Project Title: Centralized PC Inventory Management System

Objective

To develop and deploy a centralized web application enabling IT administrators to efficiently retrieve and manage detailed information about PCs within the organization. The application aims to streamline the process of querying, monitoring, and managing PC assets, ensuring efficient inventory management and compliance with organizational policies.

1. Introduction

- Project Background:
    - Purpose: Large organizations with hundreds of PCs often struggle with efficient inventory management. This project addresses the need for a centralized system that automates the retrieval and management of PC details, ensuring compliance with company policies and reducing time spent on manual asset tracking.
    - Project Scope: The system covers all stages of asset management, including data collection, storage, processing, and retrieval. It interfaces with multiple components across the organization, ensuring that all information is up-to-date and accessible to authorized personnel.

2. System Architecture

- High-Level Architecture:
    - Presentation Layer: Angular 12 / React 17
    - Application Layer: Spring Boot 2.5.4
    - Data Layer: MySQL 8.0.26 / PostgreSQL 13
- Detailed Component Overview:
    - Frontend: Built with Angular/React, providing a user-friendly interface for querying and managing PC data.
    - Backend: Spring Boot application handles the business logic, including data validation, session management, and API communication.
    - Database: MySQL/PostgreSQL database stores detailed information about each PC, including hardware specifications, software inventory, and usage history.
    - Security: Authentication and authorization are managed using Spring Security 5.5.1, with OAuth2 and JWT tokens ensuring secure access.

3. Infrastructure Setup

- Kubernetes Cluster:
    - Cluster Overview:
        - Region: AWS Asia Pacific (Mumbai) region (ap-south-1)
        - Master Node: 1 node (t3.large)
        - Worker Nodes: 9 nodes (6 t3.medium, 3 t3.large)
        - Kubernetes Version: 1.21
        - Docker Version: 20.10.7
    - Pod Configuration:
        - Total Pods: 150+ pods.
        - Containers per Pod: Typically 2-3 containers.
        - Scaling Mechanism: Horizontal Pod Autoscaler (HPA) based on CPU/memory usage.
- Networking and Security:

    - VPC Configuration:
        - VPC: Custom VPC with CIDR block 10.0.0.0/16.
        - Subnets: 4 public subnets (10.0.1.0/24, 10.0.2.0/24, 10.0.3.0/24, 10.0.4.0/24) and 4 private subnets (10.0.5.0/24, 10.0.6.0/24, 10.0.7.0/24, 10.0.8.0/24).
        - Security Groups: Defined to allow specific traffic only, ensuring a secure environment.
        - Ingress Controller: NGINX Ingress Controller (Version: 1.0.0) for managing external traffic.
        - Load Balancer: AWS ELB for distributing traffic.
- Storage Solutions:
    - Persistent Volumes: AWS EBS volumes with custom Storage Classes.
    - Data Backup: Automated daily backups using AWS Backup service.
- Disaster Recovery (DR):
    - Backup Strategy: Daily incremental backups with weekly full backups, stored in a separate AWS region (e.g., Asia Pacific (Singapore) region - ap-southeast-1).
    - DR Drill: Quarterly disaster recovery drills to ensure data integrity and service availability.
    - Failover Mechanism: Cross-region replication for critical data and automatic failover to a secondary Kubernetes cluster in case of a major outage.

4. CI/CD Pipeline

- Pipeline Overview:
    - Tools and Versions:
        - Jenkins: 2.303.1
        - Maven: 3.8.1
        - SonarQube: 8.9.2
        - OWASP ZAP: 2.10.0
        - Helm: 3.6.3
- Pipeline Stages:

1. Code Checkout: Retrieve the latest code from the GitHub repository.
2. Build with Maven: Compile the code and package it as a JAR file.
3. Static Code Analysis with SonarQube: Ensure code quality by running static analysis.
4. Security Testing with OWASP ZAP: Run automated security tests to detect vulnerabilities.
5. Docker Build & Push: Build the Docker image and push it to Docker Hub.
6. Deploy to Kubernetes: Deploy the application to the Kubernetes cluster using Helm.

- Pipeline Code:

groovy

Copy code

```
pipeline {
    agent any

    environment {
        DOCKER_CREDENTIALS_ID = 'dockerhub-credentials'
        DOCKER_IMAGE = "myorg/pc-inventory-app"
```

```
        GIT_REPO = "https://github.com/your-repo/pc-inventory.git"
        BRANCH_NAME = "main"
    }

    stages {
        stage('Checkout Code') {
            steps {
                git branch: "${env.BRANCH_NAME}", url: "${env.GIT_REPO}"
            }
        }

        stage('Build with Maven') {
            steps {
                sh 'mvn clean install'
            }
        }

        stage('Static Code Analysis with SonarQube') {
            steps {
                script {
                    withSonarQubeEnv('SonarQube') {
                        sh 'mvn sonar:sonar'
                    }
                }
            }
        }

        stage('Security Testing with OWASP ZAP') {
            steps {
                script {
                    sh 'zap-cli quick-scan --self-contained --start-options "-config api.key=12345"
http://myapp-url.com'
                }
            }
        }

        stage('Docker Build & Push') {
            steps {
                script {
                    docker.build("${env.DOCKER_IMAGE}:${env.BRANCH_NAME}")
                        .withRegistry('', "${env.DOCKER_CREDENTIALS_ID}")
                        .push()
                }
            }
```

```
        }

        stage('Deploy to Kubernetes') {
            steps {
                script {
                    sh 'helm upgrade --install pc-inventory-app ./helm/pc-inventory-app'
                }
            }
        }
    }

    post {
        always {
            cleanWs()
        }
    }
}
```

