iMBP – Architecture & Design

Charles Cai

OBJECTIVE

- Build a platform in distributed microserives architecture
- Build an application that is dynamically configurable
- Build a platform that is expandable and scalable
- Build an application with asynchronous requests and fault-tolerance

iMBP PLATFORM COMPONENTS

- API gateway
- Domain & Microservice apps
- Service Discovery & Configuration system
- Logging system
- Monitoring system
- Data storage units

EMAIL

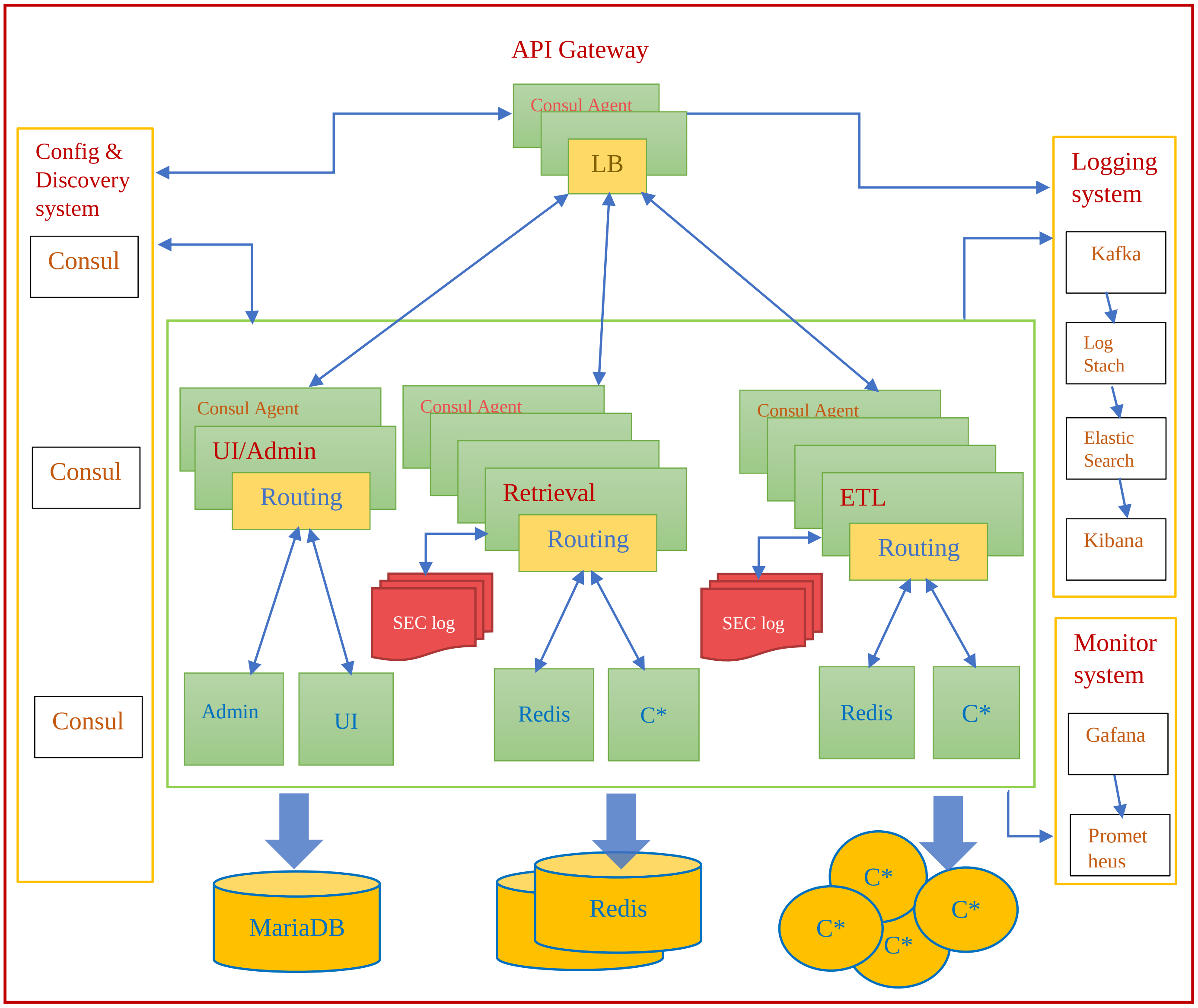
TWITTER HANDLE

TELEPHONE

LINKEDIN URL

iMBP – Architecture & Design

Architecture Diagram



EMAIL TWITTER HANDLE TELEPHONE LINKEDIN URL

API Gateway

- Direct client requests to corresponding domains, projects and microservices, the primary domains for iMBP has ETL, Retrieval(RT), UI(may need to break down to be more specific), Admin and more as the project moves on. Each URL must include domain (ETL, RT) and project (AOI, SMT) to ensure distinguishes of the requests' objective
- Load balance to underline application servers dynamically based on service discovery and availability
- Pan out and manage aggregated service calls if it's necessary
- Manage authentication and authorization requests

