

PostgreSQL Snapper Lab

The [PostgreSQL Snapper tool](#) enables periodic collection (snapping) of PostgreSQL performance related statistics and metrics. The config file used by the tool can be customized to add and remove database dictionary views and queries to be snapped as required. Snapper collects and stores the PostgreSQL database metrics in separate OS level files to have minimal impact on the database. These files can be loaded into another PostgreSQL instance by the loader script for doing analysis.

Snapper tool can be installed following the **Quick Start** instructions documented in [Snapper Github](#). For the purpose of “**Aurora PostgreSQL Performance Package**” workshop, the Snapper CloudFormation stack is already setup in your AWS Account.

! Note: To allow Snapper to collect PostgreSQL statistics while a load test is running or database is doing some activity, you need to run this lab **in conjunction** with [Lab6:RDS Performance Insights](#) in which you will generate database load using pgbench. The “**Setup and Configuration**” section below should be completed before you start [Lab6:RDS Performance Insights](#) to schedule Snapper to run every 1 minute and capture workload statistics. Then head to [Lab6:RDS Performance Insights](#) and complete it. After [Lab6:RDS Performance Insights](#) is complete, you can come back to this lab to finish rest of the steps.

The lab contains the following tasks and it should take ~**30 minutes** to complete excluding the time needed to complete [Lab6:RDS Performance Insights](#).

1. **Setup and Configuration:** Schedule Snapper to run every 1 minute using an EC2 instance.
2. **Generate Load on the PostgreSQL database:** Head to [Lab6:RDS Performance Insights](#) and complete it to generate load on Aurora PostgreSQL using pgbench.
3. **Package Snapper output:** Once the load test is over, package the Snapper output so that it can be loaded to another PostgreSQL database.
4. **Import Snapper output:** Import Snapper output to another PostgreSQL database. For the purpose of this lab, we will use the same Aurora PostgreSQL cluster from which we captured the statistics.
5. **Analyze performance metrics of the PostgreSQL database:** Snapper comes with a set of sample queries for analysis. You will use some of those queries to do analysis and derive insights like top tables and indexes by size, top SQLs by elapsed time, top tables by sequential scans, Foreign Keys with no indexes and Table bloat analysis.

Setup and Configuration

In this step, you will complete the pre-requisites for Snapper and schedule it using an EC2 instance to run every 1 minute.

1. Make sure the pre-requisites <https://aurora-pg-lab.workshop.aws/lab1.5-client.html> are complete before proceeding.
2. Open a Cloud9 terminal window and run the following commands to install `pg_stat_statements` and `aurora_stat_utils` extensions. `pg_stat_statements` module provides a means for tracking execution statistics of all SQL statements executed by a server. `aurora_stat_utils` extension provides aurora wait and log related statistics.

```
psql
create extension IF NOT EXISTS pg_stat_statements;
create extension IF NOT EXISTS aurora_stat_utils;
\dx
```

3. In the [CloudFormation console](#), select the CloudFormation stack with description “Amazon Aurora PostgreSQL Labs Stackset” and go to the Output tab. Take a note of the value for the following CloudFormation Output Keys. You will need them during the later steps.

SnapperEC2InstanceID - This is the EC2 instance ID where snapper was setup.

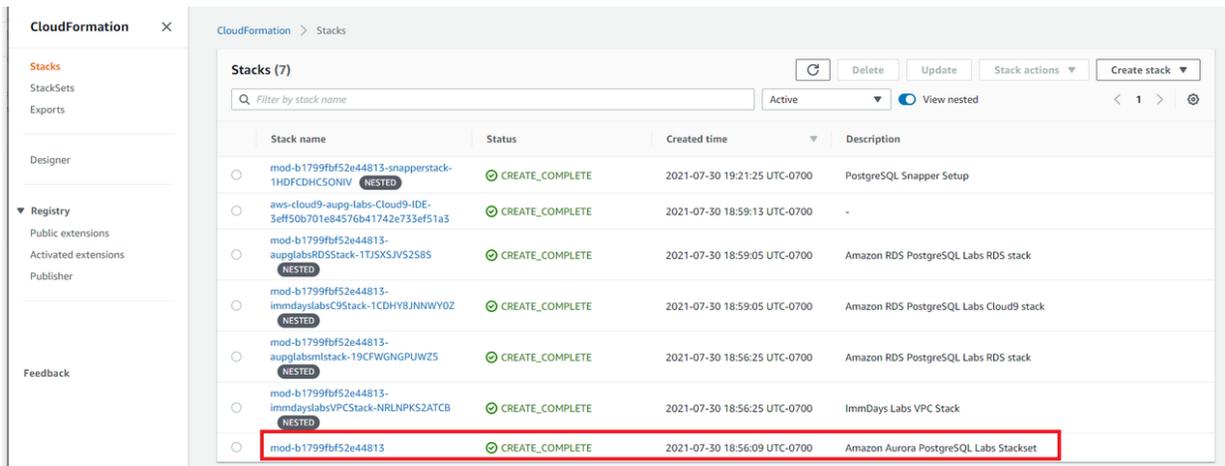
clusterEndpoint - Aurora PostgreSQL cluster endpoint

Port - Aurora PostgreSQL cluster port

DatabaseName - Aurora PostgreSQL database name

DBUsername - Aurora PostgreSQL master user name

SnapperSecretARN - AWS Secrets Manager secret ARN storing Aurora PostgreSQL master user password



The screenshot shows the AWS CloudFormation console interface. On the left, there is a navigation sidebar with options like 'Stacks', 'StackSets', 'Exports', 'Designer', 'Registry', and 'Feedback'. The main area displays a table of stacks. The table has columns for 'Stack name', 'Status', 'Created time', and 'Description'. The stack 'mod-b1799fbf52e44813-Amazon Aurora PostgreSQL Labs Stackset' is highlighted with a red border.

