Impact of containerization in reducing vendor lock-in

MSc Research Project
Cloud Computing

Ali Hamza
Student ID: x23257164

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ali Hamza
Student ID:	x23257164
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	12/12/2024
Project Title:	Impact of containerization in reducing vendor lock-in
Word Count:	7654
Page Count:	19

Signature:	Ali Hamza
Date:	12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Impact of containerization in reducing vendor lock-in

Ali Hamza
x23257164

Abstract

The adoption of cloud is on an all time rise due to the numerous advantages it provides. These advantages include pay-as-you-go pricing,flexibility, on demand resources and scalability. One of the major advantages of cloud is that it relieves users of managing the hardware infrastructure. Most companies when deploying their applications use one of the famous providers like AWS, Azure or Google. By utilizing all services from the same cloud provider and using their easy to use deploy UI services, they become dependent on that one cloud provider for all their deployments. "Vendor lock-in" is the term commonly used for these scenarios, and with the increase of use in cloud without the proper knowledge and tactics used by providers to enforce their services increase, this is becoming a challenge for organization. A potential solution to this issue is the approach of using a multi-cloud or hybrid cloud strategy to deploy applications but it is not very complex to implement and not very cost efficient to maintain. This research will explore the use of Containerization to tackle this challenge by making flexible and portable cloud deployments by using technologies like Docker. This study will highlight the performance and cost benefits of deployment containerized applications to cloud as compared to non-containerized ones and will compare the migration possibilities across multiple clouds. This will offer many guidelines on how to mitigate vendor lock-in and optimize deployments to the cloud.

Keywords: vendor lock-in, multi-cloud, Performance, cloud providers, Docker, cost efficiency, containerization

1 Introduction

In recent years, the IT industry has been revolutionized by the cloud, enabling businesses to achieve significantly more without the arduous efforts of setting up the resources themselves. Cloud deployments can be very cost-effective and easy to setup. Several leading companies like **Amazon**, **Microsoft**, and **Google** offer these cloud services, including many deployment options, computing resources, storage, to businesses across the globe. They have services that allow businesses to utilize cloud services in a **pay-as-you-go** structure. The advantages that come with these services are flexibility, cost-effectiveness, and scalability. This provides benefits to startups, who often lack the capital to invest in buying their own infrastructure. By hosting their applications with these cloud service providers, they not only lower their cost but can also scale their applications as per requirement.

1.1 Research Background and Context:

While the advantages provided by the cloud are numerous, there are certain disadvantages that cannot be ignored. The most prominent drawback being vendor lock-in. It happens when a business becomes entirely dependent upon a cloud service provider or some of the specific services that they offer, making it difficult to migrate to another provider or build their own infrastructure from scratch. Moreover, these cloud providers leverage their easy-to-use UI based services for deployment to keep their customers hooked. If these services were to change and the customer wanted to move to a different cloud provider, they would not be able to do so because the same experience or the same service might not be available with other providers. A case of this can be seen with AWS who have their own distribution of Linux as the default when you set up an **EC2** instance, which is called **Amazon Linux** and comes with pre-installed packages that the user might not be aware of. Hence, with the popularity of cloud increasing, problems like vendor lock-in can become fatal.

There have been few attempts at solutions to mitigate vendor lock-in, the most popular being the hybrid cloud approach and also a multi-cloud setup. In these solutions, organizations deploy their applications using a combination of cloud service providers or use their own infrastructure along with cloud services. This solution, on the one hand, does decrease the dependency on one single cloud but, on the other hand, increases the complexity of application management and often also leads to higher costs due to the use of several cloud platforms. Another proposed solution is to use tools and technologies that are compatible across multiple cloud platforms, which makes the process of migration more feasible. Containerization is one of these technologies which, on the correct use, mitigates the vendor lock-in and also simplifies the deployment and application hosting process across various clouds.

Containerization is a technology that packages an application and all its dependencies into a single, portable unit which can run in various environments. One of the most popular tool for containerization is Docker. This technology offers powerful tools to organize containerized applications. Advantages of containerization includes portability, faster deployment, improved resource utilization along with enabling efficient scaling on demand basis. Containers facilitate the migration of applications by decoupling the applications from the infrastructure they run on. Therefore, minimizes the risk associated with vendor lock-in.

In spite of existence of research body that explores vendor lock-in in depth. most of these revolve around focusing the solution based on multi-cloud and hybrid cloud setups, which not only complicates the process extensively but also is costly. The complexities of multi-cloud solutions are not required by all users, and in most cases a relatively simpler, efficient cloud usage is required without the risk of vendor lock-in. Moreover, the existing researches only emphasize containerization with respect to multi-cloud or cloud migration that can play a key role in improved performance of cloud architectures.

1.2 Research Question:

Can containerization help in mitigating vendor lock-in? and how will it affect performance and cost efficiency?

The focus of this question is to reach to an answer on how we can use containerization to mitigate the risks of vendor lock-in associated with deploying to cloud providers while also enhancing the cost effectiveness and performance of the deployments. The significance of this research is aided by the increase in popularity of cloud and cloud providers with some of them leading the market, it naturally draws people and organizations towards them which increases the risk of them being stuck with that single cloud provider. it can not only negatively impact the operational cost but also the organization's operational flexibility. Another supporting factor for this question to be asked is that although containerization has been increasing in popularity and used for isolating applications and optimize various tasks, its role in mitigating vendor lock-in has not been fully explored. Helping us understand how containerization can be used to reduce vendor lock-in will enable them to make more calculated and informed decisions about their cloud usage & deployments and their cloud provider choices and this may help them optimize and reform their applications for better results.

This research aims to contribute significantly to the field of cloud computing. It will deepen the knowledge of containerization by exploring the specific technologies and tools involved with it, such as **Docker**. Additionally, the study will analyze the performance and cost differences between scenarios where containerization is used versus where it is not. This can also be used as a guideline and recommendations for using containerization to deploy cloud applications.

