Monitoring Docker & Kubernetes

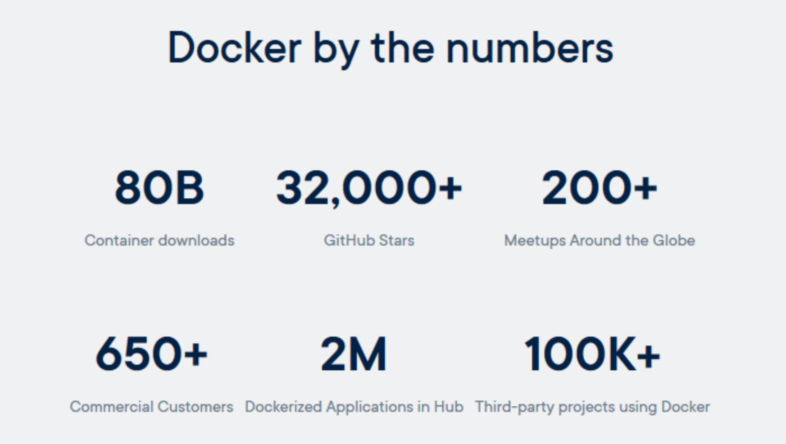
CHECKMK CONFERENCE #5 – MUNICH, APRIL 29, 2019

tribe29

Agenda

1. **DOCKER MONITORING**
2. KUBERNETES MONITORING
3. DEMO
4. OPEN QUESTIONS AND OUTLOOK

It’s a container world ...



Docker by the numbers

80B	32,000+	200+
Container downloads	GitHub Stars	Meetups Around the Globe
650+	2M	100K+
Commercial Customers	Dockerized Applications in Hub	Third-party projects using Docker

- Containers are here to stay
- Application packaging & delivery
- Light weight
- Portable
- Spun up in seconds

