2023 EDITION

FREE

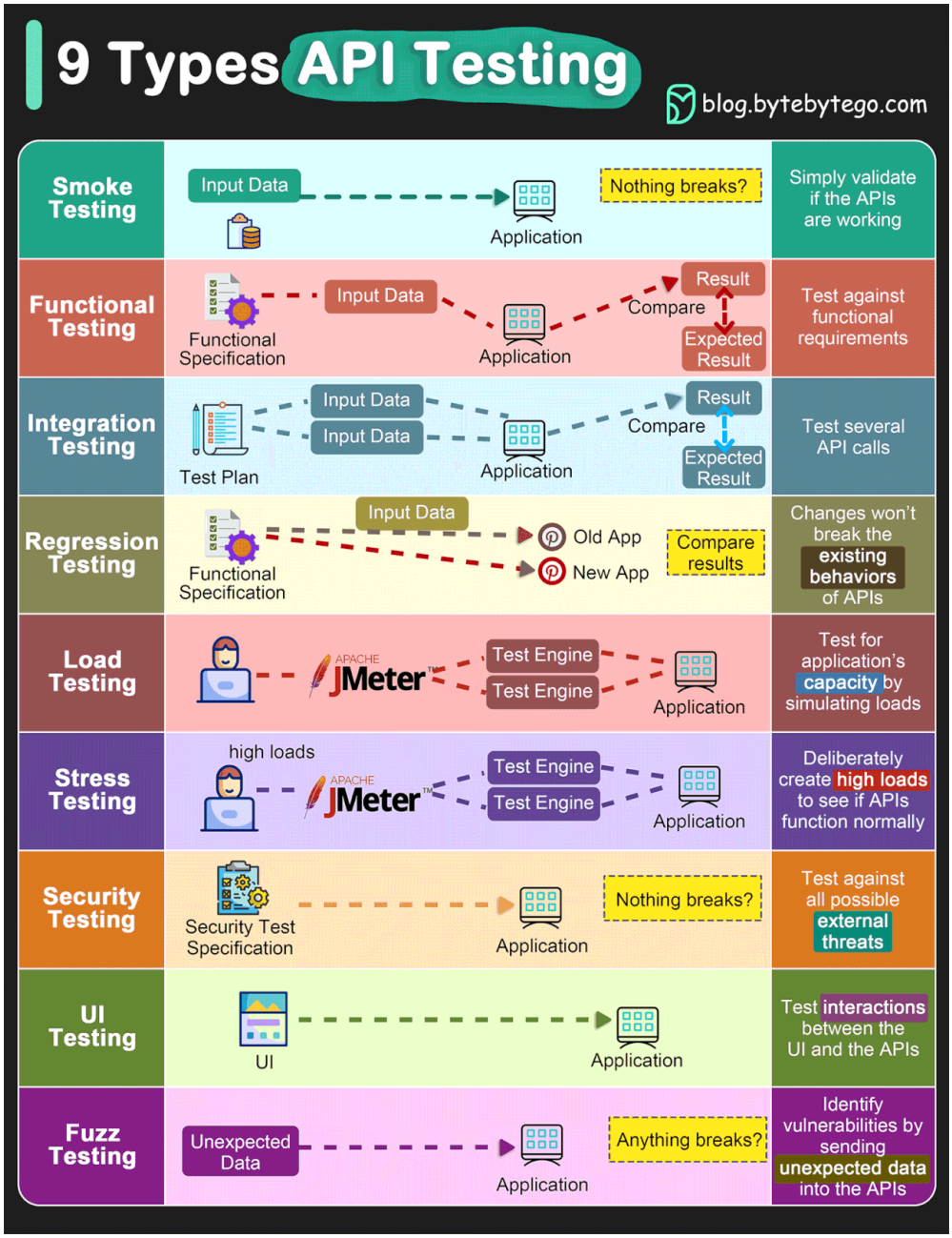
SYSTEM DESIGN

THE BIG ARCHIVE

ByteByteGo

By ByteByteGo.com

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.

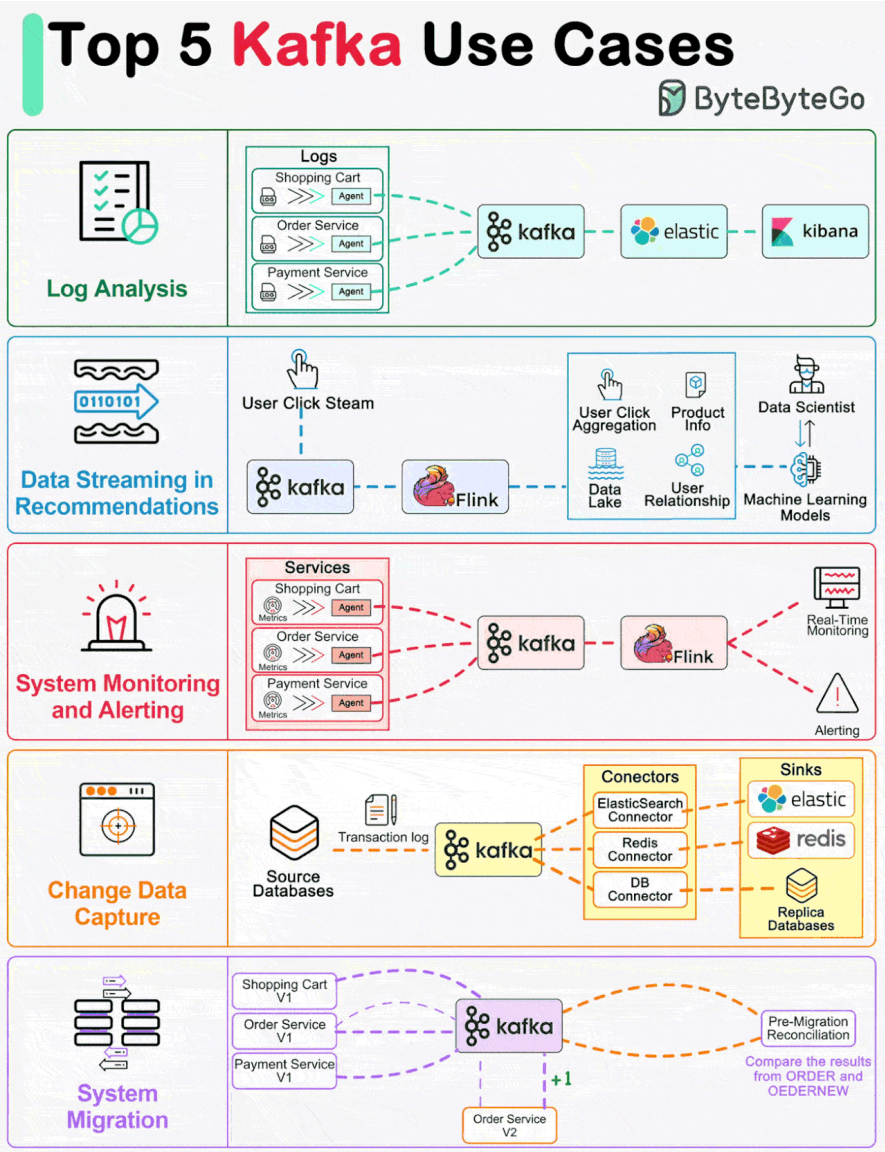
- Security Testing
  This tests the APIs against all possible external threats.

- 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.

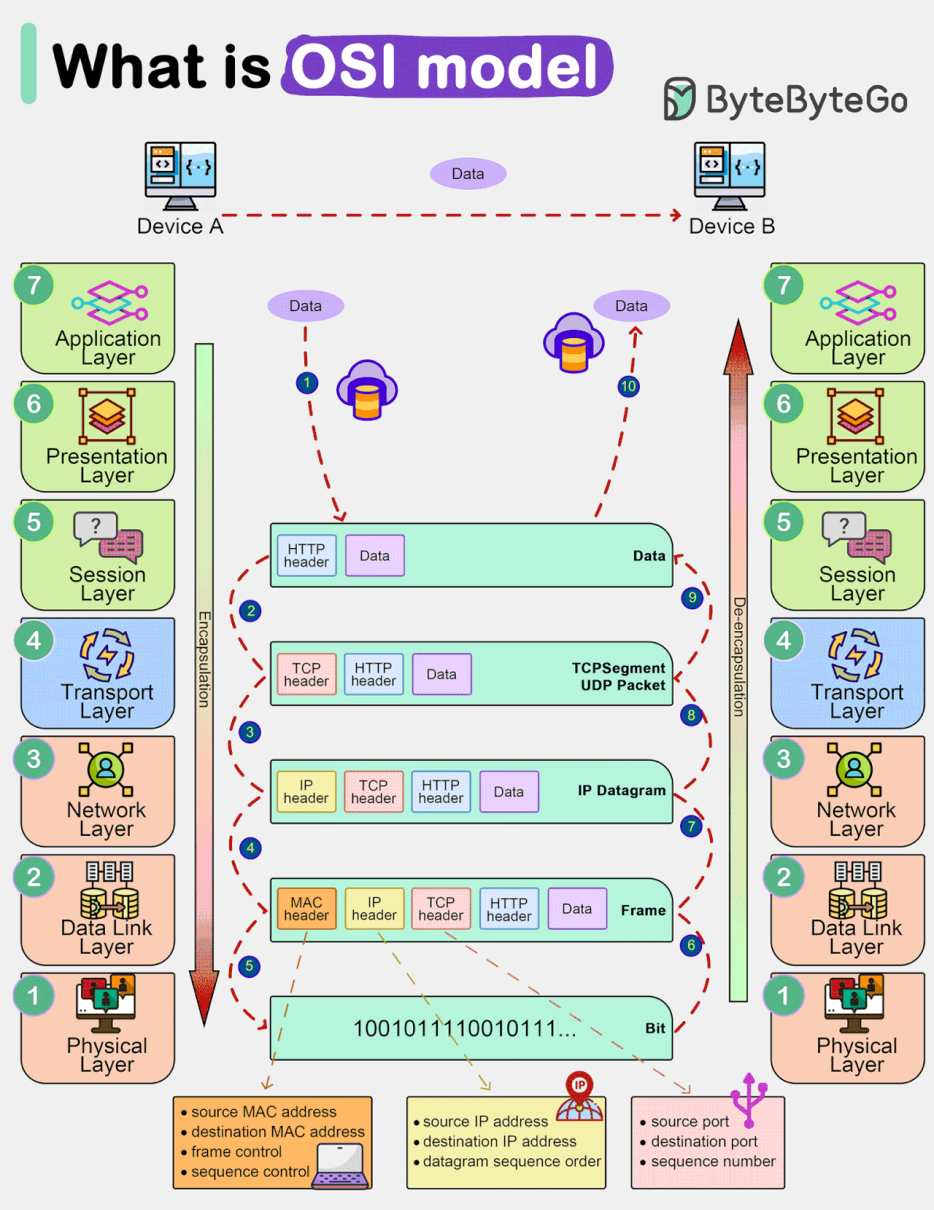


Let's review the popular Kafka use cases.

- 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?

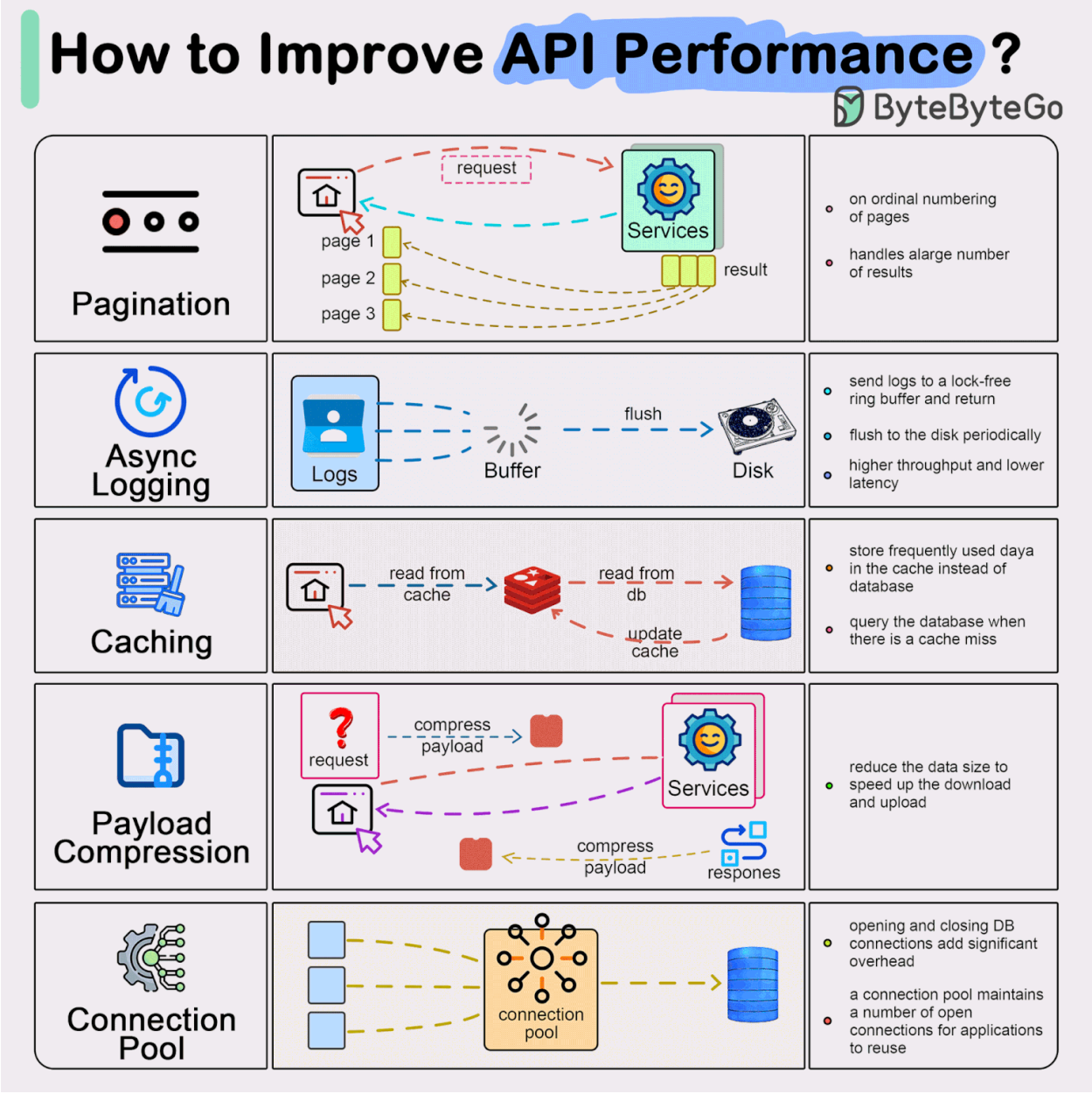
How is data sent over the internet? What does that have to do with the OSI model? How does TCP/IP fit into this?



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.

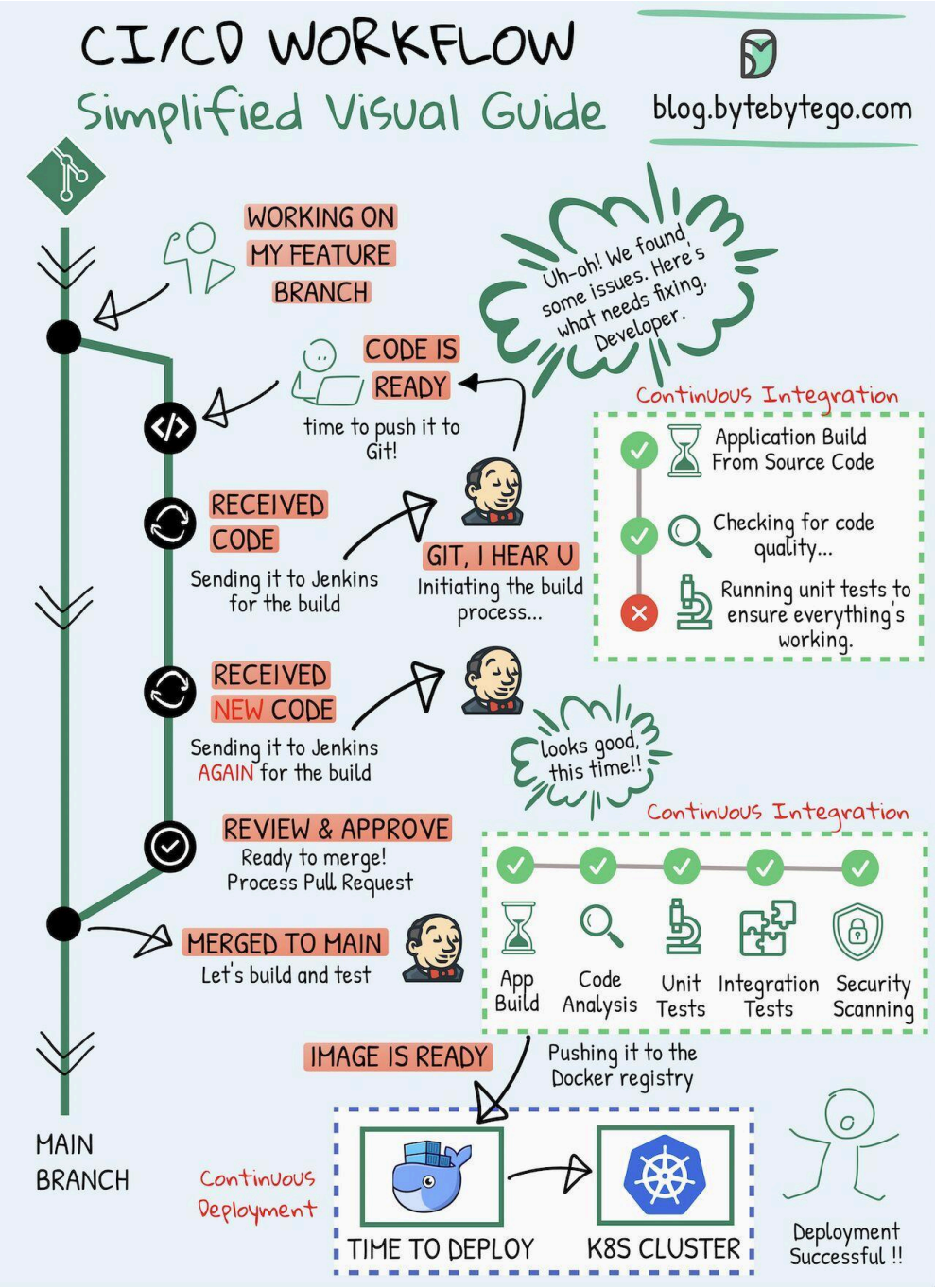
Payload Compression:
To reduce data transmission time, requests and responses can be compressed (e.g., using gzip), making the upload and download processes quicker.

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?

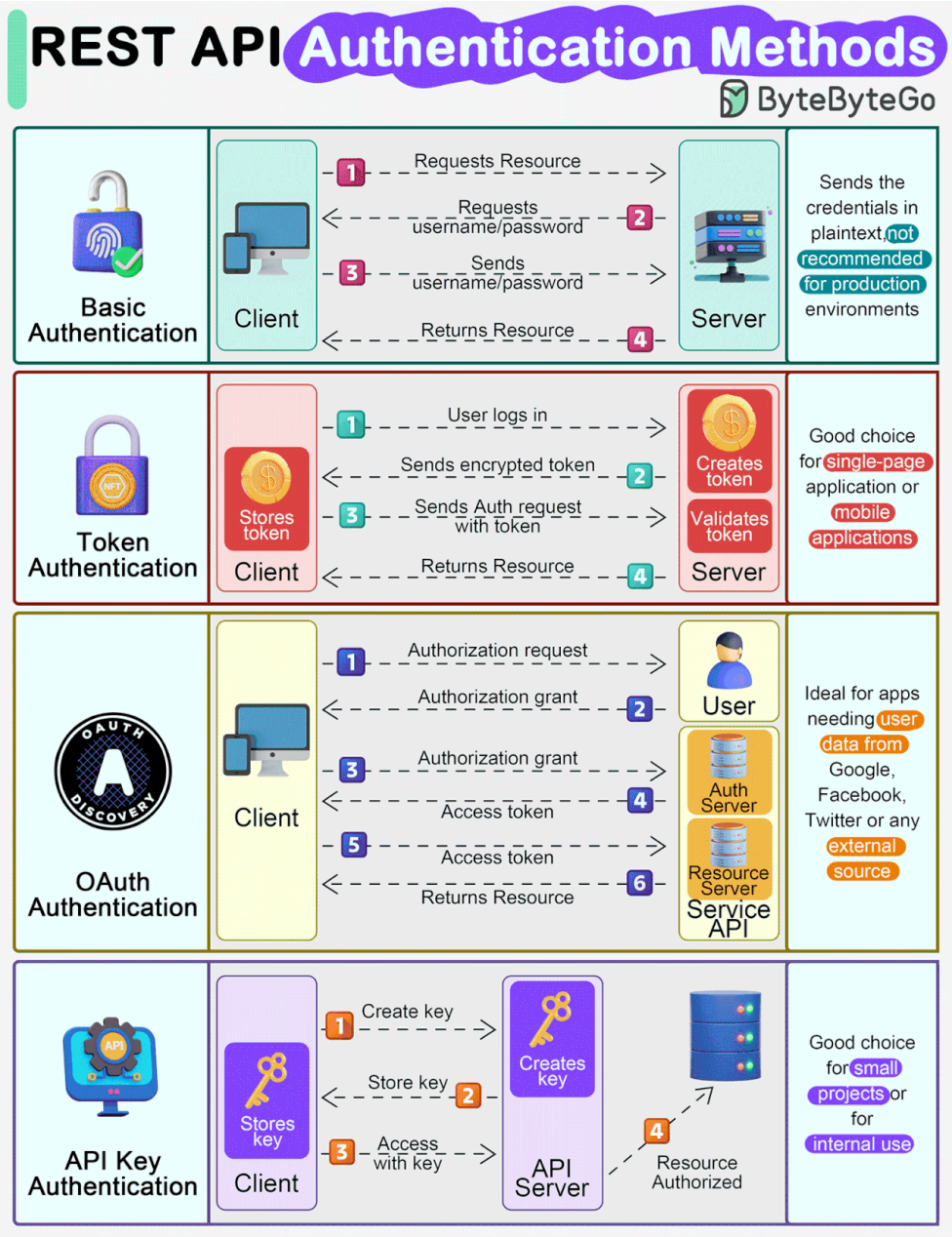
There are over 1,000 engineering blogs. Here are my top 9 favorites:

- 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.

