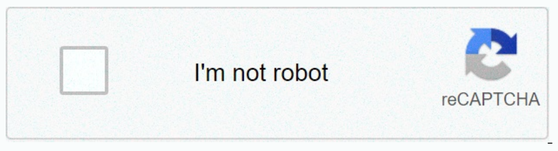

Continue

Azure arm templates infrastructure as code

The above keywords are often used in IT today, but what does all this mean? The idea of automation is not new. I think we've all used BAT files and more recently PowerShell scripts to provide a variety of levels of activity in the infrastructure. However, when automation met with virtualization, most programming languages that spoke natively to each other enabled IT to develop code and essentially orchestrate and deploy an entire infrastructure in minutes. This is ideal for easily scalable solutions that need to be deployed quickly. So if an employee says I need a server or storage with a big data share, just for me, this is now easily accessible through the power of Automation in Azure. The need to sign the investment, to talk to the SAN and the network team, etc., is all in the past. Written by Azure Virtual Machines (v2) on June 1, 2017, the new Azure VMs compared to the old ASM model are incredibly intelligent and so much more powerful. You now have a huge parallel deployment model where you can add custom scripts for PowerShell, DSC, Chef, Puppet, and so on. These VMs in Azure are constantly updated by Microsoft, so you get only the best computer to perform. Massive and parallel deployment of Virtual Machines 3 Fault Domains in Availability Sets Custom URLs for Custom Script VM Extensions for VMs Defining a Template How to Define Templates and Resource Groups is up to you and how you want to manage your solution. For example, you can deploy your three-tier application from a single template in a single resource group. You can create multiple ARM templates for specific infrastructure resources. Basically, you have a Virtual Network Resource Template and a VM Template... These can be linked together using a nested format or individual templates so that you can use them over and over again. What is JSON? JSON stands for JavaScript Object Notation, which is an open standard format. While the name suggests that it is derived from JavaScript, it can actually be read and understood from most programming languages that exist today. The templates you see are whitespace nonsensical, this means it doesn't matter how many new lines of code you insert, how many spaces there are in the syntax, it won't affect the actual functionality of this template. You can add them as much as you want to make it easy for you to read as desired. When you start creating templates, you'll see that it looks massive, but if you continue to investigate, only many and many new lines of code with a single character on a particular line. Tooling There are several ways to create your JSON templates using Notepad++, Visual Studio code, and then deploying ARM templates with Visual Studio (any edition) or PowerShell, or even now through the portal. With the power of Azure Resource Manager templates, you can create and deploy resources to Azure in minutes. With this template approach template approach can repeatedly deploy your solution throughout its lifecycle and trust that your resources will be deployed in a consistent state. In a sample scenario, you enter a situation where you now create multiple resources in Azure and you need two VMs, three VMs, and so on, loads that are balanced with static IPLs, server 2016, and so on. Realistically, how long will it take you to click through the portal or even create it locally. This is where creating ARM templates will be perfect for this type of solution, once you get the slope of coding these, it will be second knowledge. Getting Started With Some Basic ARM Template Deployments, you can switch to Github, a public repository for tons of ARM templates that are available to you for instant downloading and creating self-ads. You can even deploy these templates directly from the website that redirects you to the Azure portal. Once you've learned the basics of this template structure, you can start creating and deploying your own custom templates, create different cloud environments, develop tests, and so on. Keep Up To Date – Join The Mailing List Subscribe PowerON, Stanley Harrison House, York, YO23 1DE 0800 3029280 info@poweronplatforms.com TRy Us for Yourself The above keywords are commonly used in IT today, but what does that mean? The idea of automation is not new. I think we've all used BAT files and more recently PowerShell scripts to provide a variety of levels of activity in the infrastructure. However, when automation met with virtualization, most programming languages that spoke natively to each other enabled IT to develop code and essentially orchestrate and deploy an entire infrastructure in minutes. This is ideal for easily scalable solutions that need to be deployed quickly. So if an employee says I need a server or storage with a big data share, just for me, this is now easily accessible through the power of Automation in Azure. The need to sign the investment, to talk to the SAN and the network team, etc., is all in the past. Written by Azure Virtual Machines (v2) on June 1, 2017, the new Azure VMs compared to the old ASM model are incredibly intelligent and so much more powerful. You now have a huge parallel deployment model where you can add custom scripts for PowerShell, DSC, Chef, Puppet, and so on. These VMs in Azure are constantly updated by Microsoft, so you get only the best computer to perform. Massive and parallel deployment of Virtual Machines 3 Fault in Availability Sets Custom URLs for Custom Script VM Extensions for VMs Defining a Template How to Define Templates and Resource Groups is up to you and how you want to manage your solution. For example, you can deploy your