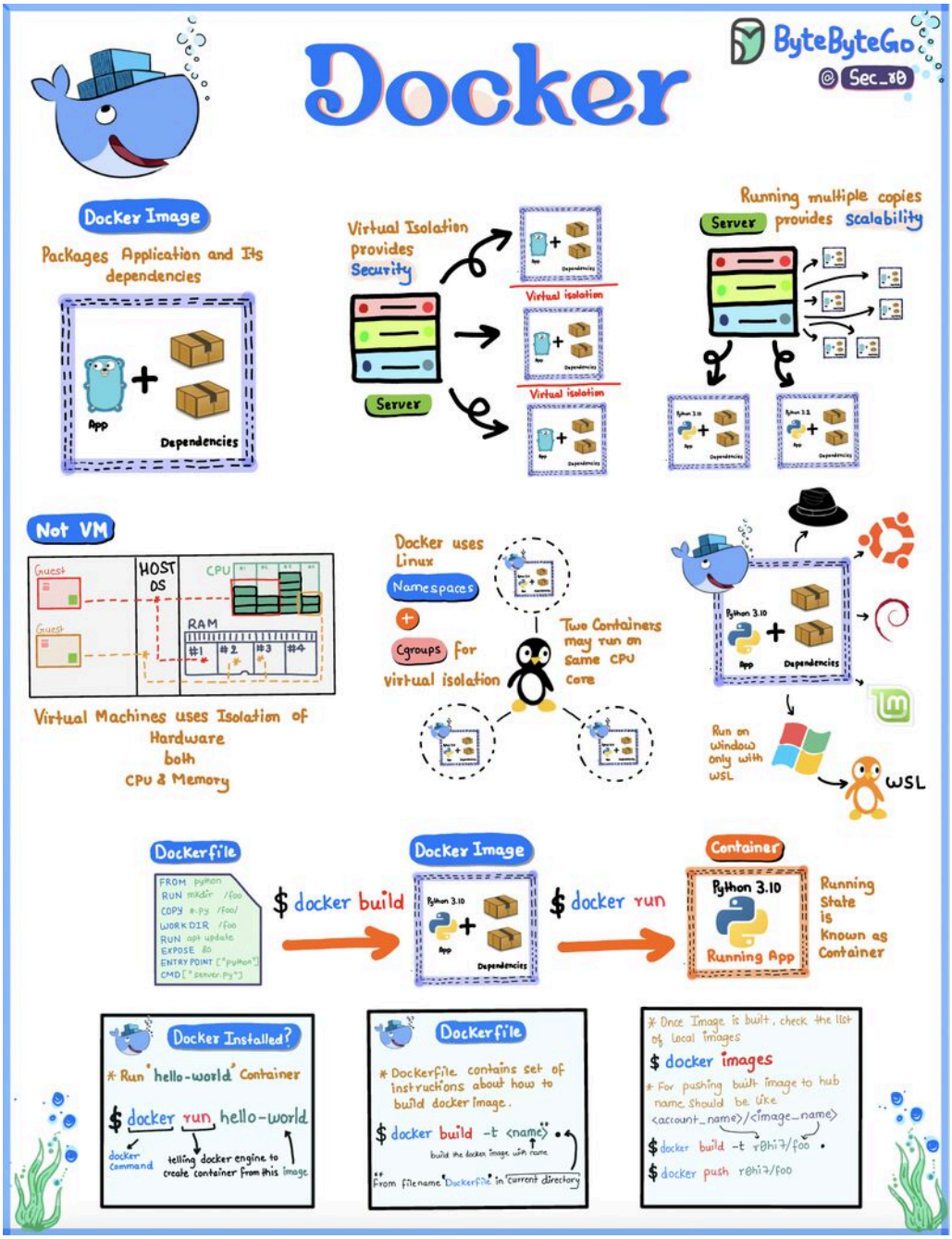
Over to you: HTTP status code 401 is for Unauthorized. Can you explain the difference between authentication and authorization, and which one does code 401 check for?

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

Docker 101: Streamlining App Deployment



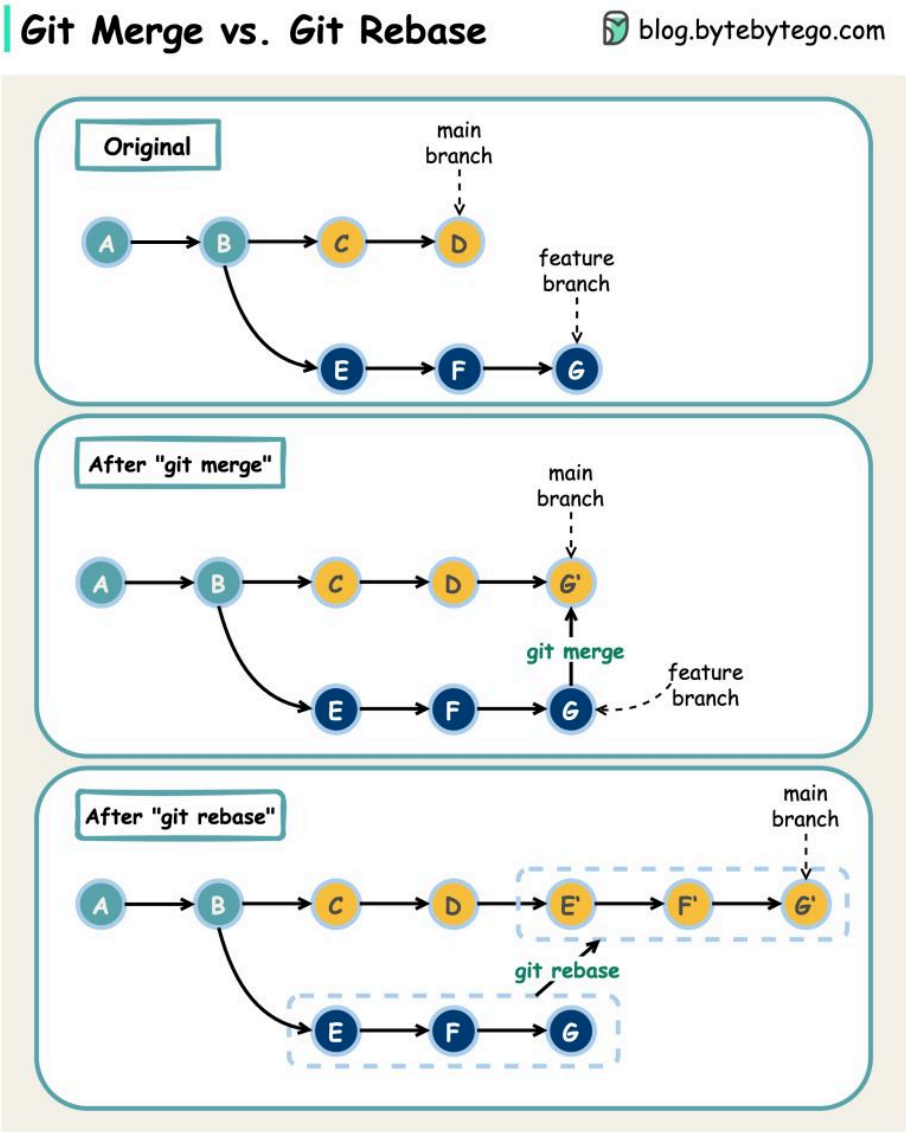
Fed up with the "it works on my machine" dilemma? Docker could be your salvation!

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**.

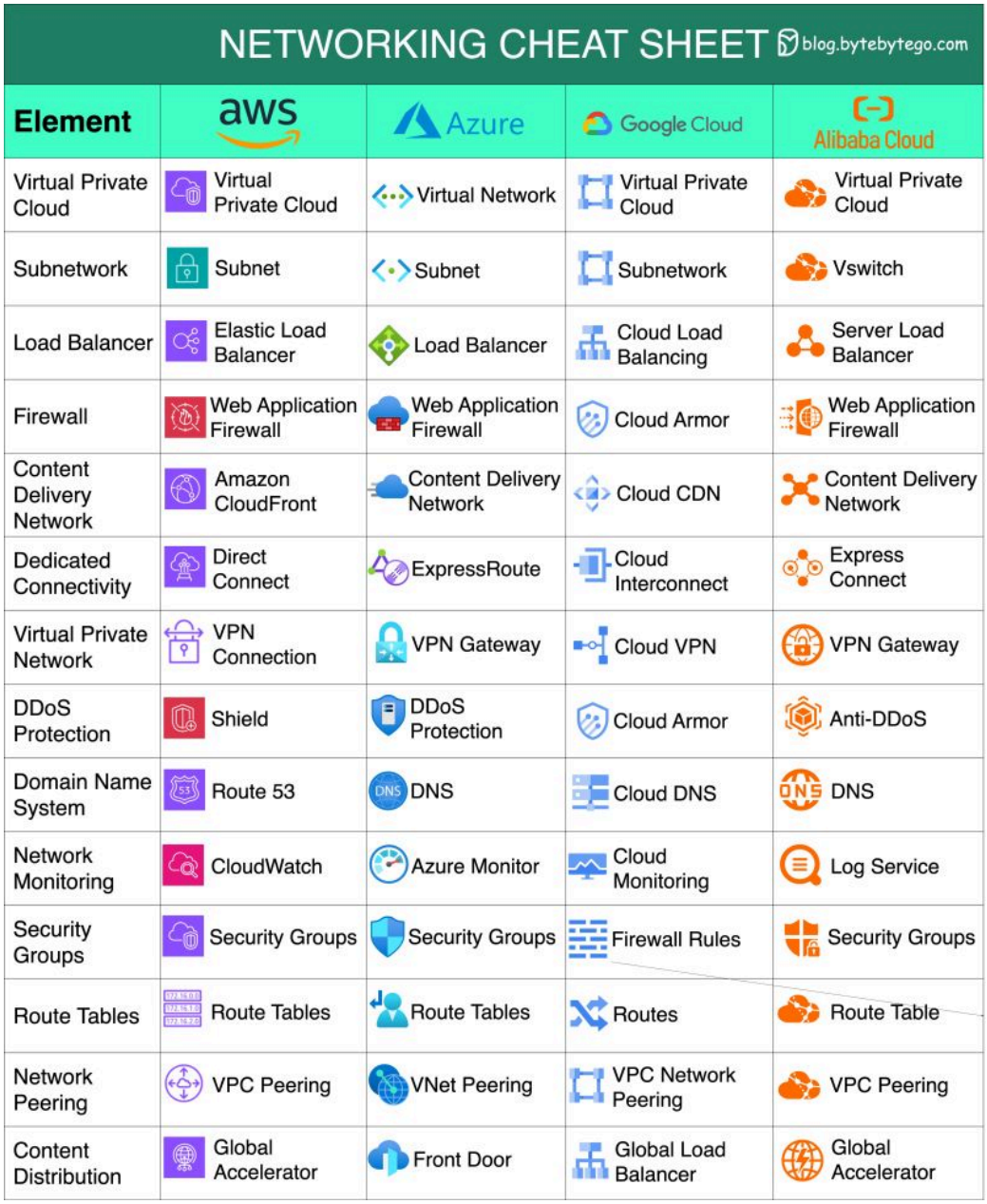
Rebase can be dangerous if "the golden rule of git rebase" is not followed.

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.



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.

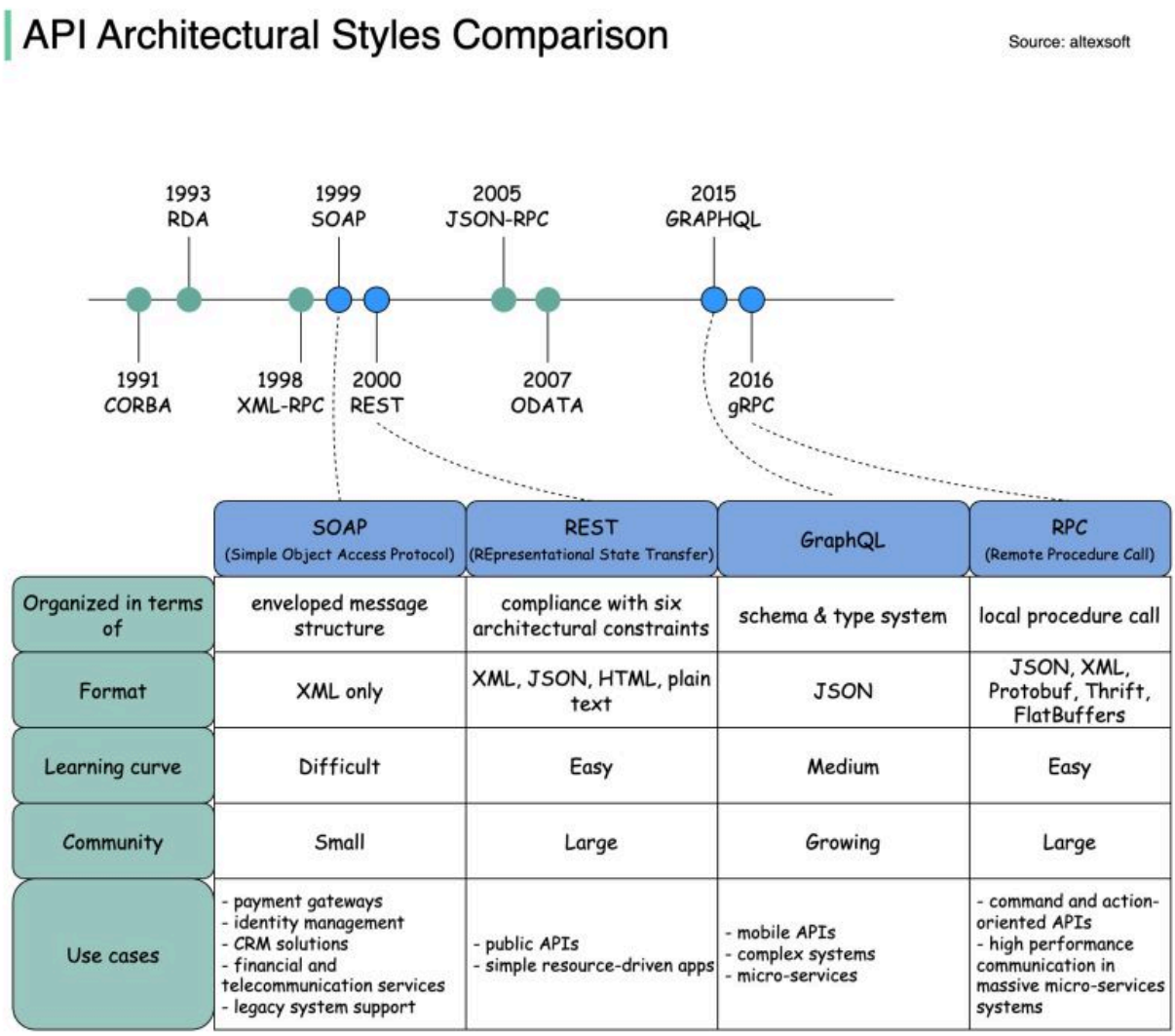
In essence, these network elements work together to create a robust and responsive cloud ecosystem that underpins modern digital services and applications.

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.



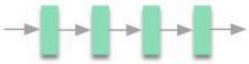
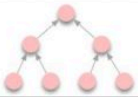
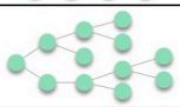
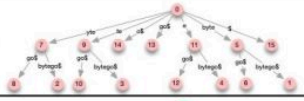
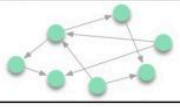
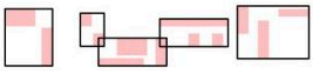
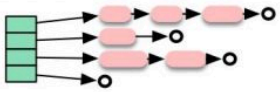
	SOAP (Simple Object Access Protocol)	REST (REpresentational State Transfer)	GraphQL	RPC (Remote Procedure Call)
Organized in terms of	enveloped message structure	compliance with six architectural constraints	schema & type system	local procedure call
Format	XML only	XML, JSON, HTML, plain text	JSON	JSON, XML, Protobuf, Thrift, FlatBuffers
Learning curve	Difficult	Easy	Medium	Easy
Community	Small	Large	Growing	Large
Use cases	- payment gateways - identity management - CRM solutions - financial and telecommunication services - legacy system support	- public APIs - simple resource-driven apps	- mobile APIs - complex systems - micro-services	- command and action-oriented APIs - high performance communication in massive micro-services systems

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

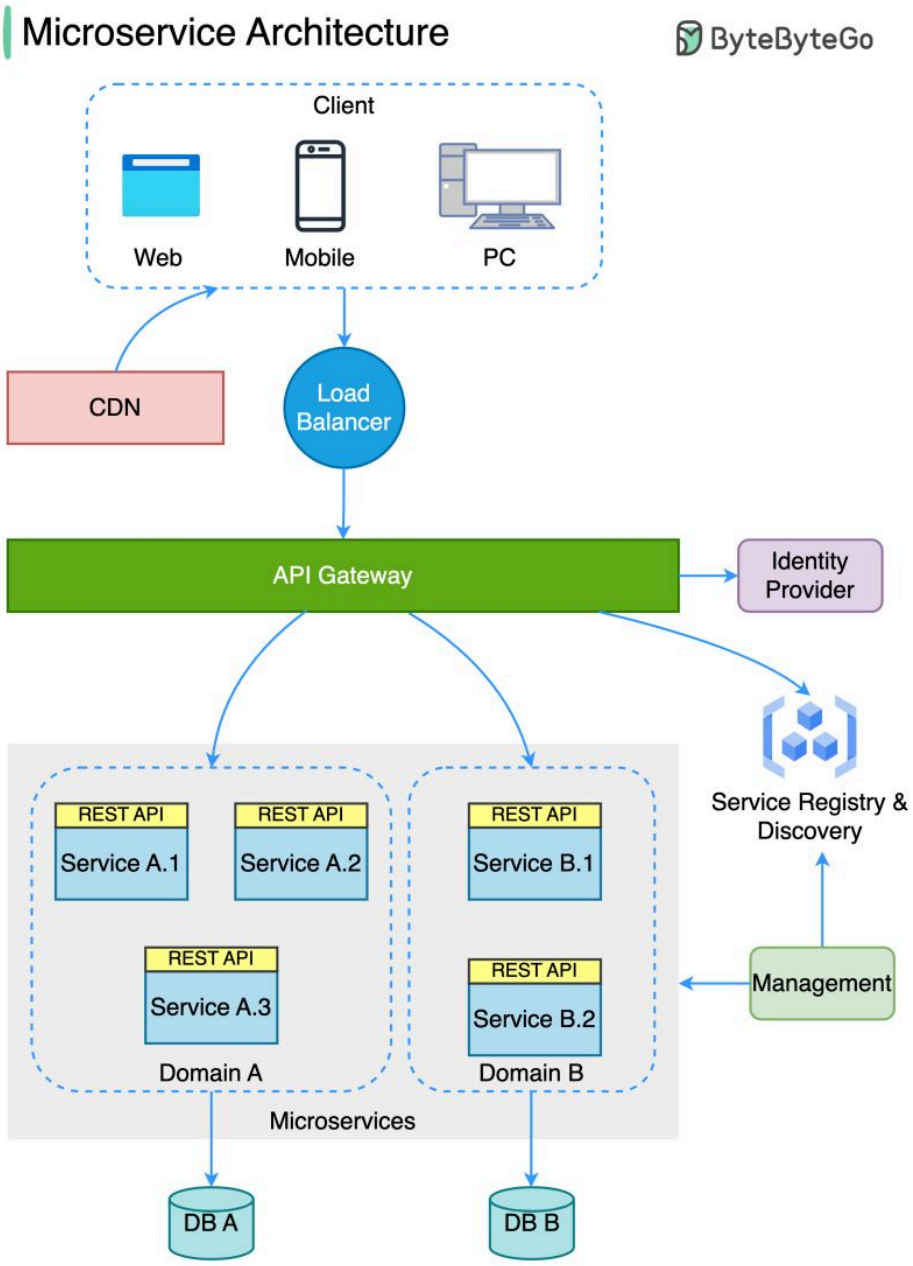
- list: keep your Twitter feeds
- stack: support undo/redo of the word editor
- queue: keep printer jobs, or send user actions in-game
- hash table: cashing systems
- Array: math operations
- heap: task scheduling
- tree: keep the HTML document, or for AI decision
- suffix tree: for searching string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

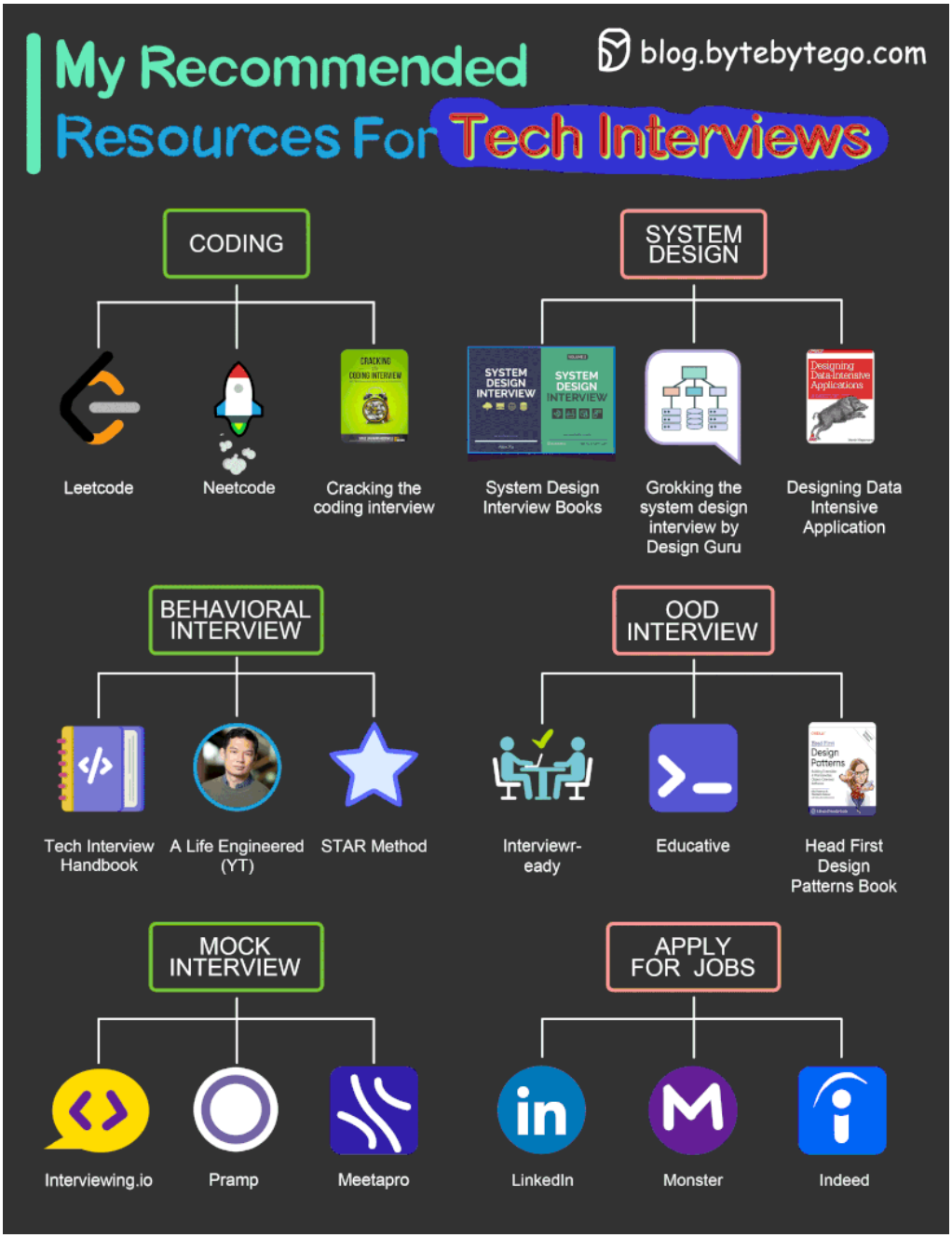
Benefits of microservices:

- 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

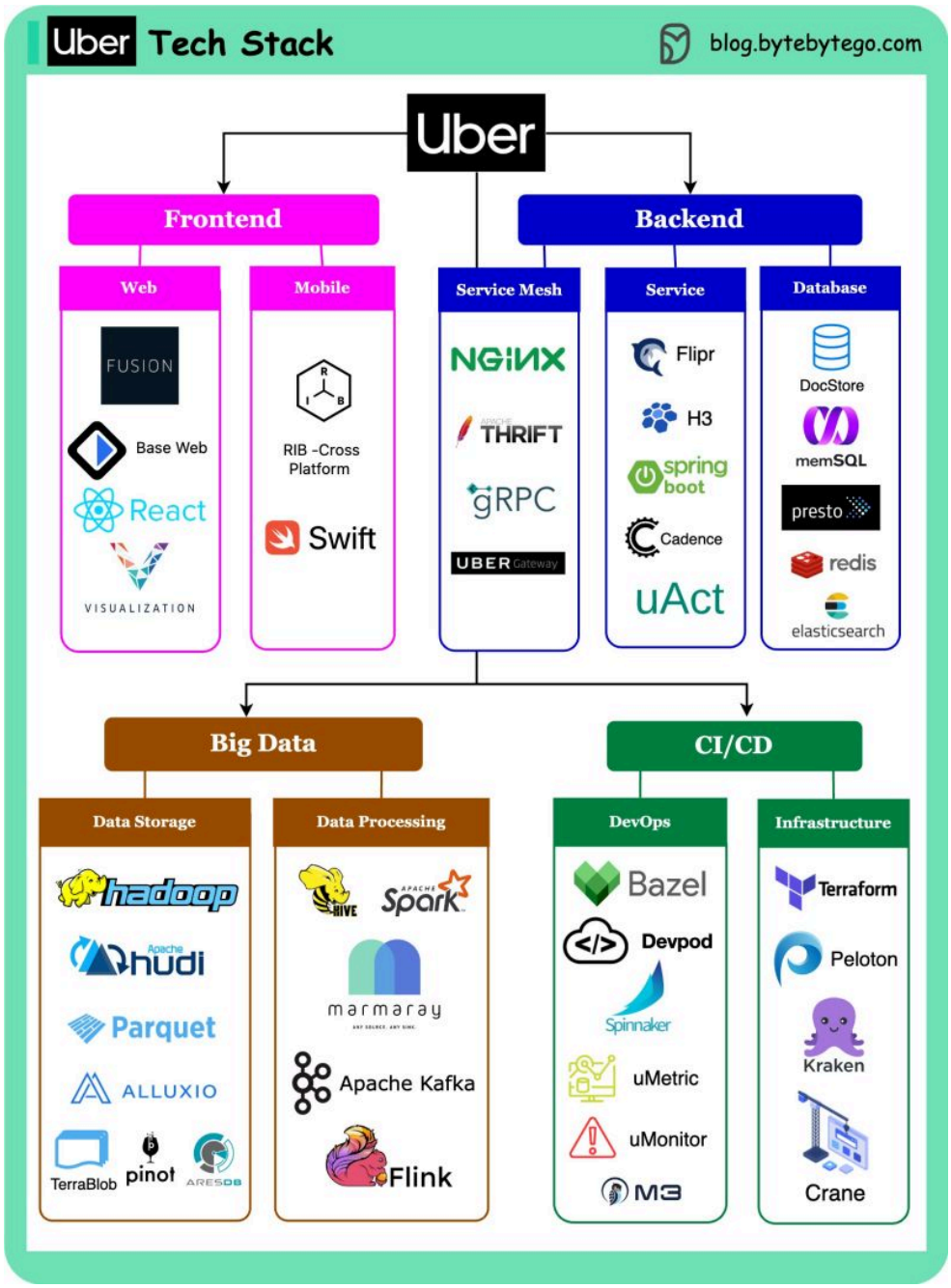
Apply for Jobs

- 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 Flipr (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.

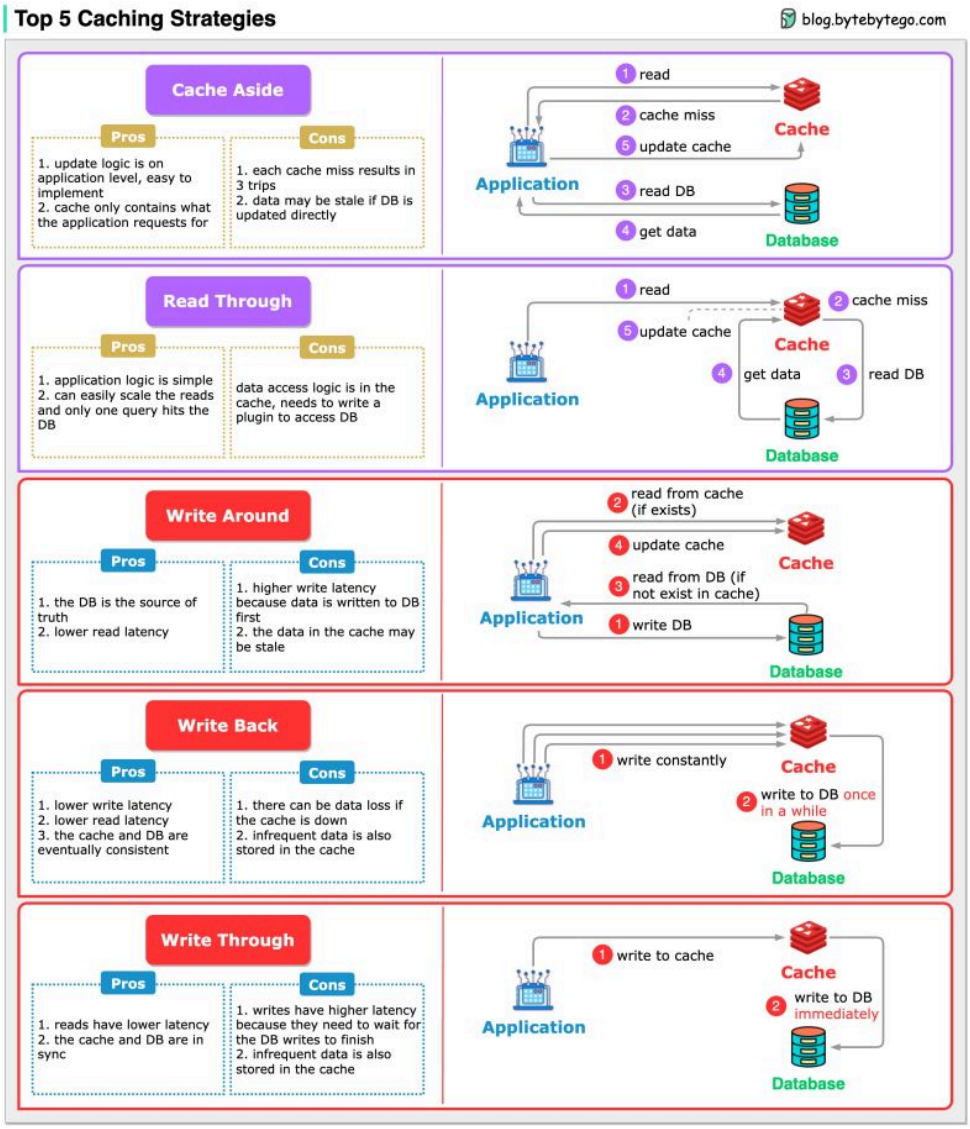
Big data: managed through the Hadoop family. Hudi and Parquet are used as file formats, and Alluxio serves as cache. Time-series data is stored in Pinot and AresDB.