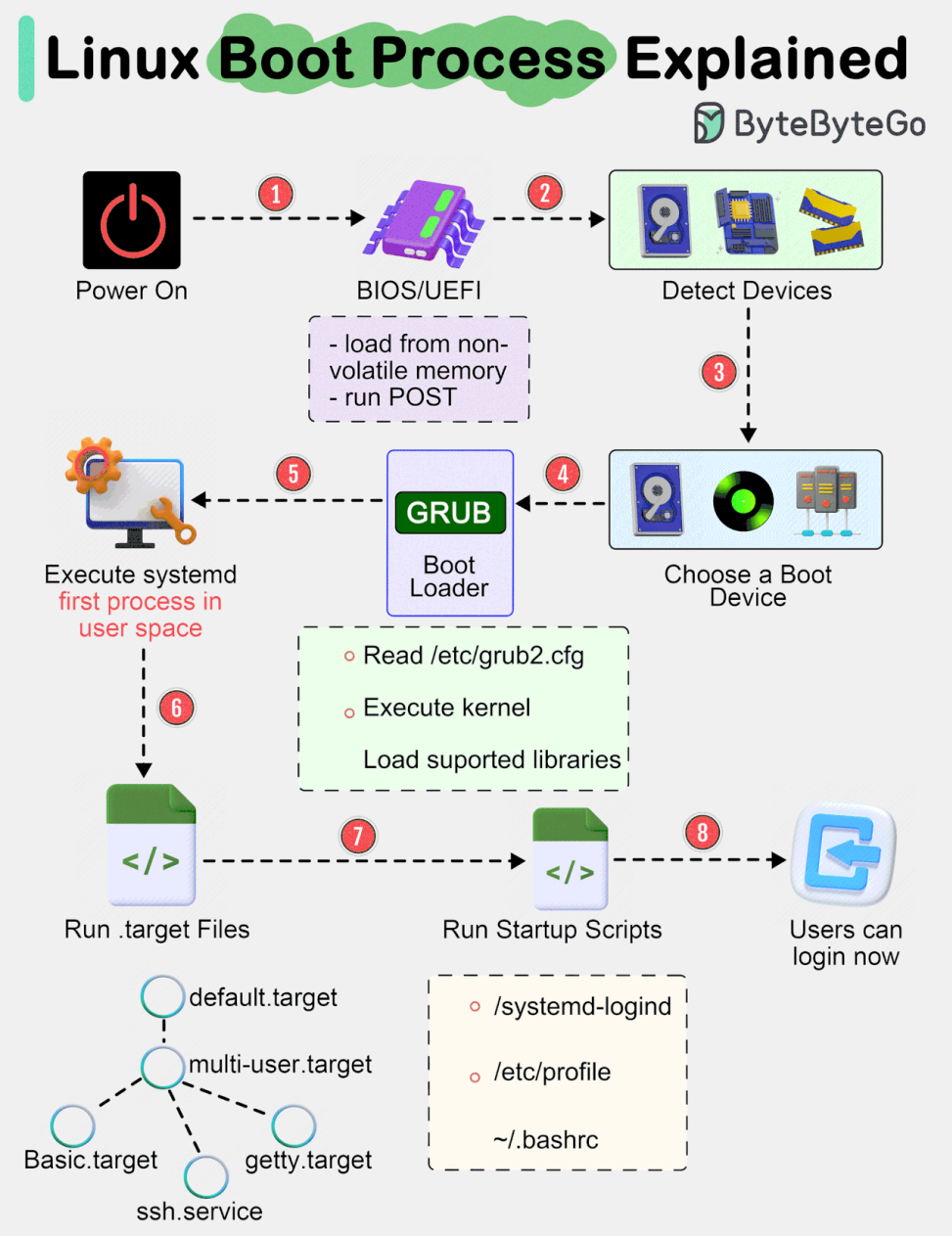
When to use:
Convenient for straightforward access control in less sensitive environments or for granting access to certain functionalities without the need for user-specific permissions.

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.

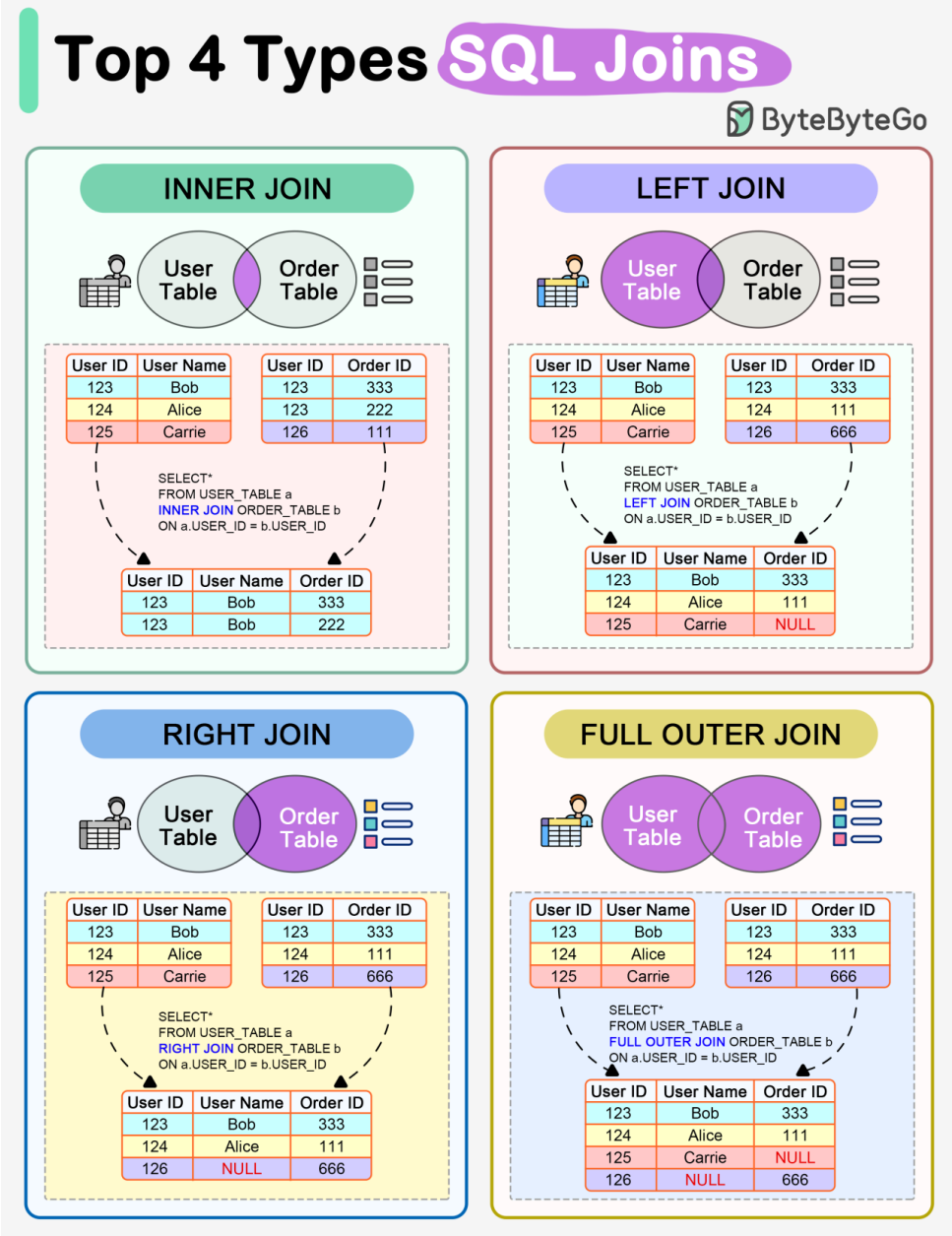
Step 7 - The system runs a set of startup scripts and configures the environment.

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.

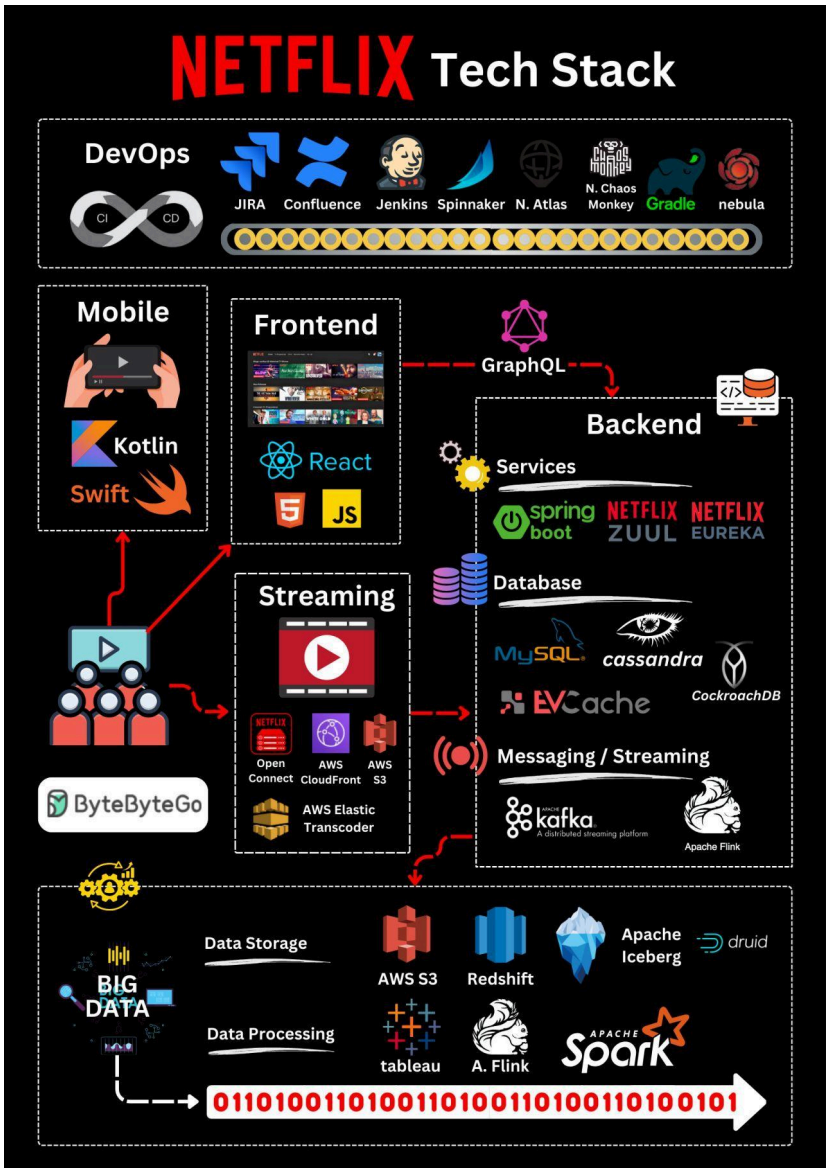
- LEFT JOIN
  Returns all records from the left table, and the matching records from the right table.

- 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.

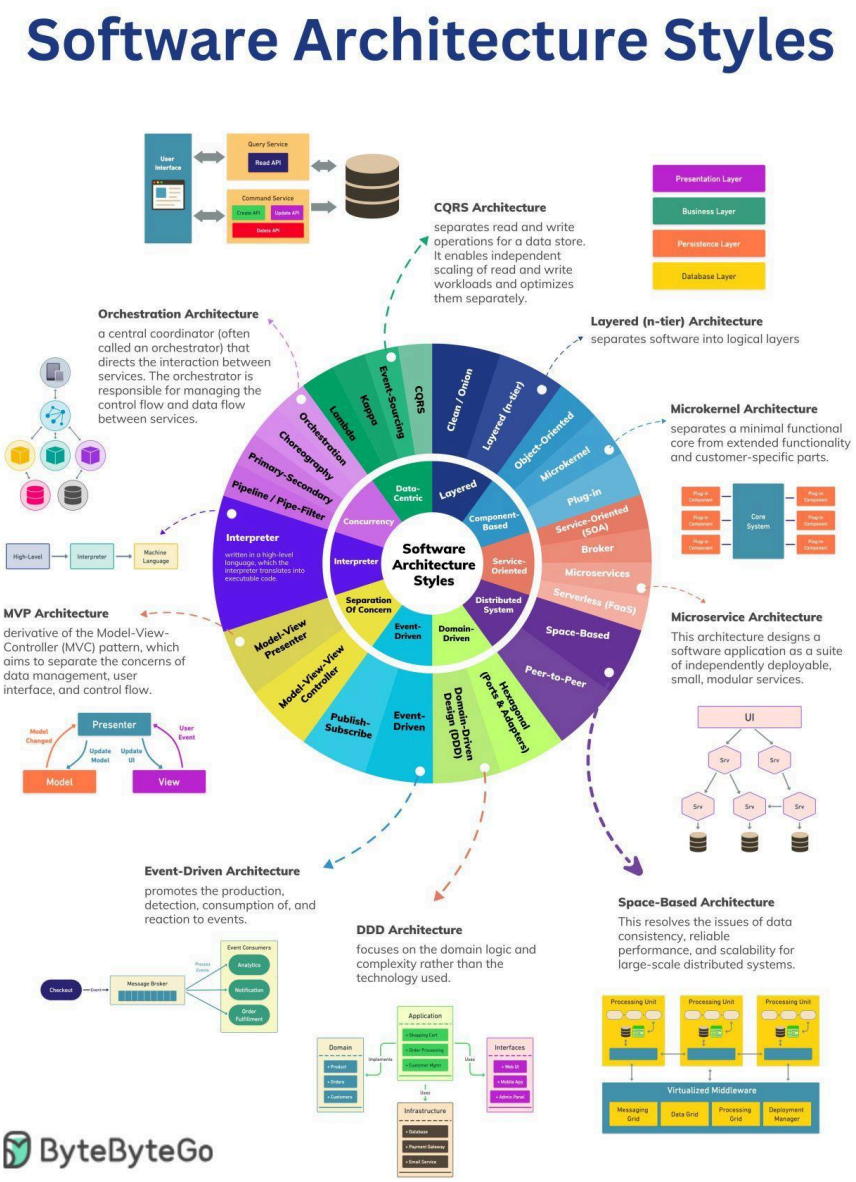
Messaging/streaming: Netflix employs Apache Kafka and Fink for messaging and streaming purposes.

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



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.

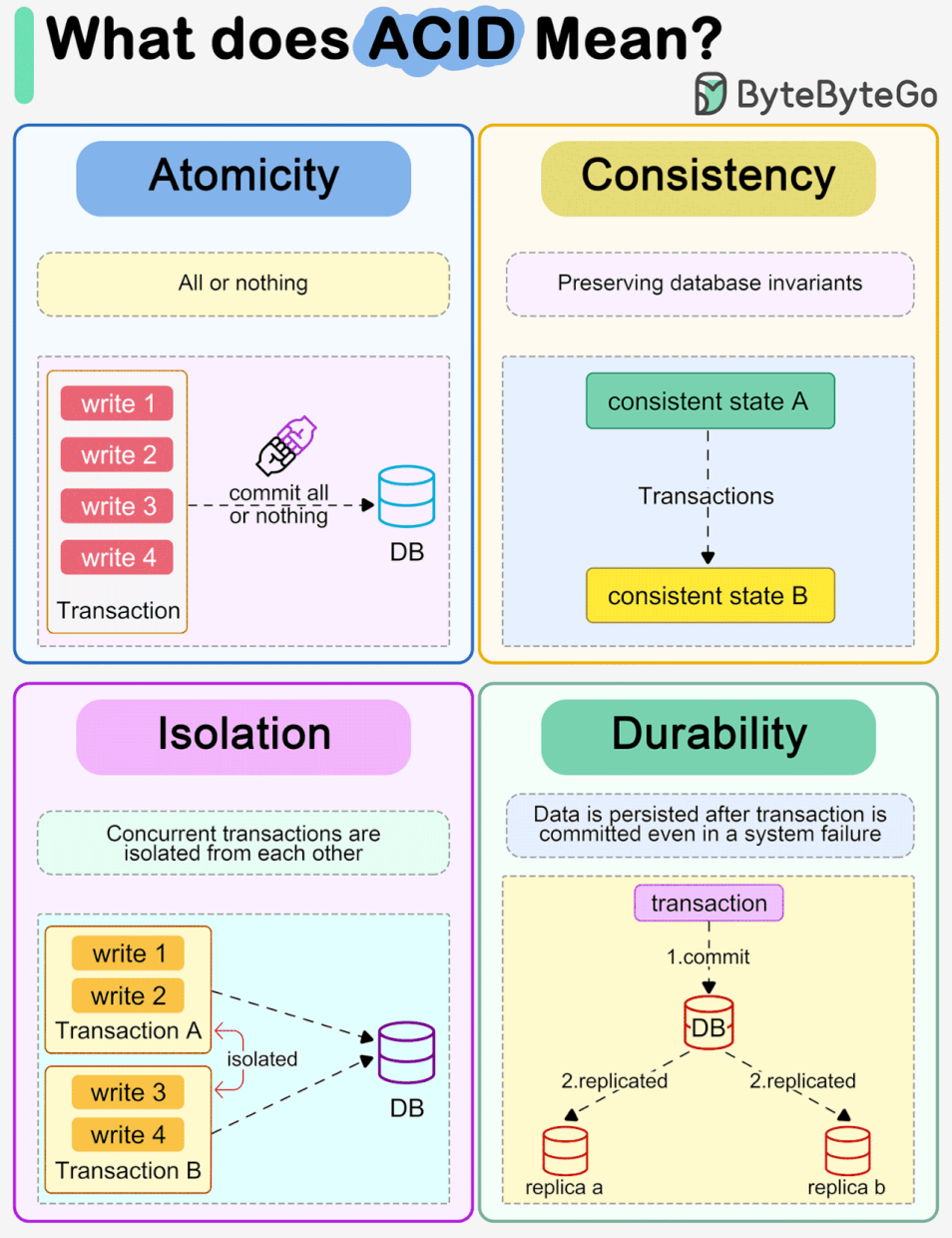
However, with the vast array of architectural styles and patterns available, it can take time to discern which approach best suits a particular project or system. Aims to shed light on these

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.

