Designing reliable Azure applications

04/10/2019 • 11 minutes to read • Contributors

In this article

Building a reliable application in the cloud is different from traditional application development. While historically you may have purchased higher-end hardware to scale up, in a cloud environment you scale out instead of up. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component.

Reliable applications are:

- **Resilient** and recover gracefully from failures, and they continue to function with minimal downtime and data loss before full recovery.
- **Highly available (HA)** and run as designed in a healthy state with no significant downtime.

Understanding how these elements work together — and how they affect cost — is essential to building a reliable application. It can help you determine how much downtime is acceptable, the potential cost to your business, and which functions are necessary during a recovery.

This article provides a brief overview of building reliability into each step of the Azure application design process. Each section includes a link to an in-depth article on how to integrate reliability into that specific step in the process. If you're looking for reliability considerations for individual Azure services, review the Resiliency checklist for specific Azure services.

Build for reliability

This section describes six steps for building a reliable Azure application. Each step links to a section that further defines the process and terms.

1. **Define requirements.** Develop availability and recovery requirements based on decomposed workloads and business needs.
2. **Use architectural best practices.** Follow proven practices, identify possible failure points in the architecture, and determine how the application will respond to failure.
3. **Test with simulations and forced failovers.** Simulate faults, trigger forced failovers, and test detection and recovery from these failures.
4. **Deploy the application consistently.** Release to production using reliable and repeatable processes.
5. **Monitor application health.** Detect failures, monitor indicators of potential failures, and gauge the health of your applications.
6. **Respond to failures and disasters.** Identify when a failure occurs, and determine how to address it based on established strategies.

Define requirements

Identify your business needs, and build your reliability plan to address them. Consider the following:

- **Identify workloads and usage.** A *workload* is a distinct capability or task that is logically separated from other tasks, in terms of business logic and data storage requirements. Each workload has different requirements for availability, scalability, data consistency, and disaster recovery.
- **Plan for usage patterns.** *Usage patterns* also play a role in requirements. Identify differences in requirements during critical and non-critical periods. For example, a tax-filing application can't fail during a filing deadline. To ensure uptime, plan redundancy across several regions in case one fails. Conversely, to minimize costs during non-critical periods, you can run your application in a single region.
- **Establish availability metrics — *mean time to recovery* (MTTR) and *mean time between failures* (MTBF).** MTTR is the average time it takes to restore a component after a failure. MTBF is how long a component can reasonably expect to last between outages. Use these measures to determine where to add redundancy and to determine service-level agreements (SLAs) for customers.
- **Establish recovery metrics — recovery time objective and recovery point objective (RPO).** *RTO* is the maximum acceptable time an application can be unavailable after an incident. *RPO* is the maximum duration of data loss that is acceptable during a disaster. To derive these values, conduct a risk assessment and make sure you understand the cost and risk of downtime or data loss in your organization.

  > **Note**
  >
  > If the MTTR of *any* critical component in a highly available setup exceeds the system RTO, a failure in the system might cause an unacceptable business disruption. That is, you can't restore the system within the defined RTO.

- **Determine workload availability targets.** To ensure that application architecture meets your business requirements, define target SLAs for each workload. Account for the cost and complexity of meeting availability requirements, in addition to application dependencies.
- **Understand service-level agreements.** In Azure, the SLA describes the Microsoft commitments for uptime and connectivity. If the SLA for a particular service is 99.9 percent, you should expect the service to be available 99.9 percent of the time.

  Define your own target SLAs for each workload in your solution, so you can determine 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.

For more information about developing requirements for reliable applications, see Developing requirements for resilient Azure applications.

Use architectural best practices

During the architectural phase, focus on implementing practices that meet your business requirements, identify failure points, and minimize the scope of failures.

- **Perform a failure mode analysis (FMA).** FMA builds resiliency into an application early in the design stage. It helps you identify the types of failures your application might experience, the potential effects of each, and possible recovery strategies.
- **Create a redundancy plan.** The level of redundancy required for each workload depends on your business needs and factors into the overall cost of your application.

- **Design for scalability.** A cloud application must be able to scale to accommodate changes in usage. Begin with discrete components, and design the application to respond automatically to load changes whenever possible. Keep scaling limits in mind during design so you can expand easily in the future.

- **Plan for subscription and service requirements.** You might need additional subscriptions to provision enough resources to meet your business requirements for storage, connections, throughput, and more.

- **Use load-balancing to distribute requests.** Load-balancing distributes your application's requests to healthy service instances by removing unhealthy instances from rotation.

- **Implement resiliency strategies.** *Resiliency* is the ability of a system to recover from failures and continue to function. Implement resiliency design patterns, such as isolating critical resources, using compensating transactions, and performing asynchronous operations whenever possible.

- **Build availability requirements into your design.** *Availability* is the proportion of time your system is functional and working. Take steps to ensure that application availability conforms to your service-level agreement. For example, avoid single points of failure, decompose workloads by service-level objective, and throttle high-volume users.

- **Manage your data.** How you store, back up, and replicate data is critical.
  - **Choose replication methods for your application data.** Your application data is stored in various data stores and might have different availability requirements. Evaluate the replication methods and locations for each type of data store to ensure that they satisfy your requirements.
  - **Document and test your failover and failback processes.** Clearly document instructions to fail over to a new data store, and test them regularly to make sure they are accurate and easy to follow.
  - **Protect your data.** Back up and validate data regularly, and make sure no single user account has access to both production and backup data.
  - **Plan for data recovery.** Make sure that your backup and replication strategy provides for data recovery times that meet your service-level requirements. Account for all types of data your application uses, including reference data and databases.

