aws academy

CSCI 5902 Adv. Cloud Architecting
Fall 2023
Instructor: Lu Yang

Module 13 Caching Content (Sections 4-5)
Dec 4, 2023

Housekeeping items and feedback

1. Start recording

Recap of our last lecture

Module overview

Sections

1. Architectural need
2. Overview of caching
3. Edge caching
4. Caching web sessions
5. Caching databases

We stopped here in the last lecture

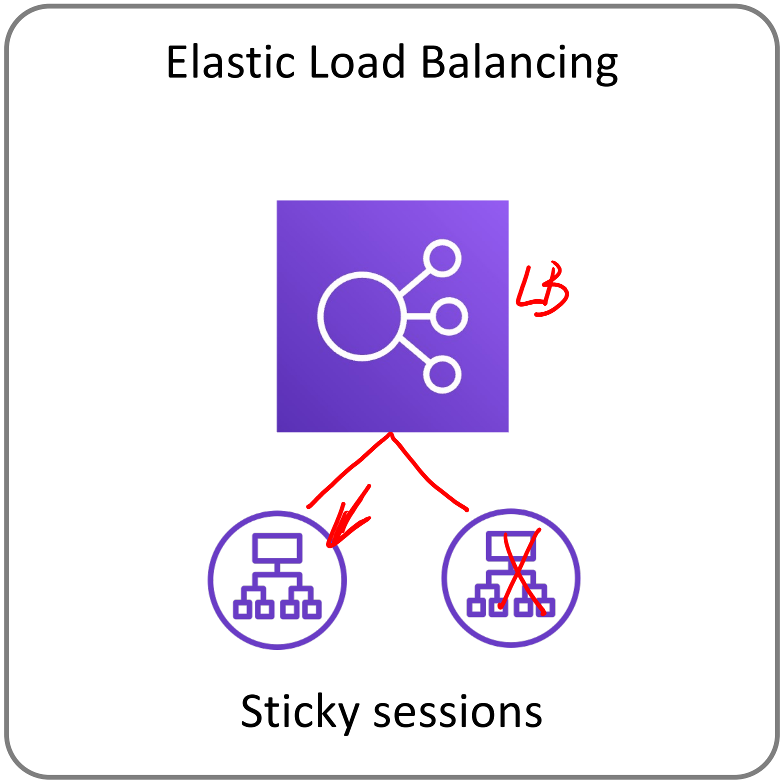
Lab

- Guided Lab: Streaming Dynamic Content Using Amazon CloudFront

Module 11: Caching Content

Section 4: Caching web sessions

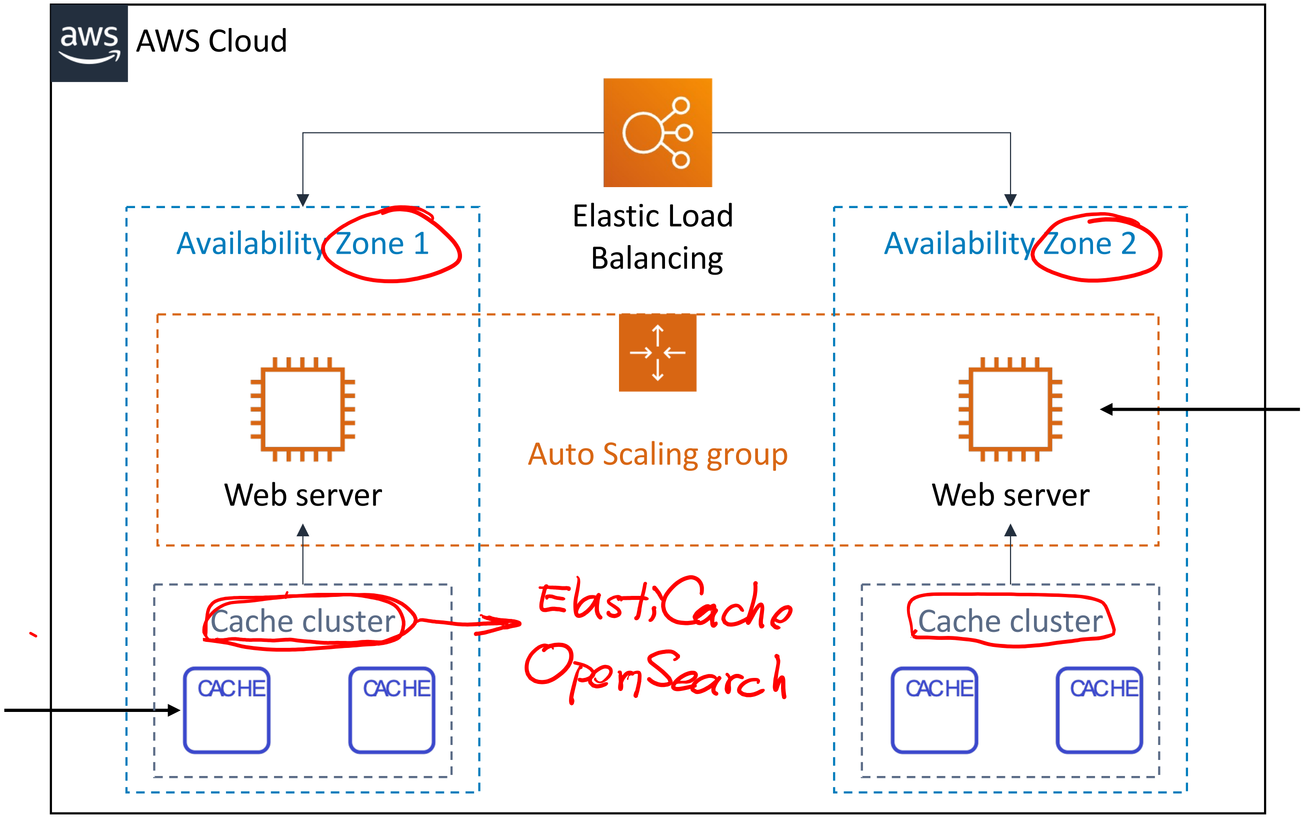
Session management: Sticky sessions



Feature that enables a load balancer to route a request to the specific server that manages the user's session.

