CLOUD TRAIN

ACCELERATE YOUR GROWTH

MODULE 6
DATABASES & IN-MEMORY DATASTORES

AWS Workshop

Contact us

TO ACCELERATE YOUR CAREER GROWTH

For questions and more details:

please call @ +91 98712 72900, or

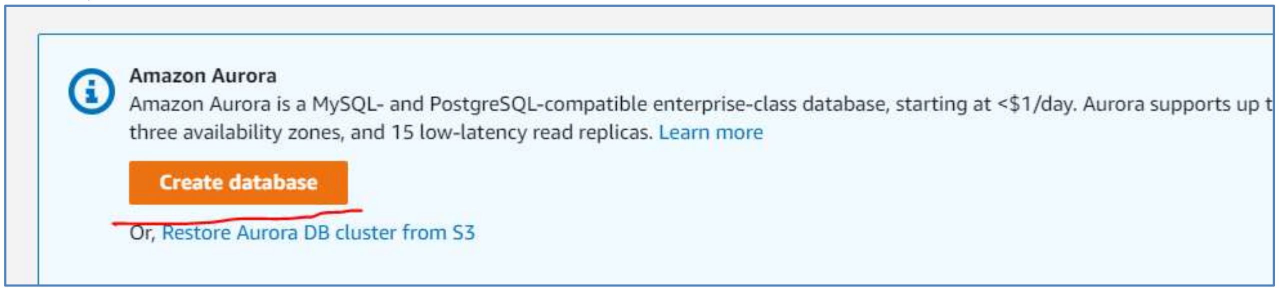
visit https://www.thecloudtrain.com/, or

email at support@thecloudtrain.com, or

WhatsApp us @ +91 98712 72900

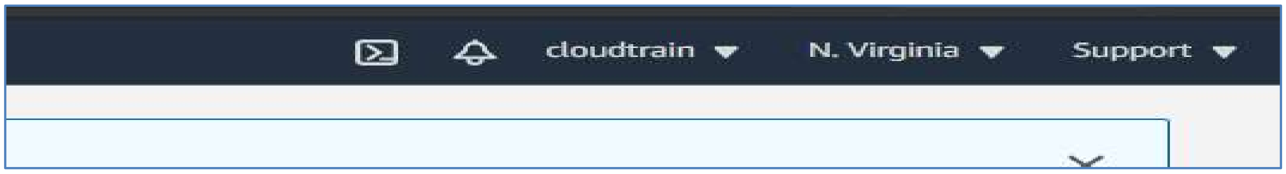
Creating an Amazon RDS instance

Step 1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

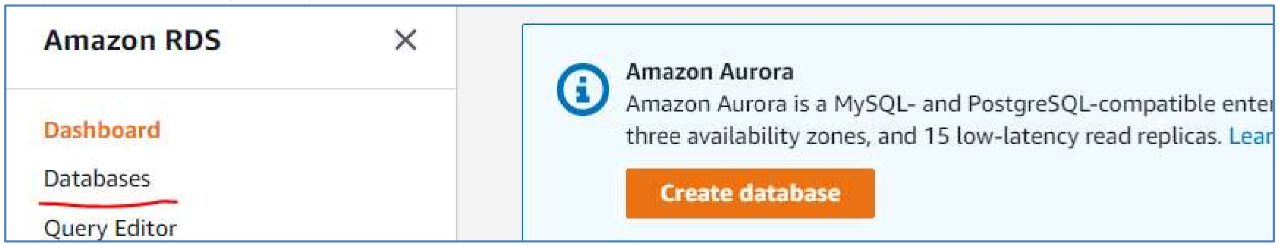


Amazon Aurora
Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <$1/day. Aurora supports up t
three availability zones, and 15 low-latency read replicas. Learn more

Step 2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB instance.

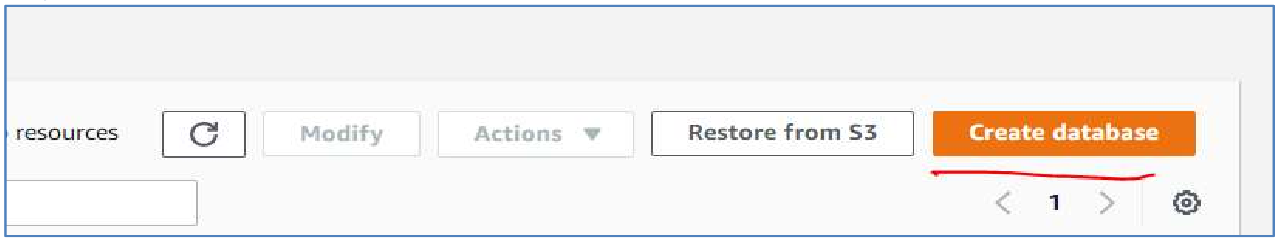


Step 3. In the navigation pane, choose **Databases**.



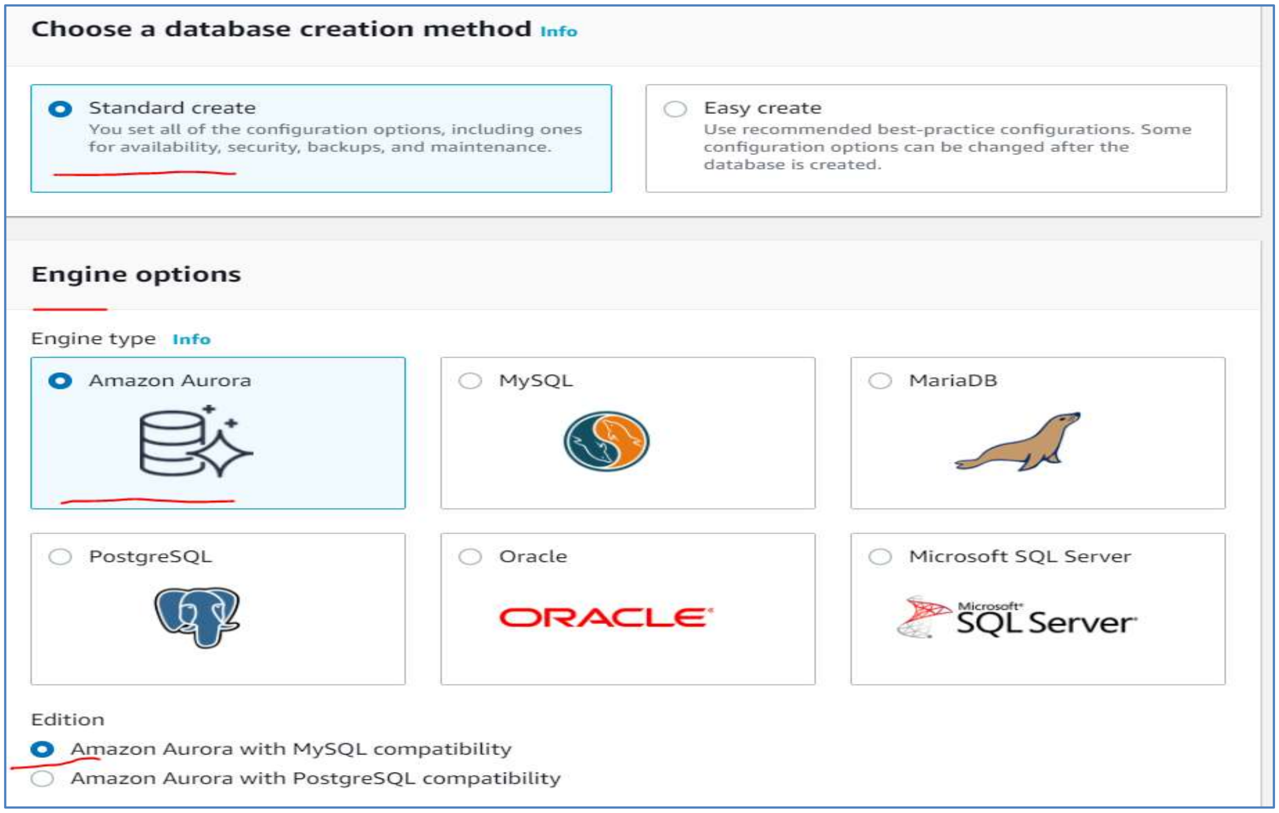
Amazon Aurora
Amazon Aurora is a MySQL- and PostgreSQL-compatible ente
three availability zones, and 15 low-latency read replicas. Lear

Step 4. Choose **Create database**.



Step 5. In **Choose a database creation method**, select **Standard Create**.

Step 6. In **Engine options**, choose the engine type: MariaDB, Microsoft SQL Server, MySQL, Oracle, or PostgreSQL. **Microsoft SQL Server** is shown here.