Stack name	Status	Created time	Description
mod-b1799fbf52e44813-snapperstack-1HDFCDHCSONIV	CREATE_COMPLETE	2021-07-30 19:21:25 UTC-0700	PostgreSQL Snapper Setup
aws-cloud9-aupg-labs-Cloud9-IDE-3ef50b701e84576b41742e733ef51a3	CREATE_COMPLETE	2021-07-30 18:59:13 UTC-0700	-
mod-b1799fbf52e44813-aupglabsRDSStack-TJJSXSJV52S85	CREATE_COMPLETE	2021-07-30 18:59:05 UTC-0700	Amazon RDS PostgreSQL Labs RDS stack
mod-b1799fbf52e44813-immdaylabsC9Stack-1CDHY8JNNWYOZ	CREATE_COMPLETE	2021-07-30 18:59:05 UTC-0700	Amazon RDS PostgreSQL Labs Cloud9 stack
mod-b1799fbf52e44813-aupglabsm1stack-19CFWGNGPUWZ5	CREATE_COMPLETE	2021-07-30 18:56:25 UTC-0700	Amazon RDS PostgreSQL Labs RDS stack
mod-b1799fbf52e44813-immdaylabsVPCStack-NRLNPKSZATCB	CREATE_COMPLETE	2021-07-30 18:56:25 UTC-0700	ImmDays Labs VPC Stack
mod-b1799fbf52e44813	CREATE_COMPLETE	2021-07-30 18:56:09 UTC-0700	Amazon Aurora PostgreSQL Labs Stackset

CloudFormation > Stacks > mod-b1799fbf52e44813

mod-b1799fbf52e44813

Stack info | Events | Resources | **Outputs** | Parameters | Template | Change sets

Stacks (7)

- mod-b1799fbf52e44813-snapperstack-1HDFCDHCSONIV
- aws-cloud9-aupp-labs-Cloud9-IDE-3eff50b701e84576b41742e733ef51a3
- mod-b1799fbf52e44813-aupglabsRDSStack-1TJXSJVS2S85
- mod-b1799fbf52e44813-immdayslabsC9Stack-1CDHY8JNNWYOZ
- mod-b1799fbf52e44813-aupglabslmstack-19CFWGNPUIWZ5
- mod-b1799fbf52e44813-immdayslabsVPStack-NRLNPKS2ATCB
- mod-b1799fbf52e44813**

Outputs (14)

Key	Value	Description	Export name
Cloud9URL	https://us-west-2.console.aws.amazon.com/cloud9/ide/3eff50b701e84576b41742e733ef51a3	Cloud9 URL	-
DBSecGroup	sg-0f609a18373ea6cd9	Database Security Group	-
DBUsername	masteruser	Database master username	-
DatabaseName	mylab	Database Name	-
LabVPC	vpc-0b1f8bf65679056ed	Aurora PostgreSQL Lab VPC	-
Port	5432	Aurora Endpoint Port	-
SnapperEC2InstanceID	i-08b3666a939bece35	EC2 Instance ID for PG Snapper	-
SnapperS3Bucket	s3://mod-b1799fbf52e44813-snapperstack-1hdfcd-s3bucket-1qidhrfz6c1m/	S3 bucket to store PG Snapper output for sharing	-
SnapperSecretARN	arn:aws:secretsmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1799fbf52e44813-snapperstack-1HDFCDHCSONIV-8P54cl	Master User Secret ARN for the PostgreSQL Instance to be monitored	-
apgcustomclusterparamgroup	mod-b1799fbf52e44813-aup-apgcustomclusterparamgro-wqyltpr85s0	Cluster Parameter Group	-
apgcustomdbparamgroup	mod-b1799fbf52e44813-aupglabslmstack-1tjxsjvs2s8s-apgcustomdbparamgroup-13m4wxaznfekl	Database Parameter Group	-
clusterEndpoint	aupp-labs-cluster.cluster-cvmeikrm7rz.us-west-2.rds.amazonaws.com	Aurora Cluster Endpoint	-
readerEndpoint	aupp-labs-cluster.cluster-ro-cvmeikrm7rz.us-west-2.rds.amazonaws.com	Aurora Reader Endpoint	-
secretArn	arn:aws:secretsmanager:us-west-2:953779585674:secret:secretDBMasterUser-mAn3pjEAqndW-YlMFz6	Database Credentials Secret ARN	-

4. In the [EC2 Dashboard](#) select the Snapper EC2 instance and click Connect. Select **Session Manager** tab and click **Connect** again.

EC2 Dashboard > Instances (1/2)

Filter instances

Instance state: running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Z
aws-cloud9-aupp-labs-Cloud9-IDE-3eff50b701e84576b41742e733ef51a3	i-0ba1db1817dbf8d35	Running	t3.small	2/2 checks passed	No alarms	us-west-2a
pg-snapper-mod-b1799fbf52e44813-snapperstack-1HDFCDHCSONIV	i-08b3666a939bece35	Running	t3.medium	2/2 checks passed	No alarms	us-west-2a

EC2 > Instances > i-08b3666a939bece35 > Connect to instance

Connect to instance

Connect to your instance i-08b3666a939bece35 (pg-snapper-mod-b1799fbf52e44813-snapperstack-1HDFCDHCSONIV) using any of these options

EC2 Instance Connect | **Session Manager** | SSH client | EC2 Serial Console

Session Manager usage:

- Connect to your instance without SSH keys or a bastion host.
- Sessions are secured using an AWS Key Management Service key.
- You can log session commands and details in an Amazon S3 bucket or CloudWatch Logs log group.
- Configure sessions on the Session Manager Preferences page.

