



aws INNOVATE

DATA EDITION

23 August, 2022

Build high-performance and resilient real-time applications with Amazon ElastiCache and MemoryDB for Redis

Orlando Andico

Senior Solutions Architect
Amazon Web Services

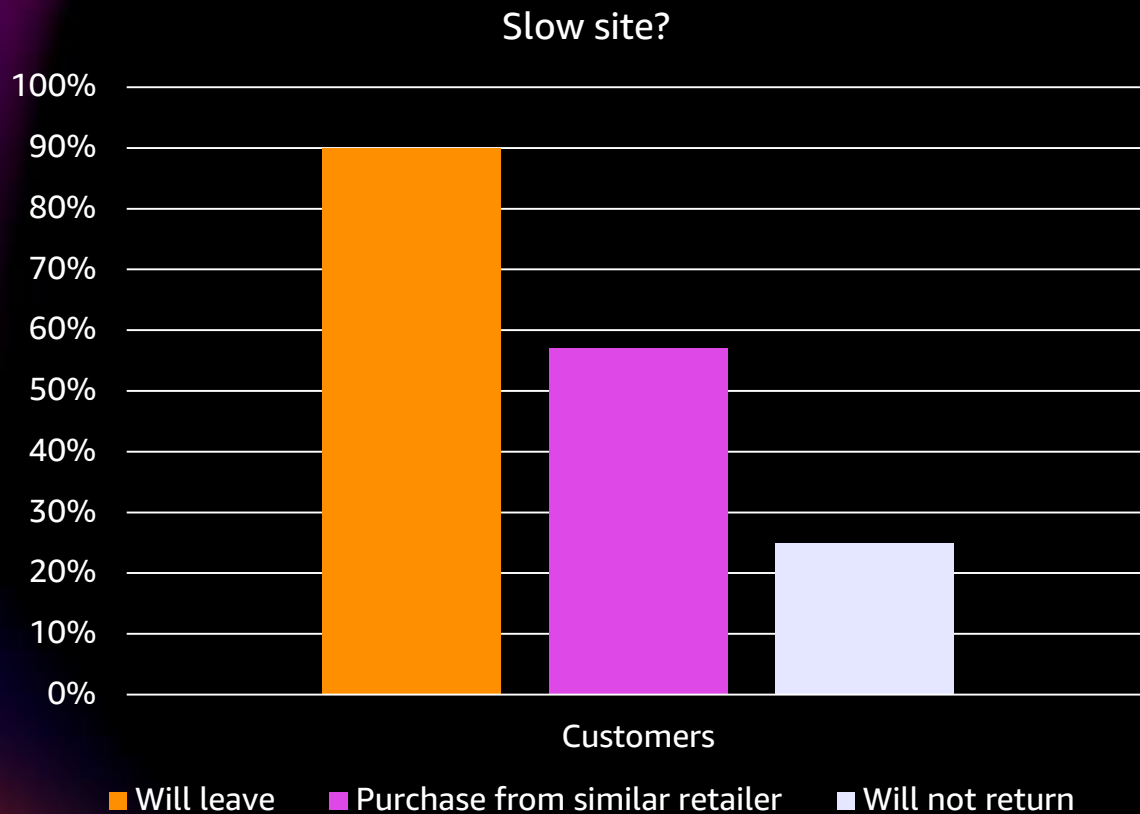


Agenda

- Why performance matters
- Why Redis
 - Amazon ElastiCache for Redis
 - Amazon MemoryDB for Redis
- Demo

Why performance matters

Why performance matters



"A 100-millisecond delay in website load time can hurt conversion rates by 7 percent"

"A two-second delay in web page load time increases bounce rate by 103 percent"

– 2017 Akamai Study

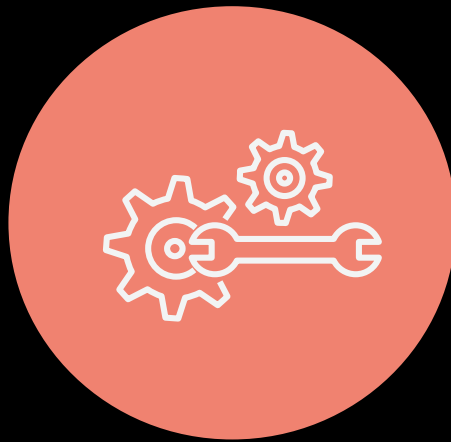
Current customer challenges

Lack of flexibility, ability to perform ultra-fast processing of massive amounts of data, and high cost

