
BWI
IT für Deutschland

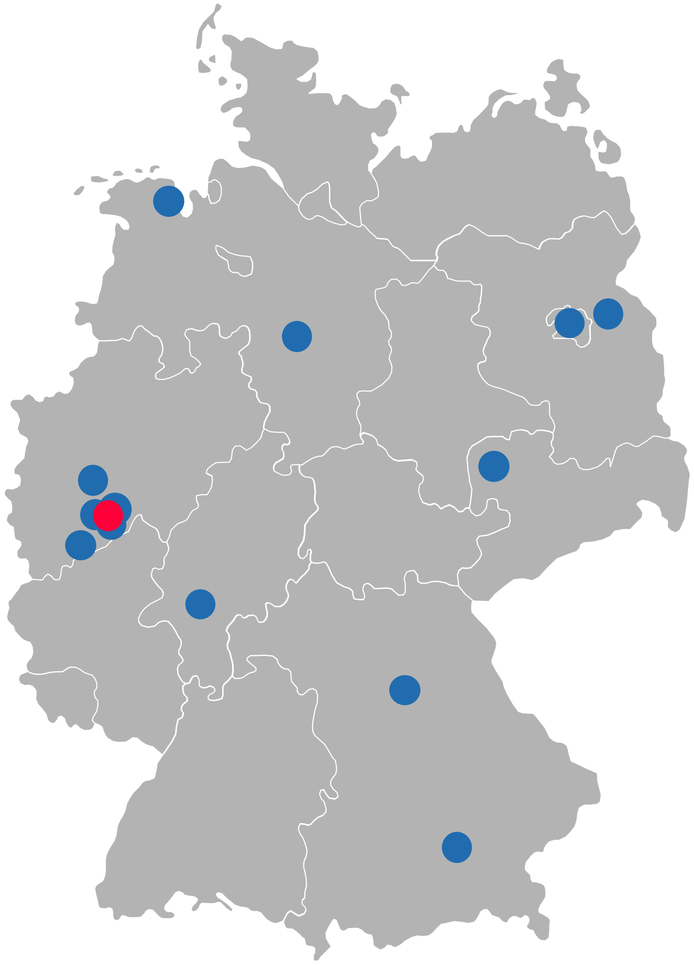
VPCs in Kubernetes

Niklas Voss

Image source: BWI/Löschke

BWI is the **Bundeswehr's primary partner** of digitalization in peace, crisis, and war.

BUNDESWEHR

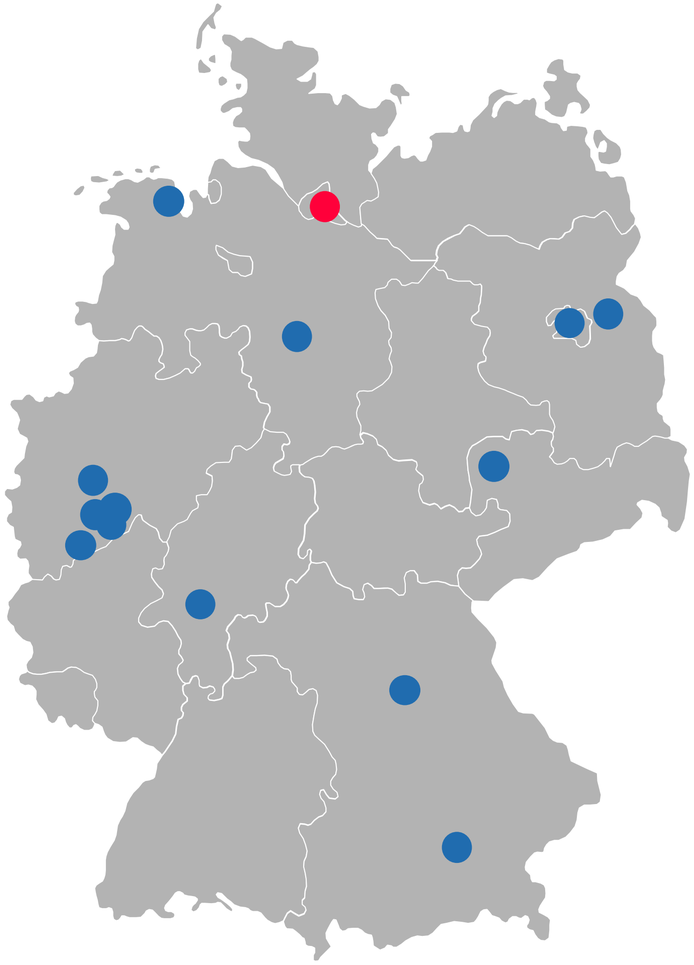


Regional presence with **14 major sites**, **>130** locations throughout Germany

Headquarters in Meckenheim

100% in-house company of the federal government

Niklas Voss
Tech Lead Private Cloud
Bundeswehr

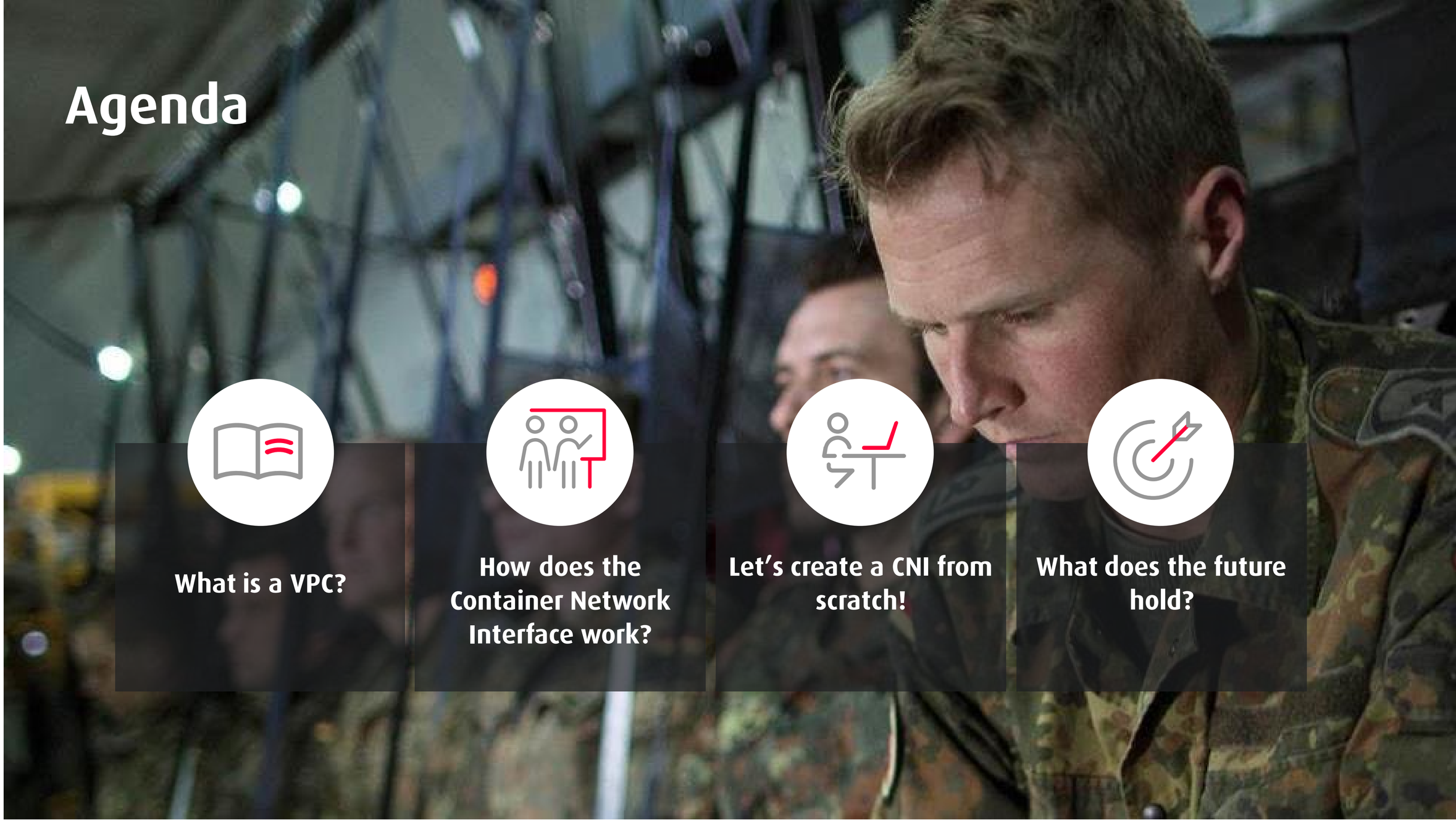


Previous employers include
Google, Finleap, ...

