CHAPTER 6

Azure Storage

In the previous chapters, we discussed and explained the most used cloud services in Azure, such as virtual machines, scale sets, and App Service, and learned how to use them. One Azure service that we mentioned but still did not go into details about is Storage Account, which will be the subject of this chapter.

This chapter covers the following topics regarding one of the most used Azure services:

- Storage Account and its features
- Storage Account management
- Security of storage accounts
- Azure disks
- Data transfer

After this chapter, we will be able to design, implement, and deploy a storage workload in Azure in various scenarios and using various tools.

Storage Accounts

When we talk about storage, in general, we are aware that without it, we will not be able to do almost anything. Storage is everywhere around us, in computers, mobile phones, tablets, even smartwatches and TVs, and logically any kind of cloud. If we focus our talk on Microsoft Azure, there are many storage options, but Storage Account is the most used. Easy and quick deployment and scaling are some of the most significant benefits of using storage accounts in Azure. Also, we are witnesses that pricing for storing data in Azure decreases over the years, with a tendency to be lower. For instance, 1 TB of stored data in a blob container in the archive tier is only 1$ per month.

© Vladimir Stefanovic and Milos Katinski 2021
V. Stefanovic and M. Katinski, *Pro Azure Administration and Automation*,
https://doi.org/10.1007/978-1-4842-7325-8_6

At first look, we could say that Azure storage accounts will be perfect for our needs, but let's go more in depth to see what we can expect from Storage Account and what it offers.

Account Types and Performance Tiers

The very first things that we need to choose when designing or implementing storage accounts are the account type and performance tier. A few options are on the table, and a few combinations as well, so we need to know what we exactly want to achieve.

In the first place, we have to decide what kind of performance we need: Standard or Premium. Storage accounts in the Standard tier will be created on magnetic drives, whereas storage accounts in the Premium tier will be created on SSDs, which is the first significant difference on the physical layer. Since the price is very different between these tiers and there is no possibility of upgrading or downgrading between storage tiers, we have to plan carefully what performance tier will be used. The Standard tier is sufficient for most workloads. However, there are also scenarios where the Premium tier is needed due to better performance and lower latency in the first place. In addition, constant improvement of Azure Storage service features and the possibility to store block and page blobs in the Premium tier could be crucial for choosing this performance tier.

Once we define our needs, we have to select an appropriate storage account performance tier and account type from the following:

- Standard: General-purpose V2
- Premium: Page blobs
- Premium: Block blobs
- Premium: File shares

The general-purpose account type is most used and can fit most Azure workloads because it could be used for storing all types of data, blobs, file shares, tables, and queues, inside of one storage account. Of course, since the Standard performance tier is cheaper than Premium, and if we do not need low latency, it will be our first choice in most cases. On the other hand, a premium storage account can be used for storing page and block blobs, as well as file shares, but not in one single storage account. If we need to use all these account types in the Premium performance tier, we need to separate them into different storage accounts. Also, append blobs, tables, and queues are still not supported in the Premium performance tier.

In conclusion, if we do not need high performance and do not need to deploy anything, the general-purpose storage account will be more than enough for our needs. Otherwise, if we need better performance for our data, we have to plan a bit more because one account type does not cover all storage services in the Premium performance tier. Last but not least, the pricing schemes for these performance tiers are very different. For instance, blob storage in the Standard performance tier costs approximately $22 per TB, whereas the Premium performance tier will cost us approximately $185 per TB. If we have to think about financial matters, it will be an additional task before deciding on the performance tier that we will use.

Storage Account Replication

In real-world scenarios, if our data are not backed up or replicated, we cannot say that we are protected. For the storage accounts in Azure, there are a few different replication plans that we can use to protect our data, and fortunately, we can benefit from them. We can divide replication plans into two main categories: replication in the primary region and replication in the secondary region.

Replication or redundancy in the primary region includes *locally redundant storage (LRS)* and *zone-redundant storage (ZRS)*. LRS is the lowest-cost redundancy option, and all our data will be replicated three times within a datacenter where a storage account is created. Even though LRS is the lowest-cost option for redundancy, Microsoft guarantees 99.999999999% (11 nines) SLA in a year. Another option for replication in the primary region is ZRS, which can be implemented only in regions with zones. This replication plan gives us three copies of our data in three separate zones in the region, with 99.9999999999% (12 nines) SLA in a year.

If we want to protect our data better, we can select one of the replication plans that replicate our data to the secondary region: *geo-redundant storage (GRS)* and *geo-zone-redundant storage (GZRS)*. Both options will give us three copies in the primary region and three copies more in the secondary region. Also, for both options, Microsoft guarantees 99.99999999999999% (16 nines) SLA in a year and an RPO of 15 minutes. Additionally, GRS and GZRS can be implemented as *read-access geo-redundant storage (RA-GRS)* and *read-access geo-zone-redundant storage (RA-GZRS)*, where replicas in the secondary region will be readable. One of the essential things when considering storage account replication is that not all redundancy options are available in all regions. However, since the improvements on the Azure infrastructure are continually processed, we can expect that “missing” features will come to specific regions in the future.

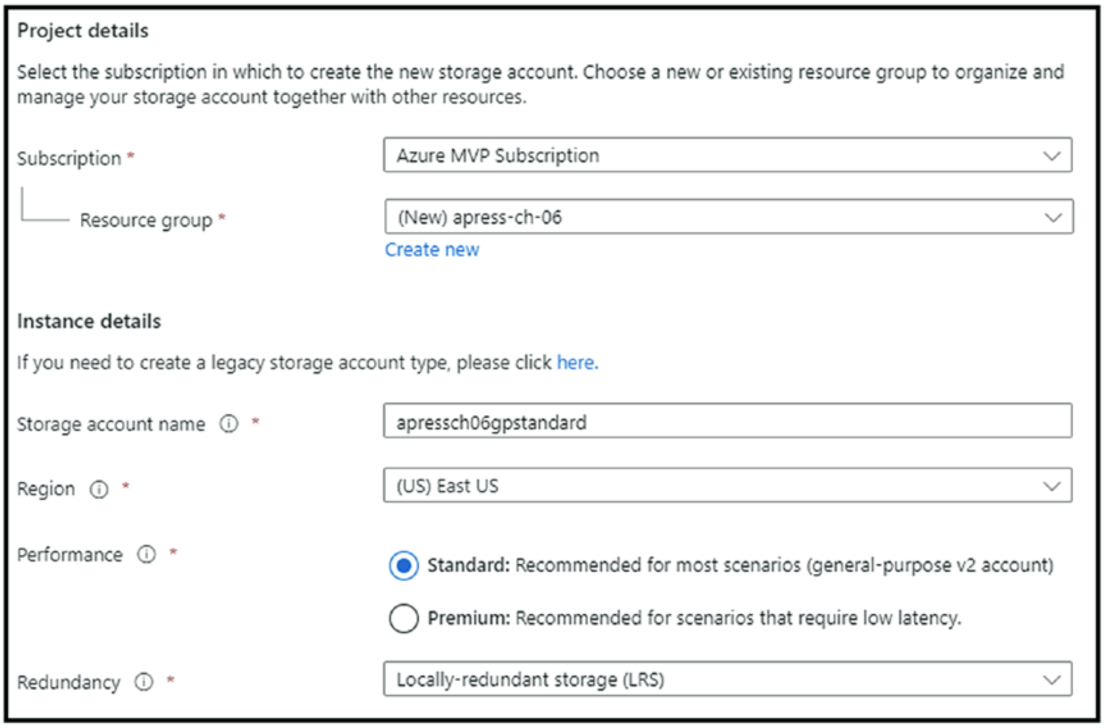
In the replication plans that include the secondary region, we must be aware of potential data loss since replication between regions is asynchronous. Microsoft does not guarantee any SLA for this replication since the data replication time depends on the amount of data.

