Hands-On Builder Session with Amazon ElastiCache for Redis

Table of Contents

Task 0 - Session Overview

In this session, you will build a solution to ingest, visualize, and augment OpenStreetMap (OSM) restaurant geodata using Amazon ElastiCache for Redis and AWS Cloud9.

You will be parsing an XML file containing geodata and storing data related to restaurant details, classifications, and locations in Redis. You will then implement sections of a web application in Python that pulls data out of Redis to allow you to visualize popularity of types of restaurants (using a Redis *sorted set*), map restaurants of a particular cuisine that are near you (using a Redis *geohash*), display details about selected restaurants (using a Redis *hash*), and rate restaurants on a scale from 1 to 5 stars (using a Redis *sorted set*).

For these exercises, it will help to reference the Redis commands (documented at https://redis.io/commands) as well as the redis-py client documentation (https://redis-py.readthedocs.io/en/latest/).

Note: In this session, you are using AWS Cloud9 to learn Redis syntax. In production, a similar AWS workload would ideally be automated.

Markups used in this guide

This guide will use the following formatting conventions:

- *Italicized text* will be referring to specific terminology relating to Redis architecture or data structures
- `monospaced font` will refer to variables or names of data sets relating to specifically this lab
- **Bold text** will be referring to files or scripts within your lab environment or the AWS console

Terminal logo commands are commands that need to be entered into your Redis or bash command line based on the instructions given prior

Information bubbles are supportive information expanding on instructions from the lab guide

Task 1 – Getting Started

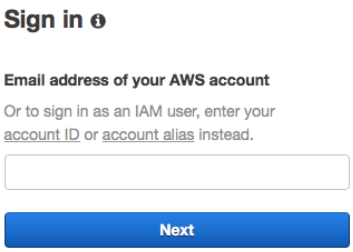
Your lab will be done on a temporary AWS account that includes some pre-configured AWS services you will be using. An AWS CloudFormation template will have been run in each account to setup the environment. You will be using AWS Cloud9 as your interactive IDE to both develop and run the web application, which performs both ingestion and visualization.

Login to your AWS console

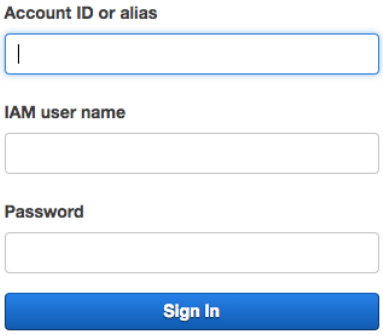
You should have received an index card with account information printed on it. This will be the information you use to login to your AWS console.

Visit https://console.aws.amazon.com and fill in the relevant login information along with the twelve-digit Account ID as it is shown on your index card. Your IAM user name for this lab is "**dat407"** and your password has been provided on the card.

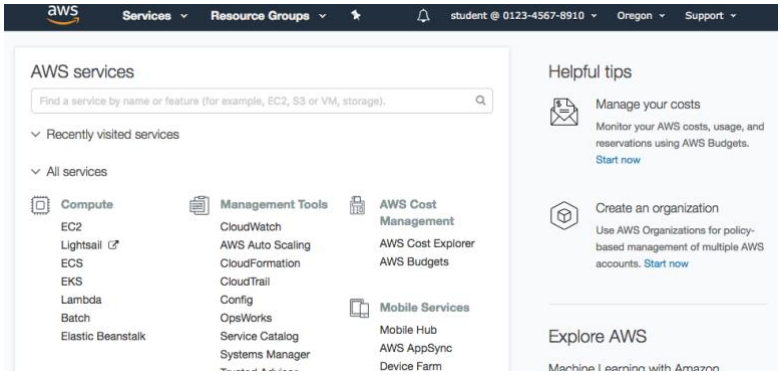
- ***Note:*** *If you have not used your current browser to login to an AWS account in the past, you will see this screen and need to enter just your account ID to proceed to the next screen.*