2 Related Work

This section will aim to talk in detail about the work that has been done to contribute towards cloud computing and especially covers the areas like vendor lock-in, containerization and cloud migrations. The previous body of work will be critically evaluated based on three things: their contributions to mitigating vendor lock-in, the advancements in containerization technologies and the performance and cost implication related to these containerized and non-containerized cloud environments. These discussions will provide a good basis and understanding of the gaps left by previous body of work and the contributions that the current research is trying to make in filling some of those gaps.

2.1 Vendor Lock-in & Multi-Cloud Solutions

Vendor lock-in is still one of the major concerns for most organizations that want to adopt to the cloud. it is the dependency on a single cloud provider's proprietary technologies which restricts organizations to move to other clouds. Various solutions to this problem have been explored in previous studies in which one of the popular approaches is a multi-cloud or a hybrid cloud setup.

As explored by Weldemicheal and Tesfaldet (2023) in their article "vendor lock-in and cloud migration challenges", combining on-premise and cloud infrastructure can provide a lot of flexibility to users. This also aligns with the current research as It also provides a solution to get rid of vendor lock-in. In Weldemicheal and Tesfaldet (2023) the authors discuss how organizations can store their sensitive and critical data locally in on-premise servers and use cloud providers for only scalable workloads. This is one of the solutions to avoid data leaks and being dependent on a cloud for all security but it raises issues like complexity in managing and also it may present operational challenges such as synchronization problems, this approach would require expertise in cloud and also a lot of capital to cover both the cost of on-premise and off-premise cloud which makes it less viable for small scale businesses.

Multi-Cloud setup is also one of the big approaches used to solve the issues related to vendor lock-in. It is the approach where organizations deploy their applications across multiple clouds in order to make sure they are not relying on a single provider and can scale on whichever cloud that meets their needs and requirements. Pellegrini et al. (2017) provides a great example of this as they introduced a multi-cloud management system that works as a central piece to deploying application across multiple cloud providers. So if an application can communicate with this interoperable cloud system, the same application will be deployed to multiple clouds using this system. throughout there research the authors work on a prototype for this type of system with its own architecture and how it can help be a messenger among different cloud services and the application. Although this setup mitigates the risk of provider lock-in it still relies on centralized orchestrion tools like terraform which create a dependency of its own. one of the major flaws of this research can be considered that by removing one dependency the research poses another one and this tool may require a lot of customization to address specific needs of the diverse workloads in different organization which not only complicates the adoption of this system but also increases the complexities involved in creating one. One other thing which might be an issue is the overhead this system might add to the cost to cloud deployment which might not be suitable for small scale organizations. Another good example is Waseem et al. (2024) which discusses the advantages of multi-cloud approaches and the analytics behind it. There is no experimentation or case study in this research but it uses good examples to prove its point. Both researches emphasize the use of containerization technologies in the process of multi-cloud to make it more efficient and easy to manage. Pellegrini et al. (2017) uses Kubernetes clusters for their prototype for all the mongo Db data storing and retrieving. Waseem et al. (2024) also list the advantages containerization can provide when opting for a multi-cloud approach as it makes everything much more easier to manage, This aligns directly with research at hand.

“Addressing Server-less Computing Vendor Lock-In through Cloud Service Abstraction” Mo et al. (2023) talks about how the recent growth of FaaS (function as a service) services have increased the risks of vendor lock-in. FaaS provides an easy solution to many users who want to just deploy small server-less functions to the cloud which are becoming more and more common because of the popularity of micro-services. These server-less functions although seem server-less up front but with most cloud providers they rely on using one of their services known as BaaS (back-end as a service). It is offered by cloud providers as a way to add consistency to user’s cloud function and make them persistent. This adds a dependency not controlled by user and it becomes very easy not to realize that the user has been using this underlying architecture until the user wants to move and realize he cannot because he is locked in by the underlying architecture of the cloud provider. The research also works on a solution to this by utilizing middleware abstraction services to write server-less functions so they know what they are using instead of relying on the cloud. They specifically work on Apache cloud abstraction library and compare it with the cloud services for metrics. If the goal is to write serverless functions and also make them portable the same can also be achieved by docker a containerization tool which helps packages code in small portable units which can be deployed with essentially any cloud service.

”A Containerized Template Approach for Vendor-Friendly Smart Home Integration,” Fleck et al. (2023) works on a solution to the problem that occurs with using smart home appliances. The issue is that there are a few dozen famous vendors for smart home appliances and with each vendor they have their own set of appliances and an application that helps use those gadgets. So if a user wants to use a light bulb from one vendor but a lamp from another he cannot control them together because it will require two separate apps. Because of this in most cases people get locked in to a single vendor for all their appliances even if they do not want to do so. The author's goal is to create a system that allows users to use different smart appliances but still be able to use them using a single centralized application which reduces to reliance on a single vendor and hence opens up options for users. This is the very essence of how Cloud vendor lock-in works and with this research it can help us better understand how vendors locking us in with their services is affecting the options that many organizations have when wanting to deploy applications to the cloud.

2.2 Advancements in Containerization

In the past few years we have seen more and more organization opt into using containerization as a way to navigate their complex architecture. This is beneficial to everyone because it opens up the floodgates and everyone can use these examples as a way to see the advantages provided by containerization such as portability and flexibility. Two of the most famous containerization tools are Docker and Kubernetes. Docker helps creating these small portable units and Kubernetes is an container orchestration tool that helps manage these portable units. The article "Study Containerization Technologies like Docker and Kubernetes and their Role in Modern Cloud Deployments" Agrawal and Singh (2024) talks about the role these technologies play in modern day cloud deployments. It talks about how Kubernetes can be used as a service to aid cloud deployments and Docker can be used with FaaS to allow portable serverless functions to exist. The article also focuses on the advantages these tools have provided us and how they have made it easier to work with multi-cloud setups by becoming a backbone of it and removing a lot of complexities from it.

