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 communities

• 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/de-de/events/webinars)
  
  [https://www.suse.com/c/author/rasadus/](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 Usecases**

**About making Choices – CaaS/Pv4 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 Use cases
Basics about Kubernetes

History/Evolution of the Kubernetes

- **2003-2004**: Omega cluster management system
- **2013**: K8s (OS of Borg) – start of K8s community
- **2014**: Kube v1.0, CNCF, K8s ecosystem
- **2015**: Helm, MiniKube, Kops, Kubeadm, Pokemon, Windows support, OpenAPI
- **2016**: Kubelet, TLS, Bootstrapping, Gardener
- **2017**: Go Ent., RBAC, dynamic prov., Istio, API Agg., GitHub, Certs, Kubeflow
- **2018**: Google Anthos and more
- **2019**: Google Anthos and more

**Note:** The timeline shows significant milestones in the development of Kubernetes from its inception to its current status, highlighting key contributors and advancements in the technology.
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

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

• 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
Kubernetes Architecture - Blueprint
Kubernetes Components & Objects
Kubernetes Components

- Kube-DNS
- CoreDNS
- KubeVirt
- Prometheust
- Metrics Server
- fluentd
- Logrotate
- Elastic/ELK Stack
- Calico
- Cilium
- Flannel
- Canal
- CNI-Genie
- Multus
- Dashboard
- Weave Scope
- Hazelcast

Service Mesh can play in delivering any of the plugins as well as building new ones 😊
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

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

Overlay network: 100.96.0.0/16

Node 1
- eth0: 172.20.33.102/19
- docker0: 100.96.1.2/24
- flannel0: 100.96.1.0/16

Node 2
- eth0: 172.20.54.98/19
- docker0: 100.96.2.1/24
- flannel0: 100.96.2.0/16

Node 3
- eth0: 172.20.34.197/19
- docker0: 100.96.3.1/24
- flannel0: 100.96.3.0/16

Host network: 172.20.32.0/19

Encapsulation in UDP:
- src: 100.96.1.2, dst: 100.96.1.3
- src: 172.20.33.102, dst: 172.20.54.98, dport: 8285, UDP
- src: 100.96.1.2, dst: 100.96.2.3

Remove UDP header:
- src: 100.96.1.2, dst: 100.96.2.3
## Network Component – a comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Flannel</th>
<th>Calico</th>
<th>Multus</th>
<th>Cilium</th>
<th>Weave Net</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief features</strong></td>
<td>Simple</td>
<td>Connects pods using the same IP networking principles as the internet</td>
<td>Interoperable</td>
<td>Flexible</td>
<td>Creates a mesh overlay network between each of the nodes in the cluster</td>
</tr>
<tr>
<td>- Flat/overlay network</td>
<td></td>
<td>- Enables security enforcement (self workload firewall)</td>
<td></td>
<td></td>
<td>- Flexible in the communication</td>
</tr>
<tr>
<td>- Each pod has unique ip, all pods run on the same network</td>
<td></td>
<td>- Enable Cloud native scalability</td>
<td></td>
<td></td>
<td>- Simple</td>
</tr>
<tr>
<td>- Runs a daemon process flannel to update the ip routing table</td>
<td></td>
<td>- leverage best practice cloud-native design patterns</td>
<td></td>
<td></td>
<td>- Enables service discovery using micro DNS</td>
</tr>
<tr>
<td>- Mapping of the subnet to host info stored in etcd</td>
<td></td>
<td>- Enables hybrid soln using BGP</td>
<td></td>
<td></td>
<td>- Encryption</td>
</tr>
<tr>
<td><strong>Net. Layering</strong></td>
<td>L3 network fabric</td>
<td>L3 &amp; 7</td>
<td>N/A</td>
<td>L3 &amp; 7</td>
<td>L3</td>
</tr>
<tr>
<td><strong>Stability</strong></td>
<td>Very high</td>
<td>Very high</td>
<td>High</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td><strong>Service meshing</strong></td>
<td>Doesn’t integrate and doesn’t allow any network policy implementation</td>
<td>Integrates and enables defining rich network policy models</td>
<td>N/A</td>
<td>Integrates and enables defining rich network policy models</td>
<td>No</td>
</tr>
<tr>
<td><strong>Gateways?</strong></td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Perf.</strong></td>
<td>Good</td>
<td>Very Good</td>
<td>N/A</td>
<td>Very Good</td>
<td>Very Good</td>
</tr>
</tbody>
</table>

