**Documentation**

**Challenge 1: AI-Driven AKS Cluster Provisioning and Monitoring**

**1. Problem Statement**

Managing cloud infrastructure can be challenging, especially when there’s no effective way to monitor resources in real-time or roll back changes in case of failures. Many existing solutions focus on provisioning resources but often lack the ability to handle errors smoothly or revert to a stable state automatically. This not only leads to potential downtime but also results in wasted resources and unnecessary costs. Our project aims to tackle this issue by building a platform that integrates resource provisioning with real-time monitoring, error detection, and an automatic rollback mechanism to ensure seamless cloud operations on Azure.

**2. Introduction**

In today’s fast-paced digital world, managing cloud infrastructure efficiently is crucial for businesses. With the increasing demand for scalable and secure systems, it has become more important than ever to streamline the process of provisioning and managing cloud resources. Our project, developed during the hackathon, focuses on automating the lifecycle management of Azure cloud infrastructure. By integrating Terraform and Python with a Flask API, we aim to not only simplify the provisioning of resources but also introduce real-time monitoring, error handling, and automated rollbacks. This ensures that cloud operations are secure, scalable, and easy to manage, reducing the need for manual intervention and improving overall system reliability.

**3. Objectives**

The key objectives of this project are to:

**Automate Cloud Resource Provisioning:**

Streamline the process of provisioning Azure cloud resources, including virtual machines and Kubernetes clusters, by integrating Terraform and Python with a Flask API. This automation reduces manual intervention and ensures consistency across deployments.

**Enable Real-Time Monitoring:**

Implement real-time monitoring solutions to track the performance and resource usage of cloud infrastructure, ensuring that potential issues are detected early and addressed promptly.

**Introduce Automated Rollbacks:**

Develop an automated rollback mechanism that triggers in case of provisioning errors or failures, allowing the system to revert to a stable state and minimize downtime.

**Ensure Secure and Scalable Infrastructure:**

Design and maintain secure cloud environments that can handle the dynamic scaling of resources while adhering to security standards and best practices, ensuring the infrastructure is both scalable and protected.

**Key Concepts:**

**Azure Cloud**

Azure is a cloud computing platform that provides a wide range of services for deploying, managing, and scaling applications. In this project, we use Azure for provisioning virtual machines and Kubernetes clusters, leveraging its robust infrastructure for scalable cloud operations.

**Terraform**

Terraform is an open-source Infrastructure as Code (IaC) tool that allows users to define and provision cloud resources through declarative configuration files. We use Terraform to automate the provisioning of cloud resources on Azure, reducing manual setup and ensuring consistency across environments.

**Flask API**

Flask is a lightweight Python web framework used to build APIs. In our project, we developed a Flask-based API to interact with Terraform, enabling users to trigger provisioning tasks, monitor the status of cloud resources, and handle rollbacks in case of errors.

**Real-Time Monitoring**

Monitoring is critical to ensure that cloud infrastructure is running efficiently. Our project integrates real-time monitoring solutions that track key metrics such as CPU usage, memory consumption, and network traffic, helping to identify potential issues and optimize resource usage.

**Automated Rollbacks**

Rollbacks are essential for minimizing the impact of errors during cloud provisioning. Our system is equipped with an automated rollback feature that detects failures and reverts to the last stable state, reducing downtime and resource wastage.

**Our Approach to the Problem:-**

To address the challenges of provisioning and managing cloud infrastructure at scale, we structured our solution into several key phases:

**Automated Provisioning**

We began by automating the cloud resource provisioning process using Terraform integrated with a Flask API. This allowed us to dynamically create Azure resources like virtual machines and Kubernetes clusters with just a few API calls, reducing manual intervention and ensuring consistent, repeatable deployments.

**Real-Time Monitoring Integration**

Once the infrastructure was provisioned, we implemented real-time monitoring to track the performance and health of the resources. By monitoring key metrics like CPU usage, memory consumption, and network traffic, we ensured that the system was running optimally and could detect any potential issues early.

