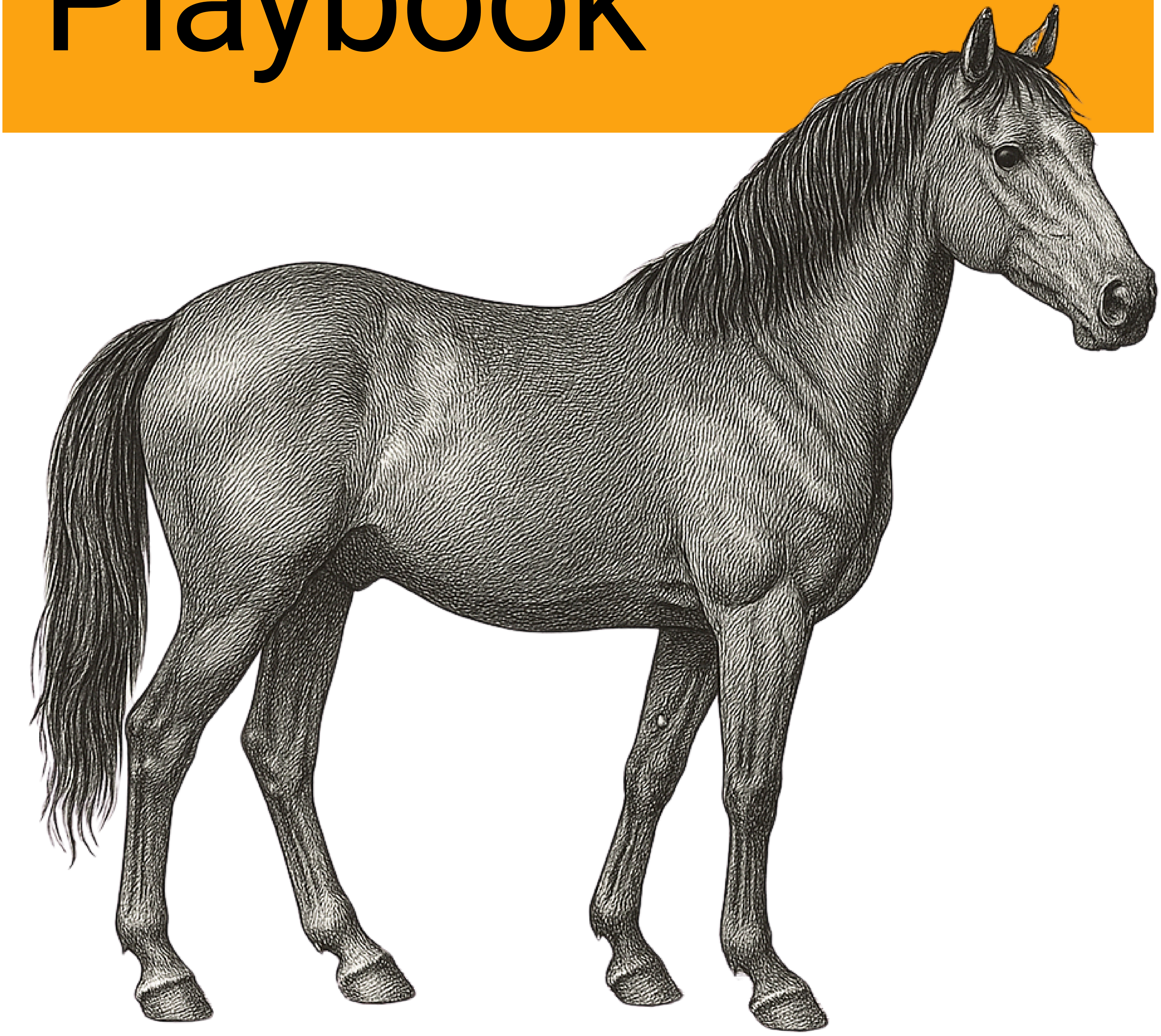
System Design Playbook



Neo Kim

4 How Database Stores Passwords Securely
5 How Micro Frontends Work
6 How Amazon S3 Achieves 99.999999999% Durability
7 How Amazon S3 Works
8 How Uber Computes ETA
9 How YouTube Supports 2.49 Billion Users With MySQL
10 How Amazon Lambda Works
11 How Apple Pay Works
12 How Figma Scaled Postgres to 4 Million Users
13 How Apple AirTag Works
14 How Stripe Prevents Double Payments
15 How Uber Finds Nearby Drivers
16 How Meta Achieves 99.99999999% Cache Consistency
17 How Tinder Scaled to 1.6 Billion Swipes a Day
18 How DNS Works
19 How JWT Works

