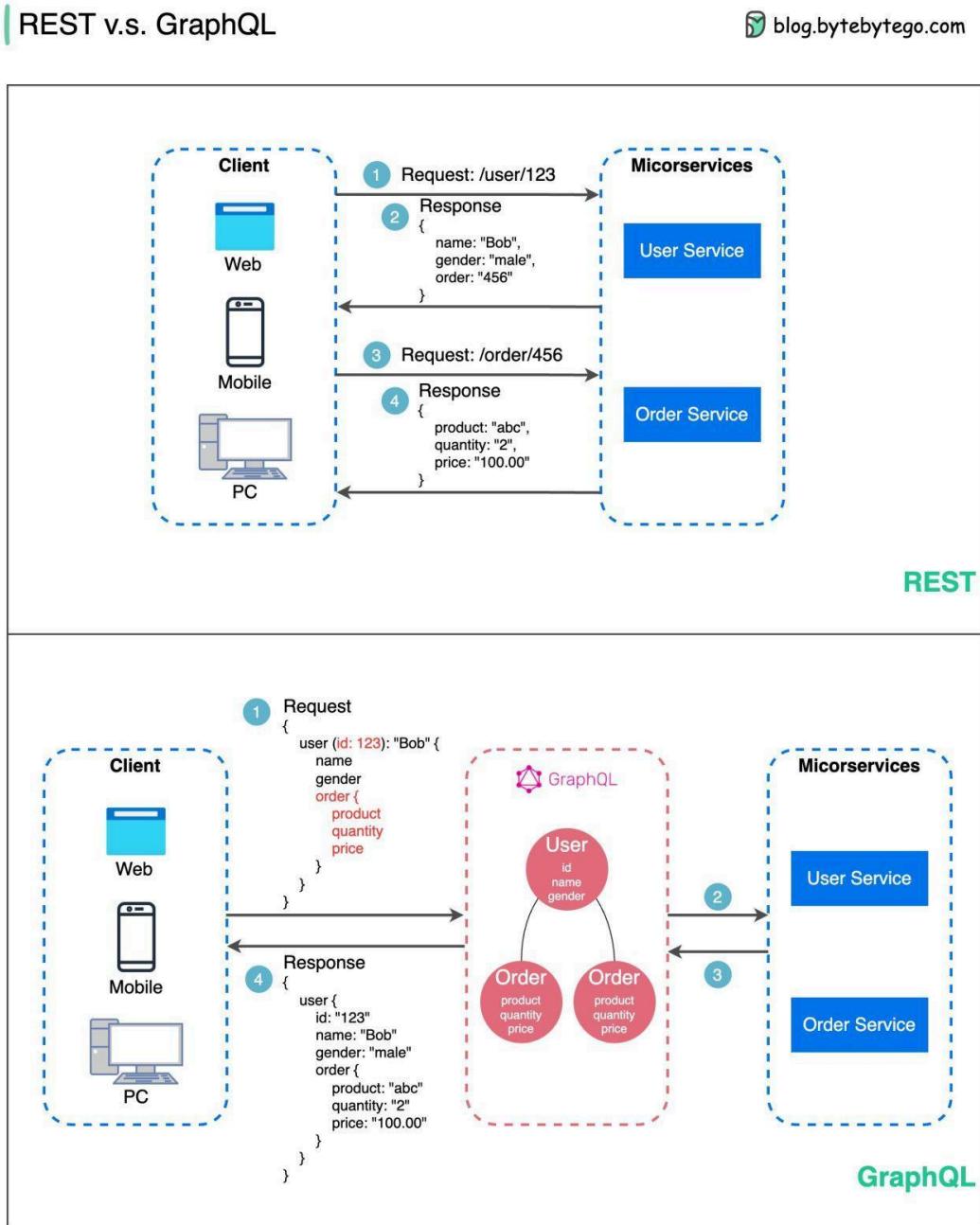
Note:

- The announcement or preview year differs from the public release year for certain services. In these cases, we've noted the service under the release year
- Unreleased services noted in announcement years

Over to you: Are you excited about all the new services, or do you find it overwhelming?

What is GraphQL? Is it a replacement for the REST API?

The diagram below shows the quick comparison between REST and GraphQL.



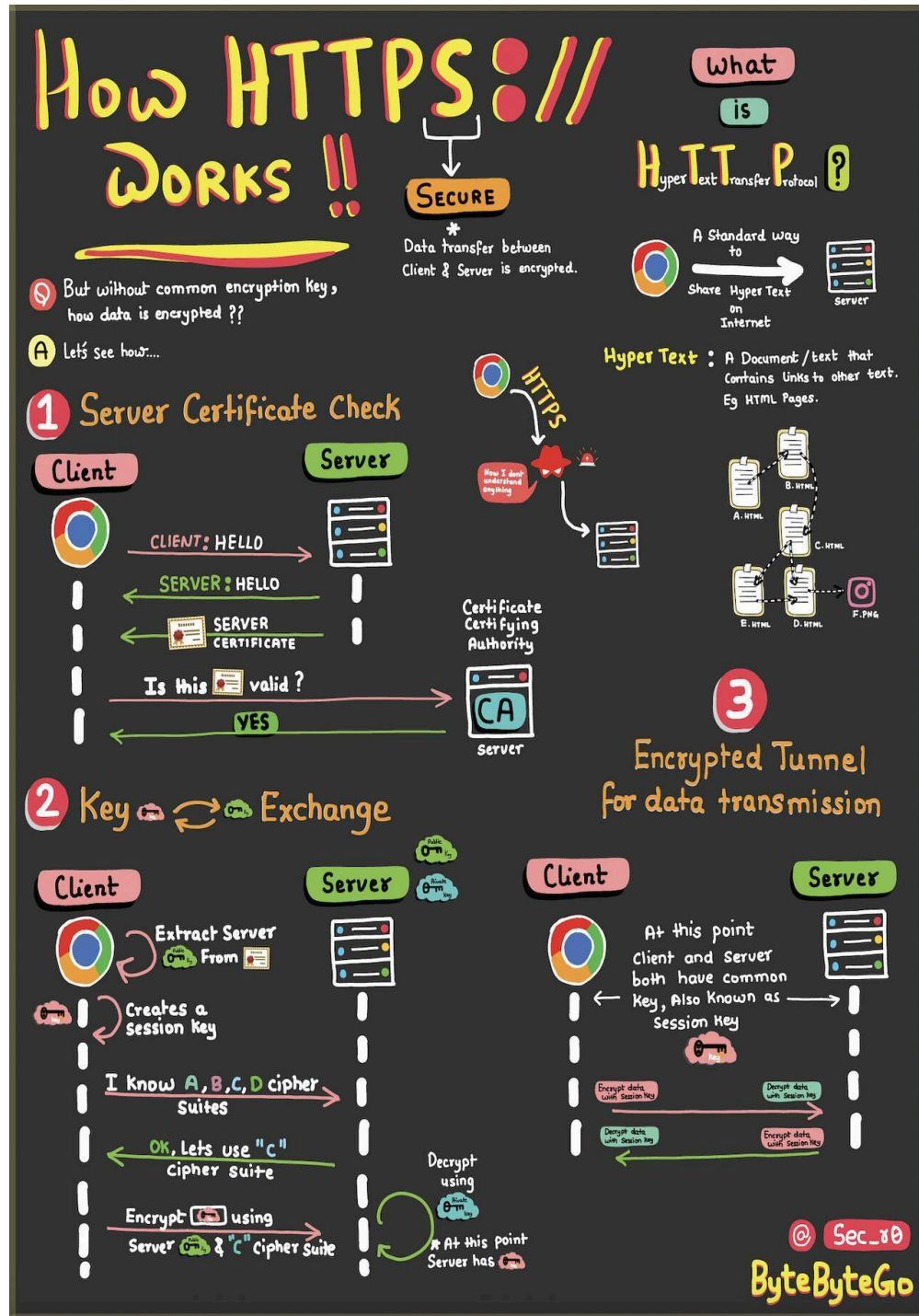
- ◆ GraphQL is a query language for APIs developed by Meta. It provides a complete description of the data in the API and gives clients the power to ask for exactly what they need.

- ◆ GraphQL servers sit in between the client and the backend services.
- ◆ GraphQL can aggregate multiple REST requests into one query. GraphQL server organizes the resources in a graph.
- ◆ GraphQL supports queries, mutations (applying data modifications to resources), and subscriptions (receiving notifications on schema modifications).

Over to you:

1. Is GraphQL a database technology?
2. Do you recommend GraphQL? Why/why not?

HTTPS, SSL Handshake, and Data Encryption Explained to Kids



HTTPS: Safeguards your data from eavesdroppers and breaches. Understand how encryption and digital certificates create an impregnable shield.

SSL Handshake: Behind the Scenes — Witness the cryptographic protocols that establish a secure connection. Experience the intricate exchange of keys and negotiation.

Secure Data Transmission: Navigating the Tunnel — Journey through the encrypted tunnel forged by HTTPS. Learn how your information travels while shielded from cyber threats.

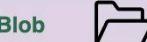
HTML's Role: Peek into HTML's role in structuring the web. Uncover how hyperlinks and content come together seamlessly. And why is it called HYPER TEXT.

Over to you: In this ever-evolving digital landscape, what emerging technologies do you foresee shaping the future of cybersecurity or the web?

A nice cheat sheet of different databases in cloud services

Cloud Database Cheat Sheet

 blog.bytebytogo.com

DB Type	AWS	Azure	Google Cloud	Open Source / 3rd Party
Structured	Relational 	RDS 	SQL Database 	Oracle 
	Columnar 	Redshift 	Synapse Analytics 	PostgreSQL 
	Key Value 	DynamoDB 	Cosmos DB 	MySQL 
	In-Memory 	ElastiCache 	Azure Cache for Redis 	SQL Server 
	Wide Column 	Keyspaces 	Cosmos DB 	Click House 
	Time Series 	Timestream 	Time Series Insights 	Redis 
	Immutable Ledger 	Quantum Ledger DB 	Confidential Ledger 	Scylla 
	Geospatial 	Keyspaces 	Cosmos DB 	Memcached 
	Graph 	Neptune 	Cosmos DB 	Cassandra 
	Document 	Document DB 	Cosmos DB 	Scylla 
Semi Structured	Text Search 	OpenSearch 	Cognitive Search 	Influx 
	Blob 	S3 	Blob Storage 	OpenTSDB 
Unstructured			Cloud Storage 	Hyper Ledger Fabric 
				PostGIS 
				geomesa 
				OrientDB 
				Dgraph 
				MongoDB 
				Couchbase 
				Elastic search 
				Elassandra 
				Ceph 
				OpenIO 

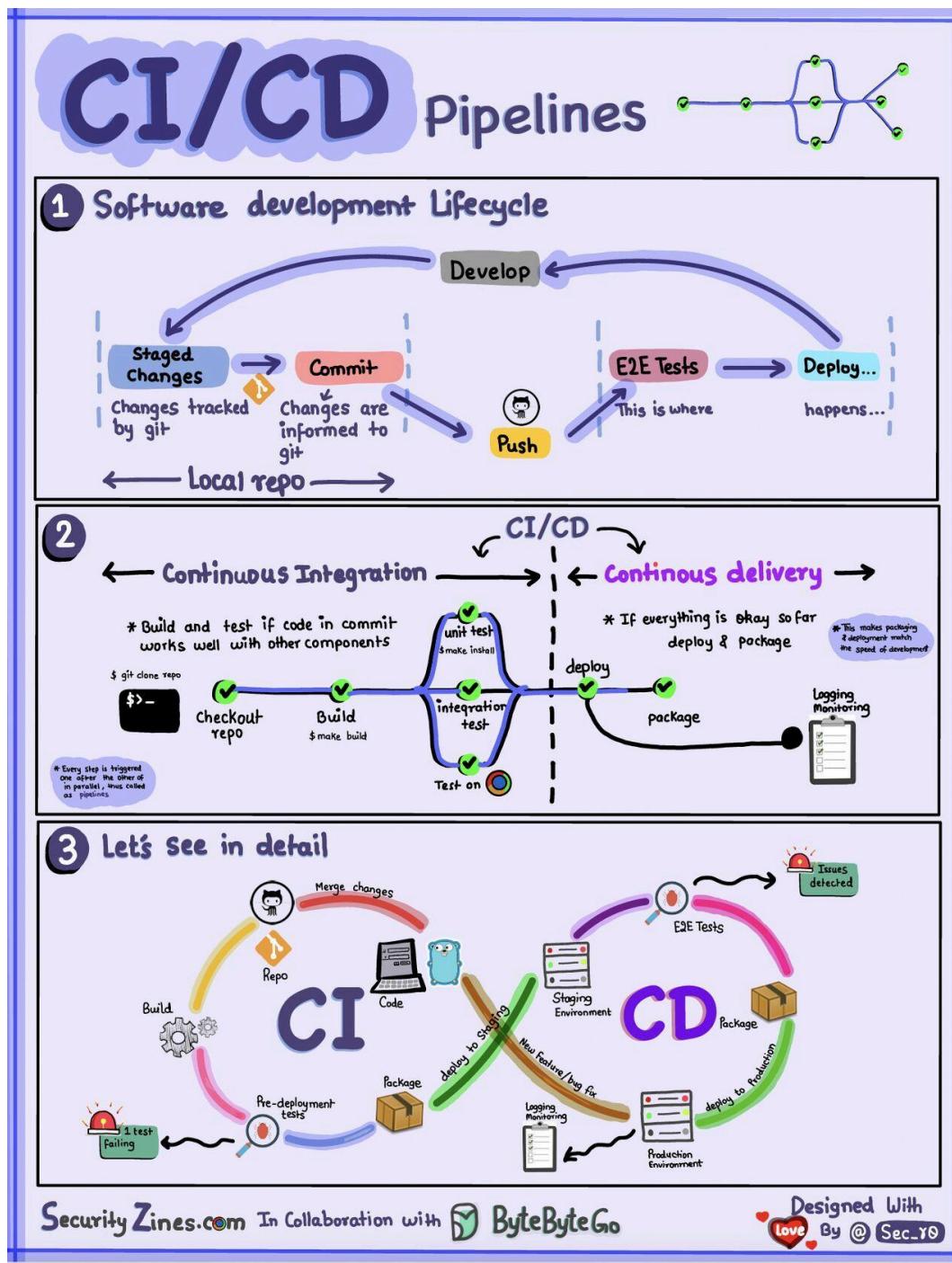
Choosing the right database for your project is a complex task. The multitude of database options, each suited to distinct use cases, can quickly lead to decision fatigue.

We hope this cheat sheet provides high level direction to pinpoint the right service that aligns with your project's needs and avoid potential pitfalls.

Note: Google has limited documentation for their database use cases. Even though we did our best to look at what was available and arrived at the best option, some of the entries may be not accurate.

Over to you: Which database have you used in the past, and for what use cases?

CI/CD Pipeline Explained in Simple Terms



Section 1 - SDLC with CI/CD

The software development life cycle (SDLC) consists of several key stages: development, testing, deployment, and maintenance. CI/CD automates and integrates these stages to enable faster, more reliable releases.

When code is pushed to a git repository, it triggers an automated build and test process. End-to-end (e2e) test cases are run to validate the code. If tests pass, the code can be automatically deployed to staging/production. If issues are found, the code is sent back to development for bug fixing. This automation provides fast feedback to developers and reduces risk of bugs in production.

Section 2 - Difference between CI and CD

Continuous Integration (CI) automates the build, test, and merge process. It runs tests whenever code is committed to detect integration issues early. This encourages frequent code commits and rapid feedback.

Continuous Delivery (CD) automates release processes like infrastructure changes and deployment. It ensures software can be released reliably at any time through automated workflows. CD may also automate the manual testing and approval steps required before production deployment.

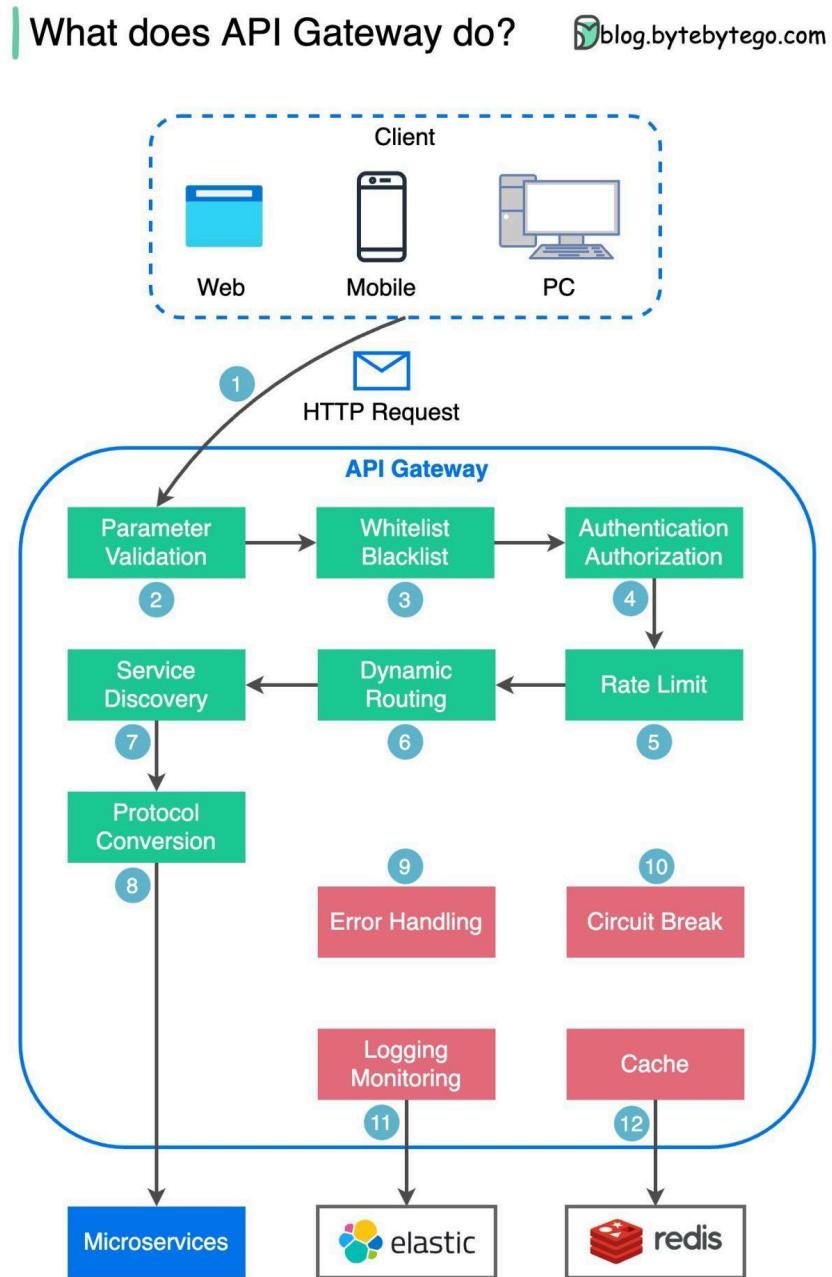
Section 3 - CI/CD Pipeline

A typical CI/CD pipeline has several connected stages:

- Developer commits code changes to source control
- CI server detects changes and triggers build
- Code is compiled, tested (unit, integration tests)
- Test results reported to developer
- On success, artifacts are deployed to staging environments
- Further testing may be done on staging before release
- CD system deploys approved changes to production

What does API gateway do?

The diagram below shows the detail.



Step 1 - The client sends an HTTP request to the API gateway.

Step 2 - The API gateway parses and validates the attributes in the HTTP request.

Step 3 - The API gateway performs allow-list/deny-list checks.

Step 4 - The API gateway talks to an identity provider for authentication and authorization.

Step 5 - The rate limiting rules are applied to the request. If it is over the limit, the request is rejected.

Steps 6 and 7 - Now that the request has passed basic checks, the API gateway finds the relevant service to route to by path matching.

Step 8 - The API gateway transforms the request into the appropriate protocol and sends it to backend microservices.

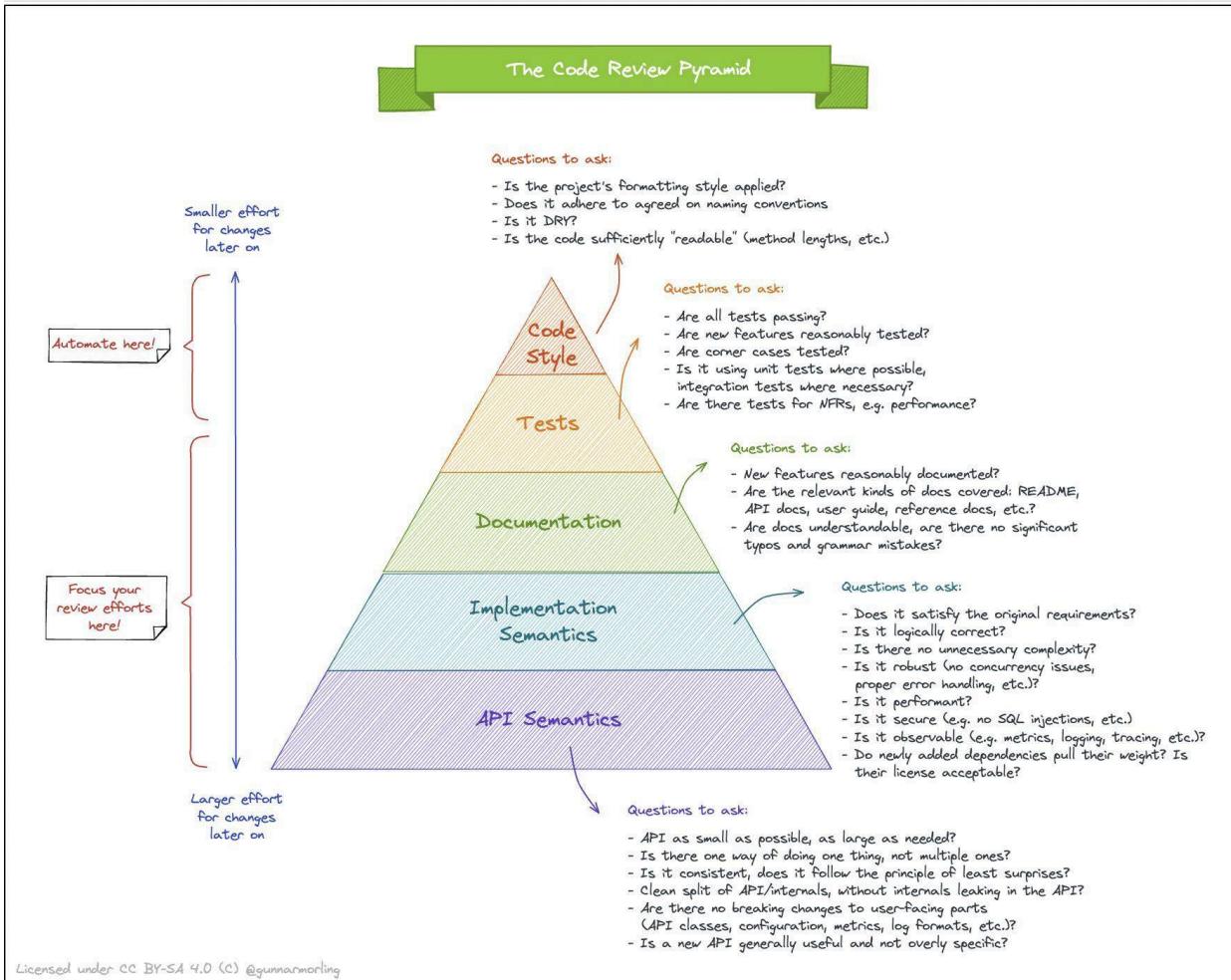
Steps 9-12: The API gateway can handle errors properly, and deals with faults if the error takes a longer time to recover (circuit break). It can also leverage ELK (Elastic-Logstash-Kibana) stack for logging and monitoring. We sometimes cache data in the API gateway.

Over to you:

1. What's the difference between a load balancer and an API gateway?
2. Do we need to use different API gateways for PC, mobile and browser separately?

The Code Review Pyramid

By [Gunnar Morling](#)

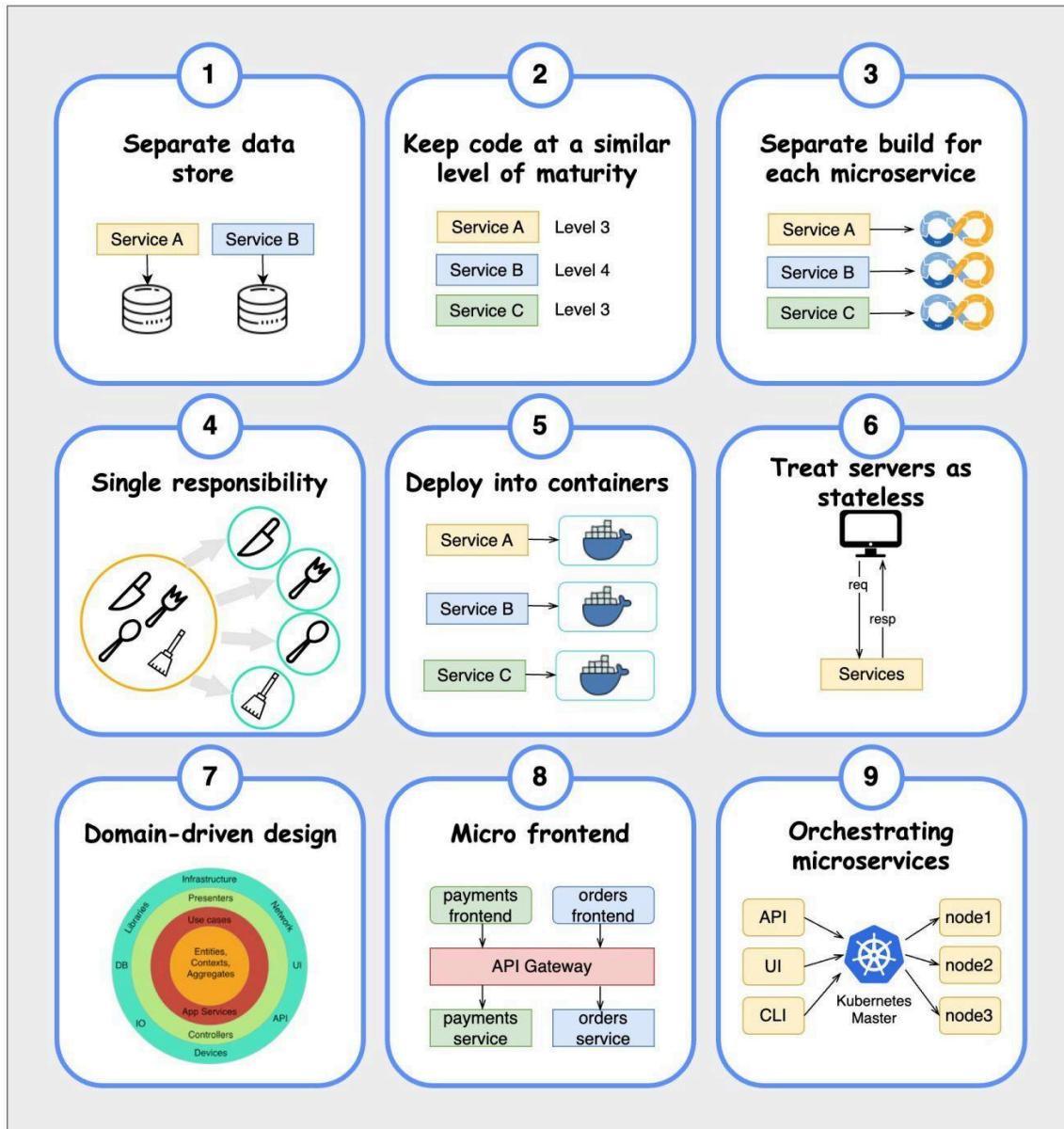


Over to you - Any other tips for effective code review?

A picture is worth a thousand words: 9 best practices for developing microservices

Microservice Best Practices

 blog.bytebytego.com



When we develop microservices, we need to follow the following best practices:

1. Use separate data storage for each microservice
2. Keep code at a similar level of maturity
3. Separate build for each microservice

4. Assign each microservice with a single responsibility
5. Deploy into containers
6. Design stateless services
7. Adopt domain-driven design
8. Design micro frontend
9. Orchestrating microservices

Over to you - what else should be included?

What are the greenest programming languages?

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

The study below runs 10 benchmark problems in 28 languages¹. It measures the runtime, memory usage, and energy consumption of each language. The abstract of the paper is shown below.

“This paper presents a study of the runtime, memory usage and energy consumption of twenty seven well-known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as, slower/faster languages consuming less/more energy, and how memory usage influences energy consumption. We show how to use our results to provide software engineers support to decide which language to use when energy efficiency is a concern”.²

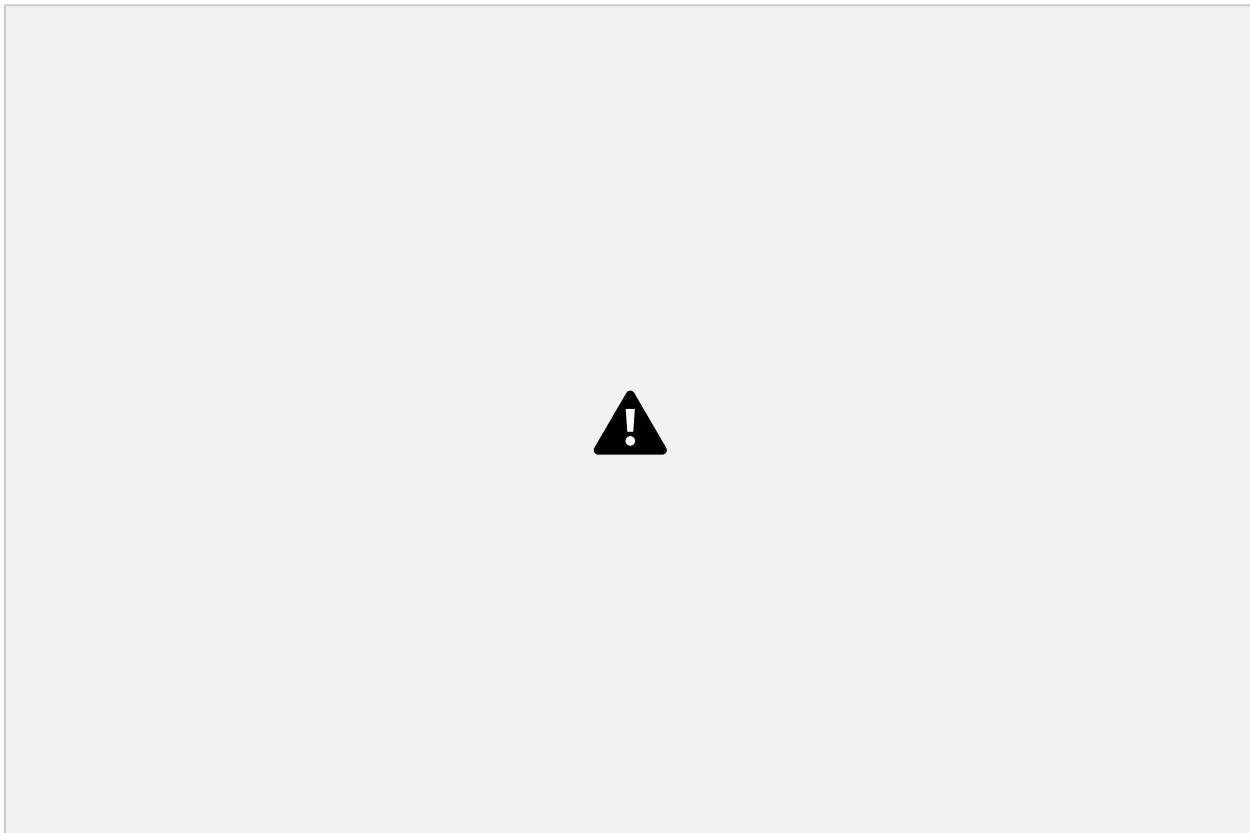
Most environmentally friendly languages: C, Rust, and C++

Least environmentally-friendly languages: Ruby, Python, Perl

Over to you: What do you think of the accuracy of this analysis?

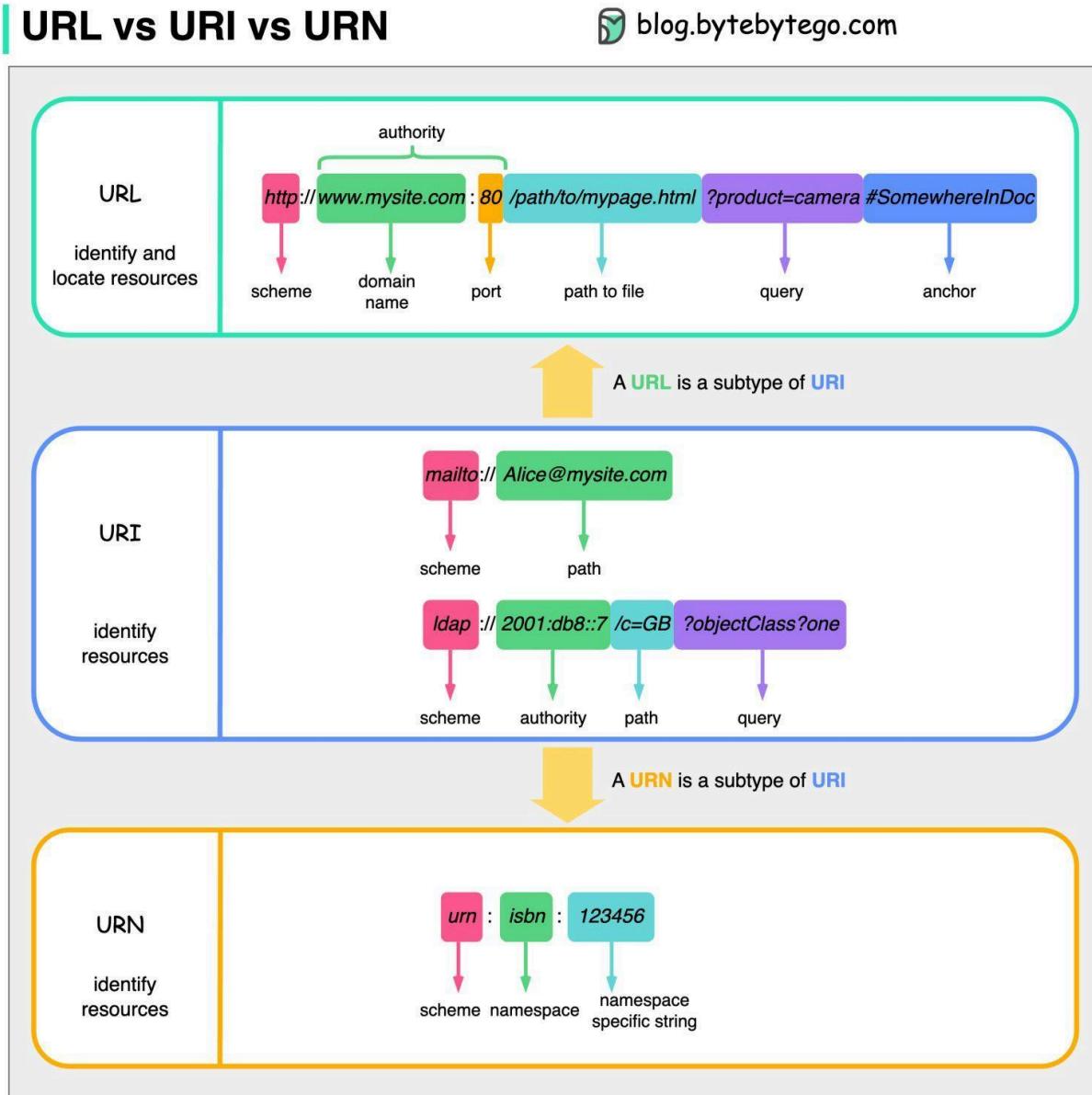
An amazing illustration of how to build a resilient three-tier architecture on AWS

Image Credit: [Ankit Jodhani](#)



URL, URI, URN - Do you know the differences?

The diagram below shows a comparison of URL, URI, and URN.



♦ URI

URI stands for Uniform Resource Identifier. It identifies a logical or physical resource on the web. URL and URN are subtypes of URI. URL locates a resource, while URN names a resource.

A URI is composed of the following parts:

`scheme://authority]path[?query][#fragment]`

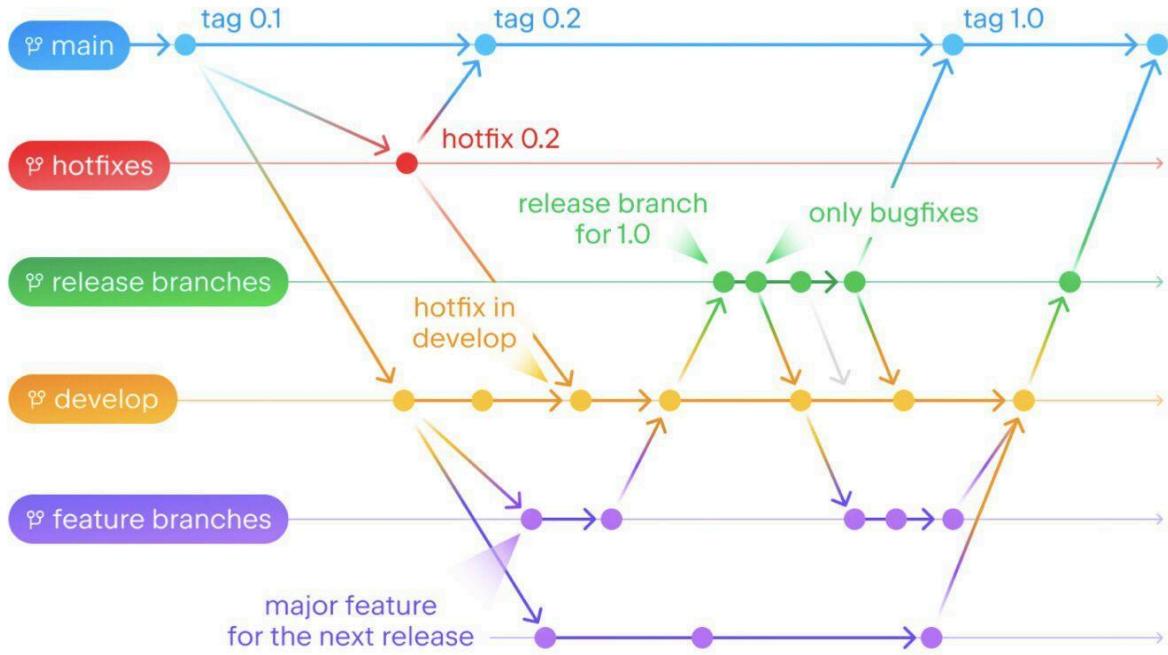
- ◆ URL
URL stands for Uniform Resource Locator, the key concept of HTTP. It is the address of a unique resource on the web. It can be used with other protocols like FTP and JDBC.
- ◆ URN
URN stands for Uniform Resource Name. It uses the urn scheme. URNs cannot be used to locate a resource. A simple example given in the diagram is composed of a namespace and a namespace-specific string.

If you would like to learn more detail on the subject, I would recommend W3C's clarification.

What branching strategies does your team use?

Git flow

Img source: <https://blog.jetbrains.com/space/2023/04/18/space-git-flow/>



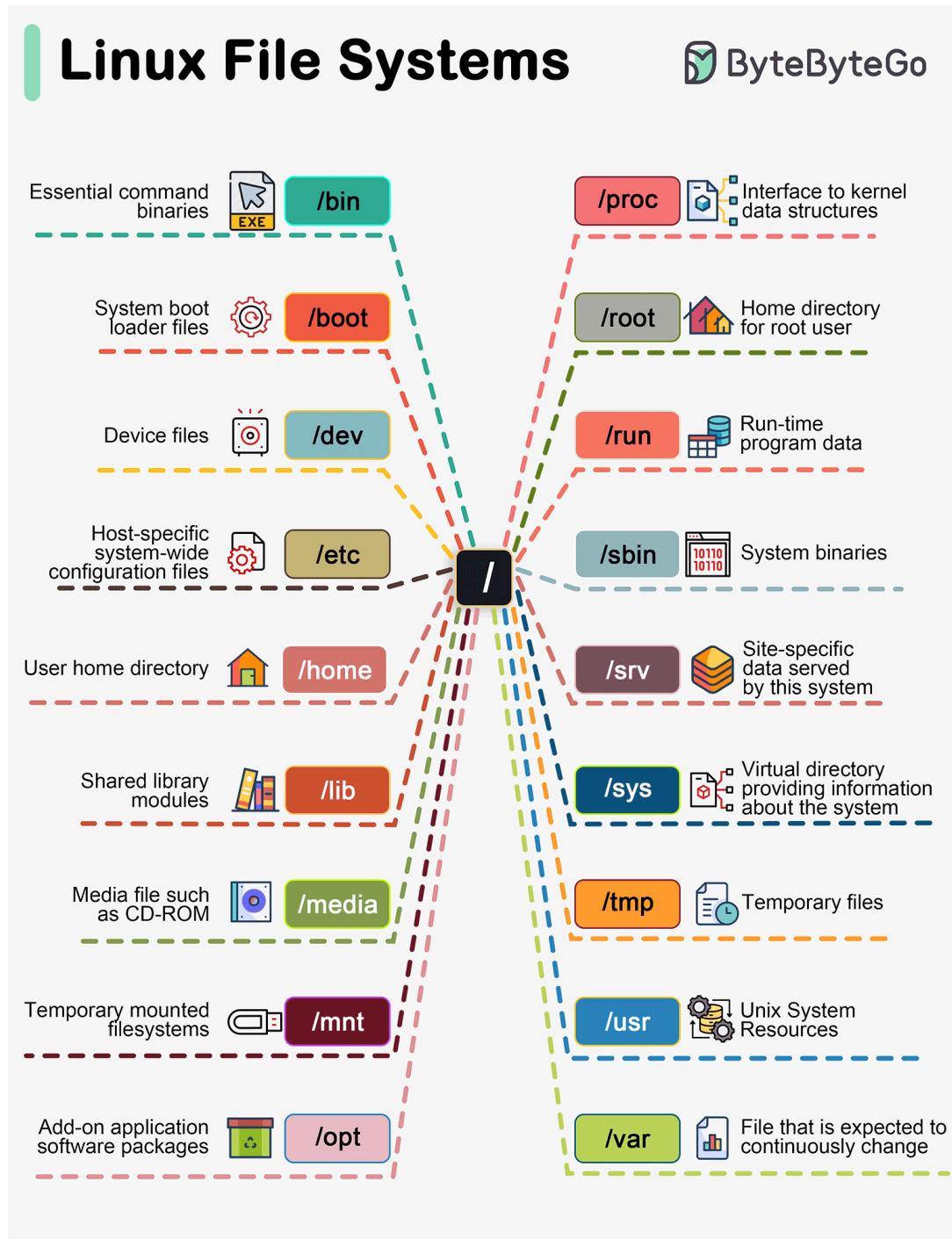
- The **main** branch is for production code only.
- The **develop** branch is for development code.
- **feature** branches are created from the **develop** branch.
- **hotfix** branches are created from the **main** branch.
- **release** branches are created from the **develop** branch.

Teams often employ various branching strategies for managing their code, such as Git flow, feature branches, and trunk-based development.

Out of these options, Git flow or its variations are the most widely favored methods. The illustration by Jetbrains explains how it works.

Linux file system explained

The Linux file system used to resemble an unorganized town where individuals constructed their houses wherever they pleased. However, in 1994, the Filesystem Hierarchy Standard (FHS) was introduced to bring order to the Linux file system.



By implementing a standard like the FHS, software can ensure a consistent layout across various Linux distributions. Nonetheless, not all Linux distributions strictly adhere to this standard. They often incorporate their own unique elements or cater to specific requirements.

To become proficient in this standard, you can begin by exploring. Utilize commands such as "cd" for navigation and "ls" for listing directory contents. Imagine the file system as a tree, starting from the root (/). With time, it will become second nature to you, transforming you into a skilled Linux administrator.

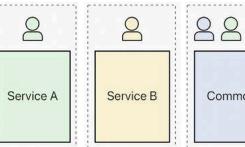
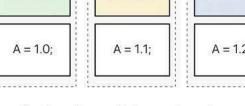
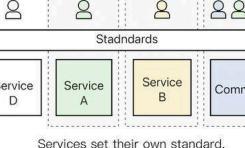
Have fun exploring!

Over to you: What Linux commands are useful for navigating and examining files?

Do you believe that Google, Meta, Uber, and Airbnb put almost all of their code in one repository?

This practice is called a monorepo.

Monorepo vs. Microrepo. Which is the best? Why do different companies choose different options?

Monorepo vs Microrepo: which is the best?		
	Monorepo	Microrepo
Company	 Google, Meta, Uber, Airbnb, Linux, Windows	
Collaboration	 Service A, Service B, Common Owners: 1 person (for each service) Common: 2 people (for both services) Services work under the same repository.	 Service A, Service B, Common Owners: 3 people (one for each service, one for common)
Dependency	 Service A, Service B, Common Dependency A = 1.1; Service share the same dependency.	 Service A, Service B, Common A = 1.0; A = 1.1; A = 1.2; Service choose their own dependency.
Scalability	 Standard: 1 person Services: Service A, Service B, Service D, Common Services share the same standard.	 Standard: 4 people (one for each service, one for common) Services: Service D, Service A, Service B, Common Services set their own standard.
Tooling	 Bazel, Buck, Nix, Lerna	 Gradle, nebula, Maven, npm, CMake

Monorepo isn't new; Linux and Windows were both created using Monorepo. To improve scalability and build speed, Google developed its internal dedicated toolchain to scale it faster and strict coding quality standards to keep it consistent.

Amazon and Netflix are major ambassadors of the Microservice philosophy. This approach naturally separates the service code into separate repositories. It scales faster but can lead to governance pain points later on.

Within Monorepo, each service is a folder, and every folder has a BUILD config and OWNERS permission control. Every service member is responsible for their own folder.

On the other hand, in Microrepo, each service is responsible for its repository, with the build config and permissions typically set for the entire repository.

In Monorepo, dependencies are shared across the entire codebase regardless of your business, so when there's a version upgrade, every codebase upgrades their version.

In Microrepo, dependencies are controlled within each repository. Businesses choose when to upgrade their versions based on their own schedules.

Monorepo has a standard for check-ins. Google's code review process is famously known for setting a high bar, ensuring a coherent quality standard for Monorepo, regardless of the business.

Microrepo can either set their own standard or adopt a shared standard by incorporating best practices. It can scale faster for business, but the code quality might be a bit different.

Google engineers built Bazel, and Meta built Buck. There are other open-source tools available, including Nix, Lerna, and others.

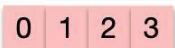
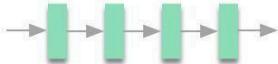
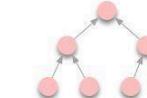
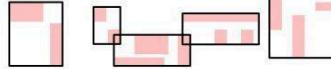
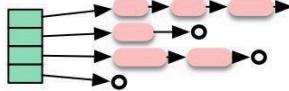
Over the years, Microrepo has had more supported tools, including Maven and Gradle for Java, NPM for NodeJS, and CMake for C/C++, among others.

Over to you: Which option do you think is better? Which code repository strategy does your company use?

What are the data structures used in daily life?

10 Data Structures Used in Daily Life

 ByteByteGo.com

Data Structure	Illustration	Use Cases
List		Twitter feeds
Array		Math operations Large data sets
Stack		Undo/Redo of word editor
Queue		Printer jobs User actions in game
Heap		Task scheduling
Tree		HTML document AI decision
Suffix Tree		Search string in document
Graph		Friendship tracking Path finding
R-tree		Nearest neighbour
Hash Table		Caching systems

- ◆ list: keep your Twitter feeds
- ◆ stack: support undo/redo of the word editor
- ◆ queue: keep printer jobs, or send user actions in-game
- ◆ heap: task scheduling
- ◆ tree: keep the HTML document, or for AI decision

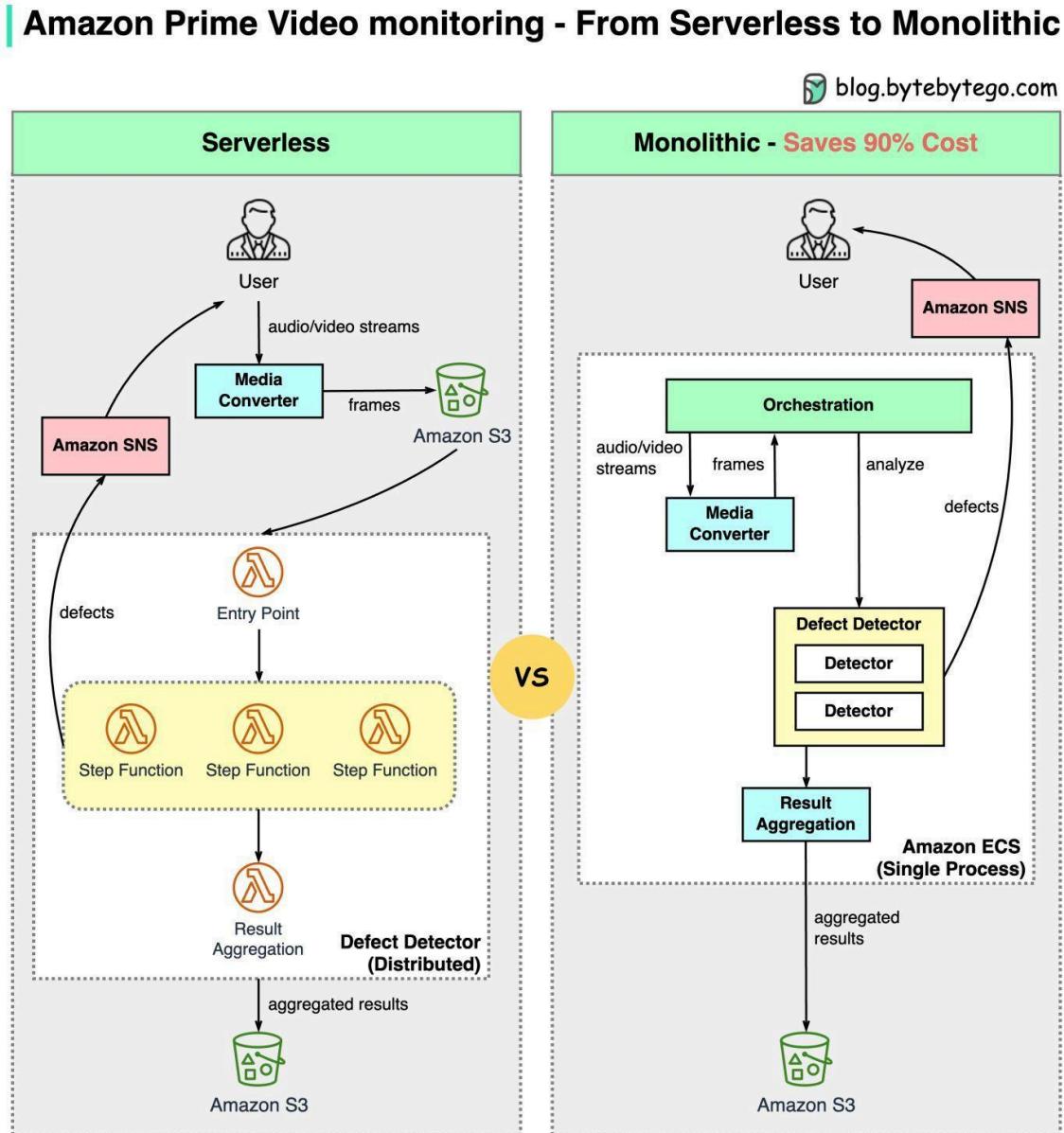
- ◆ suffix tree: for searching string in a document
- ◆ graph: for tracking friendship, or path finding
- ◆ r-tree: for finding the nearest neighbor
- ◆ vertex buffer: for sending data to GPU for rendering

To conclude, data structures play an important role in our daily lives, both in our technology and in our experiences. Engineers should be aware of these data structures and their use cases to create effective and efficient solutions.

Over to you: Which additional data structures have we overlooked?

Why did Amazon Prime Video monitoring move from serverless to monolithic? How can it save 90% cost?

The diagram below shows the architecture comparison before and after the migration.



What is Amazon Prime Video Monitoring Service?

Prime Video service needs to monitor the quality of thousands of live streams. The monitoring tool automatically analyzes the streams in real time and identifies quality issues like block

corruption, video freeze, and sync problems. This is an important process for customer satisfaction.

There are 3 steps: media converter, defect detector, and real-time notification.

- ◆ What is the problem with the old architecture?

The old architecture was based on Amazon Lambda, which was good for building services quickly. However, it was not cost-effective when running the architecture at a high scale. The two most expensive operations are:

1. The orchestration workflow - AWS step functions charge users by state transitions and the orchestration performs multiple state transitions every second.
2. Data passing between distributed components - the intermediate data is stored in Amazon S3 so that the next stage can download. The download can be costly when the volume is high.

- ◆ Monolithic architecture saves 90% cost

A monolithic architecture is designed to address the cost issues. There are still 3 components, but the media converter and defect detector are deployed in the same process, saving the cost of passing data over the network. Surprisingly, this approach to deployment architecture change led to 90% cost savings!

This is an interesting and unique case study because microservices have become a go-to and fashionable choice in the tech industry. It's good to see that we are having more discussions about evolving the architecture and having more honest discussions about its pros and cons. Decomposing components into distributed microservices comes with a cost.

- ◆ What did Amazon leaders say about this?

Amazon CTO Werner Vogels: “Building **evolvable software systems** is a strategy, not a religion. And revisiting your architectures with an open mind is a must.”

Ex Amazon VP Sustainability Adrian Cockcroft: “The Prime Video team had followed a path I call **Serverless First**...I don’t advocate **Serverless Only**”.

👉 Over to you: Does microservice architecture solve an architecture problem or an organizational problem?

18 Most-used Linux Commands You Should Know

18 Most-used Linux Commands			ByteByteGo.com
ls	cd	mkdir	
list files and directories	change current directory	create new directory	
rm	mv	chmod	
remove files or directories	move or rename files or change file or directory	change file or directories permission	
cp	find	grep	
copy files or directories	search for files or directories	search for a pattern in files	
vi	cat	tar	
edit files using text editor	display the content of files	manipulate tarball archive files	
ps	kill	top	
display process information	terminate process by sending a signal	display process and resource usage	
ifconfig	ping	du	
configure network interfaces	test network connectivity between hosts	estimate file space usage	

Linux commands are instructions for interacting with the operating system. They help manage files, directories, system processes, and many other aspects of the system. You need to become familiar with these commands in order to navigate and maintain Linux-based systems efficiently and effectively. The following are some popular Linux commands:

- ◆ ls - List files and directories
- ◆ cd - Change the current directory
- ◆ mkdir - Create a new directory
- ◆ rm - Remove files or directories
- ◆ cp - Copy files or directories
- ◆ mv - Move or rename files or directories
- ◆ chmod - Change file or directory permissions
- ◆ grep - Search for a pattern in files
- ◆ find - Search for files and directories
- ◆ tar - manipulate tarball archive files
- ◆ vi - Edit files using text editors
- ◆ cat - display the content of files
- ◆ top - Display processes and resource usage
- ◆ ps - Display processes information
- ◆ kill - Terminate a process by sending a signal
- ◆ du - Estimate file space usage
- ◆ ifconfig - Configure network interfaces
- ◆ ping - Test network connectivity between hosts

Over to you: What is your favorite Linux command?

Would it be nice if the code we wrote automatically turned into architecture diagrams?

I recently discovered a Github repo that does exactly this: Diagram as Code for prototyping cloud system architectures.

Diagram as Code

```
from diagrams import Cluster, Diagram
from diagrams.aws.compute import ECS
from diagrams.aws.database import ElastiCache, RDS
from diagrams.aws.network import ELB
from diagrams.aws.network import Route53

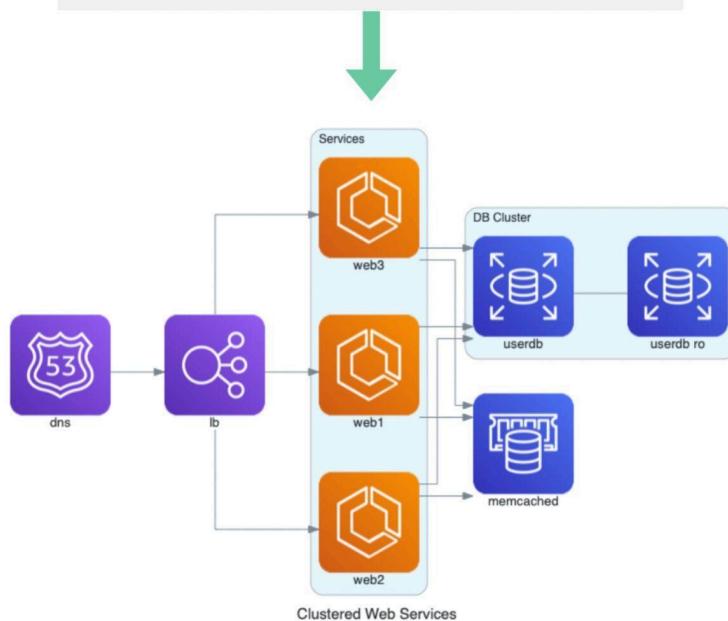
with Diagram("Clustered Web Services", show=False):
    dns = Route53("dns")
    lb = ELB("lb")

    with Cluster("Services"):
        svc_group = [ECS("web1"),
                     ECS("web2"),
                     ECS("web3")]

    with Cluster("DB Cluster"):
        db_primary = RDS("userdb")
        db_primary - [RDS("userdb ro")]

    memcached = ElastiCache("memcached")

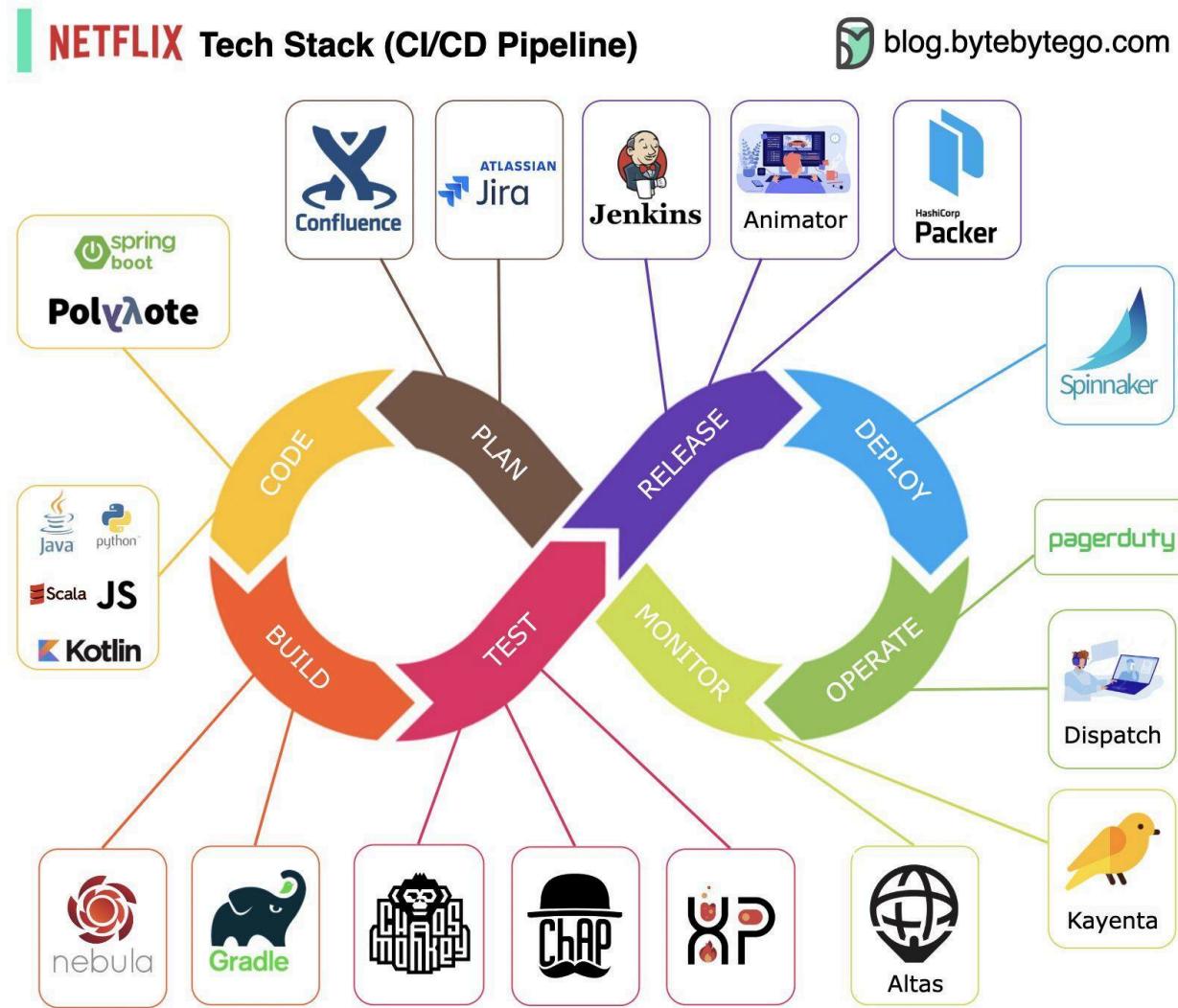
    dns >> lb >> svc_group
    svc_group >> db_primary
    svc_group >> memcached
```



What does it do?

