Why Kubernetes?

A Deep Dive in Options, Benefits and Use Cases

Bettina Bassermann
Solution Sales Specialist DevOps
Bettina.Bassermann@suse.com

Rania Mohamed
Solution Architect, Services Consulting
Rania.Mohamed@suse.com

Who is SUSE?

- **Founded in 1992**
- **Largest independent open source vendor as of March 2019**
- **Technology company**
- **Our Mission is to help customers to master the digital transformation through Open Source technology**
- **Innovating with Partners and communites**
- **Enterprise-Grade Support**

Series about modern Application Development

- Software Development, Microservices & Container Management, a SUSE webinar series on modern Application Development
- Please find all SUSE Webinars here

https://www.suse.com/de-de/events/webinars

https://www.suse.com/c/author/rasadus/

Microservices – Is it the Holy Grain? A Perspective of a Developer

Container and Cloud Native Technologies – Why do we need them and what is so great about it?

Why Kubernetes? A Deep Dive in Options, Benefits and Usecasese

About making Choices – CaaSPv4 as SUSE's empowering of Kubernetes

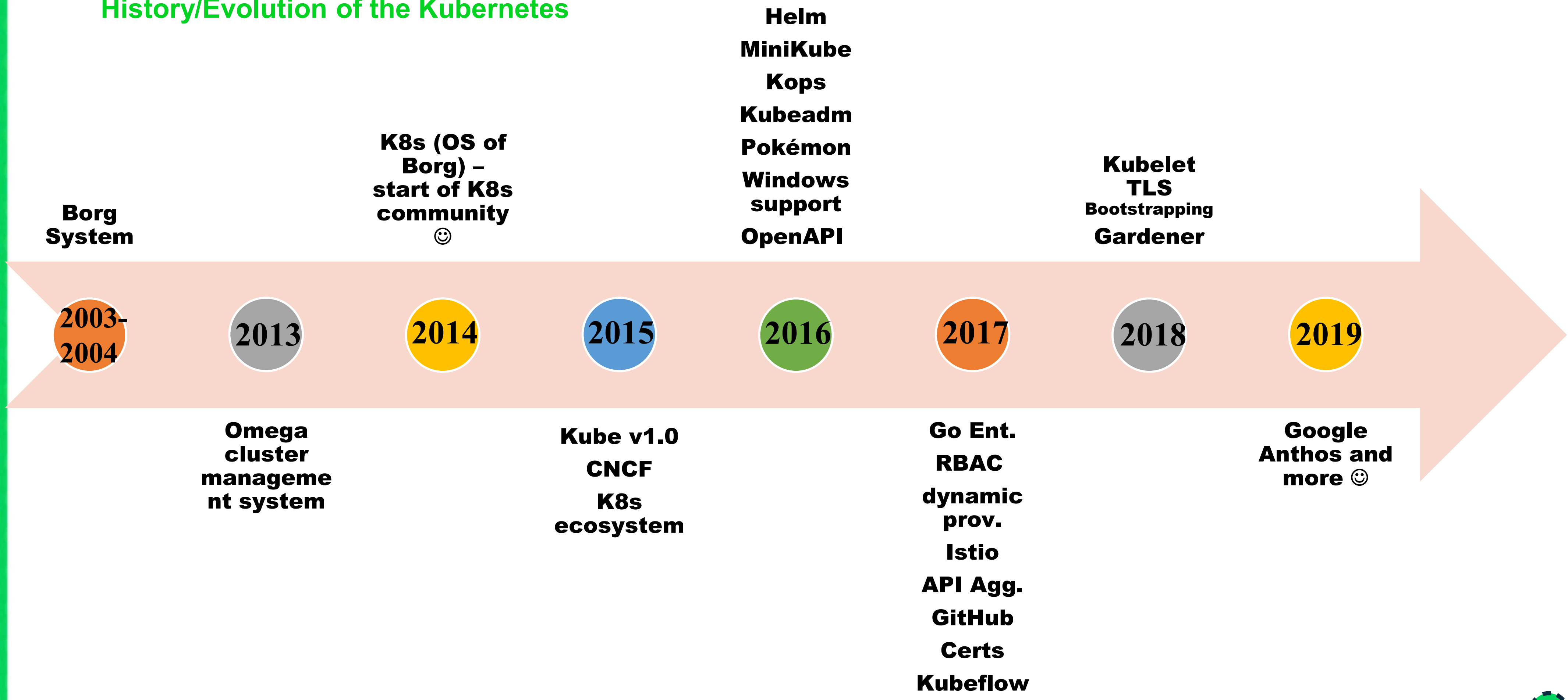
....stay tuned for the 2020 sessions with the Chamelion

Agenda

- Kubernetes evolution
- Kubernetes Architecture
- Kubernetes Components
- Deep Dive in Kubernetes Network component
- Deep Dive in Kubernetes container engine component
- Deep Dive in monitoring component
- Deep Dive in Kubernetes service Mesh Component
- Kubernetes Usecases

Basics about Kubernetes

History/Evolution of the Kubernetes



Kubernetes Architecture

Kubernetes Architecture – Main Soln Design Principles

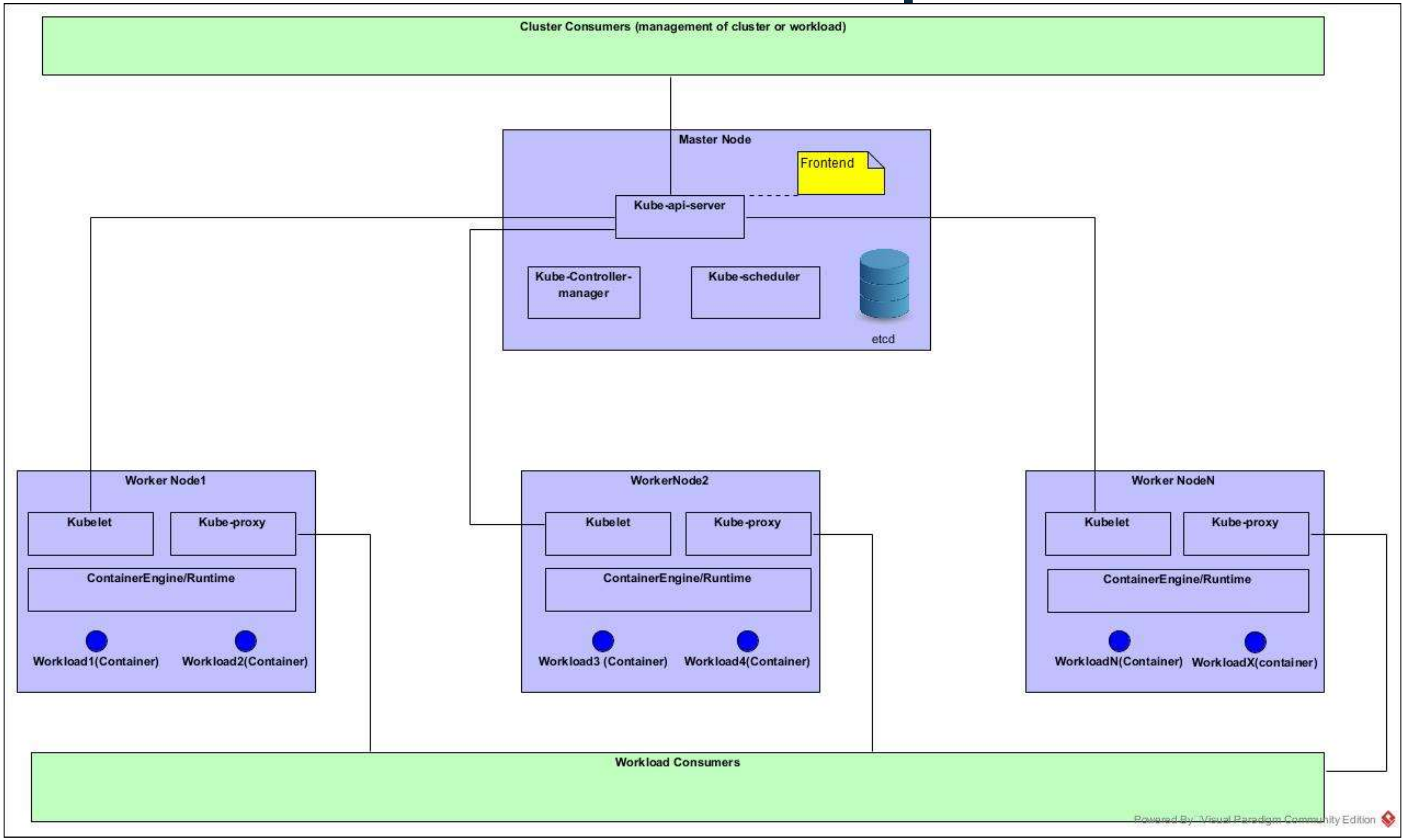
- **API Centric**
- **Separation** of **concerns** → divide and conquer ☺
- **Pluggability**
- Flexibility
- Well-defined **State management** → following MSA ☺
- **Extensibility**
- Scalability
- **Automation**
- **Simplicity**
- Standardization
- **Design** for **failure**
- **PaaS**
- Repeatability → **reconstructable** by **observation**.
- **Self-healing**
- Implements **Event** processing and even complex event processing
- **Graceful degradation**
- Target **autonomous**
- **Manage** dependencies
- **Transparency**
- Design API by **SLO** rather than implementation
- **High** Availability
- **Multi-tenancy**
- **Decentralizing** more distributed

Kubernetes Architecture – Main Target Requirements

- **Orchestrate & Manage** Containers
- Handle **Variable** load **efficiently**
- Enable **integration** with external world
- Enable **apps/workloads integration**
- Enable **business continuity**
- Support running **distributed** apps/workload and **MSA**
- Enable **hybrid**/**multi-cloud**
- Enable **CNA** development
- Support/offer **DevOps**
- Support of **pushing** workload **updates** with **no disruption**, ex blue-green
- Enable **Modernization**
- Enable **12factor** development BP
- Enable integration with **marketplace**
- Workloads **Log aggregation** and **analytics**
- Enable **ease** of **monitoring** distributed workloads and gathering data
- Enable **service discovery** for the running workloads
- Enabling **load balancing** between workload instances
- Enable **authentication** and **authorization** for the workload

