MavaraTech - ESB - Install Infra

- Install Kubernetes
- Postgres Using Swarm
  - Install Docker Swarm
  - Install Postgres
- Install NFS Server
- Install Rabbitmq and Volumes
- Install Integropia
- Install SSO

Install Kubernetes

برای هر نود hostname را تنظیم میکنیم

```
hostnamectl set-hostname {HOSTNAME}
```

برای هر نود نیاز است آدرس زیر ایجاد شود

```
mkdir -p /home/data
```

برای هر نود با استفاده از دستورات زیر پکیج های rpm را نصب میکنیم

فرض می شود همه فایل های نصب در دایرکتوری ~/ قرار دارند

```
tar -xf ./k8s_tools.tar.gz
cd ./k8s-offline
yum install -y --cacheonly --disablerepo=* ./rpm/*.rpm
```

در یکی از نود های کوبرنتیز، کانتینر registry برای کوبرنتیز را ایجاد میکنیم

(بهتر است یک registry برای ایمیج های کوبرنتیز در یکی از نود های کوبرنتیز و یک registry اصلی برای سایر ایمیج ها بر روی یکی از سرورهای دیتابیس که مقدار storage بالایی دارد ایجاد شود)

```
cd ./images
docker load -i registry.docker
docker run -d -p 5000:5000 --restart=always --name registry -v rke-registry:/var/lib/registry
registry:latest

#test:
curl -X GET http://{IP}:{PORT}/v2/_catalog
```

registry دوم مشخصات PORT و IP بجای IP و PORT مخصوص کوبرنتیز و registry اول مشخصات PORT و IP بجای IP اصلی را مشخص میکنیم

(این دستورات روی همه نود ها اجرا میشود)

```
sudo systemctl enable docker
sudo systemctl start docker
```

```
vi /etc/docker/daemon.json
{
"insecure-registries":["{IP}:{PORT}","{IP}:{PORT}"],
"group": "docker"
}
systemctl restart docker
```

دستورات زیر را برای همه دون ها اجرا میکنیم

```
adduser mydocker
passwd mydocker
```

```
sudo groupadd docker
sudo usermod -aG docker mydocker
newgrp docker

export UUUU=mydocker
sudo chmod go-w /home/$UUUU
```

از دون اصلی به ازای همه دون ها دستور زیر را اجرا میکنیم

```
ssh-keygen
cat ~/.ssh/id_rsa.pub | ssh mydocker@{IP} "mkdir -p /home/mydocker/.ssh && cat >>
/home/mydocker/.ssh/authorized_keys && chmod 600 /home/mydocker/.ssh/authorized_keys"
```

دستورات زیر را برای همه دون ها اجرا میکنیم

```
export UUUU=mydocker
sudo chmod 700 /home/$UUUU/.ssh
sudo chmod 644 /home/$UUUU/.ssh/authorized_keys
sudo chown $UUUU:$UUUU /home/$UUUU/.ssh/authorized_keys
sudo chown $UUUU:$UUUU /home/$UUUU/.ssh
sudo usermod -d /home/$UUUU $UUUU
sudo systemctl restart sshd
```

دستورات زیر را برای همه دون ها اجرا میکنیم

```
nano /etc/ssh/sshd_config

PubkeyAuthentication yes
AllowAgentForwarding yes
```

```
AllowTcpForwarding yes

sudo cat /etc/ssh/sshd_config | grep -E
"PubkeyAuthentication|AllowAgentForwarding|AllowTcpForwarding"
sudo systemctl restart sshd
```

