acing$_{thesystemdesigninterviewmanningzhiyongtan}$

[2025-03-14 Fri 06:50]

Contents

1 Part 1 Theory **1**
1.1 Chapter 1 A walkthrough 1

Study notes from : Acing the system design Interview by Zhiyong Tan Manning Press 2024 ISBN : 9781633439108

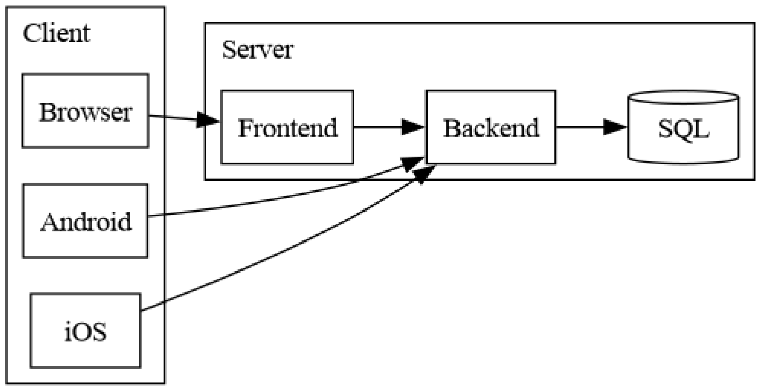
1 Part 1 Theory

1.1 Chapter 1 A walkthrough

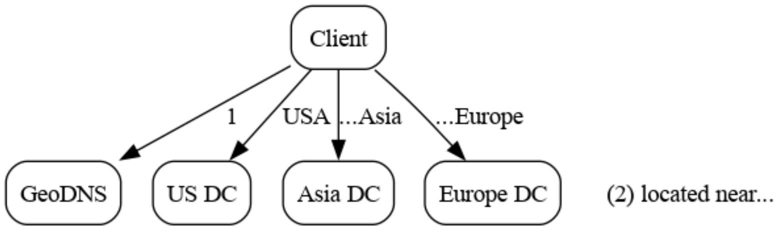
1. A bagle shop

gzip	brotli
- faster in compression and decompression	- better compression ratios
- older and better used for binary files	- newer and better for text

   (a) So we use brotli to compress the JS bundle the user downloads for our ReactJS webapp
   (b) we also have ios and android app



The service now has millions of customers and we had set up **stateless** service so we could handle them all in different locations and scale up the backend without any hiccups.



(a) Adding a caching service We now add redis in memory database as the cache.

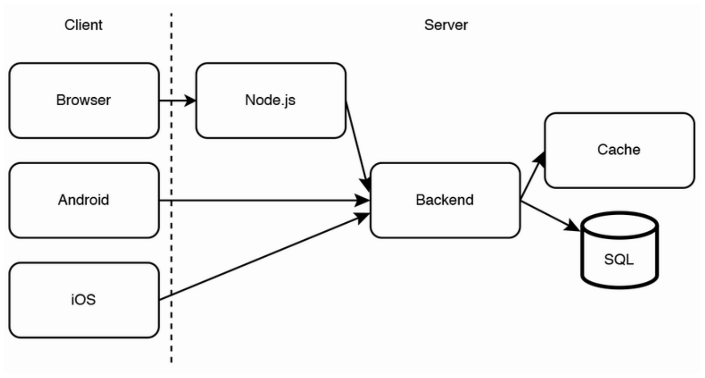


Figure 1.3 Adding a cache to our service. Certain backend endpoints with heavy traffic can be cached. The backend will request data from the database on a cache miss or for SQL databases/tables that were not cached.

(b) CDN We remove static files from source code repo to third party CDN service and serve them from CDN urls.

