QCon Workshop Runbook

Useful Links

Github: http://bit.ly/yb-qcon

Clone the following github repository - It contains two starter projects and their final versions. You will notice "//TODO: Implement this" in various code blocks. We will coding our implementations in those blocks.

```
Unset
mkdir ~/gitsandbox
cd ~/gitsandbox
git clone https://github.com/akscjo/yb-qcon-workshop.git
```

QCon Workshop Runbook

Database Setup and Various Helpful Commands (on Mac)

Create 3 node cluster:

```
./yugabyted-setup-scripts/yb_setup_all_mac.sh
```

The above command will download YugabyteDB and place it in the directory `$HOME/yb`

Create TRANSACTIONS table:

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 127.0.0.1 -f
./yugabyted-schema-scripts/init_tables.sql

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 127.0.0.1 -c "\d+"
```

Connect to database (Just an example):

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 127.0.0.1
```

List database nodes (Just an example):

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 127.0.0.1 -c "SELECT host,
cloud, region, zone FROM yb_servers() ORDER BY host;"
```

The following commands scripts are available and may be used later in the workshop:

Stop one node

QCon Workshop Runbook

```
./yugabyted-setup-scripts/yb_stop_one_mac.sh nodex
```

Restart one node

```
./yb-qcon-workshop/yugabyted-setup-scripts/yb_start_one_mac.sh
nodex
```

Stop all nodes

```
./yugabyted-setup-scripts/yb_stop_all_mac.sh
```

Restart all nodes

```
./yugabyted-setup-scripts/yb_start_all_mac.sh
```

Destroy one node

```
./yugabyted-setup-scripts/yb_destroy_one_mac.sh nodex
```

Destroy all nodes

```
./yugabyted-setup-scripts/yb_destroy_all_mac.sh
```

Add three nodes

```
./yugabyted-setup-scripts/yb_add_three_mac.sh
```

Stop three nodes

```
./yugabyted-setup-scripts/yb_stop_three_mac.sh
```

Destroy three nodes

```
./yugabyted-setup-scripts/yb_destroy_three_mac.sh
```

Database Setup and Various Helpful Commands (on Cloud node)

Use this option if you do not want to create a local YugabyteDB cluster

QCon Workshop Runbook

** When running the examples, please change userx to your assigned database name (i.e. user1, user2, etc.)

Install YugabyteDB YSQLSH client (on your Mac)

```
mkdir $HOME/yb
cd $HOME/yb
curl -sSL https://downloads.yugabyte.com/get_clients.sh | bash
```

For additional methods, refer to the YSQLSH install doc page.

Create TRANSACTIONS table

```
$HOME/yb/yugabyte-client-2.16.0.1/bin -h 13.57.241.79 -d userx -f
./yugabyted-schema-scripts/init_tables.sql
```

Connect to database

```
$HOME/yb/yugabyte-client-2.16.0.1/bin/ysqlsh -h 13.57.241.79 -d
userx
```

List database nodes

```
$HOME/yb/yugabyte-client-2.16.0.1/bin/ysqlsh -h 13.57.241.79 -d
userx -c "SELECT host, cloud, region, zone FROM yb_servers() ORDER
BY host;"
```

QCon Workshop Runbook

aSynchronous replication with xCluster

Doc Reference: xCluster (2+ regions)

We will be replicating from a DB in AWS to another DB is AWS

The instructors will configure two new databases prior to starting this exercise.

EC2 Instance	Private IP	Public IP
yb-qcon-db-node1	10.30.10.103	13.57.241.79
yb-qcon-db-node2	10.30.14.76	54.193.34.76

** When running the examples, please change userx to your assigned database name (i.e. user1, user2, etc.) **

Verify remote cluster 1 is available:

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 13.57.241.79 -c "SELECT
host, cloud, region, zone FROM yb_servers() ORDER BY host;"
```

Verify remote cluster 2 is available:

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 54.193.34.76 -c "SELECT
host, cloud, region, zone FROM yb_servers() ORDER BY host;"
```

Note: Each cluster should have only a single node. In a production deployment each database would have at least three nodes.

Create the TRANSACTIONS table on remote cluster 1:

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 13.57.241.79 -d userx -f
./yugabyted-schema-scripts/init_tables.sql
```

QCon Workshop Runbook

```
$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 13.57.241.79 -d userx -c
"\d+"
```

Create the TRANSACTIONS table on remote cluster 2:

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 54.193.34.76 -d userx -f
./yugabyted-schema-scripts/init_tables.sql

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 54.193.34.76 -d userx -c
"\d+"
```

Get the Universe UUID of remote cluster 2 (This is the TARGET database):

```
curl -s http://54.193.34.76:7000 | grep "Universe UUID" | sed -e
's/<[^>]*>/|/g' | awk -F"|" '{ print $9}'
```

Or…

```
curl -s http://54.193.34.76:7000/varz\?raw | grep
"\-\-cluster_uuid" | awk -F"=" '{ print $2 }'
```

Get the Table ID of the TRANSACTIONS table on remote cluster 1 (SOURCE):

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/ysqlsh -h 13.57.241.79 -d userx -Atc
"SELECT '0000' || lpad(to_hex(d.oid::int), 4, '0') ||
'00003000800000000000' || lpad(to_hex(c.oid::int), 4, '0') FROM
pg_class c, pg_namespace n, pg_database d WHERE n.nspname =
'public' AND c.relname = 'transactions' AND c.relnamespace = n.oid
AND d.datname = current_database();"
```

The command to set up xCluster replication has this format:

```
yb-admin \
-master_addresses <target_master_addresses> \
setup_universe_replication \
<source_universe_uuid>_<replication_name> \
<source_master_addresses> \
<comma_separated_list_of_table_ids> \
[ <comma_separated_list_of_producer_bootstrap_ids> ] \
[ transactional ]
```

We've gathered all of the necessary info above.

Set up xCluster asynchronous replication from remote cluster 1 (SOURCE) to remote cluster 2 (TARGET)... This is an example:

```
BASEDIR=$HOME/yb

$BASEDIR/yugabyte-2.19.2.0/bin/yb-admin \
-master_addresses 54.193.34.76:7100 \
setup_universe_replication \
bebaa653-42f0-4424-95a9-6a9bb69856e5_transactions \
13.57.241.79:7100 \
000040000000300080000000000402a
```

After running the above command you should receive the message “Replication setup successfully”

Now you can point the app at remote cluster 1, and observe that data is being replicated to remote cluster 2.

QCon Workshop Runbook

Workshop 1 - Workload Service Implementation

Objective

- We will create a spring boot project called Workload Service
- Code some REST API endpoints.
- Implement the code to create database tables, simulations.
- No UI knowledge is needed for this activity. We have included the prebuilt UI components and all the changes will be on the Java side.
- By the end of this session, you will have a simulator up and running.
- We will run some workload against YugabyteDB.
- Then we will stop some nodes to test the resiliency of the App and database.
- Last, we scale out our database cluster from 3 nodes to 6 nodes and see the effect on our App.

Lab 1 - Setting Coding Environment

Ensure you have JDK 17 installed on your machine. You can check the java version by running below command:

```
Unset
java -version
```

If it is missing, you can download it from here:
https://docs.aws.amazon.com/corretto/latest/corretto-17-ug/downloads-list.html

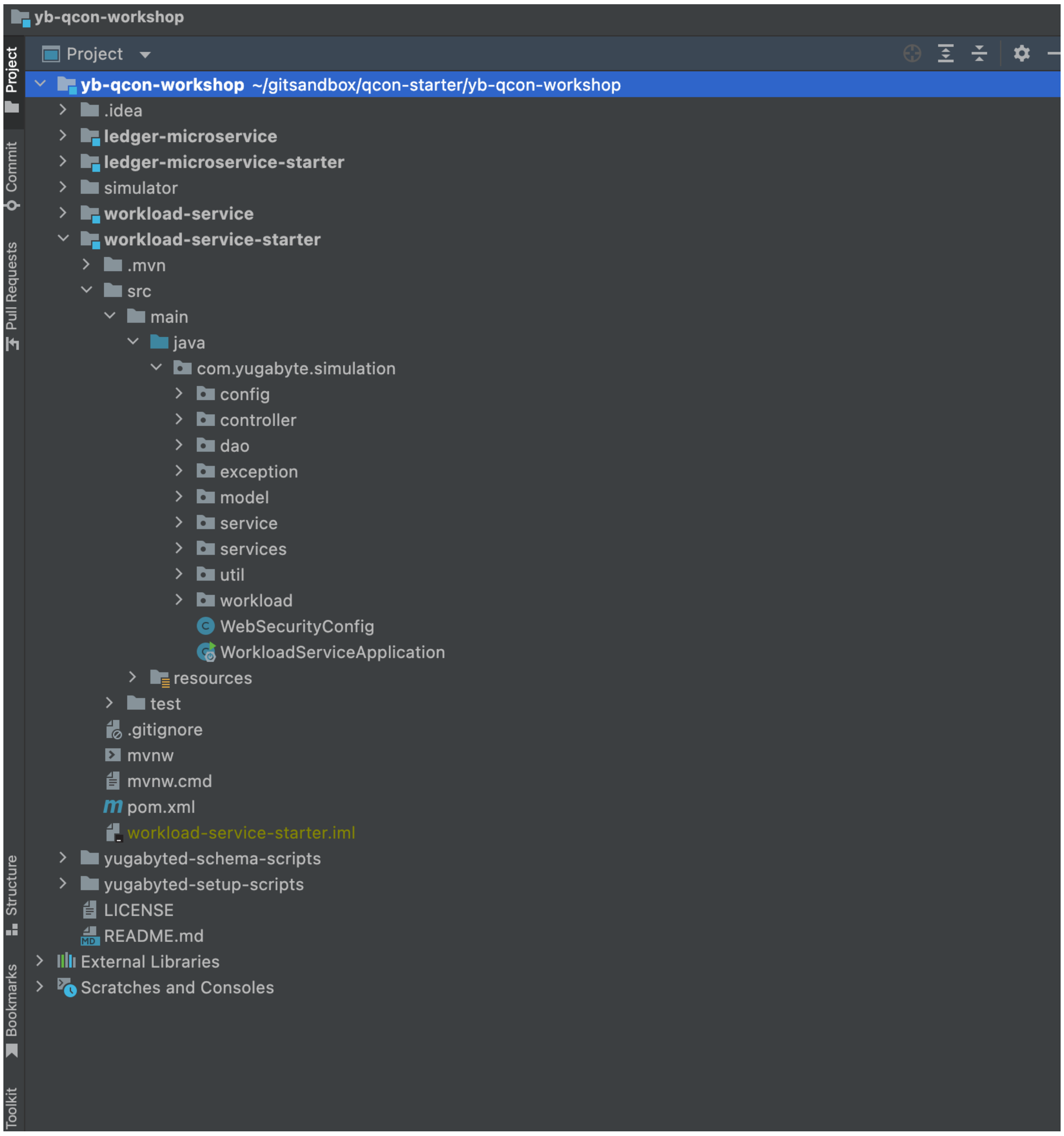
We recommend using Intellij. You can download the community version from here:
https://www.jetbrains.com/edu-products/download/other-IIE.html

Let's clone the project from github onto your local machine (you might already have completed this)

```
Unset
mkdir ~/gitsandbox
cd ~/gitsandbox
git clone https://github.com/akscjo/yb-qcon-workshop.git
```