می‌کنیم اجرا اه دون همه ی‌ارب زای‌ن دروم ی‌اه تروپ ن‌درک زاب ی‌ارب ار ری‌ز تاروت‌سد

```
sudo firewall-cmd --permanent --add-port=5000/tcp
sudo firewall-cmd --permanent --add-port=80/tcp
sudo firewall-cmd --permanent --add-port=443/tcp
sudo firewall-cmd --permanent --add-port=2376/tcp
sudo firewall-cmd --permanent --add-port=6443/tcp
sudo firewall-cmd --permanent --add-port=8472/udp
sudo firewall-cmd --permanent --add-port=9099/tcp
sudo firewall-cmd --permanent --add-port=10250/tcp
sudo firewall-cmd --permanent --add-port=10254/tcp
sudo firewall-cmd --permanent --add-port=30000-32767/tcp
sudo firewall-cmd --permanent --add-port=30000-32767/udp
sudo firewall-cmd --permanent --add-port=2379/tcp
sudo firewall-cmd --permanent --add-port=2380/tcp
sudo firewall-cmd --permanent --add-port=7946/tcp
sudo firewall-cmd --permanent --add-port=7946/udp
sudo firewall-cmd --permanent --add-port=179/tcp
sudo firewall-cmd --permanent --add-port=4789/udp
sudo firewall-cmd --permanent --add-port=5473/tcp
sudo firewall-cmd --permanent --direct --add-rule ipv4 filter FORWARD 99 -o cali+ -j ACCEPT
sudo firewall-cmd --permanent --direct --add-rule ipv4 filter FORWARD 99 -i cali+ -j ACCEPT
sudo firewall-cmd --direct --add-rule ipv4 filter FORWARD 99 -o cali+ -j ACCEPT
sudo firewall-cmd --direct --add-rule ipv4 filter FORWARD 99 -i cali+ -j ACCEPT
sudo firewall-cmd --set-log-denied=all
sudo firewall-cmd --reload
```

م‌ابقی دستورات تن‌ها برای دون اول است:

دوش شوپ لوص‌خم ی‌رتسی‌جر ی‌ور زی‌تن‌ربوک ی‌اه جی‌م‌ی‌ا ات دوش ی‌م اجرا ری‌ز روت‌سد

(دوش ی‌راذگی‌اج زی‌تن‌ربوک registry تروپ و ی‌پ ی‌آ)

```
unzip ~/k8s_images.zip -d ~/
./rancher/rancher-load-images.sh --image-list ./ALL-IMG.txt --images ~/k8s_images.tar.gz --
```

```
registry {IP}:{PORT} 2>&1 | tee ./load.log
```

دستور زیر برای نصب helm انجام می شود (به جای {VERSION} مقدار ورژن صحیح فایل موجود جایگذاری شود)

```
cd ./helm
tar -zxvf helm-{VERSION}.tar.gz
ll /usr/local/bin | grep helm
sudo cp linux-amd64/helm /usr/local/bin/helm
ll /usr/local/bin | grep helm
sudo chmod +x /usr/local/bin/helm
```

دستور زیر برای نصب kubectl انجام می شود

```
cd ./kubectl
echo "$(<kubectl.sha256) kubectl" | sha256sum --check
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

می‌کنیم آماده برای نصب rke را با استفاده از دستورات زیر فایل

```
cd ./rke
sudo chmod +x ./rke
./rke --version
```

می شویم rke گی فایل کانفی محیط را با دستور زیر وارد

```
./rke config --name cluster.yml
```

برای هر دون بصورت زیر پاسخ می‌دهیم:

```
[+] Cluster Level SSH Private Key Path [~/.ssh/id_rsa]: #پاسخ پیشفرض (اینتر میزنیم)
[+] Number of Hosts [1]: #تعداد دون ها
[+] SSH Address of host (1) [none]: #آی پی دون اول
[+] SSH Port of host (1) [22]: #پاسخ پیشفرض (اینتر میزنیم)
[+] SSH Private Key Path of host (10.10.10.12) [none]: #پاسخ پیشفرض (اینتر میزنیم)
[-] You have entered empty SSH key path, trying fetch from SSH key parameter
[+] SSH Private Key of host (10.10.10.12) [none]: #پاسخ پیشفرض (اینتر میزنیم)
[-] You have entered empty SSH key, defaulting to cluster level SSH key: ~/.ssh/id_rsa
[+] SSH User of host (10.10.10.12) [ubuntu]: mydocker
[+] Is host (10.10.10.12) a Control Plane host (y/n)? [y]: y # درصورتیکه دون دوم در نظر هستر است
y
```

