Inside Kubernetes

You have a cluster: now what?

The Plan for today…

We'll deploy a service/an app step by step

- Look at workloads
  - Pods
  - deployments
- Accessing pods
  - Within the cluster
  - Via port forwarding
  - Services
    - ClusterIP
    - NodePort
    - LoadBalancer
    - IngressController
- Adding storage
  - Dynamically provisioned
  - LocalStorage

Create our container

Dockerfile

```
FROM nginx

USER root

RUN mkdir /nginx_file

COPY content /usr/share/nginx/html/k8s-seminar
COPY default.conf /etc/nginx/conf.d/default.conf
```

Create our container

Index.html (inside directory “content”)

```
<!DOCTYPE html>
<html>
<head>
<title>SWESRC</title>
<style type='text/css'>
p {
width: 1200px;
}
</style>
</head>
<body>
<h1>Welcome!</h1>
<p>This is a static page.</p>
</body>
</html>
```

Create our container

default.conf (nginx setup)

```
server {
    listen       80;
    server_name  localhost;

    #access_log  /var/log/nginx/host.access.log  main;

    # putting the root at /k8s-seminar for making the ingress in k8s easier
    # if we have an ingress forwarding from <URL>/k8s-seminar but nginx list
    # we will have to include some middleware to remove the prefix.
    location /k8s-seminar {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }


    location /k8s-seminar/files {
        alias     /nginx_files/;
        autoindex on;
        autoindex_exact_size off;
        autoindex_localtime on;
        try_files $uri $uri/ =404;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    location ~ "/k8s-seminar/upload/([0-9a-zA-Z-.]*)$" {
        dav_methods  PUT DELETE MKCOL COPY MOVE;
        client_body_temp_path /tmp/incoming;
        alias     /nginx_files/$1;
        create_full_put_path   on;
        dav_access             group:rw  all:r;

        client_body_in_file_only on;
        client_body_buffer_size 128k;
        client_max_body_size 100M;
      }
}
```

Test our container locally

```
docker run --name nginx \
           -v /some/path:/nginx_files/:rw \
           -p 8081:80 -d
           nginx-upload:latest
```

In our browser, go to
localhost:8081/k8s-seminar/
localhost:8081/k8s-seminar/files

Upload something via
curl -T /path/to/file.txt localhost:8081/k8s-seminar/upload/file.txt

Test our container locally

```
docker run --name nginx \
           -v /some/path:/nginx_files/:rw \
           -p 8081:80 -d
           nginx-upload:latest
```

In our browser, go to
localhost:8081/k8s-seminar/
localhost:8081/k8s-seminar/files

Upload something via
curl -T /path/to/file.txt localhost:8081/k8s-seminar/upload/file.txt

Test our container locally

```
docker run --name nginx \
           -v /some/path:/nginx_files/:rw \
           -p 8081:80 -d
           nginx-upload:latest
```

In our browser, go to
localhost:8081/k8s-seminar/
localhost:8081/k8s-seminar/files

Upload something via
curl -T /path/to/file.txt localhost:8081/k8s-seminar/upload/file.txt

Let’s get started in our cluster – create a pod

nginx-pod.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: k8s-seminar
---
apiVersion: v1
kind: Pod
metadata:
  name: franz-nginx
  namespace: k8s-seminar
spec:
  containers:
  - name: nginx
    image: services.swesrc.chalmers.se/test/franz-nginx:v0.0.3
    ports:
    - containerPort: 80
```

kubectl apply -f nginx-pod.yaml

Let’s get started in our cluster – create a Pod

nginx-pod.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: k8s-seminar
---
apiVersion: v1
kind: Pod
metadata:
  name: franz-nginx
  namespace: k8s-seminar
spec:
  containers:
  - name: nginx
    image: services.swesrc.chalmers.se/test/franz-nginx:v0.0.3
    ports:
    - containerPort: 80
```

kubectl apply -f nginx-pod.yaml

Access the pod via port-forwarding

kubectl port-forward pods/franz-nginx -n k8s-seminar 85:80

curl localhost:85/k8s-seminar/

Make things more resilient – create a Deployment

nginx-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: franz-nginx-deployment
  namespace: k8s-seminar
  labels:
    app: franz-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: franz-nginx
  template:
    metadata:
      labels:
        app: franz-nginx
    spec:
      containers:
      - name: nginx
        image: services.swesrc.chalmers.se/test/franz-nginx:v0.0.3
        ports:
        - containerPort: 80
```

A deployment creates a ReplicaSet and the Deployment Controller ensures that reality reflects your desired state; i.e. 3 replicas in this case

Ensure there is one persistent IP to contact nginx – create a Service

- Pods will certainly fail and will get recreated – but with new IPs
- A service has a persistent virtual IP and routes to each replica regardless of its IP

Ensure there is one persistent IP to contact nginx – create a Service

- Pods will certainly fail and will get recreated – but with new IPs
- A service has a persistent virtual IP and routes to each replica regardless of its IP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: franz-nginx-service
  namespace: k8s-seminar
spec:
  type: ClusterIP
  selector:
    app: franz-nginx
  ports:
    - protocol: TCP
      port: 81
      targetPort: 80
```

