High availability

High availability is one of the major architectural concerns for any architect. It forms one of the core non-functional technical requirements for any serious service and its deployment. High availability refers to the feature of a service or application that keeps it operational on a continuous basis, meeting or surpassing its promised defined **service level agreement** (**SLA**). Users are promised a certain SLA based on service type. The service should be available for consumption based on its SLA. For example, an SLA can have 99% availability for an application for the entire year. This means it should be available for consumption by users for 361.35 days. If it becomes less than this, this constitutes a breach of the SLA. Most mission-critical applications define their high-availability SLA with five nines for a year. This means the application should be up, running, and available throughout the year, but it can only be down and unavailable for 5.2 hours.

It is important to note here that high availability is defined in terms of time—that is, yearly, monthly, weekly, and so on, and it could even be a combination of these.

A service or application is made up of multiple components and these components are deployed on separate tiers and layers. Moreover, it is deployed on an operating system and hosted on a physical machine or VM. It consumes network and storage services for various purposes. It might even be dependent on external systems. For these services or applications to be highly available, it is important that networks, storage, operating systems, VMs or physical machines, and each component of the application is designed with the SLA and high availability in mind. A definite application life cycle process used to ensure high availability should be baked from the start of application planning until its introduction to operations. This also involves introducing redundancy. Redundant resources should be included in the overall application and deployment architecture to ensure that if one goes down, the other takes over and serves the requests of the customer.

SLA

SLA is defined as a service-level agreement. This is an agreement between two or more parties, where one is the customer, and the others are service providers. Particular aspects of the service - quality, availability, and responsibilities - are agreed between the service provider and the service user. The most common component of the SLA is that the services should be provided to the customer as agreed upon in the contract.

Factors affecting high availability

Planned maintenance, unplanned maintenance, and application deployment architecture are the major factors affecting the high availability of an application. We will be looking into each of these factors in the following sections.

Planned maintenance

Planned maintenance refers to the process of keeping the application and its surrounding ecosystem—comprising platforms, frameworks, software, the operating system, and host and guest drivers—up to date with the latest stable releases. It is important to patch software, drivers, and operating systems with the latest updates, since this helps in keeping the environment healthy from a security, performance, and future-ready perspective. Not upgrading an environment is not an option and is a fact of life. Even applications should be upgraded with enhanced functionality, bugs, and hot fixes. Every organization plans for environment and application upgrades, and typically these involve shutting down and restarting the application and operating system. It might also involve starting the physical host operating system, which, in turn, will reboot all guest VMs running on top of it. In Microsoft Azure, you can manage, get notifications and view planned maintenance windows for VMs – please refer to <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/maintenance-notifications> for more detailed information

Unplanned maintenance

As the name suggests, unplanned maintenance refers to maintenance that cannot be planned and is ad hoc in nature. It refers to hardware failures such as storage corruption, network or router failure, power loss, and a host of other failures, due to hardware and software. Bugs in the underlying platform that bring the application down are also part of unplanned maintenance.

Application deployment architecture

Application architecture plays a crucial role in ensuring the high availability of an application. An application whose components are deployed on a single machine is not highly available. When the machine reboots, the application is not available to its users. Similarly, depending on a single instance of a resource can become a single point of failure from a high-availability point of view. Each component of an application should be designed so that it can be deployed on multiple machines, and redundancy should not be a bottleneck. Some software provides features related to high availability and is not dependent on host operating systems or other third-party tools. SQL server availability groups is an example of such features.

High availability versus scalability

High availability is different to scalability, although both are serious architectural concerns. Scalability refers to the flexibility and elasticity to add more resources or reduce resources to existing deployment to accommodate more users than normal without comprising application performance. Scalability indirectly helps in making an application highly available. However, it does not mean scalability eventually leads to high availability. High availability is an architectural concern that is not dependent on the number of users, while scalability rules are determined by a number of users consuming the service. High availability could be a requirement even if there were very few users. High availability is about services being present and operational as and when users demand its consumption. It is a function of consumption based on the SLA.

High availability versus disaster recovery

High availability is again different from disaster recovery; however, the difference can be very subtle. High availability is a function of the application being in a consumable state as and when the user asks for it. So, it is designed for operations that come before a disaster, while disaster recovery is a function that comes into the picture after a disaster. Disaster recovery refers to the architecture implementation through which services are up and running after a disaster, while high availability takes care of availability prior to a disaster. Disaster recovery includes data backup and archived and dormant servers across continents, while high availability consists of load balancers, distribution of load, and active-passive and active-active redundancy.

[Previous Section](https://subscription.packtpub.com/book/virtualization_and_cloud/9781789614503/2)[Next Section](https://subscription.packtpub.com/book/virtualization_and_cloud/9781789614503/2/ch02lvl1sec20/azure-high-availability)

## Azure high availability

Achieving high availability and meeting high SLA requirements is tough. Azure provides lots of features that enable high availability for applications, from the host and guest operating system to applications using its PaaS. Architects can use these features to get high availability in their applications using configuration instead of building these features from scratch or being dependent on third-party tools.

In this section, we will look at the features and capabilities provided by Azure to make applications highly available. Before we get into the architectural and configuration details, it is important to understand Azure's high-availability, related concepts.

### Concepts

The fundamental concepts provided by Azure to attain high availability are listed here:

* Availability sets
* Fault domain
* Update domain
* Availability zones

#### Availability sets

High availability in Azure is primarily achieved through **redundancy**. Redundancy means that there is more than one resource instance, such that in the event of a failure of a resource, the other takes over. But just having more similar resources does not make them highly available. For example, having more than one VM does not make these VMs highly available. Azure provides a resource known as an availability set, and any VM that is associated with it becomes highly available. All VMs in the availability set become highly available because they are placed on separate physical racks in the Azure data center and each VM is updated one at a time, instead of all at the same time. Availability sets provide a fault domain and updates domain to achieve this and this is discussed in the next section. In short, availability sets provide redundancy at a data center level, similar to locally redundant storage.

