Azure Resiliency – Business Continuity and Disaster Recovery

January 2022

Abstract

This document focuses on the resiliency aspect of Azure. It provides guidance on designing resilient applications on Azure and provides sample application design patterns for varied levels of resilience. The audience for this document is anyone moving applications from on-premises to Azure, or building applications on Azure, including CTOs, CIOs, Cloud Solution Architects, Business Continuity and Disaster Recovery Administrators, Application Developers and Operations team members. For more information and guidance, see the resiliency resources page at the Microsoft Azure Well-Architected Framework.

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. It integrates tools, templates, and managed services that work together to make it easier to build and manage enterprise, mobile, web, and Internet of Things (IoT) apps faster, using the tools, applications, and frameworks you choose. The cloud enables these exciting technologies.

Azure is built on trust. The Azure approach to trust is based on the following foundational principles:

- ***Security***
  Azure leverages leading security technologies to help organizations manage and control user identity and access, which are central elements in securing your environment.

- ***Compliance***
  Through rigorous and widely recognized formal standards that are certified by independent third parties, Microsoft helps organizations comply with constantly shifting requirements and regulations.

- ***Privacy***
  You have control over where your data is located, who can access it, and on what terms. You can access your own customer data at any time and for any reason.

- ***Resiliency***
  Microsoft Azure helps you to avoid many potential disasters and quickly recover if your organization gets hit by disaster.

- ***Intellectual Property (IP) protection***
  Trust in the cloud also encompasses clarity and confidence that your intellectual property will be protected against frivolous infringement claims. Microsoft Azure IP Advantage and the Shared Innovation Initiative can offer that assurance.

In a distributed system, failures can happen. Hardware can malfunction. The network can have transient failures. Rarely, an entire service or region may experience a disruption. It is important to plan for any failures through continuous monitoring and regular tests. This document describes a process for achieving resiliency, using a structured approach over the lifetime of an application, from design and implementation to deployment and operations.

What is resiliency?

Resiliency is the ability of a system to recover from failures and continue to function. It's not just about *avoiding* failures but *responding* to failures in a way that avoids downtime or data loss. The goal of resiliency is to avoid failures and if they still occur, return the application to a fully

functioning state following an occurrence. The faults can be of various levels. It is important to have protection based on your application availability requirements.

Foundation

The first step to achieving resiliency is avoiding failures in the first place. Azure invests heavily in the cloud platform to ensure that you can run your workloads reliably. Our approach to improving Azure reliability involves improving the platform capability to minimize impact during planned maintenance events and giving you control over the experience during these events. We have come a long way over the years innovating technologies such as memory preserving host updates and live migration. Azure overall availability has been trending up constantly; in fact, we're at about 99.999% reliability across the fleet. More mission critical customers are relying on Azure for their operations. For such customers, aggregate reliability is not enough – every reboot, every 30s-pause, matters. In response, Azure has moved toward a more rigorous definition of what reliability should mean, focusing on driving down the Annual Interruption Rate (AIR) – the likelihood that a given VM will see an interruption during the year. These efforts are consistent across the entire usage lifecycle.

Azure proactively mitigates potential failures, reducing the impact of failures on availability by 50 percent. Azure deep fleet telemetry enables machine learning-based failure predictions and ties them to automatic live migration for several hardware failure cases, including disk failures, IO latency, and CPU frequency anomalies. As a result, Azure can live migrate workloads off "at-risk" machines before they ever show any signs of failing. This means VMs running on Azure can be more reliable than the underlying hardware. please See Improving Service Availability of Cloud Systems by Predicting Disk Error and Predicting Node Failure in Cloud Service Systems for more information on innovative use of machine learning in Azure.

The reliability and performance of cloud services is also determined in part by the network. In addition to having more datacenter regions than any of our competitors, the Microsoft cloud computing network is also 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 - it stays on our network. This includes all traffic between Microsoft services anywhere in the world. For example, within Azure, traffic between virtual machines, storage, and SQL communication traverses only the Microsoft network, regardless of the source and destination region. Intra-region VNet-to-VNet traffic, as well as cross-region VNet-to-VNet traffic, stays on the Microsoft network. This helps ensure your applications and data are both secure and highly available.

Resilient services

Even with all the improvements made to the platform, apps can encounter downtime because of unplanned events like fires or natural disasters.

Microsoft provides a comprehensive set of built-in resiliency services that you can easily enable and control based on your business needs. Whether it's a single hardware node failure, a rack

level failure, DC outage or large-scale region outage, Microsoft offers solutions you can leverage in Azure.

Availability Sets provide a logical grouping of VMs that allows Azure to understand how your application is built in order to deliver redundancy and availability.

Availability Zones protect your applications and data from datacenter failures. You can distribute the VMs belonging to one tier across multiple availability zones within a region.

Azure Load Balancer distributes inbound traffic according to rules and health probes. You can distribute incoming client traffic using load balancers.

Azure Traffic Manager enables you to distribute traffic optimally to services across global Azure regions while providing high availability and responsiveness.

Azure Site Recovery allows you to replicate virtual machines to another Azure region for business continuity and disaster recovery needs.

Azure Backup service provides simple, secure, and cost-effective solutions to back up your data and recover it from the Microsoft Azure cloud. You can use Azure Backup to protect on-premises workloads, Azure Virtual Machines, SQL Server, and SAP HANA databases running on Azure VMs, Azure Managed Disks, Azure Blobs, Azure File Shares, and more.

