Mongo-Parallel-Order-POC

Tuesday, February 7, 2023 7:58 AM

Description:

This POC proves that multiple instances of a microservice can use atomic mongodb library commands such that multiple instances of a microservice can run with only exactly one instance modifying a particular data record. In this POC, we have a simple order system example where there is microservice that will look/listen for new orders coming in, and when they do, the first instance that picks up the record processes and no other instance of the service can process that order. This will maximize processing throughput by allowing paralell instances to process orders at the same time with no overlap.
One of the key elements which makes this work is the CustomGroceryOrderRepositoryImpl.updateGroceryOrderStatus() method where the atomic findAndModify() command is used to find all status = 'open' orders in ascending order so that it always looks for the oldest open order to process LIFO. When the multiple instances run this command on the same order, only ONE will succeed with the update and the others will get a null back from findAndModify() and go back to listening for new orders to process. This one mechanism allows for the single instance processing behavior we are going for.
This will be useful in the payor agnostic services like the validator where we want only one instance of the validator to work on any one given set of 834 data.

Steps to run POC:

1) setup free mongo atlas cluster that you have access to

2) using mongosh, run the commands in scripts/create_test_data.txt to create an example database

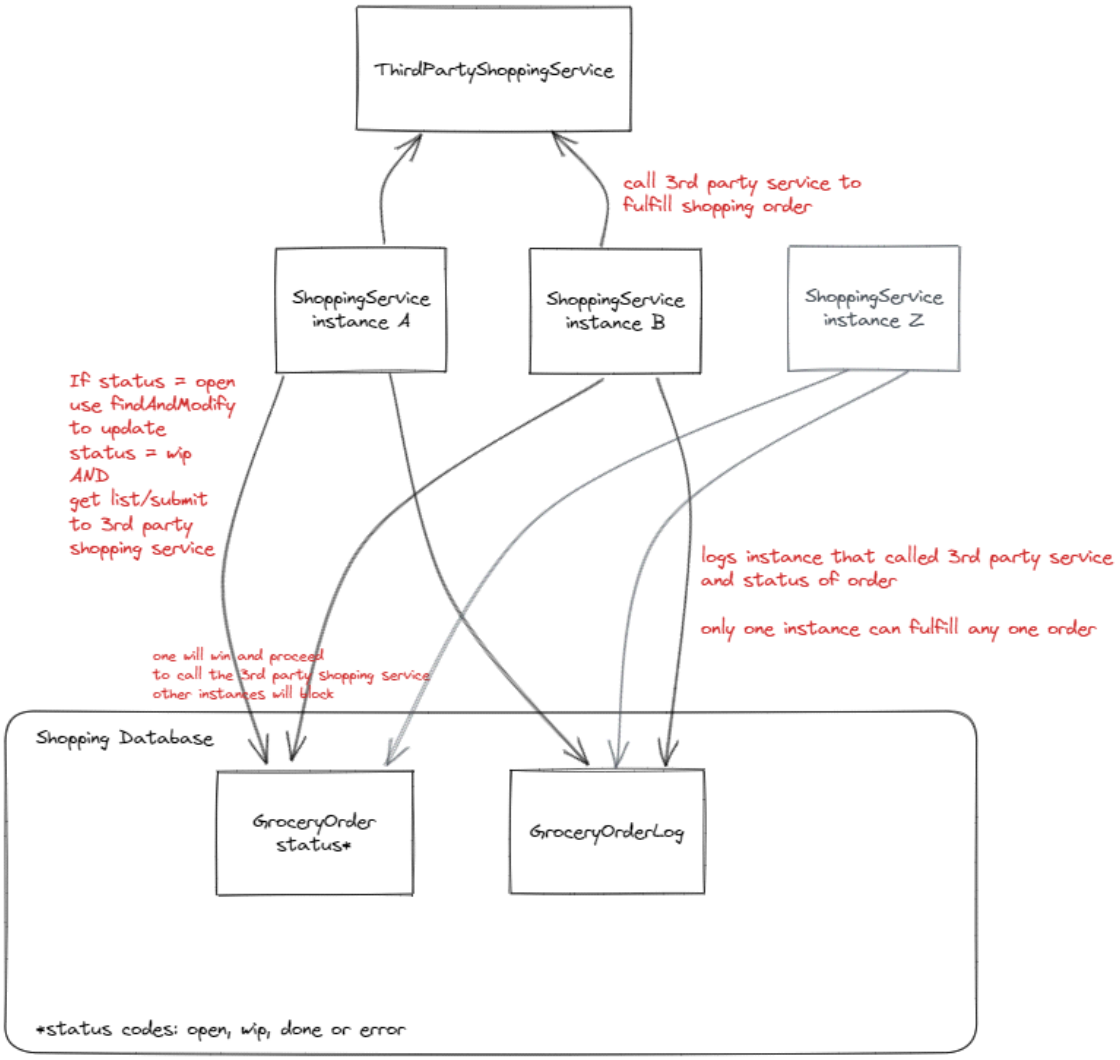
3) build the microservice

4) open 3 cmd windows and run each instance with it's own instance name, like this command which creates 'instanceA' of the service:
mvn spring-boot:run -Dmaven.test.skip=true -Dspring-boot.run.jvmArguments="-Djdk.tls.client.protocols=TLSv1.2" -Dspring-boot.run.arguments="--instance.name=instanceA --server.port=8080"

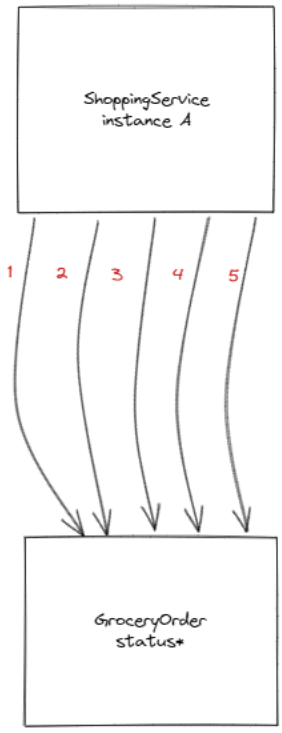
