FREE

SYSTEM DESIGN

THE BIG ARCHIVE

MAY-17-2022

ByteByteGo

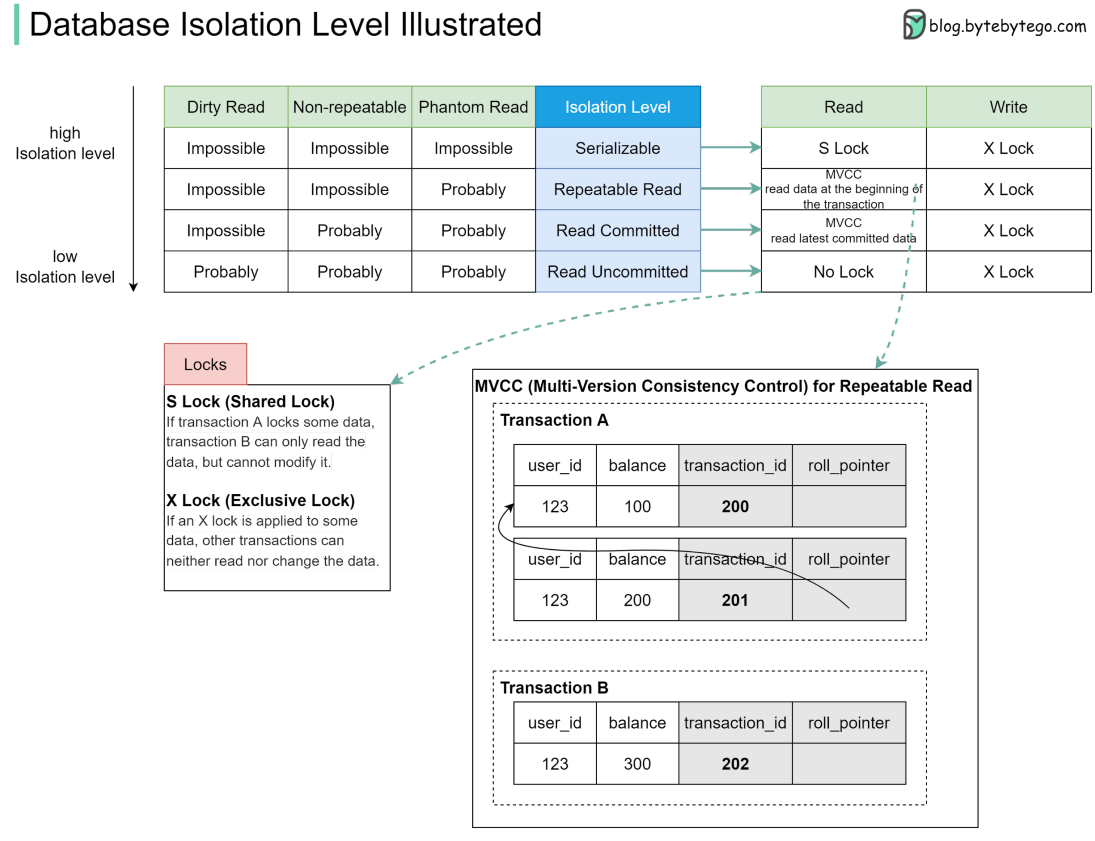
By ByteByteGo.com

System Design

What are database isolation levels? What are they used for?

Database isolation allows a transaction to execute as if there are no other concurrently running transactions.

The diagram below illustrates four isolation levels.



🔹 Serializalble: This is the highest isolation level. Concurrent transactions are guaranteed to be executed in sequence.

🔹 Repeatable Read: Data read during the transaction stays the same as the transaction starts.

🔹 Read Committed: Data modification can only be read after the transaction is committed.

🔹 Read Uncommitted: The data modification can be read by other transactions before a transaction is committed.

The isolation is guaranteed by MVCC (Multi-Version Consistency Control) and locks.

The diagram below takes Repeatable Read as an example to demonstrate how MVCC works:

There are two hidden columns for each row: transaction_id and roll_pointer. When transaction A starts, a new Read View with transaction_id=201 is created. Shortly afterward, transaction B starts, and a new Read View with transaction_id=202 is created.

Now transaction A modifies the balance to 200, a new row of the log is created, and the roll_pointer points to the old row. Before transaction A commits, transaction B reads the balance data. Transaction B finds that transaction_id 201 is not committed, it reads the next committed record(transaction_id=200).

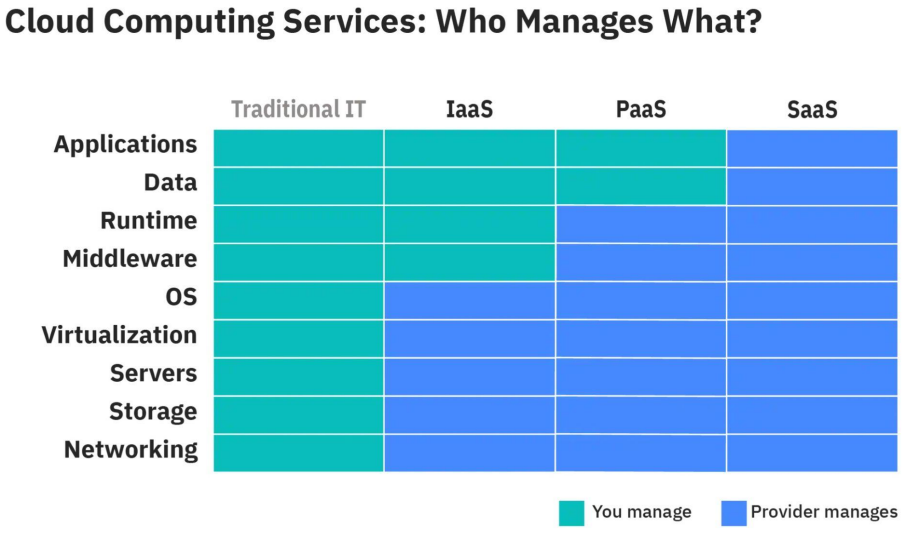
Even when transaction A commits, transaction B still reads data based on the Read View created when transaction B starts. So transaction B always reads the data with balance=100.

Over to you: have you seen isolation levels used in the wrong way? Did it cause serious outages?

—

What is IaaS/PaaS/SaaS?

The diagram below illustrates the differences between IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service), and SaaS (Software-as-a-Service).



For a non-cloud application, we own and manage all the hardware and software. We say the application is on-premises.

With cloud computing, cloud service vendors provide three kinds of models for us to use: IaaS, PaaS, and SaaS.

IaaS provides us access to cloud vendors' infrastructure, like servers, storage, and networking. We pay for the infrastructure service and install and manage supporting software on it for our application.

PaaS goes further. It provides a platform with a variety of middleware, frameworks, and tools to build our application. We only focus on application development and data.

SaaS enables the application to run in the cloud. We pay a monthly or annual fee to use the SaaS product.

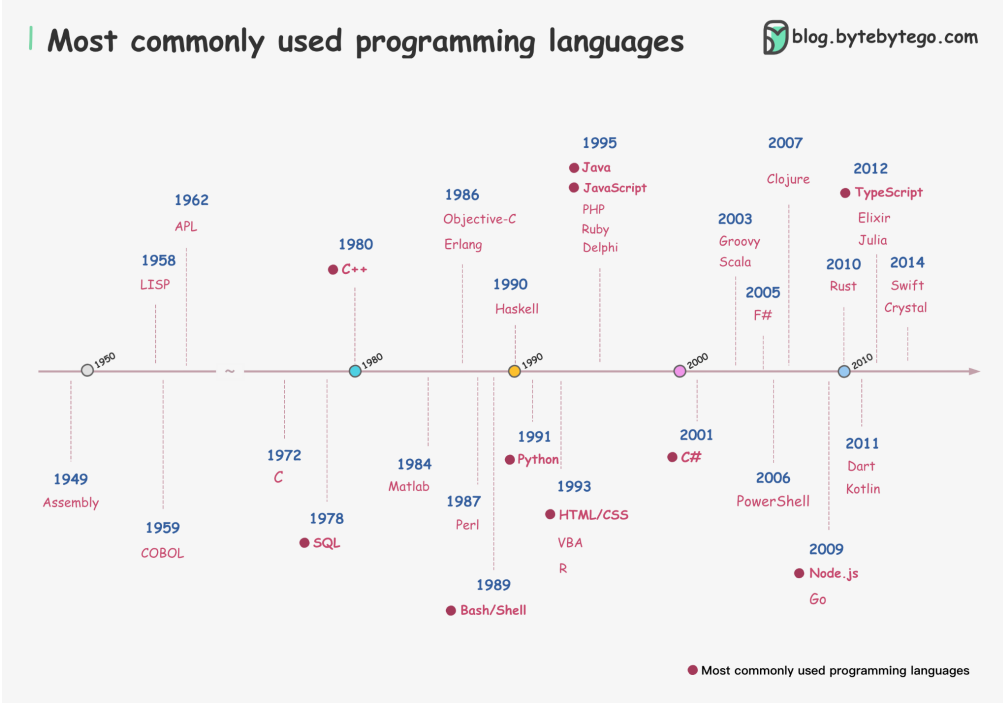
Over to you: which IaaS/PaaS/SaaS products have you used? How do you decide which architecture to use?

Image Source: https://www.ibm.com/cloud/learn/iaas-paas-saas

Most popular programming languages

Programming languages come and go. Some stand the test of time. Some already are shooting stars and some are rising rapidly on the horizon.

I draw a diagram by putting the top 38 most commonly used programming languages in one place, sorted by year. Data source: StackOverflow survey.



