Project ORP

K8s Manual

Contents

Machine Setup

```
# Fedora 33 Install
dnf update

# Disable Firewall
disable firewalld.service, enable sshd.service

# disable password login for ssh
# sudo allow no password for fedora user

# set hostname
hostnamectl set-hostname k8s.kstych.com (optional)

#
disable selinux (/etc/sysconfig/selinux)

#
dnf install git screen

#
reboot
```

K8s Cluster Setup

1. Be Super User – sudo su
2. Download k0s , kubectl binaries and make it executables

```
# K0s
curl --output /usr/local/sbin/k0s -L
https://github.com/k0sproject/k0s/releases/download/v0.9.0/k0s-v0.9.0-amd64

# kubectl
curl --output /usr/local/sbin/kubectl -L
"https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/
linux/amd64/kubectl"

# Permission
chmod +x /usr/local/sbin/{kubectl,k0s}
```

3. Now, create a new screen named **k0s** and run **kos server** .

```
# Creating Screen
screen -R k0s

# if Reinstalling, then remove previous Running K0s files
rm -rf /var/lib/k0s/run; k0s server -c ${HOME}/.k0s/k0s.yaml -enable-worker

# After Successfully run come out form screen using ctrl+a+d
```

4. Make Sure **k0s** is terminated or not ?

   Execute Command - **kubectl get node,pods -A**

5. Optional Step

   If your network driver has “lo” in its name it will be ignored when calico-node is visible so solve this using below command

```
kubectl set env daemonset/calico-node -n kube-system
IP_AUTODETECTION_METHOD=interface=wlo.*
```

8. Now wait for all pods to become running status

9. Install Lens Kubernetes IDE for Smooth Work Flow – install.

   - With Lens we can easily deploys yaml and can do many things like

     1. We can get Event logs of pods,

     2. can access pods terminals,

     3. Port forwarding from Service Pod to Localhost Machine.

     4. Graphs and many Other features.

NFS Storage

- For This Guide We are using NFS File System for Storage of our application data.
- For setting up NFS we have to follow two simple steps :-
  - ✔ First Install & Setup NFS in machine.
  - ✔ Deploy a NFS-Client Pod on k8s for communication .
- To use NFS Storage we need to define in k8s :-
  - Storage Class
  - PersistentVolume and Its Claim for Services.

1. Setting Up NFS Storage in machine

```
$ dnf -y install nfs-utils

-------
$ vi /etc/idmapd.conf
# line 5: uncomment and change to your domain name
Domain = app.k8s.kstych.com

Create Folder To Mount
----
vi /etc/exports
# for example, set [/home/nfsshare] as NFS share
/mnt/k8s/nfsroot 192.168.1.2/24(rw,no_root_squash)

#
exportfs -rav
systemctl start nfs-server.service
systemctl enable --now rpcbind nfs-server
```

2. Deploying NFS-Client Pod on k8s

```
# 02_nfsclient_pod.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-pod-provisioner
spec:
  selector:
    matchLabels:
      app: nfs-pod-provisioner
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nfs-pod-provisioner
    spec:
      containers:
        - name: nfs-pod-provisioner
          image: quay.io/external_storage/nfs-client-provisioner:latest
          volumeMounts:
            - name: nfs-provisioner-v
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME # do not change
              value: nfspod # SAME AS PROVISONER NAME VALUE IN STORAGECLASS
            - name: NFS_SERVER # do not change
              value: 192.168.1.2 # Ip of the NFS SERVER
            - name: NFS_PATH # do not change
              value: /mnt/k8s/nfsroot # path to nfs directory setup

      volumes:
       - name: nfs-provisioner-v # same as volumemouts name
         nfs:
           server: 192.168.1.2
           path: /mnt/k8s/nfsroot
```

Istio & Helm

To install istio, we just need its default binary files and then just execute it. But make sure.. kubeconfig is set in Environment.

```
# download helm file
download ./helm-3.4.1/

# download istio files
download ./istio-1.8.0/

## install istio using istioctl
https://istio.io/latest/docs/setup/install/istioctl/

export KUBECONFIG="${HOME}/.k0s/kubeconfig" (kubeconfig)

istioctl install
### other options
### --set components.egressGateways.name=istio-egressgateway --set
components.egressGateways.enabled=true


### istio ingress will not get externalIP if not running on cloud (aws etc)
or if there is no external loadbalancer eg metallb
## edit Service istio loadbalancer and add under "spec.externalIPs"
## external ip should be in your ifconfig (it will be used to bind to ports
80/443 etc)

  externalIPs:
    - 192.168.1.2



#### manual istio sidecar
## kubectl apply -f <(./istioctl kube-inject -f
../samples/httpbin/httpbin.yaml) -n orp
```

