Unleash the power of Redis with Amazon ElastiCache

Michael Labib,
Specialist Solutions Architect

aws | Pop-up Loft

What we’ll cover

- ElastiCache Redis Overview
- Common Architecture Patterns
- Best Practices
- Caching Strategies

Amazon

ElastiCache

In-Memory Key-Value Store

High-performance

Redis and Memcached

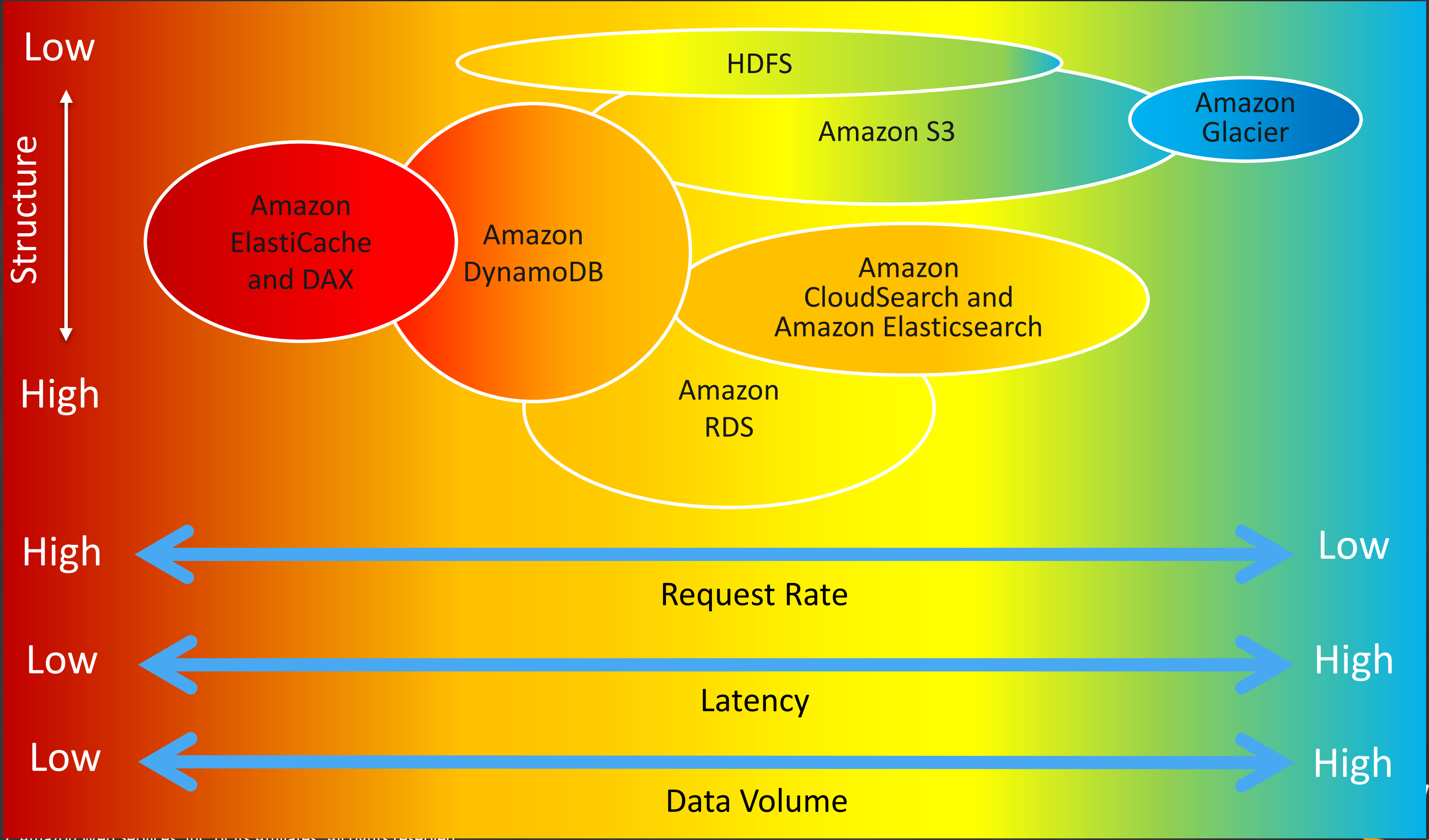
Fully managed; Zero admin

Highly Available and Reliable

Hardened by Amazon

redis

memcached



Low

Structure

High

HDFS

Amazon S3

Amazon Glacier

Amazon ElastiCache and DAX

Amazon DynamoDB

Amazon CloudSearch and Amazon Elasticsearch

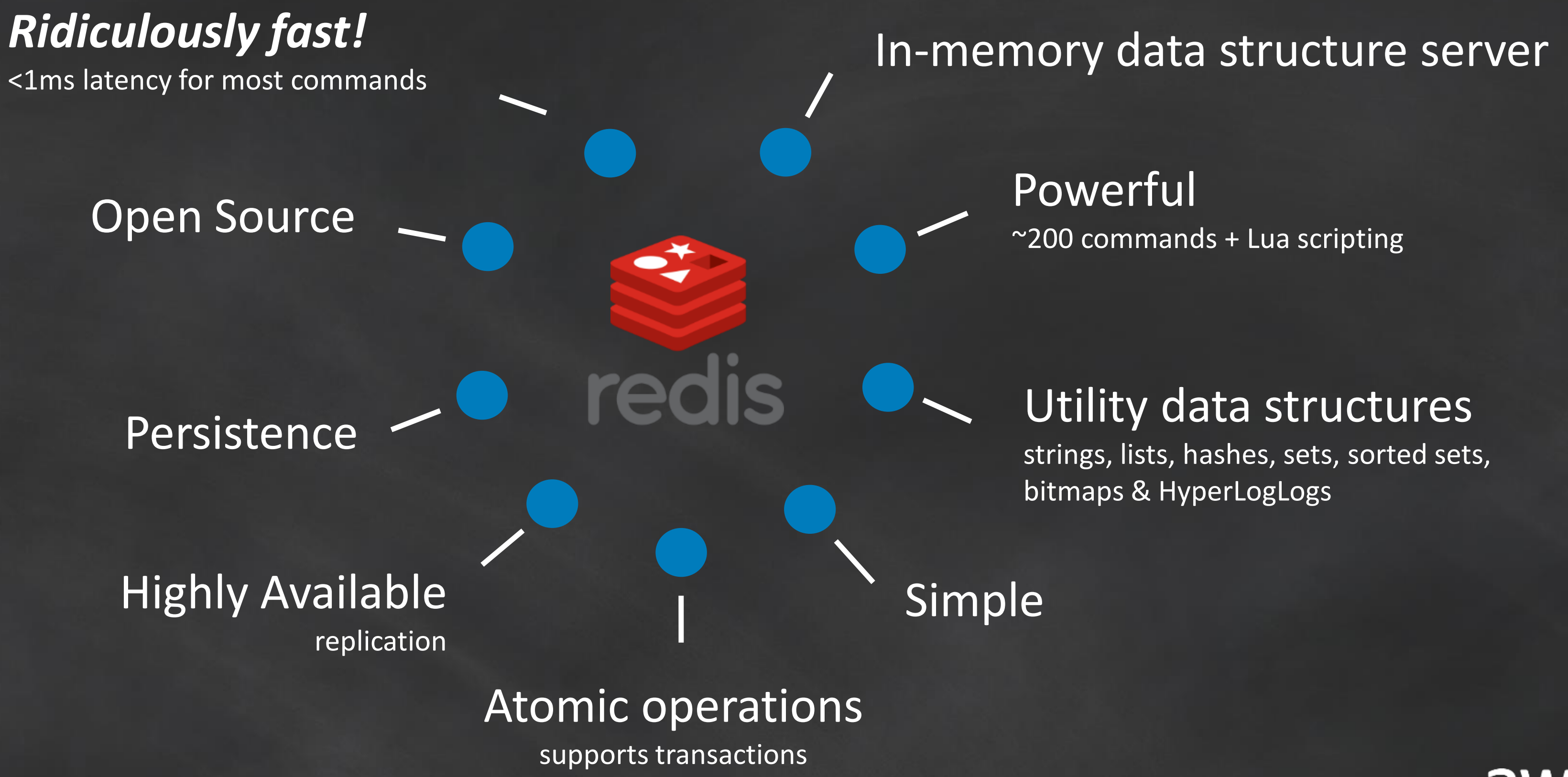
Amazon RDS

High — Request Rate — Low

Low — Latency — High

Low — Data Volume — High

Redis – The In-Memory Leader



Ridiculously fast!

<1ms latency for most commands

In-memory data structure server

Open Source

Powerful

~200 commands + Lua scripting

Persistence

Utility data structures

strings, lists, hashes, sets, sorted sets, bitmaps & HyperLogLogs

Highly Available

replication

Simple

Atomic operations

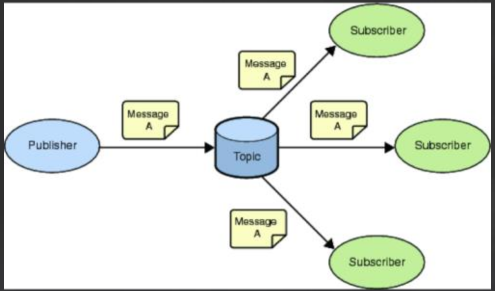
supports transactions

Redis Data Types & More!

Run Lua scripts

10 km
Lat: 48.861
Lon: 2.336
Ermont
Garges-lès-Gonesse
Villepinte
Argenteuil
Saint-Denis
Aulnay-sous-Bois
Drancy
Asnières-sur-Seine
Bobigny
Montfermeil
Nanterre
Montmartre Cemetery
Noisy-le-Sec
Chelles
Suresnes
Vincennes
Noisy-le-Grand
Boulogne-Billancourt
Ivry-sur-Seine
Clamart
Villejuif
Créteil
Sucy-en-Brie
Antony
Choisy-le-Roi
Orly
Massy
Igny
Villeneuve-Saint-Georges

Geospatial Queries!



Pub / Sub

ElastiCache Engine Enhancements

Amazon

ElastiCache

Optimized Swap Memory

- Mitigate the risk of increased swap usage during syncs and snapshots.

Dynamic write throttling

- Improved output buffer management when the node’s memory is close to being exhausted.

Smoother **failovers**

- Clusters recover faster as replicas avoid flushing their data to do a full re-sync with the primary.

Enhanced **failover quorum logic**

When majority primary nodes are missing, the cluster is still operational

ElastiCache Redis Topologies

ElastiCache Redis with Multi-AZ (non-clustered)

ElastiCache for Redis Multi-AZ

Automatic Failover to a read replica in case of primary node failure

Auto-Failover

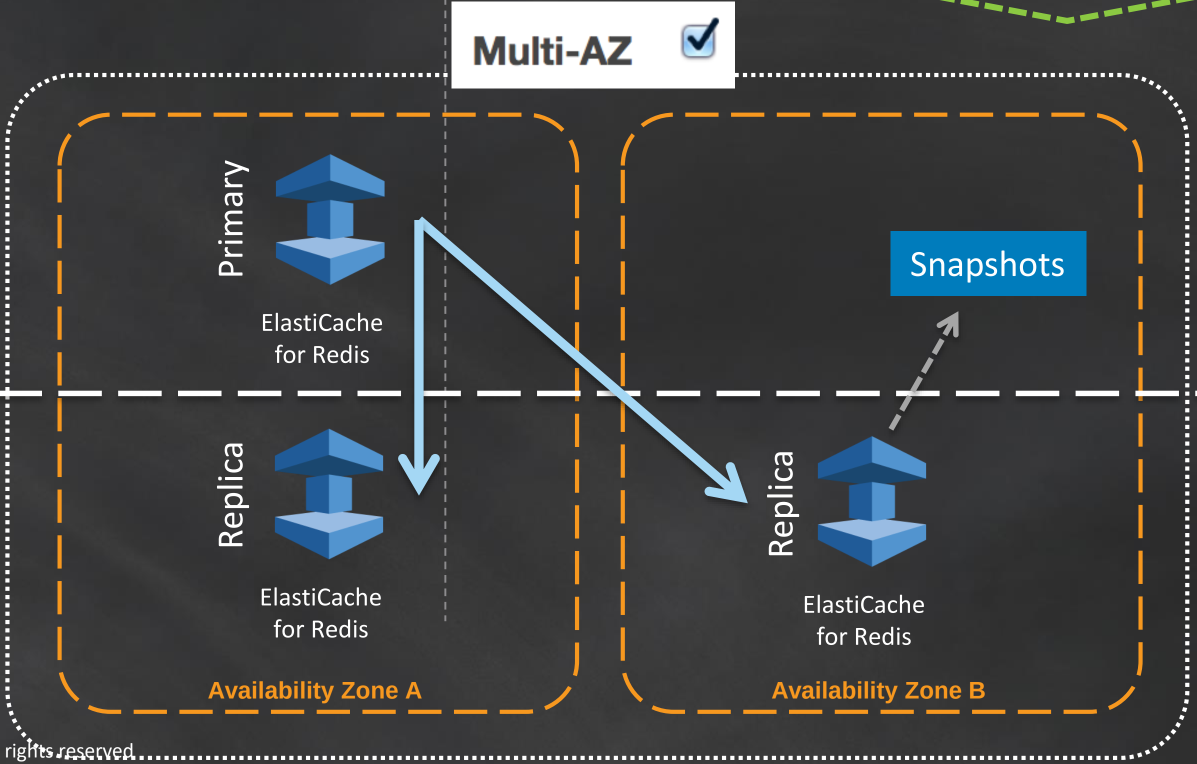
- Chooses replica with lowest replication lag
- DNS endpoint is same