three-tier application from a single template in a single resource group. You can create multiple ARM templates for specific infrastructure resources. Basically, you have a virtual network resource network resource and a VM template... These can be linked together using a nested format or individual templates so that you can use them over and over again. What is JSON? JSON stands for JavaScript Object Notation, which is an open standard format. While the name suggests that it is derived from JavaScript, it can actually be read and understood from most programming languages that exist today. The templates you see are whitespace nonsensical, this means it doesn't matter how many new lines of code you insert, how many spaces there are in the syntax, it won't affect the actual functionality of this template. You can add them as much as you want to make it easy for you to read as desired. When you start creating templates, you'll see that it looks massive, but if you continue to investigate, it's just many and many new lines of code with a single character on a particular line. Tooling There are several ways to create your JSON templates using Notepad++, Visual Studio code, and then deploying ARM templates with Visual Studio (any edition) or PowerShell, or even now through the portal. With the power of Azure Resource Manager templates, you can create and deploy resources to Azure in minutes. Using this template approach, you can repeatedly deploy your solution throughout the lifecycle and trust that your resources will be deployed in a consistent state. In a sample scenario, you enter a situation where you now create multiple resources in Azure and you need two VMs, three VMs, and so on, loads that are balanced with static IPLs, server 2016, and so on. Realistically, how long will it take you to click through the portal or even create it locally. This is where creating ARM templates will be perfect for this type of solution, once you get the slope of coding these, it will be second knowledge. Getting Started With Some Basic ARM Template Deployments, you can switch to Github, a public repository for tons of ARM templates that are available to you for instant downloading and creating self-ads. You can even deploy these templates directly from the website that redirects you to the Azure portal. Once you've got the basics of this template structure, you can start creating and deploying your own custom templates, create different cloud environments, develop tests, and so on. Keep Up To Date – Join The Mailing List Subscribe to PowerON, Stanley Harrison House, York, YO23 1DE 0800 3029280 info@poweronplatforms.com TRy Us for Yourself Azure is managed with an API: Originally it was managed with Azure Management API or ASM that controls deployments of classic. This has been replaced by the Azure Resource Manager or arm API. The resources managed by the ARM API are objects in Azure, such as .B NICs, virtual machines, hosted databases. The main advantages of the ARM API are that you can deploy multiple resources together in one unit, and that the deployments are idempotent, User declares the resource type, which name to use, and what properties it should have. The ARM API then either creates a new object that matches these details, or modifies an existing object that has the same name and type to have the same properties. ARM templates are a way to declare the desired objects, types, names, and properties in a JSON file that, like any other code file, can be checked into source control and managed. ARM templates are what really gives us the ability to introduce Azure infrastructure as code. What can ARM templates do An ARM template can contain either the contents of an entire resource group or one or more resources from a resource group. When a template is deployed, you can use either full or incremental mode. The mode completely deletes all objects that are not displayed in the template and the resource group for which you are deploying. In this scenario, you get the ability to know that you are in exactly the same state each time you deploy. Incremental deployment uses the template to add additional resources to an existing resource group. The advantage of this is that you don't lose any infrastructure that is missing from the template, but the downside is that you have to clean up all the old resources in a different way. The ideal deployment is complete, but it means that you must have a good automated deployment pipeline with at least one test environment where you can verify that the template isn't pulling your heart out of your beautiful production environment. What's not, the ARM API provides resources in Azure, but does not provide code for those resources. For example, you can use ARM to deploy a virtual machine that already has SQL Server installed, but you cannot use ARM to deploy a database from an SSDT DacPac. To save time designing solutions, it's important to understand that the ARM API is easy to use for resources, and we need to use some other technologies, such as DSC or PowerShell, to manage deployments in the infrastructure once it's deployed. How are they tested functional tests The actual ARM templates are JSON, which is essentially a block of text that should be read by a machine instead of simply reading for a human. Formatting helps, but it's still a single block of text that doesn't have an actual test framework, so testing really matters to perform a deployment and see what it creates. When used as part of a broader test suite, the test process should be: deploy in a test environment, A development/test subscription in Azure Deploy code and application tests Test Tests Run Report Results If all