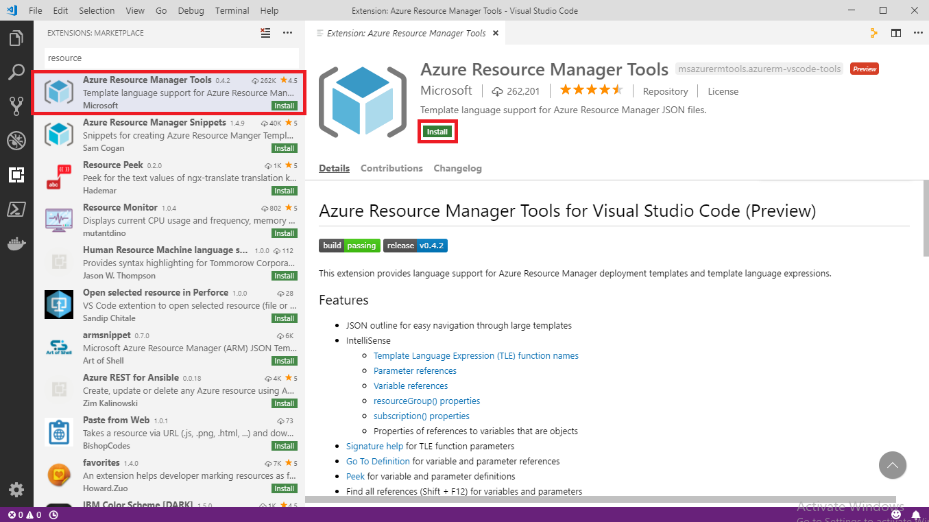
Create and deploy your first ARM template

Get tools

1. In the task we will use **Visual Studio Code** as our editor, which can be installed from here <https://code.visualstudio.com/>.
2. In **Visual Studio Code**, go to **File** > **Preferences** > **Extensions** and in the search box type **Azure Resource Manager** tools install the [Azure Resource Manager Tools](https://marketplace.visualstudio.com/items?itemName=msazurermtools.azurerm-vscode-tools)
3. [Install Azure CLI on Windows](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli), make sure you have version 2.6 or later, to check your installed version, use: az --version.

Create your first template

1. Open Visual Studio Code with the Resource Manager Tools extension installed.
2. From the **File** menu, select **New File** to create a new file.
3. From the **File** menu, select **Save as**.
4. Name the file **azuredeploy** and select the **JSON** file extension. The complete name of the file **azuredeploy.json**.
5. Save the file to your workstation. Select a path that is easy to remember because you'll provide that path later when deploying the template.
6. Copy and paste the following JSON into the file:

{

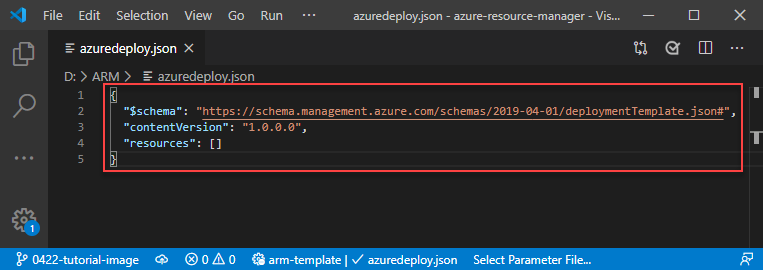
"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"resources": []

}

Here's what your VS Code environment looks like:



This template doesn't deploy any resources. We're starting with a blank template so you can get familiar with the steps to deploy a template while minimizing the chance of something going wrong.

