Bicep

Inhoud

[What is bicep. 3](#_Toc104635000)

[Bicep and ARM templates. 3](#_Toc104635001)

[Creating our first template. 4](#_Toc104635002)

[Example scenario. 4](#_Toc104635003)

[Define resources. 4](#_Toc104635004)

[Defining our first resource. 4](#_Toc104635005)

[Comparing Bicep with the portal. 6](#_Toc104635006)

[Creating a resource that depends on another. 7](#_Toc104635007)

[Sources: 9](#_Toc104635008)

# What is bicep.

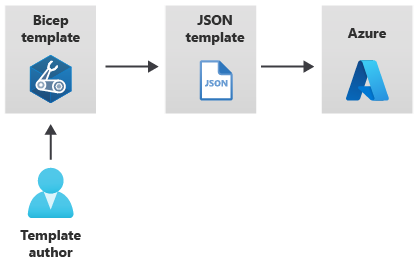
Bicep is a language for deploying azure resources by using simple templates. You can define how your Azure resources should be configured and deployed. The template you create using Bicep will be submitted to the Azure Resource Manager(ARM). The Resource Manager takes the template and deploys each resource within the template.

# Bicep and ARM templates.

ARM templates are files that represent Azure resources just like a Bicep template. Before Bicep ARM templates had to be written in a special JSON format. One of the common complains and problems with JSON templates is that they are difficult to work with because of the complex syntax. It can be hard to get started with ARM templates in JSON.

Bicep is the solution to these problems by using a much simpler language designed specifically to help you deploy resources to Azure.

In the background ARM and its JSON templates are still relevant but on the forefront you only need to use Bicep. When you submit a Bicep template to the Resource Manager, the Bicep template is converted to a JSON format.

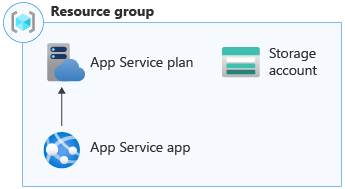


# Creating our first template.

So I use VS code to create Bicep templates with the Bicep extension. Bicep templates are files you author.

## Example scenario.

Suppose you're responsible for deploying and configuring the Azure infrastructure at a toy company. Your company is launching a new toy. You've been asked to deploy a new marketing website in preparation for the launch. The marketing team has also told you that many new toys will be released this year, and it will need a website deployed for each one. You know from past experience that you're not always given much time to get new websites up and running before a toy launches.

You'll host the website in Azure using Azure App Service. You'll incorporate a storage account for files, such as manuals and specifications, for the toy. 

# Define resources.

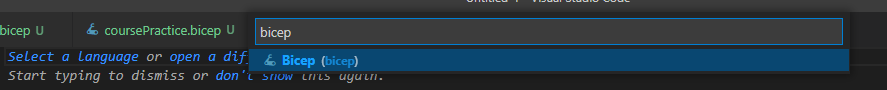
So the toy company is going to need a reusable Bicep template for product launches. The template will contain a Storage account and a App service resource.

## Defining our first resource.

So the first resource we will define in our template is the storage account I will show a step by step process on how this works in VS code:

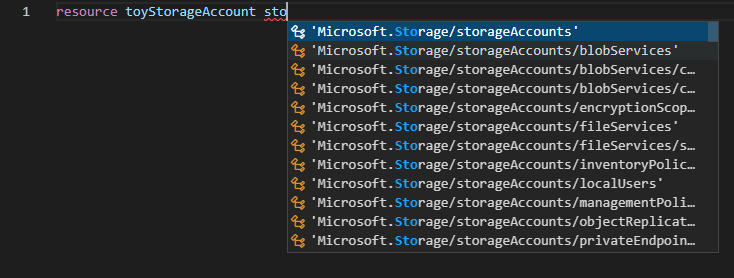
**Step 1 create a Bicep file.**

So if you have the Bicep extension installed you should have to option when creating a new file to choose Bicep as a language.

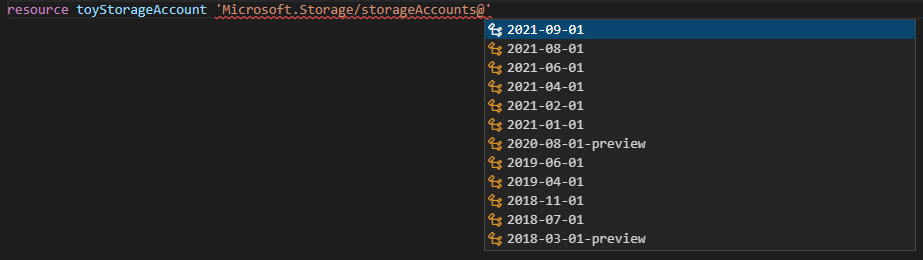


**Step 2 add a resource.**

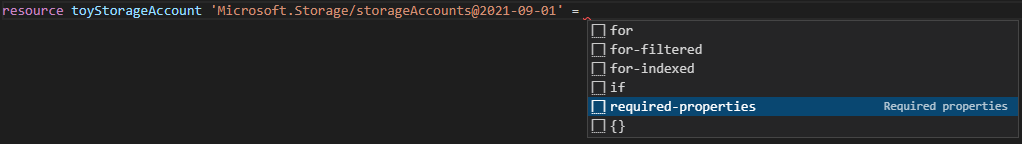
So now that we have our file we can start adding the storage account resource.