Creating a Storage Account

Like almost all Azure resources, a storage account can be created using all management tools used in the previous chapters.

Azure Portal

Before we go to the next discussion about storage services, let us create one general-purpose V2 storage account in the Standard performance tier. If we decide to use Azure Portal to perform this operation, the first step is to click **+ Create a resource**, search for “storage account” in the *Marketplace*, and click **Create**. In our cases, we will configure parameters just on the *Basics* tab, and all other things we will configure later in this chapter. As with all other Azure resources, we need to define Subscription and Resource group under *Project details*. Under *Instance details,* we will define Storage account name, which must be unique across Azure, Location, Performance tier, Account type, and Replication type. Once these parameters are filled, click **Review + create** and then **Create** to create a storage account. All these parameters are shown in Figure 6-1.



Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *	Azure MVP Subscription
Resource group *	(New) apress-ch-06

Create new

Instance details

If you need to create a legacy storage account type, please click here.

Storage account name *	apressch06gpstandard
Region *	(US) East US
Performance *	[✓ **Standard:** Recommended for most scenarios (general-purpose v2 account)] **Premium:** Recommended for scenarios that require low latency.
Redundancy *	Locally-redundant storage (LRS)

Figure 6-1. *Define Subscription and Resource group under Project details*

ARM Template, PowerShell, and Azure CLI

Since the deployment code could be pretty big, all ARM template, Azure PowerShell, and Azure CLI scripts are stored in the Apress GitHub account, available at the following URL:

https://github.com/Apress/pro-azure-admin-and-automation

Blob Containers

One of the most used storage services is the blob container that could be used for various workloads, such as storing logs, pictures, SQL backup files, and many others. Also, the blob container is the storage service that is used by many other Azure services for storing their data. For instance, if we want to configure a disaster recovery location in Azure or prepare an on-premises environment for migrations, Azure Site Recovery will use a storage account and blob containers to store migrated data. As we mentioned earlier, blob containers are available in both performance tiers, with some limitations based on the account type.

In a blob container, we can store three different blob types: block, page, and append. *Block blobs* are designed and optimized for uploading a large amount of data to the blob container. A block blob comprises a maximum of 50.000 blocks, where the maximum size of a block is 4 TB. The maximum size of a single blob is approximately 190 TB. A *page blob*, which is ideal for storing virtual machine disks, comprises 512-byte pages optimized for random read/write operations. The maximum size of a page blob is 8 TB. Similarly to a block blob, an *append blob* comprises blocks, but it is optimized for append operations. Once the blob is modified, new data will be added to the end of the blob. Updating or deleting any block is not allowed in a page blob, making a page blob ideal for storing log files. The maximum number of blocks in an append blob is 50.000, and every block has a maximum size of 4 MB, so the maximum size of an append blob is around 200 GB.

Blob Access Tier

Regardless of other storage account services, blob containers have *access tiers* along with performance tiers, but this is applicable only for the Standard performance tier. Since the Premium performance tier uses SSDs, this kind of tier is not needed. Access tiers include

- Hot: For data that is accessed frequently
- Cool: For data that is not accessed so frequently but must be accessible
- Archive: For data that will be accessed rarely

The *Hot tier* is designed and optimized for data that has to be accessed frequently, and it is the most expensive compared to other tiers. Per TB, we have to pay around $22, but transaction and data retrieval costs are cheap, or there are no costs. Also, there is no limitation on how many days data must retain in the Hot tier. On the other hand, the Cool tier is designed and optimized for data that will not be accessed so frequently but still must be online. Per TB of data in the *Cool tier*, we have to pay between $10 and $15, depending on the location. Transaction and data retrieval costs are still at a low level, but there are some costs. These access tiers could be configured on the account level or blob level. By default, every storage account during creation will get the Hot access tier if we do not configure the Cool access tier. Once the blob is uploaded or copied to the blob container, it will inherit the access tier from the storage account if that is not defined differently in the uploading or copying process. Of course, these access tiers on blobs can be changed later if needed in both directions, but we must know that data in the Cool tier must retain for 30 days. If we change the access tier from Cool to Hot or remove data from the Cool tier, there will be some penalty that we must pay.

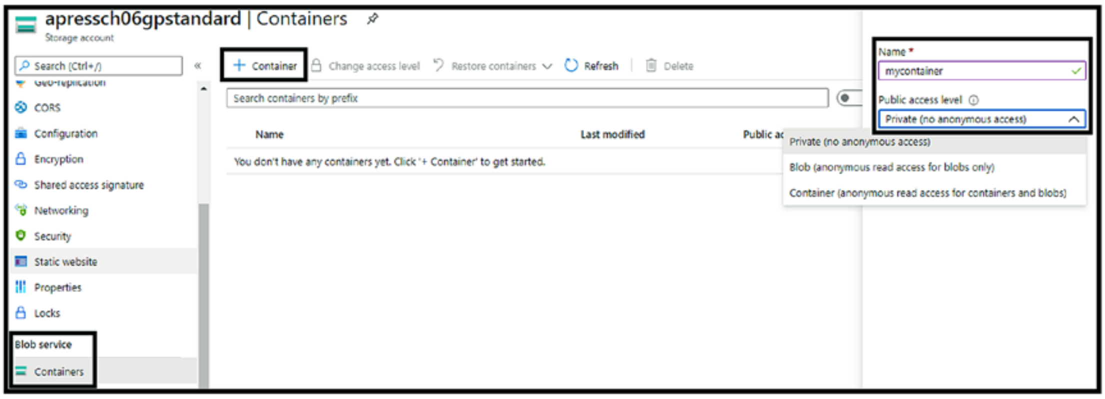
The third access tier, which cannot be configured on the account level, is the *Archive tier*. If we move data to the Archive tier, we have to pay between $1 and $3 per TB, depending on the region we select. The incredibly low price for storing data in the Archive tier brings many limitations. The Archive tier is designed and optimized for data that will be accessed rarely, and all data must remain for 180 days in the Archive tier. Also, the pricing scheme for read operations and data retrieval is expensive, mainly because all data in the Archive tier are offline and stored on magnetic tracks. This access tier is ideal for long-term backups for compliance or any other type of data that has to be kept for a long time. Also, data retrieval in the Archive tier can take several hours, especially for large files.

Creating a Blob Container

Once we have created a storage account, creating a blob container is a straightforward process and can be done using all management tools.

Azure Portal

When we are in the desired storage account user interface, in the left pane, we have to find *Containers* under *Blob service*. Then, we have to click **+ Container** and define the container name and public access level if it is needed. These steps are shown in Figure 6-2.



You don't have any containers yet. Click '+ Container' to get started.

Name * mycontainer

Public access level: Private (no anonymous access)

Figure 6-2. Define the container name and public access level if it is needed

ARM Template, PowerShell, and Azure CLI

Since the deployment code could be pretty big, all ARM template, Azure PowerShell, and Azure CLI scripts are stored in the Apress GitHub account, available at the following URL:

https://github.com/Apress/pro-azure-admin-and-automation

Azure Files

Even though blob containers are widely used and maybe have a better adoption rate, Azure Files is a storage service with its place and customers due to the nature of its service. Unlike blob containers that offer object storage, Azure Files offers fully managed file shares in the cloud accessible via standardized SMB and NFS protocols. Azure Files can be mounted on Windows, Linux, and macOS machines. In addition, Azure Files can be cached in Windows environments by implementing Azure File Sync on Windows Server, so customers will have the same experience as having a local file share while their data are in the cloud.