1. Save the file.

Sign in to Azure

To start working with Azure PowerShell/Azure CLI, sign in with your Azure credentials.

az login

If you have multiple Azure subscriptions, select the subscription you want to use:

az account set --subscription [SubscriptionID/SubscriptionName]

Deploy template

To deploy the template, use either Azure CLI or Portal. Use the resource group that was created for you. Give a name to the deployment so you can easily identify it in the deployment history. For convenience, also create a variable that stores the path to the template file. This variable makes it easier for you to run the deployment commands because you don't have to retype the path every time you deploy.

templateFile="{provide-the-path-to-the-template-file}"

az deployment group create `

--name blanktemplate `

--resource-group myResourceGroup `

--template-file $templateFile

The deployment command returns results. Look for ProvisioningState to see whether the deployment succeeded.

Add resource

To add a storage account definition to the existing template, look at the highlighted JSON in the following example. Instead of trying to copy sections of the template, copy the whole file and replace your template with its contents.

Replace **{provide-unique-name}** (including the curly brackets) with a unique storage account name.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"resources": [

{

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

"apiVersion": "2019-04-01",

"name": "{provide-unique-name}",

"location": "eastus",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

]

}

Guessing a unique name for a storage account isn't easy and doesn't work well for automating large deployments. Later in this tutorial series, you'll use template features that make it easier to create a unique name.

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name addstorage `

--resource-group myResourceGroup `

--template-file $templateFile

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.
4. You see that a storage account has been deployed.
5. Notice that the deployment label now says: **Deployments: 2 Succeeded**.

Add parameters to your ARM template

To make your template reusable, let's add a parameter that you can use to pass in a storage account name. The **storageName** parameter is identified as a string. The max length is set to 24 characters to prevent any names that are too long. The previous template always deployed a Standard\_LRS storage account. You might want the flexibility to deploy different SKUs depending on the environment. The **storageSKU** parameter allows you to choose one of the available values

Copy the whole file and replace your template with its contents.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storageName": {

"type": "string",

"minLength": 3,

"maxLength": 24

},

"storageSKU": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_LRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Standard\_ZRS",

"Premium\_LRS",

"Premium\_ZRS",

"Standard\_GZRS",

"Standard\_RAGZRS"

]

}

},

"resources": [

{

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

"apiVersion": "2019-04-01",

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

"location": "eastus",

"sku": {

"name": "[parameters('storageSKU')]"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

]

}

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name addparameters `

--resource-group myResourceGroup `

--template-file $templateFile `

--parameters storageSKU=Standard\_GRS storageName={your-unique-name}

Clean up resources

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name.
4. Delete all resources in selected resource group.

Add template functions to your ARM template

You've already used a function. When you added **"[parameters('storageName')]"**, you used the [parameters](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-deployment#parameters) function. The brackets indicate that the syntax inside the brackets is a [template expression](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-expressions). Resource Manager resolves the syntax rather than treating it as a literal value.

Functions add flexibility to your template by dynamically getting values during deployment. In this tutorial, you use a function to get the location of the resource group you're using for deployment.

The following example changes a parameter called **location**. The parameter default value calls the [resourceGroup](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-resource" \l "resourcegroup) function. This function returns an object with information about the resource group being used for deployment. One of the properties on the object is a location property. When you use the default value, the storage account location has the same location as the resource group. The resources inside a resource group don't have to share the same location. You can also provide a different location when needed.

Copy the whole file and replace your template with its contents.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storageName": {

"type": "string",

"minLength": 3,

"maxLength": 24

},

"storageSKU": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_LRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Standard\_ZRS",

"Premium\_LRS",

"Premium\_ZRS",

"Standard\_GZRS",

"Standard\_RAGZRS"

]

},

"location": {

"type": "string",

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

}

},

"resources": [

{

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

"apiVersion": "2019-04-01",

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

"location": "[parameters('location')]",

"sku": {

"name": "[parameters('storageSKU')]"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

]

}

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name addfunction `

--resource-group myResourceGroup `

--template-file $templateFile `

--parameters storageName={your-unique-name}

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.
4. You see that a storage account resource has been deployed and has the same location as the resource group.

