Introduction

2 minutes

Even small organizations can operate globally when they host their apps on Azure. To serve users around the globe, you must design an app architecture that responds rapidly to users, whatever their location.

Suppose you work for a shipping company that has a portal on Azure for customers to create, manage, and track shipments. This portal is a critical part of their business, and has a highly scalable architecture that you implemented in the East US Azure region. The portal needs to be highly available and tolerant of faults across geographic regions. The company wants to create an architecture that will provide these capabilities and identify the services for the features they need.

In this module, you'll learn how to design an application architecture for high availability and fault tolerance that's present in more than one region.

By the end of this module, you'll be able to design this kind of application architecture and understand the benefits, and limitations of its component parts.

Learning

In this module, you'll:

- Design the networking architecture for a geographically distributed application.
- Design the application architecture for a geographically distributed application.
- Design the data architecture for a geographically distributed application.

Prerequisite

- Intermediate familiarity with Azure PaaS services.
- Intermediate familiarity with architecture principles.

Next unit: Design a geographically distributed

Continue

200 XP

Design a geographically distributed architecture

10 minutes

Azure is a global system. By designing an architecture that is present in more than one Azure region, we can build an application that is resilient to even region-wide disasters.

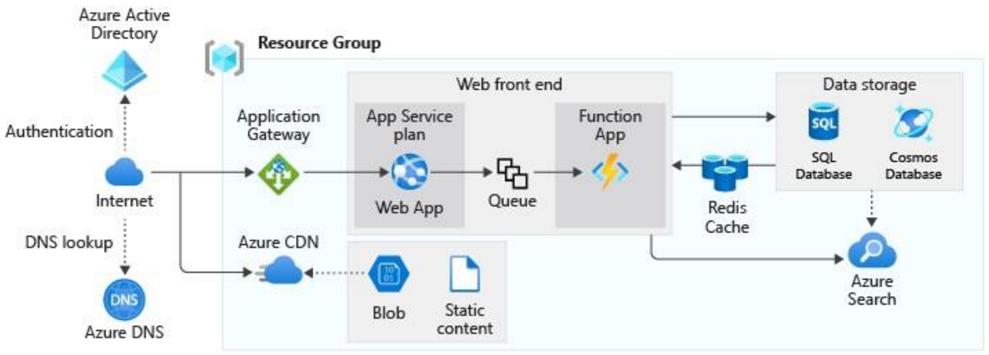
Your shipment tracking portal is scalable because it's built using a range of Azure resources that can scale. It's also resilient too many failures because its components can have multiple instances. However, your board of directors has become concerned a large-scale disaster could cause an interruption because the portal is entirely contained in the East US Azure region. You want to propose a modified architecture that can fail over to a second region if East US fails.

Here, we'll learn how to adjust our application's architecture to support a geographically distributed application. We'll also see why such architecture is advantageous for business-critical applications.

Original web app architecture

Let's have a look at how the tracking portal's architectural design and the components used in the solution. Once we understand how we use all the parts, we can investigate how to support each of these components in geo-redundancy scenarios.

The tracking portal's design is based on the reference architecture shown in this diagram:



Notice how our application makes use of a single Azure resource group. This resource group allows us to group and manage all of our resources logically and simplifies management. We chose to deploy this resource group to the East US region. Even though the resource group doesn't limit us to use the same Azure region for the included resources, we've decided to use the East US region for all resources deployed in our application.

Our application uses three categories of Azure resources. Let's have a look at each category and see which resources are in use.

Network components

The tracking portal uses the following networking services:

Service	Description
Azure DNS	We've configured Azure DNS to host our DNS records in Azure. Azure DNS allows us to manage our DNS records using our Azure credentials in the Azure portal.
Application Gateway	We've configured the Application Gateway load balancer to balance traffic between multiple instances of the web front end. This service is localized to one Azure region.
Azure CDN	We've configured Azure CDN to maximize the delivery of unsecured static content, such as graphics for our website's content. This global service caches static content at points of presence all around the world.