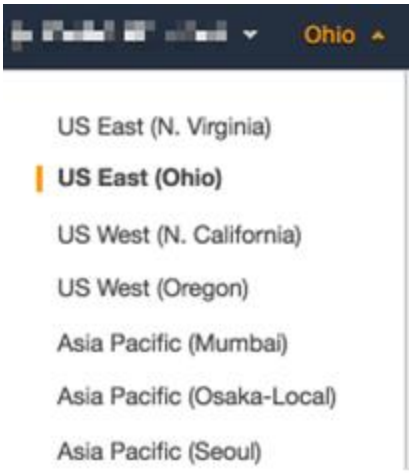
Sign in to your AWS account with the information provided on your card:



After logging in, you should be redirected to the AWS console as shown below:



Next, we'll make sure we are working in the correct region. NOTE: For this builder session, we'll be using the **US East (Ohio)** region (us-east-2). If you do not see **Ohio** in the menu bar on the top righthand part of the screen, click the region drop down and select **US East (Ohio)**.



Once you are in the Ohio region, you can continue to the next section of the session.

Open your Cloud9 Environment

Navigate to **Cloud9** by finding it in the list of services on the screen located under **Developer Tools** (Fig. 1). You can also find it by typing **Cloud9** into the search bar at the top (Fig. 2).

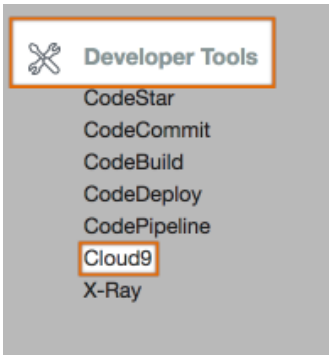


Figure 1 - Cloud 9 in the Services menu

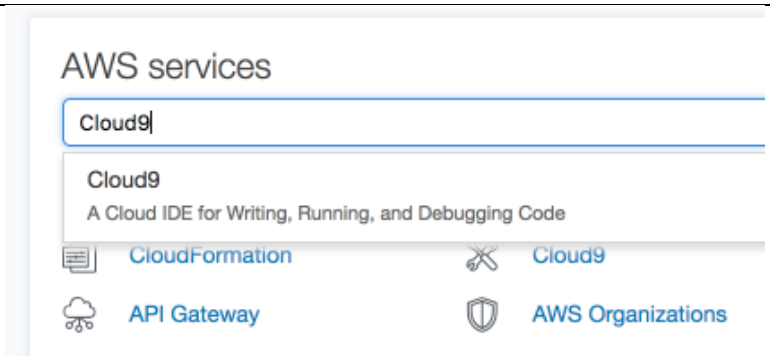
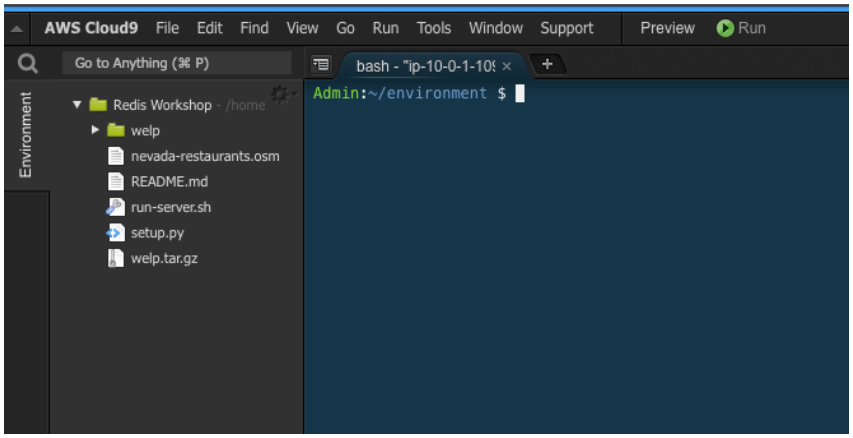


Figure 2 - Cloud9 in the Search menu

Once you have arrived at the Cloud9 homepage, locate the environment named "**Redis Workshop**" and click the "**Open IDE**" button. It may take a minute for the environment to become available as the underlying EC2 server boots.

Your AWS Cloud9 environment will boot preconfigured with all of the necessary scripts and packages needed to run the lab environment. Verify that you see folders named on the left pane of your browser window.

If you don't see a left pane, try clicking the **Environment** button on the left side of the screen.