tests return success, the template is by definition valid. Basic tests Apart from the functional tests, it is possible to verify that a template can actually be deployed to the resource group that you want to deploy. If you are using the Test-AzureRmResourceGroupDeployment PowerShell function, use takes your template, analyzes it, and verifies that it's syntactically correct, and verify that you meet requirements, such as.B not reaching offer limits before you deploy. This is really useful because the ARM template is not compiled. This prevents you from using a build step in the build process that you could otherwise use to ensure that the ARM template can even be deployed. In addition, with the full type of deployment, it wouldn't be ideal if you delete different resources and then don't create all the new resources you wanted because you've reached your quota limits in Azure. ARM template execution There are two important concepts to understand when using ARM templates. The first is that the ARM REST API is the part that actually does the heavy lifting. It is this that provides the idempotency of the entire process. The REST API is well documented, regularly updated, and available either on the Microsoft Docs website or on github if you felt like contributing: Azure REST API Reference— Azure/azure-rest-api-specs— There's a set of REST APIs called Resource Management where you send an ARM template. The REST API takes the template and: Analyzes the JSON fills in all parameters passed in Executes, calls all ARM template functions, calls the REST API of all resource types that need to be created to create them. If we look at this simple example of an ARM template that provides a single storage account, the process that the REST API goes through is to read the storageAccountType parameter from the parameters that are also passed at the same time as the deployment. Because the storageAccountType parameter also specifies what the allowedValues are, the parameter is checked against the list of allowedValues. If the passed parameter has a typo or other error, the deployment is canceled at this point. If you omit the allowedValues, this check will not take place. The REST API then creates the diagStorageAccountName variable, and the value of the variable is the result of the ARM template functions that concatenate the string 'diags' with the output of uniqueString(resourcegroup(id), which creates a unique string for this resource group. The uniqueString function passes the ID of the resource group to be used as the base string for uniqueString, so that it is unique per resource group. If you need a second string to be unique, you must pass another string to hash or omit the base string. The uniqueString function is best used with a so that subsequent deployments do not create new objects every time. The REST API then uses the resource section to call the resource-specific APIs. In this example, only one resource of type Microsoft.Storage/storageAccounts can be created. The type in the ARM template is directly mapped to another of the REST APIs types that The apiVersion tells the Resource Manager which version of the API to call, what properties you can set, and also the behavior of the API, and can change between versions, so it's important that you get the correct API version. The API version is specified for each resource, so you can deploy two things of the same type (e..B. storage account), but you can deploy them with different versions of the REST API. The location is determined from the resourceGroup() function, which returns a list of properties, one of which is the location. The value set for sku is resolved from the storageAccountName parameter. Any value in the JSON template that is surrounded by [and] is evaluated as code. The full list of template functions that can be used is documented The Resource Manager then creates the JSON, which is sent to the storageAccount REST API, which looks like this: This JSON is then sent as a PUT request to the following URI: /subscriptions/-subscriptionId/resource The rest API storageAccounts is then responsible for verifying that a storage account with the name we have. If not, a new one is created and it ensures that it has the same properties as the properties we pass. If it already exists, the Storage REST API only ensures that the properties are set correctly. If the storageAccount PUT request occurs when there is something it cannot do, such as.B an account with the same name as an existing account in another resource group or subscription (perhaps even creating another person or organization), it will fail. There are also some changes that it finds impossible, such as.B the compute API that modifies the base image for which a virtual machine was created. Sometimes you need to delete resources and start again for some changes. When the PUT request is complete, the API returns a JSON document that contains the definition of the object that was just created, so there are some things that are not known at build time. This is useful for cases where you create something like a storage account that creates the access keys when it is created. You can reference the key in your ARM template and use it in other objects that require a storage account name and key. The properties are returned regardless of whether the object already existed or not. What else can we do in ARM templates? Defining dependencies In some cases, you need to provide resources in a specific order: to create a For example, you must provide a network card, and you might want to provide a public IP address for a NETWORK card. If you are trying to create the NIC and point to a public IP address that