**GitHub Actions CI/CD Pipelines for AWS ECS Deployment**

Borys Posternakov1, Master’s degree student

ORCID: 0009-0006-8578-8299

**Scientific supervisor:** Yuliia Kuznetsova2, Ph.D., Associate Professor

ORCID: 0000-0001-9416-6981

1,2National Aerospace University – “Kharkiv Aviation Institute” Ukraine, Kharkiv

**Introduction**

Continuous Integration and Continuous Deployment (CI/CD) pipelines have become an essential part of modern software development workflows. They enable teams to automate the build, test, and deployment processes, leading to faster and more reliable software releases. GitHub Actions, coupled with Amazon Web Services (AWS) Elastic Container Service (ECS), provides a powerful combination for implementing CI/CD pipelines for deploying containerized applications. In this article, we will explore how to set up and configure GitHub Actions CI/CD pipelines for AWS ECS deployment.

1. **Understanding AWS Elastic Container Service (ECS)**

AWS Elastic Container Service (ECS) [2] is a scalable container orchestration service that simplifies the deployment and management of Docker containers [1]. ECS allows you to run containers without the need to manage the underlying infrastructure. It provides features like load balancing, auto scaling, and service discovery, making it an ideal choice for deploying containerized applications. Containerized software is a compiled source code binary, which runs in container [1] environment like Docker, ContainerD. Docker is a set of platforms as a service product that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc.

**Key features:**

* *Containerized Application Deployment* [3]: ECS allows you to deploy and run applications as Docker containers. You can package your application and its dependencies into container images, which can be easily deployed to ECS clusters.
* *Scalability and High Availability:* ECS provides built-in scalability and high availability features. You can scale your applications horizontally by adding or removing containers based on demand. ECS manages the distribution of containers across Availability Zones to ensure high availability and fault tolerance.
* *Task Definitions:* A task definition in ECS defines how your containers should be run, including the Docker image to use, resource requirements, environment variables, networking, and container dependencies. Task definitions provide a blueprint for launching containers as tasks within ECS.
* *Service Discovery and Load Balancing:* ECS integrates with AWS Cloud Map for service discovery, allowing containers to discover and communicate with each other using DNS. Additionally, ECS integrates with Elastic Load Balancing (ELB) to distribute incoming traffic across containers in a service, providing load balancing capabilities.
* *Auto Scaling:* ECS supports automatic scaling of services based on metrics such as CPU utilization, memory utilization, or custom metrics. You can define scaling policies to automatically add or remove containers based on the specified thresholds, ensuring optimal resource utilization.
* *Integration with AWS ECR:* ECS seamlessly integrates with Amazon Elastic Container Registry (ECR), a fully managed Docker container registry. You can push and pull container images from ECR, allowing for secure storage and distribution of your container images within the AWS ecosystem.

In summary, AWS ECS [2] provides a comprehensive and scalable platform for deploying and managing containerized applications. Its integration with other AWS services, built-in scalability, and support for infrastructure as code make it an ideal choice for organizations looking to leverage the power of containers in their cloud-based workflows.

1. **Understanding of the Deployments for Containerized Application**

Containerization has revolutionized application deployment by providing a lightweight and scalable approach. Amazon Web Services (AWS) Elastic Container Service (ECS) [2] offers a powerful solution for deploying and managing containerized applications in the cloud. Also, this one can be integrated with the AWS ECR (Elastic Container Registry). The AWS ECR provides an ability to store a container images. Otherwise, this one provides a container repository.

1. **Overview of the GitHub Actions**

Continuous Integration and Continuous Deployment (CI/CD) [8] pipelines have become essential in modern software development workflows. GitHub Actions provides a powerful platform for automating the build, test, and deployment processes directly from your GitHub repository. In this article, we will explore the components and steps involved in setting up a GitHub Actions CI/CD pipeline. GitHub Actions is a powerful workflow automation and CI/CD [10] platform provided by GitHub. It allows developers to define custom workflows, triggered by events such as code pushes, pull requests, or scheduled intervals. With GitHub Actions, we can build, test, and deploy applications directly from your GitHub source code repository.

**GitHub Actions enables automated CI/CD pipelines:**

* *Workflow Definition:* Define automated workflows in YAML files.
* *Triggers and Events:* Specify triggers like code pushes or pull requests.
* *Jobs and Steps:* Define tasks for building, testing, and deployment.
* *Building and Testing:* Compile code and run tests.
* *Environment Configuration:* Set variables and manage secrets securely.
* *Deployment Actions:* Deploy to servers, cloud platforms, or hosting providers.
* *Post-Deployment Steps:* Perform tasks after deployment (for example, testing of the deployed service in container).
* *Monitoring and Notifications:* Monitor pipeline execution and get alerts (Success, Fail, or Warning).
1. **Setting Up GitHub Actions Workflows**

To get started with GitHub Actions, you need to define a workflow file in your repository. This file describes the steps and actions to be performed during the CI/CD process. Let's outline the main steps involved in setting up a GitHub Actions workflow for AWS ECS deployment:

* 1. *Defining Workflow Triggers*

First, needs to configure GitHub Actions workflows to trigger on specific events. For example, you might want to trigger the workflow whenever a new commit is pushed to the repository or when a pull request is opened. By specifying the appropriate triggers, you can ensure that your CI/CD pipeline [8] runs automatically whenever there is a relevant event.

* 1. *Building and Testing the Application*

Before deploying the application, it's crucial to build and test it to ensure its quality and stability. In this step, you can define the necessary actions to build and test your containerized application. You might use tools like Docker to build the container image and testing frameworks to execute automated tests.