Add variables to your ARM template

The following example highlights the changes to add a variable to your template that creates a unique storage account name. Copy the whole file and replace your template with its contents.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storagePrefix": {

"type": "string",

"minLength": 3,

"maxLength": 11

},

"storageSKU": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_LRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Standard\_ZRS",

"Premium\_LRS",

"Premium\_ZRS",

"Standard\_GZRS",

"Standard\_RAGZRS"

]

},

"location": {

"type": "string",

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

}

},

"variables": {

"uniqueStorageName": "[concat(parameters('storagePrefix'), uniqueString(resourceGroup().id))]"

},

"resources": [

{

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

"apiVersion": "2019-04-01",

"name": "[variables('uniqueStorageName')]",

"location": "[parameters('location')]",

"sku": {

"name": "[parameters('storageSKU')]"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

]

}

Notice that it includes a variable named **uniqueStorageName**. This variable uses four functions to construct a string value.

You're already familiar with the [parameters](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-deployment#parameters) function, so we won't examine it.

You're also familiar with the [resourceGroup](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-resource" \l "resourcegroup) function. In this case, you get the **id** property instead of the **location** property, as shown in the previous tutorial. The **id** property returns the full identifier of the resource group, including the subscription ID and resource group name.

The [uniqueString](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-string" \l "uniquestring) function creates a 13 character hash value. The returned value is determined by the parameters you pass in. For this tutorial, you use the resource group ID as the input for the hash value. That means you could deploy this template to different resource groups and get a different unique string value. However, you get the same value if you deploy to the same resource group.

The [concat](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-string" \l "concat) function takes values and combines them. For this variable, it takes the string from the parameter and the string from the uniqueString function, and combines them into one string.

The **storagePrefix** parameter enables you to pass in a prefix that helps you identify storage accounts. You can create your own naming convention that makes it easier to identify storage accounts after deployment from a long list of resources.

Finally, notice that the storage name is now set to the variable instead of a parameter.

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name addnamevariable `

--resource-group myResourceGroup `

--template-file $templateFile `

--parameters storagePrefix={your-user-prefix}

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.

Add outputs

You can use outputs to return values from the template. For example, it might be helpful to get the endpoints for your new storage account.

The following example highlights the change to your template to add an output value. Copy the whole file and replace your template with its contents.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storagePrefix": {

"type": "string",

"minLength": 3,

"maxLength": 11

},

"storageSKU": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_LRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Standard\_ZRS",

"Premium\_LRS",

"Premium\_ZRS",

"Standard\_GZRS",

"Standard\_RAGZRS"

]

},

"location": {

"type": "string",

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

}

},

"variables": {

"uniqueStorageName": "[concat(parameters('storagePrefix'), uniqueString(resourceGroup().id))]"

},

"resources": [

{

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

"apiVersion": "2019-04-01",

"name": "[variables('uniqueStorageName')]",

"location": "[parameters('location')]",

"sku": {

"name": "[parameters('storageSKU')]"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

],

"outputs": {

"storageEndpoint": {

"type": "object",

"value": "[reference(variables('uniqueStorageName')).primaryEndpoints]"

}

}

}

There are some important items to note about the output value you added.

The type of returned value is set to **object**, which means it returns a JSON object.

It uses the [reference](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-resource#reference) function to get the runtime state of the storage account. To get the runtime state of a resource, you pass in the name or ID of a resource. In this case, you use the same variable you used to create the name of the storage account.

