







## Module 7: Designing Web-Scale Media



Welcome to Module 7: Designing Web-Scale Media.

# What's in This Module



-  **Part 1:** Storing Web-Accessible Content with Amazon S3
-  **Part 2:** Caching with Amazon CloudFront (Week 10)
-  **Part 3:** Managing NoSQL databases (Week 5 Extras)
-  **Part 4:** Storing Relational Data in Amazon RDS. (Week 5 Extras)

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

In part 1, you will discover how to store web accessible content with Amazon Simple Storage Service, or Amazon S3.

In part 2, you will review caching with Amazon CloudFront.

In part 3, you will learn about the features of managing NoSQL databases, or DBs.

In part 4, we'll learn how to store relational data in Amazon Relational Database Service, or Amazon RDS.

You will finish this module by completing a lab: Implementing a Serverless Architecture with AWS Managed Services.

# Part 1: Storing Web-Accessible Content with Amazon S3

Introducing Part 1: Storing Web-Accessible Content with Amazon S3.

# Reality of Web-Based Applications

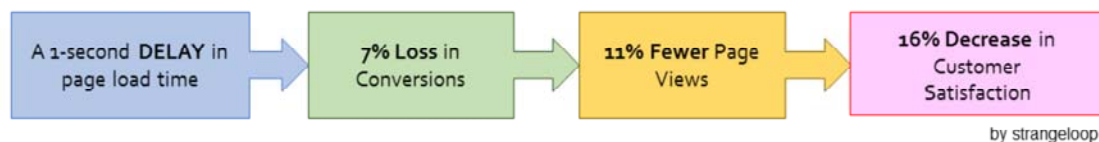


Your applications being unavailable:

- 📦 Leads to revenue loss.
- 📦 Has an impact on customer loyalty and your brand image.

Performance translates to:

- 📦 High page views.
- 📦 Better customer experience.
- 📦 Higher conversion rates.



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Web-based applications must be available, and they shouldn't take long to load. If your applications are unavailable, it could lead to revenue loss. It could also affect customer loyalty and your brand image. A 1-second delay can translate into a 16 percent decrease in customer satisfaction.

Performance translates to higher page views, better customer experience, and higher conversion rates.

# Deliver All Your Content



**Secure**

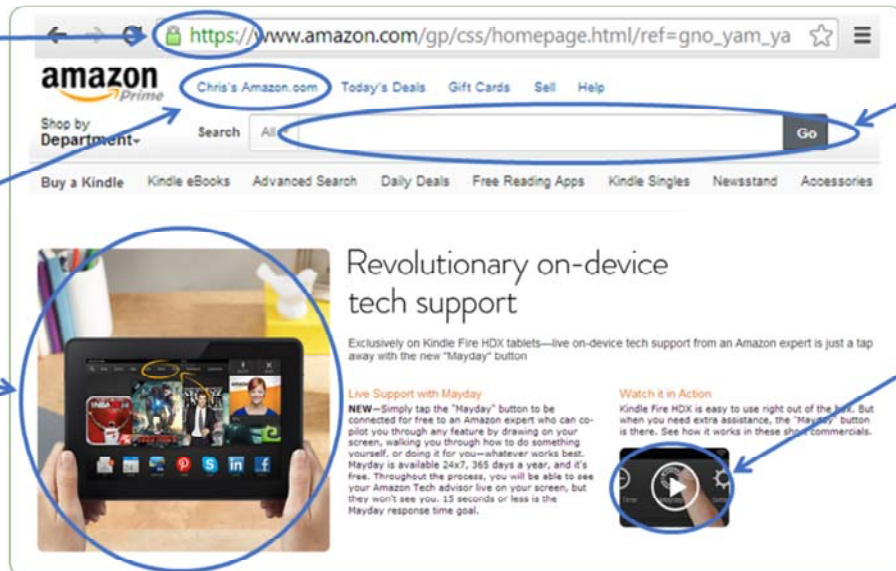
**Dynamic**  
Zero time to  
live (TTL)

**Static**

Can be  
cached!