It is important to note that availability sets provide high availability within a data center. If the entire data center is down, then the availability of the application will be impacted. To ensure that applications are still available even when a data center goes down, Azure introduced a new feature known as **availability zones**, which we will understand in a little while.

#### Fault domain

When a VM is provisioned and assigned to an availability set, it is assigned a fault domain. With **Azure Resource Manager** (**ARM**), each availability set has two or three fault domains by default, depending on the Azure regions. Some provide two, while others provide three, fault domains in an availability set. Fault domains are non-configurable by users. When multiple VMs are created, they are placed on separate fault domains. If more than five VMs are provisioned on an availability set, they are placed in a round-robin fashion on five fault domains. Fault domains are related to physical racks in the Azure data center. Fault domains provide high availability from unplanned maintenance due to hardware, power, and network failure. Since a single VM is only placed on a rack, other VMs continue running in case the rack in consideration snaps off.

#### Update domain

A fault domain takes care of unplanned maintenance and an update domain handles planned maintenance. Each VM is also assigned an update domain. There can be as many as 20 update domains in a single availability set. Update domains are non-configurable by users. When multiple VMs are created, they are placed on separate update domains. If more than 20 VMs are provisioned on an availability set, they are placed in a round-robin fashion on these update domains. Update domains take care of planned maintenance.

#### Availability zones

This is a relatively new concept introduced in Azure and is very similar to zone redundancy for storage accounts. Availability zones provide high availability within a region by placing VM instances on separate data centers within a region. Availability Zones are applicable to VMs, managed disks, VM scale sets, and load balancers. This constituted a gap in Azure for a long time and it was removed recently, from a computing high-availability perspective, which was removed recently.

Each Azure region comprises multiple data centers. Some regions have more data centers, while some could have less; however, there are at least two data centers in every region. These data centers are known as zones. Deploying VMs in an availability zone ensures that those VMs are in different data centers and obviously are on different racks and networks, and so on. These data centers in a region relate to high-speed networks and there is no lag in communication between these VMs.

### Note

More information about Availability Zones is available at [https://docs.microsoft.com/en-us/azure/availability-zones/az-overview.higher.](https://docs.microsoft.com/en-us/azure/availability-zones/az-overview)

If an application needs higher availability and wants to ensure that it is available even if an entire Azure region is down, the next rung of the ladder for availability is the traffic manager feature, which will be discussed later in this chapter.

### Load balancing

Load balancing, as the name suggests, refers to the process of balancing the load among VMs and applications. With one VM, there is no need for the load balancer because the entire load is on a single VM and there is no other VM to share the load. However, with multiple VMs containing the same application and service, it is possible to distribute the load among them through load balancing. Azure provides a couple of resources for enabling load balancing, which are as follows:

* **Load balancers**: The Azure load balancer helps in designing solutions with high availability. Within the TCP stack, it is a layer 4 transport-level load balancer. It is a layer 4 load balancer that distributes incoming traffic among healthy instances of services defined in a load-balanced set. Level 4 load balancers work at a transport level and have network-level information, such as an IP address and port, to decide the target for the incoming request. Load balancers are discussed in detail later in this chapter.
* **Application gateways**: Azure application gateway delivers high availability to your applications. They are a layer 7 load balancer that distribute the incoming traffic among healthy instances of services. Level 7 load balancers can work at the application level and have application-level information such as cookies, HTTP, HTTPS, and sessions for the incoming request. Application gateways are discussed in detail later in this chapter. Application gateways are also used when deploying Azure Kubernetes services specifically for scenarios in which ingress traffic from internet should be routed to Kubernetes services in cluster.

### VM high availability

VMs provide compute capabilities. They provide processing power and hosting for applications and services. If the application is deployed on a single VM and that machine is down, then even the application is not available. If the application is composed of multiple tiers and each tier is deployed in its own single instance of a VM, even downtime in a single VM can render the application non-available. Although Azure tries to make even single VM deployments highly available by placing them on different racks as soon as it can, Azure does not ensure or guarantee any SLA. Azure provides an SLA for those VMs that are grouped together in an availability set. It provides a five nines (99.999%) SLA for the availability of VMs if they are part of an availability set and more than two VMs are on that availability set.

### Compute high availability

Applications demanding high availability should be deployed on multiple VMs on the same availability set. If applications are composed of multiple tiers, then each tier should have a group of VMs on their dedicated availability set. In short, if there are three tiers of an application, there should be three availability sets and a minimum of six VMs (two in each availability set) to make the entire application highly available.

How does Azure provide an SLA and high availability to VMs in an availability set with multiple VMs in each availability set? This is the question that might be coming to mind.

Here, the use of concepts that we considered before comes into play—fault and update domains. When Azure sees multiple VMs in an availability set, it places those VMs on a separate fault domain. In other words, these VMs are placed on separate physical racks instead of the same rack. This ensures that at least one VM continues to be available even if there is a power, hardware, or rack failure. There are two or three fault domains in an availability set and, depending on the number of VMs in an availability set, the VMs are placed on separate fault domains or repeated in a round-robin fashion. This ensures that high availability is not impacted because of the failure of the rack.

Azure also places these VMs on a separate update domain. In other words, Azure tags these VMs internally in such a way that these VMs are patched and updated one after another, such that any reboot in an update domain does not affect the availability of the application. This ensures that high availability is not impacted because of the VM and host maintenance.

With the placement of VMs in separate fault and update domains, Azure ensures that not all of them are down at the same time and are alive and available for serving requests, even though they might be undergoing maintenance or facing physical downtime challenges:

The preceding diagram shows four VMs (two are IIS and two are SQL-related). Both IIS and SQL VMs are part of their availability set. The IIS and SQL VMs are on a separate fault domain and different racks in the data center. They would also be on separate upgrade domains.

The following diagram shows the relationship of fault and update domains.

### Storage high availability

VMs are backed up by storage accounts by storing their VHD files on them. While availability sets provide high availability to compute instances, they do not ensure the high availability of VHD files for VMs stored in storage accounts. The VHD files for all VMs might be placed on the same storage cluster, and any cluster failure can render all VMs non-available or less available than required. In short, it is not only computed services that need to be highly available, but even storage accounts storing VHD files should be placed on separate clusters such that in the event of failure, at least one or some VMs continue to be available, both from a computer and storage perspective.

