Welcome!

WWCode San Francisco - Backend Study Group

March 8, 2023

- We'll start in a moment :)
- We are **RECORDING** tonight's event
- We may plan to take screenshots for social media
- If you are comfortable, turn the video ON. If you want to be anonymous, then turn the video off
- We'll introduce the hosts & make some time for Q&A at the end of the presentation
- Feel free to take notes
- Online event best practices:
    - Don't multitask. Distractions reduce your ability to remember concepts
    - Mute yourself when you aren't talking
    - We want the session to be interactive
    - Use the 'Raise Hand' feature to ask questions
- **By attending our events, you agree to comply with our Code of Conduct**

WOMEN WHO CODE /san-francisco

Introduction & Agenda

- Welcome from WWCode!
- Our mission: Empower diverse women to excel in technology careers
- Our vision: A tech industry where diverse women and historically excluded people thrive at any level
- Backend Study Group: Learn and discuss backend engineering concepts

Prachi Shah
Instructor
Senior Software Engineer, Unity
Director, WWCode SF

Matt Hofstadt
Host
Startup Founder
Volunteer, WWCode SF

System Design - Series Part 2 of 3:
- What are the design considerations?
- How to scale systems?
- How to manage data?
- Q & A

Part 3: Interview questions (March 16th)

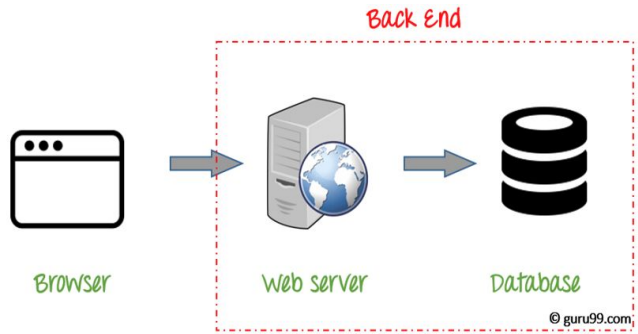
Disclaimer:
- Sessions can be heavy!
- Lots of acronyms
- Instructors don't know everything

WOMEN WHO CODE /san-francisco

Backend Engineering

- Design, build and maintain server-side web applications

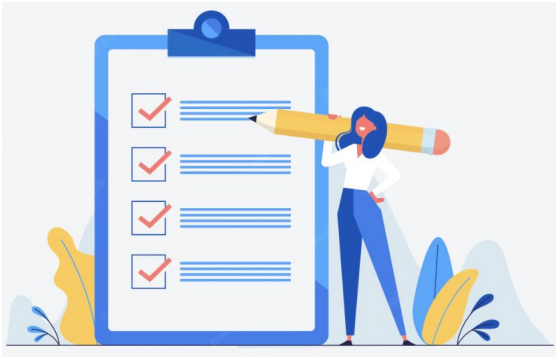
- Concepts: Client-server architecture, networking, APIs, web fundamentals, microservices, databases, security, operating systems, etc.



- Tech Stack: Java, PHP, .NET, C#, Ruby, Python, REST, AWS, Node, SQL, NoSQL, etc.

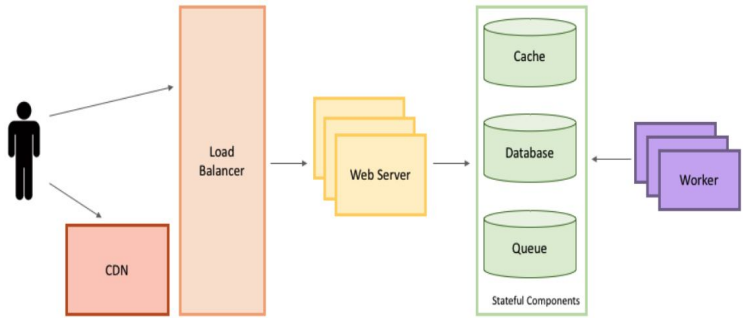
System Design

- Solve a problem or build a product
- Defining the architecture, modules, interfaces and data flow
- Architecture: Defines behavior and view of a system
- Modules: Each module corresponds to a task
- Interfaces: Defines the communication between modules
- Data flow: Flow of data and information between systems
- Define the input, output, business rules, data schema