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

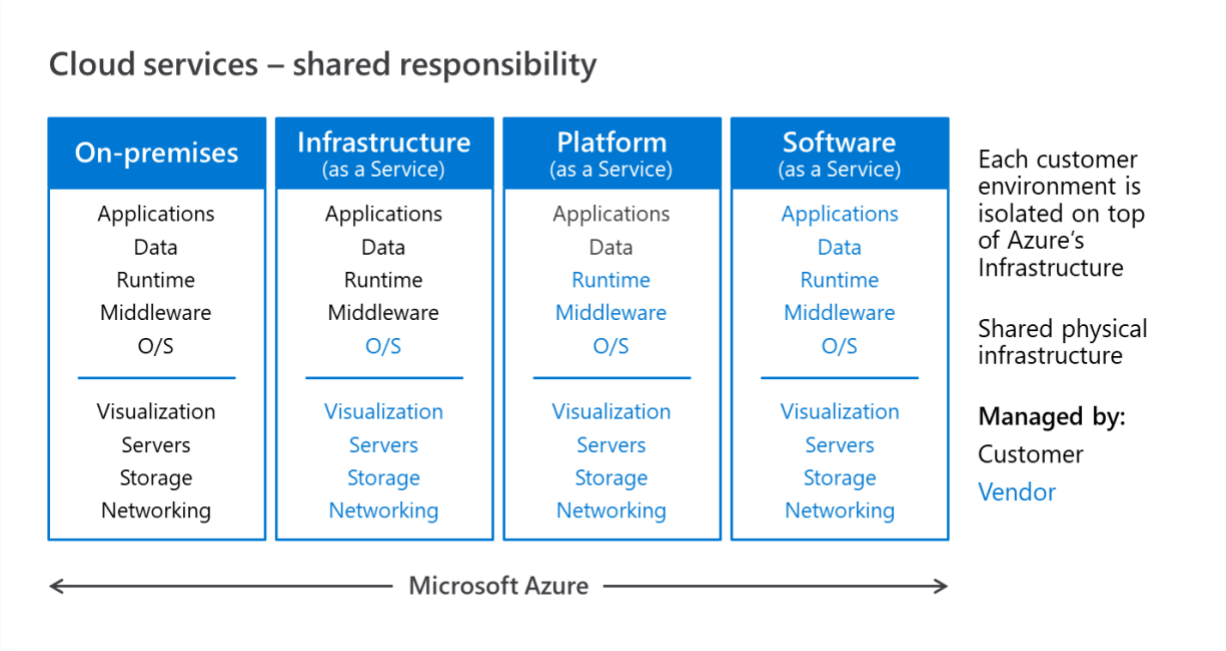
Locally-redundant storage (LRS) provides object durability by replicating your data to a storage scale unit.

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

Geo-redundant storage (GRS) is designed to provide durability of objects over a given year by replicating your data to a secondary region that is hundreds of miles away from the primary region.

Shared responsibility

Azure invests in platform resilience and complements these investments with the abovementioned offerings to ensure uptime. Being resilient in the event of any failure, however, is a shared responsibility. Who is responsible for what regarding resiliency depends on the cloud service model you use — IaaS, PaaS, or SaaS.



In the traditional on-premises model, the entire responsibility of managing, from the hardware for compute, storage and networking to the application falls on you. You must plan for various types of failures and how to deal with them on-premises. With IaaS, the cloud service provider is responsible for the core infrastructure resiliency, including storage, networking, and compute. As you move from IaaS to PaaS and then to SaaS, you'll find that you're responsible for less and the cloud service provider is responsible for more.

Design your applications

You can leverage the following model for designing your applications:

1. **Define** your resiliency requirements based on business needs.
2. **Design** the application for resiliency. Start with an architecture that follows proven practices and then identify the possible failure points in that architecture.
3. **Implement** strategies to detect and recover from failures.
4. **Test** the implementation by simulating faults and triggering forced failovers.
5. **Deploy** the application into production using a reliable and repeatable process.
6. **Monitor** the application to detect failures. By monitoring the system, you can gauge the health of the application and respond to incidents if necessary.
7. **Respond** if there are failure that require manual interventions.

Define Resiliency requirements

Availability requirements

When you design for resiliency, you must understand your availability requirements. How much downtime is acceptable? How much will potential downtime cost your business? How much should you invest in making the application highly available? You also must define what it means for the application to be available. For example, is the application "down" if you can submit an order but the system cannot process it within the normal timeframe? Also consider the probability of an outage occurring and whether a mitigation strategy is cost-effective. Resilience planning starts with business requirements. Here are some approaches for thinking about resiliency in those terms.

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. There are business decisions to be made in terms of balancing 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 the critical periods, you might have redundant deployments across several regions, so the application could fail over if one region failed. However, a multi-region deployment is potentially more expensive, so during less critical times, you might run the application in a single region.

Qualifying applications into different tiers is a common strategy. Tier 1 applications are made up of those applications that should not experience any loss in data. The RTO/RPO for this tier needs to be as close to zero as possible. Tier 2 applications consist of those applications that can lose minimal amounts of data with RTO and RPO in the order of a few minutes. Tier 3 and 4 applications are those that affect internal operations for a few hours and while inconvenient, do not pose a huge risk to the business.

Sample tiers:

Tier 1	ATM transactions, telecommunications systems
Tier 2	Video delivery, broadcast systems, online commerce

Tier 3	Internal tools like knowledge management, project tracking
Tier 4	Batch processing, data extraction, transfer, and load jobs

SLAs

In Azure, the Service Level Agreement (SLA) describes Microsoft's commitments for uptime and connectivity. If the SLA for a particular service is 99.9%, it means you should expect the service to be available 99.9% of the time.

You should define your own target SLAs for each workload in your solution. An SLA makes it possible to evaluate whether the architecture meets the business requirements. For example, if a workload requires 99.99% uptime, but depends on a service with a 99.9% SLA, that service cannot be a single-point of failure in the system.