Living
in **Hamburg**

Image source: BWI/Graphics

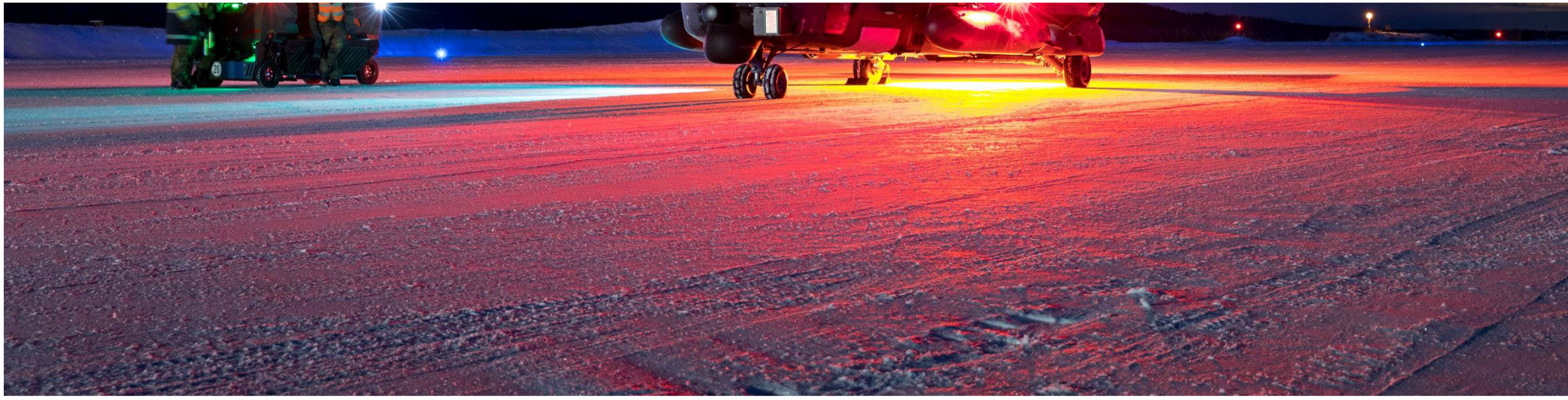
Agenda



- What is a VPC?
- How does the Container Network Interface work?
- Let’s create a CNI from scratch!
- What does the future hold?

Image source: Bundeswehr/Andrea Bienert

What is a VPC?



Quelle: Bundeswehr/Marco Durow

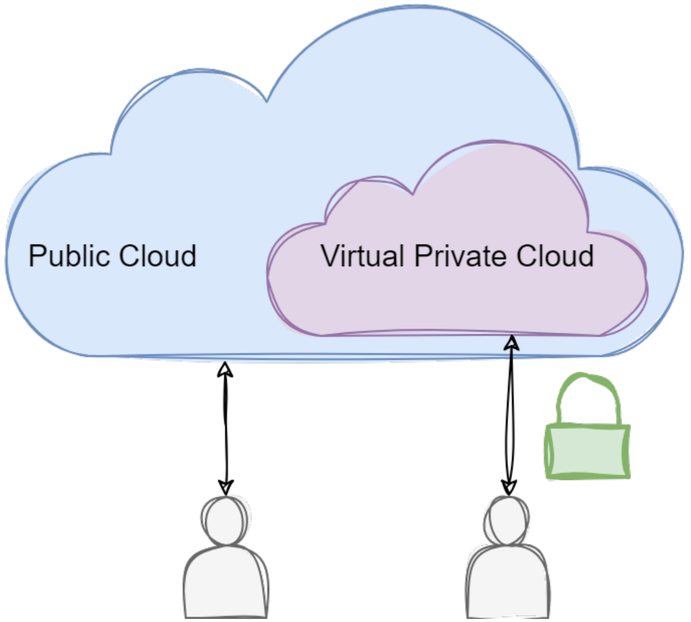
What is a VPC

According to [Cloudflare]:

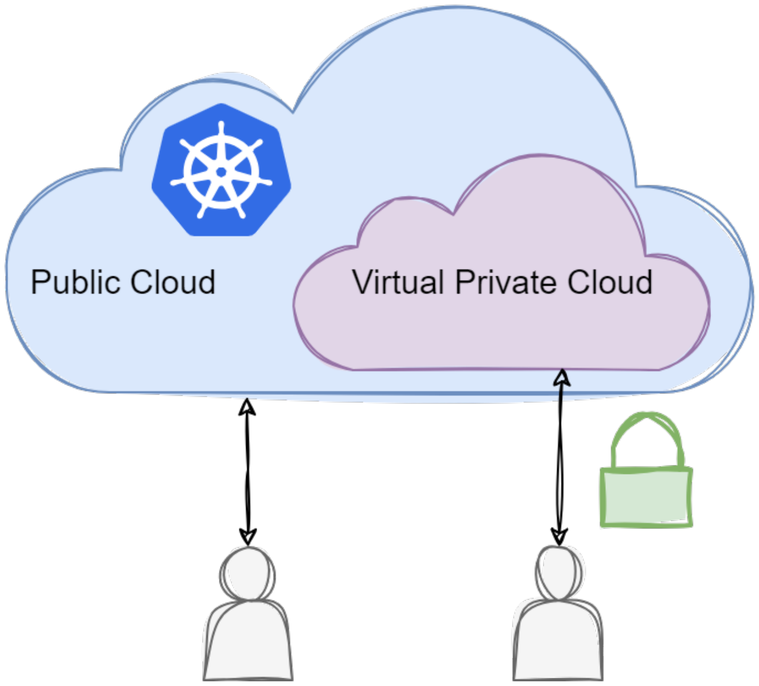
"A virtual private cloud (VPC) is a secure, isolated private cloud hosted within a public cloud."

➔ Strong tenancy isolation

➔ For Cloud Providers like AWS and Google Cloud it is a **network construct**!