From a pricing perspective, Azure Files is more expensive than blob containers, and for 1 TB of data in a file share in the Standard tier, we have to pay around $60. A file share in the Premium performance tier is around $180 per TB, but that can vary depending on location. Also, by default, there is a maximum quota of 5 TB per file share, but in the storage account configuration, we can use a large file share and increase the limit to 100 TB per file share.

Azure Files Tiers

Like blob containers, Azure Files also provides tiering, but in a little bit different way. This feature is new and still is not available in all regions, but many regions can leverage its benefits. The Premium tier is available only if the FileStorage account type is deployed, and the price per TB is around $180. All data are stored on high-performant SSD storage. For Azure Files in the Standard performance tier, we can choose one of the three available performance tiers:

- Transaction-optimized
- Hot
- Cool

The *Transaction-optimized* tier is ideal for an application that needs to have Azure Files as back-end storage. This tier does not provide latency as low as what we can get from the Premium tier. The *Hot tier* is designed and optimized for most general workloads, such as company file shares or storing data that are not used frequently. It is a good candidate for Azure File Sync, where caching will occur on the Windows Server side. For 1 TB of data in the Azure Files Hot tier, we have to pay around $30 per month, and in comparison with higher tiers, that is a significant saving. The *Cool tier* is a cost-effective Azure Files offer that could be a good option for Azure File Sync with a nonintensive workload or a SMB or NFS accessible online archive. The Cool tier is twice cheaper than the Hot tier, so $15 per TB of data is an affordable option.

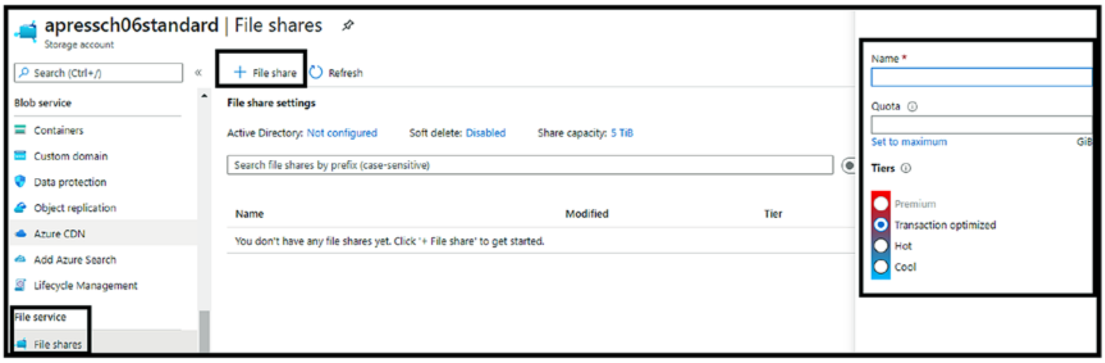
Because this functionality is pretty new, at the time of writing this chapter, there is no option to change tiering on the file level. This is possible only on the file share level, but it will take some costs, depending on the number and size of files and generated transactions.

Creating Azure Files

Once we have created a storage account, creating a file share is a straightforward process and can be done using all management tools. The process is the same, regardless of whether we have deployed a general-purpose or FileStorage account type.

Azure Portal

When we are in the desired storage account user interface, in the left pane, we have to find *File shares* under *File service*. We have to click **+ File Share** and define the file share name, quota in GB, and tier. These steps are shown in Figure 6-3.



Active Directory: Not configured Soft delete: Disabled Share capacity: 5 TiB

You don't have any file shares yet. Click '+ File share' to get started.

Figure 6-3. *Define the file share name, quota in GB, and tier*

ARM Template, PowerShell, and Azure CLI

Since the deployment code could be pretty big, all ARM template, Azure PowerShell, and Azure CLI scripts are stored in the Apress GitHub account, available at the following URL:

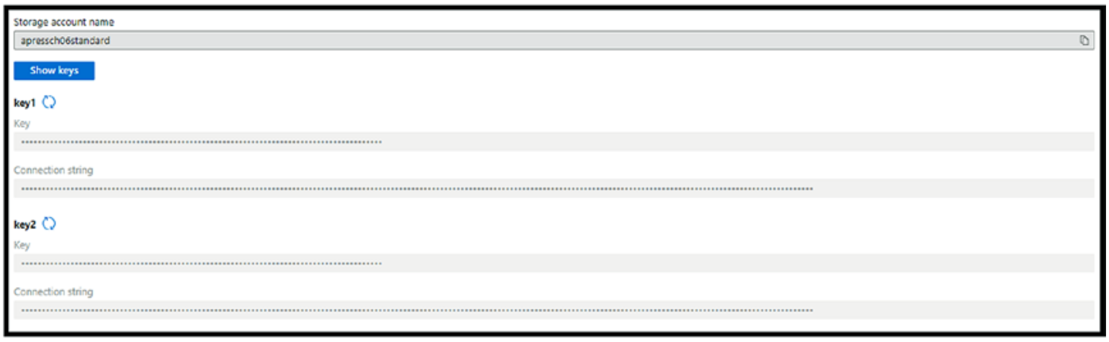
`https://github.com/Apress/pro-azure-admin-and-automation`

Storage Account Security

Even though the storage account is part of the PaaS offering in Microsoft Azure and a bigger part of security is Microsoft's responsibility, there are still many things that we need to think about in terms of security. The *shared responsibility model* that is in use when we talk about security in the cloud provides us with a certain amount of security, but in most cases, there is a need for a "final touch" from our side. For instance, if we share the storage account access key or improperly configure access to the blob containers or blobs, data loss will happen sooner or later. There are a few options that we must analyze and configure based on our needs, so let's start with exploring them.

Storage Access Keys and Shared Access Signature

Storage access keys provide the application with the possibility to authenticate the storage account when making requests. Even though this is not the best idea in a production environment, mainly because the access keys will give the application full access to the whole storage account, sometimes this is the only possible solution. Every storage account has two access keys that can be used for the same level of access. Due to the nature of permissions provided by access keys, storing those keys in the Key Vault *(a resource that will be explained in Chapter 10)* and regenerating the keys regularly is highly recommended. All information about the access keys and connection strings is visible in the storage account pane, under the section **Settings**, as shown in Figure 6-4.



