

Amazon Aurora Labs for MySQL

Welcome to the AWS workshop and lab content portal for [Amazon Aurora MySQL](#) compatible databases! Here you will find a collection of workshops and other hands-on content aimed at helping you gain an understanding of the Amazon Aurora features and capabilities.

The resources on this site include a collection of easy to follow instructions with examples, templates to help you get started and scripts automating tasks supporting the hands-on labs. These resources are focused on helping you discover how advanced features of the Amazon Aurora MySQL database operate. Prior expertise with AWS and MySQL-based databases is beneficial, but not required to complete the labs.

Getting Started

Create an AWS account

In order to complete the hands-on content on this site, you'll need an AWS Account. We strongly recommend that you use a personal account or create a new AWS account to ensure you have the necessary access and that you do not accidentally modify corporate resources. Do not use an AWS account from the company you work for unless they provide sandbox accounts just for this purpose.

If you are setting up an AWS account for the first time, follow the instructions below to [create an administrative IAM user account](#), we recommend not using your AWS account root credentials for day to day usage. If you have received credits to complete these labs follow the instructions below on [adding the credits](#) to your AWS account.

Overview of labs

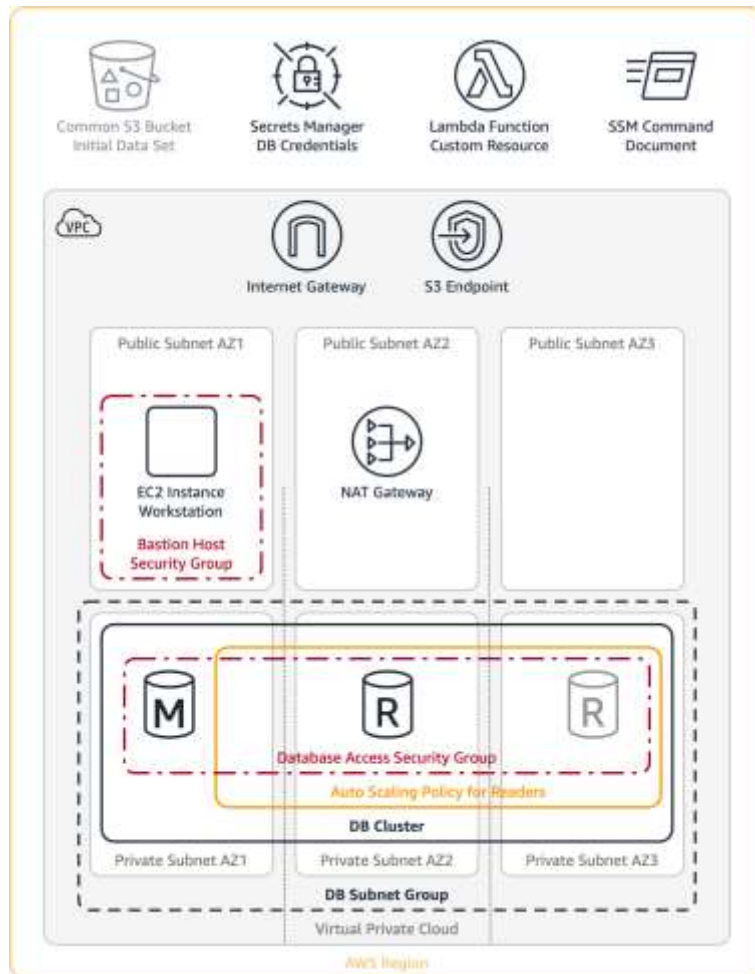
The following labs are currently available, part of this instructional website:

1 Prerequisites	Required, start here	Set up the lab environment and provision the prerequisite resources
2 Create a New Aurora Cluster	Optional	Create a new Amazon Aurora MySQL DB cluster manually
3 Connecting, Loading Data and Auto Scaling	Recommended, for provisioned clusters	Connect to the DB cluster for the first time, load an initial data set and test read replica auto scaling. The initial data set may be used in subsequent labs.
4 Cloning Clusters	Recommended, for provisioned clusters	Cloning an Aurora DB cluster and observing the divergence of the data set.
5 Backtracking a Cluster	Recommended, for provisioned clusters	Backtracking an Aurora DB cluster to fix an accidental DDL operation.
6 Using Performance Insights	Recommended, for provisioned clusters	Examining the performance of your DB instances using RDS Performance Insights
7 Creating a Serverless Aurora Cluster	Recommended, for serverless clusters	Create a new Amazon Aurora Serverless MySQL DB cluster manually. You may skip the provisioned cluster labs if you are planning to operate a serverless workload.
8 Using Aurora Serverless with Lambda Functions	Recommended, for serverless clusters	Connect to your serverless cluster using the RDS Data API and Lambda functions. Requires the previous lab.

You can also discover exercises, labs and workshops related to Amazon Aurora on the [Related Labs and Workshops](#) page.

Lab environment at a glance

To simplify the getting started experience with the labs, we have created foundational templates for [AWS CloudFormation](#) that provision the resources needed for the lab environment. These templates are designed to deploy a consistent networking infrastructure, and client-side experience of software packages and components used in the lab.



The environment deployed using CloudFormation includes several components:

- [Amazon VPC](#) network configuration with public and private subnets
- [Database subnet group](#) and relevant [security groups](#) for the cluster and workstation
- [Amazon EC2 instance](#) configured with the software components needed for the lab
- [IAM roles](#) with access permissions for the workstation and cluster permissions for [enhanced monitoring](#), S3 access and logging
- Custom cluster and DB instance [parameter groups](#) for the Amazon Aurora cluster, enabling logging and performance schema
- Optionally, [Amazon Aurora](#) DB cluster with 2 nodes: a writer and read replica
- If the cluster is created for you, the master database credentials will be generated automatically and stored in an [AWS Secrets Manager](#) secret.
- Optionally, [read replica auto scaling](#) configuration
- Optionally, [AWS Systems Manager](#) command document to execute a load test

Create an IAM user (with admin permissions)

If you don't already have an AWS IAM user with admin permissions, please use the following instructions to create one:

1. Browse to the [AWS IAM](#) console.
2. Click **Users** on the left navigation and then click **Add User**.
3. Enter a **User Name**, check the checkbox for **AWS Management Console access**, enter a **Custom Password**, and click **Next:Permissions**.
4. Click **Attach existing policies directly**, click the checkbox next to the **AdministratorAccess** policy, and click **Next:Review**.
5. Click **Create User**
6. Click **Dashboard** on the left navigation and use the **IAM users sign-in link** to login as the admin user you just created.

Add credits (optional)

If you are doing these workshop as part of an AWS sponsored event that doesn't provide AWS accounts, you will receive credits to cover the costs. Below are the instructions for entering the credits:

1. Browse to the AWS Account Settings console.

2. Enter the Promo Code you received (these will be handed out at the beginning of the workshop).
3. Enter the Security Check and click Redeem.

Additional software needed for labs

The templates and scripts setting up the lab environment install the following software in the lab environment for the purposes of deploying and running the labs:

- [mysql-client](#) package. MySQL open source software is provided under the GPL License.
- [sysbench](#) available using the GPL License.
- [test_db](#) available using the Creative Commons Attribution-Share Alike 3.0 Unported License.
- [Percona's sysbench-tpcc](#) available using the Apache License 2.0.

Prerequisites

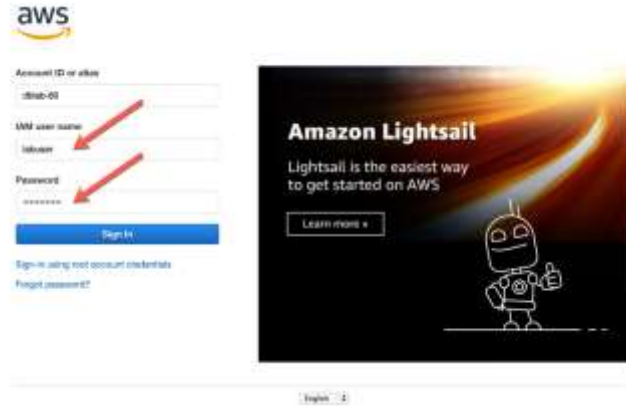
The following steps should be completed before getting started with any of the labs in this repository. Not all steps may apply to all students or environments.

This lab contains the following tasks:

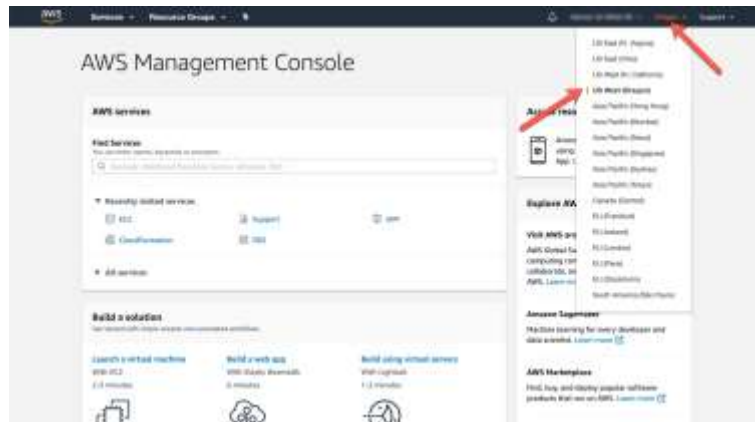
1. Signing in to the AWS Management Console
2. Creating a lab environment using AWS CloudFormation

1. Signing in to the AWS Management Console

If you are running these labs in a formal, instructional setting, please use the Console URL, and credentials provided to you to access and log into the AWS Management Console. Otherwise, please use your own credentials. You can access the console at: <https://console.aws.amazon.com/> or through the Single Sign-On (SSO) mechanism provided by your organization.



If you are running these labs in a formal, instructional setting, please use the AWS region provided. Ensure the correct AWS region is selected in the top right corner, if not use that dropdown to choose the correct region. The labs are designed to work in any of the regions where Amazon Aurora MySQL compatible is available. However, not all features and capabilities of Amazon Aurora may be available in all supported regions at this time.



2. Creating a lab environment using AWS CloudFormation

To simplify the getting started experience with the labs, we have created foundational templates for [AWS CloudFormation](#) that provision the resources needed for the lab environment. These templates are designed to deploy a consistent networking infrastructure, and client-side experience of software packages and components used in the lab.

Formal Event

If you are running these labs in a formal instructional event, the lab environment may have been initialized on your behalf. If unsure, please verify with the event support staff.

Please download the most appropriate CloudFormation template based on the labs you want to run:

I will create the [lab-no-DB cluster manually](#)

Launch Stack



Use when you wish to provision the initial cluster manually by following [Lab 1. - Creating a New Aurora Cluster](#)

Provision the [lab-with-DB cluster for me](#)

Launch Stack



Use when you wish to skip the initial cluster creation lab, and have the DB cluster provisioned for you, so you can continue from [Lab 2. - Cluster Endpoints and Read Replica Auto Scaling](#)

If you downloaded the template, save it in a memorable location such as your desktop, you will need to reference it later.

Open the [CloudFormation service console](#).

Region Check

Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.

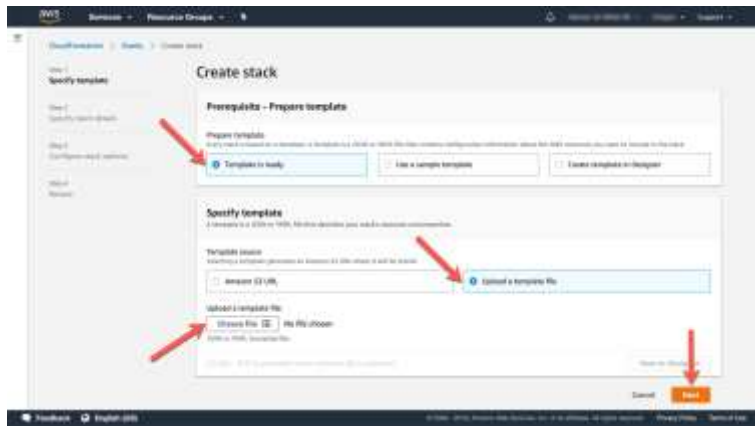
Click **Create Stack**.

Note

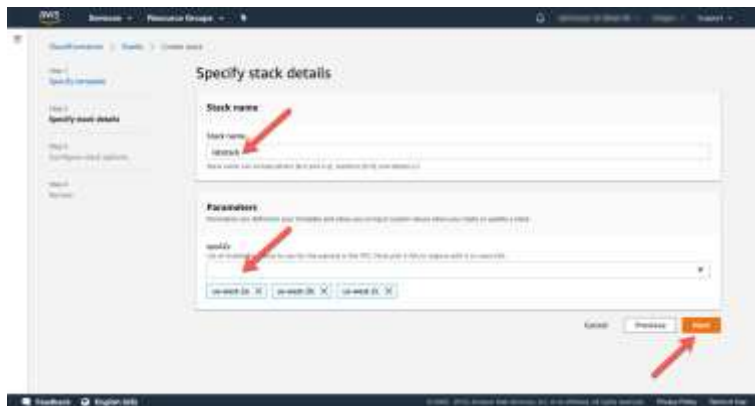
The CloudFormation console has been upgraded recently. Depending on your previous usage of the CloudFormation console UI, you may see the old design or the new design, you may also be presented with a prompt to toggle between them. In this lab we are using the new design for reference, although the steps will work similarly in the old console design as well, if you are more familiar with it.



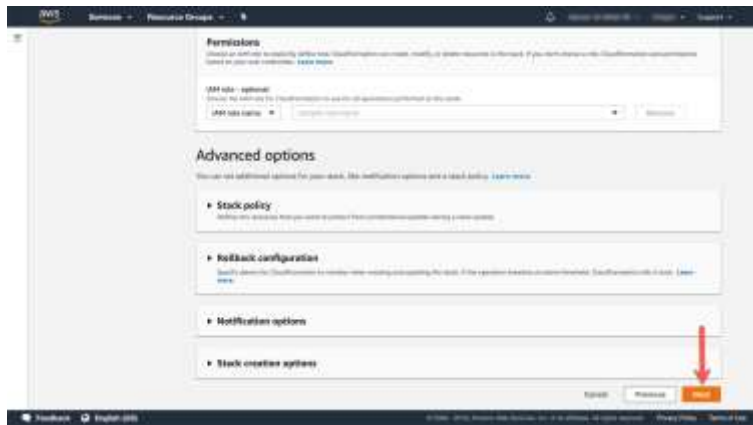
Select the radio button named **Upload a template**, then **Choose file** and select the template file you downloaded previously named and then click **Next**.



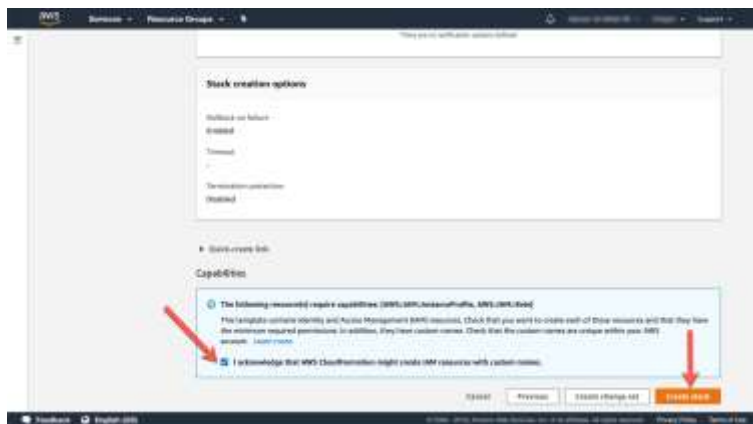
In the field named **Stack Name**, enter the value `labstack`. For the **vpcAZs** parameter select 3 availability zones (AZs) from the dropdown. If your desired region only supports 2 AZs, please select just the two AZs available. Click **Next**.