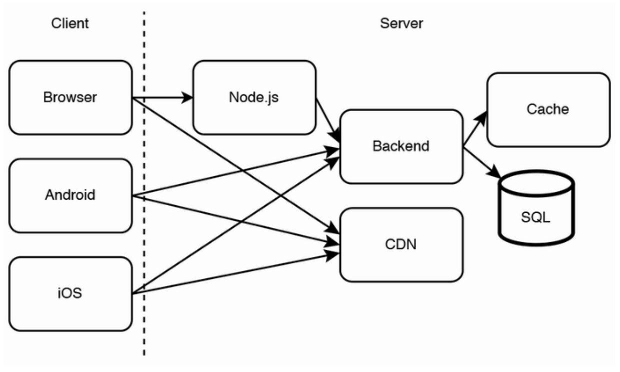


Figure 1.4 Adding a CDN to our service. Clients can obtain CDN addresses from the backend, or certain CDN addresses can be hardcoded in the clients or Node.js service.

examples : cloudflare, rackspace, aws cloudfront

(c) Brief on horizontal scalibility, cluster management and CI CD Our backends are idempotent meaning they respond with the same data when queried the same regardless of other factors. Idempotent systems are thus horizontally scalable.

   i. CI CD and IAC Infrastructure as code.IAC is the process of managing and provisioning computer data centers through machine readable config files rather than physical hardware config, or interactive configuration.
      - Docker or podman to containerize
      - docker swarm or kubernetes for clustering and load balancing
      - Ansible or Terraform for configuration management
        - Terraform DSL (Domain specific language) is compatible with multiple cloud providers and can provision infrastructure without vendor lock in.

   ii. gradual rollout and rollback rollout changes slowly like 1% , 3%, 5% and so on to 100% if no problems are detected. Roll back the change if any of these occur:
      A. Increased user churn because of performance or dislike of updates.
      B. Crash
      C. memory leaks

   D. increased latency
   E. increased resource usage : cpu, ram etc

iii. Experimentation This deals with UX change and consumer reaction to those changes rather than performance that is associated with rollout and rollback. Common approaches:
   A. A/B testing
   B. Multivariate testing
   C. Multi armed bandit

   These allow for short feedback cycles and faster iterations.

2. Functional Partitioning and centralization of cross cutting concerns Functional Partitioning : seperation of functions into different services or hosts.

   (a) Shared services

   - We create a shared search service with elasticsearch
   - On top of caching, CDN, horizontal scaling we add functional partitioning

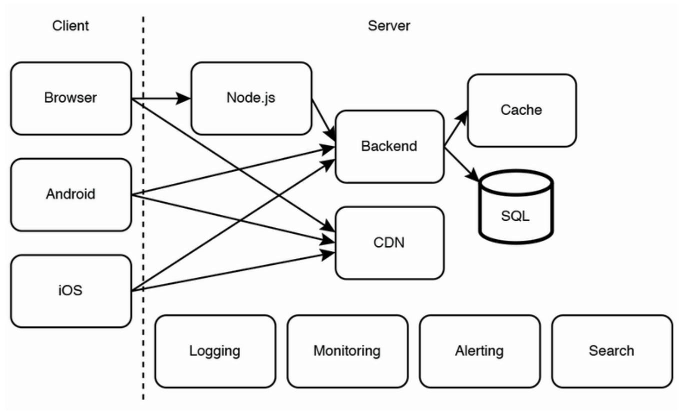


Figure 1.6 Functional partitioning. Adding shared services.

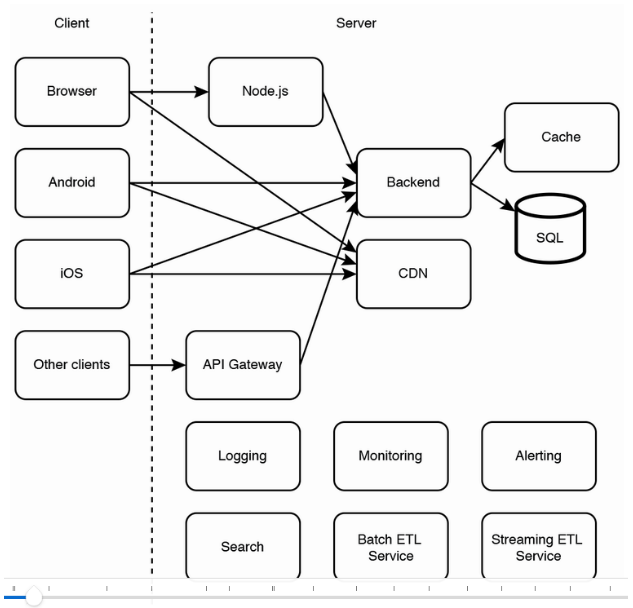
   i. Adding shared services
      - added logging service of log based message broker
      - Elastic Stack : Elasticsearch, Logstach, Kibana, Beats
      - use distributed logging system like `zipkin` or `jaegar` to trace request traversal