- Draw the cloud system architecture in Python code.
- Diagrams can also be rendered directly inside the Jupyter Notebooks.
- No design tools are needed.
- Supports the following providers: AWS, Azure, GCP, Kubernetes, Oracle Cloud, etc.

Netflix Tech Stack - Part 1 (CI/CD Pipeline)



Planning: Netflix Engineering uses JIRA for planning and Confluence for documentation.

Coding: Java is the primary programming language for the backend service, while other languages are used for different use cases.

Build: Gradle is mainly used for building, and Gradle plugins are built to support various use cases.

Packaging: Package and dependencies are packed into an Amazon Machine Image (AMI) for release.

Testing: Testing emphasizes the production culture's focus on building chaos tools.

Deployment: Netflix uses its self-built Spinnaker for canary rollout deployment.

Monitoring: The monitoring metrics are centralized in Atlas, and Kayenta is used to detect anomalies.

Incident report: Incidents are dispatched according to priority, and PagerDuty is used for incident handling.

18 Key Design Patterns Every Developer Should Know

18 Key Design Patterns Every Developer Should Know		
Abstract Factory Family creator Create groups of related items	Builder Lego master Build object step by step	Prototype Cloner Create copies from examples
Singleton The one and only With just one instance	Adapter Universal plug Connect different interfaces	Bridge Connector Link what is to how it works
Composite Tree builder Create tree-like structure	Decorator Customizer Add new features to existing object	Facade One-stop shop Single interface to all functions
Flyweight Space saver Share small, reusable items	Proxy Middle man Represent another object	Chain of responsibility Replayer Relay requests until it is handles
Command Task wrapper Turn a request into object	Iterator Explorer Assess element one by one	Mediator Hub Simplify communication between classes
Memento Capsule Capture and store object state	Observer Broadcaster Notify others about the change	Visitor Guests Explore an object without changing it

Patterns are reusable solutions to common design problems, resulting in a smoother, more efficient development process. They serve as blueprints for building better software structures. These are some of the most popular patterns:

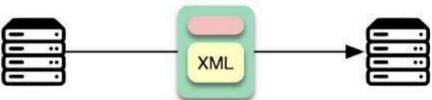
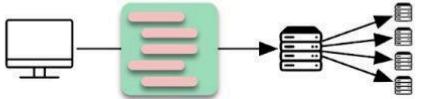
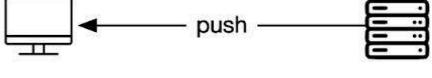
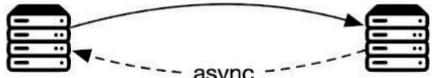
- ◆ Abstract Factory: Family Creator - Makes groups of related items.

- ◆ Builder: Lego Master - Builds objects step by step, keeping creation and appearance separate.
- ◆ Prototype: Clone Maker - Creates copies of fully prepared examples.
- ◆ Singleton: One and Only - A special class with just one instance.
- ◆ Adapter: Universal Plug - Connects things with different interfaces.
- ◆ Bridge: Function Connector - Links how an object works to what it does.
- ◆ Composite: Tree Builder - Forms tree-like structures of simple and complex parts.
- ◆ Decorator: Customizer - Adds features to objects without changing their core.
- ◆ Facade: One-Stop-Shop - Represents a whole system with a single, simplified interface.
- ◆ Flyweight: Space Saver - Shares small, reusable items efficiently.
- ◆ Proxy: Stand-In Actor - Represents another object, controlling access or actions.
- ◆ Chain of Responsibility: Request Relay - Passes a request through a chain of objects until handled.
- ◆ Command: Task Wrapper - Turns a request into an object, ready for action.
- ◆ Iterator: Collection Explorer - Accesses elements in a collection one by one.
- ◆ Mediator: Communication Hub - Simplifies interactions between different classes.
- ◆ Memento: Time Capsule - Captures and restores an object's state.
- ◆ Observer: News Broadcaster - Notifies classes about changes in other objects.
- ◆ Visitor: Skillful Guest - Adds new operations to a class without altering it.

How many API architecture styles do you know?

API Architecture Styles

 ByteByteGo.com

Style	Illustration	Use Cases
SOAP		XML-based for enterprise applications
RESTful		Resource-based for web servers
GraphQL		Query language reduce network load
gRPC		High performance for microservices
WebSocket		Bi-directional for low-latency data exchange
Webhook		Asynchronous for event-driven application

Architecture styles define how different components of an application programming interface (API) interact with one another. As a result, they ensure efficiency, reliability, and ease of integration with other systems by providing a standard approach to designing and building APIs. Here are the most used styles:

- ◆ SOAP:
Mature, comprehensive, XML-based
Best for enterprise applications
- ◆ RESTful:
Popular, easy-to-implement, HTTP methods
Ideal for web services

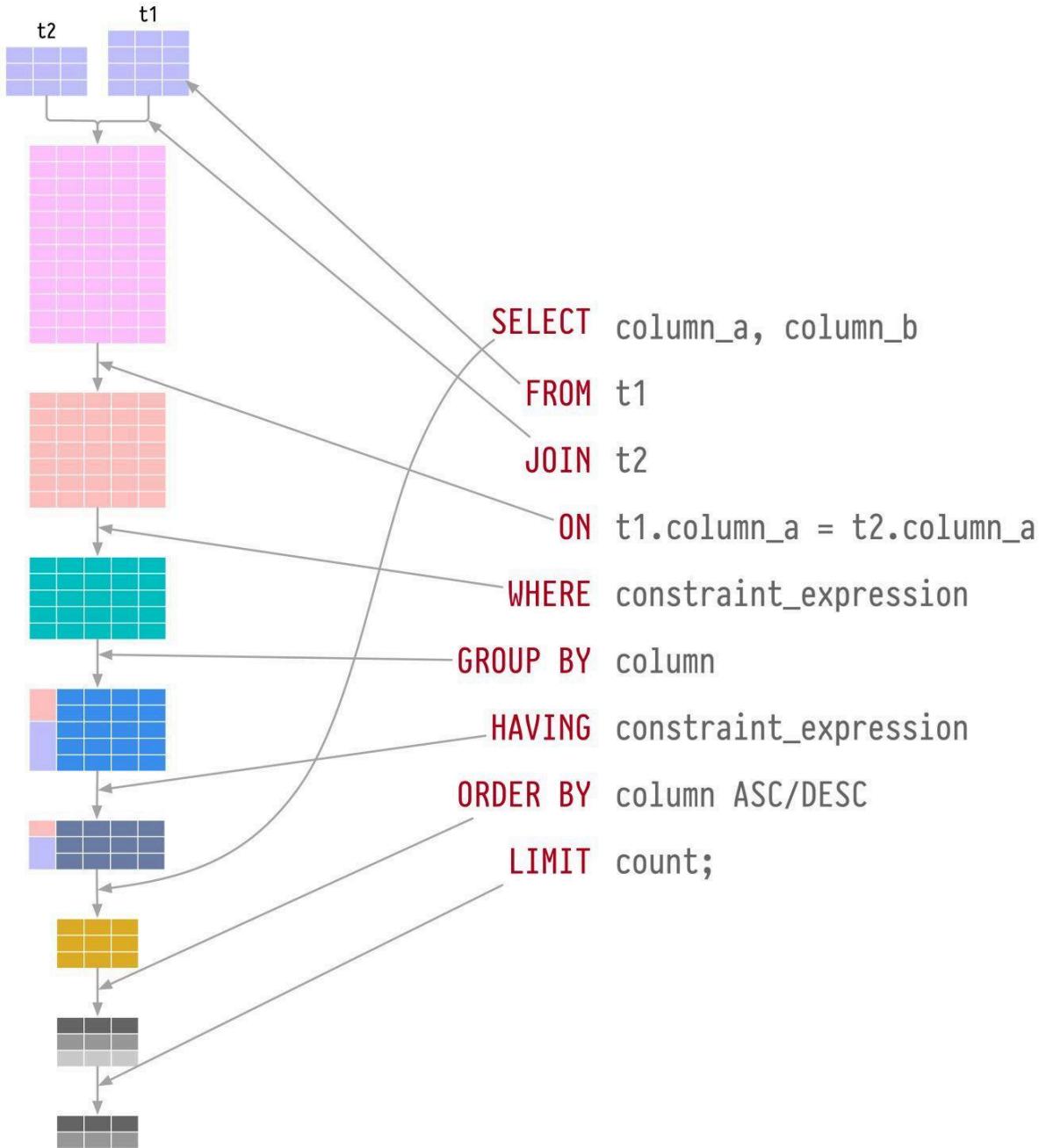
- ◆ GraphQL:
 - Query language, request specific data
 - Reduces network overhead, faster responses
- ◆ gRPC:
 - Modern, high-performance, Protocol Buffers
 - Suitable for microservices architectures
- ◆ WebSocket:
 - Real-time, bidirectional, persistent connections
 - Perfect for low-latency data exchange
- ◆ Webhook:
 - Event-driven, HTTP callbacks, asynchronous
 - Notifies systems when events occur

Over to you: Are there any other famous styles we missed?

Visualizing a SQL query

SQL Query Execution Order

 ByteByteGo.com



SQL statements are executed by the database system in several steps, including:

- Parsing the SQL statement and checking its validity
- Transforming the SQL into an internal representation, such as relational algebra

- Optimizing the internal representation and creating an execution plan that utilizes index information
- Executing the plan and returning the results

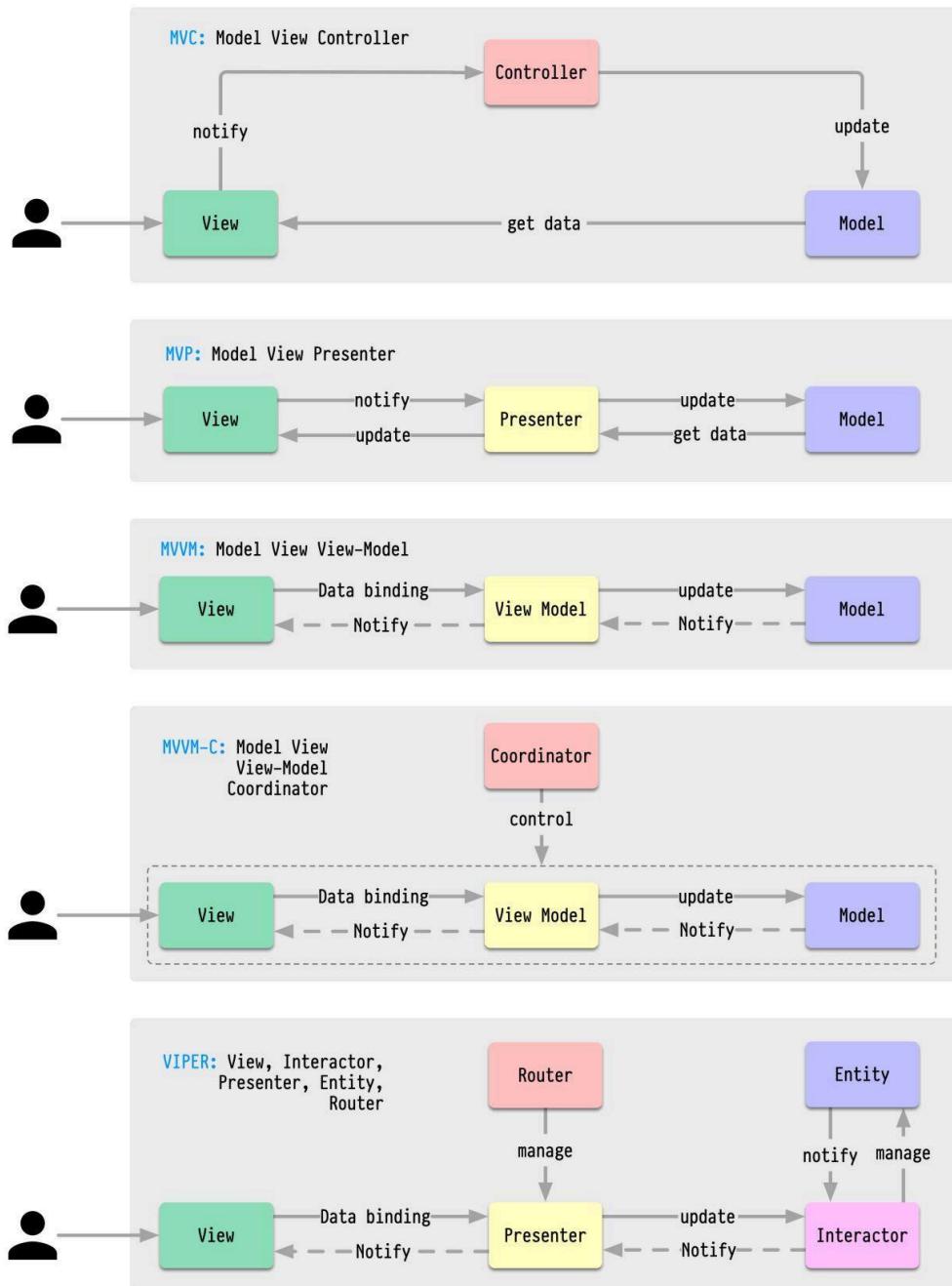
The execution of SQL is highly complex and involves many considerations, such as:

- The use of indexes and caches
- The order of table joins
- Concurrency control
- Transaction management

Over to you: what is your favorite SQL statement?

What distinguishes MVC, MVP, MVVM, MVVM-C, and VIPER architecture patterns from each other?

MVC, MVP, MVVM, VIPER patterns  ByteByteGo.com



These architecture patterns are among the most commonly used in app development, whether on iOS or Android platforms. Developers have introduced them to overcome the limitations of earlier patterns. So, how do they differ?

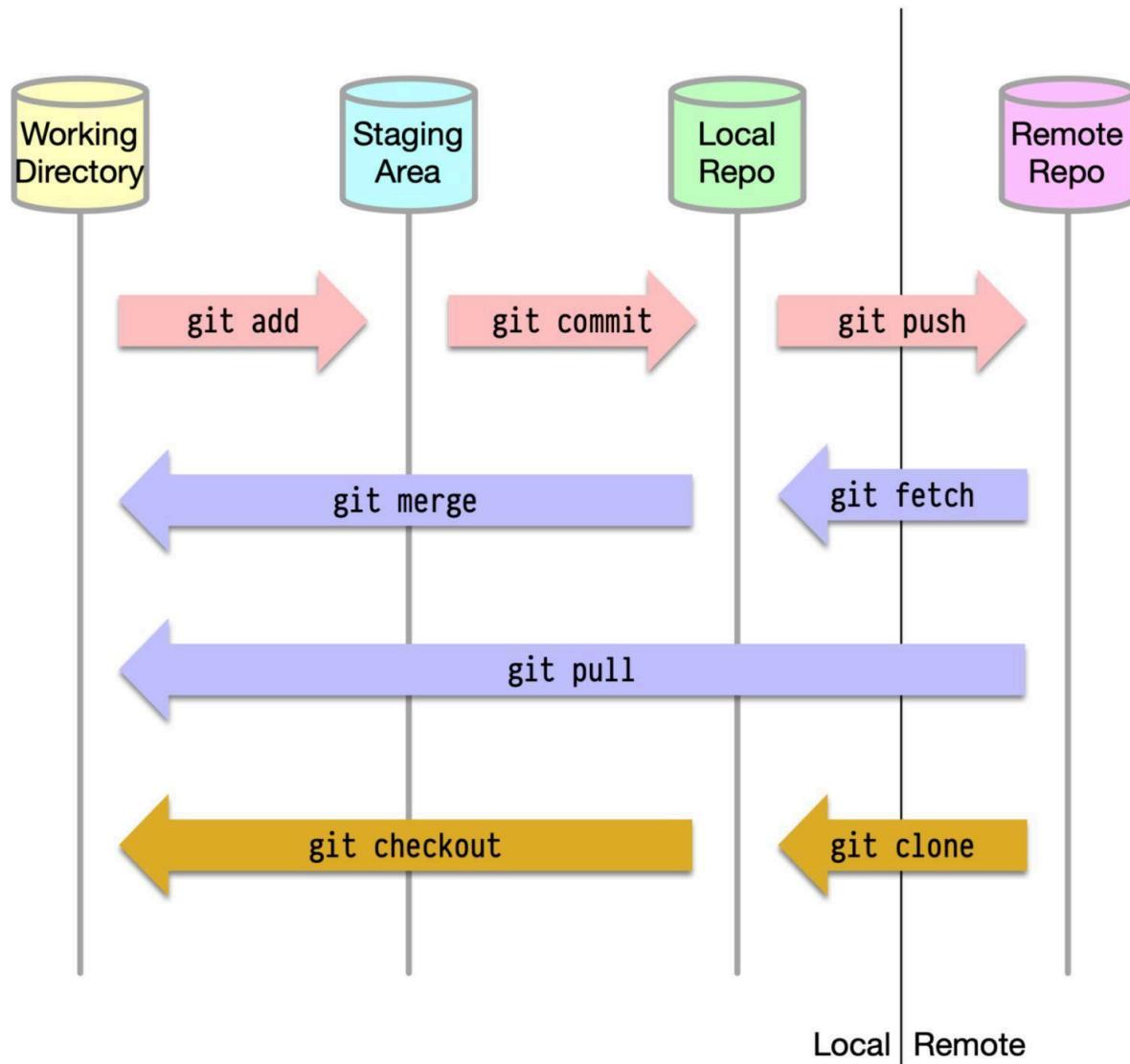
- ◆ MVC, the oldest pattern, dates back almost 50 years
- ◆ Every pattern has a "view" (V) responsible for displaying content and receiving user input
- ◆ Most patterns include a "model" (M) to manage business data
- ◆ "Controller," "presenter," and "view-model" are translators that mediate between the view and the model ("entity" in the VIPER pattern)
- ◆ These translators can be quite complex to write, so various patterns have been proposed to make them more maintainable

Over to you: keep in mind that this is not an exhaustive list of architecture patterns. Other notable patterns include Flux and Redux. How do they compare to the ones mentioned here?

Almost every software engineer has used Git before, but only a handful know how it works :)

How Git Commands work

 ByteByteGo.com



To begin with, it's essential to identify where our code is stored. The common assumption is that there are only two locations - one on a remote server like Github and the other on our local machine. However, this isn't entirely accurate. Git maintains three local storages on our machine, which means that our code can be found in four places:

- ◆ Working directory: where we edit files
- ◆ Staging area: a temporary location where files are kept for the next commit

- ◆ Local repository: contains the code that has been committed
- ◆ Remote repository: the remote server that stores the code

Most Git commands primarily move files between these four locations.

Over to you: Do you know which storage location the "git tag" command operates on? This command can add annotations to a commit.

I read something unbelievable today: Levels.fyi scaled to millions of users using Google Sheets as a backend!

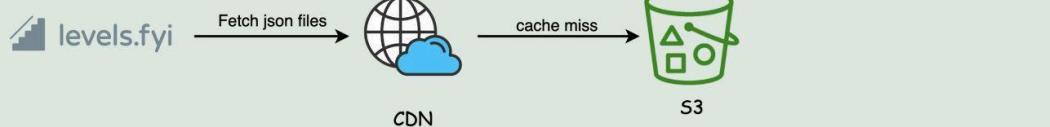
Levels.fyi scaled to millions with Google Sheets as backend

Add salary flow

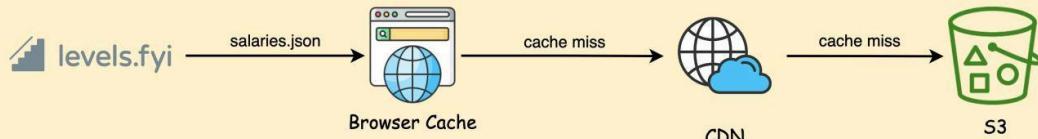


Read flow

Text



Caching strategy



Source: <https://www.levels.fyi/blog/scaling-to-millions-with-google-sheets.html>

They started off on Google Forms and Sheets, which helped them reach millions of monthly active users before switching to a proper backend.

To be fair, they do use serverless computing, but using Google Sheets as the database is an interesting choice.

Why do they use Google Sheets as a backend? Using their own words: "It seems like a pretty

counterintuitive idea for a site with our traffic volume to not have a backend or any fancy infrastructure, but our philosophy to building products has always been, start simple and iterate. This allows us to move fast and focus on what's important".

What are your thoughts? The link to the original article is embedded at the bottom of the diagram.

Best ways to test system functionality

Testing system functionality is a crucial step in software development and engineering processes.

It ensures that a system or software application performs as expected, meets user requirements, and operates reliably.

Best Ways To Test System Functionality		
Process	Illustration	Tools
Unit Testing		 pytest JUnit 5 nunit
Integration Testing		 POSTMAN cucumber SoapUI Selenium
System Testing		 Selenium Robot Framework appium JMeter™
Load Testing		 APACHE JMeter™ Gatling Locust LOAD RUNNER
Error Testing		 Gremlin
Test Automation		 Jenkins Travis CI circleci GitHub Actions

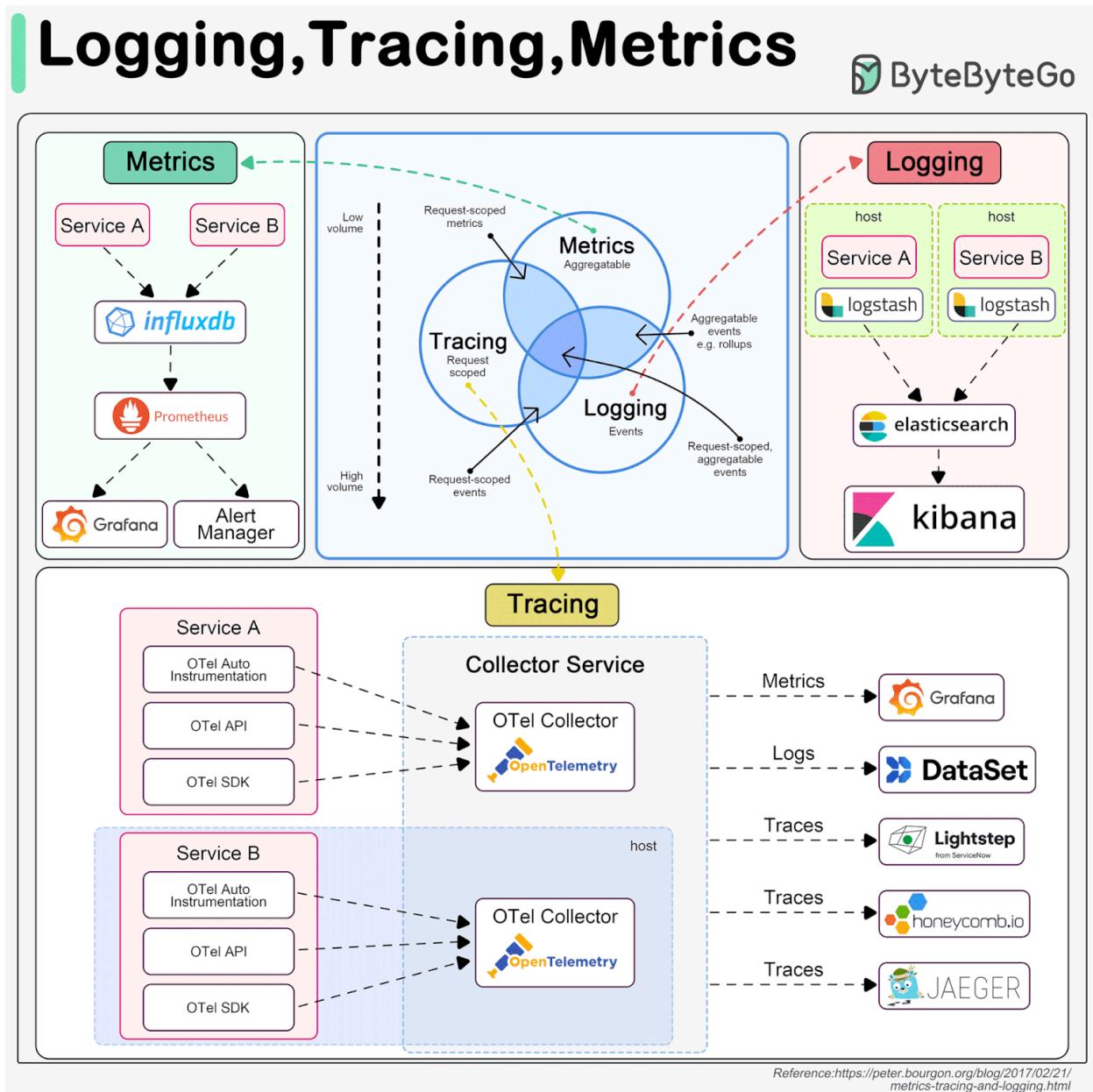
Here we delve into the best ways:

1. Unit Testing: Ensures individual code components work correctly in isolation.
2. Integration Testing: Verifies that different system parts function seamlessly together.
3. System Testing: Assesses the entire system's compliance with user requirements and performance.
4. Load Testing: Tests a system's ability to handle high workloads and identifies performance issues.
5. Error Testing: Evaluates how the software handles invalid inputs and error conditions.
6. Test Automation: Automates test case execution for efficiency, repeatability, and error reduction.

Over to you: How do you approach testing system functionality in your software development or engineering projects?

Logging, tracing and metrics are 3 pillars of system observability

The diagram below shows their definitions and typical architectures.



- Logging

Logging records discrete events in the system. For example, we can record an incoming request or a visit to databases as events. It has the highest volume. ELK (Elastic-Logstash-Kibana) stack is often used to build a log analysis platform. We often define a standardized logging format for different teams to implement, so that we can leverage keywords when searching among massive amounts of logs.

- Tracing

Tracing is usually request-scoped. For example, a user request goes through the API gateway, load balancer, service A, service B, and database, which can be visualized in the tracing systems. This is useful when we are trying to identify the bottlenecks in the system. We use OpenTelemetry to showcase the typical architecture, which unifies the 3 pillars in a single framework.

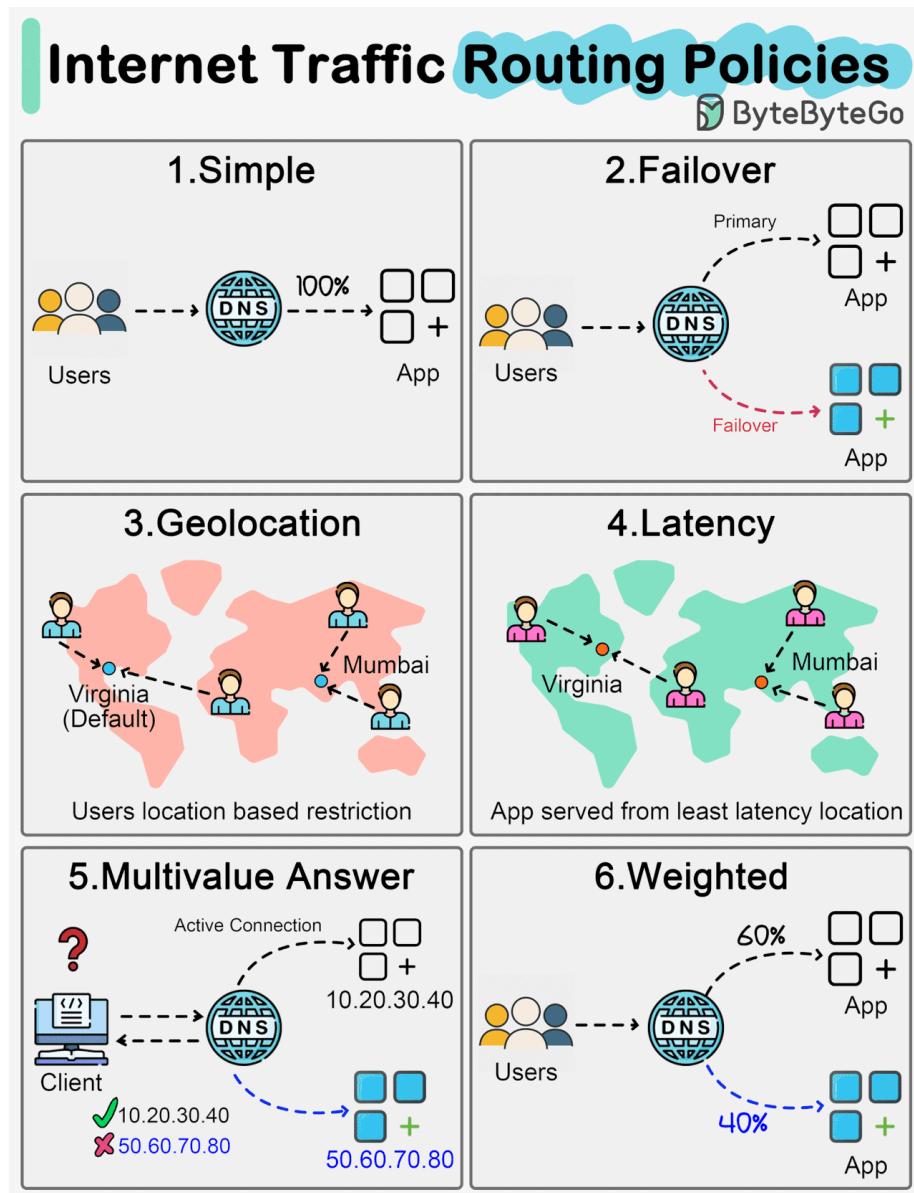
- Metrics

Metrics are usually aggregatable information from the system. For example, service QPS, API responsiveness, service latency, etc. The raw data is recorded in time-series databases like InfluxDB. Prometheus pulls the data and transforms the data based on pre-defined alerting rules. Then the data is sent to Grafana for display or to the alert manager which then sends out email, SMS, or Slack notifications or alerts.

Over to you: Which tools have you used for system monitoring?

Internet Traffic Routing Policies

Internet traffic routing policies (DNS policies) play a crucial role in efficiently managing and directing network traffic. Let's discuss the different types of policies.



1. Simple: Directs all traffic to a single endpoint based on a standard DNS query without any special conditions or requirements.
2. Failover: Routes traffic to a primary endpoint but automatically switches to a secondary endpoint if the primary is unavailable.

3. Geolocation: Distributes traffic based on the geographic location of the requester, aiming to provide localized content or services.
4. Latency: Directs traffic to the endpoint that provides the lowest latency for the requester, enhancing user experience with faster response times.
5. Multivalue Answer: Responds to DNS queries with multiple IP addresses, allowing the client to select an endpoint. However, it should not be considered a replacement for a load balancer.
6. Weighted Routing Policy: Distributes traffic across multiple endpoints with assigned weights, allowing for proportional traffic distribution based on these weights.

Over to you: Which DNS policy do you find most relevant to your network management needs?

Subjects that should be mandatory in schools

In the age of AI, what subjects should be taught in schools?

An interesting list of subjects that should be mandatory in schools by startup_rules.

Subjects That Should Be Mandatory In Schools



Taxes



Coding



Cooking



Insurance



Basic Home Repaire



Self Defense



Survival Skills



Social Etiquette



Personal Finance



Public Speaking



Car Maintenance



Stress Management

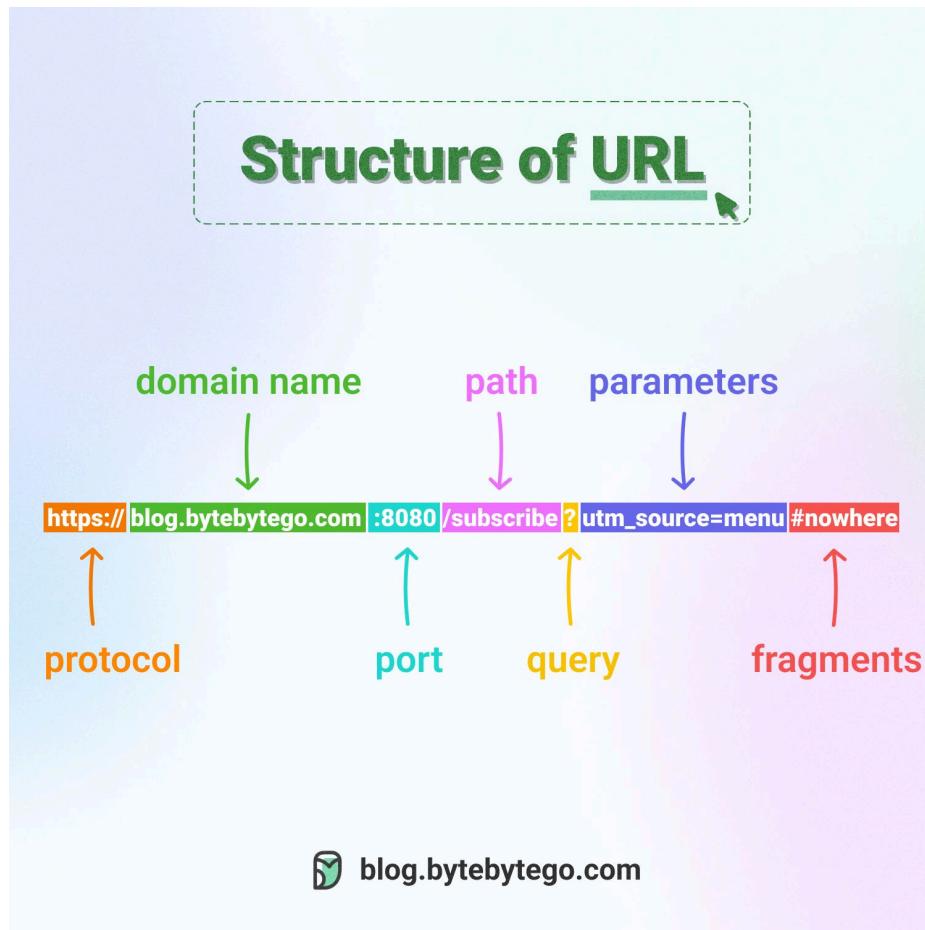
Startup
—Rules—

While academics are essential, it's crucial to acknowledge that many elements in this diagram would have been beneficial to learn earlier.

Over to you: What else should be on the list? What are the top 3 skills you wish schools would teach?

Do you know all the components of a URL?

Uniform Resource Locator (URL) is a term familiar to most people, as it is used to locate resources on the internet. When you type a URL into a web browser's address bar, you are accessing a "resource", not just a webpage.

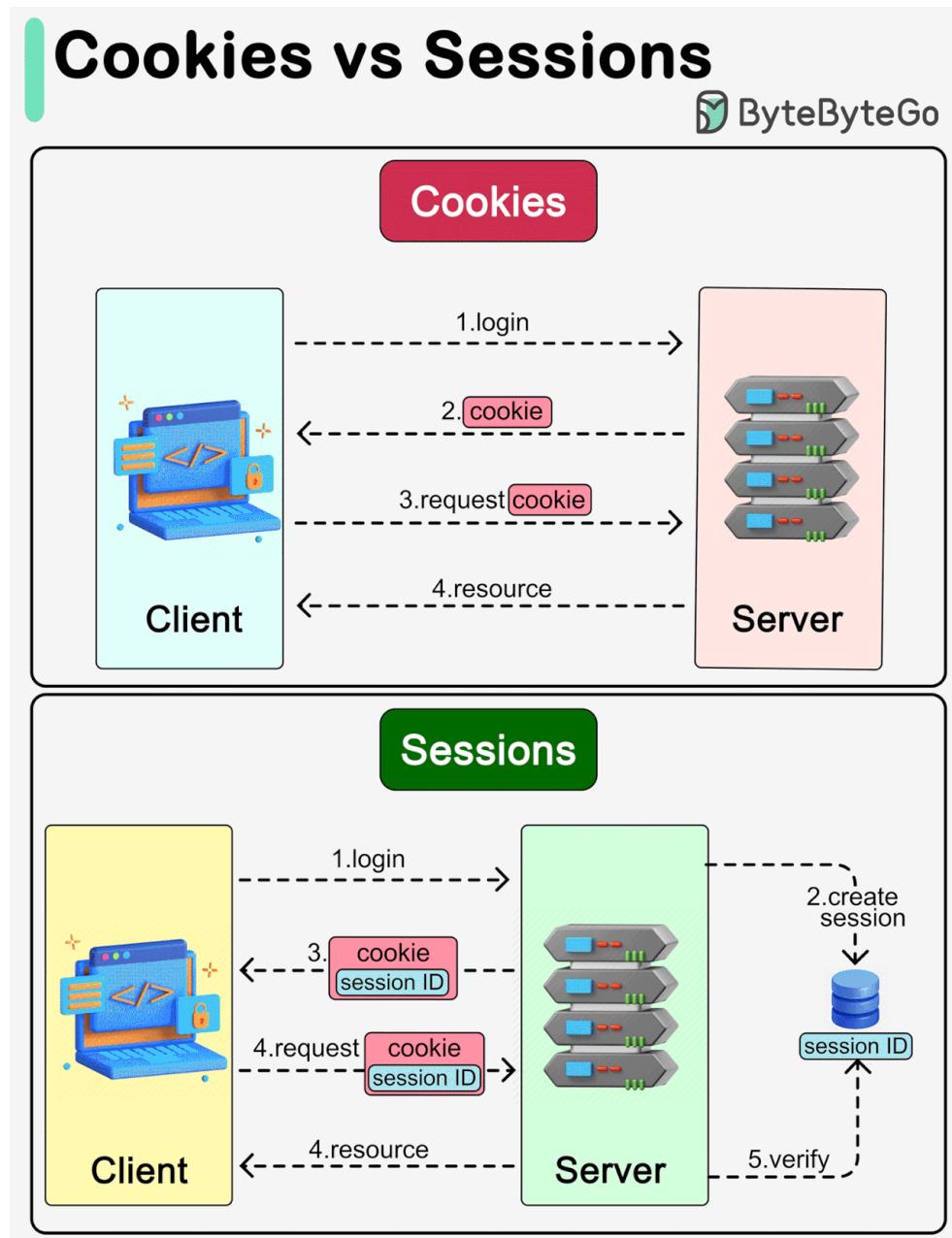


URLs comprise several components:

- The protocol or scheme, such as http, https, and ftp.
- The domain name and port, separated by a period (.)
- The path to the resource, separated by a slash (/)
- The parameters, which start with a question mark (?) and consist of key-value pairs, such as a=b&c=d.
- The fragment or anchor, indicated by a pound sign (#), which is used to bookmark a specific section of the resource.

What are the differences between cookies and sessions?

The diagram below shows how they work.



Cookies and sessions are both used to carry user information over HTTP requests, including user login status, user permissions, etc.

- Cookies

Cookies typically have size limits (4KB). They carry small pieces of information and are stored on the users' devices. Cookies are sent with each subsequent user request. Users

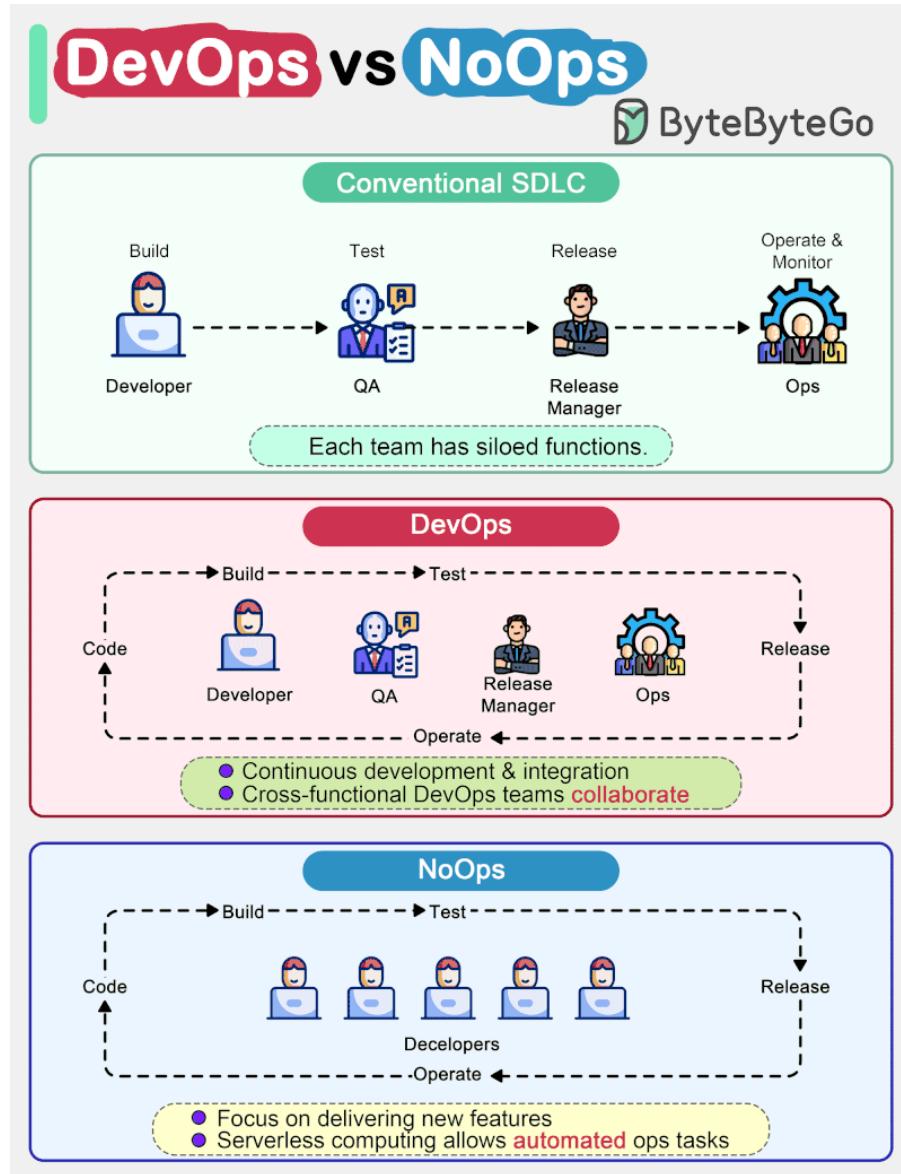
can choose to ban cookies in their browsers.

- Sessions

Unlike cookies, sessions are created and stored on the server side. There is usually a unique session ID generated on the server, which is attached to a specific user session. This session ID is returned to the client side in a cookie. Sessions can hold larger amounts of data. Since the session data is not directly accessed by the client, the session offers more security.

How do DevOps, NoOps change the software development lifecycle (SDLC)?

The diagram below compares traditional SDLC, DevOps and NoOps.



In a traditional software development, code, build, test, release and monitoring are siloed functions. Each stage works independently and hands over to the next stage.

DevOps, on the other hand, encourages continuous development and collaboration between developers and operations. This shortens the overall life cycle and provides continuous software delivery.

NoOps is a newer concept with the development of serverless computing. Since we can architect the system using FaaS (Function-as-a-Service) and BaaS (Backend-as-a-Service), the cloud service providers can take care of most operations tasks. The developers can focus on feature development and automate operations tasks.

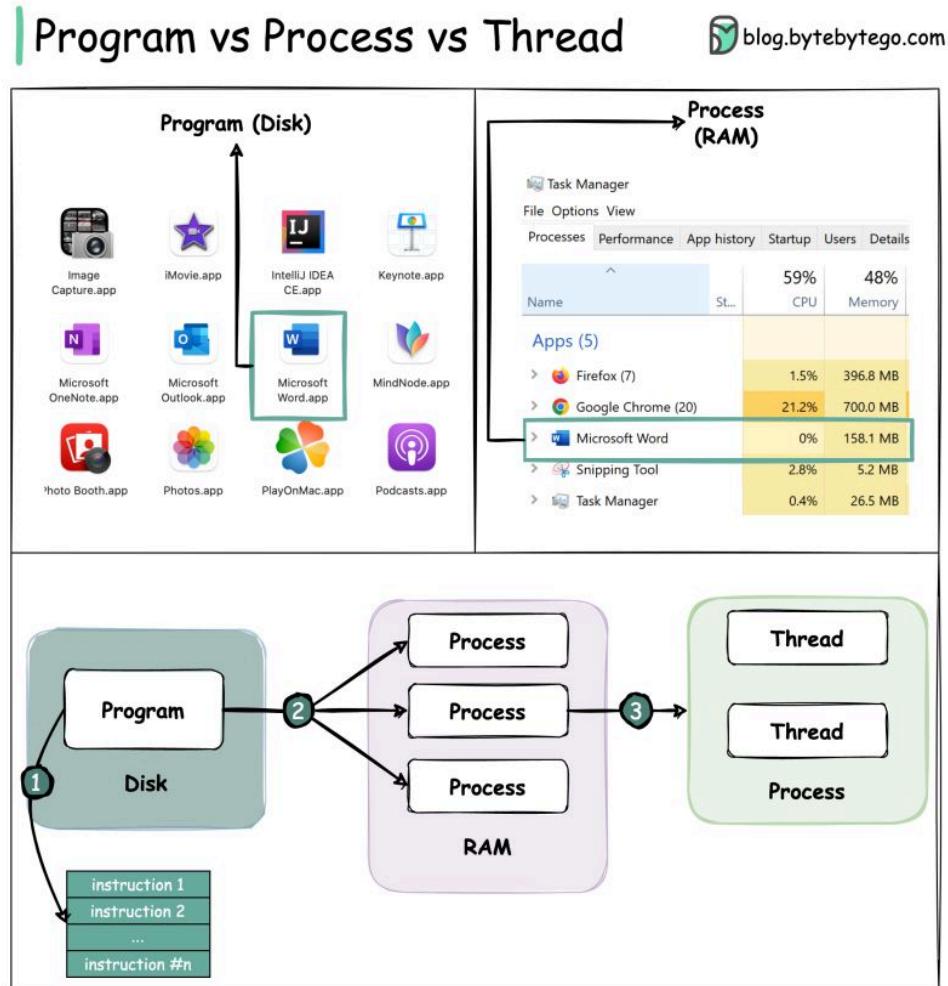
NoOps is a pragmatic and effective methodology for startups or smaller-scale applications, which moves shortens the SDLC even more than DevOps.

Popular interview question: What is the difference between Process and Thread?

To better understand this question, let's first take a look at what a Program is. A Program is an executable file containing a set of instructions and passively stored on disk. One program can have multiple processes. For example, the Chrome browser creates a different process for every single tab.

A Process means a program is in execution. When a program is loaded into the memory and becomes active, the program becomes a process. The process requires some essential resources such as registers, program counter, and stack.

A Thread is the smallest unit of execution within a process.



The following process explains the relationship between program, process, and thread.

1. The program contains a set of instructions.

2. The program is loaded into memory. It becomes one or more running processes.
3. When a process starts, it is assigned memory and resources. A process can have one or more threads. For example, in the Microsoft Word app, a thread might be responsible for spelling checking and the other thread for inserting text into the doc.

Main differences between process and thread:

- Processes are usually independent, while threads exist as subsets of a process.
- Each process has its own memory space. Threads that belong to the same process share the same memory.
- A process is a heavyweight operation. It takes more time to create and terminate.
- Context switching is more expensive between processes.
- Inter-thread communication is faster for threads.

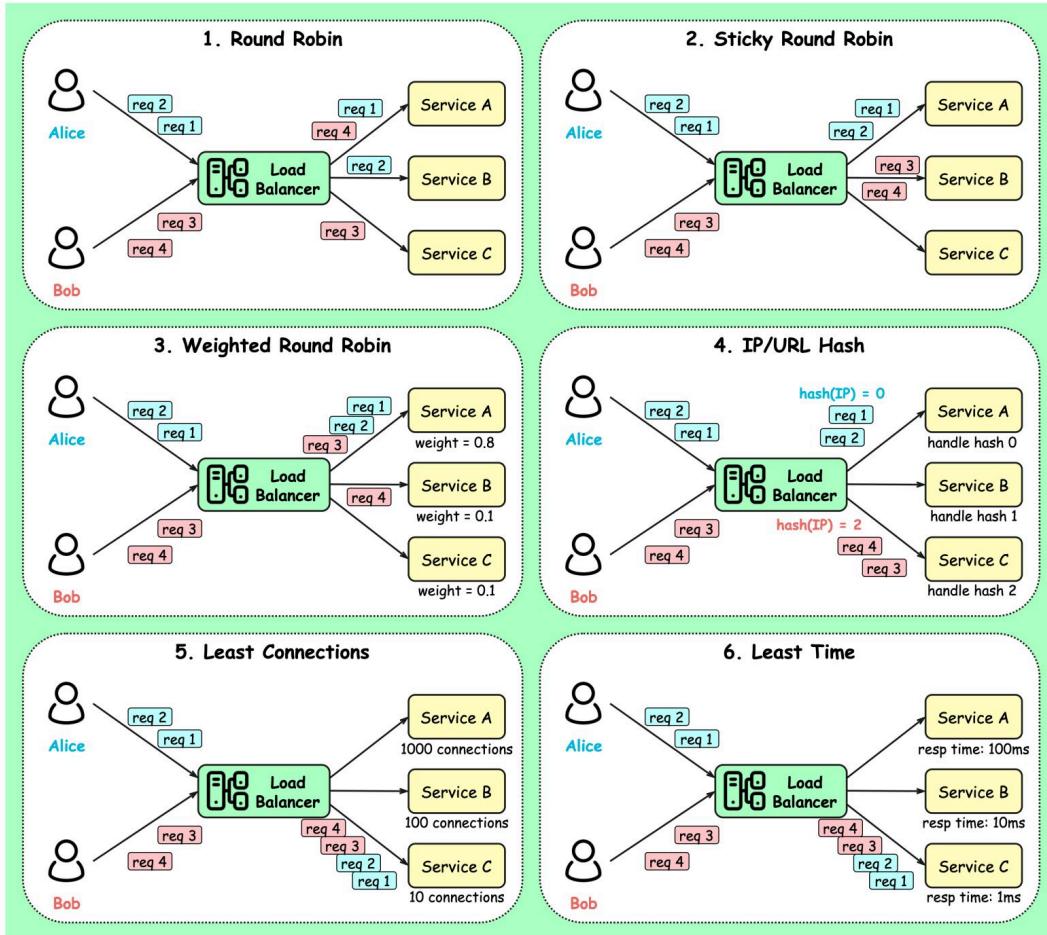
Over to you:

1. Some programming languages support coroutine. What is the difference between coroutine and thread?
2. How to list running processes in Linux?

Top 6 Load Balancing Algorithms

Load Balancing Algorithms

 blog.bytebytego.com



Static Algorithms

1. Round robin

The client requests are sent to different service instances in sequential order. The services are usually required to be stateless.

2. Sticky round-robin

This is an improvement of the round-robin algorithm. If Alice's first request goes to service A, the following requests go to service A as well.

3. Weighted round-robin

The admin can specify the weight for each service. The ones with a higher weight handle more requests than others.

4. Hash

This algorithm applies a hash function on the incoming requests' IP or URL. The requests are routed to relevant instances based on the hash function result.

Dynamic Algorithms

5. Least connections

A new request is sent to the service instance with the least concurrent connections.

6. Least response time

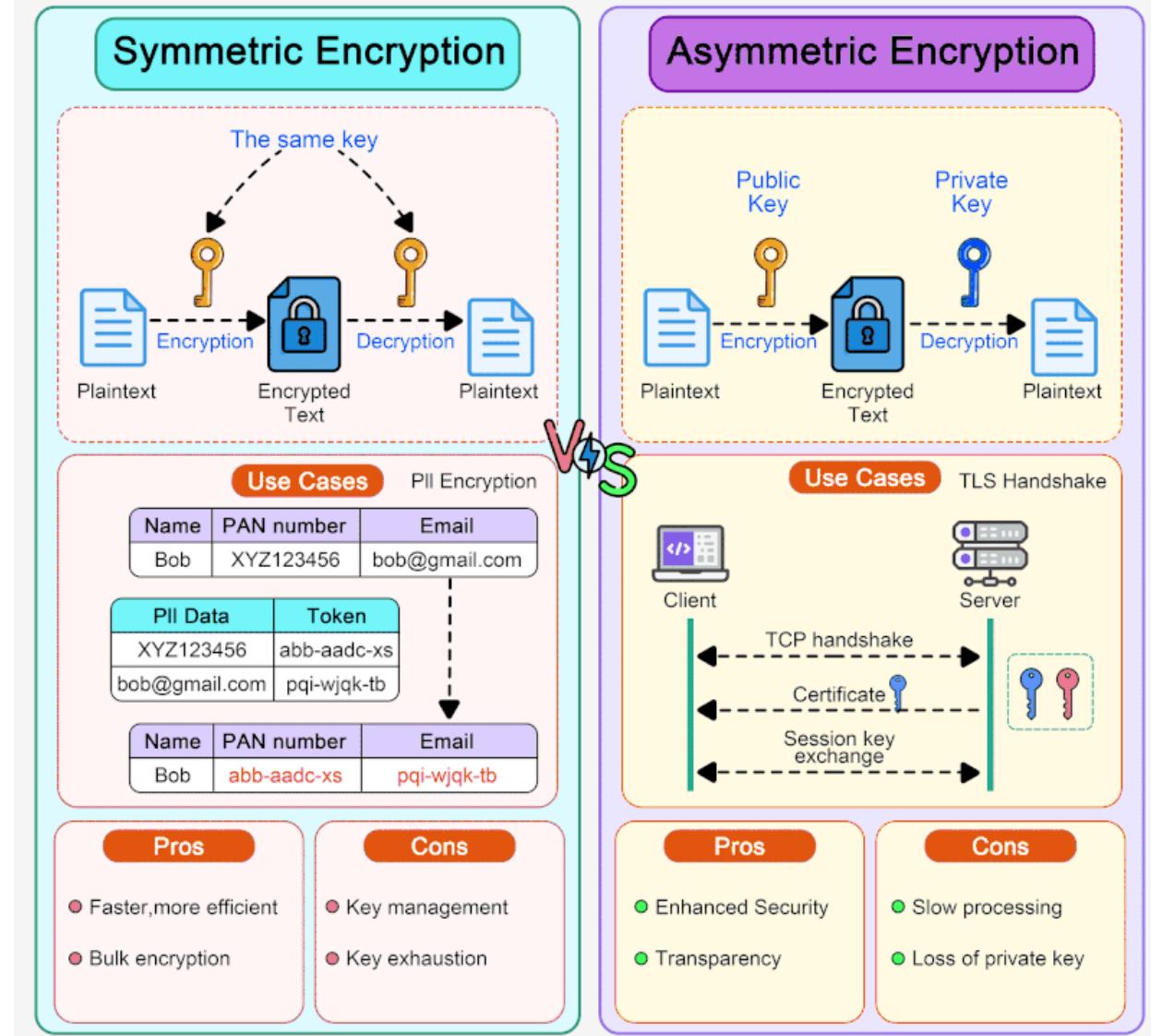
A new request is sent to the service instance with the fastest response time.

Symmetric encryption vs asymmetric encryption

Symmetric encryption and asymmetric encryption are two types of cryptographic techniques used to secure data and communications, but they differ in their methods of encryption and decryption.

Symmetric vs Asymmetric Encryption

ByteByteGo



- In symmetric encryption, a single key is used for both encryption and decryption of data. It is faster and can be applied to bulk data encryption/decryption. For example, we can use it to encrypt massive amounts of PII (Personally Identifiable Information) data. It poses challenges in key management because the sender and receiver share the same key.

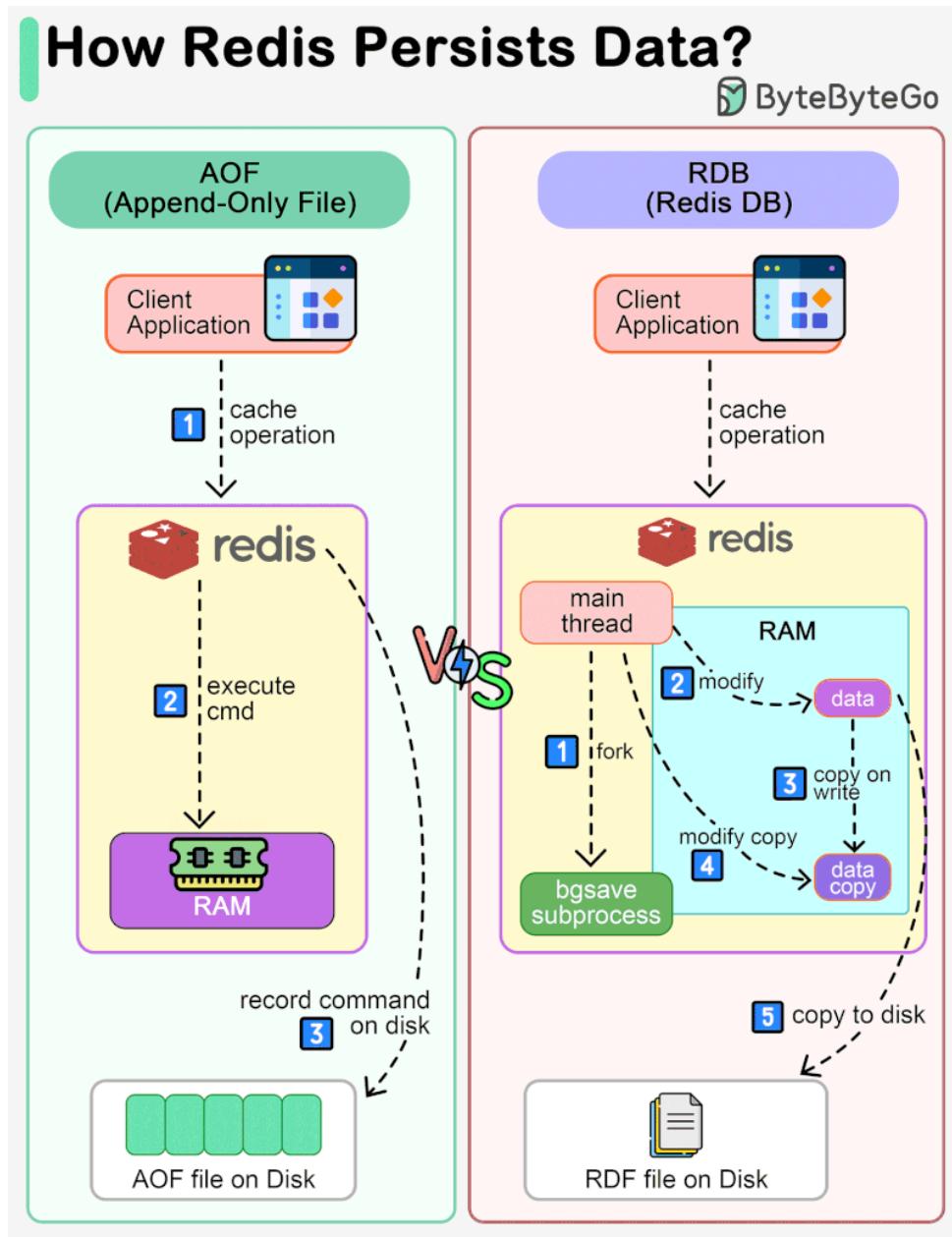
- Asymmetric encryption uses a pair of keys: a public key and a private key. The public key is freely distributed and used to encrypt data, while the private key is kept secret and used to decrypt the data. It is more secure than symmetric encryption because the private key is never shared. However, asymmetric encryption is slower because of the complexity of key generation and maths computations. For example, HTTPS uses asymmetric encryption to exchange session keys during TLS handshake, and after that, HTTPS uses symmetric encryption for subsequent communications.

How does Redis persist data?

Redis is an in-memory database. If the server goes down, the data will be lost.

The diagram below shows two ways to persist Redis data on disk:

1. AOF (Append-Only File)
2. RDB (Redis Database)



Note that data persistence is not performed on the critical path and doesn't block the write process in Redis.

- AOF

Unlike a write-ahead log, the Redis AOF log is a write-after log. Redis executes commands to modify the data in memory first and then writes it to the log file. AOF log records the commands instead of the data. The event-based design simplifies data recovery. Additionally, AOF records commands after the command has been executed in memory, so it does not block the current write operation.

- RDB

The restriction of AOF is that it persists commands instead of data. When we use the AOF log for recovery, the whole log must be scanned. When the size of the log is large, Redis takes a long time to recover. So Redis provides another way to persist data - RDB.

RDB records snapshots of data at specific points in time. When the server needs to be recovered, the data snapshot can be directly loaded into memory for fast recovery.

Step 1: The main thread forks the 'bgsave' sub-process, which shares all the in-memory data of the main thread. 'bgsave' reads the data from the main thread and writes it to the RDB file.