Domain & Microservice

- Each domain and project should have its own dedicated microservice server group to accommodate or tailor to its own specific needs or requirements. In following section, will focus **only** on ETL domain process
- ETL
    - API gateway is application's first entry point of all domains, each ETL request is load balanced through round robin to asynchronously send to downstream ETL servers. Upon receiving the request, the ETL server checks the cached metadata, then route to corresponding handler to save the data asynchronously to its designated storage unit

- Each server should have approximated same load which requires overall requests is proximate same payload on average is the application's first line of defense to prevent server hotspot/overload
- Any image has size over the 1M bytes should not send to server without compression at origin to avoid network bandwidth and storage space costs. If compressed data still remain over M bytes footprint, a separated url and servers will be dedicated to process the data that is our second line of defense to avoid server hotspot. (If iMBP needs to do compression, a POC needs to search best compression technology. If none found, will think of other way).
- Any request passing over predefined process time will be cut off to avoid resource occupation and server hanging that is our third line of defense against server hotspot. The payload of short circuited request will be committed to SEC log (**S**hort circuit and **E**xception **C**ommit log) in disk for future process, second copy will send to logging system for backup or inspection, same to any exception generated or failure during ETL process, will utilize two-copy paradigm method. A configurable SEC log size threshold should be setup to alert administrator for inspection and an action follows
- Actions on SEC log will be handled through Admin UI. Admin and Etl servers will provide several APIs to accommodate select, update, purge (single or all) and replay SEC log records. The deletion also includes second copy in ElasticSearch
- Batch request (need more information from client)

- RT
- Admin
- UI

Service discovery & Configuration

- Consul is our primary service discovery, configuration and metadata management system. Consul requires minimum three servers cluster in production, each ETL server has pre-installed consul agent. Whenever an ETL app starts, the consul agent will send a service call to consul cluster server for registration. Join or remove a service is transparent to API Gateway's load balancer
- Configuration parameters defined in the application also being loaded from consul server kv store during startup, any configuration change in consul cluster server will reflect to the application immediately without restart of app servers, the flexibility and dynamic configurability of consul greatly extends the control of app behave and flow
- Consul also provides api for configuration and service read, update and delete. The metadata will be rendered to UI from mongo, managed through consul kv store and eventually pushed to all ETL app serves
- Consul comes health checking of each app server with configurable interval

Logging

- iMBP logging system employs Kafka, Logstash, ElasticSearch and Kibana technology stack. Kafka passes data to Logstash which parses the data to desired format, then it pushes the data to ElasticSearch for storage, at the end, Kibana is used to render the data visually.

- Logging level will be dynamically configured through consul kv store
- The format of logged data will have basic information such as domain, project, serviceName, timestamp, ipAddress, type, errorMessage, rawData etc.
- Logging system will have a dedicated topic in Kafka such as Error

