

# Webhook Driven Cross Platform Docker Image Transfer: Achieving AWS-Azure Interoperability

MSc Research Project  
MSc Cloud Computing

Nikita Patel  
Student ID: 21224811

School of Computing  
National College of Ireland

Supervisor: Sean Heeney

**National College of Ireland MSc  
Project Submission Sheet**  
**School  
of Computing**



**Student Name:** Nikita Chhotelal Patel  
**Student ID:** 21224811  
**Programme:** MSc Cloud Computing **Year:** 2022-2023  
**Module:** MSc Research Project  
**Supervisor:** Sean Heeney  
**Submission Due Date:** 18-09-2023  
**Project Title:** Webhook Driven Cross Platform Docker Image Transfer:  
Achieving AWS-Azure Interoperability  
**Word Count:** 4273 **Page Count:** 17

I hereby certify that the information contained in this (my submission) is information about to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use another author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Nikita Patel

**Date:** 17-09-2023

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

|   |                          |
|---|--------------------------|
| Attach a completed copy of this sheet to each project (including multiple copies)   | <input type="checkbox"/> |
| <b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).  | <input type="checkbox"/> |
| <b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | <input type="checkbox"/> |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

|                                  |  |
|----------------------------------|--|
| <b>Office Use Only</b>           |  |
| Signature:                       |  |
| Date:                            |  |
| Penalty Applied (if applicable): |  |

# Webhook Driven Cross Platform Docker Image Transfer: Achieving AWS-Azure Interoperability

Nikita Chhotelal Patel  
21224811

## Abstract

Conventional ways of transferring Docker images between clouds may often result in inefficiency and complexity. To solve this, a Webhook-driven approach for fast and trustworthy docker image transfer was built by providing an integrated gateway between Amazon Web Services (AWS) and Microsoft Azure. This study was motivated by the need to overcome obstacles to ensuring seamless interoperability across these major cloud platforms. This demonstrates the utility of Webhook-powered cross-platform interoperability solutions. Implementation revealed significant advantages, with Webhooks increasing transfer speed and reliability, showing their potential to promote cross-platform interoperability. This study makes an important contribution to the continuing conversation about cloud integration by suggesting a feasible route towards the efficient and standardised deployment of cloud applications.

**Keywords:** *Webhook-driven, Docker image transfer, Cloud interoperability, Docker Image, Cross-platform integration*

## 1 Introduction

The necessity of smooth interoperability across heterogeneous platforms cannot be emphasized in today's dynamic cloud world, where organizations use the capabilities of

various cloud providers to optimize their operations by (Chituc, Azevedo and Toscano, 2009). Cloud interoperability is dependent on practise standardization, which allows enterprises to go beyond the constraints of distinct cloud ecosystems. This article will take you on a trip to demonstrate the critical importance of standardize container deployment in ensuring interoperability between two industries :Amazon Web Services (AWS) and Microsoft Azure (Copik *et al.*, 2021) . Because it is more well-known than other public cloud service providers, Microsoft Azure has been selected as the research's cloud computing platform. Many features and advantages encourage enterprises to embrace Azure Cloud. Azure has a bigger market reach and might make cloud services and technology more accessible to more individuals. Azure is without a doubt the winner in terms of operating standards, guiding principles, and even best practices. Comparing Azure technology against those of other cloud providers might appeal to a larger audience.

Cloud interoperability, a significant concept in modern computing, refers to the seamless interaction and exchange of data and services across multiple cloud platforms. As organizations rely more on hybrid and multi-cloud systems, it is critical to ensure good communication across diverse cloud services. The goal of cloud interoperability is to create a unified framework for optimal resource utilisation, application deployment, and data interchange by resolving challenges caused by disparate standards, protocols, and

infrastructures. Cloud interoperability allows organizations to scale more efficiently, improve scalability, and make use of the characteristics of multiple cloud providers by enabling the seamless operation of disparate cloud systems (Zhang, Wu and Cheung, 2013). This fundamental understanding lays the platform for delving into the complexity and significance of cloud interoperability in today's digital landscape.

Webhooks are an important tool for facilitating real-time communication between several online programs. By functioning as a push notification method, webhooks allow the automatic transfer of data across systems whenever specified events occur. Webhooks eliminate unnecessary inquiries, increasing efficiency and responsiveness over traditional polling approaches, which need programs to constantly check for changes. When an event happens in one program, a predefined URL, or "webhook endpoint," is enabled, causing the transmission of relevant data to another application. This rapid and perfect data transmission supports dynamic integration, enabling programmes to properly synchronise and collaborate. Webhooks provide process simplification, workflow automation, and improved interactions across several digital platforms.

Smooth interoperability across multiple platforms has become a fundamental aim in the cloud computing ecosystem. The research goes by (Zhukov, 2021) into the world of cloud interoperability concentrate on the integration of two significant cloud systems, Amazon Web Services (AWS) and Microsoft Azure. The project offers an innovative approach, focusing on webhooks as a driving force to enable rapid cross-platform Docker image transfer between AWS and Azure. The project proposes to use webhooks to provide a streamlined approach for transporting Docker images seamlessly between various cloud giants by (Abdelbaky *et al.*, 2015) which permit real-time communication between systems. Webhook integration aims to bridge the gap between AWS and Azure by addressing cross-platform data migration problems. The (Liu and Zhao, 2014) research seeks to shed light on the potential of webhooks to improve interoperability in the context of cloud-based Docker image deployment.

