**S3 Request Rate and Performance Considerations**

This topic discusses Amazon S3 best practices for optimizing performance depending on your request rates. If your workload in an Amazon S3 bucket routinely exceeds 100 PUT/LIST/DELETE requests per second or more than 300 GET requests per second, follow the guidelines in this topic to ensure the best performance and scalability.

Amazon S3 scales to support very high request rates. If your request rate grows steadily, Amazon S3 automatically partitions your buckets as needed to support higher request rates. However, if you expect a rapid increase in the request rate for a bucket to more than 300 PUT/LIST/DELETE requests per second or more than 800 GET requests per second, we recommend that you open a support case to prepare for the workload and avoid any temporary limits on your request rate. To open a support case, go to [AWS Support](https://aws.amazon.com/premiumsupport/).

**Note**

The Amazon S3 best practice guidelines in this topic apply only if you are routinely processing 100 or more requests per second. If your typical workload involves only occasional bursts of 100 requests per second and fewer than 800 requests per second, you don't need to follow these guidelines.

If your workload in Amazon S3 uses server-side encryption with AWS Key Management Service (SSE-KMS), go to [Limits](http://docs.aws.amazon.com/kms/latest/developerguide/limits.html) in the *AWS Key Management Service Developer Guide* to get more information on the request rates supported for your use case.

The Amazon S3 best practice guidance given in this topic is based on two types of workloads:

* **Workloads that include a mix of request types –** If your requests are typically a mix of GET, PUT, DELETE, or GET Bucket (list objects), choosing appropriate key names for your objects ensures better performance by providing low-latency access to the Amazon S3 index. It also ensures scalability regardless of the number of requests you send per second.
* **Workloads that are GET-intensive –** If the bulk of your workload consists of GET requests, we recommend using the Amazon CloudFront content delivery service.

**Topics**

* [Workloads with a Mix of Request Types](https://docs.aws.amazon.com/AmazonS3/latest/dev/request-rate-perf-considerations.html#workloads-with-mix-request-types)
* [GET-Intensive Workloads](https://docs.aws.amazon.com/AmazonS3/latest/dev/request-rate-perf-considerations.html#get-workload-considerations)

**Workloads with a Mix of Request Types**

When uploading a large number of objects, customers sometimes use sequential numbers or date and time values as part of their key names. For example, you might choose key names that use some combination of the date and time, as shown in the following example, where the prefix includes a time stamp:

examplebucket/*2013-26-05-15-00-00*/cust1234234/photo1.jpg

examplebucket/*2013-26-05-15-00-00*/cust3857422/photo2.jpg

examplebucket/*2013-26-05-15-00-00*/cust1248473/photo2.jpg

examplebucket/*2013-26-05-15-00-00*/cust8474937/photo2.jpg

examplebucket/*2013-26-05-15-00-00*/cust1248473/photo3.jpg

...

examplebucket/*2013-26-05-15-00-01*/cust1248473/photo4.jpg

examplebucket/*2013-26-05-15-00-01*/cust1248473/photo5.jpg

examplebucket/*2013-26-05-15-00-01*/cust1248473/photo6.jpg

examplebucket/*2013-26-05-15-00-01*/cust1248473/photo7.jpg

...

The sequence pattern in the key names introduces a performance problem. To understand the issue, let's look at how Amazon S3 stores key names.

Amazon S3 maintains an index of object key names in each AWS region. Object keys are stored in UTF-8 binary ordering across multiple partitions in the index. The key name dictates which partition the key is stored in. Using a sequential prefix, such as time stamp or an alphabetical sequence, increases the likelihood that Amazon S3 will target a specific partition for a large number of your keys, overwhelming the I/O capacity of the partition. If you introduce some randomness in your key name prefixes, the key names, and therefore the I/O load, will be distributed across more than one partition.

If you anticipate that your workload will consistently exceed 100 requests per second, you should avoid sequential key names. If you must use sequential numbers or date and time patterns in key names, add a random prefix to the key name. The randomness of the prefix more evenly distributes key names across multiple index partitions. Examples of introducing randomness are provided later in this topic.

**Note**

The guidelines provided for the key name prefixes in the following section also apply to the bucket name. When Amazon S3 stores a key name in the index, it stores the bucket names as part of the key name (for example, examplebucket/object.jpg).

**Example 1: Add a Hex Hash Prefix to Key Name**

One way to introduce randomness to key names is to add a hash string as prefix to the key name. For example, you can compute an MD5 hash of the character sequence that you plan to assign as the key name. From the hash, pick a specific number of characters, and add them as the prefix to the key name. The following example shows key names with a four-character hash.