Finally, it returns the **primaryEndpoints** property from the storage account

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name addoutputs `

--resource-group myResourceGroup `

--template-file $templateFile `

--parameters storagePrefix={your-user-prefix} storageSKU=Standard\_LRS

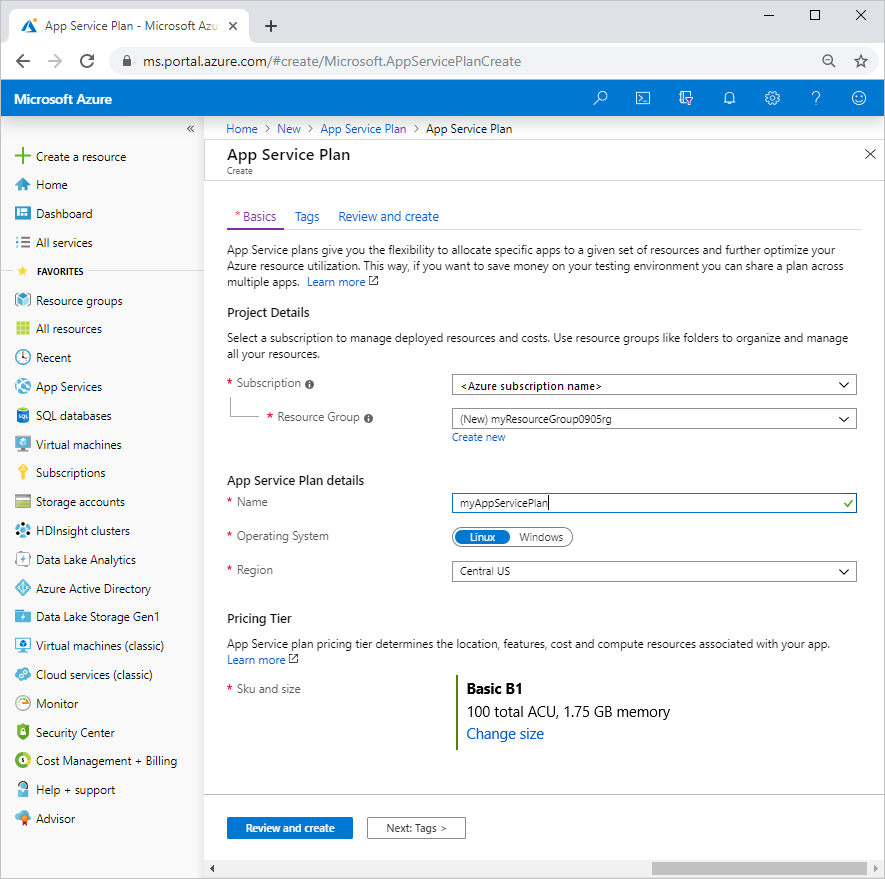
Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.

Use exported template from the Azure portal

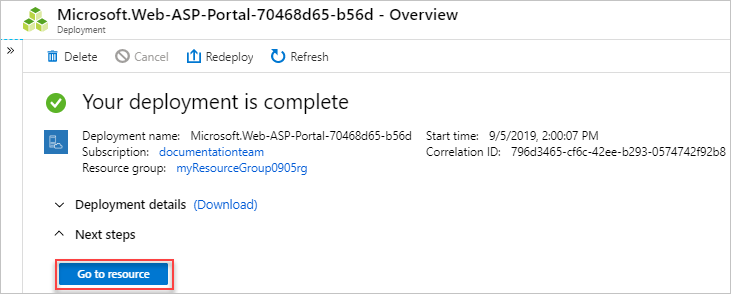
1. Sign in to the [Azure portal](https://portal.azure.com/).
2. Select **Create a resource**.
3. In **Search the Marketplace**, enter **App Service plan**, and then select **App Service plan**. Don't select **App Service plan (classic)**
4. Select **Create**.
5. Enter:
   * **Subscription**: select your Azure subscription.
   * **Resource Group**: Select **Create new** and then specify a name. Provide a different resource group name than the one you have been using in this tutorial series.
   * **Name**: enter a name for the App service plan.
   * **Operating System**: select **Linux**.
   * **Region**: select an Azure location. For example, **Central US**.
   * **Pricing tier**: to save costs, change the SKU to **Basic B1** (under Dev/Test).