Azure provides managed disks and disk management facilities. Managed disks provide better reliability for availability sets by ensuring that the disks of VMs in an availability set are sufficiently isolated from each other to avoid single points of failure. It does this by automatically placing the disks in different storage clusters. If a storage cluster fails due to hardware or software failure, only the VM instances with disks on those stamps fail. Each VM VHD in an availability set should be placed in a separate storage account, although, VMs from different availability sets can be placed in a storage account.

### PaaS high availability

Azure provides app services and cloud services for hosting managed platforms. Services and applications can be deployed on top of them. They provide flexibility, elasticity, and economies to create and deploy applications. These platforms are managed by Azure, and users do not interact with the base infrastructure on which they are deployed. They bring in a higher level of abstraction compared to IaaS by letting developers concentrate on their business problem and using the platform to fast-track their development and deployment process. This alleviates them to manage, operate, and monitor the base infrastructure. When an application is deployed in app services or cloud services, Azure provisions VMs that are not visible to users. The applications are deployed on these VMs and the Azure fabric controller is responsible for provisioning, managing, and monitoring them. The fabric controller monitors the status of the hardware and software of the host and guest machine instances. When it detects a failure, it maintains SLAs by automatically relocating the VM instances. When multiple cloud service role instances are deployed, Azure deploys these instances to different fault and update domains:

The previous diagram shows PaaS services with multiple VM instances deploying these web and worker roles on separate fault domains. Deploying on separate fault domains means deploying on separate racks within a data center. It also means that these services have separate network switches and power supplies, ensuring that even if one of the racks undergoes maintenance, or there is a disruption in the power supply to the rack or failure of the network switch, there are other instances available to serve customers requests.

### High-availability platforms

Azure has introduced a lot of new features in recent times with regard to high availability for PaaS. One of them is related to containers and the ecosystem surrounding them. Azure has introduced the following services:

* Containers in app services
* Azure container instance groups
* Azure Kubernetes services
* Other container orchestrators, such as DC/OS and Swarm

The other important platform that brings high availability is **Service Fabric**. Both Service Fabric and container orchestrators that include Kubernetes ensure that a desired number of application instances are always up and running in an environment. What this means is that even if one of the instances goes down in the environment, the orchestrator will know about it by means of active monitoring and will spin up a new instance on a different node, thereby maintaining the ideal and desired number of instances. It does this without any manual or automated interference from the administrator.

While Service Fabric allows any type of application to become highly available, orchestrators such as Kubernetes, DC/OS, and Swarm are specific to containers. Also, it is important to understand that these platforms provide features that help in rolling updates, rather than a big bank update that might effect the availability of the application.

### Data high availability

While VMs, app services, and containers provide high availability to compute, and managed disks provide high availability for storage, we also need to ensure that our data platforms and data is highly available.

Azure provides the following resources that make data highly available.

#### Azure CosmosDB

Azure CosmosDB is a truly global, highly available geo-replicated NoSQL data store. CosmosDB is available in almost all Azure regions, and it is also possible to configure geo-replication between all of these regions. CosmosDB allows for creating collections that are replicated across multiple regions asynchronously. It also provides flexibility in determining the consistency level, while configuring the replication strategy for high availability. These consistency levels help an architect to determine the criticality of the availability of data in other regions. These consistency levels are as follows:

* **Strong**: This ensures that every replicated region gets its data before returning to the user.
* **Bounded staleness**: This ensures that data is not stale in read regions beyond a point—either a fixed number of writes or a time span.
* **Sessions**: This ensures that data is consistent in a session.
* **Ordered prefixes**: This is when the writes will come to replicated regions in a similar order as they were written in the write region.
* **Eventual**: It is possible to have dirty reads here, and there is no SLA for determining the availability of data. It follows the principle of eventual consistency.

#### Azure SQL replication

Azure SQL provides the replication of the database to other regions to make them highly available. Replication can be done to any region. However, an architect should choose a peer region for replication. These peer regions are a minimum of 300 miles apart and are still connected with high-speed networks. These peer regions are also patched one at a time, and so there is no risk that the patch will happen in parallel to these regions.

The data in the replicated site can be made read-available to the applications.

#### Azure table storage

Azure provides table storage, which is a key value data storage in an Azure storage account. Azure maintains three copies of the data and makes them available in times of need. The data is stored in partitions with each partition, identified using a partition key and each row is assigned a RowId. Both the RowID and PartitionID are part of data payload. It provides storage for data without a schema, similar to NoSQL data stores. In fact, NoSQL data can be stored in Azure Tables easily.

A storage account can have multiple tables, and each table stores entities identified using partition and row identifiers.

#### Application high availability

High availability can be built-in within the software used for applications, or it can be built from the ground up within applications. One example of the high-availability feature provided by software is SQL server always-on availability groups. They help in keeping databases highly available.

Azure services also have a built-in, high-availability mechanism. In Azure SQL, data is replicated synchronously within the region. Active geo-replication allows up to four additional database copies in the same region or different regions. Azure storage has its own mechanism to make data available by replicating it to multiple data centers and regions.

### Load balancing

Azure provides two constructs to provision load balancers. It provides a level 4 load balancer that works at the transport layer within the TCP OSI stack, and a level 7 load balancer that works at the application and session level.

Although both application gateways and load balancers provide the basic features of balancing the load, they serve different purposes. There are use cases in which it makes more sense to deploy the application gateway compared to the load balancer.