How Database Stores Passwords Securely

by Neo Kim

1. The server transforms password using a hash function to create the fingerprint
2. The database stores only the fingerprint and not the password
3. The one-way hash function prevents retrieval of the password from a fingerprint
4. The system regenerates fingerprint whenever the user enters a password
5. The system compares regenerated fingerprint against the stored value to provide access
6. But it's possible to find password from a fingerprint using the rainbow table
7. A rainbow table is a map between pre-computed fingerprints and passwords
8. So the system adds salt, a unique random string, to the password to invalidate rainbow table values
9. The database stores the salt alongside the fingerprint
10. The server combines the password with saved salt to regenerate a fingerprint

1. They extend the microservices concept to the frontend
2. They slice a site's frontend into self-contained, domain-driven micro apps
3. They are technology agnostic & can be built with any framework: React or Angular
4. They're an architectural & organizational style based on domain-driven design
5. Their boundaries are set based on the business value
6. They allow autonomous teams owning a feature from the backend to the frontend
7. They support code reuse and visual consistency via shared libraries
8. They're often set up with the backend for frontend pattern
9. They communicate with each other via messages and events
10. They're good for big projects with technical and organizational maturity

How Amazon S3 Achieves 99.999999999% Durability

by Neo Kim

1. They store metadata and file content separately
2. They erasure code for data replication and low data loss
3. They measure the storage device health for durability
4. They keep free space in each storage device for high recovery throughput
5. They add checksums to find data corruption
6. They send checksum in HTTP trailer for data integrity checks at scale
7. They add checksums to erasure-coded data to find corrupted data sectors
8. They do bracketing on the erasure-coded data before returning success to the user
9. They monitor the disk failure rate to scale repair service
10. They keep data centers physically isolated with separate networking and power supply
11. They use versioning and backups to reduce user mistakes
12. They deploy changes only through a durability review

1. S3 is an object store to save unstructured data like log files
2. They use microservices architecture to build S3
3. They store metadata and file content separately for scale
4. They store metadata in a key-value database and cache it for high-availability
5. They store data in mechanical hard disks to save costs
6. They use ShardStore (LSM variant) to organize data on the hard disk efficiently
7. They replicate data across many disks and read parallel for high-throughput
8. They do erasure coding to reduce storage overhead

1. They represent the physical map as a graph.
2. They compute ETA by finding the shortest path in a directed weighted graph.
3. They don't use Dijkstra’s algorithm because it won't scale with O(n*logn) time complexity.
4. They partition the graph and then precompute the best path within each partition.
5. They reduce the time complexity from O(n^2) to O(n) by partitioning graph.
6. They populate the edge weights of the graph with traffic information.
7. They do map matching to find accurate ETA.
8. They use the Kalman filter and Viterbi algorithm for map matching.

1. They built an abstraction layer on top of MySQL and named it Vitess
2. They run a sidecar server to manage MySQL instances
3. They rewrite expensive SQL queries by adding the LIMIT clause
4. They cache popular data on the sidecar server to prevent the thundering herd problem
5. They use a proxy server to route queries to the right MySQL shard
6. They do connection pooling using a proxy server
7. They use a distributed key-value database to save shard metadata
8. They use an HTTP server to update the key-value database
9. They wrote Vitess in Go and open-sourced it

How Amazon Lambda Works

by Neo Kim

1. They use microservices to build the lambda service
2. They run lambda functions on a server called the worker
3. They use a microservice to set up and lease workers
4. They run lambda functions across many workers for scale
5. They use a microservice to track workers running a specific lambda function
6. They store worker metadata in a journal log for fault tolerance
7. They use lightweight virtual machines called microVMs for tenant-level isolation
8. They use a virtual machine manager called firecracker
9. They create microVM snapshots to reduce cold start latency by 90%
10. They lazy load the container images to reduce cold start latency
11. They find shared data between a container image's layers for optimal data delivery

How Apple Pay Works

by Neo Kim

1. They don't store credit card details on iPhone or Apple servers, instead send it to payment network
2. The payment network creates a unique number, DAN, to represent the credit card and iPhone
3. The iPhone stores DAN in the secure element, a specialized chip, for security
4. The card reader creates a transaction record when the iPhone communicates with it via NFC
5. The iPhone creates a cryptogram, single-use password, using DAN and transaction details
6. The iPhone sends only cryptogram and transaction details to the payment network
7. The payment network validates it by regenerating the cryptogram using its DAN copy
8. The payment network creates a new cryptogram using DAN, response code, cryptogram
9. The iPhone validates it by regenerating the new cryptogram and sends it to the card reader

How Figma Scaled Postgres to 4 Million Users

by Neo Kim

1. They store metadata such as file information & comments in Postgres
2. They scaled vertically by setting up a larger machine for extra CPU
3. They added database replicas to scale read traffic & increase fault tolerance
4. They added a caching layer to store frequently accessed data & reduce database load
5. They moved tables into separate databases based on domain to improve performance
6. They set up a connection pooler to avoid connection starvation & improve throughput
7. They moved high traffic columns into tables on separate databases to reduce workload
8. They split tables at row level and stored them across many databases for scalability

1. An AirTag contains a low-power CPU and a tiny amount of memory.
2. A public-private key pair gets created when a user adds an AirTag.
3. The AirTag doesn't use GPS or WiFi for communication; instead, it uses Bluetooth Low Energy.
4. They send AirTag's location, which is near its owner's iPhone, using Bluetooth or Ultra Wideband.
5. Yet Bluetooth and Ultra Wideband communication won't work if the owner's iPhone is far away from the AirTag.
6. So AirTag broadcasts its public key every 2 seconds over Bluetooth.
7. And someone else's iPhone, which is nearby, receives the broadcast signal.
8. The iPhone encrypts its location data and timestamp using the received public key.
9. The iPhone uploads the encrypted data to the Apple server over HTTPS.
10. The owner then decrypts the received location data using the private key.

