Pods

» Creating Pods with Manifests

Kubernetes Manifests

Create a new pod manifest that specifies two containers.

```
cat > pod-multi-container.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: my-two-container-pod
  namespace: myproject
  labels:
    environment: dev
spec:
  containers:
    - name: server
      image: nginx:1.13-alpine
      ports:
        - containerPort: 80
          protocol: TCP
    - name: side-car
      image: alpine:latest
      command: ["/usr/bin/tail", "-f", "/dev/null"]
  restartPolicy: Never
EOF
```

Create the pod by specifying the manifest.

```
oc create -f pod-multi-container.yaml
```

View the detail for the pod and look at the events.

```
oc describe pod my-two-container-pod
```

Let's first execute a shell session inside the server container by using the `-c` flag.

```
oc exec -it my-two-container-pod -c server -- /bin/sh
```

Run some commands inside the server container.

```
ip address
netstat -ntlp
hostname
ps
exit
```

Let's now execute a shell session inside the side-car container.

```
oc exec -it my-two-container-pod -c side-car -- /bin/sh
```

Run the same commands in side-car container. Each container within a pod runs it's own cgroup, but shares IPC, Network, and UTC (hostname) namespaces.

```
ip address
netstat -ntlp
hostname
exit
```

Verify the currently available api versions

```
oc api-versions
```

Use the `--v` flag to set a verbosity level. This will allow you to see the request/responses against the Kubernetes API.

```
oc get pods --v=8
```

Basic Operations with the Kubernetes API

Use the `kubectl proxy` command to proxy local requests on port 8001 to the Kubernetes API.

```
oc proxy --port=8001
```

Send a `GET` request to the Kubernetes API using `curl`.

```
curl -X GET http://localhost:8001
```

We can explore the OpenAPI definition file to see complete API details.

```
curl localhost:8001/openapi/v2
```

Send a `GET` request to list all pods in the environment.

```
curl -X GET http://localhost:8001/api/v1/pods
```

Use `jq` to parse the json response.

```
curl -X GET http://localhost:8001/api/v1/pods | jq .items[].metadata.name
```

We can scope the response by only viewing all pods in a particular namespace.

```
curl -X GET http://localhost:8001/api/v1/namespaces/myproject/pods
```

Get more details on a particular pod within the `myproject` namespace.

```
curl -X GET http://localhost:8001/api/v1/namespaces/myproject/pods/my-two-container-pod
```

Export the manifest associated with `my-two-container-pod` in json format.

```
oc get pods my-two-container-pod --export -o json > podmanifest.json
```

Within the manifest, replace the `1.13` version of alpine with `1.14` .

```
sed -i .bak 's|nginx:1.13-alpine|nginx:1.14-alpine|g' podmanifest.json
```

Update/Replace the current pod manifest with the newly updated manifest.

```
curl -X PUT http://localhost:8001/api/v1/namespaces/myproject/pods/my-two-container-pod -H "Content-type: application/json" -d @podman
ifest.json
```

Patch the current pod with a newer container image (`1.15`).

```
curl -X PATCH http://localhost:8001/api/v1/namespaces/myproject/pods/my-two-container-pod -H "Content-type: application/strategic-merg
e-patch+json" -d '{"spec":{"containers":[{"name": "server","image":"nginx:1.15-alpine"}]}}'
```

Delete the current pod by sending the DELETE request method.

```
curl -X DELETE http://localhost:8001/api/v1/namespaces/myproject/pods/my-two-container-pod
```

Verify the pod no longer exists.

```
curl -X GET http://localhost:8001/api/v1/namespaces/myproject/pods/my-two-container-pod
```