Application components

The tracking portal uses the following services to support code, cache, and intermediate storage requirements.

Service	Description

Service	Description
Azure Active Directory	Users access the tracking portal using Azure AD accounts. The directory and account are automatically replicated globally.
Azure App Service	The tracking portal uses two Azure App Services. The first runs a set of dynamic web pages and the second a web API.
Azure Function Apps	The tracking portal uses Azure Function Apps to run all background tasks. Some of these tasks run on a regular schedule, and other tasks operate on the messages in a queue.
Azure Storage Queues	The tracking portal uses Azure Storage Queues in conjunction with Azure Function Apps. The tracking portal places generated messages onto the queue from where the Function apps process these messages.
Redis cache	The tracking portal uses a Redis cache between the front-end app service and the data storage systems to maximize the performance of queries.
Azure Blob Storage	Static content, such as graphics and video files, are kept as Binary Large Objects (Blobs) in an Azure Storage account and are delivered through the Azure CDN.
Azure Search	We've enabled Azure Search on the tracking portal. Azure Search allows us to make all content searchable, and provide search suggestions and fuzzy search results to our users.

Data storage components

The tracking portal uses the following persisted storage services:

Service	Description
Azure SQL Database.	We're storing relational data, such as order and customer details in Azure SQL Database.
Cosmos DB.	We're storing semi-structured data, including our product catalog in Cosmos DB.

Issues with the original architecture

The existing architecture for the tracking portal is designed to allow for scalability and availability.

For example, if demand is high and responses to user web requests are slow, you can consider adding more instances of the front-end web app in the App Service. Here, the Application Gateway can route requests to these newly created instances.

However, there are scenarios in which our architectural design will have challenges to overcome or even fail. Let's have a look at each scenario to get a better understanding of the impact on the tracking portal.

Regional failures

Some significant events have the potential to interrupt an entire Azure region. Azure datacenters are designed to be highly resilient, but a massive weather event such as a hurricane or flood can disrupt service from the region.

These events are unusual occurrences, and many companies feel they can sustain that risk. However, the consequence of a regional failure disabling the tracking portal is of such a high-risk the company's executive team has decided to eliminate the risk.

Service Level Agreements

Most Azure services offer a Service Level Agreement (SLA) or a guarantee of uptime. When we design an application architecture consisting of multiple Azure services, we calculate the overall SLA for the app as a composite of all other services SLAs.

You calculate this SLA by multiplying together the SLAs of the component services. For example, assume our app consists of Azure App Service (99.95% SLA) and Azure Active Directory (99.9% SLA). The final calculated SLA is 99.85%.

If this percentage uptime isn't enough for our application, we can arrange for the application to fail over onto another region.

Global, regional, and configurable components

In our design, some components are global by default and not vulnerable to a regional failure.

Some components are confined to a single region, for example, the Application Gateway. We'll have to select an alternate service for these types of components