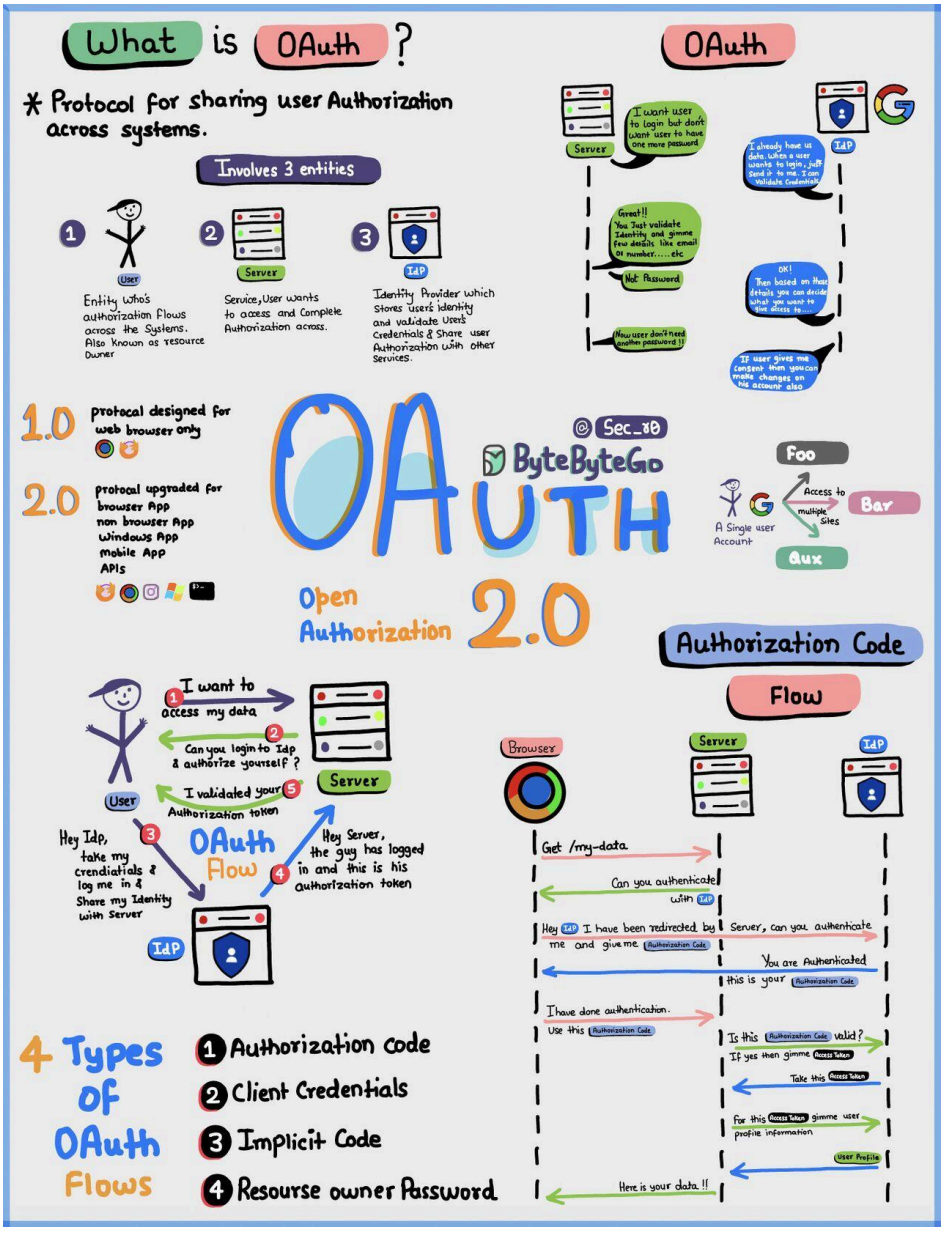
So atomicity means “all or nothing”.

- 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 loser 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.

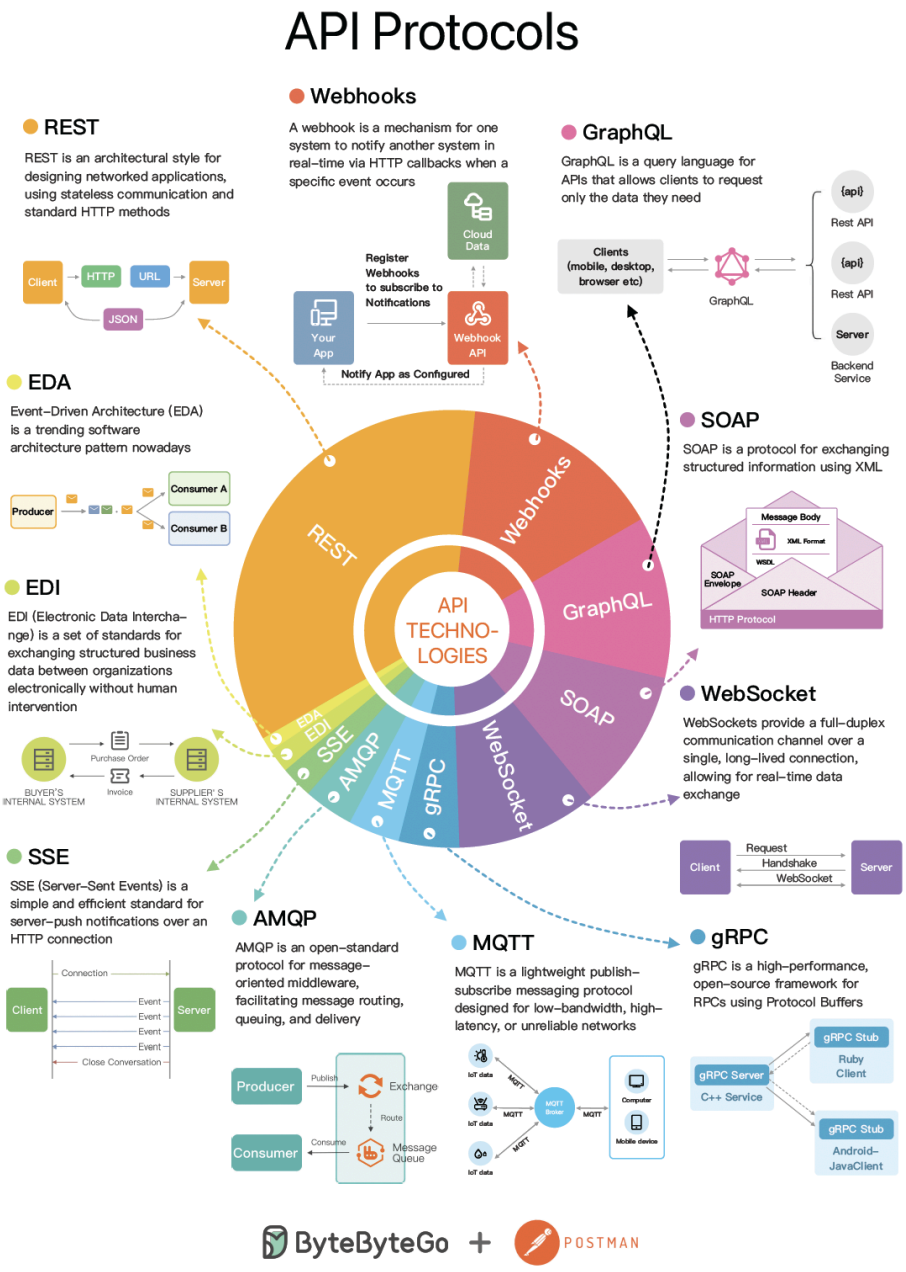
Remember, OAuth 2.0 is all about keeping you and your data safe while making your online experiences seamless and hassle-free across different applications and services.

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.

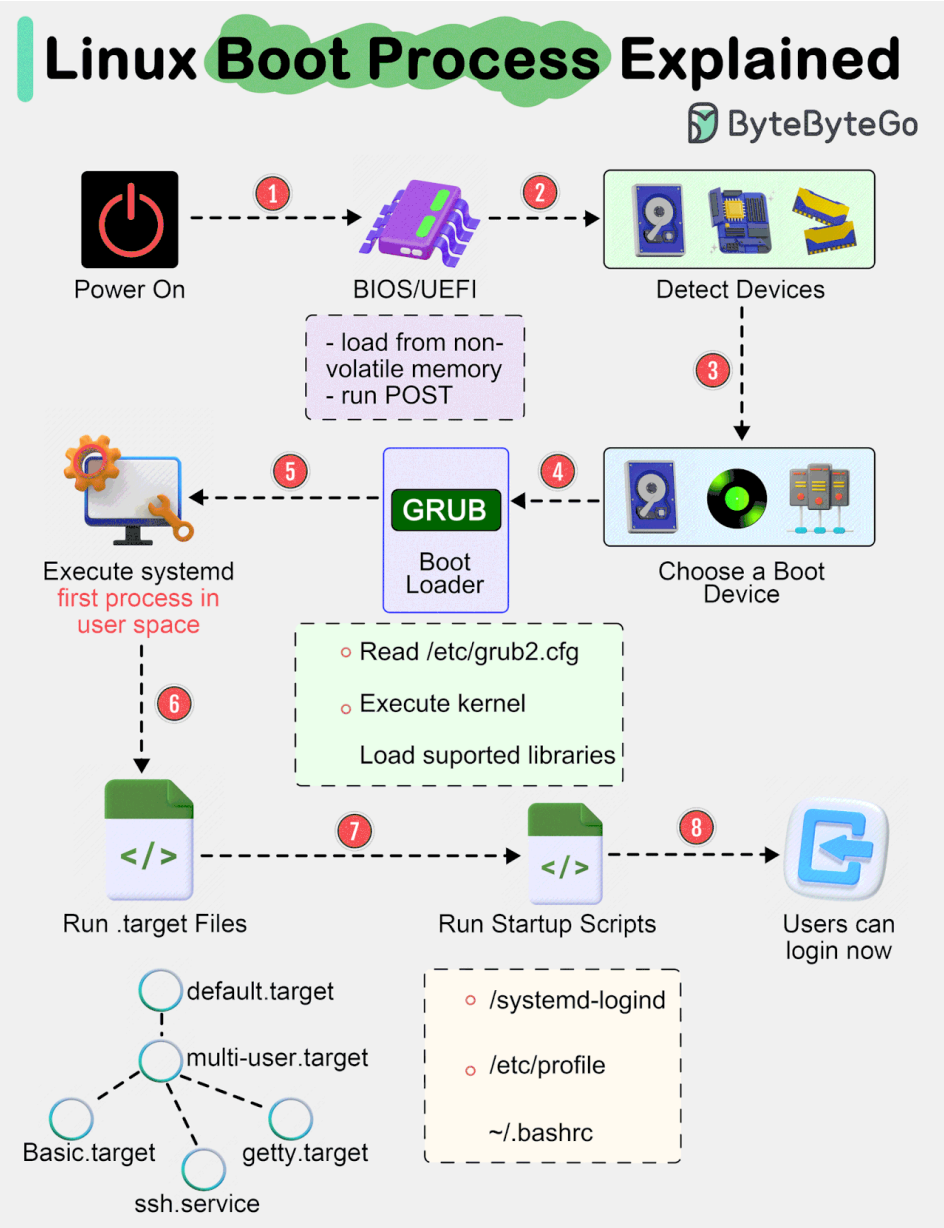


You can read the full blog post here: https://blog.postman.com/api-protocols-in-2023/

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.

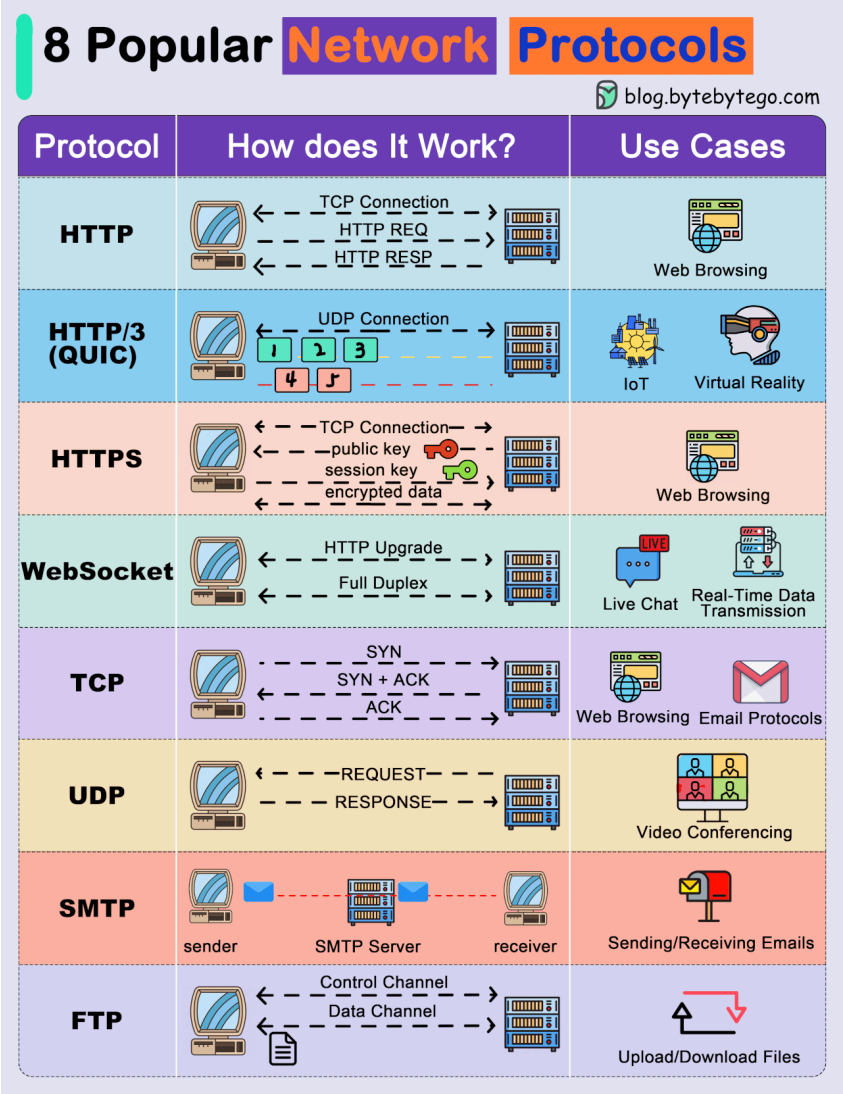
Step 6 - systemd activates the default. target unit by default when the system boots. Other analysis units are executed as well.

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.

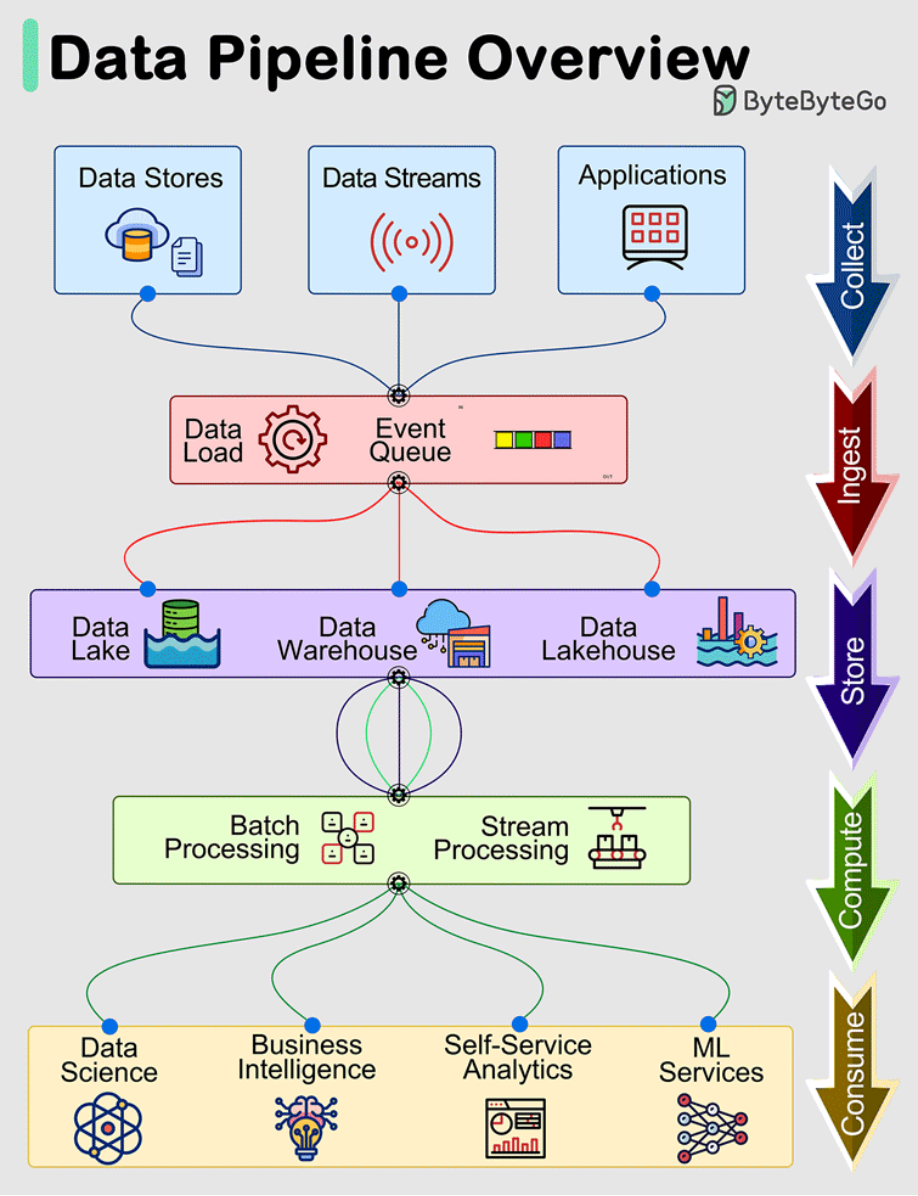
6. UDP (User Datagram Protocol)
UDP sends packets directly to a target computer, without establishing a connection first. UDP is commonly used in time-sensitive communications where occasionally dropping packets is better than waiting. Voice and video traffic are often sent using this protocol.

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.

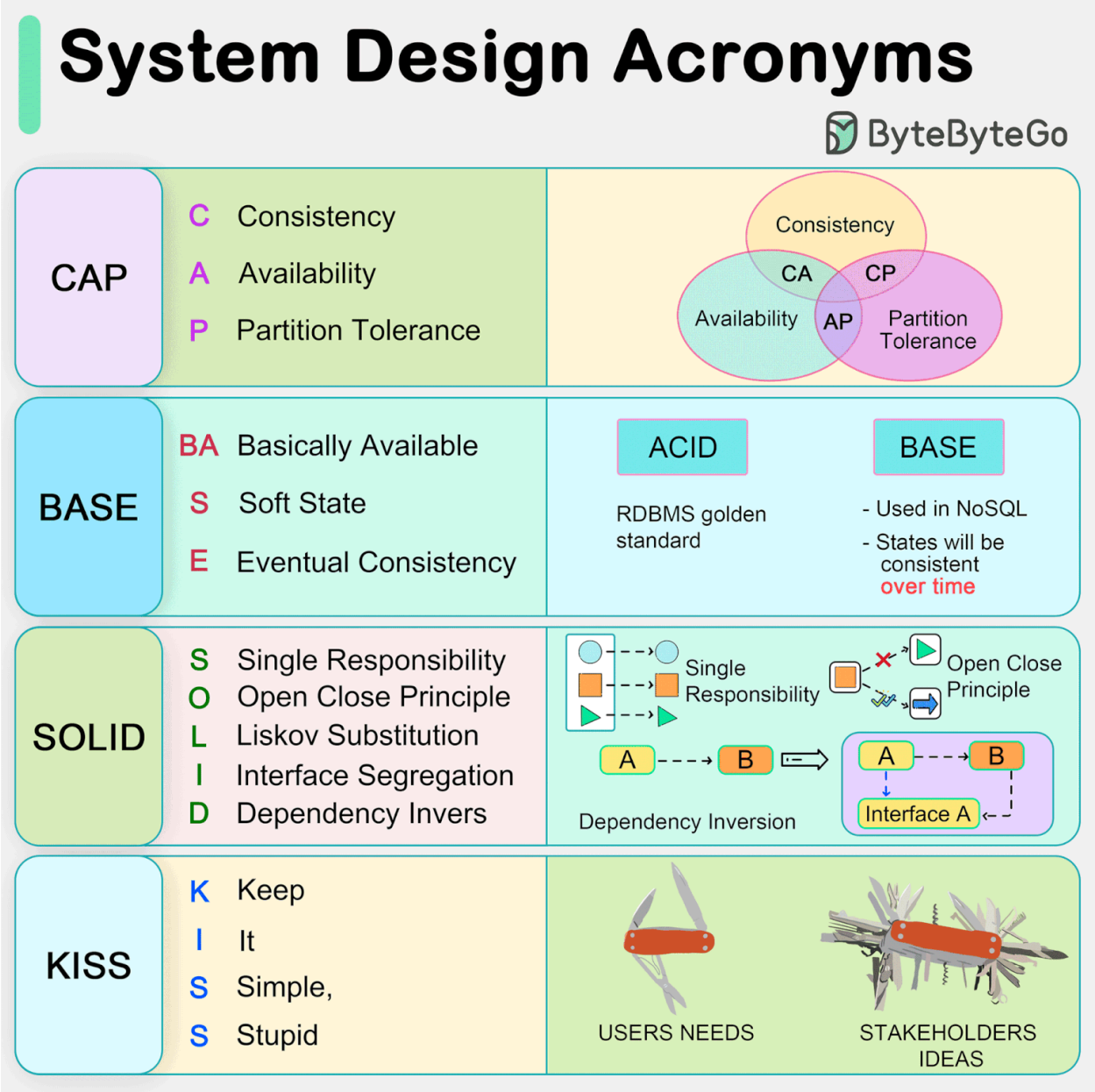
5. Consume:
Processed data is made available for consumption through analytics and visualization tools, operational data stores, decision engines, user-facing applications, dashboards, data science, machine learning services, business intelligence, and self-service analytics.

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.