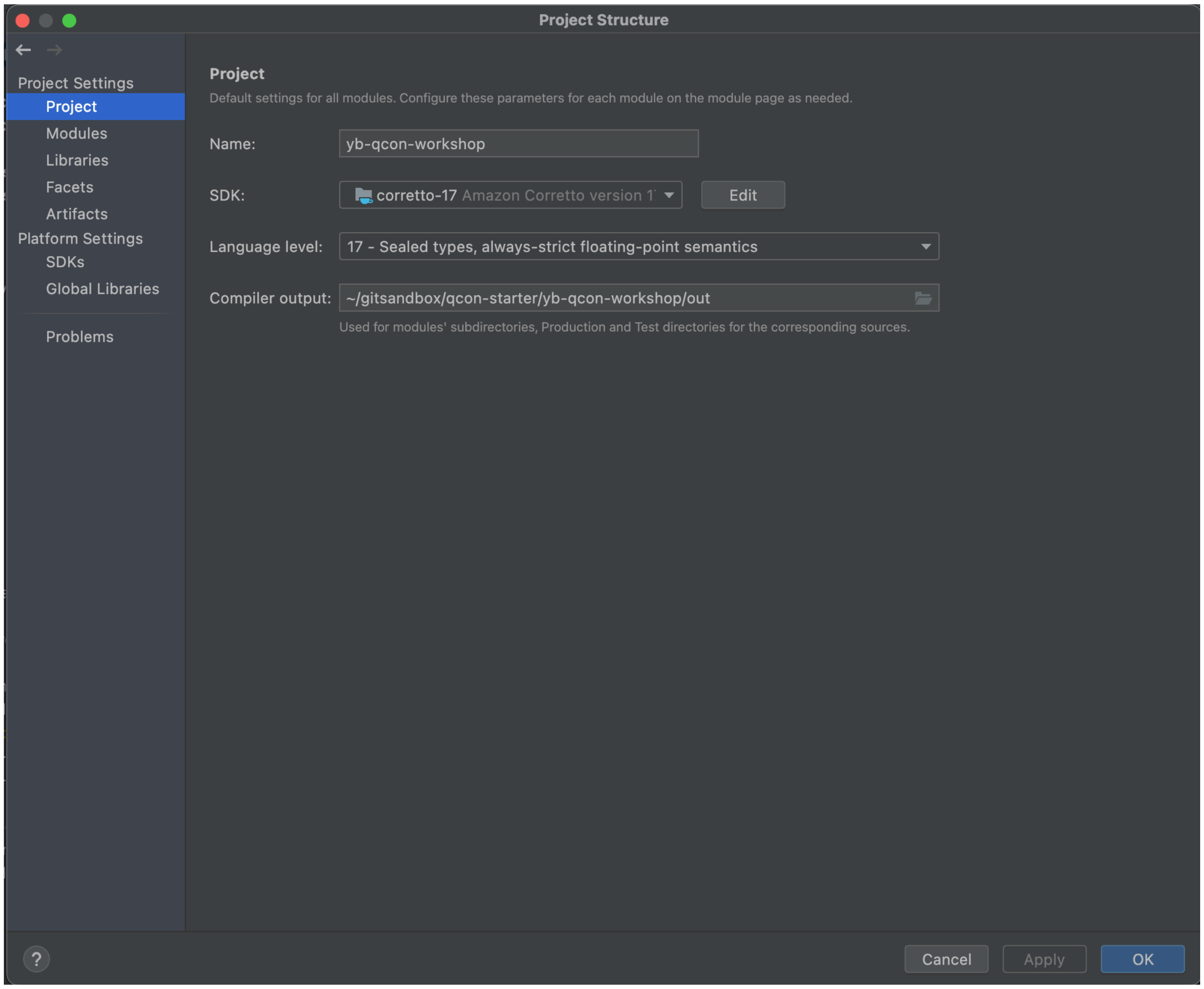
QCon Workshop Runbook

In Intellij or your favorite IDE, open the yb-qcon-workshop project from the directory you have downloaded the github repository.



```
yb-qcon-workshop
Project
yb-qcon-workshop ~/gitsandbox/qcon-starter/yb-qcon-workshop
  .idea
  ledger-microservice
  ledger-microservice-starter
  simulator
  workload-service
  workload-service-starter
    .mvn
    src
      main
        java
          com.yugabyte.simulation
            config
            controller
            dao
            exception
            model
            service
            services
            util
            workload
            WebSecurityConfig
            WorkloadServiceApplication
        resources
      test
    .gitignore
    mvnw
    mvnw.cmd
    pom.xml
    workload-service-starter.iml
  yugabyted-schema-scripts
  yugabyted-setup-scripts
  LICENSE
  README.md
External Libraries
Scratches and Consoles
```

Check the File -> Project Structure to ensure your Jdk is properly picked up by Intellij and there are no errors:



Project
Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: yb-qcon-workshop

SDK: corretto-17 Amazon Corretto version 1

Language level: 17 - Sealed types, always-strict floating-point semantics

Compiler output: ~/gitsandbox/qcon-starter/yb-qcon-workshop/out
Used for modules' subdirectories, Production and Test directories for the corresponding sources.

We will be working on “**workload-service-starter**” during this workshop.

Speakers will walk you through the code structure before we start coding.

Lab 2 - setting up the configuration stuff

First we will inspect the application.yaml file and make the changes in this file.

Lookup this file: **workload-service-starter/src/main/resources/application.yaml**

workload-service-starter/src/main/resources/application.yaml

Java

```
spring:
 workload: ${workload:exampleWorkload}
 application:
  name: ${SPRING_APPLICATION_NAME:}
 jpa:
  open-in-view: false
 flyway:
  enabled: false
 datasource:
  hikari:
   username: ${dbuser:yugabyte}
   password: ${dbpassword:yugabyte}
   connection-init-sql: 'set yb_read_from_followers to true;'
   maximumPoolSize: ${max-pool-size:10}
   maxLifeTime: ${max-life-time:600000}
   data-source-class-name: com.yugabyte.ysql.YBClusterAwareDataSource
   data-source-properties:
    serverName: ${node:127.0.0.1}
    portNumber: ${port:5433}
    databaseName: ${dbname:yugabyte}
    topologyKeys: "aws.us-west-2.*"
    #ssl: ${ssl:false}
    #sslmode: ${sslmode:disable}
    #sslrootcert: ${sslrootcert:~/.ssh/ybcloudcert/root.crt}

 data:
  cassandra:
   local-datacenter: ${datacenter_c:DC1}
   port: ${port_c:9042}
   contact-points: ${node_c:127.0.0.1}
   userid: ${userid_c:cassandra}
   password: ${password_c:yugabyte}
   sslcertpath: ${sslcertpath_c:NA}

logging.level:
 root: ERROR
 java.sql: ERROR
 com.zaxxer.hikari: TRACE
 com.yugabyte: ERROR
 com.yugabyte.simulation.workload: ERROR
 org.springframework.jdbc.core: ERROR

server:
 port: 9090
```

QCon Workshop Runbook

We are going to use a 3 node YugabyteDB cluster for this Lab installed locally.
- In case you wish to change any database properties, you can edit them under spring.datasource.hikari section.
- By default, the UI will run on port 9090. If you have something else running on 9090, you can change the port by editing server.port value.

Now let's set up our code to read the database properties from this application.yaml file. You will go to **workload-service-starter/src/main/java/com/yugabyte/simulation/config/DatasourceConfig.java**

```java
@Configuration
public class DatasourceConfig {
  @Bean
  @ConfigurationProperties(prefix = "spring.datasource.hikari")
  public HikariConfig hikariConfig() {
    // TODO: implement this
    return null;
  }

  @Bean
  @Primary // may not be required
  public DataSource dataSource() {
    //TODO: implement this
    return null;
  }
}
```

Let's add our implementation:

```java
@Configuration
public class DatasourceConfig {
 @Bean
 @ConfigurationProperties(prefix = "spring.datasource.hikari")
 public HikariConfig hikariConfig() {
   HikariConfig config = new HikariConfig();
   return config;
 }

 @Bean
 @Primary // may not be required
 public DataSource dataSource() {
   HikariDataSource ds = new HikariDataSource(hikariConfig());
   return ds;
 }
}
```

Now let's define the Workload we are going to create. Navigate to **workload-service-starter/src/main/java/com/yugabyte/simulation/config/WorkloadConfig.java**

And implement your code changes - something like this:

```java
@Configuration
public class WorkloadConfig {
  @Bean(name="ExampleWorkload")
  public WorkloadSimulation quikShipWorkload(){
    return new ExampleWorkload();
  }

  @Bean(name="QconWorkload")
  public WorkloadSimulation qconWorkload(){
    return new QconWorkload();
  }
}
```

Lab 3 - Let’s create some REST API endpoints.

We will write a new controller class:
workload-service-starter/src/main/java/com/yugabyte/simulation/controller/YBServerInfoController.java

We will add two REST API endpoints in this controller. When a client calls these endpoints, we will make a database call to pull the data from yb_servers(). Think of this as database cluster topology data. We are going to use this data later to draw our cluster topology on UI.

Before we code the controller, lets implement our sql query first.

Go to this DAO file:
workload-service-starter/src/main/java/com/yugabyte/simulation/dao/YBServerInfoDAOImpl.java

And implement the SQL query - something like this:

```java
  @Override
  public List<YBServerModel> getAll() {
       String query = "select host,port,cloud,region,zone from yb_servers()";
    return jdbcTemplate.query(query, new
BeanPropertyRowMapper<YBServerModel>(YBServerModel.class));
  }
```

Once that is done, let’s call this method from our controller:

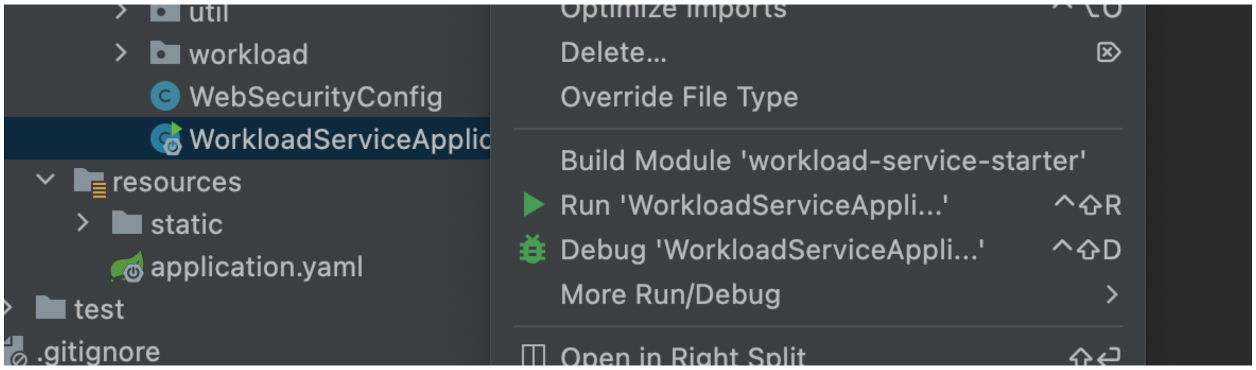
workload-service/src/main/java/com/yugabyte/simulation/controller/YBServerInfoController.java

```java
 @GetMapping("/api/ybserverinfo")
  public List<YBServerModel> getYBServerInfo(){
       return ybServerInfoDAO.getAll();
  }

  @GetMapping("/api/ybserverinfo/{target}")
```

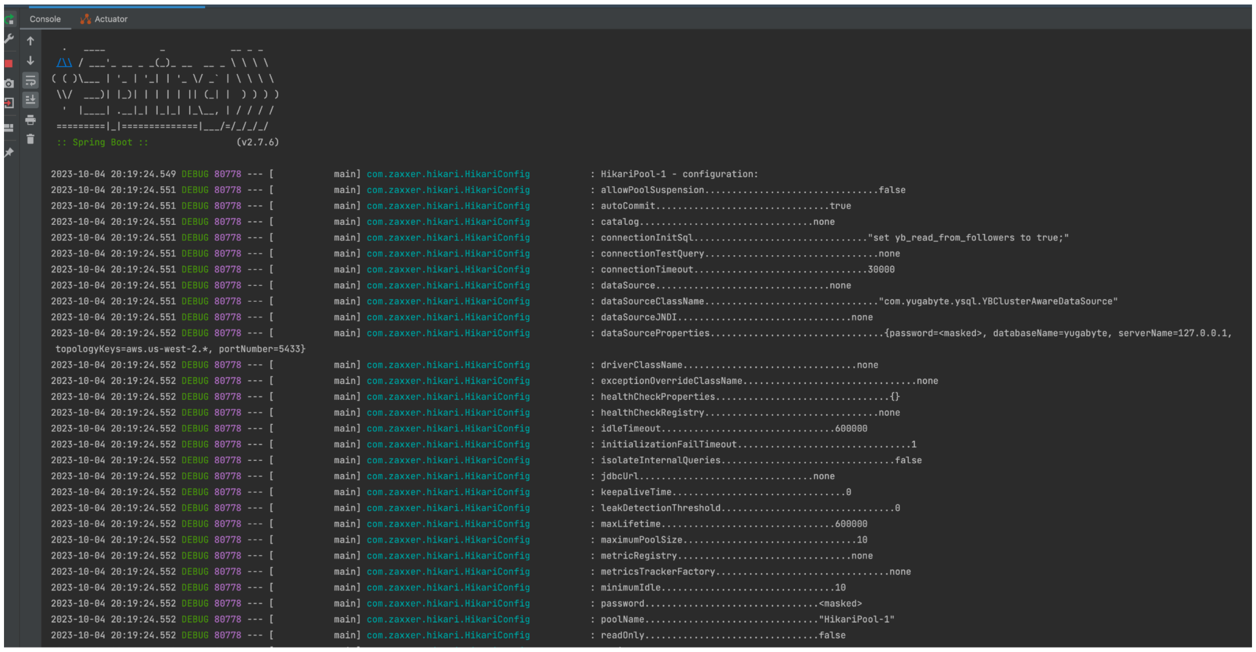
QCon Workshop Runbook

```java
  public List<YBServerModel> getYBServerInfo(@PathVariable(name = "target",
required = true) String target){
        return ybServerInfoDAO.getAll();
  }
```

You have successfully coded these two endpoints. At this point, you can start the Application by right clicking on WorkloadServiceApplication and choose “Run”



This should bring up embedded tomcat and start your application:



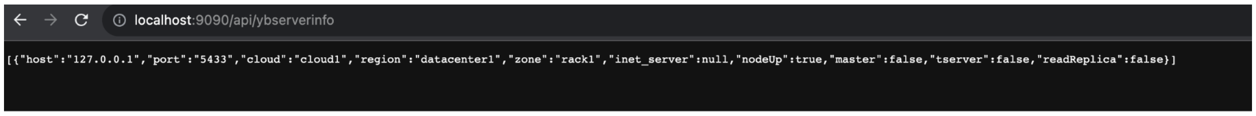
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.6)

