SUSE

We adapt. You succeed.

Container and Cloud Native Application Platform

Why do we need them and what is so great about it?

Bettina Bassermann
Product Sales Specialist Cloud & 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

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

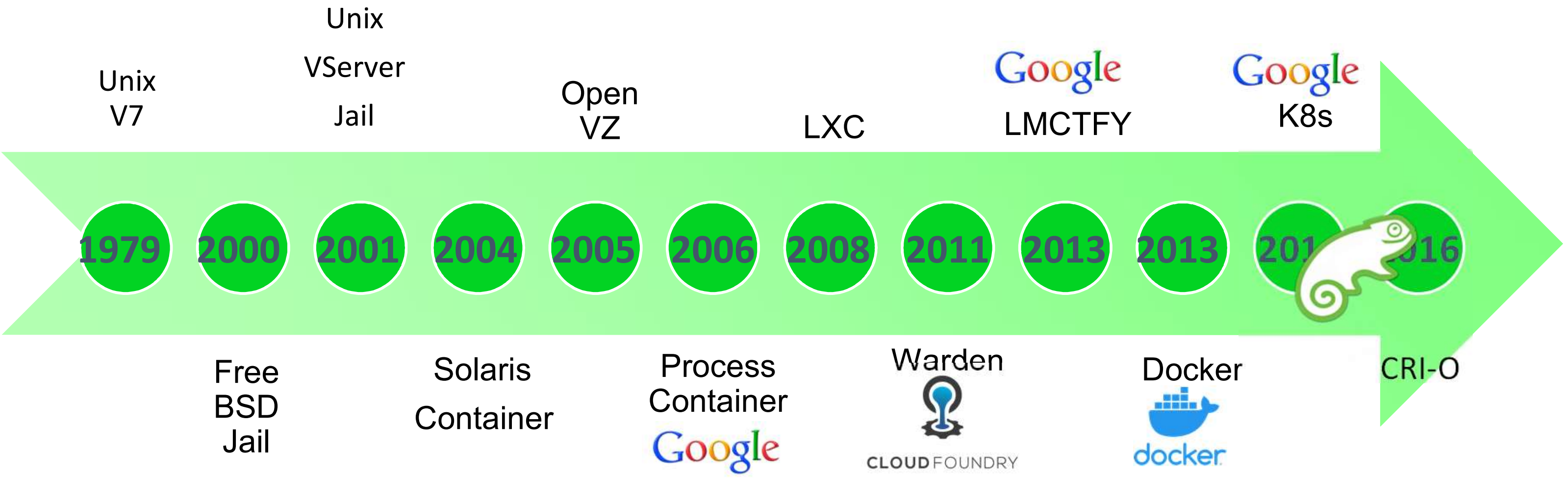
Agenda

- Basics about Containerization
- Virtualization vs Containerization
- Benefits and Challenges for Containers
- How to overcome container Challenges
- Container Engine vs Containers Orchestrator
- Docker, Kubernetes and CRI-O – a comparison
- Containers & Cloud Native Development
- What’s the business value?
- Future of Containerization (potential outlook)

Basics about Container Technologies

History/Evolution of the containerization

- **Container has been there since 1970s ☺**



1979 Unix V7

2000 Free BSD Jail

2001 Unix VServer Jail

2004 Solaris Container

2005 Open VZ

2006 Process Container Google

2008 LXC

2011 Warden CLOUD FOUNDRY

2013 Google LMCTFY

2013 Docker docker

201 Google K8s

016 CRI-O

Life Post & Pre Containers

Basics about Container Technologies - Why?

Life pre-containers

- Big development environments
- Inconsistent environments
- Hard to handle variable loads.
- Limitation in Vertical scaling
- Hard to scale horizontally
- Testing limitation even with automation
- Expensive maintenance
- Hard to troubleshoot PROD problems
- Hard to deliver high SLAs
- Big Shared databases
- Hard to isolate and maintain dependencies
- Emulators (expensive and not efficient)

Life post-containers

- Smaller footprint
- Consistency for everyone.
- Ease of handling variable loads
- Enable auto scaling as per the need
- Better testing coverage
- Ease and cost efficient maintenance
- Ease of troubleshooting PROD problems
- Enable delivering High SLAs
- Enable decentralization of data
- Enable autonomous/separation of concerns and isolation
- Enable building cost efficient emulators and simulators in the cloud and ground

Basics about Container Technologies -

Characteristics of containers

- A Container runs an Image → abstraction
- Image only has a minimal OS (no real OS), the app code, all necessary executables, binaries, libraries, & configuration files
- Containers are executed by a runtime engine
- Highly Portable
- No depends on any Guest OS or hardware
- Lightweight

- Repeatable
- Containers may communicate with each other directly
- Sharing the kernel of the host operating system
- Each container has a single executable service
- Fast provisioning of a container instance (nano to seconds depending on the size of the Image)
- Dependencies are manageable at the runtime

Containers don’t care for the host OS, its OS is minimal so it shouldn’t matter if the minimal OS is Linux or windows ☺

Containers & Virtual Machines

Virtualization vs Container Technologies

What is a VM?