## 2 Related Work

(Loutas *et al.*, 2011) announced cloud computing interoperability standardization projects, which include collaboration across various organizations and an attempt to build a common framework with standardized APIs and data formats. Adoption of these standards aims to eliminate vendor lock-in while improving application and data mobility. The author emphasises the need for researchers to reach an agreement on common principles for interoperability solutions. These results might be used in academic projects looking into cloud computing interoperability and portability.

(Dillon, Wu and Chang, 2010) mentioned that API standardization is important for making the Cloud easier to use and move around in. Creating open standards for Cloud APIs that are widely used by developers, using middleware solutions that allow seamless integration between different Cloud APIs, standardizing interfaces between different layers

of the cloud stack, and using containerization technologies to reduce differences in the underlying

infrastructure, and addressing security concerns related to Cloud computing adoption are all ways to reach this goal.

The absence of standardization in cloud computing creates heterogeneities, making interoperability, cooperation, and service portability a difficult process. (Stravoskoufos *et al.*, 2014) the purpose of this literature review is to identify and explore several ways to address the challenge of interoperability and portability in cloud computing services. The study assesses and contrasts various methodologies, highlighting potential research directions in this field.

(Harsh *et al.*, 2012) has investigated the challenges of providing true service interoperability and portability of end users' cloud applications. Among the challenges include a lack of standardization, vendor lock-in, security and privacy concerns, the complexity of distributed systems, and a lack of migration aid. The article discusses ongoing standardization efforts as well as the Contrail project, which aims to improve cloud application portability and compatibility with other cloud services and management tools. The author addresses the need to overcome these challenges to ensure the continuous adoption and progress of cloud computing.

The relevance of open standards in facilitating interoperability between cloud providers and private/public clouds is discussed by (Lewis, 2013) . It examines areas of cloud computing where standards may be advantageous for interoperability, as well as areas where standards would be useless or would need to expand in order to provide value, especially in the context of e-Government. The report also gives recommendations for cloud computing adoption independent of cloud standard development. Overall, this essay is a great resource for understanding the importance of standards in cloud computing interoperability.

The absence of a common data format, the range of cloud service description languages, and the heterogeneity of proposed solutions all pose challenges to cloud computing interoperability (CCI) by (Ayachi, Nacer and Slimani, 2022). To solve these problems, future research can focus on making a federated approach based on a generic description model of cloud services, finding the best approaches and strategies, making techniques that don't depend on a platform, and figuring out how well RESTf APIs work for standardization. These solutions try to make it possible for different cloud services to work together and give a standard way to talk about cloud services.

The Docker Image Vulnerability Diagnostic System (DIVDS) is offered as a technique for detecting the vulnerability level of each Docker image using a built-in vulnerability assessment approach. DIVDS prevents users from downloading or uploading potentially dangerous Docker images to a Docker image repository, resulting in a trustworthy Docker-based application development environment. The proposed system is investigated and disputed in the paper, which is released under a Creative Commons Attribution 4.0 Licence. (Kwon and Lee, 2020) is a Senior Member with research interests in protocol engineering and performance analysis.

(Parák and Šustr, 2014) The rOCCI Framework is proposed as a solution to the issues in achieving interoperability in IaaS cloud computing. The authors emphasise the need of

standardization efforts in areas such as virtual machine management, user-defined monitoring, and accounting. They provide practical guidance for achieving standard compliance and emphasise the need of open, standard interfaces for interoperability. This resource might be useful for academic studies on cloud computing and interoperability.

(Lynn *et al.*, 2017) a full technical comparison of seven main serverless computing platforms, including AWS Lambda, is provided, along with an overview and analysis of their features and capabilities. The authors also identify research needs and potential use applications for serverless computing in the enterprise. While serverless computing is still in its early stages, the study thinks it has the potential to be more cost-effective, user-friendly, and safe than traditional cloud computing models.

This article discussed how to use Docker Swarm to build a virtual system of systems for distributed software development across many clouds. The benefits of container-based software development are emphasized, as well as the merits and limitations of multi-cloud architecture. Because Kubernetes is an alternative to Docker Swarm, it may be built using either Docker Swarm or Kubernetes, which are both supported by both clouds. (Naik, 2016) concludes that a true Docker Swarm-based system of systems distributed over different clouds would be an exciting field for future research.