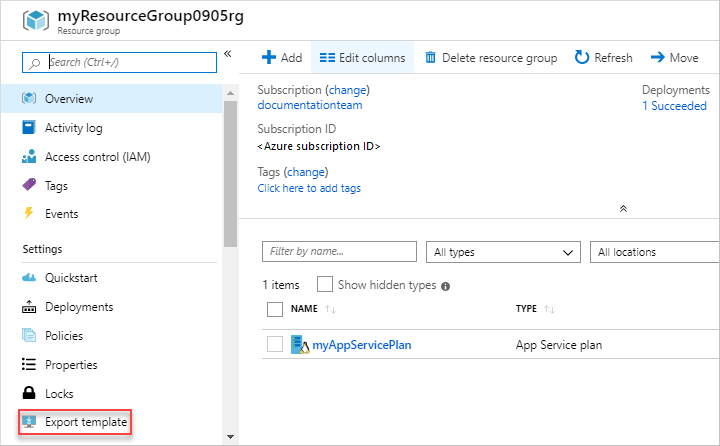
1. Select **Review and create**.
2. Select **Create**. It takes a few moments to create the resource.

Export template

1. Select **Go to resource**.



1. Select **Export template**.



The export template feature takes the current state of a resource and generates a template to deploy it. Exporting a template can be a helpful way of quickly getting the JSON you need to deploy a resource.

1. Look at the **Microsoft.Web/serverfarms** definition and the parameter definition in the exported template. You don't need to copy these sections. You can just use this exported template as an example of how you want to add this resource to your template.



Revise existing template

The exported template gives you most of the JSON you need, but you need to customize it for your template. Pay particular attention to differences in parameters and variables between your template and the exported template. Obviously, the export process doesn't know the parameters and variables that you've already defined in your template.

The following example highlights the additions to your template. It contains the exported code plus some changes. First, it changes the name of the parameter to match your naming convention. Second, it uses your location parameter for the location of the app service plan. Third, it removes some of the properties where the default value is fine.

Copy the whole file and replace your template with its contents.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storagePrefix": {

"type": "string",

"minLength": 3,

"maxLength": 11

},

"storageSKU": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_LRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Standard\_ZRS",

"Premium\_LRS",

"Premium\_ZRS",

"Standard\_GZRS",

"Standard\_RAGZRS"

]

},

"location": {

"type": "string",

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

},

"appServicePlanName": {

"type": "string",

"defaultValue": "exampleplan"

}

},

"variables": {

"uniqueStorageName": "[concat(parameters('storagePrefix'), uniqueString(resourceGroup().id))]"

},

"resources": [

{

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

"apiVersion": "2019-04-01",

"name": "[variables('uniqueStorageName')]",

"location": "[parameters('location')]",

"sku": {

"name": "[parameters('storageSKU')]"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

},

{

"type": "Microsoft.Web/serverfarms",

"apiVersion": "2016-09-01",

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

"location": "[parameters('location')]",

"sku": {

"name": "B1",

"tier": "Basic",

"size": "B1",

"family": "B",

"capacity": 1

},

"kind": "linux",

"properties": {

"perSiteScaling": false,

"reserved": true,

"targetWorkerCount": 0,

"targetWorkerSizeId": 0

}

}

],

"outputs": {

"storageEndpoint": {

"type": "object",

"value": "[reference(variables('uniqueStorageName')).primaryEndpoints]"