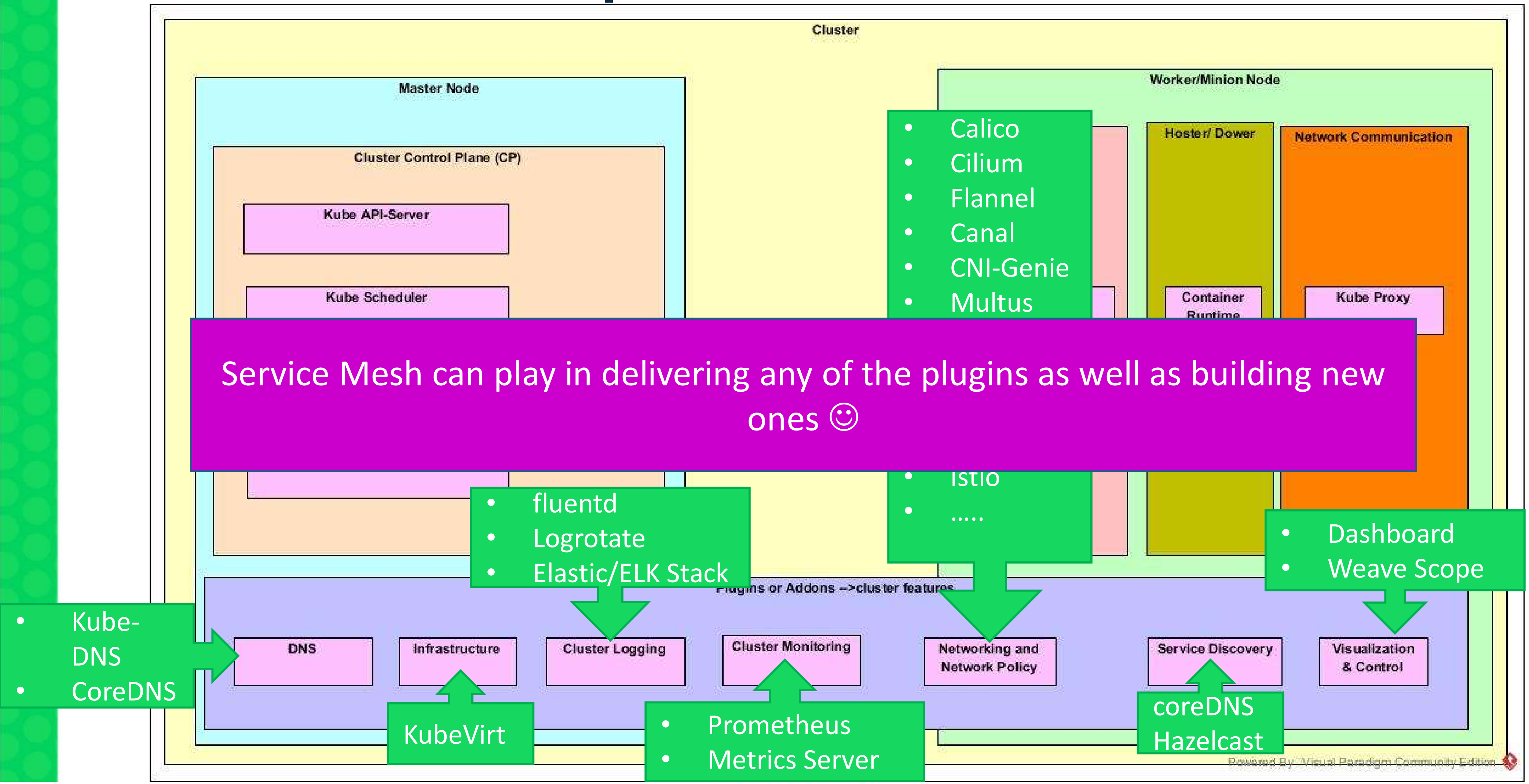
Build and facilitate enterprise environment for containers (same as an application server does for a Java app) ☺

Kubernetes Architecture - Blueprint



Kubernetes Components & Objects

Kubernetes Components



Kubernetes Objects – Main

- **Node** – a **machine** running in the cluster
- **Pod** – **smallest deployable** unit, may have 1+ containers
- **ReplicaSet** – defines **number** of **running** instances of a pod
- **Service** – defines **access** to one or more workload **API** (types: **ClusterIP**, **NodePort** & **LoadBalancer**) → sub-objects: svc, endpoint, iptables and IPVS
- **Volume** – **storage** of a container (supporting **stateful workloads**) → sub-objects: **PV** and **PVC** and **StorageClass**
- **Namespace** – resources **grouping**
- **Deployment** – declarative **packaging** of a set of pods and services
- **DaemonSet** – run a pod **instance** on **all nodes**
- **StatefulSet** – manages **Pods** that are based on **stateful** workloads.
- **Job** – creates one or more **Pods** / **service**.
- **Ingress** – manages **external access** to the services in a cluster

Deep Diving in Kubernetes world ☺

Network Component(s)

Network Component –Main Target Requirements

- **Pods** can communicate **without** using **NAT**
- **Nodes** can communicate with any **pod** **without** using **NAT**
- **Pod** internal **IP** is always seen by the same no matter the angle ☺
- MSA **Gateway** (controlling routing)
- **Securing** communication between workloads using policies
- Ability to use **different network layer** (2, 7, 3 and 4) isolation
- Ability to **mix** and **match** different **network policies**

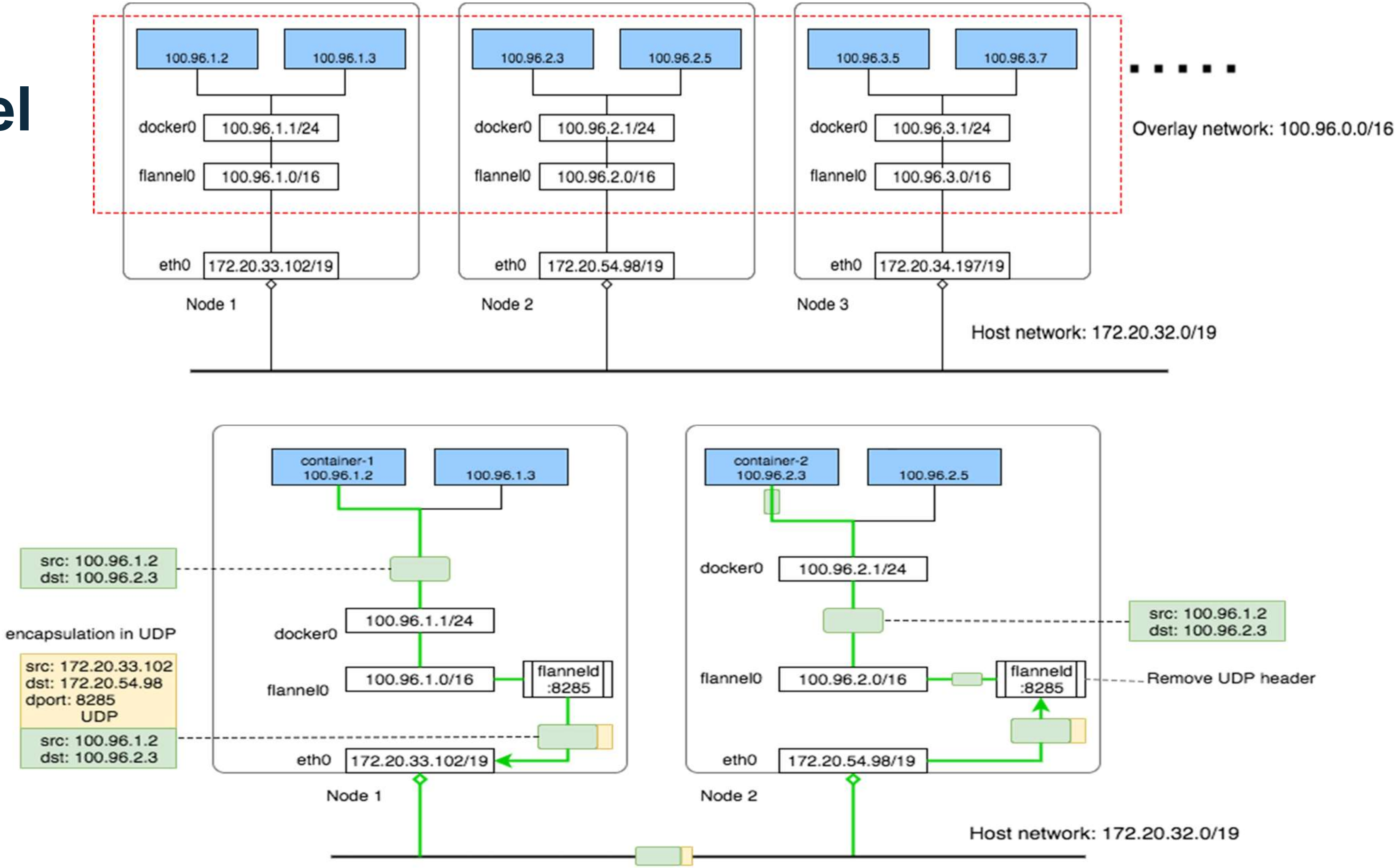
- **Standardize** network communication with container engine, kube api server and other nodes/pods → **CNI plugins**

K8s doesn't provide any default network implementation, It is a plugin☺
Aim to support secured multi-tenancy ☺

