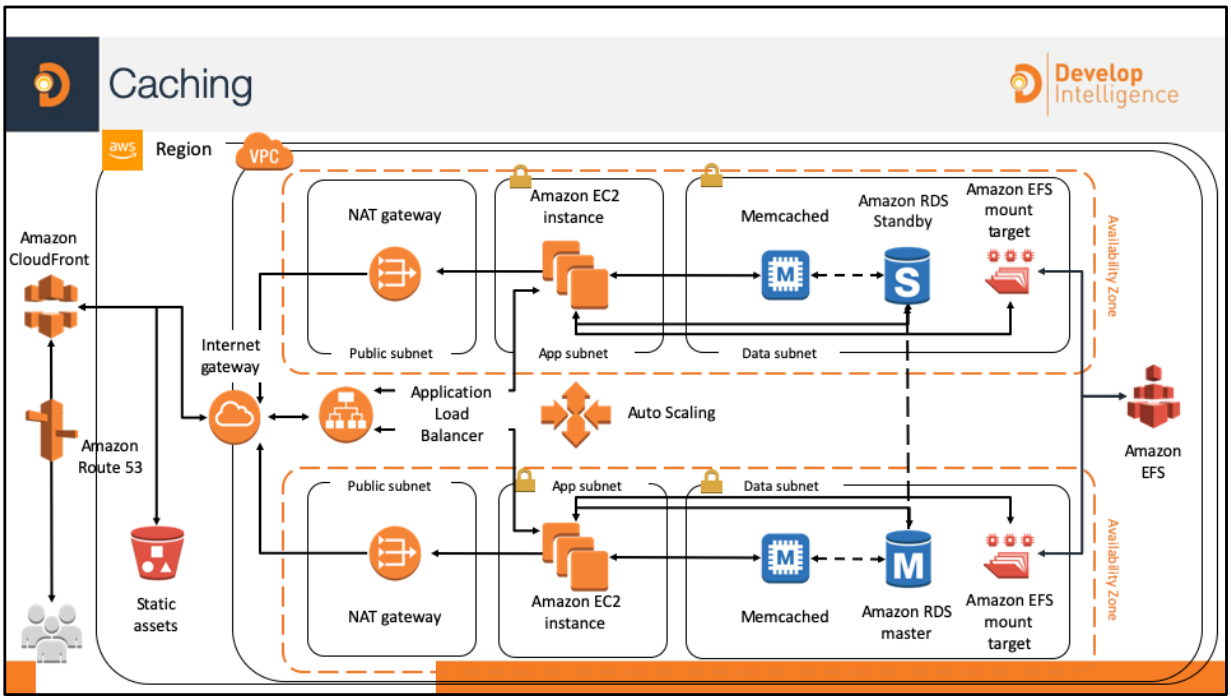
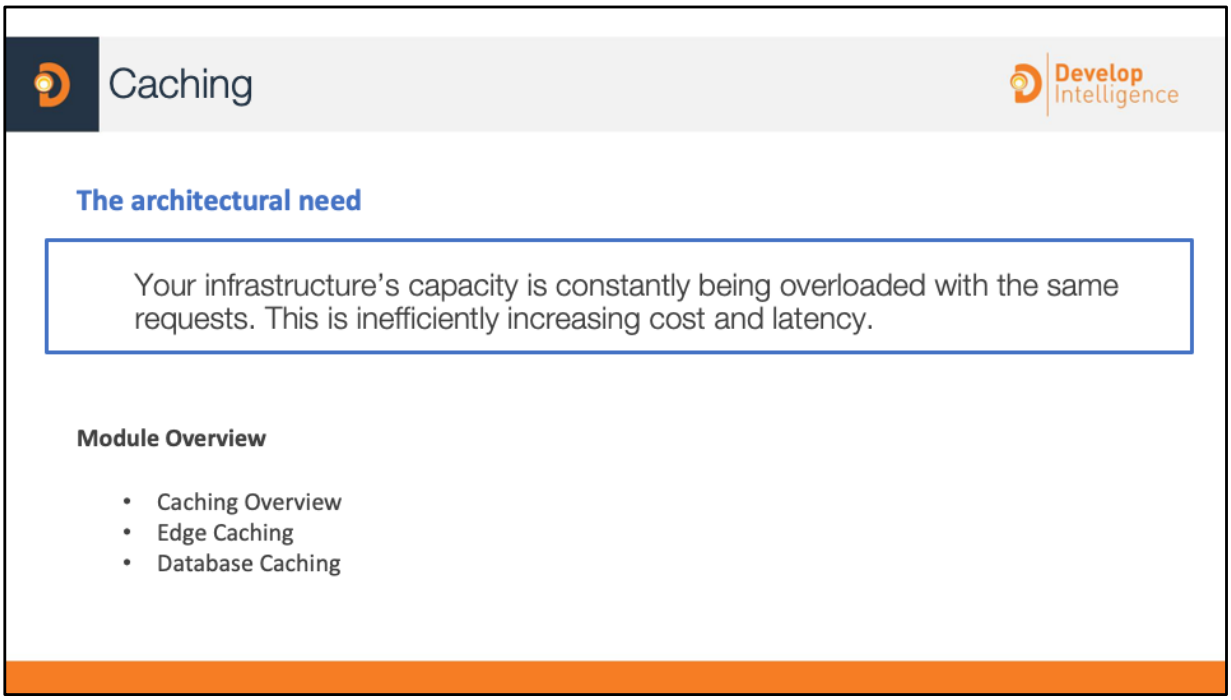
Caching

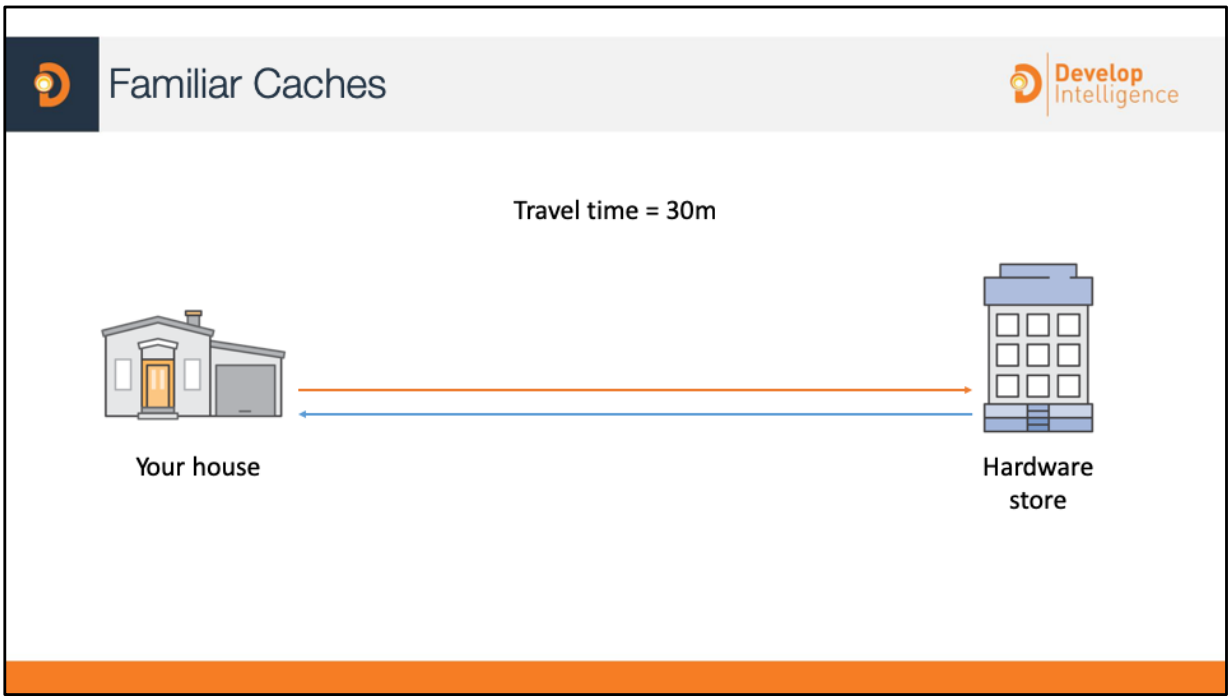
Develop Intelligence

- Module 10



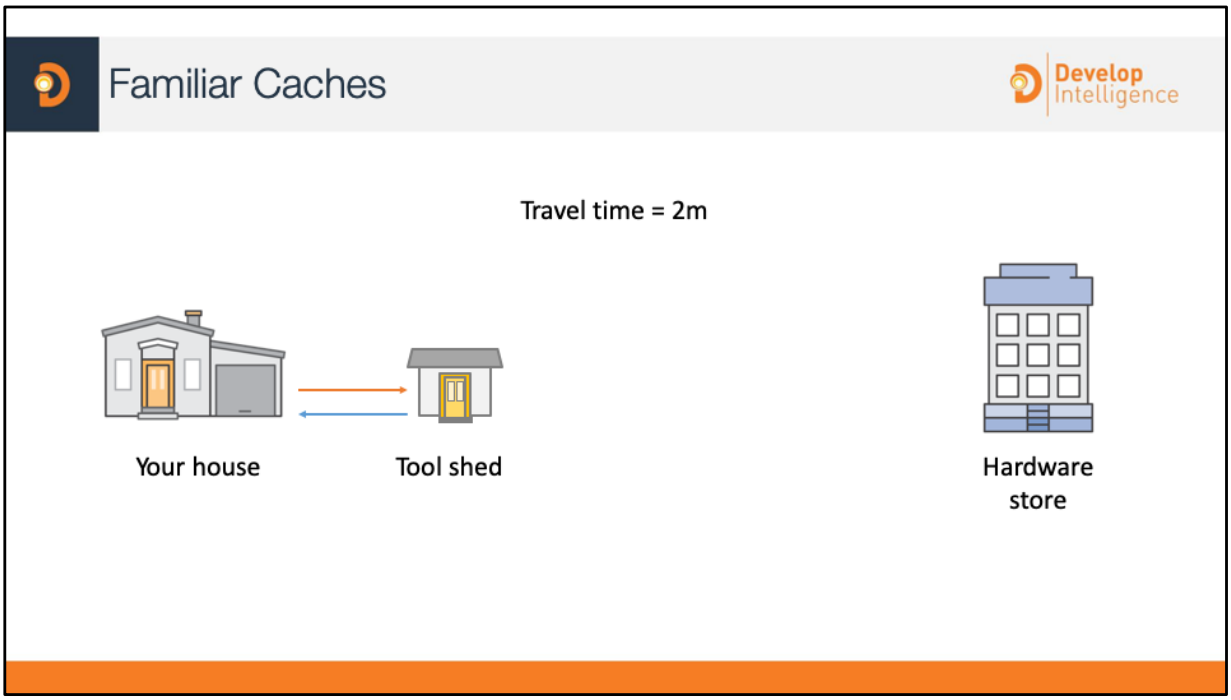
By the end of class, you will be able to understand all of the components of this architectural diagram. You will also be able to construct your own architectural solutions that are just as large and robust.


