Amazon Redshift
LAB: Redshift Spectrum

September, 2017

Table of Contents

Overview

In this lab, you will learn how to use as well as performance tune Redshift Spectrum. The lab consists of the follow sections:

- Build your Lab Environment
- Create a Redshift Spectrum Data Mart
- Querying the S3 Data Lake and Performance Diagnostics
- Optimizing Performance with Partitions
- Storage Optimizations
- Predicate Pushdown
- Comparison: Native Redshift versus Redshift with Spectrum

Prerequisites

I. Lab Facilitator's Checklist

If you're not a lab facilitator, this section isn't pertinent. You can skip ahead.

Facilitators, need to follow these instructions to prepare the lab environment ahead of the workshop.

II. Participant Checklist

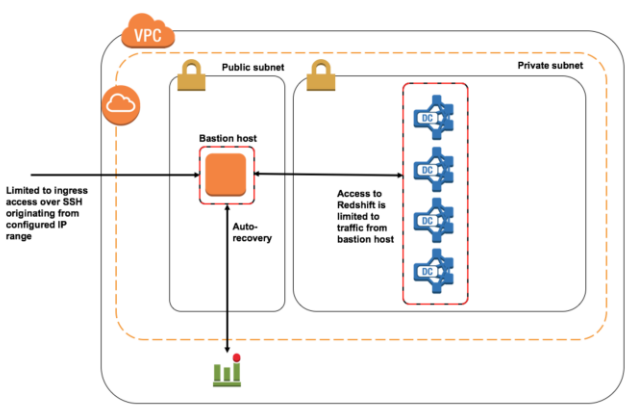
Ahead of the event, your lab facilitators should have prepared an AWS account for the workshop:

- You should have been provided credentials to log into the AWS lab account.
- You should have an EC2 key pair that originates from the region where the lab is being hosted. If you don't have one, create one.
- You should have been provided the name of two S3 buckets. These buckets contain data sets that you will use during the lab.

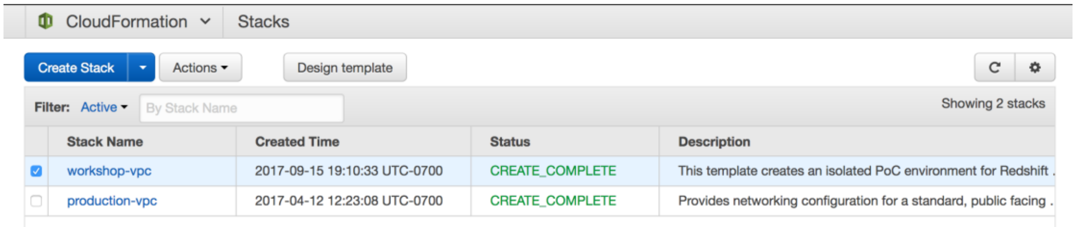
Section 1: Build your Lab Environment

Your lab facilitator may have completed this section for you to help accelerate your workshop. Please check with your lab facilitator now if you aren't sure whether this setup step has been completed for you.

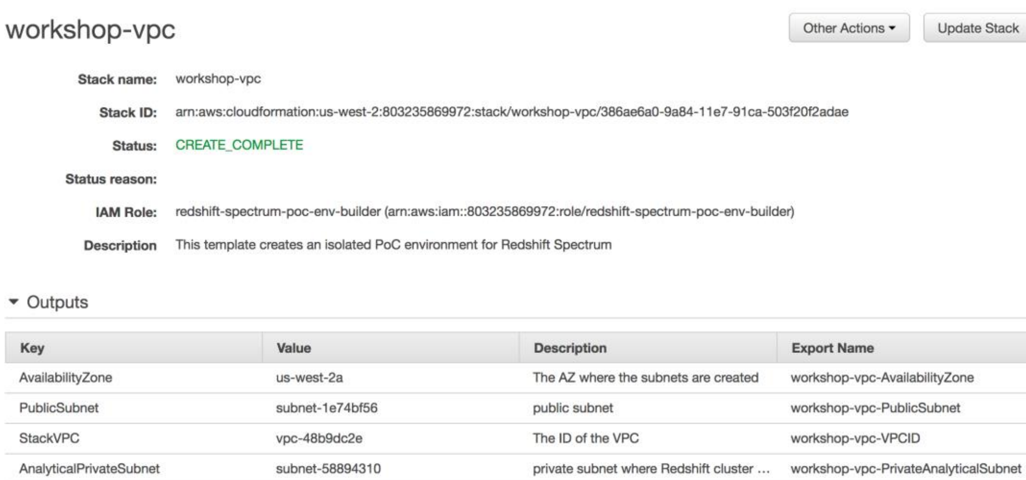
In this section, we will be building our lab environment. The diagram below illustrates the resources that will be created. The primary resources will be a Windows bastion host with a client pre-installed, and a 4-node (dc1.large) Redshift cluster.



1. Navigate to the CloudFormation service, and locate the stack "workshop-vpc." This stack should have been launched by the workshop facilitators.

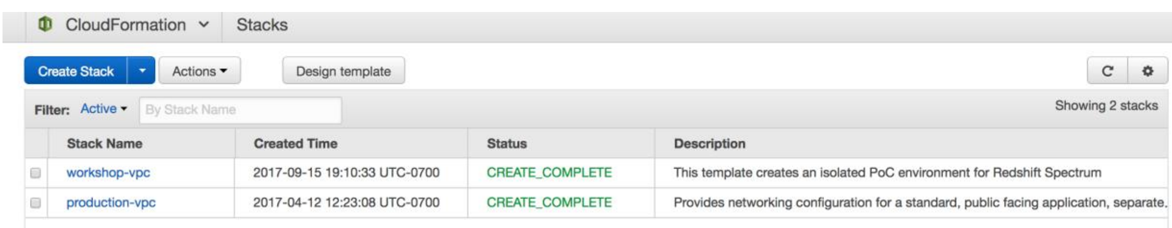


2. Click on the "workshop-vpc" link. You will be taken to a page with a number of subsections. Expand the subsection "**Outputs**." Take note of the four values as shown in the screenshot. You will need these to create your personal lab environment.

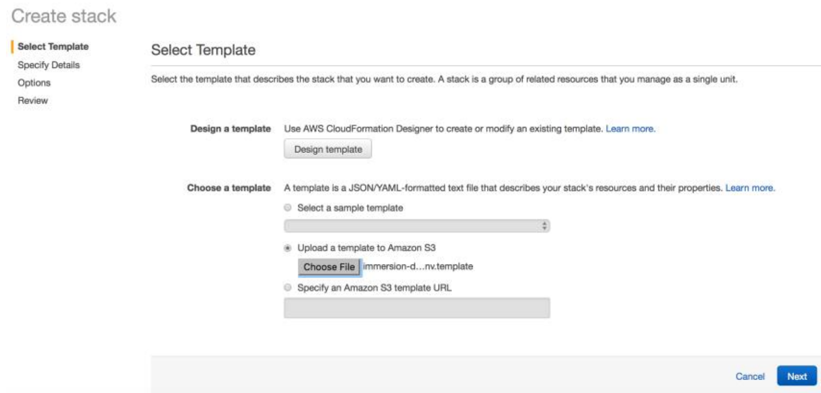


3. Download the CloudFormation template linked here. This template defines resources required by the lab, and will instruct CloudFormation to create these resources for you.