Network Component – a comparison

Feature	Flannel	Calico	Multus	Cilium	Weave Net
Brief features	- **Simple** - **Flat**/overlay network - Each pod has **unique** ip, all pods run on the same network - Runs a **daemon** process flannel to update the ip routing table - Mapping of the subnet to host info stored in etcd	- Connects pods using the same IP networking principles as the internet - **Interoperable** - **Flexible** - Enable security enforcement (**self workload firewall**) - True **Cloud native** scalability - leverage best practice cloud-native design patterns - Enables hybrid soln using **BGP**	- It enables attaching **multiple network** interfaces to the pods by creating homed pod, it is **magic** ☺ - It is a **meta-plugin** - It highly support **multitenancy**	- Supports **MSA** & **CNA** - It works by **network policies** - Supports **lightweight** protocols, such as HTTP, gRPC, Kafka - It is an **API-aware** network security **filtering**. - It uses Linux kernel technology called **BPF** - **Simple** - **Efficient** - Enables building **gateway policies** which can be enforced network-layer and **application-layer** security **policies** - **Scalability** - **Multi-tenancy** ☺ - L3 **Encryption** enforcement	- Creates a mesh **overlay** network **between** each of the nodes in the cluster - **Flexible** in the communication - **Simple** - Enables service discovery using **micro DNS** - **Encryption** - Supports **multi-cast** - Enables **portability**
Net. Layering	L3 network fabric	L3 & 7	N/A	L3 & 7	L3
Stability	Very high	Very high		High	High
Service meshing	Doesn't integrate and doesn't allow any network policy implementation	Integrates and enables defining rich network policy models	N/A	Integrates and enables defining rich network policy models	No
Gateways?	No	Yes	N/A	Yes	Yes
Perf.	Good	Very Good	N/A	Very Good	Very Good

Flannel

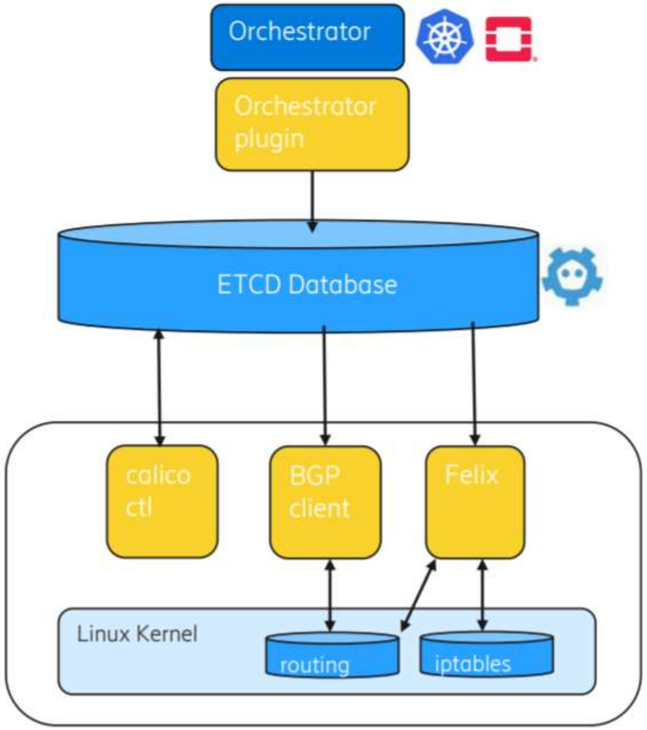


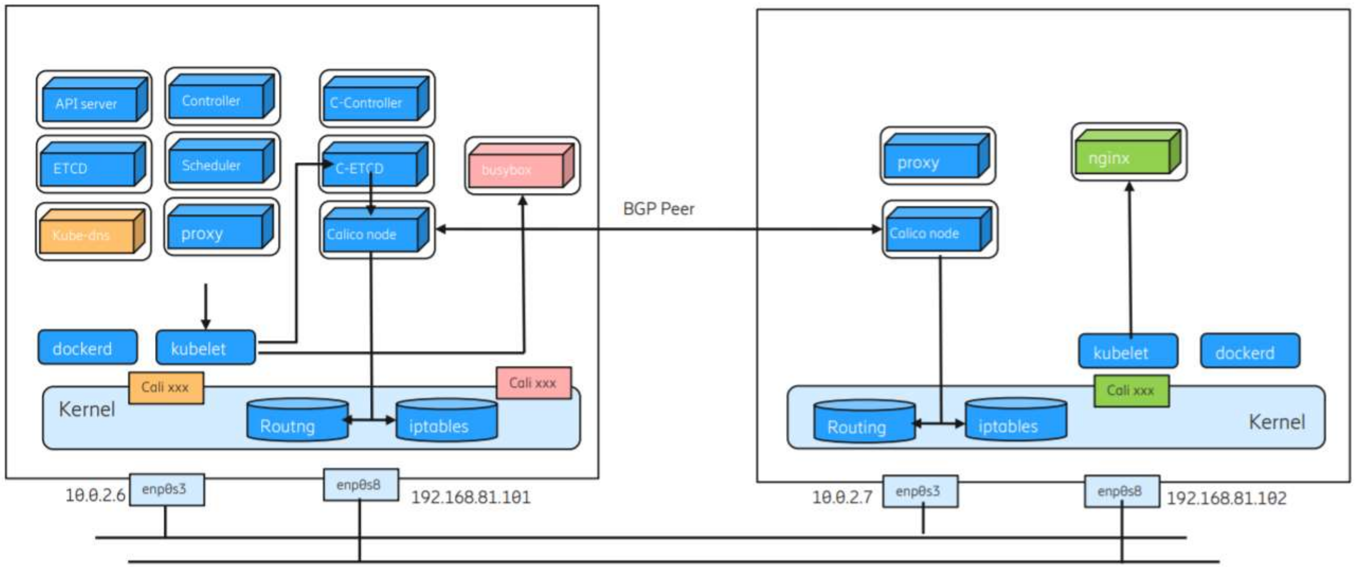
100.96.1.2
100.96.1.3
docker0 100.96.1.1/24
flannel0 100.96.1.0/16
eth0 172.20.33.102/19
Node 1
100.96.2.3
100.96.2.5
docker0 100.96.2.1/24
flannel0 100.96.2.0/16
eth0 172.20.54.98/19
Node 2
100.96.3.5
100.96.3.7
docker0 100.96.3.1/24
flannel0 100.96.3.0/16
eth0 172.20.34.197/19
Node 3
Overlay network: 100.96.0.0/16
Host network: 172.20.32.0/19
container-1 100.96.1.2
100.96.1.3
src: 100.96.1.2 dst: 100.96.2.3
encapsulation in UDP
src: 172.20.33.102 dst: 172.20.54.98 dport: 8285 UDP
src: 100.96.1.2 dst: 100.96.2.3
docker0 100.96.1.1/24
flannel0 100.96.1.0/16
flanneld :8285
eth0 172.20.33.102/19
Node 1
container-2 100.96.2.3
100.96.2.5
docker0 100.96.2.1/24
src: 100.96.1.2 dst: 100.96.2.3
flannel0 100.96.2.0/16
flanneld :8285
Remove UDP header
eth0 172.20.54.98/19
Node 2
Host network: 172.20.32.0/19

Network Component – a comparison

Feature	Flannel	Calico	Multus	Cilium	Weave Net
Brief features	- **Simple** - **Flat**/overlay network - Each pod has **unique** ip, all pods run on the same network - Runs a **daemon** process flannel to update the ip routing table - Mapping of the subnet to host info stored in etcd	- Connects pods using the same IP networking principles as the internet - **Interoperable** - **Flexible** - Enable security enforcement (**self workload firewall**) - True **Cloud native** scalability - leverage best practice cloud-native design patterns - Enables hybrid soln using **BGP**	- It enables attaching **multiple network** interfaces to the pods by creating homed pod, it is **magic** ☺ - It is a **meta-plugin** - It highly support **multitenancy**	- Supports **MSA** & **CNA** - It works by **network policies** - Supports **lightweight** protocols, such as HTTP, gRPC, Kafka - It is an **API-aware** network security **filtering**. - It uses Linux kernel technology called **BPF** - **Simple** - **Efficient** - Enables building **gateway policies** which can be enforced network-layer and **application-layer** security **policies** - **Scalability** - **Multi-tenancy** ☺ - L3 **Encryption** enforcement	- Creates a mesh **overlay** network **between** each of the nodes in the cluster - **Flexible** in the communication - **Simple** - Enables service discovery using **micro DNS** - **Encryption** - Supports **multi-cast** - Enables **portability**
Net. Layering	L3 network fabric	L3 & 7	N/A	L3 & 7	L3
Stability	Very high	Very high		High	High
Service meshing	Doesn't integrate and doesn't allow any network policy implementation	Integrates and enables defining rich network policy models	N/A	Integrates and enables defining rich network policy models	No
Gateways?	No	Yes	N/A	Yes	Yes
Perf.	Good	Very Good	N/A	Very Good	Very Good

Calico





default via 10.0.2.1 dev enp0s3
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.6
192.168.81.0/24 dev enp0s8 proto kernel scope link src 192.168.81.101
blackhole 192.200.59.192/26 proto bird
192.200.59.193 dev calidf072d3c423 scope link
192.200.59.198 dev cali0aa3720a2c7 scope link
192.200.203.0/26 via 192.168.81.102 dev tunl0 proto bird onlink

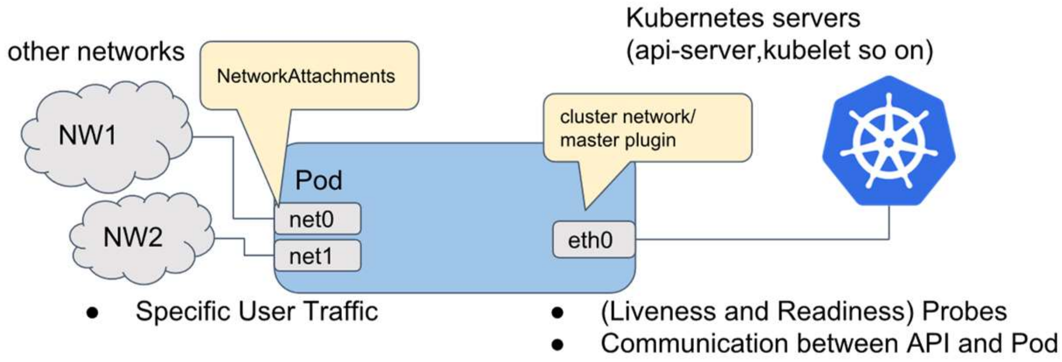
default via 10.0.2.1 dev enp0s3
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.7
192.168.81.0/24 dev enp0s8 proto kernel scope link src 192.168.81.102
192.200.59.192/26 via 192.168.81.101 dev tunl0 proto bird onlink
blackhole 192.200.203.0/26 proto bird
192.200.203.4 dev cali7bb4560a7c2 scope link

