Kubernetes WSL Kind Podman desktop

`2024-08-23T11:46:23.428+02:00`

These instructions will enable you to run `podman`, `kubectl` and `kind` on Windows 11 using Podman Desktop.

Set Up Kind

1. Install Podman Desktop
2. During installation install `compose`, `kubectl`, `podman` and set up `podman-machine-default`
3. Install `kubens` and `kubectx` via `choco` (first install `choco` if not installed)

   ```
   choco install kubens kubectx
   ```

4. Create a yaml-file for the kind cluster `cluster-01.yml` with following content

   ```
   kind: Cluster
   apiVersion: kind.x-k8s.io/v1alpha4
   nodes:
   - role: control-plane
     kubeadmConfigPatches:
     - |
       kind: InitConfiguration
       nodeRegistration:
         kubeletExtraArgs:
           node-labels: "ingress-ready=true"
     extraPortMappings:
     - containerPort: 80
       hostPort: 80
       protocol: TCP
   ```

5. Set up the kind cluster in the console

   ```
   kind create cluster --config cluster-01.yml --name cluster-01
   # [output]
   # enabling experimental podman provider
   # Creating cluster "cluster-01" ...
   #  ✓ Ensuring node image (kindest/node:v1.30.0) 🖼
   #  ✓ Preparing nodes 📦
   #  ✓ Writing configuration 📜
   #  ✓ Starting control-plane 🕹️
   #  ✓ Installing CNI 🔌
   #  ✓ Installing StorageClass 💾
   # Set kubectl context to "kind-cluster-01"
   # You can now use your cluster with:

   # kubectl cluster-info --context kind-cluster-01

   # Thanks for using kind! 😊
   ```

6. Check if the cluster was created

   ```
   kubectl get node --show-labels
   # [output]
   # NAME                       STATUS   ROLES           AGE   VERSION   LABELS
   # cluster-01-control-plane   Ready    control-plane   32m   v1.30.0
   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,ingress-
   ready=true,kubernetes.io/arch=amd64,kubernetes.io/hostname=cluster-01-control-plane,kubernetes.io/os=linux,node-
   role.kubernetes.io/control-plane=
   ```

7. Create a `namespace.yml` with following content

```
apiVersion: v1
kind: Namespace
metadata:
  name: echo-space
```

8. Set up the namespace

```
kubectl apply -f .\namespace.yml
```

9. Activate namespace defined in `namespace.yml`

```
kubens echo-space
# [output] > ✔ Active namespace is "echo-space"
```

10. Create new deployment

```
kubectl create deployment echo-app --image=k8s.gcr.io/echoserver:1.4
```

11. Create a load balancer and expose the service

```
kubectl expose deployment echo-app --type=LoadBalancer --port=80 --target-port=8080
```

12. Confirm service and service and pod are running

```
kubectl get pod -o wide
# [output]
# NAME                           READY   STATUS    RESTARTS   AGE     IP           NODE                         NOMINATED NODE
READINESS GATES
# echo-app-677767f685-s2lvz    1/1     Running   0          3m58s   10.244.0.5   cluster-01-control-plane   <none>
<none>
```

13. Deploy the nginx controller directly from the repo

```
kubectl apply --filename=https://raw.githubusercontent.com/kubernetes/ingress-
nginx/master/deploy/static/provider/kind/deploy.yaml
# [output]
# namespace/ingress-nginx created
# serviceaccount/ingress-nginx created
# serviceaccount/ingress-nginx-admission created
# role.rbac.authorization.k8s.io/ingress-nginx created
# role.rbac.authorization.k8s.io/ingress-nginx-admission created
# clusterrole.rbac.authorization.k8s.io/ingress-nginx created
# clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
# rolebinding.rbac.authorization.k8s.io/ingress-nginx created
# rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
# clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
# clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
# configmap/ingress-nginx-controller created
# service/ingress-nginx-controller created
# service/ingress-nginx-controller-admission created
# deployment.apps/ingress-nginx-controller created
# job.batch/ingress-nginx-admission-create created
# job.batch/ingress-nginx-admission-patch created
# ingressclass.networking.k8s.io/nginx created
# validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created

kubectl wait --namespace=ingress-nginx --for=condition=ready pod --selector=app.kubernetes.io/component=controller --timeout=180s
# [output] > pod/ingress-nginx-controller-8fb8cdb7c-jqgv7 condition met
```

