**Requirements**

**Ecommerce:** Design the ecommerce which fulfil the following requirements.

1. Requirement: 10,00000/Sec {Peak hours: 50,00000/Sec}
2. High Concurrency (Millions requests per sec), for the e-commerce business.
3. Strong consistency of the user transactions, which involves **making payment and product** delivery.
4. **Look Up < 50 ms** for certain validation use case, to complete the workflow. (Cache is used wherever required)
5. Strong data **encryption** in motion and at rest for complete data security. (**https**) and Encryption at rest (**Payment history, user data will be encrypted using feature always on AES 256 feature**)
6. For large data payload Json payload shall be replaced by **binary payload**.
7. Standard open auth protocols to authenticate via google and Facebook.
8. **API metering** and payment for business APIs. All APIs shall have **common entry point**, even when diversely deployed at different Infra. (**API Gateway**)
9. Data **replication for High availability**, setting up database in the **Read Write config** for better performance.
10. Following **AP in the CAP** theorem, what are key changes required for this?
11. **Caching static content** including videos and photographs. (**CDN: Multi-vendor**)
12. Processing large data for the **OLAP (analytics)**, which may **process billions** for records for higher management reporting and predictive analytics.
13. **Re-trial strategy** and **Dead letter queue** to reconcile any kind of transaction failure. Introduce **compensating transaction for a distributed system**.
14. All **support audio calls** shall be persisted for 6 months (preferably on a blob storage) and then slowly it shall move towards archival over next 1 year.
15. Business insight of number of requests per user.

**Design Requirement:** Observability is required to measure the SLA and SLO

**Assumption:**

1. Application is used by the users across the Globe.
2. User will use Mobile/Desktop/Laptop
3. Search is required, also based on missed search analysis we can identify the intent/demand of the users to add more catalog items.
4. User can put rating to the product.
5. If millions of users visit the site, assuming 25% user purchase, 75% users visit the application and check product.
6. We will not be handling payment details, rather will be handled by Payment service provider.

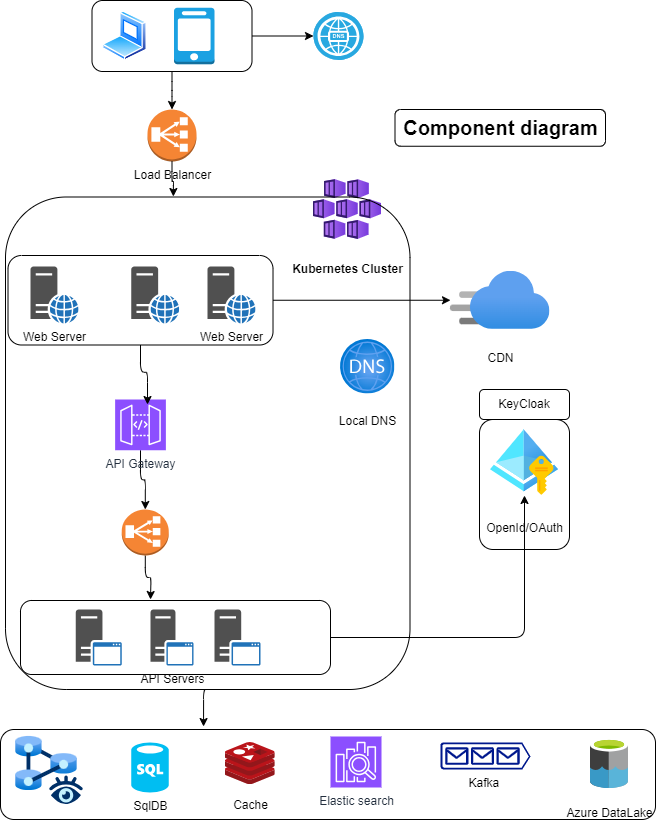
**Functional requirements**

1. User can visit the site.
2. Show the default products.
3. Can search the catalog/products, and filter the products.
4. Add items to Cart.
5. Add items to wish list.
6. Proceed for checkout, make payment.
7. Item count in cart list can be eventually consistent.
8. Payment details (user account, card details) will not be saved locally. Only payment reference for given order will be saved. Web hook will be needed to get the payment status which will update the status.
9. Following are the different payment methods:

* CC/Debit /NEFT

1. Notification service for the various use cases placing order, changing the order status and payment completion/failure.
2. View order and order history: Recent Order (this can be served from the hot DB)/Past orders/Historic (Archived) orders.
3. User and payment related details should be encrypted.
4. Payment: When user makes payment, it should be **idempotent** (Uniquid)
5. When placing an order: payment is successful: reduce the inventory count.