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:

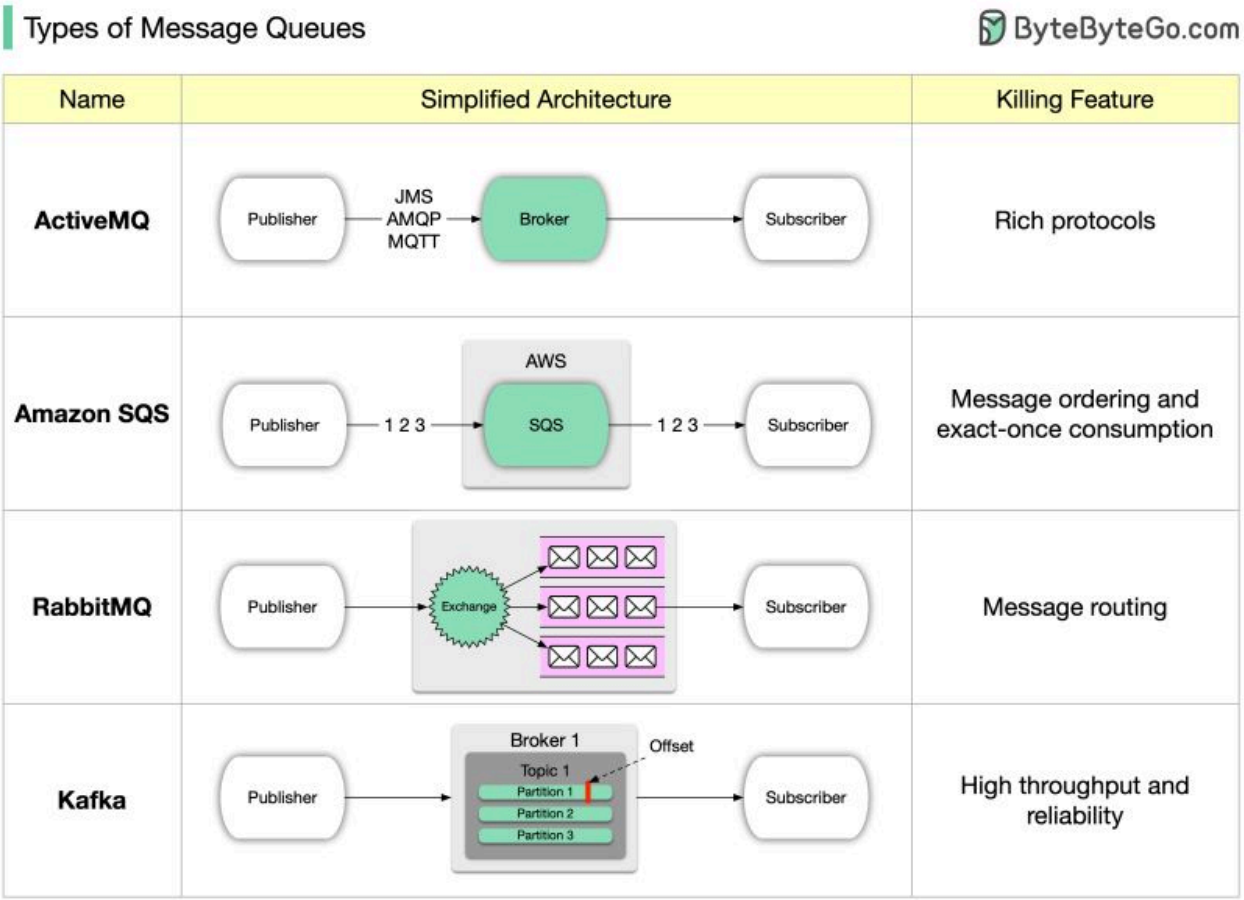
  Write around

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?



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

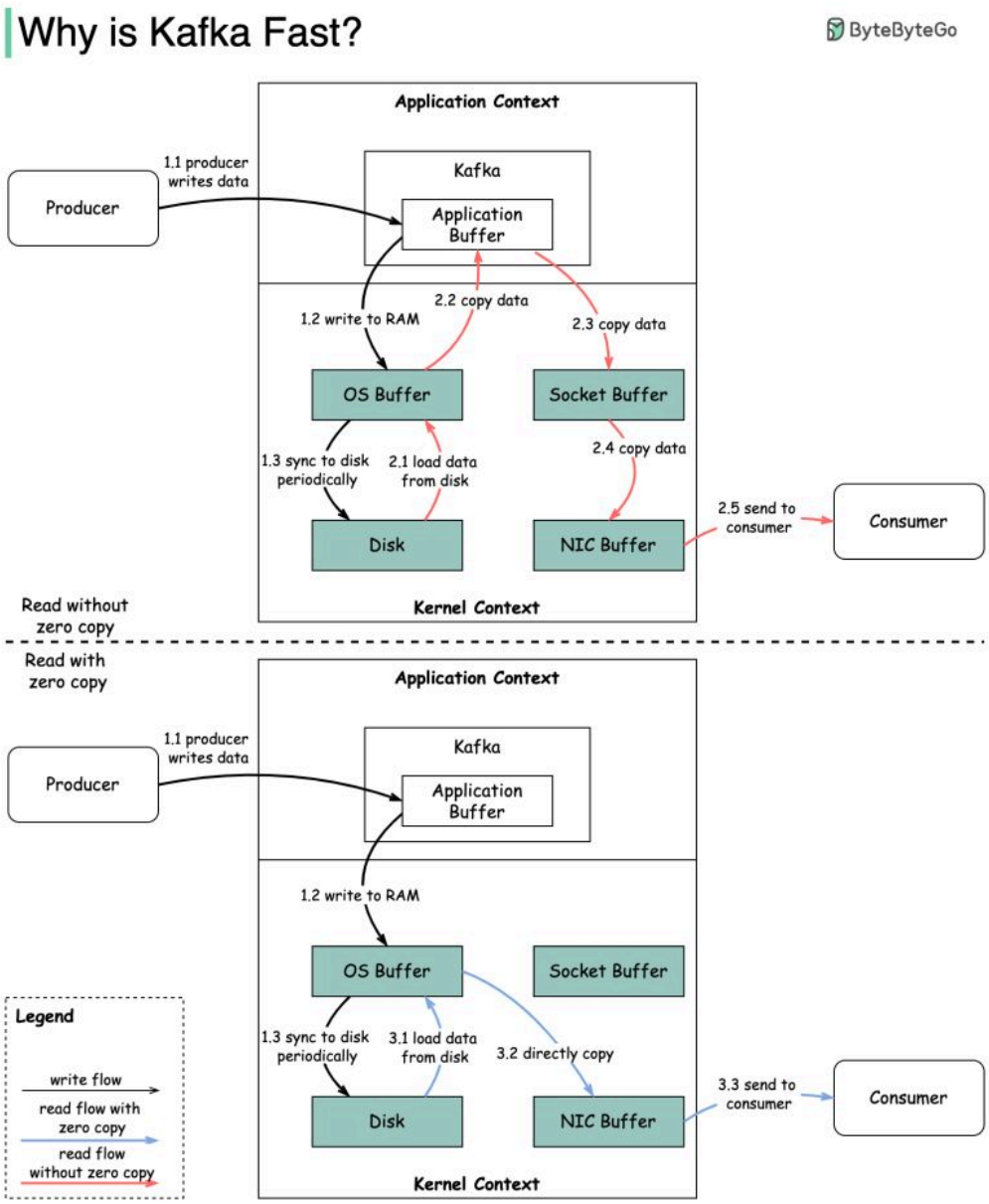
Try out a message queue and practice sending and receiving messages until you're comfortable. Choose an easy one like Kafka and experiment with sending and receiving messages. Read books or take online courses as you get more comfortable. Build little projects and learn from those who have already been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

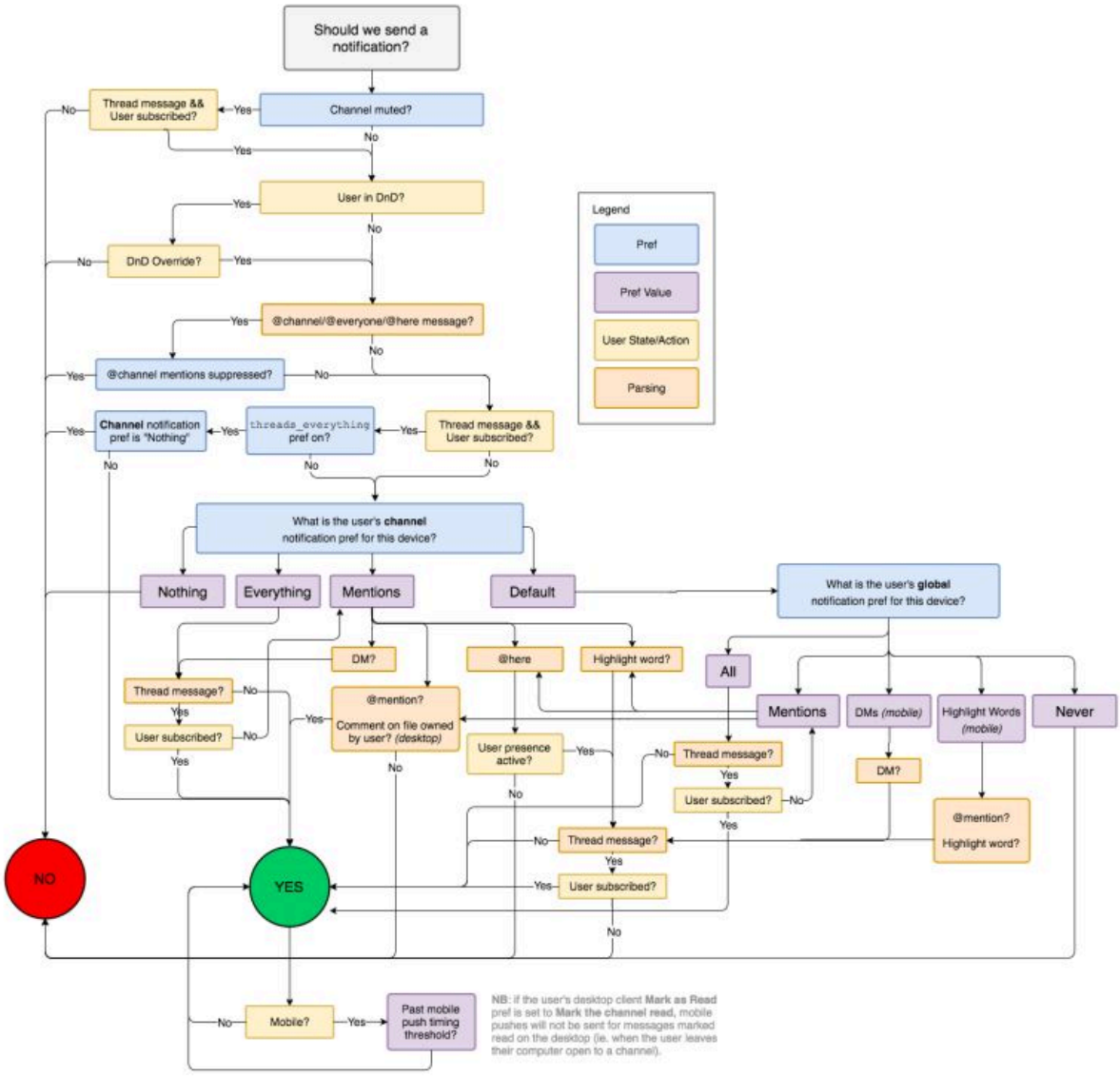
- Step 2: Consumer reads data without zero-copy
  2.1: The data is loaded from disk to OS cache
  2.2 The data is copied from OS cache to Kafka application
  2.3 Kafka application copies the data into the socket buffer
  2.4 The data is copied from socket buffer to network card
  2.5 The network card 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.

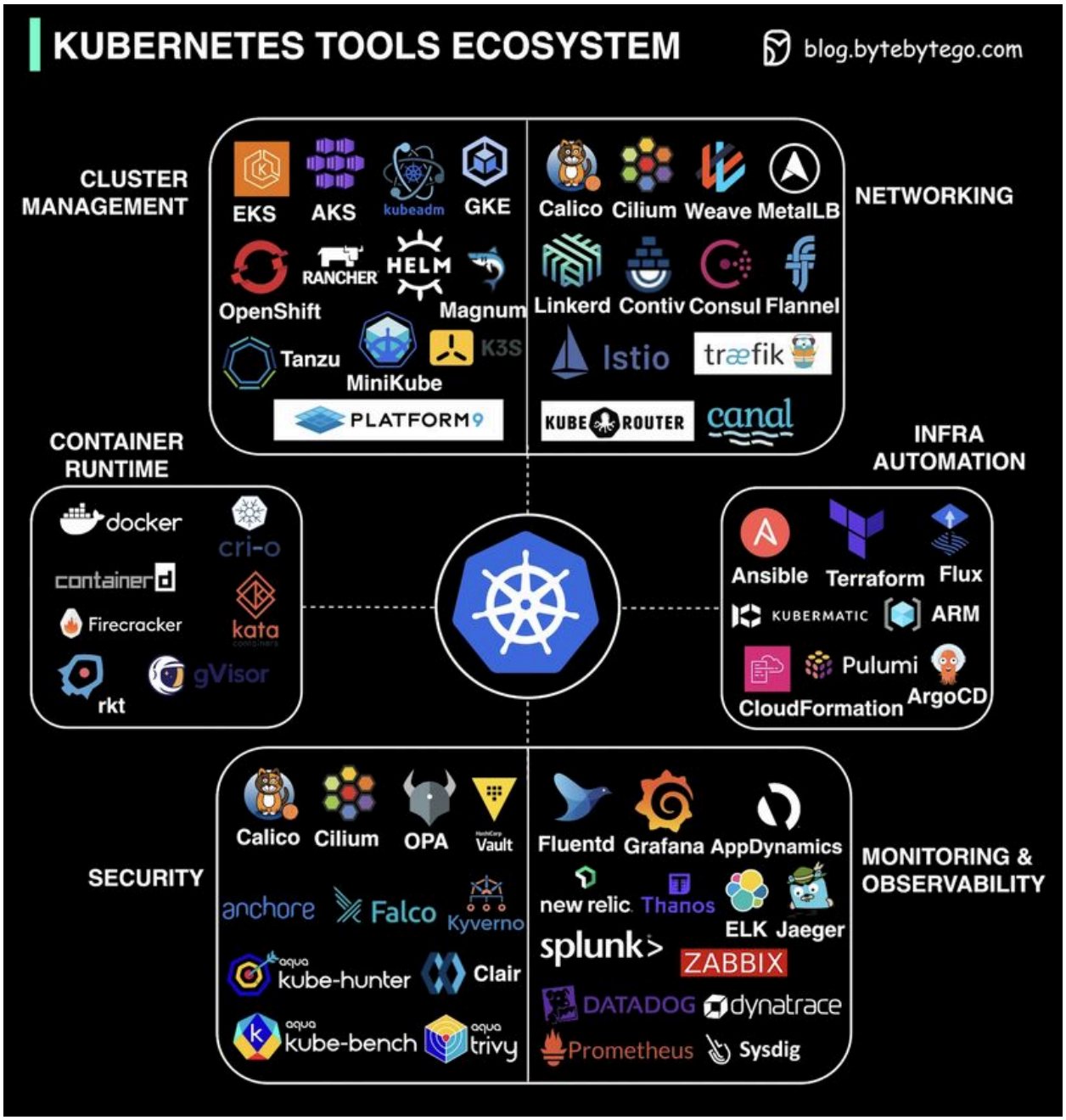
When we have a great design, users may not notice the complexity because it feels like the feature is just working as intended.

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:

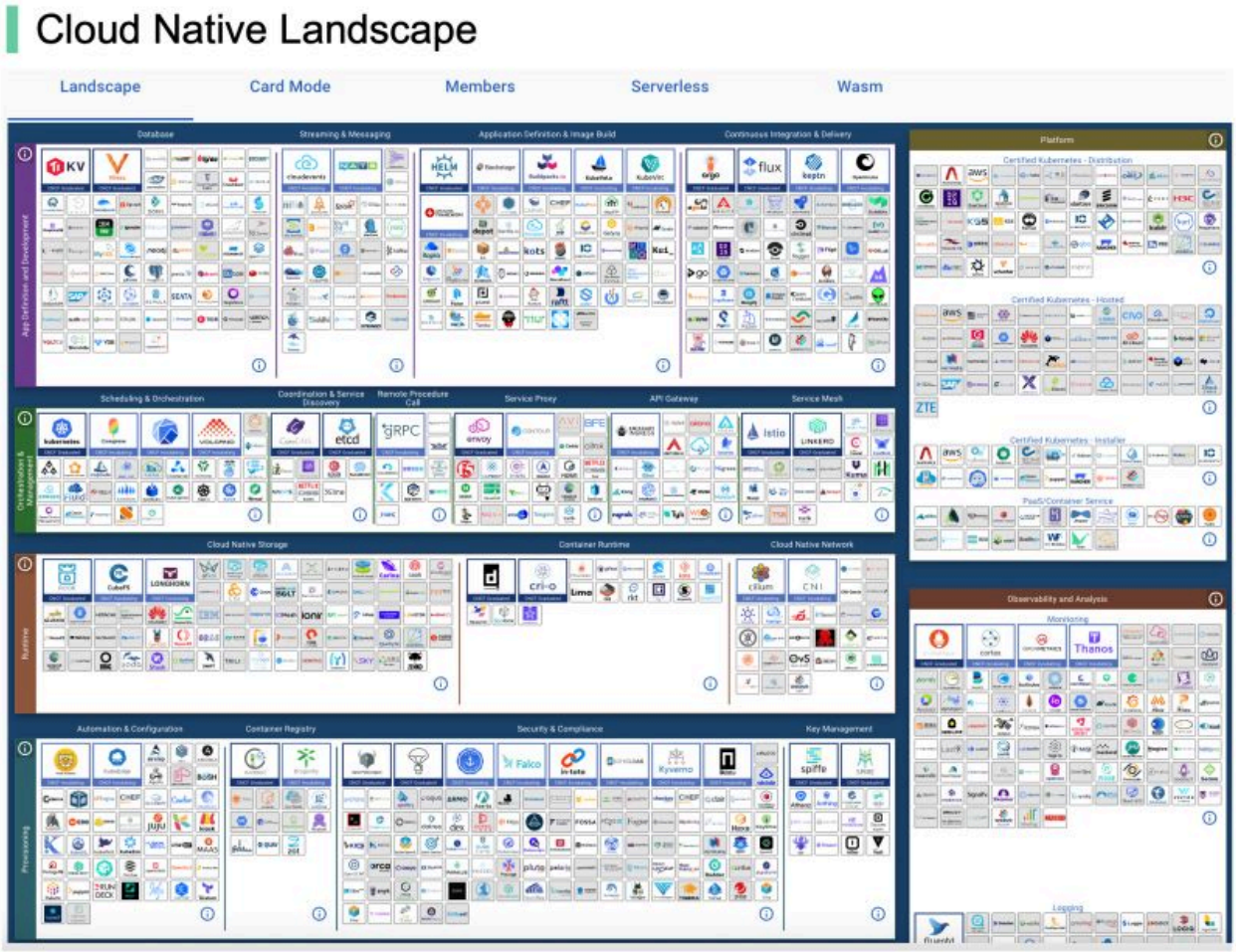
How have Kubernetes tools enhanced your containerized application management?

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.

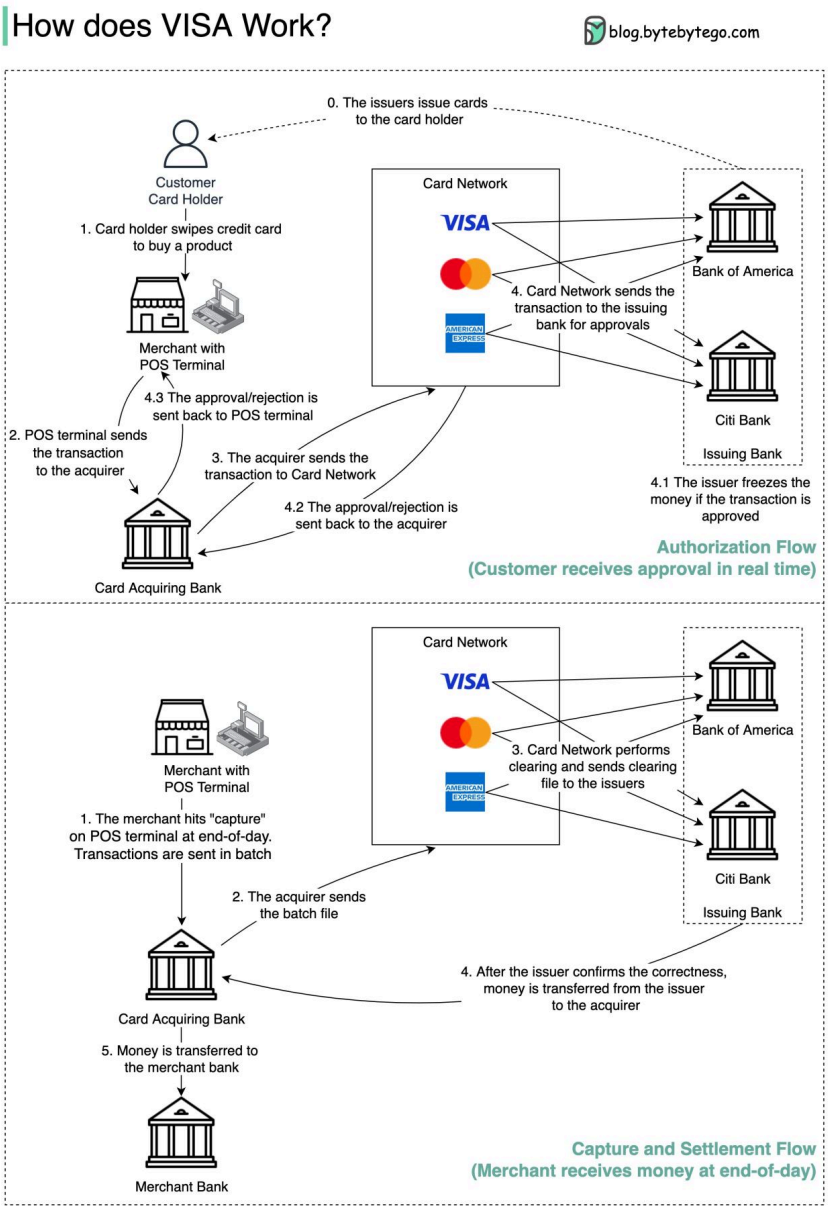
Over to you all: What do you think is causing this fragmentation in tech stack choices?

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.

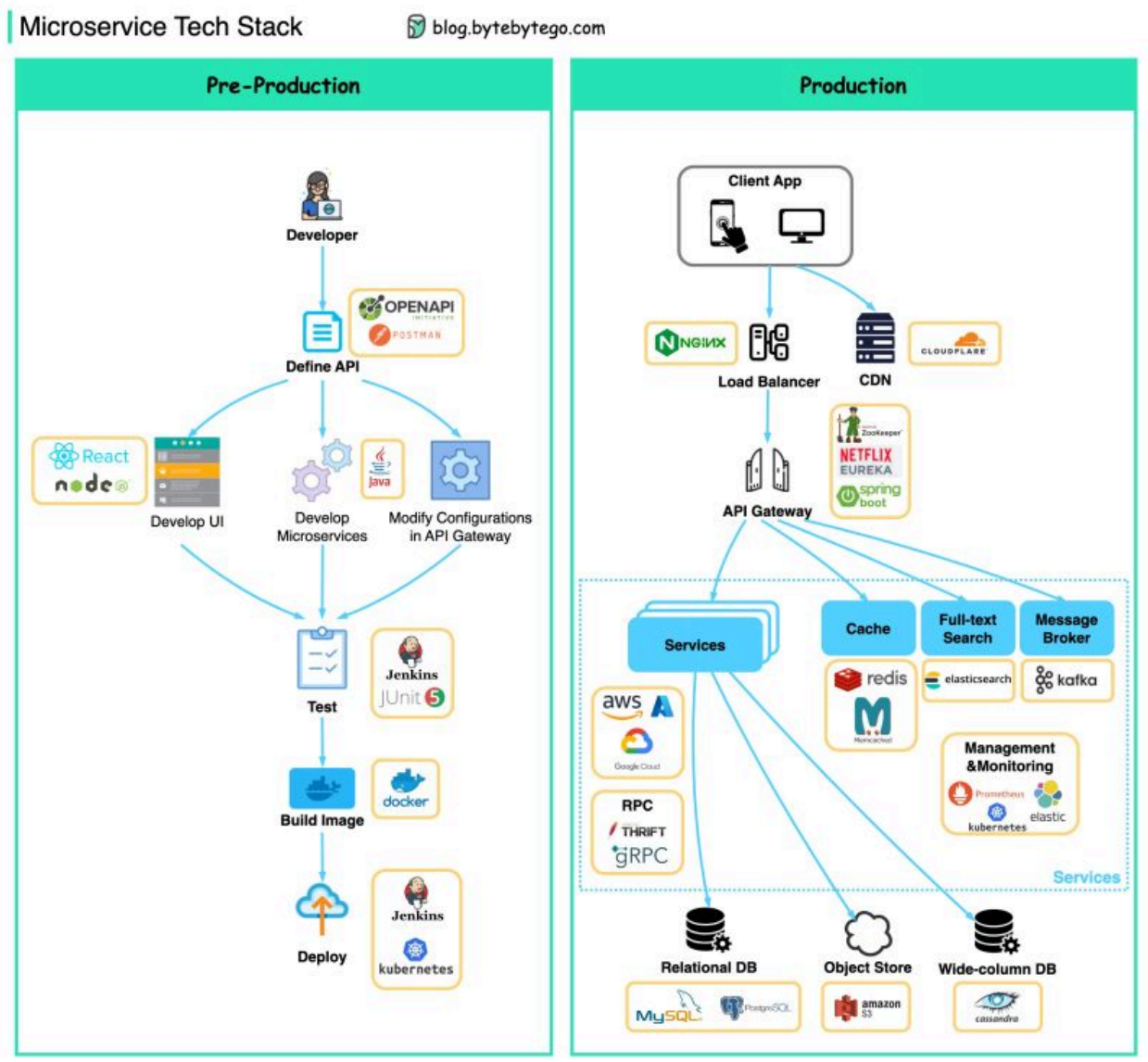
In the process, the card network takes on the burden of talking to each bank and receives service fees in return.

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.

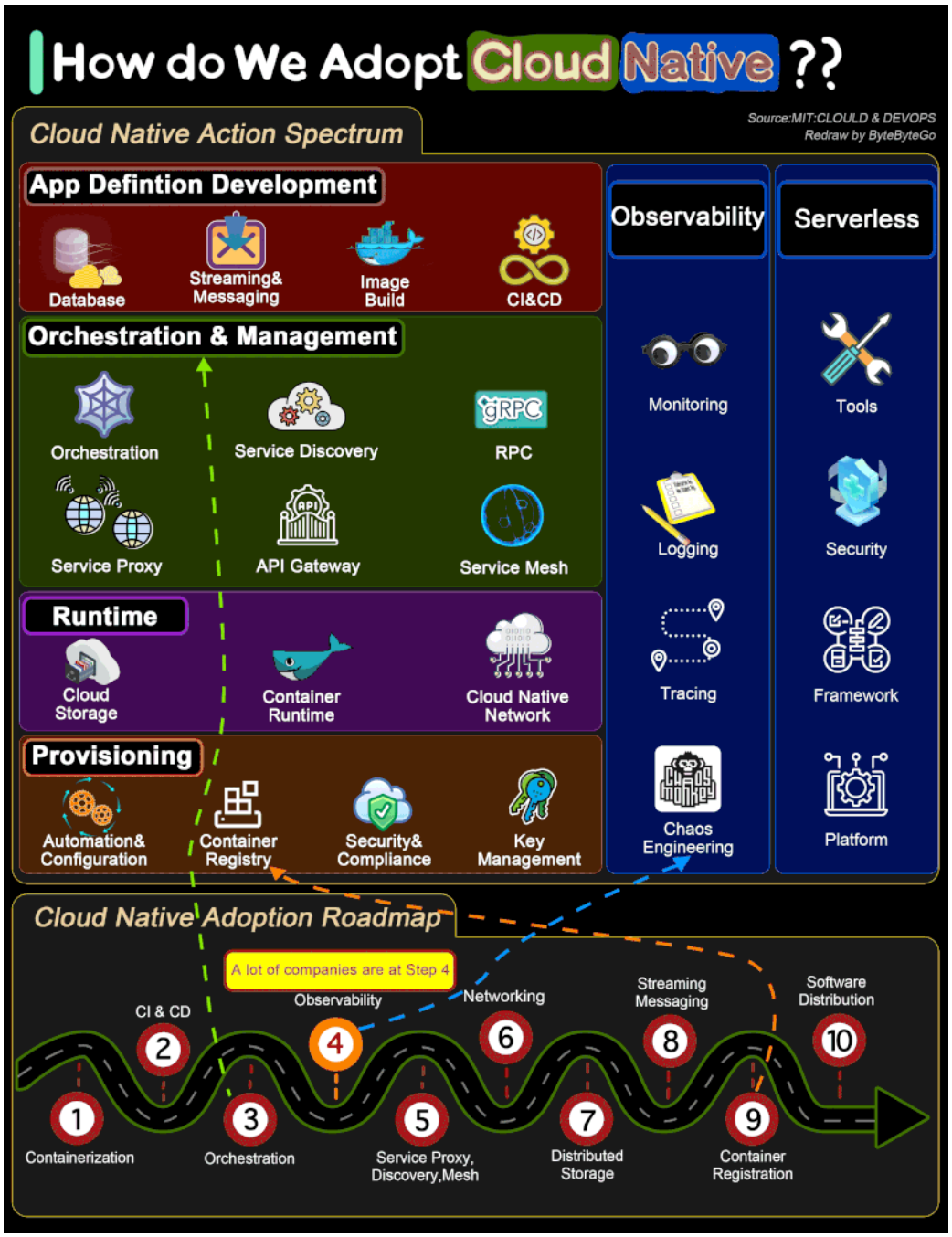
▶️ **Production**

- 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:

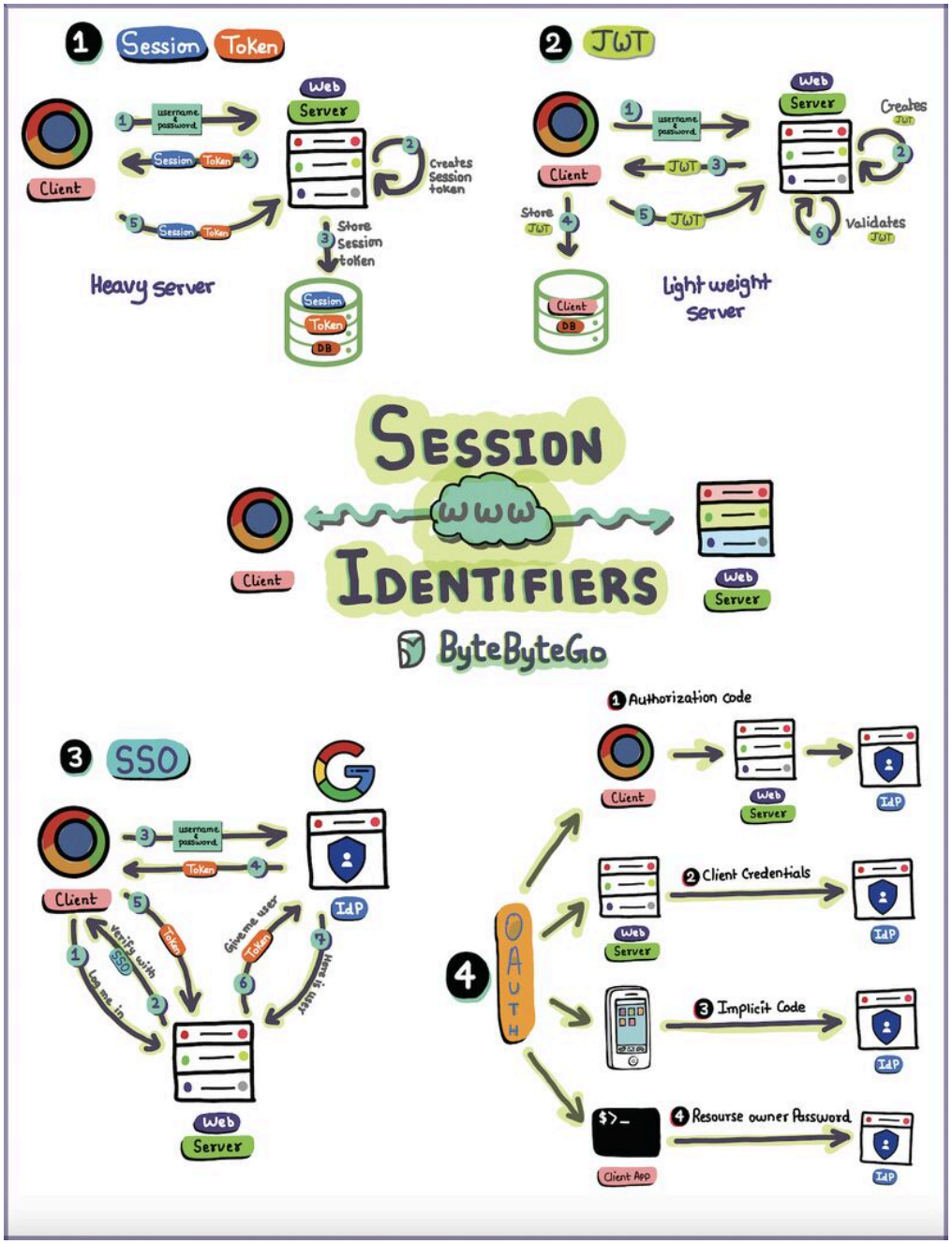
1. Application definition development
2. Orchestration and management