```
[+] Is host (10.10.10.12) a Worker host (y/n)? [n]: y # دروصرت‌ی‌ه دون دروم رظن مسترا ست y
[+] Is host (10.10.10.12) an etcd host (y/n)? [n]: y # دروصرت‌ی‌ه دون دروم رظن وکرا ست y
[+] Override Hostname of host (10.10.10.12) [none]: node1 # ه‌است نی‌م دون
[+] Internal IP of host (10.10.10.12) [none]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Docker socket path on host (10.10.10.12) [/var/run/docker.sock]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
```

رد ن‌ه‌ای‌ت جواب سوال‌ات زی‌ر ا‌ر پاسخ می‌ده‌ی‌م:

```
[+] Network Plugin Type (flannel, calico, weave, canal, aci) [canal]: canal #ن‌وع شب‌ک‌ه
[+] Authentication Strategy [x509]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Authorization Mode (rbac, none) [rbac]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Kubernetes Docker image [rancher/hyperkube:v1.26.8-rancher1]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Cluster domain [cluster.local]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Service Cluster IP Range [10.43.0.0/16]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Enable PodSecurityPolicy [n]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Cluster Network CIDR [10.42.0.0/16]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Cluster DNS Service IP [10.43.0.10]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
[+] Add addon manifest URLs or YAML files [no]: #پاسخ پی‌فشرض (ا‌ن‌ت‌ر می‌زن‌م)
```

فای‌ل cluster.yml ا‌ی‌جاد ش‌ده ا‌ر ا‌ب‌ر‌گ‌ش‌ی‌ا‌ری‌و زاب م‌ی‌ن‌ک‌ی‌م و م‌ی‌ز‌ا‌ر ا‌ر ت‌غ‌ی‌ی‌ر می‌ده‌ی‌م:

```
kubelet:
  extra_binds:
    - "/home/data:/home/data"

--------------------

ssh_agent_auth: true



--------------------

addons: |-
  ---
  apiVersion: v1
  kind: Service
  metadata:
    name: custom-ingress-nginx-controller
    namespace: ingress-nginx
```

```
  spec:
    ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    - name: https
      port: 443
      protocol: TCP
      targetPort: https
    selector:
      app: ingress-nginx
      app.kubernetes.io/instance: ingress-nginx
    sessionAffinity: None
    type: LoadBalancer
```

همچنین آی پی و پورت registry را برای همه ایمیج ها در قسمت system_images تنظیم میکنیم
مثال: بطور (قبل از image ها /{PORT}:{IP} جایگذاری میکنیم)

```
system_images:
  etcd: rancher/mirrored-coreos-etcd:v3.5.6    ----- تبدیل شود به ----->  etcd:
{IP}:{PORT}/rancher/mirrored-coreos-etcd:v3.5.6
```

دستور زیر را اجرا میکنیم (دستور خط 5 برای چک کردن اتصال با ssk-key به همه دو ها انجام میشود)

```
eval "$(ssh-agent -k)" && eval $(ssh-agent) && ssh-add ~/.ssh/id_rsa
ssh-add -l

#test:
ssh -t mydocker@{IP} "docker ps"
```

با استفاده از دستور زیر نصب شروع می شود

```
./rke up
```

با استفاده از دستورات زیر فایل کانفیگ کوبرنتیز نصب شده برای استفاده در kubectl
آدرس مربوطه کپی میکنیم

```
mkdir -p /root/.kube/
cp ./rke/kube_config_cluster.yml /root/.kube/config
```

```
kubectl get nodes
```

با استفاده از دستور زیر از صحت نصب کوبرنتیز اطمینان حاصل میکنیم:

```
kubectl get nodes -o wide
```

Postgres Using Swarm

Install Docker Swarm

تنظیمات فایروال را برای همه دون ها ست میکنیم:

```
firewall-cmd --add-port=2376/tcp --permanent
firewall-cmd --add-port=2377/tcp --permanent
firewall-cmd --add-port=7946/tcp --permanent
firewall-cmd --add-port=7946/udp --permanent
firewall-cmd --add-port=4789/udp --permanent
firewall-cmd --reload
```

برای همه دون ها HostName مناسب ست میکنیم

```
hostnamectl set-hostname {NEW_NAME}
```

یک دون را مستر در نظر میگیریم و در دون مستر دستور زیر را میزنیم:

```
docker swarm init
```

دستور زیر را در دون Manager فعلی اجرا می کنیم تا یک کامند دارای توکن تولید کند و مطابق مراحلی که نتیجه کامند میدهد پیش می رویم.