Docker world – a hardware and a software view

Hardware

Node_001
CPU: 3000
RAM: 64 GB

Nodes

- Node = single machine
- Can be physical or virtual
- Described by CPU and RAM

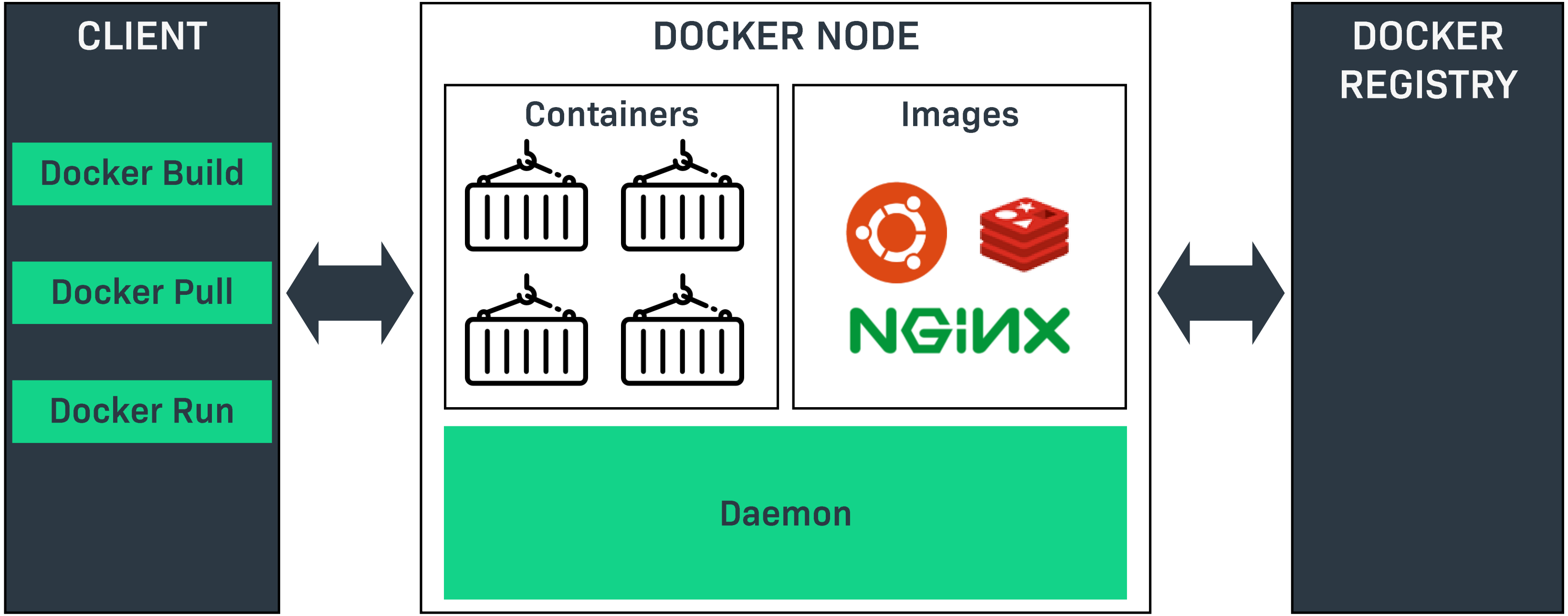
Software

Containers

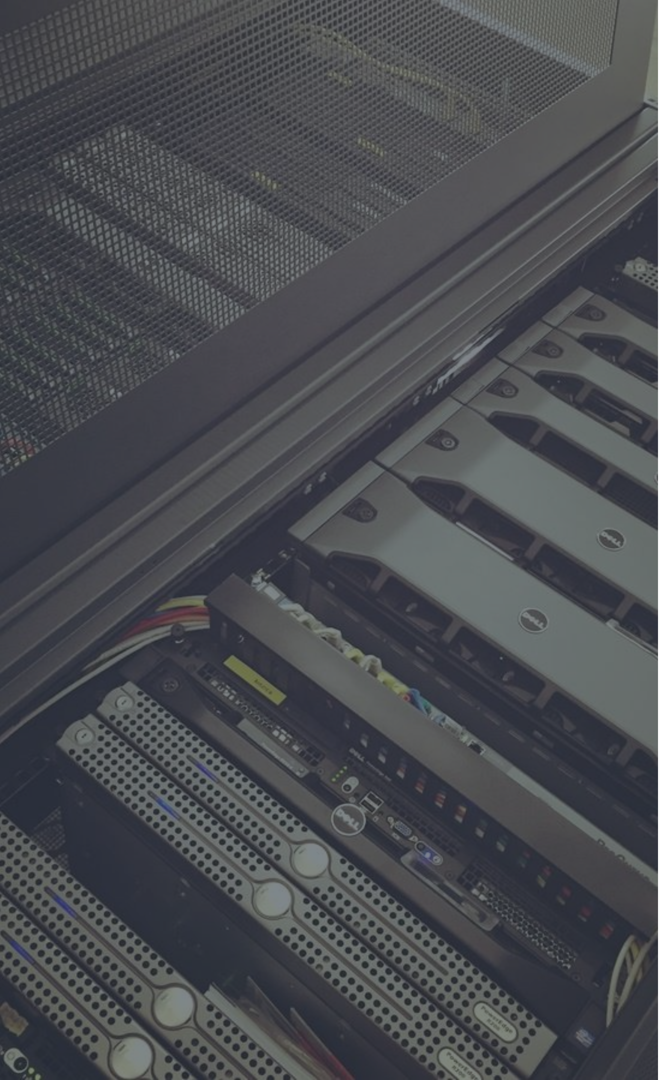
- Programs need to run in a container
- Rather have many small containers

DOCKER MONITORING

A look at the main components



tribe29



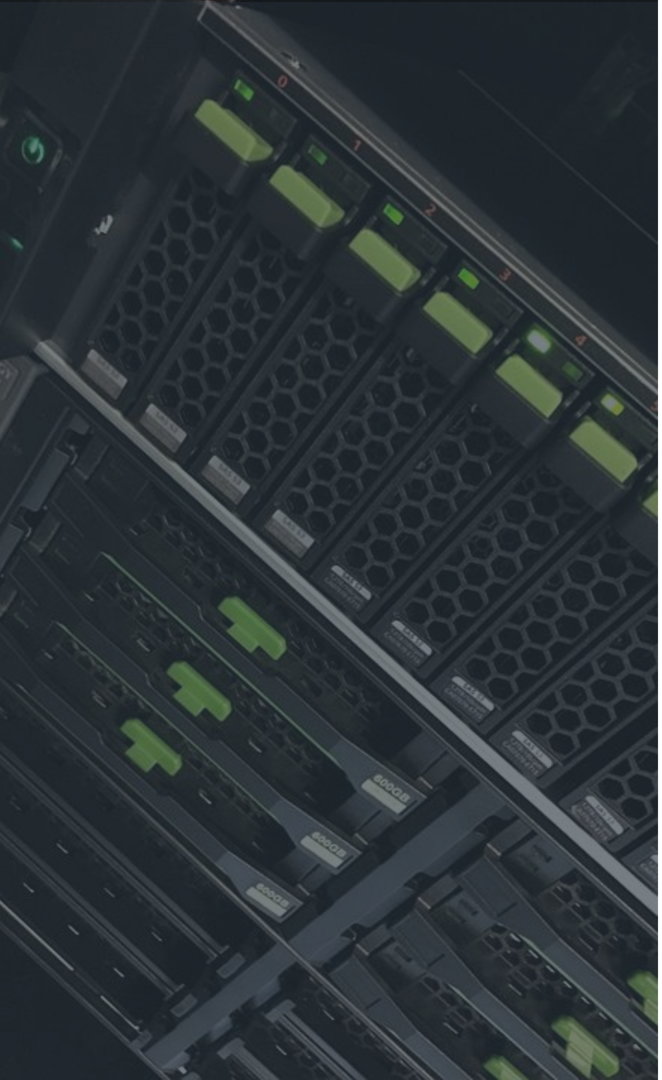
DOCKER MONITORING

Current Docker Monitoring (1.5)

- Docker command line interface with JSON output
- Combined approach with two plug-ins:
  - **’mk_docker_node’:** all node information
  - **’mk_docker_container_piggybacked’:** all container info
- Plugins generate hosts, services and inventory

Current Docker Checks

	Checks	Inventory
Nodes	- System Status - #images - #containers - Disk usage	- Docker version - Labels - Networks
Containers	- CPU utilization - Disk throughput - Container health - Memory usage - Status	- Node running on - Labels - Networks
Images		- Time created - Labels - Size - #containers (state) - Repository - Tag - ID - #images



DOCKER MONITORING

Best practice for current set-up

- Best results:
  - Agent installed in the container (monitor from ’within’)
  - Each container set up as host (otherwise: node attribute)