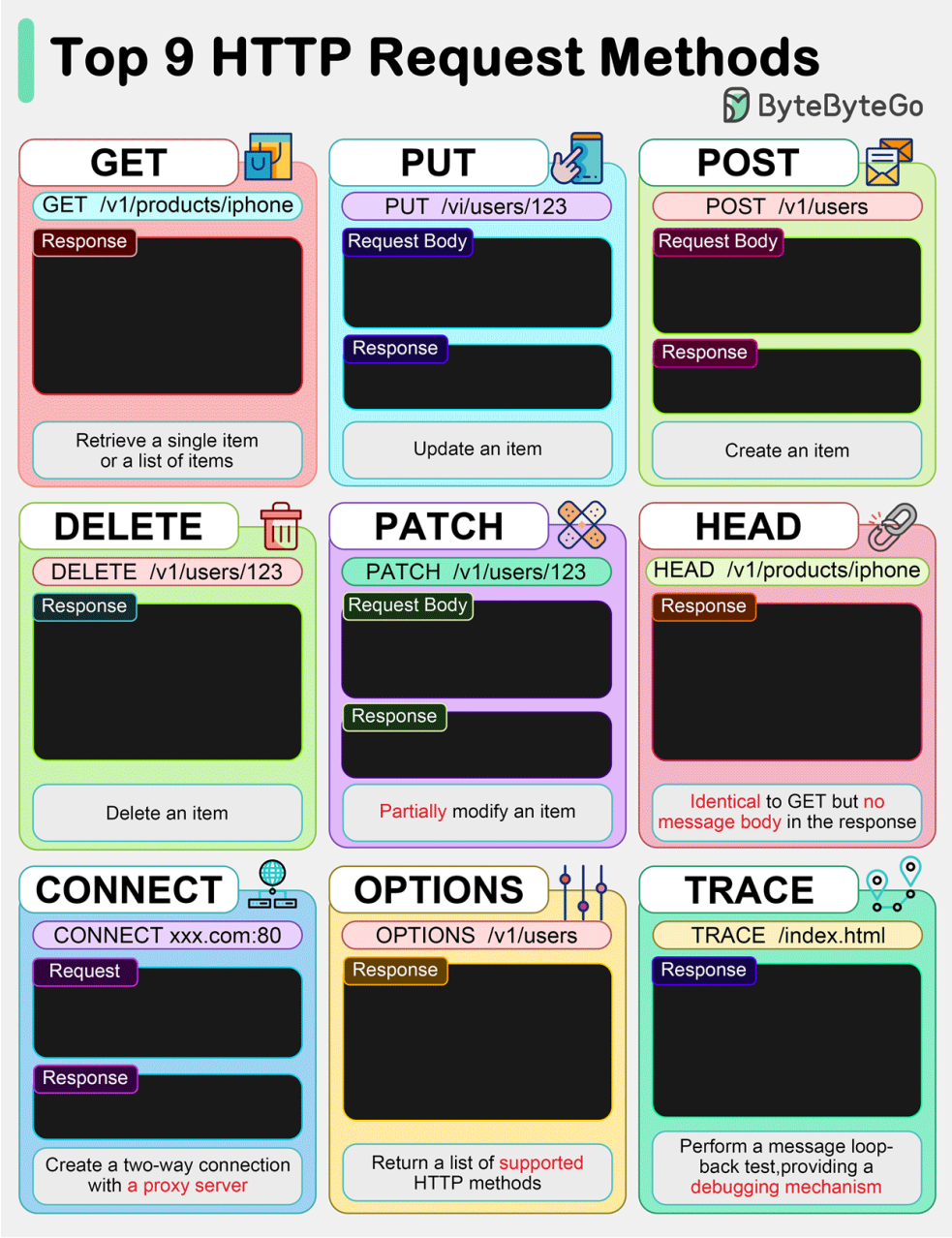
  4. ISP (Interface Segregation Principle)
  Expose multiple interfaces with specific responsibilities.

  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.

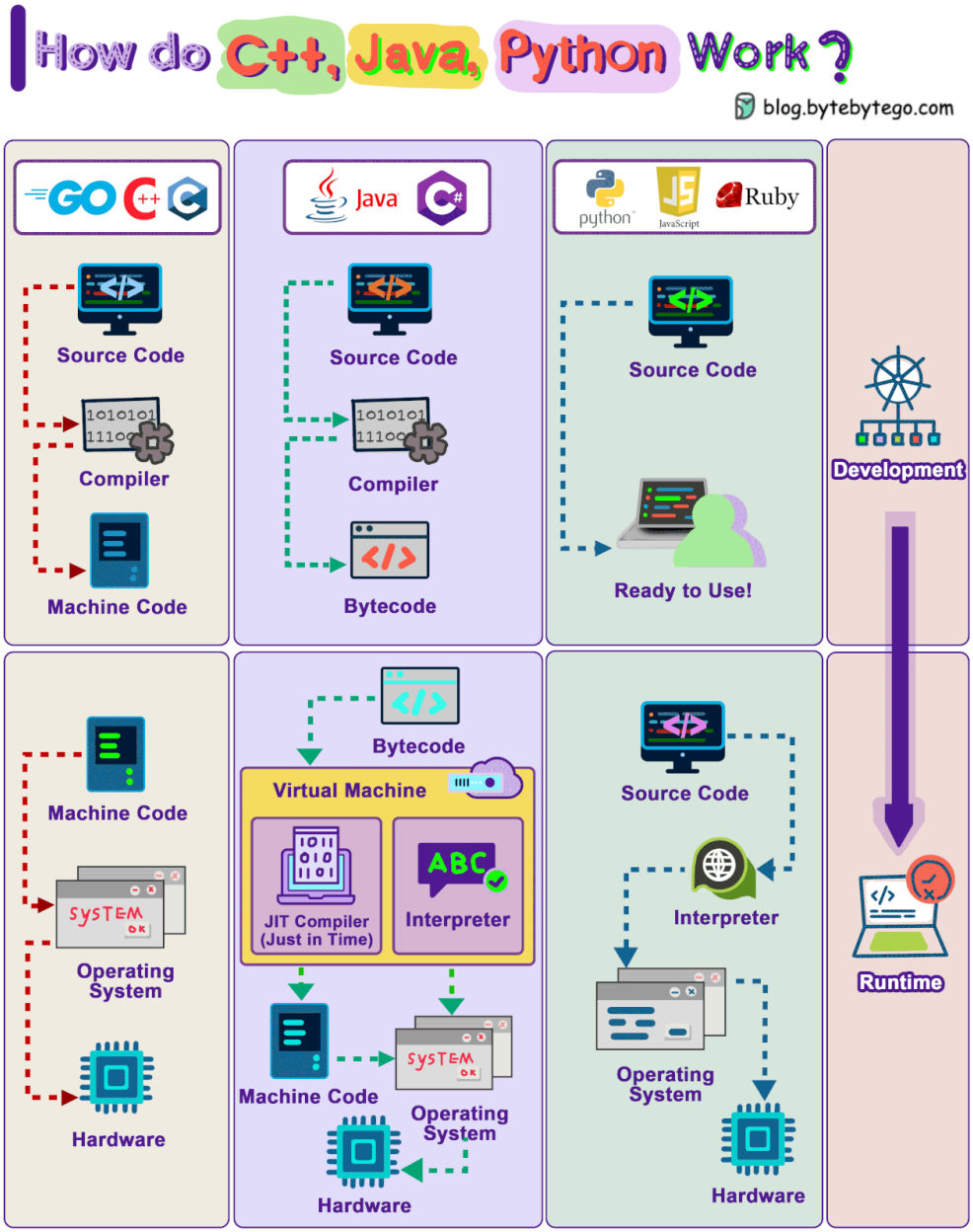
8. HTTP OPTIONS
   This describes the communication options for the target resource.

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

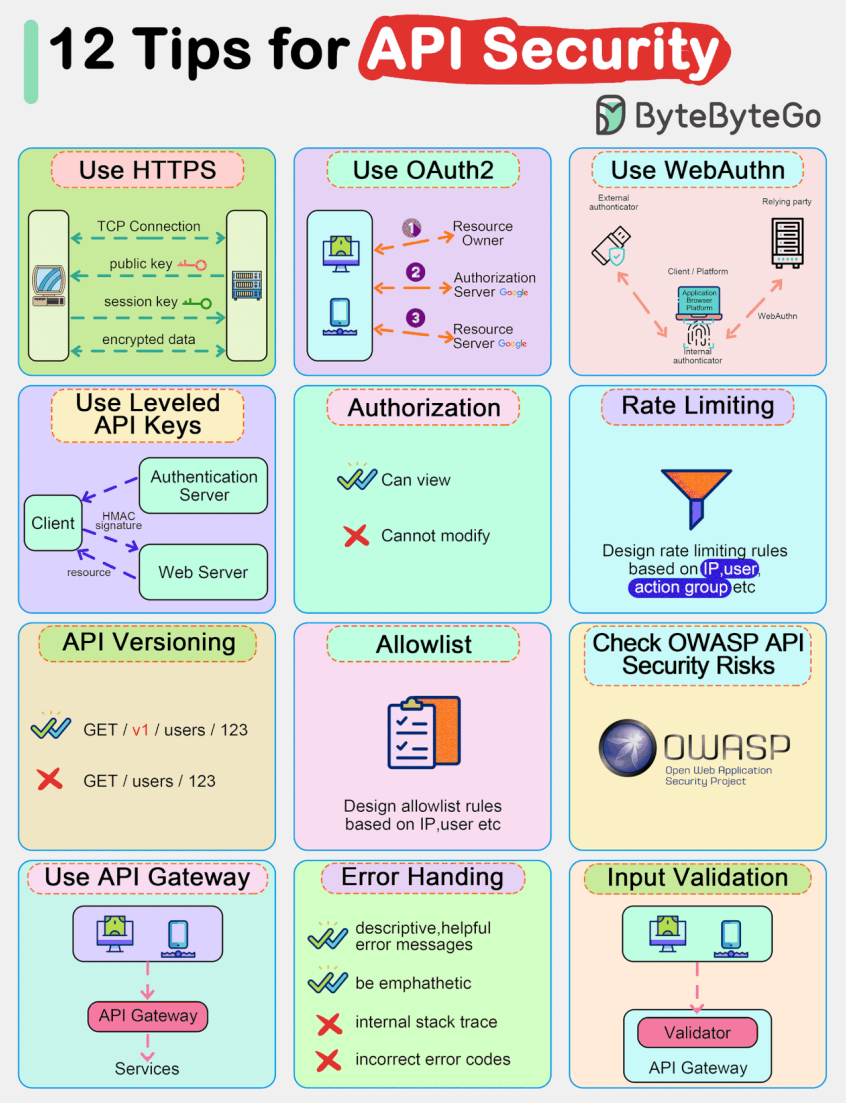
machine code to speed up the execution. Examples: Java, C#

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?

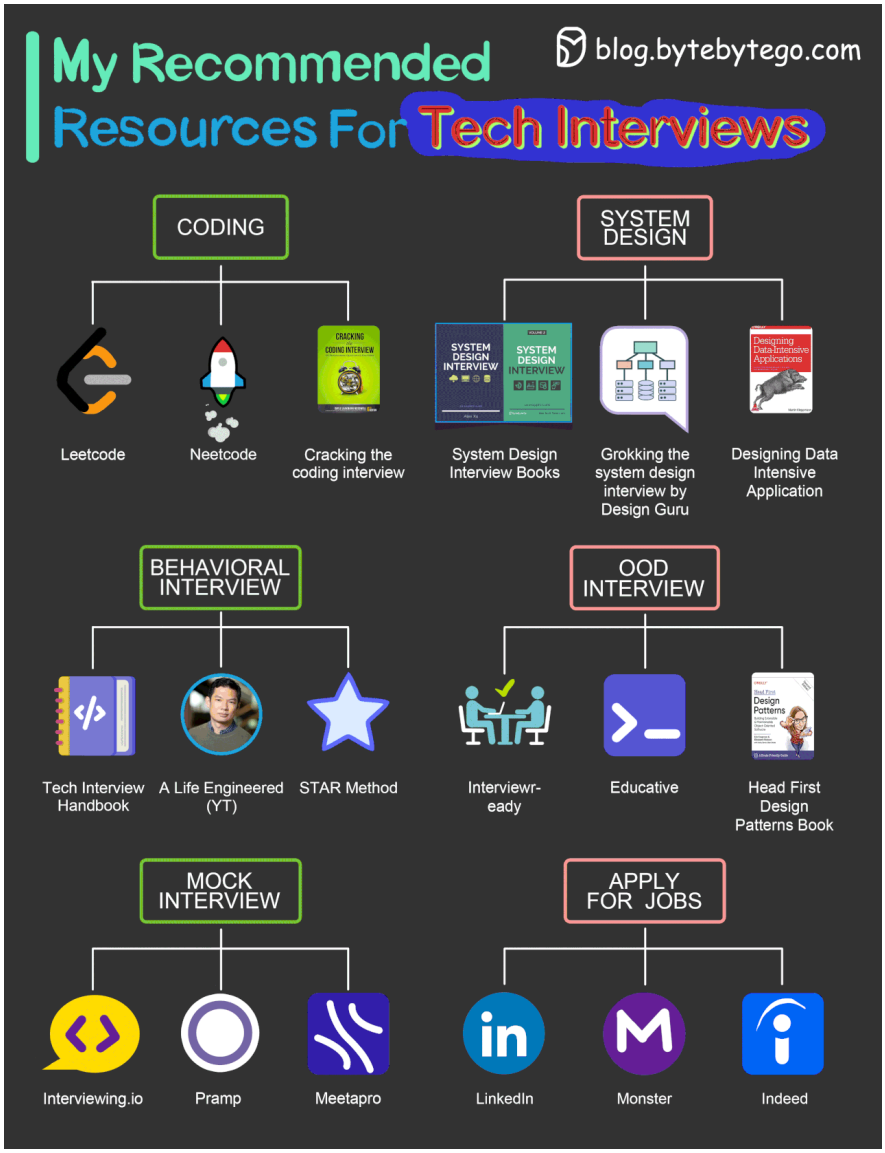
Top 12 Tips for API Security



- 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

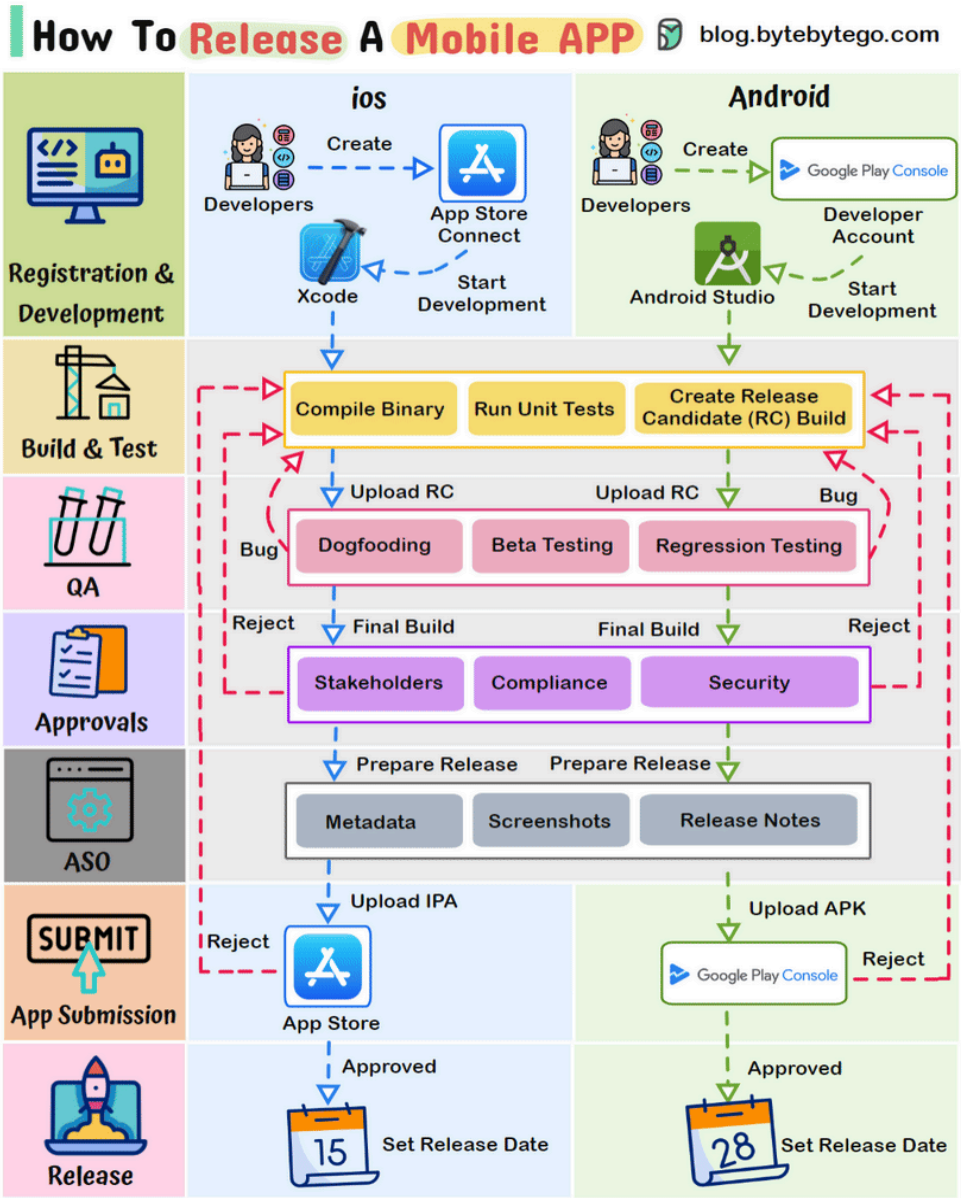
Apply for Jobs

- 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)

What's included?

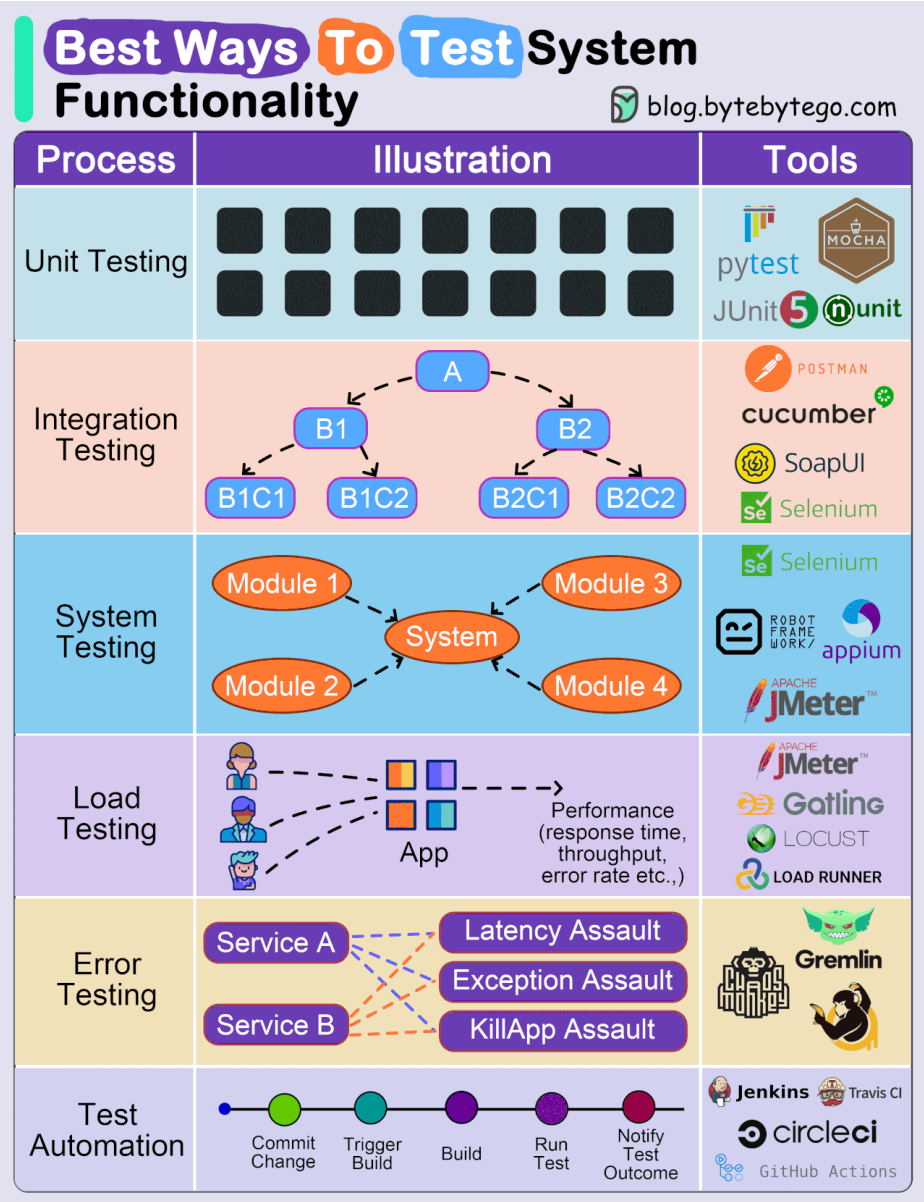
- AWS, Azure, Google Cloud, Oracle Cloud, Alibaba Cloud
- Cloud servers
- Databases

- 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.



