Resiliency checklist for specific Azure services

11/26/2018 • 14 minutes to read • Contributors all

In this article

Resiliency is the ability of a system to recover from failures and continue to function. Every technology has its own particular failure modes, which you must consider when designing and implementing your application. Use this checklist to review the resiliency considerations for specific Azure services. For more information about designing resilient applications, see Design reliable Azure applications.

App Service

Use Standard or Premium tier. These tiers support staging slots and automated backups. For more information, see Azure App Service plans in-depth overview

Avoid scaling up or down. Instead, select a tier and instance size that meet your performance requirements under typical load, and then scale out the instances to handle changes in traffic volume. Scaling up and down may trigger an application restart.

Store configuration as app settings. Use app settings to hold configuration settings as app settings. Define the settings in your Resource Manager templates, or using PowerShell, so that you can apply them as part of an automated deployment / update process, which is more reliable. For more information, see Configure web apps in Azure App Service.

Create separate App Service plans for production and test. Don't use slots on your production deployment for testing. All apps within the same App Service plan share the same VM instances. If you put production and test deployments in the same plan, it can negatively affect the production deployment. For example, load tests might degrade the live production site. By putting test deployments into a separate plan, you isolate them from the production version.

Separate web apps from web APIs. If your solution has both a web front end and a web API, consider decomposing them into separate App Service apps. This design makes it easier to decompose the solution by workload. You can run the web app and the API in separate App Service plans, so they can be scaled independently. If you don't need that level of scalability at first, you can deploy the apps into the same plan, and move them into separate plans later, if needed.

Avoid using the App Service backup feature to back up Azure SQL databases. Instead, use SQL Database automated backups. App Service backup exports the database to a SQL BACPAC file, which costs DTUs.

Deploy to a staging slot. Create a deployment slot for staging. Deploy application updates to the staging slot, and verify the deployment before swapping it into production. This reduces the chance of a bad update in production. It also ensures that all instances are warmed up before being swapped into production. Many applications have a significant warmup and cold-start time. For more information, see Set up staging environments for web apps in Azure App Service.

Create a deployment slot to hold the last-known-good (LKG) deployment. When you deploy an update to production, move the previous production deployment into the LKG slot. This makes it easier to roll back a bad deployment. If you discover a problem later, you can quickly revert to the LKG version. For more information, see Basic web application.

Enable diagnostics logging, including application logging and web server logging. Logging is important for monitoring and diagnostics. See Enable diagnostics logging for web apps in Azure App Service

Log to blob storage. This makes it easier to collect and analyze the data.

Create a separate storage account for logs. Don't use the same storage account for logs and application data. This helps to prevent logging from reducing application performance.

Monitor performance. Use a performance monitoring service such as New Relic or Application Insights to monitor application performance and behavior under load. Performance monitoring gives you real-time insight into the application. It enables you to diagnose issues and perform root-cause analysis of failures.

Application Gateway

Provision at least two instances. Deploy Application Gateway with at least two instances. A single instance is a single point of failure. Use two or more instances for redundancy and scalability. In order to qualify for the SLA, you must provision two or more medium or larger instances.

Cosmos DB

Replicate the database across regions. Cosmos DB allows you to associate any number of Azure regions with a Cosmos DB database account. A Cosmos DB database can have one write region and multiple read regions. If there is a failure in the write region, you can read from another replica. The Client SDK handles this automatically. You can also fail over the write region to another region. For more information, see How to distribute data globally with Azure Cosmos DB.

Event Hubs

Use checkpoints. An event consumer should write its current position to persistent storage at some predefined interval. That way, if the consumer experiences a fault (for example, the consumer crashes, or the host fails), then a new instance can resume reading the stream from the last recorded position. For more information, see Event consumers.

Handle duplicate messages. If an event consumer fails, message processing is resumed from the last recorded checkpoint. Any messages that were already processed after the last checkpoint will be processed again. Therefore, your message processing logic must be idempotent, or the application must be able to deduplicate messages.

Handle exceptions.. An event consumer typically processes a batch of messages in a loop. You should handle exceptions within this processing loop to avoid losing an entire batch of messages if a single message causes an exception.

Use a dead-letter queue. If processing a message results in a non-transient failure, put the message onto a dead-letter queue, so that you can track the status. Depending on the scenario, you might retry the message later, apply a compensating transaction, or take some other action. Note that Event Hubs does not have any built-in dead-letter queue functionality. You can use Azure Queue Storage or Service Bus to implement a dead-letter queue, or use Azure Functions or some other eventing mechanism.

Implement disaster recovery by failing over to a secondary Event Hubs namespace. For more information, see Azure Event Hubs Geo-disaster recovery.

