

|  |
| --- |
| **Azure PROGRAM** |
| Azure ARM examples handbook for EPAMers. |

*PREFACE*

This guide give you examples of how to use Azure PowerShell with Resource Manager templates to deploy your resources to Azure. This tutorial will list some best practices that may be useful when working on your current homework and on future work on projects. By following them you can improve the quality of written templates....

Contents

[1 Deployment and infrastructure schemas, resources 4](#_Toc1946895)

[1.1 Parameters template 6](#_Toc1946896)

[1.2 Main template (Base.json) 7](#_Toc1946897)

[1.2.1 PARAMETERS SECTION (Base.json) 8](#_Toc1946898)

[1.2.2 Description & Comments 8](#_Toc1946899)

[1.2.3 artifacts location Sas Token 9](#_Toc1946900)

[1.2.4 environment parameter 9](#_Toc1946901)

[1.2.5 KeyVault, VmAdm, SqlAdm Names 10](#_Toc1946902)

[1.2.6 VARIABLES SECTION (Base.json) 10](#_Toc1946903)

[1.2.7 RESOURCES SECTION (Base.json) 11](#_Toc1946904)

[1.2.8 api Version 11](#_Toc1946905)

[1.2.9 depends On & reference 12](#_Toc1946906)

[1.2.10 Template link 13](#_Toc1946907)

[1.2.11 contentVersion 15](#_Toc1946908)

[1.2.12 OUTPUTS SECTION (Base.json) 16](#_Toc1946909)

[2 Resource templates 16](#_Toc1946910)

[2.1 vNetNsgs.json 16](#_Toc1946911)

[2.2 vNet.json 18](#_Toc1946912)

[2.2.1 Working with arrays 19](#_Toc1946913)

[2.3 availabilitySets.json 21](#_Toc1946914)

[2.3.1 Tags 21](#_Toc1946915)

[2.3.2 Sku 21](#_Toc1946916)

[2.3.3 Outputs 22](#_Toc1946917)

[2.4 vm.json 23](#_Toc1946918)

[2.5 webApp.json 24](#_Toc1946919)

[2.6 SQL.json 26](#_Toc1946922)

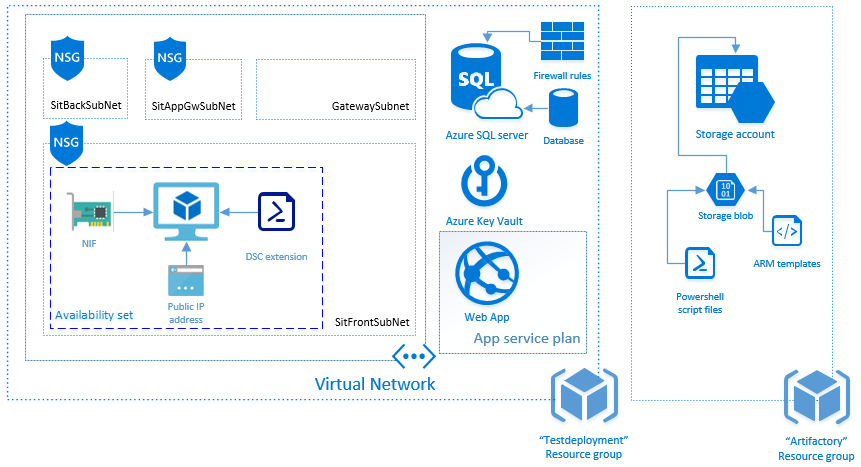
[3 DSC 28](#_Toc1946923)

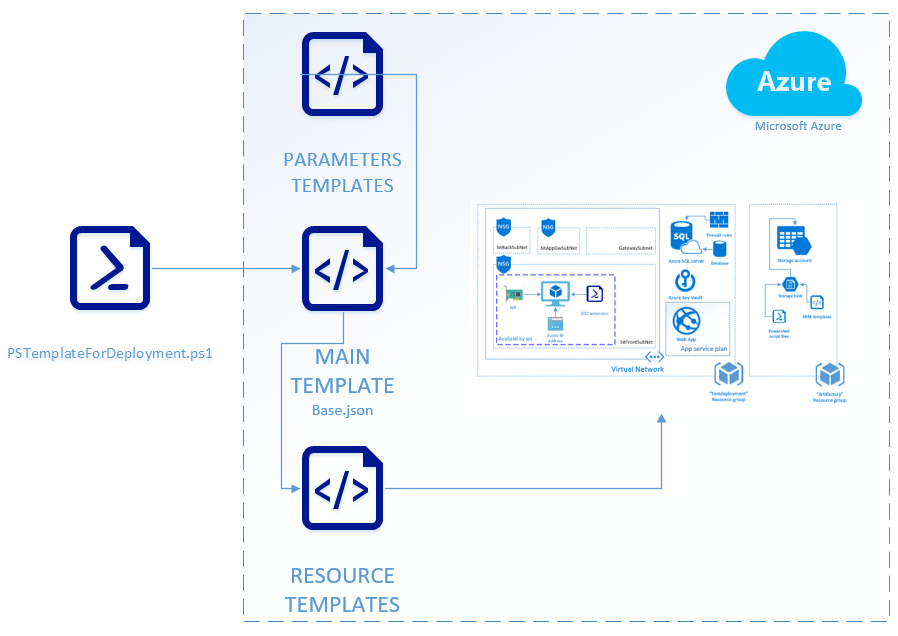
[3.1 autoupgrademinorversion option 28](#_Toc1946924)

