Azure Resource Manager Template with DSC for SonarQube Installation in Azure

# What is SonarQube

Technical Debt is a growing problem for many developers, as while an application matures and code becomes more stable, it can also pile up to create inefficiencies and drain productivity from the team to create new enhancements. This occurs largely because the code become more complex, and by extension more difficult to understand. This creates unforeseen roadblocks that can slow the team’s ability to deliver value.

SonarQube is an open source platform that assist developers in isolating and managing technical debt. It provides in debt analysis and scanning and can be leveraged as part of a Build process. Over the past few months, SonarQube has greatly expanded its support of not only Java, but .net and JavaScript code analysis.

Despite all these advantages, the biggest roadblock to adoption for many organizations is the configuration and steps required to deploy a SonarQube instance. This solution provides a method of leveraging Azure Resource Manager (ARM) templates, and Powershell DSC (Desired State Configuration) to automate the creation of the necessary components to standup a SonarQube installation to support your development environment.

# What is Desired State Configuration (DSC)

# **What is Azure Resource Manager**

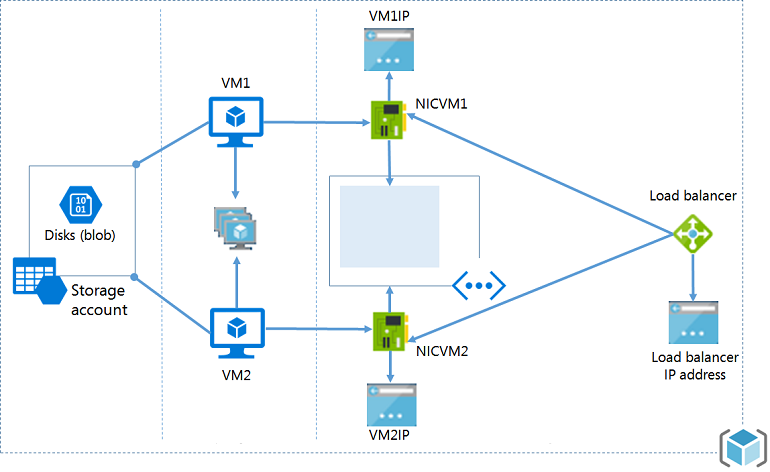
Azure Resource Manager allows you to provision your applications using a declarative template. In a single template, you can deploy multiple services along with their dependencies. You use the same template to repeatedly deploy your application during every stage of the application lifecycle.

<https://azure.microsoft.com/en-us/documentation/templates/>

One of the first questions when creating a template is "how to start?". One can start from a blank template, following the basic structure described in [Authoring Template article](https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/#template-format), and add the resources and appropriate parameters and variables. A good alternative would be to start by going through the [quickstart gallery](https://github.com/Azure/azure-quickstart-templates) and look for similar scenarios to the one you are trying to create. You can merge several templates or edit an existing one to suit your own specific scenario.

Let's take a look at a common infrastructure:

* Two virtual machines that use the same storage account, are in the same availability set, and on the same subnet of a virtual network.
* A single NIC and VM IP address for each virtual machine.
* A load balancer with a load balancing rule on port 80



This topic walks you through the steps of creating a Resource Manager template for that infrastructure. The final template you create is based on a Quickstart template called [2 VMs in a Load Balancer and load balancing rules](https://azure.microsoft.com/documentation/templates/201-2-vms-loadbalancer-lbrules/).

But, that's a lot to build all at once, so let's first create a storage account and deploy it. After you have mastered creating the storage account, you will add the other resources and re-deploy the template to complete the infrastructure.

##### **Note:**

You can use any type of editor when creating the template. Visual Studio provides tools that simplify template development, but you do not need Visual Studio to complete this tutorial. For a tutorial on using Visual Studio to create a Web App and SQL Database deployment, see [Creating and deploying Azure resource groups through Visual Studio](https://azure.microsoft.com/en-us/documentation/articles/vs-azure-tools-resource-groups-deployment-projects-create-deploy/).