1 JavaScript
2 HTML/CSS
3 Python
4 SQL
5 Java
6 Node
7 TypeScript
8 C
9 Bash/Shell
10 C
11 PHP

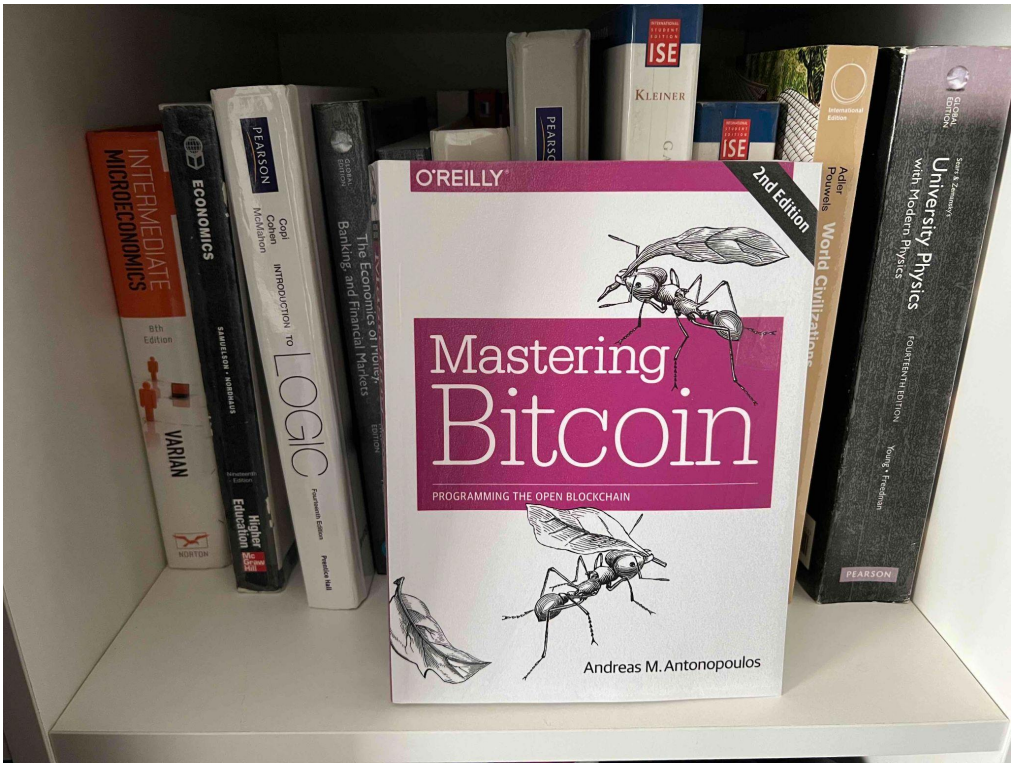
12 C
13 PowerShell
14 Go
15 Kotlin
16 Rust
17 Ruby
18 Dart
19 Assembly
20 Swift
21 R
22 VBA
23 Matlab
24 Groovy
25 Objective-C
26 Scala
27 Perl
28 Haskell
29 Delphi
30 Clojure
31 Elixir
32 LISP
33 Julia
34 F
35 Erlang
36 APL
37 Crystal
38 COBOL

Over to you: what's the first programming language you learned? And what are the other languages you learned over the years?

What is the future of online payments?

I don’t know the answer, but I do know one of the candidates is the blockchain.

As a fan of technology, I always seek new solutions to old challenges. A book that explains a lot about an emerging payment system is ‘Mastering Bitcoin’ by Andreas M. Antonopoulos. I want to share my discovery of this book with you because it explains very clearly bitcoin and its underlying blockchain. This book makes me rethink how to renovate payment systems.



Here are the takeaways:

1. The bitcoin wallet balance is calculated on the fly, while the traditional wallet balance is stored in the database. You can check chapter 12 of System Design Interview Volume 2, on how to implement a traditional wallet (https://amzn.to/34G2vmC).

2. The golden source of truth for bitcoin is the blockchain, which is also the journal. It's the same if we use Event Sourcing architecture to build a traditional wallet, although there are other options.

3. There is a small virtual machine for bitcoin - and also Ethereum. The virtual machine defines a set of bytecodes to do basic tasks such as validation.

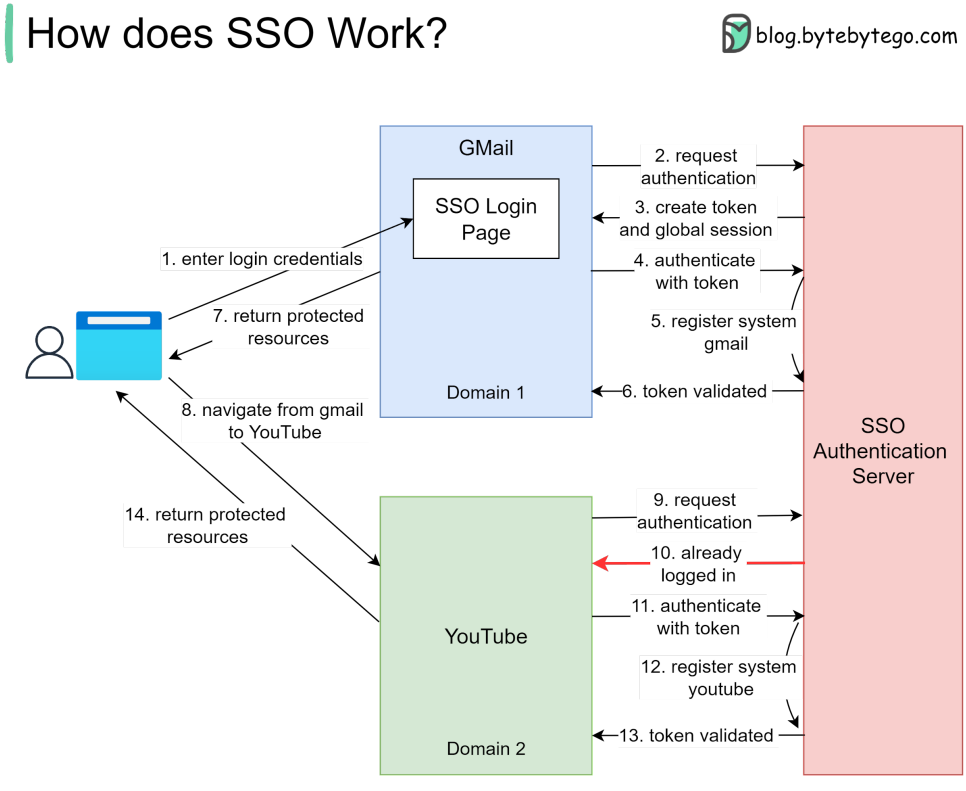
Over to you: if Elon Musk set up a base on planet Mars, what payment solution will you recommend?

What is SSO (Single Sign-On)?

A friend recently went through the irksome experience of being signed out from a number of websites they use daily. This event will be familiar to millions of web users, and it is a tedious process to fix. It can involve trying to remember multiple long-forgotten passwords, or typing in the names of pets from childhood to answer security questions. SSO removes this inconvenience and makes life online better. But how does it work?

Basically, Single Sign-On (SSO) is an authentication scheme. It allows a user to log in to different systems using a single ID.

The diagram below illustrates how SSO works.



Step 1: A user visits Gmail, or any email service. Gmail finds the user is not logged in and so redirects them to the SSO authentication server, which also finds the user is not logged in. As a result, the user

is redirected to the SSO login page, where they enter their login credentials.

Steps 2-3: The SSO authentication server validates the credentials, creates the global session for the user, and creates a token.

Steps 4-7: Gmail validates the token in the SSO authentication server. The authentication server registers the Gmail system, and returns “valid.” Gmail returns the protected resource to the user.

Step 8: From Gmail, the user navigates to another Google-owned website, for example, YouTube.

Steps 9-10: YouTube finds the user is not logged in, and then requests authentication. The SSO authentication server finds the user is already logged in and returns the token.

Step 11-14: YouTube validates the token in the SSO authentication server. The authentication server registers the YouTube system, and returns “valid.” YouTube returns the protected resource to the user.

The process is complete and the user gets back access to their account.

Over to you:

Question 1: have you implemented SSO in your projects? What is the most difficult part?

Question 2: what’s your favorite sign-in method and why?

—

How to store passwords safely in the database?

Let’s take a look.

Things NOT to do

- Storing passwords in plain text is not a good idea because anyone with internal access can see them.

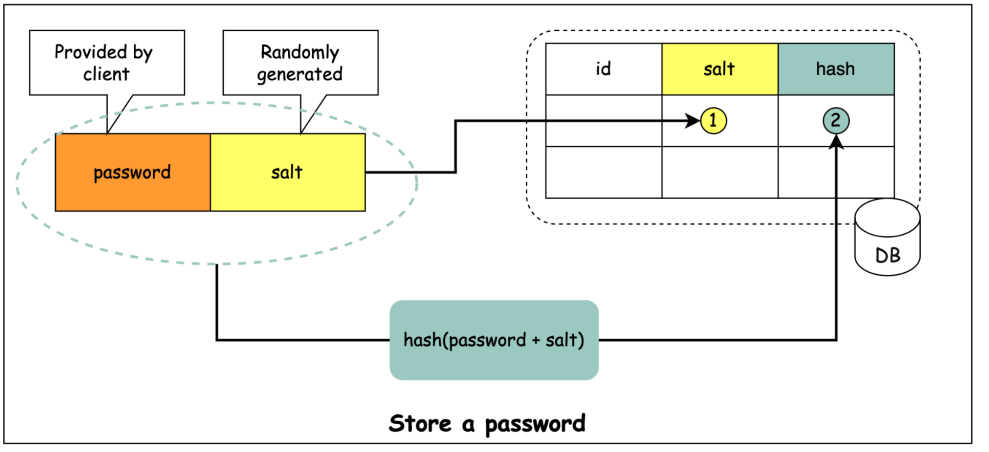
- Storing password hashes directly is not sufficient because it is pruned to precomputation attacks, such as rainbow tables.

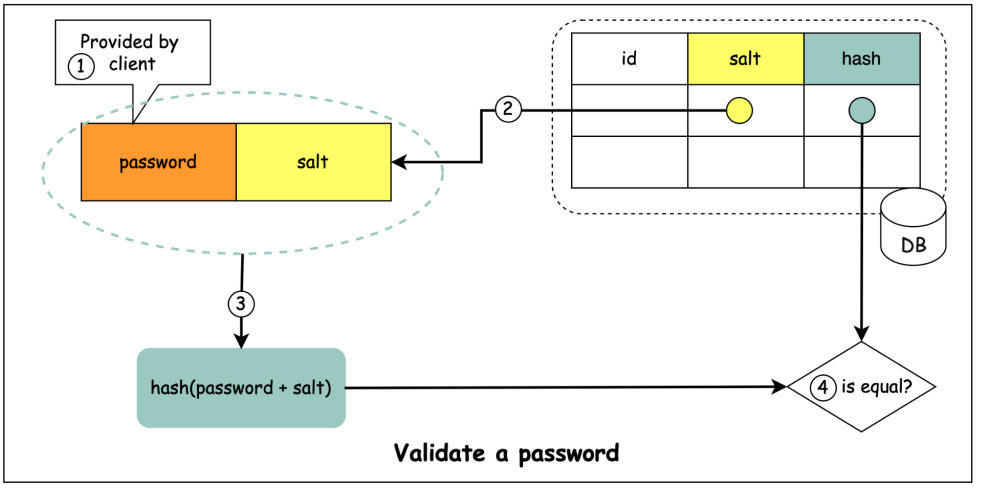
- To mitigate precomputation attacks, we salt the passwords.