How Stripe Prevents Double Payments

by Neo Kim

1. They use idempotent APIs to prevent double payment.
2. They use UUID as the idempotency key.
3. They send the idempotency key through HTTP headers.
4. They create a new idempotency key whenever the request payload changes.
5. They store the idempotency keys in a database on the server side.
6. They query the database with keys to check if a request was processed earlier.
7. They process a request only if it's new and then store its idempotency key in the database.
8. They roll back a transaction using the ACID database on a server error.
9. They remove the idempotency keys from the database after 24 hours for reuse.
10. They use exponential back-off with jitter to avoid the thundering herd problem.

How Uber
Finds Nearby Drivers

by Neo Kim

1. They created a hexagonal-shaped hierarchical geospatial index called H3
2. They find nearby drivers by indexing locations using the H3 library
3. They divide Earth's surface into cells on a flat grid & use the rider's nearby cells to find drivers
4. They use hexagonal cells because it's easier to measure the distance between cells
5. They subdivide each hexagon into 7 hexagons to support different data resolutions
6. They do bitwise operations to switch between data resolutions in constant time
7. They index an area as small as 1 square meter & use the consistent hash ring for scalability

1. They use a distributed cache to scale reads.
2. The database & cache are aware of each other, while they use a simple data model.
3. But race conditions with data events might cause cache inconsistency.
4. So they use an observability solution to improve cache consistency.
5. They set up a separate service to monitor cache inconsistency, called Polaris.
6. Polaris receives cache invalidation events & queries cache servers to check cache inconsistency.
7. Polaris queries the database only at intervals of 1, 5, or 10 minutes for efficiency.
8. They use a separate invalidation event stream between the client & Polaris.
9. They set up a tracing library & embedded it on each cache server.
10. They log only data changes that occur during the race condition time window.
11. So Polaris finds cache inconsistency quickly, while tracing finds why it occurred.

1. They use Google S2 library to index locations & find nearby users.
2. They store user information in a key-value database.
3. They push out changes to a table into different places using change data capture pattern.
4. They run 500 microservices using a service mesh.
5. They provide public APIs through the API Gateway.
6. They put swipes into a data stream & check for profile matches using a separate server.
7. They notify users if there's a match in real-time via websockets.
8. They store profiles disliked by a person in an object store to improve recommendations.
9. Hot shard might occur from time zone differences. So they put many shards on the same server to prevent it.
10. They use a caching layer to scale reads and avoid the hot shard problem for reads.
11. They rate limit requests to avoid the hot shard problem for writes.
12. They do exponential back-off with jitter on failures for fault tolerance.

1. **User** - The browser checks its cache and the operating system's cache for the IP address.
2. **DNS Client** - The browser then queries the resolver server to find the correct DNS server for lookup. The resolver server also checks its cache.
3. **Root Server** - The resolver server then queries the root server to find the right TLD server, such as com or org.
4. **TLD Server** - The top-level domain (TLD) server routes the query to the correct authoritative name server.
5. **Authoritative Name Server** - It contains the IP address and responds to the resolver server.
6. **Web Server**- The resolver server returns the IP address to the browser, which then makes a direct call to the site's web server.

How JWT Works

by Neo Kim

1.Workflow

- The server creates a JWT on user authentication.
- The server gives the JWT to the client after signing it.
- The client includes JWT in each request.
- The server allows a request only if the JWT's signature is valid.

2. Authorization

- The server doesn't store session information.
- Instead JWT includes the necessary authorization details.
- The server checks the JWT on each request.

3. Structure

- JWT doesn’t look like a JSON object, but a set of characters.
- It has 3 parts: header, payload, and signature.
- The server creates the signature by running an algorithm on the header, payload, and a secret key. Only the server has access to the secret key.
- The server verifies the payload by recomputing the signature on the received JWT.

4. Security

- A JWT might get stolen. So JWT should be sent over HTTPS for security.
- Also minimum roles and an expiry time should be set to reduce the damage.
- A denial list is maintained on the server to reject stolen JWTs.

This ebook is for educational use only.

Please don’t sell it, repackage it, or copy it without permission.

If you write about system design, be original—don’t copy from this.

Thanks for respecting the work.

newsletter.systemdesign.one