- Virtualizing a machine hardware running a complete OS
- Must run on a Hypervisor running on the host OS to allocate resources for the VM
- Allow running different OS on the same hardware
- Each VM owns its own kernel

Point of Comparison	Physical servers	Virtual Machine
Scalability	Hard to scale.	Easy to scale vertically.
Maintenance	Hardware gets old and maintenance gets expensive and hard.	Maintenance is better especially from a hardware perspective.
Cost	Very expensive.	Much less expensive.
Performance	Better performance as the full power of the physical hardware is dedicated to the application.	It is not as good as the physical server's capacity.
Footprint	Large footprint.	Smaller footprint.
Security	You are in control and charge of hardware and network so advanced security policies can be implemented	it is limited as you cannot physically isolate the network, the data communication and storage. You still can implement security on the data and the routing of it.
Portability	Is not portable at all.	Highly portable

Virtualization vs Container Technologies

Virtualization Challenges

- Failover
- High SLA
- Complexity
- Scaling (horizontally and vertically)
- Cost of Maintenance
- Cost of Hardware and software upgrades
- Time and effort of building a VM
- Portability
- Agility

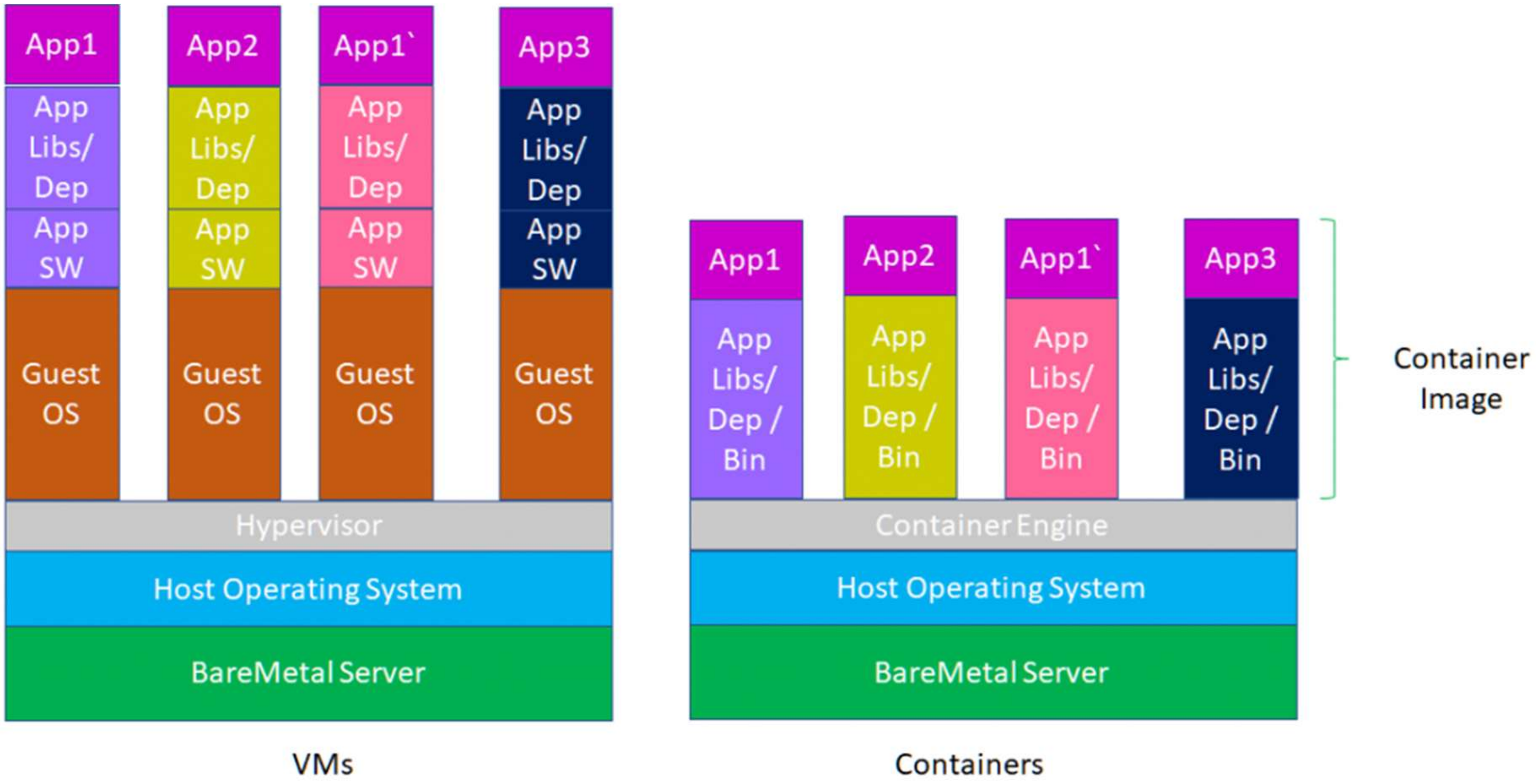
Virtualization vs Container Technologies