### **3 Research Methodology**

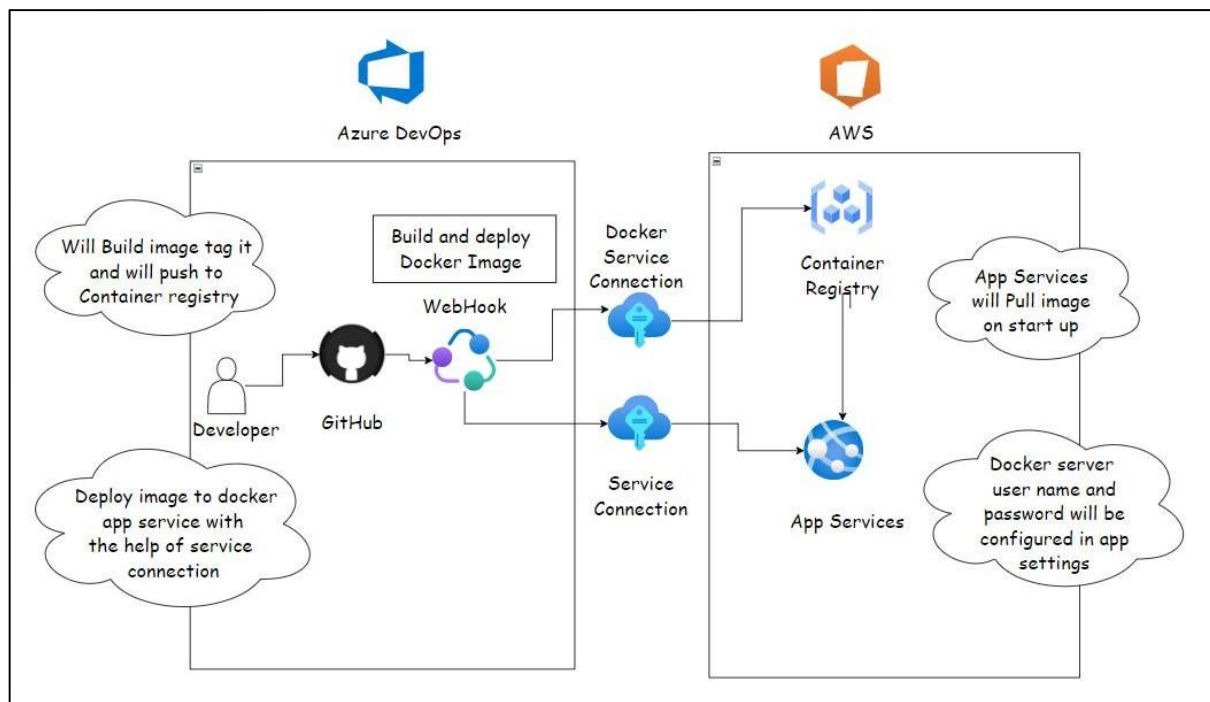
A thorough picture of the cloud computing interoperability environment is provided by the researched collection of research publications, which also include several major shortcomings. Although these studies provide helpful theoretical frameworks and answers, a recurring drawback is the absence of thorough practical implementations to confirm their efficacy in actual circumstances. Moreover, certain articles show a limited focus on particular aspects of interoperability, such as API standardization or virtual machine administration (Petcu *et al.*, 2011), this can overlook the more significant difficulties that cut across many cloud ecosystem tiers. An integrated strategy that takes into account the interconnection of cloud services, infrastructure, and administration is required to enable smooth interoperability is essential (Singh *et al.*, 2019). The absence of complete evaluation against a variety of use cases and scenarios could lead to incomplete insights into the strengths and weaknesses of various strategies.

There are many compelling reasons why a webhook-driven Docker image transfer between AWS and Azure is required to ensure interoperability. Traditional data transfer techniques usually entail human interventions or planned transfers, which may cause data synchronization delays and discrepancies. Webhooks offer a solution by allowing for real-time data transfer, guaranteeing that data is provided as soon as important events occur. Webhooks, in addition to their real-time capabilities, simplify and automate the data transmission process. Allowing the sharing of Docker images between AWS and Azure reduces human work greatly, lowering the likelihood of mistakes. This efficiency is particularly important in today's cloud systems, where the interchange of accurate and timely data is critical.

The dynamic scalability of webhooks well matches the features of cloud systems. Webhooks may adapt to changing responsibilities and resource needs, ensuring that data transfers stay efficient and responsive. This scalability is particularly beneficial for cloud-hosted applications with fluctuating resource requirements (Lampesberger, 2016). Webhooks' event-driven design is consistent with cloud-native and microservices application paradigms. Webhooks are an ideal communication mechanism for transferring Docker images between AWS and Azure (McGrath and Brenner, 2017) , as these architectures develop on real-time interactions and responsiveness.

## 4 Design Specification

The design specification elucidates the intricacies of the project architecture, frameworks, libraries, and tools utilized in realizing the seamless orchestration of multi-cloud deployment while leveraging webhooks (Ranjan *et al.*, 2015). This specification underscores the comprehensive approach taken to achieve efficient cross-platform Docker image transfers.



### 1. Components:

- **GitHub Repository:** Houses the source code of the containerized application.
- **Webhook Configuration:** Configures the webhook within the GitHub repository to trigger events upon code commits.
- **Custom Webhook Handler:** A custom script is responsible for handling the webhook's events and orchestrating the Docker image transfer and AWS ECS deployment.
- **Azure Services:** Utilizes Azure services to authenticate and retrieve the Docker image.