Redis Cache

Configure Geo-replication. Geo-replication provides a mechanism for linking two Premium tier Azure Redis Cache instances. Data written to the primary cache is replicated to a secondary read-only cache. For more information, see How to configure Geo-replication for Azure Redis Cache

Configure data persistence. Redis persistence allows you to persist data stored in Redis. You can also take snapshots and back up the data, which you can load in case of a hardware failure. For more information, see How to configure data persistence for a Premium Azure Redis Cache

If you are using Redis Cache as a temporary data cache and not as a persistent store, these recommendations may not apply.

Search

Provision more than one replica. Use at least two replicas for read high-availability, or three for read-write high-availability.

Configure indexers for multi-region deployments. If you have a multi-region deployment, consider your options for continuity in indexing.

- If the data source is geo-replicated, you should generally point each indexer of each regional Azure Search service to its local data source replica. However, that approach is not recommended for large datasets stored in Azure SQL Database. The reason is that Azure Search cannot perform incremental indexing from secondary SQL Database replicas, only from primary replicas. Instead, point all indexers to the primary replica. After a failover, point the Azure Search indexers at the new primary replica.

- If the data source is not geo-replicated, point multiple indexers at the same data source, so that Azure Search services in multiple regions continuously and independently index from the data source. For more information, see Azure Search performance and optimization considerations.

Service Bus

Use Premium tier for production workloads. Service Bus Premium Messaging provides dedicated and reserved processing resources, and memory capacity to support predictable performance and throughput. Premium Messaging tier also gives you access to new features that are available only to premium customers at first. You can decide the number of messaging units based on expected workloads.

Handle duplicate messages. If a publisher fails immediately after sending a message, or experiences network or system issues, it may erroneously fail to record that the message was delivered, and may send the same message to the system twice. Service Bus can handle this issue by enabling duplicate detection. For more information, see Duplicate detection.

Handle exceptions. Messaging APIs generate exceptions when a user error, configuration error, or other error occurs. The client code (senders and receivers) should handle these exceptions in their code. This is especially important in batch processing, where exception handling can be used to avoid losing an entire batch of messages. For more information, see Service Bus messaging exceptions.

Retry policy. Service Bus allows you to pick the best retry policy for your applications. The default policy is to allow 9 maximum retry attempts, and wait for 30 seconds but this can be further adjusted. For more information, see Retry policy – Service Bus.

Use a dead-letter queue. If a message cannot be processed or delivered to any receiver after multiple retries, it is moved to a dead letter queue. Implement a process to read messages from the dead letter queue, inspect them, and remediate the problem. Depending on the scenario, you might retry the message as-is, make changes and retry, or discard the message. For more information, see Overview of Service Bus dead-letter queues.

Use Geo-Disaster Recovery. Geo-disaster recovery ensures that data processing continues to operate in a different region or datacenter if an entire Azure region or datacenter becomes unavailable due to a disaster. For more information, see Azure Service Bus Geo-disaster recovery.

Storage

For application data, use read-access geo-redundant storage (RA-GRS). RA-GRS storage replicates the data to a secondary region, and provides read-only access from the secondary region. If there is a storage outage in the primary region, the application can read the data from the secondary region. For more information, see Azure Storage replication.

For VM disks, use managed disks. Managed disks provide better reliability for VMs in an availability set, because the disks are sufficiently isolated from each other to avoid single points of failure. Also, managed disks aren't subject to the IOPS limits of VHDs created in a storage account. For more information, see Manage the availability of Windows virtual machines in Azure.

For Queue storage, create a backup queue in another region. For Queue storage, a read-only replica has limited use, because you can't queue or dequeue items. Instead, create a backup queue in a storage account in another region. If there is a storage outage, the application can use the backup queue, until the primary region becomes available again. That way, the application can still process new requests.

SQL Database

Use Standard or Premium tier. These tiers provide a longer point-in-time restore period (35 days). For more information, see SQL Database options and performance.

Enable SQL Database auditing. Auditing can be used to diagnose malicious attacks or human error. For more information, see Get started with SQL database auditing.

Use Active Geo-Replication Use Active Geo-Replication to create a readable secondary in a different region. If your primary database fails, or simply needs to be taken offline, perform a manual failover to the secondary database. Until you fail over, the secondary database remains read-only. For more information, see SQL Database Active Geo-Replication.

Use sharding. Consider using sharding to partition the database horizontally. Sharding can provide fault isolation. For more information, see Scaling out with Azure SQL Database.

Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time. For more information, see Recover an Azure SQL database using automated database backups.

Use geo-restore to recover from a service outage. Geo-restore restores a database from a geo-redundant backup. For more information, see Recover an Azure SQL database using automated database backups.