Caching
Develop Intelligence
The architectural need
Your infrastructure's capacity is constantly being overloaded with the same requests. This is inefficiently increasing cost and latency.
Module Overview
Caching Overview
Edge Caching
Database Caching



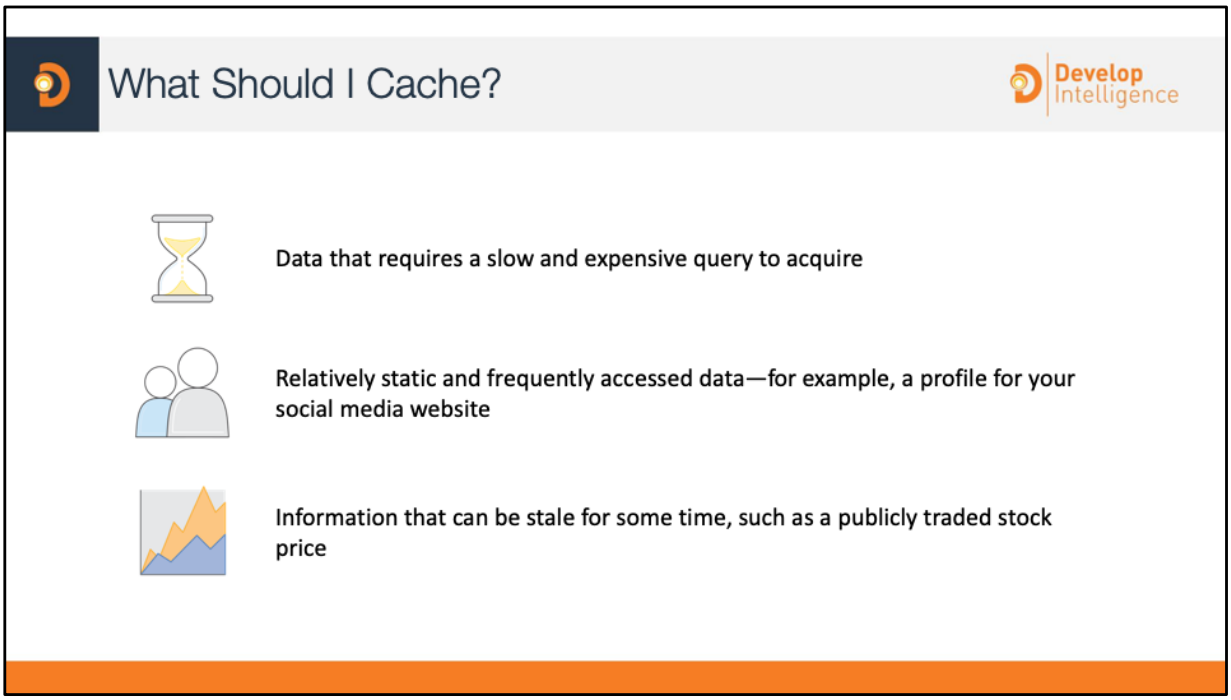
To illustrate how a cache improves performance, think about making a trip to a hardware store.

If the store is miles away, it's going to take considerable effort to go there every time you need something.



Instead, if you have a storage unit, a cache, close by, fill it with the supplies you need. Then, instead of going to the store when you need something, you can just go to your storage.

However, the store is always an option if you need to refresh your stockpile.

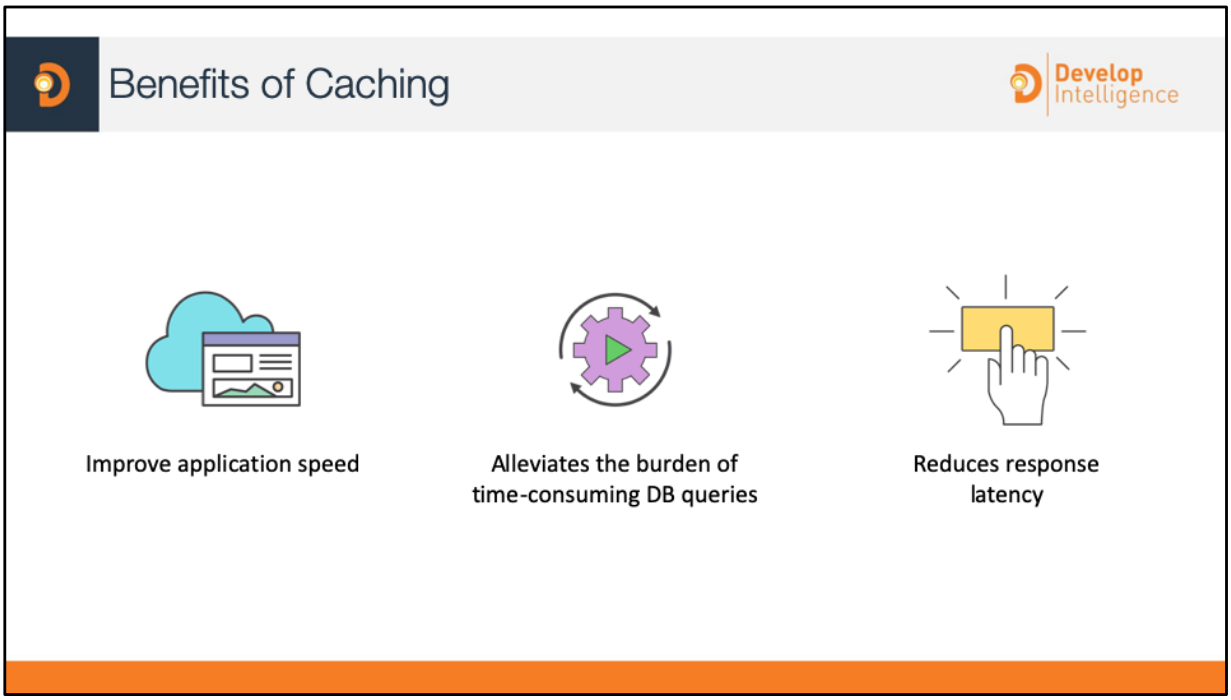


Speed and Expense – It's always slower and more expensive to acquire data from a database than from a cache. Some database queries are inherently slower and more expensive than others. For example, queries that perform joins on multiple tables are significantly slower and more expensive than simple, single-table queries. If acquiring the interesting data requires a slow and expensive query, it's a candidate for caching. If acquiring the data requires a relatively quick and simple query, it might still be a candidate for caching, depending on other factors.

Data and Access Pattern – Determining what to cache also involves understanding the data itself and its access patterns. For example, it doesn't make sense to cache data that is rapidly changing or is seldom accessed. For caching to provide a meaningful benefit, the data should be relatively static and frequently accessed, such as a personal profile on a social media site. Conversely, you don't want to cache data if caching it provides no speed or cost advantage. For example, it doesn't make sense to cache webpages that return the results of a search because such queries and results are almost always unique.

Staleness – By definition, cached data is stale data—even if in certain circumstances it isn't stale, it should always be considered and treated as stale. In determining whether your data is a candidate for caching, you need to determine your application's tolerance for stale data. Your application might be able to tolerate stale data in one context, but not another.

For example, when serving a publicly traded stock price on a website, staleness might be acceptable, with a disclaimer that prices might be up to *n* minutes delayed. But when serving up the price for the same stock to a broker making a sale or purchase, you want real-time data.

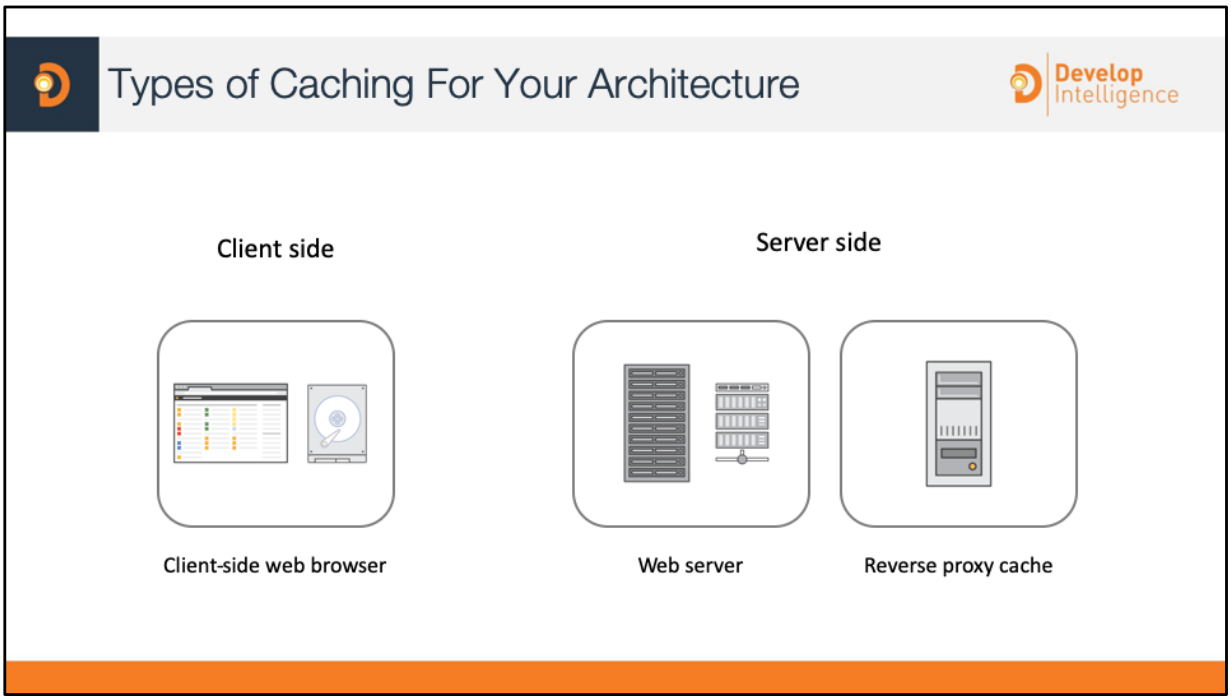


A cache provides high throughput, low-latency access to commonly accessed application data, by storing the data in memory. Caching can improve the speed of your application. Caching reduces the response latency experienced by the users of your application. Time-consuming database queries and complex queries often create bottlenecks in applications. In read-intensive applications, caching can provide significant performance gains by reducing application processing time and database access time.

Write-intensive applications typically do not see as great a benefit from caching. However, even write-intensive applications normally have a read/write ratio greater than 1, which implies that read caching will still be beneficial

You should consider caching your data if your data:

- Is slow or expensive to acquire when compared to cache retrieval
- Is accessed with sufficient frequency
- Is relatively static or if rapidly changing and staleness is not a significant issue



In computing, a *cache* is a high-speed data storage layer that stores a subset of data, typically transient in nature, so that future requests for that data are served up faster than is possible by accessing the data's primary storage location. Caching allows you to efficiently reuse previously retrieved or computed data.

Web caching is performed by retaining HTTP responses and web resources in the cache for the purpose of fulfilling future requests from cache rather than from the origin servers.