Just to make the point that we can chose the port the Service “listens” on

nginx listens on port 80 in the pod

```yaml
apiVersion: v1
kind: Service
metadata:
  name: franz-nginx-service
  namespace: k8s-seminar
spec:
  type: NodePort
  selector:
    app: franz-nginx
  ports:
    - protocol: TCP
      port: 81
      targetPort: 80
```

Ensure there is one persistent IP to contact nginx – create a Service

- Pods will certainly fail and will get recreated – but with new IPs
- A service has a persistent virtual IP and routes to each replica regardless of its IP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: franz-nginx-service
  namespace: k8s-seminar
spec:
  type: ClusterIP
  selector:
    app: franz-nginx
  ports:
    - protocol: TCP
      port: 81
      targetPort: 80
```

NodePort exposes the service on a high node port on the hypervisor itself

```yaml
apiVersion: v1
kind: Service
metadata:
  name: franz-nginx-service
  namespace: k8s-seminar
spec:
  type: NodePort
  selector:
    app: franz-nginx
  ports:
    - protocol: TCP
      port: 81
      targetPort: 80
```

Ensure there is one persistent IP to contact nginx – create a Service of Type: LoadBalancer

- So far our service was not available to the outside world
- OpenStack allows us to create a Service of type LoadBalancer which has a floating IP and thus gains us that very access

Ensure there is one persistent IP to contact nginx – create a Service of Type: LoadBalancer

- So far our service was not available to the outside world
- OpenStack allows us to create a Service of type LoadBalancer which has a floating IP and thus gains us that very access

```
apiVersion: v1
kind: Service
metadata:
  name: franz-nginx-service
  namespace: k8s-seminar
spec:
  type: LoadBalancer
  loadBalancerIP: 129.16.122.45
  selector:
    app: franz-nginx
  ports:
    - protocol: TCP
      port: 81
      targetPort: 80
```


We fix the IP. Without it the cloud controller will generate a new one each time we bring up this service

Enable the floating IP to act as entrypoint to several services – create an Ingress

- In the current setup, we can only access nginx from that floating IP
- We may want to connect to different services on the same IP
- So we go back to our Service of type ClusterIP and implement an Ingress

Enable the floating IP to act as entrypoint to several services – create an Ingress

- In the current setup, we can only access nginx from that floating IP
- We may want to connect to different services on the same IP
- So we go back to our Service of type ClusterIP and implement an Ingress

```yaml
apiVersion: v1
kind: Service
metadata:
  name: franz-nginx-service
  namespace: k8s-seminar
spec:
  type: ClusterIP
  selector:
    app: franz-nginx
  ports:
    - protocol: TCP
      port: 81
      targetPort: 80
```

Service name needs to match

Ports need to match

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: franz-nginx-ingress
  namespace: k8s-seminar
spec:
  ingressClassName: traefik
  rules:
  - host: services.swesrc.chalmers.se
    http:
      paths:
      - path: /k8s-seminar
        pathType: Prefix
        backend:
          service:
            name: franz-nginx-service
            port:
              number: 81
```

Enable the floating IP to act as entrypoint to several services – create an Ingress

- In the current setup, we can only access nginx from that floating IP
- We may want to connect to different services on the same IP
- So we go back to our Service of type ClusterIP and implement an Ingress

```yaml
apiVersion: v1
kind: Service
metadata:
  name: franz-nginx-service
  namespace: k8s-seminar
spec:
  type: ClusterIP
  selector:
    app: franz-nginx
  ports:
    - protocol: TCP
      port: 81
      targetPort: 80
```

We use Traefik as ingress controller → comes with its own service of type LoadBalancer which listens at services.swesrc.chalmers.se