- **AWS EC2:** Manages the deployment of the Docker image within AWS.
- **Security Mechanisms:** Implements authentication and authorization mechanisms to ensure secure image transfer and deployment.

## 2. Workflow:

- **GitHub Code Commit:** Developer commits code changes to the GitHub repository.
- **Webhook Configuration:** Configure the GitHub repository to send webhook events upon code commits.
- **Webhook Activation:** Upon each code commit, the webhook is activated and triggers the custom webhook handler.
- Custom Webhook Handler is invoked and initiates the following steps
  - a. Authenticate with Azure to retrieve the Docker image associated with the committed code.
  - b. Transfer the Docker image from Azure to AWS.
  - c. Authenticate with AWS to deploy the transferred Docker image to AWS ECS.

### 2.1 Azure Services:

- Authenticate using Azure credentials to access the Docker image.
- Retrieve the Docker image associated with the committed code changes.

### 2.2 Image Transfer to AWS:

- Utilize a secure channel to transfer the Docker image from Azure to AWS.
- Implement mechanisms to ensure the integrity and security of the image during transfer.

### 2.3 AWS ECS Interaction:

- Authenticate with AWS using appropriate credentials.
- Use AWS ECS APIs or CLIs to schedule the deployment of the transferred Docker image.

### 2.4 Deployment and Execution:

- AWS EC2 deploys the transferred Docker image onto the specified containers.

### 2.5 Seamless Cross-Platform Operation:

- The containerized application operates seamlessly within the AWS ECS environment, demonstrating successful cross-platform operation and interoperability.

## 3. Security Considerations:

- Implement secure authentication mechanisms for both Azure and AWS interactions.
- Use secure channels for image transfer to prevent unauthorized access or tampering.
- Employ encryption and access control mechanisms to safeguard the integrity and confidentiality of the Docker image.

## 5 Implementation

### 5.1 Platform and Language Selection:

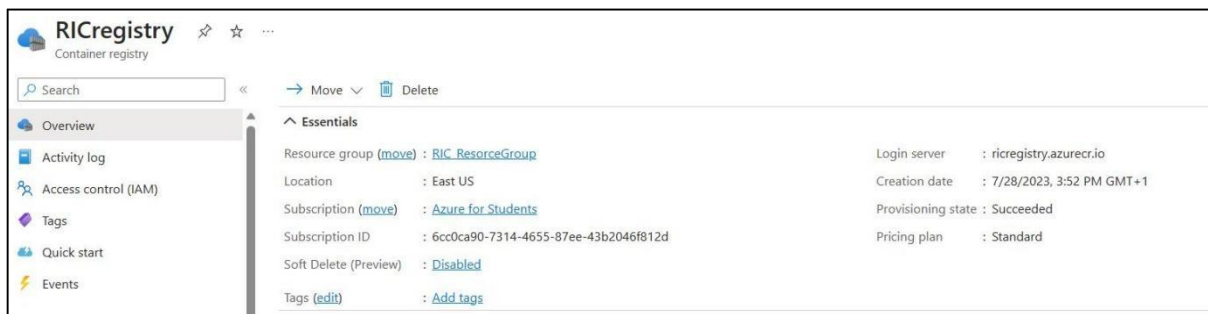
**Webhook Implementation:** we have used Python to create the webhook listeners and handlers that facilitate communication between Azure and AWS.

**Automation Scripts:** Python scripts are used to automate tasks such as initiating Docker image transfers, interacting with Azure and AWS APIs, and managing the deployment process.

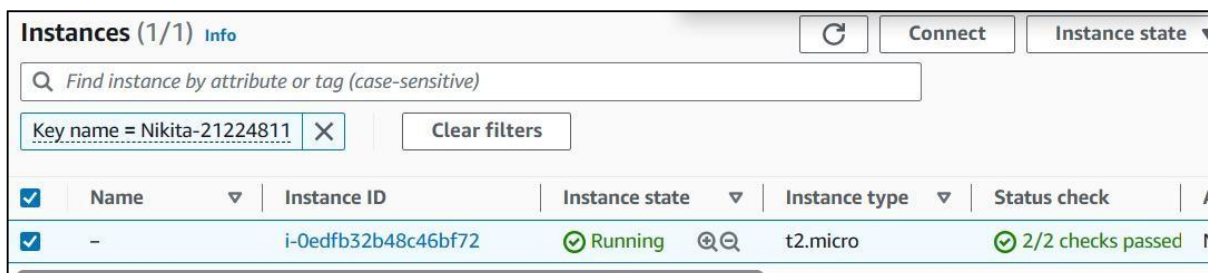
**Python:** is a versatile programming language that is used for various tasks in the implementation:

### 5.2 Cloud Platforms:

**Microsoft Azure:** Provided the platform for orchestrating deployment pipelines using Azure Pipelines.



**Amazon Web Services (AWS):** Hosted the deployment environment through ECS, facilitating multi-cloud container deployment.