```
docker swarm join-token manager
```

در این مرحله برچسب های دون‌ها در کلاستر Swarm بر اساس HOSTNAME آن ها، تنظیم می کنیم. برچسب گذاری کمک می کند برای استقرار سرویس ها در دون صحیح، براساس برچسب هر دون تصمیم گیری کنیم؛ در واقع برچسب هایی که در این مرحله برای دون‌های عضو کلاستر تعیین می شود، در زمان بندی و استقرار WORKLOAD ها (برای مثال Postgres) روی دون‌ها مورد استفاده قرار می گیرد.

```
docker node update --label-add region={LABEL} {HOSTNAME}
```

جهت مشاهده لیست دون‌ها به همراه label ها از دستور زیر استفاده می شود.

```
docker node ls -q | xargs docker node inspect -f '{{ .ID }} [{{ .Description.Hostname }}]: {{
.Spec.Labels }} - {{ .Status.Addr }}'
```

Install Postgres

دایرکتوری زیر را در همه نود های دیتابیس ایجاد میکنیم:

```
mkdir -p /home/postgres/data
```

یک فایل docker-compose.yaml ایجاد کرده بصورت زیر: (درصورتی که تعداد سرور ها بیش از 2 عدد است فایل را تغییر دهید)

```
version: "3.6"
services:
  zk1:
    image: {REGISTRY_IP}:{REGISTRY_PORT}/bitnami/zookeeper:3.9.1
    deploy:
      placement:
        constraints:
          - "node.labels.region==db-1"
    ports:
      - target: 2181
        published: 2181
        mode: host
      - target: 2888
        published: 2888
        mode: host
      - target: 3888
        published: 3888
        mode: host
    networks:
      db_net: null
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes
      - ZOO_SERVER_ID=1
      - ZOO_SERVERS=0.0.0.0:2888:3888,{SECOND_SERVER_IP}:2888:3888
  zk2:
    image: {REGISTRY_IP}:{REGISTRY_PORT}/bitnami/zookeeper:3.9.1
    deploy:
```

```
    placement:
      constraints:
        - "node.labels.region==db-2"
  ports:
    - target: 2181
      published: 2181
      mode: host
    - target: 2888
      published: 2888
      mode: host
    - target: 3888
      published: 3888
      mode: host
  networks:
    db_net: null
  environment:
    - ALLOW_ANONYMOUS_LOGIN=yes
    - ZOO_SERVER_ID=2
    - ZOO_SERVERS={FIRST_SERVER_IP}:2888:3888,0.0.0.0:2888:3888

pgnode1:
  image: {REGISTRY_IP}:{REGISTRY_PORT}/ghcr.io/zalando/spilo-15:3.0-p1
  ports:
    - target: 5432
      published: 5432
      mode: host
  deploy:
    replicas: 1
    placement:
      constraints:
        - "node.labels.region==db-1"
  environment:
    ZOOKEEPER_HOSTS: {FISRT_SERVER_IP}:2181,{SECOND_SERVER_IP}:2181
    PGPASSWORD_STANDBY: {PASSWORD}
    PGPASSWORD_ADMIN: {PASSWORD}
    PGPASSWORD_SUPERUSER: {PASSWORD}
    SCOPE: pgCluster
  networks:
    db_net: null
  volumes:
```

```
      - /home/postgres/data:/home/postgres/pgdata

pgnode2:

    image: {REGISTRY_IP}:{REGISTRY_PORT}/ghcr.io/zalando/spilo-15:3.0-p1
    ports:
      - target: 5432
        published: 5432
        mode: host
    deploy:
      replicas: 1
      placement:
        constraints:
          - "node.labels.region==db-2"
    environment:
      ZOOKEEPER_HOSTS: {FISRT_SERVER_IP}:2181,{SECOND_SERVER_IP}:2181
      PGPASSWORD_STANDBY: {PASSWORD}
      PGPASSWORD_ADMIN: {PASSWORD}
      PGPASSWORD_SUPERUSER: {PASSWORD}
      SCOPE: pgCluster
    networks:
      db_net: null
    volumes:
      - /home/postgres/data:/home/postgres/pgdata

networks:
  db_net:
    name: mavara-pgsql-network
    external: true
    driver: overlay
```

مقادیر {REGISTRY_IP} ، {REGISTRY_PORT} ، {FIRST_SERVER_IP} ، {SECOND_SERVER_IP} و {PASSWORD} را بر اساس مقادیر صحیح تنظیم کنید

دستورات زیر را اجرا میکنیم Manager Swarm دون یک