Then, Install Cert manager Pods

```
## https://cert-manager.io/docs/installation/kubernetes/

kubectl apply -f
https://github.com/jetstack/cert-manager/releases/download/v1.1.0/cert-
manager.yaml

## verify => kubectl get pods --namespace cert-manager
## kubectl describe certificate -A
```

Then, Prometheus and kiali

```
### global monitoring prometheus and kiali install
helm3 repo add stable https://charts.helm.sh/stable
helm3 repo add prometheus-community
https://prometheus-community.github.io/helm-charts

helm3 repo update

kubectl create namespace monitoring
helm3 install prometheus-stack prometheus-community/kube-prometheus-stack -n
monitoring
## -f prom-values.yaml

helm3 install --set cr.create=true --set cr.namespace=istio-system --
namespace monitoring --repo https://kiali.org/helm-charts kiali-operator
kiali-operator
## edit kiali configmap and change prometheus/grafana/jaeger urls (restart
kiali deployment)
kubectl get secret -n istio-system $(kubectl get sa kiali-service-account -n
istio-system -o jsonpath={.secrets[0].name}) -o jsonpath={.data.token} |
base64 -d


### istio monitoring prometheus (used by kiali only)
kubectl apply -f ${ISTIO_HOME}/samples/addons/prometheus.yaml
kubectl apply -f ${ISTIO_HOME}/samples/addons/jaeger.yaml


kubectl get secret --namespace monitoring prometheus-stack-grafana -o yaml
## get admin-password , admin-user
## base64 decode (admin/prom-operator)
```

Time to Configure, Virtual Services, Gateway for all new Deployments

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: monitoring-issuer
  namespace: istio-system #retuired istio-system (same as Certificate)
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: monitoring-certs
  namespace: istio-system #retuired istio-system (same as ingress-gateway
service)
spec:
  dnsNames:
  - prometheus.k8s.kanzi.in
  - grafana.k8s.kanzi.in
  secretName: monitoring-certs-tls
  issuerRef:
    kind: Issuer
    name: monitoring-issuer
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: monitoring-gateway
  namespace: monitoring
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - prometheus.k8s.kanzi.in
    - grafana.k8s.kanzi.in
    tls:
      httpsRedirect: true # sends 301 redirect for http requests
  - port:
      number: 443
      name: https-monitoring443
      protocol: HTTPS
    hosts:
    - prometheus.k8s.kanzi.in
    - grafana.k8s.kanzi.in
    tls:
      mode: SIMPLE
      credentialName: monitoring-certs-tls # This should match the Certifcate
secretName
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: prometheus
  namespace: monitoring
spec:
  hosts:
  - prometheus.k8s.kanzi.in
  gateways:
  - monitoring-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 9090
        host: prometheus-operated
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: grafana
  namespace: monitoring
spec:
  hosts:
  - grafana.k8s.kanzi.in
  gateways:
  - monitoring-gateway
  http:
  - match:
    - uri:
        prefix: /kiali
    route:
    - destination:
        port:
          number: 20001
        host: kiali.istio-system.svc.cluster.local
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 80
        host: prometheus-stack-grafana
```

CICD Setup

➢ For Setting CICD on K8s you need follow below steps:-

- ✔ Create a namespace for CICD so that other things not mix with these.
- ✔ Now we will need a storage so we will create a new storage class for cicd then we will create PersistentVolume and PersistentVolumeClaim for each services that require storage.
- ✔ Now We are ready to deploy our cicd applications so at first we will deploy Gitea for using remote git.
- ✔ After Gitea, Deploy Registry so that we can put our images.
- ✔ Now for using https we need ssl certificates so we will deploy yaml that will generate certificates for us
- ✔ After successfully generating certificates we will now apply istio-ingress so that we can communicate with our services from outside the cluster.
- ✔ Now finally Deploy Jenkins and its done for CICD.

1. Creating new Namespace for cicd

```
# 01_namespace.yaml

apiVersion: v1
kind: Namespace
metadata:
  name: cicd
```

2. Creating Storage Class for CICD

