Building with Amazon Aurora Databases

**SPL-TF-200-DBWAAD-1 - Version 1.0.9**

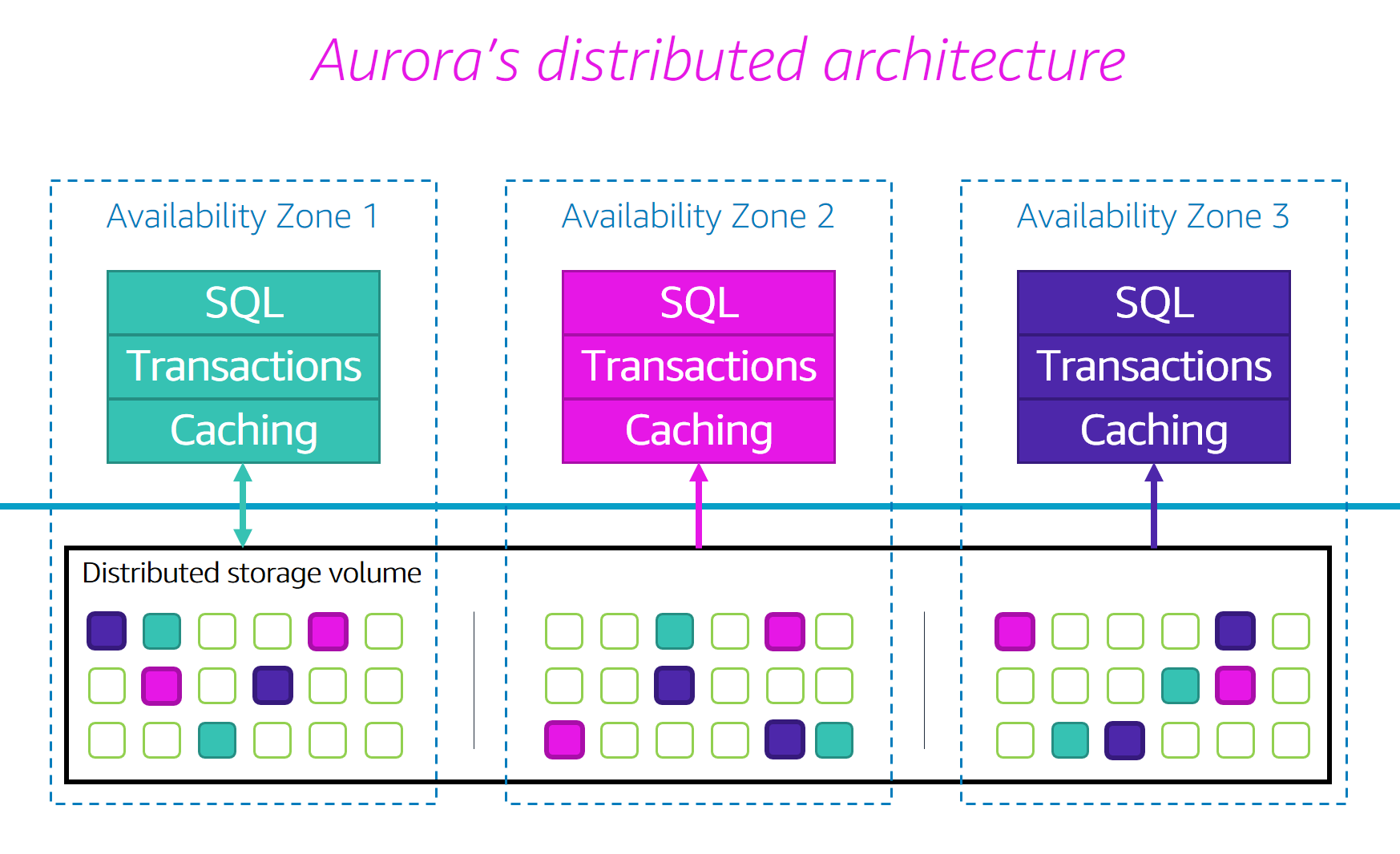
© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Overview**

Amazon Aurora is a MySQL- and PostgreSQL-compatible relational database engine built for the cloud. Aurora is fully managed by Amazon Relational Database Service (RDS), which automates time-consuming administration tasks like hardware provisioning, database setup, patching, and backups. Aurora is built on a modern, purpose-built distributed storage system. All data is distributed in three different AWS Availability Zones, across hundreds of storage nodes, with two copies per zone. The Aurora MySQL- and PostgreSQL-compatible database engines are customized to take advantage of the fast distributed storage.



By default, a query co-locates all scanned data to a single head node within the Aurora cluster and performs all the query processing there. To further improve performance, you can enable parallel query, an optimization in which Aurora delegates or pushes down some of the I/O and computation of data-intensive statements to the storage nodes. Aurora parallel query can be a good fit for analytical workloads requiring fast statement performance on large tables with fresh data. Workloads of this type are often operational in nature.

In this lab, you will investigate how and when parallel query is applied to a statement. You will also learn how to ensure that parallel query is applied where it provides the most benefit.

**Objectives**

After completing this lab, you will be able to:

* Understand how parallel query can benefit your workload
* Understand the benefit of parallel query for large datasets
* Determine some of the situations that trigger parallel query

**Pre-requisites**

This lab requires:

* Access to a notebook computer with Wi-Fi and Microsoft Windows, macOS X, or Linux (Ubuntu, SuSE, or Red Hat).
* **Note:** You can use an iPad or tablet device to access these directions in the lab console.
* An Internet browser such as Chrome, Firefox, or IE9+.
* **Note:** Previous versions of Internet Explorer are not supported.
* An SSH client such as PuTTY.