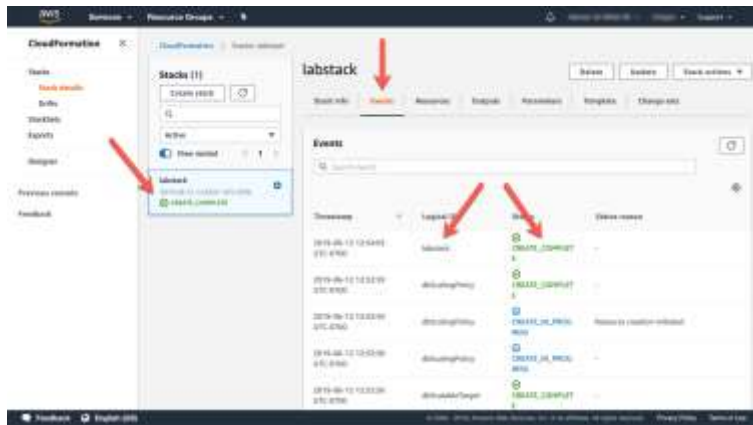
On the **Configure stack options** page, leave the defaults as they are, scroll to the bottom and click **Next**.



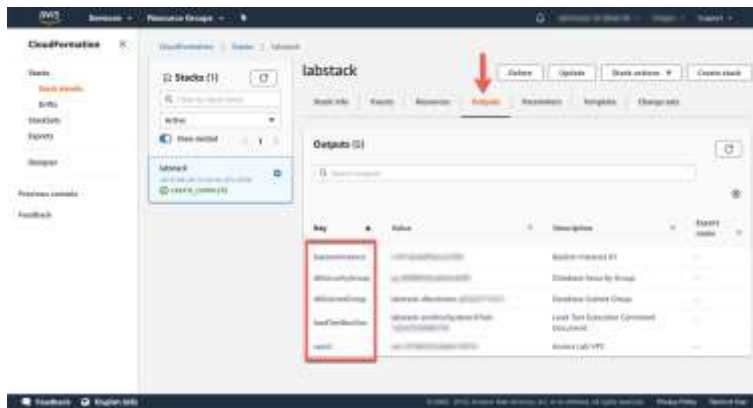
On the **Review labstack** page, scroll to the bottom, check the box that reads: **I acknowledge that AWS CloudFormation might create IAM resources with custom names** and then click **Create**.



The stack will take approximatively 20 minutes to provision, you can monitor the status on the **Stack detail** page. You can monitor the progress of the stack creation process by refreshing the **Events** tab. The latest event in the list will indicate `CREATE_COMPLETE` for the stack resource.



Once the status of the stack is `CREATE_COMPLETE`, click on the **Outputs** tab. The values here will be critical to the completion of the remainder of the lab. Please take a moment to save these values somewhere that you will have easy access to them during the remainder of the lab. The names that appear in the **Key** column are referenced directly in the instructions in subsequent steps, using the parameter format: `[outputKey]`



Creating a New Aurora Cluster

Note

If you are familiar with the basic concepts of Amazon Aurora MySQL, and have created a cluster in the past, you may skip this module, by using provisioning the lab environment using the [lab-with-cluster.yml](#) CloudFormation template, so the DB cluster is provisioned for you. Skip to [Connecting, Loading Data and Auto Scaling](#).

This lab will walk you through the steps of creating an Amazon Aurora database cluster manually, and configuring app the parameters required for the cluster components. At the end of this lab you will have a database cluster ready to be used in subsequent labs.

This lab contains the following tasks:

1. Creating the DB cluster
2. Retrieving the DB cluster endpoints
3. Assigning an IAM role to the DB cluster
4. Creating a replica auto scaling policy

This lab requires the following lab modules to be completed first:

- [Prerequisites](#) (using `lab-no-cluster.yml` template is sufficient)

1. Creating the DB cluster

Open the [Amazon RDS service console](#).

Region Check

Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.

Click **Create database** to start the configuration process

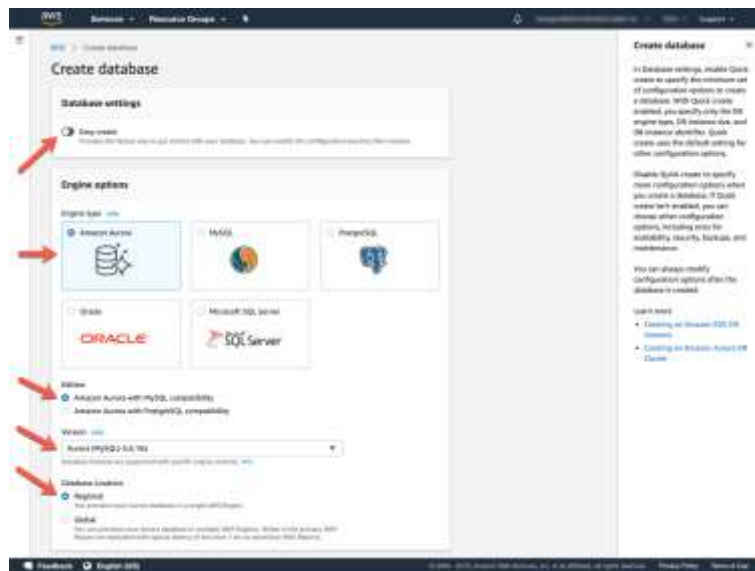
Note

The RDS console database creation workflow has been simplified recently. Depending on your previous usage of the RDS console UI, you may see the old workflow or the new one, you may also be presented with a prompt to toggle between them. In this lab we are using the new workflow for reference, although the steps will work similarly in the old console workflow as well, if you are more familiar with it.



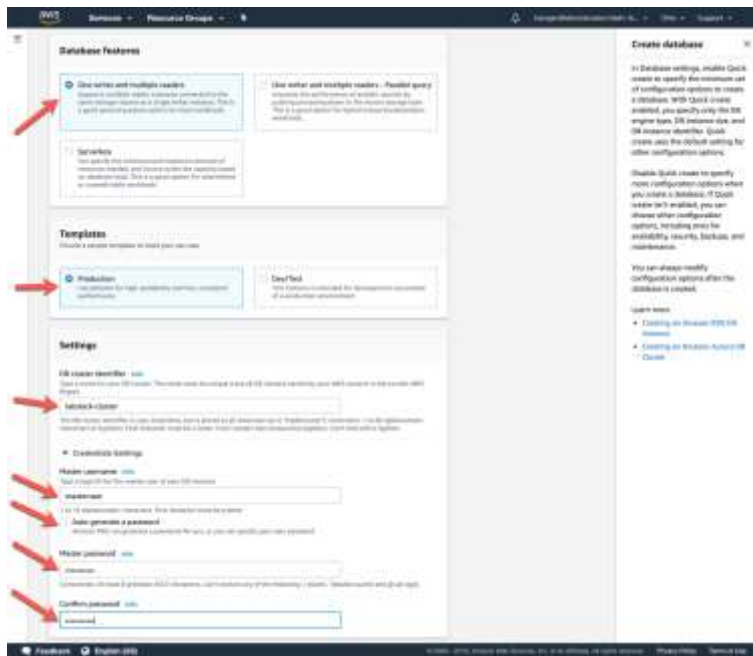
In the first configuration section of the **Create database** page, called **Database settings** ensure the **Easy create** toggle button is turned **OFF** (grey color).

Next, in the **Engine options** section, choose the **Amazon Aurora** engine type, the **Amazon Aurora with MySQL compatibility** edition, the **Aurora (MySQL 5.6) 1.19.2** version and the **Regional** database location.



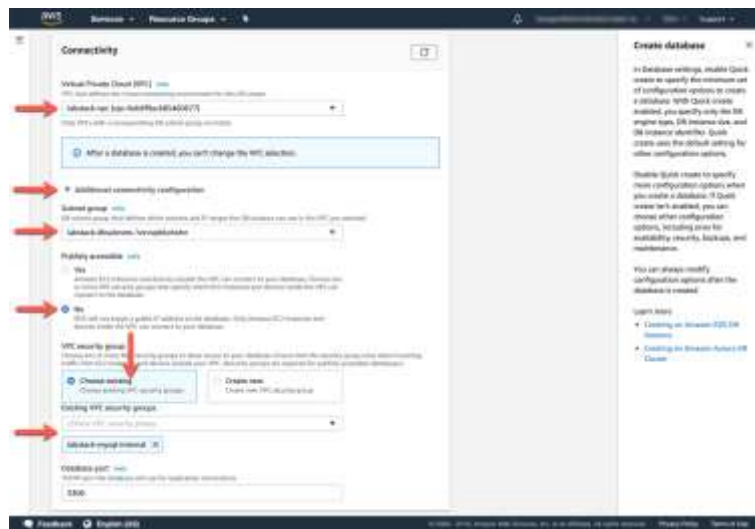
In the **Database features** section, select **One writer and multiple readers**, and in the **Templates** section, select **Production**. The selections so far will instruct AWS to create an Aurora MySQL database cluster with the most recent version of the MySQL 5.6 compatible engine in a highly available configuration with one writer and one reader database instance in the cluster.

In the **Settings** section give your database cluster a recognizable identifier, such as `labstack-cluster`. Configure the name and password of the master database user, with the most elevated permissions in the database. We recommend to use the user name `masteruser` for consistency with subsequent labs and a password of your choosing. For simplicity ensure the check box **Auto generate a password** is **not checked**.



In the **Connectivity** section, expand the sub-section called **Additional connectivity configuration**. This section allows you to specify where the database cluster will be deployed within your defined network configuration. To simplify the labs, the CloudFormation stack you deployed in the preceding [Prerequisites](#) module, has configured a VPC that includes all resources needed for an Aurora database cluster. This includes the VPC itself, subnets, DB subnet groups, security groups and several other networking constructs. All you need to do is select the appropriate existing connectivity controls in this section.

Pick the **Virtual Private Cloud (VPC)** named after the CloudFormation stack name, such as `labstack-vpc`. Similarly make sure the selected **Subnet Group** also matches the stack name (e.g. `labstack-dbsubnets-[hash]`). Make sure the cluster **Publicly accessible** option is set to **No**. The lab environment also configured a **VPC security group** that allows your lab workspace EC2 instance to connect to the database. Make sure the **Choose existing** security group option is selected and from the dropdown pick the security group with a name ending in `-mysql-internal` (eg. `labstack-mysql-internal`). Please remove any other security groups, such as `default` from the selection.



Next, expand the **Advanced configuration** section. Set the **Initial database name** to `mylab`. For the **DB cluster parameter group** and **DB parameter group** selectors, choose the groups with the stack name in their name (e.g. `labstack-[...]`). Choose a 7 days **Backup retention period**. Check the box to **Enable encryption** and select the `[default] aws/rds` for the **Master key**.

Additional configuration

Database options

☒ **Enable performance insights**

Initial database name:

DB instance parameter group:

DB parameter group:

Options group:

Master key:

Backup

Backup retention period:

Encryption

☒ **Enable encryption**

Master key:

Create database

In Database settings, enable Quick Create to specify the minimum set of configuration options to create a database. With Quick Create enabled, you specify only the DB engine type, DB instance class, and DB instance identifier. Quick Create uses the default setting for other configuration options.

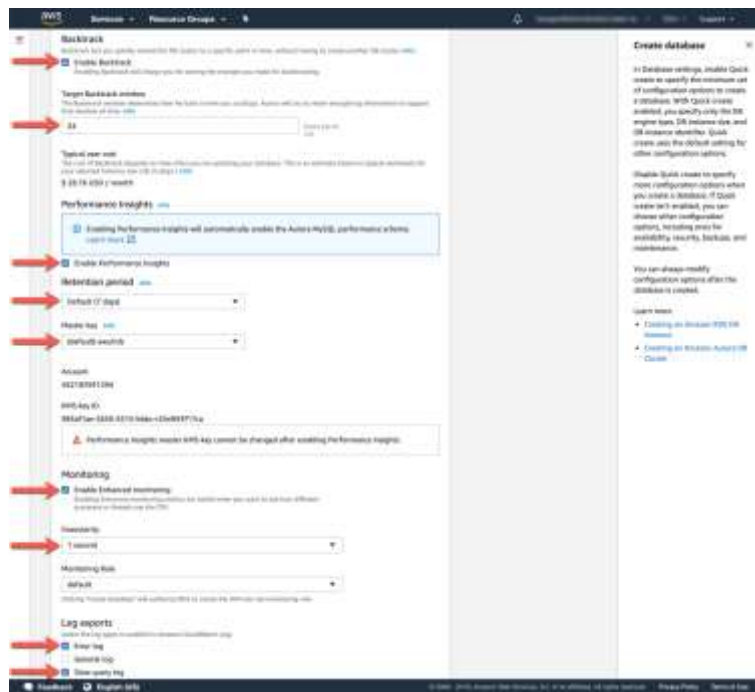
When Quick Create is specified, you can specify a database. If Quick Create is disabled, you can choose other configuration options, including ones for availability, security, backups, and maintenance.

You can check/modify configuration options after the database is created.

Learn more

- Creating an Amazon RDS DB instance
- Creating an Amazon Aurora DB instance

Enable the backtrack capability by checking the **Enable Backtrack** box and set a **Target backtrack window** of 24 hours. Check the box to **Enable Performance Insights** with a **Retention period of Default (7 days)** and use the [default] `aws/rds` Master key **for monitoring data encryption**. Next, check the **Enable Enhanced Monitoring** box, and select a **Granularity of 1 second**. For Log exports check the **Error log** and **Slow query log**** boxes.



Also in the **Advanced configuration** section, de-select the check box **Enable delete protection**. In a production use case, you will want to leave that option checked, but for testing purposes, un-checking this option will make it easier to clean up the resources once you have completed the labs.

Before continuing, let's summarize the configuration options selected. You will create a database cluster with the following characteristics:

- Aurora MySQL 5.6 compatible (latest stable engine version)
- Regional cluster composed of a writer and a reader DB instance in different availability zones (highly available)
- Deployed in the VPC and using the network configuration of the lab environment
- Using custom database engine parameters that enable the slow query log, S3 access and tune a few other configurations
- Automatically backed up continuously, retaining backups for 7 days

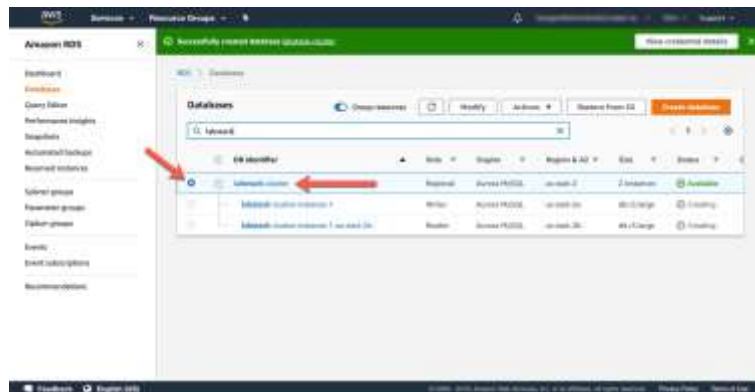
- Using data at rest encryption
- Retaining 24 hours worth of change data for backtrack purposes
- With Enhanced Monitoring and Performance Insights enabled

Click **Create database** to provision the DB cluster.