Design Considerations

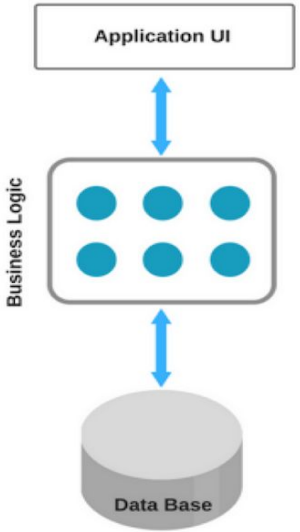
- Scaling: Change in performance as per changing application demands
- Availability: System uptime and downtime
- Reliability: System performs the tasks as expected
- Robustness: Functional when errors or disturbances
- Load Balancing: Network traffic distribution across servers
- Caching: Data storage layer
- Data Partitioning: Distribute data across systems to improve querying performance
- SQL vs. NoSQL: Relational vs. Non-relational data model
- Performance: Glitch-free* and fast
- Extensibility: Future growth
- Error Handling and Security: UX and secure data

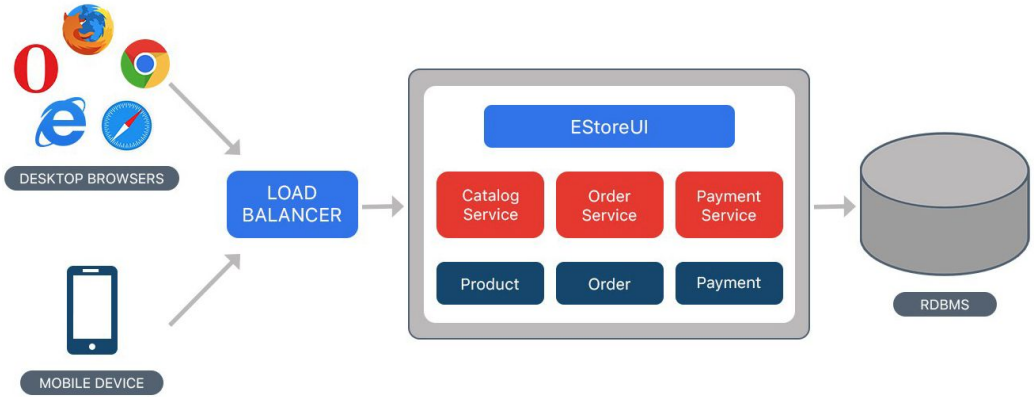


** requirements may vary*

Monolith Architecture

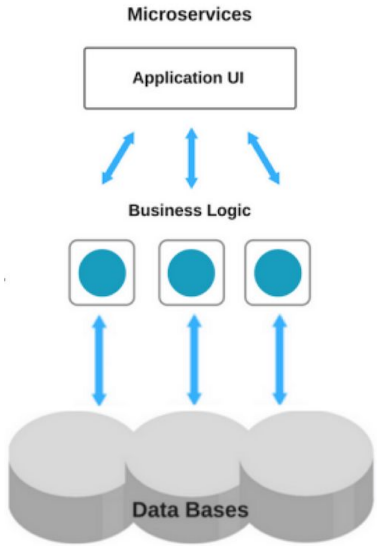
- Application stack resides one server
- Business logic, APIs, UI (images, files) and database are bundled
- Code and database on single server
- Scale up: More CPU, memory and external disk space
- Single point of failure, longer downtimes
- Example: E-commerce application