## Create the Resource Manager template

The template is a JSON file that defines all of the resources you will deploy. It also permits you to define parameters that are specified during deployment, variables that constructed from other values and expressions, and outputs from the deployment.

Let's start with the simplest template:

Copy to clipboardCopy

{

"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": { },

"variables": { },

"resources": [ ],

"outputs": { }

}

Save this file as **azuredeploy.json** (note that the template can have any name you want, just that it must be a json file).

## Create a storage account

Within the **resources** section, add an object that defines the storage account, as shown below.

Copy to clipboardCopy

"resources": [

{

"type": "Microsoft.Storage/storageAccounts",

"name": "[parameters('storageAccountName')]",

"apiVersion": "2015-06-15",

"location": "[resourceGroup().location]",

"properties": {

"accountType": "Standard\_LRS"

}

}

]

You may be wondering where these properties and values come from. The properties **type**, **name**, **apiVersion**, and **location** are standard elements that are available for all resource types. You can learn about the common elements at [Resources](https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/#resources). **name** is set to a parameter value that you pass in during deployment and **location** as the location used by the resource group. We'll look at how you determine **type** and **apiVersion** in the sections below.

The **properties** section contains all of the properties that are unique to a particular resource type. The values you specify in this section exactly match the PUT operation in the REST API for creating that resource type. When creating a storage account, you must provide an **accountType**. Notice in the [REST API for creating a Storage account](https://msdn.microsoft.com/library/azure/mt163564.aspx) that the properties section of the REST operation also contains an **accountType** property, and the permitted values are documented. In this example, the account type is set to **Standard\_LRS**, but you could specify some other value or permit users to pass in the account type as a parameter.

Now let's jump back to the **parameters** section, and see how you define the name of the storage account. You can learn more about the use of parameters at [Parameters](https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/#parameters).

Copy to clipboardCopy

"parameters" : {

"storageAccountName": {

"type": "string",

"metadata": {

"description": "Storage Account Name"

}

}

}

Here you defined a parameter of type string that will hold the name of the storage account. The value for this parameter will be provided during template deployment.

## Deploying the template

We have a full template for creating a new storage account. As you recall, the template was saved in **azuredeploy.json** file:

Copy to clipboardCopy

{

"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters" : {

"storageAccountName": {

"type": "string",

"metadata": {

"description": "Storage Account Name"

}

}

},

"resources": [

{

"type": "Microsoft.Storage/storageAccounts",

"name": "[parameters('storageAccountName')]",

"apiVersion": "2015-06-15",

"location": "[resourceGroup().location]",

"properties": {

"accountType": "Standard\_LRS"

}

}

]

}

There are quite a few ways to deploy a template, as you can see in the [Resource Deployment article](https://azure.microsoft.com/en-us/documentation/articles/resource-group-template-deploy/). To deploy the template using Azure PowerShell, use:

# **Using SonarQube ARM Template**

# **Visual Studio Project:**

The solution is primarily made up of a C# Azure Resource Group project template. This is primarily consisting of two files

\*.json: Which is the configuration of the template.

\*.parameters.json: Which is a dictionary listing of the parameters to be leveraged during the deployment to customize the solution.

Azure Deploy Json file identifies several key points to delve into the configuration of this template:

* Variables: Contains a listing of the variables that are leveraged for configuration of this template.
* Resources: Contains all resources that are implemented as part of this ARM Solution. This includes, the VM, storage account, and NIC.

NOTE: Keep the name short, there is a 24-character limit for storage accounts, and the concatenation will add a guid, so keep the names of storage accounts short.

# **Template:**

This template contains several artifacts

Currently the key components from this template are the following items:

* **An SQL Azure Server with a SQL Azure Database:** This platform-as-a-service provides support for the underlying database required by SonarQube.
* **An IaaS(Infrastructure as a Service) Virtual Machine with jdk8:** This server provides the required support for the SonarQube application. And has SonarQube installed, with configuration to point to the SQL Azure database identified above.