```
# 02_storage_nfs.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cicdstorage
provisioner: nfspod
parameters:
  archiveOnDelete: “false”
```

3. Deploying Gitea

```
# 03_gitea.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: cicdvolume
  labels:
    name: cicdvolume # name can be anything
spec:
  storageClassName: cicdstorage # same storage class as pvc
  capacity:
    storage: 40Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    server: 192.168.1.2 # ip addres of nfs server
    path: "/mnt/k8s/nfsroot"  # path to directory, make sure directory is
available

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gitea-data
  namespace: cicd
spec:
  storageClassName: cicdstorage
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: gitea-service
  namespace: cicd
spec:
  selector:
    app: gitea
  ports:
  - name: http-gitea
    port: 3000
    targetPort: 3000

---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gitea-deployment
  namespace: cicd
  labels:
    app: gitea
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gitea
  template:
    metadata:
      labels:
        app: gitea
    spec:
      containers:
      - name: gitea
        image: gitea/gitea:latest
        ports:
        - containerPort: 3000
          name: gitea
        volumeMounts:
        - mountPath: /data
          name: gitea-data-vol
          subPath: gitea
      volumes:
      - name: gitea-data-vol
        persistentVolumeClaim:
          claimName: gitea-data
```

4. Deploying registry

```
# 04_registry.yaml
apiVersion: v1
kind: Secret
metadata:
  name: cicd-registry-secret
  labels:
    app: registry
  namespace: cicd
type: Opaque
data:
  htpasswd:
"Y2ljZDokMnkkMDUkRTN5ZlFCY3ovdVFHcWxnaEE3TEMuT0E4MGlRVWpTQjFRLmhlN09Vd1VEUy9J
M0xDdm92NXk="
  # $ htpasswd -Bbn cicd yb9738z
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: registrystorage
  labels:
    name: registrystorage # jenkins claim for volume
spec:
  storageClassName: cicdstorage # same storage class as pvc
  capacity:
    storage: 40Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    server: 192.168.1.2 # ip addres of nfs server
    path: "/mnt/k8s/nfsroot"  # path to directory, make sure directory is
available
---
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: registry-data
  namespace: cicd
spec:
  storageClassName: cicdstorage
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: registry-service
  namespace: cicd
spec:
  ports:
  - name: registry
    port: 5000
    protocol: TCP
    targetPort: 5000
  selector:
    app: registry

---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: registry-deployment
  namespace: cicd
  labels:
    app: registry
spec:
  replicas: 1
  selector:
    matchLabels:
      app: registry
  template:
    metadata:
      labels:
        app: registry
    spec:
      volumes:
      - name: auth
        secret:
          secretName: cicd-registry-secret
          items:
          - key: htpasswd
            path: htpasswd

      - name: registry-vol
        persistentVolumeClaim:
          claimName: registry-data

      containers:
        - image: registry:2
          name: registry
          env:
            - name: REGISTRY_AUTH
              value: "htpasswd"
            - name: REGISTRY_AUTH_HTPASSWD_REALM
              value: "Registry Realm"
            - name: REGISTRY_AUTH_HTPASSWD_PATH
              value: "/auth/htpasswd"
          ports:
            - containerPort: 5000
          volumeMounts:
          #- name: certs-vol
          #  mountPath: /certs
```

```
          - name: registry-vol
            mountPath: /var/lib/registry
            subPath: registry
          - name: auth
            mountPath: /auth
            readOnly: true
```

5. Generate Certificates

```
# 05_certificates.yaml

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: cicd-letsencrypt
  namespace: cert-manager #retuired cert-manager (same as cert-manager)
spec:
  acme:
    email: k8s@kstych.com
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: cicd-letsencrypt-key
    solvers:
      - http01:
          ingress:
            class:  istio
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: cicd-certs
  namespace: istio-system #retuired istio-system (same as ingress-gateway
service)
spec:
  dnsNames:
  - git.k8s.kstych.com
  - registry.k8s.kstych.com
  - jenkins.k8s.kstych.com
  secretName: cicd-certs-tls
  issuerRef:
    kind: ClusterIssuer
    name: cicd-letsencrypt
```

After Applying Above yaml Wait for issuance of all certificates before proceeding next step.

6. Applying Istio Ingress