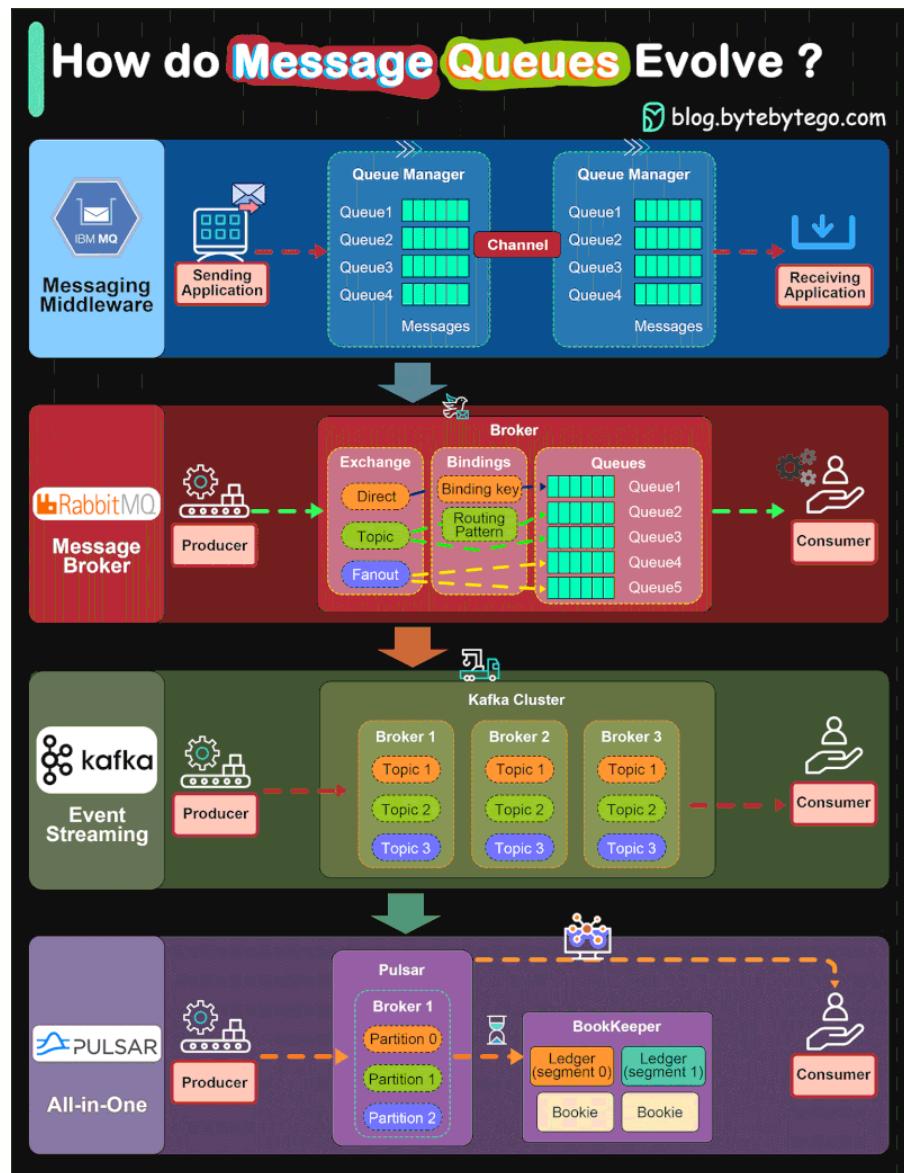
Steps 2 and 3: If the main thread modifies data, a copy of the data is created.

Steps 4 and 5: The main thread then operates on the data copy. Meanwhile 'bgsave' sub-process continues to write data to the RDB file.

- Mixed

Usually in production systems, we can choose a mixed approach, where we use RDB to record data snapshots from time to time and use AOF to record the commands since the last snapshot.

IBM MQ -> RabbitMQ -> Kafka ->Pulsar, How do message queue architectures evolve?



- **IBM MQ**
IBM MQ was launched in 1993. It was originally called MQSeries and was renamed WebSphere MQ in 2002. It was renamed to IBM MQ in 2014. IBM MQ is a very successful product widely used in the financial sector. Its revenue still reached 1 billion dollars in 2020.
- **RabbitMQ**
RabbitMQ architecture differs from IBM MQ and is more similar to Kafka concepts. The producer publishes a message to an exchange with a specified exchange type. It can be direct, topic, or fanout. The exchange then routes the message into the queues based on

different message attributes and the exchange type. The consumers pick up the message accordingly.

- **Kafka**

In early 2011, LinkedIn open sourced Kafka, which is a distributed event streaming platform. It was named after Franz Kafka. As the name suggested, Kafka is optimized for writing. It offers a high-throughput, low-latency platform for handling real-time data feeds. It provides a unified event log to enable event streaming and is widely used in internet companies.

Kafka defines producer, broker, topic, partition, and consumer. Its simplicity and fault tolerance allow it to replace previous products like AMQP-based message queues.

- **Pulsar**

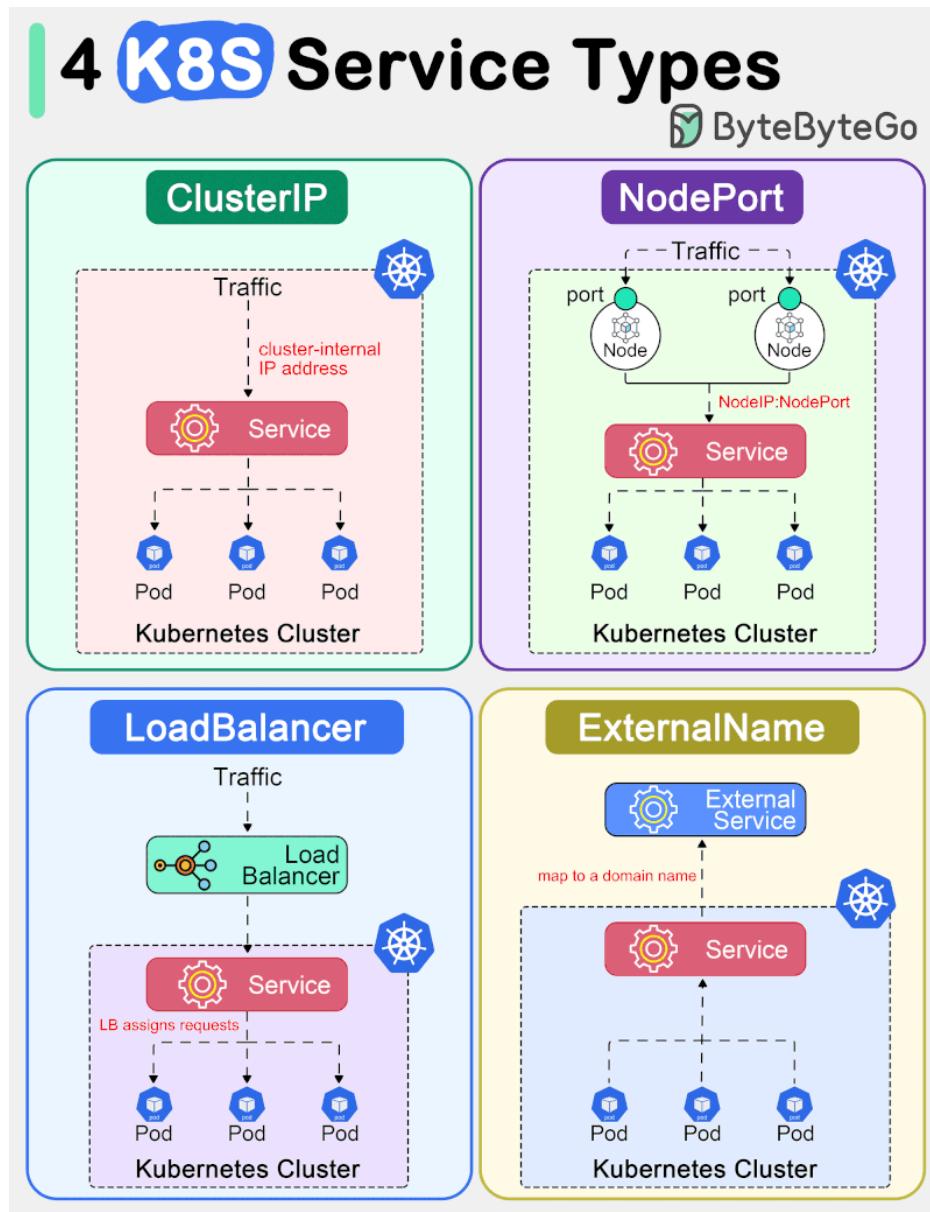
Pulsar, developed originally by Yahoo, is an all-in-one messaging and streaming platform. Compared with Kafka, Pulsar incorporates many useful features from other products and supports a wide range of capabilities. Also, Pulsar architecture is more cloud-native, providing better support for cluster scaling and partition migration, etc.

There are two layers in Pulsar architecture: the serving layer and the persistent layer. Pulsar natively supports tiered storage, where we can leverage cheaper object storage like AWS S3 to persist messages for a longer term.

Over to you: which message queues have you used?

Top 4 Kubernetes Service Types in one diagram

The diagram below shows 4 ways to expose a Service.



In Kubernetes, a Service is a method for exposing a network application in the cluster. We use a Service to make that set of Pods available on the network so that users can interact with it.

There are 4 types of Kubernetes services: ClusterIP, NodePort, LoadBalancer and ExternalName. The “type” property in the Service's specification determines how the service is exposed to the network.

- ClusterIP

ClusterIP is the default and most common service type. Kubernetes will assign a

cluster-internal IP address to ClusterIP service. This makes the service only reachable within the cluster.

- **NodePort**

This exposes the service outside of the cluster by adding a cluster-wide port on top of ClusterIP. We can request the service by NodeIP:NodePort.

- **LoadBalancer**

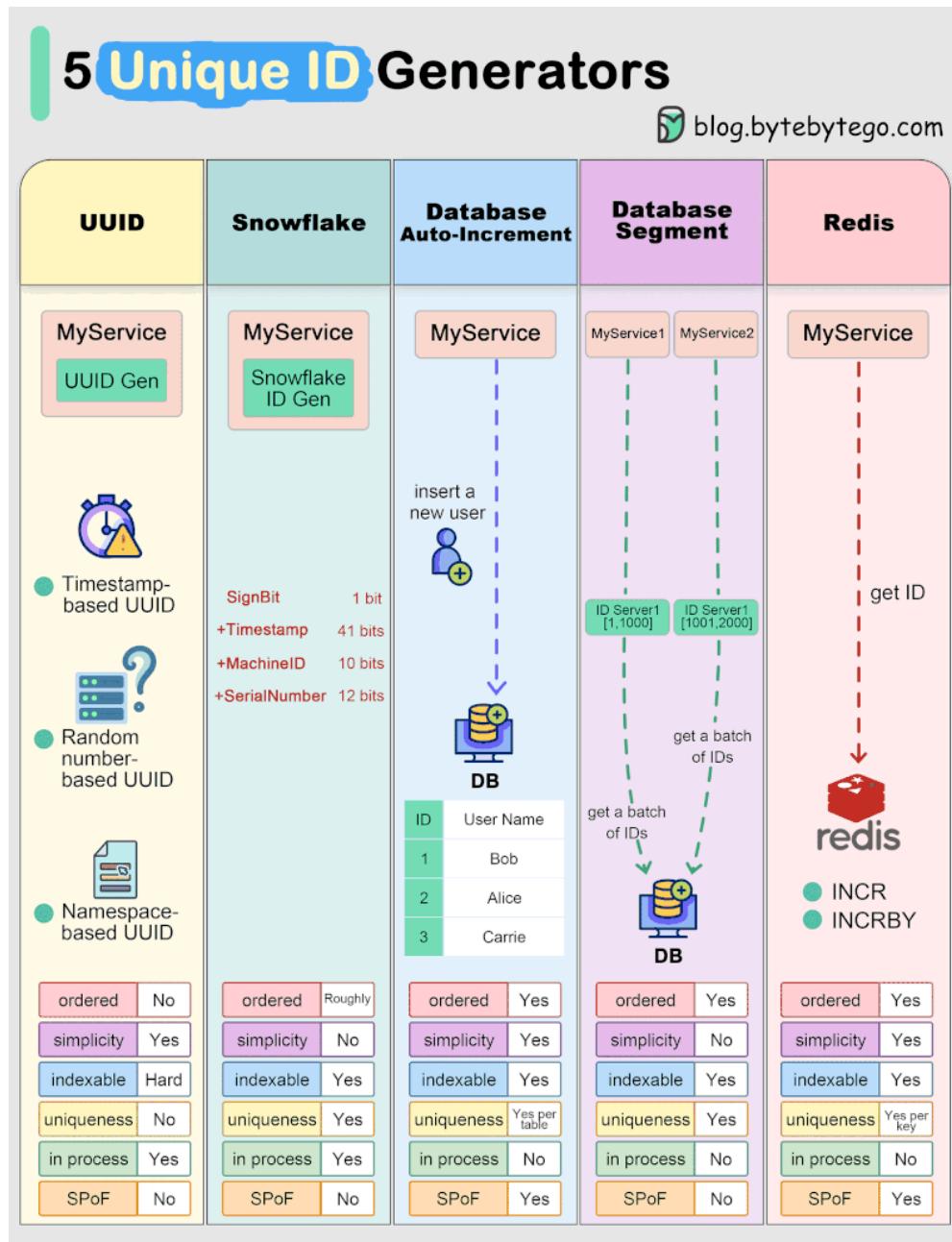
This exposes the Service externally using a cloud provider's load balancer.

- **ExternalName**

This maps a Service to a domain name. This is commonly used to create a service within Kubernetes to represent an external database.

Explaining 5 unique ID generators in distributed systems

The diagram below shows how they work. Each generator has its pros and cons.



1. UUID

A UUID has 128 bits. It is simple to generate and no need to call another service. However, it is not sequential and inefficient for database indexing. Additionally, UUID doesn't guarantee global uniqueness. We need to be careful with ID conflicts (although the chances are slim.)

2. Snowflake

Snowflake's ID generation process has multiple components: timestamp, machine ID, and serial number. The first bit is unused to ensure positive IDs. This generator doesn't need to talk to an ID generator via the network, so is fast and scalable.

Snowflake implementations vary. For example, data center ID can be added to the "MachineID" component to guarantee global uniqueness.

3. DB auto-increment

Most database products offer auto-increment identity columns. Since this is supported in the database, we can leverage its transaction management to handle concurrent visits to the ID generator. This guarantees uniqueness in one table. However, this involves network communications and may expose sensitive business data to the outside. For example, if we use this as a user ID, our business competitors will have a rough idea of the total number of users registered on our website.

4. DB segment

An alternative approach is to retrieve IDs from the database in batches and cache them in the ID servers, each ID server handling a segment of IDs. This greatly saves the I/O pressure on the database.

5. Redis

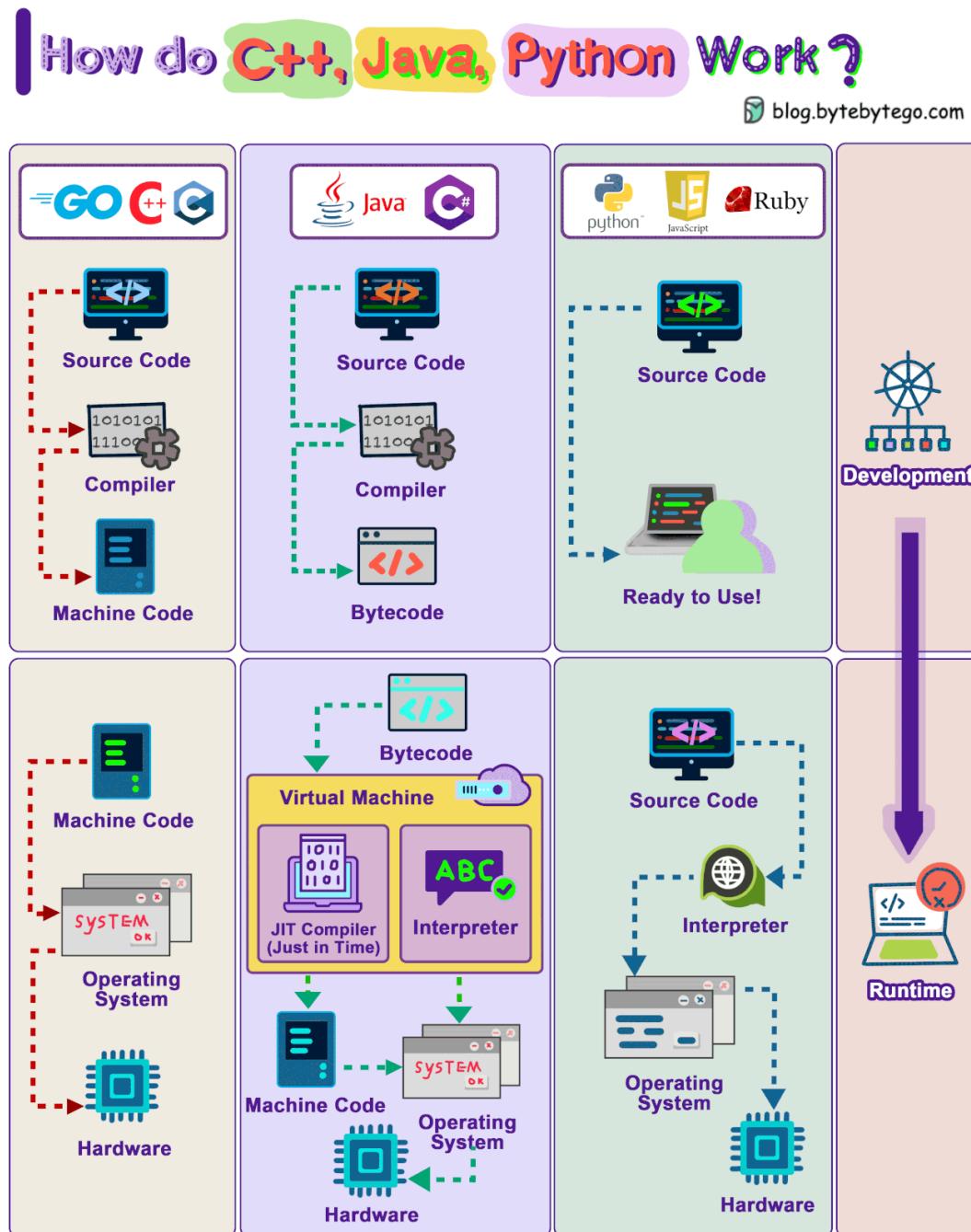
We can also use Redis key-value pair to generate unique IDs. Redis stores data in memory, so this approach offers better performance than the database.

- Over to you - What ID generator have you used?

How Do C++, Java, and Python Function?

We just made a video on this topic.

The illustration details the processes of compilation and execution.



Languages that compile transform source code into machine code using a compiler. This machine code can subsequently be run directly by the CPU. For instance: C, C++, Go.

In contrast, languages like Java first convert the source code into bytecode. The Java Virtual Machine (JVM) then runs the program. Occasionally, a Just-In-Time (JIT) compiler translates the source code into machine code to enhance execution speed. Some examples are Java and C#.

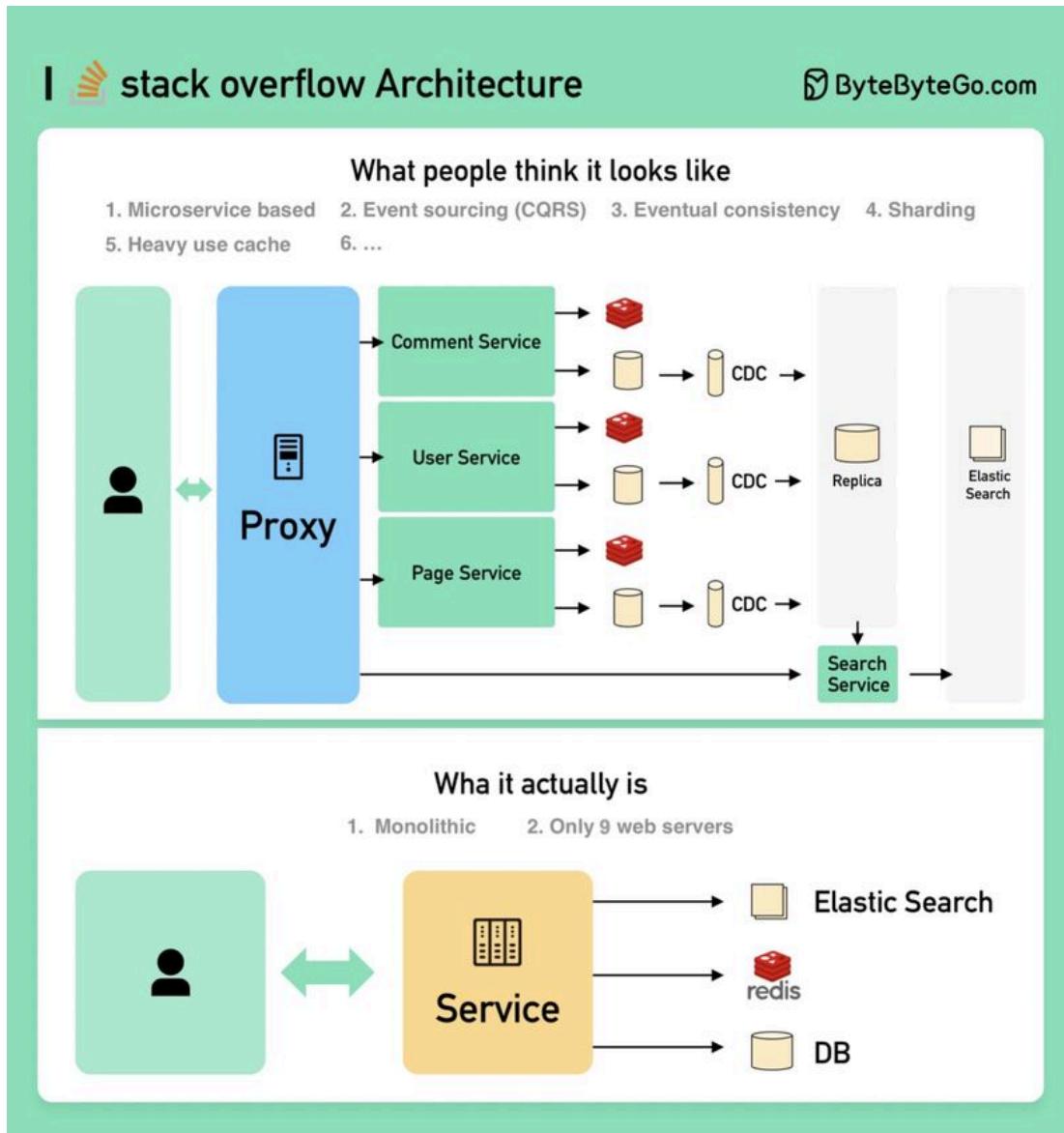
Languages that are interpreted don't undergo compilation. Instead, their code is processed by an interpreter during execution. Python, Javascript, and Ruby are some examples.

Generally, compiled languages have a speed advantage over interpreted ones.

Watch the whole video here: <https://lnkd.in/ezpN2jH5>

How will you design the Stack Overflow website?

If your answer is on-premise servers and monolith (on the right), you would likely fail the interview, but that's how it is built in reality!



What people think it should look like

The interviewer is probably expecting something on the left side.

1. Microservice is used to decompose the system into small components.
2. Each service has its own database. Use cache heavily.
3. The service is sharded.
4. The services talk to each other asynchronously through message queues.
5. The service is implemented using Event Sourcing with CQRS.

6. Showing off knowledge in distributed systems such as eventual consistency, CAP theorem, etc.

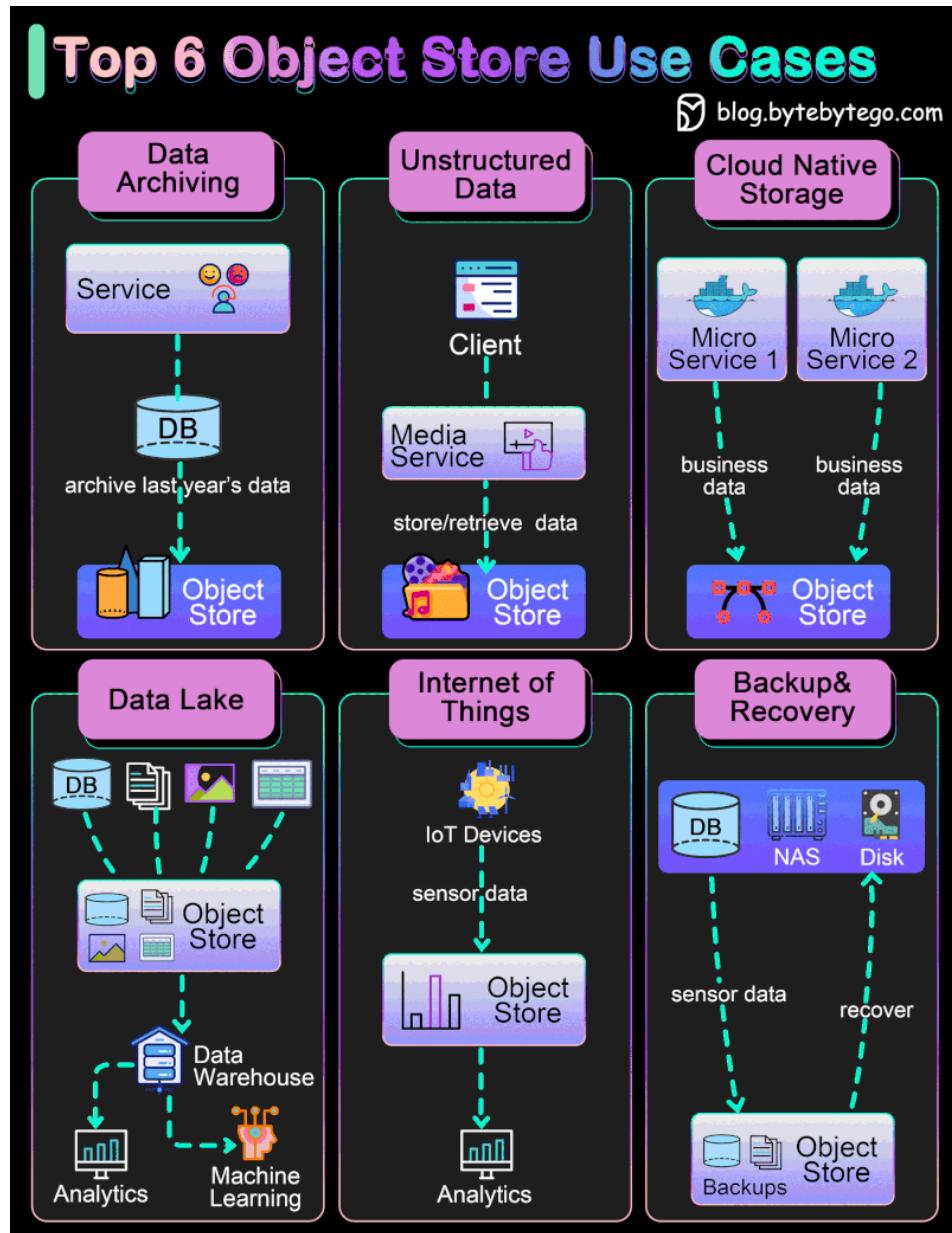
What it actually is

Stack Overflow serves all the traffic with only 9 on-premise web servers, and it's a monolith! It has its own servers and does not run on the cloud.

This is contrary to all our popular beliefs these days.

Over to you: what is good architecture, the one that looks fancy during the interview or the one that works in reality?

Explain the Top 6 Use Cases of Object Stores



- What is an object store?

Object store uses objects to store data. Compared with file storage which uses a hierarchical structure to store files, or block storage which divides files into equal block sizes, object storage stores metadata together with the objects. Typical products include AWS S3, Google Cloud Storage, and Azure Blob Storage.

An object store provides flexibility in formats and scales easily.

- Case 1: Data Archiving

With the ever-growing amounts of business data, we cannot store all the data in core storage systems. We need to have layers of storage plan. An object store can be used to archive old data that exists for auditing or client statements. This is a cost-effective approach.

- Case 2: Unstructured Data Storage

We often need to deal with unstructured data or semi-structured data. In the past, they were usually stored as blobs in the relational database, which was quite inefficient. An object store is a good match for music, video files, and text documents. Companies like Spotify or Netflix uses object store to persist their media files.

- Case 3: Cloud Native Storage

For cloud-native applications, we need the data storage system to be flexible and scalable. Major public cloud providers have easy API access to their object store products and can be used for economical storage choices.

- Case 4: Data Lake

There are many types of data in a distributed system. An object store-backed data lake provides a good place for different business lines to dump their data for later analytics or machine learning. The efficient reads and writes of the object store facilitate more steps down the data processing pipeline, including ETL(Extract-Transform-Load) or constructing a data warehouse.

- Case 5: Internet of Things (IoT)

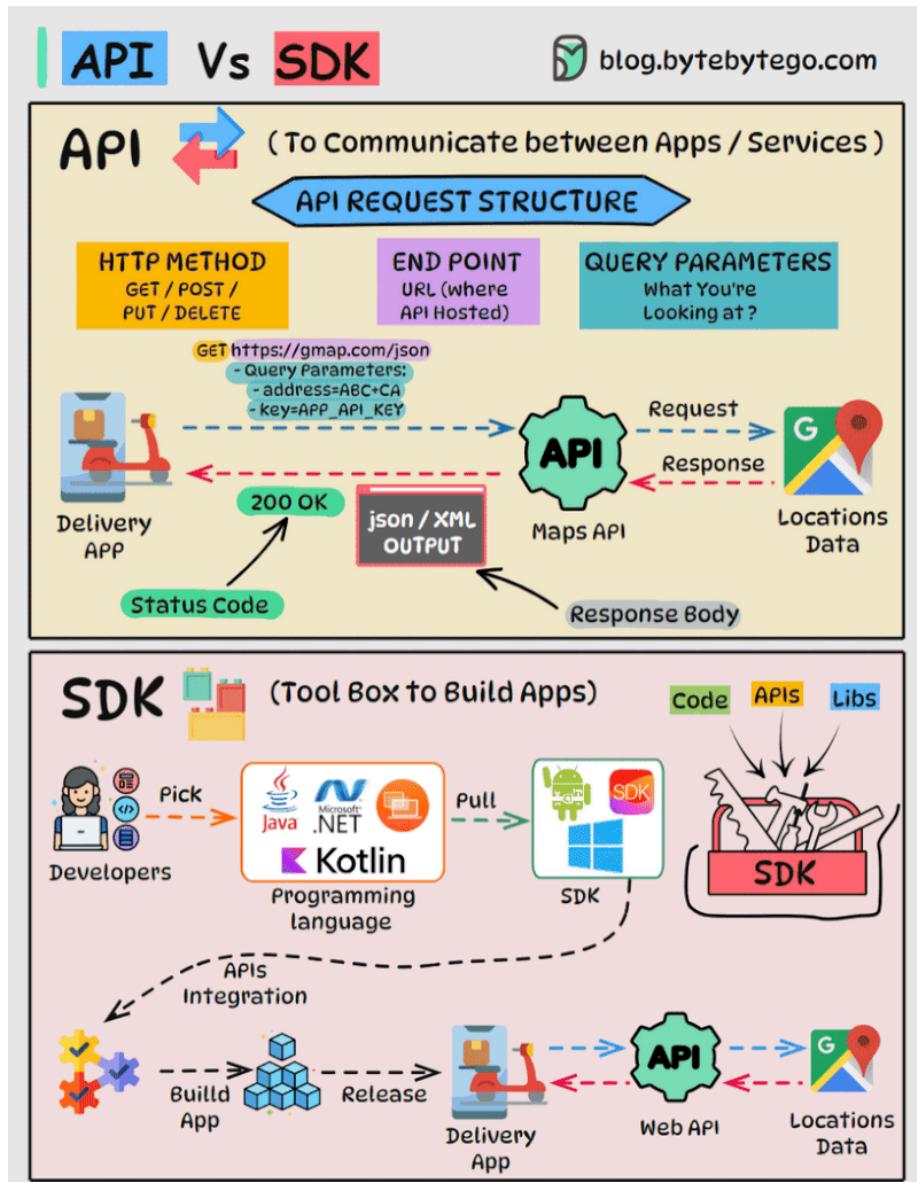
IoT sensors produce all kinds of data. An object store can store this type of time series and later run analytics or AI algorithms on them. Major public cloud providers provide pipelines to ingest raw IoT data into the object store.

- Case 6: Backup and Recovery

An object store can be used to store database or file system backups. Later, the backups can be loaded for fast recovery. This improves the system's availability.

Over to you: What did you use object store for?

API Vs SDK!



API (Application Programming Interface) and SDK (Software Development Kit) are essential tools in the software development world, but they serve distinct purposes:

API:

An API is a set of rules and protocols that allows different software applications and services to communicate with each other.

1. It defines how software components should interact.
2. Facilitates data exchange and functionality access between software components.
3. Typically consists of endpoints, requests, and responses.

SDK:

An SDK is a comprehensive package of tools, libraries, sample code, and documentation that assists developers in building applications for a particular platform, framework, or hardware.

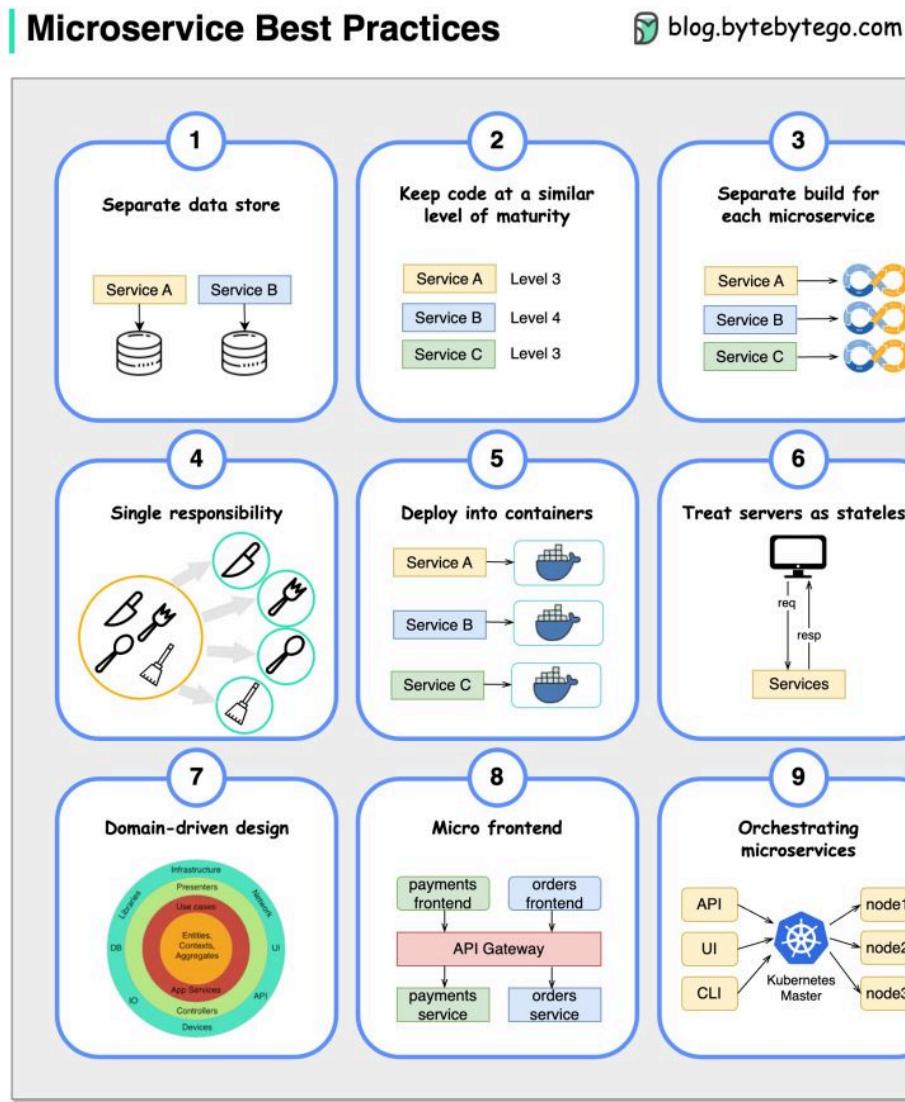
1. Offers higher-level abstractions, simplifying development for a specific platform.
2. Tailored to specific platforms or frameworks, ensuring compatibility and optimal performance on that platform.
3. Offer access to advanced features and capabilities specific to the platform, which might be otherwise challenging to implement from scratch.

The choice between APIs and SDKs depends on the development goals and requirements of the project.

Over to you:

Which do you find yourself gravitating towards – APIs or SDKs – Every implementation has a unique story to tell. What's yours?

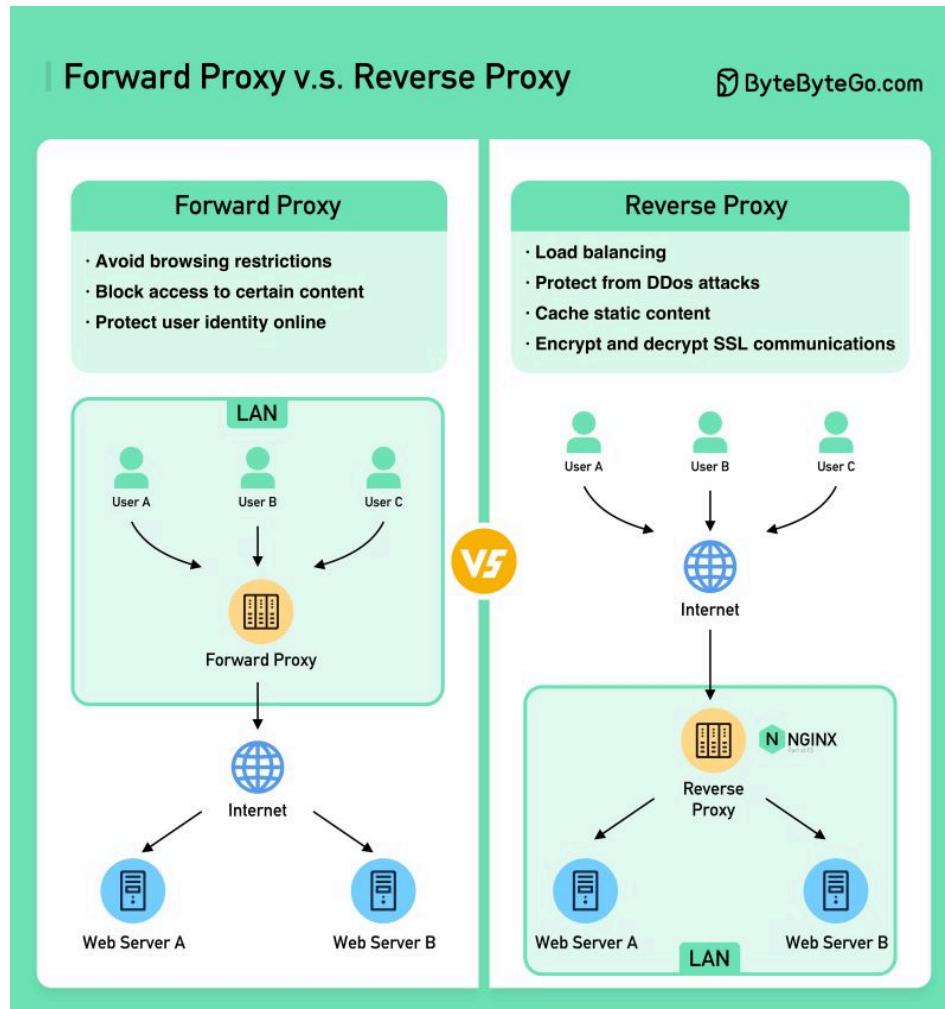
A picture is worth a thousand words: 9 best practices for developing microservices



When we develop microservices, we need to follow the following best practices:

1. Use separate data storage for each microservice
2. Keep code at a similar level of maturity
3. Separate build for each microservice
4. Assign each microservice with a single responsibility
5. Deploy into containers
6. Design stateless services
7. Adopt domain-driven design
8. Design micro frontend
9. Orchestrating microservices

Proxy Vs reverse proxy



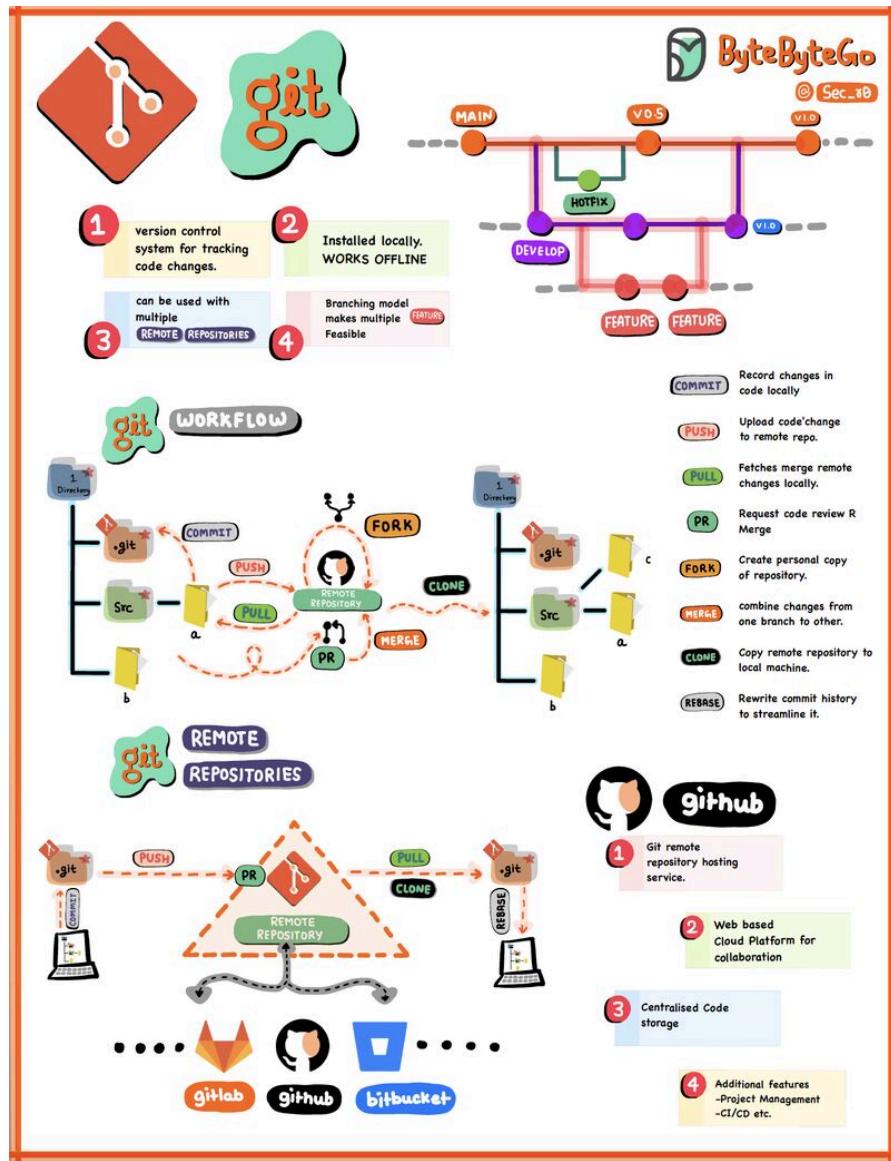
A forward proxy is a server that sits between user devices and the internet. A forward proxy is commonly used for:

- Protect clients
- Avoid browsing restrictions
- Block access to certain content

A reverse proxy is a server that accepts a request from the client, forwards the request to web servers, and returns the results to the client as if the proxy server had processed the request. A reverse proxy is good for:

- Protect servers
- Load balancing
- Cache static contents
- Encrypt and decrypt SSL communications

Git Vs Github



Dive into the fascinating world of version control.

First, meet Git, a fundamental tool for developers. It operates locally, allowing you to track changes in your code, much like taking snapshots of your project's progress. This makes collaboration with your team a breeze, even when you're working on the same project.

Now, let's talk about GitHub. It's more than just a platform; it's a powerhouse for hosting Git repositories online. With GitHub, you can streamline team collaboration and code sharing.

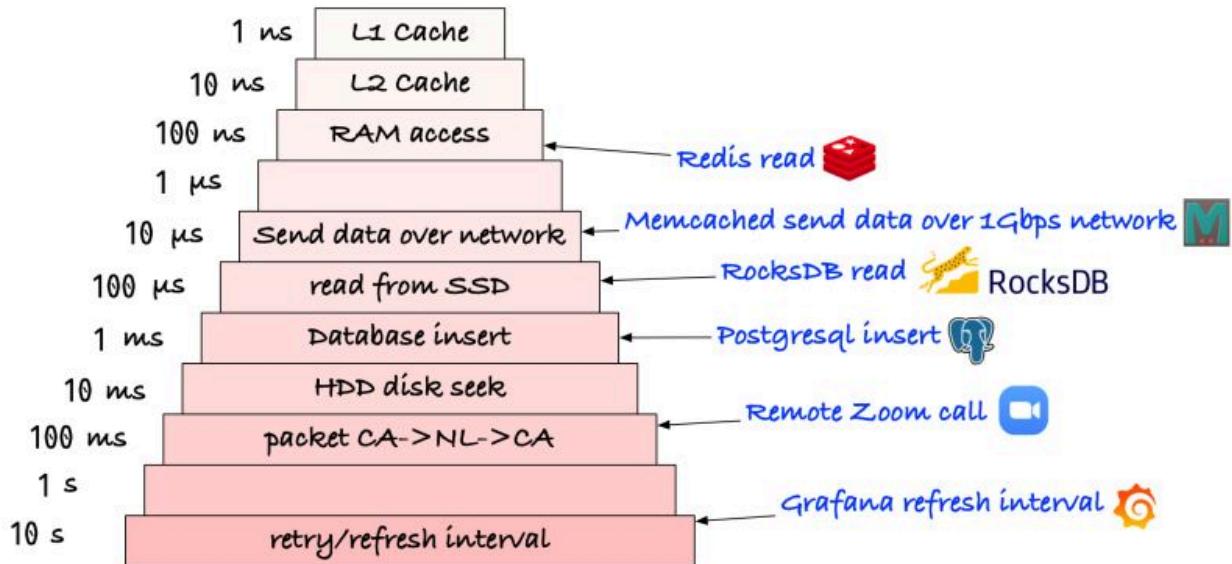
Learning Git and GitHub is a fundamental part of software engineering, so definitely try your best to master them

Which latency numbers should you know

Please note those are not precise numbers. They are based on some online benchmarks (Jeff Dean's latency numbers + some other sources).

Latency Numbers You Should Know

 ByteByteGo.com



- L1 and L2 caches: 1 ns, 10 ns

E.g.: They are usually built onto the microprocessor chip. Unless you work with hardware directly, you probably don't need to worry about them.

- RAM access: 100 ns

E.g.: It takes around 100 ns to read data from memory. Redis is an in-memory data store, so it takes about 100 ns to read data from Redis.

- Send 1K bytes over 1 Gbps network: 10 us

E.g.: It takes around 10 us to send 1KB of data from Memcached through the network.

- Read from SSD: 100 us

E.g.: RocksDB is a disk-based K/V store, so the read latency is around 100 us on SSD.

- Database insert operation: 1 ms.

E.g.: Postgresql commit might take 1ms. The database needs to store the data, create the index, and flush logs. All these actions take time.

- Send packet CA->Netherlands->CA: 100 ms
 - E.g.: If we have a long-distance Zoom call, the latency might be around 100 ms.
- Retry/refresh internal: 1-10s
 - E.g: In a monitoring system, the refresh interval is usually set to 5~10 seconds (default value on Grafana).

Notes:

1 ns = 10^{-9} seconds

1 us = 10^{-6} seconds = 1,000 ns

1 ms = 10^{-3} seconds = 1,000 us = 1,000,000 ns

Eight Data Structures That Power Your Databases. Which one should we pick?

The answer will vary depending on your use case. Data can be indexed in memory or on disk. Similarly, data formats vary, such as numbers, strings, geographic coordinates, etc. The system might be write-heavy or read-heavy. All of these factors affect your choice of database index format.

8 Data Structures That Power Your Databases



Types	Illustration	Use Case	Note
Skiplist		In-memory	used in Redis
Hash index		In-memory	Most common in-memory index solution
SSTable		Disk-based	Immutable data structure. Seldom used alone
LSM tree		Memory + Disk	High write throughput. Disk compaction may impact performance
B-tree		Disk-based	Most popular database index implementation
Inverted index		Search document	Used in document search engine such as Lucene
Suffix tree		Search string	Used in string search, such as string suffix match
R-tree		Search multi-dimension shape	Such as the nearest neighbor

The following are some of the most popular data structures used for indexing data:

- Skiplist: a common in-memory index type. Used in Redis
- Hash index: a very common implementation of the “Map” data structure (or “Collection”)

- SSTable: immutable on-disk “Map” implementation
- LSM tree: SkipList + SSTable. High write throughput
- B-tree: disk-based solution. Consistent read/write performance
- Inverted index: used for document indexing. Used in Lucene
- Suffix tree: for string pattern search
- R-tree: multi-dimension search, such as finding the nearest neighbor

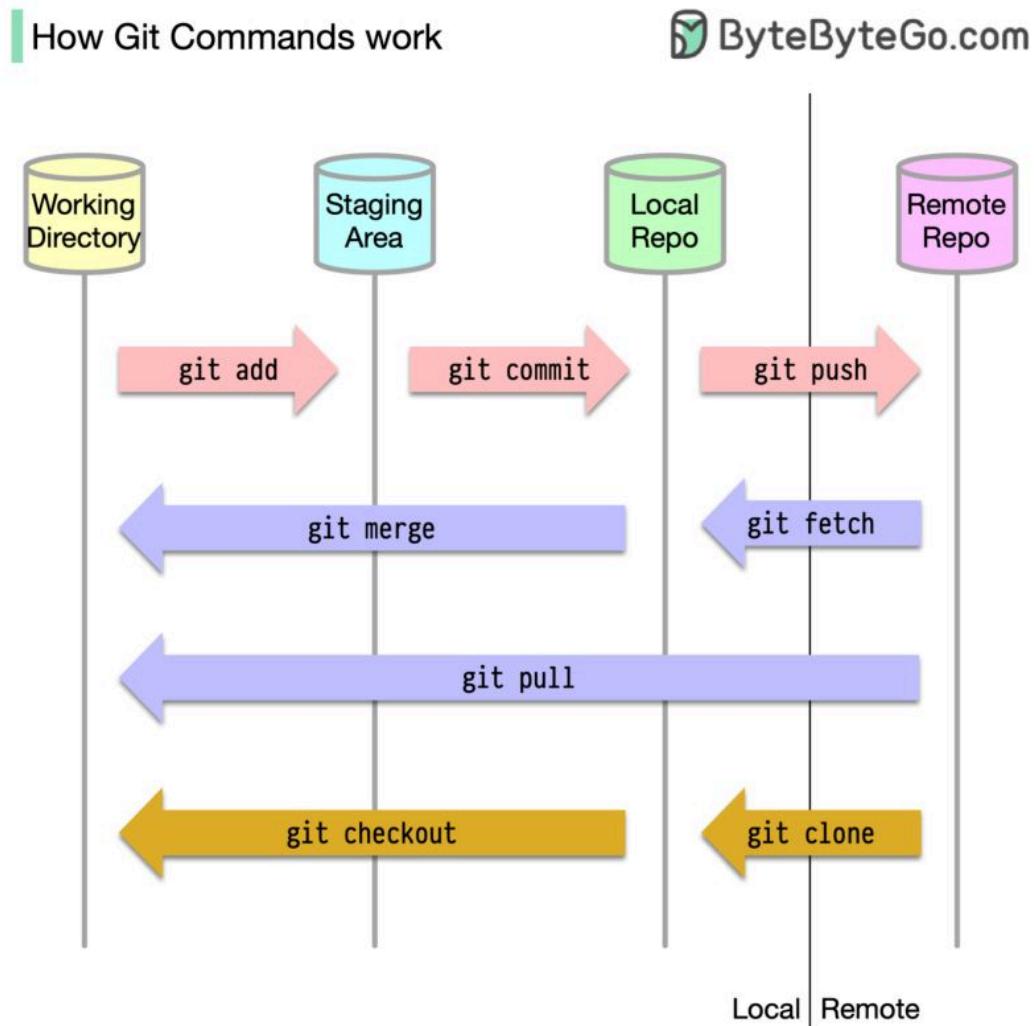
This is not an exhaustive list of all database index types.

Over to you:

1. Which one have you used and for what purpose?
2. There is another one called “reverse index”. Do you know the difference between “reverse index” and “inverted index”?

How Git Commands Work

Almost every software engineer has used Git before, but only a handful know how it works.



To begin with, it's essential to identify where our code is stored. The common assumption is that there are only two locations - one on a remote server like Github and the other on our local machine. However, this isn't entirely accurate. Git maintains three local storages on our machine, which means that our code can be found in four places:

- Working directory: where we edit files
- Staging area: a temporary location where files are kept for the next commit
- Local repository: contains the code that has been committed
- Remote repository: the remote server that stores the code

Most Git commands primarily move files between these four locations.

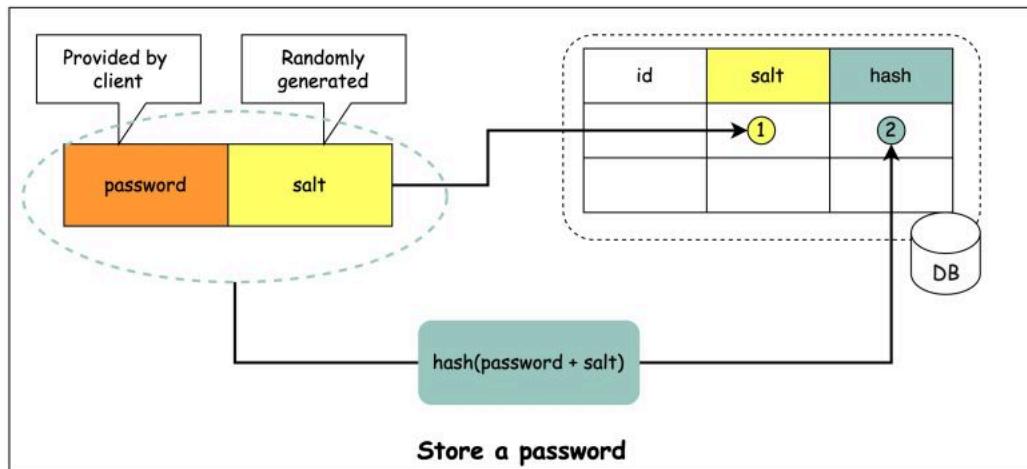
Over to you: Do you know which storage location the "git tag" command operates on? This command can add annotations to a commit.

How to store passwords safely in the database and how to validate a password?

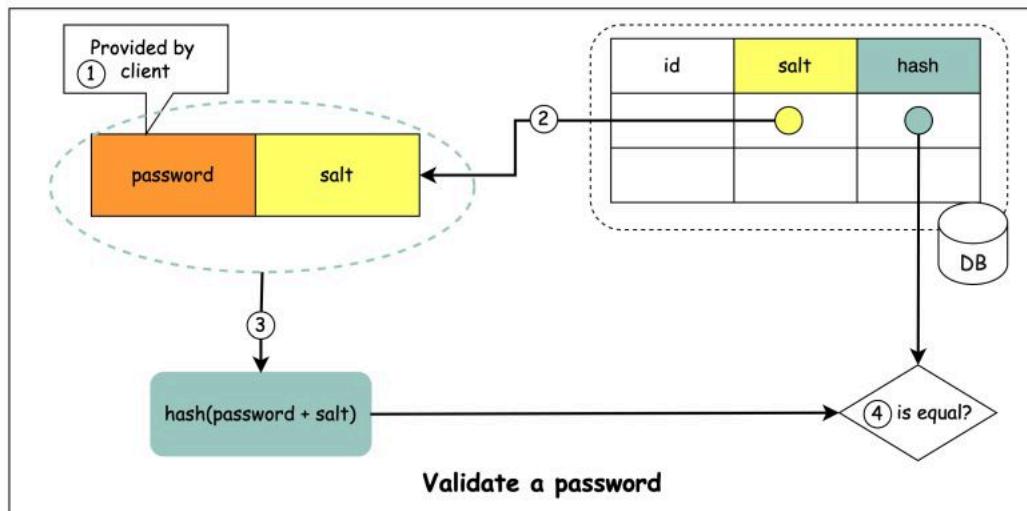
Let's take a look.

How to store passwords in DB?

 blog.bytebytego.com



Store a password



Validate a password

Things NOT to do

- Storing passwords in plain text is not a good idea because anyone with internal access can see them.
- Storing password hashes directly is not sufficient because it is prone to precomputation attacks, such as rainbow tables.
- To mitigate precomputation attacks, we salt the passwords.

What is salt?

According to OWASP guidelines, “a salt is a unique, randomly generated string that is added to each password as part of the hashing process”.

How to store a password and salt?

1. A salt is not meant to be secret and it can be stored in plain text in the database. It is used to ensure the hash result is unique to each password.
2. The password can be stored in the database using the following format: *hash(password + salt)*.

How to validate a password?

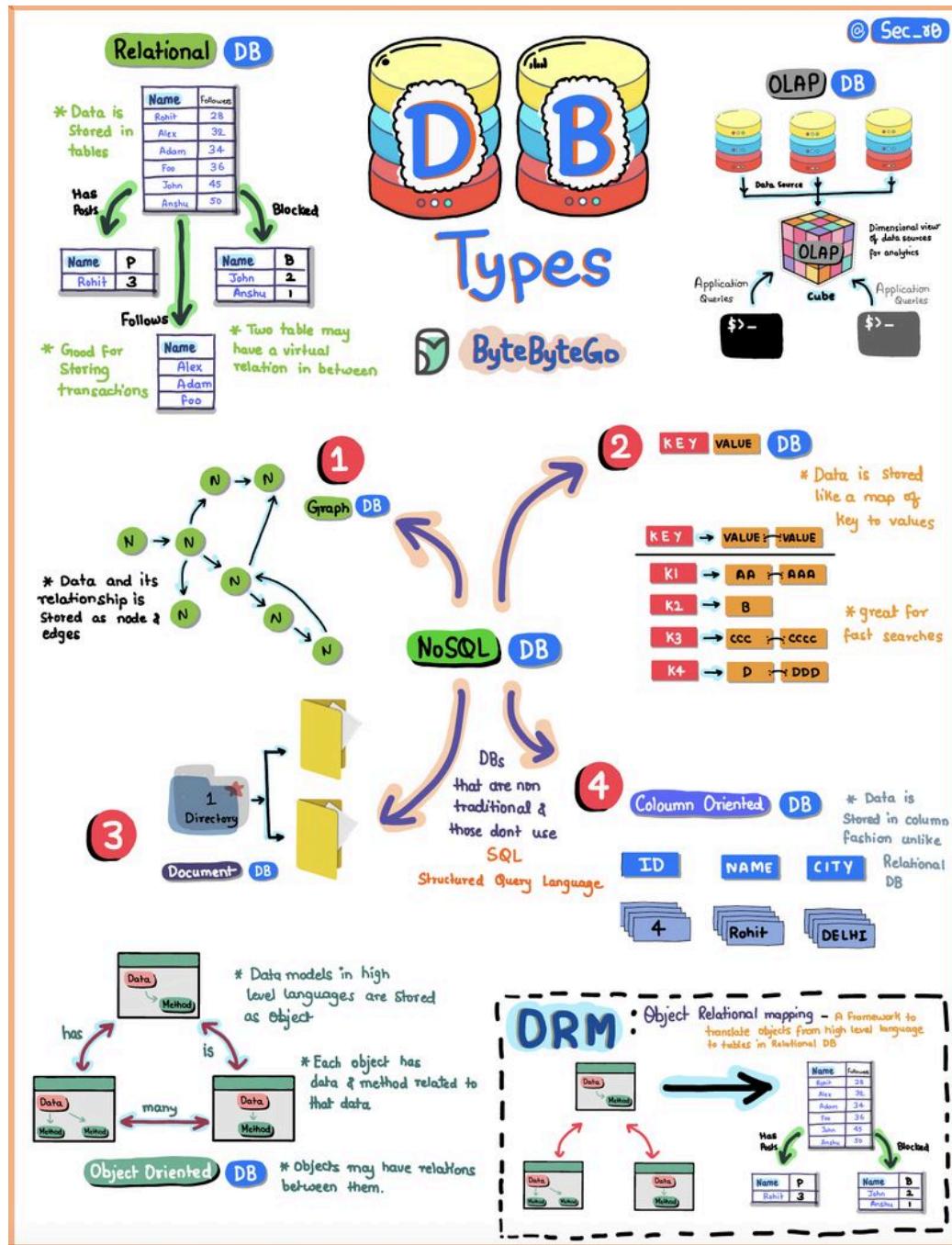
To validate a password, it can go through the following process:

1. A client enters the password.
2. The system fetches the corresponding salt from the database.
3. The system appends the salt to the password and hashes it. Let's call the hashed value H1.
4. The system compares H1 and H2, where H2 is the hash stored in the database. If they are the same, the password is valid.

Over to you: what other mechanisms can we use to ensure password safety?

What is a database? What are some common types of databases?