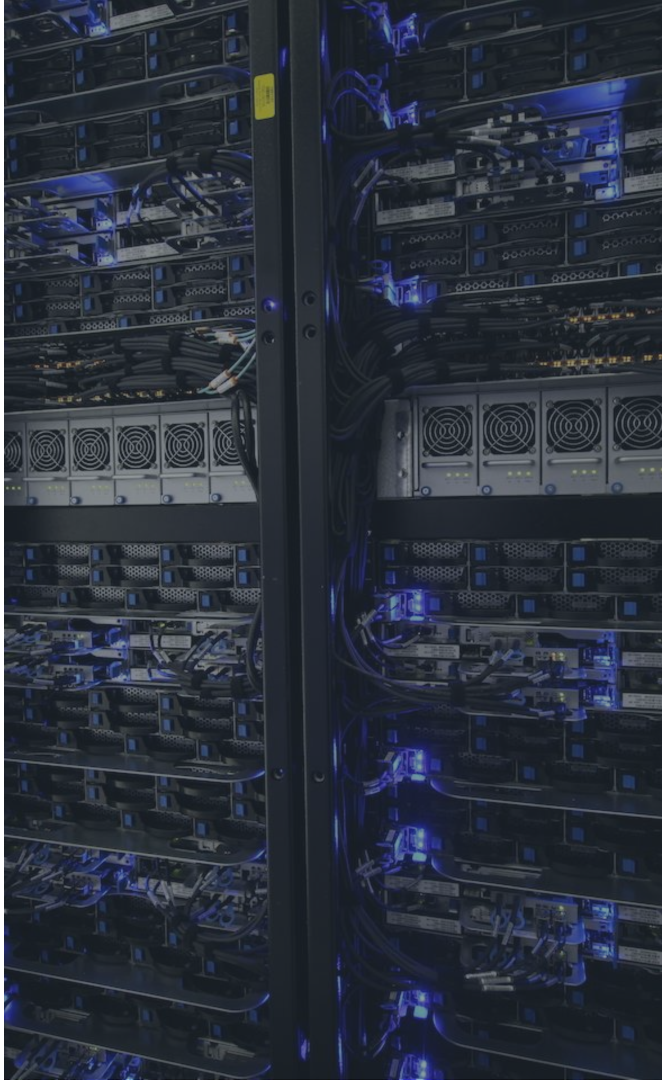
- To create hosts, use the short container ID as name
  - Manual or scripted in 1.5
  - Automated in 1.6

DOCKER MONITORING

Challenges of current set-up

- Older Docker versions: No JSON support
- Command line interface: Performance issues

tribe29



The new Docker Monitoring Plugin: mk_docker.py (1.6)

- Leverages Docker’s Python-API
- Combines functionality of two existing 1.5 plug-ins
- 1.5 plug-ins will become deprecated
- Adds more configuration options
- Requirement: 'docker-py' python library