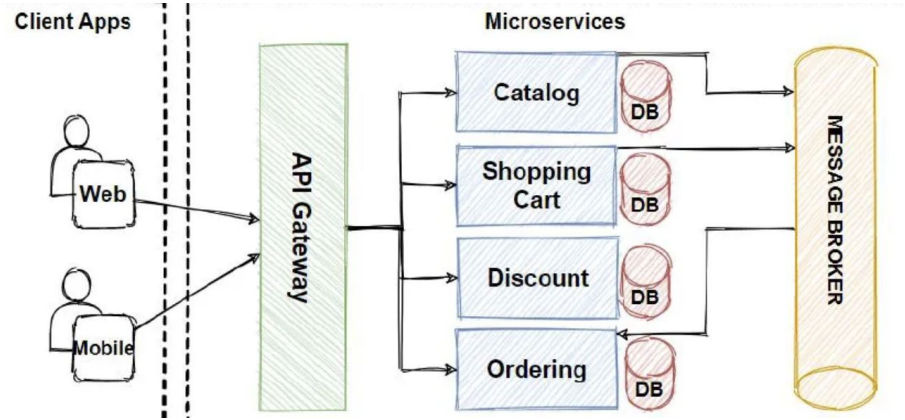




Microservice Architecture

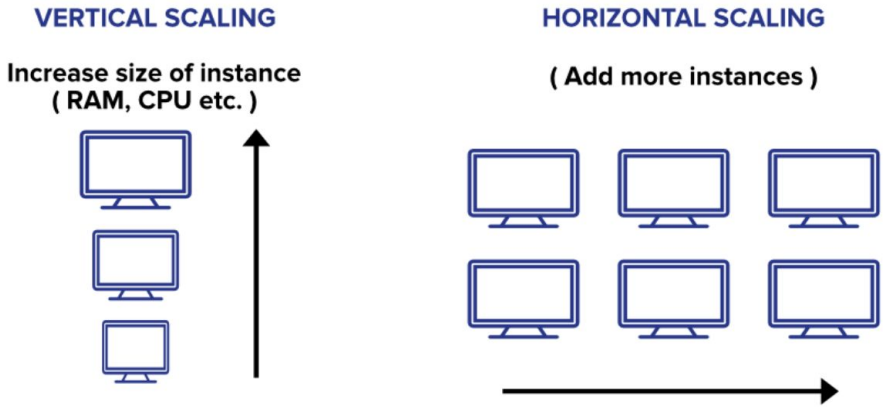
- Application stack resides on >1 servers
- Business logic, APIs, UI (images, files) and database are distributed
- Scale up: More servers and databases
- Higher availability, lower application downtime, maintainability
- Example: E-commerce application





Scaling

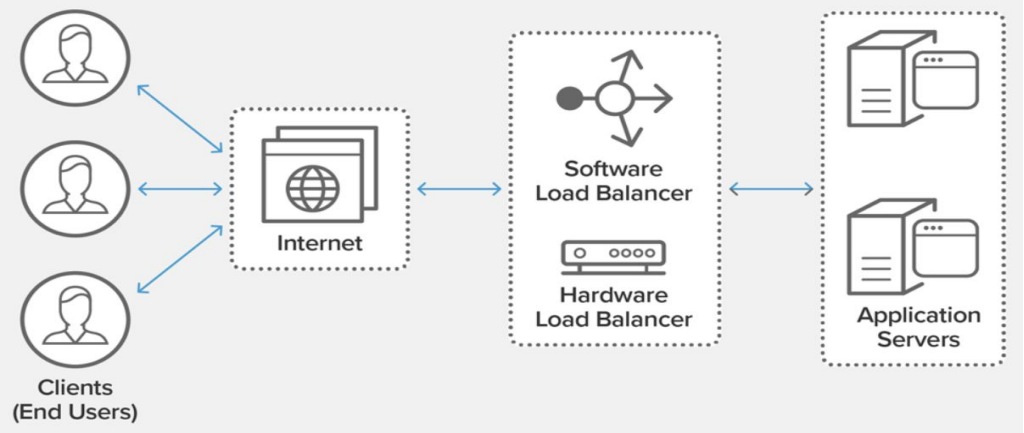
- Load Balancing
- Caching
- Data Partitioning
- SQL vs. NoSQL
- ACID and CAP Theorem



