**Module Title**: Network Systems and Administration CA 2024  
**Module Code**: B9IS121  
**Module Instructor**: Kingsley Ibomo / Dr. Obinna Izima  
**Assessment Title**: Automated Container Deployment and Administration in the Cloud  
**Assessment Number**: 1  
**Assessment Type**: Practical (lab-based)  
**Restrictions on Time/Length**: N/A  
**Individual/Group**: Group  
**Assessment Weighting**: 60%  
**Planned Feedback/Results Release Date**: Within three weeks of submission  
**Mode of Submission**: Online (Moodle)

# Summary

The objective of this work was to develop a process of automatically deploying the given Docker container into an IaaS cloud infrastructure in order to explore how one can consistently set up, configure, and deploy a sample application in a generic and repeatable manner. Aim focused on provisioning the infrastructure in AWS, setting up environment own the Docker, and finally on having or creating continuous integration and deployment. To achieve this, we deployed the infrastructure using Terraform, configured and deployed the applications using Ansible and implemented Continuous Integration with Continuous Deployment (CI-CD) to Azure DevOps. During the development of this project, issues such as working with networks or installing various dependencies for smooth running and setting proper permission for deployment were met to be solved through scripting and tuning of the tools.

The solution was to design a fully automated system which could build, push and deploy Docker containers on its own without much human interaction. Some of the things we establish are the significance of using structured automation pipeline to improve the means as well as stability of deployment and also, the way tools like Ansible and Terraform are useful in ensuring reliability of infrastructure and applications. The enactment of infrastructure-as-code as well as CI/CD processes presented in this project shows how to obtain replicable and efficient practices for orchestration and deployment that are well-aligned with modern cloud environments.

**Table of Contents**

# Introduction

This report presents an overview of the work performed, as well as results, of a project based on automation of cloud infrastructure provisioning and server configuration with the help of Terraform, Ansible, Docker and Azure DevOps. In today’s world of cloud computing analogy where deployment automation is a major factor, the goals of the project were aimed at creating an easy, quick and efficiently deployable sample application in the Docker container format. Automation tools were selected for their ability to streamline these processes: To provision the cloud resources I used Terraform, to configure Network and servers I used Ansible, I used Docker to package the application and I used Azure DevOps both for CI for Continuous Integration as well as CD for Continuous Delivery. These tools made the deployment very predictable and minimised manual interference; making use of automation as a key way of achieving scalability, resiliency and consistency in cloud platform. The report describes each of these tasks, the approaches used to solve project needs, and the solutions used to tackle issues in developing the efficient, automated deployment pipeline.

# Infrastructure Setup

The cloud infrastructure environment focus was to deliver a strong environment for a hosting of the application server with code options for provisioning resources. To achieve the required flexibility and repeatability of infrastructure components deployment on AWS, the infrastructure-as-code tool Terraform was employed to provision and manipulate the required AWS cloud resources. This choice allowed for exerting accurate control over the configurations, provided the resources’ inter-consistency, as well as collaboration to work with the version-controlled script.

The setup started with the creation of an Amazon VPC which provided a virtual network on which the project resources were to be allocated. In this VPC, a public subnet was created so that the EC2 instance hosted here could be used as the main application server. An Internet Gateway was also included for connection to other networks and route tables for subnet and gateway traffic. These configurations were defined and provided the scripts in a well-structured and reusable form using Terraform hence making it easy to deploy and modify for future scaling of the application.

Security measures had taken a central position during the setting up of the work settings security. Security groups are set up for enabling specific inbound and outbound connections with the instance, such as enabling SSH connection type on port 22, and HTTP/HTTPS connection types on ports 80 and 443, but still provide enough liberty for future application requirements. On the same note, there were IAM roles and policies in the Terraform configuration to give the right access to managing the infrastructure and automate the deployment processes. All the scripts applicable to the Terraform are provided in the appendix and can be retrieved in the GitHub following the link provided at the beginning of the paper, and they give a clear understanding of the process of infrastructures setup as well as the documentation to follow in the future.

# Configuration Management

As far as the server was concerned the configuration management was performed with the help of Ansible, a tool that helps in organizing the installation of software as well as the setting up of systems. They selected Ansible because of basic setup, idempotent functionality so that configuration can be run multiple times without changing state, compatibility with many environments and the fact that it is really easy to use them as quite effective tools for the daily setup of servers. In this particular playbook, I used the task to update the server’s package manager, install docker and make docker daemon to start on boot. This made the server’ configuration to be well checked for any bad setting when we automated these tasks to ensure that setting ‘up the server was well done without much interference from man.

The Ansible playbook written also had some specific steps that were included in the Docker installation and the prerequisites to set, it also ensured the Docker service to be active. To make Docker operational when the system boots up, the playbook used the systemd module to automate enabling and starting Docker as a serviced-on boot. This configuration made Docker start right from the moment the server was on and the containers to come up without needing to be told to do so.

An information file in the form of a README file is included with the Ansible playbook and guides the user on the next steps, the expected results as well as any possible pitfalls to be expected. This documentation reconciles how many tasks are involved in the automation process, what each task entails and how to modify the playbook depending on the environment or distribution of the hypervisor. The README guarantees that anyone planning to replicate or alter the method has clear, precise instructions to do so efficiently.