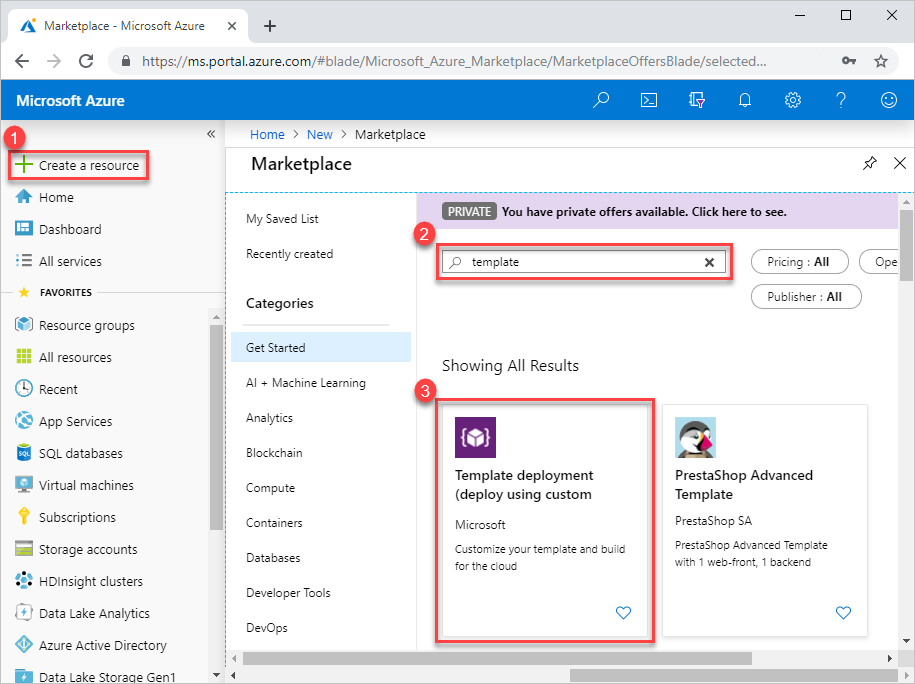
}

}

}

Deploy template from portal

1. To deploy a customized template through the portal, select **Create a resource**, search for **template**. and then select **Template deployment**.



1. Select **Create**.
2. Select **Build your own template in editor**: Create your own template in the portal template editor.
3. Import template file
4. Import parameters file. Customize parameters if needed.
5. Launch deployment

Use Azure Quickstart templates

## Find template

1. Open [Azure Quickstart templates](https://azure.microsoft.com/resources/templates/)
2. In **Search**, enter **deploy linux web app**.
3. Select the one with the title **Deploy a basic Linux web app**. If you have trouble finding it, here's the [direct link](https://azure.microsoft.com/resources/templates/101-webapp-basic-linux/).
4. Select **Browse on GitHub**.
5. Select **azuredeploy.json**.
6. Review the template. In particular, look for the Microsoft.Web/sites resource.

Revise existing template

Merge the quickstart template with the existing template:

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storagePrefix": {

"type": "string",

"minLength": 3,

"maxLength": 11

},

"storageSKU": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_LRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Standard\_ZRS",

"Premium\_LRS",

"Premium\_ZRS",

"Standard\_GZRS",

"Standard\_RAGZRS"

]

},

"location": {

"type": "string",

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

},

"appServicePlanName": {

"type": "string",

"defaultValue": "exampleplan"

},

"webAppName": {

"type": "string",

"metadata": {

"description": "Base name of the resource such as web app name and app service plan "

},

"minLength": 2

},

"linuxFxVersion": {

"type": "string",

"defaultValue": "php|7.0",

"metadata": {

"description": "The Runtime stack of current web app"

}

}

},

"variables": {

"uniqueStorageName": "[concat(parameters('storagePrefix'), uniqueString(resourceGroup().id))]",

"webAppPortalName": "[concat(parameters('webAppName'), uniqueString(resourceGroup().id))]"

},

"resources": [

{

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

"apiVersion": "2019-04-01",

"name": "[variables('uniqueStorageName')]",

"location": "[parameters('location')]",

"sku": {

"name": "[parameters('storageSKU')]"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

},

{

"type": "Microsoft.Web/serverfarms",

"apiVersion": "2016-09-01",

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

"location": "[parameters('location')]",

"sku": {

"name": "B1",

"tier": "Basic",

"size": "B1",

"family": "B",

"capacity": 1

},

"kind": "linux",

"properties": {

"perSiteScaling": false,

"reserved": true,

"targetWorkerCount": 0,

"targetWorkerSizeId": 0

}

},

{

"type": "Microsoft.Web/sites",

"apiVersion": "2018-11-01",

"name": "[variables('webAppPortalName')]",

"location": "[parameters('location')]",

"dependsOn": [

"[resourceId('Microsoft.Web/serverfarms', parameters('appServicePlanName'))]"