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

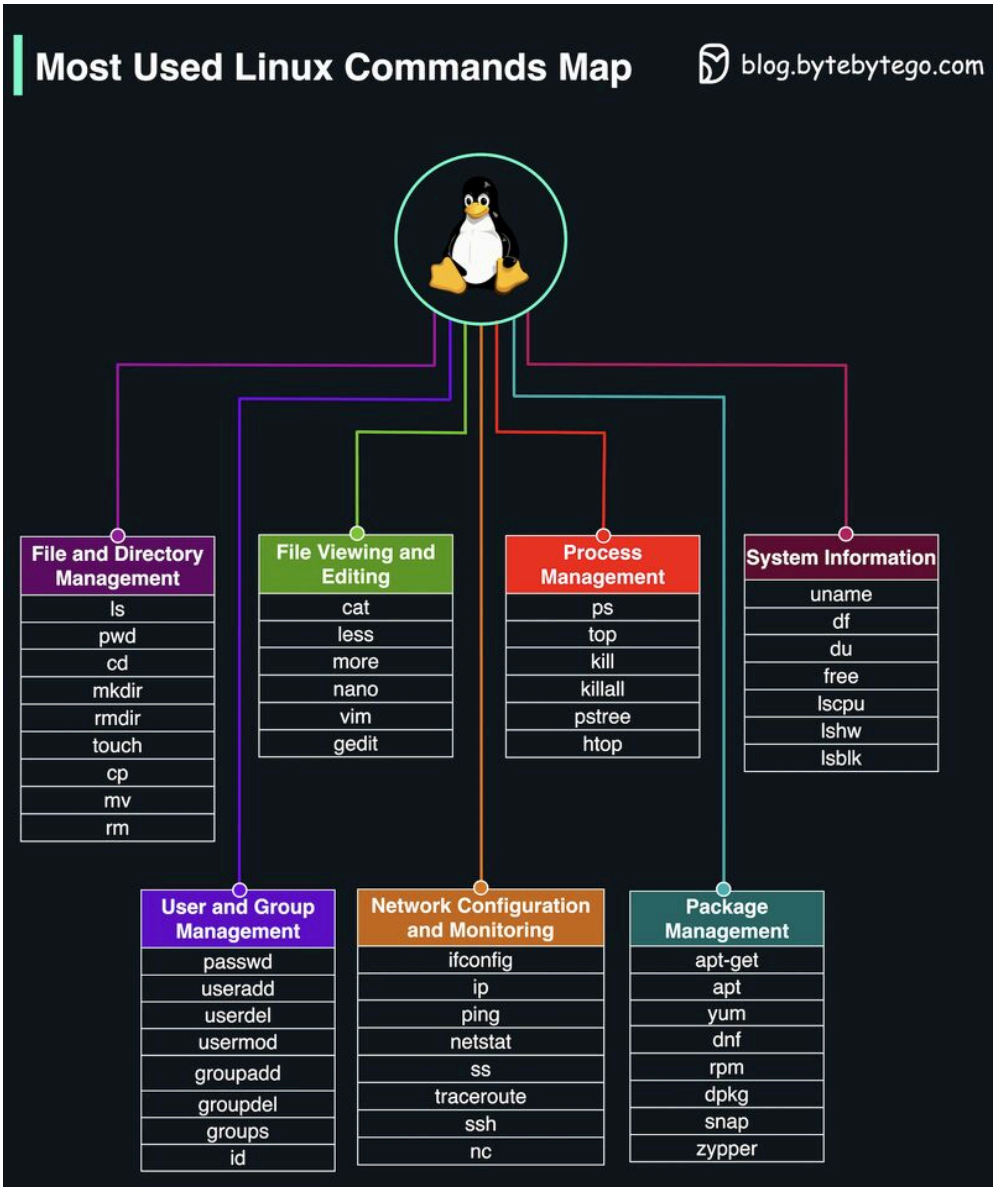
Explaining Sessions, Tokens, JWT, SSO, and OAuth in One Diagram



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

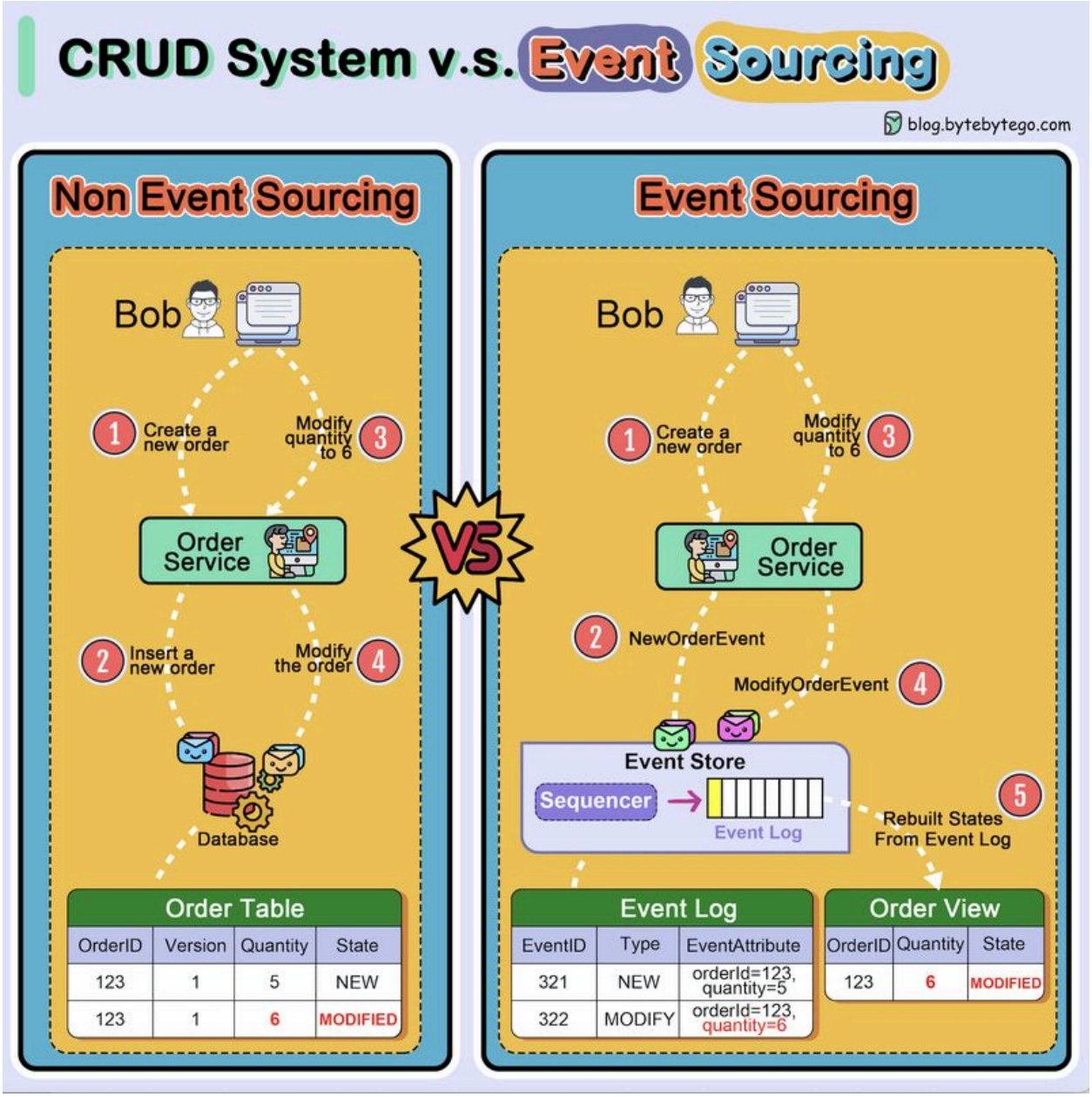


1. File and Directory Management
2. File Viewing and Editing
3. Process Management
4. System Information
5. User and Group Management
6. Network Configuration and Monitoring
7. Package Management

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.

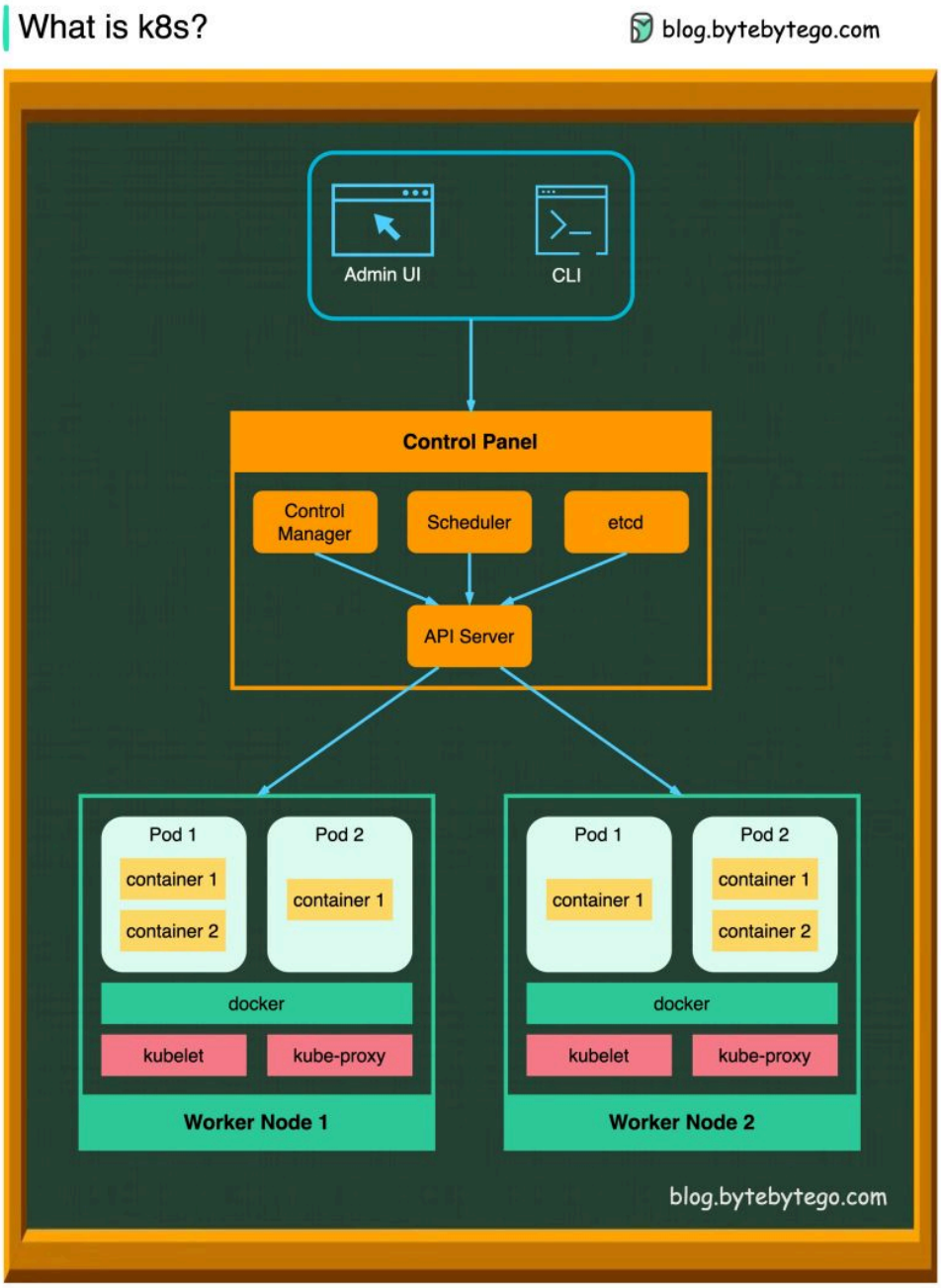
  Steps 3 and 4: Bob wants to change the quantity from 5 to 6. A ModifyOrderEvent is created, sequenced, and persisted in the event store with eventID=322.

  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.

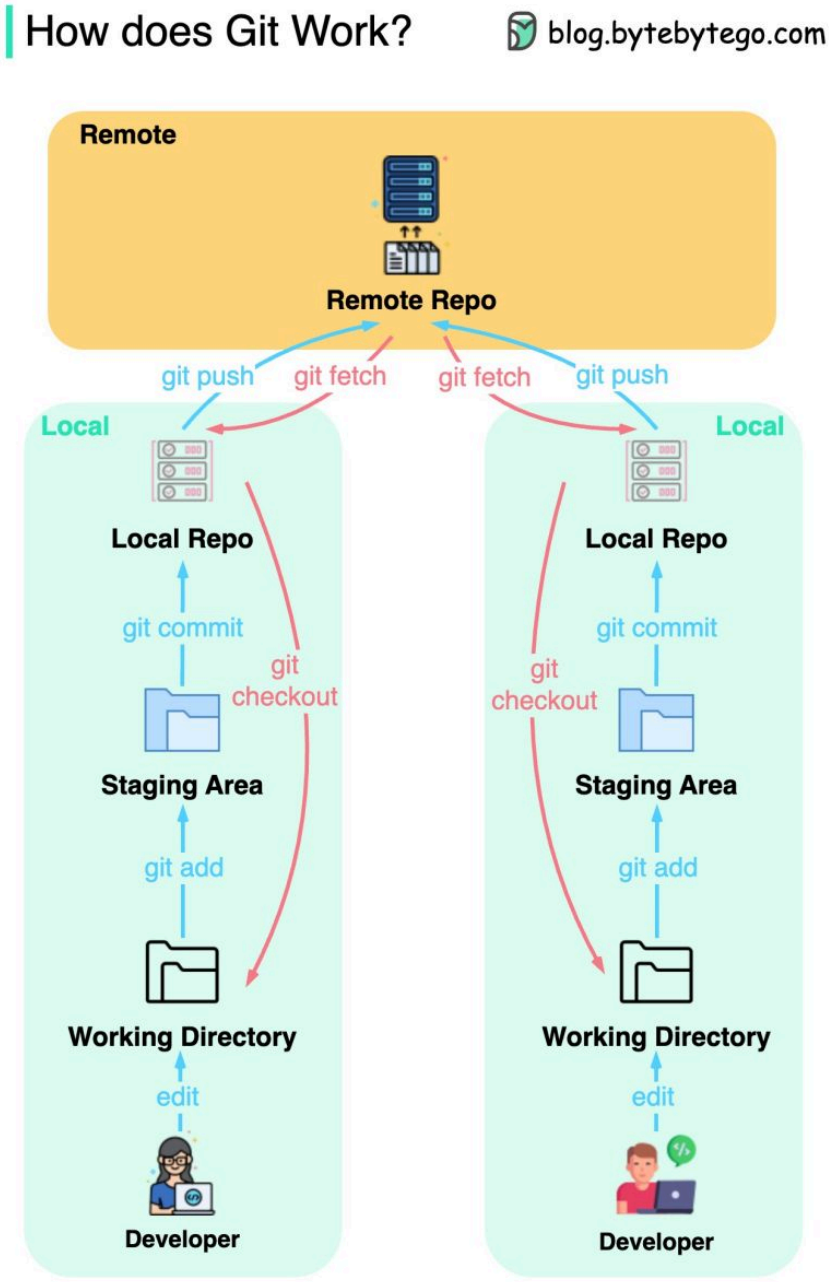
2. Kubelet
   An agent that runs on each node in the cluster. It ensures containers are running in a Pod.

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.



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?

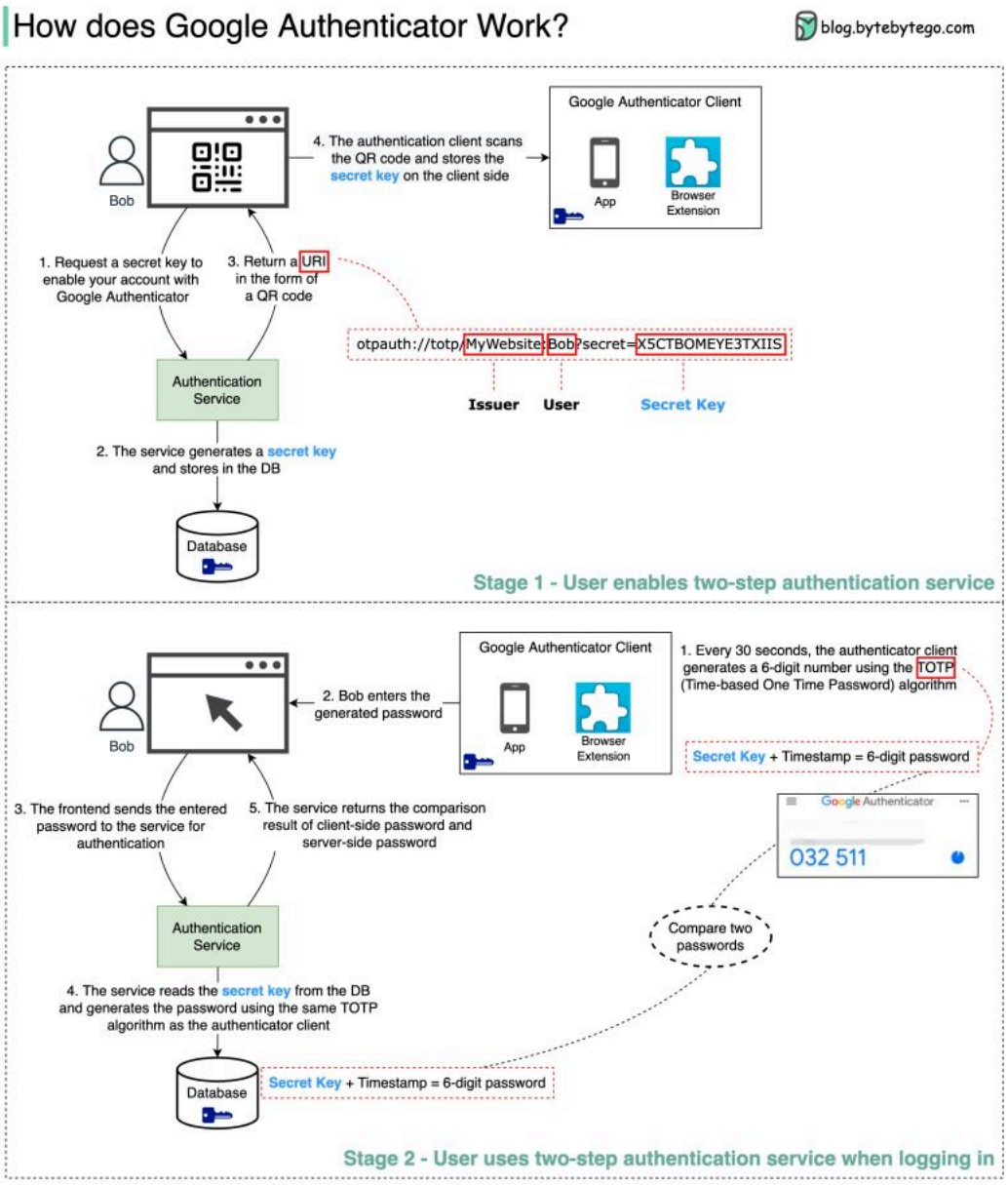
Google authenticator is commonly used for logging into our accounts when 2-factor authentication is enabled. How does it guarantee security?

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.



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.

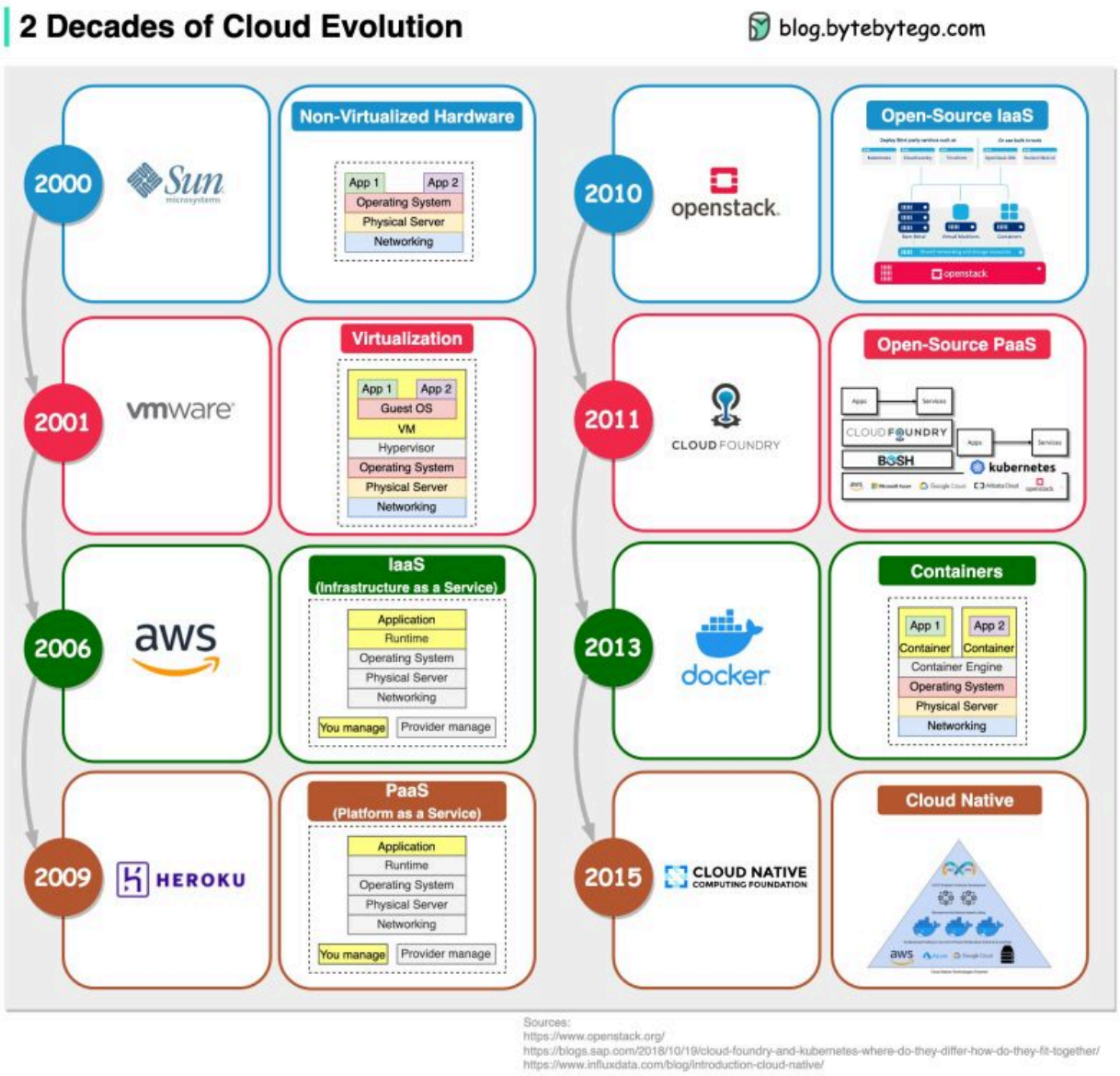
- Can the 6-digit password be guessed by hackers?

- 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.

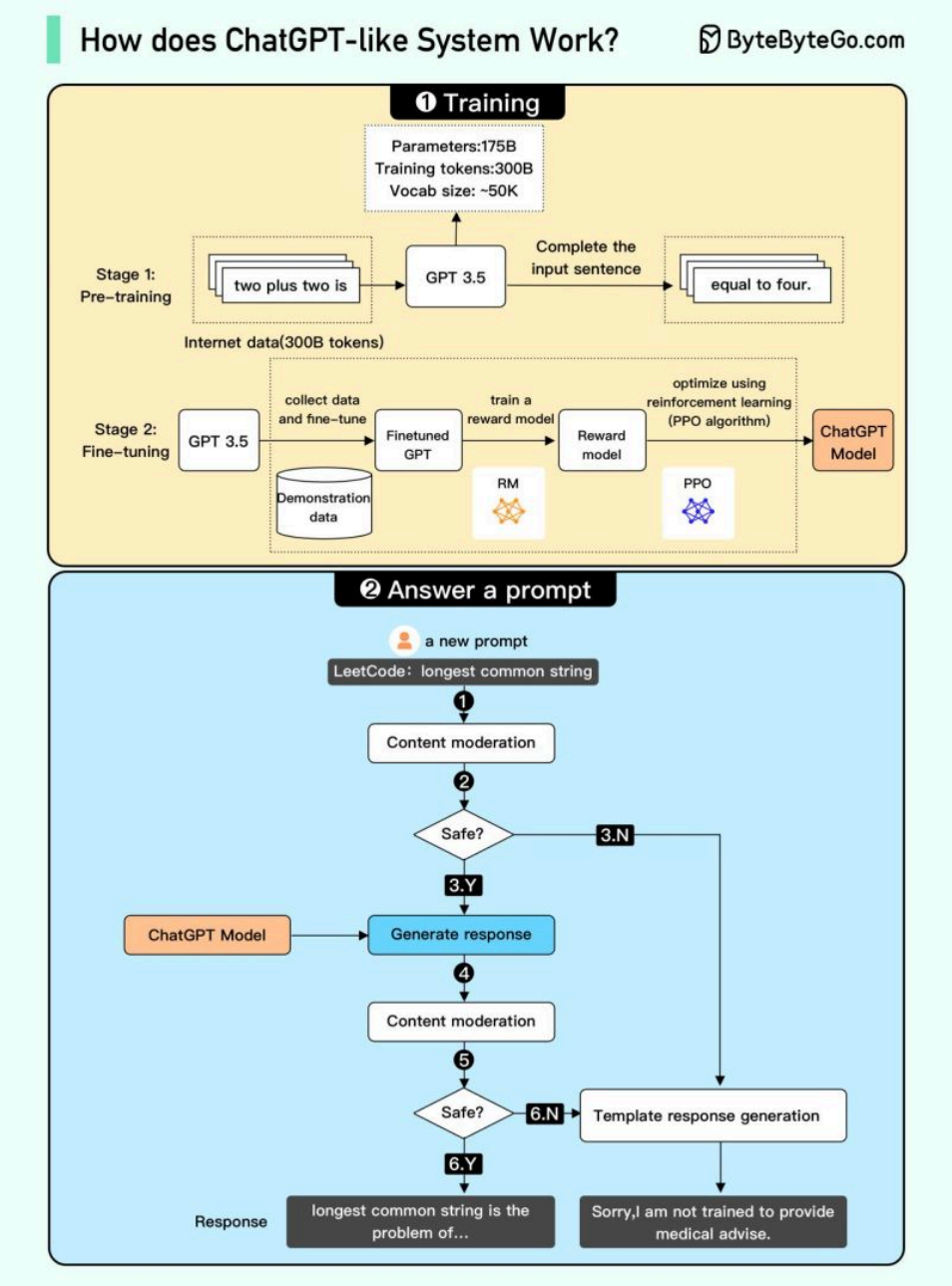


2001 - VMWare - Virtualization via hypervisor
2006 - AWS - IaaS (Infrastructure as a Service)
2009 - Heroku - PaaS (Platform as a Service)
2010 - OpenStack - Open-source IaaS
2011 - CloudFoundry - Open-source PaaS
2013 - Docker - Containers
2015 - CNCF (Cloud Native Computing Foundation) - Cloud Native

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.

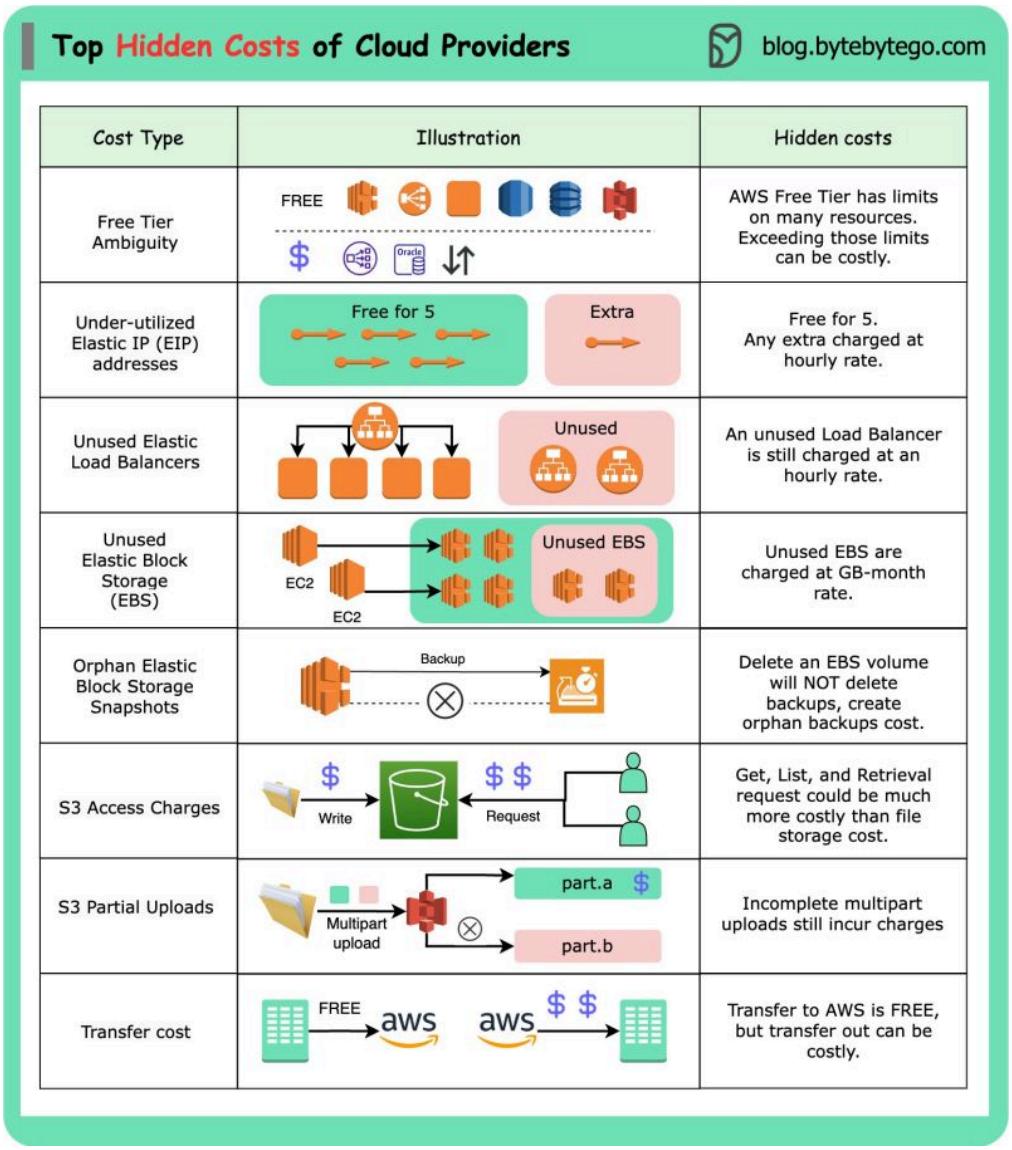
Top Hidden Costs of Cloud Providers

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.



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.

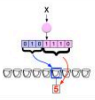
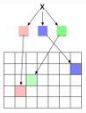
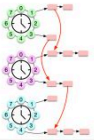
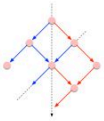
Over to you: Have you ever been surprised by an unexpected cloud bill? Share your experiences with us!

