AMAZON ELASTICACHE

14-848 CLOUD COMPUTING, FALL 2018

KESDEN

Today's lecture is a very slight adaptation of the first part of the following paper, from which it borrows most language and illustrations:

- Wiger, Nate, *Performance at Scale With Amazon ElastiCache*, Amazon, May 2015.
- https://d0.awsstatic.com/whitepapers/performance-at-scale-with-amazon-elasticache.pdf

REMEMBER SOCRATIVE

- https://api.socrative.com/rc/Nfu6Lp

OVERVIEW

- In-memory caching improves application performance by storing frequently accessed data items in memory, so that they can be retrieved without access to the primary data store.
- Properly leveraging caching can result in an application that not only performs better, but also costs less at scale.
- Amazon ElastiCache is a managed service that reduces the administrative burden of deploying an in-memory cache in the cloud.
- Beyond caching, an in-memory data layer also enables advanced use cases, such as analytics and recommendation engines.

ROLE OF AMAZON ELASTICACHE

- The Amazon ElastiCache architecture is based on the concept of deploying one or more cache clusters for your application.
- Once your cache cluster is up and running, the service automates common administrative tasks such as resource provisioning, failure detection and recovery, and software patching.
- Amazon ElastiCache provides detailed monitoring metrics associated with your cache nodes, enabling you to diagnose and react to issues very quickly. For example, you can set up thresholds and receive alarms if one of your cache nodes is overloaded with requests.

ALTERNATIVES

- Use a CDN, such as Amazon CloudFront
  - Generally caches whole objects, but not individual components that are only useful to the application
- Read-only replicas, such as of databases
  - Still not in-memory
- On host caching
  - Difficult to coordinate among many hosts serving application

KEY-VALUE ENGINES

- These should look familiar:
- Memcached
  - Because Memcached is designed as a pure caching solution with no persistence, ElastiCache manages Memcached nodes as a pool that can grow and shrink, similar to an Amazon EC2 Auto Scaling group. Individual nodes are expendable, and ElastiCache provides additional capabilities here such as automatic node replacement and Auto Discovery.
- Redis
  - Because of the replication and persistence features of Redis, ElastiCache manages Redis more as a relational database. Redis ElastiCache clusters are managed as stateful entities that include failover, similar to how Amazon RDS manages database failover.