Choose a database creation method Info

- [✓ Standard create] You set all of the configuration options, including ones for availability, security, backups, and maintenance.
- Easy create: Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type Info

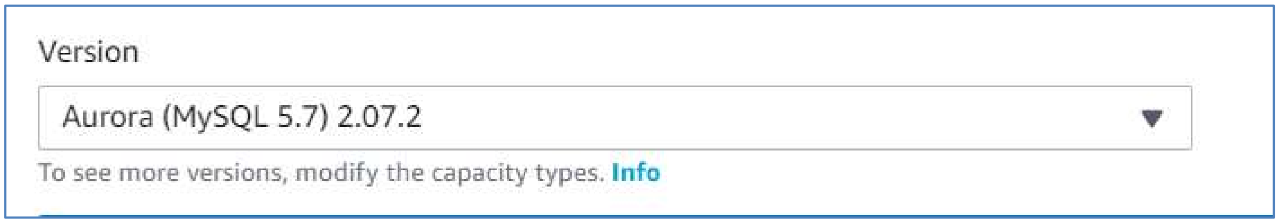
- [✓ Amazon Aurora]
- MySQL
- MariaDB
- PostgreSQL
- Oracle
- Microsoft SQL Server

Edition

- [✓ Amazon Aurora with MySQL compatibility]
- Amazon Aurora with PostgreSQL compatibility

Step 7. For **Edition**, if you're using Oracle or SQL Server choose the DB engine edition that you want to use.MySQL has only one option for the edition, and MariaDB and PostgreSQL have none.

Step 8. For **Version**, choose the engine version.



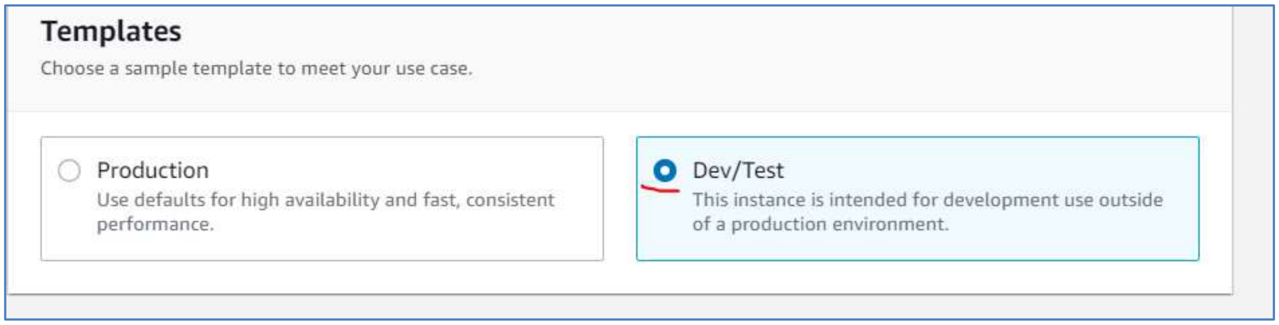
Version

Aurora (MySQL 5.7) 2.07.2

To see more versions, modify the capacity types. Info

Step 9. In **Templates**, choose the template that matches your use case. We use Dev here but if you choose Production, the following are preselected in a later step:

- **Multi-AZ** failover option
- **Provisioned IOPS** storage option
- **Enable deletion protection** option



Templates

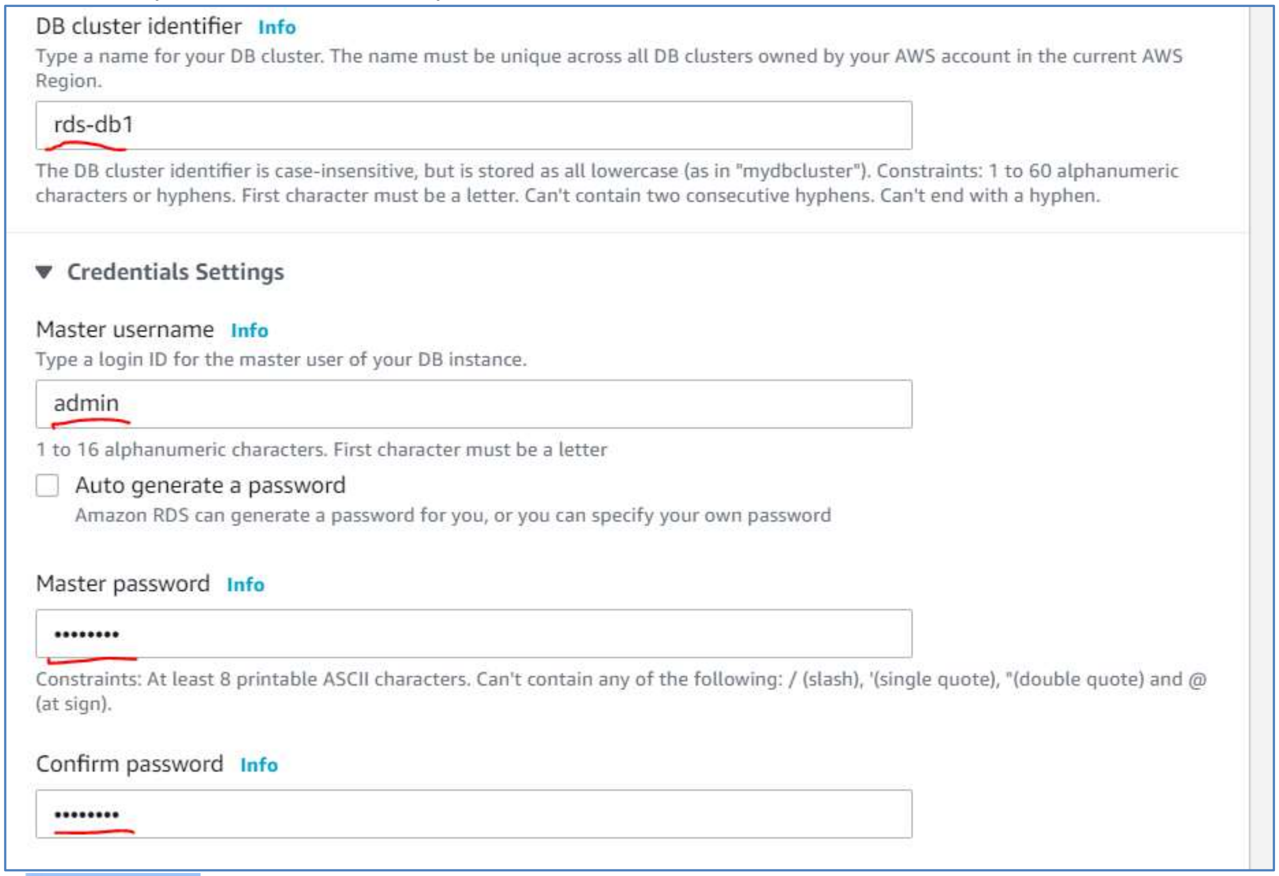
Choose a sample template to meet your use case.

- ○ Production — Use defaults for high availability and fast, consistent performance.
- ● Dev/Test — This instance is intended for development use outside of a production environment.

Step 10. Choose a name for your DB, say **rds-db1**:

Step 11. To enter your master password, do the following:

- In the **Settings** section, open **Credential Settings**.
- Clear the **Auto generate a password** check box.
- (Optional) Change the **Master username** value and enter the same password in **Master password** and **Confirm password**.



DB cluster identifier Info

Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

rds-db1

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username Info

Type a login ID for the master user of your DB instance.

admin

1 to 16 alphanumeric characters. First character must be a letter

☐ Auto generate a password

Amazon RDS can generate a password for you, or you can specify your own password

Master password Info

••••••••

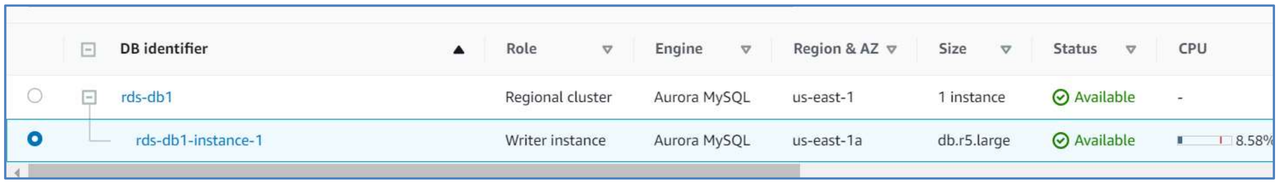
Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password Info

••••••••

Step 12. For the remaining sections, use default settings.

Step 13. Choose **Create database**.



	DB identifier	Role	Engine	Region & AZ	Size	Status	CPU
○	rds-db1	Regional cluster	Aurora MySQL	us-east-1	1 instance	Available	-
●	rds-db1-instance-1	Writer instance	Aurora MySQL	us-east-1a	db.r5.large	Available	8.58%

Step 14. To connect to the DB instance as the master user, use the user name and password that appear.