- Use client-side cookies
- Are cost-effective
- Speed up retrieval of sessions
- Have disadvantages –
  - Loss of sessions when you have an instance failure
  - Limit scalability: Uneven load distribution and increased latency

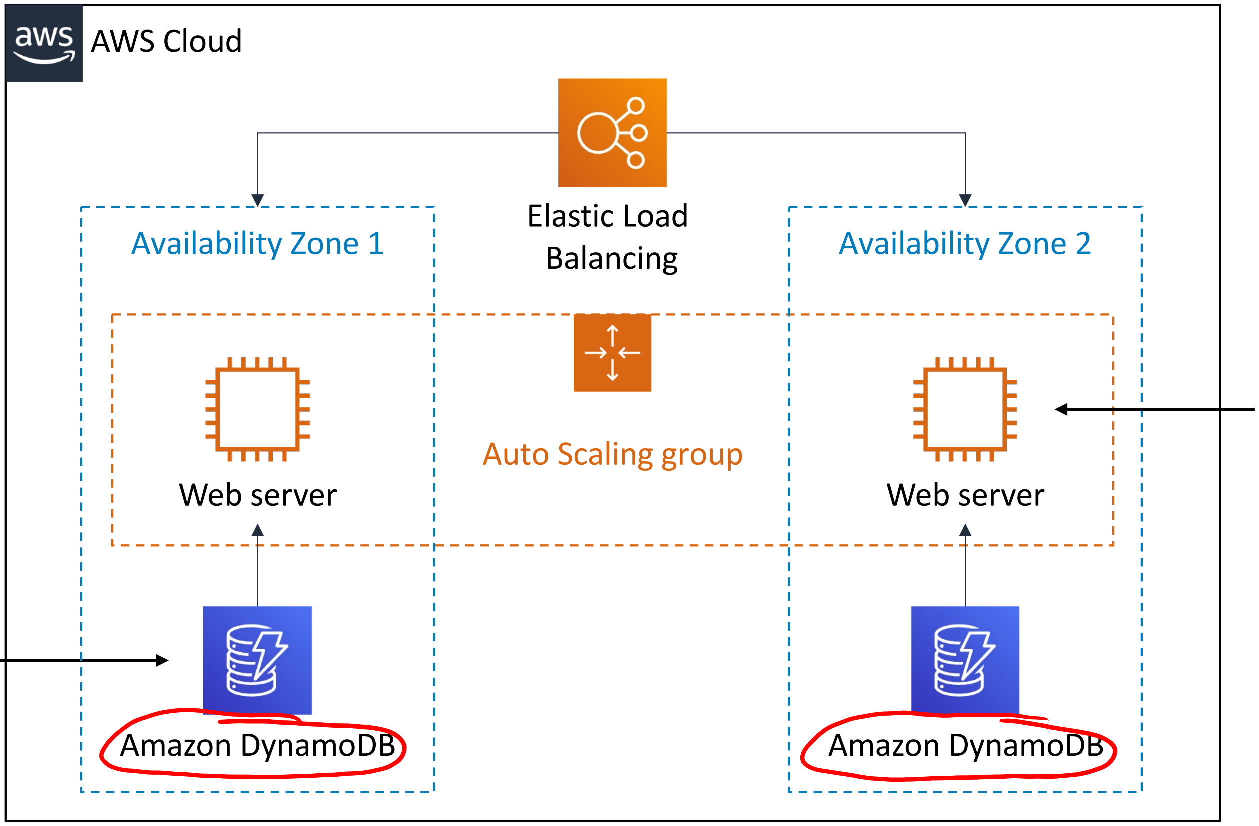
Instead of sticky sessions: Persist sessions inside a distributed cache