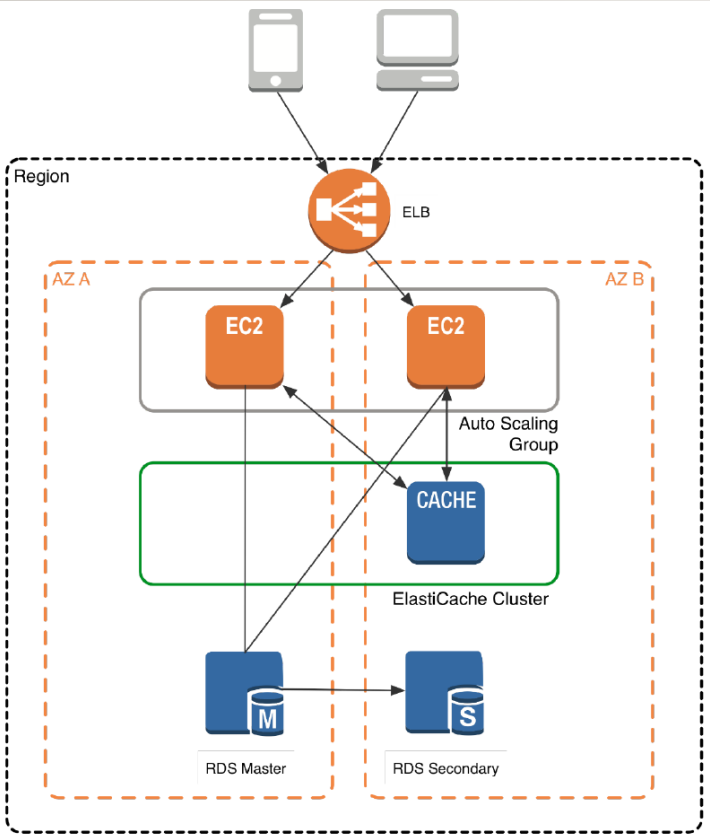
REDIS VS MEMCACHED

- Is object caching your primary goal, for example to offload your database? If so, use Memcached.
- Are you interested in as simple a caching model as possible? If so, use Memcached.
- Are you planning on running large cache nodes, and require multithreaded performance with utilization of multiple cores? If so, use Memcached.
- Do you want the ability to scale your cache horizontally as you grow? If so, use Memcached.
- Does your app need to atomically increment or decrement counters? If so, use either Redis orMemcached.
- Are you looking for more advanced data types, such as lists, hashes, and sets? Ifso, use Redis.
- Does sorting and ranking datasets in memory help you, such as with leaderboards? If so, use Redis.
- Are publish and subscribe (pub/sub) capabilities of use to your application? If so, use Redis. • Is persistence of your key store important? If so, useRedis.
- Do you want to run in multiple AWS Availability Zones (Multi-AZ) with failover? If so, use Redis.

REDIS VS MEMCACHED COMMON USE CASES

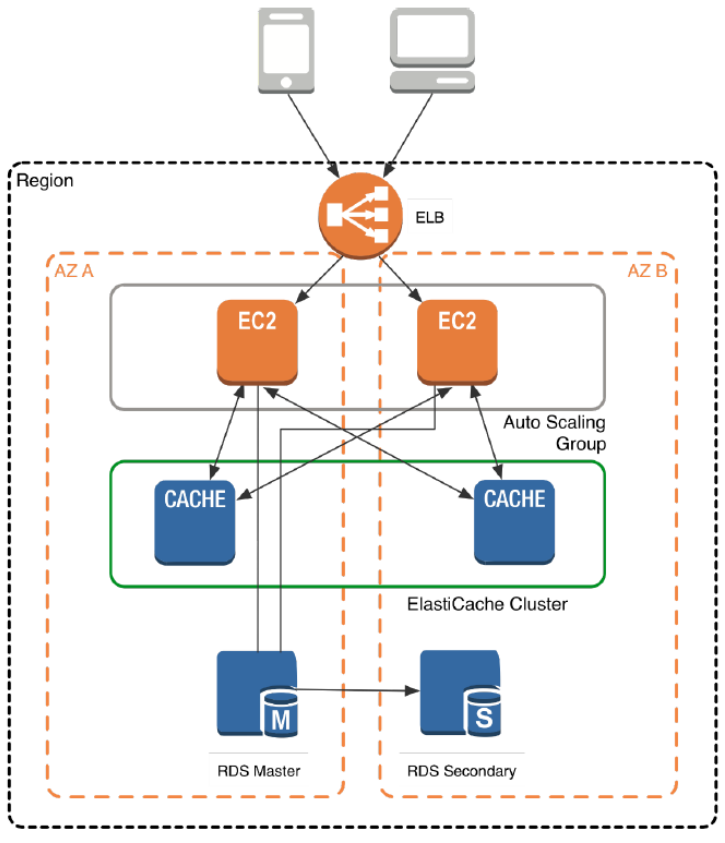
- Memcached as an in-memory cache pool
- Redis for advanced datasets such as game leaderboards and activity streams.