- Vertical:
  - Add more resources like CPU, memory, disk to existing server
  - Limited capability, single point of failure, run out of resources eventually
- Horizontal:
  - Add more devices/servers with application code
  - Distributed system with load balancing of incoming traffic
- Hybrid: Both vertical and horizontal scaling

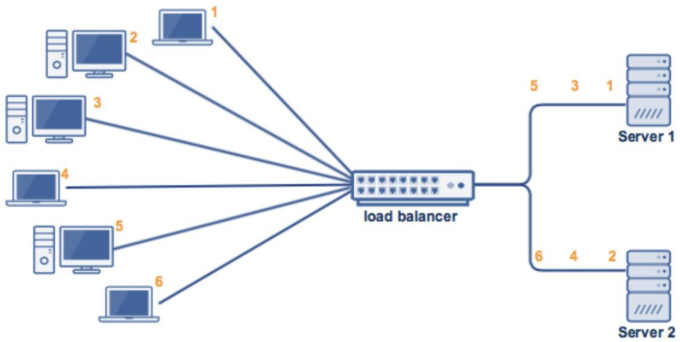
Load Balancing

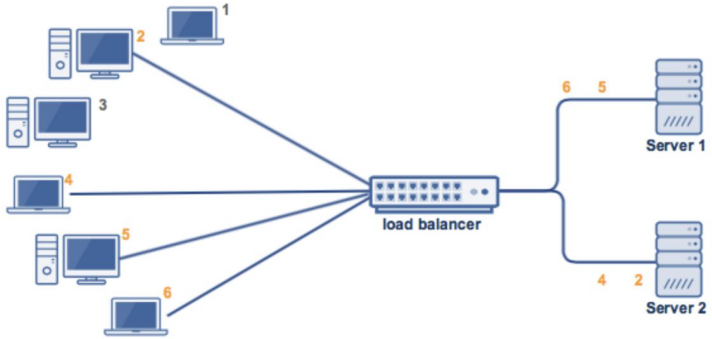
- Assign incoming client requests to distributed resources
- Prevents overloading resources (servers, database)
- No single point of failure
- Distribution across:
    - Hardware: High performance, hard to configure, expensive
    - Software: Installed on server/virtual machine, easy to configure, inexpensive
- Example: NGINX Plus



Traffic Routing Algorithms

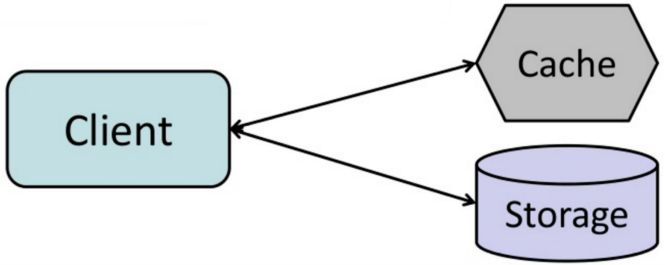
- Round Robin: Sequentially rotating allocation, same number of requests/server
- Random: Uses a random number generator to select a server
- Least loaded: Requests to server with the lowest number of active connections
- Application-specific: Allocation per application needs
- Example: Round Robin; Least loaded





Caching

- Optimize system for read operations (ideally)
- Reduces load on servers and databases
- Optimizes distributed traffic management
- Client-side and CDN:
    - Caching in the browser (page data) and Conter
- Web server:
    - Reverse Proxy: Server to redirect requests to web & application servers
    - Cache requests, static content (html pages, images, media)
- Database:
    - Database cache for read-heavy requests
- Application:
    - Memcached or Redis (key-value store) cache for data storage
    - Faster access to frequently queried data