2023-10-04 20:19:24.549 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : HikariPool-1 - configuration:
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : allowPoolSuspension.............................false
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : autoCommit................................true
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : catalog................................none
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionInitSql...............................“set yb_read_from_followers to true;"
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionTestQuery.............................none
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionTimeout...............................30000
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSource................................none
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceClassName.............................“com.yugabyte.ysql.YBClusterAwareDataSource"
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceJNDI................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceProperties............................{password=<masked>, databaseName=yugabyte, serverName=127.0.0.1,
 topologyKeys=aws.us-west-2.*, portNumber=5433}
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : driverClassName................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : exceptionOverrideClassName.......................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : healthCheckProperties...........................{}
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : healthCheckRegistry.............................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : idleTimeout................................600000
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : initializationFailTimeout.........................1
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : isolateInternalQueries..........................false
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : jdbcUrl................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : keepaliveTime................................0
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : leakDetectionThreshold..........................0
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : maxLifetime................................600000
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : maximumPoolSize...............................10
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : metricRegistry................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : metricsTrackerFactory...........................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : minimumIdle................................10
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : password................................<masked>
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : poolName................................"HikariPool-1"
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : readOnly................................false
```

Go to your web browser and test the REST call:

QCon Workshop Runbook

```
Unset
http://localhost:9090/api/ybserverinfo
```

You should see the output showing your database cluster in json format:



localhost:9090/api/ybserverinfo

```
[{"host":"127.0.0.1","port":"5433","cloud":"cloud1","region":"datacenter1","zone":"rack1","inet_server":null,"nodeUp":true,"master":false,"tserver":false,"readReplica":false}]
```

You don't need to go to browser and try this with curl as well:

```
Unset
❯ curl http://localhost:9090/api/ybserverinfo

[{"host":"127.0.0.1","port":"5433","cloud":"cloud1","region":"datacenter1","zon
e":"rack1","inet_server":null,"nodeUp":true,"master":false,"tserver":false,"rea
dReplica":false}]%
```

You have successfully completed Lab 3!

Lab 4 - Setting up workload simulator service

Now we will code this service called **workload-service-starter/src/main/java/com/yugabyte/simulation/service/ExampleWorkload.java**

This is not a controller class per se. What we have done as part of this project is that we have built several utilities which enable us to drop any new workload simulation service by just adding a new Java class - for example: ExampleWorkload.java.

We will walk you through the structure of this Java file.

Ok, Now let's get started with the coding. We will first define our database table schema for the tables. You will provide the Display name for your Application and then write queries for creating the tables, dropping the tables and truncating them.

QCon Workshop Runbook

Java
```java
 @Override
  public String getName() {
    return "QCon Example Workload";
  }
```

Java
```java
 private static final String CREATE_PRODUCT_TYPE = "CREATE TYPE  product_type_enum
AS ENUM ('book', 'technology');";

  private static final String CREATE_PRODUCTS = "CREATE TABLE if not exists
products(\n" +
      "    id SERIAL PRIMARY KEY,\n" +
      "    title VARCHAR(255),\n" +
      "    author VARCHAR(255),\n" +
      "    imageLink VARCHAR(255),\n" +
      "    price decimal(12, 2),\n" +
      "    product_type product_type_enum\n" +
      ");";

  private static final String CREATE_ORDERS = "CREATE TABLE if not exists
orders(\n" +
      "    id SERIAL PRIMARY KEY,\n" +
      "    total decimal(12, 2),\n" +
      "    products VARCHAR(255)\n" +
      ");";



  private static final String DROP_PRODUCT_TYPE = "drop type if exists
product_type_enum;";
  private static final String DROP_PRODUCTS = "drop table if exists products
cascade;";
  private static final String TRUNCATE_PRODUCTS = "truncate table products;";
  private static final String DROP_ORDERS = "drop table if exists orders cascade;";
  private static final String TRUNCATE_ORDERS = "truncate table orders;";

  private static final String INSERT_RECORD_ORDERS = "insert into
orders(total,products) values(?,?);";
```

```
  private final String POINT_SELECT_QUERY_ORDERS = "select id, total, products
from orders where id = ?;";
```

Then we will create different workload types. These will show up on UI:

```java
  @Override
  public List<WorkloadDesc> getWorkloads() {
    return Arrays.asList(
        createTablesWorkload
        , seedingWorkload
        , simulationFixedWorkload
        , runningWorkload
    );
  }
```

This method is already coded but let’s look at this. Based on UI action by user, we will be invoking one of the workloads:

```java
 @Override
  public InvocationResult invokeWorkload(String workloadId, ParamValue[] values) {
    WorkloadType type = WorkloadType.valueOf(workloadId);
    try {
      switch (type) {
        case CREATE_TABLES:
          this.createTables();
          return new InvocationResult("Ok");
        case SEED_DATA:
          this.seedData(values[0].getIntValue(), values[1].getIntValue());
          return new InvocationResult("Ok");
        case RUN_SIMULATION:
          this.runSimulation(values);
          return new InvocationResult("Ok");
        case RUN_SIMULATION_FIXED_WORKLOAD:
          this.runSimulationFixedWorkload(values);
          return new InvocationResult("Ok");
      }
      throw new IllegalArgumentException("Unknown workload "+ workloadId);
    }
    catch (Exception e) {
      return new InvocationResult(e);
    }
  }
```

For initiating a simulation, we will look at the user inputs and start multiple threads to perform the queries against the database:

```Java
  private void runSimulationFixedWorkload(ParamValue[] values) {
    int numOfInvocations = values[0].getIntValue();
    int maxThreads = values[1].getIntValue();
    boolean runInserts = values[2].getBoolValue();
    seedingWorkloadType
        .createInstance(serviceManager)
        .execute(maxThreads, numOfInvocations, (customData, threadData) -> {
          int id = LoadGeneratorUtils.getInt(1,ROWS_TO_PRELOAD);;
          runPointReadOrders(id);
          if(runInserts){
            runInserts();
          }
          return threadData;
        });
  }
```

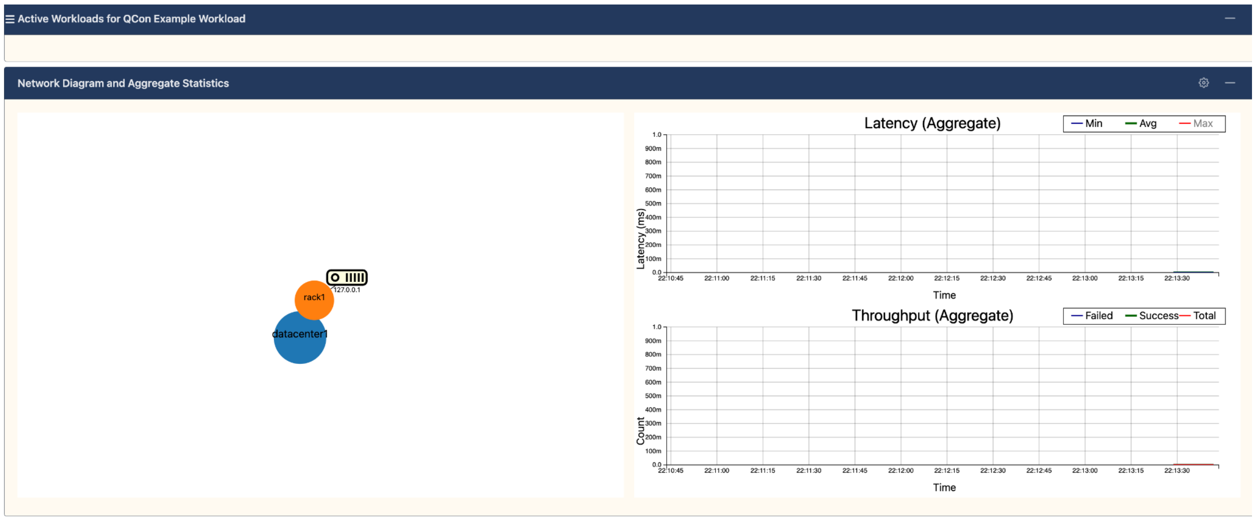
We will coding following two methods to plugin our database table reads and writes. For example:

```java
  private void runPointReadOrders(int id){
    String query = POINT_SELECT_QUERY_ORDERS;
    jdbcTemplate.query(query, new Object[] {id}, new int[] {Types.INTEGER},
        new RowCallbackHandler() {
          @Override
          public void processRow(ResultSet rs) throws SQLException {
//                                    System.out.printf("id=%s, col1='%s',
col2=%s \n",
//                                             rs.getString("id"),
//                                             rs.getString("total"),
//                                             rs.getString("products")
//                                     );
          }
        });
  }


  private void runInserts(){
    UUID uuid = LoadGeneratorUtils.getUUID();
    jdbcTemplate.update(INSERT_RECORD_ORDERS,
        LoadGeneratorUtils.getDouble(1.00,1000.00),
        LoadGeneratorUtils.getText(10,40)
    );
  }
```

Now, the backend code is complete. Let’s start the WorkloadServiceApplication.java