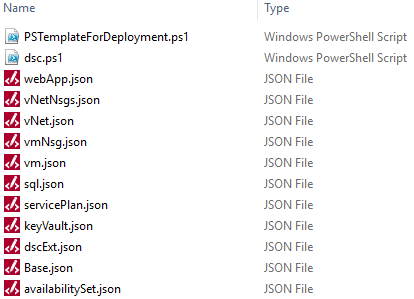
[4 Powershell 32](#_Toc1946925)

# Deployment and infrastructure schemas, resources

This scheme is deployed by using the following resources (PowerShell scripts and ARM templates).







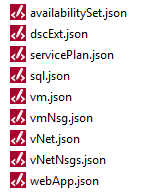
*Parameters template*. A number of parameters will be transferred to this template from deployment script for subsequent use in the deployment process:



*Main template* - Base.json:



*Resource templates*. ARM Templates with deployment configuration for each resource described in Base.json template:



* *vNetNsgs.json* – this template contains a configuration for three network security groups (AppGwSubnetNsgName, FrontSubnetNsgName, BackSubnetNsgName) with the different set of parameters of inbound security rules;
* *vNet.json* – this template contains configuration of three virtual network
* *availabilitySet.json* – it is contains availability set with configured set of parameters: number of fault domains, number of update domains and disabled option “IsManaged”
* *vmNsg.json* – this template contains a configuration for network security group of virtual machine
* *vm.json* – here we has a resources for a configuration of Public IP address and Network Interface for VM, configuration for IaaSAntimalware extension etc.
* *dscExt.json* – it contains a configuration for DSC extension
* *sql.json* – this template has configuration for database and firewall rules.
* *servicePlan.json* – it contains a configuration of App service plan: SKU parameters, max count of workers. Also here we have configuration for auto scale settings: rules with triggers.
* *webApp.json* – this template contains a configuration of site and its logs retention policy

dscExt.ps1 (DSC configuration which will be applied):

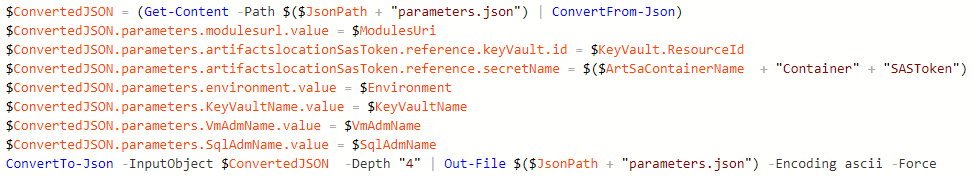


PSTemplateForDeployment.ps1 (a script for deployment):



## Parameters template

In our solution, we use a separate file with parameters to build a more flexible deployment. Using PowerShell script, we generate some general values and then pass these values to the template parameters for further deployment. Here's what it looks like in the code:



And here’s what we get in the template:



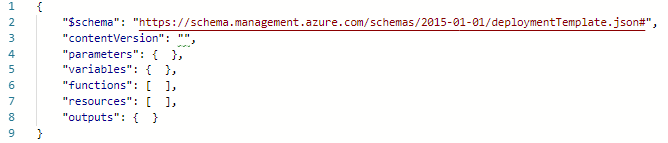
Now this template is ready to use. In the POWERSHELL section (*page 29*) we will see in details how to deploy it and what other parameters are transmitted.

## Main template (Base.json)

This ARM template gives us insight how many resources will be deployed and in what order:



In a simplest structure, ARM template has the following elements:



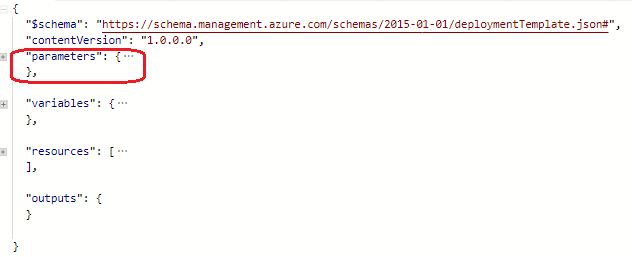
For our Base.json template, in addition to the required $schema we used the following elements:

* parameters,
* variables,
* resources,
* outputs.

Let’s go through each of them and find some useful examples.