writes

Use Primary Endpoint

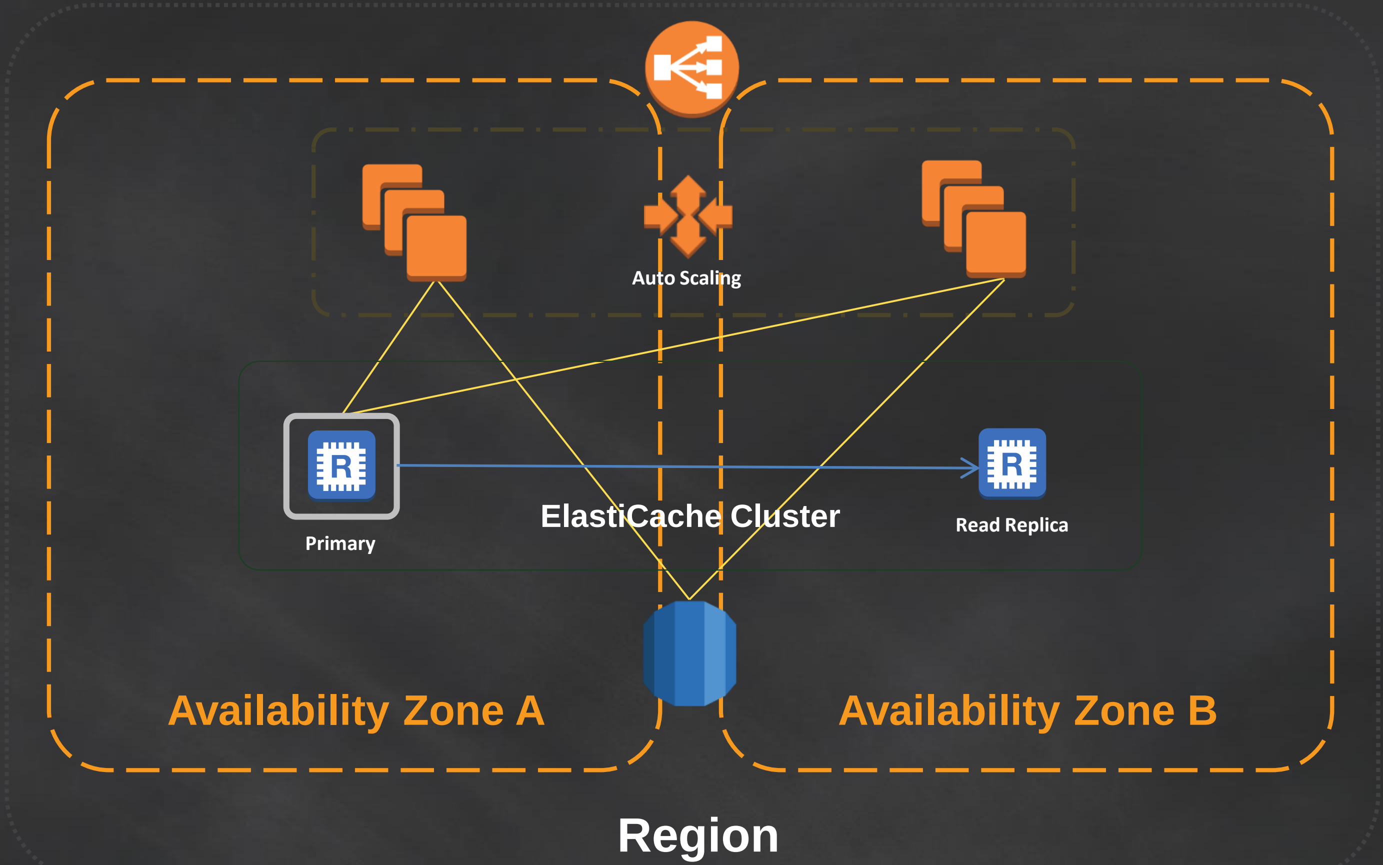
reads

Use Read Replicas

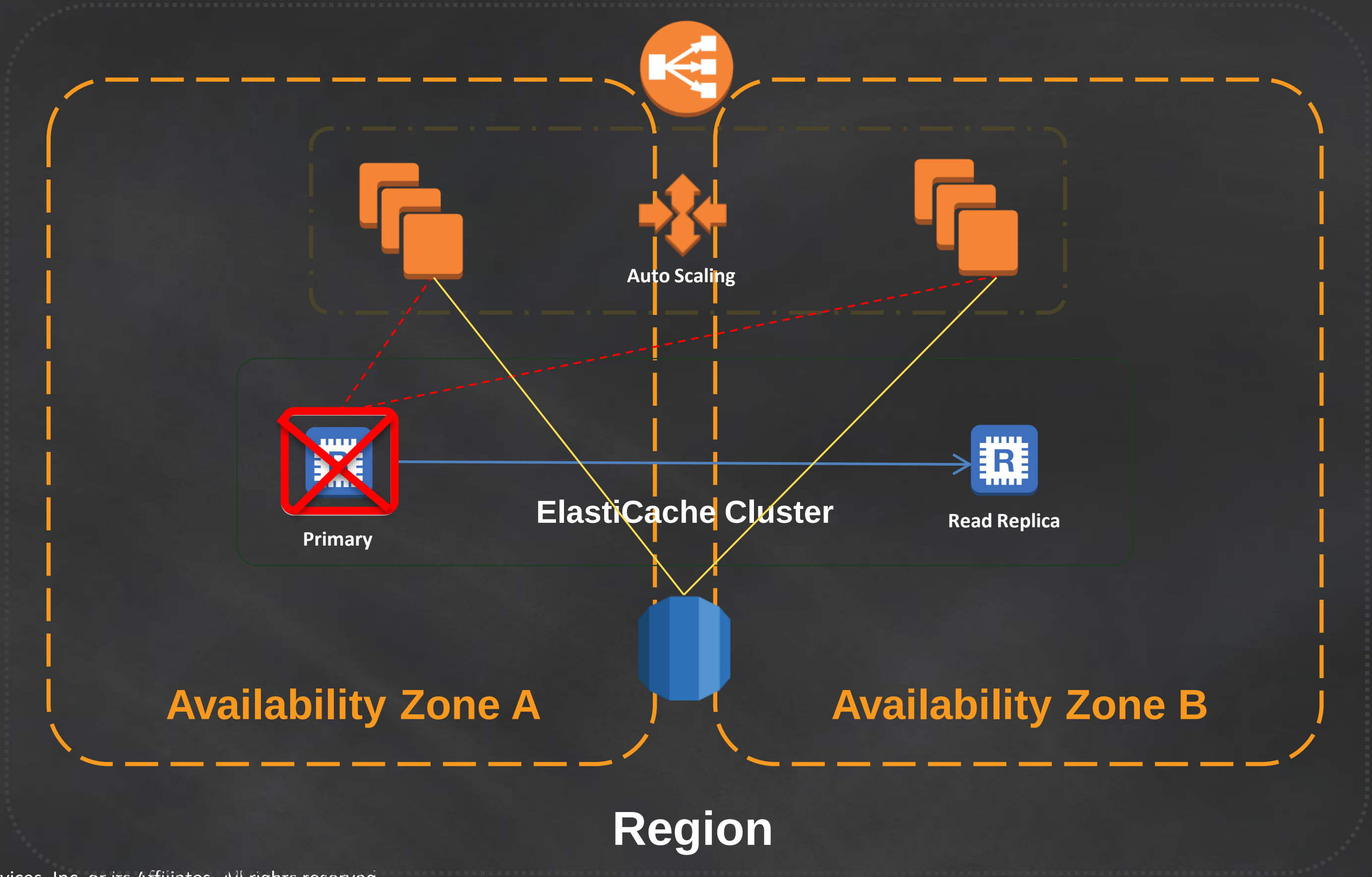


ElastiCache Automates snapshots for persistence

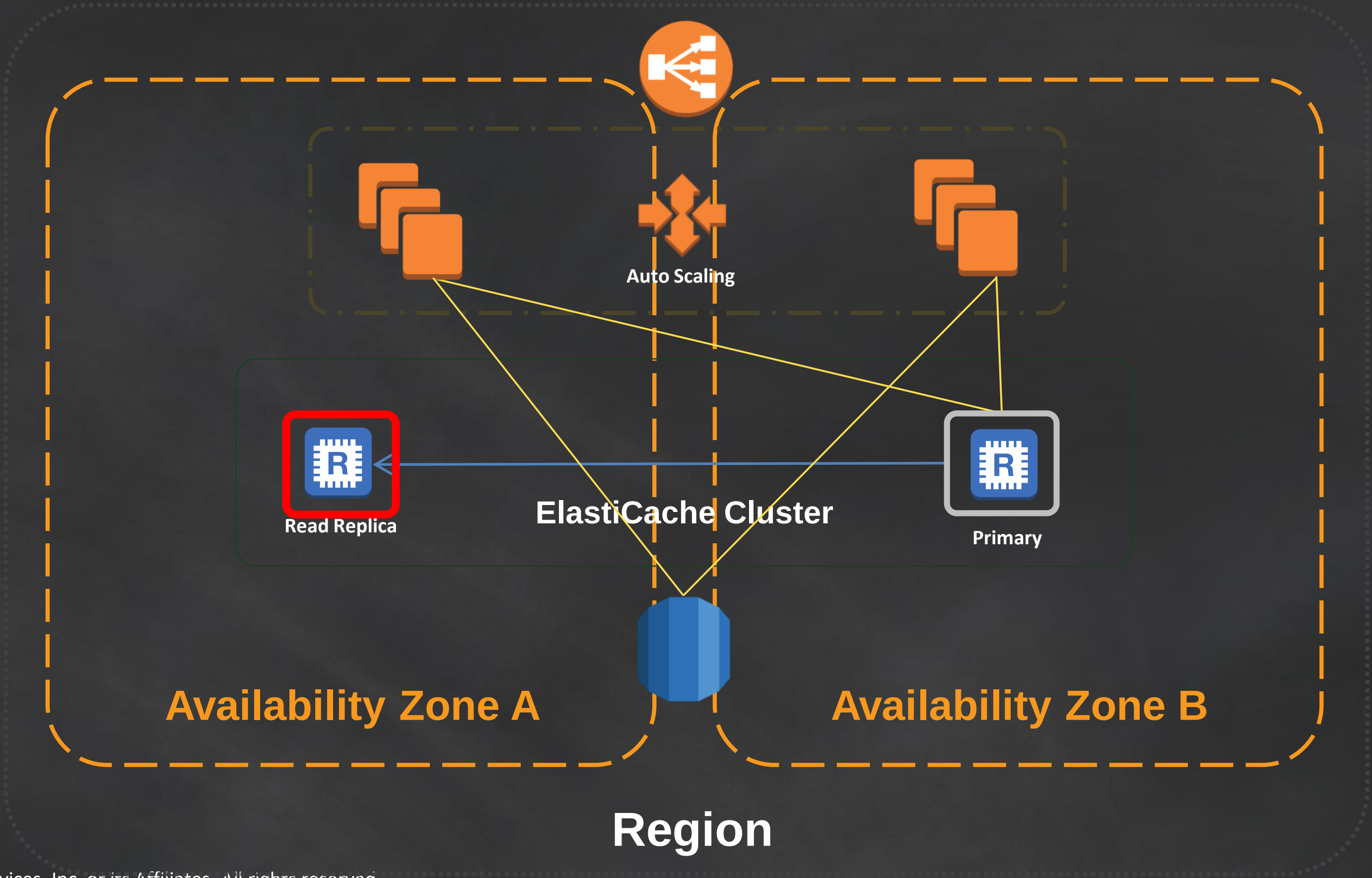
ElastiCache with Redis Multi-AZ



ElastiCache with Redis Multi-AZ



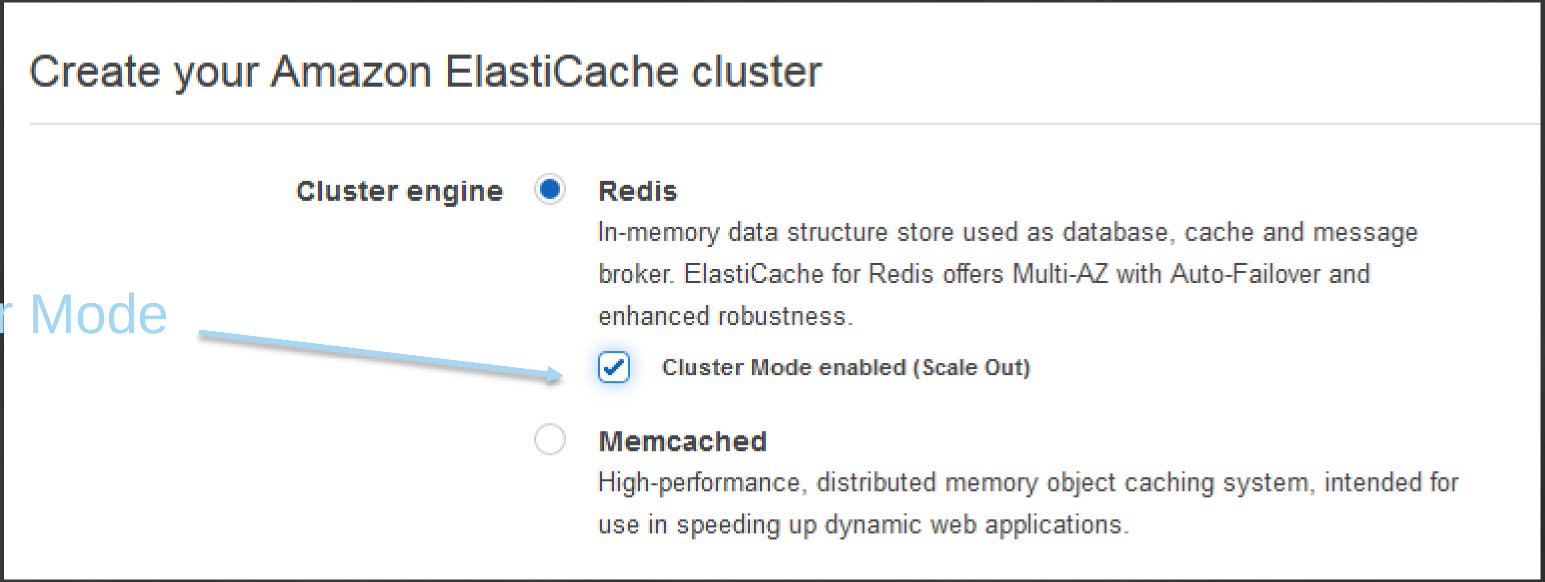
ElastiCache with Redis Multi-AZ



Scaling with Redis Cluster (clustered mode enabled)

Scaling with Redis Cluster (clustered mode enabled)

Setting up Redis Cluster - Console



Create your Amazon ElastiCache cluster

Cluster engine

● **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.

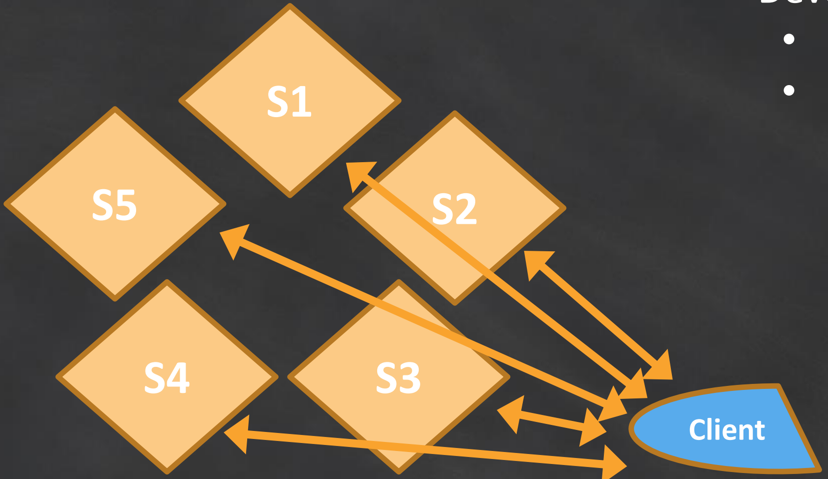
☑ **Cluster Mode enabled (Scale Out)**

○ **Memcached**

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

Cluster Mode

Redis Cluster – Automatic Client-Side Sharding



- 16384 hash slots per Cluster
  - Slot for a key is CRC16 modulo {key}
- Slots are distributed across the Cluster into Shards
- Developers must use a Redis cluster client!
  - Clients are redirected to the correct shard
  - Smart clients store a map

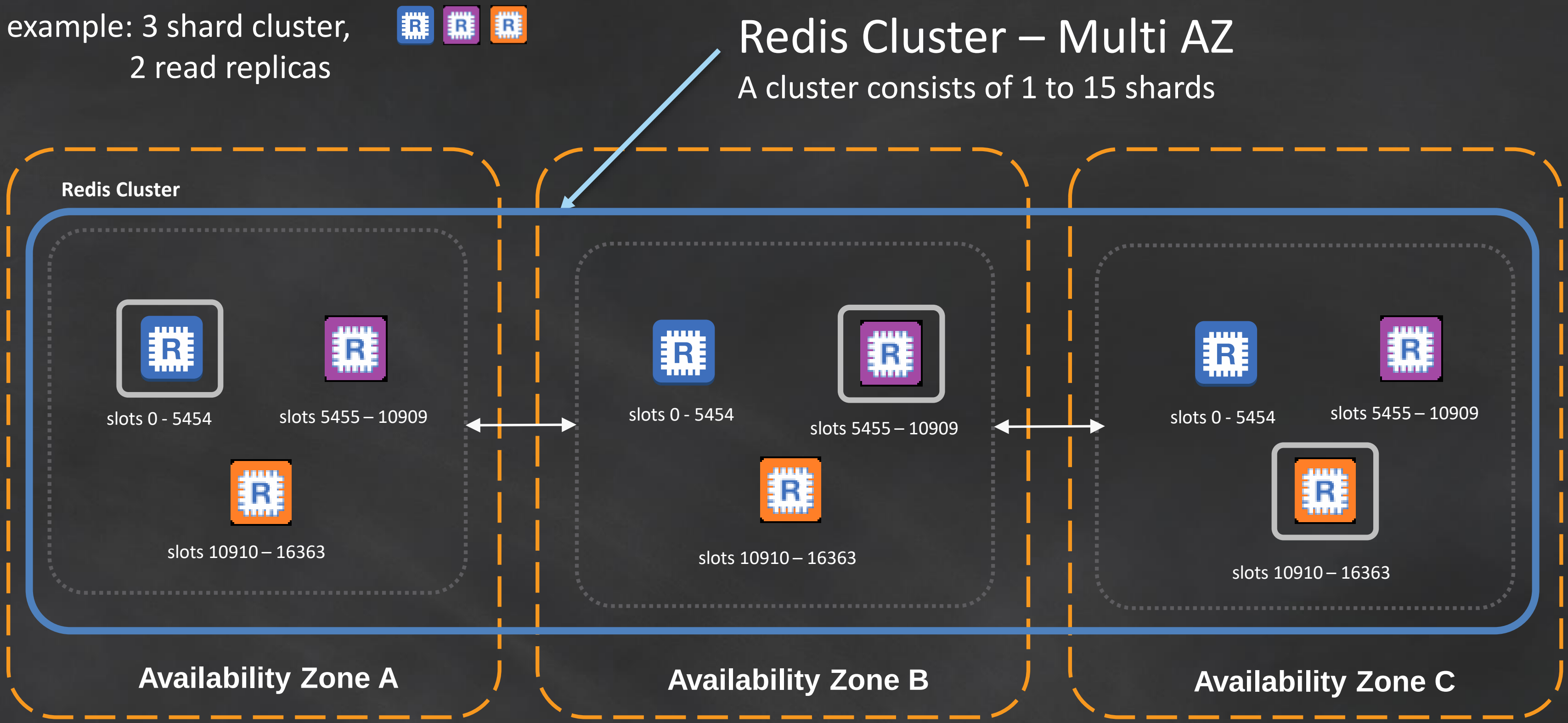
Shard S1 = slots 0 – 3276
Shard S2 = slots 3277 – 6553
Shard S3 = slots 6554 – 9829
Shard S4 = slots 9830 – 13106
Shard S5 = slots 13107 - 16383