First off, what's a database? Think of it as a digital playground where we organize and store loads of information in a structured manner. Now, let's shake things up and look at the main types of databases.



Relational DB: Imagine it's like organizing data in neat tables. Think of it as the well-behaved sibling, keeping everything in order.

OLAP DB: Online Analytical Processing (OLAP) is a technology optimized for reporting and analysis purposes.

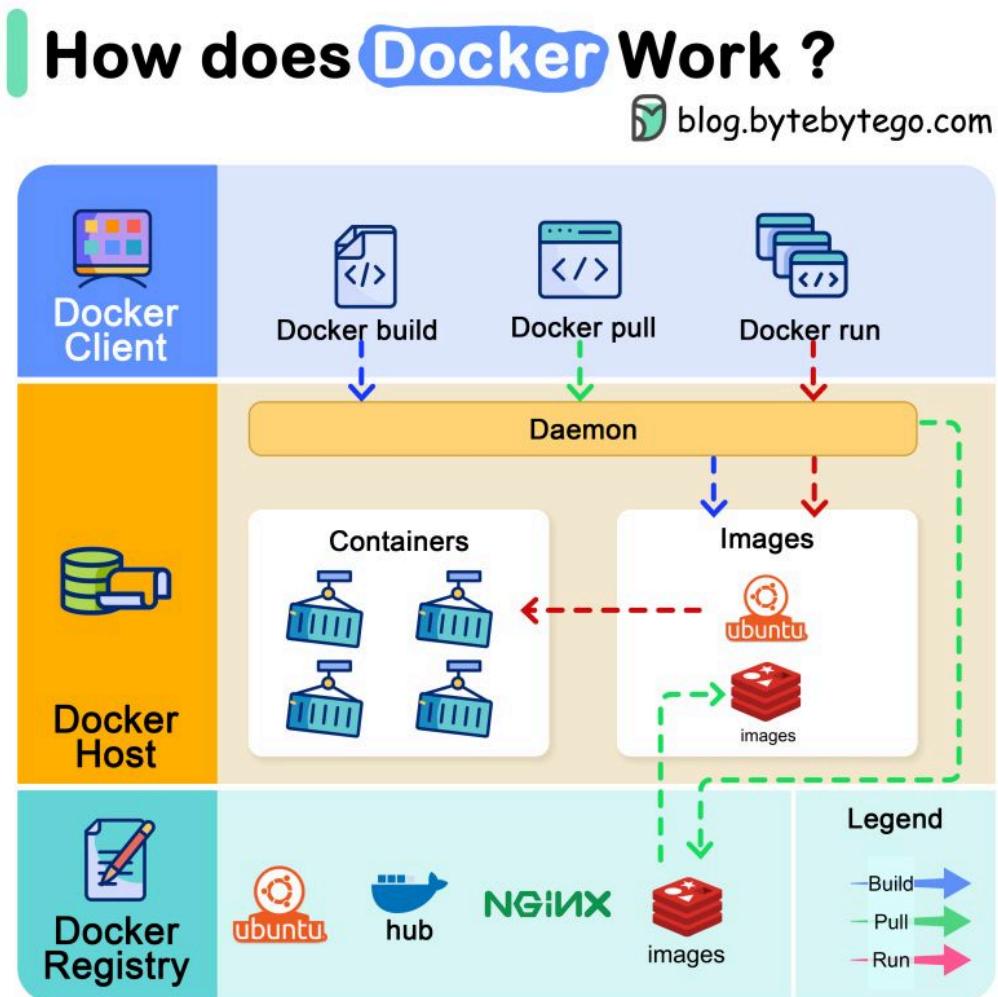
NoSQL DBs: These rebels have their own cool club, saying "No" to traditional SQL ways. NoSQL databases come in four exciting flavors:

- Graph DB: Think of social networks, where relationships between people matter most. It's like mapping who's friends with whom.
- Key-value Store DB: It's like a treasure chest, with each item having its unique key. Finding what you need is a piece of cake.
- Document DB: A document database is a kind of database that stores information in a format similar to JSON. It's different from traditional databases and is made for working with documents instead of tables.
- Column DB: Imagine slicing and dicing your data like a chef prepping ingredients. It's efficient and speedy.

Over to you: So, the next time you hear about databases, remember, it's a wild world out there - from orderly tables to rebellious NoSQL variants! Which one is your favorite? Share your thoughts!

How does Docker Work? Is Docker still relevant?

We just made a video on this topic.



Docker's architecture comprises three main components:

- Docker Client

This is the interface through which users interact. It communicates with the Docker daemon.

- Docker Host

Here, the Docker daemon listens for Docker API requests and manages various Docker objects, including images, containers, networks, and volumes.

- Docker Registry

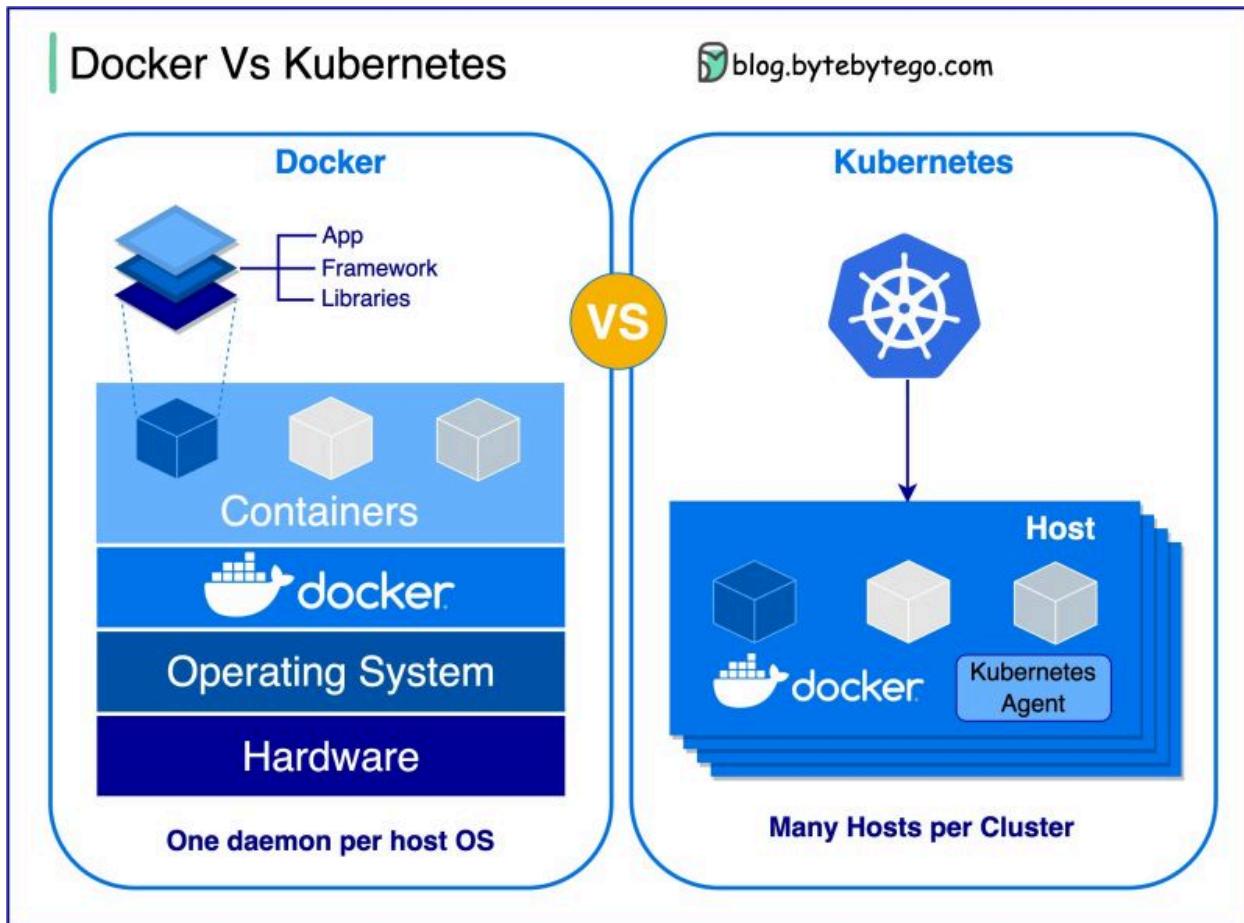
This is where Docker images are stored. Docker Hub, for instance, is a widely-used public registry.

Let's take the "docker run" command as an example.

1. Docker pulls the image from the registry.
2. Docker creates a new container.
3. Docker allocates a read-write filesystem to the container.
4. Docker creates a network interface to connect the container to the default network.
5. Docker starts the container.

Is Docker still relevant? Watch the whole video here: https://lnkd.in/eKDKkq_m

Docker vs. Kubernetes. Which one should we use?



What is Docker?

Docker is an open-source platform that allows you to package, distribute, and run applications in isolated containers. It focuses on containerization, providing lightweight environments that encapsulate applications and their dependencies.

What is Kubernetes?

Kubernetes, often referred to as K8s, is an open-source container orchestration platform. It provides a framework for automating the deployment, scaling, and management of containerized applications across a cluster of nodes.

How are both different from each other?

Docker: Docker operates at the individual container level on a single operating system host.

You must manually manage each host and setting up networks, security policies, and storage for multiple related containers can be complex.

Kubernetes: Kubernetes operates at the cluster level. It manages multiple containerized applications across multiple hosts, providing automation for tasks like load balancing, scaling, and ensuring the desired state of applications.

In short, Docker focuses on containerization and running containers on individual hosts, while Kubernetes specializes in managing and orchestrating containers at scale across a cluster of hosts.

Over to you: What challenges prompted you to switch from Docker to Kubernetes for managing containerized applications?

Writing Code that Runs on All Platforms

Developing code that functions seamlessly across different platforms is a crucial skill for modern programmers.

The need arises from the fact that users access software on a wide range of devices and operating systems. Achieving this universal compatibility can be complex due to differences in hardware, software environments, and user expectations.

Writing Code that Runs on All Platforms		
Context	Description	Trade-offs To Consider
 Cross Platform Language	Choose a cross-platform programming language or interpreter.	Constraints on speed, memory, syntax, and libraries
 Cross Platform Framework	Enables writing code once for multiple platforms	Constraints on customization and code overhead
 Abstract Platform Specific Code	Isolate platform-specific code into modules or classes	May increase performance overhead and code complexity
 Testing Across Platforms	Use emulators and simulators to simulate different environments	Demands time and resources for testing, revealing compatibility concerns
 Internationalization and Localization	Start with an adaptable code plan for multiple languages and regions	Requires multiple language files, possibly raising maintenance work
 Community and Forums	Engage in cross-platform dev communities for guidance and sharing	Over-reliance on community support can hinder problem-solving

Creating code that works on all platforms requires careful planning and understanding of the unique challenges presented by each platform.

Better planning and comprehension of cross-platform development not only streamline the process but also contribute to the long-term success of a software project.

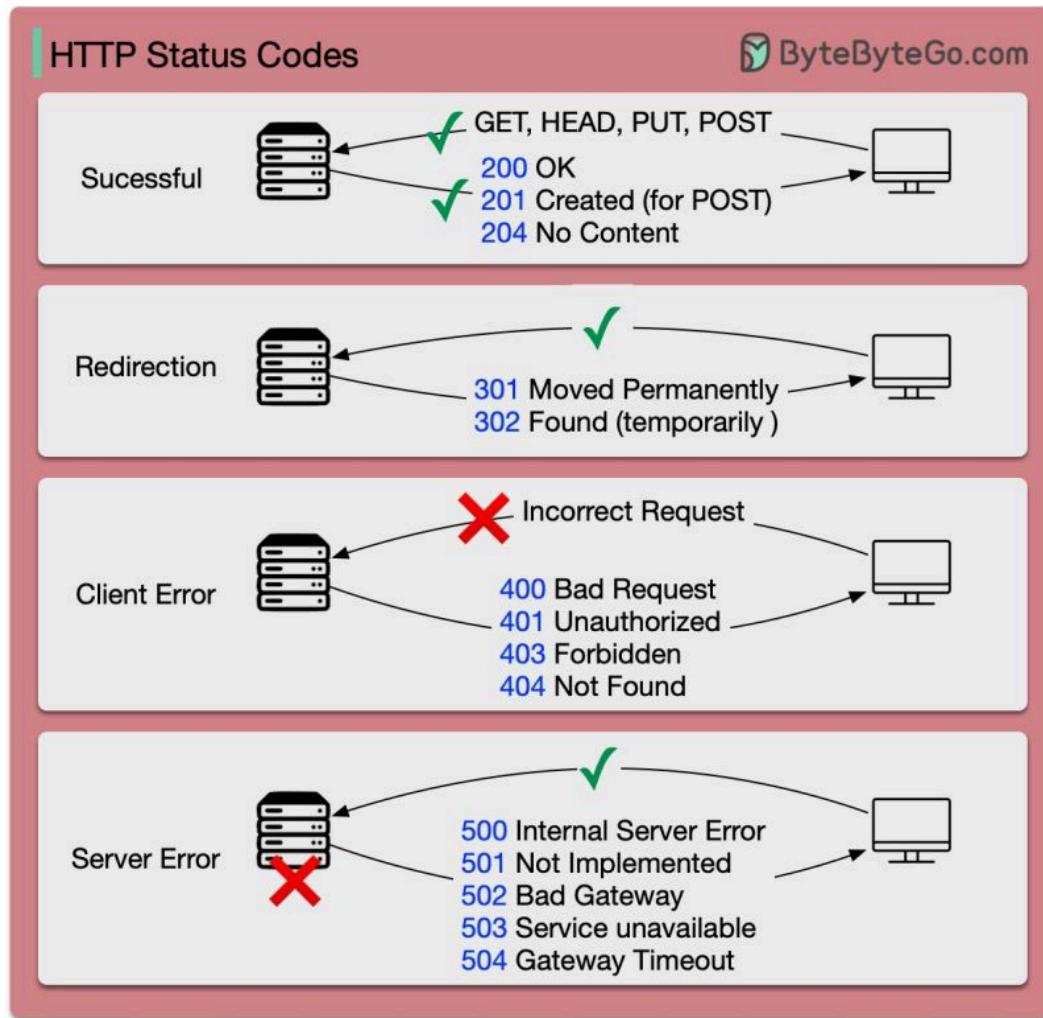
It reduces redundancy, simplifies maintenance, ensures consistency, boosting satisfaction and market reach.

Here are key factors for cross-platform compatibility

Over to you: How have you tackled cross-platform compatibility challenges in your projects?
Share your insights and experiences!

HTTP Status Code You Should Know

We just made a YouTube video on this topic. The link to the video is at the end of the post.



The response codes for HTTP are divided into five categories:

Informational (100-199)

Success (200-299)

Redirection (300-399)

Client Error (400-499)

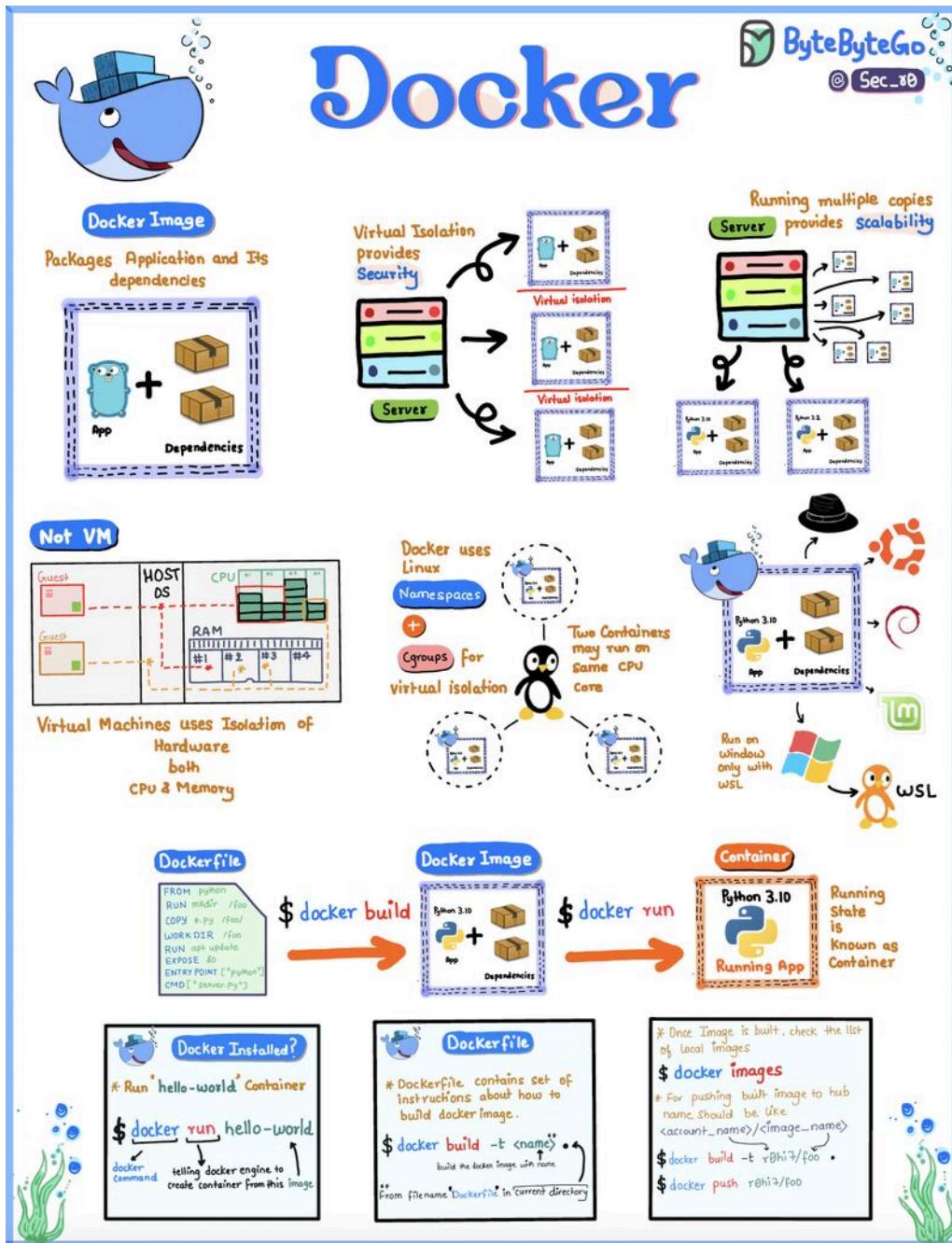
Server Error (500-599)

These codes are defined in RFC 9110. To save you from reading the entire document (which is about 200 pages), here is a summary of the most common ones:

Over to you: HTTP status code 401 is for Unauthorized. Can you explain the difference between authentication and authorization, and which one does code 401 check for?

Watch the whole video here: <https://lnkd.in/eZVjhXDt>

Docker 101: Streamlining App Deployment



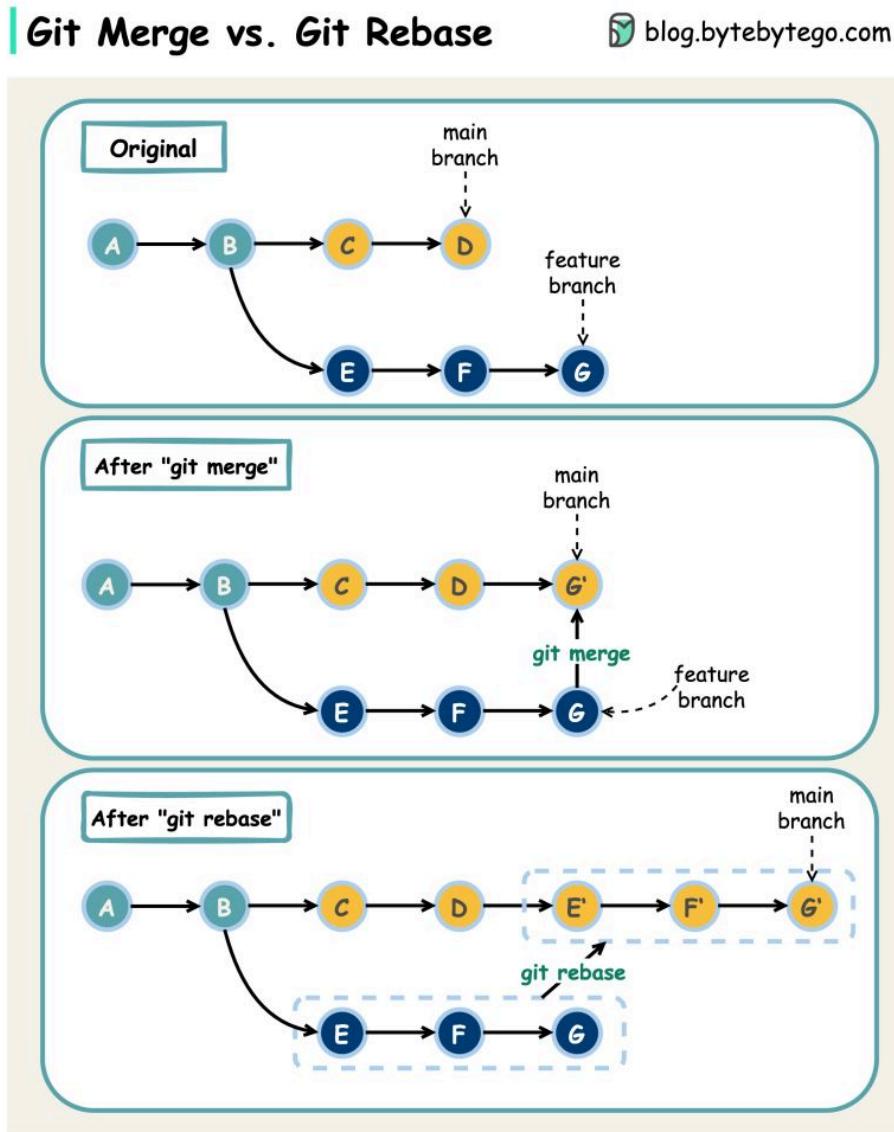
Fed up with the "it works on my machine" dilemma? Docker could be your salvation!

Docker revolutionizes software development and deployment. Explore the essentials:

1. Bundle Everything: Docker packs your app and its dependencies into a portable container – code, runtime, tools, libraries, and settings – a tidy, self-contained package.

2. Virtual Isolation: Containers offer packaging and isolation. Run diverse apps with different settings on a single host without conflicts, thanks to Linux namespaces and cgroups.
3. Not VMs: Unlike resource-heavy VMs, Docker containers share the host OS kernel, delivering speed and efficiency. No VM overhead, just rapid starts and easy management. 
4. Windows Compatibility: Docker, rooted in Linux, works on Windows too. Docker Desktop for Windows uses a Linux-based VM, enabling containerization for Windows apps.

Git Merge vs. Rebase vs. Squash Commit



What are the differences?

When we **merge changes** from one Git branch to another, we can use 'git merge' or 'git rebase'. The diagram below shows how the two commands work.

Git Merge

This creates a new commit **G'** in the main branch. **G'** ties the histories of both main and feature branches.

Git merge is **non-destructive**. Neither the main nor the feature branch is changed.

Git Rebase

Git rebase moves the feature branch histories to the head of the main branch. It creates new commits E', F', and G' for each commit in the feature branch.

The benefit of rebase is that it has **linear commit history**.

Rebase can be dangerous if “the golden rule of git rebase” is not followed.

The Golden Rule of Git Rebase

Never use it on public branches!

Cloud Network Components Cheat Sheet

Network components form the backbone of cloud infrastructure, enabling connectivity, scalability, and functionality in cloud services.

NETWORKING CHEAT SHEET <small>blog.bytebytego.com</small>				
Element	AWS	Azure	Google Cloud	Alibaba Cloud
Virtual Private Cloud	 Virtual Private Cloud	 Virtual Network	 Virtual Private Cloud	 Virtual Private Cloud
Subnetwork	 Subnet	 Subnet	 Subnetwork	 Vswitch
Load Balancer	 Elastic Load Balancer	 Load Balancer	 Cloud Load Balancing	 Server Load Balancer
Firewall	 Web Application Firewall	 Web Application Firewall	 Cloud Armor	 Web Application Firewall
Content Delivery Network	 Amazon CloudFront	 Content Delivery Network	 Cloud CDN	 Content Delivery Network
Dedicated Connectivity	 Direct Connect	 ExpressRoute	 Cloud Interconnect	 Express Connect
Virtual Private Network	 VPN Connection	 VPN Gateway	 Cloud VPN	 VPN Gateway
DDoS Protection	 Shield	 DDoS Protection	 Cloud Armor	 Anti-DDoS
Domain Name System	 Route 53	 DNS	 Cloud DNS	 DNS
Network Monitoring	 CloudWatch	 Azure Monitor	 Cloud Monitoring	 Log Service
Security Groups	 Security Groups	 Security Groups	 Firewall Rules	 Security Groups
Route Tables	 Route Tables	 Route Tables	 Routes	 Route Table
Network Peering	 VPC Peering	 VNet Peering	 VPC Network Peering	 VPC Peering
Content Distribution	 Global Accelerator	 Front Door	 Global Load Balancer	 Global Accelerator

These components include routers, load balancers, and firewalls, which ensure data flows efficiently and securely between servers and clients.

Additionally, Content Delivery Networks (CDNs) optimize content delivery by caching data at edge locations, reducing latency and improving user experience.

In essence, these network elements work together to create a robust and responsive cloud ecosystem that underpins modern digital services and applications.

This cheat sheet offers a concise yet comprehensive comparison of key network elements across the four major cloud providers.

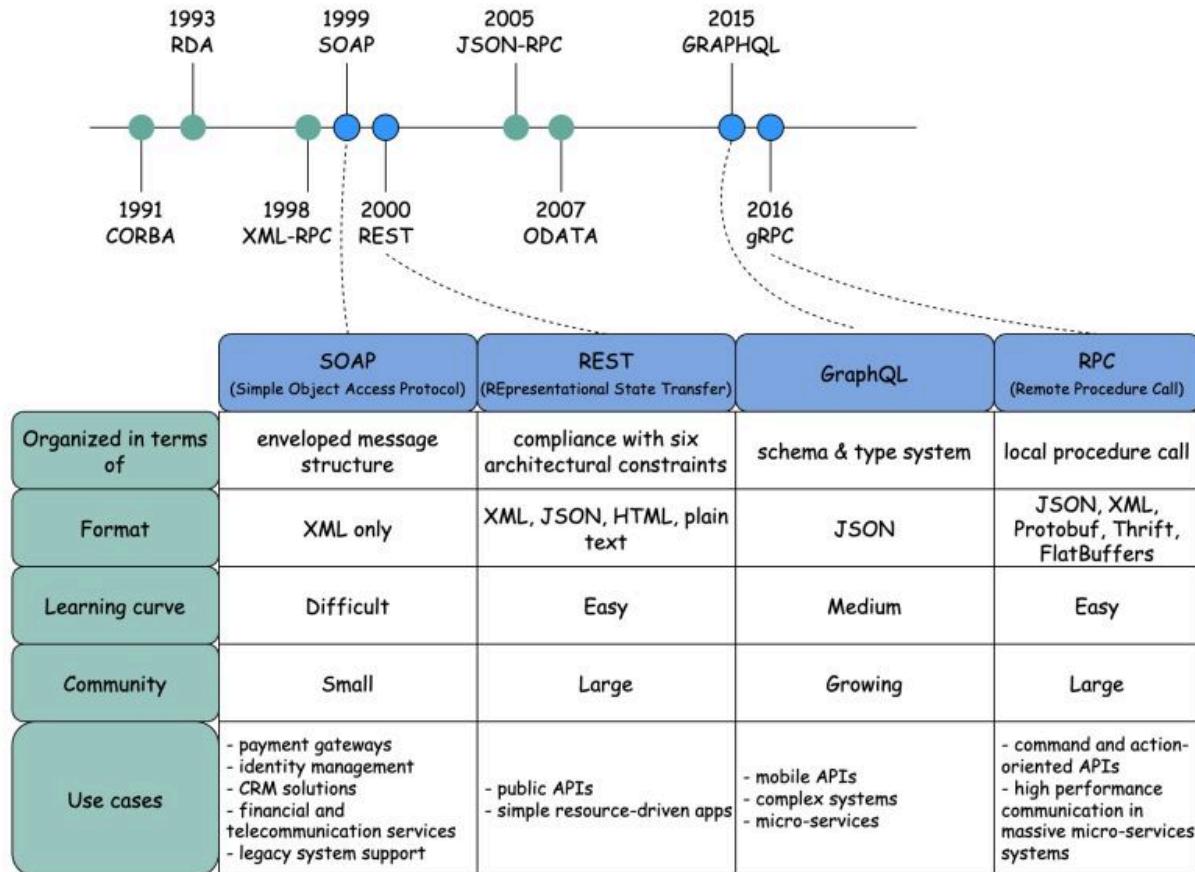
Over to you: How did you tackle the complexity of configuring and managing these network components?

SOAP vs REST vs GraphQL vs RPC

The diagram below illustrates the API timeline and API styles comparison.

API Architectural Styles Comparison

Source: altexsoft



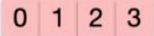
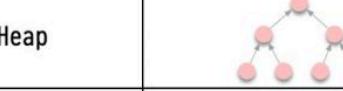
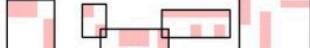
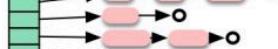
Over time, different API architectural styles are released. Each of them has its own patterns of standardizing data exchange.

You can check out the use cases of each style in the diagram.

10 Key Data Structures We Use Every Day

10 Data Structures Used in Daily Life

 ByteByteGo.com

Data Structure	Illustration	Use Cases
List		Twitter feeds
Array		Math operations Large data sets
Stack		Undo/Redo of word editor
Queue		Printer jobs User actions in game
Heap		Task scheduling
Tree		HTML document AI decision
Suffix Tree		Search string in document
Graph		Friendship tracking Path finding
R-tree		Nearest neighbour
Hash Table		Caching systems

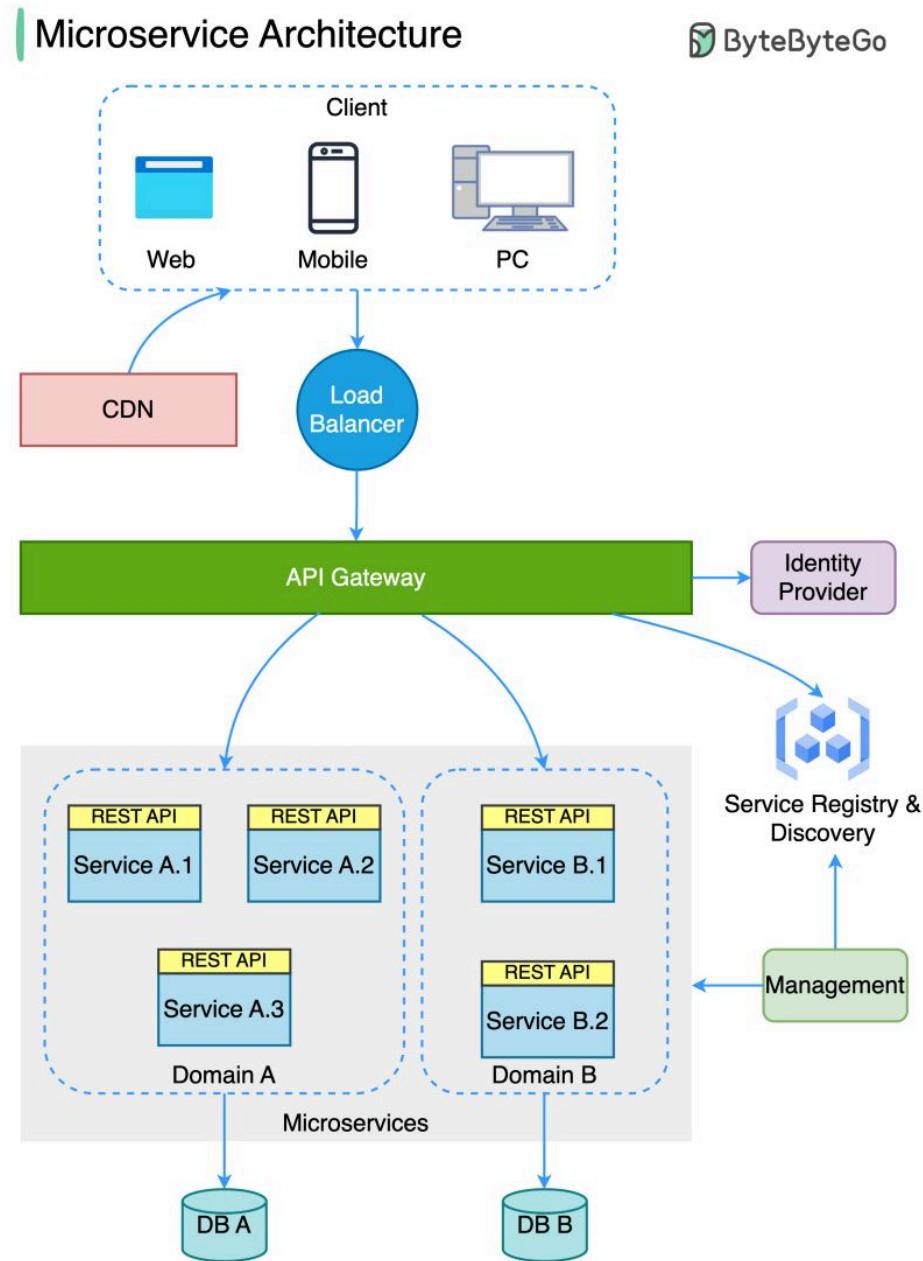
- list: keep your Twitter feeds
- stack: support undo/redo of the word editor
- queue: keep printer jobs, or send user actions in-game
- hash table: caching systems
- Array: math operations
- heap: task scheduling
- tree: keep the HTML document, or for AI decision
- suffix tree: for searching string in a document
- graph: for tracking friendship, or path finding

- r-tree: for finding the nearest neighbor
- vertex buffer: for sending data to GPU for rendering

Over to you: Which additional data structures have we overlooked?

What does a typical microservice architecture look like? 👉

The diagram below shows a typical microservice architecture.



- Load Balancer: This distributes incoming traffic across multiple backend services.
- CDN (Content Delivery Network): CDN is a group of geographically distributed servers that hold static content for faster delivery. The clients look for content in CDN first, then progress to backend services.

- API Gateway: This handles incoming requests and routes them to the relevant services. It talks to the identity provider and service discovery.
- Identity Provider: This handles authentication and authorization for users.
- Service Registry & Discovery: Microservice registration and discovery happen in this component, and the API gateway looks for relevant services in this component to talk to.
- Management: This component is responsible for monitoring the services.
- Microservices: Microservices are designed and deployed in different domains. Each domain has its own database. The API gateway talks to the microservices via REST API or other protocols, and the microservices within the same domain talk to each other using RPC (Remote Procedure Call).

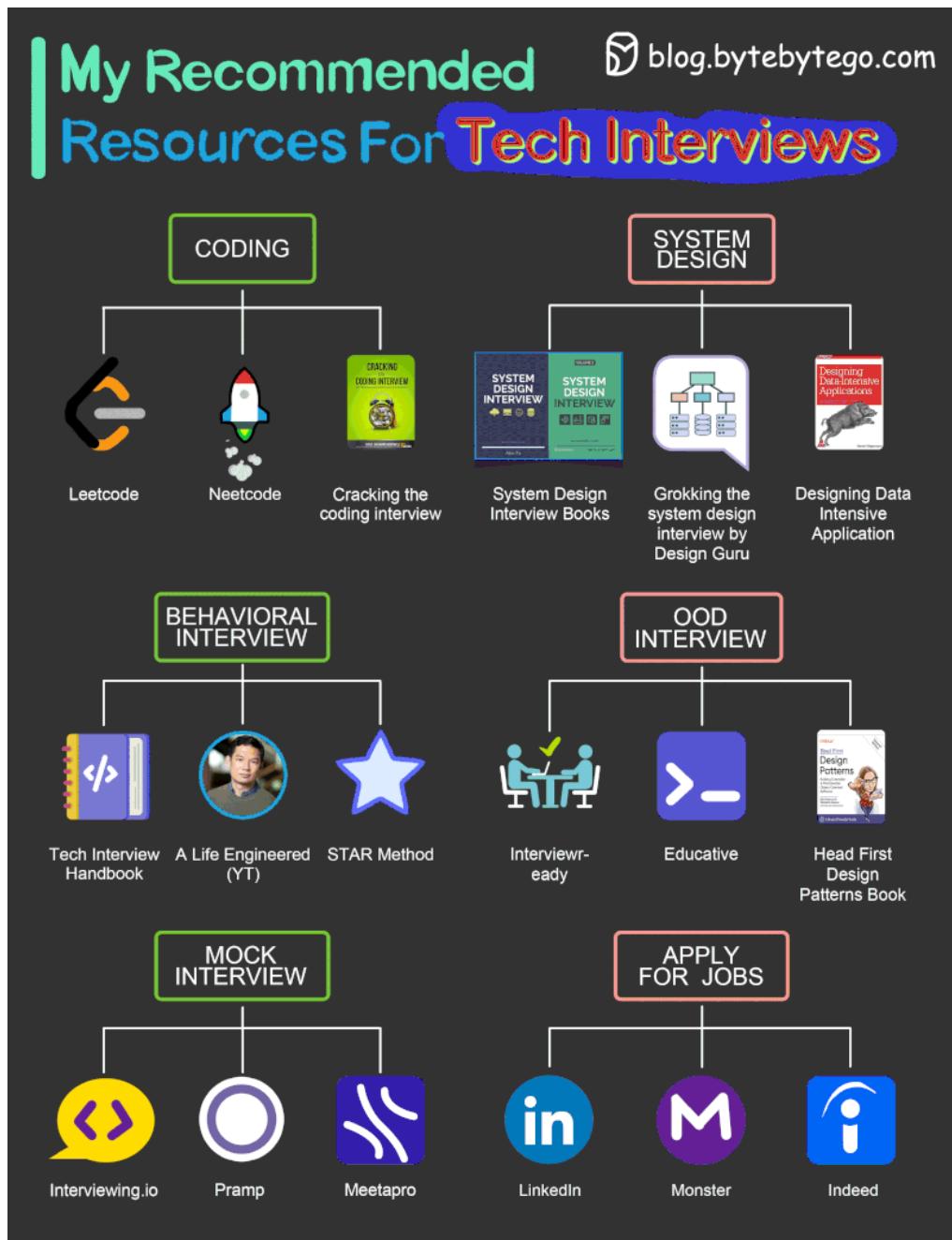
Benefits of microservices:

- They can be quickly designed, deployed, and horizontally scaled.
- Each domain can be independently maintained by a dedicated team.
- Business requirements can be customized in each domain and better supported, as a result.

Over to you:

1. What are the drawbacks of the microservice architecture?
2. Have you seen a monolithic system be transformed into microservice architecture? How long does it take?

My recommended materials for cracking your next technical interview



Coding

- Leetcode
- Cracking the coding interview book
- Neetcode

System Design Interview

- System Design Interview book 1, 2 by Alex Xu, Sahn Lam
- Grokking the system design by Design Guru
- Design Data-intensive Application book

Behavioral interview

- Tech Interview Handbook (Github repo)
- A Life Engineered (YT)
- STAR method (general method)

OOD Interview

- Interviewready
- OOD by educative
- Head First Design Patterns Book

Mock interviews

- Interviewingio
- Pramp
- Meetapro

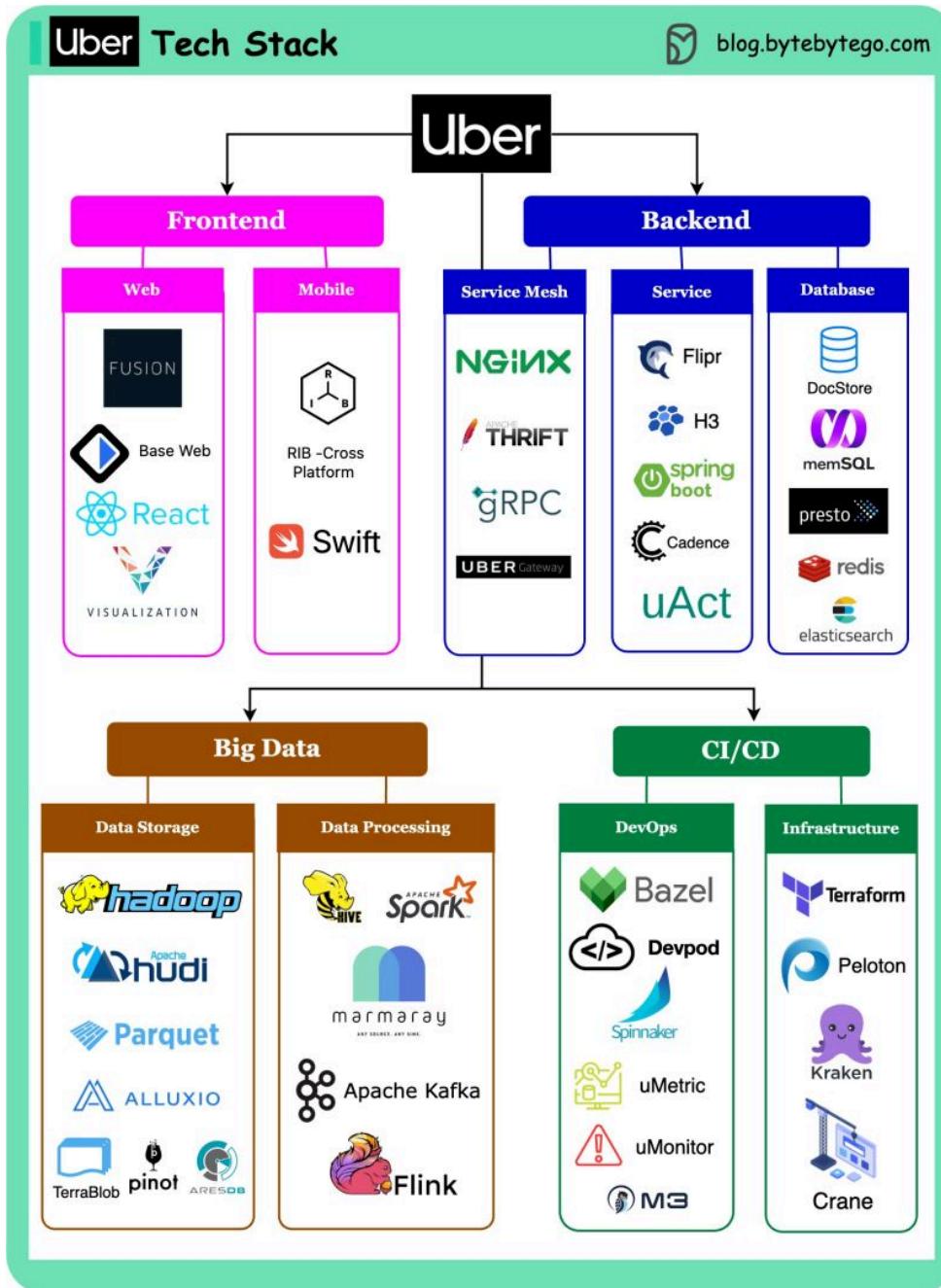
Apply for Jobs

- Linkedin
- Monster
- Indeed

Over to you: What is your favorite interview prep material?

Uber Tech Stack

This post is based on research from many Uber engineering blogs and open-source projects. If you come across any inaccuracies, please feel free to inform us. The corresponding links are added in the comment section.



Web frontend: Uber builds Fusion.js as a modern React framework to create robust web applications. They also develop visualization.js for geospatial visualization scenarios.

Mobile side: Uber builds the RIB cross-platform with the VIPER architecture instead of MVC. This architecture can work with different languages: Swift for iOS, and Java for Android.

Service mesh: Uber built Uber Gateway as a dynamic configuration on top of NGINX. The service uses gRPC and QUIC for client-server communication, and Apache Thrift for API definition.

Service side: Uber built a unified configuration store named Flirp (later changed to UCDP), H3 as a location-index store library. They use Spring Boot for Java-based services, uAct for event-driven architecture, and Cadence for async workflow orchestration.

Database end: the OLTP mainly uses the strongly-consistent DocStore, which employs MySQL and PostgreSQL, along with the RocksDB database engine.

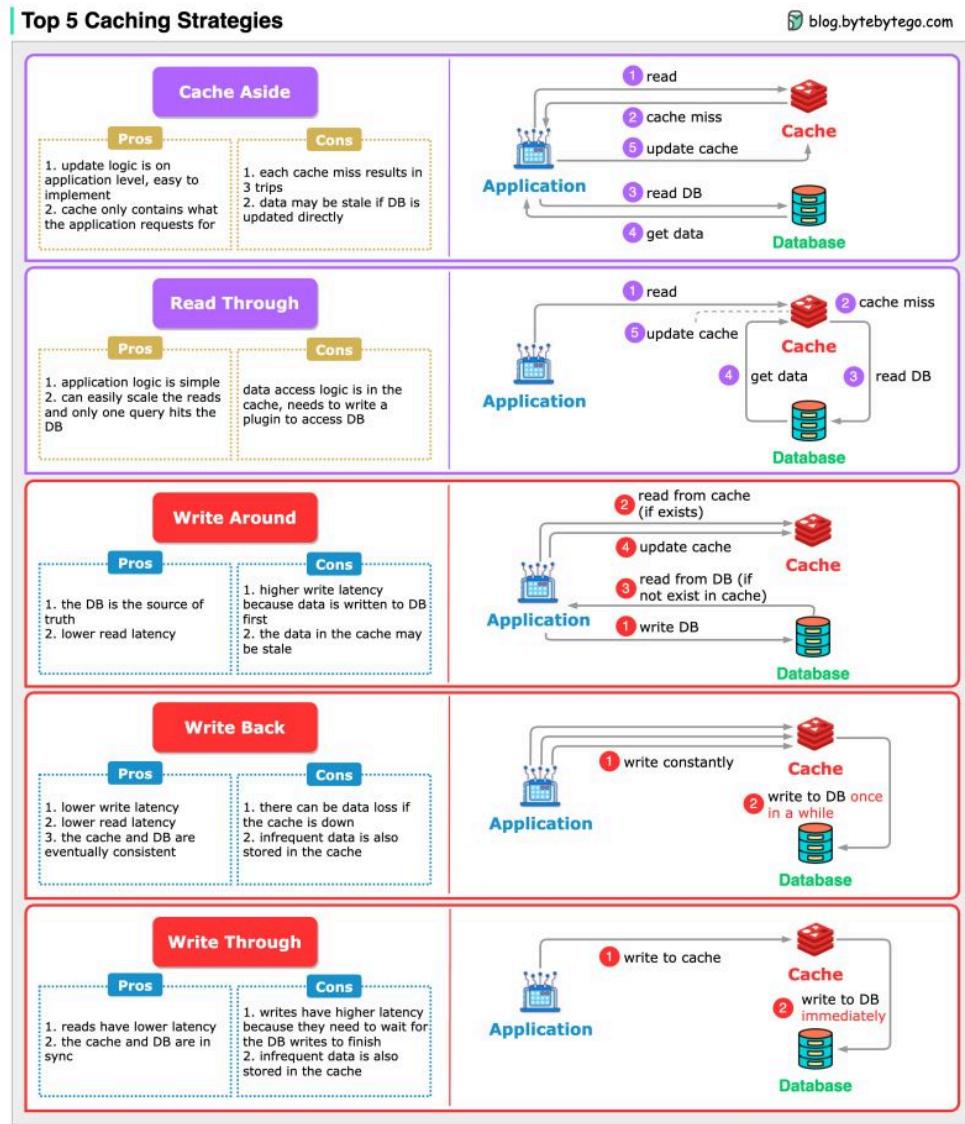
Big data: managed through the Hadoop family. Hudi and Parquet are used as file formats, and Alluxio serves as cache. Time-series data is stored in Pinot and AresDB.

Data processing: Hive, Spark, and the open-source data ingestion framework Marmaray. Messaging and streaming middleware include Apache Kafka and Apache Flink.

DevOps side: Uber utilizes a Monorepo, with a simplified development environment called devpod. Continuous delivery is managed through Netflix Spinnaker, metrics are emitted to uMetric, alarms on uMonitor, and a consistent observability database M3.

Top 5 Caching Strategies

When we introduce a cache into the architecture, synchronization between the cache and the database becomes inevitable.



Let's look at 5 common strategies how we keep the data in sync.

- Read Strategies:

Cache aside

Read through

- Write Strategies:

Write around

Write back

Write through

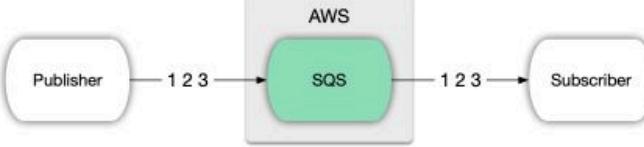
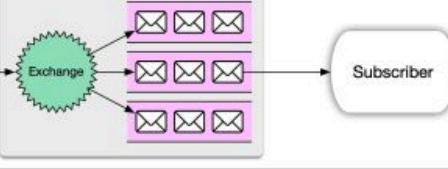
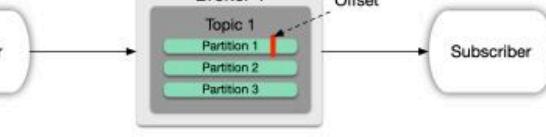
The caching strategies are often used in combination. For example, write-around is often used together with cache-aside to make sure the cache is up-to-date.

Over to you: What strategies have you used?

How many message queues do you know?

Types of Message Queues

 ByteByteGo.com

Name	Simplified Architecture	Killing Feature
ActiveMQ		Rich protocols
Amazon SQS		Message ordering and exact-once consumption
RabbitMQ		Message routing
Kafka		High throughput and reliability

Like a post office, a message queue helps computer programs to communicate in an organized manner. Imagine little digital envelopes being passed around to keep everything on track. There are few key features to consider when selecting message queues:

- Speed: How fast messages are sent and received
- Scalability: Can it grow with more messages
- Reliability: Will it make sure messages don't get lost
- Durability: Can it keep messages safe over time
- Ease of Use: Is it simple to set up and manage
- Ecosystem: Are there helpful tools available
- Integration: Can it play nice with other software
- Protocol Support: What languages can it speak

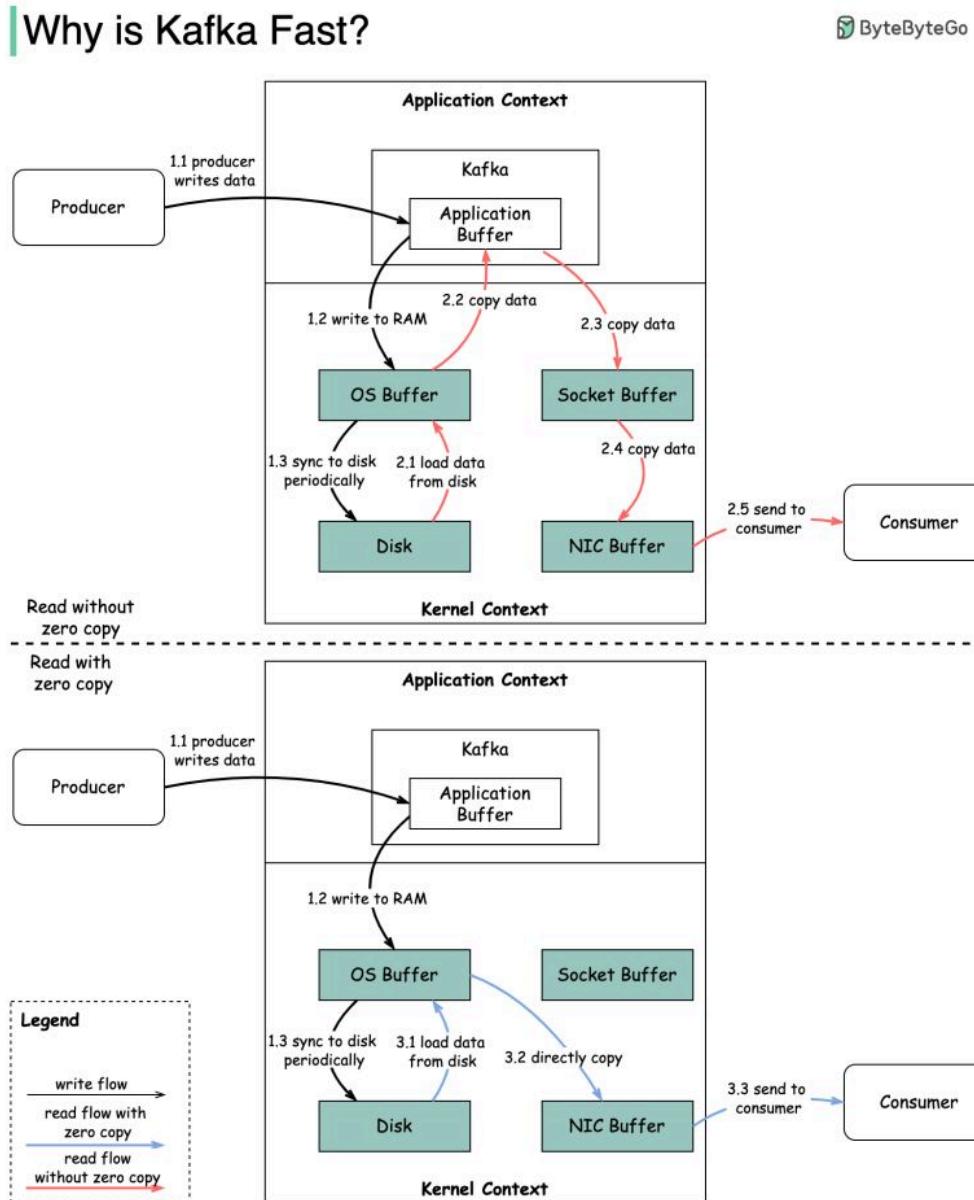
Try out a message queue and practice sending and receiving messages until you're comfortable. Choose an easy one like Kafka and experiment with sending and receiving messages. Read books or take online courses as you get more comfortable. Build little projects and learn from those who have already been there. Soon, you'll know everything about message queues.

Why is Kafka fast?

There are many design decisions that contributed to Kafka's performance. In this post, we'll focus on two. We think these two carried the most weight.

1. The first one is Kafka's reliance on Sequential I/O.
2. The second design choice that gives Kafka its performance advantage is its focus on efficiency: zero copy principle.

The diagram below illustrates how the data is transmitted between producer and consumer, and what zero-copy means.

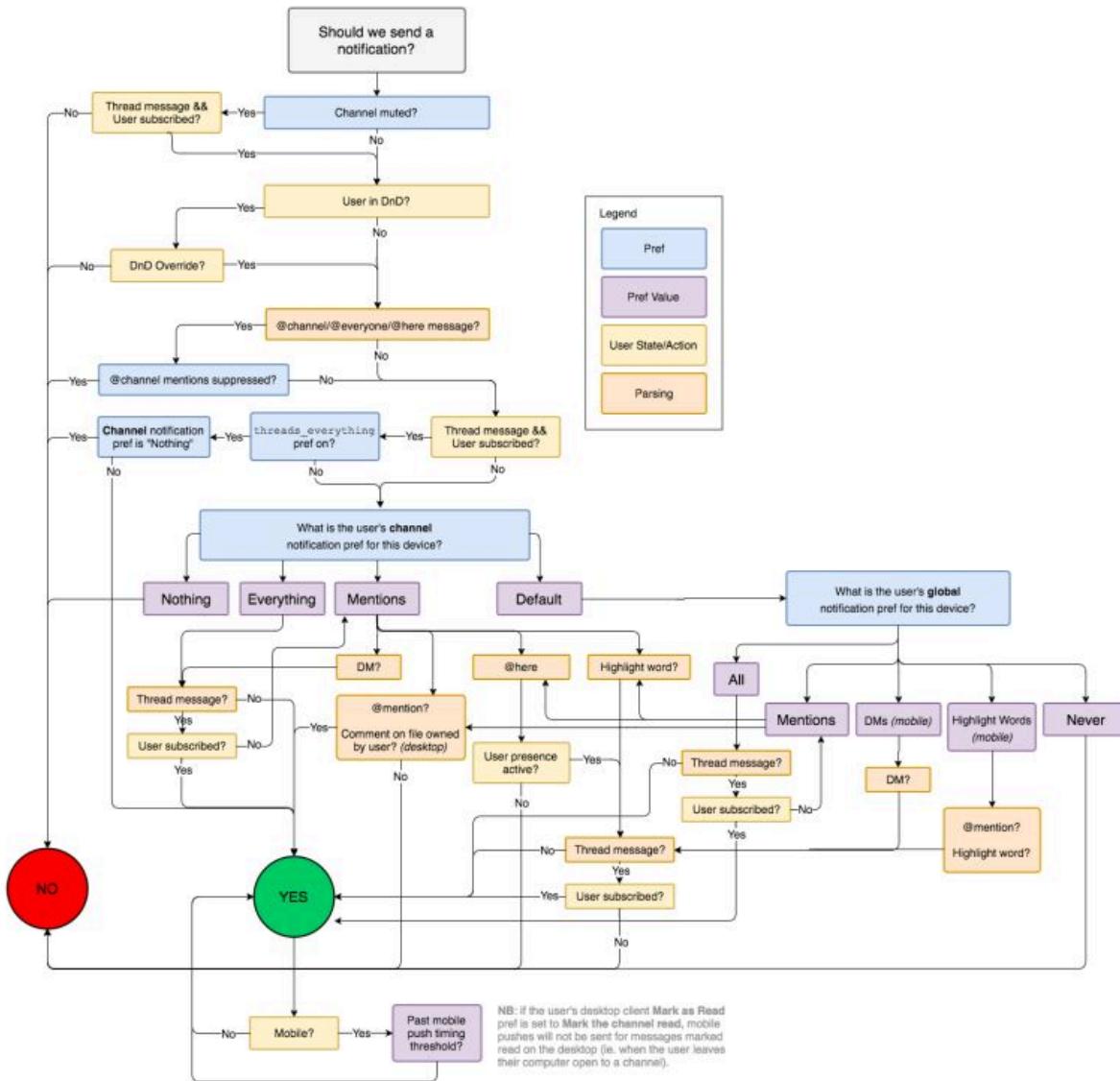


- Step 1.1 - 1.3: Producer writes data to the disk
- Step 2: Consumer reads data without zero-copy
 - 2.1: The data is loaded from disk to OS cache
 - 2.2 The data is copied from OS cache to Kafka application
 - 2.3 Kafka application copies the data into the socket buffer
 - 2.4 The data is copied from socket buffer to network card
 - 2.5 The network card sends data out to the consumer
- Step 3: Consumer reads data with zero-copy
 - 3.1: The data is loaded from disk to OS cache
 - 3.2 OS cache directly copies the data to the network card via sendfile() command
 - 3.3 The network card sends data out to the consumer

Zero copy is a shortcut to save multiple data copies between the application context and kernel context.

How slack decides to send a notification

This is the flowchart of how slack decides to send a notification.



It is a great example of why a simple feature may take much longer to develop than many people think.

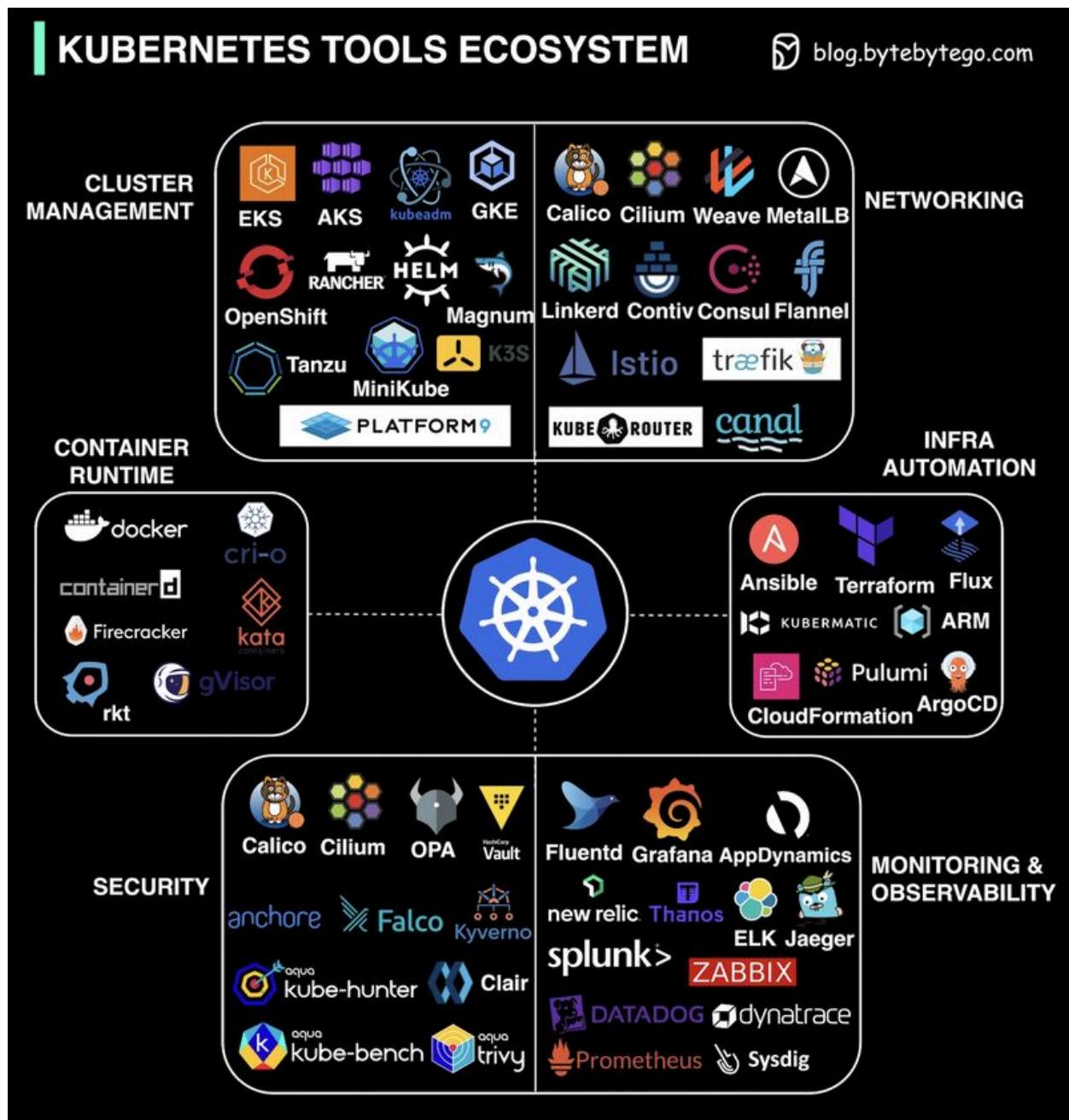
When we have a great design, users may not notice the complexity because it feels like the feature is just working as intended.