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.

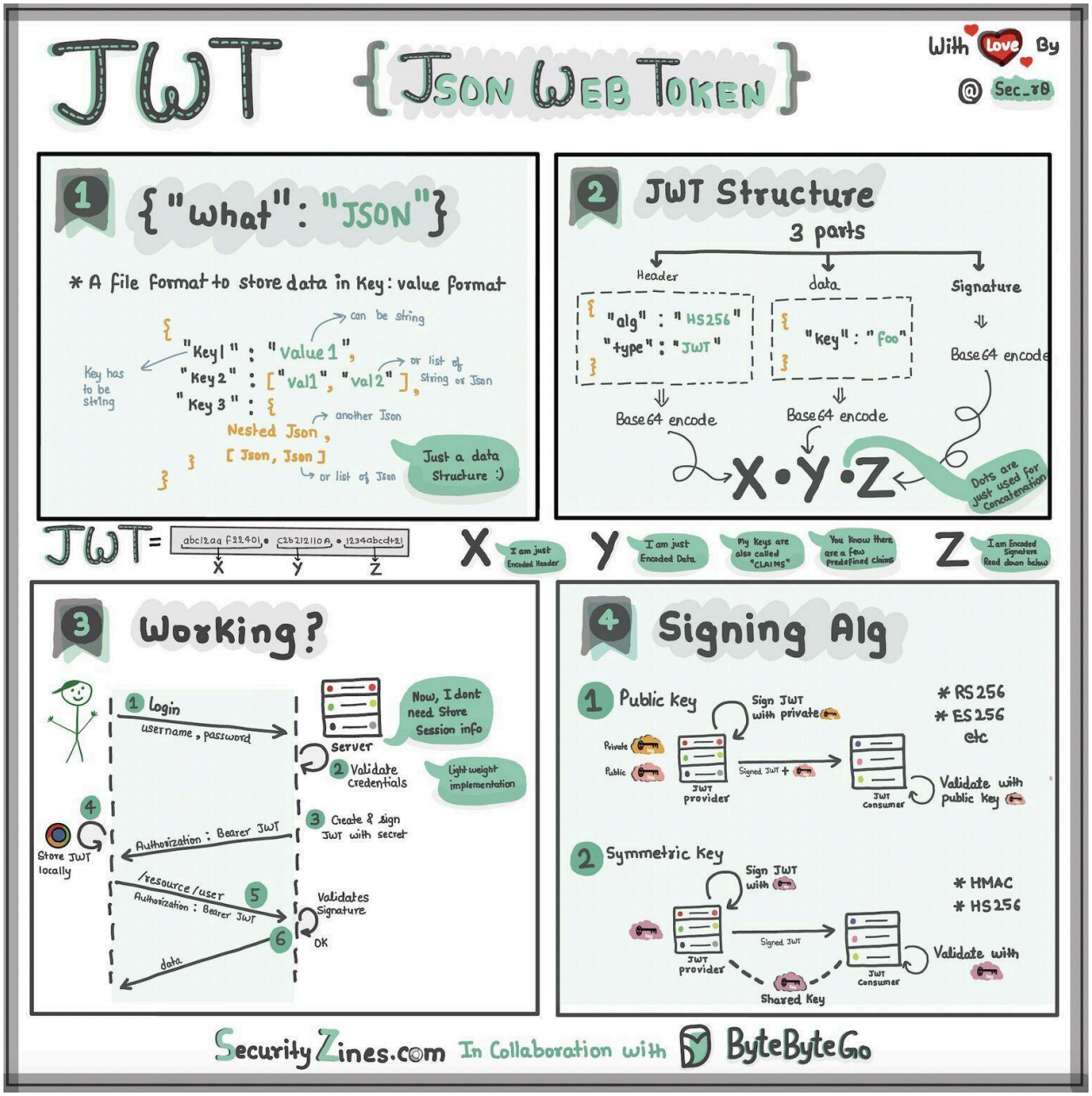
6. Test Automation: Automates test case execution for efficiency, repeatability, and error reduction.

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



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.

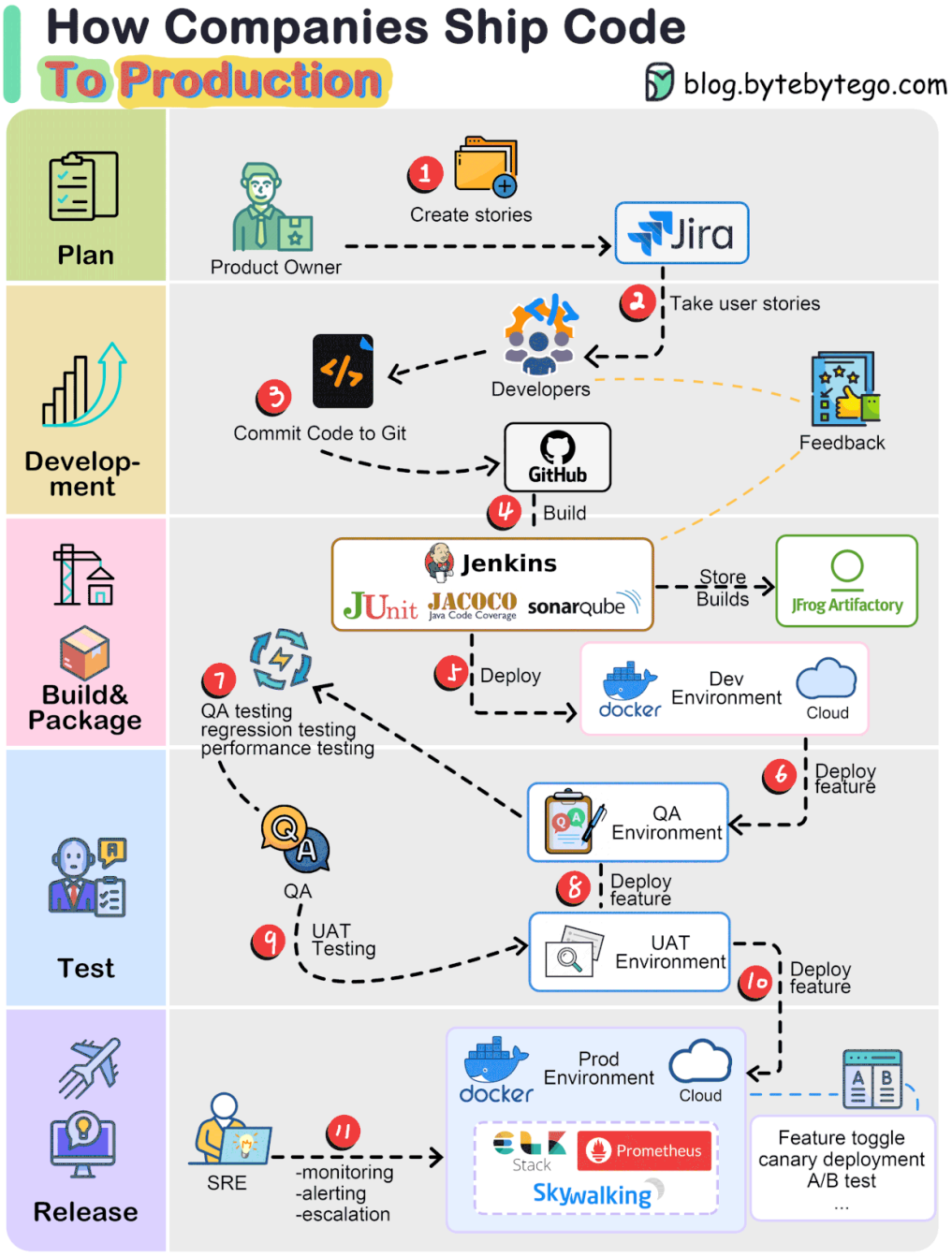
Now, the signature is what makes the JWT secure. It's like a special seal that only the sender knows how to create. The signature is created using a secret code, kind of like a password. This signature ensures that nobody can tamper with the contents of the JWT without the sender knowing about it.

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.

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

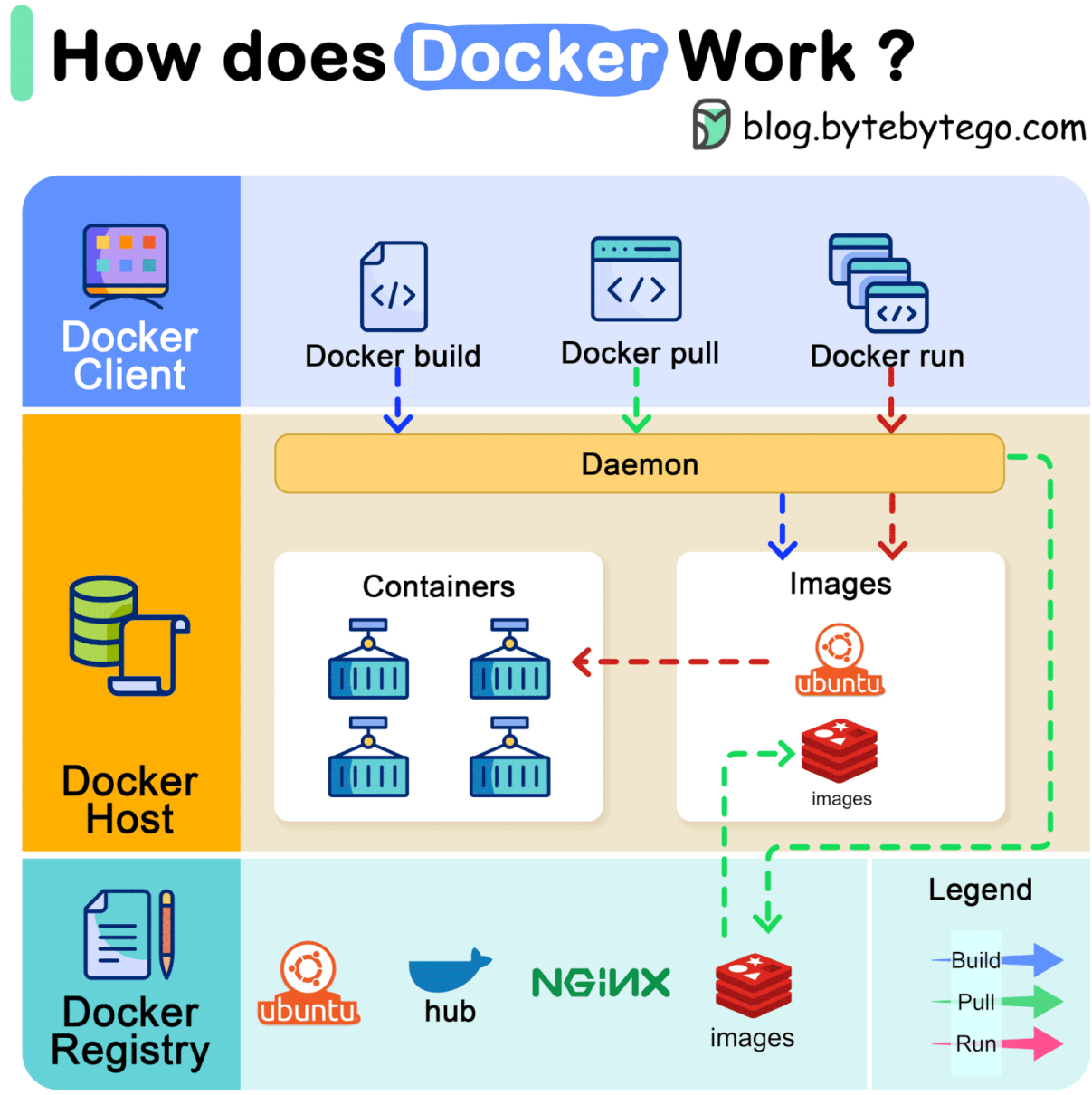
Step 9: If the UAT testing is successful, the builds become release candidates and will be deployed to the production environment on schedule.

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.



Docker's architecture comprises three main components:

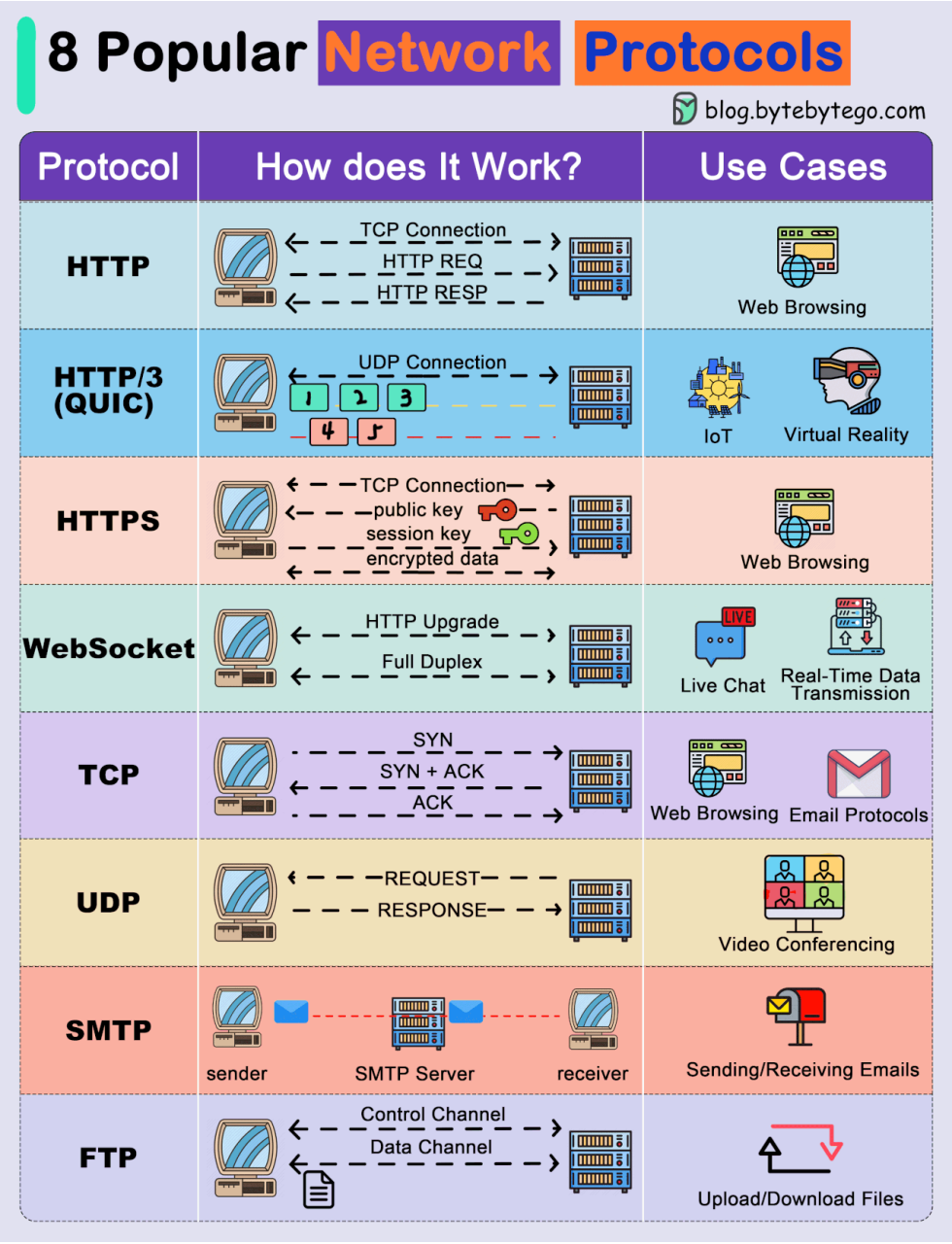
- Docker Client
  This is the interface through which users interact. It communicates with the Docker daemon.

- 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 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.

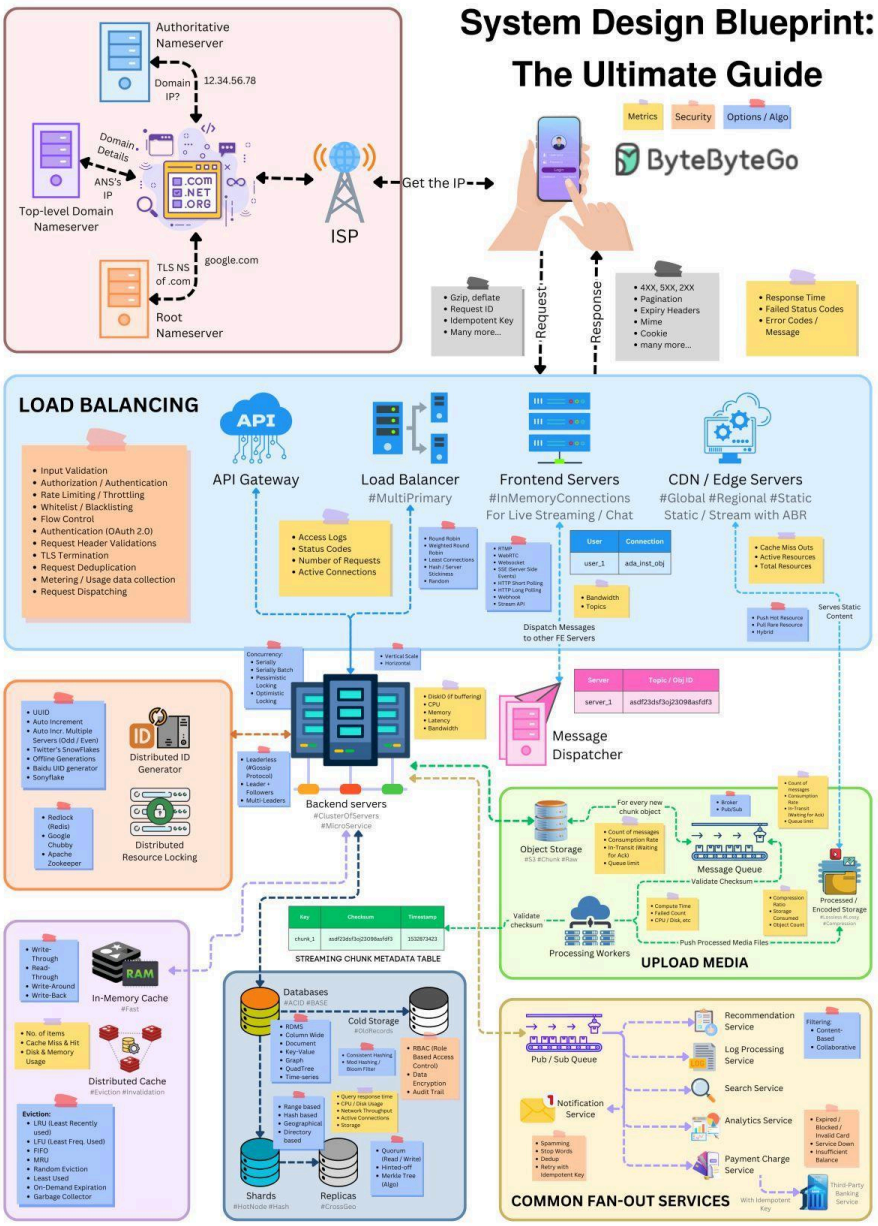
6. UDP (User Datagram Protocol)
   UDP sends packets directly to a target computer, without establishing a connection first. UDP is commonly used in time-sensitive communications where occasionally dropping packets is better than waiting. Voice and video traffic are often sent using this protocol.

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.