VM vs Containers

Point of Comparison	Virtual Machine	Container
Scalability	**Easy** to scale horizontally but have **some limitation** on the vertical scaling. Scalability is **expensive**.	**Easy** to **scale** horizontally; no need to scale vertically. Scalability is not **expensive**.
Maintenance	Maintenance is **hard**.	Maintenance is **very simple** and **more efficient**. The owner of the image is the one responsible for maintaining it.
Cost	Much **more expensive** than a container.	**Almost zero cost**, depending on the software used by the image. No cost of the operating system as it is very light. You are only paying tor licenses and support for the software installed above the operating system. **Licenses** in this case are **much cheaper** than the VM because most of the software licensing models are based on VCores/Cores — allowing you to host a number containers above it.
Performance	**Better performance** as no kernel is shared.	It is **very good** even though the same kernel is shared in the hosting environment (whether it is a VM or a Bare Metal machine).
Footprint	**Larger footprint**.	**Extremely small** footprint.
Security	**Better security** because no sharing occurs in the kernel.	**Security** can be a **challenge** when using containers given that containers are sharing the **same kernel**. With containers, you **cannot physically isolate** the network and the data communication and storage. Potentially the community has started working on building a **lightweight** VM with a very **small footprint** like a container but with its **isolated runtime** and **kernel**.

Virtualization vs Container Technologies

VM vs Containers



Virtualization vs Container Technologies

Is a VM being replaced by containers?

- Answer is NO ☺
- Container were introduced because of the challenges in App Delivery on VMs
- Containers don’t replace entirely the VMs
- It depends on the needs and requirements

Virtualization vs Container Technologies – When?

Go for VM when

- Solution is simple
- Business is stable or small
- No frequent development on the app happening
- Load is predictable and there is not big variant between the peak and low loads.
- Integration is not complex
- No nonfunction issues with the application

Go for Containerization when

- Solution is complex
- Requirements are changing frequently
- Load is a variable
- Have big teams of development
- Have multiple third party developing services
- Targeting Digital Business
- Building an as a service (PaaS, SaaS or XaaS)
- Targeting multi-cloud

Again, don’t build a ship to sail to your home because you love sailing, biking can still get you home in 5 minutes ☺

Containerization Benefits & Challenges

Benefits and Challenges for Containers?

Benefits of Containers

- Scalability
- Simplicity
- Cost efficient
- Enable digital transformation
- Supports time to market apps
- Consistency – you see what the developer sees ☺
- Smaller footprint
- Standards
- Enable Autonomous
- Flexibility

Benefits and Challenges for Containers?

Challenges of Containers

- Security
- Governance
- Integration
- E2E Troubleshooting
- Orchestration of containers lifecycle
- Management
- Logging
- Monitoring

How to overcome containers challenges?

- Powerful Containers Orchestrator engine
- Secured Containers (e.g. KataContainers)
- Standard and secured Container Runtime (e.g. CRI-O)
- Service Mesh
- Event Driven Architecture
- FaaS

Containerization & Workload Orchestration

Container engine vs Containers Orchestrator

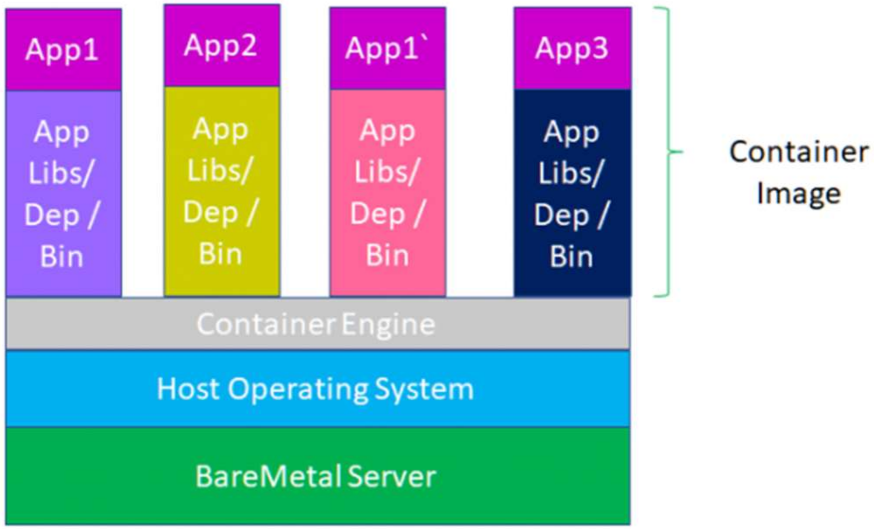
Containers engine/Runtime

- 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
- High level and low level container runtime

