**Task 1** and create a GKE cluster named `hello-world-ewk0` with the specified configurations, follow these steps:

### Step-by-Step Guide:

1. \*\*Set the necessary environment variables:\*\*

```sh

export CLUSTER\_NAME=hello-world-ewk0

export ZONE=us-central1-c

export MIN\_NODES=2

export MAX\_NODES=6

export NUM\_NODES=3

```

2. \*\*Set the compute zone:\*\*

```sh

gcloud config set compute/zone $ZONE

```

3. \*\*Create the GKE cluster:\*\*

```sh

gcloud container clusters create $CLUSTER\_NAME \

--zone $ZONE \

--release-channel regular \

--cluster-version 1.27.8 \

--enable-autoscaling \

--num-nodes $NUM\_NODES \

--min-nodes $MIN\_NODES \

--max-nodes $MAX\_NODES

```

### Explanation:

- `gcloud container clusters create`: This command creates a new GKE cluster.

- `--zone $ZONE`: Specifies the zone for the cluster.

- `--release-channel regular`: Uses the Regular release channel for cluster updates.

- `--cluster-version 1.27.8`: Specifies the cluster version.

- `--enable-autoscaling`: Enables autoscaling for the cluster.

- `--num-nodes $NUM\_NODES`: Sets the initial number of nodes to 3.

- `--min-nodes $MIN\_NODES`: Sets the minimum number of nodes to 2.

- `--max-nodes $MAX\_NODES`: Sets the maximum number of nodes to 6.

### Verify the Objective:

After running the above commands, click "Check my progress" in the provided sandbox environment to verify that you have successfully created the GKE cluster with the specified configurations.

**Task 2**

### Step-by-Step Guide:

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

1. \*\*Enable Prometheus managed collection:\*\*

```sh

gcloud container clusters update $CLUSTER\_NAME --zone=$ZONE --enable-managed-prometheus

```

#### Step 2: Create a Namespace

1. \*\*Create the namespace named `gmp-8uj9`:\*\*

```sh

kubectl create namespace gmp-8uj9

```

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

1. \*\*Download the Prometheus application manifest:\*\*

```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 (lines 35-38) with the following configurations:

```yaml

containers:

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

name: prometheus-test

ports:

- name: metrics

containerPort: 1234

```

3. \*\*Deploy the updated application onto the `gmp-8uj9` namespace:\*\*

```sh

kubectl apply -f prometheus-app.yaml --namespace=gmp-8uj9

```

#### 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 and update the `<todo>` sections (lines 18-24) with the following configurations:

```yaml

metadata:

name: prometheus-test

labels:

app.kubernetes.io/name: prometheus-test

spec:

selector:

matchLabels:

app: prometheus-test

endpoints:

- port: metrics

interval: 50s

```

3. \*\*Apply the pod monitoring configuration onto the `gmp-8uj9` namespace:\*\*

```sh

kubectl apply -f pod-monitoring.yaml --namespace=gmp-8uj9

```

### Summary of Commands

```sh

# Enable Prometheus Managed Collection

gcloud container clusters update $CLUSTER\_NAME --zone=$ZONE --enable-managed-prometheus

# Create Namespace

kubectl create namespace gmp-8uj9

# Download and Update Prometheus App

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

sed -i 's|<todo>|nilebox/prometheus-example-app:latest|g' prometheus-app.yaml

sed -i 's|<todo>|prometheus-test|g' prometheus-app.yaml

sed -i 's|<todo>|metrics|g' prometheus-app.yaml

kubectl apply -f prometheus-app.yaml --namespace=gmp-8uj9

# Download and Update Pod Monitoring Configuration

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

sed -i 's|<todo>|prometheus-test|g' pod-monitoring.yaml

sed -i 's|<todo>|app.kubernetes.io/name: prometheus-test|g' pod-monitoring.yaml

sed -i 's|<todo>|app: prometheus-test|g' pod-monitoring.yaml

sed -i 's|<todo>|50s|g' pod-monitoring.yaml

kubectl apply -f pod-monitoring.yaml --namespace=gmp-8uj9

```

### Verify the Objective

After running the above commands, you can verify your progress by checking the status of the deployments and configurations in the provided sandbox environment.

### **Task 3: Deploy an Application onto the GKE Cluster**

In this task, you will deploy a Kubernetes manifest onto the cluster and inspect the issue.

#### Step-by-Step Guide:

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

1. \*\*Download the `hello-app` directory:\*\*

```sh

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

```

#### Step 2: Create the Deployment

1. \*\*Navigate to the `manifests` folder inside the `hello-app` directory:\*\*

```sh

cd hello-app/manifests

```

2. \*\*Deploy the `helloweb-deployment.yaml` manifest to the `gmp-8uj9` namespace:\*\*

```sh

kubectl apply -f helloweb-deployment.yaml --namespace=gmp-8uj9

```

#### Step 3: Inspect the Deployment for Errors

1. \*\*Verify the deployment and navigate to the deployment details page:\*\*

```sh

kubectl get deployments -n gmp-8uj9

```

2. \*\*Describe the deployment to check for errors:\*\*

```sh

kubectl describe deployment helloweb -n gmp-8uj9

```

You should see an error related to an invalid image name.

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

1. \*\*Open the Google Cloud Console and navigate to Logs Explorer.\*\*

2. \*\*Create a query that exposes warnings/errors you saw in the previous section on the cluster:\*\*