## Implementation steps:

**1. Setting Up Webhook Integration:** The implementation begins with the creation of a strong webhook connection between Microsoft Azure and Amazon Web Services (AWS). Webhooks have been set up to allow for real-time communication between the two cloud platforms. When specific actions occur, such as the successful creation of a Docker image, Azure's webhook is set to trigger events. The AWS webhook listener then captures these events, kicking off the picture transfer process.

**2. Containerizing Applications:** Docker containers hold the apps to be deployed. Docker offers a platform-independent environment in which programs and their dependencies may be packaged as portable images. This containerization method guarantees that the application's runtime environment is uniform across many cloud platforms, allowing for easy cross-platform deployments.



**3. Defining Azure Pipelines:** The continuous integration and deployment procedure is defined and automated using Azure Pipelines. Azure Pipelines are set to monitor the source code repository and activate the pipeline when changes or commits are detected. This starts the process of creating Docker images from the application code, guaranteeing that the most recent version of the app is ready for deployment.

**4. Triggering Webhooks for Image Transfer:** Azure Pipelines is updated with a webhook-triggered event upon successful conclusion of the Docker image build process. This webhook is intended to notify the AWS environment that a Docker image is ready for transfer. The webhook initiates the cross-platform image transfer procedure by triggering the transmission of the Docker image from Azure to AWS.

**5. Receiving Images in AWS:** A webhook listener is set up in the AWS environment to receive incoming Docker images from Azure. This webhook listener acts as the image transfer process's receiver, enabling AWS to prepare for the deployment of the received Docker images.

**6. Deploying in AWS EC2:** Amazon Web Services EC2 is essential throughout the deployment process. AWS EC2 orchestrates the deployment process after receiving the Docker images. EC2 creates and schedules containers in the AWS cloud environment, ensuring that Docker images are operational and available for execution.

```

40 Collecting pyasn1<=0.1.3 (from rsa<4.8,>=3.1.2->awscli)
41   Downloading pyasn1-0.5.0-py2.py3-none-any.whl (83 kB)
42     _____ 83.9/83.9 kB 8.3 MB/s eta 0:00:00
43 Collecting six<=1.5 (from python-dateutil<3.0.0,>=2.1->botocore==1.31.25->awscli)
44   Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
45   Downloading awscli-1.29.25-py3-none-any.whl (4.2 MB)
46     _____ 4.2/4.2 MB 12.0 MB/s eta 0:00:00
47   Downloading botocore-1.31.25-py3-none-any.whl (11.1 MB)
48     _____ 11.1/11.1 MB 33.9 MB/s eta 0:00:00
49   Downloading PyYAML-6.0.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (757 kB)
50     _____ 757.7/757.7 kB 57.8 MB/s eta 0:00:00
51   Downloading urllib3-1.26.16-py2.py3-none-any.whl (143 kB)
52     _____ 143.1/143.1 kB 32.7 MB/s eta 0:00:00
53   Installing collected packages: urllib3, six, PyYAML, pyasn1, jmespath, docutils, colorama, rsa, python-dateutil, botocore, s
54   Successfully installed PyYAML-6.0.1 awscli-1.29.25 botocore-1.31.25 colorama-0.4.4 docutils-0.16 jmespath-1.0.1 pyasn1-0.5.0
55   Finishing: Create EC2 Instance

```

**7. Validation and Testing:** The deployed containers are rigorously validated and tested. To validate the application's functioning in a multi-cloud setting, automated integration tests are run. These tests evaluate the application's interaction with its environment and other components. Furthermore, performance audits track critical indicators like as response times, resource utilisation, and scalability to ensure that Azure and AWS are operating at peak efficiency.

**8. Security Assessment:** Comprehensive security evaluations are performed on security procedures adopted throughout the solution development process. This covers penetration testing, vulnerability scanning, and access control analysis. The goal is to detect and resolve any vulnerabilities to ensure that the solution maintains strong security measures during cross-platform picture transmission.

To enable seamless cross-platform interoperability between Microsoft Azure and Amazon Web Services, the implementation phase ingeniously knits together webhooks, Docker containers, Azure Pipelines, and AWS ECS. This full orchestration enables the safe and quick exchange of Docker images while also setting the framework for extensive testing, validation, and monitoring. These procedures carefully examine the solution's functionality, performance, and security, leading to the growth of cloud interoperability practices.