User  
Input

Video



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Let's take a moment to review Amazon.com and consider where we might deliver content.

In the address bar, you can note that the application is transported over `https://`, or port 443. This tells you that Amazon.com uses Secure Sockets Layer, or SSL. You can then note that the application content is dynamic because it's personalized for Chris, which tells you that it has zero time to live, or TTL. You can also search for items by entering user input. The static images can be cached and stored in Amazon S3.

With the caching and acceleration technology of Amazon CloudFront, we can deliver all of your content, including static images, videos, and content based on user input.

How should web-accessible content be stored? Let's discuss how to store content in Amazon S3.

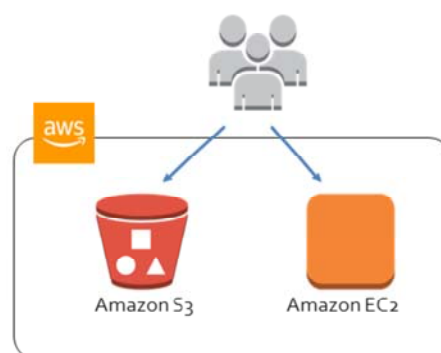
# Web Storage Pattern



Static assets—which are assets that aren't going to change, such as text, images, and video content—can be stored in Amazon S3.

## Problem

- ❏ Delivery of large files from a web server can become a problem in terms of network latency.
- ❏ User-generated content needs to be distributed across all your web servers.



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Static assets—content that isn't going change, such as text, images, and video content—can be stored in Amazon S3.

Let's consider this pattern. An Amazon Elastic Compute Cloud—or Amazon EC2—instance is being used to serve static images and videos, which is not a proper use for that Amazon EC2 instance. Instead, you can store those images and videos in an Amazon S3 bucket that is enabled for public web hosting. You know that Amazon S3 will be durable, and that it will serve content quickly. When you use Amazon S3 to store static content, you don't need to worry about scaling out an Amazon EC2 instance to serve videos and images.

Delivery of large files from a web server can become a problem in terms of network latency.

The use of Amazon S3 eliminates the need to worry about network loads and data capacity on your web servers. You can also use Amazon CloudFront, which is a content delivery network—or CDN—that can be used as a global caching layer in front of Amazon S3 to accelerate content to your end users. User-generated content needs to be distributed across all your web servers.

# Web Storage Pattern



## Solution

- Store static asset files (CSS, multimedia, etc.) in Amazon S3 and deliver the files from there.
- Objects that are stored in Amazon S3 can be accessed directly by users if set to public.
- Amazon S3 performs backups in at least three different data centers for extremely high durability.



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

One solution is to store static asset files—such as Cascading Style Sheets, or CSS; multimedia, and others—in Amazon S3 and serve the files from there.

Objects that are stored in Amazon S3 can be accessed directly by users if the bucket is public.

Amazon S3 performs backups in at least three different data centers, and thus has extremely high durability.

# Choosing a Region



## Choosing a Region:

- ❏ Performance depends on the **proximity** to your users.
- ❏ Collocating with compute and other AWS resources affects performance.



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

How do you choose a Region for Amazon S3? You will want to have geographical proximity to your users. You should also collocate it with your Amazon EC2 instances, and with other AWS services that upload and download bucket objects. It is better to locate services as close as possible to the Amazon S3 buckets that must be accessed. Collocating all of your relevant AWS services and Amazon S3 buckets in the same Region can help reduce costs and latencies.

Data that is stored in an Amazon S3 bucket is replicated across multiple data centers in a geographical region. It is important to carefully select the AWS Region where you store an object because response latencies grow when requests have to travel long distances over the internet.



