System Integration™

GitHub Action

GitHub Action Pipeline Specifications:

Deploying ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Action

Version 1.0.0

BMV SYSTEM INTEGRATION PRIVATE LIMITED

Idea... Implementation... Innovation...

:: CORPORATE HEAD OFFICE::

A503, The First,
Behind The ITC Narmada Hotel & Keshavbaug Party Plot,
Off 132 ft Road, Vastrapur, Ahmedabad.
Gujarat- 380015
Phone: +91 (79) 40 30 53 02
Website: www.systemintegration.in

Mail: info@systemintegration.in

Revision History

Date	**Doc Version**	**Author**	**Status Created/Updated/Reviewed/Approved/Verified) In case of updates- Mention what has been updated. give change/issue/req id as ref.**
14-Feb-2024	1.0.0	Hardik Kalkani	Deploying ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Action

Table of Contents

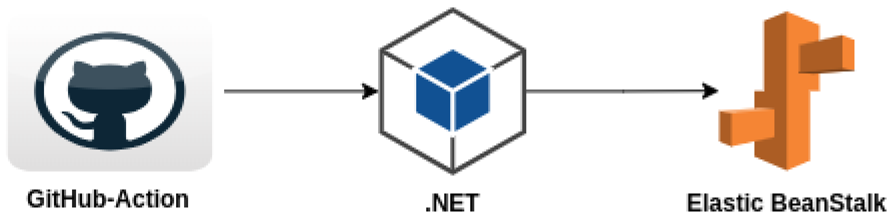
Deploying ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Action

Introduction

Deploying an ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Actions is a streamlined and efficient process that allows developers to automate the deployment of their applications with ease. AWS Elastic Beanstalk is a fully managed service that makes it simple to deploy, scale, and manage applications in the AWS Cloud, while GitHub Actions provides a powerful automation platform for continuous integration and continuous deployment (CI/CD).

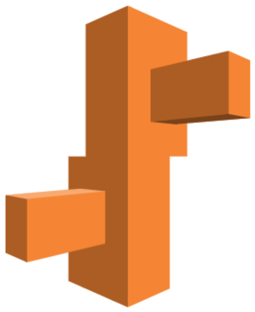
This document aims to guide developers through the step-by-step process of setting up a GitHub Action workflow to deploy an ASP.NET Core Web API to AWS Elastic Beanstalk. By leveraging the seamless integration between GitHub and AWS services, you can automate the deployment pipeline, ensuring that your application is deployed consistently and reliably.

Whether you are a seasoned developer looking to enhance your deployment process or a newcomer seeking a straightforward way to deploy your ASP.NET Core Web API to AWS Elastic Beanstalk, this guide will walk you through the essential steps. With the combined power of GitHub Actions and AWS Elastic Beanstalk, you can efficiently manage your application's lifecycle, from code changes to deployment, fostering a more agile and productive development environment.



Deploying ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Action

What's Elastic Beanstalk?



Amazon Elastic Beanstalk is a fully managed service provided by Amazon Web Services (AWS) that simplifies the deployment and management of applications in the cloud. It is designed to make the process of deploying and scaling web applications and services more accessible to developers by handling the underlying infrastructure complexities.

Key features of AWS Elastic Beanstalk include

Ease of Use: Elastic Beanstalk abstracts the complexities of infrastructure management, allowing developers to focus more on writing code and less on managing servers. It supports multiple programming languages, including Java, .NET, Python, Node.js, Ruby, Go, and more.

Automatic Scaling: Elastic Beanstalk enables automatic scaling of your application based on demand. It can automatically adjust the number of instances running your application, ensuring optimal performance during traffic spikes and cost efficiency during periods of lower demand.

Managed Environment: AWS Elastic Beanstalk provisions and manages the necessary AWS resources, such as Amazon EC2 instances, load balancers, and databases, on your behalf. This simplifies the setup process and eliminates the need for manual configuration.

Environment Customization: While abstracting the underlying infrastructure, Elastic Beanstalk also allows developers to customize their application environments. You can configure settings, such as instance types, database configurations, and more, to match the specific requirements of your application.

Integrated Developer Tools: Elastic Beanstalk seamlessly integrates with popular development tools and platforms. It supports version control systems like Git, CI/CD services like AWS CodePipeline, and provides integration with the AWS Management Console, AWS Command Line Interface (CLI), and SDKs.