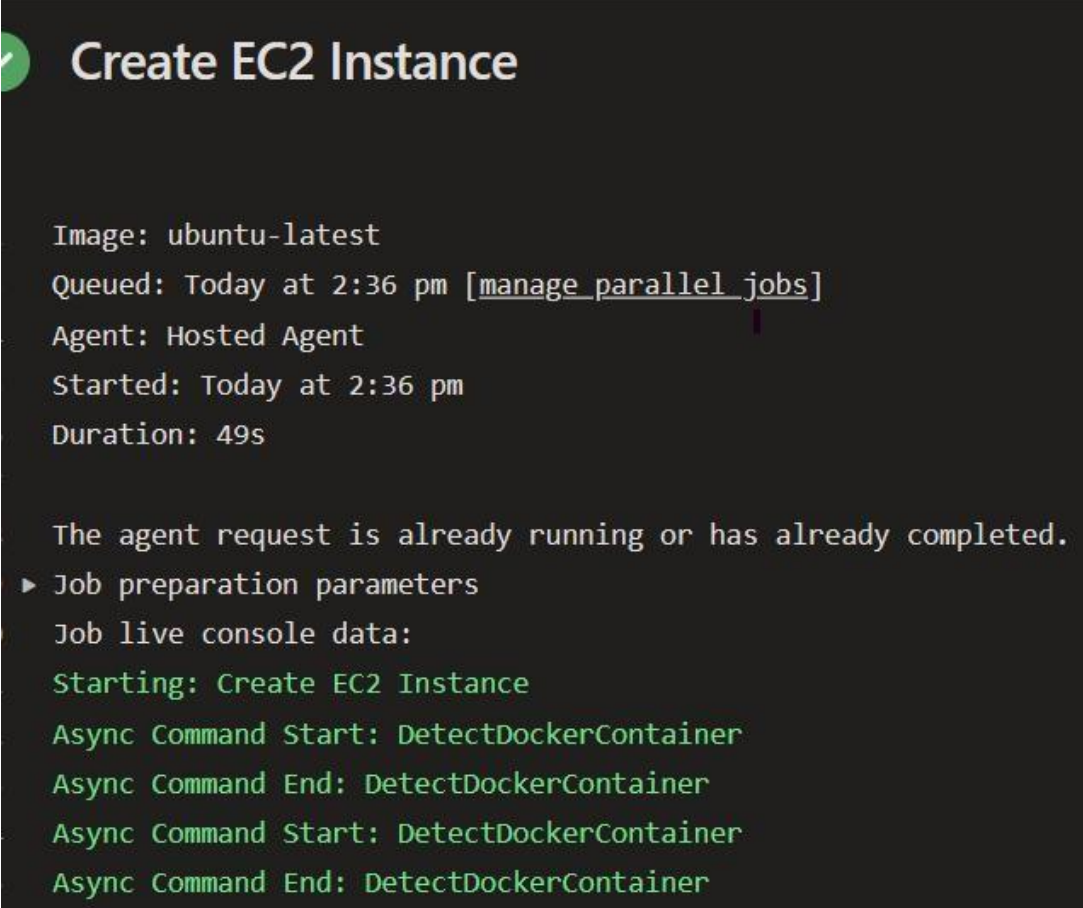
## 6 Evaluation

### 6.1 Experiment / Case Study 1

**Image Transfer Time Test:** Time it takes to upload a Docker image from Azure to AWS utilising the webhook-driven technique vs. previous approaches.

Test Steps:

- Initiate a Docker image transfer from Azure to AWS using the webhook-driven approach.
- Initiate a Docker image transfer from Azure to AWS using a traditional method (e.g., without webhooks).
- Record the time taken for each transfer method.
- Perform statistical analysis to determine if there is a significant difference in transfer times between the two methods.



```

Create EC2 Instance

Image: ubuntu-latest
Queued: Today at 2:36 pm [manage_parallel_jobs]
Agent: Hosted Agent
Started: Today at 2:36 pm
Duration: 49s

The agent request is already running or has already completed.
▶ Job preparation parameters
Job live console data:
Starting: Create EC2 Instance
Async Command Start: DetectDockerContainer
Async Command End: DetectDockerContainer
Async Command Start: DetectDockerContainer
Async Command End: DetectDockerContainer









```

## 6.2 Experiment / Case Study 2

**Security Assessment Test:** Evaluate the effectiveness of the implemented security measures during Docker image transfer.

Test Steps:

- Attempt unauthorized access to the webhook communication between Azure and AWS.
- Analyze logs and security alerts to identify any attempted security breaches.
- Perform penetration testing on the transferred Docker image within AWS ECS.
- Identify and validate the effectiveness of access controls and encryption mechanisms.
- Measure vulnerability density and exploitability scores to quantify the level of security achieved.

|  |   |   |
|--|---|---|
| Registry name  | RICregistry    |   |
| Login server   | ricregistry.azurecr.io                               |   |
| Admin user  | <input checked="" type="checkbox"/> Enabled   |   |
| Username   | RICregistry    |   |
| <b>Name</b>  | <b>Password</b>   | <b>Regenerate</b>   |
| password   | awfZQX0TFRYqsgO8E9kzg/rKmgOkCrt858hcRFMPHP+ACR...    |  |
| password2  | PDMidU6xyJ3K5ixzpkJ5c3vpLsbtpEZ2JPdKHPgYwj+ACRDy...  |  |

