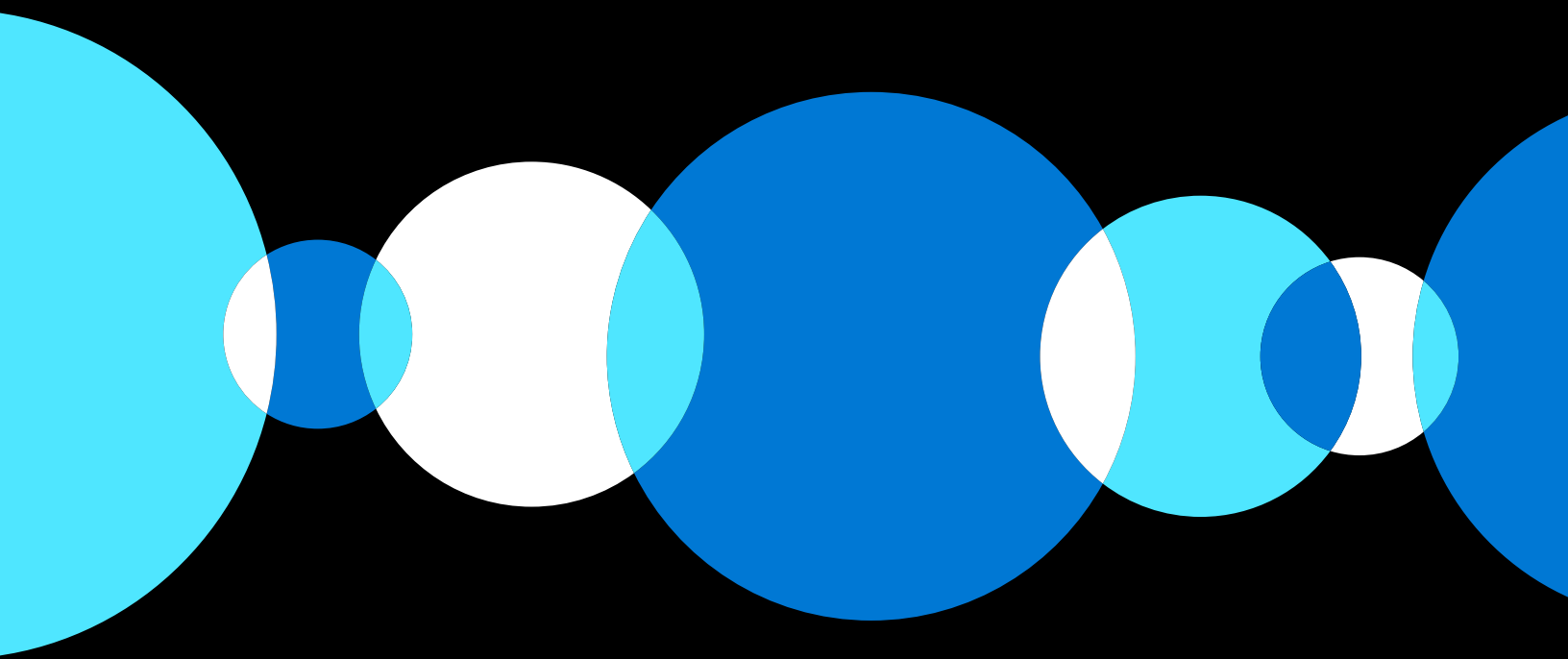
Microsoft

Resilience in Microsoft Azure

Designing resilient infrastructure as a service applications on Azure

August 2019



Abstract

This document provides guidance on designing resilient infrastructure as a service (IaaS) applications on Azure and provides sample application design patterns for varied levels of resilience. The audience for this document is those who move applications from on-premises environments to Azure or who build applications on Azure, such as cloud solution architects, business continuity and disaster recovery administrators, application developers and operations team members, as well as CTOs, CIOs, and those involved in policy, planning, and other strategic roles.

Table of contents

Introduction

Azure is a rapidly growing cloud computing platform that provides an ever-expanding suite of cloud services. These include analytics, computing, database, mobile, networking, storage, and web services. Azure integrates tools, templates, and managed services that work together to help make it easier to build and manage enterprise, mobile, web, and Internet of Things (IoT) apps faster, using the tools, applications, and frameworks that customers choose.

Azure is built on trust. The Azure approach to trust is based on five foundational principles: security, compliance, privacy, resilience, and intellectual property (IP) protection. A well-designed Azure application should focus on these five pillars of software quality.

This document describes a process for achieving resilience using a structured approach throughout the lifetime of an application—from design and implementation to deployment and operations.

Pillar	Description
Scalability	A system's ability to handle increased load.
Availability	The proportion of time that a system is functional and working.
Resilience	A system's ability to recover from failures and continue to function.
Management	Operations processes which keep a system running in production.
Security	To protect applications and data from threats.

Concepts and terminology

Achieving resilience requires understanding how it's measured, designed, and implemented. The concepts and terms in this section lay the foundation for the example resilience design patterns that follow.

Shared responsibility

Being resilient in the event of any failure is a shared responsibility. The responsibility of the customer versus those of the cloud provider, in terms of resilience, depends on the cloud service model being used: infrastructure as a service (IaaS), platform as a service (PaaS), or software as a service (SaaS).

Metrics to measure resilience

The following metrics are used to measure resilience:

Recovery time objective (RTO) is the maximum acceptable time that an application can be unavailable after an incident.

Recovery point objective (RPO) is the maximum duration for which data loss can occur during a disaster.

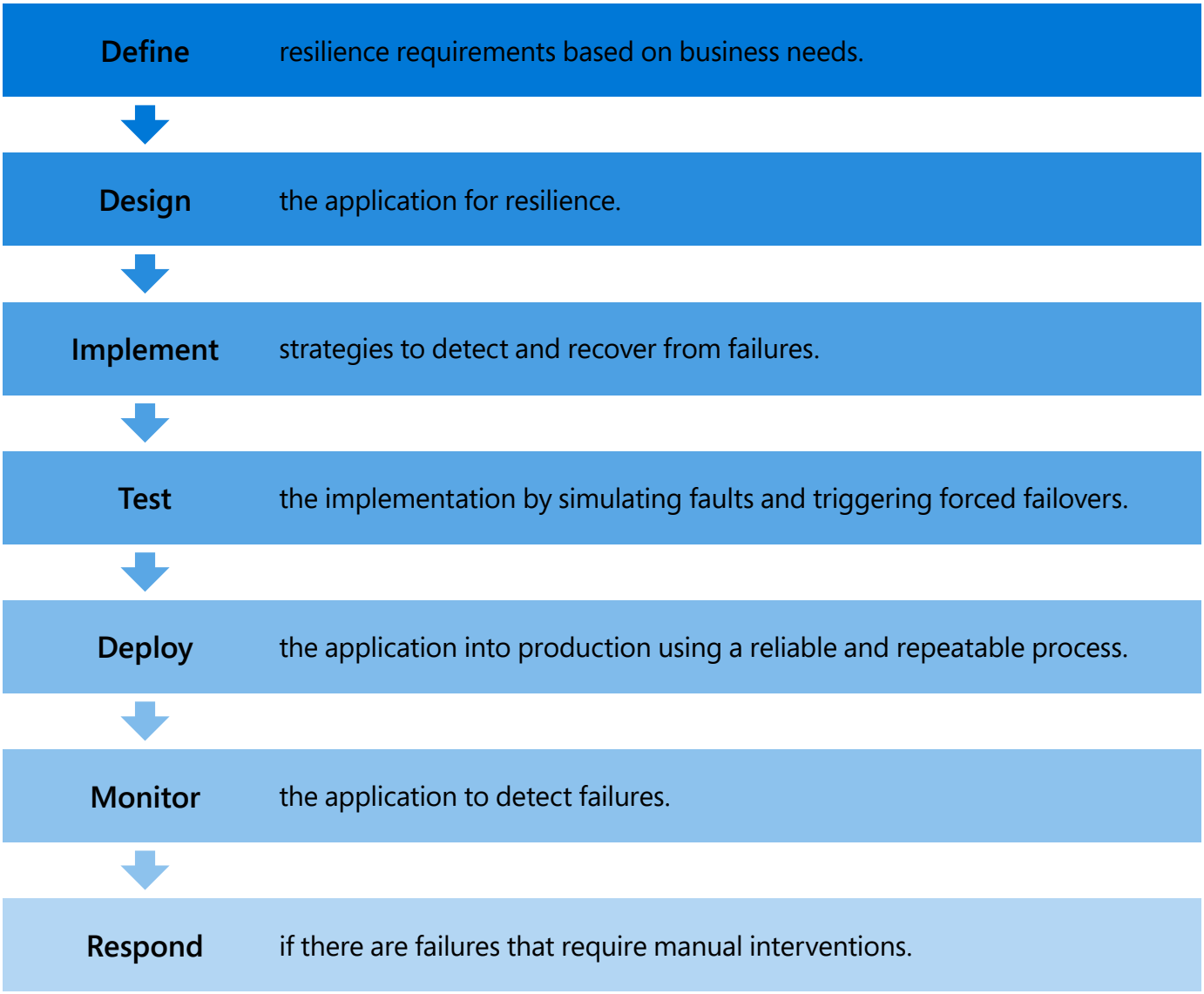
Mean time to recover (MTTR) is the average time that it takes to restore a component after a failure.

Mean time between failures (MTBF) is the runtime that a component can reasonably expect to last between outages.

The service-level agreement (SLA) in Azure describes our commitments regarding uptime and connectivity. If the SLA for a service is 99.9 percent, customers can expect the service to be available 99.9 percent of the time. Dealing with multiple services that have different agreements adds complexity. In this case, it's necessary to calculate a composite SLA.