Redis Cluster – Architecture

example: 3 shard cluster,
2 read replicas

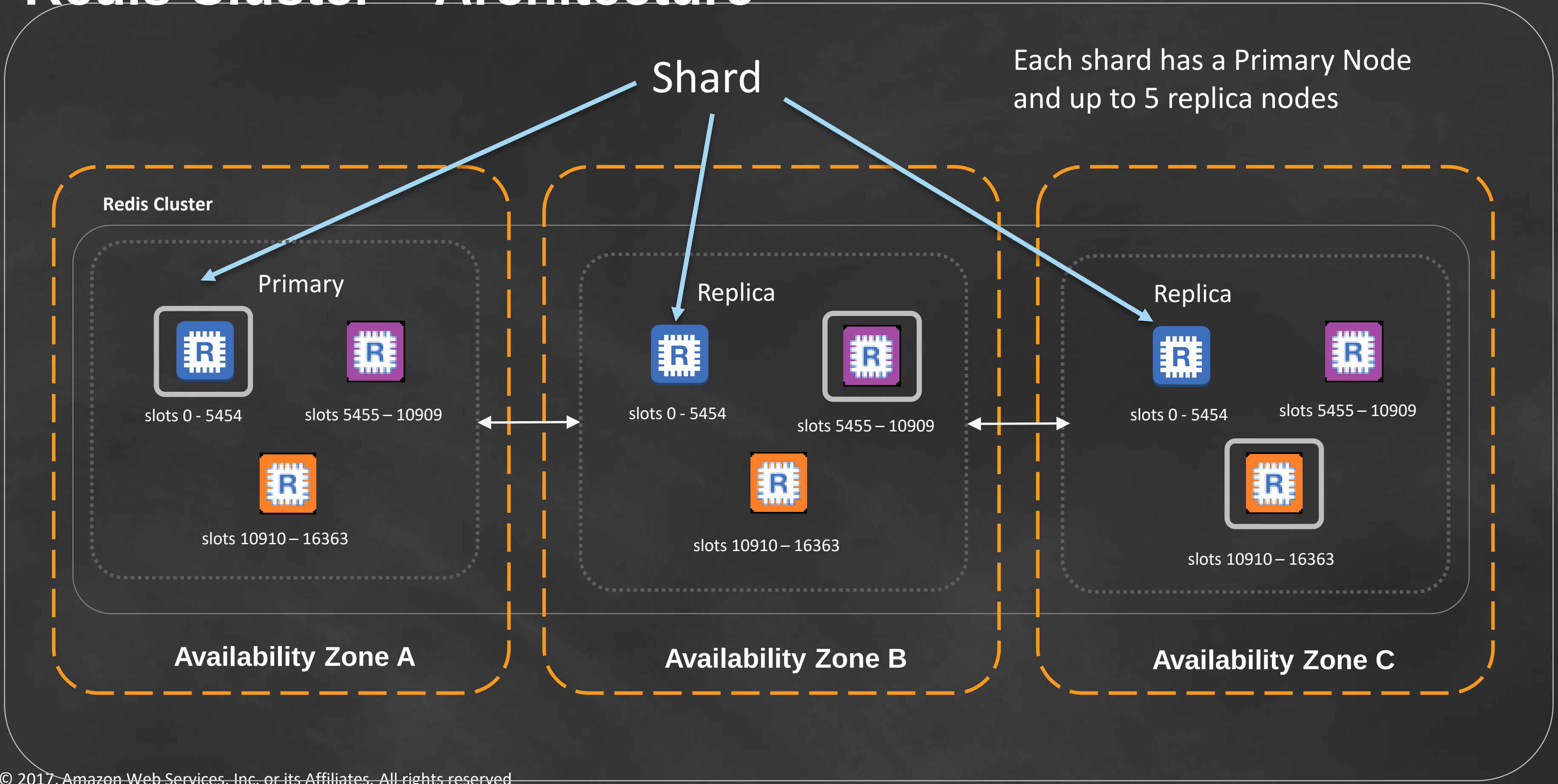
Redis Cluster – Multi AZ

A cluster consists of 1 to 15 shards



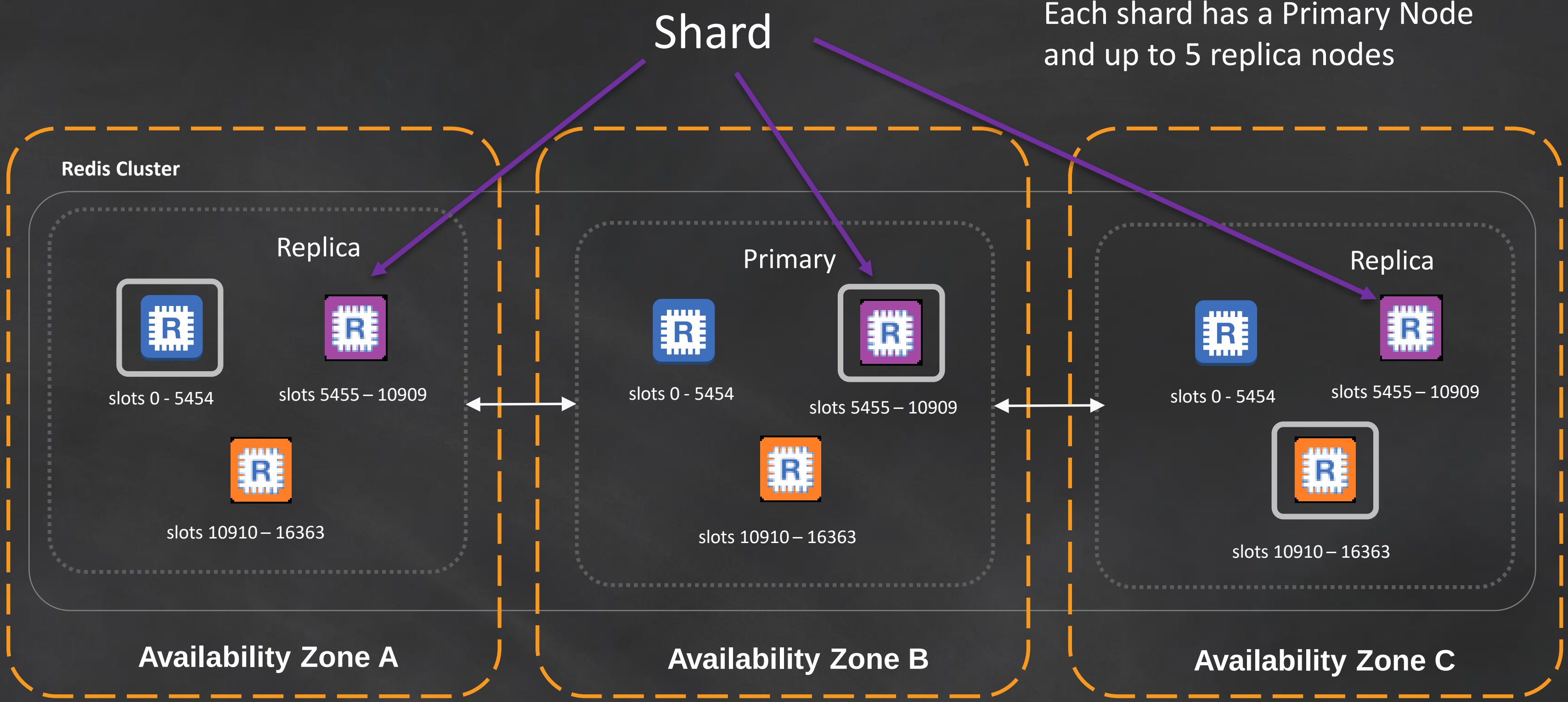
Redis Cluster – Architecture

Each shard has a Primary Node and up to 5 replica nodes



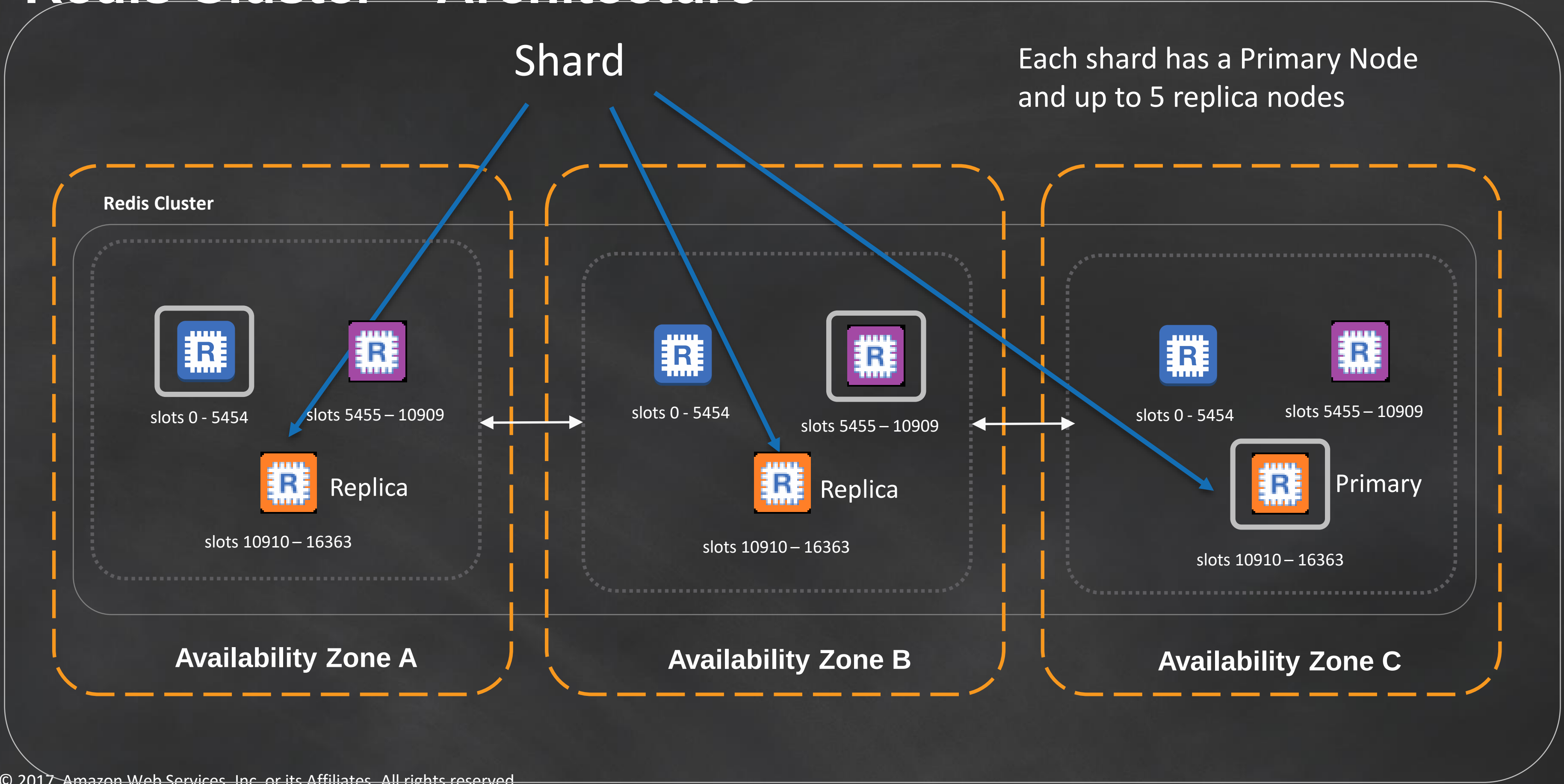
Redis Cluster – Architecture

Each shard has a Primary Node and up to 5 replica nodes



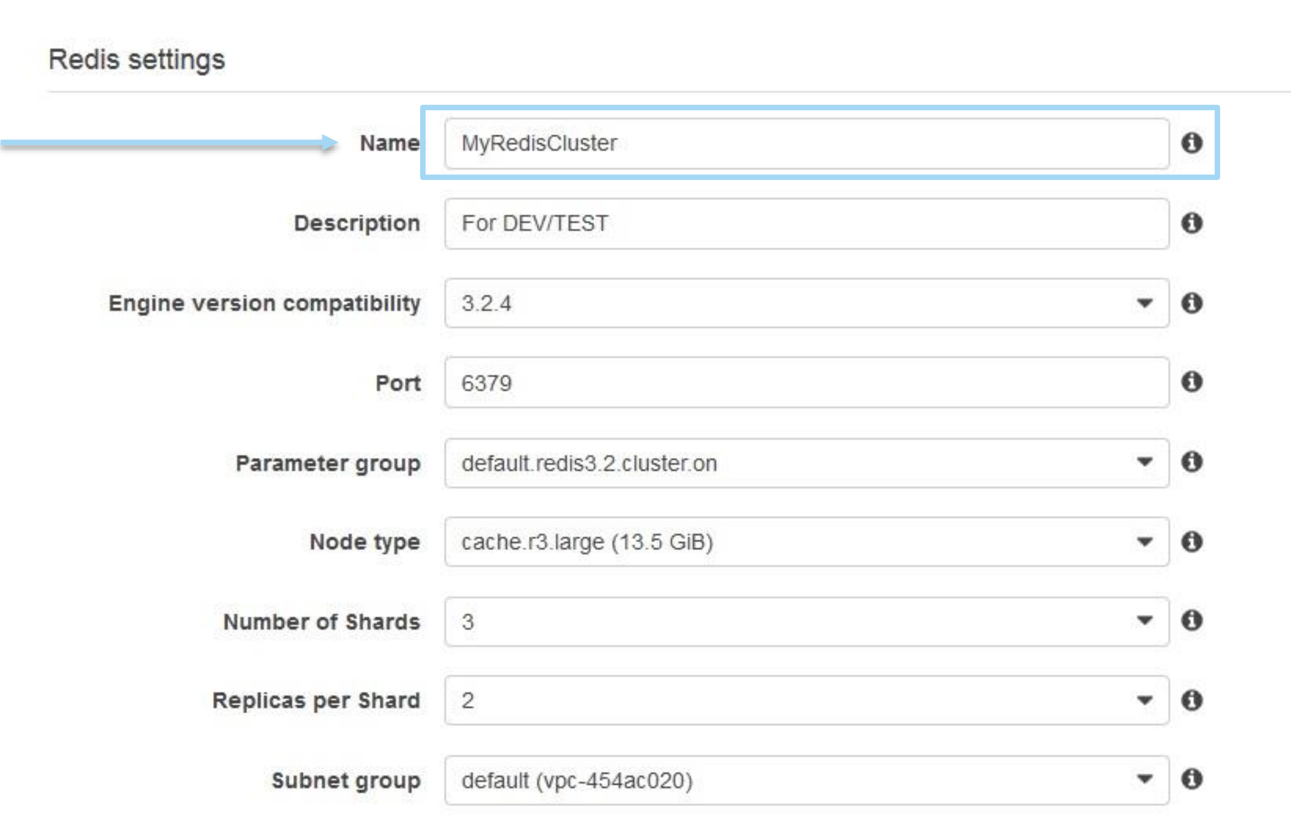
Redis Cluster – Architecture

Each shard has a Primary Node and up to 5 replica nodes



Setting up Redis Cluster - Console

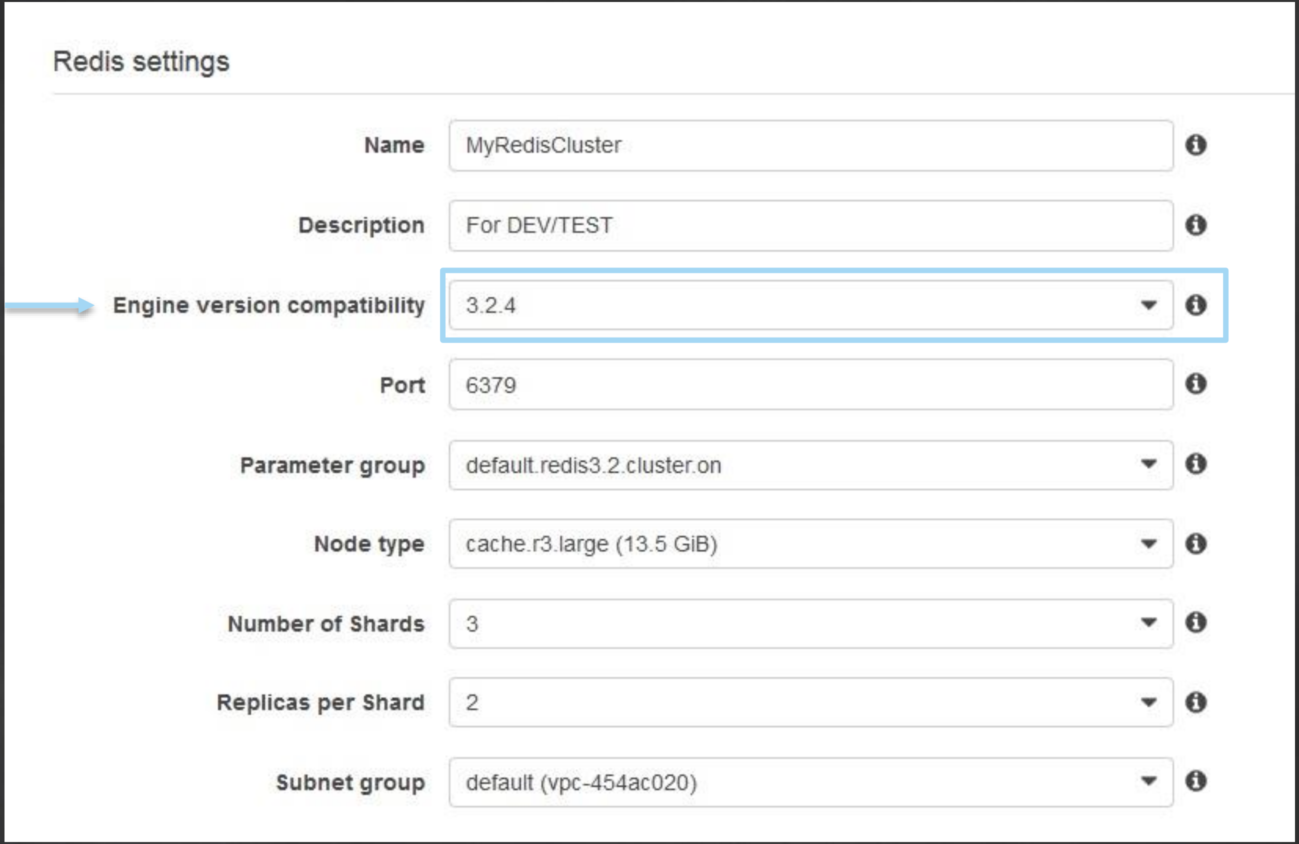
Cluster Name



Redis settings

Name	MyRedisCluster
Description	For DEV/TEST
Engine version compatibility	3.2.4
Port	6379
Parameter group	default.redis3.2.cluster.on
Node type	cache.r3.large (13.5 GiB)
Number of Shards	3
Replicas per Shard	2
Subnet group	default (vpc-454ac020)

Setting up Redis Cluster - Console

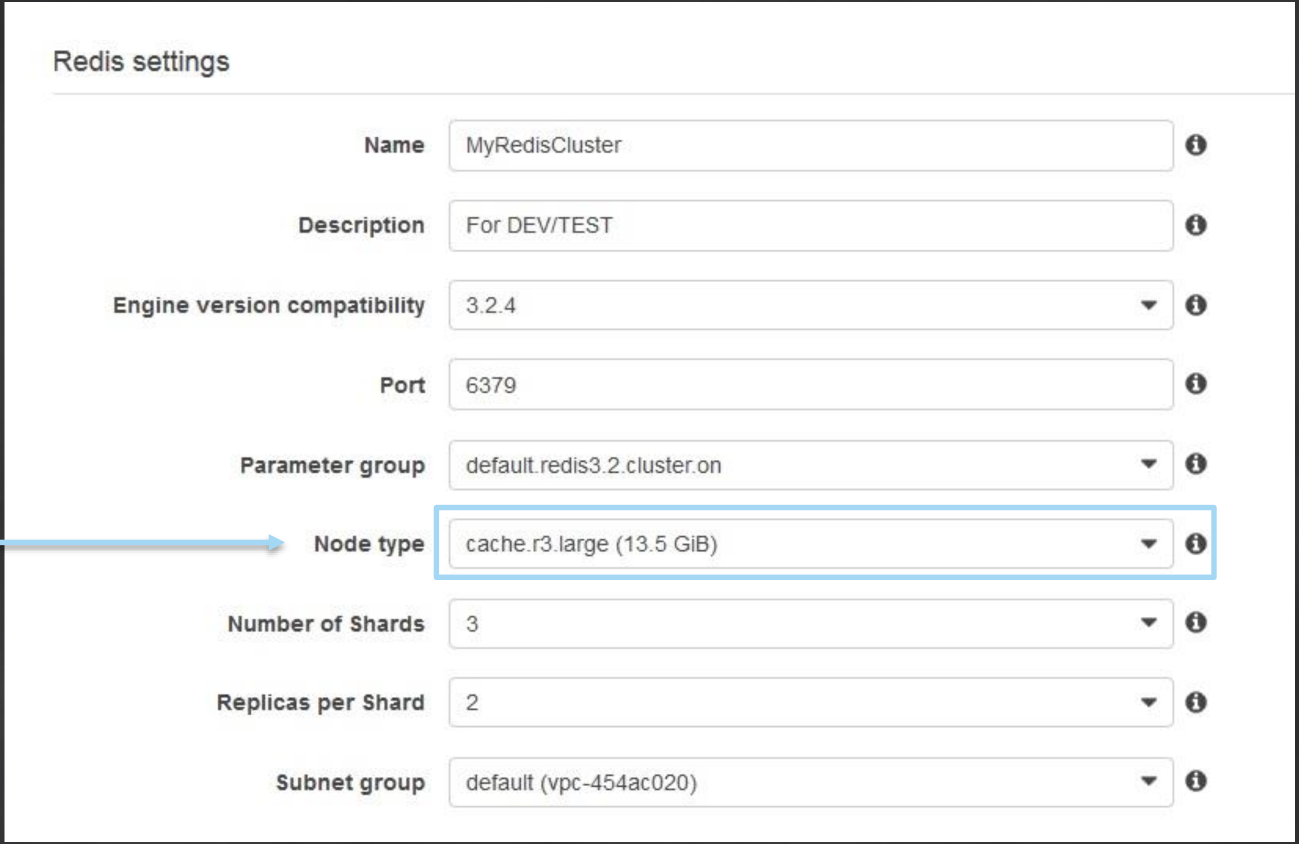


Redis settings

Name	MyRedisCluster
Description	For DEV/TEST
Engine version compatibility	3.2.4
Port	6379
Parameter group	default.redis3.2.cluster.on
Node type	cache.r3.large (13.5 GiB)
Number of Shards	3
Replicas per Shard	2
Subnet group	default (vpc-454ac020)