Health Monitoring and Logging: Elastic Beanstalk provides built-in monitoring and logging capabilities. Developers can access performance metrics, logs, and other diagnostics to troubleshoot issues and optimize application performance.

Multi-Tier Architectures: Elastic Beanstalk supports multi-tier application architectures, allowing you to deploy web applications, worker environments, and other components in a coordinated and scalable manner.

By using AWS Elastic Beanstalk, developers can accelerate the deployment of their applications, reduce operational overhead, and ensure a more seamless and reliable experience for end-users. Whether you're deploying a simple web application or a complex, multi-tiered architecture, Elastic Beanstalk simplifies the process and allows you to focus on delivering value through your code.

What’s GitHub Action?

GitHub Actions is an integrated continuous integration and continuous deployment (CI/CD) service provided by GitHub. It allows developers to automate various aspects of their software development workflows directly within the GitHub repository. GitHub Actions can be used to build, test, and deploy code, making it easier to maintain and release high-quality software.

Key features of GitHub Actions include

Workflow Automation: GitHub Actions enables the creation of custom workflows, which are sets of automated steps defined in YAML files within the repository. Workflows can include tasks such as building code, running tests, and deploying applications.

Event-Driven Triggers: Workflows can be triggered by various events, such as pushes to the repository, pull requests, issue comments, or scheduled events. This event-driven model allows for flexibility in triggering workflows based on specific actions or changes in the repository.

Diverse Ecosystem: GitHub Actions supports a wide range of programming languages, platforms, and third-party integrations. Users can leverage pre-built actions from the GitHub Marketplace or create custom actions tailored to their specific needs.

Matrix Builds: GitHub Actions allows you to define matrix builds, which execute a workflow with different configurations. This is particularly useful for testing code across multiple versions of a programming language, operating systems, or other parameters.

Parallel and Sequential Jobs: Workflows can include parallel or sequential execution of jobs, enabling efficient use of resources. Parallel jobs run concurrently, while sequential jobs wait for the completion of the previous job before starting.

Secrets Management: GitHub Actions provides a secure way to manage and use secrets, such as API keys or access tokens, in workflows. Secrets are encrypted and can be used to authenticate with external services without exposing sensitive information.

Visualizations and Logs: GitHub Actions offers visualizations of workflow runs, allowing developers to quickly identify successful runs and troubleshoot any failures. Detailed logs provide insights into each step of the workflow, aiding in debugging and optimization.

Self-Hosted Runners: While GitHub provides hosted runners for running workflows, users also have the option to set up self-hosted runners on their own infrastructure. This allows for greater control over the execution environment.

GitHub Actions simplifies the process of setting up CI/CD pipelines by integrating directly with the version control system. It promotes automation and collaboration, streamlining development workflows and helping teams deliver software more efficiently. Whether you're building open-source projects or working on private repositories, GitHub Actions provides a versatile and powerful toolset for automating various aspects of the software development lifecycle.

System Integration™

Understanding CI/CD with AWS

In our exploration of AWS services, let's delve into deploying our ASP.NET Core Web API to AWS Elastic Beanstalk through Continuous Integration and Continuous Delivery (CI/CD) with GitHub Actions.

Beginning with the creation of an ASP.NET Core Web API in Visual Studio, we push the code to a GitHub Repository, initiating GitHub Actions. Concurrently, we verify the existence of a configured AWS Elastic Beanstalk Application and Environment, ready for Web API hosting. GitHub Actions takes charge of automating the deployment, seamlessly integrating AWS CLI commands to upload and deploy the application onto Elastic Beanstalk, creating a smooth transition from GitHub to the AWS Cloud.

The workflow begins with GitHub Actions, where we specify the following configurations:

Source: Our GitHub repository. AWS is granted permission to read from our GitHub repository.

Build: GitHub Actions takes care of building our application using the .NET 6 runtime. It pulls the source code from GitHub, builds it, and packages our Web API into executable DLLs (as specified in the **.github/workflows/deploy.yml** file we'll include).

Deploy: Using the AWS CLI within GitHub Actions, we upload the build's zip file to an S3 bucket. Subsequently, the AWS CLI commands facilitate the deployment of the application to the AWS Elastic Beanstalk Environment. Configuration settings for the Elastic Beanstalk deployment can be specified in AWS CLI commands or referenced files.