Resilient design

Failures can vary in the scope of their impact. There are many different failure types, including hardware, datacenter, regional or transient failure, as well as dependency service, heavy load, accidental data deletion/corruption, and application deployment failures. Performing a failure mode analysis (FMA) during the design phase helps identify possible points of failure and defines how the application will respond to those failures.



What's resilience?

Resilience is a system's ability to recover from failures and continue to function. It's not only about *avoiding* failures but also involves *responding* to failures in a way that minimizes downtime or data loss. Because failures can occur at various levels, it's important to have protection for all types based on your application availability requirements.

Foundation

The first step toward achieving resilience is to avoid failures in the first place. Microsoft Azure invests heavily in cloud services to ensure that customers can run their workloads reliably. Our approach to improving Azure reliability involves improving the platform's capability to minimize impact during planned maintenance events and giving customers control over their experience during these events.

Cloud technology at Microsoft has come a long way over the years, adding innovative features such as memory-preserving host updates and live migration. With Azure, overall availability has been trending up constantly and is now at approximately 99.999 percent reliability across the fleet. As a result, more customers are running mission-critical workloads on Azure.

For such customers, aggregate reliability isn't enough—every reboot and every 30 second pause matters. Thus, Azure has moved to a more rigorous definition of what reliability should mean. We now focus on driving down the annual interruption rate (AIR)—the likelihood that a given virtual machine (VM) will see an interruption during the year—keeping our efforts focused across the usage lifecycle.

Proactive mitigation

Azure proactively mitigates potential failures—reducing the failure impact on availability by 50 percent. Using deep fleet telemetry, Microsoft enables failure predictions with machine learning (ML) and ties them to automatic live migration for several types of hardware failure cases, including disk failures, input/output (I/O) latency, and CPU frequency anomalies. As a result, VMs are live migrated off of "at-risk" machines before they ever show signs of failing. This means VMs running on Azure are more reliable than the underlying hardware.

For more details on Azure's innovative use of ML, please see the linked papers on disk failure prediction and node failure prediction.

Network considerations

The reliability and performance of cloud services is determined in part by the network over which the transactions take place. Microsoft has more datacenter regions than any of its competitors, and its network is one of the largest in the world. Unlike many other public cloud providers, data that traverses between Azure datacenters and regions doesn't go through the public internet. Rather, it stays safe within the Microsoft network.

This includes all traffic between Microsoft services, anywhere in the world. For example, within Azure, traffic between VMs, storage, and SQL communication traverses only the Microsoft network, regardless of the source and destination region. Cross-region traffic stays on the Azure Virtual Network as well. This keeps customers' applications and data both secure and highly available.

Built-in Azure resilience services

Despite all the investments Microsoft puts into making the platform reliable, apps can suffer from downtime because of unplanned events such as power failures, data corruption, ransomware attacks, as well as fires or natural disasters. The graphic below illustrates key Azure service offerings for various types of failures that can occur (Figure 1).

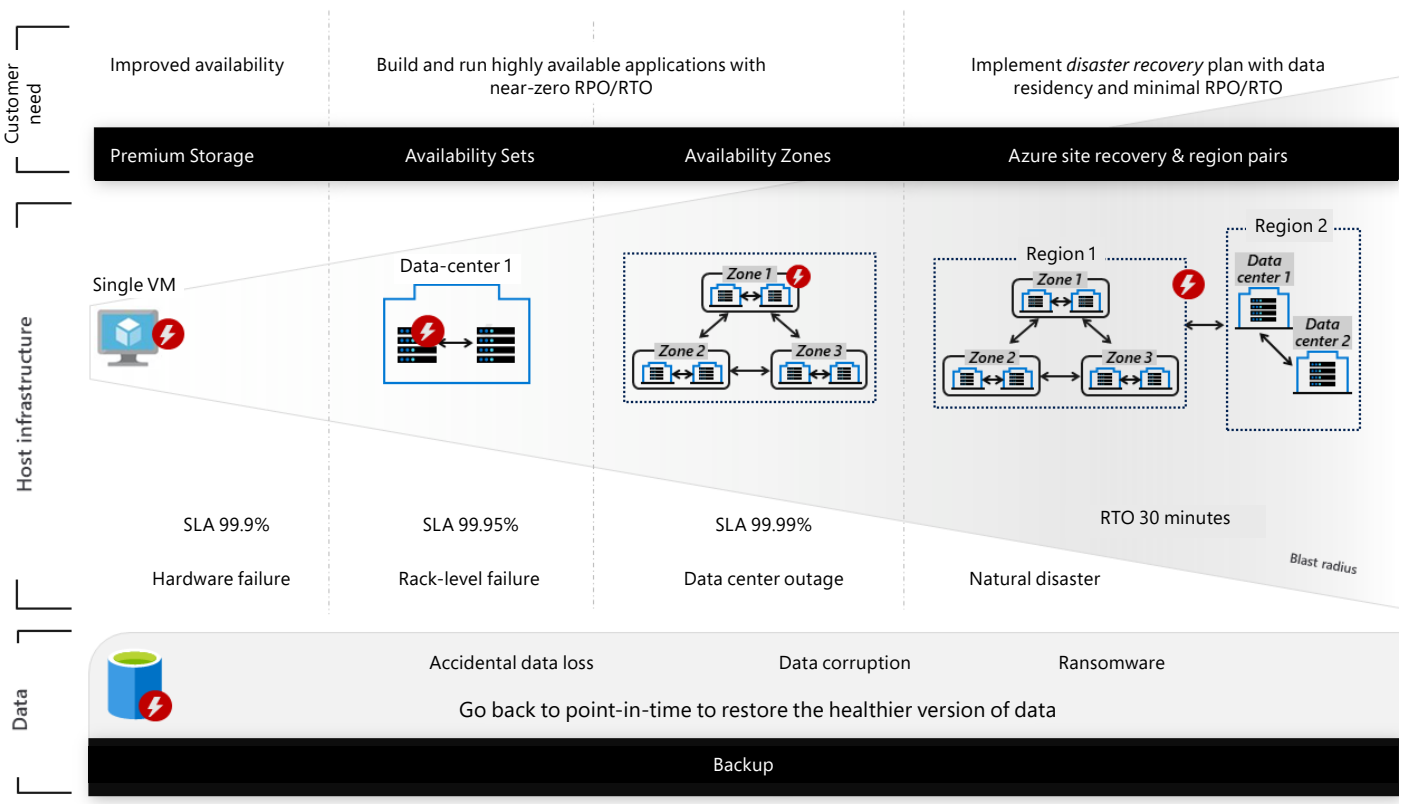


Figure 1. Azure resilience services

To help protect against the consequences of such failures, Azure provides a comprehensive set of built-in resilience services that customers can easily enable and control based on individual business needs. **Whether it's a single hardware node failure, a rack level failure, a datacenter outage, or a large-scale regional outage, Azure offers solutions, including the following:**

Availability Sets ensure that the VMs deployed on Azure are distributed across multiple isolated hardware nodes in a cluster.

Availability Zones protect customers' applications and data from datacenter failures across multiple physical locations within a region.

Azure Load Balancer distributes inbound traffic according to rules and health probes.

Azure Traffic Manager enables optimal traffic distribution to services across global Azure regions, while providing high availability (HA) and responsiveness.

Azure Site Recovery replicates workloads from a primary VM to a secondary failover location—allowing for business continuity and disaster recovery needs.

Azure Backup serves as a general backup solution for cloud and on-premises workflows that run on VMs or physical servers.

Geo-replication for Azure SQL Database allows an application to perform quick disaster recovery of individual databases in case of a regional disaster or large-scale outage.

Locally redundant storage (LRS) provides object durability by replicating customer data to a storage scale unit.

Zone redundant storage (ZRS) replicates customer data synchronously across three storage clusters in a single region.

Geo-redundant storage (GRS) provides object durability over a given year by replicating customer data to a secondary region that's hundreds of miles away from the primary region.

Shared responsibility

Azure invests in platform resilience and complements those investments with the abovementioned offerings to ensure uptime for customers. Being resilient in the event of any failure, however, is a shared responsibility. Who's responsible for what, in terms of resilience, depends on the cloud service model that's being used—IaaS, PaaS, or SaaS (Figure 2).

In the traditional on-premises model, the entire responsibility of managing—from the hardware for compute, storage, and networking to the application—falls on the customer. That requires planning for various types of failures and knowing how to deal with them on-premises. With IaaS, Azure is responsible for the core infrastructure resilience, which includes storage, networking, and compute. As the model moves from IaaS to PaaS and then to SaaS, the customer is responsible for less and the cloud service provider is responsible for more.

For more information about the shared responsibility model as it pertains to Azure cloud services, download the paper Shared Responsibilities for Cloud Computing Today.

