Developing requirements for resilient Azure applications

04/10/2019 • 6 minutes to read • Contributors

In this article

Building *resiliency* (recovering from failures) and *availability* (running in a healthy state without significant downtime) into your apps begins with gathering requirements. For example, how much downtime is acceptable? How much does potential downtime cost your business? What are your customer's availability requirements? How much do you invest in making your application highly available? What is the risk versus the cost?

Identify distinct workloads

Cloud solutions typically consist of multiple application *workloads*. A workload is a distinct capability or task that is logically separated from other tasks in terms of business logic and data storage requirements. For example, an e-commerce app might have the following workloads:

- Browse and search a product catalog.
- Create and track orders.
- View recommendations.

Each workload has different requirements for availability, scalability, data consistency, and disaster recovery. Make your business decisions by balancing cost versus risk for each workload.

Also decompose workloads by service-level objective. If a service is composed of critical and less-critical workloads, manage them differently and specify the service features and number of instances needed to meet their availability requirements.

Plan for usage patterns

Identify differences in requirements during critical and non-critical periods. Are there certain critical periods when the system must be available? For example, a tax-filing application can't fail during a filing deadline and a video streaming service shouldn't lag during a live event. In these situations, weigh the cost against the risk.

- To ensure uptime and meet service-level agreements (SLAs) in critical periods, plan redundancy across several regions in case one fails, even if it costs more.
- Conversely, during non-critical periods, run your application in a single region to minimize costs.
- In some cases, you can mitigate additional expenses by using modern serverless techniques that have consumption-based billing.

Establish availability and recovery metrics

Create baseline numbers for two sets of metrics as part of the requirements process. The first set helps you determine where to add redundancy to cloud services and which SLAs to provide to customers. The second set helps you plan your disaster recovery.

Availability metrics

Use these measures to plan for redundancy and determine customer SLAs.

- **Mean time to recover (MTTR)** is the average time it takes to restore a component after a failure.
- **Mean time between failures (MTBF)** is the how long a component can reasonably expect to last between outages.

Recovery metrics

Derive these values by conducting a risk assessment, and make sure you understand the cost and risk of downtime and data loss. These are nonfunctional requirements of a system and should be dictated by business requirements.

- **Recovery time objective (RTO)** is the maximum acceptable time an application is unavailable after an incident.
- **Recovery point objective (RPO)** is the maximum duration of data loss that's acceptable during a disaster.

If the MTTR value 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

Define your own target SLAs for each workload in your solution so you can determine whether the architecture meets the business requirements.

Consider cost and complexity

Everything else being equal, higher availability is better. But as you strive for more nines, the cost and complexity grow. An uptime of 99.99% translates to about five minutes of total downtime per month. Is it worth the additional complexity and cost to reach five nines? The answer depends on the business requirements.

Here are some other considerations when defining an SLA:

- To achieve four nines (99.99%), you can't rely on manual intervention 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 that your SLA is measured against. The smaller the window, the tighter the tolerances. It doesn't make sense to define your SLA in terms of hourly or daily uptime.
- Consider the MTBF and MTTR measurements. The higher your SLA, the less frequently the service can go down and the quicker the service must recover.
- Get agreement from your customers for the availability targets of each piece of your application, and document it. Otherwise, your design may not meet the customers' expectations.

Identify dependencies

Perform dependency-mapping exercises to identify internal and external dependencies. Examples include dependencies relating to security or identity, such as Active Directory, or third-party services such as a payment provider or e-mail messaging service.

Pay particular attention to external dependencies that might be a single point of failure or cause bottlenecks. If a workload requires 99.99% uptime but depends on a service with a 99.9% SLA, that service can't be a single point of

failure in the system. One remedy is to have a fallback path in case the service fails. Alternatively, take other measures to recover from a failure in that service.

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

Understand service-level agreements

In Azure, the Service Level Agreement describes Microsoft's commitments for uptime and connectivity. If the SLA for a particular service is 99.9%, you should expect the service to be available 99.9% of the time. Different services have different SLAs.

The Azure SLA also includes provisions for obtaining a service credit if the SLA is not met, along with specific definitions of *availability* for each service. That aspect of the SLA acts as an enforcement policy.

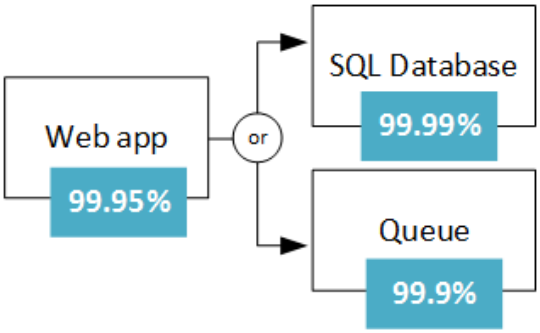
Composite SLAs

Composite SLAs involve multiple services supporting an application, each with differing levels of availability. For example, 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. 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.

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%

There are tradeoffs to this approach. The application logic is more complex, you are paying for the queue, and you need to consider data consistency issues.

SLAs for multiregion deployments

SLAs for multiregion deployments involve a high-availability technique to deploy the application in more than one region and use Azure Traffic Manager to fail over if the application fails in one region.

The composite SLA for a multiregion deployment is calculated as follows:

- *N* is the composite SLA for the application deployed in one region.
- *R* is the number of regions where the application is deployed.

The expected chance that the application fails 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 − (1 − 0.9995) ^ 2) = 99.999975%
- The combined SLA for four regions = (1 − (1 − 0.9995) ^ 4) = 99.999999%

The SLA for Traffic Manager is also a factor. Failing over is not instantaneous in active-passive configurations, which can result in downtime during a failover. See Traffic Manager endpoint monitoring and failover.

Next steps

Architect for resiliency and availability