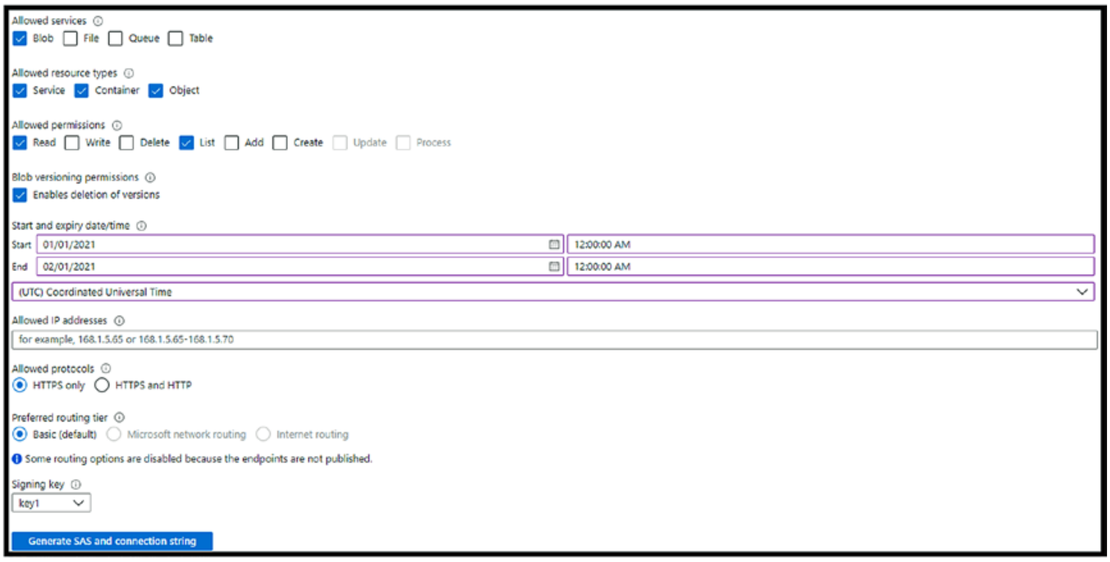
Storage account name
apressch06standard

key1
Key
Connection string

key2
Key
Connection string

Figure 6-4. *Information about the access keys and connection strings is visible in the storage account pane*

Another option to give someone access to a storage account, which is highly preferable whenever is possible, is the *shared access signature (SAS)*. This option provides us with the ability to define permissions more granularly and avoid unneeded access to our data in the storage account. Using this feature, we can define what services and resource types will be included, the kind of permissions, and start and end times for specific permissions. Additionally, we can define if only HTTPS access will be allowed and if there is any IP-based restriction that will be included in the SAS. If we want to create a shared access signature, we can find this feature under the storage account's **Settings** section. As shown in Figure 6-5, we are creating a SAS that allows us to read blob objects only between January 1 and February 1.


Allowed services
[✓ Blob] File Queue Table

Allowed resource types
[✓ Service] [✓ Container] [✓ Object]

Allowed permissions
[✓ Read] Write Delete [✓ List] Add Create Update Process

Blob versioning permissions
[✓ Enables deletion of versions]

Start and expiry date/time
Start 01/01/2021 12:00:00 AM
End 02/01/2021 12:00:00 AM
(UTC) Coordinated Universal Time

Allowed IP addresses
for example, 168.1.5.65 or 168.1.5.65-168.1.5.70

Allowed protocols
[✓ HTTPS only] HTTPS and HTTP

Preferred routing tier
[✓ Basic (default)] Microsoft network routing Internet routing

Some routing options are disabled because the endpoints are not published.

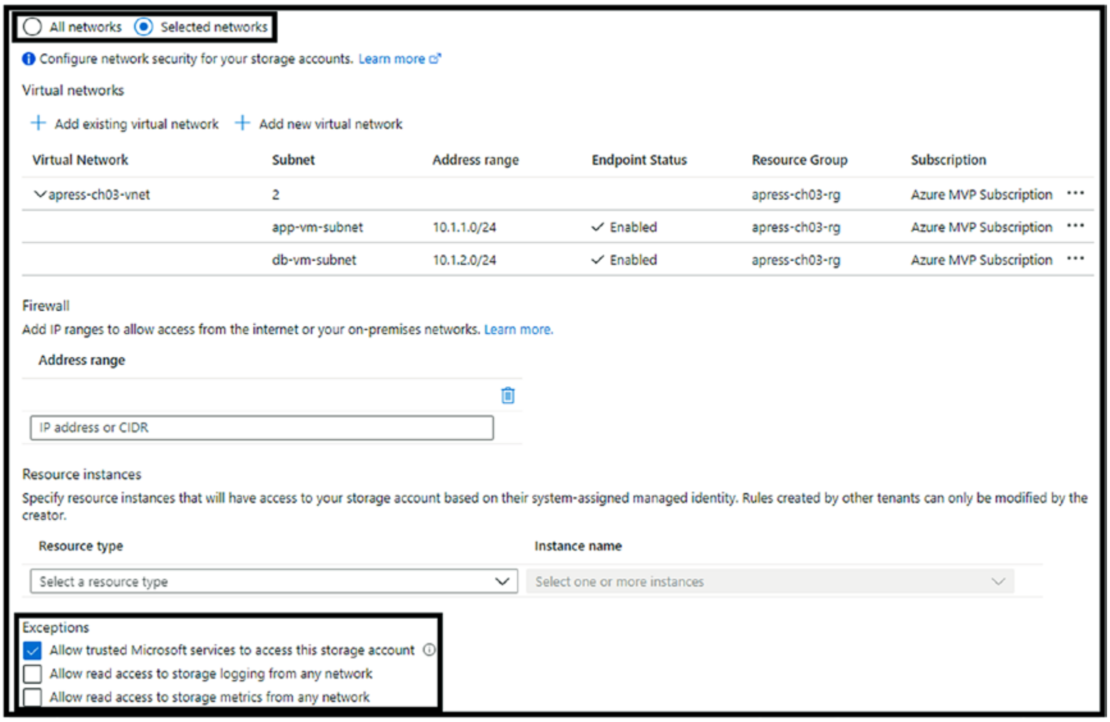
Signing key
key1

Figure 6-5. *Creating a SAS that allows us to read blob objects between January 1 and February 1*

Storage Networking

Regardless of what type of storage access we have chosen to implement, the storage account could be protected on the network layer as well. By default, the storage accounts are accessible by the HTTPS protocol only, and this should not be changed, although it is possible. Removing a security layer from the HTTP protocol is not the best idea because all data will be transferred without encryption. Also, by default, the storage account allows access from any network. That means every user or application with permissions to access the storage account will be able to make requests to it without any restrictions. In most scenarios, that is not necessary, and network access should be narrowed to specific networks.

In Chapter 3, we explained services and private endpoints for the services. A storage account is one of the services that could leverage these features to provide better network isolation. Once we decide to use these features, we have to navigate to **Networking** under the section **Settings** in the storage account. We can start with configuring network access by switching from *All networks* to *Selected networks*, and more options will be available then, such as connection with specific virtual networks, adding IP address ranges to the firewall, or selecting a specific resource that will have access to the storage account. Additionally, if we block all network traffic to the storage account, we can still make some exceptions and allow trusted Microsoft services access to the storage account or read access for logging and metrics purposes *(see Figure 6-6)*. The private endpoint is also available for the storage account. As for the other resources that support this feature, it provides us with the ability to create a private IP address for communication with other Azure resources in a more secure and isolated manner.



All networks [✓ Selected networks]

Configure network security for your storage accounts. Learn more

Virtual networks

+ Add existing virtual network + Add new virtual network

Virtual Network	Subnet	Address range	Endpoint Status	Resource Group	Subscription
apress-ch03-vnet	2			apress-ch03-rg	Azure MVP Subscription
	app-vm-subnet	10.1.1.0/24	✓ Enabled	apress-ch03-rg	Azure MVP Subscription
	db-vm-subnet	10.1.2.0/24	✓ Enabled	apress-ch03-rg	Azure MVP Subscription

Firewall

Add IP ranges to allow access from the internet or your on-premises networks. Learn more.

Address range

IP address or CIDR

Resource instances

Specify resource instances that will have access to your storage account based on their system-assigned managed identity. Rules created by other tenants can only be modified by the creator.