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

How to store passwords in DB?

blog.bytebytego.com





How to store a password and salt?

1 A salt is not meant to be secret and it can be stored in plain text in the database. It is used to ensure the hash result is unique to each password.

2 The password can be stored in the database using the following format: *hash*(*password* + *salt*).

How to validate a password?

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

1 A client enters the password.

2 The system fetches the corresponding salt from the database.

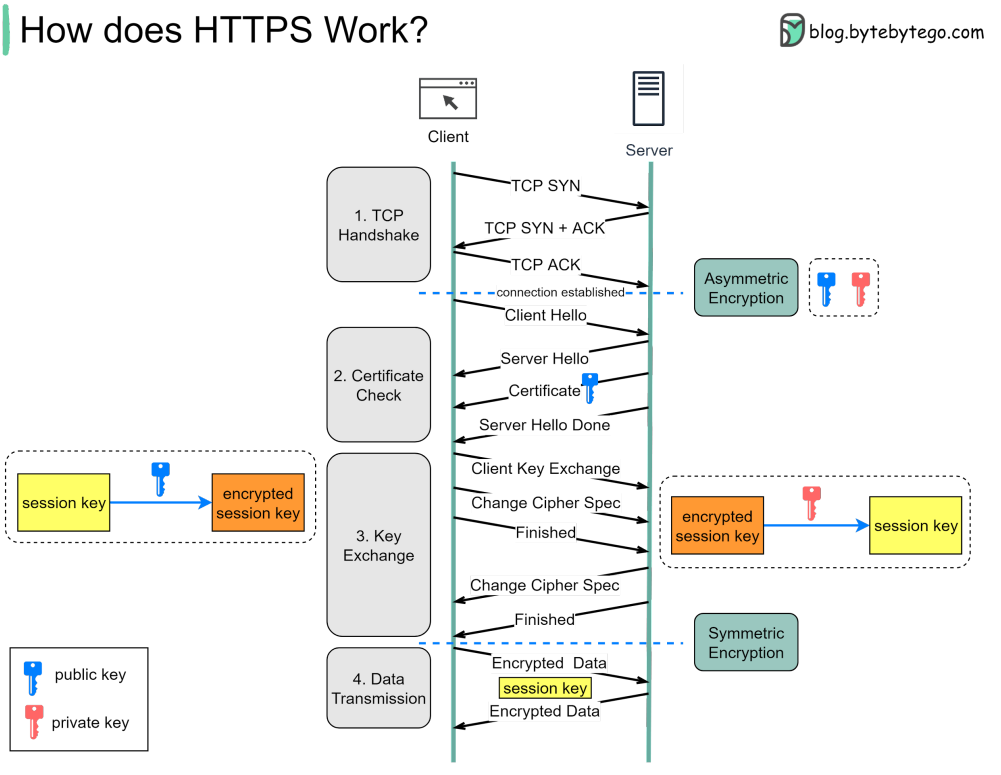
3️⃣ The system appends the salt to the password and hashes it. Let's call the hashed value H1.
4️⃣ The system compares H1 and H2, where H2 is the hash stored in the database. If they are the same, the password is valid.

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

—

How does HTTPS work?

Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP.) HTTPS transmits encrypted data using Transport Layer Security (TLS.) If the data is hijacked online, all the hijacker gets is binary code.



How is the data encrypted and decrypted?

Step 1 - The client (browser) and the server establish a TCP connection.

Step 2 - The client sends a “client hello” to the server. The message contains a set of necessary encryption algorithms (cipher suites) and the latest TLS version it can support. The server responds with a “server hello” so the browser knows whether it can support the algorithms and TLS version.

The server then sends the SSL certificate to the client. The certificate contains the public key, host name, expiry dates, etc. The client validates the certificate.

Step 3 - After validating the SSL certificate, the client generates a session key and encrypts it using the public key. The server receives the encrypted session key and decrypts it with the private key.

Step 4 - Now that both the client and the server hold the same session key (symmetric encryption), the encrypted data is transmitted in a secure bi-directional channel.

Why does HTTPS switch to symmetric encryption during data transmission? There are two main reasons:

1. Security: The asymmetric encryption goes only one way. This means that if the server tries to send the encrypted data back to the client, anyone can decrypt the data using the public key.

2. Server resources: The asymmetric encryption adds quite a lot of mathematical overhead. It is not suitable for data transmissions in long sessions.

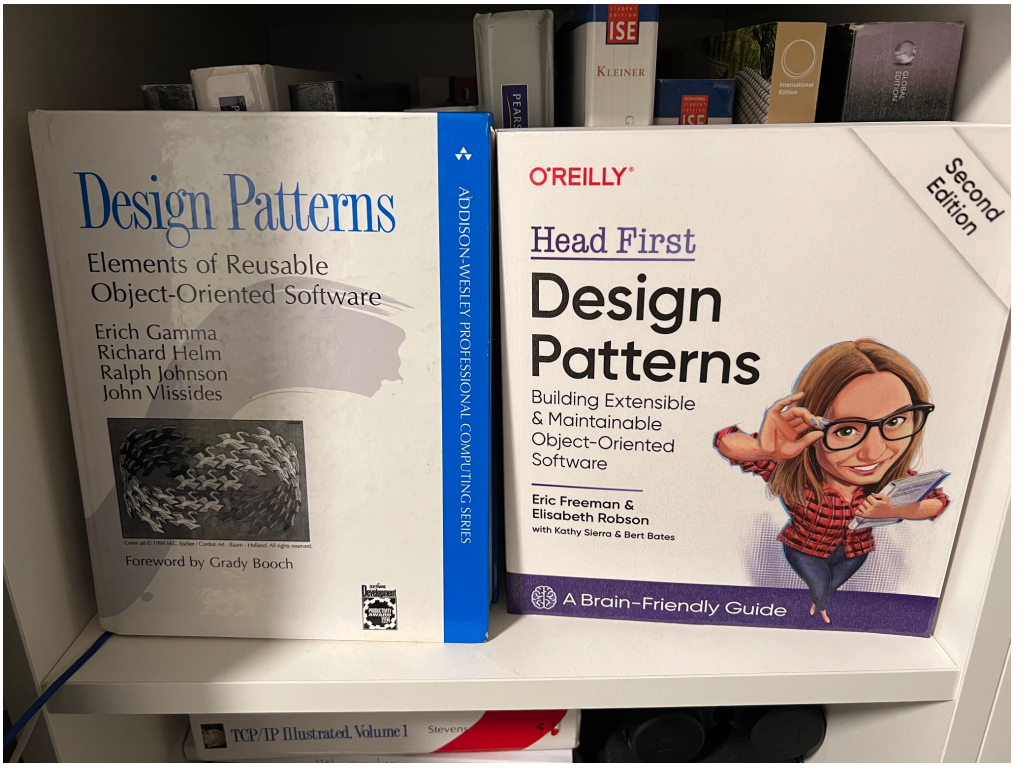
Over to you: how much performance overhead does HTTPS add, compared to HTTP?

—

How to learn design patterns?

Besides reading a lot of well-written code, a good book guides us like a good teacher.

Head First Design Patterns, second edition, is the one I would recommend.



When I began my journey in software engineering, I found it hard to understand the classic textbook, **Design Patterns,** by the Gang of Four. Luckily, I discovered Head First Design Patterns in the school library. This book solved a lot of puzzles for me. When I went back to the Design Patterns book, everything looked familiar and more understandable.

Last year, I bought the second edition of Head First Design Patterns and read through it. Here are a few things I like about the book:

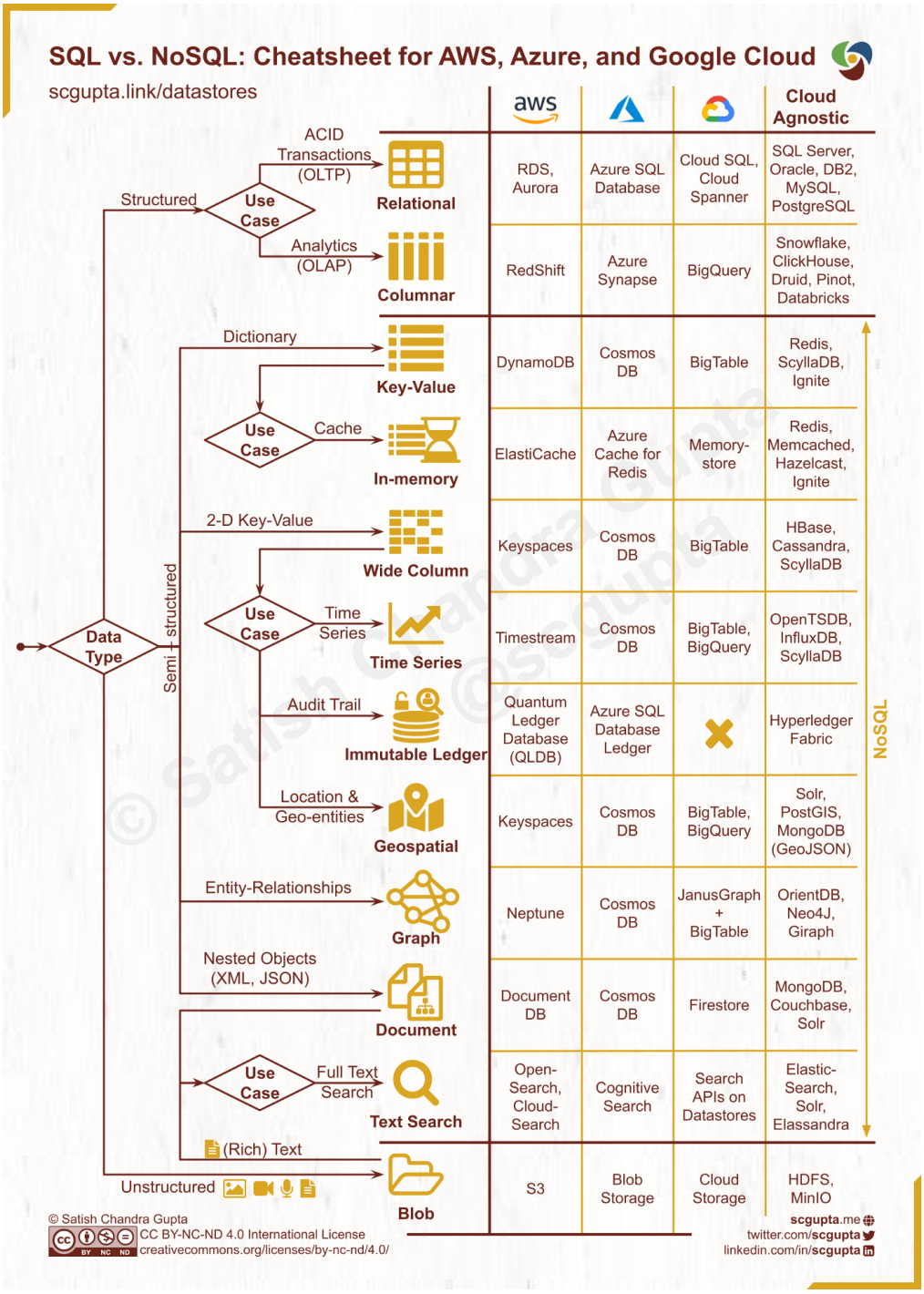
🔹 This book solves the challenge of software's abstract, "invisible" nature. Software is difficult to build because we cannot see its architecture; its details are embedded in the code and binary files. It is even harder to understand software design patterns because these are higher-level abstractions of the software. The book fixes this by using visualization. There are lots of diagrams, arrows, and comments on almost every page. If I do not understand the text, it's no problem. The diagrams explain things very well.