- 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
- It is a meta-plugin
- It highly support multi-tenancy
- 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
**Network Component – a comparison**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Flannel</th>
<th>Calico</th>
<th>Multus</th>
<th>Cilium</th>
<th>Weave Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief features</td>
<td>Simple</td>
<td>Connects pods using the same IP networking</td>
<td>It enables attaching multiple network</td>
<td>Supports MSA &amp; CNA</td>
<td>Creates a mesh overlay network between each</td>
</tr>
<tr>
<td></td>
<td>Flat/overlay network</td>
<td>principles as the internet</td>
<td>interfaces to the pods by creating homed</td>
<td>It works by network policies</td>
<td>of the nodes in the cluster</td>
</tr>
<tr>
<td></td>
<td>Each pod has unique ip, all</td>
<td>Interoperable</td>
<td>pod, it is magic 😊</td>
<td>Supports lightweight protocols, such as</td>
<td>Flexible in the</td>
</tr>
<tr>
<td></td>
<td>pods run on the same network</td>
<td>Flexible</td>
<td>It is a meta-plugin</td>
<td>HTTP, gRPC, Kafka</td>
<td>communication</td>
</tr>
<tr>
<td></td>
<td>Runs a daemon process</td>
<td>Enable security enforcement (self</td>
<td>It highly support multi-tenancy</td>
<td>It is an API-aware network security</td>
<td>Enables service</td>
</tr>
<tr>
<td></td>
<td>flannel to update the ip</td>
<td>workload firewall)</td>
<td></td>
<td>filtering.</td>
<td>discovery using micro DNS</td>
</tr>
<tr>
<td></td>
<td>routing table</td>
<td>True Cloud native scalability</td>
<td></td>
<td></td>
<td>Encryption</td>
</tr>
<tr>
<td></td>
<td>Mapping of the subnet</td>
<td>leverage best practice cloud-native design</td>
<td></td>
<td></td>
<td>Supports multi-cast</td>
</tr>
<tr>
<td></td>
<td>to host info stored in etcd</td>
<td>patterns</td>
<td></td>
<td></td>
<td>Enables portability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enables hybrid soln using BGP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Net. Layering               | L3 network fabric            | L3 & 7                                       | N/A                                         | L3 & 7                                      | L3                                            |
| Stabilty                    | Very high                    | Very high                                   | High                                        | High                                        |                                               |
| Service meshing             | Doesn’t integrate and doesn’t | Integrates and enables defining rich network | Integrates and enables defining rich        |                                               |                                               |
|                            | allow any network policy     | policy models                               | network policy models                       |                                               |                                               |
| Gateways?                   | No                           | Yes                                         | N/A                                         | Yes                                         |                                               |
| Perf.                       | Good                         | Very Good                                   | N/A                                         | Very Good                                   | Very Good                                    |

Notes:
- Magic 😊: This feature is particularly powerful, enabling the attachment of multiple network interfaces to the pods by creating homed pod.
- Meta-plugin: It is a flexible, lightweight solution that supports cloud-native design patterns and enables hybrid network policies.
- API-aware: It uses Linux kernel technology called BPF for API-aware network security filtering.
- BGP: It works with BGP for true cloud-native scalability.
- MSA & CNA: It supports MSA and CNA for cloud-native design patterns.
- Container networking: It works with container networking for lightweight protocols such as HTTP, gRPC, Kafka.
- Network policy models: It integrates and enables defining rich network policy models.
- Encryption enforcement: Enables L3 encryption enforcement between each node in the cluster.
Multus

Kubelet

CNI

Multus

Pod without CNI meta-plugin

Pod with CNI meta-plugin

Kubernetes servers (api-server, kubectl so on)