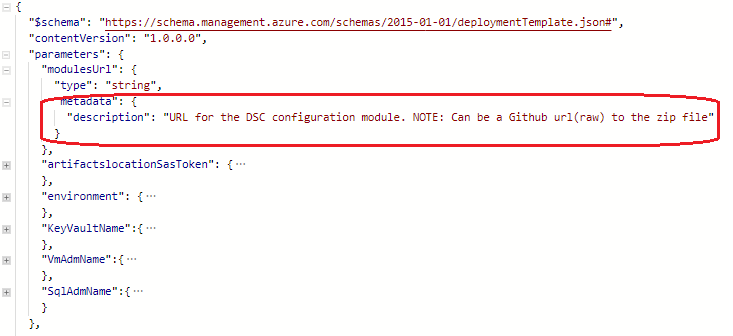
### PARAMETERS SECTION (Base.json)

In the parameters section of the template, you specify which values you can input when deploying the resources.



### Description & Comments

We have a few options for adding comments and metadata to our template. We can add a metadata object almost anywhere in our template. Resource Manager ignores the object, but our JSON editor may warn you that the property isn't valid. In the object, define the properties we need. In our case we used only metadata object with a description property in parameters section:

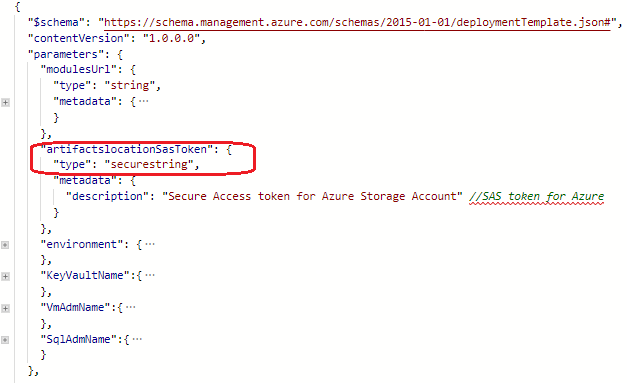


For inline comments, you can use "//" syntax:



### artifacts location Sas Token

Also in this section we used “artifactslocationSasToken” to call a previously generated Shared Access Signature (SAS) token when working with the repository. This SAS token can be generated for an optional amount of time. We used the secure object type – “securestring” for that because all template parameters with secure string or secure object types can't be read after resource deployment:



### environment parameter

These parameter value enable you to customize the deployment by providing values that are tailored for a particular environment (such as Dev, Sit, Uat and Prd). The following example shows a simple parameter definition. It defines the name of the parameter, and specifies that it takes a string value. The parameter only accepts values that make sense for its intended use. We set these values as an array:

****

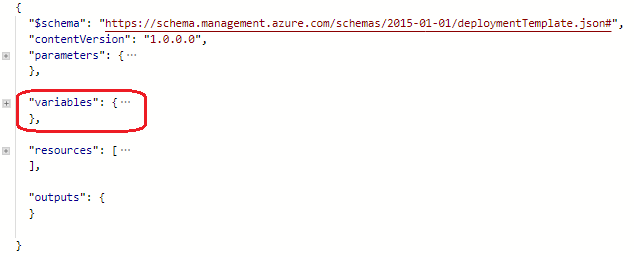
As a result, we will protect ourselves when passing invalid environment values into deploy script.

### KeyVault, VmAdm, SqlAdm Names

All these three parameters values are passed to main template via parameters.json template:



### VARIABLES SECTION (Base.json)

****

In this section, we have applied “replace” function in conjunction with another several template functions (first,toUpper) when constructing a value, so that any transferred environment variable (which will later compose the names of different services) will always starts with a capital letter. This was made for easier reading of deployed service names (еще для чего?).



And now let's divide this line into those used in it.

The first parameter of the “replace” function we declare the value to be changed:



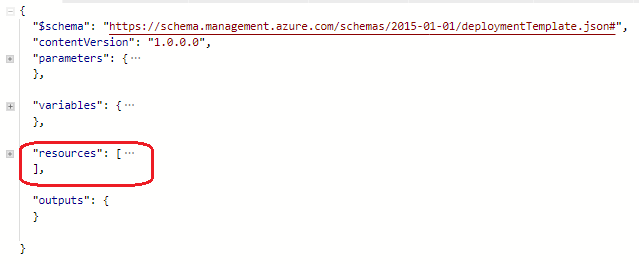
Then after the delimiting comma we declare that part of the string to be replaced :



Finally, we replace this value with the same but already capitalized one:



### RESOURCES SECTION (Base.json)

****

### api Version

During the modifying this resource section you need to make sure that the schema of your template is compatible with the API version you are using. It may happen that a new API-version is released and the old one is no longer working.You can use Get-AzureRmResourceProvider cmdlet to identify the list of available API-versions of desired resources.



### depends On & reference

In the resources section, we defined the resources that are deployed or updated. In the Base.json template we can define the order for deploying resources. For example, a SQL server must exist before attempting to deploy a SQL database. You define this relationship by marking one resource as dependent on the other resource. You define a dependency with the “dependsOn” element, or by using “the reference function”. Resource Manager evaluates the dependencies between resources, and deploys them in their dependent order. When resources aren't dependent on each other, Resource Manager deploys them in parallel. You only need to define dependencies for resources that are deployed in the same template.