Customers need:



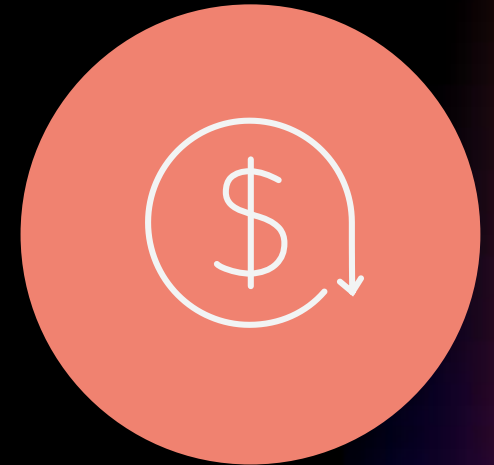
High-performance
with durability and
scalability



Developer friendly,
easy-of-use



Massive parallel
processing of Data



Lower Total cost
of ownership (TCO)

Use cases for in-memory datastores



Caching



Real-time
analytics



Gaming
leaderboards



Geospatial



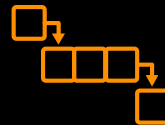
Media
streaming



Session
store



Chat apps



Message
queues










Machine
learning

Why Redis?

Why Redis?

#1 in-memory datastore, #1 key-value database, most loved database



Rank			DBMS	Database Model	Score		
Jul 2022	Jun 2022	Jul 2021			Jul 2022	Jun 2022	Jul 2021
1.	1.	1.	Redis 	Key-value, Multi-model 	173.62	-1.69	+5.32
2.	2.	2.	Amazon DynamoDB 	Multi-model 	83.94	+0.05	+8.74
3.	3.	3.	Microsoft Azure Cosmos DB 	Multi-model 	40.08	-0.90	+3.38
4.	4.	4.	Memcached	Key-value	24.12	-0.44	-1.22
5.	5.	5.	etcd	Key-value	10.43	-0.37	+0.33
6.	6.	6.	Hazelcast	Key-value, Multi-model 	10.15	-0.23	+0.95

* <https://db-engines.com/en/ranking/key-value+store>

Developers love it because:

1. Blazing fast
2. Versatile
3. Feature rich and easy to use
4. Open source + cloud

Self-managing Redis is challenging and time-consuming



Difficult to manage

Manage server provisioning, software patching, setup, configuration, and backups

Time better spent on building compelling customer experiences



Hard to make highly available

Need to implement fast error detection and remediation



Difficult to scale

Online scaling can be error prone, replication performance needs to be monitored, inelastic