Step 15. Login to an EC2 instance and install the MySQL command-line to connect to your rds instance.

```
sudo yum install mariadb
```

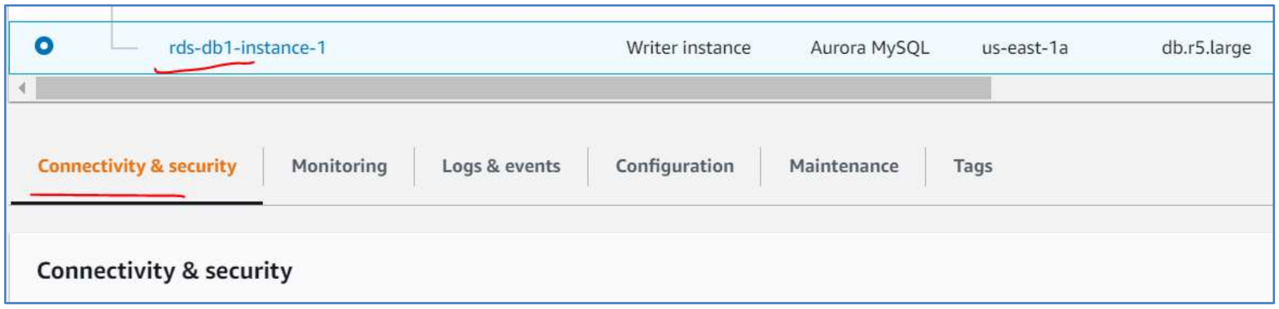
Step 16. Now, connect to your rds instance using mysql command. Change the instance name as per yours:

```
mysql -h rds-db1-instance-1.ckuog0pqacpd.us-east-1.rds.amazonaws.com  -P
3306 -u admin -p
```

Step 17. If you get below error while connecting, then follow next steps to resolve it:

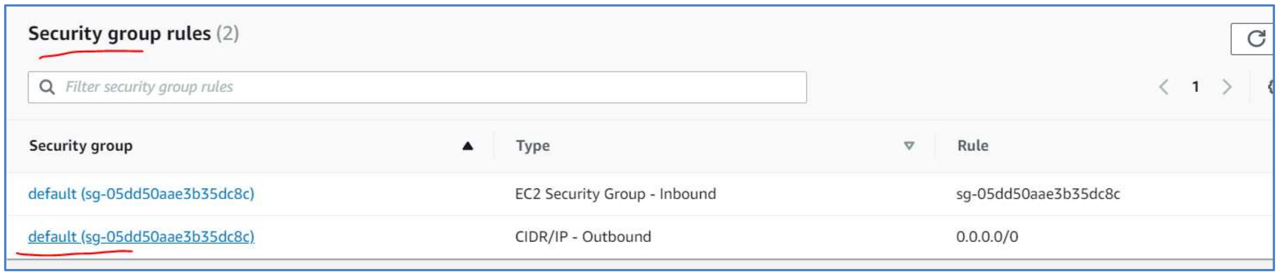
```
[ec2-user@ip-172-31-7-204 ~]$ mysql -h rds-db1-instance-1.ckuog0pqacpd.us-east-1.rds.amazonaws.com  -P 3306 -u admin -p
Enter password:
ERROR 2003 (HY000): Can't connect to MySQL server on 'rds-db1-instance-1.ckuog0pqacpd.us-east-1.rds.amazonaws.com' (110)
```

Step 18. Go to **Connectivity & Security** tab of your instance and scroll to **Security group rules.** Edit the Security Group of type **CIDR/IP - Outbound**



rds-db1-instance-1 | Writer instance | Aurora MySQL | us-east-1a | db.r5.large

Connectivity & security

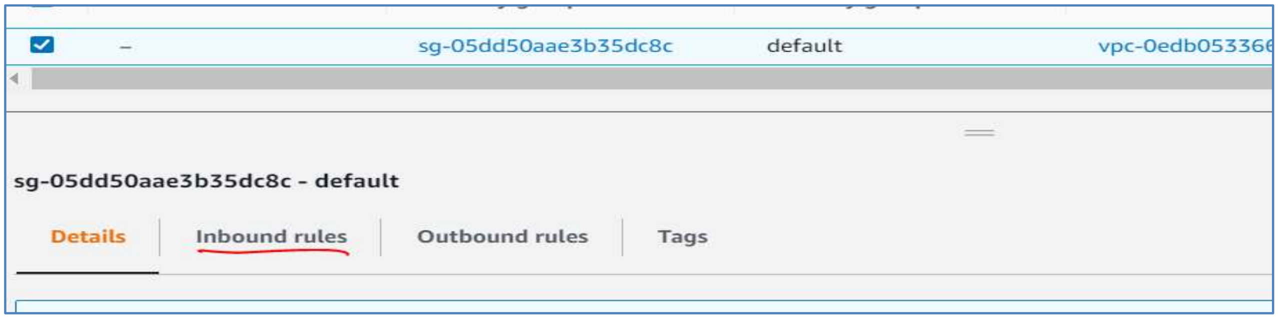


Security group rules (2)

Filter security group rules

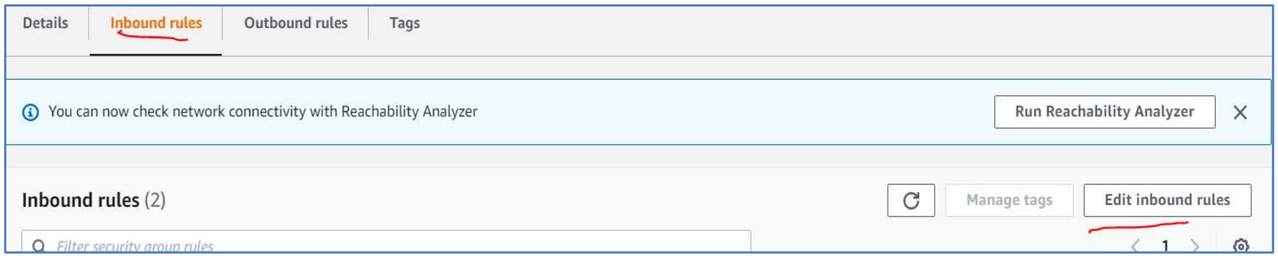
Security group	Type	Rule
default (sg-05dd50aae3b35dc8c)	EC2 Security Group - Inbound	sg-05dd50aae3b35dc8c
default (sg-05dd50aae3b35dc8c)	CIDR/IP - Outbound	0.0.0.0/0

Step 19. Click on the Security group and open it in new tab. Edit the Inbound rules as below:



– sg-05dd50aae3b35dc8c default vpc-0edb05336

sg-05dd50aae3b35dc8c - default

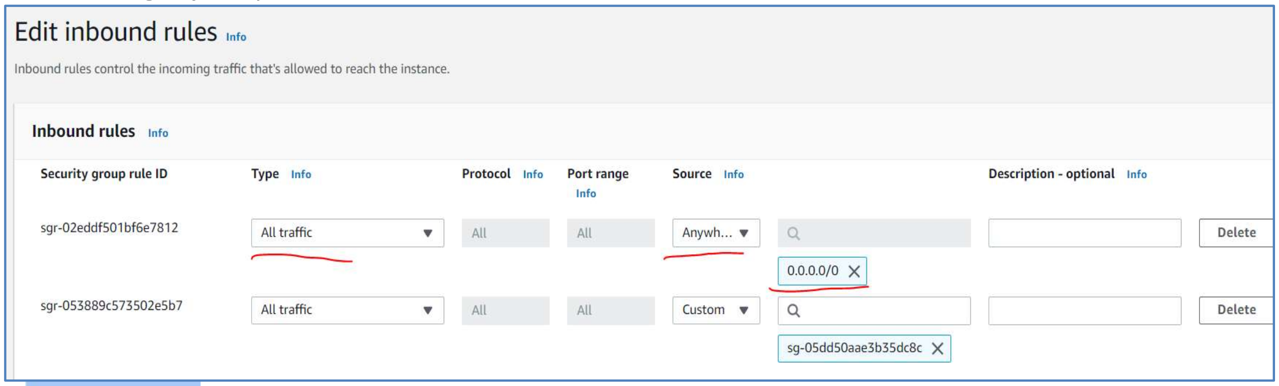


You can now check network connectivity with Reachability Analyzer

Inbound rules (2)

Filter security group rules

Step 20. Add a new rule to allow access from any where or you can restrict to your EC2 IP only by selecting **My IP** option:



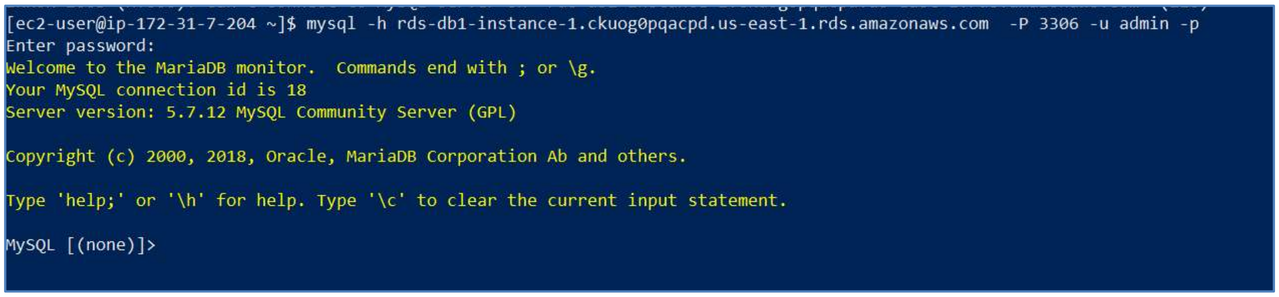
Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules Info

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
sgr-02eddf501bf6e7812	All traffic	All	All	Anywh...	0.0.0.0/0		Delete
sgr-053889c573502e5b7	All traffic	All	All	Custom	sg-05dd50aae3b35dc8c		Delete

Step 21. You should be able to connect now:



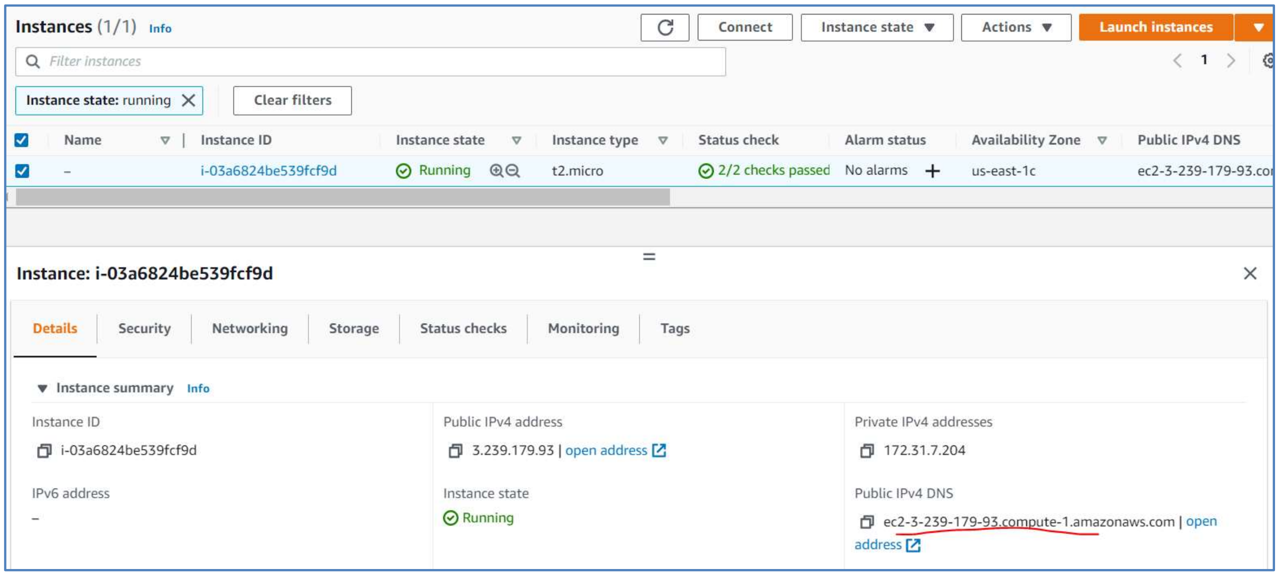
```
[ec2-user@ip-172-31-7-204 ~]$ mysql -h rds-db1-instance-1.ckuog0pqacpd.us-east-1.rds.amazonaws.com  -P 3306 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.7.12 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MySQL [(none)]>
```

Cache the static and frequently accessed application data in the memory using Amazon ElastiCache

Setting up the AWS Environment

Step 1. Start by deploying an EC2 instance in your AWS environment. The instance will be used to run a sample application that leverages a Redis cluster. For simplicity, we will use the same Amazon VPC for both the EC2 instance and the ElastiCache cluster so that additional configuration isn't required.

Step 2. Once your EC2 instance is up and running, go into its details and copy the **"Public IPv4 DNS"** URL. It will be used later to access the example web application.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
–	i-03a6824be539fcf9d	Running	t2.micro	2/2 checks passed	No alarms	us-east-1c	ec2-3-239-179-93.co

Instance: i-03a6824be539fcf9d

Instance ID
i-03a6824be539fcf9d

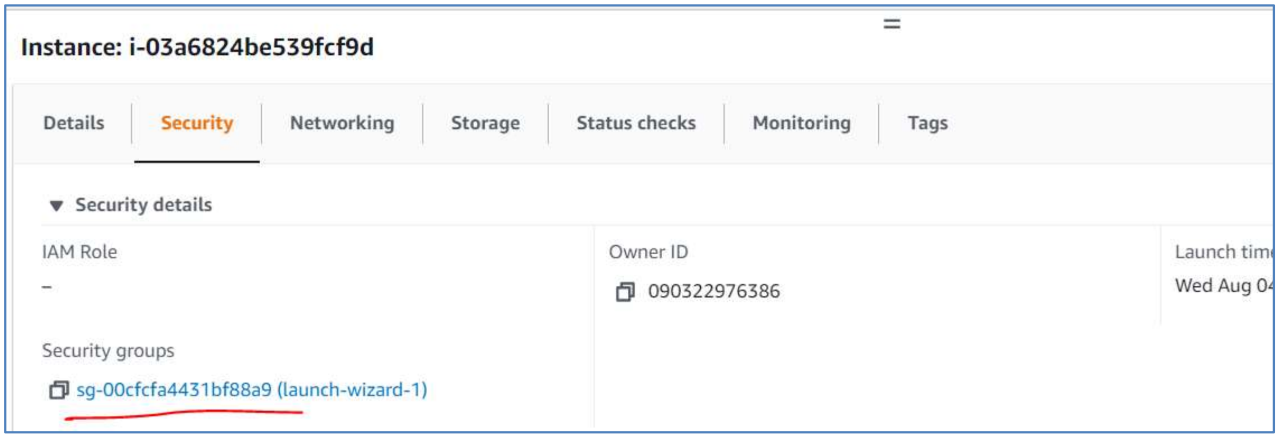
Public IPv4 address
3.239.179.93 | open address

Private IPv4 addresses
172.31.7.204

IPv6 address
–

Instance state
Running

Public IPv4 DNS
ec2-3-239-179-93.compute-1.amazonaws.com | open address


Instance: i-03a6824be539fcf9d

Security details

IAM Role: –

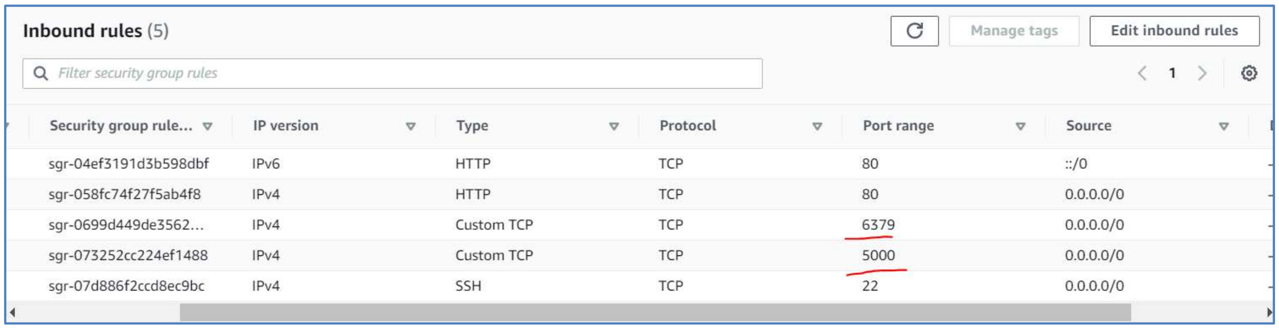
Owner ID: 090322976386

Launch tim Wed Aug 04

Security groups: sg-00cfcfa4431bf88a9 (launch-wizard-1)

Step 3. The security group assigned to the EC2 instance should allow two custom TCP inbound rules: one for the Redis default TCP port (6379) using the same security group as source—allowing connections on this port from any instance within that security group—and another for the example application on the TCP port (5000) and using 0.0.0.0/0 as source to enable access to the sample web application from your computer.