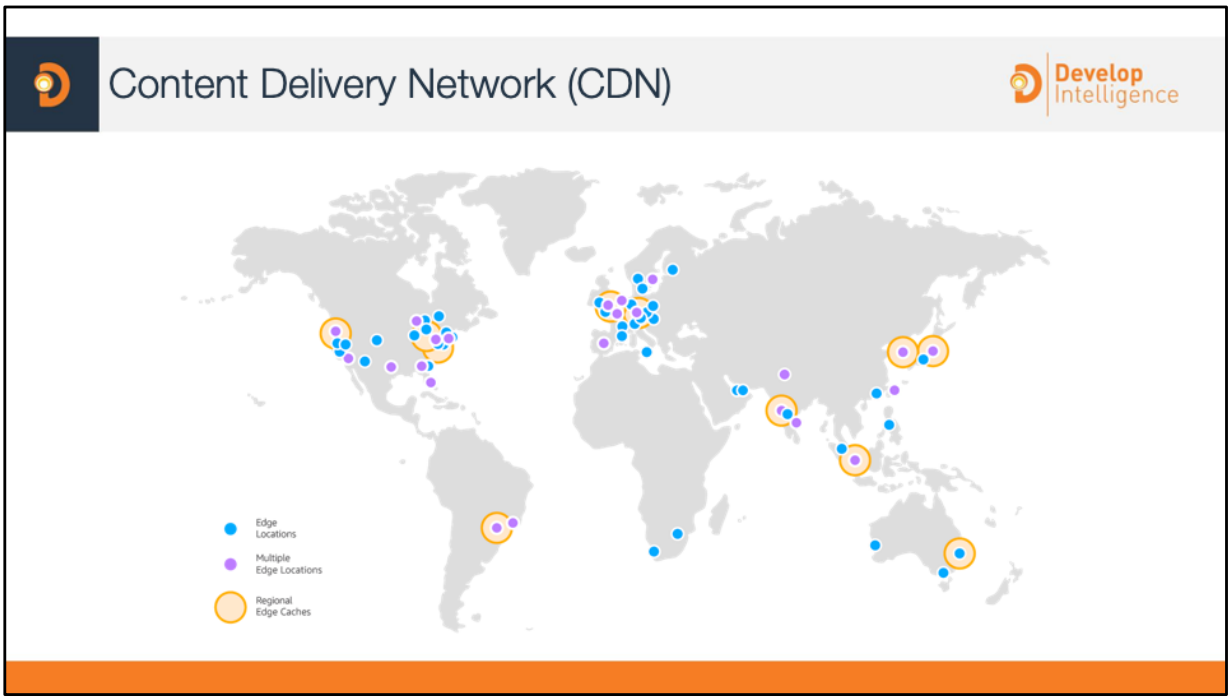
Various techniques can effectively utilize a web cache. The most basic level is client-side web caching. Data is stored in a browser rather than making repeated queries to a web server. HTTP cache headers provide the details on how long the browser can fulfill future responses from the cache for the stored web content.

On the server side, various web caching techniques can be used to improve the performance of a website.

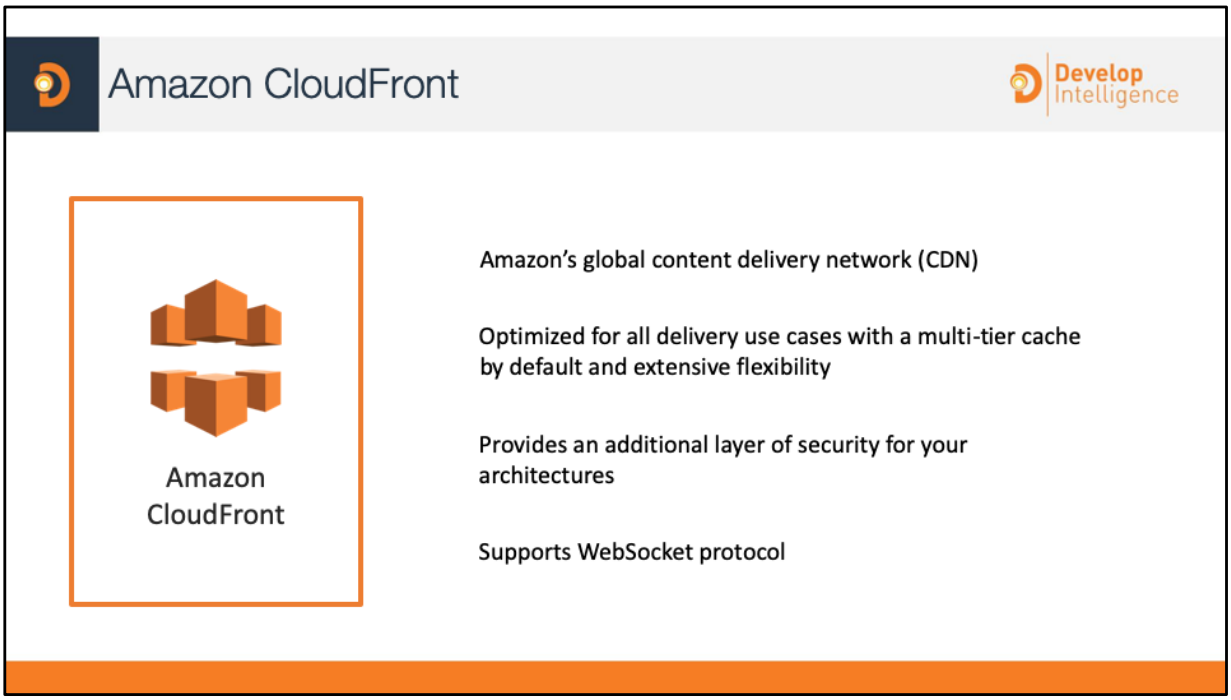
Reverse proxy caches or web application accelerators can be placed in front of application and web servers in order to serve a cached version of the HTTP responses retained from them. These caches are implemented by site administrators and act as intermediaries between the browser and origin servers. They are also commonly based on HTTP cache directives.

The Data Journey

Let's Talk About Edge Caching



When your web traffic is geo-dispersed, it's not always feasible (and certainly not cost effective) to replicate your entire infrastructure across the globe. A CDN gives you the ability to use its global network of edge locations to deliver a cached copy of web content (such as videos, webpages, images and so on) to your customers. To reduce response time, the CDN uses the nearest edge location to the customer or originating request location. Throughput is dramatically increased, given that the web assets are delivered from cache. For dynamic data, many CDNs can be configured to retrieve data from the origin servers.



Amazon CloudFront is a global CDN service that accelerates delivery of your websites, APIs, video content or other web assets. It integrates with other AWS products to give developers and businesses an easy way to accelerate content to end users with no minimum usage commitments.

Amazon CloudFront is optimized for both, providing extensive flexibility for optimizing cache behavior, coupled with network-layer optimizations for latency and throughput.

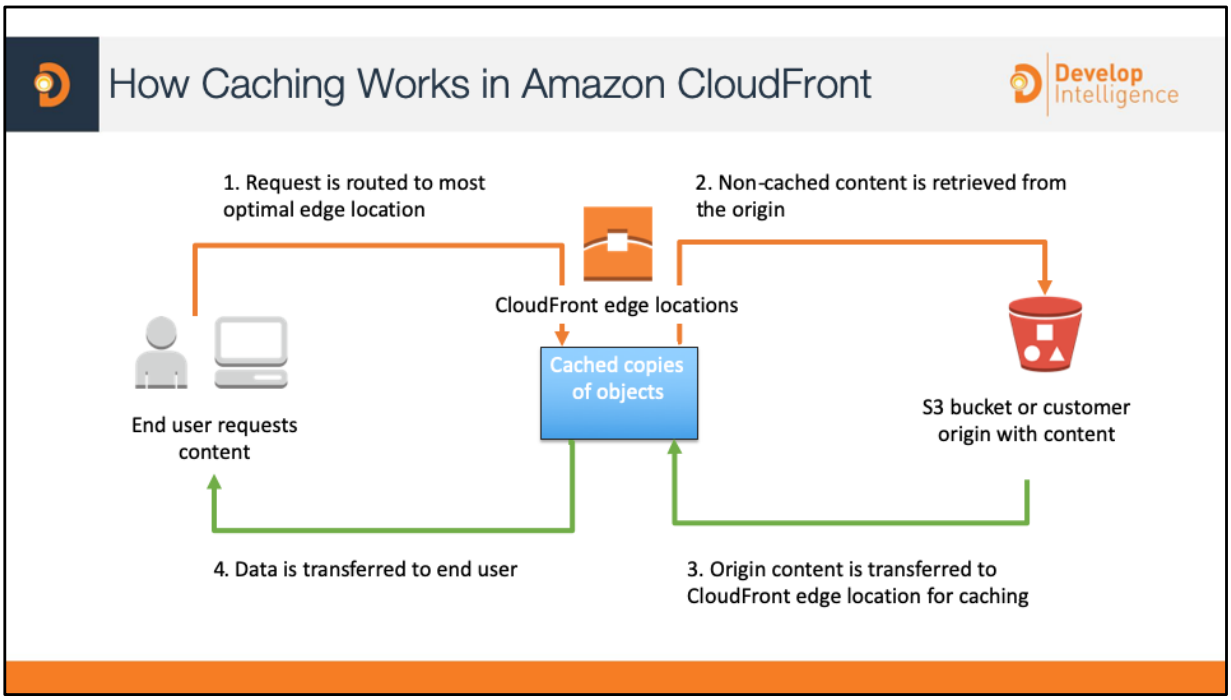
The Content Delivery Network (CDN) offers a multi-tier cache by default, with regional Edge caches that improve latency and lower the load on your origin servers when the object is not already cached at the Edge

CloudFront provides an easy and cost-effective way to distribute content with low latency and high data transfer speeds. Like other AWS services, CloudFront is a self-service, pay-per-use offering, requiring no long-term commitments or minimum fees. With CloudFront, files are delivered to end-users using a global network of edge locations.

Amazon CloudFront supports real-time, bidirectional communication over the WebSocket protocol. This persistent connection allows clients and servers to send real-time data to one another without the overhead of repeatedly opening connections. This is especially useful for communications applications such as chat, collaboration, gaming, and financial trading.

Support for WebSockets in CloudFront allows customers to manage WebSocket traffic through the same avenues as any other dynamic and static content. CloudFront's global network of edge locations brings traffic closer to users, improving responsiveness and reliability, and also allows customers to take advantage of DDoS protection using the built-in CloudFront integrations with AWS Shield and AWS WAF.

The WebSockets protocol removes the overhead of regular TCP connections by removing the need to constantly open new connections any time a client would like to receive updated data or send new information. WebSocket connections remain open until closed by the client or server, and allows them to send data to one another so long as the connection remains open.