Using “dependsOn”

The following part of template Base.json shows a virtual network that depends on Network security groups:



Using the “reference function”

As a second option you can use a reference function in the parameter section of necessary resource. In this case you need to configure appropriate parameters in Base.json and add “outputs” section in resource template of dependent resource. Here is how it should be configured in both places.

In resource template (vNetNsgs.json):



In Base.json:



Therefore, until Network security groups are deployed and outputs with their resource ID’s are created you can’t start rolling out a virtual network. You need to know that by using “dependsOn”, you potentially impact deployment time because Resource Manager doesn't deploy in parallel two resources that have a dependency. Also please be aware that you can use the “reference function” only in “output” and “parameters” sections of ARM templates.

You can also read more about defining of recourse dependencies here:

[**https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-define-dependencies**](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-define-dependencies)

### Template link

When linking to a template, the Resource Manager service must be able to access it. You can't specify a local file or a file that is only available on your local network. You can only provide a URI value that includes either http or https. One option is to place your linked template in a storage account, and use the URI for that item. In order to get link to the downloaded template we use “concat” function and several incoming parameters:



Let's divide this string into parts.

The first elements **of concatenated** string is “modulesUrl”:



it came to us from parameters.json template:



The next one is hardcoded slash symbol “/”:



Then we have environment variables converted to lower case (toLower function). It is necessary because we are creating a link, therefore, all letter must be in a lower case:



Next comes hardcoded name of necessary resource template. In our case it is vNetNsgs.json:



And finally we put SAS token, which as well as “modulesUrl” was transmitted to us with a file of parameters:



Parameters.json:



As a result, we get a link to the account for the necessary resource (vNetNsgs.json).

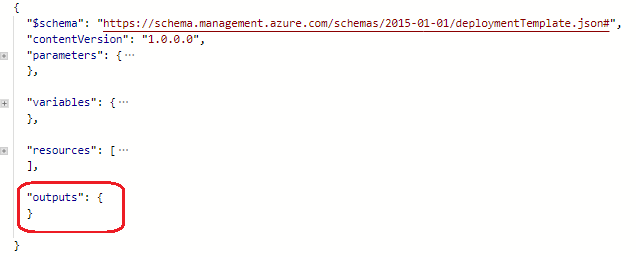
### contentVersion

Version of the template (such as 1.0.0.0). You can provide any value for this element. Use this value to document significant changes in your template. When deploying resources using the template, this value can be used to make sure that the right template is being used.



### OUTPUTS SECTION (Base.json)

In the “Outputs” section, you specify values that are returned from deployment. For example, you could return the URI to access a deployed resource. In our main template we aren’t using output section.



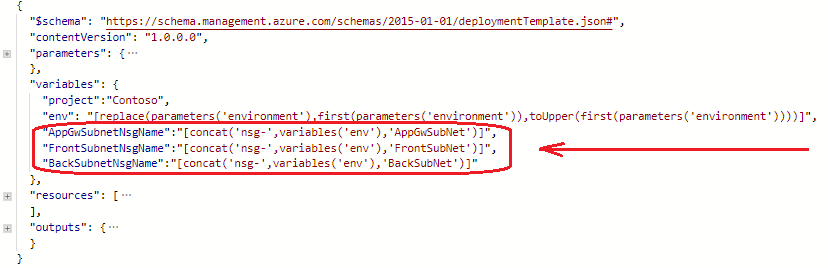
# Resource templates

## vNetNsgs.json

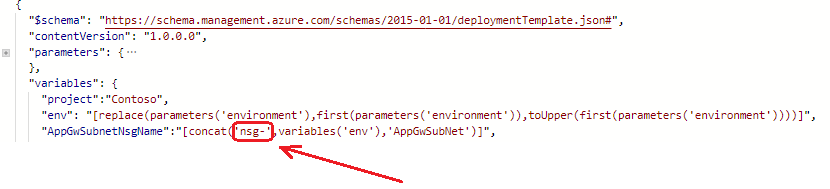
To link to another template, we added a deployments resource to our main template In this section, we will not describe fully all resource templates, so we will only highlight the most useful points. The first deployed recourse that linked with base.json is Network Security Group (vNetNsgs.json). To link this external template we used “templateLink” property:



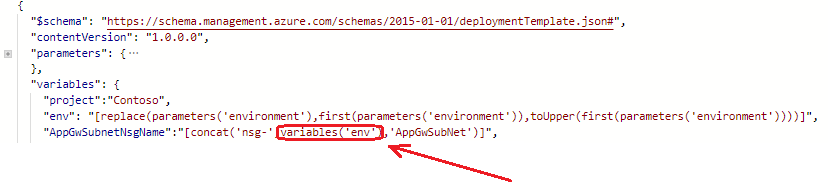
In “variables” section of this resource template vNetNsgs.json we specified NSG names for all three deployed network security groups using by “concat” string function:



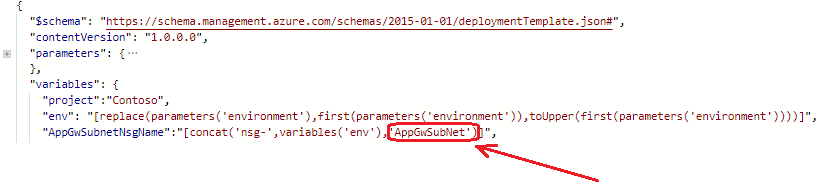
As in the other examples described earlier, we combine several values into one. We took hardcoded ‘nsg-‘ :



Than we added transmitted environment variables value:



And finally we put hardcoded ‘AppGwSubNet’ value to the end of the concatenated string:



The output from the preceding example with the default values is “nsg-SitAppGwSubNet”.

