Maximizing Speed, Costs and UX with AWS ElastiCache Serverless

Indika Wimalasuriya

DevOps 2024

Agenda

- Importance of Performance
- Challenges in Traditional (Server-Based) ElastiCache
- Overview and Capabilities of ElastiCache Serverless
- Implementation Overview
- Anti-Patterns to Avoid and Best Practices

System performance is corelated with revenue!

- **Walmart found that for every 1 second improvement in page load time, conversions increased by 2%**
- **COOK increased conversions by 7% by reducing page load time by 0.85 seconds**
- **Mobify found that each 100ms improvement in their homepage's load time resulted in a 1.11% increase in conversion**
- **Study by HubSpot found that even a few milliseconds can significantly impact the user experience (UX), conversion rates, and, ultimately, revenue**

Solution : Cache the frequently access data

Caching is a mechanism, whether hardware or software, that stores frequently accessed data for faster retrieval compared to the original source, typically databases, resulting in high performance and low-latency access.

Primary advantages :

- Low Latency: Enables real-time responses.
- High Throughput: Supports a significant volume of data processing.
- High Scalability: Easily scales to handle increasing workloads.

Top use cases :

- Real-time Analytics
- Financial Trading Systems
- Caching and Session Storage
- Online Transaction Processing (OLTP)
- Gaming and Multimedia Applications
- Recommendation Engines
- Ad Tech and Digital Marketing
- IoT Data Processing
- Scientific and Research Applications

AWS in memory cache options

- **ElasticCache for MemCachedd** - Is simple, non-persistent caching
- **ElasticCache for Redis** - Adds persistence, replication, and more capabilities
- **MemoryDB for Redis** - Optimizes for ultra low sub-millisecond latency applications

Feature	ElastiCache for MemCachedd	ElastiCache for Redis	MemoryDB for Redis
Cache Engine	MemCachedd	Redis	Redis
Use Case	Caching, session storage	Caching, session stores, queues, leaderboards, transient data	Caching, session stores, real-time apps needing ultra low latency
Multi-AZ Support	No	Yes	Yes
Read Replicas	No	Yes	Yes
Durability	Non-persistent	Persistent	Persistent
Data Persistence	No	Yes	Yes
Automatic Backups	No	Yes	Yes
Sub-millisecond Latency	No	No	Yes
Automatic Failover	No	Yes	Yes
Data Partitioning/Sharding	No	Yes	Yes
Multi-Threaded Architecture	Yes	Yes	Yes
Security	In-Transit Encryption, IAM Authentication	In-Transit Encryption, IAM Authentication, Encryption at Rest	In-Transit Encryption, IAM Authentication, Encryption at Rest
Global Data Distribution	No	Yes (Redis Global Datastore)	Yes (Redis Global Datastore)
Monitoring and Logging	CloudWatch Metrics, Enhanced Monitoring	CloudWatch Metrics, Enhanced Monitoring	CloudWatch Metrics, Enhanced Monitoring
Compatibility with Redis Commands	Limited	Extensive	Extensive
Scalability	Horizontal scaling with MemCachedd nodes	Horizontal and Vertical scaling	Horizontal and Vertical scaling
Ease of Use	Simple	Simple	Simple
Managed Service	Yes	Yes	Yes

Challenges in Server-Based In-Memory Implementations

Managing Capacity: Capacity management in traditional server-based in-memory implementations relies on peak points, causing performance impacts during spikes—an inherent challenge.

Scaling Complexity: Scaling traditional in-memory databases requires intervention and careful capacity planning, introducing complexity

Operational Overhead: Operational tasks in traditional in-memory databases can be time-consuming, diverting focus from development efforts.