Loss of sessions is a risk if sessions are stored locally on nodes

Persist sessions in a distributed cache

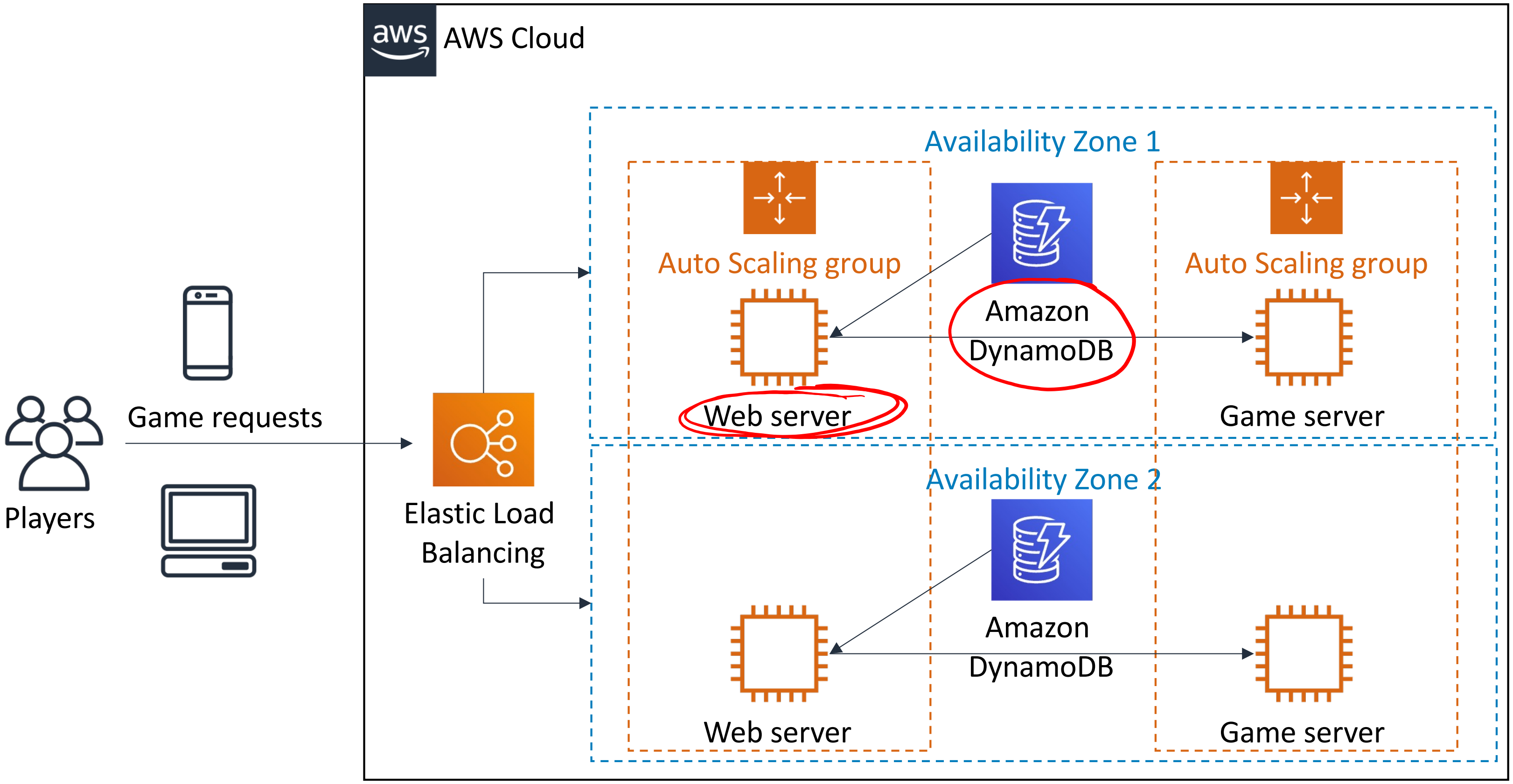
Instead of sticky sessions:
Persist sessions inside a DynamoDB table



Persist sessions in an Amazon DynamoDB database

Loss of sessions is a risk if sessions are stored locally on nodes

Example: Storing session states for an online gaming application



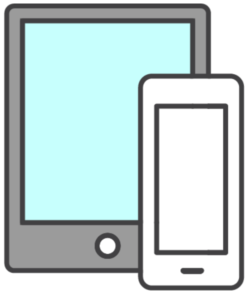
Section 4 key takeaways

- Sessions are used to manage user authentication and store user data while the user interacts with the application.
- You can manage sessions with sticky sessions, which is a feature of Elastic Load Balancing load balancers. Sticky sessions route requests to the specific server that's managing the user's session.
- You can also manage sessions by persisting session data outside the web server instance—for example, in a distributed cache or DynamoDB table.

Module 11: Caching Content

Section 5: Caching databases

When should you cache your database?



You are concerned about response times for your customer.

latency

You have a high volume of requests that are inundating your database.

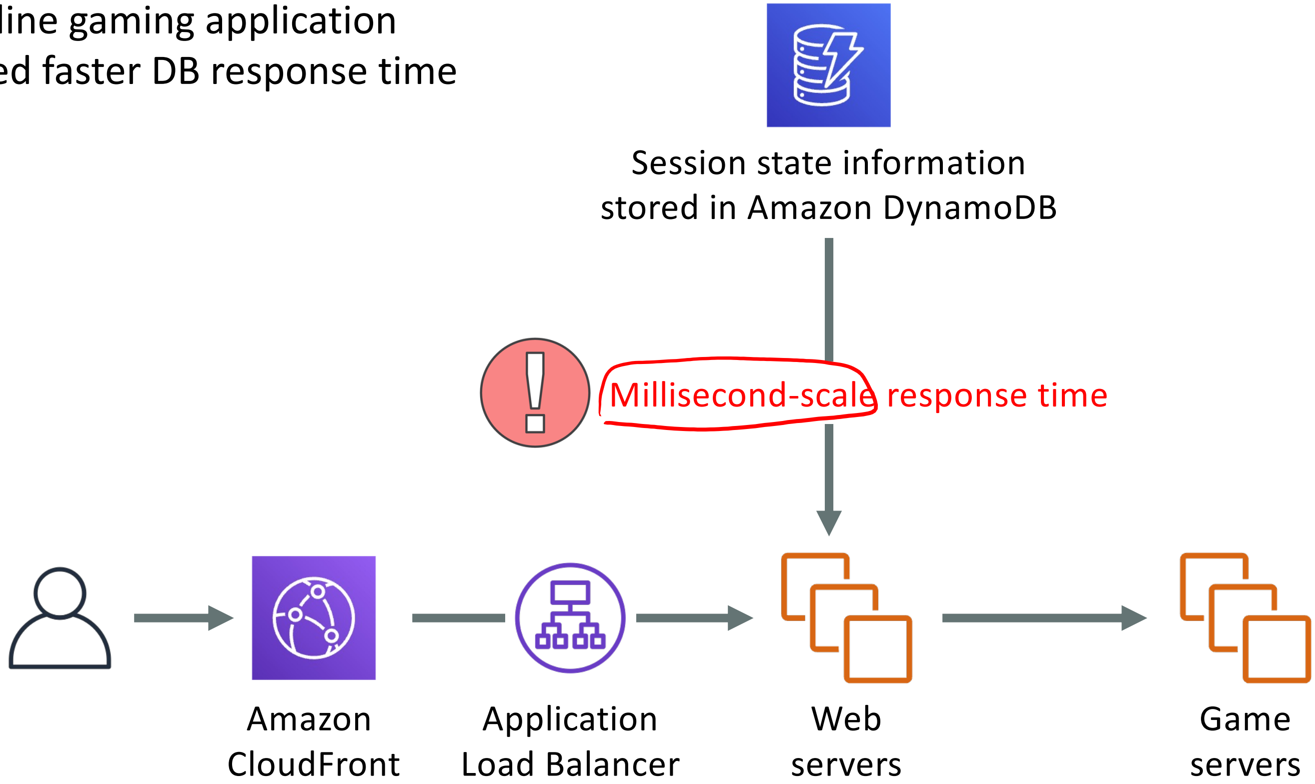
You would like to reduce your database costs.

