Deployment strategies and best practices for optimizing performance on Azure Search

03/01/2019 • 8 minutes to read • Contributors

In this article

This article describes best practices for advanced scenarios with sophisticated requirements for scalability and availability.

Develop baseline numbers

When optimizing for search performance, you should focus on reducing query execution time. To do that, you need to know what a typical query load looks like. The following guidelines can help you arrive at baseline query numbers.

1. Pick a target latency (or maximum amount of time) that a typical search request should take to complete.
2. Create and test a real workload against your search service with a realistic dataset to measure these latency rates.
3. Start with a low number of queries per second (QPS) and the gradually increase the number executed in the test until the query latency drops below the defined target latency. This is an important benchmark to help you plan for scale as your application grows in usage.
4. Wherever possible, reuse HTTP connections. If you are using the Azure Search .NET SDK, this means you should reuse an instance or SearchIndexClient instance, and if you are using the REST API, you should reuse a single HttpClient.
5. Vary the substance of query requests so that search occurs over different parts of your index. Variation is important because if you continually execute the same search

requests, caching of data will start to make performance look better than it might
with a more disparate query set.

6. Vary the structure of query requests so that you get different types of queries. Not every search query performs at the same level. For example, a document lookup or search suggestion is typically faster than a query with a significant number of facets and filters. Test composition should include various queries, in roughly the same ratios as you would expect in production.

While creating these test workloads, there are some characteristics of Azure Search to keep in mind:

- It is possible overload your service by pushing too many search queries at one time. When this happens, you will see HTTP 503 response codes. To avoid a 503 during testing, start with various ranges of search requests to see the differences in latency rates as you add more search requests.

- Azure Search does not run indexing tasks in the background. If your service handles query and indexing workloads concurrently, take this into account by either introducing indexing jobs into your query tests, or by exploring options for running indexing jobs during off peak hours.

Note

Visual Studio Load Testing is a really good way to perform your benchmark tests as it allows you to execute HTTP requests as you would need for executing queries against Azure Search and enables parallelization of requests.

Scaling for high query volume and throttled requests

When you are receiving too many throttled requests, or exceed your target latency rates from an increased query load, you can look to decrease latency rates in one of two ways:

1. **Increase Replicas:** A replica is like a copy of your data allowing Azure Search to load balance requests against the multiple copies. All load balancing and replication of data across replicas is managed by Azure Search and you can alter the number of replicas allocated for your service at any time. You can allocate up to 12 replicas in a Standard search service and 3 replicas in a Basic search service. Replicas can be adjusted either from the Azure portal or PowerShell.
2. **Increase Search Tier:** Azure Search comes in a number of tiers and each of these tiers

2. **Increase Search Tier:** Azure Search comes in a number of tiers and each of these tiers offers different levels of performance. In some cases, you may have so many queries that the tier you are on cannot provide sufficiently low latency rates, even when replicas are maxed out. In this case, you may want to consider leveraging one of the higher search tiers such as the Azure Search S3 tier that is well-suited for scenarios with large numbers of documents and extremely high query workloads.

Scaling for slow individual queries

Another reason for high latency rates is a single query taking too long to complete. In this case, adding replicas will not help. Two options possible options that might help include the following:

1. **Increase Partitions** A partition is a mechanism for splitting your data across extra resources. Adding a second partition splits data into two, a third partition splits it into three, and so forth. One positive side-effect is that slower queries sometimes perform faster due to parallel computing. We have noted parallelization on low selectivity queries, such as queries that match many documents, or facets providing counts over a large number of documents. Since significant computation is required to score the relevancy of the documents, or to count the numbers of documents, adding extra partitions helps queries complete faster.

   There can be a maximum of 12 partitions in Standard search service and 1 partition in the basic search service. Partitions can be adjusted either from the Azure portal or PowerShell.

2. **Limit High Cardinality Fields:** A high cardinality field consists of a facetable or filterable field that has a significant number of unique values, and as a result, consumes significant resources when computing results. For example, setting a Product ID or Description field as facetable/filterable would count as high cardinality because most of the values from document to document are unique. Wherever possible, limit the number of high cardinality fields.