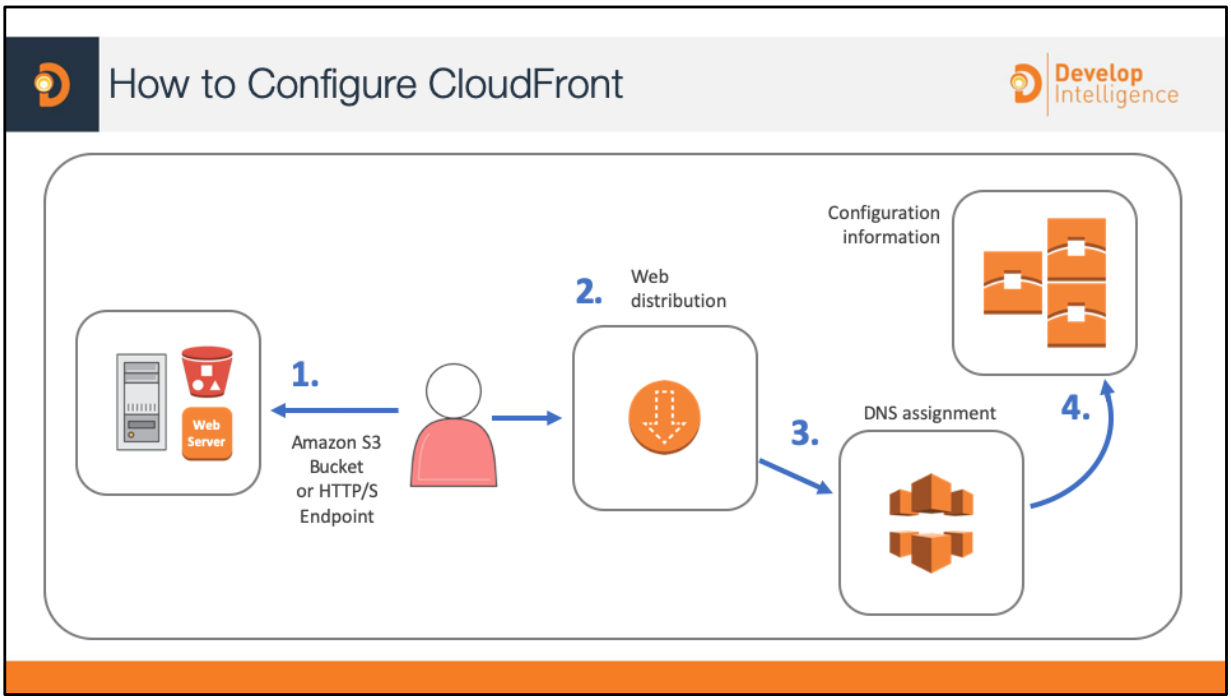
When a user requests content that you are serving with CloudFront, the user is routed to the edge location that provides the lowest latency (time delay), so content is delivered with the best possible performance. If the content is already in the edge location with the lowest latency, CloudFront delivers it immediately. If the content is not currently in that edge location, CloudFront retrieves it from an Amazon S3 bucket or an HTTP server (for example, a web server) that you have identified as the source for the definitive version of your content.

In the example above, if content isn't cached, the requested object is retrieved from the origin. Steps 1, 2, 3, and 4 are performed to retrieve and return the data requested by the user.

If content is cached, the cached object request is routed to the most optimal edge location and the cached objects are retrieved as shown in steps 1 and 4.

With the caching and acceleration technology of CloudFront, AWS can deliver all of your content from static images to user-inputted content.

Static: images, js, html, etc. with high time-to-live (TTL)
Video: rtmp and http streaming support
Dynamic: customizations and non-cacheable content
User input: http action support including Put/Post
Secure: Serve the content securely with SSL (https)



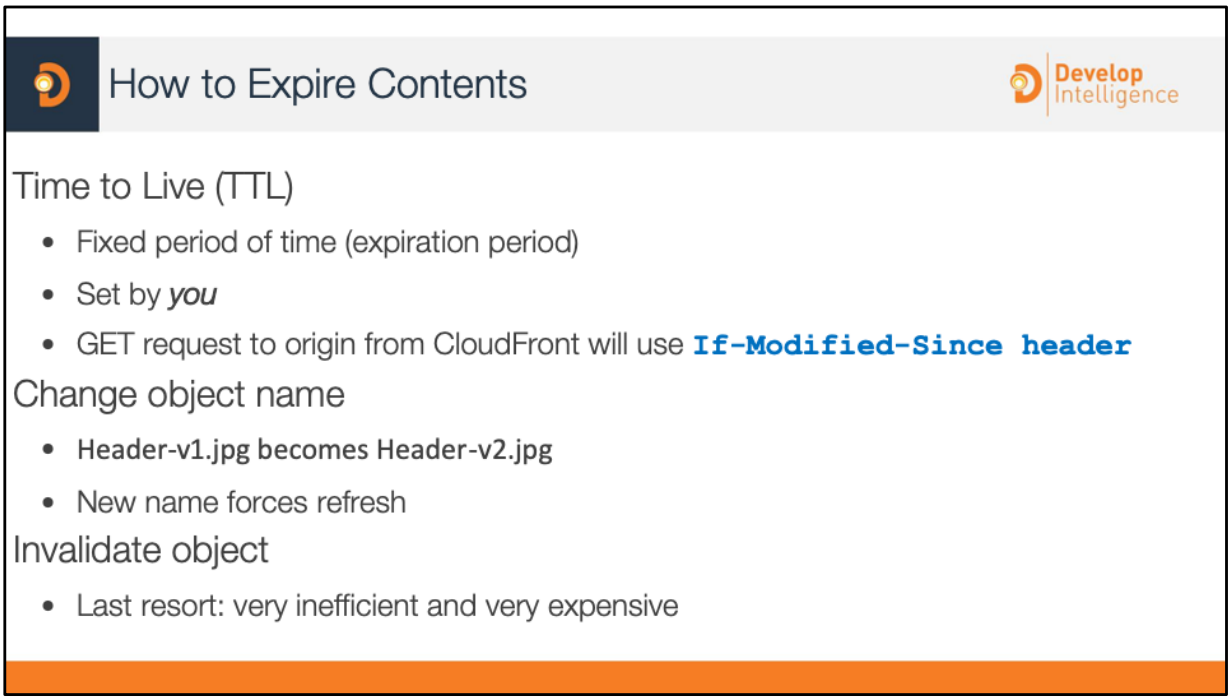
1. You specify an *origin server*, like an Amazon S3 bucket or your own HTTP server, from which CloudFront gets your files. These will be distributed from CloudFront edge locations all over the world.

An origin server stores the original, definitive version of your objects. If you're serving content over HTTP, your origin server is either an Amazon S3 bucket or an HTTP server, such as a web server. Your HTTP server can run on an Amazon Elastic Compute Cloud (Amazon EC2) instance or on server on-premises that you manage. These servers are also known as *custom origins.*

2. You create a CloudFront *distribution*, which tells CloudFront which origin servers to get your files from when users request the files through your web site or application. At the same time, you specify details such as whether you want CloudFront to log all requests and whether you want the distribution to be enabled as soon as it's created.

3. CloudFront assigns a domain name to your new distribution.

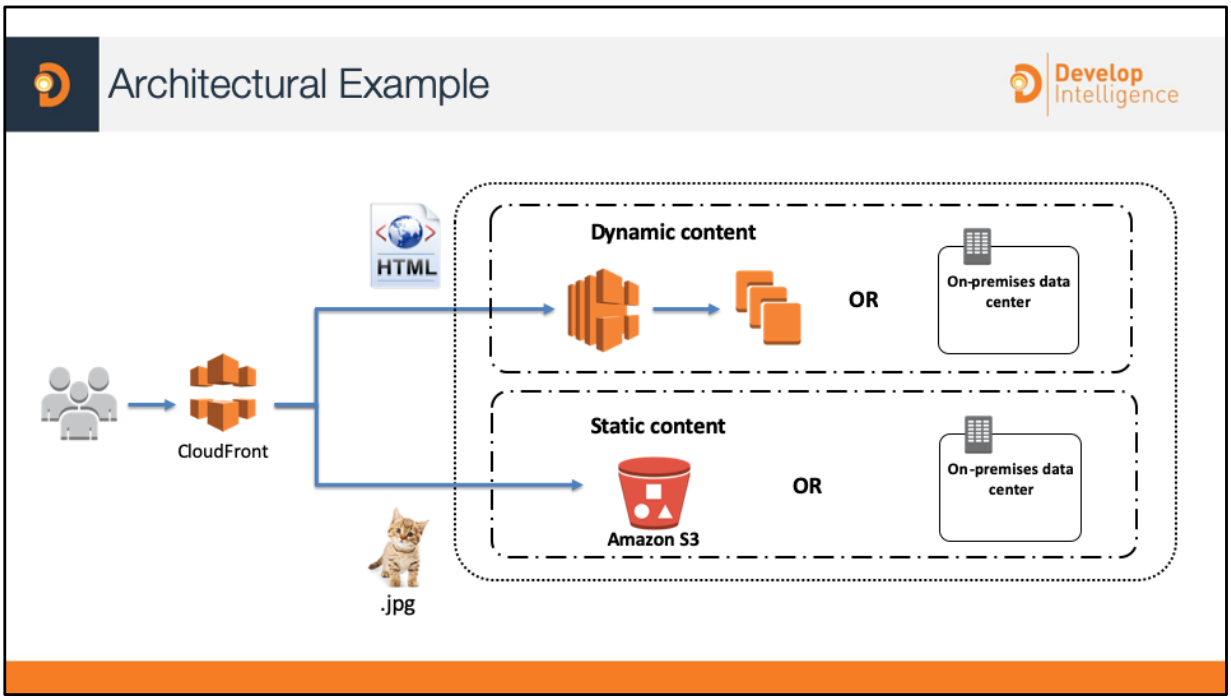
4. CloudFront sends your distribution's configuration but not the content to all of its **edge locations.** Edge locations are collections of servers in geographically dispersed data centers where CloudFront caches copies of your objects.



There are three ways to wear out cached content: the first two are much preferred, and TTL is easiest if the replacement does not need to be immediate. (For more information, see http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/RequestAndResponseBehaviorS3Origin.html.)

If you set the TTL for a particular origin to 0, CloudFront will still cache the content from that origin. It will then make a GET request with an If-Modified-Since header, thereby giving the origin a chance to signal that CloudFront can continue to use the cached content if it hasn't changed at the origin.
The second way requires more effort but is immediate (there might be some support for this in some CMS systems). Although you can update existing objects in a CloudFront distribution and use the same object names, it is not recommended. CloudFront distributes objects to edge locations only when the objects are requested, not when you put new or updated objects in your origin. If you update an existing object in your origin with a newer version that has the same name, an edge location won't get that new version from your origin until both of the listed events occur.

The third method should only be used sparingly for individual objects: it is a bad solution (the system must forcibly interact with all edge locations).