The research Lohumi et al. (2023) focuses on Cloud to Cloud migrations and how when migrating a cloud VM to another VM an organization can suffer from losses due to the downtime. The purpose of this research is to dive deep into C2C migrations and also get a detailed overview of how everything works when migrating a VM. It's proposed solution is a containerized approach which reduces the time it takes to migrate due to the increased portability which in return decreases the downtime. The same can be seen with Rovnyagin et al. (2023) which uses Docker as its focus to improve the latency and overhead it takes to migrate applications between clouds. It uses a Platform as a service architecture with metrics to showcase the reduction of latency when implementing Docker with deployments. The same concept will be applied in this research to showcase how properly configuring docker can not only make it easier to migrate with reduced latency but also reduce the chances of making your application so complex that it removes the possibility of being able to migrate due to the overhead it costs.

2.3 Containerization & Cost Reduction

We have discussed previously what vendor lock-in is and what containerization is and how it used in various ways in cloud and how it effects migrations and multi-cloud setups and enhances performance but, in this section, I want to talk about the cost side of containerization and how it effects cost according to previous body of work and what more can be accomplished.

As previously discussed cases of vendor lock-in and how containerization can be used to make effective changes to the architecture to reduce them. This section can be used as an example of how containerization can not only help with that but also reduce the cost of a big application with the right configurations. Dang (2022) talks about how containerized deployments have a lesser migration costs than normally deployed applications because of how these tools isolate resources and do not rely on the underlying architecture of it all. This case is supported by Rovnyagin et al. (2023) which talks about the advantages of using Docker due its open source-ness and self-sufficiency in deploying applications. The claim of this article is that most organization will eventually shift to using containerization technologies for their deployments but there is no statistic provided to backup this claim.The authors Ramesan (2023) work on a solution with maximum cost efficiency with Kubernetes and showcase how its orchestration ability can help manage even the most complex architecture but this research lacks in metrics and an actual implementation. The article Anon (n.d.) talks about the advantages and disadvantages of using different containerization tools and also emphasizes on many ways on how the costs can be reduced by using some of these tools.

2.4 Research Niche

There are many researches that talk about vendor lock-in scenarios and containerization but most focus on state of the art and high cost solutions like multi-cloud and hybrid approaches. Containerization can be used to facilitate these solutions but it can also be used to mitigate the need to go for that solution in the first place and the potential of that has still not been fully explored. This research will aim to focus on that area of the cloud and work with containerization technologies to deploy applications in a way that they are portable and can be easily migrated to other clouds. I will use the previous body of work as a stepping stone and inspiration to support the argument that containerization provides the possibility to create and deploy an application that can be cost effective and also not be subjected to vendor lock-in. This research will incorporate the use of a real world case scenario by trying to navigate cloud deployment with a freshly created application and provide insights and analysis on containerized deployment performed as compared to a normally deployed application.

3 Methodology

This section will be used to discuss the methodology used for this research and the plan of steps that will help make sure the research is properly and effectively done. the main purpose is to explore the impact of containerization in cloud deployments especially in reducing vendor lock-in. This will involve finding the tools and technologies to be used and configurations involved with cloud deployments. I will compare many different approaches to cloud deployment and also use two separate cloud provider to come up with useful findings and results.

3.1 Research Tools & Technologies

This step will involve identifying the tools and resources necessary to conduct the research effectively. After reviewing and comparing several tools and data, I will decide on the ones most relevant to the research objectives. Finding the right resources is crucial for carrying out solid research and experimentation, so this will be the first step. This involves finding the right tools and technologies to conduct various experiments. After careful consideration of many tools a few will be selected which are most relevant to the research at hand. Making sure all the tools are carefully picked is necessary for the success in finding useful insights from this research. The tools and technologies are as follows:

- Docker for containerization
- AWS Cloud Watch and Azure monitoring for monitoring
- AWS and Azure as two separate cloud service providers
- Node Js and Express for Back-end and React and Tailwind for Front-end

3.2 Setup and Experimentation

After deciding on the tools and technologies in the next part is deciding on a setup and the criteria on how to properly conduct these experiments. Deciding on a setup helps come up the scenarios and the experiments better because it sets a baseline on what needs to be done in order to perform these experiments and what type of resources are required. The setup will include:

- Creating a full-stack application with a third party API integration
- Configuring the application on PaaS services on AWS
- Configuring the application on IaaS services on AWS
- Migrating the Application from PaaS to azure and IaaS to Azure
- Setting up proper monitoring tools to get metrics like CloudWatch and Azure monitoring

The experimentation is the most crucial step of this research as it will help us come to understanding of the results related to the research in question and come up with useful findings that be used to support the arguments placed in the research.The experimentation can be divided into three different sections based on the above mentioned scenarios and their details are as follows:

- Deploy Back-end to Elastic Beanstalk on AWS and setup Front-end on S3 bucket
- Deploy Dockerized Back-end and Front-end on EC2 instances
- Setup RDS for Elastic Beanstalk and Dockerized SQL server with back-end
- Migrate the EC2 instances to Azure VM with help of Docker

3.3 Evaluation

This part is the last setup in our research and will include the decision of what findings need to be focused on to showcase the results and best support the research and the claims. The decision will be based on multiple factors such as which metrics are available as common for each of the cloud services and also which metrics are the most useful for the deployed application. Some of the metrics that will be focused on for insights are as follows

- metrics like CPU utilization, Network in and Network Out
- Flexibility and Scalability of the containerized application
- Cost evaluation for the Dockerized instances
- Cost evaluation of Elastic Beanstalk and Database setup with it
- Performance and Cost metrics of Azure Virtual Machine after migration

4 Design Specification

This sections provides an overview of the techniques, architecture, frameworks, and deployment strategies used for the implementation of the proposed system. Developed to support a full-stack web application environment, the system leverages cloud infrastructure to improve scalability, fault tolerance, and manageability. The design incorporates streamlined integration and delivery, enabling the smooth development and deployment cycles. Below, the architecture in two distinct scenarios have been discussed, highlighting the difference in their interaction with each other and the different components.

4.1 System Architecture Overview

The diagram explains the flow from development till the deployment and showcases the two distinct architecture scenarios for the full-stack web application. While bother scenarios use similar core technologies, they utilize different hosting strategies and deployment configurations.