**Error Handling and Automated Rollbacks**

To enhance system resilience, we introduced an error detection mechanism that continuously checks for issues during the provisioning process. In the event of a failure, the system automatically rolls back to a stable state, minimizing downtime and resource wastage.

**Security and Scalability**

We focused on building a secure and scalable architecture. By integrating network security groups and private endpoints, we ensured that all communication between resources was secure. Additionally, the infrastructure was designed to scale dynamically, handling increased workloads without compromising performance.

**End-to-End Workflow Automation**

Finally, we brought everything together into a seamless, API-driven workflow. Users can provision, monitor, and manage cloud resources through simple API calls, with Terraform and Flask handling the heavy lifting in the background.

**Task Overview:**

The primary goal of this project was to automate the lifecycle management of cloud resources on Azure using Terraform and a Flask API. The tasks included developing API routes for provisioning virtual machines and Kubernetes clusters, implementing real-time monitoring of resources, and integrating automated rollback mechanisms for error handling. Additionally, the system was designed with security in mind, incorporating secure networking environments and scaling capabilities. Each task was aimed at streamlining the cloud provisioning process, ensuring operational efficiency, and maintaining a secure, scalable infrastructure.

**Automating Azure Resource Provisioning with User-Friendly Management**

This document outlines the development of a system for automating Azure resource provisioning, progressing through three distinct versions to enhance functionality and user experience.

**Version 1: Laying the Foundation with Automation**

Focus: This initial version established a foundation for automating Azure resource provisioning using Terraform.

**API Development:** A REST API was created to interface with Terraform configuration files. Initially, these files contained hardcoded values, but later evolved to accept dynamic inputs through JSON payloads.

**Supported Resources:** The API facilitated provisioning of various Azure resources, including virtual machines, Kubernetes clusters, and Databricks clusters.

**Initial Exploration:** Manual resource provisioning on Azure was conducted to identify key input parameters and ensure successful replication through Terraform.

**Hardcoded Terraform Configuration:** To test the API's functionality, a Terraform file with hardcoded values was integrated.

**Dynamic Input Handling:** The system transitioned to accept user-defined parameters via API calls for dynamic resource creation.

**Rollback Mechanism:** A critical component was implemented to handle errors during resource creation. This ensures automatic deletion of any partially created resources, maintaining infrastructure consistency.

**API Payload Structure:** JSON payloads containing necessary configuration details were used for provisioning different Azure resources. Examples were provided for virtual machines, Kubernetes clusters, and Databricks clusters.

**Workflow Overview:** A detailed breakdown of the workflow explained the interaction between the API client, API server, Terraform execution, and response generation.

**Version 2: Empowering Users with a Management Interface**

Focus: Version 2 introduced a user interface to simplify virtual machine management.

**User Interface Development:** A user-friendly interface with four key buttons was created: Submit, Destroy, Monitor, and Download PEM.

**Submit Button:** This button streamlined virtual machine provisioning in Azure.

**Destroy Button:** Users could directly delete a virtual machine with this button.

**Monitor Button:** Clicking this button provided access to a new page displaying real-time metrics for the virtual machine.

**Download PEM Button:** This feature facilitated the download of the virtual machine's secret key in PEM format.

**Enhanced Monitoring:** A new endpoint was implemented to continuously fetch metrics from the Azure REST API, enabling real-time monitoring of CPU usage, available memory, and network activity.

**Automatic Token Generation:** Seamless access to the metrics API was ensured through the implementation of an automatic token generation mechanism.

**Static HTML Integration:** Two static HTML files were introduced: index.html serving as the home page for user input, and monitor.html for displaying the fetched metrics.