Containers Orchestrator

- Manage and operate the lifecycle of a container(creation, termination, failure…)
- Auto Scaling
- Mange communication and integration for containers
- Manage containers dependencies
- Deliver containers cluster by the idea of deployments
- Offers different flavours of containers clusters like daemon set, static and others.
- Containers Storage management
- Enable Infrastructure services for containers such as load balancers
- Enables service discoverability

Container engine vs Containers Orchestrator



App1
App Libs/ Dep / Bin

App2
App Libs/ Dep / Bin

App1`
App Libs/ Dep / Bin

App3
App Libs/ Dep / Bin

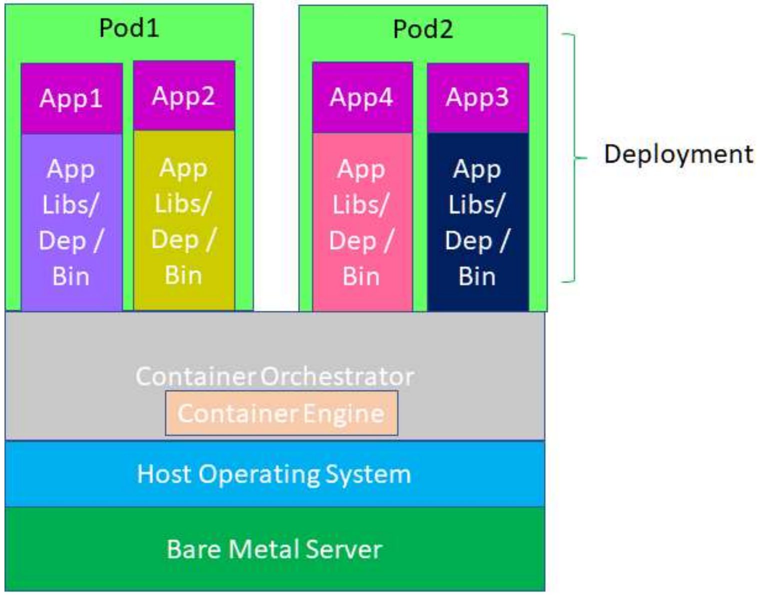
Container Image

Container Engine

Host Operating System

BareMetal Server

Containers



Pod1
App1
App Libs/ Dep / Bin
App2
App Libs/ Dep / Bin

Pod2
App4
App Libs/ Dep / Bin
App3
App Libs/ Dep / Bin

Deployment

Container Orchestrator
Container Engine

Host Operating System

Bare Metal Server

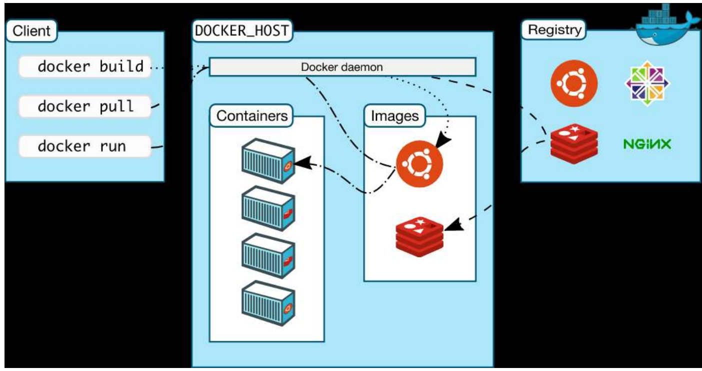
Container as a Service

Docker Vs CRI-O Vs K8s

Docker, Kubernetes and CRI-O – a comparison

What is Docker?

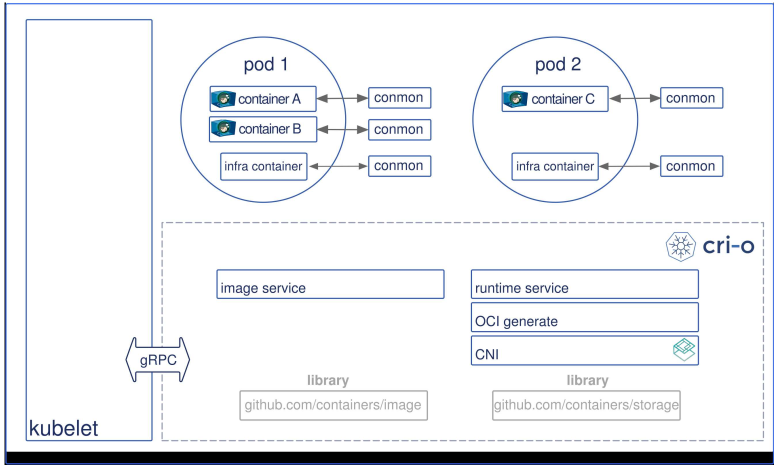
- The most Famous Container runtime ☺
- It starts by LXC implementation then moved to containerd (docker containerization and virtualization library)
- It has its own famous clis (docker and dockercompose)
- More into client-server architecture.
- Fat daemon (always running) – dockerd
- Uses Linux control groups
- Uses Linux namespaces to isolate containers
- No separation of concerns everything is done by dockerd
- No Standards and no limitations ☺



Docker, Kubernetes and CRI-O – a comparison

What is CRIO?