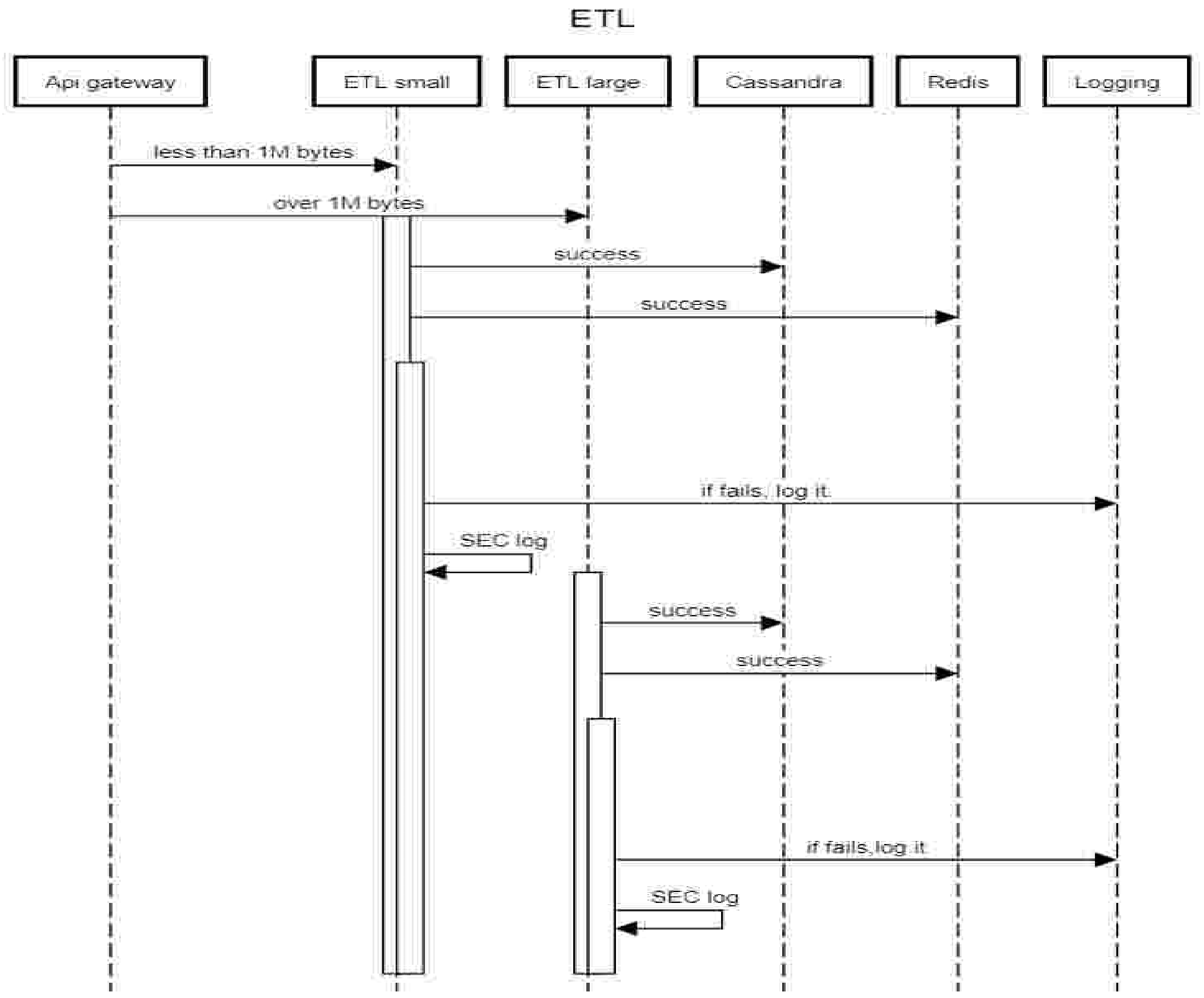
Monitoring

- Spring boot provides library allowing app to pass server related statistics such memory, CPU etc. to Prometheus. Gafana will be used as graphic interface for statistic display
- Domain and project related statistics such as # of request, size of SEC log etc. need to be defined as the project progresses

Data Storage

- MariaDB will be used to store administration, metadata, authentication and authorization related data etc. Consul configuration data also will be saved in the database as backup
- Redis and Cassandra are out of the design scope for now

ETL Sequence Flow


ETL
Api gateway
ETL small
ETL large
Cassandra
Redis
Logging
less than 1M bytes
over 1M bytes
success
success
if fails, log it
SEC log
success
success
if fails,log it
SEC log

SEC log Algorithm

- **SEC log algorithm is based on simplified LSM algorithm with variation of update and save operations. The SEC log system always operates on appending data at end of a file without modifying and deleting single data for fast write performance, also SEC log system provides index access which is stored in both disk and memory for durability and fast random read access.**
- **Save: index id is generated using input payload with Murmur3 hash function for both SEC log and iMBP logging system. To prevent duplicated entry, the id is checked against index file, app will do nothing if index exists and doesn't mark as a tombstone, otherwise, app will bring tombstone to a live data.**
- **Update: generate a new index id and old one will be marked as a tombstone, payload will be appended to data file.**
- **Purge: to purge a single data, the index of the data will be marked as a tombstone, the data will not be removed from data file. To purge all of the data, both index and data sec files will be cleaned up.**
- **Re-run: after saving single file to Cassandra, index will be marked as a tombstone. After running all of the data, index and data files will be emptied out.**