آماده سازی محیط تسک

1) ساخت کلاستر و متصل کردن آن به load balancer

در این مرحله ابتدا می خواهیم کلاستر را درست کنیم برای استفاده می کنیم و ip سروری که برای lb استفاده می کنیم را به آن می دهیم.

```
sudo kubeadm init --control-plane-endpoint "130.185.120.90:6443" \
  --upload-certs \
  --pod-network-cidr=192.168.0.0/16
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

سپس در master1 calico را راه می اندازیم و taint این نور را بر می داریم تا بتوانیم روی آن workload بیاندازیم.

در مرحله بعدی برای اضافه کردن master2 باید join token و crets را از master1 بگیریم.

```
sudo kubeadm token create --print-join-command
sudo kubeadm init phase upload-certs --upload-certs
```

سپس master2 را ابتدا پاک سازی می کنیم kubeadm را ریست می کنیم.
با دادن دستور join به این مستر با ارور مواجه شدیم که پس از بررسی دیدیم که ip ای که برای ساخت کلاستر استفاده کردیم اشتباه بوده و ip مستر1 بوده است پس اول باید آن را تغییر دهیم.

```
* The cluster has a stable controlPlaneEndpoint address.
```

برای این کار نیاز است configmap را در مستر1 تغییر دهیم.

```
kubectl -n kube-system edit configmap kubeadm-config
controlPlaneEndpoint: "130.185.123.175:6443"
```

مجدد در join ارور می خوریم. برای ترابلشوت ابتدا چک می کنیم که مستر2 دسترسی به lb ip دارد.

```
ping -c 3 130.185.123.175
telnet 130.185.123.175 6443
```

با این تست متوجه شدیم که مشکلی در این بخش نداریم.

سپس فایروال ufw مستر1 را غیرفعال کرده و دوباره تلاش می کنیم.
با ارور جدید متوجه می شویم که مستر دو تلاش می کند به endpoint اشتباه متصل شود.
پس aipserver مستر1 را ادیت می کنیم.
دوباره crets را از مستر1 می گیریم و در آخر با دستور زیر موفق به وصل کردن مستر2 به کلاستر می شویم.

```
sudo kubeadm join 130.185.123.175:6443 \
  --token vbyxjk.kmnd69jsxotd1c15 \
  --discovery-token-ca-cert-hash
sha256:a04bbf4db0762004ce2974063874d193f8c37309b5f4074e2c64357f498bb
10d \
  --control-plane --certificate-key
75364df55e411c5e3628c008711c2e9ca48c6f676403a5eee0dc19202e9dd334
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

برای کاهش هزینه ها taint مستر2 را برمیداریم تا دیگر نیازی به اضافه کردن نود worker نداشته باشیم.

```
kubectl taint nodes master2 node-role.kubernetes.io/control-plane:NoSchedule-
```

حال برای چک کردن کلاستر node ها را می گیریم.
در این مرحله متوجه شدیم که مستر2 در حالت not ready می باشد. برای پیدا کردن مشکل این نود را describe می کنیم.

```
Ready False ... Reason:KubeletNotReady message:Network plugin returns error:
cni plugin not initialized
```

مشکل در بخش cni هست.
با دستور زیر نود های calico را می گیریم و متوجه می شویم یکی از آن ها not ready است.
سپس log های آن نود را چک می کنیم.

```
kubectl get pods -n kube-system -o wide | grep calico
kubectl logs -n kube-system calico-node-vlhdd -c calico-node
```

```
Calico node started successfully
bird: Unable to open configuration file /etc/calico/confd/config/bird.cfg: No such
file or directory
```

bird: Unable to open configuration file /etc/calico/confd/config/bird6.cfg: No such file or directory

متوجه می شویم که مشکل از BIRD هست.
برای ریست کردن پاد مشکل دار آن را پاک می کنیم و صبر می کنیم تا دوباره ساخته شود.
با درست نشد مشکل دوباره log های فیلتر شده را چک می کنیم.

```
kubectl logs -n kube-system calico-node-7hxnq -c calico-node | grep -E "bird|Error|Failed|confd"
```

