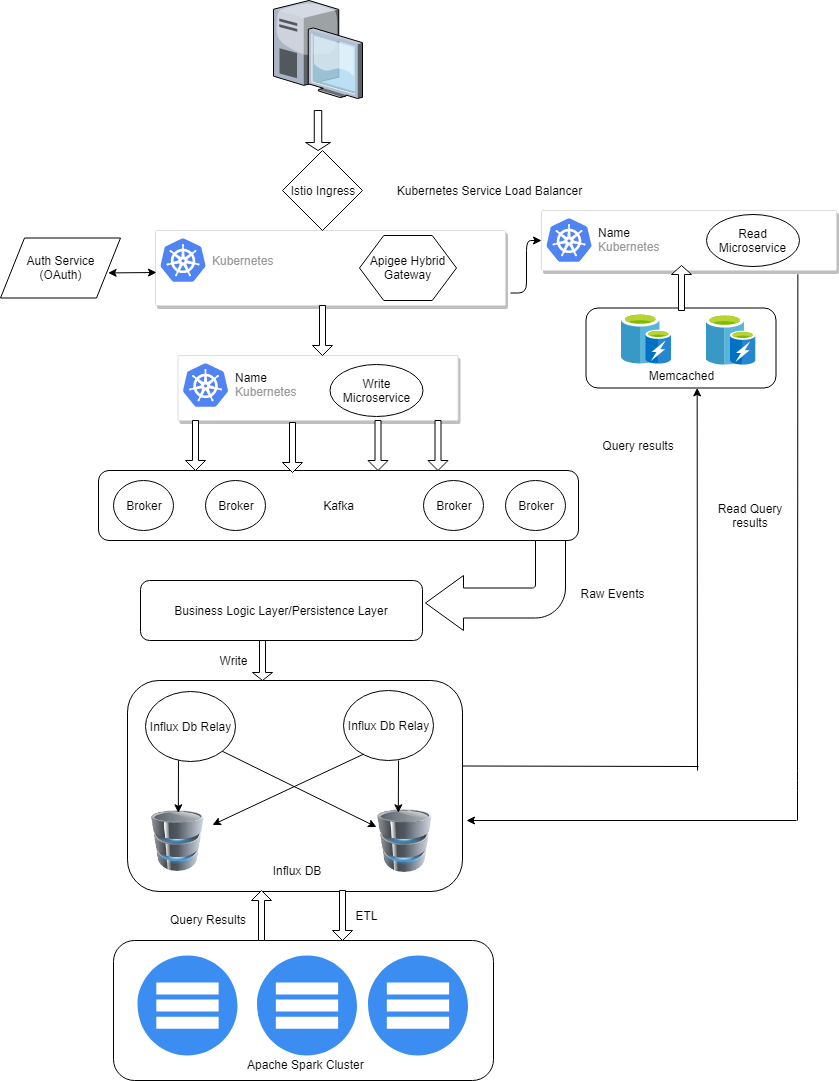
**ANALYTICS BACKEND SYSTEM DESIGN**

****

**Properties of the proposed architecture**

* Horizontal Scaling
* Fault tolerant
* High availability
* No single point of failure
* Modular

**Components:**

**APIGEE Hybrid Gateway** (https://docs.apigee.com/hybrid/beta2/what-is-hybrid):

Apigee Hybrid Gateway comes bundled with Istio Ingress.

Apigee Hybrid helps you manage all internal and external APIs with a single platform.

Apigee Hybrid is deployed on Kubernetes cluster in an environment close to our backend microservices thereby reducing latency.

It fronts are backend services with a proxy layer and provides an abstraction for our backend service APIs and provides security, rate limiting, quotas, analytics, and more.

It will act as the edge service or entry point for incoming requests from client browsers, devices. It will also be responsible for authentication of incoming requests by validating the tokens against the auth service which in turn will provide the JWT Token. The token is then used to authenticate to Write API/Read API

API Gateway will receive incoming requests over HTTPS and will be responsible for SSL termination at the API Gateway

**Kubernetes**:

Kubernetes is an open source orchestrator for deploying and managing containerized applications at scale. It has self-healing properties, intelligent scheduling, horizontal scaling and many more. All our micro services along with Apigee gateway will be deployed on Kubernetes.

**Web services :**

We will have three microservices:

* Auth API
* Writer API
* Read API

**Auth API**: It will provide authentication to the Writer Service and Read Service to ensure whether caller is authorized to make calls or not.

**Writer API**: The Write API will accept request of user activity and store the raw time series events in database which is then fed to spark clusters for processing.

The WRITE API will accept request from user website click event that needs to get stored in database on top of which processing and reporting can be done.

**Read API**: The Read API will receive request which has queries from the users.

These requests are then published to a separate Kafka topic on the Kafka service.

The READ API will accept request from Merchant users that want to see Analytics metrics, time series metrics

**Kafka**:

Kafka is a distributed streaming and message queue platform.

It is linearly scalable and fault tolerant.

High throughput.

This will serve as buffer between web service and DB to handle the traffic spikes.

**Influx DB with Influx DB Relay:**

Influx DB is a scalable datastore for metrics, events, and real-time analytics.InfluxDB is designed to be scalable, simple to install and manage, and fast to get data in and out.

Influx DB relay adds a layer of high availability on top of influx db.

**Memcached:**

Memcached is in-memory data storage and high-performance distributed memory cache service.

Memcached utilizes multiple cores hence memcahed becomes useful when dealing with high volumes of data like in our case.

Since we can only store a limited amount of data in cache, "Cache-aside" mechanism known as lazy loading serves best for design. Only requested data is cached, which avoids filling up the cache with data that isn't requested.

**Apache Spark**:

In-Memory Computation in Spark: Here the data is being cached so we need not fetch data from the disk every time thus the time is saved

Dynamic in Nature: We can easily develop a parallel application, as Spark provides 80 high-level operators

Fault Tolerance in Spark: Apache Spark provides fault tolerance through Spark abstraction-RDD. Spark RDDs are designed to handle the failure of any worker node in the cluster. Thus, it ensures that the loss of data is reduced to zero

**FLOW:**

**Write API:**

* When user clicks on the website a click event or request is sent to APIGEE Gateway through istio ingress.
* The Gateway is the entry point of the backend system it pulls out the token from the header which is unique to that website and sends that token to the auth service which comes back with the JWT token.
* Once the JWT token is received we use that that token to call the Write API (based on the request type).
* The write api then publishes the events to a Kafka topic. Then the business logic layer will write these raw events to the Influx DB.(Every event will have a unique timestamp).

**Read API:**

* The user's request for metrics is handled by the different REST Api.
* Firstly the metrics will be checked in memcached for the returning query if the data is not there raw data will be fetched from unprocessed DB and pushed to Spark cluster for processing.
* Once the data is processed it is pushed back in to the processed db so that there is no conflict between processed data and raw data because of same timestamp which is then fetched by the read api. Since the processing of data is being done on Spark which is 100 times faster than Hadoop it is expected to produce the results quickly.