```
# 06_istio_ingress.yaml

### wait for certificates order to be completed
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: cicd-gateway
  namespace: cicd
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - git.k8s.kstych.com
    - registry.k8s.kstych.com
    - jenkins.k8s.kstych.com
    tls:
      httpsRedirect: true # sends 301 redirect for http requests
  - port:
      number: 443
      name: https-443
      protocol: HTTPS
    hosts:
    - git.k8s.kstych.com
    - registry.k8s.kstych.com
    - jenkins.k8s.kstych.com
    tls:
      mode: SIMPLE
      credentialName: cicd-certs-tls # This should match the Certifcate
secretName
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: gitea
  namespace: cicd
spec:
  hosts:
  - git.k8s.kstych.com
  gateways:
  - cicd-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 3000
        host: gitea-service
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: jenkins
  namespace: cicd
spec:
  hosts:
  - jenkins.k8s.kstych.com
  gateways:
  - cicd-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 8080
        host: jenkins-service
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: registry
  namespace: cicd
spec:
  hosts:
  - registry.k8s.kstych.com
  gateways:
  - cicd-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 5000
        host: registry-service
```

7. Deploying Jenkins

Before Deploying Jenkins make sure you have jenkins image in your registry.

```
# 07_jenkins.yaml

kind: Secret
apiVersion: v1
metadata:
  name: cicdregistry-basic-auth
  namespace: cicd
data:
  .dockerconfigjson: >-

eyJhdXRocyI6eyJyZWdpc3RyeS5rOHMua3N0eWNoLmNvbSI6eyJ1c2VybmFtZSI6ImNpY2QiLCJwY
XNzd29yZCI6InliOTczOHoiLCJhdXRoIjoiWTJsalpEcDVZamszTXpoNiJ9fX0=
type: kubernetes.io/dockerconfigjson
#kubectl create secret docker-registry cicdregistry-basic-auth --docker-
server=registry.k8s.kstych.com --docker-username=cicd --docker-
password=yb9738z -n cicd
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkinstorage
  labels:
    name: jenkinstorage # jenkins claim for volume
spec:
  storageClassName: cicdstorage # same storage class as pvc
  capacity:
    storage: 40Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    server: 192.168.1.2 # ip addres of nfs server
    path: "/mnt/k8s/nfsroot"  # path to directory, make sure directory is
available

---
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jenkins-data
  namespace: cicd
spec:
  storageClassName: cicdstorage
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
---
apiVersion: v1
kind: Service
metadata:
  name: jenkins-service
  namespace: cicd
spec:
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    app: jenkins
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
  namespace: cicd
spec:
  replicas: 1
  selector:
    matchLabels:
        app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      containers:
      - name: jenkins
        image: registry.k8s.kstych.com/cicd/jenkinscicd # make sure image is
available in registry
        securityContext:
          privileged: true # required to build images from /dev/fuse
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: jenkins-home
          mountPath: /var/lib/jenkins
          subPath: jenkins
        #- name: jenkins-home #remote disk can be slow
          #mountPath: /var/lib/containers
          #subPath: jenkins-containers
      imagePullSecrets:
      - name: cicdregistry-basic-auth
      volumes:
      - name: jenkins-home
        persistentVolumeClaim:
          claimName: jenkins-data
```

APP Setup

- Now its time to deploy our Micro-services for doing that follow below steps:-
  - ✔ At First We will create a new namespace for using our micro-services & generate secrets for our registry so that our micro-services can pull images from that.
  - ✔ Now we will need a storage so We will create a new storage class .
  - ✔ Now we are ready to deploy our micro-services so lets start with keycloack[Authentication System] for deploying keycloak we need a postgress database so at first we will deploy postgress database service for keycloak.
  - ✔ After Postgress deployment deploy keycloack .
  - ✔ Once keycloak is deployed so we need to get ssl certificates so now we generate certificates for our all microservices in single go otherwise you can go one by one.
  - ✔ After successfully generating certificates Apply istio-ingress so that we can our services from outside the cluster using domain.
  - ✔ Now we will deploy services one by one like Log,DB,Http services etc using istio-sidecar.
  - ✔ After deploying all services now we have to define istio destination rule so that our micro-services can talk to each other.

1. Creating Namespace & Secret for App

```
# 01_namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: orp
---
kind: Secret
apiVersion: v1
metadata:
  name: orp-registry-secret
  namespace: orp
data:
  .dockerconfigjson: >-

eyJhdXRocyI6eyJyZWdpc3RyeS5rOHMua3N0eWNoLmNvbSI6eyJ1c2VybmFtZSI6ImNpY2QiLCJwY
XNzd29yZCI6InliOTczOHoiLCJhdXRoIjoiWTJsalpEcDVZamszTXpoNiJ9fX0=
type: kubernetes.io/dockerconfigjson