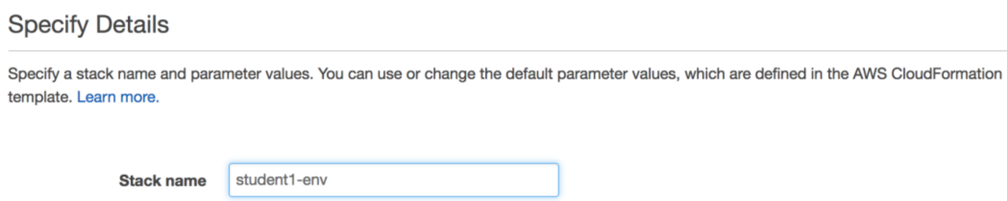
4. Return to the main page of the CloudFormation service, and click on "Create Stack."



5. Select the **Upload to Amazon S3** option, and upload the template you just downloaded. Click next.



6. You will be presented with a form that requires your input. The first parameter that requires your input is “**Stack name**.” Provide a name that is unique among your peers, and remember this name. You will be responsible for deleting this stack at the end of the lab.

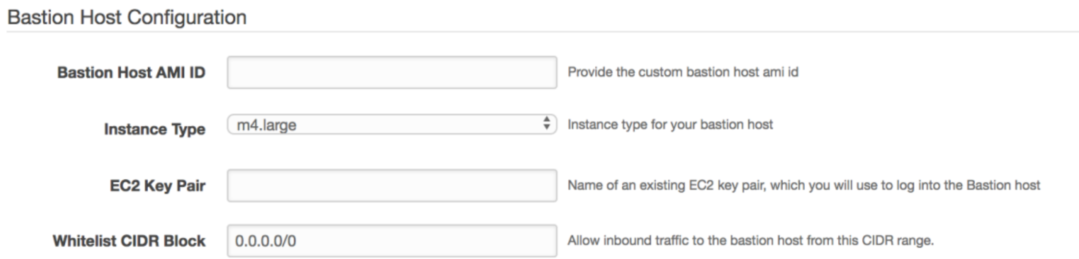


7. The next set of parameters informs the template about the VPC where your lab resources will be deployed.

Refer back to the output values from the “workshop-vpc” stack that you were asked to take note of.

- For **Workshop VPC ID**, enter the value associated with the key “StackVPC.”
- For **Public Subnet ID**, enter the value associated with the key “PublicSubnet.”
- For **Private Subnet ID**, enter the value associated with the key " AnalyticalPrivateSubnet.”

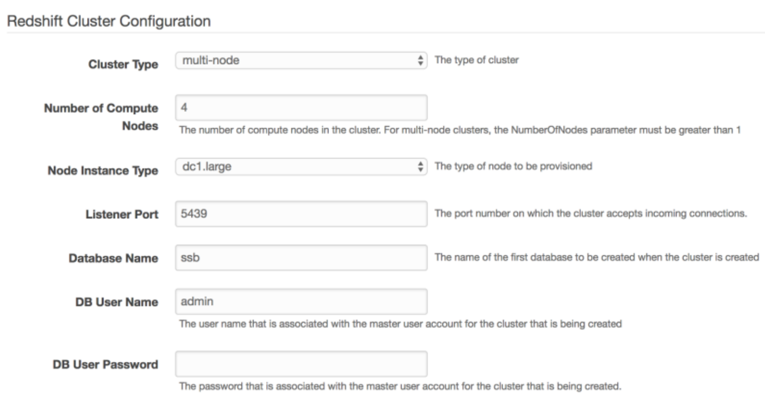
8. The parameters in the next section are used to configure your bastion host.



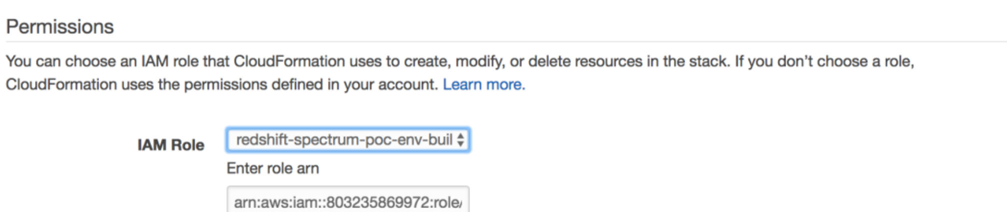
For **Bastion Host AMI ID**, enter the custom ami id that was provided by your lab facilitator. It should resemble the pattern like ami-xxxxxxx.

Leave the **Instance Type** as the default value. For **EC2 Key Pair**, enter the name of your private key (don't include the .pem extension in your input text). Leave **Whitelist CIDR Block** as the default.

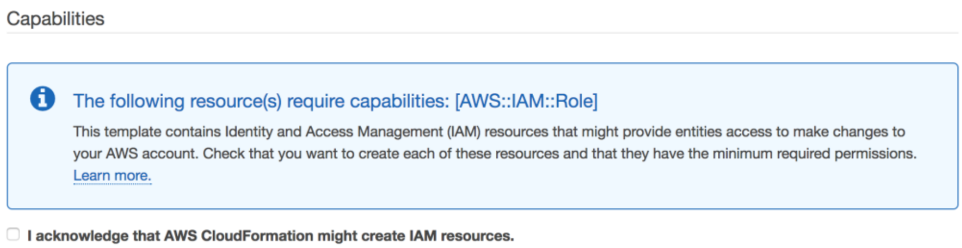
9. The next section consists of configurations required to provision your Redshift cluster. Leave all the values as their defaults. You will need to provide a **DB User Password**. Your password needs to be between 8 to 64 characters, and has to contain at least one digit and capital letter. Click next once you have provided a valid password.



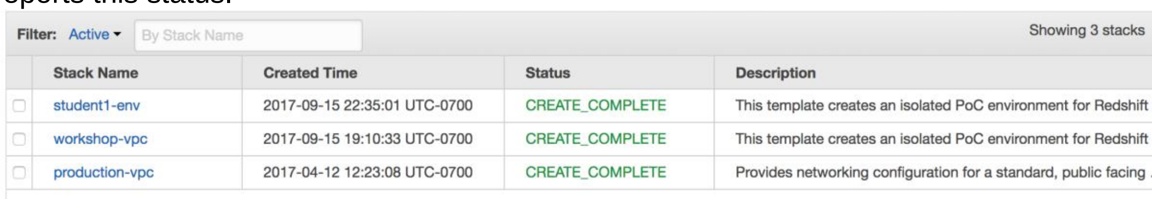
10. Scroll down to the **Permissions** sub section, and select the "redshift-spectrum-lab-builder" role. If you don't see the role, ask your lab facilitator for the name of the role that was created for this workshop. This role provides CloudFormation with the necessary permissions to launch the resources for your lab environment. Click on the Next button at the bottom of the page.



11. Select the checkbox “**I acknowledge that AWS CloudFormation might create IAM resources**.” Click on the Create button at the bottom of the page.



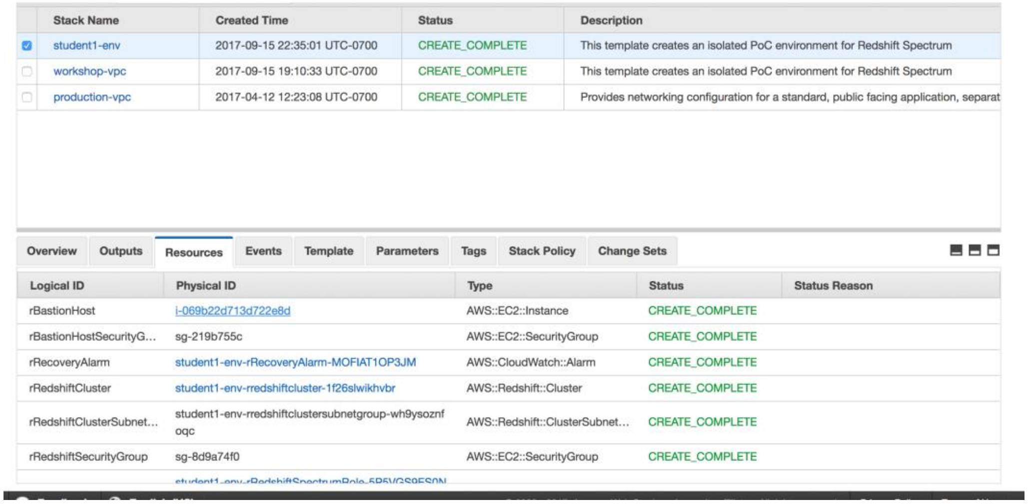
12. It will take 5-10 minutes for your resources to be provisioned. Your environment has been created once the status of your stack shows up as “CREATE_COMPLETE” on the main CloudFormation service page. You can continue with the lab once your stack reports this status.