ARCHITECTURE (SIMPLIFIED) ELASTICACHE FOR MEMCACHED



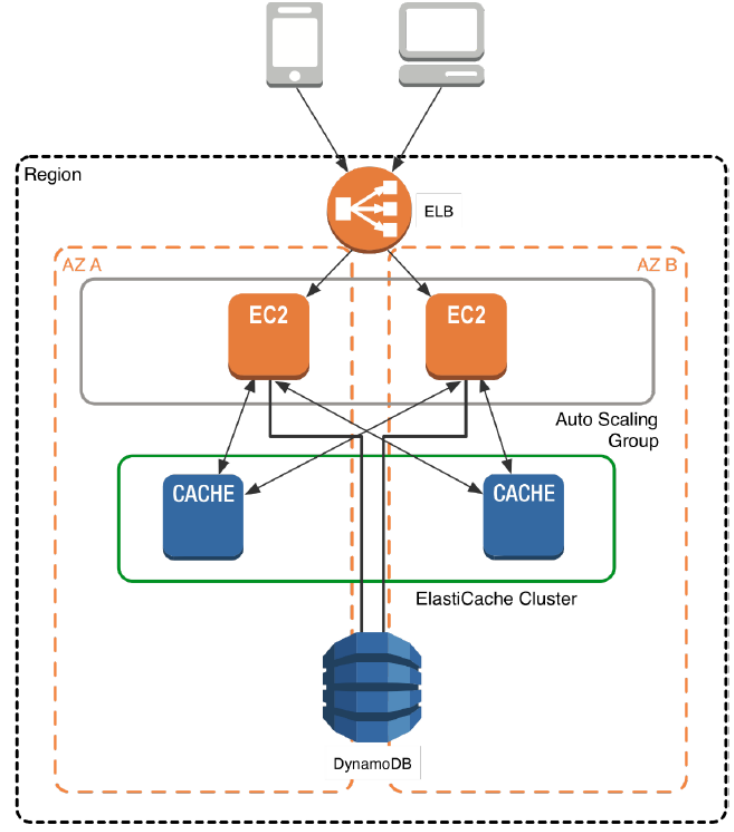
- ELB load balances among ECS application instances
- Application in EC2
- RDS Database
- ElastiCache between application and database
  - Coordinated by application in EC2
- Two availability zones (locations)

ARCHITECTURE (2 CACHE CLUSTERS) ELASTICACHE FOR MEMCACHED



- Notice this diagram has two cache clusters, one per AZ
- Cache clusters are coordinated by application in EC2
  - This means application is handing sharding

ARCHITECTURE (2 CACHE CLUSTERS + DYNAOMDB) ELASTICACHE FOR MEMCACHED



- Notice this solution uses DynamoDB
  - DynamoDB is not the same as Dynamo
- DynamoDB is Amazon's NoSQL Database
  - Implementation is black-boxed
  - Key-Value store
  - JSON-style documents

ACCESS CONTROL

- Neither Memcached nor Redis has any serious authentication or encryption capabilities.
- Like other Amazon web services, ElastiCache supports security groups.
  - You can use security groups to define rules that limit access to your instances based on IP address and port.
- ElastiCache supports both subnet security groups in Amazon Virtual Private Cloud (Amazon VPC) and classic Amazon EC2 security groups.
  - We strongly recommend you deploy ElastiCache and your application in Amazon VPC, unless you have a specific need otherwise (such as for an existing application).
  - Amazon VPC offers several advantages, including fine-grained access rules and control over private IP addressing.
- When launching your ElastiCache cluster in VPC, launch it in a private subnet with no public connectivity for best security.

HOW TO APPLY CACHING

- Is it safe to use a cached value? The same piece of data can have different consistency requirements in different contexts. For example, during online checkout, you need the authoritative price of an item, so caching might not be appropriate. On other pages, however, the price might be a few minutes out of date without a negative impact on users.
- Is caching effective for that data? Some applications generate access patterns that are not suitable for caching—for example, sweeping through the key space of a large dataset that is changing frequently. In this case, keeping the cache up to date could offset any advantage caching could offer.
- Is the data structured well for caching? Simply caching a database record can often be enough to offer significant performance advantages. However, other times, data is best cached in a format that combines multiple records together. Because caches are simple key-value stores, you might also need to cache a data record in multiple different formats, so you can access it by different attributes in the record.