Step 4. Open the security group and edit inbound rules:

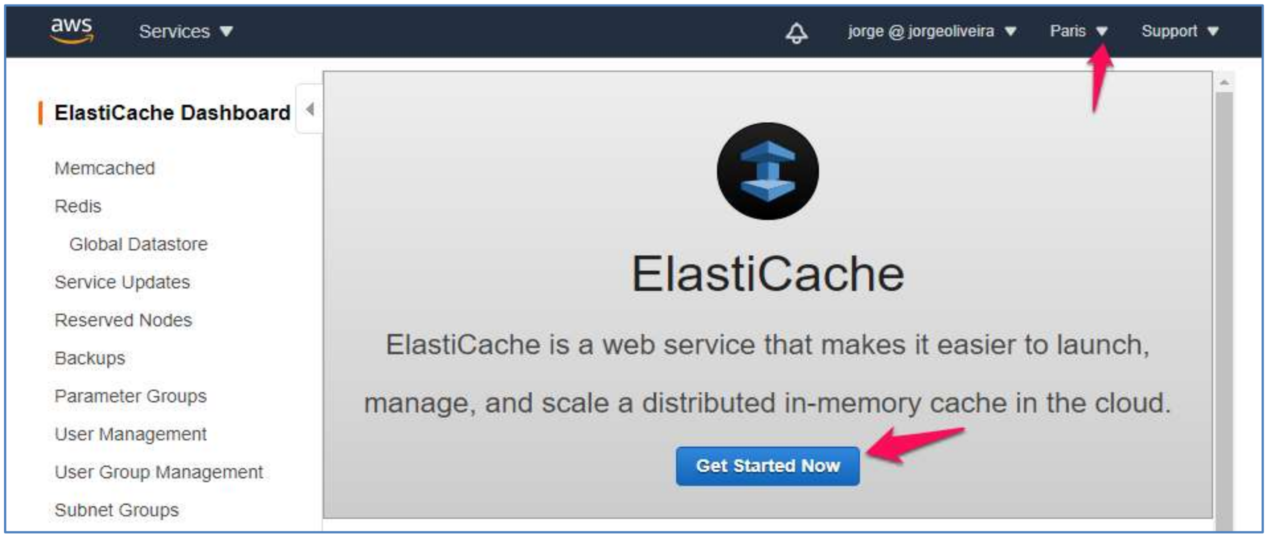

Inbound rules (5)

Security group rule...	IP version	Type	Protocol	Port range	Source
sgr-04ef3191d3b598dbf	IPv6	HTTP	TCP	80	::/0
sgr-058fc74f27f5ab4f8	IPv4	HTTP	TCP	80	0.0.0.0/0
sgr-0699d449de3562...	IPv4	Custom TCP	TCP	6379	0.0.0.0/0
sgr-073252cc224ef1488	IPv4	Custom TCP	TCP	5000	0.0.0.0/0
sgr-07d886f2ccd8ec9bc	IPv4	SSH	TCP	22	0.0.0.0/0

Create an AWS ElastiCache Cluster for Redis

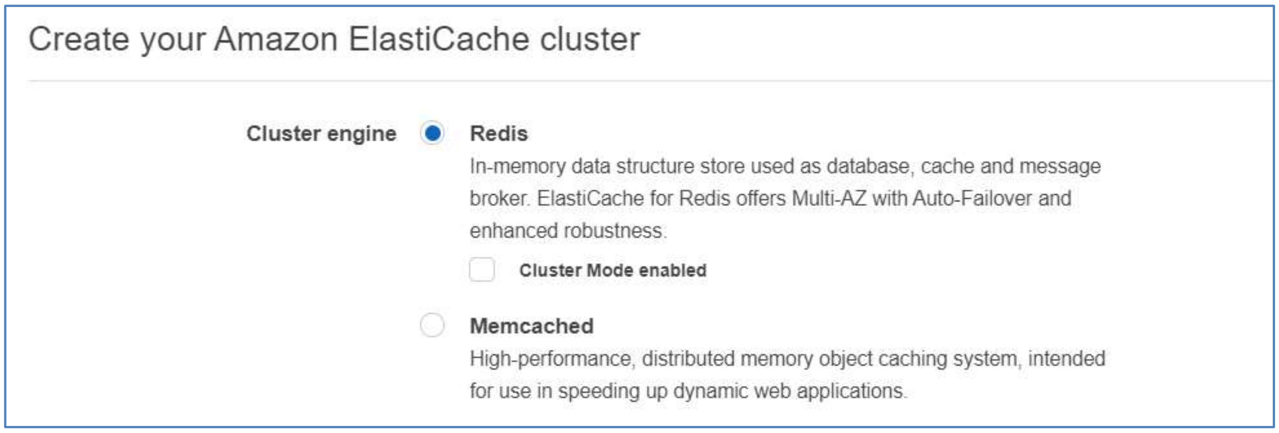
Now we'll see how to create an AWS ElastiCache cluster for Redis.

Step 1. Open the **ElastiCache Dashboard** in the AWS Console and click on the **"Get Started Now"** button. Keep in mind that the AWS Region selected in the top right corner will be used as a location for your AWS Redis cache cluster deployment. Use the same region where your EC2 instance is located.



ElastiCache is a web service that makes it easier to launch, manage, and scale a distributed in-memory cache in the cloud.

Step 2. Start by creating a new cluster and selecting “**Redis”** as the ElastiCache engine. Notice that AWS Redis Cluster Mode can optionally be enabled. While not required for this example, it is good to keep in mind that Cluster Mode enables you to horizontally scale the AWS Redis cluster and provision up to 500 primary nodes, making it highly available and increasing its fault tolerance.



Create your Amazon ElastiCache cluster

Cluster engine: [✓ Redis] Memcached

Redis
In-memory data structure store used as database, cache and message broker. ElastiCache for Redis offers Multi-AZ with Auto-Failover and enhanced robustness.

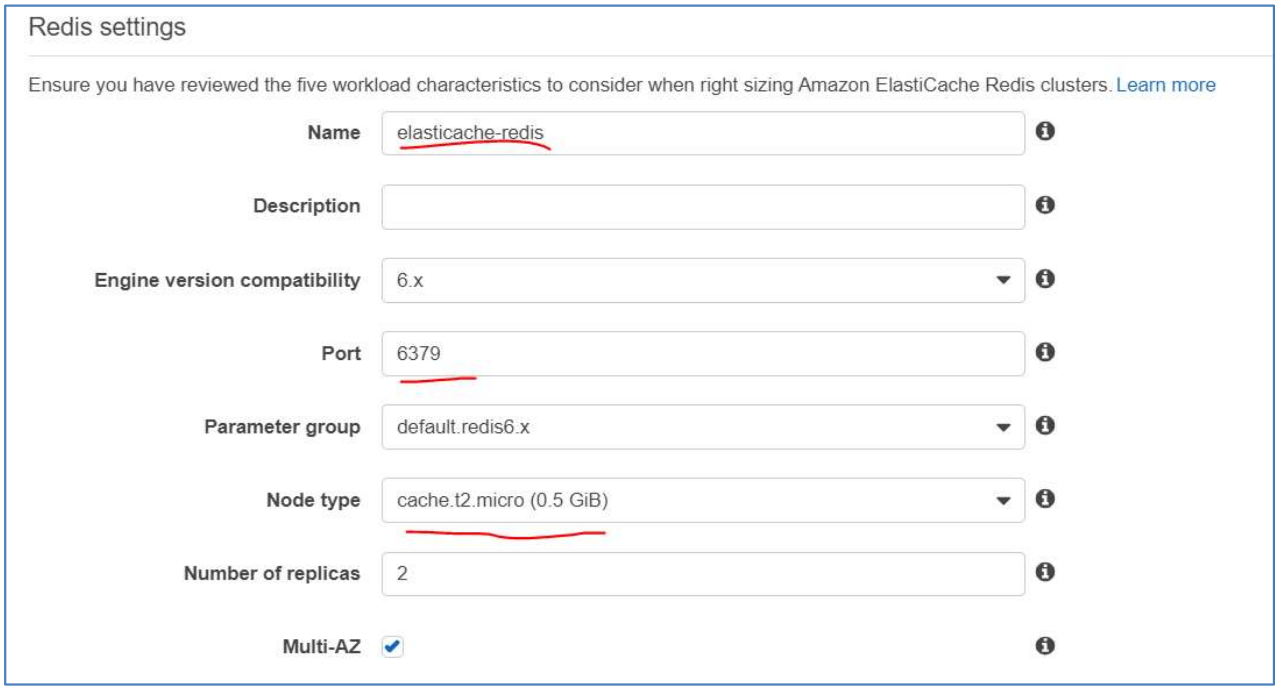
☐ Cluster Mode enabled

Memcached
High-performance, distributed memory object caching system, intended for use in speeding up dynamic web applications.

Step 3. Use the default Amazon Cloud location for the cluster and provide a unique name to identify the cluster (e.g. “elasticache-redis”).