#kubectl create secret docker-registry orp-registry-secret --docker-
server=registry.k8s.kstych.com --docker-username=cicd --docker-
password=yb9738z -n cicd
```

2. Creating Storage for App

```
# 02_storage_nfs.yaml

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: orpstorage
provisioner: nfspod
parameters:
  archiveOnDelete: “false”
```

3. Deploying Postgres for keycloak

```
# 03_keycloack_postgres.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: orp-keycloak-volumne
  labels:
    name: orp-keycloak-volumne # name can be anything
spec:
  storageClassName: orpstorage # same storage class as pvc
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    server: 192.168.1.2 # ip addres of nfs server
    path: "/mnt/k8s/nfsroot/orp" # path to directory, make sure directory is
available
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: keycloak-postgres-data
  namespace: orp
spec:
  storageClassName: orpstorage
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: keycloak-postgres-service
  namespace: orp
spec:
  selector:
    app: postgres-keycloak
  ports:
  - name: http-gitea
    port: 5432
    targetPort: 5432
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-keycloak
  namespace: orp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-keycloak
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: postgres-keycloak
    spec:
      containers:
      - name: postgres
        env:
        - name: POSTGRES_DB
          value: keycloak
        - name: POSTGRES_PASSWORD
          value: password
        - name: POSTGRES_USER
          value: keycloak
        - name: PGDATA
          value: /var/lib/postgresql/data/pgdata
        image: postgres
        ports:
        - containerPort: 5432
        volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgres-data-vol
          subPath: postgreskeycloak
      restartPolicy: Always
      volumes:
      - name: postgres-data-vol
        persistentVolumeClaim:
          claimName: keycloak-postgres-data
```

4. Deploying Keycloack

```
# 04_keycloack.yaml

apiVersion: v1
kind: Service
metadata:
  name: keycloak
  namespace: orp
  labels:
    app: keycloak
spec:
  ports:
  - name: http
    port: 8080
    targetPort: 8080
  selector:
    app: keycloak
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: keycloak
  namespace: orp
  labels:
    app: keycloak
spec:
  replicas: 1
  selector:
    matchLabels:
      app: keycloak
  template:
    metadata:
      labels:
        app: keycloak
    spec:
      containers:
      - name: keycloak
        image: quay.io/keycloak/keycloak:11.0.3
        env:
        - name: KEYCLOAK_USER
          value: "admin"
        - name: KEYCLOAK_PASSWORD
          value: "yb9738z"
        - name: PROXY_ADDRESS_FORWARDING
          value: "true"
        - name: DB_VENDOR
          value: "POSTGRES"
        - name: DB_ADDR
          value: "keycloak-postgres-service"
        - name: DB_DATABASE
          value: "keycloak"
        - name: DB_USER
          value: "keycloak"
        - name: DB_SCHEMA
          value: "public"
        - name: DB_PASSWORD
          value: "password"

        ports:
        - name: http
          containerPort: 8080
        - name: https
```

```
          containerPort: 8443
        readinessProbe:
          httpGet:
            path: /auth/realms/master
            port: 8080
```

5. Generate Certificates

```
# 05_certificates.yaml

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: orp-letsencrypt
  namespace: cert-manager #retuired cert-manager (same as cert-manager)
spec:
  acme:
    email: k8s@kstych.com
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: orp-letsencrypt-key
    solvers:
      - http01:
          ingress:
            class:  istio
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: orp-certs
  namespace: istio-system #retuired istio-system (same as ingress-gateway
service)
spec:
  dnsNames:
  - auth.k8s.kstych.com
  - app.k8s.kstych.com
  secretName: orp-certs-tls
  issuerRef:
    kind: ClusterIssuer
    name: orp-letsencrypt
```

6. Applying Istio Ingress

```
# 06_keycloak_ingress.yaml

apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: orp-gateway
  namespace: orp
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "auth.k8s.kstych.com"
    - "app.k8s.kstych.com"
    tls:
      httpsRedirect: true # sends 301 redirect for http requests
  - port:
      number: 443
      name: https-443
      protocol: HTTPS
    hosts:
    - app.k8s.kstych.com
    - auth.k8s.kstych.com
    tls:
      mode: SIMPLE
      credentialName: orp-certs-tls # This should match the Certifcate
secretName
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: keycloak
  namespace: orp
spec:
  hosts:
  - "auth.k8s.kstych.com"
  gateways:
  - orp-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 8080
        host: keycloak
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: app-http
  namespace: orp
spec:
  hosts:
  - "app.k8s.kstych.com"
  gateways:
  - orp-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 80
        host: http-service
```

7. Deploying Log Service