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

Of course, higher availability is better, everything else being equal. But as you strive for more 9s, the cost and complexity to achieve that level of availability grows. An uptime of 99.99% translates to about 5 minutes of total downtime per month. Is it worth the additional complexity and cost to reach five 9s? The answer depends on the business requirements.

To achieve four 9s (99.99%), you probably can't rely on manual intervention to recover from failures. The application must be self-diagnosing and self-healing. Beyond four 9s, it is challenging to detect outages quickly enough to meet the SLA. Think about the time window that your SLA is measured against. The smaller the window, the tighter the tolerances. It probably doesn't make sense to define your SLA in terms of hourly or daily uptime. Consider the MTBF and MTTR

measurements. The lower your SLA, the less frequently the service can go down, and the quicker the service must recover.

Composite SLAs

Consider an App Service web app that writes to Azure SQL Database. At the time of this writing, these Azure services have the following SLAs:

- App Service Web Apps = 99.95%
- SQL Database = 99.99%

What is the maximum downtime you would expect for this application? If either service fails, the whole application fails. In general, the probability of each service failing is independent, so the composite SLA for this application is 99.95% × 99.99% = 99.94%. That's lower than the individual SLAs, which isn't surprising, because an application that relies on multiple services has more potential failure points.

On the other hand, you can improve the composite SLA by creating independent fallback paths. For example, if SQL Database is unavailable, put transactions 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%

But there are tradeoffs to this approach. The application logic is more complex, you are 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 Azure 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%,

- The combined SLA for two regions = (1 − (0.9995 ^ 2)) = 99.999975%
- The combined SLA for four regions = (1 − (0.9995 ^ 4)) = 99.999999%

You must also factor in the SLA for Traffic Manager. At the time of this writing, the SLA for Traffic Manager SLA is 99.99%.

Also, failing over is not instantaneous in active-passive configurations, which can result in some downtime during a failover. See Traffic Manager endpoint monitoring and failover for more information.

RTO and RPO

Two important metrics to consider are the recovery time objective and recovery point objective, as they pertain to disaster recovery.

- **Recovery time objective** (RTO) is the maximum acceptable time that an application can be unavailable after an incident. If your RTO is 90 minutes, you must be able to restore the application to a running state within 90 minutes from the start of a disaster. If you have a very low RTO, you might keep a second regional deployment continually running an active/passive configuration on standby, to protect against a regional outage. In some cases, you might deploy an active/active configuration to achieve even lower RTO.
- **Recovery point objective** (RPO) is the maximum duration of data loss that is acceptable during a disaster. For example, if you store data in a single database, with no replication to other databases, and perform hourly backups, you could lose up to an hour of data.

RTO and RPO are non-functional requirements of a system and should be dictated by business requirements. To derive these values, it's a good idea to conduct a risk assessment, and clearly understanding the cost of downtime or data loss.

MTTR and MTBF

Two other common measures of availability are mean time to recover (MTTR) and mean time between failures (MTBF).

Mean time to recover (MTTR) is the average time that it takes to restore a component after a failure. MTTR is an empirical fact about a component. Based on the MTTR of each component, you can estimate the MTTR of an entire application. Building applications from multiple components with low MTTR values results in an application with a low overall MTTR — one that recovers quickly from failures.

Mean time between failures (MTBF) is the runtime that a component can reasonably expect to last between outages. This metric can help you to calculate how frequently a service will become unavailable. An unreliable component has a low MTBF, resulting in a low SLA number for that component. However, a low MTBF can be mitigated by deploying multiple instances of the component and implementing failover between them.

Design for resiliency

Failures can vary in the scope of their impact. The below table lists out various types of failures that can occur for your applications. This list is not exhaustive, but a good directional guidance to think about various failures.

Failure type	Description
Hardware failure	Any hardware component failure including compute, network, or storage hardware
Datacenter failure	Entire data center impacted by issues such as power grid outage
Regional failure	Any natural disaster-like event impacting multiple data centers in a region, causing the entire region to go down.
Transient failure	An intermittent issue causing the request between various components to fail intermittently. If not handled properly, end user requests will fail.
Dependency service failure	Any service that the application is dependent on not functioning correctly.
Heavy load	Sudden spike in the incoming requests make the application unable to service the requests.
Accidental data deletion or corruption	Critical data can often be deleted accidentally by the customer, or the data can be corrupted due to unforeseen reasons.
Application deployment failure	Failure caused by an issue while updating the production application deployments.

Failure mode analysis (FMA)

During the design phase, you should perform a failure mode analysis (FMA). The goal of an FMA is to identify possible points of failure and define how the application will respond to those failures. Depending on your application's resiliency and availability requirements, you can design response strategies for various types of failure. Below are the questions to help define your application's design for resiliency.

- 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?

The FMA should be part of the architecture and design phases so that you can build failure recovery into the system from the beginning.

To conduct an FMA:

1. Identify all the components in the system. Include external dependencies, such as identity providers, third-party services, and so on.

2. For each component, identify potential failures that could occur. A single component may have more than one failure mode. For example, you should consider read failures and write failures separately, because the impact and possible mitigations will be different.
3. Rate each failure mode according to its overall risk. Consider these factors:
    - What is the likelihood of the failure? You don't need exact numbers; the purpose is to rank the priority.
    - What is the impact on the application, in terms of availability, data loss, monetary cost, and business disruption?
4. For each failure mode, determine how the application will detect, respond, and recover. Consider tradeoffs in cost and application complexity.

Implement resiliency strategies

This section provides common resiliency strategies that can be implemented for your applications for various types of failures. Most of these are not limited to a particular technology; each is intended to give you a general idea of how to plan and implement resiliency strategies.