# Docker Container Deployment

Containerization of a sample application and deploying Docker container implied auto-deployment to cloud platforms. The first step to containerization was to provide Dockerfile that lists down the dependencies, environment variables and command to run the application. This Dockerfile contained commands to fetch a base image, install suitable packages, and to move files that make up the application into the docker container. This form of containerization has made it possible to deploy an application in a lighter mobile environment that can be easily ported across the different infrastructures.

To host the Docker container on the cloud infrastructure automation scripts were employed to minimize the involvement of hands-on procedures. An Ansible playbook was used to manage the configuration and deployment process which to install Docker and ensure that it is properly on the cloud server. The playbook then downloaded the Docker image from the repository to launch the container and check that the sample application was available. This automatized deployment enabled an easy way of configuring the application on the cloud in such a way that it can be redeployed in light of changed code or new infrastructure. It became clear that the Dockerfile for containerizing the application was combined with Ansible to provide a fast, scalable and cloud-friendly solution.

# CI/CD Pipeline Integration

CI/DC pipeline was deployed through the use of Azure DevOps to help in the building as well as deployment process. Azure DevOps was an integrated solution for having version control build and deployment pipeline which facilitates easy direct deployment in response to code changes. It was set to activate whenever there was a new code push on the GitHub repository where the pipeline was located so that changes were built and deployed to the cloud infrastructure correctly.

While defining the pipeline it focused on to build docker image, to push it on container registry and to deploy it in cloud environment. This process was initiated by Azure DevOps where the code was fetched from GitHub for constructing a Docker image by referencing the Dockerfile. From here the image was then exported to the Azure Container Registry to be ready for deployment. During the deployment stage, Azure DevOps employed defined service connections to perform authentication into the Azure cloud platform to upload the container to an Azure Web App environment.

This CI/CD configuration made continuous delivery possible, so updates could be printed without eliciting human responses. While utilizing Azure DevOps to perform the deployment, it was also frequently tested and then monitored for effectiveness not allowing many possible errors to creep in and make the deployment less reliable.

# Documentation & Reflection

This automation project that was done for implementing Docker container into cloud infrastructure brought experience of infrastructure management, managing configuration, applicative containerization, and CI/CD integration. This end – to – end approach showed the overall picture of automating cloud infrastructure with Terraform, configuring the server with Ansible, and handling the deployment with Docker, and utilizing Azure DevOps. Every stage was interesting, and there were difficulties, for example, in the setup of the network using Terraform, the setup of different distributions of OS in Ansible because it requires conditional setup, and to find an appropriate method of secure authentication of users and keys within the frameworks of CI/CD pipeline.

The dozes learnt also pointed out at the necessity of compatibility of the tools used in the processes. Another task that was essential to fully automate the environment was to ensure that the installation and configuration of most of the infrastructure on board were consistent and compatible with a number of environments. For example, the use of conditional scripts in Ansible for Amazon Linux and Ubuntu to respond to conditions stopped system errors that occurred during Docker installation and thus contributed to deployment reliability. Moreover, the usage of CI/CD technology, which connects GitHub with Azure DevOps, highlighted the necessity to provide security for connections to other external services like Docker Hub or Azure Container Registry using Azure DevOps service connection and the storage of credentials.

Future implementations could be improved by several related approaches. For a tool to manage multiple containerized applications across instances, on demand at that, a tool like Kubernetes that is specifically designed as a container orchestration engine would add scalability and fault tolerance. Alternatively, the approach is serverless functions as Azure Functions or AWS Lambda which could not require constant server oversight. Recommendations include sending errors in the pipeline as soon as possible to quickly find out what went wrong and creating better documentation to help onboard new team members quickly. From this project, many learned the potential automation tools hold in developing reliable and efficient large-scale processes with significant opportunities for improvement and improvement of automation methods in future operational processes.

# Conclusions

To sum up, the project was accomplished and it applied tools as Terraform, Ansible, Docker, and Azure DevOps in order to initiate the Docker container that runs sample application on the cloud environment. This entailed initiating the cloud services, assigning the server, configuring the container and setting up a continuous integration and deployment environment. Some of the issues that were faced in this include the configuration of the network, compatibility of system in different environment and integration of multiple services with robust security features Though these challenges where encountered, they were well managed through proper planning and effective use of automation tools.

Some of the things that Organization can learn are the cross-tool compatibility, that automation removes transitional work, and that secure integration of services in CI/CD pipeline is necessary. All the same the project managed to achieve the intended goal of automating the deployment process of cloud infrastructure hence proving the fact that cloud infrastructure can be managed and scaled in case adequate tools are put in place.

# References

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2020). The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*, *35*, 63-75.

Srirama, S. N., Adhikari, M., & Paul, S. (2020). Application deployment using containers with auto-scaling for microservices in cloud environment. *Journal of Network and Computer Applications*, *160*, 102629.

Dittakavi, R. S. S. (2022). Evaluating the efficiency and limitations of configuration strategies in hybrid cloud environments. *International Journal of Intelligent Automation and Computing*, *5*(2), 29-45.

Kanth, T. C. (2023). EFFICIENT STRATEGIES FOR SEAMLESS CLOUD MIGRATIONS USING ADVANCED DEPLOYMENT AUTOMATIONS.