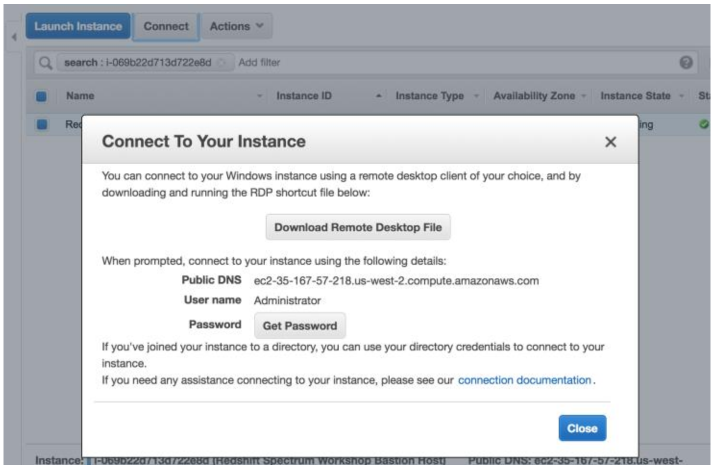
Section 2: Create a Redshift Spectrum Data Mart

In this section, you will be creating a data mart that consists of clickstream data in S3 serving as a fact table, and two dimension tables (time and customer attributes) residing in your Redshift cluster.

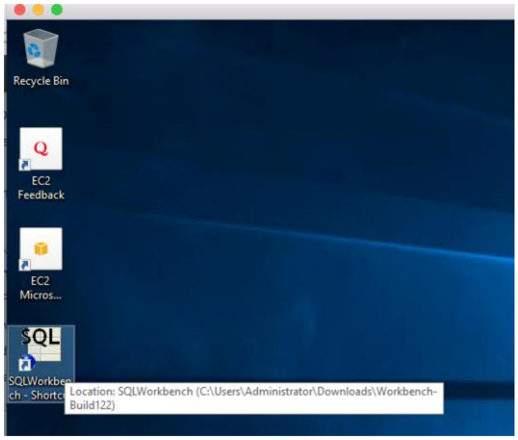
1. Log on to your bastion host. You can find your bastion host by navigating to the **Resource** tab of your CloudFormation tab. At the top of the tab, you should see the resource labeled “rBastionHost” under the Logical ID column. Click on the link under the corresponding Physical ID column. This will take you to the EC2 dashboard with the list of instances selected and filtered on your bastion host.



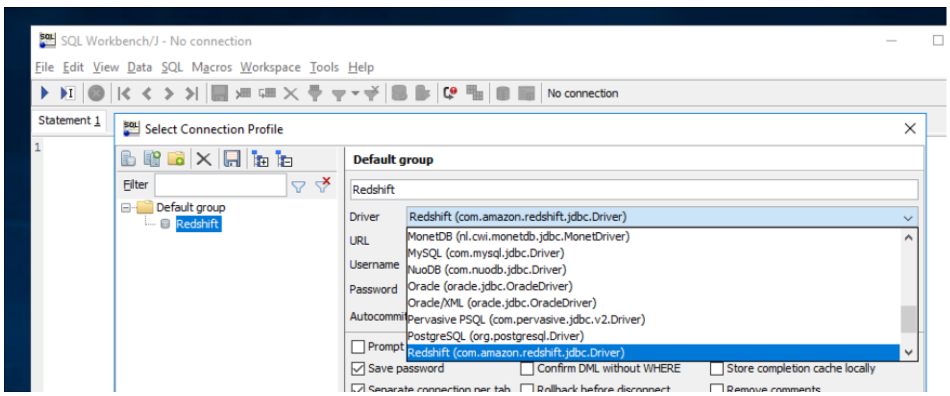
2. Click on the **Connect** button and follow the instructions to obtain your password using the private key that you provided to CloudFormation while you were launching your stack. Use Windows Remote Desktop to log on to the instance with the information provided.



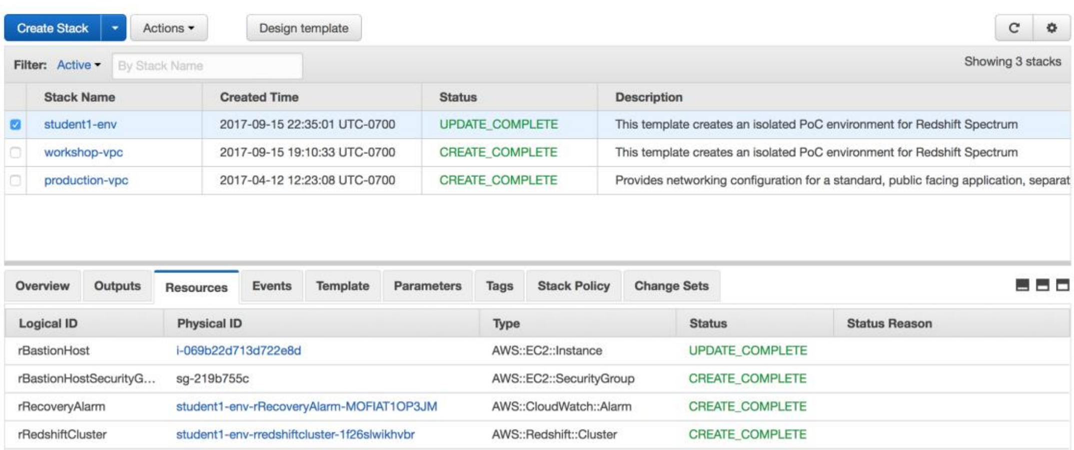
3. The custom AMI that was provided to you has SQL Workbench, Java and Redshift JDBC drivers pre-installed for you. Start-up SQL Workbench from the short-cut on the desktop.



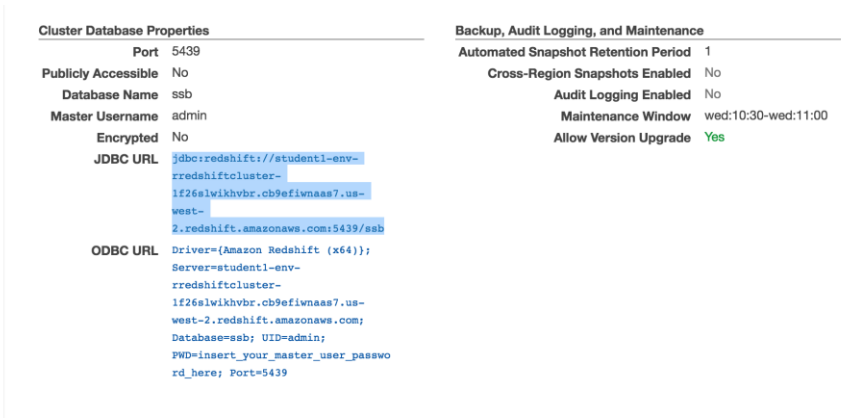
4. Configure a connection profile in SQL Workbench. Name the profile Redshift, and select Redshift from the driver drop down list.