Redis Version

Setting up Redis Cluster - Console

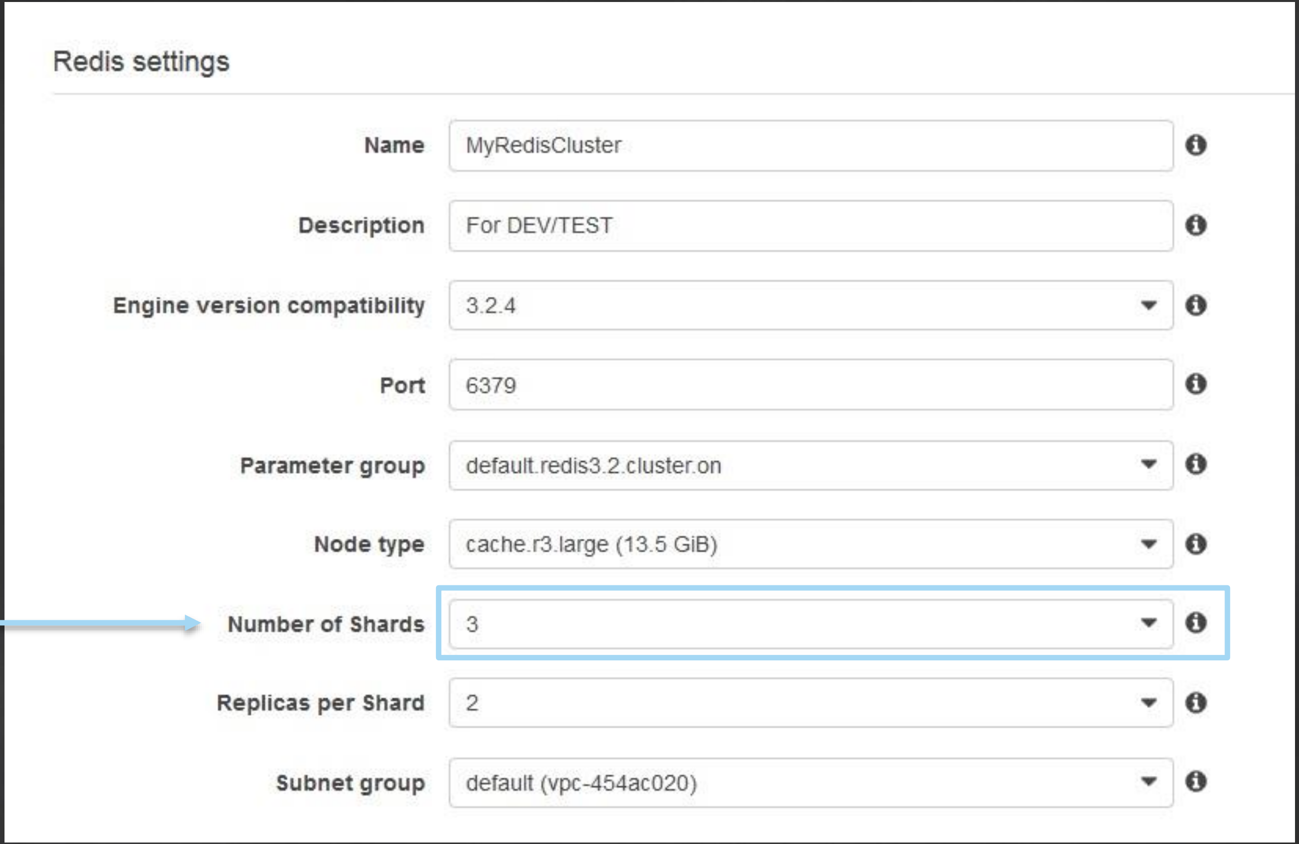


Redis settings

Name	MyRedisCluster
Description	For DEV/TEST
Engine version compatibility	3.2.4
Port	6379
Parameter group	default.redis3.2.cluster.on
Node type	cache.r3.large (13.5 GiB)
Number of Shards	3
Replicas per Shard	2
Subnet group	default (vpc-454ac020)

Instance

Setting up Redis Cluster - Console

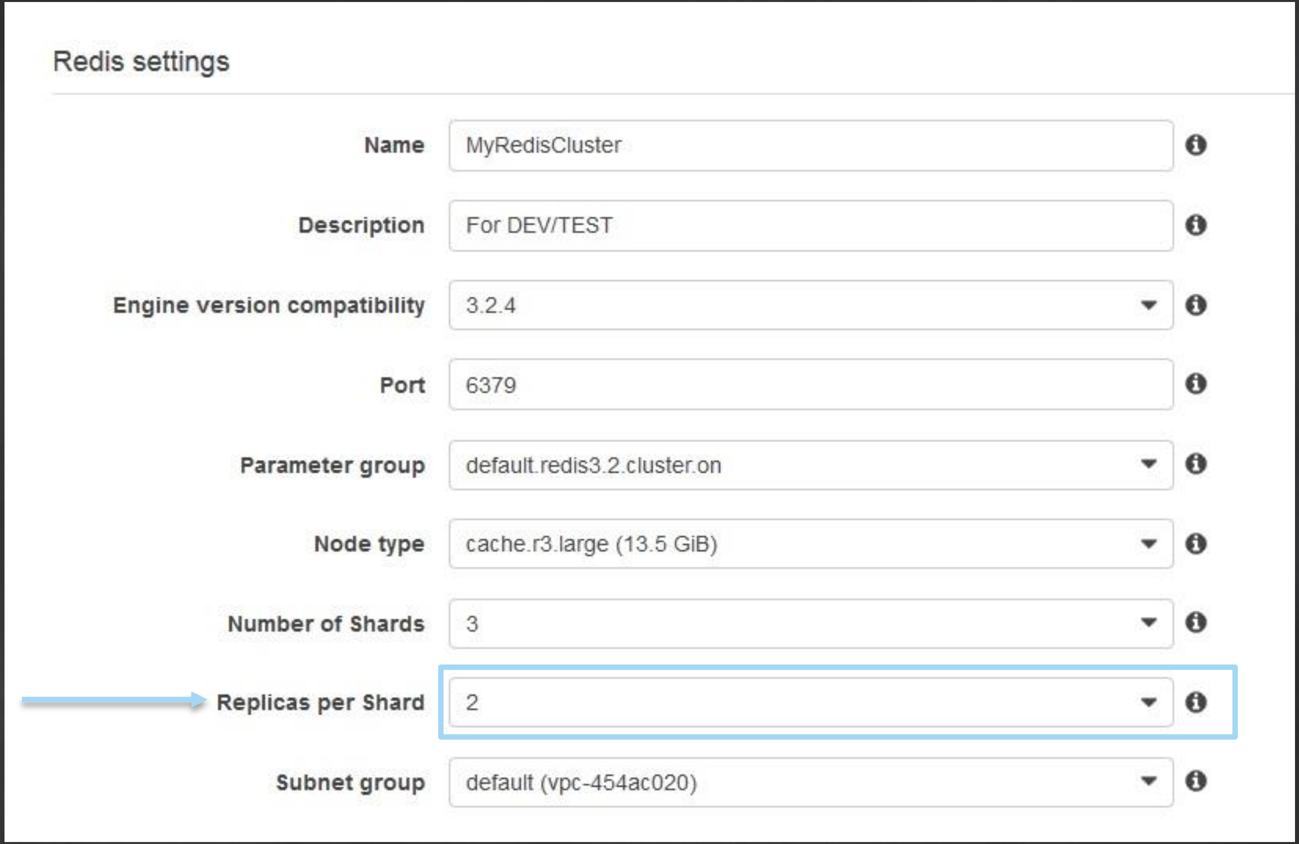


Redis settings

Name	MyRedisCluster
Description	For DEV/TEST
Engine version compatibility	3.2.4
Port	6379
Parameter group	default.redis3.2.cluster.on
Node type	cache.r3.large (13.5 GiB)
Number of Shards	3
Replicas per Shard	2
Subnet group	default (vpc-454ac020)

# of Shards

Setting up Redis Cluster - Console



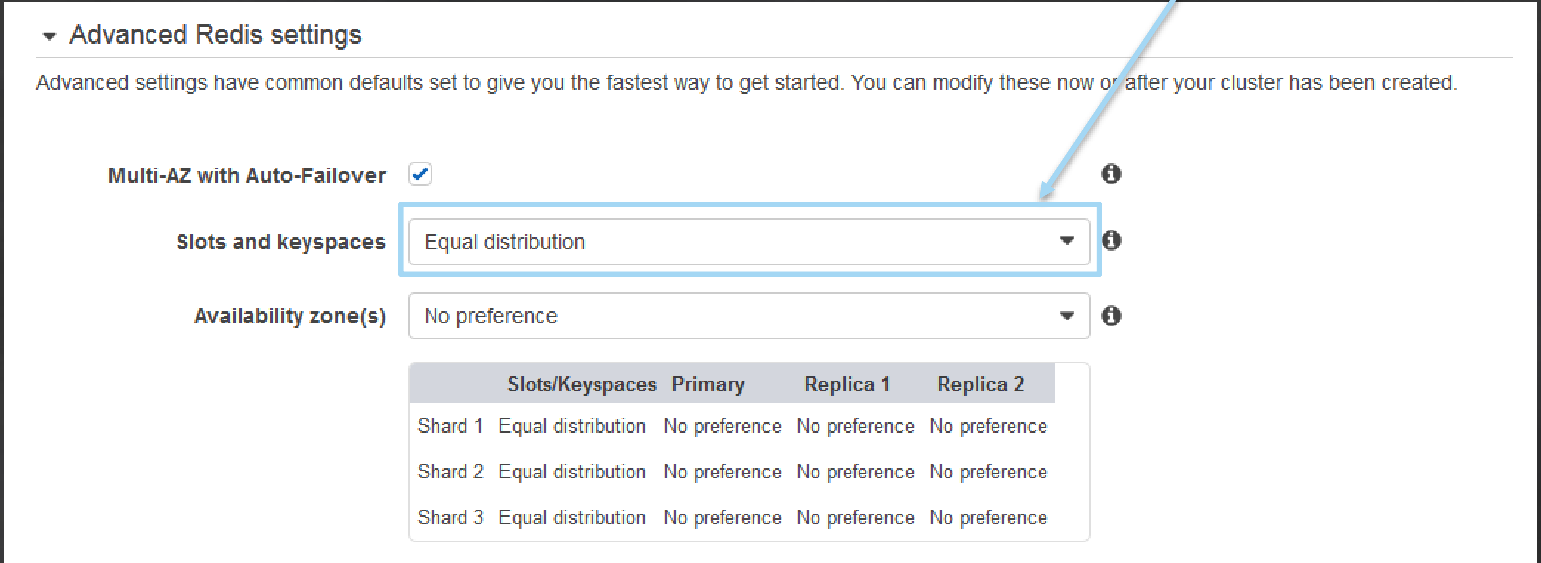
Redis settings

Name	MyRedisCluster
Description	For DEV/TEST
Engine version compatibility	3.2.4
Port	6379
Parameter group	default.redis3.2.cluster.on
Node type	cache.r3.large (13.5 GiB)
Number of Shards	3
Replicas per Shard	2
Subnet group	default (vpc-454ac020)

# of Replica's

Setting up Redis Cluster - Console

Slots Distribution



▾ 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.

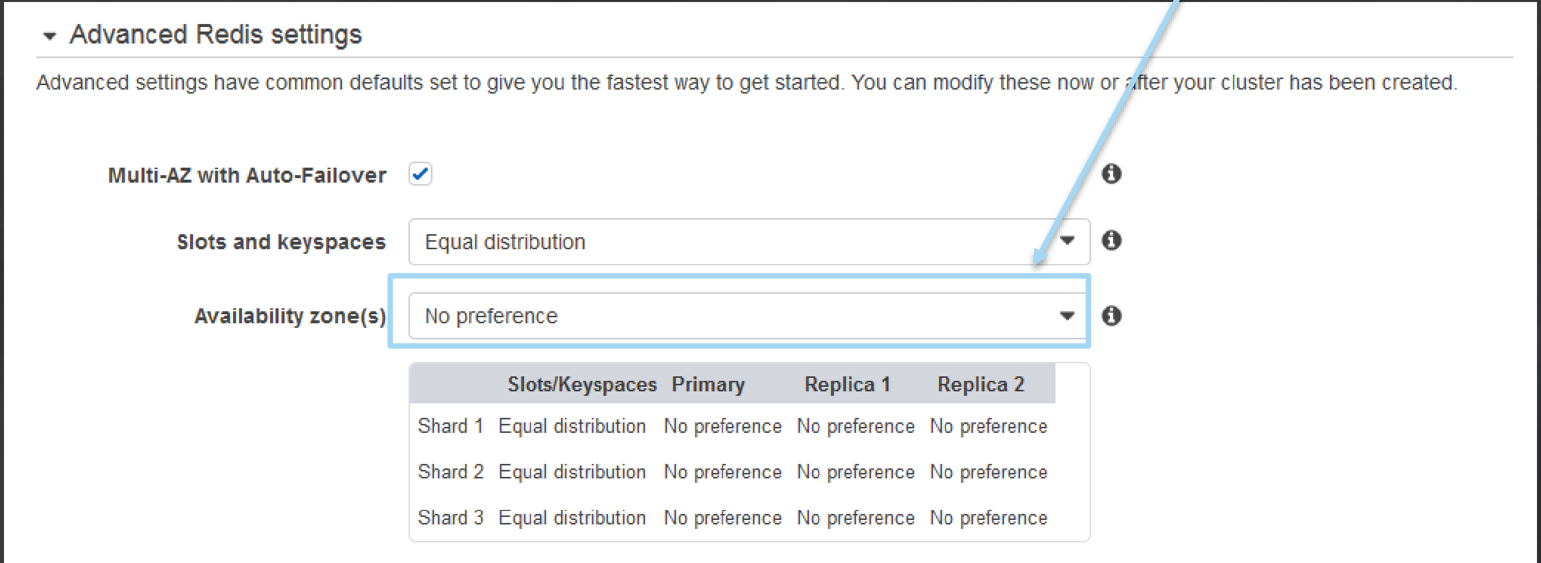
Multi-AZ with Auto-Failover ☑

Slots and keyspaces Equal distribution ▾

Availability zone(s) No preference ▾

	Slots/Keyspaces	Primary	Replica 1	Replica 2
Shard 1	Equal distribution	No preference	No preference	No preference
Shard 2	Equal distribution	No preference	No preference	No preference
Shard 3	Equal distribution	No preference	No preference	No preference

Setting up Redis Cluster - Console



Select AZs

▾ 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 ☑

Slots and keyspaces Equal distribution

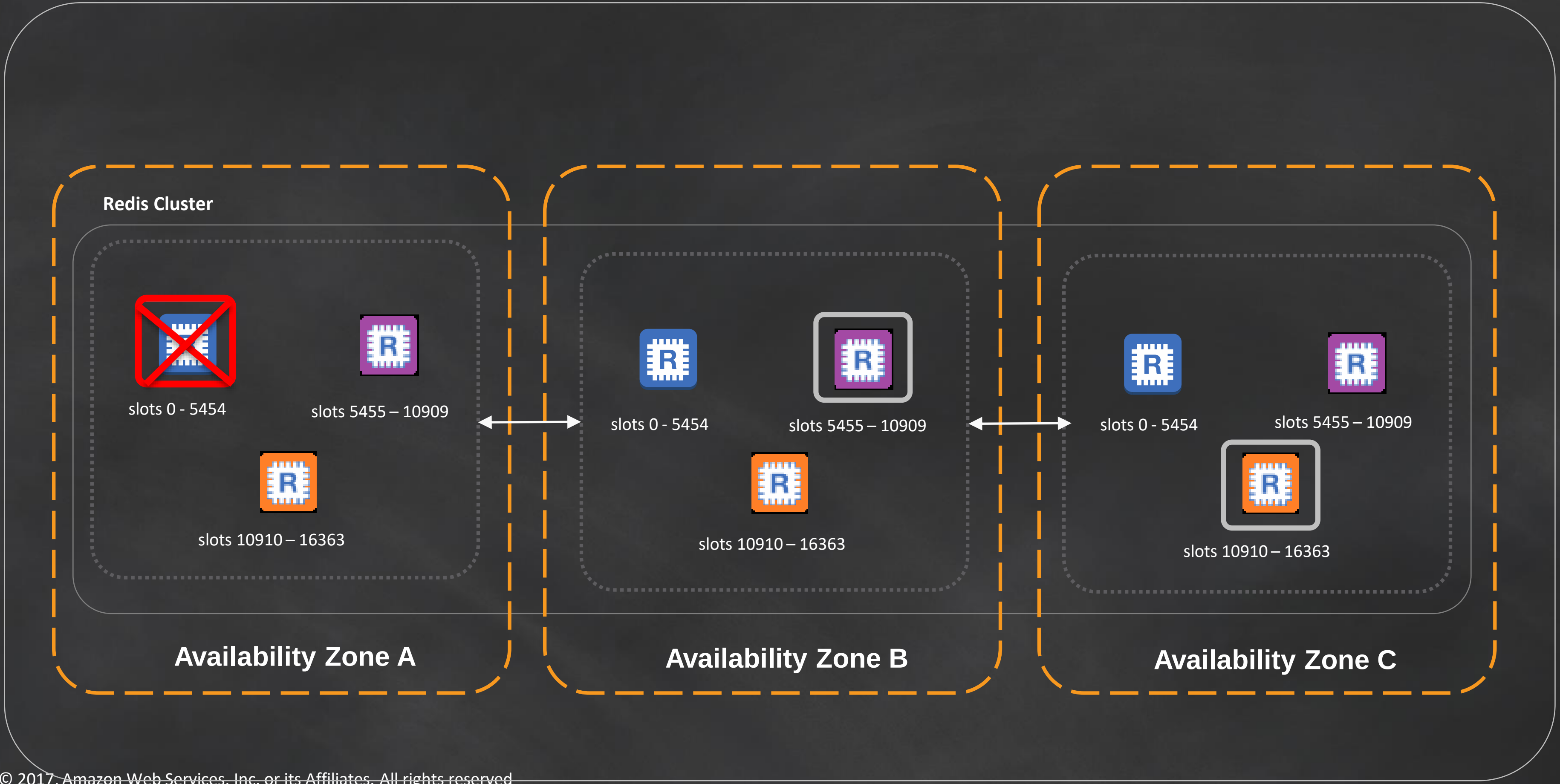
Availability zone(s) No preference

	Slots/Keyspaces	Primary	Replica 1	Replica 2
Shard 1	Equal distribution	No preference	No preference	No preference
Shard 2	Equal distribution	No preference	No preference	No preference
Shard 3	Equal distribution	No preference	No preference	No preference