3. **Increase Search Tier:** Moving up to a higher Azure Search tier can be another way to improve performance of slow queries. Each higher tier provides faster CPUs and more memory, both of which have a positive impact on query performance.

Scaling for availability

Replicas not only help reduce query latency but can also allow for high availability. With a single replica, you should expect periodic downtime due to server reboots after software updates or for other maintenance events that will occur. As a result, it is important to consider if your application requires high availability of searches (queries) as well as writes (indexing events). Azure Search offers SLA options on all the paid search offerings with the following attributes:

- 2 replicas for high availability of read-only workloads (queries)
- 3 or more replicas for high availability of read-write workloads (queries and indexing)

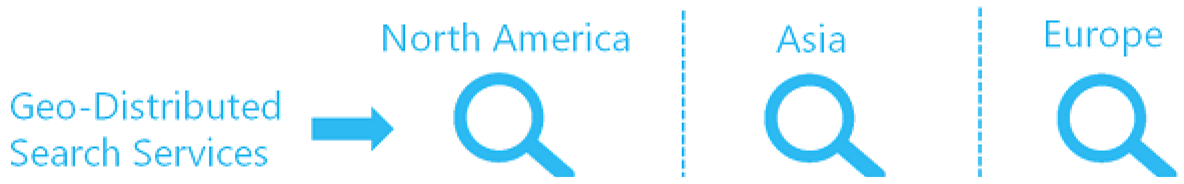
For more details on this, please visit the Azure Search Service Level Agreement.

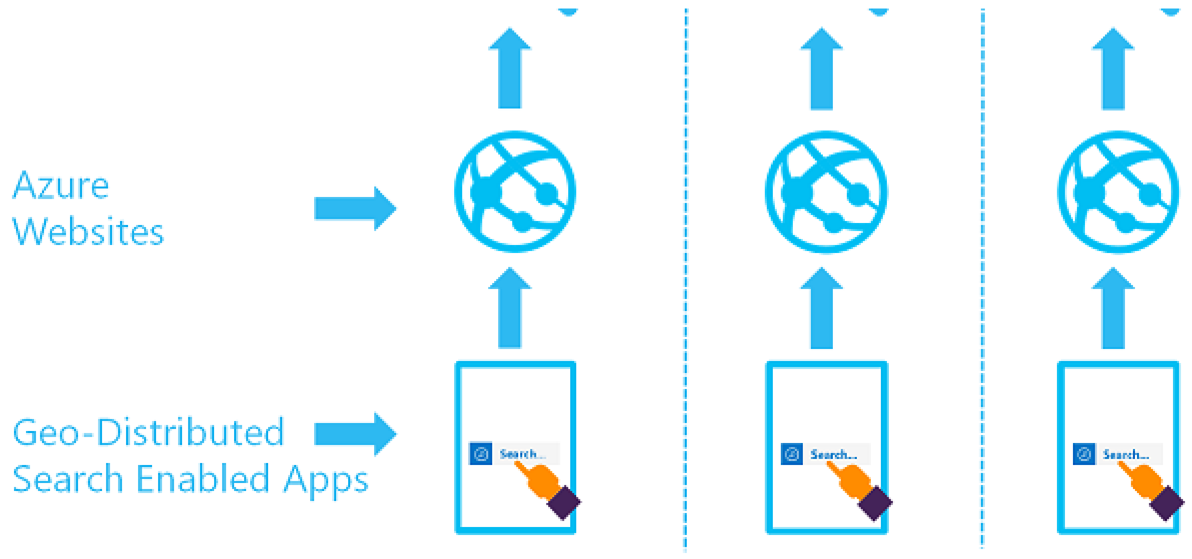
Since replicas are copies of your data, having multiple replicas allows Azure Search to do machine reboots and maintenance against one replica while allowing query execution to continue on other replicas. Conversely, if you take replicas away, you'll incur query performance degradation, assuming those replicas were an under-utilized resource.

Scaling for geo-distributed workloads and geo-redundancy

For geo-distributed workloads, users who are located far from the data center hosting Azure Search will have higher latency rates. One mitigation is to provision multiple search services in regions with closer proximity to these users. Azure Search does not currently provide an automated method of geo-replicating Azure Search indexes across regions, but there are some techniques that can be used that can make this process simple to implement and manage. These are outlined in the next few sections.