MEMCHACHED LIBRARIES FEW EXAMPLES

Language	Library
Ruby	Dalli, Dalli::ElastiCache
Python	Memcache Ring, django-elasticache
Node.js	node-memcached
PHP	ElastiCache AutoDiscover Client
Java	ElastiCache AutoDiscover Client, spymemcached
C#/.NET	ElastiCache AutoDiscover Client, Enyim Memcached

CACHING STRATEGIES

- Simple Hashing
  - Like sophomore year (Ouch)
- Consistent Hashing
  - Complex to implement – but many good libraries
- Lazy Caching, e.g. LRU
  - Data pulled in by reads and expires
- Write-Through
  - Store data when written
  - Good if likely to be used again soon, not so good if it pushes out what will be
  - If not also doing Lazy caching, hot items won't be pulled back into cache after failure

THUNDERING HERDS

- The situation
  - Item not in cache (maybe brand new cache node)
  - TTL expires and item pushed out of cache
  - Many clients race to query underlying data in parallel
- One solution
  - Pre-warm cache

ELASTICACHE WITH REDIS

- Redis data structures cannot be horizontally sharded.
  - As a result, Redis ElastiCache clusters are always a single node, rather than the multiple nodes we saw with Memcached.
- Redis supports replication, both for high availability and to separate read workloads from write workloads.
  - A given ElastiCache for Redis primary node can have one or more replica nodes.
  - A Redis primary node can handle both reads and writes from the app.
  - Redis replica nodes can only handle reads, similar to Amazon RDS Read Replicas.
- Because Redis supports replication, you can also fail over from the primary node to a replica in the event of failure.
  - You can configure ElastiCache for Redis to automatically fail over by using the Multi-AZ feature.
- Redis supports persistence, including backup and recovery.
  - However, because Redis replication is asynchronous, you cannot completely guard against data loss in the event of a failure.