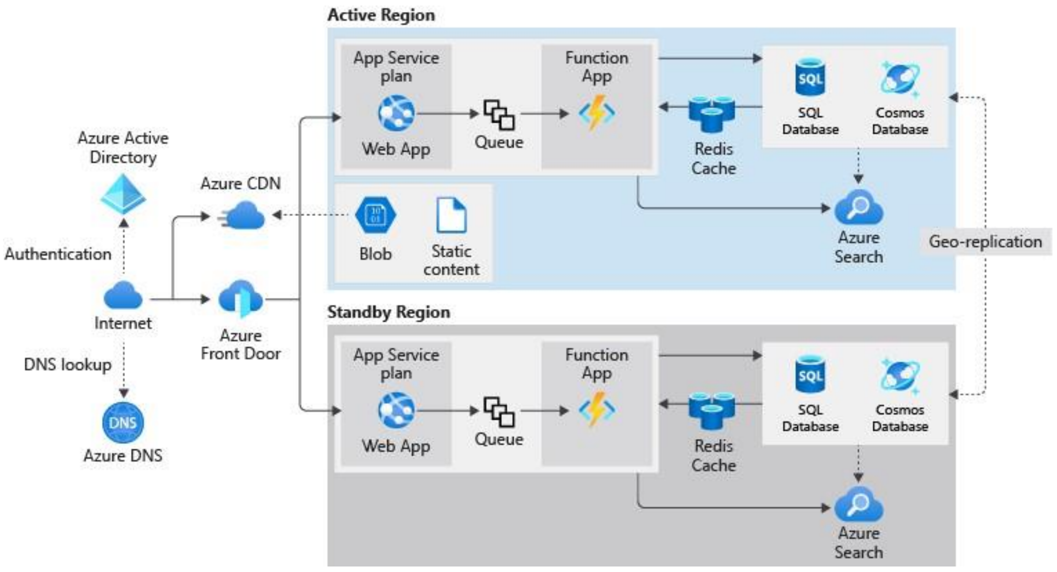
Some components can be configured to support multiple regions. For example, we can use the Geo-Redundant Storage (GRS) option in the Azure Storage account that stores static content. GRS replicates blobs to another region.

This table shows which components are global, regional, and configurable:

Component	Support for multiple regions	Comments
Azure DNS	Global	No changes are necessary.
Application Gateway	Regional	Each instance of Application Gateway is located in a single region.
Azure CDN	Global	No changes are necessary, and content is cached globally by default.
Azure Active Directory	Global	No changes are necessary.
Azure App Service	Regional	Each instance of the app is located in a single region.
Azure Function Apps	Regional	Each instance of the function app is located in a single region.
Azure Storage Queues	Configurable	You can choose to replicate a Storage Account to multiple regions.
Azure Redis Cache	Regional	Each instance of the cache is located in a single region.
Azure Blob Storage	Configurable	You can choose to replicate a Storage Account to multiple regions.
Azure Search	Regional	Each instance of the search service is located in a single region.
Azure SQL Database	Configurable	You can use geo-replication to synchronize data to multiple regions
Azure Cosmos DB	Configurable	You can use geo-replication to synchronize data to multiple regions

Proposed distributed architecture

After some investigation, you propose the architecture in this diagram:



In this design, there's an active region (East US) and a standby region (West US). The East US region handles all requests by the components under ordinary circumstances. If a disaster causes a regional failure, the application fails over onto the West US region.

Let's examine, at a high level, how you've modified the original architecture. We'll explore these changes in more detail in later units.

Networking

Azure DNS and Azure CDN are by default global systems and already resilient to regional failures. We'll leave them in place.

When we create an Azure Application Gateway, we assign the service to a single region. We'll remove this vulnerability by replacing this service with Azure Front Door. Front Door can poll multiple App Services and will handle the App Service failover from the East US region to the West US region.

Application Services

Azure AD is a global system and needs no modification.

Azure Storage accounts can be configured to replicate content to multiple regions. We'll use one of the geo-redundant storage options to change our configuration.

The other components, which include the App Service, Function Apps, the Redis cache, and Azure Search, are regional. We'll create duplicate instances of these components in the West US region in our new architectural design. In this design, the new region can take over when a failover occurs.

Data Storage

Azure SQL Database and Azure Cosmos DB both support geo-replication of data to other regionsWe'll configure these services to replicate East US data to the equivalent services in West US.

Regional Pairs

An Azure region is an area with a single geography that contains one or more Azure datacenters. All regions pair with other regions in the same geography. Within these pairs, updates and planned maintenance are done on only one region at a time. If there's a failure that affects multiple regions, at least one region in each pair is prioritized for rapid recovery.

The best practice is to place a two-region architecture for your app on the two regions in a regional pair. As an example, East US pairs with West US. Our proposed design uses East US for its active region and West US for its standby region.