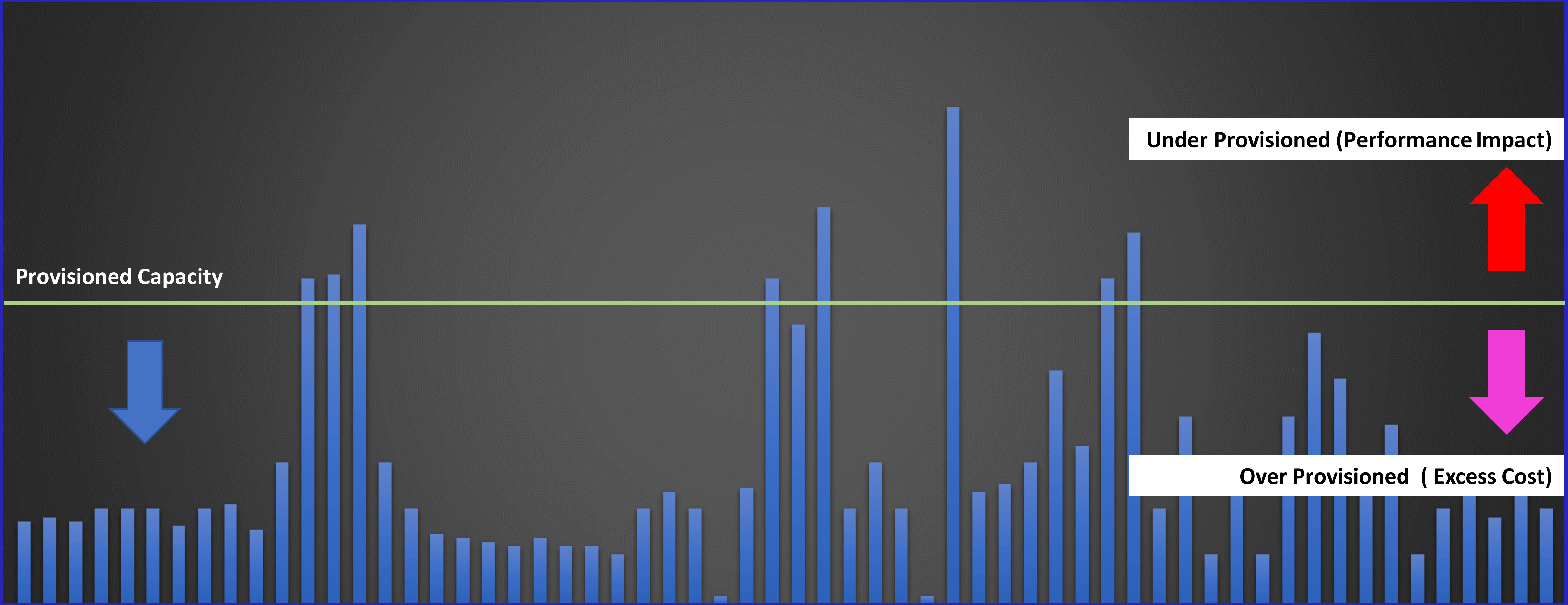
Manual High Availability Setup: Ensuring high availability in traditional in-memory databases necessitates manual implementation of redundancy and failover mechanisms.

Cost Overhead: Implementations may suffer from either over-provisioning or under-provisioning, leading to cost inefficiencies.

Infrastructure Management Burden: Managing servers for in-memory databases involves significant operational tasks, including provisioning, patching, and monitoring.

Development Slowdown: Initial setup and ongoing maintenance efforts in traditional approaches may impede development speed.

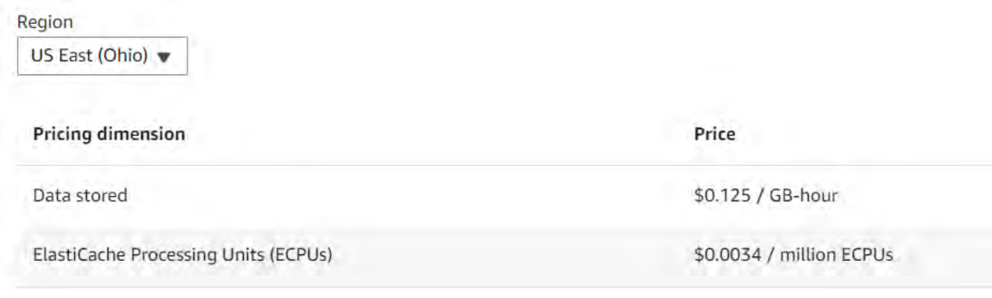
Challenges in Capacity Management



Amazon ElastiCache Serverless

- Create a cache in under a minute
- No capacity management
- 700 microseconds at p50, 1.3 milliseconds at p99
- Up to 5 TB of storage
- Pay-per-use
- 99.99% availability SLA
- Single endpoint experience
- PCI-DSS, SOC compliant, and HIPAA eligible

Pricing :