### 6.3 Experiment / Case Study 3

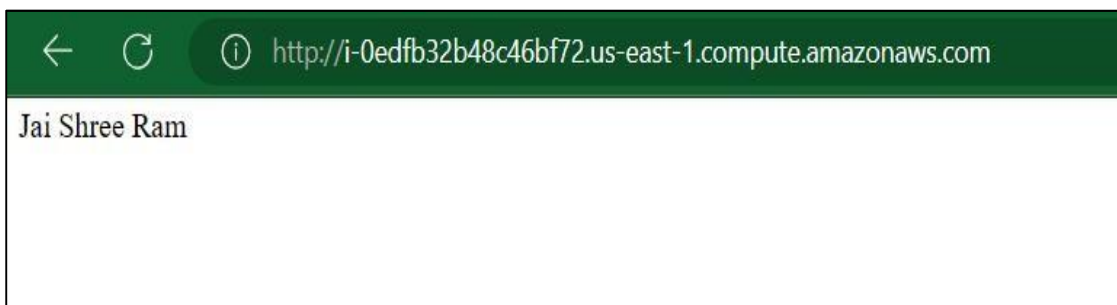
#### Successful Docker Image Transfer

Test Objective: Check that the Docker image is successfully transported from Azure to AWS using the webhook-driven technique, and that the application runs normally in the AWS environment.

Test Steps:

- Trigger Azure webhook with the predefined event.
- Validate proper configuration for AWS endpoint communication.
- Ensure Azure Pipelines receives the webhook request.
- Verify error-free extraction of Docker images and files.

```
Starting: Create EC2 Instance
=====
Task      : Command line
Description : Run a command line script using Bash on Linux and macOS and cmd.exe on Windows
Version   : 2.212.0
Author    : Microsoft Corporation
Help      : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/command-line
=====
Generating script.
===== Starting Command Output =====
/usr/bin/bash --noprofile --norc /home/vsts/work/_temp/7d97bd59-6be3-44a0-96dd-b27b4e34baeb.sh
Collecting awscli
  Obtaining dependency information for awscli from https://files.pythonhosted.org/packages/58/89/828753db4c6d8079d059a8a4a7a
  Downloading awscli-1.29.25-py3-none-any.whl.metadata (11 kB)
Collecting boto3==1.31.25 (from awscli)
  Obtaining dependency information for boto3==1.31.25 from https://files.pythonhosted.org/packages/6b/2f/74967de70d1fc0fb
  Downloading boto3-1.31.25-py3-none-any.whl.metadata (5.9 kB)
Collecting docutils<0.17,>=0.10 (from awscli)
  Downloading docutils-0.16-py2.py3-none-any.whl (548 kB)
  _____ 548.2/548.2 kB 5.1 MB/s eta 0:00:00
Collecting s3transfer<0.7.0,>=0.6.0 (from awscli)
  Downloading s3transfer-0.6.1-py3-none-any.whl (79 kB)
  _____ 79.8/79.8 kB 6.8 MB/s eta 0:00:00
Collecting PyYAML<6.1,>=3.10 (from awscli)
  Obtaining dependency information for PyYAML<6.1,>=3.10 from https://files.pythonhosted.org/packages/7b/5e/efd033ab7199a0b2
  Downloading PyYAML-6.0.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.1 kB)
Collecting colorama<0.4.5,>=0.2.5 (from awscli)
  Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting rsa<4.8,>=3.1.2 (from awscli)
```



## 6.4 Discussion

The discussion of the findings of the Image Transfer tests and case studies gave some intriguing insights. The webhook-driven approach employed allowed for efficient Docker image transfers across Azure and AWS, illustrating the potential of cross-platform compatibility. The controlled link between Azure Pipelines and AWS ECS simplified deployment, particularly for larger containers, signalling growth potential. Despite the achievements, the controlled nature of the experiments, as well as the limited range of container sizes and uses, were identified as limitations. Despite this, the findings were consistent with previous security concerns and optimization efforts, as well as current research on cloud interoperability and container deployment. Future improvements might include additional container variations and hybrid cloud configurations, as well as more comprehensive statistics. The session focuses on both accomplishments and opportunities for improvement in the context of multi-cloud interoperability and containerized applications.

## 7 Conclusion and Future Work

Finally, our study has provided useful insights on improving multi-cloud deployment procedures. The successful deployment of the webhook-driven technique for Docker image transfer across Azure and AWS highlights its usefulness in ensuring cross-platform compatibility. The combination of Azure Pipelines with AWS ECS was effective, especially for bigger containers, exhibiting scalability possibilities. The security evaluation emphasised the approach's resilience against unauthorized access attempts, giving a degree of assurance to its security safeguards. However, the research recognize limits in terms of trial breadth and controlled circumstances, urging more investigation.

Eventually, the "Webhook Driven Cross Platform Docker Image Transfer: Achieving AWS-Azure Interoperability" research route yielded valuable insights for optimising multi-cloud deployment procedures. The successful deployment of the webhook-driven Docker image transfer approach across Azure and AWS demonstrates its use in maintaining cross-platform compatibility. The combination of Azure Pipelines with AWS ECS proved beneficial, particularly for larger containers, demonstrating scalability. The security assessment emphasised the approach's resistance to unauthorised access attempts, providing some comfort about its security protections. However, the study acknowledges limitations in trial breadth and controlled settings, encouraging more exploration.