The tab that is currently open should be a terminal window. If you do not see this, go to **Window > New Terminal** to open a new terminal session.

Review ElastiCache Environment

Now we will run some basic AWS CLI commands to validate and connect to your Amazon ElastiCache for Redis environment.

In the Cloud9 terminal that you created in the previous step, run the following command:

```
aws elasticache describe-cache-clusters --show-cache-node-info
```

describe-cache-clusters

Returns information about all provisioned clusters if no cluster identifier is specified, or about a specific cache cluster if a cluster identifier is supplied. You can use the optional *ShowCacheNodeInfo* flag to retrieve detailed information about the cache nodes associated with the clusters. These details include the DNS address and port for the cache node endpoint.

After this command is run, you will get a JSON object as an output, similar to the one shown below:

```
Admin:~/environment $ aws elasticache describe-cache-clusters --show-cache-node-info
{
    "CacheClusters": [
        {
            "Engine": "redis",
            "AuthTokenEnabled": false,
            "CacheNodes": [
                {
                    "CacheNodeId": "0001",
                    "Endpoint": {
                        "Port": 6379,
                        "Address": "welp.a3urbw.0001.use1.cache.amazonaws.com"
                    },
                    "CacheNodeStatus": "available",
                    "ParameterGroupStatus": "in-sync",
                    "CacheNodeCreateTime": "2019-11-22T18:55:56.606Z",
                    "CustomerAvailabilityZone": "us-east-1c"
                }
            ],
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "CacheParameterGroupName": "default.redis5.0",
                "ParameterApplyStatus": "in-sync"
            },
```

Take a moment to browse the response output. Be sure to note your **cluster's endpoint address (hostname)** for later steps**.**

For this workshop environment, we have built an ElastiCache for Redis cluster with a single node. In a production environment, this cluster would be configured to be replicated across multiple availability zones for redundancy which is a best practice for achieving HA.

We will now connect to your Redis cluster. Since you will be using the AWS CLI later in this guide, let's open a new terminal window by going to **Window > New Terminal** in your Cloud9 environment. This will open a terminal in a new tab. You can navigate to your other terminal by clicking on the previous tab.

Once your new terminal has loaded, type in the following command to connect to your Redis cluster:

```
redis-cli -h [cluster hostname]
```

The above Redis CLI command is used to connect to your Redis cluster. Replace the cluster endpoint with your cluster's hostname. We have used the default Redis port (6379).

Upon successful connection, you should see something similar to below.

You can use the following command to see key information about the node, including key count, memory usage, throughput, etc.

```
INFO
```

INFO [section]
Returns information and statistics about the server in a simple format.

Key fields to understand are **used_memory_human**, which indicates how much memory Redis is using and should be a few MB at most without any data, **connected_clients**, which shows how many current connections are established to the Redis server, and the **"# Keyspace"** section, which will indicate how many keys are present in the Redis server.