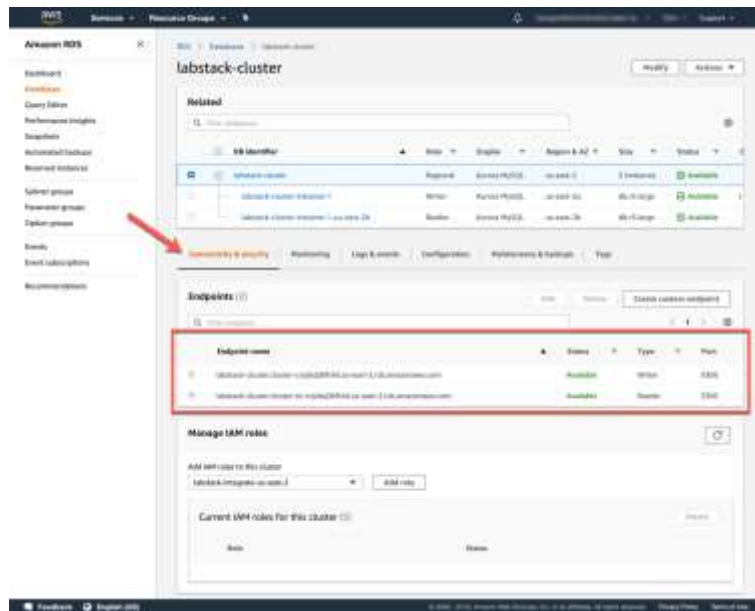


2. Retrieving the DB cluster endpoints

The database cluster may take several minutes to provision, including the DB instances making up the cluster. In order to connect to the DB cluster and start using it in subsequent labs, you need to retrieve the DB cluster endpoints. There are two endpoints created by default. The **Cluster Endpoint** will always point to the **writer** DB instance of the cluster, and should be used for both writes and reads. The **Reader Endpoint** will always resolve to one of the **reader** DB instances and should be used to offload read operations to read replicas. In the RDS console, go to the DB cluster detail view by clicking on the cluster DB identifier.

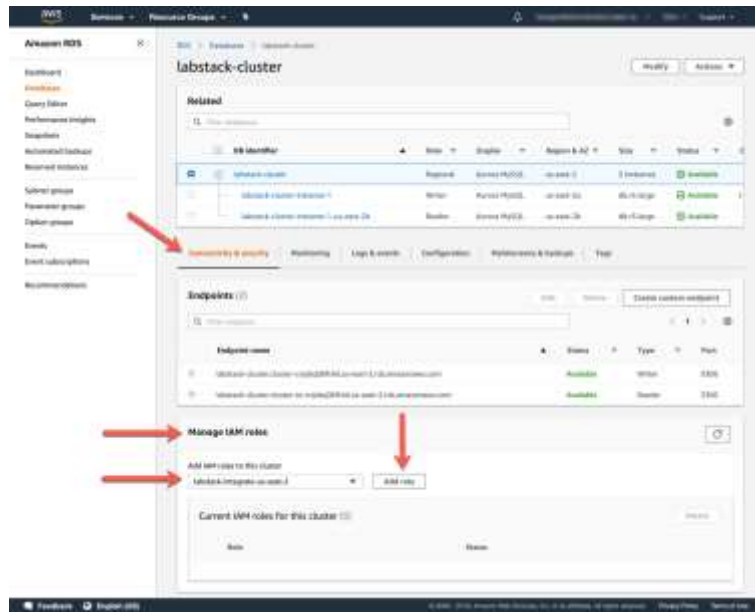


The **Endpoints** section in the **Connectivity and security** tab of the details page displays the endpoints. Note these values down, as you will use them later.



3. Assigning an IAM role to the DB cluster

Once created, you should assign an IAM role to the DB cluster, in order to allow the cluster access to Amazon S3 for importing and exporting data. The IAM role has already been created using CloudFormation when you created the lab environment. On the same DB cluster detail page as before, in the **Manage IAM roles** section, choose the IAM role named after the stack name, ending in `-integrate-[region]` (e.g. `labstack-integrate-[region]`). Then click **Add role**.

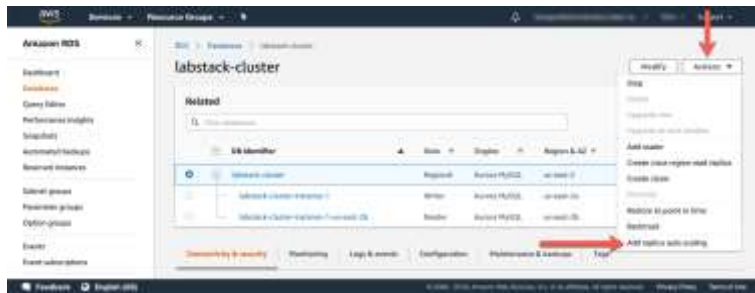


Once the operation completes the **Status** of the role will change from `Pending` to `Active`.

4. Creating a replica auto scaling policy

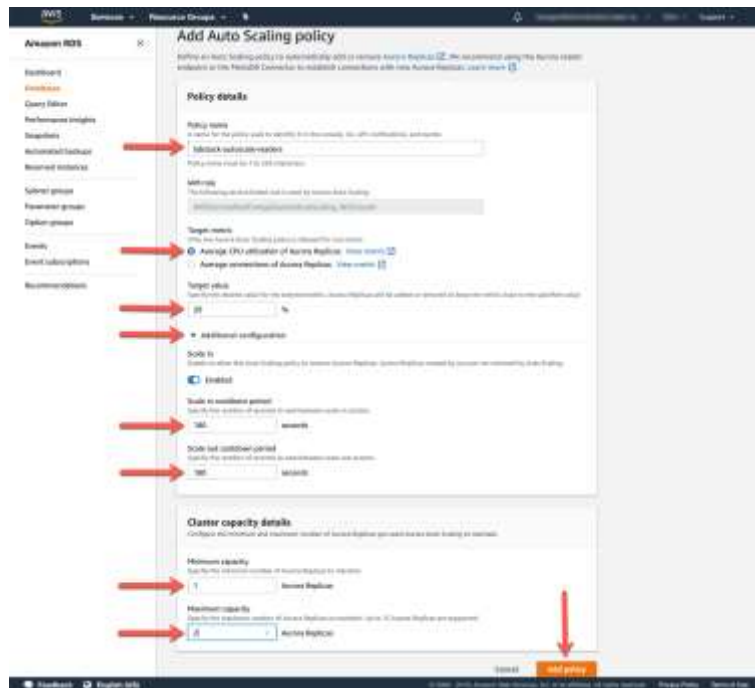
Finally, you will add a read replica auto scaling configuration to the DB cluster. This will allow the DB cluster to scale the number of reader DB instances that operate in the DB cluster at any given point in time based on the load.

In the top right corner of the details page, click on **Actions** and then on **Add replica auto scaling**.



Provide a **Policy name** based on the stack name, such as `labstack-autoscale-readers`. For the **Target metric** choose **Average CPU utilization of Aurora Replicas**. Enter a **Target value** of 20 percent. In a production use case this value may need to be set much higher, but we are using a lower value for demonstration purposes. Next, expand the **Additional configuration** section, and change both the **Scale in cooldown period** and **Scale out cooldown period** to a value of 180 seconds. This will reduce the time you have to wait between scaling operations in subsequent labs.

In the **Cluster capacity details** section, set the **Minimum capacity** to 1 and **Maximum capacity** to 2. In a production use case you may need to use different values, but for demonstration purposes, and to limit the cost of associated with the labs we limit the number of readers to two. Next click **Add policy**.



Connecting, Loading Data and Auto Scaling

This lab will walk you through the process of connecting to the DB cluster you have just created, and using the cluster for the first time. At the end you will test out how Aurora read replica auto scaling works in practice using a load generator script.

This lab contains the following tasks:

1. Connecting to your workstation EC2 instance
2. Connecting to the DB cluster
3. Loading an initial data set from S3

4. Running a read-only workload

This lab requires the following lab modules to be completed first:

- [Prerequisites](#)
- [Creating a New Aurora Cluster](#) (conditional, if creating a cluster manually)

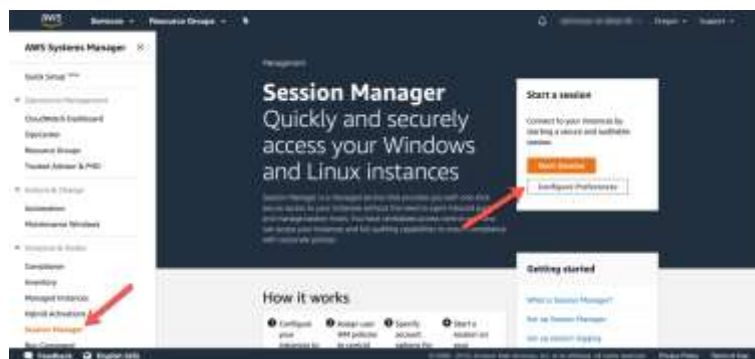
1. Connecting to your workstation EC2 instance

To interact with the Aurora database cluster, you will use an Amazon EC2 Linux instance that acts like a workstation for the purposes of the labs. All necessary software packages and scripts have been installed and configured on this EC2 instance for you. To ensure a unified experience, you will be interacting with this workstation using [AWS Systems Manager Session Manager](#). With Session Manager you can interact with your workstation directly from the management console.

Open the [Systems Manager service console](#). In the left hand menu, click on **Session Manager**. Then click the **Configure Preferences** button.

Region Check

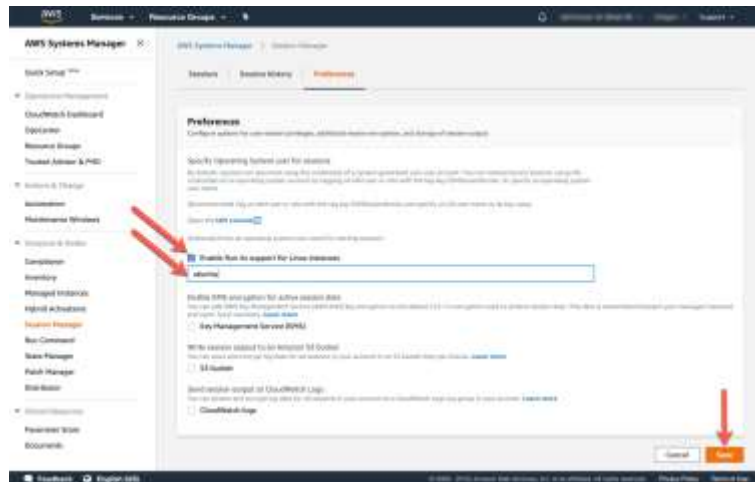
Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.



Check the box next to **Enable Run As support for Linux instances**, and enter `ubuntu` in the text field. This will instruct Session Manager to connect to the workstation using the `ubuntu` operating system user account. Click **Save**.

Note

You will only need to set the preferences once for the purposes of the labs. However, if you use Session Manager for other use cases you may need to revert the changes as needed.



Next, navigate to the **Sessions** tab, and click the **Start session** button.



Please select the EC2 instance to establish a session with. The workstation is named `labstack-bastion-host`, select it and click **Start session**.



If you see a black command like terminal screen and a prompt, you are now connected to the EC2 based workstation. With Session Manager it is not necessary to allow SSH access to the EC2 instance from a network level, reducing the attack surface of that EC2 instance.

If you have completed the previous lab, and created the Aurora DB cluster manually, please execute the following commands to ensure you have a consistent experience compared for subsequent labs. These commands will save the database username and password in environment variables. When the cluster is provisioned automatically by CloudFormation this is done automatically.

```
bash
cd ~
export DBUSER="masteruser"
export DBPASS="<type your password>"
echo "export DBPASS=\"$DBPASS\"" >> /home/ubuntu/.bashrc
echo "export DBUSER=$DBUSER" >> /home/ubuntu/.bashrc
```

If you have not created the DB cluster manually, execute the following commands to ensure a consistent experience.

```
bash
cd ~
```

2. Connecting to the DB cluster

Connect to the Aurora database just like you would to any other MySQL-based database, using a compatible client tool. In this lab you will be using the `mysql` command line tool to connect. The command is as follows:

```
mysql -h [clusterEndpoint] -u$DBUSER -p"$DBPASS" mylab
```

If you have completed the previous lab, and created the Aurora DB cluster manually, you would input the **Cluster Endpoint** of that cluster as displayed at the end of that lab.

However, if you have skipped that lab and provisioned the cluster using the CloudFormation template, you can find the value for the cluster endpoint parameter in the stack outputs. We have set the DB cluster's database credentials automatically for you, and have also created the schema named `mylab` as well. The credentials were saved to an [AWS SecretsManager](#) secret.

Command parameter values at a glance:

-h	[clusterEndpoint]	See CloudFormation stack output	See previous lab	The cluster endpoint of the Aurora DB cluster.
-u	\$DBUSER	Set automatically, see Secrets Manager	<code>masteruser</code> or manually set	The user name of the MySQL user to authenticate as.
-p	\$DBPASS	Set automatically, see Secrets Manager	Manually set	The password of the MySQL user to authenticate as.
	[database]	<code>mylab</code>	<code>mylab</code> or manually set	The schema (database) to use by default.

