Mục Đích:

Script Go này được thiết kế để kiểm tra trạng thái của các Pod trong một namespace Kubernetes, xác định Pod có mức sử dụng Memory cao, và thực hiện hành động như xóa Pod để giải phóng tài nguyên.

Yêu Cầu

1. Công cụ và môi trường:

- Kubernetes cluster.
- Cài đặt công cụ kubectl để quản lý cluster.
- Go version >= 1.23.4.
- Metrics Server đã được triển khai trên cluster (để cung cấp cácsố liệu CPU/Memory)

2. Thư viện Go (Dependencies):

- `k8s.io/client-go`: Cung cấp các công cụ để thiết lập kết nối, xử lý xác thực, và cấu hình các client, nhưng không cung cấp các client cụ thể cho các tài nguyên Kubernetes.
- `k8s.io/api`: Để sử dụng các kiểu đối tượng Kubernetes như Pod, Namespace, ResourceMetrics. Thư viện này chứa các kiểu dữ liệu API của Kubernetes, giúp bạn định nghĩa và thao tác với các đối tượng Kubernetes.
- `metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"`: Thư viện cung cấp các đối tượng và công cụ liên quan đến metadata của các đối tượng Kubernetes, chẳng hạn như tên, nhãn, tên không gian, và các thuộc tính khác cần thiết để quản lý tài nguyên.
- `k8s.io/metrics/pkg/client/clientset/versioned`: Để lấy thông tin chỉ số (metrics) từ các tài nguyên Kubernetes. Thư viện này giúp bạn theo dõi việc sử dụng tài nguyên (CPU, Memory) của các Pods hoặc các đối tượng khác trong cluster.
- `k8s.io/client-go/kubernetes`: Cung cấp các hàm giúp tương tác với các tài nguyên Kubernetes như Pods, Deployments, Services, và nhiều đối tượng khác thông qua các client trong Kubernetes.
- `k8s.io/client-go/rest`: Thư viện hỗ trợ cấu hình và tạo kết nối RESTful đến Kubernetes API server, giúp thiết lập các kết nối với cluster và xử lý thông tin xác thực khi kết nối.
- `k8s.io/client-go/tools/clientcmd`: Hỗ trợ sử dụng và cấu hình các tệp kubeconfig để kết nối với Kubernetes cluster, giúp việc xác thực và quản lý thông tin cấu hình kết nối trở nên dễ dàng hơn.

3. Quyền truy cập

- ServiceAccount có quyền:
  - get, list, delete trên pods.
  - get trên pods.metrics.k8s.io.