The first scenario use Amazon Web Services(AWS) Elastic Beanstalk to deploy the back-end server, meanwhile storing the front-end application in an S3 bucket. The back-end server communicates with an Amazon RDS SQL instance to streamline database operations. The entire application is monitored using AWS Cloud watch.

The second scenarios puts emphasis on containerization using Docker. Instead of Elastic Beanstalk, multiple EC2 instances have been utilized by the architecture to deploy the applications, in which each instances is being run as a Docker container. Front-end, back-end and database(SQL Server) are all hosted in separate Docker containers in this configuration, Similar to Scenario 1, the monitoring is handled by Cloud Watch.

4.2 Scenario 1: Using Elastic Beanstalk and S3 Architecture

Scenario 1 is a streamlined architecture that utilized AWS-managed services to minimize operational complexity, This is ideal for the teams that are unable to handle the intricacies of complex server management.

- **Front-end:** built using React, is stored and served from an S3 bucket. S3 is an object storage device from AWS which is highly scalable, durable and cost-effective. Hosting static assets (such as HTML, CSS, and JavaScript) on S3 ensure the delivery of low-latency content to users.

- **Back-end:** AWS Elastic Beanstalk is used to deploy the back-end application, which is a platform-as-a-service (PaaS) solution that eliminates the need to manage the underlying infrastructure. Elastic Beanstalk handles the provisioning of EC2 instances, load balancing, and scaling, allowing developers to deploy their application code directly. Due to the asynchronous event-driven architecture, Node.js is chosen for the back-end, which efficiently handles a large number of requests. Providing a robust set of features for building web and mobile applications, Express.js is a flexible and minimal framework of Node.js.

- **Database:** The architecture uses Amazon RDS for the database layer, which is a managed relational database service supported by SQL databases. A traditional SQL based database is assumed in the scenario, i.e. MySQL or PostgreSQL, that are used for robust data storage. Through automatic backups, multi-AZ deployments, and automated fail-over, the RDS instance is configured for high availability and durability,

- **Monitoring:** AWS CloudWatch monitors the entire architecture, providing insights to the health and performance of the application. CloudWatch automatically reacts to the changes in the environment, collects and tracks metrics, and set alarms. It can be used to monitor RDS, Elastic Beanstalk and the application entirely.

4.3 Scenario 2: Dockerized EC2 Instances

In the Second Scenario, the architecture is developed in such a way to provide more control over the infrastructure using Docker for containerization and EC2 instances to deploy the application. This scenario is more complex compared to scenario 1 but is ideal for the team that require specialized infrastructure.

- **Front-end:** The React application is packaged in a Docker container and is deployed using an EC2 instance rather than hosting the front-end on S3. This approach utilizes the containerization ability to provide a consistent environment across development, testing, and production stages, eliminating many problems. Nginx or any other lightweight web server can be used to serve the containerized fronted inside the Docker container, delivering static files to the client.

- **Back-end:** Running a Docker container, the back-end is also containerized and deployed on an EC2 instance separately. Docker ensures the consistent back-end environment, similar to front-end, from local development to production. This makes the decoupling of services easy, allowing horizontal scaling where each service(front-end, back-end, and database) can scale independently.

- **Database:** Following the same pattern, the database is also containerized and deployed on a separate EC2 instance. The architecture uses a SQL server, ensuring the compatibility with Docker. SQL Server is run in its own container, making the sure of the advantages like isolation and compatibility as other components, This is particularly valuable for the teams that require a self-managed database rather than using managed services like RDS. Using a Dockerized database, the team is able to customize the configurations, manage backups, and restored operations independently of AWS services.

- **Monitoring:** Similar to Scenario 1, CloudWatch is used to monitor the performance and health of the EC2 instance and Docker containers running on the them. CloudWatch is able to track resource utilization(CPU, memory, etc.), container statuses and the overall performance of the application.

4.4 Comparative Analysis of Both Scenarios

Both scenarios described before differ in terms of flexibility, scalability, and management overhead. The visual representation of architecture diagram can been seen in Figure 1

- Scenario 1 offers easy management and a simplified process by offloading most infrastructure responsibilities to AWS. It is the most suitable for the teams focused on development while outsourcing infrastructure management to a cloud provider.
- Scenario 2 provides detailed insights, which makes it the most compatible for teams that require control over configurations or desire to self-manage infrastructure. This scenarios enables them for increased flexibility but comes with the trade-off of more complexity and management overhead.

Both scenarios are able to efficiently scale, depending on the use case and organizational requirements. Particularly, Docker-based deployments (Scenario 2) grant more modularity, with each component (front-end, back-end, database) independently scalable across multiple instances or clusters. However, Elastic Beanstalk (Scenario 1) removes much of the complexity and offer auto-scaling and load balancing out of the box.