```
unzip postgres-cluster.zip
cd postgres-cluster
chmod +x install-postgres-cluster.sh
./install-postgres-cluster.sh {REGISTRY_IP}:{REGISTRY_PORT}
```

دوجود docker-compose.yaml در صورت موفقیت آمیز بودن دستور بالا، در دایرکتوری که فایل
است دستورات زیر را اجرا میکنیم

```
docker network create -d overlay --attachable mavara-pgsql-network
docker stack deploy --compose-file docker-compose.yml the-pgsql-stack
```

کامند های زیر به بررسی وضعیت کلاستر PostgreSQL می پردازد.

```
docker service ls
docker stack services the-pgsql-stack
```

دستورات زیر برای بررسی لاگ سرویس ها میباشد (شماره instance ها را در آخر نام سرویس
تغییر دهید)

```
docker service logs the-pgsql-stack_zk1
docker service logs the-pgsql-stack_pgnode1
```

جهت پایین آوردن کلاستر، کامند `docker stack rm` از طریق یکی دون Manager ها اجرا می شود

```
docker stack rm the-pgsql-stack
```

Install NFS Server

برای استفاده Storage اشتراکی بین سرورهای Kubernetes، نیاز است یکی از سرورها بعنوان File Server تعیین گردد.

معمولا یکی از سرورهای دیتابیس را برای این منظور در نظر میگیریم و NFS Server را روی آن نصب میکنیم

پکیج های RPM مربوط به NFS Server در پکیج k8s-offline موجود میباشد

```
tar -xf ./k8s_tools.tar.gz
cd ./k8s-offline
yum install -y --cacheonly --disablerepo=* ./rpm/*.rpm
```

برای فعال کردن NFS Server از دستورات زیر استفاده میکنیم:

```
systemctl enable rpcbind
systemctl enable nfs-server
systemctl enable nfs-lock
systemctl enable nfs-idmap
systemctl start rpcbind
systemctl start nfs-server
systemctl start nfs-lock
systemctl start nfs-idmap
```

یک دایرکتوری برای جهت NFS ایجاد کرده و permission های زیر را به آن میدهیم

```
mkdir /home/itg
chown -R nfsnobody:nfsnobody /home/itg
chmod -R 777 /home/itg
```

دایرکتوری مشخص شده را در فایل `etc/exports/` با دسترسی رنج آیپی استفاده کننده مشخص میکنیم

```
/home/itg       xxx.xxx.xxx.0/24(rw,sync,no_subtree_check,no_root_squash)
```

nfs server را ریستارت میکنیم

```
systemctl restart nfs-server
```

در آخر rool های فایروال را هم ست میکنیم

```
firewall-cmd --permanent --zone=public --add-service=nfs
firewall-cmd --permanent --zone=public --add-service=mountd
firewall-cmd --permanent --zone=public --add-service=rpc-bind
firewall-cmd --reload
```

Install Rabbitmq and Volumes

ابتدا میبایست در تمام نود های کوبرنتیز دایرکتوری زیر ایجاد شود:

```
mkdir -p /home/data/logs
```

در دستور زیر مقادیر {NFS_IP} و {REGISTRY_PORT} ، {REGISTRY_IP} را بصورت صحیح جایگزاری و اجرا کنید

```
unzip install-rabbitmq-vol.zip
cd install-rabbitmq-vol
chmod +x install_volumes_rabbitmq.sh
./install_volumes_rabbitmq.sh {REGISTRY_IP}:{REGISTRY_PORT} {NFS_IP}
```

با استفاده از دستور زیر وضعیت اجرای rabbitmq را بررسی میکنیم:

```
kubectl -n infra get all
```

با استفاده از دستور زیر وضعیت Volume ها را بررسی میکنیم

```
kubectl -n itg get pv,pvc
```

Install Integropia

و {REGISTRY_PORT} ، {REGISTRY_IP} در دستور زیر مقادیر {NFS_IP} را بصورت صحیح جایگزاری
اجرا کنید

```
unzip install-itg.zip
cd install-itg
chmod +x install.sh
./install.sh
```

با استفاده از دستور زیر وضعیت اجرای ESB را چک میکنیم

```
kubectl -n itg get all
```

Install SSO

مستند نصب SSO با Keycloak

این مستند مراحل ایجاد و استقرار یک سرویس Single Sign-On (SSO) با استفاده از Keycloak ژرو Docker Image، بیلد و پوش به Docker Registry، و دیپلوی 23 ار توضیح میدهد. مراحل شامل ساخت Docker Image، بیلد و پوش به Docker Registry، و دیپلوی Kubernetes است.

مراحل

1. ساخت Docker Image

برای ایجاد Docker Image، فایل `Dockerfile` زیر استفاده میشود:

```
FROM quay.io/keycloak/keycloak:23.0.3 as builder

# Enable health and metrics support
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true

# Configure a database vendor
ENV KC_DB=postgres

WORKDIR /opt/keycloak
# for demonstration purposes only, please make sure to use proper certificates in production
instead
RUN keytool -genkeypair -storepass password -storetype PKCS12 -keyalg RSA -keysize 2048 -dname
"CN=server" -alias server -ext "SAN:c=DNS:localhost,IP:127.0.0.1" -keystore
conf/server.keystore
```

```
ADD --chown=keycloak:keycloak ./com.mavaratech.login-natcode-jar-with-dependencies.jar
/opt/keycloak/providers/com.mavaratech.login-natcode-jar-with-dependencies.jar

RUN /opt/keycloak/bin/kc.sh build

FROM quay.io/keycloak/keycloak:23.0.3

COPY --from=builder /opt/keycloak/ /opt/keycloak/

# change these values to point to a running postgres instance
ENV KC_DB=postgres
#ENV KC_DB_URL=<DBURL>
#ENV KC_DB_USERNAME=<DBUSERNAME>
#ENV KC_DB_PASSWORD=<DBPASSWORD>
ENV KC_HOSTNAME=localhost
ENV KC_HOSTNAME_STRICT_HTTPS=false
ENV TZ=Asia/Tehran
ENTRYPOINT ["/opt/keycloak/bin/kc.sh", "start", "--debug", "--spi-theme-static-max-age=-1","--
spi-theme-cache-themes=false", "--spi-theme-cache-templates=false"]
```

در این فایل، پلاگین جاوا کامپایل شده به دایرکتوری `/opt/keycloak/providers/` اضافه شده است.

دستور بیلد:

```
docker build -t keycloak-custom:23 .
```

توضیح: این دستور Docker Image را با نام `keycloak-custom` و نسخه `23` از محتوای تعریف‌شده در `Dockerfile` می‌سازد.

---

2. پوش Docker Image به Docker Registry

پوش می‌کنیم `192.168.30.26:5000` روی سرور Docker ریجستری به را Docker Image ساخته‌شده:

دستورات:

```
docker tag keycloak-custom:23 192.168.30.26:5000/keycloak-custom:23
```

توضیح: این دستور یک تگ جدید برای Docker Image ساخته‌شده ایجاد می‌کند تا برای پوش کردن به رجیستری آماده شود.