Cancel | **Connect**

5. Session Manager uses **ssm-user** user to connect to the EC2 instance by default. Change user to **ec2-user** by running the following command:

```
sudo su -l ec2-user
```

6. Review the Snapper script usage by running the following command.

```
[ec2-user@ip-172-31-14-11 ~]$ /home/ec2-user/scripts/pg_perf_stat_snapper.py -h

usage: pg_perf_stat_snapper.py [-h] -e ENDPOINT -P PORT -d DBNAME -u USER -s
                               SECRETARN -m MODE [-o OUTPUTDIR] -r REGION

Snap PostgreSQL performance statistics and exit

optional arguments:
  -h, --help            show this help message and exit
  -e ENDPOINT, --endpoint ENDPOINT
                        PostgreSQL Instance Endpoint (default: None)
  -P PORT, --port PORT  Port (default: None)
  -d DBNAME, --dbname DBNAME
                        Database Name where Application objects are stored
                        (default: None)
  -u USER, --user USER Database UserName (default: None)
  -s SECRETARN, --SecretARN SECRETARN
                        AWS Secrets Manager stored Secret ARN (default: None)
  -m MODE, --mode MODE  Mode in which the script will run: Specify either snap
                        or package (default: None)
  -o OUTPUTDIR, --outputdir OUTPUTDIR
                        Output Directory (default:
                        /home/ec2-user/scripts/output)
  -r REGION, --region REGION
                        AWS region (default: None)
```

7. Run the Snapper script manually once using the following command and review the log file generated under **/home/ec2-user/scripts/log/** sub-directory. By default, all the output will be stored under **/home/ec2-user/scripts/output/** sub-directory.

```
/home/ec2-user/scripts/pg_perf_stat_snapper.py -e <PostgreSQL Instance EndPoint. Cloudformation Output Key: clusterEndpoint> -P <Database Port. Cloudformation Output Key: Port> -d <Database Name where Application objects are stored. Cloudformation Output key: DatabaseName> -u <Master UserName. Cloudformation Output Key: DBUsername> -s <AWS Secretes Manager ARN. Cloudformation Output Key: SnapperSecretARN> -m snap -r <AWS Region for e.g. us-west-2>
```

e.g.

```
home/ec2-user/scripts/pg_perf_stat_snapper.py -e aupg-labs-cluster.cluster-cvmeikrm7zrz.us-west-2.rds.amazonaws.com -P 5432 -d mylab -u masteruser -s arn:aws:secretsmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1799fbf52e44813-snapperstack-1HDFCDHC5ONIV-8PS4cI -m snap -r us-west-2
```

```

[ec2-user@ip-10-0-2-193 ~]$ /home/ec2-user/scripts/pg_perf_stat_snapper.py -e augg-labs-cluster.cluster-cvmeikm7zrz.us-west-2.rds.amazonaws.com -P 5432 -d mylab -u masteruser -s arn:aws:secretmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1795fbf52e44813-snapperstack-IHDFC8C5ONIV-8P84ci -m snap -r us-west-2
[ec2-user@ip-10-0-2-193 ~]$
[ec2-user@ip-10-0-2-193 ~]$
[ec2-user@ip-10-0-2-193 ~]$ ls -l /home/ec2-user/scripts/log/augg-labs-cluster.cluster-cvmeikm7zrz.us-west-2.rds.amazonaws.com/mylab/pg_perf_stat_snapper.log
-rw-rw-r-- 1 ec2-user ec2-user 8367 Jul 31 03:56 /home/ec2-user/scripts/log/augg-labs-cluster.cluster-cvmeikm7zrz.us-west-2.rds.amazonaws.com/mylab/pg_perf_stat_snapper.log
[ec2-user@ip-10-0-2-193 ~]$
[ec2-user@ip-10-0-2-193 ~]$ ls -l /home/ec2-user/scripts/output/augg-labs-cluster.cluster-cvmeikm7zrz.us-west-2.rds.amazonaws.com/mylab/
total 704
-rw-rw-r-- 1 ec2-user ec2-user 1211 Jul 31 03:56 aurora_log_report_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 204 Jul 31 03:56 pg_swr_snapshots.csv
-rw-rw-r-- 1 ec2-user ec2-user 5340 Jul 31 03:56 pg_locks_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 7503 Jul 31 03:56 pg_stat_activity_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 28916 Jul 31 03:56 pg_stat_all_indexes_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 20443 Jul 31 03:56 pg_stat_all_tables_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 164 Jul 31 03:56 pg_stat_bgwriter_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 1471 Jul 31 03:56 pg_stat_database_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 27815 Jul 31 03:56 pg_statio_all_indexes_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 13232 Jul 31 03:56 pg_statio_all_tables_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 0 Jul 31 03:56 pg_stat_progress_vacuum_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 348224 Jul 31 03:56 pg_stat_statements_history.csv
-rw-rw-r-- 1 ec2-user ec2-user 0 Jul 31 03:40 pg_temp_table_history.csv
[ec2-user@ip-10-0-2-193 ~]$

```

8. Schedule the Snapper script in crontab to run every 1 minute using crontab.

```
crontab -e
```

Press **i** to enter insert mode and Paste the following in the editor.

```

*/1 * * * * /home/ec2-user/scripts/pg_perf_stat_snapper.py -e <PostgreSQL Instance EndPoint. Cloudformation Output Key: clusterEndpoint> -P <Database Port. Cloudformation Output Key: Port> -d <Database Name where Application objects are stored. Cloudformation Output key: DatabaseName> -u <Master UserName. Cloudformation Output Key:DBUsername> -s <AWS Secretes Manager ARN. Cloudformation Output Key: SnapperSecretARN> -m snap -r <AWS Region for e.g. us-west-2>

```

Enter **:wq!** to save and exit the editor.