ElastiCache for Redis Failure Scenarios

```
REDIS:6379> GET quote:failure

"Everything fails, all the time."

                          -- Werner Vogels, CTO Amazon.com --
```

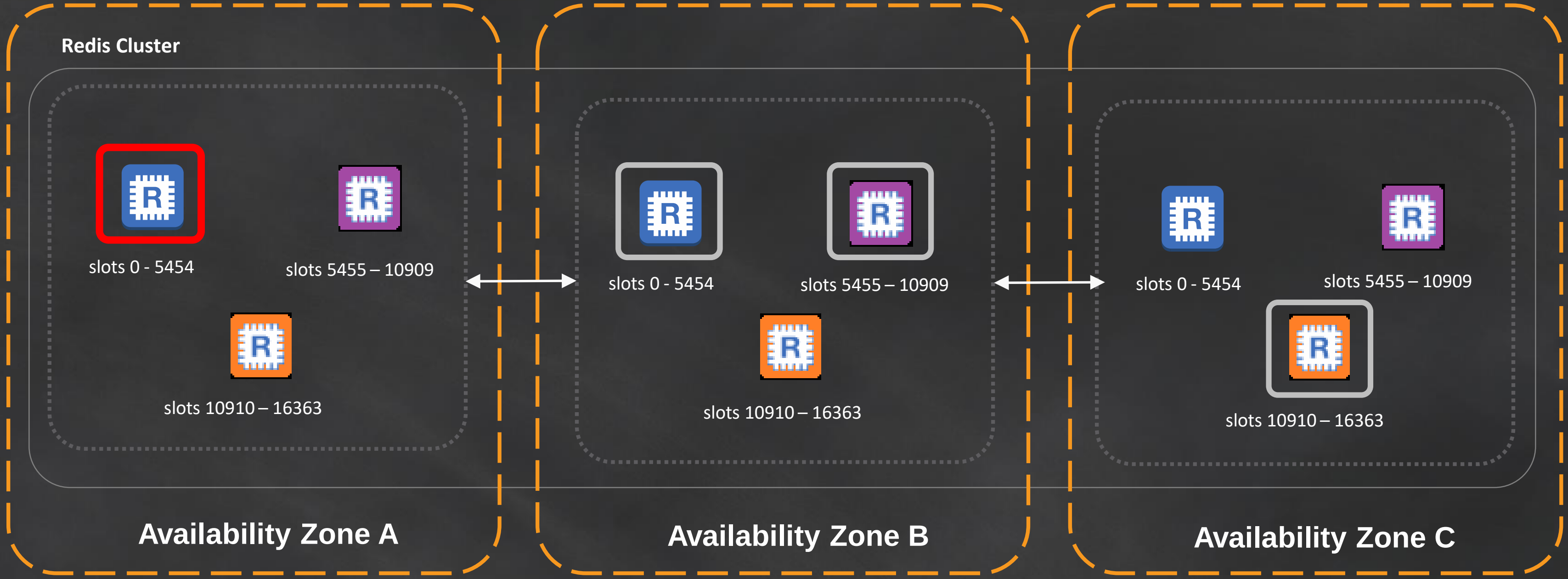
Scenario 1: Single Primary Failure



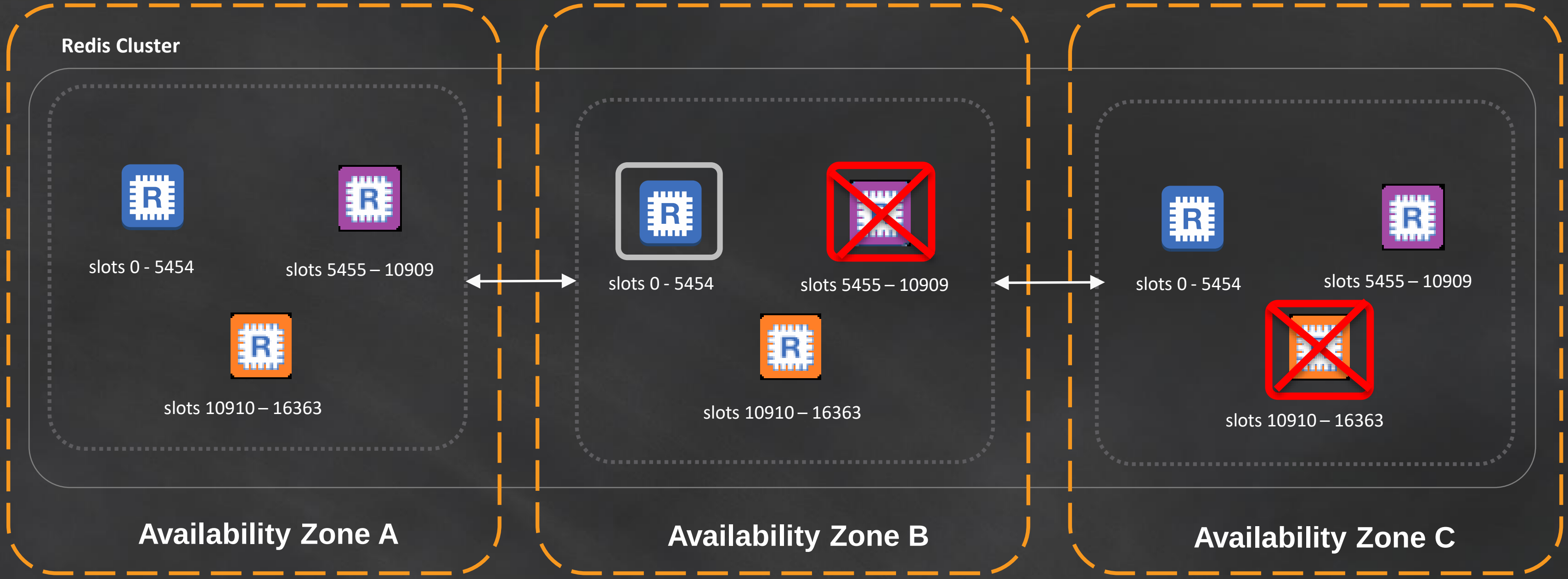
Scenario 1: Single Primary Failure

Mitigation:

1. Automatic Failure Detection & Replica Promotion (~15-30s)
2. Repair Failed Node



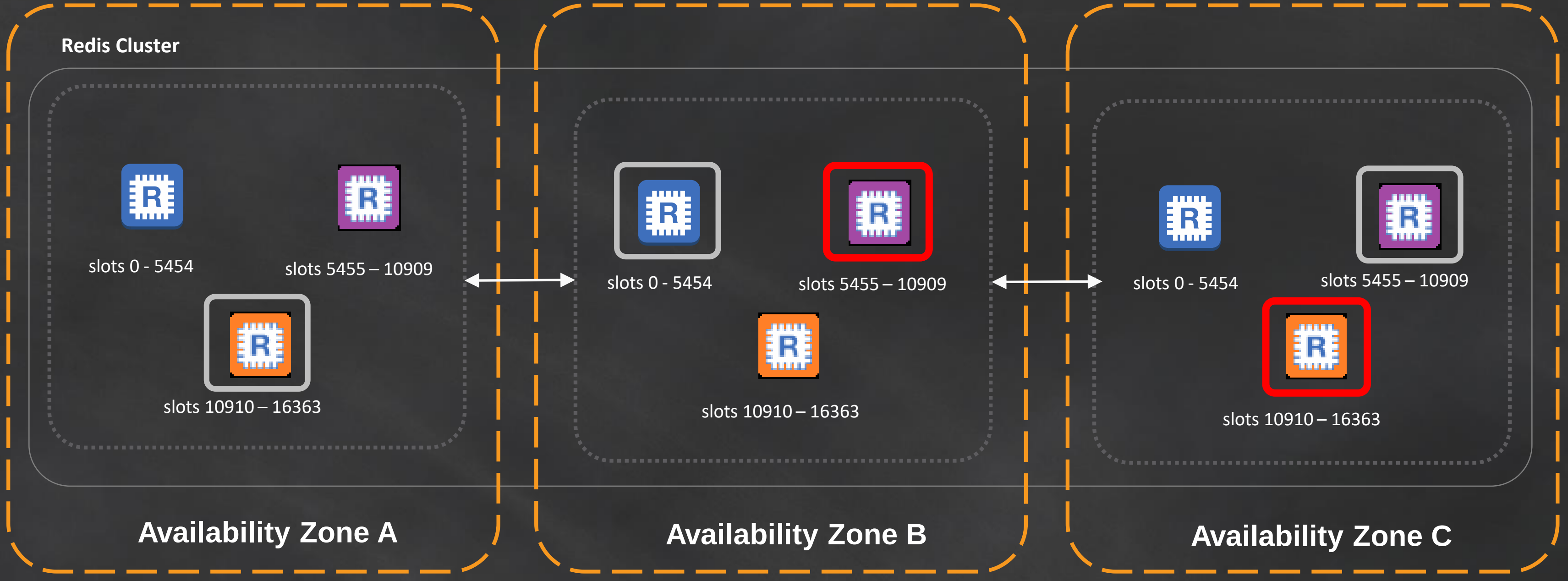
Scenario 2: Majority of Primaries Fail



Scenario 2: Majority of Primaries Fail

Mitigation: Redis enhancements on ElastiCache

- Automatic Failure Detection and Replica Promotion
- Repair Failed Nodes



REDIS CLUSTER-MODE ENABLED VS DISABLED

Feature	Enabled	Disabled
FAILOVER	15-30s (NON-DNS)	1.5m-2m (DNS BASED)
FAILOVER RISK	• WRITES effected - Partial dataset (less risk with more partitions) • READS available	• Writes effected on entire dataset. • READS available
PERFORMANCE	SCALES with cluster size (90 nodes – 15 primaries + 0-5 replicas per shard.	6 Nodes (1 Primary + 0-5 replicas)
MAX CONNECTIONS	• PRIMARIES (65K X 15 = 975,000) • REPLICAS (65K X 75 = 4,875,000)	• PRIMARY: 65K • REPLICAS: (65K x 5 = 325,000)
STORAGE	3.5TiB +	237GB
COST	Smaller nodes but more $$	Larger nodes less $
Example: Assume workload needs 175GB	9 x cache.r3.xlarge (0.455hr) = $4.095hr 255.6 GB	1 X cache.r3.8xlarge = $3.640 , 237 GB

ElastiCache Best Practices

Cluster Sizing Best Practices

- Storage – Clusters should have adequate Memory
  - Recommended: Memory needed + 25% reserved memory (for Redis) + some room for growth (optional 10%).
  - Optimize using eviction policies and TTLs
  - Scale up or out when before reaching max-memory using Cloudwatch alarms
  - Use memory optimized nodes for cost effectiveness
- Performance – Performance should not be compromised
  - Benchmark operations using Redis Benchmark tool
    - For more READIOPS – Add Replicas
    - For more WRITEIOPS – Add shards (scale out)
    - For more Network IO – Use network optimized instances and scale out
  - Use pipelining for bulk reads/writes
  - Consider Big(O) time complexity for data structure commands
- Cluster Isolation (apps sharing key space) – Chose a strategy that works for your workload
  - Identify what kind of isolation is needed based on the workload and environment
  - Isolation: No Isolation $ | Isolation by Purpose $$ | Full Isolation $$$

Redis Benchmark Tool

Open source utility to benchmark performance

- example: `src/redis-benchmark -h r3-xlarge-perf.foio87.0001.use1.cache.amazonaws.com -p 6379 -n -150000 -d 100`

Syntax:

redis-benchmark -h <host> -p <port> -c 50 -n 1000 -d 500 –q

-c <clients> - Specifies the number of parallel connections (default 50).

-n <requests> - Specifies the number of requests (default 1000000).

-d <size> - Specifies the data size of GET and SET values in bytes.

-t <test1,test2> - Comma separated list of tests to perform.

-q – Quiet operation, displays only the result.

Redis max-memory Policies

Select a max-memory policy based on your workload needs

- noeviction: return errors when the memory limit was reached and the client is trying to execute commands that could result in more memory to be used.
- allkeys-lru: evict keys trying to remove the less recently used (LRU) keys first.
- volatile-lru: evict keys trying to remove the less recently used (LRU) keys first, but only among keys that have an expire set.
- allkeys-random: evict random keys in order to make space for the new data added.
- volatile-random: evict random keys in order to make space for the new data added, but only evict keys with an expire set.
- volatile-ttl: evict only keys with an expire set, and try to evict keys with a shorter time to live (TTL) first.

The policies volatile-lru, volatile-random and volatile-ttl behave like noeviction if there are no keys to evict matching the prerequisites.

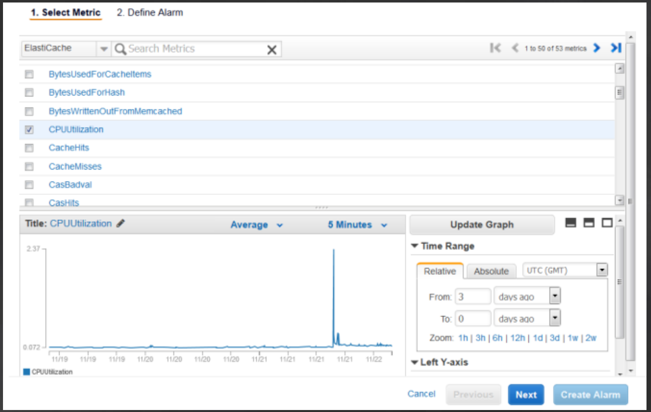
Architecting for Availability & Performance

- Upgrade to the latest engine version – 3.2.4
- Set reserved-memory to 25-30% of total available memory
- Swap usage should be zero or very low. Scale if not.
- Put read-replicas in a different AZ from the primary
- For important workloads use 2 read replicas per primary
- Write to the primary, read from the read-replicas
- Take snapshots from read-replicas
- For Redis Cluster have odd number of shards.
- Use newer Intel processors for best IO performance
- Use Failover API to environment

Key ElastiCache CloudWatch Metrics

- CPUUtilization
  - Memcached – up to 90% ok
  - Redis – divide by cores (ex: 90% / 4 = 22.5%)
- SwapUsage low
- CacheMisses / CacheHits Ratio low / stable
- Evictions near zero
  - Exception: Russian doll caching
- CurrConnections stable

- Setup alarms with CloudWatch Metrics



1. Select Metric 2. Define Alarm

ElastiCache | Search Metrics | 1 to 50 of 53 metrics

- □ BytesUsedForCacheItems
- □ BytesUsedForHash
- □ BytesWrittenOutFromMemcached
- ☑ CPUUtilization
- □ CacheHits
- □ CacheMisses
- □ CasBadval
- □ CasHits

Title: CPUUtilization | Average | 5 Minutes

Time Range: Relative | UTC (GMT)

From: 3 days ago

To: 0 days ago

- Whitepaper: http://bit.ly/elasticache-whitepaper

ElastiCache Modifiable Parameters

- Maxclients: 65000 (unchangeable)
  - Use connection pooling
  - timeout – Closes a connection after its been idle for a given interval
  - tcp-keepalive – Detects dead peers given an interval
- Databases: 16 (Default)
  - Logical partition
- Reserved-memory: 0 (Default)
  - Recommended
    - 50% of maxmemory to use before 2.8.22
    - 25% after 2.8.22 – ElastiCache
- Maxmemory-policy:
  - The eviction policy for keys when maximum memory usage is reached.
  - Possible values: volatile-lru, allkeys-lru, volatile-random, allkeys-random, volatile-ttl, noeviction

Amazon ElastiCache

Common Usage Patterns

redis

Expedia

Grab

News Corp

BANDAI NAMCO BANDAI NAMCO Studios

vodafone™

Experian

sumologic

TRUECar™

ThermoFisher SCIENTIFIC

SCHOLASTIC

ancestry

intuit

mlbam

scrippsnetworks interactive

Healthguru

Kenshoo™

gogo

Prezi

KICKSTARTER

runkeeper

Loggly

duolingo

A&E

dash

ZIFF DAVIS

instacart

KASPERSKY lab

univision

cookpad

99designs

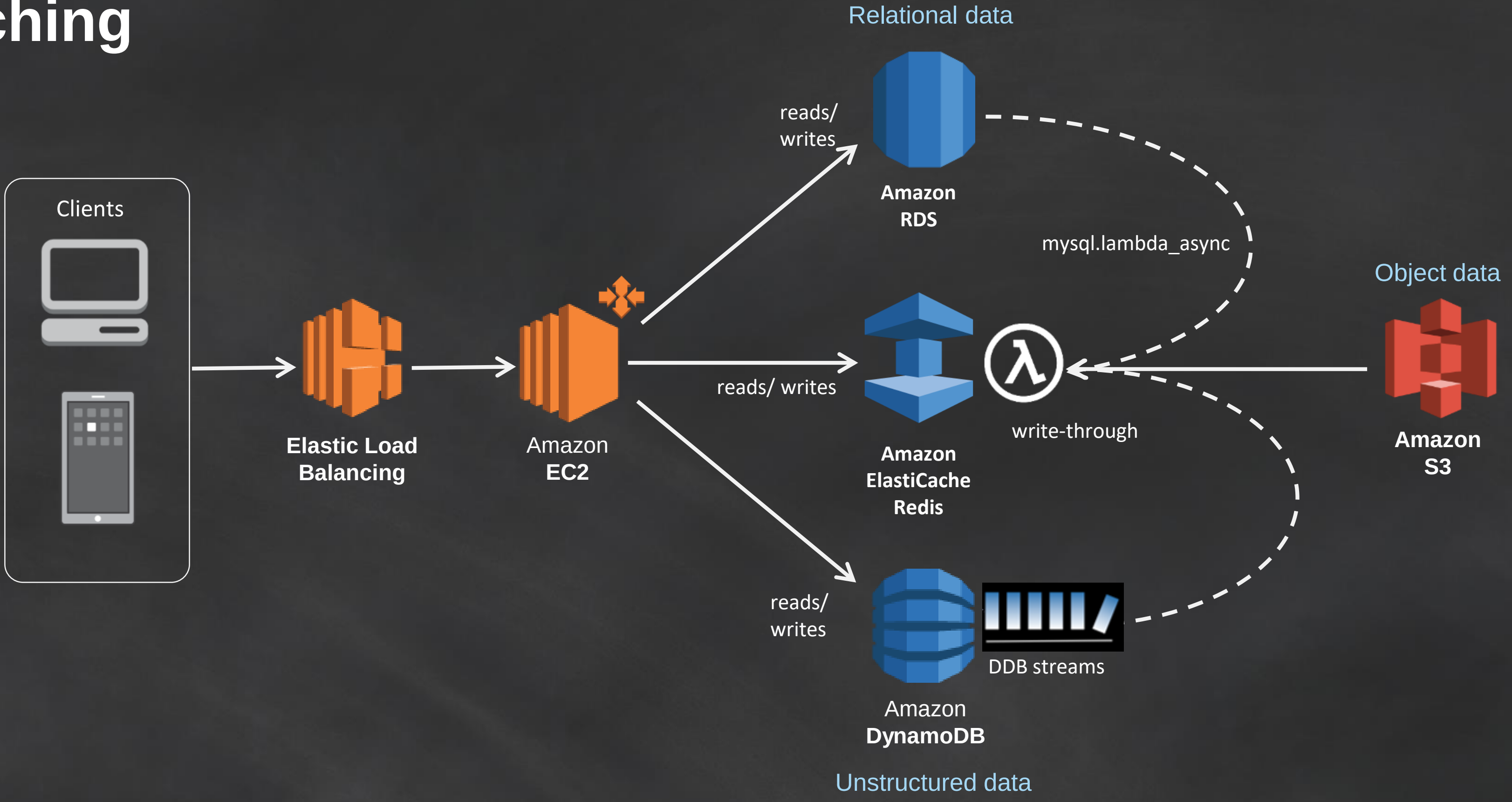
GREE

Fender

Usage Patterns

- Session Management
- Database Caching
- APIs (HTTP responses)
- IOT
- Streaming Data Analytics (Filtering/Aggregation)
- Pub/Sub
- Social Media (Sentiment Analysis)
- Standalone Database (Metadata Store)
- Leaderboards