**Overall high-level design:**



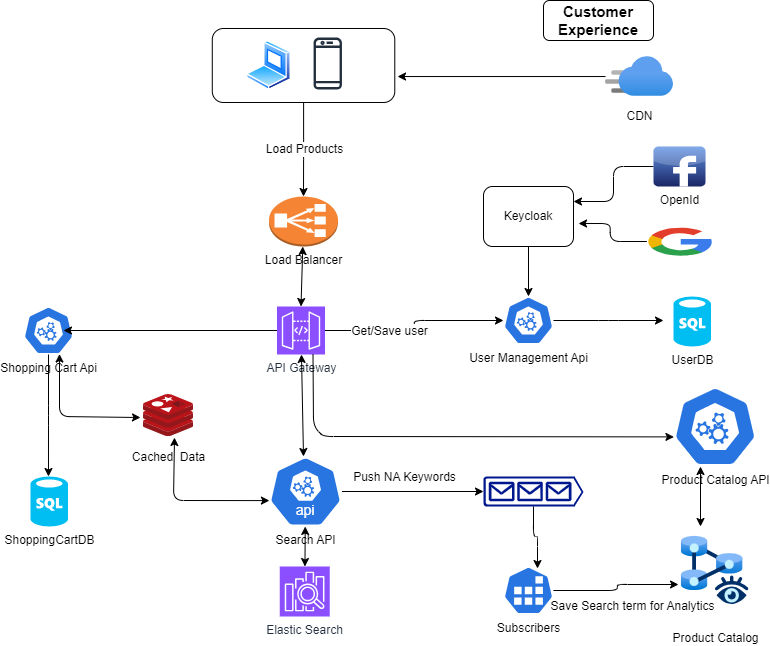
**Component details:**

Since this application is traffic heavy, it should be deployed in the cloud.

1. Azure/AWS can be chosen as a cloud provider, it depends upon the choice of the organization.
2. Since traffic volume is high, to serve the purpose we can deploy the application on the AKS/EKS
3. Redis is chosen as a cache provider because of its scalability and enterprise grade capability.

Following is the component diagram for the user visit to the application:

Calculate Time: 500 microsecond+



**User Experience: Visit the application: Following are the steps.**

**Critical (This component is should be performant and highly available to keep customer engaged)**

1. Users will be using the application across the globe; application must be deployed across the globe.
2. Whenever application is requested by the user, based on the user IP user will be directed to the nearest region. Example (www.myecommerce.com-> [www.myecommerce.in](http://www.myecommerce.in))
3. Request will be received by the load balancer and it will pass the request to the particular Web Server. Web server will call the Web API.
4. API request will pass to the API gateway and the can flow to the particular API.
5. We need to rate limit the API calls at gateway layer.
6. CDN will serve us the images and media of the product.

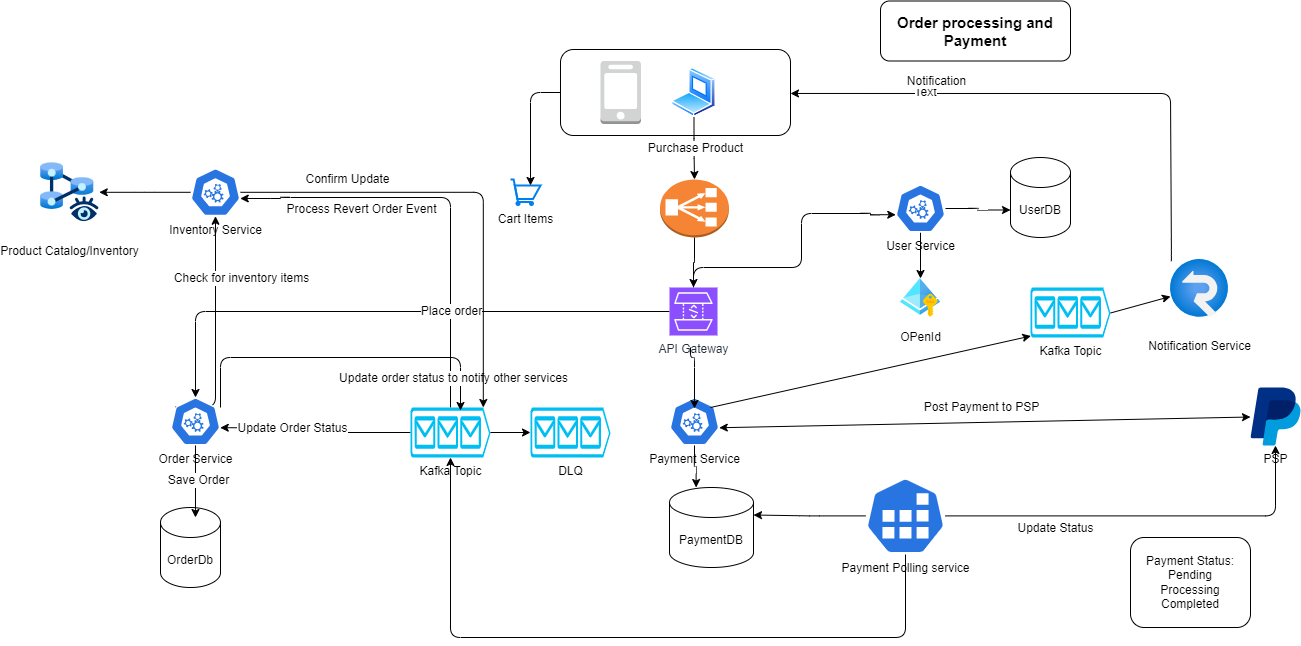
**User view the catalog** (landing page):

