# AZ-104T00A – Administer Azure Resources

Welcome to this new exciting module of our AZ-104 course. Last week we discussed in the first lesson Azure Identity with Azure Active Directory and User & Groups Management. Then in the second lesson we talked about Governance and compliance with Subscriptions, Azure Policy and Role-Based Access Control. Do you have any questions? For all those who were not able to participate because they didn’t receive the invitation, we are going to schedule some recovery sessions. I’ll keep you posted about this. Today, we are going to talk about Azure Resources and how to manage them. We’ll start discovering Azure service portfolio, we are not going to cover all the services, because they are more than 100, but we’ll focus on the main ones. Then we’ll discuss Azure Resource Manager. It’s the foundation of the management layer in Azure. All of the other tools are going to work through the Azure Resource Manager. Whether you are working with Azure Portal or the Cloud Shell, PowerShell, the command line interface, the Azure mobile app or even if you design your own Rest API based tools to get access to Azure, all of those tools are being channelled and handled by the Azure Resource Manager. ARM takes requests from tools and translate them to a very specific language, called JavaScript Object Notation (or JSON for friends). Keep in mind that when Azure was launched, the only way you had to manage it was through PowerShell. Over time, Microsoft has greatly increased the tools that allow us to manage our Azure environment.

## Azure Resources Overview

Azure Resources

In the first part of this lesson, we are going to see, the most used resources that we can manage in Azure starting from the three pillars of any IT infrastructure. Compute, Storage, and Networking.

Azure Compute service

Let's start with Compute. In Azure's early days, Microsoft offered only one type of compute service: virtual machines, or VMs for short. These are machines that run either Windows or Linux. If you currently have an application running on a Windows or Linux server, then the most straightforward way to migrate it to Azure is to do what's called a "lift and shift" migration. That is, you simply lift the application from your on-premises server and shift it to a virtual server in the cloud. Azure VMs are known as Infrastructure-as-a-Service because they're traditional IT infrastructure components that are offered as a service. Later, Microsoft came out with what's known as a Platform-as-a-Service offering called Azure App Service. This platform lets you host web and mobile applications without having to worry about the underlying infrastructure. After doing a minor amount of configuration, you can just upload your code to an App Service instance and let Azure take care of the details. In most cases, this is a better solution than using virtual machines, but there are times when it makes more sense to use VMs. For example, if you have an application that's not a web or mobile app, then you can't use App Service, so you'll have to use a VM. These days, the hottest compute technology is containers. These are self-contained software environments. For example, a container might include a complete application plus all of the third-party packages it needs. Containers are somewhat like virtual machines except they don't include the operating system. This makes it easy to deploy them because they're very lightweight compared to virtual machines. In fact, containers run on virtual machines. Microsoft provides many ways to run containers. The simplest way is to use Azure Container Instances. This service lets you run a container using a single command. If you have a more complex application that involves multiple containers, then you'll probably want to use Azure Kubernetes Service, which is what's known as a container orchestrator. It makes it easy to deploy and manage multi-container applications. Before we move on, I should mention one more compute service. It's called Azure Functions, and it's Microsoft's main "serverless" offering. Azure Functions is kind of like Azure App Service except that it executes individual functions rather than entire applications, and you only pay for it when it gets used. When you provision an App Service instance, it runs until you shut it down, and you pay for it the whole time it's running. Although it's possible to configure Azure Functions in the same way, it's usually better to use the Consumption plan, which means that it only uses resources when a function is running, so you only pay when a function is running.

Azure Storage service

Now let's move on to storage. The simplest form of storage is called Blob storage. It's referred to as object storage, but really, it's just a collection of files. It's not like a normal file system, though, because it doesn't have a hierarchical folder structure. It has a flat structure. It's typically used for unstructured data, such as images, videos, and log files. One of the great things about it is that it has multiple access tiers: hot, cool, and archive. The hot tier is for frequently accessed files. The cool tier is for files you expect to access only about once a month or less. The advantage is that it costs less than the hot tier as long as you don't access it frequently. The archive tier is for files that are rarely accessed, such as backup files. It has the lowest storage costs but the highest retrieval costs. It also takes several hours to retrieve files from the archive tier. If you need hierarchical file storage, there are a couple of options. The one that will probably seem more familiar is Azure File Storage, which is like a typical SMB file server. It serves up file shares that you can mount on Windows servers. The less familiar option is Azure Data Lake Storage Gen2. This is Hadoop-compatible storage for use with data analytics applications.