],

"kind": "app",

"properties": {

"serverFarmId": "[resourceId('Microsoft.Web/serverfarms', parameters('appServicePlanName'))]",

"siteConfig": {

"linuxFxVersion": "[parameters('linuxFxVersion')]"

}

}

}

],

"outputs": {

"storageEndpoint": {

"type": "object",

"value": "[reference(variables('uniqueStorageName')).primaryEndpoints]"

}

}

}

The web app name needs to be unique across Azure. To prevent having duplicate names, the **webAppPortalName** variable has been updated from **"webAppPortalName": "[concat(parameters('webAppName'), '-webapp')]"** to **"webAppPortalName": "[concat(parameters('webAppName'), uniqueString(resourceGroup().id))]"**.

Add a comma at the end of the Microsoft.Web/serverfarms definition to separate the resource definition from the Microsoft.Web/sites definition.

There are a couple of important features to note in this new resource.

You'll notice it has an element named **dependsOn** that is set to the app service plan. This setting is required because the app service plan must exist before the web app is created. The **dependsOn** element tells Resource Manager how to order the resources for deployment.

The **serverFarmId** property uses the [resourceId](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions-resource" \l "resourceid) function. This function gets the unique identifier for a resource. In this case, it gets the unique identifier for the app service plan. The web app is associated with one specific app service plan.

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name addwebapp `

--resource-group myResourceGroup `

--template-file $templateFile `

--parameters storagePrefix={your-user-prefix} storageSKU=Standard\_LRS webAppName=demoapp

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.

Add tags

You tag resources to add values that help you identify their use. For example, you can add tags that list the environment and the project. You could add tags that identify a cost center or the team that owns the resource. Add any values that make sense for your organization.

The following example highlights the changes to the template. Copy the whole file and replace your template with its contents.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storagePrefix": {

"type": "string",

"minLength": 3,

"maxLength": 11

},

"storageSKU": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_LRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Standard\_ZRS",

"Premium\_LRS",

"Premium\_ZRS",

"Standard\_GZRS",

"Standard\_RAGZRS"

]

},

"location": {

"type": "string",

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

},

"appServicePlanName": {

"type": "string",

"defaultValue": "exampleplan"

},

"webAppName": {

"type": "string",

"metadata": {

"description": "Base name of the resource such as web app name and app service plan "

},

"minLength": 2

},

"linuxFxVersion": {

"type": "string",

"defaultValue": "php|7.0",

"metadata": {

"description": "The Runtime stack of current web app"

}

},

"resourceTags": {

"type": "object",

"defaultValue": {

"Environment": "Dev",

"Project": "Tutorial"

}

}

},

"variables": {

"uniqueStorageName": "[concat(parameters('storagePrefix'), uniqueString(resourceGroup().id))]",

"webAppPortalName": "[concat(parameters('webAppName'), uniqueString(resourceGroup().id))]"

},

"resources": [

{

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

"apiVersion": "2019-04-01",

"name": "[variables('uniqueStorageName')]",

"location": "[parameters('location')]",

"tags": "[parameters('resourceTags')]",

"sku": {

"name": "[parameters('storageSKU')]"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

},

{

"type": "Microsoft.Web/serverfarms",

"apiVersion": "2016-09-01",

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

"location": "[parameters('location')]",

"tags": "[parameters('resourceTags')]",

"sku": {

"name": "B1",

"tier": "Basic",

"size": "B1",

"family": "B",

"capacity": 1

},

"kind": "linux",

"properties": {

"perSiteScaling": false,

"reserved": true,

"targetWorkerCount": 0,

"targetWorkerSizeId": 0

}

},

{

"type": "Microsoft.Web/sites",

"apiVersion": "2016-08-01",

"name": "[variables('webAppPortalName')]",

"location": "[parameters('location')]",

"dependsOn": [

"[parameters('appServicePlanName')]"