Region

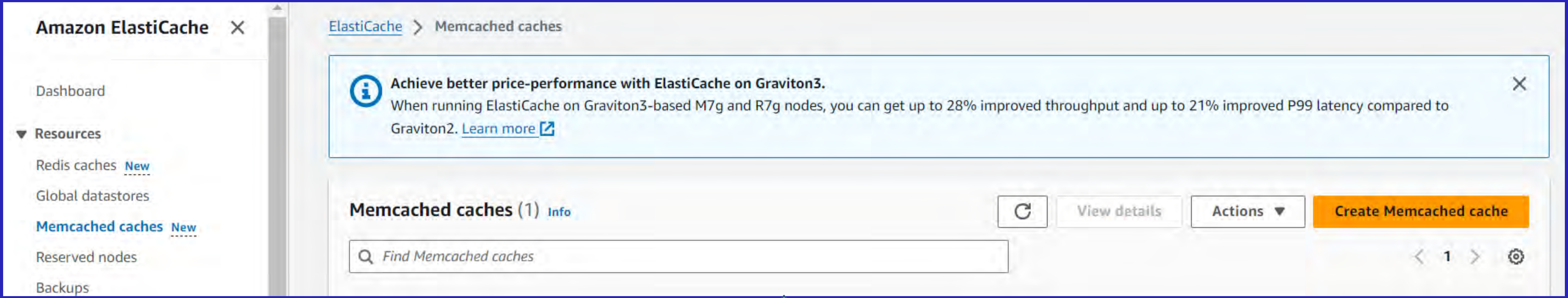
US East (Ohio) ▼

Pricing dimension	Price
Data stored	$0.125 / GB-hour
ElastiCache Processing Units (ECPUs)	$0.0034 / million ECPUs

Data stored: Pay for ElastiCache Serverless based on data stored, measured in gigabyte-hours (GB-hrs). Continuous monitoring calculates hourly averages, and each cache is metered for a minimum of 1 GB.

ElastiCache Processing Units (ECPUs): Pay for requests in ECPUs, covering vCPU time and data transfer. Each read or write consumes 1 ECPU per kilobyte (KB) transferred. Additional vCPU time or data transfer over 1 KB scales ECPUs proportionally.

