Deployment – Docs

Table of Contents

Introduction

This documentation provides step-by-step instructions for deploying a containerized application to an Azure Kubernetes Service (AKS) cluster using an Azure Container Registry (ACR) with a PowerShell script. No technical expertise is required to follow the steps outlined in this guide.

Prerequisites

Before using this script, ensure the following requirements are met:

1. You have a Windows machine with PowerShell 5.1 or later installed.
2. You have Git installed on your machine. If not, download and install it from Git website.
3. You have Docker installed on your machine. If not, download and install it from Docker website.
4. Scripts must be allowed to run on the local machine.
    - You can verify this by opening PowerShell and run the following command:

```
# If the output is "Restricted" you are not allowed to run scripts
Get-ExecutionPolicy
```

    - If scripts are not allowed to be run on the system, open PowerShell with administrator privileges and run the command:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

5. Azure PowerShell module is installed. If not, run the following command in PowerShell to install it:

```
Install-Module -Name Az -AllowClobber -Scope CurrentUser
```

6. The powershell-yaml module is installed. If not, run the following command in PowerShell to install it:

```
Install-Module -Name powershell-yaml
```

Overview

This PowerShell script helps users deploy a containerized application to an Azure Kubernetes Service (AKS) cluster using an Azure Container Registry (ACR). The script uses a graphical user interface (GUI) to collect user inputs and automate deployment tasks.

Input Parameters

The script requires the following user inputs:

1. Resource Group: The Azure resource group where your resources are located.
2. Git Repository URL: The web address of the Git repository containing the application source code. If not provided, the default branch will be used.
3. Branch Name (Optional): The name of the branch in you Git Repository to use for the deployment.
4. Azure Container Registry (ACR) Name: The name of your ACR.
5. Azure Kubernetes Service (AKS) Name: The name of your AKS cluster.
6. Namespace (Optional): The Kubernetes namespace to which the application should be deployed. If not provided, the "default" namespace will be used.
7. Networking Capabilities: Optional selections to enhance your application.
    - Storage Account (optional): Provides scalable cloud storage.
    - Redis (optional): Offers an in-memory data structure store.

Execution Flow

The script follows these steps:

1. Validates the required modules are installed.
2. Loads WPF and creates a window using XAML.
3. Fetches existing resource groups, ACRs, and AKS clusters from the Azure account.
4. Clones the Git repository containing the application source code.
5. If provided, the script switches to the specified remote branch of the Git repository.
6. The script tests the Git repository URL and displays an error message if the URL is not valid.
7. Connects to the Azure account.
8. The script creates a Kubernetes namespace if it does not exist.
9. Builds and pushes the container image to the ACR.
10. Deploys the container image to the AKS cluster.
11. If selected, configures the application with the chosen networking capabilities (Storage Account, Redis, and/or Database).
12. Cleans up temporary files and displays a success message.

Required Azure Roles

The Azure account used with this script must have the following roles:

- Contributor role for managing resources (e.g., Resource Groups, ACRs, AKS clusters).
- Reader role for reading Azure resources' properties.
- AcrPush role for pushing images to the ACR.

Running the Script

To run the script:

1. Open the folder where the deployment script is located, right-click it and choose "Run with PowerShell". A graphical interface will appear.
2. Provide the necessary input parameters.
3. Provide the optional parameters if necessary: the branch name in your Git repository and the Kubernetes namespace for the deployment.
4. Click the Deploy button.

Troubleshooting and Tips

- Ensure the selected Azure account has sufficient permissions to perform all required operations. If you are unsure about your permissions, consult your Azure administrator.
- Check your internet connection to ensure you can access Azure services and the Git repository.
- If the Git repository URL is not valid or the specified branch does not exist, an error message will be displayed. Ensure the Git repository URL and branch name are correct.
- If errors occur, read the PowerShell console output for detailed error messages and further guidance. You can also consult online forums or contact support if necessary.
- If you encounter issues with the GUI, verify that the required assemblies are loaded correctly. If issues persist, consider reaching out to support or a knowledgeable colleague for assistance.

Access Container locally

kubectl port-forward is a command that forwards one or more local ports to a pod in a Kubernetes cluster. This allows you to access the pod's services as if they were running on your local machine. Here's how to use kubectl port-forward to forward a local port to a Redis instance running in a Kubernetes pod:

First, identify the name of the Redis pod. You can do this by running:

```
kubectl get pods -n <namespace>
```

Replace <namespace> with the namespace where your Redis instance is running. Once you have the Redis pod name, use the kubectl port-forward command to forward a local port to the Redis pod:

```
kubectl port-forward -n <namespace> <redis-pod-name> <local-port>:<redis-port>
```

Replace <namespace> with the namespace where the Redis instance is running, <redis-pod-name> with the name of the Redis pod, <local-port> with the local port number you want to use, and <redis-port> with the port number on which Redis is running (usually 6379).

For example, if your Redis pod's name is redis-container, is located within the "application" namespace, and you want to forward local port 6379 to the Redis pod's port 6379, you would run:

```
kubectl port-forward -n application rediscontainer 6379:6379
```

After running the kubectl port-forward command, you should see output in the terminal indicating that the port forwarding is active, like this:

```
Forwarding from 127.0.0.1:6379 -> 6379

Forwarding from [::1]:6379 -> 6379
```

Now you can access the redis instance on your local machine on 127.0.0.1:6379