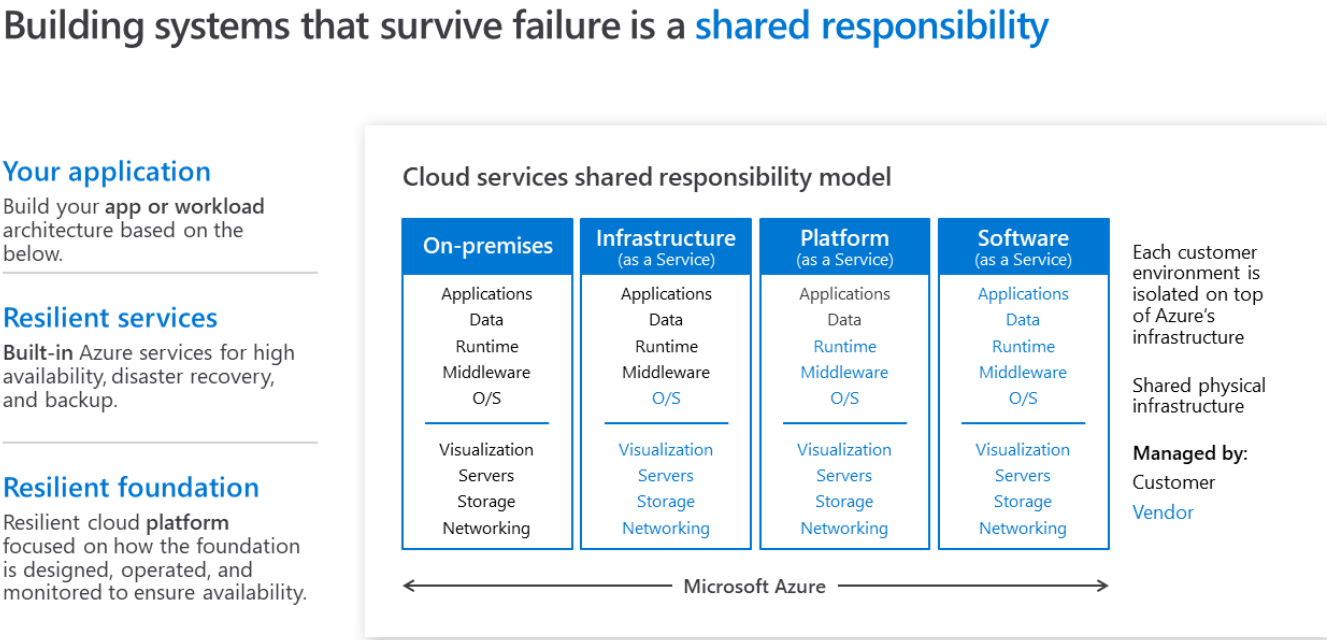
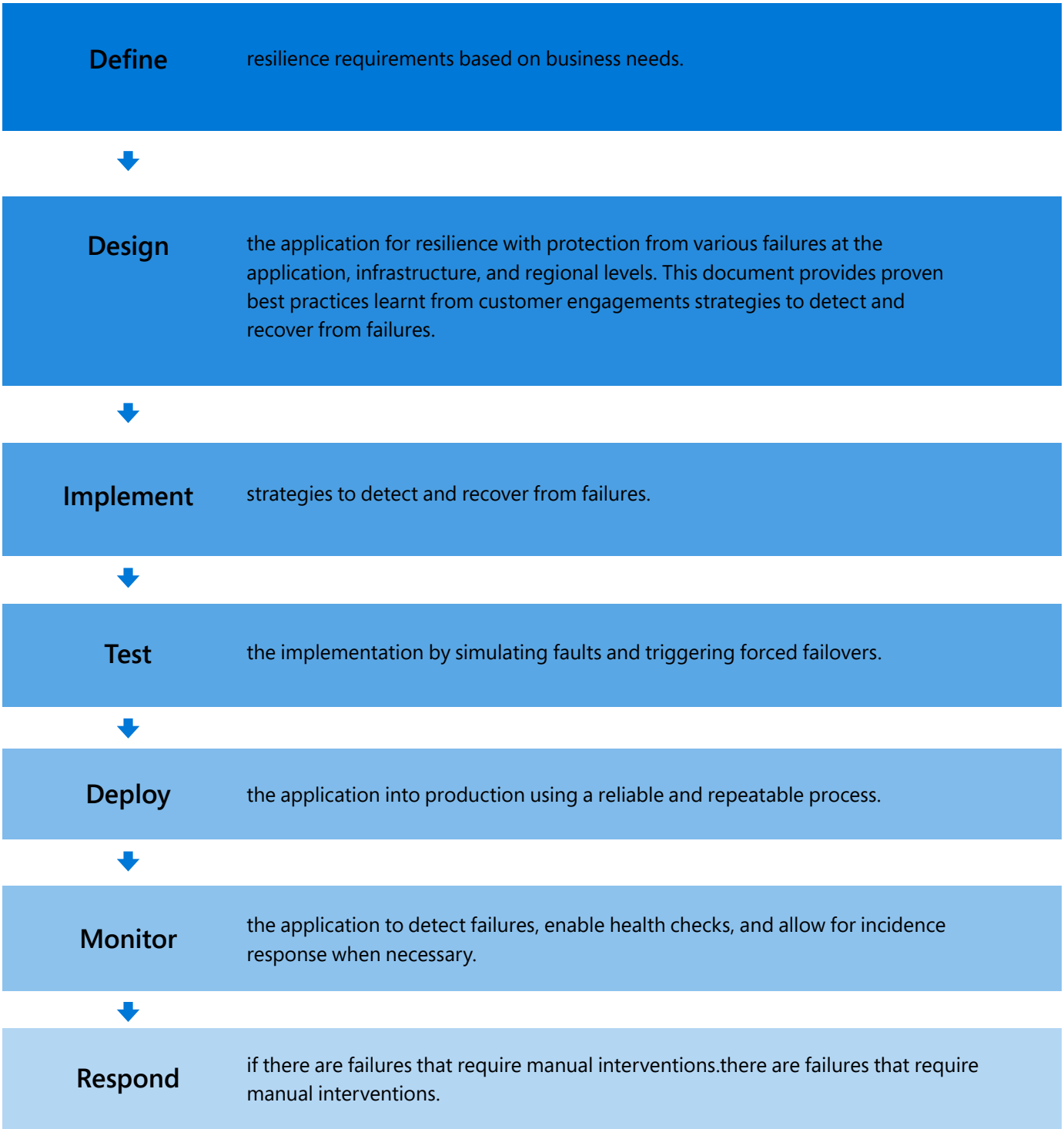


Figure 2: Shared responsbility for cloud services

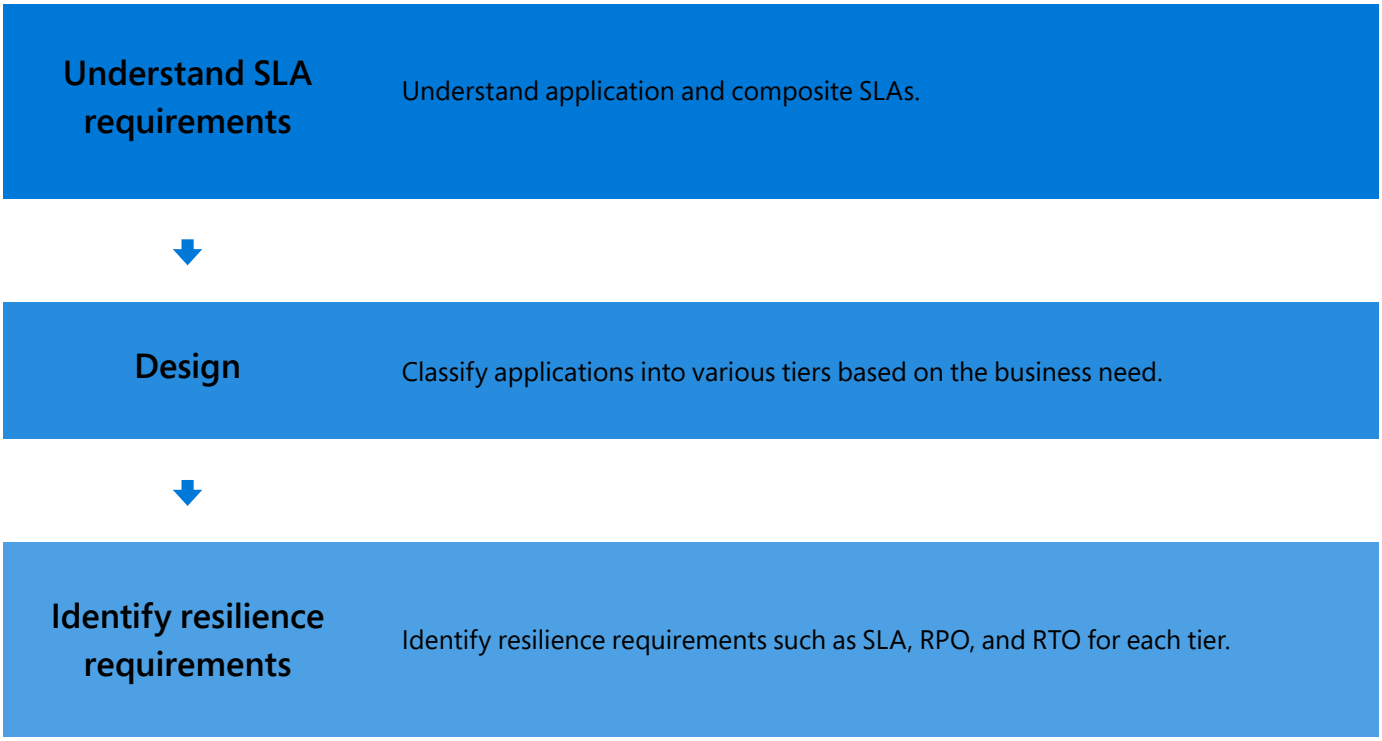
Design your applications

The following model provides a framework for designing resilient applications on Azure:



Define resilience requirements

The required level of resilience depends on several considerations. The first step is to define resilience requirements based on business needs, which include availability requirements and an understanding of the SLA.



Availability requirements

Understanding the applicable availability requirements is an important part of designing for resilience.

To better understand these requirements, ask the following questions:

- How much downtime is acceptable?
- How much will potential downtime cost the business?
- How much money and time can the business realistically invest in making the application highly available?

It's also important to define what it means for the application to *be available*. For example, is the application considered *down* if someone can submit an order, but the system can't process it within the normal timeframe?

Also, consider the probability of an outage occurring and whether a mitigation strategy is cost-effective. Resilience planning begins with business requirements. The following are some approaches to help in thinking about resilience in those terms.

SLAs

In Azure, the SLA describes our commitment regarding uptime and connectivity. If the SLA for a particular service is 99.9 percent, that means customers can expect the service to be available 99.9 percent of the time.