Note

You can view and retrieve the credentials stored in the secret using the following command:

```
aws secretsmanager get-secret-value --secret-id [secretArn] | jq -r '.SecretString'
```

Once connected to the database, use the code below to create a stored procedure we'll use later in the labs, to generate load on the DB cluster. Run the following SQL queries:

```
DELIMITER $$  
DROP PROCEDURE IF EXISTS minute_rollup$$
```

```

CREATE PROCEDURE minute_rollup(input_number INT)
BEGIN
  DECLARE counter int;
  DECLARE out_number float;
  set counter=0;
  WHILE counter <= input_number DO
    SET out_number=SQRT(rand());
    SET counter = counter + 1;
  END WHILE;
END$$
DELIMITER ;

```

3. Loading an initial data set from S3

Once connected to the DB cluster, run the following SQL queries to create an initial table:

```

DROP TABLE IF EXISTS `sbtest1`;
CREATE TABLE `sbtest1` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `k` int(10) unsigned NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

Next, load an initial data set by importing data from an Amazon S3 bucket:

```

LOAD DATA FROM S3 MANIFEST
's3-eu-west-1://aurorareinvent2018/output.manifest'
REPLACE
INTO TABLE sbtest1
CHARACTER SET 'latin1'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\r\n';

```

Data loading may take several minutes, you will receive a successful query message once it completes. When completed, exit the MySQL command line:

```
quit;
```

4. Running a read-only workload

Once the data load completes successfully, you can run a read-only workload to generate load on the cluster. You will also observe the effects on the DB cluster topology. For this step you will use the **Reader Endpoint** of the cluster. If you created the cluster manually, you can find the endpoint value as indicated at the end of that lab. If the DB cluster was created automatically for you the value can be found in your CloudFormation stack outputs.

Run the load generation script from the Session Manager workstation command line:

```
python3 loadtest.py -e [readerEndpoint] -u $DBUSER -p "$DBPASS" -d mylab
```

Command parameter values at a glance:

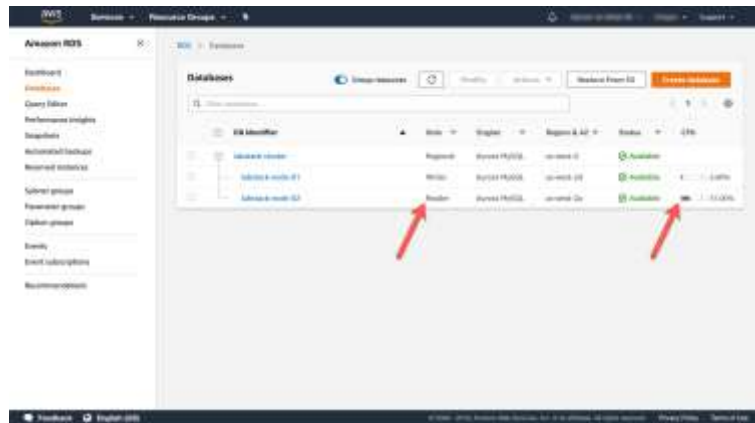
-e	[readerEndpoint]	See CloudFormation stack output	See previous lab	The reader endpoint of the Aurora DB cluster.
-u	\$DBUSER	Set automatically, see Secrets Manager	masteruser or manually set	The user name of the MySQL user to authenticate as.
-p	\$DBPASS	Set automatically, see Secrets Manager	Manually set	The password of the MySQL user to authenticate as.
-d	[database]	mylab	mylab or manually set	The schema (database) to generate load against.
-t		64 (default)	64 (default)	The number of client connections (threads) to use concurrently.

Now, open the [Amazon RDS service console](#) in a different browser tab.

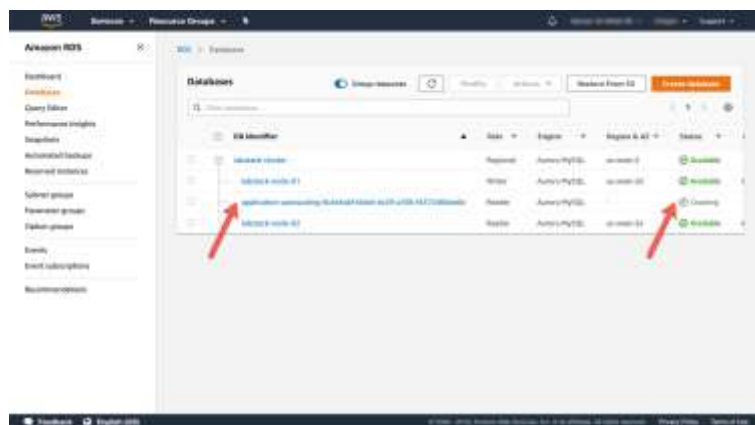
Region Check

Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.

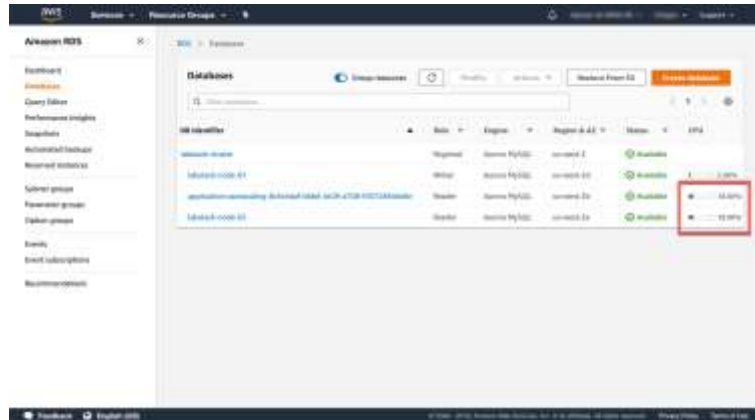
Take note that the reader node is currently receiving load. It may take a minute or more for the metrics to fully reflect the incoming load.



After several minutes return to the list of instances and notice that a new reader is being provisioned to your cluster.



Once the new replica becomes available, note that the load distributes and stabilizes (it may take a few minutes to stabilize).



DB instance	Role	Engine	Engine & AZ	Status	CPU
db-1	Primary	Amazon RDS	us-east-1	Available	10.00%
db-2	Secondary	Amazon RDS	us-east-1	Available	10.00%
db-3	Secondary	Amazon RDS	us-east-1	Available	10.00%

You can now toggle back to the Session Manager command line, and type `CTRL+C` to quit the load generator. After a while the additional reader will be removed automatically.

Cloning Clusters

This lab will walk you through the process of [cloning a DB cluster](#). Cloning creates a separate, independent DB cluster, with a consistent copy of your data set as of the time you cloned it. Database cloning uses a copy-on-write protocol, in which data is copied at the time that data changes, either on the source databases or the clone databases. The two clusters are isolated, and there is no performance impact on the source DB cluster from database operations on the clone, or the other way around.

This lab contains the following tasks:

1. Creating a clone DB cluster
2. Verifying that the data set is identical
3. Changing data on the clone

4. Verifying that the data diverges

This lab requires the following lab modules to be completed first:

- [Prerequisites](#)
- [Creating a New Aurora Cluster](#) (conditional, if creating a cluster manually)
- [Connecting, Loading Data and Auto Scaling](#) (connectivity and data loading sections only)

1. Creating a clone DB cluster

Workload State Check

Before cloning the DB cluster ensure you have stopped any load generating tasks from the previous lab, and exited out of the MySQL client command line using `quit;.`

On the Session Manager workstation command line [see the previous lab](#), enter:

```
aws rds restore-db-cluster-to-point-in-time \  
--restore-type copy-on-write \  
--use-latest-restorable-time \  
--source-db-cluster-identifier labstack-cluster \  
--db-cluster-identifier labstack-cluster-clone \  
--vpc-security-group-ids [dbSecurityGroup] \  
--db-subnet-group-name [dbSubnetGroup] \  
--backtrack-window 86400
```

You can find the parameter values for the [placeholders] above in the Outputs section of the CloudFormation stack you deployed when [completing the prerequisites](#). If you have opted to create the DB cluster manually, and have specified a different DB cluster identifier, please use that value.

Next, check the status of the creation of your clone, by using the following command. The cloning process can take several minutes to complete. See the example output below.

```
aws rds describe-db-clusters \  
--db-cluster-identifier labstack-cluster-clone \  
| jq -r '.DBClusters[0].Status, .DBClusters[0].Endpoint'
```


Take note of both the status and the endpoint in the command output. Once the **status** becomes **available**, you can add a DB instance to the cluster and once the DB instance is added, you will want to connect to the cluster via the **endpoint** value, which represents the cluster endpoint.



Note

Creating a DB cluster clone, even without adding DB instances, can be a useful and cost effective safety net measure, if you are about to make significant changes to your source cluster (such as an upgrade or risky DDL operation). The clone then becomes a quick rollback target, in case you encounter issues on the source as a result of the operation. You simply add a DB instance and point your application to the clone in such an event. We do not recommend using clones as long term point in time snapshot tools, as you are limited to 15 clones derived directly or indirectly from the same source.

Add a DB instance to the cluster once the status of the cluster becomes **available**, using the following command:

```
aws rds create-db-instance \  
--db-instance-class db.r5.large \  
--engine aurora \  
--db-cluster-identifier labstack-cluster-clone \  
--db-instance-identifier labstack-cluster-clone-instance
```

Check the creation of the DB instance within the cluster, by using the following command:

```
aws rds describe-db-instances \  
--db-instance-identifier labstack-cluster-clone-instance \  
& jq -r '.DBInstances[0].DBInstanceStatus'
```



Repeat the command to monitor the creation status. Once the **status** changes from **creating** to **available**, you have a functioning clone. Creating a node in a cluster also takes several minutes.

2. Verifying that the data set is identical

Verify that the data set is identical on both the source and cloned DB clusters, before we make any changes to the data. You can verify by performing a checksum operation on the **sbtest1** table.

Connect to the cloned database using the following command (use the endpoint you retrieved from the `describe-db-cluster` command above):

```
mysql -h [cluster endpoint of clone] -u$DBUSER -p"$DBPASS" mylab
```

Command parameter values at a glance:

-h	[cluster endpoint of clone]	See above	See above	The cluster endpoint of the Aurora cloned DB cluster.
-u	\$DBUSER	Set automatically, see Secrets Manager	masteruser or manually set	The user name of the MySQL user to authenticate as.
-p	\$DBPASS	Set automatically, see Secrets Manager	Manually set	The password of the MySQL user to authenticate as.
	[database]	mylab	mylab or manually set	The schema (database) to use by default.

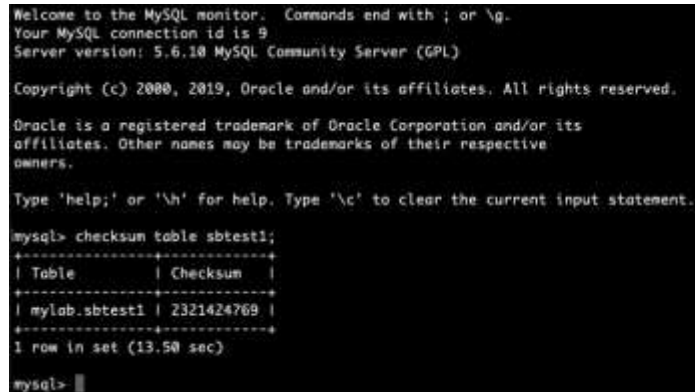
Note

The database credentials you use to connect are the same as for the source DB cluster, as this is an exact clone of the source.

Next, issue the following command on the clone:

```
checksum table sbtest1;
```

The output of your commands should look similar to the example below. Please take note of the value for your specific clone cluster.



```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.6.10 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> checksum table sbtest1;
+-----+-----+
| Table | Checksum |
+-----+-----+
| mylab.sbtest1 | 2321424769 |
+-----+-----+
1 row in set (13.50 sec)

mysql>
```

Now disconnect from the clone and connect to the source cluster with the following sequence:

```
quit;
```

```
mysql -h [clusterEndpoint] -u$DBUSER -p"$DBPASS" mylab
```

Execute the same checksum command that you ran on the clone:

```
checksum table sbtest1;
```

Please take note of the value for your specific source cluster. The checksum value should be the same as for the cloned cluster above.

3. Changing data on the clone

Disconnect from the original cluster (if you are still connected to it) and connect to the clone cluster with the following sequence:

```
quit;
```

```
mysql -h [cluster endpoint of clone] -u$DBUSER -p"$DBPASS" mylab
```

Delete a row of data and execute the checksum command again:

```
delete from sbtest1 where id = 1;
```

```
checksum table sbtest1;
```

The output of your commands should look similar to the example below. Notice that the checksum value changed, and is no longer the same as calculated in section 2 above.

```
Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> delete from sbtest1 where id = 1;
Query OK, 1 row affected (0.02 sec)

mysql> checksum table sbtest1;
+-----+-----+
| Table      | Checksum |
+-----+-----+
| mylab.sbtest1 | 822511330 |
+-----+-----+
1 row in set (1.04 sec)

mysql> 
```

4. Verifying that the data diverges

Verify that the checksum value did not change on the source cluster as a result of the delete operation on the clone. Disconnect from the clone (if you are still connected) and connect to the source cluster with the following sequence:

```
quit;
```

```
mysql -h [clusterEndpoint] -u$DBUSER -p"$DBPASS" mylab
```

Execute the same checksum command that you ran on the clone:

```
checksum table sbtest1;
```

Please take note of the value for your specific source cluster. The checksum value should be the same as calculated for the source cluster above in section 2.

Backtracking a Cluster

This lab will walk you through the process of backtracking a DB cluster. [Backtracking](#) "rewinds" the DB cluster to the time you specify. While it is not a replacement for backing up your DB cluster for DR purposes, backtracking allows you to easily undo mistakes quickly, or explore earlier data changes.

This lab contains the following tasks:

1. Making unintended data changes
2. Backtracking to recover from unintended changes

This lab requires the following lab modules to be completed first:

- [Prerequisites](#)
- [Creating a New Aurora Cluster](#) (conditional, if creating a cluster manually)
- [Connecting, Loading Data and Auto Scaling](#) (connectivity and data loading sections only)

1. Making unintended data changes

Connect to the DB cluster endpoint using the MySQL client, if you are not already connected after completing the previous lab:

```
mysql -h [clusterEndpoint] -u$DBUSER -p"$DBPASS" mylab
```

Command parameter values at a glance:

-h	[cluster endpoint of clone]	See above	See above	The cluster endpoint of the Aurora cloned DB cluster.
-u	<code>\$DBUSER</code>	Set automatically, see Secrets Manager	<code>masteruser</code> or manually set	The user name of the MySQL user to authenticate as.
-p	<code>\$DBPASS</code>	Set automatically, see Secrets Manager	Manually set	The password of the MySQL user to authenticate as.
	[database]	<code>mylab</code>	<code>mylab</code> or manually set	The schema (database) to use by default.

Drop the `sbtest1` table:

Note

Consider executing the commands below one at a time, waiting a few seconds between each one. This will make it easier to determine a good point in time for testing backtrack. In a real world situation, you will not always have a clean marker to determine when the unintended change was made. Thus you might need to backtrack a few times to find the right point in time.

```
SELECT current_timestamp();
```

```
DROP TABLE sbtest1;
```

```
SELECT current_timestamp();
```

```
quit;
```

Remember or save the time markers displayed above, you will use them as references later, to simplify determining the right point in time to backtrack to, for demonstration purposes.

```
mysql> SELECT current_timestamp();
+-----+
| current_timestamp() |
+-----+
| 2019-07-02 05:26:47 |
+-----+
1 row in set (0.01 sec)

mysql> DROP TABLE sbtest1;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT current_timestamp();
+-----+
| current_timestamp() |
+-----+
| 2019-07-02 05:27:00 |
+-----+
1 row in set (0.00 sec)

mysql> quit;
Bye
ubuntu@ip-172-31-0-45:~$
```

Now, run the following command to replace the dropped table using the sysbench command, from your EC2-based workstation command line:

```
sysbench oltp_write_only \
--threads=1 \
--mysql-host=[clusterEndpoint] \
--mysql-user=$DBUSER \
--mysql-password="$DBPASS" \
--mysql-port=3306 \
--tables=1 \
--mysql-db=mylab \
--table-size=1000000 prepare
```

Command parameter values at a glance:

--threads	1	1	Number of concurrent threads.
-----------	---	---	-------------------------------

<code>--mysql-host</code>	<code>[clusterEndpoint]</code>	See CloudFormation stack output	See previous labs	The cluster endpoint of the Aurora DB cluster.
<code>--mysql-user</code>	<code>\$DBUSER</code>	Set automatically, see Secrets Manager	<code>masteruser</code> or manually set	The user name of the MySQL user to authenticate as.
<code>--mysql-password</code>	<code>\$DBPASS</code>	Set automatically, see Secrets Manager	Manually set	The password of the MySQL user to authenticate as.
<code>--mysql-port</code>		3306	3306	The port the Aurora database engine is listening on.
<code>--tables</code>		1	1	Number of tables to create.
<code>--mysql-db</code>		<code>mylab</code>	<code>mylab</code> or manually set	The schema (database) to use by default.
<code>--table-size</code>		1000000	1000000	The number of rows to generate in the table.