Resource type	Instance name
Select a resource type	Select one or more instances

Exceptions

- [x] Allow trusted Microsoft services to access this storage account
- [ ] Allow read access to storage logging from any network
- [ ] Allow read access to storage metrics from any network

Figure 6-6. *Allow trusted Microsoft services access to the storage account or read access for logging and metrics purposes*

Blob Container Access

Blob containers have their mechanism for controlling public access that could be combined with the aforementioned security methods. When creating a blob container, we have to choose one of the three public access levels: *Private (no anonymous access)*, which is the default option, allows access to data in blob containers only to storage account owners. Of course, if someone has storage account access keys, they will be classified as the owner, and data will be visible, as well as if there is SAS with appropriate permissions. *Blob (anonymous read access for blobs only)* allows anonymous users to have access to blobs only. This option is useful if we want to make data available for access or download without requiring access keys or SAS. In this case, only blobs can be read by anonymous users, whereas container data are not available, and anonymous users cannot enumerate blobs in the container. The third option, *Container (anonymous read access for containers and blobs)*, is the least restrictive and allows the anonymous to read and list blobs in the whole container.

Blob public access can be configured on the storage account level and the single container level. All options that we mentioned in this section are part of the single container configuration. By default, on the storage account level, blob public access is enabled, giving us the possibility to configure blob containers differently. If we disable blob public access on the storage account level, public access will not be allowed, even if containers and blob access levels are set to public. Even though this is highly recommended, we have to ensure that our application can work with this configuration.

Encryption

Storage accounts provide us with encryption of data in transit as well as at rest. We have already mentioned that secure transfer is enabled by default for all storage accounts, which means that we must connect using HTTPS or SMB with encryption. All other connection requests will be rejected. Along with secure transfer, which will encrypt data in transit, storage accounts support encryption of data at rest automatically. Once data is written to Azure datacenters, it will be encrypted and automatically decrypted when we want to access it. By default, data is encrypted by Microsoft-managed keys, but customer-managed keys are also available, but only for blobs and files inside the storage account.

If data encryption on the service level is not something that is enough for specific scenarios, an additional level of encryption can be included. If *infrastructure encryption* is enabled, data will be encrypted twice, once on the service level and another on the storage infrastructure level, but with different keys. This option must be enabled during storage account creation and cannot be added later. To use this feature, we have to register it for our tenant using PowerShell or Azure CLI and then re-register the Azure Storage resource provider.

Azure PowerShell

```
Register-AzProviderFeature -ProviderNamespace Microsoft.Storage
-FeatureName AllowRequireInfraStructureEncryption

Register-AzResourceProvider -ProviderNamespace 'Microsoft.Storage'
```

Azure CLI

```
az feature register --namespace Microsoft.Storage --name
AllowRequireInfraStructureEncryption
```

```
az provider register --namespace 'Microsoft.Storage'
```

As we already said, communication between a client and a storage account is encrypted. Transport Layer Security (TLS), as a standard cryptographic protocol, is used to ensure integrity and privacy between parties. Although Azure Storage uses TLS 1.2, it supports backward compatibility with TLS 1.1 and TLS 1.0. If we want to force clients to use the latest version of TLS, we can change a specific configuration setting during the storage account creation process or later at any time. We just have to be careful with this change for the existing storage accounts because a lower TLS version will not be allowed.

Data Transfer

Once we have implemented a storage account, the next logical step is to have some data in that storage account. If the storage account is used programmatically, in most cases, our application will be responsible for the whole process, which includes uploading, downloading, data management, and others. Nevertheless, in some cases, such as storing offsite backup files or using Azure Files, we are responsible for transferring data to Azure. There are many different options to transfer data to the cloud, and choosing the right tool depends on the data amount.

Storage Explorer

Storage Explorer is a standalone GUI-based application that could be installed on Windows, Linux, and macOS operating systems in order to make storage account management more manageable. Using this tool, we can upload/download files to/from blob containers and Azure Files, create/get SAS and access keys, or even copy files between containers or shares. Storage Explorer is designed to analyze storage accounts in the first place, but some of the management tasks, such as uploading data to storage accounts, could be performed by this tool. It is also available as a storage account feature in Azure Portal, but since it is still in preview, there are only specific features, including uploading and downloading blobs and files. We have to keep in mind that transfer will go through the network, and if we have a large amount of data, that will not be the best possible option.

AzCopy

Another tool that allows us to transfer files over the network to Azure is the command-line tool AzCopy. It allows us to copy blobs or files to and from the storage account, with many additional features that give us more possibilities. It also can be installed on Windows, Linux, and macOS operating systems, which gives us the possibility to use it in the same way on various operating systems. AzCopy must be installed on the operating system, and then the first step is to authorize AzCopy to our subscription. That could be done using Azure AD, or we can choose to use a SAS token for transferring data. If we want to use Azure AD, we can use user identity, managed identity, or service principal. Otherwise, the SAS token that we will use for AzCopy must have appropriate permissions for the action that we want to take.

AzCopy gives us the possibility to upload or download data, but also we can use AzCopy for synchronizing our data to the cloud. For example, if we want to upload a local directory to a blob container, we need to run the following command:

```
azcopy.exe copy 'C:\my-data\' https://apressch06standard.blob.core.windows.
net/uploaded-data/SASTOKEN --recursive
```

If our scenario requires us to synchronize blobs or files periodically or constantly from the local directory to Azure or vice versa, we can use the AzCopy functionality *synchronize*. With this functionality, AzCopy will compare the name and last modified values of the files and synchronize only if some of these values are new. By default, AzCopy will not delete files on the destination if they are deleted on the source. In case that we want to enable that option, we have to add the parameter `--delete-destination=true`, and AzCopy will remove deleted files on the destination automatically, or we can set that parameter to value `prompt`, and we will be asked to confirm deleting files from a destination:

```
azcopy.exe sync 'C:\my-data\' https://apressch06standard.blob.core.windows.
net/sync-data/SASTOKEN
```

By default, AzCopy will use Internet bandwidth as much as possible, and if we have a large amount of data to transfer, it could be a potential problem. However, if we add the parameter `--cap-mbps`, we can define upload or download bandwidth.

Data Box Gateway and Azure Stack Edge

Azure Stack Edge is a physical network appliance that acts as a storage gateway, creating a link between the on-premises site and Azure Storage. Using this physical device, which provides a local cache and optimizes network traffic, sending data into an Azure storage account and out from it is much more comfortable. AI-enabled computing capabilities included in Stack Edge analyze, process, and transform on-premises data before uploading it to the cloud. On the other hand, Data Box Gateway is a virtual device provisioned in the on-premises virtual environment that enables seamlessly sending data into an Azure storage account. All data will be sent to Data Box Gateway using the NFS or SMB protocol, and cached and processed data will be then transferred to an Azure block blob, a page blob, or Azure Files.

Both solutions are used for continuous ingestion of a large amount of data to an Azure storage account from on-premises networks, where caching and background transfer are essential.

Import/Export Service

If we need to migrate dozens of terabytes of data from an on-premises network to an Azure storage account within a reasonable period, we must use offline methods for transferring data. One of them is Azure Import/Export, which transfers a large amount of data to Azure securely by sending drives to the Azure datacenter. The same service could be used to collect all data from a storage account and send drives to an on-premises datacenter.