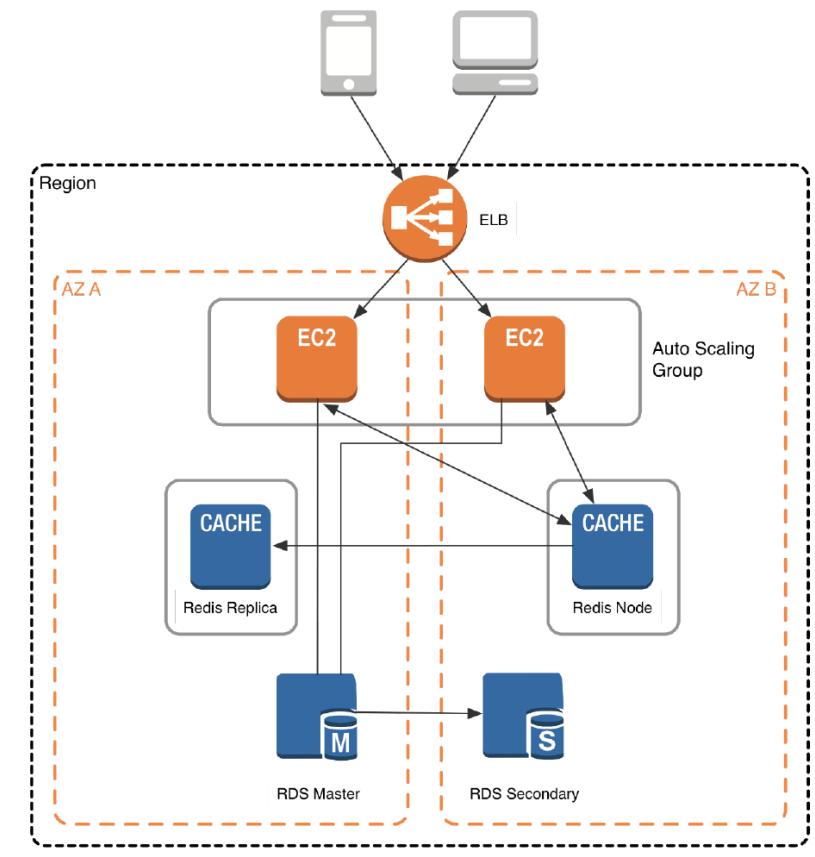
ELASTICACHE WITH REDIS: ARCHITECTURE

- As with Memcached, when you deploy an ElastiCache for Redis cluster, it is an additional tier in your app.
- Unlike Memcached, ElastiCache clusters for Redis only contain a single primary node.
- After you create the primary node, you can configure one or more replica nodes and attach them to the primary Redis node.
- An ElastiCache for Redis replication group consists of a primary and up to five read replicas.
  - Redis asynchronously replicates the data from the primary to the read replicas.

ELASTICACHE WITH REDIS: ARCHITECTURE NOTE

- Because Redis supports persistence, it is technically possible to use Redis as your only data store.
- In practice, customers find that a managed database such as Amazon DynamoDB or Amazon RDS is a better fit for most use cases of long-term data storage.
  - Redis provides rich types and simple data structures
  - Databases provide much stronger ability to structure data and organize and validate constraints

ELASTICACHE WITH REDIS: ARCHITECTURE (SIMPLE)

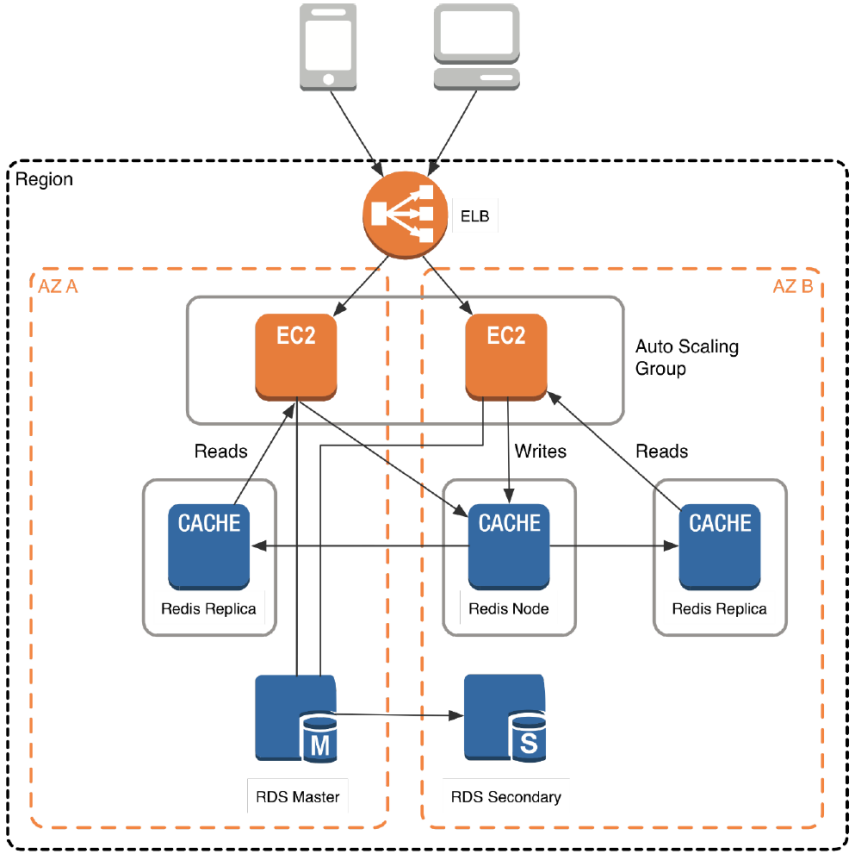


- Notice Simple replication across Azs
- ElastiCache for Redis has the concept of a primary endpoint, which is a DNS name that always points to the current Redis primary node.
- If a failover event occurs, the DNS entry will be updated to point to the new Redis primary node.

