VIETNAM NATIONAL UNIVERSITY HOCHIMINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF INFORMATION SYSTEMS

TRẦN THỊ MỸ HUYỀN

ASSIGNMENT REPORT

WEB PROJECT MANAGEMENT IN CLOUD ENVIRONMENT

CLASS: IS402.O22.HTCL

HO CHI MINH CITY, 2024

VIETNAM NATIONAL UNIVERSITY HOCHIMINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF INFORMATION SYSTEMS

TRẦN THỊ MỸ HUYỀN – 21520269

ASSIGNMENT REPORT

WEB PROJECT MANAGEMENT IN CLOUD ENVIRONMENT

CLASS: IS402.O22.HTCL

PROJECT ADVISOR

MSc. Hà Lê Hoài Trung

HO CHI MINH CITY, 2024

TABLES OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Theoretical Foundation and Problem

1.1 Problem set

The group of tasks involves various aspects of processing and handling data, with processing being the main focus:

Storage:

- Azure Container Registry: Used for storing and managing Docker container images securely and efficiently, providing a central place to store and maintain container images for deployment.
- Azure Key Vault: Designed to securely store and access sensitive information such as API keys, passwords, certificates, and cryptographic keys, ensuring that sensitive data is protected and managed with high availability.

Collection:

- SonarQube: Provides continuous inspection of code quality, performing automatic reviews to detect bugs, code smells, and security vulnerabilities in codebases, helping maintain high code standards.
- Snyk: Specializes in software composition analysis (SCA) to identify and fix vulnerabilities in open-source dependencies, ensuring that the third-party components used in applications are secure.
- Trivy: An open-source vulnerability scanner for container images, file systems, and Git repositories, ensuring that containers are secure and free of known vulnerabilities before deployment.

Visualization:

- Grafana/Prometheus: Used together for powerful monitoring and visualization. Prometheus collects and stores metrics, while Grafana provides a flexible and visually appealing dashboard for analyzing and visualizing the collected data.
- Azure Monitor: A comprehensive monitoring solution that collects, analyzes, and acts on telemetry data from cloud and on-premises environments, providing insights into application and resource performance.

Processing (the main focus):

- Jenkins: An open-source automation server that facilitates CI/CD, enabling continuous integration and continuous delivery of software projects. It automates the building, testing, and deployment of applications.
- Docker: Provides containerization technology to package applications and their dependencies into containers, ensuring consistent environments across development, testing, and production.
- Azure Kubernetes Services (AKS): Manages containerized applications with Kubernetes, offering automated deployment, scaling, and operations of containerized applications, making it easier to manage clusters.
- Terraform: An infrastructure as code (IaC) tool that allows for the definition and provisioning of infrastructure using a declarative configuration language, enabling consistent and repeatable infrastructure management.

These tools and services work together to create a robust system for managing and processing web applications, ensuring efficient, secure, and scalable operations across the entire development and deployment lifecycle.

1.2 Data type

The group of tasks includes handling various types of content in detail:

- **Source Code**: This includes programming and configuration files such as Java source code, YAML configuration files, and Dockerfiles for containerization.
- **Build Products**: These are the results of the build process, including JAR files for Java applications and Docker images for containerized deployments.
- **Configuration Files:** Essential for deployment and automation, this category includes Kubernetes manifests for container orchestration, Terraform scripts for infrastructure as code, and Jenkinsfiles for defining CI/CD pipelines.
- **Security and Quality Reports**: To ensure code and container security, reports from tools like Snyk (for software composition analysis), SonarQube (for code quality), and Trivy (for container security) are utilized.

- **Monitoring Data**: For operational insights and troubleshooting, this includes logs, performance metrics, and alerts generated by the system.
- **Encryption Keys and Secrets**: To maintain security, this involves managing sensitive information such as encryption keys, secret texts, and certificates.

These elements are crucial for building, configuring, securing, and monitoring the system effectively, ensuring robust and reliable operations.

1.3 Comparison

Aspect	Cloud-based Systems	Traditional Systems
Scalability	Offer unparalleled scalability with easy resource scaling up or down based on demand.	Require significant upfront investment in hardware to handle peak loads, which often remain underutilized during off-peak times.
Cost	Follow a pay-as-you-go model, reducing capital expenditures and offering predictable operational costs.	Involve substantial initial costs for hardware setup and ongoing costs for maintenance, power, and cooling.
Maintenance	Cloud providers handle maintenance tasks, including hardware updates, patch management, and backups.	Require in-house teams to manage all aspects of maintenance, which can be resource-intensive and costly.
Security	Invest heavily in security measures with data encryption, compliance certifications, and advanced threat detection systems.	Achieving the same level of security requires significant investment in technology and expertise, and may be more vulnerable to physical threats.

Flexibility	Provide high flexibility with integration support for various services and platforms, facilitating a DevOps culture.	Often face limitations due to hardware constraints and require extensive manual configuration.
Accessibility	Allow access to resources and management from anywhere with an internet connection, supporting remote work and global collaboration.	Typically require physical presence or complex VPN setups to access resources remotely.
Performance	Offer globally distributed data centers to reduce latency and improve performance for users worldwide.	Limited by physical location and can experience higher latency for distant users.

Table 1: Cloud-based vs Traditional System

1.4 Components:

1.4.1 Infrastructure as a Service (IaaS)

Virtual Machines

- Azure Virtual Machines (VMs): These are scalable compute resources that provide flexibility and control over your infrastructure. You can deploy and manage these VMs to run your applications, services, or databases. They can be customized with various operating systems (Windows, Linux) and configurations based on your needs.
- Azure Kubernetes Service (AKS) - Partially Managed: In this setup, Azure manages the Kubernetes control plane (including the API server and other core components), but you are responsible for managing and maintaining the worker nodes and the overall infrastructure. This gives you more control and flexibility

while leveraging Azure's capabilities to handle the more complex aspects of Kubernetes management.

1.4.2 Platform as a Service (PaaS)

- AKS (Fully Managed): In this configuration, Azure handles the complete lifecycle management of the Kubernetes clusters, including updates, scaling, and monitoring. This allows you to focus on deploying and managing your applications without worrying about the underlying infrastructure.
- Azure Container Registry (ACR): A managed Docker container registry service used for storing and managing container images. It integrates seamlessly with Azure Kubernetes Service (AKS), allowing you to build, store, and deploy container images efficiently.
- Azure Key Vault: A cloud service for securely storing and accessing secrets, such as API keys, passwords, certificates, and cryptographic keys. It helps you control access and manage the lifecycle of sensitive information with ease.

1.4.3 Software as a Service (SaaS)

- Azure Monitor: A comprehensive monitoring service that provides full-stack visibility across your applications and infrastructure. It offers features like metrics collection, log analytics, and alerts, helping you to diagnose issues, gain insights, and maintain the health of your environment.
- Grafana: An open-source platform for monitoring and observability, which can be used for visualizing time-series data and creating dashboards. If hosted as a service, Grafana provides managed infrastructure, simplifying deployment and management.
- Prometheus: An open-source systems monitoring and alerting toolkit designed for reliability and scalability. If hosted as a service, Prometheus offers a managed solution for collecting and querying metrics, often integrated with Grafana for visualization.

- Snyk (If Using Managed Service): A security platform designed for developers, helping to find and fix vulnerabilities in code, dependencies, containers, and infrastructure as code (IaC). When used as a managed service, Snyk provides automated security monitoring, integration with CI/CD pipelines, and compliance reporting.

1.4.4 Applications/Tools Installed on Cloud

- Terraform: An open-source infrastructure as code (IaC) tool that allows you to define and provision infrastructure using a high-level configuration language. It enables you to automate the creation, modification, and management of resources across various cloud providers.
- SonarQube: A tool for continuous inspection of code quality, detecting bugs, vulnerabilities, and code smells. It integrates with various CI/CD pipelines to provide ongoing code analysis and maintain code quality standards.
- Jenkins: An open-source automation server that helps to automate parts of the software development process, such as building, testing, and deploying code. It supports continuous integration and continuous delivery (CI/CD), making it a crucial tool for DevOps practices.

Chapter 2: **Theoretical Basis**

2.1 About Datatype

2.1.1 Source Code

– Containing the code and configuration settings required for your applications and services.

- Java Source Files (.java): These files contain the Java programming language source code. Each .java file typically defines a single class or interface and can include fields, methods, and other elements needed for the functionality of the application.
- YAML Configuration Files (.yaml or .yml): YAML files are used for configuration purposes, often in a human-readable format. They are commonly used to configure settings for applications, services, and orchestration tools like Kubernetes.
- Dockerfile: A Dockerfile is a text document that contains all the commands needed to assemble an image. This file defines the environment in which your application will run, specifying base images, dependencies, configuration steps, and commands to be executed inside the container.
- Jenkinsfile: A Jenkinsfile is a script that defines the CI/CD pipeline for Jenkins. It outlines the steps for building, testing, and deploying applications, using a declarative or scripted syntax to automate these processes.

2.1.2 Build Artifacts

– The resulting files from the process of compiling and packaging the source code. They are typically in binary format, ready for deployment and execution.

- Java Archive Files (.jar): JAR files are the compiled and packaged form of Java applications. They contain compiled Java classes and associated

metadata and resources (such as text, images) needed for the application to run.

- Docker Images: Docker images are lightweight, standalone, and executable software packages that include everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files. These images are used to create Docker containers, which can be deployed and run on any Docker-enabled system.

2.1.3 Configuration Files

– Containing information needed to configure resources and infrastructure. They are essential for defining the setup and behavior of various components in your environment.

- Kubernetes Manifest Files (.yaml): These YAML files are used to define the configuration for Kubernetes resources such as Pods, Deployments, Services, and ConfigMaps. They specify the desired state of the resources and how they should be managed within the Kubernetes cluster.
- Terraform Configuration Files (.tf): These files are used to define and manage infrastructure as code. Terraform scripts describe the desired state of cloud and on-premises infrastructure using a high-level configuration language. These scripts can provision, update, and version the infrastructure efficiently.

2.1.4 Security and Quality Reports

– These reports provide insights into the security posture and code quality of the applications. They include detailed analyses and findings that help in identifying and addressing issues.

- Snyk Security Reports: Indentifying and detailing vulnerabilities in the code, dependencies, containers, and infrastructure as code. Snyk provides recommendations and automated fixes to remediate these vulnerabilities, ensuring the security of the applications.

- SonarQube Quality Reports: Analyzing the source code for quality issues, including bugs, code smells, and security vulnerabilities. SonarQube provides a comprehensive overview of code quality, with metrics and actionable insights to improve code health.
- Trivy Vulnerability Reports: From scanning container images for security vulnerabilities. Trivy provides detailed information about discovered vulnerabilities, including severity, affected components, and remediation steps.

2.1.5 Monitoring Data

– Data includes performance metrics, logs, and alerts that provide insights into the health and status of the system. It is crucial for maintaining the reliability and efficiency of the infrastructure.

- Log Files: These files contain detailed records of system activity. Logs are used to track events, errors, and interactions within the system, aiding in troubleshooting and audit processes.
- Performance Metrics: These metrics measure the performance and resource utilization of the system. Metrics can include CPU usage, memory usage, network throughput, and application-specific indicators. They are often stored in databases for aggregation and analysis.
- Alert Information: Alerts notify about incidents or abnormal conditions in the system. They are generated based on predefined thresholds and rules, helping to promptly address potential issues.

2.1.6 Secret Management

– Azure Key Vault: A cloud service for securely storing and managing secrets, encryption keys, and certificates. It provides centralized secret management, enabling secure access and control over sensitive information. Key Vault integrates with other Azure services and helps in maintaining security compliance.

2.2 About Processing Algorithms

2.2.1 IaaS

Virtualization Algorithms

- Azure uses Hyper-V, a native hypervisor developed by Microsoft, for its virtualization. Hyper-V allows the creation of virtual machines on x86-64 systems running Windows. The hypervisor is responsible for allocating physical resources such as CPU, memory, and I/O devices to VMs. It uses advanced scheduling algorithms to ensure optimal performance and isolation between VMs.

Snapshot and Clone Algorithms

- Azure VMs can create snapshots, which are point-in-time backups of the virtual hard disks. The snapshot algorithm captures the state of a VM's disk at a specific time, allowing users to revert back to this state if needed. This is particularly useful for backup and disaster recovery scenarios. Snapshots in Azure are managed using the Azure Resource Manager, which ensures consistency and efficient storage management.
- Azure uses managed disks to handle storage for VMs. Managed disks automatically handle the storage account's scalability, performance, and availability. When creating snapshots or cloning disks, Azure's algorithms ensure data integrity and high availability.

Load Balancing Algorithms

- Azure Load Balancer distributes incoming traffic among multiple VMs to ensure no single VM becomes a bottleneck. The load balancing algorithm uses a hash-based distribution, taking into account factors like source IP address, destination IP address, source port, and destination port. This ensures that traffic is evenly distributed, providing high availability and reliability.
- This provides more advanced load balancing, including SSL termination, cookie-based session affinity, and URL-based routing. The Application

Gateway uses round-robin and least connections algorithms to distribute traffic effectively among backend servers.

Auto-scaling Algorithms

- Azure VM Scale Sets allow you to automatically increase or decrease the number of VM instances based on demand. The auto-scaling algorithm monitors metrics like CPU usage, memory usage, and custom metrics to determine when to scale in or out. This helps ensure that applications remain responsive under varying loads and helps manage costs by scaling down when demand decreases.

Security Algorithms

- Azure Security Center integrates with Azure VMs to provide continuous security assessments. Algorithms in Azure Security Center scan VMs for vulnerabilities, recommend security improvements, and help in detecting threats. The vulnerability scanning algorithm identifies outdated software, misconfigurations, and potential security gaps, providing actionable insights to improve the VM's security posture.

Disaster Recovery Algorithms

- Azure Site Recovery provides disaster recovery as a service (DRaaS) for Azure VMs. It uses replication algorithms to continuously replicate VMs to a secondary location. In the event of a failure, the failover algorithm ensures that the VM can be quickly restored from the replicated data, minimizing downtime and data loss.

Scheduling Algorithm

- Kubernetes Scheduler uses a scheduling algorithm to allocate Pods to Nodes based on available resources and the Pod's requirements. Factors like CPU, memory, and node constraints are considered to ensure that Pods are efficiently allocated and load is balanced among Nodes.
- Horizontal Pod Autoscaler (HPA) automatically adjusts the number of Pods in a Deployment or ReplicaSet based on metrics such as CPU usage or custom

metrics. The HPA algorithm continuously monitors workload and adjusts the number of Pods to meet resource demands.

- Cluster Autoscaler automatically adjusts the number of Nodes in a cluster based on workload demand. When Pods cannot be scheduled due to insufficient resources, the Cluster Autoscaler adds new Nodes; conversely, when Nodes are no longer needed, it removes surplus Nodes to save resources.

2.2.2 PaaS

Container Image Management Algorithms

- Tagging uses algorithms to assign labels (tags) to versions of container images. This helps easily manage and retrieve different versions of container images.
- Garbage Collection is an algorithm that automatically deletes unused container images to free up storage space. This algorithm identifies unlinked images and safely removes them.

Security, Encryption Algorithms

- Vulnerability Scanning uses algorithms to scan and detect security vulnerabilities in container images. Tools like Azure Security Center integrate with Azure Container Registry to scan images and provide detailed reports on detected vulnerabilities.
- Azure Key Vault employs encryption algorithms to secure secrets before storing them. These algorithms utilize hardware security modules (HSMs) to provide robust encryption, ensuring that data is protected against unauthorized access.
- The retrieval process uses secure API calls to access secrets. These algorithms ensure that only authenticated and authorized users or applications can retrieve secrets, maintaining data confidentiality and integrity.
- RBAC algorithms manage permissions for users and applications. These algorithms assign roles with specific access rights, controlling who can read, write, or delete secrets within the vault.

- Azure Key Vault supports secret versioning, allowing multiple versions of a secret to be maintained. Version control algorithms track changes to secrets, enabling users to revert to previous versions if necessary.
- Auditing and Monitoring
- Azure Key Vault incorporates algorithms that log and monitor access and operations performed on secrets. These algorithms facilitate security audits and compliance by providing detailed records of who accessed or modified the secrets and when.

2.2.3 SaaS and Others

Log Analysis Algorithms

- Azure Monitor leverages machine learning algorithms to analyze and predict issues from log data. For example, anomaly detection algorithms identify patterns in log data that deviate from the norm, helping to detect potential issues before they become critical. These algorithms continuously learn from the incoming data to improve their accuracy over time.
- Azure Monitor uses aggregation algorithms to collect and store monitoring data from various sources. These algorithms ensure that data from different services and applications is aggregated efficiently, providing a unified view of the system's health and performance. The aggregated data is then stored in a scalable and resilient manner to support real-time analysis and long-term retention.

Data Storage and Query Algorithms

- Prometheus uses a time-series database (TSDB) to store monitoring data. The storage engine uses algorithms optimized for writing high-frequency data and querying it efficiently. It employs a custom, append-only storage format that minimizes disk I/O and supports high read/write throughput.
- Prometheus's query language, PromQL, uses algorithms that optimize the retrieval of time-series data. These algorithms support complex queries, aggregations, and functions to analyze the collected metrics in real-time.

- Prometheus supports threshold-based alerting, where alerts are triggered based on predefined thresholds for specific metrics. The alerting algorithm continuously evaluates the metrics against the thresholds and generates alerts when the conditions are met.
- Prometheus also supports rule-based alerting, allowing users to define custom rules for alerting based on multiple conditions. These algorithms evaluate the rules periodically and trigger alerts when the conditions are satisfied.

Source Code Analysis Algorithms

- SonarQube uses static analysis algorithms to analyze source code and detect bugs, code smells, and security vulnerabilities. One key technique used is the Abstract Syntax Tree (AST) analysis, which represents the code's structure and allows the detection of patterns that indicate potential issues.
- SonarQube employs algorithms to evaluate and rank the quality of source code based on predefined metrics. These metrics include code coverage, duplication, complexity, and maintainability. The quality gate algorithm aggregates these metrics to provide an overall quality assessment, helping teams identify areas that need improvement.