**Technical knowledge prerequisites**

To successfully complete this lab, you should be familiar with:

* Familiarity with MySQL database syntax and operation

**Duration**

This lab requires **60 minutes** to complete.

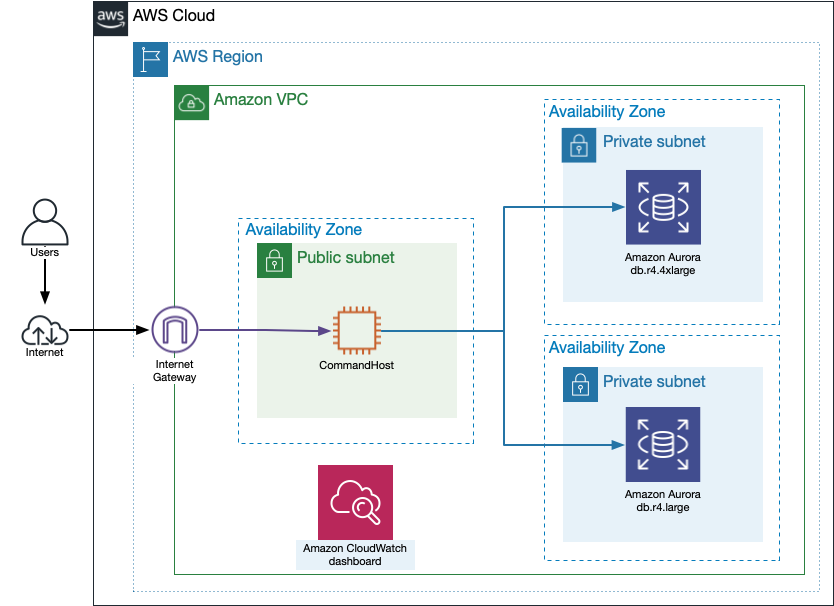
**AWS services not used in this lab**

AWS services that are not used in this lab are disabled in the lab environment. In addition, the capabilities of the services used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

**Lab environment**

This lab environment consists of two Aurora database instances in a single cluster. The instances are different sizes and are preloaded with the same example dataset. The environment also includes an Amazon Elastic Compute Cloud (Amazon EC2) instance for accessing the database instances and an Amazon CloudWatch dashboard with preconfigured widgets.

The following image is a network diagram of the environment:



Five years of United States flight data, approximately 30 million records, compiled by the Office of Airline Information, Bureau of Transportation Statistics has been preloaded for you. In this lab, you will explore several statements that benefit from parallel query. To better understand how parallel query impacts server load, you will examine buffer utilization, result times, and metric graphs shown on a CloudWatch dashboard.

Data set(s) used in this lab were compiled by Office of Airline Information, Bureau of Transportation Statistics, US carrier scheduled and actual arrival and departure times (1987-2019), available at https://www.transtats.bts.gov/DatabaseInfo.asp?DB\_ID=120&DB\_URL=Mode\_ID=1&Mode\_Desc=Aviation&Subject\_ID2=0.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Design considerations for using parallel query**

Parallel query does not depend on the CPU capacity of the database server. When the parallel query feature is enabled, the Aurora query engine automatically determines when statements can benefit. Examples of work that can be parallelized in the storage layer include:

* Retrieving rows from storage
* Extracting column values
* Determining which rows match the conditions in WHERE and JOIN clauses

Best practice is to conduct tests to determine whether parallel query will benefit your workload and to identify the schema, statements, or settings that best take advantage of this feature.

Several design considerations are important when configuring your Aurora cluster to use parallel query, including:

* Region
* Instance class
* Database engine version
* Compression format
* Storage engine

Aurora MySQL engine versions **2.09** (Aurora MySQL 5.7 compatible) or higher, and **3.1** (Aurora MySQL 8.0 compatible) or higher versions, you can enable parallel by setting the parameters

aurora\_parallel\_query=ON

 (For Aurora MySQL version 2.09 and higher minor versions, or Aurora MySQL version 3) and

aurora\_disable\_hash\_join=0

 (For Aurora MySQL version 2.09 and higher minor versions) in a custom **DB cluster parameter group**. In this lab you are provided a parallel query-enabled Aurora MySQL cluster of engine version **2.10.2** with a custom DB parameter group.

I/O costs for your statement are metered at the storage layer and will be the same or larger with parallel query enabled.

**Learn more** For a full list of considerations, refer to [Working with Parallel Query for Amazon Aurora MySQL](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-mysql-parallel-query.html) and [Amazon Aurora FAQs](https://aws.amazon.com/rds/aurora/faqs/#Parallel_Query).

**Learn more** You can also read more about parallel query from Jeff Barr’s blog post, [New - Parallel Query for Amazon Aurora](https://aws.amazon.com/blogs/aws/new-parallel-query-for-amazon-aurora/).

**Task 1: Explore the lab environment**

In this task, you will sign in to the AWS Management Console to review the two Aurora clusters. You will investigate the parallel query setting and the basic monitoring available for Aurora clusters.

1. In the **AWS Management Console** *Search* field, type

RDS

.

1. Select **RDS** from search results.
2. In the left navigation pane, click **Databases**.

This lab contains one cluster with two instances associated with it: one serves as the writer and the other serves as the reader.

1. In the left navigation pane, click **Parameter groups**.
2. In the table, select the name of the parameter group that contains **dbclusterparametergroup** belonging to **Family** **aurora-mysql5.7** and is of **Type** **DB cluster parameter group**.

This page lists the parameters and provides the allowed values and a description of what each parameter controls.