The application gateway provides the following features that are not available in the Azure load balancers:

* **Web application firewall**: This is an additional firewall on top of the operating system firewall and has the capability to peek into incoming messages. This helps in identifying and preventing common web-based attacks such, as SQL injection, cross-site scripting attacks, and session hijacks.
* **Cookie-based session affinity**: Load balancers distribute incoming traffic to service instances that are healthy and relatively free. A request can be served by any service instance. There are applications that need advance features in which all subsequent requests following the first request should be processed by the same service instance. This is known as cookie-based session affinity. The application gateway provides cookie-based session affinity to keep a user session on the same service instance using cookies.
* **Secure Sockets Layer** (**SSL**) **offload**: The encryption and decryption of request and response data is performed by SSL and is generally a costly operation. Web servers should ideally be spending resources on processing and serving requests, rather than encryption and decryption of traffic. SSL offload helps in transferring this cryptography process from the web server to the load balancer, thereby providing more resources to web servers serving users. The request from the user is encrypted, but gets decrypted at the application gateway instead of the web server. The request from the application gateway to the web server is unencrypted.
* **End-to-end SSL**: While SSL offload is a nice feature for a certain application, there are certain mission-critical secure applications that need complete SSL encryption and decryption even if traffic passes through load balancers. An application gateway can be configured for an end-to-end SSL cryptography as well.
* **URL-based content routing**: Application gateways are also useful for redirecting the traffic to different servers based on the URL content of incoming requests. This helps in hosting multiple services alongside other applications.

#### Azure load balancers

The Azure load balancer distributes incoming traffic based on transport level information available to it. It relies on the following:

* An originating IP address
* A target IP address
* An originating port number
* A target port number
* A type of protocol—TCP or HTTP

#### Public load balancing

In this configuration, load balancers are assigned a public IP address. Assigning a public IP address ensures that the load balancer can accept requests coming in from the internet. Without a public IP address, it is not possible to access the resource from the internet. The load balancer can be configured with load-balancing rules. Load-balancing rules work at the port level. It accepts a source and destination ports map them together such that whenever a load balancer receives a request for the source port, the request is forwarded to a VM from a group of VMs attached to the load balancer on the destination port. This is shown in the following diagram:

But how does this entire thing work? How is a public IP address assigned to a load balancer? What does the load balancer contain? How is it configured with load balancer rules? How does the load balancer send requests to the VMs? How does the VM know that it is attached to the load balancer? The answers to all these questions are visible in the following diagram:

In this configuration, the load balancer is assigned a public IP address. The load balancer is accessible from the internet and can accept client requests. The load balancer can be configured with load-balancing and NAT rules. Both NAT and load-balancing rules are part of the frontend configuration. The frontend configuration sends client requests to one of the IP addresses available in the backend pool. These IP addresses are associated with the network interface card, which, in turn, is attached to VMs.

#### Internal load balancing

The following diagram shows the workings of an internal load balancer. You can see that the request comes from resources from Azure itself, since it is not accessible on the internet. In this configuration, the load balancer is assigned a private IP address. The load balancer is only accessible within the virtual network to which it is attached. It cannot be accessed through the internet. The remainder of its configuration is similar to a public load balancer. The load balancer can be configured with load-balancing and NAT rules:

The following diagram shows how multiple load balancers can be deployed to create solutions. In this, there is a public load balancer that accepts client requests and an internal load balancer for the database tier. The database-tier VMs are not accessible on the internet, but only through the load balancer on port 1433:

### Port forwarding

At times, there is a need for a request to always redirect to a VM. The Azure load balancer helps us achieve this with the NAT rules. NAT rules are evaluated after load-balancing rules are evaluated and none of these rules are satisfied. NAT rules are evaluated for each incoming request and, once it finds them, it forwards the request to that VM through a backend pool. It is to be noted that a VM cannot register the same port for both port forwarding using NAT and load-balancing rules.

### Azure application gateway

The Azure load balancer helps us to enable solutions at the infrastructure level. However, there are times when advanced services and features are required from the load balancer itself. These advance services include SSL termination, sticky sessions, advanced security, and more. Azure application gateway are built on top of Azure load balancers to provide these additional features. The Azure application gateway is a level 7 load balancer that works with the application and session payload in a TCP OSI stack. Application gateways have more information compared to the Azure load balancer to make decisions on request routing and load balancing between servers. Application gateways are managed by Azure and are highly available.

An application gateway sits in between the users and VMs, as shown in the following diagram:

Application gateways are internally implemented using VMs. The **internet information service** (**IIS**) is installed and configured with **Application Request Routing** (**ARR**) on these VMs. These gateways can be installed on multiple VMs, providing high availability for the gateways themselves. Although not visible, Azure load balancers distribute loads among multiple application gateway servers. Creating an application gateway requires an internal or public IP address and that is used by users to send requests to it. This public IP or internal IP is provided by the Azure load balancer working at the transport level (TCP/UDP) and having all incoming network traffic being load balanced to the application gateway worker instances. The application gateway then routes the HTTP/HTTPS traffic based on its configuration, whether it's a VM, cloud service, or internal or external IP address.

An application gateway is similar to an Azure load balancer from a configuration perspective, with additional constructs and features. It provides a frontend IP, protocol, certificate, and port configuration, backend pool, port, session affinity, and protocol configuration.

### Azure Traffic Manager

After gaining a good understanding of both the Azure load balancer and the application gateway, it's time to get into the details of Traffic Manager. Azure load balancers and application gateways are much-needed resources for high availability within a data center and region; however, to achieve high availability across regions and data centers, there is a need for another resource, and that is Traffic Manager.

Traffic Manager helps us to create highly available solutions that span multiple geographies, regions, and data centers. Traffic Manager is not similar to load balancers. It uses DNS to redirect requests to an appropriate endpoint determined by their health and configuration. Traffic Manager is not a proxy or a gateway. Traffic Manager does not see the traffic passing between the client and the service. It simply redirects the request based on the most appropriate endpoints.

Azure Traffic Manager enables you to control the distribution of traffic across your application endpoints. An endpoint is any internet-facing service hosted inside or outside of Azure.

Endpoints are internet-facing, reachable public URLs. Applications are provisioned within multiple geographies and Azure regions. Applications deployed to each region have a unique endpoint referred by **DNS CNAME**. These endpoints are mapped to the Traffic Manager endpoint. When a Traffic Manager is provisioned, it gets an endpoint by default with a .trafficmanager.net URL extension.