Azure Database service

In an on-premises Microsoft environment, SQL Server is the most commonly used database. The cloud equivalent is Azure SQL Database. It's very similar to SQL Server, although it's not 100% compatible. If you need to run an open-source database, then Microsoft still has you covered. It offers Azure Database for MySQL, MariaDB, and PostgreSQL. All of these databases, including both SQL Database and the open-source options, are suitable for online transaction processing. On the other hand, if you need to build a data warehouse, then Azure Synapse Analytics is the best choice. If you release an application that attracts a very large number of users, you may find that a traditional relational database can't scale to meet the demand. One common solution is to use a so-called NoSQL database. These databases are designed to handle far more data than relational databases. However, in order to achieve that massive scalability, they have to sacrifice something, so they don't support all of the features of relational databases. Nonetheless, they have become a cornerstone of many cloud-based applications. Microsoft's main NoSQL offering is called Cosmos DB. It's an amazing database service that can scale globally. Another NoSQL service is Azure Cache for Redis, which is typically used to speed up applications by caching frequently requested data.

Azure Networking service

When you create a virtual machine on Azure, you have to put it in a virtual network, or VNet. A virtual network is very similar to an on-premises network. Each virtual machine in a VNet gets an IP address, and it can communicate with other VMs in the same VNet. You can also divide a VNet into subnets and define routes to specify how traffic should flow between them. By default, all outbound traffic from a VM to the internet is allowed. If you also want to allow inbound traffic, then you need to assign a public IP address to the VM. If you want VMs in one VNet to be able to communicate with VMs in another VNet, then you can connect the VNets together using VNet peering. By the way, so far, I've only been talking about virtual machines, but other resources, such as Kubernetes clusters, can be in VNets, too. Then, If you want to create a secure connection between a VNet and an on-premises network, then you can use either a VPN, which stands for Virtual Private Network, or Azure ExpressRoute. A VPN sends encrypted traffic over the public internet, whereas ExpressRoute communicates over a private, dedicated connection between your site and Microsoft's Azure network. ExpressRoute is much more expensive than a VPN, but it provides higher speed and reliability since it's a dedicated connection. Then we have Azure DNS, a hosting service for DNS domains that provides name resolution by using Microsoft Azure infrastructure. By hosting your domains in Azure, you can manage your DNS records by using the same credentials, APIs, tools, and billing as your other Azure services.

Other important services

DevOps suite called **Azure DevOps** with the most important service in this suite called **Azure Pipelines**, that lets you create automated workflows to continuously build, test and deploy code. **Azure DevTest Labs** which makes it easy to spin up non-production environments. You could do this in other ways, but DevTest Labs gives you some extra capabilities, such as allowing administrators to control costs by setting limits on how many VMs can be deployed at once and ensuring that VMs are shut down when they're not in use. One really helpful service for speeding up the responsiveness of your applications is **Azure Content Delivery Network**, which lets you take advantage of Microsoft's extensive global network. It caches your most frequently accessed content in locations around the world so your end-users will retrieve it from the closest point on the network. This really helps with making your web applications feel more like local applications. Microsoft offers a suite of services to help organizations connect, monitor, and control IoT devices. The simplest way to get started is to use **Azure IoT Central**, which is a fully managed SaaS solution that takes care of the technical details for you. It lets you create IoT applications without writing any code. Microsoft also offers a solution called **Azure Sphere** to make your IoT devices more secure. It includes certified chips, the Azure Sphere operating system, and the Azure Sphere Security Service, all of which provide layers of protection for your IoT devices. Microsoft offers lots of different AI services. If you're new to AI, then the best place to start is **Azure Cognitive Services.** This is a collection of pre-built artificial intelligence tools. These services let you add AI capabilities to applications even if you don't know anything about machine learning. They're grouped into five categories: decision, language, speech, vision, and web search. For example, the vision category includes the Computer Vision API, which can classify images, and the Face API, which can detect faces in images. A related offering is **Azure Bot Service**, which gives you the tools to create a chatbot. This is an intelligent agent that can answer questions. For example, you could create a chatbot to handle simple support requests from customers. If you have some basic knowledge of machine learning, then you might want to try **Azure Machine Learning Studio**. It lets you train and deploy machine learning models without any coding, using a drag-and-drop interface. I highly recommend it for learning the basics of machine learning. A much more sophisticated option is **Azure Machine Learning Services**, which gives you full control over every stage of the machine learning process. You can use any Python-based machine learning framework, such as TensorFlow or PyTorch, train models using services such as **Azure Databricks**, and deploy models using services such as **Azure Kubernetes Service**. Azure Machine Learning Services is usually the best solution when you need to build your own custom artificial intelligence application.