For more information about architecting reliable applications, see Architecting Azure applications for resiliency and availability.

Test with simulations and forced failovers

Testing for reliability requires measuring how the end-to-end workload performs under failure conditions that only occur intermittently.

- **Test for common failure scenarios by triggering actual failures or by simulating them.** Use fault injection testing to test common scenarios (including combinations of failures) and recovery time.
- **Identify failures that occur only under load.** Test for peak load, using production data or synthetic data that is as close to production data as possible, to see how the application behaves under real-world conditions.
- **Run disaster recovery drills.** Have a disaster recovery plan in place, and test it periodically to make sure it works.
- **Perform failover and failback testing.** Ensure that your application's dependent services fail over and fail back in the correct order.
- **Run simulation tests.** Testing real-life scenarios can highlight issues that need to be addressed. Scenarios should be controllable and non-disruptive to the business. Inform management of simulation testing plans.
- **Test health probes.** Configure health probes for load balancers and traffic managers to check critical system components. Test them to make sure that they respond appropriately.
- **Test monitoring systems.** Be sure that monitoring systems are reliably reporting critical information and accurate data to help identify potential failures.
- **Include third-party services in test scenarios.** Test possible points of failure due to third-party service disruption, in addition to recovery.

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

For more information about testing for application reliability, see Testing Azure applications for resiliency and availability.

Deploy the application consistently

Deployment includes provisioning Azure resources, deploying application code, and applying configuration settings. An update may involve all three tasks or a subset of them.

After an application is deployed to production, updates are a possible source of errors. Minimize errors with predictable and repeatable deployment processes.

- **Automate your application deployment process.** Automate as many processes as possible.
- **Design your release process to maximize availability.** If your release process requires services to go offline during deployment, your application is unavailable until they come back online. Take advantage of platform staging and production features. Use blue-green or canary releases to deploy updates, so if a failure occurs, you can quickly roll back the update.
- **Have a rollback plan for deployment.** Design a rollback process to return to a last known good version and to minimize downtime if a deployment fails.
- **Log and audit deployments.** If you use staged deployment techniques, more than one version of your application is running in production. Implement a robust logging strategy to capture as much version-specific information as possible.
- **Document the application release process.** Clearly define and document your release process, and ensure that it's available to the entire operations team.

For more information about application reliability and deployment, see Deploying Azure applications for resiliency and availability.

Monitor application health

Implement best practices for monitoring and alerts in your application so you can detect failures and alert an operator to fix them.

- **Implement health probes and check functions.** Run them regularly from outside the application to identify degradation of application health and performance.

- **Check long-running workflows.** Catching issues early can minimize the need to roll back the entire workflow or to execute multiple compensating transactions.

- **Maintain application logs.**
  - Log applications in production and at service boundaries.
  - Use semantic and asynchronous logging.
  - Separate application logs from audit logs.

- **Measure remote call statistics, and share the data with the application team.** To give your operations team an instantaneous view into application health, summarize remote call metrics, such as latency, throughput, and errors in the 99 and 95 percentiles. Perform statistical analysis on the metrics to uncover errors that occur within each percentile.

- **Track transient exceptions and retries over an appropriate time frame.** A trend of increasing exceptions over time indicates that the service is having an issue and may fail.

- **Set up an early warning system.** Identify the key performance indicators (KPIs) of an application's health, such as transient exceptions and remote call latency, and set appropriate threshold values for each of them. Send an alert to operations when the threshold value is reached.

- **Operate within Azure subscription limits.** Azure subscriptions have limits on certain resource types, such as the number of resource groups, cores, and storage accounts. Watch your use of resource types.

- **Monitor third-party services.** Log your invocations and correlate them with your application's health and diagnostic logging using a unique identifier.

- **Train multiple operators to monitor the application and to perform manual recovery steps.** Make sure there is always at least one trained operator active.

For more information about monitoring for application reliability, see Monitoring Azure application health.

Respond to failures and disasters

Create a recovery plan, and make sure that it covers data restoration, network outages, dependent service failures, and region-wide service disruptions. Consider your VMs, storage, databases, and other Azure platform services in your recovery strategy.

- **Plan for Azure support interactions.** Before the need arises, establish a process for contacting Azure support.
- **Document and test your disaster recovery plan.** Write a disaster recovery plan that reflects the business impact of application failures. Automate the recovery process as much as possible, and document any manual steps. Regularly test your disaster recovery process to validate and improve the plan.
- **Fail over manually when required.** Some systems can't fail over automatically and require a manual failover. If an application fails over to a secondary region, perform an operational readiness test. Verify that the primary region is healthy and ready to receive traffic again before failing back. Determine what the reduced application functionality is and how the app informs users of temporary problems.
- **Prepare for application failure.** Prepare for a range of failures, including faults that are handled automatically, those that result in reduced functionality, and those that cause the application to become unavailable. The application should inform users of temporary issues.
- **Recover from data corruption.** If a failure happens in a data store, check for data inconsistencies when the store becomes available again, especially if the data was replicated. Restore corrupt data from a backup.
- **Recover from a network outage.** You might be able to use cached data to run locally with reduced application functionality. If not, consider application downtime or fail over to another region. Store your data in an alternate location until connectivity is restored.
- **Recover from a dependent service failure.** Determine which functionality is still available and how the application should respond.
- **Recover from a region-wide service disruption.** Region-wide service disruptions are uncommon, but you should have a strategy to address them, especially for critical applications. You might be able to redeploy the application to another region or redistribute traffic.

For more information about responding to failures and disaster recovery, see Failure and disaster recovery for Azure applications.

Next steps

Develop requirements for resilient applications