```
http://localhost:9090/
```

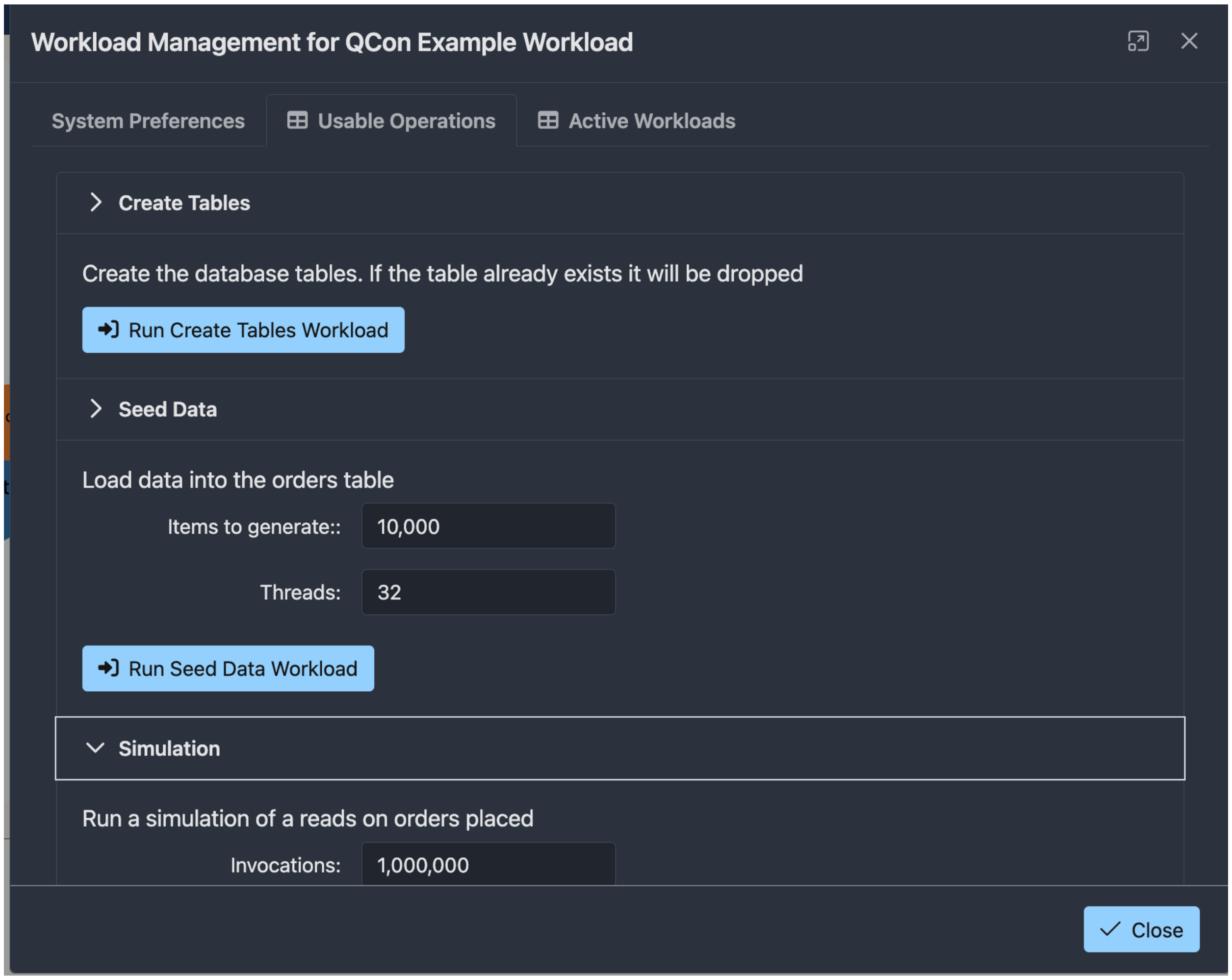
QCon Workshop Runbook



You will see a database network diagram. You will also notice there are two charts showing Latency and Throughput. These charts are going to show us the results of our simulations.

We will click on the hamburger on left hand side and see the options we have coded:

QCon Workshop Runbook


Workload Management for QCon Example Workload

Create Tables

Create the database tables. If the table already exists it will be dropped

Seed Data

Load data into the orders table

Items to generate::	10,000
Threads:	32

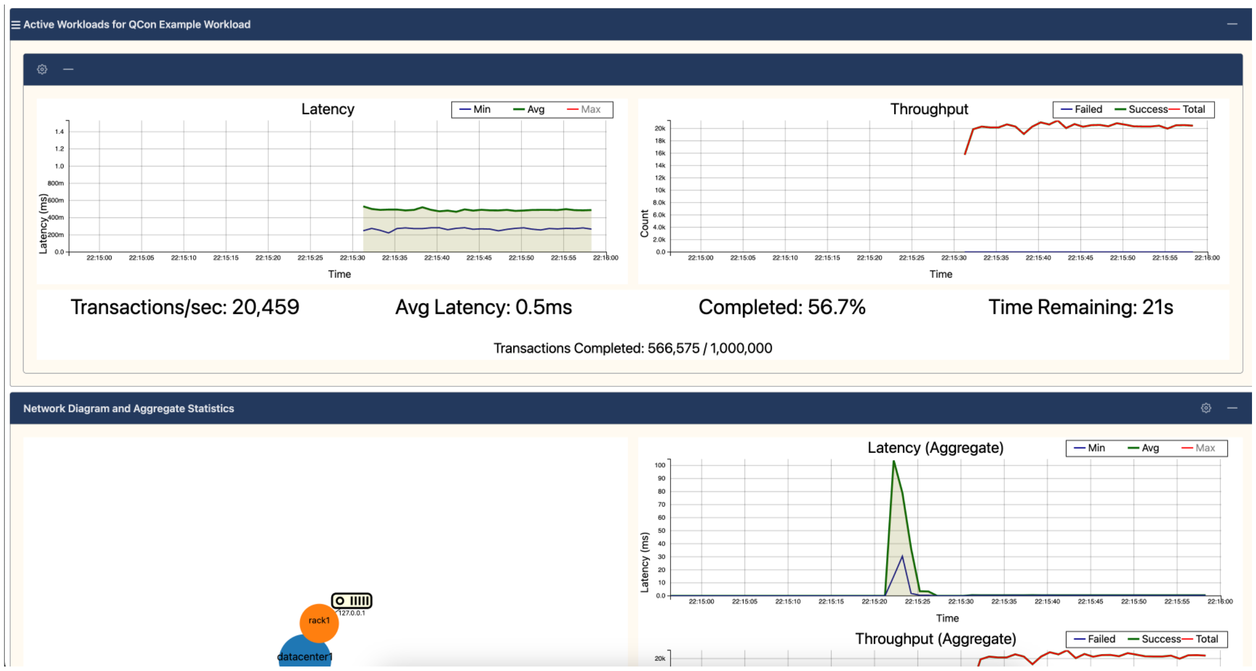
Simulation

Run a simulation of a reads on orders placed

Invocations:	1,000,000

We will now do following items from the UI:
1. Create tables
2. Load some data.
3. Start the workload simulation.

You will see the data starting to popup in the charts.



Transactions/sec: 20,459 Avg Latency: 0.5ms Completed: 56.7% Time Remaining: 21s

Transactions Completed: 566,575 / 1,000,000

That's it for this Lab. Look around the UI for the different options it shows. Play around with starting workloads and observe the impact throughput and latencies.

Lab 5 - Resiliency testing

Now with workload running, we want to bring down one of the nodes and see the impact on the application. Scripts for this are present under QCon Workshop Runbook section.

Instructors will explain in depth by showing what goes under the hood when a node goes down in a distributed database.

Lab 6 - Scale up the Cluster from 3 Nodes to 6 Nodes

We will now scale out our cluster from 3 nodes to 6 nodes and observe the results on the App. We will see how data is automatically sharded and moved around by database. Scripts for this are present under QCon Workshop Runbook section

Instructors will explain in depth by showing what goes under the hood when a cluster is scaled.

Congratulations! You did it!!!

Workshop 2 - Yugabank Transactions Monitoring

Objective

- We will create a spring boot project called Ledger MicroService
- Code some REST API endpoints.
- We will build a basic UI to show our transactions.
- No UI knowledge is needed for this activity.
- Components and all the changes will be mostly on the Java side. We will delve into little bit of javascript.
- By the end of this session, you will have some api's coded and also using them to show on UI.
- We will run a prebuilt simulator to call these newly coded transactions.
- Then we will stop some nodes to test the resiliency of the App and database.
- Last, we scale out our database cluster from 3 nodes to 6 nodes and see the effect on our App.

Lab 1 - Setting Coding Environment

Ensure you have JDK 17 installed on your machine. You can check the java version by running below command:

```
Unset
java -version
```

If it is missing, you can download it from here:
https://docs.aws.amazon.com/corretto/latest/corretto-17-ug/downloads-list.html

We recommend using Intellij. You can download the community version from here:
https://www.jetbrains.com/edu-products/download/other-IIE.html

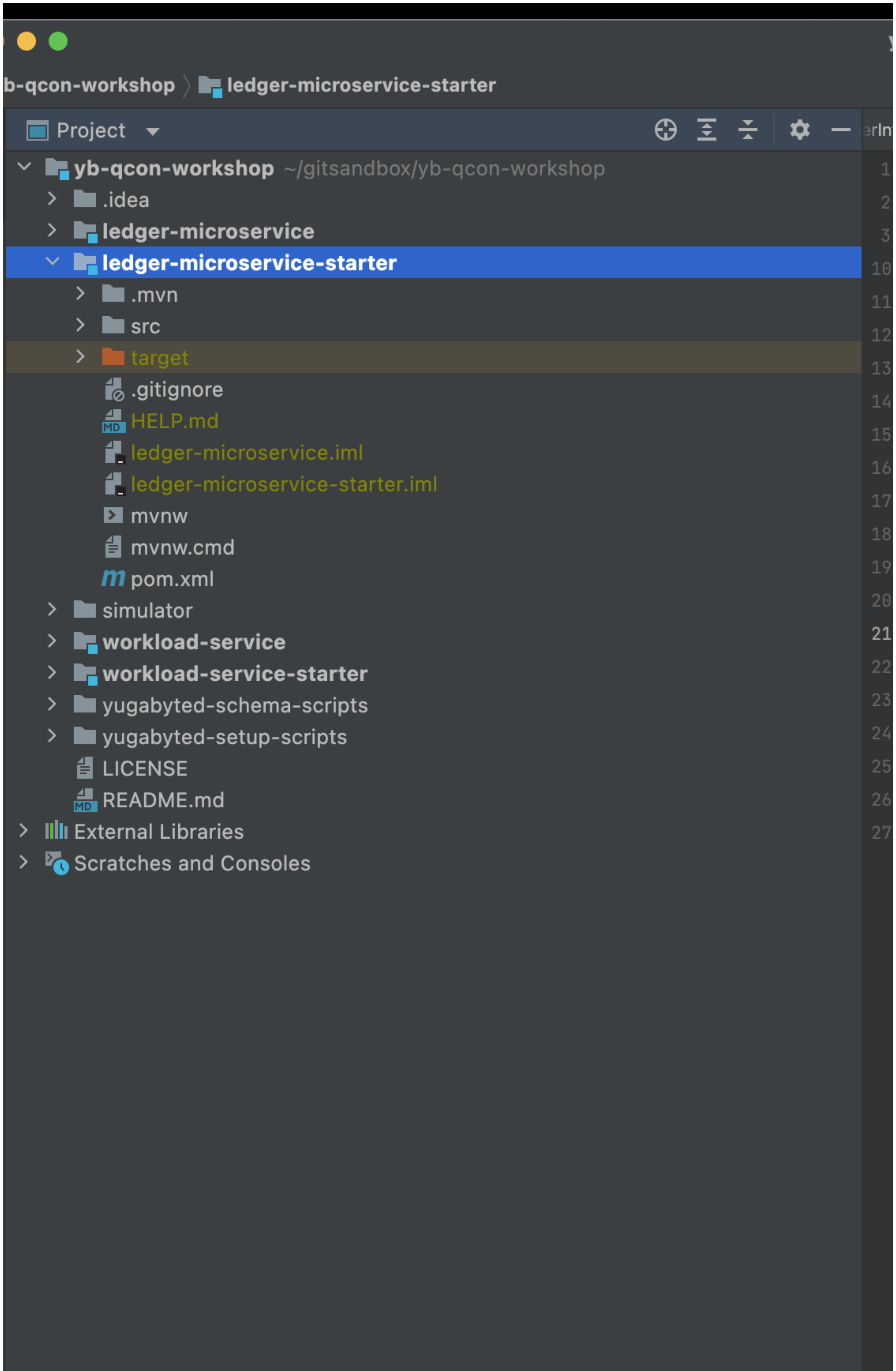
Let's clone the project from github onto your local machine (you might already have completed this)

QCon Workshop Runbook

```
Unset
mkdir ~/gitsandbox
cd ~/gitsandbox
git clone https://github.com/akscjo/yb-qcon-workshop.git
```

In Intellij or your favorite IDE, open the yb-qcon-workshop project from the directory you have downloaded the github repository.

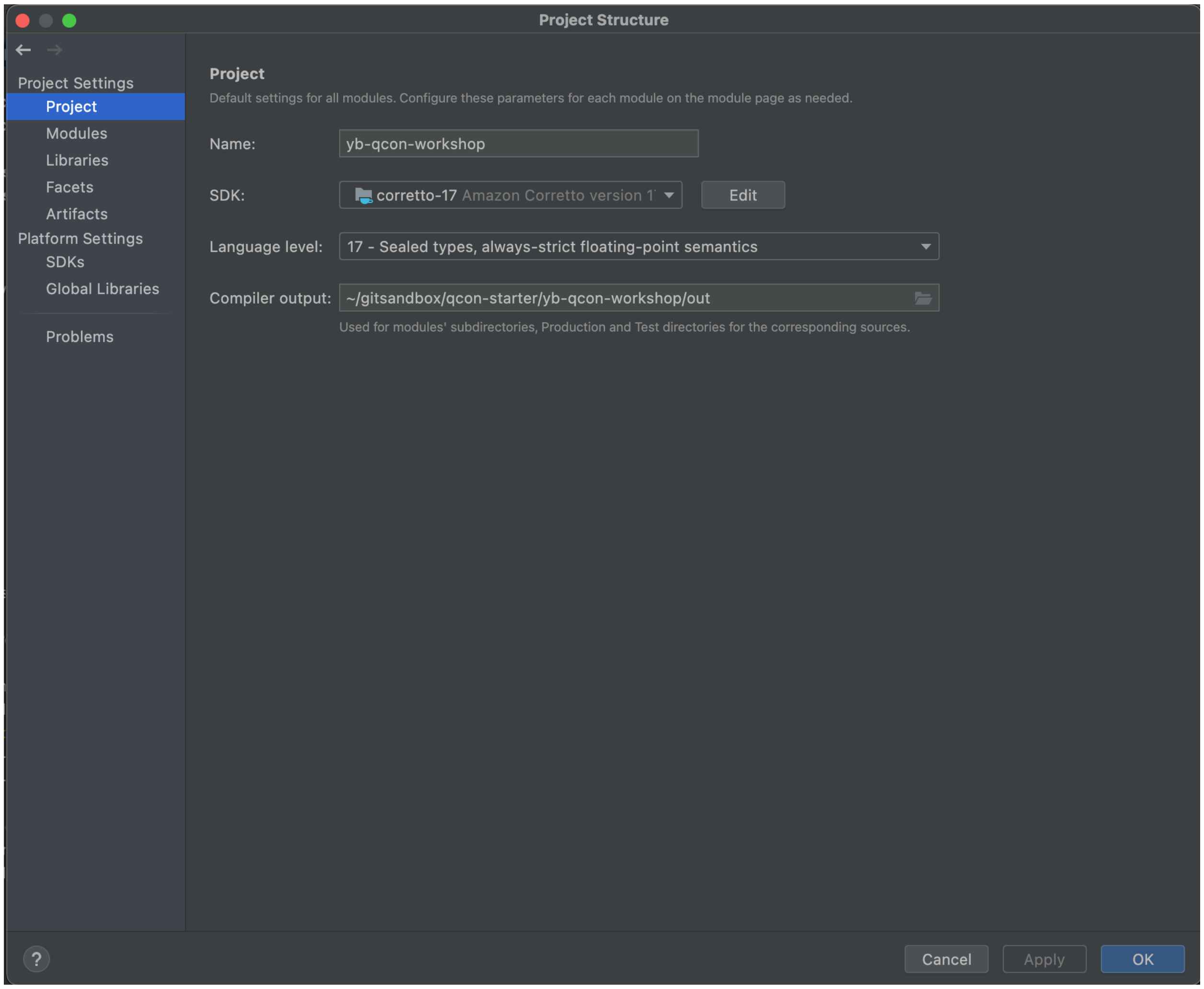
QCon Workshop Runbook