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

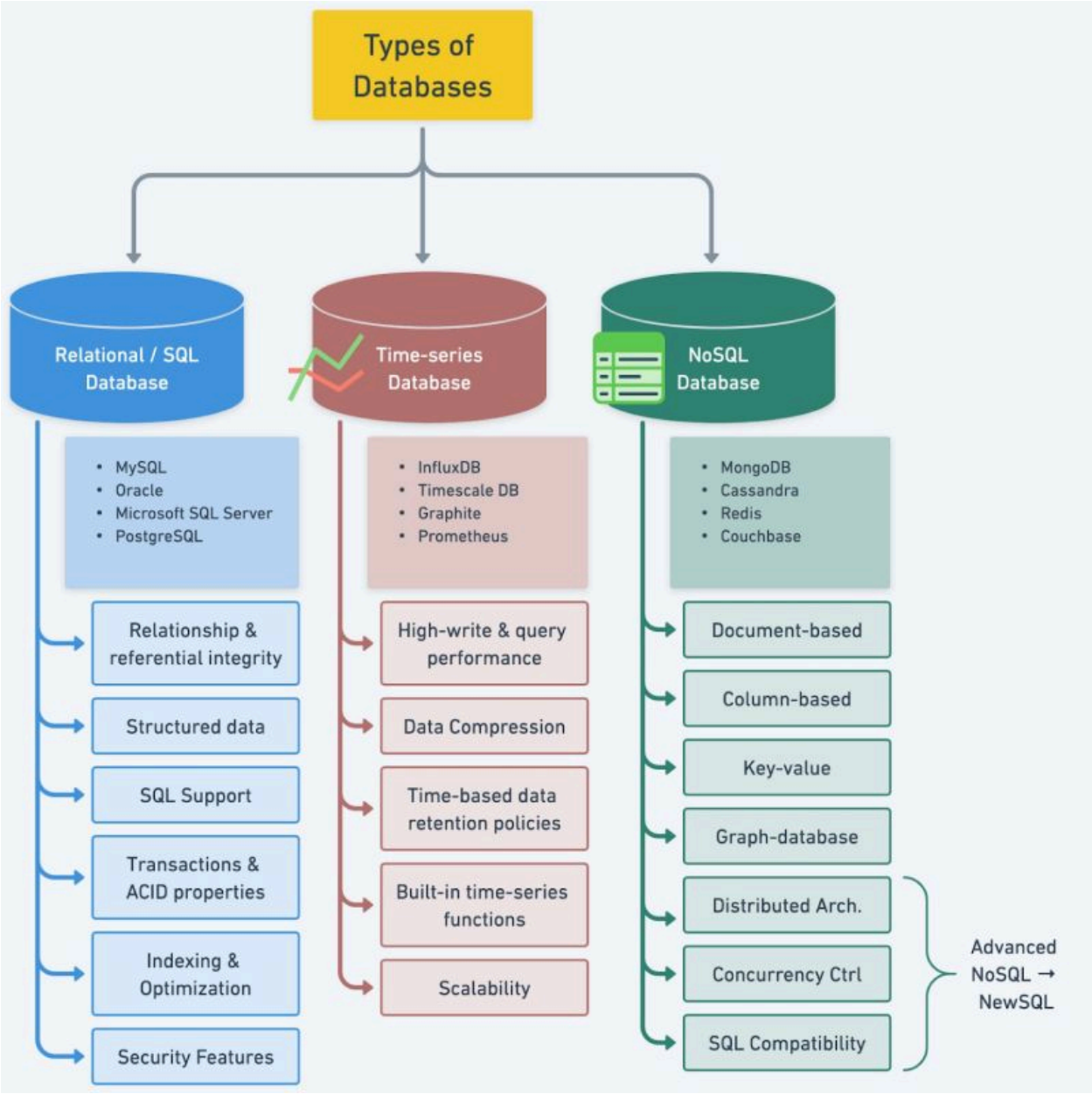
Algorithm	How it Works	Priority	Use Cases
Geohash		★★★★★	Location based service
Quadtree		★★★★★	Location based service
Consistent Hashing		★★★★★	Balance the load within a cluster of services
Leaky bucket		★★★★★	Rate limiter
Token bucket		★★★★★	Rate limiter
Trie		★★★★★	Search autocomplete
Rsync		★★★☆☆	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

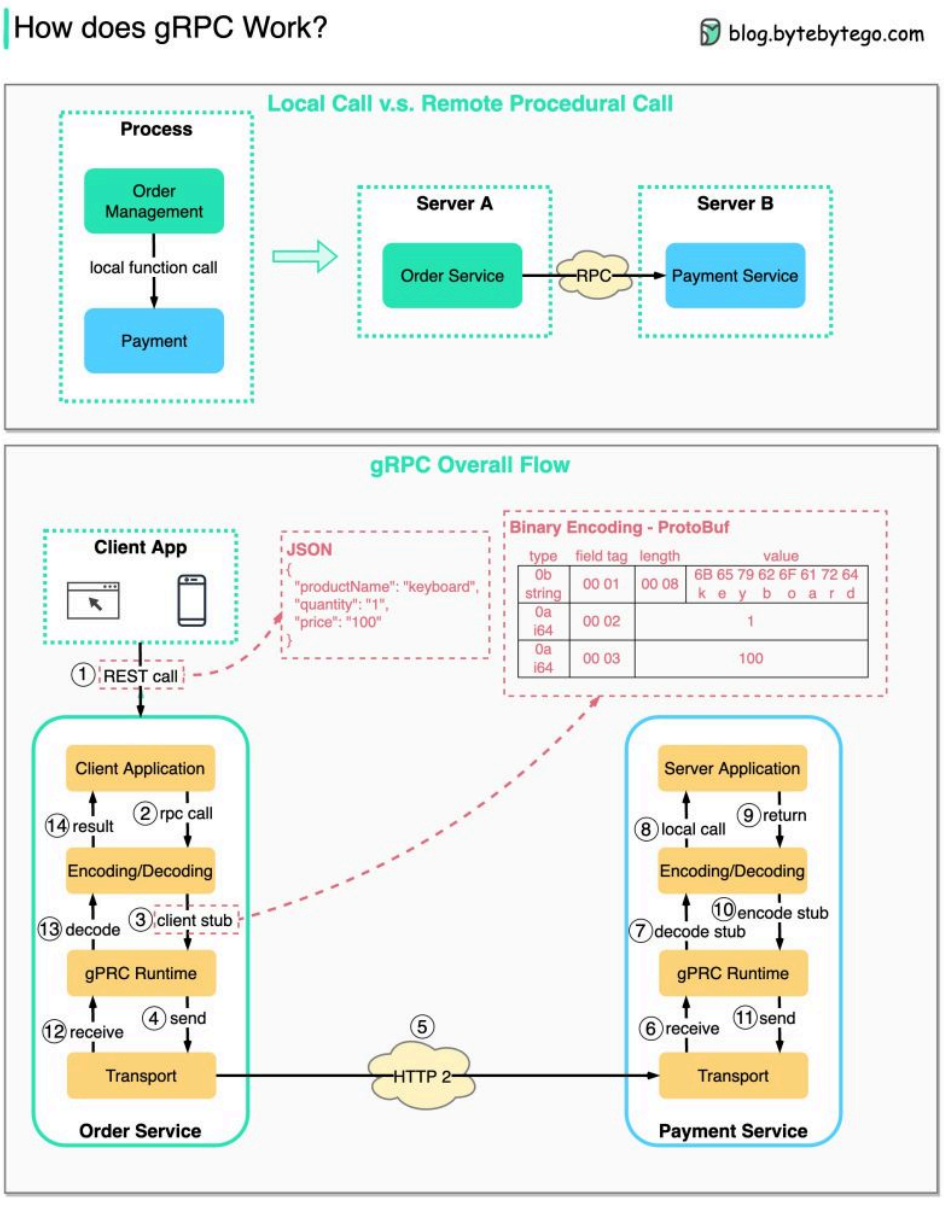
To make the best decision for our projects, it is essential to understand the various types of databases available in the market. We need to consider key characteristics of different database types, including popular options for each, and compare their use cases.



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.

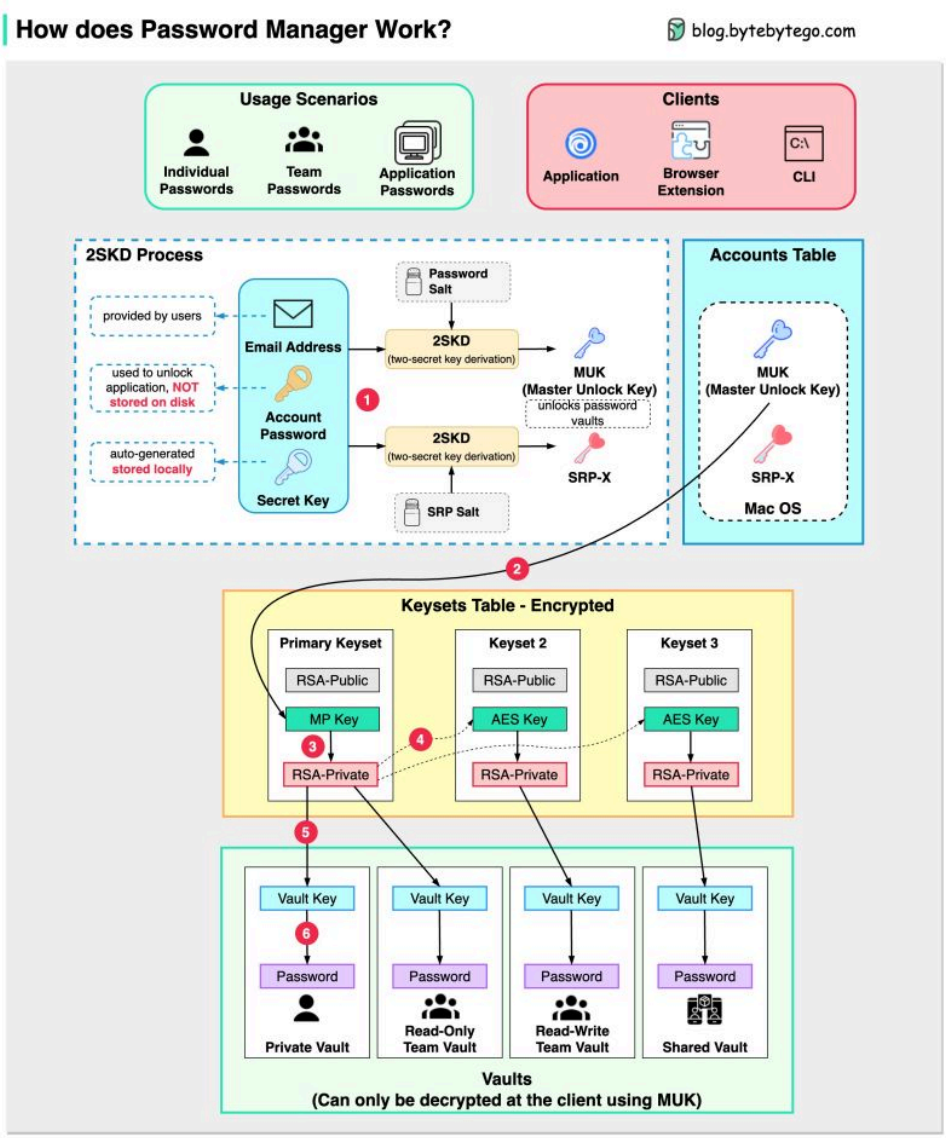
Steps 9 - 11: The result is returned from the server application, and gets encoded and sent to the transport layer.

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.

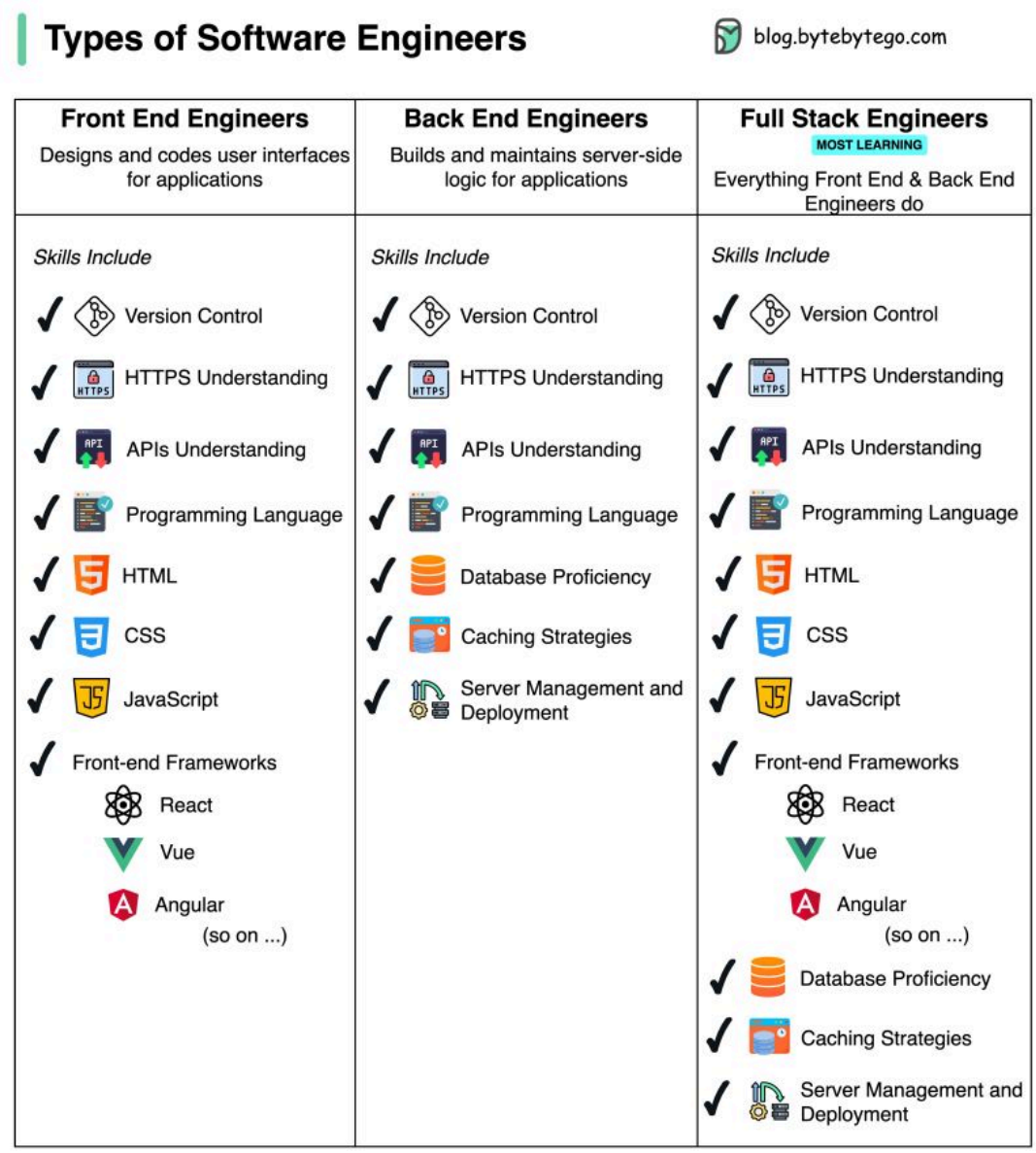
Steps 3-5: The MP key is then used to generate a private key, which can be used to generate AES keys in other keysets. The private key is also used to generate the vault key. Vault stores a collection of items for us on the server side. The items can be passwords notes etc.

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



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.

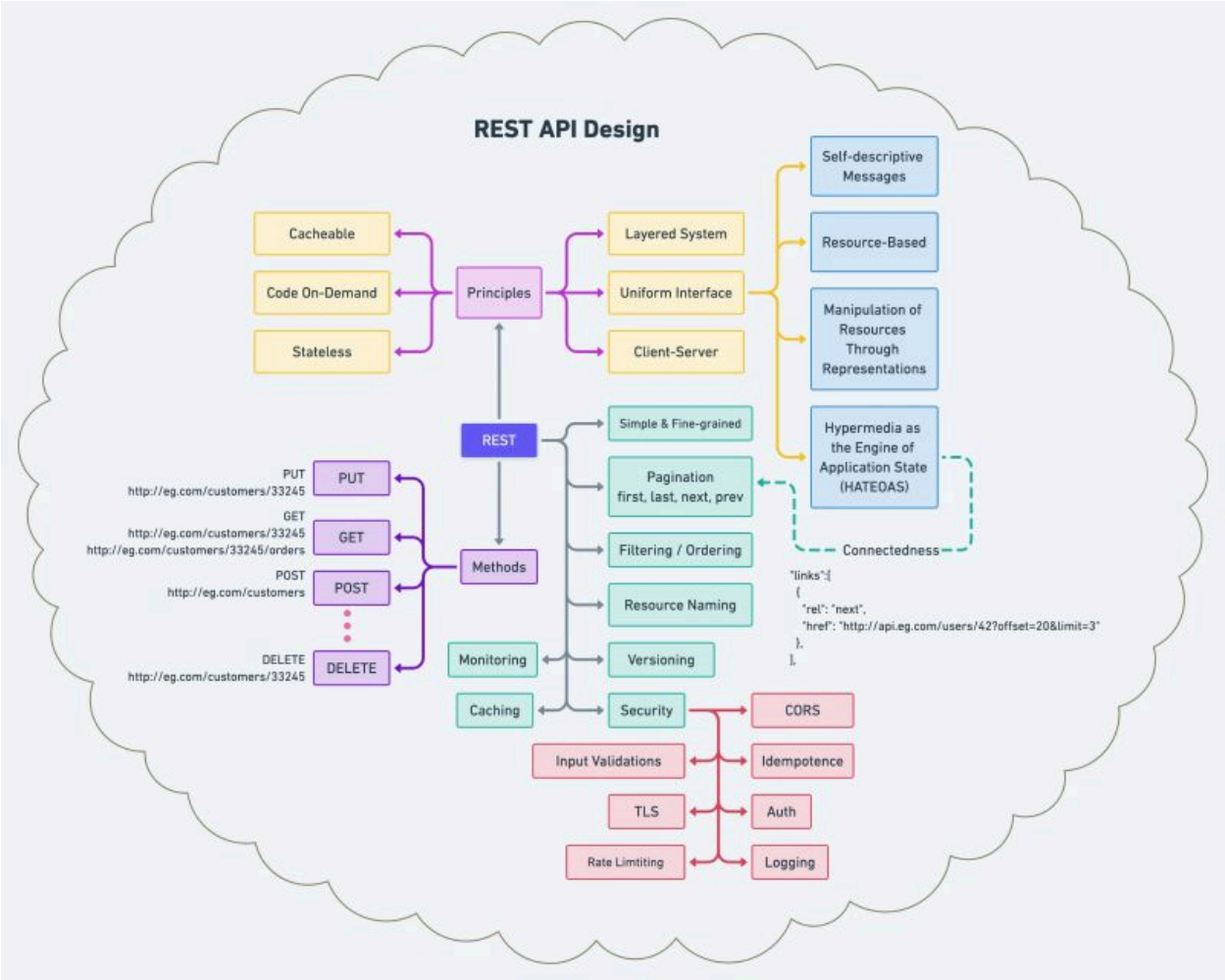
2. Back-End Engineer:
   Works on the server-side of web applications, managing data, business logic, and server infrastructure to ensure functionality, performance, and security.

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?

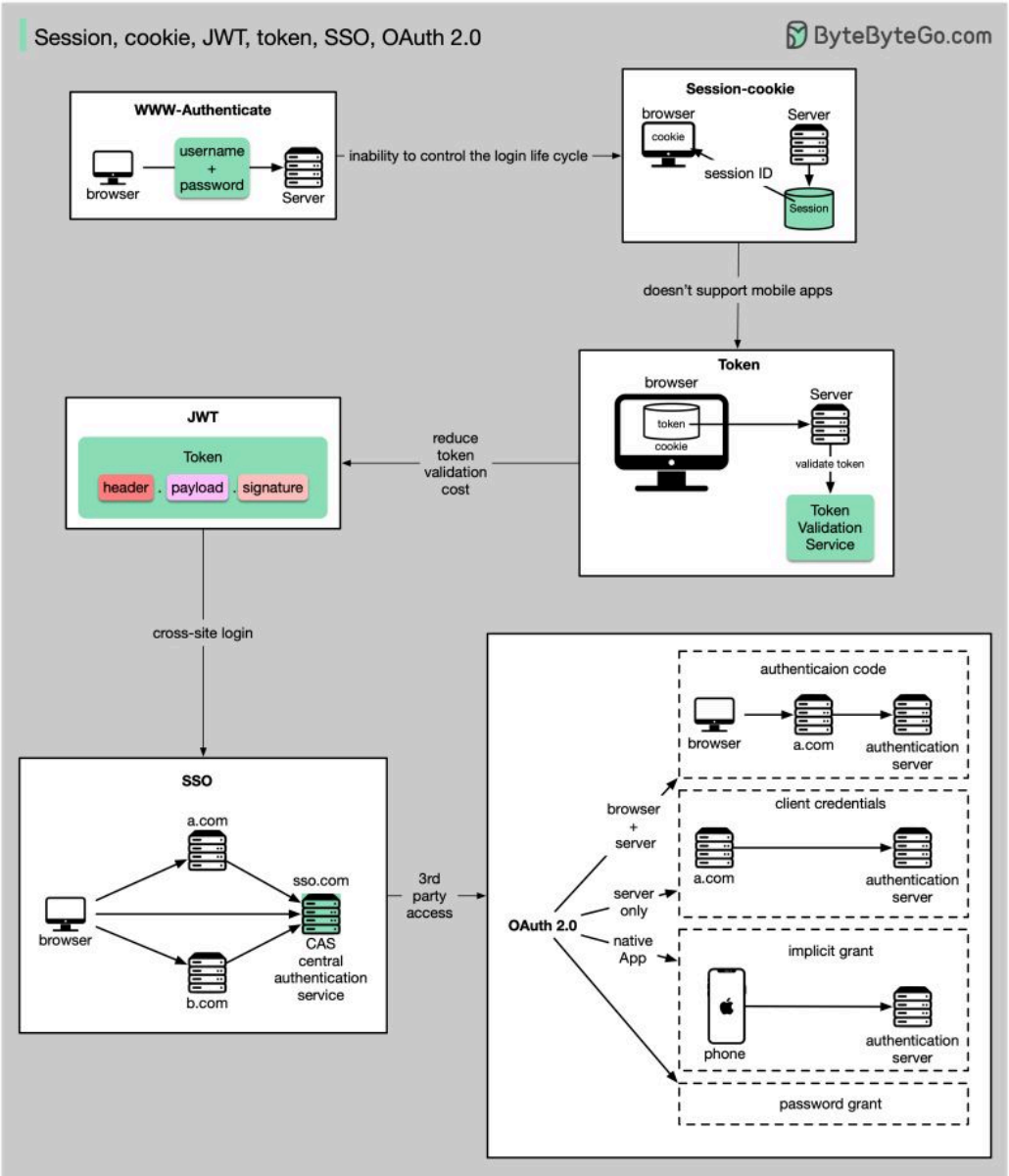
How does REST API work?

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.



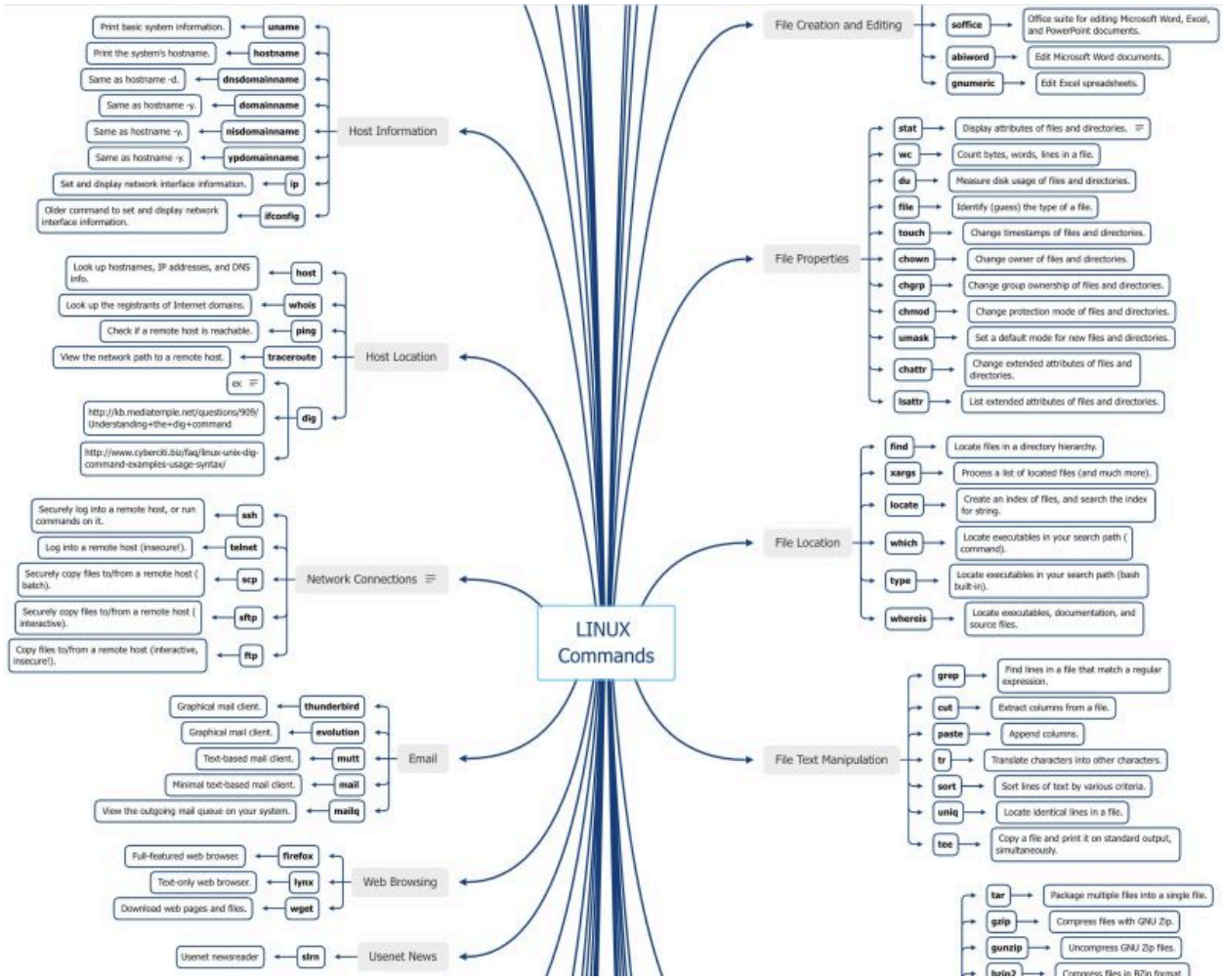
From simple to complex, here is my understanding of user identity management:

- 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 :)

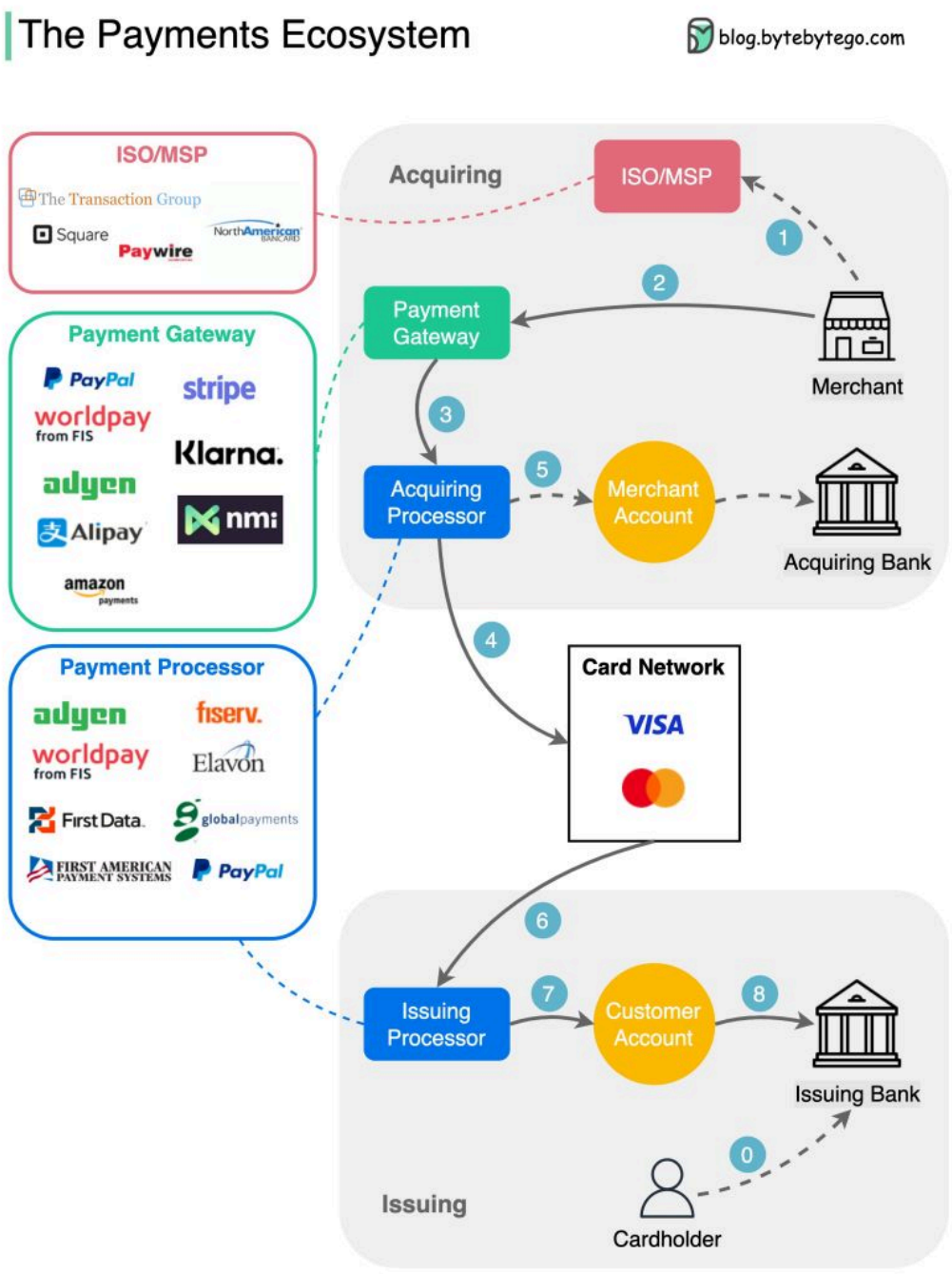


- Controlling processes: kill, killall, nice
- Scheduling jobs: sleep, watch, crontab
- Host location: host, whois, ping, traceroute
- Network connections: ssh, telnet, scp, ftp
- Screen output: echo, printf, seq, clear
- Viewing Processes: ps, uptime, top, free
- And many more

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

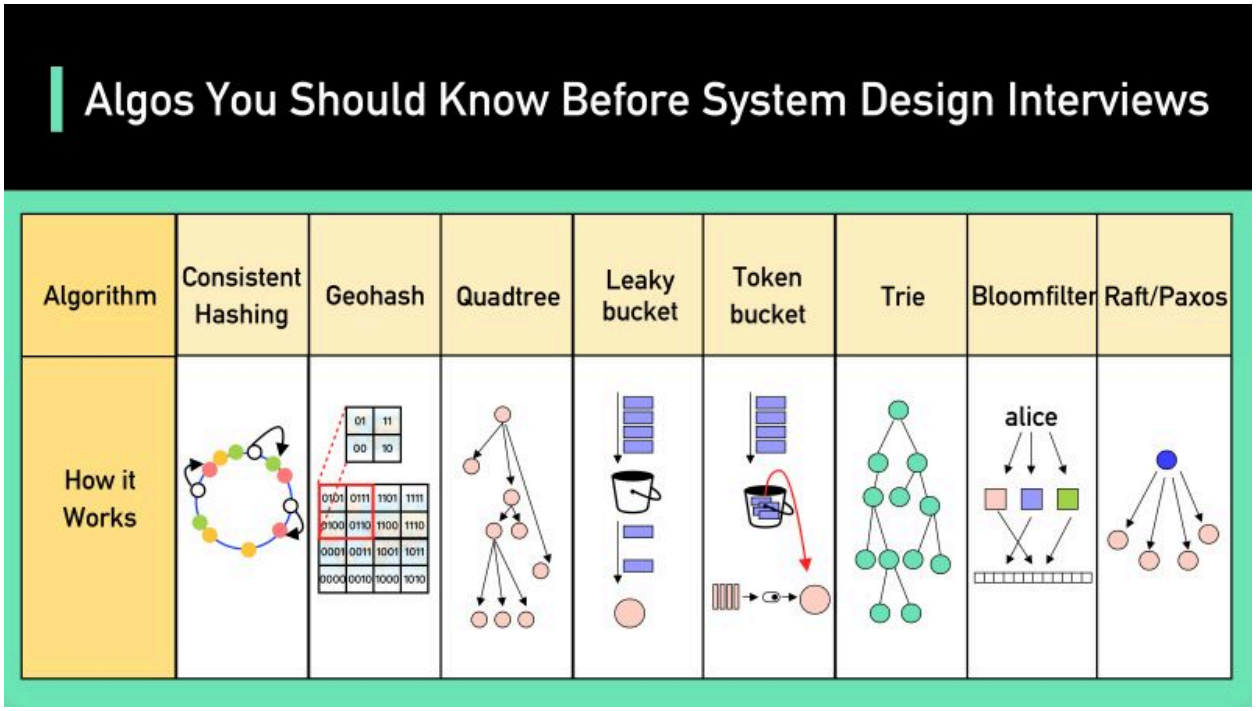
Provider) for in-store sales. ISO/MSP partners with payment processors to open merchant accounts.

