Service Orchestrierung mit Apache Mesos

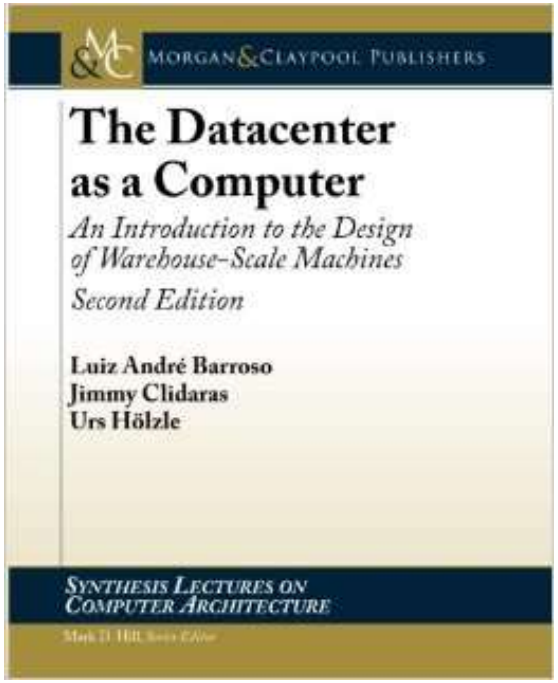
Herausforderungen beim Betrieb von Docker-Containern und Microservice Architekturen

About

- Ralf Ernst
- Senior IT-Architect im IT-Systemhaus der BA
  - Schwerpunkt Systemarchitektur und Infrastruktur von großen verteilten Systemen
  - davor Software-Architekt, Projektleiter, Lead Developer in verschiedenen JEE-Projekten
  - davor Developer im UNIX Umfeld (C, Perl, sh)
  - davor Developer Mainframes (BS2000)
- Twitter: @ralfernst

Apache Mesos –Open Source DataCenter Computing

- Top-Level Apache Projekt
- Ursprünglich entwickelt in den AmpLabs der UC Berkeley
  - Weiterentwickelt von Twitter und AirBnb
- Verwaltung von Ressourcen eines Hardware-Clusters (Ports, CPU, RAM, Disk, I/Os, …)
- Offenes System - APIs in C++, Java/Scala, Python, Go, Erlang, Haskell
- Hochverfügbar, Skalierung über 10000te Knoten



Google: Systeme sollen nicht automatisiert werden, sondern automatisch funktionieren!