4. Label cho pod và hpa:

- Phải dùng label app để lấy các pods, hpa và giá trị của label này phải giống nhau. Vd:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx #label cho pod
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx #label cho pod
    spec:
      containers:
      - name: main
        image: nginx:1.25.1
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
  labels:
    app: nginx #label cho hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 50
```

Luồng hoạt động

1. Lấy danh sách pods từ namespace
2. Duyệt qua các pod và lấy tên pod lưu vào 1 mãng, để sau đó kiểm tra pod mới đã ready chưa khi tăng hpa vì có độ trễ sau khi tăng pod => lấy pod đang có thay vì pod mới được tạo.
3. Kiểm tra và xử lý từng pod
4. Kiểm tra Memory usage, so sánh với memory limit hoặc request tùy cấu hình (env CHECK_BY).
5. Mở rộng thêm pod bằng cách tăng minReplica của HPA.
6. Kiểm tra Pod mới đã sẵn sàng hay chưa
7. Cập nhật lại Min Replica HPA về giá trị ban đầu.
8. Xóa pod cũ.

Hàm

`checkMemoryUsage`

Mô tả

Hàm `checkMemoryUsage` kiểm tra mức sử dụng bộ nhớ của một container cụ thể (`main`) trong một pod Kubernetes so với ngưỡng đã định, dựa trên `memory limit` hoặc `memory request`. Hàm này giúp xác định liệu container có vượt qua tỷ lệ phần trăm bộ nhớ cho phép hay không.

Tham số

Tham số	Kiểu	Mô tả
`clientset`	`*kubernetes.Clientset`	Client Kubernetes được sử dụng để tương tác với API server Kubernetes.
`metricsClient`	`*versioned.Clientset`	Client Metrics của Kubernetes được sử dụng để lấy thông tin về metrics của các pod.
`pod`	`v1.Pod`	Đối tượng pod cần kiểm tra mức sử dụng bộ nhớ.
`checkBy`	`string`	Xác định cách kiểm tra ngưỡng bộ nhớ: "limit" để kiểm tra theo `memory limit` hoặc "request" để kiểm tra theo `memory request`.

Trả về

- **`true`**: Nếu mức sử dụng bộ nhớ của container `main` vượt quá ngưỡng đã chỉ định (theo `memory limit` hoặc `memory request`).
- **`false`**: Nếu không có lỗi hoặc nếu mức sử dụng bộ nhớ của container `main` dưới ngưỡng cho phép.

Chi tiết hoạt động

1. **Lấy metrics của pod**: Hàm gọi API để lấy thông tin metrics của pod từ `metricsClient`. Nếu không lấy được metrics, hàm sẽ ghi log và trả về `false`.
2. **Kiểm tra container `main`**: Hàm tìm kiếm container có tên là "main" trong pod. Nếu không tìm thấy hoặc không có dữ liệu sử dụng bộ nhớ, hàm trả về `false`.
3. **Lấy ngưỡng bộ nhớ**: Dựa vào tham số `checkBy`, hàm lấy ngưỡng bộ nhớ:
   - Nếu `checkBy == "limit"`, hàm lấy `memory limit` của container.
   - Nếu `checkBy == "request"`, hàm lấy `memory request` của container.
     Nếu không có ngưỡng (limit hoặc request), hàm sẽ ghi log và trả về `false`.
4. **So sánh mức sử dụng bộ nhớ**: Mức sử dụng bộ nhớ thực tế của container `main` được so sánh với ngưỡng. Nếu mức sử dụng vượt quá tỷ lệ phần trăm cho phép (được định nghĩa trong biến môi trường `PERCENT_THRESHOLD`), hàm sẽ ghi log và trả về `true` để chỉ ra rằng container có thể cần được xử lý (ví dụ, terminate).
5. **Trả về kết quả**: Nếu mức sử dụng bộ nhớ không vượt quá ngưỡng, hàm sẽ ghi log và trả về `false`.

Ví dụ sử dụng

```
clientset, _ := kubernetes.NewForConfig(config)
metricsClient, _ := metrics.NewForConfig(config)

pod := v1.Pod{ /* Pod data */ }
checkBy := "limit" // Kiểm tra theo memory limit

if checkMemoryUsage(clientset, metricsClient, pod, checkBy) {
    log.Println("Container 'main' sử dụng bộ nhớ vượt mức giới hạn.")
} else {
    log.Println("Container 'main' không cần xử lý thêm.")
}
```

Hàm `scaleHPA`

Mô tả:

Hàm `scaleHPA` được sử dụng để cập nhật số lượng pod tối thiểu (`MinReplicas`) của một Horizontal Pod Autoscaler (HPA) trong Kubernetes. Hàm này nhận vào tên không gian tên (`namespace`) và bộ lọc nhãn (`labelSelector`) để tìm kiếm HPA phù hợp và điều chỉnh số lượng pod tối thiểu của nó.

Tham số:

- `clientset *kubernetes.Clientset`: Một đối tượng `clientset` của Kubernetes, được sử dụng để truy cập các API của Kubernetes.
- `namespace string`: Tên không gian tên trong Kubernetes nơi HPA sẽ được tìm kiếm.
- `labelSelector string`: Bộ lọc nhãn (label selector) dùng để lọc các HPA trong không gian tên.

Trả về:

- `(*v2.HorizontalPodAutoscaler, int32, error)`: Trả về đối tượng HPA đã được cập nhật, giá trị `MinReplicas` cũ và lỗi nếu có.

Quy trình thực hiện:

1. **Lấy danh sách các HPA**: Dùng `LabelSelector` để tìm các HPA trong không gian tên được chỉ định.
2. **Kiểm tra lỗi**: Nếu không tìm thấy HPA nào hoặc gặp lỗi khi truy vấn, hàm sẽ trả về lỗi.
3. **Lấy giá trị `MinReplicas` cũ**: Nếu HPA tồn tại, giá trị `MinReplicas` sẽ được lấy từ cấu hình hiện tại.
4. **Kiểm tra số lượng replica hiện tại**: Nếu số lượng pod hiện tại đã đạt `MaxReplicas`, giá trị `MaxReplicas` sẽ được tăng lên thêm 1.
5. **Cập nhật `MinReplicas`**: Giá trị `MinReplicas` sẽ được cập nhật thành số lượng pod hiện tại cộng thêm 1.
6. **Cập nhật HPA**: Gửi yêu cầu cập nhật HPA với `MinReplicas` mới.
7. **Trả về kết quả**: Trả về đối tượng HPA đã được cập nhật, giá trị `MinReplicas` cũ và lỗi (nếu có).

Code:

```
func scaleHPA(clientset *kubernetes.Clientset, namespace string, labelSelector
string) (*v2.HorizontalPodAutoscaler, int32, error) {
    // Lấy HPA hiện tại
    hpaList, err :=
clientset.AutoscalingV2().HorizontalPodAutoscalers(namespace).List(context.TODO
(), metav1.ListOptions{
        LabelSelector: labelSelector, // Sử dụng label selector để lọc HPA
    })

    if err != nil {
        return nil, 0, fmt.Errorf("không thể lấy HPA: %v", err)
    }
    // Kiểm tra danh sách HPA có phần tử nào không
    if len(hpaList.Items) == 0 {
        return nil, 0, fmt.Errorf("không tìm thấy HPA nào với labelSelector:
%s", labelSelector)
    }
    // Lấy phần tử đầu tiên
    var minReplica int32
    hpa := hpaList.Items[0]
    if hpa.Spec.MinReplicas != nil {
        minReplica = *hpa.Spec.MinReplicas
    }
    if hpa.Status.CurrentReplicas == hpa.Spec.MaxReplicas {
        hpa.Spec.MaxReplicas = hpa.Spec.MaxReplicas + 1
    }
    var newMinReplicas = hpa.Status.CurrentReplicas + 1

    // Cập nhật lại HPA với minReplicas mới
    hpa.Spec.MinReplicas = &newMinReplicas
    updatedHPA, err :=
clientset.AutoscalingV2().HorizontalPodAutoscalers(namespace).Update(context.TO
DO(), &hpa, metav1.UpdateOptions{})
```

```
    if err != nil {
        return nil, minReplica, fmt.Errorf("không thể cập nhật HPA %s: %v",
hpa.Name, err)
    }

    log.Printf("Đã cập nhật HPA %s với minReplicas = %d", hpa.Name,
newMinReplicas)
    return updatedHPA, minReplica, nil
}
```

Hàm `checkForPodReady`

Mô tả:

Hàm `checkForPodReady` dùng để kiểm tra xem có bất kỳ Pod nào trong một namespace với label selector đã được tạo và sẵn sàng nhận yêu cầu chưa. Hàm này gồm hai vòng lặp:

1. **Vòng lặp ngoài**: Kiểm tra xem Pod mới có được tạo ra hay không. Nếu không, hàm sẽ thử lại sau một khoảng thời gian.
2. **Vòng lặp trong**: Kiểm tra xem Pod đã được tạo và có sẵn sàng nhận yêu cầu chưa, với một số lần thử lại nếu Pod chưa sẵn sàng.

Tham số:

- `clientset *kubernetes.Clientset`: Đối tượng `clientset` của Kubernetes, dùng để truy cập các API của Kubernetes.
- `namespace string`: Tên không gian tên (namespace) của các Pods.
- `pollInterval int`: Khoảng thời gian (tính bằng giây) giữa các lần thử.
- `labelSelector string`: Bộ lọc nhãn (label selector) để lọc các Pods cần kiểm tra.

Trả về:

- `bool`: Trả về `true` nếu ít nhất một Pod đã sẵn sàng, ngược lại trả về `false`.

Quy trình thực hiện:

1. **Kiểm tra sự tồn tại của Pod mới**:
    - Vòng lặp ngoài sẽ lặp lại tối đa 3 lần để kiểm tra xem Pod có được tạo mới hay không.
    - Nếu không có Pod mới nào, hàm sẽ đợi 10 giây và thử lại.
2. **Kiểm tra trạng thái "Ready" của Pod**:
    - Sau khi xác định Pod mới đã được tạo, vòng lặp trong sẽ kiểm tra xem Pod đó có sẵn sàng (trạng thái "Ready") hay không.
    - Nếu Pod chưa sẵn sàng, hàm sẽ thử lại tối đa 3 lần với khoảng thời gian chờ 30 giây giữa các lần thử.
3. **Trả về kết quả**:

- Nếu Pod được xác nhận là sẵn sàng sau các lần thử, hàm sẽ trả về `true`.
- Nếu sau 3 lần thử mà Pod vẫn không sẵn sàng, hàm sẽ trả về `false`.

Code:

```
func checkForPodReady(clientset *kubernetes.Clientset, namespace string,
pollInterval int, labelSelector string) bool {
    var result = false
    maxExistRetries := 3
    existRetries := 0
    // Lấy danh sách Pods với label selector
    for existRetries < maxExistRetries {
        pods, err := clientset.CoreV1().Pods(namespace).List(context.TODO(),
metav1.ListOptions{
            LabelSelector: labelSelector,
        })

        if err != nil {
            log.Printf("Lỗi khi lấy danh sách Pods: %v", err)
            return result
        }

        // Sắp xếp danh sách Pods theo thời gian tạo (creation timestamp) giảm
dần
        sort.SliceStable(pods.Items, func(i, j int) bool {
            return
pods.Items[i].CreationTimestamp.After(pods.Items[j].CreationTimestamp.Time)
        })
        if len(pods.Items) > 0 {
            pod := pods.Items[0]
            podExists := false
            for _, name := range podNames {
                if pod.Name == name {
                    podExists = true
                    break
                }
            }
            if !podExists {
                podNames = append(podNames, pod.Name)
                maxRetries := 3
                retries := 0
                // Kiểm tra xem Pod đã sẵn sàng chưa, tối đa 3 lần thử
                for retries < maxRetries {
                    if isPodReady(clientset, pod.Name, pod.Namespace) {
                        log.Printf("Pod %s đã sẵn sàng và có thể nhận
request!", pod.Name)
                        result = true
                        break
                    } else {
                        log.Printf("Pod %s chưa sẵn sàng, thử lại lần %d",
pod.Name, retries+1)
```

```
                        retries++
                        // Nếu chưa sẵn sàng, đợi trước khi thử lại
                        time.Sleep(time.Duration(30) * time.Second)
                    }
                }

                // Nếu sau 3 lần thử mà Pod vẫn chưa sẵn sàng, ghi lại thông
báo lỗi
                if retries == maxRetries {
                    log.Printf("Pod %s vẫn chưa sẵn sàng sau %d lần thử.",
pod.Name, maxRetries)
                }
                break
            } else {
                // Nếu pod đã tồn tại trong danh sách podNames, đợi 10 giây và
thử lại 3 lần mà không kiểm tra lại
                log.Printf("Pod mới chưa được tạo, thử lại sau 10 giây.")
                existRetries++
                time.Sleep(10 * time.Second)
            }
            if existRetries == maxExistRetries {
                log.Printf("Pod mới vẫn chưa được tạo sau %d lần thử.",
maxExistRetries)
                return result
            }
        } else {
            log.Printf("Không có pod nào trong namespace %s với label %s.",
namespace, labelSelector)
            break
        }
    }
    return result
}
```

Hàm `isPodReady`

Mô tả:

Hàm `isPodReady` được sử dụng để kiểm tra xem một Pod cụ thể trong Kubernetes có sẵn sàng để nhận yêu cầu hay không. Hàm này sẽ kiểm tra trạng thái "Ready" của Pod và trả về `true` nếu Pod đã sẵn sàng, ngược lại trả về `false`.

Tham số:

- `clientset *kubernetes.Clientset`: Một đối tượng `clientset` của Kubernetes, được sử dụng để truy cập các API của Kubernetes.
- `podName string`: Tên của Pod cần kiểm tra.
- `namespace string`: Tên không gian tên (namespace) mà Pod đang tồn tại.

Trả về:

- `bool`: Trả về `true` nếu Pod đã sẵn sàng, ngược lại trả về `false`.

Quy trình thực hiện:

1. **Lấy thông tin Pod**: Hàm sẽ gọi API của Kubernetes để lấy thông tin Pod theo tên và không gian tên.
2. **Kiểm tra trạng thái "Ready"**: Hàm sẽ kiểm tra xem Pod có điều kiện "Ready" và trạng thái là `True` không.
3. **Trả về kết quả**: Nếu Pod đã sẵn sàng, hàm sẽ trả về `true`, ngược lại trả về `false`.

Code:

```
func isPodReady(clientset *kubernetes.Clientset, podName, namespace string)
bool {
    // Lấy thông tin pod
    pod, err := clientset.CoreV1().Pods(namespace).Get(context.TODO(), podName,
metav1.GetOptions{})
    if err != nil {
        log.Printf("Không thể lấy thông tin pod %s: %v", podName, err)
        return false
    }