این بار فایل های مورد نیاز را می سازیم و سپس پاد معیوب را دوباره پاک می کنیم.

```
kubectl get node master2 -o json | jq '.metadata.annotations | select(."projectcalico.org/IPv4IPIPTunnelAddr")'
```

با درست نشد مشکل فایل مورد نظر را چک می کنیم.

```
# On master2 or master1
kubectl get node master2 -o json | jq '.metadata.annotations | select(."projectcalico.org/IPv4IPIPTunnelAddr")'

"projectcalico.org/IPv4Address": "10.10.1.54/24",
"projectcalico.org/IPv4IPIPTunnelAddr": "10.244.180.0",
```

متوجه می شویم که مشکل پیش آمده به دلیل به وجود آمدن یک نوع race condition هست و برای درست کردن آن انوشین مشکل دار را پاک می کنیم و سپس پاد معیوب را ریست می کنیم.

```
kubectl annotate node master2 projectcalico.org/IPv4Address- \
projectcalico.org/IPv4IPIPTunnelAddr-
```

با درست نشدن مشکل این بار تلاش می کنیم مشکل را host level حل بکنیم.

```
sudo journalctl -u kubelet | tail -n 50 | grep -E "master2|calico|NotReady|failed"
E1021 11:13:44.808748 4412 kubelet.go:2920] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Network plugin returns error: cni plugin not initialized"
```

Kubelet را ریست می کنیم و calico را هم بصورت کامل ریست می کنیم.

```
CALICO_POD_NAME=$(kubectl get pods -n kube-system -o wide | grep master2 |
grep calico-node | awk '{print $1}')
kubectl delete pod $CALICO_POD_NAME -n kube-system
```

با انجام این کار مستر2 در حالت ready قرار گرفته و پاد های calico نیز 1/1 شدند.
حال در مرحله بعدی می خواهیم مستر3 را اضافه بکنیم.
در مراحل نصف kubeadm بر روی مستر مشکل می خوریم و در نتیجه پکیج های مورد نیاز آن را بر روی سیستم خودمان دانلود کرده و آن را به مستر3 انتقال داده و از این راه آن ها را نصب می کنیم.
در این مرحله شک کردیم که این سرور مشکل دارد ولی ادامه دادیم.
با اضافه کردن این مستر به کلاستر کل کلاستر خراب می شود و همه ی نود های مربوط به cni 0/1 می شوند.
این مستر را از کلاستر drain کرده و به جای calico از flannel برای کلاستر استفاده می کنیم.
ابتدا کل فایل های مربوطه به calico را پاک کرده سپس flannel را بر روی کلاستر دانلود کرده و فایل kube-subnet را ادیت می کنیم تا اطلاعات خودش را از فایل flannel.yaml استفاده کند و سپس فایل flannel.yaml را بر روی کلاستر apply می کنیم.
با چک کردن نود ها هر دو مستر ready هستند اما با چک کردن پاد های فلنل می بینیم که پاد های آن CrashLoopBackOff را به ما می دهند.
فلنل را پاک کرده و دوباره دانلود می کنیم و این بار فایل های آن را تغییر نمی دهیم و فایل flannel.yaml را اپلای می کنیم و این مشکل درست می شود.

در این مرحله ابتدا یک دور همه چیز را چک می کنیم پس به سراغ چک کردن lb می رویم.
برای چک کردن درست کار کردن آن فایل kube-apiserver مستر1 را جا به جا می کنیم و از دستورات kubectl استفاده می کنیم تا ببینیم از سرور دیگر استفاده می کند یا نه که با شکست مواجه می شویم. پس به سراغ فایل های lb می رویم. در این فایل متوجه می شویم که داریم از ip های عمومی استفاده می کنیم و ip مستر2 در این فایل وجود ندارد و فراموش کرده ایم آن را اضافه کنیم.

```
server 130.185.120.90:6443 max_fails=3 fail_timeout=5s;
```

این خط را به این خط تغییر می دهیم.

```
server 10.10.1.53:6443 max_fails=1 fail_timeout=1s;
server 10.10.1.175:6443 max_fails=1 fail_timeout=1s;
```