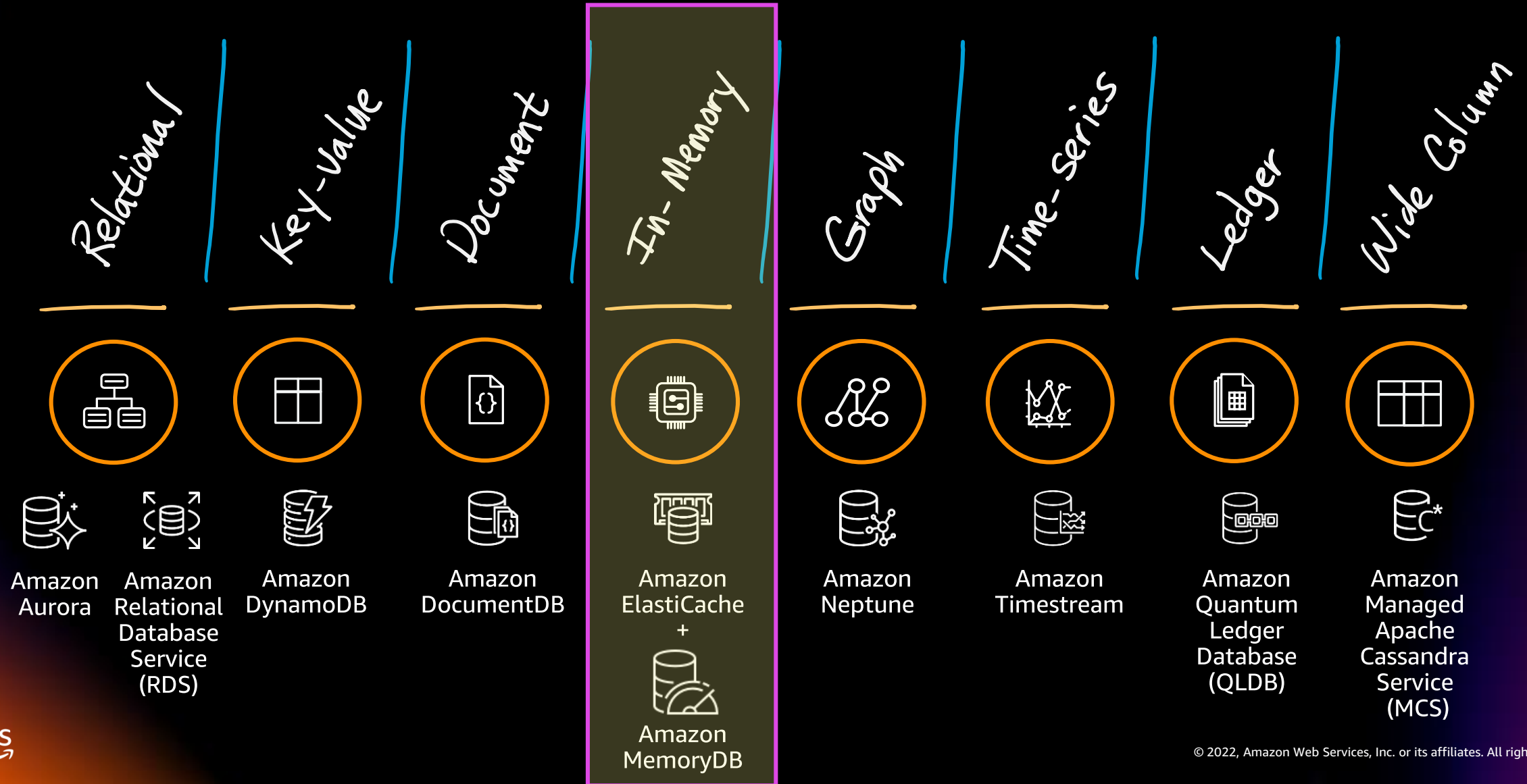


Expensive

Invest in people, processes, hardware, and software

Real-time demands often have huge spikes

Purpose-built databases



Amazon ElastiCache and Amazon MemoryDB for Redis

Amazon ElastiCache – Fully Managed Service

Redis &
Memcached compatible



Fully compatible with
open source Redis
and Memcached

Extreme
performance



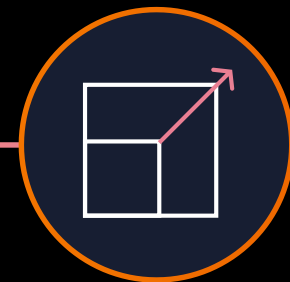
In-memory data store
and cache for microsecond
response times

Secure
and reliable



Network isolation, encryption
at rest/transit, HIPAA, PCI,
FedRAMP, multi AZ, global
datastore, and automatic
failover

Easily scales to
massive workloads



Scale reads and writes
with sharding
and replicas

Amazon ElastiCache

Built on the full power of AWS



- Available in all AWS regions
- Multi-AZ auto failover
- Global datastore - cross region replication
- Up to 250 nodes per cluster (500 soon)
- Up to 170TB database
- ~ 50 million reads and ~10 million writes/sec
- T2, M5, R5 node support
- Scheduled snapshot support
- Scale out/in (sharded configuration)
- Scale up/down (all configurations)



Amazon MemoryDB for Redis

Redis-compatible, durable, in-memory database service that delivers ultra-fast performance

Ultra-fast performance

Microsecond read and single-digit millisecond write latencies with millions of TPS