1. User loads the application very first time, we can load the top commonly browsed different category products on the home page. This data can be prepared daily, hourly based on the basis of the business requirement.
2. When user searches any items call Search API:
3. Check if item in the cache if present return. (We can say top 1000 searched Catalog items). On different occasions most of the users **searches specific items**,
4. Else pull items from the Elastic (indexed items) search.
5. If item is not present push to the topic, later it will be saved to missed search keywords in the database. It will help to know the user intent.
6. In order to show the details user clicks on the particular item, Product catalog API will be called which shows the details of the product.

**User adding items to the cart:**

1. Select the item on the detailed page.
2. While adding items to the cart, data will be saved in the cache later can be saved to the database.
3. We can have strategy allow max items to the cache.

**Order placement and payment**



**Place the Order: Critical (**This is the critical path of the application reliability is very important)

**Order and Payment estimation**

Peak business time: Festival time (50 orders/second): 43,00000/day

Normal business: 18 orders/sec= 1500000/day

1. User must be authenticated to place the order.
2. User selects the items in the cart to be purchased.
3. **User Place the order and make the payment**: Checkout Process

* Select Address for the delivery.
* Select the payment method. Proceed (In this step various validation steps are performed)
* Validate if items in cart are present in the Inventory:

If yes then update (**reduce the number of items in inventory)** and save the order creation request event (CartId, NumberOfITems, CustomerId) to the Order topic. This will help to revert in case order creation failed. If items not present notify the buyer.

* Save order in the order database.

Order is created in the order database (OrderId, Payment method). Publish event (OrderId, CartId, UserId, Status)

If order creation **fails, retry exponentially** then. If order creation fails in the OrderDB, publish event of order failure (CartId, UserId, NumberOfItems)

**If payment is not done after a certain specified time, Cancel the order and publish the CancelOrder event (OrderId, CartId, for the failure for Order in the Kafka. Notification will be sent to the various stockholders (user, admin).**

1. Once order creation is successful in the order database, **Payment API** is called with **order Id, UserId**.
2. Payment API will save the (OrderId, PaymentOrder) details in the PaymentDb, then redirect to the **PSP page**.
3. User provide details (in case NEFT) then (Payment service URL and OrderId will be send to the PSP).
4. Payment service URL: This is registered with PSP at the time of purchase.
5. PSP will return the response, if its successful status will be saved in the PaymentDB. User will be redirected to the success page. If payment status received as following
6. Timed out
7. Failed
8. Network Failure
9. If taking long time, PSP will call the Web Hook url and deliver the status later.
10. PSP provides the API to get the status, we can poll (on hourly or business specified time basis).
11. Once status is updated in the PaymentDB, event can be published to the Kafka topic.
12. Order service will update the order status and notification will be sent to the user using the notification service.
13. Every end of the all PSP provides the reconciliation file, using this we can reconcile the payment. If any change is required, will be notified to the Finance team.
14. There are 2 parties which will receive the payment(buyer account to ecommerce account, ecommerce account to the seller)