Additionally, we implemented the endpoint http://localhost:5000/monitor, which continuously retrieves metrics from the Azure REST API at the following URL:

perl

Copy code

https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Compute/virtualMachines/{VM\_name}/providers/microsoft.insights/metrics?api-version=2018-01-01&metricnames=Percentage%20CPU,Available%20Memory%20Bytes,Network%20In%20Total,Network%20Out%20Total&timespan=PT1H&aggregation=Average&interval=PT1H

This endpoint fetches metrics such as CPU usage, available memory, and network activity.

To ensure seamless access to the metrics API, we implemented an automatic token generation mechanism using the following endpoint for acquiring the access token:

bash

Copy code

https://login.microsoftonline.com/{subscriptionId}/oauth2/v2.0/token

Whenever the token expires, a new token is generated automatically to maintain continuous access.

We introduced two static HTML files:

index.html: Serves as the home page for user input.

monitor.html: Displays the fetched metrics on-screen, updating every 20 seconds.

Finally, we added the functionality to download the PEM file via the Download PEM button.

With these enhancements, users can easily manage virtual machines, monitor their performance, and download necessary credentials, all from a streamlined interface.

**Version 3: Leveraging AI for Predictive Insights**

**Focus:** This final version incorporated AI capabilities for analyzing and responding to virtual machine metrics, providing proactive recommendations.

**New API Endpoint:** A new API endpoint, /predict, was introduced to handle fetching VM metrics and utilizing AI for analysis.

**AI Integration:** The predict.py script handled the back-end logic, fetching metrics, passing them to the AI model (Gemini), and generating predictions for users.

**AI Model Interaction:** The ChatGoogleGenerativeAI model from the LangChain library was employed to interact with Gemini.

**Metric Collection:** Real-time VM metrics like CPU usage, available memory, and network traffic were collected via the Azure Monitor Metrics API.

**AI Predictions:** The Gemini model processed these metrics to provide three key insights:

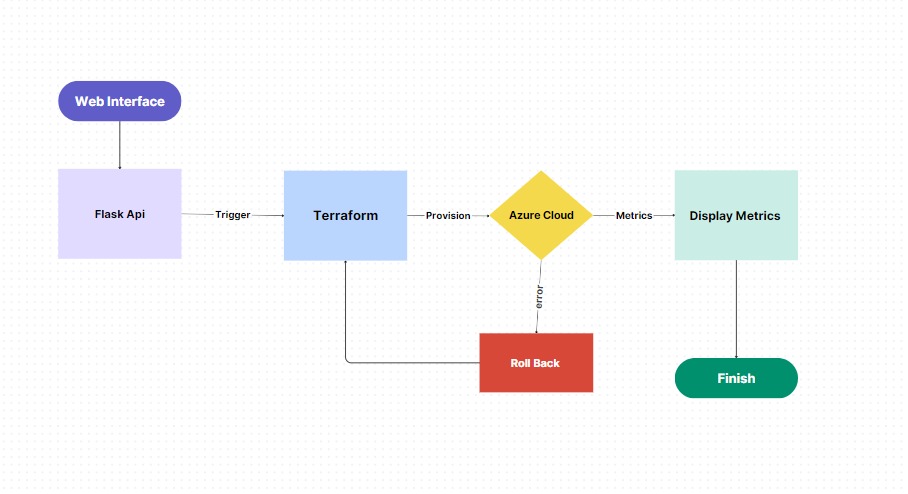
**Predictions:** Anticipated future performance issues based on current trends.

