Deciphering Kubernetes Networking

Victor Morales

Victor Morales

- +18 yrs as a Software Engineer
- .NET, Java, python, Go programmer
- OpenStack, OPNFV, ONAP and CNCF contributor.

https://about.me/electrocucaracha

Main goal

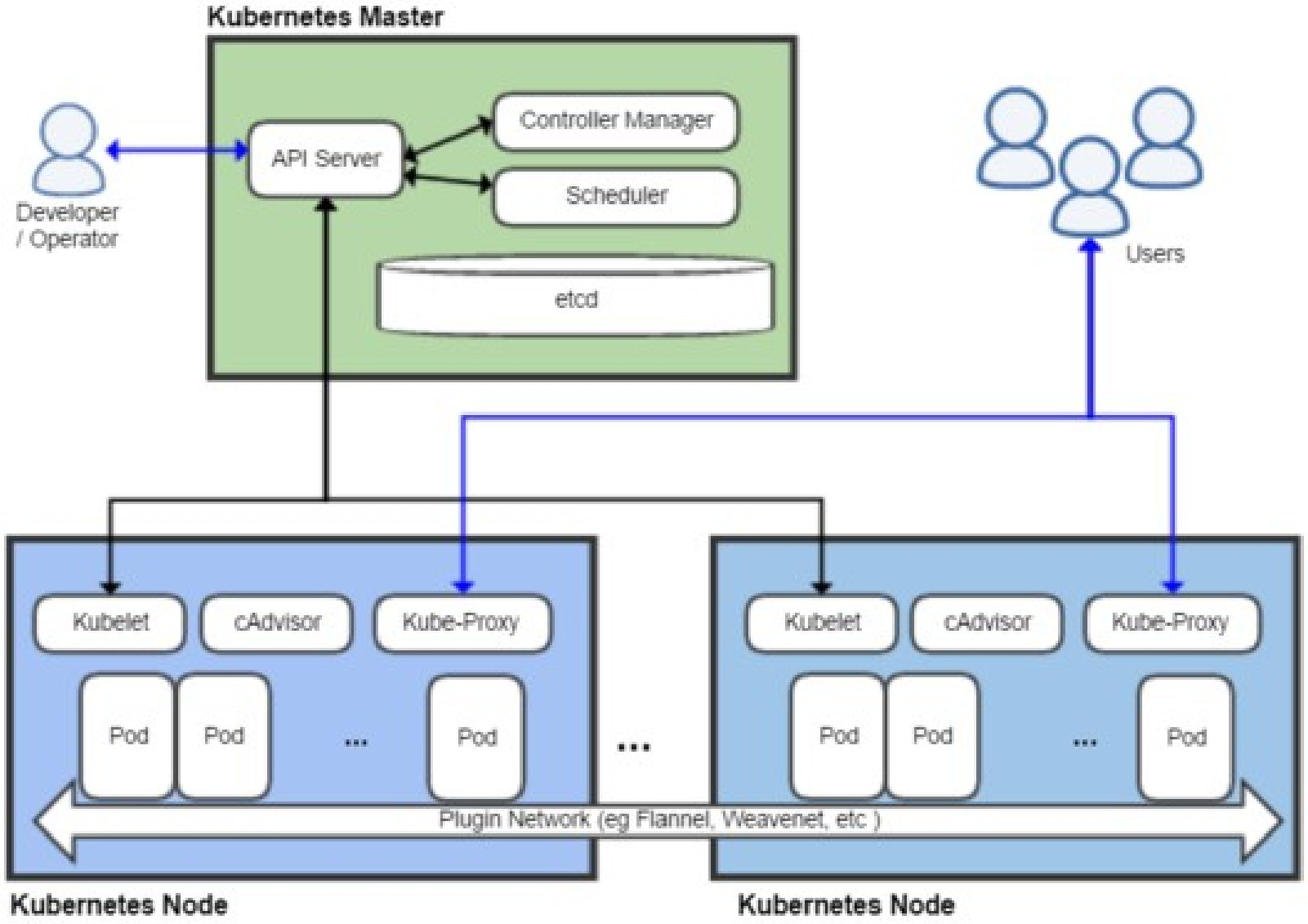
Understand the Containers Networking setup process during the creation of Pods in Kubernetes

References:

- https://www.altoros.com/blog/kubernetes-networking-writing-your-own-simple-cni-plug-in-with-bash/
- https://www.tkng.io/cni/
- https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/

