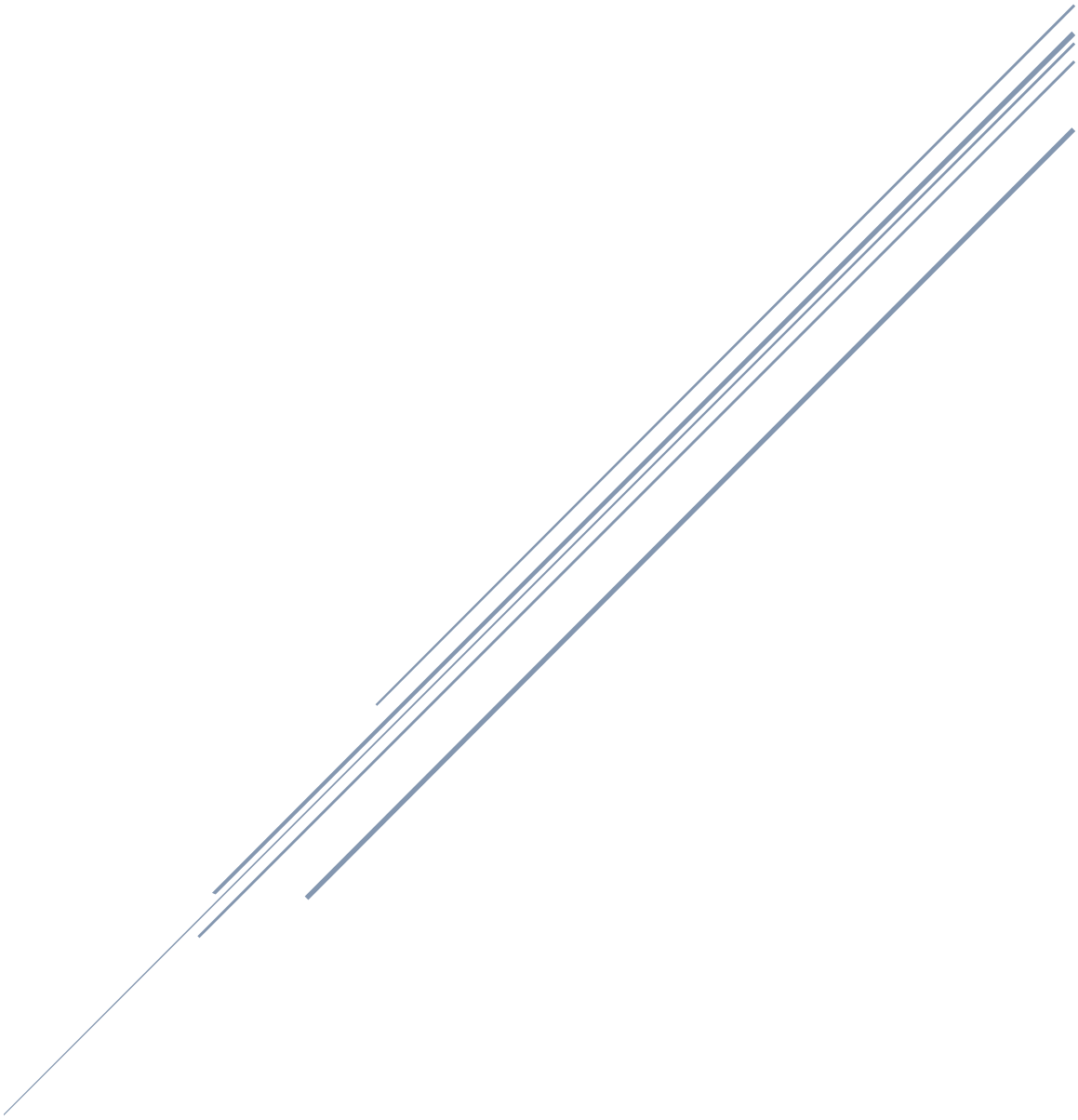
POLYGLOT PERSISTENCE

NoSQL Assignment



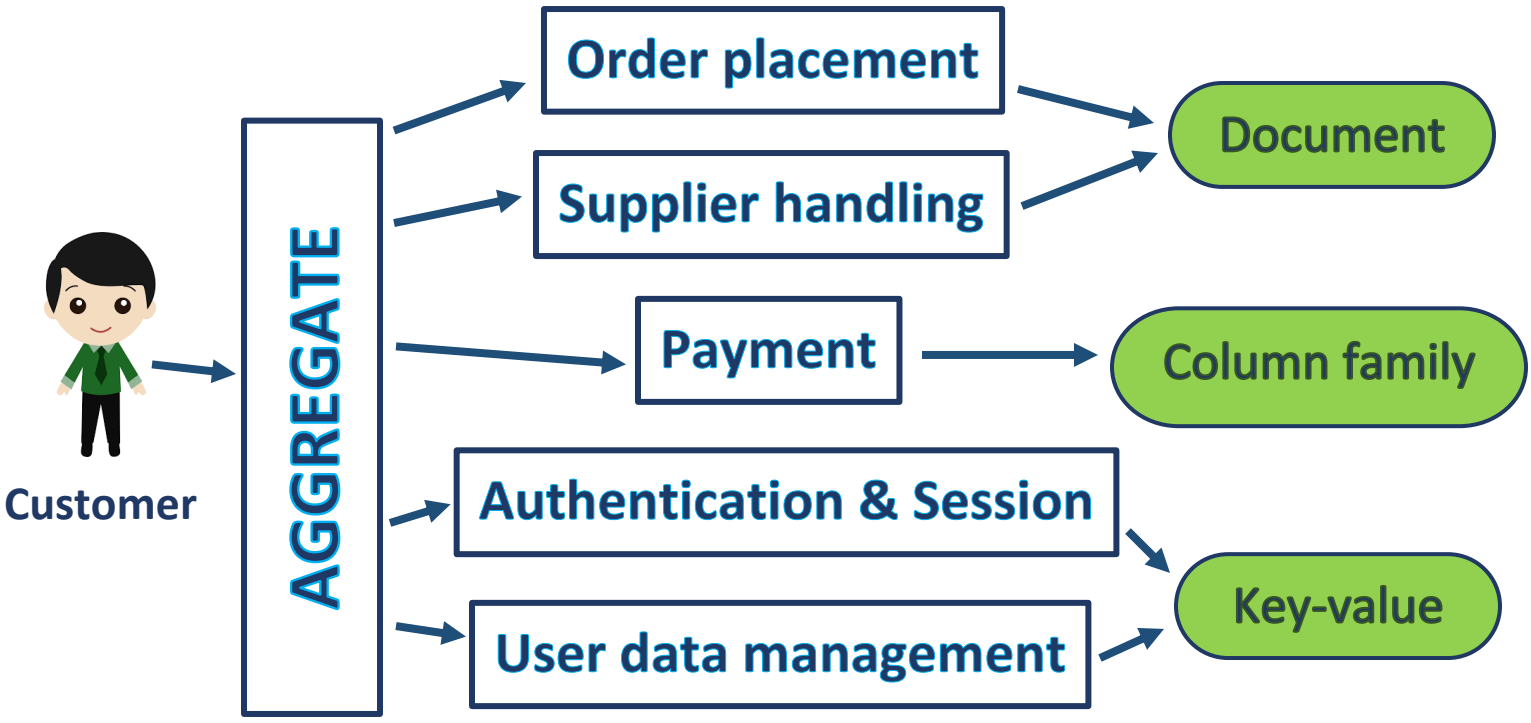
H.S. Abeygunawardana
18000029

Contents

Order Delivery System (ODS)

With the current circumstances of the world, the future of all business models has become remote and web based. The plus side of this is, a vast range of database systems can be employed to manage all sorts of data generated by a business to gain various kinds of benefits rather than having the work done manually. A carefully built system of microservices can enhance the business's productivity by many folds.

High level architecture design for ODS



Authentication, Session & User data management

The person attempting to log into the system will be verified and authenticated by the system. Session or token should be issued to the user that succeeds in login in, which is to be used identify user's online activity. User data management is a functionality that is only available for order management staff, which allows the manual order data entry and supplier management.