```
# 07_log_service.yaml

# kubectl apply -f <(../03_istio_helm/istio-1.8.0/bin/istioctl kube-inject -f
07_log_service.yaml) -n orp

kind: Service
apiVersion: v1
metadata:
  name: log-service
  namespace: orp
  labels:
    app: log-service
spec:
  selector:
    app: log-app
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 8085
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: log-app
  namespace: orp
  version: v1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: log-app
      version: v1
  template:
    metadata:
      labels:
        app: log-app
        version: v1
    spec:
      containers:
      - name: log-container
        image: registry.k8s.kstych.com/app_log:latest
        ports:
        - containerPort: 8085
      imagePullSecrets:
      - name: orp-registry-secret
```

8. Deploying DB with seperate postegres Service

```
# 08_db_service.yaml

# kubectl apply -f <(../03_istio_helm/istio-1.8.0/bin/istioctl kube-inject -f
08_db_service.yaml) -n orp

# Create Storage Class
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: orpstorage
provisioner: nfspod
parameters:
  archiveOnDelete: “false”
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: orp-db-volumne
  labels:
    name: orp-db-volumne # name can be anything
spec:
  storageClassName: orpstorage # same storage class as pvc
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    server: 192.168.1.2 # ip addres of nfs server
    path: "/mnt/k8s/nfsroot/orp" # path to directory, make sure directory is
available
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: orp-db-volumne
  namespace: orp
spec:
  storageClassName: orpstorage
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
```

```
      storage: 10Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-appdb
  namespace: orp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-appdb
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: postgres-appdb
    spec:
      containers:
      - name: postgres
        env:
        - name: POSTGRES_DB
          value: appdb
        - name: POSTGRES_PASSWORD
          value: password
        - name: POSTGRES_USER
          value: user
        - name: PGDATA
          value: /var/lib/postgresql/data/pgdata
        image: postgres
        ports:
        - containerPort: 5432
        volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgres-data-vol
          subPath: postgresappdb
      restartPolicy: Always
      volumes:
      - name: postgres-data-vol
        persistentVolumeClaim:
          claimName: orp-db-volumne
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: postgres-appdb
  namespace: orp
spec:
  selector:
    app: postgres-appdb
  ports:
  - name: postgres-appdb-port
    port: 5432
    targetPort: 5432
---

kind: Service
apiVersion: v1
metadata:
  name: db-service
  namespace: orp
  labels:
    app: db-service
spec:
  selector:
    app: db-app
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 8081
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db-app
  namespace: orp
  version: v1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: db-app
      version: v1
  template:
    metadata:
      labels:
        app: db-app
        version: v1
    spec:
      containers:
      - name: db-container
        image: registry.k8s.kstych.com/appdb:latest
        ports:
        - containerPort: 8081
      imagePullSecrets:
        - name: orp-registry-secret
```

9. Deploying Http Service

```
# 09_apphttp.yaml

# kubectl apply -f <(../03_istio_helm/istio-1.8.0/bin/istioctl kube-inject -f
09_apphttp.yaml) -n orp

kind: Service
apiVersion: v1
metadata:
  name: http-service
  namespace: orp
  labels:
    app: http-service
spec:
  selector:
    app: http-app
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 8082
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: http-app
  namespace: orp
  version: v1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: http-app
      version: v1
  template:
    metadata:
      labels:
        app: http-app
        version: v1
    spec:
      containers:
      - name: http-app
        image: registry.k8s.kstych.com/apphttp:latest
        ports:
        - containerPort: 8082
      imagePullSecrets:
      - name: orp-registry-secret
```

10. Applying Destination Rule for micro-services to connect each other.

```
# 10_destrules.yaml

apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: http-service
  namespace: orp
spec:
  host: http-service
  subsets:
  - name: v1
    labels:
      version: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: db-service
  namespace: orp
spec:
  host: db-service
  subsets:
  - name: v1
    labels:
      version: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: log-service
  namespace: orp
spec:
  host: log-service
  subsets:
  - name: v1
    labels:
      version: v1
```