🔹 We all have questions we are afraid to ask when we first learn a new skill. Maybe we think it's an easy one. This book is good at tackling design patterns from the student's point of view. It guides us by asking our questions and clearly answering them. There is a Guru in the book and there's also a Student.

Over to you: which book helped you understand a challenging topic? Why do you like it?

A visual guide on how to choose the right Database

Picking a database is a long-term commitment so the decision shouldn't be made lightly. The important thing to keep in mind is to choose the right database for the right job.



Data can be structured (SQL table schema), semi-structured (JSON, XML, etc.), and unstructured (Blob).

Common database categories include:

- Relational
- Columnar
- Key-value
- In-memory
- Wide column
- Time Series
- Immutable ledger
- Geospatial
- Graph
- Document
- Text search
- Blob

Thanks, Satish Chandra Gupta

Over to you - Which database have you used for which workload?

Do you know how to generate globally unique IDs?

In this post, we will explore common requirements for IDs that are used in social media such as Facebook, Twitter, and LinkedIn.

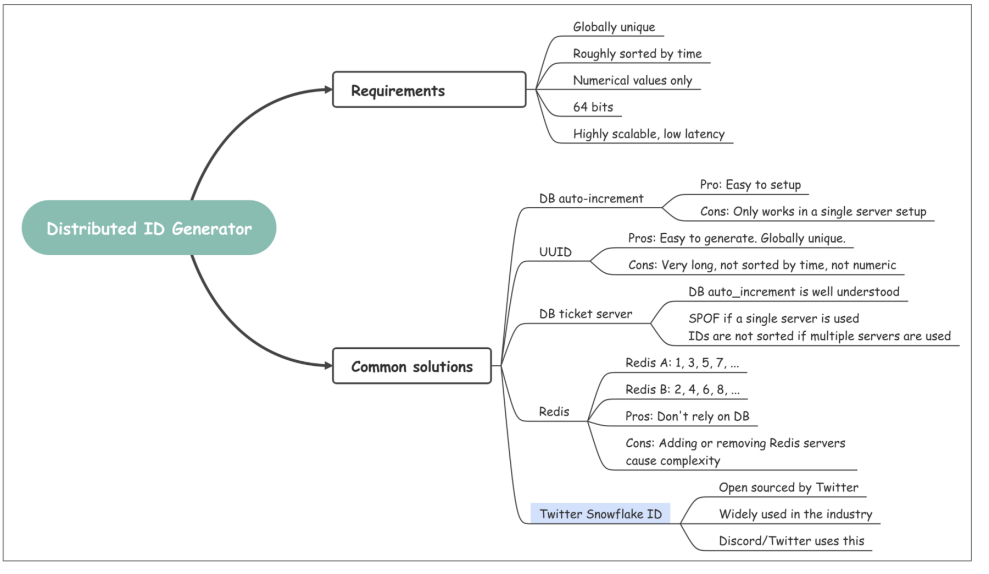
Requirements:

- Globally unique
- Roughly sorted by time
- Numerical values only
- 64 bits
- Highly scalable, low latency

Unique ID Generator

blog.bytebytego.com

	Reasons
Globally unique	If IDs are not globally unique, there could be collisions.
Roughly sorted by time	So user IDs, post IDs can be sorted by time without fetching additional info
Numerical values only	Naturally sortable by time
64 bits	2^32 = ~4 billion -> not enough IDs. 2^64 is big enough 2^128 wastes space and is too long
Highly scalable, low latency	Ability to generate a lot of IDs per second in low latency fashion is critical.



The implementation details of the algorithms can be found online so we will not go into detail here.

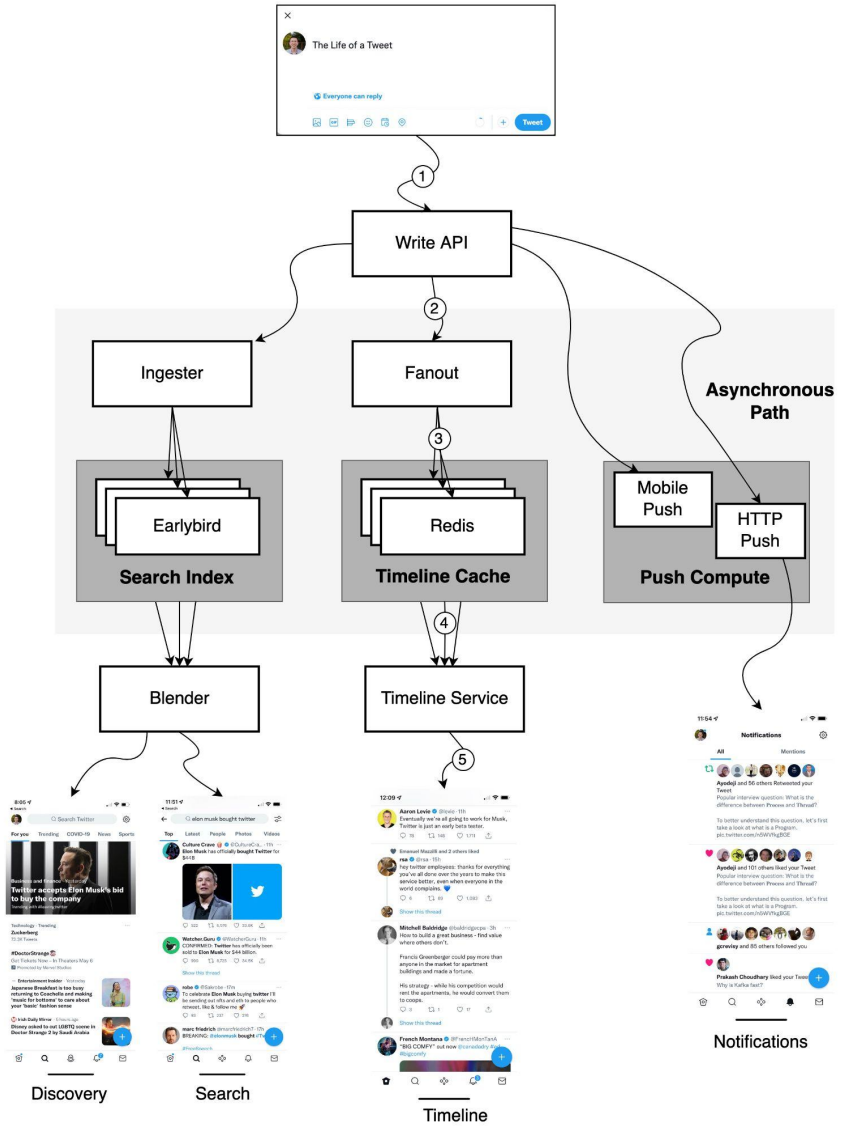
Over to you: What kind of ID generators have you used?

—

Check out our bestselling system design books.
Paperback: Amazon Digital: ByteByteGo.

How does Twitter work?

This post is a summary of a tech talk given by Twitter in 2013. Let's take a look.



The Life of a Tweet:

1. A tweet comes in through the Write API.
2. The Write API routes the request to the Fanout service.
3. The Fanout service does a lot of processing and stores them in the Redis cache.

4️⃣ The Timeline service is used to find the Redis server that has the home timeline on it.
5️⃣ A user pulls their home timeline through the Timeline service.

Search & Discovery

- Ingester: annotates and tokenizes Tweets so the data can be indexed.
- Earlybird: stores search index.
- Blender: creates the search and discovery timelines.