# Naming Buckets and Keys



## Amazon S3 maintains bucket's object names:

- 📁 Keys are represented as strings.
- 📁 Namespace is partitioned by a prefix with multiple partitions to store object keys.
- 📁 The object names you choose dictate keymap.

## Amazon S3 is object file storage:

- 📁 You want a bucket capable of routinely exceeding 3,500 PUT/POST/DELETE or 5,500 GET requests per second.
- 📁 Increase your read or write performance exponentially.

[Learn more about Amazon S3 performance.](#)

### Naming Rules

- Starts with reversed timestamp, epoch time or counter.
- First 3-4 characters are critical.
- Randomize prefixes to be distributed across bucket's partitions.

Learn more. 

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon S3 maintains a “map” of each bucket’s object names, or “keys.” Internally, keys are all represented in Amazon S3 as strings with the format of bucket name/key name. The namespace is partitioned by a prefix. Each bucket contains multiple partitions to store object keys. Keys partition the bucket key name space, and define the partition that they are stored in.

Each key is stored and retrieved based on the name that is provided when the object is first put into Amazon S3, which means that the object names that you choose actually dictate the keymap. Naming is critical when you want to achieve a processing rate of more than 100 transactions per second—or TPS—for the Amazon S3 bucket. Although buckets can be named with any alphanumeric character, you should follow some naming rules to ensure that you can reference your bucket by using the following convention: <bucketname>.s3.amazonaws.com. What makes a good or a bad Amazon S3 key? Plain globally unique identifiers, or GUIDs; timestamps; and running counters are not good candidates for key names because they use repeating prefixes. A good key starts with a reversed timestamp, a reversed epoch time, or a reversed counter. The most significant part of the name is the first 3-4 characters of a key. The best practice for creating good Amazon S3 keys is to randomize prefixes as much as possible so they can be distributed across a bucket's partitions.

For additional information, go to the following page about bucket restrictions and naming:  
<https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html>

Amazon S3 is object file storage, and it behaves like a flat file system. It automatically scales to high request rates. For example, an application can achieve at least 3,500 PUT, POST, or DELETE

requests and 5,500 GET requests per second per prefix in a bucket. There are no limits to the number of prefixes in a bucket. It is simple to increase your read or write performance exponentially. For example, if you create 10 prefixes in an Amazon S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second.

For more information, go to the following page about Amazon S3 performance:

<https://docs.aws.amazon.com/AmazonS3/latest/dev/request-rate-perf-considerations.html>

# Amazon S3 Partitions



Amazon S3 automatically partitions your buckets according to the prefixes of your files.

Partition 1

```
/album1/photo1.jpg  
/album1/photo2.jpg  
/album1/photo3.jpg  
/album1/photo4.jpg
```

Partition 2

```
/album2/photo1.jpg  
/album2/photo2.jpg  
/album2/photo3.jpg  
/album2/photo4.jpg
```

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

As mentioned previously, each bucket contains multiple partitions to store object keys. Amazon S3 automatically partitions your buckets according to the prefixes of your files. Keys partition the bucket keyname space, and they define the partition they are stored in. Additionally, keys in Amazon S3 are partitioned by prefix.

This example shows two partitions. The partitions are album1 and album2, which have photos. If you start having more than 300 requests per second, you can create a hotspot because many of the albums are named in the same way. They are all in the same partition because they are all in the same folder.

# Example: Automatic Partitions



Amazon S3 automatically partitions your buckets according to the prefixes of your files.

Partition 1

```
/album1/photo1.jpg  
/album1/photo2.jpg  
/album1/photo3.jpg  
/album1/photo4.jpg
```

Partition 2

```
/album2/photo1.jpg  
/album2/photo2.jpg  
/album2/photo3.jpg  
/album2/photo4.jpg
```

← If a process requests  
← the photos from  
← album1, all requests  
← will go to the **same**  
← **partition.**

**Anti-pattern**

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

[Learn more.](#)

If you store a few objects and you need to add more objects, incrementing the last number of your file name is fine. If you store thousands of objects in an Amazon S3 bucket, this file naming convention could limit performance when Amazon S3 partitions your bucket. Adding a random value to the key can provide higher performance.