We see there are a lot of services available. Since we use so many applications and services, it would be nice to be able to perform certain tasks on them automatically. For example, suppose you have an Azure blob container that your customer uploads documents to, and you'd like to be notified by email as soon as one arrives so that you can respond to it as quickly as possible. Microsoft offers a service called **Azure Logic Apps** that lets you automate this sort of task without writing any code. You can create a logic app using a drag-and-drop interface. In this example, the logic app would be able to detect events that occur in Blob storage, but in most cases, you'd need to use another service called **Azure Event Grid** to notify your logic app of particular events. For example, if you want to get an email every time a virtual machine is created in a subscription, then you would configure Event Grid to send a message to your logic app whenever this occurs. These examples are actually pretty trivial. Azure Logic Apps can be used to build complex workflows involving not only Azure services but also third-party services, such as Twitter and Dropbox.

## Configure Azure Resources with Tools

Demonstration Azure Portal Setting Preferences

Now that we have seen most of Azure’s services, let’s review what tools are available to manage them. There are many ways to interact with Azure. The first we are going to show is the Azure portal. It runs in a browser, so you don't need to install anything to use it. If you're following along on your own account, go to portal.azure.com. You can change the default settings of the Azure portal to meet your own preferences. Most settings are available from the Settings menu in the top right section of global page header, this little gear icon. After exploring all the settings, we can customize a dashboard. After exploring the Dashboard creation, we can see how to create a support request.

Demonstration Azure Portal creating VMs

There are a few different ways to get to the place where you can create a VM. In this search box, you can start typing virtual machine and you'll see it come up at the top of the search results. Another way is to go into the menu on the left and select virtual machines. It takes you to the same place. Once you're here, you can click Create and then Virtual Machine. The first thing it wants to know is what subscription to put this VM in. It always sets this to your default subscription, so you can usually just leave it that way. Next, it's asking for the resource group. Last week we discussed about resource group and we said is a collection of resources, and that a subscription can have multiple resource groups so it's a way of further grouping the resources within a subscription. The best practice is to group related resources together, such as a VM and its associated storage account. Generally speaking, the resources in a group should be created and deleted at the same time, which makes sense if they are components that work together to provide a solution.

So, let's create a new resource group to hold this VM. I'll call it RG-test.

Now we need to give the VM a name. I'll call it test-vm01.

Next, we need to decide which region the VM should run in and we select SwitzerlandNorth.

You need to set an availability option if you want to make sure your application will still be available, even if Azure experiences an outage that affects this VM. The first option is "Availability zone". A region that supports availability zones has at least three data centers, each of which is called a zone. If you put VMs in multiple zones, then a datacenter outage won't take down your application, because your VMs and other zones will still be running.

It added another field called "Availability zone". We can choose Zone 1, 2, or 3. It doesn't really matter which one we choose for this VM, but if we were to create a second VM, we'd need to put it in a different availability zone. I'll choose "Zone 1".

Next, we need to choose a VM image. This is a copy of the disk that will be used for the VM. The choices are different versions of Linux and Windows, such as Ubuntu, Red Hat, and Windows Server. This is only a small subset of the images that are available. You can click on this link to see more. For example, suppose you want an image that includes not only Windows Server but also SQL Server, click on Databases. There are lots of SQL Server image options. For this example, we'll just go back and pick one of the basic options though. I'll pick the Ubuntu server option.