- MKP available (from 1.5.0p12) → check**mk** exchange

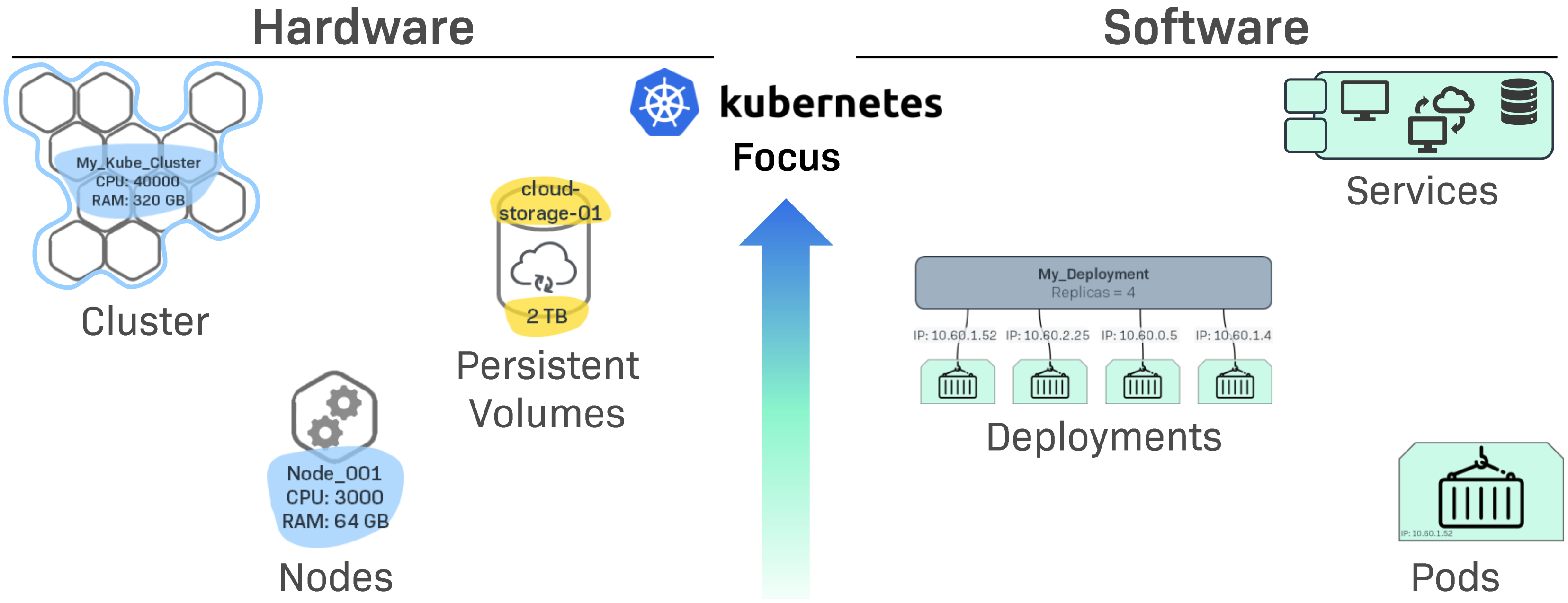
Agenda

1. DOCKER MONITORING
2. **KUBERNETES MONITORING**
3. DEMO
4. OPEN QUESTIONS AND OUTLOOK

tribe29

KUBERNETES MONITORING

101: Hard- & Software in the Kubernetes World



Hardware

Software

kubernetes

Focus

My_Kube_Cluster
CPU: 40000
RAM: 320 GB

Cluster

cloud-storage-01
2 TB

Persistent Volumes

Node_001
CPU: 3000
RAM: 64 GB

Nodes

Services

My_Deployment
Replicas = 4

IP: 10.60.1.52
IP: 10.60.2.25
IP: 10.60.0.5
IP: 10.60.1.4

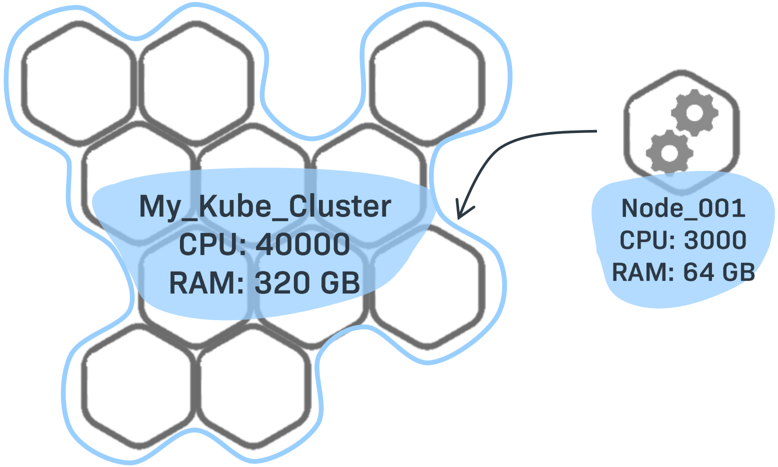
Deployments

IP: 10.60.1.52

Pods

KUBERNETES MONITORING

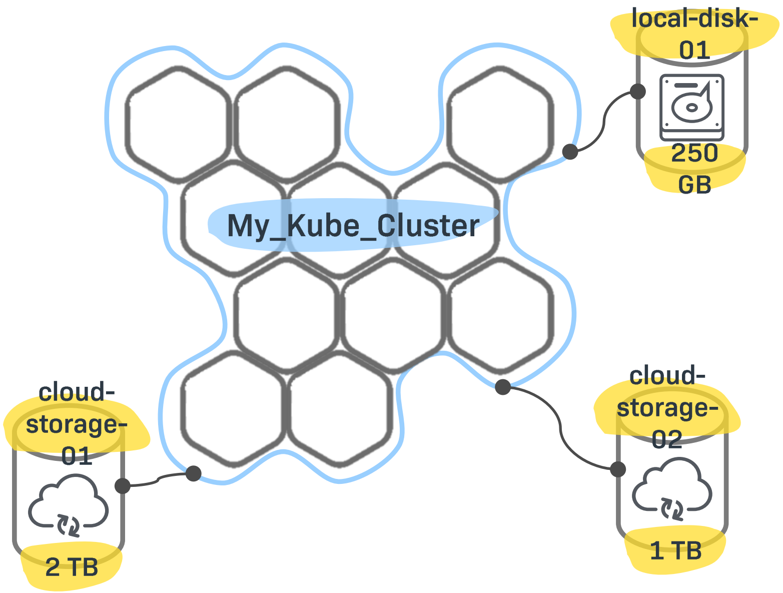
Hardware: Think in *clusters*, not *nodes*



- Cluster = many nodes
- Resources are pooled together
- Kubernetes magically distributes work

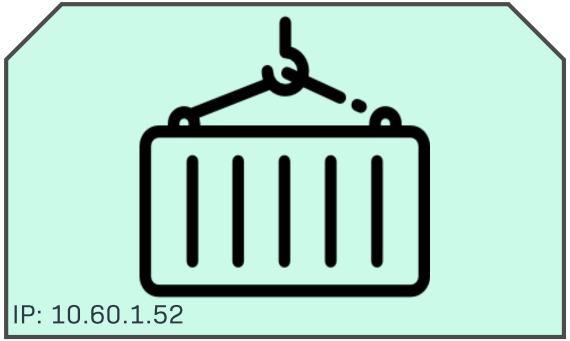
tribe29

HW: *Persistent volumes* – where data is stored



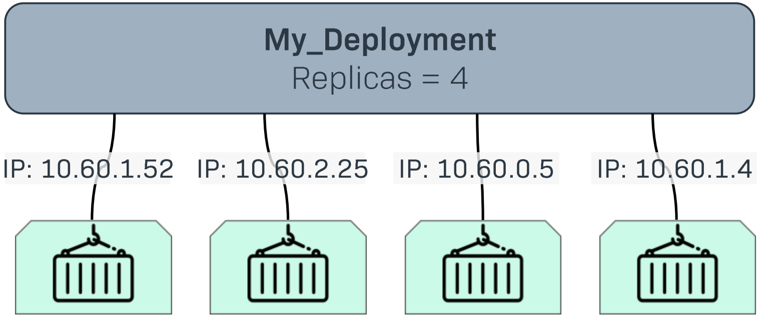
- Storage on node does not persist
- Data is stored on “Persistent Volumes”
- Mounted to the cluster

SW: Kubernetes manages *Pods*, not containers



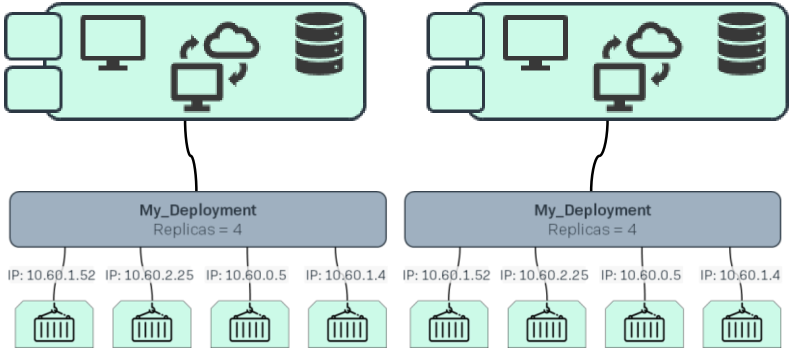
- Usually one container
- Can consist of several containers
- Share same resources & local network
- Pods are Kubernetes’ replication unit
- Design as small as possible