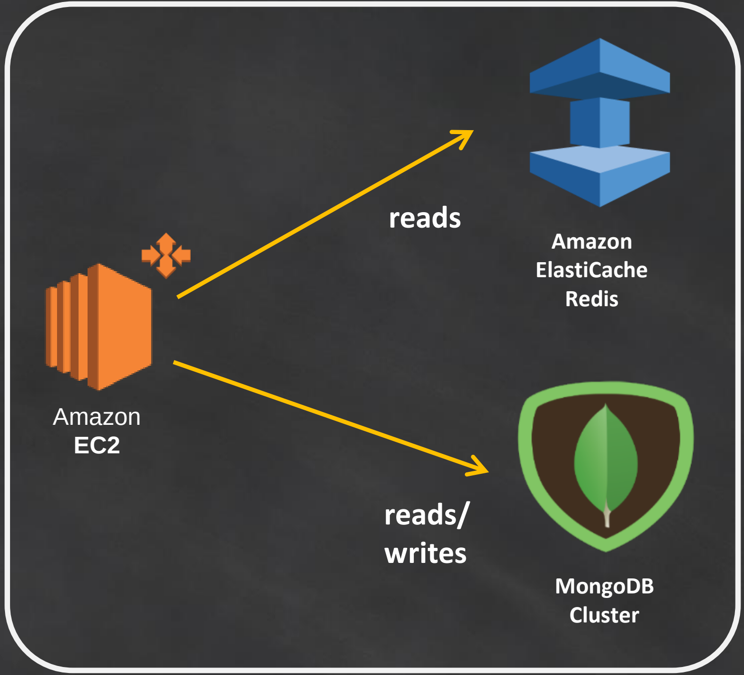
Caching



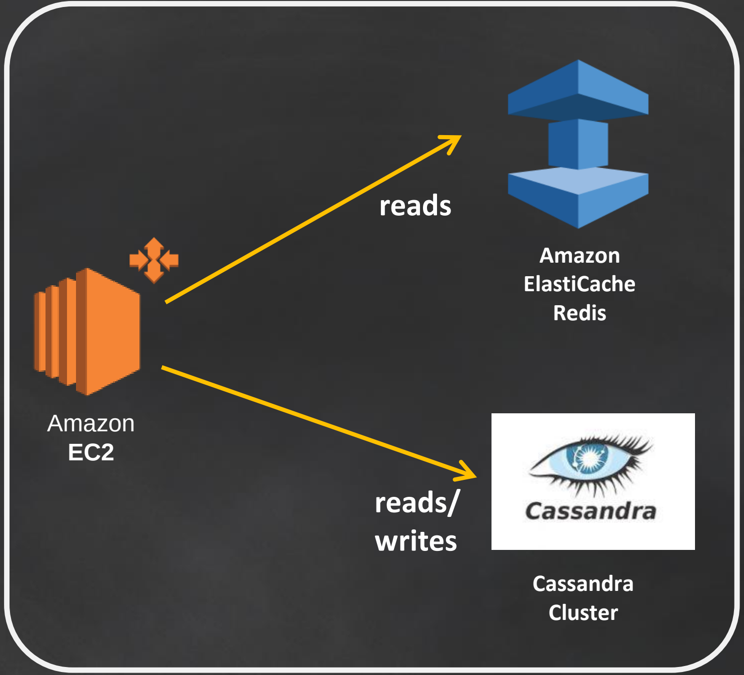
https://aws.amazon.com/caching/database-caching/

Caching NoSQL Databases with ElastiCache

- ✓ **Smaller NoSQL DB Clusters needed = Lower Costs**
- ✓ **Faster Data Retrieval = Better Performance**



DBObject doc = collection.findOne();
Cache Serialized DBObject in Redis (Good)
Cache rows in Redis Hash (Faster/More efficient)



ResultSet rs = session.execute(stmt);
Cache Serialized ResultSet in Redis (Good)
Cache rows in Redis Hash (Faster/More efficient)

Session Caching

PHP Example

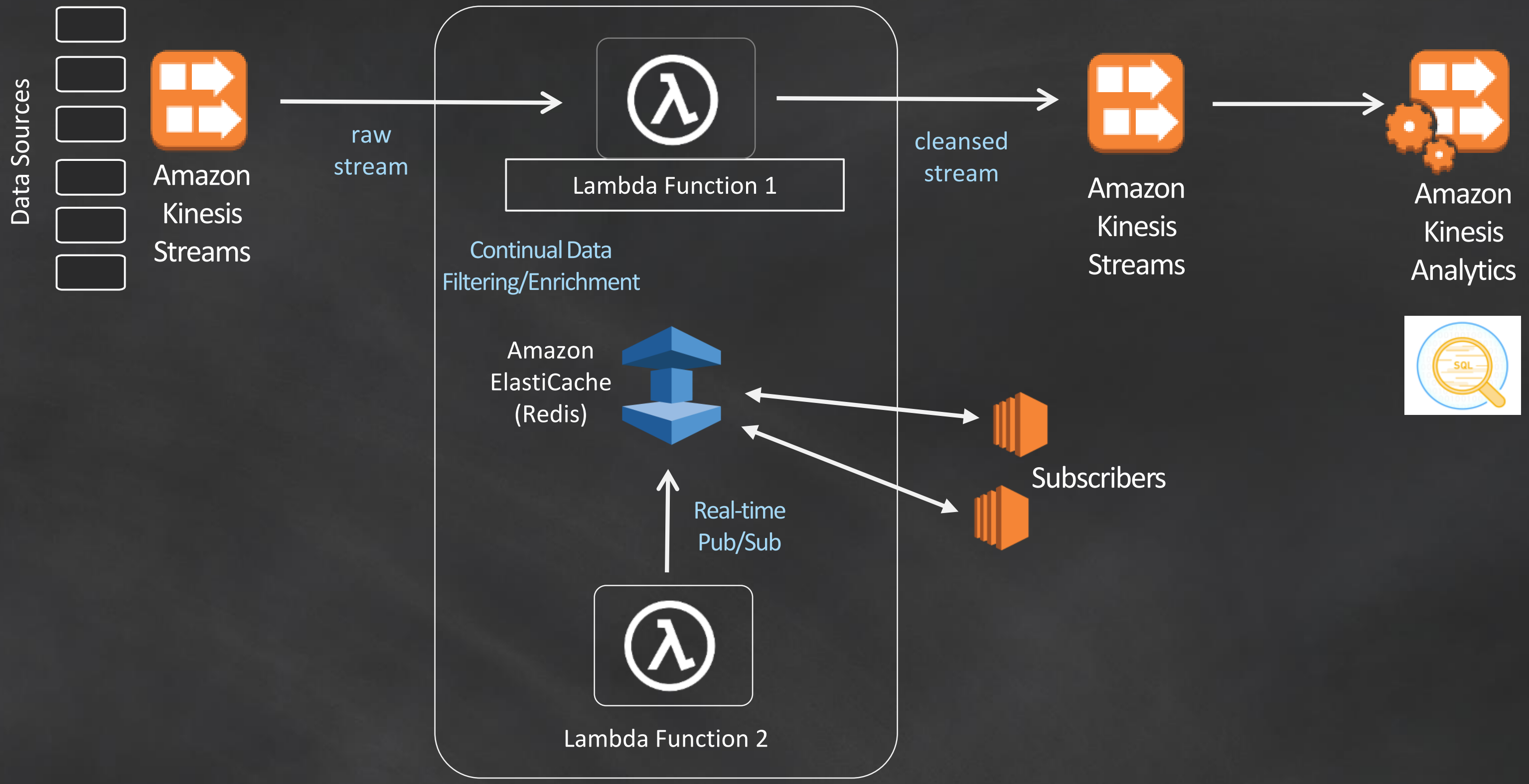
- For situations where you need an external session store
- Especially needed when using ASGs
- Cache is optimal for high-volume reads

Auto Scaling group

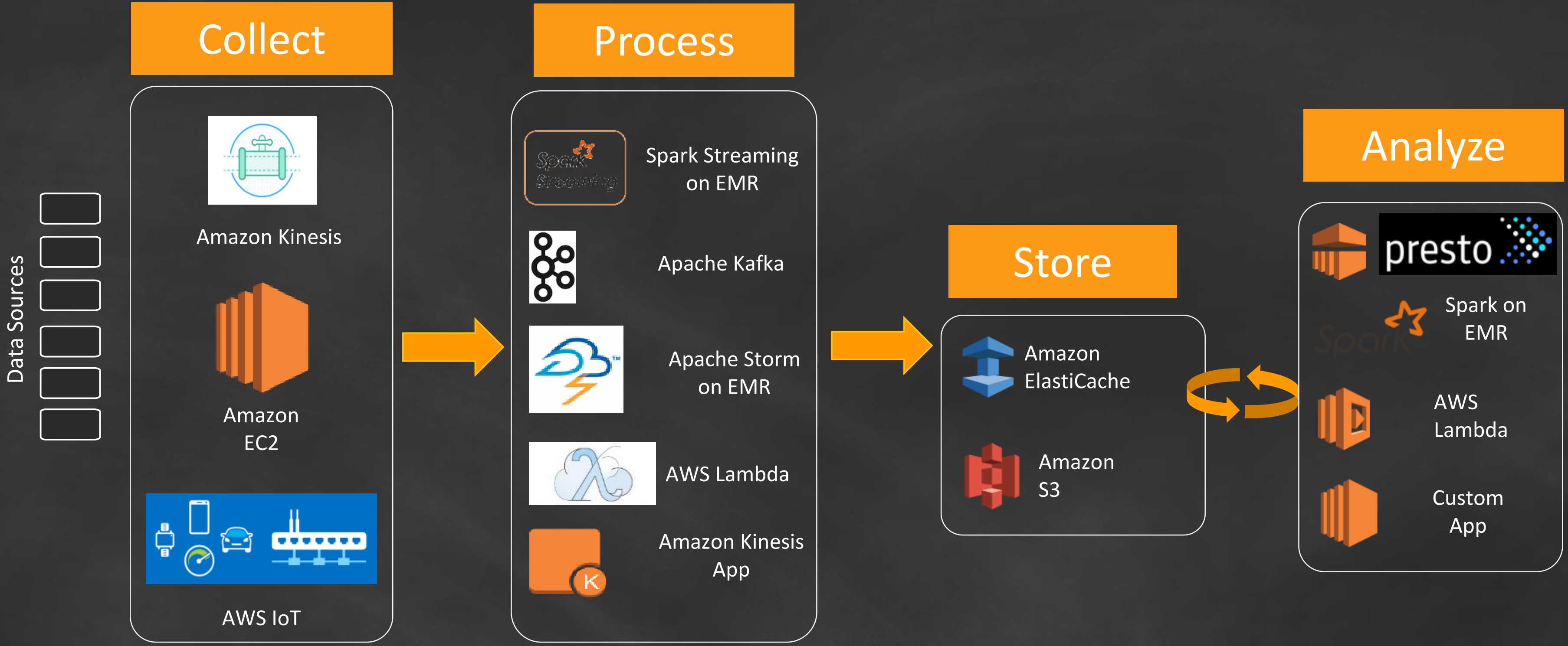
- 1) Install php, apache php memcache client
- e.g. yum install php apache php-pecl-memcache
- 2) Configure "php.ini"
- session.save_handler = memcache
- session.save_path=
- "tcp://node1:11211, tcp://node2:11211"
- 3) Configure "php.d/memcache.ini"
- memcache.hash_strategy = consistent
- memcache.allow_failover = 1
- memcache.session_redundancy=3*
- 4) Restart httpd
- 5) Begin using Session Data:

https://github.com/mikelabib/elasticache-memcached-php-demo

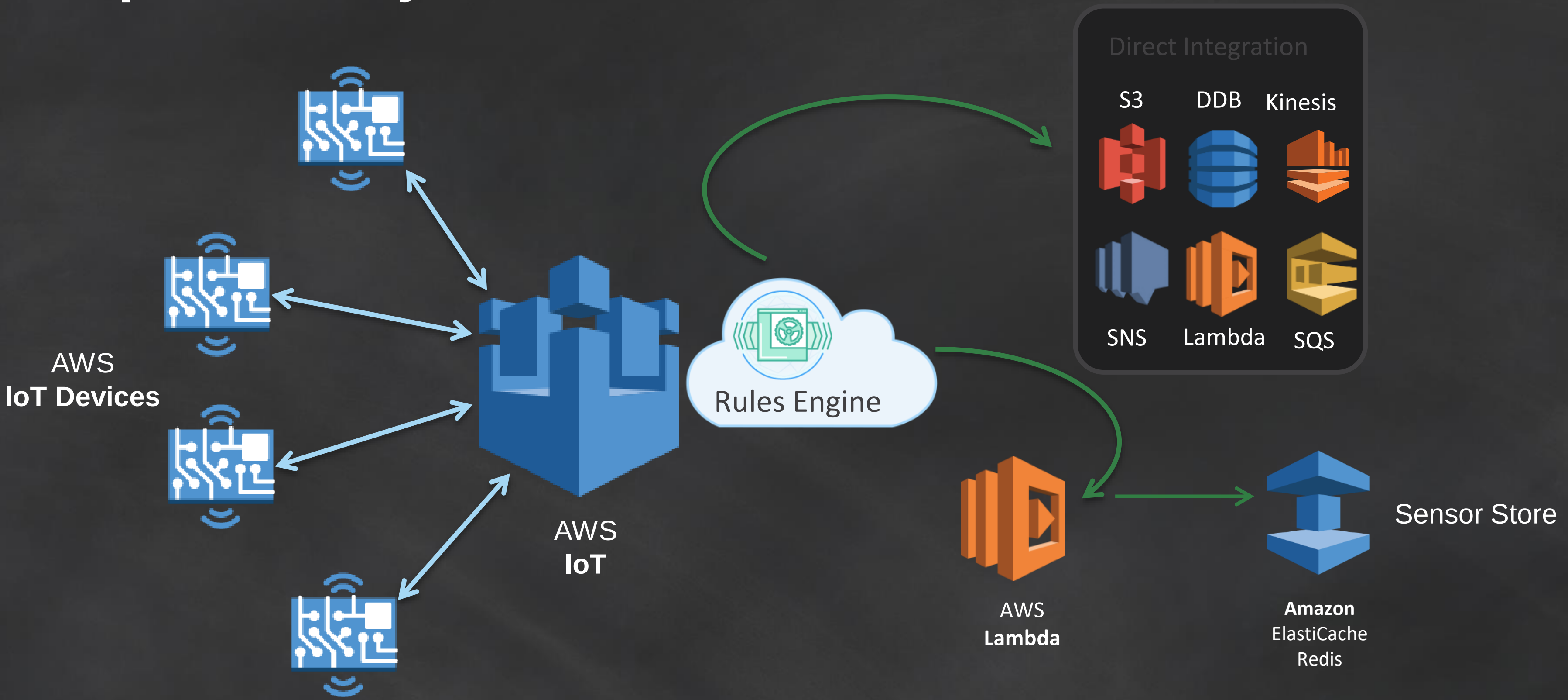
Streaming Data Enrichment / Processing



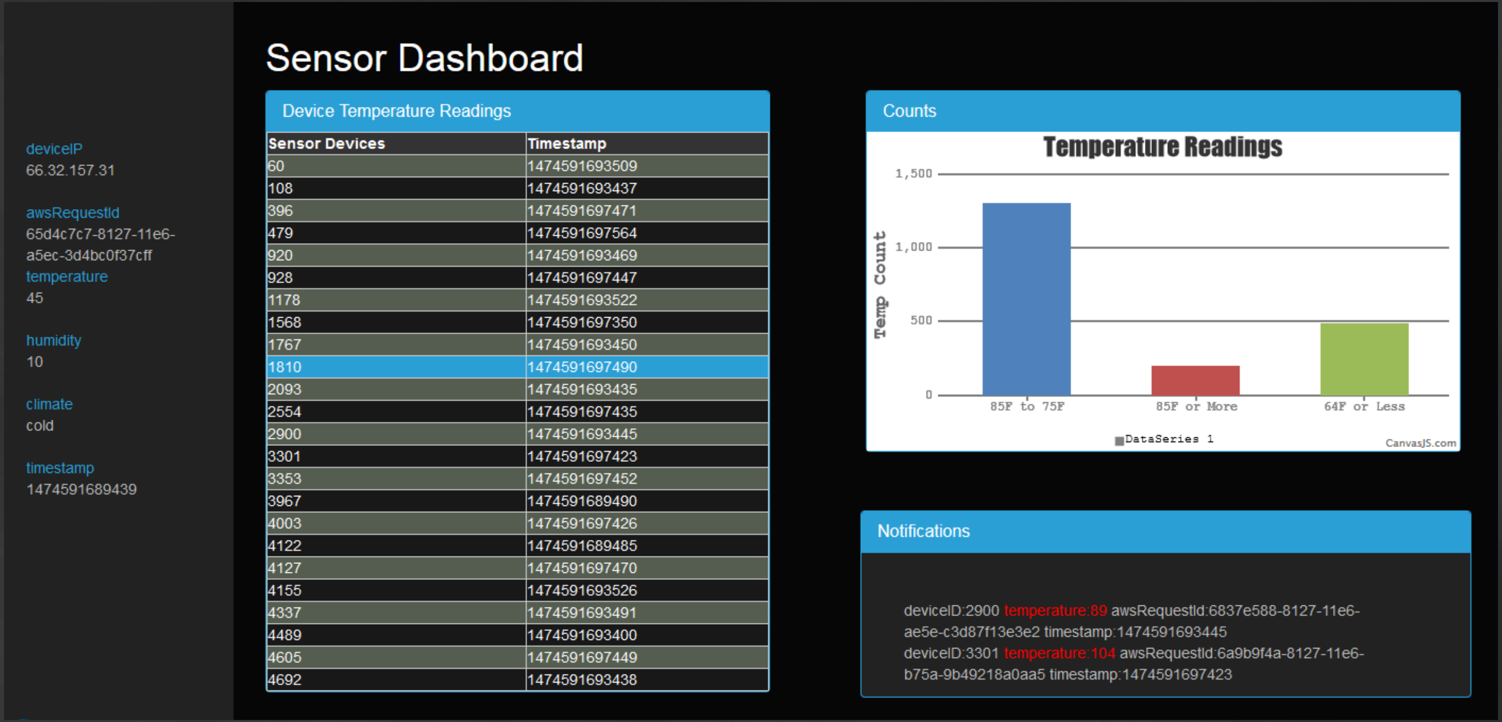
Big Data Architectures using Redis



IoT powered by ElastiCache



IoT Demo – AWS IoT + Lambda + ElastiCache



deviceIP
66.32.157.31

awsRequestId
65d4c7c7-8127-11e6-a5ec-3d4bc0f37cff

temperature
45

humidity
10

climate
cold

timestamp
1474591689439

Sensor Dashboard

Device Temperature Readings

Sensor Devices	Timestamp
60	1474591693509
108	1474591693437
396	1474591697471
479	1474591697564
920	1474591693469
928	1474591697447
1178	1474591693522
1568	1474591697350
1767	1474591693450
1810	1474591697490
2093	1474591693435
2554	1474591697435
2900	1474591693445
3301	1474591697423
3353	1474591697452
3967	1474591689490
4003	1474591697426
4122	1474591689485
4127	1474591697470
4155	1474591693526
4337	1474591693491
4489	1474591693400
4605	1474591697449
4692	1474591693438