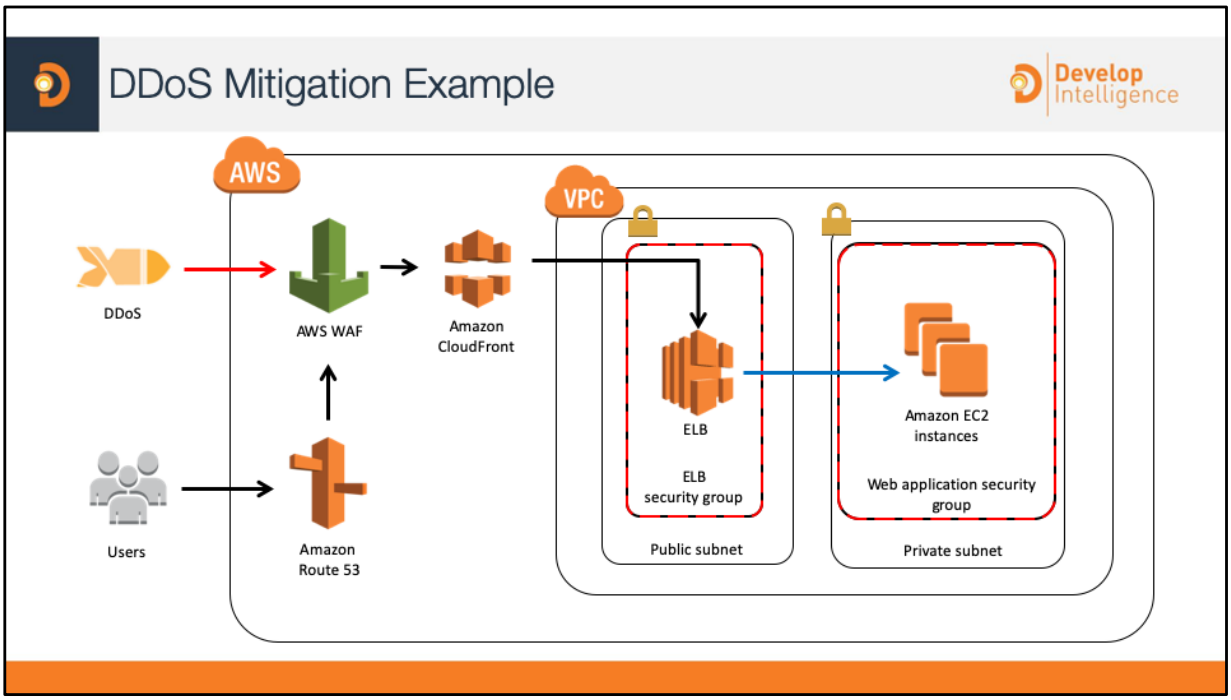
In general, you only cache static content. However, dynamic or unique content affects the performance of your application. Depending on the demand, you might still get some performance gain by caching the dynamic or unique content in Amazon S3.

Static content off-loading:

- Create absolute URL references to your static assets instead of relative.
- Store static assets in Amazon S3.
- Optimize for "Write Once Read Many."

Additionally, you can geo-restrict your content. Geo-restriction, also known as *geoblocking*, prevents users in specific geographic locations from accessing content that you are distributing through a CloudFront web distribution. To use geo-restriction, you have two options:

- Use the CloudFront geo-restriction feature to restrict access to all of the files that are associated with a distribution and to restrict access at the country level.
- Use a third-party geolocation service to restrict access to a subset of the files that are associated with a distribution or to restrict access at a finer granularity than the country level.



This is an example of a resilient architecture that can help to prevent or mitigate DDoS attacks.

The strategy to minimize the attack surface area is to (a) reduce the number of necessary Internet entry points, (b) eliminate non-critical Internet entry points, (c) separate end user traffic from management traffic, (d) obfuscate necessary Internet entry points to the level that untrusted end users cannot access them, and (e) decouple Internet entry points to minimize the effects of attacks. This strategy can be accomplished with Amazon Virtual Private Cloud.

Within AWS, you can take advantage of two forms of scaling: horizontal and vertical scaling. In terms of DDoS, there are three ways to take advantage of scaling in AWS: (1) select the appropriate instance types for your application, (2) configure services such as Elastic Load Balancing and Auto Scaling to automatically scale, and (3) use the inherent scale built into the AWS global services like Amazon CloudFront and Amazon Route 53.

Because ELB only supports valid TCP requests, DDoS attacks such as UDP and SYN floods are not able to reach your instances.
You can set a condition to incrementally add new instances to the Auto Scaling group when network traffic is high (typical of DDoS attacks).

Services that are available within AWS regions, like Elastic Load Balancing and Amazon EC2, allow you to build DDoS resiliency and scale to handle unexpected volumes of traffic within a given region. Services that are available in AWS edge locations, like Amazon CloudFront, AWS WAF, Amazon Route 53, and Amazon API Gateway, allow you to take advantage of a global network of edge locations that can provide your application with greater fault tolerance and increased scale for managing larger volumes of traffic.

The benefits of using each these services to build resiliency against infrastructure layer and application layer DDoS attacks are discussed in the following sections.

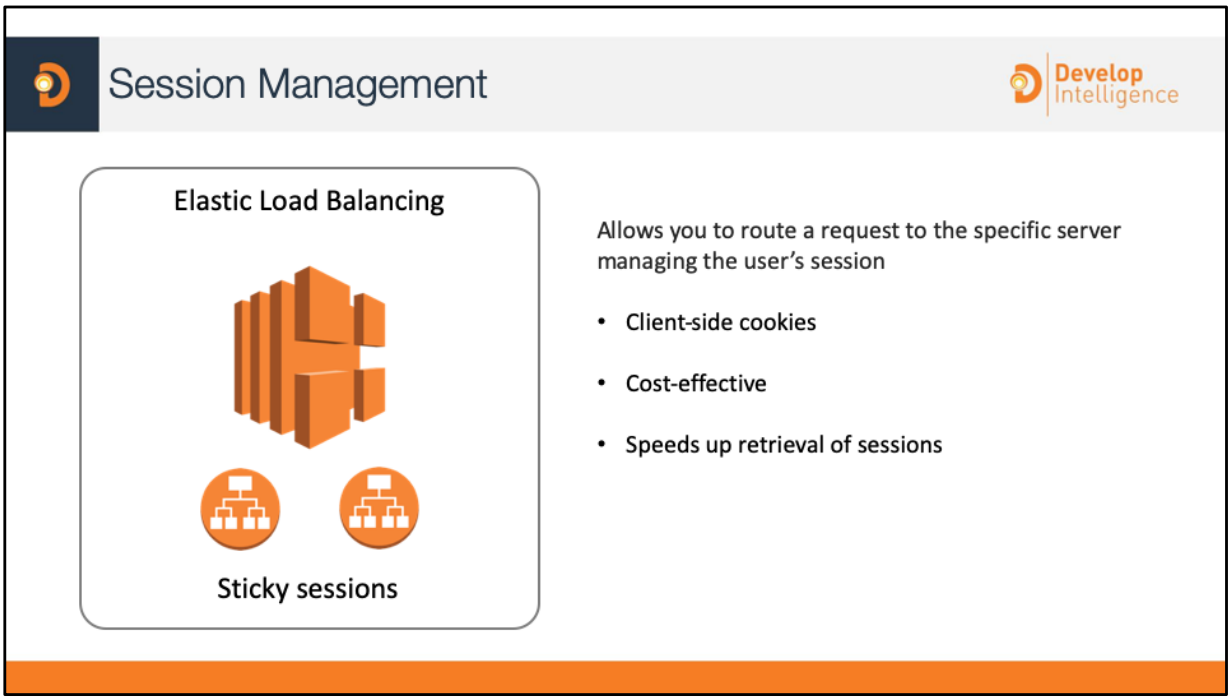
Amazon CloudFront also has filtering capabilities to ensure that only valid TCP connections and HTTP requests are made while dropping invalid requests.

A WAF (web application firewall) is a tool that applies a set of rules to HTTP traffic, in order to filter web requests based on data such as IP addresses, HTTP headers, HTTP body, or URI strings. They can be useful for mitigating DDoS attacks by offloading illegitimate traffic.

AWS now offers a managed WAF service. For more on AWS WAF, see:
http://docs.aws.amazon.com/waf/latest/developerguide/what-is-aws-waf.html

Whitepaper: AWS Best Practices for DDoS Resiliency:
https://d0.awsstatic.com/whitepapers/Security/DDoS_White_Paper.pdf

The Data Journey



It might not seem as if session management is related to caching but it is.

Basically, a web session is a sequence of HTTP transactions from the same user into an environment. HTTP was designed to transfer documents. It does not manage state. Every request is independent of previous transactions. Do you really want people to send credentials for every request made to your server? Does your server have the network and compute power needed to scale as users and their requirements increase?

Session management is authentication and access control. Common approaches to managing state include using sticky sessions or a distributed cache.

Sticky sessions, also called *session affinity*, allow you to route a request to the specific server managing the user's session. A session's validity can be determined by a number of methods; client-side cookies or parameters set at a load balancer.

Sticky sessions are cost-effective because you are storing sessions on the web servers running your applications. This eliminates network latency and speeds up retrieval of those sessions. However, in the event of failure, you are likely to lose the sessions stored on a node.

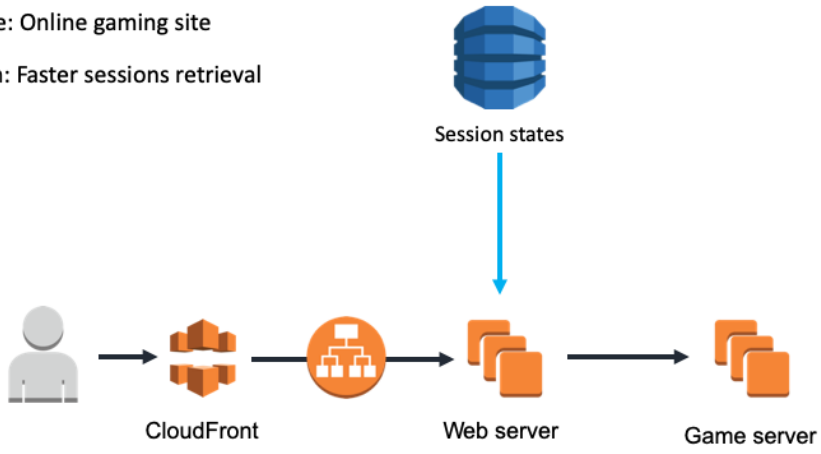
When scaling, it's possible traffic might be unequally spread across servers as active sessions prevent routing traffic to the increased capacity.

When Using DynamoDB for State Information

Develop Intelligence

Use case: Online gaming site

Problem: Faster sessions retrieval



The Data Journey

Develop Intelligence

When Should You Start Caching Your Database?