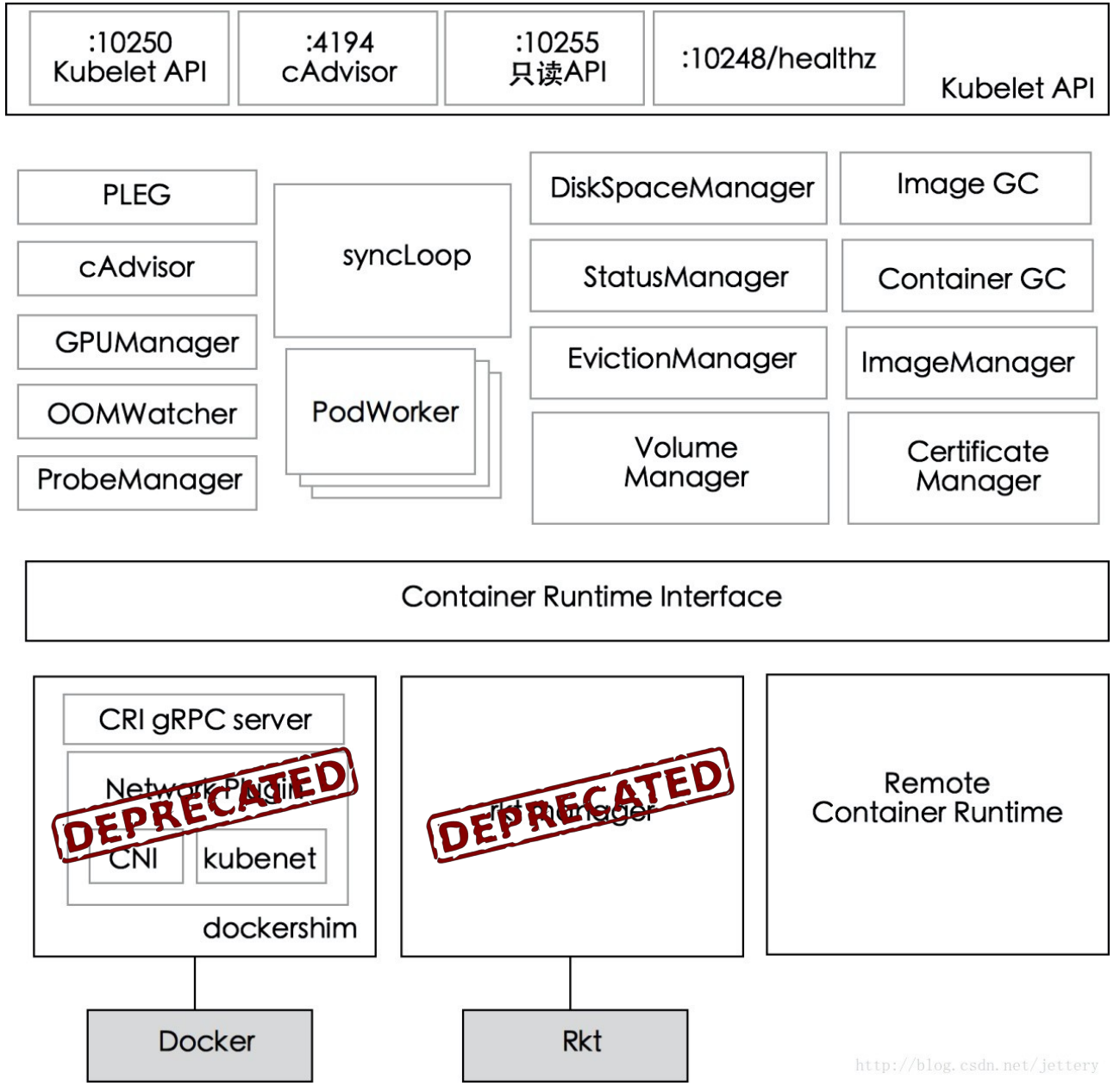
kubernetes

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.



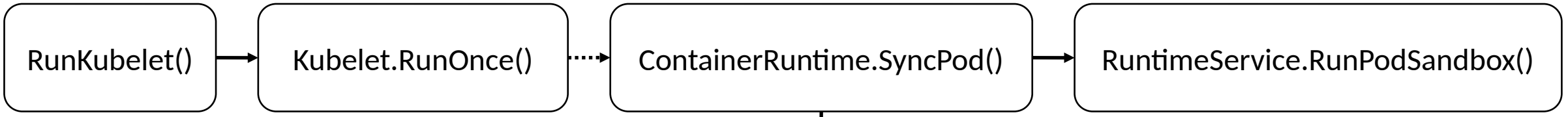
Kubelet’s component architecture

1. Kubelet API
2. syncLoop layer (https://github.com/kubernetes/kubernetes/blob/v1.24.2/pkg/kubelet/kubelet.go#L1980-L2025)
   - PLEG
   - cAdvisor
   - PodWorkers
   - OOMWatcher
   - Container GC
   - Image GC
   - Managers
3. Container Runtime Interface



https://www.sobyte.net/post/2022-03/kubelet-pod-creation-workflow/

Kubelet workflow



Syncs the running pod into the desired pod.

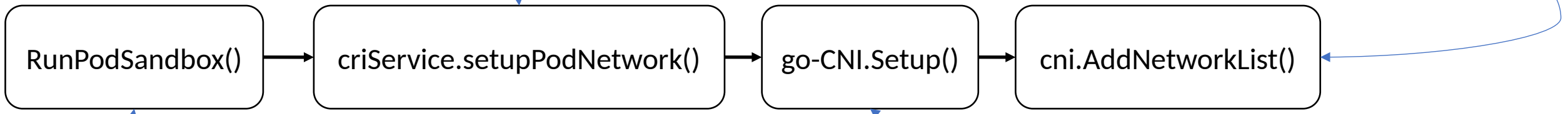
1. Compute sandbox and container changes
2. Kill pod sandbox if necessary
3. Kill any containers that should not be running
4. **Create sandbox if necessary**
5. Create ephemeral containers
6. Create init containers
7. Create normal containers

https://github.com/kubernetes/kubernetes/blob/v1.24.2/pkg/kubelet/kuberuntime/kuberuntime_manager.go#L800

ContainerD workflow

https://github.com/containerd/containerd/blob/v1.6.6/pkg/cri/server/sandbox_run.go#L376-L405

https://github.com/containernetworking/cni/blob/v1.1.1/libcni/api.go#L417-L433



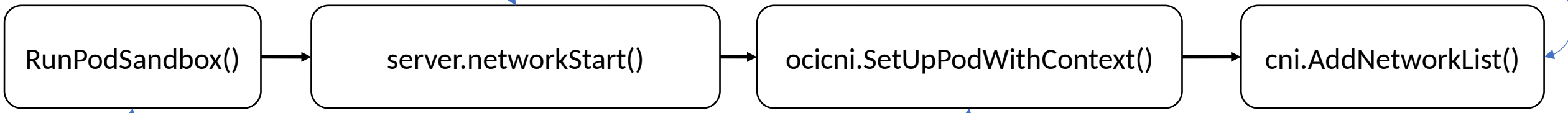
https://github.com/containerd/go-cni/blob/main/cni.go#L154-L168

https://github.com/containerd/containerd/blob/v1.6.6/pkg/cri/server/sandbox_run.go#L59-L359

CRI-O workflow

https://github.com/cri-o/cri-o/blob/v1.24.1/server/sandbox_network.go#L22-L129

https://github.com/containernetworking/cni/blob/v1.1.1/libcni/api.go#L417-L433



https://github.com/cri-o/ocicni/blob/v0.4.0/pkg/ocicni/ocicni.go#L539-L574

https://github.com/cri-o/cri-o/blob/v1.24.1/server/sandbox_run.go#L65-L69

```
417  // AddNetworkList executes a sequence of plugins with the ADD command
418  func (c *CNIConfig) AddNetworkList(ctx context.Context, list *NetworkConfigList, rt *RuntimeConf
419          var err error
420          var result types.Result
421          for _, net := range list.Plugins {
422                  result, err = c.addNetwork(ctx, list.Name, list.CNIVersion, net, result, rt)
423                  if err != nil {
424                          return nil, fmt.Errorf("plugin %s failed (add): %w", pluginDescription(r
425                  }
426          }
427
428          if err = c.cacheAdd(res
429                  return nil, fmt
430          }
431
432          return result, nil
433  }
```

```
393  func (c *CNIConfig) addNetwork(ctx context.Context, name, cniVersion string, net *NetworkConfig, prevResul
394          c.ensureExec()
395          pluginPath, err := c.exec.FindInPath(net.Network.Type, c.Path)
396          if err != nil {
397                  return nil, err
398          }
399          if err := utils.ValidateContainerID(rt.ContainerID); err != nil {
400                  return nil, err
401          }
402          if err := utils.ValidateNetworkName(name); err != nil {
403                  return nil, err
404          }
405          if err := utils.ValidateInterfaceName(rt.IfName); err != nil {
406                  return nil, err
407          }
408
409          newConf, err := buildOneConfig(name, cniVersion, net, prevResult, rt)
410          if err != nil {
411                  return nil, err
412          }
413
414          return invoke.ExecPluginWithResult(ctx, pluginPath, newConf.Bytes, c.args("ADD", rt), c.exec)
415  }
```