```
docker push 192.168.30.26:5000/keycloak-custom:23
```

توضیح: این دستور Docker Image را به رجیستری مشخص‌شده (`192.168.30.26:5000`) ارسال می‌کند.

---

3. دیپلوی روی Kubernetes

برای استقرار Keycloak، از فایل `keycloak-deploy.yaml` زیر استفاده می‌شود:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: keycloak-deployment
  namespace: nioc
  labels:
    app: keycloak
spec:
  replicas: 1
  selector:
    matchLabels:
      app: keycloak
  template:
    metadata:
      labels:
        app: keycloak
    spec:
      containers:
        - name: keycloak
          image: image
          volumeMounts:
            - mountPath: /opt/keycloak/themes
              name: files-vol
```

```
          subPath: keycloak/themes
      env:
        - name: KC_DB_URL
          valueFrom:
            configMapKeyRef:
              name: integration-configs
              key: keycloak-db-url
        - name: KC_DB_USERNAME
          valueFrom:
            configMapKeyRef:
              name: integration-configs
              key: keycloak-db-username
        - name: KC_DB_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: integration-configs
              key: keycloak-db-password
        - name: KC_HOSTNAME
          valueFrom:
            configMapKeyRef:
              name: integration-configs
              key: keycloak-db-hostname
        - name: KEYCLOAK_ADMIN
          valueFrom:
            configMapKeyRef:
              name: integration-configs
              key: keycloak-admin-user
        - name: KEYCLOAK_ADMIN_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: integration-configs
              key: keycloak-admin-password
        - name: KC_HOSTNAME_STRICT_HTTPS
          value: "false"
        - name: KC_HOSTNAME_STRICT
          value: "false"
        - name: DEBUG_PORT
          value: "*:8787"
  volumes:
    - name: files-vol
```

```
          persistentVolumeClaim:
            claimName: pvc-files


---
apiVersion: v1
kind: Service
metadata:
  name: keycloak-service
  namespace: nioc
spec:
  selector:
    app: keycloak
  ports:
    - name: app
      protocol: TCP
      port: 8443
      targetPort: 8443
    - name: debug
      protocol: TCP
      port: 8787
      targetPort: 8787
```

دستور دیپلوی:

```
kubectl apply -f keycloak-deploy.yaml
```

توضیح: این دستور فایل `keycloak-deploy.yaml` را روی کلاستر Kubernetes اعمال کرده و دیپلوی Keycloak را آغاز می‌کند.

---

4. تنظیم Ingress برای ارائه SSO

با استفاده از فایل `ingress.yaml` زیر، سرویس SSO روی ساب‌دامین ارائه می‌شود:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
```

```
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/keep-alive-requests: "10000"
    nginx.ingress.kubernetes.io/large-client-header-buffers: 8 512k
    nginx.ingress.kubernetes.io/max-worker-connections: "100000"
    nginx.ingress.kubernetes.io/proxy-body-size: 8m
    nginx.ingress.kubernetes.io/proxy-buffer-size: 256k
    nginx.ingress.kubernetes.io/proxy-buffering: "on"
    nginx.ingress.kubernetes.io/proxy-buffers: 4 512k
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/upstream-keepalive-connections: "2000"
    nginx.ingress.kubernetes.io/use-forwarded-headers: "true"
    nginx.ingress.kubernetes.io/use-proxy-protocol: "true"
    nginx.org/server-snippets: |
      listen 80 443 backlog=4096;
  name: integration-ingress-ssl
  namespace: nioc
spec:
  rules:
  - host: mysso.nioc.ir
    http:
      paths:
      - backend:
          service:
            name: keycloak-service
            port:
              number: 8443
        path: /
        pathType: Prefix
```

دستور اعمال:

```
kubectl apply -f ingress.yaml
```

توضیح: این دستور فایل `ingress.yaml` را روی کلاستر Kubernetes اعمال کرده و تنظیمات Ingress برای دسترسی به سرویس SSO را ایجاد می‌کند.

نتیجه

پس از اجرای مراحل بالا، سرویس SSO روی سابدامین `mysso.nioc.ir` در دسترس خواهد بود. اگر مشکلی در هر مرحله پیش آمد، از لاگ‌های مرتبط برای عیب‌یابی استفاده کنید.

دستورات مفید برای بررسی:

بررسی وضعیت پادها:

```
kubectl get pods -n nioc
```

توضیح: این دستور وضعیت پادهای مستقرشده در namespace `nioc` را نمایش می‌دهد.

بررسی لاگ‌های پاد:

```
kubectl logs [POD_NAME] -n nioc
```

توضیح: این دستور لاگ‌های یک پاد مشخص را برای بررسی خطا یا مشکلات نمایش می‌دهد.