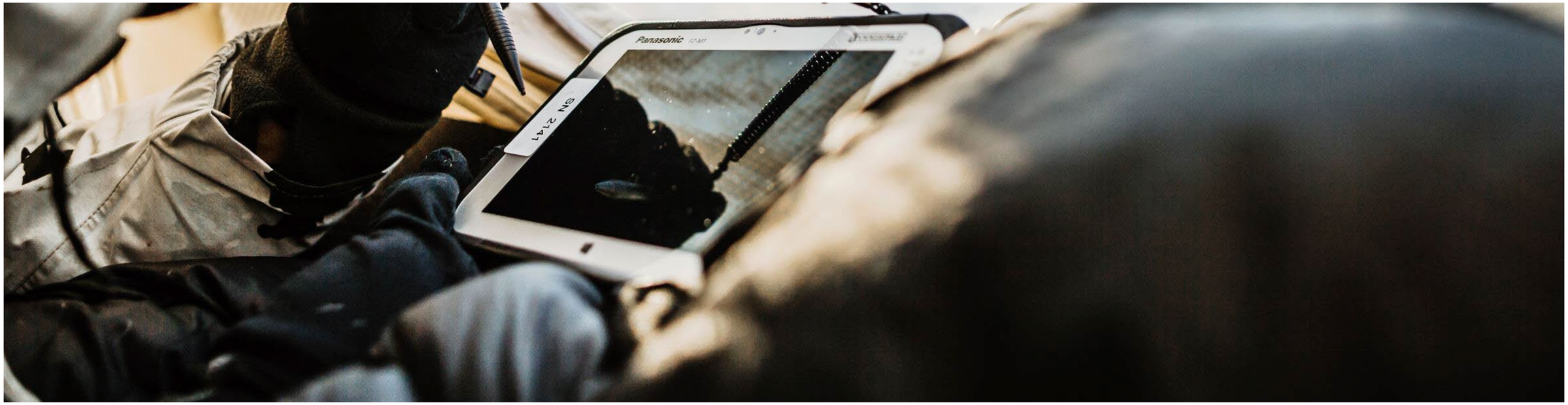


We want to build a Cloud on top of Kubernetes



Kubernetes Logo: kubernetes/logo/LICENSE at master · kubernetes/kubernetes · GitHub

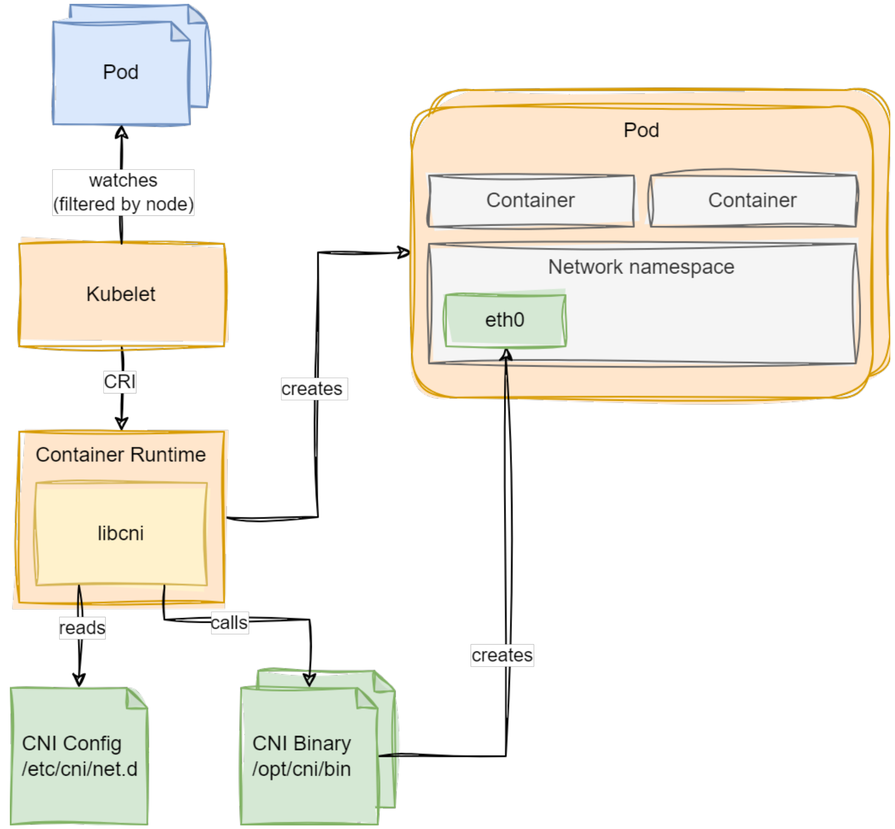
How does the Container Network Interface work?



Bildquelle: Bundeswehr/Jana Neumann

From Kubelet to CNI

1. The kubelet watches pods for its node
2. Pods are created via CRI
3. Container Runtime sets up the containers and namespaces and interacts with libcni
4. libcni reads the CNI config and calls CNI binary
5. Container is started



CNI Config

CNI config is JSON!

WHAT!? NO YAML!?

CNI is independent of Kubernetes and agnostic of container orchestrators.

The spec is intentionally simple to implement.

```
{
  "cniVersion": "1.1.0",
  "name": "mynet",
  "plugins": [
    {
      "type": "bridge",
      // plugin specific paramaters
      "bridge": "cni0",
      "keyA": ["some more", "plugin specific", "configuration"],
      "ipam": {
        "type": "host-local",
        // ipam specific
        "subnet": "10.1.0.0/16",
        "gateway": "10.1.0.1",
        "routes": [
            {"dst": "0.0.0.0/0"}
        ]
      },
      "dns": {
        "nameservers": [ "10.1.0.1" ]
      }
    }
  ]
}
```


Pikachu Source: Surprised Pikachu keychain and magnet von plomdawg | Kostenloses STL-Modell herunterladen | Printables.com

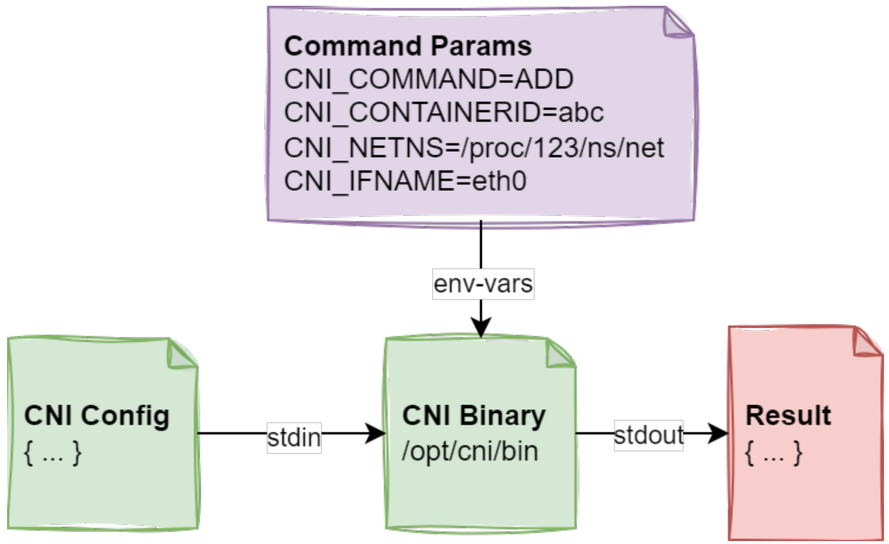
Code Diagram Source: BWI/Voss

CNI Config

- CNI config is in JSON format
- Versioned CNI specification
  - So far: 0.1.0 → 0.2.0 → 0.3.1 → 0.4.0 → 1.0.0 → 1.1.0
- Plugins can be chained
- Provides sane defaults for handling IPAM, DNS and more
- Container Runtimes will only take first config (first sorted by ASCII)

```
{
  "cniVersion": "1.1.0",
  "name": "mynet",
  "plugins": [
    {
      "type": "bridge",
      // plugin specific parameters
      "bridge": "cni0",
      "keyA": ["some more", "plugin specific", "configuration"],
      "ipam": {
        "type": "host-local",
        // ipam specific
        "subnet": "10.1.0.0/16",
        "gateway": "10.1.0.1",
        "routes": [
            {"dst": "0.0.0.0/0"}
        ]
      },
      "dns": {
        "nameservers": [ "10.1.0.1" ]
      }
    }
  ]
}
```