Check your knowledge

1. Complete this sentence: The composite SLA uptime for an application built using Azure services is calculated...

- By adding together the individual SLAs and dividing them by the number of services.
- By considering the lowest SLA for any of the services being used.
- [✓ By multiplying together the individual SLAs for any of the services being used.]

  A simple example would be an app consisting of Azure App Service (99.95% SLA) and Azure Active Directory (99.9% SLA); the resultant SLA will be 99.85%.

2. You're modifying an application architecture that uses Azure DNS to resolve host names to IP addresses. You want the new architecture to support failover to a standby region. What should you do with Azure DNS?

- [✓ Make no changes to Azure DNS.]

  Azure DNS records are replicated globally by default.
- Configure Azure DNS to replicate its records to the standby region.
- Replace Azure DNS with Azure Traffic Manager.

Next unit: Design a geographically distributed networking architecture

Continue

200 XP

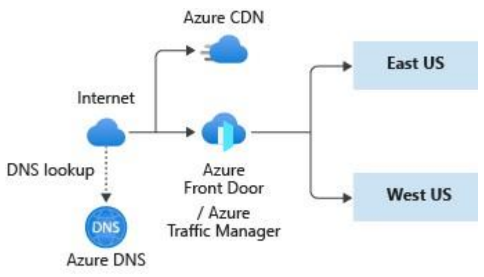
Design a geographically distributed networking architecture

6 minutes

In a distributed app, it's essential to ensure that components can communicate reliably, and requests can route to a different component or region when there's a failure.

We've decided to rearchitect our shipping portal in Azure to reduce its vulnerability to regional failure. We want to ensure, when the primary region is unavailable, the application fails over onto components in a secondary region with minimal disruption in service delivery to users.

Here, we'll learn how Azure DNS, Traffic Manager, Front Door, and Azure CDN support our shipping company's app architecture.



Azure DNS

Recall from earlier that we don't need any changes for our Azure DNS implementation. We use Azure DNS to host the domain name records that that identifies our app.

Azure DNS provides name resolution entirely through the Azure infrastructure. This service is inherently multi-regional, and that's why there's no need to modify our existing Azure DNS configuration to support the feature in our new architectural design.

The Azure DNS SLA also has a 100% guarantee that valid DNS requests will receive a response from at least one Azure DNS name server all the time.

Choose a traffic router

We need a service that can load balance and redirect traffic across multiple regions with distributed web applications.

Azure provides several different services that can route traffic between front-end components. Recall that we need to replace our Azure Application Gateway as it's single region bound. If that region fails, there's nothing to do the routing.

There are two traffic routers in Azure that can do global routing between multiple regions and aren't vulnerable to a single region outage:

- Azure Traffic Manager
- Azure Front Door

Let's examine these services in more detail so that we can choose the right router for our application.

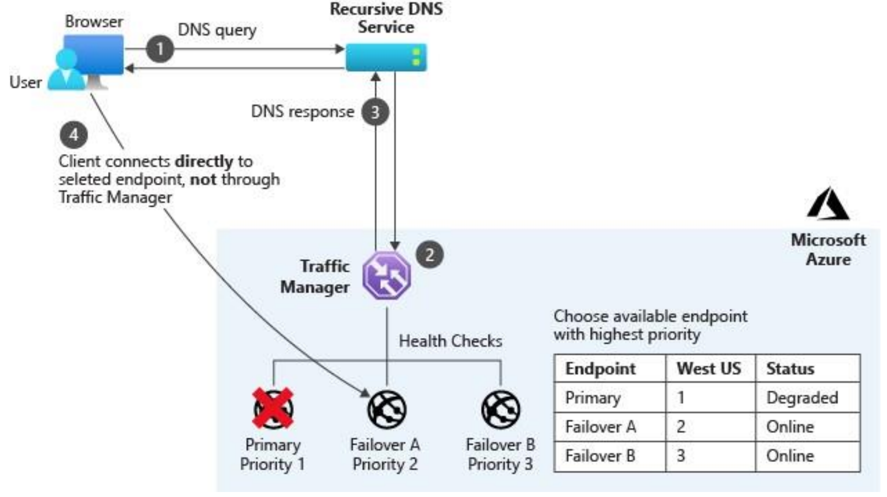
What is Azure Traffic Manager?