read replicas

Cache

Using DynamoDB for state information

Use case: Online gaming application
Problem: Need faster DB response time


Session state information stored in Amazon DynamoDB
Millisecond-scale response time
Amazon CloudFront
Application Load Balancer
Web servers
Game servers

Amazon DynamoDB Accelerator

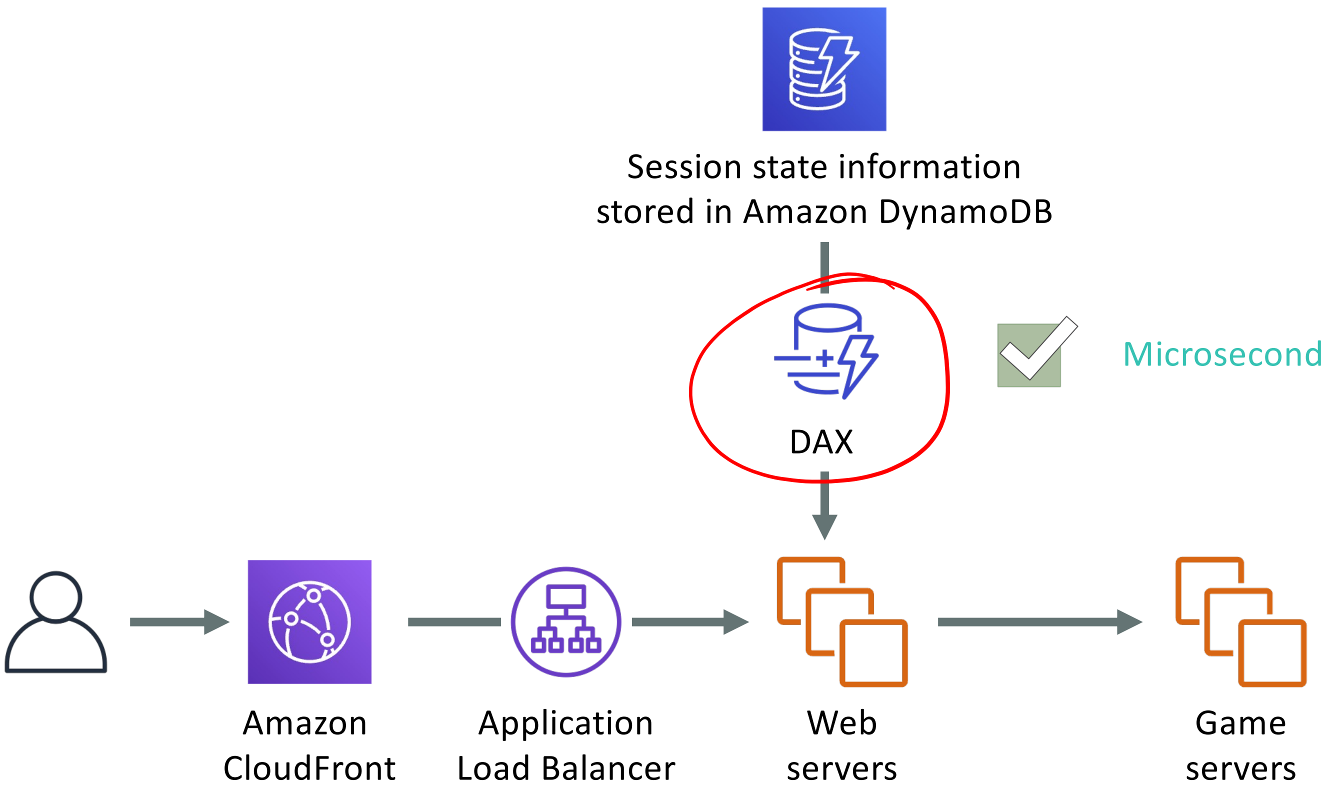
DAX

Amazon DynamoDB Accelerator

Fully managed, highly available, in-memory cache for DynamoDB

- Extreme performance (microsecond-scale response time)
- Highly scalable
- Fully managed
- Integrated with DynamoDB
- Flexible
- Secure

