Problem Statement 3 (Technical):

Step #1:

Create a GoLang Program which reflects the current date & time and host it on GitHub

Push that code to DockerHub

In other words: Use docker to create a web application with date & time as the only content

Step #2:

Using the declarative approach to deploy the container with 2 replicas to k8s

Step #3:

Expose the app to the Internet (on WAN)

Resource Hint/Help:

For k8s resources, you can use Qwiklabs (https://www.qwiklabs.com/ (https://www.qwiklabs.com/)) it gives you around 30 to 60 mins of k8s resources or you can use your own GCP account or any online available platform like https://labs.play-with-k8s.com (https://labs.play-with-k8s.com), etc

GoLang Date/Time App Deployment on Kubernetes

1. Create GoLang Program

Create a file named main.go with the following code:

```
package main

import (
    "fmt"
    "net/http"
    "time"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Current Date and Time: %s", time.Now().Format(time.RFC1123))
}

func main() {
    http.HandleFunc("/", handler)
    fmt.Println("Server started on port 8080")
    http.ListenAndServe(":8080", nil)
}
```

This program will:

- Import the necessary packages: fmt for printing, net/http for creating a web server, and time for getting the current date and time.
- Define a handler function that writes the current date and time in RFC1123 format to the HTTP response.
- Define the main function that sets up a handler for the root path ("/"), starts the web server on port 8080, and prints a message to the console.

2. Create Dockerfile

Create a file named Dockerfile in the same directory as main.go with the following content:

```
FROM golang:1.22-alpine AS builder
WORKDIR /app
COPY go.mod go.sum ./
RUN go mod download
COPY . ./
RUN go build -o /bin/app

FROM alpine:latest
COPY --from=builder /bin/app /bin/app
EXPOSE 8080
CMD ["/bin/app"]
```

This Dockerfile will:

- Use the golang:1.22-alpine image as the base image for building the Go application.
- Set the working directory to /app.
- Copy the go.mod and go.sum files and download the dependencies.
- Copy the source code.
- Build the Go application and output the binary to /bin/app.
- Use the alpine:latest image as the base image for the final image.
- Copy the binary from the builder stage to /bin/app.
- Expose port 8080.
- Set the command to run the application.

3. Build and Push Docker Image

- Initialize a Go module:

```
go mod init myapp
```

- Build the Docker image:

```
docker build -t <your-dockerhub-username>/datetime-app:v1 .
```

Replace <your-dockerhub-username> with your DockerHub username.

- Log in to DockerHub:

```
docker login
```

- Push the Docker image to DockerHub:

```
docker push <your-dockerhub-username>/datetime-app:v1
```

4. Deploy to Kubernetes

Create a file named deployment.yaml with the following Kubernetes Deployment definition:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: datetime-app-deployment
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
```

```
    app: datetime-app
  template:
    metadata:
      labels:
        app: datetime-app
    spec:
      containers:
        - name: datetime-app
          image: <your-dockerhub-username>/datetime-app:v1
          ports:
            - containerPort: 8080
```

Replace <your-dockerhub-username> with your DockerHub username.
This Deployment will:

- Create a Deployment named datetime-app-deployment in the default namespace.
- Ensure that 2 replicas of the application are running.
- Use the datetime-app:v1 image from DockerHub.
- Expose port 8080 on the container.

Apply the Deployment:

```
kubectl apply -f deployment.yaml
```

5. Expose to the Internet

To expose the application to the internet, you can create a Kubernetes Service of type LoadBalancer. This will provision an external load balancer (if your Kubernetes cluster is running in a cloud provider like GCP, AWS, or Azure) and make your application accessible via an external IP address.
Create a file named service.yaml with the following Kubernetes Service definition:

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

This Service will:

- Create a Service named datetime-app-service in the default namespace.
- Select the Pods with the label app: datetime-app (which are the Pods created by the Deployment).
- Forward traffic on port 80 to port 8080 on the Pods.
- Use a LoadBalancer to expose the service externally.
  Apply the Service:

```
kubectl apply -f service.yaml
```

Get the external IP address of the Service:

```
kubectl get service datetime-app-service
```

The output will show the EXTERNAL-IP address. You can access the application in your browser using this IP address. It might take a few minutes for the load balancer to be provisioned and the IP address to become available.

Note:

- If you are using Minikube, it does not directly support the LoadBalancer type. You can use minikube service datetime-app-service --url to get a URL to access the service.
- If you are using a local Kubernetes cluster (like kind), you might need to use an Ingress controller or port forwarding to expose the service.
- For production deployments, consider using Ingress with a proper domain name and TLS certificate for security.