

## مقدمه

در این پروژه باید وب سرویسی پیاده سازی می شد که بتواند در بستر کوبرنتیز و در قالب چندین instance عملیاتی شود. هم چنین برای اینکه ریکوئست ها بین instance های مختلف پخش شوند هم یک Load Balancer باید در جلوی این سرویس قرار می گرفت. خروجی endpoint این وب سرویس باید آیدی instance می بود که نشان دهنده ی Load balancing صورت گرفته، باشد.

## پیاده سازی

این پروژه به زبان Go پیاده سازی شده که در قسمت زیر آمده است.

```
func instanceIDHandler(w http.ResponseWriter, r *http.Request) {
    podName := os.Getenv("POD_NAME")

    if podName == "" {
        http.Error(w, "Pod Name not found", http.StatusInternalServerError)
        return
    }

    parts := strings.Split(podName, "-")
    podID := parts[len(parts)-1]

    fmt.Fprintf(w, "Pod id: %s", podID)
}

func main() {
    http.HandleFunc("/instance-id", instanceIDHandler)
    fmt.Println("Server is starting on port 8080...")
    http.ListenAndServe(":8080", nil)
}
```

## آماده کردن ایمیج

پس از بیلد کردن این پروژه، با کمک Dockerfile زیر، containerization صورت گرفته و ایمیج نهایی در docker hub پوش شده است.

```
FROM frovlad/alpine-glibc:glibc-2.33

COPY app app

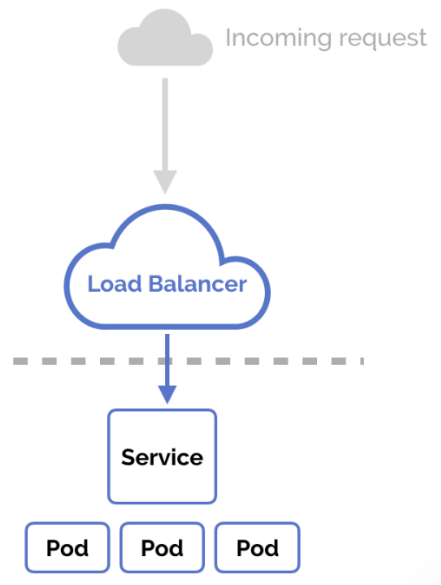
CMD ["/app"]
```

## استفاده از کوبرنتیز

با کمک POD\_NAME که یک متغیر محلی است و از خود podهای کوبرنتیز گرفته می‌شود، آیدی مورد نظر برای خروجی دادن endpoint استخراج شده است.

```
env:
- name: POD_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
```

آماده‌سازی کلاستر کوبرنتیز با کمک minikube انجام شده و موارد زیر در آن بالا آمده اند:



## - Service

که کمک می‌کند بتوانیم با پادها ارتباط داشته باشیم و به آن‌ها ریکوئست بزنیم.

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

## - Deployment

که وظیفه‌ی بالا آورده پادها و تعداد آن‌ها را برعهده دارد.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-app-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: go-app
  template:
    metadata:
      labels:
        app: go-app
    spec:
      containers:
        - name: go-app
          image: alinazariiii/ds-hw2:1.0.1
          ports:
            - containerPort: 8080
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
```

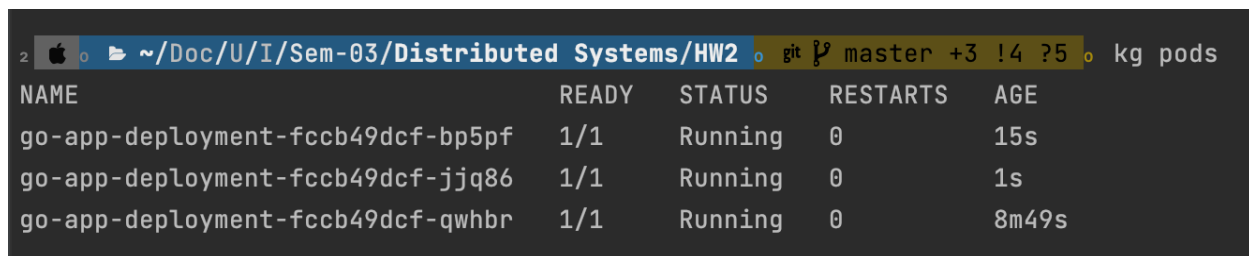
## - Ingress

که مانند nginx وظیفه‌ی load balance را برعهده دارد و ریکوئست‌ها را به سرویس به صورت round-robin ارسال می‌کند.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: go-app-ingress
spec:
  rules:
    - http:
        paths:
          - path: /instance-id
            pathType: Prefix
            backend:
              service:
                name: go-app-service
                port:
                  number: 80
```

## اجرا

همان طور که در ویدیوی پیوست شده هم نمایش داده شده، با بالا آمدن پادها می توان به سرویس مورد نظر ریکوئست زد و در هر بار فراخوانی، آیدی پاد بعدی که به صورت round-robin بین آن ها پخش می شود، برگردانده خواهد شد.

A terminal window showing the command 'kg pods' and its output. The terminal has a dark background with a blue title bar. The title bar contains the file path '~ / Doc / U / I / Sem-03 / Distributed Systems / HW2' and the git status 'master +3 !4 ?5'. The command 'kg pods' is entered at the prompt. The output is a table with 5 columns: NAME, READY, STATUS, RESTARTS, and AGE. There are three rows of data, all showing 'Running' status and '0' restarts.

NAME	READY	STATUS	RESTARTS	AGE
go-app-deployment-fccb49dcf-bp5pf	1/1	Running	0	15s
go-app-deployment-fccb49dcf-jjq86	1/1	Running	0	1s
go-app-deployment-fccb49dcf-qwhbr	1/1	Running	0	8m49s