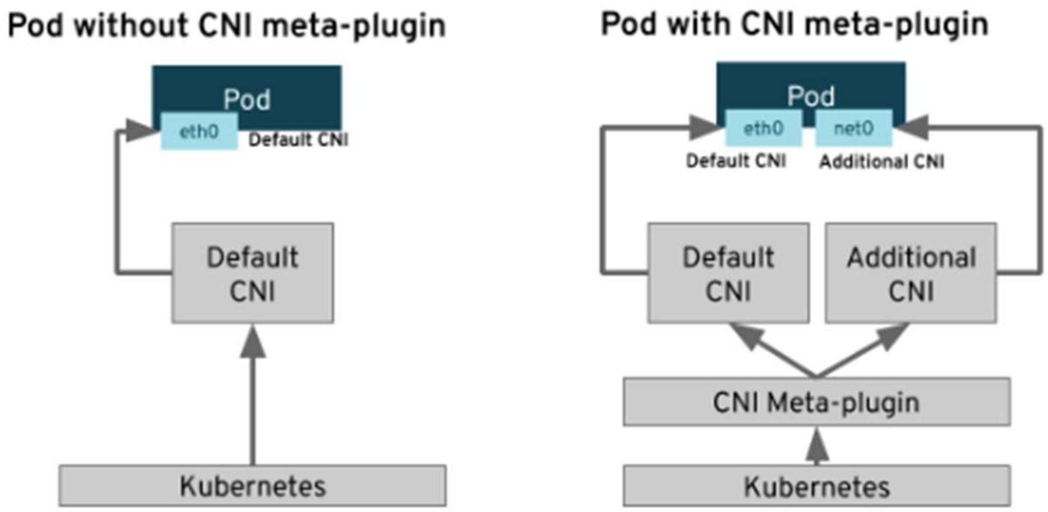
Network Component – a comparison

Feature	Flannel	Calico	Multus	Cilium	Weave Net
Brief features	- **Simple** - **Flat**/overlay network - Each pod has **unique** ip, all pods run on the same network - Runs a **daemon** process flannel to update the ip routing table - Mapping of the subnet to host info stored in etcd	- Connects pods using the same IP networking principles as the internet - **Interoperable** - **Flexible** - Enable security enforcement (**self workload firewall**) - True **Cloud native** scalability - leverage best practice cloud-native design patterns - Enables hybrid soln using **BGP**	- It enables attaching **multiple network** interfaces to the pods by creating homed pod, it is **magic** ☺ - It is a **meta-plugin** - It highly support **multitenancy**	- Supports **MSA** & **CNA** - It works by **network policies** - Supports **lightweight** protocols, such as HTTP, gRPC, Kafka - It is an **API-aware** network security **filtering**. - It uses Linux kernel technology called **BPF** - **Simple** - **Efficient** - Enables building **gateway policies** which can be enforced network-layer and **application-layer** security **policies** - **Scalability** - **Multi-tenancy** ☺ - L3 **Encryption** enforcement	- Creates a mesh **overlay** network **between** each of the nodes in the cluster - **Flexible** in the communication - **Simple** - Enables service discovery using **micro DNS** - **Encryption** - Supports **multi-cast** - Enables **portability**
Net. Layering	L3 network fabric	L3 & 7	N/A	L3 & 7	L3
Stability	Very high	Very high		High	High
Service meshing	Doesn't integrate and doesn't allow any network policy implementation	Integrates and enables defining rich network policy models	N/A	Integrates and enables defining rich network policy models	No
Gateways?	No	Yes	N/A	Yes	Yes
Perf.	Good	Very Good	N/A	Very Good	Very Good

Multus

```
+----------------+
|                |
|    Kubelet     |
|                |
+-------+--------+
        | calls
 +------v------+              +------------------+
 |             |              |                  |
 |     CNI     +------------->|      Multus      |
 |             |              |                  |
 +-------------+              +--------+---------+
                                       |  calls
                                +------v------+
                                |             |
                                |     CNI     |
                                |             |
                                +------+------+
                                       |
                                +------v-------+
                                |              |
                                |   Flannel    |
                                |              |
                                +--------------+
```

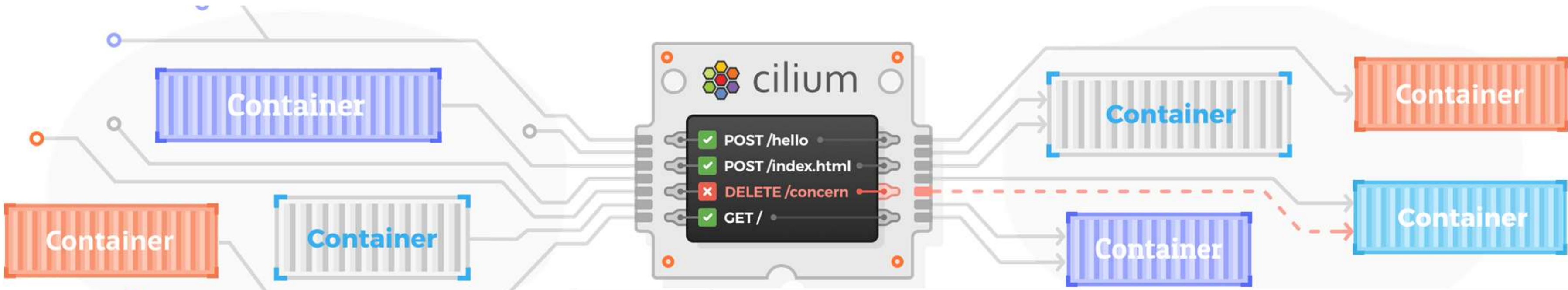


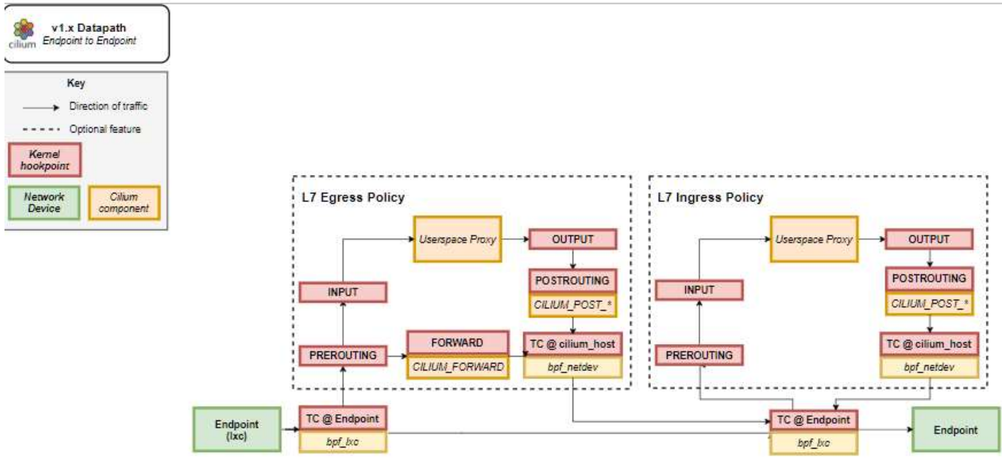


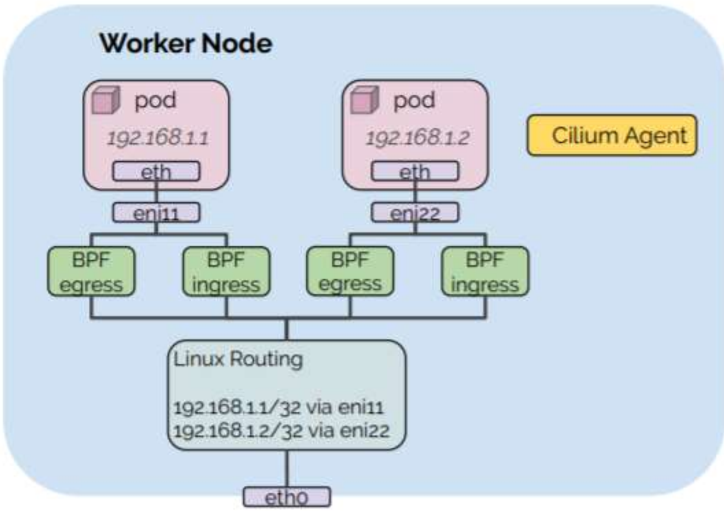
Network Component – a comparison

Feature	Flannel	Calico	Multus	Cilium	Weave Net
Brief features	- **Simple** - **Flat**/overlay network - Each pod has **unique** ip, all pods run on the same network - Runs a **daemon** process flannel to update the ip routing table - Mapping of the subnet to host info stored in etcd	- Connects pods using the same IP networking principles as the internet - **Interoperable** - **Flexible** - Enable security enforcement (**self workload firewall**) - True **Cloud native** scalability - leverage best practice cloud-native design patterns - Enables hybrid soln using **BGP**	- It enables attaching **multiple network** interfaces to the pods by creating homed pod, it is **magic** ☺ - It is a **meta-plugin** - It highly support **multitenancy**	- Supports **MSA** & **CNA** - It works by **network policies** - Supports **lightweight** protocols, such as HTTP, gRPC, Kafka - It is an **API-aware** network security **filtering**. - It uses Linux kernel technology called **BPF** - **Simple** - **Efficient** - Enables building **gateway policies** which can be enforced network-layer and **application-layer** security **policies** - **Scalability** - **Multi-tenancy** ☺ - L3 **Encryption** enforcement	- Creates a mesh **overlay** network **between** each of the nodes in the cluster - **Flexible** in the communication - **Simple** - Enables service discovery using **micro DNS** - **Encryption** - Supports **multi-cast** - Enables **portability**
Net. Layering	L3 network fabric	L3 & 7	N/A	L3 & 7	L3
Stability	Very high	Very high		High	High
Service meshing	Doesn't integrate and doesn't allow any network policy implementation	Integrates and enables defining rich network policy models	N/A	Integrates and enables defining rich network policy models	No
Gateways?	No	Yes	N/A	Yes	Yes
Perf.	Good	Very Good	N/A	Very Good	Very Good