Azure Traffic Manager is a global load balancer that uses DNS records to route traffic to destinations in multiple Azure regions.

We can configure Traffic Manager to route all requests to our primary region and monitor the responsiveness of the App Service in that region. If the App Service in the primary region fails, Traffic Manager automatically reroutes user requests to the App Service in the secondary region. This reroute executes the failover that ensures continuous service. We call this arrangement th**priority routing mode**

Because Traffic Manager uses the DNS system to route traffic, it routes any protocol, not just HTTP traffic. However, Traffic Manager can't route or filter traffic based on HTTP properties, such as client country codes or user-agent headers. It also can't do Transport Layer Security (TLS) protocol termination, where the router decrypts requests and encrypts responses to take that load off the App Service virtual servers. If we need either of these features, we'll have to use Azure Front Door.

Traffic Manager uses highly configurable endpoint monitoring. For example, we can define the protocol, port, path, custom header settings, expected status code ranges, tolerated number of failures, and so on. Endpoint monitoring gives us a continuous idea of the overall health of all parts of our application.



Endpoint	West US	Status
Primary	1	Degraded
Failover A	2	Online
Failover B	3	Online

What is Azure Front Door?

Like Traffic Manager, Azure Front Door is a global load balancer. Unlike Traffic Manager, it works at the network application layer, Layer 7, and uses HTTP and HTTPS properties to do filtering and routing.

With Front Door, we can do many types of routing that Traffic Manager doesn't support. For example, we can route traffic based on the browser's country code. Front Door also supports TLS protocol termination.

There is, however, an exception. If we want to route traffic for any protocol other than HTTP and HTTPS, we'll have to use Traffic Manager.

Front Door allows us to assign priorities to the various backends that make up the tracking portal. These priorities allow Front Door to route requests as needed. We'll assign our primary region services with a top priority and our secondary region service with a lower priority.

Front Door implements health probes to monitor the health status of our services, and if there's a failure it can route traffic correctly. The priority routing mode and endpoint monitoring in Front Door is similar to those features in Traffic Manager, except that health probes always work over HTTP.

All the traffic for our shipping portal's web UI and its APIs are done over HTTPS and allows us to switch out Azure Traffic Manager with Front Door. We'll also configure Front Door with priority backend assignment.

Azure CDN

In our single-region architecture, we used Azure CDN to cache static content from Azure Blob Storage. The Azure CDN service is a global network of servers that caches static content close to users. We don't need to modify this service for the multi-region architectureHowever, there are considerations with regards to our Azure Storage account that we'll cover in the next unit.

Check your knowledge

1. When should you perform a full failover to another region?

- As soon as the application displays an error.
- If you experience notable slowness or a lack of responsiveness
- [✓ When you know definitively, that a region is down.]

  First there may be isolated reports from different sources, but they combine together and grow in number and volume until regional failure is the only explanation. Additionally, the Azure team will update the Azure status page with this information.

2. What is the SLA for Azure DNS?

- [✓ 100%]

  Azure DNS guarantees that valid DNS requests will receive a response from at least one name server 100% of the time.
- 99.95%
- 99.85%