```
ubuntu@ip-172-31-0-45:~$ sysbench oltp_write_only --threads=1 --mysql-host=labstack-cluster.us-west-2.rds.amazonaws.com --mysql-user=$DBUSER --mysql-password=$DBPASS --mysql-port=3306 --tables=1 --mysql-db=mylab --table-size=1000000 prepare
sysbench 1.0.17 (using bundled LuaJIT 2.1.0-beta2)

Creating table 'sbtest1'...
Inserting 1000000 records into 'sbtest1'
Creating a secondary index on 'sbtest1'...
```

Reconnect to the DB cluster, and run the checksum table operation, the checksum value should be different than the source cluster value calculated in the [Cloning Clusters](#) lab:

```
mysql -h [clusterEndpoint] -u$DBUSER -p"$DBPASS" mylab
```



```
checksum table sbtest1;
```

```
quit;
```

2. Backtracking to recover from unintended changes

Backtrack the database to a time slightly after the second time marker. (Right after dropping the table).

Note

Backtrack operations occur at the DB cluster level, the entire database state is rolled back to a designated point in time, even though the example in this lab illustrates the effects of the operation on an individual table.

```
aws rds backtrack-db-cluster \  
--db-cluster-identifier labstack-cluster \  
--backtrack-to "yyyy-mm-ddThh:mm:ssZ"
```

```
ubuntu@ip-172-31-0-45:~$ aws rds backtrack-db-cluster --db-cluster-identifier labstack-cluster --backtrack-to "2019-07-0  
2T05:27:00Z"  
{  
  "DBClusterIdentifier": "labstack-cluster",  
  "BacktrackIdentifier": "2b9403c3-5acf-4443-a7f0-b707980b968c",  
  "BacktrackTo": "2019-07-02T05:27:00Z",  
  "BacktrackRequestCreationTime": "2019-07-02T05:38:15.986Z",  
  "Status": "PENDING"  
}
```

Run the below command to track the progress of the backtracking operation. The operation should complete in a few minutes.

```
aws rds describe-db-clusters \  
--db-cluster-identifier labstack-cluster \  
| jq -r '.DBClusters[0].Status'
```

```
ubuntu@ip-172-31-0-61:~$ aws rds describe-db-clusters --db-cluster-identifier labstack-cluster | jq -r '.DBClusters[0].Status'  
available  
ubuntu@ip-172-31-0-61:~$
```



Connect back to the database. The `sbtest1` table should be missing from the database.

```
mysql -h [clusterEndpoint] -u$DBUSER -p"$DBPASS" mylab
```

```
show tables;
```

```
quit;
```

```
ubuntu@ip-172-31-0-45:~$ mysql -h labstack-cluster-...us-west-2.rds.amazonaws.com -u$DBUSER -p$DBPASS mylab
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.6.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
Empty set (0.00 sec)

mysql>
```

Now backtrack again to a time slightly before the first time marker above. (Right before dropping the table).

```
aws rds backtrack-db-cluster \
--db-cluster-identifier labstack-cluster \
--backtrack-to "yyyy-mm-ddThh:mm:ssZ"
```

Track the progress of the backtracking operation, using the command below. The operation should complete in a few minutes.

```
aws rds describe-db-clusters \
--db-cluster-identifier labstack-cluster \
| jq -r '.DBClusters[0].Status'
```

Connect back to the database again. The `sbtest1` table should now be available in the database again, but contain the original data set.

```
mysql -h [clusterEndpoint] -u$DBUSER -p"$DBPASS" mylab
```

```
show tables;
```

```
checksum table sbtest1;
```

```
quit;
```

Using Performance Insights

This lab will demonstrate the use of [Amazon RDS Performance Insights](#). Amazon RDS Performance Insights monitors your Amazon RDS DB instance load so that you can analyze and troubleshoot your database performance.

This lab contains the following tasks:

1. Generating load on your DB cluster
2. Understanding the Performance Insights interface
3. Examining the performance of your DB instance

This lab requires the following lab modules to be completed first:

- [Prerequisites](#)
- [Creating a New Aurora Cluster](#) (conditional, if creating a cluster manually)
- [Connecting, Loading Data and Auto Scaling](#) (connectivity section only)

1. Generating load on your DB cluster

You will use Percona's TPCC-like benchmark script based on sysbench to generate load. For simplicity we have packaged the correct set of commands in an [AWS Systems Manager Command Document](#). You will use [AWS Systems Manager Run Command](#) to execute the test.

On the Session Manager workstation command line [see the Connecting, Loading Data and Auto Scaling lab](#), enter one of the following commands.

If you have completed the [Creating a New Aurora Cluster](#) lab, and created the Aurora DB cluster manually execute this command:

```
aws ssm send-command \  
--document-name [loadTestRunDoc] \  

```

```
--instance-ids [bastionInstance] \
--parameters \
clusterEndpoint=[clusterEndpoint],\
dbUser=$DBUSER,\
dbPassword="$DBPASS"
```

If AWS CloudFormation has provisioned the DB cluster on your behalf, and you skipped the **Creating a New Aurora Cluster** lab, you can run this simplified command:

```
aws ssm send-command \
--document-name [loadTestRunDoc] \
--instance-ids [bastionInstance]
```

Command parameter values at a glance:

--	[loadTestRunDoc]	See	See	The name
document-		CloudFormation	CloudFormation	of the
name		stack output	stack output	command
				document
				to run on
				your behalf.
--instance-	[bastionInstance]	See	See	The EC2
ids		CloudFormation	CloudFormation	instance to
		stack output	stack output	execute this
				command
				on.

-- clusterEndpoint=[clusterEndpoint],dbUser=\$DBUSER,dbPassword="\$DBPASS" N/A
parameters

Substitute the DB cluster endpoint with the values configured manually Additional command parameters.

The command will be sent to the workstation EC2 instance which will prepare the test data set and run the load test. It may take up to a minute for CloudWatch to reflect the additional load in the metrics. You will see a confirmation that the command has been initiated.