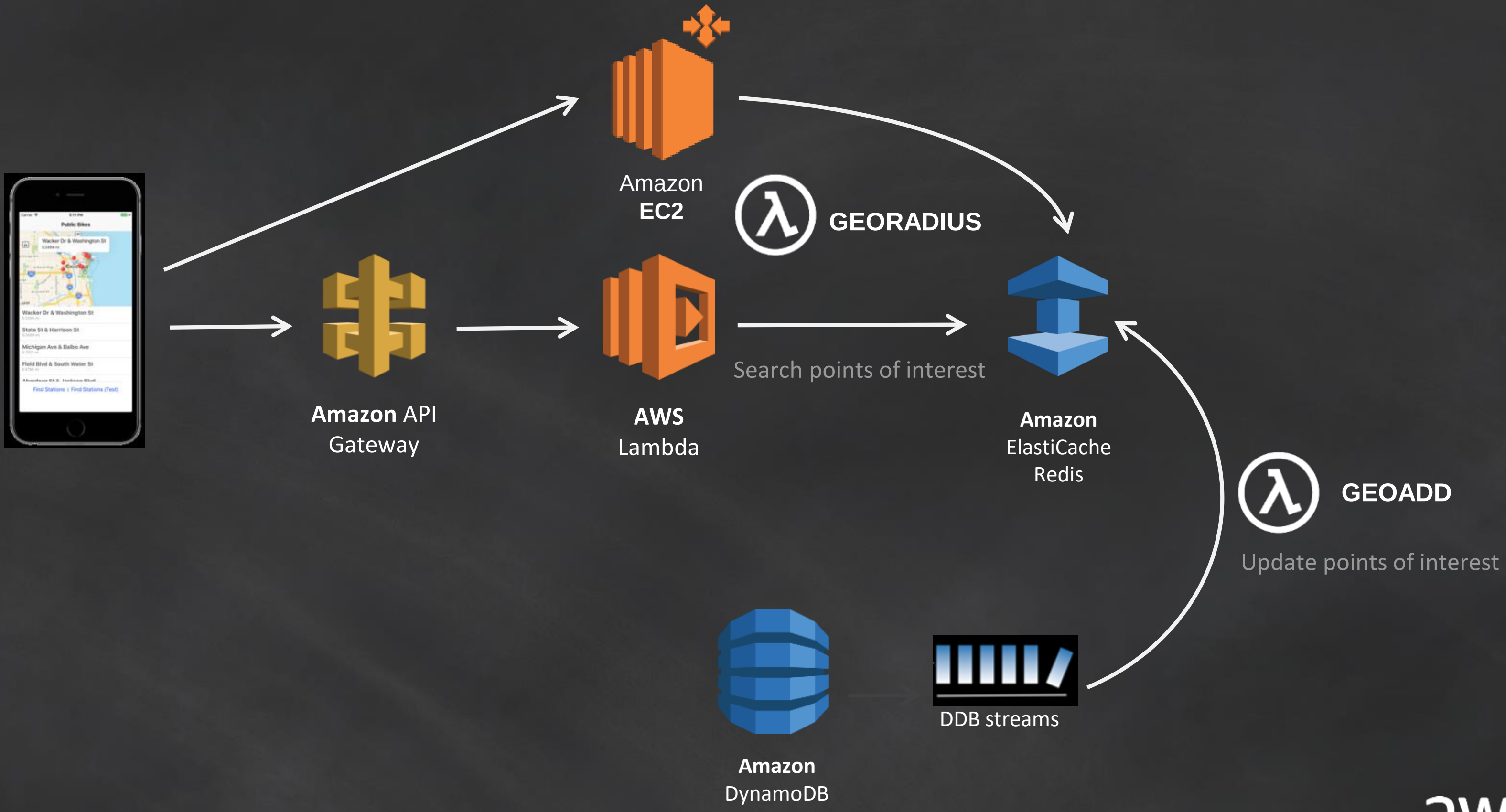
Counts

Notifications

deviceID:2900 temperature:89 awsRequestId:6837e588-8127-11e6-ae5e-c3d87f13e3e2 timestamp:1474591693445
deviceID:3301 temperature:104 awsRequestId:6a9b9f4a-8127-11e6-b75a-9b49218a0aa5 timestamp:1474591697423

https://aws.amazon.com/blogs/database/managing-iot-and-time-series-data-with-amazon-elasticache-for-redis/

Mobile Apps Powered by ElastiCache


Carrier
5:11 PM
Public Bikes
Wacker Dr & Washington St
Chicago
Wacker Dr & Washington St
State St & Harrison St
Michigan Ave & Balbo Ave
Field Blvd & South Water St
Find Stations | Find Stations (Test)

Amazon **EC2**

GEORADIUS

Amazon API Gateway

AWS Lambda

Search points of interest

Amazon ElastiCache Redis

GEOADD

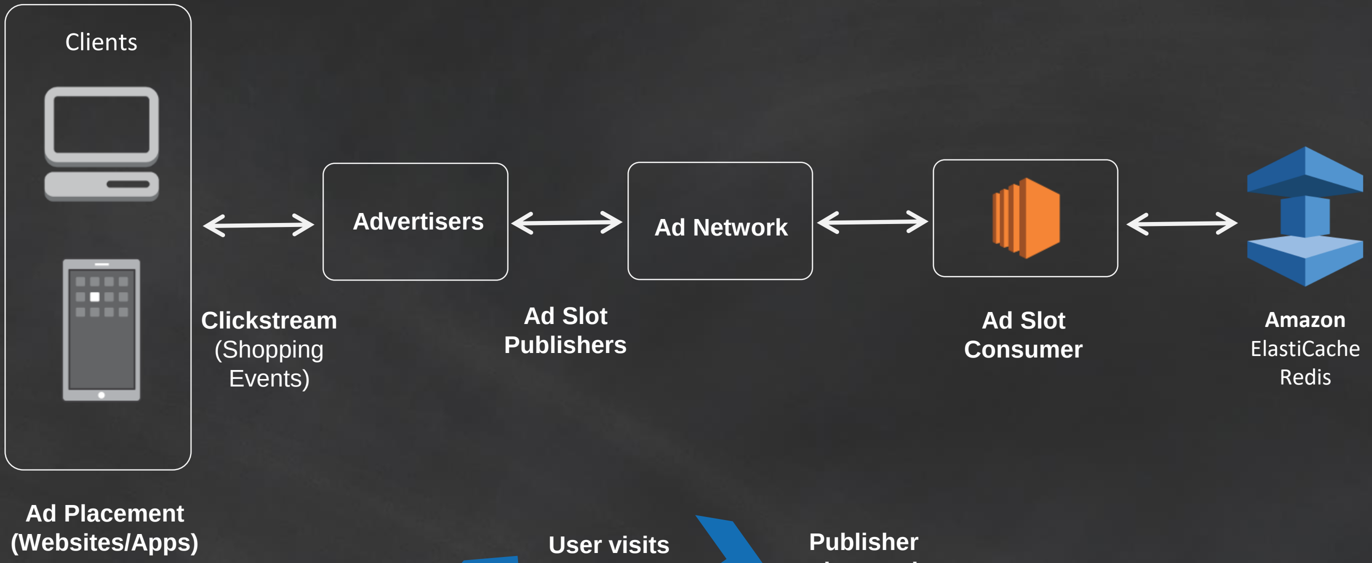
Update points of interest

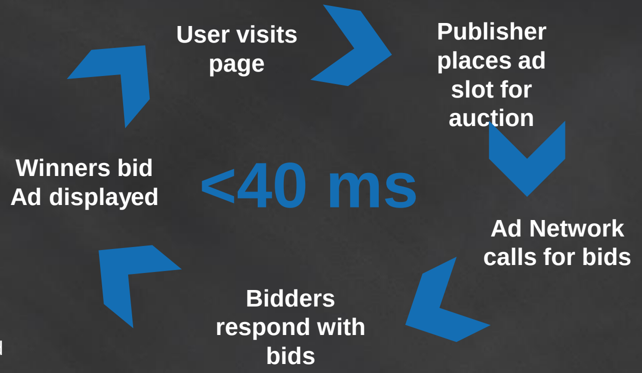
Amazon DynamoDB

DDB streams

https://aws.amazon.com/blogs/database/amazon-elasticache-utilizing-redis-geospatial-capabilities/

Ad Tech

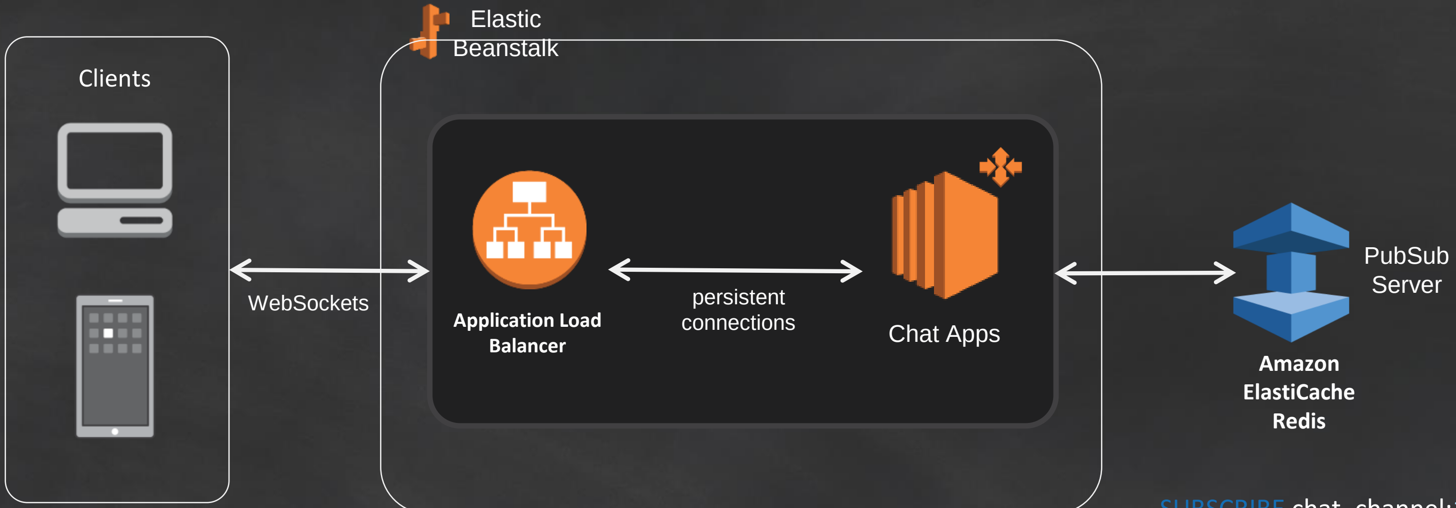




User visits page
Publisher places ad slot for auction
Ad Network calls for bids
Bidders respond with bids
Winners bid Ad displayed
<40 ms

https://aws.amazon.com/caching/database-caching/

Chat Apps powered by ElastiCache



```
SUBSCRIBE chat_channel:114
PUBLISH chat_channel:114 "Hello all"
>> ["message", "chat_channel:114", "Hello all"]
UNSUBSCRIBE chat_channel:114
```

https://aws.amazon.com/blogs/database/amazon-elasticache-utilizing-redis-geospatial-capabilities/

Gaming - Real-time Leaderboard

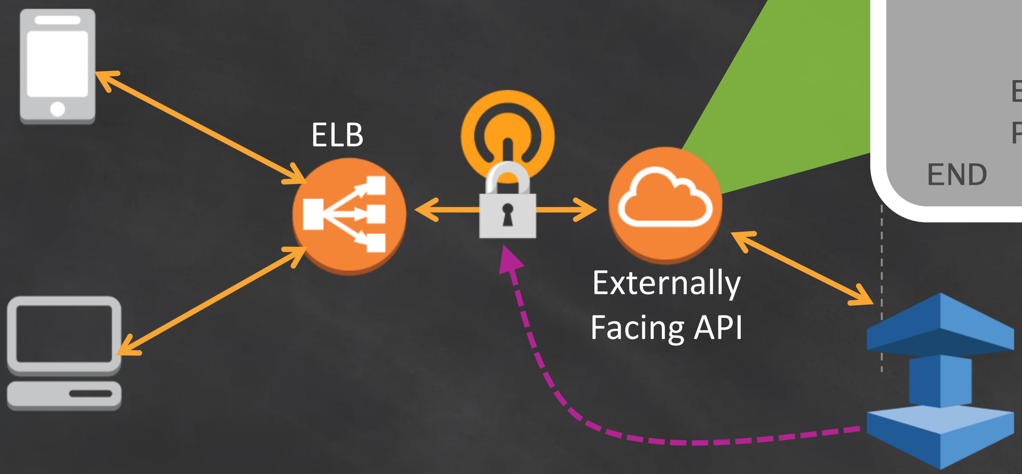
- Very popular for gaming apps which need uniqueness + ordering.
- Easy with Redis Sorted Sets

```
ZADD "leaderboard" 1201 "Gollum”
ZADD "leaderboard" 963  "Sauron"
ZADD "leaderboard" 1092 "Bilbo"
ZADD "leaderboard" 1383 "Frodo”

ZREVRANGE "leaderboard" 0 -1
1) "Frodo"
2) "Gollum"
3) "Bilbo"
4) "Sauron”

ZREVRANK "leaderboard" "Sauron"
(integer) 3
```

Rate Limiting

- Ex: Throttling requests to an API
- Leverages Redis Counters

```
FUNCTION LIMIT_API_CALL(APIaccesskey)
limit = HGET(APIaccesskey, “limit”)
time  = CURRENT_UNIX_TIME()
keyname = APIaccesskey + ":” + time
count = GET(keyname)
IF current != NULL && count > limit THEN
    ERROR ”API request limit exceeded"
ELSE
    MULTI
        INCR(keyname)
        EXPIRE(keyname,10)
    EXEC
    PERFORM_API_CALL()
END
```



Reference: http://redis.io/commands/INCR

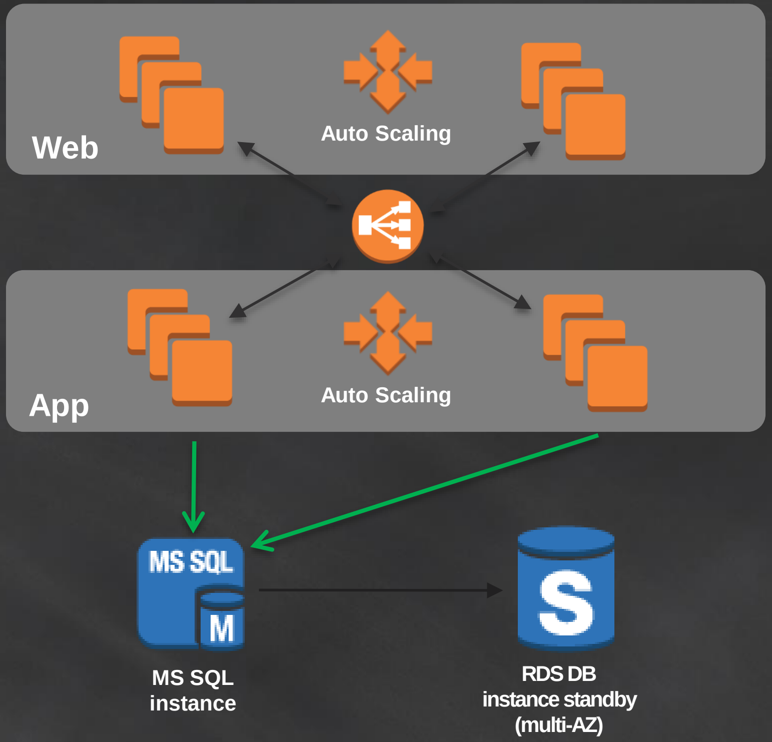
Recommendation Engine - Ratings

- Popular for recommendation engines and message board ranking
- Redis counters – increment likes/dislikes
- Redis hashes – list of everyone’s ratings
- Process with algorithm like Slope One or Jaccardian similarity
- Ruby example - https://github.com/davidcelis/recommendable

```
INCR item:38927:likes
HSET item:38927:ratings "Susan" 1

INCR item:38927:dislikes
HSET item:38927:ratings "Tommy" -1
```