```plaintext

resource.type="k8s\_pod"

severity=WARNING

```

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

- Metric type: Counter

- Log Metric Name: `pod-image-errors`

4. \*\*Create an alerting policy based on the logs-based metric:\*\*

- 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

- Use notification channel: Disable

- Alert policy name: Pod Error Alert

### Summary of Commands

```sh

# Download the hello-app directory

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

# Navigate to the manifests folder

cd hello-app/manifests

# Deploy the helloweb-deployment.yaml manifest to the gmp-8uj9 namespace

kubectl apply -f helloweb-deployment.yaml --namespace=gmp-8uj9

# Verify the deployment

kubectl get deployments -n gmp-8uj9

# Describe the deployment to check for errors

kubectl describe deployment helloweb -n gmp-8uj9

```

### Verify the Objective

After following the above steps, you can verify your progress by checking the status of the deployments and configurations in the provided sandbox environment. If you see the "invalid image name error," you have successfully identified the issue.

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

#### Step-by-Step Guide:

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

1. \*\*Open the Google Cloud Console and navigate to Logs Explorer.\*\*

2. \*\*Create a query that exposes warnings/errors:\*\*

```plaintext

resource.type="k8s\_pod"

severity=WARNING

```

3. \*\*Run the query and ensure you see errors such as `InvalidImageName` and `Failed to apply default image tag "<todo>": couldn't parse image reference "<todo>": invalid reference format`.\*\*

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

- Click on the "Create metric" button.

- For Metric type, select \*\*Counter\*\*.

- For Log Metric Name, use `pod-image-errors`.

#### Step 2: Create an Alerting Policy

1. \*\*Navigate to Monitoring -> Alerting -> Create Policy.\*\*

2. \*\*Create a condition:\*\*

- Condition type: Metric threshold

- Select the custom metric `pod-image-errors`.

- 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

3. \*\*Configure notifications:\*\*

- You can disable notification channels for this exercise.

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

- Use `Pod Error Alert`.

5. \*\*Save the alerting policy.\*\*

### **Task 5: Update and Re-Deploy Your App**

#### Step-by-Step Guide:

#### Step 1: Update the Deployment Manifest

1. \*\*Navigate to the `hello-app/manifests` directory:\*\*

```sh

cd hello-app/manifests

```

2. \*\*Edit the `helloweb-deployment.yaml` file:\*\*

```yaml

containers:

- name: hello-app

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

ports:

- containerPort: 8080

```

#### Step 2: Delete the Existing Deployment

1. \*\*Delete the `helloweb` deployment:\*\*

```sh

kubectl delete deployment helloweb -n gmp-8uj9

```

#### Step 3: Re-Deploy the Updated Manifest

1. \*\*Apply the updated manifest:\*\*

```sh

kubectl apply -f helloweb-deployment.yaml -n gmp-8uj9

```

2. \*\*Verify the deployment:\*\*

```sh

kubectl get deployments -n gmp-8uj9

kubectl describe deployment helloweb -n gmp-8uj9

```

### **Task 6: Containerize Your Code and Deploy It onto the Cluster**

#### Step-by-Step Guide:

#### Step 1: Update the Application Code

1. \*\*Navigate to the `hello-app` directory:\*\*

```sh

cd ~/hello-app

```

2. \*\*Edit the `main.go` file:\*\*

```go

// Line 49: Update the version to 2.0.0

fmt.Fprintf(w, "Version: 2.0.0\n")

```

#### Step 2: Build and Push the Docker Image

1. \*\*Configure Docker to use Artifact Registry:\*\*

```sh

gcloud auth configure-docker $REGION-docker.pkg.dev --quiet

```

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

```sh

docker build -t $REGION-docker.pkg.dev/$PROJECT\_ID/demo-repo/hello-app:v2 .

```

3. \*\*Push the Docker image to Artifact Registry:\*\*

```sh

docker push $REGION-docker.pkg.dev/$PROJECT\_ID/demo-repo/hello-app:v2

```

#### Step 3: Update the Deployment to Use the New Image

1. \*\*Update the `helloweb` deployment image:\*\*

```sh

kubectl set image deployment/helloweb -n gmp-8uj9 hello-app=$REGION-docker.pkg.dev/$PROJECT\_ID/demo-repo/hello-app:v2

```

#### Step 4: Expose the Deployment with a LoadBalancer Service

1. \*\*Expose the deployment:\*\*

```sh

kubectl expose deployment helloweb -n gmp-8uj9 --name=helloweb-service-thhn --type=LoadBalancer --target-port 8080 --port 8080

```

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

```sh

kubectl get svc helloweb-service-thhn -n gmp-8uj9

```

3. \*\*Navigate to the external IP address in your browser to verify the service:\*\*

- You should see:

```

Hello, world!

Version: 2.0.0

Hostname: helloweb-<pod-id>

```

### Summary

By following these steps, you have successfully:

- Created a GKE cluster and enabled Managed Prometheus.

- Deployed a sample application and fixed deployment errors.

- Created a logs-based metric and alerting policy.

- Updated and re-deployed your application with the correct image.

- Containerized your application code and updated the deployment with the new version.

- Exposed your deployment with a LoadBalancer service.