When a request arrives at the Traffic Manager URL, it finds the most appropriate endpoint out of its list and redirects the request to it. In short, Traffic Manager acts as a global DNS to identify the region that will serve the request.

However, how does Traffic Manager know which endpoints to use and redirect the client request to? There are two aspects that the Traffic Manager implements to determine the most appropriate endpoint and region.

First, Traffic Manager actively monitors the health of all endpoints. It can monitor the health of VMs, cloud services, and app services. If it determines that the health of an application deployed to a region is not suitable for redirecting traffic, it redirects the requests to a healthy endpoint.

Second, the Traffic Manager can be configured with routing information. There are four traffic routing methods available in Traffic Manager, which are as follows:

* **Priority**: Should be used when all traffic should go to a default endpoint, and backups are available in case the primary endpoints are unavailable.
* **Weighted**: Should be used to distribute traffic across endpoints evenly, or according to defined weights.
* **Performance**: Should be used for endpoints in different regions, and users should be redirected to the closest endpoint based on their location. This has a direct impact on network latency.
* **Geographic**: This should be used to redirect users from a specific geography to an endpoint (Azure, external, or nested) available in that geography or nearest to that geography. Examples include complying with data sovereignty mandates, localization of content and user experience, and measuring traffic from different regions.
* **Subnet**: This is a new routing method added and it helps in providing clients different endpoints based on their IP addresses. In this method, a range of IP addresses are assigned to each endpoint. These IP address ranges are mapped to the client IP address to determine an appropriate returning endpoint. Using this routing method, it is possible to provide different content to different people based on their originating IP address.
* **Multivalue**: This is also a new method added in Azure. In this method, multiple endpoints are returned back to the client and any of them can be used. This ensures that if one endpoint is unhealthy, other endpoints can be used. This helps in increasing the overall availability of the solution.

It should be noted that after the Traffic Manager determines a valid healthy endpoint, clients connect directly to the application.

Architectural considerations for high availability

Azure provides High availability to various means and at various levels. High availability could be at data center level, it could be at a region level, or across Azure.  In this section, we will go through some of the architectures for high availability.

High availability within Azure regions

The architecture shown next shows high-availability deployment within a single Azure region. High availability is designed at the individual resource level. In this architecture, as shown in the following diagram, there are multiple VMs at each tier connected through either an application gateway or load balancer, and they are part of an availability set. Each tier is associated with an availability set. These VMs are placed on separate fault and update domains. While the web servers are connected to application gateways, the rest of the tiers, such as application and database tiers, have internal load balancers:

High availability across Azure regions

The architecture shown next shows similar deployments on two different Azure regions. As shown in the diagram, both the regions have the same resources deployed. High availability is designed at the individual resource level within these regions. There are multiple VMs at each tier, connected through the load balancer, and they are part of the availability set. These VMs are placed on separate fault and update domains. While the web servers are connected to external load balancers, the rest of the tiers, such as application and database tiers, have internal load balancers. It should be noted that application load balancers can be used for web servers and application tiers instead of Azure load balancers if there is a need for advanced services, such as session affinity, SSL termination, advance security using WAF, and path-based routing. Databases in both the regions are connected to each other using VNET peering and gateways. This is helpful in configuring log shipping, SQL Server AlwaysOn, and other data synchronization techniques.

The endpoints of load balancers from both the regions are used to configure Traffic Manager endpoints, and traffic is routed based on the priority load balancing method. Traffic Manager helps in routing all requests to the East US region and, after failover, to West Europe in the case of the non-availability of the first region:

Best practices

This section describes high-availability best practices. They have been categorized into application, deployment, data management, and monitoring.

Application high availability

An application should be built with high availability as one of the most important architectural concerns. Some of the important application-related, high-availability practices are mentioned next:

* An application should implement appropriate exception handling to gracefully recover and inform stakeholders about the issue
* An application should try to perform the same operation again for a fixed interval and a certain number of times before exiting in the event of an error or exception
* An application should have the built-in timeout capability to decide that an exception cannot be recovered from
* Maintaining logs and writing logs for all errors, exceptions, and execution should be adopted within the application
* Applications should be profiled to find their actual resource requirements in terms of compute, memory, and network bandwidth for a different number of users

Note

Please refer to <https://docs.microsoft.com/en-us/azure/architecture/checklist/availability> to learn more about applications and other high-availability best practices.

Deployment

A deployment strategy, to a large extent, affects the availability of the application and overall environment. Here are the few following things you should do:

* Deploy multiple instances of Azure resources, including multiple instances for VMs, cloud services, and other resources.
* Deploy VMs on availability sets or availability zones. They cannot be used together.
* Deploy multiple instances of VMs across multiple regions.
* Create multiple environments and keep at least one of them in standby mode.

Data management

Some of the important data-related best practices for high availability include the following:

* If possible, store data on Azure-provided services, such as Azure SQL, Cosmos DB, and table storage
* Use storage accounts that are based on the geo-redundant type
* Ensure that data is replicated to multiple regions and not only within a zone or data center
* Take periodic backups and conduct restore tests frequently
* If storing data in VMs, ensure that there are multiple VMs and that they are either on availability sets or availability zones
* Use keys and secrets to data stored in the Azure key vault

Monitoring

Some of the important monitoring-related best practices for high availability are as follows:

* Use OMS (log analytics) to monitor the environment and enable log auditing
* Use application insights to capture telemetry information from the custom application and environment related to compute, storage and networks, and other log information
* Ensure alerts are configured on OMS for issues related to the availability of the environment and application
* Visit the Azure monitor frequently to gather recommendations related to high availability

## calability

Running applications and systems that are available to users for consumption is important for architects of any serious application. However, there is another equally important application feature that is one of the top priorities for architects, and this is the scalability of the application.

Imagine a situation in which an application is deployed and obtains great performance and availability with a few users, but both availability and performance degrades as the number of users start begins to increase. There are times when an application under normal load performs well, but degrades in performance with the increase in the number of users. This can happen if there is a sudden increase in the number of users and the environment is not built for such a large number of users.