5. Return to the Cloud Formation service page, and select your stack. Navigate to the **Resources** tab again. Find the label “rRedshiftCluster” under the Logical ID column, and click on the link on the corresponding Physical ID. This provides a shortcut to the Redshift cluster that was created for you.

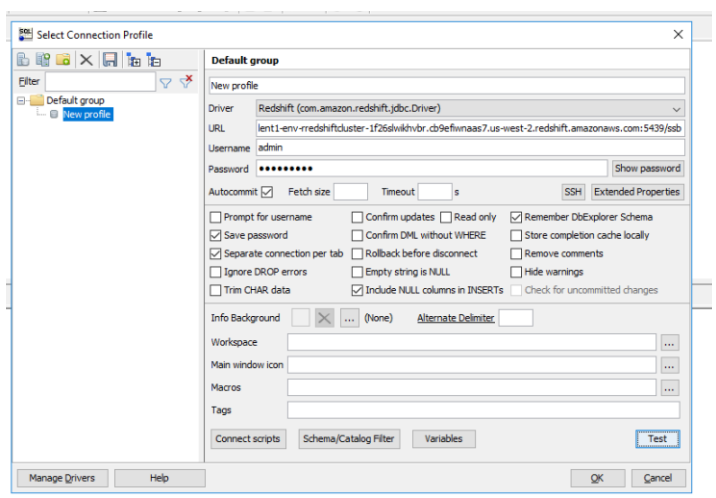


6. Scroll down to the **Cluster Database Properties** section of the Redshift cluster dashboard, and copy the **JDBC URL**.

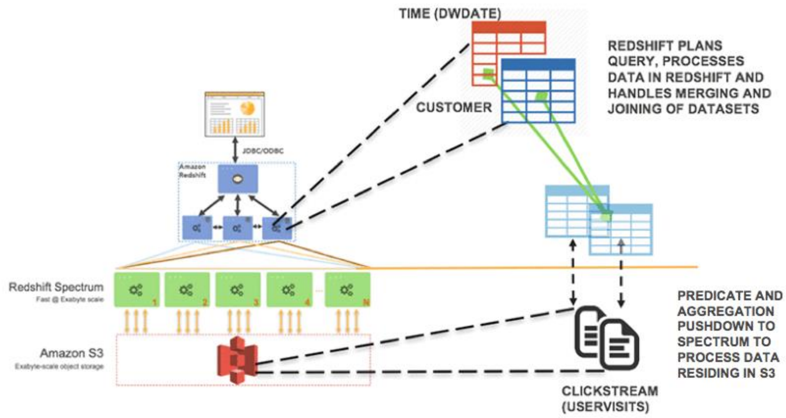


Return to SQL Workbench and paste this JDBC URL into the URL textbox of your connection profile.

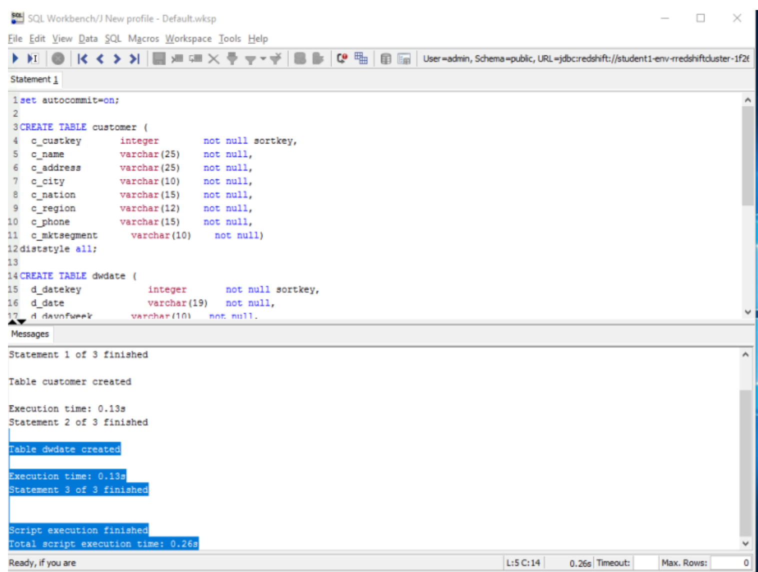
7. Fill the Username and Password textboxes with your credentials. The user name is admin, and the password is the one you provided to CloudFormation when you launched your stack. Click on the checkbox **Autocommit** just under the password textbox. Next, click on the test button to confirm that you have configured your profile correctly. Click OK once you have confirmation.



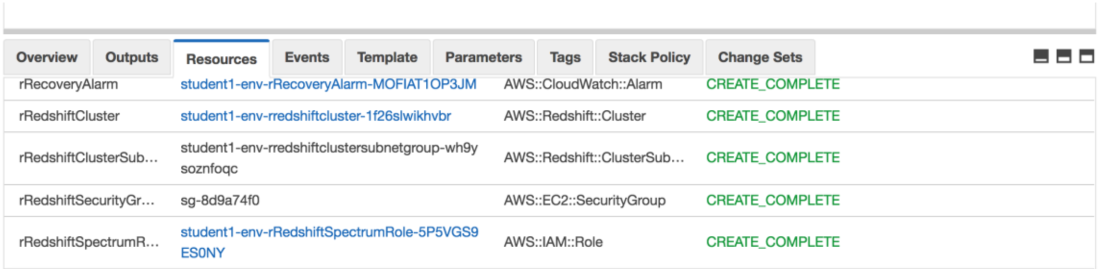
8. We are now going to start building our data warehouse and data lake. In this lab, we're going create a [star schema data model](#) by creating dimension tables in your Redshift cluster, and fact tables in S3 as show in the diagram below.



Create the dimension tables by running this script: [create-dimensions.sql]from the SQL Workbench query editor.



9. Next, we will load a couple of datasets into our dimension table from the [star schema benchmark]. You will do this by running this script: [load-dimension-data.sql.] You will need to provide an IAM role with the permissions to run the COPY command on your cluster. You can use the IAM role that was created for you by your CloudFormation template. Return to the **Resources** tab of your CloudFormation stack, and click on the link under Physical ID that corresponds to the “rRedshiftSpectrumRole” resource.



Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Policy	Change Sets

rRecoveryAlarm	student1-env-rRecoveryAlarm-MOFIAT1OP3JM	AWS::CloudWatch::Alarm	CREATE_COMPLETE
rRedshiftCluster	student1-env-rredshiftcluster-1f26slwikhvbr	AWS::Redshift::Cluster	CREATE_COMPLETE
rRedshiftClusterSub...	student1-env-rredshiftclustersubnetgroup-wh9y soznfoqc	AWS::Redshift::ClusterSub...	CREATE_COMPLETE
rRedshiftSecurityGr...	sg-8d9a74f0	AWS::EC2::SecurityGroup	CREATE_COMPLETE
rRedshiftSpectrumR...	student1-env-rRedshiftSpectrumRole-5P5VGS9 ES0NY	AWS::IAM::Role	CREATE_COMPLETE

Replace the two strings in the script "arn:aws:iam::<aws-account-id>:role/<role-name>" with the ARN of your IAM role, and execute the script. This will load the data set from S3 into your Redshift cluster. Expect the script to take a few minutes to complete. The customer and time dimension consists of **3M records, and 2556 records** respectively.