Next unit: Design a geographically distributed application architecture

200 XP

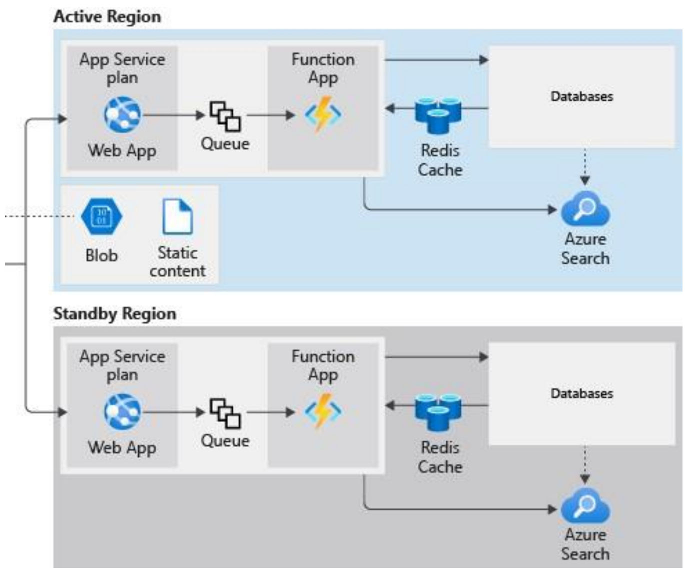
Design a geographically distributed application architecture

8 minutes

When our networking components route requests to multiple regions to mitigate the effects of a regional outage, we must design application services that can respond to those requests in both primary and standby regions.

Recall from earlier that we'll configure Azure Front Door with priority backend assignment. We'll assign the East US region as our primary region, and the West US region as our standby region. When a regional failure occurs, requests will route to the App Service in the none failing region. We have to configure resources in each region to support these failovers for user access, replicated storage, and application code.

Here, we'll learn about Active Directory, static content storage, web apps, web APIs, queues, Azure functions, and data caches in a multi-region architecture.



Azure Active Directory

In our shipments tracking portal, users can track the delivery of their purchases by entering a tracking number. However, regular users can register for membership to access advanced features, such as delivery promptness and other statistics. We've developed the tracking portal to store user accounts in Azure Active Directory (AD).

Azure AD is designed as a global system by default. As such, it's not vulnerable to regional failures, and we don't have to modify this component of the system.

Azure Blob Storage

Static content, such as images and videos, are stored in Azure Storage accounts as Binary Large Objects (Blobs) and served to users through the Azure CDN.

In our original design, the storage account is contained in a single region because we chose to use Locally Redundant Storage (LRS). Our data is replicated only within a single datacenter with LRS. The storage account, therefore, is unavailable if there's a regional outage in this configuration. Any static content that has already been cached by the CDN remains available to users.

The same is true of Zone Redundant Storage (ZRS). Even though data replicates to different data centers in this configuration, all these data centers are still in the same region. A regional outage will also affect the storage account in this configuration.

In our design, we rely heavily on our CDN configuration to cache static content. There's a chance that, during an outage, a user might request a static file that isn't yet in the CDN cache. This request would result in a graphic or video that can't be displayed.

We can eliminate this possibility by replicating the storage account to multiple regions when we choose a geo-redundant storage option. We also have to option to select a read-only replication option if we want to support the inability to add static content during a regional outage.

We have two options to choose from when we need to enable geo-redundancy. These options are Read-Access Geo-Redundant Storage (RA-GRS) and Read-Acces Geo-Zone-Redundant Storage (RA-GZRS). The choice we make will depend on our budget and the percentage up time that we need.

Azure App Service and Azure Function Apps

Our shipments tracking portal implements two Azure App Services. The first App Services hosts a web app that implements the user-facing web interface, and the second hosts a web API used by mobile apps to track shipments data. All of our background tasks run as Azure Function apps.

In our original design, each Azure App Service is localized to a single Azure region. We'll create a second App Service in the secondary region (West US) and deploy the web project there to support the new multi-region architecture. We'll configure the Azure Front Door priority routing mode to send requests to our secondary region when the primary region is unavailable.

To ensure the failover is as smooth as possible, make sure the web application doesn't store any session state information in memory. We'll change our website to make sure we don't end up with data loss. For example, if our code stores a list of the users' shipments in memory, then this list would be lost if a failover occurred.