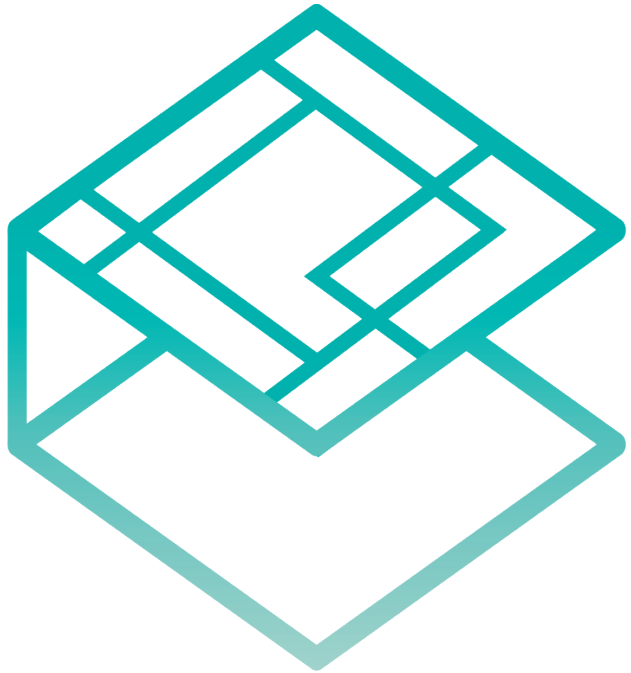
libcn
i

invoke

```
16
17  import (
18          "bytes"
19          "context"
20          "encoding/json"
21          "fmt"
22          "io"
23          "os/exec"
24          "strings"
25          "time"
26
27          "github.com/containernetworking/cni/pkg/types"
28  )
```

```
115 func ExecPluginWithResult(ctx context.Context, pluginPath string, netconf [
116         if exec == nil {
117                 exec = defaultExec
118         }
119
120         stdoutBytes, err := exec.ExecPlugin(ctx, pluginPath, netconf, args.
121         if err != nil {
122                 return nil, err
123         }
124
125         resultVersion, fixedBytes, err := fixupResultVersion(netconf, stdou
126         if err != nil {
127                 return nil, err
128         }
129
130         return create.Create(resultVersion, fixedBytes)
131 }
```

```
33
34  func (e *RawExec) ExecPlugin(ctx context.Context, pluginPath string, stdinData []byte, en
35          stdout := &bytes.Buffer{}
36          stderr := &bytes.Buffer{}
37          c := exec.CommandContext(ctx, pluginPath)
38          c.Env = environ
39          c.Stdin = bytes.NewBuffer(stdinData)
40          c.Stdout = stdout
41          c.Stderr = stderr
42
43          // Retry the command on "text file busy" errors
44          for i := 0; i <= 5; i++ {
45                  err := c.Run()
46
47                  // Command succeeded
48                  if err == nil {
49                          break
50                  }
51
52                  // If the plugin is currently about to be written, then we wait a
53                  // second and try it again
54                  if strings.Contains(err.Error(), "text file busy") {
55                          time.Sleep(time.Second)
56                          continue
57                  }
58
59                  // All other errors except than the busy text file
60                  return nil, e.pluginErr(err, stdout.Bytes(), stderr.Bytes())
61          }
62
63          // Copy stderr to caller's buffer in case plugin printed to both
64          // stdout and stderr for some reason. Ignore failures as stderr is
65          // only informational.
66          if e.Stderr != nil && stderr.Len() > 0 {
67                  _, _ = stderr.WriteTo(e.Stderr)
68          }
69          return stdout.Bytes(), nil
70  }
```



CNI

https://github.com/cncf/artwork

CNI (*Container Network Interface*), a Cloud Native Computing Foundation project, consists of a **specification** and **libraries** for writing plugins to configure network interfaces in Linux containers, along with a number of supported plugins. CNI concerns itself only with **network connectivity of containers** and removing allocated resources when the container is deleted.

ADD: Add container to network, or apply modifications

A CNI plugin, upon receiving an `ADD` command, should either

- create the interface defined by `CNI_IFNAME` inside the container at `CNI_NETNS`, or
- adjust the configuration of the interface defined by `CNI_IFNAME` inside the container at `CNI_NETNS`.

If the CNI plugin is successful, it must output a result structure (see below) on standard out. If the plugin was supplied a `prevResult` as part of its input configuration, it MUST handle `prevResult` by either passing it through, or modifying it appropriately.

If an interface of the requested name already exists in the container, the CNI plugin MUST return with an error.

A runtime should not call `ADD` twice (without an intervening DEL) for the same `(CNI_CONTAINERID, CNI_IFNAME)` tuple. This implies that a given container ID may be added to a specific network more than once only if each addition is done with a different interface name.

Input:

The runtime will provide a JSON-serialized plugin configuration object (defined below) on standard in.

