

# **BUILDING A FAST SESSION STORE FOR ONLINE APPLICATIONS WITH AMAZON ELASTICACHE FOR REDIS**

*Prepared in the partial fulfillment of the Summer Internship Program on AWS*

AT



*Under the guidance of*

**Mrs. Sumana Bethala, APPSDC**

**Mr. Rama Krishna, APSSDC**

*Submitted by*

**Prem Sagar Bommanaboena, Y20cs026**

**RVRJC College of Engineering**

*(August, 2023)*

## ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of my summer internship project at **Andhra Pradesh Skill Development Corporation (APSSDC)**. This opportunity has been an enriching and transformative experience for me, and I am truly thankful for the support, guidance, and encouragement I have received along the way.

First and foremost, I extend my sincere regards to Mrs Sumana Bethala and Mr Rama Krishna, my supervisors and mentors, for providing me with valuable insights, constant guidance, and unwavering support throughout the duration of the internship. Their expertise and encouragement have been instrumental in shaping the direction of this project.

I would like to thank the entire team at **Andhra Pradesh Skill Development Corporation (APSSDC)** for fostering a collaborative and innovative environment. The camaraderie, knowledge sharing, and feedback I received from my colleagues significantly contributed to the development and success of this project.

In conclusion, I am honored to have been a part of this internship program, and I look forward to leveraging the skills and knowledge gained to contribute positively to future endeavors.

Thank you.

Sincerely,

prem sagar bommanaboena, Y20C026

[premsagar15102002@gmail.com](mailto:premsagar15102002@gmail.com)

## ABSTRACT

There are many ways of managing user sessions in web applications, ranging from cookies-only to distributed key/value databases, including server-local caching. Storing session data in the web server responding to a given request may seem convenient, as accessing the data incurs no network latency. The main drawback is that requests have to be routed carefully so that each user interacts with one server and one server only. Another drawback is that once a server goes down, all the session data is gone as well. A distributed, in-memory key/value database can solve both issues by paying the small price of a tiny network latency. Storing all the session data in cookies is good enough most of the time; if you plan to store sensitive data, then using server-side sessions is preferable. The project aims to enhance the performance and responsiveness of online applications by implementing a high-speed session store using Amazon ElastiCache for Redis. In modern online applications, maintaining user sessions and their associated data efficiently is crucial for delivering a seamless user experience. Amazon ElastiCache for Redis offers an in-memory data store that can significantly accelerate data retrieval, making it an ideal solution for storing and managing session-related information. This project involves integrating Amazon ElastiCache for Redis into the application's architecture as a session store. By doing so, the project will leverage the memory-based storage capabilities of Redis to store and retrieve session data rapidly. This will lead to reduced response times, improved scalability, and enhanced user satisfaction.

## TABLE OF CONTENTS

	Page No.
1. Introduction	5
1.1 Background	5
1.2 Problem Statement	6
1.3 Objectives	6
1.4 Limitations of the Existing Techniques	7
2. Methodologies	
2.1. Redis	8
2.2 Amazon ElastiCache for Redis	9
2.3. Use Cases for Amazon ElastiCache for Redis	10
3. System Design and Architecture	
3.1 System Design	12
3.2 Architecture	13
4. Implementation	
4.1 Prerequisites	15
4.2 Create a Redis Cluster	16
4.3 Session Caching with Redis	23
4.4 Cleanup	29
5. Results	32
6. Conclusion	33

# 1. INTRODUCTION

Session state refers to the data that captures the ongoing user interactions within applications, whether they are websites, games, or other interactive platforms. A typical web application retains a session for each user, maintaining their context as long as they remain logged in. This session state holds critical information such as user identity, login credentials, preferences, recent actions, shopping cart contents, and more.

Efficiently reading and writing session data during each user interaction is vital to maintaining a seamless user experience. Behind the scenes, session state is cached data specific to a user or application, enabling rapid responses to user actions. Therefore, while a user session is active, unnecessary round-trips to a central database should be avoided.

The final step in the session state lifecycle occurs when the user disconnects. While some data may be persisted in the database for future use, transient information can be discarded once the session ends.

## 1.1 Background:

In the fast-paced world of online applications, delivering exceptional user experiences has become a top priority. Slow response times and performance bottlenecks can deter users and impact an application's success. One critical aspect of optimizing user experience is effective session management. This involves maintaining user-specific data and context during their interactions with the application. Traditional session storage methods, often relying on disk-based databases, may hinder performance as the application scales. As a result, there is a growing need for efficient and high-speed session storage solutions.

### Challenges and Best Practices for Session State:

- Session state management presents challenges related to isolation, volatility, and persistence. While a session is active, the application exclusively reads from and writes to the in-memory session store. This ensures faster write operations but offers no data loss tolerance. Since session store data isn't a simple snapshot of another database, it must be

highly durable and consistently available.

- Session state resembles caching, but it operates differently in terms of read and write cycles. A cache tolerates data loss and can be restored from a primary database at any time. Writing to a cache often necessitates writing to the underlying database. Conversely, session state restoration only occurs from the primary data source at the start of a user session and is persisted back to the source when the session concludes.
- Session state can be volatile or permanent, dictating whether data is discarded or persisted to disk storage after the session ends. Volatile data, like page navigation history, might be discarded, while a durable shopping cart in an e-commerce app must be saved permanently.
- Storing session state involves using key-value pairs with user identifiers as keys and session data as values. This ensures that different user sessions remain isolated.

## **1.2 Problem Statement:**

How can the challenge of creating a session management system that ensures seamless and rapid data retrieval for improved application responsiveness be overcome? Additionally, what innovative solutions can be explored to address the issues related to traditional session storage mechanisms and provide a high-speed, reliable, and scalable session management mechanism?

## **1.3 Objectives:**

This project aims to leverage Amazon ElastiCache for Redis to build a high-speed session store that enhances the performance of online applications. The specific objectives include:

- Integrating Amazon ElastiCache for Redis into the application architecture to serve as the primary session store.
- Establishing efficient mechanisms for storing and retrieving session-related data using Redis data structures.
- Implementing data serialization techniques to optimize storage and retrieval of complex session data.
- Setting up expiration and cleanup mechanisms to manage session data lifecycle effectively.

- Ensuring robust security measures to safeguard sensitive session information.
- Testing the performance of the application with the Redis-backed session store and optimizing configurations for maximum efficiency.
- Providing comprehensive documentation outlining the integration process, best practices, and usage guidelines.

## **1.4 Limitation of Existing Techniques:**

The limitations associated with existing session storage techniques highlight the need for innovative solutions like Amazon ElastiCache for Redis. Traditional methods often suffer from issues such as disk I/O bottlenecks, scalability challenges, data loss risk, latency, complex configuration, and limited in-memory capabilities. They might lack advanced features, raise security concerns, entail high overhead, and struggle with complex data structures. In contrast, Redis, with its in-memory capabilities, optimized data structures, and advanced features like data expiration and automatic failover, offers a compelling solution to these challenges. It ensures high-speed session storage, scalability, and efficient session management, making it a strong candidate for powering responsive modern online applications.

## 2. METHODOLOGIES

### 2.1. Redis:

Redis, or Remote Dictionary Server, is a high-performance, open-source in-memory data store that serves as a database, cache, message broker, and queue. It was developed by Salvatore Sanfilippo to address scalability challenges in his startup. Redis stores data in memory, offering sub-millisecond response times and enabling millions of requests per second for real-time applications in various industries.