Step 4. ElastiCache provides a variety of cache instance types to choose from, each targeting different performance and storage needs. For this example, you can select the “cache.t2.micro” type which is enough for this demonstration and is free tier eligible.

Step 5. Proceed with the default options, but take note of some advanced features available such as different engine versions, multi-replica and multi-AZ support for enhanced high availability and compatibility.



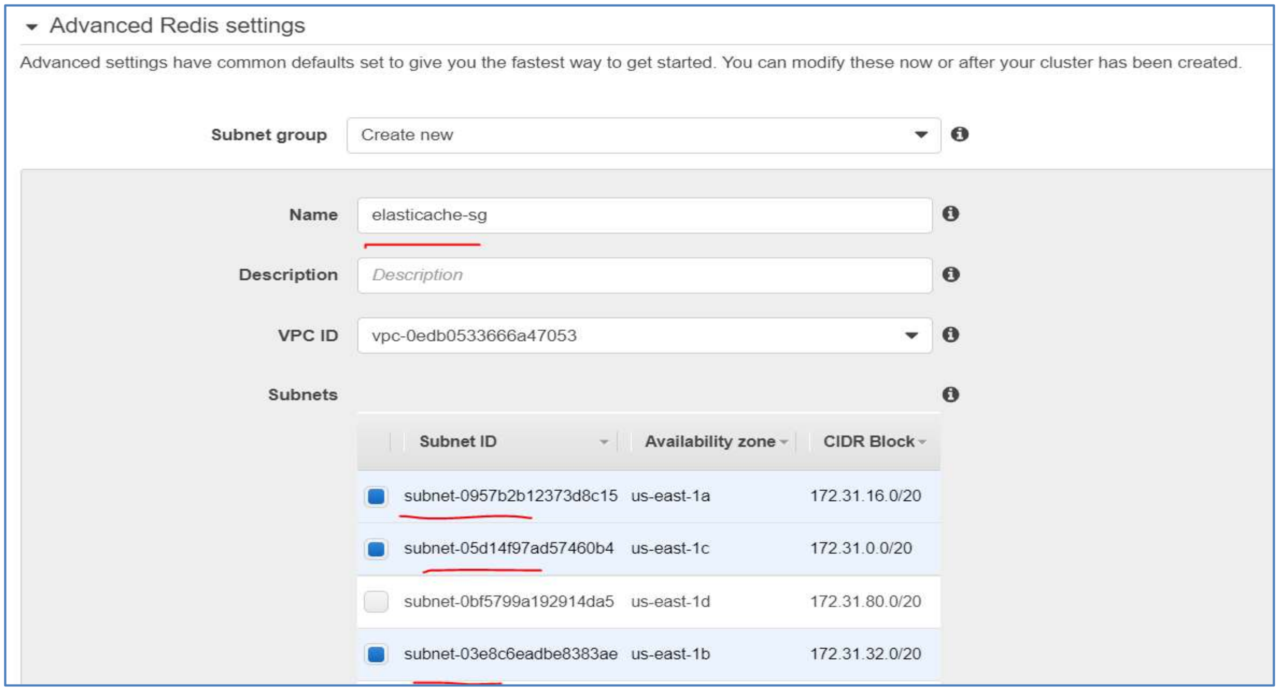
Redis settings

Ensure you have reviewed the five workload characteristics to consider when right sizing Amazon ElastiCache Redis clusters. Learn more

Name	elasticache-redis
Description	
Engine version compatibility	6.x
Port	6379
Parameter group	default.redis6.x
Node type	cache.t2.micro (0.5 GiB)
Number of replicas	2
Multi-AZ	✓

Step 6. Under Advanced settings, create a new subnet group, providing a unique name to identify it. Remember to select the same VPC as your EC2 Instance, and at least two subnets in case you want to try the Multi-AZ feature.

Step 7. Under the Security panel, select the previously created security group that was assigned to the EC2 instance.



Advanced Redis settings

Advanced settings have common defaults set to give you the fastest way to get started. You can modify these now or after your cluster has been created.

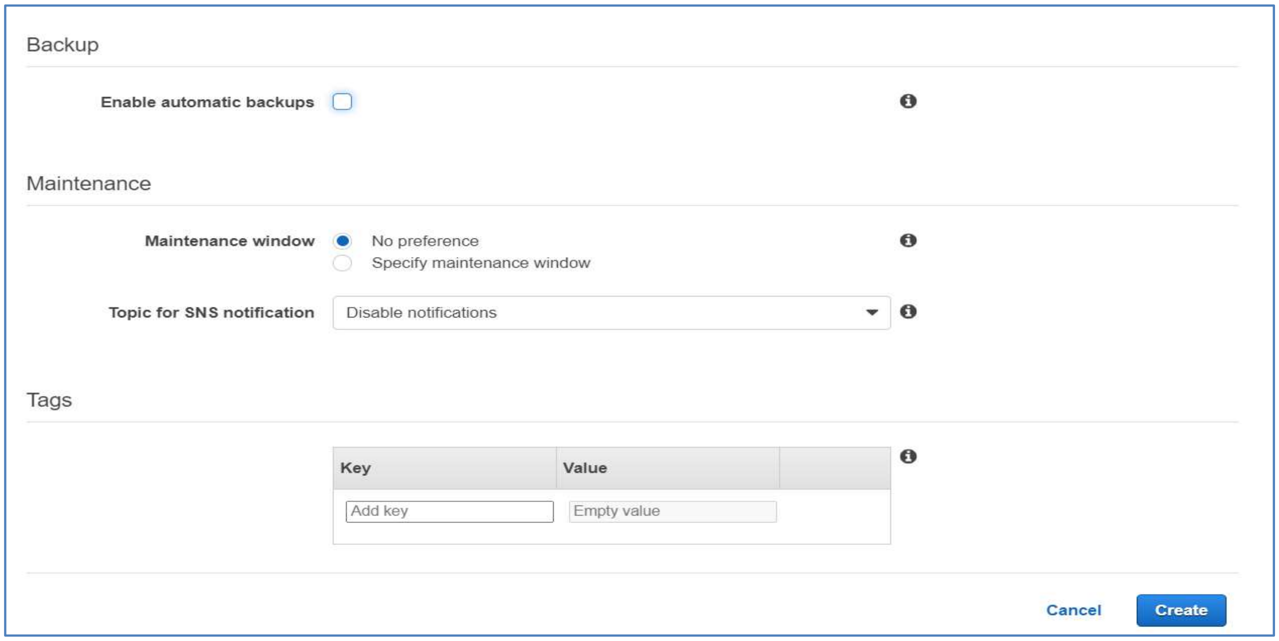
Subnet group	Create new
Name	elasticache-sg
Description	Description
VPC ID	vpc-0edb0533666a47053

Subnets

	Subnet ID	Availability zone	CIDR Block
✓	subnet-0957b2b12373d8c15	us-east-1a	172.31.16.0/20
✓	subnet-05d14f97ad57460b4	us-east-1c	172.31.0.0/20
	subnet-0bf5799a192914da5	us-east-1d	172.31.80.0/20
✓	subnet-03e8c6eadbe8383ae	us-east-1b	172.31.32.0/20

Step 8. While the automatic backup feature is selected by default, you can opt-out and uncheck that box for this example since you won't be needing it.

Step 9. Once you review all the settings, click the "Create" button.



Backup

Enable automatic backups ☐

Maintenance

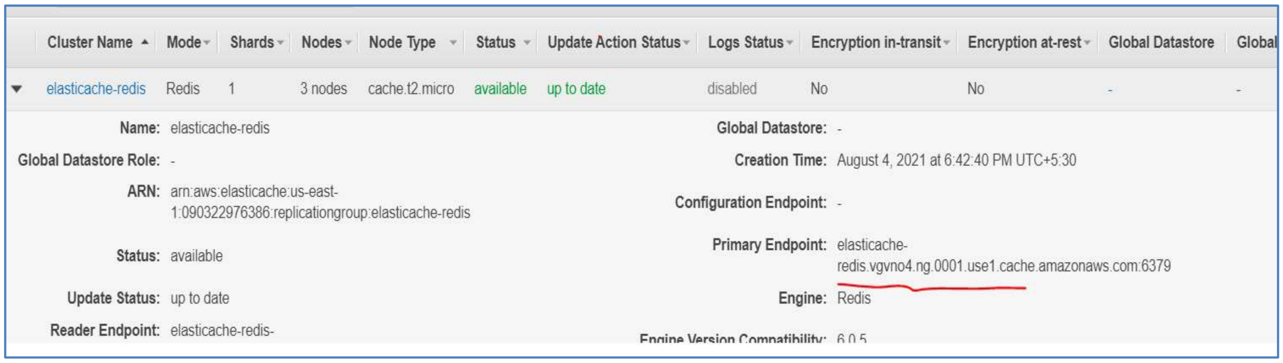
Maintenance window [✓ No preference] Specify maintenance window