**Prescriptions:** Offered recommendations for optimizing system performance.

**Prevention:** Highlighted potential risks leading to downtime and suggested preventive measures.

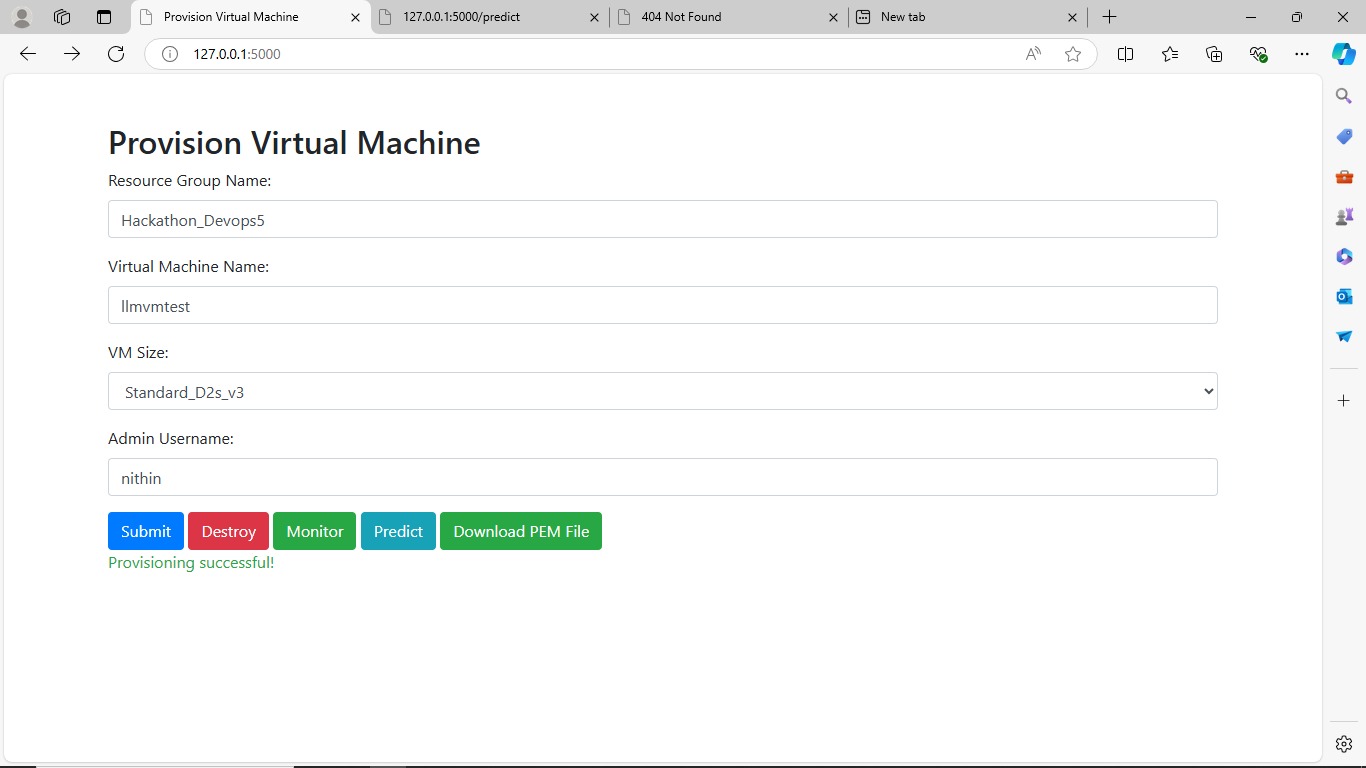
**User Interface Enhancements:** The predict.html front-end component displayed the results and suggestions generated by the AI.

