Design and Implementation of Platform as a service for Pipelined Execution

Domain of the Problem

We are dealing with an ETL process of a Data Warehouse and business Intelligence system where we will be providing different kind of services for extraction, transformation and loading of real-time streaming data so that it can be later used for OLAP Queries.

Expected Data Stream

We have a raw data set of orders from an online e-commerce OLTP system. Transformations need to be done on this data set so that it can be used for quicker responses to OLAP queries. Table schema for raw data set is given in fig (a) and it needs to be transformed with the help of various services. We are streaming records of orders in real-time. And the transformed data will be passed between the services as a pipelined stream.

Order_number	Customer_id	Customer_fname	Customer_lname	Date_of_birth	Product_name

Product_price	Product_category	Order_timestamp	Delivery_timestamp	Currency_format	Prime_membership_expiry_date	Time_zone

Identified Services for the platform

1. Merge Service
2. Calculation Service
    a. Age Calculation Microservice
    b. Expiry Calculation Microservice
3. Format Conversion Service
    a. Currency Conversion
    b. Date Conversion
    c. Time Conversion
4. Data Loading Service
5. OLAP Query Service

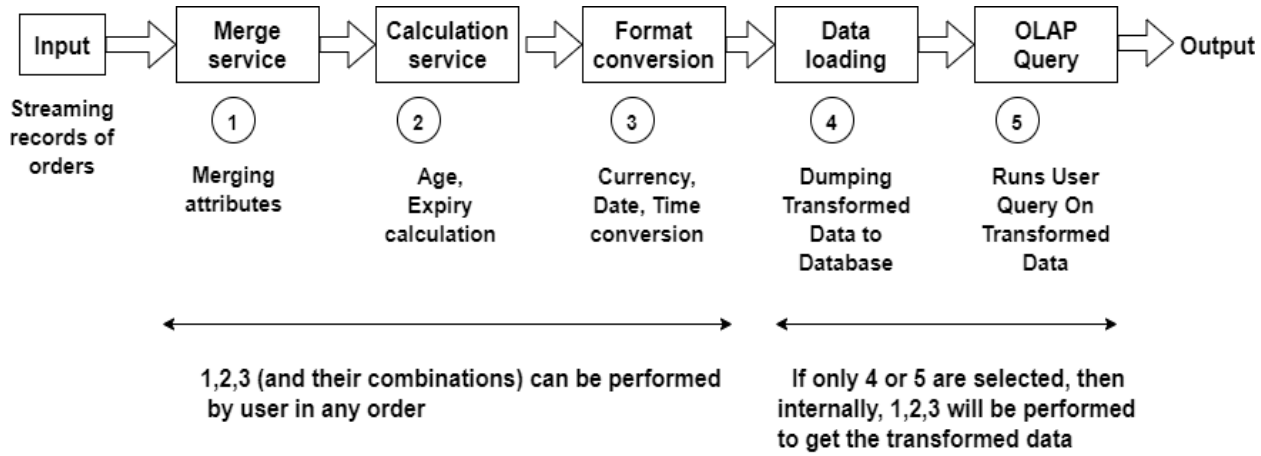


Fig. Pipelining of Services

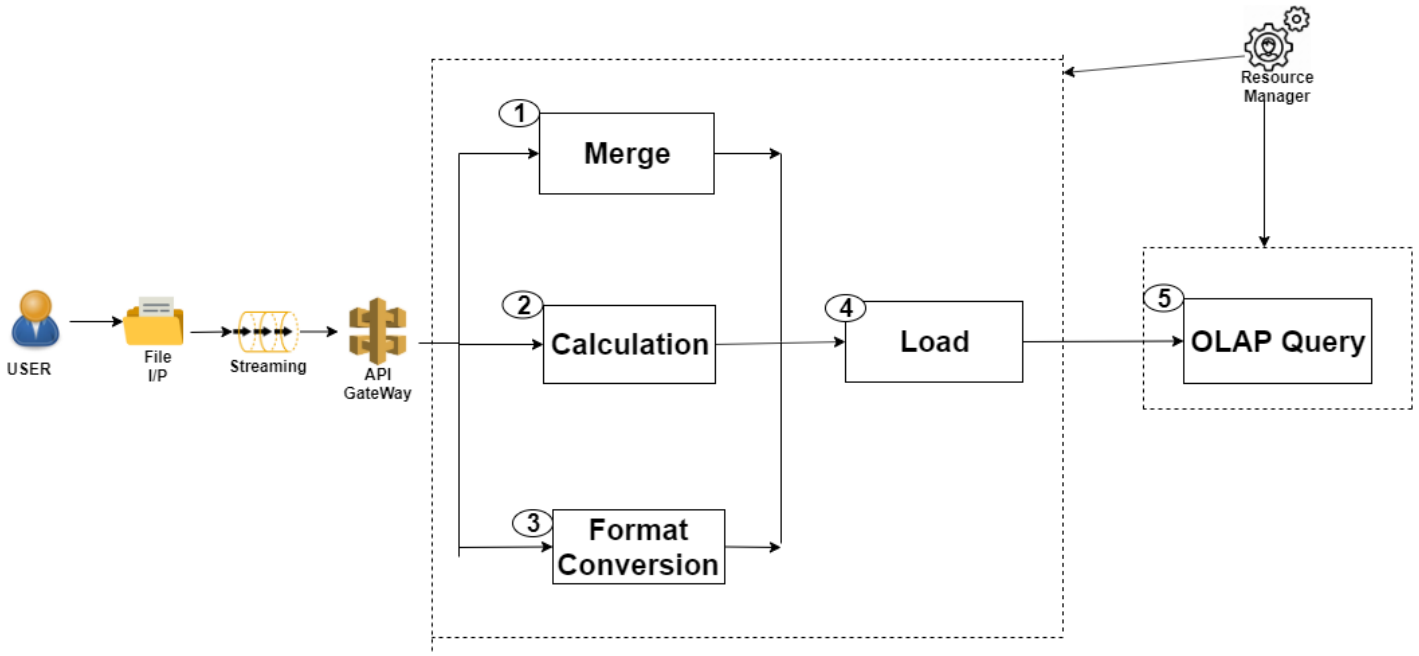


Fig. System Architecture Diagram

1. **Merge Service**

→ **Monolithic or a set of Micro services**

- Monolithic Service