The goal of a geo-distributed set of search services is to have two or more indexes available in two or more regions where a user is routed to the Azure Search service that provides the lowest latency as seen in this example:





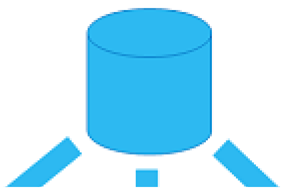
Keeping data in sync across multiple Azure Search services

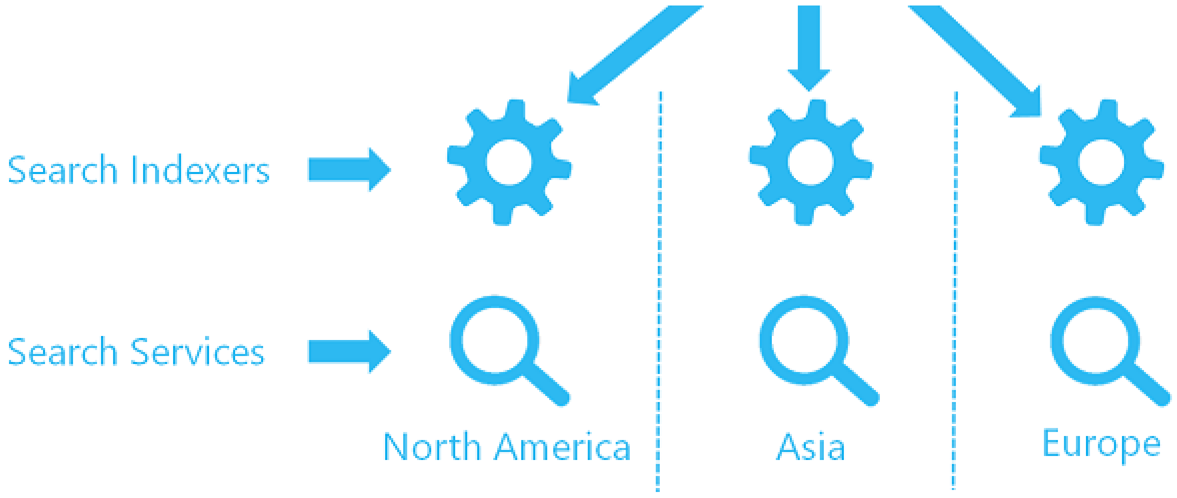
There are two options for keeping your distributed search services in sync which consist of either using the Azure Search Indexer or the Push API (also referred to as the Azure Search REST API).

Use indexers for updating content on multiple services

If you are already using indexer on one service, you can configure a second indexer on a second service to use the same data source object, pulling data from the same location. Each service in each region has its own indexer and a target index (your search index is not shared, which means data is duplicated), but each indexer references the same data source.

Here is a high-level visual of what that architecture would look like.