Redis compatibility

Flexible and friendly Redis APIs and data structures

Durability and high availability

Multi-AZ transactional log for durability and high availability

Fully managed

AWS-managed hardware and software setup, configuration, monitoring, and snapshots

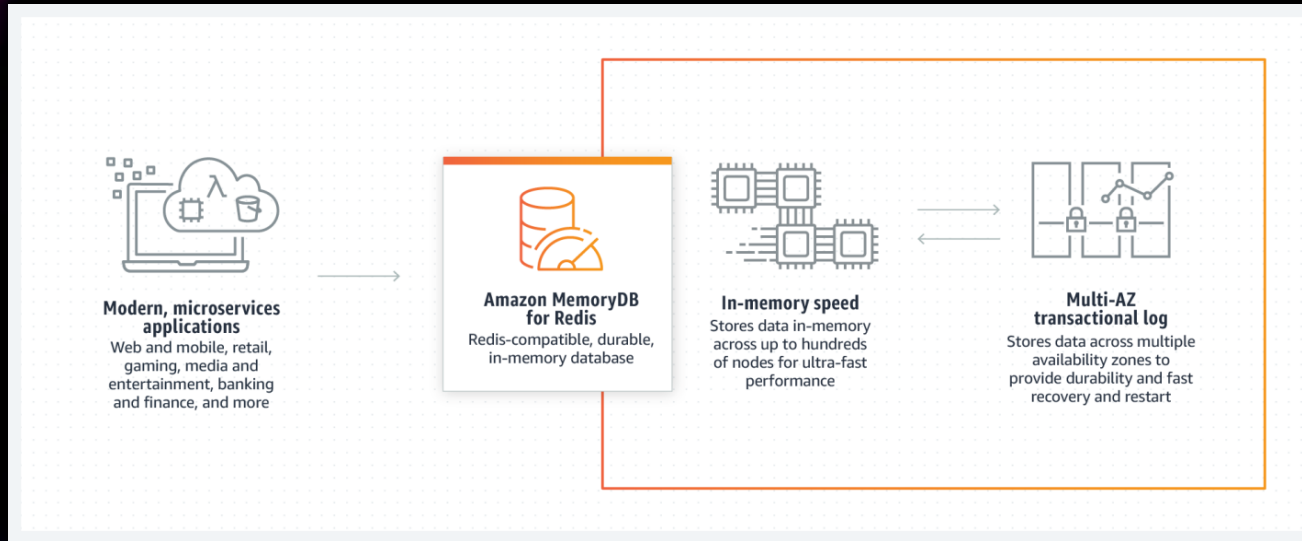
High Scalability

More than 100 TB of storage per cluster (with 1 replica per shard)

Security

Amazon VPC, encryption at-rest and in-transit, Access Control List (ACL)

Fast: In-memory performance with Multi-AZ Durability



- **Database design**

- All data in memory
- All writes committed to Multi-AZ transactional log
- Dynamic sharding and replication