Fig 2.1.1 Working of Amazon ElastiCache for Redis

### Benefits of Redis:

**Performance:** Redis stores data in memory, resulting in low-latency and high-throughput data access. This translates to fast response times and the ability to handle a high number of operations per second.

**Flexible Data Structures:** Redis provides a wide range of data structures such as strings, lists, sets, sorted sets, hashes, bitmaps, hyperloglogs, streams, geospatial data, JSON, and more. This diversity allows developers to tailor data storage to specific application needs.

**Simplicity:** Redis simplifies coding by offering straightforward command structures. Developers can achieve complex tasks with minimal lines of code, enhancing productivity.



**Replication and Persistence:** Redis employs a primary-replica architecture for improved read performance and faster recovery in case of primary server failures. It supports point-in-time backups for data persistence.

**High Availability and Scalability:** Redis supports both single node and clustered topologies, enabling highly available solutions and scalability through options to scale up or out.

**Open Source:** Redis is an open-source project supported by a thriving community, making it a widely adopted solution.

## **2.2 Amazon ElastiCache for Redis:**

Amazon ElastiCache for Redis is a fully managed service provided by AWS that offers a Redis-compatible in-memory data store. It leverages the speed and versatility of open-source Redis while adding AWS manageability, security, and scalability.

### **Benefits of Amazon ElastiCache for Redis:**

**Great Performance:** ElastiCache offers blazing-fast sub-millisecond response times, making it suitable for demanding applications requiring quick data access.

**Security:** ElastiCache supports Role-Based Access Control (RBAC), Amazon VPC isolation, and encryption in transit and at rest, ensuring the security of sensitive data.

**Fully Managed:** ElastiCache handles management tasks such as hardware provisioning, patching, monitoring, and backups. This allows developers to focus on application development.

**Compatibility:** ElastiCache maintains compatibility with open-source Redis data formats and APIs, enabling seamless migration of self-managed Redis workloads.

**High Availability:** ElastiCache supports automatic failover, read replicas, and enhanced failover logic to ensure high availability of Redis clusters.

**Scalability:** ElastiCache enables easy scaling with online cluster resizing and support for up to 500 nodes and 500 shards in Redis Cluster environments.

## 2.3. Use Cases for Amazon ElastiCache for Redis:

**Caching:** ElastiCache is ideal for implementing a highly available, distributed, and secure in-memory cache. It reduces access latency, increases throughput, and lightens the load on backend databases and applications.

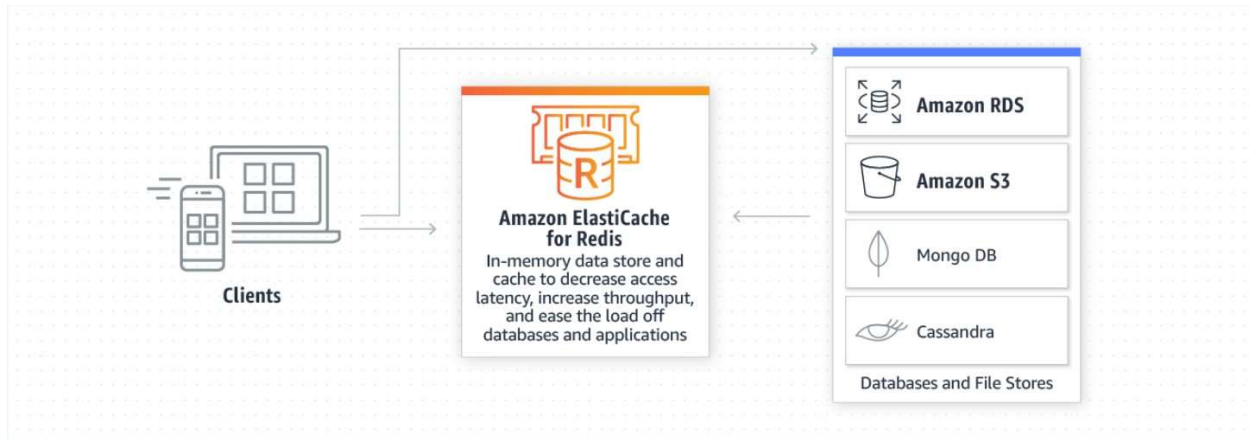


Fig 2.3.1 Use Cases for Amazon ElastiCache for Redis in caching

**Chat and Messaging:** ElastiCache supports the PUB/SUB standard, making it suitable for high-performance chat rooms, real-time comment streams, and server intercommunication.



Fig 2.3.2 Use Cases for Amazon ElastiCache for Redis in Chat and Messaging

**Geospatial:** ElastiCache offers specialized data structures for managing real-time geospatial data, enabling location-based features in applications.

**Machine Learning:** ElastiCache's fast in-memory data store is beneficial for deploying machine learning models quickly, supporting use cases such as fraud detection, real-time bidding, and matchmaking.

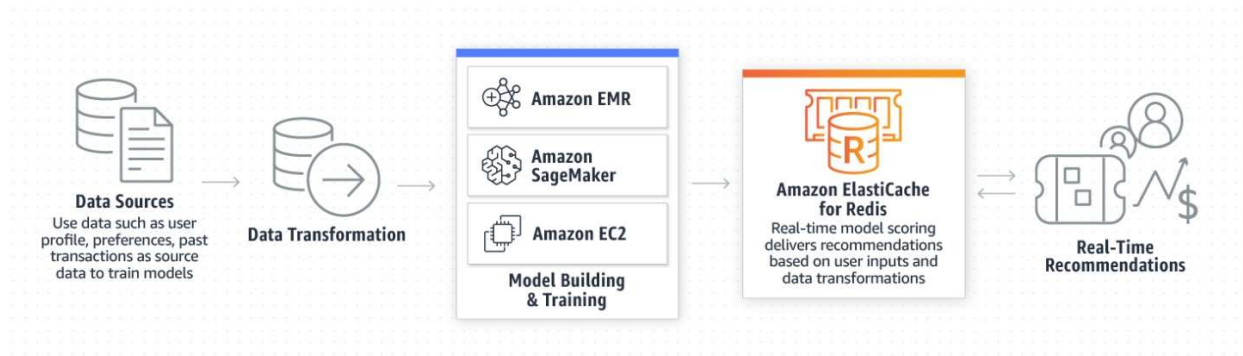


Fig 2.3.3 Use Cases for Amazon ElastiCache for Redis in machine Learning

**Media Streaming:** ElastiCache powers live streaming use cases by storing metadata, authentication information, and manifest files, enabling efficient content delivery.

**Queues:** ElastiCache's List data structure supports lightweight, persistent queues, making it suitable for applications requiring reliable message brokering.

**Real-time Analytics:** ElastiCache can be used with streaming solutions like Apache Kafka and Amazon Kinesis for real-time data analysis, suitable for social media, ad targeting, and IoT applications.

**Session Store:** ElastiCache serves as an excellent session store for managing user authentication tokens, session state, and more in online applications.

The combination of Redis and Amazon ElastiCache for Redis addresses the limitations of traditional session storage techniques by providing high performance, scalability, ease of use, and robust security, making them well-suited for modern online applications.