```
b-qcon-workshop 〉 ledger-microservice-starter

Project

yb-qcon-workshop ~/gitsandbox/yb-qcon-workshop
  .idea
  ledger-microservice
  ledger-microservice-starter
    .mvn
    src
    target
    .gitignore
    HELP.md
    ledger-microservice.iml
    ledger-microservice-starter.iml
    mvnw
    mvnw.cmd
    pom.xml
  simulator
  workload-service
  workload-service-starter
  yugabyted-schema-scripts
  yugabyted-setup-scripts
  LICENSE
  README.md
External Libraries
Scratches and Consoles
```

QCon Workshop Runbook

Check the File -> Project Structure to ensure your Jdk is properly picked up by Intellij and there are no errors:



Project

Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: yb-qcon-workshop

SDK: corretto-17 Amazon Corretto version 1

Language level: 17 - Sealed types, always-strict floating-point semantics

Compiler output: ~/gitsandbox/qcon-starter/yb-qcon-workshop/out

Used for modules' subdirectories, Production and Test directories for the corresponding sources.

We will be working on “**ledger-microservice-starter**” during this workshop.

Speakers will walk you through the code structure before we start coding.

Lab 2 - setting up the configuration stuff

First we will inspect the application.yaml file and make the changes in this file.

Lookup this file: **workload-service-starter/src/main/resources/application.yaml**

QCon Workshop Runbook

workload-service-starter/src/main/resources/application.yaml

Java

```
spring:
 jpa:
  open-in-view: false
 flyway:
  enabled: false
 datasource:
  hikari:
   username: ${dbuser:yugabyte}
   password: ${dbpassword:yugabyte}
   connection-init-sql: 'set yb_read_from_followers to true;'
   maximumPoolSize: ${max-pool-size:10}
   maxLifeTime: ${max-life-time:600000}
   data-source-class-name: com.yugabyte.ysql.YBClusterAwareDataSource
   data-source-properties:
    serverName: ${node:127.0.0.1}
    portNumber: ${port:5433}
    databaseName: ${dbname:yugabyte}
    topologyKeys: "cloud.region.*"
    #ssl: ${ssl:false}
    #sslmode: ${sslmode:disable}
    #sslrootcert: ${sslrootcert:~/.ssh/ybcloudcert/root.crt}
logging.level:
 root: ERROR
 java.sql: ERROR
 com.zaxxer.hikari: TRACE
 com.yugabyte: ERROR
 com.yugabyte.simulation.workload: ERROR
 org.springframework.jdbc.core: ERROR

server:
 port: 8080
```

We are going to use a 3 node YugabyteDB cluster for this Lab installed locally.
- In case you wish to change any database properties, you can edit them under spring.datasource.hikari section.
- By default, the UI will run on port 8080. If you have something else running on 8080, you can change the port by editing server.port value.

Now let's set up our code to read the database properties from this application.yaml file. You will go to

workload-service-starter/src/main/java/com/yugabyte/simulation/config/DatasourceConfig.java

Java
```java
@Configuration
public class DatasourceConfig {
  @Bean
  @ConfigurationProperties(prefix = "spring.datasource.hikari")
  public HikariConfig hikariConfig() {
    // TODO: implement this
    return null;
  }

  @Bean
  @Primary // may not be required
  public DataSource dataSource() {
    //TODO: implement this
    return null;
  }
}
```

Let’s add our implementation:

Java
```java
@Configuration
public class DatasourceConfig {
  @Bean
  @ConfigurationProperties(prefix = "spring.datasource.hikari")
  public HikariConfig hikariConfig() {
    HikariConfig config = new HikariConfig();
    return config;
  }

  @Bean
  @Primary // may not be required
  public DataSource dataSource() {
    HikariDataSource ds = new HikariDataSource(hikariConfig());
    return ds;
  }
}
```

Now let's define the Workload we are going to create. Navigate to **workload-service-starter/src/main/java/com/yugabyte/simulation/config/WorkloadConfig.java**

And implement your code changes - something like this:

```java
@Configuration
public class DatasourceConfig {
    @Bean
    @ConfigurationProperties(prefix = "spring.datasource.hikari")
    public HikariConfig hikariConfig() {
        HikariConfig config = new HikariConfig();
        return config;
    }

    @Bean
    @Primary // may not be required
    public DataSource dataSource() {
        HikariDataSource ds = new HikariDataSource(hikariConfig());
        return ds;
    }
}
```

QCon Workshop Runbook

Lab 3 - Let’s create some REST API endpoints.

We will write a new controller class:
ledger-microservice/src/main/java/com/qcon/controller/YBServerInfoController.java

We will add two REST API endpoints in this controller. When a client calls these endpoints, we will make a database call to pull the data from yb_servers(). Think of this as database cluster topology data. We are going to use this data later to draw our cluster topology on UI.

Before we code the controller, let's implement our sql query first.

Go to this DAO file:
ledger-microservice/src/main/java/com/qcon/dao/YBServerInfoDAOImpl.java

And implement the SQL query - something like this:

```java
  @Override
  public List<YBServerModel> getAll() {
       String query = "select host,port,cloud,region,zone from yb_servers()";
    return jdbcTemplate.query(query, new
BeanPropertyRowMapper<YBServerModel>(YBServerModel.class));
  }
```

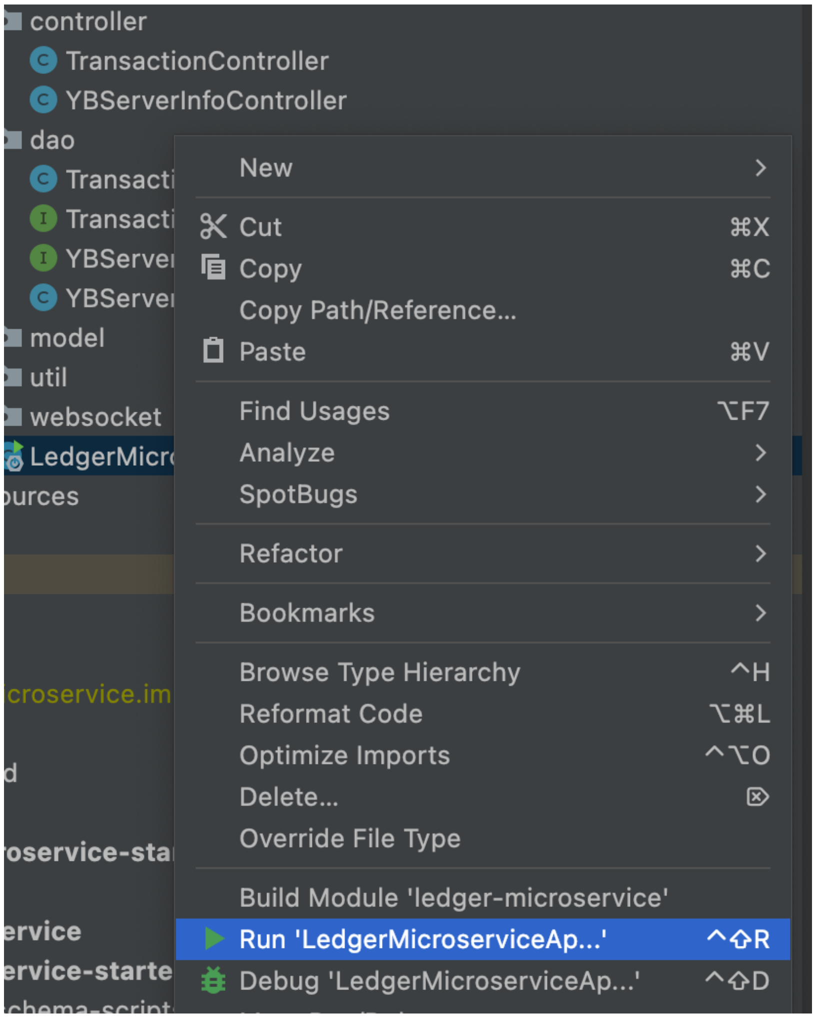
Once that is done, let’s call this method from our controller:

ledger-microservice/src/main/java/com/qcon/dao/YBServerInfoDAOImpl.java

```java
  @GetMapping("/ybserverinfo")
  public List<YBServerModel> getYBServerInfo(){
       return ybServerInfoDAO.getAll();
  }
```

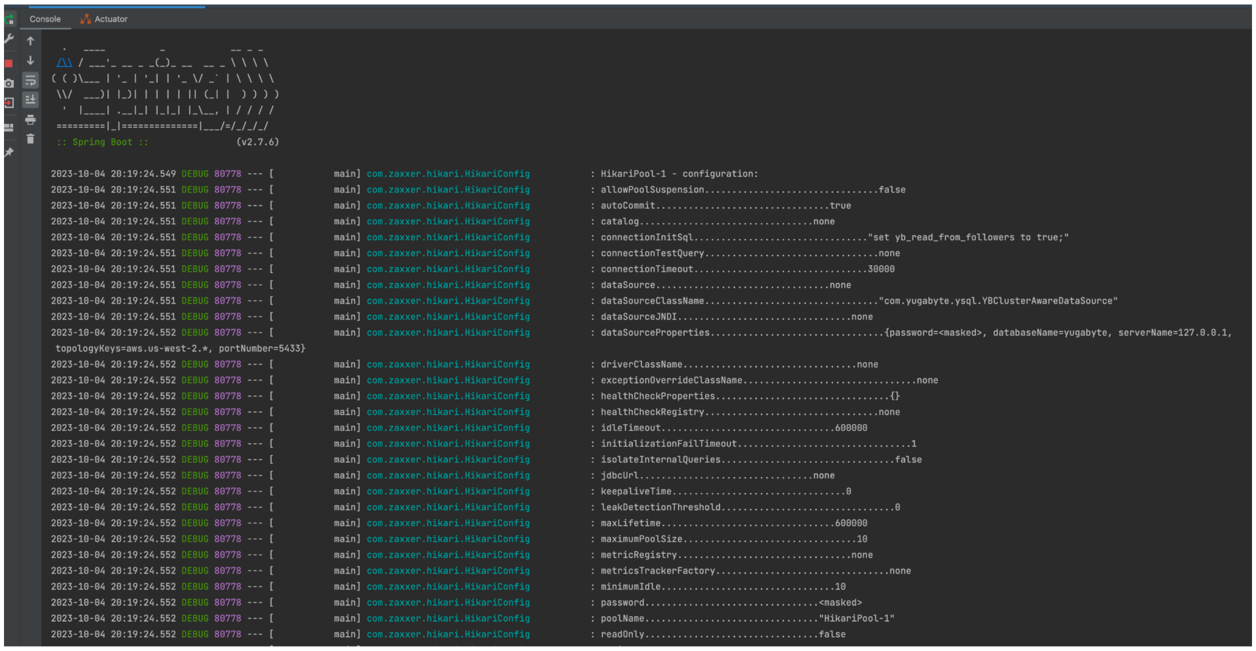
QCon Workshop Runbook

You have successfully coded these two endpoints. At this point, you can start the Application by right clicking on WorkloadServiceApplication and choose “Run”



This should bring up embedded tomcat and start your application:

QCon Workshop Runbook

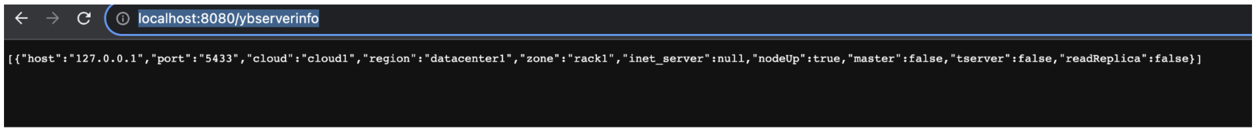


```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.6)

2023-10-04 20:19:24.549 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : HikariPool-1 - configuration:
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : allowPoolSuspension.............................false
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : autoCommit................................true
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : catalog................................none
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionInitSql................................"set yb_read_from_followers to true;"
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionTestQuery................................none
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionTimeout................................30000
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSource................................none
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceClassName................................"com.yugabyte.ysql.YBClusterAwareDataSource"
2023-10-04 20:19:24.551 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceJNDI................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceProperties................................{password=<masked>, databaseName=yugabyte, serverName=127.0.0.1,
 topologyKeys=aws.us-west-2.*, portNumber=5433}
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : driverClassName................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : exceptionOverrideClassName................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : healthCheckProperties................................{}
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : healthCheckRegistry................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : idleTimeout................................600000
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : initializationFailTimeout................................1
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : isolateInternalQueries................................false
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : jdbcUrl................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : keepaliveTime................................0
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : leakDetectionThreshold................................0
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : maxLifetime................................600000
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : maximumPoolSize................................10
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : metricRegistry................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : metricsTrackerFactory................................none
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : minimumIdle................................10
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : password................................<masked>
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : poolName................................"HikariPool-1"
2023-10-04 20:19:24.552 DEBUG 80778 --- [           main] com.zaxxer.hikari.HikariConfig           : readOnly................................false
```

Go to your web browser and test the REST call:

```
Unset
http://localhost:8080/ybserverinfo
```

You should see the output showing your database cluster in json format:



localhost:8080/ybserverinfo

```
[{"host":"127.0.0.1","port":"5433","cloud":"cloud1","region":"datacenter1","zone":"rack1","inet_server":null,"nodeUp":true,"master":false,"tserver":false,"readReplica":false}]
```

You don't need to go to browser and try this with curl as well:

```
Unset
 curl http://localhost:8080/ybserverinfo
[{"host":"127.0.0.1","port":"5433","cloud":"cloud1","region":"datacenter1","zon
e":"rack1","inet_server":null,"nodeUp":true,"master":false,"tserver":false,"rea
dReplica":false}]%
```

You have successfully completed Lab 3!

Lab 4- Setup Ledger Microservice

In this lab we will code
ledger-microservice/src/main/java/com/qcon/controller/TransactionController.java

Instructors will walk you through the list of REST endpoints we need to build here. Each endpoint will make a database call. These methods are defined in the following interface.

