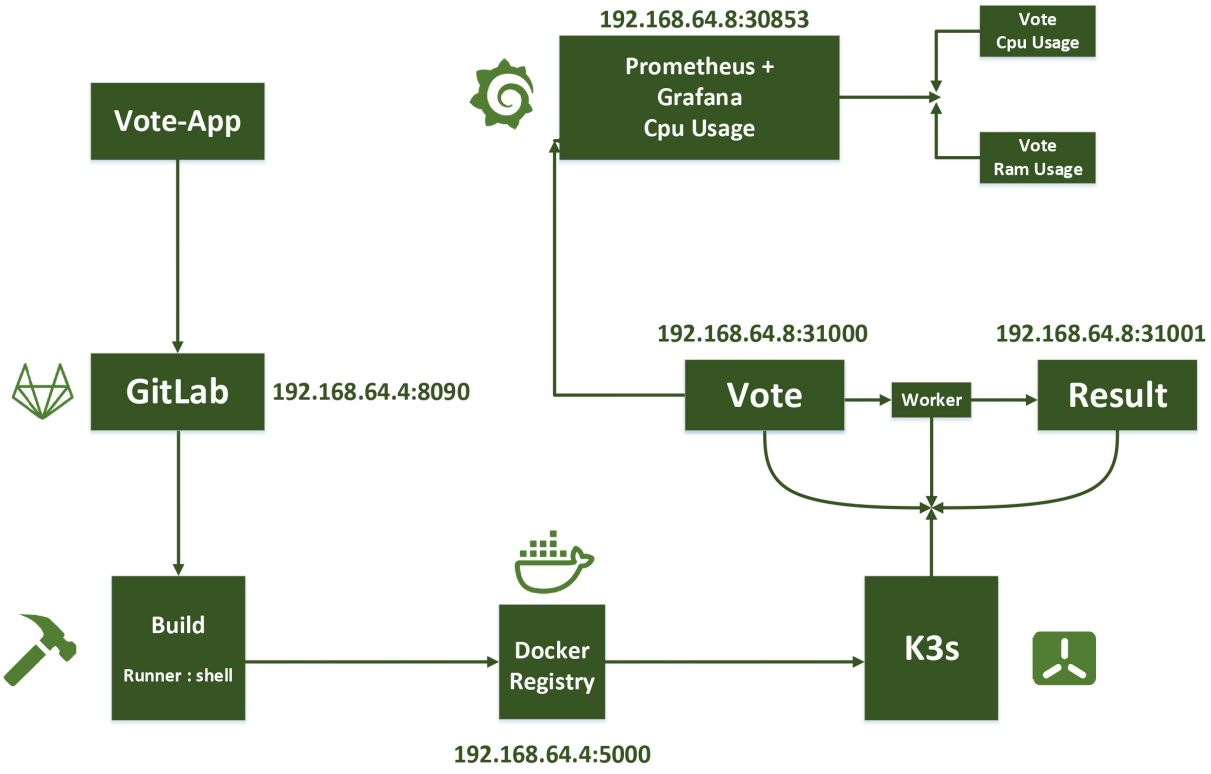
# Devops Project

Kasra Jamshidi




## توضیح ساختار پروژه Vote App

### ساختار کلی پروژه:

این پروژه یک برنامه (Multi-Service App) است و برای استقرار آن از Ansible، GitLab CI/CD، و رجیستری دام استفاده شده است. پروژه شامل دو بخش اصلی است:

- **ساختار زیرساخت (Infrastructure)**
- **ساختار برنامه (Deployment)**

vote-app/

## ├── ansible/

│ ├── group_vars/
│ │ └── infra.yml # متغیرهای عمومی مثل IP، مسیرها، پسورد
│ ├── playbooks/
│ │ ├── infra-setup.yml # اجرای common, docker, gitlab, runner
│ │ ├── deploy-setup.yml # نصب k3s , helm
│ │ └── register-runner.yml # نصب و رجیستر کردن runner

```
│   ├── roles/
│   │   ├── common/
│   │   │   └── tasks/main.yml # نصب ابزارهای پایه مثل curl، vim، ...
│   │   ├── docker/
│   │   │   └── tasks/main.yml # نصب docker و docker-compose
│   │   ├── gitlab/
│   │   │   └── tasks/main.yml # نصب gitlab
│   │   ├── runner/
│   │   │   └── tasks/main.yml # نصب و رجیستر gitlab runner
│   │   └── k3s/
│   │   └── tasks/main.yml # نصب و رجیستر gitlab runner
│   │   └── Helm/
│   │   └── tasks/main.yml # نصب و رجیستر gitlab runner
```