Develop Intelligence

You are concerned about response times for your customer

You find heavy-hitting, high-volume requests inundating your database

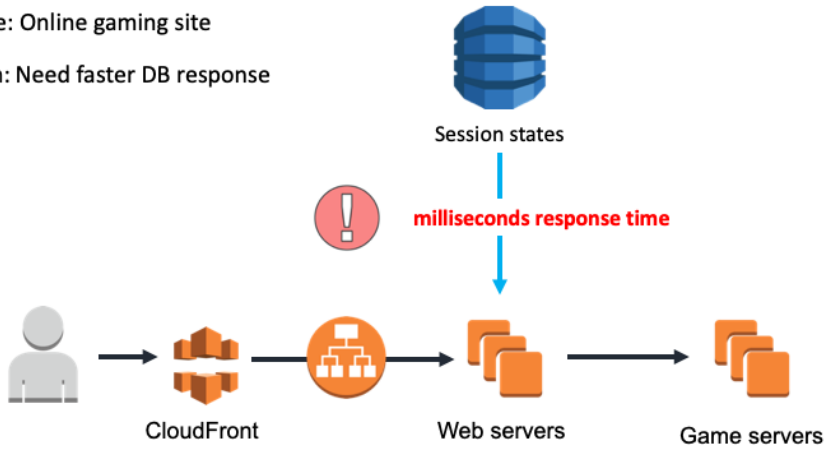
You would like to reduce your database costs

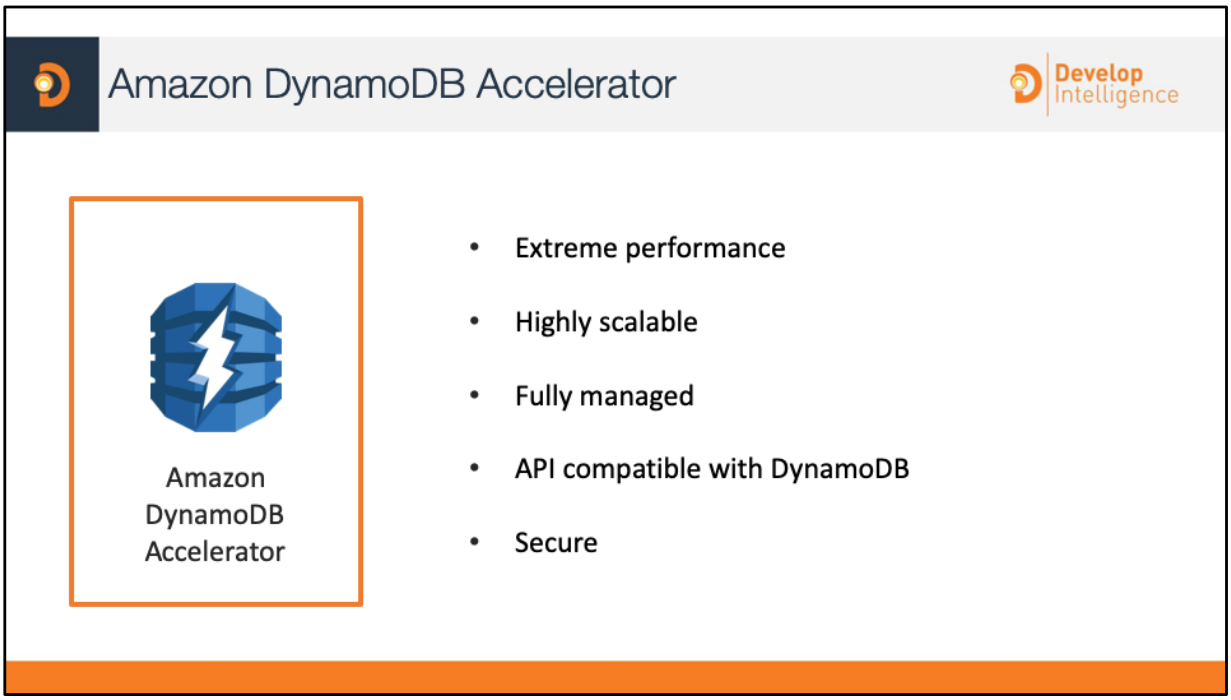
When Using DynamoDB for State Information

Develop Intelligence

Use case: Online gaming site

Problem: Need faster DB response





Extreme Performance
DynamoDB offers consistent single-digit millisecond latency. DynamoDB plus DAX have response times in microseconds for millions of requests per second for read-heavy workloads.

Highly Scalable
DAX has on-demand scaling. You can start with a three-node DAX cluster and add capacity by as required up to a ten-node cluster.

Fully Managed
Like DynamoDB, DAX is fully managed. DAX takes care of management tasks including provisioning, setup and configuration, software patching, and replicating data over nodes during scaling operations. DAX automates common administrative tasks such as failure detection, failure recovery, and software patching.

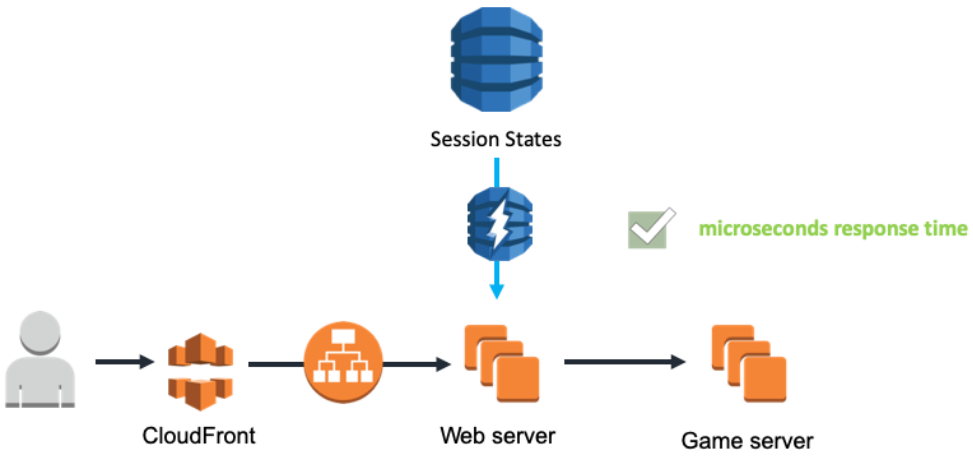
API Compatible with DynamoDB
DAX is API-compatible with DynamoDB and there is no need to make any functional application code changes. Provision a DAX cluster, use the DAX client SDK to point existing DynamoDB API calls at the DAX cluster, and DAX handles the rest.

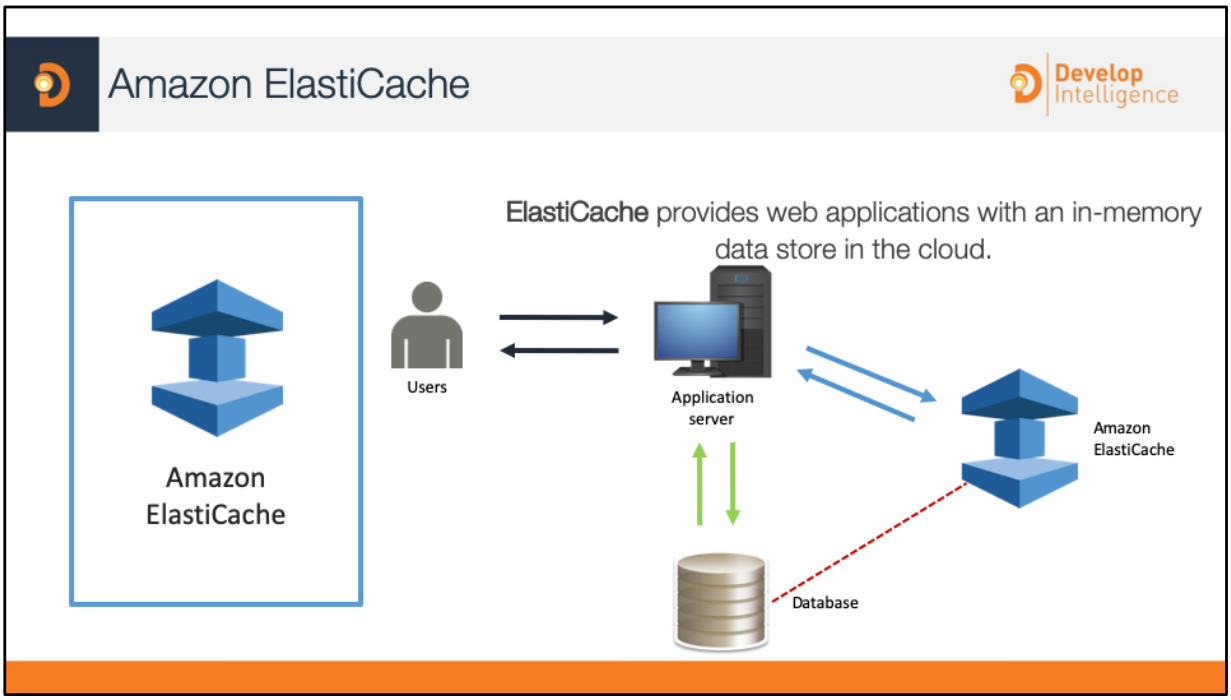
Flexible
You can provision one DAX cluster for multiple DynamoDB tables, multiple DAX clusters for a single DynamoDB table or a combination of both.

Secure
DAX fully integrates with AWS services to enhance security. You can use Identity and Access Management (IAM) to assign unique security credentials to each user and control each user's access to services and resources. Amazon CloudWatch enables you to gain system-wide visibility into resource utilization, application performance, and operational health. Integration with AWS CloudTrail enables you to easily log and audit changes to your cluster configuration. DAX supports Amazon Virtual Private Cloud (VPC) for secure and easy access from your existing applications. Tagging provides you additional visibility to help you manage your DAX clusters.

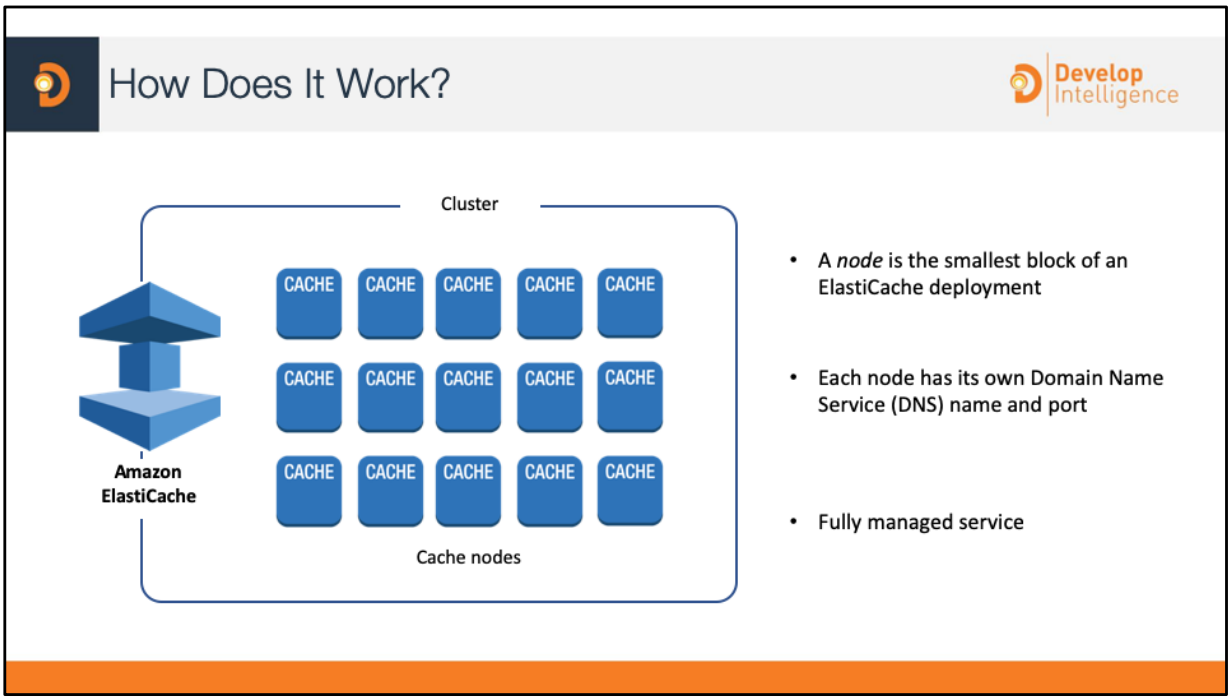
The retrieval of cached data reduces the read load on existing DynamoDB tables. This means it may reduce provisioned read capacity and lower overall operational costs.