If you use an Azure spot instance for this VM, then the cost will be dramatically lower, but the VM might be shut down with only 30 seconds' notice. So, you should only use this option for non-critical workloads.

There are lots of options for the size of the VM. There's a size with only one virtual CPU, half a gig of memory, and four gigs of temporary storage. You can see that the cost per month is very low. Some of these other options are much bigger, but so is the cost. Since we're not actually going to use this VM, I'll choose the cheapest one.

Here, we need to create the administrator account. In most cases, you should use an SSH public key for authenticating to the VM. But to keep it simple for this demo, I'll use a password. You can name your administrator account almost anything.

The inbound port rules let you open up the VM to access from the internet. If this VM will be acting as a web server, then you'll want to open up ports 80 and 443. There's also an option to open up the SSH port so you can log into the VM remotely. The problem with opening up the SSH port here is that it will allow access from all internet addresses, which is dangerous. There are other more secure ways to give yourself access to the VM. So, you shouldn't select this unless you're just doing some testing. I'm going to set this to SSH just for this demo.

That's it for the basic settings, but there are a few more settings we should have a look at. We'll click this button to go to the disks tab. The operating system disk is set to premium SSD by default. That option has the highest performance, but it's much more expensive than the other two. I'll set it to Standard SSD.You can also add data disks if you want.

And now let's have a look at the networking options. By default, it creates a virtual network, a subnet inside that virtual network, and a public IP address. If we already had another virtual network, we could put it in there instead if we wanted to.

Okay, that takes care of the compute, storage, and networking options for this VM. So, let's create it. First, click "Review + Create". It says that our settings have passed validation testing. Then it summarizes all of our settings. Now click the Create button. It will take a little while to create. Everything is clear so far? Are you all still awake? Cool. After a minute we can find the Virtual Machine created.

Here it shows the details of this VM, such as its status, and its public IP address. Up here it has some controls that let you stop and restart the VM or even delete it. If we had set up SSH authentication, then we could connect to it from here too. In the menu on the left, there are all kinds of options for things like activity logs, security, backups, and monitoring. We can access this virtual machine with SSH through a terminal. We can copy the public IP address and connect. As you can see with lsblk command we have 2 disks /dev/sda and /dev/sdb mounted on /mnt. If we open the /mnt folder there is a text file with warning. This disk is a temporary and usually it is reinitialized when you redeploy a vm or when Microsoft has some service faults.

Demonstration Azure Cloud Shell

Azure Cloud Shell is an interactive, browser-accessible shell for managing Azure resources. It provides the flexibility of choosing the shell experience that best suits the way you work. Linux users can opt for a Bash experience, while Windows users can opt for PowerShell. Cloud Shell enables access to a browser-based command-line experience built with Azure management tasks in mind. You can use Cloud Shell to work untethered from a local machine in a way only the cloud can provide. Cloud shell is temporary and requires a new or existing Azure Files share to be mounted. Times out after 20 minutes without interactive activity. From here you can use both Azure CLI and Azure Powershell.

Demonstration Azure PowerShell

Azure PowerShell is a module that you add to Windows PowerShell or PowerShell Core to enable you to connect to your Azure subscription and manage resources. Azure PowerShell requires PowerShell to function. PowerShell provides services such as the shell window and command parsing. Azure PowerShell adds the Azure-specific commands. It is available in two ways: inside a browser via the Azure Cloud Shell, or with a local installation on Linux, MacOS or the Windows OS. In both cases you have two modes from which to choose: you can use it in interactive mode in which you manually issue one command at a time, or in scripting mode where you execute a script that consists of multiple commands. I’m going to show how to install on a windows machine since on my Mac I’ve already installed through homebrew. So, you have to enter a command prompt as Administrator and launch the command **Install-module Az**. To install for all users, we can use the command **Install-Module -Name Az -AllowClobber**. Then, if you already have NuGet Package provider installed it’s ok otherwise it will ask to download and install.

By default, the PowerShell Gallery isn't configured as a trusted repository for PowerShellGet. The first time you use the PowerShell Gallery, you will be prompted. You can download with other fantastic ransomwares or you can install with an offline package. Since we don’t have time and since I’ll destroy this VM after the lab I’ll continue downloading. It will take a while to complete.