To accommodate such spikes in the number of users, you might provision the hardware and bandwidth for handling spikes. The challenge with this is that the additional capacity is not used for a majority of the year and does not provide any return on investment. It is provisioned for use only during the holiday season or sales. I hope you are getting to know the problems architects are trying to solve. All these problems are related to capacity sizing and the scalability of an application. The focus of this chapter is to understand scalability as an architectural concern and to check out services provided by Azure for implementing scalability.

Capacity planning and sizing are a few of the top priorities for architects for their applications and services. Architects must find a balance between buying and provisioning too many resources versus fewer resources. Having fewer resources can lead to not being able to serve all users, turning them to the competition, while having more resources can hurt your budget and return on investment because most of the resources remain unused most of the time. Moreover, the problem is amplified with a varied level of demand during different times. It is almost impossible to predict the number of users for the application round the clock and year. However, it is possible to find an approximate number using past information and continuous monitoring.

Scalability refers to:

**"Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources (typically hardware) are added."**

Scalability refers to the ability to handle a growing number of users and provide them with the same level of performance when there are fewer users in application deployment, processes, and technology. Scalability might refer to serving more requests without degradation of performance, or it might refer to handling larger and more time-consuming work without any loss of performance in both cases.

Capacity planning and sizing exercises should be undertaken by architects at the very beginning of the project during the planning phase to provide scalability to applications.

Some applications have stable demand patterns, while it is difficult to predict others. Scalability requirements are known for stable demand applications, while it is a more involved process for variable demand applications. Auto scaling, a concept we will review in the next section, should be used for such applications whose demands cannot be predicted.

### Scalability versus performance

It is quite easy to get confused between scalability and performance architectural concerns , because scalability is all about ensuring that, no matter the number of users consuming the application, all get the same pre-determined level of performance.

Performance relates to application features that ensure that the application caters to predefined response times and throughput. Scalability refers to having provisions for more resources when needed, to accommodate more users without sacrificing performance.

It is better to understand this using an analogy. The speed of a train refers to the performance of railway systems. However, accommodating more trains to run in parallel at the same or a higher speed will be referred to the scalability of the railway network.

### Azure scalability

In this section, we will look at the features and capabilities provided by Azure to make applications highly available. Before we get into the architecture and configuration details, it is important to understand Azure's high-availability concepts.

#### Concepts

The fundamental constructs provided by Azure to attain high availability are as follows:

* Scaling
* Scaling up and down
* Scaling out and in
* Auto scaling
* Rolling updates

##### **Scaling**

Scaling refers to the transformation that either increases or decreases the units of resources used to serve requests from users. Scaling can be automatic or manual. Manual scaling requires an administrator to manually initiate the scaling process, while automatic scaling refers to an automatic increase or decrease in resources based on events available from the environment and ecosystem, such as memory and CPU availability. Scaling can be effected up or down, or out and in, which will be explained later in this section.

##### **Scaling up**

Scaling up a virtual machine or service refers to the adding of additional resources to existing servers, such as CPU, memory, and disks. Its aim is to increase the capacity of existing physical hardware and resources:

##### **Scaling down**

Scaling down a virtual machine or service refers to the removal of existing resources from existing servers, such as CPU, memory, and disks. Its aim is to decrease the capacity of existing physical and virtual hardware and resources.

##### **Scaling out**

Scaling out refers to the process of adding additional hardware in terms of additional servers and capacity. This typically involves adding new servers, assigning them IP addresses, deploying applications on them, and making them part of the existing load balancers such that traffic can be routed to them. Scaling out can be automatic or manual as well. However, for better results, automation should be used:

##### **Scaling in**

Scaling in refers to the process of removing the existing hardware in terms of existing servers and capacity. This typically involves removing existing servers, de-allocating their IP addresses, and removing them from the existing load balancer configuration such that traffic cannot be routed to them. In the same way as scaling out, scaling in can be automatic or manual.

##### **Auto scaling**

Auto scaling refers to the process of either scaling up/down or scaling out/in dynamically based on application demand, and it happens using automation. Auto scaling is helpful because it ensures that deployment always consists of a correct and ideal number of server instances. Auto scaling helps in building applications that are fault tolerant. It not only helps in scalability, but also makes applications highly available. Finally, it provides the best cost management. Auto scaling helps in having the optimal configuration for server instances based on demand. It helps in not over-provisioning servers that are underutilized and removes servers that are no longer required after scaling out.

### PaaS scalability

Azure provides app services for hosting managed applications. App services is a PaaS offering from Azure. It provides the web and mobile platform. Behind the web and mobile platform is a managed infrastructure that is managed by Azure on behalf of its users. Users do not see or manage the infrastructure; however, they have the capability to extend the platform and deploy their applications on top of it. With this, architects and developers can concentrate on their business problems instead of worrying about the base platform and infrastructure provisioning, configuration, and troubleshooting. Developers have the flexibility to choose any language, operating system, and framework to develop their applications. App services provide multiple plans and, based on the plans chosen, capabilities of scalability are available. App services provide the following five plans:

* **Free**: This uses shared infrastructure. It means multiple applications will be deployed on the same infrastructure from the same or multiple tenants. It provides 1 GB of storage free of cost. No scaling facility is available in this plan.
* **Shared**: This also uses shared infrastructure and provides 1 GB of storage free of cost. Additionally, custom domains are also provided as an extra feature. No scaling facility is available in this plan.
* **Basic**: This has three different **stock keeping units**(**SKU**)—B1, B2, and B3. They have increasing units of resources available to them in terms of CPU and memory. In short, they provide improved configuration of the VMs backing these services. Additionally, they provide storage, custom domains, and SSL support. The basic plan provides basic features for manual scaling. There is no automatic scaling available in this plan. A maximum of three instances can be used for scaling out the application.
* **Standard**: This also has three different SKUs—S1, S2, and S3. They have increasing units of resources available to them in terms of CPU and memory. In short, they provide improved configuration of the VMs backing these services. Additionally, they provide storage, custom domains, and SSL support similar to the basic plan. This plan also provides a traffic manager, staging slots, and one daily backup as an additional feature on top of the basic plan. The standard plan provides features for automatic scaling. A maximum of 10 instances can be used for scaling out the application.