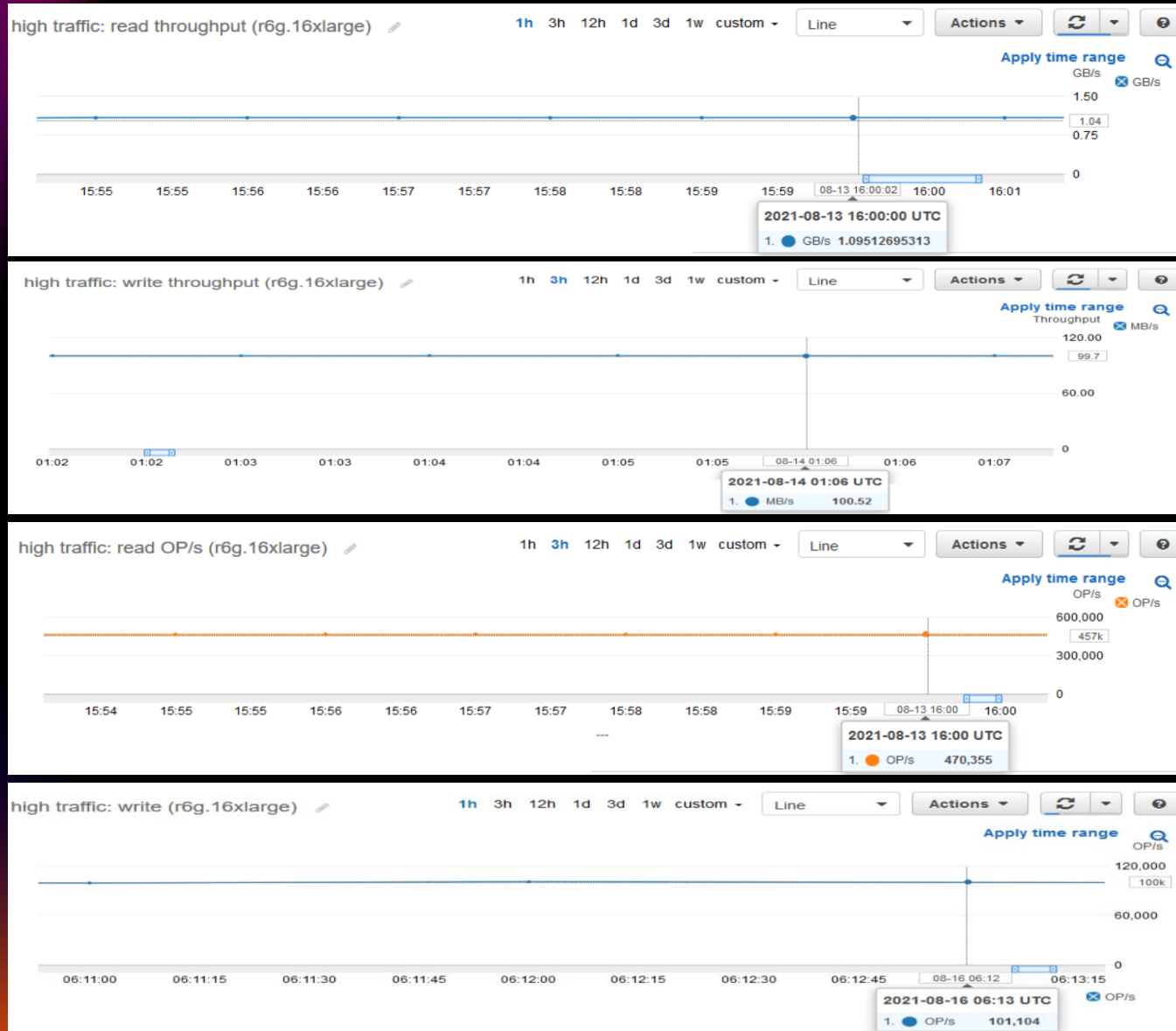
- **Low latency**

- Microsecond reads
- Single-digit millisecond writes

- **High throughput**

- Sharded parallelism and pipelined writes
- Millions of TPS

Ultra-fast performance



- Access data with microsecond read and single-digit millisecond write latency
- Amazon MemoryDB clusters can handle more than 13 trillion requests per day to support peaks of more than 160 million requests per second
- Up to 390K read and 100K write requests per second, up to 1.3 GB/s read and 100 MB/s write throughput* per node

* - Based on internal testing on R6g 16xlarge read-only and write-only workloads

Durability and high availability

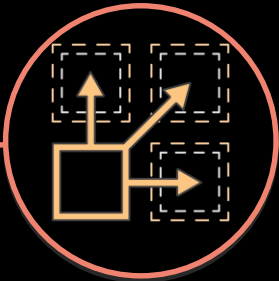


- Amazon MemoryDB stores your data using a Multi-AZ transactional log for data durability, consistency, and recoverability
- Supports high availability with data replicated automatically across multiple AZs
- Create a MemoryDB cluster with Multi-AZ availability with up to five replicas in different AZs

Scalability

- Scale your cluster horizontally by adding/removing shards or vertically by moving to larger/smaller node types.
- Each cluster can have up to 500 shards. Each shard has one primary node and up to five replica nodes.
- You can scale to more than 100 TB of storage per cluster (with 1 replica per shard)
- Your cluster continues to stay online and support read and write operations during a resizing operation.

