Setting up Kubernetes on bare metal VM instance for Local Development

These are shell commands to configure Kubernetes bare metal on a VM instance.

```
# We need to allow IP forwarding here and preserve on reboot.
sudo cat <<EOF | sudo tee /etc/sysctl.d/k8s-networking.conf
net.ipv4.ip_forward                 = 1
EOF

sudo sysctl --system

# Find the latest Kubernetes versions.
sudo dnf list kubernetes1.??

# Install K8s and CRI-O daemon and CLI, CNI plugins and Helm packager utility.
sudo dnf install kubernetes1.33 kubernetes1.33-kubeadm kubernetes1.33-client cri-o1.33 cri-tools1.33
containernetworking-plugins helm

# Enable FirewallD (if it's not already running, you should, security first!).
sudo systemctl enable --now firewalld

# Allow CRI-O to be able pull images from the local insecure Podman registry (this directory isn't created
by the RPM for some reason).
sudo mkdir -v /etc/crio/crio.conf.d

sudo cat <<EOF | sudo /etc/crio/crio.conf.d/local-registry.conf
[crio.image]
```

```
insecure_registries = ["localhost:5000"]
EOF
```

Enable CRI-O for container runtime management daemon.

```
sudo systemctl enable --now crio
```

Pull K8s images for setup.

```
sudo kubeadm config images pull
```

Enable Kubernetes node agent itself.

```
sudo systemctl enable --now kubelet
```

Download and install the latest Flannel networking CNI plugin.

```
sudo bash -c "cd /tmp && curl -L -Ss
https://github.com/flannel-io/cni-plugin/releases/download/v1.7.1-flannel1/cni-plugin-flannel-linux-amd64
-v1.7.1-flannel1.tgz | tar -zxvf - && mv -v flannel-amd64 /usr/libexec/cni/flannel"
```

Enable the local image registry to be started on reboot, we'll use this for a demo app later in the next Post.

```
sudo cat <<EOF | sudo tee /etc/containers/systemd/local-registry.volume
[Unit]
Description=Podman Local Registry Volume

[Volume]
Label=app=local-registry
EOF
```

```
sudo cat <<EOF | sudo tee /etc/containers/systemd/local-registry.container
[Unit]
Description=Podman Local Registry Container

[Container]
Label=app=registry
ContainerName=registry
Image=registry:2
Volume=local-registry.volume:/var/lib/registry
PublishPort=5000:5000

[Install]
WantedBy=multi-user.target default.target
EOF

sudo systemctl daemon-reload
sudo systemctl start local-registry.service
```

```
# Setup Kubernetes with a Pod Network and set the Container runtime to use CRI-O.
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --cri-socket=unix:///var/run/crio/crio.sock
```

```
# Give your local user access to the Kubernetes control plane.
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
# Allow Pods to be scheduled onto this control plane, since we're a single node here
kubectl taint nodes fedora.example.com node-role.kubernetes.io/control-plane:NoSchedule-
```

```
# Install the latest Flannel K8s resources.
export ARCH=amd64
curl -sSL "https://github.com/coreos/flannel/blob/master/Documentation/kube-flannel.yml?raw=true" | sed
"s/amd64/${ARCH}/g" | kubectl create -f -

# Setup and install local for storage class to use for Persistent volumes.
cat >> /tmp/local-storage.yaml << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/local-storage-provisioner
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Retain # You can use Delete too as needed
parameters:
  path: /mnt/data # Path on the node
EOF

kubectl create -f /tmp/local-storage.yaml

# Install latest Kubernetes Gateway API framework.
kubectl apply -f
https://github.com/kubernetes-sigs/gateway-api/releases/download/v1.2.1/standard-install.yaml

# Install latest release and configure MetalLB which is a bare metal load balancer you can use without a
cloud provider.
kubectl apply -f
```

```
https://raw.githubusercontent.com/metallb/metallb/v0.14.9/config/manifests/metallb-native.yaml
```

Configure MetalLB, I just use the VM's IP address here, the IP network is bridged

```
cat > /tmp/metalb-config.yaml << EOF
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: my-ip-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.2.100/32 # The IP of VM instance is on a Bridge network
EOF

kubectl apply -f /tmp/metalb-config.yaml
```

Install latest Istio release.

```
curl -L https://istio.io/downloadIstio | sh -
cd istio-1.26.0/
export PATH=$PWD/bin:$PATH
istioctl install --set profile=minimal -y
```

Create the test namespace.

```
kubectl create ns test
```

Optional: Ensure Istio sidecar proxy is injected in the test namespace, you may use the annotation in your application **Deployment** instead.

```
kubectl label namespace test istio-injection=enabled
```

Create CA and Server cert and create a Secret for the Gateway, setup accordingly.

```
openssl genrsa -aes256 -out example-ca.key 4096
openssl req -x509 -new -nodes -key example-ca.key -sha256 -days 1024 -out example-ca.crt
openssl genrsa -out example-server.key 4096
openssl req -new -key example-server.key -out example-server.csr
openssl x509 -req -in example-server.csr -CA example-ca.crt -CAkey example-ca.key -CAcreateserial -out
example-server.crt -days 500 -sha256

kubectl create -n test secret generic tls-cert --from-file=tls.key=example-server.key
--from-file=tls.crt=example-server.crt --from-file=ca.crt=example-ca.crt
```

Create a Gateway resource, we'll only run this in our test namespace.

```
cat > /tmp/gateway.yaml << EOF
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: test
  namespace: test
spec:
  gatewayClassName: istio
  listeners:
  - name: https
    hostname: '*.example.com'
    port: 443
    protocol: HTTPS
    tls:
      mode: Terminate
```

```
      certificateRefs:
        - name: tls-cert
    allowedRoutes:
      namespaces:
        from: Same

kubectl create -f /tmp/gateway.yaml
```

From here, you should have a functional Kubernetes node running in your VM.

```
$ kubectl get nodes -o wide
NAME               STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE                              KERNEL-VERSION               CONTAINER-RUNTIME
fedora.example.com   Ready    control-plane   5h25m   v1.33.1   192.168.2.100   <none>        Fedora Linux 43 (Rawhide Prerelease)   6.15.0-0.rc7.58.fc43.x86_64   cri-o://1.33.0

$ kubectl get gateway -n test
NAME   CLASS   ADDRESS         PROGRAMMED   AGE
test   istio   192.168.2.100   True         3m46s

$ kubectl get ipaddresspools -n metallb-system
NAME         AUTO ASSIGN   AVOID BUGGY IPS   ADDRESSES
my-ip-pool   true          false             ["192.168.2.100/32"]

$ kubectl get svc -n test
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                         AGE
test-istio   LoadBalancer   10.110.95.223   192.168.2.100   15021:30521/TCP,443:30824/TCP   3m43s
```