Desired qualities

1. Availability

Sufficient data to uniquely verify any user that attempts to log in should be widely available, and the login process should be quick and easy. Session or token data of the user should also be kept available, as user returning after a disrupted connection should be readily admitted.

2. Durability

User data including credentials used for authentication should last for a long time without the user having to update and re-register in the system frequently. However, session and token data of a user needs not to be durable than a couple of hours as a user is not expected to stay online longer than that.

Database selection

1. Database type

Key-Value databases, because they generally have great performance and can be easily scaled due to their use of primary-key access.

2. Database of choice

Riak, due to its short read and write times, high availability, and durability. But scalability factor will be slightly compromised relative to the other Key-value databases.

3. Database query

For inserting user data

```
# Choose the bucket to store data
bucket = client.bucket('test')

# Supply a key to store data
# ``data`` can be of any form Python's ``json`` encoder can handle
user = bucket.new('consumer 1', data={
    'name': 'Hiruna',
    'age': 24,
    'organisation': 'UCSC',
})

# Save the object to Riak
user.store()
```

Order placement & Supplier handling

An order should be recorded immediately after the customer places it in the system. Then this microservice raises an event for the unclaimed order, so that the supplier handler can trigger and assign the order to a fitting supplier. Once a supplier claims the order, another event will be raised to update the order record.

Desired qualities

1. High scalability

The number of order records can grow to large values very easily. Hence it should be possible to scale the system without added effort and without consuming much time.

2. Highspeed read-write functions

The rate of incoming orders per second and the number of matches made between orders and suppliers can be overwhelming. Hence the system should be capable of entering those records instantly without keeping any backlog.

Database selection

1. Database type

Document databases, because they can provide effortless scalability with great performance speeds due to their ability to create indexes and load balancing.

2. Database of choice

MongoDB, for the fact that it being the most popular NoSQL database for scalability, even its name implying the *Humongous* nature of it.

3. Database query

For updating customer data

```
db.Customer.update(
   { name:"Hiruna" },
   {
       $set: { Age:”24, Residence:"Kesbewa" },
       $currentDate: { lastModified:true }
   }
)
```

Payment

Once an order is completed and delivered to the customer, payment is received either online or on delivery. This payment is to be split among the Order delivery system and the supplier according to predetermined percentages, and the due amounts should be recorded in the respective accounts following ACID policies.

Desired qualities

1. ACID properties

The transactions should be atomic (all or nothing), consistent, independent from other records (isolated), and durable (with high security).

2. Minimal latency and fault tolerant

To conserve the ACID properties, a transaction needs to be executed within very small time period to reduce vulnerability for possible occurrences of technical errors.

Database selection

1. Database type

Column family databases, with their cleverly distributed systems, are able to give out fast processing speeds along with the ability to withstand system failures with resilience.

2. Database of choice

Cassandra, has proven its ability to replicate data effectively, resulting low latency and high fault tolerance, while providing all other expected qualities such as scalability without compromise.

3. Database query

For transaction data insertion

```
INSERT INTO transaction(PaymentId, CustomerId,
Supplier_1_payment,
Supplier_2_payment,
Supplier_3_payment)
VALUES(123, 12, 4000.00, 2500.00, 3500.00);
```

Assumptions

- Scaling of the system with the addition of affordable devices and machinery (batta vans) is possible without meeting up with any additional bottlenecks.
- The microservices operate independently as separate clusters and any sort of communication between them happens through the events posted by the event handler. This helps to keep track of the changes made to the system, giving the ability to rewind to previous stable snapshots in case of failure. It also leaves a good audit trail of the system's processes.
- The system is read intensive and focuses on availability over accuracy (rapid update) as the human interactions (ordering) takes relatively longer time and the change in services provided is not instantaneous and unannounced.

References

- https://www.thoughtworks.com/insights/blog/nosql-databases-overview
- https://youtu.be/7kX3fs0pWwc
- https://cassandra.apache.org/doc/latest/
- https://beginnersbook.com/2017/09/introduction-to-nosql/