ELASTICACHE WITH REDIS: ARCHITECTURE (DISTRIBUTING READS AND WRITES)

- Using read replicas with Redis, you can separate your read and write workloads.
- This separation lets you scale reads by adding additional replicas as your application grows.
- In this pattern, you configure your application to send writes to the primary endpoint.
- Then you read from one of the replicas.
- With this approach, you can scale your read and write loads independently, so your primary node only has to deal with writes.

ELASTICACHE WITH REDIS: ARCHITECTURE (DISTRIBUTING READS AND WRITES)

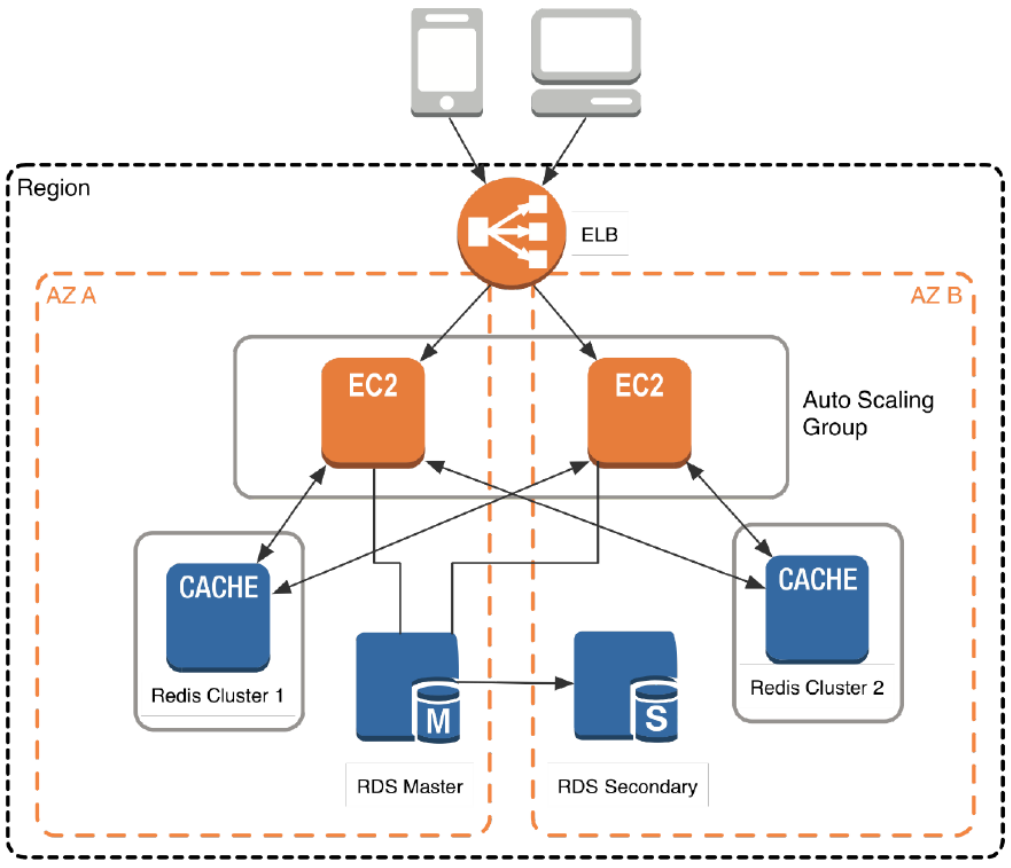


- Notice that reads are only from the replicas
- And the primary node absorbs the initial write and then replicates it.