### 3.System Design and Architecture

Here is a system design and architecture to implement a fast session store for your online applications with Amazon ElastiCache for Redis:

#### 3.1 System Design

The system will consist of the following components:

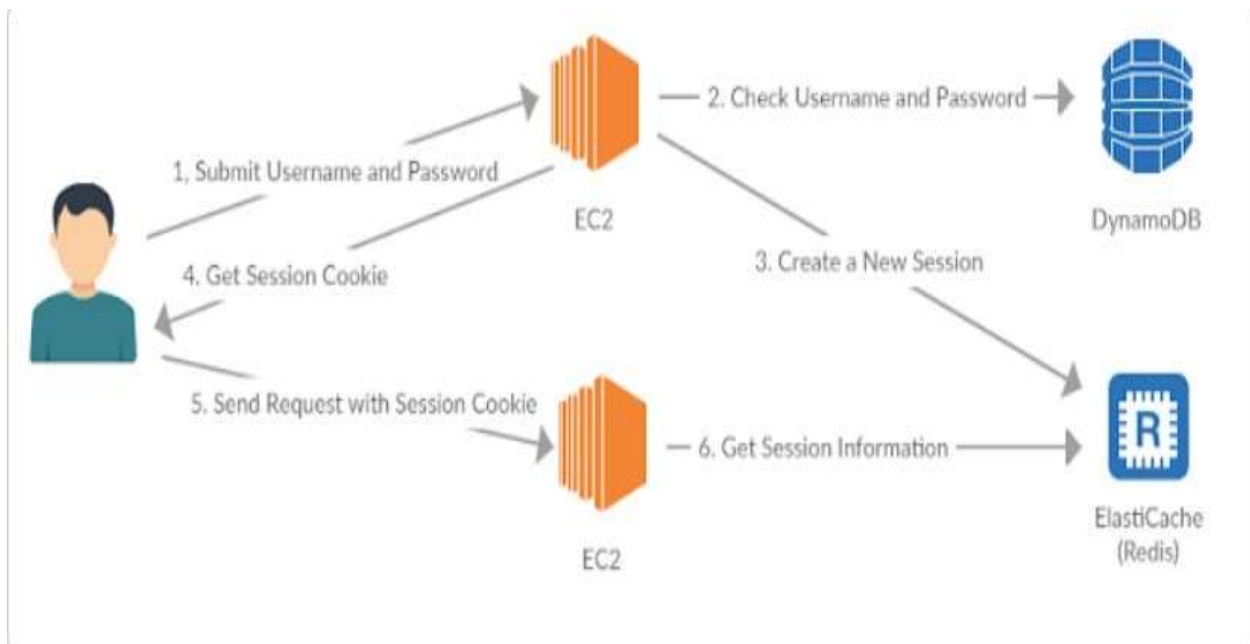


Fig 3.1.1 System Design

**Web server:** This is the front-end of the application that users interact with. It will be responsible for handling HTTP requests from users and storing session data in Redis.

**Amazon ElastiCache for Redis:** This is a managed in-memory data store that provides high performance and scalability for session data.

**Database:** This is a persistent data store that can be used to store user data that needs to be persisted beyond the lifetime of a session.

## 3.2 Architecture

The architecture will be as follows:

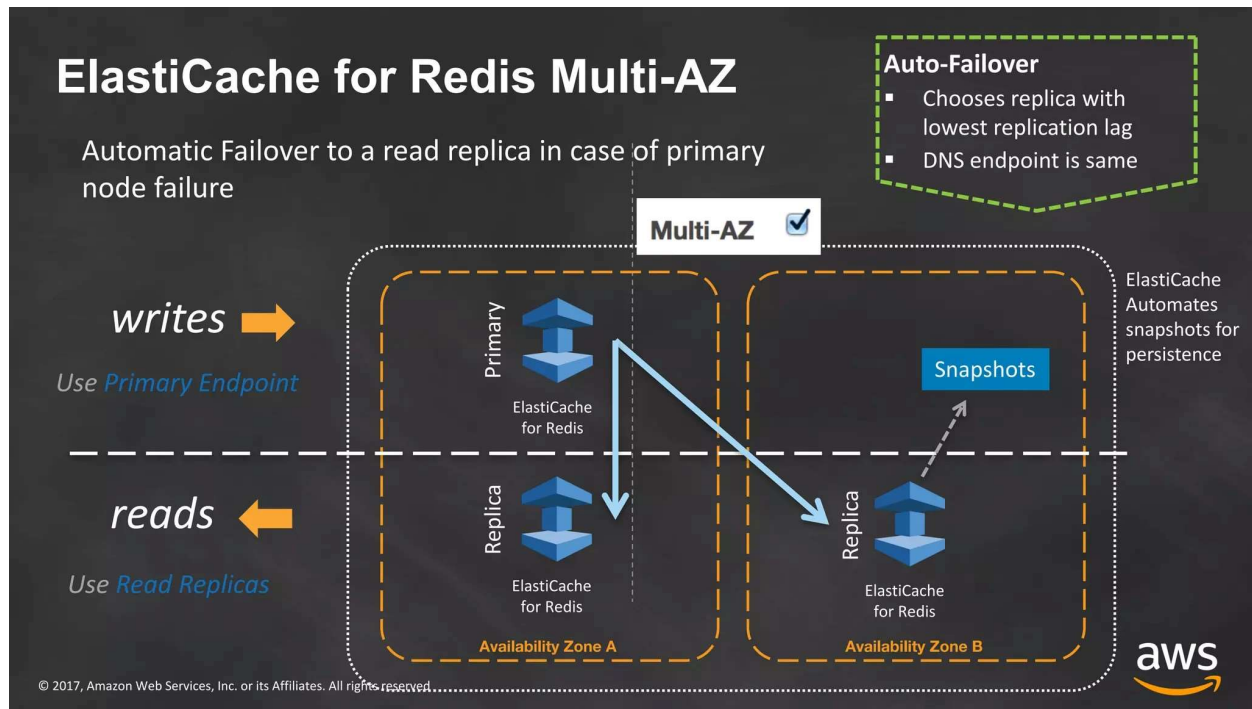


Fig 3.2.1 Architecture for ElastiCache for Redis Multi-AZ

- When a user logs into the application, the web server will create a session for the user and store it in Redis.
- The web server will then use the session ID to access the user's session data in Redis.
- The user's session data will be stored in Redis until it expires.
- When the user logs out of the application, the web server will delete the user's session from Redis.

### Steps to implement the system

- **Create an Amazon ElastiCache for Redis cluster**

In this step, you will create an Amazon ElastiCache for Redis cluster in the AWS region and Availability Zone where your application is running. You will need to choose the node type and number of nodes for your cluster. You can also choose to

enable clustering and replication, which will improve the performance and availability of your cluster.