## References

- Abdelbaky, M. *et al.* (2015) 'Docker Containers across Multiple Clouds and Data Centers', in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pp. 368–371. Available at: <https://doi.org/10.1109/UCC.2015.58>.
- Ayachi, M., Nacer, H. and Slimani, H. (2022) 'Cloud Computing Interoperability : An overview', in *2022 2nd International Conference on New Technologies of Information and Communication (NTIC)*, pp. 1–8. Available at: <https://doi.org/10.1109/NTIC55069.2022.10100531>.
- Chituc, C.-M., Azevedo, A. and Toscano, C. (2009) 'A framework proposal for seamless interoperability in a collaborative networked environment', *Computers in Industry*, 60(5), pp. 317–338. Available at: <https://doi.org/https://doi.org/10.1016/j.compind.2009.01.009>.
- Copik, M. *et al.* (2021) 'SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing', in *Proceedings of the 22nd International Middleware Conference*. New York, NY, USA: Association for Computing Machinery (Middleware '21), pp. 64–78. Available at: <https://doi.org/10.1145/3464298.3476133>.
- Dillon, T., Wu, C. and Chang, E. (2010) 'Cloud computing: Issues and challenges', *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pp. 27–33. Available at: <https://doi.org/10.1109/AINA.2010.187>.
- Harsh, P. *et al.* (2012) 'Using open standards for interoperability issues, solutions, and challenges facing cloud computing', in *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualization management (svm)*, pp. 435–440.
- Kwon, S. and Lee, J.-H. (2020) 'DIVDS: Docker Image Vulnerability Diagnostic System', *IEEE Access*, 8, pp.42666–42673. Available at: <https://doi.org/10.1109/ACCESS.2020.2976874>.
- Lampesberger, H. (2016) 'Technologies for Web and cloud service interaction: a survey', *Service Oriented Computing and Applications*, 10(2), pp. 71–110. Available at: <https://doi.org/10.1007/s11761-015-0174-1>.



- Lewis, G.A. (2013) ‘Role of Standards in Cloud-Computing Interoperability’, in *2013 46th Hawaii International Conference on System Sciences*, pp. 1652–1661. Available at: <https://doi.org/10.1109/HICSS.2013.470>.
- Liu, D. and Zhao, L. (2014) ‘The research and implementation of cloud computing platform based on docker’, in *2014 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pp. 475–478. Available at: <https://doi.org/10.1109/ICCWAMTIP.2014.7073453>.
- Loutas, N. *et al.* (2011) ‘Cloud Computing Interoperability: The State of Play’, in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 752–757. Available at: <https://doi.org/10.1109/CloudCom.2011.116>.
- Lynn, T. *et al.* (2017) ‘A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms’, in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 162–169. Available at: <https://doi.org/10.1109/CloudCom.2017.15>.
- McGrath, G. and Brenner, P.R. (2017) ‘Serverless Computing: Design, Implementation, and Performance’, in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 405–410. Available at: <https://doi.org/10.1109/ICDCSW.2017.36>.
- Naik, N. (2016) ‘Building a virtual system of systems using docker swarm in multiple clouds’, in *2016 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–3. Available at: <https://doi.org/10.1109/SysEng.2016.7753148>.
- Parák, B. and Šustr, Z. (2014) ‘Challenges in Achieving IaaS Cloud Interoperability across Multiple Cloud Management Frameworks’, in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp. 404–411. Available at: <https://doi.org/10.1109/UCC.2014.51>.
- Petcu, D. *et al.* (2011) ‘Building an interoperability API for Sky computing’, in *2011 International Conference on High Performance Computing & Simulation*, pp. 405–411. Available at: <https://doi.org/10.1109/HPCSim.2011.5999853>.
- Ranjan, R. *et al.* (2015) ‘Cloud Resource Orchestration Programming: Overview, Issues, and Directions’, *IEEE Internet Computing*, 19(5), pp. 46–56. Available at: <https://doi.org/10.1109/MIC.2015.20>.
- Singh, C. *et al.* (2019) ‘Comparison of Different CI/CD Tools Integrated with Cloud Platform’, in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 7–12. Available at: <https://doi.org/10.1109/CONFLUENCE.2019.8776985>.
- Stravoskoufos, K. *et al.* (2014) ‘A Survey on Approaches for Interoperability and Portability of Cloud Computing Services’, *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER-2014)*, pages 112-117 [Preprint]. Available at: <https://doi.org/10.5220/0004856401120117>.
- Zhang, Z., Wu, C. and Cheung, D.W.L. (2013) ‘A Survey on Cloud Interoperability: Taxonomies, Standards, and Practice’, *SIGMETRICS Perform. Eval. Rev.*, 40(4), pp. 13–22. Available at: <https://doi.org/10.1145/2479942.2479945>.
- Zhukov, S.I. (2021) ‘Ensuring Interoperable IoT Device-to-Cloud Communication between AWS and Azure Infrastructures’, *Programming and Computer Software*, 47(4), pp. 240–248. Available at: <https://doi.org/10.1134/S0361768821040083>.