When Using DynamoDB for State Information



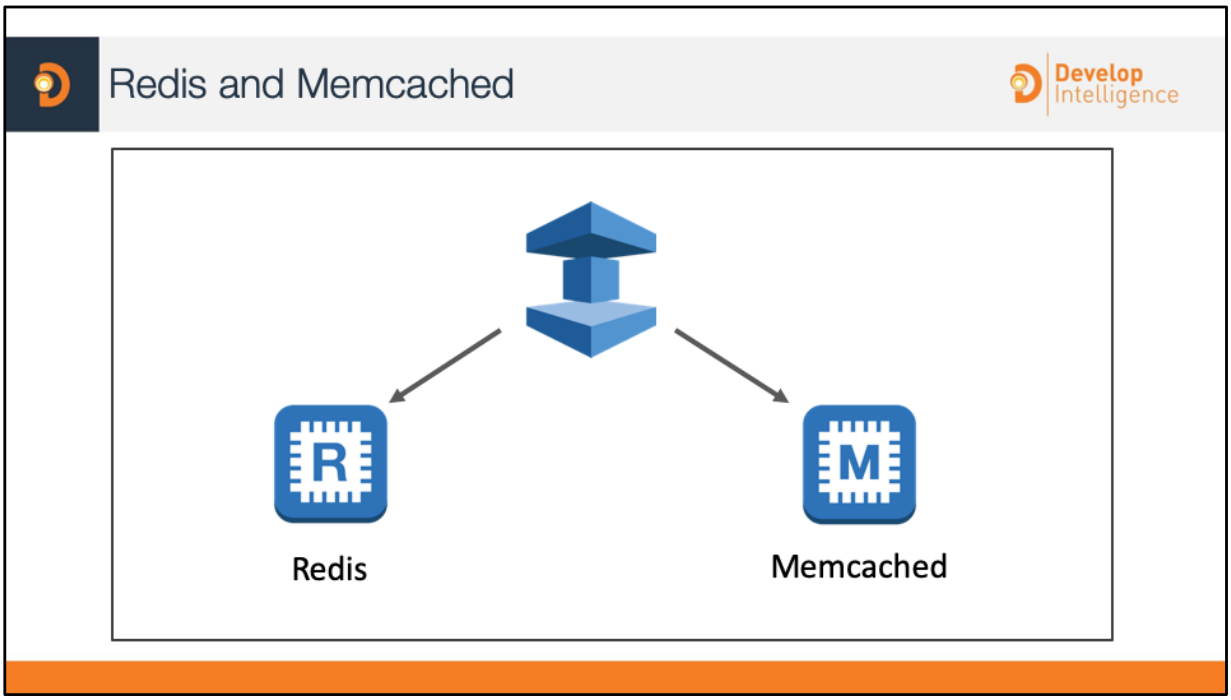


Amazon ElastiCache is a web service used to deploy, operate, and scale an in-memory cache in the cloud. ElastiCache improves the performance of web applications by allowing you to retrieve information from fast, managed, in-memory data stores, instead of relying on slower disk-based databases. Whenever possible, applications will retrieve data from ElastiCache and defer to databases when data isn't found in cache.



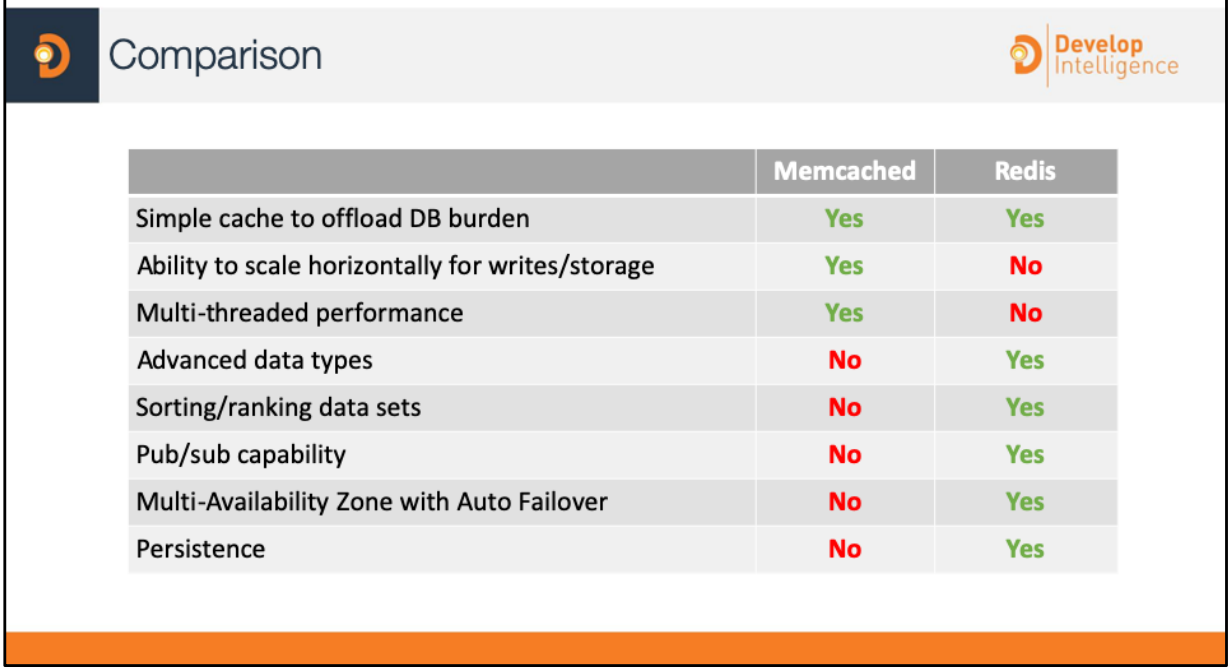
A *cache node* is the smallest building block of an ElastiCache deployment. It can exist in isolation from other nodes, or in some relationship to other nodes, which is also known as a *cluster*.

Amazon ElastiCache is fully managed, so you're not stuck running cache nodes you're not using, and if you need more, your cluster can scale out to accommodate your capacity needs.

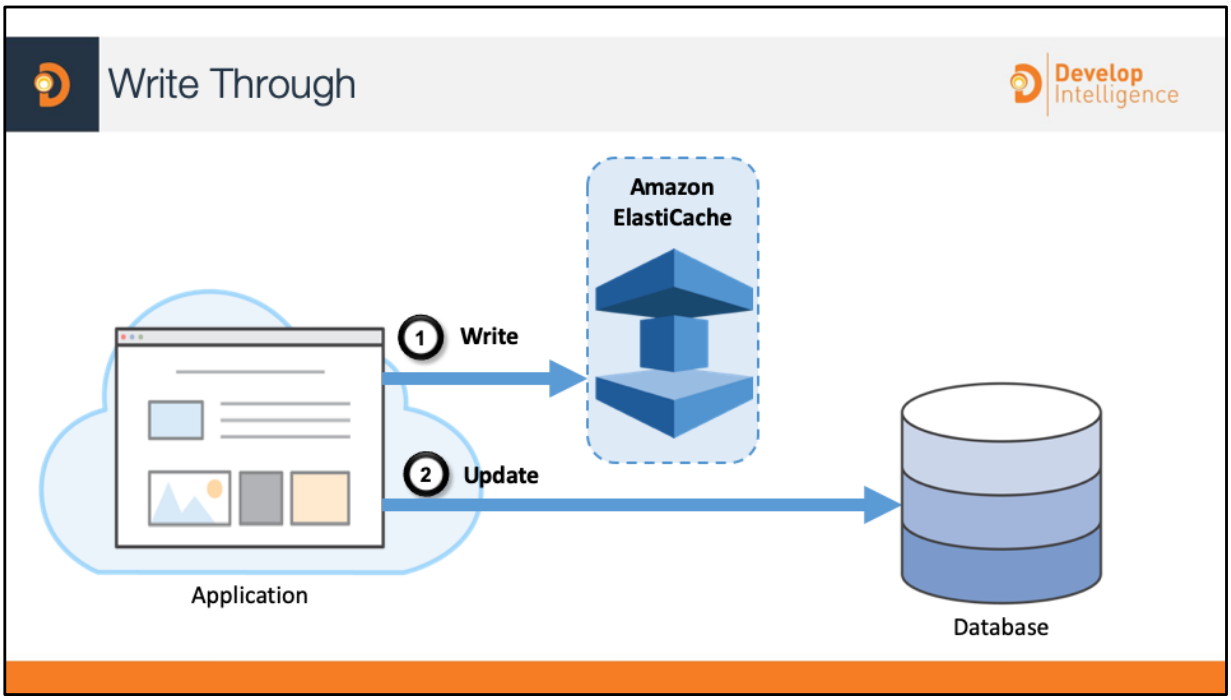


ElastiCache with Memcached can scale up to 20 nodes per cluster, while ElastiCache for Redis can scale up to 90 nodes for increased data access performance. ElastiCache supports Amazon VPC, enabling you to isolate your cluster to the IP ranges you choose for your nodes.

ElastiCache runs on the same highly reliable infrastructure used by other AWS services. For Redis workloads, ElastiCache provides high-availability through Multi-AZ with automatic failover. For Memcached workloads, data is partitioned across all nodes in the cluster in case of node failure. With ElastiCache, you no longer need to perform management tasks such as hardware provisioning, software patching, monitoring, failure recovery, and backups. ElastiCache continuously monitors your clusters to keep your workloads up and running so that you can focus on application development.