1. In the **Filter parameters** field at the top of the table, enter

aurora\_parallel\_query

The parameter should be set to

ON

. This is a dynamic parameter which can be set globally at the cluster or at the session level

1. In the **Filter parameters** field , enter

aurora\_disable\_hash\_join

The parameter should be set to

0

, indicating hash join optimization is also enabled for the cluster to improve the performance of large table joins. This is a dynamic parameter which can be set globally at the cluster level

 Congratulations! You have successfully explored the lab environment.

**Task 2: Explore the data and parallel query settings**

In this task, you will first connect to the publicly accessible Amazon EC2 instance named **CommandHost**, from which you will connect to the private Aurora database instances. Once you have connected to the database, you will run several statements to see how the database is configured.

1. Copy the **CommandHostSessionUrl** value from the left side of the lab page, and paste it in a new browser tab. The terminal for the CommandHost instance opens.

**Note:** If you encounter a problem connecting to Session Manager, [click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#ssh-instructions) for help with connecting to the EC2 instance using an SSH client.

1. **Command:** To access the **home** directory, run the following command:

cd ~

UNDERSTANDING THE DATA

1. **Copy/Paste:** Copy and paste the following command to a text editor:

mysql -u (DBUserName) -p -h (SmallDBEndpoint)

1. Replace **(DBUserName)** and **(SmallDBEndpoint)** with the corresponding values from the left side of the lab page.
2. **Copy/Paste:** To connect to the Aurora instance, copy and paste the updated command from the text editor into the terminal window. Press ENTER to run the command.
3. When prompted for the password, copy and paste the **DBUserPasswd** value from the left side of the lab page, and press ENTER.

**Note:** When you enter the password, the terminal cursor does not move.

1. **Command:** To use the **ontimeflights** database for the following steps, run the following statement:

USE ontimeflights;

**Expected Similar output:**

MySQL [ontimeflights]>

1. **Command:** To show the tables in the database, run the following statement:

SHOW TABLES;

The two tables shown are preloaded for you. The **carriers** table contains a list of all airline carrier identification codes and names, while the **flightdata** table contains information about specific flights such as date, time, delay, and reporting airline. The tables are related on the **carrierCode** column from the **carriers** table and the **Reporting\_Airline** column from the **flightdata** table.

**Understand the carriers table**

1. **Command:** To show details about the **carriers** table, run the following statements:

DESCRIBE carriers;

SELECT FORMAT(COUNT(carrierCode), 0) AS 'Number of Carriers' FROM carriers;

The first output shows the structure of the table. The table contains two columns, **carrierCode** and **carrierName**. This table can be joined to the **flightdata** table so that the output shows the name of the carrier instead of the carrier code. The second output shows that this table contains 1,928 records.

1. **Command:** To view a sample of the data, run the following statement:

SELECT \* FROM carriers LIMIT 10;

**Understand the flightdata table**

1. **Command:** To show details about the **flightdata** table, run the following statement:

DESCRIBE flightdata;

The output shows 46 columns and their data types. Take a moment to look at the columns available in this table. You will use them later in the lab to run prebuilt statements to test parallel query.

1. **Command:** To show the row count for this table, run the following statement:

SELECT FORMAT(COUNT(Year), 0) AS 'Number of Flights' FROM flightdata;

**Note:** This statement takes about 2 minutes to return a result.

The output from this statement shows that the **flightdata** table has around 30 million rows.

Parallel query is a feature that is only effective on very large tables when complex statements are run. Parallel query optimization provides the most benefit for long-running statements that take minutes or hours to complete. In the following tasks, you will run several statements to see how parallel query can improve query performance.

CONFIGURE PARALLEL QUERY USING THE AWS CLI

There are several session variables that control how parallel query works. In this subtask, you will investigate these settings.

1. **Command:** To determine if parallel query is enabled on the instance for your session, run the following statement:

SELECT @@aurora\_parallel\_query;

SELECT @@aurora\_disable\_hash\_join;

**Expected output:**

+-------------------------+

| @@aurora\_parallel\_query |

+-------------------------+

| 1 |

+-------------------------+

+----------------------------+

| @@aurora\_disable\_hash\_join |

+----------------------------+

| 0 |

+----------------------------+

This confirms that the instance and your session is enabled for parallel query along with hash join optimization.

TOGGLE THE PARALLEL QUERY SETTINGS

You will be running statements to show the improvements that parallel query offers. To accomplish this, you must know how to enable and disable parallel query.

1. **Command:** To *disable* parallel query on the instance for your session, run the following statement:

SET SESSION aurora\_parallel\_query=0;

You have now disabled parallel query on the instance for the duration of your session.

1. **Command:** To confirm this, run the following statement:

SELECT @@aurora\_parallel\_query;

1. **Command:** To *enable* parallel query on the instance for your session, run the following statement:

SET SESSION aurora\_parallel\_query = 1;

By default, Aurora determines which statements will benefit from parallel query as you run them. Adding the **SET SESSION** statement before a statement that returns results enables parallel query for that specific session without changing the global setting.

You are also able to force parallel query to run by setting the **aurora\_pq\_force** parameter in the same manner.

1. **Command:** To make sure that all the settings are configured correctly, run the following statements:

SELECT @@aurora\_parallel\_query;

SELECT @@aurora\_pq\_force;

You should see the following returned values:

* aurora\_parallel\_query: **1**
* aurora\_pq\_force: **0**

**Note:** If any of the values are not correct, use **SET SESSION** statements to update them.

**Challenge** Use **SET SESSION** statements to change the current settings for the **aurora\_pq\_force** parameter. [Click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#solution-1) to review the sample solution.