- An implementation of CRI enabling OCI runtime compatible
- Has a set of powerful utilities and clis such as crictl, podman, buildah, skopeo
- Is a distributed services architecture, more into MicroServices.
- Implements CNI
- No Fat Daemon
- CRIO generates OCI JSON file which is used to run the container on an OCI compatible runtime (runc)
- Containers are monitored separately using conmon
- More into Kubernetes standards



Docker, Kubernetes and CRI-O – a comparison

What is K8s?

- Market leading Container Orchestrator
- Architecture is based on API-centric and plugin design principles
- It enable building a cluster running containers to support building caas and paas
- A cluster has two types of machines (nodes), master and worker/minion nodes
- Master node holds the control plane of the cluster.
- Worker node holds the workloads running in the cluster.
- Both nodes run a container runtime (if Master CP is containerized)

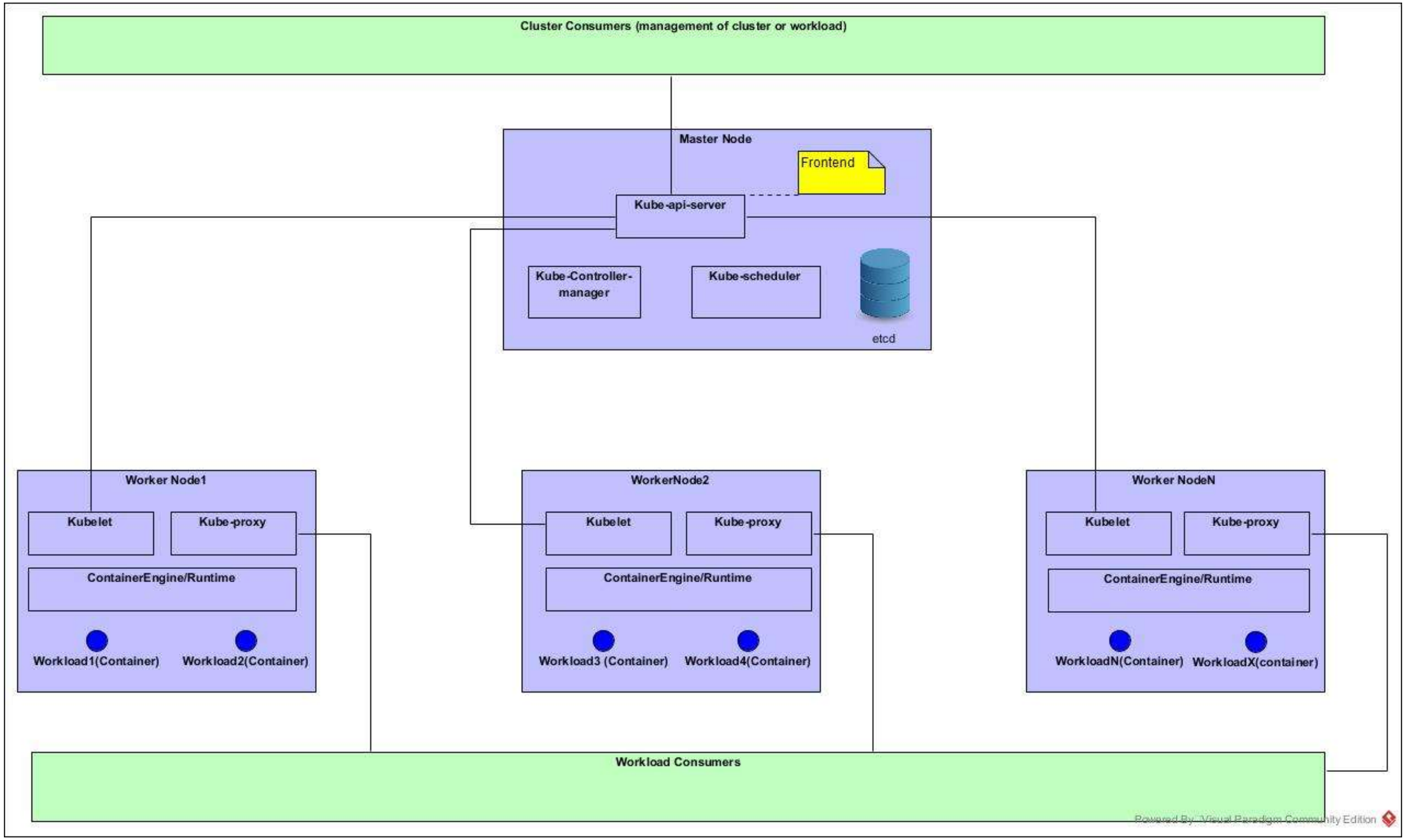
Docker, Kubernetes and CRI-O – a comparison

What is K8s?