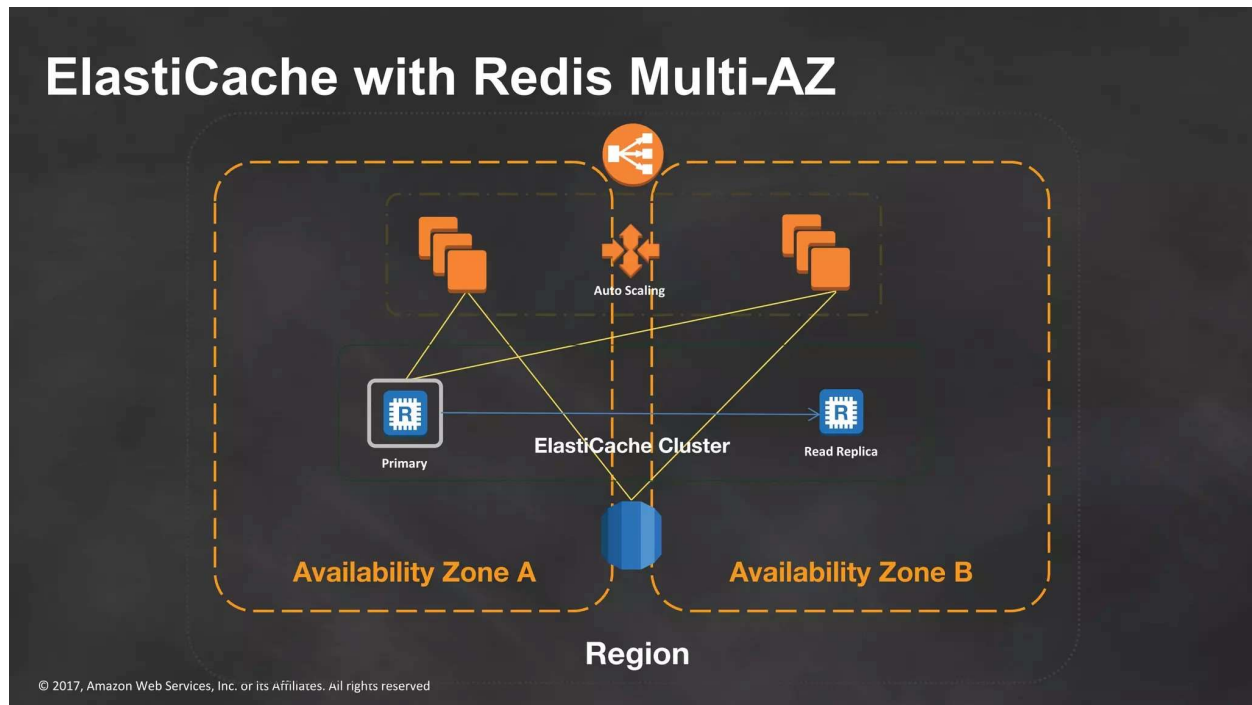


Fig 3.2.2 Multiple Availability Zones in Redis Cluster

- **Configure the web server to use Redis for session management.**

In this step, you will configure your web server to use Redis for session management. This involves setting the `session_save_path` directive in your web server configuration file to the Redis connection string. You will also need to set the `session_cookie_lifetime` directive to the desired session timeout.

- **Update the database to store user data that needs to be persisted beyond the lifetime of a session.**

In this step, you will update your database to store user data that needs to be persisted beyond the lifetime of a session. This data could include user profiles, shopping carts, and order history.

## 4. IMPLEMENTATION

A detailed guide for implementing a fast session store for online applications using Amazon ElastiCache for Redis. It includes several steps, prerequisites, and code examples.

### 4.1 Prerequisites(EC2 instance Creation)

1. Access to an EC2 instance.
2. Ensure the security group of the EC2 instance allows incoming TCP connections on port 5000.

Once you have access to your EC2 instance, run the following commands:

*syntax: shell*

```
$ sudo yum install git
```

```
$ sudo yum install python3
```

```
$ 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
```

```
$ export FLASK_APP=example-1.py
```

```
$ export SECRET_KEY=some_secret_string
```

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



Description	Status Checks	Monitoring	Tags
Instance ID	i-0237d4dbf7933c05e		
Instance state	running		
Instance type	t2.micro		
Public DNS (IPv4)	ec2-54-175-201-152.compute-1.amazonaws.com		
IPv4 Public IP	54.175.201.152		
IPv6 IPs	-		

Fig 4.1.1 Running of EC2 instance showing public DNS

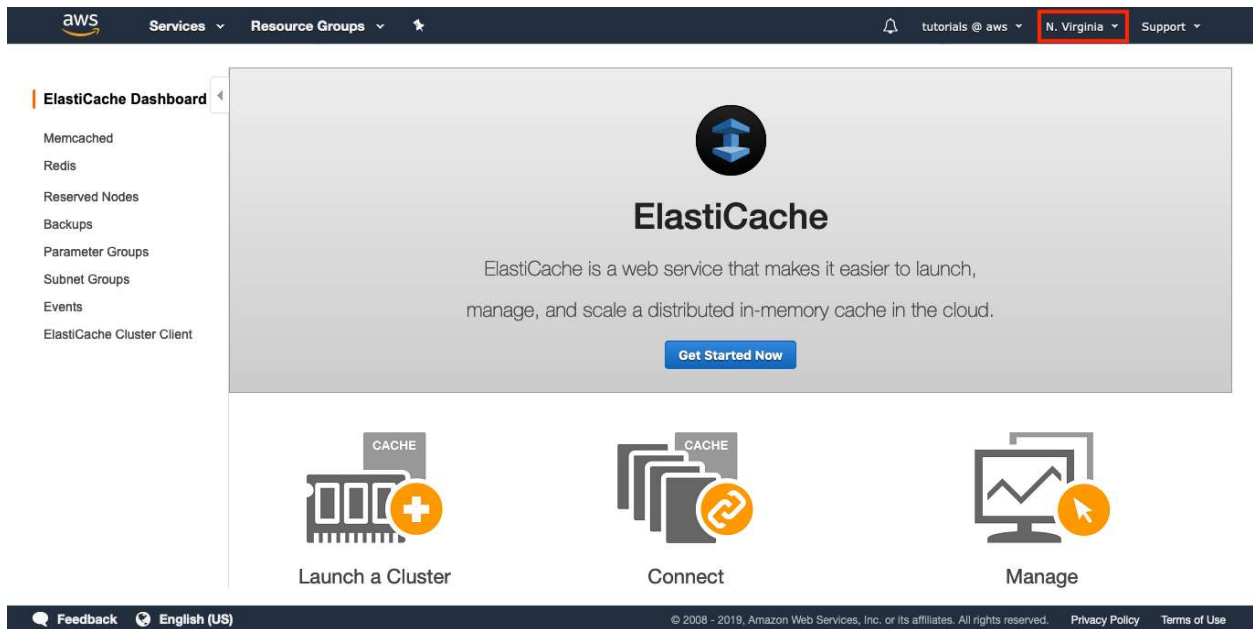
## 4.2 Create a Redis Cluster:

1. Open the Amazon ElastiCache Dashboard and select the desired region.
2. Click "Get Started Now" and select "Redis" as the Cluster engine.
3. Choose a name for the Redis Cluster.
4. Configure the node type, number of replicas, multi-AZ with auto-failover, and preferred availability zones.
5. Set up security group settings to allow incoming TCP connections on port 6379.
6. Review the settings and click "Create."