```java
public interface TransactionRepositoryInterface {
  List<Transaction> findAll();
  Transaction findById(Long id);
  void save(Transaction transaction);
  void update(Transaction transaction);
  void delete(Long id);
  List<Transaction> getLatestTransactions();
  List<Transaction> findTop10FromAccounts(LocalDateTime timestamp);
  Map<String, Integer> getTransactions(LocalDateTime timestamp);
}
```

Let's implement these methods one by one. Here is the sample implementation for ledger-microservice/src/main/java/com/qcon/dao/TransactionRepository.java

```java
@Repository
public class TransactionRepository implements TransactionRepositoryInterface{
  private final JdbcTemplate jdbcTemplate;

  @Autowired
  public TransactionRepository(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
  }

  @Override
  public List<Transaction> findAll() {
    String sql = "SELECT * FROM transactions limit 10000";
    return jdbcTemplate.query(sql, new
BeanPropertyRowMapper<>(Transaction.class));
```

```
  }

  @Override
  public Transaction findById(Long id) {
    String sql = "SELECT * FROM transactions WHERE transaction_id = ?";
    RowMapper<Transaction> rowMapper = new
BeanPropertyRowMapper<>(Transaction.class);

    List<Transaction> transactions = jdbcTemplate.query(sql, rowMapper, id);

    // Check if any results were returned
    if (!transactions.isEmpty()) {
      return transactions.get(0);
    } else {
      return null; // or throw an exception, depending on your use case
    }
  }

  @Override
  public void save(Transaction transaction) {
    String sql = "INSERT INTO transactions (from_acct, to_acct, from_route,
to_route, amount) " +
        "VALUES (?, ?, ?, ?, ?)";
    jdbcTemplate.update(
        sql,
        transaction.getFromAcct(),
        transaction.getToAcct(),
        transaction.getFromRoute(),
        transaction.getToRoute(),
        transaction.getAmount()
    );
  }

  @Override
  public void update(Transaction transaction) {
    String sql = "UPDATE transactions SET from_acct = ?, to_acct = ?, from_route
= ?, to_route = ?, amount = ?, timestamp = ? " +
        "WHERE transaction_id = ?";
    jdbcTemplate.update(
        sql,
        transaction.getFromAcct(),
        transaction.getToAcct(),
        transaction.getFromRoute(),
        transaction.getToRoute(),
        transaction.getAmount(),
        transaction.getTimestamp(),
        transaction.getTransactionId()
    );
  }

  @Override
  public void delete(Long id) {
    String sql = "DELETE FROM transactions WHERE transaction_id = ?";
    int code = jdbcTemplate.update(sql, id);
    System.out.println("Delete transaction code: " + code);
  }

  @Override
  public List<Transaction> getLatestTransactions() {
    String sql = "SELECT * FROM transactions order by transaction_id desc limit
100";
    return jdbcTemplate.query(sql, new
BeanPropertyRowMapper<>(Transaction.class));
  }

  @Override
  public List<Transaction> findTop10FromAccounts(LocalDateTime timestamp) {
    String sql = "SELECT *\n" +
            "FROM transactions\n" +
            "WHERE \"timestamp\" >= ? \n" +
            "ORDER BY \"amount\" DESC\n" +
            "LIMIT 10;";
    return jdbcTemplate.query(sql, new
BeanPropertyRowMapper<>(Transaction.class), Timestamp.valueOf(timestamp));
  }

  @Override
  public Map<String, Integer> getTransactions(LocalDateTime timestamp) {
    int numSeconds = 60;
    String sql = "SELECT COUNT(*) AS total_transactions " +
        "FROM transactions " +
        "WHERE \"timestamp\" >= ?";

    Timestamp timestampParam =
Timestamp.valueOf(timestamp.minusSeconds(numSeconds)); // Calculate 60 seconds
ago

    int totalTransactions = jdbcTemplate.queryForObject(sql, Integer.class,
timestampParam);
```

```
    Map<String, Integer> response = new HashMap<>();
    response.put("total_transactions", totalTransactions);

    return response;
  }
```

Now we have coded the methods to pull data out of the database, let's call these methods from our controller.

ledger-microservice/src/main/java/com/qcon/controller/TransactionController.java . You will see the empty methods in your class. Instructors will walk you through what each method does.

```
Java
@RestController
@RequestMapping("/transactions")
public class TransactionController {
  private final TransactionRepository transactionRepository;

  @Autowired
  public TransactionController(TransactionRepository transactionRepository) {
    this.transactionRepository = transactionRepository;
  }

  @GetMapping
  public List<Transaction> getAllTransactions() {
    return transactionRepository.findAll();
  }

  @GetMapping("/{id}")
  public Transaction getTransactionById(@PathVariable Long id) {
    return transactionRepository.findById(id);
  }

  @PostMapping("/create")
  public void createTransaction(@RequestBody Transaction transaction) {
    transactionRepository.save(transaction);
  }

  @PutMapping("/update/{id}")
  public void updateTransaction(@PathVariable Long id, @RequestBody Transaction
transaction) {
    transaction.setTransactionId(id);
    transactionRepository.update(transaction);
  }

  @DeleteMapping("/delete/{id}")
  public void deleteTransaction(@PathVariable Long id) {
    System.out.println("Deleting transaction with id: " + id);
    transactionRepository.delete(id);
    System.out.println("Transaction deleted");
  }

  @GetMapping("/latest")
  public List<Transaction> getLatestTransactions() {
    // Retrieve the latest transactions from your database
    return transactionRepository.getLatestTransactions();
  }

  @GetMapping ("/create-random-transaction")
  public ResponseEntity<String> createRandomTransaction() {
    try{
      transactionRepository.save(GeneralUtility.getRandomTransaction());
      return ResponseEntity.ok("Transaction created");
    }
    catch (Exception e){
      return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
          .body("Error creating transaction: " + e.getMessage());
    }

  }

  @GetMapping("/top-10-from-accts")
  public List<Transaction> getTop10FromAccounts() {
    // Define a timestamp for 1 minutes ago
    LocalDateTime oneMinuteAgo = LocalDateTime.now().minusMinutes(1);

    // Call a custom repository method to retrieve the top 10 from_acct based on
total amount
    return transactionRepository.findTop10FromAccounts(oneMinuteAgo);
  }

  @GetMapping("/avg-transactions")
  public Map<String, Integer> getTop10HighValueAccounts() {
    // Define a timestamp for 1 minutes ago
```

```
      LocalDateTime oneMinuteAgo = LocalDateTime.now().minusMinutes(1);
      return transactionRepository.getTransactions(oneMinuteAgo);
    }



  }
```

You are all done with the backend code.

Lab 5 - Setup the UI

We want to call some of the endpoints we have coded from the UI and see the live transactions every second. In this UI we will have few sections:
- show latest 100 transactions
  - Show the top 10 accounts with highest transaction activity in a bar graph.
  - Show the total number of transactions in the last 60 seconds.
  - Display our database cluster topology in a table

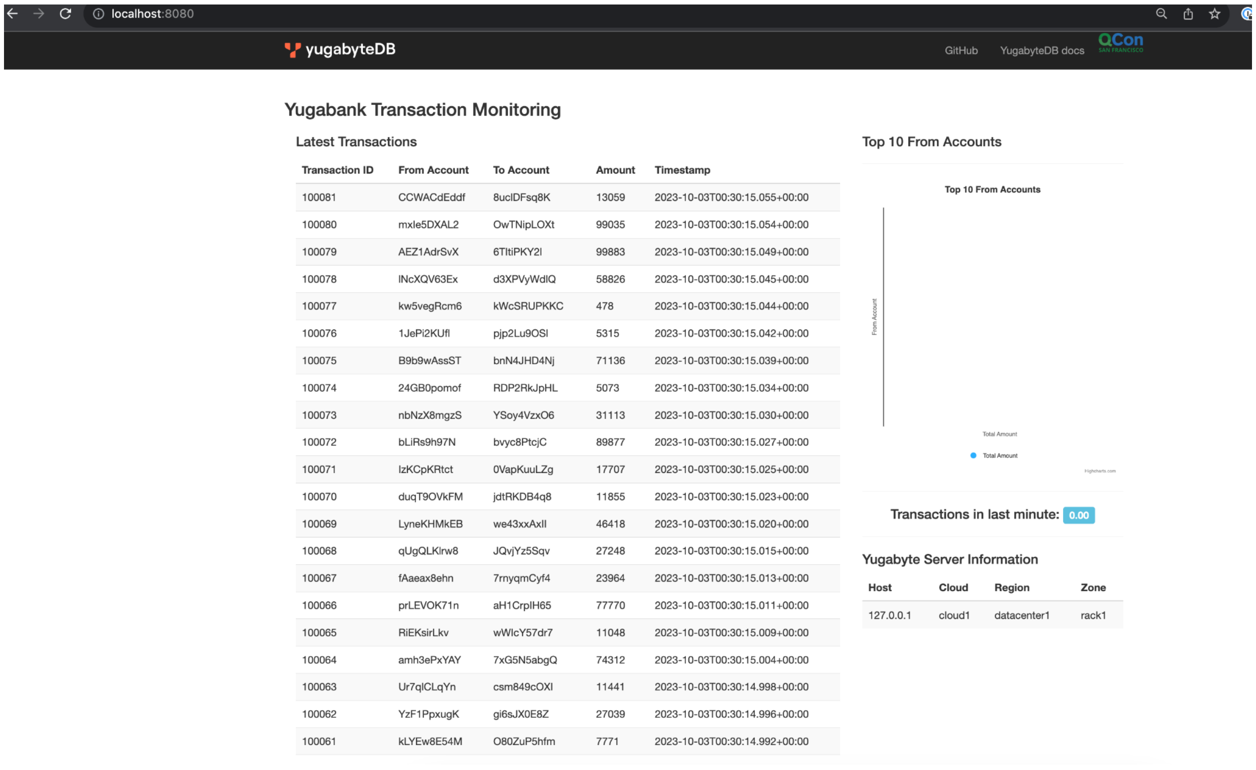
Let's go to the index.html file:
ledger-microservice/src/main/resources/static/index.html

Instructors will walk you through the details of this html file. Then we will code the javascript methods to implement the different API calls every second. You will expect your code to look something like this after you are done through implementation:

```JavaScript
<script>
  function pollLatestTransactions() {
    setInterval(() => {
      // Perform an AJAX GET request to retrieve the latest transactions
      $.get('/transactions/latest', (data) => {
        // Process the received data and update your UI
        updateTransactionUI(data);
      });
    }, 1000); // Poll every 5 seconds (adjust as needed)
  }

  function updateTransactionUI(transactions) {
    // Clear the existing table rows
    $('#transactionTableBody').empty();

    // Loop through the received transactions and append them to the table
    for (const transaction of transactions) {
      const row = `
                    <tr>
                        <td>${transaction.transactionId}</td>
                        <td>${transaction.fromAcct}</td>
                        <td>${transaction.toAcct}</td>
                        <td>${transaction.amount}</td>
                        <td>${transaction.timestamp}</td>
                    </tr>
                `;
      $('#transactionTableBody').append(row);
    }
  }

  // Function to poll top 10 from accounts data
  function pollTop10FromAccounts() {
    setInterval(() => {
      $.get('/transactions/top-10-from-accts', (data) => {
        updateTop10FromAccountsChart(data);
      });
    }, 5000); // Poll every 1 second (adjust as needed)
  }



  // Function to update top 10 from accounts chart
  function updateTop10FromAccountsChart(accounts) {
    const categories = accounts.map((account) => account.fromAcct);
    const data = accounts.map((account) => account.amount);

    // Create or update the chart using the global top10FromAccountsChart
variable
    // if (top10FromAccountsChart) {
    //     //top10FromAccountsChart.destroy(); // Destroy the existing chart if
it exists
    // }

    top10FromAccountsChart = Highcharts.chart('top10FromAccountsChart', {
      chart: {
        type: 'bar'
      },
      title: {
        text: 'Top 10 From Accounts'
      },
      xAxis: {
        categories: categories,
        title: {
          text: 'From Account'
        }
      },
      yAxis: {
        title: {
          text: 'Total Amount'
        }
      },
      series: [{
        name: 'Total Amount',
        data: data
      }]
    });

  }



  // Function to poll Yugabyte Server Information data
  function pollYbServerInfo() {
    setInterval(() => {
      $.get('/ybserverinfo', (data) => {
        updateYbServerInfoUI(data);
      });
    }, 1000); // Poll every 1 second (adjust as needed)
  }

  // Function to update Yugabyte Server Information data in the UI
  function updateYbServerInfoUI(serverInfo) {
    $('#ybServerInfoTable').empty();
    for (const server of serverInfo) {
      const row = `
              <tr>
                  <td>${server.host}</td>
                  <td>${server.cloud}</td>
                  <td>${server.region}</td>
                  <td>${server.zone}</td>
              </tr>
          `;
      $('#ybServerInfoTable').append(row);
    }
  }