**Results& Discussions:-**



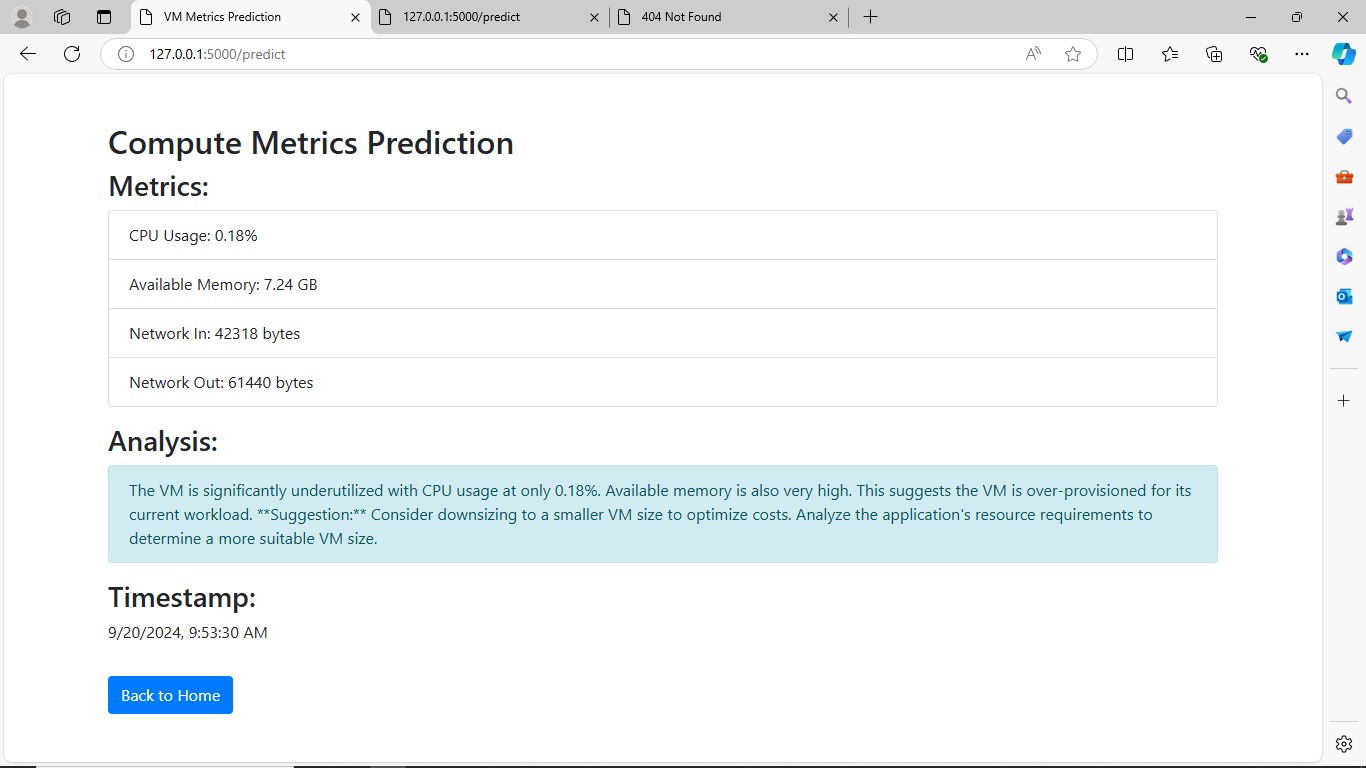
**Fig:1 Cloud Resource Provisioning Workflow**

This diagram illustrates a typical workflow for deploying infrastructure on Azure using Terraform. It starts with a web interface that triggers a Flask API to initiate the deployment process. Terraform is then used to provision the necessary resources on Azure Cloud. Metrics are collected and displayed to monitor the deployment. If an error occurs, a roll-back mechanism is implemented to revert the changes. Finally, the deployment is marked as finished upon successful completion.