# ├── app-vote/

```
│   ├── vote/ # سرویس vote
│   ├── result/ # سرویس result
│   ├── worker/ # سرویس worker
│   └── version.txt # شماره نسخه برای docker tag
│   └── k3s/ # مخزن سرویس های کوبر
└── .gitlab-ci.yml # فایل CI/CD برای build, push, deploy
```

# K3S

## چرا برای هر سرویس یک Deployment و Service تعریف کردیم؟

در معماری مدرن مبتنی بر **کلاستر Kubernetes**، هر بخش از یک اپلیکیشن (مثلاً backend ،frontend، پایگاه داده، queue و ...) باید به صورت جداگانه قابل مدیریت، مقیاس‌پذیر و مستقل باشد. به همین دلیل، برای **هر سرویس یک Deployment و یک Service مجزا** تعریف کردیم.

## بخش اول: Deployment — تعریف پادها و نحوه اجرای اپلیکیشن

```
apiVersion: apps/v1       # هست apps/v1 از نوع Deployment می‌گه این یه
(ورژن API)
kind: Deployment          # نوع این شیء "Deployment" هست
metadata:
  labels:
    app: vote             # یه label به اسم app:vote تعریف شده
  name: vote              # اسم Deployment رو گذاشتیم "vote"
spec:
  replicas: 1             # اجرا کن vote فقط یه نسخه (پاد) از اپلیکیشن
  selector:
    matchLabels:
```

```
      app: vote           # این Deployment فقط پادهایی رو کنترل می‌کنه که
labelشون app: vote باشه
  template:
    metadata:
      labels:
        app: vote         # این پادهایی که ساخته می‌شن label رو دارن
    spec:
      containers:
      - image: 192.168.64.4:5000/vote:TAG_TO_REPLACE
        name: vote         # اسم کانتینر داخل پاد vote هست
        ports:
        - containerPort: 80       # اپلیکیشن داخل کانتینر روی پورت 80 گوش
می‌ده
          name: vote              # یه اسم به این پورت دادیم (اختیاری)
```

## بخش دوم: Service — ایجاد دسترسی به اون پاد از بیرون

```
apiVersion: v1              # نسخه API مربوط به Service
kind: Service               # نوع Resource که می‌خوایم بسازیم (Service)
metadata:
  name: nginx-service       # نامی که به این Service می‌دیم برای شناسایی
  labels:
    app: nginx              # برچسبی برای دسته‌بندی این Service
spec:
  type: NodePort            # نوع سرویس که دسترسی از بیرون کلاستر رو با
پورت نود میده
  ports:
  - port: 80                # پورتی که داخل کلاستر سرویس گوش میده
    targetPort: 80          # پورتی که روی پاد (کانتینر) بازه
    nodePort: 30080         # پورتی که روی نود کلاستر باز میشه برای دسترسی
بیرونی
    protocol: TCP           # پروتکل ارتباطی (معمولا TCP)
    name: http              # نام دلخواه برای پورت
  selector:
    app: nginx              # این سرویس ترافیک رو به پادهایی که این برچسب
رو دارن میفرسته
```