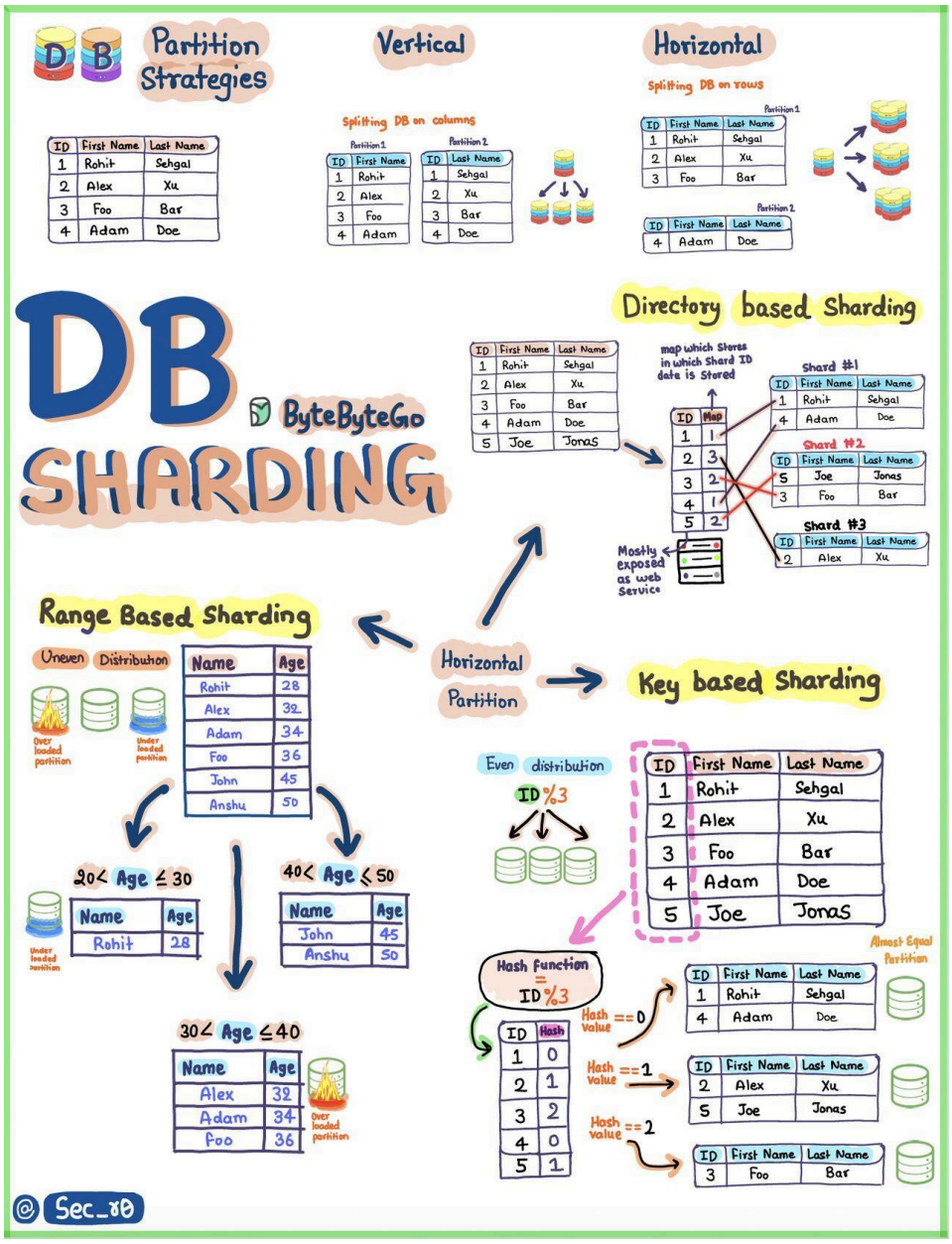
This briefly touches on the following discussion points:

- 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.



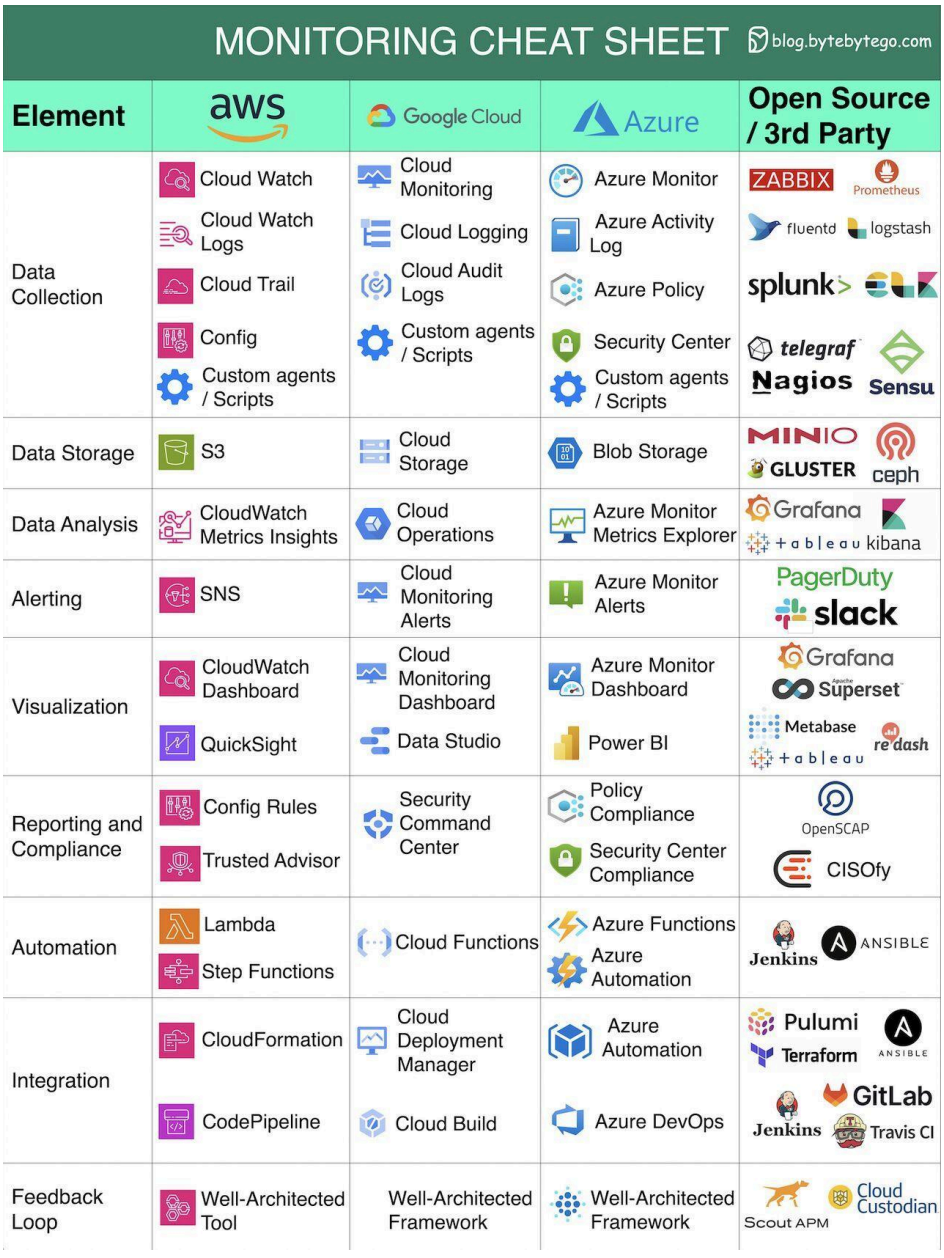
1. Range-Based Sharding: Splitting your data into distinct ranges. Think of it as organizing your books by genre on separate shelves.

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.



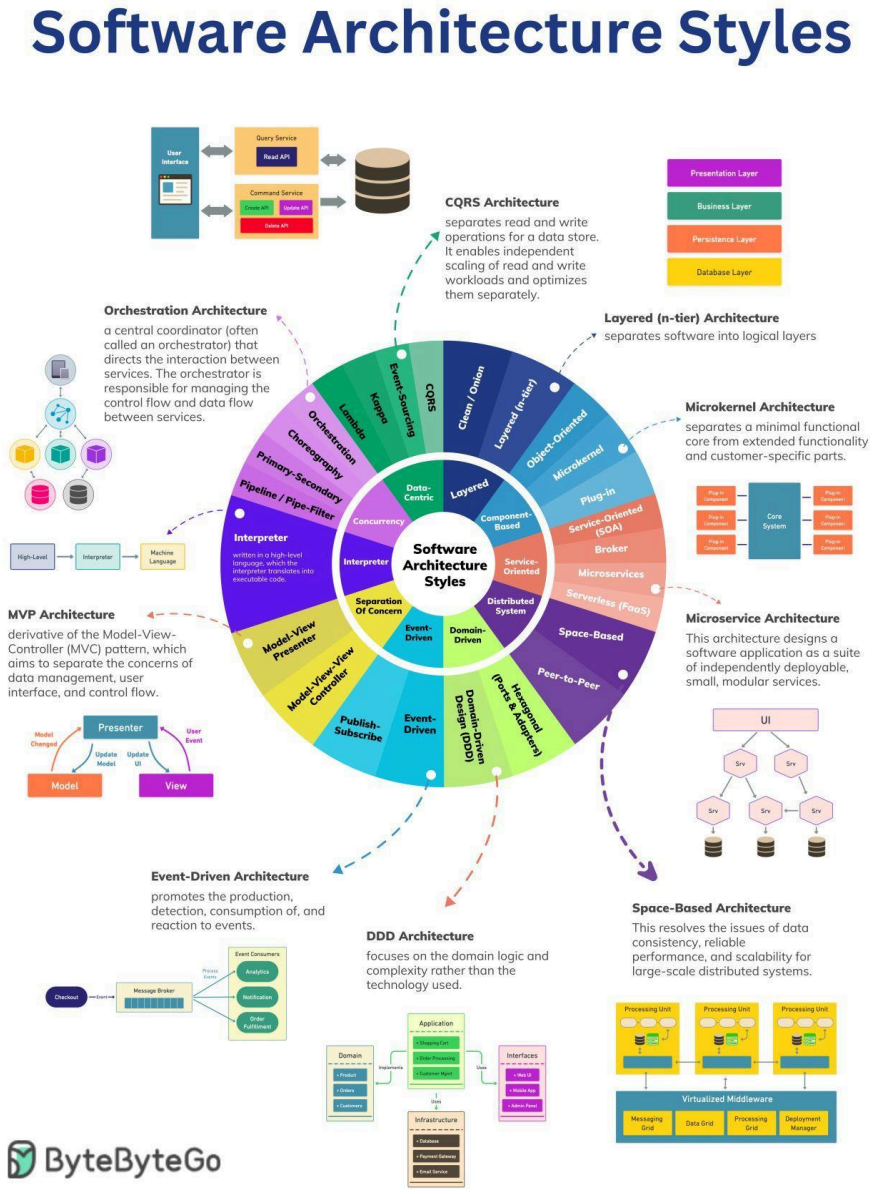
Let's delve into the essential monitoring aspects covered:

- 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

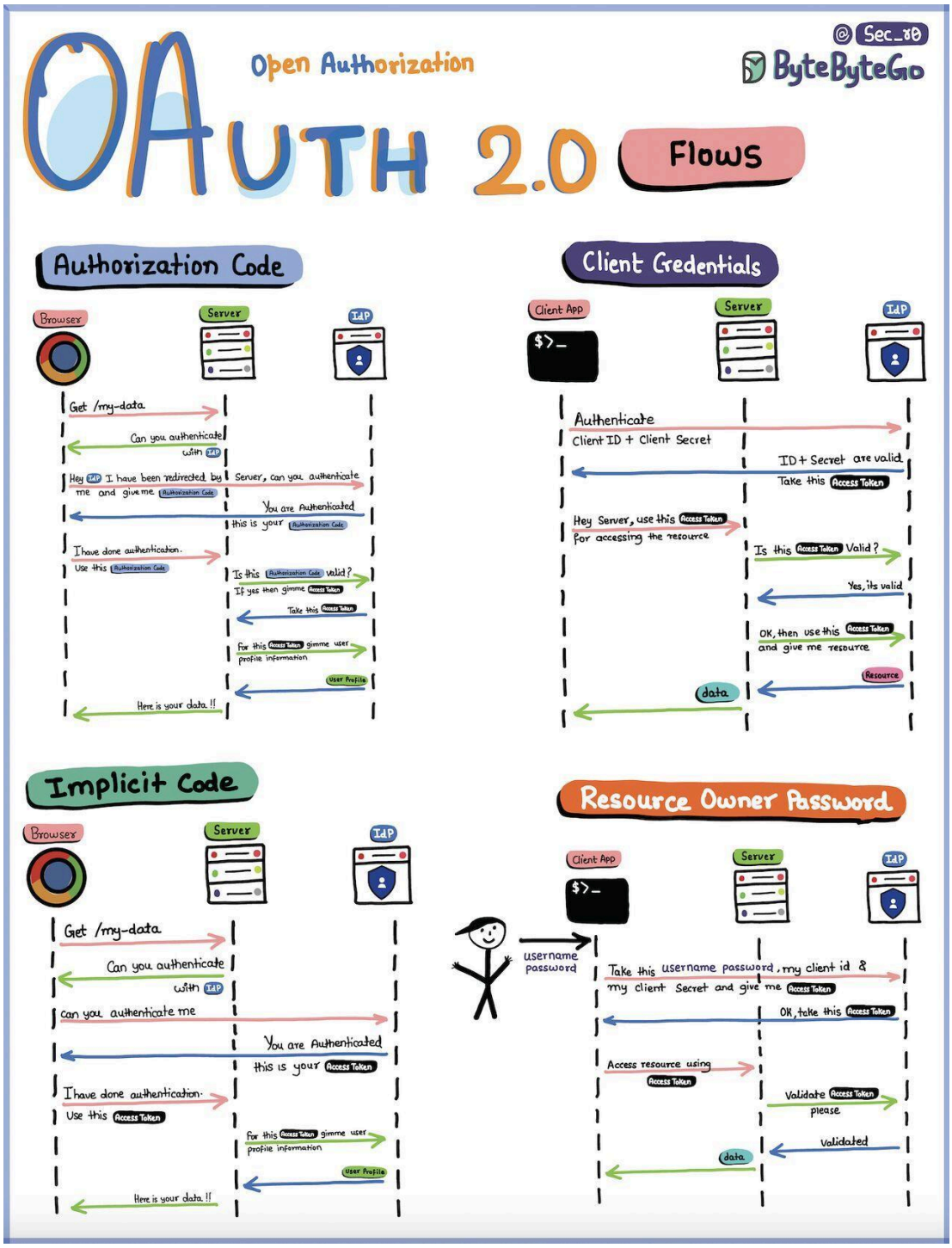


In software development, architecture plays a crucial role in shaping the structure and behavior of software systems. It provides a blueprint for system design, detailing how components interact with each other to deliver specific functionality. They also offer solutions to common problems, saving time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

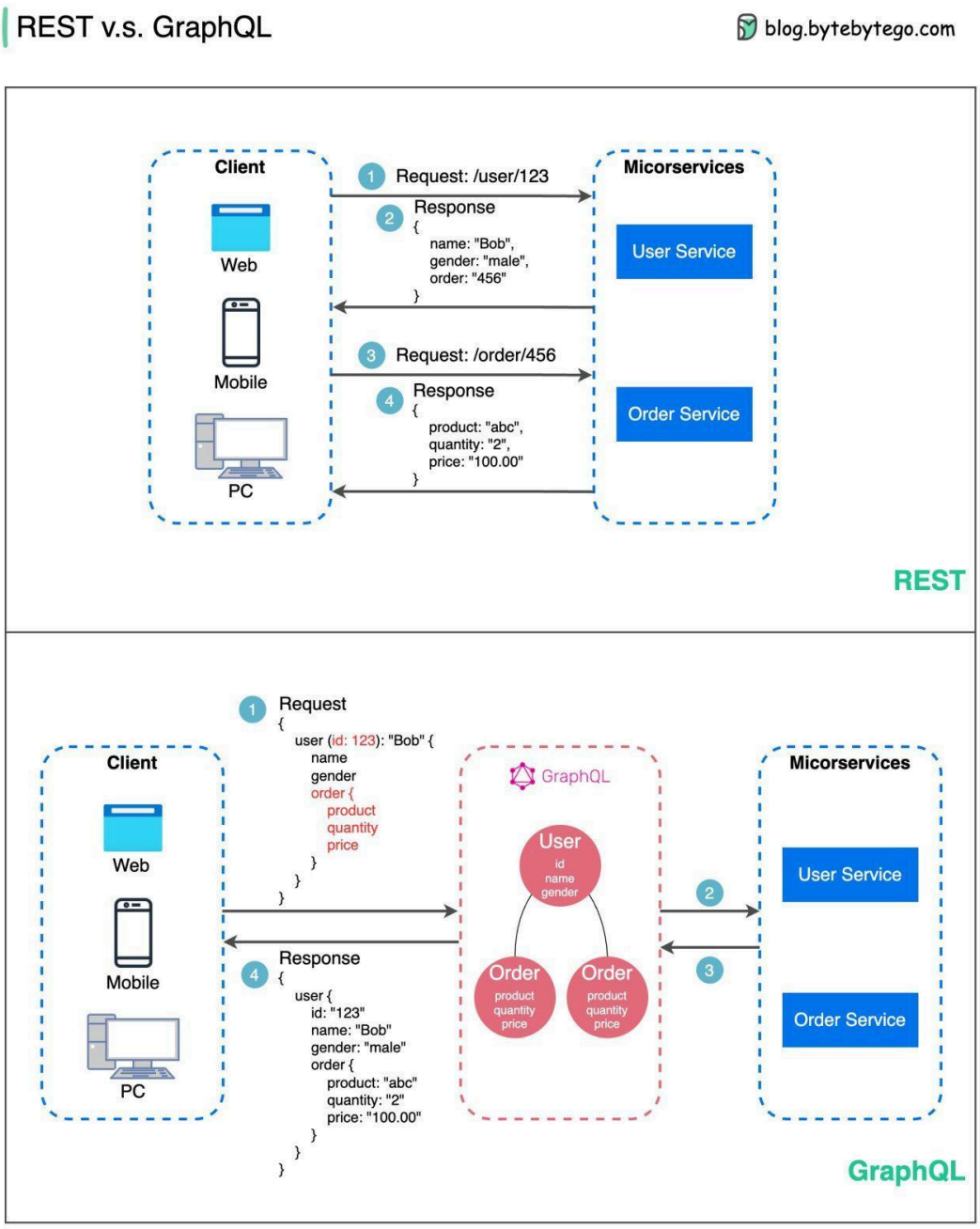
Note:

- 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.

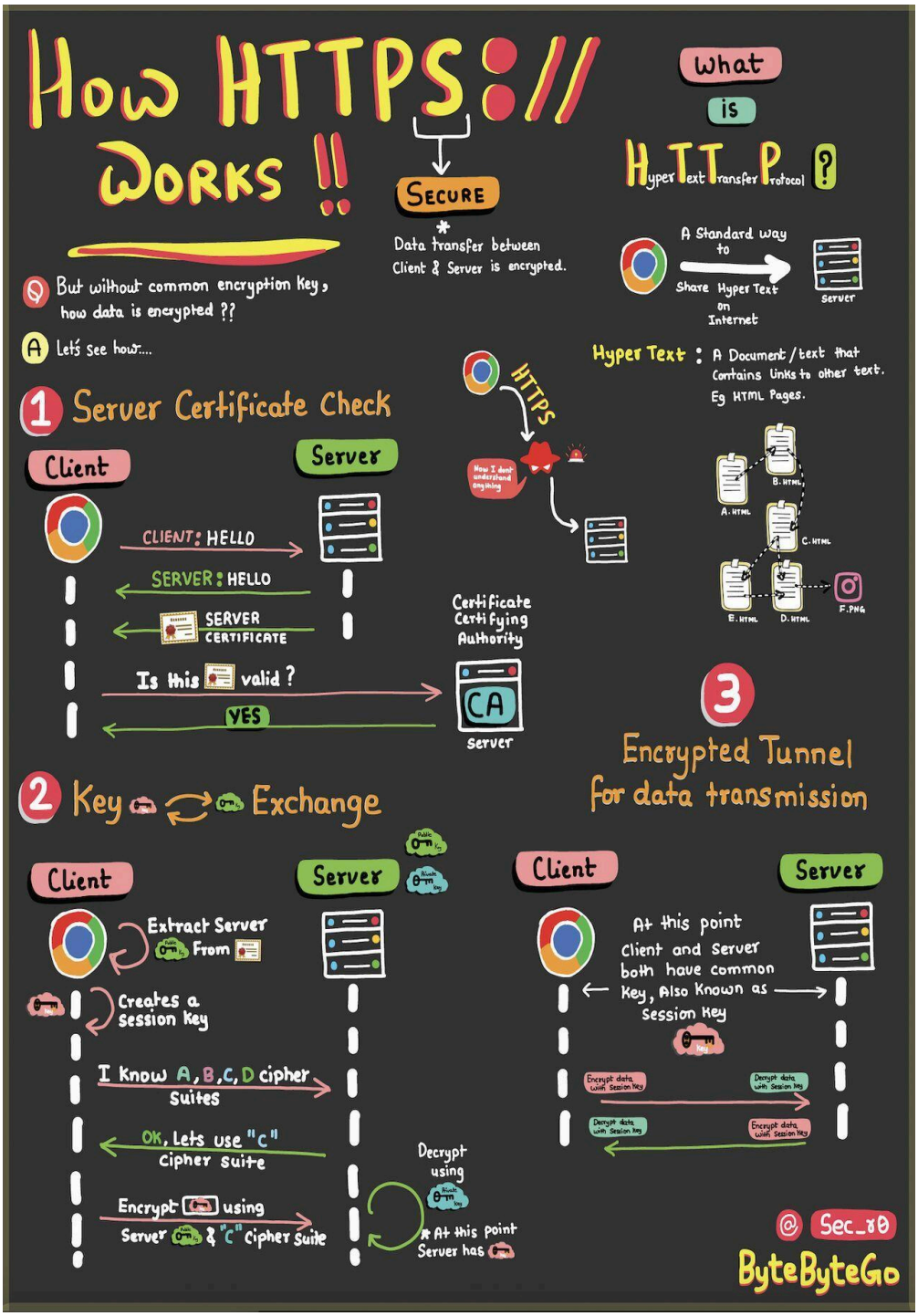
- GraphQL can aggregate multiple REST requests into one query. GraphQL server organizes the resources in a graph.