**Scale out
in minutes**



Scale up to 500 shards

**Scale up
in minutes**



Scale from
13 to 419 GiB of RAM
per node

**Scale reads in
minutes**



Scale up to 5 replicas

When to use Amazon MemoryDB versus Amazon ElastiCache for Redis?

Amazon MemoryDB for Redis use cases

- Applications requiring a durable database that provides ultra-fast performance
- Applications using Redis's APIs and data structures but looking for durability to avoid the risk of data loss
- Applications using a cache and a database for low latency and looking to simplify architecture for saving costs

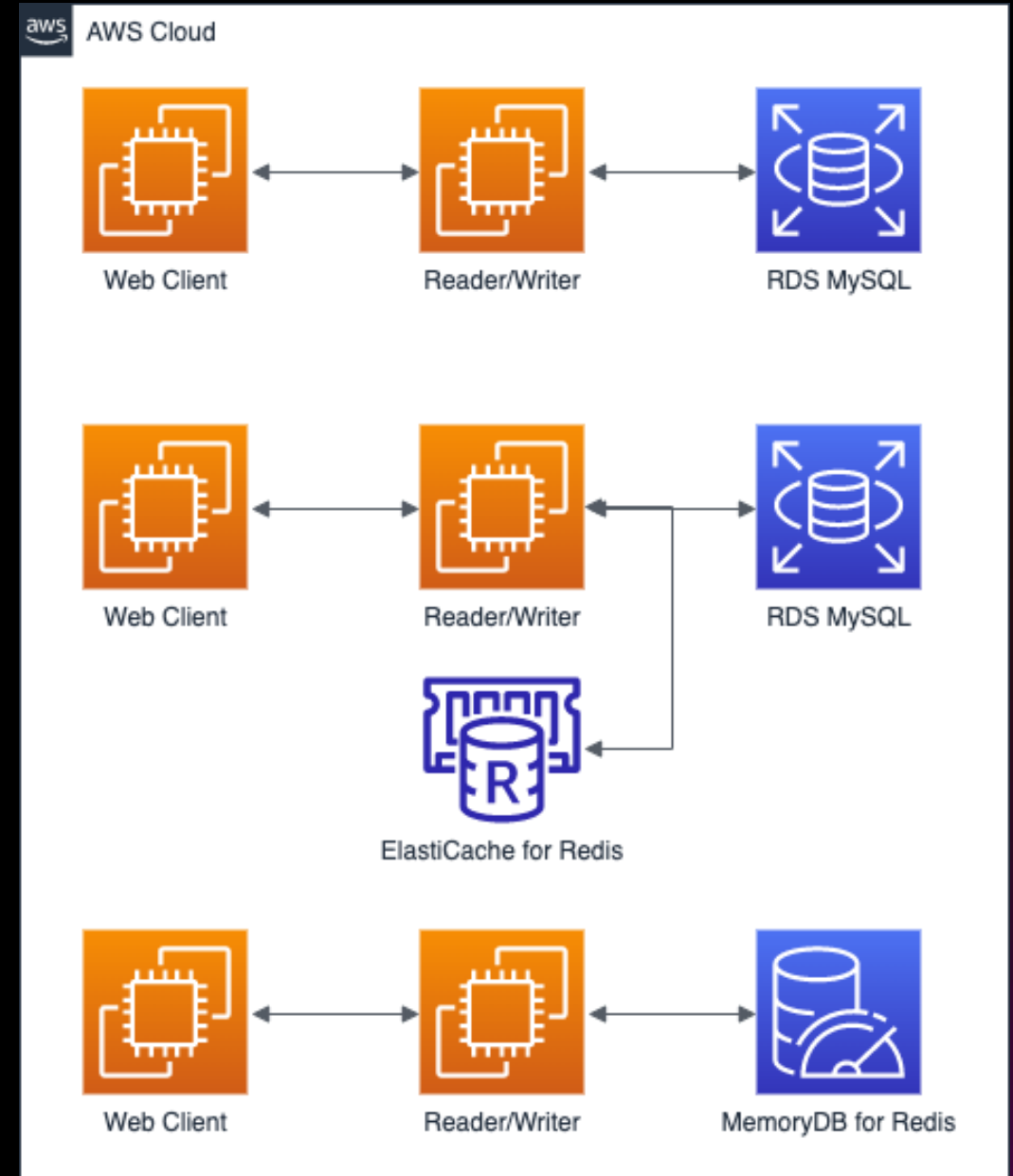
Amazon ElastiCache for Redis use cases

- For caching workloads where you want to accelerate data access with your existing primary database
- If you want to use the Redis data structures and APIs to access data stored in a primary database or data store