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)

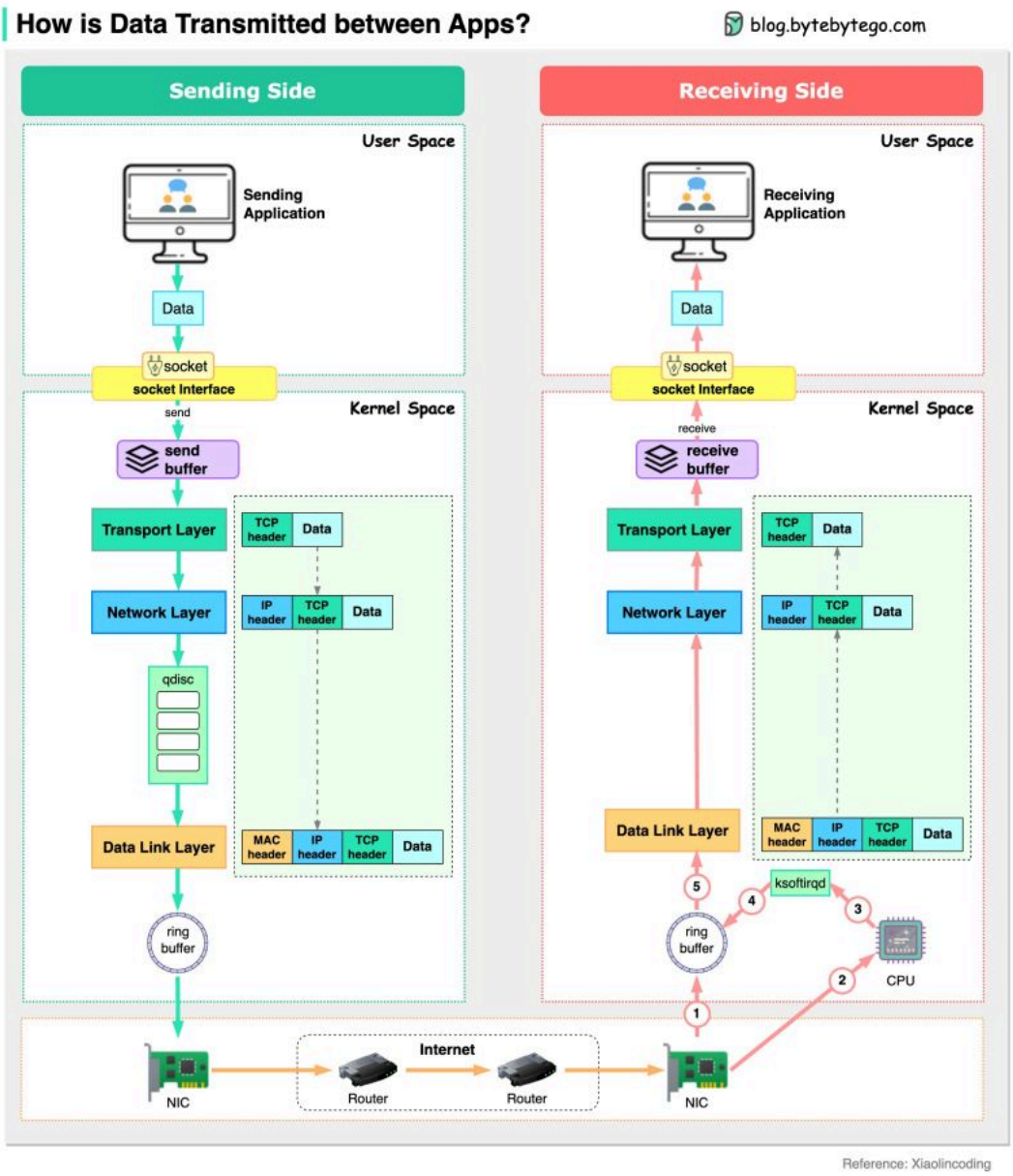


- Consistent hashing
- Spatial Indexing
- Rate Limiting
- Tries
- Bloom Filters
- Consensus Algorithms

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.

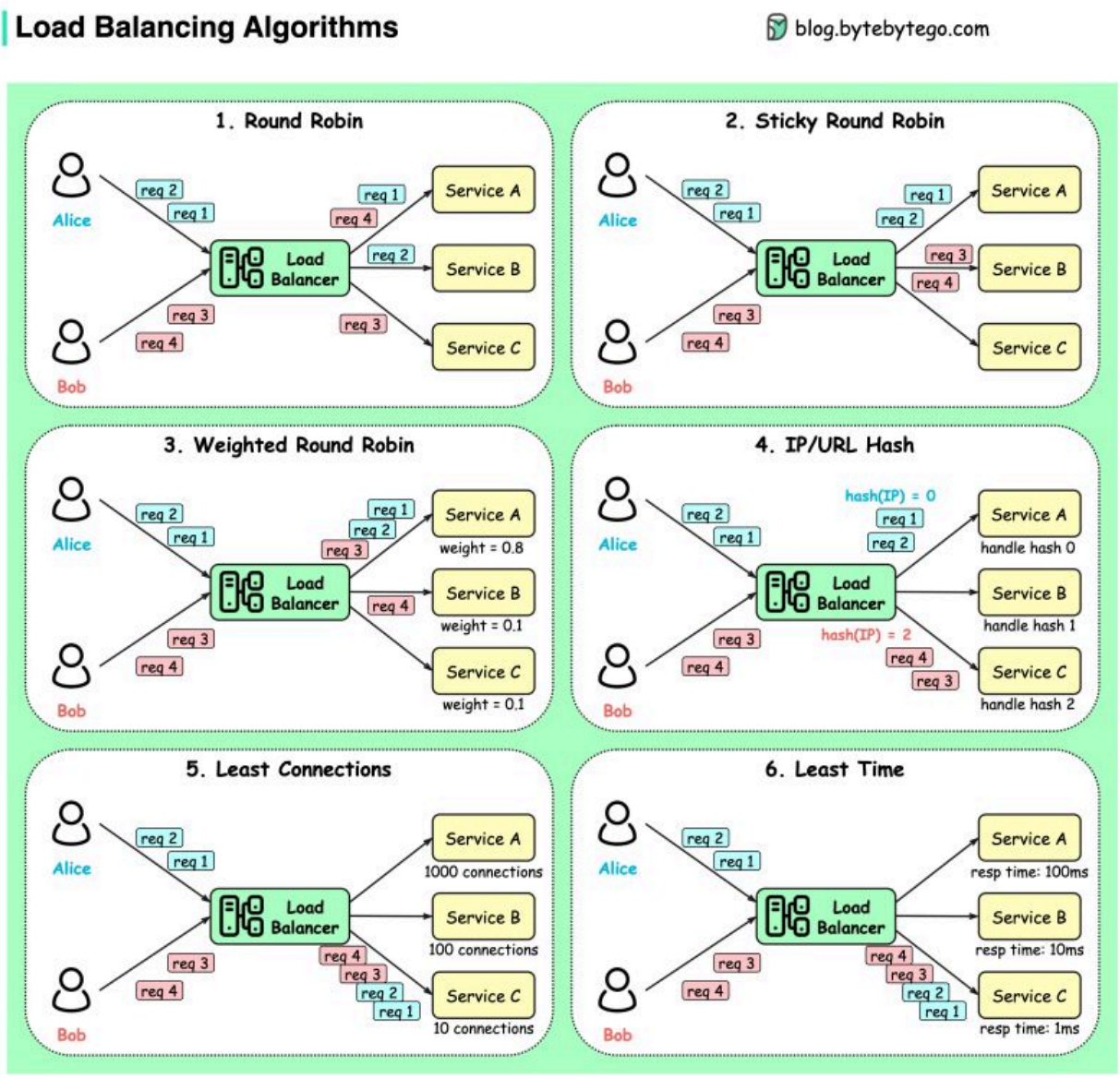
The data is sent to the internet via NIC. After many hops among routers and switches, the data arrives at the NIC of the receiving server.

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.



- 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.

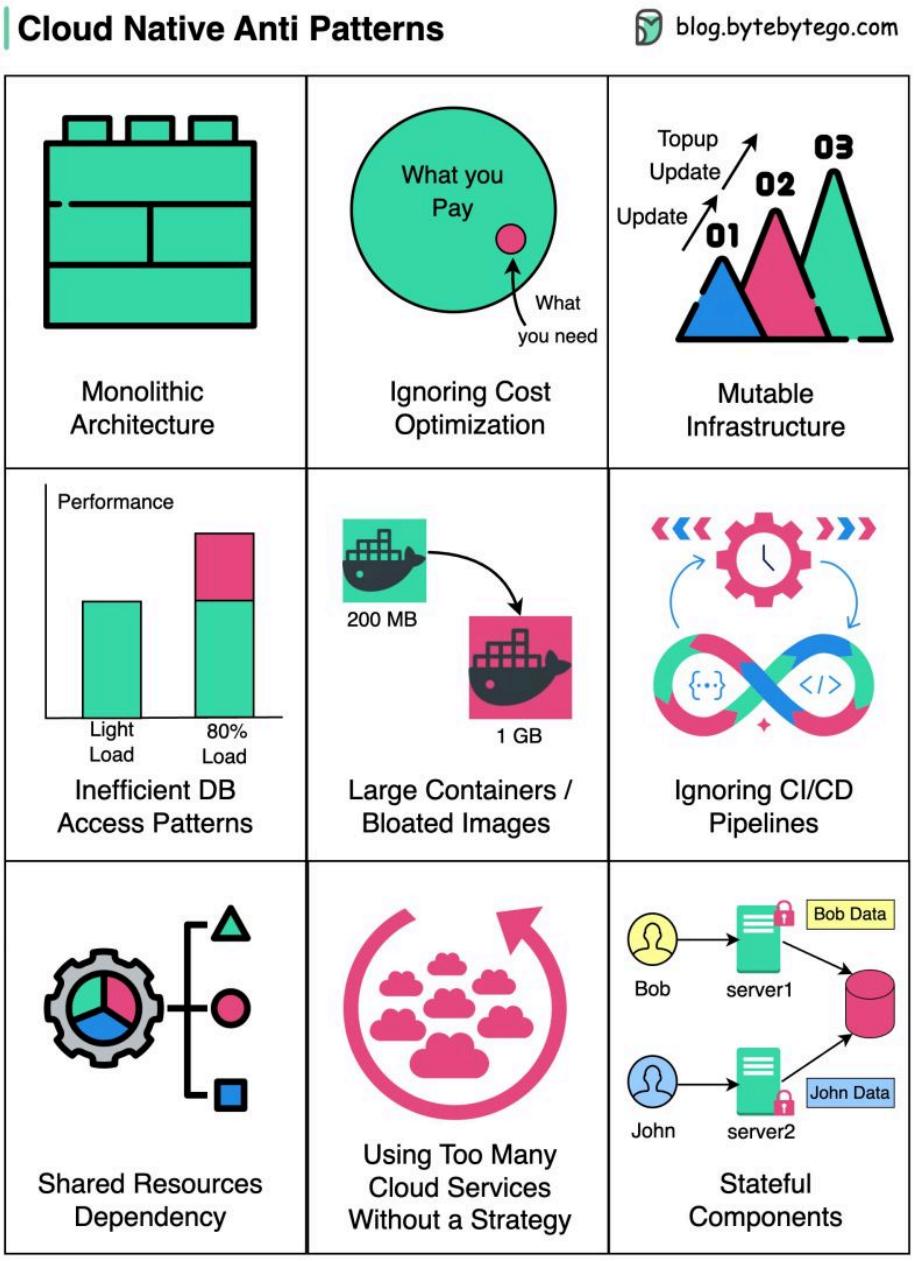
6. Least response time
   A new request is sent to the service instance with the fastest response time.

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.

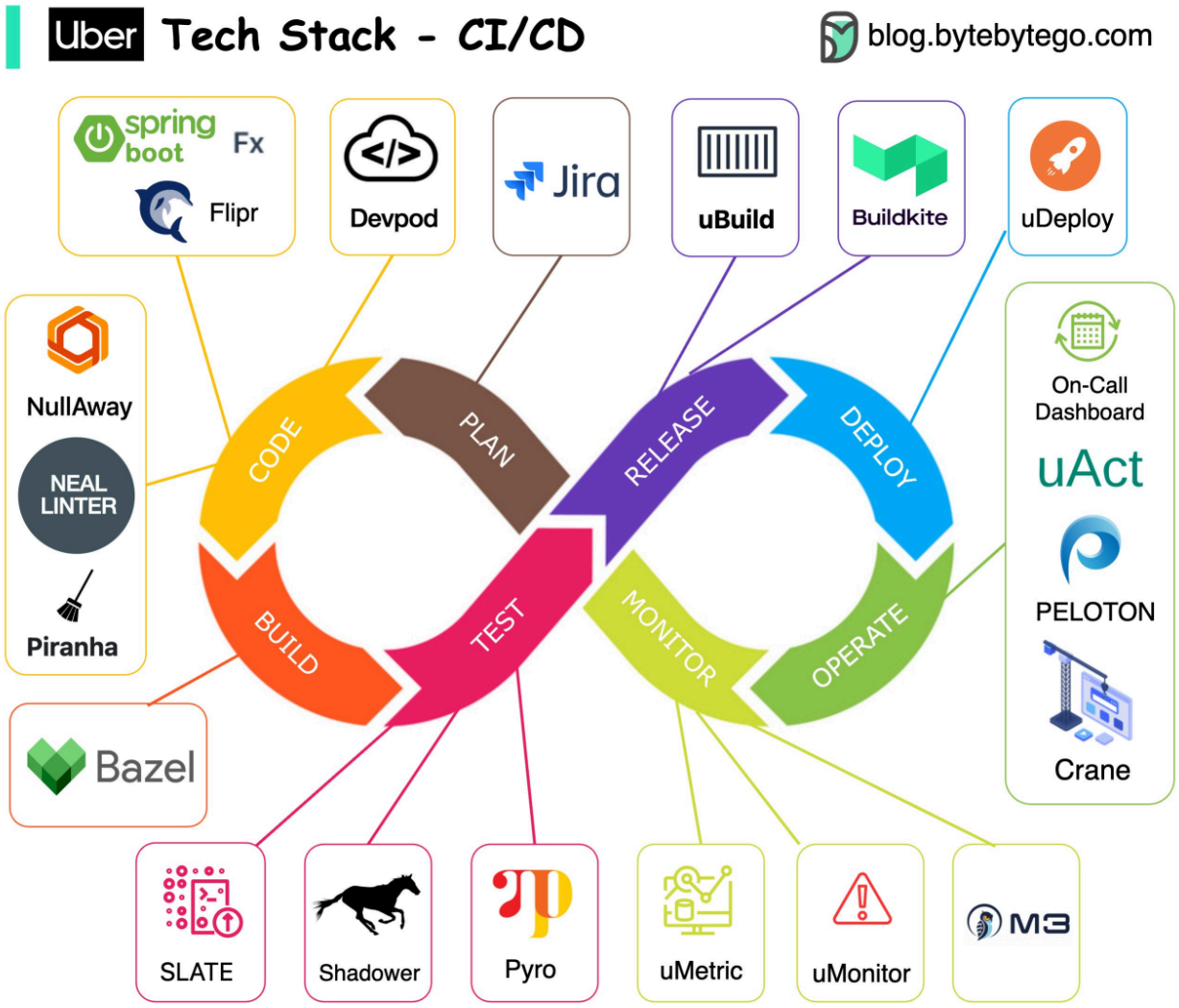
9. Stateful Components:
   Relying on persistent state in applications can introduce complexity, hinder scalability, and limit fault tolerance

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.

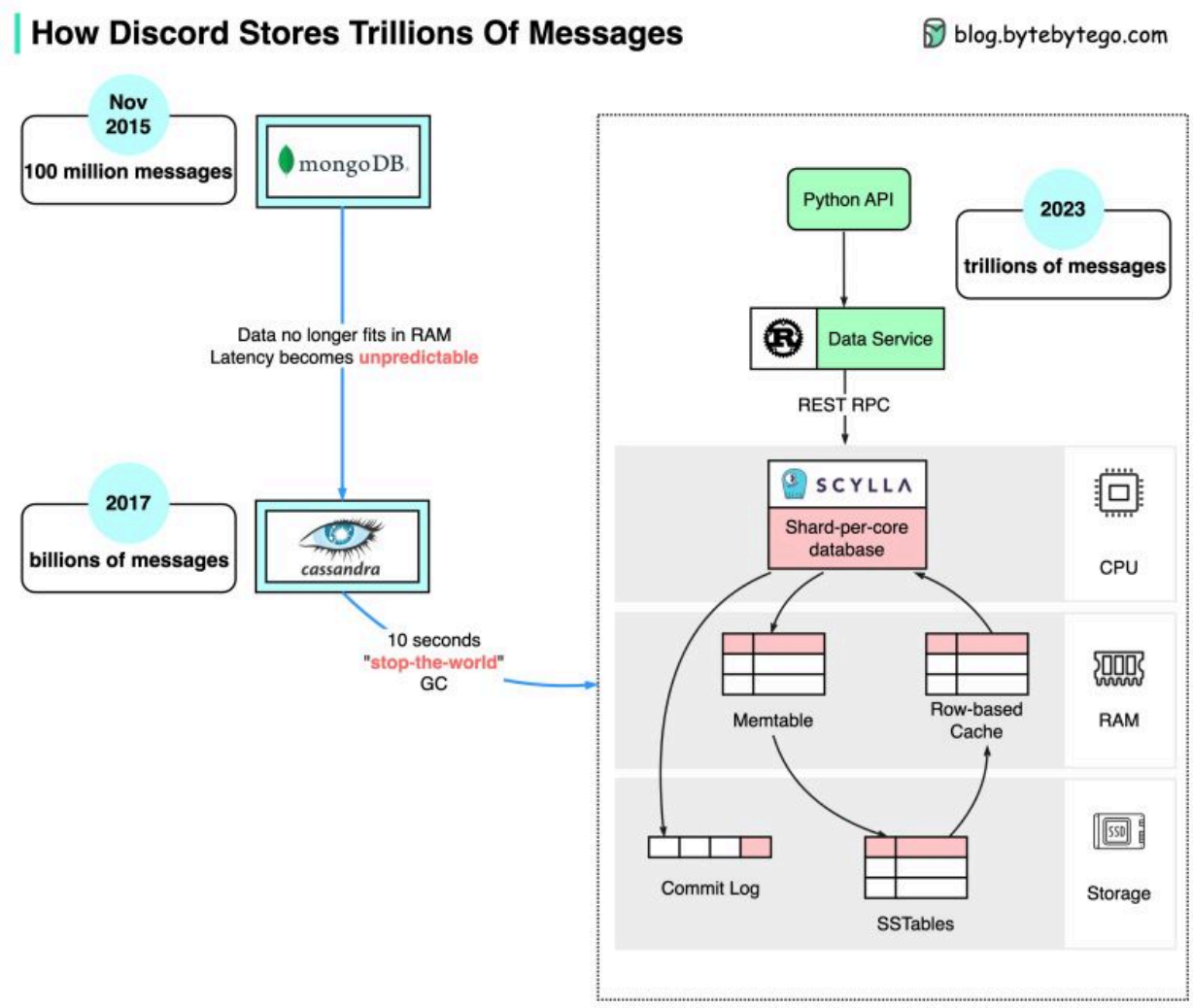
Special tooling: Uber relies on Peloton for capacity planning, scheduling, and operations. Crane builds a multi-cloud infrastructure to optimize costs. And with uAct and the OnCall dashboard, they've got event tracing and on-call duty management covered.

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.

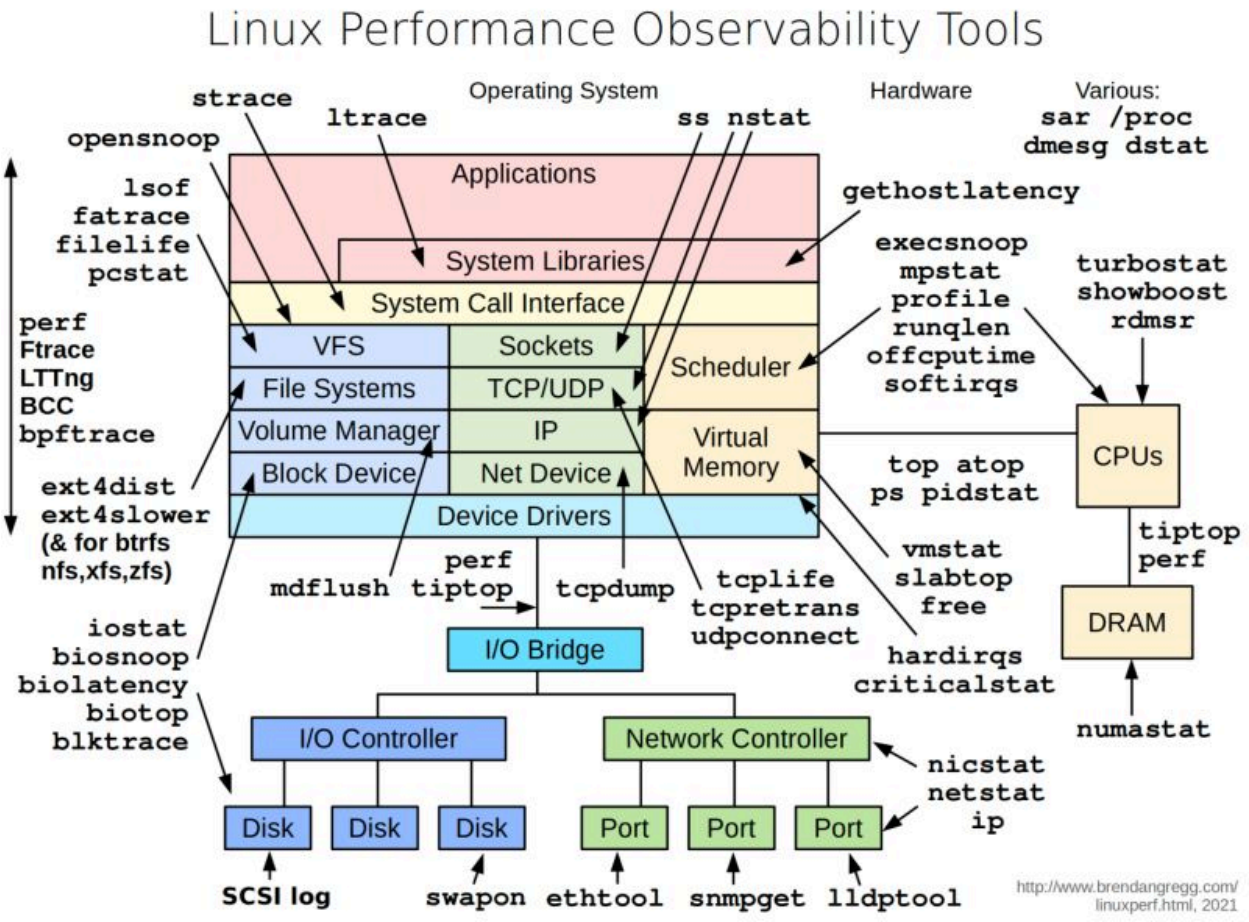
Over to you: What kind of NoSQL database have you used? How do you like it?

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.

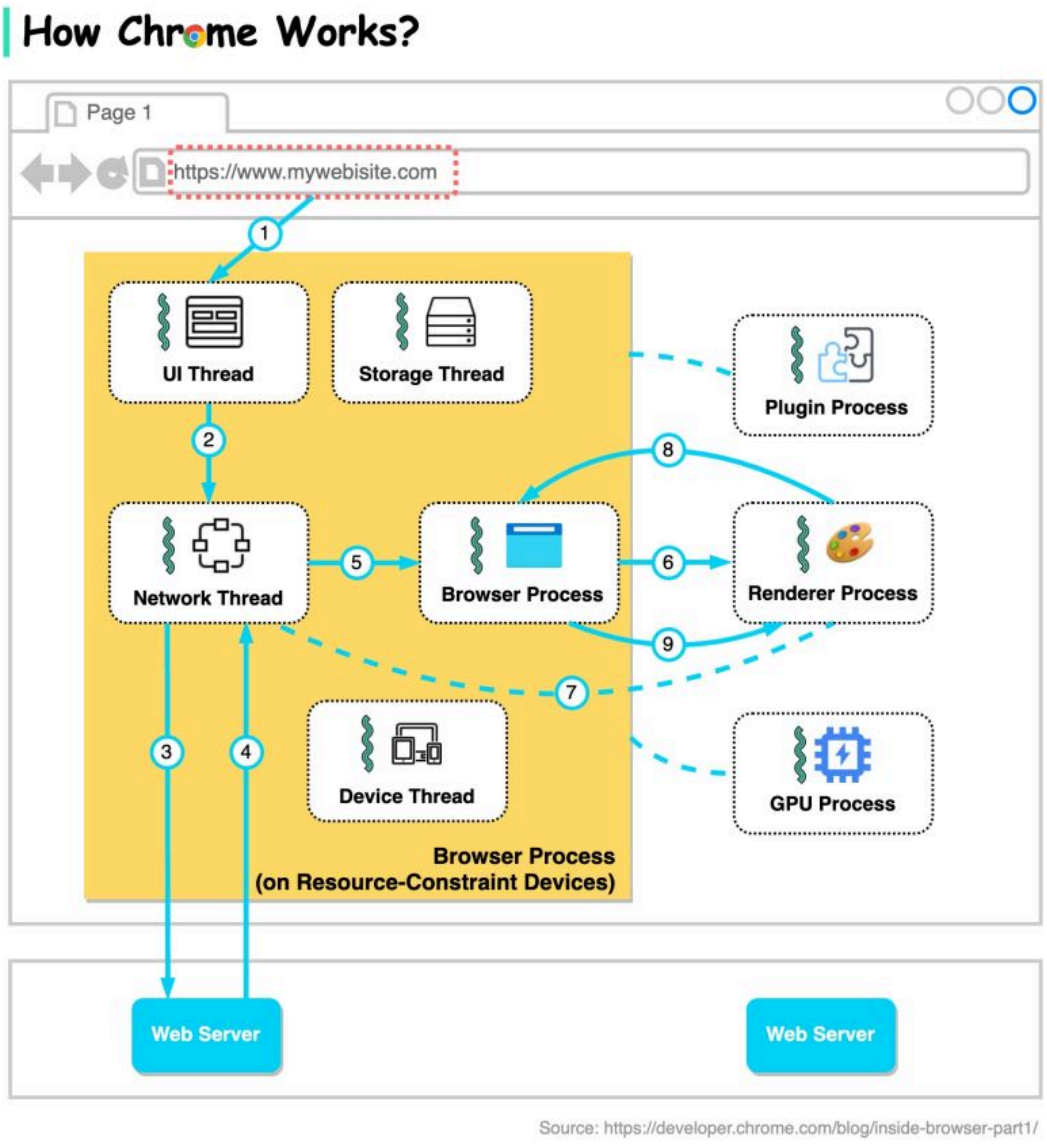


- 'vmstat' - reports information about processes, memory, paging, block IO, traps, and CPU activity.
- 'iostat' - reports CPU and input/output statistics of the system.
- 'netstat' - displays statistical data related to IP, TCP, UDP, and ICMP protocols.
- 'lsof' - lists open files of the current system.
- '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.



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.

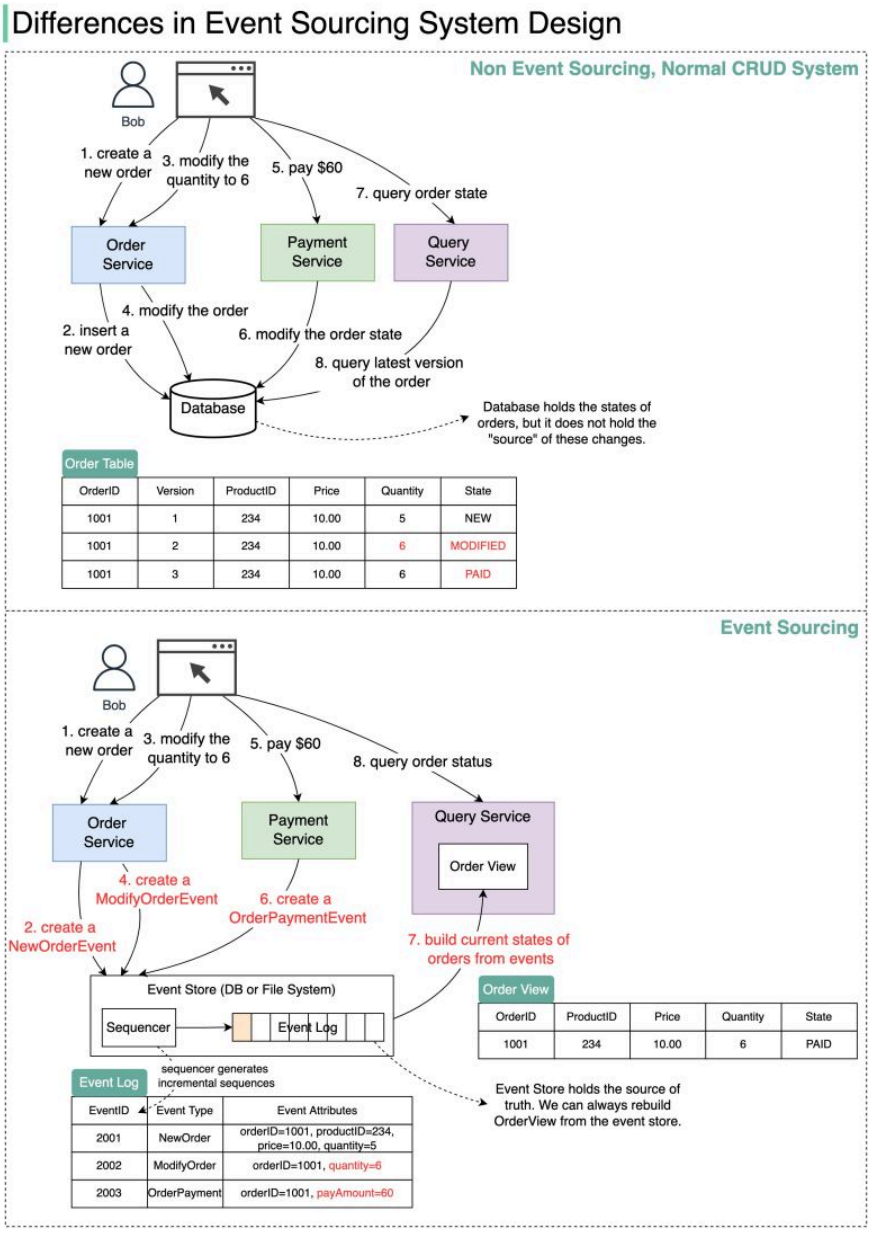
Over to you: Why does Chrome assign each tab a renderer process?

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.