In the resources section of vNetNsgs.json we defined the resources that are deployed. Here we used the previously generated name of NSG:



When deploying a template, you must provide a location for each resource. Different resource types are supported in different locations. To get the supported locations for a resource type, please follow the link below:

<https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-supported-services> .

We used next parameter to specify a location for resources, and set the default value to:



If you need to hardcode the location in your template, provide the name of one of the supported regions.

Example:

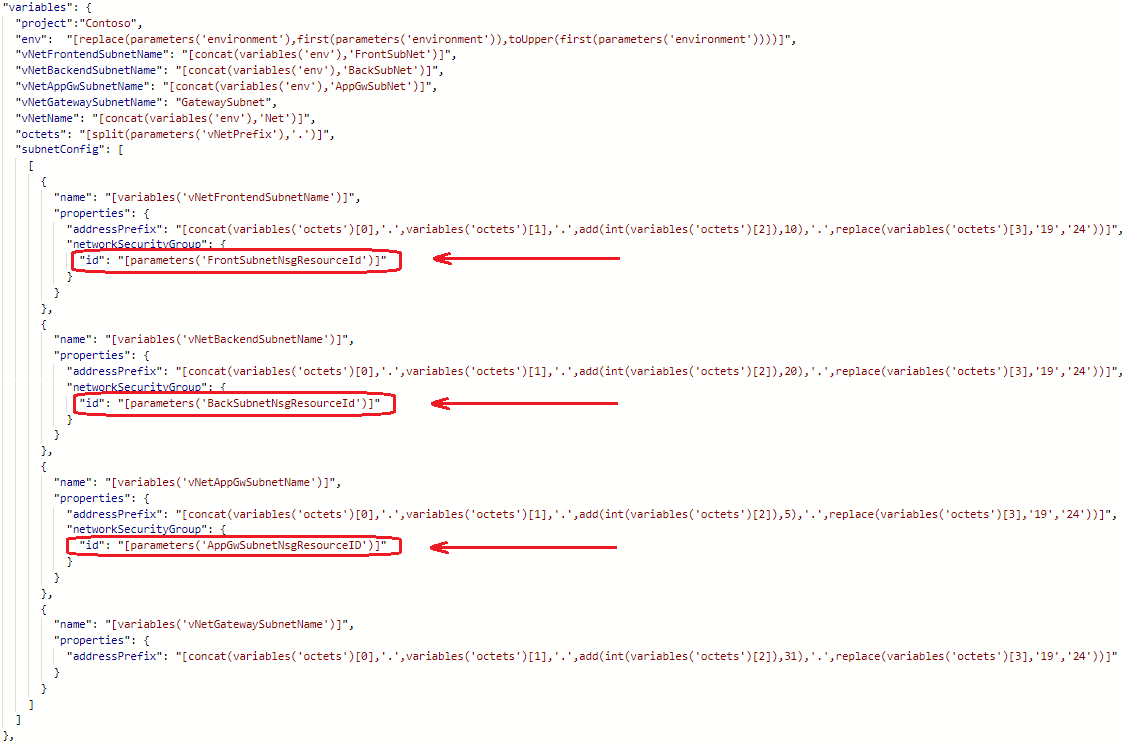


## vNet.json

To configure all virtual networks, we need a reference to resources (NSG) which aren’t described in this linked template. In the “outputs” section, you specify values that are returned from deployment. Therefore we used output object from vNetNsgs.json:

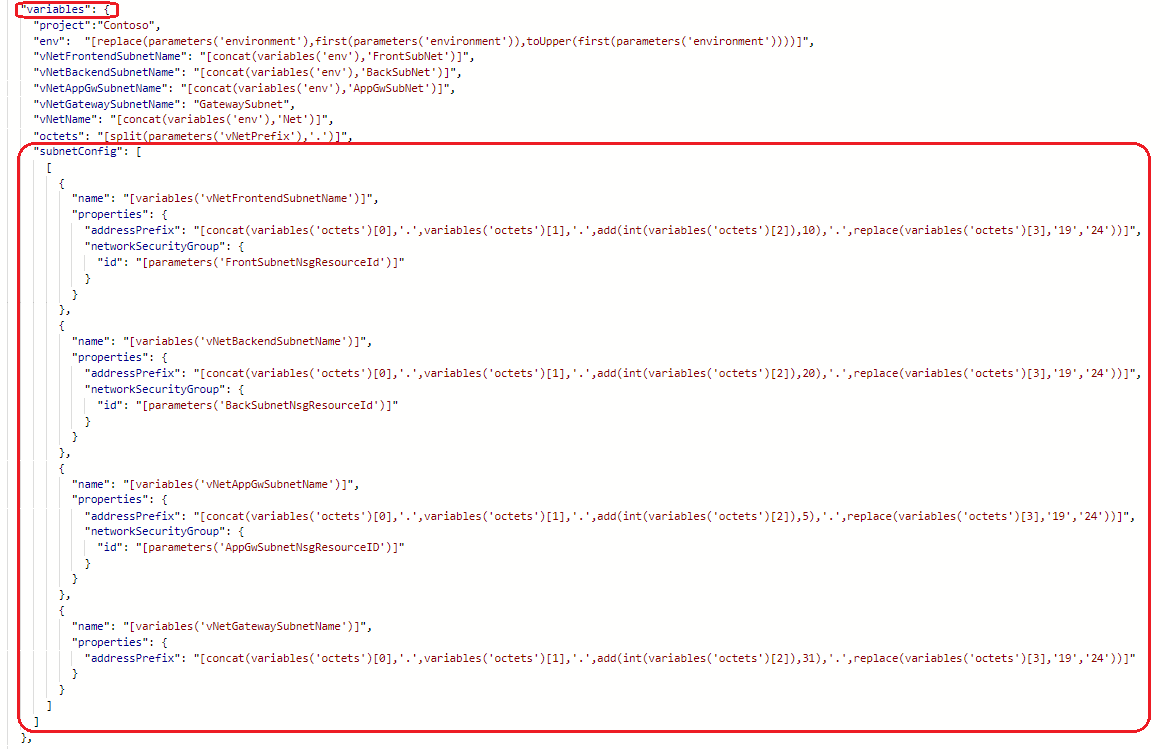


And in vNet.json in “variables” section:



### Working with arrays

To specify address prefix in the variable section of this resource template, we created an array in which there is one created object called "subnetconfig", this array contains a child array of objects with parameters of our subnets:



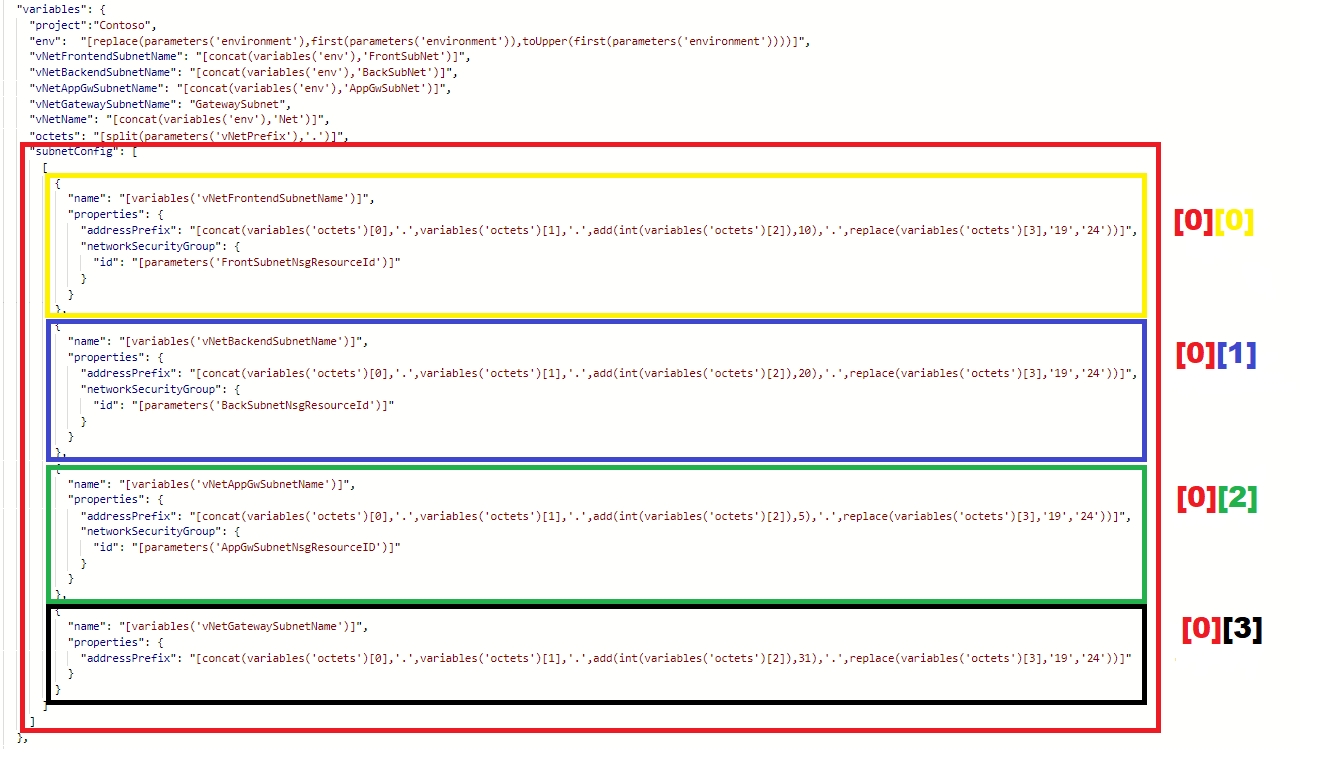
In order to extract the necessary data from these arrays and then use as an output for another templates we applied the following approach:



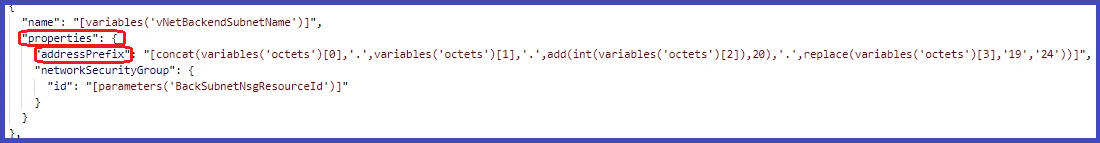
Let’s take a look closer to one of the value:



To obtain this value we go to variable “subnetConfig” and iterate over the elements of the array of this variable. We have given below a picture for a visual understanding of the structure of our array:



After we chose necessary object we dive into the “property” and “addressPrefix”:



## availabilitySets.json

An availability set is a group of virtual machines that are deployed across fault domains and update domains. Availability sets make sure that your application is not affected by single points of failure, like the network switch or the power unit of a rack of servers.

### Tags

Using this template as an example, we would like to show you how we used “tags” element. We apply tags to our Azure resources giving metadata to logically organize them into a taxonomy. Each tag consists of a name and a value pair. In example below, as well as “Environment” and “ServiceModel” we applied the name "Project" and the value "Contoso" to the resource availability set in Contoso project:



After we apply tags, we can retrieve all the resources in our subscription with that tag name and value. Tags enable us to retrieve related resources from different resource groups. This approach is helpful when you need to organize resources for billing or management.

Please find some limitation during applying tags on this link:

<https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-templates-resources#tags>

### Sku

To create managed availability sets with VMs using managed disks, we added the “sku” object to the availability set resource and applied the function “if” added some condition for applying the value:



In this case if the value of parameters ('IsManaged') is equal to “Yes”, then $true will be returned and the variable ('skuName') value will be substituted.

On the other hand, If the parameters ('IsManaged') contain something different from “Yes”, then $False will return and json ('null') will be applied (i.e. zero value). So, depending on what value was transferred from base.json to the IsManaged parameter of availabilitySets json, in this case the corresponding condition will be met. This is done for more flexible use of the passed parameters.

### Outputs

In this resource template, we use the “outputs” section in order to pass the name of availability set as a parameter into parameters section of Base.json for further configuration of the virtual machine.

Here is how it is configured in the output section of availabilitySets.json:



Than we make a reference to above output in parameters section of “VM” resource in Base.json:



## vm.json

In Base.json part of deploying virtual machine we specified an image configuration from which VM will be deployed and then passed it as object to parameter “vmImages” of vm.json.

Base.json:



vm.json:



The same approach we used to implemented for the passing specified “sku” parameters for an App Service Plan into linked servicePlan.json:



As you can see we also have configured default settings for “sku” parameter in case there is no other value from the main template.

## webApp.json

A system assigned managed identity enables Azure resources to authenticate to cloud services (e.g. Azure Key Vault) without storing credentials in code. Once enabled, all necessary permissions can be granted via Azure role-based-access-control. The lifecycle of this type of managed identity is tied to the lifecycle of this resource.

## To add a system-assigned identity option to our deployment we need to configure appropriate section in resource template.

## 

You can learn more about Managed identities here: <http://go.microsoft.com/fwlink/?LinkId=854449>

Also in this template we enable Failed Request Tracing (for gathering diagnostic information on failed requests) and Detailed Error Messages (for gathering detailed error messages from your web app) options on our Azure App Service. For this we applied boolean type when converting the transmitted value "true":



## SQL.json

In this template we want to show you how you can configure firewall rules in resource section of Sql.json linked template. To create a “firewallRules” resource, you need to add the following JSON to the resources section:



# DSC

Also in our deployment process, we use the DSC script for the process of configuring IIS on our virtual machine. First of all we created a configuration with necessary resources (dsc.ps1):



Then we prepared ARM templates: added description of this resource in base.json and created a separate resource template - dscExt.json. In base.json file we described some parameters, added dependency and link to resource template location. In dscExt.json we added link to desired configuration - dsc.ps1.