Required environment parameters:

- `CNI_COMMAND`
- `CNI_CONTAINERID`
- `CNI_NETNS`
- `CNI_IFNAME`

Optional environment parameters:

- `CNI_ARGS`
- `CNI_PATH`

https://www.cni.dev/docs/spec/#add-add-container-to-network-or-apply-modifications

CNI plugins Ecosystem

flannel

cilium

PROJECT CALICO

weave net

https://github.com/cncf/artwork

CNI plugin written in BASH

https://github.com/electrocucaracha/k8s-NetworkingDeepDive-demo/tree/master/bash

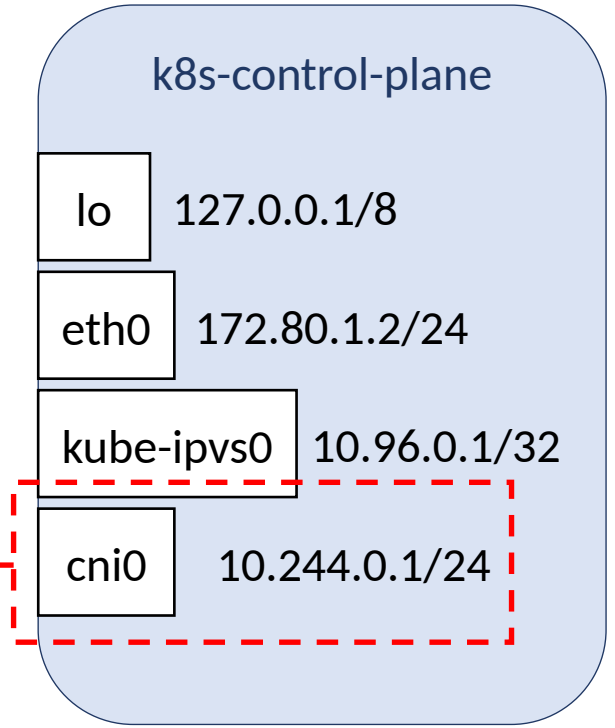
Setup

```
21  function _get_pod_cidr {
22      pod_cidr=""
23      attempt_counter=0
24      max_attempts=5
25
26      until [ "$pod_cidr" ]; do
27          pod_cidr=$(kubectl get node "$1" -o jsonpath='{.spec.podCIDR}')
28          if [ "$pod_cidr" ]; then
29              echo "$pod_cidr"
30              break
31          elif [ ${attempt_counter} -eq ${max_attempts} ];then
32              error "Max attempts reached"
33          fi
34          attempt_counter=$((attempt_counter+1))
35          sleep $((attempt_counter*2))
36      done
37  }
```

```
81  for node in $(sudo docker ps --filter "name=k8s-*" --format "{{.Names}}"); do
82      pod_cidr=$(_get_pod_cidr "$node")
83      cat << EOF > /tmp/10-bash-cni-plugin.conf
84  {
85    "cniVersion": "0.3.1",
86    "name": "mynet",
87    "type": "bash-cni",
88    "network": "$network_id",
89    "subnet": "$pod_cidr"
90  }
91  EOF
92      cloud_init="
93  brctl addbr cni0
94  ip link set cni0 up
95  ip addr add ${pod_cidr%.*}.1/24 dev cni0
96  "
97      sudo docker cp /tmp/10-bash-cni-plugin.conf "$node":/etc/cni/net.d/10-bash-cni-plugin.conf
98      sudo docker exec "$node" bash -c "$cloud_init"
99  done
```

Bridge

A bridge behaves like a network switch. It forwards packets between interfaces that are connected to it. It's usually used for forwarding packets on routers, on gateways, or between VMs and network namespaces on a host.



Demo(ContainerD
)

```
kubectl run test --image=busybox:1.35.0 -- sleep infinity
trap 'kubectl delete pod test' EXIT
kubectl wait --for=condition=Ready pod test

info "Getting the IP address assigned to the Pod"
kubectl exec test -- ip address show eth0
for node in $(sudo docker ps --filter "name=k8s-*" --format "{{.Names}}"); do
    sudo docker exec "$node" cat /var/log/bash-cni-plugin.log
done
```

```
166 function main {
167     [[ "$CNI_ARGS" == *'K8S_POD_NAMESPACE=default;'* ]] && export DEBUG=true
168
169     exec 3>&1 # make stdout available as fd 3 for the result
170     exec &>> /var/log/bash-cni-plugin.log
171
172     stdin=$(cat /dev/stdin)
173     debug "stdin: $stdin"
174
175     debug "CNI envs: $(printenv | grep CNI_)"
176     # CNI_COMMAND: indicates the desired operation
177     case $CNI_COMMAND in
178         ADD)
179             add
180         ;;
181         DEL)
182             del
183         ;;
184         GET)
185             error "GET not supported"
186         ;;
187         VERSION)
188             echo '{"cniVersion": "0.3.1","supportedVersions": [ "0.3.0", "0.3.1", "0.4.0" ]}' >&3
189         ;;
190         *)
191             echo "Unknown cni command: $CNI_COMMAND"
192             exit 1
193         ;;
194     esac
195 }
196
197 if [[ "${__name__:-"__main__"}" == "__main__" ]]; then
198     main
199 fi
```

```
21:13:29 - DEBUG: CNI envs: CNI_CONTAINERID=130d9e7a9b5505372e1db478bbf95f5bb46
CNI_IFNAME=eth0
CNI_NETNS=/var/run/netns/cni-099e7256-f5fa-a0d2-9bdf-61829dc0aa4a
CNI_COMMAND=ADD
CNI_PATH=/opt/cni/bin
CNI_ARGS=K8S_POD_INFRA_CONTAINER_ID=130d9e7a9b5505372e1db478bbf95f5bb4697793983
```

```
function add {
    subnet=$(echo "$stdin" | jq -r ".subnet")
    subnet_mask_size="${subnet#*/}"

    # IPAM
    info "Discover IP addresses"
    output=$(allocate_ip "$subnet")
    # shellcheck disable=SC2206
    output=(${output[@]})
    gw_ip=${output[0]}
    debug "gw_ip: $gw_ip"
    container_ip=${output[1]}
    if [[ -z "$container_ip" ]]; then
        [ -f "$reserved_ips_file" ] && rm "$reserved_ips_file"
        error "It couldn't discover an IP address for the container"
    fi
    debug "container_ip: $container_ip"
    _add_rollback "sed -i \"/$container_ip/d\" $reserved_ips_file;"
```

```
21:13:29 - INFO: Discover IP addresses
21:13:29 - DEBUG: gw_ip: 10.244.0.1
21:13:29 - DEBUG: container_ip: 10.244.0.5
```

```
function allocate_ip {
    local subnet="$1"
    local output=""

    all_ips=$(_get_all_ip_list "$subnet")
    # shellcheck disable=SC2206
    all_ips=(${all_ips[@]})
    if (( ${#all_ips[@]} == 0 )); then
        [ -f "$all_ips_file" ] && rm "$all_ips_file"
        error "The IP addresses list is empty"
    fi
    output+="${all_ips[1]}\n"
    reserved_ips=$(_get_reserved_ip_list "${all_ips[0]}" "${all_ips[1]}")

    for ip in "${all_ips[@]}"; do
        if [[ "${reserved_ips[*]}" != *${ip}* ]]; then
            echo "$ip" >> $reserved_ips_file
            output+="$ip\n"
            break
        fi
    done

    echo -e "$output"
}

function _get_all_ip_list {
    if [ ! -f "$all_ips_file" ]; then
        prips "$1" > "$all_ips_file"
    fi
    cat "$all_ips_file"
}

function _get_reserved_ip_list {
    reserved_ips=$(cat $reserved_ips_file 2> /dev/null || printf '%s\n' "$@" )
    # shellcheck disable=SC2206
    reserved_ips=(${reserved_ips[@]})
    printf '%s\n' "${reserved_ips[@]}" | sort | uniq | tee  $reserved_ips_file
}
```

```
113     info "Binding IP address"
114     if_name="$(_get_rand_if_name)"
115     host_if_name="veth$if_name"
116     tmp_if_name="tmp$if_name"
117     echo ""
118     # CNI_CONTAINERID: Container ID.
119     if ip link show type veth "$tmp_if_name" > /dev/null; then
120         error "The $CNI_CONTAINERID container is unable to use $tmp
121     fi
122     debug "host_if_name: $host_if_name"
123     ip link add "$tmp_if_name" type veth peer name "$host_if_name"
124     _add_rollback "ip link delete $tmp_if_name;"
125
```