10. Next, we will create an "external schema" that references datasets that reside outside of your Redshift cluster. Define this schema by running the following command:

    CREATE EXTERNAL SCHEMA clickstream from data catalog database '<your_uniqueId>_rs_spectrum_clickstreams' iam_role 'arn:aws:iam::<aws-account-id>:role/<role-name>' **CREATE EXTERNAL DATABASE IF NOT EXISTS**;

    You will need to replace the section 'arn:aws:iam::<aws-account-id>:role/<role-name>' with the ARN of the IAM role that you used in the previous step. Also replace the substring, "<your_uniqueId>" with a unique label to identify your external database. Redshift stores the meta-data that describes your external databases and schemas in the AWS Glue data catalog by default. Once created, you can view the schema from Glue or Athena. The namespace within Athena is shared by your account, so we need to ensure your database is uniquely named among all the lab participants.

11. Ahead of the lab, you workshop facilitators should have prepared two S3 buckets that contain clickstream data. In the next step, we will define external tables that reference these buckets.

    Run the following SQL scripts to create your two tables:

    - create-clickstream-csv10.sql
    - create-clickstream-parquet1.sql

    The main parts of these scripts consist of a command to create the table like the one below:

    CREATE EXTERNAL TABLE clickstream.uservisits_parquet1(
    custKey int4,
    yearmonthKey int4,
    visitDate int4,
    adRevenue float,
    countryCode char(3),
    destURL varchar(100),
    duration int4,
    languageCode char(6),
    searchWord varchar(32),
    sourceIP varchar(116),
    userAgent varchar(256))


```
PARTITIONED BY(customer int4, visitYearMonth int4)
STORED AS parquet
LOCATION 's3://redshift-spectrum-datastore-parquet1/'
TABLE PROPERTIES ('numRows'='3758774345');
```

The command is followed by 504 commands that add partitions to the external table. Below is an example of one of the commands:

```
ALTER TABLE clickstream.uservisits_parquet1 add partition(customer=1, visitYearMonth=199201)
LOCATION 's3://redshift-spectrum-datastore-
parquet1/52a17f02aa5675c8399b182d9351da5a79b0522ca1080270c15b1767031babf4/customer
=1/visitYearMonth=199201/';
```

These commands reference an S3 bucket called **s3://redshift- spectrum-datastore-parquet1** as highlighted in the examples above. You will need modify these scripts by replacing this string with the S3Uri corresponding to the buckets that were created for your lab. Your facilitators should provide you with this info. These scripts will take a couple of minutes to complete.

These external tables contain **nearly 3.8B records of clickstream data.**

Section 3: Querying the S3 Data Lake and Performance Diagnostics

In this section, we will learn how to blend data in a S3 data lake and a Redshift data warehouse by querying and joining the tables that we created in the previous section. Additionally, you will learn the diagnostic tools available to help performance tune your Redshift Spectrum queries.

Query with Redshift Spectrum

1. Run the query below from your SQL Workbench query editor. This query performs a join between dimension tables in Redshift, and the clickstream fact table in S3 effectively blending data from the data Lake and data warehouse:

```
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, SUM(uv.adRevenue)
FROM clickstream.uservisits_csv10 as uv
RIGHT OUTER JOIN
customer as c
ON c.c_custkey = uv.custKey
INNER JOIN
(SELECT DISTINCT d_yearmonthnum, (d_month||','||d_year) as prettyMonthYear
FROM dwdate WHERE d_yearmonthnum >= 199810) as t
ON uv.yearMonthKey = t.d_yearmonthnum
WHERE c.c_custkey <= 3
GROUP BY c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.yearMonthKey
ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey  ASC
```

- **Data lake**
- **Data warehouse**

Expect this query to take a few minutes to complete as nearly **3.8B records will be accessed**. The results of the query should be as follows:

c_name	c_mktsegment	Prettymonthyear	totalrevenue
Customer#000000001	BUILDING	October,1998	3596847.84
Customer#000000001	BUILDING	November,1998	3776957.04
Customer#000000001	BUILDING	December,1998	3674480.43
Customer#000000002	AUTOMOBILE	October,1998	3593281.28
Customer#000000002	AUTOMOBILE	November,1998	3777930.64
Customer#000000002	AUTOMOBILE	December,1998	3671834.14
Customer#000000003	AUTOMOBILE	October,1998	3596234.31
Customer#000000003	AUTOMOBILE	November,1998	3776715.02
Customer#000000003	AUTOMOBILE	December,1998	3674360.28

This query returns the total ad revenue in the last 3 months of our dataset by market segment for customers 1 to 3. The ad revenue data originates from S3 while the customer and time attributes like market segment originate from the dimension tables in Redshift.

Performance Diagnostics

There are two key utilities that provide visibility into Redshift Spectrum:

- EXPLAIN: provides the query execution plan, which includes info around what processing is pushed down to Spectrum. Steps in the plan that include the prefix S3 are executed on Spectrum; for instance, the plan for the query above has a step *"S3 Seq Scan clickstream.uservisits_csv10*" indicating that Spectrum performs a scan on S3 as part of the query execution.

- SVL_S3QUERY_SUMMARY: statistics for Redshift Spectrum queries are stored in this table. While the execution plan presents cost estimates, this table stores actual statistics of past query runs.

1. Run the following query on the SVL_S3QUERY_SUMMARY table:

```
select query, elapsed, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows,
s3query_returned_bytes, files, avg_request_parallelism
from svl_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

The query should return results similar to:

Query	elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
1652	209773697	3758774345	6.61358E+11	66270117	1060321872	5040	9.77

The diagnostics reveal why our query took so long. For instance, "s3_scanned_row" reveals that the query scans nearly 3.8B records, which is the entire data set.

2. Run the same Redshift Spectrum query again, but with EXPLAIN:

```
EXPLAIN
SELECT
….
ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey  ASC;
```

The output will look similar to the example below. Don't worry about understanding the details of the query plan at this time. However, take note of the section highlighted in red below.