### Detailed Explanation (Module 1):

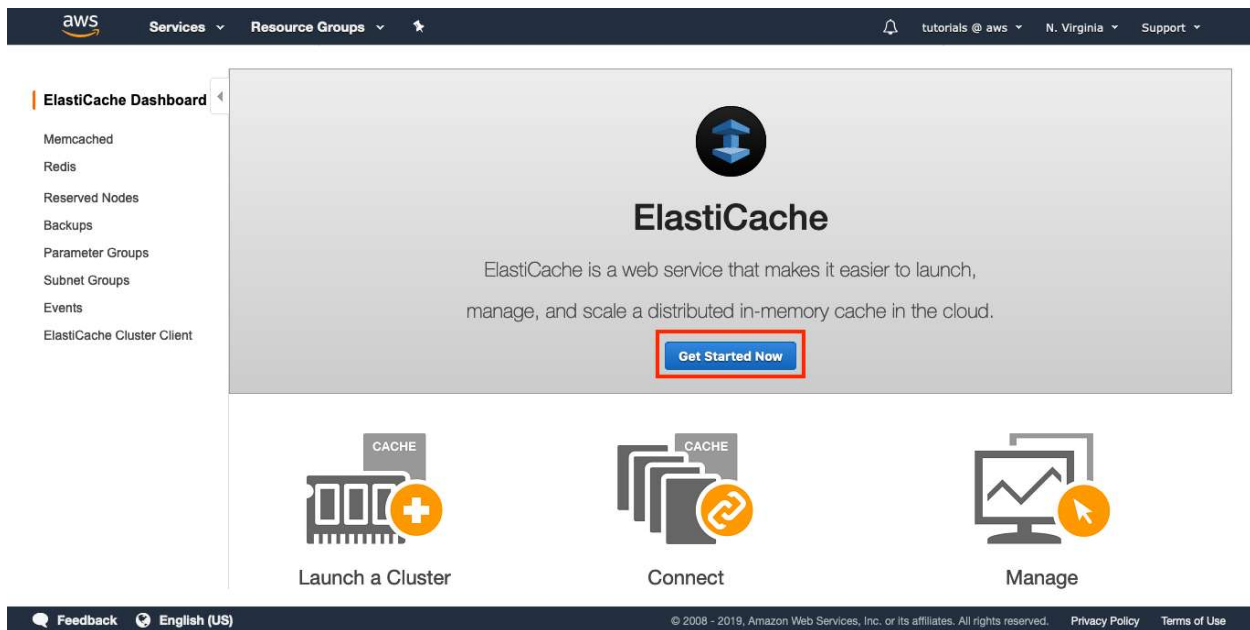
Open the Amazon ElastiCache Dashboard, then:

- 1.1 — On the top right corner, select the region where you want to launch your Redis Cluster.

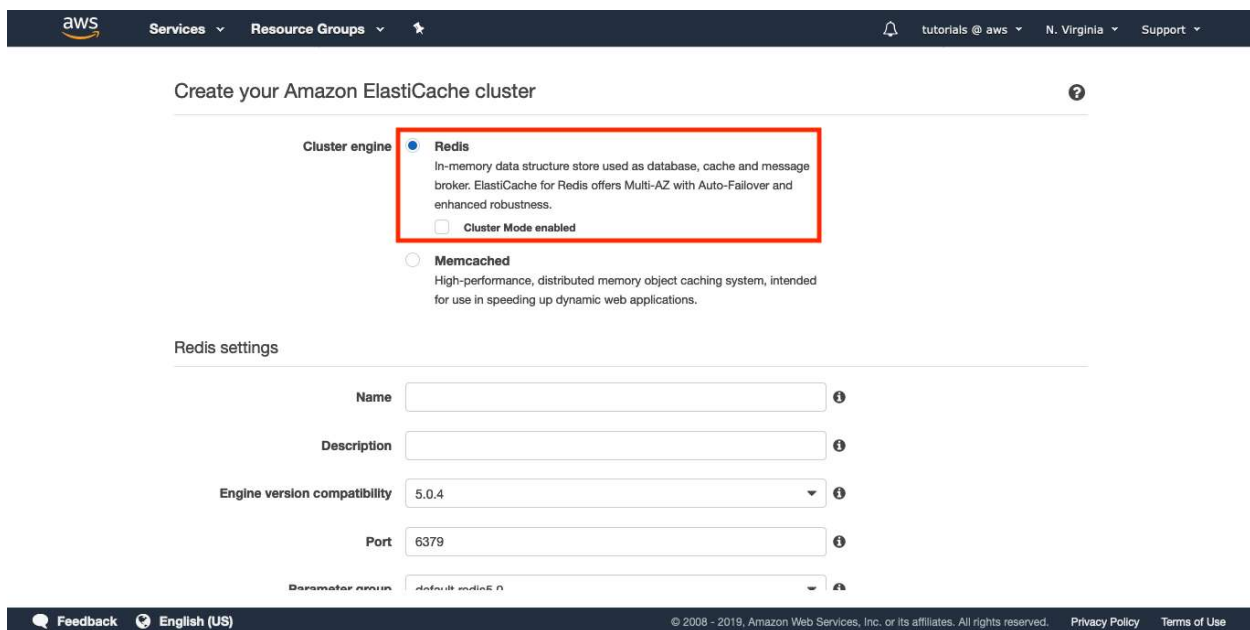


- 1.2 — Click on “Get Started Now”.





1.3 — Select “Redis” as your Cluster engine.



## Redis settings

1.4 — Choose a name for your Redis Cluster, e.g. “elc-tutorial”.

aws Services Resource Groups

tutorials @ aws N. Virginia Support

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

Redis settings

Name elc-tutorial ⓘ

Description ⓘ

Engine version compatibility 5.0.4 ⓘ

Port 6379 ⓘ

Parameter group default.redis5.0 ⓘ

Node type cache.t2.micro (0.5 GiB) ⓘ

Number of replicas 1 ⓘ

Advanced Redis settings

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

1.5 — Change the Node type to cache.t2.micro.

aws Services Resource Groups

tutorials @ aws N. Virginia Support

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

Redis settings

Name elc-tutorial ⓘ

Description ⓘ

Engine version compatibility 5.0.4 ⓘ

Port 6379 ⓘ

Parameter group default.redis5.0 ⓘ

Node type cache.t2.micro (0.5 GiB) ⓘ

Number of replicas 1 ⓘ

Advanced Redis settings

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

That node type is fine for this tutorial, but for a production cluster the size of the node should depend on your workload and you should start with the m5 or r5 instance families.

1.6 — In Number of replicas, select 1.

aws Services Resource Groups

tutorials @ aws N. Virginia Support

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

Redis settings

Name: elc-tutorial ⓘ

Description: ⓘ

Engine version compatibility: 5.0.4 ⓘ

Port: 6379 ⓘ

Parameter group: default.redis5.0 ⓘ

Node type: cache.t2.micro (0.5 GiB) ⓘ

Number of replicas: 1 ⓘ

Advanced Redis settings

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

That read-only replica will allow you to scale your reads. In case of a failure, an automatic failover will be triggered and the replica will take over the role of the master node.

## Advanced Redis settings

1.7 — Check the box for “Multi-AZ with Auto-Failover”.

aws Services Resource Groups

tutorials @ aws N. Virginia Support

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.

Multi-AZ with Auto-Failover ☒ ⓘ

Subnet group: default (vpc-ed72db88) ⓘ

Preferred availability zone(s): ☒ No preference ⓘ  
☐ Select zones

Security

Security groups: redis (sg-09f9faa152d44a8a8) ⓘ

Encryption at-rest: ☐ ⓘ

Encryption in-transit: ☐ ⓘ

Import data to cluster

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

1.8 — Select a Subnet group.

aws Services Resource Groups

tutorials @ aws N. Virginia Support

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.