Push Compute

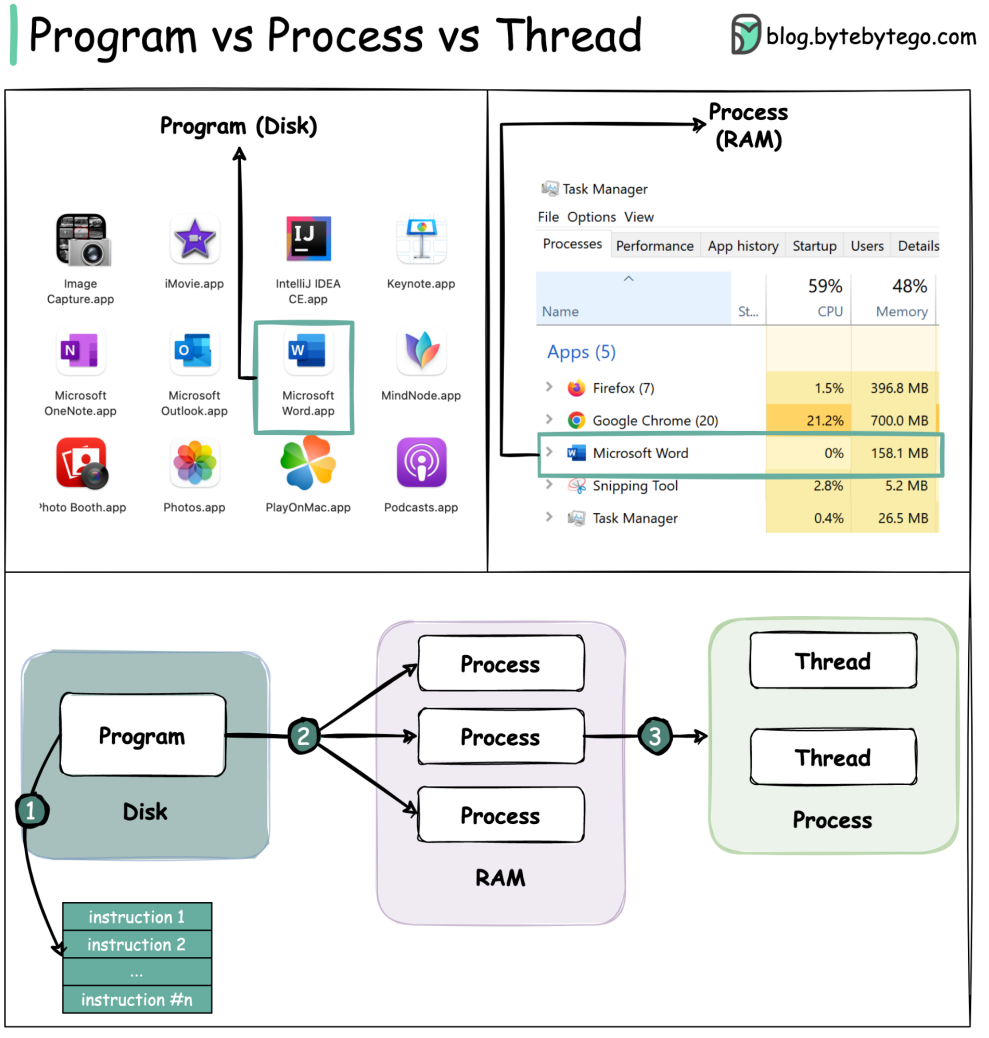
- HTTP push
- Mobile push

Disclaimer: This article is based on the tech talk given by Twitter in 2013 (https://bit.ly/3vNfjRp). Even though many years have passed, it's still quite relevant. I redraw the diagram as the original diagram is difficult to read.

Over to you:
Do you use Twitter? What are some of the biggest differences between LinkedIn and Twitter that might shape their system architectures?

—

What is the difference between Process and Thread?



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

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

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

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

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

Main differences between process and thread:

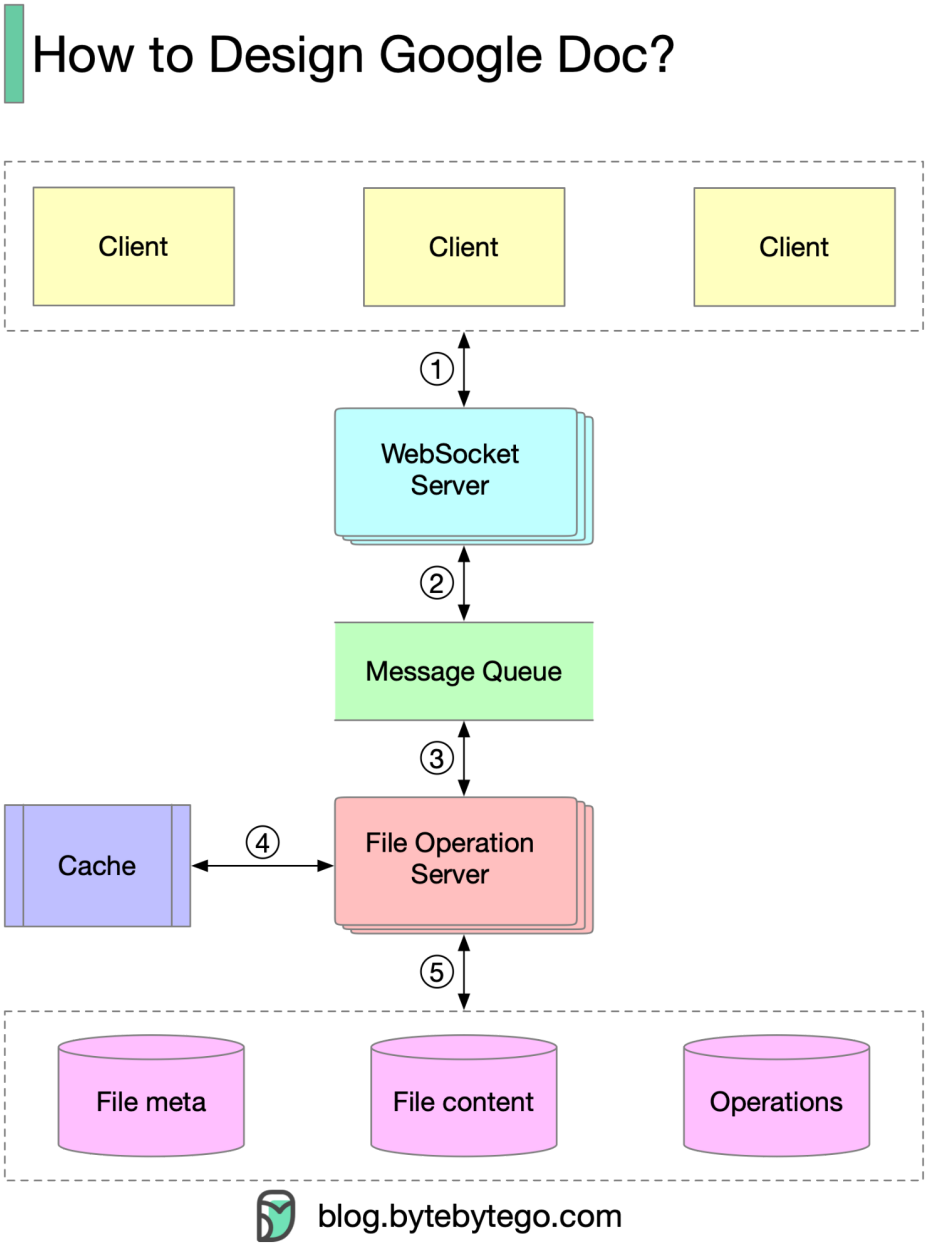
- Processes are usually independent, while threads exist as subsets of a process.
- Each process has its own memory space. Threads that belong to the same process share the same memory.
- A process is a heavyweight operation. It takes more time to create and terminate.
- Context switching is more expensive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

Over to you:
1). Some programming languages support coroutine. What is the difference between coroutine and thread?

2). How to list running processes in Linux?

—

Interview Question: design Google Docs



1 Clients send document editing operations to the WebSocket Server.
2 The real-time communication is handled by the WebSocket Server.
3 Documents operations are persisted in the Message Queue.

4️⃣ The File Operation Server consumes operations produced by clients and generates transformed operations using collaboration algorithms.
5️⃣ Three types of data are stored: file metadata, file content, and operations.

One of the biggest challenges is real-time conflict resolution. Common algorithms include:

- Operational transformation (OT)
- Differential Synchronization (DS)
- Conflict-free replicated data type (CRDT)

Google Doc uses OT according to its Wikipedia page and CRDT is an active area of research for real-time concurrent editing.

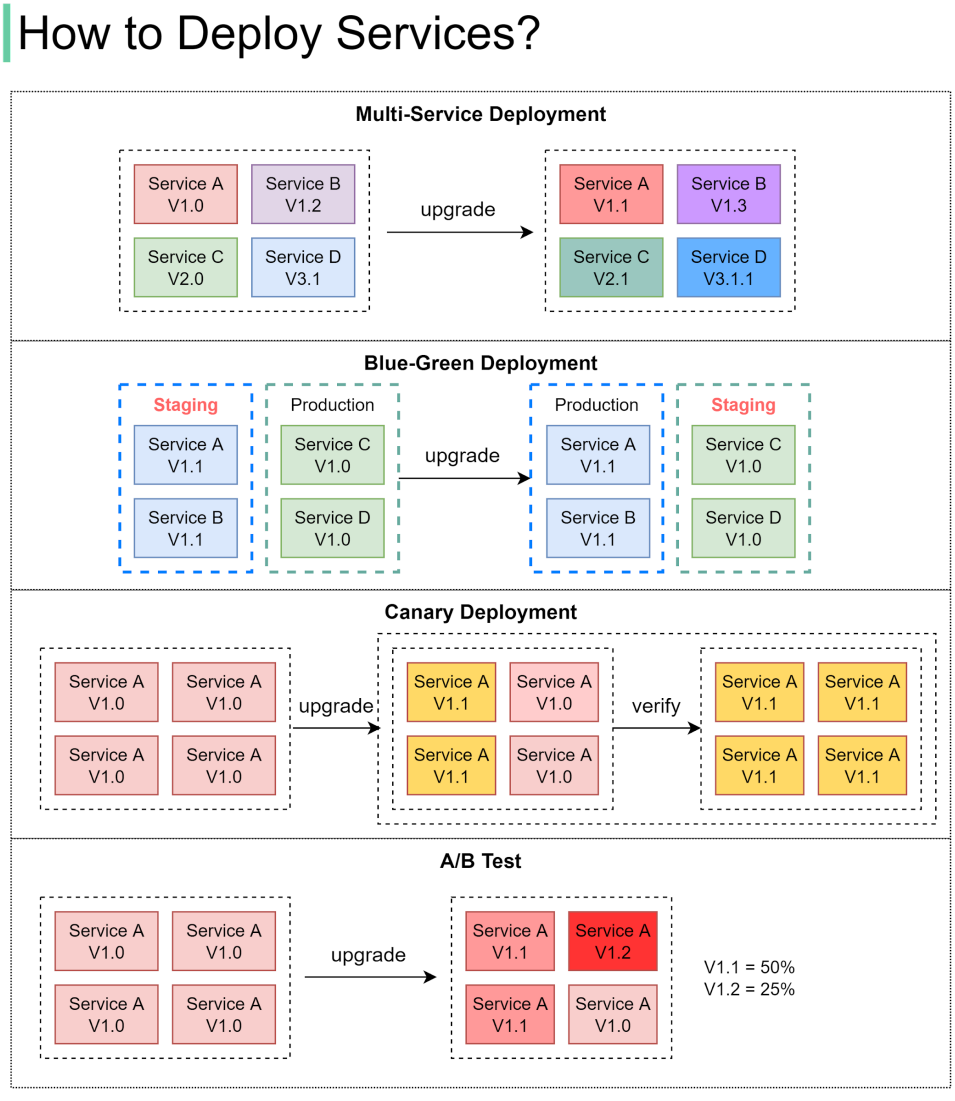
Over to you - Have you encountered any issues while using Google Docs? If so, what do you think might have caused the issue?

—
Check out our bestselling system design books.
Paperback: Amazon Digital: ByteByteGo.

Deployment strategies

Deploying or upgrading services is risky. In this post, we explore risk mitigation strategies.

The diagram below illustrates the common ones.