SW: *Deployments* – one more abstraction layer



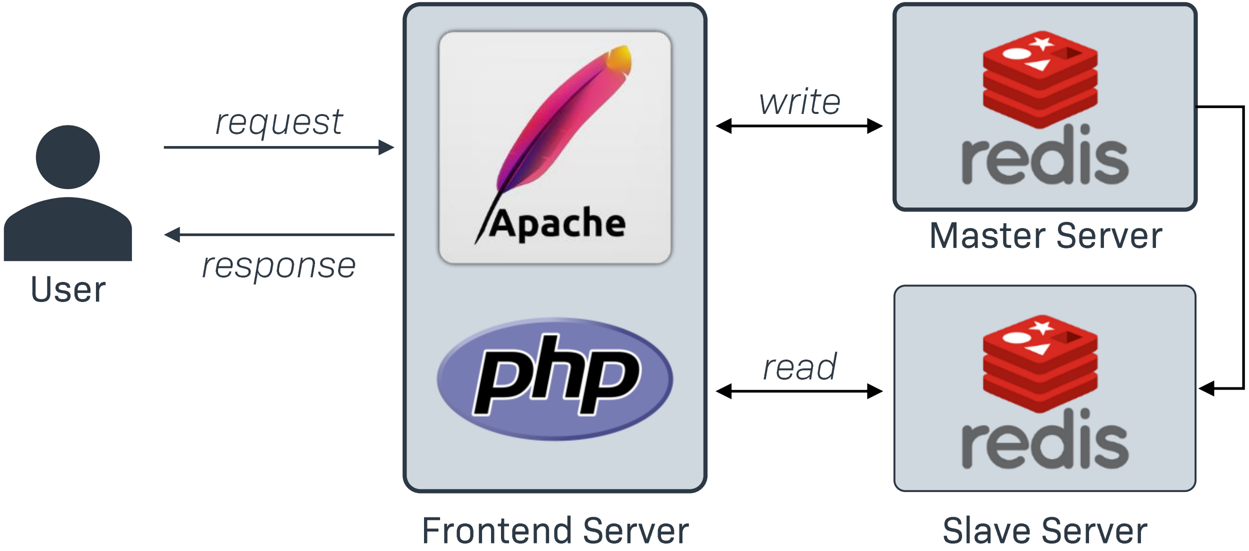
- Manages pods
- No need to deal with pods manually
- Just define a desired state, e.g. 4 pods
- Deployment ensures availability of pods

SW: *Microservices*

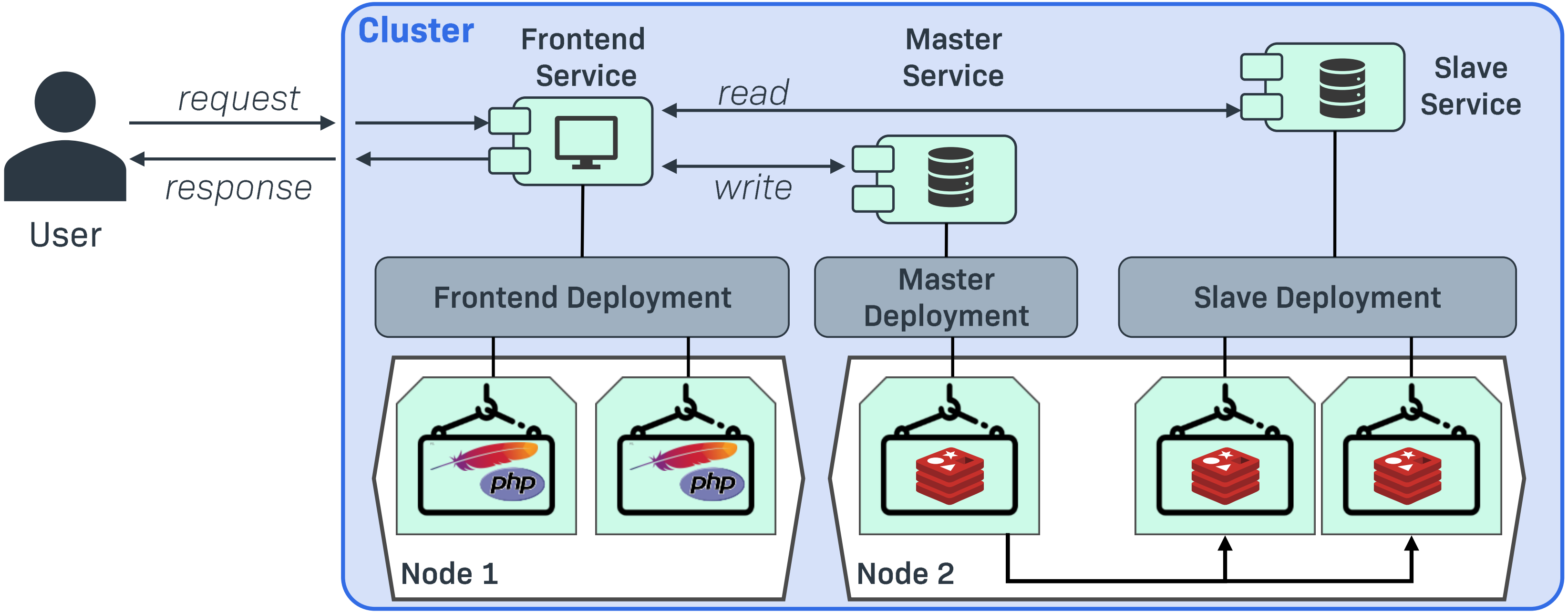


- Microservices serve as communication layer with the outside world
- Lower abstraction layers (pods, deployments) do not interact directly
- Interaction organized through services