- **Master Node components:**
  - **Kube-API Server**, it is the frontend for the K8s cluster
  - **Kube-Scheduler**, component responsible for scheduling and managing the workloads (pod), it is more the implementer. It has the power to determine where the workload best to be scheduled
  - **Kube-Controller manager**, component is the brain of K8s, it is always watching the cluster current state and determine the proper actions to achieve the desire state
  - **Etcd**, it is a key-value database/store hosting k8s cluster state and collected data
- **Worker Node components:**
  - **Kubelet**, it is the node agent through which the kube-API Server fetch data from the node and send updates to the node running workloads.
  - **Kube-proxy**, it controls the network of the node, including the service implementation for load balancing and request forwarding

Docker, Kubernetes and CRI-O – a comparison

What is K8s?



Docker, Kubernetes and CRI-O – a comparison

Docker

- Not Standard
- Heavyweight/fat daemon
- Central architecture
- Has security constraints
- Has no limitation ☺

CRI-O

- Standard implements CRI and support OCI
- 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

K8s

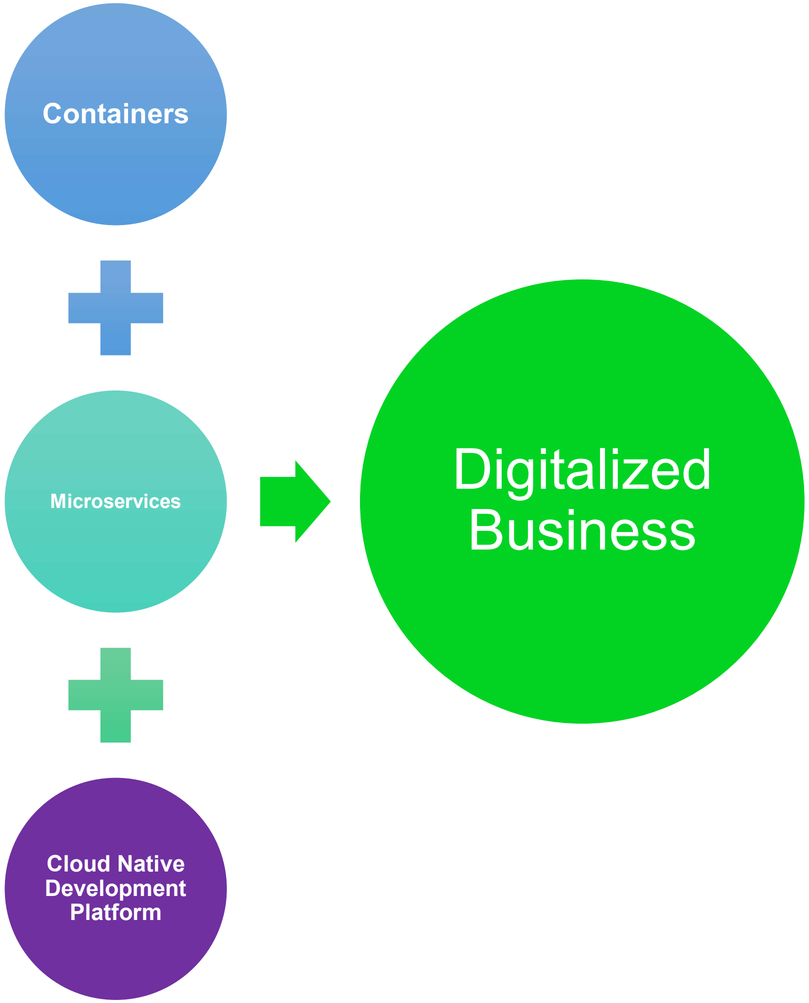
- Define the orchestration standards
- Lightweight and more API-Centric and plugins approach
- Decentralized architecture, more distributed solution
- More into monitoring and orchestration, doesn’t run containers

Containers & Cloud Native

Containers & Cloud Native Development

- Containers can be used to implement cloud native apps.
- Cloud Native Development is more about designing principles.
- Containers supports basic principles of the cloud native but doesn’t enforce it (Stateless, event driven architecture, autonomous…)
- Container Images can be used as a base runtime for Cloud Native Development, like in Cloud Foundry

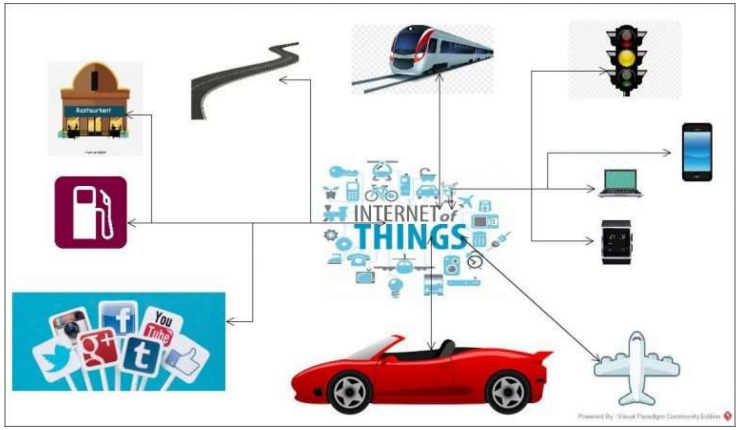
What is the business value?