Multi-Service Deployment

In this model, we deploy new changes to multiple services simultaneously. This approach is easy to implement. But since all the services are upgraded at the same time, it is hard to manage and test dependencies. It's also hard to rollback safely.

Blue-Green Deployment
With blue-green deployment, we have two identical environments: one is staging (blue) and the other is production (green). The staging environment is one version ahead of production. Once testing is done in the staging environment, user traffic is switched to the staging environment, and the staging becomes the production. This deployment strategy is simple to perform rollback, but having two identical production quality environments could be expensive.

Canary Deployment
A canary deployment upgrades services gradually, each time to a subset of users. It is cheaper than blue-green deployment and easy to perform rollback. However, since there is no staging environment, we have to test on production. This process is more complicated because we need to monitor the canary while gradually migrating more and more users away from the old version.

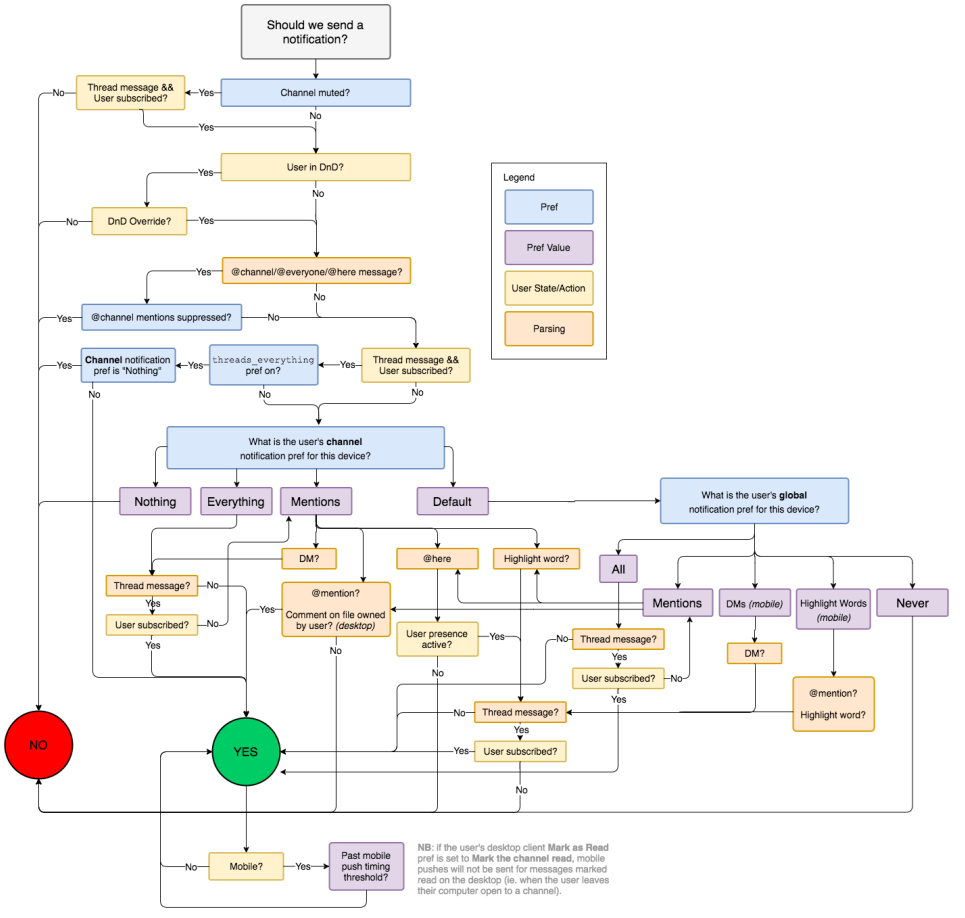
A/B Test
In the A/B test, different versions of services run in production simultaneously. Each version runs an “experiment” for a subset of users. A/B test is a cheap method to test new features in production. We need to control the deployment process in case some features are pushed to users by accident.

Over to you - Which deployment strategy have you used? Did you witness any deployment-related outages in production and why did they happen?

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.

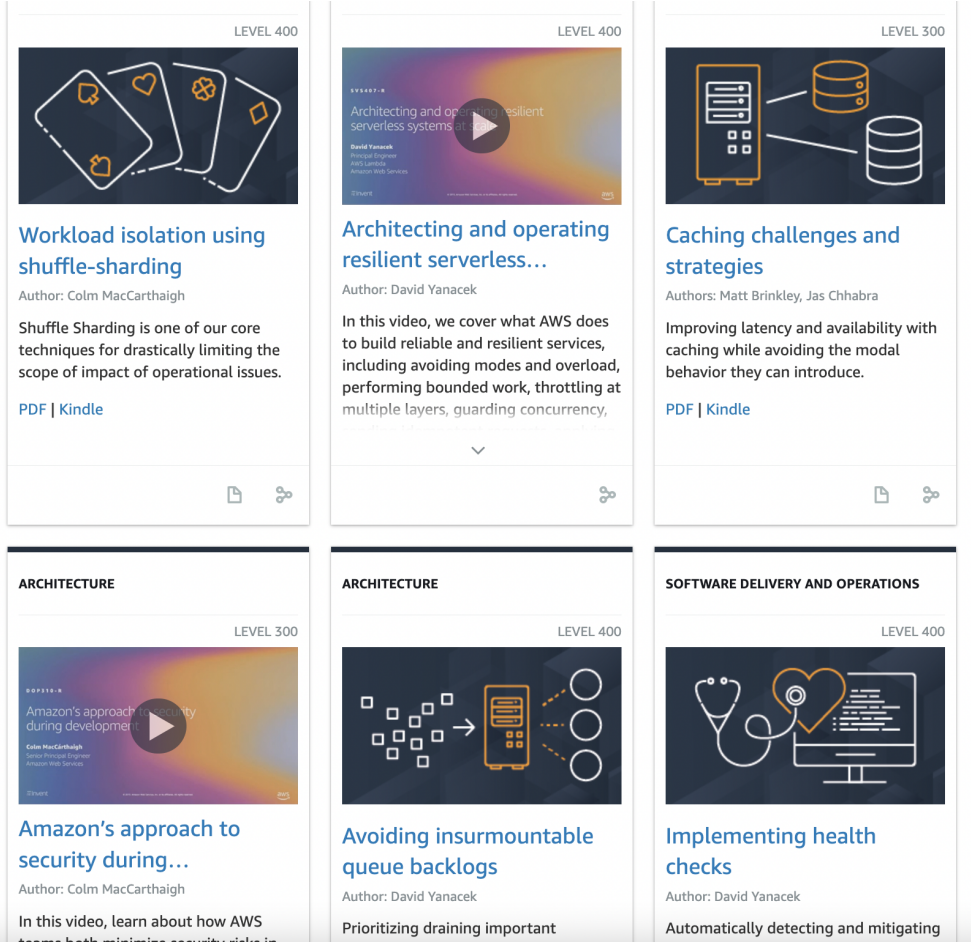


What's your takeaway from this diagram?

Image source:
https://slack.engineering/reducing-slacks-memory-footprint/

How does Amazon build and operate the software?

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



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

- Making retries safe with idempotent APIs
- Timeouts, retries, and backoff with jitter
- Beyond five 9s: Lessons from our highest available data planes
- Caching challenges and strategies
- Ensuring rollback safety during deployments
- Going faster with continuous delivery

- Challenges with distributed systems
- Amazon's approach to high-availability deployment

Over to you: what’s your favorite place to learn system design and design principles?

Link to The Amazon Builders' Library: aws.amazon.com/builders-library

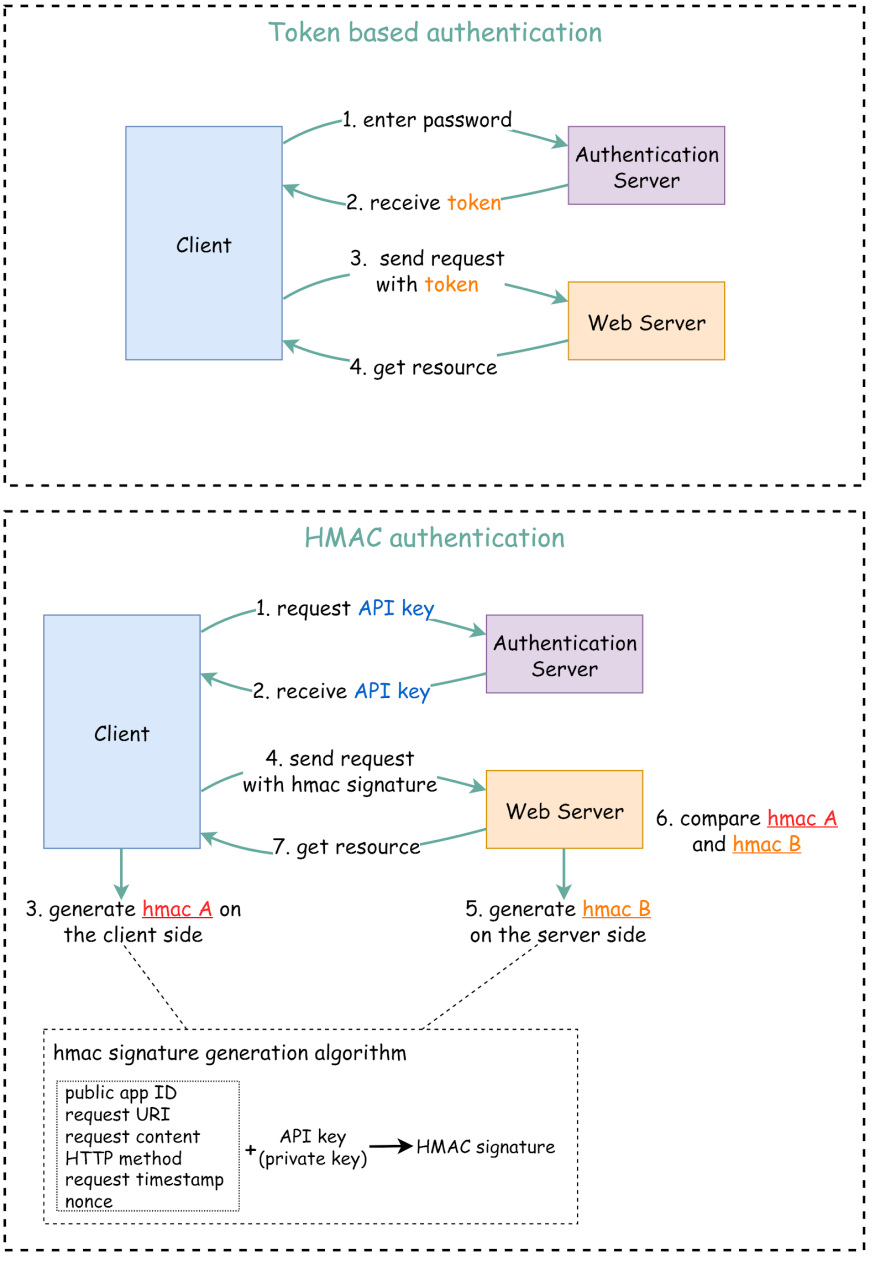
How to design a secure web API access for your website?