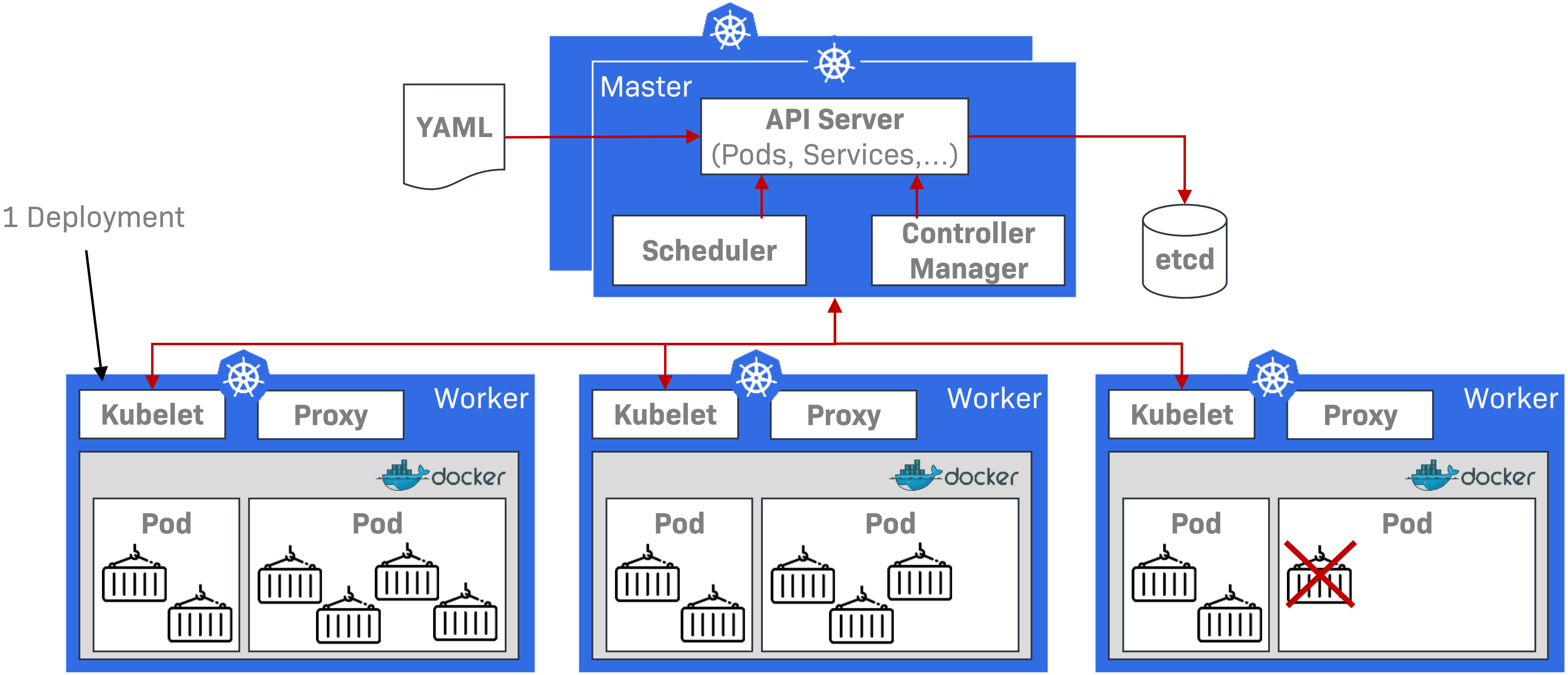
A classical application deployment...



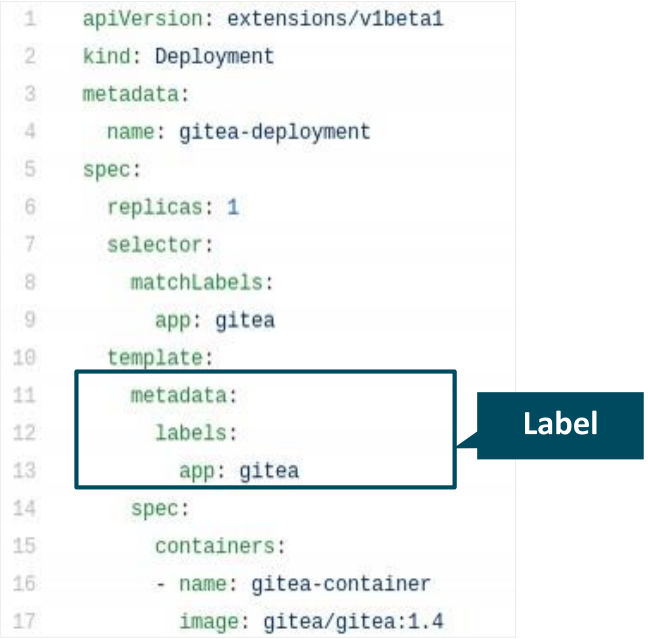
...translated to the Kubernetes world



Reality is often a bit more complex



One more thing: The role of *labels*

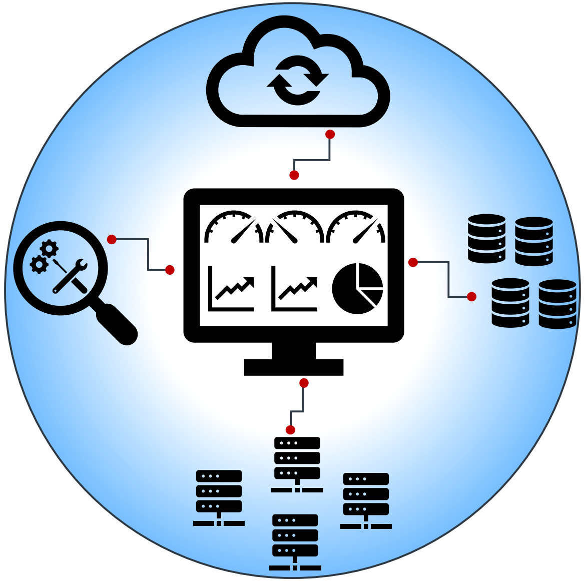


```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: gitea-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gitea
  template:
    metadata:
      labels:
        app: gitea
    spec:
      containers:
      - name: gitea-container
        image: gitea/gitea:1.4
```