5) Look in the database at the GroceryOrder collection and notice all orders are moved from 'open' to 'wip'. Also look at the GroceryOrderLog collection and observe that no single order was updated by any more than one instance of the microservice and that the different records were processed by different instances of the service.

The Model

ShoppingServiceFullment Domain



ShoppingServiceFullment Service Flows



1 at startup, search for new grocery order status = open since last start

2 listen for new grocery order added/updated events

3 atomic modify order status to wip and call 3rd party to fufill orders

4 update status to done or error

5 log transaction/errors

ShoppingServiceFullment Service Pseudo Code

```
ShoppingService
instance A


final static instance = "instanceA"
getGroceries() // to find open status docs already there
collection.watch().filter(Filters.eq("operationType", "update"))
    .filter(Filters.eq("fullDocument.status", "open"))
    .forEach((Consumer<ChangeStreamDocument<Document>>) change -> {
        getGroceries()
    });

getGroceries(
dbObj = findAndModify({
  query: { status: "open" },
  sort: { list_update_timestamp: -1 },
  update: { status: "wip" }
})
try {
  if(dbObj != null) {
    callThirdPartyShoppingSeriveWrapper()
    logsuccess(instance,dbObj)
  } else {
    throw error
  }
} catch(exception e) {
  logerror(instance,dbObj,error)
  set status: "error"
}
```

```
ShoppingService
instance B


final static instance = "instanceB"
getGroceries() // to find open status docs already there
collection.watch().filter(Filters.eq("operationType", "update"))
    .filter(Filters.eq("fullDocument.status", "open"))
    .forEach((Consumer<ChangeStreamDocument<Document>>) change -> {
        getGroceries()
    });

getGroceries(
dbObj = findAndModify({
  query: { status: "open" },
  sort: { list_update_timestamp: -1 },
  update: { status: "wip" }
})
try {
  if(dbObj != null) {
    callThirdPartyShoppingSeriveWrapper()
    logsuccess(instance,dbObj)
  } else {
    throw error
  }
} catch(exception e) {
  logerror(instance,dbObj,error)
  set status: "error"
}
```

one will win and proceed
to call the 3rd party shopping service
other instances will block and get null
from findAndModify()

GroceryOrder collection in OnlineShoppingApp