* **Premium**: This also has three different SKUs—P1, P2, and P3. They have increasing units of resources available to them in terms of CPU and memory. In short, they provide improved configuration of the VMs backing these services. Additionally, they provide storage, custom domains, and SSL support similar to the basic plan. This plan also provides a traffic manager, staging slots, and 50 daily backups as an additional feature on top of the basic plan. The standard plan provides features for automatic scaling. A maximum of 20 instances can be used for scaling out the application.

#### PaaS – Scaling up and down

Scaling up and down services hosted in app services is quite simple. The Azure app services menu items to scale up, which opens a new blade with all plans and their SKUs listed. Choosing a plan and SKU will scale the service up or down, as shown in the following screenshot:

#### PaaS – Scaling out and in

Scaling out and in services hosted in app services is also quite simple. The Azure app services menu items to scale out, which opens a new blade with scaling configuration options.

By default, auto scaling is disabled for both premium and standard plans. It can be enabled using the **Scale Out** menu item and by clicking on the **Enable autoscale** button, as shown:

Manual scaling does not require configuration but auto scaling helps in configuring with the aid of the following properties:

* **Mode of scaling**: Based on some metric, such as CPU or memory usage or just scale to specify the number of instances.
* **When to scale**: Multiple rules can be added that determine when to scale out and in. Each rule can determine the criteria such as CPU or memory consumption, whether to increase or decrease instances, and how many instances to increase or decrease at a time. At least one rule for scale out and one rule for scale in should be configured. Threshold definitions help in defining the upper and lower limits that should trigger the auto scale—either increase or decrease the number of instances.
* **How to scale**: Specifies how many instances to create or remove in each scale out or in operation:

This is a quite to good feature to enable in any deployment. However readers should enable both scaling out and in together to ensure that the environment is back to normal capacity after scaling out.

### IaaS scalability

There are users who want to have complete control over the base infrastructure, platform, and application. They prefer to consume IaaS solutions compared to PaaS solutions. For such customers, when they create VMs, they are also responsible for capacity sizing and scaling. There is no out-of-the-box configuring for manual or auto scaling VMs. These customers will have to write their own automation scripts, triggers, and rules to achieve auto scaling. With VMs comes the responsibility of maintaining them as well. Patching, updating, and the upgrading of VMs is the responsibility of owners. Architects should think about both planned as well as unplanned maintenance. How these VMs should be patched, the order, grouping, and other factors must be thought through to ensure that both the scalability and availability of an application is not compromised. To help alleviate such problems, Azure provides VM scale sets as a solution.

### VM scale sets

**VM scale sets** (**VMSS**) are an Azure compute resource that you can use to deploy and manage a set of identical VMs. With all VMs configured in the same way, scale sets are designed to support true auto scaling, and no pre-provisioning of VMs is required. It helps in provisioning multiple identical VMs connected to each other through a virtual network and subnet.

VMSS consists of multiple VMs, but they are managed at VMSS level.  All VMs are part of this unit and any changes made are applied to the unit, which, in turn, applies it to VMs using a pre-determined algorithm:

It allows these VMs to be load balanced using the Azure load balancer or application gateways. All the VMs could be either Windows or Linux operating systems. They can run automated scripts using a PowerShell extension and they can be managed centrally using desired state configuration. They can be monitored as a unit and individually using log analytics as well.

VMSS can be provisioned from the Azure portal, Azure command-line interface, Azure resource manager templates, REST APIs, and PowerShell cmdlets. It is possible to invoke REST APIs and Azure CLI from any platform, environment, and operating system, and in any language.

Already, a lot of Azure services use VMSS as its underlying architecture. Among them are Azure Batch, Azure Service Fabric, and Azure container services. Azure container services, in turn, provision Kubernetes and DC/OS on these VM scale sets.

#### VMSS architecture

VMSS allows for the creation of up to 1,000 VMs in a scale set when using a platform images and 100 VMs if using a custom image. If the number of VMs is less than 100 in a scale set, they are placed in a single availability set; however, if they are greater than 100, multiple availability sets are created, known as placement groups, and VMs are distributed among these availability sets. We know from the last chapter that VMs in an availability set are placed on separate fault and update domains. Availability sets related to VMSS have five fault and update domains by default. VMSS provides a model that holds metadata information for the entire set. Changing this model and applying changes impacts all VM instances. This information includes maximum and minimum VM instances, the operating system SKU and version, the current number of VMs, fault and update domains, and more.  This is demonstrated in the following diagram:

#### VMSS scaling

Scaling refers to an increase or decrease in compute and storage resources. VMSS is a feature-rich resource that makes scaling easy and efficient. It provides auto scaling, which helps in scaling up or down based on external events and data such as CPU and memory usage.  Some of the VMSS scaling features are mentioned as follows.

##### **Horizontal versus vertical scaling**

Scaling can be horizontal or vertical, or both. Horizontal scaling is another name for scaling out and in, while vertical scaling is about scaling up and down.

##### **Capacity**

VMSS have a capacity property that determines the number of VMs in a scale set. VMSS can be deployed with zero as a value for this property. It will not create a single VM; however, if you provision VMSS by providing a number for the capacity property, that number of VMs are created.

##### **Auto scaling**

Automatic scaling of VMs in VMSS refers to the addition or removal of VM instances based on configured environments to meet the performance and scalability demands of an application. Generally, in the absence of VMSS, this is achieved using automation scripts and runbooks.

VMSS helps in this automation process with the help of configuration. Instead of writing scripts, VMSS can be configured for automated scaling up and down.

Auto scaling consists of multiple integrated components to achieve its end goal. Auto scaling continuously monitors the VMs and collects telemetry data from them. It stores this data combines it and then evaluates it against a set of rules to determine whether it should trigger the auto scale. The trigger could be to scale out or scale in. It could also be to scale up or down.