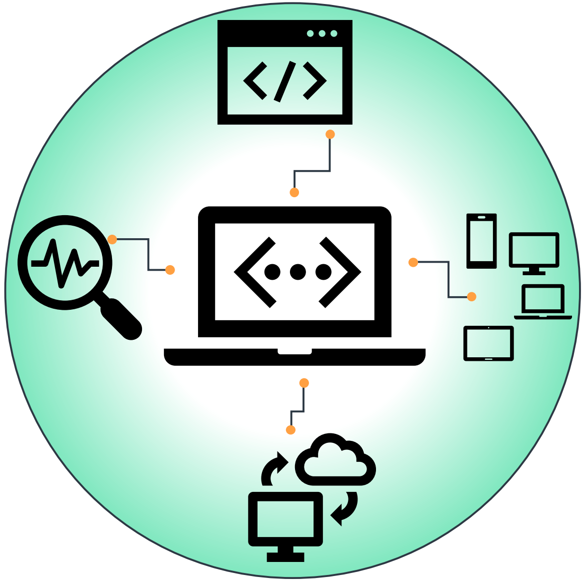
Label

- Used to organize and select subsets of objects
- User-defined key-value pairs associated with Kubernetes resources
- Certain labels automatically applied by Kubernetes

Two views – the admin and the developer view

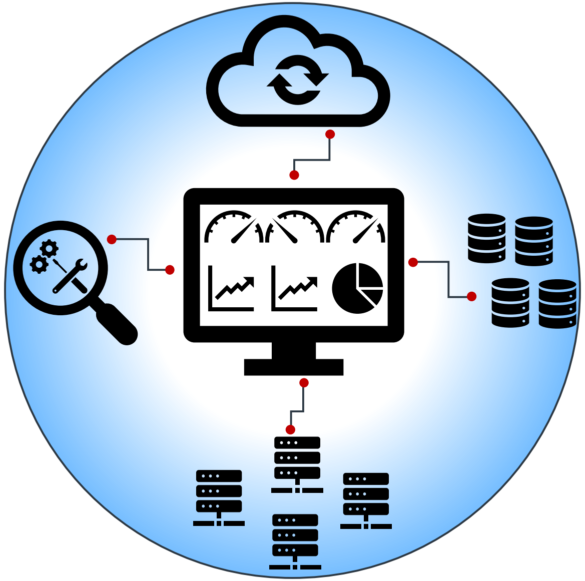


Admin view



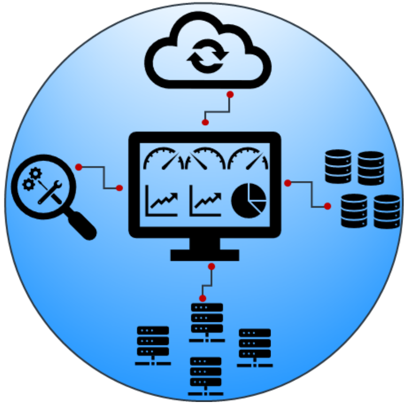
Developer view

Admin view: Is everything working, can apps run?

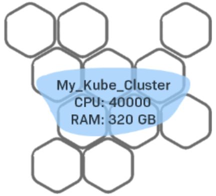


- Typical admin view questions:
  - Health of nodes?
  - How is the cluster? More CPU needed?
  - Enough storage?

Admin view: what we monitor

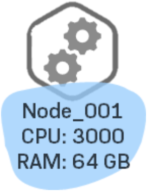


Cluster



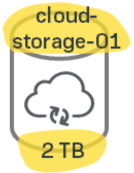
- Components
- CPU
- Memory
- Pods
- Namespaces
- Interfaces
- Roles
- Storage classes
- Filesystems

Node



- Conditions (Ready, Disk Pressure, Memory Pressure)
- CPU
- Pods
- Memory
- Interfaces
- Filesystems

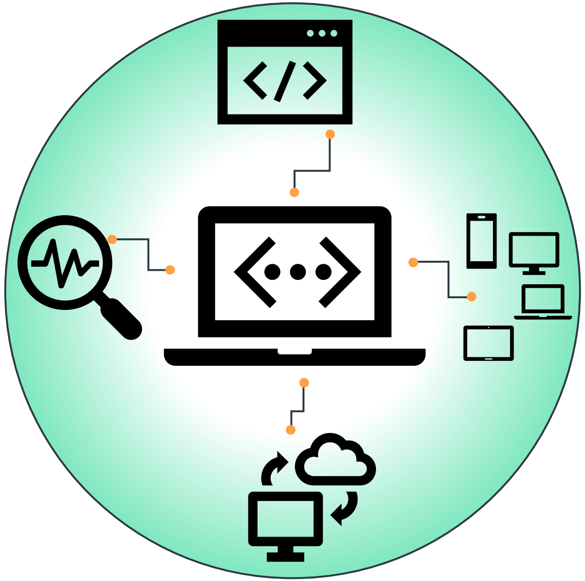
Persistent Volume



- Space Used
- ...

Developer view: Does my application work (well)?

- Typical developer questions:
  - What is the load of my apache/nginx pods?
  - What errors have occurred there?
  - Do my databases have enough space?

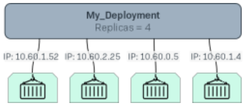


Developer view: what we monitor

Service

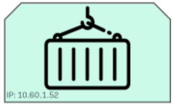
- HW/SW inventory (Cluster IP, Service Type, Load Balancer IP, Pod selectors)
- Used ports

Deployment

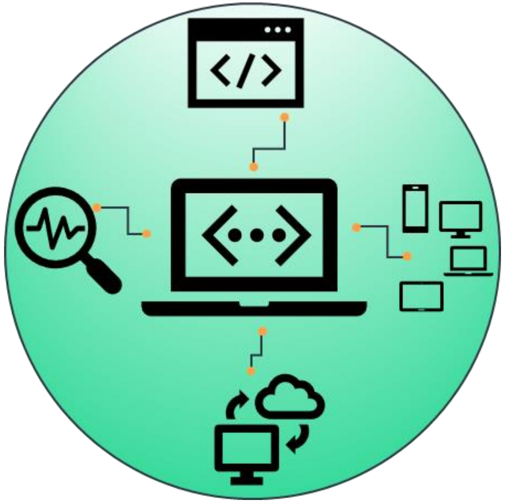


- Replica status (2/4 replicas ready),
- Update strategy (determines thresholds)

