### Task 1: Create a GKE Cluster ###

To create a GKE cluster with the provided configurations, follow these steps:

1. \*\*Log in to Google Cloud Console\*\* with Cymbal Owner credentials.

2. \*\*Open Cloud Shell\*\* or use your local terminal with `gcloud` installed.

3. \*\*Set the required configurations\*\*:

- Replace `ZONE` with the desired Google Cloud zone.

- Replace `PROJECT\_ID` with your Google Cloud project ID.

4. \*\*Create the GKE Cluster\*\*:

- Use the following `gcloud` command to create the cluster with the specified configurations:

```sh

gcloud container clusters create hello-world-v45n \

--zone us-east1-c \

--release-channel regular \

--num-nodes 3 \

--enable-autoscaling \

--min-nodes 2 \

--max-nodes 6

```

Replace `ZONE` and `PROJECT\_ID` with the appropriate values for your environment.

### Task 2: Enable Managed Prometheus on the Cluster

To enable Managed Prometheus on your GKE cluster and deploy the sample Prometheus application and pod monitoring, follow these steps:

### Step 1: Enable Managed Prometheus on the GKE Cluster

1. \*\*Enable the Prometheus managed collection\*\* on the GKE cluster:

```sh

gcloud container clusters update <cluster-name> \

--zone <zone> \

--monitoring=SYSTEM,WORKLOAD \

--logging=SYSTEM,WORKLOAD

```

Replace `<cluster-name>` and `<zone>` with your actual cluster name and zone.

### Step 2: Create a Namespace

1. \*\*Create a namespace\*\* on the cluster:

```sh

kubectl create namespace <namespace-name>

```

Replace `<namespace-name>` with your desired namespace name.

### Step 3: Download and Update Prometheus App

1. \*\*Download the sample Prometheus app\*\*:

```sh

gsutil cp gs://spls/gsp510/prometheus-app.yaml .

```

2. \*\*Update the `prometheus-app.yaml` file\*\*:

Open the `prometheus-app.yaml` file and update the `<todo>` sections with the following configuration:

```yaml

containers:

- name: prometheus-test

image: nilebox/prometheus-example-app:latest

ports:

- name: metrics

containerPort: 8080

```

### Step 4: Deploy the Prometheus Application

1. \*\*Deploy the application\*\* onto the `<namespace-name>` namespace:

```sh

kubectl apply -f prometheus-app.yaml --namespace <namespace-name>

```

### Step 5: Download and Update Pod Monitoring Configuration

1. \*\*Download the pod-monitoring.yaml file\*\*:

```sh

gsutil cp gs://spls/gsp510/pod-monitoring.yaml .

```

2. \*\*Update the `pod-monitoring.yaml` file\*\*:

Open the `pod-monitoring.yaml` file and update the `<todo>` sections with the following configuration:

```yaml

metadata:

name: prometheus-test

spec:

selector:

matchLabels:

app: prometheus-test

endpoints:

- port: metrics

interval: 30s

```

### Step 6: Apply the Pod Monitoring Resource

1. \*\*Apply the pod monitoring resource\*\* onto the `<namespace-name>` namespace:

```sh

kubectl apply -f pod-monitoring.yaml --namespace <namespace-name>

```

### Verification

Once you have completed these steps, you can verify the objective by checking the progress in your lab environment. These steps will enable Managed Prometheus on your GKE cluster, create the necessary namespace, deploy the Prometheus application, and apply the pod monitoring configuration.

### Task 3: Deploy a Kubernetes Manifest and Debug Errors

1. \*\*Download the Kubernetes Manifest\*\*:

- Ensure you have access to the manifest file. If not, create a basic example.

```yaml

apiVersion: v1

kind: Pod

metadata:

name: hello-app

spec:

containers:

- name: hello-app

image: "gcr.io/spls/gsp510/hello-app"

ports:

- containerPort: 8080

```

2. \*\*Deploy the Manifest\*\*:

- Use the following command to deploy the manifest:

```sh

kubectl apply -f hello-app.yaml

```

3. \*\*Debug Deployment Errors\*\*:

- Check the status of the pod:

```sh

kubectl get pods

```

- Describe the pod to see detailed error messages if any:

```sh

kubectl describe pod hello-app

```

### Task 4: Create a Logs-Based Metric and Alerting Policy

1. \*\*Create a Logs-Based Metric\*\*:

- Navigate to \*\*Logging\*\* > \*\*Logs-based metrics\*\*.