سپس nginx را ریلود می کنیم.
با ارور مواجه می شویم و متوجه می شویم که کلاستر از lb ip استفاده نمی کند پس کانفینگ آن را تغییر می دهیم.

sudo nano $HOME/.kube/config
server: https://130.185.123.175:6443

پس از چک کردن مجدد می بینیم که مشکل رفع شده است و با جا به جا کردن apiserver مستر1 همچنان می توانیم از kubectl استفاده کنیم.
در این مرحله نیازی به تغییر کانفیگ مستر2 نیست زیرا در دستور join مستر2 آن را با endpoint مستر1 معرفی کرده ایم و مستر2 عضوی از کلاستر ha ما هست.
برای درست کردن مستر 3 می خواهیم در آینده یک سرور جدید بگیریم.
در آخر برای ساختنnamespace مورد نیاز از دستور استفاده کرده و آن را می سازیم.

kubectl create namespace hamravesh-project

2) ساختن منیفست های مورد نیاز

در ابتدا اپ ما یک صفحه سفید ساده است که یک درخواست برای وصل شدن به دیتابیسمان می فرستد و در صورت وصل شدن پیام مربوط به آن را نمایش می دهد.

app deployment file.1
این فایل اصلی مربوط به دیپلویمنت ما است.
برای شروع تعداد replica های آن را برابر با 2 قرار می دهیم.
هم چنین با containerPort: 5000 پورت 5000 را برای دریافت کردن ترافیک داخل کانتینر قرار می دهیم.
سپس credential های مربوط به db را به آن می دهیم.

seting db creds

```
- name: DB_HOST
  value: "db-service.hamravesh-project.svc.cluster.local"
- name: DB_NAME
  value: "mydb"
- name: POSTGRES_USER
  value: "user"
- name: POSTGRES_PASSWORD
  value: "password"
```

در ادامه برای اضافه کردن قابلیت اتوریستارت پاد در صورت خراب بودن آن را با استفاده از liveness و برای چک کردن اینکه کانتینر ما قابلیت دریافت و ارسال ترافیک را دارد ازreadiness استفاده می نماییم و تنظیمات آن را به صورت زیر انجام می دهیم.

initialDelaySeconds: 10
periodSeconds: 5
timeoutSeconds: 3
failureThreshold: 5

این تنظیمات به این معنا است که بعد از 10 ثانیه از شروع پاد شروع به چک کردن در هر 5 ثانیه می کند و اگر بعد از 3 ثانیه تست pass نشود تست شکست می خورد. در صورتی که 5 تست شکست بخوردند پاد ریستارت می شود.
در آخر هم برای هر پاد حداقل و لیمیت های مورد نیاز از ریسورس ها را مشخص می کنیم.

```
resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
```

flask service file.2
این فایل برای مشخص کردن نحوه ارتباط پاد قبلی است.
مدل آن را type: NodePort قرار می دهیم. این به این معنا است که اپ ما از یک پورت مشخص بر روی هر نود در کلاستر قابل دسترس است.
پورتی که کانتینر از طریق آن قابل دسترس است را به صورت targetPort: 5000 مشخص می کنیم.

flask app ingress.3
Ingress به عنوان یک entry point لایه 7 برای کلاستر ما عمل می کند.
ترافیک از اینترنت می آید به سمت سرور lb ما و سپس به سمت کنترلر ingress ما هدایت می شود.
کنترلر با توجه به قوانین تعریف شده داخل این فایل ترافیک را به سمت سرویس های ما می فرستد.
سپس سرویس ما ترافیک را به پورت مورد نظر پاد ما ارسال می کند.

db deployment file.4
این فایل مربوط به بخش db ما هست.
ابتدا تعداد replica های آن را برابر با 1 قرار می دهیم.
برای دیتابیس از image: postgres:15-alpine استفاده می کنیم.
پورت استاندارد postgres را ست می کنیم - containerPort: 5432 .
سپس credential دیتابیس را ست می کنیم.

```
- name: POSTGRES_DB
  value: "mydb"
- name: POSTGRES_USER
  value: "user"
- name: POSTGRES_PASSWORD
  value: "password"
```

