Question 1 .... **(5 Marks)**

You have developed a Flask-based Python application that listens on port 8080, with the source code located in a local "app" folder. This application relies on two microservices: a private login service (myrepo/login-service:latest) that listens on port 8081, and a MySQL database, where the login service retrieves data from the MySQL database to operate. Since a Docker image for your Python application has not yet been created, how can you utilize Docker Compose to run your application along with its dependencies? Write the YAML configuration while considering relevant DevOps principles applied to this context.

Question 2 .... **(10 Marks)**

(a) **(5 Marks)** You are tasked with setting up an automated build process for an application that can be compiled for different platforms: Windows, macOS, and Linux. Each platform requires distinct build commands and tools. The team would like to initiate builds from Jenkins but needs a way to specify which platform to target for each build. How would you configure your Jenkins job to facilitate this requirement?

(b) **(5 Marks)** You are responsible for maintaining a Jenkins pipeline that automates the deployment of a web application. The deployment process includes a step where a custom version tag must be applied to the build artifacts. The development team frequently releases new versions, and they want to include a specific version identifier or release note with each deployment.

The team would like to trigger the deployment job in Jenkins and provide a custom text string that represents the version identifier or release note for that deployment. This string should be included in the deployment metadata and displayed in the application's dashboard after deployment.

How would you set up your Jenkins job to enable the development team to specify a version identifier or release note for each deployment, ensuring that this information

Continue...

252681

is captured and utilized effectively during the build process?

Question 3 .... (5 Marks)

Your organization is developing a large-scale microservices application with multiple repositories, each managed by different teams. The development teams have adopted a continuous integration and continuous delivery (CI/CD) approach, where changes are frequently made to their respective codebases.

Due to the architecture of the application, a change in one microservice may necessitate simultaneous updates in multiple other services, leading to tightly-coupled dependencies. The teams rely heavily on a centralized Jenkins server that polls all repositories for changes every minute to trigger builds and deployments. However, this polling strategy is causing several issues:

- The Jenkins server experiences performance degradation due to the high frequency of polling requests across numerous repositories.
- The build queue becomes congested, leading to delayed feedback for developers and increased build times.
- Frequent polling leads to unnecessary builds when changes are made that do not affect the main application, wasting resources and time. Considering these challenges, the teams are re-evaluating their approach to CI/CD in Jenkins.

What alternative strategies could be implemented to manage code changes more efficiently without relying on polling, and what advantages would those strategies offer in this scenario?

Question 4 .... (5 Marks)

You have been given the following Kubernetes YAML configuration for deploying a web application in a namespace called my-app. The configuration includes a Secret, a ConfigMap, a Deployment, and a Service. However, the YAML file contains several errors and misconfigurations that need to be identified and corrected to ensure the application can be accessed externally and functions correctly.

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-app

---
apiVersion: v1
kind: Secret
metadata:
  name: my-app-secret
  namespace: my-app
type: Opaque
```

0073

```
data:
  password: cGFzc3dvcmQ=  # base64 encoded 'password'
  username: dXNlcm5hbWU=   # base64 encoded 'username'

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-config
  namespace: my-app
data:
  APP_MODE: "production"
  APP_PORT: "8080"

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-deployment
  namespace: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        env: production
    spec:
      containers:
        - name: my-app-container
          image: my-app-image:latest
          ports:
            - containerPort: 80
          env:
            - name: DB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: my-app-secret
                  key: username
            - name: DB_PASSWORD
              valueFrom:
```

Continue...

0073

```
                secretKeyRef:
                  name: my-app-secret
                  key: password
            - name: CONFIG_FILE
              valueFrom:
                configMapKeyRef:
                  name: my-app-configuration
                  key: APP_MODE

---
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
  namespace: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 80
  type: ClusterIP
```