Vulnerability Scanning Algorithms

- Static Analysis and Dependency Scanning: Snyk uses static analysis algorithms to scan the source code and its dependencies for known vulnerabilities. These algorithms compare the code and dependencies against a database of known vulnerabilities, identifying potential security risks.
- In addition to static analysis, Snyk uses dynamic analysis algorithms to monitor applications at runtime, identifying vulnerabilities that might not be evident from the code alone.

Automatic Fix Algorithms

- Snyk's algorithms can automatically generate patches and suggest fixes for detected vulnerabilities. These algorithms analyze the context of the vulnerability, propose code changes, and, in some cases, automatically apply

the fixes. This helps developers quickly remediate security issues without extensive manual intervention.

Terraform Process Algorithms

- Users define the desired infrastructure state using HashiCorp Configuration Language (HCL) or JSON. This declarative approach ensures a consistent and reproducible infrastructure setup.
- Terraform prepares the working environment by downloading provider plugins and configuring the backend for state storage. After that, it generates an execution plan by comparing the desired state with the current state. It builds a dependency graph to determine the correct order of operations and identifies the necessary actions to align the current infrastructure with the desired state.
- Terraform executes the actions specified in the execution plan to achieve the desired state. It performs API calls to provision, update, or destroy resources, using parallelism to improve efficiency when dependencies allow.
- Terraform maintains a state file to track the current state of the infrastructure, ensuring accurate tracking and planning of future changes.

Chapter 3: **Architecture**

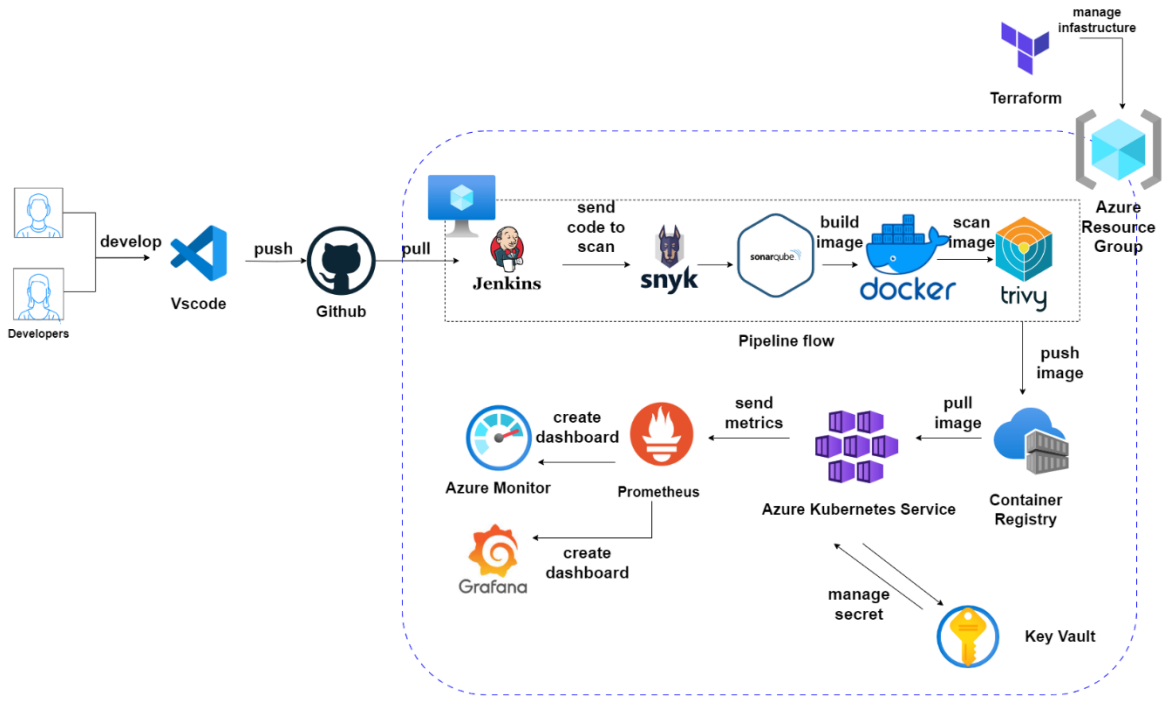

Figure 1: Solution Architecture

3.1 Components

- Developers

Role: Write and maintain code for applications.

- VS Code

Function: A popular integrated development environment (IDE) used by developers to write and edit code.

- GitHub

Function: A platform for version control and collaboration. Developers push their code to GitHub repositories.

- Jenkins

Function: A CI/CD tool that automates the building, testing, and deploying of applications. Jenkins pulls code from GitHub and orchestrates the pipeline flow.

- Snyk

Function: A security tool integrated into the CI/CD pipeline to scan code for vulnerabilities. Jenkins sends the code to Snyk for scanning.

- SonarQube

Function: A tool for continuous inspection of code quality to perform automatic reviews, detecting bugs, code smells, and security vulnerabilities. Jenkins sends code to SonarQube for analysis.

- Docker

Function: A platform for developing, shipping, and running applications in containers. Jenkins builds Docker images of the application code.

- Trivy

Function: An open-source vulnerability scanner for container images. Jenkins scans the Docker images using Trivy to detect vulnerabilities.

- Terraform

Function: An infrastructure as code (IaC) tool used to define and provision infrastructure using configuration files. Manages the infrastructure in the Azure Resource Group.

- Azure Resource Group

Function: A container for managing and grouping resources in Azure. Terraform manages the infrastructure within this group.

- Container Registry

Function: A service to store and manage container images. The pipeline pushes built Docker images to the Azure Container Registry.

- Azure Kubernetes Service (AKS)

Function: A managed Kubernetes service for running containerized applications. Pulls images from the Azure Container Registry and deploys them.

- Azure Key Vault

Function: A cloud service for securely storing and managing secrets, keys, and certificates. Manages secrets required by the applications running on AKS.

- Azure Monitor

Function: A comprehensive monitoring service that provides metrics and logs for the applications and infrastructure. Creates dashboards for monitoring.

- Prometheus

Function: An open-source systems monitoring and alerting toolkit. Collects metrics from the applications running on AKS and sends them to Azure Monitor and Grafana.

- Grafana

Function: An open-source platform for monitoring and observability. Creates dashboards for visualizing metrics collected by Prometheus.

- Summary

1. Development: Code is written in VS Code and pushed to GitHub.
2. CI/CD Pipeline: Jenkins pulls the code, and Snyk and SonarQube scan it for security and quality issues. Jenkins builds Docker images, which are scanned by Trivy.
3. Deployment: Built images are pushed to the Azure Container Registry and deployed to AKS.
4. Monitoring and Management: Prometheus collects metrics from AKS, which are visualized using Grafana and Azure Monitor. Terraform manages the infrastructure, and Azure Key Vault securely stores application secrets.

3.2 Read/Write Speed

In a modern DevOps architecture, read and write speeds are crucial factors that impact the overall performance of the system

1. VS Code

The read/write speed in Visual Studio Code depends on the performance of the hard drive and the capabilities of the developer's personal computer. SSDs typically offer faster speeds than HDDs.

2. GitHub

The read/write speed when pushing and pulling code from GitHub depends on network bandwidth and the performance of GitHub servers. GitHub provides high-performance code hosting services to minimize wait times during code synchronization.

3. Jenkins

Jenkins performs numerous read/write tasks such as fetching code from GitHub, storing build artifacts, and writing logs. Read/write performance can be enhanced by using SSDs on the Jenkins server and configuring proper caching.

4. Snyk and SonarQube

Both Snyk and SonarQube require reading source code and writing analysis results. Using high-performance servers and storage helps reduce scanning and analysis time.

5. Docker

Building and pushing Docker images require high read/write speeds. Using SSDs and high-speed networks will improve build and push/pull speeds for Docker images.

6. Trivy

Trivy reads Docker images and writes scan results. The speed of the disk and network impacts the scanning speed.

7. Azure Container Registry

Azure Container Registry is optimized to provide high read/write speeds when storing and retrieving Docker images. This ensures quick and efficient pulling of images to Kubernetes.

8. Azure Kubernetes Service (AKS)

AKS uses disks attached to nodes in the cluster. Using Azure Premium Storage or Azure Ultra Disk can improve read/write speeds for containerized applications.

9. Azure Key Vault

Azure Key Vault is designed to provide quick and secure access to secrets. This ensures that applications on AKS can access necessary secrets swiftly without compromising performance.

10. Azure Monitor and Prometheus

Both Azure Monitor and Prometheus require writing and reading a large amount of metric and log data. Using high-performance storage infrastructure helps maintain stable and fast read/write speeds.

11. Grafana

Grafana accesses and displays data from Prometheus and Azure Monitor. To ensure dashboards update quickly, high read/write performance from the data sources is necessary.

3.3 Delay Time When Set Up in Different Regions on Azure

- Deploying applications and infrastructure across different regions in Azure can lead to varying deployment times due to network latency, resource provisioning times, and data replication delays.

1. GitHub

- Generally minimal. However, pushing and pulling code might see slight delays (milliseconds to a few seconds) depending on the geographic distance between the developer's location and GitHub's servers.

2. Jenkins

- Pulling code from GitHub may introduce additional latency (a few seconds) if Jenkins servers are located in a different region.
- Storing build artifacts in remote regions can add delays ranging from a few seconds to minutes depending on the size of the artifacts and the network speed.

3. Snyk and SonarQube

- Network latency can cause delays of several seconds to minutes if Snyk and SonarQube servers are far from the Jenkins servers.
- Fetching scan results might add extra seconds to the process.

4. Docker and Trivy

- Building Docker images can be delayed by several seconds to minutes due to network latency when pushing the built images to a registry in a different region.
- Scanning images with Trivy may add similar delays if the scanner and the image registry are in different regions.

5. Azure Container Registry

- Pushing Docker images to an Azure Container Registry in a different region can take additional minutes, especially for large images.

- Pulling images from a remote registry can add similar delays, affecting the overall deployment time.

6. Azure Kubernetes Service (AKS)

- Setting up AKS clusters in a remote region can take extra time (up to several minutes) due to the need to provision resources in that specific region.
- Pulling Docker images from a registry in a different region can add additional minutes to the deployment process.

7. Azure Key Vault

- Accessing secrets from a Key Vault in a different region can introduce delays of a few milliseconds to seconds, depending on the frequency and size of the data being accessed.

8. Azure Monitor and Prometheus

- Collecting and sending metrics from AKS to monitoring services in different regions can add a few seconds to the data transfer time.
- Real-time alerts might be slightly delayed (seconds) due to the time taken for data to traverse regions.

9. Grafana

- Retrieving data for dashboards from Prometheus and Azure Monitor in different regions can introduce delays of a few seconds, affecting the real-time update of dashboards.

To minimize deployment delays across regions, consider the following strategies:

1. Choose Azure regions that are geographically closer to each other to reduce network latency.
2. Where possible, deploy resources in the same region to minimize cross-region data transfer.
3. Use Azure Traffic Manager to optimize traffic routing and reduce latency.
4. Ensure data is replicated across regions to avoid delays in accessing critical resources.

3.4 Optimization

To enhance the performance and efficiency of the depicted DevOps architecture, several optimization strategies can be implemented.

1. Geographical Optimization

- Ensure that critical components such as Jenkins, Azure Container Registry, AKS, and Azure Key Vault are deployed in the same or nearby Azure regions. This reduces network latency and speeds up data transfer.
- Use Azure Traffic Manager to optimize traffic routing and ensure that users and applications access the nearest available resources.

2. Infrastructure Optimization

- Equip servers (e.g., Jenkins, Docker build servers) with SSDs instead of HDDs to improve read/write speeds for build and deployment processes.
- Utilize Azure Premium Storage or Azure Ultra Disk for high-performance read/write operations, especially for AKS nodes and other critical components.

3. CI/CD Pipeline Optimization

- Configure Jenkins to run builds in parallel where possible, reducing overall build time.
- Implement caching mechanisms to store dependencies and intermediate build artifacts. Tools like Jenkins pipeline caching can significantly reduce build times by avoiding repeated downloads and compilations.
- Use incremental builds to compile only the parts of the application that have changed, rather than rebuilding the entire project from scratch.

4. Docker and Container Registry Optimization

- Optimize Dockerfile to leverage layer caching, which avoids rebuilding unchanged layers. This can drastically reduce build times.
- Regularly clean up and remove unused Docker images from the registry to optimize storage and reduce retrieval times.
- Use Content Delivery Network to cache and distribute Docker images across different regions, reducing latency for image pulls.

5. Security and Quality Scanning Optimization

- Integrate pre-scan stages in the local development environment to catch issues early before they reach the CI/CD pipeline.
- Use optimized settings in Snyk and SonarQube to balance thoroughness and speed of security and quality scans. Configure scans to focus on critical parts of the codebase first.

6. Monitoring and Logging Optimization

- Configure Prometheus to scrape metrics at optimal intervals to avoid overloading the system with excessive data collection.
- Use Azure Log Analytics to aggregate logs from different regions, enabling centralized and efficient log management.
- Set appropriate alerting thresholds in Prometheus and Azure Monitor to avoid alert fatigue and ensure timely response to critical issues.

7. Azure Key Vault Optimization

- Implement secret caching mechanisms to reduce frequent calls to Azure Key Vault, which can improve application performance.
- Fine-tune access policies to ensure minimal and necessary access, reducing overhead and potential security risks.

8. Optimizing Azure Kubernetes Service (AKS) with KEDA

- KEDA (Kubernetes Event-Driven Autoscaling) is a Kubernetes-based event-driven autoscaler that helps to scale applications based on the number of events needing to be processed. Integrating KEDA with Azure Kubernetes Service (AKS) can significantly optimize resource utilization and performance for event-driven workloads.

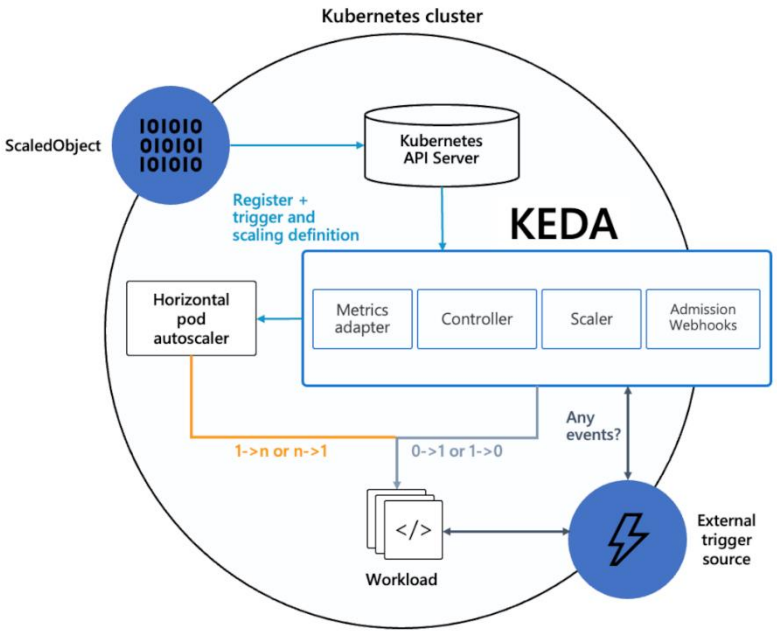

Figure 2: KEDA with K8s

- KEDA allows user to define autoscaling policies based on external events or metrics. This is particularly useful for applications that have variable workloads and need to scale dynamically in response to events such as messages in a queue, HTTP requests, or custom metrics.

Example Use Cases

1. Queue Processing: Automatically scale worker pods based on the length of an Azure Storage Queue.
2. HTTP Request Load: Scale web applications in response to HTTP request metrics collected by Azure Monitor.
3. Custom Metrics: Use custom application metrics exposed via Prometheus to drive scaling decisions.

Chapter 4: Deployment

4.1 Resource visualization

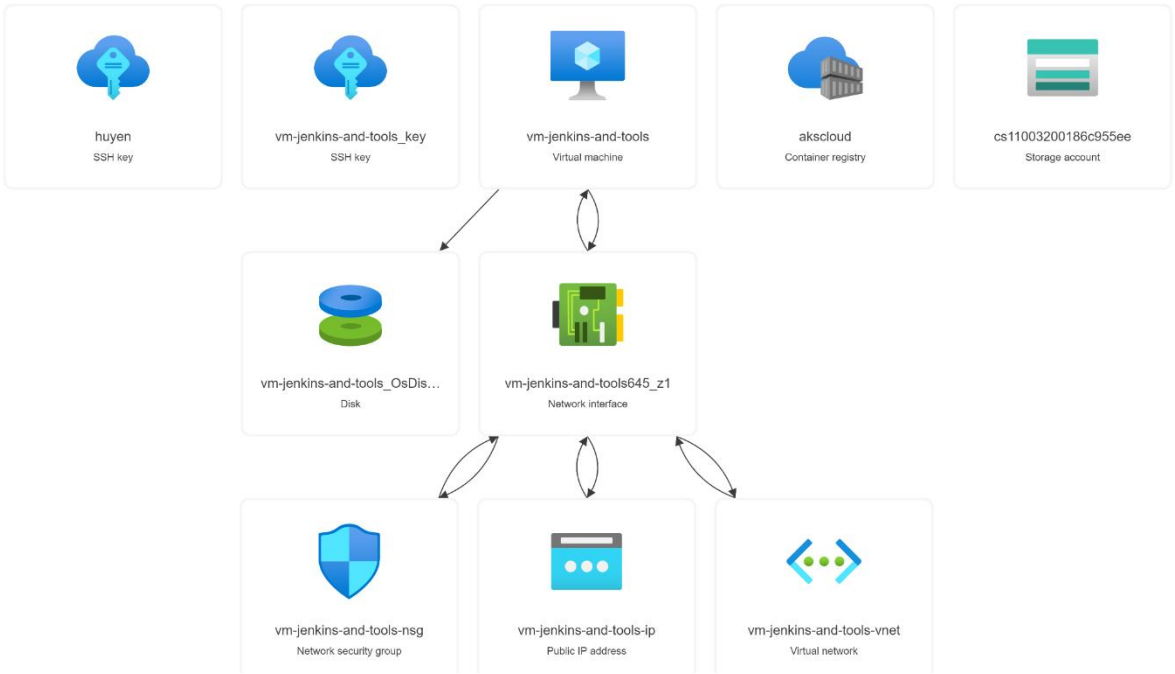

Figure 3: Resource visualization for Dev/Test environment

Components

1. huyen (SSH key)

Function: This SSH key is used for securely accessing the virtual machine (VM) remotely. It ensures that only authorized users can log in to the VM.