```
QUERY PLAN
XN Merge  (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
Merge Key: c.c_name, c.c_mktsegment, uv.yearmonthkey
->  XN Network  (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
Send to leader
->  XN Sort  (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
Sort Key: c.c_name, c.c_mktsegment, uv.yearmonthkey
->  XN HashAggregate  (cost=141056760589.25..141056761639.25 rows=420000 width=78)
->  XN Hash Join DS_DIST_ALL_NONE  (cost=37.34..141056596142.86 rows=13155711 width=78)
Hash Cond: ("outer".yearmonthkey = "inner".d_yearmonthnum)
->  XN Hash Join DS_DIST_ALL_NONE  (cost=4.50..141051766084.98 rows=375877435 width=46)
Hash Cond: ("outer".custkey = "inner".c_custkey)
->  XN Partition Loop  (cost=0.00..94063327993.62 rows=3758774345000 width=16)
->  XN Seq Scan PartitionInfo of clickstream.uservisits_csv10 uv  (cost=0.00..10.00 rows=1000 width=0)
->  XN S3 Query Scan uv  (cost=0.00..93969358.62 rows=3758774345 width=16)
->  S3 Seq Scan clickstream.uservisits_csv10 uv location:"s3://redshift-spectrum-datastore-csv10" format:TEXT  (cost=0.00..56381615.17 rows=3758774345 width=16)
Filter: ((custkey <= 3) AND (yearmonthkey >= 199810))
->  XN Hash  (cost=3.75..3.75 rows=300 width=38)
->  XN Seq Scan on customer c  (cost=0.00..3.75 rows=300 width=38)
Filter: (c_custkey <= 3)
->  XN Hash  (cost=32.82..32.82 rows=7 width=36)
->  XN Subquery Scan t  (cost=0.00..32.82 rows=7 width=36)
->  XN Unique  (cost=0.00..32.75 rows=7 width=18)
->  XN Seq Scan on dwdate  (cost=0.00..32.43 rows=64 width=18)
Filter: (d_yearmonthnum >= 199810)
```

The takeaway is that the query plan reveals how Redshift Spectrum is leverage in the query. The section in red indicates that Redshift Spectrum is leveraged as part of the query execution to perform a scan. It also reveals that our partitions weren't used. We will explore this in more detail in the next part of the lab.

Section 4: Optimizing Performance with Partitions

In this section, you will learn about partitions, and how they can be used to improve the performance of your Redshift Spectrum queries.

Partitioning is a key means to improving scan efficiency. Previously, we ran a script to create our external tables along with partitions.

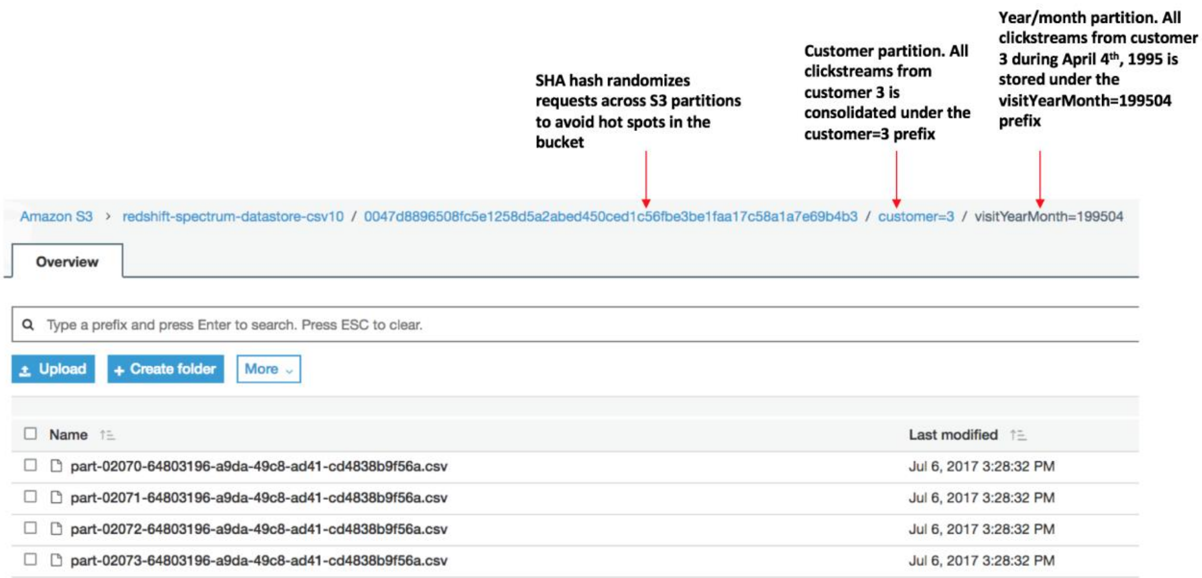
The script included a clause in our table creation statement to define a partition as highlighted in green:

```
CREATE EXTERNAL TABLE clickstream.uservisits_csv10
…
PARTITIONED BY(customer int4, visitYearMonth int4)
```

It was followed by 504 statements to define the individual partitions for each customer and year/month combination:

```
ALTER TABLE clickstream.uservisits_csv10
ADD PARTITION(customer=1, visitYearMonth=199201)
LOCATION 's3://redshift-spectrum-datastore-csv10/52a17f02aa5675c8399b182d9351da5a79b0522ca1080270c15b1767031babf4/customer=1/visitYearMonth=199201/';
```

If you have interest in understanding the details of how partitions were setup, refer to the [documentation](), and explore the S3 buckets that are serving our Redshift Spectrum datasets. The data records are stored in batches within CSV and Parquet files, and are organized in the hierarchy illustrated below:



In essence, the entire 3.8 billion-row dataset is organized as a collection of large files where each file contains data exclusive to a particular customer and month in a year. This allows you to partition your data into logical subsets by customer and year/month as exemplified above. With partitions, the query engine can target a subset of files:

- Only for specific customers
- Only data for specific months
- A combination of specific customers and year/months

Take note that the right choice of partitions is dependent on your workload. Partitions should be selected based on the primary queries you want to optimize, and your data profile. For those implementing their own clickstream analytics, a partition scheme like year/month/region often makes sense. The choice of using customer in the partition scheme isn't optimal for a use case where there is a very large number of customers, and little data for each one. The data set, and scheme used in this example is a practical one for scenarios like a multi-tenant ad-tech platform, or an IoT platform. In these cases, there are a moderate number of customers (tenants), and a lot of data per customer.

1. Observe the effects of leveraging partitioning on our query by running the following [query]().

```
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, SUM(uv.adRevenue)
FROM clickstream.uservisits_csv10 as uv
RIGHT OUTER JOIN
customer as c
ON c.c_custkey = uv.customer
INNER JOIN
```

```
(SELECT DISTINCT d_yearmonthnum, (d_month||','||d_year) as prettyMonthYear
FROM dwdate WHERE d_yearmonthnum >= 199810) as t
ON uv.yearMonthKey = t.d_yearmonthnum
WHERE c.c_custkey <= 3
GROUP BY c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.yearMonthKey
ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey  ASC
```

The join condition from the previous query has been modified. This change has been highlighted in green above. Instead of joining on the synthetic key, custKey, we use the partition key, customer, that we created as part of the data modeling process. This query should run approximately **2X faster** than the previous.

3. Run the same Redshift Spectrum query again, but with EXPLAIN. Unlike before, you should see a Filter clause as part of the PartitionInfo scan that indicates partition pruning is executed as part of the query plan:

```
…
  -> XN Seq Scan PartitionInfo of clickstream.uservisits_csv10 uv  (cost=0.00..12.50 rows=334 width=4)
           Filter: ((customer <= 3) AND (subplan 4: (customer = $2)))
…
```

2. Re-run SVL_S3QUERY_SUMMARY:

```
select query, elapsed, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows,
s3query_returned_bytes, files, avg_request_parallelism
from svl_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

You should observe the following results:

Query	elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
5534	113561847	1898739653	3.34084E+11	66270117	795241404	2520	9.71

Note that “s3_scanned_rows” reveals that the rows scanned has been halved when compared with the previous query. This explains why our query ran roughly twice as fast.

The results are due to the fact that our data is evenly distributed across all customers, and by querying 3 of 6 customers with our customer partition key, the database engine is able to intelligently scan the subset of data containing customers 1,2 and 3 instead of the entire data set. However, the scan is still very inefficient, and we can benefit from utilizing our year/month partition key as well.

3. Run the query below:

```
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, SUM(uv.adRevenue)
FROM clickstream.uservisits_csv10 as uv
RIGHT OUTER JOIN
customer as c
ON c.c_custkey = uv.customer
INNER JOIN
(SELECT DISTINCT d_yearmonthnum, (d_month||','||d_year) as prettyMonthYear
FROM dwdate WHERE d_yearmonthnum >= 199810) as t
ON uv.visitYearMonth = t.d_yearmonthnum
WHERE c.c_custkey <= 3
```



```
GROUP BY c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.yearMonthKey
ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey  ASC
```

Our latest query utilizes both customer and time partitions. If you run this query a few times, you should see execution time in the range of 8s, which is a 22.5X improvement on our original query!