MemCached : Implementationon Overview



Amazon ElastiCache

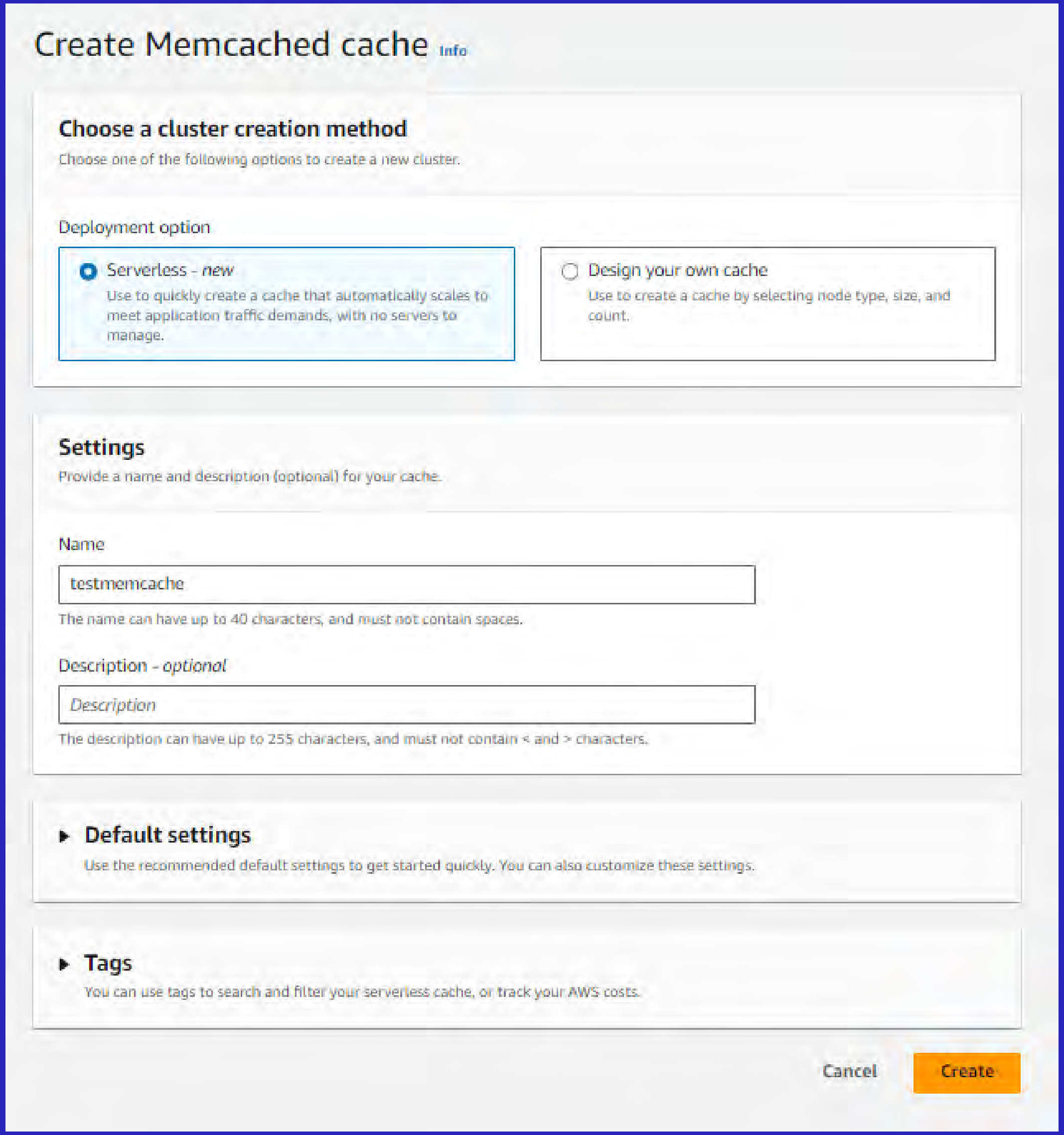
Achieve better price-performance with ElastiCache on Graviton3.
When running ElastiCache on Graviton3-based M7g and R7g nodes, you can get up to 28% improved throughput and up to 21% improved P99 latency compared to Graviton2. Learn more

Memcached caches (1) Info

Go to Amazon ElastiCache-> MemCached -> Create MemCached cache

MemCached : Implementationon Overview

1. Select Serverless
2. Give a Name
3. Just Create



Create Memcached cache Info

Choose a cluster creation method

Choose one of the following options to create a new cluster.

Deployment option

● Serverless - *new*
Use to quickly create a cache that automatically scales to meet application traffic demands, with no servers to manage.

○ Design your own cache
Use to create a cache by selecting node type, size, and count.

Settings

Provide a name and description (optional) for your cache.

Name

testmemcache

The name can have up to 40 characters, and must not contain spaces.

Description - *optional*

Description

The description can have up to 255 characters, and must not contain < and > characters.

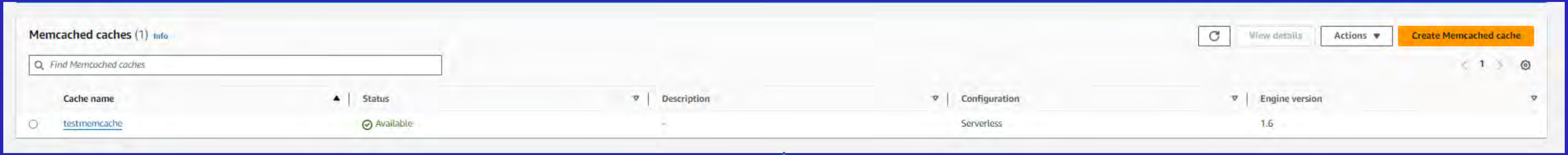
▸ **Default settings**

Use the recommended default settings to get started quickly. You can also customize these settings.

▸ **Tags**

You can use tags to search and filter your serverless cache, or track your AWS costs.