Multi-AZ with Auto-Failover ☒

Subnet group default (vpc-ed72db88)

Preferred availability zone(s) ☒ No preference ☐ Select zones

Security

Security groups redis (sg-09f9faa152d44a8a8)

Encryption at-rest ☐

Encryption in-transit ☐

Import data to cluster

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

1.9 — For Preferred availability zone(s), select “No preference”.

aws Services Resource Groups

tutorials @ aws N. Virginia Support

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.

Multi-AZ with Auto-Failover ☒

Subnet group default (vpc-ed72db88)

Preferred availability zone(s) ☒ No preference ☐ Select zones

Security

Security groups redis (sg-09f9faa152d44a8a8)

Encryption at-rest ☐

Encryption in-transit ☐

Import data to cluster

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Each node will be created in a different availability zone. This is a best practice for improved reliability.

## Configure the security settings

For this example we won't use encryption, but keep in mind you can configure both encryption for data at-rest and for data in-transit.

## 1.10 — Select a Security group for your Redis Cluster.

The screenshot shows the AWS Management Console interface for configuring a Redis cluster. The top navigation bar includes the AWS logo, 'Services', 'Resource Groups', and user information. The main content area is titled 'Advanced Redis settings' and includes a sub-header: '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.'

Under the 'Advanced Redis settings' section, the following options are visible:

- Multi-AZ with Auto-Failover:** Checked (checkbox).
- Subnet group:** A dropdown menu showing 'default (vpc-ed72db88)'.
- Preferred availability zone(s):** Radio buttons for 'No preference' (selected) and 'Select zones'.

The 'Security' section is expanded, showing the following options:

- Security groups:** A dropdown menu showing 'redis (sg-09f9faa152d44a8a8)' with a red box around it.
- Encryption at-rest:** An unchecked checkbox.
- Encryption in-transit:** An unchecked checkbox.

At the bottom of the 'Security' section, there is a link for 'Import data to cluster'.

This is important: make sure the Security group you select allows incoming TCP connections on port 6379 from your EC2 instance. If that's not the case, you won't be able to connect to your Redis nodes.


### Import data to cluster

For this example, we won't load any seed RDB file so we can skip this configuration step altogether. Just keep in mind that this option is available.

### Configure backups

Daily backups are important for most use cases, and a good recommendation is to enable backups with a retention period that will give you enough time to act in case anything bad happens. For this tutorial, we won't use any backups.

## 1.11 — Uncheck "Enable automatic backups".


Services
Resource Groups
tutorials @ aws
N. Virginia
Support

### Import data to cluster

Seed RDB file S3 location  ⓘ

Use comma to separate multiple paths in the field

### Backup

Enable automatic backups ☐ ⓘ

### Maintenance

Maintenance window ☒ No preference ⓘ  
☐ Specify maintenance window


Topic for SNS notification  ⓘ

Cancel Create

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

## Maintenance settings

1.12 — Specify a maintenance window that suits your needs.


Services
Resource Groups
tutorials @ aws
N. Virginia
Support

### Import data to cluster

Seed RDB file S3 location  ⓘ

Use comma to separate multiple paths in the field

### Backup

Enable automatic backups ☐ ⓘ

### Maintenance

Maintenance window ☒ No preference ⓘ  
☐ Specify maintenance window

Topic for SNS notification  ⓘ

Cancel Create

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

## Review and create

After a review of all the fields in the form, click “Create”.

1.13 — Click on “Create”.

Import data to cluster

Seed RDB file S3 location  ⓘ  
Use comma to separate multiple paths in the field

Backup

Enable automatic backups ☐ ⓘ

Maintenance

Maintenance window ☒ No preference ⓘ  
☐ Specify maintenance window

Topic for SNS notification  ⓘ

[Cancel](#) [Create](#)

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

A Redis Cluster will get initialized and once it becomes “available” you will be able to continue the next step.

### 4.3 Session Caching with Redis:

1. Implement a basic Flask application that handles user login and logout.
2. Configure environment variables, export them, and run the Flask application.
3. Test your connection to the Redis cluster by pinging the Redis node.
4. Implement session caching using Redis hashes to store session data.
5. Set a Time to Live (TTL) for the session data to automatically expire after a certain time.

### Detailed Explanation (Module 2):

#### Basic Behaviour

This application uses a cookie to store a username. When you first visit the application, there’s no username stored in the session object. When you go to “/login” and provide a username, the value is stored in a signed cookie and you get redirected to the homepage. Then, if you go to “/logout”, the username is removed from the session and you are back where you started.

## Source code for Flask Application

```
import os

from flask import Flask, session, redirect, escape, request

app = Flask(__name__)

app.secret_key = os.environ.get('SECRET_KEY', default=None)

@app.route('/')

def index():

    if 'username' in session:

        return 'Logged in as %s' % escape(session['username'])

    return 'You are not logged in'

@app.route('/login', methods=['GET', 'POST'])

def login():

    if request.method == 'POST':

        session['username'] = request.form['username']

        return redirect('/')

    return '''

    <form method="post">

    <p><input type="text" name="username">

    <p><input type="submit" value="Login">

    </form>

    '''

@app.route('/logout')

def logout():
```