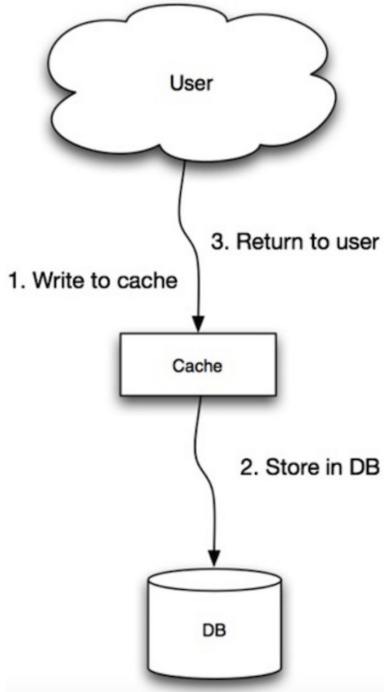


Caching Strategies

- Cache aside: Read data from the database on cache miss, update the cache, time consuming, set time-to-live (ttl)
- Write through: Cache is the primary datastore, cache updates the database
- Write behind (Write back): Application updates cache, asynchronously update the database, inconsistent data with system failures

- Distributed Caching: Data stored across caches, CDN, databases

- Cache Invalidation:
    - FIFO: First in first out, based on time
    - LIFO: Lst in first out, based on time
    - LRU: Least recently used (old data)
    - LFU: Least frequently used (fewer reads)



Data Partitioning

- Splitting data across multiple tables and datastores
- Improve maintainability, performance, availability, load balancing and cost effectiveness
- Horizontal partitioning (Sharding):
    - Data divided into partitions (or multiple smaller tables) that are accessed separately
    - Routing algorithm decides which partition (shard) to store the data
    - Data range-based: Based on the range of data. Example: Zip codes, location
    - All partitions have the same data schema
    - Easily add more machines for better load balancing, shorten response time (query)
    - Uneven distribution, application code needs to collate data from different shard

- Vertical partitioning:
    - Application feature-wise distribution of data
    - Easy to implement, low impact on the application
    - All partitions may have different data schema

Data Partitioning

Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston
3	Carrie	Conway	Chicago
4	David	Doe	Denver

Vertical Shards

VS1

CUSTOMER ID	FIRST NAME	LAST NAME
1	Alice	Anderson
2	Bob	Best
3	Carrie	Conway
4	David	Doe

VS2

CUSTOMER ID	CITY
1	Austin
2	Boston
3	Chicago
4	Denver

Horizontal Shards

HS1

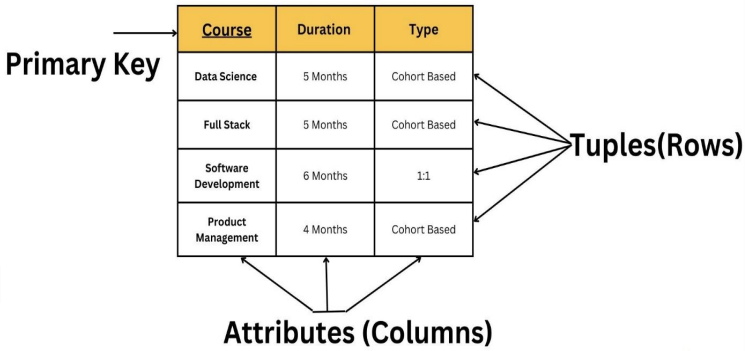
CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston

HS2

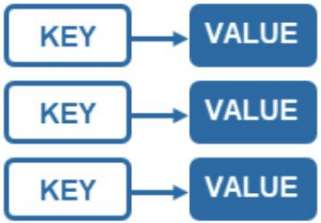
CUSTOMER ID	FIRST NAME	LAST NAME	CITY
3	Carrie	Conway	Chicago
4	David	Doe	Denver

Data Schema

- Data can be structured (SQL table schema), semi-structured (JSON, XML, etc.), and unstructured (Blob, Images)
- Common database categories include:
- Relational: Tables (row, column) [SQL, Oracle]
- Key-Value: ID, data [Redis]
- Column Wide: Flexible columns [Cassandra]
- Graph: Relationship between objects [OrientDB]
- Document: Query JSON-like document [MongoDB]
- Blob: Binary data (images, audio, etc.)