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: franz-nginx-ingress
  namespace: k8s-seminar
spec:
  ingressClassName: traefik
  rules:
  - host: services.swesrc.chalmers.se
    http:
      paths:
      - path: /k8s-seminar
        pathType: Prefix
        backend:
          service:
            name: franz-nginx-service
            port:
              number: 81
```

Add some storage to our pod – create a persistent volume (claim)

- If we now upload data to our server, those will end up in /nginx_files – but due to loadbalancing by the service, this will be different between the 3 different replicas
- Need to add a common (aka shared) storage to our pods

Add some storage to our pod – create a persistent volume (claim)

- If we now upload data to our server, those will end up in /nginx_files – but due to loadbalancing by the service, this will be different between the 3 different replicas
- Need to add a common (aka shared) storage to our pods

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
  namespace: k8s-seminar
  labels:
    app: franz-nginx
  annotations:
    helm.sh/resource-policy: keep
spec:
  accessModes:
  - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: cinder-storage
```

```
apiVersion: v1
kind: Pod
metadata:
  name: franz-nginx
  namespace: k8s-seminar
spec:
  containers:
  - name: nginx
    image: services.swesrc.chalmers.se/test/franz-nginx:v0.0.3
    ports:
    - containerPort: 80
    volumeMounts:
    - mountPath: "/nginx_files"
      name: nginx-volume
  volumes:
    - name: nginx-volume
      persistentVolumeClaim:
        claimName: nginx-pvc
```

Add some storage to our pod – create a persistent volume (claim)

- If we now upload data to our server, those will end up in /nginx_files – but due to loadbalancing by the service, this will be different between the 3 different replicas
- Need to add a common (aka shared) storage to our pods

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
  namespace: k8s-seminar
  labels:
    app: franz-nginx
  annotations:
    helm.sh/resource-policy: keep
spec:
  accessModes:
  - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: cinder-storage
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cinder-storage
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: cinder.csi.openstack.org
allowVolumeExpansion: true
```

This is so-called dynamically provisioned storage as the underlying persistent volume is created by the driver/cloud controller

CSI cinder drivers

Add some storage to our pod – create a persistent volume (claim)

- If we now upload data to our server, those will end up in /nginx_files – but due to loadbalancing by the service, this will be different between the 3 different replicas
- Need to add a common (aka shared) storage to our pods

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
  namespace: k8s-seminar
  labels:
    app: franz-nginx
  annotations:
    helm.sh/resource-polic
spec:
  accessModes:
  - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: cinder-storage
```

```
apiVersion: v1
kind: Pod
metadata:
  name: franz-nginx
```

```
:.chalmers.se/test/franz-nginx:v0.0.3
```

```
Files"
```

```
im:
pvc
```

However, we cannot use this PVC in more than one of our pods – the trouble is that the underlying RBD cannot be mounted more than once.
→ CephFS to the rescue! (soon)

Add some storage to our pod – create a persistent volume (claim)

- Until we have cephFS up and running, we use an NFS-exported Manila share that is mounted locally on the nodes (VMs) to provide share storage

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    helm.sh/resource-policy: keep
  name: nginx-pv-localstorage
  labels:
    storage: nginx-localstorage
spec:
  capacity:
    storage: 4Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /manila/k8s-seminar
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - swesrc-k8s-prod-large-ram-rocky-v1-34-6-6ngxq-bsz2b
          - swesrc-k8s-prod-large-ram-rocky-v1-34-6-6ngxq-ddkrt
          - swesrc-k8s-prod-large-ram-rocky-v1-34-6-6ngxq-rx77f
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    helm.sh/resource-policy: keep
  name: nginx-pvc-localstorage
  namespace: k8s-seminar
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Filesystem
  storageClassName: local-storage
  resources:
    requests:
      storage: 4Gi
  selector:
    matchLabels:
      storage: nginx-localstorage
```

Add the shared PVC to our deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: franz-nginx-deployment
  namespace: k8s-seminar
  labels:
    app: franz-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: franz-nginx
  template:
    metadata:
      labels:
        app: franz-nginx
    spec:
      containers:
      - name: nginx
        image: services.swesrc.chalmers.se/test/franz-nginx:v0.0.3
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: "/nginx_files"
          name: nginx-volume
      volumes:
        - name: nginx-volume
          persistentVolumeClaim:
            claimName: nginx-pvc-localstorage
```

PVC name needs to match

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    helm.sh/resource-policy: keep
  name: nginx-pvc-localstorage
  namespace: k8s-seminar
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Filesystem
  storageClassName: local-storage
  resources:
    requests:
      storage: 4Gi
  selector:
    matchLabels:
      storage: nginx-localstorage
```

Attempt of a schematic overview :-)