The process of sending data to an Azure storage account using the Azure Import service consists of several phases that must be completed in order. We have to provide our driver and comply with the strict procedure before drives are shipped to the Azure datacenter:

- A customer prepares drives using the WAImportExport tool.
- A customer encrypts drives using BitLocker.

- A customer creates an import job in Azure Portal.
- A customer ships drives to the Azure datacenter.
- Drives are processed at the Azure datacenter.
- Data are copied to the storage account.
- Drives are packaged for return shipping.
- Drives are shipped back to the customer.

The process of getting data from a storage account has similar phases but in a reverse way:

- A customer creates an export job in Azure Portal.
- A customer ships drives to the Azure datacenter.
- Drives are processed at the Azure datacenter.
- Data are copied to the drives and encrypted with BitLocker.
- Drives are packaged for return shipping.
- Drives are shipped back to the customer.

The Azure Import/Export service is limited to ten drives, and the time needed to complete this process is between 7 and 10 days. Also, although there is no charge for the data transfer, we need to know that device shipping to the Azure datacenter and back is our expense, as well as the device handling at the Azure datacenter.

Data Box

If we do not buy drives for sending data to Azure or we have more data to transfer, the Import/Export service most probably will not be suitable for us. Fortunately, we can still plan to transfer data to the Azure datacenter offline because Azure can provide us with *Data Box* or *Data Box Disk*. If the amount of our data is not larger than 35 TB, we can order Data Box Disk, and we will get from Microsoft up to five SSD disks of 8 TB. All disks have a USB 3.0 interface, which guarantees that transfer will be quick, and they are secured by AES 128-bit encryption all the time. Once we order Data Box Disk, we will get the key in Azure Portal, which must be used for unlocking disks that we will receive. After transferring data to the storage account, all disks will be wiped to comply with the NIST 800-88r1 standard.

In a scenario where we need to upload more than 35 TB of data, we have to use Azure Data Box. Data Box is an inexpensive and reliable way to transfer a large amount of data to the Azure datacenter by using a proprietary device provided by Microsoft, limited to 80 TB. Data Box is equipped with 1 Gpbs or 10 Gbps network interfaces, allowing fast data transfer to the device. From a security perspective, Data Box comes with AES 256-bit encryption all time. It can be unlocked only with a key provided in Azure Portal during ordering of Data Box. Like Data Box Disk, once data are transferred to the storage account, the device will be wiped to comply with the NIST 800-88r1 standard.

Of course, if we need to transfer more than 80 TB of data, we can order more than one Data Box device or order Data Box Heavy, allowing us to send up to 770 TB of data to the Azure datacenter. The process is pretty much the same as we have for other Data Box options, which includes ordering devices, returning with copied data, and uploading data to a storage account. Like Data Box, the rugged device case is protected by AES 256-bit encryption all the time and can be unlocked only with a key provided in Azure Portal during ordering.

Storage Account Management

Once we have implemented a storage account for any kind of workload, in most scenarios, we can say that we have completed the solution. Nevertheless, in reality, we have to perform the storage account maintenance tasks or even make some reconfiguration. Luckily, some of these actions could be done at any time, without affecting stored data nor causing downtime.

Changing Security Parameters

In every storage account, as we already said in this chapter, there are many security configurations. Almost all of them can be changed at any time if they are not adequately configured, or we need to change them due to new company policies or compliance.

Access Keys and Shared Access Keys

If we suspect that the storage account access keys are compromised, we can regenerate them at any time quickly. Of course, we have to know that all previous connections using these access keys will stop working and we need to replace these keys.

On the other hand, if we use SAS tokens for communication with the storage account and want to block that kind of access, we have two potential options: remove the SAS policy that defines SAS tokens or disable shared access key access.

Storage Networking and Encryption

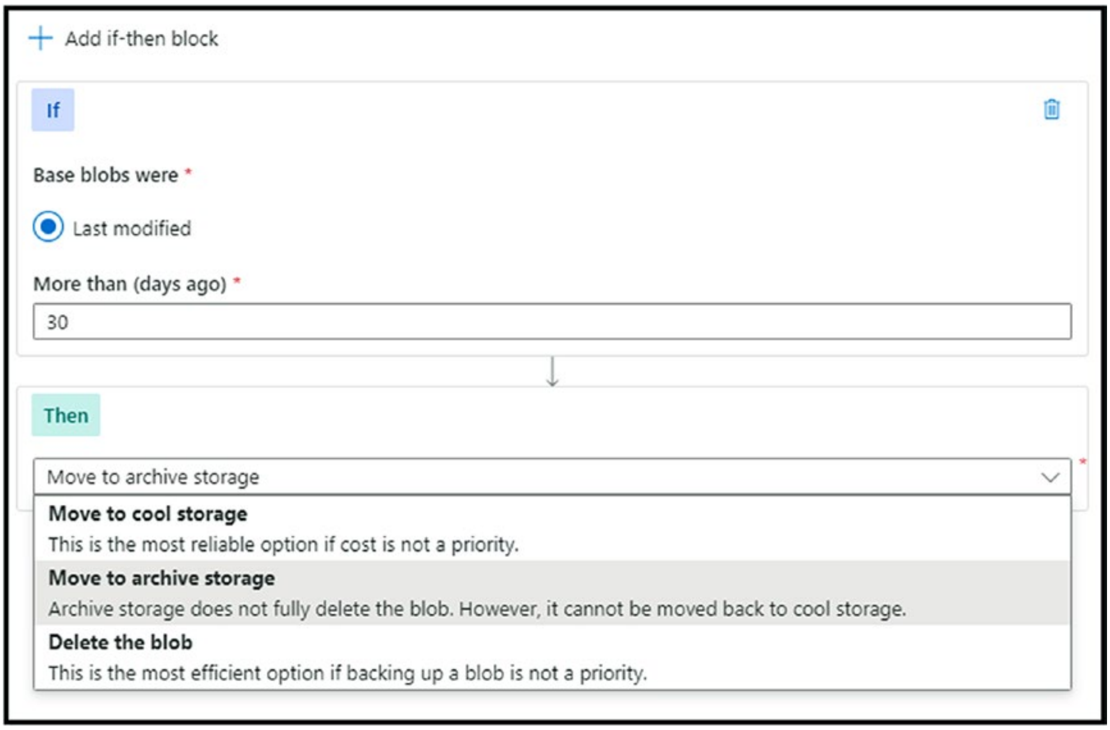
For networking for storage accounts, we already explained possible options that we can implement. All of those options can be reconfigured quickly, and we can allow additional access to the storage account or even remove unneeded access. We already said that the storage account supports HTTP and HTTPS traffic, but restricting access to HTTPS only is highly recommended. Although allowing HTTP traffic is not the best idea, it is possible if there is a specific need for it. Also, by default the minimum TLS version that is supported is 1.0. At any time, we can change that value to version 1.1 or 1.2 and increase security for the storage account. Of course, if the TLS version is already set to 1.2 and we need to decrease it, it is also possible.

Storage Replication

As we already explained in this chapter, storage accounts can have different types of replication. If we initially set locally redundant replication for our storage account, we can change that at any time and select geo-redundant or read-access geo-redundant replication. At the moment, there is no possibility to switch from locally redundant to zone-redundant if an account is not initially created to support zones.