What is the business value?

What is digitalized Business?

- Integration of different business contexts → smart digital market
- Allow business entity learns from other business entities
- Business entity builds a big ecosystem enabling
  - Continuous innovation
  - End user experience enhancements
  - Responsive business rather than reactive business



DATA is the fuel of the Digitalized business

Complex event processing and event processing is th

engine of the digitalized business

What is the business value?

Why digitalized Business? Why take the risk?

- **To avoid losing and shrinking business ☺**

"In today's era of volatility, there is no other way but to re-invent. The only sustainable advantage you can have over others is agility, that's it. Because nothing else is sustainable, everything else you create, somebody else will replicate."

Jeff Bezos, Amazon

"At least 40% of all businesses will die in the next 10 years… if they don't figure out how to change their entire company to accommodate new technologies"

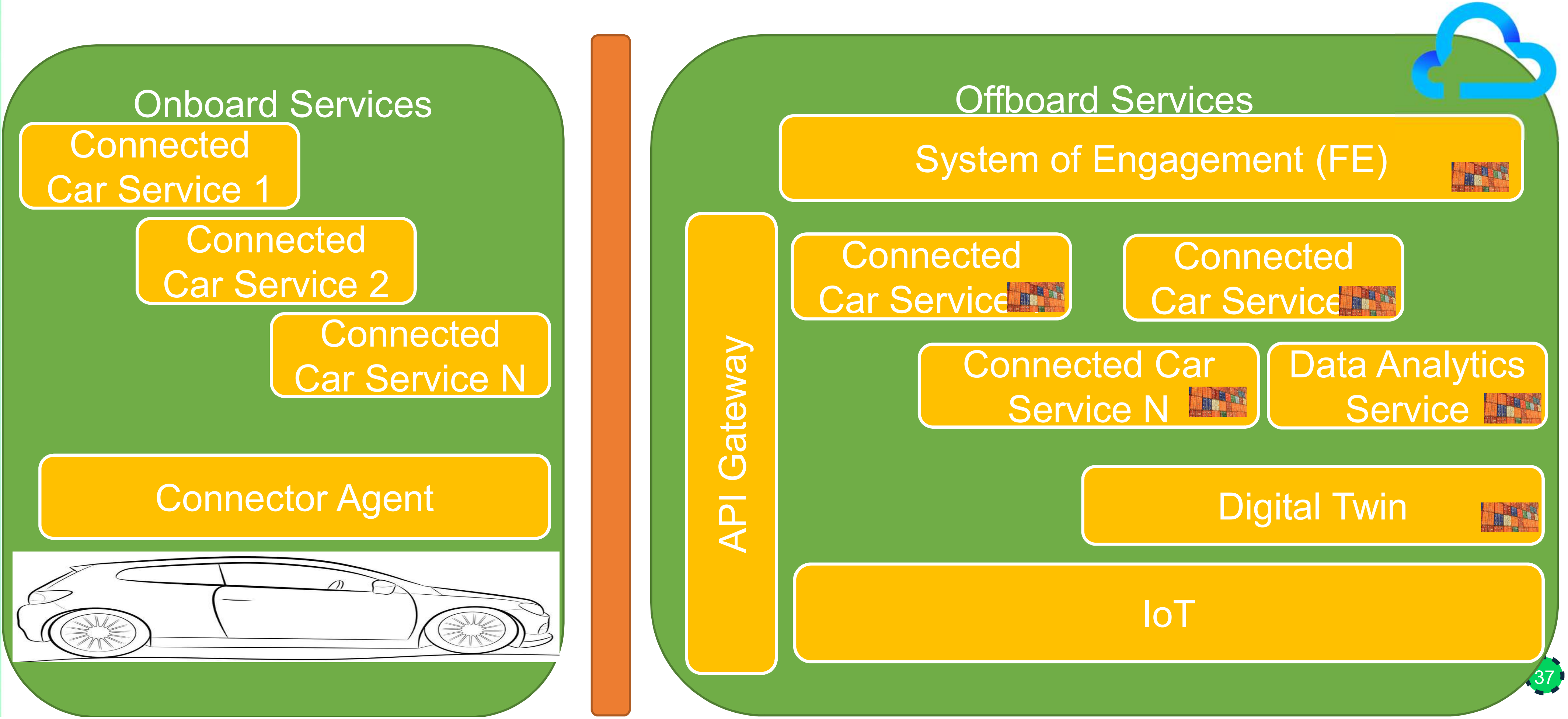
John Chambers, Cisco

Enterprises not considering digitalizing their business are destined to be part of the 80% of companies that are at risk of collapse according to TechCrunch

Business UseCase

What is the business value?