Task 2 – Application setup

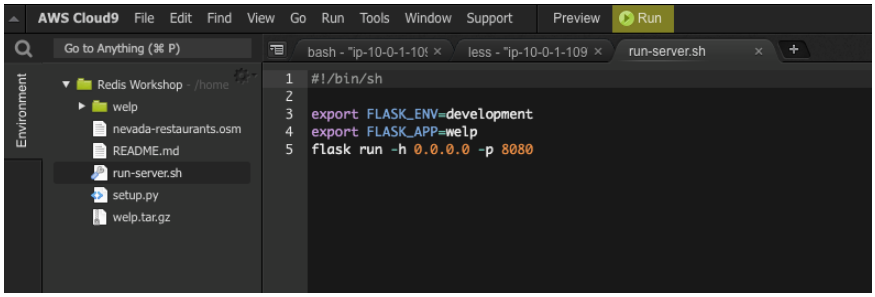
With Redis set up successfully, we are now ready to start ingesting data. In this task, we will be parsing an OpenStreetMap (OSM) file of locations corresponding to restaurants in the state of Nevada. OSM files are XML-formatted and contain *nodes* (points), *ways* (connections), and *relations* (street and object properties). We have preprocessed this file to only contain nodes that contain a tag called "*cuisine*" which indicates the given location is a restaurant.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="osmfilter 1.4.0">
        <bounds minlat="35.00053" minlon="-120.0074" maxlat="42.00391" maxlon="-114.0379"/>
        <node id="302178859" lat="36.2425178" lon="-115.2359547" version="4" timestamp="2017-12-25T14:42:08Z" changeset="0">
                <tag k="amenity" v="pub"/>
                <tag k="created_by" v="Merkaartor 0.11"/>
                <tag k="cuisine" v="beer_cheese"/>
                <tag k="name" v="Draft House"/>
                <tag k="opening_hours" v="24/7"/>
                <tag k="wheelchair" v="yes"/>
        </node>
        <node id="306342921" lat="36.113433" lon="-115.1541095" version="5" timestamp="2019-06-13T04:39:44Z" changeset="0">
                <tag k="addr:housenumber" v="4165"/>
                <tag k="addr:postcode" v="89169-6512"/>
                <tag k="addr:street" v="Paradise Road"/>
                <tag k="amenity" v="restaurant"/>
                <tag k="brand" v="P.F. Chang&amp;#39;s"/>
                <tag k="brand:wikidata" v="Q5360181"/>
                <tag k="brand:wikipedia" v="en:P. F. Chang&amp;#39;s China Bistro"/>
                <tag k="cuisine" v="asian"/>
                <tag k="name" v="P.F. Chang&amp;#39;s"/>
                <tag k="opening_hours" v="Su-Th 11:00-22:00; Fr-Sa 11:00-23:00"/>
                <tag k="phone" v="+17027922207"/>
                <tag k="website" v="https://www.pfchangs.com/"/>
        </node>
```

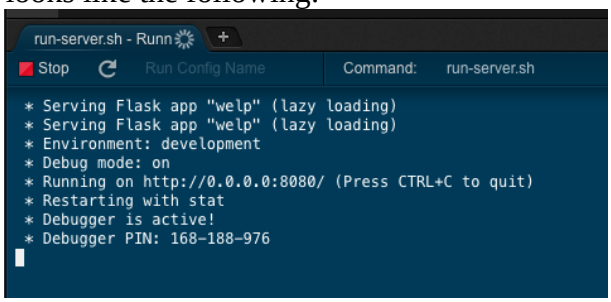
We will first start the application. It won't do anything yet since there is no Redis node connected.

1. In the left pane of your Cloud9 environment, you'll see a script named **run_server.sh**. Double click on it to load the script. This script will launch the application locally in development mode, and will automatically reload when you make code changes.

2. Click the "Run" button, which is located in the top toolbar of your Cloud9 workstation:



3. You should a screen that looks like the following:



4. At the top of the toolbar, click “Preview” and then “Preview Running Application”. This will launch an embedded browser where you can interact with the web application. If you prefer, you can click the top right button in the corner to view the webapp in your native browser. You should see “Welp” at the top with some buttons.
5. We will tell the application where to connect to your Redis node. Open up **__init__.py** within the **welp** folder and change the **hostname** variable (line 9) to your node’s hostname from the prior exercise. Save this file, and the application should automatically reload.