Verify that crontab was successfully installed by running the following command.

```
crontab -l
```

```

[ec2-user@ip-10-0-2-193 ~]$ crontab -l
*/1 * * * * /home/ec2-user/scripts/pg_perf_stat_snapper.py -e augg-labs-cluster.cluster-cvmeikm7zrz.us-west-2.rds.amazonaws.com -P 5432 -d mylab -u masteruser -s arn:aws:secretmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1795fbf52e44813-snapperstack-IHDFC8C5ONIV-8P84ci -m snap -r us-west-2
[ec2-user@ip-10-0-2-193 ~]$

```

Generate Load on the PostgreSQL database

Proceed to [Lab6:RDS Performance Insights](#) now and complete it to generate load on the database. Once you are done with that lab, come back to this lab to finish rest of the steps.

Package the Snapper output

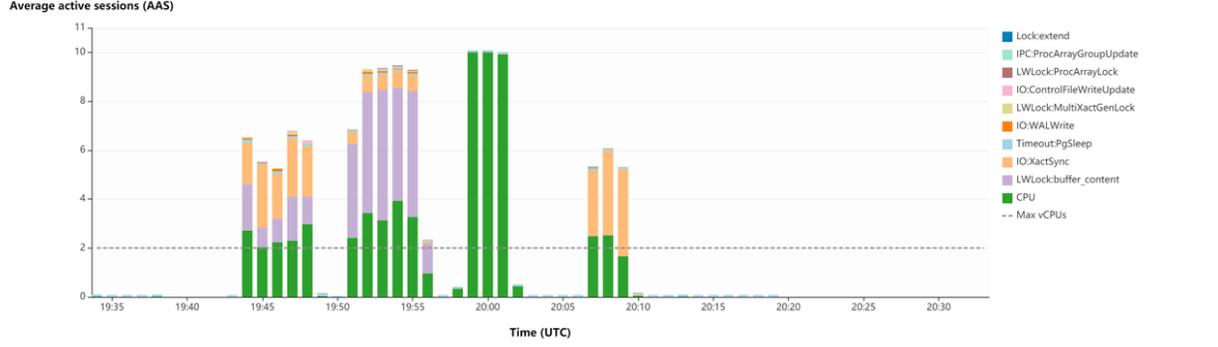
In this step, you will package the Snapper output so that it can be loaded to another PostgreSQL database.

By completing [Lab6:RDS Performance Insights](#), you generated some load on the Aurora PostgreSQL database as shown by RDS Performance Insights dashboard below.

Database load

Current activity measured in average active sessions (AAS)

Show max vCPU



Top waits | **Top SQL** | Top hosts | Top users | Top databases | Top applications

Top SQL (10) [Learn more](#)

Find SQL statements

Load by waits (AAS)	SQL statements	Calls/sec	Rows/sec	Blk hits/sec
0.92	select hr.add_employee_data(?)	48.47	48.47	9286.62
0.47	select hr.update_employee_data_fname(?)	0.03	0.03	9373.58
0.33	end	80.26	0.00	0.00
0.11	select hr.update_employee_data_empid(?)	28.90	28.90	1638.17
0.06	copy (select ?, * from aurora_log_report(?) to stdout (format csv)	0.00	0.00	0.00
0.03	select hr.update_employee_data_fname(?)	0.00	0.00	0.00
0.03	select hr.update_employee_data_fname(?)	0.00	0.00	0.00
0.01	select hr.update_employee_data_fname(?)	0.06	0.06	0.00
0.01	autovacuum: ANALYZE hr.employees	0.00	0.00	0.00
< 0.01	select hr.add_employee_data(?)	0.00	0.00	0.00

Since Snapper was scheduled using crontab, it has been collecting PostgreSQL statistics every 1 minute. Go ahead and comment out the Snapper job in crontab since we have already collected all the required PostgreSQL performance statistics for analysis. We will package the output next so that we can load it into a PostgreSQL database for processing.

1. If your EC2 session was lost, reconnect using Session Manager and change user to **ec2-user** by running the following command:

```
sudo su -l ec2-user
```

2. Edit the crontab and comment out the Snapper job

```
crontab -e
```

Press i to enter insert mode. Enter # to comment out the entry as follows:

```
##*/1 * * * * /home/ec2-user/scripts/pg_perf_stat_snapper.py -e <PostgreSQL Instance EndPoint. Cloudformation Output Key: clusterEndpoint> -P <Database Port. Cloudformation Output Key: Port> -d <Database Name where Application objects are stored. Cloudformation Output key: DatabaseName> -u <Master UserName. Cloudformation Output
```

```
Key:DBUsername> -s <AWS Secretes Manager ARN. Cloudformation Output Key: SnapperSecretARN> -m snap -r <AWS Region for e.g. us-west-2>
```

Enter **:wq!** to save and exit the editor.