The CNI Binary

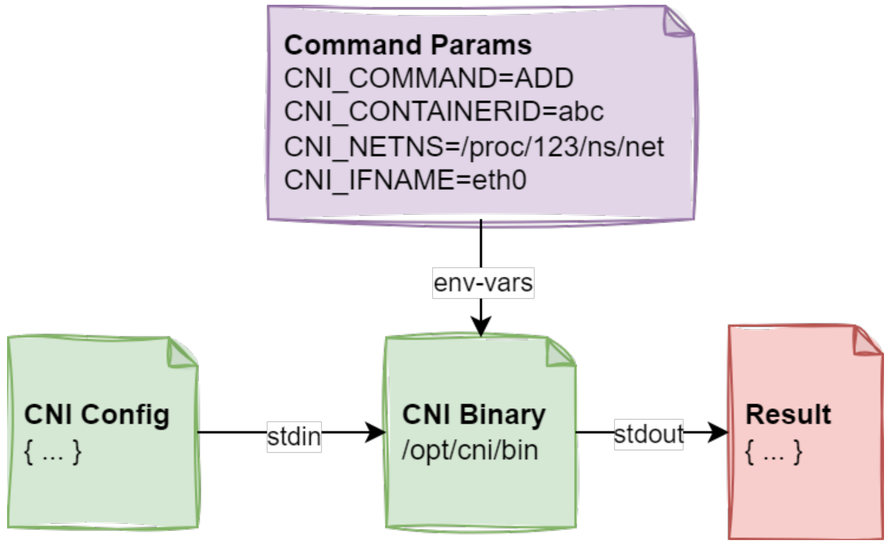


- Config provided on stdin
- Parameters / command arguments passed via environment variables
- Result on stdout

Available Commands:

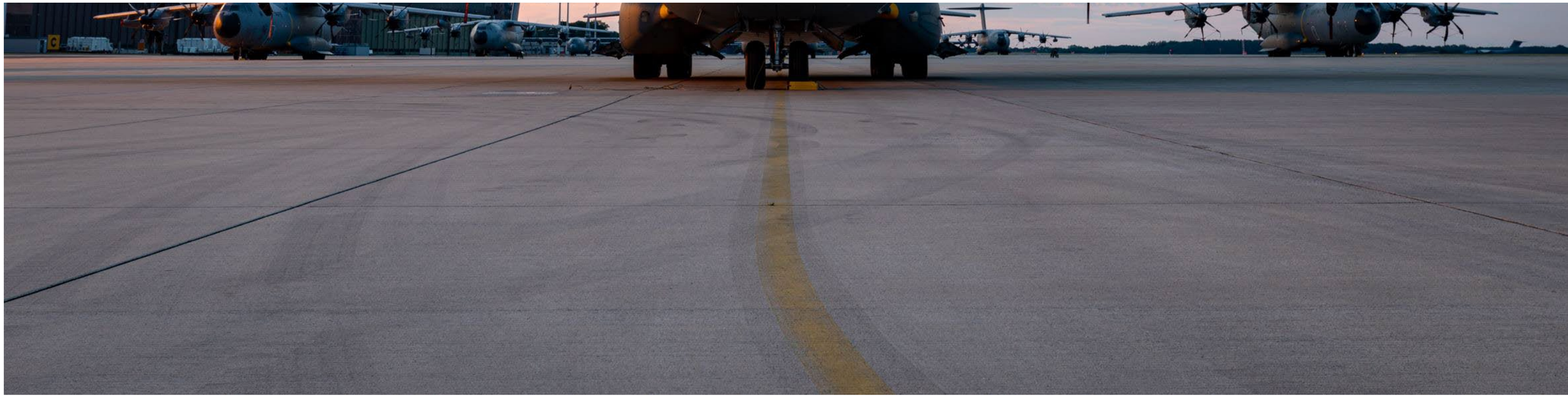
- **ADD**: Add container to network, or apply modifications
- **DEL**: Remove container from network, or un-apply modifications
- **CHECK**: Check container's networking is as expected
- **STATUS**: Check plugin status
- **VERSION**: probe plugin version support
- **GC**: Clean up any stale resources

The CNI Binary



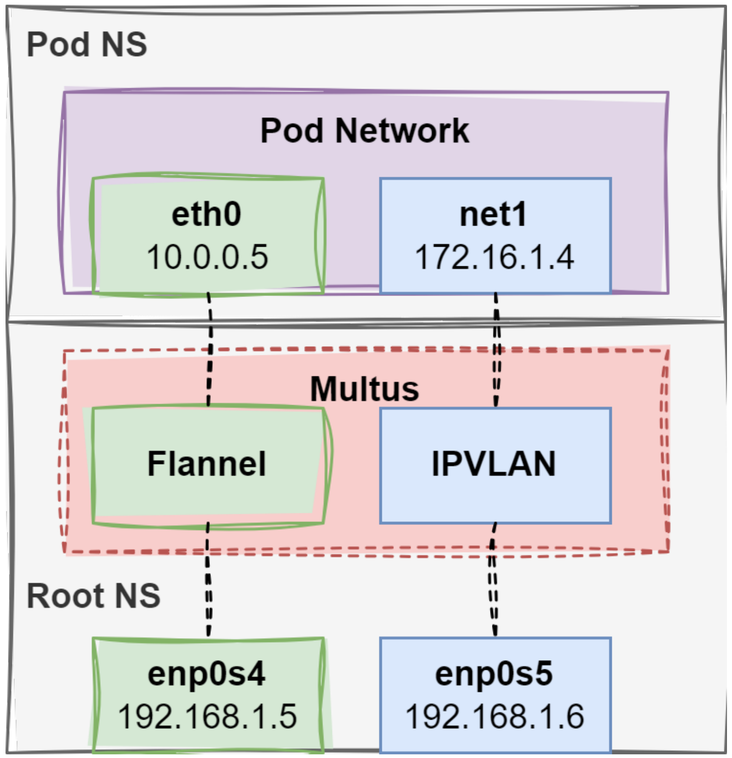

```
{
    "ips": [
        {
          "address": "10.1.0.5/16",
          "gateway": "10.1.0.1"
        }
    ],
    "routes": [
      {
        "dst": "0.0.0.0/0"
      }
    ],
    "dns": {
      "nameservers": [ "10.1.0.1" ]
    }
}
```

Let’s create a CNI from Scratch!



Bildquelle: Bundeswehr/Patrik Bransmöller

Multiple CNIs

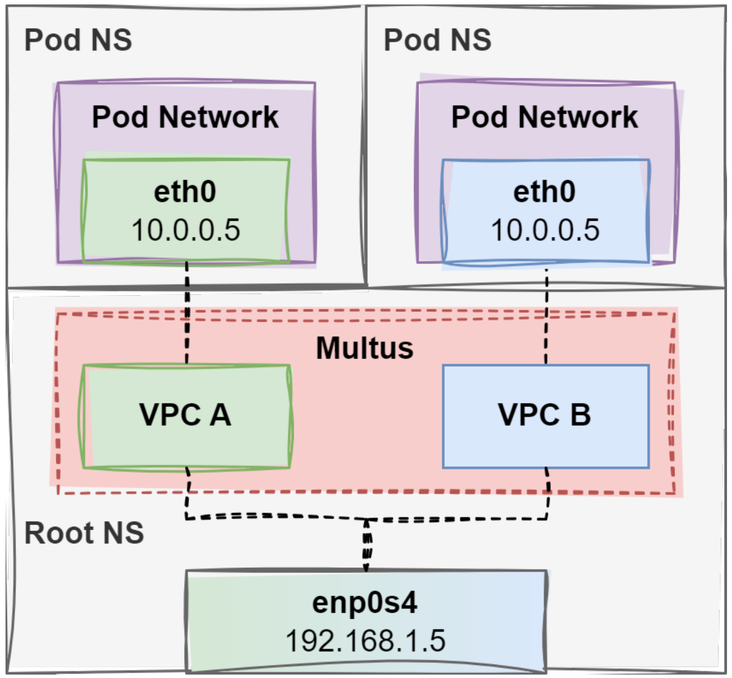


MULTUS

- Multus CNI is “meta-plugin”, a CNI plugin that can call multiple other CNI plugins
- Allows attaching multiple network interfaces to pods
- Override default network