After the module installation we have to disable the policy of restricted execution on Powershell with the command **Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope LocalMachine** then to secure our traffic we can enable TLv1.2 with this command **{** then we can use Azure Powershell command. We can connect to our azure account with the command **connect-azaccount.** It will ask credential through a web session with 2 factor authentication. Now we can create a resource for this demo. The command is quite simple **New-AzResourceGroup -Name Demo-PowerShell -location Switzerlandnorth** The we can obtain info with command **Get-AzResourceGroup** and finally we can delete the resource with **Remove-AzResourceGroup -Name RG-DevEnv -Confirm:$false -Force**. Opppsss… I’m jocking. This is another Resource group on which I put a lock. Remember to use locks on your critical resources to prevent this kind of issues. Let’s remove with the right name. **Remove-AzResourceGroup -Name Demo-PowerShell -Confirm:$false -Force.**

Demonstration Azure CLI

So, the last tool I want to show you is Azure CLI. It’s a command-line program to connect to Azure and execute administrative commands on Azure resources. It runs on Linux, macOS, and Windows, and allows administrators and developers to execute their commands through a terminal, command-line prompt, or script instead of a web browser. Let’s check some basic command. Then we can restart a VM with the command **az vm restart –resource-group RG-test -n test-vm01**. Azure CLI lets you control nearly every aspect of every Azure resource. You can work with resource groups, storage, VMs, Azure Active Directory (Azure AD), containers, machine learning, and so on. Commands in the CLI are structured in *groups* and *subgroups*. Each group represents a service provided by Azure, and the subgroups divide commands for these services into logical groupings. For example, the storage group contains subgroups including **account**, **blob**, **share**, and **queue**. One useful command is **az find** that can help a lot. For example, if you want to find commands that might help you manage a storage blob, you can use the find command: **az find blob**. If you already know the name of the command you want, the **--help** argument for that command will get you more detailed information on the command, and for a command group, a list of the available subcommands. For example, here's how you can get a list of the subgroups and commands for managing blob storage: **az storage blob –help.** We can use also the interactive version of AZ with the command az interactive. So let’s switch to cloudshell andcreate a simple web server with Azure CLI: First, we need to download the files for a sample website. I use a Git repository to get the files. Using git clone command, we simply copy the files locally in our Cloud Shell Home directory. git clone <https://github.com/Azure-Samples/html-docs-hello-world.git> I already download it and we can enter in the directory to create a web service with the content within. We can launch the command to create a web app: az webapp up --location switzerlandnorth --name testwebsvc5544 –-html Then after the creation we can check it on our browser. And that’s all. Remember to delete the resource group in order to save money if you don’t need these resources anymore: az group delete --name resourcegroupname

Compare Administrator Tools

We have seen several methods for managing Azure Resource Manager and of course there are pros and cons in all of them. This slide summarizes the main aspect of each tool.

## Configure Resources with ARM Template

If you needed to deploy one virtual machine, then you’d probably use the Azure Portal, which makes it easy. But what if you needed to deploy a hundred virtual machines? Or what if you needed to create one new VM every day, and you had to make sure they were all configured in the same way? That’s when you’d want to use an Azure Resource Manager template.

Review ARM Template Advantages

Resource Manager templates provide a common language for you and others to describe your deployments. Regardless of the tool or SDK that you use to deploy the template, the structure, format, and expressions inside the template remain the same. Templates enable you to deploy multiple resources in the correct order. For example, you wouldn’t want to deploy a virtual machine prior to creating an operating system disk or network interface. Resource manager maps out each resource and its dependent resources, and creates dependent resources first. Templates reduce manual, error-prone tasks. Manually creating and connecting resources can be time consuming, and it's easy to make mistakes. Resource Manager ensures that the deployment happens the same way every time. Templates express your requirements through code. Think of a template as a type of Infrastructure as Code that can be shared, tested, and versioned similar to any other piece of software. Also, because templates are code, you can create a "paper trail" that you can follow. The template code documents the deployment. Most users maintain their templates under some kind of revision control, such as GIT. When you change the template, its revision history also documents how the template (and your deployment) has evolved over time. Your template can contain parameters that are filled in when the template runs. A parameter can define a username or password, a domain name, and so on. Template parameters enable you to create multiple versions of your infrastructure, such as staging and production, while still using the exact same template. Templates simplify orchestration. You only need to deploy the template to deploy all of your resources. Normally this would take multiple operations.