Verify that crontab was successfully installed by running the following command.

```
crontab -l
```

3. Package the snapper output by running the following command:

```
/home/ec2-user/scripts/pg_perf_stat_snapper.py -e <PostgreSQL Instance EndPoint. Cloudformation Output Key: clusterEndpoint> -P <Database Port. Cloudformation Output Key: Port> -d <Database Name where Application objects are stored. Cloudformation Output key: DatabaseName> -u <Master UserName. Cloudformation Output Key:DBUsername> -s <AWS Secretes Manager ARN. Cloudformation Output Key: SnapperSecretARN> -m package -r <AWS Region for e.g. us-west-2>
```

for e.g.

```
/home/ec2-user/scripts/pg_perf_stat_snapper.py -e aupt-labs-cluster.cluster-cvmeikrm7zrz.us-west-2.rds.amazonaws.com -P 5432 -d mylab -u masteruser -s arn:aws:secretsmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1799fbf52e44813-snapperstack-1HDFCDHC5ONIV-8PS4cI -m package -r us-west-2
```

```
[ec2-user@ip-10-0-2-193 ~]$ /home/ec2-user/scripts/pg_perf_stat_snapper.py -e aupt-labs-cluster.cluster-cvmeikrm7zrz.us-west-2.rds.amazonaws.com -P 5432 -d mylab -u masteruser -s arn:aws:secretsmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1799fbf52e44813-snapperstack-1HDFCDHC5ONIV-8PS4cI -m package -r us-west-2
Packaging Completed Successfully ...
```

Import Snapper output

Snapper output can be loaded to any PostgreSQL database for analysis. For this lab, we will import the output to the same Aurora PostgreSQL database from which we collected the statistics. Import the output by running the following:

```
/home/ec2-user/scripts/pg_perf_stat_loader.py -e <PostgreSQL Instance EndPoint. Cloudformation Output Key: clusterEndpoint> -P <Database Port. Cloudformation Output Key: Port> -d <Database Name where Application objects are stored. Cloudformation Output key: DatabaseName> -u <Master UserName. Cloudformation Output Key:DBUsername> -s <AWS Secretes Manager ARN. Cloudformation Output Key: SnapperSecretARN> -o <snapper output directory containing the generated output files> -r <AWS Region for e.g. us-west-2>
```

for e.g.

```
/home/ec2-user/scripts/pg_perf_stat_loader.py -e aupt-labs-cluster.cluster-cvmeikrm7zrz.us-west-2.rds.amazonaws.com -P 5432 -d mylab -u masteruser -s arn:aws:secretsmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1799fbf52e44813-snapperstack-1HDFCDHC5ONIV-8PS4cI -o /home/ec2-user/scripts/output/aupt-labs-cluster.cluster-cvmeikrm7zrz.us-west-2.rds.amazonaws.com/mylab/ -r us-west-2
```

When prompted by the script, enter a database name e.g. mylab_snap which Snapper will create and then load all the statistics into it.

```
[ec2-user@ip-10-0-2-193 ~]$ /home/ec2-user/scripts/pg_perf_stat_loader.py -e aupp-laba-cluster.cluster-cvmeikm7zz.us-west-2.rds.amazonaws.com -P 5432 -d mylab -u masteruser -g arn:aws:se
cretmanager:us-west-2:953779585674:secret:pg_snapper/mod-b1799fbf52e44813-snapperstack-IHDPCDRCS0NIV-8PS4c1 -o /home/ec2-user/scripts/output/aupp-laba-cluster.cluster-cvmeikm7zz.us-west
-2.rds.amazonaws.com/mylab/ -r us-west-2
Enter Database name to be created for importing PostgreSQL performance statistics:mylab_snap
Loading of all pgawr related data completed successfully ...
```

Analyze performance metrics of the PostgreSQL database

Now that we have loaded all the Snapper collected statistics, lets analyze the data with sample SQL scripts provided with Snapper and see what insights we can derive.

The analysis can be done from any machine where **psql** is installed and has connectivity to the database where the snapper output was loaded. For this lab, we will use Cloud9, since it was setup with PostgreSQL client software.

1. Open a terminal window in AWS cloud9 and run the following commands to download the sample SQLs provided by Snapper.

```
mkdir -p /home/ec2-user/Snapper
cd /home/ec2-user/Snapper/
svn checkout "https://github.com/aws-samples/aurora-and-database-migration-labs/trunk/
cd SQLs
ls -l
```

2. Connect to the PostgreSQL database where we loaded the snapper output for e.g. mylab_snap.

```
psql
\c <database name where snapper output was loaded e.g. mylab_snap>
```

3. Run the Snapper menu showing list of available SQLs for analysis.

```
\i snappermenu.sql
```

```

mylab_snap=> \i snappermenu.sql
Pager usage is off.
Default footer is off.

==SNAPSHOT DETAILS==

list_snaps.sql                List snapshots available with time window

==SET SNAPSHOT WINDOW==

set_snaps.sql                 Set Begin and End Snapshot ID for Analysis

==INSTANCE AND DATABASE STATS==

db_and_schema_sizes.sql      Database and Schema Sizes
tables_and_indexes_tot_size.sql Top 20 Tables and Indexes by total Size
cache_hit_ratio.sql          Cache hit ratio in a time window
db_stats.sql                  Database Level statistics in a time window
checkpoint_stats_by_snap_id.sql Checkpoints stats in a time window
temp_file_by_snap_id.sql     Temp file stats by Snap ID
temp_table_cnt_by_snap_id.sql Temp tables count by Snap ID

==SESSION STATS==

session_cnt.sql               Total Sessions and Session count by state in a time window
session_activity_hist.sql     Sessions activity with wait events in a time window
blockers_and_waiters_hist.sql Blocking and Waiting Sessions in a time window
vacuum_history.sql            Vacuum activity in a time window

==SQL STATS==

top_20_sqls_by_calls.sql      Top 20 queries by Executions/Calls in a time window
top_20_sqls_by_elapsed_time.sql Top 20 queries by Elapsed time in a time window
top_10_sqls_by_cpu_by_snap_id.sql Top 10 SQL queries by CPU by Snap ID
sql_stat_history.sql          Execution trend of a query of interest in a time window

==TABLE STATS==

table_cols.sql                Details of Table columns
table_pk.sql                   Details of Table Primary Key
table_fks.sql                  Details of Foreign Keys referencing the Primary Key of the Parent Table
table_options.sql              Table Options for fill factor and Vacuuming
top_20_tables_by_seq_scans.sql Top 20 Tables by number of Sequential or Full scans
top_20_tables_by_dmls.sql      Top 20 Tables by DML activity
table_bloat.sql                Table Bloat Analysis
sqls_touching_table.sql        List SQLs touching a table

==INDEX STATS==

indexes_on_table.sql           Indexes on a table
fks_with_no_index.sql          Foreign Keys with no Index
needed_indexes.sql             Needed Indexes
top_20_indexes_by_scans.sql     Top 20 Indexes by number of Scans initiated in the index
top_20_indexes_by_avg_tuple_reads.sql TOP 20 Indexes by average Tuples Reads/Scan
unused_indexes.sql             Unused Indexes
duplicate_indexes.sql           Duplicate Indexes
index_bloat.sql                Index Bloat Analysis