MemCached : Implementationon Overview

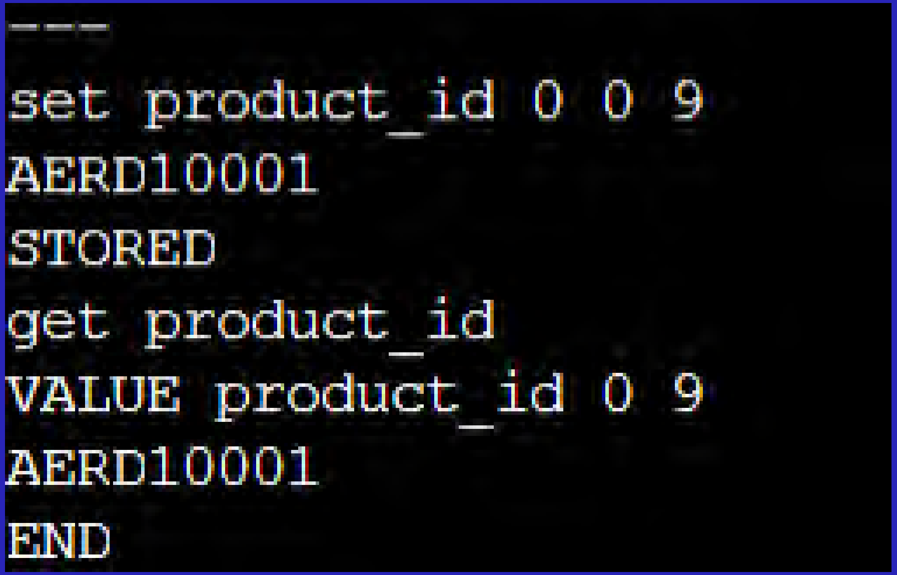


Memcached caches (1) Info

	Cache name	Status	Description	Configuration	Engine version
○	testmemcache	✓ Available	-	Serverless	1.6

Cache will get created under a 1 min

MemCached : Implementation Overview



```
---
set product_id 0 0 9
AERD10001
STORED
get product_id
VALUE product_id 0 9
AERD10001
END
```

Command	Description
/usr/bin/openssl s_client -connect <MemCached end point> -crlf	Initiates a secure connection to the MemCachedd server using OpenSSL and specifies the endpoint and port (11212).
set product_id 0 0 9	Sets the variable product_id ' with an expiration time of 0 (no expiration) and a data size of 9 bytes.
AERD10001	Assigns the value "AERD10001 " to the variable 'product_id “
STORED	Indicates that the data was successfully stored in the MemCachedd server.
get product_id	Retrieves the value of the variable 'a'.
VALUE product 0 5	Indicates that the variable 'a' has a data size of 5 bytes and starts the output of the variable's value.
AERD10001	Displays the value assigned to the variable 'a', in this case, "hello".
END	Marks the end of the response.

Anti-Patterns to Avoid

Anti-Pattern	Description	How to Avoid
Over-Reliance on Caching	Avoid relying excessively on caching. Critical data should still be retrievable from the primary data source to ensure accuracy.	• Perform periodic assessments to identify critical data that should always be retrieved from the primary source. • Implement fallback mechanisms to fetch data from the primary source when not available in the cache.
Not Handling Cache Misses	Implement strategies to handle cache misses effectively, ensuring that your application gracefully handles scenarios where data is not in the cache.	• Develop error-handling mechanisms to gracefully manage cache misses. • Implement a mechanism to retrieve data from the primary source when a cache miss occurs. - Consider using default values when appropriate to maintain application functionality.
Neglecting Security Measures	Don't overlook security considerations. Implement proper authentication and authorization mechanisms to protect sensitive data in the cache.	• Implement robust authentication and authorization mechanisms for access to the cache. • Encrypt sensitive data stored in the cache. - Regularly review and update security measures to address emerging threats.

Best Practices to follow

Best Practices	Importance	Action	Examples
Optimize Data Access Patterns:	Maximizing the benefits of serverless caching.	Design cache access patterns tailored to your application's specific needs.	Read-through, write-through, write-behind, cache-aside, refresh-ahead, cache-aside with write-behind
Use Efficient Serialization:	Reducing data transfer costs and improving overall performance.	Opt for efficient data serialization formats.	MessagePack, Protocol Buffers, Apache Avro, JSON, FlatBuffers
Leverage Cache Keys Wisely:	Facilitating easy retrieval and minimizing cache collisions.	Choose meaningful and efficient cache keys.	user_profile:{user_id}, product_info:{product_id}, session_data:{session_id}
Monitor and Analyze:	Identifying performance issues, usage patterns, and bottlenecks.	Implement robust monitoring,	Cache hit & miss rate, latency/response time, cache eviction rate, data transfer volume, cache size

Thank you!