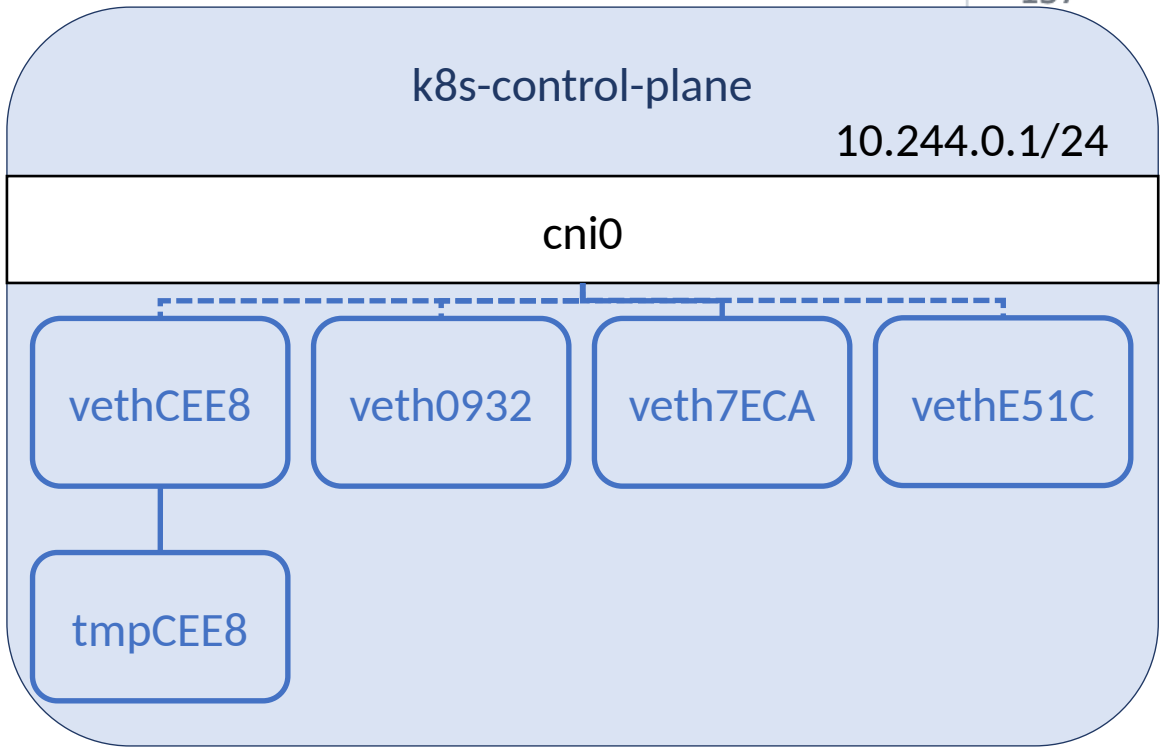
```
21:13:29 - INFO: Binding IP address
Device "tmpCEE8" does not exist.

21:13:29 - DEBUG: host_if_name: vethCEE8
```

VETH

The VETH (virtual Ethernet) device is a local Ethernet tunnel. Devices are created in pairs, packets transmitted on one device in the pair are immediately received on the other device. These 2 devices can be imagined as being connected by a network cable.

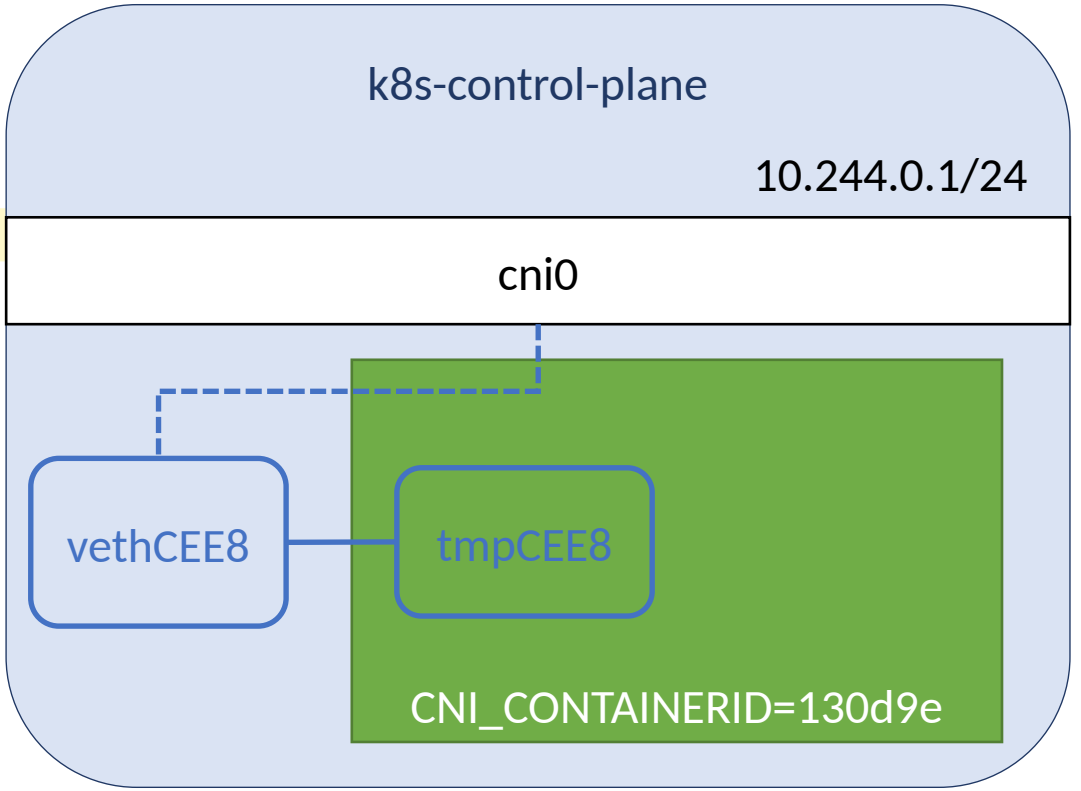
```
126     info "Connecting $host_if_name to cni0"
127     ip link set "$host_if_name" up
128     ip link set "$host_if_name" master cni0
129     if [[ "${DEBUG:-false}" == "true" ]]; then
130         echo ""
131         brctl show cni0
132     fi
133
134     # NOTE: Enable ip netns access to CNI_NETNS namespace
135     mkdir -p /var/run/netns/
136     # CNI_NETNS: A reference to the container's "isolation domain".
137     ln -sfT "$CNI_NETNS" "/var/run/netns/$CNI_CONTAINERID"
    _add_rollback "rm -rf /var/run/netns/$CNI_CONTAINERID;"
```