If everything proceeds smoothly, your application should already be deployed to the cloud, accessible via a publicly accessible endpoint—all accomplished seamlessly through the automation power of GitHub Actions. This approach simplifies the CI/CD process, streamlining the deployment of your ASP.NET Core Web API to AWS Elastic Beanstalk directly from your GitHub repository.

Prerequisites

AWS Account: Ensure you have an AWS Account, and the Free Tier is sufficient for this deployment.

Development Environment: Have Visual Studio or your preferred Integrated Development Environment (IDE) installed on your machine.

.NET 6.0+ SDK: Install the .NET 6.0 or a later version Software Development Kit (SDK) on your machine for ASP.NET Core development.

GitHub Account: You'll need an active GitHub account for version control and repository management.

With these prerequisites in place, you're ready to deploy your ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Actions.

Creating an AWS Elastic Beanstalk Environment & Application:

Log in to AWS:

Log into your AWS account and navigate to AWS Elastic Beanstalk.

Create Application:

Click on "Create Application." Provide a meaningful name for your web application.

Platform and Environment Configuration:

Select the following platforms and branches:

- Platform: .NET Core on Linux
- Platform Branch: .NET Core running on 64bit Amazon Linux 2023
- Platform Version: 3.0.3 Latest

Application Code: Optionally, you can provide a sample code or your code for your application.

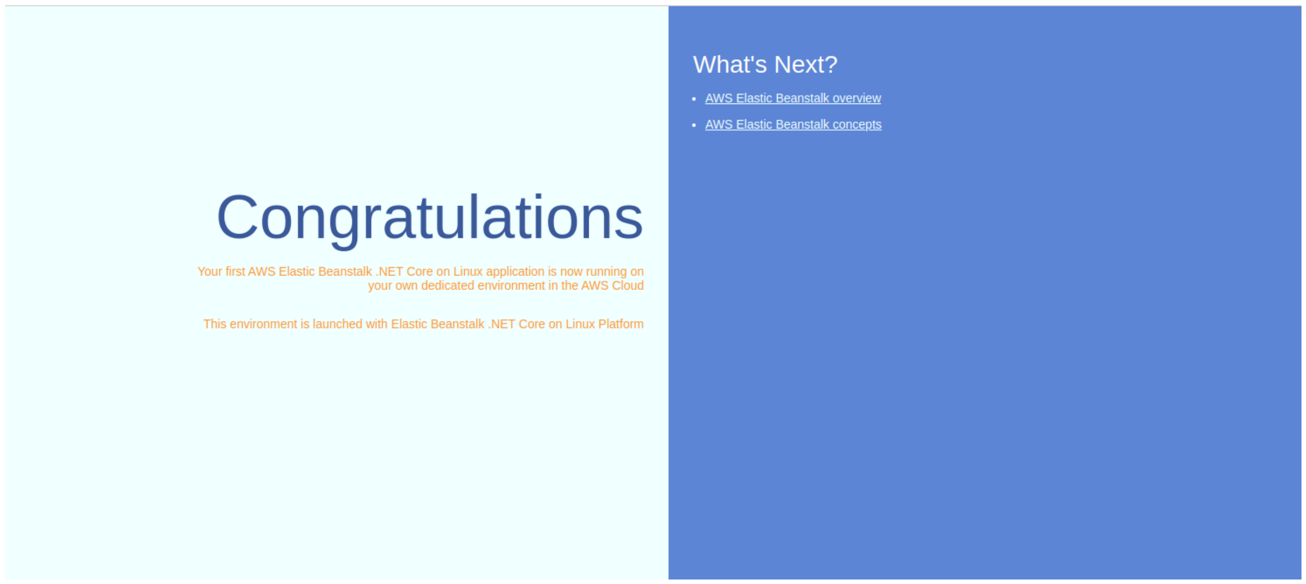
Provision Resources:

After configuring, initiate the creation process. This will take approximately 5 minutes to provision necessary resources, including EC2 instances, load balancers, auto scalers, security groups, and other essentials.

Access URL:

Once provisioning is complete, you'll receive a URL. This is where your ASP.NET Core Web API will be accessible. Opening this URL at this stage will showcase a sample application running on the server. It will look like the below reference image.