Topic for SNS notification: Disable notifications

Tags

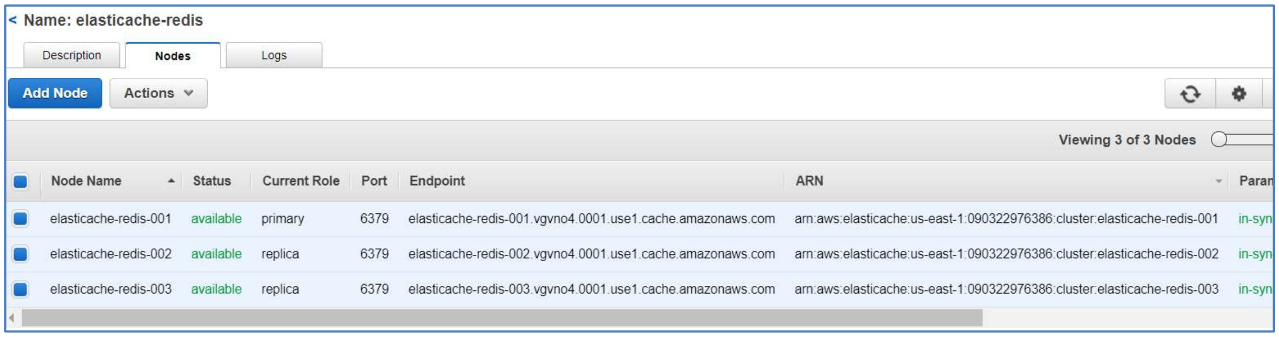
Key	Value
Add key	Empty value

Step 10. Once the Redis Cluster becomes available, click on the arrow to display it's configuration details, and copy the **"Primary Endpoint"** URL to be used later on by the example web application.



	Cluster Name	Mode	Shards	Nodes	Node Type	Status	Update Action Status	Logs Status	Encryption in-transit	Encryption at-rest	Global Datastore	Global
▼	elasticache-redis	Redis	1	3 nodes	cache.t2.micro	available	up to date	disabled	No	No	-	-

Name:	elasticache-redis	Global Datastore:	-
Global Datastore Role:	-	Creation Time:	August 4, 2021 at 6:42:40 PM UTC+5:30
ARN:	arn:aws:elasticache:us-east-1:090322976386:replicationgroup:elasticache-redis	Configuration Endpoint:	-
Status:	available	Primary Endpoint:	elasticache-redis.vgvno4.ng.0001.use1.cache.amazonaws.com:6379
Update Status:	up to date	Engine:	Redis
Reader Endpoint:	elasticache-redis-	Engine Version Compatibility:	6.0.5



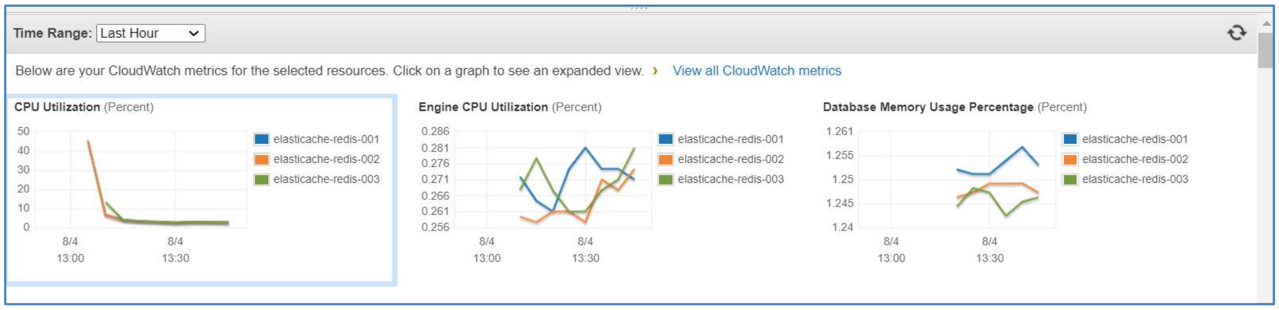
< Name: elasticache-redis

Description | Nodes | Logs

Add Node | Actions

Viewing 3 of 3 Nodes

	Node Name	Status	Current Role	Port	Endpoint	ARN	Paran
■	elasticache-redis-001	available	primary	6379	elasticache-redis-001.vgvno4.0001.use1.cache.amazonaws.com	arn:aws:elasticache:us-east-1:090322976386:cluster:elasticache-redis-001	in-syn
■	elasticache-redis-002	available	replica	6379	elasticache-redis-002.vgvno4.0001.use1.cache.amazonaws.com	arn:aws:elasticache:us-east-1:090322976386:cluster:elasticache-redis-002	in-syn
■	elasticache-redis-003	available	replica	6379	elasticache-redis-003.vgvno4.0001.use1.cache.amazonaws.com	arn:aws:elasticache:us-east-1:090322976386:cluster:elasticache-redis-003	in-syn



Time Range: Last Hour

Below are your CloudWatch metrics for the selected resources. Click on a graph to see an expanded view. › View all CloudWatch metrics

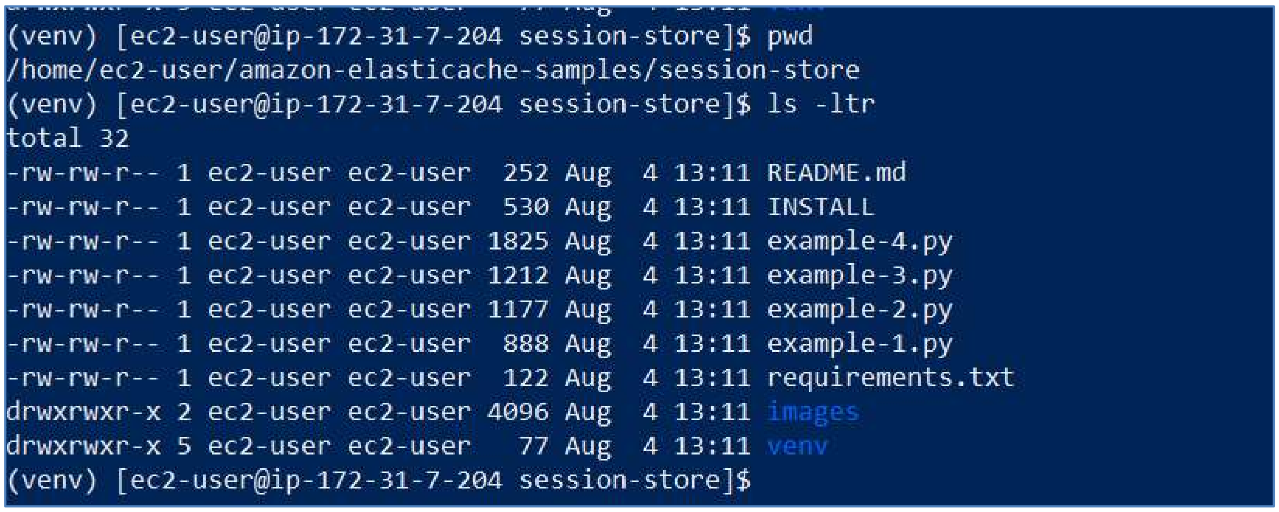
How to Use Redis as a Session Store for a Web Application

Redis is commonly used as a session store in scalable web applications, where storing and managing the users' session data is needed.
In this example, we will use a simple web application that enables visitors to log in and log out and uses Redis to store their session data. The application is part of AWS own example collection for ElastiCache and you can find it's code among other examples here.

Step 1. Establish an interactive remote SSH session with the EC2 instance you deployed earlier, and run the following commands in order to install the required tools and dependencies needed by the application:

```
sudo yum install git -y
sudo yum install python3 -y
sudo pip3 install virtualenv
git clone https://github.com/aws-samples/amazon-elasticache-samples/
cd amazon-elasticache-samples/session-store
virtualenv venv
source ./venv/bin/activate
pip3 install -r requirements.txt
```

Step 2. Once all steps above completed successfully, then you should be able to see below files in the current directory



```
(venv) [ec2-user@ip-172-31-7-204 session-store]$ pwd
/home/ec2-user/amazon-elasticache-samples/session-store
(venv) [ec2-user@ip-172-31-7-204 session-store]$ ls -ltr
total 32
-rw-rw-r-- 1 ec2-user ec2-user  252 Aug  4 13:11 README.md
-rw-rw-r-- 1 ec2-user ec2-user  530 Aug  4 13:11 INSTALL
-rw-rw-r-- 1 ec2-user ec2-user 1825 Aug  4 13:11 example-4.py
-rw-rw-r-- 1 ec2-user ec2-user 1212 Aug  4 13:11 example-3.py
-rw-rw-r-- 1 ec2-user ec2-user 1177 Aug  4 13:11 example-2.py
-rw-rw-r-- 1 ec2-user ec2-user  888 Aug  4 13:11 example-1.py
-rw-rw-r-- 1 ec2-user ec2-user  122 Aug  4 13:11 requirements.txt
drwxrwxr-x 2 ec2-user ec2-user 4096 Aug  4 13:11 images
drwxrwxr-x 5 ec2-user ec2-user   77 Aug  4 13:11 venv
(venv) [ec2-user@ip-172-31-7-204 session-store]$
```

Step 3. Define the following environment variables for the application:

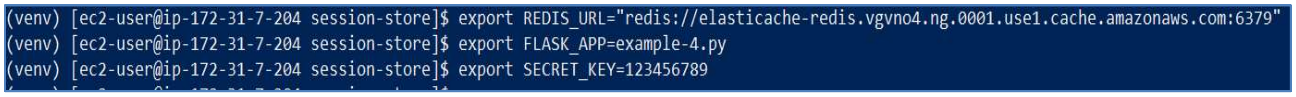
- REDIS_URL
- FLASK_APP
- SECRET_KEY.