INVESTIGATE THE CLOUDWATCH DASHBOARD

Before you start running statements, you need to open the CloudWatch dashboard you will use to monitor instance metrics. It is recommended that you place the browser tabs for your terminal session and CloudWatch dashboard side by side if possible so that you can run statements and monitor the graphs for changes as the statements run.

1. Return to the browser tab with the **AWS Management Console**, in the *Search* field enter

CloudWatch

.

1. select **CloudWatch**.
2. In the left navigation pane, click **Dashboards**.
3. Click the name of the **ontimeflights-Dashboard** dashboard.

The dashboard contains graphs of several metrics that will allow you to monitor the database instance and see how parallel query is impacting your instance.

1. Observe the **RDS: Instance PQ Stats** graph.

Note the graph and color of the three metrics that are charted in this graph. They should all be at a baseline of 0 because you have not yet run any statements that would trigger parallel query. The **Aurora\_pq\_request\_executed** metric will be used to identify how often parallel query is being used when you are running statements. You will return to this screen later in the lab to observe the metrics.

 Congratulations! You have successfully explored the environment.

**Task 3: Understand the impact of parallel query**

Now that you are familiar with the Aurora cluster, instances, and CloudWatch dashboard, you are ready to start running statements that will trigger the parallel query feature. As you learned in the last task, manually setting the parallel query parameters for specific statements can help you test the feature. You will manually set the **aurora\_parallel\_query** and **aurora\_pq\_force** session variables before running several prebuilt statements to test the feature.

**Note:** The **aurora\_pq\_force** parameter is designed for testing purposes.

INVESTIGATE BASIC STATEMENT PERFORMANCE

The **EXPLAIN** statement shows the query plan, including an indication of whether parallel query is under consideration. This will allow you to identify statement options that impact potential use of parallel query.

1. Return to the tab with the **CommandHost** terminal window.

**Note:** If you closed the terminal window tab or lost the connection, [click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#reconnect) for instructions on how to reconnect.

1. **Command:** To determine if the query plan will recommend parallel query, run the following statement:

EXPLAIN

SELECT

Origin, Dest, Reporting\_Airline,

AVG(DepDelayMinutes) 'Avg Departure Delay', COUNT(DepDelay) 'Delayed Flights'

FROM flightdata;

In the output, the **Extra** column should show **NULL**. You should **NOT** see **Using parallel query**. Parallel query is only considered when the query is complex enough or impacts enough rows.

Next, you will add several WHERE clauses to the statement and see how that impacts the query plan.

1. **Command:** To determine if the query plan will recommend parallel query, run the following statement:

EXPLAIN

SELECT

Origin, Dest, Reporting\_Airline,

AVG(DepDelayMinutes) 'Avg Departure Delay', COUNT(DepDelay) 'Delayed Flights'

FROM flightdata

WHERE DepDelay > 0

AND Origin NOT IN ('TWF', 'SNA', 'ORD');

In the output, the **Extra** column should show **Using parallel query**.

EXPLAIN helps you identify which statements are candidates for using parallel query when run. The WHERE clause is one of several SQL constructs that can trigger parallel query operations. Be aware that Aurora will evaluate the statement when run to determine if it will run using parallel query or not, so this does not guarantee that it will be utilized.

Now you will see how making the statement more complex impacts the query plan.

1. **Command:** To determine if the query plan will recommend parallel query, run the following statement:

EXPLAIN

SELECT

Origin, Dest, Reporting\_Airline,

AVG(DepDelayMinutes) 'Avg Departure Delay', COUNT(DepDelay) 'Delayed Flights'

FROM flightdata

WHERE DepDelay > 0

AND Origin NOT IN ('TWF', 'SNA', 'ORD')

AND ArrDelayMinutes > DepDelayMinutes;

As the statement is made more complex, the evaluation engine updates the query plan. Notice the changes in the **Extra** column of the output between the three statements. This ensures that all variables are taken into account when a final query plan is built.

For more information about why particular SQL statements use or don’t use parallel query, see [How Parallel Query Works with SQL Constructs](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-mysql-parallel-query.html).

RUN STATEMENTS WITH PARALLEL QUERY DISABLED

To understand the impact that parallel query can have on a production environment, you must know how the buffer pool is used during SQL statement processing. The InnoDB buffer pool is the memory space that holds many in-memory data structures of InnoDB including buffers, caches, indexes, and row data. You will use statistics from this pool to determine the effectiveness of parallel query.

1. **Command:** To display the current **Innodb\_buffer\_pool** statistics, run the following statement:

SHOW GLOBAL STATUS LIKE 'Innodb\_buffer\_pool%';

**Expected Similar output:**



1. From the output, copy and paste the **Innodb\_buffer\_pool\_read\_requests** value to a text editor. It is the value to the left of the green arrow in the previous image.

**Note:** To copy from the terminal window in Microsoft Windows, you MUST use the right-click menu. If you press CTRL+C, you will exit the MySQL database and be at the *sh-4.2$* prompt. [Click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#reconnect-mysql) for directions on how to reconnect.

This is your pre-statement value. You will compare this to the post-statement value. The difference between these two values shows you the impact of running the statement with parallel query disabled.

The following statements will disable both parallel query parameters for this session and issue a complex statement. The statement is one you tested earlier. The query plan showed that it should be run with parallel query; however, because you are disabling parallel query, this will process all the data on the database instance and not utilize the storage layer.