# .gitlab-ci.yml

```
stages:
  - build
  - push
```

```
  - deploy
variables:
  DOCKER_REGISTRY: "192.168.64.4:5000"
build_images:
  stage: build
  script:
    - IMAGE_TAG=$(cat version.txt)
    - echo "[+] Replacing TAG_TO_REPLACE in vote index.html..."
    - sed -i "s/TAG_TO_REPLACE/$IMAGE_TAG/g" ./vote/templates/index.html
    - echo "[+] Building vote image..."
    - docker build -t $DOCKER_REGISTRY/vote:$IMAGE_TAG ./vote/
    - docker build -t $DOCKER_REGISTRY/result:$IMAGE_TAG ./result/
    - docker build -t $DOCKER_REGISTRY/worker:$IMAGE_TAG ./worker/
  tags:
    - Runner
push_images:
  stage: push
  script:
    - IMAGE_TAG=$(cat version.txt)
    - echo "[+] Pushing vote image..."
    - docker push $DOCKER_REGISTRY/vote:$IMAGE_TAG
    - docker push $DOCKER_REGISTRY/result:$IMAGE_TAG
    - docker push $DOCKER_REGISTRY/worker:$IMAGE_TAG
  tags:
    - Runner
  needs:
    - build_images
deploy_to_k3s:
  stage: deploy
  script:
    - IMAGE_TAG=$(cat version.txt)
    - echo "[+] Replacing image tags in k3s manifests..."
    - |
      VOTE_IMAGE="image: 192.168.64.4:5000/vote:$IMAGE_TAG"
      RESULT_IMAGE="image: 192.168.64.4:5000/result:$IMAGE_TAG"
      WORKER_IMAGE="image: 192.168.64.4:5000/worker:$IMAGE_TAG"
      sed -i "s|image:
192.168.64.4:5000/vote:TAG_TO_REPLACE|$VOTE_IMAGE|g" ./k3s/vote-
deployment.yaml
      sed -i "s|image:
192.168.64.4:5000/result:TAG_TO_REPLACE|$RESULT_IMAGE|g" ./k3s/result-
deployment.yaml
      sed -i "s|image:
```

```
192.168.64.4:5000/worker:TAG_TO_REPLACE|$WORKER_IMAGE|g" ./k3s/worker-
deployment.yaml
      echo "[+] Replacing version tag in index.html..."
      sed -i "s|Version : TAG_TO_REPLACE|Version : $IMAGE_TAG|g"
./vote/templates/index.html

    - echo "[+] Copying manifests to remote server..."
    - sshpass -p "$K3S_SSH_PASSWORD" scp -o StrictHostKeyChecking=no -r
./k3s/ debian@192.168.64.9:/home/debian/k3suser/
    - echo "[+] Deploying to K3s..."
    - sshpass -p "$K3S_SSH_PASSWORD" ssh -o StrictHostKeyChecking=no
debian@192.168.64.9 "kubectl apply -f /home/debian/k3suser/k3s"
  tags:
    - Runner
  needs:
    - push_images
```

# Versioning

### توی CI/CD من چجوری ورژنینگ رو انجام دادم؟

از یک فایل به اسم version.txt استفاده کردم

```
 - IMAGE_TAG=$(cat version.txt)
```

1. موقع **build کردن ایمیج**

```
docker build -t 192.168.64.4:5000/vote:$IMAGE_TAG ./vote/
```

2. موقع **push کردن به رجیستری**:

```
docker push 192.168.64.4:5000/vote:$IMAGE_TAG
```

# Monitoring

## مانیتورینگ K3s با Prometheus + Grafana

ما برای مانیتور کردن وضعیت سرویس‌ها و نودها توی K3s از **Prometheus** و **Grafana** استفاده کردیم. این دو تا ابزار معمولاً با هم نصب می‌شن و مکمل هم هستن:

## ✅ Prometheus

ابزار جمع‌آوری داده‌های مانیتورینگ (مثل میزان CPU، RAM، وضعیت پادها، ...).

خودش به صورت دوره‌ای از سرویس‌ها داده جمع می‌کنه و ذخیره می‌کنه.

## ✅ Grafana

برای **نمایش گرافیکی** اون داده‌هاست.

با Prometheus ارتباط برقرار می‌کنه و داشبوردهای مختلف می‌سازه.

دستور نصب (مثال):

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update

helm install monitoring prometheus-community/kube-prometheus-stack
```

این دستور کل ابزارهای مانیتورینگ رو یکجا نصب می‌کنه:

- Prometheus
- Grafana
- Node Exporter
- Alertmanager

## دوتا Query کاربردی برای Grafana

### مصرف RAM پاد vote:

```
container_memory_usage_bytes{pod="vote-557dbdb5f8-jrbvs", container!="",
container!="POD"}
```

### مصرف CPU پاد vote:

```
rate(container_cpu_usage_seconds_total{pod="vote-557dbdb5f8-jrbvs",
container!="", container!="POD"}[2m])
```