Failure type	**Resiliency strategy**
Hardware failure	**Build redundancy** into the application by deploying components across different fault domains. For example, ensure your VMs are placed in two different racks.
Datacenter failure	**Build redundancy** into the application with fault isolation zones across data centers.
Regional failure	**Replicate the data and components** into another region so that application can be quickly recovered.
Transient failure	**Retry transient failures.**
Heavy load	**Load balance across instances** to handle spike in usage.
Dependency service failure	**Degrade gracefully** if a service fails and there is no failover path, providing an acceptable user experience.
Accidental data deletion or corruption	**Backup the data** so that it can be restored if there is any deletion or corruption.
Application deployment failure	**Automate deployments** with rollback plan.

Build redundancy

Build redundancy into your application to avoid single point of failure. Ensure your VMs get deployed into different fault domains by creating an availability set and keeping load balancer in front of it. You can also deploy VMs across two or more availability zones with zone redundant load balancer in front of it.

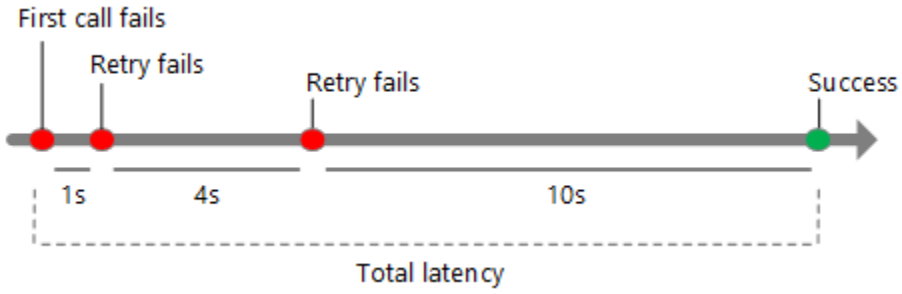
Replicate data and components

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, 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; this implies an eventual consistency model and potential data loss if a replica fails.

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 is transparent to the caller. See Retry service specific guidance for more information.

Each retry attempt adds to the total latency. Also, 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 so on; this 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. For scalability, a cloud application should be able to scale out by adding more instances. This approach also improves resiliency because unhealthy instances can be removed from rotation. For example:

- Put two or more VMs behind a load balancer. The load balancer distributes traffic to all the VMs. See Load balance VMs for high availability for more information.
- Scale out an Azure App Service app to multiple instances. App Service automatically balances load across instances. See Basic web application for more information.

- Use Azure Traffic Manager to distribute traffic across a set of endpoints.

Degrade gracefully. If a service fails and there is no failover path, the application might 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.
- Show the user an error message. (This option is better than having the application stop responding to requests.)

Backup the data. You should always configure backups for all your data sources – VMs, databases, storage, and so on. Accidental deletions or data corruptions can happen anytime. It could be days before such issues are noticed. For this reason, it is important to configure longer retention for your backup copies depending on the nature and criticality of the application.

Key Azure services

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

Single VM. Azure provides an uptime SLA for single VMs. (The VM must use premium storage for all Operating System Disks and Data Disks.) Although you can get a higher SLA by running two or more VMs, a single VM can be reliable enough for some workloads. For production workloads, however, using two or more VMs for redundancy is recommended.

Availability sets. To protect against localized hardware failures, such as a disk or network switch failing, deploy two or more VMs in an availability set. An availability set is a logical grouping of VMs that allows Azure to understand how your application is built to provide for redundancy and availability. Each virtual machine in your availability set is assigned an **update domain** and a **fault domain** by the underlying Azure platform. Each availability set can be configured with up to three fault domains and twenty update domains. Update domains indicate groups of virtual machines and underlying physical hardware that can be rebooted at the same time. Fault domains define the group of virtual machines that share a common power source and network switch. See Availability sets overview and Availability options for Azure Virtual Machines for more details. 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.

Availability zones. An availability zone is a physically separate zone within an Azure region. Each availability zone has a distinct power source, network, and cooling. Deploying VMs across availability zones helps to protect an application against datacenter-wide failures. Not all regions support availability zones. See What are Availability Zones in Azure? for a list of supported regions and services.

If you are planning to use availability zones in your deployment, first validate that your application architecture and code base can support this configuration. If you are deploying commercial off-the-shelf software, consult with the software vendor and test adequately before deploying into production. An application must be able to maintain state and prevent loss of data during an outage within the configured zone. 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 virtual machines to another Azure region for business continuity and disaster recovery. You can conduct periodic disaster recovery drills to ensure you meet the compliance needs. The VM will be replicated with the specified settings to the selected region so that you can recover your applications in the event of outages in the source region. See Replicate Azure VMs using ASR for more information. You should factor in the RTO and RPO numbers for your solution here and ensure that when testing, the recovery time and recovery point is appropriate for your needs.

Paired regions. To protect an application against a regional outage, you can deploy the application across multiple regions, using Azure Traffic Manager to distribute internet traffic to the different regions. Each Azure region is paired with another region. Together, these form a regional pair. Except for Brazil South, regional pairs are located within the same geography to meet data residency requirements for tax and law enforcement jurisdiction purposes.

When you design a multi-region application, remember that network latency across regions is higher than within a region. For example, if you are replicating a database to enable failover, use synchronous data replication within a region, but asynchronous data replication across regions.

When you select paired regions, ensure both regions have required Azure services. See Products available by region for a list of services by region. It's also critical to select the right deployment topology for disaster recovery, especially if your RPO/RTO requirements are low. To ensure the failover region has enough capacity to support your workload, select either an active/passive (full replica) topology or an active/active topology. Keep in mind these deployment topologies might increase complexity and cost as resources in the secondary region are pre-provisioned and can sit idle. See Deployment topologies for disaster recovery for more information.