SQL Data Warehouse

Do not disable geo-backup. By default, SQL Data Warehouse takes a full backup of your data every 24 hours for disaster recovery. It is not recommended to turn this feature off. For more information, see Geo-backups.

SQL Server running in a VM

Replicate the database. Use SQL Server Always On availability groups to replicate the database. Provides high availability if one SQL Server instance fails. For more information, see Run Windows VMs for an N-tier application

Back up the database. If you are already using Azure Backup to back up your VMs, consider using Azure Backup for SQL Server workloads using DPM. With this approach, there is one backup administrator role for the organization and a unified recovery procedure for VMs and SQL Server. Otherwise, use SQL Server Managed Backup to Microsoft Azure.

Traffic Manager

Perform manual failback. After a Traffic Manager failover, perform manual failback, rather than automatically failing back. Before failing back, verify that all application subsystems are healthy. Otherwise, you can create a situation where the application flips back and forth between datacenters. For more information, see Run VMs in multiple regions for high availability.

Create a health probe endpoint. Create a custom endpoint that reports on the overall health of the application. This enables Traffic Manager to fail over if any critical path fails, not just the front end. The endpoint should return an HTTP error code if any critical dependency is unhealthy or unreachable. Don't report errors for non-critical services, however. Otherwise, the health probe might trigger failover when it's not needed, creating false positives. For more information, see Traffic Manager endpoint monitoring and failover.

Virtual Machines

Avoid running a production workload on a single VM. A single VM deployment is not resilient to planned or unplanned maintenance. Instead, put multiple VMs in an availability set or virtual machine scale set, with a load balancer in front.

Specify an availability set when you provision the VM. Currently, there is no way to add a VM to an availability set after the VM is provisioned. When you add a new VM to an existing availability set, make sure to create a NIC for the VM, and add the NIC to the back-end address pool on the load balancer. Otherwise, the load balancer won't route network traffic to that VM.

Put each application tier into a separate Availability Set. In an N-tier application, don't put VMs from different tiers into the same availability set. VMs in an availability set are placed across fault domains (FDs) and update domains (UD). However, to get the redundancy benefit of FDs and UDs, every VM in the availability set must be able to handle the same client requests.

Replicate VMs using Azure Site Recovery. When you replicate Azure VMs using Site Recovery, all the VM disks are continuously replicated to the target region asynchronously. The recovery points are created every few minutes. This gives you a Recovery Point Objective (RPO) in the order of minutes. You can conduct disaster recovery drills as many times as you want, without affecting the production application or the ongoing replication. For more information, see Run a disaster recovery drill to Azure.

Choose the right VM size based on performance requirements. When moving an existing workload to Azure, start with the VM size that's the closest match to your on-premises servers. Then measure the performance of your actual workload with respect to CPU, memory, and disk IOPS, and adjust the size if needed. This helps to ensure the application behaves as expected in a cloud environment. Also, if you need multiple NICs, be aware of the NIC limit for each size.

Use managed disks for VHDs. Managed disks provide better reliability for VMs in an availability set, because the disks are sufficiently isolated from each other to avoid single points of failure. Also, managed disks aren't subject to the IOPS limits of VHDs created in a storage account. For more information, see Manage the availability of Windows virtual machines in Azure.

Install applications on a data disk, not the OS disk. Otherwise, you may reach the disk size limit.

Use Azure Backup to back up VMs. Backups protect against accidental data loss. For more information, see Protect Azure VMs with a Recovery Services vault.

Enable diagnostic logs. Include basic health metrics, infrastructure logs, and boot diagnostics. Boot diagnostics can help you diagnose a boot failure if your VM gets into a non-bootable state. For more information, see Overview of Azure Diagnostic Logs.

Use the AzureLogCollector extension. (Windows VMs only.) This extension aggregates Azure platform logs and uploads them to Azure storage, without the operator remotely logging into the VM. For more information, see AzureLogCollector Extension.

Virtual Network

To whitelist or block public IP addresses, add a network security group to the subnet. Block access from malicious users, or allow access only from users who have privilege to access the application.

Create a custom health probe. Load Balancer Health Probes can test either HTTP or TCP. If a VM runs an HTTP server, the HTTP probe is a better indicator of health status than a TCP probe. For an HTTP probe, use a custom endpoint that reports the overall health of the application, including all critical dependencies. For more information, see Azure Load Balancer overview.

Don't block the health probe. The Load Balancer Health probe is sent from a known IP address, 168.63.129.16. Don't block traffic to or from this IP in any firewall policies or network security group rules. Blocking the health probe would cause the load balancer to remove the VM from rotation.

Enable Load Balancer logging. The logs show how many VMs on the back-end are not receiving network traffic due to failed probe responses. For more information, see Log analytics for Azure Load Balancer.