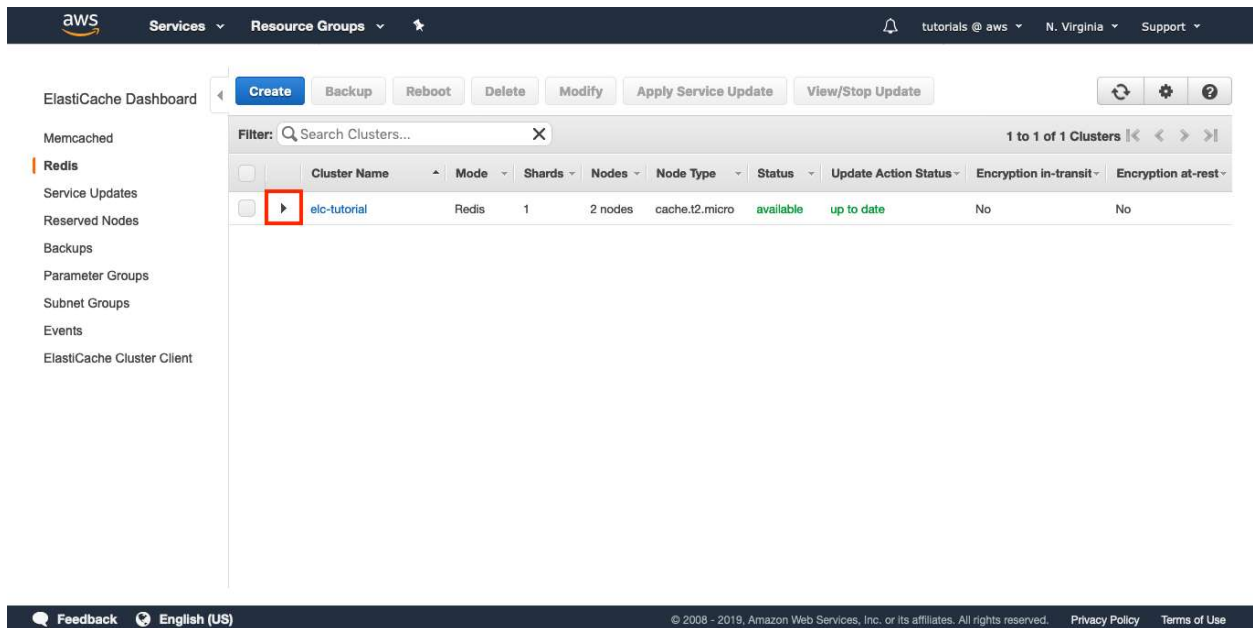


```
session.pop('username', None)
```

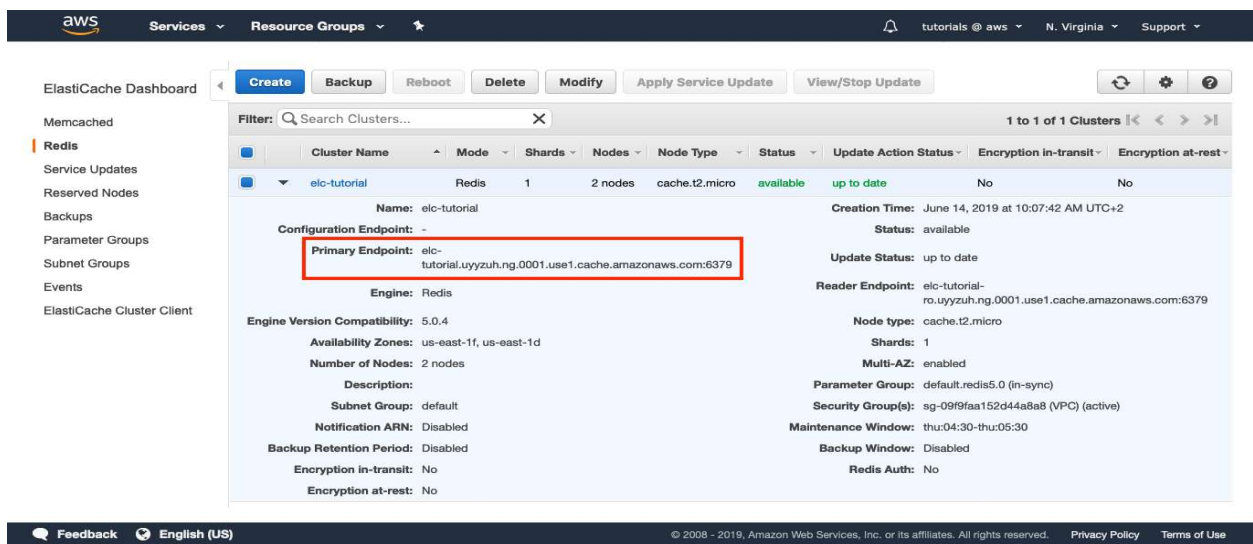
```
return redirect('/')
```

## Test your connection to Redis

2.1 — Click on the arrow to display the Redis Cluster details.



2.2 — Copy the Primary Endpoint.



2.3 — In your EC2 instance, configure the REDIS\_URL environment variable.

*syntax: shell*

```
$ export REDIS_URL="redis://your_redis_endpoint:6379"
```

2.4 — From your EC2 instance, enter the Python interactive interpreter:

*syntax: shell*

```
$ python
```

2.5 — Now run these commands to test the connection to your Redis node.

*syntax: python*

```
>>> import redis
```

```
>>> client = redis.Redis.from_url('redis://your_redis_endpoint:6379')
```

```
>>> client.ping()
```

```
**True**
```

### **Session caching with Redis hashes**

Introduce some small changes in code that will allow you to store session data in Redis.

```
import os
```

```
import redis
```

```
from flask import Flask, session, redirect, escape, request
```

```
# Configure the application name with the FLASK_APP environment variable.
```

```
app = Flask(__name__)
```

```
# Configure the secret_key with the SECRET_KEY environment variable.
```

```
app.secret_key = os.environ.get('SECRET_KEY', default=None)
```

```
# Configure the REDIS_URL constant with the REDIS_URL environment variable.
```

```
REDIS_URL = os.environ.get('REDIS_URL')
```

```
class SessionStore:
```

```
    """Store session data in Redis."""
```

```
    def __init__(self, token, url='redis://localhost:6379', ttl=10):
```

```
        self.token = token
```

```
        self.redis = redis.Redis.from_url(url)
```

```
        self.ttl = ttl
```

```
    def set(self, key, value):
```

```
        self.refresh()
```

```
        return self.redis.hset(self.token, key, value)
```

```
    def get(self, key, value):
```

```
        self.refresh()
```

```
        return self.redis.hget(self.token, key)
```

```
    def incr(self, key):
```

```
        self.refresh()
```

```
        return self.redis.hincrby(self.token, key, 1)
```

```
    def refresh(self):
```

```
        self.redis.expire(self.token, self.ttl)
```

```
@app.route('/')
```

```
def index():
```

```
    if 'username' in session:
```

```
        username = escape(session['username'])
```

```
        store = SessionStore(username, REDIS_URL)
```

```
        visits = store.incr('visits')
```

```
        return ""
```

```

        Logged in as {0}.<br>
        Visits: {1}
    """.format(username, visits)
    return 'You are not logged in'

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect('/')
    return """
    <form method="post">
    <p><input type="text" name="username">
    <p><input type="submit" value="Login">
    </form>
    """

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect('/')

```

Now Redis is imported and initialized, and index takes care of incrementing a counter and displaying it to the user.

Note that the Redis command you used for incrementing the counter is HINCRBY. That's because you are using a Redis hash for each user's session.

## Expire sessions

Now you will set a Time to Live (TTL) to the server-side session. It's just one extra line of code, and this snippet will only include the / route.

```

@app.route('/')
def index():
    if 'username' in session:
        username = escape(session['username'])
        visits = store.hincrby(username, 'visits', 1)

        # BEGIN NEW CODE #
        store.expire(username, 10)
        # END NEW CODE #

    return """
        Logged in as {0}.<br>
        Visits: {1}

        """.format(username, visits)

    return 'You are not logged in'

```

The key where the session is stored will expire in 10 seconds. It's a very short TTL, useful only for this example. If you run the application now, you will see how visiting the index updates the Visits counter, but if you let 10 seconds elapse without visiting that page, on the next refresh the counter will be back at 1.

#### 4.4 Cleanup :

1. Go to the Amazon ElastiCache Dashboard.

The screenshot shows the AWS ElastiCache Dashboard. The left sidebar has a menu with 'Redis' highlighted in a red box. The main area displays a table of clusters. The table has columns: Cluster Name, Mode, Shards, Nodes, Node Type, Status, Update Action Status, Encryption in-transit, and Encryption at-rest. A single cluster named 'elc-tutorial' is listed with Mode 'Redis', 1 Shard, 2 nodes, Node Type 'cache.t2.micro', Status 'available', and Update Action Status 'up to date'. The bottom of the dashboard includes a footer with 'Feedback', 'English (US)', and copyright information.