Azure customers should define their own target SLAs for each workload in their solutions. An SLA makes it possible to evaluate whether the architecture meets the business requirements. For example, if a workload requires 99.99 percent uptime, but depends on a service with a 99.9 percent SLA, that service can't be a single point of failure in the system.

Of course, higher availability is always better when everything else is equal. But as customers strive for higher percentages, the cost and complexity to achieve that level of availability grows. An uptime of 99.99 percent translates to about five minutes of total downtime per month. Is it worth the additional complexity and cost to reach that percentage? The answer depends on the individual business requirements.

To achieve four nines (99.99 percent), manual intervention can't be relied on to recover from failures. The application must be self-diagnosing and self-healing. Beyond four nines, it's challenging to detect outages quickly enough to meet the SLA. Think about the time window against which the SLA is measured: the smaller the window, the tighter the tolerances.

It probably doesn't make sense to define the SLA in terms of hourly or daily uptime. Consider the MTBF and MTTR measurements. The lower the SLA, the less frequently the service can go down, and the more quickly the service must recover.

The following table shows the potential cumulative downtime for various SLA levels:

SLA	Downtime per week	Downtime per month	Downtime per year
99%	1.68 hours	7.2 hours	3.65 days
99.9%	10.1 minutes	43.2 minutes	8.76 hours
99.95%	5 minutes	21.6 minutes	4.38 hours
99.99%	1.01 minutes	4.32 minutes	52.56 minutes
99.999%	6 seconds	25.9 seconds	5.26 minutes

Composite SLAs

Dealing with multiple services that have different SLAs adds complexity. In this case, it's necessary to calculate a composite SLA. Consider the Web Apps feature of Azure App Service which writes to Azure SQL Database. **At the time of this writing, these Azure services have the following SLAs:**

App Service Web Apps = 99.95 percent

SQL Database = 99.99 percent

What's the maximum downtime that should be expected for this application? If either service fails, the whole application fails. In general, the probability of each service failing is independent of the other. **So, the composite SLA for this application is the following:**

99.95% × 99.99% = 99.94%

That's lower than the individual SLAs. This isn't surprising, because an application that relies on multiple services has more potential failure points.

On the other hand, the composite SLA can be improved by creating independent fallback paths. For example, if SQL Database is unavailable, transactions are put into a queue to be processed later.

With this design, the application is still available even if it can't connect to the database. However, it fails if the database and the queue both fail at the same time. **The expected percentage of time for a simultaneous failure is 0.0001 × 0.001, so the composite SLA for this combined path is:**

Database OR queue = 1.0 − (0.0001 × 0.001) = 99.99999%

The total composite SLA is:

Web app AND (database OR queue) = 99.95% × 99.99999% = ~99.95%

There are tradeoffs to this approach. The application logic is more complex, the customer is paying for the queue, and there may be data consistency issues to consider.

SLA for multi-region deployments

Another HA technique is to deploy the application in more than one region and use Traffic Manager to fail over if the application fails in one region. **For a multi-region deployment, the composite SLA is calculated as follows:**

Let N be the composite SLA for the application deployed in one region, and R be the number of regions where the application is deployed. The expected chance that the application will fail in all regions at the same time is ((1 − N) ^ R).

For example, if the single-region SLA is 99.95 percent:

The combined SLA for two regions = (1 − (0.9995 ^ 2)) = 99.999975%

The combined SLA for four regions = (1 − (0.9995 ^ 4)) = 99.999999%

The SLA for Traffic Manager must also be factored in. At the time of this writing, the SLA for Traffic Manager is 99.99 percent.

Also, failing over isn't instantaneous in active/passive configurations, which can result in some downtime during a failover.

See Traffic Manager endpoint monitoring.

Decompose by workload

Many cloud solutions consist of multiple application workloads. The term workload in this context means a discrete capability or computing task, which can be logically separated from other tasks in terms of business logic and data storage requirements. **For example, an e-commerce app might include the following workloads:**

Browse and search a product catalog.

Create and track orders.

View recommendations.

These workloads might have different requirements for availability, scalability, data consistency, and disaster recovery. This means business decisions are required to balance cost versus risk.

Also consider usage patterns. Are there certain critical periods when the system must be available? For example, a tax-filing service can't go down right before the filing deadline, a video streaming service must stay up during a big sports event, and so on. During these critical periods, using redundant deployments across several regions will help provide resilience, so the application can fail over if one region fails. Because multi-region deployment is potentially more expensive, it's more cost effective to run the application in a single region during less critical times.

Categorizing applications into different tiers is a common strategy. Tier zero and tier one applications are made up of those applications that should experience very minimal data loss and downtime.

The RTO/RPO for this tier needs to be zero or as close to nil as possible. Tier two applications consist of applications for which it's acceptable to lose minimal amounts of data with RTO and RPO of the order of a few minutes. With tier three and tier four applications, downtime can affect internal operations for a few hours. While this is inconvenient, it doesn't pose a huge risk to the business.

The categorization of applications and the exact availability SLA, RPO, and RTO requirements vary from industry to industry, and from customer to customer. For example, an e-commerce website is tier zero or tier one for a retail company where the only sales channel is through the website. For a retail company where brick and mortar stores are the primary sales channel, the point of sales (POS) application is tier zero or tier one. Thus, it's important to categorize the application inventory into multiple tiers based on the organization's business needs and resilience requirements.

Sample tiers are as follows:

	Availability SLA	RPO	RTO
Tier 0	99.995	0	0
Tier 1	99.99	5 minutes	1 hour
Tier 2	99.95	30 minutes	4 hours
Tier 3	99.95	4 hours	8 hours
Tier 4	99	24 hours	72 hours

Design for resilience

Failures can vary in the scope of their impact. The table below describes various types of application failures. While this list isn't exhaustive, it provides a starting point to help customers think about various failure types.

Failure type	Description
Hardware failure	A failure of any hardware component including computer, network, or storage hardware.
Datacenter failure	An entire datacenter is impacted by issues such as a power grid outage.
Regional failure	This includes any natural disaster-like event that impacts multiple datacenters in a region, causing the entire region to go down.
Transient failure	Requests between various components fail intermittently. End user requests will fail when this isn't handled properly.
Dependency service failure	This occurs when any service on which the application is dependent is not functioning correctly.
Heavy load	A sudden spike in incoming requests prevents the application from servicing requests.
Accidental data deletion or corruption	Customers mistakenly delete critical data or data has been corrupted due to unforeseen reasons.
Application deployment failure	A failure that takes place when updating production application deployments.

Failure mode analysis

During the design phase, perform an FMA to identify possible points of failure and define how the application will respond to those failures. Customers can design response strategies for various types of failure depending on the application's resilience and availability requirements. Answering the following questions will help define an application's design for resilience.

- How will the application detect this type of failure?
- How will the application respond to this type of failure?
- How will you log and monitor this type of failure?

Making the FMA part of the architecture and design phases ensures that failure recovery is built into the system from the beginning.

Here are the steps for conducting an FMA:

- Identify all system components and include external dependencies, such as identity providers, third-party services, and so on.
- For each component, identify potential failures that could occur, keeping in mind that a single component may have more than one failure mode. For example, consider read failures and write failures separately, because the impact and possible mitigations of each will be different.

Rate each failure mode according to its overall risk. Consider these factors:

- What's the likelihood of the failure? Exact numbers aren't necessary because the purpose is to help rank the priority.
- What's the impact on the application, in terms of availability, data loss, monetary cost, and business disruption?

For each failure mode, determine how the application will detect, respond, and recover. Consider trade-offs in terms of cost and application complexity.

Implement resilience strategies for Azure applications

This section provides guidance on implementing common resilience strategies for applications to prevent or respond to various types of failures. Most of these aren't limited to a particular technology but provide a general idea of how to plan and implement resilience strategies.

Failure type	Resilience strategy
Hardware failure	Build redundancy into the application by deploying components across different fault domains. For example, ensure that Azure VMs are placed in different racks by using Availability Sets.
Datacenter failure	Build redundancy into the application with fault isolation zones across datacenters. For example, ensure that Azure VMs are placed in different fault-isolated datacenters by using Azure Availability Zones.
Regional failure	Replicate the data and components into another region so that applications can be quickly recovered. For example, use Azure Site Recovery to replicate Azure VMs to another Azure region.
Transient failure	Retry transient failures. For many Azure services, the client software development kit (SDK) implements automatic retries in a way that's transparent to the caller.
Dependency service failure	Degrade gracefully if a service fails without a failover path, providing an acceptable user experience.
Heavy load	Load balance across instances to handle spikes in usage. For example, put two or more Azure VMs behind a load balancer to distribute traffic to all VMs.
Accidental data deletion or corruption	Back up data so it can be restored if there's any deletion or corruption. For example, use Azure Backup to periodically back up your Azure VMs.
Application deployment failure	Automate deployments with a rollback plan.

Each resilience strategy is discussed in more detail in the subsections below.

Build redundancy. Build redundancy into applications to avoid a single point of failure. Ensure that VMs are deployed into different fault domains by creating an Availability Set and keeping a load balancer in front of it. You can also deploy VMs across two or more Availability Zones with a zone redundant load balancer in front of it.

Replicate data and components. Replicating data is a general strategy for handling non-transient failures in a data store. Many storage technologies provide built-in replication, including Azure Storage, Azure SQL Database, Azure Cosmos DB, and Apache Cassandra. It's important to consider both the read and write paths. Depending on the storage technology, you might have multiple writable replicas or a single writable replica and multiple read-only replicas.

To maximize availability, replicas can be placed in multiple regions. However, this increases the latency when replicating the data. Typically, replicating across regions is done asynchronously, which implies an eventual consistency model and potential data loss if a replica fails.