```

ubuntu@ip-172-31-0-55: $ aws ssm send-command --document-name labstack-ssmDocSysbenchTest --instance-ids i-
--parameters clusterEndpoint=labstack-cluster.cluster-...us-east-2.rds.amazonaws.com,dbUser=masteruser,dbPassword="
{
  "Command": {
    "CommandId": "...",
    "DocumentName": "labstack-ssmDocSysbenchTest",
    "DocumentVersion": "",
    "Comment": "",
    "ExpiresAfter": 1565121882.264,
    "Parameters": {
      "clusterEndpoint": [
        "labstack-cluster.cluster-...us-east-2.rds.amazonaws.com"
      ],
      "dbPassword": [
        "..."
      ],
      "dbUser": [
        "masteruser"
      ]
    },
    "InstanceIds": [
      "..."
    ],
    "Targets": [],
    "RequestedDateTime": 1565114682.264,
    "Status": "Pending",
    "StatusDetails": "Pending",
    "OutputS3BucketName": "",
    "OutputS3KeyPrefix": "",
    "MaxConcurrency": "50",
    "MaxErrors": "0",
    "TargetCount": 1,
    "CompletedCount": 0,
    "ErrorCount": 0,
    "DeliveryTimedOutCount": 0,
    "ServiceRole": "",
    "NotificationConfig": {
      "NotificationArn": "",
      "NotificationEvents": [],
      "NotificationType": ""
    },
    "CloudWatchOutputConfig": {
      "CloudWatchLogGroupName": "",
      "CloudWatchOutputEnabled": false
    }
  }
}
ubuntu@ip-172-31-0-55: $

```

2. Understanding the Performance Insights interface

While the command is running, open the [Amazon RDS service console](#) in a new tab, if not already open.

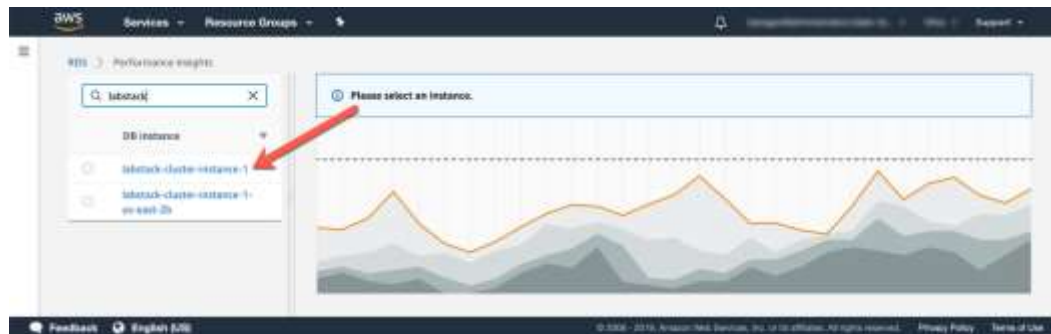
Region Check

Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.

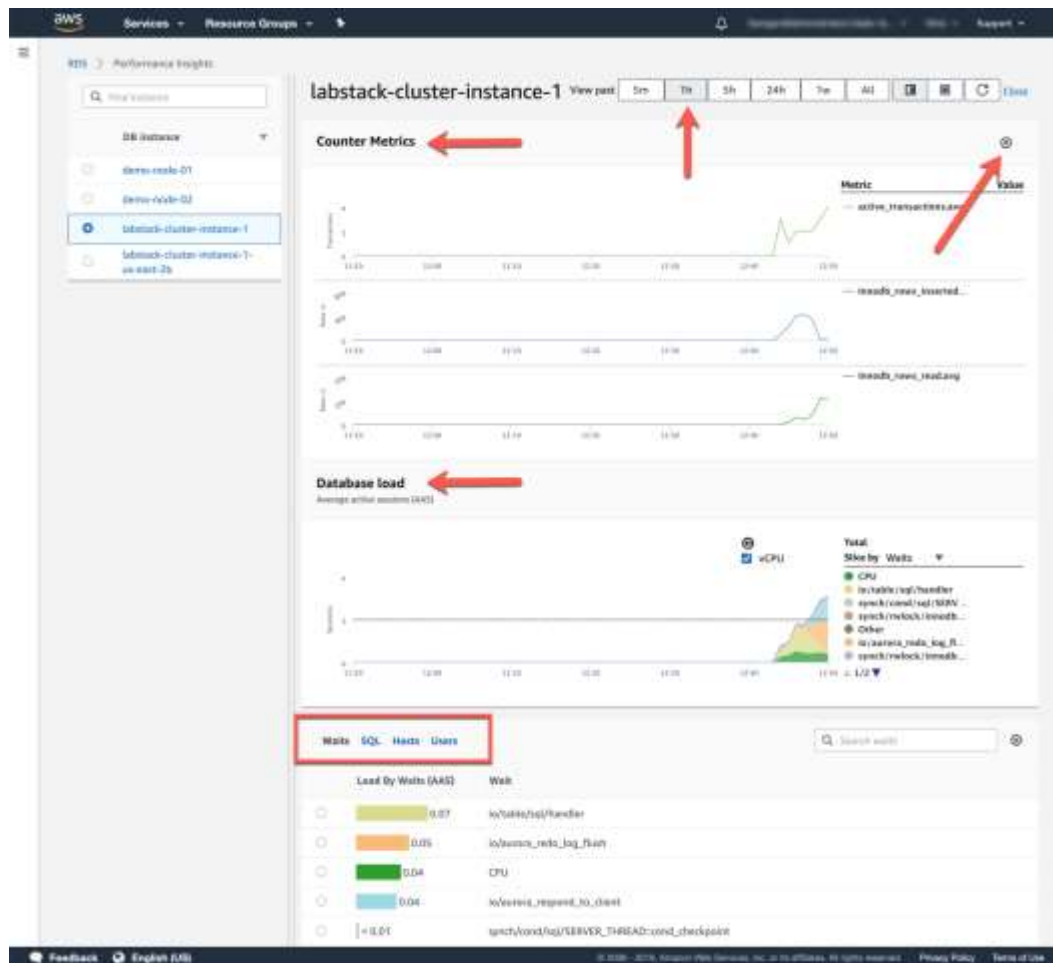
In the menu on the left hand side, click on the **Performance Insights** menu option.



Next, select the desired **DB instance** to load the performance metrics for. For Aurora DB clusters, performance metrics are exposed on an individual DB instance basis. As the different Db instances comprising a cluster may run different workload patterns, and might not all have Performance Insights enabled. For this lab we are generating load on the **Writer** (master) DB instance only. Select the DB instance where the name either ends in `-node-01` or `-instance-1`



Once a DB instance is selected, you will see the main dashboard view of RDS Performance Insights. The dashboard is divided into 3 sections, allowing you to drill down from high level performance indicator metrics down to individual queries, waits, users and hosts generating the load.



The performance metrics displayed by the dashboard are a moving time window. You can adjust the size of the time window by clicking the buttons across the top right of the interface (5m, 1h, 5h, 24h, 1w, all). You can also zoom into a specific period of time by dragging across the graphs.

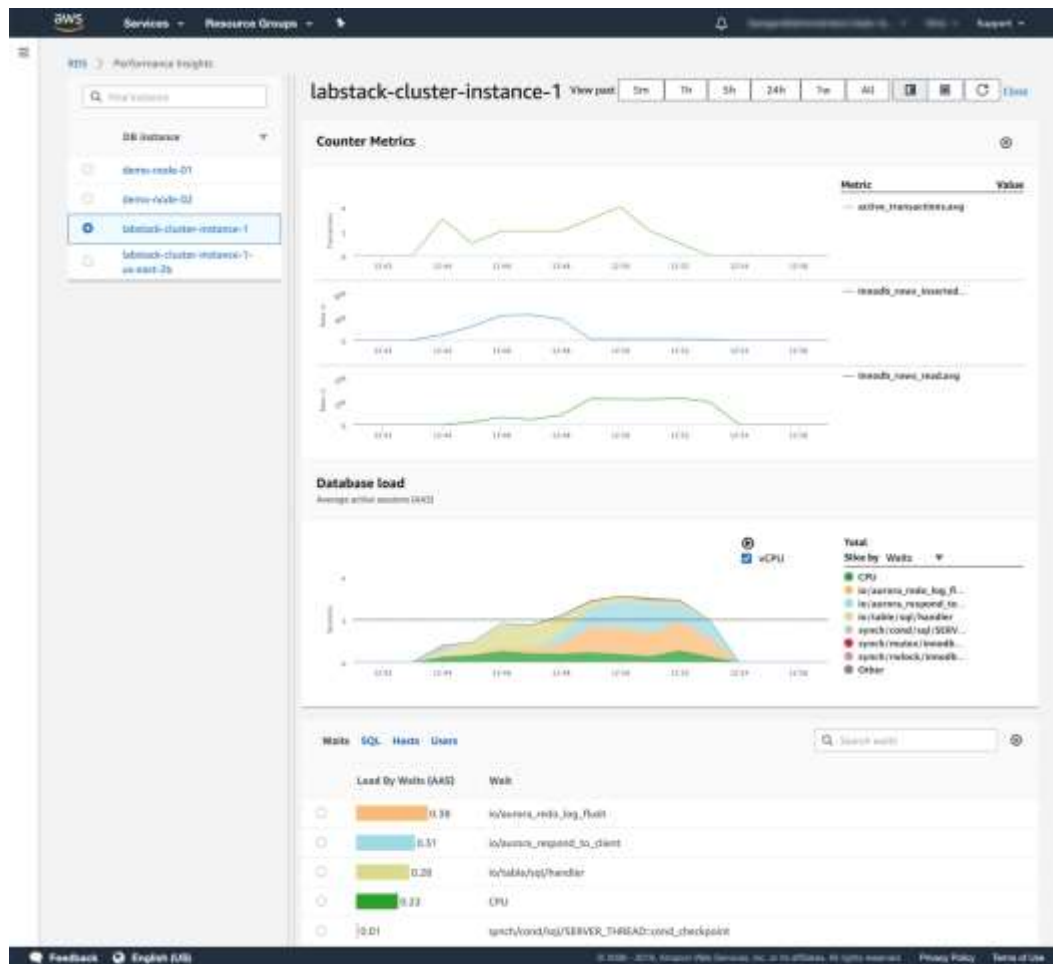
Note

All dashboard views are time synchronized. Zooming in will adjust all views, including the detailed drill-down section at the bottom.

Counter Metrics	Click cog icon in top right corner to select additional counters	This section plots internal database counter metrics over time, such as number of rows read or written, buffer pool hit ratio, etc. These counters are useful to correlate with other metrics, including the database load metrics, to identify causes of abnormal behavior.
Database load	Load can be sliced by waits (default), SQL commands, users and hosts	This metric is design to correlate aggregate load (sliced by the selected dimension) with the available compute capacity on that DB instance (number of vCPUs). Load is aggregated and normalized using the Average Active Session (AAS) metric. A number of AAS that exceeds the compute capacity of the DB instance is a leading indicator of performance problems.
Granular Session Activity	Sort by Waits , SQL (default), Users and Hosts	Drill down capability that allows you to get detailed performance data down to the individual commands.

3. Examining the performance of your DB instance

After running the load generator workload above, you will see a performance profile similar to the example below in the Performance Insights dashboard. The load generator command will first create an initial data set using `sysbench prepare`. And then will run an OLTP workload for the duration of 5 minutes, consisting of concurrent transactional reads and writes using 4 parallel threads.



Amazon Aurora MySQL specific wait events are documented in the [Amazon Aurora MySQL Reference guide](#). Use the Performance Insights dashboard and the reference guide documentation to evaluate the workload profile of your load test, and answer the following questions:

1. Is the database server overloaded at any point during the load test?
2. Can you identify any resource bottlenecks during the load test? If so how can they be mitigated?

3. What are the most common wait events during the load test?
4. Why are the load patterns different between the first and second phase of the load test?

Creating a Serverless Aurora Cluster

This lab will walk you through the steps of creating an Amazon Aurora Serverless database cluster manually, and configuring the scaling parameters of the cluster.

This lab contains the following tasks:

1. Creating the serverless DB cluster
2. Creating a secret to store the credentials

This lab requires the following lab modules to be completed first:

- [Prerequisites](#) (using `lab-no-cluster.yml` template is sufficient)

1. Creating the serverless DB cluster

Open the [Amazon RDS service console](#).

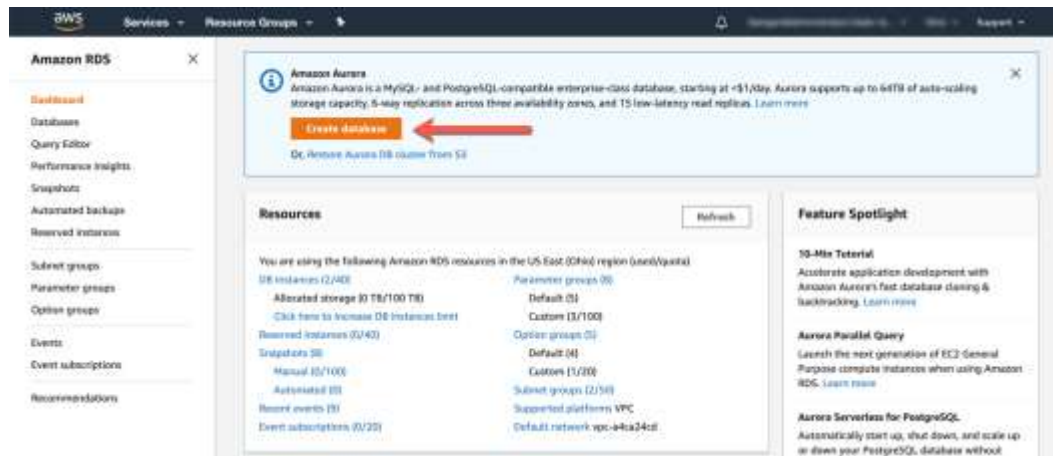
Region Check

Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.

Click **Create database** to start the configuration process

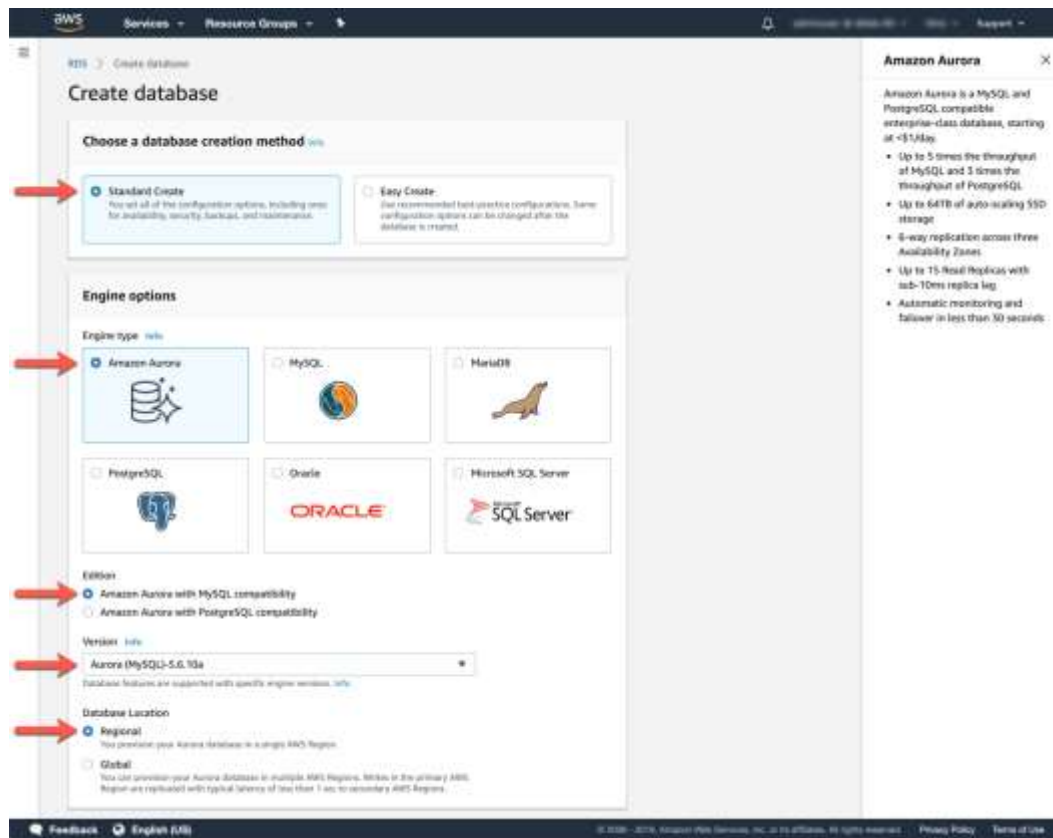
Note

The RDS console database creation workflow has been simplified recently. Depending on your previous usage of the RDS console UI, you may see the old workflow or the new one, you may also be presented with a prompt to toggle between them. In this lab we are using the new workflow for reference, although the steps will work similarly in the old console workflow as well, if you are more familiar with it.



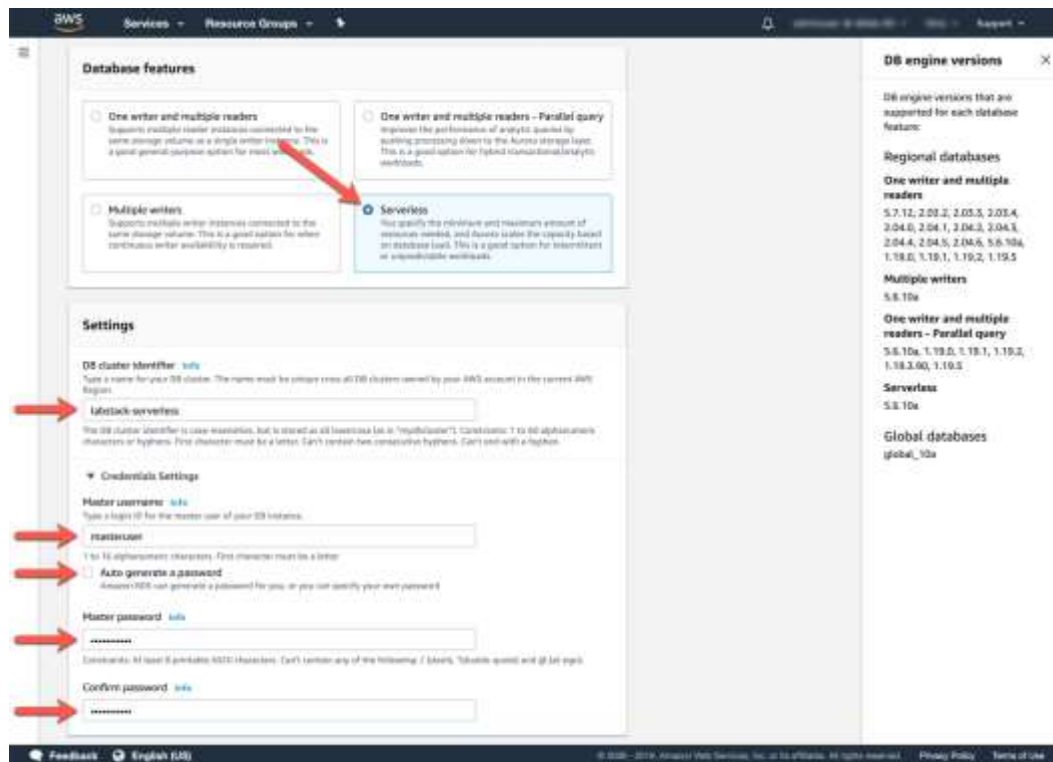
In the first configuration section of the **Create database** page, ensure the **Standard Create** database creation method is selected.

Next, in the **Engine options** section, choose the **Amazon Aurora** engine type, the **Amazon Aurora with MySQL compatibility edition**, the Aurora (MySQL)-5.6.10a version and the Regional** database location.



In the **Database features** section, select **Serverless**. The selections so far will instruct AWS to create an Aurora MySQL database cluster with the most recent version of the MySQL 5.6 compatible engine in a serverless configuration.

In the **Settings** section set the database cluster identifier to `labstack-serverless`. Configure the name and password of the master database user, with the most elevated permissions in the database. We recommend to use the user name `masteruser` for consistency with subsequent labs and a password of your choosing. For simplicity ensure the check box **Auto generate a password** is **not checked**.

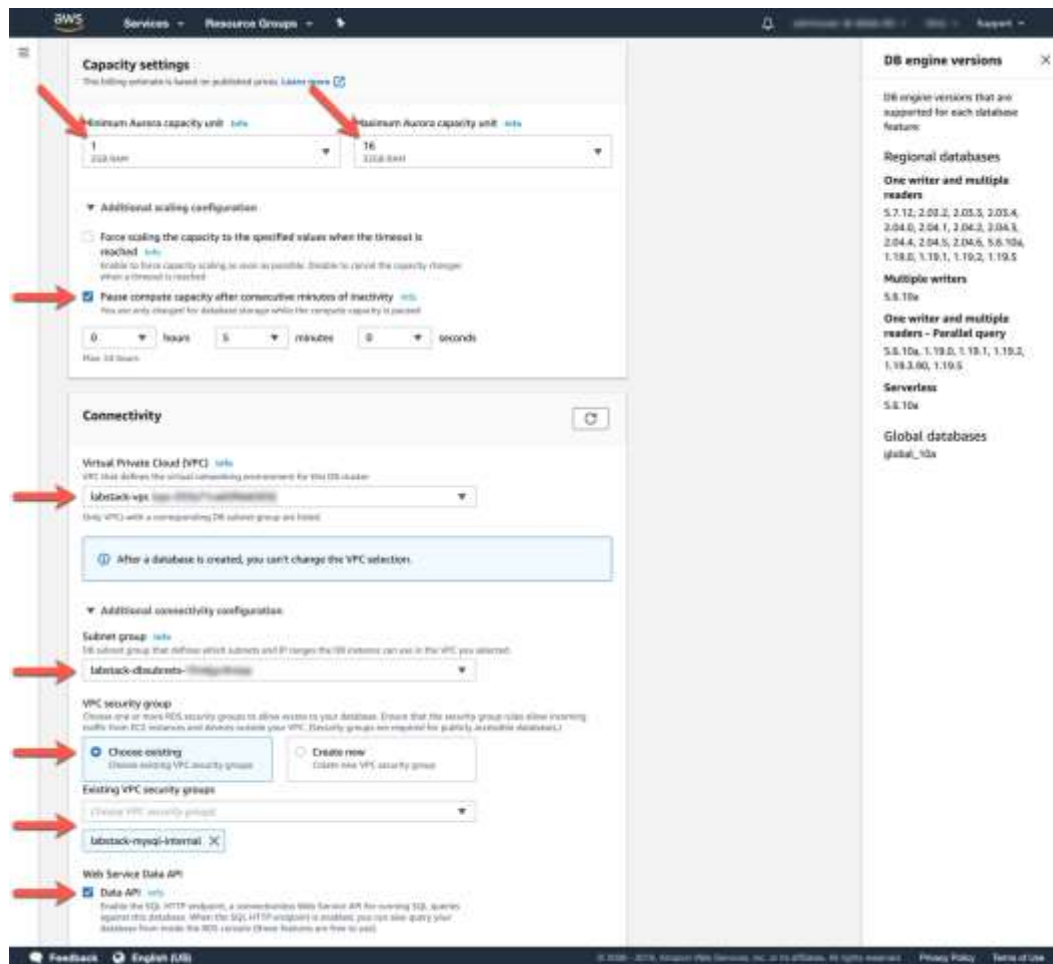


In the **Capacity settings** section, choose a **Minimum Aurora capacity unit** of 1 (2GB RAM) and a **Maximum Aurora capacity unit** of 16 (32 GB RAM). Next, expand the **Additional scaling configuration** section, and **check** the box next to **Pause compute capacity after consecutive minutes of inactivity**. This configuration will allow Aurora Serverless to scale the capacity of your DB cluster between 1 capacity unit and 32 capacity units, and to suspend capacity entirely after 5 minutes of inactivity.

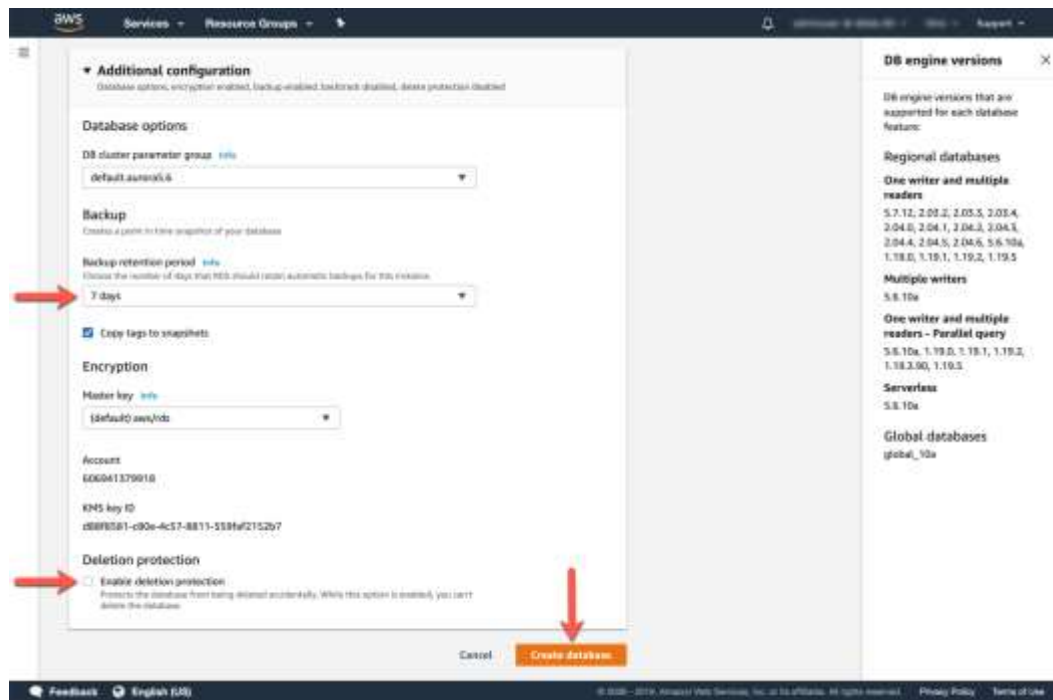
In the **Connectivity** section, expand the sub-section called **Additional connectivity configuration**. This section allows you to specify where the database cluster will be deployed within your defined network configuration. To simplify the labs, the CloudFormation stack you deployed in the preceding [Prerequisites](#) module, has configured a VPC that includes all resources needed for an Aurora database cluster. This includes the VPC itself, subnets, DB subnet groups, security groups and several other networking constructs. All you need to do is select the appropriate existing connectivity controls in this section.