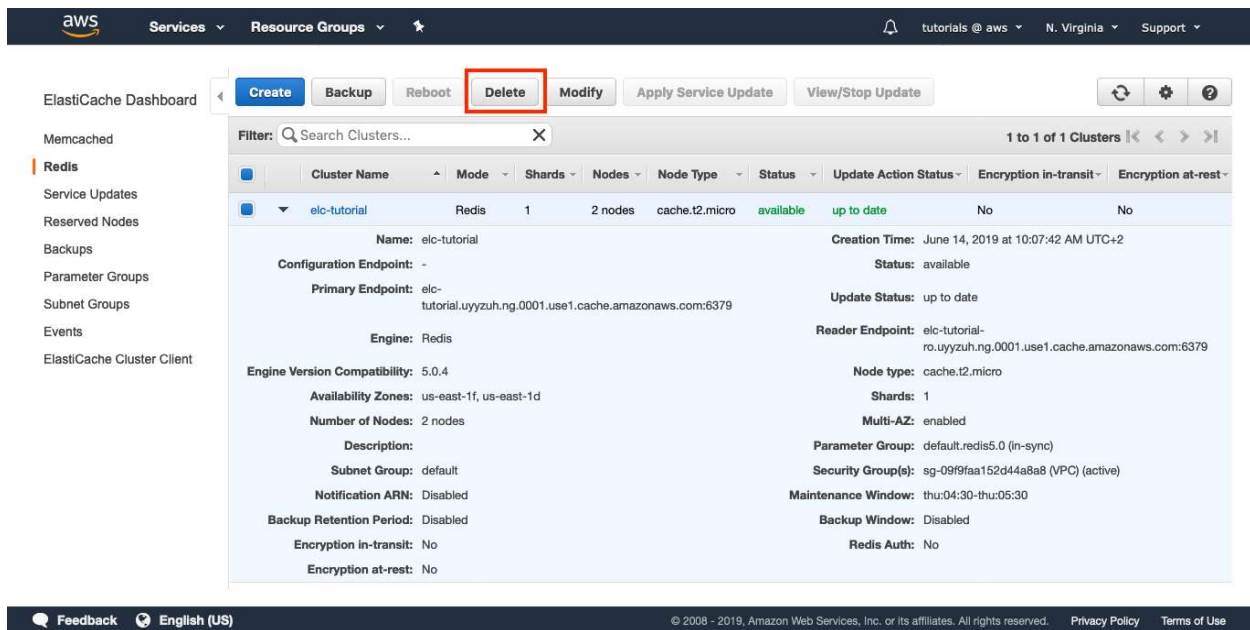
Cluster Name	Mode	Shards	Nodes	Node Type	Status	Update Action Status	Encryption in-transit	Encryption at-rest
elc-tutorial	Redis	1	2 nodes	cache.t2.micro	available	up to date	No	No

2. Select the Redis Cluster created earlier.

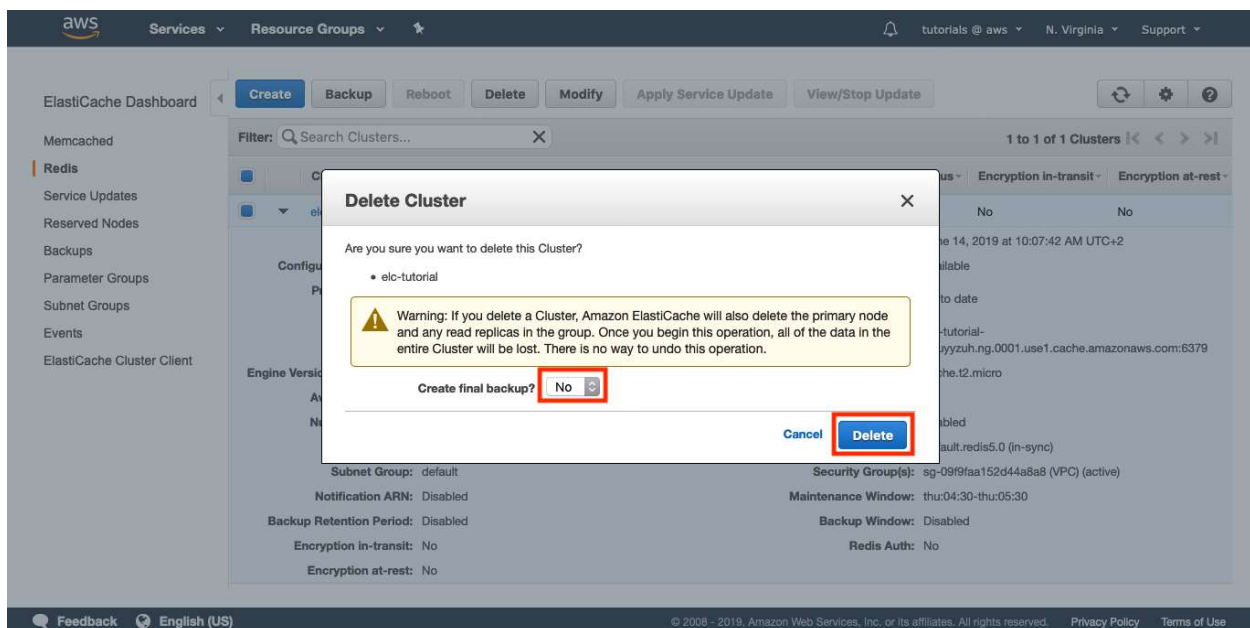
This screenshot is identical to the one above, but with a red box around the checkbox in the first column of the cluster table, indicating that the 'elc-tutorial' cluster is selected.

Cluster Name	Mode	Shards	Nodes	Node Type	Status	Update Action Status	Encryption in-transit	Encryption at-rest
elc-tutorial	Redis	1	2 nodes	cache.t2.micro	available	up to date	No	No

3. Click "Delete" and confirm.



4. Optionally, choose whether to create a final backup.



This implementation covers setting up a Redis Cluster, implementing session caching with Redis for a Flask application, testing the connection to Redis, and cleaning up by deleting the Redis Cluster when no longer needed.

## 5.Results

Implementing a fast session store for online applications using Amazon ElastiCache for Redis yields improved performance and scalability.

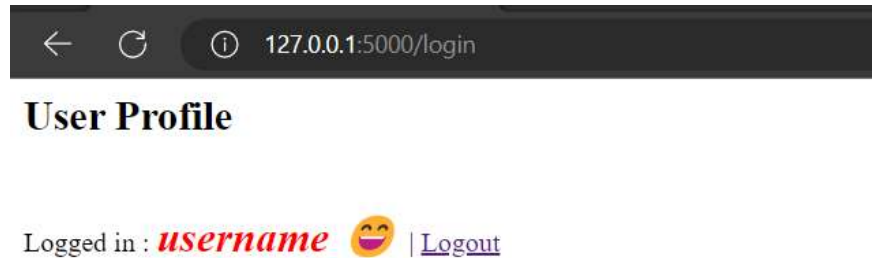


Fig 5.1 Username display on screen

It enables sub-millisecond latency, better handling of traffic spikes, and automatic failover for high availability. Redis's in-memory nature accelerates data access, while features like encryption and data persistence enhance security. The implementation simplifies session management, reduces database load, and enhances user experience by providing rapid access to session data. Overall, ElastiCache for Redis optimizes application performance, making it a valuable addition to modern web applications.



## **6.Conclusion**

In conclusion, implementing a fast session store for online applications using Amazon ElastiCache for Redis offers significant advantages in terms of performance, scalability, and reliability. By leveraging the in-memory capabilities of Redis, applications can achieve sub-millisecond response times, enhancing user experience and enabling real-time interactions. The use of ElastiCache for Redis further enhances these benefits by providing managed services that simplify deployment, maintenance, and scaling. The integration of Redis as a session store reduces the load on primary databases, improves application responsiveness, and ensures seamless session management. With features like automatic failover, data persistence, and security measures, ElastiCache for Redis offers a robust solution for building high-performance, low-latency online applications. Overall, this implementation empowers developers to create efficient, responsive, and user-friendly web applications, making it a valuable tool in the modern digital landscape.