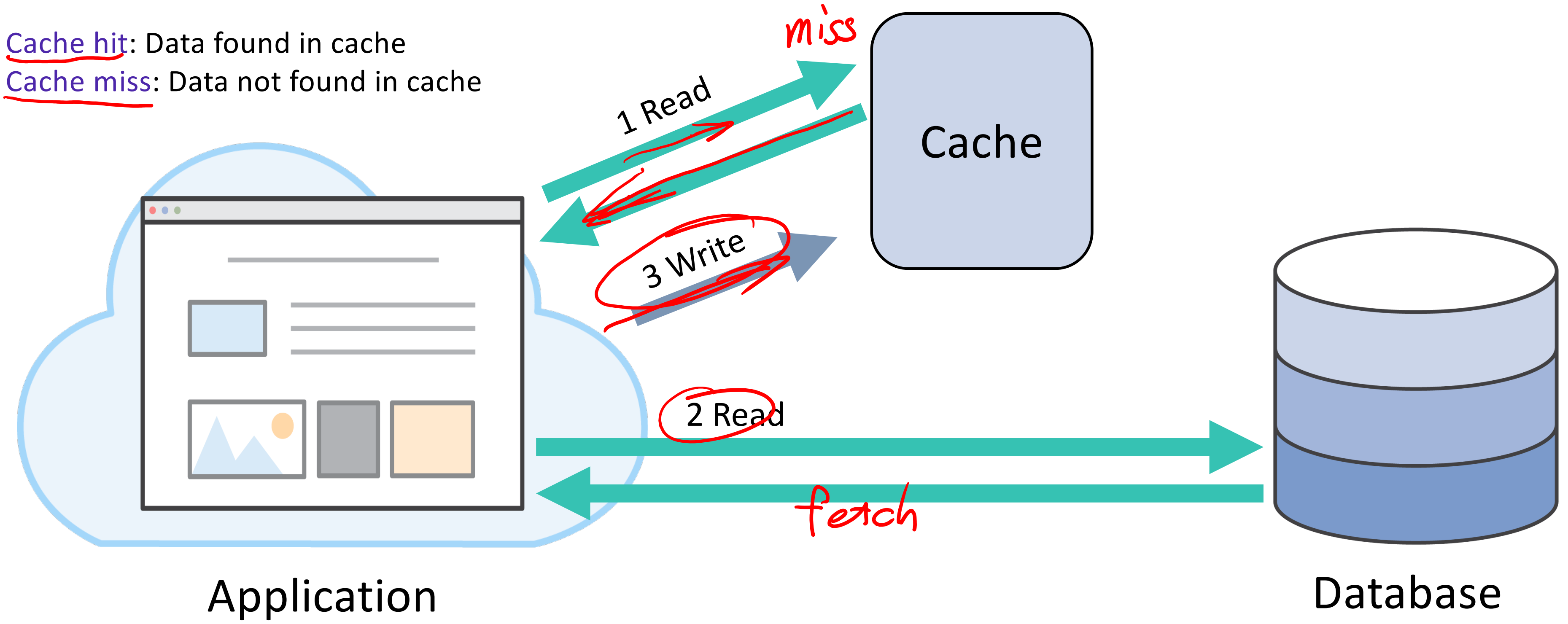
Using DynamoDB with DAX to accelerate response time


Session state information stored in Amazon DynamoDB
DAX
Amazon CloudFront
Application Load Balancer
Web servers
Game servers

✓ Microsecond-scale response time

Remote or side caches

Cache hit: Data found in cache
Cache miss: Data not found in cache



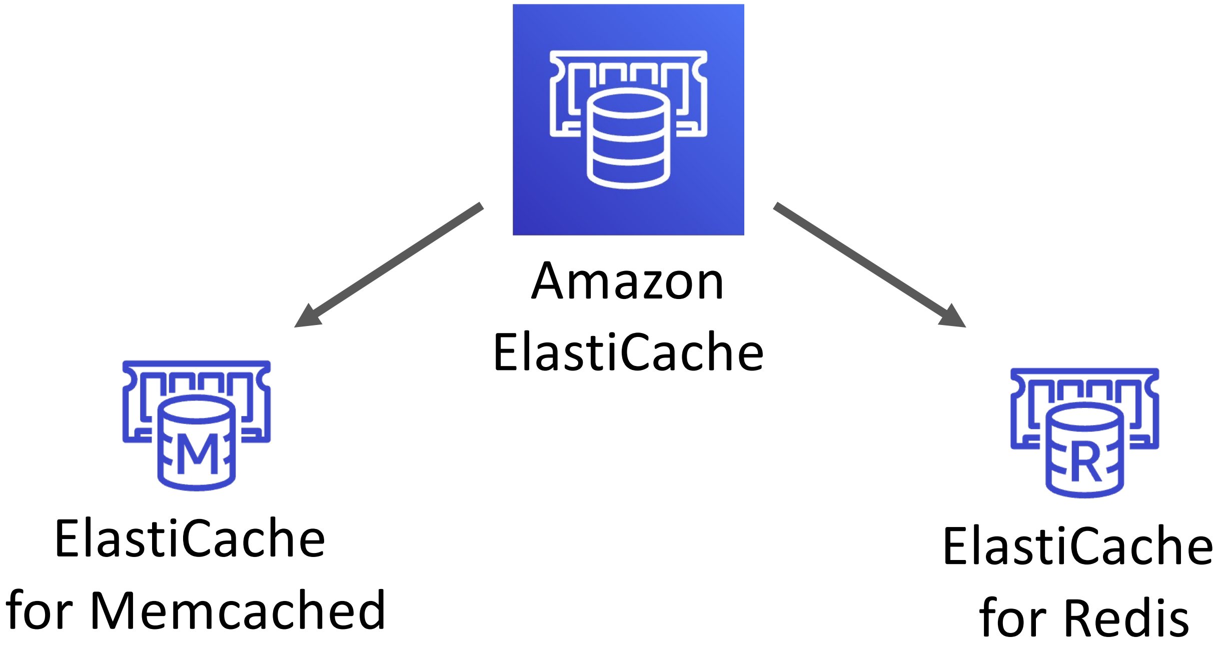
Amazon ElastiCache

Amazon ElastiCache

ElastiCache provides web applications with an in-memory data store in the cloud.