Pick the **Virtual Private Cloud (VPC)** named after the CloudFormation stack name, such as `labstack-vpc`. Similarly make sure the selected **Subnet Group** also matches the stack name (e.g. `labstack-dbsubnets-[hash]`). The lab environment also configured a **VPC security group** that allows your lab workspace EC2 instance to connect to the database. Make sure the **Choose existing** security group option is selected and from the dropdown pick the security group with a name ending in `-mysql-internal` (eg. `labstack-mysql-internal`). Please remove any other security groups, such as `default` from the selection.

Additionally, please check the box **Data API**, to enable integration with the RDS Data API.



Next, expand the **Advanced configuration** section. Choose a 7 days **Backup retention period**. **De-select** the check box **Enable delete protection**. In a production use case, you will want to leave that option checked, but for testing purposes, un-checking this option will make it easier to clean up the resources once you have completed the labs.

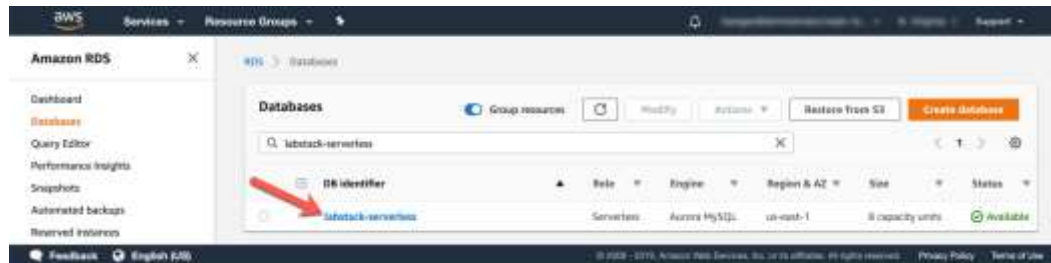


Before continuing, let's summarize the configuration options selected. You will create a database cluster with the following characteristics:

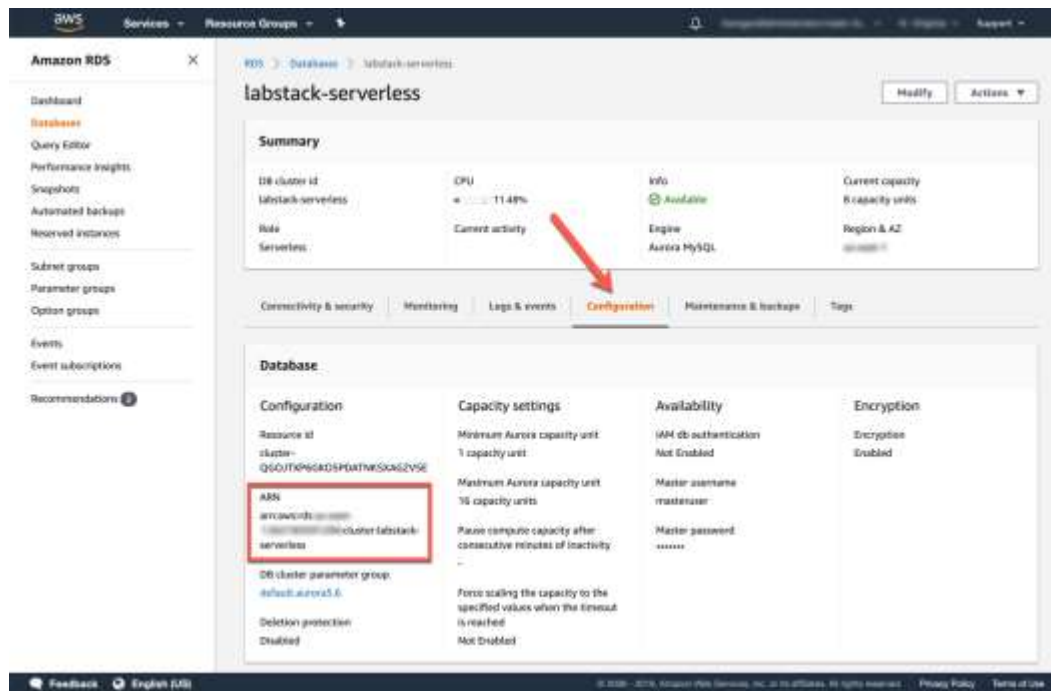
- Aurora MySQL 5.6 compatible (latest stable engine version)
- Serverless db cluster scaling between 1 and 16 capacity units, and pausing compute capacity after 5 minutes of inactivity
- Deployed in the VPC and using the network configuration of the lab environment
- Integrated with the RDS Data API
- Automatically backed up continuously, retaining backups for 7 days
- Using data at rest encryption

Click **Create database** to provision the DB cluster.

Once back to the list of databases, click the name of the new database in the listing.



In the details view of the cluster, click on the **Configuration** tab. Note the value for **ARN**. Write this down, you will need it later.



2. Creating a secret to store the credentials

Open the [AWS Secrets Manager service console](#).

Region Check

Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.

Click **Store a new secret** to start the configuration process.



In the **Select secret type** section, choose **Credentials for RDS database**, then input the **User name** (e.g. `masteruser`) and **Password** that you provided when you created the serverless DB cluster.

Next, in the **Select which RDS database this secret will access** section, choose the DB cluster identifier you assigned to your cluster (e.g. `labstack-serverless`). Click **Next**.

AWS Services - Resource Groups - AWS Secrets Manager - Secrets - Store a new secret

Store a new secret

Step 1: Secret type

Step 2: Name and description

Step 3: Configure rotation

Step 4: Review

Select secret type

☒ Credentials for RDS database

☐ Credentials for Redis cluster

☐ Credentials for DocumentDB database

☐ Credentials for other database

☐ Other type of secret (exp. API key)

Specify the user name and password to be stored in this secret

User name:

Password:

☐ Show password

Select the encryption key

Select the AWS KMS key to use to encrypt your secret information. You can encrypt using the default service encryption key that AWS Secrets Manager creates on your behalf or a customer-managed key (CMK) that you have created in AWS IAM.

[Add new key](#)

Select which RDS database this secret will access

DB instance	DB engine	Status	Creation date
<input type="radio"/> arn56-ord-cluster	aurora	available	3/26/19
<input type="radio"/> labstack-cluster	aurora	available	6/7/19
<input type="radio"/> labstack-cluster-clone	aurora	available	6/8/19
<input checked="" type="radio"/> labstack-serverless	aurora	available	6/8/19

[Cancel](#) [Next](#)

Name the secret `labstack-serverless-secret` and provide a relevant description for the secret, then click **Next**.

Step 1: Secret type

Step 2: Name and description

Step 3: Configure rotation

Step 4: Review

Store a new secret

Secret name and description [Info](#)

Secret name

Use the secret's name to create policies you use to find and manage it easily.

Secret names must contain only alphanumeric characters and the characters `/`, `=`, and `@`.

Description - optional

Master credential for the lobstack-serverless DB cluster

Maximum 255 characters

Tags - optional

Key

Value - optional

Enter key

Enter value

Remove

Add

Cancel Previous **Next**

Finally, in the **Configure automatic rotation** section, leave the option of **Disable automatic rotation** selected. In a production environment you will want to use database credentials that rotate automatically for additional security. Click **Next**.

The screenshot shows the AWS Secrets Manager console in the 'Store a new secret' step. The left sidebar lists steps: Step 1: Secret type, Step 2: Name and description, Step 3: Configure rotation (active), Step 4: Review. The main content area has a title 'Store a new secret' and a warning box about automatic rotation. The 'Configure automatic rotation' section has two radio buttons: 'Disable automatic rotation' (selected) and 'Enable automatic rotation'. Below this is a 'Select rotation interval' dropdown set to '30 days'. Further down is a 'New AWS Lambda function name' field with 'SecretsManager' entered. At the bottom right, a red arrow points to the 'Store' button.

In the **Review** section you have the ability to check the configuration parameters for your secret, before it gets created. Additionally, you can retrieve sample code in popular programming languages, so you can easily retrieve secrets into your application. Click **Store** at the bottom of the screen.

Once created, identify the **ARN** of the newly created secret. This value will be needed in subsequent labs. In the list of **Secrets** in the console, click on the name of the newly created secret.



In the detail view of the secret, note the value for **Secret ARN**. Write this down, you will need it later.



Using Aurora Serverless with Lambda Functions

This lab will show you how to connect to and interact with Amazon Aurora Serverless database clusters using AWS Lambda functions and the RDS Data API.

This lab contains the following tasks:

1. Creating a Lambda execution role
2. Creating a Lambda function
3. Connecting to the database using the RDS Data API

This lab requires the following lab modules to be completed first:

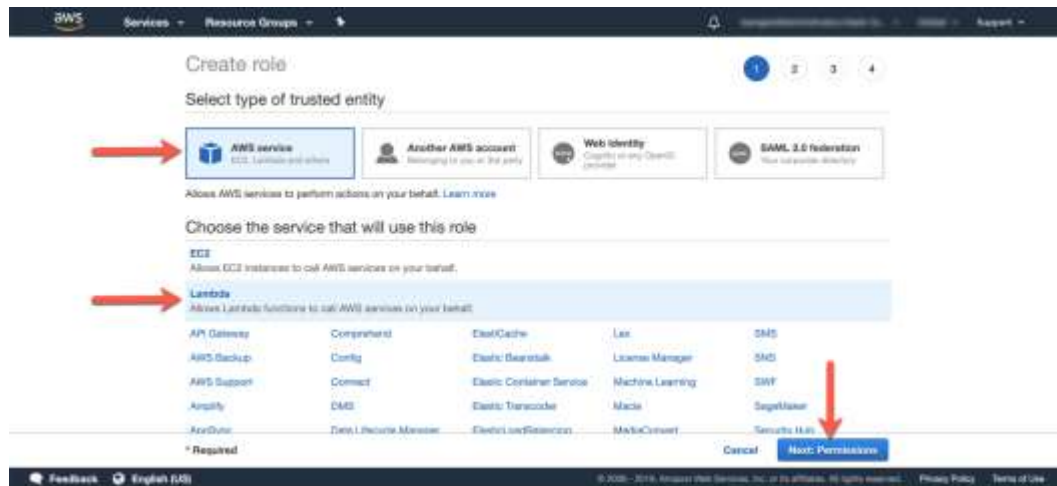
- [Prerequisites](#) (using `lab-no-cluster.yml` template is sufficient)
- [Creating a Serverless Aurora Cluster](#)

1. Creating a Lambda execution role

Before you create an AWS Lambda function, you need to configure an IAM execution role. This will contain the permissions you are granting the function to interact with AWS resources via the APIs. Open the [Identity and Access Management \(IAM\) service console](#). Choose **Roles** from the left hand side menu, if it isn't already selected, and click **Create role**.



In the **Select type of trusted entity** section, choose **AWS service**. Next, in the **Choose the service that will use this role** section, choose **Lambda**, then click **Next: Permissions**.



Click the **Create policy** button in the **Attach permissions policies** section.



In the new browser tab that opens up, toggle to the **JSON** interface tab. Ignore any message that may be displayed warning that the policy validation failed - we have not created a policy yet. Paste the policy listed below in the text editor, and substitute the [SecretARN] placeholder with the ARN of the secret you created in the previous lab. Click **Review policy**.

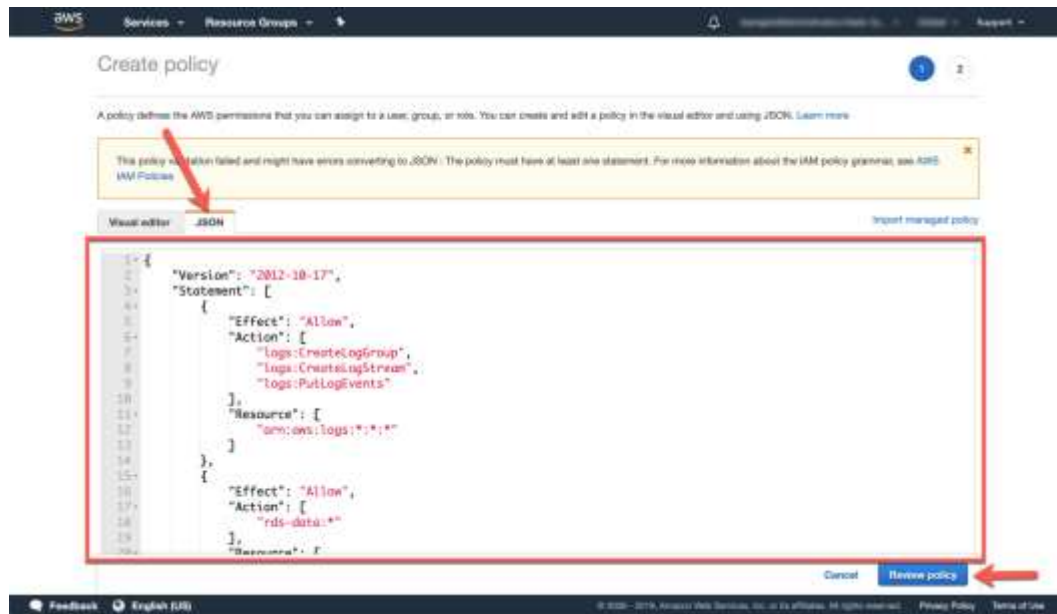
```
{
  "Version": "2012-10-17",
  "Statement": [
    {

```