4. Re-run [SVL_S3QUERY_SUMMARY:]

```
select query, elapsed, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows,
s3query_returned_bytes, files, avg_request_parallelism
from svl_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

Upon reviewing the statistics for this query, you should observe that Redshift Spectrum scans and returns the exact number of rows (66,270,117) required to compute the query.

Query	Elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
1939	7124877	66270117	11660676734	66270117	795241404	90	5.87

Section 5: Storage Optimizations

Redshift Spectrum performs processing through large-scale infrastructure external to your Redshift cluster. It is optimized for performing large scans and aggregations on S3; in fact, with the proper optimizations, Redshift Spectrum may even out-perform a small to medium size Redshift cluster on these types of workloads. There are two important variables to consider for optimizing large scans and aggregations:

- **File size and count.** As a general rule, use files sizes between 50-500MB for non-splittable files, this is optimal for Redshift Spectrum. However, the number of files operating on a query is directly correlated with the parallelism achievable by a query. There is an inverse relationship between file size and count: the bigger the files, the fewer files there are for the same dataset. Consequently, there is a trade-off between optimizing for object read performance, and the amount of parallelism achievable on a particular query. Large files are best for large scans as the query likely operates on sufficiently large number of files. For queries that are more selective and for which fewer files are operating, you may find that smaller files allow for more parallelism.
- **Data format.** Redshift Spectrum supports various data formats. Columnar formats like Parquet can sometimes lead to substantial performance benefits by providing compression and more efficient I/O for certain workloads. Generally, format types like Parquet should be used for query workloads involving large scans, and high attribute selectivity. Again, there are trade-offs as formats like Parquet require more compute power to process than plaintext. For queries on smaller subsets of data, the I/O efficiency benefit of Parquet is diminished. At some point, Parquet may perform the same or slower than plaintext. Latency, compression rates, and the trade-off between user experience and cost should drive your decision. In most cases, formats like Parquet is optimal.

To help illustrate how Spectrum performs on these large aggregation workloads, let's consider a basic query that aggregates the entire 3.7B+ record dataset on Redshift Spectrum, and compared that with running the query exclusively on Redshift:

```
SELECT uv.custKey, COUNT(uv.custKey)
FROM <your clickstream table> as uv
GROUP BY uv.custKey
ORDER BY uv.custKey ASC
```

In the interest of time, we won't go through this exercise in the lab; nonetheless, it is helpful to understand the results of running this test.

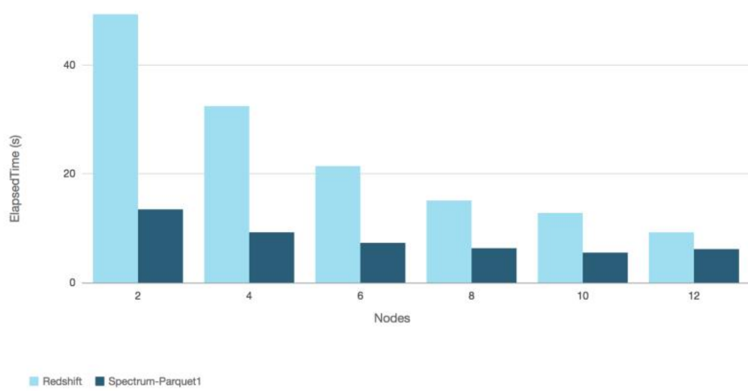
For the Redshift-only test case, the clickstream data is loaded, and distributed evenly across all nodes (even distribution style) with optimal column compression encodings prescribed by Redshift's ANALYZE command.

The Redshift Spectrum test case utilizes a Parquet data format with one file containing all the data for a particular customer in a month; this results in files mostly in the range of 220-280MB, and in effect, is the largest file size for this partitioning scheme. If you run tests with the other datasets provided, you will see that this data format and size is optimal and will out-perform others by 60X+.

Take heed that the presented quantifications shouldn't be applied generically as performance differences will vary depending on scenario. Instead take note of the testing strategy, the evidence, and the characteristics of the workload where Spectrum is likely to yield performance benefits.

Chart 1 below compares the query execution time for the two scenarios. The results indicate that you will need to pay for 12 X DC1.Large nodes to get performance comparable to using Spectrum with the support of a small Redshift cluster in this particular scenario.

Chart 1: Simple Aggregation on 3.7B+ Records



Section 6: Predicate Pushdown

In the last section, we learned that Spectrum excels at performing large aggregations. In this section, we'll experiment the results of pushing more work down to Redshift Spectrum.

1. Run the following query:

```
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.totalRevenue
FROM
((SELECT customer, visitYearMonth, SUM(adRevenue) as totalRevenue
FROM clickstream.uservisits_parquet1
WHERE customer <= 3 and visitYearMonth >= 199810
GROUP BY  customer, visitYearMonth) as uv
RIGHT OUTER JOIN
customer as c
ON c.c_custkey = uv.customer
INNER JOIN
(SELECT DISTINCT d_yearmonthnum, (d_month||','||d_year) as prettyMonthYear
FROM dwdate WHERE d_yearmonthnum >= 199810) as t
ON uv.visitYearMonth = t.d_yearmonthnum)
ORDER BY c.c_name, c.c_mktsegment, uv.visitYearMonth ASC;
```

After running this query a few times, you should observe execution times in the range of 4 seconds.

This query improves on our previous one in a couple of ways.

1. Noticed that we are querying the clickstream.uservisits_parquet1 table instead of clickstream.uservisits_csv10. These two tables contain the same data set, but they have been processed in different ways. The table clickstream.uservisits_parquet1 contains data in parquet format. Parquet is a columnar format, and yields I/O benefits for analytical workloads by providing compression and efficient retrieval of the attributes that are selected by the queries. Furthermore, the “1” vs “10” suffix indicates that all the data for each partition is stored in a single file instead of ten files like the CSV data set. The latter case has less overhead involved in processing large scans and aggregations.

2. The query part highlighted in green pushes the aggregation work down to Redshift Spectrum. When we analyzed the query plan previously, we observed that Spectrum is used for scanning. When you analyze the above query, you will see that aggregations are also performed at the Spectrum layer.

2. Re-run [SVL_S3QUERY_SUMMARY:](SVL_S3QUERY_SUMMARY)

```
select query, elapsed, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows,
s3query_returned_bytes, files, avg_request_parallelism
from svl_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

You obtain the following results:

query	Elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	Files	avg_request_parallelism
1946	1990509	66270117	**531159030**	**9**	**72**	9	0.88

The statistics reveal the source of some of the performance improvements:

- The bytes scanned is reduced even though the same number of rows are scanned as a result of compression.

- The number of rows returned is reduced to 9 from ~66.3M. This results in only 72 bytes returned from the Spectrum layer versus 795MBs. This is the result of pushing the aggregation down to the Spectrum layer. Our data is stored at the day-level granularity, and our query rolls that up to the month-level. By pushing the aggregation down to the Spectrum fleet, we only need to return 9 records that aggregate ad revenue up to the month level so that they can be joined with the required dimension attributes.