Lifecycle Management

When we talk about data management in storage accounts, blob containers especially, we have to think about data retention and tiering. For instance, we have a large amount of data on a daily basis sent to the storage account, but we do not need these data after 30 days, or we need to keep these data but only for archiving purposes. Luckily, lifecycle management functionality, which is part of the blob service, provides us with the ability to create specific rules that will manage data based on our plan. Very quickly and easily, we can configure one or more rules, and we do not think about data management. At the time of writing this chapter, only block and append blobs are supported by this functionality. This functionality is based on the *if-then* model. The *If* statement is related to the blob's *Last modified* attribute with the *More than* operator. The *Then* statement gives us three options, as we can see in Figure 6-7, which are more than enough for 99% of cases: Move to cool storage, Move to archive storage, and Delete the blob.



Base blobs were *

[✓ Last modified]

More than (days ago) *

30

Move to archive storage

Move to cool storage
This is the most reliable option if cost is not a priority.

Move to archive storage
Archive storage does not fully delete the blob. However, it cannot be moved back to cool storage.

Delete the blob
This is the most efficient option if backing up a blob is not a priority.

Figure 6-7. *The "Then" statement provides a prompt for three options*

Managed Disks

A managed disk is another essential service from the storage family, even though it is not directly related to storage accounts from an end user perspective. Since every virtual machine has to be equipped with an OS disk at least, and sometimes with data disks as well, the importance of managed disks is inevitable. Also, a managed disk is not mandatory for virtual machines in Azure because we can use a storage account for storing VHD files that are attached to virtual machines. Nevertheless, performance and management tasks are the most significant advantages from using managed disks because managed disks are block-level storage volumes that Microsoft Azure fully manages with 99.999% of availability.

Types of Managed Disks

If we talk about types of managed disks, we have four categories, based on their performances and the underlying storage in use: ultra disk, premium SSD disk, standard SSD disk, and standard HDD disk.

Standard HDD Disk

If we do not want to spend a significant amount of money on our disks or we have a workload that is not I/O intensive, the standard disk could be an ideal choice for us. All data are stored on the HDD, and latency will be up to 20 ms for read operations and 10 ms for write operations. IOPS and throughput performances are not guaranteed and may vary more significantly than SSD-based disks, so these disks are not intended for any production nor I/O-intensive workload. On the other hand, standard disks are more affordable than SSD-based disks, so we can save much money for dev or test environments. For instance, the S10 disk (128 GB) will cost less than $6 per month. One crucial thing that we need to consider is the number of storage transactions. Once we pay $6 for the 128 GB standard disk, we will be charged for the additional storage transactions. Although these transactions are not expensive and 1 million transactions will cost just 5 cents, we need to consider that as unknown costs at the starting point. For smaller disks, that will not be a big problem, but if we have large disks (1 TB or more), sometimes the transaction cost could be higher than the pricing for disk size.

Standard SSD Disk

If we know that our on-premises workload, which has been or will be moved to Azure, has an issue with disk operations, we maybe need to move that workload to SSD-based disks. Standard SSD disks are cost-effective disks that provide consistent performances on the lower level in terms of IOPS and throughput. For the workloads that are not so intensive, like small websites or blogs, standard SSD disks could be an ideal entry point with a balance between costs and performances. From a pricing perspective, standard SSD disk E10 (128 GB) costs $10, which is more than 50% higher than the cost of the standard HDD disk. Transaction costs apply for the standard SSD disk in the same manner as for the standard HDD disk, but with a different pricing model. For 1 million transactions, we have to pay 20 cents, which is still very cheap, but at the same time is four times expensive than transaction costs for the standard HDD. For smaller disks, that will not be a big deal, but we have to be careful if we have larger disks (1+ TB) with many transactions, such as a SQL Server development environment.

Premium SSD Disk

If a standard SSD disk is not good enough for our workload and we need something better, it is not a problem, because we can use a premium SSD disk. Azure guarantees low latency and high performance for premium SSD disks, so we know what we can expect and whether that will be okay for our workload. This disk type is designed to support mission-critical workloads, such as I/O-intensive applications or SQL Server workloads. Also, with premium SSD disks, we can plan our costs. Although the initial price is higher, almost double that of standard SSD disks, there are no transaction costs. For instance, a P10 disk (128 GB) will cost $20 per month, and there are no additional hidden costs, regardless of the number of transactions.

While performance tiers for standard HDD and SSD disks are defined as “up-to” for IOPS and throughput, premium SSD disks come with guaranteed performances. For example, the aforementioned P10 disk provisions 500 IOPS and 100 MB/sec throughput. In addition, some of the performance tiers have enabled the feature *bursting*, which allows disks to have a much better performance for 30 minutes during the day. That allows us to have a better performance for some spikes without the need to change the performance tier, which could help us to save money. The P10 disk, for instance, can reach 3500 IOPS and 170 MB/sec throughput if there is a need for bursting for specific actions. At the moment, bursting is allowed only for P20 (512 GB) disks and lower.

In contrast, P30 disks (1 TB) and higher can be reserved for 1 year. That option allows us to plan expenses accordingly, especially if we have many big disks. Although the savings with reservation is approximately $7 per month per TB of disk space, let’s imagine that we have 50+ TB in provisioned premium SSD disks and calculate what the monthly savings will be. Moreover, there are still no transaction costs.

Ultra Disk

We are still struggling with our workload performance, even though we have implemented the premium SSD disks. For this, the ultra disk should be our choice. The ultra disk is designed to deliver high throughput, high IOPS, and consistently low latency. Also, we can dynamically change disk performance without the need to reboot a virtual machine, which is a mandatory task for all other disk types. Transaction-heavy workloads, such as SAP HANA, SQL, or Oracle, are ideal candidates for ultra disks.

Even though max throughout is more than doubled and max IOPS is eight times higher than the premium SSD disk, the ultra disk pricing plan cannot be declared advantageous. At the same time, the pricing model is pretty complicated, and we need to pay for capacity, IOPS, and throughput in combination. For instance, the P30 disk (1 TB) provisions 5000 IOPS and 200 MB/sec throughput, with a price of $135 per month. If we need higher performance, we can move our disk to a higher performance tier, but performance will not be doubled, and the price will be doubled. Alternatively, we can use the ultra disk and define the performance that we need. For example, if we need a 1 TB disk, with 10000 IOPS and 500 MB/sec throughput, we have to pay approximately $1100 per month. In that case, we must have a perfect reason to use the ultra disk for our workload.

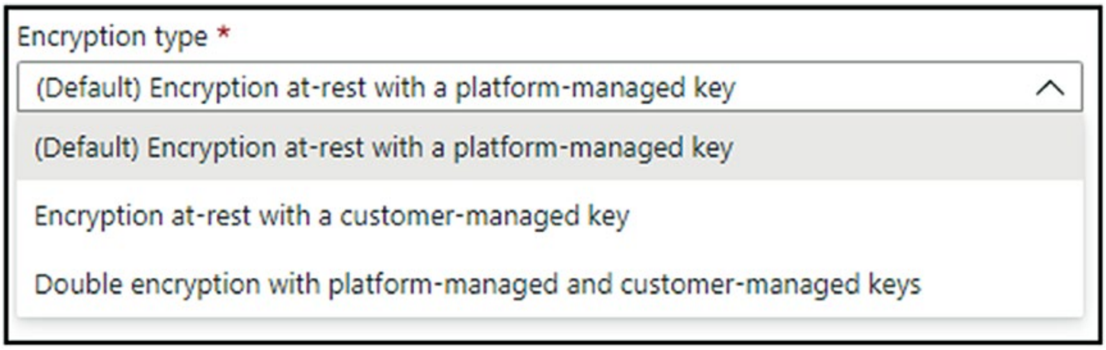
Performance Tiers

All disk types, except ultra disks, have their performance tiers. It is essential to know what that means and how it will affect our billing at the end of the month. We have already said that for standard HDD and SSD disks, we have to pay for the performance tier and the transactions as well, whereas the premium SSD disks do not count transactions as billable. When we select a performance tier, regardless of disk type, we pay for the performance, not for the capacity. That means that if we use the P30 (1 TB) performance tier, we will pay for 5000 IOPS and 200 MB/sec throughput, with a maximum capacity of 1 TB. If we need a disk of 1.5 TB, we need to move our disk to the next performance tier (P40), which will give us 7500 IOPS and 250 MB/sec throughput, with a maximum capacity of 2 TB. Even though we do not use the maximum capacity, we have to pay a full price. In some scenarios, that is not the best possible solution, especially if we do not need better performance or we just need to increase disk capacity for 100 GB or so.