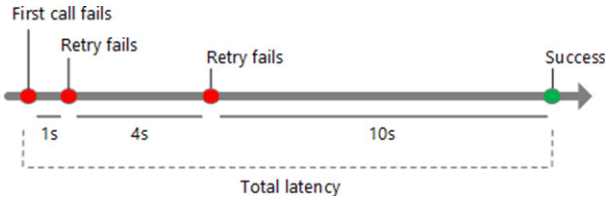


Figure 3. Total latency

Retry transient failures. Transient failures can be caused by momentary loss of network connectivity, a dropped database connection, or a timeout when a service is busy. Often, a transient failure can be resolved simply by retrying the request. For many Azure services, the client SDK implements automatic retries in a way that's transparent to the caller. For more on this, see Retry guidance for specific services.

Each retry attempt adds to the total latency (Figure 3). In addition, too many failed requests can cause a bottleneck as pending requests accumulate in the queue. These blocked requests might hold critical system resources such as memory, threads, database connections, and more, which can cause cascading failures. To avoid this, increase the delay between each retry attempt, and limit the total number of failed requests.

Load balance across instances. Scalability means a cloud application should be able to scale out by adding more instances. This approach also improves resilience, allowing for unhealthy instances to be removed from rotation. **For example:**

- When you put two or more VMs behind a load balancer, it distributes traffic to all of the VMs. See Run load-balanced VMs for scalability and availability.
- Scaling out an Azure App Service app to multiple instances automatically balances the load across instances. See how to Run a basic web application in Azure.
- Use Traffic Manager to distribute traffic across a set of endpoints.

Degrade gracefully. If a service fails and there's no failover path, the application may be able to degrade gracefully while still providing an acceptable user experience. **For example:**

- Put a work item on a queue to be handled later.
- Return an estimated value.
- Use locally cached data.
- Display an error message. (This option is better than having the application stop responding to requests.)

Back up data. Always configure backups for all critical production data sources. This includes VMs, databases, storage, among others. Accidental deletions or data corruptions can happen at any time. Personnel might not become aware of some of these until a few days or even weeks later. Thus, it's important to configure longer retention times for backup copies, depending on the nature and criticality of the application.

Key Azure services

Azure has features to make an application redundant at every level of failure, from an individual VM to an entire region. **These include:**

Single VM. Azure provides an uptime SLA for single VMs. (Note that the VM must use premium storage for all operating system disks and data disks.) Although running two or more VMs can result in a higher SLA, a single VM may be reliable enough for some workloads. However, for production workloads, Microsoft recommends using two or more VMs for redundancy.

Availability Sets. To protect against localized hardware failures, such as a disk or network switch failure, deploy two or more VMs in an Availability Set. An Availability

Set consists of two or more fault domains that share a common power source and network switch. VMs in an Availability Set are distributed across the fault domains, so if a hardware failure affects one fault domain, network traffic can still be routed to the VMs in the other fault domains. For more information about Availability Sets, see Manage the availability of Windows virtual machines in Azure.

Availability Zones. An Availability Zone is a physically separate location within an Azure region. They provide a combination of low latency and high availability through the strategic physical location separation within an Azure region. Each Availability Zone has independent physical infrastructure with a distinct power source, network, and cooling system. Deploying VMs across Availability Zones helps protect an application against datacenter-wide failures. See What are Availability Zones in Azure? for a list of supported regions and services.

When planning to use Availability Zones in a deployment, first validate that the application architecture and code base can support this configuration. When deploying commercial off-the-shelf software, consult with the software vendor and test adequately before deploying into production. The application must support running in an elastic and distributed infrastructure with no hard-coded infrastructure components specified in the code base.

Azure Site Recovery. Azure Site Recovery helps to replicate Azure VMs to another Azure region for business continuity and disaster recovery. Conduct periodic disaster recovery drills to ensure compliance requirements are met. The VM will be replicated with the specified settings to the selected region—ensuring customers can recover their applications in the event of outages in the source region. For more information, see Set up disaster recovery to a secondary Azure region for an Azure VM.

Customers should factor in the RTO and RPO numbers for their solutions here and ensure that when testing, the recovery time and recovery point is appropriate for their needs.

Paired regions. To protect an application in case of a regional outage, deploy the application across multiple regions using Traffic Manager. This distributes internet traffic to the different regions and pairs each Azure region with another region—forming a regional pair. With the exception of Brazil South, regional pairs are located within the same geography in order to

meet data residency requirements for taxation and law enforcement jurisdictional purposes.

When designing a multi-region application, network latency across regions is higher than within a region. For this reason, when replicating a database to enable failover, use synchronous data replication within a region, but asynchronous data replication across regions.

When selecting paired regions, ensure both regions have the required Azure services. For a list of services by region, see Products available by region.

It's also critical to select the best deployment topology for disaster recovery, especially if RPO/RTO requirements are low. To ensure the failover region has enough capacity to support the workload, select either an active/passive (full replica) topology or an active/active topology. Note that these deployment topologies might increase complexity and cost as resources in the secondary region are pre-provisioned and may sit idle. For more information, see Failure and disaster recovery for Azure applications.

Azure Backup. Azure Backup is the final and most powerful line of defense against permanent data loss. An effective backup strategy requires more than simply making copies of data. It needs to take the application's data architecture and infrastructure into consideration. The app may manage many kinds of data of varying importance, spread widely across filesystems, databases, and other storage services both in the cloud and on premises. Using the right services and products for the job will simplify the backup process and increase recovery time if a backup needs to be restored. Azure Backup serves as a general-purpose backup solution for cloud and on-premises workflows that run on VMs or physical servers. It's designed to be a drop-in replacement for traditional backup solutions, which stores data in Azure instead of archive tapes or other local physical media.

Azure Monitor. Azure Monitor maximizes the availability and performance of customer applications by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from both cloud and on-premises environments. It helps customers understand how their applications are performing and proactively identifies issues that affect them and the resources on which they depend.

The following table compares Availability Sets, Availability Zones, and Azure Site Recovery/paired regions:

	Availability Set	Availability Zone	Azure Site Recovery/Paired region
Scope of failure	Rack	Datacenter	Region
Request routing	Load Balancer	Cross-zone Load Balancer	Traffic Manager
Network latency	Very low	Low	Mid to high
Virtual network	Azure Virtual Network	Azure Virtual Network	Cross-region virtual network peering

Test failures and response strategies

Generally, resilience can't be tested in the same way as application functionality (that is, by running unit tests and so on). Instead, test how the end-to-end workload performs under failure conditions that only occur intermittently.

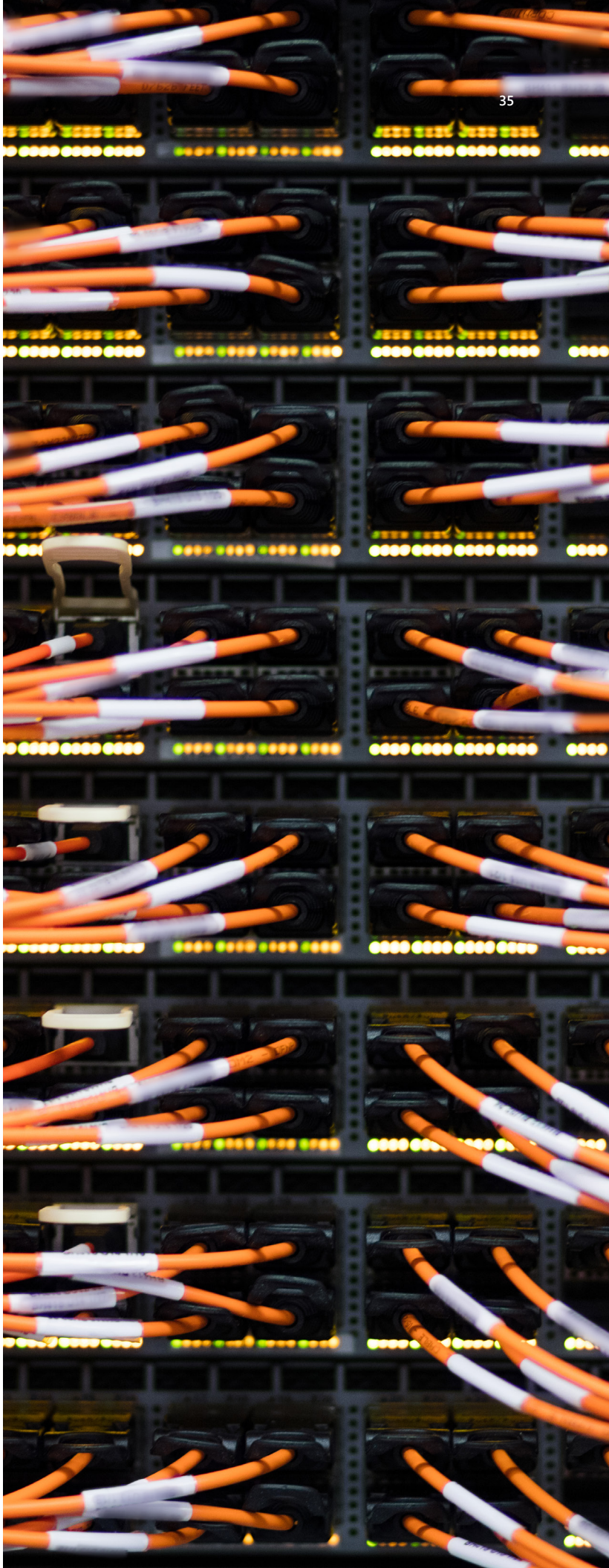
Testing is an iterative process. Test the application, measure the outcome, analyze and address any failures that result, and repeat the process.