    // Kiểm tra điều kiện "Ready"
    for _, cond := range pod.Status.Conditions {
        if cond.Type == "Ready" && cond.Status == "True" {
            return true
        }
    }
    return false
}
```

Hàm `deletePodAndWait`

Mô tả:

Hàm `deletePodAndWait` dùng để xóa một Pod và sau đó kiểm tra liên tục trạng thái của Pod cho đến khi Pod đó bị xóa hoàn toàn. Nếu không thể xóa Pod hoặc gặp lỗi khi truy vấn trạng thái của Pod, hàm sẽ ghi lại thông báo lỗi.

Tham số:

- `clientset *kubernetes.Clientset`: Đối tượng `clientset` của Kubernetes để truy cập các API của Kubernetes.
- `podNamespace string`: Tên không gian tên (namespace) của Pod.
- `podName string`: Tên của Pod cần xóa.

Quy trình thực hiện:

1. **Xóa Pod**:
    - Hàm sẽ gửi yêu cầu xóa Pod từ namespace xác định.
    - Nếu không thể xóa Pod, hàm sẽ ghi lại lỗi và dừng thực hiện.
2. **Chờ Pod bị xóa**:
    - Sau khi yêu cầu xóa Pod được gửi, hàm sẽ kiểm tra trạng thái của Pod trong vòng lặp.
    - Nếu Pod vẫn còn tồn tại, hàm sẽ chờ một khoảng thời gian xác định (theo biến môi trường `TIME_DELETE_WAITING`) và thử lại.
    - Khi Pod không còn tồn tại (bị xóa hoàn toàn), hàm sẽ dừng vòng lặp và ghi lại thông báo rằng Pod đã bị terminate.

Trả về:

- Hàm không trả về giá trị, nhưng sẽ ghi log thông báo về tình trạng của Pod trong quá trình thực hiện.

Code:

```
func deletePodAndWait(clientset *kubernetes.Clientset, podNamespace, podName
string) {
    // Gửi yêu cầu xóa Pod
    err := clientset.CoreV1().Pods(podNamespace).Delete(context.TODO(),
podName, metav1.DeleteOptions{})
    if err != nil {
        log.Printf("Không thể xóa pod cũ %s: %v", podName, err)
        return
    }
    log.Printf("Đã gửi yêu cầu xóa pod: %s", podName)

    // Vòng lặp để kiểm tra trạng thái của pod
    for {
        _, err := clientset.CoreV1().Pods(podNamespace).Get(context.TODO(),
podName, metav1.GetOptions{})
        if err != nil {
            // Nếu pod không tồn tại nữa, coi như đã bị terminate
            log.Printf("Pod %s đã bị terminate.", podName)
            break
        }
        log.Printf("Pod %s vẫn đang tồn tại. Chờ...", podName)
        // Thời gian chờ giữa các lần kiểm tra
        waitTimeSeconds := helper.GetEnvAsInt("TIME_DELETE_WAITING", 5)
        time.Sleep(time.Duration(waitTimeSeconds) * time.Second)
    }
}
```

Hàm `processPod`

Mô tả:

Hàm `processPod` dùng để xử lý một Pod Kubernetes, bao gồm việc kiểm tra mức sử dụng bộ nhớ của Pod, thay thế Pod bằng một Pod mới nếu cần thiết, và sau đó xóa Pod cũ. Nó còn cập nhật Horizontal Pod Autoscaler (HPA) để điều chỉnh số lượng bản sao tối thiểu trước khi thay thế Pod và sau khi pod mới đã được tạo.

Tham số:

- `clientset *kubernetes.Clientset`: Đối tượng `clientset` của Kubernetes để truy cập các API của Kubernetes.
- `metricsClient *versioned.Clientset`: Đối tượng `metricsClient` của Kubernetes để truy cập dữ liệu metrics.
- `pod *v1.Pod`: Đối tượng Pod mà bạn muốn xử lý.
- `checkBy string`: Tham số xác định cách thức kiểm tra mức sử dụng bộ nhớ (ví dụ: theo tên hoặc theo một chỉ số cụ thể).
- `pollInterval int`: Thời gian gián đoạn (đơn vị giây) giữa các lần kiểm tra Pod.

Quy trình thực hiện:

1. **Xử lý Pod**:
    - Hàm bắt đầu bằng việc ghi log tên Pod đang được xử lý.
    - Lấy namespace của Pod từ biến môi trường `NAMESPACE`.
2. **Kiểm tra mức sử dụng bộ nhớ**:
    - Gọi hàm `checkMemoryUsage` để kiểm tra mức sử dụng bộ nhớ của Pod. Nếu không cần thiết phải thay thế Pod (trong trường hợp mức sử dụng bộ nhớ không vượt quá ngưỡng), hàm sẽ trả về ngay lập tức.
3. **Thay thế Pod**:
    - Nếu mức sử dụng bộ nhớ quá cao, hàm sẽ tiến hành thay thế Pod bằng cách:
        - Gọi hàm `scaleHPA` để thay đổi cấu hình HPA (Horizontal Pod Autoscaler) và điều chỉnh số lượng bản sao tối thiểu.
        - Kiểm tra xem Pod mới đã sẵn sàng chưa bằng cách gọi hàm `checkForPodReady`.
        - Cập nhật lại HPA về giá trị ban đầu với số lượng bản sao tối thiểu.
4. **Xóa Pod cũ**:
    - Gọi hàm `deletePodAndWait` để xóa Pod cũ và chờ cho đến khi Pod bị xóa hoàn toàn.

Trả về:

- Hàm không trả về giá trị, nhưng sẽ ghi log thông báo về trạng thái của Pod trong quá trình thực hiện.

Code:

```
func processPod(clientset *kubernetes.Clientset, metricsClient
*versioned.Clientset, pod *v1.Pod, checkBy string, pollInterval int) {
    // Log thời gian và thông tin pod đang được xử lý
    log.Printf("Xử lý pod: %s", pod.Name)
    namespace := helper.GetEnv(constants.FieldNames.NAMESPACE,
constants.NAMESPACE)

    // Kiểm tra mức sử dụng bộ nhớ
    shouldTerminate := checkMemoryUsage(clientset, metricsClient, *pod,
checkBy)
    if !shouldTerminate {
        return
    }

    // Tạo pod mới thay thế pod cũ
    updatedHPA, minReplica, err := scaleHPA(clientset, namespace,
fmt.Sprintf("app=%s", pod.Labels["app"]))

    // Chờ pod mới sẵn sàng trước khi xóa pod cũ
    checkForPodReady(clientset, namespace, pollInterval, fmt.Sprintf("app=%s",
pod.Labels["app"]))

    // Lấy thông tin HPA mới và cập nhật lại minReplicas
    hpa, err :=
clientset.AutoscalingV2().HorizontalPodAutoscalers(namespace).Get(context.TODO(
), updatedHPA.Name, metav1.GetOptions{})
    hpa.Spec.MinReplicas = &minReplica

    // Cập nhật lại HPA về giá trị ban đầu
    _, err =
clientset.AutoscalingV2().HorizontalPodAutoscalers(namespace).Update(context.TO
DO(), hpa, metav1.UpdateOptions{})
    if err != nil {
        fmt.Errorf("không thể cập nhật lại HPA %s với minReplicas ban đầu: %v",
updatedHPA.Name, err)
        return
    }
    log.Printf("Đã cập nhật lại HPA %s với minReplicas ban đầu = %d",
updatedHPA.Name, minReplica)

    // Xóa pod cũ
    deletePodAndWait(clientset, pod.Namespace, pod.Name)
}
```

Các vấn đề:

- Trước khi thực hiện kiểm tra xem có đang có pod nào bị lỗi không.
- Khi tăng pod mới và xóa pod cũ dùng cơ chế nào phù hợp. Nếu tăng replica của deployment thì sẽ bị down nếu HPA thấy dư resource cần.

- Nếu tăng minRep trong hpa thì sẽ tăng 1 pod mới và sau khi xóa pod cũ sẽ giảm minReplica về lại, nhưng làm sao để biết pod mới đã lên chưa?
- Phải dùng label app để lấy các pods, hpa.

nếu thất bại thì phải cho vào hàng đợi + gửi mail.
Thêm lưu trữ để lưu lại các pod mà lỗi nếu số lượng nhiều thì không thực hiện nữa. Mà alert critical.
Lưu trữ các pod đã được tìm thấy trước đó để tìm kiếm
Node có đủ resource để thêm pod không?

scale lên bằng deployment replica sau 10s tự down, kể cả khi thiết lập down policy là 5 phút.
scale lên và xóa label luôn thì không biết pod đã chạy chưa