```

    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:*:*:*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "rds-data:*"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager:GetSecretValue"
    ],
    "Resource": [
        "[SecretARN]"
    ]
}
]
}

```



Assign the IAM policy the name `LabstackFunctionPolicy`, then click **Create policy**.

Create policy

Review policy

Name* LabstackFunctionPolicy

Use alphanumeric and "_" characters. Maximum 128 characters.

Description

Maximum 1000 characters. Use alphanumeric and "_" characters.

Summary

This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose [Show remaining](#). [Learn more](#)

Service	Access level	Resource	Request condition
Allow (3 of 185 services) Show remaining 182			
CloudWatch Logs	Limited: Write	arn:aws:logs:*:*	None
EC2 Data API	Full access	All resources	None
Secrets Manager	Limited: Read	arn:aws:secretsmanager:*:*:*:*:secret:*-*	None

* Required

Cancel Previous **Create policy**

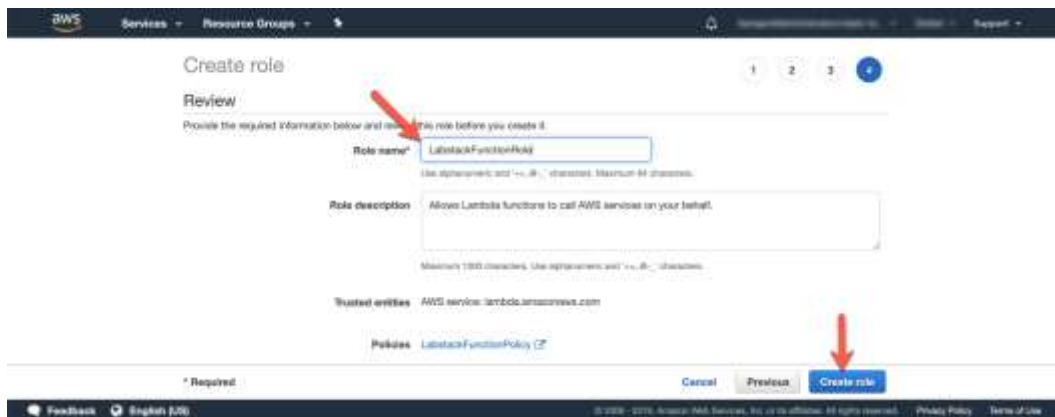
Once the policy has been created successfully, you can return to the other browser tab and continue configuring the role. You can also close the policy browser tab.



Back on the browser tab for creating the role, click the refresh icon in the top right of the policy list, then use the filter input field to search for the name of the policy you have just created (e.g. LabstackFunctionPolicy). Select that policy, and click **Next: tags**.



Skip the **Add tags** section, and click **Next: Review**. Then assign the role the name `LabstackFunctionRole`, and click **Create role**.



2. Creating a Lambda function

Open the [AWS Lambda service console](#).

Region Check

Ensure you are still working in the correct region, especially if you are following the links above to open the service console at the right screen.

Choose **Functions** from the left hand side menu, if it isn't already selected, and click **Create function**.



Choose the option to **Author from scratch**, set the **Function name** to `LabstackFunction` and select **Node.js 10.x** for **Runtime**. Under **Permissions**, expand the sub-section called **Choose or create an execution role**. In the **Execution role** dropdown, select **Use an existing role**, then in the **Existing role** dropdown, select the execution role you have created previously, named `LabstackFunctionRole`. Click **Create function**.

The screenshot shows the AWS Lambda 'Create function' page. At the top, there are three options: 'Author from scratch' (selected), 'Use a blueprint', and 'Browse serverless app repository'. Below these is the 'Basic information' section with the following fields:

- Function name:** LabstackFunction
- Runtime:** Node.js 10.x
- Permissions:** Choose or create an execution role
- Execution role:** Use an existing role
- Existing role:** LabstackFunctionRole

At the bottom right, there is a red arrow pointing to the 'Create function' button.

Make sure the **Configuration** tab is selected. In the **Function code** section, select **Edit code inline** for **Code entry type**, if not already selected, and leave the values for **Runtime** and **Handler** as default (`Node.js 10.x` and `index.handler` respectively). Paste the code snippet below into the editor, and change the placeholders as follows:

[ClusterARN] The ARN of the serverless database cluster resource.
RDS Data API will establish connectivity with this database on your behalf.

See the previous lab: [Creating a Serverless Aurora Cluster](#) at step 1. *Creating the serverless DB cluster.*

[SecretARN] The ARN of the secret used to store the database credentials. RDS Data API will access this secret and connect to the database using those credentials.

See the previous lab: [Creating a Serverless Aurora Cluster](#) at step 2. *Creating a secret to store the credentials.*

```
// require the AWS SDK
const AWS = require('aws-sdk')
const rdsDataService = new AWS.RDSDataService()

exports.handler = (event, context, callback) => {
  // prepare SQL command
  let sqlParams = {
    secretArn: '[SecretARN]',
    resourceArn: '[ClusterARN]',
    sql: 'SHOW TABLES;',
    database: 'information_schema',
    includeResultMetadata: true
  }

  // run SQL command
  rdsDataService.executeStatement(sqlParams, function (err, data) {
    if (err) {
      // error
      console.log(err)
      callback('Query Failed')
    } else {
      // init
      var rows = []
      var cols = []

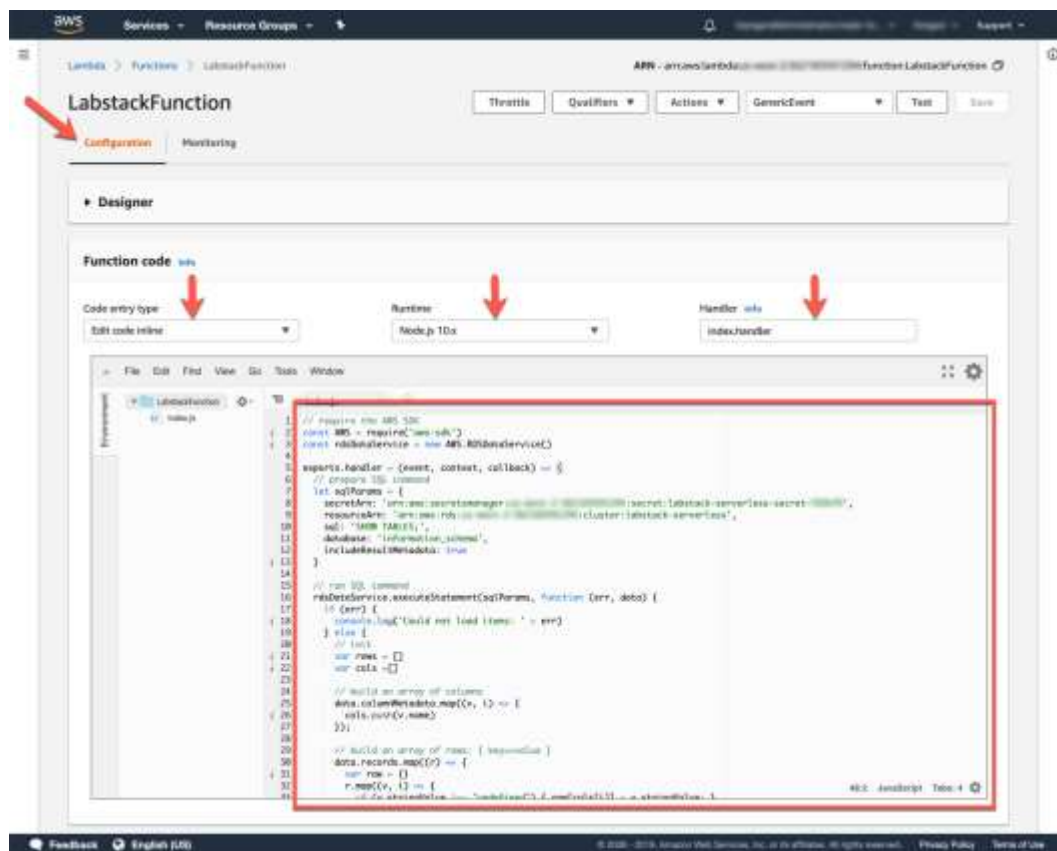
      // build an array of columns
      data.columnMetadata.map((v, i) => {
        cols.push(v.name)
      });

      // build an array of rows: { key=>value }
      data.records.map((r) => {
        var row = {}
        r.map((v, i) => {
```

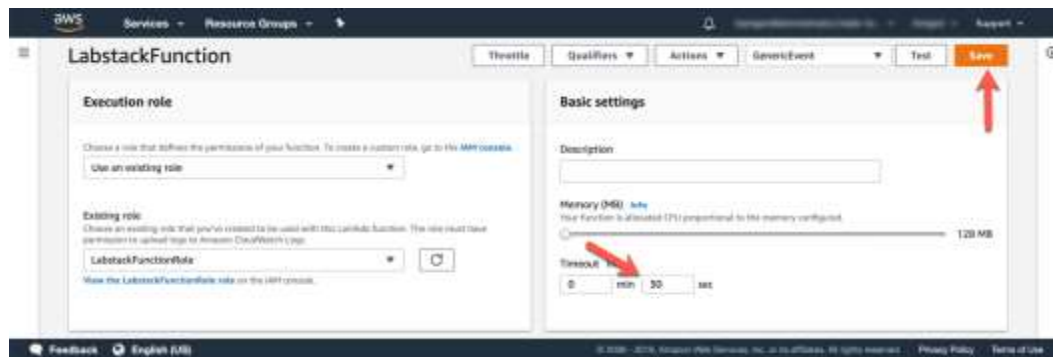


```
        if (v.stringValue !== "undefined") { row[cols[i]] = v.stringValue; }
        else if (v.blobValue !== "undefined") { row[cols[i]] = v.blobValue; }
        else if (v.doubleValue !== "undefined") { row[cols[i]] = v.doubleValue; }
        else if (v.longValue !== "undefined") { row[cols[i]] = v.longValue; }
        else if (v.booleanValue !== "undefined") { row[cols[i]] = v.booleanValue; }
        else if (v.isNull) { row[cols[i]] = null; }
    })
    rows.push(row)
  })

  // done
  console.log('Found rows: ' + rows.length)
  callback(null, rows)
}
})
}
```



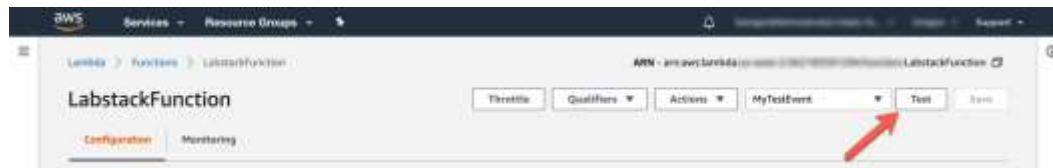
Once you have pasted and replaced the placeholders in the code, scroll down to the **Basic settings** section, and change the function **Timeout** to 30 sec. Click **Save** in the top right area of the console. We are increasing the timeout as it will take longer to respond to the first query against the serverless DB cluster. The compute capacity will be allocated only when the first request is received.



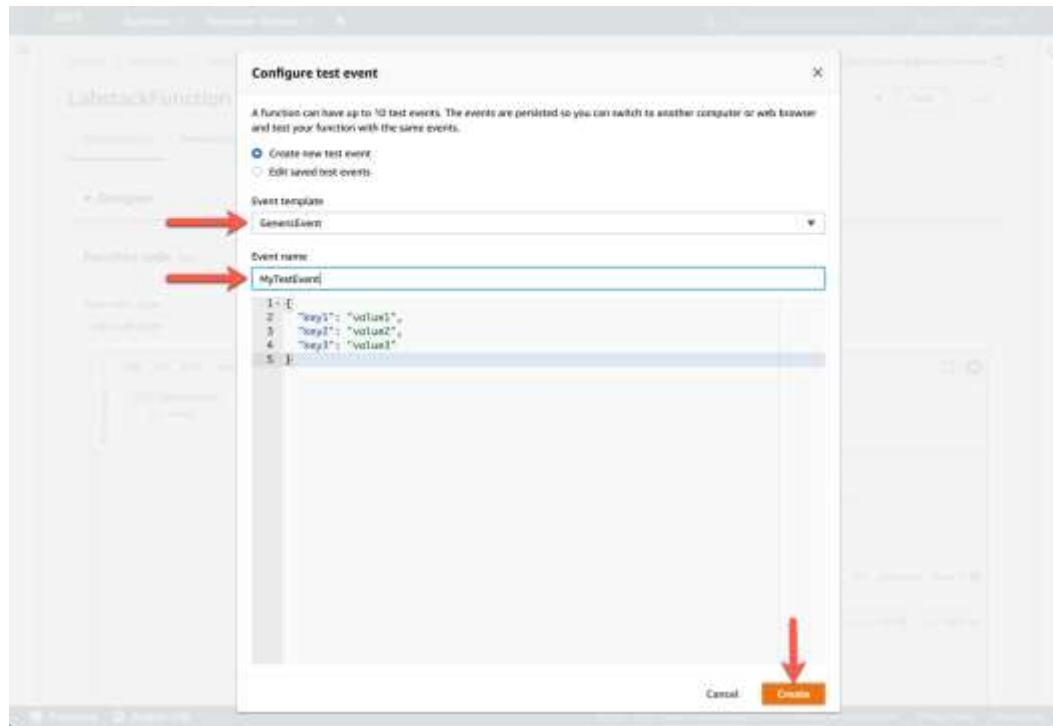
3. Connecting to the database using the RDS Data API

Now you are ready to connect to the database from a Lambda function, by using the RDS Data API. The function doesn't bundle a database driver, it simply uses a RESTful AWS API call to send a SQL Query, `SHOW TABLES;` in this example, retrieves the result as a JSON data structure. This is accomplished in a minimal number of lines of code.

Execute the function by clicking the **Test** button.



You will be asked to configure a test event the first time you try. The format and content for the test event are not relevant for this exercise, so pick any **Event template**, such as `Hello World`, input a memorable **Event name**, such as `MyTestEvent` and hit **Create**.



If you have created a new test event, you may need to click the **Test** button again. After the function has completed running you can review the results, and see the function response. You may need to expand the **Details** sub-section of the **Execution result: succeeded** notification to see the results.

ServicesResource Groups

ARN: arn:aws:lambda:us-east-1:4471-82c3-be5a82def06a:function:LabstackFunction

LabstackFunction

ThrottleQualifiersActionsMyTestEventTestSave

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
{
  "TABLE_NAME": "CHARACTER_SETS",
},
{
  "TABLE_NAME": "COLLATIONS",
},
{
  "TABLE_NAME": "COLLATION_CHARACTER_SET_APPLICABILITY",
},

```

Summary

Code SHA-256
+vfrWylouUZXUjH4u6u9VjyuLH6AR68VYH1IEP+

Request ID
1f8bb3d7-9a15-4471-82c3-be5a82def06a

Duration
1780.45 ms

Billed duration
1800 ms

Provisioned configured
128 MB

Max memory used
89 MB

Log output

The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: 1f8bb3d7-9a15-4471-82c3-be5a82def06a Version: $LATEST
2019-08-10T21:10:23.337Z    1f8bb3d7-9a15-4471-82c3-be5a82def06a    INFO    Found rows: 69
END RequestId: 1f8bb3d7-9a15-4471-82c3-be5a82def06a
REPORT RequestId: 1f8bb3d7-9a15-4471-82c3-be5a82def06a  Duration: 1780.45 ms    Billed Duration: 1800 ms    Memory Size: 128 MB
Max Memory Used: 89 MB
```

FeedbackEnglish (US)

© 2019 – 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy PolicyTerms of Use