**Note**

A hashed prefix of three or four characters should be sufficient. We strongly recommend using a hexadecimal hash as the prefix.

examplebucket/*232a*-*2013-26-05-15-00-00*/cust1234234/photo1.jpg

examplebucket/*7b54*-*2013-26-05-15-00-00*/cust3857422/photo2.jpg

examplebucket/*921c*-*2013-26-05-15-00-00*/cust1248473/photo2.jpg

examplebucket/*ba65*-*2013-26-05-15-00-00*/cust8474937/photo2.jpg

examplebucket/*8761*-*2013-26-05-15-00-00*/cust1248473/photo3.jpg

examplebucket/*2e4f*-*2013-26-05-15-00-01*/cust1248473/photo4.jpg

examplebucket/*9810*-*2013-26-05-15-00-01*/cust1248473/photo5.jpg

examplebucket/*7e34*-*2013-26-05-15-00-01*/cust1248473/photo6.jpg

examplebucket/*c34a*-*2013-26-05-15-00-01*/cust1248473/photo7.jpg

...

Note that this randomness does introduce some interesting challenges. Amazon S3 provides a [GET Bucket (List Objects)](http://docs.aws.amazon.com/AmazonS3/latest/API/RESTBucketGET.html) operation, which returns a UTF-8 binary ordered list of key names. Here are some side-effects:

* Because of the hashed prefixes, however, the listing will appear randomly ordered.
* The problem gets compounded if you want to list object keys with specific date in the key name. The preceding example uses a four-character hex hash, so there are 65536 possible character combinations (four character prefix, and each character can be any of the hex characters 0-f). So you will be sending 65536 List Bucket requests each with a specific prefix that is a combination of 4-digit hash and the date. For example, suppose that you want to find all keys with 2013-26-05 in the key name. Then you will send List Bucket requests with prefixes such [0-f][0-f][0-f][0-f]2013-26-05.

You can optionally add more prefixes in your key name, before the hash string, to group objects. The following example adds animations/ and videos/ prefixes to the key names.

examplebucket/*animations/232a*-2013-26-05-15-00-00/cust1234234/animation1.obj

examplebucket/*animations/7b54*-2013-26-05-15-00-00/cust3857422/animation2.obj

examplebucket/*animations/921c*-2013-26-05-15-00-00/cust1248473/animation3.obj

examplebucket/*videos/ba65*-2013-26-05-15-00-00/cust8474937/video2.mpg

examplebucket/*videos/8761*-2013-26-05-15-00-00/cust1248473/video3.mpg

examplebucket/*videos/2e4f*-2013-26-05-15-00-01/cust1248473/video4.mpg

examplebucket/*videos/9810*-2013-26-05-15-00-01/cust1248473/video5.mpg

examplebucket/*videos/7e34*-2013-26-05-15-00-01/cust1248473/video6.mpg

examplebucket/*videos/c34a*-2013-26-05-15-00-01/cust1248473/video7.mpg

...

In this case, the ordered list returned by the GET Bucket (List Objects) operation will be grouped by the prefixes animations and videos.

**Note**

Again, the prefixes you add to group objects should not have sequences, or you will again overwhelm a single index partition.

**Example 2: Reverse the Key Name String**

Suppose your application uploads objects with key names whose prefixes include an increasing sequence of application IDs.

examplebucket/*2134857*/data/start.png

examplebucket/*2134857*/data/resource.rsrc

examplebucket/*2134857*/data/results.txt

examplebucket/*2134858*/data/start.png

examplebucket/*2134858*/data/resource.rsrc

examplebucket/*2134858*/data/results.txt

examplebucket/*2134859*/data/start.png

examplebucket/*2134859*/data/resource.rsrc

examplebucket/*2134859*/data/results.txt

In this key naming scheme, write operations will overwhelm a single index partition. If you reverse the application ID strings, however, you have the key names with random prefixes:

examplebucket/*7584312*/data/start.png

examplebucket/*7584312*/data/resource.rsrc

examplebucket/*7584312*/data/results.txt

examplebucket/*8584312*/data/start.png

examplebucket/*8584312*/data/resource.rsrc

examplebucket/*8584312*/data/results.txt

examplebucket/*9584312*/data/start.png

examplebucket/*9584312*/data/resource.rsrc

examplebucket/*9584312*/data/results.txt

Reversing the key name string lays the groundwork for Amazon S3 to start with the following partitions, one for each distinct first character in the key name. The examplebucket refers to the name of the bucket where you upload application data.

examplebucket/7

examplebucket/8

examplebucket/9

This example illustrates how Amazon S3 can use the first character of the key name for partitioning, but for very large workloads (more than 2000 requests per seconds or for bucket that contain billions of objects), Amazon S3 can use more characters for the partitioning scheme. Amazon S3 can automatically split these partitions further as the key count and request rate increase over time.

**GET-Intensive Workloads**

If your workload is mainly sending GET requests, in addition to the preceding guidelines, you should consider using Amazon CloudFront for performance optimization.

Integrating Amazon CloudFront with Amazon S3, you can distribute content to your users with low latency and a high data transfer rate. You will also send fewer direct requests to Amazon S3, which will reduce your costs.

For example, suppose that you have a few objects that are very popular. Amazon CloudFront fetches those objects from Amazon S3 and caches them. Amazon CloudFront can then serve future requests for the objects from its cache, reducing the number of GET requests it sends to Amazon S3. For more information, go to the [Amazon CloudFront](https://aws.amazon.com/cloudfront/) product detail page.