Each web request is handled without impacting the other when no session state is stored. If a failover occurs in the middle of a user's session, the failover shoul be transparent to the user.

We'll make a similar change to our Azure Function apps. We'll create a separate instance of the Azure Function in the secondary region and deploy the same custom code to it as runs in the primary region.

Important

When you deploy an update to the custom code in the App Service or Function App service, remember to distribute it to all the instances of the App Service. If you want to automate this process, Azure DevOps has tools that can help.

Azure Storage Queues

In our original single-region architecture, we used a queue in an Azure Storage account to manage communications between the App Service and the function app. When the web app or the web API needs to run a background task, it places a message with all the required information in the queue. The function app monitors the queue for new messages and executes the background task by running the necessary queries against the data stores.

We can manage a high demand in web requests in an orderly way when we use a queue in this way.When there are many background tasks to run, the queue may build up, but tasks will not be dropped by and stay in the queue until they're processed. The function apps work through the queue and reduce its size when demand falls. If demand persists, we'll increase the number of instances of the function app.

For the multi-region version of the shipments tracking portal, we must make sure that queue items aren't lost when failover occurs. Our queue is defined in Azure Storage, and we can use a redundancy option for geo-replication.

Keep in mind that we can't use a read-access redundancy option since our queue supports read and write operations. The App Service must add items to the queue, and the function app must remove completed items from the queue. Use Geo-Redundant Storage (GRS) or Geo-Zone-Redundant Storage (GZRS) instead.

Azure Redis Cache

We're using Azure Redis Cache to maximize the performance of data storage. Redis caches all query results generated from our apps as they request data from our database. The following queries for similar data don't need a database query and are fetched from the Redis cache.

For the multi-region architecture, we'll create a Redis Cache instance in both primary and standby regions. Keep in mind that when a failover occurs, the Redis Cache in the standby region is likely to be empty. That empty cache won't cause any errors, but performance may temporarily drop, as data fills the new cache.

Check your knowledge

1. Which components of the shipping company architecture should be explicitly copied to another region?

- Azure DNS and Azure AD
- [✓ Azure App Service, Azure Function App, Redis Cache, queue]

  These components should be explicitly copied or replicated in another region.
- Azure DNS, Azure AD, Azure Storage

2. Complete this sentence: If a regional failure takes out Azure Storage, data loss...

- Won't occur because all data is automatically copied to a secondary region.
- [✓ May briefly occur because data is copied asynchronously.]

  With RA-GRS, data is copied asynchronously from primary to secondary so data loss is a possibility.

200 XP

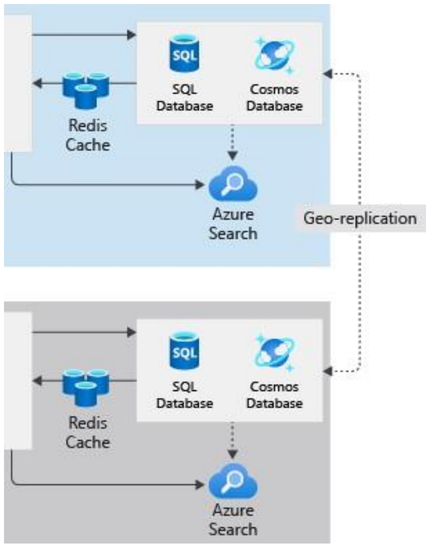
Design a geographically distributed data architecture

6 minutes

The final part of our application's architectural design to consider is the data storage tier. We want to make sure that data is both readable and writable with full functionality after a region-wide failure.

In the shipment tracking portal, we chose to use Azure Front Door to send all requests to App Services in East US. If East US fails, Front Door detects the failure and sends requests to duplicate App Services components in West US. In our original, single-region architecture, we stored relational data in Azure SQL Database and semi-structured data in Cosmos DB. Now, we want to understand how we can ensure that both databases remain available if East US fails.

Here, we'll learn how to replicate data between regions and how to ensure that failover can occur quickly if necessary.