With the AWS Elastic Beanstalk environment and application set up, you are now ready to proceed with deploying your ASP.NET Core Web API to this environment using GitHub Actions.



Congratulations

Your first AWS Elastic Beanstalk .NET Core on Linux application is now running on your own dedicated environment in the AWS Cloud

This environment is launched with Elastic Beanstalk .NET Core on Linux Platform

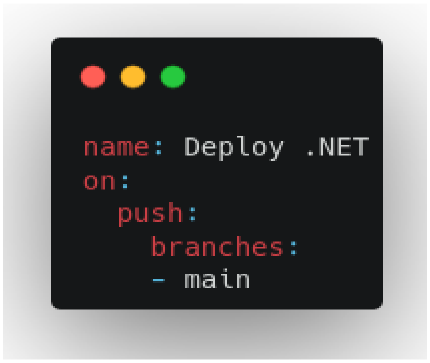
What's Next?

- AWS Elastic Beanstalk overview
- AWS Elastic Beanstalk concepts

Creating a GitHub Action YAML file - step-by-step

Let's break down each block in the provided YAML file:

1)

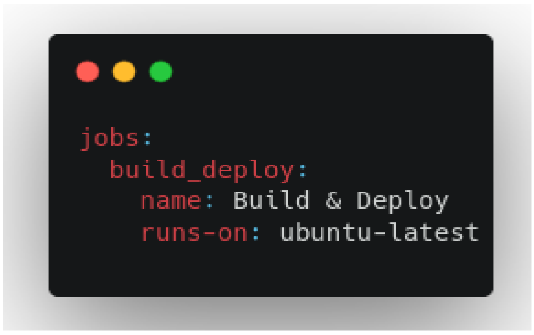


```yaml
name: Deploy .NET
on:
  push:
    branches:
    - main
```

Description: Specifies the name of the GitHub Action. In this case, it's set to "Deploy .NET." and when the GitHub Action should be triggered.

Trigger: It triggers a push event to the "main" branch.

2)



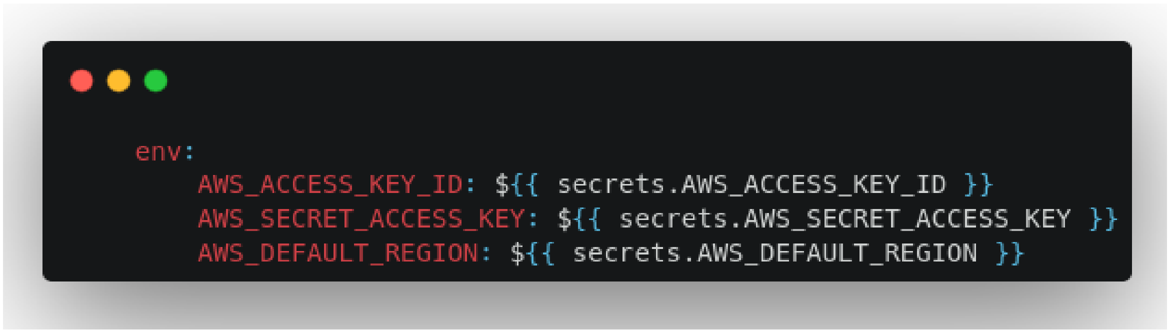
```yaml
jobs:
  build_deploy:
    name: Build & Deploy
    runs-on: ubuntu-latest
```

Description: Begins the definition of a job named "build_deploy" that runs on an Ubuntu environment.

Job Name: "Build & Deploy."

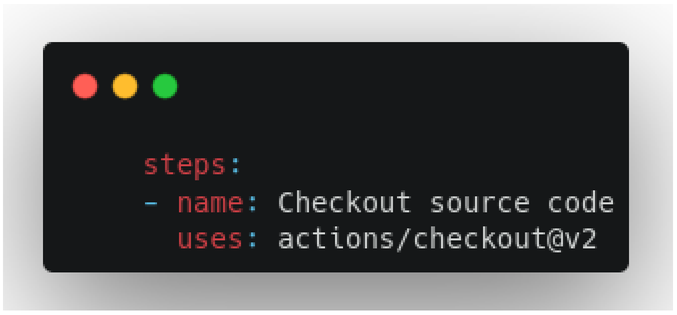
Environment: Uses the latest version of Ubuntu as the execution environment.