- 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.

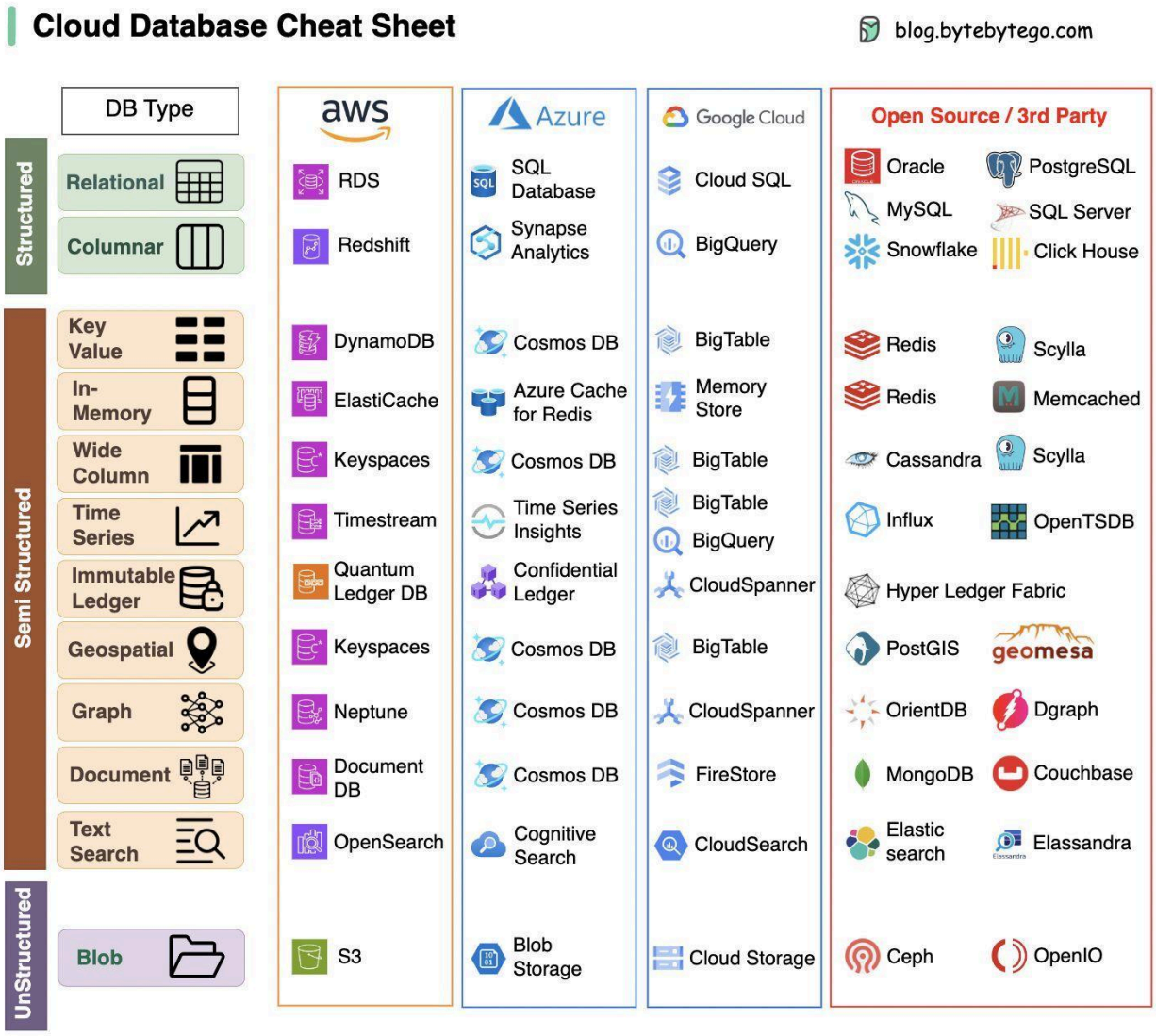
SSL Handshake: Behind the Scenes — Witness the cryptographic protocols that establish a secure connection. Experience the intricate exchange of keys and negotiation.

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



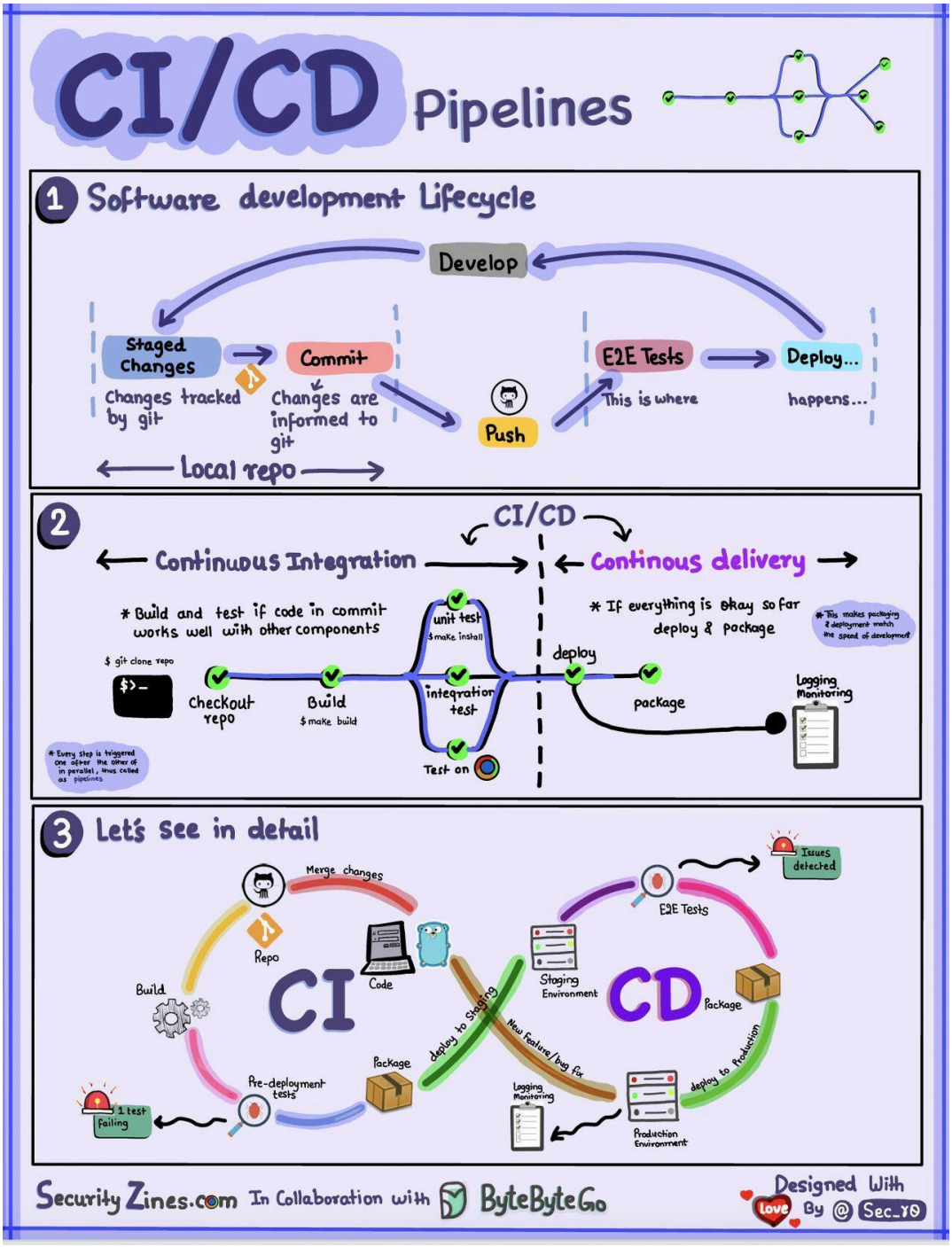
Choosing the right database for your project is a complex task. The multitude of database options, each suited to distinct use cases, can quickly lead to decision fatigue.

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.

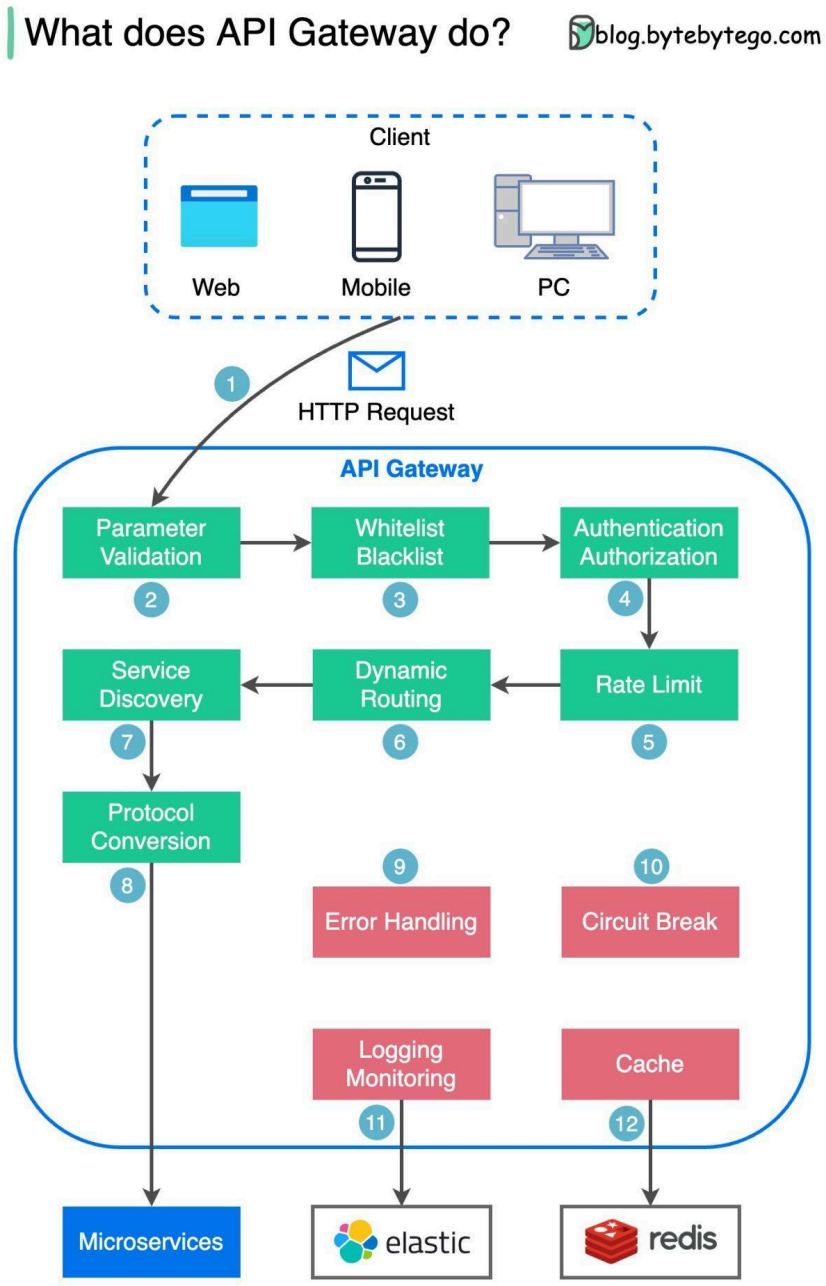
Continuous Delivery (CD) automates release processes like infrastructure changes and deployment. It ensures software can be released reliably at any time through automated workflows. CD may also automate the manual testing and approval steps required before production deployment.

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.

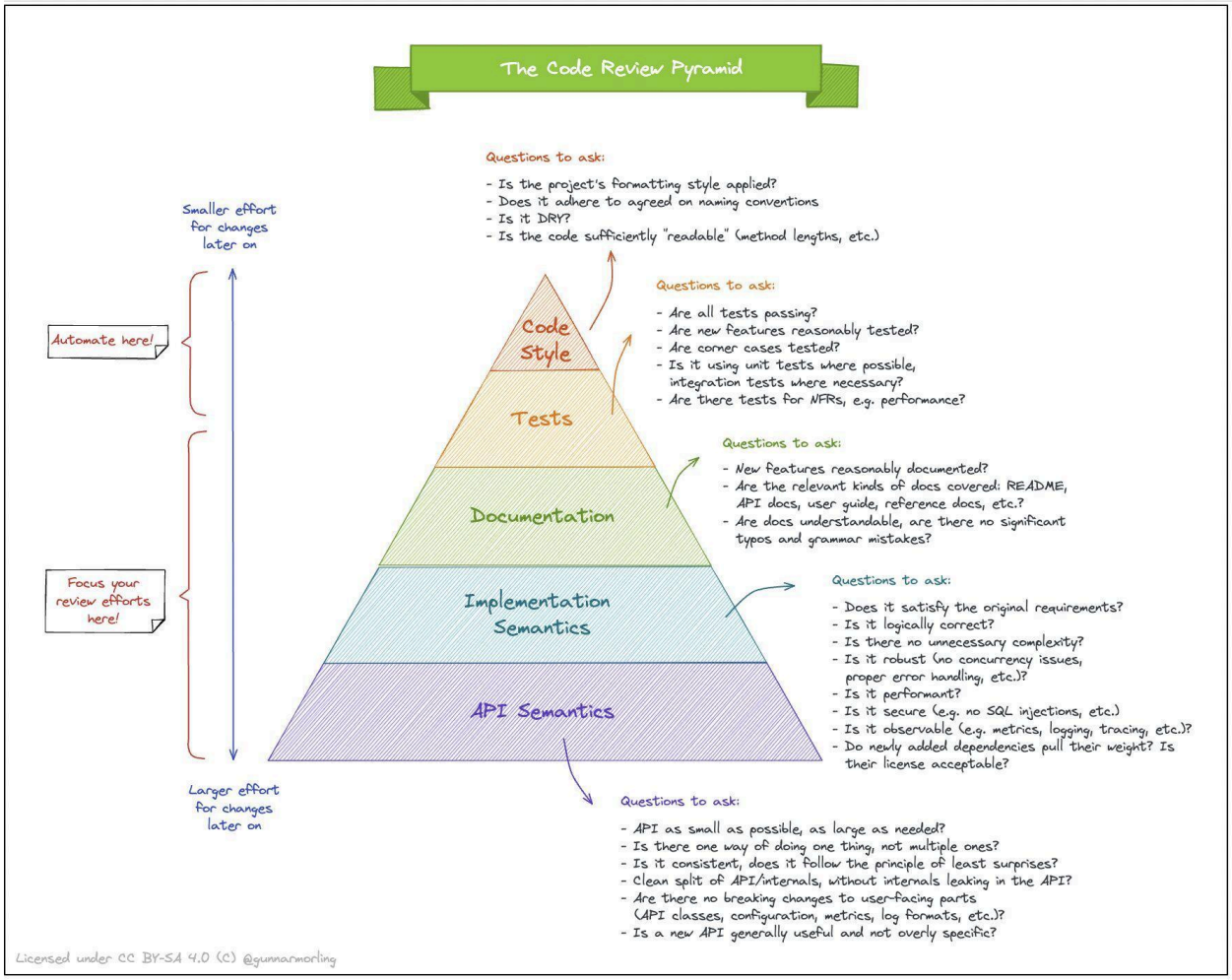
Steps 9-12: The API gateway can handle errors properly, and deals with faults if the error takes a longer time to recover (circuit break). It can also leverage ELK (Elastic-Logstash-Kibana) stack for logging and monitoring. We sometimes cache data in the API gateway.

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?

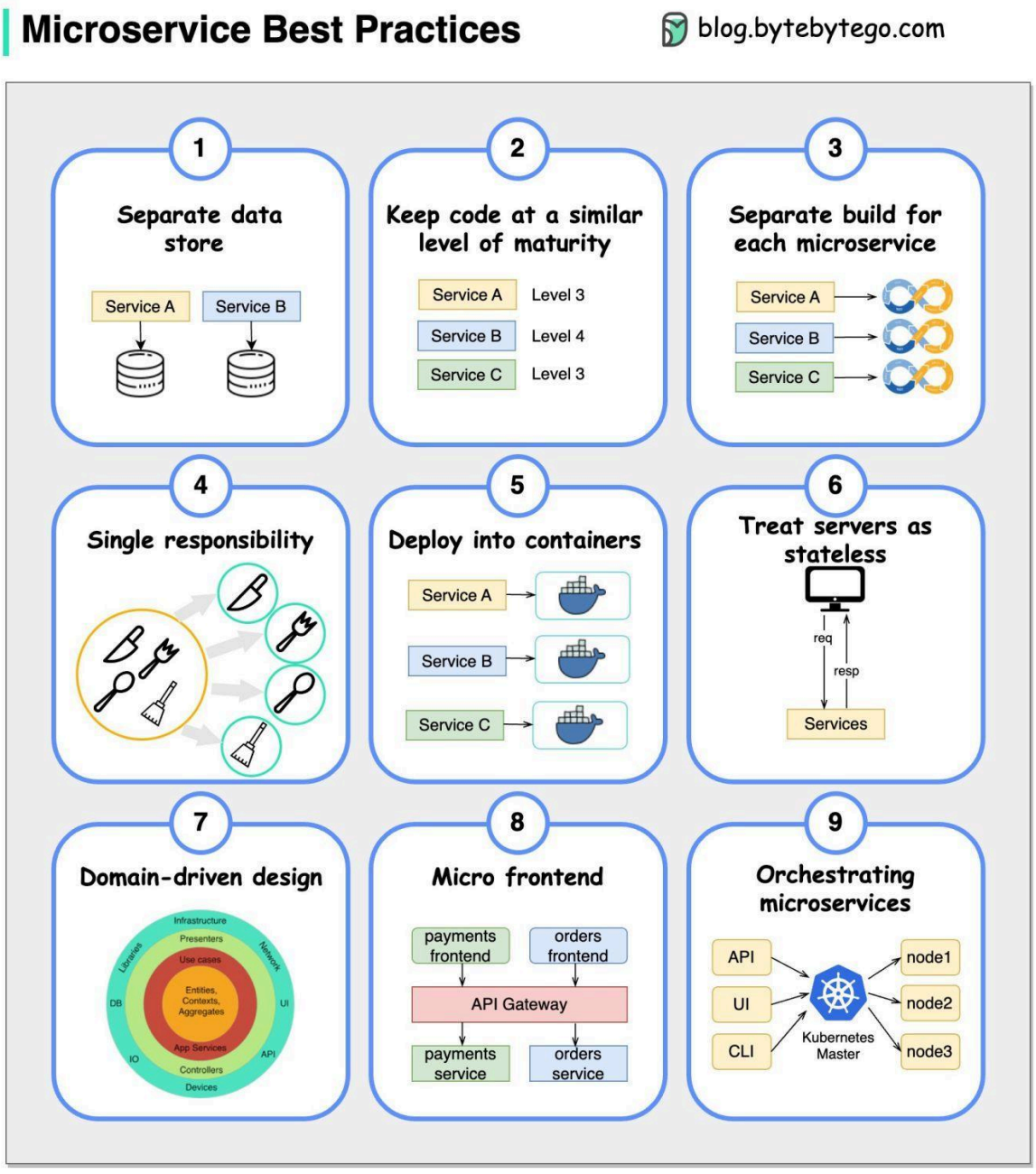
The Code Review Pyramid

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



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 languages[1]. It measures the runtime, memory usage, and energy consumption of each language. The abstract of the paper is shown below.