],

"tags": "[parameters('resourceTags')]",

"kind": "app",

"properties": {

"serverFarmId": "[resourceId('Microsoft.Web/serverfarms', parameters('appServicePlanName'))]",

"siteConfig": {

"linuxFxVersion": "[parameters('linuxFxVersion')]"

}

}

}

],

"outputs": {

"storageEndpoint": {

"type": "object",

"value": "[reference(variables('uniqueStorageName')).primaryEndpoints]"

}

}

}

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name addwebapp `

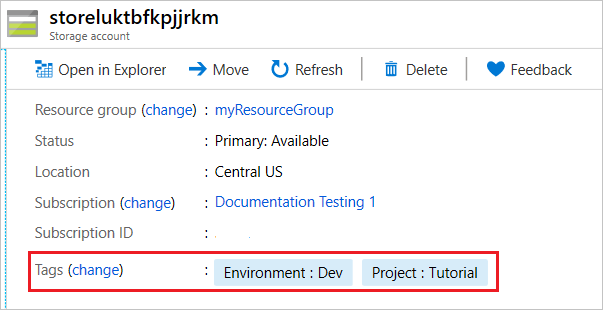
--resource-group myResourceGroup `

--template-file $templateFile `

--parameters storagePrefix={your-user-prefix} storageSKU=Standard\_LRS webAppName=demoapp

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.
4. Select one of the resources, such as the storage account resource. You see that it now has tags.

Use parameter files to deploy your ARM template

Parameter files are JSON files with a structure that is similar to your template. In the file, you provide the parameter values you want to pass in during deployment.

Within the parameter file, you provide values for the parameters in your template. The name of each parameter in your parameter file must match the name of a parameter in your template. The name is case-insensitive but to easily see the matching values we recommend that you match the casing from the template.

You don't have to provide a value for every parameter. If an unspecified parameter has a default value, that value is used during deployment. If a parameter doesn't have a default value and isn't specified in the parameter file, you're prompted to provide a value during deployment.

You can't specify a parameter name in your parameter file that doesn't match a parameter name in the template. You get an error when unknown parameters are provided.

In VS Code, create a new file with following content. Save the file with the name **azuredeploy.parameters.dev.json**.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storagePrefix": {

"value": "devstore"

},

"storageSKU": {

"value": "Standard\_LRS"

},

"appServicePlanName": {

"value": "devplan"

},

"webAppName": {

"value": "devapp"

},

"resourceTags": {

"value": {

"Environment": "Dev",

"Project": "Tutorial"

}

}

}

}

This file is your parameter file for the development environment. Notice that it uses Standard\_LRS for the storage account, names resources with a **dev** prefix, and sets the **Environment** tag to **Dev**.

Again, create a new file with the following content. Save the file with the name **azuredeploy.parameters.prod.json**.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storagePrefix": {

"value": "contosodata"

},

"storageSKU": {

"value": "Standard\_GRS"

},

"appServicePlanName": {

"value": "contosoplan"

},

"webAppName": {

"value": "contosowebapp"

},

"resourceTags": {

"value": {

"Environment": "Production",

"Project": "Tutorial"

}

}

}

}

This file is your parameter file for the production environment. Notice that it uses Standard\_GRS for the storage account, names resources with a **contoso** prefix, and sets the **Environment** tag to **Production**. In a real production environment, you would also want to use an app service with a SKU other than free, but we'll continue to use that SKU for this tutorial.

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

templateFile="{path-to-the-template-file}"

devParameterFile="{path-to-azuredeploy.parameters.dev.json}"

az deployment group create `

--name devenvironment `

--resource-group myResourceGroupDev `

--template-file $templateFile `

--parameters $devParameterFile

Now, we'll deploy to the production environment.

prodParameterFile="{path-to-azuredeploy.parameters.prod.json}"

az deployment group create \

--name prodenvironment \

--resource-group myResourceGroupProd \

--template-file $templateFile \

--parameters $prodParameterFile

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.

Clean up resources

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name.
4. Delete all resources in selected resource group.