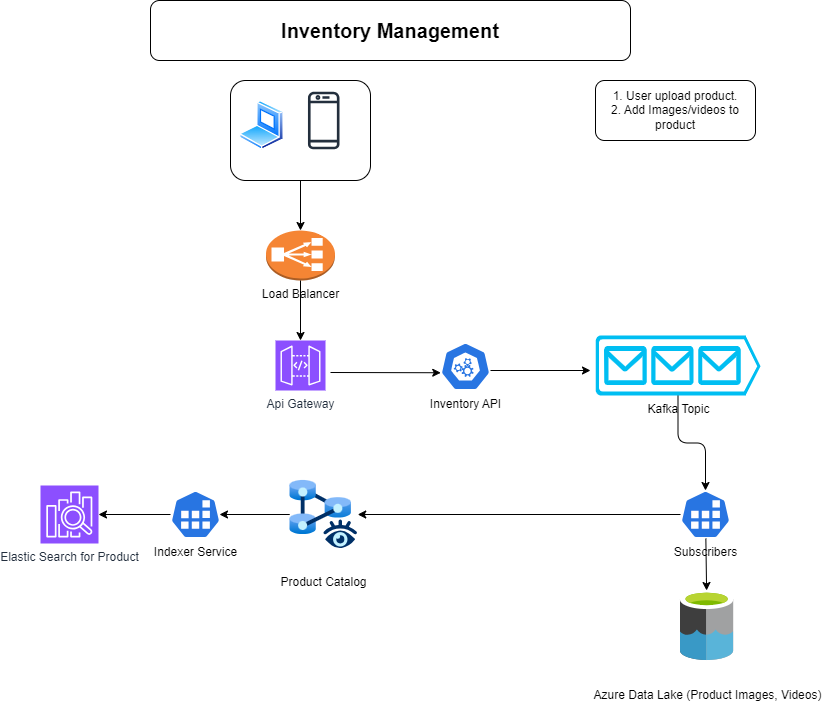
Order data partition:

**Partition Key:** Country\_Week: Large countries should have in separate partition and small countries can be grouped. Avg order 18/Sec (18\*86500) =15 million.

For 2 weeks we can keep in the hot partition, then move to warm partition.

If request is greater than 2 weeks then, move to warm storage and will be served from the warm.

**Inventory management:**



**Creating Inventory:**

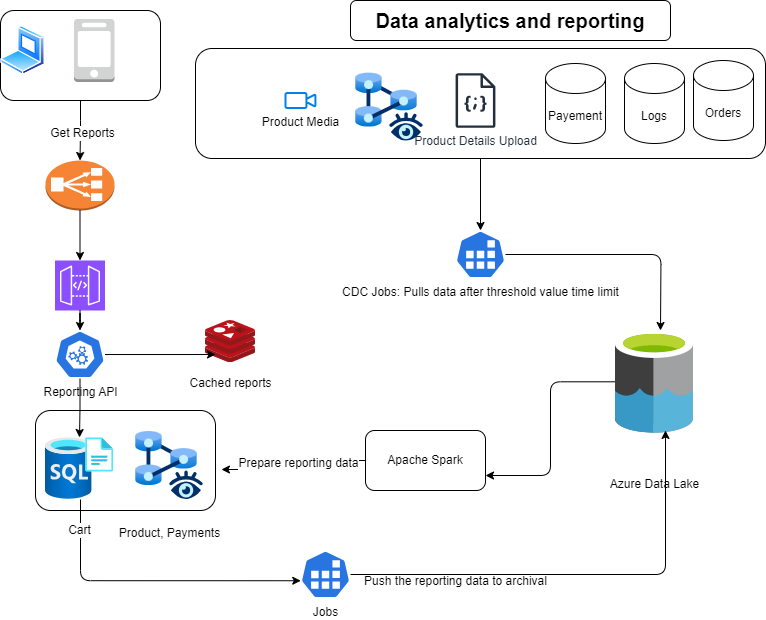
1. User should be authenticated to add/update products.
2. Authenticated users can add update items in following method:
3. Upload JSON
4. Add using UI
5. Product file will be uploaded to the Data Lake and path will be returned, path along with details will be saved to the Kafka product creation topic. Subscriber will pick the messages and save to the product catalog database.

Why through queue?

There could be many uploads, if we upload immediate service may face high load. In case it fails user to click retry.

If catalog creation is in the background, user will get the status through notifications.

**Reporting and analytics**



**Analytics and reporting:**

1. Periodically we can pull the data from the live/hot storage and keep saving to the Data Lake.
2. Using Apache spark we can process the data in the desired form and place to the relational database, Cassandra as need. Cassandra is good fit for analytics load. We can’t use the live database because it can cost the performance of the application.

**Data partitioning strategy:**

Its required multi-level of data partitioning strategy, listed below

1. Partition by country: It’s for the data regulatory rules of the country

* Partition by province.
* Partition by City.
* Partition by userid.

**Replication:**

For high availability we need maintain 1 master copy and at least 2 replicas of the data.

One copy in the same region and another copy should be geo replicated.

**Availability and Partition tolerance:**

To achieve AP we need our system to be eventually consistent. We need to trade consistency for AP.

Payment flow:

1. Hit pay button, request not reached then retry using exponential backoff mechanism.
2. Request is sent, but no response: Put the request in Processing state.

If its successful the delivered the response on webhook.

1. Poll for some time.
2. If payment is successful:
3. Send notification to customer.
4. On failed payment:
5. Update the order status
6. Update the inventory
7. Notify customer.