Mesos – User

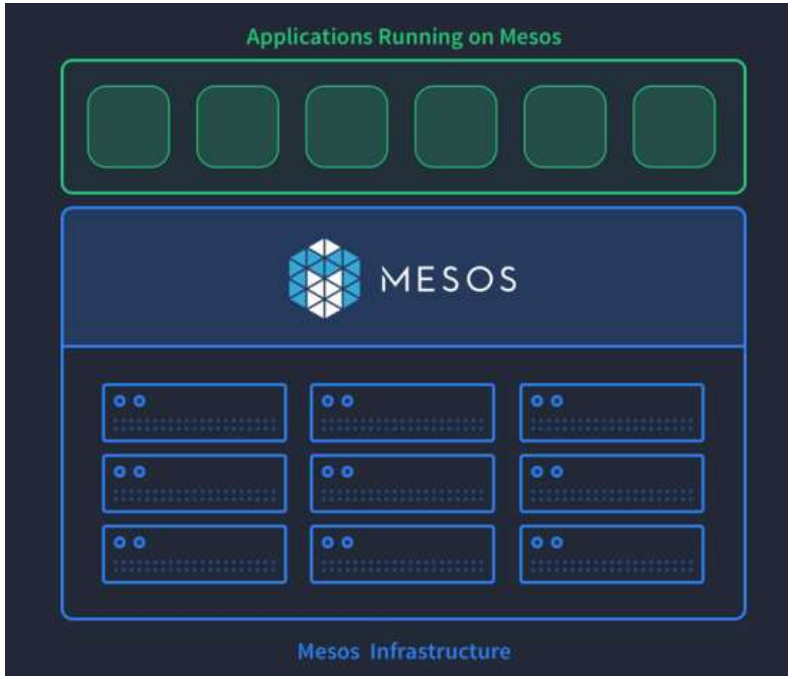
- Airbnb
- Allegro
- Altocloud
- Apple
- Artirix
- Atigeo
- Atlassian
- Auchan Ecommerce France
- Blue Yonder
- Bol.com
- Branding Brand
- Brigade
- Categorize
- CERN
- Cisco
- CloudPhysics
- Conviva
- CorvisaCloud
- Coursera
- CRP-Gabriel Lippmann
- Daemon
- DataMan
- Delivery Hero Holding GmbH
- Devicescape
- DueDil
- eBay
- Ericsson
- Foursquare
- GenOuest
- GoCardless
- Groupon
- GSShop
- Gutefrage.net
- Hootsuite
- HubSpot
- Ignidata
- Jobrapido
- iQIYI
- Learnium
- LIFX
- Linkernetworks
- Localsensor
- Magine TV
- Medallia
- MediaCrossing
- Medidata Solutions
- meemo
- Mesosphere
- Motus
- Netflix
- Oakmore Labs
- Orbitz Worldwide
- OpenCredo
- OpenTable
- Oscar Insurance
- PayPal
- Pinkbike
- ProfitStars
- Qubit
- Qunar
- RelateIQ
- Revisely
- Sabre Labs
- Sailthru
- Scrapinghub
- Sharethrough
- Sigmoid Analytics
- SiQueries
- Sloppy.io
- SmartProcure
- Squarespace
- TellApart
- The Factory
- Time Warner Cable
- Twitter
- Uber
- UCSF
- UC Berkeley
- Udacity
- Uninett AS
- URX
- Viadeo
- Virdata
- Whisk
- Wizcorp
- WooRank
- Yelp
- Yieldbot
- Yodle
- Xiaomi
- Xogito

Mesosphere DC/OS

- Mesosphere DC/OS
  - Seit 19.4.2016 Open-Source http://dcos.io
  - 60 Partner u.a.: Microsoft, HPE, Cisco, Citrix, Accenture, NGINX, Puppet, Chef, EMC, NetApp, Autodesk, …
  - Kommerzielles Angebot von Mesosphere selbst für die Zielgruppe „Fortune 2000“ z.B. Apple, Verizon, PayPal
- DC/OS = PaaS auf Basis von Mesos
  - Package Manager und Installer für CentOS, UI, CLI
  - Universe: App Shop für Mesos Frameworks - One Click Install von Spark, Kafka, Cassandra, Jenkins, ArangoDB, ….
  - Security, Netzzonen, DNS, Load-Balancing, Service-Discovery, Admin-Tools, Monitoring etc.

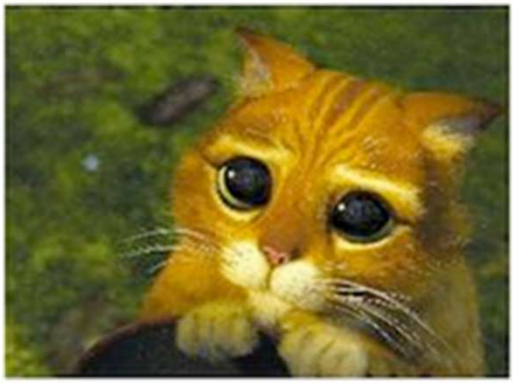
Mesos - Motivation

- Abstraktion der Infrastruktur durch Zusammenfassung aller Ressourcen eines HW-Clusters zu einem „Mainframe“
- Isolation der Ressourcen durch LINUX-Container
  - Docker ist hier nur ein Spezialfall
  - Ressourcen = CPU, RAM, Disk, Ports, …
- Vollkommene Entkopplung zwischen den Applikationen und der Infrastruktur
- Erlaubt klare Rollentrennung zwischen Dev und Ops