Fault injection testing. Test the resilience of the system during failures, either by triggering actual failures or by simulating them. **The following are some common failure scenarios to test:**

- Crash processes.
- Expire certificates.
- Change access keys.
- Shut down the DNS service on domain controllers.
- Limit available system resources, such as RAM or number of threads.
- Unmount disks.
- Redeploy a VM.

Measure the recovery times and verify that business requirements are met. Test combinations of failure modes as well. Make sure that failures don't cascade and are handled in an isolated way.

This is another reason it's important to analyze possible failure points during the design phase. The results of that analysis should be input into the test plan.



Load testing. Load testing is crucial for identifying failures that only happen under load, such as the backend database being overwhelmed or service throttling. Test for peak load, using production data or synthetic data that's as close to production data as possible. The goal is to observe how the application behaves under real-world conditions.

Disaster recovery drills. It's not enough to have a good disaster recovery plan in place. It must also be tested periodically to ensure that the recovery plan works properly when it matters most. For Azure virtual machines, use Azure Site Recovery to replicate and perform disaster recovery drills—all without impacting production applications or ongoing replication.

Deploy using a reliable process

After an application is deployed to production, updates present a potential source of errors. In the worst case, a bad update can cause downtime. To avoid this, the deployment process must be predictable and repeatable. Deployment includes provisioning Azure resources, deploying application code, and applying configuration settings. An update may involve all three or a subset of these.

The crucial point is that manual deployments are prone to errors. Therefore, the best practice is to have an automated, idempotent process that can be run on demand and re-run if something fails.

Use Terraform, Ansible, Chef, Puppet, Azure PowerShell, Azure Command-Line Interface (CLI), or Azure Resource Manager templates to automate the provisioning of Azure resources.

Use Azure Automation Desired State Configuration (DSC) to configure Windows VMs. Use Cloud-init for Linux VMs.

Use Azure DevOps Services or Jenkins to automate application deployment.

Resilient deployment concepts

Infrastructure as Code (IaC) and *immutable infrastructure* are two important concepts in resilient deployments. **They're defined as follows:**

IaC is the practice of using code to provision and configure infrastructure. IaC may use a declarative approach, an imperative approach, or a combination of both.

Immutable infrastructure is the principle that infrastructure shouldn't be modified after it's deployed to production. Undocumented ad hoc changes are difficult to track and to troubleshoot, ultimately making it harder to address issues with the system.

A declarative approach focuses on what needs to be accomplished, whereas an imperative approach focuses on how to accomplish it. Resource Manager templates are an example of a declarative approach and PowerShell scripts are an example of an imperative approach.

Application updates

When rolling out an application update, Microsoft recommends blue-green deployment or canary releases to push updates in a highly controlled way and minimize possible impacts from a bad deployment. **These techniques are explained below:**

Blue-green deployment is a technique whereby an update is deployed into a production environment separately from the live application. After validating the deployment, switch the traffic routing to the updated version. For example, Azure App Service enables this with staging slots.

Canary releases are similar to blue-green deployments, but instead of switching all traffic to the updated version, the update is rolled out to a small percentage of users by routing a portion of the traffic to the new deployment. If there's a problem, back off and revert to the old deployment. Otherwise, incrementally route more traffic to the new version until it's handling 100 percent of the traffic.

Regardless of which approach is used, ensure the ability to roll back to the last known good deployment in case the new version isn't functioning. In addition, have a strategy in place to roll back database changes and any other changes to dependent services. If errors occur, the application logs must indicate which version caused the error.

Monitor to detect failures

Monitoring is crucial for resilience. If something fails, it's essential to know that it failed, and to get insights into the cause(s) of the failure.

Monitoring a large-scale distributed system poses a significant challenge. Think about an application that runs on a few dozen VMs. It's not practical to log into each VM one at a time and look through each log file to troubleshoot a problem. Moreover, the number of VM instances is usually not static because VMs are added and removed as the application scales in and out, and occasionally an instance may fail and need to be reprovisioned. To further complicate matters, a typical cloud application might use multiple data stores (Azure Storage, SQL Database, Cosmos DB, and Azure Cache for Redis) and a single user action may span multiple subsystems.

The monitoring process operates like a pipeline with several distinct stages:

Instrumentation. The raw data for monitoring comes from a variety of sources, including application logs, operating system performance metrics, Azure resources, Azure subscriptions, and Azure tenants. Most Azure services expose metrics that can be configured to analyze and determine the cause of problems.

Collection and storage. Raw instrumentation data can be held in various locations and in various formats (application trace logs, Internet Information Services logs, performance counters, and others). These disparate sources are collected, consolidated, and put into reliable data stores such as Azure Application Insights, Azure Monitor Metrics, Azure Service Health, storage accounts, and Azure Log Analytics.

Analysis and diagnosis. After the data is consolidated in these different data stores, it can be analyzed to troubleshoot issues and provide an overall view of application health. Generally, customers can search for the data in Application Insights and Log Analytics using Kusto queries. Azure Advisor provides recommendations with a focus on resilience and optimization.

Visualization and alerts. In this stage, telemetry data is presented in such a way that an operator can quickly notice problems or trends. Examples include dashboards or email alerts. With Azure dashboards, customers can build a single pane of glass view to monitor Application Insights graphs, Log Analytics, Azure Monitor metrics, and service health. Alerts from Azure Monitor enable notifications for service health and resource health issues.

Monitoring isn't the same as failure detection. For example, an application might detect a transient error and retry, resulting in no downtime. But it should also log the retry operation, so customers can monitor the error rate in order to get an overall picture of application health.

Application logs are an important source of diagnostics data. **Best practices for application logging include:**

- Log in production so as not to lose insight where it's needed most.
- Log events at service boundaries. Include a correlation ID that flows across service boundaries. If a transaction flows through multiple services and one of them fails, the correlation ID will help pinpoint why the transaction failed.
- Use semantic logging, also known as structured logging. Unstructured logs make it hard to automate the consumption and analysis of the log data, which is needed at cloud scale.
- Use asynchronous logging to avoid application failure. Otherwise, the logging system itself can cause requests to back up when waiting to write a logging event.

Application logging isn't the same as auditing. Auditing may be done for compliance or regulatory reasons. This means audit records must be complete, and it's not acceptable to drop or exclude any records while processing transactions. If an application requires auditing, this should be kept separate from diagnostics logging.

For more information, see Monitoring and diagnostics guidance.

Respond to failures

Previous sections have focused on automated recovery strategies, which are critical for high availability. However, sometimes manual intervention is needed.

Alerts. Monitor the application for warning signs that may require proactive intervention. For example, if SQL Database or Cosmos DB consistently throttle the application, increase your database capacity or optimize queries. In this example, even though the application might handle the throttling errors transparently, telemetry should still raise an alert so that you can follow up. Microsoft recommends configuring alerts on Azure resource metrics and diagnostics logs against the service limits and quota thresholds and further recommends setting up alerts on metrics, as they are lower latency than diagnostics logs. In addition, Azure is able to provide some out-of-the-box health statuses through resource health, which can help diagnose throttling of Azure services.

Failover. Configure a disaster recovery strategy for the application. The appropriate strategy will depend on the SLAs. For most scenarios, an active-passive implementation is sufficient. For more information, see Deployment topologies for disaster recovery. Most Azure services allow for either manual or automated failover. For example, in an IaaS application, use Azure Site Recovery for the web and logic tiers and SQL AlwaysOn availability groups for the database tier. Traffic Manager provides automated failover across regions.

Operational readiness testing. Perform an operational readiness test for both failover to the secondary region and failback to the primary region. Many Azure services support manual failover or test failover for disaster recovery drills. Alternatively, simulate an outage by shutting down or removing services.

Data consistency check. If a failure occurs in a data store, there may be data inconsistencies when the store becomes available again, especially if the data was replicated. For Azure services that provide cross-regional replication, look at the RTO and RPO to understand the expected data loss in a failure. Review the SLAs for Azure services to understand whether cross-regional failover can be initiated manually or if it will be initiated by Microsoft. For some services, Microsoft decides when to perform the failover. Microsoft may prioritize the recovery of data in the primary region, only failing over to a secondary region if data in the primary region is deemed unrecoverable. For example, GRS and Azure Key Vault follow this model.

Restoring from backup. In some scenarios, restoring from backup is only possible within the same region. This is the case for Azure Backup. Other Azure services, such as Azure Cache for Redis, provide geo-replicated backups. The purpose of backups is to protect against accidental deletion or corruption of data by restoring the application to an earlier functional version. Therefore, while backups can serve as a disaster recovery solution in some cases, the inverse isn't always true. Disaster recovery won't protect you against accidental deletion or corruption of data.

Example resilience design patterns

This section will focus on resilience design best practices for various application deployments with varied resilience requirements. Typically, customers classify the applications into various categories or tiers based on their resilience requirements, as discussed in the **resilience requirements** section. The following sections take an example application from each category and discuss how to design that application to be resilient against various types of failures.

Tier 4 application (99 application SLA, 24-hour RPO, 72-hour RTO)

The first category of applications will be the one with the least stringent availability requirements. The disaster recovery requirements are also on the low side with the acceptable data loss (RPO) of 24 hours and acceptable downtime (RTO) of three days. These can be internal applications such as tooling applications, build servers, project document share websites, and more.

A multi-tier web application in this category can be deployed within an Azure region as a single instance VM for each tier. If an explicit SLA guarantee at the VM level is desired, use premium storage for the VMs. Microsoft recommends using premium storage for the database VM after doing the trade-off with premium storage cost if the application is relatively important within this category. Azure is the first and only public cloud to provide an explicit SLA on single instance VMs.

The databases can be backed up using any backup software. For example, use Azure Backup and configure SQL database backups for SQL servers running on Azure VMs. Have Azure Resource Manager templates pre-created for the VMs so that VMs can be redeployed if there's an issue with the single instance VM in any tier in that region. For database VMs, use database backup copy to recreate databases.