Task 3 – Populating cuisine rankings widget

In the first section, we will ingest a subset of the data into a Redis *sorted set*, which will allow us to visualize the most popular types of restaurants in Nevada. A *sorted set* is a Redis data structure that consists of a non-repeating collection of strings. Unlike a regular *set*, however, every member of a *sorted set* is associated with a score that can make the set ordered. While members are unique, scores may be repeated. We will be able to query this *sorted set* to create a leaderboard for restaurant rankings in the state, which will also serve as our cuisine selection widget.

Data ingestion – sorted set

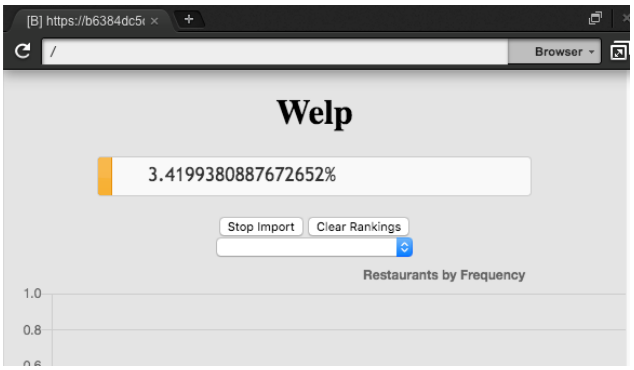
For the application to function, the OSM data file must be parsed and stored into Redis data structures. A parser has already been provided for you, but you must complete the Python code to process the data and store it into Redis.

1. Open the **data_import.py** file and inspect the **process_restaurant** function. You will fill in this implementation. To start, we want to create a single *sorted set* that allows you to determine how many unique restaurants exist for each type of cuisine. To do this, we’ll leverage the ZINCRBY function as we iterate through the entire dataset.

ZINCRBY key increment member

Increments the score of **member** in the sorted set stored at **key** by **increment**. If **member** does not exist, it is added. If **key** does not exist, it is also created. Note that increment must be a **float**.

2. After adding the command, save the file and then click on “Import Data” on the application. You should see the file being parsed with a progress bar on top:



3. Now go back to the redis-cli tab and inspect the contents of Redis to see the results. You can use the following command to see all the keys we currently have in our Redis datastore.

SCAN 0

SCAN cursor [MATCH pattern] [COUNT count]

Iterates through keys in the database, optionally matching a pattern. This command paginates through the data so is safe to run with many items, unlike the alternative **KEYS** command.

Data visualization – sorted set

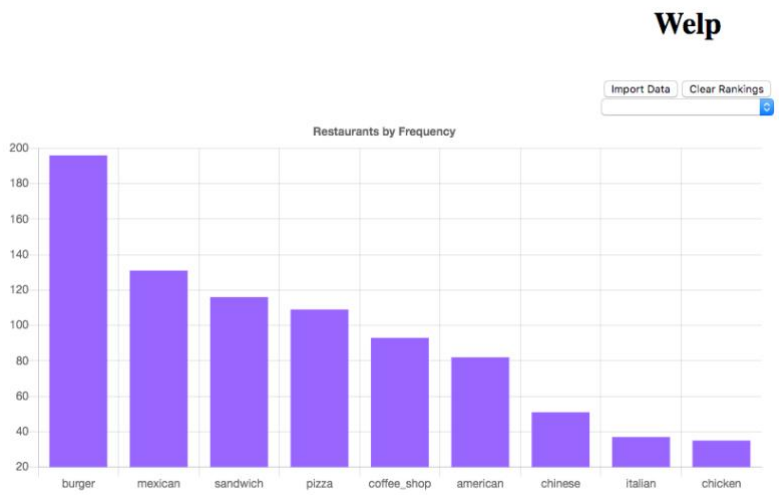
1. Now that data is ingested in Redis, we want to access the data and display a graph for the top restaurant cuisines in the area. Open up **restaurants.py** and examine the **restaurant_frequencies** command. Here, we’ll use the ZREVRANGEBYSCORE function. Make sure to leverage the redis-cli terminal to try out commands outside of the application context.

   Note: Pay attention to the return value of the function (a dictionary) vs the return value of the Redis function (a list). Redis does not currently have a dictionary return type.

ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]

Returns all the elements in the sorted set at **key** with a score between **max** and **min** (inclusive). Elements are returned in sorted order from highest score to lowest score (reverse order). The special identifiers **+inf** and **-inf** can be used for **max** and **min.**