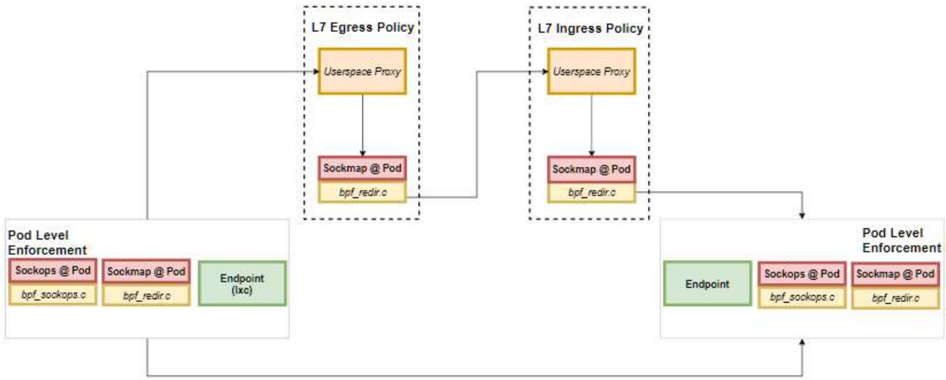
Cilium







Cilium
- Load-balancing
- Network policy
- Encryption
- Multi-cluster
- Visibility



Network Component – a comparison

Feature	Flannel	Calico	Multus	Cilium	Weave Net
Brief features	- **Simple** - **Flat**/overlay network - Each pod has **unique** ip, all pods run on the same network - Runs a **daemon** process flannel to update the ip routing table - Mapping of the subnet to host info stored in etcd	- Connects pods using the same IP networking principles as the internet - **Interoperable** - **Flexible** - Enable security enforcement (**self workload firewall**) - True **Cloud native** scalability - leverage best practice cloud-native design patterns - Enables hybrid soln using **BGP**	- It enables attaching **multiple network** interfaces to the pods by creating homed pod, it is **magic** ☺ - It is a **meta-plugin** - It highly support **multitenancy**	- Supports **MSA** & **CNA** - It works by **network policies** - Supports **lightweight** protocols, such as HTTP, gRPC, Kafka - It is an **API-aware** network security **filtering**. - It uses Linux kernel technology called **BPF** - **Simple** - **Efficient** - Enables building **gateway policies** which can be enforced network-layer and **application-layer** security **policies** - **Scalability** - **Multi-tenancy** ☺ - L3 **Encryption** enforcement	- Creates a mesh **overlay** network **between** each of the nodes in the cluster - **Flexible** in the communication - **Simple** - Enables service discovery using **micro DNS** - **Encryption** - Supports **multi-cast** - Enables **portability**
Net. Layering	L3 network fabric	L3 & 7	N/A	L3 & 7	L3
Stability	Very high	Very high		High	High
Service meshing	Doesn't integrate and doesn't allow any network policy implementation	Integrates and enables defining rich network policy models	N/A	Integrates and enables defining rich network policy models	No
Gateways?	No	Yes	N/A	Yes	Yes
Perf.	Good	Very Good	N/A	Very Good	Very Good

Container Component(s)

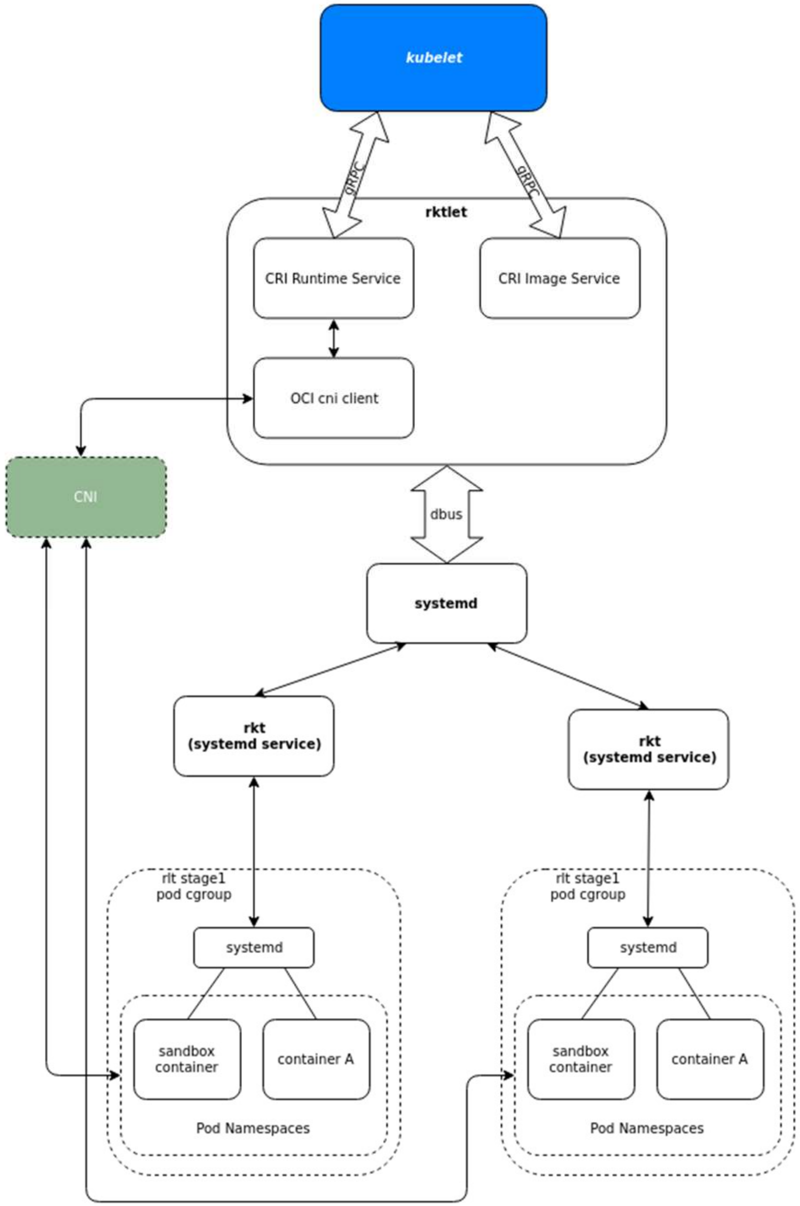
Container Runtime Component – Main Target Requirements

- Standardize the communication with Container engine, Container Runtime Interface CRI.
- Manages the namespace isolation and resource allocation at the OS levels using Linus cgroup and namespaces
- Creates & build a container using an Image
- It the runtime the container run above
- Abstract the container from the hosting OS
- Integrates with image registry
- Supports High level and low level container runtime
- Manages containers' lifecycle
- Support both running stateful (storage is a must here) and stateless containers
- Support logging and troubleshooting of a running container

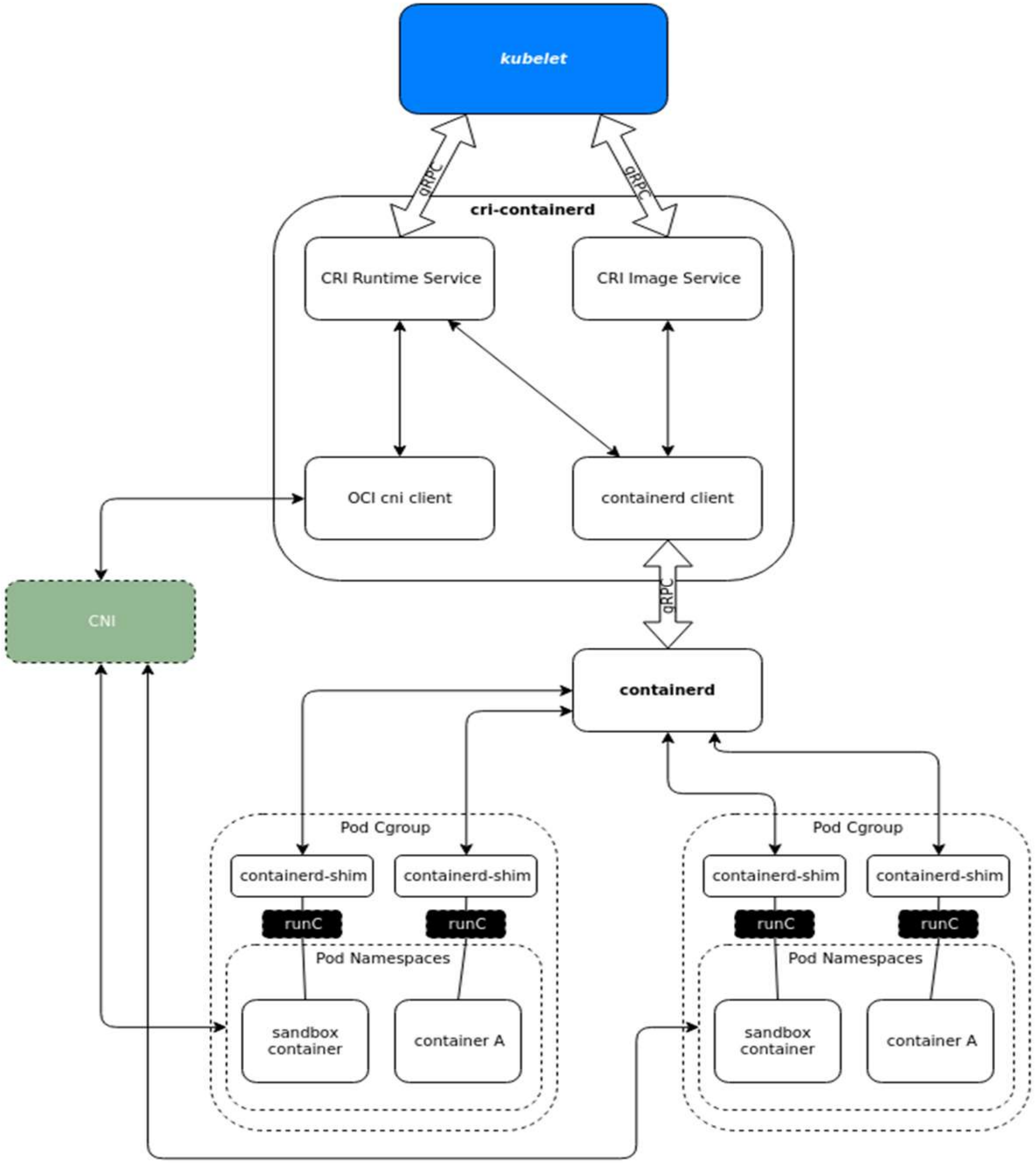
Container Runtime Component – a comparison