What's your takeaway from this diagram?

Source: [Slack Engineering Blog](#)

Kubernetes Tools Ecosystem

Kubernetes, the leading container orchestration platform, boasts a vast ecosystem of tools and components that collectively empower organizations to efficiently deploy, manage, and scale containerized applications.



Kubernetes practitioners need to be well-versed in these tools to ensure the reliability, security, and performance of containerized applications within Kubernetes clusters.

To introduce a holistic view of the Kubernetes ecosystem, we've created an illustration covering the aspects of:

1. Security
2. Networking
3. Container Runtime
4. Cluster Management
5. Monitoring and Observability
6. Infrastructure Orchestration

Over to you:

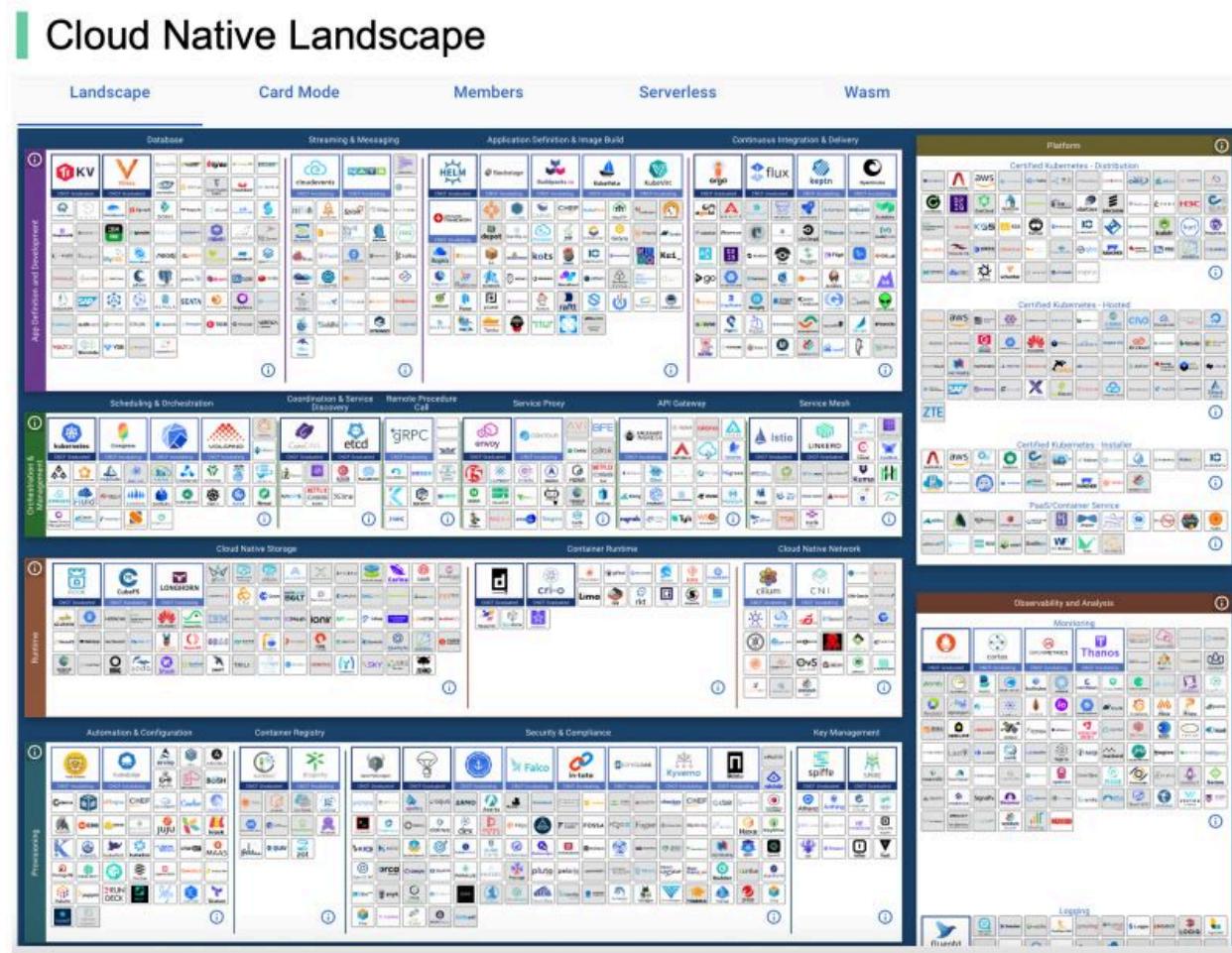
How have Kubernetes tools enhanced your containerized application management?

Cloud Native Landscape

Many Are Looking for the Definitive Guide on How to Choose the Right Stack

The ANSWER is...

There is no one-size-fits-all guide; it all depends on your specific needs, and picking the right stack is HARD.



Fortunately, at this point in time, technology is usually no longer a limiting factor. Most startups should be able to get by with most technologies they find. So spend less time on picking the perfect tech; instead, focus on your customers and keep building.

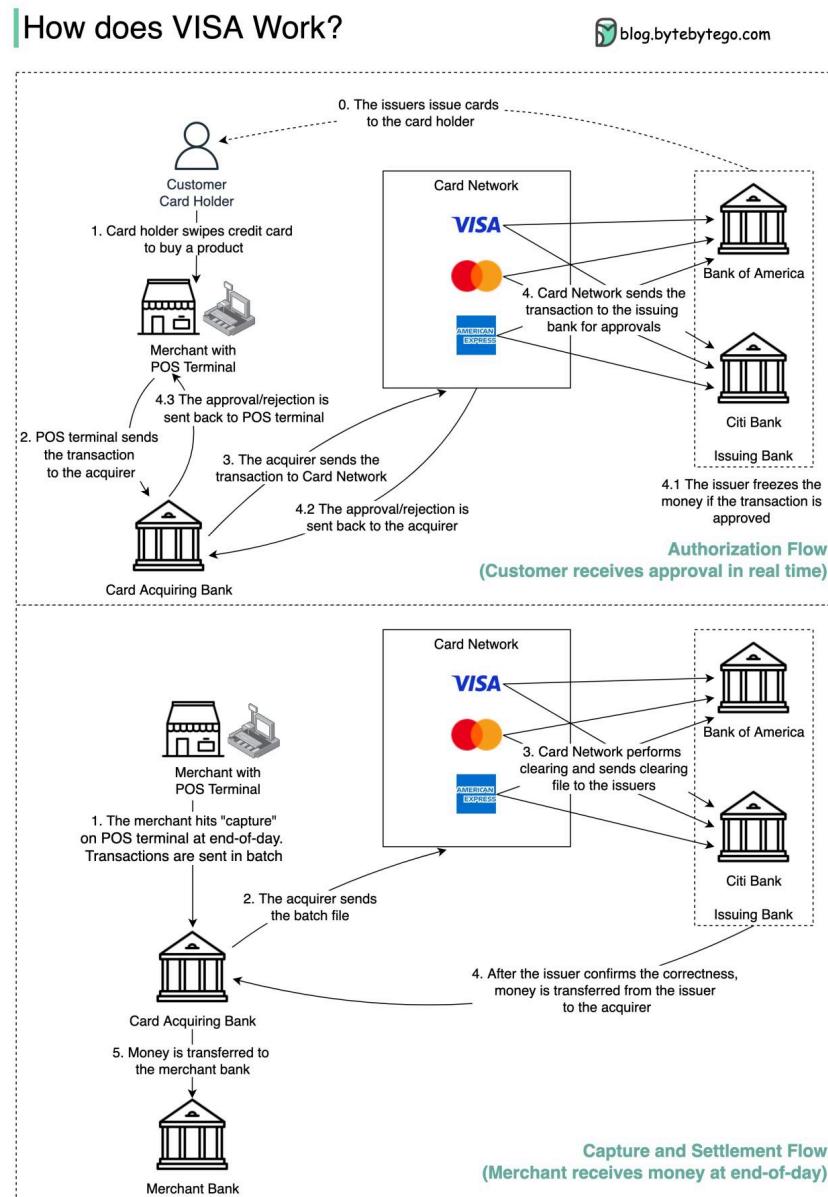
Over to you all: What do you think is causing this fragmentation in tech stack choices?

Source: [CNCF Cloud Native Interactive Landscape](#)

How does VISA work when we swipe a credit card at a merchant's shop?

VISA, Mastercard, and American Express act as card networks for the clearing and settling of funds. The card acquiring bank and the card issuing bank can be – and often are – different. If banks were to settle transactions one by one without an intermediary, each bank would have to settle the transactions with all the other banks. This is quite inefficient.

The diagram below shows VISA's role in the credit card payment process. There are two flows involved. Authorization flow happens when the customer swipes the credit card. Capture and settlement flow happens when the merchant wants to get the money at the end of the day.



- Authorization Flow

Step 0: The card issuing bank issues credit cards to its customers.

Step 1: The cardholder wants to buy a product and swipes the credit card at the Point of Sale (POS) terminal in the merchant's shop.

Step 2: The POS terminal sends the transaction to the acquiring bank, which has provided the POS terminal.

Steps 3 and 4: The acquiring bank sends the transaction to the card network, also called the card scheme. The card network sends the transaction to the issuing bank for approval.

Steps 4.1, 4.2 and 4.3: The issuing bank freezes the money if the transaction is approved. The approval or rejection is sent back to the acquirer, as well as the POS terminal.

- Capture and Settlement Flow

Steps 1 and 2: The merchant wants to collect the money at the end of the day, so they hit "capture" on the POS terminal. The transactions are sent to the acquirer in batch. The acquirer sends the batch file with transactions to the card network.

Step 3: The card network performs clearing for the transactions collected from different acquirers, and sends the clearing files to different issuing banks.

Step 4: The issuing banks confirm the correctness of the clearing files, and transfer money to the relevant acquiring banks.

Step 5: The acquiring bank then transfers money to the merchant's bank.

Step 4: The card network clears the transactions from different acquiring banks. Clearing is a process in which mutual offset transactions are netted, so the number of total transactions is reduced.

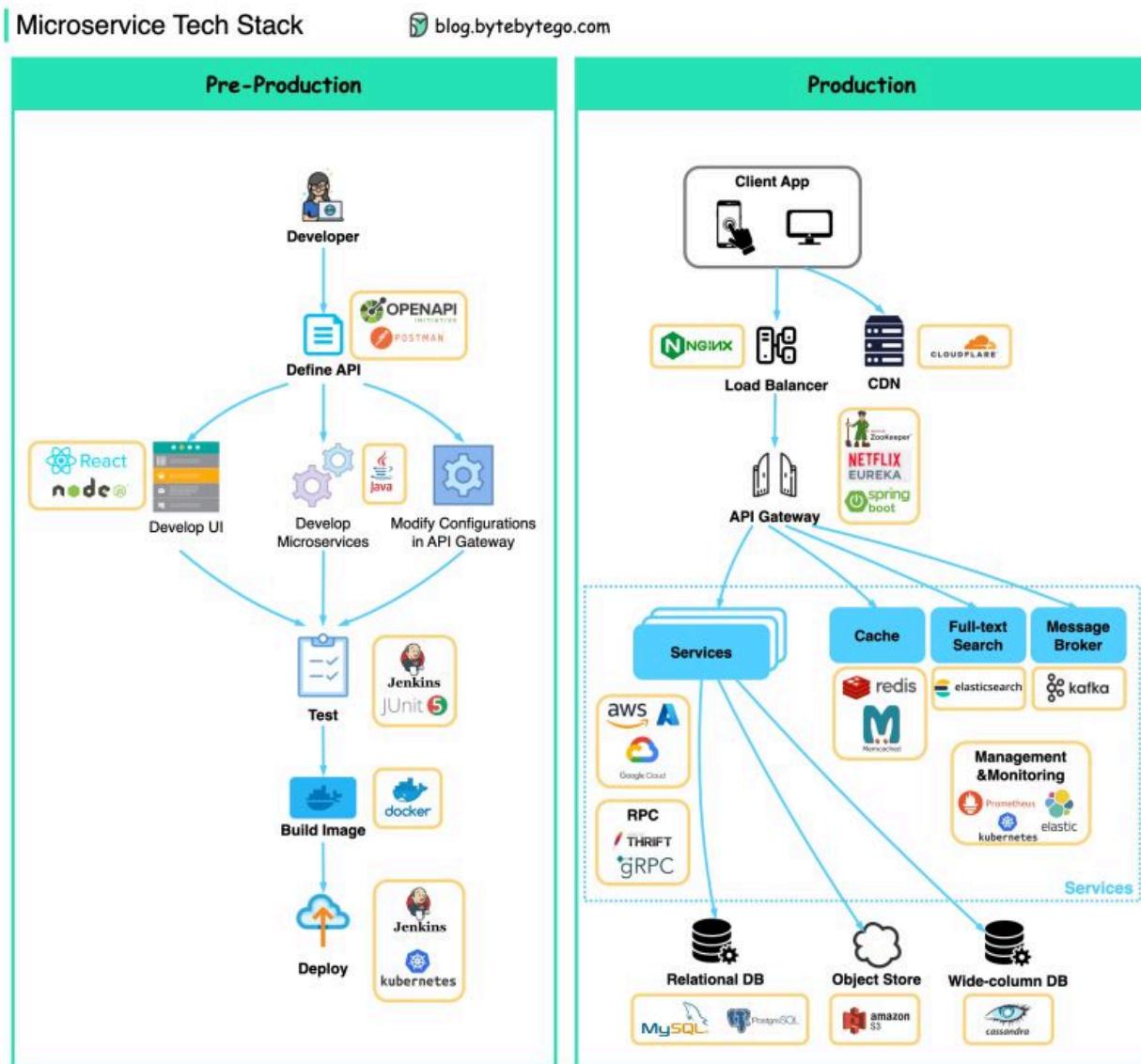
In the process, the card network takes on the burden of talking to each bank and receives service fees in return.

Over to you: Do you think this flow is way too complicated? What will be the future of payments in your opinion?

A simple visual guide to help people understand the key considerations when designing or using caching systems

What tech stack is commonly used for microservices?

Below you will find a diagram showing the microservice tech stack, both for the development phase and for production.



► Pre-production

- Define API - This establishes a contract between frontend and backend. We can use Postman or OpenAPI for this.
- Development - Node.js or react is popular for frontend development, and java/python/go for backend development. Also, we need to change the configurations in the API gateway according to API definitions.

- Continuous Integration - JUnit and Jenkins for automated testing. The code is packaged into a Docker image and deployed as microservices.

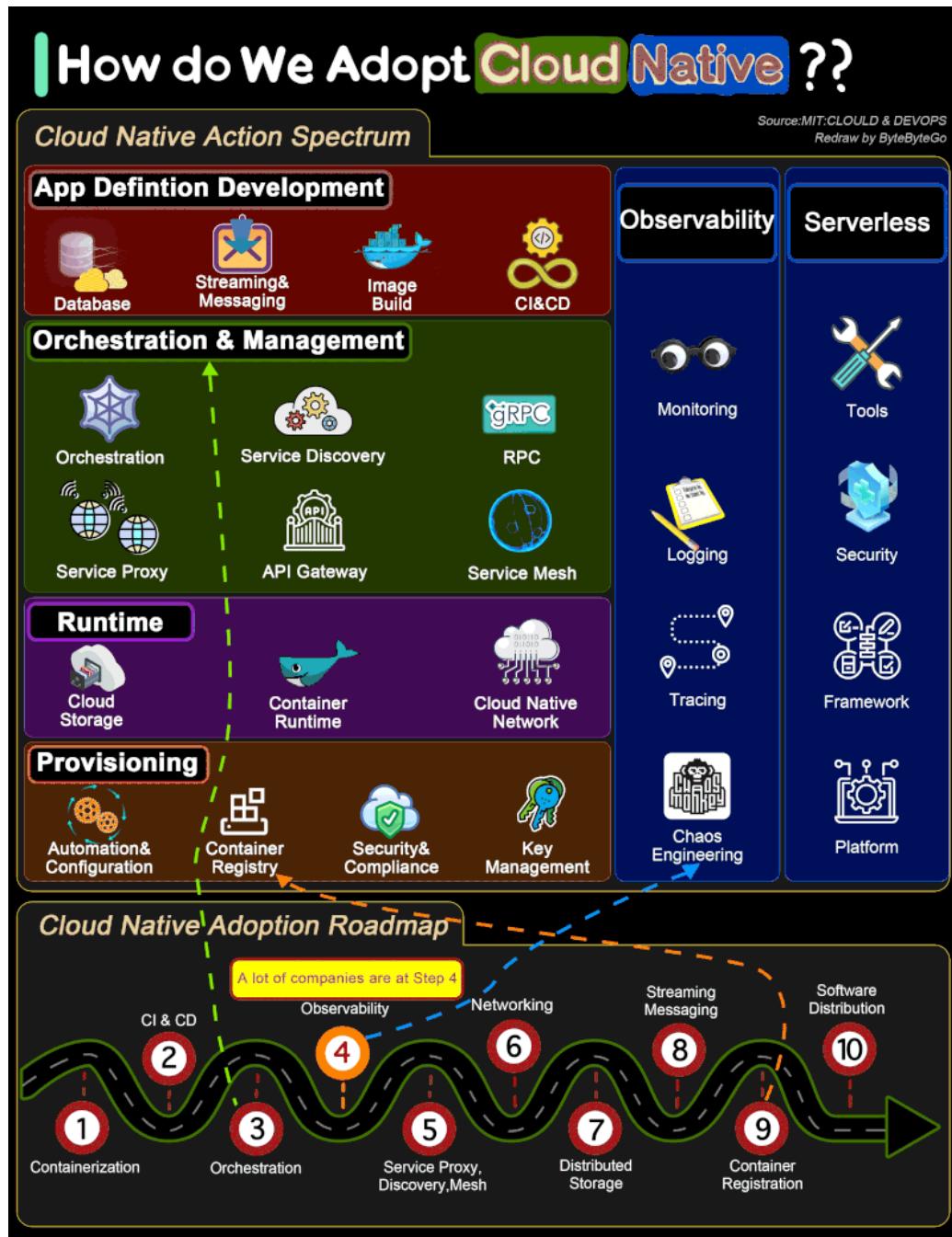
Production

- NGinx is a common choice for load balancers. Cloudflare provides CDN (Content Delivery Network).
- API Gateway - We can use spring boot for the gateway, and use Eureka/Zookeeper for service discovery.
- The microservices are deployed on clouds. We have options among AWS, Microsoft Azure, or Google GCP.
- Cache and Full-text Search - Redis is a common choice for caching key-value pairs. ElasticSearch is used for full-text search.
- Communications - For services to talk to each other, we can use messaging infra Kafka or RPC.
- Persistence - We can use MySQL or PostgreSQL for a relational database, and Amazon S3 for object store. We can also use Cassandra for the wide-column store if necessary.
- Management & Monitoring - To manage so many microservices, the common Ops tools include Prometheus, Elastic Stack, and Kubernetes.

Over to you: Did I miss anything? Please comment on what you think is necessary to learn microservices.

How do we transform a system to be Cloud Native?

The diagram below shows the action spectrum and adoption roadmap. You can use it as a blueprint for adopting cloud-native in your organization.



For a company to adopt cloud native architecture, there are 6 aspects in the spectrum:

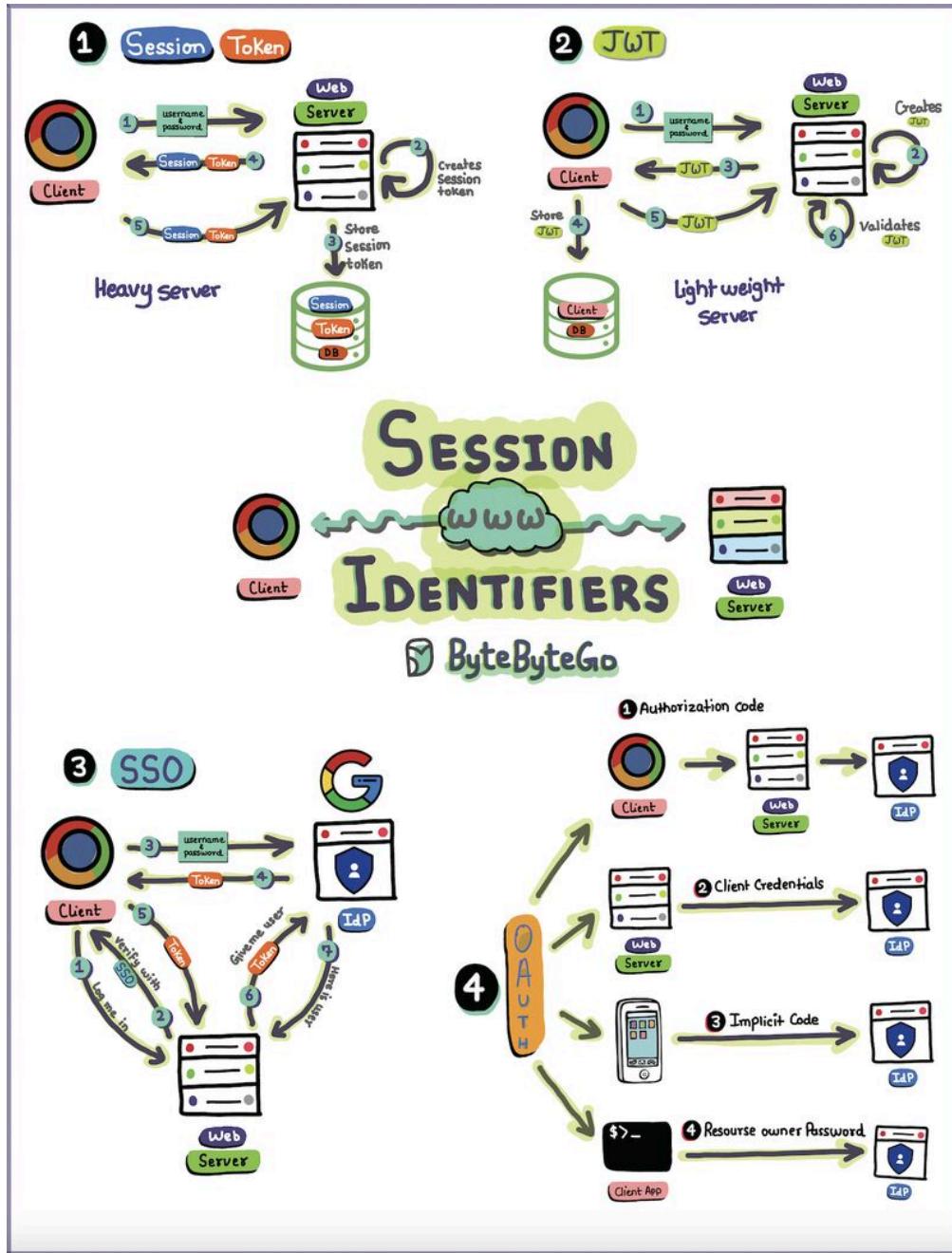
1. Application definition development
2. Orchestration and management

- 3. Runtime
- 4. Provisioning
- 5. Observability
- 6. Serverless

Over to you: Where does your system stand in the adoption roadmap?

Reference: Cloud & DevOps: Continuous Transformation by MIT
Redrawn by ByteByteGo

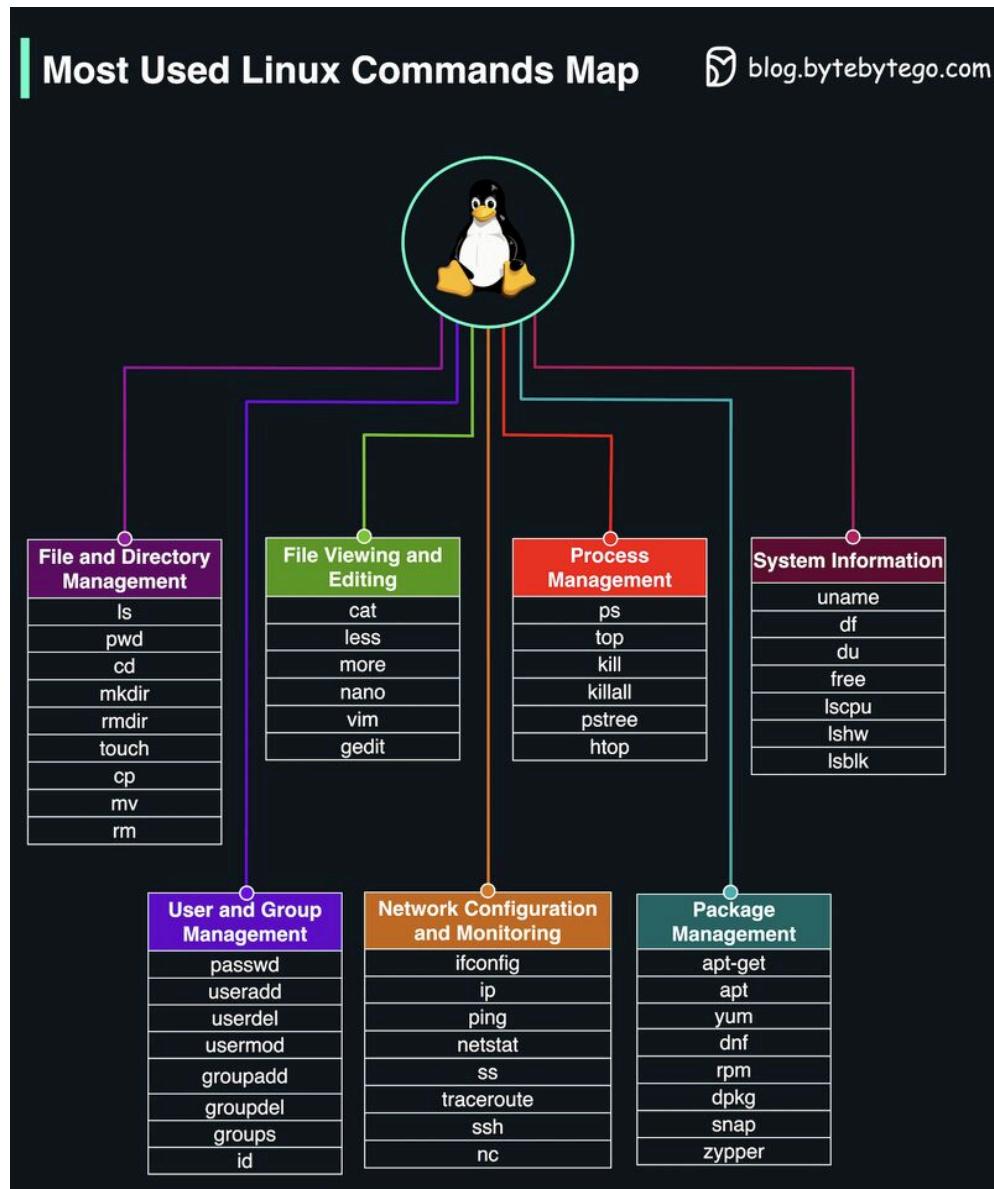
Explaining Sessions, Tokens, JWT, SSO, and OAuth in One Diagram



Understanding these backstage maneuvers helps us build secure, seamless experiences.

How do you see the evolution of web session management impacting the future of web applications and user experiences?

Most Used Linux Commands Map

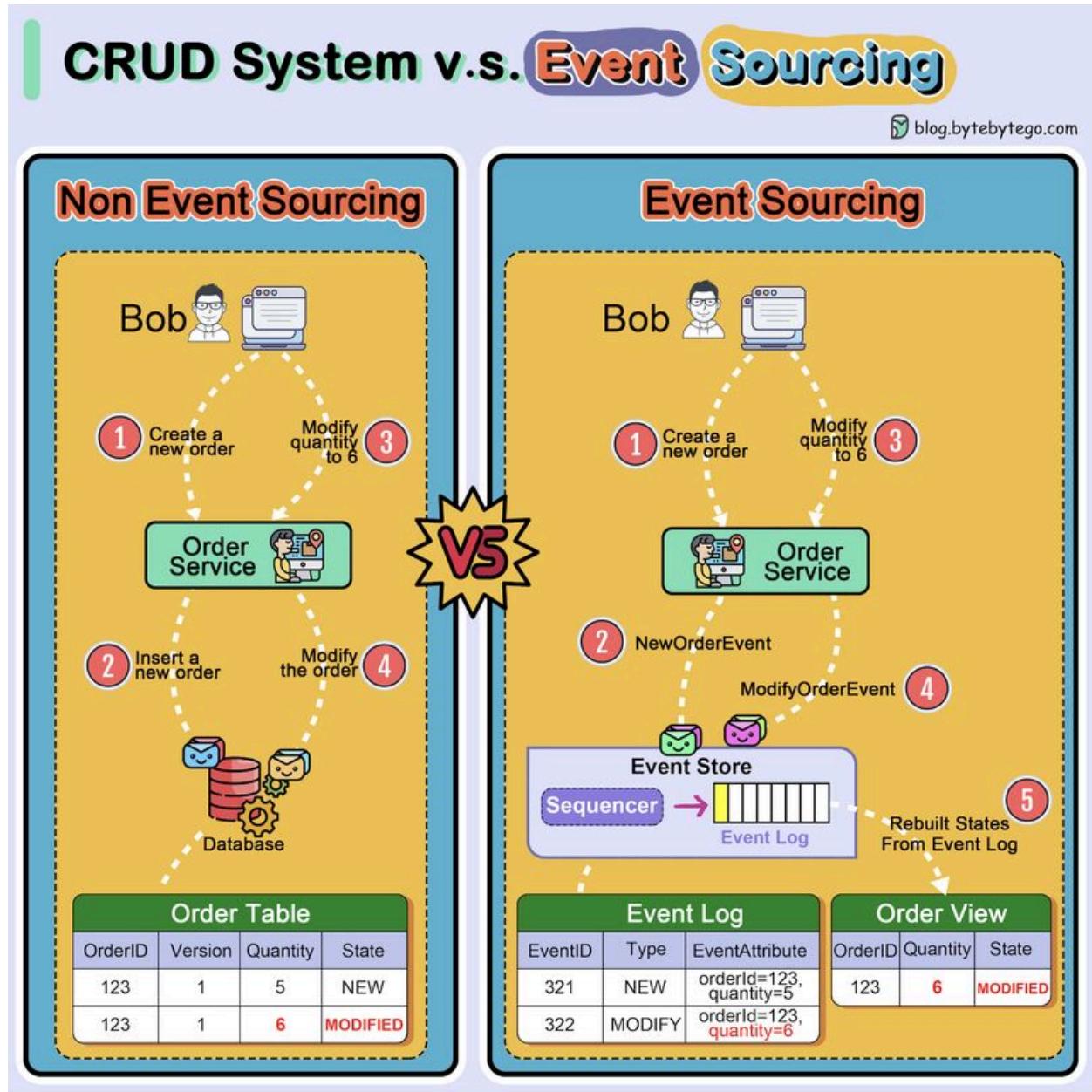


1. File and Directory Management
2. File Viewing and Editing
3. Process Management
4. System Information
5. User and Group Management
6. Network Configuration and Monitoring
7. Package Management

Over to you: Which command category did you use the most in your daily Linux tasks?

What is Event Sourcing? How is it different from normal CRUD design?

The diagram below shows a comparison of normal CRUD system design and event sourcing system design. We use an order service as an example.



The event sourcing paradigm is used to design a system with determinism. This changes the philosophy of normal system designs.

How does this work? Instead of recording the order states in the database, the event sourcing design persists the events that lead to the state changes in the event store. The event store is an append-only log. The events must be sequenced with incremental numbers to guarantee their

ordering. The order states can be rebuilt from the events and maintained in OrderView. If the OrderView is down, we can always rely on the event store which is the source of truth to recover the order states.

Let's look at the detailed steps.

- Non-Event Sourcing

Steps 1 and 2: Bob wants to buy a product. The order is created and inserted into the database.

Steps 3 and 4: Bob wants to change the quantity from 5 to 6. The order is modified with a new state.

- Event Sourcing

Steps 1 and 2: Bob wants to buy a product. A NewOrderEvent is created, sequenced, and stored in the event store with eventID=321.

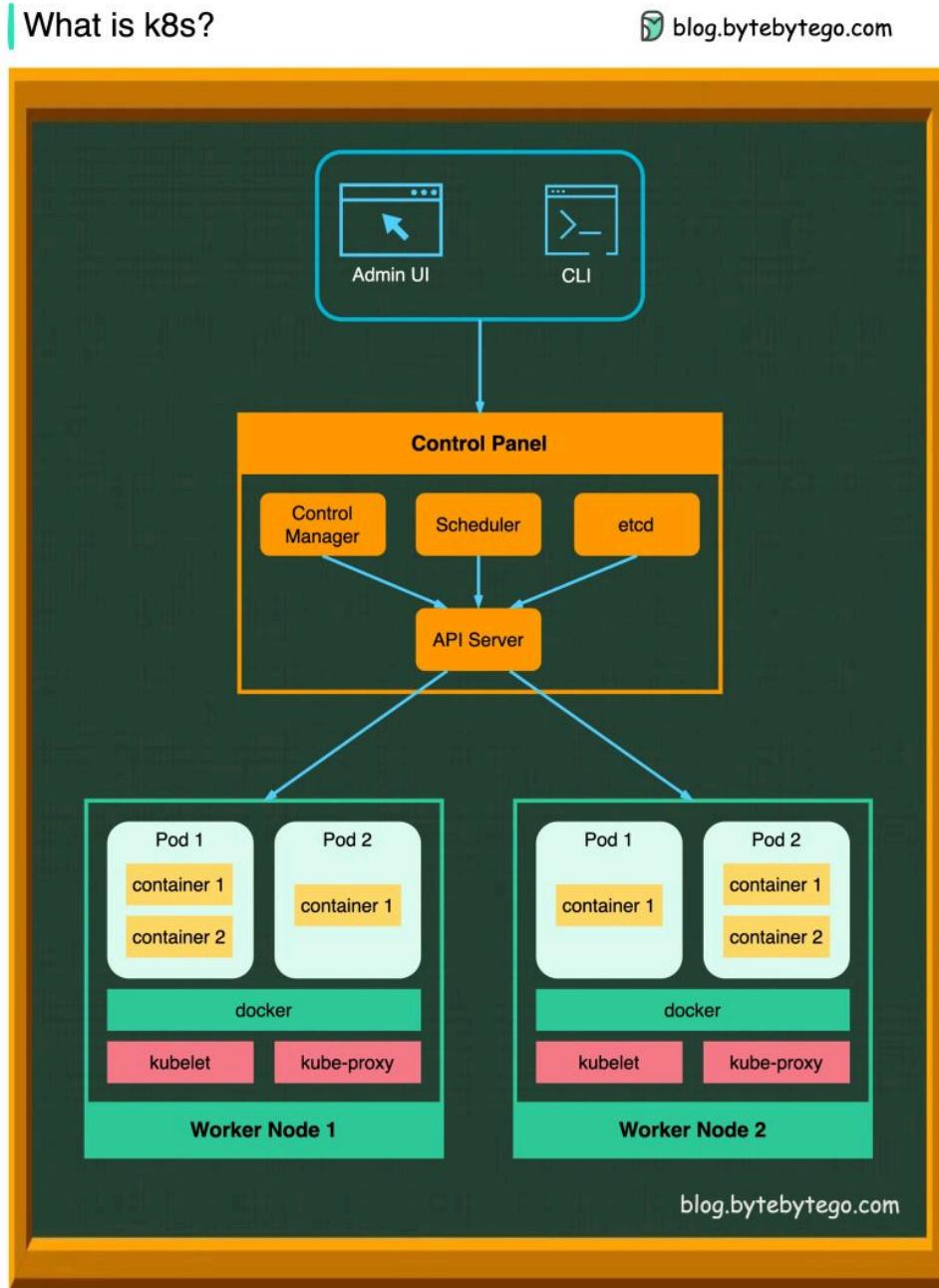
Steps 3 and 4: Bob wants to change the quantity from 5 to 6. A ModifyOrderEvent is created, sequenced, and persisted in the event store with eventID=322.

Step 5: The order view is rebuilt from the order events, showing the latest state of an order.

Over to you: Which type of system is suitable for event sourcing design? Have you used this paradigm in your work?

What is k8s (Kubernetes)?

k8s is a container orchestration system. It is used for container deployment and management. Its design is greatly impacted by Google's internal system Borg.



A k8s cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers, and a cluster usually runs multiple nodes, providing fault tolerance and high availability.

- Control Plane Components

1. API Server

The API server talks to all the components in the k8s cluster. All the operations on pods are executed by talking to the API server.

2. Scheduler

The scheduler watches pod workloads and assigns loads on newly created pods.

3. Controller Manager

The controller manager runs the controllers, including Node Controller, Job Controller, EndpointSlice Controller, and ServiceAccount Controller.

4. etcd

etcd is a key-value store used as Kubernetes' backing store for all cluster data.

- Nodes

1. Pods

A pod is a group of containers and is the smallest unit that k8s administers. Pods have a single IP address applied to every container within the pod.

2. Kubelet

An agent that runs on each node in the cluster. It ensures containers are running in a Pod.

3. Kube Proxy

kube-proxy is a network proxy that runs on each node in your cluster. It routes traffic coming into a node from the service. It forwards requests for work to the correct containers.

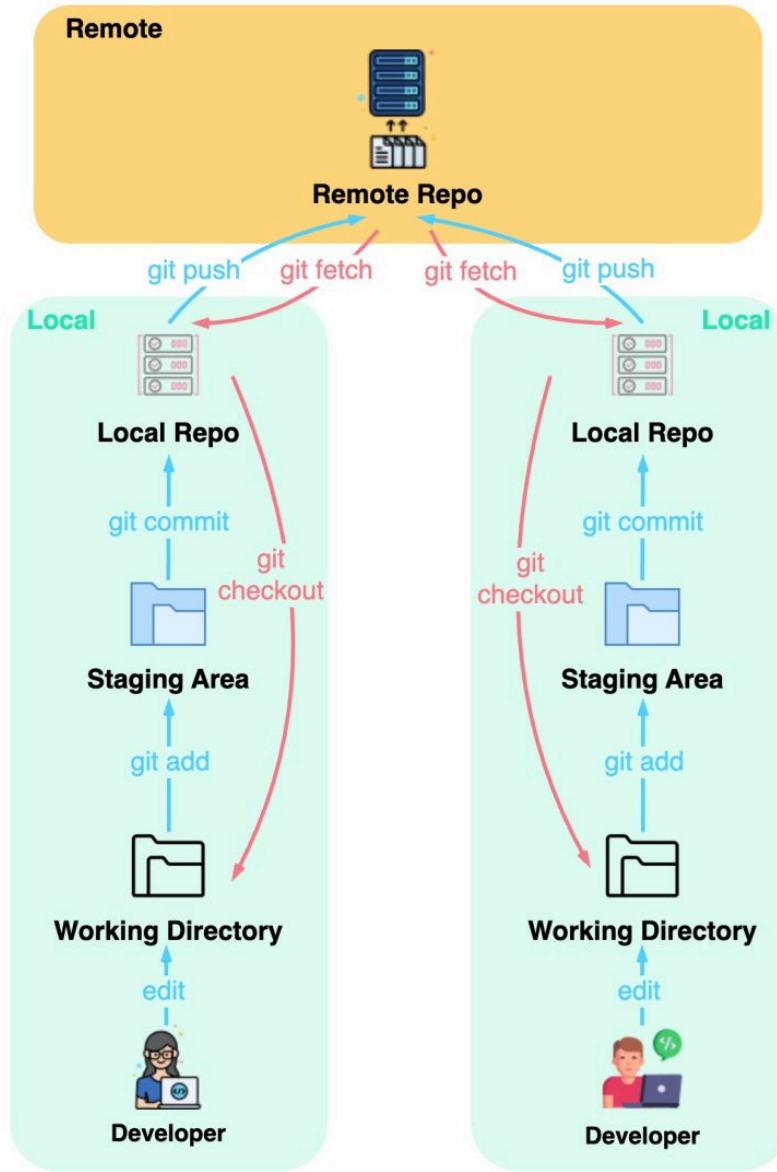
Over to you: Do you know why Kubernetes is called “k8s”? Despite its power, K8s can be intimidating. What do you think about it?

How does Git Work?

The diagram below shows the Git workflow.

How does Git Work?

 blog.bytebytego.com



Git is a distributed version control system.

Every developer maintains a local copy of the main repository and edits and commits to the local copy.

The commit is very fast because the operation doesn't interact with the remote repository.

If the remote repository crashes, the files can be recovered from the local repositories.

Over to you: Which Git command do you use to resolve conflicting changes?

How does Google Authenticator (or other types of 2-factor authenticators) work?

Google authenticator is commonly used for logging into our accounts when 2-factor authentication is enabled. How does it guarantee security?

Google Authenticator is a software-based authenticator that implements a two-step verification service. The diagram below provides detail.

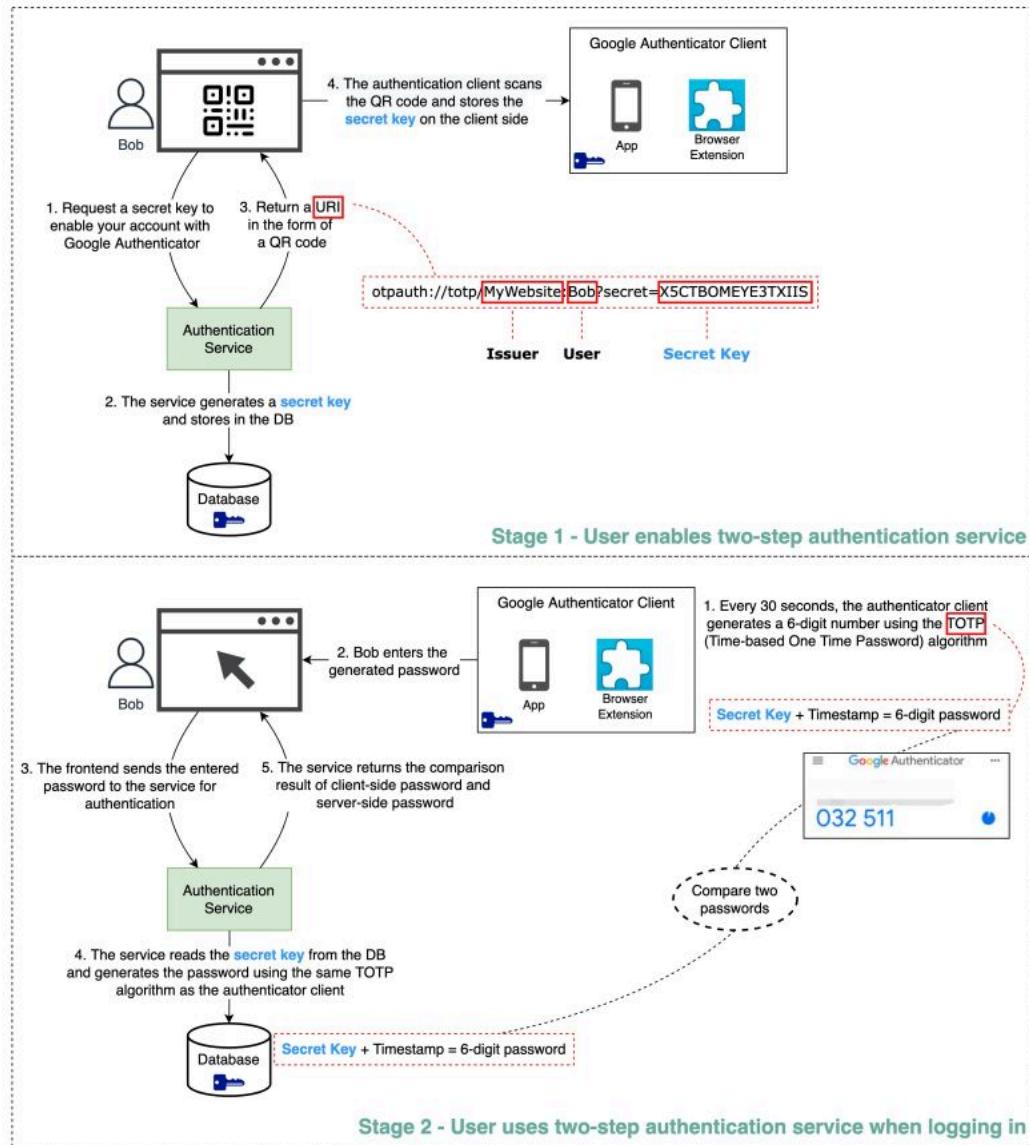
There are two stages involved:

- Stage 1 - The user enables Google two-step verification
- Stage 2 - The user uses the authenticator for logging in, etc.

Let's look at these stages.

How does Google Authenticator Work?

 blog.bytebytego.com



Stage 1

Steps 1 and 2: Bob opens the web page to enable two-step verification. The front end requests a secret key. The authentication service generates the secret key for Bob and stores it in the database.

Step 3: The authentication service returns a URI to the front end. The URI is composed of a key issuer, username, and secret key. The URI is displayed in the form of a QR code on the web page.

Step 4: Bob then uses Google Authenticator to scan the generated QR code. The secret key is stored in the authenticator.

Stage 2

Steps 1 and 2: Bob wants to log into a website with Google two-step verification. For this, he needs the password. Every 30 seconds, Google Authenticator generates a 6-digit password using TOTP (Time-based One Time Password) algorithm. Bob uses the password to enter the website.

Steps 3 and 4: The front end sends Bob's password to the backend for authentication. The authentication service reads the secret key from the database and generates a 6-digit password using the same TOTP algorithm as the client.

Step 5: The authentication service compares the two passwords generated by the client and the server, and returns the comparison result to the front. Bob can proceed with the login process only if the two passwords match.

Is this authentication mechanism **safe**?

- Can the secret key be obtained by others?

We need to make sure the secret key is transmitted using HTTPS. The authenticator client and the database store the secret key, and we need to ensure the secret keys are encrypted.

- Can the 6-digit password be guessed by hackers?
- No. The password has 6 digits, so the generated password has 1 million potential combinations. Plus, the password changes every 30 seconds. If hackers want to guess the password in 30 seconds, they need to enter 30,000 combinations per second.

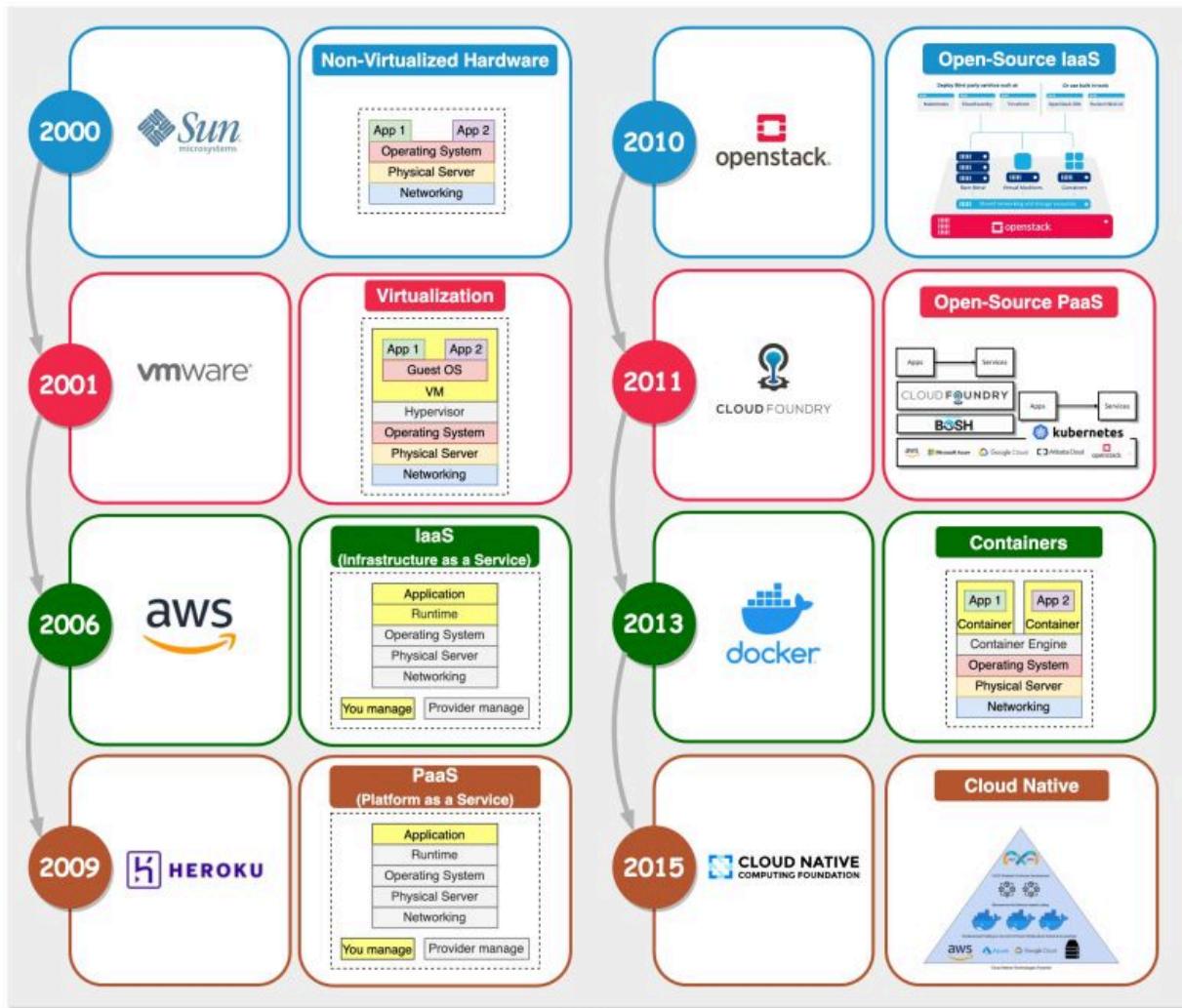
Over to you: What are some of the other 2-factor authentication devices you used?

IaaS, PaaS, Cloud Native... How do we get here?

The diagram below shows two decades of cloud evolution.

2 Decades of Cloud Evolution

 blog.bytebytogo.com



Sources:
<https://www.openstack.org/>
<https://blogs.sap.com/2018/10/19/cloud-foundry-and-kubernetes-where-do-they-differ-how-do-they-fit-together/>
<https://www.influxdata.com/blog/introduction-cloud-native/>

2001 - VMWare - Virtualization via hypervisor

2006 - AWS - IaaS (Infrastructure as a Service)

2009 - Heroku - PaaS (Platform as a Service)

2010 - OpenStack - Open-source IaaS

2011 - CloudFoundry - Open-source PaaS

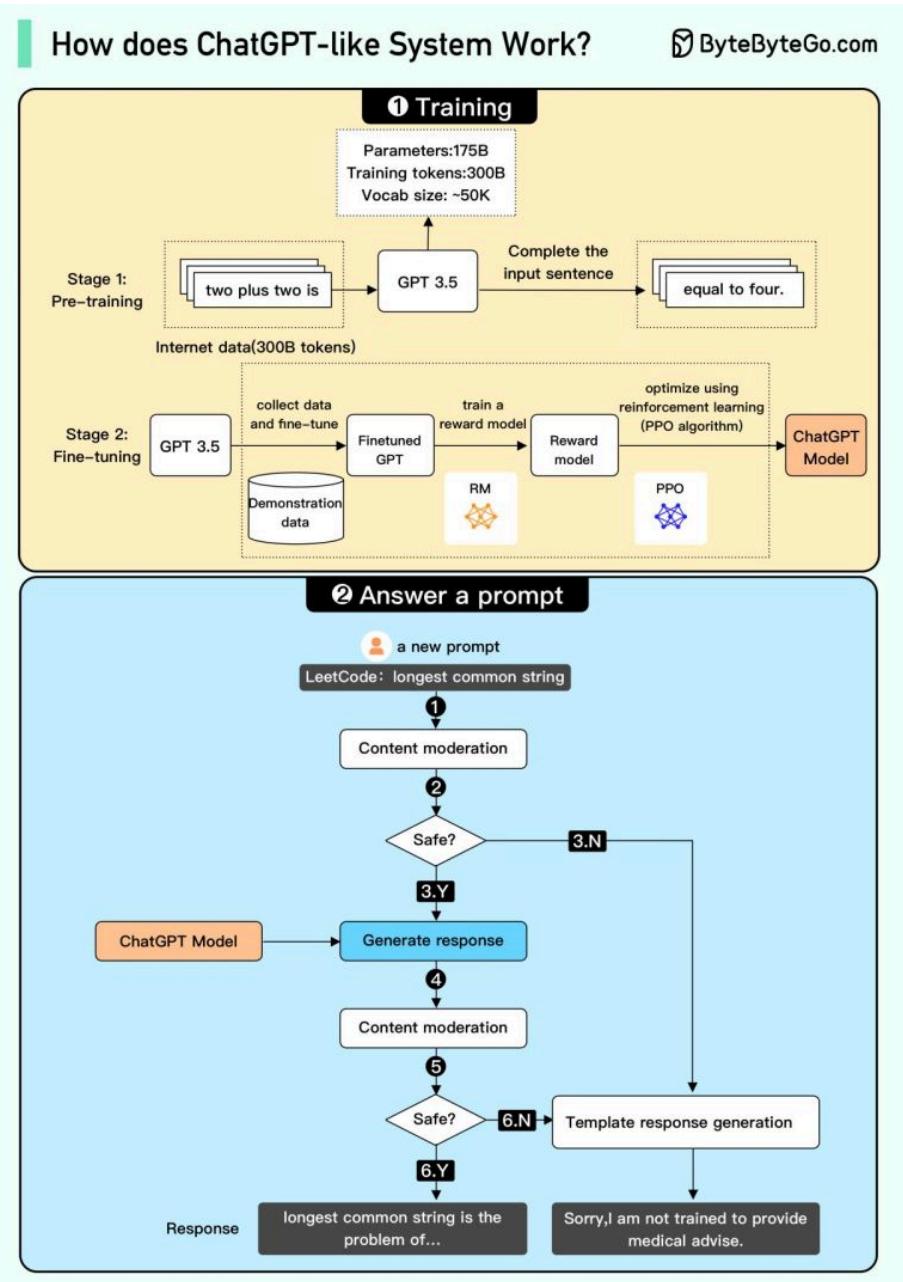
2013 - Docker - Containers

2015 - CNCF (Cloud Native Computing Foundation) - Cloud Native

Over to you: Which ones have you used?

How does ChatGPT work?

Since OpenAI hasn't provided all the details, some parts of the diagram may be inaccurate.



1. Training. To train a ChatGPT model, there are two stages:

- Pre-training: In this stage, we train a GPT model (decoder-only transformer) on a large chunk of internet data. The objective is to train a model that can predict future words given a sentence in a way that is grammatically correct and semantically meaningful

similar to the internet data. After the pre-training stage, the model can complete given sentences, but it is not capable of responding to questions.

- Fine-tuning: This stage is a 3-step process that turns the pre-trained model into a question-answering ChatGPT model:
 - Collect training data (questions and answers), and fine-tune the pre-trained model on this data. The model takes a question as input and learns to generate an answer similar to the training data.
 - Collect more data (question, several answers) and train a reward model to rank these answers from most relevant to least relevant.
 - Use reinforcement learning (PPO optimization) to fine-tune the model so the model's answers are more accurate.

2. Answer a prompt

- Step 1: The user enters the full question, “Explain how a classification algorithm works”.
- Step 2: The question is sent to a content moderation component. This component ensures that the question does not violate safety guidelines and filters inappropriate questions.
- Steps 3-4: If the input passes content moderation, it is sent to the chatGPT model. If the input doesn't pass content moderation, it goes straight to template response generation.
- Step 5-6: Once the model generates the response, it is sent to a content moderation component again. This ensures the generated response is safe, harmless, unbiased, etc.
- Step 7: If the input passes content moderation, it is shown to the user. If the input doesn't pass content moderation, it goes to template response generation and shows a template answer to the user.

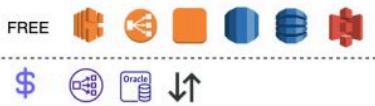
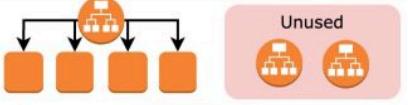
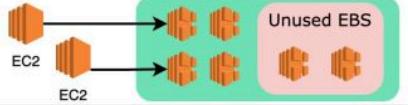
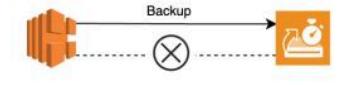
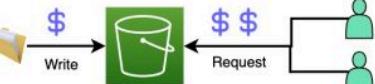
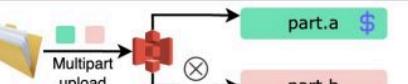
Top Hidden Costs of Cloud Providers

Is the cloud really free or inexpensive?

While it may be inexpensive or even free to get started, the complexity often leads to hidden costs, resulting in large cloud bills.

The purpose of this post is not to discourage using the cloud. I'm a big fan of the cloud. I simply want to raise awareness about this issue, as it's one of the critical topics that isn't often discussed.

While AWS is used as an example, similar cost structures apply to other cloud providers.

Cost Type	Illustration	Hidden costs
Free Tier Ambiguity		AWS Free Tier has limits on many resources. Exceeding those limits can be costly.
Under-utilized Elastic IP (EIP) addresses		Free for 5. Any extra charged at hourly rate.
Unused Elastic Load Balancers		An unused Load Balancer is still charged at an hourly rate.
Unused Elastic Block Storage (EBS)		Unused EBS are charged at GB-month rate.
Orphan Elastic Block Storage Snapshots		Delete an EBS volume will NOT delete backups, create orphan backups cost.
S3 Access Charges		Get, List, and Retrieval request could be much more costly than file storage cost.
S3 Partial Uploads		Incomplete multipart uploads still incur charges
Transfer cost		Transfer to AWS is FREE, but transfer out can be costly.

1. Free Tier Ambiguity: AWS offers three different types of free offerings for common services. However, services not included in the free tier can charge you. Even for services that do provide free resources, there's often a limit. Exceeding that limit can result in higher costs than anticipated.
2. Elastic IP Addresses: AWS allows up to five Elastic IP addresses. Exceeding this limit incurs a small hourly rate, which varies depending on the region. This is a recurring charge.
3. Load Balancers: They are billed hourly, even if not actively used. Furthermore, you'll face additional charges if data is transferred in and out of the load balancer.
4. Elastic Block Storage (EBS) Charges: EBS is billed on a GB-per-month basis. You will be charged for attached and unattached EBS volumes, even if they're not actively used.
5. EBS Snapshots: Deleting an EBS volume does not automatically remove the associated snapshots. Orphaned EBS snapshots will still appear on your bill.
6. S3 Access Charges: While the pricing for S3 storage is generally reasonable, the costs associated with accessing stored objects, such as GET and LIST requests, can sometimes exceed the storage costs.
7. S3 Partial Uploads: If you have an unsuccessful multipart upload in S3, you will still be billed for the successfully uploaded parts. It's essential to clean these up to avoid unnecessary costs.
8. Data Transfer Costs: Transferring data to AWS, for instance, from a data center, is free. However, transferring data out of AWS can be significantly more expensive.

Over to you: Have you ever been surprised by an unexpected cloud bill? Share your experiences with us!

Algorithms You Should Know Before You Take System Design Interviews

These algorithms aren't just useful for acing system design interviews - they're also great tools for building real-world systems.