- Works an in-memory data store and cache
- Offers high performance microsecond
- Is fully managed
- Is scalable
- Supports Redis and Memcached

Redis and Memcached



Amazon ElastiCache

ElastiCache for Memcached

ElastiCache for Redis

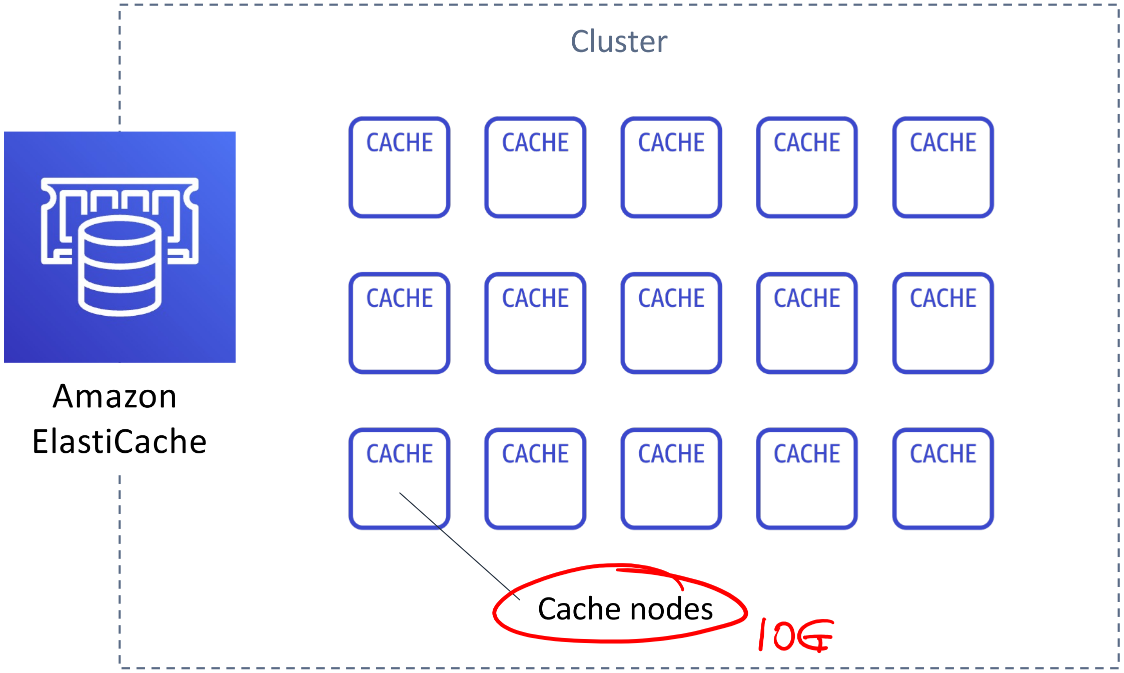
Scales up to 20 nodes per cluster

Scales up to 250 nodes per cluster

Memcached versus Redis comparison

Feature	Memcached	Redis
Sub-millisecond latency	Yes	Yes
Ability to scale horizontally for writes and storage	Yes	No
Multi-threaded performance	Yes	No
Advanced data structures	No	Yes
Sorting and ranking datasets	No	Yes
Publish/subscribe messaging	No	Yes
Multi-AZ deployments with automatic failover	No	Yes
Persistence	No	Yes

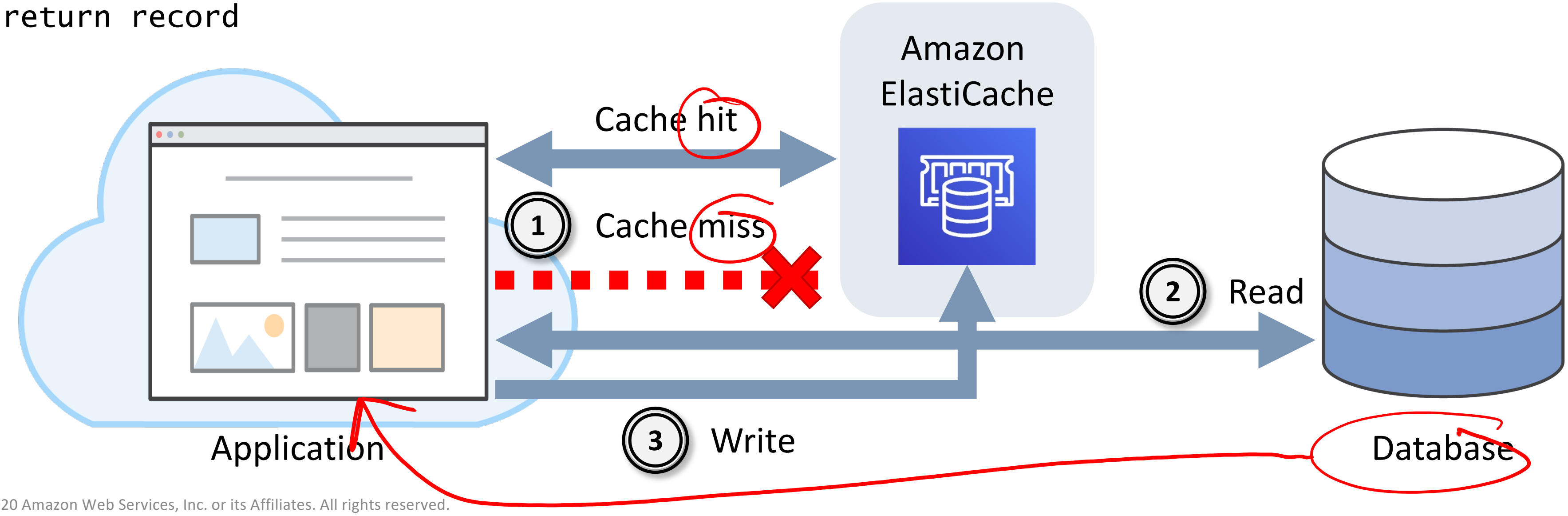
ElastiCache components



- A node is the smallest block of an ElastiCache deployment
- Each node has its own DNS name and port
- A cluster is a logical grouping of one or more nodes