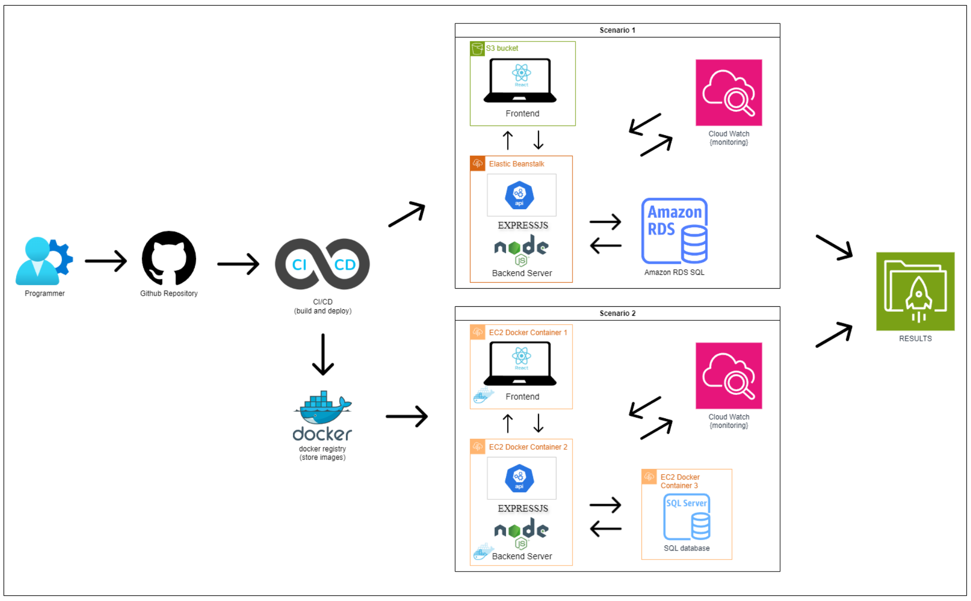


Figure 1: architecture diagram

5 Implementation

The Implementation of the system was carried out according to the architecture defined in the design specification process. A front-end and a back-end application was made and integrated with each other with the plan of deploying it to cloud and also monitoring the outputs to get results. The process involved the use of multiple languages and tools in order to achieve useful analytics of system functionality , scalability and manageability.

5.1 Front-end

The front-end of the application was constructed using the React JS[1] library with the help of tailwind CSS[2] for styling purposes. The application is a Anime listing website in which the user can see all the available animated movies and series. sort them according the popularity, search through them, add them to their personal list and also manage the list. The shows and movies are shown using the The Movie Database TMDB API[3] and tweaking it so it only showcases anime and animated listing. This front-end application will serve as a website which is used to simulate a generic website with basic CRUD operations and API usage. The front-end is dockerized with ngnix configurations so it can be deployed to a virtual machine by just running a few commands and the build files are also generated so it can be hosted on any static website hosting service provided by many cloud providers. Table 1 below showcases the front-end implementation in tabular form.

Table 1: Frontend Details

Aspect	**Description**
Frameworks and Tools	React, JavaScript, CSS
Generated Assets	Production-ready static files (HTML, CSS, JS)
Deployment Method	Stored in S3 with CloudFront for Scenario 1 & Dockerized for Scenario 2
Performance Features	Error Handling, Lazy loading and caching optimizations

5.2 Back-end & Database

The back-end of the application is constructed with nodeJS[4] which is a javascript runtime popular for creating servers. I have used the ExpressJS[5] framework for node to create the server. User management and List management endpoints were created in this process, security and efficient communication was also made sure. A big part was also handling all the errors and checking the inputs in all the endpoints using Validation and User authentication middle wares. The whole backend was also dockerized so it can be deployed to any virtual machine using just a few docker commands.Database configuration for both mysql and Amazon RDS was added for this application In order to store all the user lists

[1]React: `https://react.dev`
[2]Tailwind CSS: `https://tailwindcss.com`
[3]TMDB: `https://developer.themoviedb.org/reference/intro/getting-started`
[4]NodeJS: `https://nodejs.org/en`
[5]ExpressJS: `https://expressjs.com`

and also properly simulate a fully functioning backend server. Table 2 showcases the implementation.

Table 2: Backend and Database Details

Aspect	**Description**
Frameworks and Tools	Node.js, Express.js for the backend; Amazon RDS or Dockerized SQL Server for the database
Generated Outputs	RESTful API endpoints for application functionality and persistent data storage
Deployment Method	Backend deployed via AWS Elastic Beanstalk with dockerized Mysql for Scenario 1 or Dockerized on EC2 instances with Amazon RDS for Scenario 2
Performance Features	Asynchronous handling , optimized middleware usage, automatic backups, Multi-AZ deployments, and scalable architecture

5.3 Deployment

The deployment of the application is divided into multiple scenarios. The scenarios will be compared to come out with an answer to the research question and provide some analytics and result. Each of the scenario is explained as follows.

5.3.1 Scenario 1

In this scenario the back-end of the application is deployed on an EC2 instance using docker as the main method of running the code. docker is installed in the instance and code is cloned from GitHub and deployed using docker commands and the database is also deployed using docker. The same is done with front-end but in a separate machine and the integration between the front-end and back-end is verified.

5.3.2 Scenario 2

In this scenario the focus is on the native amazon services like elastic beanstalk and S3 bucket. First the back-end is uploaded to elastic beanstalk environment and after all the configuration is done and environment health is OK I attached an amazon relational database (RDS) instance to the environment and deploy the back-end. The front-end of the application is deployed as a static single page application on S3 bucket and the integration between back-end and front-end in verified.

5.3.3 Scenario 3

In Scenario 3, the focus is on the migration of the application to another cloud and for this instance I used azure. I uploaded the application to the azure virtual machine using docker and also uploaded the front-end on azure static website hosting service but it was not successful. the purpose was to showcase the flexibility of dockerized applications as compared to vanilla applications.

All the deployments are maintained and monitored using amazon cloud-watch and azure monitoring services. Table 3 showcases the deployment details in tabular form.

Table 3: Deployment Details

Aspect	**Description**
Deployment Strategies	Scenario 1: Backend deployed on AWS Elastic Beanstalk; frontend stored in an S3 bucket; database managed using Amazon RDS. Scenario 2: All components (frontend, backend, and database) Dockerized and deployed on separate EC2 instances. Scenario 3: Frontend and backend co-hosted on an Azure Virtual Machine, with the database hosted on the same VM or a managed Azure SQL Database.
Monitoring Tools	AWS CloudWatch for tracking application performance, resource utilization (CPU, memory, etc.), and container health (Scenarios 1 and 2). Azure Monitor for logging and performance tracking on the Virtual Machine, including diagnostic metrics and custom alerts (Scenario 3).
Performance Optimization	Automated scaling through Elastic Beanstalk (Scenario 1); manual scaling of EC2 instances (Scenario 2); Azure Autoscale and proactive monitoring for VM resources in Scenario 3.

6 Evaluation

This section provides an in-depth analysis of the implementation of the system, its results and findings. The evaluation done will be focused around the research question and provides metrics like performance charts, cost analysis and also the benefits of containerization in cloud infrastructure. Visual graphs will be used to demonstrate the findings and their significance will be discussed.