* 1. *Preparing ECS Deployment*

In this article, we will explore the steps involved in deploying containerized applications with AWS ECS [2].

To deploy the application to AWS ECS [2], you need to prepare the necessary infrastructure and configurations. This step involves setting up an ECS cluster [2], creating a task definition, and configuring any required AWS resources like load balancers, VPC, security groups. These configurations can be defined as code using AWS CloudFormation, AWS CDK, or HashiCorp Terraform. Also needs define in GitHub Actions a credentials for the AWS Account and create an AWS ECR repository for storing Image builds [1, 3 – 5].

* 1. *. Deploying a Containerized Application to the AWS ECS*

Once the infrastructure is ready, you can deploy the application to AWS ECS [2]. GitHub Actions provides built-in actions, or you can create custom actions to interact with the AWS APIs and perform the deployment. The deployment process typically involves creating or updating ECS services, registering task definitions, and handling any necessary environment variables or secrets (Fig. 1).



Fig. 1. ECR Deployment with GitHub repository and GitHub Actions Workflow

* 1. *. Handling Secrets and Environment Variables*

When working with CI/CD pipelines [8], it's essential to handle sensitive information like access keys, passwords, or API tokens securely. GitHub Actions [7] provides a Secrets feature that allows you to store and retrieve encrypted secrets during the pipeline execution. You can store AWS credentials, or any other secrets required for the deployment and access them securely within your workflow.

* 1. *Monitoring and Error Handling*

Monitoring the CI/CD [8] pipeline and handling errors are crucial aspects of maintaining a robust deployment process. GitHub Actions provides various ways to monitor and visualize the pipeline's execution, including logs and status checks. Additionally, you can configure notifications or alerts to be triggered in case of failures or issues during the deployment. Also, if the deployment to the AWS ECS was failed, the GitHub Actions pipeline can roll-back to the previous version.

**Conclusions**

In conclusion, leveraging GitHub Actions for AWS ECS deployment brings automation and efficiency to the CI/CD pipeline. By combining the power of GitHub Actions with the flexibility of AWS ECS, developers can streamline the process of building, testing, and deploying containerized applications. With GitHub Actions, you can define workflows that trigger automatically based on specific events, such as code pushes or pull requests. These workflows consist of jobs and steps that execute tasks like building the application, running tests, and deploying to AWS ECS.

The integration between GitHub Actions and AWS ECS [2] simplifies the deployment process. You can leverage AWS ECS [2] features like task definitions, clusters, services, and load balancing to ensure scalability, high availability, and efficient resource utilization. Additionally, GitHub Actions provides monitoring and notification capabilities, allowing you to track the execution of the CI/CD pipeline and receive alerts in case of failures or issues. This helps maintain the health and reliability of your deployments. By adopting GitHub Actions CI/CD [8] pipelines for AWS ECS deployment, development teams can automate software delivery, reduce manual efforts, and achieve faster release cycles.

The combination of these two powerful tools enables seamless integration and efficient management of containerized applications on the AWS cloud platform. In conclusion, GitHub Actions CI/CD pipelines for AWS ECS deployment offer a robust and streamlined approach to containerized application delivery, empowering developers to deliver high-quality software with speed and efficiency [4 – 10].

In summary, GitHub Actions and AWS ECS together provide a powerful solution for automating the deployment of containerized applications on the AWS cloud platform [1-10]. Developers can enhance their software delivery process and efficiently manage their containerized applications by integrating GitHub Actions flexible workflows with AWS ECS's robust infrastructure.

**References:**

1. What is Containerization [Електронний ресурс]. – Режим доступу:
<https://aws.amazon.com/what-is/containerization/?nc1=h_ls>
2. What is containerization? Explore the history of containerization technology, the benefits and advantages of utilizing the technology, and how it relates to virtualization [Електронний ресурс]. – Режим доступу: <https://www.ibm.com/topics/containerization>
3. Amazon Elastic Container Service FAQs [Електронний ресурс]. – Режим доступу: <https://aws.amazon.com/ecs/faqs/>
4. Deploying a containerized web application [Електронний ресурс]. – Режим доступу: <https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app>
5. Deploy Docker Containers on Amazon ECS [Електронний ресурс]. – Режим доступу: <https://aws.amazon.com/getting-started/hands-on/deploy-docker-containers/>
6. Deploying to Amazon Elastic Container Service [Електронний ресурс]. – Режим доступу: [https://docs.github.com/en/actions/deployment/
deploying-to-your-cloud-provider/deploying-to-amazon-elastic-container-service](https://docs.github.com/en/actions/deployment/deploying-to-your-cloud-provider/deploying-to-amazon-elastic-container-service)
7. Amazon ECS “Deploy Task Definition” Action for GitHub Actions [Електронний ресурс]. – Режим доступу: <https://github.com/marketplace/actions/amazon-ecs-deploy-task-definition-action-for-github-actions>
8. Create a CI/CD pipeline for Amazon ECS with GitHub Actions and AWS CodeBuild Tests [Електронний ресурс]. – Режим доступу: <https://aws.amazon.com/blogs/containers/create-a-ci-cd-pipeline-for-amazon-ecs-with-github-actions-and-aws-codebuild-tests/>
9. What is CI/CD? [Електронний ресурс]. – Режим доступу: <https://www.redhat.com/en/topics/devops/what-is-ci-cd>
10. Continuous integration vs. delivery vs. deployment [Електронний ресурс]. – Режим доступу: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>