Feature	RunC	Rkt	CRI-O	Docker	Containerd
Brief features	- **Low-level** container runtime - Implements **OCI** - Requires **expertise** of the underlying host OS and configuration. - Does **not verify** container images or prepare the FS - **No centralized daemon**	- **High level** container engine - Built as an **alternative** to docker in K8s - Can group containers/apps in a shared context (**pod**) - **No centralized** init **daemon** - Support different container/pod **configurations** (like isolation parameters) - Better **isolation** Each pod runs in a different process - Supports **OCI**	- **High level** container engine - support **OCI** and implements **CRI** - It uses **runC** by default - Can plug any **OCI runtime** - **Light weight** (lots of small components, with defined roles & collaborating flows) - **Decentralized architecture** - **Secured** by as CRI-O containers are children of the process that spawned it - Fully compatible with **K8s Roadmap** and community - Implements **CNI** which make it more standard from a network setup - **Fast** - Can run Docker images	- **Not Standard** - **Heavyweight**/fat daemon - Central architecture - Has **security constraints** - Has no **limitation** ☺	- Runs as a **daemon** - Implements **CRI**
Security	Yes	Yes	Yes	No	No
Perf.	Good	Very Good	Very Good	Very Good	Very Good
Standard	Yes	Yes	Yes	No	Yes
Stability	Very high	High	Very High	Very high	High

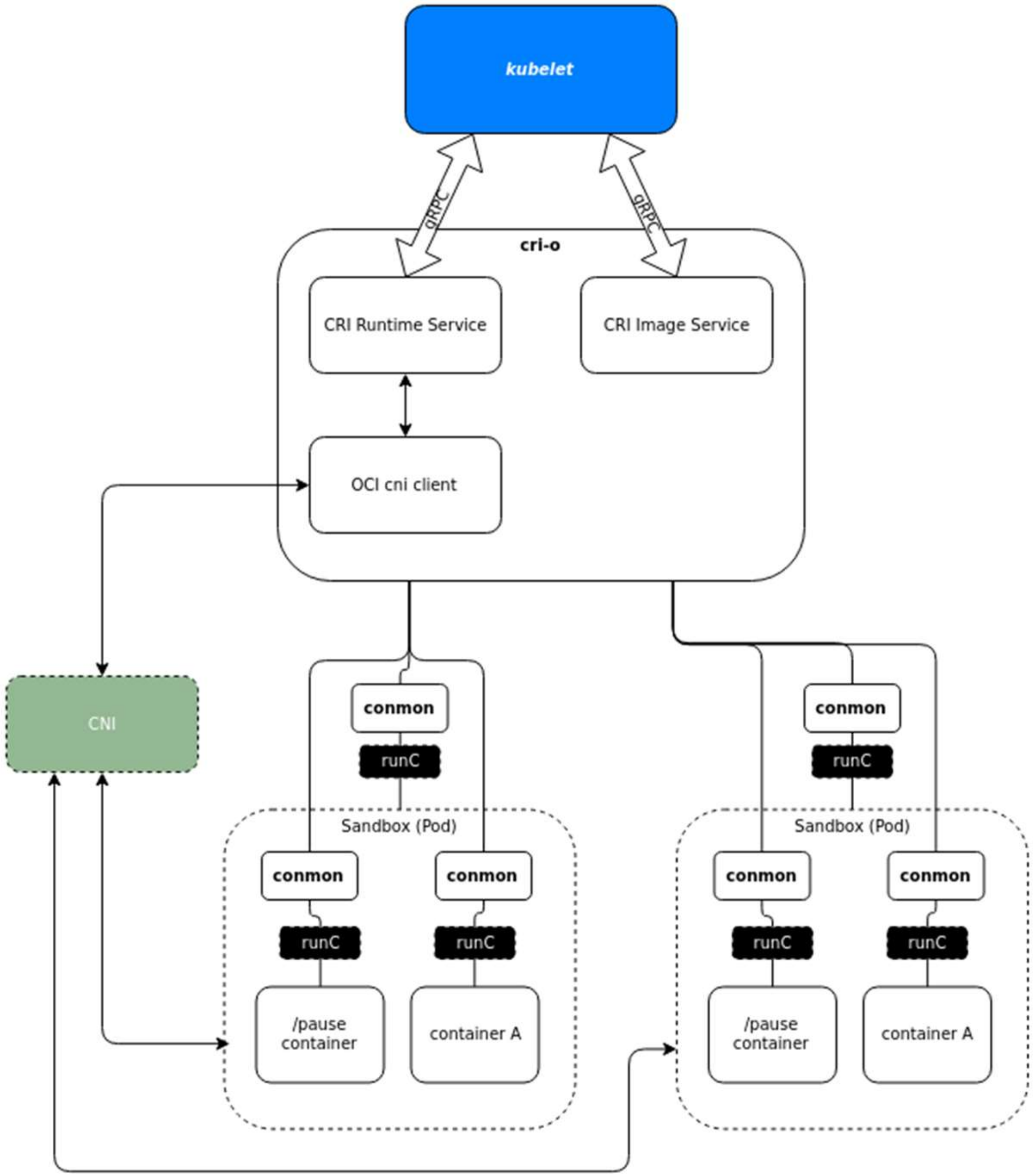
rkt


kubelet
gRPC
gRPC
rktlet
CRI Runtime Service
CRI Image Service
OCI cni client
CNI
dbus
systemd
rkt
(systemd service)
rkt
(systemd service)
rlt stage1
pod cgroup
systemd
sandbox
container
container A
Pod Namespaces
rlt stage1
pod cgroup
systemd
sandbox
container
container A
Pod Namespaces

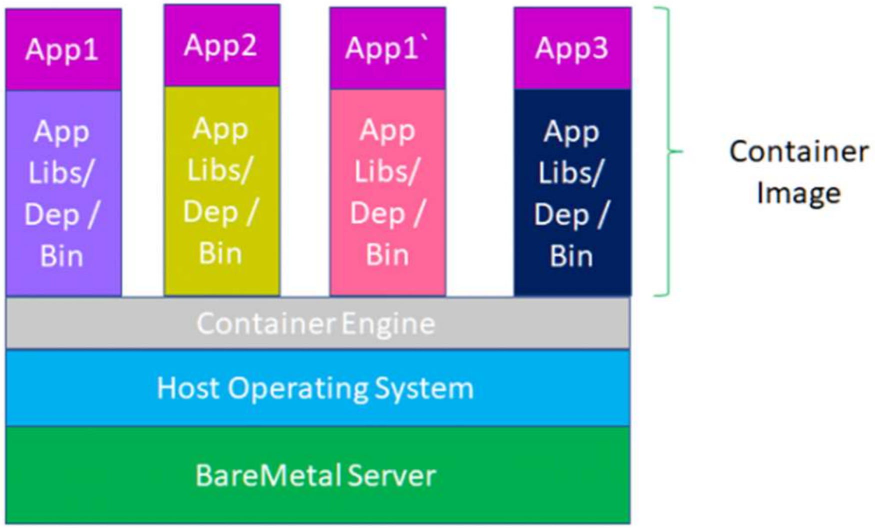
containerd


kubelet
gRPC
gRPC
cri-containerd
CRI Runtime Service
CRI Image Service
OCI cni client
containerd client
gRPC
CNI
containerd
Pod Cgroup
containerd-shim
containerd-shim
runC
runC
Pod Namespaces
sandbox container
container A
Pod Cgroup
containerd-shim
containerd-shim
runC
runC
Pod Namespaces
sandbox container
container A

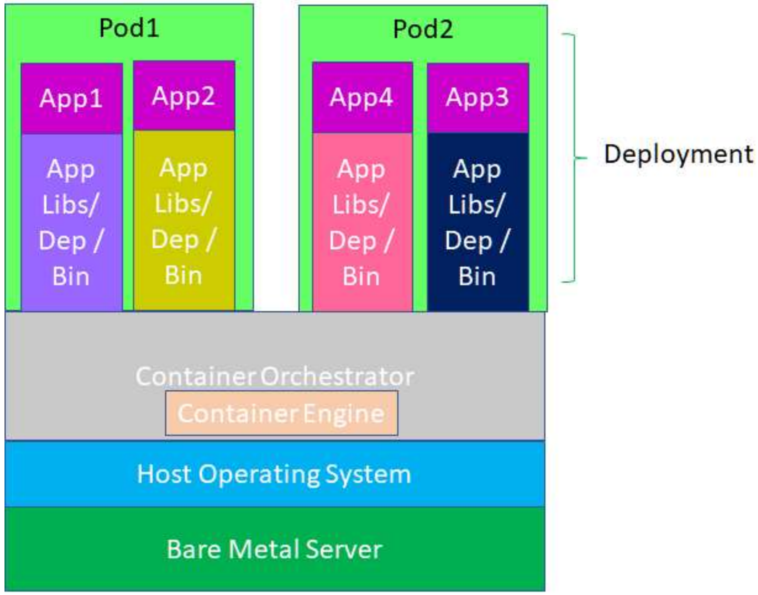
CRIO


kubelet
gRPC
gRPC
cri-o
CRI Runtime Service
CRI Image Service
OCI cni client
CNI
conmon
runC
conmon
runC
Sandbox (Pod)
Sandbox (Pod)
conmon
conmon
conmon
conmon
runC
runC
runC
runC
/pause container
container A
/pause container
container A

Container engine vs Containers Orchestrator



Containers



Container as a Service

Monitoring Component(s)

Monitoring Component – Main Target Requirements

- Monitor both cluster metrics as well as app/workload metrics
- Monitor the health of the cluster
- Monitor resources consumptions/utilization (node & pods)
- Availability (node and K8s objects)
- Gather k8s, apps and container metrics

Monitoring Component – solutions

Heapster, InfluxDB, & Grafana	Prometheus & Grafana	Heapster & ELK Stack	Datadog	Dynatrace
- Heapster, is a uniform platform which push monitoring metrics to a external tool to process - InfluxDB is used to store the collected metrics - Grafana is used to visualize the collected info - Simple	- Prometheus is a platform to gather metrics - Grafana is used to visualize the collected info - Simple - Flexible	- Heapster, is a uniform platform which push monitoring metrics to a external tool to process - Use ELK or Ealstic stack which includes Elasticsearch, Logstash, and Kibana, which define the data pipeline. - Central logging and dashboard and can hold some sort of analytics on the gathered data. - Powerful in analytics - Flexible	- Simple - Flexible data pipeline - Uses DaemonSet agent	- Uses DaemonSet agent - Not flexible in the gather metrics - complex

Service Mesh / Service Collaboration Component(s)

Service Mesh / Service Collaboration Component

What is Service Mesh?