6.1 Case Study 1: Elastic Beanstalk for Back-end

Elastic Beanstalk was utilized for the back-end of the server and it was evaluated as a back-end option for a web-based application. The key metrics that were collected and focused on during the experiment was CPU utilization and Network In/out. These will be used to highlight the efficiency of the back-end using elastic beanstalk in handling the back-end server requests.

Figure 2 showcases the CPU utilization, the Network In & Out and also the environment health. These findings indicate that elastic beanstalk provides a straight forward managed back-end server. The resource utilization on this server was also the lowest among all other methods used in this research but there are also some negatives that come with it. It's auto scalability might not be cost efficient for high throughput applications and also it needs a separate service to manage the Database like RDS in this scenario which has drastic effects on the overall cost of this setup.

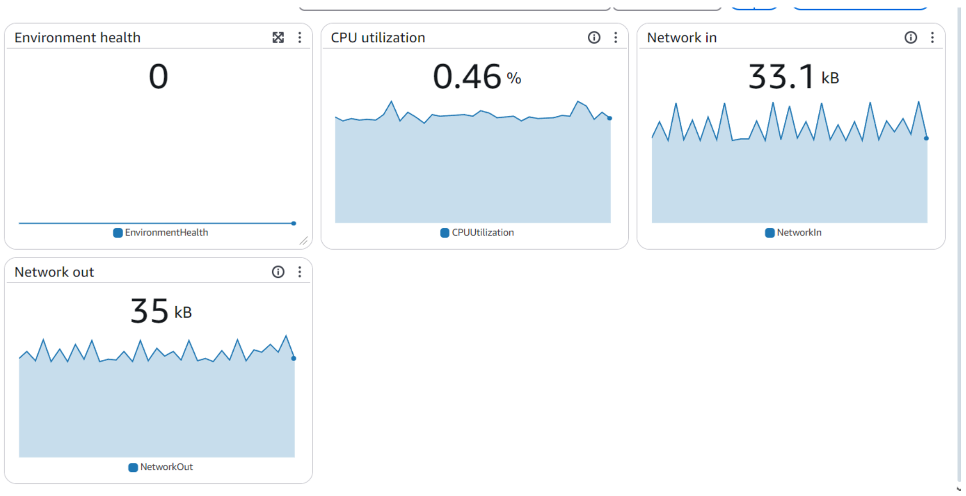


Figure 2: Amazon Elastic Beanstalk metrics

6.2 Case Study 2: S3 bucket for Front-end

S3 bucket was utilized to deploy a static react single page website to use in combination with elastic beanstalk. Although no specific metrics were collected for this service but it proved to be a lightweight solution for applications that do not require any computation on the front-end. The lack of need for a compute resource like virtual machines make it a very cost friendly solution for front-end deployment. It can be integrated with Cloud-front to further enhance performance and reduced latency.

6.3 Case Study 3: Dockerized EC2 instances

This experiment was performed with containerization and cloud migration in mind. using only services that can easily be available with any other cloud provider and also mitigating any dependencies on the current provider. One of the EC2 instances were used for the dockerized back-end and MySQL the other instance was used for front-end of the application. Both of these were analyzed on metrics like CPU utilization and Network In/Out.

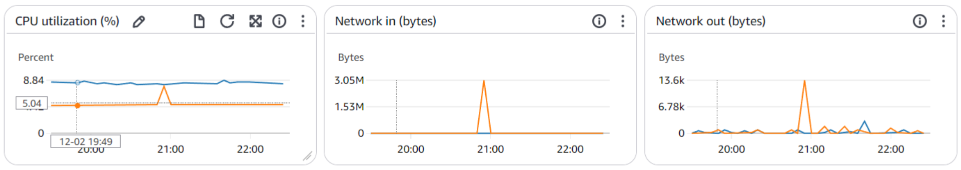


Figure 3: Amazon EC2 metrics

As shown in Figure 3 this setup had high resource usage especially compared to elastic beanstalk. However, when the SQL server and the back-end is combined and dockerized to run independently in the same VM it is hight cost effective. Since it does not rely on AWS services to run and can be deployed on any cloud provider with a VM architecture so the potential for migration is increased. In this scenario Containerization allowed for precise resource utilization and allocation and also customization when it comes to scaling.

6.4 Case Study 4: Migration to Azure

This experiment was done to evaluate the portability of containerized environment in IaaS models as compared to PaaS models provided by cloud providers. The dockerized back-end and front-end was fairly easy to just setup and run on the Azure VM and the same metrics like CPU utilization and network In and Out were analyzed to compare the difference in performance.

The process of migration showcased that containerization is very effective in enabling a seamless "lift-and-shift" deployment. The performance metrics are showcased in Figure 4. When I tried doing the same thing with the S3 bucket and Elastic beanstalk blockers like extra cost to store RDS backups locally slowed the process down and made it complex and less cost efficient. I tried deploying the front end to azure static website hosting but it was a different process then deploying on S3 bucket which makes it less beginner friendly and adds extra overhead.