سپس برای این که با خراب شدن پاد db دیتا های ما از بین نروند از

```
persistentVolumeClaim:
    claimName: postgres-pvc
```

استفاده می کنیم.
در آخر هم فایل سرویس دیتابیس را در همین فایل تعریف می کنیم.

pvc file.5

این فایل برای این که دیتا های ما با خراب شدن پاد از بین نروند حیاتی است.
به شکل زیر نحوه دسترسی به آن را معلوم می کنیم.

```
accessModes:
  - ReadWriteOnce
```

این به این معنی است که یک نود می تواند آن را مونت کند.
در آخر هم ریسورس مورد نیاز آن را به این شکل storage: 1Gi مشخص می کنیم.

hpa file.6

این فایل برای اضافه کردن قابلیت horizontal pod autoscale به کلاستر ما است.
پادی که باید اسکیل شود را به شکل زیر به آن می دهیم.

```
scaleTargetRef:
   name: flask-app-deployment
```

سپس لیمیت های تعداد پاد را برای آن مشخص می کنیم.

```
minReplicas: 2
maxReplicas: 10
```

سپس معیار محاسبه تعداد پاد مورد نیاز را برای آن مشخص می کنیم. نحوه ی این محاسبات را در بخش مربوطه توضیح خواهیم داد.

```
metrics:
 - type: Resource
   resource:
    name: cpu
    target:
     type: Utilization
     averageUtilization: 70
```

gitlab ci file.7

در آخر فایل مربوط به CICD runner را اضافه می نماییم.
در ابتدای این فایل متغیر های مورد نیاز را به آن می دهیم برای مثال

```
CI_REGISTRY: registry.hamdocker.ir/aliamini83
```

که آدرس کانتینر رجیستری را مشخص می کند.
در ادامه job های مورد نیاز را تعریف می کنیم که رانر باید دو job را انجام دهد.

```
stages:
 - build
 - deploy
```

با build job شروع می کنیم.
در ابتدا عملیات لاگین و build را داریم.

```
  - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
$CI_REGISTRY
  - docker build -t $DOCKER_IMAGE_NAME:$CI_COMMIT_SHA .
```

سپس آن را تگ زده و آن را پوش می کنیم.
در ادامه ی فایل deploy job را داریم.
برای این جاب ابتدا از bitnami برای image استفاده کردیم که گویا فیلتر بود و در نهایت از google cloud sdk استفاده کردیم.

ابتدا باید کانفیگ کلاستر را باشیم.

```
- export KUBECONFIG=$KUBE_CONFIG
```

به این شکل کانفیگ را از وریبل هایی که در ستینگ CICD ذخیره کردیم استفاده می کنیم که این بخش در ادامه کامل تر توضیح داده می شود.
سپس فایل های مربوط به db را اول اپلای می کنیم زیرا پاد های اپ به آن نیاز دارند.
سپس secret مورد نیاز برای پول کردن image ای که در جاب قبلی ساخته ایم را درست می کنیم.
قبل از اپلای کردن منیفست های مربوط به دیپلویمنت اپ نیاز است که با استفاده از sed تغییری در path داده شد در فایل app deployment ایجاد کنیم و path درست را با آن جایگزین کنیم.

```
- sed -i "s|REPLACE_WITH_YOUR_REGISTRY_PATH/simple-web-
app:latest|$DOCKER_IMAGE_NAME:$CI_COMMIT_SHA|g" app-deployment.yaml
```

سپس بقیه فایل هارا به کلاستر اپلای کرده و با استفاده از دستور rollout صبر می کنیم تا پاد های مورد نیاز ما ready شوند.

```
- kubectl rollout status deployment/flask-app-deployment -n
$KUBE_NAMESPACE --timeout=5m
```

3)راه اندازی یک local gitlab runner

برای این کار ابتدا به فایل کانفیگ آن می رویم.

```
sudo nano /etc/gitlab-runner/config.toml
```