****

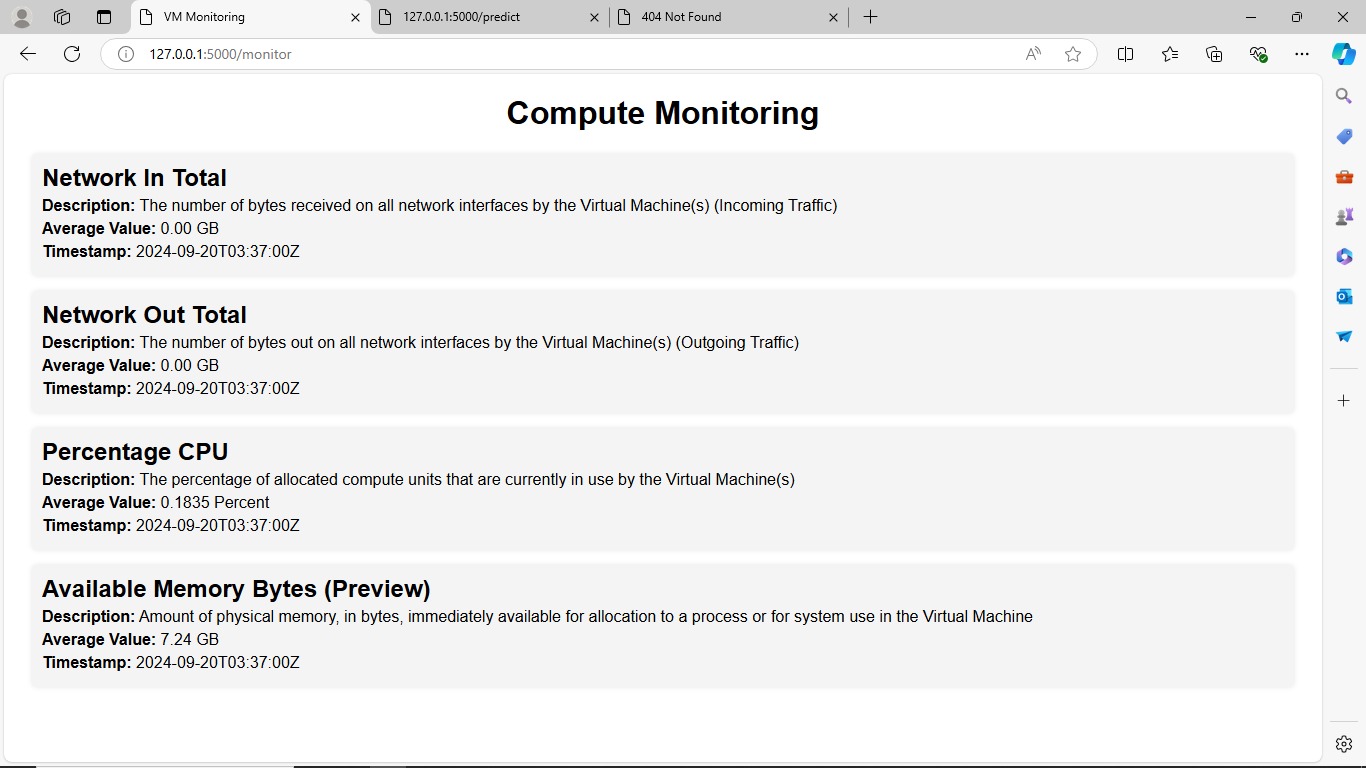
**Fig:2. Provision Virtual Machine Form Virtual Machine Provisioning Interface VM Deployment Form**

This is a web page where users can provision virtual machines (VMs). The page allows users to enter information such as the resource group, virtual machine name, VM size, and admin username. Once the information is entered, users can submit the form to provision the VM. The page also provides buttons to monitor, predict, download the PEM file, and destroy the VM.

****

**Fig:3. VM Performance Predictor**

This is a web page showing the predicted metrics for a virtual machine (VM). The page displays the VM's current CPU usage, available memory, and network in/out. It also provides an analysis suggesting that the VM might be over-provisioned and recommends downsizing to a smaller VM size to optimize costs. The timestamp indicates when the prediction was made.



**Fig:4. VM Monitoring Dashboard**

This is a web page showing the monitoring metrics for a virtual machine (VM). The page displays the VM's network in, network out, CPU usage, and available memory. The data is presented in a tabular format with descriptions and average values for each metric. The timestamp indicates when the data was collected.

**Conclusion:**

This document has presented the incremental development of a system for automating Azure resource provisioning, culminating in a user-friendly platform powered by AI for proactive management of virtual machines. With each version, the system has grown in functionality and user experience.