3)


```
env:
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    AWS_DEFAULT_REGION: ${{ secrets.AWS_DEFAULT_REGION }}
```

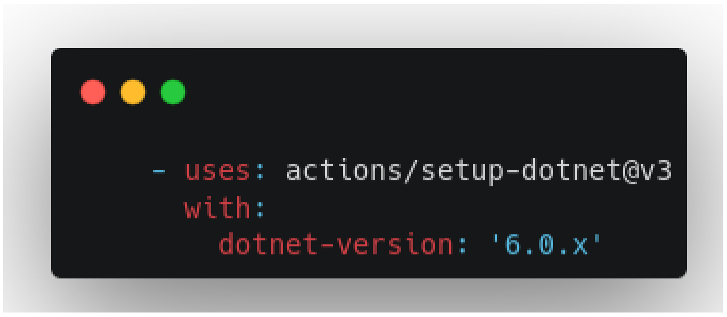
Description: Sets environment variables for AWS credentials and region using GitHub Secrets.

4)


```
steps:
- name: Checkout source code
  uses: actions/checkout@v2
```

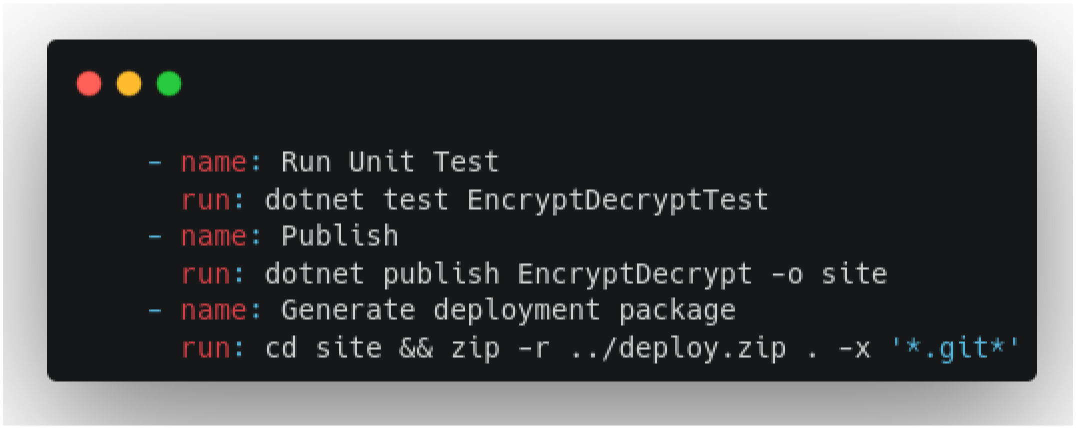
Description: Check out the source code of the repository.

5)


```
- uses: actions/setup-dotnet@v3
  with:
    dotnet-version: '6.0.x'
```

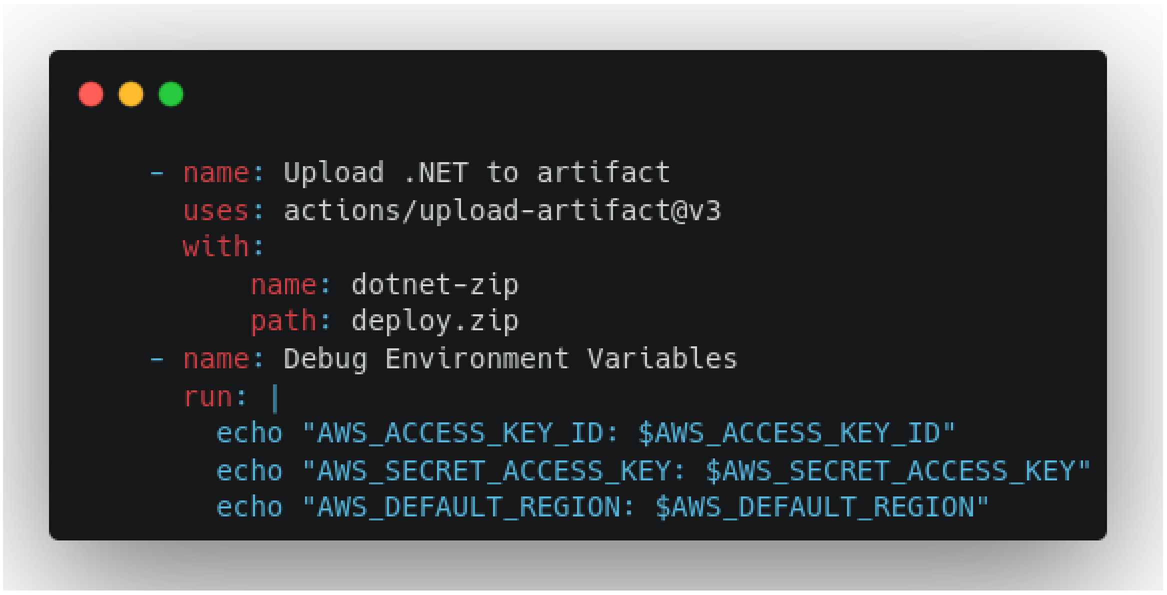
Description: Set up the .NET environment, specifying version '6.0.x'.