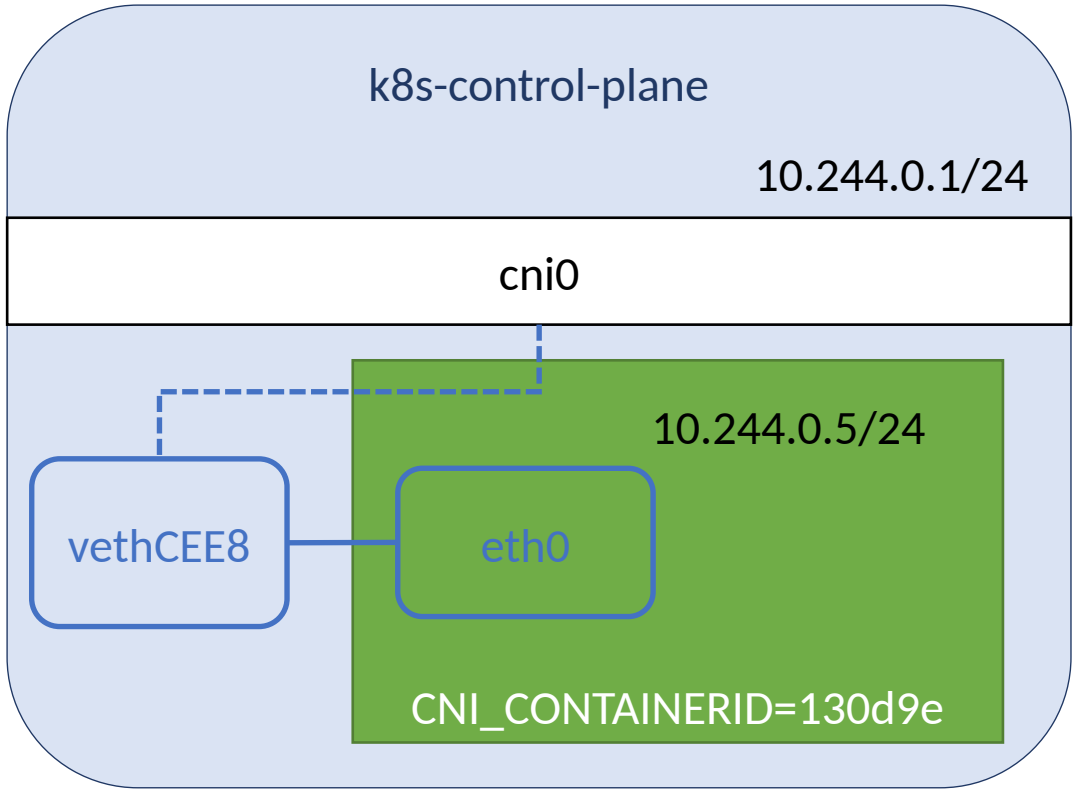



```
21:13:29 - INFO: Connecting vethCEE8 to cni0
bridge name     bridge id               STP enabled     interfaces
cni0            8000.525b9233bb86       no              veth0932
                                                        veth7ECA
                                                        vethCEE8
                                                        vethE51C
```

```
21:13:29 - INFO: Setting tmpCEE8 of 130d9e7a9b5505372e1db478bbf95f5bb46977939833d2568a849a214e7414dc container
```

```
134     # NOTE: Enable ip netns access to CNI_NETNS namespace
135     mkdir -p /var/run/netns/
136     # CNI_NETNS: A reference to the container's "isolation domain".
137     ln -sfT "$CNI_NETNS" "/var/run/netns/$CNI_CONTAINERID"
138     _add_rollback "rm -rf /var/run/netns/$CNI_CONTAINERID;"
139
140     info "Setting $tmp_if_name of $CNI_CONTAINERID container"
141     ip link set "$tmp_if_name" netns "$CNI_CONTAINERID"
142
```






```
143     # CNI_IFNAME: Name of the interface to create inside the container
144     ip netns exec "$CNI_CONTAINERID" ip link set "$tmp_if_name" name "$CNI_IFNAME"
145     ip netns exec "$CNI_CONTAINERID" ip link set "$CNI_IFNAME" up
146     ip netns exec "$CNI_CONTAINERID" ip addr add "$container_ip/$subnet_mask_size" dev "$CNI_IFNAME"
147     ip netns exec "$CNI_CONTAINERID" ip route add default via "$gw_ip" dev "$CNI_IFNAME"
```