در این فایل تگ و اسم رانر خود را اضافه می کنیم.
سپس با استفاده از sudo systemctl restart gitlab-runner آن را ریست کرده و با status وضعیت آن را چک می کنیم.

```
● gitlab-runner.service - GitLab Runner
     Loaded: loaded (/etc/systemd/system/gitlab-runner.service; enabled; preset>
     Active: active (running) since Thu 2025-10-23 13:23:30 +0330; 5h 48min ago
   Main PID: 1556 (gitlab-runner)
      Tasks: 16 (limit: 38235)
     Memory: 74.4M (peak: 81.8M)
        CPU: 20.385s
     CGroup: /system.slice/gitlab-runner.service
             └─1556 /usr/local/bin/gitlab-runner run --config /etc/gitlab-runne>
```

همچین در بخش تنظیماتcicd در سایت هم می توانیم رانر خود را چک کنیم.

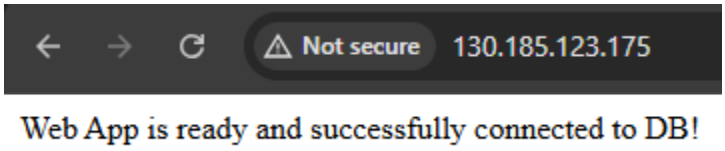
نکته اول این است که تگ رانر باید با تگ جاب های مورد نظر یکی باشد.

نکته بعدی برای وصل شدن رانر به کلاستر ما باید کانفیگ کلاستر خود را به آن به دهیم برای این کار در بخش متغیر ها در سایت این متغیر را ساخته و مقدار آن را با استفاده از دستور زیر بر روی مستر خود می گیریم.

```
sudo cat /etc/kubernetes/admin.conf
```

یکی از باگ هایی که در این بخش با آن مواجه شدیم این بود که اگر رانر را به شکل عادی روی سیستم خود ران کنیم نمی تواند از داکر استفاده کند برای همین فایل کانفیگ آن را لوکال ساخته و از sudo systemctl برای ران کردن رانر استفاده می کنیم.

پس از راه اندازی پایپ لاین می توانیم اپ خود را با استفاده از lb ip مشاهده کنیم.



سناریو ها

بخش بزرگی از سناریو ها را در مرحله های قبلی انجام دادیم.

1) امکان استقرار نسخه جدید برنامه پس از ایجاد تگ جدید در گیت هاب

برای این بخش تنها کافی است در انتهای جاب دلخواه only – tags را اضافه کنیم که هم اکنون در فایل ci قرار ندارد تا بعد از هر کامیت جاب ها ایجاد شوند.

2) مکانیزم ریستارت کردن پاد در صورت ناسالم بودن آن

نحوه ی پیاده سازی این مکانیزم را در بخش فایل app deployment توضیح دادیم. اکنون می خواهیم به عملکرد آن بپردازیم.

```
ubuntu@master1:~$ kubectl get pods -n hamravesh-project -w
NAME                                READY   STATUS    RESTARTS      AGE
flask-app-deployment-844dbbb5f-czsm8   1/1     Running   3 (29h ago)   3d21h
postgres-db-bdc88cd66-cwx7v         1/1     Running   0             4d
```

در ابتدا دو نود بالا را داریم و می خواهیم نود اول را خراب کنیم.برای این کار از دستور زیر استفاده می کنیم.

kubectl exec -n hamravesh-project flask-app-deployment-844dbbb5f-czsm8 -- /bin/sh -c "kill -s TERM 1"

```
flask-app-deployment-844dbbb5f-czsm8   0/1     Completed   3 (29h ago)   3d21h
flask-app-deployment-844dbbb5f-czsm8   0/1     Running     4 (1s ago)    3d21h
flask-app-deployment-844dbbb5f-czsm8   1/1     Running     4 (13s ago)   3d21h
```

همانطور که مشاهده می کنید پاد جدید به درستی ایجاد شده و در حالت running قرار می گیرد.