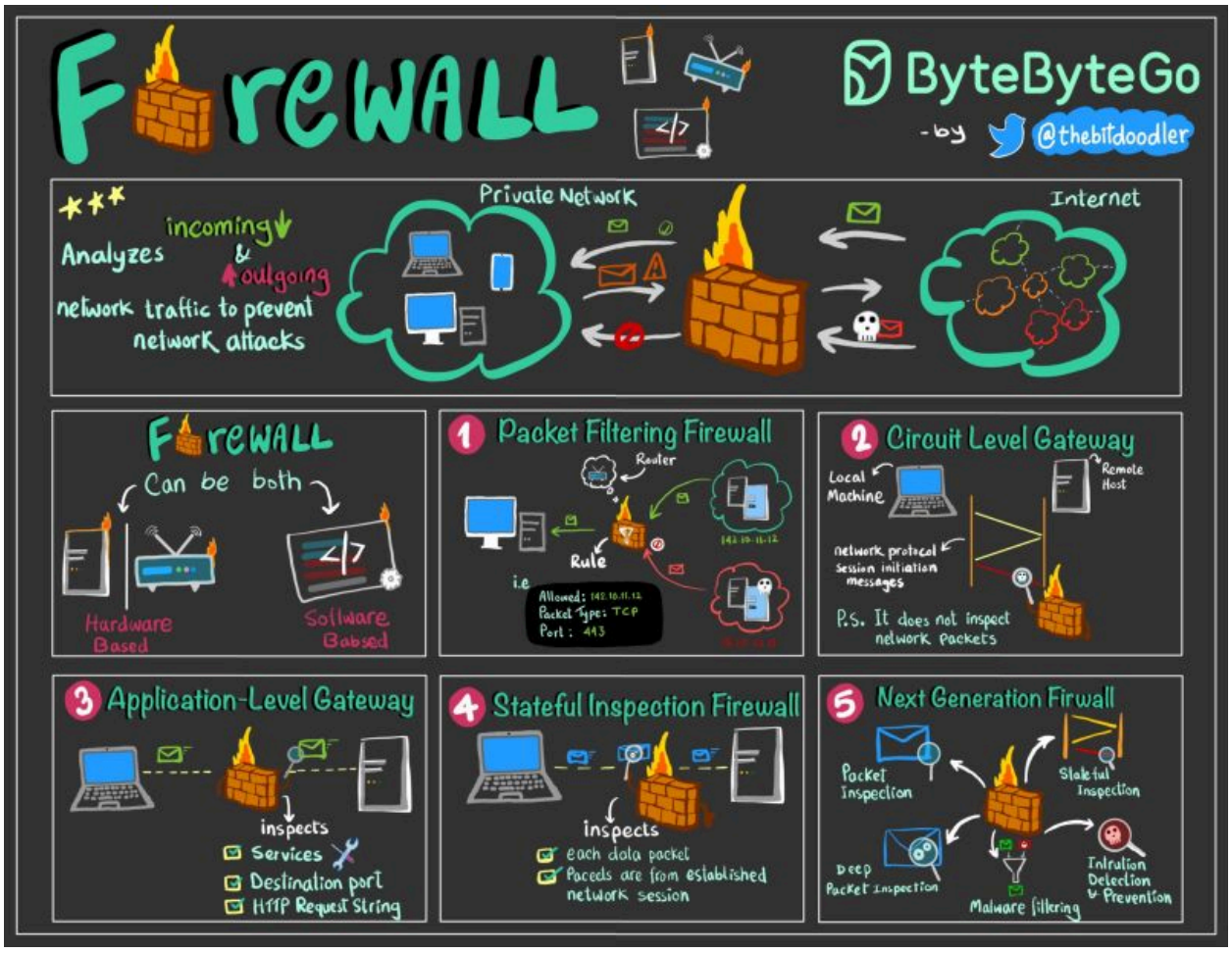
  Step 7: OrderView listens on the events published from the event store, and builds the latest state for the orders. Although OrderView receives 3 events, it applies the events one by one and keeps the latest state.

  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.

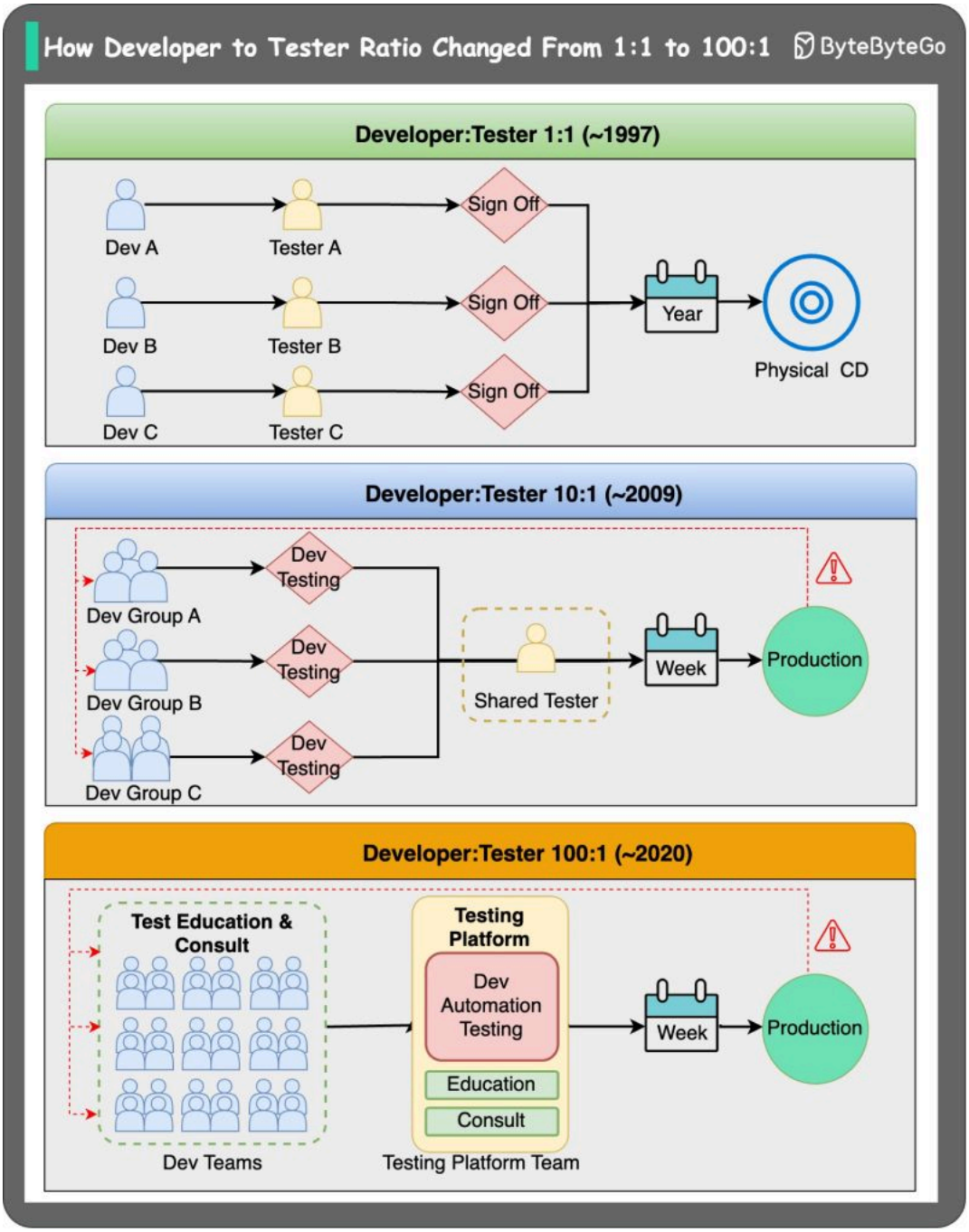
5. Next-Generation Firewalls (NGFWs): Advanced firewalls that integrate traditional methods with functionalities like intrusion prevention systems, deep packet analysis, and application awareness.

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.

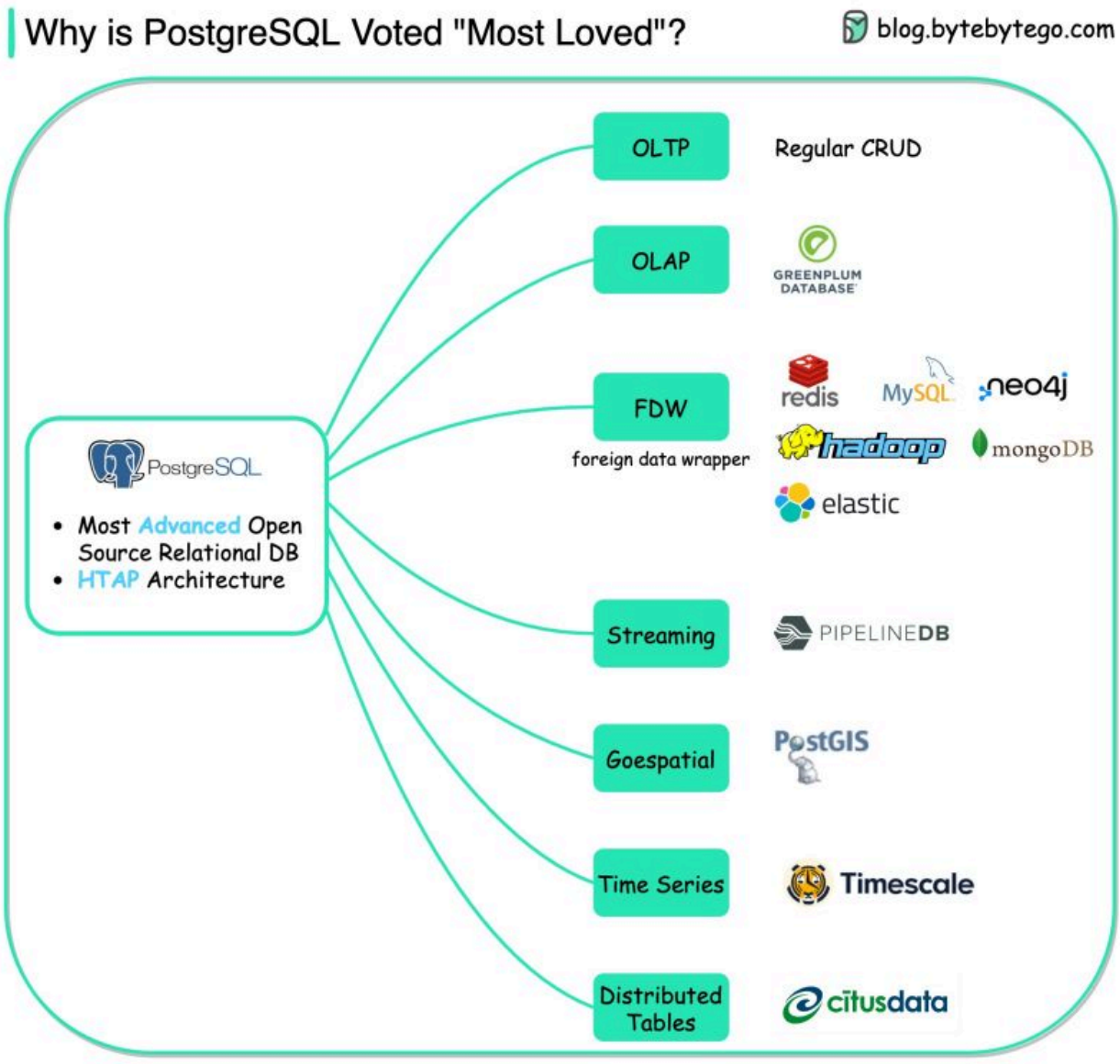
Secondly, testing knowledge is disseminated through education and consulting.

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++.

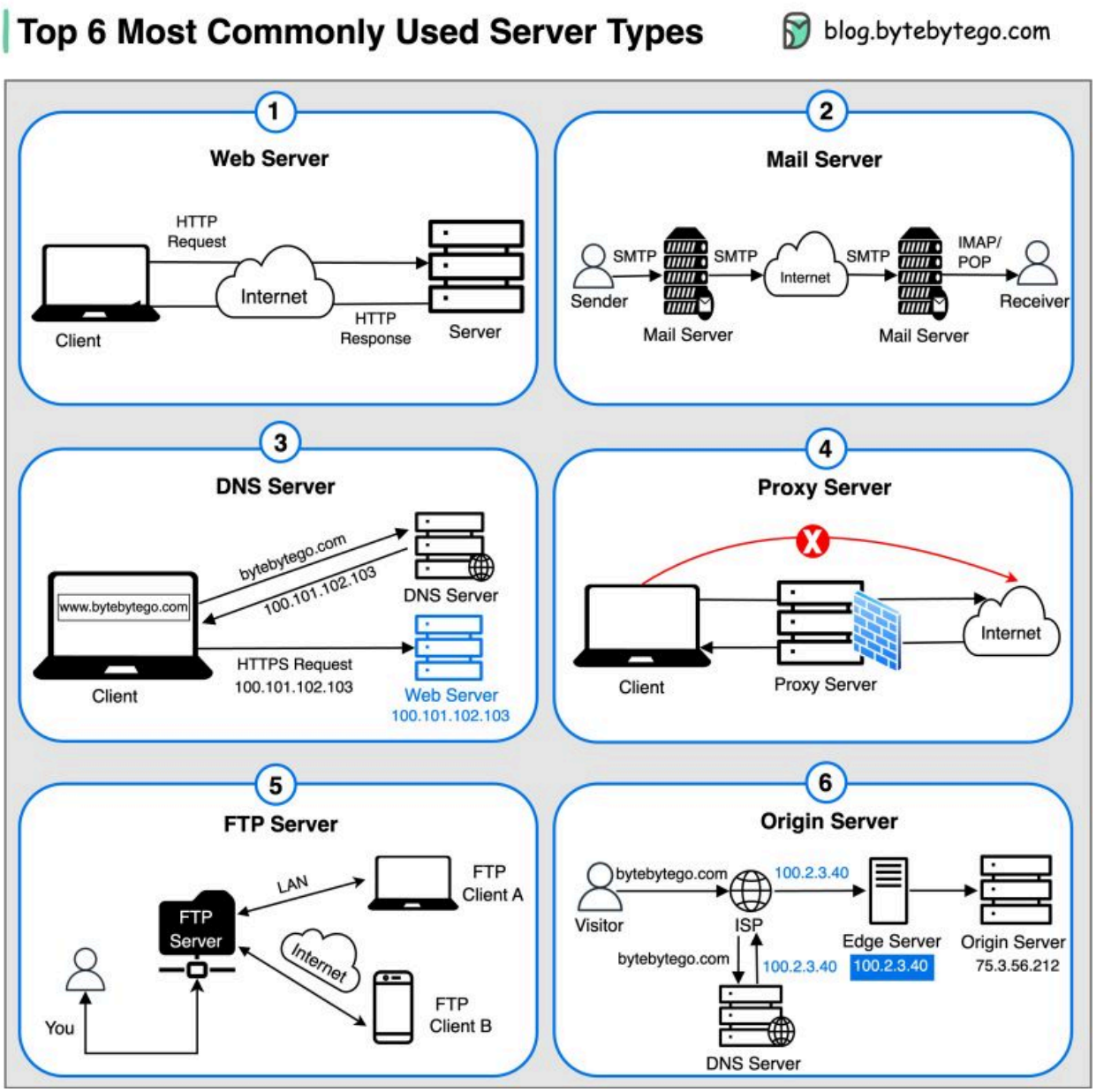
Why is OOP Important? OOP allows you to create blueprints (called classes) for digital objects, and these objects know how to communicate with one another to make amazing things happen

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



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.

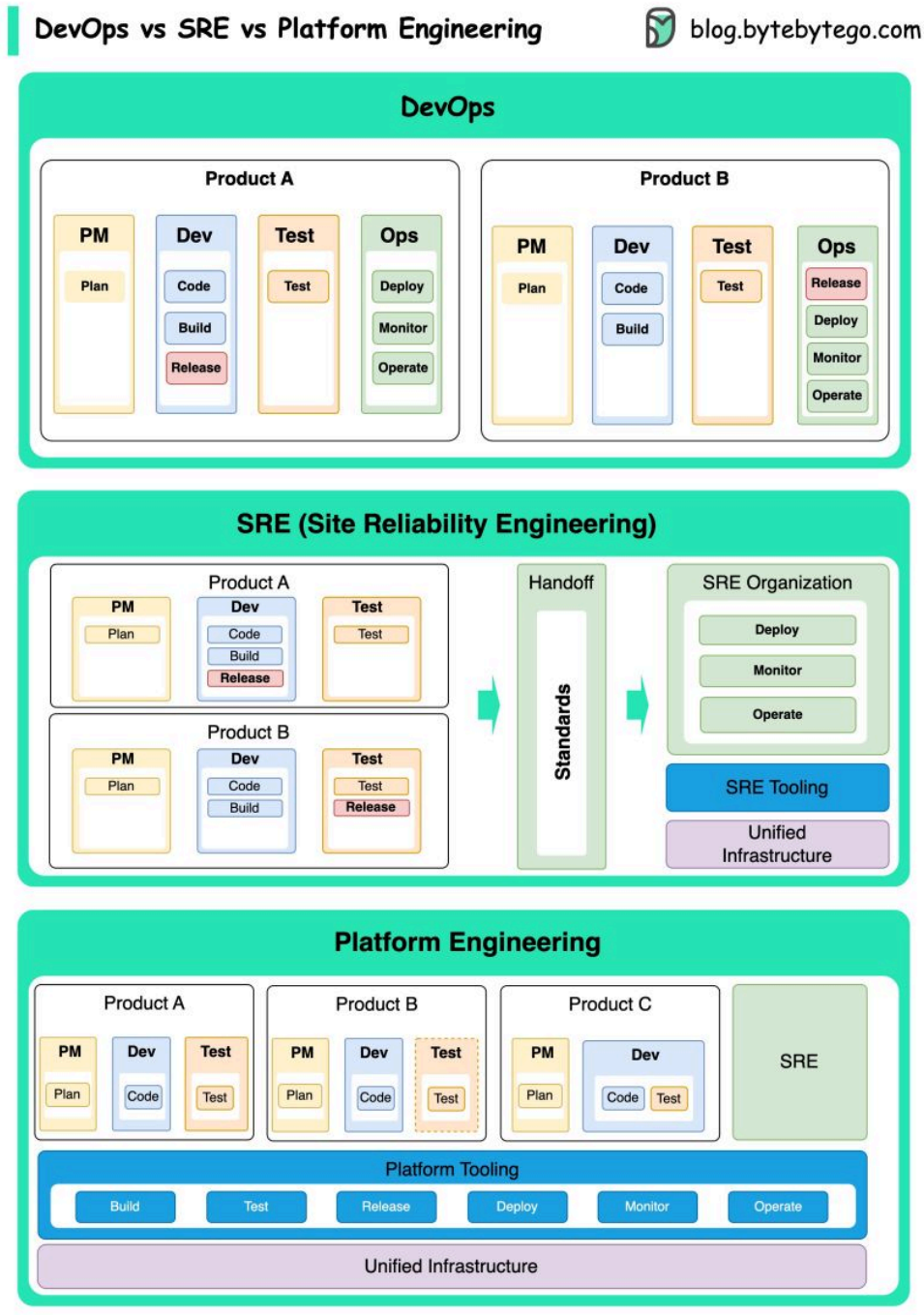
4. Proxy Server: An intermediary server that acts as a gateway between clients and other servers, providing additional security, performance optimization, and anonymity.

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?

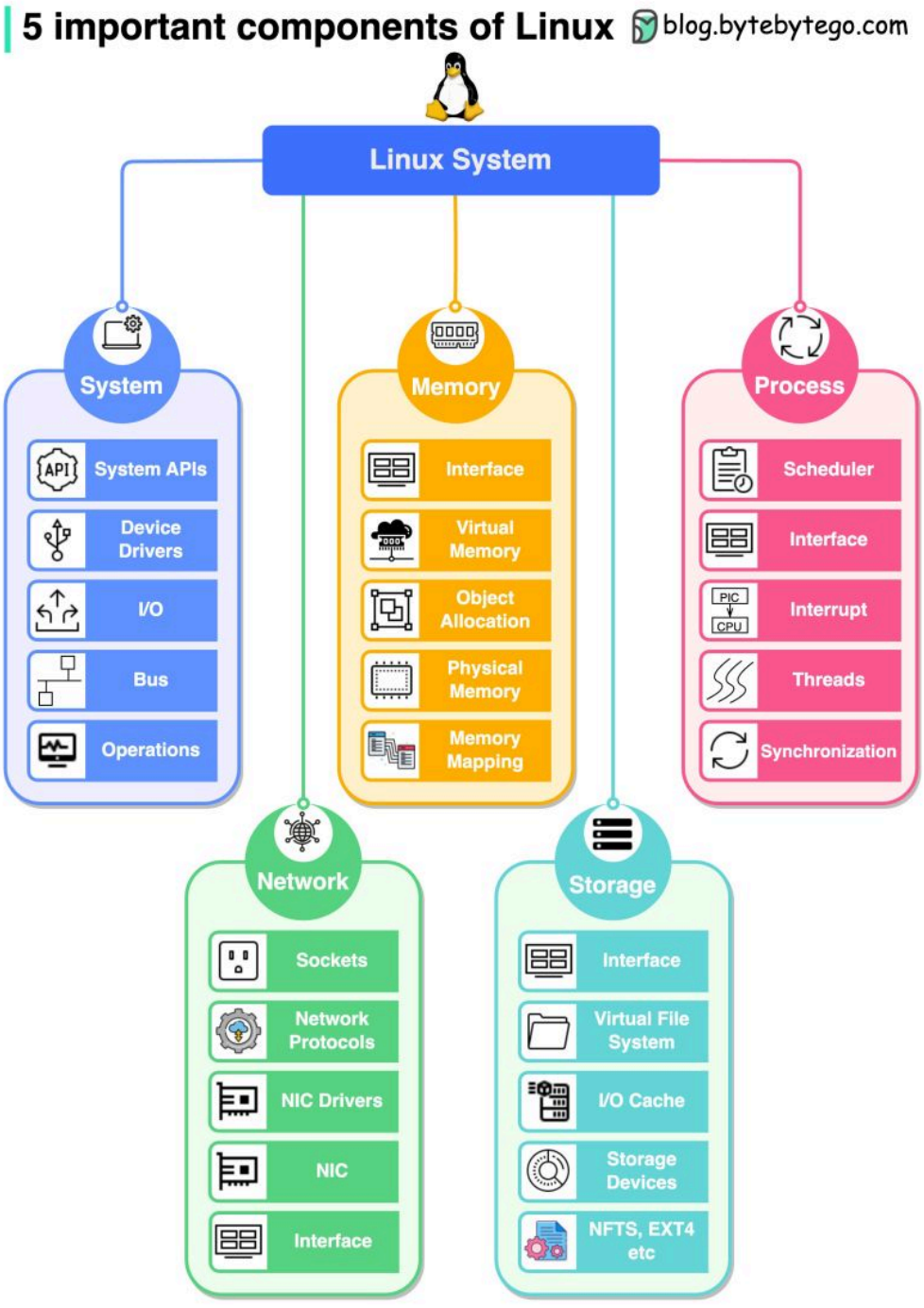
DevOps vs. SRE vs. Platform Engineering. Do you know the differences?



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



- 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.

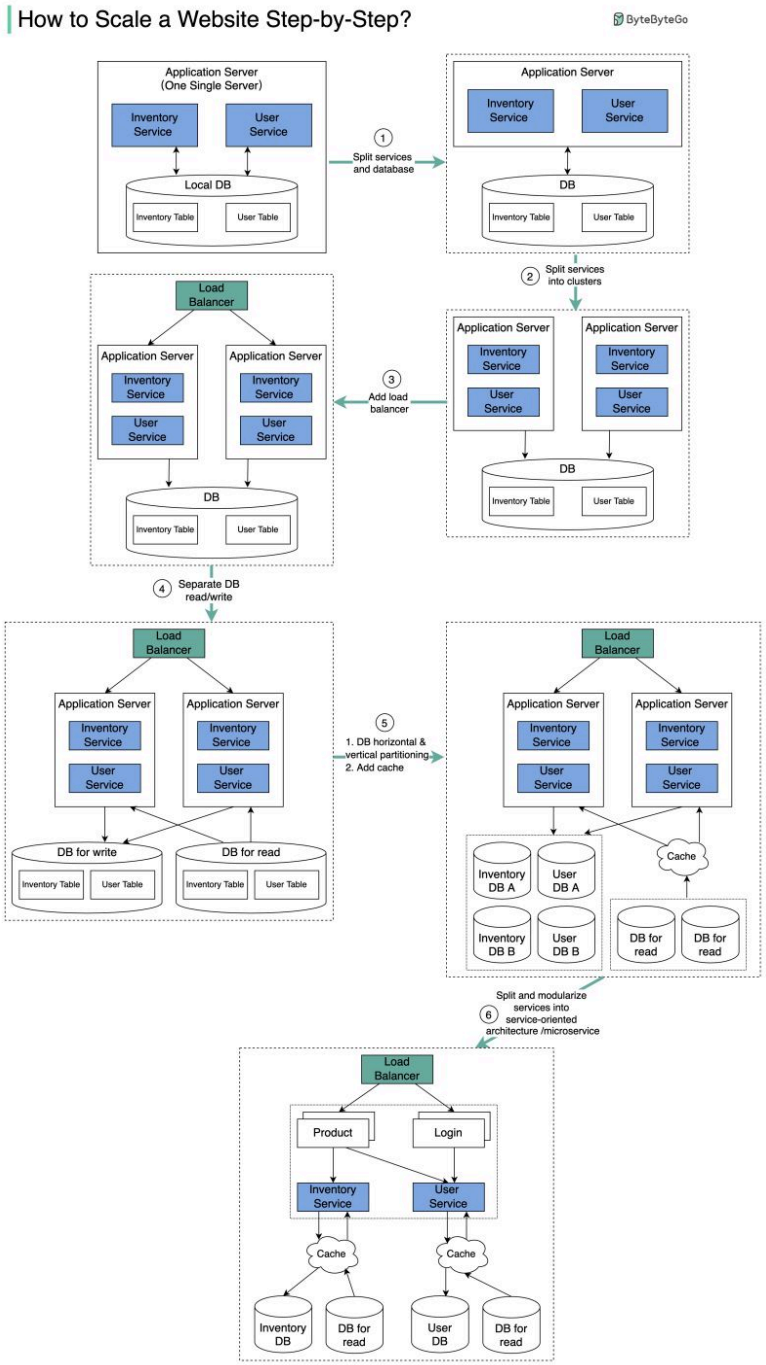
- Network
  In the network component, we need to learn important modules like network protocols, sockets, NIC drivers, etc.

- 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.

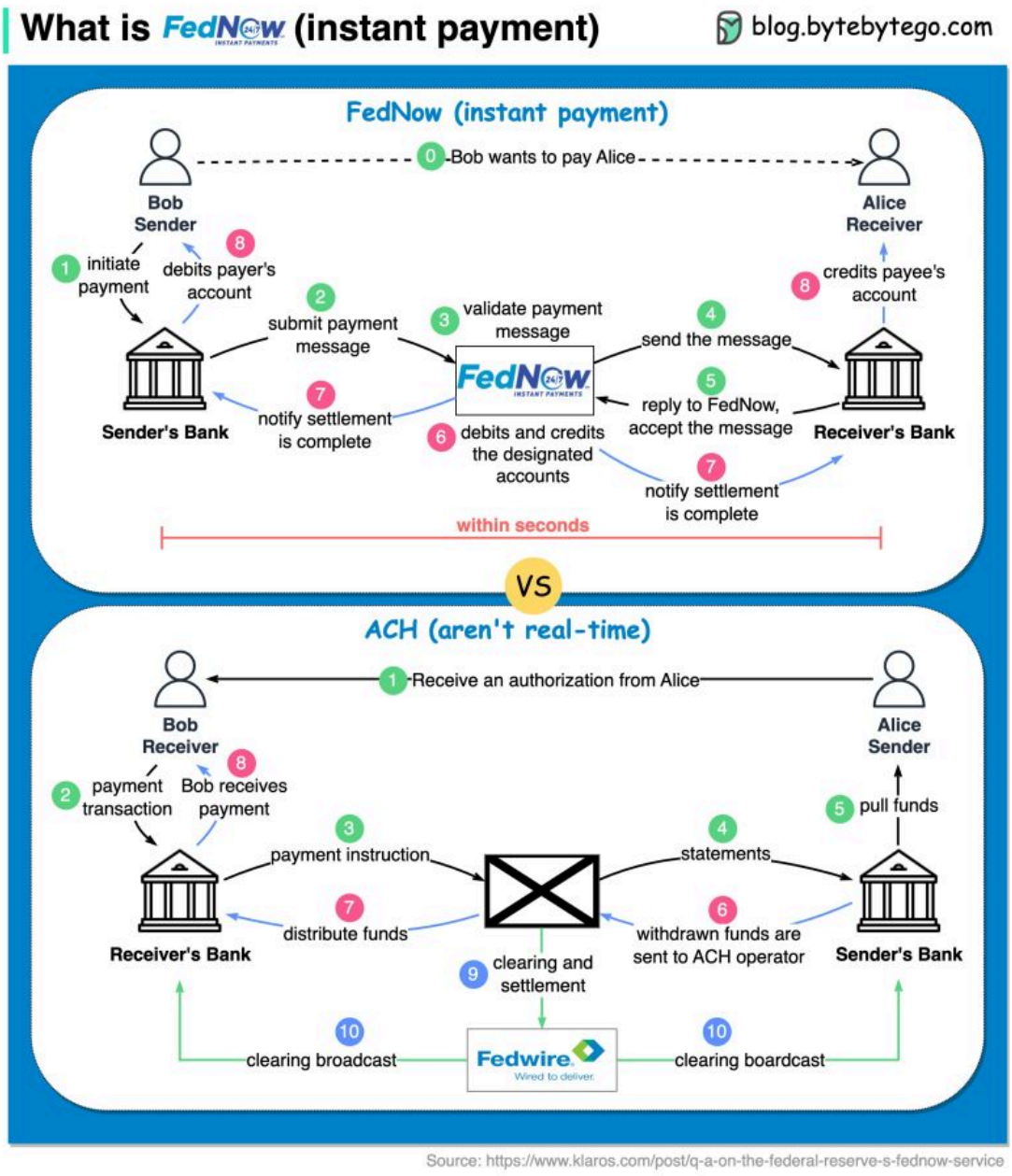
Federal Reserve launched FedNow instant payment service on 20 Jul. It allows retail clients to send and receive money within seconds and it is available 24x7.

- 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.



- 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.

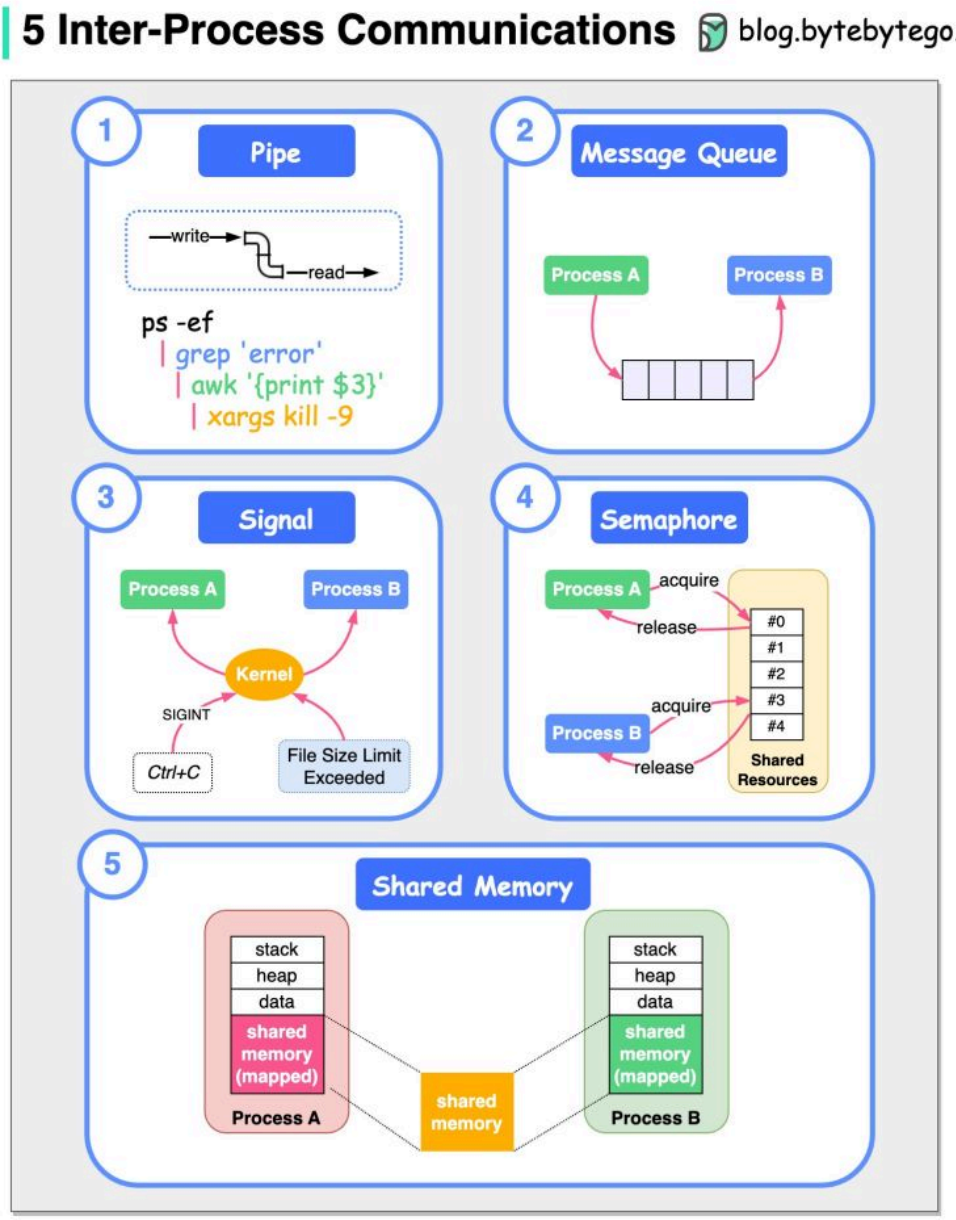
- ACH
  Step 1 - Bob receives authorization from Alice that he can deduct from Alice's account.
  Step 2 - The payment transaction is sent to the receiver's bank.
  Step 3 - The bank collects files in batches and sends them to the ACH operator.
  Step 4 - The ACH operator sends the files to the sender's 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.