Multus Logo: GitHub - k8snetworkplumbingwg/multus-cni: A CNI meta-plugin for multi-homed pods in Kubernetes
Diagram Source: BWI/Voss

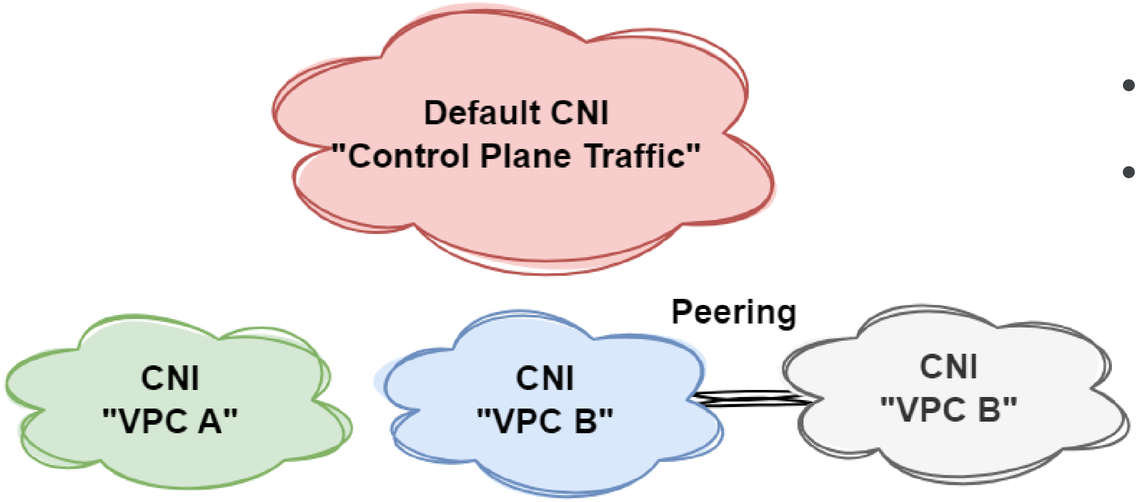
Multiple CNIs



- We need to override the default network
- Every VPC is a separate CNI config
  - ➔ Same binary, but different config
  - ➔ Each CNI is an independent overlay network

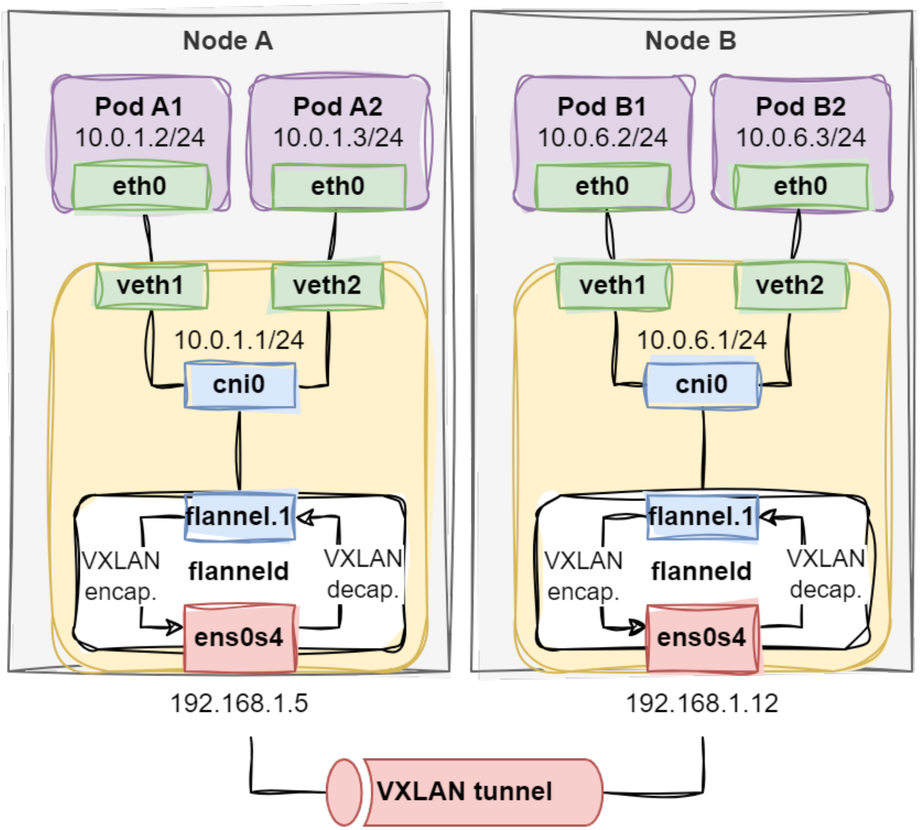
Diagram Source: BWI/Voss

Multiple CNIs



- We need to override the default network
- Every VPC is a separate CNI config
  - ➔ Same binary, but different config
  - ➔ Each CNI is an independent overlay network

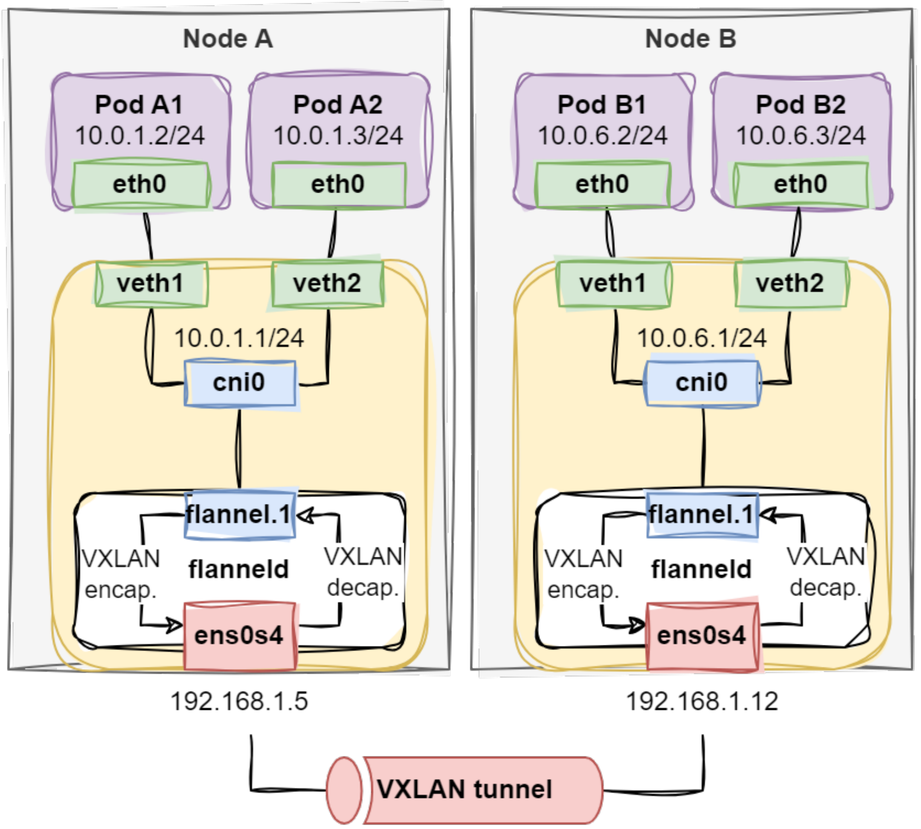
VXLAN



- Most CNI implementations today leverage VXLAN (Cilium, Calico, Flannel, ...)
- Every node gets a subnet
- Traffic within Pod IP range is encapsulated