does not exist, then the deployment fails. The answer is that dependencies and each resource can have a dependsOn section that lists all dependencies that must be created before the resource can be created. Dependencies are really useful, but should be used with caution because they limit the number of concurrent activities of the resource manager. The more dependencies you have and the longer the dependency chains are, the longer the deployments take. It is also possible to define dependencies by nesting resources, and in this case, child dependencies are created after the parent resource is created. Finally, there is a third way to create a dependency to use the Reference template function. This is done to get the properties of another resource, the same as in the output for the PUT request above. This allows you to use, for example.B an account key from a storage account that may not be known at creation time. When you use a reference to access another resource in the template, an implicit dependency is created for you. The dependencies are documented by Microsoft: copy sections When we deploy resources, we might just want one, but it is more likely that a number of resources similar to virtual machines will be created. To help with this, there is the Copy section, which specifies how many copies of a resource we need. This is great because it means that if we want 100 virtual machines, we don't have to define 100 resources in the ARM template. It works in such a way that you add a copy block to your resource, e.B.: That is, we should have created 3 storage accounts. There are two things to keep in mind here: first, because we say that we have to make 3 copies of it unique among the three objects. We do this so that instead of using just the account name variable, we attach the copyIndex, which is the ID of the copy operation. We do this because if we pass 1 into the copyIndex function, it starts at 1 instead of 0, so we get accountName1, accountName2 and accountName3. The second thing is that the copy block has a name. This is really useful because we can use it in a dependsOn block and the dependent resources are not created until all three storage accounts have been created. For example.B, is useful when creating virtual machines that take longer to deploy the first computer than the last. If you be able to refer by copy name, you would have to add a dependsOn to all resources individually, which quickly becomes chaotic. Conditionals Conditionals are a recent addition to ARM templates. You specify whether to deploy a specific resource. This can be useful if you are dealing with different environments in which you are developing a virtual machine without load balancers, but in production you might want five machines with a load balancer. To use the same template, you can pass the number as a parameter, but to choose whether you want to deploy the load balancer, you can return false in the Condition property. In this example, we have a condition property that checks whether a parameter is set to yes or not. Conditionals should definitely be used with caution, and I would suggest at least one pre-production environment in which the entire template is deployed. To be fair, they meet a need that was missing and which previously involved passing whole blobs from JSON as parameters. How do you start? There are a few approaches, the easiest of which is to use the quickstart templates created by Microsoft that give examples of how most resources are used. The quick start templates are available here: The second approach is to deploy some resources: In the azure portal, there is a button called Automation Script that generates the ARM template to deploy the entire resource group. This always creates a script for the entire resource group, even if you click the automation script for a specific resource. Not every type of resource can be generated today, although it seems that Microsoft is increasing the amount of resources for which they generate ARM templates. You may only have to wait a few weeks. In my opinion, the automation script does not generate the ideal templates. The names are odd and although they set names behind parameters, they use the name of the resource in the parameter names, making it difficult to read and edit the scripts. I would say that they are best used to show how a resource is configured, and simply take the parts from the script that you need to reuse and manually create the other parts of the template or use a company-shared template to create the ARM template. Automation also generates the scripts that can be used by various clients such as PowerShell, Bash, and C. This gives you the code you need to send your ARM template to the ARM REST API for processing. These, too, are quite detailed and you don't need everything. If you are using .B PowerShell, you must create the resource group yourself if it does not exist, and then call New-AzureRmResourceGroupDeployment from the AzureRM module. Documentation The ARM template functions themselves, as opposed to the actual resources, are documented: The and their properties are documented: This documents only the latest version of an API. For the exact details of what can and cannot be used in an ARM template for a resource, the JSON schemas are available: Although they are not so simple Follow the documentation page and provide the exact details of what can and cannot be deployed. Sometimes the REST APIs are updated before the documentation, if you use something that is rioting, you may be lucky if you look at the documentation for the REST API, see what properties you can set, and then simply insert them into the ARM template. Template.