1. **Command:** Run the following statements:

SET SESSION aurora\_parallel\_query=0;

SET SESSION aurora\_pq\_force=0;

SELECT AVG(DepDelayMinutes + ArrDelayMinutes) AS "Average Delay"

FROM flightdata

WHERE Distance > 200

AND OriginCityName NOT IN ('Chicago IL', 'Miami FL');

**Note:** This statement takes a few minutes to return a result.

1. **Command:** To display the current **Innodb\_buffer\_pool** statistics, run the following statement:

SHOW GLOBAL STATUS LIKE 'Innodb\_buffer\_pool%';

1. **Copy/Paste:** From the output, copy and paste the **Innodb\_buffer\_pool\_read\_requests** value to a text editor.

This is your post-statement value.

1. Calculate and record the difference between the pre- and post-statement values. Record this value in the text editor.

Sample:

| **Run** | **Innodb\_buffer\_pool\_read\_requests** |
| --- | --- |
| Pre-statement | 1437605865 |
| Post-statement | 1468813279 |
| Difference | 31207414 |

The difference represents the impact running this statement had on the **Innodb\_buffer\_pool\_read\_requests** value when parallel query was disabled.

INVESTIGATE HOW DATABASE CACHE IMPACTS RESULTS

By default, Aurora stores all statement results temporarily in cache. This allows consecutive statements to run faster and is generally a good thing. However, using a cache would skew the results of the lab test.

1. **Command:** To see the impact of having a database cache, run the following statements:

SET SESSION aurora\_parallel\_query=0;

SET SESSION aurora\_pq\_force=0;

SELECT AVG(DepDelayMinutes + ArrDelayMinutes) AS "Average Delay"

FROM flightdata

WHERE Distance > 200

AND OriginCityName NOT IN ('Chicago IL', 'Miami FL');

Notice that the results were returned very quickly. This is because the results were stored in cache.

UNDERSTAND HOW CACHE IMPACTS RESULTS

Amazon Aurora has been designed with a built-in caching layer for improved performance. For testing purposes, you need to disable the cache to ensure the database is processing the request and not just returning a cached result. You can do this by adding the **SQL\_NO\_CACHE** optimizer hint to your statements as needed.

**Note:** The **SQL\_NO\_CACHE** hint is rarely used in a production environment.

1. **Command:** To run the statement without parallel query or cache, run the following statements:

SET SESSION aurora\_parallel\_query=0;

SET SESSION aurora\_pq\_force=0;

SELECT SQL\_NO\_CACHE AVG(DepDelayMinutes + ArrDelayMinutes) AS "Average Delay"

FROM flightdata

WHERE Distance > 200

AND OriginCityName NOT IN ('Chicago IL', 'Miami FL');

The statement takes a few minutes to return a result. This is a similar result to what you saw when you first ran the statement.

RUN STATEMENTS WITH PARALLEL QUERY ENABLED

Now that you have investigated the impact of running statements without parallel query and without caching, it is time to see the impact that parallel query can have on these statements.

Start by identifying the current read requests value in the InnoDB buffer pool statistics.

1. **Command:** To display the current **Innodb\_buffer\_pool** statistics, run the following statement:

SHOW GLOBAL STATUS LIKE 'Innodb\_buffer\_pool%';

**Expected Similar output:**



1. **Copy/Paste:** Copy and paste the **Innodb\_buffer\_pool\_read\_requests** value to a text editor. It is the value to the left of the green arrow in the previous image.

Just as you did before, you will compare a pre-statement value to the post-statement value.

Next, you will use session settings to enable and force parallel query. Because the statement has already been cached, you will use the **SQL\_NO\_CACHE** hint.

1. **Command:** To determine the average flight delay, run the following statements:

SET SESSION aurora\_parallel\_query=1;

SET SESSION aurora\_pq\_force=1;

SELECT SQL\_NO\_CACHE AVG(DepDelayMinutes + ArrDelayMinutes) AS "Average Delay"

FROM flightdata

WHERE Distance > 200

AND

OriginCityName NOT IN ('Chicago IL', 'Miami FL');

The results should return in about 25 seconds. This is much faster than without parallel query but not as fast a cached result. Although cached results are the fastest, you cannot always rely on them because any data added to the database results in the cache being invalidated.

Now, let’s see how much parallel query impacts **Innodb\_buffer\_pool\_read\_requests**.

1. **Command:** To display the current **Innodb\_buffer\_pool** statistics, run the following statement:

SHOW GLOBAL STATUS LIKE 'Innodb\_buffer\_pool%';

1. **Copy/Paste:** Copy and paste the **Innodb\_buffer\_pool\_read\_requests** value to a text editor.

This is your post-statement value.

1. Calculate and record the difference between the pre- and post-statement values. Record this value in the text editor.

**Expected Sample output:**

| **Run** | **Innodb\_buffer\_pool\_read\_requests** |
| --- | --- |
| Pre-statement | 1499756746 |
| Post-statement | 1499773288 |
| Difference | 16542 |

The difference represents the impact running this statement had on the **Innodb\_buffer\_pool\_read\_requests** value when parallel query was enabled. Notice that the number of read requests is significantly lower with parallel query enabled. Parallel query operations decrease buffer pool read requests.

 Congratulations! You have successfully configured and deployed an RDS instance.

**Task 4: Understand how instance size impacts parallel query**

You have been exploring parallel query on a single instance to see the performance gains it offers. Next, you will look at how different instance sizes impact the number of parallel query sessions available to the concurrently running SQL statements in those database instances.

INVESTIGATE THE AURORA INSTANCES

1. Return to the browser tab with the **Amazon CloudWatch Management console** open.
2. In the **AWS Management Console** *Search* field, type

RDS

.

1. Select **RDS** from search results.
2. In the left navigation pane, click **Databases**.
3. Look at the **Size** column for each instance.

The writer is a **db.r5.4xlarge**, while the reader is a **db.r5.large**. Both instances are configured to use parallel query. Later in the lab, you will use these two instances to compare the results of statements that trigger the parallel query feature.