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

In this post, we explore two common ways:
1. Token based authentication
2. HMAC (Hash-based Message Authentication Code) authentication

The diagram below illustrates how they work.

How to Design Secure Web API?



Token based

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

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

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

HMAC based

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

Steps 1 and 2 - the server generates two keys, one is Public APP ID (public key) and the other one is API Key (private key).
Step 3 - we now generate a HMAC signature on the client side (hmac A). This signature is generated with a set of attributes listed in the diagram.

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

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

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

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

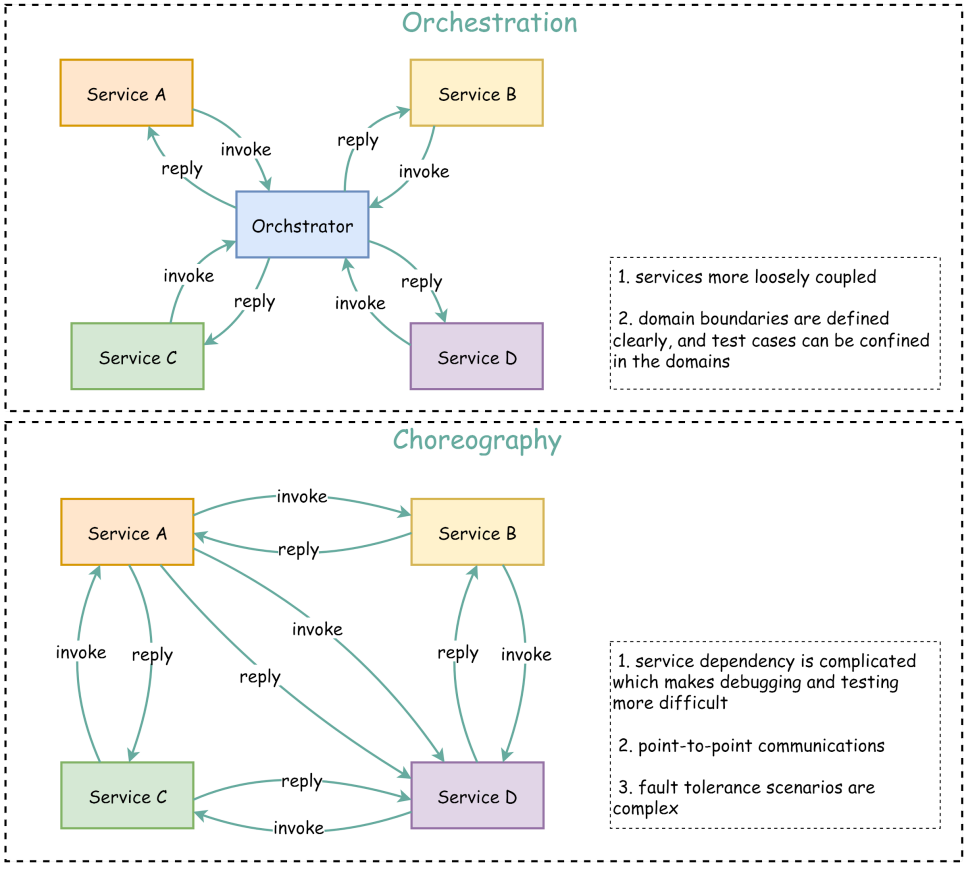
—

How do microservices collaborate and interact with each other?

There are two ways: **orchestration** and **choreography**.

The diagram below illustrates the collaboration of microservices.

> Orchestration v.s. Choreography of Microservices



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

Orchestration is different. The orchestrator acts as a center of authority. It is responsible for invoking and combining the services. It

describes the interactions between all the participating services. It is just like a conductor leading the musicians in a musical symphony. The orchestration pattern also includes the transaction management among different services.

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

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

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

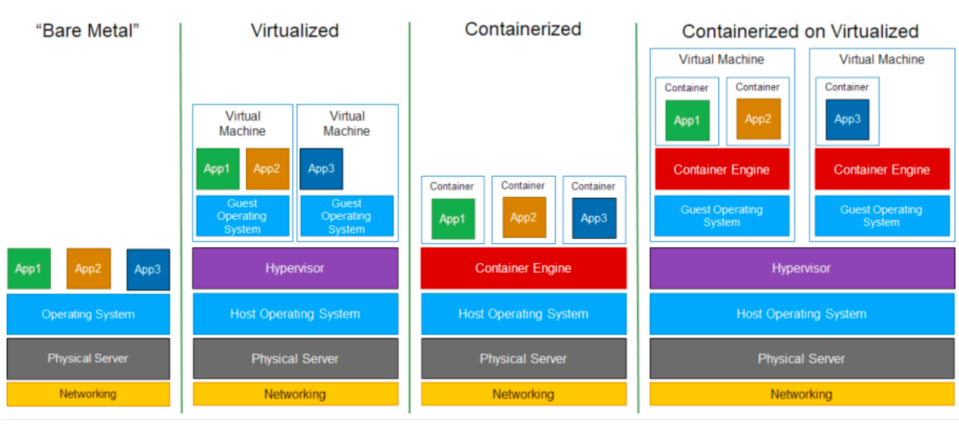
Question - Have you used orchestrator products in production? What are their pros & cons?

—

What are the differences between Virtualization (VMware) and Containerization (Docker)?

The diagram below illustrates the layered architecture of virtualization and containerization.

> Virtualization vs Containerization



"Virtualization is a technology that allows you to create multiple simulated environments or dedicated resources from a single, physical hardware system" [1].

"Containerization is the packaging together of software code with all its necessary components like libraries, frameworks, and other dependencies so that they are isolated in their own "container" [2].

The major differences are:

- In virtualization, the hypervisor creates an abstraction layer over hardware, so that multiple operating systems can run alongside each other. This technique is considered to be the first generation of cloud computing.

- Containerization is considered to be a lightweight version of virtualization, which virtualizes the operating system instead of hardware. Without the hypervisor, the containers enjoy faster resource provisioning. All the resources (including code, dependencies) that are

needed to run the application or microservice are packaged together, so that the applications can run anywhere.

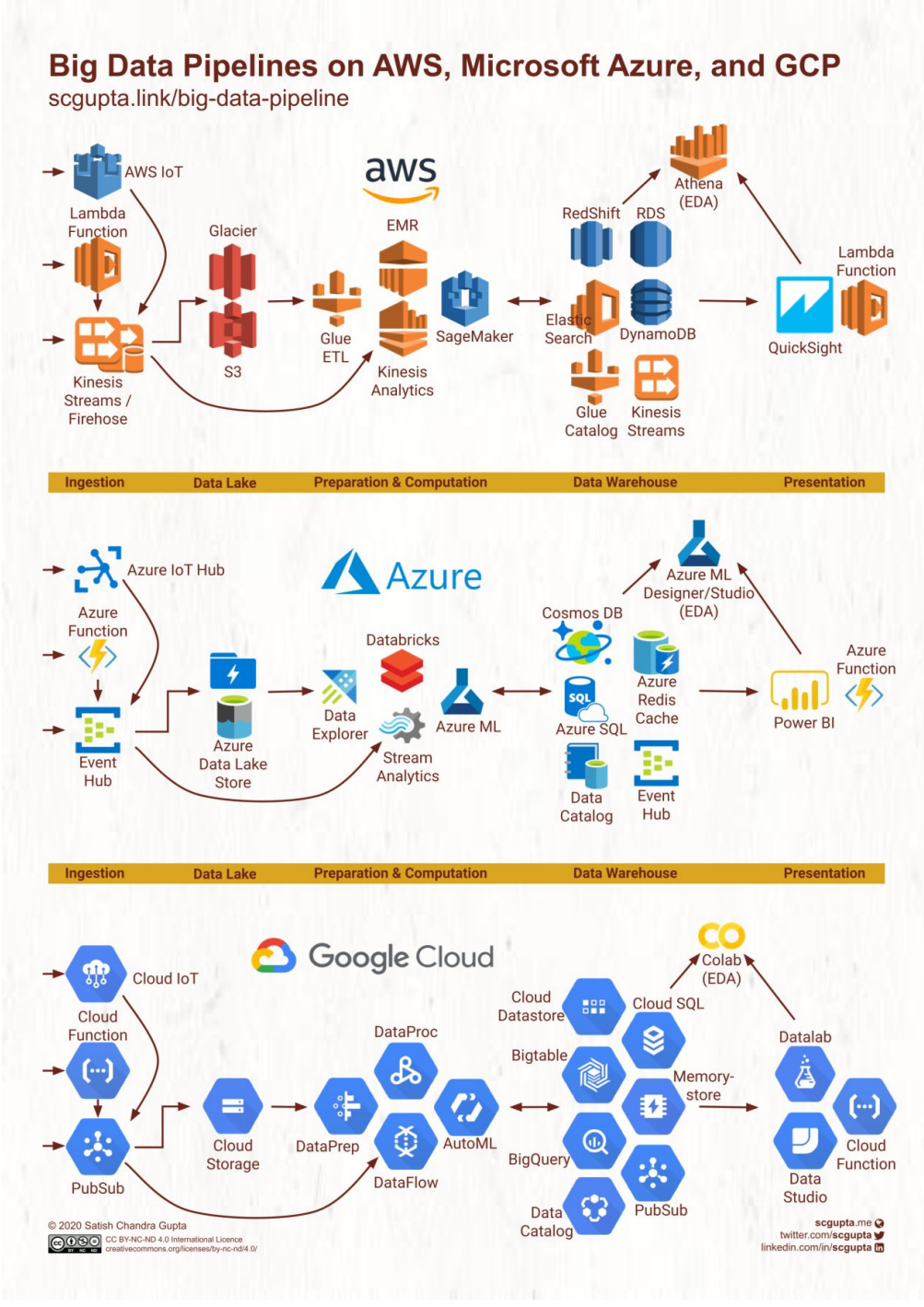
Question: how much performance differences have you observed in production between virtualization, containerization, and bare-metal?