Azure SQL Database

To create a multi-region implementation of Azure SQL Database to store relational data, we can use either:

- **Active geo-replication**
- **Auto failover groups**

Let's examine these options in more detail:

Active geo-replication

Azure SQL Database can automatically replicate a database and all its changes from one database to another with the active geo-replication feature. Only the primary logical server hosts a writable copy of the database. We can create up to four other logical servers that host read-only copies of the database.

For the shipment tracking portal, create a secondary database in West US and configure geo-replication from East US. When a regional failure occurs, Front Door redirects user requests to App Services in West US. App Services and Azure Functions can get access the relational data because a copy has already been replicated to West US.

This change is automatic, but remember that the secondary database in West US is read-only. If a user tries to modify data, for example, by creating a new shipment, errors may arise. We can manually initiate a failover to West US as soon as we notice the problem in that Azure portal. If we want to automate this process, our developers can write code that calls the `failover` method in the Azure SQL Database REST API.

Note

Managed instances of Azure SQL Database do not support Active geo-replication. Managed instances are designed to make it simple to migrate data from an on-premises SQL Server while maintaining security. If we're using a managed instance, consider using failover groups instead.

Auto failover groups

An auto failover group is a group of databases where data replicates automatically from a primary to one or more secondary servers. This design is like active geo-replication and uses the same data replication method. However, we can automate the response to a failure by defining a policy.

For the shipping portal, we'll create a secondary database in West US. We'll then add a policy that fails over the primary replica of the database to West US if a catastrophic failure occurs in East US. If that happens, the West US replica automatically becomes the writable primary database, and full functionality is maintained.

Consider using an auto failover group if you want to automate the failover of the writeable database without writing custom code to trigger it. Also, use auto failover groups if our database runs in a managed instance of Azure SQL Database.

Important

The replication that underlies both active geo-replication and auto-failover groups are asynchronous. An acknowledgment is sent to the client when a change is applied to the primary replica. At this point, the transaction is considered complete, and replication occurs. If a failure occurs, the latest changes made in the primary database may not have replicated to the secondary. Keep in mind that, after a disaster, the most recent database changes may have been lost.

Azure Cosmos DB

Our configuration is less complex with Azure Cosmos DBCosmos DB is a multi-model database, able to store relational data, semi-structured data, and other forms of data. That's because Cosmos DB was designed as a multi-regional cloud database system. Even if we run Cosmos DB in a single region, data is replicated to multiple instances across different fault-domains for the best availability.

When we create a Cosmos DB account multi-region, we can choose from the following modes:

- **Multi-region accounts with multiple write regions.**

  In this mode, all copies of the database are writable at all times. If a region fails, no failover is necessary.

- **Multi-region accounts with a single write region.**

  In this mode, only the primary region contains writable databases. The data replicated to the secondary regions are read-only. Updates are disabled by default when the primary region fails. However, we can selec**enable automatic failover**so that Cosmos DB automatically fails over the primary, writable copy of the database to another region.

Important

In Cosmos DB, data replication is synchronous. When a change is applied, the transaction is not considered complete until replicated to a quorum of replicas. Then an acknowledgment is sent to the client. When a failure occurs, no recent changes are lost because replication has already occurred.

Check your knowledge

1. In the shipment tracking application, you want to automatically fail over write access to the SQL database, when there is a regional outage. You don't want to write custom code. What should you do?

- Use active geo-replication
- [✓ Use an auto-failover group]

  By using an auto-failover group, you can configure the primary database to fail over onto the replica in the secondary region automatically.
- Use multiple write regions

2. You want to ensure that no completed transactions are lost in a regional outage. What should you do?

- [✓ Use Azure Cosmos DB with a single write region.]

  Azure Cosmos DB using synchronous replication, in which a transaction is only complete when it has replicated to a quorum of replicas.
- Use Azure SQL Database with active geo-replication.
- Use Azure SQL Database with an auto-failover group.