1. Click the name of the instance with the **Writer** role.
2. Click the **Configuration** tab.

Note the instance class, number of vCPUs, and amount of RAM the writer has.

1. Click the name of the instance with the **Reader** role.

Notice that the reader is configured with a smaller instance class, which has fewer vCPUs and less RAM compared to the writer.

Each Aurora DB instance can run only a certain number of parallel query sessions at one time. If a query has multiple parts that use parallel query, such as **subqueries**, **joins**, or **UNION** operators, those phases run in sequence. The statement only counts as a single parallel query session at any one time. You can monitor the number of active sessions using the parallel query status variables. You can check the limit on concurrent sessions for a given DB instance by querying the status variable **Aurora\_pq\_max\_concurrent\_requests**.

TEST THE SMALLER INSTANCE

1. **Command:** To view the maximum concurrent parallel query sessions allowed with this instance type, run the following statement in the Command Host session window connected to mysql:

SHOW GLOBAL STATUS LIKE 'aurora\_pq\_max\_concurrent\_requests';

**Expected output:**

+-----------------------------------+-------+

| Variable\_name | Value |

+-----------------------------------+-------+

| Aurora\_pq\_max\_concurrent\_requests | 1 |

+-----------------------------------+-------+

This indicates the maximum number of parallel query sessions that can run conncurrently on this instance is **1**. You will observe the impact of this towards the end of the task.

1. **Command:** To disconnect from the database server, run the following command:

exit

1. **Command:** Run the following command on the Command Host session:

sudo su - ec2-user && cd ~

1. **Command:** Copy and paste the following command to a text editor:

./run-pq.sh (DBUserName) (SmallDBEndpoint) (DBUserPasswd)

1. Replace **(DBUserName)**, **(SmallDBEndpoint)**, and **(DBUserPasswd)** with the corresponding values from the left side of the lab page.

**Note:** This is the SQL query used by the

run-pq.sh

 script. The SQL query fetches flights where the distance is less than 1,000 miles with a carrier code of AA, UA, DL, or b6.:

SET SESSION aurora\_parallel\_query=1;

SET SESSION aurora\_pq\_force=1;

SELECT SQL\_NO\_CACHE

carriers.carrierName AS 'Carrier',

flightdata.Origin AS 'Origin Airport',

flightdata.Dest AS 'Destination Airport',

AVG(flightdata.ArrDelay) AS 'Average Arrival Delay',

COUNT(flightdata.ArrDelay) AS 'Number of Delays',

SUM(flightdata.Cancelled) AS 'Cancelled Flights'

FROM flightdata

JOIN carriers ON (carriers.carrierCode = flightdata.Reporting\_Airline)

WHERE flightdata.Distance < 1000

AND flightdata.Reporting\_Airline IN ("AA", "UA", "DL", "B6")

GROUP BY Origin, Dest

ORDER BY carriers.carrierName;

1. **Copy/Paste:** Copy and paste the updated command from the text editor into the terminal window. Press ENTER to run the code. The script concurrently runs the SQL query using parallel query, in **two** different database sessions.

**Expected output:**

query1 start

query2 start

query1 elapsed time: 46.86 seconds

query1 done

query2 elapsed time: 88.90 seconds

query2 done

Successful

**Check**: The first query should use significantly less time than the second. This is due to the fact that this smaller instance could only perform 1 parallel query at a time. So query2 was not able to leverage the parallel query ability that was already being consumed by query1.

TEST THE LARGER INSTANCE

In this subtask, you will connect to the larger instance and run the statements again with and without parallel query enabled.

1. **Copy/Paste:** Copy and paste the following command to a text editor:

mysql -u (DBUserName) -p -h (LargeDBEndpoint)

1. Replace **(DBUserName)** and **(LargeDBEndpoint)** with the corresponding values from the left side of the lab page.
2. Copy and paste the updated command from the text editor into the terminal window. Press ENTER to run the code.
3. When prompted for the password, copy and paste the **DBUserPasswd** value from the left side of the lab page, and press ENTER.
4. **Command:** To use the **ontimeflights** database, run the following statement:

USE ontimeflights;

1. **Command:** To view the maximum concurrent parallel query sessions allowed with this instance type, run the following statement:

SHOW GLOBAL STATUS LIKE 'aurora\_pq\_max\_concurrent\_requests';

**Expected output:**

+-----------------------------------+-------+

| Variable\_name | Value |

+-----------------------------------+-------+

| Aurora\_pq\_max\_concurrent\_requests | 8 |

+-----------------------------------+-------+

This indicates the maximum number of parallel query sessions that can run conncurrently on this instance is **8**. We will observe the impact of this towards the end of the task.

1. **Command:** To disconnect from the database server, run the following command:

exit

1. **Command:** Copy and paste the following command to a text editor:

./run-pq.sh (DBUserName) (LargeDBEndpoint) (DBUserPasswd)

1. Replace **(DBUserName)**, **(LargeDBEndpoint)**, and **(DBUserPasswd)** with the corresponding values from the left side of the lab page.
2. Copy and paste the updated command from the text editor into the terminal window. Press ENTER to run the code. The script concurrently runs the SQL query using parallel query, in **two** different database sessions.

**Expected output:**

query1 start

query2 start

query1 elapsed time: 16.79 seconds

query1 done

query2 elapsed time: 16.65 seconds

query2 done

Successful

**Check**: Both queries on the larger instance should have roughly the same time elapsed, since the larger number of concurrent parallel queries supported by the larger instance allowed both queries to use the feature.