Automotive Usecase – Connected Car



Onboard Services

Connected Car Service 1

Connected Car Service 2

Connected Car Service N

Connector Agent

Offboard Services

System of Engagement (FE)

API Gateway

Connected Car Service

Connected Car Service

Connected Car Service N

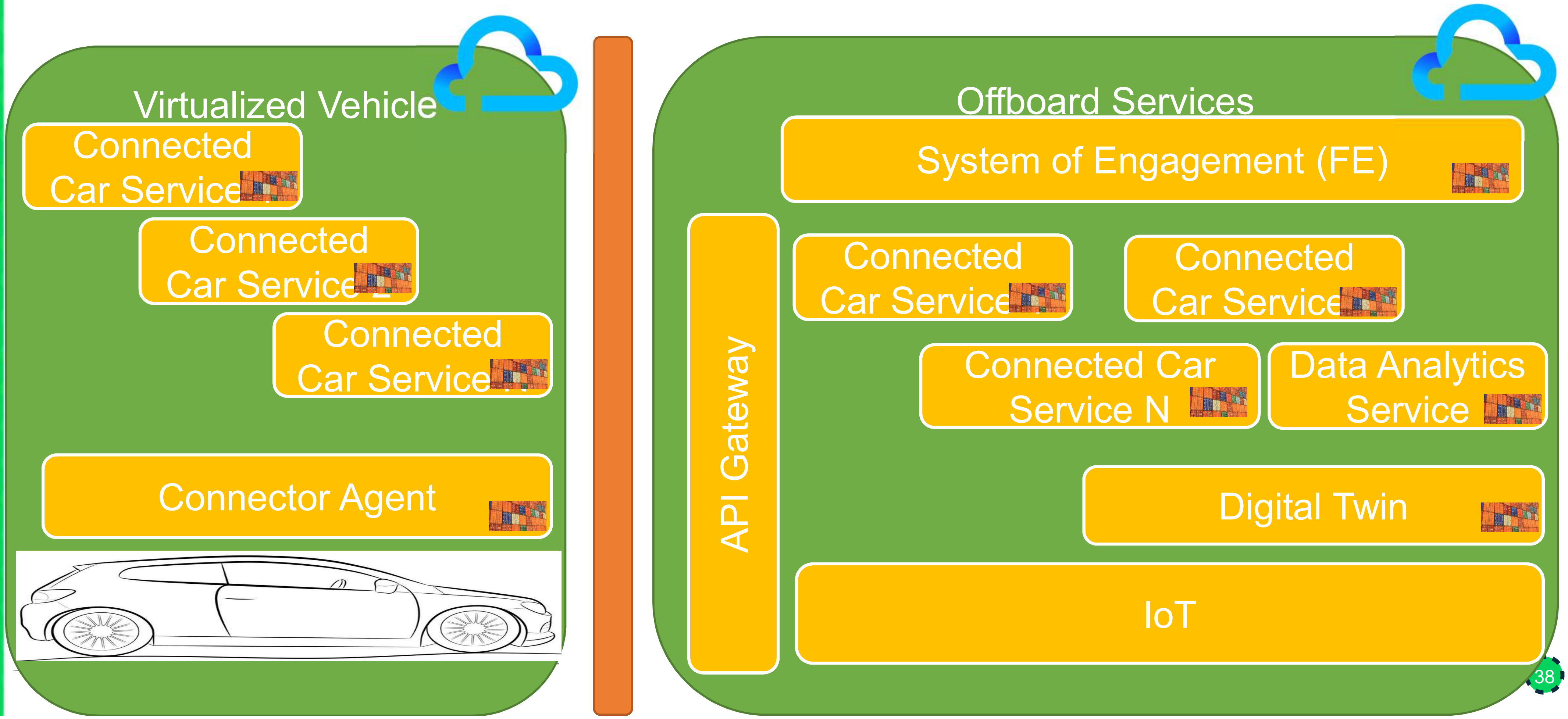
Data Analytics Service

Digital Twin

IoT

What is the business value?

Automotive Usecase – Connected Car



A vision for the future of containerization

Future of Containerization (potential outlook)

- CRI-O is the future of container runtime
- Kubernetes enables cluster of clusters (as a hierarchy and tree of organizations)
- Kubernetes enables running MSA in devices such as vehicles □ like a Micro or nano cluster →kubeEdge
- Kubernetes gets involved more in Machine to Machine communication:
- Controlling the lifecycle of events
- Enabling publishing of patches for MSA into devices
- Managing the lifecycle of machine/device service releases
- More and more engagement in the implementation of the system of engagement as well as system of records

Please join us on our next session:

November 22nd 2019
09:00 AM GMT

Why Kubernetes?

A Deep Dive in Options, Benefits and Usecases

Q&A

Thank you

Unpublished Work of SUSE LLC. All Rights Reserved.

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.