Column based



Key-value

Graph

Document

SQL

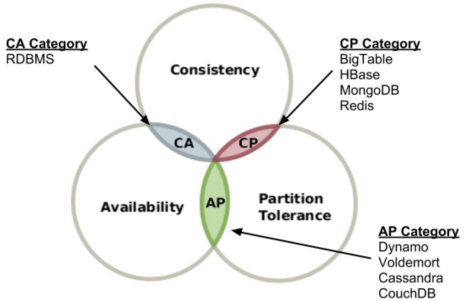
- Relational Database Management System
- Data is stored in tabular format organized as rows and columns
- Tables have relationship between them and the data model is rigid
- Pros:
    - No extensive coding is needed
    - Almost everyone is familiar with basic SQL commands
    - Supports ACID properties
    - Supports JOIN between tables
    - High demand in the industry. Large user community
- Cons:
    - Rigid data schema. No way to store unstructured data
    - Querying slows as data grows
    - Hard to scale horizontally. Can only be scaled vertically
    - Hardware maintenance is expensive

NoSQL

- Data is stored in various formats: JSON document, key-value pairs, wide column, graphs (nodes and edges), etc.
- Data has no relationship between them
- Cannot JOIN between data
- Suitable for huge volume of data
- Pros:
    - Scale out architecture. Just keep adding more nodes. No single point of failure
    - Less hardware management. Usually cheap hardware is used
    - Flexibility of storing unstructured data, data model is flexible
    - Can store massive amounts of data and is typically used in the big data world
    - Very agile and a great flexibility and adaptability, and faster querying
- Cons:
    - No ACID properties support only eventual consistency of data (Speed & availability)
    - Learning curve is stiff for new developers, smaller developer community

ACID and CAP Theorem

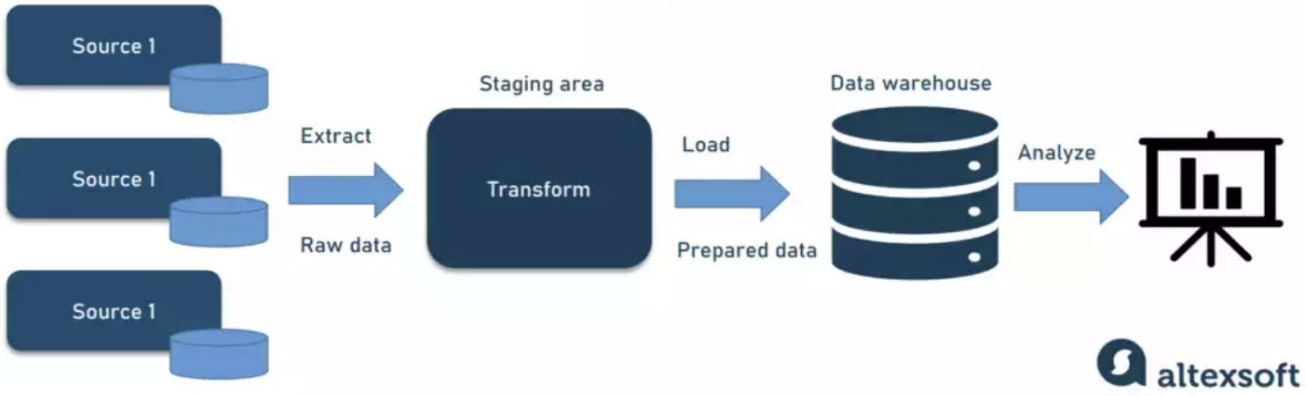
ACID:
• Atomicity: All changes to data and transactions are executed completely and as a single operation. If that isn't possible, none of the changes are performed. It's all or nothing
• Consistency: The data must be valid and consistent at the start and end of a transaction
• Isolation: Multiple transactions can occur without stepping on each other
• Durability: When a transaction is completed, its associated data is permanent and cannot be changed