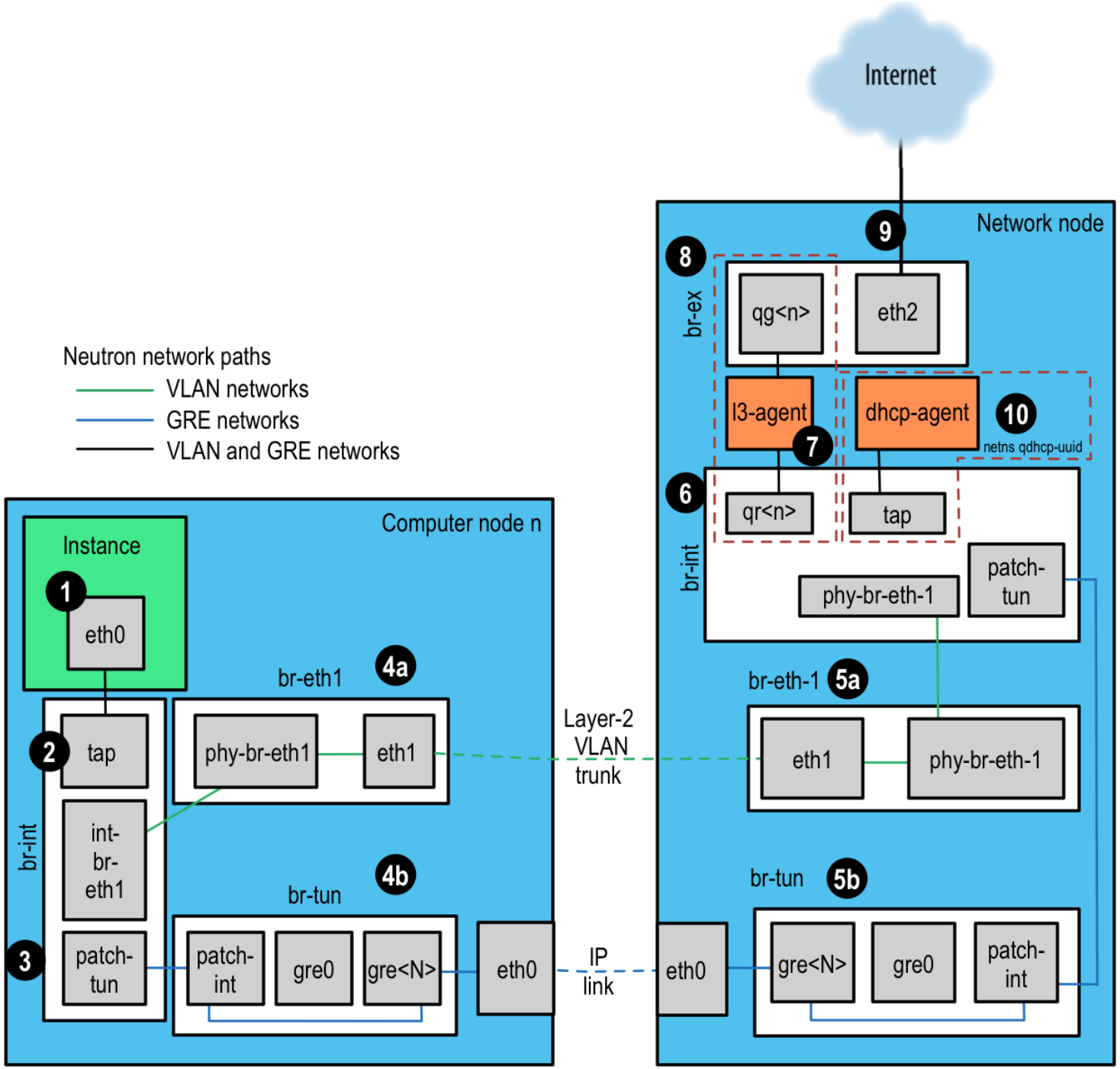
```
149     mac=$(ip netns exec "$CNI_CONTAINERID" ip link show "$CNI_IFNAME" | awk '/ether/ {print $2}')
150     sdtout="{\"cniVersion\": \"0.3.1\",
151     \"interfaces\": [{\"name\": \"$CNI_IFNAME\",\"mac\": \"$mac\",\"sandbox\": \"$CNI_NETNS\"}],
152     \"ips\": [{\"version\": \"4\",\"address\": \"$container_ip/$subnet_mask_size\",
153     \"gateway\": \"$gw_ip\",\"interface\": 0}]}"
154     debug "sdtout: $(echo "$sdtout" | jq -r .)"
155     echo "$sdtout" >&3
156 }
```

```
21:13:29 - DEBUG: sdtout: {
  "cniVersion": "0.3.1",
  "interfaces": [
    {
      "name": "eth0",
      "mac": "6a:4c:78:6a:e9:d7",
      "sandbox": "/var/run/netns/cni-099e7256-f5fa-a0d2-9bdf-61829dc0aa4a"
    }
  ],
  "ips": [
    {
      "version": "4",
      "address": "10.244.0.5/24",
      "gateway": "10.244.0.1",
      "interface": 0
    }
  ]
}
pod "test" deleted
```

Is that related with Neutron?