2. vm-jenkins-and-tools_key (SSH key)

Function: Another SSH key specifically generated for the vm-jenkins-and-tools VM. It provides secure access for users managing Jenkins and other tools on this VM.

3. vm-jenkins-and-tools (Virtual machine)

Function: This is the main virtual machine where Jenkins and other tools are installed. It serves as the core component for continuous integration and deployment tasks.

4. akscloud (Container registry)

Function: This Azure Container Registry stores Docker images. It is integrated with the CI/CD pipeline managed by Jenkins to store and retrieve container images used in deployments.

5. cs11003200186c955ee (Storage account)

Function: This Azure Storage Account provides blob, file, queue, and table storage services. It can be used to store build artifacts, logs, and other data required by the VM and applications running on it.

6. vm-jenkins-and-tools_OsDisk (Disk)

Function: This is the OS disk attached to the vm-jenkins-and-tools VM. It stores the operating system and all installed applications, including Jenkins and other tools.

7. vm-jenkins-and-tools645_z1 (Network interface)

Function: This network interface card (NIC) connects the vm-jenkins-and-tools VM to the virtual network. It manages the VM's network traffic, IP addresses, and security rules.

8. vm-jenkins-and-tools-nsg (Network security group)

Function: This Network Security Group (NSG) contains security rules that control incoming and outgoing traffic to the vm-jenkins-and-tools VM. It ensures that only allowed traffic can access the VM.

9. vm-jenkins-and-tools-ip (Public IP address)

Function: This public IP address is associated with the vm-jenkins-and-tools VM, allowing it to be accessible over the internet. This is crucial for remote access and external integrations.

10. vm-jenkins-and-tools-vnet (Virtual network)

Function: This virtual network (VNet) provides an isolated network environment for the VM. It includes subnets, IP address ranges, and network security configurations that define the network topology.

Interactions and Relationships

- SSH Keys (huyen and vm-jenkins-and-tools_key): These SSH keys are used to securely access the vm-jenkins-and-tools VM. Each key is uniquely associated with users or administrators managing the VM.

- vm-jenkins-and-tools VM: This VM is the central component where Jenkins and other tools are installed. It interacts with various other resources for its operations:
    1. Disk (vm-jenkins-and-tools_OsDisk): The OS disk is directly attached to the VM, storing its operating system and applications.
    2. Network Interface (vm-jenkins-and-tools645_z1): This NIC connects the VM to the virtual network and manages its network traffic.
    3. Public IP Address (vm-jenkins-and-tools-ip): The NIC is associated with this public IP, making the VM accessible from the internet.
    4. Network Security Group (vm-jenkins-and-tools-nsg): The NSG applies security rules to the NIC, controlling access to the VM.
- Azure Container Registry (akscloud): This registry stores Docker images that are used by applications running on the VM. Jenkins can push built images to this registry and pull them during deployments.
- Storage Account (cs11003200186c955ee): This storage account is used to store various data needed by the VM, including build artifacts and logs. The VM can interact with this storage account to read and write data.
- Virtual Network (vm-jenkins-and-tools-vnet): The VNet provides an isolated network environment for the VM and its components, ensuring secure and efficient communication within the network.

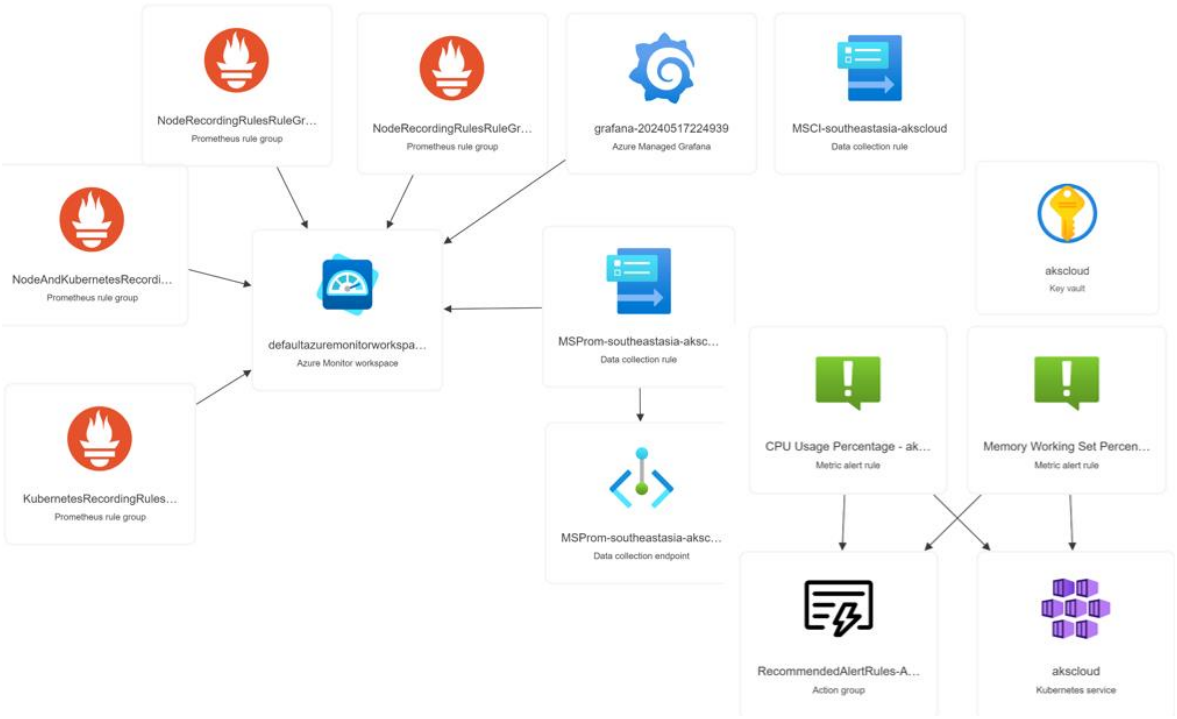

Figure 4: Resource visualization for Deployment environment

Components

1. defaultazuremonitorworkspace (Azure Monitor Workspace)

Function: This is the central workspace in Azure Monitor where metrics and logs are collected and analyzed. It aggregates data from different sources to provide a unified view of the system's performance and health.

2. Prometheus Rule Groups (NodeRecordingRulesRuleGroup, NodeAndKubernetesRecordingRules, KubernetesRecordingRules, etc.)

Function: These Prometheus rule groups define a set of recording and alerting rules. Recording rules precompute frequently needed queries and store the results as new time series, while alerting rules trigger alerts based on specified conditions.

3. Prometheus Data Collection Rule (MSCI-southeastasia-akscloud, Prom-southeastasia-akscloud)

Function: These rules define how data is collected from the Prometheus endpoints and sent to the Azure Monitor workspace. They ensure that metrics from the Kubernetes cluster are ingested into Azure Monitor.

4. Prometheus Data Collection Endpoint (Prom-southeastasia-akscloud)

Function: This endpoint represents the target from which Prometheus scrapes metrics. It serves as the data source for the Prometheus rule groups and data collection rules.

5. Metric Alert Rules (CPU Usage Percentage - akscloud, Memory Working Set Percentage - akscloud)

Function: These rules monitor specific metrics, such as CPU usage and memory working set percentage, and trigger alerts when the values exceed defined thresholds. This helps in identifying potential issues early.

6. Alert Group (RecommendedAlertRules-Akscloud)

Function: This group aggregates multiple alert rules and defines the actions to be taken when alerts are triggered, such as sending notifications or executing automated remediation scripts.

7. Grafana (grafana-20240517224939)

Function: Azure Managed Grafana is used to visualize the metrics and logs collected in Azure Monitor. It provides dashboards for monitoring the health and performance of the Kubernetes cluster.

8. Azure Key Vault (akscloud)

Function: Azure Key Vault securely stores secrets, keys, and certificates used by the applications running on the AKS cluster. It ensures secure access to sensitive information.

9. AKS Cluster (akscloud)

Function: Azure Kubernetes Service (AKS) provides a managed Kubernetes environment. It runs containerized applications and integrates with various Azure services for monitoring, security, and management.

Interactions and Relationships

- Prometheus Rule Groups: The Prometheus rule groups define the logic for recording and alerting based on metrics collected from the AKS cluster. These rules send data to the Azure Monitor workspace for aggregation and analysis.

- Azure Monitor Workspace: This workspace collects metrics and logs from Prometheus and other sources. It acts as the central repository for monitoring data, enabling analysis and alerting.
- Data Collection Rules and Endpoints: The data collection rules define how data is ingested from Prometheus endpoints into the Azure Monitor workspace. The Prometheus endpoints serve as the data sources, providing real-time metrics.
- Metric Alert Rules: These rules continuously monitor critical metrics such as CPU and memory usage. When thresholds are breached, the rules trigger alerts to notify administrators or take automated actions.
- Alert Group: The alert group consolidates multiple alert rules and specifies the actions to be taken when an alert is triggered. This ensures a coordinated response to potential issues.
- Grafana: Grafana accesses data from the Azure Monitor workspace to create visual dashboards. These dashboards provide insights into the performance and health of the Kubernetes cluster, making it easier to detect and troubleshoot issues.
- Azure Key Vault: Key Vault stores and manages secrets required by applications in the AKS cluster. It ensures secure access to these secrets, enhancing the security posture of the applications.
- AKS Cluster: The AKS cluster runs the containerized applications and integrates with Azure Monitor and Prometheus for monitoring. It relies on Azure Key Vault for secure secret management.

4.2 K8s architecture

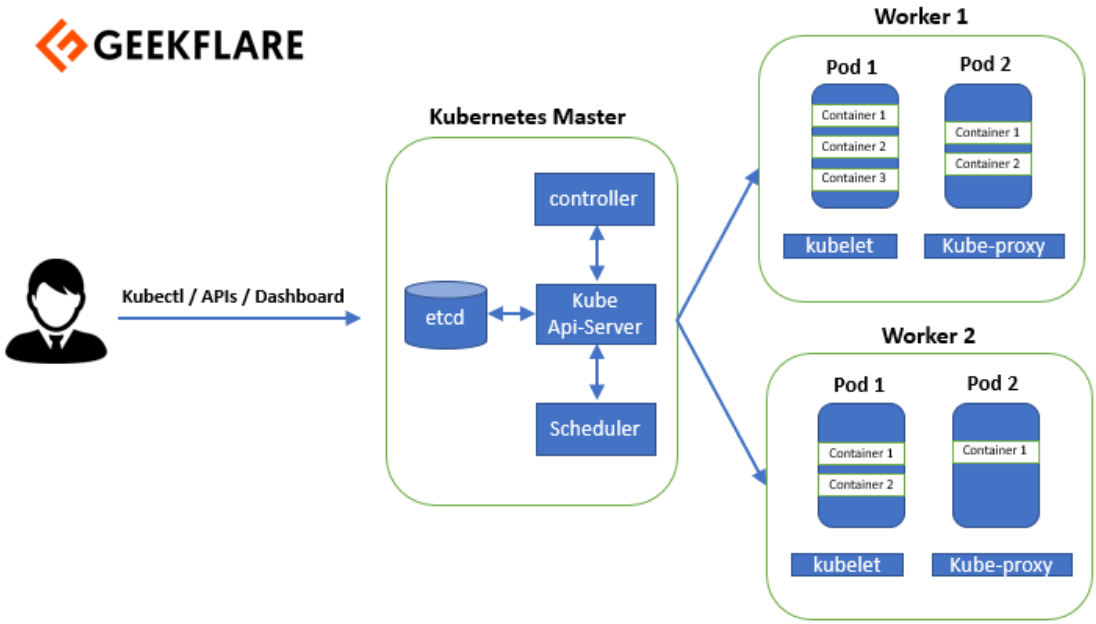

Figure 5: Kubernetes architecture

- Kubernetes follows master-slave architecture. Kubernetes architecture has a master node and worker nodes. There are four components of a master node.

Master Node Components

1. Kube API Server

Function: Serves as the front end for the Kubernetes control plane. It exposes the Kubernetes API and handles RESTful operations. It is the entry point for all administrative tasks and manages the state of the cluster.

2. Controller Manager

Function: Runs controllers that regulate the state of the cluster. Examples include the node controller, replication controller, and endpoint controller. It ensures that the desired state of the cluster matches the current state.

3. Scheduler

Function: Assigns pods to nodes based on resource availability. It considers factors like resource requirements, policy constraints, affinity and anti-affinity specifications, data locality, and more to optimize workload distribution.

4. etcd

Function: A distributed key-value store used for storing all cluster data. It holds configuration data, state information, and metadata about the cluster. It is essential for ensuring cluster consistency and reliability.

Worker Node Components

1. Kubelet

Function: An agent that runs on each worker node. It ensures that containers are running in a pod. Kubelet receives pod specifications from the API server and ensures the containers described in those pod specifications are running and healthy.

2. Kube-proxy

Function: Manages network communication inside and outside of the cluster. It maintains network rules on nodes and allows network communication to your pods from network sessions inside or outside of your cluster.

3. Container Runtime

Function: The software responsible for running containers. Examples include Docker, containerd, and CRI-O. It pulls container images from a registry, starts and stops containers, and manages container storage and networking.

4.3 Blue-Green deployments

- Blue-green deployment is a strategy in software development where two identical environments are maintained, with one serving as the live environment ("blue") and the other as a staging or testing environment ("green"). This method allows a team to switch traffic between two environments, ensuring that new versions of an application can be deployed and tested without impacting the live environment. Once testing is complete and the new version is ready to go live, traffic is switched from the blue environment to the green environment, making green the new production environment.

Benefits of Blue-Green Deployment in Kubernetes

- By having two identical production environments, one active and one idle, the switch can be made in an instant (often by changing a router's settings), thus minimizing downtime.

- If issues arise with the new version in the green environment, you can quickly revert back to the blue environment, reducing the impact on end users.
- Testing can be conducted in the green environment under the same conditions as the production environment without disrupting the live service.

Implementing Blue-Green Deployment in Kubernetes

Kubernetes, with its robust orchestration capabilities, is well-suited for implementing blue-green deployments. Here's how to set up a blue-green deployment strategy in Kubernetes:

1. Create two identical production environments within Kubernetes. This typically involves setting up two sets of resources (pods, services, etc.)—one for each environment.
2. Use Kubernetes services to manage the network traffic to the pods. Each environment should have its own service, and the service should be configured to route traffic to the appropriate set of pods.

```
apiVersion: apps/v1
kind: Deployment # Kubernetes resource kind we are creating
metadata:
  name: ekart-blue-deployment
spec:
  selector:
    matchLabels:
      app: ekart
      version: blue
  replicas: 3 # Number of replicas that will be created for this
deployment
  template:
    metadata:
      labels:
        app: ekart
        version: blue
    spec:
      containers:
        - name: ekart
          image: akscloud.azurecr.io/shopping-cart:latest # Image
that will be used to containers in the cluster
          imagePullPolicy: Always
          ports:
```

```
            - containerPort: 8070 # The port that the container is
running on in the cluster
```

Figure 6: Blue Deployment (blue.yaml)

```
apiVersion: apps/v1
kind: Deployment # Kubernetes resource kind we are creating
metadata:
  name: ekart-green-deployment
spec:
  selector:
    matchLabels:
      app: ekart
      version: green
  replicas: 3 # Number of replicas that will be created for this
deployment
  template:
    metadata:
      labels:
        app: ekart
        version: green
    spec:
      containers:
        - name: ekart
          image: akscloud.azurecr.io/shopping-cart:latest # Image
that will be used to containers in the cluster
          imagePullPolicy: Always
          ports:
            - containerPort: 8070 # The port that the container is
running on in the cluster
```

Figure 7: Green Deployment (green.yaml)

```yaml
apiVersion: v1 # Kubernetes API version
kind: Service # Kubernetes resource kind we are creating
metadata: # Metadata of the resource kind we are creating
  name: ekart-ssvc
spec:
  selector:
    app: ekart
    version: blue
  ports:
    - protocol: "TCP"
      port: 80 # The port that the service is running on in the
cluster
      targetPort: 8070 # The port exposed by the service
  type: ClusterIP
```

Figure 8: Blue-green service (Change when needed)

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: cart-ingress
  # #namespace: default
spec:
  ingressClassName: nginx
  #tls:
   # - secretName: frontend-tls
     # hosts:
       # - meraviglioso.id.vn
  rules:
    - host: meraviglioso.id.vn
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: ekart-ssvc
                port:
                  number: 80
```

Figure 9: Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster (with TLS if needed)

4.4 Demonstration

4.4.1 Source code and Database

- All related information about source code and Database can be found at: https://github.com/Meraviglioso8/Ekart
- Some information about Database

```
-- password in plaintext: "password"
INSERT INTO USER (user_id, password, email, username, name,
last_name, active)
VALUES
  (1, '$2a$06$OAPObzhRdRXBCbk7Hj/ot.jY3zPwR8n7/mfLtKIgTzdJa4.6TwsIm',
'user@mail.com', 'user', 'Name', 'Surname',
   1);
-- password in plaintext: "password"
INSERT INTO USER (user_id, password, email, username, name,
last_name, active)
VALUES
  (2, '$2a$06$OAPObzhRdRXBCbk7Hj/ot.jY3zPwR8n7/mfLtKIgTzdJa4.6TwsIm',
'johndoe@gmail.com', 'johndoe', 'John', 'Doe', 1);
-- password in plaintext: "password"
INSERT INTO USER (user_id, password, email, username, name,
last_name, active)
VALUES (3,
'$2a$06$OAPObzhRdRXBCbk7Hj/ot.jY3zPwR8n7/mfLtKIgTzdJa4.6TwsIm',
'name@gmail.com', 'namesurname', 'Name',
        'Surname', 1);