- Create a new logs-based metric that filters for error logs in the Kubernetes cluster.

2. \*\*Create an Alerting Policy\*\*:

- Navigate to \*\*Monitoring\*\* > \*\*Alerting\*\*.

- Create a new alerting policy based on the logs-based metric you created.

### Task 5: Fix Manifest Errors, Containerize Application, and Push to Artifact Registry

1. \*\*Fix Manifest Errors\*\*:

- Ensure the manifest file is correct and addresses any errors identified in the debugging step.

2. \*\*Containerize the Application\*\*:

- Use Docker to build and push the application to Artifact Registry.

```sh

docker build -t "gcr.io/PROJECT\_ID/hello-app:v1" .

docker push "gcr.io/PROJECT\_ID/hello-app:v1"

```

3. \*\*Update the Manifest\*\*:

- Update the manifest file to use the new image:

```yaml

apiVersion: v1

kind: Pod

metadata:

name: hello-app

spec:

containers:

- name: hello-app

image: "gcr.io/PROJECT\_ID/hello-app:v1"

ports:

- containerPort: 8080

```

4. \*\*Deploy the Updated Manifest\*\*:

```sh

kubectl apply -f hello-app.yaml

```

### Task 6: Expose a Service and Verify Updates

1. \*\*Create a Service\*\*:

- Expose the application using a Kubernetes service:

```yaml

apiVersion: v1

kind: Service

metadata:

name: hello-app-service

spec:

selector:

app: hello-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

type: LoadBalancer

```

2. \*\*Deploy the Service\*\*:

```sh

kubectl apply -f hello-app-service.yaml

```

3. \*\*Verify the Service\*\*:

- Get the external IP of the service and verify it:

```sh

kubectl get services

```

- Access the application via the external IP to ensure it is running correctly.

By following these steps, you should be able to complete the tasks and demonstrate your skills in managing Kubernetes deployments.

**Task 2. Enable Managed Prometheus on the GKE cluster**

To enable Managed Prometheus on your GKE cluster and deploy a sample Prometheus application, follow these steps:

### Step 1: Enable Managed Prometheus

1. \*\*Open Cloud Shell\*\* or use your local terminal with `gcloud` installed.

2. \*\*Set the required configurations\*\*:

- Replace `ZONE` with the appropriate Google Cloud zone.

- Replace `CLUSTER\_NAME` with your GKE cluster name.

- Replace `PROJECT\_ID` with your Google Cloud project ID.

3. \*\*Enable Managed Prometheus\*\*:

```sh

gcloud container clusters update CLUSTER\_NAME \

--zone ZONE \

--enable-managed-prometheus \

--project PROJECT\_ID

```

### Step 2: Create a Namespace

1. \*\*Create the namespace\*\*:

- Replace `NAMESPACE\_NAME` with your desired namespace name.

```sh

kubectl create namespace NAMESPACE\_NAME

```

### Step 3: Download and Update Prometheus App Configuration

1. \*\*Download the sample Prometheus app configuration\*\*:

```sh

gsutil cp gs://spls/gsp510/prometheus-app.yaml .

```

2. \*\*Update the `prometheus-app.yaml` file\*\*:

- Open the `prometheus-app.yaml` file in a text editor.

- Update lines 35-38 as follows:

```yaml

containers:

- image: nilebox/prometheus-example-app:latest

name: prometheus-test

ports:

- name: metrics

containerPort: 80

```

3. \*\*Deploy the Prometheus app\*\*:

```sh

kubectl apply -f prometheus-app.yaml --namespace=NAMESPACE\_NAME

```

### Step 4: Download and Update Pod Monitoring Configuration

1. \*\*Download the pod monitoring configuration\*\*:

```sh

gsutil cp gs://spls/gsp510/pod-monitoring.yaml .