6)


```
- name: Run Unit Test
  run: dotnet test EncryptDecryptTest
- name: Publish
  run: dotnet publish EncryptDecrypt -o site
- name: Generate deployment package
  run: cd site && zip -r ../deploy.zip . -x '*.git*'
```

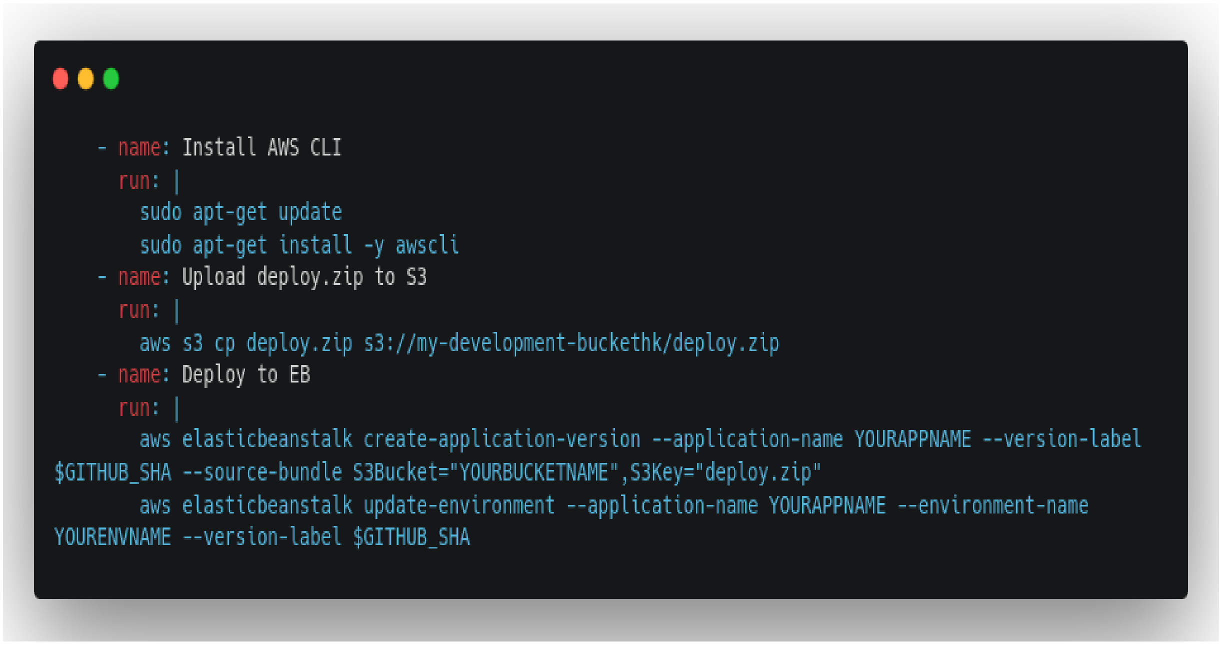
- **Description:** Executes unit tests for the specified project.
- **Description:** Publishes the project, generating output in the "site" directory.
- **Description:** Creates a deployment package by compressing the contents of the "site" directory, excluding git-related files.