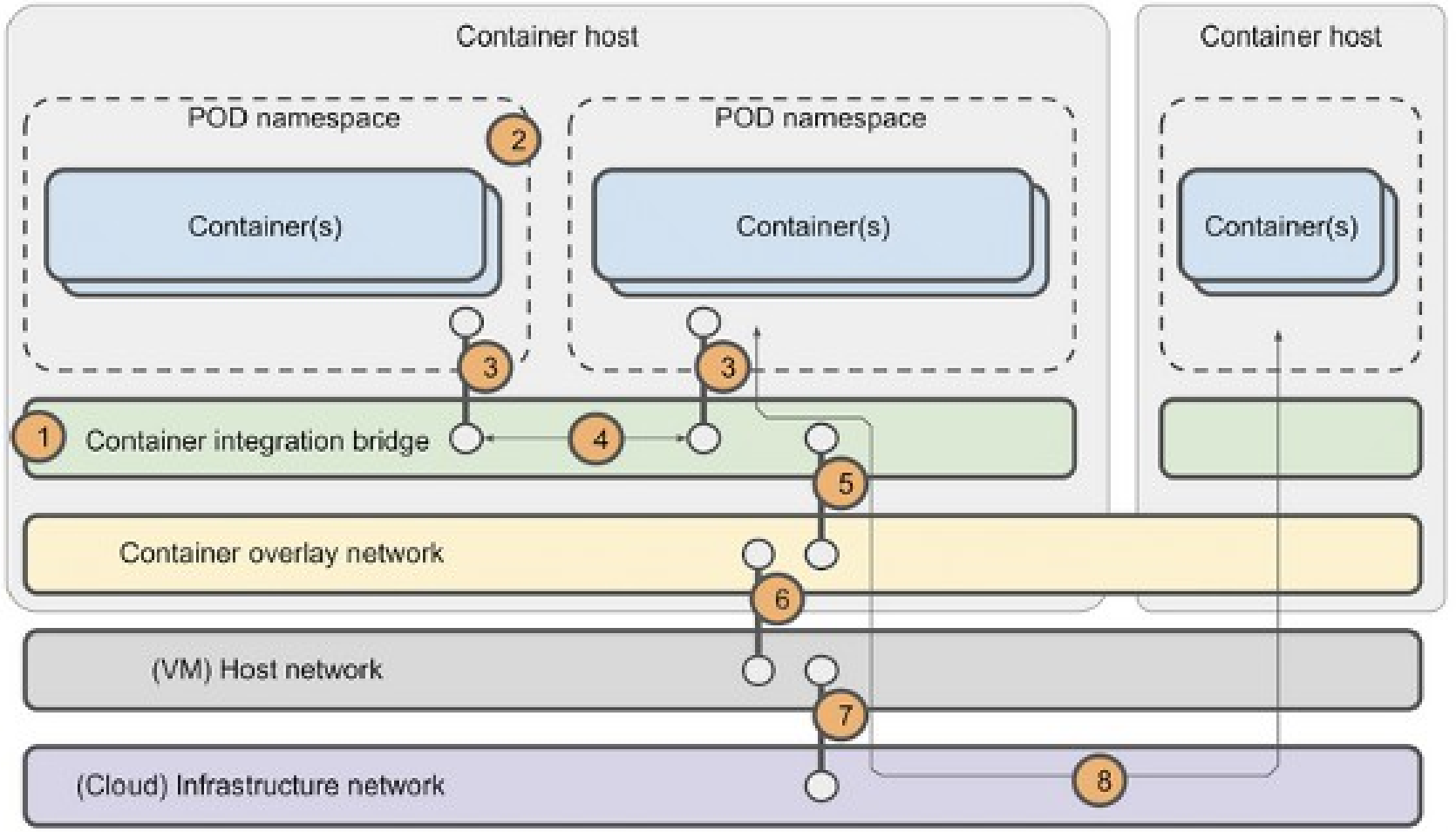
NEUTRON

an OpenStack Community Project



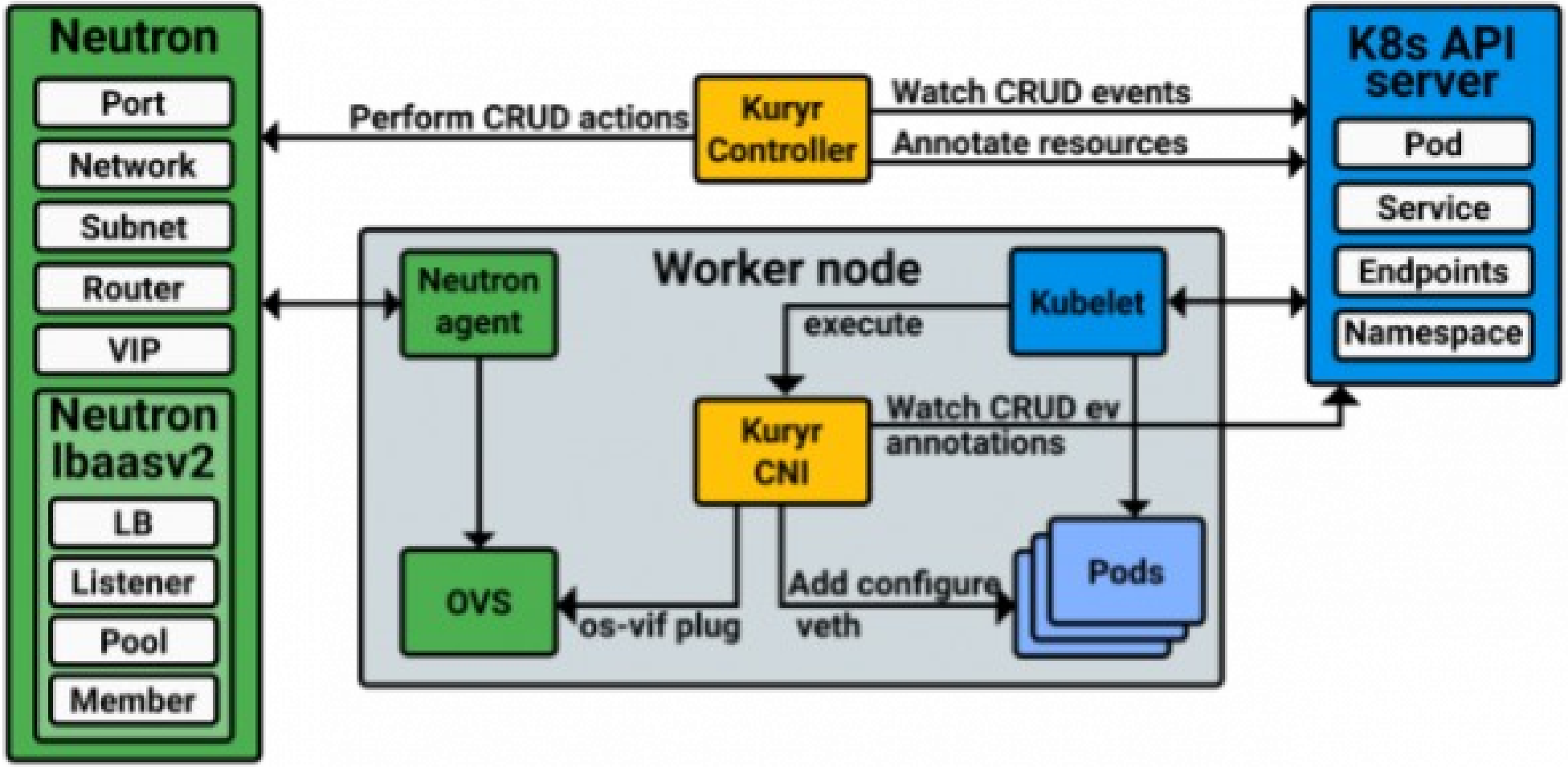
https://docs.openstack.org/operations-guide/ops-network-troubleshooting.html#visualizing-openstack-networking-service-traffic-in-the-cloud

https://www.eficode.com/blog/debugging-kubernetes-networking



Double-tunneling

will have negative impact on data-plane **performance** (e.g. Kubernetes ‘flannel’ tunnel encapsulated in OpenStack ‘vxlan’ tunnel when running Kubernetes on top of OpenStack).



The two Kuryr-Kubernetes components depicted with all the main components they interact with.

KURYR

an OpenStack Community Project

https://docs.openstack.org/kuryr-kubernetes/latest/

https://superuser.openstack.org/articles/networking-kubernetes-kuryr/

Thanks/Gracias

@electrocucaracha