Service Mesh / Service Collaboration Component – main challenges be4 service mesh

- MSA is small and fabulous but hard to control, watch and govern
- CNA is awesome but hard to troubleshoot
- MSA and CNA is about hybrid development and agility targeting time to market so how to balance that keeping the aspects of governance & quality→ hard balance ☺
- Security
- Monitoring
- Managing dependencies

Service Mesh / Service Collaboration Component – target requirements

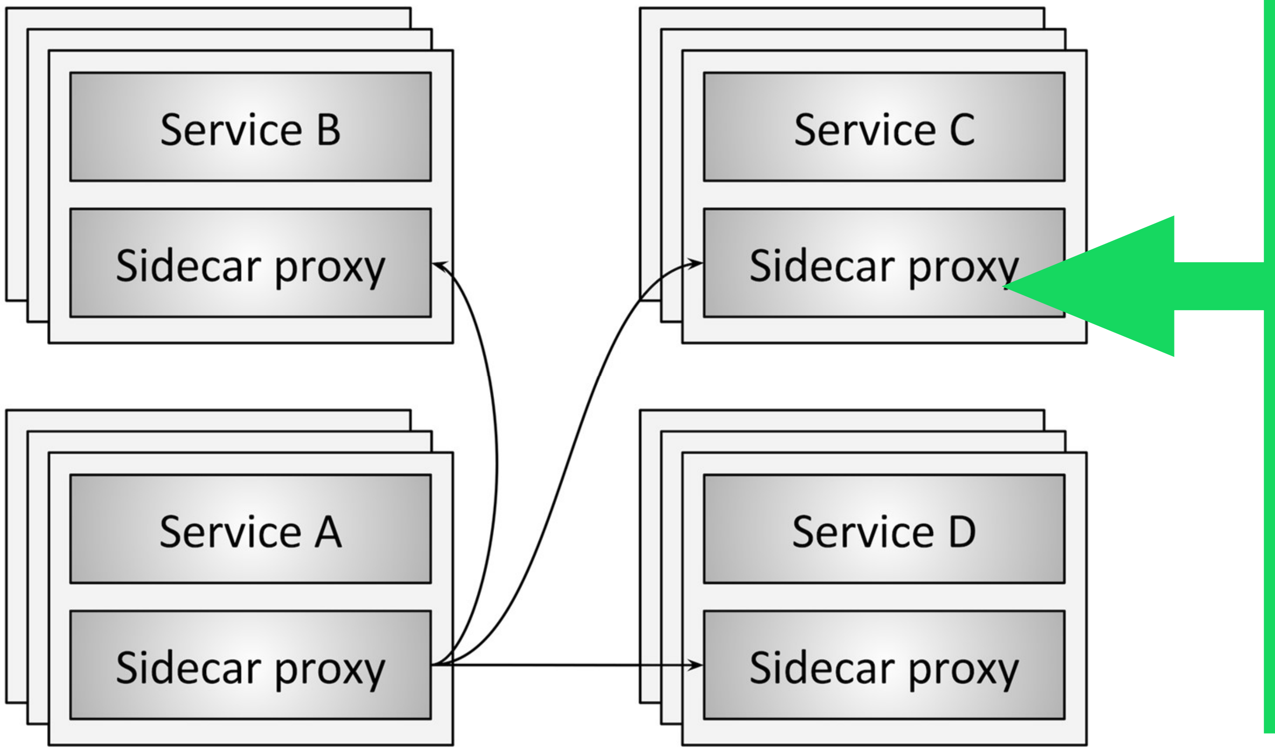
- Services or app must be **self**:
    - **Governed** – follow all policies and notify/get notified for changes.
    - **Secured** – not only the 2 authes but ability to defend itself
    - **Monitored** – metrics gathering
    - **Logging** – metrics gathering
    - **Gateway** (service/app discovery)
- Not compromising to the freedom of the developer ☺
- Must be **FAST** and **lightweight**
- **Observer** and **enforcer** and not just an **reactor**
- Supports well known protocols as HTTP2, gRPC …

It is just like Aspect Oriented Programming, AOP and IoC, Injection and Inversion of Controller ☺

Service Mesh / Service Collaboration Component – available Service Mesh FWs

- **Istio**, super powerful implementation ☺ → it uses envoy as a data plane, think of it as an extension.
- **Google Service Mesh**, it uses Istio but with more visualization capabilities and troubleshooting
- **AWS app Mesh**, it uses envoy it only provides parts of the requirements as the proxying and the monitoring metrics
- **Envoy**, it only focus on the data more than the control (i.e. what to do with the data)
- **Azure Service Mesh**, it is fully built by Microsoft, it covers most of the requirements

Service Mesh / Service Collaboration Component – More into Service Mesh – Istio &



<u>Data Plane (mainly gathering data transforming or converting it to events and forward it):</u>

1. The 2 authes
2. Service discovery
3. Monitoring metrics and Health checking
4. Load balancing and Routing
5. Observation to any metrics or events
6. Enables lots of MSA patterns such as Circuit breakers
7. K8s and CNA/MSA deployment strategies

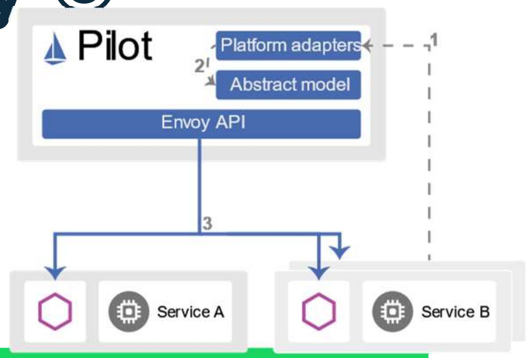
Service mesh works on the network packet level so it is an INF layer still

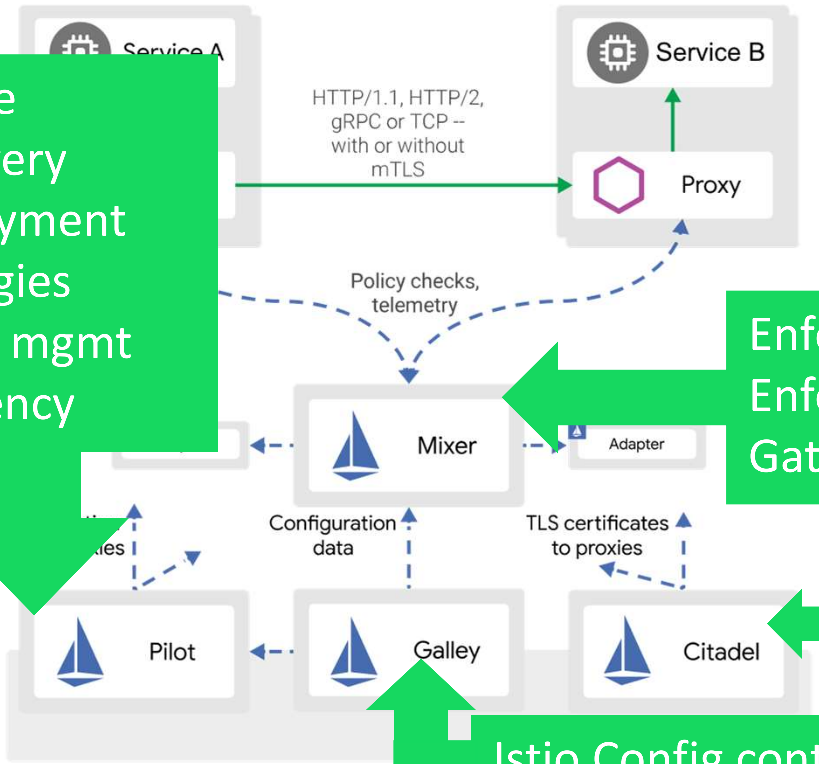
Sidecars are always stateless ☺

Service Mesh / Service Collaboration Component – More into Service Mesh – Istio & Envoy ☺

Is it Enough?

Service Mesh / Service Collaboration Component – More into Service Mesh – Istio & Envoy ☺





service discovery
Deployment strategies
Traffic mgmt
Resiliency

Enforces access control
Enforces policies (app level)
Gather Telemetric metrics for consumption calc.