7)


```
- name: Upload .NET to artifact
  uses: actions/upload-artifact@v3
  with:
      name: dotnet-zip
      path: deploy.zip
- name: Debug Environment Variables
  run: |
    echo "AWS_ACCESS_KEY_ID: $AWS_ACCESS_KEY_ID"
    echo "AWS_SECRET_ACCESS_KEY: $AWS_SECRET_ACCESS_KEY"
    echo "AWS_DEFAULT_REGION: $AWS_DEFAULT_REGION"
```

- **Description:** Uploads the deployment package as a GitHub artifact named "dotnet-zip."
- **Description:** Outputs AWS environment variables for debugging purposes.

8)



```
    - name: Install AWS CLI
      run: |
        sudo apt-get update
        sudo apt-get install -y awscli
    - name: Upload deploy.zip to S3
      run: |
        aws s3 cp deploy.zip s3://my-development-buckethk/deploy.zip
    - name: Deploy to EB
      run: |
        aws elasticbeanstalk create-application-version --application-name YOURAPPNAME --version-label
$GITHUB_SHA --source-bundle S3Bucket="YOURBUCKETNAME",S3Key="deploy.zip"
        aws elasticbeanstalk update-environment --application-name YOURAPPNAME --environment-name
YOURENVNAME --version-label $GITHUB_SHA
```

- **Description:** Installs the AWS CLI on the Ubuntu environment.
- **Description:** Copies the deployment package to an S3 bucket.
- **Description:** Deploys the application to AWS Elastic Beanstalk. Creates a new application version and updates the environment with the latest version.

This GitHub Action workflow automates the build, testing, packaging, and deployment of your .NET application to AWS Elastic Beanstalk, integrating with AWS services and utilising GitHub Secrets for secure credential management.

Setting up secrets and variables in the GitHub repository

Setting up secrets and variables in a GitHub repository involves utilizing GitHub's repository settings to manage sensitive information securely. Here's a step-by-step guide:

- **Navigate to Your Repository:**
  - Open your GitHub repository in a web browser.
- **Access Repository Settings:**
  - Click on the "Settings" tab at the top of your repository.

- **Navigate to Secrets:**
  - In the left sidebar, find and click on "Secrets."
- **Add New Secret:**
  - Click on the "New repository secret" button.
- **Define Secret Name and Value:**
  - Provide a descriptive name for your secret (e.g., AWS_ACCESS_KEY_ID).
  - Enter the corresponding secret value.
- **Repeat for Additional Secrets:**
  - If you have multiple secrets (e.g., AWS_SECRET_ACCESS_KEY, AWS_DEFAULT_REGION), repeat the process to add them.
- **Save and Confirm:**
  - Save your secrets and environment variables.

Now, you've securely stored sensitive information like AWS credentials and environment-specific variables in your GitHub repository. These secrets and variables can be referenced in your GitHub Actions workflows, allowing for secure access without exposing sensitive information in your code. Always ensure to follow best practices for securing and managing secrets in your development workflow.

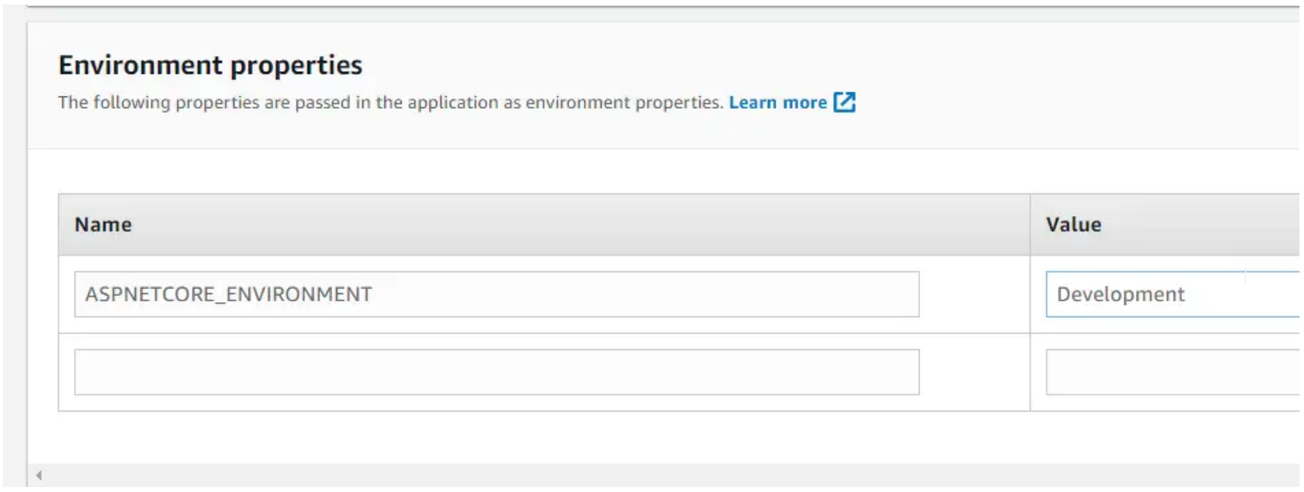
Setting Environment Variable in AWS Elastic Beanstalk