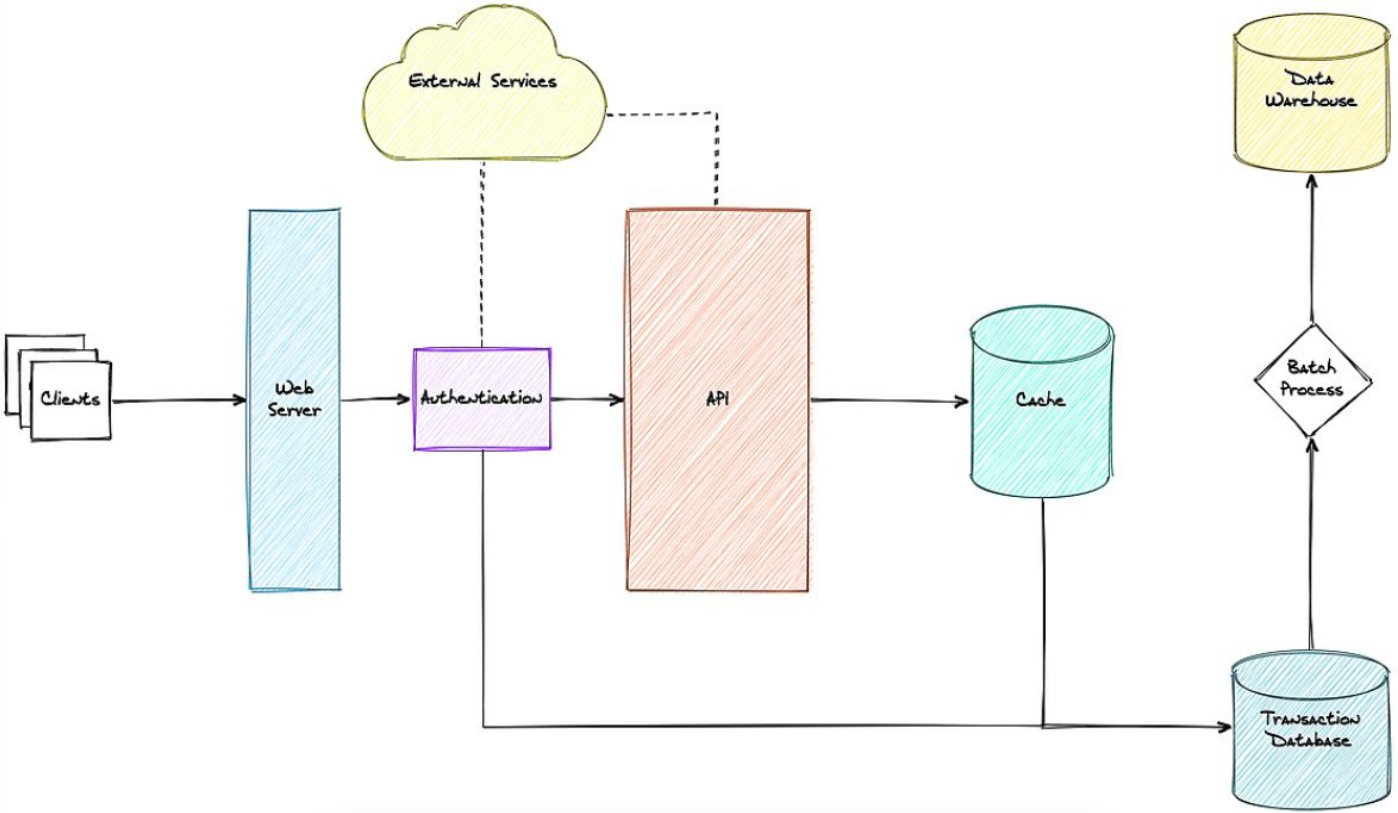
CAP Theorem:
• Consistency: All clients see the same data at the same time
• Availability: System continues to operate even with node/device failures
• Partition-Tolerance: System continues to operate even with network failures
• In the event of a network failure on a distributed database, we only get either consistency or availability - but not both