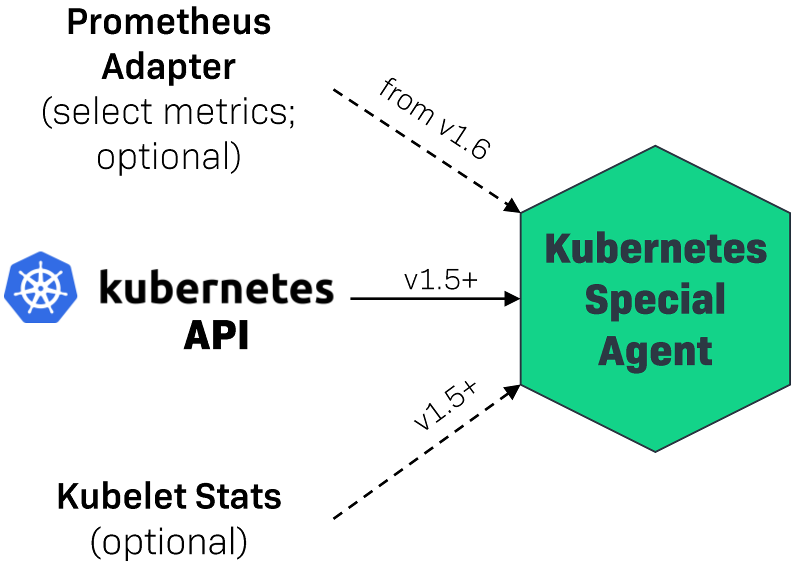
Pod



- CPU
- Memory
- Ready Containers
- HW/SW inventory (Pod: Cluster IP, Pod IP, namespace, node)
- Container: Name, Image, Ready – y/n, Restart count, Image ID, Container ID



We built a special agent addressing both needs


Prometheus Adapter (select metrics; optional)
from v1.6
kubernetes **API**
v1.5+
Kubernetes Special Agent
v1.5+
Kubelet Stats (optional)

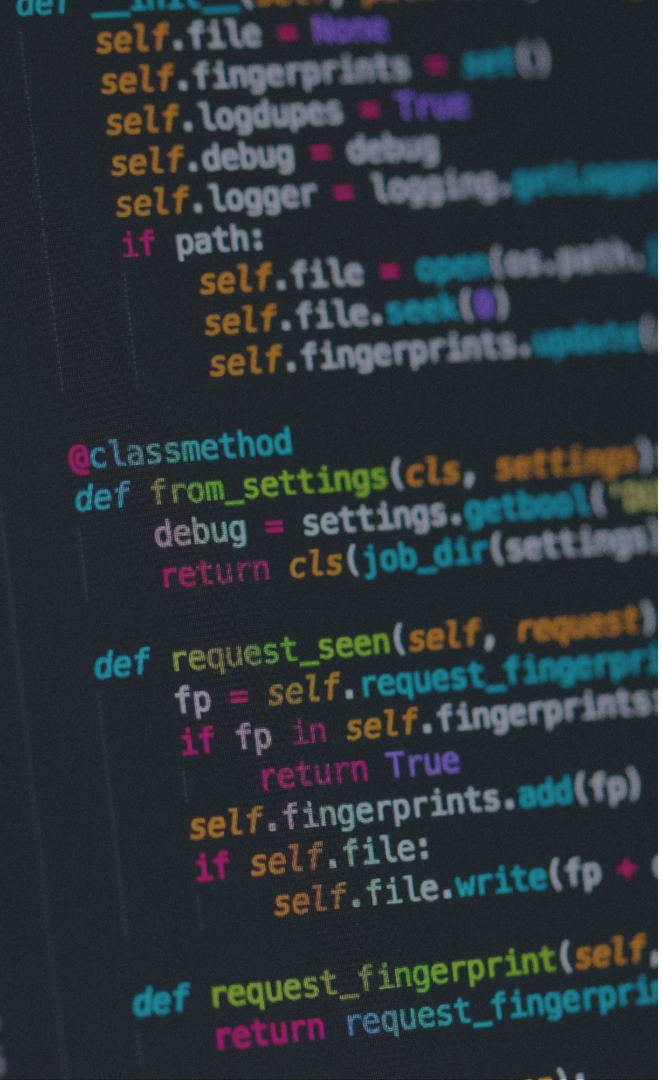
- Kubernetes special agent allows for various data sources
- Key features implemented with check**mk** 1.5+ release
- Additional features with check**mk** 1.6
  - Prometheus Adapter
  - More metrics from Kubernetes API

KUBERNETES MONITORING

1.5+: Focus on admin view

- Monitoring of the Control Plane and worker nodes
- With check**mk** agent on each node, all data available
  - Network
  - File system
  - Roles
- Many metrics can be monitored through Special Agent
- More to come with check**mk** 1.6 (e.g. Disk I/O)

tribe**29**



1.6: Dev view coming to town

- Enabled by Dynamic Configuration Daemon (DCD)
- DCD allows monitoring of application-specific data
  - Pods
  - Deployments
  - Services
- Additionally a „real“ application view can be created through BI aggregations

Lets see how this works!

SHOWCASE

Agenda

1. DOCKER MONITORING
2. KUBERNETES MONITORING
3. DEMO
4. **OPEN QUESTIONS AND OUTLOOK**

tribe29

GOING FORWARD

What about OpenShift & others?

- Kubernetes Monitoring with OpenShift possible (1.5p13 – see handbook: …..)
- Will offer full OpenShift monitoring through dedicated Special Agent by 1.6

GOING FORWARD

Topics we are assessing

- Dynamic dashboards
- Dedicated Kubernetes Agent (Deamon Set)
- Dedicated event-driven Kubernetes Connector for DCD
- Event-driven troubleshooting
- Prometheus Scraping / Integration
- Mesos / Docker Swarm
- Best metric strategy incl. aggregation and StatsD

Thank you!

tribe29

tribe29 GmbH
Kellerstraße 29
81667 München
Deutschland

Web — tribe29.com
E-Mail — mail@tribe29.com