	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

	Availability Set	Availability Zone	Azure Site Recovery/Paired region
Network latency	Very low	Low	Mid to high
Virtual network	VNet	VNet	Cross-region VNet peering

Test failures and response strategies

Generally, you can't test resiliency in the same way that you test application functionality (by running unit tests and so on). Instead, you must 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 resiliency of the system during failures, either by triggering actual failures or by simulating them. Here are some common failure scenarios to test:

- Shut down VM instances.
- 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 your 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 why it's important to analyze possible failure points during the design phase. The results of that analysis should be inputs into your 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 is as close to production data as possible. The goal is to see how the application behaves under real-world conditions.

Disaster recovery drills. It is not enough if you have a good disaster recovery plan in place. You need to test it periodically to ensure your recovery plan works fine when it matters. For Azure virtual machines, you can use Azure Site Recovery to replicate and perform disaster recovery drills without impacting production applications or ongoing replication.

With the cross-region restore feature provided by Azure Backup, you can conduct business continuity and disaster recovery (BCDR) drills for audit or compliance purposes with the secondary region data. This enables you to perform a restore of backed up data in the secondary region.

Deploy using reliable process

Once an application is deployed to production, updates are a possible 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.

The crucial point is that manual deployments are prone to error. Therefore, it's recommended to have an automated, idempotent process that you can run on demand, and re-run if something fails.

- To automate provisioning of Azure resources you can use Terraform, Ansible, Chef, Puppet, PowerShell, CLI , or Azure Resource Manager templates.
- Use Azure Automation Desired State Configuration (DSC) to configure VMs. For Linux VMs, you can use Cloud-init.
- You can automate application deployment using Azure DevOps Services or Jenkins.

Two concepts related to resilient deployment are *infrastructure as code* and *immutable infrastructure*.

- **Infrastructure as code** is the practice of using code to provision and configure infrastructure. Infrastructure as code can use a declarative approach or an imperative approach (or a combination of both). Resource Manager templates are an example of a declarative approach. PowerShell scripts are an example of an imperative approach.
- **Immutable infrastructure** is the principle that you shouldn't modify infrastructure after it's deployed to production. Otherwise, you can get into a state where ad hoc changes have been applied and it's unclear what was changed—as well as more difficult to think effectively about the system.

Another question is how to roll out an application update. Azure recommends techniques such as blue-green deployment or canary releases that push updates in highly controlled way to minimize possible impacts from a bad deployment.

- Blue-green deployment is a technique in which an update is deployed into a production environment separate from the live application. After you validate the deployment, switch the traffic routing to the updated version. For example, Azure App Service Web Apps enables this with staging slots.
- Canary releases are like blue-green deployments. Instead of switching all traffic to the updated version, you roll out the update to a small percentage of users, by routing a portion of the traffic to the new deployment. If there is a problem, back off and revert to the old deployment. Otherwise, route more of the traffic to the new version, until it gets 100% of the traffic.

Whatever approach you take, make sure that you can roll back to the last-known-good deployment, in case the new version is not functioning. Also 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 resiliency. If something fails, you need to know that it failed, and you need insights into the cause 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 log files, trying to troubleshoot a problem. Moreover, the number of VM instances is probably not static VMs get added and removed as the application scales in and out, and occasionally an instance may fail and need to be reprovisioned. In addition, a typical cloud application might use multiple data stores (Azure storage, SQL Database, Cosmos DB, Redis cache), and a single user action can span multiple subsystems.

You can think of the monitoring process as a pipeline with several distinct stages:

- **Instrumentation**. The raw data for monitoring comes from a variety of sources, including application logs, operating systems performance metrics, Azure monitoring resources, Azure Service Health and subscriptions and Azure tenants. Most Azure services expose metrics that you can configure to analyze and determine the cause of problems.
- **Collection and storage**. Raw instrumentation data can be held in various locations and with various formats (for example, application trace logs, IIS logs, performance counters). These disparate sources are collected, consolidated, and put into reliable data stores such as Application Insights, Azure Monitor metrics, Service Health, storage accounts and 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, you can search for the data in Application Insights and Log Analytics using Kusto queries. Azure Advisor provides recommendations with a focus on resiliency.
- **Visualization and alerts**. In this stage, telemetry data is presented so that an operator can quickly notice problems or trends. Examples include dashboards or email alerts. With Azure dashboards, you can build a single-pane of glass view of monitoring graphs originating from Application Insights, Log Analytics, Azure Monitor metrics and service health. With Azure Monitor alerts, you can create alerts on service health and resource health.

Monitoring is not the same as failure detection. For example, your application might detect a transient error and retry, resulting in no downtime. But it should also log the retry operation, so that you can monitor the error rate 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. Otherwise, you lose insight where you need it 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 you 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. Otherwise, the logging system itself can cause the application to fail by causing requests to back up as they block while waiting to write a logging event.
- Application logging is not the same as auditing. Auditing may be done for compliance or regulatory reasons. As such, audit records must be complete, and it's not acceptable to drop any while processing transactions. If an application requires auditing, this should be kept separate from diagnostics logging.

When you implement monitoring and diagnostics for a critical application, it is also vital to monitor the health of the periodic backup jobs. Azure Backup offers Backup Center, a single unified management experience in Azure for enterprises to govern, monitor, operate, and analyze backups at scale. As a backup admin, Backup Center gives you a single pane of glass to monitor your jobs and backup inventory daily. You also can use Backup Center to perform your regular operations, such as responding to on-demand backup requests, restoring backups, creating backup policies, and so on. For analyzing historical trends and gaining deeper insights on your backups, Backup Center provides an interface to Backup Reports which uses Azure Monitor Logs and Azure Workbooks.

See Best practices for monitoring cloud applications for more information about monitoring and diagnostics.