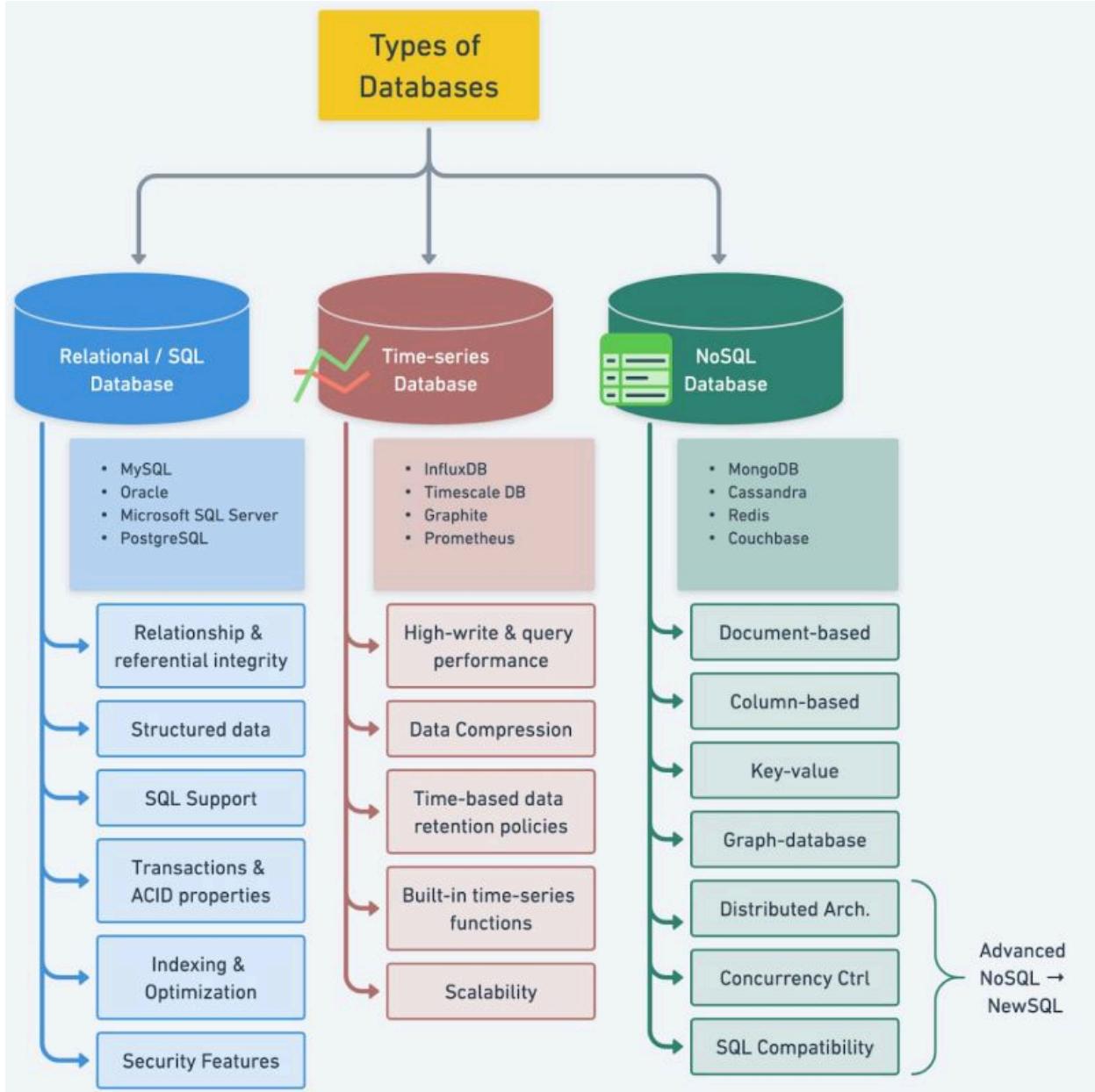
Algorithms you should know before system design interviews		ByteByteGo.com	
Algorithm	How it Works	Priority	Use Cases
Geohash		★★★★★	Location based service
Quadtree		★★★★★	Location based service
Consistent Hashing		★★★★★	Balance the load within a cluster of services
Leaky bucket		★★★★★	Rate limiter
Token bucket		★★★★★	Rate limiter
Trie		★★★★★	Search autocomplete
Rsync		★★★☆☆	File transfers
Raft/Paxos		★★★☆☆	Consensus algorithms
Bloomfilter		★★★☆☆	Eliminate costly lookups
Merkle tree		★★★☆☆	Identify inconsistencies between nodes
HyperLogLog		★☆☆☆☆	Count unique values fast
Count-min sketch		★☆☆☆☆	Estimate frequencies of items
Hierarchical timing wheels		★☆☆☆☆	Job scheduler
Operational transformation		★☆☆☆☆	Collaborative editing

We made a video on this topic. The video contains an updated list and provides real-world case studies.

Watch here: <https://lnkd.in/ecMErZkg>

Understanding Database Types

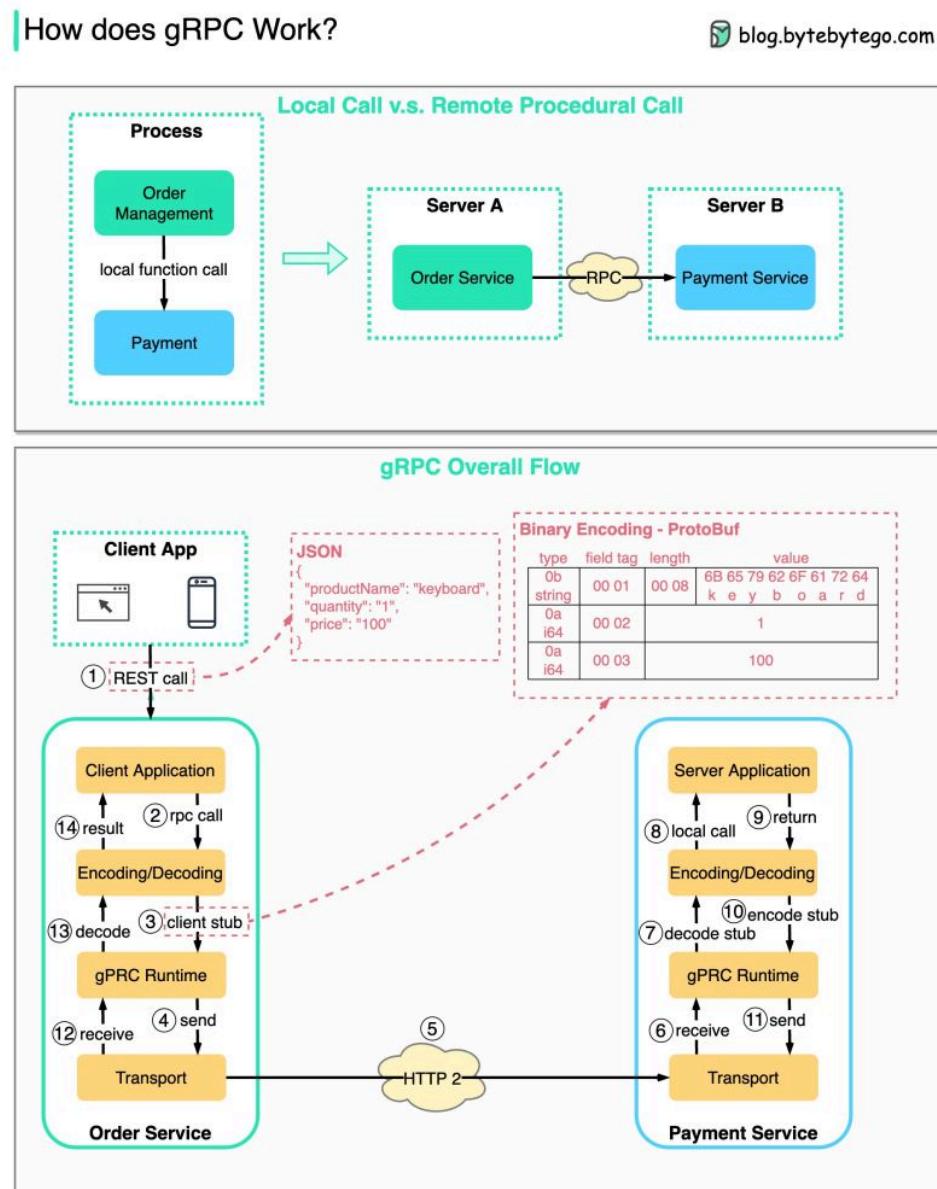
To make the best decision for our projects, it is essential to understand the various types of databases available in the market. We need to consider key characteristics of different database types, including popular options for each, and compare their use cases.



How does gRPC work?

RPC (Remote Procedure Call) is called “remote” because it enables communications between remote services when services are deployed to different servers under microservice architecture. From the user’s point of view, it acts like a local function call.

The diagram below illustrates the overall data flow for gRPC.



Step 1: A REST call is made from the client. The request body is usually in JSON format.

Steps 2 - 4: The order service (gRPC client) receives the REST call, transforms it, and makes an RPC call to the payment service. gPRC encodes the **client stub** into a binary format and sends it to the low-level transport layer.

Step 5: gRPC sends the packets over the network via HTTP2. Because of binary encoding and network optimizations, gRPC is said to be 5X faster than JSON.

Steps 6 - 8: The payment service (gRPC server) receives the packets from the network, decodes them, and invokes the server application.

Steps 9 - 11: The result is returned from the server application, and gets encoded and sent to the transport layer.

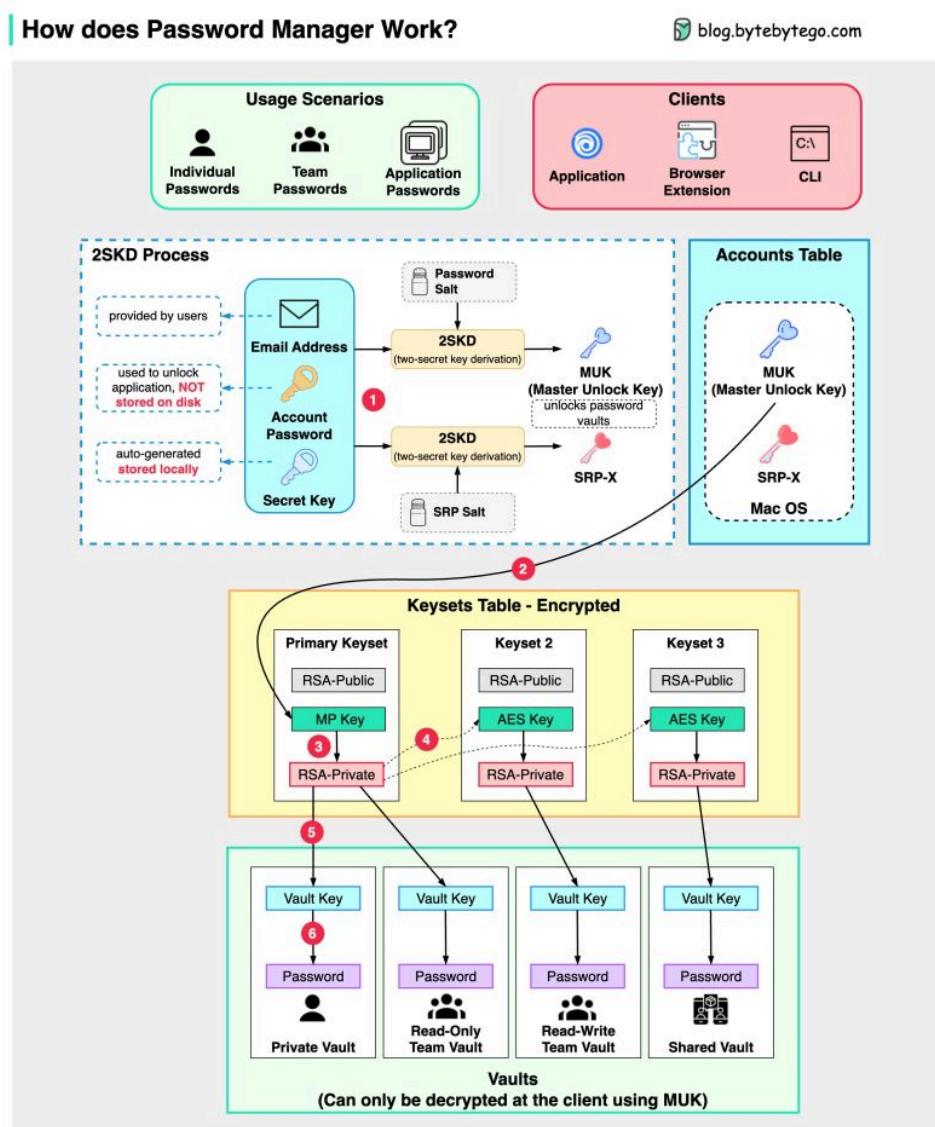
Steps 12 - 14: The order service receives the packets, decodes them, and sends the result to the client application.

Over to you: Have you used gPRC in your project? What are some of its limitations?

How does a Password Manager such as 1Password or Lastpass work?

How does it keep our passwords safe?

The diagram below shows how a typical password manager works.



A password manager generates and stores passwords for us. We can use it via application, browser extension, or command line.

Not only does a password manager store passwords for individuals but also it supports password management for teams in small businesses and big enterprises.

Let's go through the steps.

Step 1: When we sign up for a password manager, we enter our email address and set up an account password. The password manager generates a secret key for us. The 3 fields are used to generate MUK (Master Unlock Key) and SRP-X using the 2SKD algorithm. MUK is used to decrypt vaults that store our passwords. Note that the secret key is stored locally, and will not be sent to the password manager's server side.

Step 2: The MUK generated in Step 1 is used to generate the encrypted MP key of the primary keyset.

Steps 3-5: The MP key is then used to generate a private key, which can be used to generate AES keys in other keysets. The private key is also used to generate the vault key. Vault stores a collection of items for us on the server side. The items can be passwords notes etc.

Step 6: The vault key is used to encrypt the items in the vault.

Because of the complex process, the password manager has no way to know the encrypted passwords. We only need to remember one account password, and the password manager will remember the rest.

Over to you: Which password manager have you used?

Types of Software Engineers and Their Typically Required Skills

Types of Software Engineers

 blog.bytebytego.com

Front End Engineers	Back End Engineers	Full Stack Engineers <small>MOST LEARNING</small>
Designs and codes user interfaces for applications	Builds and maintains server-side logic for applications	Everything Front End & Back End Engineers do
<p>Skills Include</p> <ul style="list-style-type: none">✓  Version Control✓  HTTPS Understanding✓  APIs Understanding✓  Programming Language✓  HTML✓  CSS✓  JavaScript✓ Front-end Frameworks<ul style="list-style-type: none"> React Vue Angular(so on ...)	<p>Skills Include</p> <ul style="list-style-type: none">✓  Version Control✓  HTTPS Understanding✓  APIs Understanding✓  Programming Language✓  Database Proficiency✓  Caching Strategies✓  Server Management and Deployment	<p>Skills Include</p> <ul style="list-style-type: none">✓  Version Control✓  HTTPS Understanding✓  APIs Understanding✓  Programming Language✓  HTML✓  CSS✓  JavaScript✓ Front-end Frameworks<ul style="list-style-type: none"> React Vue Angular(so on ...)✓  Database Proficiency✓  Caching Strategies✓  Server Management and Deployment

In this overview, we'll explore three key types of Software engineers:

1. Front-End Engineer:

Specializes in creating user interfaces using HTML, CSS, and JavaScript. They focus on ensuring that apps are visually appealing and user-friendly.

2. Back-End Engineer:

Works on the server-side of web applications, managing data, business logic, and server infrastructure to ensure functionality, performance, and security.

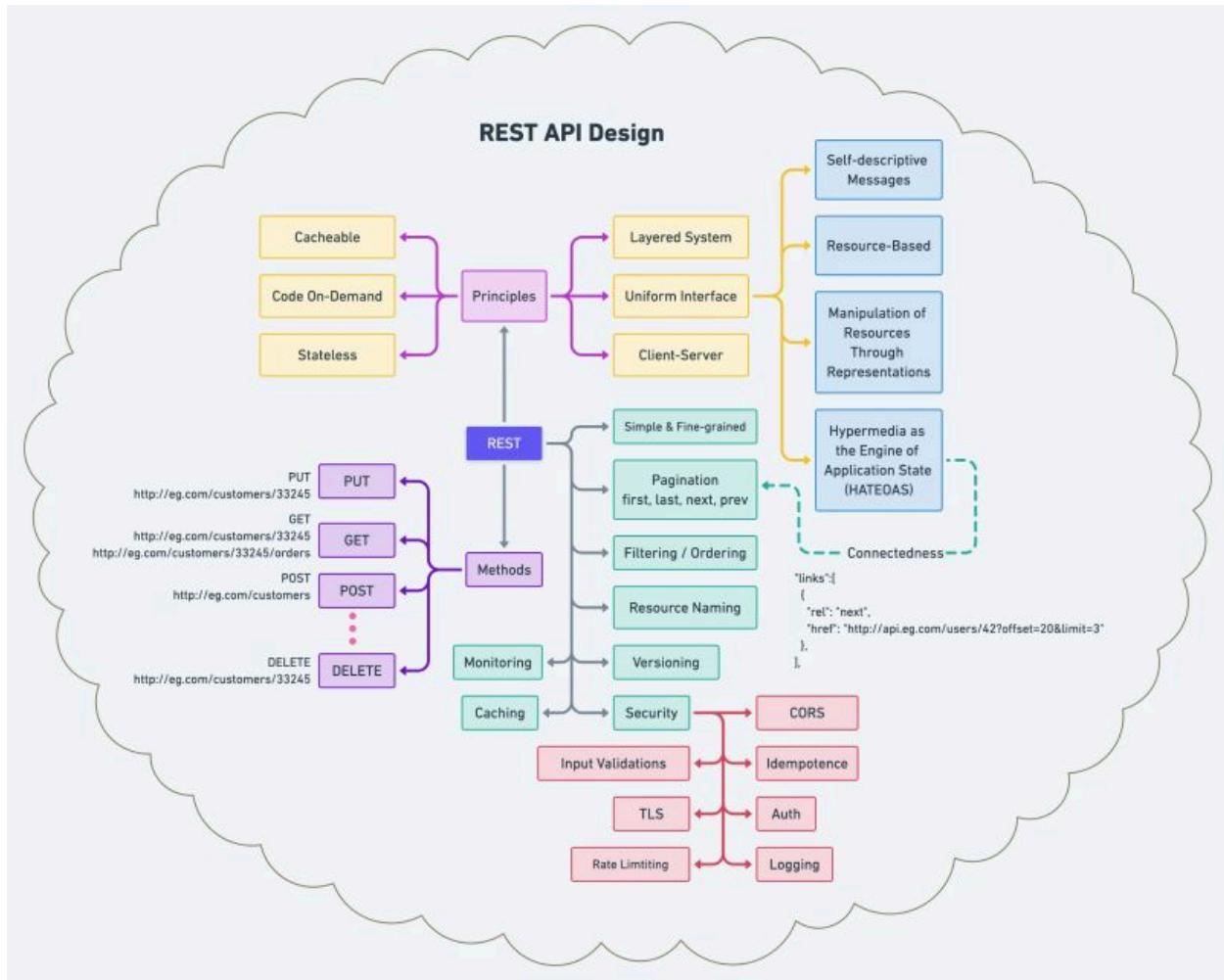
3. Full-Stack Engineer:

A versatile expert who combines the roles of Front-End and Back-End engineers, handling UI design, server-side tasks, databases, APIs, and ensuring seamless application integration. They cover the entire development spectrum from start to finish.

Over to you: Which type of software engineer resonates most with your interests and career aspirations?

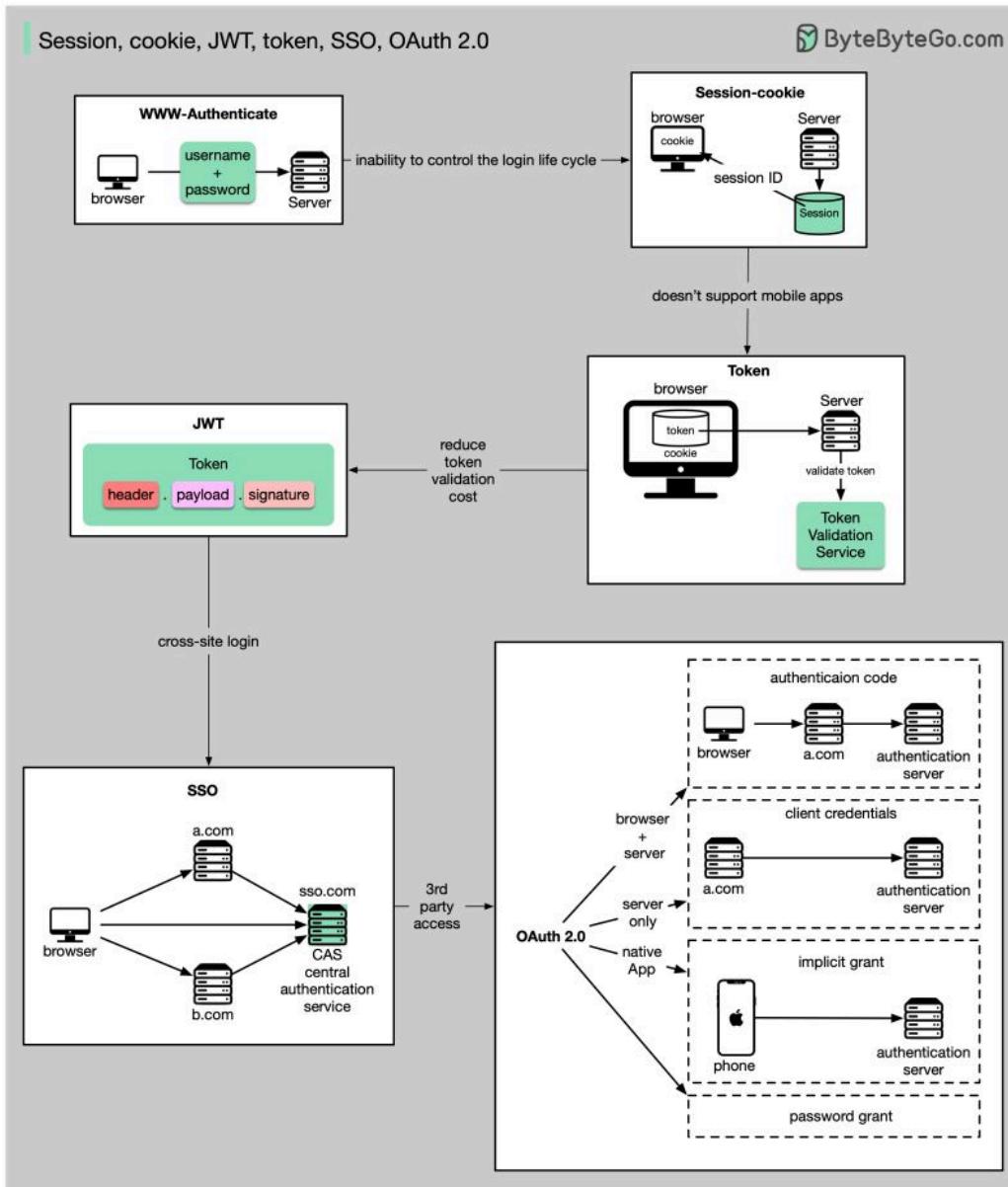
How does REST API work?

What are its principles, methods, constraints, and best practices? We hope the diagram below gives you a quick overview.



Session, cookie, JWT, token, SSO, and OAuth 2.0 - what are they?

These terms are all related to user identity management. When you log into a website, you declare who you are (identification). Your identity is verified (authentication), and you are granted the necessary permissions (authorization). Many solutions have been proposed in the past, and the list keeps growing.



From simple to complex, here is my understanding of user identity management:

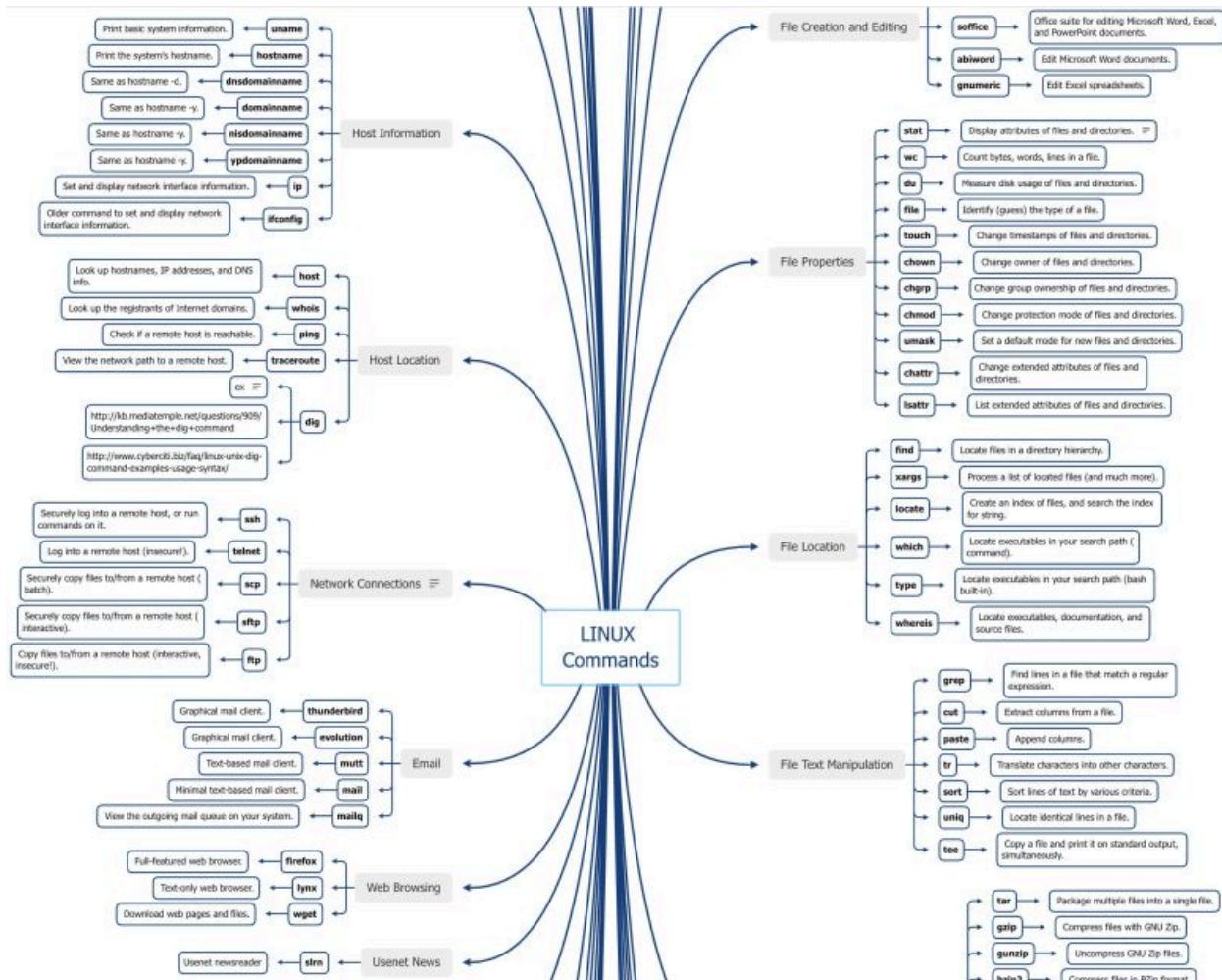
- WWW-Authenticate is the most basic method. You are asked for the username and password by the browser. As a result of the inability to control the login life cycle, it is seldom used today.
- A finer control over the login life cycle is session-cookie. The server maintains session storage, and the browser keeps the ID of the session. A cookie usually only works with browsers and is not mobile app friendly.
- To address the compatibility issue, the token can be used. The client sends the token to the server, and the server validates the token. The downside is that the token needs to be encrypted and decrypted, which may be time-consuming.
- JWT is a standard way of representing tokens. This information can be verified and trusted because it is digitally signed. Since JWT contains the signature, there is no need to save session information on the server side.
- By using SSO (single sign-on), you can sign on only once and log in to multiple websites. It uses CAS (central authentication service) to maintain cross-site information
- By using OAuth 2.0, you can authorize one website to access your information on another website

Over to you:

Nowadays, some websites allow you to log in by scanning the QR code using your phone. Do you know how it works?

Linux commands illustrated on one page!

Take a look at how many you know :)

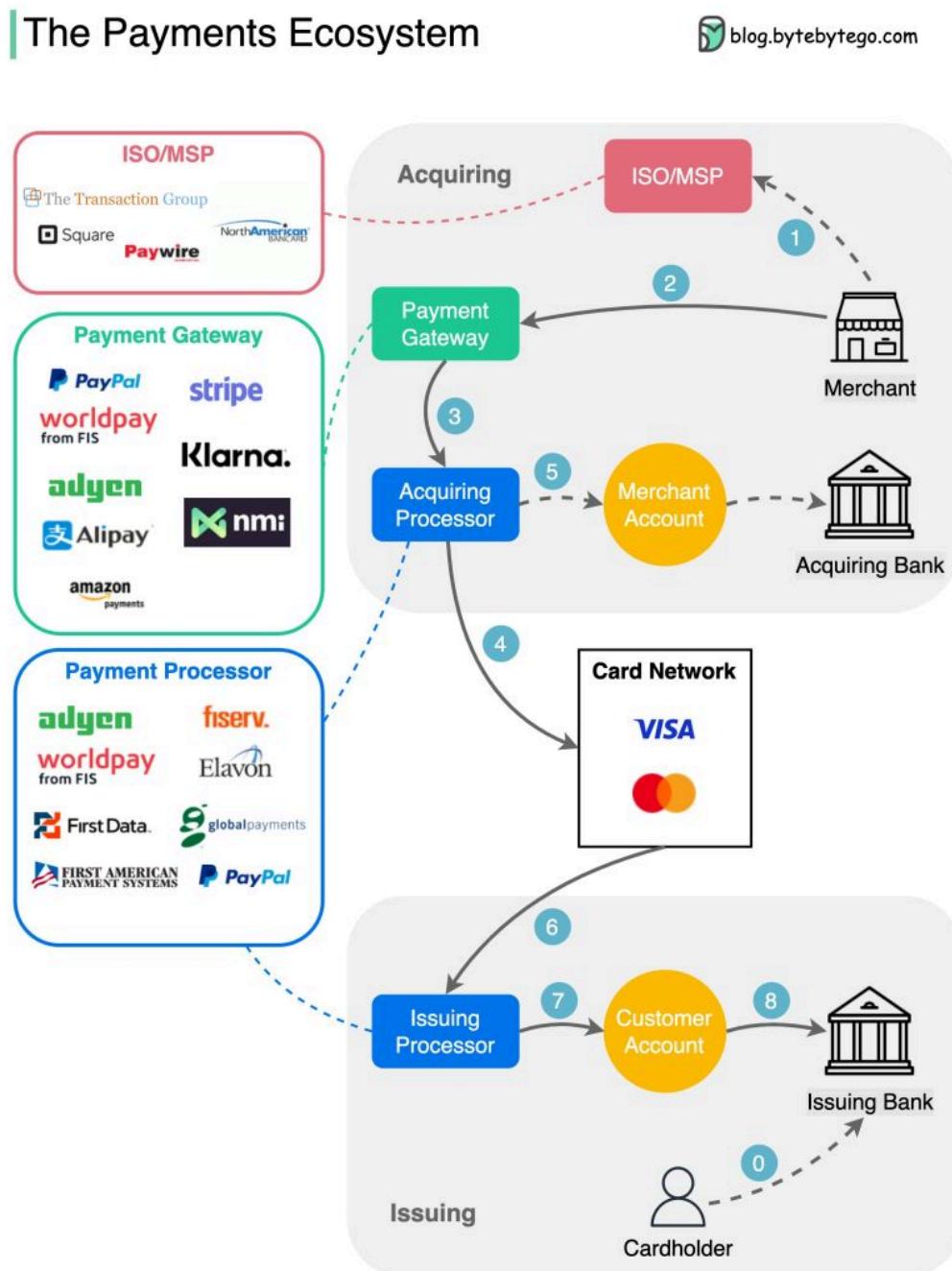


- Controlling processes: kill, killall, nice
- Scheduling jobs: sleep, watch, crontab
- Host location: host, whois, ping, traceroute
- Network connections: ssh, telnet, scp, ftp
- Screen output: echo, printf, seq, clear
- Viewing Processes: ps, uptime, top, free
- And many more

Linux commands: <https://xmind.app/m/WwtB/>

The Payments Ecosystem

How do fintech startups find new opportunities among so many payment companies? What do PayPal, Stripe, and Square do exactly?



Steps 0-1: The cardholder opens an account in the issuing bank and gets the debit/credit card. The merchant registers with ISO (Independent Sales Organization) or MSP (Member Service)

Provider) for in-store sales. ISO/MSP partners with payment processors to open merchant accounts.

Steps 2-5: The acquiring process.

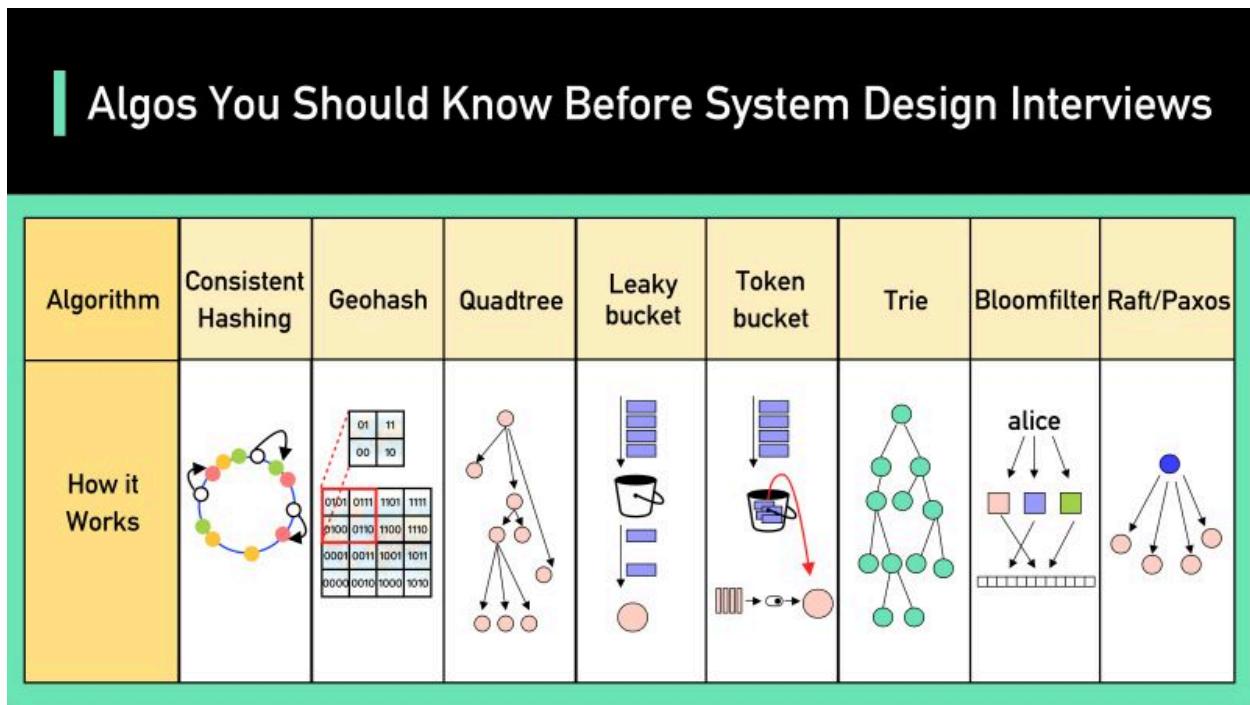
The payment gateway accepts the purchase transaction and collects payment information. It is then sent to a payment processor, which uses customer information to collect payments. The acquiring processor sends the transaction to the card network. It also owns and operates the merchant's account during settlement, which doesn't happen in real-time.

Steps 6-8: The issuing process.

The issuing processor talks to the card network on the issuing bank's behalf. It validates and operates the customer's account.

I've listed some companies in different verticals in the diagram. Notice payment companies usually start from one vertical, but later expand to multiple verticals.

Algorithms You Should Know Before You Take System Design Interviews (updated list)

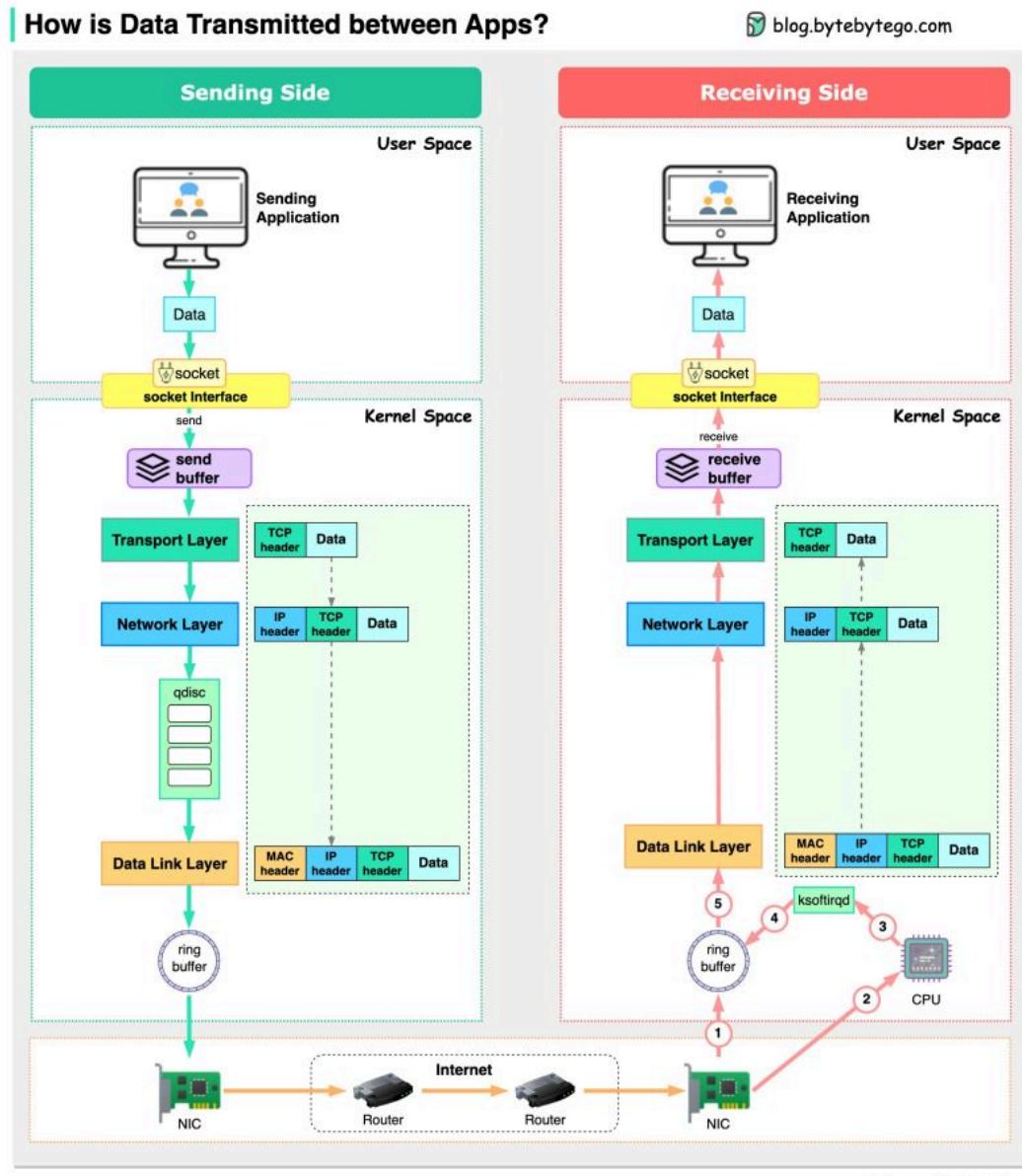


- Consistent hashing
- Spatial Indexing
- Rate Limiting
- Tries
- Bloom Filters
- Consensus Algorithms

Watch the whole video here: <https://lnkd.in/eMYFDjVU>

How is data transmitted between applications?

The diagram below shows how a server sends data to another server.



Assume a chat application running in the user space sends out a chat message. The message is sent to the send buffer in the kernel space. The data then goes through the network stack and is wrapped with a TCP header, an IP header, and a MAC header. The data also goes through qdisc (Queueing Disciplines) for flow control. Then the data is sent to the NIC (Network Interface Card) via a ring buffer.

The data is sent to the internet via NIC. After many hops among routers and switches, the data arrives at the NIC of the receiving server.

The NIC of the receiving server puts the data in the ring buffer and sends a hard interrupt to the CPU. The CPU sends a soft interrupt so that ksoftirqd receives data from the ring buffer. Then the data is unwrapped through the data link layer, network layer and transport layer. Eventually, the data (chat message) is copied to the user space and reaches the chat application on the receiving side.

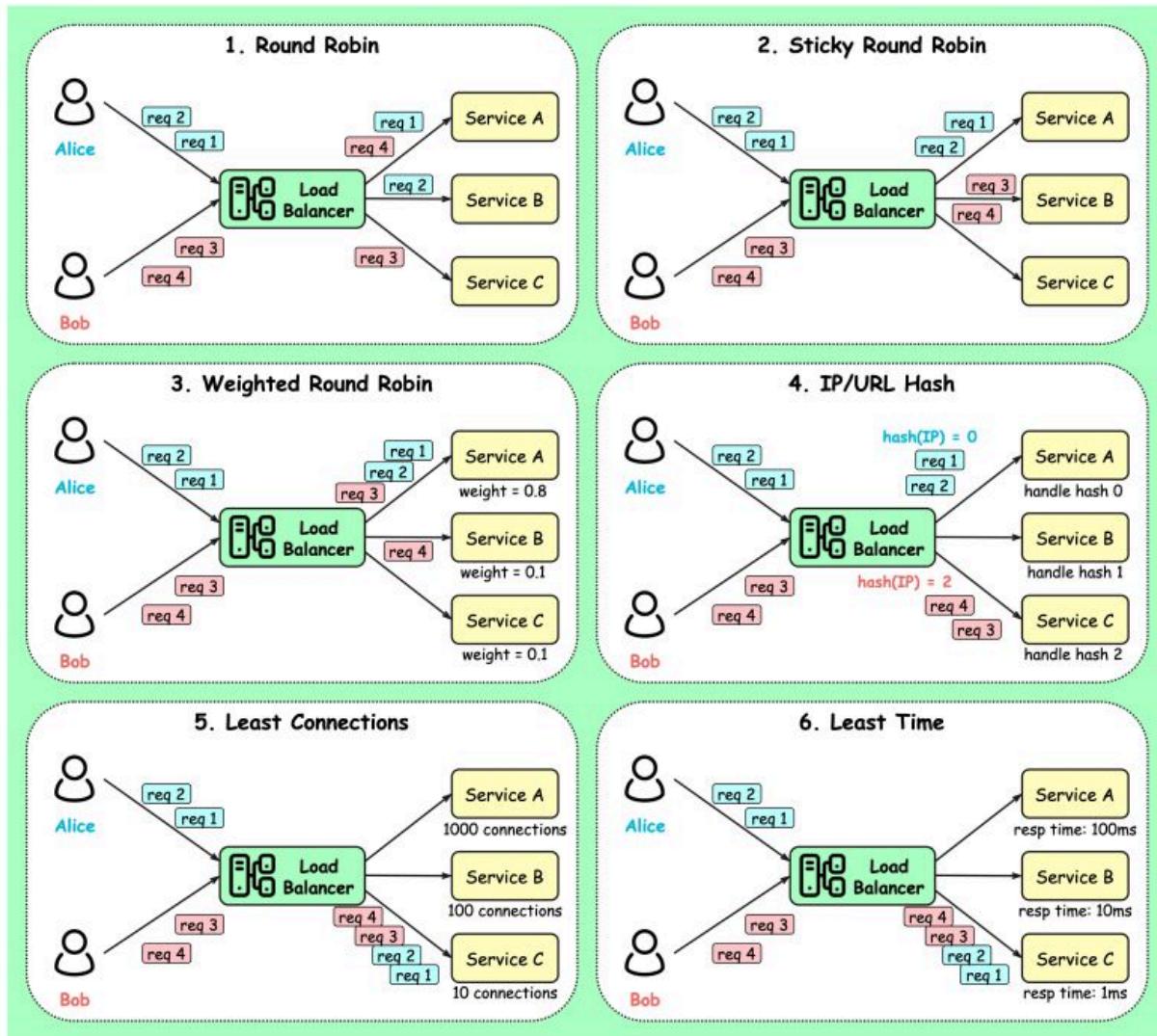
Over to you: What happens when the ring buffer is full? Will it lose packets?

What are the common load-balancing algorithms?

The diagram below shows 6 common algorithms.

Load Balancing Algorithms

 blog.bytebytogo.com



- Static Algorithms

1. Round robin

The client requests are sent to different service instances in sequential order. The services are usually required to be stateless.

2. Sticky round-robin

This is an improvement of the round-robin algorithm. If Alice's first request goes to service A, the following requests go to service A as well.

3. Weighted round-robin

The admin can specify the weight for each service. The ones with a higher weight handle more requests than others.

4. Hash

This algorithm applies a hash function on the incoming requests' IP or URL. The requests are routed to relevant instances based on the hash function result.

- Dynamic Algorithms

5. Least connections

A new request is sent to the service instance with the least concurrent connections.

6. Least response time

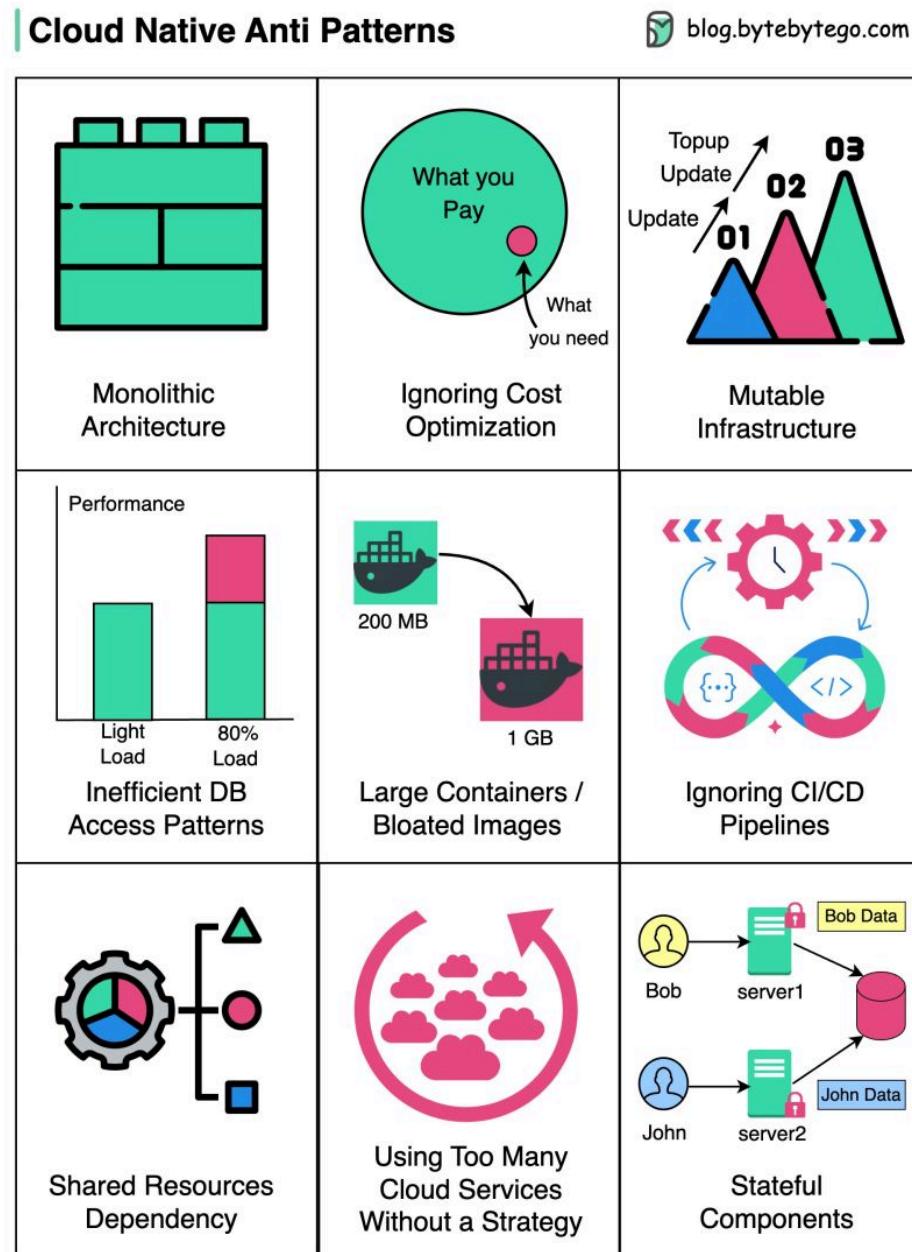
A new request is sent to the service instance with the fastest response time.

Over to you:

1. Which algorithm is most popular?
2. We can use other attributes for hashing algorithms. For example, HTTP header, request type, client type, etc. What attributes have you used?

Cloud Native Anti Patterns

By being aware of these anti-patterns and following cloud-native best practices, you can design, build, and operate more robust, scalable, and cost-efficient cloud-native applications.



1. Monolithic Architecture:

One large, tightly coupled application running on the cloud, hindering scalability and agility

2. Ignoring Cost Optimization:

Cloud services can be expensive, and not optimizing costs can result in budget overruns

3. Mutable Infrastructure:

- Infrastructure components are to be treated as disposable and are never modified in place
- Failing to embrace this approach can lead to configuration drift, increased maintenance, and decreased reliability

4. Inefficient DB Access Patterns:

Use of overly complex queries or lacking database indexing, can lead to performance degradation and database bottlenecks

5. Large Containers or Bloated Images:

Creating large containers or using bloated images can increase deployment times, consume more resources, and slow down application scaling

6. Ignoring CI/CD Pipelines:

Deployments become manual and error-prone, impeding the speed and frequency of software releases

7. Shared Resources Dependency:

Applications relying on shared resources like databases can create contention and bottlenecks, affecting overall performance

8. Using Too Many Cloud Services Without a Strategy:

While cloud providers offer a vast array of services, using too many of them without a clear strategy can create complexity and make it harder to manage the application.

9. Stateful Components:

Relying on persistent state in applications can introduce complexity, hinder scalability, and limit fault tolerance

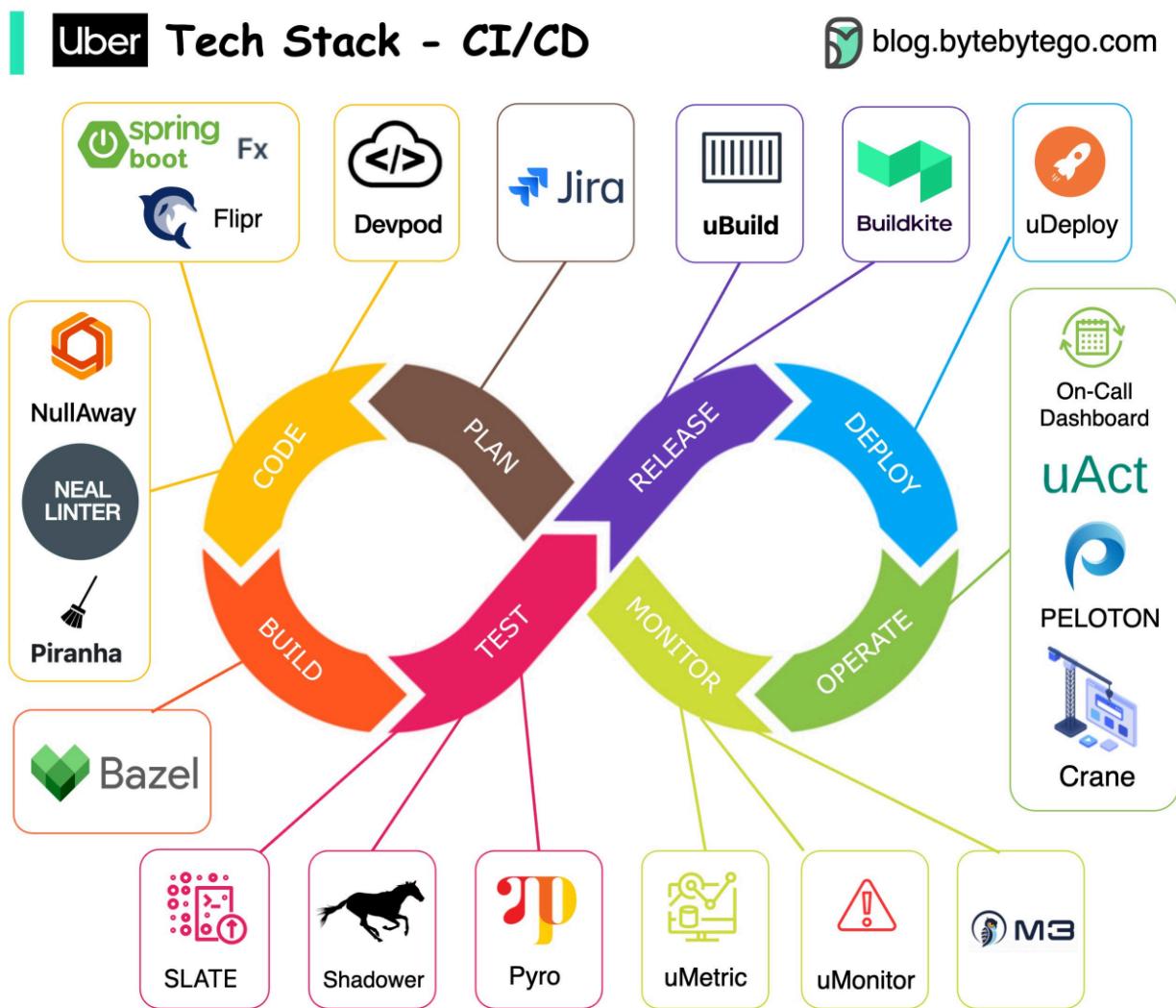
Over to you:

What anti-patterns have you faced in your cloud-native journey? How did you conquer them?

Uber Tech Stack - CI/CD

Uber is one of the most innovative companies in the engineering field. Let's take a look at their CI/CD tech stacks.

Note: This post is based on research on Uber engineering blogs. If you spot any inaccuracies, please let us know.



Project planning: JIRA

Backend services: Spring Boot to develop their backend services. And to make things even faster, they've created a nifty configuration system called Flipr that allows for speedy configuration releases.

Code issues: They developed NullAway to tackle NullPointer problems and NEAL to lint the code. Plus, they built Piranha to clean out-dated feature flags.

Repository: They believe in Monorepo. It uses Bazel on a large scale.

Testing: They use SLATE to manage short-lived testing environments and rely on Shadower for load testing by replaying production traffic. They even developed Ballast to ensure a smooth user experience.

Experiment platform: it is based on deep learning and they've generously open-sourced parts of it, like Pyro.

Build: Uber packages their services into containers using uBuild. It's their go-to tool, powered by Buildkite, for all the packaging tasks.

Deploying applications: Netflix Spinnaker. It's their trusted tool for getting things into production smoothly and efficiently.

Monitoring: Uber built their own monitoring systems. They use the uMetric platform, built on Cassandra, to keep things consistent.

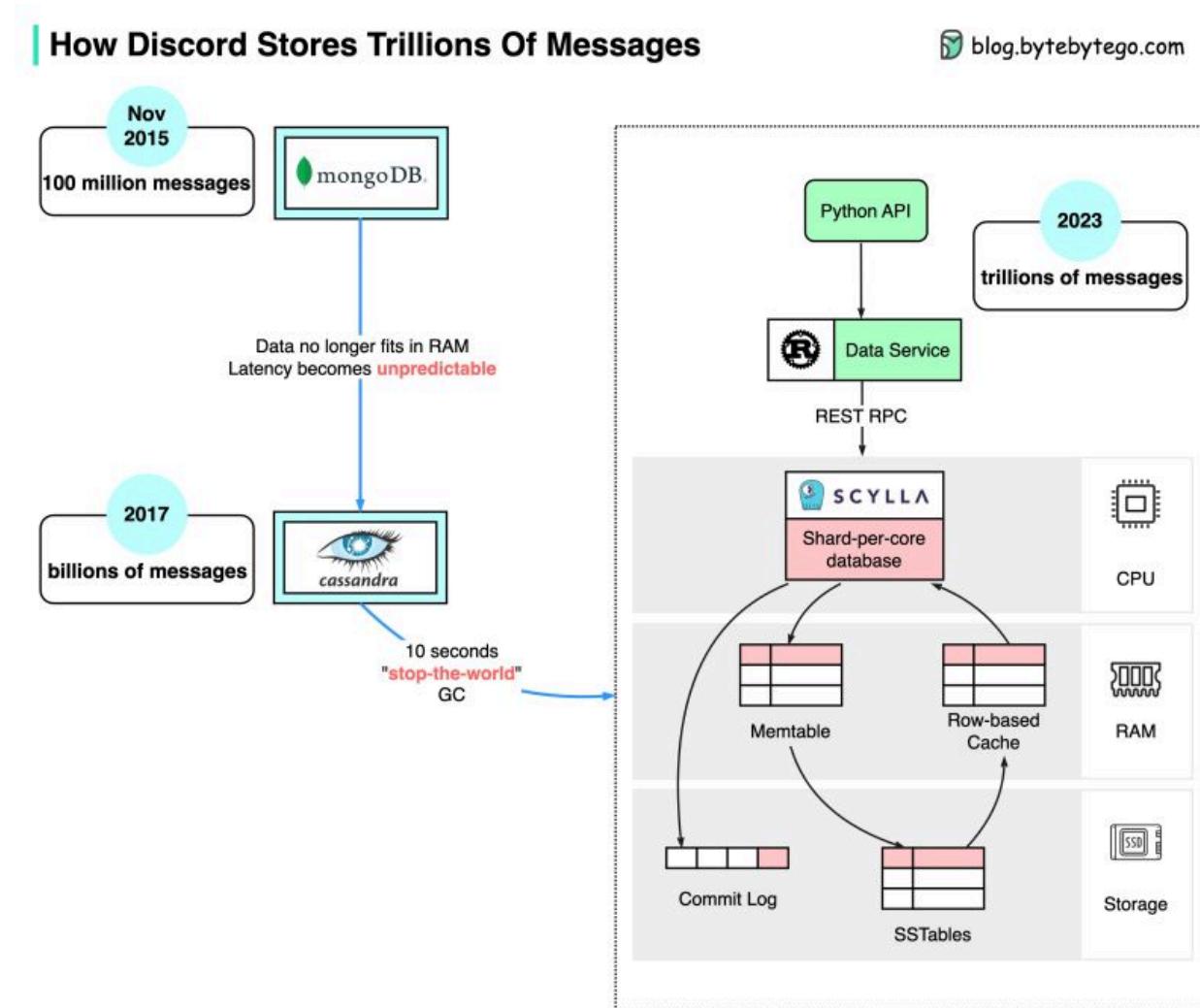
Special tooling: Uber relies on Peloton for capacity planning, scheduling, and operations. Crane builds a multi-cloud infrastructure to optimize costs. And with uAct and the OnCall dashboard, they've got event tracing and on-call duty management covered.

Have you ever used any of Uber's tech stack for CI/CD? What are your thoughts on their CI/CD setup?

How Discord Stores Trillions Of Messages

The diagram below shows the evolution of message storage at Discord:

MongoDB → Cassandra → ScyllaDB



In 2015, the first version of Discord was built on top of a single MongoDB replica. Around Nov 2015, MongoDB stored 100 million messages and the RAM couldn't hold the data and index any longer. The latency became unpredictable. Message storage needs to be moved to another database. Cassandra was chosen.

In 2017, Discord had 12 Cassandra nodes and stored billions of messages.

At the beginning of 2022, it had 177 nodes with trillions of messages. At this point, latency was unpredictable, and maintenance operations became too expensive to run.

There are several reasons for the issue:

- Cassandra uses the LSM tree for the internal data structure. The reads are more expensive than the writes. There can be many concurrent reads on a server with hundreds of users, resulting in hotspots.
- Maintaining clusters, such as compacting SSTables, impacts performance.
- Garbage collection pauses would cause significant latency spikes

ScyllaDB is a Cassandra compatible database written in C++. Discord redesigned its architecture to have a monolithic API, a data service written in Rust, and ScyllaDB-based storage.

The p99 read latency in ScyllaDB is 15ms compared to 40-125ms in Cassandra. The p99 write latency is 5ms compared to 5-70ms in Cassandra.