2. After adding the command, save the file and then refresh the web page. You should see a graph of the data you ingested in the ingestion exercise. Each bar graph will be clickable and allow you to advance on to the next exercise.



Task 4 – Populating the map widget

Now that we have data to populate the cuisine selection widget, we want to be able to click on one and visualize where the given subset of restaurants are on a map. To do this, we can use a Redis *geohash*, which stores a set of {latitude, longitude, name} entries and allows querying for entries within a given radius of a point.

Data ingestion - geohash

We need to extract the latitude and longitude information from our dataset to store into Redis. Open **data_import.py** and use the **latitude** and **longitude** to create one geohash per cuisine type. This will allow us to query restaurants within the context of a specific cuisine.

Note: members must be unique: the **id** field can be reused as a unique internal identifier, and we will map the internal identifier to an external name in the next task. Save the page and refresh.

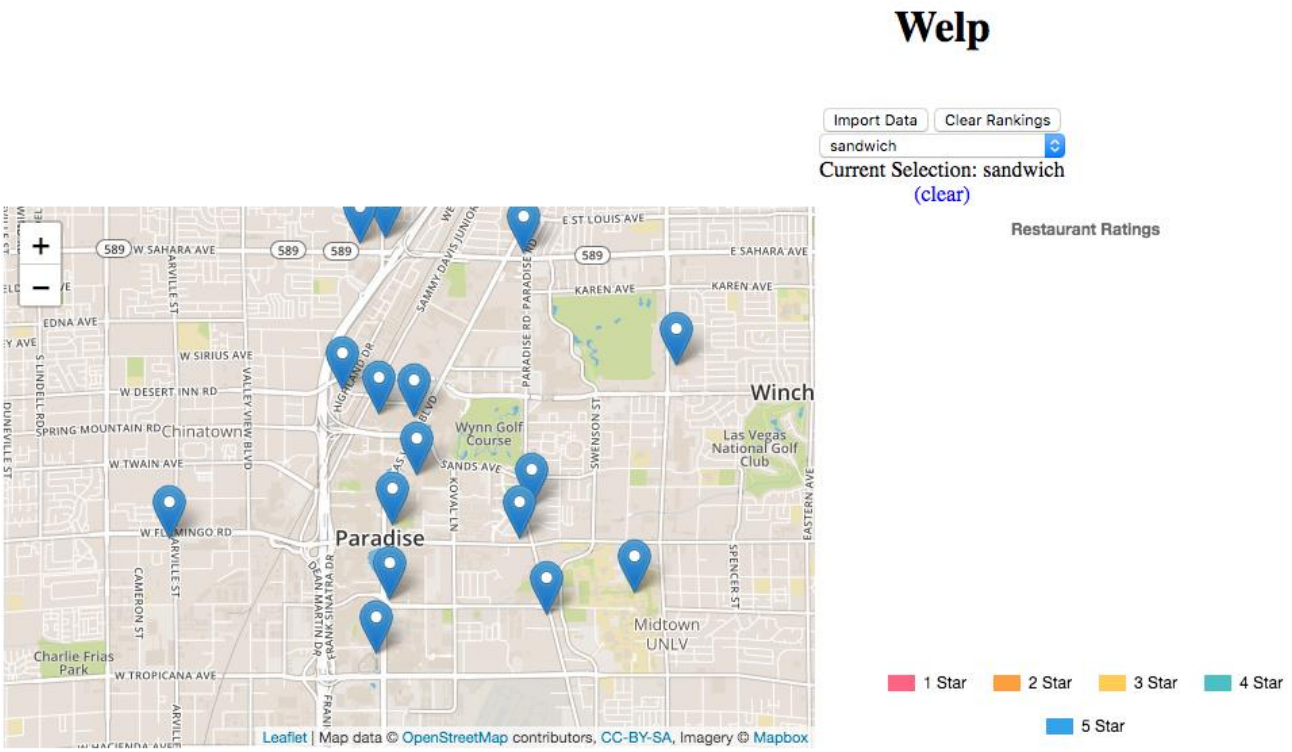
GEOADD key longitude latitude member [longitude latitude member ...]
Adds the specified geospatial items (**latitude**, **longitude**, **member**) to the specified **key**.

Data visualization - geohash

Now open **map.py** and add code to query the geohash you created in the **map_data** function. Save and refresh the page, making sure to import data again. When you click on a cuisine, the map should populate with points.

GEORADIUS key longitude latitude radius m|km|ft|mi [WITHCOORD] [COUNT count]

Returns the members of a sorted set populated with geospatial information which are within the borders of the area specified with the center location and the maximum distance from the center (the radius).



Task 5 – Populating the details of each restaurant

Now that the map shows points, we want to make them clickable and be able to display information about each of the selected restaurants. We will use a Redis *hash* to store attributes about each restaurant.

Data ingestion – hash data

We need to store all the tags associated with the restaurant, including (importantly) its name, which we will use for display purposes (the restaurant's name is given by a tag named 'name'). Open **data_import.py** and use **data** to store in a hash data structure for each restaurant. See if you can store all the attributes using one Redis call instead of one per attribute, which will minimize latency.

HSET key field value [field value ...]

Sets the specified fields to their respective values in the hash stored at **key**.

Data visualization – hash data

Now open **restaurants.py** and look at the **get_restaurant_data** method. We want the ability to return a single field (e.g. the name of a restaurant), or all stored fields (for a more detailed view). After completing this function, the map should display the name of each restaurant as you click on it instead of "Unknown."

HGET key field

Returns the value associated with **field** in the hash stored at **key**.

HGETALL key