→ **Specify its needs such as a storage, database etc.**

- No Storage/Database, Python runtime environment

→ **Specify the format of input it takes**

- Order_number
- Customer_id
- **Customer_fname**
- **Customer_lname**
- Date_of_birth
- Product_name
- Product_price
- Product_category
- Order_timestamp
- Delivery_timestamp
- Currency_format
- Prime_membership_expiry_date
- Time_zone

→ **Explain the coarse-grained steps it will follow**

- Customer_fname + Customer_lname = Customer_name

→ **Specify the format of output it gives**

The output record is appended with

- Customer Name

→ **Identify what technical alternatives exist for connecting the services in the pipeline and chose one with justification**

- Merge Service can be pipelined to any of the other transformation services (2 &3) or the **data loading service** directly

2. Calculation Service

→ **Monolithic or a set of Micro services**

- Set of Microservices
  i. Age calculation
  ii. Prime Expiry Calculation

→ **Specify its needs such as a storage, database etc.**

- No Storage/Database, Python runtime environment

→ **Specify the format of input it takes**

- Order_number
- Customer_id
- Customer_fname
- Customer_lname
- **Date_of_birth**
- Product_name
- Product_price
- Product_category
- Order_timestamp
- Delivery_timestamp
- Currency_format
- **Prime_membership_expiry_date**
- Time_zone

→ **Explain the coarse-grained steps it will follow**

- Current_system_date - date_of_birth = Age
- Current_system_date – prime_membership_expiry_date = Prime_Expiry_Time_left