Explore the JSON Template Schema

An ARM template contains all of the details that Azure needs to know to deploy a resource, such as a storage account or a virtual machine.

If you want to see one, you can go to an Azure resource you’ve deployed, such as this storage account, and click “Export template”. This is the ARM template for the account.

To make it easier to read, I’m going to download it, and open it in Visual Studio Code, which is Microsoft’s free code editor. You’ll notice that it was downloaded as a zip file. That’s because it needed to download two files: the ARM template and the parameters file. I’ll tell you more about the parameters file in a bit, but for now, let’s open the template file.

As you can see, there’s a lot in here, so I’m not going to go through every line, but I’ll give you the highlights. The first line has the **URL of the schema file** that describes the template language. Microsoft updates the language periodically, so if you want to use the features of the latest version of the language, then you’ll need to change this URL when a new one comes out.

The **content version** is the version of this template file. You can set it to whatever you want, but normally, you’d initially set it to 1.0 and then update the version number when you make changes to the template.

Next, there’s a parameters section. We’ll come back to it in a minute.

Then there’s a variables section, but it’s empty in this template. I’ll explain variables later.

The next section, called “resources”, is the heart of an ARM template. It specifies the details of all of the resources we want to deploy. Each resource has a type that says what it is.

The first part is called the resource provider. If you want to see a list of all of the resource providers available to you, go to your subscription in the Azure Portal and click on “Resource providers”. There’s “**Microsoft.Storage**”, which is the resource provider we saw in the template. If we click on it, we can see the list of its resource types in this dropdown menu. There’s “**storageAccounts**”. Don’t worry if it seems like it might be hard to find what you’re looking for in this massive list of resource providers and types. I’ll show you an easier way to fill in the resource type when we build an ARM template from scratch.

Let’s get back to the resource definition. First, it gives the API version for this resource type, which determines the properties that can be set for it. Each resource type has its own specific set of properties. All resources have a name, some of them require a location, some require a sku (that is, a feature set and pricing tier), and some require a kind, such as “**StorageV2”** in this case, which is short for “Standard general-purpose v2 storage account”.

Now we see the resource-specific properties. These are the configuration settings for the resource. Storage accounts have lots of settings, such as whether public network access is enabled or which access tier to use by default (“Hot” in this case). And that’s it.

That’s the full definition for this storage account. But what’s the rest of this stuff down below? Well, those are actually other resource types, but really, they’re part of this storage account. You’ll see that the type for this one starts with the storage account type and ends with “**blobServices**”. That’s because each storage account lets you store four types of data: blobs, files, queues, and tables. So, there’s one resource definition for each of those data types. There’s another clue that these resources are connected to the storage account resource.

This **dependsOn** element says that this resource can’t be deployed until this resource has been deployed. That makes sense because you can’t have blobs, files, queues, or tables outside of a storage account.

Explore the JSON Template Parameters

Notice that the part with the storage account name starts with “**parameters**”. Now we can see how the parameters section is used. This is the only parameter in this template. The definition starts with the name of the parameter, and then inside the curly braces, there’s a default value and a type. The type is “**String**”, which means that the value of this parameter must be text.

The default value of the parameter is set to “**stgdevrm077**”, which is the name of my storage account. So, when this template is used to deploy these resources, the value of this parameter will go here. The value is “**stgdevrm077**”, so that’s what it will replace this with. This might seem kind of pointless. Why wouldn’t it just put “**stgdevrm077**” here instead of turning it into a parameter? Well, if you wanted to use this template to create another storage account, you’d have to use a different name for the account. But you wouldn’t want to have to change every instance of that name in the template every time you deployed a new storage account using the template. By making the name a parameter, you can specify a new name on the command line, and you won’t have to modify the template.