```

- List all the available snap IDs and then set the begin and end snap ID for analysis as per your requirement by running the following:

```

\i list_snaps.sql
\i set_snaps.sql

```

For this lab, we can set the begin Snap ID to the minimum snap_id and end Snap ID to the maximum snap_id shown in the list_snaps.sql output.

```
mylab_snap=> \i list_snaps.sql
snap_id |          sample_start_time          |          sample_end_time          |
-----|-----|-----|
1 | 2021-07-31 19:42:01.853929+00 | 2021-07-31 19:42:06.979376+00 |
2 | 2021-07-31 19:43:01.694341+00 | 2021-07-31 19:43:07.056669+00 |
3 | 2021-07-31 19:44:01.512056+00 | 2021-07-31 19:44:06.884285+00 |
4 | 2021-07-31 19:45:02.424206+00 | 2021-07-31 19:45:07.77129+00 |
5 | 2021-07-31 19:46:02.4723+00 | 2021-07-31 19:46:07.791952+00 |
6 | 2021-07-31 19:47:02.471167+00 | 2021-07-31 19:47:07.829257+00 |
7 | 2021-07-31 19:48:02.621649+00 | 2021-07-31 19:48:07.745697+00 |
8 | 2021-07-31 19:49:02.405863+00 | 2021-07-31 19:49:07.542439+00 |
9 | 2021-07-31 19:50:02.225921+00 | 2021-07-31 19:50:07.340355+00 |
10 | 2021-07-31 19:51:01.867422+00 | 2021-07-31 19:51:07.23372+00 |
11 | 2021-07-31 19:52:02.114801+00 | 2021-07-31 19:52:07.462119+00 |
12 | 2021-07-31 19:53:02.125023+00 | 2021-07-31 19:53:07.454679+00 |
13 | 2021-07-31 19:54:02.35168+00 | 2021-07-31 19:54:07.649471+00 |
14 | 2021-07-31 19:55:02.410906+00 | 2021-07-31 19:55:07.750837+00 |
15 | 2021-07-31 19:56:02.291247+00 | 2021-07-31 19:56:07.41854+00 |
16 | 2021-07-31 19:57:02.303861+00 | 2021-07-31 19:57:07.420602+00 |
17 | 2021-07-31 19:58:02.044195+00 | 2021-07-31 19:58:07.647863+00 |
18 | 2021-07-31 19:59:02.295996+00 | 2021-07-31 19:59:07.975837+00 |
19 | 2021-07-31 20:00:01.683925+00 | 2021-07-31 20:00:07.359834+00 |
20 | 2021-07-31 20:01:02.406562+00 | 2021-07-31 20:01:07.812285+00 |
21 | 2021-07-31 20:02:02.401635+00 | 2021-07-31 20:02:07.526785+00 |
22 | 2021-07-31 20:03:02.127368+00 | 2021-07-31 20:03:07.224645+00 |
23 | 2021-07-31 20:04:01.777014+00 | 2021-07-31 20:04:06.90884+00 |
24 | 2021-07-31 20:05:02.454367+00 | 2021-07-31 20:05:07.565136+00 |
25 | 2021-07-31 20:06:02.286421+00 | 2021-07-31 20:06:07.621813+00 |
26 | 2021-07-31 20:07:02.261388+00 | 2021-07-31 20:07:07.893297+00 |
27 | 2021-07-31 20:08:02.446604+00 | 2021-07-31 20:08:07.835294+00 |
28 | 2021-07-31 20:09:02.27943+00 | 2021-07-31 20:09:07.38857+00 |
29 | 2021-07-31 20:10:01.836392+00 | 2021-07-31 20:10:06.965417+00 |
30 | 2021-07-31 20:11:01.701827+00 | 2021-07-31 20:11:06.831395+00 |
31 | 2021-07-31 20:12:02.365505+00 | 2021-07-31 20:12:07.482436+00 |
32 | 2021-07-31 20:13:01.902551+00 | 2021-07-31 20:13:07.03987+00 |
33 | 2021-07-31 20:14:02.020914+00 | 2021-07-31 20:14:07.143523+00 |
34 | 2021-07-31 20:15:01.552357+00 | 2021-07-31 20:15:06.688599+00 |
35 | 2021-07-31 20:16:02.369059+00 | 2021-07-31 20:16:07.500147+00 |
36 | 2021-07-31 20:17:02.158305+00 | 2021-07-31 20:17:07.268707+00 |
37 | 2021-07-31 20:18:01.880372+00 | 2021-07-31 20:18:07.016943+00 |
```

```
mylab_snap=> \i set_snaps.sql
enter Begin Snap ID: 1
enter End Snap ID: 37
mylab_snap=>
```