If you issue queries against Amazon S3 buckets that store a large number of objects and the data is not partitioned, these queries might affect the GET request rate limits in Amazon S3, and then lead to Amazon S3 exceptions. To prevent errors, you should partition your data. Additionally, consider tuning your Amazon S3 request rates. For more information, see [Request Rate and Performance Considerations](#).

# Adding a Hex Hash Prefix



You can add a hex hash prefix (or any base) to the key to reduce the chance that a request reads from the same partition multiple times at once.

/2e4f/album2/photo7.jpg
/3a79/album1/photo2.jpg
/7b54/album1/photo3.jpg
/8761/album2/photo6.jpg
/921c/album1/photo4.jpg
/b71f/album1/photo1.jpg
/ba65/album2/photo5.jpg
/c34a/album2/photo8.jpg

Best practice

← If a process  
← requests the first  
← four photos, the  
← requests will be  
← **split between the**  
← **partitions.**

**Note:** This example partitioning scheme is for demonstration purposes only. Your Amazon S3 bucket may be partitioned differently based on how its contents are named.

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

In the example of photo storage, if the Amazon S3 bucket receives more than 300 requests per second, the read request can have limited performance because it must increment numerically. For example, a query must begin at photo1.jpg when it needs to locate photo200.jpg.

Like the photo example, Amazon S3 workloads often have content that is organized by user ID, or game ID, album ID, or other similar semi-meaningless identifier. These identifiers often involve incrementally increasing numbers, or date-time constructs of various types. The difficulty of this naming choice for Amazon S3 scaling is twofold:

- First, all new content will necessarily end up being owned by a single partition.
- Second, all the partitions that hold slightly older—and generally less hot—content get cold more quickly than partitions that use other naming conventions. This practice effectively wastes the available operations per second that each partition can support by making all of the old content cold over time.

A way to make this work with Amazon S3 at nearly any request rate is to simply reverse the order of the digits in this identifier. In other words, you can use seconds of precision for date- or time-based identifiers. These identifiers effectively start with a random number, which fans out the transactions across many potential child partitions. Each of those child partitions scales, so that no meaningful operations per second budget are wasted. In fact, Amazon S3 even has an algorithm to detect this parallel type of write

pattern, and it will automatically create multiple child partitions from the same parent simultaneously. This process increases the system's operations per second budget when request heat is detected.

Amazon S3 buckets are somewhat similar to Hashtable data structures in Java. Each bucket has multiple partitions to store objects keys, which helps spread the transaction load. You can add a hex hash prefix or any base in front of the bucket name to reduce the chance that a request reads from the same partition multiple times at once. If a process requests the first four photos, the requests will be split between the partitions. Larger bases provide better distribution.

In summary, keys partition the key name space in a bucket, and they define the partition that they are stored in. The most significant part is the first 3-4 characters of a key, and this number might increase along with the amount of objects that are stored in a bucket. The best practice of coming up with good Amazon S3 keys is to randomize their prefixes as much as possible so that they can be better distributed across a bucket's partitions.

## What If Your Bucket Is Constantly Under Load?



To more easily retrieve your files, maintain a **secondary index** and hash all key names.

Example: A secondary index stored in an Amazon DynamoDB table:

App ID	Month	Key
38229	04	/2e4f/var/logs/system.log.6
49113	03	/3a79/var/logs/system.log.1
49113	03	/7b54/var/logs/system.log.2
67812	04	/8761/var/logs/system.log.5

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

What if your bucket is constantly under load? To more easily retrieve your files, you can use the prefix hash method of naming objects in the Amazon S3 bucket, and store it in Amazon DynamoDB. That way, it creates a secondary index—which acts like a legend—that points to where all of the files are located.