Azure Backup can be used on single instance VMs to protect data and test backups using the restore feature. If there's any data or VM level corruption, recover the file, folder, disk, or VM using Azure Backup restore capabilities.

For disaster recovery during a regional failure scenario, consider replicating only the database VM with Azure Site Recovery. The web and app tier VMs can be redeployed in another Azure region using Azure Resource Manager templates if they're stateless or can be recovered from the backup copy. Note that the recovery time could be high for this approach. If a higher trade-off on cost is acceptable, replicate the VMs across all tiers to another region. Azure Site Recovery doesn't require running additional VM instances in the disaster recovery region. The VMs are created only when the user performs failover operations.

Monitor the health of the web application by using an automation script that periodically checks to determine whether the website endpoint is reachable. Create a custom endpoint that reports on the overall health of the application. The endpoint should return an HTTP error code if any critical dependency is unhealthy or unreachable. Don't report errors for non-critical services.

Use a pre-created script to monitor the simple health metrics of the VMs to detect whether there's an issue with the VM. Troubleshoot issues by checking the VM health metrics in the portal. Check for component health (such as CPU, memory, and disk) to determine whether there are potential load issues. If there are consistent issues with components such as CPU or RAM, consider increasing the VM to a larger size or consider scaling the application by deploying more VMs at each tier.

Application software updates can be deployed on the VMs using automation scripts during a weekend maintenance window. Ensure the automation script is in place to roll back if any issues are encountered during the deployment process.

The following table shows appropriate resilience strategies for each failure type for tier four applications:

Failure type	Resilience strategy
Hardware failure	Operate ready-to-use templates to deploy another instance using backup copies (if required). Test templates by deploying VMs into a test subnet or a test virtual network.
Datacenter failure	Operate ready-to-use templates to deploy another instance using backup copies (if required) in another zone. Test templates by deploying VMs into a test subnet or a test virtual network in another availability zone.
Regional failure	Use Azure Site Recovery to replicate the database VM. Test the disaster recovery using test failover and Azure Site Recovery plans Perform a disaster recovery failover in the event of an extended outage in the source region.
Heavy load	Load balance across instances to handle spikes in usage. For example, put two or more Azure VMs behind a load balancer to distribute traffic to all VMs.
Accidental data deletion or corruption	Use Azure Backup to back up the VMs. Test data recovery by restoring files, disks, and VMs. Restore data if there's an accidental deletion
Application deployment failure	Use automation scripts to deploy updates. If there's an issue observed during the update process or after the update, roll back to the previous version with an automated script.

Tier 3 application (99.95 application SLA, 4-hour RPO, 8-hour RTO)

The next category of applications is critical to business and requires high application SLA (Figure 4). However, it's acceptable to have some downtime for these applications. The disaster recovery RPO and RTO requirements can be a few hours. These can be internal applications such as expense management or travel management applications, which can have some impact if the applications are down for a few hours, but there won't be significant revenue loss. Less revenue-generating, customer-facing applications can also be part of this category.

Build redundancy for the applications in this category by deploying them as two or more VMs at each tier as part of an Availability Set. Availability Sets ensure that the VMs are placed in different fault domains and this guarantees that hardware failures such as a cluster or rack failure don't impact the end application.

Having two or more VMs in an Availability Set provides 99.95 percent availability for each tier. This will help get the overall composite SLA of the application to ebb within 99.9 percent. For the database VMs, use built-in synchronous replication to get high availability and avoid data loss. For example, use SQL AlwaysOn availability groups with asynchronous replication for the SQL databases.

Use load balancers between each tier so that traffic can be load balanced and routed to the healthy VM instances. If there's an issue with one of the VMs in a tier, the application will continue to work without any impact.

Use Azure Backup on all VMs to protect the data and test backups using the restore feature. The same feature can be used to recover a file, folder, disk, or VM if there's any data or VM level corruption. Use the SQL server database backup capability that's offered by Azure Backup to get more granular (as low as 15 minutes) database copies.

For disaster recovery in a regional failure scenario, consider replicating only the database VM with Azure Site Recovery. The web and app tier VMs can be redeployed in another Azure region using Azure Resource Manager templates if they're stateless or can be recovered from the backup copy. Note that the recovery time could be high for this approach. If a higher trade-off on cost is acceptable, replicate the VMs across all tiers to another region. Azure Site Recovery doesn't require running additional VM instances in

the disaster recovery region. The VMs are created only when the user performs failover operations.

Monitor the health of the web application by using an automation script that periodically checks to determine whether the website endpoint is reachable. Create a custom endpoint that reports on the overall health of the application. The endpoint should return an HTTP error code if any critical dependency is unhealthy or unreachable.

Use a pre-created script to monitor the simple health metrics of the VMs to detect if there's an issue with the VM. Troubleshoot any issues by checking the VM health metrics in the portal. Check for component health (such as CPU, memory, and disk) to determine whether there are potential load issues. If there are consistent issues with components such as CPU or RAM, consider increasing the VM to a larger size or consider scaling the application by deploying more VMs at each tier. Monitor advanced metrics for a VMs health as well as activities such as database failover when using asynchronous replication.

Application software updates can be deployed on the VMs using automation scripts during a weekend maintenance window. Ensure that the automation script is in place to roll back if any issues are encountered during the deployment process.

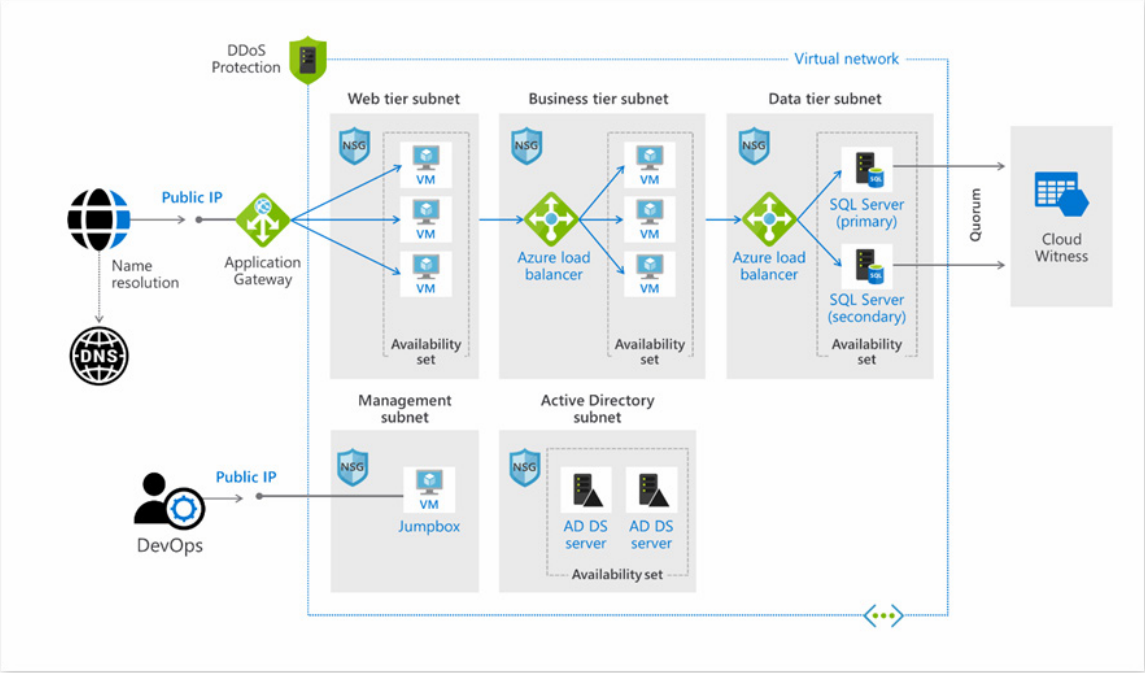


Figure 4. Typical resilience pattern for a tier 3 application

The following table shows appropriate resilience strategies for each failure type for tier three applications:

Failure type	Resilience strategy
Hardware failure	Build redundancy by deploying two or more instances in an Availability Set within a datacenter.
Datacenter failure	Operate ready-to-use templates to deploy another instance using backup copies (if required) in another availability zone. Test templates by deploying VMs into a test subnet or a test virtual network in another zone.
Regional failure	Use Azure Site Recovery to replicate the database VM. Test the disaster recovery using test failover and Azure Site Recovery plans. Perform a disaster recovery failover in the event of an extended outage in the source region.
Heavy load	Use monitoring tools to identify load surges on the VM. Increase the size of the VM or scale up by adding more instances.
Accidental data deletion or corruption	Use Azure Backup to back up VMs. Test data recovery by restoring files, disks, VMs, or SQL databases. Restore data if an accidental deletion occurs.
Application deployment failure	Use automation scripts to deploy updates. If an issue is observed during the update process or after the update, roll back to the previous version with an automated script.

Tier 2 application (99.99 application SLA, 30-minute RPO, 4-hour RTO)

This category of applications contains business critical apps whereby it can have significant impact on revenues if downtime occurs (Figure 5). These applications can be external customer-facing e-commerce websites, content streaming platforms, financial transaction handling applications, and the like. The applications should be highly available with resilience for all component failures.