„Pets vs. Cattle“ Randy Bias, CloudScaling bekannt durch Vortrag *CERN Data Centre Evolution*

Pets vs. Cattle - CERN Data Centre Evolution

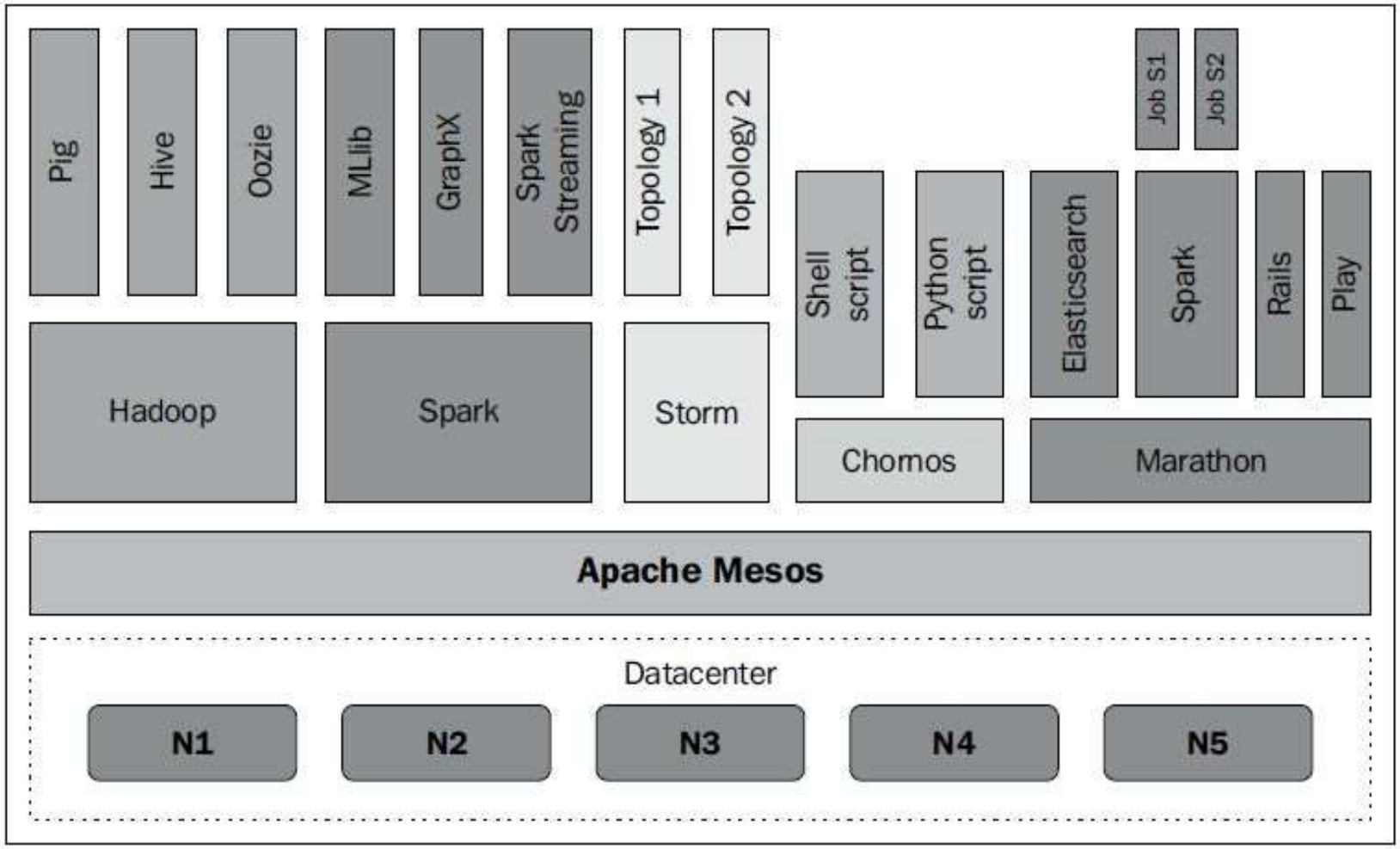


- Pets are given names like pussinboots.cern.ch
- They are unique, lovingly hand raised and cared for
- When they get ill, you nurse them back to health