5. Lets see the top 20 tables and indexes by total size, by running the following SQL:

```
\i tables_and_indexes_by_tot_size.sql
```

```
mylab_snap=> \i tables_and_indexes_by_tot_size.sql
oid | schema_name | table_name | row_estimate | total_size | table_size | index_size | toast_size |
-----|-----|-----|-----|-----|-----|-----|-----|
20516 | public | pgbench_accounts | 10000000 | 1495 MB | 1281 MB | 214 MB | |
21056 | hr | employees | 4672776 | 746 MB | 645 MB | 100 MB | |
20513 | public | pgbench_tellers | 1000 | 120 kB | 48 kB | 40 kB | |
20519 | public | pgbench_branches | 100 | 56 kB | 8192 bytes | 16 kB | |
21053 | hr | departments | 27 | 24 kB | 8192 bytes | 16 kB | |
21059 | hr | job_history | 10 | 24 kB | 8192 bytes | 16 kB | |
21068 | hr | regions | 4 | 24 kB | 8192 bytes | 16 kB | |
21050 | hr | countries | 25 | 24 kB | 8192 bytes | 16 kB | |
21065 | hr | locations | 23 | 24 kB | 8192 bytes | 16 kB | |
21062 | hr | jobs | 19 | 24 kB | 8192 bytes | 16 kB | |
20503 | public | eventerrormsg | 0 | 8192 bytes | 0 bytes | 0 bytes | 8192 bytes |
20500 | public | statusflag | 0 | 8192 bytes | 8192 bytes | 0 bytes | |
20510 | public | pgbench_history | 0 | 0 bytes | 0 bytes | 0 bytes | |
20497 | public | cloneeventtest | 0 | 0 bytes | 0 bytes | 0 bytes | |
```

public.pgbench_accounts is the largest table followed by hr.employees.

6. Lets see the top SQL by elapsed time, by running the following SQL:

```
\i top_20_sqls_by_elapsed_time.sql
```

```
mylab_snap-> \i top_20_sqls_by_elapsed_time.sql
```

dbid	userid	queryid	avg_elapsed_time	calls	avg_shared_blks_hit	avg_rows	avg_shared_blks_dirtied	avg_shared_blks_read	avg_shared_blks_written	avg_temp_blks_written	avg_blk_read_time
16400	16399	6455889722256851027	15295.25	85	290643	1		0			
16400	16399	7781150283536014261	20.15	179840	186	1		0			
16384	10	-1214005784398965945	12.00	1	1	1					
16384	10	5643446292128394509	11.85	33	1	1					
16384	10	400430826648107333	10.86	36	1	1					
16384	10	-5412147542262639899	10.00	3		23					
16384	10	-4677044602036370826	9.15	72		1					
16384	10	-5514832750173122837	7.75	4		23					
16384	10	-5065622313111912070	7.23	13		21					
16384	10	1944619356053753085	3.50	2		23					
16384	10	8216883835551573255	3.25	4		1					
16384	10	998690969743055843	3.14	7		21					
16400	16399	-6883431918726892561	3.09	111906	58	1		0			
16384	10	8177024167513950062	3.00	2		22					
16384	10	-6413736282569810849	1.91	33	0	1					
16384	10	7148398129656748013	1.00	1		9					
16384	10	-3562459011652548624	0.94	33	0	1					
16384	10	5785233184570382019	0.78	36	1	1					
16384	10	30467291230705309	0.61	33		1					
16384	10	-6351436768293579039	0.58	36	1	1					

To see the SQL text and other execution statistics, run the following SQL and pass the queryid of interest shown in the above query output.

```
\i sql_stat_history.sql
```

```
mylab_snap-> \i sql_stat_history.sql
Enter queryid (shown in Top x queries reports): 6455889722256851027
query
select hr.update_employee_data_fname($1)
-----
sample_start_time | dbid | userid | queryid | snap_id | calls | delta_calls | rows/exec | elapsed_time_msec/exec | shared_blk_hit/exec | shared_blks_read/exec | shared_blks_written/exec |
temp_blks_written/exec | blk_read_time/exec
-----
```

2021-07-31 19:59:02.295996+00	16400	16399	6455889722256851027	18	39							
2021-07-31 20:00:01.683925+00	16400	16399	6455889722256851027	19	78	39	1	16507.31	312584	0		
2021-07-31 20:01:02.406562+00	16400	16399	6455889722256851027	20	119	41	1	14276.61	263866	0		
2021-07-31 20:02:02.401635+00	16400	16399	6455889722256851027	21	124	5	1	14194.00	339078	0		
2021-07-31 20:03:02.127368+00	16400	16399	6455889722256851027	22	124	0	0	0.00	0	0		
2021-07-31 20:04:01.777014+00	16400	16399	6455889722256851027	23	124	0	0	0.00	0	0		
2021-07-31 20:05:02.454367+00	16400	16399	6455889722256851027	24	124	0	0	0.00	0	0		
2021-07-31 20:06:02.286421+00	16400	16399	6455889722256851027	25	124	0	0	0.00	0	0		
2021-07-31 20:07:02.261388+00	16400	16399	6455889722256851027	26	124	0	0	0.00	0	0		
2021-07-31 20:08:02.446604+00	16400	16399	6455889722256851027	27	124	0	0	0.00	0	0		
2021-07-31 20:09:02.27943+00	16400	16399	6455889722256851027	28	124	0	0	0.00	0	0		
2021-07-31 20:10:01.836392+00	16400	16399	6455889722256851027	29	124	0	0	0.00	0	0		
2021-07-31 20:11:01.701827+00	16400	16399	6455889722256851027	30	124	0	0	0.00	0	0		
2021-07-31 20:12:02.365505+00	16400	16399	6455889722256851027	31	124	0	0	0.00	0	0		
2021-07-31 20:13:01.902551+00	16400	16399	6455889722256851027	32	124	0	0	0.00	0	0		
2021-07-31 20:14:02.020914+00	16400	16399	6455889722256851027	33	124	0	0	0.00	0	0		
2021-07-31 20:15:01.552357+00	16400	16399	6455889722256851027	34	124	0	0	0.00	0	0		
2021-07-31 20:16:02.369059+00	16400	16399	6455889722256851027	35	124	0	0	0.00	0	0		
2021-07-31 20:17:02.158305+00	16400	16399	6455889722256851027	36	124	0	0	0.00	0	0		
2021-07-31 20:18:01.880372+00	16400	16399	6455889722256851027	37	124	0	0	0.00	0	0		