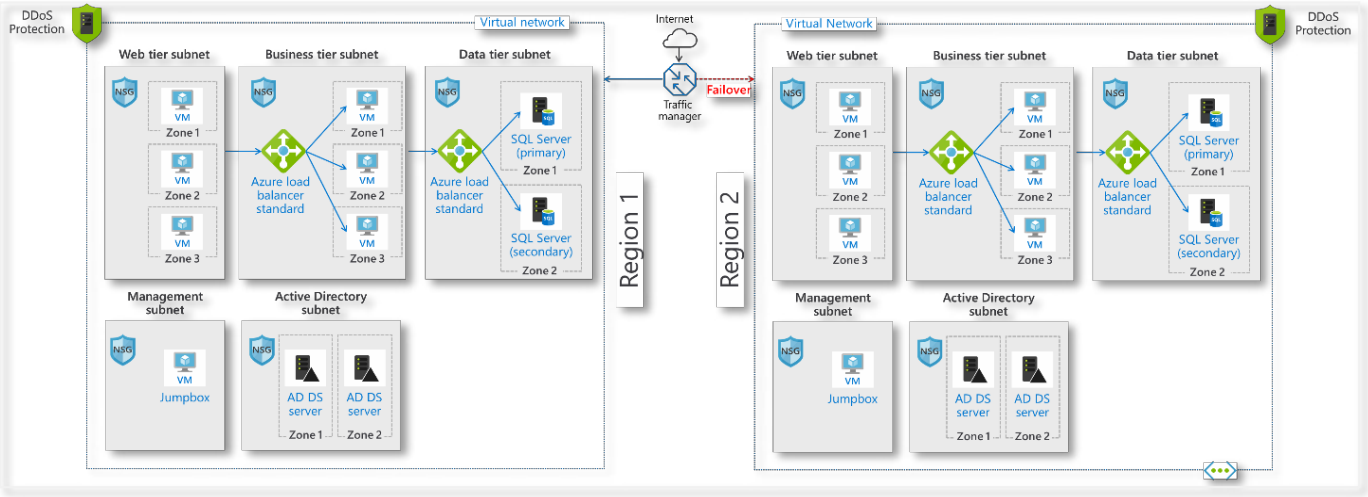


Figure 5. Typical resilience pattern for a tier 2 application

The following table shows appropriate resilience strategies for each failure type for tier two applications:

Failure type	Resilience strategy
Hardware failure	Build redundancy by deploying two or more instances across availability zones within a region.
Datacenter failure	Build redundancy by deploying two or more instances across availability zones within a region.
Regional failure	Use Azure Site Recovery to replicate all the VMs. Test the disaster recovery using test failover and Azure Site Recovery plans. Perform a disaster recovery failover in the event of an extended outage in the source region.
Heavy load	Provision enough capacity into the application. Use tools to monitor the load and add more instances that automatically use scripts if the threshold (for example, 70 percent) is reached.
Accidental data deletion or corruption	Use Azure Backup to back up all VMs and SQL databases. Test data recovery by restoring files, disks, VMs, or SQL databases. Restore data if an accidental deletion occurs.
Application deployment failure	Use safe deployment practices to roll out the updates to a minimal set of customers before deploying them widely. Use automation scripts to deploy updates with the automatic roll back capability built in if there's an issue with the update deployment. Configure alerts to send alarms/notifications if there is an issue occurs after an update deployment. If so, have the automated roll back script ready to execute.

Tier 1 application (99.99 application SLA, 5-minute RPO, 1-hour RTO)

This category of applications includes business and mission-critical apps that have strict requirements regarding data loss and recovery time. More than a few minutes of data loss can significantly impact the business and revenue. Customer-facing applications such as order processing systems and banking applications fall into this category.

The following table shows appropriate resilience strategies for each failure type for tier one applications:

Failure type	Resilience strategy
Hardware failure	Build redundancy by deploying two or more instances across availability zones within a region.
Datacenter failure	Build redundancy by deploying two or more instances across availability zones within a region.
Regional failure	Use Azure Site Recovery to replicate all VMs in the web tier and middle tier. Use native replication technologies such as SQL AlwaysOn. Test the disaster recovery of the complete application (including SQL AlwaysOn failover using Azure Site Recovery plans) and test failover capabilities. Perform a disaster recovery failover in the event of an extended outage in the source region.
Heavy load	Provision enough capacity into the application. Use tools to monitor the load and add more instances that automatically use scripts if the threshold (for example, 70 percent) is reached.
Accidental data deletion or corruption	Use Azure Backup to back up all VMs and SQL databases. Test data recovery by restoring files, disks, VMs, or SQL databases. Restore data if an accidental deletion occurs.
Application deployment failure	Use safe deployment practices to roll out the updates to a minimal set of customers before deploying them widely. Use automation scripts to deploy updates with the automatic roll back capability built in if there's an issue with the update deployment. Configure alerts to send alarms/notifications if an issue occurs after an update deployment. If so, have the automated roll back script ready to execute.

Tier 0 application (99.995 application SLA, 0 RPO, 0 RTO)

A few business applications will be mission critical and will require close to 100 percent availability, no data loss, and no downtime. An example of this is an e-commerce website where the only sales channel is through a website. This site can't accommodate any downtime or data loss, especially during the holiday season. Similarly, a stock trading website for a financial services company can't have any downtime or data loss.

Traditionally, implementing highly available applications across multiple datacenters that are hundreds of kilometers apart wasn't feasible. With the new technologies and services now available in Microsoft Azure, a mission-critical business application can be made highly available across regions. Note that such availability guarantees come at a high cost.

Customers can architect their applications on Azure using various modern service offerings such as App Service plan, Cosmos DB, Azure Active Directory, Azure Cache for Redis, and Azure Search to ensure the application is highly available across multiple regions and will run with low latencies (Figure 6).

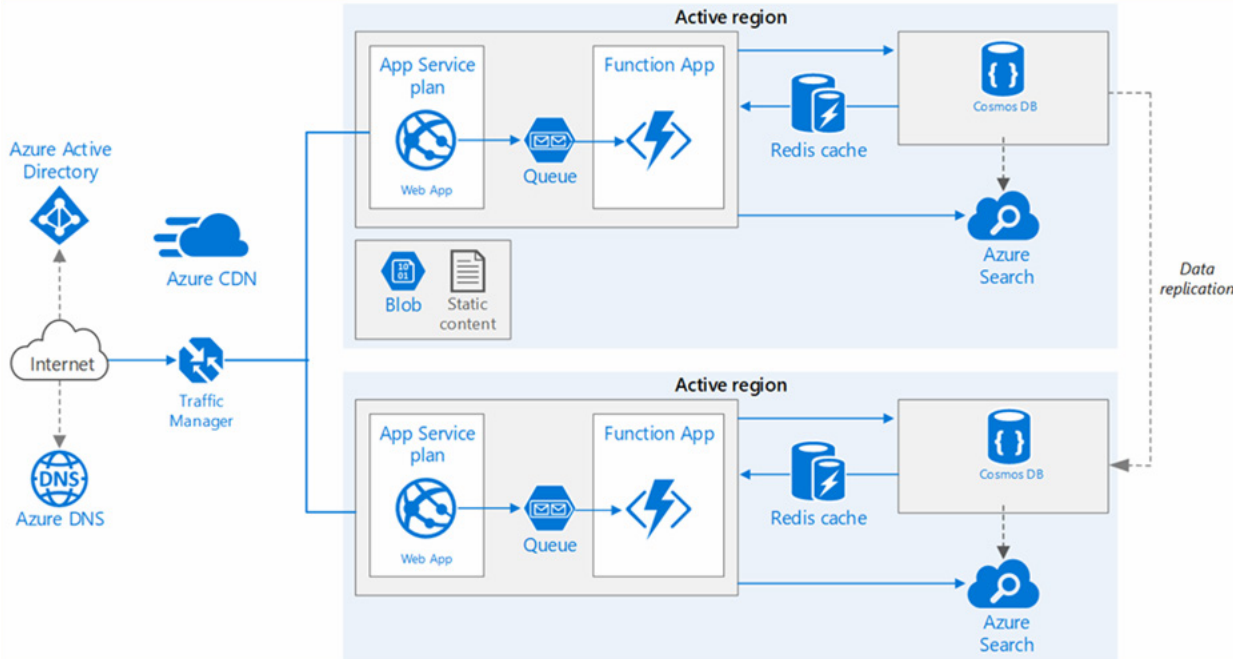


Figure 6. Typical resilience pattern for a tier 0 application

The following table shows appropriate resilience strategies for each failure type for tier zero applications:

Failure type	Resilience strategy
Hardware failure	Build redundancy by deploying two or more VM instances across availability zones within a region.
Datacenter failure	Build redundancy by deploying two or more VM instances across availability zones within a region.
Regional failure	Use Azure Site Recovery to replicate all VMs in the web tier and middle tier. Use global data distribution with Cosmos DB. Test the disaster recovery of the complete application (including Cosmos DB failover). Perform disaster recovery failover in the event of an extended outage in a source region.
Heavy load	Provision enough capacity into the application. Use tools to monitor the load and add more instances that automatically use scripts if the threshold (for example, 70 percent) is reached.
Accidental data deletion or corruption	Use Azure Backup to back up all VMs and SQL databases. Test data recovery by restoring files, disks, VMs or SQL databases. Restore the data if an accidental deletion occurs.
Application deployment failure	Use safe deployment practices to roll out the updates to a minimal set of customers before deploying it widely. Use automation scripts to deploy updates with the automatic roll back capability built in if there's an issue with the update deployment. Have alerts configured to send alarms if an issue occurs after an update deployment. If any occur, have the automated roll back script ready to execute.

Conclusion

This document discusses the importance of resilience when designing applications on Azure and the process of designing and deploying highly resilient applications. It's designed to help customers understand how Microsoft continuously improves Azure's platform reliability by investing in the foundation aspects. It also provides an overview of the built-in services offered by Azure that can be leveraged to design and deploy resilient applications.

It's important for customers to understand the shared responsibility model and the expectations that go along with the SLA when deploying applications in the cloud. Microsoft can help organizations design their application components to be resilient to failures.

The example resilience design patterns discussed in this document are meant to help customers as they start running their production applications on Azure.

To provide feedback or to share the best practices you followed to deploy highly resilient applications on Azure, please **contact us**.