5. Environment Flow

- Development Environment:
    - Tools: IntelliJ IDEA (Version: 2021.2), Minikube (Version: 1.22), Docker Desktop (Version: 3.5.2)
    - Database: MySQL/PostgreSQL with sample data for testing.
- Preproduction Environment:
    - Cluster Size: 3 nodes (t3.medium)
    - Pods: 30+ pods.
    - Access: Restricted to QA team and DevOps engineers for testing and validation.
- Production Environment:
    - Region: AWS Asia Pacific (Mumbai) region (ap-south-1)
    - Cluster Size: 9 nodes (t3.medium and t3.large)
    - Pods: 150+ pods distributed across the nodes.
    - High Availability: Configured with multiple instances to ensure uptime and reliability.
    - Disaster Recovery: Cross-region replication with automatic failover mechanisms in place.

6. Monitoring and Logging

- Monitoring Tools:
    - Prometheus (Version: 2.29.1): Collects and stores metrics from the Kubernetes cluster and application components.
    - Grafana (Version: 8.1.2): Visualizes metrics collected by Prometheus, with dashboards set up for real-time monitoring.
- Logging Tools:
    - ELK Stack (Version: 7.13.4):

        - Elasticsearch: Stores logs.
        - Logstash: Collects and processes logs from various sources.
        - Kibana: Provides a web interface to visualize and analyze logs.
- Alerts and Notifications:
    - Alertmanager (Version: 0.22.2): Sends alerts based on Prometheus metrics.
    - Slack Integration: Alerts are pushed to a dedicated Slack channel for immediate response by the DevOps team.

7. Patch Management

- Patch Deployment:
    - Regular Updates: Monthly patch cycles are scheduled, with patches tested in the staging environment before production deployment.
    - Automation: Patches are deployed automatically using Jenkins pipelines, with rollback mechanisms in place in case of failures.
    - Compliance Monitoring: Regular audits ensure that all systems are patched according to organizational policies.

8. Challenges and Solutions

- Scaling Issues:
    - Challenge: Managing increasing load and scaling the application to serve 600+ clients.
    - Solution: Implemented Horizontal Pod Autoscaler (HPA) to automatically adjust the number of pods based on CPU and memory usage.
- Security Vulnerabilities:
    - Challenge: Identifying and mitigating security risks in the application.
    - Solution: Integrated OWASP ZAP into the CI/CD pipeline for automated security testing and regular vulnerability assessments.
- Resource Optimization:
    - Challenge: Efficient utilization of cluster resources to minimize costs.
    - Solution: Utilized spot instances for non-critical workloads and implemented Kubernetes resource requests and limits to optimize performance.

9. Team Structure

- DevOps Solution Team:
    - Size: 11-15 members
    - Responsibilities: Check and resolve bugs, provide solutions to users based on issues.
- Developer DevOps Team:
    - Size: 11-15 members
    - Responsibilities:
        - Architecture Design: Develop and maintain the architecture of the application.
        - Pipeline Scripts: Write and manage CI/CD pipeline scripts.
        - Docker Images: Manage Docker images and their deployment.
        - Kubernetes Configurations: Write and manage Kubernetes configuration files.
        - Server Management: Oversee server provisioning and management.

        - Cluster Management: Handle the management of Kubernetes clusters.
- DevOps Infrastructure Team:
    - Size: 7-8 members
    - Responsibilities: Focus on infrastructure setup and management.
- DevOps Monitoring Team:
    - Size: 4-5 members
    - Responsibilities: Manage Helm, Prometheus, and server monitoring.

10. Responsibilities of the Developer DevOps Team

- Architecture Design:
    - Collaborate with stakeholders to design scalable and resilient application architecture.
    - Ensure alignment with organizational standards and best practices.
- Pipeline Scripts:
    - Develop and maintain CI/CD pipelines for automating build, test, and deployment processes.
    - Integrate various tools (Jenkins, Maven, SonarQube) into the pipeline.
- Docker Images:
    - Create and manage Docker images for application deployment.
    - Ensure images are optimized and free from vulnerabilities.
- Kubernetes Configurations:
    - Write and maintain Kubernetes manifests for deploying and managing application workloads.
    - Configure services, ingress, and other Kubernetes resources.
- Server Management:
    - Provision and manage server resources, including configuring virtual machines and storage.
    - Ensure high availability and performance of the server infrastructure.
- Cluster Management:
    - Manage Kubernetes clusters, including node management and resource allocation.
    - Implement and oversee scaling policies and cluster maintenance activities.