Diagram Source: BWI/Voss

VXLAN




```
# Create VXLAN and bridge once per node
ip link add vxlan200 type vxlan \
        id 200 \
        dstport 4789 \
        local ${NODE_IP} \
        nolearning
ip link add br200 type bridge
ip link set vxlan200 master br200
ip link set br200 type bridge stp_state 0
ip link set up dev br200
ip link set up dev vxlan200

# Create veth pair for each pod
ip link add dev ${DEV} type veth peer name ${POD_DEV}
ip link set dev ${POD_DEV} netns ${POD_NS}
ip addr add ${ADDR} dev ${DEV}
ip link set ${DEV} master br200
ip link set dev ${DEV} up
```


Diagram Source: BWI/Voss

Implementation in Go

[plugins/plugins/sample/main.go at main · containernetworking/plugins · GitHub]

```go
package main

import (
    "encoding/json"

    "github.com/containernetworking/cni/pkg/skel"
    "github.com/containernetworking/cni/pkg/types"
    "github.com/containernetworking/cni/pkg/version"
    bv "github.com/containernetworking/plugins/pkg/utils/buildversion"
)

type PluginConf struct {
    types.NetConf

    MyAwesomeFlag     bool   `json:"myAwesomeFlag"`
    AnotherAwesomeArg string `json:"anotherAwesomeArg"`
}


func parseConfig(stdin []byte) (*PluginConf, error) {
    conf := PluginConf{}
    if err := json.Unmarshal(stdin, &conf); err != nil {
        return nil, fmt.Errorf("failed to parse network configuration: %v", err)
    }
    // ...
}

func cmdAdd(args *skel.CmdArgs) error {
    conf, err := parseConfig(args.StdinData)
    // ...

    return types.PrintResult(result, conf.CNIVersion)
}

func cmdDel(args *skel.CmdArgs) error {}
func cmdCheck(_ *skel.CmdArgs) error {}

func main() {
    skel.PluginMainFuncs(skel.CNIFuncs{
        Add:   cmdAdd,
        Check: cmdCheck,
        Del:   cmdDel,
    }, version.All, bv.BuildString("mycni"))
}
```

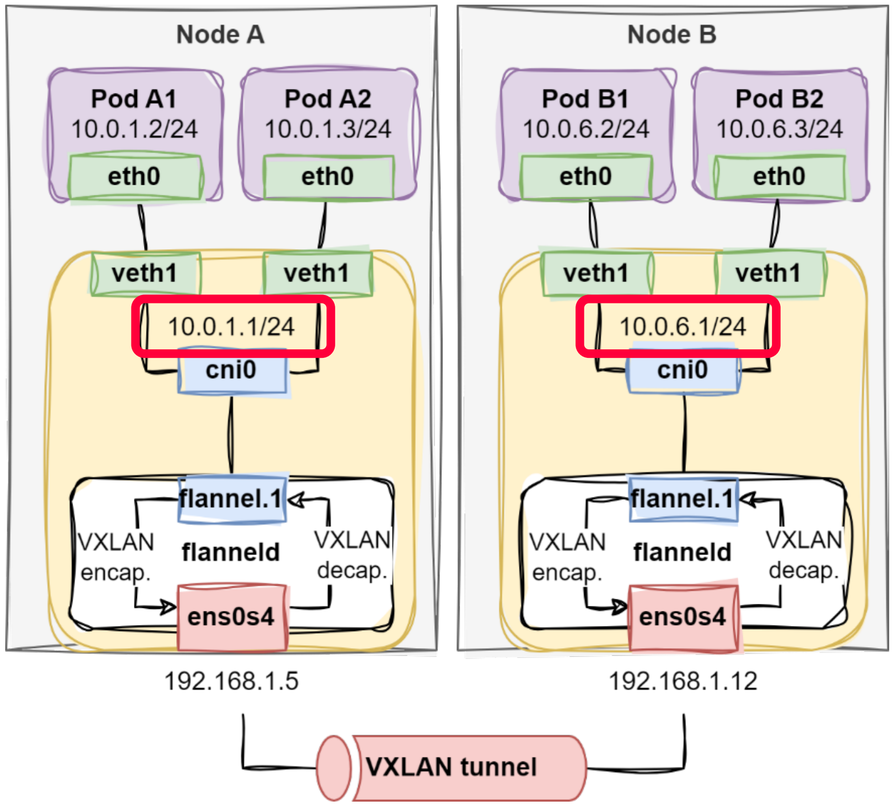
Implementation in Go

[GitHub - vishvananda/netlink: Simple netlink library for go.]

[GitHub - trevex/homelab]

```go
func ensureBridge(n *NetConf, brName string) (*netlink.Bridge, error) {
    err := netlink.LinkAdd(&netlink.Bridge{
        LinkAttrs: netlink.LinkAttrs{
            Name:   brName,
            MTU:    n.MTU,
            TxQLen: -1,
        },
    })

    // We ignore error if it the bridge already existed
    if err != nil && err != syscall.EEXIST {
        return nil, fmt.Errorf("could not add %q: %v", brName, err)
    }

    // Fetch the bridge
    br, err := bridgeByName(brName)
    if err != nil {
        return nil, err
    }

    // We want to own the routes for this interface
    _, err = sysctl.Sysctl(fmt.Sprintf("net/ipv6/conf/%s/accept_ra", brName), "0")
    if err != nil {
        return nil, err
    }

    // We disable Spanning Tree Protocol
    if err := os.WriteFile(fmt.Sprintf("/sys/class/net/%s/bridge/stp_state", brName), []byte("0"), 0644); err != nil {
        return nil, err
    }

    // And make sure it is up
    if err := netlink.LinkSetUp(br); err != nil {
        return nil, err
    }

    return br, nil
}
```

But what about routing!?

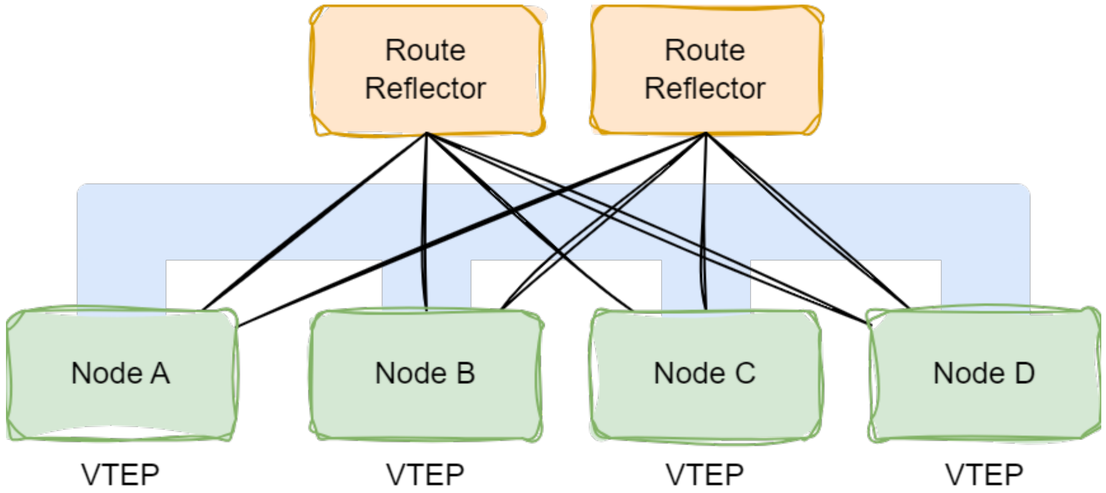


- We want to avoid allocating a range per node
  - VPC come in all sizes
  - Not every node will be active for a particular VPC
- How is overlay route management usually handled?