## autoupgrademinorversion option

During the configuration of the DSC resource in the template, we determine whether indicates the extension version should be automatically updated to a newer minor version. Accepts the values true or false:

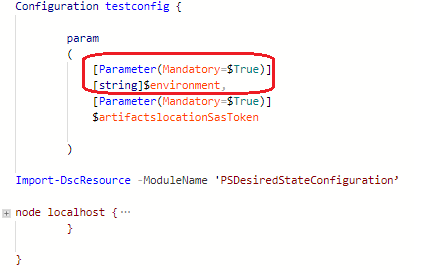


Also in this template we have specified the url with DSC script location. As usual we used “concat” function for the construction of the link to the storage container:



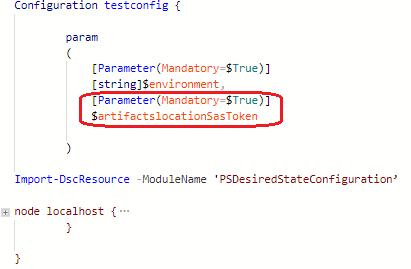
In the “configurationArguments” section we can define any parameters that we want to pass to our DSC configuration. In our case we passed parameter "environment". Please note that this property is not encrypted.





Azure VM extension protected setting data is encrypted, and only decrypted on the target virtual machine. In our DSC resource template we used the “protectedSettings” section when passing security data to DSC configuration. Properties listed under “protectedSettings” are encrypted with a certificate and are not shown in plain text in the settings file on the VM:

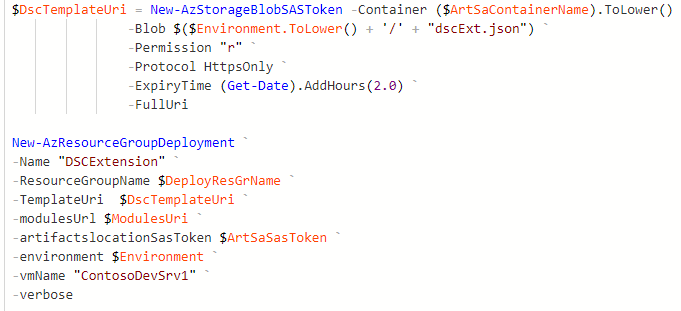




After it we Uploaded DSC script to Azure blob storage via Powershell cmdlet - Publish-AzureRmVMDscConfiguration. Also we upload resource template dscExt.json to Azure storage blob simultaneously with another resource templates.



The next step is deploying all above resources via PowerShell.



# Powershell

We'll start by describing the list of main steps during preparing and starting our deployment in Azure.

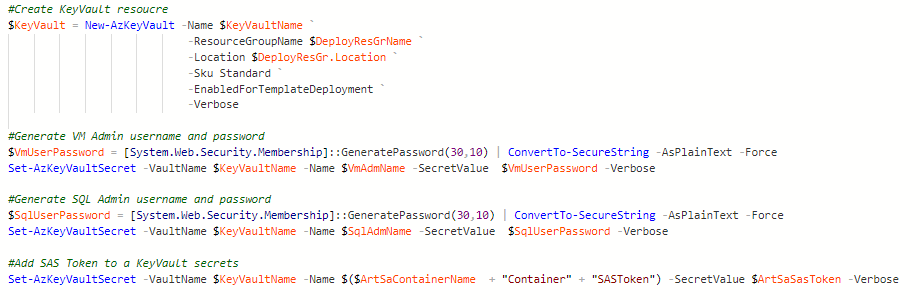
Describing variables:



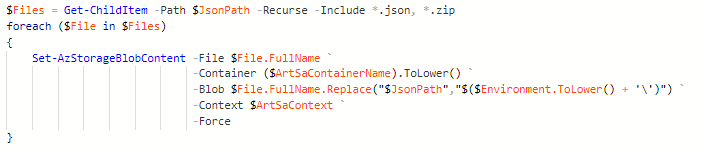
Creating resource groups with initial resources (two resource groups, one storage account with container). One recourse group called “Artifactory” will contain storage account with all templates which will be used for deployment, another group “TestDeploy” will contain whole environment with full set of resources. Please note that resource group cannot be created by ARM Template. *Нужно четкое обоснование для чего мы используем разные группы?*



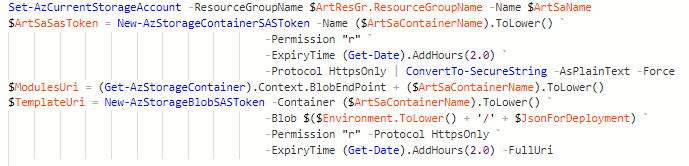
Creating KeyVault service with passwords and secrets. The elements of this service will be used to set the password for VM and SQL administrator when creating a resource.



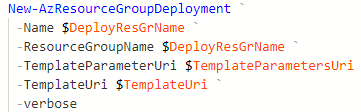
Uploading templates to early created storage account.



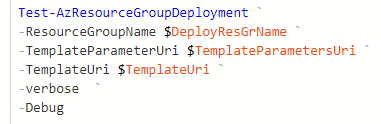
Generating Shared Access Signature (SAS) token for storage account. We will use this SAS token to securely connect and pull templates from the storage account.



Deploying our main template Base.json



Also before we start our deployment we can check whether an Azure resource group deployment template and its parameter values are valid. For this purpose in PowerShell we have appropriate cmdlet Test-AzResourceGroupDeployment.



For this cmdlet we specifies the name of the resource group, the URI of a template parameters file and the URI of a JSON template file.