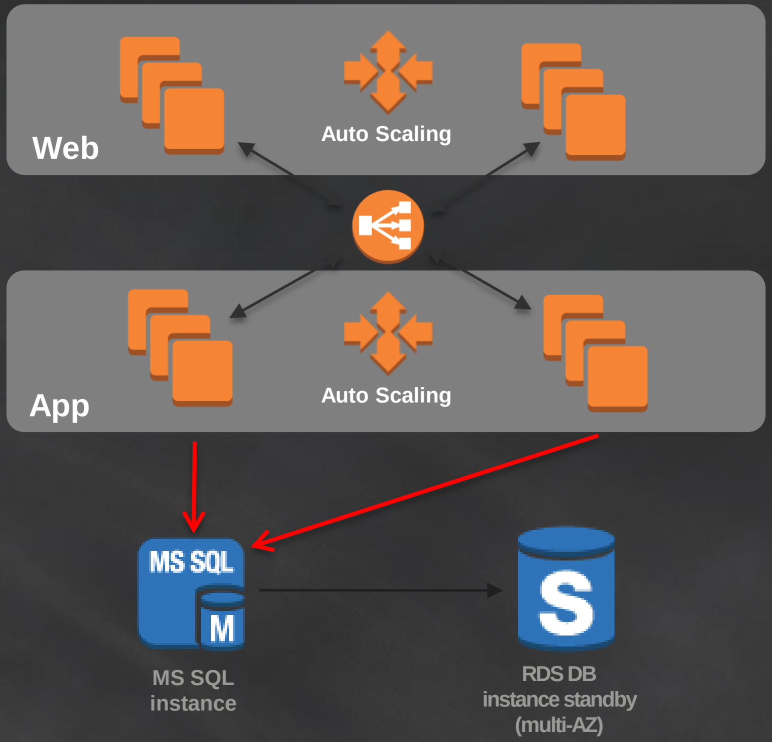
Database Caching Strategies

WHY CACHING



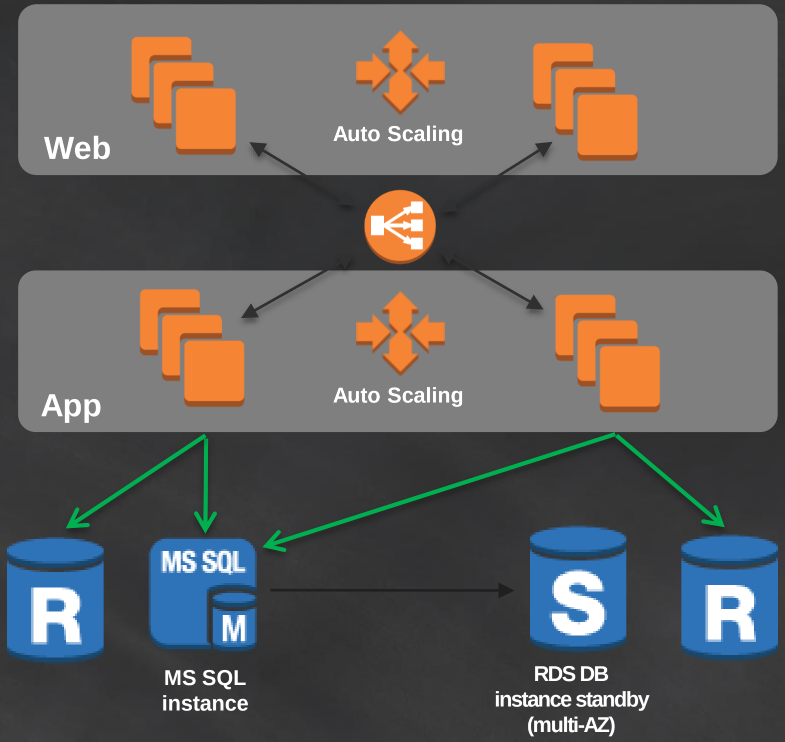
✓ Scalability
✓ Durability
Latency
✓ Cost

WHY CACHING



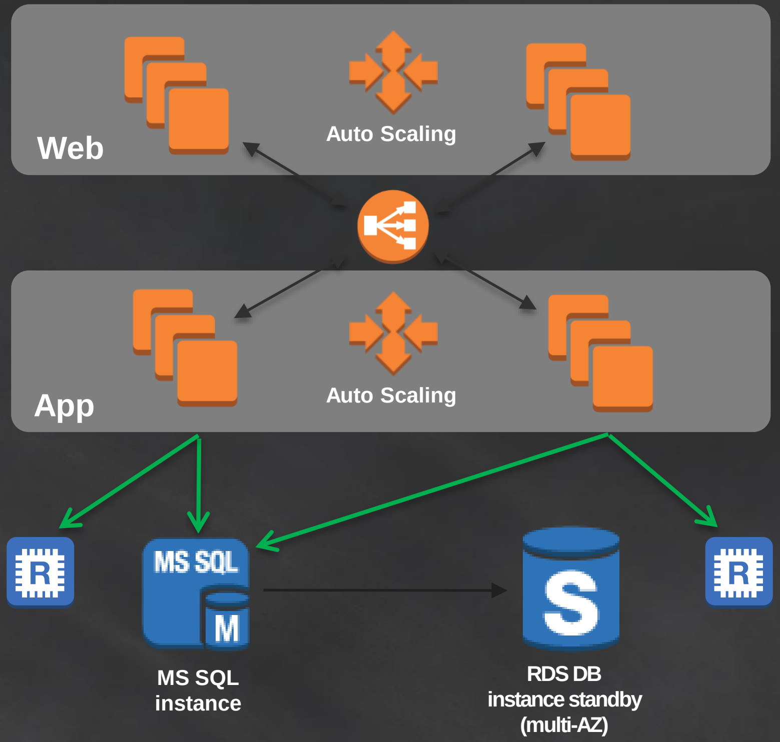
Scalability
✓ Durability
Latency
✓ Cost

WHY CACHING



- ✓ Scalability
- ✓ Durability
- Latency
- Cost

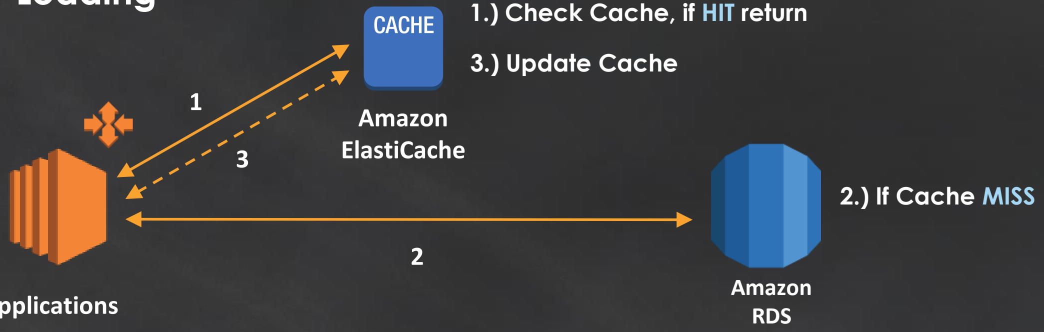
WHY CACHING



- ✓ Scalability
- ✓ Durability
- ✓ Latency
- ✓ Cost

Caching - Patterns

Cache-Aside - Lazy Loading



1.) Check Cache, if HIT return

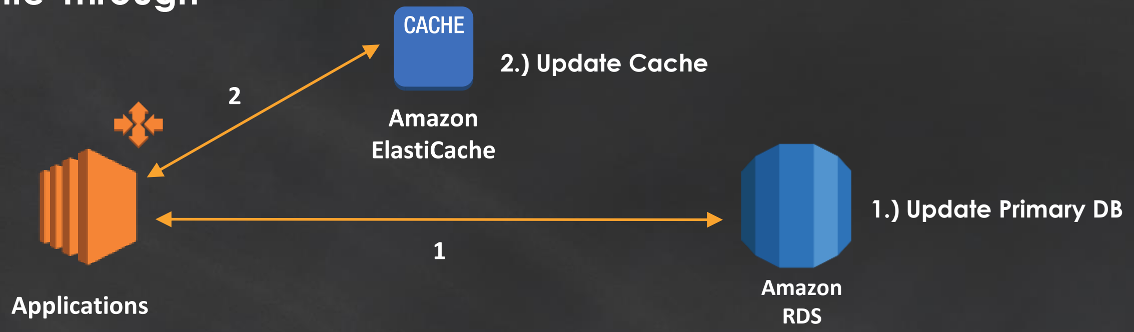
3.) Update Cache

2.) If Cache MISS

https://d0.awsstatic.com/whitepapers/Database/database-caching-strategies-using-redis.pdf

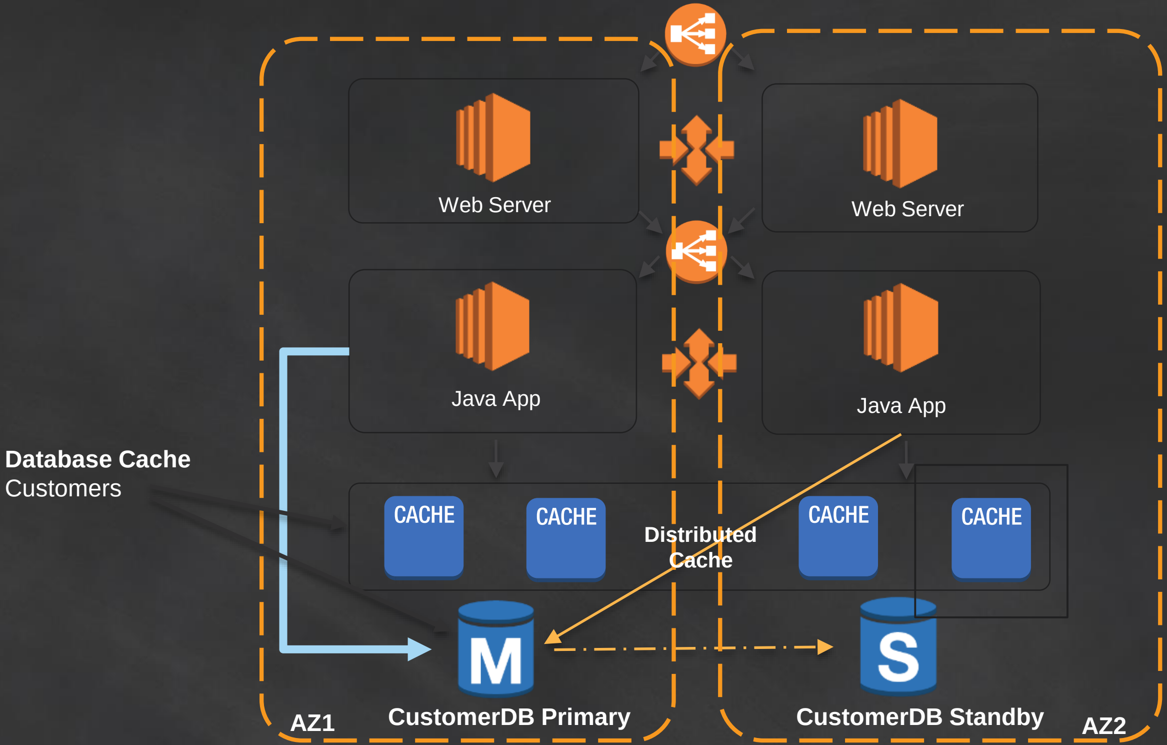
Caching - Patterns

Cache-Aside - Write-Through



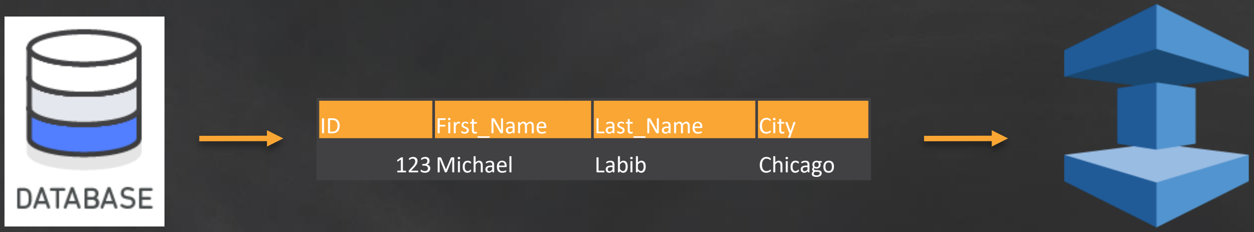
https://d0.awsstatic.com/whitepapers/Database/database-caching-strategies-using-redis.pdf

Example workload topology: Customer Data



Caching Strategies

1. Cache Database SQL ResultSet (Row)



ID	First_Name	Last_Name	City
123	Michael	Labib	Chicago

SELECT * FROM x WHERE y

ResultSet Object (ROW)

Key: Query, Value: CRS as byte array

PRO

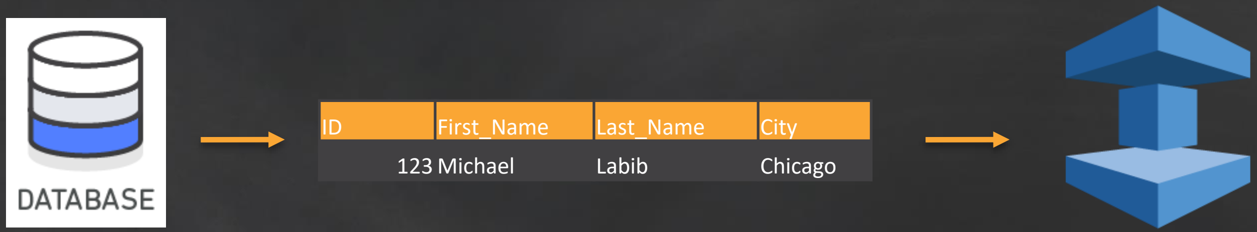
When data retrieval logic is abstracted from the code consuming the ResultSet, caching the ROW can be extremely effective and can be implemented against any RDBMS.

CON

- Does not speed up processing time

Caching Strategies

2. Cache database values into custom format in a Redis String



ID	First_Name	Last_Name	City
123	Michael	Labib	Chicago

SELECT * FROM x WHERE y

String firstName = rs.getString(First_Name)

Key: 123, Value: '{ "firstname": "Michael",
"lastname": "Labib",
"city": "Chicago" } '

PRO

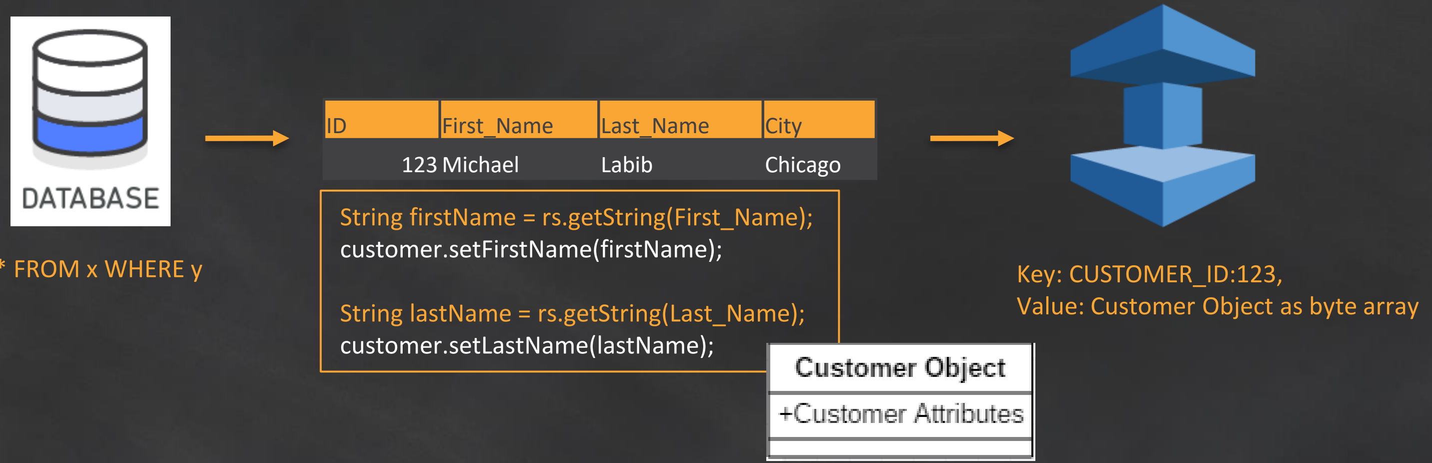
Very easy to implement. Cache any desired database fields and values into a Redis String. For example, store your retrieved data into a JSON object stored in a Redis String.

CON

No JSON specific query support

Caching Strategies

3. Cache serialized application object (e.g. Java Object)



SELECT * FROM x WHERE y

ID	First_Name	Last_Name	City
123	Michael	Labib	Chicago

```
String firstName = rs.getString(First_Name);
customer.setFirstName(firstName);

String lastName = rs.getString(Last_Name);
customer.setLastName(lastName);
```

Key: CUSTOMER_ID:123,
Value: Customer Object as byte array

PRO

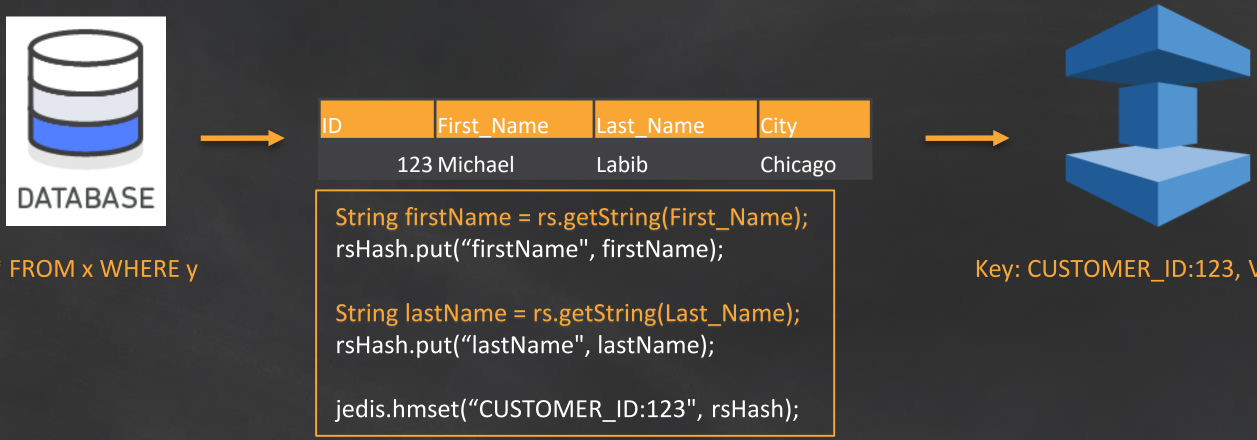
Utilize application objects in their native structure and data state when serialized.

CON

Advanced application development use case.

Caching Strategies

4. Leverage advanced Redis Data Structures for cached data



SELECT * FROM x WHERE y

ID	First_Name	Last_Name	City
123	Michael	Labib	Chicago

```
String firstName = rs.getString(First_Name);
rsHash.put(“firstName", firstName);

String lastName = rs.getString(Last_Name);
rsHash.put(“lastName", lastName);

jedis.hmset(“CUSTOMER_ID:123", rsHash);
```

Key: CUSTOMER_ID:123, Value: rsHash

PRO

In addition to reducing data retrieval latency, cache data into specific data structure that simplifies the data access pattern.

aws | Pop-up Loft

Everything and Anything Startups Need to Get Started on AWS

aws.amazon.com/activate