INSERT INTO ROLE (role_id, role)
VALUES (1, 'ROLE_ADMIN');
INSERT INTO ROLE (role_id, role)
VALUES (2, 'ROLE_USER');

INSERT INTO USER_ROLE (user_id, role_id)
VALUES (1, 1);
INSERT INTO USER_ROLE (user_id, role_id)
VALUES (1, 2);
INSERT INTO USER_ROLE (user_id, role_id)
VALUES (2, 2);
INSERT INTO USER_ROLE (user_id, role_id)
VALUES (3, 2);

INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Soap', 'Pears baby soap for Kids', 1, 35.75);
```

```
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Tooth Brush', 'Signal Tooth Brushes Size in (L, M, S)', 5,
34.50);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Shirt', 'Casual Shirt imported from France', 3, 1500.00);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Office Bag', 'Leather bag imported from USA', 40, 1000.00);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Bottle', 'Hot Water Bottles', 80, 450.45);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Wrist Watch', 'Imported wrist watches from swiss', 800,
2500.00);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Mobile Phone', '3G/4G capability', 700, 45000.00);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Shampoo', 'Head and Shoulders Shampoo', 500, 300.00);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Leather Wallets', 'Imported Leather Wallets from AUS', 1000,
500.00);
INSERT INTO PRODUCT (name, description, quantity, price)
VALUES ('Camera', 'Imported Canon camera from USA', 10, 85000.00);
```

Figure 10: Database

4.4.2 Main steps

Step 1: Create Resource Group

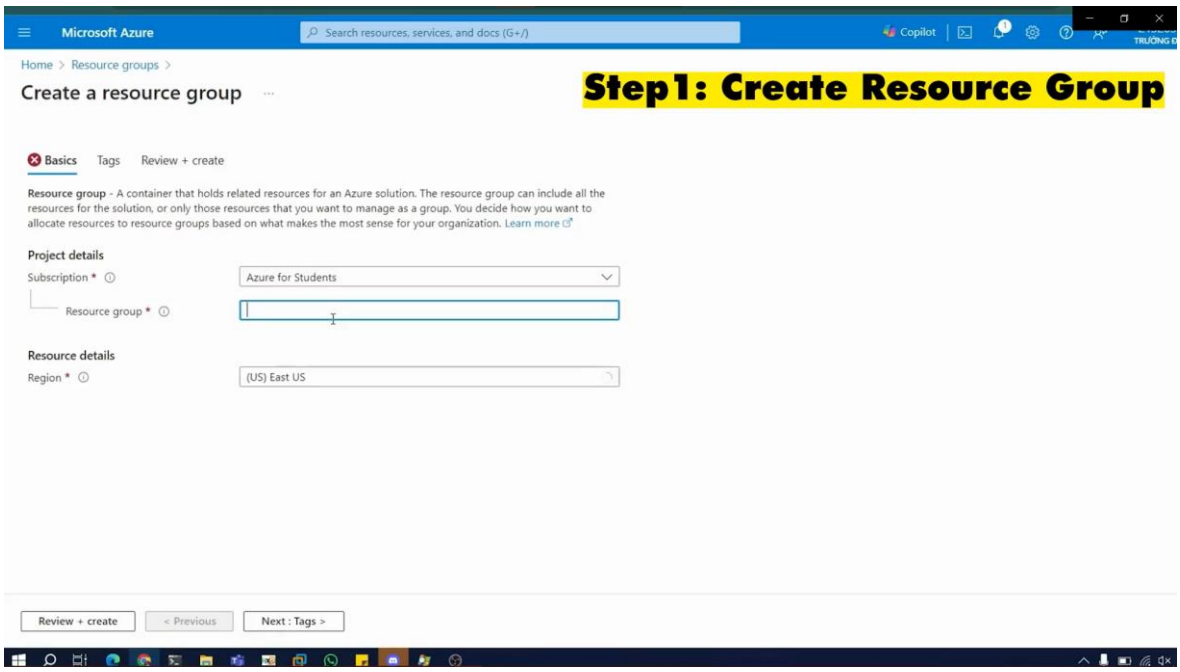

Figure 11: Create Resource Group

Step 2: Create Virtual Machine

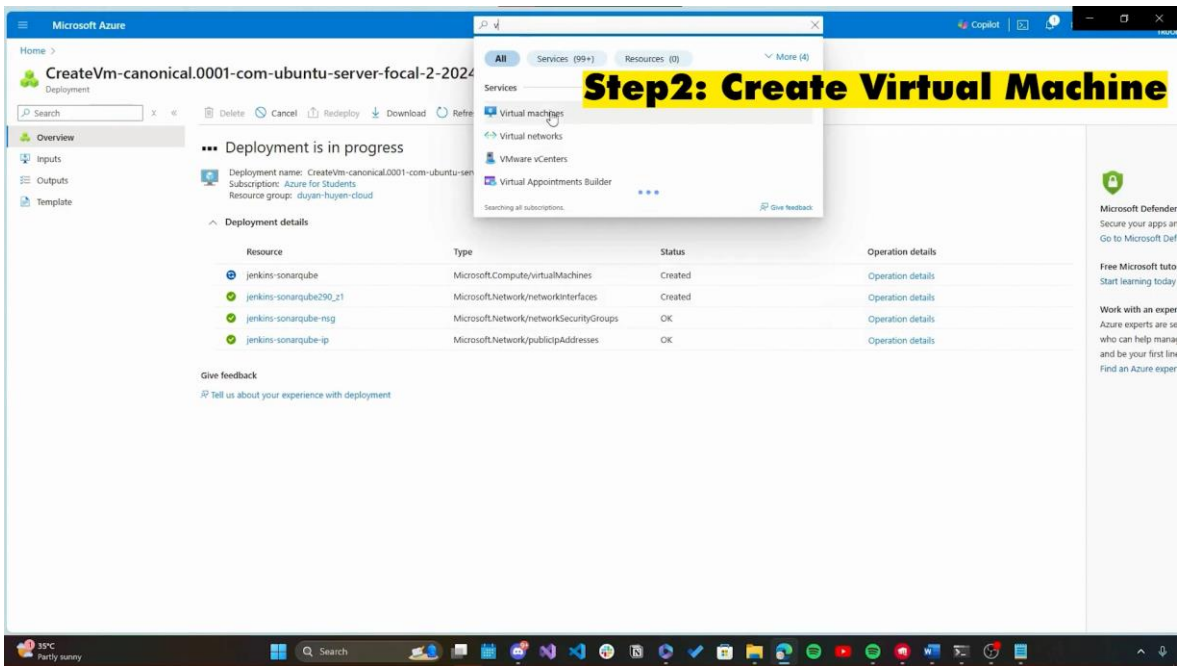

Figure 12: Create Virtual Machine

Step 3: Configure Virtual Machine

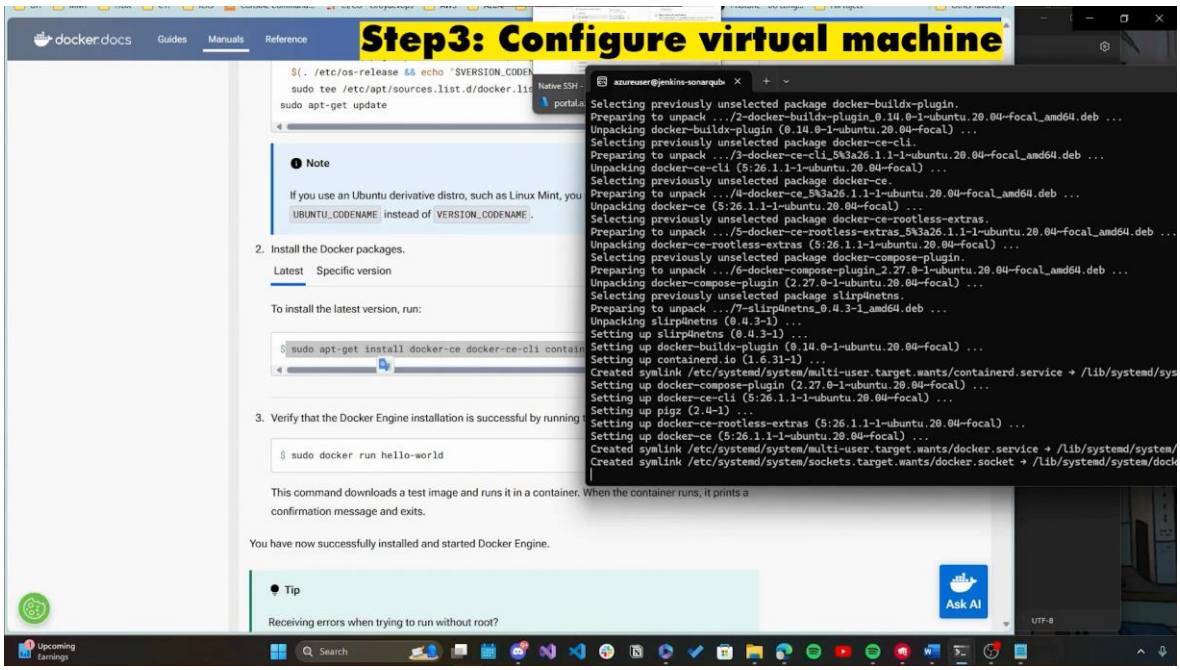

Figure 13: Configure Virtual Machine

Step 4: Install Jenkins, SonarQube

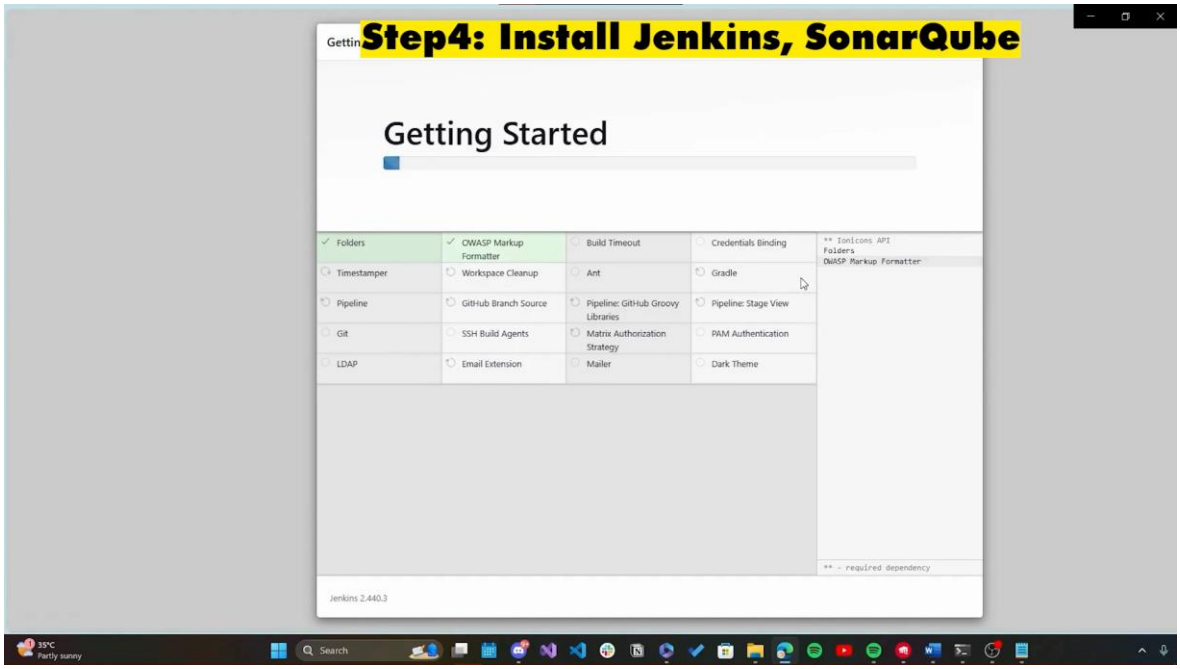

Figure 14: Install Jenkins, SonarQube

Step 5: Create Azure Container Registry

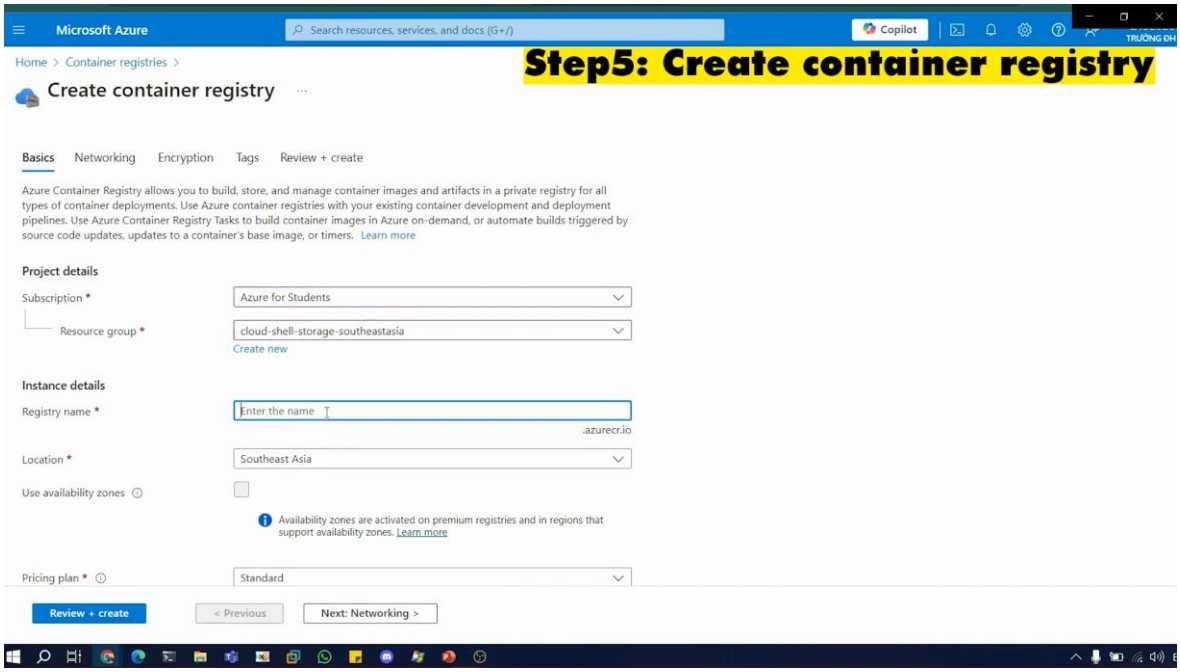

Figure 15: Create container registry

Step 6: Create K8s cluster

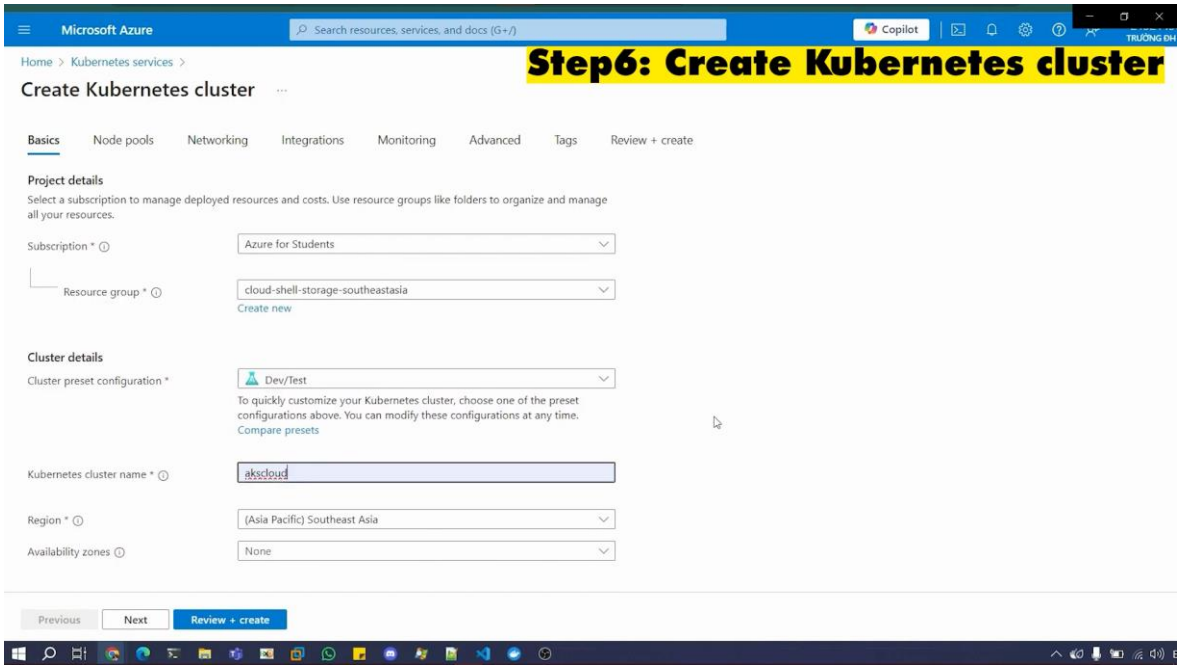

Figure 16: Create Kubernetes cluster

Step 7: Create Azure Key Vault

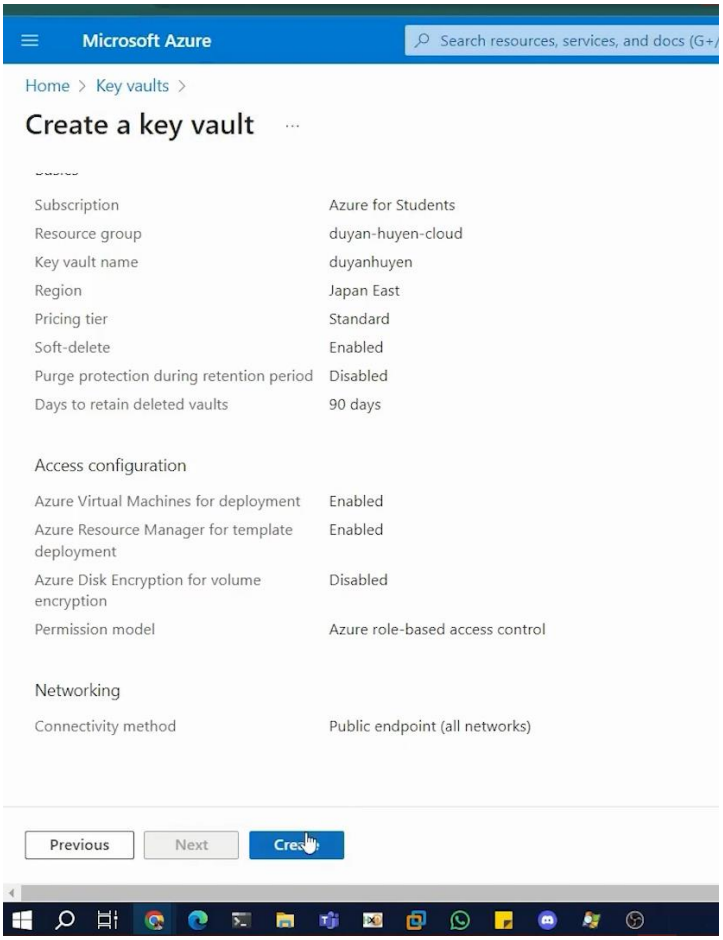

Figure 17: Create Azure Key Vault

Step 8: Create Service Principle

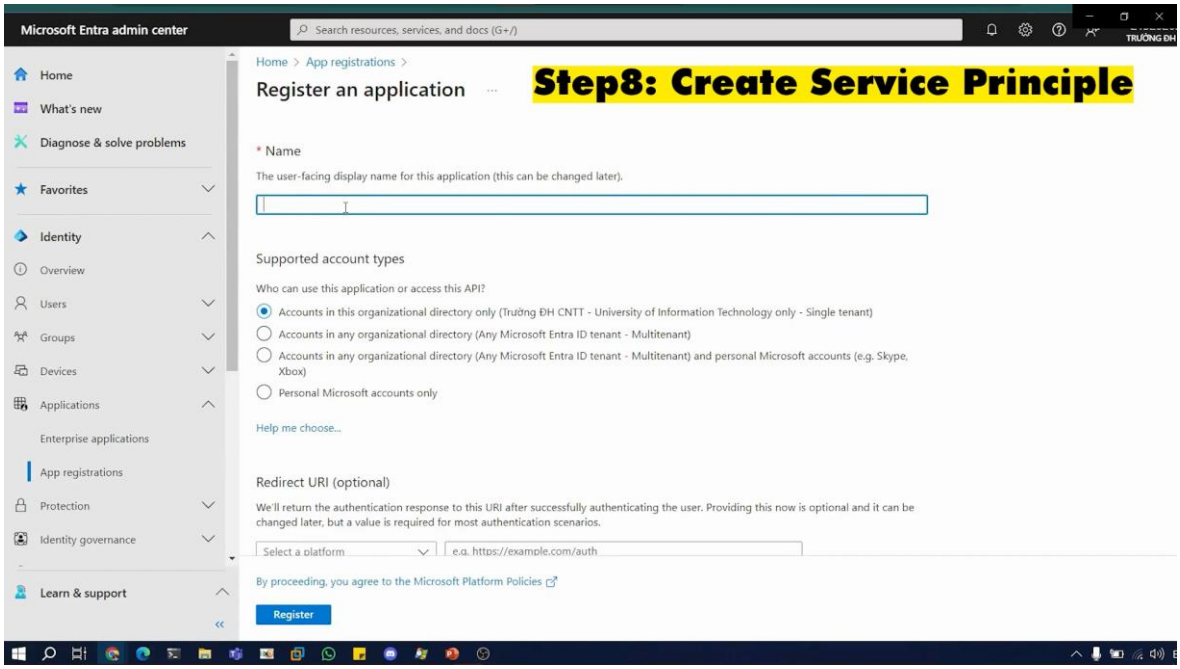

Figure 18: Create Service Principle

Step 9.1: Install Jenkins's plugin

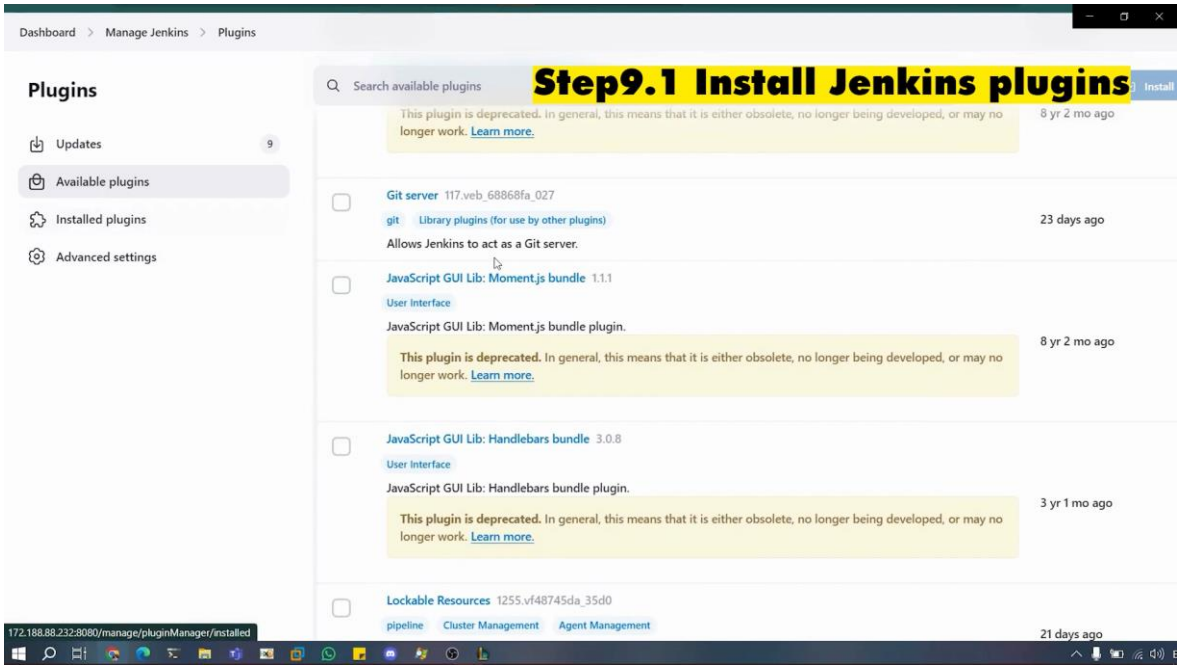

Figure 19: Install Jenkins plugins

Step 9.2: Set up Jenkins's tools

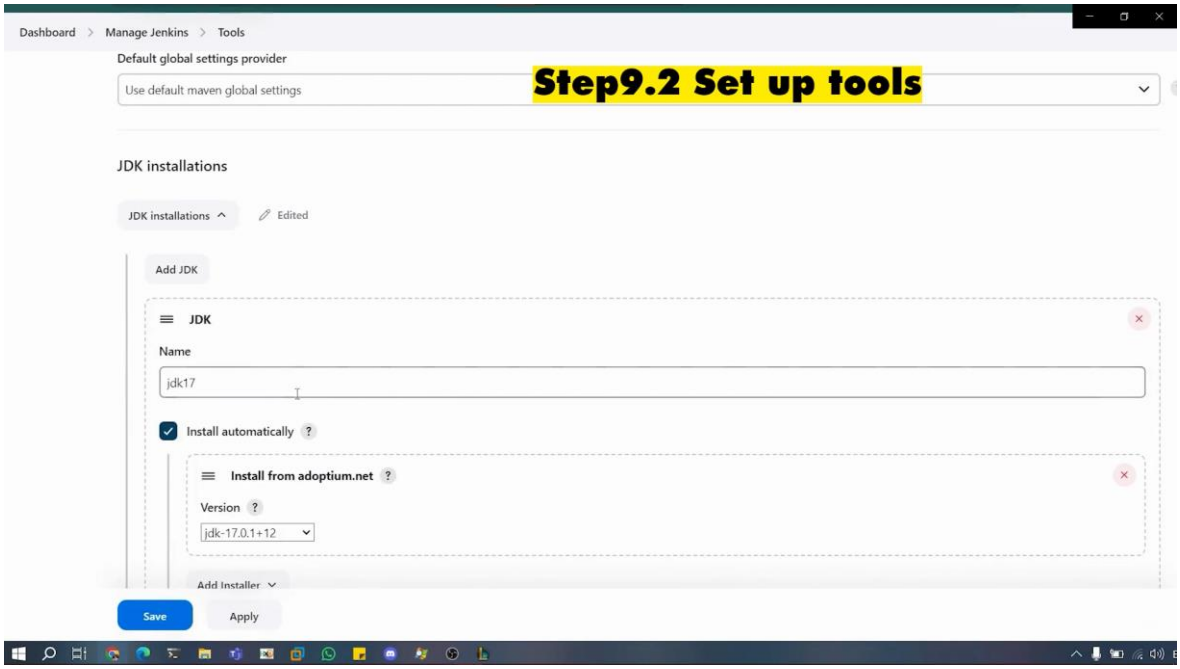

Figure 20: Set up Jenkins's tools

Step 9.3: Set up system variables

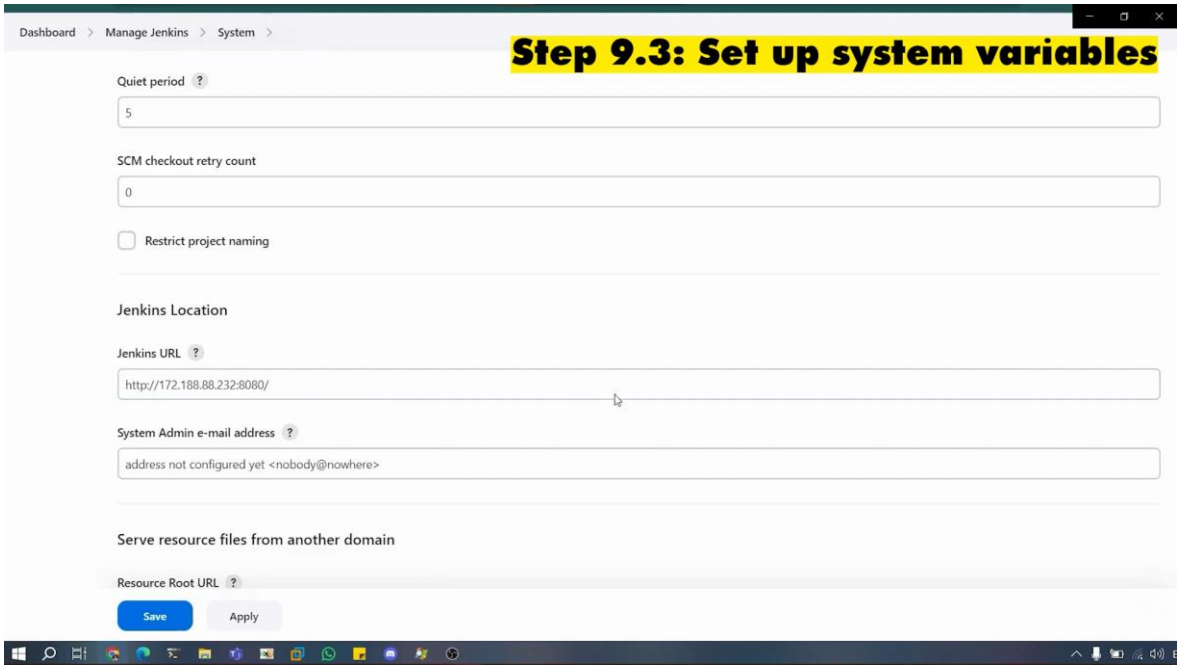

Figure 21: Set up Jenkins's system variables

Step 9.4: Set up Jenkins pipeline

Figure 22: Set up Jenkins pipeline