Respond to failures

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

- **Alerts**. Monitor your application for warning signs that proactive intervention is needed. For example, if you see that SQL Database or Cosmos DB consistently throttles your application, you might need to increase your database capacity or optimize your queries. In this example, even though the application might handle the throttling errors transparently, your telemetry should still raise an alert so that you can follow up. It is recommended to configure alerts on Azure resources metrics and diagnostics logs against the services limits and quotas thresholds. We recommend to setup alerts on metrics as they are lower latency compared with diagnostics logs. In addition, Azure is able to provide with some out-of-the-box health status through resource health that can help diagnose throttling of Azure services.

- **Failover**. Configure a disaster recovery strategy in your application. The appropriate strategy will depend on your SLAs. For most scenarios, an active-passive implementation is sufficient. For more information, see Backup and disaster recovery for Azure applications. Most Azure services support either manual or automated failover. For example, in an IaaS application, use Azure Site Recovery for the web and logic tiers and SQL Always On 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, you can simulate an outage by shutting down or removing services.

- **Data consistency check**. If a failure happens 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 is initiated by Microsoft. For some services, Microsoft decides when to perform the failover. Microsoft might 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, Geo-redundant storage and Key Vault follow this model.

- **Restoring from backup**. In case of accidental deletion or corruption of data, it is required to revert the data to a previous known state using backups. Azure Backup helps restore the Azure Virtual Machines quickly from recent backups using instant restore feature thus eliminating need to transfer backup from vault storage to customer's subscription there by lowering RTO. It also offers various restore choices like restoring to create new virtual

machine or restore the disks or just specific file/folder that fits different needs. By configuring backups on Azure Blobs, Azure Disks, Azure File Shares, you also get the ability to restore data stored by your application on various storage solutions on Azure. Databases are critical to many applications. With Azure Backup, you can gain point in time recovery of the SQL Server and SAP HANA databases running on Azure Virtual Machines providing RPO as low as few minutes.

Example resiliency design patterns

This section will focus on resiliency design best practices for various application deployments with varied resiliency requirements. Typically, customers classify the applications into various categories or tiers based on their resiliency requirements as discussed in the resiliency requirements section. In this section, we will take an example application from each category and discuss how you can design that application to be resilient from various types of failures.

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

The first category of the applications will be the one with less stringent availability requirements. The disaster recovery requirements are also lower, with an acceptable data loss (RPO) of 24 hours and an acceptable downtime (RTO) of 3 days. Tier 4 applications can be internal applications, such as tooling applications, build servers, project document share website, and so on.

A multi-tier web application in this category can be deployed within an Azure region as single instance VM for each tier. If you want an explicit SLA guarantee at the VM level, you can use premium storage for your VMs. It is recommended to use premium storage for database VM if the application is relatively important within this category after doing the trade-off with premium storage cost. Azure is the first and only public cloud to give explicit SLA on single instance VMs. The databases can be backed up using any backup software. For example, use Azure Backup and configure database backups for SQL servers and SAP HANA running on Azure VMs. You also can keep your Azure Resource Manager (ARM) templates pre-created for your VMs so that you can redeploy VMs if there is an issue with the single instance VM in any tier in that region. For database VM, you can use database backup copy to recreate databases.

Azure Backup helps protect on-premises workloads, Azure Virtual Machines, SQL Server, and SAP HANA Databases running on Azure Virtual Machines, Azure Managed Disks, Azure Blobs, Azure File Shares, and more. It offers application consistent backups for workloads running on Windows and Linux Azure Virtual Machine with ability to restore entire Virtual Machine or Disks or specific File/Folders. It offers instant restore feature for the backups stored in Operational tier with ability to perform quick restore and meet lower RTO requirements. To protect against ransomware attacks, it copies backups to vault storage tier with additional abilities like soft delete and encrypting backup data with customer managed key. Based on vault storage redundancy support, backups can be replicated to ZRS / GRS giving additional backup data resiliency options with ability to restore to a secondary paired region using the cross-region restore feature in the

event of regional failure or while conducting audit drill. To optimize backup storage cost, it offers long terms retention of backups in low-cost storage namely Archive storage tier eliminating need for tape drives.

You can use Azure Backup on your single instance VMs to protect your data and test backups using restore feature of Azure Backup. If there is any data or VM level corruption, you can recover the file, folder, disk or VM using restore capabilities of Azure Backup. With Azure Backup, you can also protect your application data stored across Azure Blobs, Azure Disks, and Azure File Shares providing recovery against data corruption and or loss.

For disaster recovery during regional failure scenario, you can consider replicating only the database VM with Azure Site Recovery. The web and app tier VMs can be redeployed in another Azure region using ARM templates if they are stateless or can be recovered from the backup copy. Note that the recovery time could be high for this approach. If you can take higher trade off on cost, then it is recommended to replicate the VMs across all tiers to another region. Azure Site Recovery does not require running additional VM instances in the disaster recovery region. The VMs are created only the user performs failover operations.

You can monitor the health of the web application by using an automation script that periodically checks if 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, however.

You can have a pre-created script to monitor the simple health metrics of the VMs to see if there is any issue with the VM. You can also troubleshoot any issues by checking the VM health metrics in portal. You can also check for component health such as CPU, Memory disk to see if there are any potential load issues. If you consistently see issues with components such as CPU or RAM, consider increasing the VM to a higher size or consider scaling the application by deploying more VMs at each tier.

You can deploy application software updates on the VMs using automation scripts during a weekend maintenance window. Make sure you have the automation script to roll back if any issues are hit during the deployment process.

Failure type	**Resilience strategy**
Hardware failure	• Use ready-to-use templates to deploy another instance using the backup copies (if required). • Test your templates by deploying VMs into a test subnet or a test virtual network.