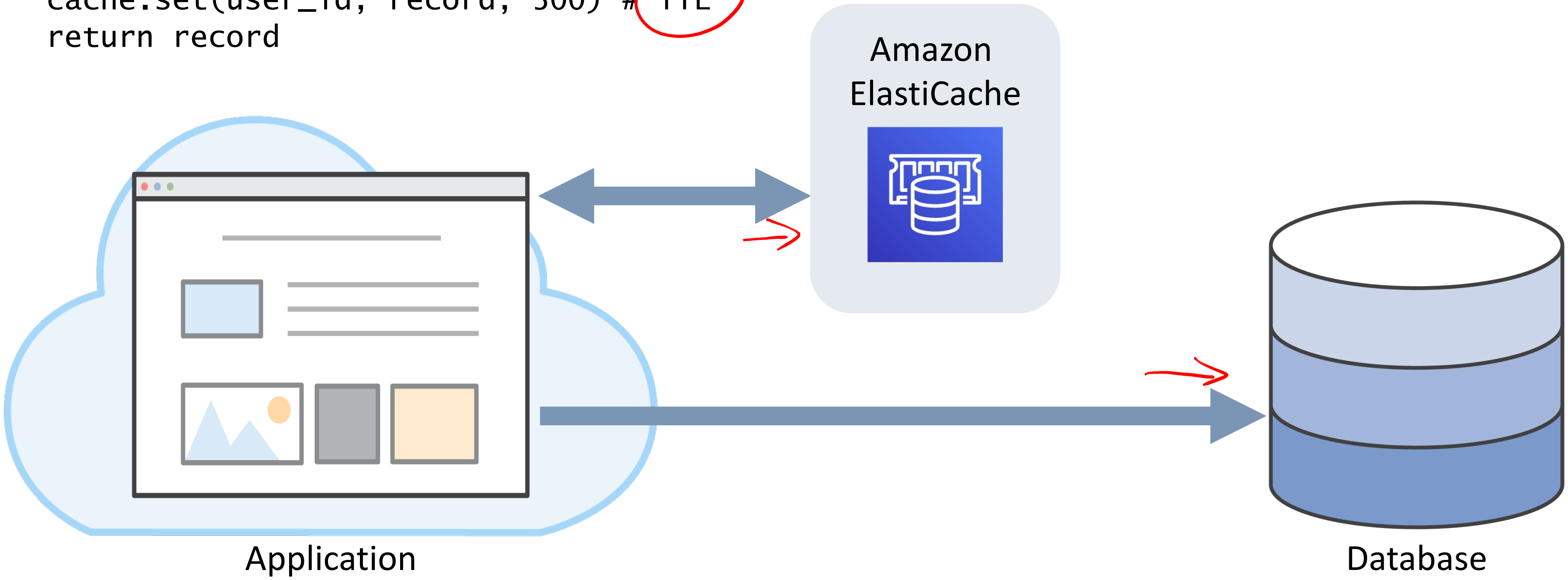
Caching strategies: Lazy loading

```
def get_user(user_id):
    # Check the cache
    record = cache.get(user_id)
    if record is None:
        # Run a DB query
        record = db.query("select * from users where id = ?", user_id)
        # Populate the cache
        cache.set(user_id, record)
    return record
```