```
pipeline {
    agent any

    environment {
    REGISTRY_NAME = "PLACE-HOLDER-REGISTRY_NAME" // Change to yours
    ACR_LOGIN_SERVER = "${REGISTRY_NAME}.azurecr.io"
    REPOSITORY_NAME = "PLACE-HOLDER-REPOSITORY_NAME" // Change to yours
    AZURE_CREDENTIALS_ID = credentials('azure-sp-credentials') // Change to your credentials ID
    AKS_RESOURCE_GROUP = "PLACE-HOLDER-RESOURCE-GROUP-NAME" // Change to yours
    AKS_CLUSTER_NAME = "PLACE-HOLDER-CLUSTER-NAME" // Change to yours
    AZURE_SP_TENANT = "PLACE-HOLDER-SP-TENANT" // Change to yours
    }
```

```
        tools {
            jdk 'jdk17'
            maven 'maven3'
        }

        stages {
            stage('Git Checkout') {
                steps {
                    git branch: 'main', url: 'https://github.com/Meraviglioso8/Ekart.git'
                }
            }

            stage('Compile') {
                steps {
                    sh "mvn compile"
                }
            }

            stage('Unit Tests') {
                steps {
                    sh "mvn test -DskipTests=true"
                }
            }

            stage('OWASP Dependency Check') {
                steps {
                    sh 'mvn org.apache.maven.plugins:maven-dependency-plugin:2.10:tree -
Dverbose=true'
```

```
        }
    }

    stage('Snyk Security Scan') {
        steps {
            snykSecurity(
                snykInstallation: 'snyk-latest',
                snykTokenId: 'snykToken',
                severity: 'critical',
                failOnIssues: false,
                monitorProjectOnBuild: true
            )
        }
    }

    stage('SonarQube Analysis') {
        steps {
            withSonarQubeEnv('sonarqube') {
                // Corrected the typos and parameterized the token
                sh 'mvn clean verify sonar:sonar -Dmaven.test.skip=true -Dmaven.test.failure.ignore=true -Dsonar.projectName=ekart -Dsonar.projectKey=ekart -Dsonar.projectVersion=1.0 -Dsonar.exclusions=**/*.ts'
            }
        }
    }

    stage('Build') {
        steps {
            sh "mvn package -DskipTests=true"
```

```
        }
    }

    stage('Build & Tag Docker Image') {
        steps {
            script {
                withDockerRegistry(credentialsId: 'dockerhub', toolName: 'docker') {
                    sh "docker build -t shopping-cart -f docker/Dockerfile ."
                    sh "docker tag shopping-cart ${REGISTRY_NAME}.azurecr.io/${REPOSITORY_NAME}:latest"
                }
            }
        }
    }

    stage('Trivy Scan') {
        steps {
            sh '''
                # Run Trivy scan on the Docker image
                if command -v trivy &> /dev/null; then
                    trivy image ${REGISTRY_NAME}.azurecr.io/${REPOSITORY_NAME}:latest > trivy-report.txt
                else
                    docker run --rm -v /var/run/docker.sock:/var/run/docker.sock aquasec/trivy:latest image ${REGISTRY_NAME}.azurecr.io/${REPOSITORY_NAME}:latest > trivy-report.txt
                fi
```

```
            '''
        }
        }
        stage('Upload Image to ACR') {
         steps{
            script {
              withCredentials([usernamePassword(credentialsId: 'acr-credentials', usernameVariable: 'SERVICE_PRINCIPAL_ID', passwordVariable: 'SERVICE_PRINCIPAL_PASSWORD')]) {
                sh "docker login ${ACR_LOGIN_SERVER} -u $SERVICE_PRINCIPAL_ID -p $SERVICE_PRINCIPAL_PASSWORD"
                sh " docker push ${REGISTRY_NAME}.azurecr.io/${REPOSITORY_NAME}:latest"
              }
            }
         }
        }

        stage('Deploy to AKS') {
           steps {
              script {
                withCredentials([usernamePassword(credentialsId: 'azure-sp-credentials', usernameVariable: 'AZURE_SP_APP_ID', passwordVariable: 'AZURE_SP_PASSWORD')]) {
                  sh '''
                    az login --service-principal -u $AZURE_SP_APP_ID -p $AZURE_SP_PASSWORD --tenant $AZURE_SP_TENANT
                    az aks get-credentials --resource-group $AKS_RESOURCE_GROUP --name $AKS_CLUSTER_NAME
```

```
                    for yaml_file in $(find . -name '*.yaml'); do
                        kubectl apply -f $yaml_file
                    done
                '''
            }
        }
    }
}

//stage('Push The Docker Image') {
  // steps {
     // script {
        // withDockerRegistry(credentialsId: 'dockerhub', toolName: 'docker') {
        //     sh "docker push meraviglioso8/shopping-cart:latest"
        // }

     //}
   //}
//}

// stage('Kubernetes Deploy') {
//     steps {
//         withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8-token', namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'https://172.31.15.138:6443') {
//             sh "kubectl apply -f deploymentservice.yml -n webapps"
//             sh "kubectl get svc -n webapps"
//         }
//     }
```