Returns all fields and values of the hash stored at **key**.

Listing restaurants by name

We now have the ability to query restaurant *IDs* by geolocation, but we don't have a data structure that provides a list of restaurant *names* by geolocation. We'd like to display a table whenever a cuisine is selected with all restaurant names on the map. We could do this in two calls by first querying the geohash, then fetching the name for each restaurant, but this would increase latency and data transfer required between the Redis server and web application. Redis allows composing operations with a simple server-side scripting language using Lua. Using a Lua script, we can do the two operations using a single Redis call.

Open **data_import.py** and examine the **restaurant_list** function. This function must combine data from the geohash we created earlier (to determine the set of restaurants in a given radius) and the hash we just created (to fetch details from each). The script template is provided, but you must fill in the syntax for the two functions. Refer to https://redis.io/commands/eval to see how to call Redis functions and refer to arguments and keys passed in to the script evaluation function.

EVAL script numkeys key [key ...] arg [arg ...]

Executes a Lua script within Redis. The script can operate on multiple keys specified by the **key** arguments, and can be passed other non-key arguments as well.

- Subway

Key	Value
Rating	★★★★★
Amenity	fast_food
Brand	Subway
Brand:wikidata	Q244457
Brand:wikipedia	en:Subway (restaurant)
Takeaway	yes

Task 6 – Populating the ratings widget

The last feature we want to add to the application is the ability to rate restaurants on a scale of one to five stars and visualize the ratings we've assigned within the specific restaurant categories. Specifically, we want to create a pie chart of the breakdown of ratings for each cuisine type, as well as the ability to record and find each restaurant's rating. For this, we will use a *sorted set* for each cuisine type.

Add ratings functionality – sorted set

1. Open **ratings.py** and look at the **get_rating** and **set_rating** functions, which will be used to query for and rate a given restaurant.

ZADD key score member [score member ...]

Adds all the specified members with the specified scores to the sorted set stored at **key**.

ZSCORE key member

Get the score associated with the given **member** within the sorted set **key**.

2. Now examine the **cuisine_ratings** function, where we want to associate each star rating value (1-5) with the number of restaurants that were rated with this score. Similar to last exercise, we need to make multiple calls to Redis to gather all of the data. This time, instead of using a Lua script, try using client-side batching to execute the set of requests. This can lower latency significantly by sending all the requests to the server simultaneously without waiting for each successive response to be returned, lowering latency as well as CPU utilization on the Redis server.

ZCOUNT key min max

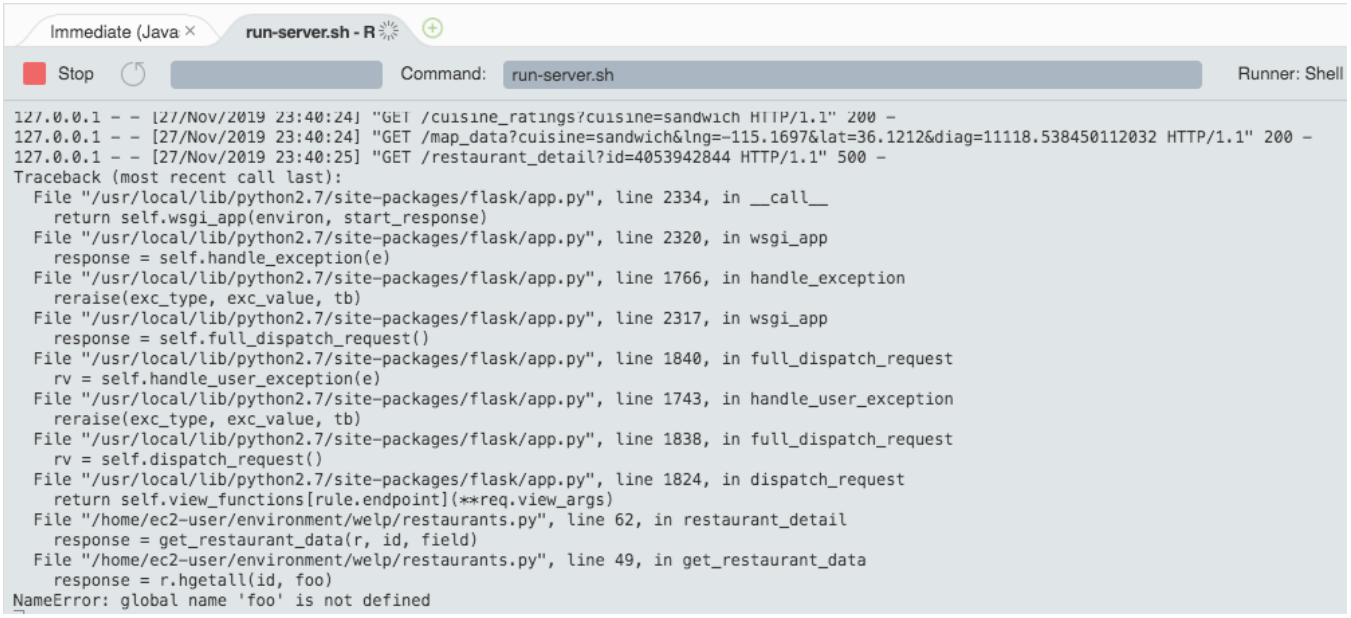