➔ **BGP EVPN**

(Border Gateway Protocol Ethernet Virtual Private Network)

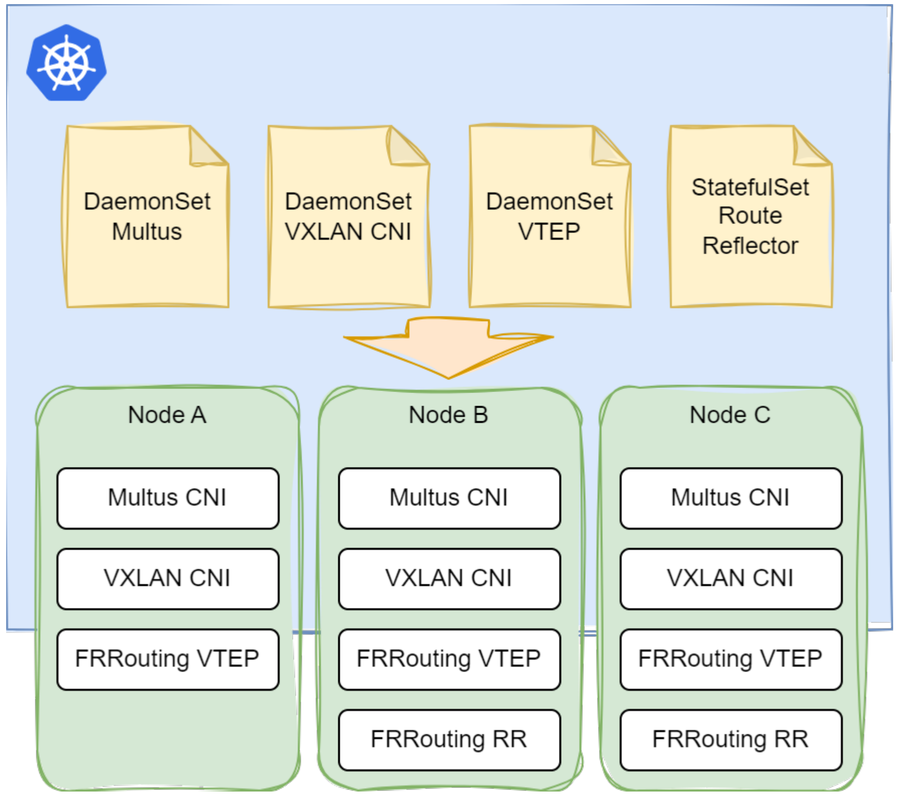
BGP EVPN



- All VXLAN Tunnel Endpoint (VTEP) routes are advertised to route reflectors (RR)
- Each VTEP receives necessary routes for local VNIs (VXLAN network identifier)
- Symmetric IRB (Integrated Bridging and Routing): MP-BGP can simultaneously advertise Layer 2 MAC addresses and Layer 3 routing information

➔ Automatic discovery of routes depending on local VNIs

Deployment



1. **Multus DaemonSet**: allow secondary CNI implementation
2. **Our CNI DaemonSet**: install CNI plugin
3. **VTEP DaemonSet**: Announce endpoints
4. **RR StatefulSet**: Distribute routing information

Kubernetes Logo: kubernetes/logo/LICENSE at master · kubernetes/kubernetes · GitHub
Diagram Source: BWI/Voss

Testing

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pod-a
spec:
  config: '{
  "type": "vxlan",
  "vni": 300,
  "addr": "172.16.32.1/24" # no ipam
}'
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-a
  annotations:
    k8s.v1.cni.cncf.io/networks: pod-a
spec:
  containers:
  - name: test
    image: centos/tools
    imagePullPolicy: IfNotPresent
    command:
    - /sbin/init
  nodeSelector:
    kubernetes.io/hostname: compute-1
```

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pod-b
spec:
  config: '{
  "type": "vxlan",
  "vni": 300,
  "addr": "172.16.32.2/24" # no ipam
}'
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-b
  annotations:
    k8s.v1.cni.cncf.io/networks: pod-b
spec:
  containers:
  - name: test
    image: centos/tools
    imagePullPolicy: IfNotPresent
    command:
    - /sbin/init
  nodeSelector:
    kubernetes.io/hostname: compute-2
```

Testing

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pod-a
spec:
  config: '{
  "type": "vxlan",
  "vni": 300,
  "addr": "172.16.32.1/24" # no ipam
}'
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-a
  annotations:
    k8s.v1.cni.cncf.io/networks: pod-a
spec:
  containers:
  - name: test
    image: centos/tools
    imagePullPolicy: IfNotPresent
    command:
    - /sbin/init
  nodeSelector:
    kubernetes.io/hostname: compute-1
```

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pod-b
spec:
  config: '{
  "type": "vxlan",
  "vni": 300,
  "addr": "172.16.32.2/24" # no ipam
}'
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-b
  annotations:
    k8s.v1.cni.cncf.io/networks: pod-b
spec:
  containers:
  - name: test
    image: centos/tools
    imagePullPolicy: IfNotPresent
    command:
    - /sbin/init
  nodeSelector:
    kubernetes.io/hostname: compute-2
```

```
k exec -it pod-a -- sh
ping ${IP_B}%net1
nc -l -p 3000
# on the other
k exec -it pod-b -- sh
nc ${IP_A}%net1 3000
```

What does the future hold?

Bildquelle: Bundeswehr/Torsten Kraatz

Current developments

Multi-Pool (Beta)
The Multi-Pool IPAM mode supports allocating PodCIDRs from multiple different IPAM pools, depending on properties of the workload defined by the user, e.g. annotations.

IP pools
IP pools are ranges of IP addresses from which Calico assigns pod IPs. […]
You can control which pool Calico uses for each pod using node selectors, or annotations on the pod or the pod's namespace.

OVN-Kubernetes (Open Virtual Networking - Kubernetes) is an open-source project that provides a robust networking solution for Kubernetes clusters with OVN (Open Virtual Networking) and Open vSwitch (Open Virtual Switch) at its core. It is a Kubernetes networking conformant plugin written according to the CNI (Container Network Interface) specifications.

Cilium Logo: Cilium brand guidelines
Calico Logo: Project Calico (tigera.io)
OVN Kubernetes Logo: GitHub - ovn-org/ovn-kubernetes: A robust Kubernetes networking platform

What is (Hyper-)Converged Infrastructure?

According to Wikipedia:

"Hyper-converged infrastructure (HCI) is a software-defined IT infrastructure that virtualizes all of the elements of conventional "hardware-defined" systems. HCI includes, at a minimum, virtualized computing (a hypervisor), software-defined storage, and virtualized networking (software-defined networking). HCI typically runs on commercial off-the-shelf (COTS) servers."