```
            // }
        }
}
```

Figure 23: Full Jenkins pipeline

4.4.3 Pipeline's steps

Step 1: Commit code

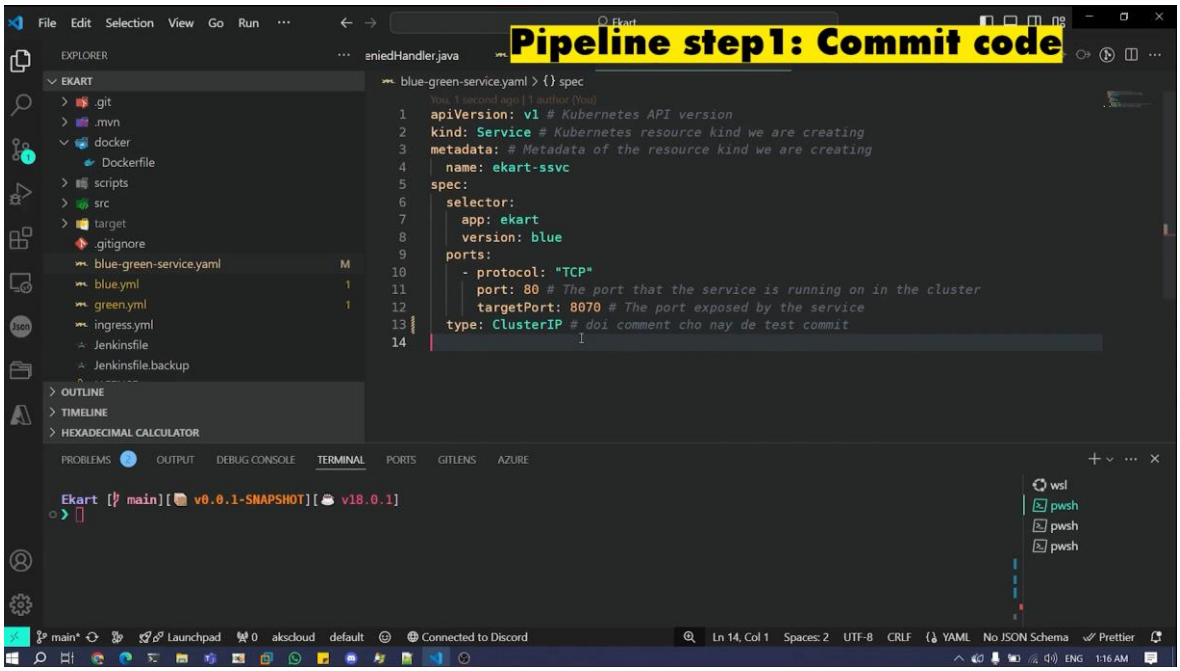

Figure 24: Pipeline step 1

Step 2: Check pipeline configuration

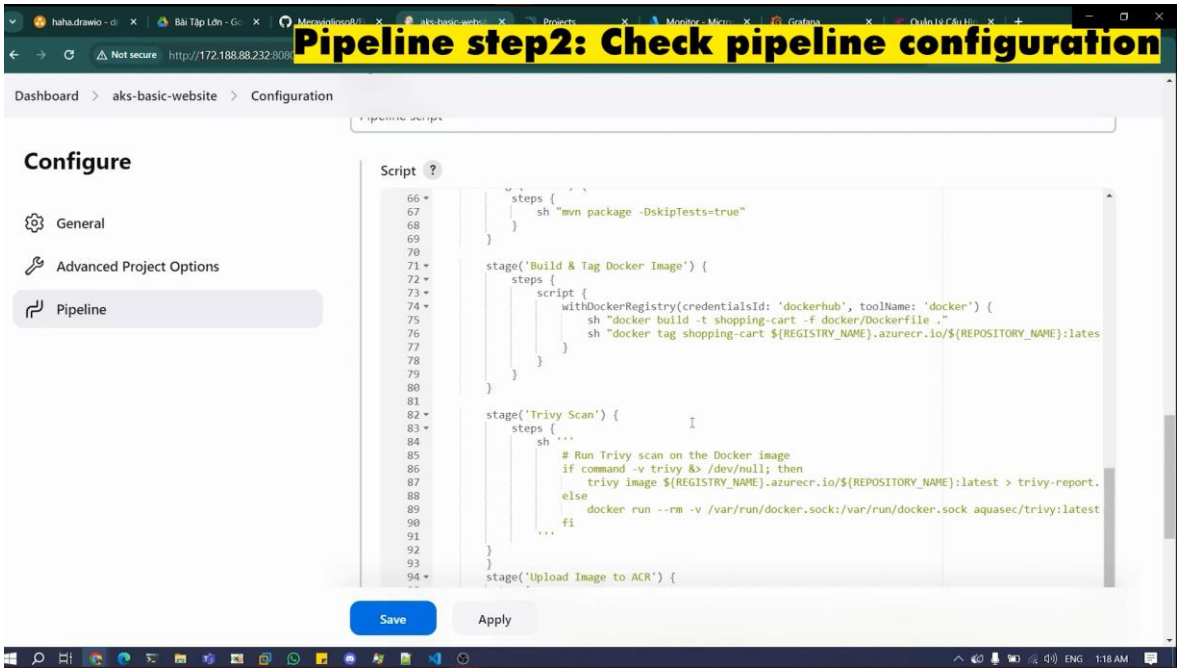

Figure 25: Pipeline step 2

Step 3: Run pipeline

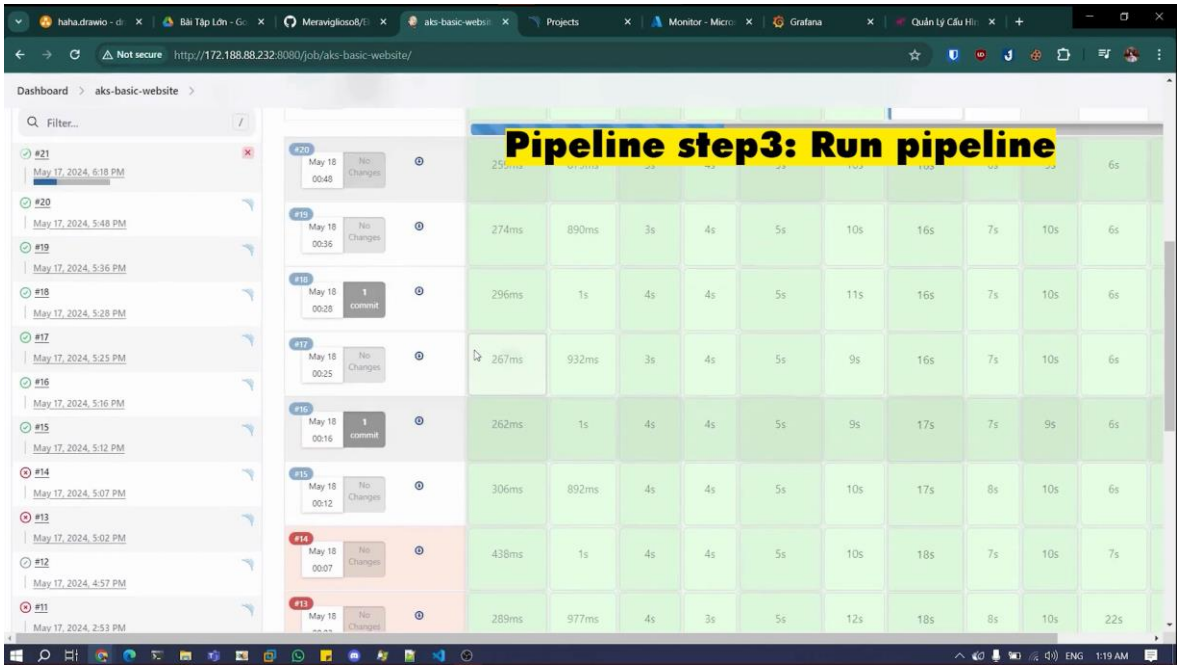

Figure 26: Pipeline step 3

Step 4: Check pipeline's status/output

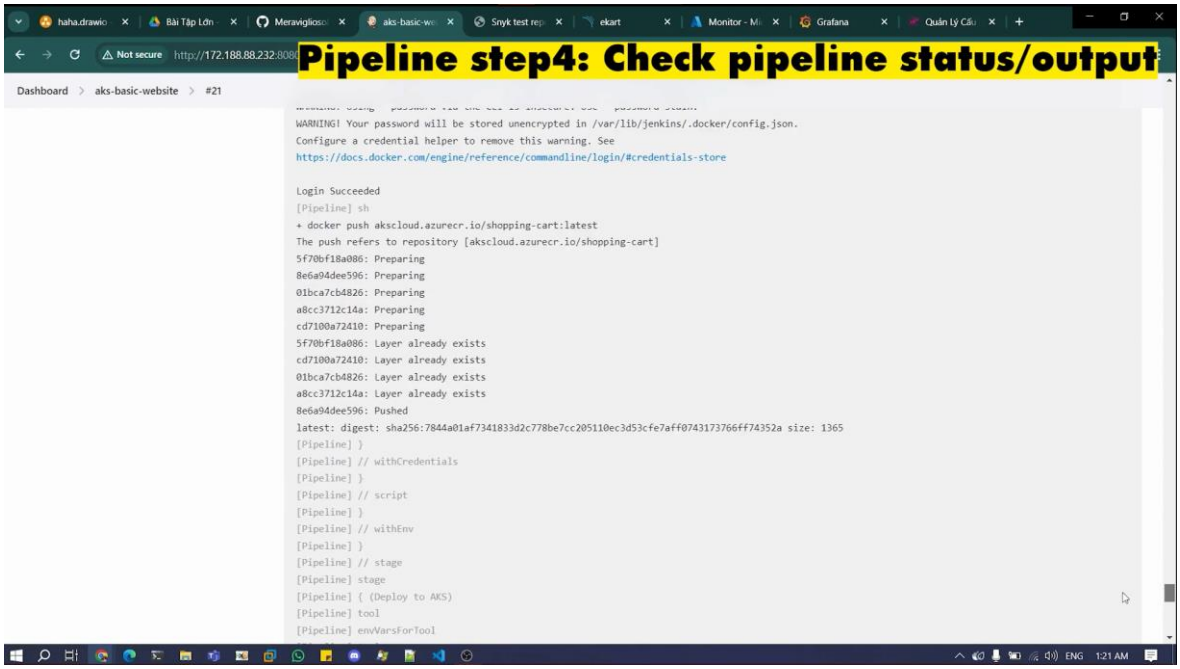

Figure 27: Pipeline step 4

Step 5: Check deployment status

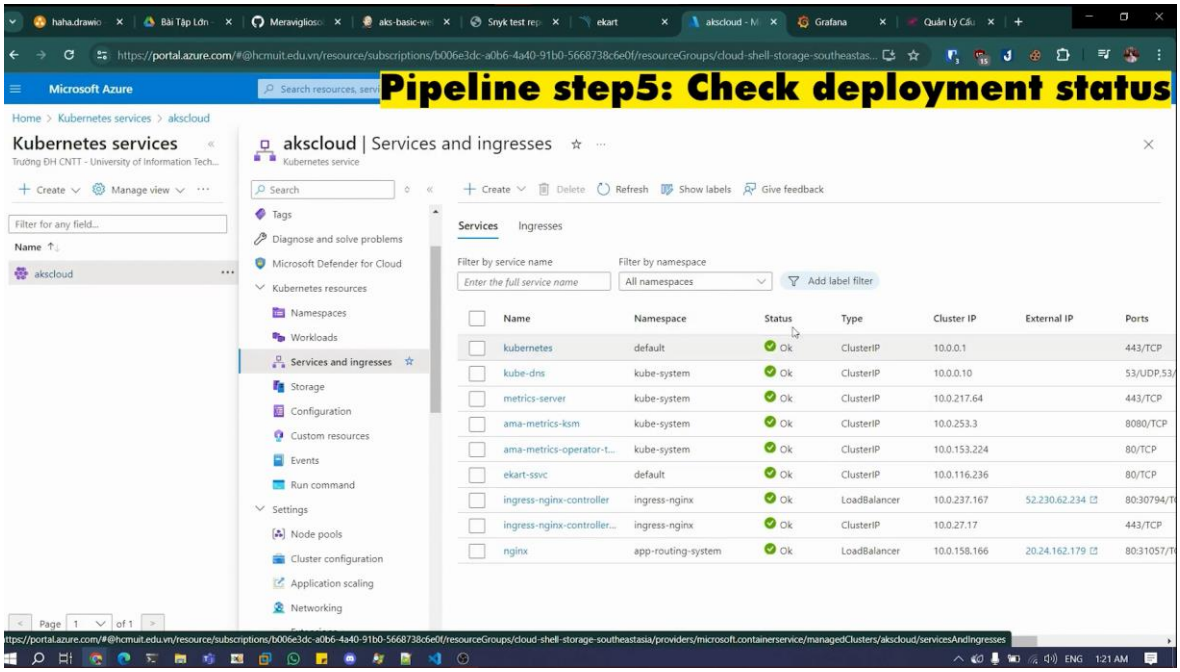

Figure 28: Pipeline step 5.1

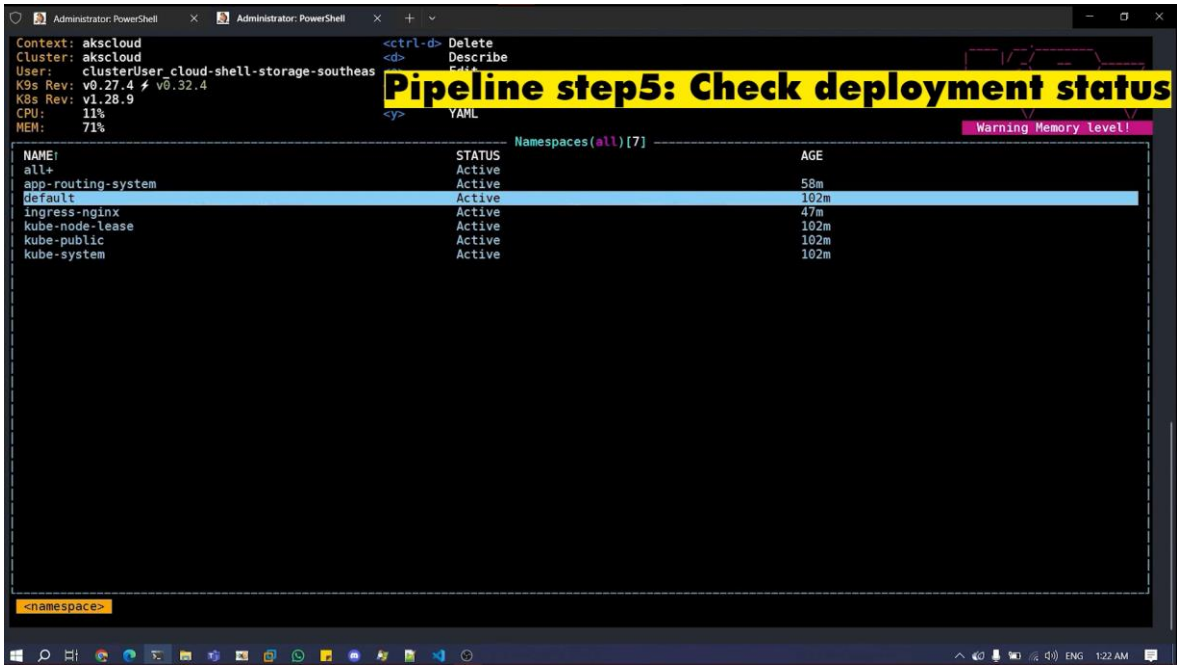

Figure 29: Pipeline step 5.2

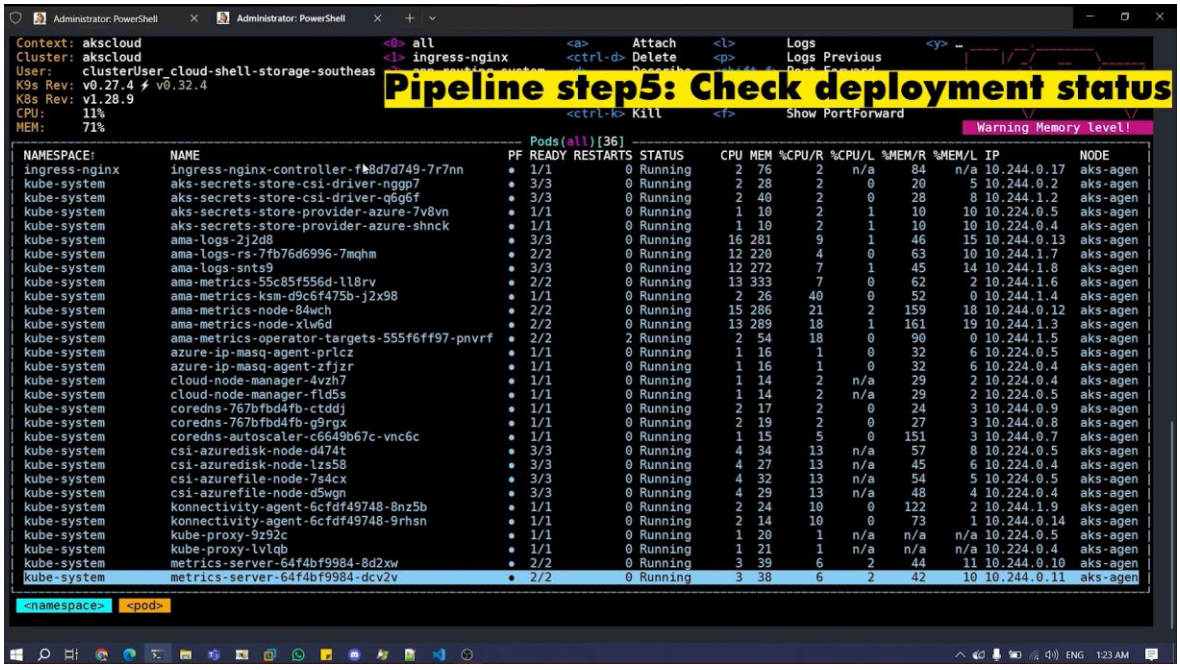

Figure 30: Pipeline step 5.3

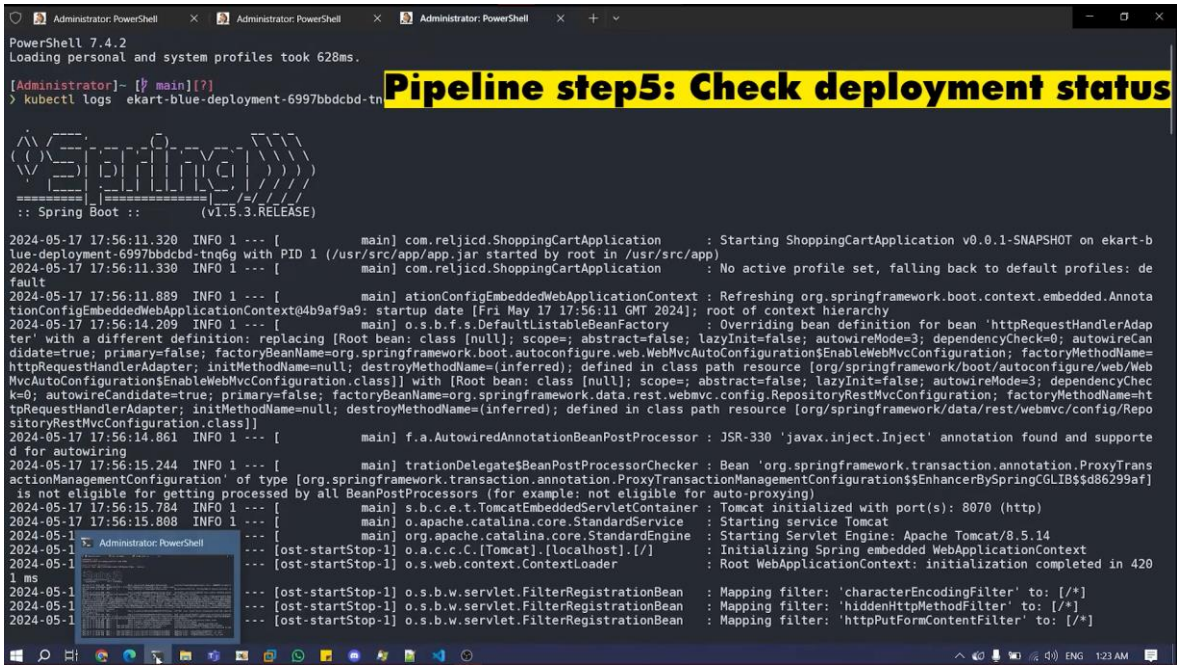

Figure 31: Pipeline step 5.4

4.4.4 Back to main steps

Step 10: Check web status

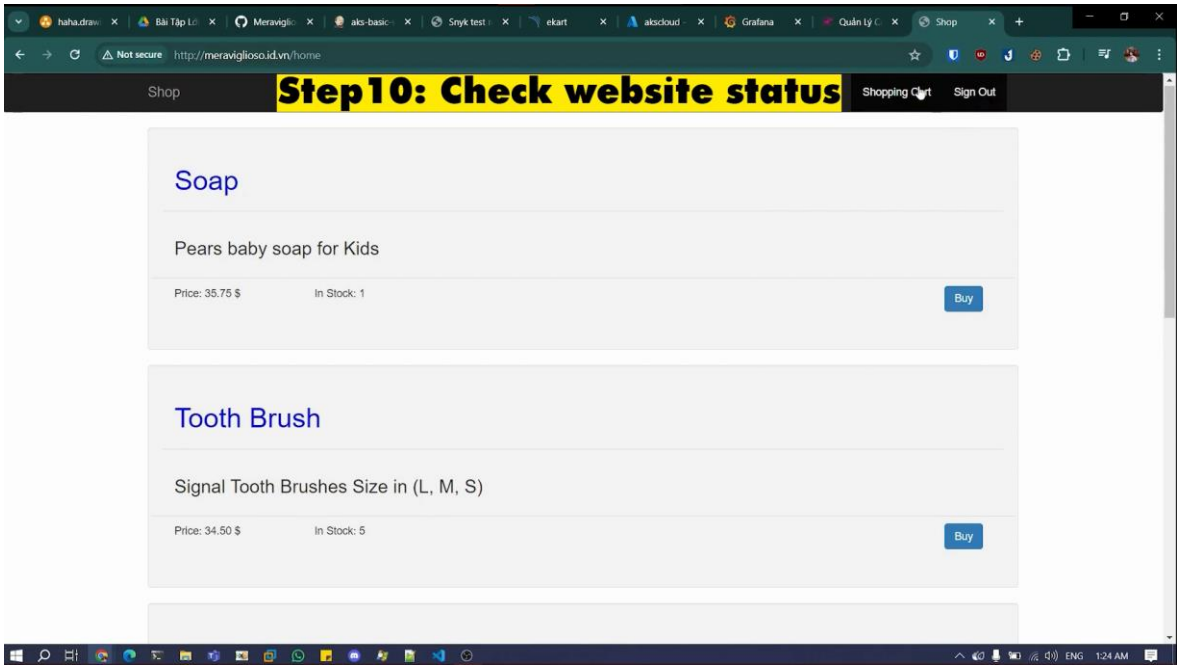

Figure 32: Check web status

Step 11: Check all scan services

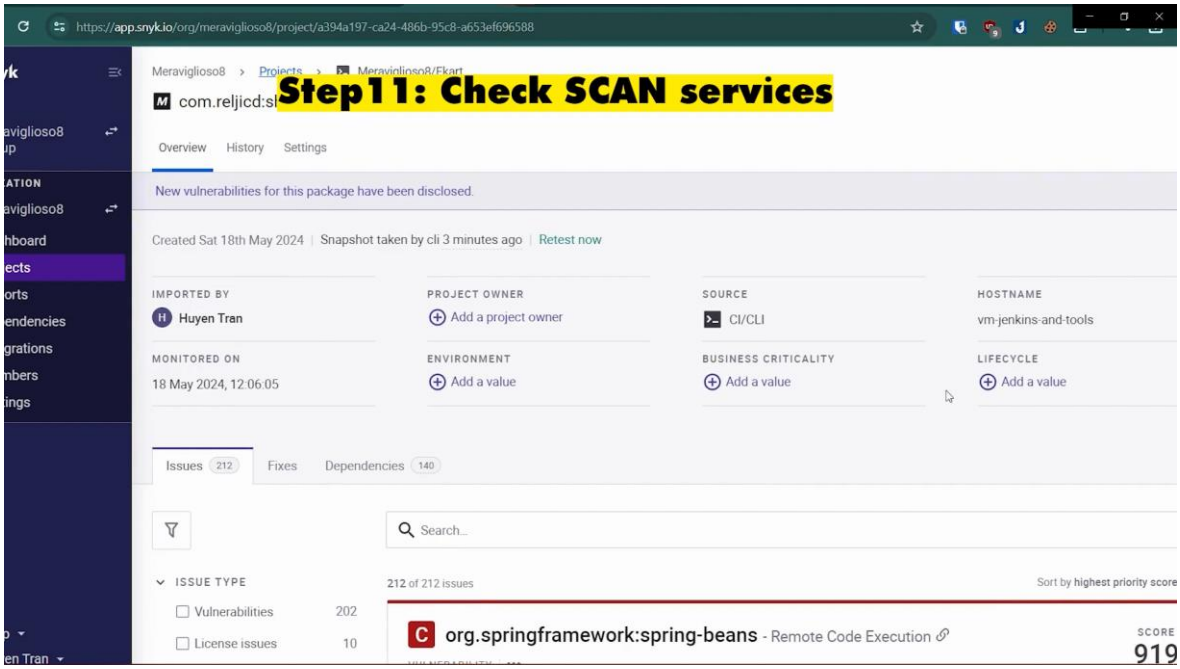

Figure 33: Check Scan service 1

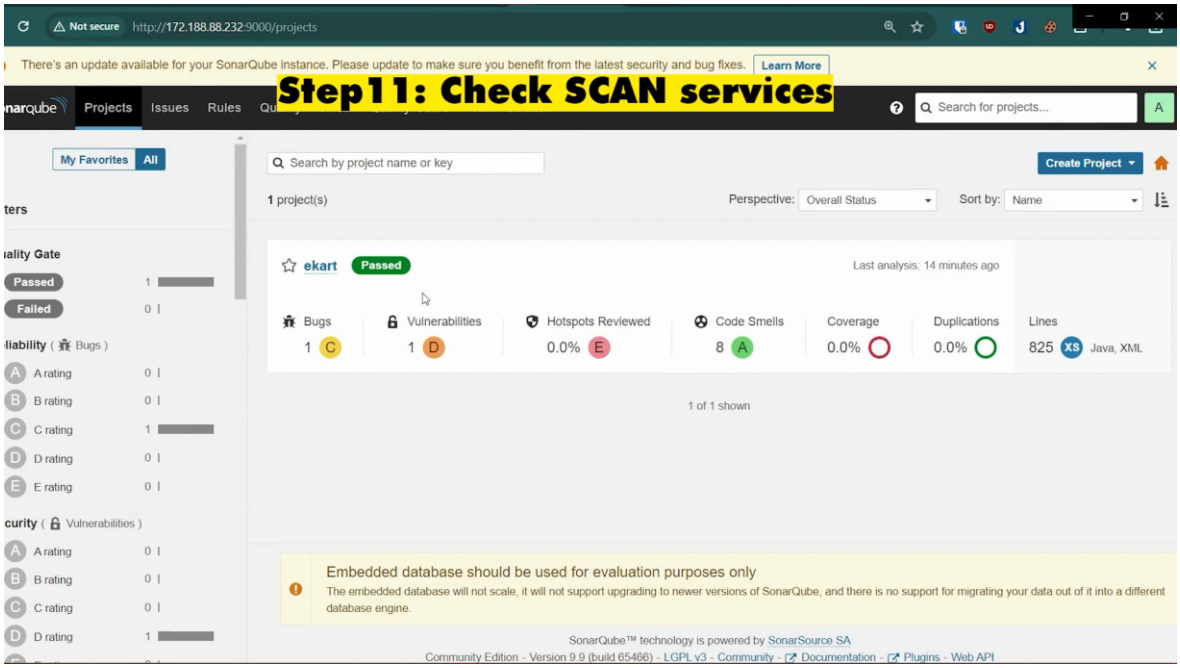

Figure 34: Check Scan service 2

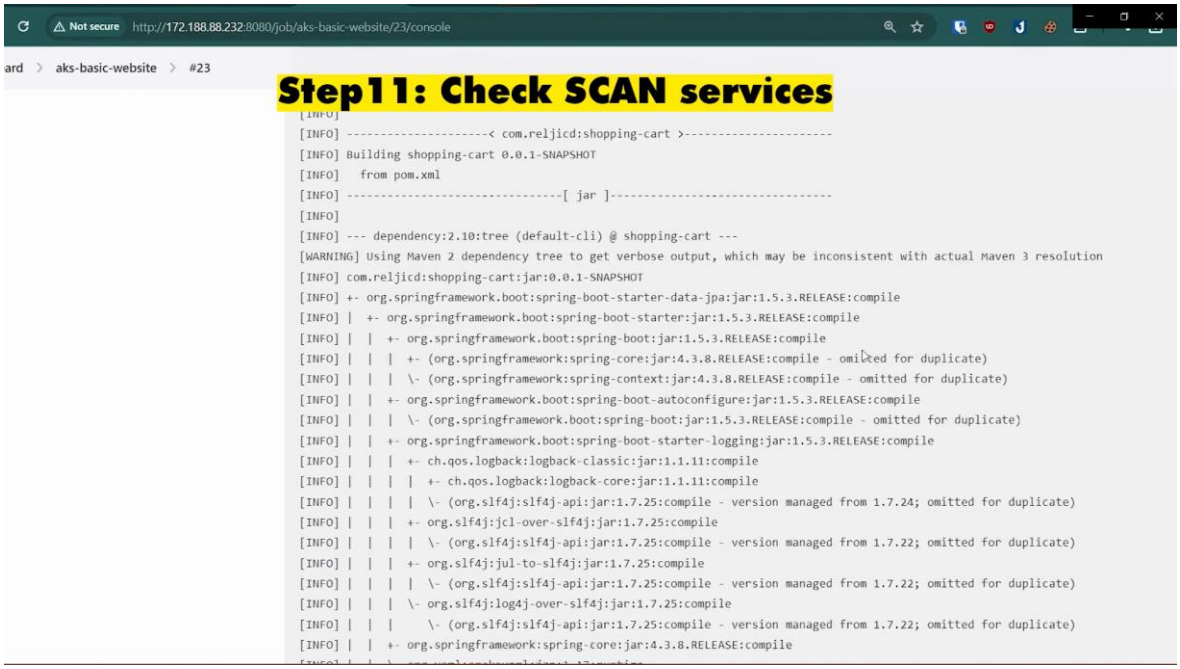

Figure 35: Check Scan service 3

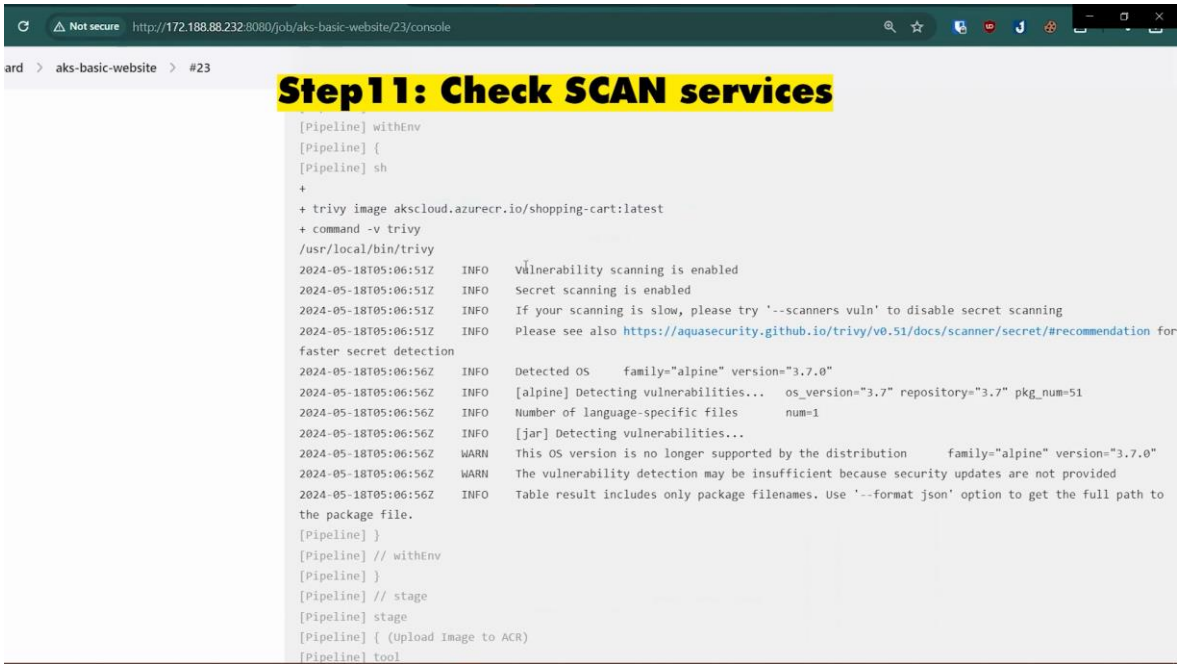

Figure 36: Check Scan service 4

Step 12: Check container image and ACR

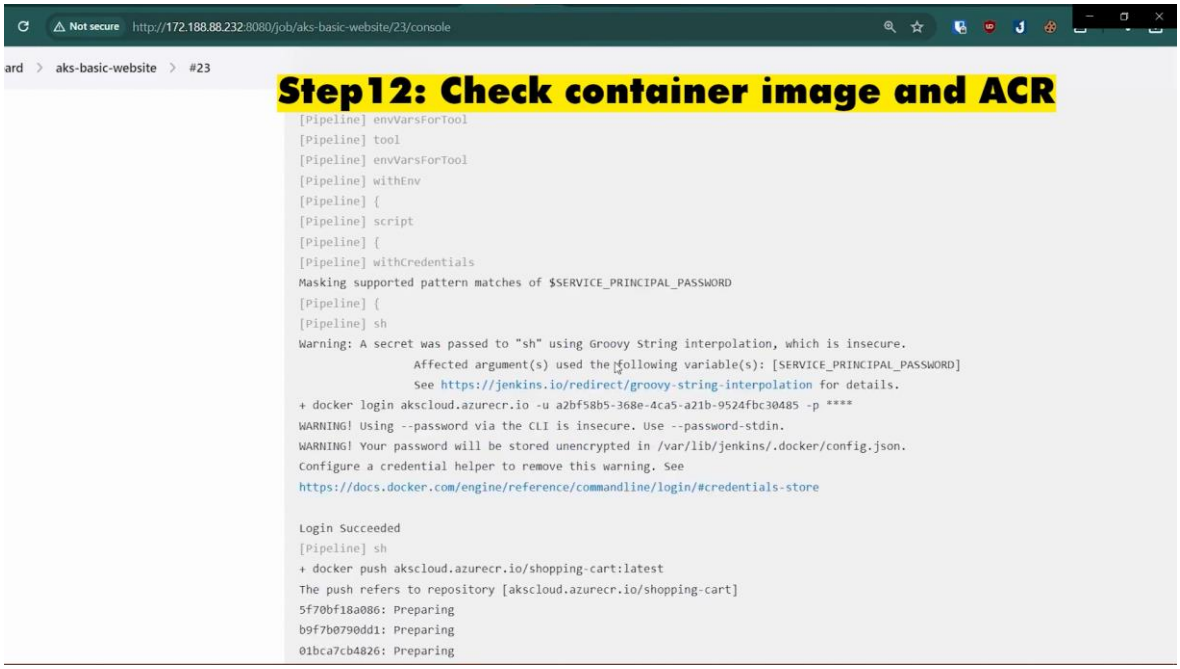

Figure 37: Check container image and ACR.1

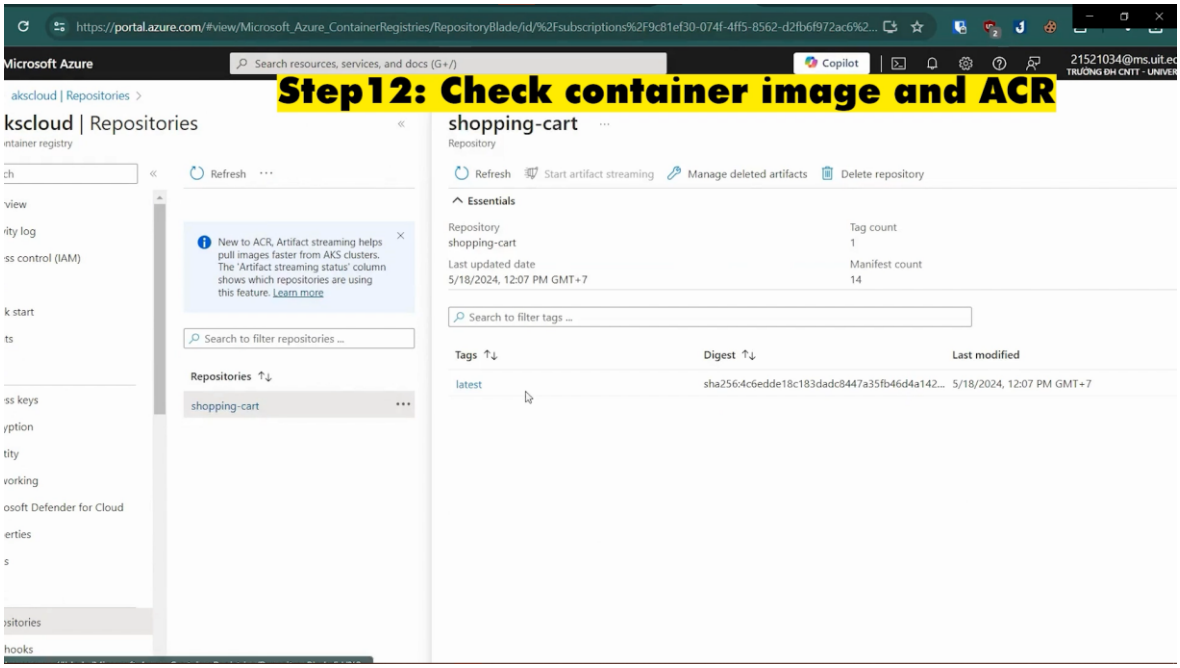

Figure 38: Check container image and ACR.2

Step 13: Inspect Azure Monitor and Grafana

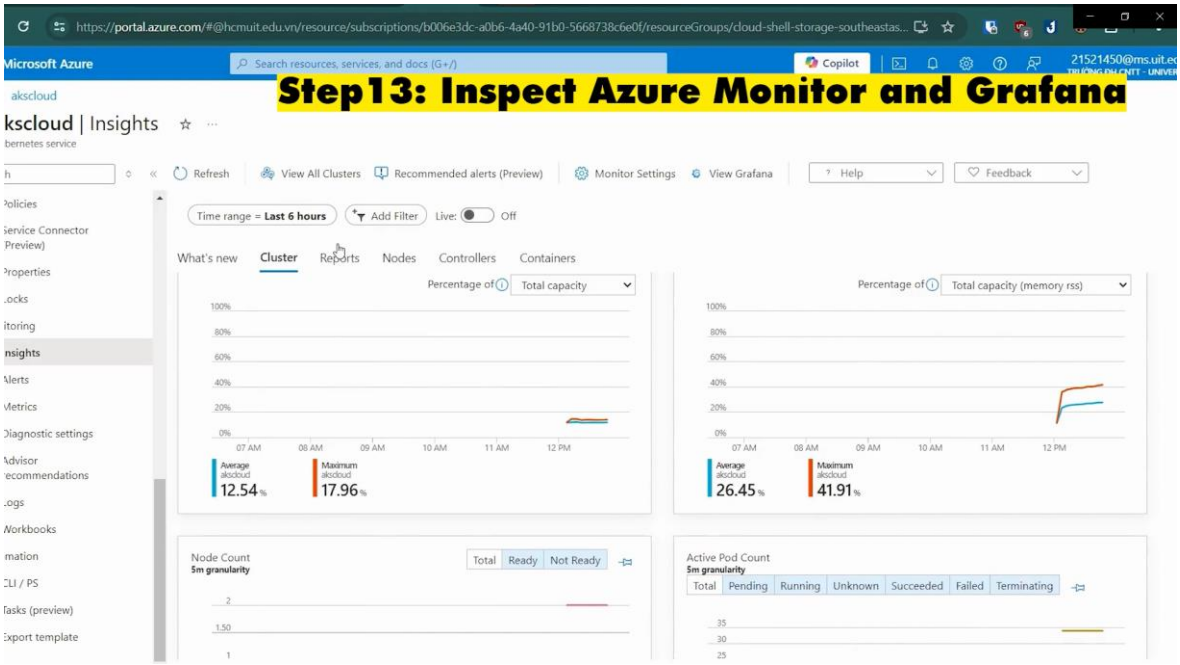

Figure 39: Inspect Azure Monitor

Figure 40: Inspect Grafana

Step 14: Using terraform for IaC

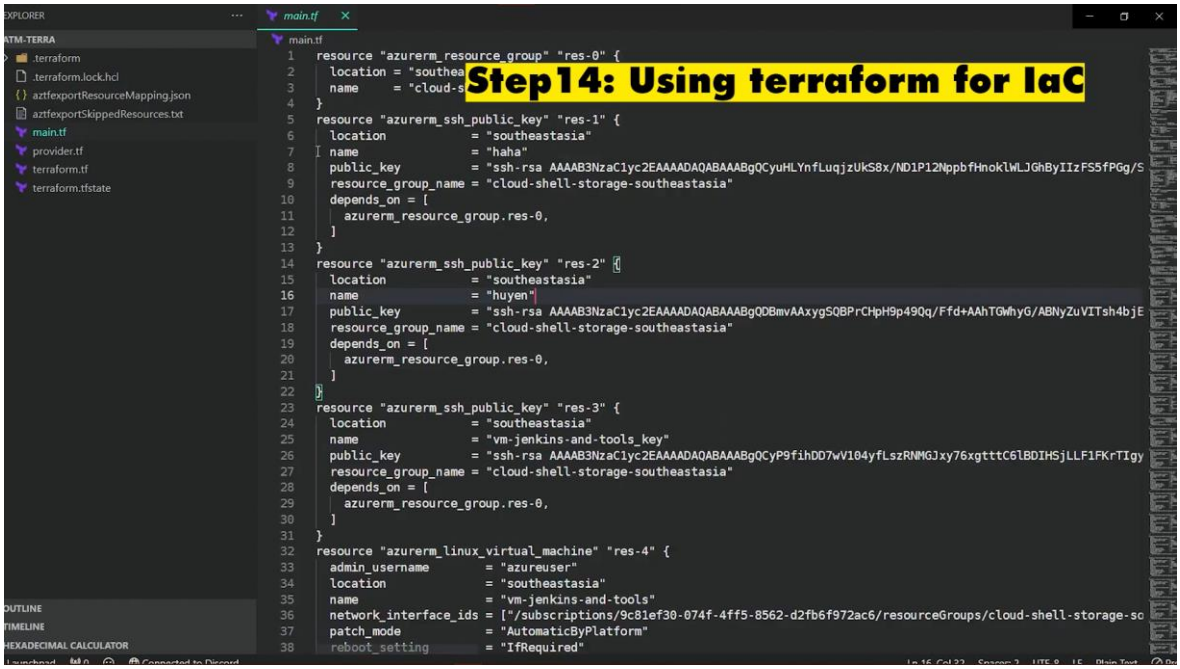

Figure 41: Using terraform for IaC.1

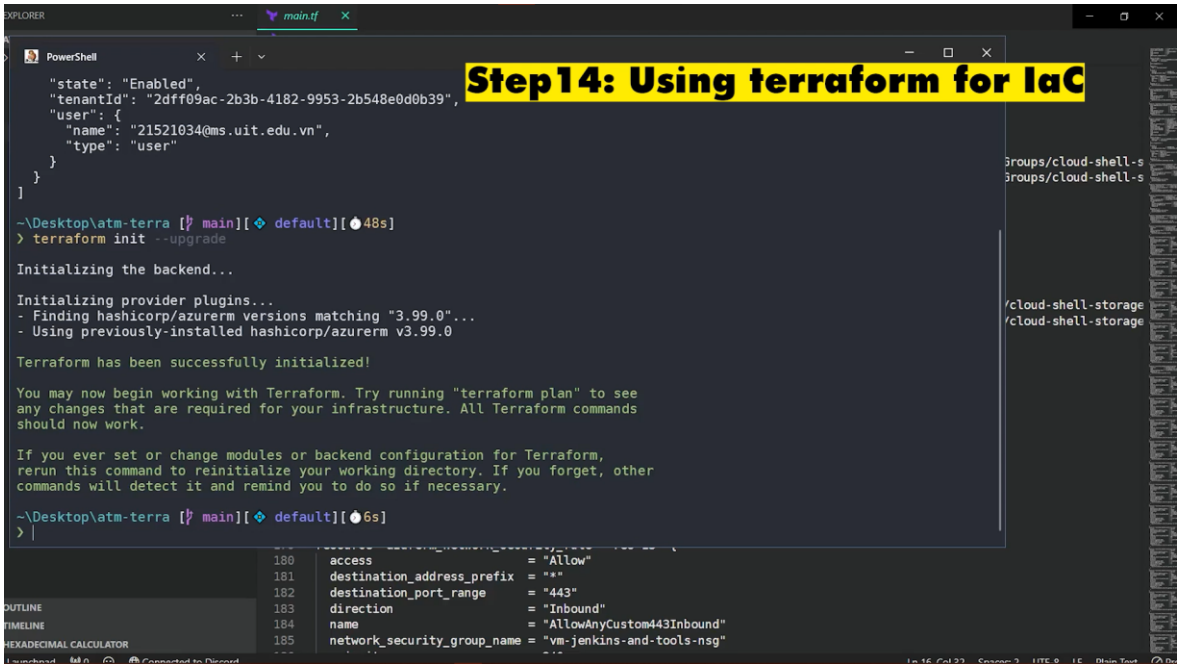

Figure 42: Using terraform for IaC.2

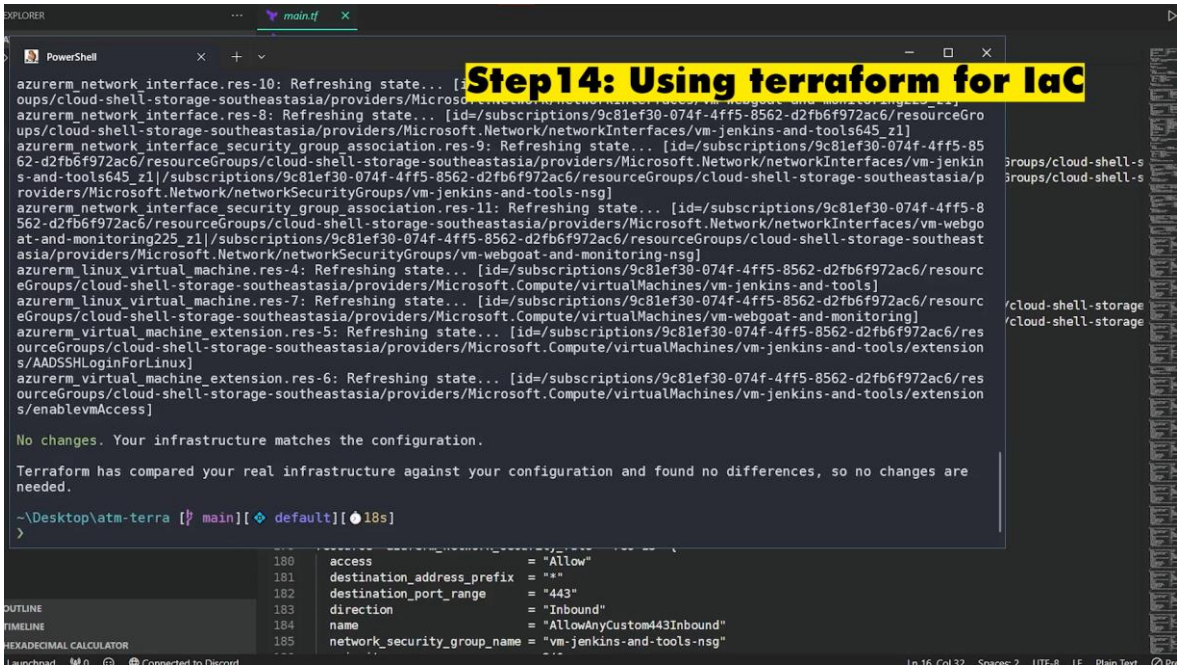

Figure 43: Using terraform for IaC.3

Chapter 5**: References**

1) Simplilearn (2023) *Cloud computing vs traditional computing: Simplilearn, Simplilearn.com*. Available at: https://www.simplilearn.com/cloud-computing-vs-traditional-computing-article (Accessed: 01 June 2024).
2) Kirby, J. (2021) *How cloud computing differs from traditional IT infrastructure, Micro Pro*. Available at: https://micropro.com/blog/cloud-computing-vs-traditional-it-infrastructure/ (Accessed: 01 June 2024).
3) Tejaswikolli-Web *Azure Container Registry Documentation, Microsoft Learn*. Available at: https://learn.microsoft.com/en-us/azure/container-registry/ (Accessed: 01 June 2024).
4) Azure Container Registry Microsoft Azure. Available at: https://azure.microsoft.com/en-us/products/container-registry/ (Accessed: 02 June 2024).