Datacenter failure	• Use ready to use templates to deploy another instance using the backup copies (if required) in another zone. • Test your 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 recovery plans. • Perform disaster recovery failover in the event of an extended outage in source region. • Azure Backup's Cross Region Restore (CRR) lets you restore Azure VMs in a secondary paired region. You can restore your data in the secondary region anytime, during partial or full outages, or at the time you choose.
Heavy load	• Use monitoring tools to identify any 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 the Azure Virtual Machine, protect the SQL Server and SAP Databases running on Azure VM and to back up the data stored in Azure Disks, Azure Blobs and Azure File Shares and restore them during data loss/corruption.
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. • Also, use Azure Backup's on-demand backup feature to take backup of protected resources before an application deployment or upgrade activity. Use it restore quickly to the previous known state, in case of deployment failure.

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

Tier 3 applications are those that will require high application SLA that are critical to the business, but acceptable to have some downtime. Also, the disaster recovery RPO and RTO requirements can be as long as a few hours. These can be internal applications such as expense management or travel management applications that can have some impact if the applications are down for a few hours but will not result in significant revenue impact. Lower revenue-generating, customer-facing applications can also be part of this category.

You can build redundancy for the applications in this category by deploying as VMs (at least two) at each tier part of an availability set. Availability sets ensure that the VMs are placed in different fault domains and that guarantees hardware failures such as cluster or rack failure does not impact the end application. If you keep two or more VMs in an availability set, you get 99.95%

availability for each tier. This will help you in getting the overall composite SLA of the application to within 99.9%. For the database VMs, you can use in-built synchronous replication to get high availability as well as avoid data loss. For example, you can use SQL Always On availability groups with asynchronous replication for the SQL databases.

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

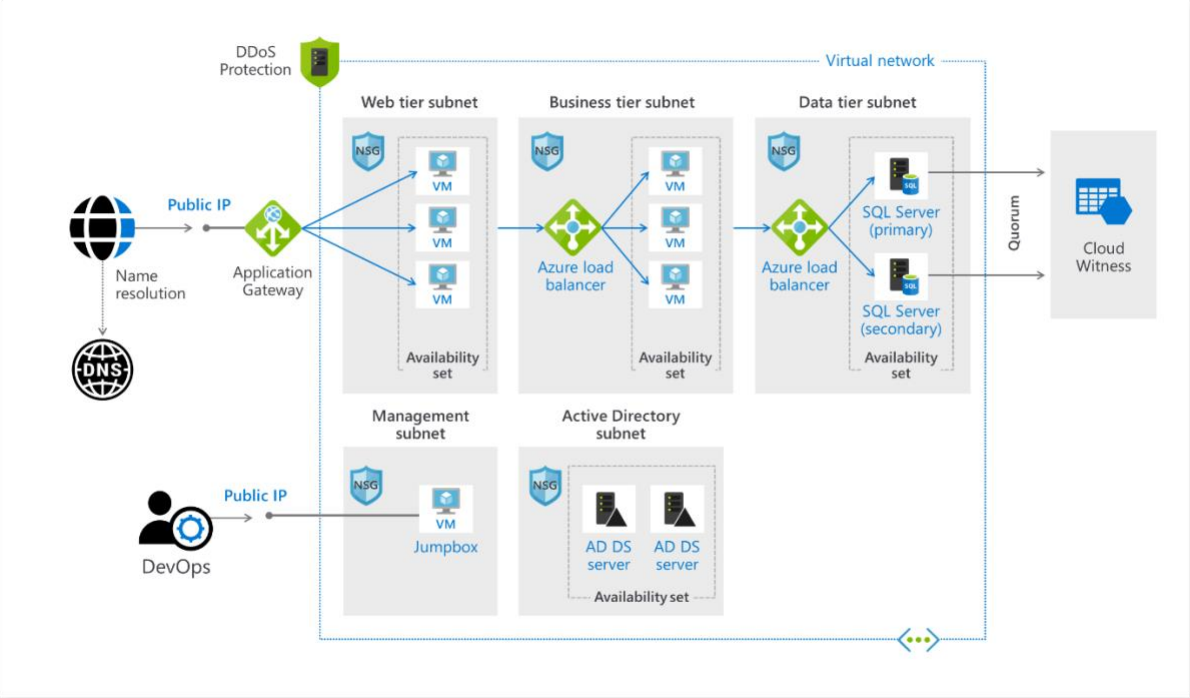
Azure Backup helps restore the Azure Virtual Machines quickly from recent backups using instant restore feature thus eliminating need to transfer backup from vault storage to customer's subscription there by lowering RTO. It also offers various restore choices like restoring to create new virtual machine or restore the disks or just specific file/folder that fits different needs. By configuring backups on Azure Blobs, Azure Disks, Azure File Shares, you also get the ability to restore data stored by your application on various storage solutions on Azure. Databases are critical to many applications. With Azure Backup, you can gain point in time recovery of the SQL Server (with support for backing up SQL Server always on availability groups) and SAP HANA databases running on Azure Virtual Machines providing RPO as low as 15 minutes.

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

You can monitor the health of the web application by using an automation script that periodically checks if 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.

You can set up a pre-created script to monitor the simple health metrics of the VMs to see if there is any issue with the VM. You also can troubleshoot any issues by checking the VM health metrics in portal. You also can check for component health such as CPU and Memory disk to see if there are any potential load issues. If you consistently see issues with components such as CPU or RAM, consider increasing the VM to a higher size or consider scaling the application by deploying more VMs at each tier. You should also monitor advanced metrics for VM health as well as activities such as database failover if you are using asynchronous replication.

You can deploy application software updates on the VMs using automation scripts during a weekend maintenance window. You should ensure you have the automation script to roll back if any issues happen during the deployment process.