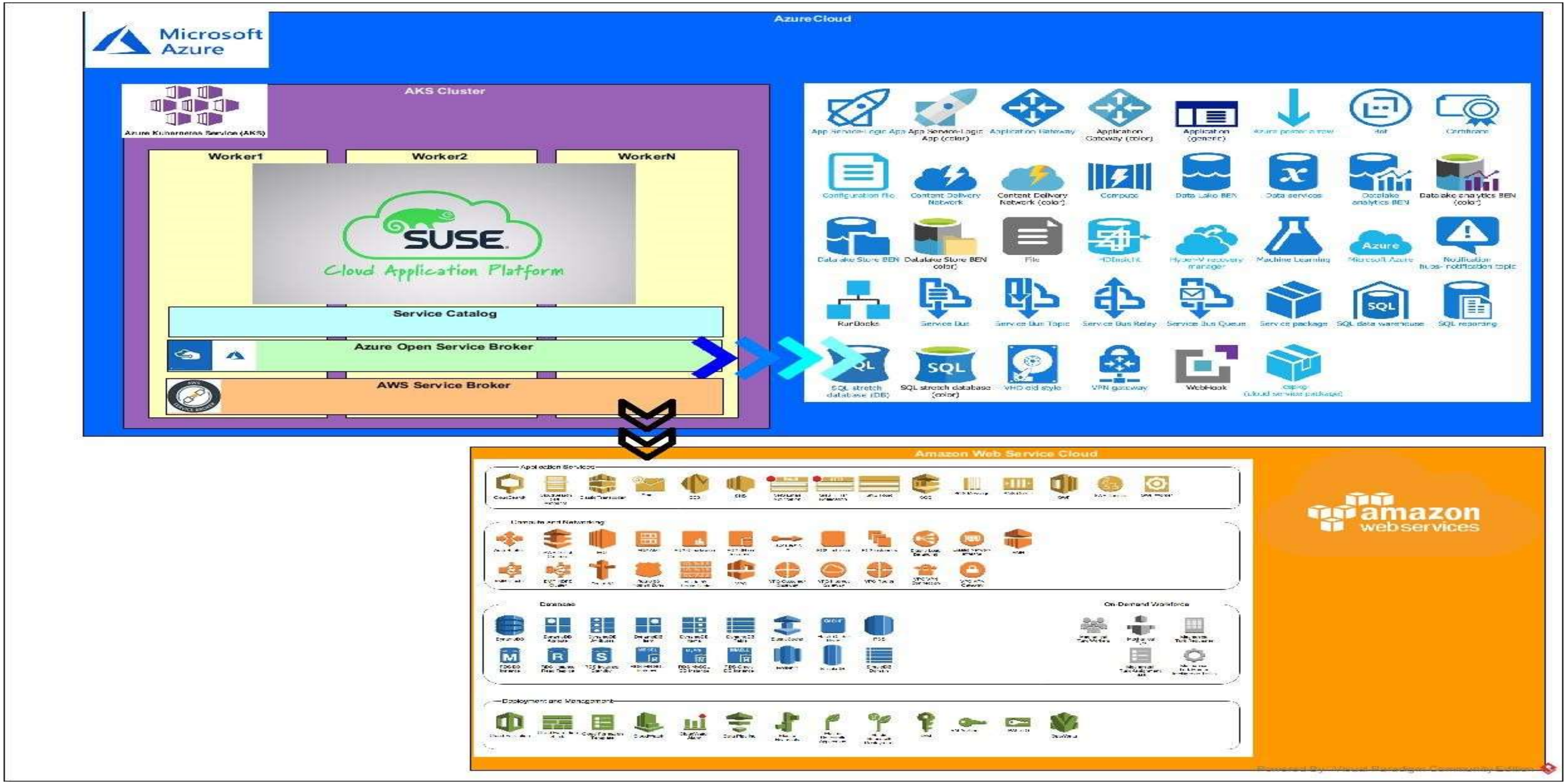
Service and end user authentication, it can be enforce identity based encryption

Istio Config controller (validator, processor, distributor…)

Data Plane is Magic as it provides network abstraction and still works on fact but it is not enough it still require and a fairy and their magical wands ☺

Kubernetes UseCase ☺

Multi/Hybrid – Cloud



https://www.suse.com/c/cloud-native-applications-in-azure-supporting-hybrid-cloud/

Business Benefits Multicloud

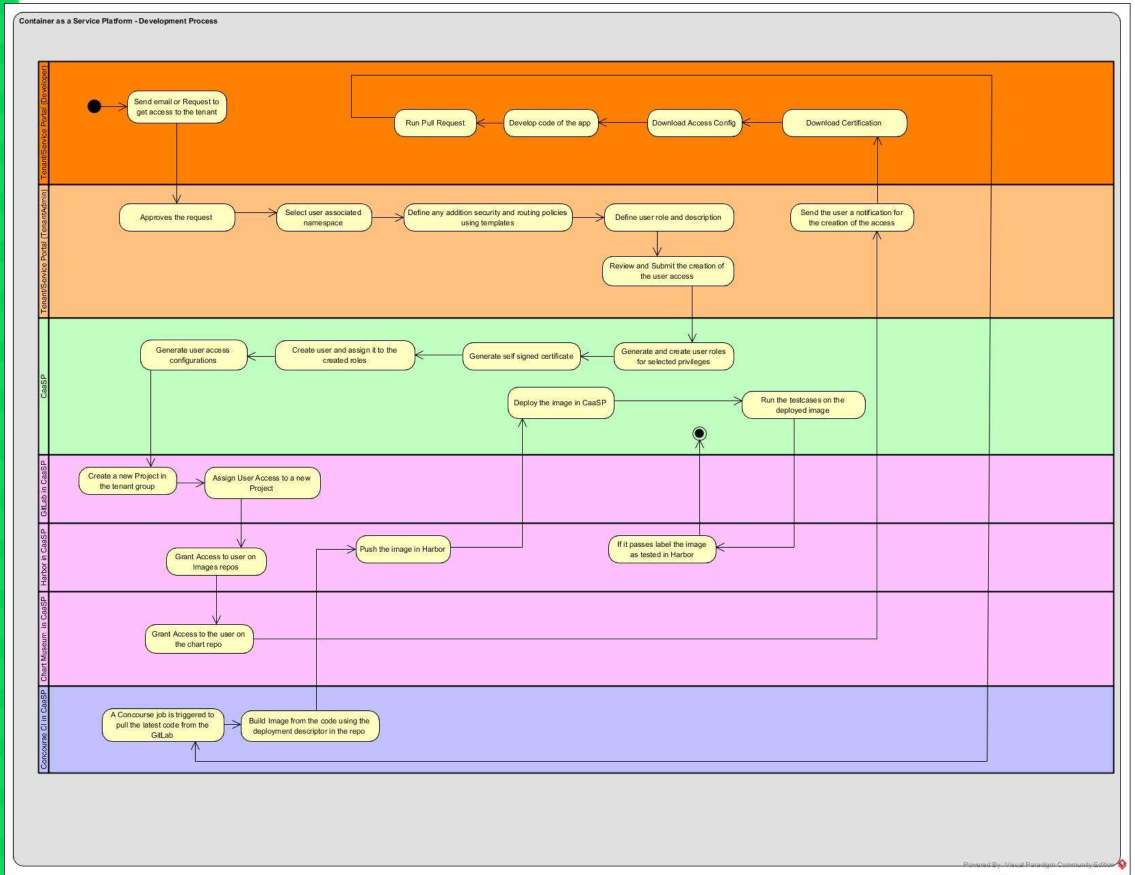
- Elasticity/scalability

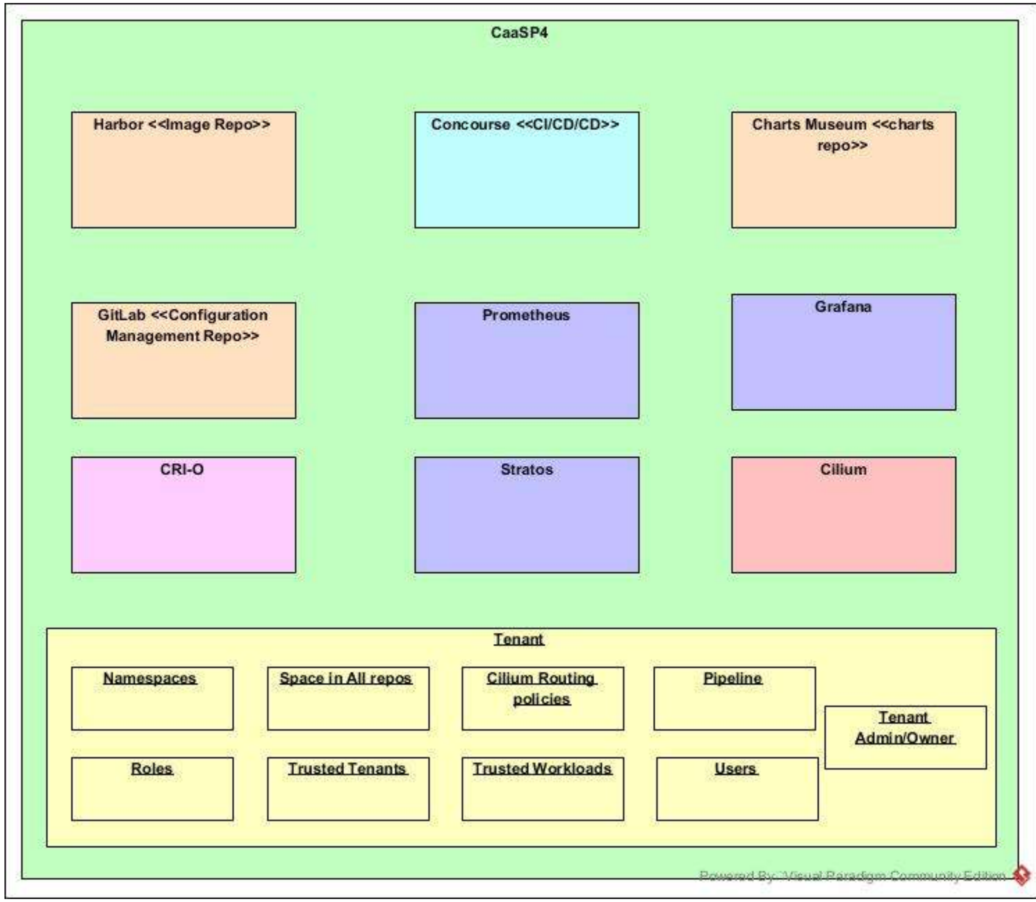
  Pay as you grow/On-demand availability

  cost effective and scalable
- More vendor agnostic/ higher flexibility
- Speed

  accelerate time to market
- Innovation (due to competition in Cloud)
- vast variety of different services (IaaS, SaaS, PaaS)
- Cloudmanagement and optimization increase enhancements in the process
- Greater Choice enables better cost control BUT danger of Cloud Sprawl and need for a „single pane of glass“ to prevent Sprawl

DevOps





https://www.suse.com/c/highly-automated-and-secured-multi-tenancy-using-suse-caas-platform-4/

Business Benefits DevOps

- Managing services not „IT assests“ – business driven
- Faster delivery of features or improvents in the sw
- Improved communication and collaboration
- More time to innovate
- Happier and more productive teams
- Faster recovery from incidents
- Lower change failure rate (due to smaller components to oversee)
- Results in increased Customer Satisfaction
- Faster revenue generation
- Increased efficiency and productivity

Please join us on our next session:

January 17th 2020
09:00 AM GMT

About making Choices – CaaSPv4 as SUSE's empowering of Kubernetes

Q&A

Thank you