`select hr.update_employee_data_fname($1)` is the top query by elapsed time since it was not using index as we saw in [Lab6:RDS Performance Insights](#).

7. To see the top tables by sequential scan (or Full table scan), run the following query:

```
\i top_20_tables_by_seq_scans.sql
```

```
mylab_snap-> \i top_20_tables_by_seq_scans.sql
relid | schemaname | relname | total_full_table_scan | avg_rows_per_fts | avg_rows_fetch_per_index | avg_rows_inserted | avg_rows_updated | avg_rows_deleted | avg_rows_hot_updated | max_live_rows | max_dead_r
ows | total_autovacuum_initiated
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----
21862 | hr | jobs | 5247968 | 14.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0
21853 | hr | departments | 4685248 | 20.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0
21856 | hr | employees | 441 | 4672352.00 | 1.97 | 0.00 | 0.68 | 0.00 | 0.00 | 0 | 0
146
```

The output shows some indexing opportunities for employees tables where we had 441 full table scans with average 4672352 rows returning in each scan.

8. To see the foreign keys which doesn't have an index on them (as per schema design best practice), run the following query:

```
\i fks_with_no_index.sql
```

```
mylab_snap-> \i fks_with_no_index.sql
schema_name | table_name | fk_name | issue | table_mb | writes | table_scans | parent_name | parent_mb | parent_writes | cols_list | indexdef
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----
hr | countries | countr_reg_fk | no index | 0 | 25 | 2 | regions | 0 | 4 | {region_id} |
hr | departments | dept_loc_fk | no index | 0 | 27 | 9334493 | locations | 0 | 23 | {location_id} |
hr | departments | dept_mgr_fk | no index | 0 | 27 | 9334493 | employees | 645 | 5304150 | {manager_id} |
hr | employees | emp_dept_fk | no index | 645 | 5304150 | 447 | departments | 0 | 27 | {department_id} |
hr | employees | emp_job_fk | no index | 645 | 5304150 | 447 | jobs | 0 | 19 | {job_id} |
hr | employees | emp_manager_fk | no index | 645 | 5304150 | 447 | employees | 645 | 5304150 | {manager_id} |
hr | job_history | jhist_dept_fk | no index | 0 | 10 | 5 | departments | 0 | 27 | {department_id} |
hr | job_history | jhist_job_fk | no index | 0 | 10 | 5 | jobs | 0 | 19 | {job_id} |
hr | locations | loc_c_id_fk | no index | 0 | 23 | 2 | countries | 0 | 25 | {country_id} |
```

The output above shows that there are some tables in hr schema with missing foreign key indexes. Depending on the number of queries accessing those tables and their execution frequency, indexing those columns will reduce full table (sequential) scans and pressure on the IO subsystem.

9. To see the table bloat across all tables, run the following query:

```
\i table_bloat.sql
```

```
mylab_snap-> \i table_bloat.sql
current_database | schemaname | tblname | real_size | extra_size | extra_ratio | fillfactor | bloat_size | bloat_ratio | is_na
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----
mylab | hr | countries | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | hr | departments | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | hr | employees | 645 MB | 59 MB | 9.17 | 100 | 59 MB | 9.17 | f
mylab | hr | job_history | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | hr | jobs | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | hr | locations | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | hr | regions | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | information_schema | sql_features | 56 kB | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | information_schema | sql_implementation_info | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | information_schema | sql_languages | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | information_schema | sql_packages | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | information_schema | sql_parts | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | information_schema | sql_sizing | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | information_schema | sql_sizing_profiles | 0 bytes | 0 bytes | 0 | 100 | 0 | 0 | t
mylab | pg_catalog | pg_aggregate | 16 kB | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | pg_catalog | pg_am | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | t
mylab | pg_catalog | pg_amop | 56 kB | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | pg_catalog | pg_amproc | 32 kB | 0 bytes | 0 | 100 | 0 | 0 | f
mylab | pg_catalog | pg_attrdef | 0 bytes | 0 bytes | 0 | 100 | 0 | 0 | t
mylab | pg_catalog | pg_attribute | 472 kB | 64 kB | 13.56 | 100 | 64 kB | 13.56 | t
mylab | pg_catalog | pg_auth_members | 8192 bytes | 0 bytes | 0 | 100 | 0 | 0 | f
```

The above output shows that hr.employees table has 9% bloat which can be freed up using a Full Vacuum or pg_repack.

This concludes the lab!

In this lab, we saw how we can derive insights using PostgreSQL dictionary stats with the help of Snapper tool. Feel free to explore other sample SQLs provided by Snapper as time permits.