→ **Specify the format of output it gives**

The output record is appended with

- Age
- Prime_Expiry_Days_Left

→ **Identify what technical alternatives exist for connecting the services in the pipeline and chose one with justification**

- Calculation Service can be pipelined to any of the other transformation services (1 &3) or the **data loading service** directly

3. Format Conversion Service

→ **Monolithic or a set of Micro services**

- Set of micro services
  i. Currency Conversion
  ii. Date Conversion
  iii. Time Conversion

→ **specify its needs such as a storage, database etc.**

- No Storage/Database, Python runtime environment

→ **specify the format of input it takes**

- Order_number
- Customer_id
- Customer_fname
- Customer_lname
- Date_of_birth
- Product_name
- **Product_price**
- Product_category
- **Order_timestamp**
- **Delivery_timestamp**
- **Currency_zone**
- Prime_membership_expiry_date
- **Time_zone**

→ **explain the coarse-grained steps it will follow**

- Product price is converted to standard prices by selecting the currency format and applying the required conversion factor to it.
- Order Timestamp is converted to standard Date-Time using the Time Zone attribute value

→ **Specify the format of output it gives**

The output record is appended with

- Standard Price ($)
- UTC Timestamp

→ **Identify what technical alternatives exist for connecting the services in the pipeline and chose one with justification**

- Conversion Service can be pipelined to any of the other transformation services (1 & 2) or the **data loading service** directly

4. Data Loading Service

→ **Monolithic or a set of Micro services**

- Set of Microservices
  i. Order Sales by Category
  ii. Order Sales by Country/Time Zone

→ **specify its needs such as a storage, database etc.**

- Mysql Database, Python runtime environment

→ **specify the format of input it takes**

- Order_number
- Customer_id
- Customer_fname
- Customer_lname
- Date_of_birth
- Product_name
- Product_price

- Product_category
- Order_timestamp
- Delivery_timestamp
- Currency_format
- Prime_membership_expiry_date
- Time_zone

Each streaming record contains above data along with newly appended attributes.

→ **Explain the coarse-grained steps it will follow**

- This service will receive the transformed data and put that data into the table as requested by user.

→ **Specify the format of output it gives**

- Transformed Data is dumped to the database. The output will be status 200 if data is loaded into the database correctly. Else appropriate error message is displayed to the user.

→ **Identify what technical alternatives exist for connecting the services in the pipeline and chose one with justification**

- The input pipeline to Data Loading Service can be from any of the above 1,2,3 services but the output has to be pipelined to OLAP Query Service

5. OLAP Query Service

→ **Monolithic or a set of Micro services**

- Monolithic Service

→ **specify its needs such as a storage, database etc.**

- No Storage/Database, Python runtime environment

→ **specify the format of input it takes**

- User Defined OLAP Query

→ **explain the coarse-grained steps it will follow**

- OLAP query is fired upon the data and results are reflected to the user

→ **specify the format of output it gives**

- Depends on the type of query user fires

→ **Identify what technical alternatives exist for connecting the services in the pipeline and chose one with justification**

- This is the last service of the pipeline and will directly produce the result.

Auto Scaling

→ **Compute your system's capacity.**

Capacity of a system can be broadly classified into two categories namely: -

- Storage Capacity

  Let us consider the storage capacity available to one VM is 2 giga bytes of storage. There are 3 VM in one physical Machine and we have 3 physical machines in total.

  Capacity of one Physical Machine: 6GB

  Capacity of Whole System: 18 GB

- Performance capacity

  Performance is an end-to-end property which requires all associated underlying resources to work together, including network, storage and computational resources. The main objective of performance is to provide low latency of user requests.

  RAM allocated to one VM is 2 GB. It is scaled vertically till 10GB in each particular physical machine.

→ **Identify the parameters for deciding on auto-scaling**

- CPU Utilization
- Memory Utilization
- Storage Utilization
- Number of Instances/VM

→ **Explain how do you monitor these parameters**

- We will track and store per instance metrics including request count and latency, CPU and RAM Utilization with the help of Linux shell commands.
- Minimum number of instances to run is fixed at 1.
- Maximum number of instances to run is fixed at 5.