Returns the number of elements in the sorted set at **key** with a score between **min** and **max**.

3. If you still have time, you can implement the **clear_ratings** function as well. See how you can use the SCAN function to delete only the sorted sets containing the rating data.

Troubleshooting Problems

Safari does not support previewing using Cloud9 by default, as it blocks third party cookies. To fix this, you can navigate to the Safari menu, select "Settings for This Website," and then uncheck "Enable content blockers."

If something goes wrong during development, there a few things to check. First, Python errors from the application should be shown on the console screen in your **run-server.sh** session.



Most errors in the application development should be displayed here. In some cases, reloading the application explicitly by clicking "Stop" and "Start" can fix errors with out of sync resources.

After importing data, you can check that the Redis data structures match what you expect by using redis-cli to interact and query the data. This will pinpoint the problem to ingestion or visualization.

Though the client-side JavaScript was written to be robust, there may still be an issue with rendering within the browser. To debug this, launch your browser's debugging tools and look at the Console to see if there are any JavaScript errors. In this mode, you can also inspect the requests and responses to/from the server to see if they contain expected data.

Build On – Additional resources

Workshop setup on your own account

If you wish to re-run the workshop after re:Invent, you can download the sample code used as well as the CloudFormation template that creates the Cloud9 and ElastiCache Redis environments.

1. Create the CloudFormation stack using https://s3.amazonaws.com/reinvent2019-elasticache-workshop/template.yaml
2. Log in to the Cloud9 IDE and bootstrap it with Redis libraries:

   ```
   sudo pip install redis
   sudo yum install -y redis
   ```

3. Download and extract the workshop files:

   ```
   aws s3 cp s3://reinvent2019-elasticache-workshop/welp.tar.gz .
   tar -xzvf welp.tar.gz
   ```

4. Now fill in the **hostname** variable as instructed in Task 1. In addition to this, you must register for and fill in the **map_access_token** (line 16) in order to obtain map tiles. To obtain your own free token, visit https://www.mapbox.com and sign up for a free account. Usage below a specified threshold is free: see the MapBox usage terms for more information.

Note that if you want to get the full completed demo up and running, you can download https://s3.amazonaws.com/reinvent2019-elasticache-workshop/welp-completed.tar.gz.

Additional resources

Now that you have a better understanding of Amazon ElastiCache for Redis internals, we hope you will continue to explore the capabilities that Redis has to offer. There is a wealth of content available that can further your understanding.

Amazon ElastiCache for Redis Resources:

- https://aws.amazon.com/elasticache/redis/resources

Additional Labs:

- https://aws.amazon.com/getting-started/projects/
- Check the “Databases” box on the left