Tutorial: Deploy a multi-container group using a Resource Manager template

Article • 06/17/2022 • 4 minutes to read

Azure Container Instances supports the deployment of multiple containers onto a single host using a container group. A container group is useful when building an application sidecar for logging, monitoring, or any other configuration where a service needs a second attached process.

In this tutorial, you follow steps to run a simple two-container sidecar configuration by deploying an Azure Resource Manager template using the Azure CLI. You learn how to:

- ✓ Configure a multi-container group template
- ✓ Deploy the container group
- ✓ View the logs of the containers

A Resource Manager template can be readily adapted for scenarios when you need to deploy additional Azure service resources (for example, an Azure Files share or a virtual network) with the container group.

> **Note**
>
> Multi-container groups are currently restricted to Linux containers.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

You can use either the Azure Cloud Shell or a local Azure CLI.

- Azure Cloud Shell with the Bash environment. Or launch the Cloud Shell here.

  Launch Cloud Shell

- Local Azure CLI, see how to install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - Sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the

Azure CLI.

- When you first use Azure CLI, install the Azure CLI extension. For more information about extensions, see Use extensions with the Azure CLI.
- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Configure a template

Start by copying the following JSON into a new file named `azuredeploy.json`. In Azure Cloud Shell, you can use Visual Studio Code to create the file in your working directory:

```
code azuredeploy.json
```

This Resource Manager template defines a container group with two containers, a public IP address, and two exposed ports. The first container in the group runs an internet-facing web application. The second container, the sidecar, makes an HTTP request to the main web application via the group's local network.

JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "containerGroupName": {
      "type": "string",
      "defaultValue": "myContainerGroup",
      "metadata": {
        "description": "Container Group name."
      }
    }
  },
  "variables": {
    "container1name": "aci-tutorial-app",
    "container1image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
    "container2name": "aci-tutorial-sidecar",
    "container2image": "mcr.microsoft.com/azuredocs/aci-tutorial-sidecar"
  },
  "resources": [
    {
      "name": "[parameters('containerGroupName')]",
      "type": "Microsoft.ContainerInstance/containerGroups",
      "apiVersion": "2019-12-01
```

```
    "location": "[resourceGroup().location]",
    "properties": {
      "containers": [
        {
          "name": "[variables('container1name')]",
          "properties": {
            "image": "[variables('container1image')]",
            "resources": {
              "requests": {
                "cpu": 1,
                "memoryInGb": 1.5
              }
            },
            "ports": [
              {
                "port": 80
              },
              {
                "port": 8080
              }
            ]
          }
        },
        {
          "name": "[variables('container2name')]",
          "properties": {
            "image": "[variables('container2image')]",
            "resources": {
              "requests": {
                "cpu": 1,
                "memoryInGb": 1.5
              }
            }
          }
        }
      ],
      "osType": "Linux",
      "ipAddress": {
        "type": "Public",
        "ports": [
          {
            "protocol": "tcp",
            "port": 80
          },
          {
              "protocol": "tcp",
              "port": 8080
          }
        ]
      }
    }
  }
],
"outputs": {
  "containerIPv4Address": {
```

```
      "type": "string",
      "value": "[reference(resourceId('Microsoft.ContainerInstance/containerGroups/',
parameters('containerGroupName'))).ipAddress.ip]"
    }
  }
}
```

To use a private container image registry, add an object to the JSON document with the following format. For an example implementation of this configuration, see the ACI Resource Manager template reference documentation.

JSON

```
"imageRegistryCredentials": [
  {
    "server": "[parameters('imageRegistryLoginServer')]",
    "username": "[parameters('imageRegistryUsername')]",
    "password": "[parameters('imageRegistryPassword')]"
  }
]
```

Deploy the template

Create a resource group with the az group create command.

Azure CLI

```
az group create --name myResourceGroup --location eastus
```

Deploy the template with the az deployment group create command.

Azure CLI

```
az deployment group create --resource-group myResourceGroup --template-file
azuredeploy.json
```

Within a few seconds, you should receive an initial response from Azure.

View deployment state

To view the state of the deployment, use the following az container show command:

Azure CLI


```
az container show --resource-group myResourceGroup --name myContainerGroup --output table
```

If you'd like to view the running application, navigate to its IP address in your browser. For example, the IP is `52.168.26.124` in this example output:

Console

```
Name              ResourceGroup    Status    Image
IP:ports              Network    CPU/Memory       OsType    Location
---------------  --------------  --------  ----------------------------------------
-------------------------------------------------  -------------------  ---------
--------------  --------  ----------
myContainerGroup  danlep0318r      Running   mcr.microsoft.com/azuredocs/aci-tutorial-
sidecar,mcr.microsoft.com/azuredocs/aci-helloworld:latest  20.42.26.114:80,8080  Public
1.0 core/1.5 gb  Linux     eastus
```

View container logs

View the log output of a container using the az container logs command. The `--container-name` argument specifies the container from which to pull logs. In this example, the `aci-tutorial-app` container is specified.

Azure CLI

```
az container logs --resource-group myResourceGroup --name myContainerGroup --container-
name aci-tutorial-app
```

Output:

Console

```
listening on port 80
::1 - - [02/Jul/2020:23:17:48 +0000] "HEAD / HTTP/1.1" 200 1663 "-" "curl/7.54.0"
::1 - - [02/Jul/2020:23:17:51 +0000] "HEAD / HTTP/1.1" 200 1663 "-" "curl/7.54.0"
::1 - - [02/Jul/2020:23:17:54 +0000] "HEAD / HTTP/1.1" 200 1663 "-" "curl/7.54.0"
```

To see the logs for the sidecar container, run a similar command specifying the `aci-tutorial-sidecar` container.

Azure CLI

```
az container logs --resource-group myResourceGroup --name myContainerGroup --container-
name aci-tutorial-sidecar
```

Output:

Console

```
Every 3s: curl -I http://localhost                          2020-07-02 20:36:41

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0  1663    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 29 Nov 2017 06:40:40 GMT
ETag: W/"67f-16006818640"
Content-Type: text/html; charset=UTF-8
Content-Length: 1663
Date: Thu, 02 Jul 2020 20:36:41 GMT
Connection: keep-alive
```

As you can see, the sidecar is periodically making an HTTP request to the main web application via the group's local network to ensure that it is running. This sidecar example could be expanded to trigger an alert if it received an HTTP response code other than `200 OK`.

Next steps

In this tutorial, you used an Azure Resource Manager template to deploy a multi-container group in Azure Container Instances. You learned how to:

- ✓ Configure a multi-container group template
- ✓ Deploy the container group
- ✓ View the logs of the containers

For additional template samples, see Azure Resource Manager templates for Azure Container Instances.

You can also specify a multi-container group using a YAML file. Due to the YAML format's more concise nature, deployment with a YAML file is a good choice when your deployment includes only container instances.