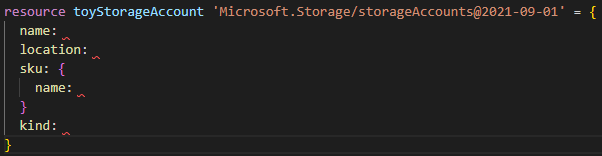
So first we need to type resource and then give it a name the name of the resource does not matter since it will not be used by azure so make it a name that is relevant for you. Now after the name you need to specify what kind of resource to is in this case we want a storage account. As you can see thanks to the extension if we start typing out storage we get a lot of options we just pick the default storage account one.



Now that we have done that it will ask us for what looks like a date and it is but what it represents in the API version of the resource I usually take the latest one.



Now that we have done that and add a = at the end we get these options we choose required-properties here and what this does is it will add a bunch of text kind of like a form that needs to be filled in in order to create the Storage account.



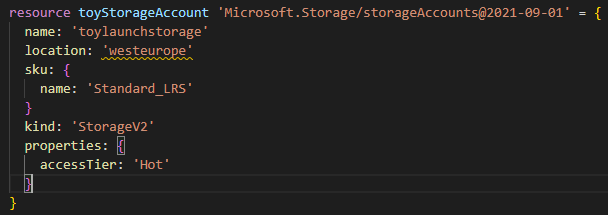
A quick recap of what all the steps we just moved through actually do.

The **resource** keyword at the start tells Bicep that you're about to define a resource.

Next, you give the resource a symbolic name. In the example, the resource's symbolic name is **toyStorageAccount**. Symbolic names are used within Bicep to refer to the resource, but they won't ever show up in Azure.

**Microsoft.Storage/storageAccounts@2021-09-01** is the resource type and API version of the resource. **Microsoft.Storage/storageAccounts** tells Bicep that you're declaring an Azure storage account. **2021-09-01** is the version of the Azure Storage API that Bicep will use when it creates the resource.

Now all we need to do is fill in the blanks and we are done with our first resource



Now as you might have noticed we get a waring at the location variable that is because Bicep wants you to use parameters for this variable we will get to that soon.

## Comparing Bicep with the portal.

So lets compare the options in Bicep and the portal real quick so we can see how easy it is to create a resource in Bicep.

So we create a storage account in bicep

resource toyStorageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = {

  name: 'toylaunchstorage'

  location: 'westeurope'

  sku: {

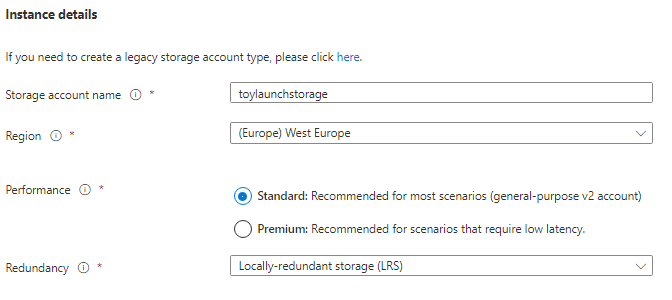
    name: 'Standard\_LRS'

  }

  kind: 'StorageV2'

}

Now lets see if the minimum required options here are the same as in the portal.



So here we can see the basic option in the portal are identical to the minimum required options in Bicep.

Name: = storage account name.  
Location: = region  
Sku: name = Redundancy option.  
Kind: = Preformance

# Creating a resource that depends on another.

A Bicep template will probably have multiple resources. Often some of those resources depend on the others. You might need to use some information from one resource in order to create the other. Or in this case you are going to need a App service plan before you can create a web application. These relationships are called **dependencies**.

So in order to create a web App we are going to need a App service plan. The App service plan represents the server-hosting resources and is declared in Bicep like this:

//First we create the appServicePlan

resource appServicePlan 'Microsoft.Web/serverFarms@2021-03-01' = {

  name: 'toy-product-launch-plan-starter' //Make sure to pick a unqiue name

  location: locationWEU

  sku: {

    name: 'F1'

  }

}

Now that we have our app service plan we can go ahead and add our Web App like so:

//We create webApp with service plan id.

resource appServiceApp 'Microsoft.Web/sites@2021-03-01' = {

  name: 'toy-product-launch-1' //Make sure to pick a unqiue name

  location: locationWEU

  properties: {

    serverFarmId: appServicePlan.id

    httpsOnly: true

  }

}

# Modules.

Modules are Bicep files that you can reference to in you main bicep file. Basically they are small building blocks that come together in the main file so instead of putting everything in the main you split it up in modules. This makes the files easier to manage and provides greater reusability.

## Outputs.

# Sources:

<https://docs.microsoft.com/en-us/azure/azure-resource-manager/bicep/quickstart-create-bicep-use-visual-studio-code?tabs=CLI>

<https://docs.microsoft.com/en-us/learn/modules/build-first-bicep-template/>

<https://docs.microsoft.com/en-us/learn/paths/fundamentals-bicep/>

<https://docs.microsoft.com/en-us/learn/paths/intermediate-bicep/>

<https://docs.microsoft.com/en-us/learn/paths/advanced-bicep/>

<https://docs.microsoft.com/en-us/cli/azure/manage-azure-subscriptions-azure-cli>

<https://docs.microsoft.com/en-us/azure/azure-resource-manager/bicep/file>