  // Function to poll average transactions data
  function pollAverageTransactions() {
    setInterval(() => {
      $.get('/transactions/avg-transactions', (data) => {
        // Ensure data is a number
        const average = parseFloat(data.total_transactions);
        if (!isNaN(average)) {
          updateAverageTransactionsValue(average);
        }
        else{
          updateAverageTransactionsValue(0);
        }
      });
    }, 1000); // Poll every 1 second
  }

  // Function to update the "Transactions/Sec" value
  function updateAverageTransactionsValue(average) {
    $('#transactionsPerSecValue').text(average.toFixed(2));
  }

  // Start polling when the page loads
  $(document).ready(() => {
    pollLatestTransactions();
    pollTop10FromAccounts();
    pollYbServerInfo();
    pollAverageTransactions();
  });
</script>
```

Let’s start the App. You won’t see any data on your UI - That’s ok - since we haven’t inserted any transactions yet. We will do simulation with parallel users through a tool in upcoming lab,

QCon Workshop Runbook



Yugabank Transaction Monitoring

Latest Transactions

Transaction ID	From Account	To Account	Amount	Timestamp
100081	CCWACdEddf	8ucIDFsq8K	13059	2023-10-03T00:30:15.055+00:00
100080	mxle5DXAL2	OwTNipLOXt	99035	2023-10-03T00:30:15.054+00:00
100079	AEZ1AdrSvX	6TltiPKY2l	99883	2023-10-03T00:30:15.049+00:00
100078	lNcXQV63Ex	d3XPVyWdlQ	58826	2023-10-03T00:30:15.045+00:00
100077	kw5vegRcm6	kWcSRUPKKC	478	2023-10-03T00:30:15.044+00:00
100076	1JePi2KUfl	pjp2Lu9OSl	5315	2023-10-03T00:30:15.042+00:00
100075	B9b9wAssST	bnN4JHD4Nj	71136	2023-10-03T00:30:15.039+00:00
100074	24GB0pomof	RDP2RkJpHL	5073	2023-10-03T00:30:15.034+00:00
100073	nbNzX8mgzS	YSoy4VzxO6	31113	2023-10-03T00:30:15.030+00:00
100072	bLiRs9h97N	bvyc8PtcjC	89877	2023-10-03T00:30:15.027+00:00
100071	lzKCpKRtct	0VapKuuLZg	17707	2023-10-03T00:30:15.025+00:00
100070	duqT9OVkFM	jdtRKDB4q8	11855	2023-10-03T00:30:15.023+00:00
100069	LyneKHMkEB	we43xxAxII	46418	2023-10-03T00:30:15.020+00:00
100068	qUgQLKlrw8	JQvjYz5Sqv	27248	2023-10-03T00:30:15.015+00:00
100067	fAaeax8ehn	7rnyqmCyf4	23964	2023-10-03T00:30:15.013+00:00
100066	prLEVOK71n	aH1CrpIH65	77770	2023-10-03T00:30:15.011+00:00
100065	RiEKsirLkv	wWIcY57dr7	11048	2023-10-03T00:30:15.009+00:00
100064	amh3ePxYAY	7xG5N5abgQ	74312	2023-10-03T00:30:15.004+00:00
100063	Ur7qlCLqYn	csm849cOXI	11441	2023-10-03T00:30:14.998+00:00
100062	YzF1PpxugK	gi6sJX0E8Z	27039	2023-10-03T00:30:14.996+00:00
100061	kLYEw8E54M	O80ZuP5hfm	7771	2023-10-03T00:30:14.992+00:00

Top 10 From Accounts

Transactions in last minute: 0.00

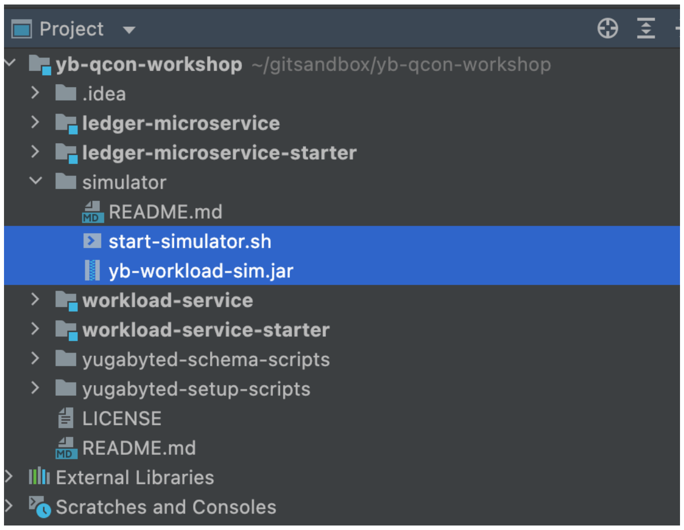
Yugabyte Server Information

Host	Cloud	Region	Zone
127.0.0.1	cloud1	datacenter1	rack1

Lab 6 - Simulate the transactions from multiple users.

Now we want to simulate these transactions. There are many ways to do it. We can use some tool like Locust. For the purpose of this workshop, we have created one of our own called workload simulator and jar for that can be found under the module called “simulator”

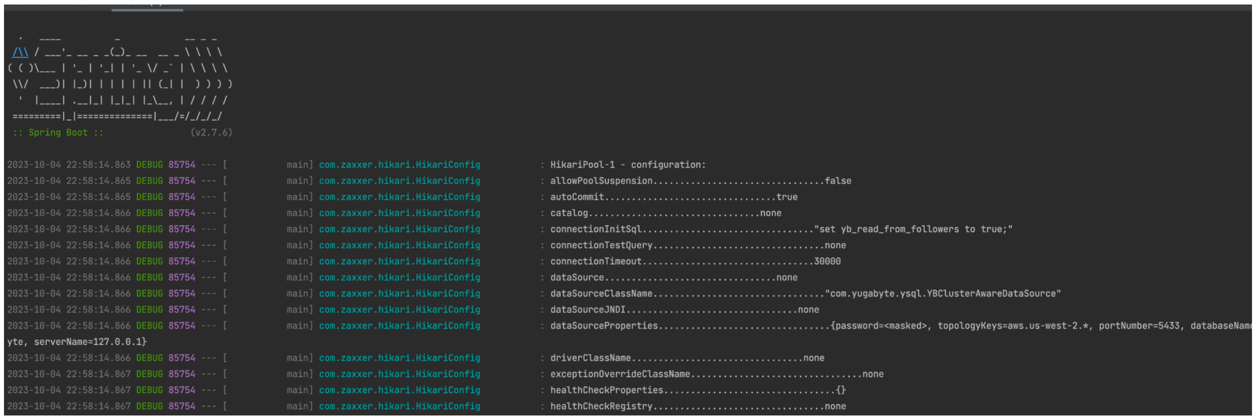
QCon Workshop Runbook



```
Project
yb-qcon-workshop ~/gitsandbox/yb-qcon-workshop
  .idea
  ledger-microservice
  ledger-microservice-starter
  simulator
    README.md
    start-simulator.sh
    yb-workload-sim.jar
  workload-service
  workload-service-starter
  yugabyted-schema-scripts
  yugabyted-setup-scripts
  LICENSE
  README.md
External Libraries
Scratches and Consoles
```

We will start the simulator app with this command:

```
Unset
./start-simulator.sh
```

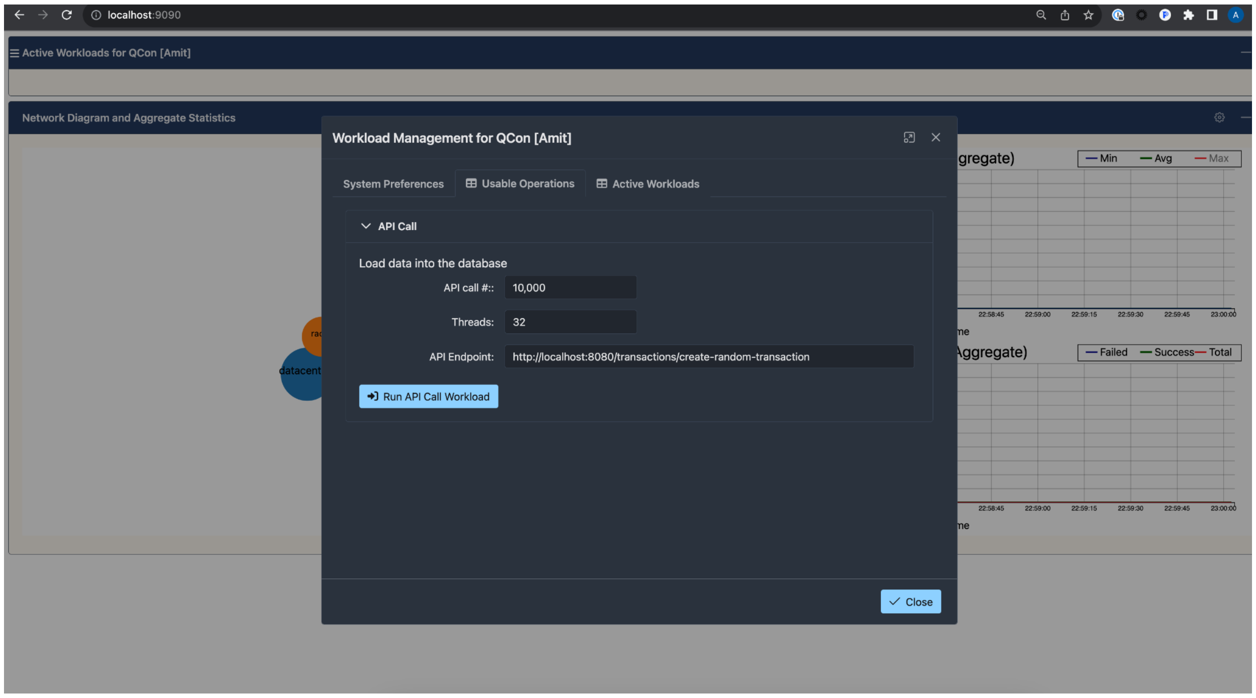


```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.6)

2023-10-04 22:58:14.863 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : HikariPool-1 - configuration:
2023-10-04 22:58:14.865 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : allowPoolSuspension.............................false
2023-10-04 22:58:14.865 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : autoCommit................................true
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : catalog................................none
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionInitSql..............................."set yb_read_from_followers to true;"
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionTestQuery.............................none
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : connectionTimeout...............................30000
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSource................................none
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceClassName............................."com.yugabyte.ysql.YBClusterAwareDataSource"
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceJNDI................................none
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : dataSourceProperties............................{password=<masked>, topologyKeys=aws.us-west-2.*, portNumber=5433, databaseName
yte, serverName=127.0.0.1}
2023-10-04 22:58:14.866 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : driverClassName................................none
2023-10-04 22:58:14.867 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : exceptionOverrideClassName.............................none
2023-10-04 22:58:14.867 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : healthCheckProperties...............................{}
2023-10-04 22:58:14.867 DEBUG 85754 --- [           main] com.zaxxer.hikari.HikariConfig           : healthCheckRegistry.............................none
```

Then we will access the UI from port 9090