INVESTIGATE THE **CLOUDWATCH DASHBOARD** FOR COMPARATIVE ANALYSIS OF PARALLEL QUERY PERFORMANCE BETWEEN THE TWO INSTANCES

**Overview:** Aurora MySQL Global status counters are published as metrics to CloudWatch. These will be used to interpret the CloudWatch dashboard graphs **RDS: Instance PQ Stats** and **RDS: Large Instance PQ Stats**

**Additional information:** [Monitoring parallel query](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-mysql-parallel-query.html#aurora-mysql-parallel-query-monitoring)

**Consider:** The key metric for this sub-task is **Aurora\_pq\_request\_throttled** - The number of times parallel query wasn’t chosen due to the maximum number of concurrent parallel queries already running on a particular Aurora DB instance.

1. Return to the browser tab with the **AWS Management Console**, in the *Search* field enter

CloudWatch

.

1. select **CloudWatch**.
2. In the left navigation pane, click **Dashboards**.
3. Click the name of the **ontimeflights-Dashboard** dashboard.
4. Hover over the latest peaks on the widget **RDS: Instance PQ Stats**.

**Note:** You might need to refresh the graph for a few minutes to see the results, as they are slightly delayed from when the script ran.

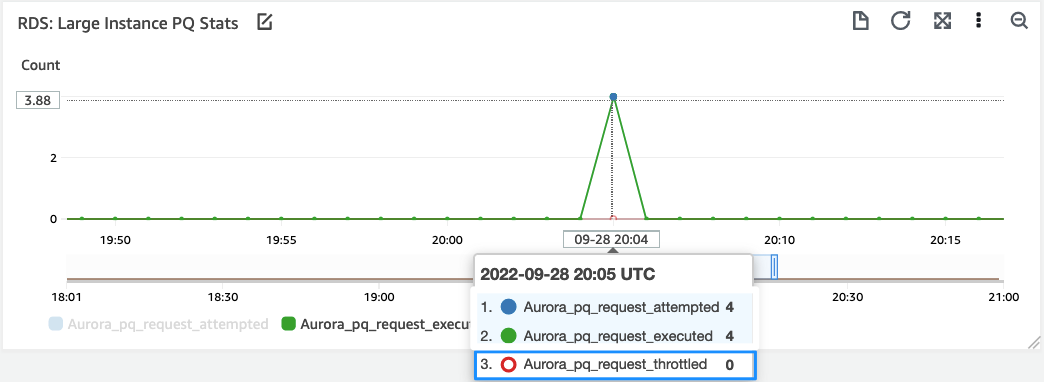


Results:

**Aurora\_pq\_request\_throttled**: 2 (Parallel query sessions that could not be requested during that 1 minute interval)

Since the **Aurora\_pq\_max\_concurrent\_requests** is **1** for the smaller instance, only 1 out of the two queries was able to use parallel query. Each parallel query session isn’t necessarily a one-to-one mapping with the queries ran on the database. In this particular case, the query had 2 steps utilizing parallel query sessions. The other concurrently running query couldn’t request parallel query sessions for the same steps and the counter **Aurora\_pq\_request\_throttled** was incremented by **2**.

1. Hover over the latest peaks on the widget **RDS: Large Instance PQ Stats** .



Results:

**Aurora\_pq\_request\_throttled**: 0 (Parallel query sessions that could not be requested during that 1 minute interval)

Since the maximum concurrent parallel query requests supported by this instance class is **8**, and **2** concurrently running queries needed **2** parallel query sessions each, there was no throttling observed (**Aurora\_pq\_request\_throttled**: 0).

 Congratulations! You have successfully configured and deployed an RDS instance.

When looking at the results, you can see the maximum number of parallel query sessions that can run concurrently is a fixed number that depends on the AWS DB instance class. This is another consideration when selecting the instance types for your Aurora cluster according to workload.

As you have seen, parallel query offers a potential advantage for complex, large statements that interact with millions of records. Large analytical statements that need to be run periodically on fresh data within your Aurora relational database are a potential fit for parallel query. There are other database options available from AWS, such as Amazon Redshift, that are designed by default to run intensive analytical statements. Parallel query offers the advantage of increased performance on its native database engine.

**Conclusion**

 Congratulations! You now have successfully:

* Understood how parallel query can benefit your workload
* Understood the benefit of parallel query for large datasets
* Determined some of the situations that trigger parallel query

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional resources**

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).

**Appendix**

CHANGE THE AURORA\_PQ\_FORCE PARAMETER SETTINGS

**Command:** To enable the force parameter:

SET SESSION aurora\_pq\_force = 1;

**Command:** To disable the force parameter:

SET SESSION aurora\_pq\_force = 0;

[Click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#Challenge-1) to return to the task.

RECONNECT TO THE COMMANDHOST AND DATABASE INSTANCE

To reconnect to the CommandHost and the database instance in Session Manager, follow these steps:

* STEP 1: Copy the **CommandHostSessionUrl** value from the left side of the lab page and paste the URL in a new browser tab.
* **Command:** STEP 2: To connect to the **home** directory, run the following command:

cd ~

* STEP 3: To connect to the database instance:
  + **Copy/Paste:** Copy the following command to a text editor:

mysql -u (DBUserName) -p -h (SmallDBEndpoint)

* + Replace **(DBUserName)** and **(SmallDBEndpoint)** with the corresponding values from the left side of the lab page.
  + Copy and paste the updated statement from the text editor into the terminal window. Press ENTER to run the statement.
  + When prompted for the password, copy and paste the **DBUserPasswd** value from the left side of the lab page, and press ENTER.
  + **Command:** Run the following statement:

USE ontimeflights;

[Click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#reconnect-home1) to return to the task.

