**Serverless Microservices Development:**

[SERVERLESS EVENT-DRIVEN COMPUTE](https://aws.amazon.com/lambda/):

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume—there is no charge unless your code is running. With Lambda, you can run code for virtually any type of application or backend service—all with zero administration. You can set up your code to automatically trigger from other AWS services, or you can call it directly from any web or mobile app.

[SERVERLESS CONTAINERS](https://aws.amazon.com/fargate/)

AWS Fargate is a compute engine for Amazon ECS, a high-performance [container](https://aws.amazon.com/containers/) orchestration service that allows you to run containers without having to manage servers or clusters. With AWS Fargate, you no longer have to provision, configure, and scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing. AWS Fargate removes the need for you to interact with or think about servers or clusters.

[API MANAGEMENT](https://aws.amazon.com/api-gateway/)

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It offers a comprehensive platform for [API management](https://aws.amazon.com/api-gateway/api-management/). Amazon API Gateway allows you to process hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.

[MESSAGE QUEUE SERVICE](https://aws.amazon.com/sqs/)

Amazon SQS is a fast, reliable, scalable, fully managed [message queuing](https://aws.amazon.com/message-queue/) service. Amazon SQS makes it simple and cost-effective to decouple the components of a cloud application. You can use Amazon SQS to transmit any volume of data, without losing messages or requiring other services to be always available.

[PUBLISH/SUBSCRIBE MESSAGING](https://aws.amazon.com/sns/)

Amazon SNS is a highly available, fully managed [pub/sub messaging](https://aws.amazon.com/pub-sub-messaging/) service that enables you to decouple microservices, distributed systems, and serverless applications. Using Amazon SNS topics, your publisher systems can fan out messages to a large number of subscriber endpoints for parallel processing, including Amazon SQS queues, AWS Lambda functions, and HTTP/S webhooks.

[ORCHESTRATION](https://aws.amazon.com/step-functions/)

AWS Step Functions makes it easy to coordinate the components of distributed applications and microservices using visual workflows. Building applications from individual components that each perform a discrete function lets you scale and change applications quickly.

[MYSQL AND POSTGRESQL DATABASE](https://aws.amazon.com/rds/aurora/)

Amazon Aurora Serverless is a MySQL and PostgreSQL compatible relational database engine that combines the speed and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases. Amazon Aurora provides up to five times better performance than MySQL with the security, availability, and reliability of a commercial database at one-tenth the cost.

[NO SQL DATABASE](https://aws.amazon.com/dynamodb/)

Amazon DynamoDB is a fast and flexible [NoSQL database](https://aws.amazon.com/nosql/) service for applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed database and supports both document and key-value data models. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, ad-tech, IoT, and many other applications.

[OBJECT STORAGE](https://aws.amazon.com/s3/)

Amazon S3 is [object storage](https://aws.amazon.com/what-is-cloud-object-storage/) with a simple web service interface to store and retrieve any amount of data from anywhere on the web. It is designed to deliver 99.999999999% durability, and scales past trillions of objects worldwide.

**API interaction for your code:**

**Resources**

* [**https://docs.aws.amazon.com/general/latest/gr/api-retries.html**](https://docs.aws.amazon.com/general/latest/gr/api-retries.html)
* [**https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-scan.html**](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-scan.html)

**DynamoDB best practices:**

* In general, Scan operations are less efficient than other operations in DynamoDB. A Scan operation always scans the entire table or secondary index. It then filters out values to provide the result you want, essentially adding the extra step of removing data from the result set.
* For faster response times, design your tables and indexes so that your applications can use Query instead of Scan. (For tables, you can also consider using the *GetItem* and *BatchGetItem* APIs.)
* By default, the Scan operation processes data sequentially, the throughput of a Scan is constrained by the maximum throughput of a single partition. The larger the table or index being scanned, the more time the Scan will take to complete this can be addressed by parallel scans.
* Amazon DynamoDB supports two types of secondary indexes:
* **Global secondary index**—An index with a partition key and a sort key that can be different from those on the base table. A global secondary index is considered "global" because queries on the index can span all of the data in the base table, across all partitions. A global secondary index has no size limitations and has its own provisioned throughput settings for read and write activity that are separate from those of the table.
* **Local secondary index**—An index that has the same partition key as the base table, but a different sort key. A local secondary index is "local" in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value. As a result, the total size of indexed items for any one partition key value can't exceed 10 GB. Also, a local secondary index shares provisioned throughput settings for read and write activity with the table it is indexing.

**S3 Best Practices:**

Resources

* <https://docs.aws.amazon.com/AmazonS3/latest/dev/optimizing-performance.html>
* <https://aws.amazon.com/about-aws/whats-new/2018/07/amazon-s3-announces-increased-request-rate-performance/>

Key Points

* Although S3 bucket names are globally unique, each bucket is stored in a Region that you select when you create the bucket. To optimize performance, we recommend that **you access the bucket from Amazon EC2 instances in the same AWS Region when possible.** This helps reduce network latency and data transfer costs.
* Use the SDK,

For example, the SDKs include logic to automatically retry requests on HTTP 503 errors

The SDKs also provide the Transfer Manager, which automates horizontally scaling connections to achieve thousands of requests per second, using byte-range requests where appropriate. It’s important to use the latest version of the AWS SDKs to obtain the latest performance optimization features.

* If a workload is sending repeated GET requests for a common set of objects, you can use a cache such as Amazon CloudFront or Amazon ElastiCache to optimize performance
* Amazon S3 now provides increased performance to support at **least 3,500 requests per second to add data and 5,500 requests per second to retrieve data**, which can save significant processing time for no additional charge.
* This S3 request rate performance increase removes any previous guidance to randomize object prefixes to achieve faster performance. That means you can now use logical or sequential naming patterns in S3 object naming without any performance implications
* Objects over **5GB should be uploaded using multipart upload**