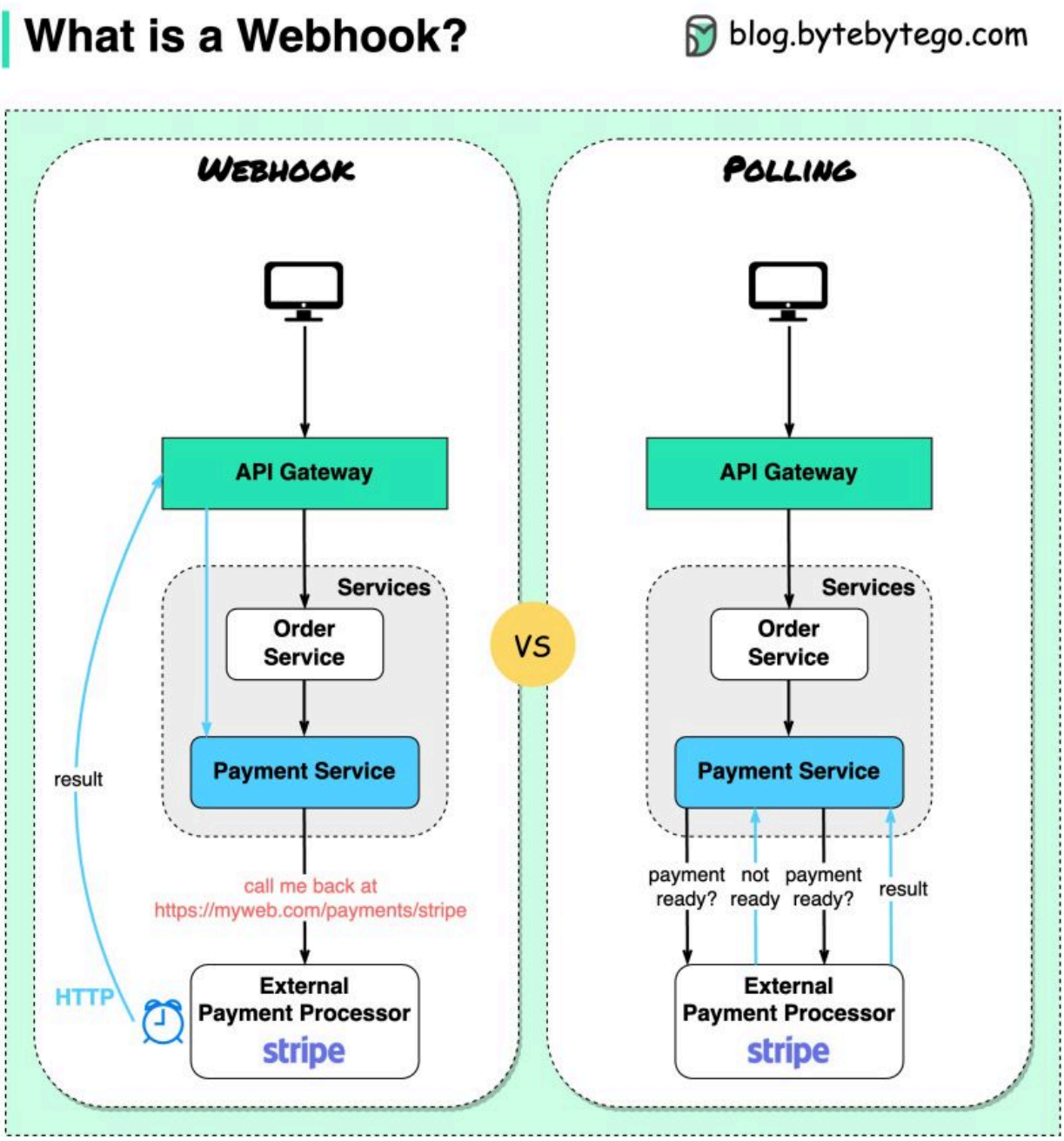
4. Semaphore
   A semaphore is a location in memory whose value can be tested and set by more than one process. Depending on the result of the test and set operation one process may have to sleep until the semaphore's value is changed by another process.

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.



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.

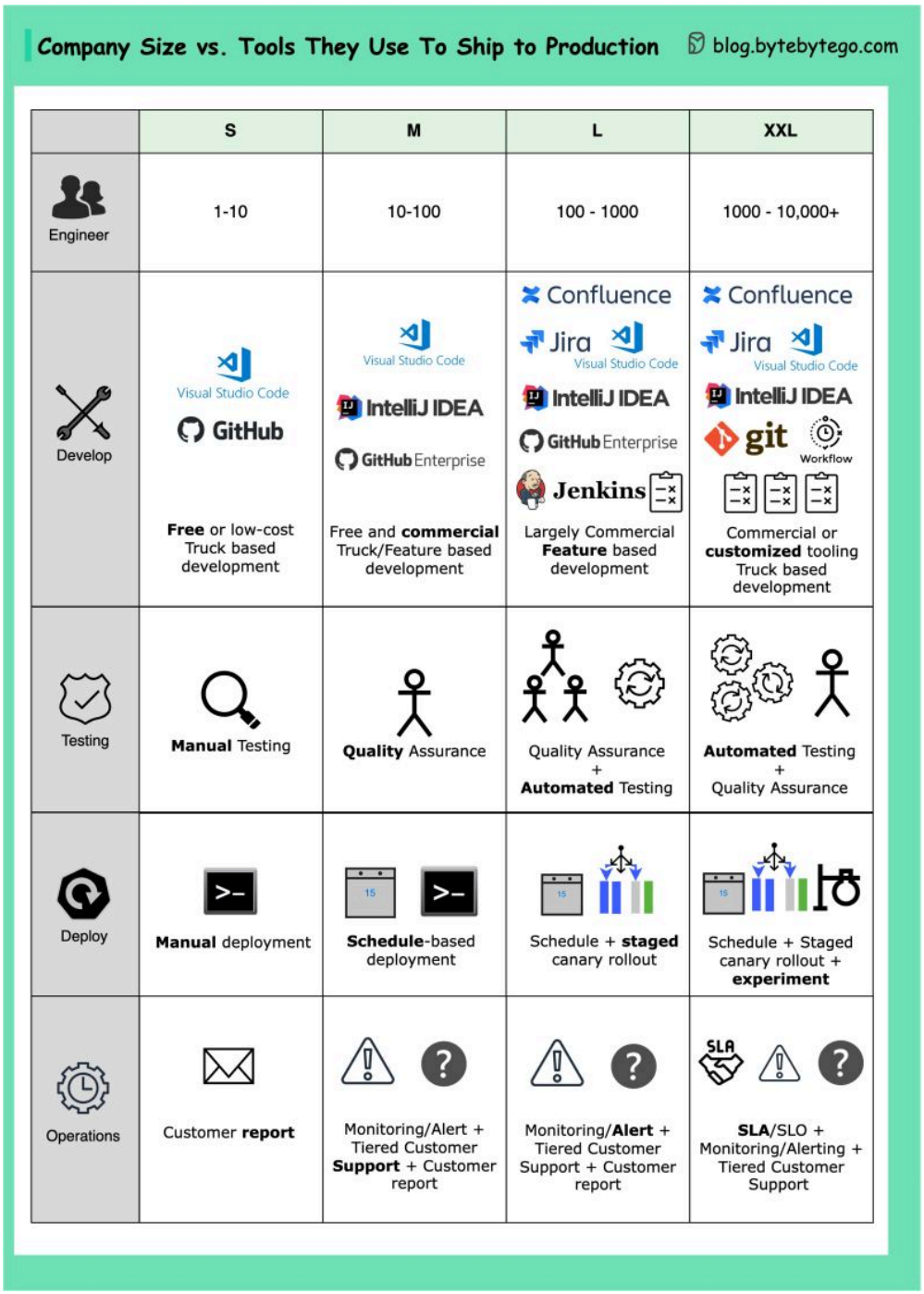
What if the PSP never calls back? We can set up a housekeeping job to check payment status every hour.

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.



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.

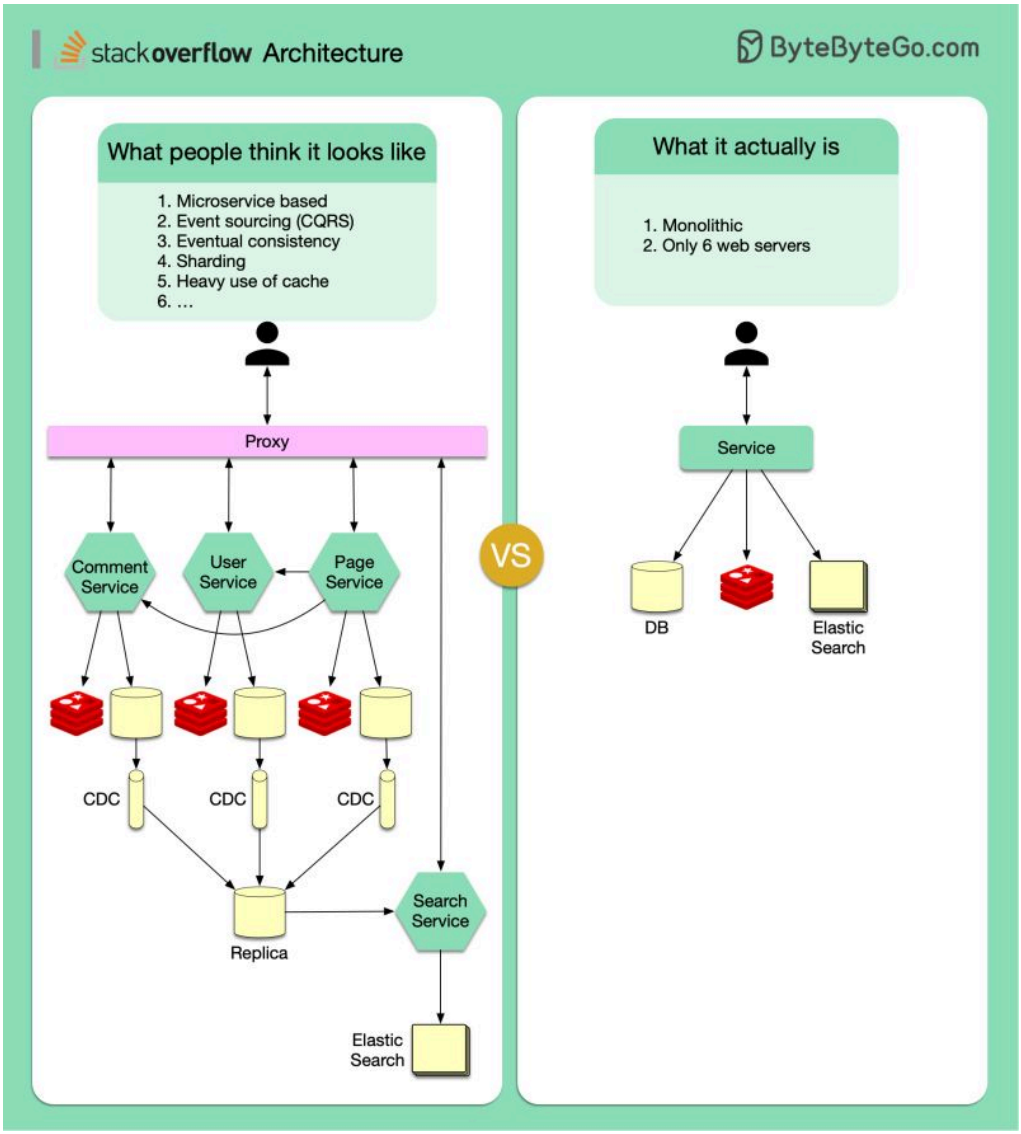
100-1,000 employees: When a company's go-to-market strategy proves successful, and the product scales and grows rapidly, it starts to optimize its engineering efficiency. More commercial tools can be purchased, such as Atlassian products. A certain level of standardization across tools is introduced, 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.

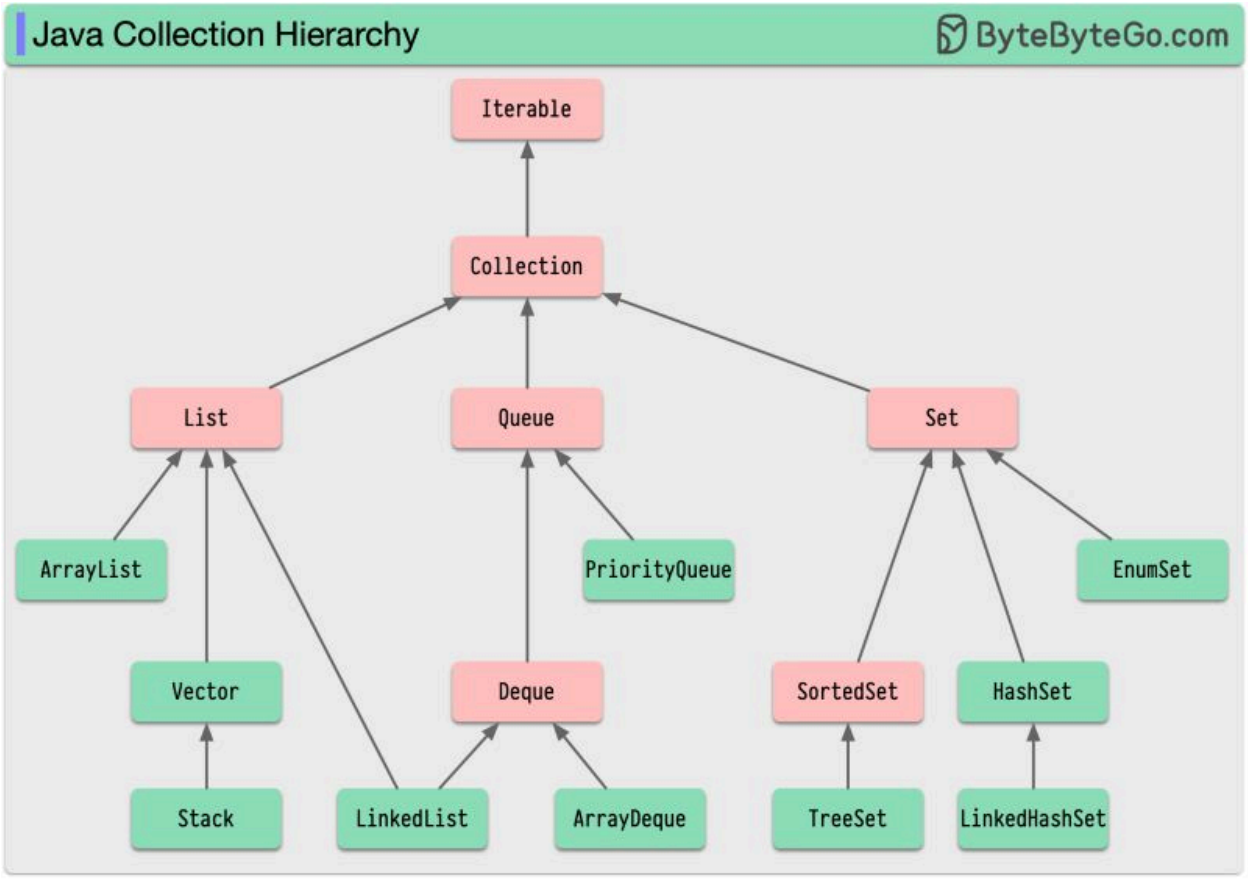
SQL Server has 1.5TB of RAM with no caching layer.

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.

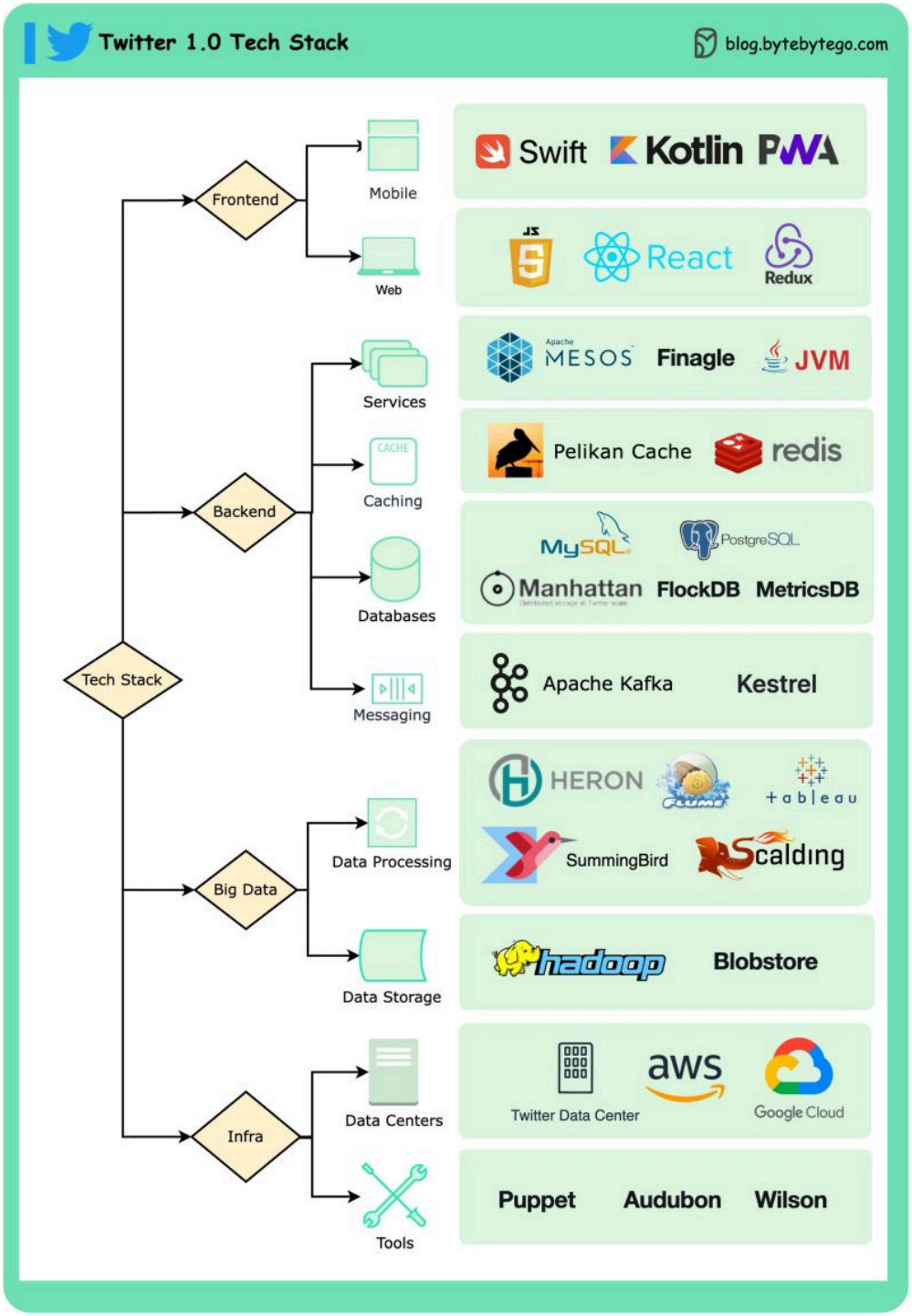
Java engineers need to be familiar with the Java Collection hierarchy to make informed decisions when choosing the right data structure for a particular problem. We can write more

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?

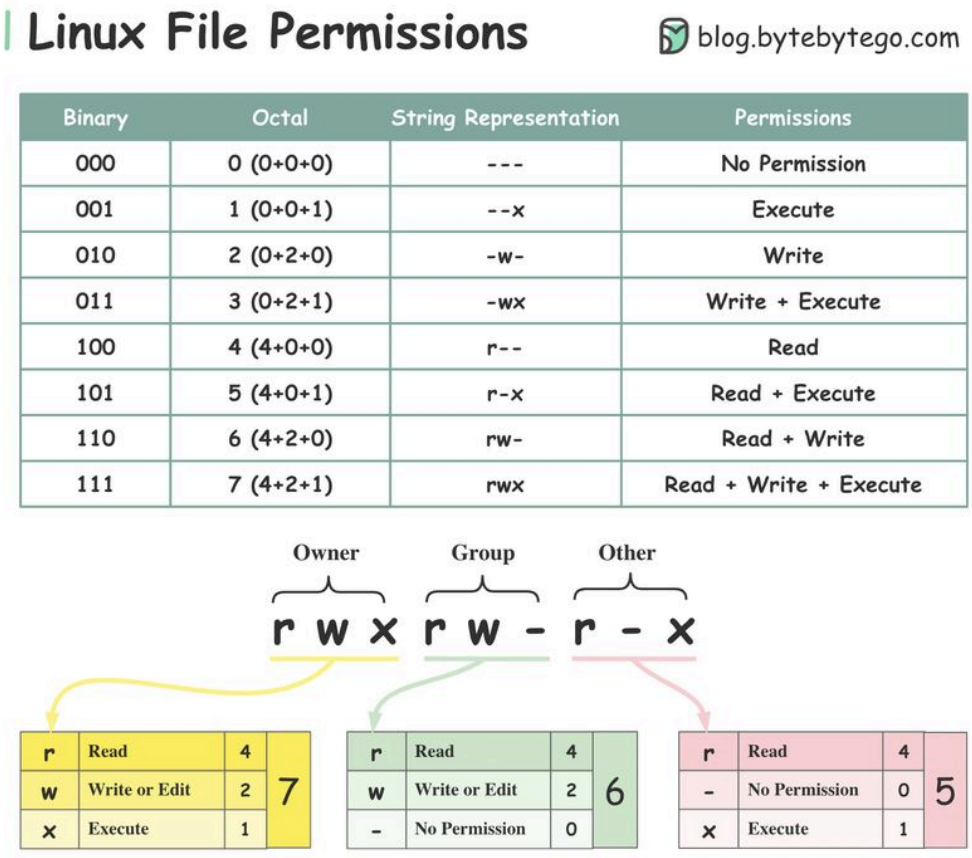
Twitter 1.0 Tech Stack

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



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.

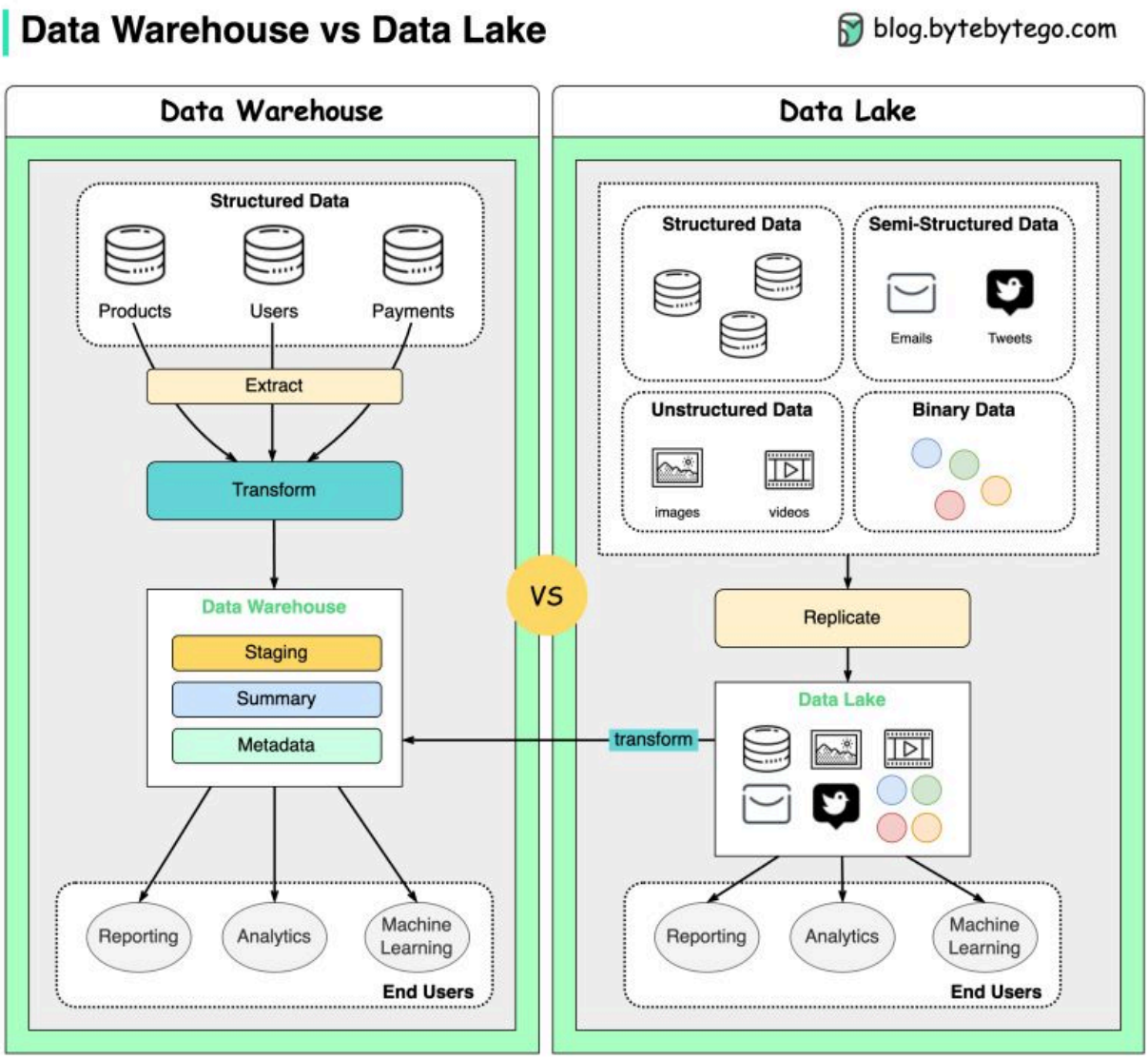
Permission
There are only three types of permissions for a file or directory.

- 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.



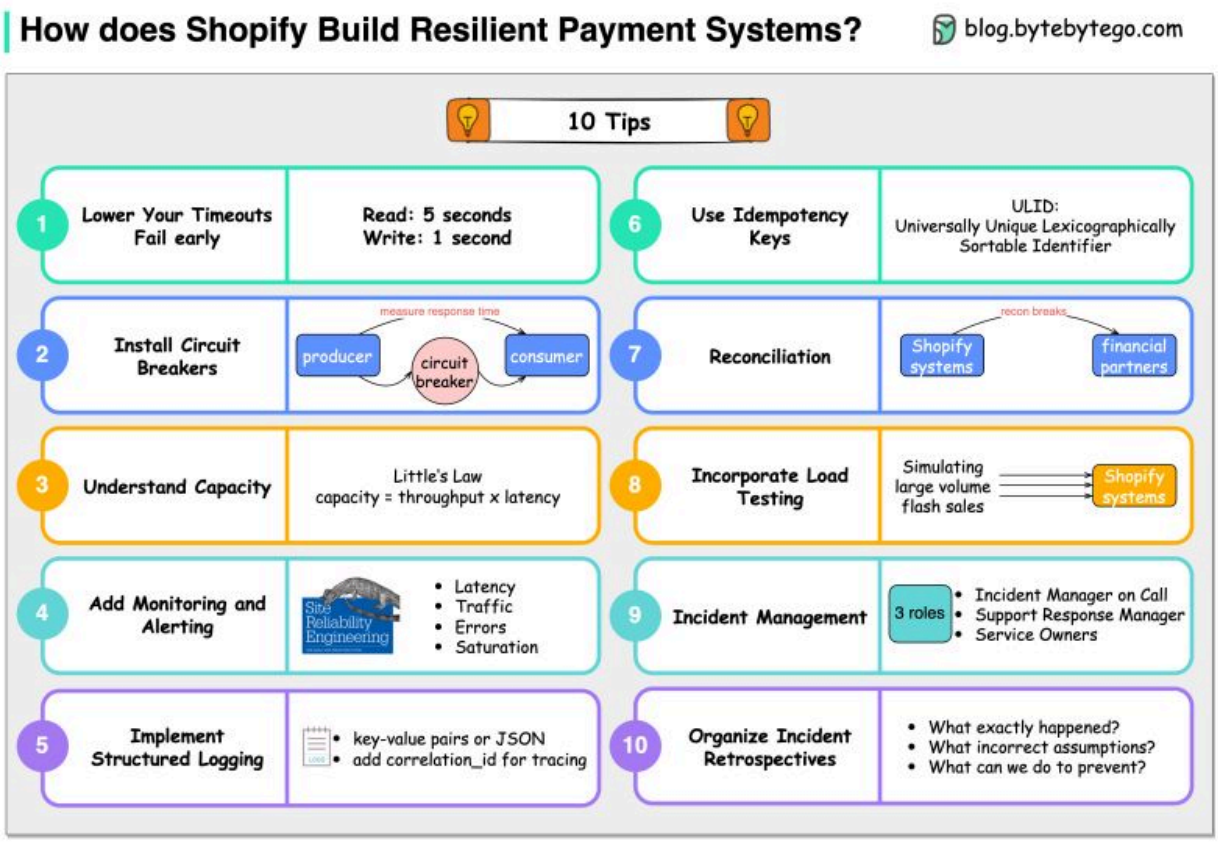
- A data warehouse processes structured data, while a data lake processes structured, semi-structured, unstructured, and raw binary data.
- A data warehouse leverages a database to store layers of structured data, which can be expensive. A data lake stores data in low-cost devices.
- A data warehouse 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.