3) مکانیزم مقیاس پذیر خودکار pod ها

مشکل اولی که برای این سناریو با آن مواجه شدیم با استفاده از دستور

kubectl get hpa -n hamravesh-project -w

قصد داشتیم که عملکرد این سیستم را بسنجیم اما در بخشی که مقدار استفاده از cpu را نشان میداد علامت سوال بود پس متوجه شدیم که سیستم متریک ما مشکل دارد.

kubectl get pods -n kube-system -l k8s-app=metrics-server

با اجرای دستور بالا متوجه شدیم که metric server بر روی کلاستر ما قرار ندارد.

با استفاده از دستور زیر آن را بر روی کلاستر خود قرار دادیم.

kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

```
ubuntu@master1:~$ kubectl get pods -n kube-system -l k8s-app=metrics-server
NAME                              READY   STATUS    RESTARTS   AGE
metrics-server-7fbfbcc44c-5kffz   1/1     Running   0          27h
```

حال برای تست کردن عملکرد این سیستم با دستوری که پیش تر گفته بودیم پاد ها را نگاه کردیم.
با اجرای این دستور اطلاعات زیر به همراه پاد های موجود به ما نمایش داده می شود.

```
Every 5.0s: kubectl get hpa,pods -n hamrav...  master1: Thu Oct 23 16:40:48 2025

NAME                                                                    REFERENCE
            TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/flask-app-deployment   Deployment/flask-app-
deployment   cpu: 0%/50%   1         10        1          27h
```

نحوه ی محاسبات تعداد پاد ها به این صورت هست که تعداد پاد مورد نیاز برابر است با تعداد پاد های فعلی * حاصل تقسیم استفاده از cpu بر روی تارگت اورجی که برای آن ست کرده ایم.

برای نشان دادن تاثیر این تست عدد تارگت را از 50 به 20 رسانده ایم.

با استفاده از دستور زیر فایل hpa را ادیت کرده ایم.

kubectl edit hpa flask-app-deployment -n hamravesh-project
targetCPUUtilizationPercentage: 20

برای ایجاد لود از دستور زیر بر روی ترمینال لوکال خود استفاده کرده ایم. ابتدا hey را نصب کرده سپس:

hey -n 450000 -c 900 -z 90s http://130.185.123.175

```
Every 5.0s: kubectl get hpa,pods -n hamrav...  master1: Thu Oct 23 17:09:00 2025

NAME                                                                    REFERENCE
            TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/flask-app-deployment   Deployment/flask-app-
deployment   cpu: 1%/20%   1         10        8          27h

NAME                                          READY   STATUS    RESTARTS      AGE
pod/flask-app-deployment-844dbbb5f-2tns5      1/1     Running   0             95s
pod/flask-app-deployment-844dbbb5f-7tbjq      1/1     Running   0             110s
pod/flask-app-deployment-844dbbb5f-cpjz4      1/1     Running   0             95s
pod/flask-app-deployment-844dbbb5f-czsm8      1/1     Running   3 (36m ago)   2d17h
pod/flask-app-deployment-844dbbb5f-lm7mn      1/1     Running   1 (41s ago)   110s
pod/flask-app-deployment-844dbbb5f-qjlt4      1/1     Running   1 (66s ago)   110s
pod/flask-app-deployment-844dbbb5f-tpqtz      1/1     Running   1 (21s ago)   95s
pod/flask-app-deployment-844dbbb5f-vz86d      1/1     Running   0             95s
pod/postgres-db-bdc88cd66-cwx7v               1/1     Running   0             2d19h
```

پس از گذشت مدتی کوتاه تعداد replica ها افزایش یافته و استفاده cpu را منطقی می کنند.
با توجه به عکس بالا scale up به درستی انجام شده است.
با صبر کردن برای مدتی scale down انجام شده و replica ها به عدد minpods می رسند که در این بخش این عدد برابر با 1 هست.

```
AME                                          READY   STATUS    RESTARTS      AGE
od/flask-app-deployment-844dbbb5f-czsm8      1/1     Running   3 (42m ago)   2d17h
od/postgres-db-bdc88cd66-cwx7v               1/1     Running   0             2d19h
```