Setting environment variables in AWS Elastic Beanstalk can be done through the AWS Management Console or the AWS Command Line Interface (CLI). Here's how you can do it through the console:

The requirement is to set the **ASPNETCORE_ENVIRONMENT** variable for the AWS Elastic Beanstalk Environment as **Development**. This will give us access to **Swagger**.

- **Navigate to Elastic Beanstalk:**
  - Open the AWS Management Console.
  - Navigate to the Elastic Beanstalk service.
- **Select Your Environment:**
  - In the Elastic Beanstalk console, choose your application, and then select the environment for which you want to set environment variables.
- **Go to Configuration:**

- ◦ In the left sidebar, click on "Configuration" under the appropriate environment.
- **Edit Updates, monitoring, and logging:**
  - Scroll down to the updates, monitoring, and logging
  - Find the "Environment properties" card and click on the "Edit" button.
- **Add Environment Variables:**
  - In the "Edit environment properties" dialog, you can add new environment variables.
  - For each variable, provide a name and a value.
- **Save Changes:**
  - After adding the desired environment variables, click the "Apply" button to save the changes.
- **Restart the Environment (If Necessary):**
  - Depending on your application, you may need to restart your environment for the changes to take effect. You can do this from the Elastic Beanstalk console.



Environment properties

The following properties are passed in the application as environment properties. Learn more

Name	Value
ASPNETCORE_ENVIRONMENT	Development

Summary

In summary, the process of deploying an ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Actions involves several key steps:

- **Prerequisites:**
    - Ensure you have an AWS account (Free Tier is sufficient), a development environment (e.g., Visual Studio), .NET 6.0+ SDK installed, and a GitHub account.
- **AWS Elastic Beanstalk Setup:**
    - Log into your AWS account and navigate to AWS Elastic Beanstalk.
    - Create a new application, providing a name for your web application.
    - Choose the appropriate platform and configuration, such as .NET Core on the Linux platform.
    - Wait for the provisioning process to complete, obtaining a URL for accessing your ASP.NET Core Web API.
- **GitHub Action Workflow:**
    - Create a GitHub Action workflow by defining a YAML file in your repository.
    - Configure the workflow to trigger on pushes to the main branch.
    - Specify jobs for building, testing, packaging, and deploying the application.
    - Utilize GitHub Secrets to securely manage sensitive information such as AWS credentials and region.
    - Leverage GitHub Actions to automate the entire CI/CD pipeline, from code changes to deployment.
- **Deployment Steps in GitHub Action Workflow:**
    - Check out the source code from the repository.
    - Set up the .NET environment using the specified version.
    - Run unit tests, publish the application, and generate a deployment package.
    - Upload the deployment package as a GitHub artifact.
    - Output environment variables for debugging purposes.
    - Install the AWS CLI in the GitHub Actions environment.
    - Upload the deployment package to an S3 bucket.
    - Deploy the application to AWS Elastic Beanstalk by creating a new application version and updating the environment.
- **Environment Variable Configuration in AWS Elastic Beanstalk:**
    - Set environment variables for your ASP.NET Core Web API in the AWS Elastic Beanstalk console.

- Navigate to the Elastic Beanstalk environment, go to "Configuration," and edit the software configuration to add environment variables.
- Save the changes and, if necessary, restart the environment.

By following these steps, you can automate the deployment of your ASP.NET Core Web API to AWS Elastic Beanstalk using GitHub Actions, ensuring a streamlined and efficient CI/CD process.

References

https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-dotnet-core-linux.html

https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/platforms-linux.html

https://github.com/SystemIntegration