- monitoring and alerting system for customer complaints and customer service teams

(b) **TODO** Ways to centralize cross cutting concerns

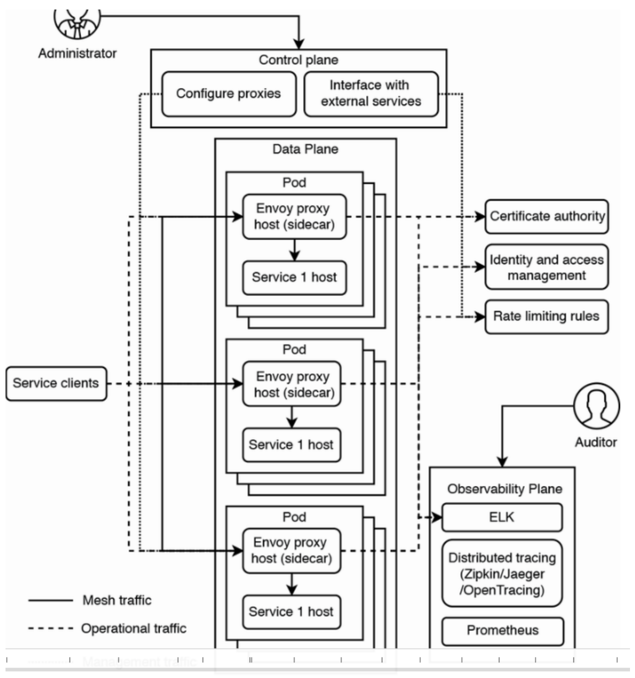
i. API gateway We expose our services to other companies and external developers. We implement

- Authorization and authentication
- rate limiting
- billing
- logging monitoring and alerting on request level
- analytics



ii. Service mesh / sidecar pattern This solves the awkward architecture, latency, complexity of API gateway.

- Use service mesh framework like Istio.



- Sidecarless Servicemesh In early stages of development and the cutting edge.

iii. CQRS : Command Query Responsibility Segregation

- microservices pattern where read and write are functionally partitioned into seperate services.
- low latency, scalable, easier to maintain and use
- seperate table or databases for reading from and writing into and they are synced periodically as required.

iv. Decorator pattern

v. Aspect Oriented Programming

Look at these two topics yourself

(c) Batch Streaming ETL

- Use event streaming systems like Kafka or AWS Kinesis and batch ETL tools like Airflow for batch jobs.
- Preprocess most requested things and cache them to serve multiple users rather than processing for each user

- delay non essential things like user statistics and logs and place them in a queue to be executed when resources are not under heavy load
- Apache Flink can be used of we want to to streaming rather than batch jobs.

(d) Other common Services

   i. Customer / external credential management
   ii. Storage services
   iii. Asynchronous processing
   iv. privacy and compliance teams
   v. notebook and data analytics / machine learning services
   vi. fraud detection
   vii. internal search and subproblems

This author seems biased towards Cloud perhaps because of his pedigree and history with this companies. I disregard this advice and instead choose bare metal for data security, cost efficiency and independence from vendor lock in. Shared servers can easily simulate serverless systems now, there is no reason to bleed money to cloud providers. For more on this see fn project, cloudflare workered, knative. knative is CNCF incubation project