→ **Identify ways to optimize resources with auto-scale.**

- Specifying maximum and minimum instances in a physical machine provides an automatic scaling mechanism.
- We will develop an algorithm which will first determines the current VMs/Containers with the use of physical resources above or below given threshold numbers.
  - If the uses of resources of all VMs/Containers are above the given upper threshold, a new VM/ Containers will be replicated, provisioned, started, and then perform the same computing tasks in the virtual cluster.
  - If the uses of resources of some VMs are below a given lower threshold and with at least one VM that has no computing job, the idle VM will be terminated from the virtual cluster.
  - If the uses of resources of all VMs in a virtual cluster exceeds the threshold, then requests are transferred to the next node present.

Infrastructure

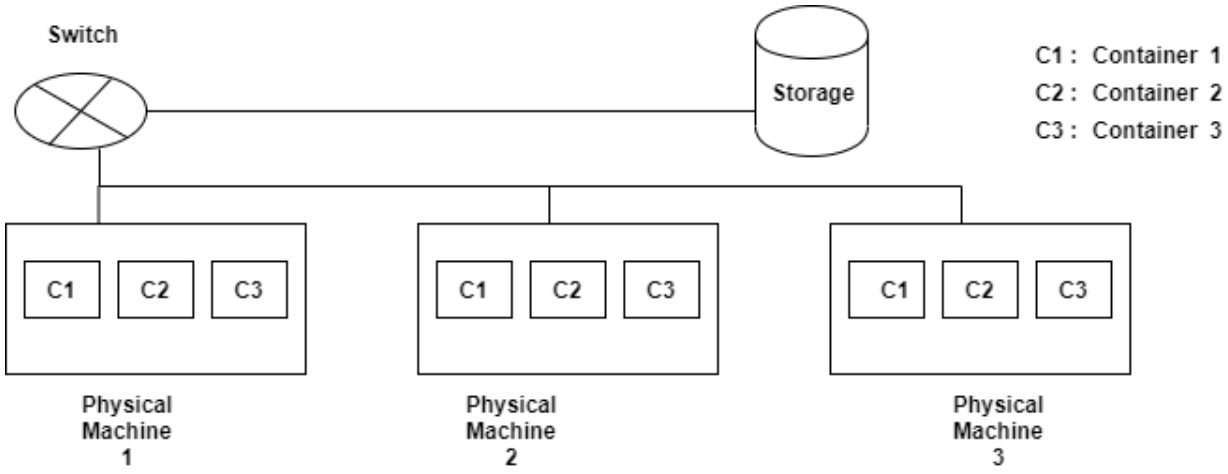


Fig. Infrastructure depicting horizontal and vertical scalability

The above figure demonstrates how vertical scalability will be provided by containers (virtual clusters) and horizontal scalability will be provided by physical machines.

→ **Will your services run in VMs or in containers? Justify.**

Our services will be running on **containers** as they have a **lower overhead than VMs** and container systems typically target environments where **thousands of containers are in play**. Container systems usually **provide service isolation** between containers.

→ **When multiple users identify different pipelines with streams at different data rates, it becomes very important to optimize your resource usage. Explain resource monitoring and resource scheduling of your platform.**

We have fixed minimum number of instances to be 1. If there is single user with a low data rate, then only resource usage will be in one VM. In case, if multiple users came with different data rates then we need to monitor CPU and RAM utilization and if

that exceeds the threshold, then we first scale up vertically within virtual cluster, in case when virtual cluster resource consumption gets exhausted then we scale horizontally by counting in new nodes which are available in cluster. If some of the users become idle and stop sending the requests, then we check the resource consumption of the VM and if it is below a threshold or a VM is being idle we terminate the VM and send it to idle state. This helps in optimal utilization of resources.