14. Create file `ingress.yml` for ingress controller

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: echo-ingress
  namespace: echo-space
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: echo-app
            port:
              number: 80
```

15. Apply the yml file for the ingress controller

```
kubectl apply -f .\ingress.yml
# [output] > ingress.networking.k8s.io/echo-ingress created
```

16. Check that the ingress controller is correctly set up

```
kubectl describe ingress echo-ingress
# [output]
# Name:             echo-ingress
# Labels:           <none>
# Namespace:        echo-space
# Address:          localhost
# Ingress Class:    <none>
# Default backend:  <default>
# Rules:
#   Host        Path  Backends
#   ----        ----  --------
#   *
#               /   echo-app:80 (10.244.0.5:8080)
# Annotations:  nginx.ingress.kubernetes.io/rewrite-target: /
# Events:
#   Type    Reason  Age                    From                      Message
#   ----    ------  ----                   ----                      -------
#   Normal  Sync    2m56s (x2 over 3m41s)  nginx-ingress-controller  Scheduled for sync
```

17. Send curl request to localhost to check if the service is running
    ```
    curl localhost
    # [output]
    # StatusCode        : 200
    # StatusDescription : OK
    # Content           : CLIENT VALUES:
    #                      client_address=10.244.0.8
    #                      command=GET
    #                      real path=/
    #                      query=nil
    #                      request_version=1.1
    #                      request_uri=http://localhost:8080/

    #                      SERVER VALUES:
    #                      server_version=nginx: 1.10.0 - lua: 10001

    #                      HEADERS REC...
    # RawContent        : HTTP/1.1 200 OK
    #                      Transfer-Encoding: chunked
    #                      Connection: keep-alive
    #                      Content-Type: text/plain
    #                      Date: Fri, 23 Aug 2024 08:25:14 GMT

    #                      CLIENT VALUES:
    #                      client_address=10.244.0.8
    #                      command=GET
    #                      real path=/
    #                      q...
    # Forms             : {}
    # Headers           : {[Transfer-Encoding, chunked], [Connection, keep-alive], [Content-Type, text/plain], [Date, Fri, 23 Aug
    2024 08:25:14 GMT]}
    # Images            : {}
    # InputFields       : {}
    # Links             : {}
    # ParsedHtml        : mshtml.HTMLDocumentClass
    # RawContentLength  : 541
    ```
18. Reset everything to working conditions (podman, kubectl etc. will keep being installed)
    ```
    kubectl delete service echo-app
    # [output] > service "echo-app" deleted
    kubectl delete deployment echo-app
    # [output] > deployment.apps "echo-app" deleted
    kubens default
    # [output] > ✔ Active namespace is "default"
    kubectl delete namespace echo-space
    # [output] > namespace "echo-space" deleted
    kubectl delete --filename=https://raw.githubusercontent.com/kubernetes/ingress-
    nginx/master/deploy/static/provider/kind/deploy.yaml
    # [output] > delete statements for all the services in the ingress file
    ```

Installing Krew Packet Manager and Stern

1. Download Krew (`krew.exe`) from here
2. Open an elevated administrator `cmd`, go to download folder and execute

   ```
   .\krew install krew
   ```

3. Add `krew` binary folder to your `PATH` variable (folder C:/Users/[username]/.krew/bin)
4. Restart your `shell`
5. Now install `stern` from another elevated administrator `cmd`

   ```
   kubectl krew install stern
   # [output]
   # Updated the local copy of plugin index.
   # Installing plugin: stern
   # Installed plugin: stern
   # \
   #  | Use this plugin:
   #  |      kubectl stern
   #  | Documentation:
   #  |      https://github.com/stern/stern
   # /
   # WARNING: You installed plugin "stern" from the krew-index plugin repository.
   #    These plugins are not audited for security by the Krew maintainers.
   #    Run them at your own risk.
   ```

6. Now you can run `stern` with following commands

   ```
   kubectl-stern . --all-namespaces
   # [output] > all logs of all namespaces … please be aware that this can be a lot of logs
   kubectl-stern . -n kube-system --tail 0
   # [output] > log output of namespace "kube-system"
   ```

7. Read up on more use cases for stern here