3. Run the query again with EXPLAIN:

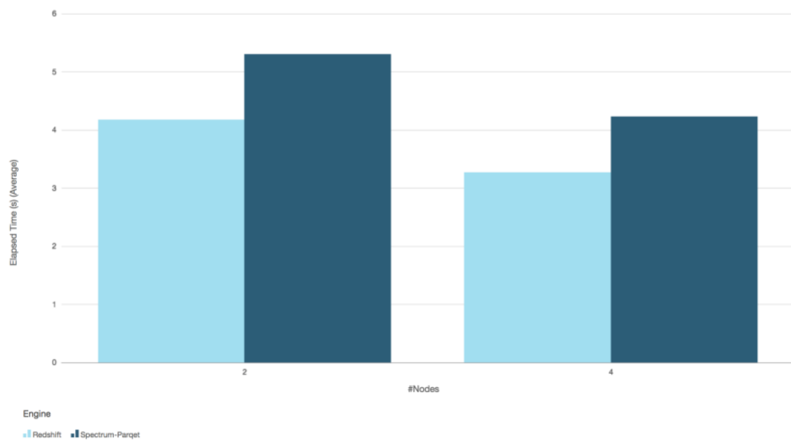
   The query plan should include an “S3 Aggregate” step, which indicates that the Spectrum layer offloads the aggregation processing for this query.

```
QUERY PLAN
XN Merge  (cost=1000094008880.16..1000094008880.18 rows=7 width=78)
  Merge Key: c.c_name, c.c_mktsegment, uv.visityearmonth
  ->  XN Network  (cost=1000094008880.16..1000094008880.18 rows=7 width=78)
        Send to leader
        ->  XN Sort  (cost=1000094008880.16..1000094008880.18 rows=7 width=78)
              Sort Key: c.c_name, c.c_mktsegment, uv.visityearmonth
              ->  XN Hash Join DS_DIST_ALL_NONE  (cost=94008878.97..94008880.06 rows=7 width=78)
                    Hash Cond: ("outer".customer = "inner".c_custkey)
                    ->  XN Hash Join DS_DIST_ALL_NONE  (cost=93971378.97..93971379.61 rows=7 width=48)
                          Hash Cond: ("outer".visityearmonth = "inner".d_yearmonthnum)
                          ->  XN Subquery Scan uv  (cost=93971346.13..93971346.42 rows=23 width=16)
                                ->  XN HashAggregate  (cost=93971346.13..93971346.19 rows=23 width=16)
                                      ->  XN Partition Loop  (cost=93969358.63..93970506.13 rows=112000 width=16)
                                            ->  XN Seq Scan PartitionInfo of clickstream.uservisits_parquet1  (cost=0.00..17.50 rows=112 width=8)
                                                  Filter: ((customer <= 3) AND (visityearmonth >= 199810) AND (visityearmonth >= 199810))
                                            ->  XN S3 Query Scan uservisits_parquet1  (cost=46984679.32..46984689.32 rows=1000 width=8)
                                                  ->  S3 Aggregate  (cost=46984679.32..46984679.32 rows=1000 width=8)
                                                        ->  S3 Seq Scan clickstream.uservisits_parquet1 location:"s3://redshift-spectrum-datastore-parquet1" format:PARQUET  (cost=0.00..37587743.45 rows=3758774345 width=8)
                          ->  XN Hash  (cost=32.82..32.82 rows=7 width=36)
                                ->  XN Subquery Scan t  (cost=0.00..32.82 rows=7 width=36)
                                      ->  XN Unique  (cost=0.00..32.75 rows=7 width=18)
                                            ->  XN Seq Scan on dwdate  (cost=0.00..32.43 rows=64 width=18)
                                                  Filter: (d_yearmonthnum >= 199810)
                    ->  XN Hash  (cost=30000.00..30000.00 rows=3000000 width=38)
                          ->  XN Seq Scan on customer c  (cost=0.00..30000.00 rows=3000000 width=38)
```

Section 7: Native Redshift versus Redshift with Spectrum Performance

At this point, you might be asking yourself, why would I ever not use Spectrum? Well, you still get additional value from loading data into Redshift.

In fact, it turns out that our last query runs even faster when executed exclusively in native Redshift. Running a full test is beyond the time we have for the lab, so let’s review test results that compares running the last query with Redshift Spectrum versus exclusively with Redshift on various cluster sizes.



As a rule of thumb, queries that aren't dominated by I/O and involve multiple joins are better optimized in native Redshift.

Furthermore, the variability in latency in native Redshift is significantly lower. For use cases where you have tight performance SLAs on queries, you may want to consider using Redshift exclusively to support those queries.

On the other hand, when you have the need to perform large scans, you could benefit from the best of both worlds: higher performance at lower cost. For instance, imagine we needed to enable our business analysts to interactively discover insights across a vast amount of historical data.

1. Instead of running our previous query on 3 months of data, let's perform the analysis on 7-years. The query is as follows:

```
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.totalRevenue
FROM
((SELECT customer, visitYearMonth, SUM(adRevenue) as totalRevenue
FROM clickstream.uservisits_parquet1
WHERE customer <= 3 and visitYearMonth >= 199201
GROUP BY  customer, visitYearMonth) as uv
RIGHT OUTER JOIN
customer as c
ON c.c_custkey = uv.customer
INNER JOIN
(SELECT DISTINCT d_yearmonthnum, (d_month||','||d_year) as prettyMonthYear
FROM dwdate WHERE d_yearmonthnum >= 199201) as t
ON uv.visitYearMonth = t.d_yearmonthnum)
ORDER BY c.c_name, c.c_mktsegment, uv.visitYearMonth ASC;
```

   The sections in green highlight the modifications. Unlike our previous queries, the time range filter starts at January, 1992 instead of October, 1998. This query should run in under 10 seconds after it is executed a few times.

2. Inspect the SVL_S3QUERY_SUMMARY again by running the query:

```
select query, elapsed, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows,
s3query_returned_bytes, files, avg_request_parallelism
from svl_s3query_summary
where query = pg_last_query_id()
```

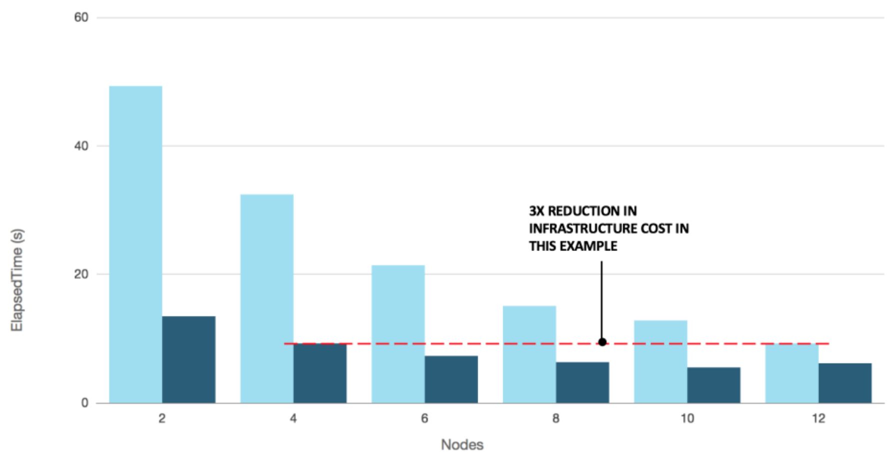

order by query,segment;

You should observe the results below. Note that this query scans nearly 1.9B records, which is half the data set to aggregate data across 7-years.

query	elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
6195	9005338	**1898739653**	15217906256	252	2016	252	8.15

This is a lot of data to process, yet the query runs in under 10 seconds. This is substantially faster that the queries we ran at the start of the lab, which queried the same result set. The difference in performance is a result of the improvements we made to the data format, size and pushing the aggregation work down to the Spectrum layer.

We don't have the time to do a full performance comparison between running this query exclusively in Redshift versus leveraging Redshift Spectrum, but if you did, you expect to see results similar to the what is presented in the chart below:



For this particular query, leveraging Spectrum has substantial cost and performance benefit. We achieve better performance with a fraction of the cost. The chart shows that we get performance from 4 X DC1.Large + Spectrum similar to native redshift and 12 X DC1.Large.

Also, note that the performance for Spectrum plateaus in the chart above. If the query involved aggregating data from more files, we would see a continued linear improvement in performance as well.

Before You Leave

Once you are done with the lab, decommission your resources by deleting your CloudFormation stack.