SLA

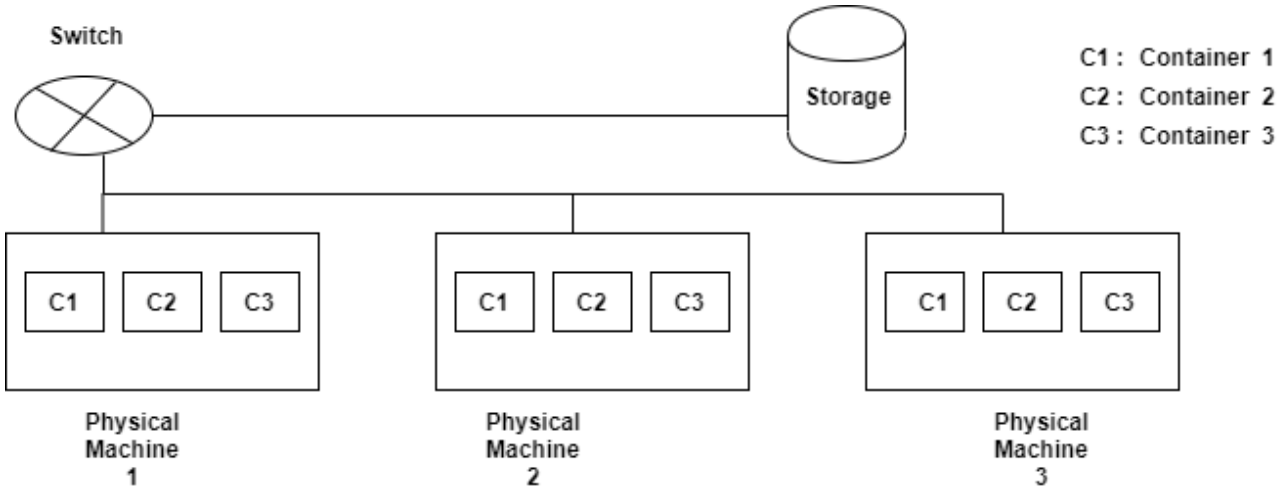
→ **SLA parameters**

- Availability of Service (Uptime)
- Latency or response Time
- Service Components Reliability
- Each party accountability
- Warranties

→ **SLA monitoring**

- Reports to be generated on a regular basis regarding all of the SLA parameters and sent to the consumer.
- We are planning to use open source tool SmokePing for SLA monitoring.
- We will ensure that availability and response time are periodically reported. Immediate e-mail alerts of intrusion detection, breaches and outages are delivered.

Availability



Assumptions:

Network failure is 0%.

Storage failure is 0%.

Availability of node 1 = 99%

Availability of node 2 = 99%

Availability of node 3 = 99%

Calculation of availability and failure:

- We are using "**Active-Passive**" configuration. Means, initially requests will come to node 1. If node 1 fails or gets overloaded, node 2 will become active. Similarly, if node 2 fails or gets overloaded node 3 will become active.
- Once the node recovers it will be in stand-by mode, so in this sense we are using "**N plus 1**" configuration.
- System will fail only when all three nodes will fail.

Since availability of node 1 = 99%

Therefore, failure rate of node 1 = 1% = 0.01

Similarly, failure rate of node 2 = 1% = 0.01

And failure rate of node 3 = 1% = 0.01

Therefore, the total failure rate of system = 0.01 * 0.01 * 0.01 = $(0.01)^3$

Hence Availability of system = 1 - $(0.01)^3$

= 99.9999 %

Pricing

→ **Design a pricing mechanism for your service?**

We will be charging on an hourly basis for usage of RAM, CPU along with charges per transferred GB basis.

→ **How will you monitor?**

We will be monitoring usage with the help of container manager stats services. An example is demonstrated in the figure below.

```
$ docker stats redis1 redis2

CONTAINER           CPU %               MEM USAGE / LIMIT     MEM %
redis1              0.07%               796 KB / 64 MB        1.21%
redis2              0.07%               2.746 MB / 64 MB      4.29%
```