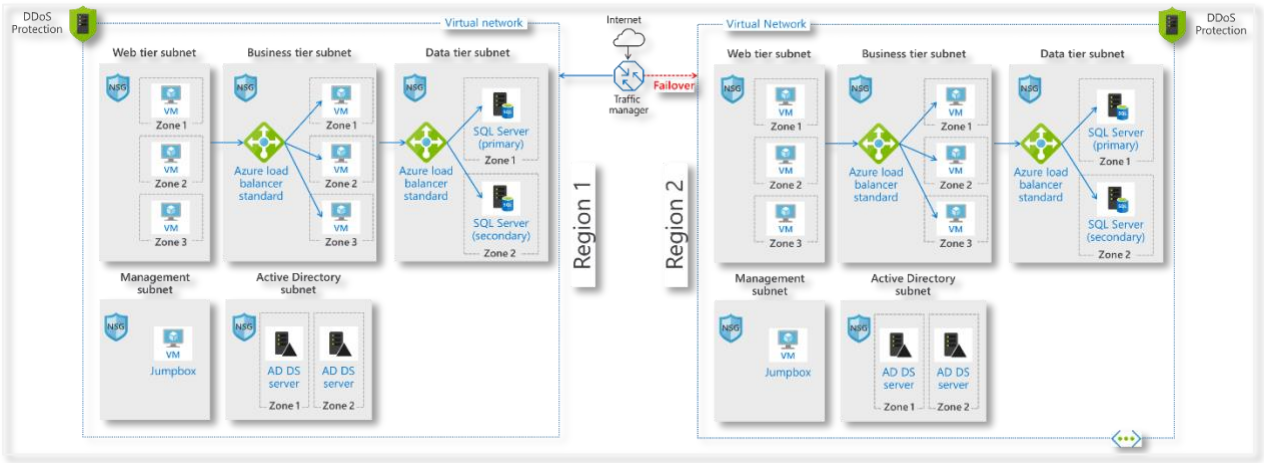


Failure type	Resiliency strategy
Hardware failure	• Build redundancy by deploying two or more instances in an availability set within a datacenter.
Datacenter failure	• Use ready to use templates to deploy another instance using the backup copies (if required) in another availability zone. • Test your 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 recovery plans. • Perform disaster recovery failover in the event of an extended outage in source region. • Azure Backup's Cross Region Restore (CRR) lets you restore Azure VMs in a secondary paired region. You can restore your data in the secondary region anytime, during partial or full outages, or at the time you choose.
Heavy load	• Use monitoring tools to identify any 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 the Azure Virtual Machine, protect the SQL Server and SAP Databases running on Azure VM and to back up the data stored in Azure Disks, Azure Blobs and Azure File Shares and restore them during data loss/corruption.
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. • Additionally, use Azure Backup's on-demand backup feature to take backup of protected resources before an application deployment or upgrade activity. Use it restore quickly to the previous known state, in case of deployment failure.

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

This category of applications is business critical and can have significant impact on revenues if there is downtime. These can be external customer facing e-commerce websites, content streaming platform, financial transaction handling applications. The application should be highly available with resilience to all component failures.



Failure type	Resiliency 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 the database VM. • Test the disaster recovery using Test failover and Azure Site Recovery recovery plans. • Perform disaster recovery failover in the event of an extended outage in source region. • Azure Backup's Cross Region Restore (CRR) lets you restore Azure VMs in a secondary paired region. You can restore your data in the secondary region anytime, during partial or full outages, or at the time you choose.
Heavy load	• Provision enough capacity into the application. • Use tools to monitor the load and add more instances automatically using scripts if the threshold is hit (say, 70%).
Accidental data deletion or corruption	• Use Azure Backup to back up the Azure Virtual Machine, protect the SQL Server and SAP Databases running on Azure VM and to back up the data stored in Azure Disks, Azure Blobs and Azure File Shares and restore them during data loss/corruption.
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. • Additionally, use Azure Backup's on-demand backup feature to take backup of protected resources before an application deployment or upgrade activity. Use it restore quickly to the previous known state, in case of deployment failure.

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

Tier 1 applications are business and mission critical, like Tier 2. They have stricter requirements for data loss as well as the recovery time. More than a few minutes of data loss can significantly impact the revenues and business. Customer facing applications such as order processing systems and banking applications fall into this category.

Failure type	**Resiliency 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 in web tier and middle tier. • Use native replication technologies such as SQL Always On. • Test the disaster recovery of the complete application including SQL Always On failover using Azure Site Recovery recovery plans and Test failover capabilities. • Perform disaster recovery failover in the event of an extended outage in source region,
Heavy load	• Provision enough capacity into the application. • Use tools to monitor the load and add more instances automatically using scripts if the threshold is hit (say, 70%).
Accidental data deletion or corruption	• Use Azure Backup to back up the Azure Virtual Machine, protect the SQL Server and SAP Databases running on Azure VM and to back up the data stored in Azure Disks, Azure Blobs and Azure File Shares and restore them during data loss/corruption.
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

	• Additionally, use Azure Backup's on-demand backup feature to take backup of protected resources before an application deployment or upgrade activity. Use it restore quickly to the previous known state, in case of deployment failure.

Conclusion

In this document, we discussed the importance of resiliency when designing applications on Azure and the process of designing and deploying highly resilient applications. Azure continuously is making improvements to platform reliability by investing in the foundation aspects. Azure also offers inbuilt services that can be leveraged to design and deploy resilient applications. Customers should understand the shared responsibility model when deploying applications in the cloud and ensure they design the application components to be resilient to failures.

The example resiliency design patterns discussed in this document can be helpful in your thinking and planning when you start running your production applications on Azure. For more information and guidance, see the resiliency resources page at the Microsoft Azure Well-Architected Framework.