```
Unset
http://localhost:9090/
```

QCon Workshop Runbook



Load data into the database

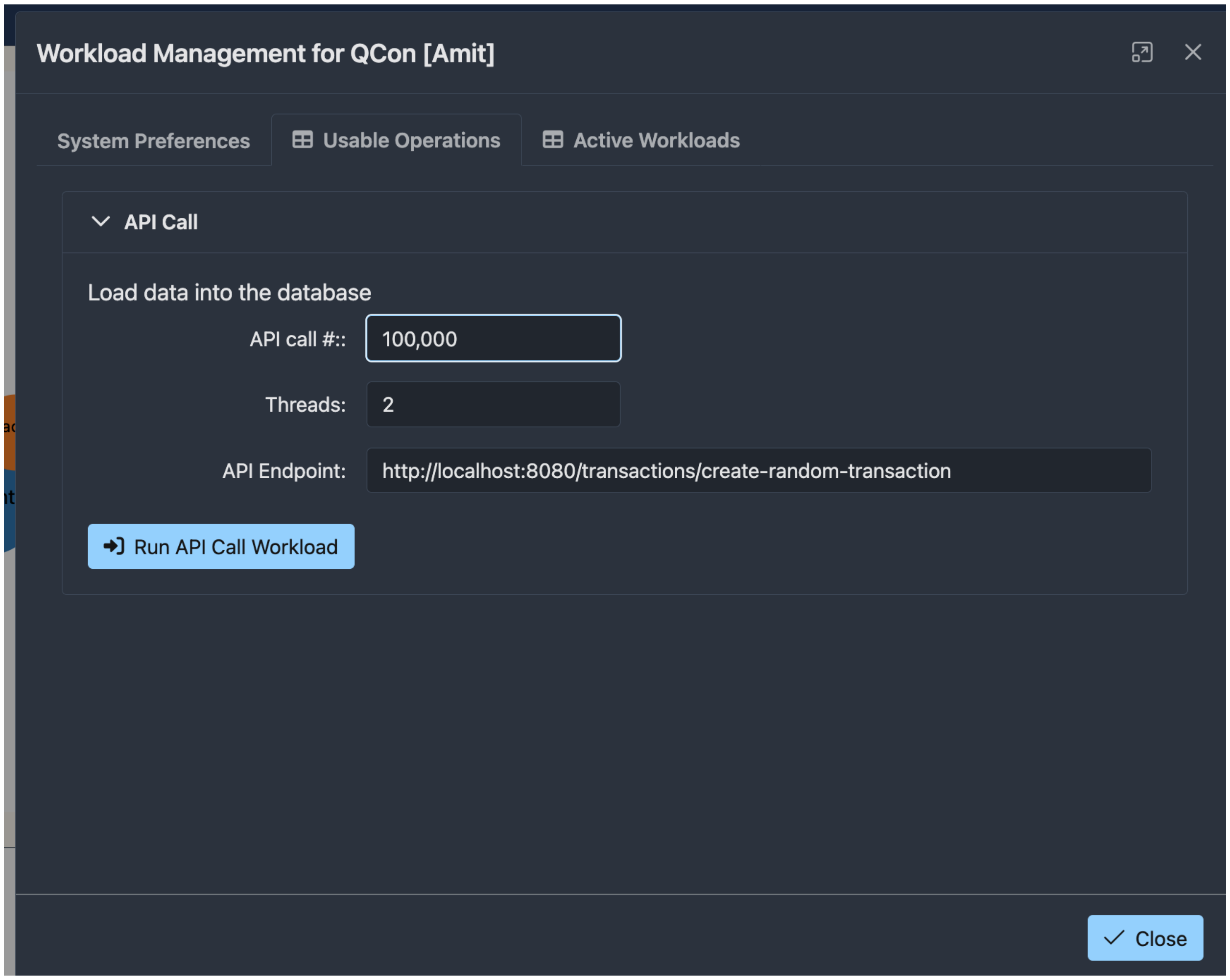
API call #:: 10,000

Threads: 32

API Endpoint: http://localhost:8080/transactions/create-random-transaction

From this UI, we can invoke any REST call with # threads for parallelism and watch the throughput and latency of these API calls. These calls will be creating random transactions in the database for us and we will observe these transactions on UI which we have built in previous labs.

Let’s keep the threads to lower count and increase the API call # to bigger number (say 100,000) so it runs for some time.



Workload Management for QCon [Amit]

Load data into the database

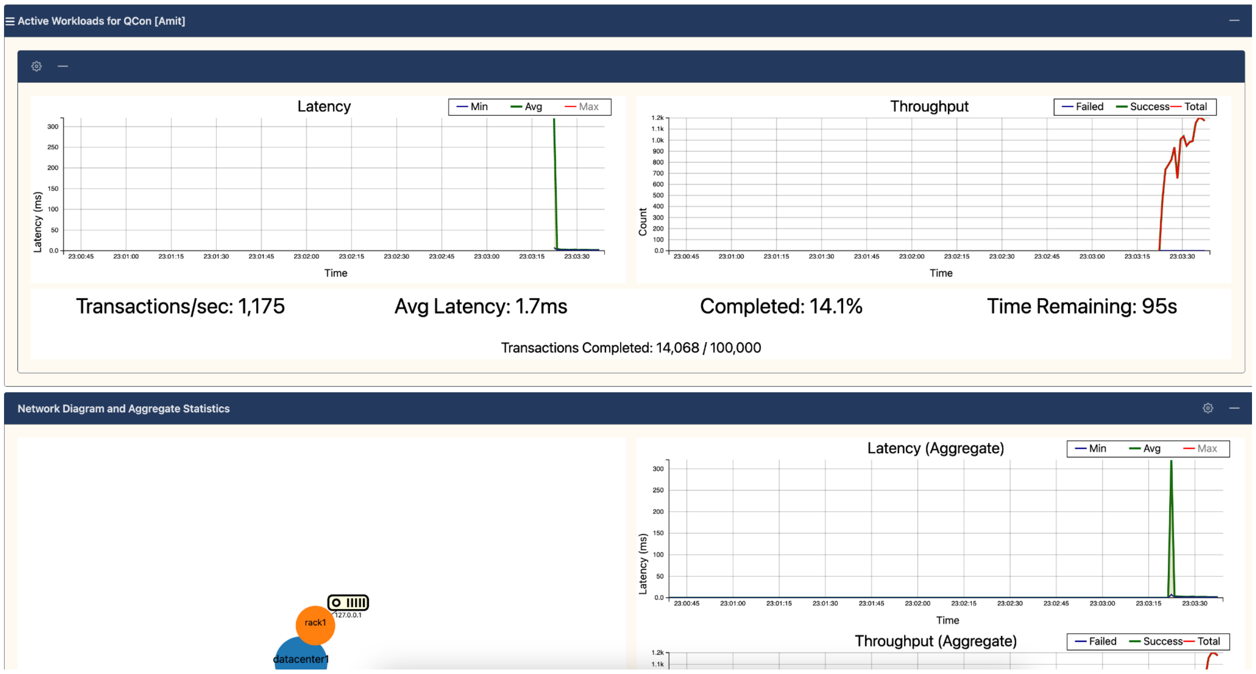
API call #:: 100,000

Threads: 2

API Endpoint: http://localhost:8080/transactions/create-random-transaction

QCon Workshop Runbook

You will notice the your transactions start up:

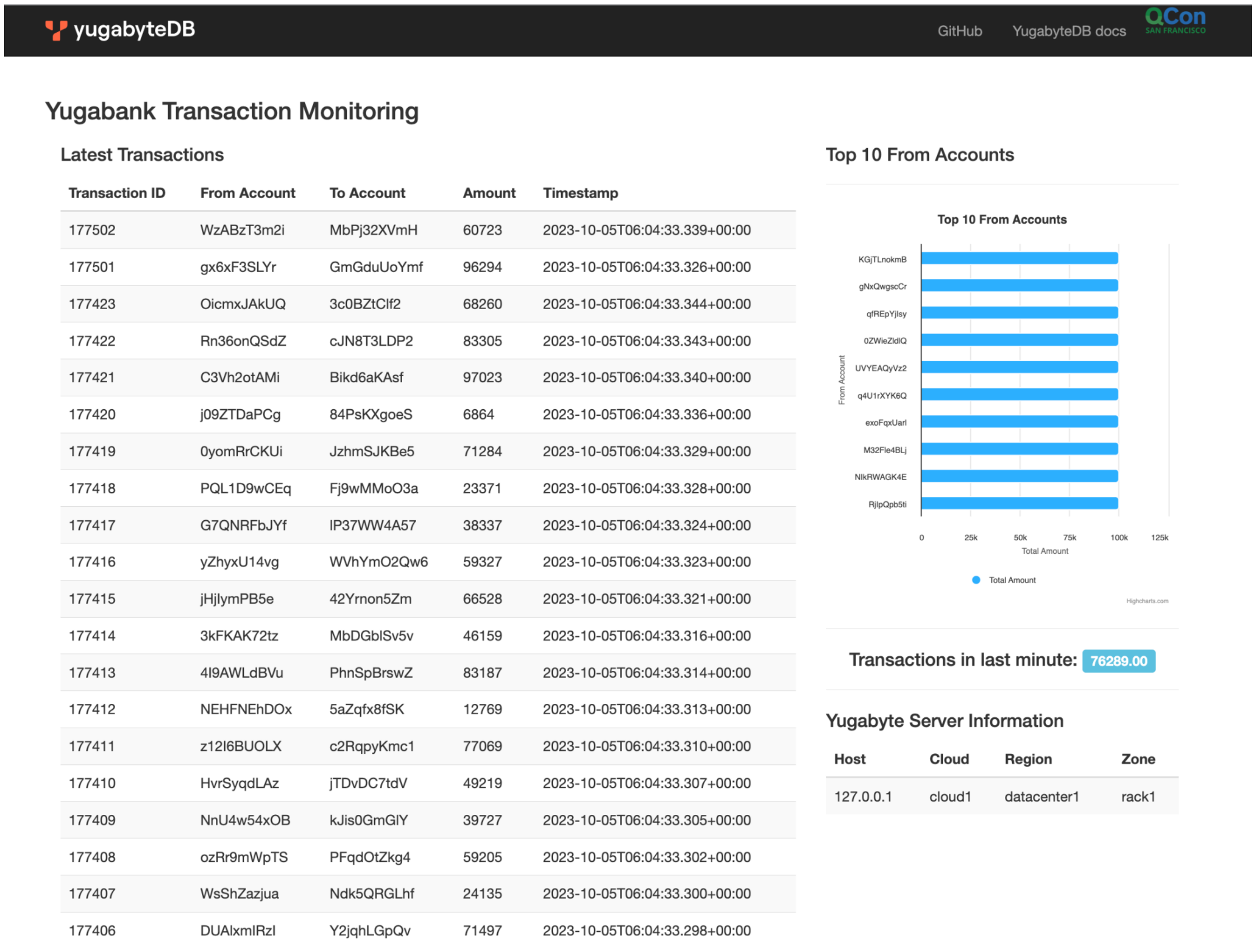


Transactions/sec: 1,175 Avg Latency: 1.7ms Completed: 14.1% Time Remaining: 95s

Transactions Completed: 14,068 / 100,000

Now let's go to our Yugabank UI and see our microservice in action:

QCon Workshop Runbook

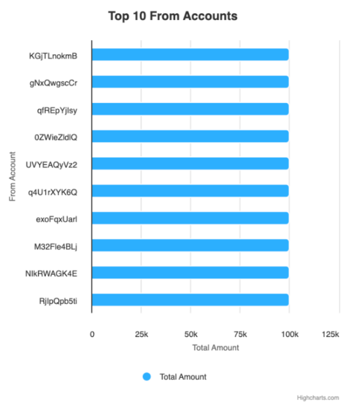


Yugabank Transaction Monitoring

Latest Transactions

Transaction ID	From Account	To Account	Amount	Timestamp
177502	WzABzT3m2i	MbPj32XVmH	60723	2023-10-05T06:04:33.339+00:00
177501	gx6xF3SLYr	GmGduUoYmf	96294	2023-10-05T06:04:33.326+00:00
177423	OicmxJAkUQ	3c0BZtClf2	68260	2023-10-05T06:04:33.344+00:00
177422	Rn36onQSdZ	cJN8T3LDP2	83305	2023-10-05T06:04:33.343+00:00
177421	C3Vh2otAMi	Bikd6aKAsf	97023	2023-10-05T06:04:33.340+00:00
177420	j09ZTDaPCg	84PsKXgoeS	6864	2023-10-05T06:04:33.336+00:00
177419	0yomRrCKUi	JzhmSJKBe5	71284	2023-10-05T06:04:33.329+00:00
177418	PQL1D9wCEq	Fj9wMMoO3a	23371	2023-10-05T06:04:33.328+00:00
177417	G7QNRFbJYf	IP37WW4A57	38337	2023-10-05T06:04:33.324+00:00
177416	yZhyxU14vg	WVhYmO2Qw6	59327	2023-10-05T06:04:33.323+00:00
177415	jHjlymPB5e	42Yrnon5Zm	66528	2023-10-05T06:04:33.321+00:00
177414	3kFKAK72tz	MbDGblSv5v	46159	2023-10-05T06:04:33.316+00:00
177413	4I9AWLdBVu	PhnSpBrswZ	83187	2023-10-05T06:04:33.314+00:00
177412	NEHFNEhDOx	5aZqfx8fSK	12769	2023-10-05T06:04:33.313+00:00
177411	z12I6BUOLX	c2RqpyKmc1	77069	2023-10-05T06:04:33.310+00:00
177410	HvrSyqdLAz	jTDvDC7tdV	49219	2023-10-05T06:04:33.307+00:00
177409	NnU4w54xOB	kJis0GmGlY	39727	2023-10-05T06:04:33.305+00:00
177408	ozRr9mWpTS	PFqdOtZkg4	59205	2023-10-05T06:04:33.302+00:00
177407	WsShZazjua	Ndk5QRGLhf	24135	2023-10-05T06:04:33.300+00:00
177406	DUAlxmIRzI	Y2jqhLGpQv	71497	2023-10-05T06:04:33.298+00:00

Top 10 From Accounts



Transactions in last minute: 76289.00

Yugabyte Server Information

Host	Cloud	Region	Zone
127.0.0.1	cloud1	datacenter1	rack1

This UI will keep refreshing every second and showing streaming transactions and charts on UI.

That's it for this Lab

Lab 7 - Resiliency testing

Now with workload running, we want to bring down one of the nodes and see the impact on the application. Scripts for this are present under QCon Workshop Runbook section.

Instructors will explain in depth by showing what goes under the hood when a node goes down in a distributed database.

Lab 8 - Scale up the Cluster from 3 Nodes to 6 Nodes

We will now scale out our cluster from 3 nodes to 6 nodes and observe the results on the App. We will see how data is automatically sharded and moved around by database. Scripts for this are present under QCon Workshop Runbook section

Instructors will explain in depth by showing what goes under the hood when a cluster is scaled.

Congratulations! You did it!!!