```

2. \*\*Update the `pod-monitoring.yaml` file\*\*:

- Open the `pod-monitoring.yaml` file in a text editor.

- Update lines 18-24 as follows:

```yaml

metadata:

name: prometheus-test

labels:

app.kubernetes.io/name: prometheus-test

spec:

selector:

matchLabels:

app: prometheus-test

endpoints:

- port: metrics

interval: 30s

```

3. \*\*Apply the pod monitoring resource\*\*:

```sh

kubectl apply -f pod-monitoring.yaml --namespace=NAMESPACE\_NAME

```

### Verify the Setup

1. \*\*Check the namespace\*\*:

```sh

kubectl get namespaces

```

2. \*\*Check the deployment\*\*:

```sh

kubectl get deployments --namespace=NAMESPACE\_NAME

```

3. \*\*Check the pods\*\*:

```sh

kubectl get pods --namespace=NAMESPACE\_NAME

```

4. \*\*Check the pod monitoring configuration\*\*:

```sh

kubectl get servicemonitor --namespace=NAMESPACE\_NAME

```

By following these steps, you should successfully enable Managed Prometheus on your GKE cluster and deploy the sample Prometheus application along with the necessary monitoring configuration.

## Task 3. Deploy an application onto the GKE cluster

To deploy an application onto the GKE cluster, create a logs-based metric and alerting policy, and fix the invalid image name error, follow these steps:

### Step 1: Download the Demo Deployment Manifest Files

1. \*\*Open Cloud Shell\*\* or use your local terminal with `gcloud` installed.

2. \*\*Download the demo deployment manifest files\*\*:

```sh

gsutil cp -r gs://spls/gsp510/hello-app/ .

```

### Step 2: Create the Deployment

1. \*\*Navigate to the directory containing the manifest files\*\*:

```sh

cd hello-app/manifests

```

2. \*\*Create the deployment\*\*:

- Replace `NAMESPACE\_NAME` with your namespace name.

```sh

kubectl apply -f helloweb-deployment.yaml --namespace=NAMESPACE\_NAME

```

3. \*\*Verify the deployment\*\*:

```sh

kubectl get deployments --namespace=NAMESPACE\_NAME

```

### Step 3: Check the Error and Create a Logs-Based Metric

1. \*\*Navigate to the GKE cluster\*\* in the Google Cloud Console.

2. \*\*Go to the "Workloads" section\*\* and find the `helloweb` deployment. You should see the invalid image name error.

### Step 4: Create a Logs-Based Metric and Alerting Policy

1. \*\*Open the Logs Explorer\*\* in the Google Cloud Console.

2. \*\*Filter the logs\*\* to find the invalid image name error:

```plaintext

resource.type="k8s\_container"

resource.labels.namespace\_name="NAMESPACE\_NAME"

resource.labels.container\_name="helloweb"

severity="ERROR"

```

3. \*\*Create a logs-based metric\*\*:

- Click on \*\*Create Metric\*\*.

- Name the metric `invalid-image-name-errors`.

- Set the filter to include the fields you identified.

4. \*\*Create an alerting policy\*\*:

- Go to the \*\*Monitoring\*\* section in the Google Cloud Console.

- Select \*\*Alerting\*\* > \*\*Create Policy\*\*.

- Add a condition based on the logs-based metric `invalid-image-name-errors`.

- Set up notifications and other configurations as required.

### Step 5: Fix the Invalid Image Name Error

1. \*\*Open the `helloweb-deployment.yaml` file\*\* in a text editor.