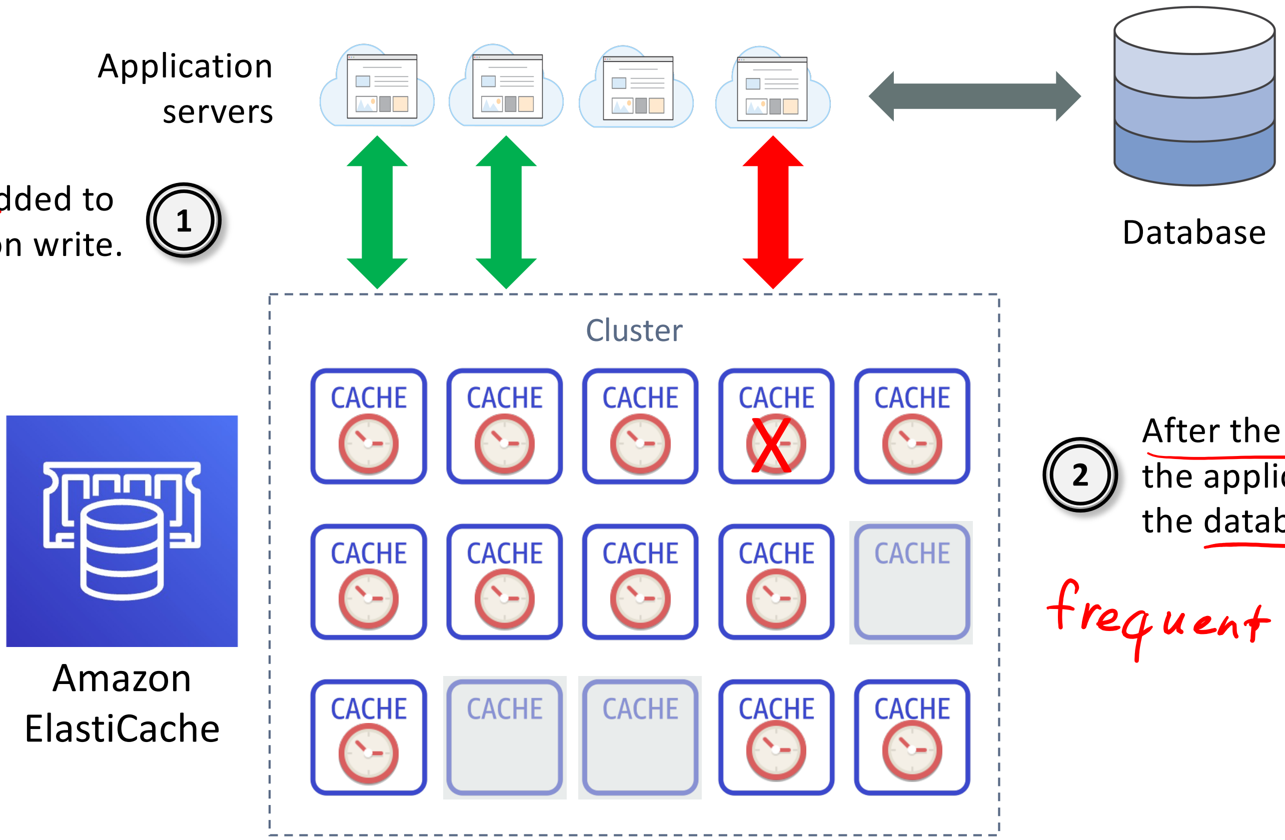


Caching strategies: Write-through

```
def save_user(user_id, values):
    # Save to DB
    record = db.query(“update users…where id = ?”, user_id, values)
    # Push into cache
    cache.set(user_id, record, 300) # TTL
    return record
```



Adding TTL

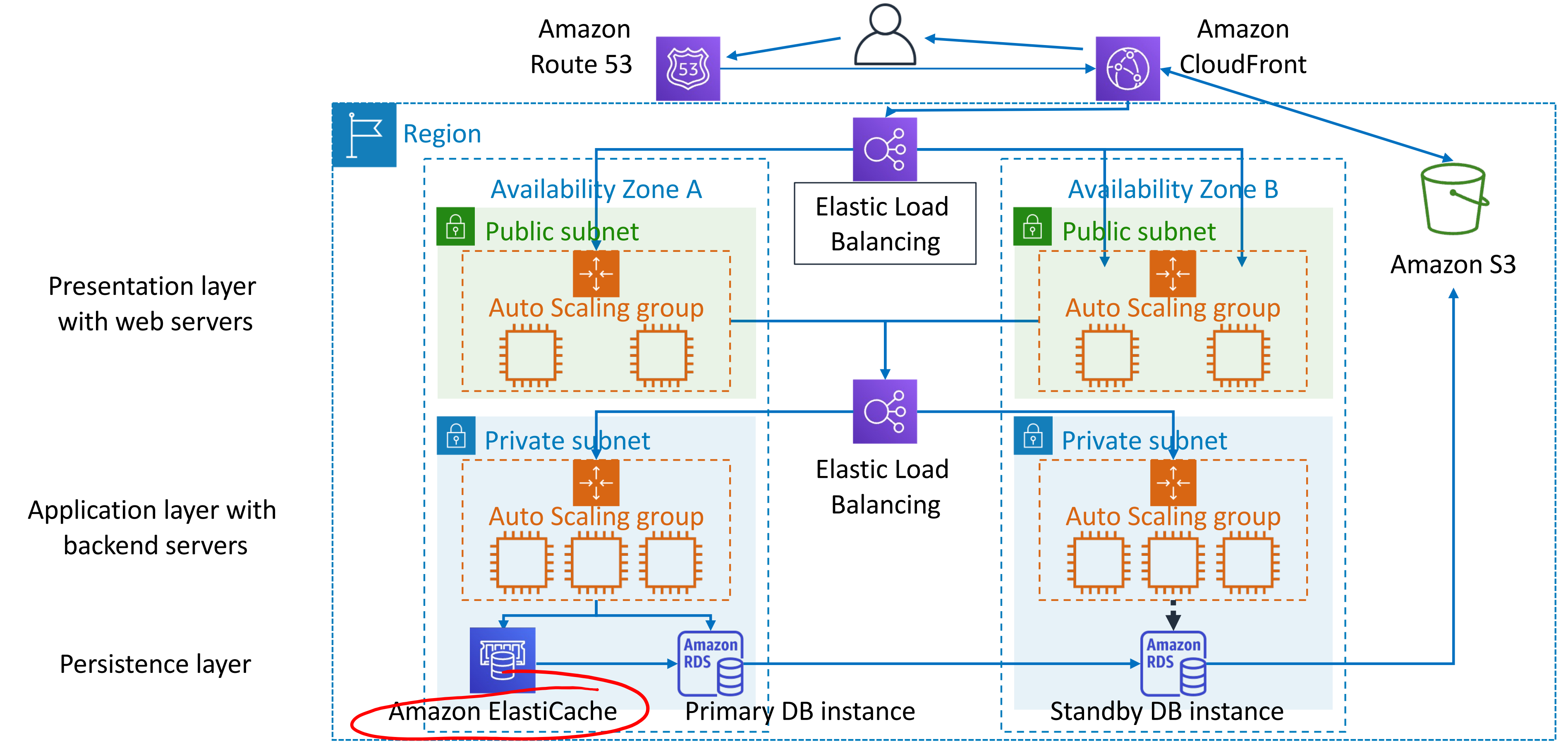


1. A TTL value is added to each application write.

2. After the TTL expires, the application queries the database for data.

frequent

Three-tier web hosting architecture



Section 5 key takeaways

- A database cache supplements your primary database by removing unnecessary pressure on it, typically in the form of frequently accessed read data
- DAX is a fully managed, highly available, in-memory cache for DynamoDB that delivers a performance improvement of up to 10 times—from milliseconds to microseconds
- Amazon ElastiCache is a side cache that works as an in-memory data store to support the most demanding applications that require microsecond response times

Module 13: Caching Content

Module wrap-up

Module summary

In summary, in this module, you learned how to:

- Identify how caching content can improve application performance and reduce latency
- Create architectures that use Amazon CloudFront to cache content
- Identify how to design architectures that use edge locations for distribution and distributed denial of service (DDoS) protection
- Recognize how session management relates to caching
- Describe how to design architectures that use Amazon ElastiCache

Thank you

aws academy