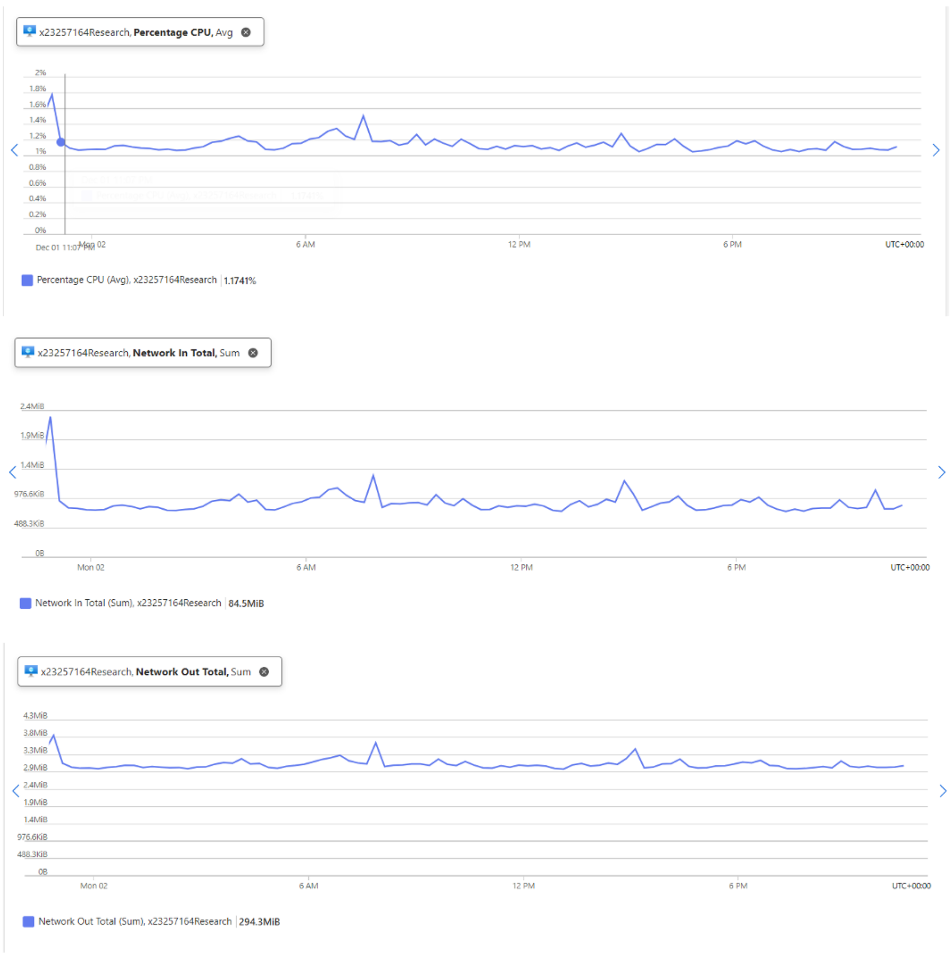


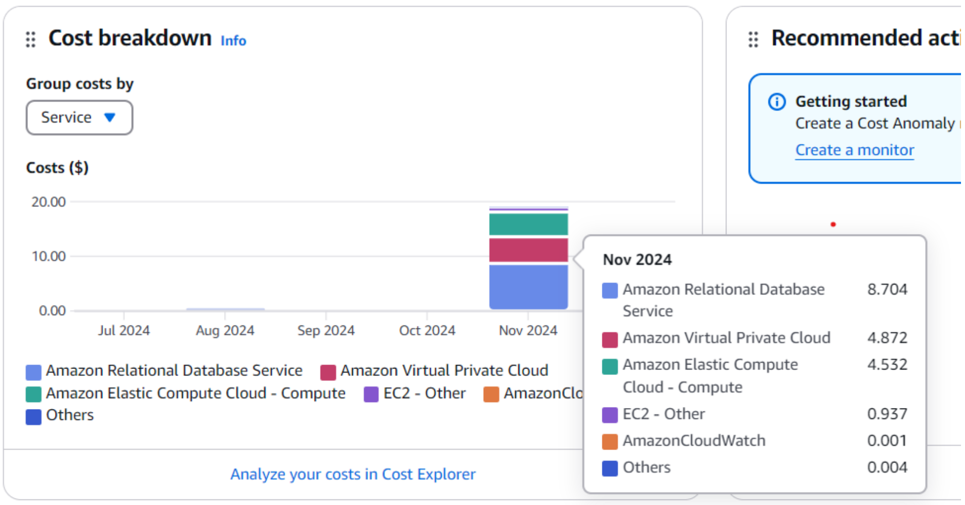
Figure 4: Azure VM metrics

6.5 Cost and Performance Analysis

This analysis was conducted to compare the cost when compared to the performance provided by each of the methods use in this research.

- **Elastic Beanstalk**: A higher cost for operation but efficient in resource usage, which emphasizes ease of use and performance over cost efficiency.
- **EC2 dockerized instances**: Overall the best performance to cost balance providing a balance between portability and efficiency particularly when considering the customizations and the cost saved in using one VM to run multiple docker containers.
- **RDS vs Dockerized SQL server**: the high cost of RDS cannot be justified with ease of use especially with applications with high volume of database dependency and usage. this just reaffirms the advantages of setting up your own SQL server with help of containerization.
- **S3 bucket**: Minimal costs and simplicity when deploying static websites makes it the best option for single page web application.

Figure 5 and Figure 6 showcase the accumulated costs of all services used in this research to provide the better understanding of the analysis. A detailed cost analysis could not be retrieved due to the restrictions caused by using AWS academy resources which restrict accessing AWS cost explorer and with Azure I could only retrieve the usage of credits they provided with their free tier.



🟦 Amazon Relational Database Service 🟥 Amazon Virtual Private Cloud
🟩 Amazon Elastic Compute Cloud - Compute 🟪 EC2 - Other 🟧 AmazonClo
🟦 Others

Nov 2024

🟦 Amazon Relational Database Service	8.704
🟥 Amazon Virtual Private Cloud	4.872
🟩 Amazon Elastic Compute Cloud - Compute	4.532
🟪 EC2 - Other	0.937
🟧 AmazonCloudWatch	0.001
🟦 Others	0.004