5) Team, C.A. (2023) Azure Container Registry Overview: Build, manage, and store container images and artifacts in a private registry, Cloud Academy. Available at: https://cloudacademy.com/blog/azure-container-registry-overview-build-manage- (Accessed: 02 June 2024).
6) Code quality tool & secure analysis with SonarQube (no date) Clean Code: Writing Clear, Readable, Understandable & Reliable Quality Code. Available at: https://www.sonarsource.com/products/sonarqube/ (Accessed: 02 June 2024).
7) Guide to software composition analysis (SCA) (2022) Snyk. Available at: https://snyk.io/series/open-source-security/software-composition-analysis-sca/ (Accessed: 02 June 2024).
8) Develop fast. stay secure. (no date) Snyk. Available at: https://go.snyk.io/Forrester-Wave-SCA-2023.html (Accessed: 02 June 2024).
9) *Trivy home* (2023) *Trivy*. Available at: https://trivy.dev/ (Accessed: 02 June 2024).

10) Trivy open source vulnerability scanner (2022) Aqua. Available at: https://www.aquasec.com/products/trivy/ (Accessed: 02 June 2024).
11) Aquasecurity. (n.d.). GitHub - aquasecurity/trivy: Find vulnerabilities, misconfigurations, secrets, SBOM in containers, Kubernetes, code repositories, clouds and more. GitHub. https://github.com/aquasecurity/trivy
12) What is Prometheus? | Grafana documentation. (n.d.). Grafana Labs. https://grafana.com/docs/grafana/latest/fundamentals/intro-to-prometheus/
13) Prometheus. (n.d.). Overview | Prometheus. https://prometheus.io/docs/introduction/overview/
14) Rboucher. (n.d.). Azure Monitor documentation - Azure Monitor. Microsoft Learn. https://learn.microsoft.com/en-us/azure/azure-monitor/
15) Technical documentation | Grafana Labs. (n.d.). Grafana Labs. https://grafana.com/docs/
16) Scooley. (2022, April 26). Introduction to Hyper-V on Windows 10. Microsoft Learn. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/
17) Roygara. (2024, April 12). Azure Disk Storage overview - Azure Virtual Machines. Microsoft Learn. https://docs.microsoft.com/en-us/azure/virtual-machines/managed-disks-overview
18) Mbender-Ms. (2024, April 17). What is Azure Load Balancer? - Azure Load Balancer. Microsoft Learn. https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-overview
19) Greg-Lindsay. (2023, September 27). What is Azure Application Gateway. Microsoft Learn. https://docs.microsoft.com/en-us/azure/application-gateway/overview

20) Ju-Shim. (2023, April 11). Azure Virtual Machine Scale Sets overview - Azure Virtual Machine Scale Sets. Microsoft Learn. https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview
21) Dcurwin. (2023, November 16). What is Microsoft Defender for Cloud? - Microsoft Defender for Cloud. Microsoft Learn. https://docs.microsoft.com/en-us/azure/security-center/security-center-intro
22) ankitaduttaMSFT. (2024, March 27). About Azure Site Recovery - Azure Site Recovery. Microsoft Learn. https://docs.microsoft.com/en-us/azure/site-recovery/site-recovery-overview
23) Kubernetes scheduler. (2024, February 16). Kubernetes. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/
24) Horizontal pod autoscaling. (2024, February 18). Kubernetes. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
25) Documentation | Terraform | HashiCorp Developer. (n.d.). Documentation | Terraform | HashiCorp Developer. https://www.terraform.io/docs/index.html
26) Brikman, Y. (n.d.). Terraform: Up and Running, 3rd edition. O'Reilly Online Learning. https://www.oreilly.com/library/view/terraform-up-and/9781098116736/
27) Welcome | English | Terraform Best Practices. (n.d.). https://www.terraform-best-practices.com/
28) Msmbaldwin. (2024, January 30). Azure Key Vault Overview - Azure Key Vault. Microsoft Learn. https://docs.microsoft.com/en-us/azure/key-vault/general/overview
29) Msmbaldwin. (2024b, May 23). Best practices for using Azure Key Vault. Microsoft Learn. https://docs.microsoft.com/en-us/azure/key-vault/general/best-practices
30) Msmbaldwin. (2024b, April 4). Azure Quickstart - Set and retrieve a secret from Key Vault using Azure portal. Microsoft Learn. https://docs.microsoft.com/en-us/azure/key-vault/secrets/quick-create-portal

31) Greg-Lindsay. (2023a, August 15). Azure Traffic Manager. Microsoft Learn. https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview
32) Roygara. (2023, November 23). Azure premium storage: Design for high performance - Azure Virtual Machines. Microsoft Learn. https://learn.microsoft.com/en-us/azure/virtual-machines/premium-storage-performance
33) aKumoSolutions. (2024, February 2). Speed up Jenkins Pipelines with Caching - aKumoSolutions Dev - Medium. Medium. https://medium.com/akumosolutions-dev/speed-up-jenkins-pipelines-with-caching-b5fa650203a8#:~:text=Implementing%20caching%20in%20Jenkins%20pipelines,process%20more%20streamlined%20and%20responsive.
34) "Overview of best practices for writing Dockerfiles." (2024, March 22). Docker Documentation. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
35) Avi. (2022, November 4). Understanding Kubernetes architecture. Geekflare. https://geekflare.com/kubernetes-architecture/
36) Naveed, M. (2023, May 18). A Guide to Blue-Green Deployments with Kubernetes | Cloud Native Daily. Medium. https://medium.com/cloud-native-daily/blue-green-deployments-with-kubernetes-a-comprehensive-guide-5d196dad1976