Demo

Demo scenarios

- Traditional application writing to MySQL and reading from MySQL
- Traditional application writing to MySQL but using Amazon ElastiCache Redis as a lazy-loading read cache from MySQL
- Application using Amazon MemoryDB for Redis as a durable system of record



Scenario 1: Writing to MySQL

```
# Connect to the database
connection = pymysql.connect(host='[REDACTED].ap-southeast-1.rds.amazonaws.
com',
    user='orly',
    password='[REDACTED]',
    database='telemetry',
    cursorclass=pymysql.cursors.DictCursor)

cursor = connection.cursor()

for batches in range(10000):
    t0 = time.time()
    for rows in range(batchsize):
        car_id=randint(100000000, 999999999)
        rpm=randint(750, 4000)
        water_temp=randint(60, 99)
        oil_temp=randint(80, 120)
        oil_pressure=randint(30, 150)
        sql = "insert into cars(car_id,rpm,water_temp,oil_temp,oil_pressure) values(%s,%s,%s,
        %s,%s) on duplicate key update rpm=%s, water_temp=%s, oil_temp=%s, oil_pressure=%s"
        try:
            cursor.execute(sql,(car_id, rpm, water_temp, oil_temp, oil_pressure, rpm,
            water_temp, oil_temp, oil_pressure))
        except:
            print("Ouch..")

    connection.commit()

    t1 = time.time()
    rps = batchsize / (t1 - t0)
    print("Pid %d Batch %d Rows/sec %5.2f" % (pid, batches, rps))
```

Scenario 1: Reading from MySQL

```
# Connect to the database
connection = pymysql.connect(host='[REDACTED].ap-southeast-1.rds.amazonaws.com',
                             user='orly',
                             password='[REDACTED]',
                             database='telemetry',
                             cursorclass=pymysql.cursors.DictCursor)

cursor = connection.cursor()

# keep trying until we get a row
rows = 0
while (rows == 0):
    # select a random Car ID in a small range so we know it will cache properly
    car_id=randint(50000000, 50001000)

    sql = "select avg(rpm) rpm, avg(water_temp) water_temp, avg(oil_temp) oil_temp, avg(oil_pressur
    rows = cursor.execute(sql, (car_id, car_id) )

d = cursor.fetchone()
%>

<table width="80%" align="center">
<tr><td>Metric</td><td>Value</td></tr>
<tr><td>Car ID</td><td><%= car_id %></td></tr>
<tr><td>RPM</td><td><%= d['rpm'] %></td></tr>
<tr><td>Water Temperature</td><td><%= d['water_temp'] %></td></tr>
<tr><td>Oil Temperature</td><td><%= d['oil_temp'] %></td></tr>
<tr><td>Oil Pressure</td><td><%= d['oil_pressure'] %></td></tr>
</table>
</body>
</html>
```


Scenario 2: Reading from MySQL with Caching

```
# time to live in cache
TTL = 120

# Connect to Redis first and look for the Car ID
host = "██████████.clustercfg.apse1.cache.amazonaws.com"

startup_node = [ {"host": host, "port": "6379" } ]
r = r.RedisCluster(
    startup_nodes=startup_node,
    decode_responses=True,
    skip_full_coverage_check=True,
    ssl=False
)

# select a random Car ID in a small range so we know it will cache properly
car_id=randint(50000000, 50001000)

# try to get the value from the Redis cluster
payload = r.get(car_id)
from_cache="Yes"

# if there is no payload, fetch from database and cache it
if (payload == None):
    from_cache="No"
    # print("%s not found in cache" % car_id)
    # Connect to the database
    connection = pymysql.connect(host='██████████.ap-southeast-1.rds.amazonaws.c
        user='orly',
        password='██████████',
        database='telemetry',
        cursorclass=pymysql.cursors.DictCursor)
```