RECONNECT TO A DATABASE INSTANCE

To reconnect to a database instance in Session Manager, follow these steps:

* **Copy/Paste:** Copy the following command to a text editor:

mysql -u (DBUserName) -p -h (SmallDBEndpoint)

* Replace **(DBUserName)** and **(SmallDBEndpoint)** with the corresponding values from the left side of the lab page.
* Copy and paste the updated statement from the text editor into the terminal window. Press ENTER to run the statement.
* When prompted for the password, copy and paste the **DBUserPasswd** value from the left side of the lab page, and press ENTER.
* **Command:** Run the following statement:

USE ontimeflights;

[Click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#reconnect-mysql-home) to return to the task.

ACCESS A LINUX EC2 INSTANCE: WINDOWS USERS

Access to an Amazon EC2 Linux instance requires a secure connection using an SSH client. The following directions walk you through the process of connecting to your Amazon Linux EC2 instance.

**Note:** Only perform the following steps if you are connecting from a Windows machine. If you are connecting from a macOS or Linux machine, [click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#ssh-MACLinux) for instructions.

1. On the left side of the lab page, click  **Download PPK**. Save the PPK file to the directory of your choice.
2. Open PuTTY (from the **Start** menu, choose **PuTTY** > **PuTTY**).

**Note:** If PuTTY is not already installed on your computer, download and install it from the following URL: <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>. If you already have an older version of PuTTY installed, we recommend that you download the latest version.

1. In the **Category** pane, choose **Session** and configure the following:

* For **Host Name**, enter

<user\_name>@<CommandHost>

, where

<CommandHost>

 is the IP value for the instance from the left side of the lab page.

**Note:** For Amazon Linux 2 or the Amazon Linux AMI, the user name is

ec2-user

. For an Ubuntu AMI, the user name is

ubuntu

.

* For **Connection type**, select **SSH**
* Ensure that the **Port** value is **22**

1. (Optional) Configure PuTTY to automatically send ‘keepalive’ data at regular intervals to keep the session active. This is useful to avoid disconnecting from your instance due to session inactivity. To do this, in the **Category** pane, choose **Connection**. For **Seconds between keepalives**, enter

30

.

1. In the **Category** pane, expand **Connection**, expand **SSH**, and then choose **Auth**. Complete the following:

* Choose **Browse**.
* Select the .ppk file that you downloaded earlier, and choose **Open**.

**Note:** This .ppk file is usually located in the **Downloads** folder on your PC.

* (Optional) If you plan to start this session again later, you can save the session information for future use. Under **Category**, choose **Session**, enter a name for the session in **Saved Sessions**, and then choose **Save**.

1. To connect to the instance: In the **Category** pane, choose **Session**, and then choose **Open**.
2. If this is the first time you have connected to this instance, PuTTY displays a security alert dialog box that asks whether you trust the host to which you are connecting. Choose **Yes**. A window opens and you are connected to your instance.

**Note:** If you receive an error while attempting to connect to your instance, see [Troubleshooting Connecting to Your Instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html).

ACCESS A LINUX EC2 INSTANCE: MACOS AND LINUX USERS

**Note:** Only perform the following steps if you are connecting from a macOS or Linux machine. If you are connecting from a Windows machine, [click here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#ssh-instructions) for instructions.

1. On the left side of the lab page, click  **Download PEM**. Save the file to the directory of your choice.
2. Open the SSH client on your local computer.

**Note:** Your local computer most likely has an SSH client installed by default. You can check for an SSH client by typing

ssh

 at the command line. If your local computer doesn’t recognize the command, you can install an SSH client. For information about installing an SSH client on Linux or macOS X, see [http://www.openssh.com](http://www.openssh.com/).

Complete the remaining connection steps in the terminal window.

1. Change the directory to the folder where you downloaded the PEM file.

**Note:** The PEM file is usually located in the **Downloads** folder on your computer. Access this directory by typing

cd ~/Downloads

1. Your key must not be publicly viewable for SSH to work. Change the permissions on the PEM file by running the following command. Replace

<PEM\_FILE>

 with the name of the PEM file you downloaded:

chmod 400 <PEM\_FILE>

1. Log in to the remote instance by running the following command. Replace

<PEM\_FILE>

 with the name of the PEM file you downloaded,

<user\_name>

 with the user name for the instance type you are connecting to, and

<CommandHost>

 is the IP value for the instance from the left side of the lab page:

ssh -i <PEM\_FILE> <user\_name>@<CommandHost>

**Note:** For Amazon Linux 2 or the Amazon Linux AMI, the user name is

ec2-user

. For an Ubuntu AMI, the user name is

ubuntu

.

1. **Expected Sample output:** If this is the first time you have connected to this instance, you see a response similar to the following:

The authenticity of host 'ec2-192-0-2-111.compute-1.amazonaws.com (192.0.2.111)'

can't be established.

RSA key fingerprint is 1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f.

Are you sure you want to continue connecting (yes/no)?

1. When prompted, enter

yes

You are now connected to your instance.

**Note:** If you receive an error while attempting to connect to your instance, see [Troubleshooting Connecting to Your Instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html).

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. While not all of the icons will be used, the following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A note, tip, or important guidance.
* **Copy/Paste:** A code block that displays the contents of a script or file you need to copy and paste that has been pre-created for you. When you need to copy only a certain part of a code block, there will be numbered **TODO** comments in the code.
* **WARNING:** An action that is irreversible and could potentially impact the failure of a command or process (including warnings about configurations that cannot be changed after they are made).
* **Additional information:** Where to find more information.

[Return to the instructions](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWAAD-1%3A1.0.9-b4d8812b/en-US#prerequisites)