Auto scale uses diagnostic logs for collecting telemetry data from VMs. These logs are stored in storage accounts as diagnostic metrics. Auto scale also uses the insight monitoring service that reads these metrics, combines them together, and stores them in their own storage account.

Auto scale background jobs run continually to read the insights' storage data, evaluate them based on all the rules configured for auto scaling, and execute the process of auto scaling, should any of the rules or combination of rules return positive. The rules can take into consideration metrics from guest VMs as well as the host server.

### Note

The rules defined using the property descriptions are available at <https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview>.

The auto scale architecture is shown in the following diagram:

Auto scale can be configured for scenarios that are more complex than general metrics available from environments. For example, scaling could be based on any of the following:

* Scaling on a specific day
* Scaling on a recurring schedule such as weekends
* Scaling differently on weekdays and weekends
* Scaling during holidays, that is, one of the events
* Scaling on multiple resource metrics

These can be configured using the schedule property of insight resources that help in registering rules.

Architects should ensure that at least two actions—scale out and scale in—are configured together. Scaling in or scaling out configuration will not help in achieving the scaling benefits provided by VMSS.

## Upgrades and maintenance

After VMSS and applications are deployed, they need to be actively maintained. Planned maintenance should be conducted periodically to ensure that both the environment and application is up to date with the latest features, and the environment is current from a security and resilience point of view.

Upgrades can be associated with applications, the guest VM instance, or the image itself. Upgrades can be quite complex because they should happen without affecting the availability, scalability, and performance of environments and applications. To ensure that updates can take place one instance at a time using rolling upgrade methods, it is important that VMSS supports and provides capabilities for these advanced scenarios.

There is a utility provided by the Azure team to manage updates for VMSS. It's a Python-based utility that can be downloaded from <https://github.com/gbowerman/vmssdashboard>. It makes REST API calls to Azure to manage scale sets. This utility can be used for starting, stopping, upgrading, and re-imaging VMs on a fault domain or group of VMs, as shown in the following screenshot:

### Application updates

Application updates in VMSS should not be executed manually. They must be executed as part of release management and pipelines using automation. Moreover, the update should happen one application instance at a time, not affecting the overall availability and scalability of the application. Configuration management tools, such as the desired state configuration, should be deployed to manage application updates. The DSC pull server can be configured with the latest version of the bits and they should be applied on a rolling basis to each instance.

### Guest updates

Updates to VM are the responsibility of administrator. Azure is not responsible for patching guest VMs. Guest updates are in preview mode and users should control patching manually or using custom automation such as runbooks and scripts. However, rolling patch upgrades are in preview mode and can be configured in the ARM template using the upgrade policy, as shown here:

Copy

"upgradePolicy": {

"mode": "Rolling",

"automaticOSUpgrade": "true" or "false",

"rollingUpgradePolicy": {

"batchInstancePercent": 20,

"maxUnhealthyUpgradedInstanceCount": 0,

"pauseTimeBetweenBatches": "PT0S"

}

}

### Image updates

VMSS can update the OS version without any downtime. OS updates involve changing the version or SKU of the OS or changing the URI of a custom image. Updating without downtime means updating VMs one at a time or in groups (such as one fault domain at a time) rather than all at once. By doing so, any VMs that are not being upgraded can keep running.

### Best practices of scaling provided by VMSS

In this section, we will go through some of the best practices that applications should implement to take advantage of the scalability capability provided by VMSS.

#### The preference for scaling out

Scaling out is a better scaling solution compared to scaling up. Scaling up or down means resizing VM instances. When a VM is resized, it generally needs to be restarted, which has its own disadvantages. First, there is downtime for the machine. Second, if there are active users connected to the application on that instance, they might face the unavailability of the application, or they might even have lost transactions. Scaling out does not impact existing VMs. It provisions newer machines and adds them to the group.

#### Bare-metal versus dormant instances

Scaling new instances can take two broad approaches: creating the new instance from scratch, which means to install applications, configure, and test, or start the dormant, sleeping instances when they are needed due to scalability pressure on other servers.

#### Configuring the maximum and minimum number of instances appropriately

Setting a value of two for both the minimum and maximum instance count, with the current instance count being two, means no scaling action can occur. There should be an adequate margin between the maximum and minimum instance counts, which are inclusive. Auto scale always scales between these limits.

#### Concurrency

Applications are designed for scalability to focus on concurrency. Applications should use asynchronous patterns to ensure that client requests do not wait indefinitely to acquire resources if resources are busy serving other requests. Implementing asynchronous patterns in code ensures that threads do not wait for resources and systems are exhausted of all available threads. Applications should implement the concept of timeouts if intermittent failures are expected.

#### Stateless

Applications and services should be designed to be stateless. Scalability can become a challenge to achieve with stateful services, and it is quite easy to scale stateless services. With states comes the requirement for additional components and implementations, such as replication, centralized or decentralized repository, maintenance, and sticky sessions. All these are impediments on the path to scalability. Imagine a service maintaining an active state on a local server. No matter the number of requests on the overall application or on the individual server, the subsequent requests must be served by the same server. Subsequent requests cannot be processed by other servers. This makes scalability implementation a challenge.

#### Caching and CDN

Applications and services should take advantage of caching. Caching helps eliminate multiple subsequent calls to either databases or filesystems. This helps in making resources available and free for more requests. The **Content Distribution Network** (**CDN**) is another mechanism for caching static files such as images and JavaScript libraries. They are available on servers across the globe. They also make resources available and free for additional client requests. This makes applications highly scalable.

#### N+1 design

**N+1** **design** refers to building redundancy within the overall deployment for each component. It means to plan for some redundancy even when it is not required. It could mean additional VMs, storage, and network interface cards.

## Summary

High availability and scalability are important and crucial architectural concerns. Almost every application and every architect tries to implement high availability. Azure is a mature platform that understands the need for these architectural concerns in applications and provides resources to implement them at multiple levels. These architectural concerns are not an afterthought, and should be part of the application life cycle development, starting from the planning phase itself.