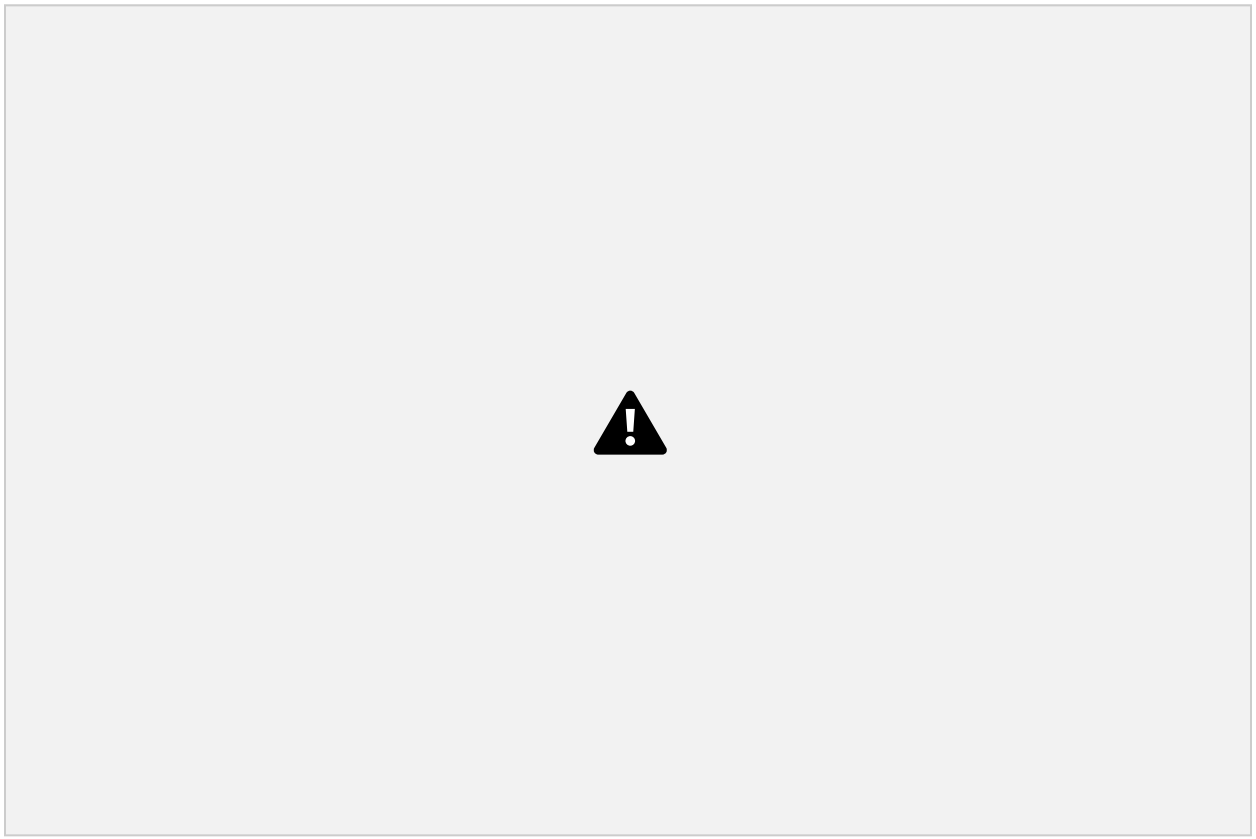
“This paper presents a study of the runtime, memory usage and energy consumption of twenty seven well-known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as, slower/faster 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?

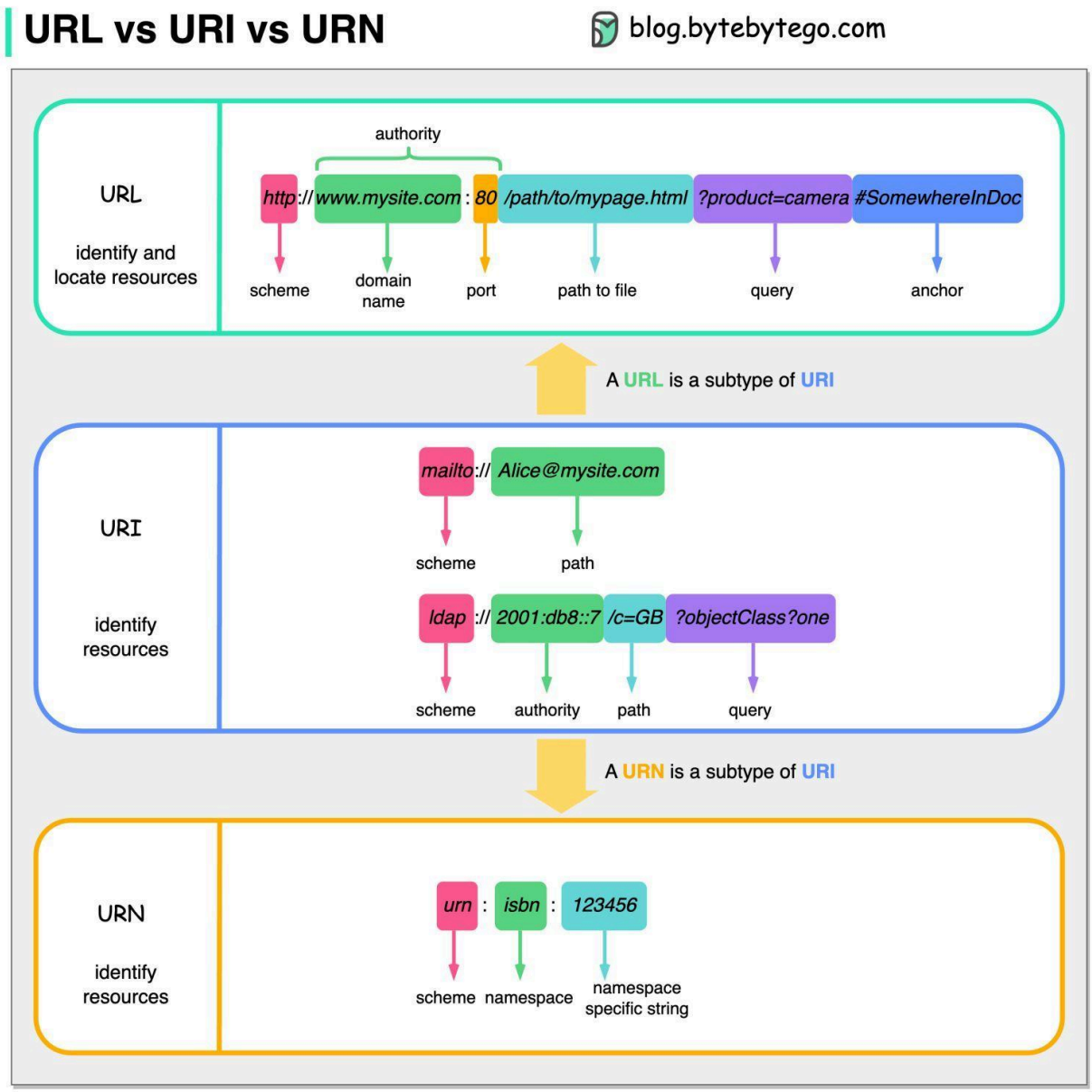
An amazing illustration of how to build a resilient three-tier architecture on AWS

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:

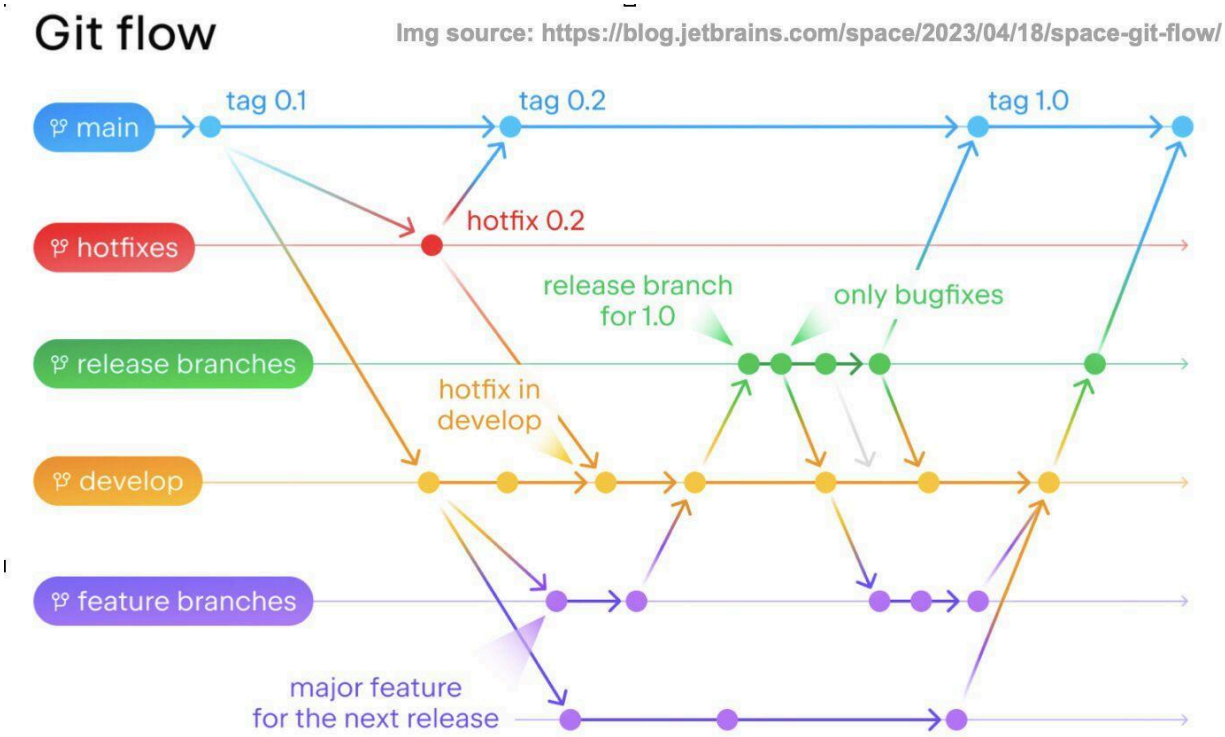
scheme:[//authority]path[?query][#fragment]

- 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?



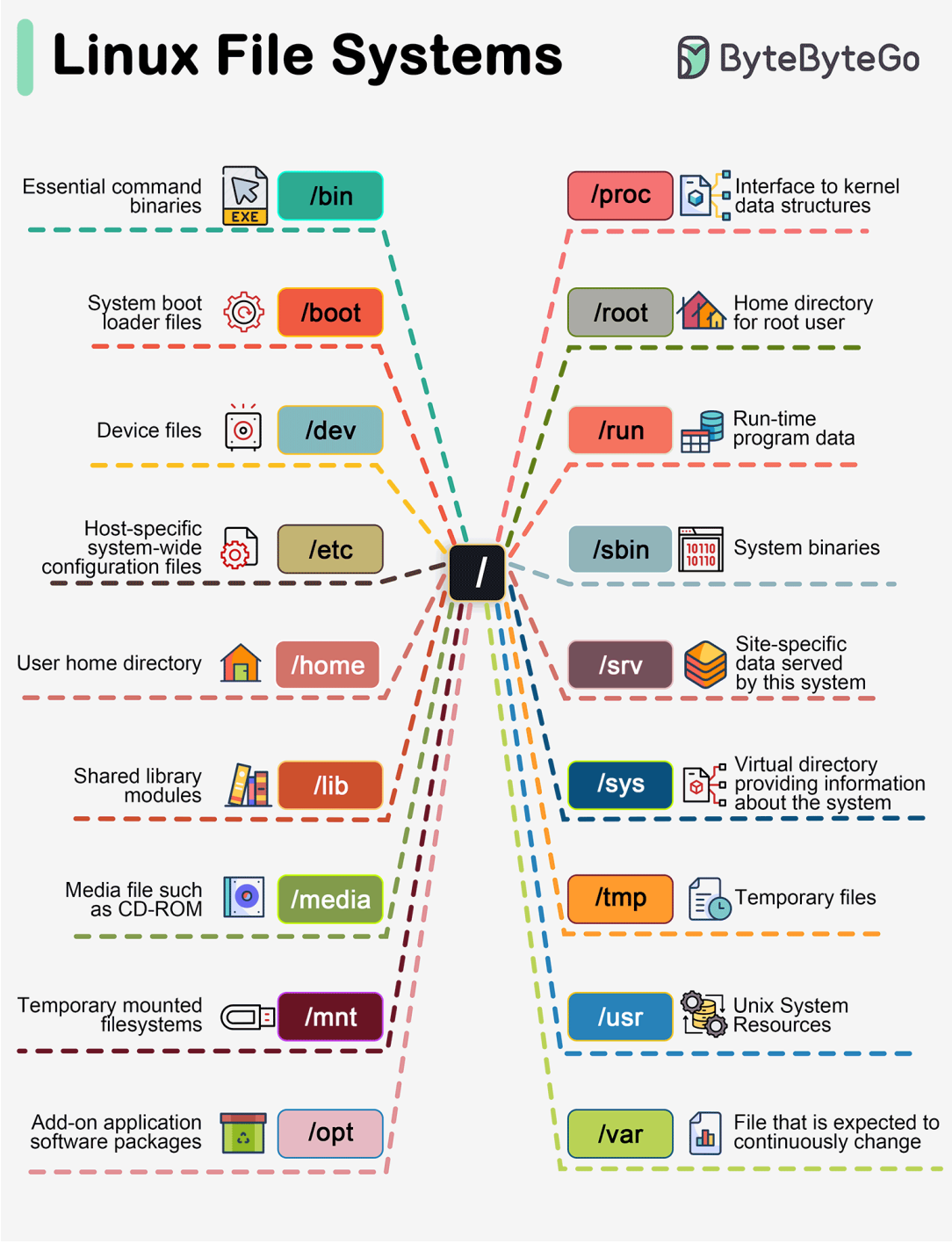
- The `main` branch is for production code only.
- The `develop` branch is for development code.
- `feature` branches are created from the `develop` branch.
- `hotfix` branches are created from the `main` branch.
- `release` branches are created from the `develop` branch.

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.

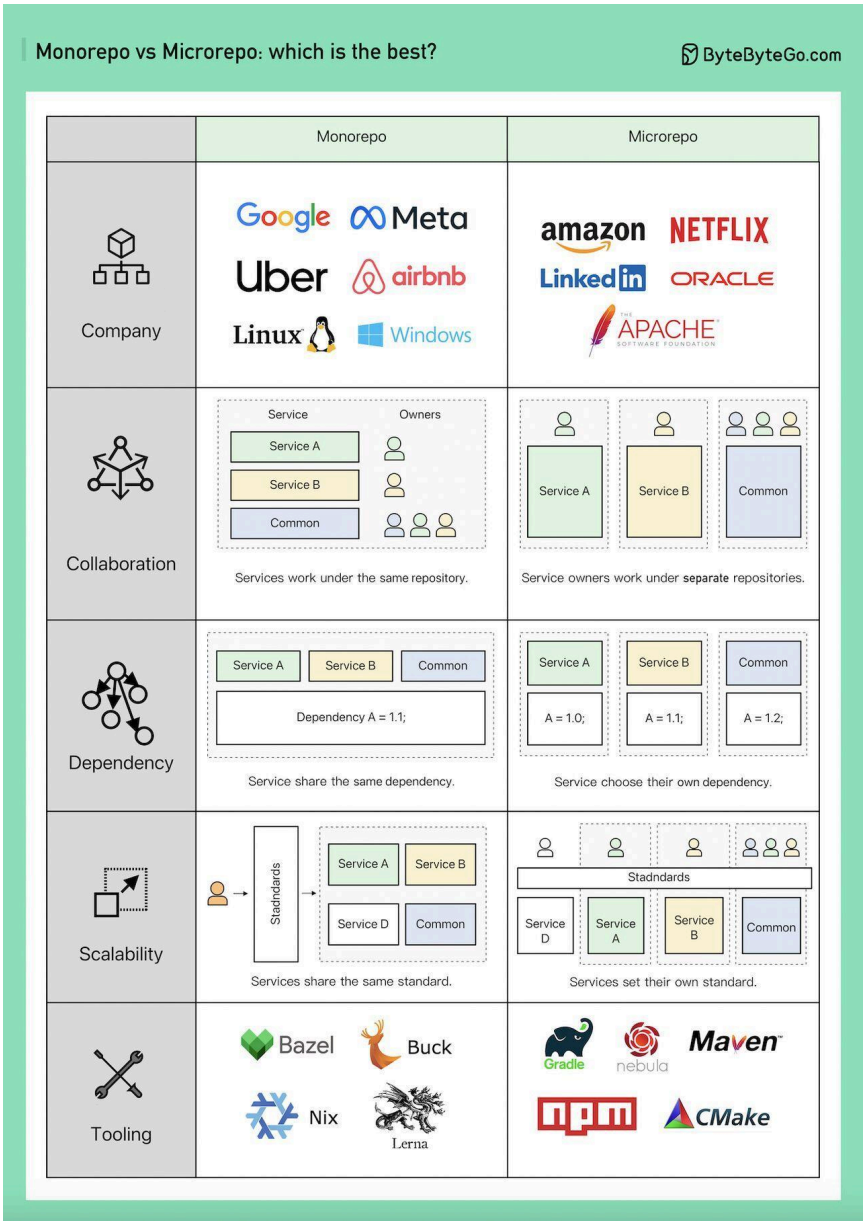
Have fun exploring!

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 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.

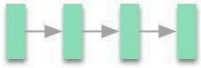
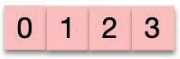
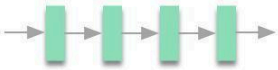
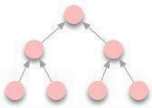
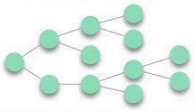
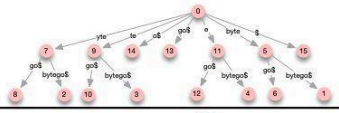
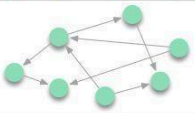
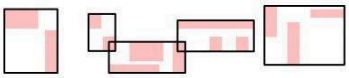
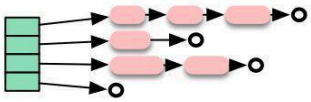
Google engineers built Bazel, and Meta built Buck. There are other open-source tools available, including Nix, Lerna, and others.

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

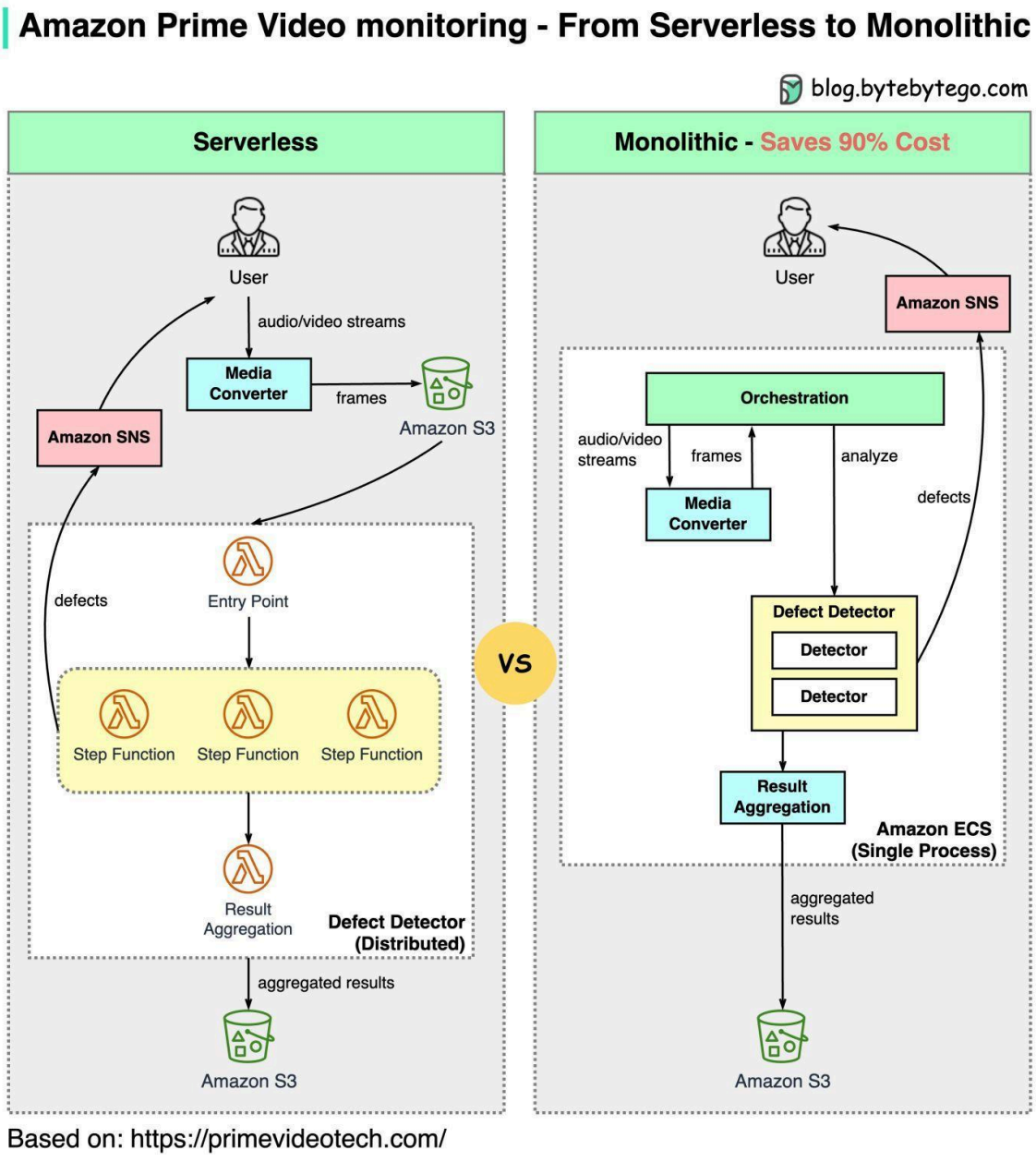
- suffix tree: for searching string in a document
- graph: for tracking friendship, or path finding
- r-tree: for finding the nearest neighbor
- vertex buffer: for sending data to GPU for rendering

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.



Based on: https://primevideotech.com/

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

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?