	Memcached	Redis
Simple cache to offload DB burden	Yes	Yes
Ability to scale horizontally for writes/storage	Yes	No
Multi-threaded performance	Yes	No
Advanced data types	No	Yes
Sorting/ranking data sets	No	Yes
Pub/sub capability	No	Yes
Multi-Availability Zone with Auto Failover	No	Yes
Persistence	No	Yes



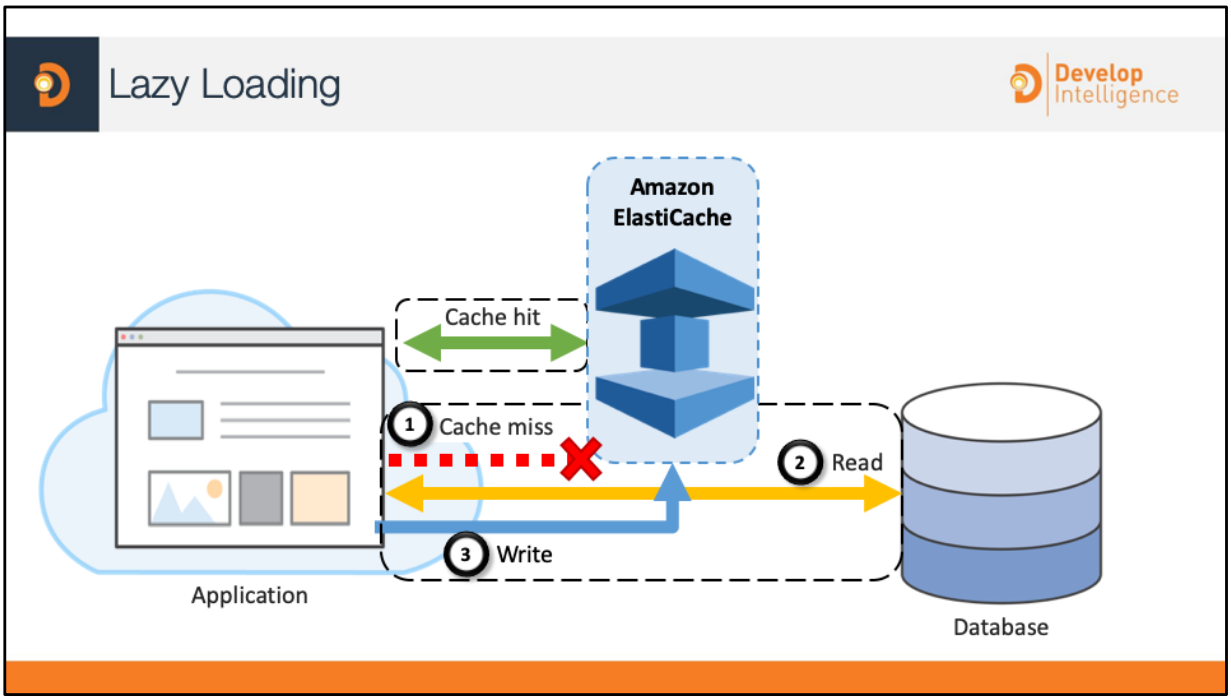
The write through strategy adds data or updates data in the cache whenever data is written to the database.

Advantages of Write Through

- The data in the cache is never stale. Since the data in the cache is updated every time it is written to the database, the data in the cache is always current.

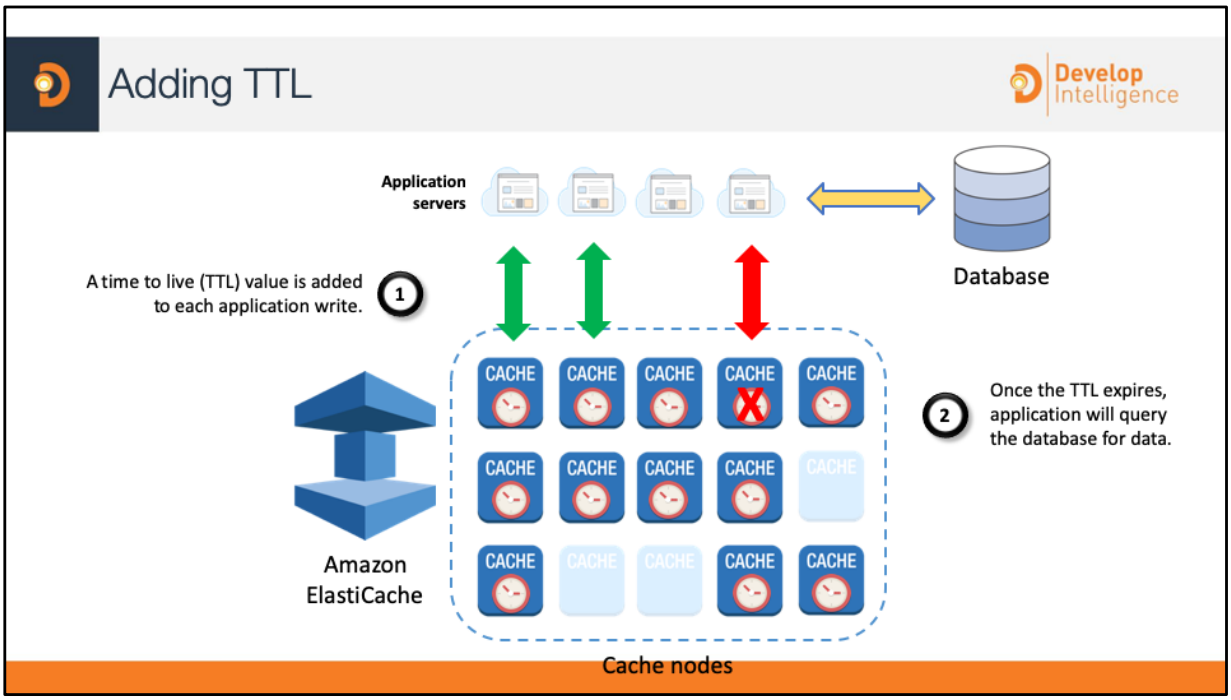
Disadvantages of Write Through

- Write penalty: Every write involves two trips - a write to the cache and a write to the database .
- Missing data: When a new node is created to scale up or to replace a failed node, the node does not contain all data. Data continues to be missing until it is added or updated in the database. In this scenario, you might choose to use a lazy caching approach to repopulate the cache.
- Unused data: Since most data is never read, there can be a lot of data in the cluster that is never read.
- Cache churn: The cache may be updated often if certain records are updated repeatedly.

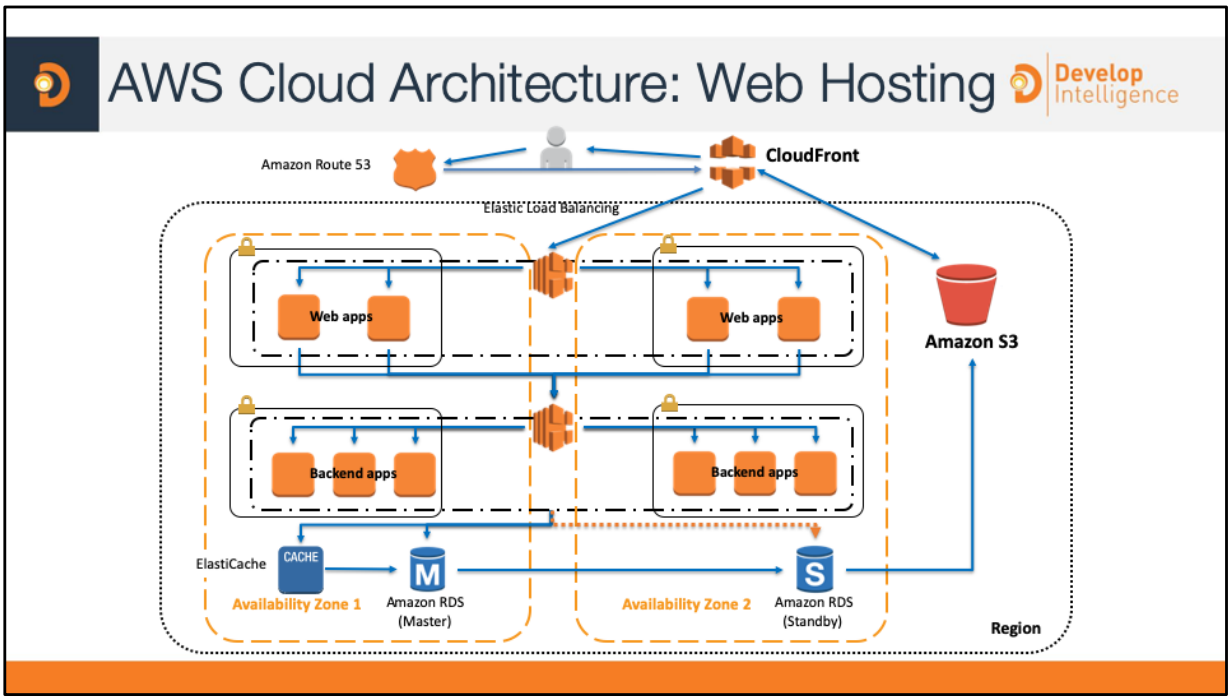


Lazy loading is a caching strategy that loads data into the cache only when necessary. In this deployment, ElastiCache sits between your application and the data store, or database, that it accesses. Whenever your application requests data, it first makes the request to the ElastiCache cache. If the data exists in the cache and is current, a cache hit occurs and ElastiCache returns the data to your application. Otherwise, your application requests the data from your data store which returns the data to your application. Your application then writes the data received to the cache so it can be retrieved more quickly the next time it is requested.

With lazy loading, only requested data is cached. Since most data is never requested, lazy loading avoids filling up the cache with unnecessary data. However, there is a cache miss penalty. Each cache miss results in three trips, which can cause a noticeable delay in data getting to the application. Also, if data is only written to the cache when there is a cache miss, data in the cache can become stale since there are no updates to the cache when data is changed in the database. The Write Through and Adding TTL strategies address this issue, and we'll cover those next.



Lazy loading allows for stale data, while Write Through ensures that data is always fresh, but may populate the cache with unnecessary data. By adding a time to live, or TTL, value to each write, you can enjoy the advantages of each strategy and largely avoid cluttering up the cache with data. TTL is an integer value or key that specifies the number of seconds or milliseconds, depending on the in-memory engine, until the key expires. When an application attempts to read an expired key, it is treated as though the data is not found in cache, meaning that the database is queried and the cache is updated. This keeps data from getting too stale and requires that values in the cache are occasionally refreshed from the database.



Traditional web hosting architecture implements a common three-tier web application model that separates the architecture into presentation, application, and persistence layers. Scalability is provided by adding hosts at the presentation, persistence, or application layers.

Web application hosting in the AWS Cloud

The traditional web hosting architecture is easily portable to the cloud services provided by AWS products with only a small number of modifications, but the first question that should be asked concerns the value of moving a classic web application hosting solution into the AWS cloud. If you decide that the cloud is right for you, you'll need a suitable architecture.

- *Amazon Route 53* provides DNS services to simplify domain management and zone APEX support.
- *Amazon CloudFront* provides edge caching for high-volume contents.
- *Elastic Load Balancing* spreads traffic to web server Auto Scaling groups in this diagram.
- *Exterior Firewall* is moved to every web server instance via security groups.
- *Backend Firewall* is moved to every back-end instance.
- *App server load balancer* on Amazon EC2 instances spreads traffic over the app server cluster.
- *Amazon ElastiCache* provides caching services for the app, which removes load from the database tier.