If we are using Windows Server for our workload, we can leverage the feature *Storage Spaces* to increase disk performance for almost the same price. This feature, which is similar to RAID, allows us to group more disks in a pool to get better resilience or performance. For instance, if we add two P30 (1 TB) disks to a virtual machine and configure Storage Spaces to use both disks in the simple mode (similar to striping in RAID), we will get one disk of 2 TB, which is the same as the P40 performance tier. Nevertheless, the maximum IOPS and throughput will be doubled from the P30 performance tier. Our new disk will have 10000 IOPS and 400 MB/sec throughput, which are higher than those of the P40 disk initially, and the price will be almost the same. This could be very useful for bigger disks, P50 (4 TB) and higher, because the performance tiers have a vast discrepancy in capacity, performance, and pricing.

Managed Disk Security

Like storage accounts, a managed disk also has a few security features that could be configured. In the first place, every managed disk is encrypted at rest, and that functionality cannot be disabled. By default, all managed disks are encrypted by the platform-managed key, but there is an option to use a customer-managed key or even combine them to double encryption layers. See Figure 6-8.


Encryption type *

(Default) Encryption at-rest with a platform-managed key

(Default) Encryption at-rest with a platform-managed key

Encryption at-rest with a customer-managed key

Double encryption with platform-managed and customer-managed keys

Figure 6-8. *The option to use a customer-managed key or combine them to double encryption layers*

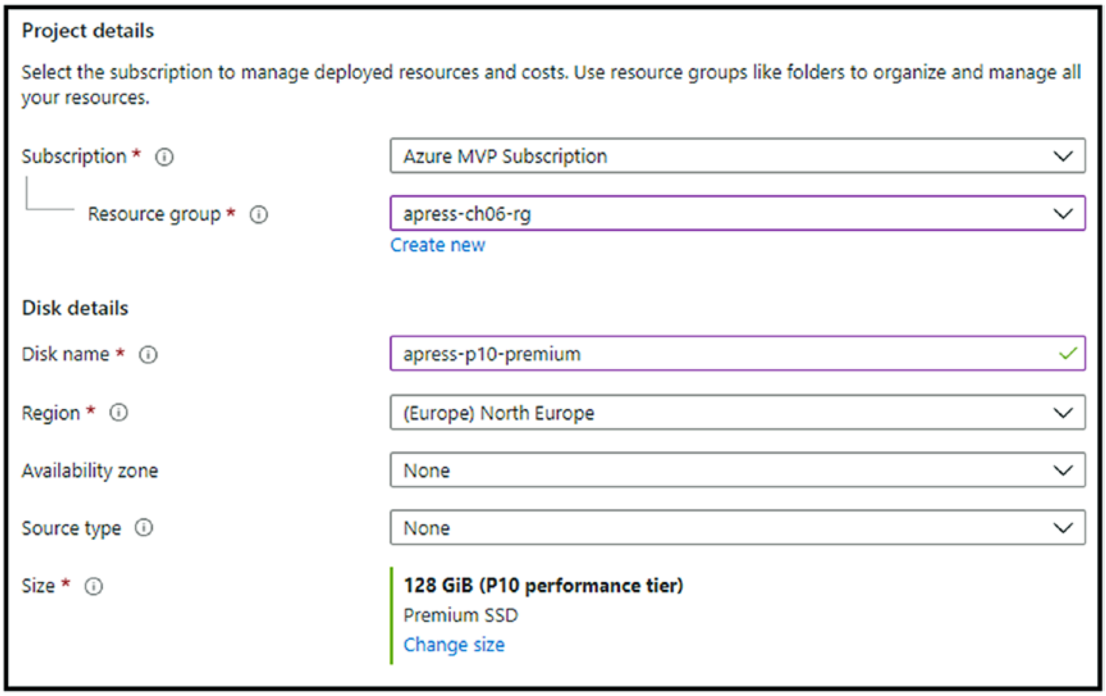
An additional option to secure access to a managed disk is network isolation. By default, the connectivity method is configured to *Public endpoint*, which allows all networks in our subscription to use a managed disk. If we decide to use *Private endpoint*, it will narrow down what virtual networks can access the managed disk. In this case, we have to create *Disc Access*, which will create a private endpoint for the managed disk and allow access to the disk only from specific virtual networks. As a last option, we can select *Deny all* and block access to the managed disk completely.

Creating a Managed Disk

Like almost all Azure services, a managed disk can be created using all management tools, and the process is straightforward.

Azure Portal

If we decide to use Azure Portal to perform this operation, the first step is to click **+ Create a resource**, search for "managed disk" in the *Marketplace*, and click **Create**. As with other Azure resources, we need to define Subscription and Resource group under *Project details*. Under *Instance details,* we will define *Disk name*, which must be unique in the resource group, *Location*, *Availability Zone* if it is applicable, *Source type,* and *Size*. Once these parameters are filled, click **Review + create** and then **Create** to create a managed disk. All these parameters are shown in Figure 6-9. In this scenario, we will not take care of disk encryption and network isolation.



Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Azure MVP Subscription
Resource group *	apress-ch06-rg
	Create new

Disk details

Disk name *	apress-p10-premium
Region *	(Europe) North Europe
Availability zone	None
Source type	None
Size *	**128 GiB (P10 performance tier)** Premium SSD Change size

Figure 6-9. *Enlisting Azure Portal to perform the operation*

In our cases, we will configure parameters just on the *Basics* tab, and all other things we will configure later in this book. As with all other Azure resources, we need to define Subscription and Resource group under *Project details*. Under *Instance details,* we will define *Storage account name*, which must be unique across Azure, *Location*, *Performance tier*, *Account type,* and *Replication* type. Once these parameters are filled, click **Review + create** and then **Create** to create a storage account. All these parameters are shown in Figure 6-1.

ARM Template, PowerShell, and Azure CLI

Since the deployment code could be pretty big, all ARM template, Azure PowerShell, and Azure CLI scripts are stored in the Apress GitHub account, available at the following URL:

`https://github.com/Apress/pro-azure-admin-and-automation`

Chapter Recap

In this chapter, we have learned what Azure Storage is and how we can use it, especially because this is one of the most used services in Azure. Even though we can find more storage options in Azure, such as NetApp Files, a storage account will be the first choice for most scenarios, without a doubt. A storage account is suitable for various workloads and types of transfer. Regardless of whether the interaction method is programmatical or in any other way, one storage account can meet all our needs.

In the next chapter, we will learn about advanced networking functionalities in Azure that are important for hybrid environments and intersite connectivity, regardless of whether we need to enable communication between two or more Azure networks or between on-premises and Azure networks.