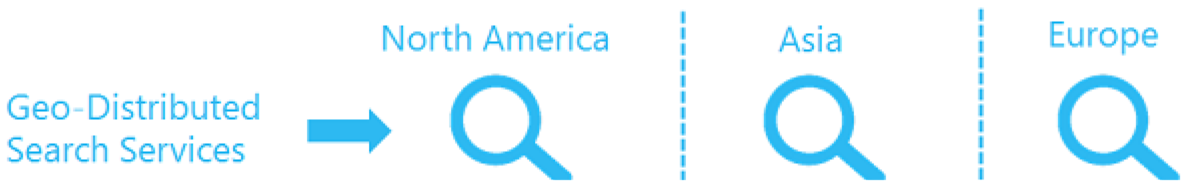


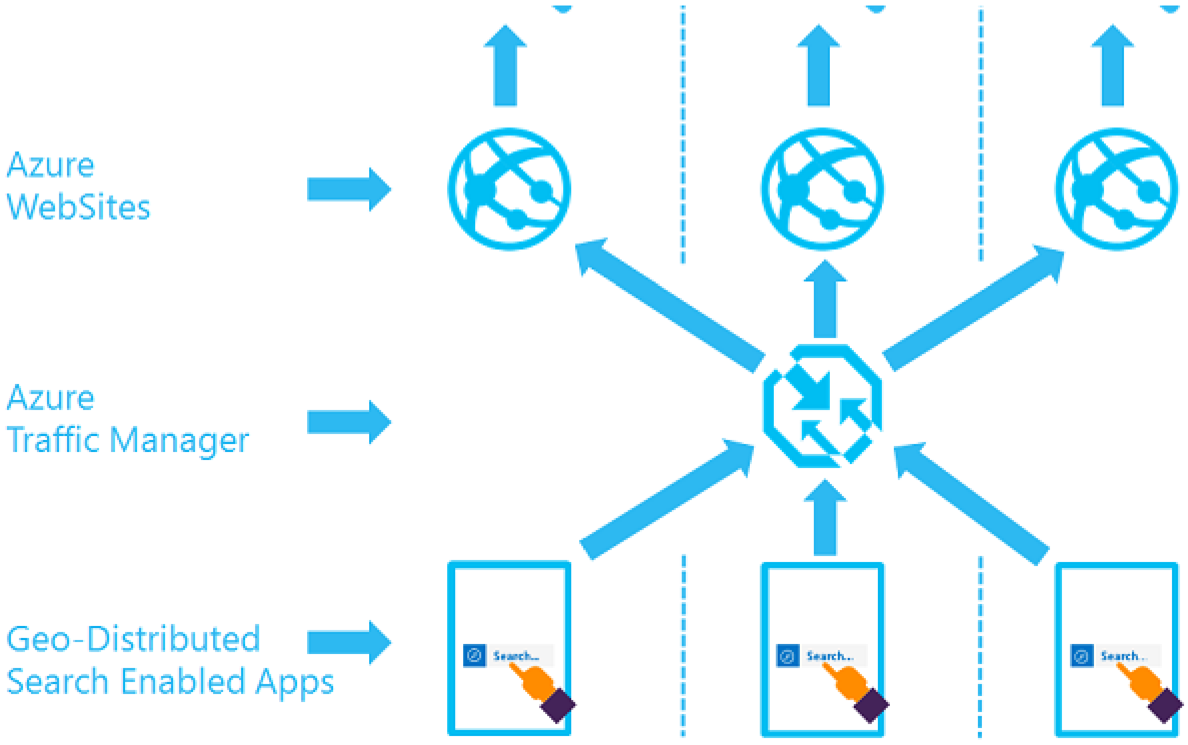
Use REST APIs for pushing content updates on multiple services

If you are using the Azure Search REST API to push content in your Azure Search index, you can keep your various search services in sync by pushing changes to all search services whenever an update is required. In your code, make sure to handle cases where an update to one search service fails but fails for other search services.

Leverage Azure Traffic Manager

Azure Traffic Manager allows you to route requests to multiple geo-located websites that are then backed by multiple Azure Search Services. One advantage of the Traffic Manager is that it can probe Azure Search to ensure that it is available and route users to alternate search services in the event of downtime. In addition, if you are routing search requests through Azure Web Sites, Azure Traffic Manager allows you to load balance cases where the Website is up but not Azure Search. Here is an example of what the architecture that leverages Traffic Manager.





Monitor performance

Azure Search offers the ability to analyze and monitor the performance of your service through search traffic analytics. When you enable this functionality and add instrumentation to your client app, you can optionally log the individual search operations as well as aggregated metrics to an Azure Storage account that can then be processed for analysis or visualized in Power BI. Metrics captures this way provide performance statistics such as average number of queries or query response times. In addition, the operation logging allows you to drill into details of specific search operations.

Traffic analytics is useful for understanding latency rates from that Azure Search perspective. Since the query performance metrics logged are based on the time a query takes to be fully processed in Azure Search (from the time it is requested to when it is sent out), you are able to use this to determine if latency issues are from the Azure Search service side or outside of the service, such as from network latency.

Next steps

To learn more about the pricing tiers and services limits for each one, see Service limits in Azure Search.

Visit Capacity planning to learn more about partition and replica combinations.

For more drilldown on performance and to see some demonstrations of how to implement the optimizations discussed in this article, watch the following video:

34:25