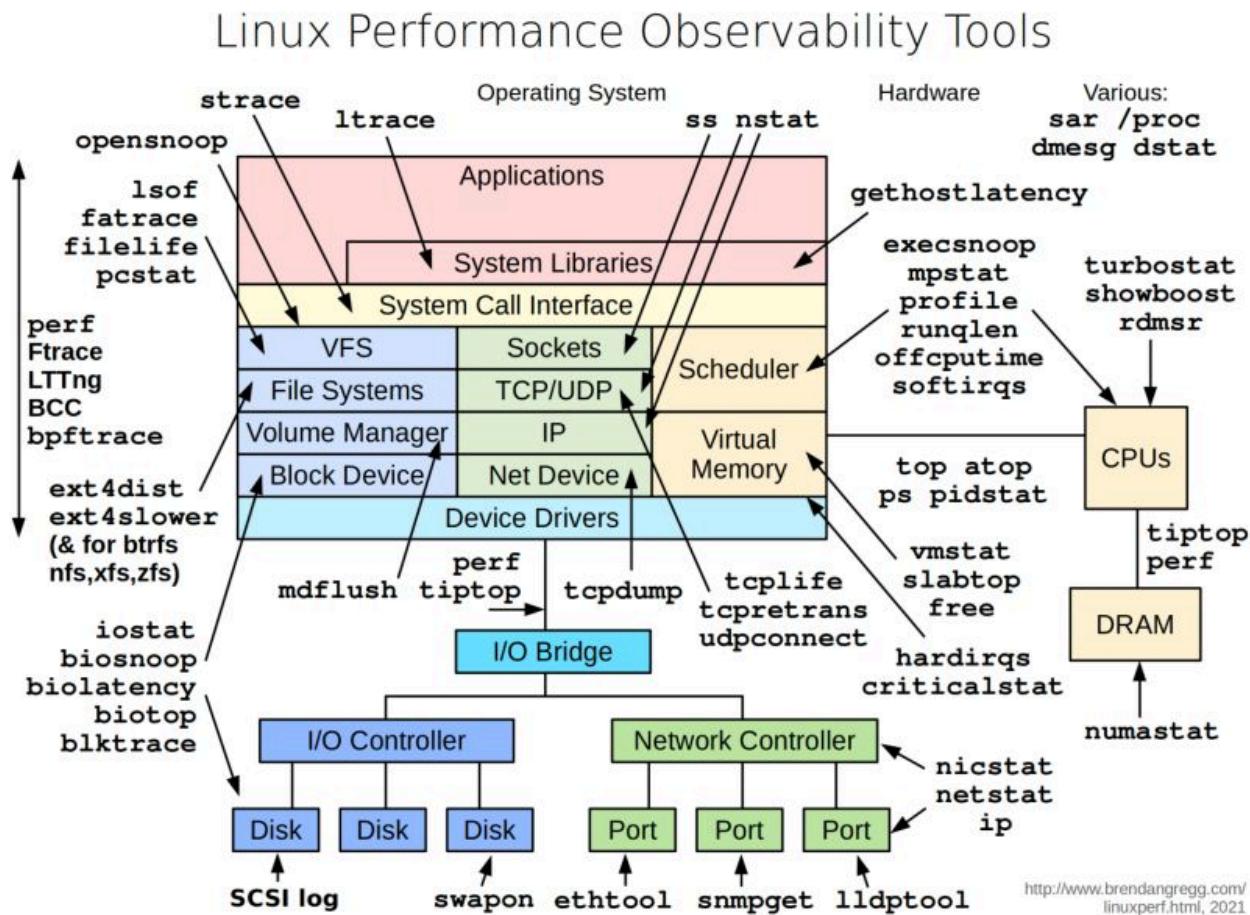
Over to you: What kind of NoSQL database have you used? How do you like it?

References:

- [Shards per core architecture](#)
- [How discord stores trillions of messages](#)

How to diagnose a mysterious process that's taking too much CPU, memory, IO, etc?

The diagram below illustrates helpful tools in a Linux system.



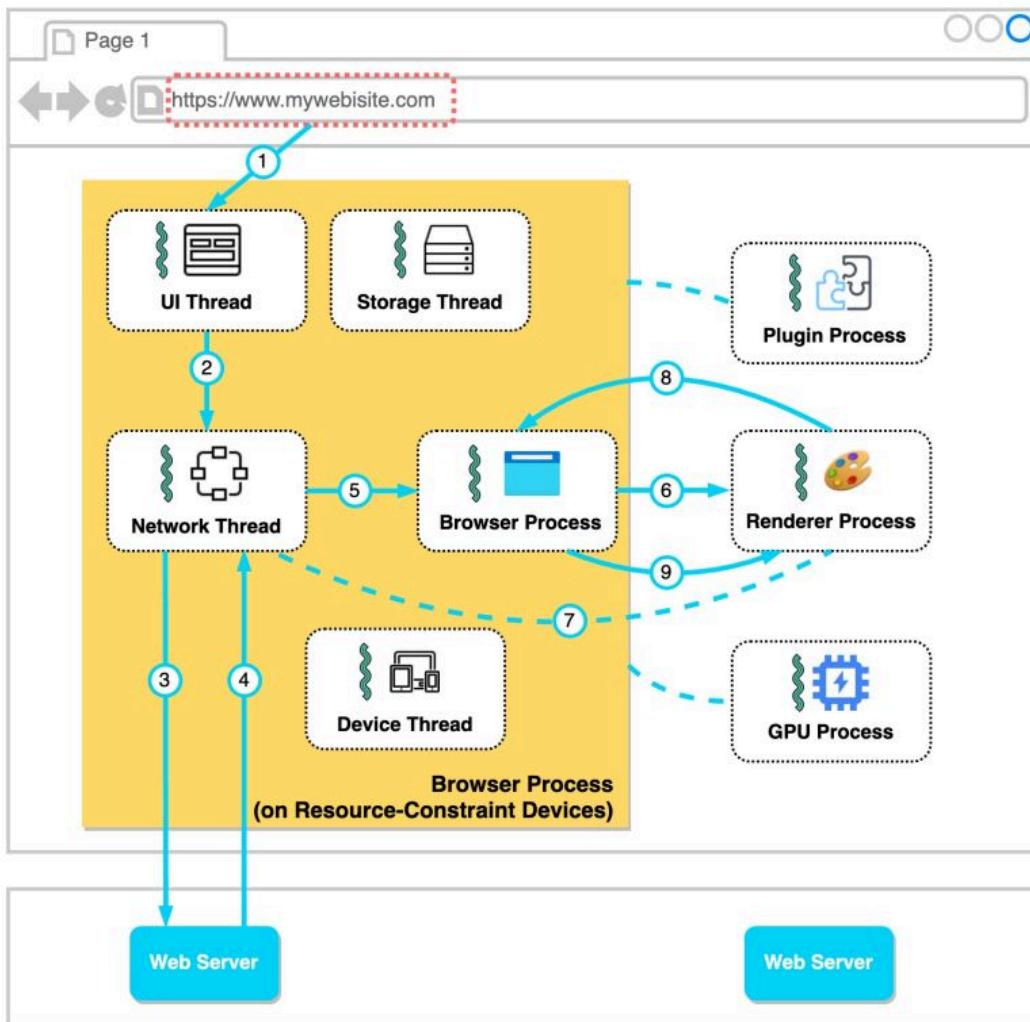
- ‘vmstat’ - reports information about processes, memory, paging, block IO, traps, and CPU activity.
- ‘iostat’ - reports CPU and input/output statistics of the system.
- ‘netstat’ - displays statistical data related to IP, TCP, UDP, and ICMP protocols.
- ‘lsof’ - lists open files of the current system.
- ‘pidstat’ - monitors the utilization of system resources by all or specified processes, including CPU, memory, device IO, task switching, threads, etc.

Diagram Credit: Linux Performance by Brendan Gregg

How does Chrome work?

The diagram below shows the architecture of a modern browser. It is based on our understanding of “Inside look at modern web browser” published by the chrome team.

How Chrome Works?



Source: <https://developer.chrome.com/blog/inside-browser-part1/>

There are in general 4 processes: browser process, renderer process, GPU process, and plugin process.

- Browser process controls the address bar, bookmarks, back and forward buttons, etc.
- Renderer process controls anything inside of the tab where a website is displayed.
- GPU process handles GPU tasks.
- Plugin process controls the plugins used by the websites.

The browser process coordinates with other processes.

When Chrome runs on powerful hardware, it may split each service in the browser process into different threads, as the diagram below shows. This is called Servicification.

Now let's go through the steps when we enter a URL in Chrome.

Step 1: The user enters a URL into the browser. This is handled by the UI thread.

Step 2: When the user hits enter, the UI thread initiates a network call to get the site content.

Steps 3-4: The network thread goes through appropriate network protocols and retrieves the content.

Step 5: When the network thread receives responses, it looks at the first few bytes of the stream. If it is an HTML file, it is passed to the renderer process by the browser process.

Steps 6-9: An IPC is sent from the browser process to the renderer process to commit the navigation. A data pipe is established between the network thread and the renderer process so that the renderer can receive data. Once the browser process hears confirmation that the commit has happened in the renderer process, the navigation is complete and the document loading phase begins.

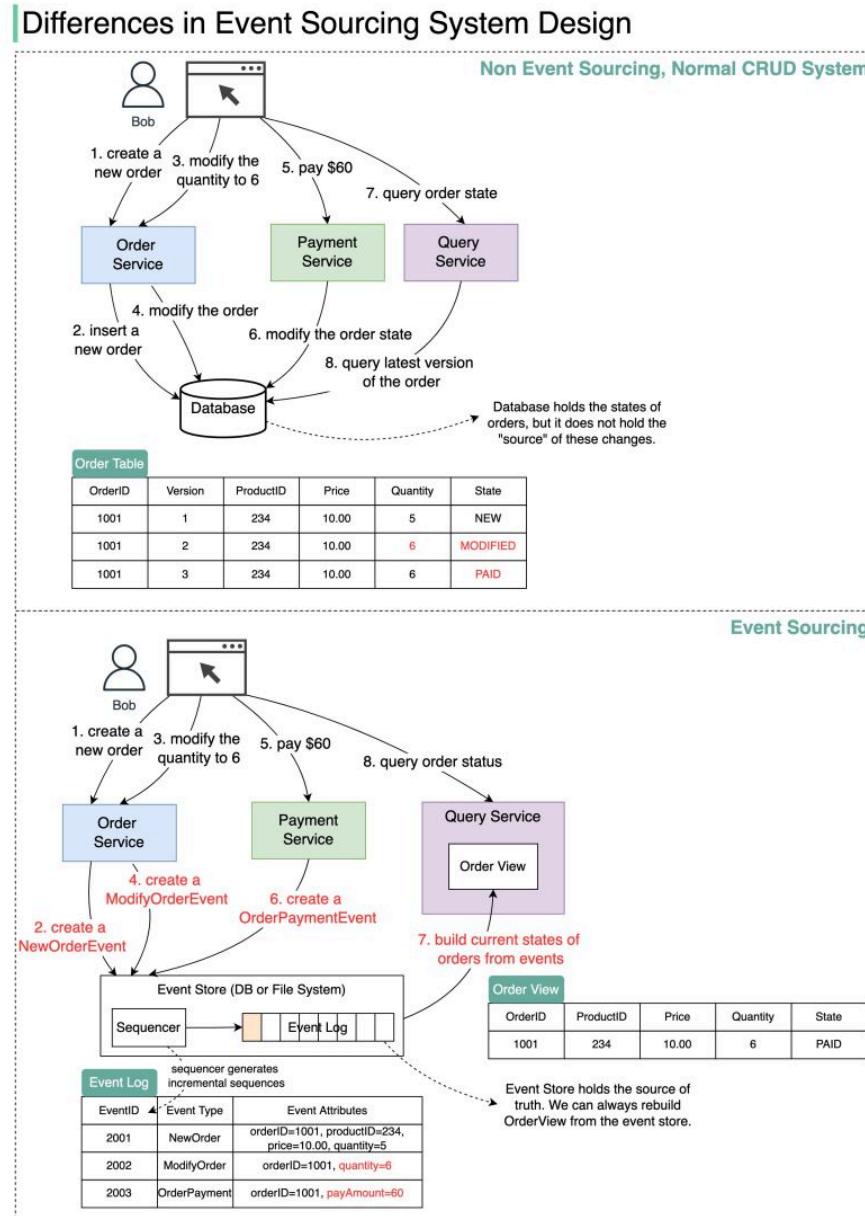
Over to you: Why does Chrome assign each tab a renderer process?

Reference: [Inside look at modern web browser](#)

Differences in Event Sourcing System Design

How do we design a system using the **event sourcing** paradigm? How is it different from normal system design? What are the benefits? We will talk about it in this post.

The diagram below shows the comparison of a normal CRUD system design with an event sourcing system design. We use an e-commerce system that can place orders and pay for the orders to demonstrate how event sourcing works.



The event sourcing paradigm is used to design a system with determinism. This changes the philosophy of normal system designs.

How does this work? Instead of recording the order states in the database, the event sourcing design persists the events that lead to the state changes in the event store. The event store is an append-only log. The events must be sequenced with incremental numbers to guarantee their ordering. The order states are rebuilt from the events and maintained in OrderView. If the OrderView is down, we can always rely on the event store which is the source of truth to recover the order states.

Let's look at the steps in detail.

- Non-Event Sourcing

Steps 1 and 2: Bob wants to buy a product. The order is created and inserted into the database.

Steps 3 and 4: Bob wants to change the quantity from 5 to 6. The order is modified with a new state.

Steps 5 and 6: Bob pays \$60 for the order. The order is complete and the state is Paid.

Steps 7 and 8: Bob queries the latest order state. Query service retrieves the state from the database.

- Event Sourcing

Steps 1 and 2: Bob wants to buy a product. A NewOrderEvent is created, sequenced and stored in the event store with eventID=2001.

Steps 3 and 4: Bob wants to change the quantity from 5 to 6. A ModifyOrderEvent is created, sequenced, and persisted in the event store with eventID=2002.

Steps 5 and 6: Bob pays \$60 for the order. An OrderPaymentEvent is created, sequenced, and stored in the event store with eventID=2003. Notice the different event types have different event attributes.

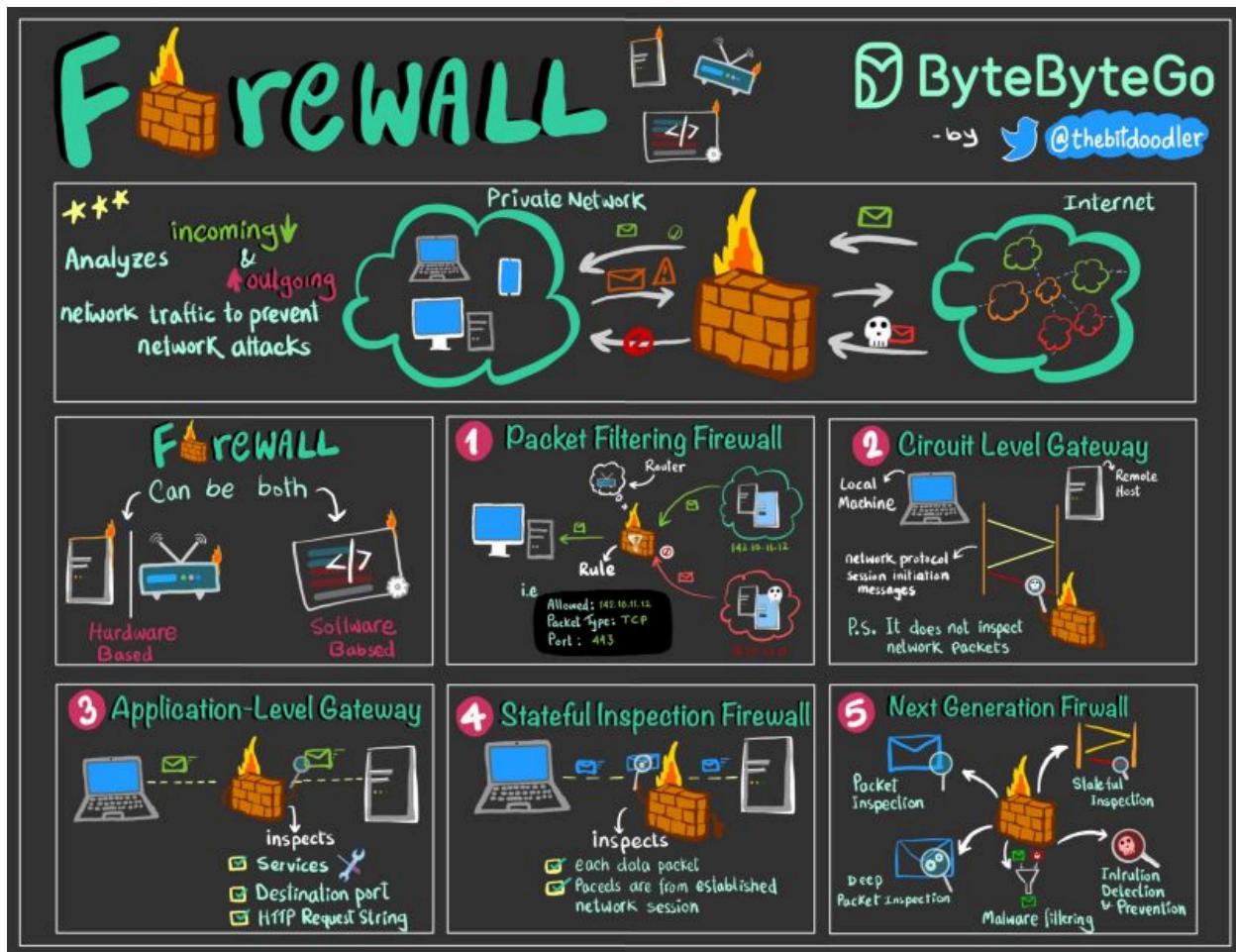
Step 7: OrderView listens on the events published from the event store, and builds the latest state for the orders. Although OrderView receives 3 events, it applies the events one by one and keeps the latest state.

Step 8: Bob queries the order state from OrderService, which then queries OrderView. OrderView can be in memory or cache and does not need to be persisted, because it can be recovered from the event store.

Over to you: Which type of system is suitable for event sourcing design? Have you used this paradigm in your work?

Firewall explained to Kids... and Adults

A firewall is a network security system that controls and filters network traffic, acting as a watchman between a private network and the public Internet.



They come in two broad categories:

Software-based: installed on individual devices for protection

Hardware-based: stand-alone devices that safeguard an entire network.

Firewalls have several types, each designed for specific security needs:

1. **Packet Filtering Firewalls:** Examines packets of data, accepting or rejecting based on source, destination, or protocols.
2. **Circuit-level Gateways:** Monitors TCP handshake between packets to determine session legitimacy.

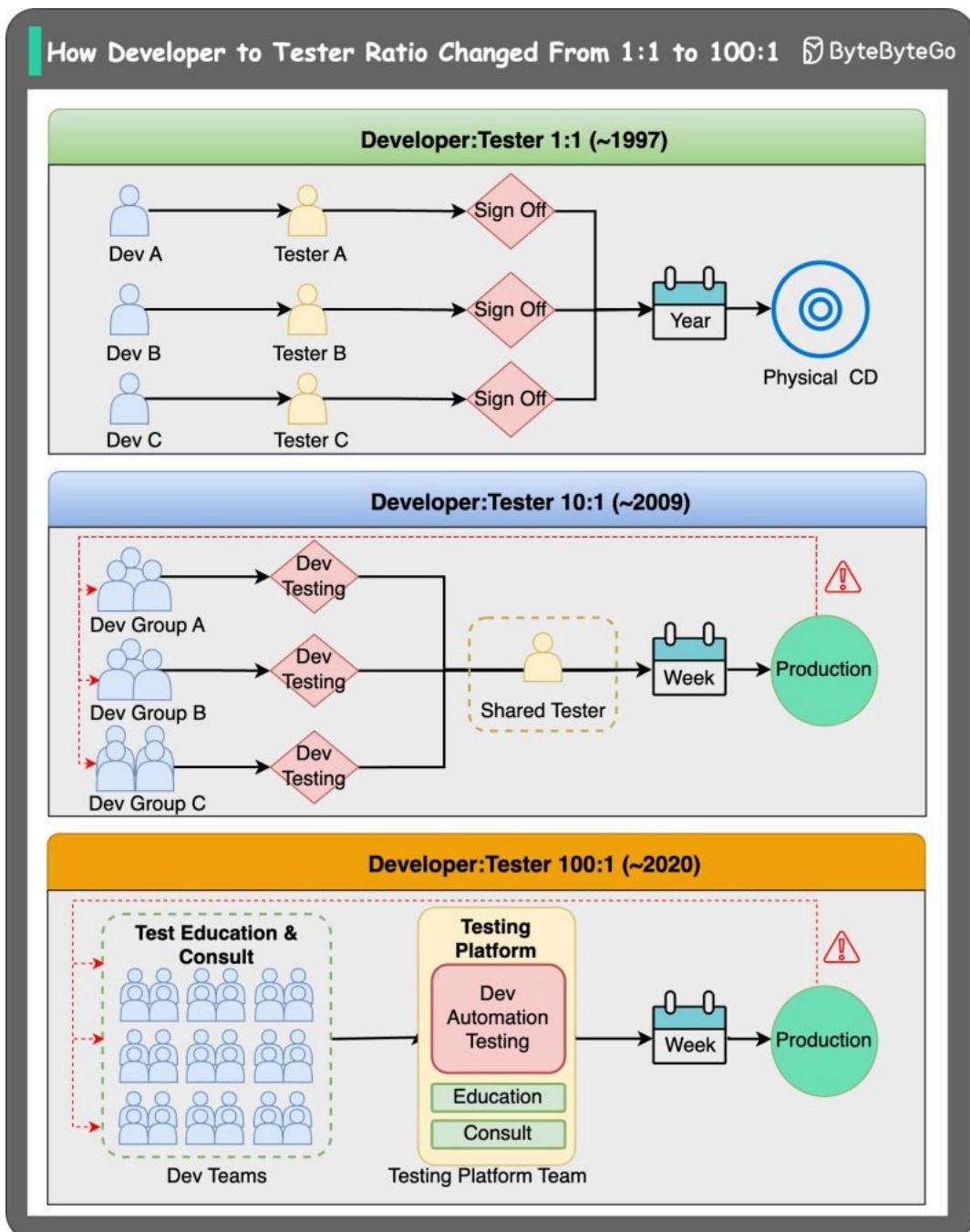
3. Application-level Gateways (Proxy Firewalls): Filters incoming traffic between your network and traffic source, offering a protective shield against untrusted networks.
4. Stateful Inspection Firewalls: Tracks active connections to determine which packets to allow, analyzing in the context of their place in a data stream.
5. Next-Generation Firewalls (NGFWs): Advanced firewalls that integrate traditional methods with functionalities like intrusion prevention systems, deep packet analysis, and application awareness.

Over to you: Do you know what firewalls your company uses?

Paradigm Shift: How Developer to Tester Ratio Changed From 1:1 to 100:1

This post is inspired by the article "The Paradigm Shifts with Different Dev:Test Ratios" by [Carlos Arguelles](#)

I highly recommend that you read the original article here: <https://lnkd.in/ehbZzZck>



1:1 ratio (~1997)

Software used to be burned onto physical CDs and delivered to customers. The development process was waterfall-style, builds were certified, and versions were released roughly every three years.

If you had a bug, that bug would live forever. It wasn't until years later that companies added the ability for software to ping the internet for updates and automatically install them.

10:1 ratio (~2009)

Around 2009, the release-to-production speed increased significantly. Patches could be installed within weeks, and the agile movement, along with iteration-driven development, changed the development process.

For example, at Amazon, the web services are mainly developed and tested by the developers. They are also responsible for dealing with production issues, and testing resources are stretched thin (10:1 ratio).

100:1 ratio (~2020)

Around 2015, big tech companies like Google and Microsoft removed SDET or SETI titles, and Amazon slowed down the hiring of SDETs.

But how is this going to work for big tech in terms of testing?

Firstly, the testing aspect of the software has shifted towards highly scalable, standardized testing tools. These tools have been widely adopted by developers for building their own automated tests.

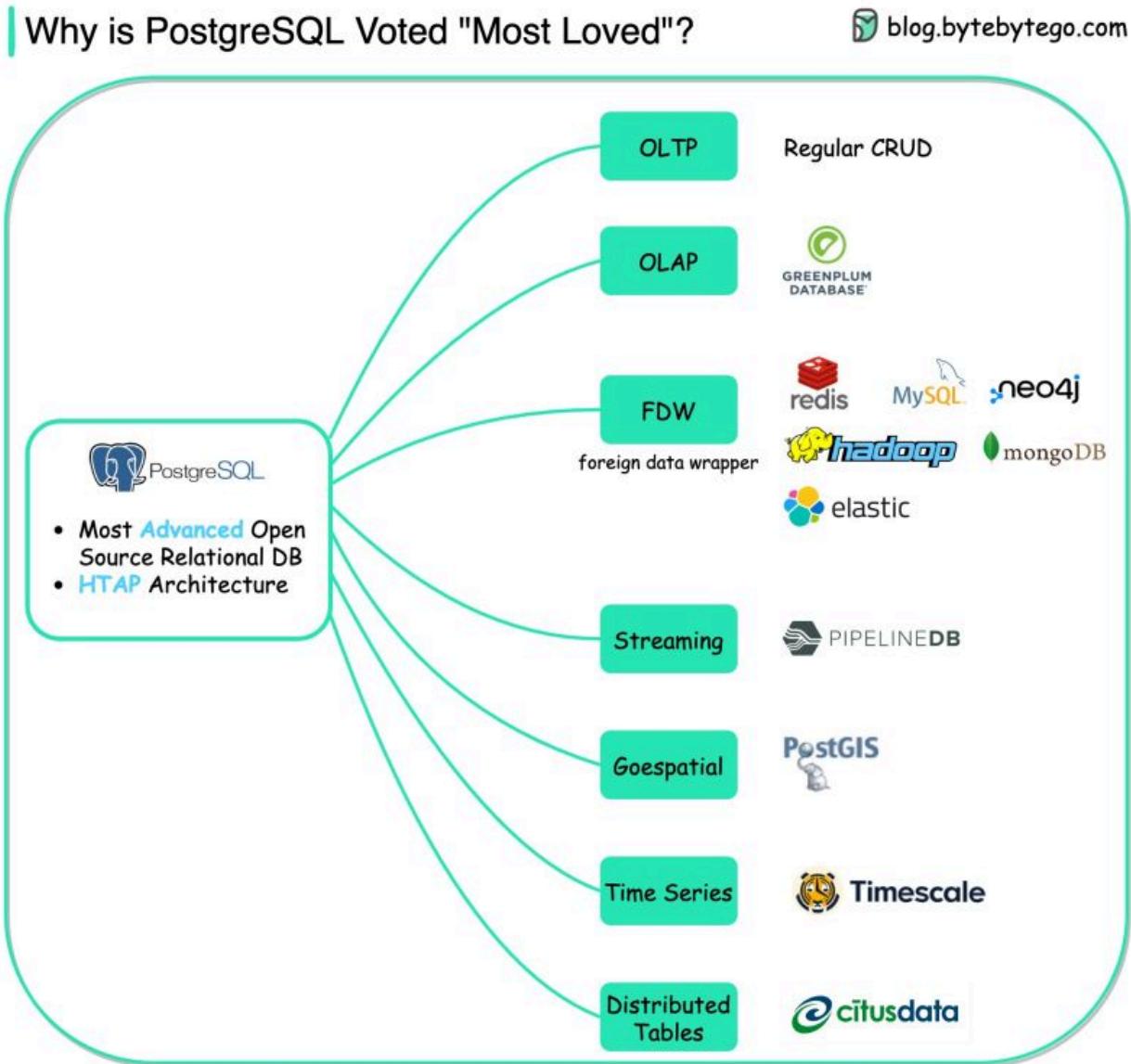
Secondly, testing knowledge is disseminated through education and consulting.

Together, these factors have facilitated a smooth transition to the 100:1 testing ratio we see today.

Over to you: What does the future hold for testing, and how is it currently working for you?

Why is PostgreSQL voted as the most loved database by developers?

The diagram shows the many use cases by PostgreSQL - one database that includes almost all the use cases developers need.



OLTP (Online Transaction Processing)

We can use PostgreSQL for CRUD (Create-Read-Update-Delete) operations.

OLAP (Online Analytical Processing)

We can use PostgreSQL for analytical processing. PostgreSQL is based on HTAP (Hybrid transactional/analytical processing) architecture, so it can handle both OLTP and OLAP well.

FDW (Foreign Data Wrapper)

A FDW is an extension available in PostgreSQL that allows us to access a table or schema in one database from another.

Streaming

PipelineDB is a PostgreSQL extension for high-performance time-series aggregation, designed to power real-time reporting and analytics applications.

Geospatial

PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects, allowing location queries to be run in SQL.

Time Series

Timescale extends PostgreSQL for time series and analytics. For example, developers can combine relentless streams of financial and tick data with other business data to build new apps and uncover unique insights.

Distributed Tables

CitusData scales Postgres by distributing data & queries.

8 Key OOP Concepts Every Developer Should Know

Object-Oriented Programming (OOP) has been around since the 1960s, but it really took off in the 1990s with languages like Java and C++.

8 Key OOP Concepts Everyone Should Know ByteByteGo.com

Class	Template
A blueprint for creating objects, encapsulating data and methods that operate on that data	
Object	Instance
Object is an instance of a class, embodying the data and methods defined in the class.	
Encapsulation	Data hiding
Bundling data and methods operating on that data within a single unit, called a class	
Inheritance	Code Reusability
Allows a class to inherit attributes and methods from another class, promoting code reusability	
Polymorphism	Multiple Forms
Polymorphism enables one interface or method to be used for different data types and classes	
Association	has-a
One class uses another. "has-a" relationship, so there is no dependency on each other	
Aggregation	Whole-part
A group, body, or mass composed of many distinct parts or individuals. no life time ownership	
Composition	Ownership
An object of one class owns objects of another class and is responsible for its lifetime	

Why is OOP Important? OOP allows you to create blueprints (called classes) for digital objects, and these objects know how to communicate with one another to make amazing things happen

in your software. Having a well-organized toolbox rather than a jumbled drawer of tools makes your code tidier and easier to change.

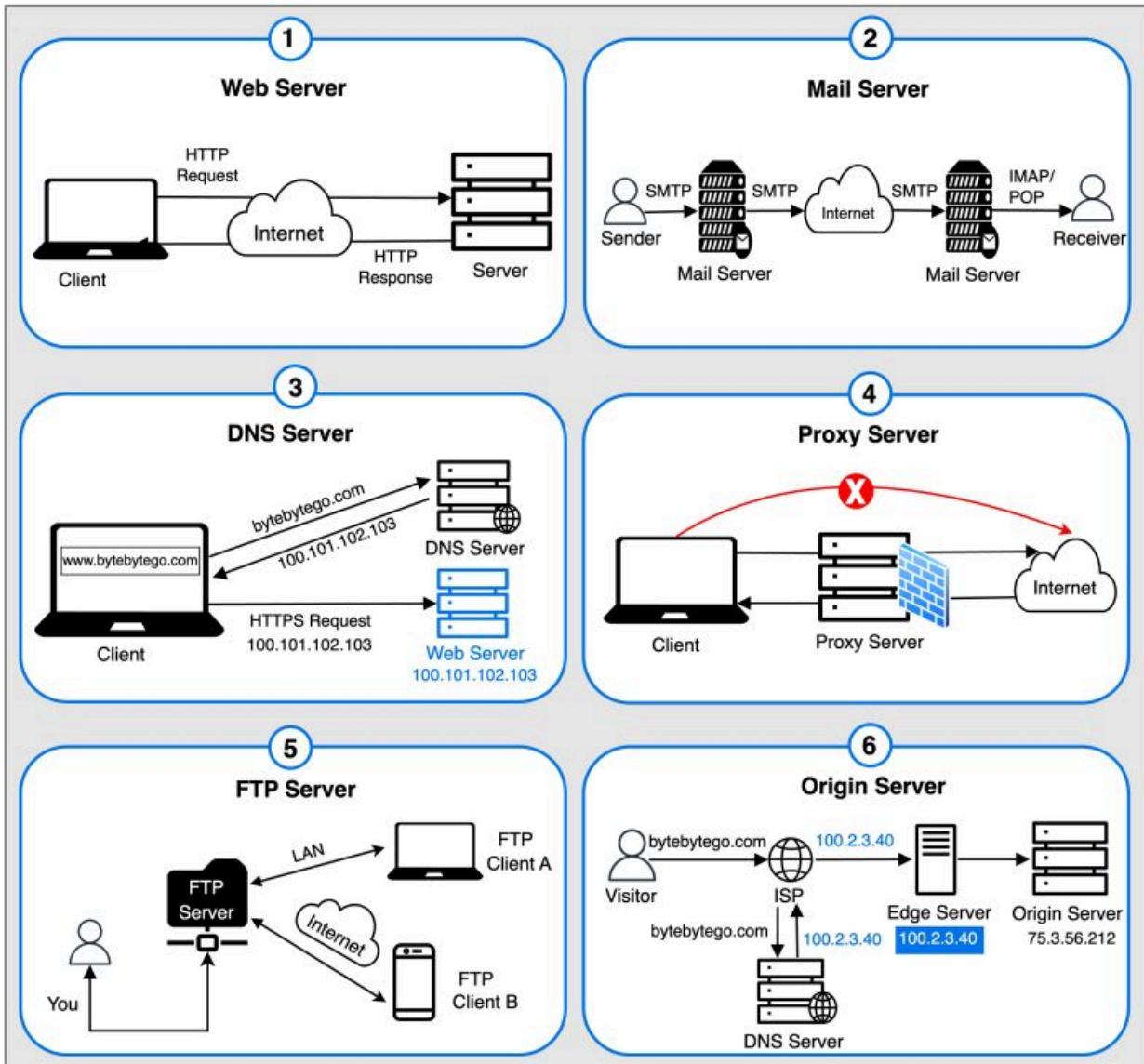
In order to get to grips with OOP, think of it as creating digital Lego blocks that can be combined in countless ways. Take a book or watch some tutorials, and then practice writing code - there's no better way to learn than to practice!

Don't be afraid of OOP - it's a powerful tool in your coder's toolbox, and with some practice, you'll be able to develop everything from nifty apps to digital skyscrapers!

Top 6 most commonly used Server Types

Top 6 Most Commonly Used Server Types

 blog.bytebytego.com



1. Web Server: Hosts websites and delivers web content to clients over the internet
2. Mail Server: Handles the sending, receiving, and routing of emails across networks
3. DNS Server: Translates domain names (like bytebytego.com) into IP addresses, enabling users to access websites by their human-readable names.

4. Proxy Server: An intermediary server that acts as a gateway between clients and other servers, providing additional security, performance optimization, and anonymity.
5. FTP Server: Facilitates the transfer of files between clients and servers over a network
6. Origin Server: Hosts central source of content that is cached and distributed to edge servers for faster delivery to end users.

Over to you: Which type of server do you find most crucial in your online experience?

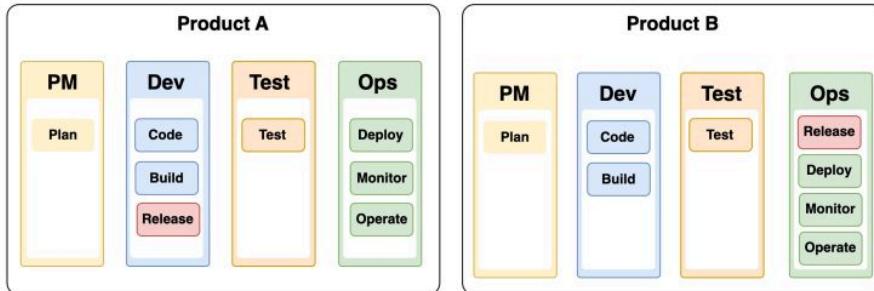
DevOps vs. SRE vs. Platform Engineering. Do you know the differences?

DevOps vs SRE vs Platform Engineering

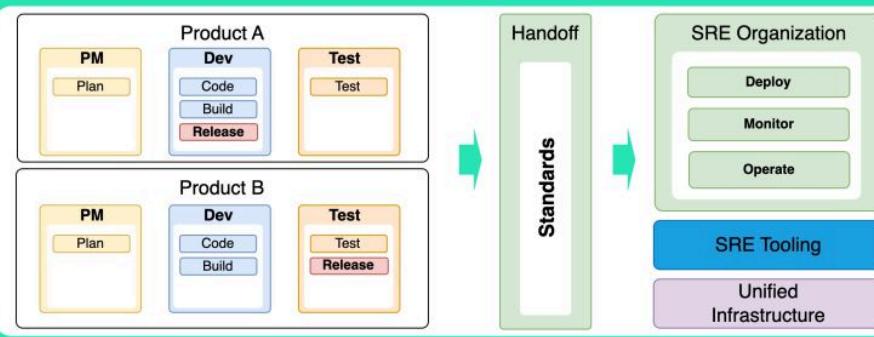


blog.bytebytego.com

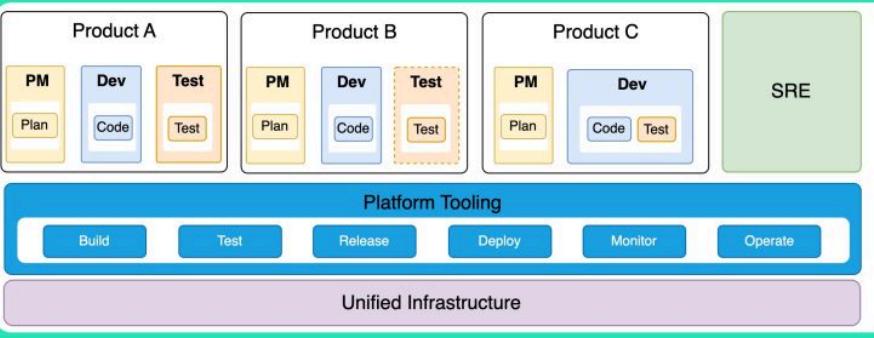
DevOps



SRE (Site Reliability Engineering)



Platform Engineering



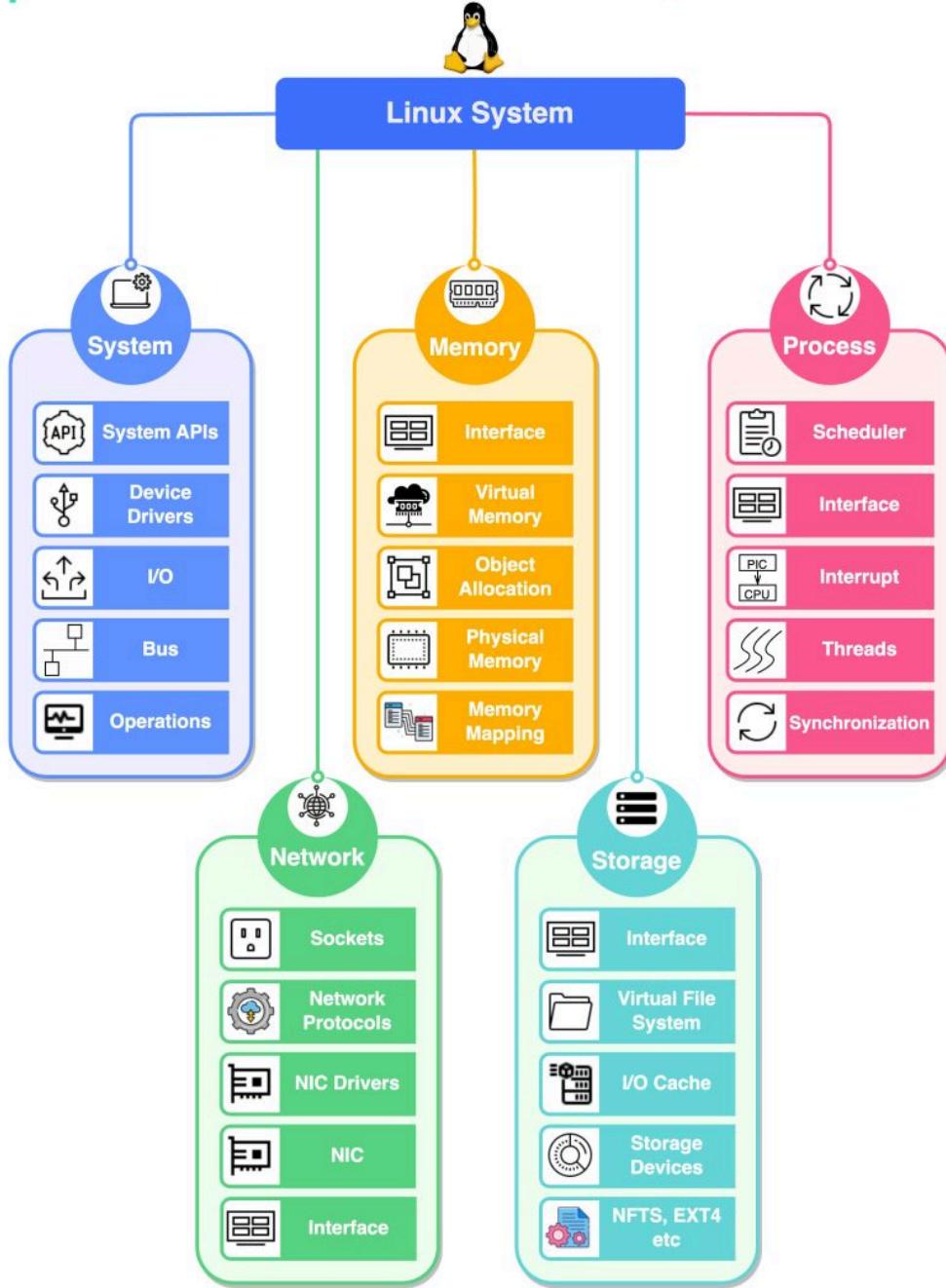
In this video, we will talk about:

- Who invented DevOps?
- What is SRE? What are some of the best SRE practices and tools?

- What is Platform Engineering? How is it different from others?
- How can they be used to improve collaboration, automation, and efficiency in software development and operations?

5 important components of Linux

5 important components of Linux blog.bytebytego.com



- **System**

In the system component, we need to learn modules like system APIs, device drivers, I/O, buses, etc.

- **Memory**

In memory management, we need to learn modules like physical memory, virtual memory, memory mappings, object allocation, etc.

- **Process**

In process management, we need to learn modules like process scheduling, interrupts, threads, synchronization, etc.

- **Network**

In the network component, we need to learn important modules like network protocols, sockets, NIC drivers, etc.

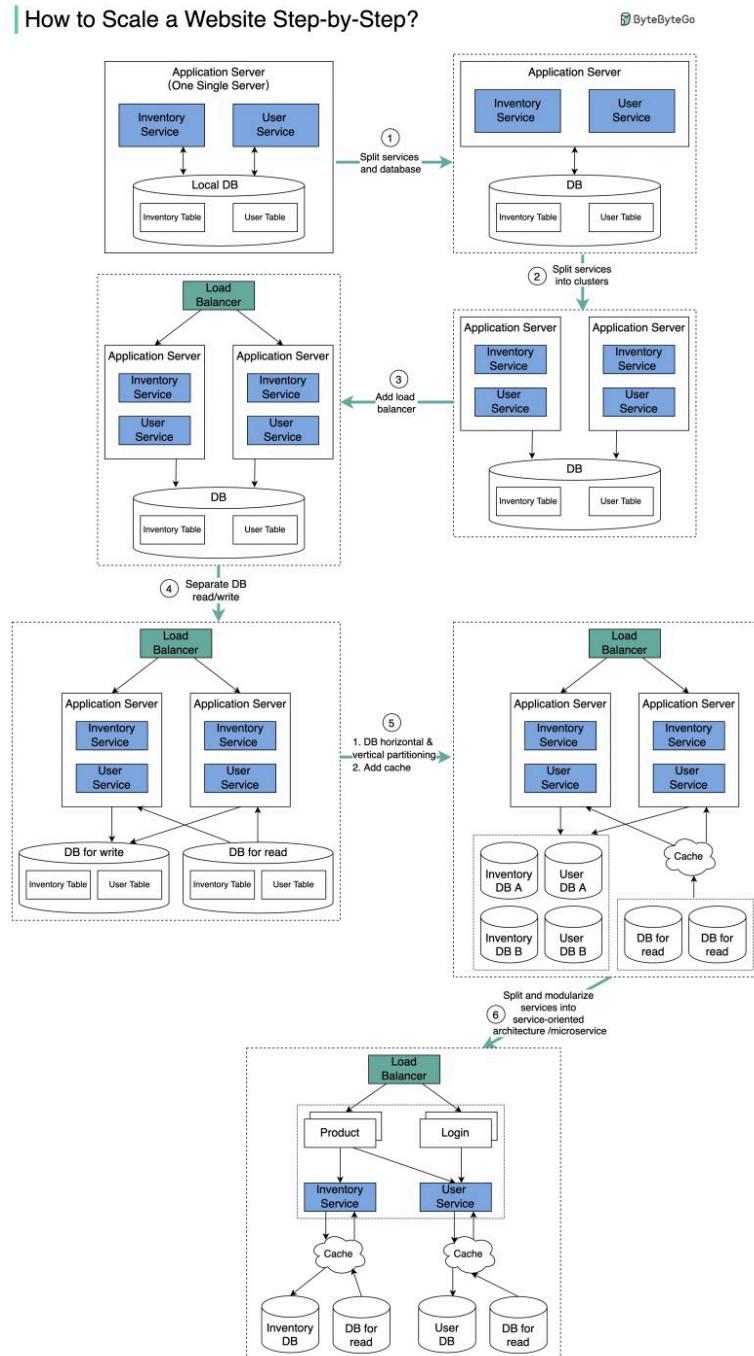
- **Storage**

In system storage management, we need to learn modules like file systems, I/O caches, different storage devices, file system implementations, etc.

How to scale a website to support millions of users?

We will explain this step-by-step.

The diagram below illustrates the evolution of a simplified eCommerce website. It goes from a monolithic design on one single server, to a service-oriented/microservice architecture.



Suppose we have two services: inventory service (handles product descriptions and inventory management) and user service (handles user information, registration, login, etc.).

Step 1 - With the growth of the user base, one single application server cannot handle the traffic anymore. We put the application server and the database server into two separate servers.

Step 2 - The business continues to grow, and a single application server is no longer enough. So we deploy a cluster of application servers.

Step 3 - Now the incoming requests have to be routed to multiple application servers, how can we ensure each application server gets an even load? The load balancer handles this nicely.

Step 4 - With the business continuing to grow, the database might become the bottleneck. To mitigate this, we separate reads and writes in a way that frequent read queries go to read replicas. With this setup, the throughput for the database writes can be greatly increased.

Step 5 - Suppose the business continues to grow. One single database cannot handle the load on both the inventory table and user table. We have a few options:

Step 6 - Now we can modularize the functions into different services. The architecture becomes service-oriented / microservice.

What is FedNow (instant payment)

JPMorgan, Wells Fargo, and other major banks will use the new Federal Reserve's 'FedNow' instant payment system. Let's take a look at how it works.

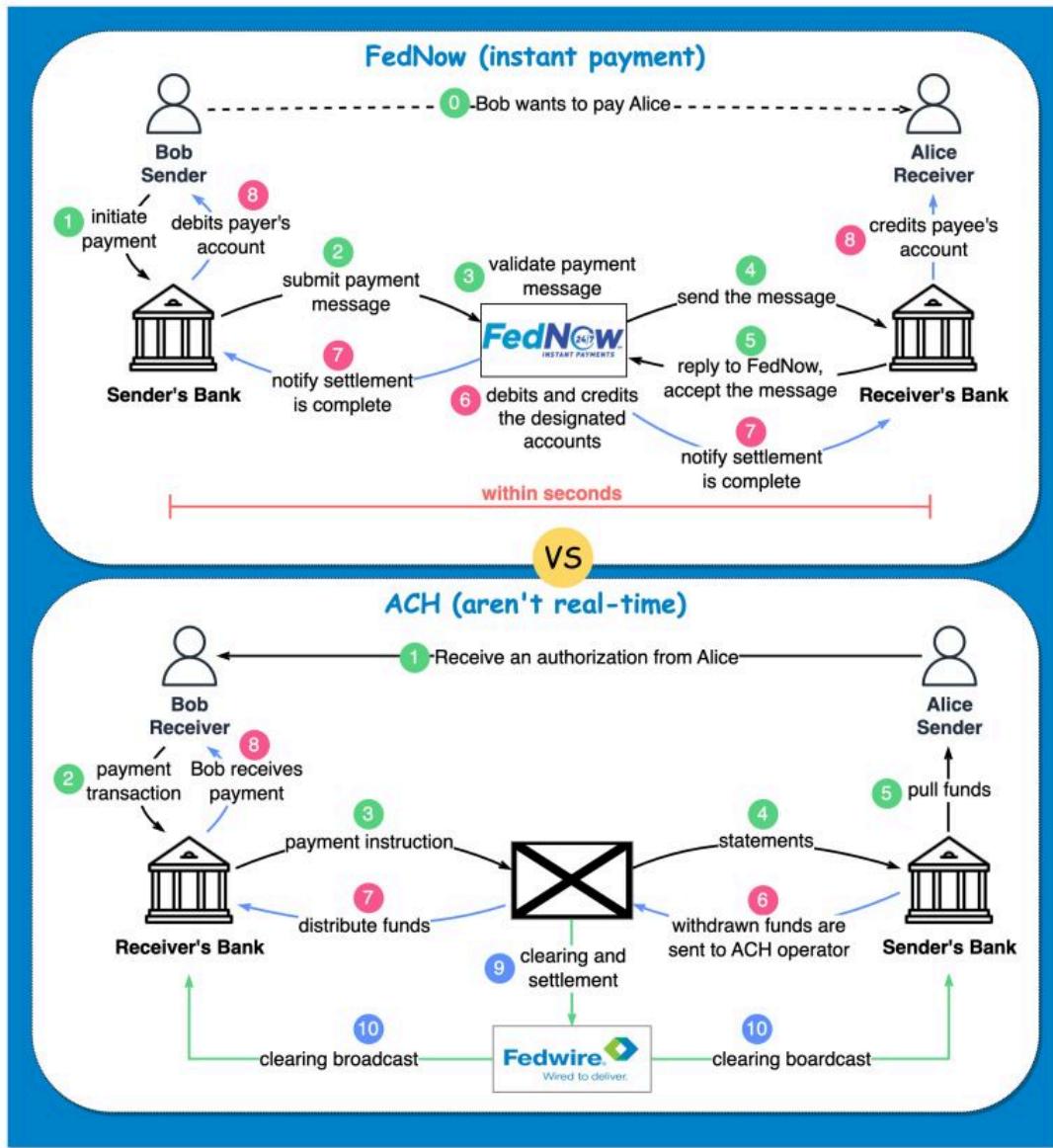
Federal Reserve launched FedNow instant payment service on 20 Jul. It allows retail clients to send and receive money within seconds and it is available 24x7.

- What does this mean?
- 1. Peer-to-peer payment services in the private sector like Venmo or PayPal act as intermediaries between banks, so we need to leverage payment schemes for clearing and Fed systems for settlement. However, FedNow can directly settle the transactions in central bank accounts. [1]
- 2. Fedwire, another real-time payments system, will still function in large-value or low-value payments. FedNow is not designed to replace Fedwire.

The diagram below shows a comparison between FedNow and ACH (Automated Clearing House), which is used in domestic low-value payments.

What is FedNow (instant payment)

 blog.bytebytogo.com



Source: <https://www.klaros.com/post/q-a-on-the-federal-reserve-s-fednow-service>

- FedNow [2]
 - Step 0 - Bob wants to pay Alice \$1000.
 - Step 1 - Bob initiates a payment transaction using FedNow.
 - Step 2 - The sender's bank submits a payment message to FedNow.
 - Step 3 - The FedNow service validates the payment message.
 - Step 4 - The FedNow service sends the payment message to the receiver's bank, where it is confirmed.
 - Step 5 - The receiver's bank replies to FedNow, confirming that the payment is accepted.
 - Step 6 - The FedNow service debits and credits the designated accounts of the sender and receiver's banks.

Step 7 - The FedNow service notifies the sender's bank and receiver's bank that the settlement is complete.

Step 8 - The banks debit and credit the bank accounts.

- ACH

Step 1 - Bob receives authorization from Alice that he can deduct from Alice's account.

Step 2 - The payment transaction is sent to the receiver's bank.

Step 3 - The bank collects files in batches and sends them to the ACH operator.

Step 4 - The ACH operator sends the files to the sender's bank.

Step 5 - The sender's bank pulls funds from Alice's account.

Step 6 - Withdrawn funds are sent to the ACH operator.

Step 7 - The ACH operator distributes funds to Bob's bank.

Step 8 - Bob receives the fund.

Step 9 - The clearing instructions are sent to Fedwire.

Step 10 - Fedwire sends clearing broadcasts to banks for settlements.

Over to you: What types of instant payment systems does your country provide?

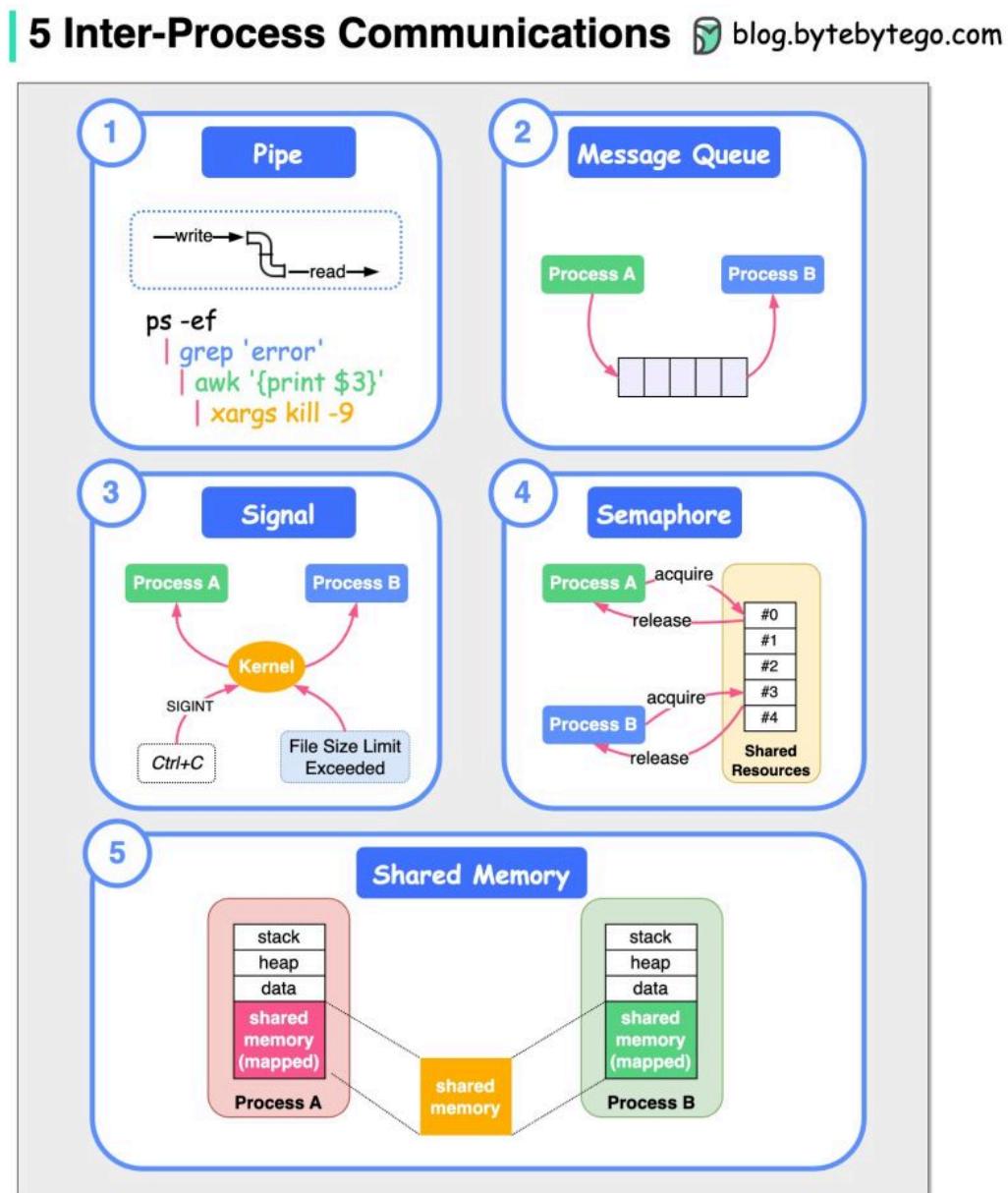
Reference:

[1] [Federal Reserve launches FedNow instant payment service that could bypass Venmo and PayPal](#)

[2] [Q&A on the Federal Reserve's FedNow Service](#)

5 ways of Inter-Process Communication

How do processes talk to each other on Linux? The diagram below shows 5 ways of Inter-Process Communication.



1. Pipe

Pipes are unidirectional byte streams that connect the standard output from one process to the standard input of another process.

2. Message Queue

Message queues allow one or more processes to write messages, which will be read by one or more reading processes.

3. Signal

Signals are one of the oldest inter-process communication methods used by Unix systems. A signal could be generated by a keyboard interrupt or an error condition such as the process attempting to access a non-existent location in its virtual memory. There are a set of defined signals that the kernel can generate or that can be generated by other processes in the system. For example, Ctrl+C sends a SIGINT signal to process A.

4. Semaphore

A semaphore is a location in memory whose value can be tested and set by more than one process. Depending on the result of the test and set operation one process may have to sleep until the semaphore's value is changed by another process.

5. Shared Memory

Shared memory allows one or more processes to communicate via memory that appears in all of their virtual address spaces. When processes no longer wish to share the virtual memory, they detach from it.

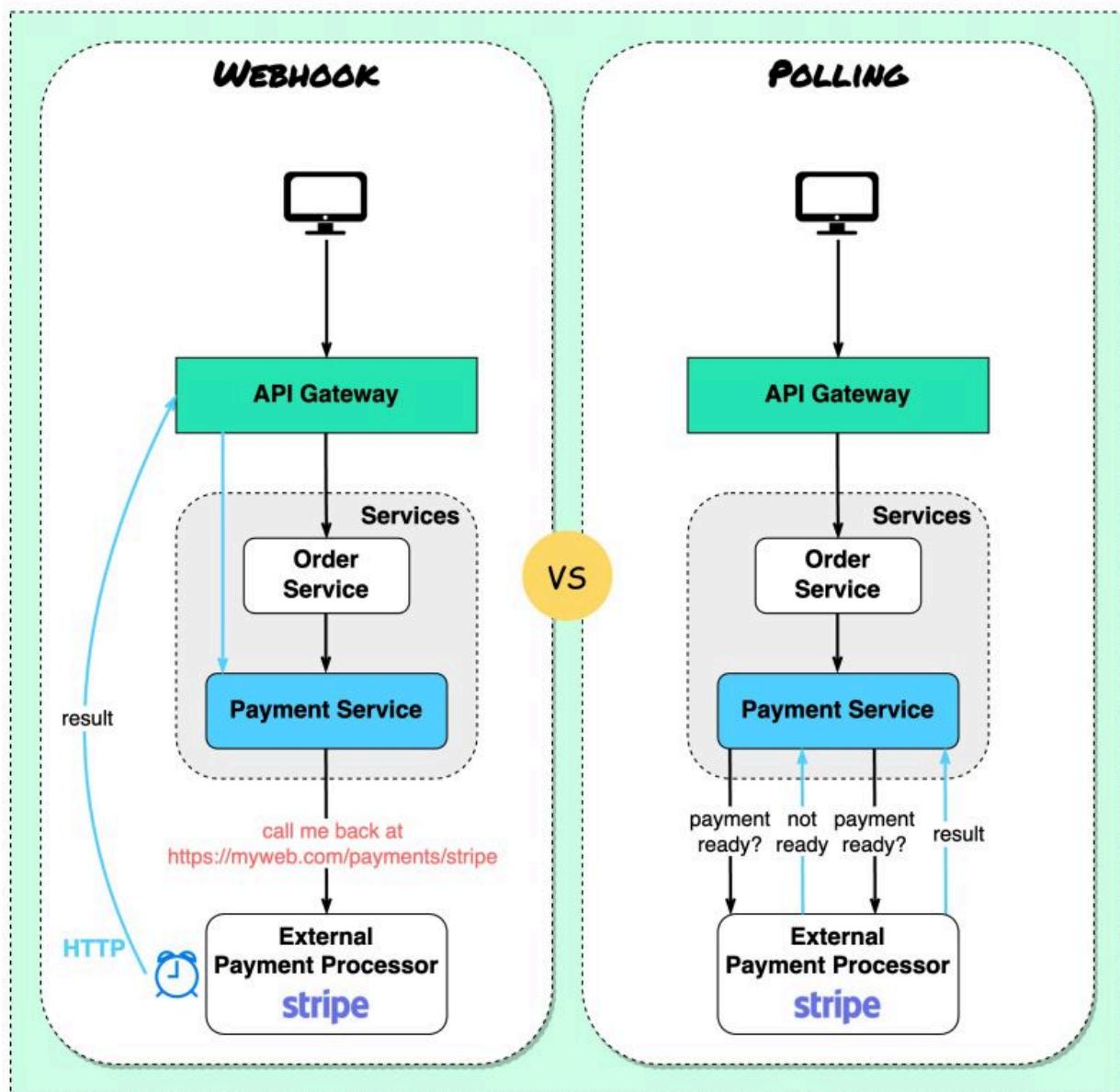
Reference: [Interprocess Communication Mechanisms](#)

What is a webhook?