2. \*\*Find the `image` field\*\* and correct the image name:

```yaml

containers:

- name: helloweb

image: gcr.io/YOUR\_PROJECT\_ID/helloweb:v1 # Replace with the correct image name

```

3. \*\*Update the deployment\*\*:

```sh

kubectl apply -f helloweb-deployment.yaml --namespace=NAMESPACE\_NAME

```

4. \*\*Verify the deployment\*\* again:

```sh

kubectl get deployments --namespace=NAMESPACE\_NAME

```

### Step 6: Expose the Service and Verify

1. \*\*Expose the service\*\*:

```sh

kubectl expose deployment helloweb --type=LoadBalancer --name=helloweb-service --namespace=NAMESPACE\_NAME

```

2. \*\*Get the external IP\*\* of the service:

```sh

kubectl get services --namespace=NAMESPACE\_NAME

```

3. \*\*Verify the application\*\* by accessing the external IP in your browser.

By following these steps, you should successfully deploy the application onto the GKE cluster, create a logs-based metric and alerting policy, and fix the invalid image name error.

## Task 4. Create a logs-based metric and alerting policy

To create a logs-based metric and alerting policy, follow these steps:

### Step 1: Create a Logs-Based Metric

1. \*\*Open the Logs Explorer\*\* in the Google Cloud Console.

2. \*\*Create a query\*\* to expose the warnings/errors you saw in the previous section. The query should filter by Resource Type and Severity:

```plaintext

resource.type="k8s\_container"

severity="ERROR"

```

This query should show logs related to `InvalidImageName` and other relevant errors.

3. \*\*Run the query\*\* to verify that you see the expected errors, such as:

```plaintext

Error: InvalidImageName

Failed to apply default image tag "<todo>": couldn't parse image reference "<todo>": invalid reference format

```

4. \*\*Create a logs-based metric\*\* from this query:

- Click on \*\*Create Metric\*\*.

- Set the \*\*Metric type\*\* to \*\*Counter\*\*.

- Name the log metric `pod-image-errors`.

5. \*\*Save the metric\*\*.

### Step 2: Create an Alerting Policy

1. \*\*Go to the Monitoring section\*\* in the Google Cloud Console.

2. \*\*Navigate to Alerting\*\* and click on \*\*Create Policy\*\*.

3. \*\*Add a condition\*\*:

- Select \*\*Add Condition\*\*.

- Choose \*\*Log-based metric\*\* and select the `pod-image-errors` metric you just created.

- Configure the condition as follows:

- \*\*Rolling Window\*\*: 10 min

- \*\*Rolling window function\*\*: Count

- \*\*Time series aggregation\*\*: Sum

- \*\*Condition type\*\*: Threshold

- \*\*Alert trigger\*\*: Any time series violates

- \*\*Threshold position\*\*: Above threshold

- \*\*Threshold value\*\*: 0

4. \*\*Add a notification channel\*\*:

- For this task, disable the notification channel.

5. \*\*Name the alert policy\*\*:

- Alert policy name: `Pod Error Alert`

6. \*\*Save the alerting policy\*\*.

### Verification

To verify, click on the \*\*Check my progress\*\* button in the respective platform or interface.

By following these steps, you should be able to successfully create a logs-based metric and alerting policy to monitor and alert on Kubernetes pod errors and warnings.

## Task 5. Update and re-deploy your app

To update and re-deploy your app with the correct image, follow these steps:

### Step 1: Update the Deployment Manifest

1. \*\*Open the `helloweb-deployment.yaml`\*\* file in a text editor.

2. Replace the `<todo>` in the image section with the following image reference:

```yaml

image: us-docker.pkg.dev/google-samples/containers/gke/hello-app:1.0

```

The updated `helloweb-deployment.yaml` should look something like this:

```yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: helloweb

namespace: <namespace-name>

spec:

replicas: 3

selector:

matchLabels:

app: helloweb

template:

metadata:

labels:

app: helloweb

spec:

containers:

- name: helloweb

image: us-docker.pkg.dev/google-samples/containers/gke/hello-app:1.0

ports:

- containerPort: 8080

```

### Step 2: Delete the Existing Deployment

1. \*\*Open the Google Cloud Console\*\*.