Data Pipeline

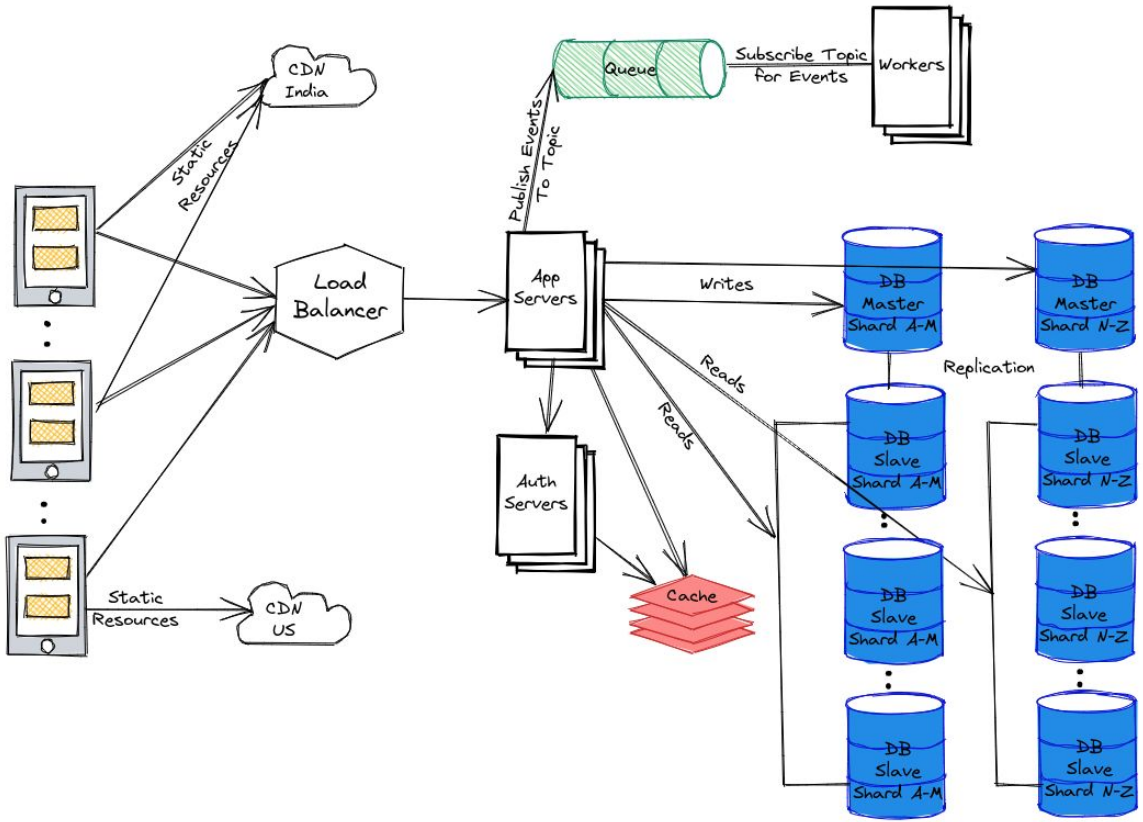
- Raw data is ingested from various data sources and ported to datastore (data lake or data warehouse) for analysis
- Data processing occurs before storage to data repository, is isolated and asynchronous
- Data ingestion can happen for structured or unstructured data
- Data processing includes filtering, transformation, normalization (remove redundancies)
- Data visualization: Presentation layer with dashboards, reporting & real-time notifications
- Tools, Informatica Power Center (ETL), Snowflake, BigQuery, MS Azure, Kafka (real-time)



Design



Design



Backend Study Group

References:

- System Design Primer

Backend Study Group:

- Presentations on GitHub and session recordings available on WWCode YouTube channel
- March 16th, 2023: Part 3 - System Design - Interview questions
- March 23rd, 2023: Introduction to GPT3
- April 6th, 2023: SQL Queries 101

Women Who Code:

- Technical Tracks and Digital Events for more events
- Join the Digital mailing list for updates about WWCode
- Contacts us at: contact@womenwhocode.com
- Join our Slack workspace and join *#backend-study-group*!

You can unmute and talk or use the chat

WOMEN WHO CODE /san-francisco