Scenario 3: Writing to Amazon MemoryDB

```
# Connect to MemoryDB
host = "██████████.clustercfg.memorydb.ap-southeast-1.amazonaws.com"

startup_node = [ {"host": host, "port": "6379" }]
r = rd.RedisCluster(
    startup_nodes=startup_node,
    decode_responses=True,
    skip_full_coverage_check=True,
    ssl=False
)

for batches in range(10000):
    t0 = time.time()
    for rows in range(batchsize):
        # generate some random car values
        car_id=randint(50000000, 50050000)
        rpm=randint(750, 4000)
        water_temp=randint(60, 99)
        oil_temp=randint(80, 120)
        oil_pressure=randint(30, 150)

        # store it in the cache
        payload = "%s:%s:%s:%s:%s" % (car_id, rpm, water_temp, oil_temp, oil_pressure)
        r.set(car_id, payload)

    t1 = time.time()
    rps = batchsize / (t1 - t0)
    print("Pid %d Batch %d Rows/sec %5.2f" % (pid, batches, rps))

r.connection_pool.disconnect()
```

Scenario 3: Reading from Amazon MemoryDB

```
import pymysql.cursors
from random import seed
from random import randint
import time
import rediscluster as rd
import json

# Connect to MemoryDB and look for the Car ID
host = "██████████.clustercfg.memorydb.ap-southeast-1.amazonaws.com"

startup_node = [ {"host": host, "port": "6379" }]
r = rd.RedisCluster(
    startup_nodes=startup_node,
    decode_responses=True,
    skip_full_coverage_check=True,
    ssl=False
)

# select a random Car ID in a small range so we know it will be there
car_id=randint(50000000, 50050000)

# try to get the value from the Redis cluster
payload = r.get(car_id)
from_cache="Yes"

# if there is no payload, return a fake value (this shouldn't happen!)
if (payload == None):
    from_cache="No"
    payload = "%s:%s:%s:%s:%s" % (car_id, 0, 0, 0, 0)

m = payload.split(":", 5)
```

Visit the AWS Data resource hub

A modern data strategy can help you manage, act on, and react to your data so you can make better decisions, respond faster, and uncover new opportunities. Dive deeper with these resources today.

- Harness data to reinvent your organization
- In unpredictable times, a data strategy is key
- Make data a strategic asset
- Rewiring your culture to be data-driven
- Put your data to work with a modern analytics approach
- ... and more!



<https://tinyurl.com/data-hub-aws>

[Visit resource hub](#)

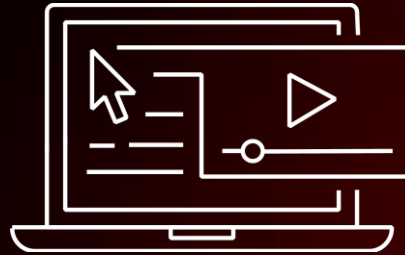
AWS Training and Certification for Data and Analytics



AWS Data & Analytics FREE Training Resources

Discover how to harness data, one of the world's most valuable resources, and innovate at scale.

<https://bit.ly/3Ntlhy7>



AWS Data Analytics Learning Plan

This learning plan expose you to the fastest way to get answers from all your data to all your users. It can also help prepare you for the AWS Certified Data Analytics - Specialty certification exam.

<https://bit.ly/3wBVjD1>



AWS Certified Data Analytics - Specialty

Earning AWS Certified Data Analytics – Specialty validates expertise in using AWS data lakes and analytics services.

<https://go.aws/3lwF0RR>

Thank you for attending AWS Innovate – Data Edition

We hope you found it interesting! A kind reminder to **complete the survey**.
Let us know what you thought of today's event and how we can improve the event experience for you in the future.



aws-apj-marketing@amazon.com



twitter.com/AWSCloud



facebook.com/AmazonWebServices



youtube.com/user/AmazonWebServices



slideshare.net/AmazonWebServices



twitch.tv/aws

Thank you!