The diagram below shows a comparison between polling and webhook.

What is a Webhook?

 blog.bytebytego.com



Assume we run an eCommerce website. The clients send orders to the order service via the API gateway, which goes to the payment service for payment transactions. The payment service then talks to an external payment service provider (PSP) to complete the transactions.

There are two ways to handle communications with the external PSP.

1. Short polling

After sending the payment request to the PSP, the payment service keeps asking the PSP about the payment status. After several rounds, the PSP finally returns with the status.

Short polling has two drawbacks:

- Constant polling of the status requires resources from the payment service.
- The External service communicates directly with the payment service, creating security vulnerabilities.

2. Webhook

We can register a webhook with the external service. It means: call me back at a certain URL when you have updates on the request. When the PSP has completed the processing, it will invoke the HTTP request to update the payment status.

In this way, the programming paradigm is changed, and the payment service doesn't need to waste resources to poll the payment status anymore.

What if the PSP never calls back? We can set up a housekeeping job to check payment status every hour.

Webhooks are often referred to as reverse APIs or push APIs because the server sends HTTP requests to the client. We need to pay attention to 3 things when using a webhook:

1. We need to design a proper API for the external service to call.
2. We need to set up proper rules in the API gateway for security reasons.
3. We need to register the correct URL at the external service.

What tools does your team use to ship code to production and ensure code quality?

The approach generally depends on the size of the company. There is no one-size-fits-all solution, but we try to provide a general overview.

Company Size vs. Tools They Use To Ship to Production					blog.bytebytogo.com
	S	M	L	XXL	
 Engineer	1-10	10-100	100 - 1000	1000 - 10,000+	
 Develop	 Visual Studio Code  GitHub Free or low-cost Truck based development	 Visual Studio Code  IntelliJ IDEA  GitHub Enterprise Free and commercial Truck/Feature based development	 Confluence  Jira  Visual Studio Code  IntelliJ IDEA  GitHub Enterprise  Jenkins Largely Commercial Feature based development	 Confluence  Jira  Visual Studio Code  IntelliJ IDEA  git  Workflow Commercial or customized tooling Truck based development	
 Testing	 Manual Testing	 Quality Assurance	 Quality Assurance + Automated Testing	 Automated Testing + Quality Assurance	
 Deploy	 Manual deployment	 Schedule-based deployment	 Schedule + staged canary rollout	 Schedule + Staged canary rollout + experiment	
 Operations	 Customer report	 Monitoring/Alert + Tiered Customer Support + Customer report	 Monitoring/Alert + Tiered Customer Support + Customer report	 SLA/SLO + Monitoring/Alerting + Tiered Customer Support	

1-10 employees: In the early stages of a company, the focus is on finding a product-market fit. The emphasis is primarily on delivery and experimentation. Utilizing existing free or low-cost tools, developers handle testing and deployment. They also pay close attention to customer feedback and reports.

10-100 employees: Once the product-market fit is found, companies strive to scale. They are able to invest more in quality for critical functionalities and can create rapid evolution processes, such as scheduled deployments and testing procedures. Companies also proactively establish customer support processes to handle customer issues and provide proactive alerts.

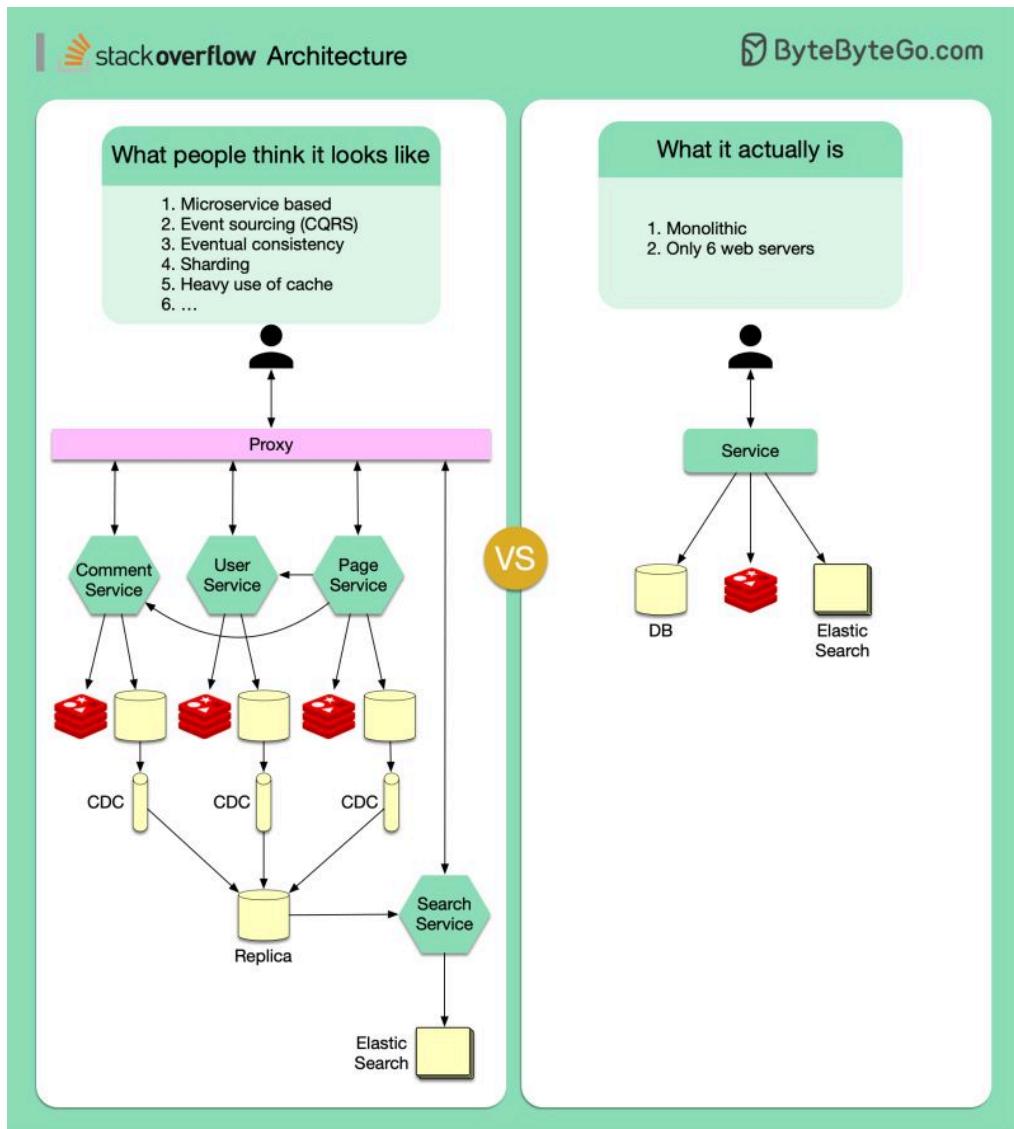
100-1,000 employees: When a company's go-to-market strategy proves successful, and the product scales and grows rapidly, it starts to optimize its engineering efficiency. More commercial tools can be purchased, such as Atlassian products. A certain level of standardization across tools is introduced, and automation comes into play.

1,000-10,000+ employees: Large tech companies build experimental tooling and automation to ensure quality and gather customer feedback at scale. Netflix, for example, is well known for its "Test in Production" strategy, which conducts everything through experiments.

Over to you: Every company is unique. What stage is your company currently at, and what tools do you use?

Stack Overflow's Architecture: A Very Interesting Case Study

Stack Overflow is a multi-tenant monolithic application serving 2 billion monthly page views across 200 sites.



It's on-prem, with only 9 IIS web servers.

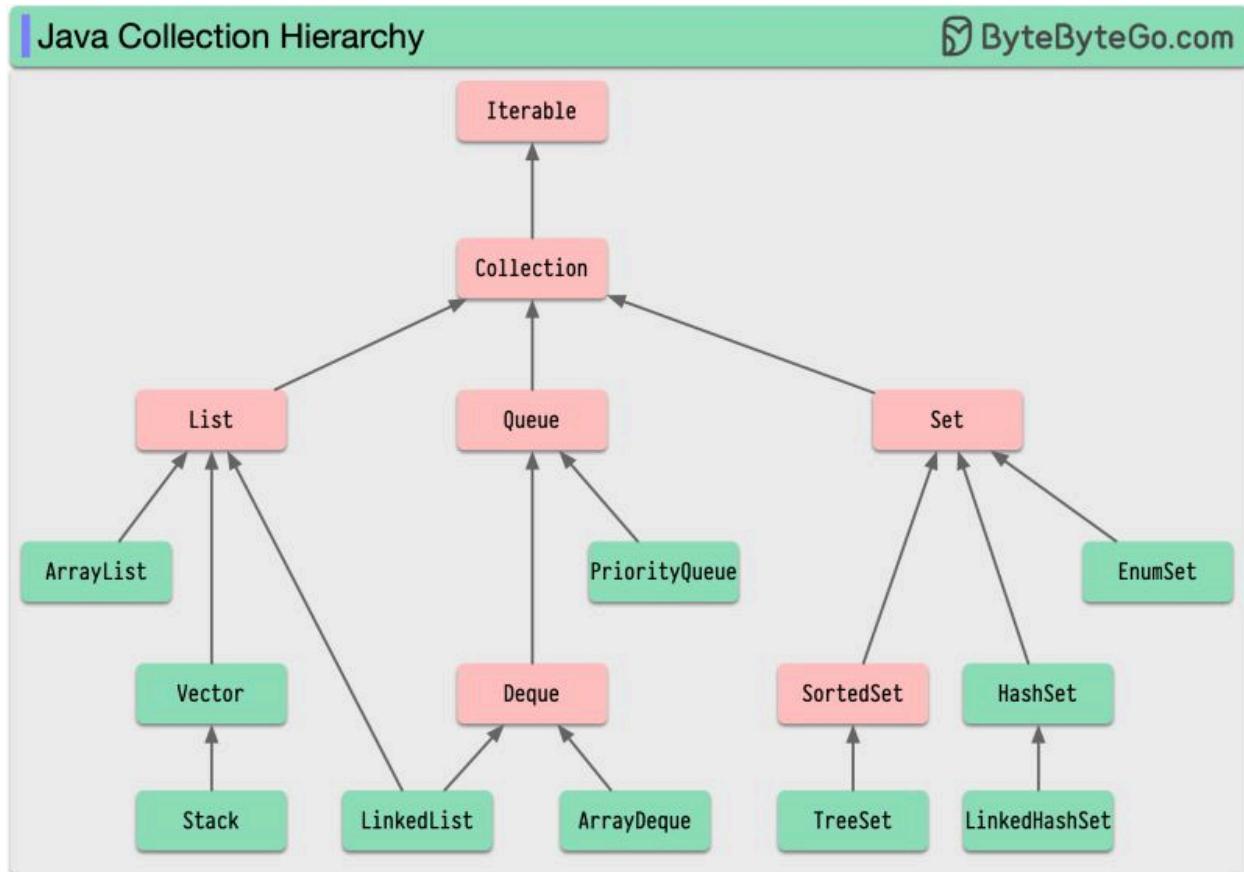
SQL Server has 1.5TB of RAM with no caching layer.

We conducted an in-depth research on this topic.

Watch and subscribe here: <https://lnkd.in/eSPvVrXz>

Are you familiar with the Java Collection Framework?

Every Java engineer has encountered the Java Collections Framework (JCF) at some point in their career. It has enabled us to solve complex problems in an efficient and standardized manner.



JCF is built upon a set of interfaces that define the basic operations for common data structures such as lists, sets, and maps. Each data structure is implemented by several concrete classes, which provide specific functionality.

Java Collections are based on the **Collection** interface. A collection class should support basic operations such as adding, removing, and querying elements. Through the enhanced for-loop or iterators, the **Collection** interface extends the **Iterable** interface, making it convenient to iterate over the elements.

The **Collection** interface has three main subinterfaces: **List**, **Set**, and **Queue**. Each of these interfaces has its unique characteristics and use cases.

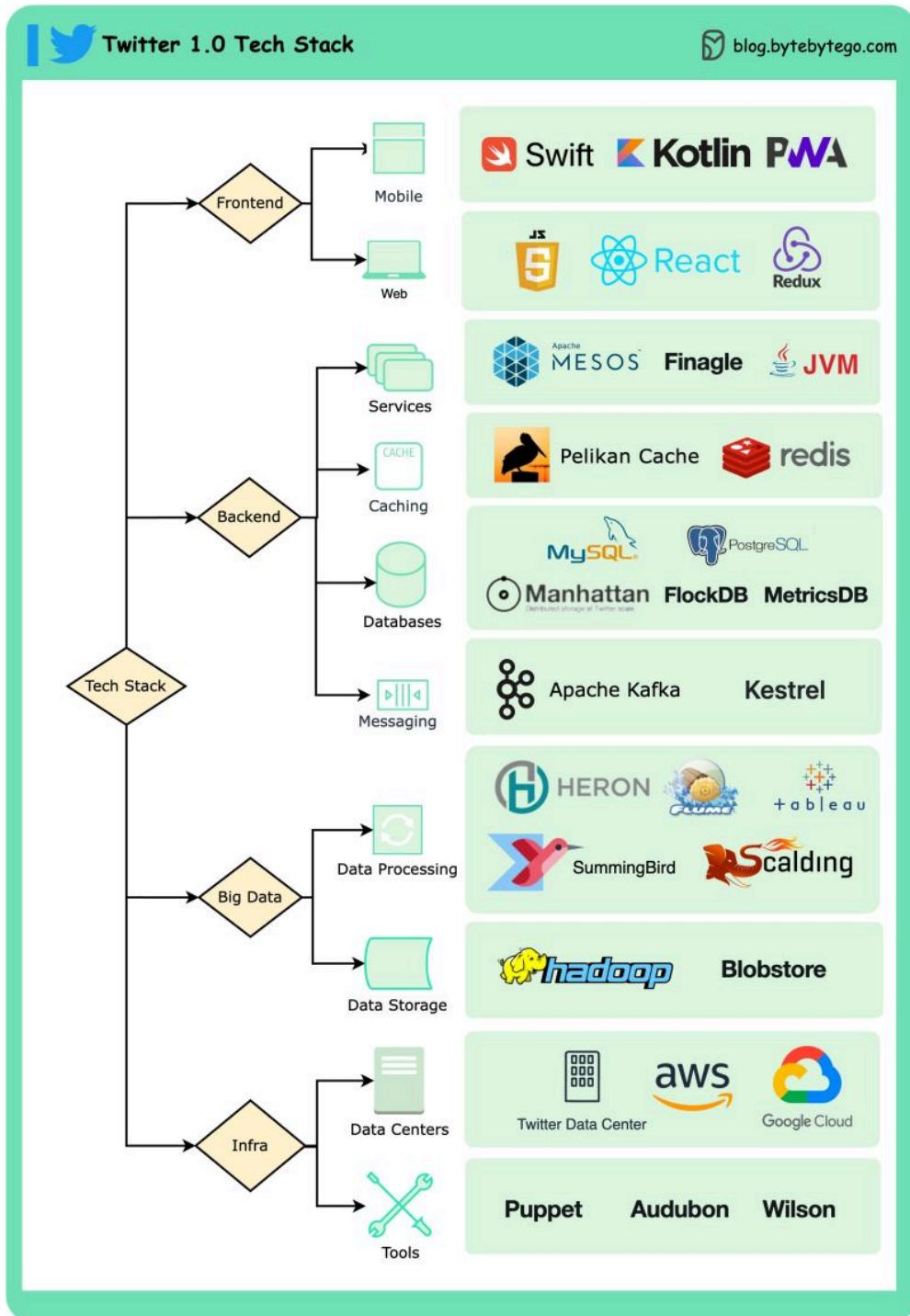
Java engineers need to be familiar with the Java Collection hierarchy to make informed decisions when choosing the right data structure for a particular problem. We can write more

efficient and maintainable code by familiarizing ourselves with the key interfaces and their implementations. We will undoubtedly benefit from mastering the JCF as it is a versatile and powerful tool in our Java arsenal

Over to you: You may noticed that Map did not appear in the picture. Do you know why?

Twitter 1.0 Tech Stack

This post is based on research from many Twitter engineering blogs and open-source projects. If you come across any inaccuracies, please feel free to inform us.



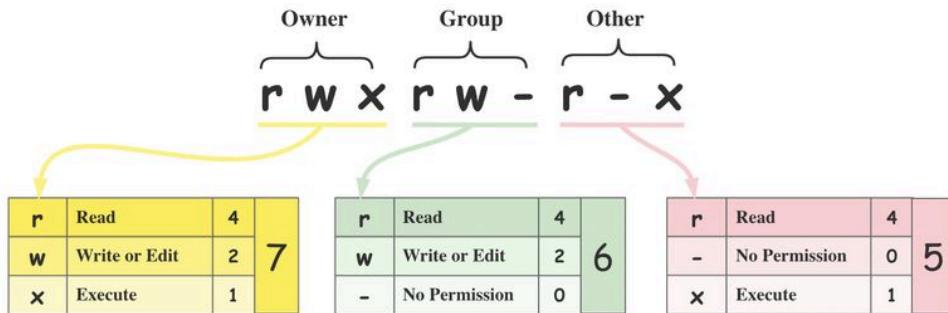
Mobile: Swift, Kotlin, PWA
Web: JS, React, Redux
Services: Mesos, Finagle
Caching: Pelikan Cache, Redis
Databases: Manhattan, MySQL, PostgreSQL, FlockDB, MetricsDB
Message queues: Kafka, Kestrel
Data processing: Heron, Flume, Tableau, SummingBird, Scalding
Data storage: Hadoop, blob store
Data centers: Twitter data center, AWS, Google Cloud
Tools: Puppet, Audubon, Wilson

Linux file permission illustrated

Linux File Permissions

 blog.bytebytego.com

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



Ownership

Every file or directory is assigned 3 types of owner:

- Owner: the owner is the user who created the file or directory.
- Group: a group can have multiple users. All users in the group have the same permissions to access the file or directory.
- Other: other means those users who are not owners or members of the group.

Permission

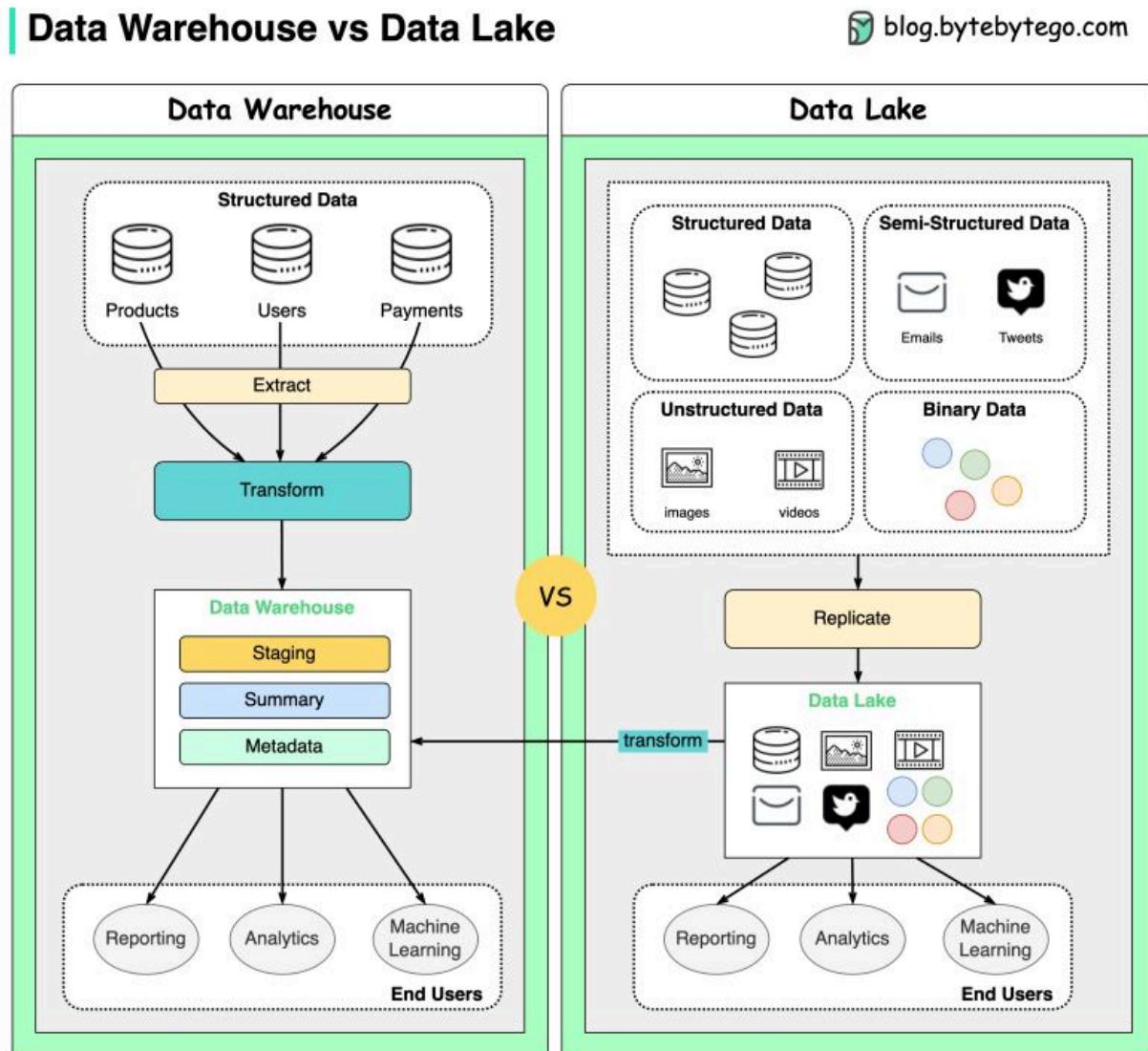
There are only three types of permissions for a file or directory.

- Read (r): the read permission allows the user to read a file.
- Write (w): the write permission allows the user to change the content of the file.
- Execute (x): the execute permission allows a file to be executed.

Over to you: chmod 777, good idea?

What are the differences between a data warehouse and a data lake?

The diagram below shows their comparison.



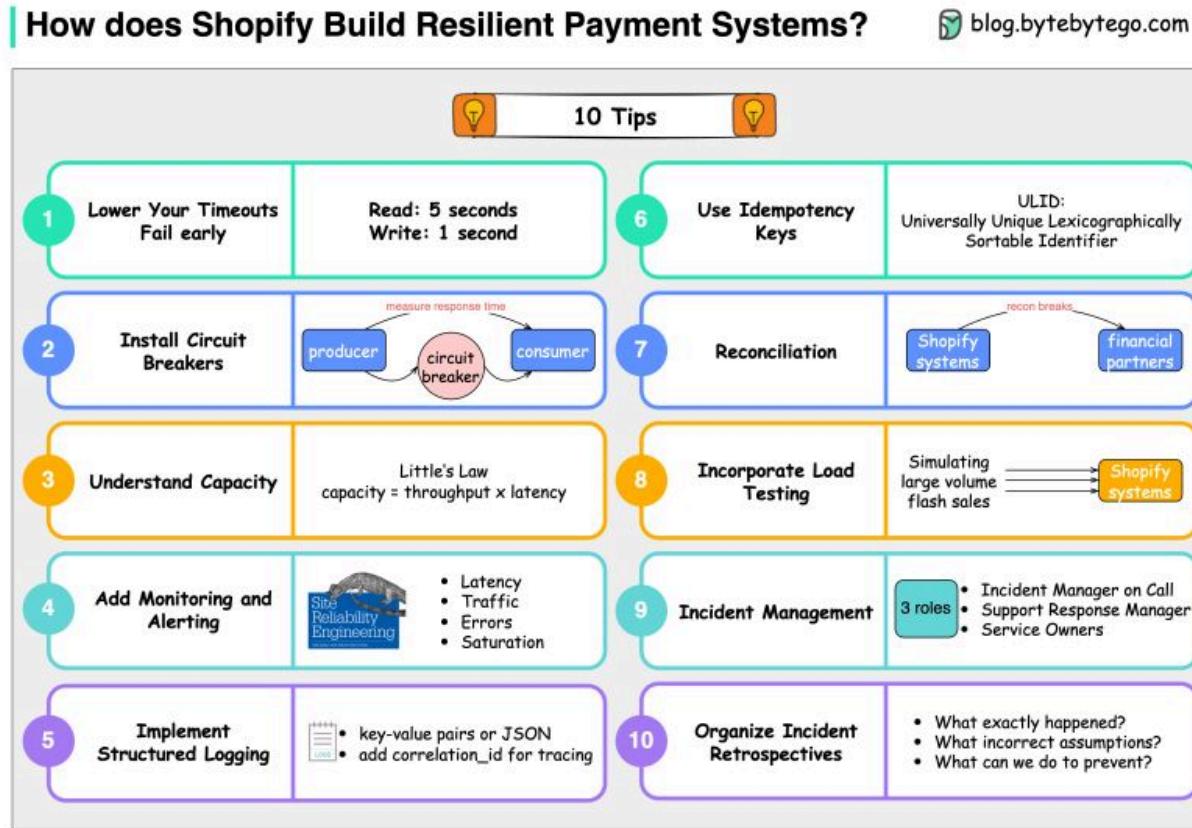
- A data warehouse processes structured data, while a data lake processes structured, semi-structured, unstructured, and raw binary data.
- A data warehouse leverages a database to store layers of structured data, which can be expensive. A data lake stores data in low-cost devices.
- A data warehouse performs Extract-Transform-Load (ETL) on data. A data lake performs Extract-Load-Transform (ELT).

- A data warehouse is schema-on-write, which means the data is already prepared when written into the data warehouse. A data lake is schema-on-read, so the data is stored as it is. The data can then be transformed and stored in a data warehouse for consumption.

Over to you: Do you use a data warehouse or a data lake to retrieve data?

10 principles for building resilient payment systems (by Shopify).

Shopify has some precious tips for building resilient payment systems.



1. Lower the timeouts, and let the service fail early
The default timeout is 60 seconds. Based on Shopify's experiences, read timeout of 5 seconds and write timeout of 1 second are decent setups.
2. Install circuit breaks
Shopify developed Semian to protect Net::HTTP, MySQL, Redis, and gRPC services with a circuit breaker in Ruby.
3. Capacity management
If we have 50 requests arrive in our queue and it takes an average of 100 milliseconds to process a request, our throughput is 500 requests per second.
4. Add monitoring and alerting
Google's site reliability engineering (SRE) book lists four golden signals a user-facing system should be monitored for: latency, traffic, errors, and saturation.
5. Implement structured logging
We store logs in a centralized place and make them easily searchable.

6. Use idempotency keys
Use Universally Unique Lexicographically Sortable Identifier (ULID) for these idempotency keys instead of a random version 4 UUID.
7. Be consistent with reconciliation
Store the reconciliation breaks with Shopify's financial partners in the database.
8. Incorporate load testing
Shopify regularly simulates the large volume flash sales to get the benchmark results.
9. Get on top of incident management
Each incident channel has 3 roles: Incident Manager on Call (IMOC), Support Response Manager (SRM), and service owners.
10. Organize incident retrospectives
For each incident, 3 questions are asked at Shopify: What exactly happened? What incorrect assumptions did we hold about our systems? What we can do to prevent this from happening?

Reference: [10 Tips for Building Resilient Payment Systems](#)

Kubernetes Periodic Table

A comprehensive visual guide that demystifies the key building blocks of this powerful container orchestration platform.

The Kubernetes Periodic Table is a visual representation of 120 components organized into 18 categories. The categories are color-coded and include:

- Infrastructure and Control Plane** (Blue): No, Op, Cl, Cp, Kt, As, Et, Kc.
- Core Components** (Green): Sv, Pd, Dp, Sc, Ds, Sh, Cm, Ru, Mls, Esg, Pep, Pms, La, Sm, Is, Pmo, Pr, Km, Csd, Ns, Pv, Cj, Lt, Lp, Cbs, Tm, Caw, Azw, Ev, Ot, Smp, Ag, Ed, Ch, Sp, Vsc.
- Configuration and Data Management** (Red): Cfm, St, Ct, Nm, Cb, Cls, Cma, Crv, Ms, Lg, Al, Pe, Pm, Fw, Sct, Crd, Dp, Sc, Ds, Sh, Cm, Ru, Mls, Esg, Pep, Pms, La, Sm, Is, Pmo, Pr, Km, Csd, Rq, Rls, Rm, Ra, Pa, Paa, Na, Nsr, Tt, Pdb, Gc, Hm, Ku, Of, Cs.
- Other Elements** (Purple): Monitoring and Observability, Networking, Stateful Applications and Data Management.
- Backup, Restore, and Disaster Recovery** (Teal): Br, Rb, Psp, Rbm, Iam.
- Security and Identity Management** (Orange): RbAC, RBAC Manager, Identity and Access Management.
- Governance and Compliance** (Yellow): Continuous Integration and Deployment, Autoscaling and Load Balancing, Resource Management, Package Management and Configuration.
- Networking** (Light Green): Br, Rb, Psp, Rbm, Iam.
- Stateful Applications and Data Management** (Pink): Br, Rb, Psp, Rbm, Iam.
- Monitoring and Observability** (Light Blue): Br, Rb, Psp, Rbm, Iam.
- Resource Management** (Light Green): Br, Rb, Psp, Rbm, Iam.
- Package Management and Configuration** (Light Blue): Br, Rb, Psp, Rbm, Iam.
- Continuous Integration and Deployment** (Dark Blue): Br, Rb, Psp, Rbm, Iam.
- Autoscaling and Load Balancing** (Dark Orange): Br, Rb, Psp, Rbm, Iam.
- Resource Management** (Dark Green): Br, Rb, Psp, Rbm, Iam.
- Package Management and Configuration** (Dark Blue): Br, Rb, Psp, Rbm, Iam.

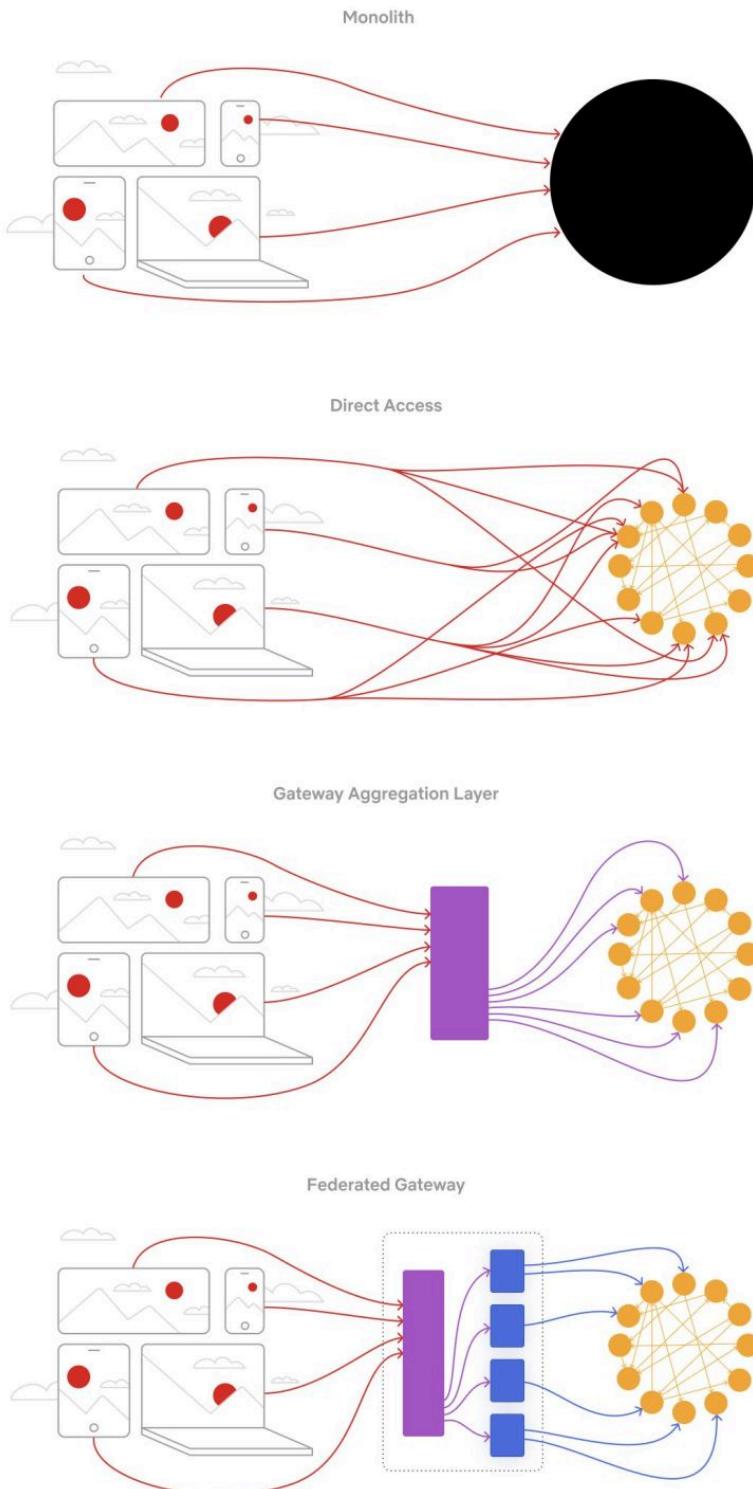
The table also includes numbered slots from 1 to 120, corresponding to specific components like Ci, Cd, Go, Cnd, Bg, Rud, Ca, Goo, Cat, Bp, Nap, Hpa, Vpa, Cas, Lb, Rq, Rls, Rm, Ra, Pa, Paa, Na, Nsr, Tt, Pdb, Gc, Hm, Ku, Of, and Cs.

This Kubernetes Periodic Table sheds light on the 120 crucial components that make up the Kubernetes ecosystem.

Whether you're a developer, system administrator, or cloud enthusiast, this handy resource will help you navigate the complex Kubernetes landscape.

Evolution of the Netflix API Architecture

Evolution of an API Architecture



The Netflix API architecture went through 4 main stages.

- Monolith
- Direct access
- Gateway aggregation layer
- Federated gateway

We explain the evolution in a 4-minute video. Watch and subscribe here:

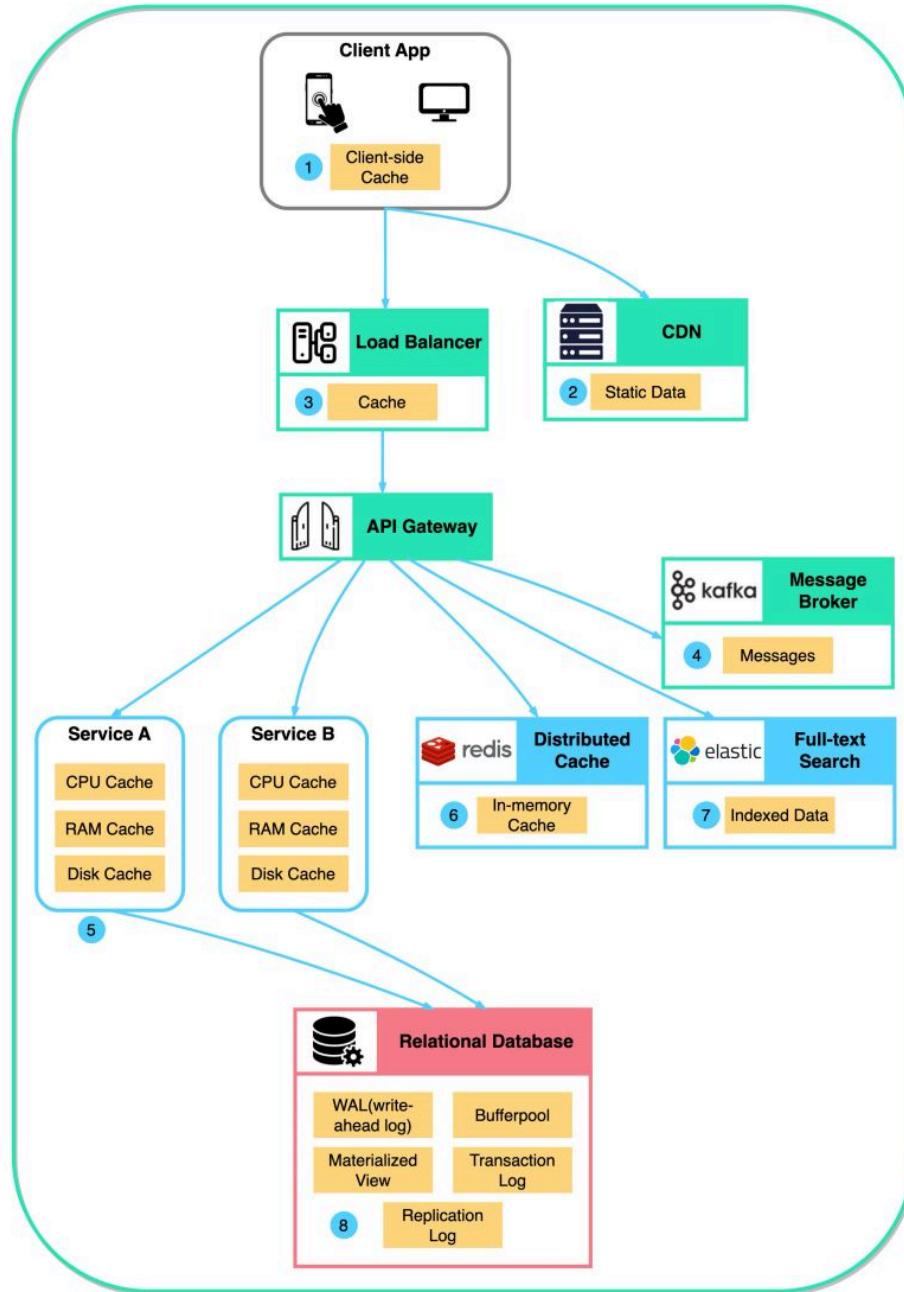
<https://lnkd.in/e9yycpU6>

Where do we cache data?

Data is cached everywhere, from the front end to the back end!

This diagram illustrates where we cache data in a typical architecture.

Where do We Cache Data?  blog.bytebytogo.com



There are multiple layers along the flow.

Client apps: HTTP responses can be cached by the browser. We request data over HTTP for the first time, and it is returned with an expiry policy in the HTTP header; we request data again, and the client app tries to retrieve the data from the browser cache first.

CDN: CDN caches static web resources. The clients can retrieve data from a CDN node nearby.

Load Balancer: The load Balancer can cache resources as well.

Messaging infra: Message brokers store messages on disk first, and then consumers retrieve them at their own pace. Depending on the retention policy, the data is cached in Kafka clusters for a period of time.

Services: There are multiple layers of cache in a service. If the data is not cached in the CPU cache, the service will try to retrieve the data from memory. Sometimes the service has a second-level cache to store data on disk.

Distributed Cache: Distributed cache like Redis hold key-value pairs for multiple services in memory. It provides much better read/write performance than the database.

Full-text Search: we sometimes need to use full-text searches like Elastic Search for document search or log search. A copy of data is indexed in the search engine as well.

Database: Even in the database, we have different levels of caches:

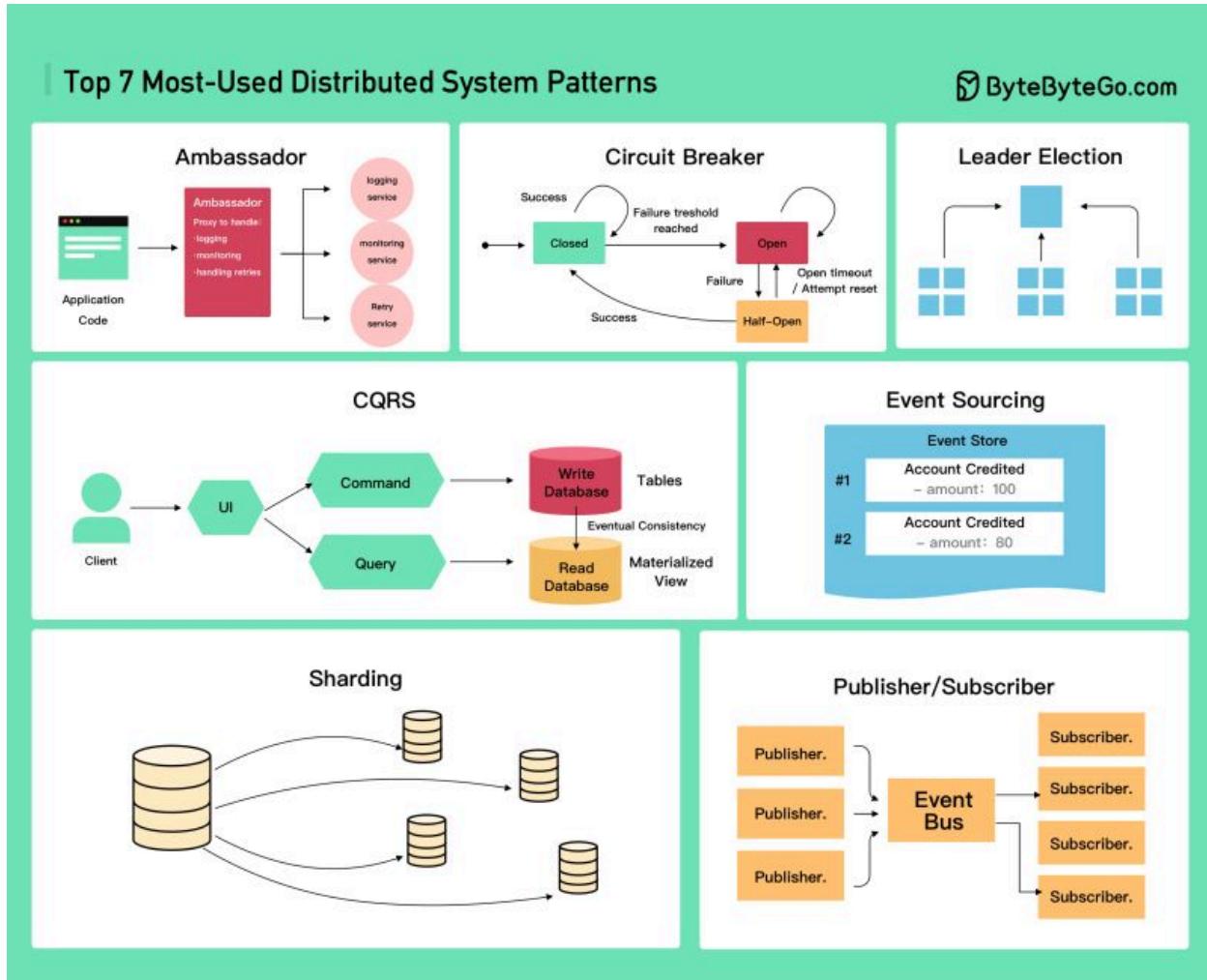
- WAL(Write-ahead Log): data is written to WAL first before building the B tree index
- Bufferpool: A memory area allocated to cache query results
- Materialized View: Pre-compute query results and store them in the database tables for better query performance

Transaction log: record all the transactions and database updates

Replication Log: used to record the replication state in a database cluster

Over to you: With the data cached at so many levels, how can we guarantee the sensitive user data is completely erased from the systems?

Top 7 Most-Used Distributed System Patterns ↓



- Ambassador
- Circuit Breaker
- CQRS
- Event Sourcing
- Leader Election
- Publisher/Subscriber
- Sharding

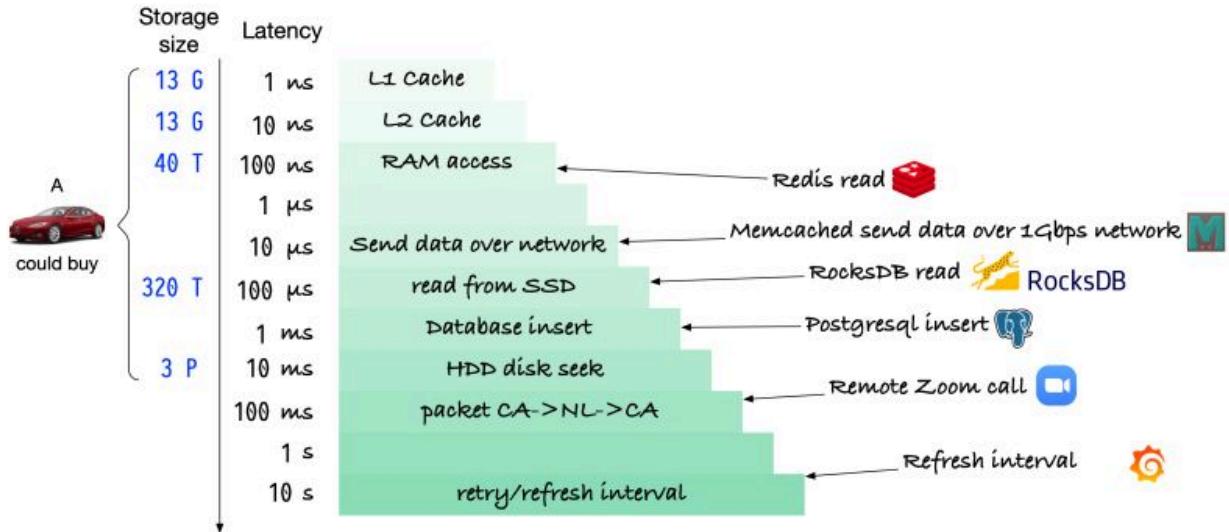
Which additional patterns have we overlooked?

How much storage could one purchase with the price of a Tesla Model S? ↓

There's a trade-off between the price of a storage system and its access latency. Naturally, one might wonder how much storage could be obtained if one is willing to sacrifice latency.

1 Tesla Model S = How much storage?

ByteByteGo.com



To make this calculation more intriguing, let's use the price of a Tesla Model S as a benchmark. Here are the relevant prices:

- Tesla Model S: \$87,490 per car
- L1 cache: \$7 per megabyte
- L2 cache: \$7 per megabyte
- RAM: \$70 for 32G
- SSD: \$35 for 128G
- HDD: \$350 for 12T

How to choose between RPC and RESTful?

RPC vs. RESTful



	RPC	RESTful
		
Coupling	Strong coupling	Weak coupling
Data format	Binary thrift, protobuf, Avro	Text XML, JSON
Communication protocol	TCP	HTTP/1.1, HTTP/2
Performance	High	Lower than RPC
Interface definition language (IDL)	thrift, protobuf	Swagger
Client code generation	Auto-generated stub	Auto-generated stub
Language framework	gRPC, thrift	SpringMVC, JAX-RS
Developer friendliness	not human readable hard to debug	human readable easy to debug

Communication between different software systems can be established using either RPC (Remote Procedure Call) or RESTful (Representational State Transfer) protocols, which allow multiple systems to work together in distributed computing.

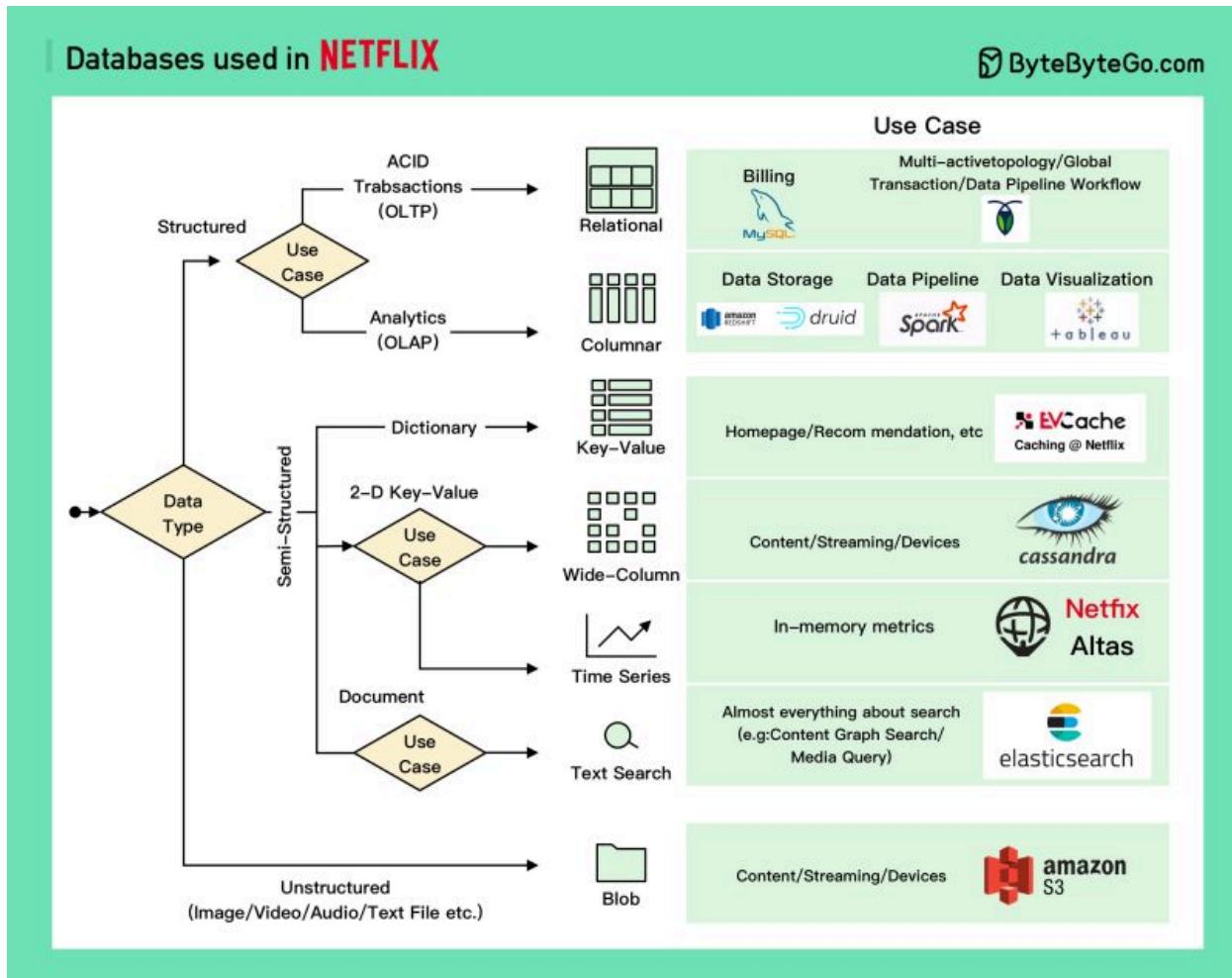
The two protocols differ mainly in their design philosophy. RPC enables calling remote procedures on a server as if they were local procedures, while RESTful applications are resource-based and interact with these resources via HTTP methods.

When choosing between RPC and RESTful, consider your application's needs. RPC might be a better fit if you require a more action-oriented approach with custom operations, while RESTful would be a better choice if you prefer a standardized, resource-based approach that utilizes HTTP methods.

Over to you: What are the best practices for versioning and ensuring backward compatibility of RPC and RESTful APIs?

Netflix Tech Stack - Databases

The Netflix Engineering team selects a variety of databases to empower streaming at scale.



Relational databases: Netflix chooses MySQL for billing transactions, subscriptions, taxes, and revenue. They also use CockroachDB to support a multi-region active-active architecture, global transactions, and data pipeline workflows.

Columnar databases: Netflix primarily uses them for analytics purposes. They utilize Redshift and Druid for structured data storage, Spark and data pipeline processing, and Tableau for data visualization.

Key-value databases: Netflix mainly uses EVCache built on top of Memcached. EVCache has been with Netflix for over 10 years and is used for most services, caching various data such as the Netflix Homepage and Personal Recommendations.

Wide-column databases: Cassandra is usually the default choice at Netflix. They use it for almost everything, including Video/Actor information, User Data, Device information, and Viewing History.

Time-series databases: Netflix built an open-source in-memory database called Atlas for metrics storage and aggregation.

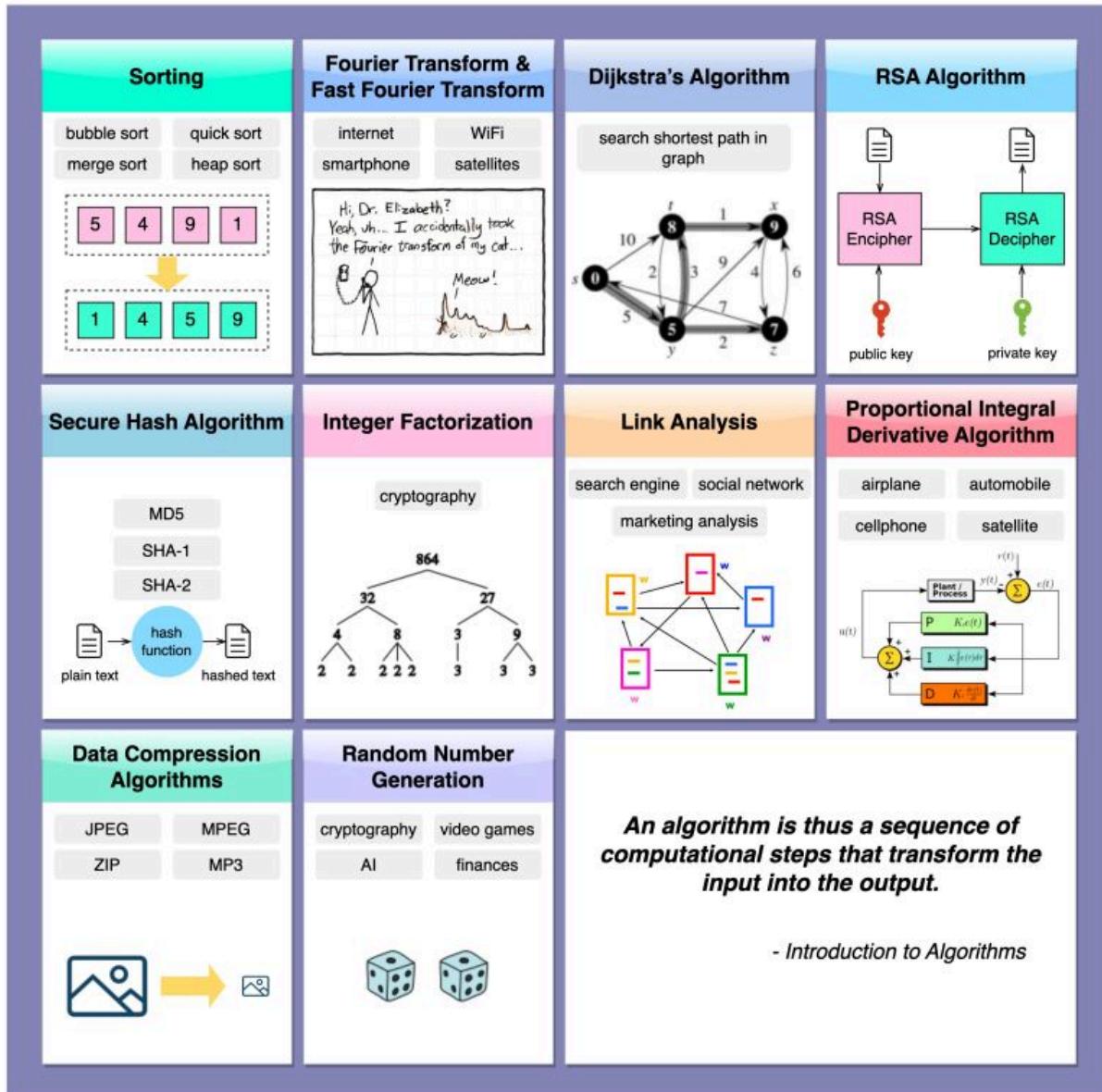
Unstructured data: S3 is the default choice and stores almost everything related to Image/Video/Metrics/Log files. Apache Iceberg is also used with S3 for big data storage.

If you work for a large company and wish to discuss your company's technology stack, feel free to get in touch with me. By default, all communications will be treated as anonymous.

The 10 Algorithms That Dominate Our World

The diagram below shows the most commonly used algorithms in our daily lives. They are used in internet search engines, social networks, WiFi, cell phones, and even satellites.

The 10 Algorithms That Dominate Our World blog.bytebytogo.com



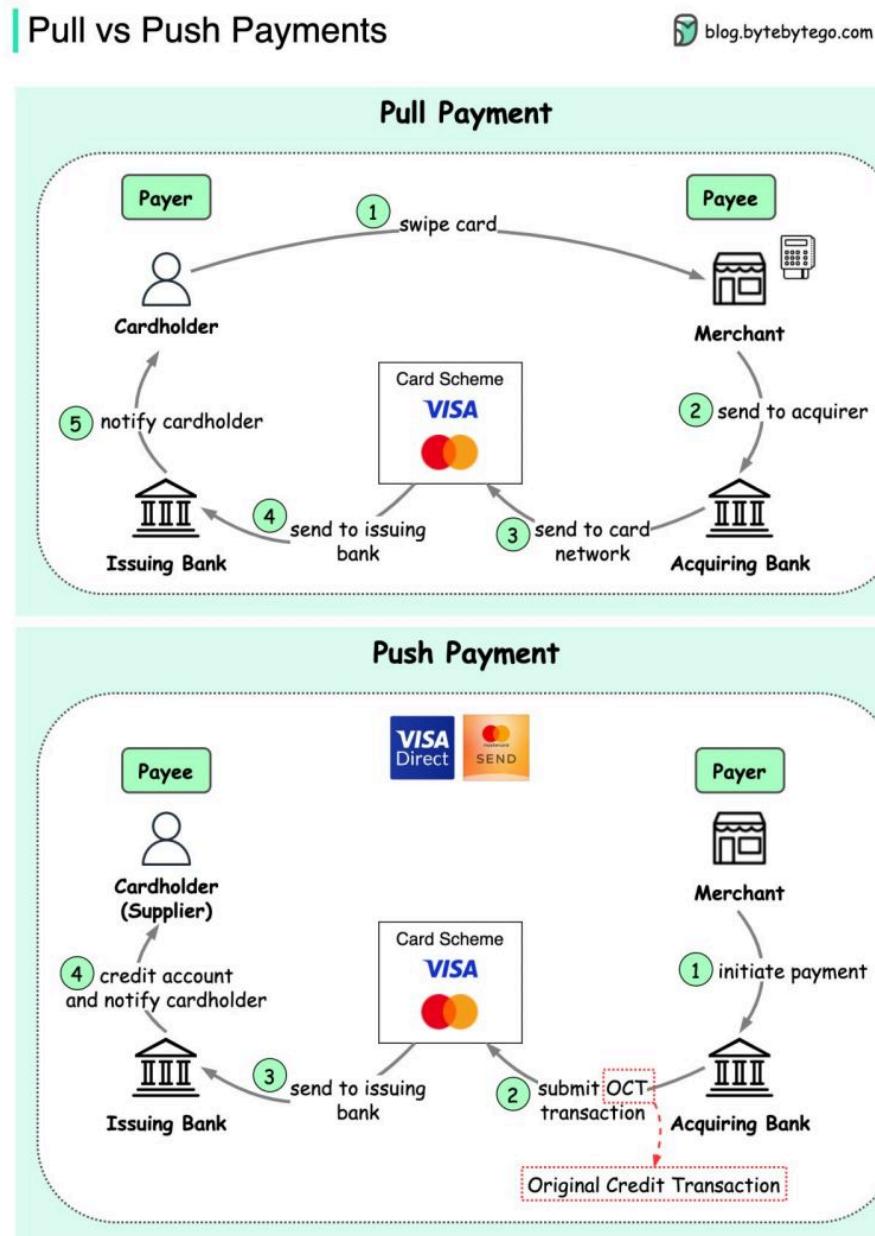
1. Sorting
2. Fourier Transform and Fast Fourier Transform
3. Dijkstra's algorithm

4. RSA algorithm
5. Secure Hash Algorithm
6. Integer factorization
7. Link Analysis
8. Proportional Integral Derivative Algorithm
9. Data compression algorithms
10. Random Number Generation

👉 Over to you: Any other commonly used algorithms?

What is the difference between “pull” and “push” payments?

The diagram below shows how the pull and push payments work.



- When we swipe a credit/debit card at a merchant, it is a pull payment, where the money is sent from the cardholder to the merchant. The merchant pulls money from the cardholder's account, and the cardholder approves the transaction.
- With Visa Direct or Mastercard Send, the push payments enable merchant, corporate, and government disbursements.

Step 1: The merchant initiates the push payment through a digital channel. It can be a mobile phone or a bank branch etc.

Step 2: The acquiring bank creates and submits an OCT (Original Credit Transaction) to the card scheme.

Step 3: The transaction is routed to the receiving institution.

Step 4: The issuing bank credits the cardholder's account and notifies the cardholder. The money is deposited into a Visa account that can be accessed at an ATM or PoS terminal or a digital wallet.

Note that the push payments work for cross-border transactions.

Push payments are indeed an interesting innovation, which complements the digital wallet strategy in Visa and Mastercard. The abstraction of "account" masks the complication of different funding or consuming channels.

Over to you: What is your most frequently used payment method? Is it pull-based or push-based?