Figure 5: Amazon Billing

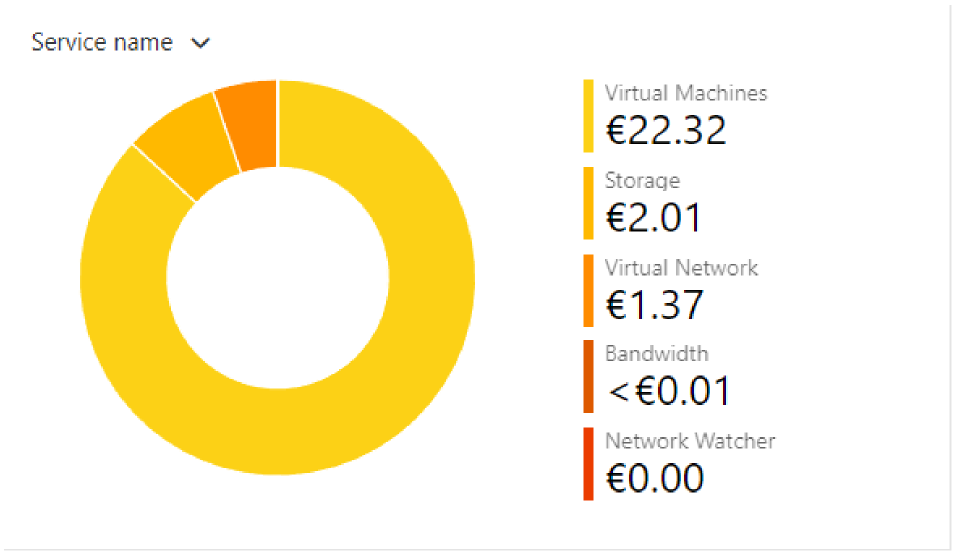


Figure 6: Azure Virtual Machine billing

6.6 Discussion

The experiments provide valuable insights into the trade offs and comparisons between provider hosted services and self-managed and containerized solutions like Docker containers etc. By using the results and findings from the implementation and design process and also the experiments done it is evident that containerization significantly enhances the flexibility, cost efficiency and portability of the applications and it also mitigates the risks involved with vendor lock-in by reducing the usage of PaaS services provided by many providers and makes the use of IaaS services easier and beginner friendly leveraging technologies like docker.

6.6.1 Improvements

- The metrics collection could have been extended by also adding memory usage, disk I/O and response times. It would make for a more comprehensive performance analysis and help capture the full operational footprint of each deployment.
- Stress testing could've been done for a better understanding and comparison between the auto scaling of cloud services and the custom scaling set in docker containers. The performance of the systems could've been tested better by increasing the load on the application.
- The experiment could've been conducted over a longer period of time to see the effects on the system with overtime usage of the application.
- Individual breakdowns of the cost would've been helpful to provide a more detailed analysis cost report but it was not possible in this research due to using free trial credits provided by these cloud providers.

7 Conclusion and Future Work

The goal of this research was to explore the deployment of applications to cloud while mitigating the risk of vendor lock-in. By using containerization as the core technology, we evaluated its effectiveness in facilitating seamless migrations and reducing dependency on cloud providers' platform-specific services. Throughout the experiments, we deployed the application in multiple configurations, including AWS Elastic Beanstalk, S3 Bucket, Dockerized EC2 instances, and an Azure Virtual Machine environment. These experiments provided valuable insights into cost, performance, and the practicality of containerized application hosting.

The findings showcased the strength of containerization in enabling portability and flexibility. Migrating a Dockerized application from AWS EC2 to Azure proved the simplicity and efficiency of the approach as compared to the challenges of migrating a PaaS deployed application. Furthermore, hosting the back-end and database in a single Dockerized environment on EC2 proved to be significantly more cost-effective than using AWS RDS.

However, the research also identified several limitations. The experiments were conducted with a single application and set of metrics, limiting the scope of the analysis. Additionally, long-term monitoring and security considerations were not thoroughly evaluated.

For future work, several directions can be pursued:

- Conducting experiments with diverse application types and varying workloads to assess performance under different scenarios.
- Evaluating security measures for containerized environments, including addressing potential vulnerabilities in Docker containers.
- Exploring additional tools and practices like Kubernetes

In conclusion, this research highlights the potential of containerization as a viable solution to mitigate vendor lock-in and improve the cost-effectiveness of cloud deployments. The proposed directions for future work aim to build on these findings and extend their applicability to broader and more complex use cases.

References

Agrawal, S. and Singh, D. (2024). Study containerization technologies like docker and kubernetes and their role in modern cloud deployments, *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, Pune, India, pp. 1–5.

Anon (n.d.). Containerization drives cost optimization in the cloud. [Online].
URL: *https://www.softobotics.com/blogs/how-containerization-drives-cost-optimization-in-the-cloud*

Dang, M. (2022). How containerization makes cloud migration services possible at a low cost. [Online] Trigent.
URL: *https://trigent.com/blog/how-containerization-is-reducing-migration-costs*

Fleck, J., Sorgalla, J., Katzenberg, F. and Sachweh, S. (2023). A containerized template approach for vendor-friendly smart home integration, *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Dortmund, Germany, pp. 352–355.

Lohumi, Y., Srivastava, P., Gangodkar, D. and Tripathi, V. (2023). Recent trends, issues and challenges in container and vm migration, *2023 International Conference on Computer Science and Emerging Technologies (CSET)*, Bangalore, India, pp. 1–5.

Mo, D. Cordingly, R., Chinn, D. and Lloyd, W. (2023). Addressing serverless computing vendor lock-in through cloud service abstraction, *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Naples, Italy, pp. 193–199.

Pellegrini, R., Rottmann, P. and Strieder, G. (2017). Preventing vendor lock-ins via an interoperable multi-cloud deployment approach, *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, Cambridge, UK, pp. 382–387.

Ramesan, R. (2023). A guide to kubernetes and cloud migration to cut costs and maximize benefits. [Online] CloudControl.
URL: *https://www.ecloudcontrol.com/a-guide-to-kubernetes-and-cloud-migration-to-cut-costs-and-maximize-benefits*

Rovnyagin, M. M., Sinelnikov, D. M., Varykhanov, S. S., Magazov, T. R., Kiamov, A. A. and Shirokikh, T. A. (2023). Intelligent docker container orchestration for low scheduling latency and fast migration in paas, *2023 Seminar on Information Computing and Processing (ICP)*, Saint Petersburg, Russian Federation, pp. 181–185.

Waseem, M., Ahmad, A., Liang, P., Akbar, M., Khan, A., Ahmad, I., Setälä, M. and Mikkonen, T. (2024). Containerization in multi-cloud environment: Roles, strategies, challenges, and solutions for effective implementation, *arXiv* .

Weldemicheal and Tesfaldet (2023). *Vendor lock-in and its impact on cloud computing migration*, PhD thesis, Jönköping University.
URL: *https://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-62090*