If you only have one parameter, then setting it on the command line is a good solution, but if you have a lot of parameters, then it’s easier to set all of the values in a file and pass that parameters file on the command line. Does that sound familiar? That was the second file in the zip file we downloaded.

Let’s open it to see what it looks like. It’s pretty simple. It basically just lists the parameters that were defined in the template and lets you set a value for each one. For example, we could set the storage account’s name to “storagecostalot”. Make sure you put it in quotes, though, or you’ll get an error.

I’ll just mention one more thing before we move on from this template. Since storage account names have to be unique across all of Azure, If you tried to do a deployment using this template and you didn’t set the name parameter to something else, wouldn’t it try to use the default value and then get an error because that account name is already used? Well, no, actually. If you refer to an existing resource, then the ARM template will act on that resource, not on a new one. If you don’t change anything in the template, then nothing will happen to the resource. But if you do change something, such as a configuration detail, then it will try to make that change to the existing resource. I should warn you, that sometimes, exported templates are not completely correct and will generate errors if used. I was quite surprised when I discovered this. In fact if we try to launch this with **az deployment group create –resource-group demo-azcli –template-file template.json –parameters @parameters.json** it will report some error even if it creates the resource. We have to check the syntax in the template file.

All right, I showed you how to export the ARM template for a single resource (a storage account, in this case), but what if you wanted to export an ARM template for multiple resources, such as a storage account and an App Service plan? There’s an easy way to do that. If they’re in the same resource group, then you just need to go to that resource group in the portal and export the template from there. If you only want to include some of the resources, then select the ones you want, and select “Export template” from the menu instead.

While exported templates can be useful, especially for reviewing or updating existing resources, they’re not usually the best choice for deploying new resources. One alternative is to use quickstart templates. If you search for “template”, you can choose “Deploy a custom template”. It lists a few common templates, such as “Create a Linux virtual machine” and “Create a web app”. If you need something else, you can look for a quick-start template down here.

These templates are typically created by people outside of Microsoft, so you need to use them with caution. It can also be kind of hard to find what you’re looking for. Let’s say we want a template to create a storage account. If we type “storage”, it comes back with lots of templates, but they don’t look like what we need. We can narrow it down by typing, “storage-account”. There are still quite a few, but now we can see the one we want: “**storage-account-create”.** To see it, click “**Edit template**”. There are a couple of interesting things to see in this one. Under this parameter for storage account type, it lists the allowed values. These are the only values that you can set this parameter to.

Something else that’s new is this outputs section at the bottom. This tells Azure what to print out after the deployment is finished. In this case, it’ll print out the storage account name and the storage account ID. One more thing I want to point out is that the resources section is way shorter than the one in the template we looked at before. It doesn’t include anything for blobs, files, queues, or tables. It doesn’t even include any properties for the storage account itself. It only has the bare minimum details, such as the name and the location. If you don’t include all of those other details, then it just uses the defaults for all of them, so this will work just fine. And that’s it for this overview of ARM templates.

Consider Bicep templates

Azure Bicep is a tremendous gym workout to massively increase your biceps power and for this month, you can have it for just 10 dollars… Of course, I’m joking, just to test your attention. Someone had already taken the dumbbells but, unfortunately, it’s only another boring domain-specific language. It uses declarative syntax to deploy Azure resources. You can use Bicep instead of JSON to develop your Azure Resource Manager templates (ARM templates). The JSON syntax to create an ARM template can be verbose and require complicated expressions. Bicep syntax reduces that complexity and improves the development experience. Bicep is a transparent abstraction over ARM template JSON and doesn't lose any of the JSON template capabilities. So, let’s create a storage account using Azure Bicep. First, we have to install the extension in Visual Studio Code. I’ve already installed so I’ll move forward creating a simple file called **main.bicep**, with this extension Visual Code recognized the format and set the Language mode to Bicep.

Then, we can start typing **stor** and select **res-storage** from the list. If we try to modify for example the location with a custom value (not a parameter) it tells us that we cannot use it and it will fix it for us. Then we can launch the deployment through the terminal directly from Visual code with the command

**Set-AzDefault -resourceGroup Test**

**New-AzResourceGroupDeployment -TemplateFile main.bicep**