ELASTICACHE WITH REDIS: ARCHITECTURE (DISTRIBUTING READS AND WRITES)

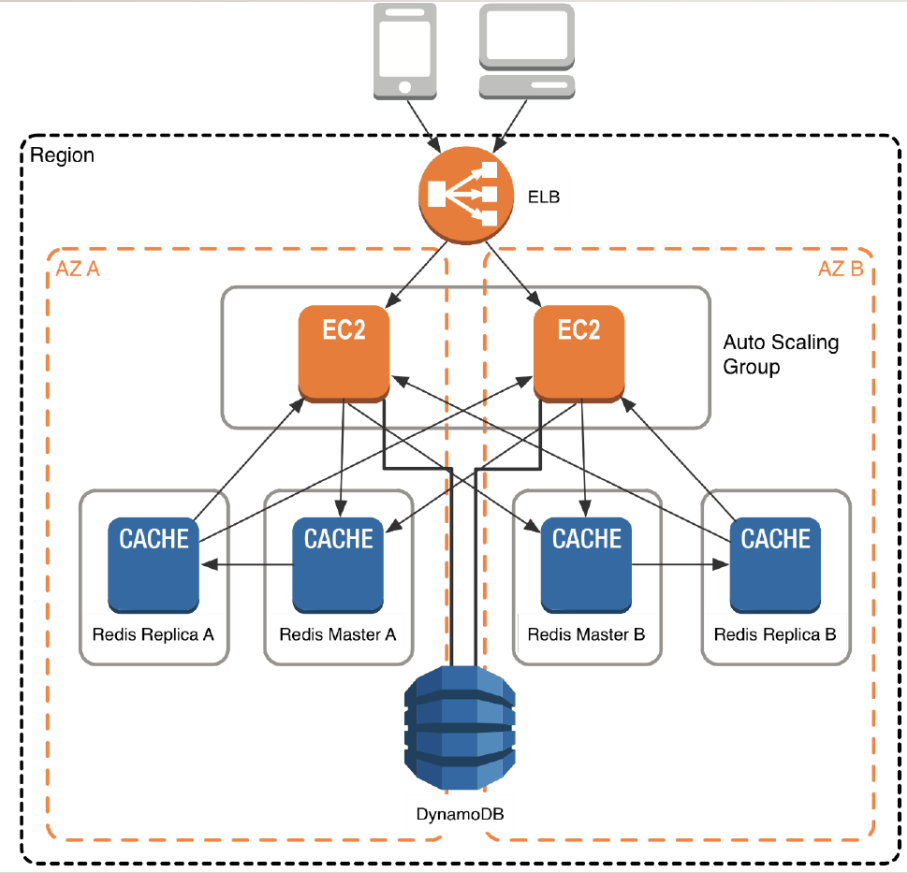
- **The main caveat to this approach is that reads can return data that is slightly out of date compared to the primary node, because Redis replication is asynchronous.**
- Is the value being used only for display purposes? If so, being slightly out of date is probably okay.
- • Is the value a cached value, for example a page fragment? If so, again beingslightly out of date is likely fine.
- • Is the value being used on a screen where the user might have just edited it? In this case, showing an old value might look like an application bug.
- • Is the value being used for application logic? If so, using an old value can be risky.
- • Are multiple processes using the value simultaneously, such as a lock or queue?If so, the value needs to be up-to-date and needs to be read from the primarynode.

ELASTICACHE WITH REDIS: ARCHITECTURE (SHARDING)



- Redis has two categories of data structures: simple keys and counters, and multidimensional sets, lists, and hashes.
  - The bad news is the second category cannot be sharded horizontally.
  - But the good news is that simple keys and counters can.
    - Range partitioning, Hash partitioning
- In the simplest case, you can treat a single Redis node just like a single Memcached node.
  - Just like you might spin up multiple Memcached nodes, you can spin up multiple Redis clusters, and each Redis cluster is responsible for part of the sharded dataset.

ELASTICACHE WITH REDIS: ARCHITECTURE (SHARDING + SPLIT READS AND WRITES)



- Notice two masters
  - Each with a replica