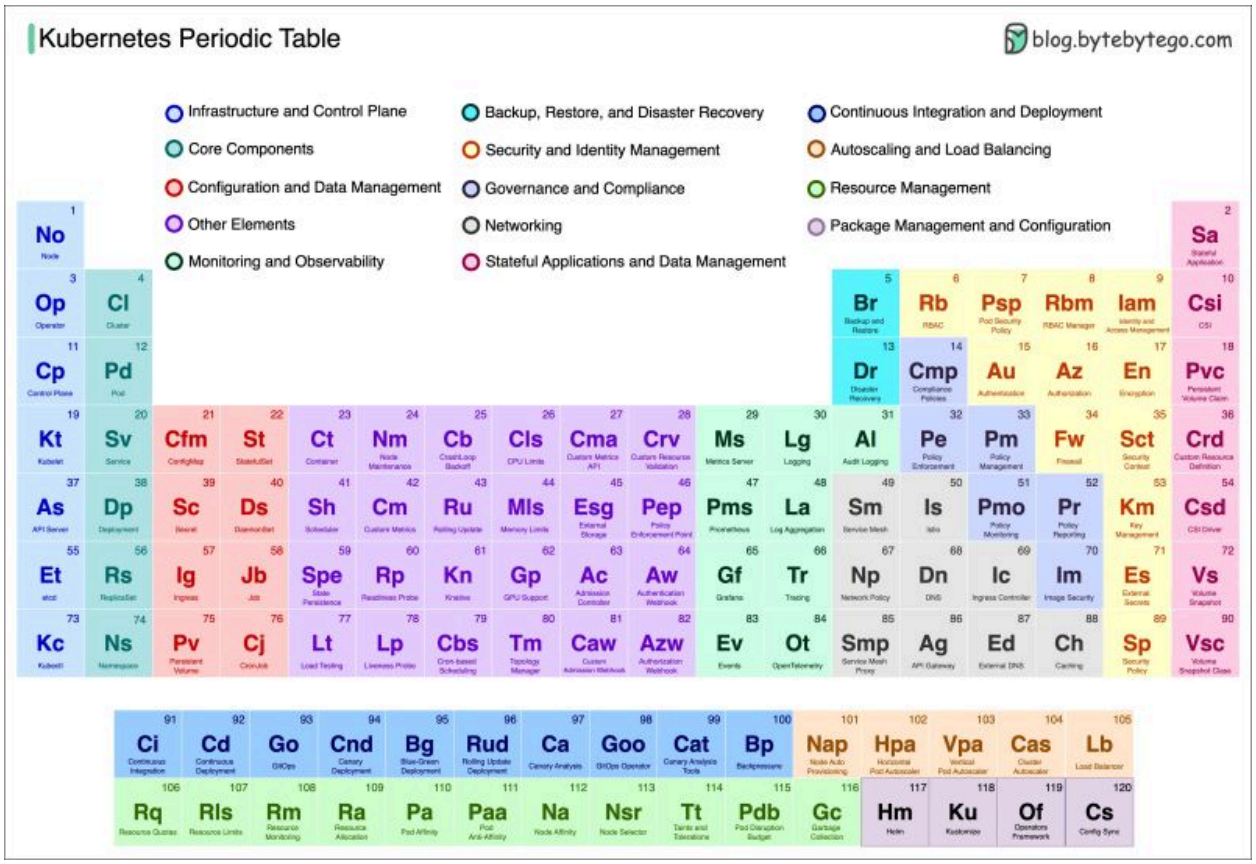
9. Get on top of incident management
   Each incident channel has 3 roles: Incident Manager on Call (IMOC), Support Response Manager (SRM), and service owners.

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

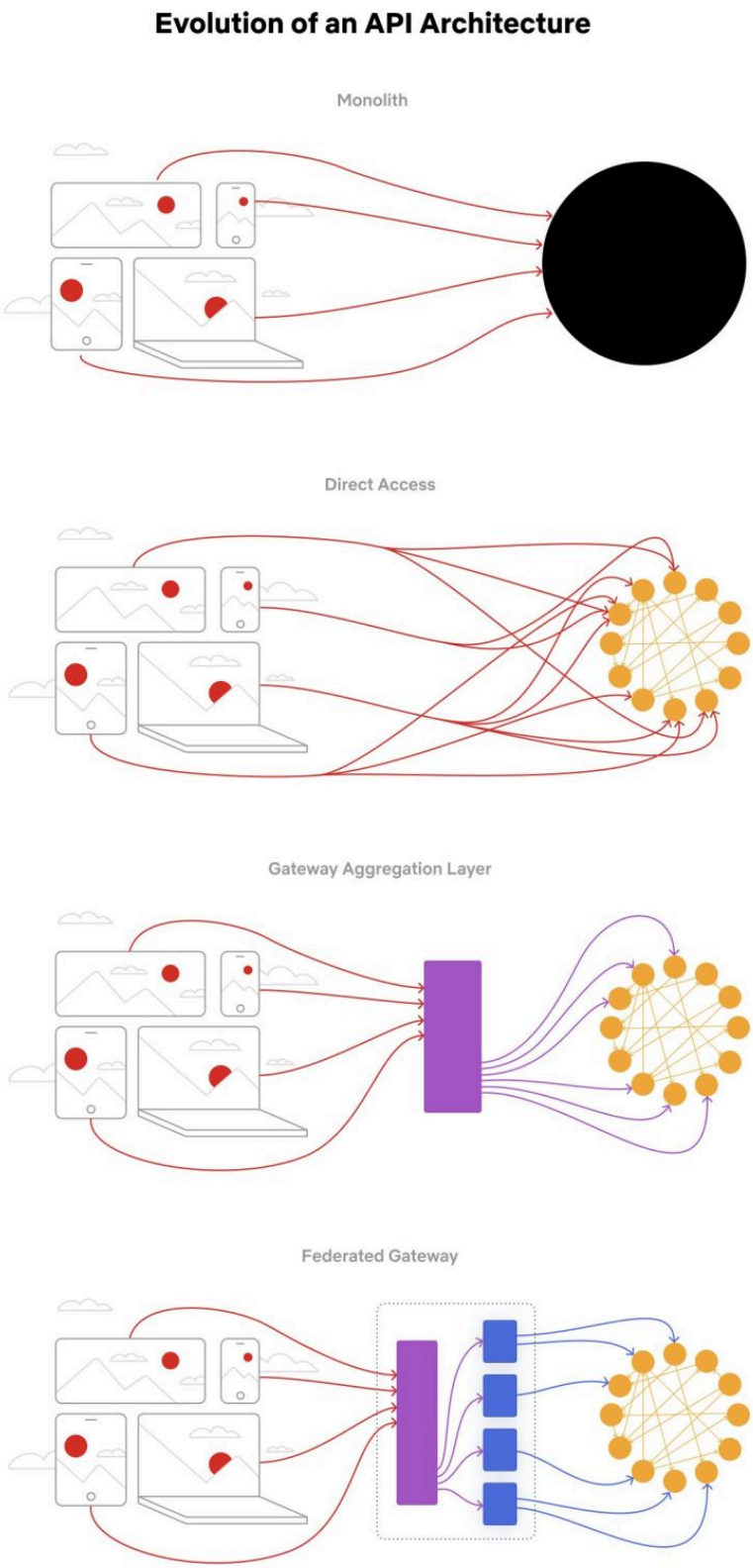
A comprehensive visual guide that demystifies the key building blocks of this powerful container orchestration platform.



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



The Netflix API architecture went through 4 main stages.

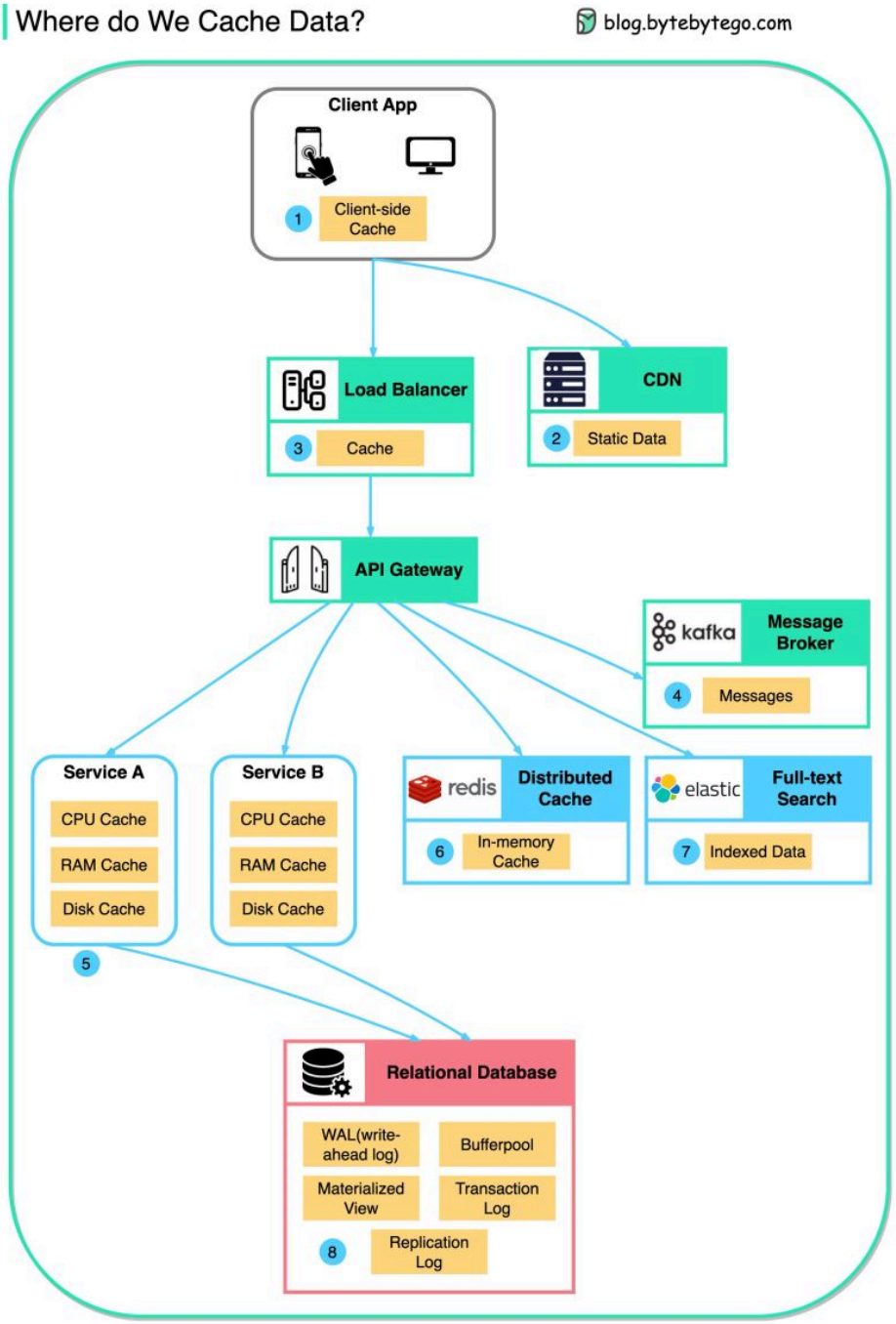
- Monolith
- Direct access
- Gateway aggregation layer
- Federated gateway

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.



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:

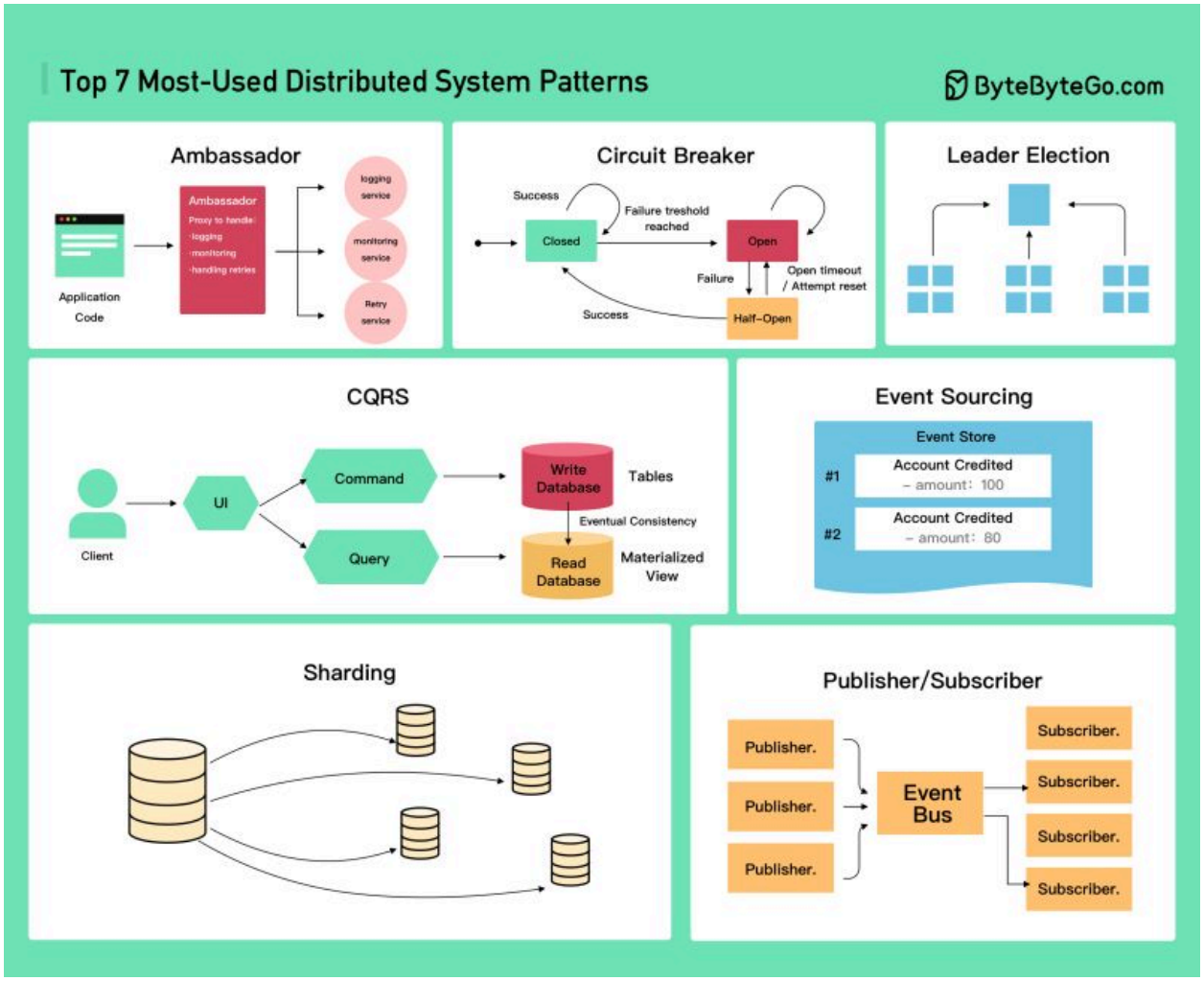
- WAL(Write-ahead Log): data is written to WAL first before building the B tree index
- Bufferpool: A memory area allocated to cache query results
- Materialized View: Pre-compute query results and store them in the database tables for better query performance

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 ↓

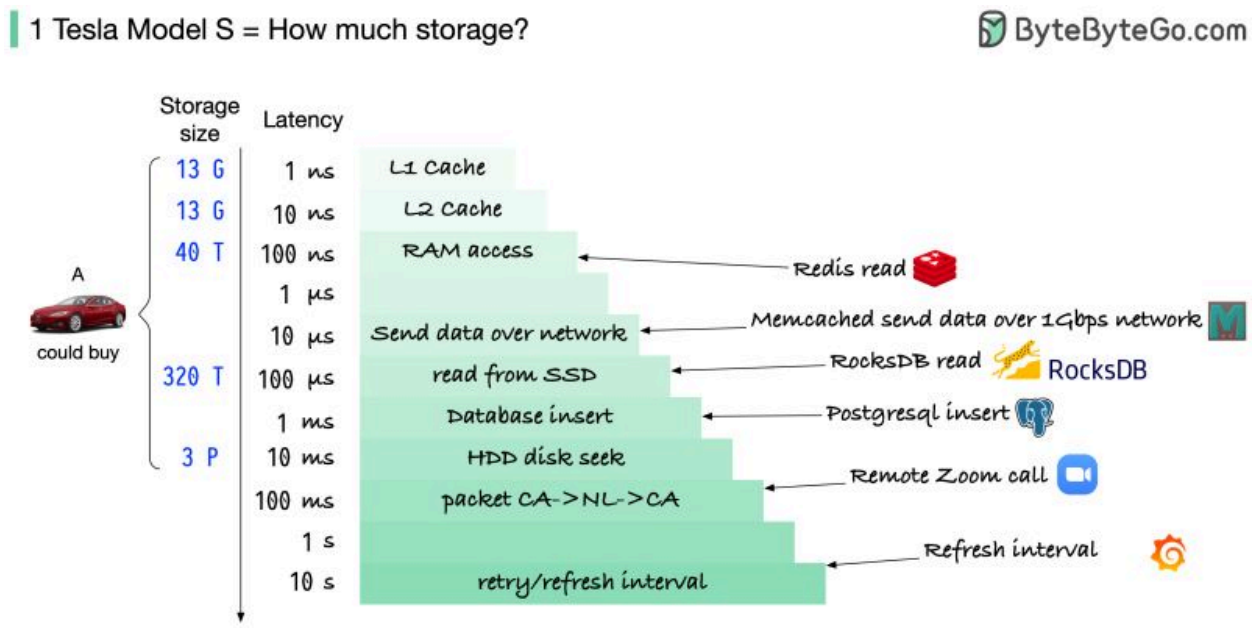


- Ambassador
- Circuit Breaker
- CQRS
- Event Sourcing
- Leader Election
- Publisher/Subscriber
- Sharding

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.

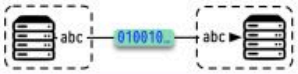


To make this calculation more intriguing, let's use the price of a Tesla Model S as a benchmark. Here are the relevant prices:

- 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 — ByteByteGo.com

	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 friendness	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.

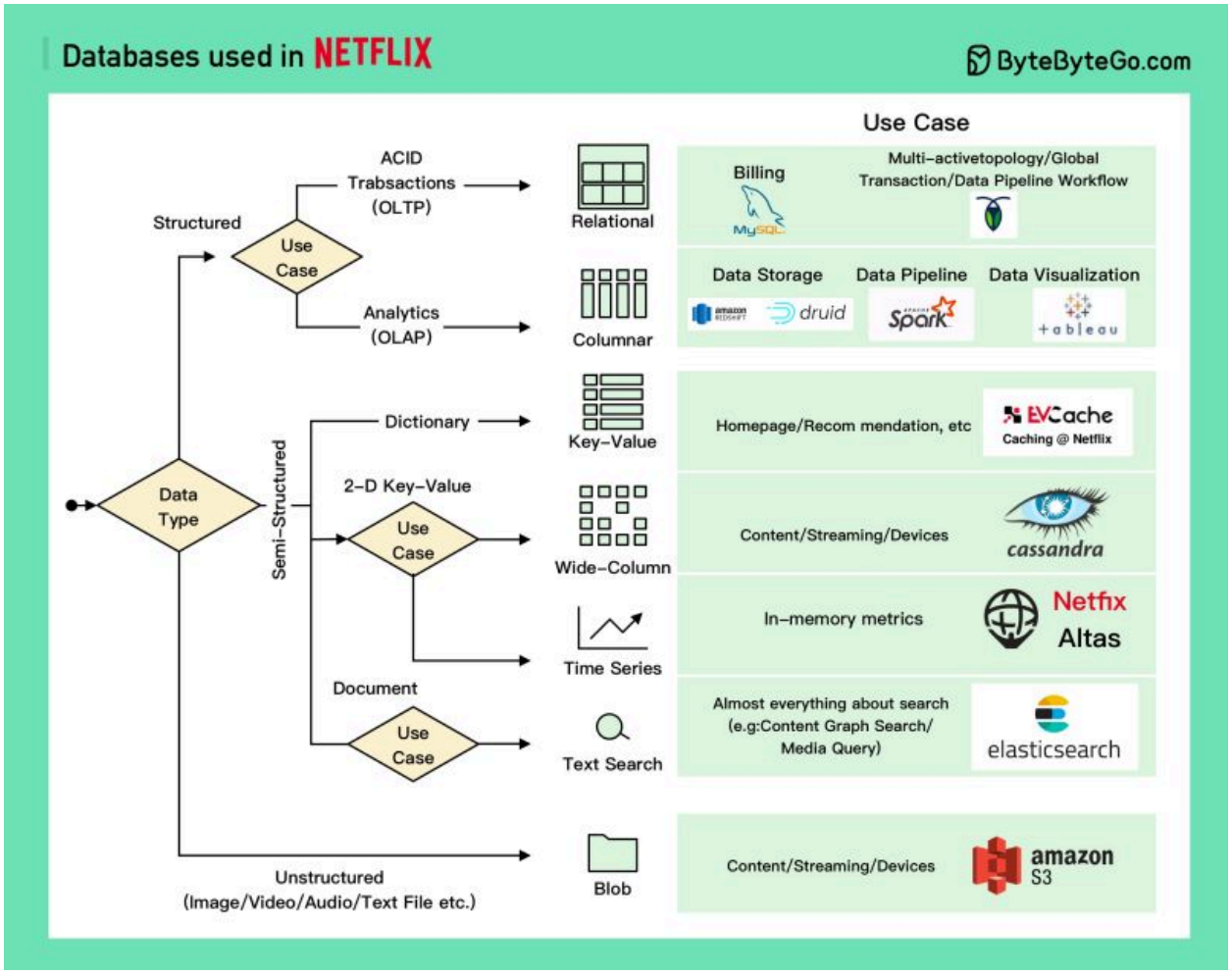
The two protocols differ mainly in their design philosophy. RPC enables calling remote procedures on a server as if they were local procedures, while RESTful applications are resource-based and interact with these resources via HTTP methods.

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.

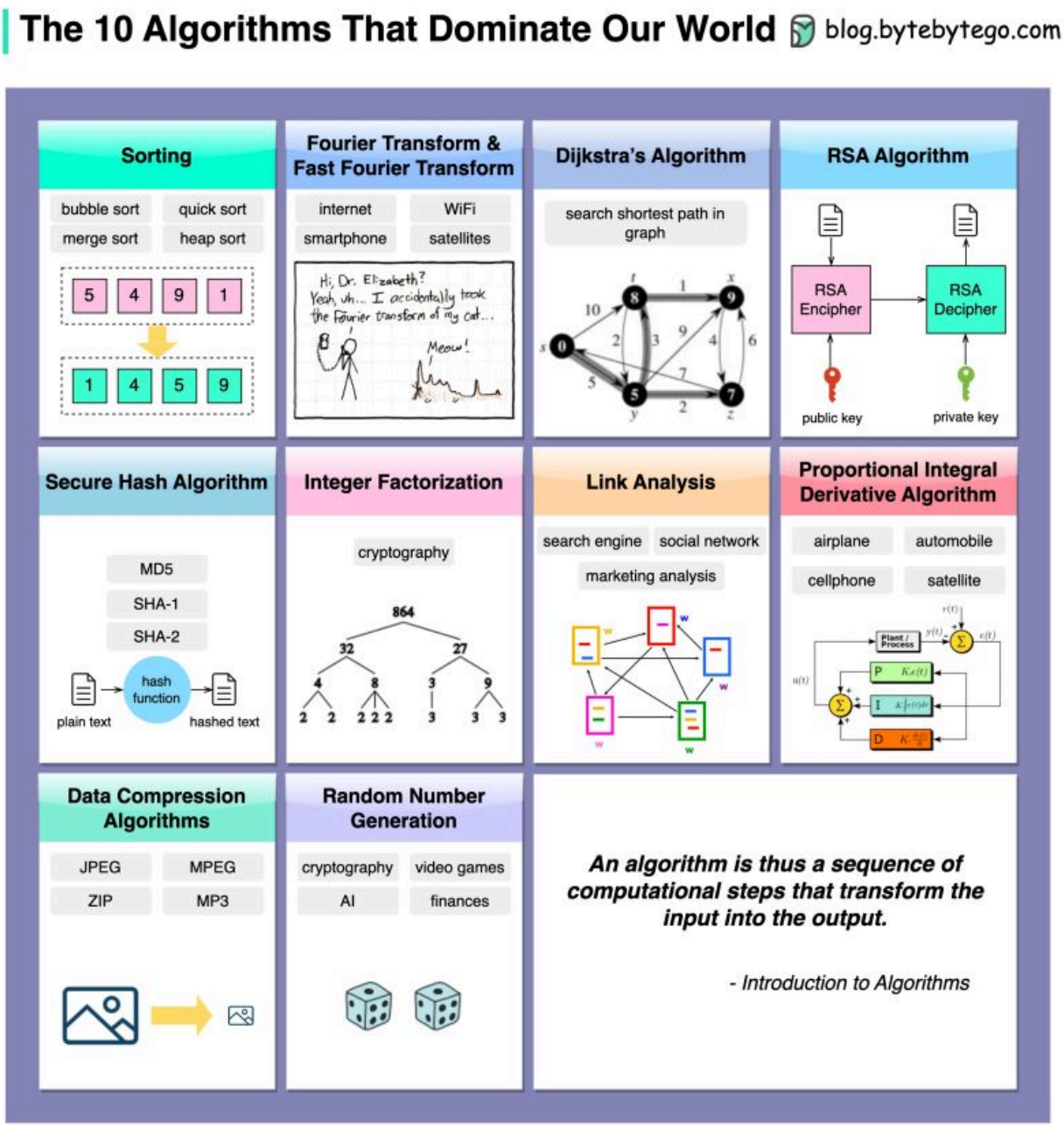
Time-series databases: Netflix built an open-source in-memory database called Atlas for metrics storage and aggregation.

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.



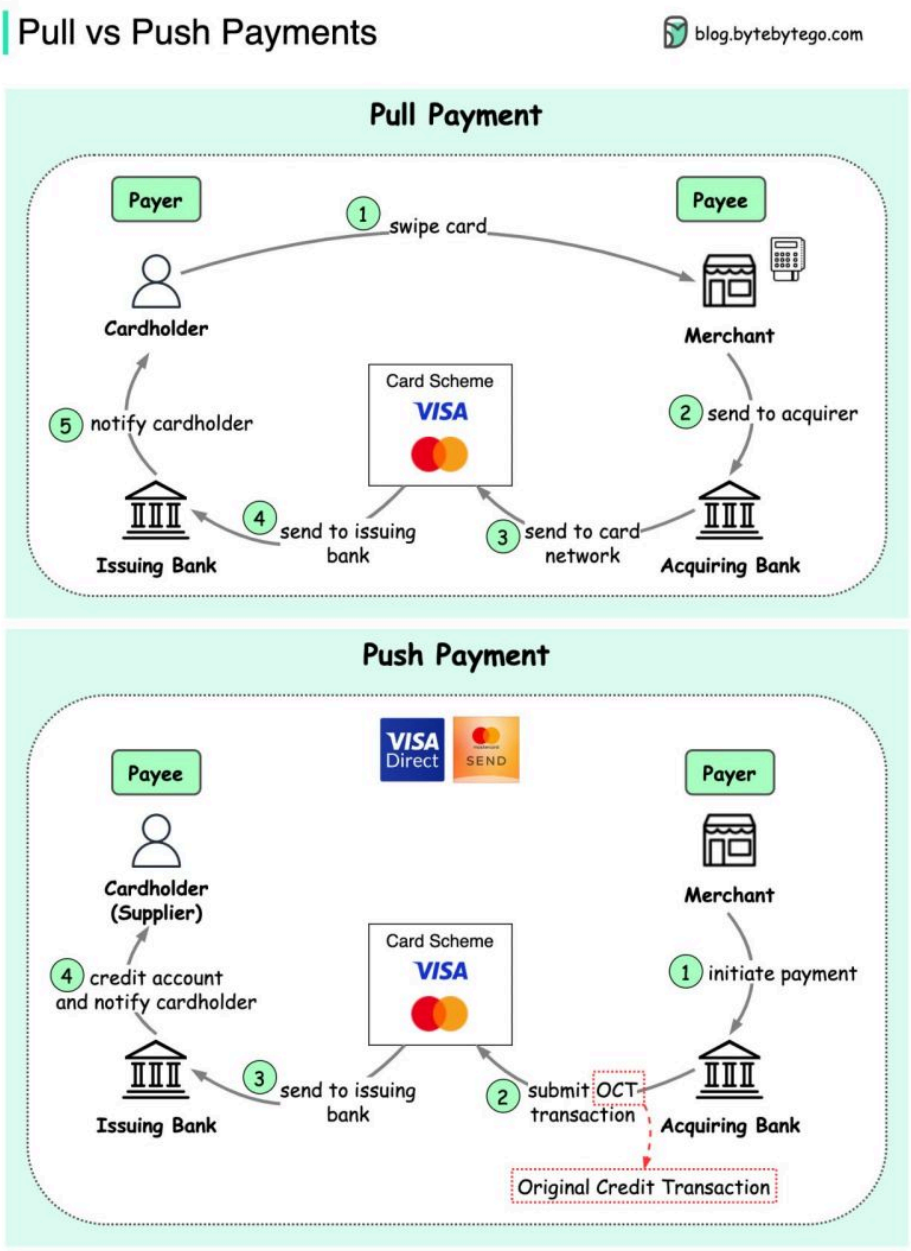
1. Sorting
2. Fourier Transform and Fast Fourier Transform
3. Dijkstra's algorithm

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.

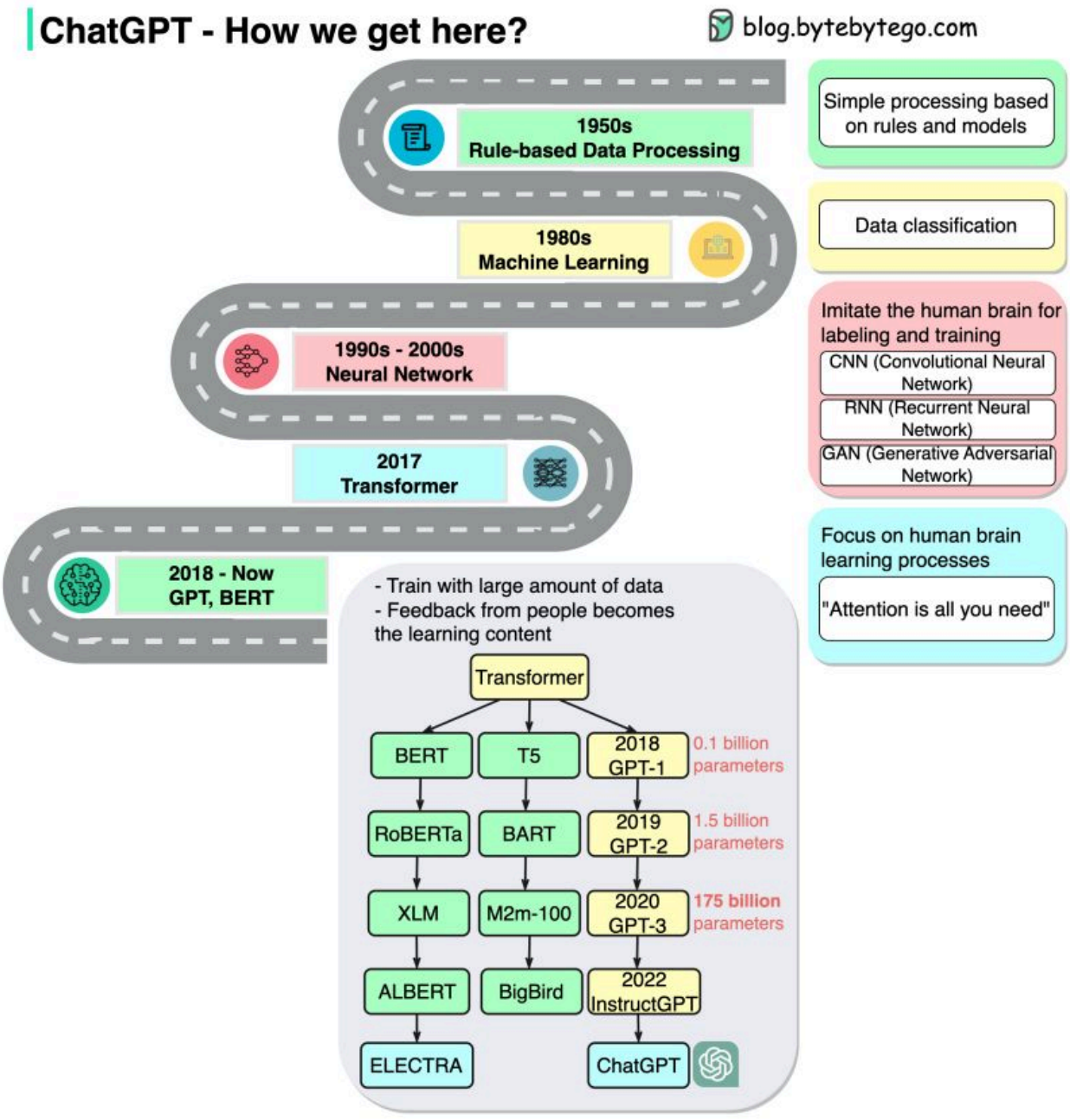
Push payments are indeed an interesting innovation, which complements the digital wallet strategy in Visa and Mastercard. The abstraction of "account" masks the complication of different funding or consuming channels.

Over to you: What is your most frequently used payment method? Is it pull-based or push-based?

ChatGPT - timeline

A picture is worth a thousand words. ChatGPT seems to come out of nowhere. Little did we know that it was built on top of decades of research.

The diagram below shows how we get here.



- 1950s
  In this stage, people still used primitive models that are based on rules.