Image Source: https://lnkd.in/gaPYcGTz

Sources:
[1] Understanding virtualization: https://lnkd.in/gtQY9gkx
[2] What is containerization?: https://lnkd.in/gm4Qv_x2

Which cloud provider should be used when building a big data solution?

The diagram below illustrates the detailed comparison of AWS, Google Cloud, and Microsoft Azure.



The common parts of the solutions:

1. Data ingestion of structured or unstructured data.
2. Raw data storage.
3. Data processing, including filtering, transformation, normalization, etc.
4. Data warehouse, including key-value storage, relational database, OLAP database, etc.
5. Presentation layer with dashboards and real-time notifications.

It is interesting to see different cloud vendors have different names for the same type of products.

For example, the first step and the last step both use the serverless product. The product is called “lambda” in AWS, and “function” in Azure and Google Cloud.

Question - which products have you used in production? What kind of application did you use it for?

Source: S.C. Gupta’s post

How to avoid crawling duplicate URLs at Google scale?

Option 1: Use a Set data structure to check if a URL already exists or not. Set is fast, but it is not space-efficient.

Option 2: Store URLs in a database and check if a new URL is in the database. This can work but the load to the database will be very high.

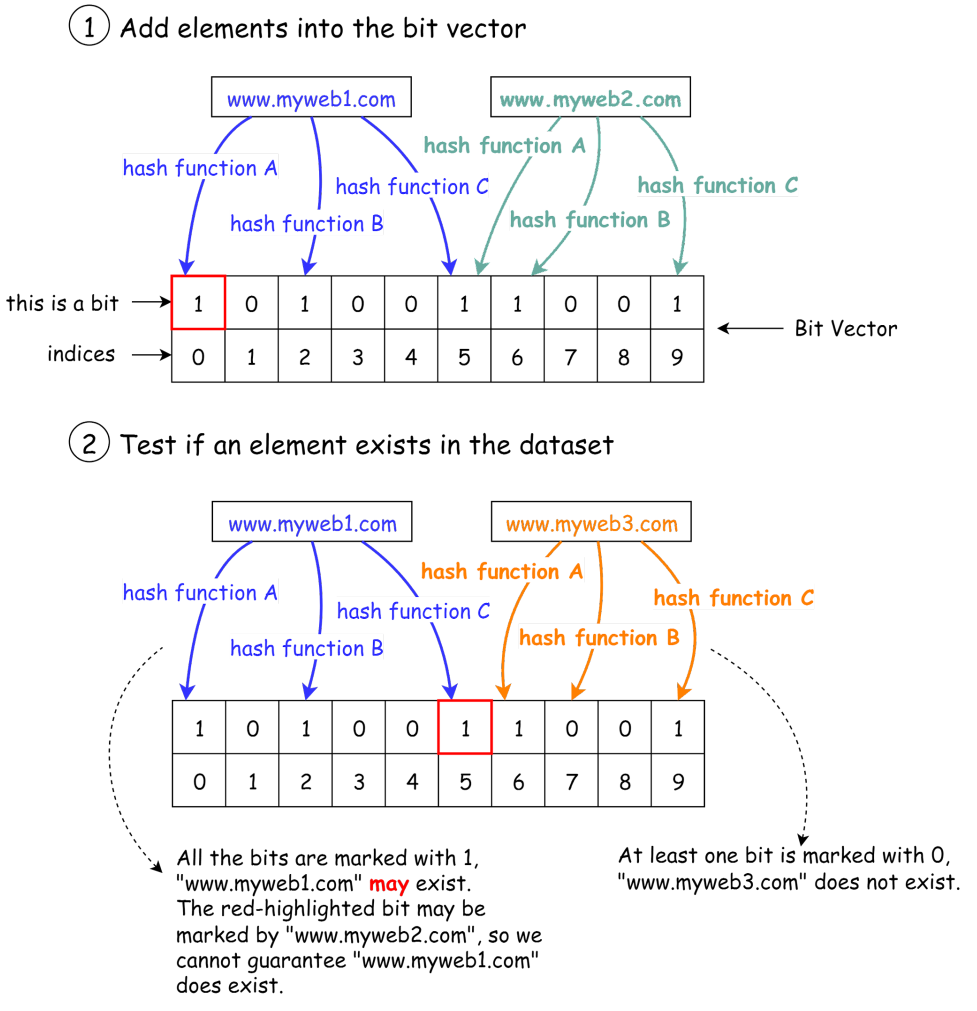
Option 3: **Bloom filter**. This option is preferred. Bloom filter was proposed by Burton Howard Bloom in 1970. It is a probabilistic data structure that is used to test whether an element is a member of a set.

- false: the element is definitely not in the set.
- true: the element is probably in the set.

False-positive matches are possible, but false negatives are not.

The diagram below illustrates how the Bloom filter works. The basic data structure for the Bloom filter is Bit Vector. Each bit represents a hashed value.

How to Dedupe Massive URLs



Step 1: To add an element to the bloom filter, we feed it to 3 different hash functions (A, B, and C) and set the bits at the resulting positions. Note that both “www.myweb1.com” and “www.myweb2.com” mark the same bit with 1 at index 5. False positives are possible because a bit might be set by another element.

Step 2: When testing the existence of a URL string, the same hash functions A, B, and C are applied to the URL string. If all three bits are

1, then the URL may exist in the dataset; if any of the bits is 0, then the URL definitely does not exist in the dataset.

Hash function choices are important. They must be uniformly distributed and fast. For example, RedisBloom and Apache Spark use murmur, and InfluxDB uses xxhash.

Question - In our example, we used three hash functions. How many hash functions should we use in reality? What are the trade-offs?

—

Why is a solid-state drive (SSD) fast?

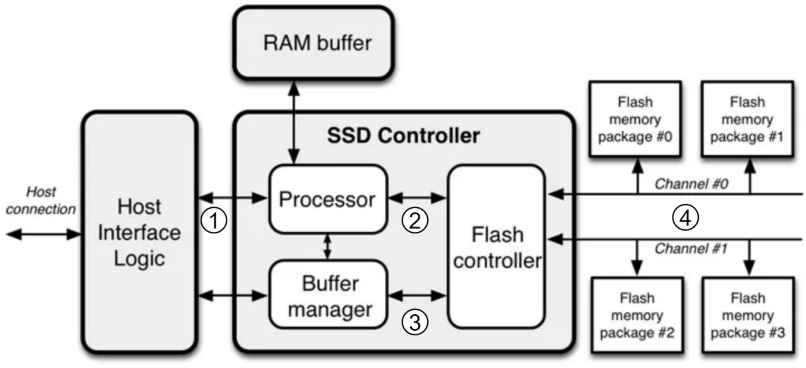
“A solid state drive reads up to 10 times faster and writes up to 20 times faster than a hard disk drive.” [1].

“An SSD is a flash-memory based data storage device. Bits are stored into cells, which are made of floating-gate transistors. SSDs are made entirely of electronic components, there are no moving or mechanical parts like in hard drives (HDD)” [2].

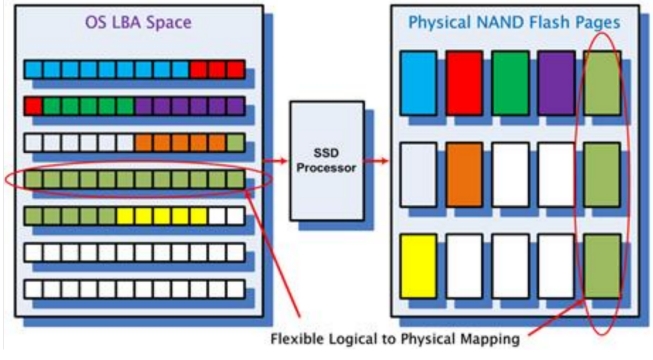
The diagram below illustrates the SSD architecture.

Why is SSD(Solid State Drive) Fast?

SSD Architecture



Mapping of Logical and Physical Pages



Step 1: “Commands come from the user through the host interface” [2]. The interface can be Serial ATA (SATA) or PCI Express (PCIe).
Step 2: “The processor in the SSD controller takes the commands and passes them to the flash controller” [2].
Step 3: “SSDs also have embedded RAM memory, generally for caching purposes and to store mapping information” [2].
Step 4: “The packages of NAND flash memory are organized in gangs, over multiple channels” [2].

The second diagram illustrates how the logical and physical pages are mapped, and why this architecture is fast.

SSD controller operates multiple FLASH particles in parallel, greatly improving the underlying bandwidth. When we need to write more than one page, the SSD controller can write them in parallel [3], whereas the HDD has a single head and it can only read from one head at a time.

Every time a HOST Page is written, the SSD controller finds a Physical Page to write the data and this mapping is recorded. With this mapping, the next time HOST reads a HOST Page, the SSD knows where to read the data from FLASH [3].

Question - What are the main differences between SSD and HDD?

If you are interested in the architecture, I recommend reading Coding for SSDs by Emmanuel Goossaert in reference [2].

Sources:
[1] SSD or HDD: Which Is Right for You?:
https://www.avg.com/en/signal/ssd-hdd-which-is-best
[2] Coding for SSDs:
https://codecapsule.com/2014/02/12/coding-for-ssds-part-1-introduction-and-table-of-contents/
[3] Overview of SSD Structure and Basic Working Principle:
https://www.elinfor.com/knowledge/overview-of-ssd-structure-and-basic-working-principle1-p-11203

Handling a large-scale outage

This is a true story about handling a large-scale outage written by Staff Engineers at Discord Sahn Lam.

About 10 years ago, I witnessed the most impactful UI bugs in my career.

It was 9PM on a Friday. I was on the team responsible for one of the largest social games at the time. It had about 30 million DAU. I just so happened to glance at the operational dashboard before shutting down for the night.

Every line on the dashboard was at zero.

At that very moment, I got a phone call from my boss. He said the entire game was down. Firefighting mode. Full on.

Everything had shut down. Every single instance on AWS was terminated. HA proxy instances, PHP web servers, MySQL databases, Memcache nodes, everything.

It took 50 people 10 hours to bring everything back up. It was quite a feat. That in itself is a story for another day.

We used a cloud management software vendor to manage our AWS deployment. This was before Infrastructure as Code was a thing. There was no Terraform. It was so early in cloud computing and we were so big that AWS required an advanced warning before we scaled up.

What had gone wrong? The software vendor had introduced a bug that week in their confirmation dialog flow. When terminating a subset of nodes in the UI, it would correctly show in the confirmation dialog box the list of nodes to be terminated, but under the hood, it terminated everything.

Shortly before 9PM that fateful evening, one of our poor SREs fulfilled our routine request and terminated an unused Memcache pool. I could only imagine the horror and the phone conversation that ensured.