➔ Everything is software defined

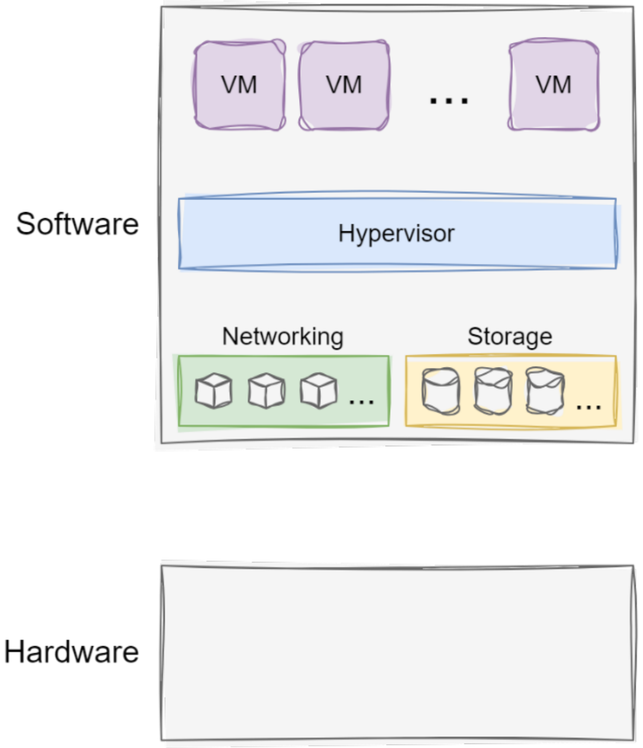


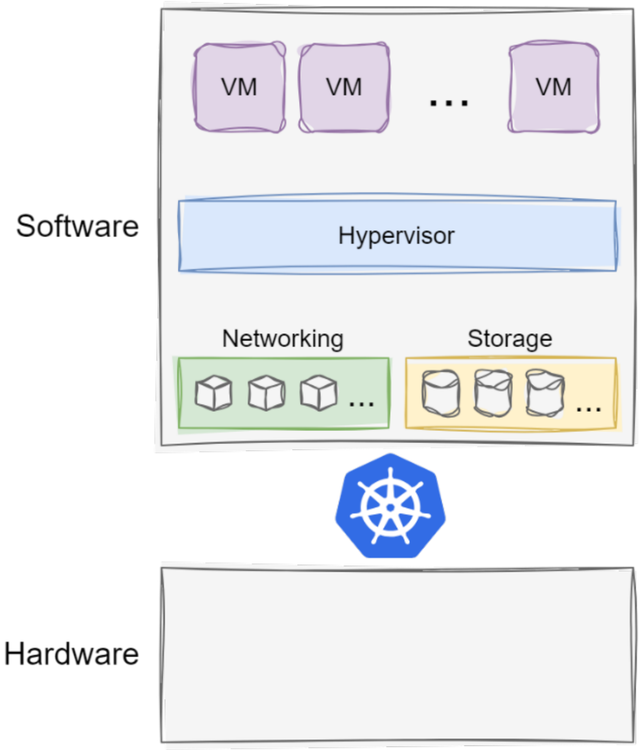
Diagram Source: BWI/Voss

What is (Hyper-)Converged Infrastructure?

According to Wikipedia:

"Hyper-converged infrastructure (HCI) is a software-defined IT infrastructure that virtualizes all of the elements of conventional "hardware-defined" systems. HCI includes, at a minimum, virtualized computing (a hypervisor), software-defined storage, and virtualized networking (software-defined networking). HCI typically runs on commercial off-the-shelf (COTS) servers."

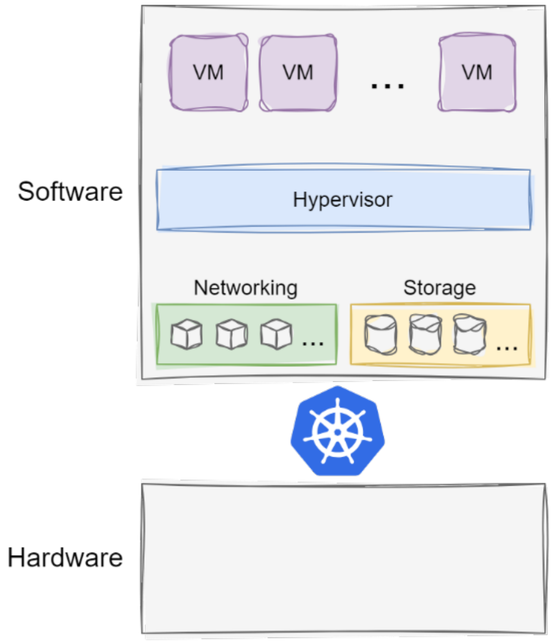
➔ Everything is software defined in Kubernetes



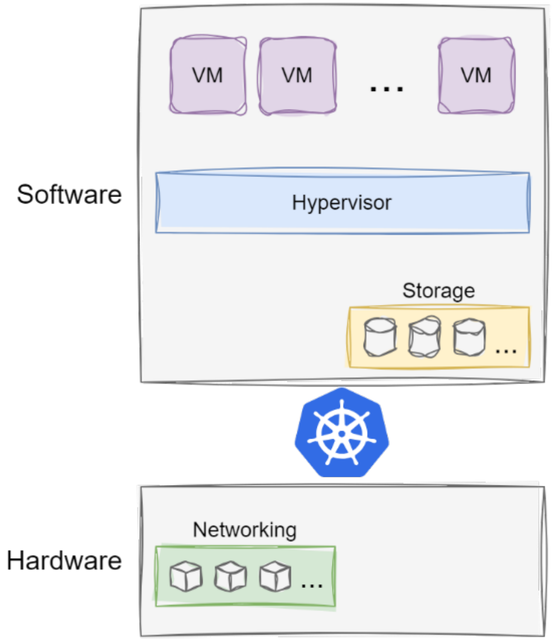
Kubernetes Logo: kubernetes/logo/LICENSE at master · kubernetes/kubernetes · GitHub
Diagram Source: BWI/Voss

Current trends

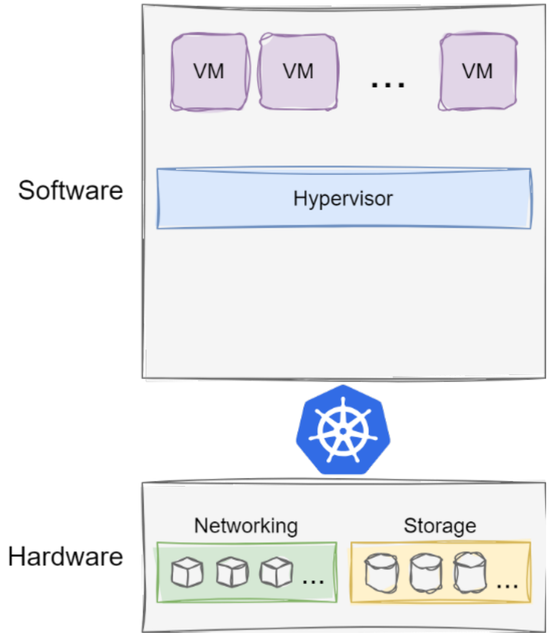
Converged 1.0



Converged 2.0



Converged 3.0



Kubernetes Logo: kubernetes/logo/LICENSE at master · kubernetes/kubernetes · GitHub
Diagram Source: BWI/Voss

Current trends

- Offloading to DPU/IPUs is becoming popular, particularly at scale
  - AWS Nitro System
  - Microsoft acquired Fungible (DPU vendor)
  - Google partnering with Intel to acquire E2100 IPUs
- Offloading support is gaining traction in the CNCF ecosystem
  - SR-IOV VMs in Kubevirt
  - Kube OVN DPDK Kubevirt fork
  - IronCore by SAP
  - Userspace CNI by Intel

➔ With more FOSS developments in this area everyone can benefit

Conclusion

- Converged infrastructure brings new requirements into the CNCF ecosystem
- Networking in Kubernetes is flexible
- The CNI spec is approachable
- Ideas how to leverage Kubernetes in new scenarios

Questions?

Code

Slides