- Specific User Traffic
- (Liveness and Readiness) Probes
- Communication between API and Pod
# Network Component – a comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Flannel</th>
<th>Calico</th>
<th>Multus</th>
<th>Cilium</th>
<th>Weave Net</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simple</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flat/overlay network</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Each pod has unique ip,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>all pods run on the same network</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Runs a daemon process flannel to update the ip routing table</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mapping of the subnet to host info stored in etcd</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Layering</strong></td>
<td>L3 network fabric</td>
<td>L3 &amp; 7</td>
<td>N/A</td>
<td>L3 &amp; 7</td>
<td>L3</td>
</tr>
<tr>
<td><strong>Stability</strong></td>
<td>Very high</td>
<td>Very high</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td><strong>Service meshing</strong></td>
<td>Doesn’t integrate and doesn’t allow any network policy implementation</td>
<td>Integrate and enables defining rich network policy models</td>
<td>N/A</td>
<td>Integrates and enables defining rich network policy models</td>
<td>No</td>
</tr>
<tr>
<td><strong>Gateways?</strong></td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Perf.</strong></td>
<td>Good</td>
<td>Very Good</td>
<td>N/A</td>
<td>Very Good</td>
<td>Very Good</td>
</tr>
</tbody>
</table>

- 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
Cilium

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

Worker Node

- pod 192.168.11
  - eni1
  - BPF ingress
  - Linux Routing
    - 192.168.11/32 via eni1
    - 192.168.12/32 via eni2
  - eth0

- pod 192.168.12
  - eni2
  - BPF ingress
  - BPF ingress
  - BPF ingress
  - Cilium Agent

Cilium Agent

- POST /hello
- POST /index.html
- DELETE /concern
- GET /

Cilium Diagram:

1. Load-balancing
2. Network policy
3. Encryption
4. Multi-cluster
5. Visibility
## Network Component – a comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Flannel</th>
<th>Calico</th>
<th>Multus</th>
<th>Cilium</th>
<th>Weave Net</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief features</strong></td>
<td>Simple</td>
<td>- Connects pods using the same IP networking principles as the internet - <strong>Interoperable</strong> - <strong>Flexible</strong> - Enable security enforcement (self workload firewall) - True Cloud native scalability - leverage best practice cloud-native design patterns - Enables hybrid soln using BGP</td>
<td>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</td>
<td>- Supports MSA &amp; 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</td>
<td>- Creates a mesh overlay network between each of the nodes in the cluster - <strong>Flexible</strong> in the communication - Simple - Enables service discovery using micro DNS - Encryption - Supports multi-cast - Enables portability</td>
</tr>
<tr>
<td><strong>Net. Layering</strong></td>
<td>L3 network fabric</td>
<td>L3 &amp; 7</td>
<td>N/A</td>
<td>L3 &amp; 7</td>
<td>L3</td>
</tr>
<tr>
<td><strong>Stability</strong></td>
<td>Very high</td>
<td>Very high</td>
<td>High</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td><strong>Service meshing</strong></td>
<td>Doesn’t integrate and doesn’t allow any network policy implementation</td>
<td>Integrates and enables defining rich network policy models</td>
<td>N/A</td>
<td>Integrates and enables defining rich network policy models</td>
<td>No</td>
</tr>
<tr>
<td><strong>Gateways?</strong></td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Perf.</strong></td>
<td>Good</td>
<td>Very Good</td>
<td>N/A</td>
<td>Very Good</td>
<td>Very Good</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Security</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf.</td>
<td>Good</td>
<td>Very Good</td>
<td>Very Good</td>
<td>Very Good</td>
<td>Very Good</td>
</tr>
<tr>
<td>Standard</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Stability</td>
<td>Very high</td>
<td>High</td>
<td>Very High</td>
<td>Very high</td>
<td>High</td>
</tr>
</tbody>
</table>
rkt
containerd
Container engine vs Containers Orchestrator
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

<table>
<thead>
<tr>
<th>Heapster, InfluxDB, &amp; Grafana</th>
<th>Prometheus &amp; Grafana</th>
<th>Heapster &amp; ELK Stack</th>
<th>Datadog</th>
<th>Dynatrace</th>
</tr>
</thead>
</table>
| - Heapster, is a uniform platform which push monitoring metrics to a external tool to process | - 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 Elastic 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 \(\Rightarrow\) 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 works on the network packet level so it is an INF layer still.

Sidecars are always stateless 😊

**Data Plane (mainly gathering data transforming or converting it to events and forward it):**

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 / 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

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)
• Cloud management 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

Business Benefits DevOps

- Managing services not „IT assets“ – business driven
- Faster delivery of features or improvements 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 – CaaS Pv4 as SUSE’s empowering of Kubernetes
Q&A
Thank you
Unpublished Work of SUSE LLC. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE LLC. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.