▪ The value of REDIS_URL <your_redis_endpoint> will be set to the ElastiCache Primary Endpoint value saved earlier.

▪ The FLASK_APP value should point to the Python file (example-4.py) of the web application example we are going to run.

▪ The SECRET_KEY value can be filled with any random string since it’s only used by the example application as a seed to generate the session cookie.

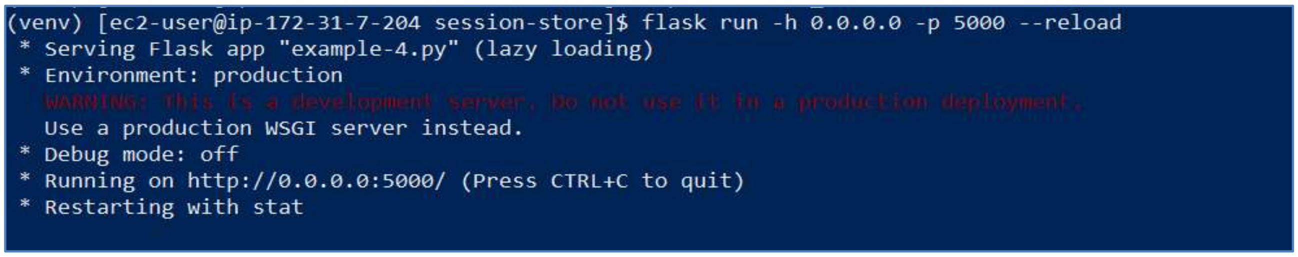
```
export REDIS_URL="redis://elasticache-redis.vgvno4.ng.0001.use1.cache.ama
zonaws.com:6379"
export FLASK_APP=example-4.py
export SECRET_KEY=123456789
```



```
(venv) [ec2-user@ip-172-31-7-204 session-store]$ export REDIS_URL="redis://elasticache-redis.vgvno4.ng.0001.use1.cache.amazonaws.com:6379"
(venv) [ec2-user@ip-172-31-7-204 session-store]$ export FLASK_APP=example-4.py
(venv) [ec2-user@ip-172-31-7-204 session-store]$ export SECRET_KEY=123456789
```

Step 4. Run the example web application using the command:

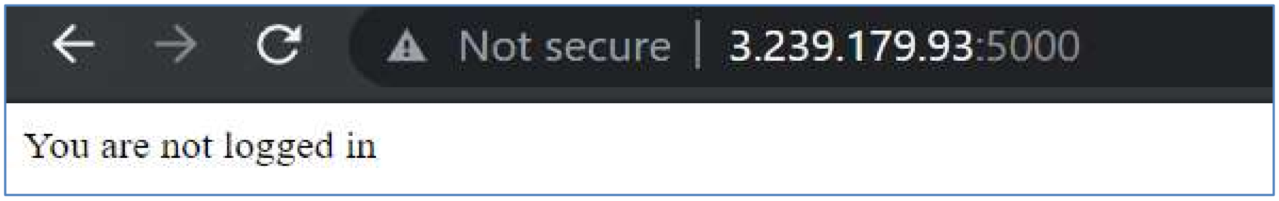
```
flask run -h 0.0.0.0 -p 5000 --reload
```



```
(venv) [ec2-user@ip-172-31-7-204 session-store]$ flask run -h 0.0.0.0 -p 5000 --reload
 * Serving Flask app "example-4.py" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
```

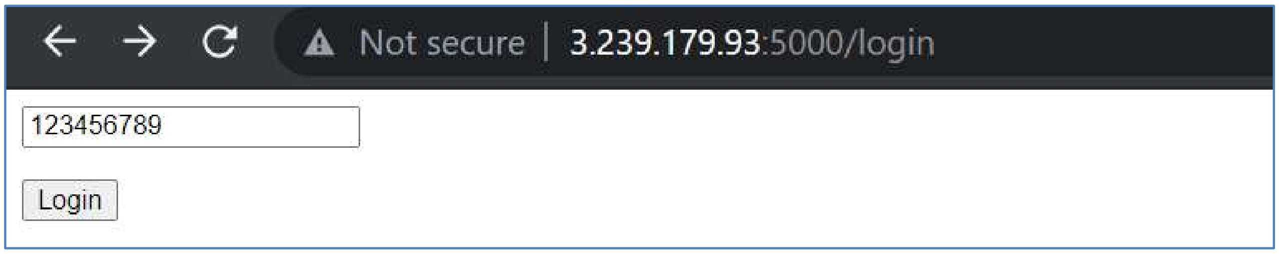
Step 5. With the application running in the background, use the Public DNS name from your EC2 instance details to access the web application. In your browser, visit the URL in the port number 5000.

The URL will be **http://<your_ec2_public_dns>:5000**. By default, three endpoints will be made available, **“/”**, **“/login”** and **“/logout”.**



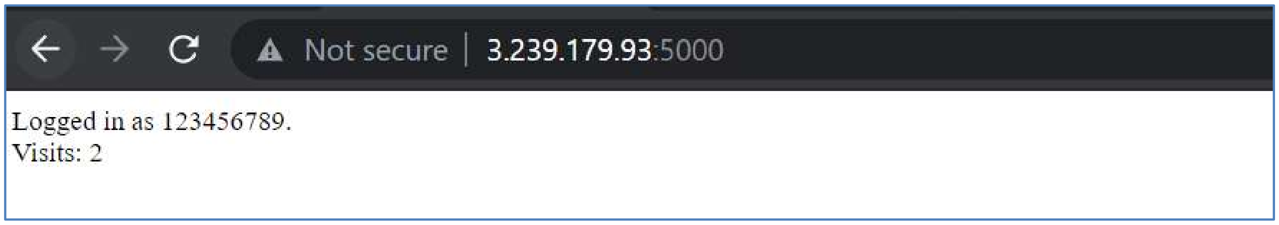
Not secure | 3.239.179.93:5000

You are not logged in



Not secure | 3.239.179.93:5000/login

123456789

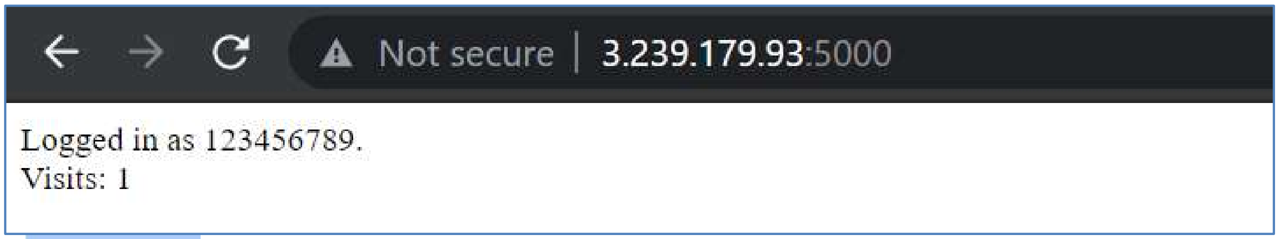


Not secure | 3.239.179.93:5000

Logged in as 123456789.
Visits: 2

Step 6. Log in into the application using the /login endpoint using any random credentials and refresh the page a few times. You will notice that the number of visits will increment every time the same user visits, and if you allow 10 seconds to elapse without refreshing the page the counter will be reset to 1.

Step 7. Under the hood, upon login the application generates a unique token that represents the Redis key under which the user session data will be stored in the cluster. Every user access to the web page will increment the counter in Redis with the number of visits. Since the web application sets a Time to Live (TTL) of 10 seconds for the user data stored in Redis, the session and user counter will be automatically reset after the time elapses.



Not secure | 3.239.179.93:5000

Logged in as 123456789.
Visits: 1