2. Navigate to \*\*Kubernetes Engine\*\* > \*\*Workloads\*\*.

3. Find the `helloweb` deployment.

4. Click the \*\*three dots\*\* (menu) next to the `helloweb` deployment and select \*\*Delete\*\*.

5. Confirm the deletion.

### Step 3: Deploy the Updated Manifest

1. \*\*Open the Google Cloud Shell\*\*.

2. Navigate to the directory containing your updated `helloweb-deployment.yaml` file.

```sh

cd path/to/your/helloweb-deployment.yaml

```

3. Deploy the updated manifest onto your cluster in the specified namespace:

```sh

kubectl apply -f helloweb-deployment.yaml --namespace <namespace-name>

```

4. Verify that the deployment has been created successfully with no errors:

```sh

kubectl get deployments --namespace <namespace-name>

kubectl get pods --namespace <namespace-name>

```

5. Go to the \*\*Kubernetes Engine\*\* > \*\*Workloads\*\* page in the Google Cloud Console to ensure that the `helloweb` deployment is running without any issues.

By following these steps, you will have updated the image reference in your deployment manifest, deleted the existing deployment, and deployed the updated application onto your GKE cluster. Verify that the deployment is running correctly with no errors.

## Task 6. Containerize your code and deploy it onto the cluster

To containerize your application code, update an image in Artifact Registry, and deploy it onto the GKE cluster, follow these steps:

### Step 1: Update the Code

1. \*\*Open the `main.go` file\*\* in the `hello-app` directory.

2. Update line 49 to use `Version: 2.0.0`:

```go

fmt.Fprintf(w, "Hello, world!\nVersion: 2.0.0\nHostname: %s\n", hostname)

```

### Step 2: Build the Docker Image

1. \*\*Navigate to the `hello-app` directory\*\* in your terminal.

2. Build the Docker image with the `v2` tag:

```sh

docker build -t us-central1-docker.pkg.dev/your-project-id/repo-name/hello-app:v2 .

```

Replace `your-project-id` and `repo-name` with your actual project ID and repository name.

### Step 3: Push the Docker Image to Artifact Registry

1. \*\*Authenticate Docker with Artifact Registry\*\*:

```sh

gcloud auth configure-docker us-central1-docker.pkg.dev

```

2. \*\*Push the Docker image\*\* to your Artifact Registry repository:

```sh

docker push us-central1-docker.pkg.dev/your-project-id/repo-name/hello-app:v2

```

### Step 4: Update the Deployment Manifest

1. \*\*Open the `helloweb-deployment.yaml`\*\* file.

2. Update the image section to use the new image:

```yaml

image: us-central1-docker.pkg.dev/your-project-id/repo-name/hello-app:v2

```

3. \*\*Apply the updated manifest\*\* to your GKE cluster:

```sh

kubectl apply -f helloweb-deployment.yaml --namespace <namespace-name>

```

### Step 5: Expose the Deployment with a LoadBalancer Service

1. \*\*Create a service\*\* to expose your deployment:

```yaml

apiVersion: v1

kind: Service

metadata:

name: service-name

namespace: <namespace-name>

spec:

type: LoadBalancer

selector:

app: helloweb

ports:

- protocol: TCP

port: 8080

targetPort: 8080

```

2. \*\*Apply the service manifest\*\*:

```sh

kubectl apply -f service.yaml --namespace <namespace-name>

```

### Step 6: Verify the Deployment

1. \*\*Get the external IP address\*\* of the LoadBalancer service:

```sh

kubectl get services --namespace <namespace-name>

```

2. \*\*Navigate to the external IP address\*\* in your web browser. You should see the following text:

```

Hello, world!

Version: 2.0.0

Hostname: helloweb-6fc7476576-cvv5f

```

By following these steps, you will have successfully containerized your application code, pushed a new version to Artifact Registry, updated the deployment in your GKE cluster, and exposed the deployment using a LoadBalancer service. Verify that the application is running correctly and that the correct version is displayed.