- Cattle are given numbers like vm0042.cern.ch
- They are almost identical to other cattle
- When they get ill, you get another one

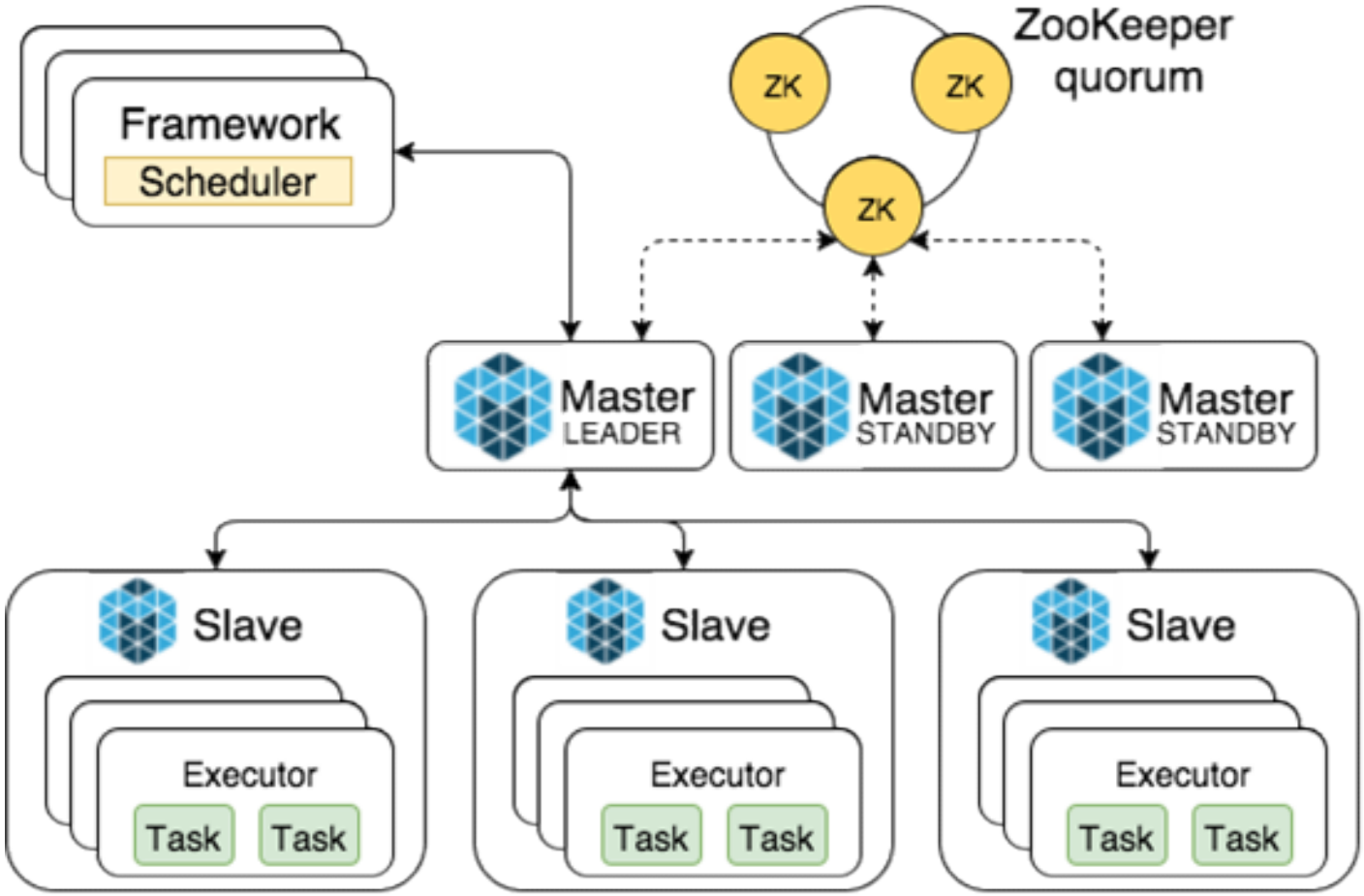
- Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed

Mesos als „DataCenter Kernel“

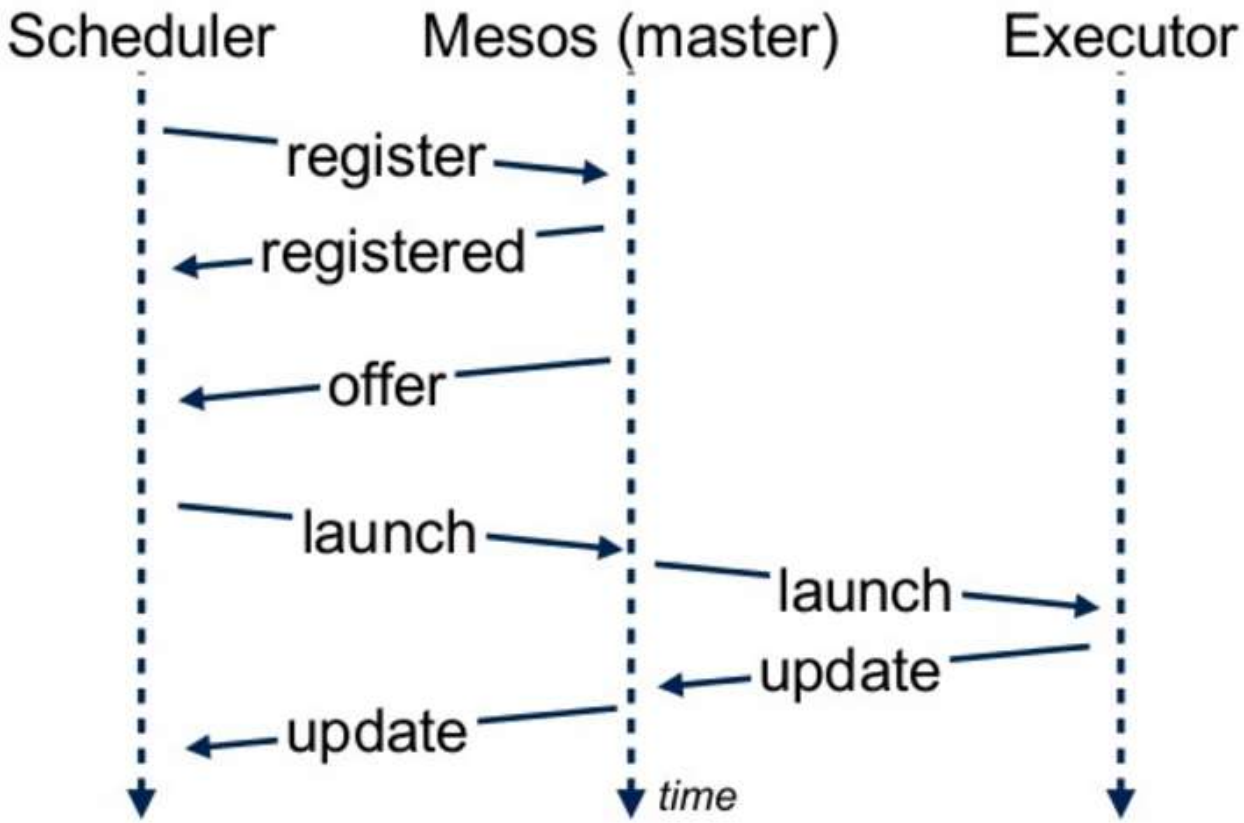


Mesos Frameworks – ein offenes erweiterbares Ökosystem

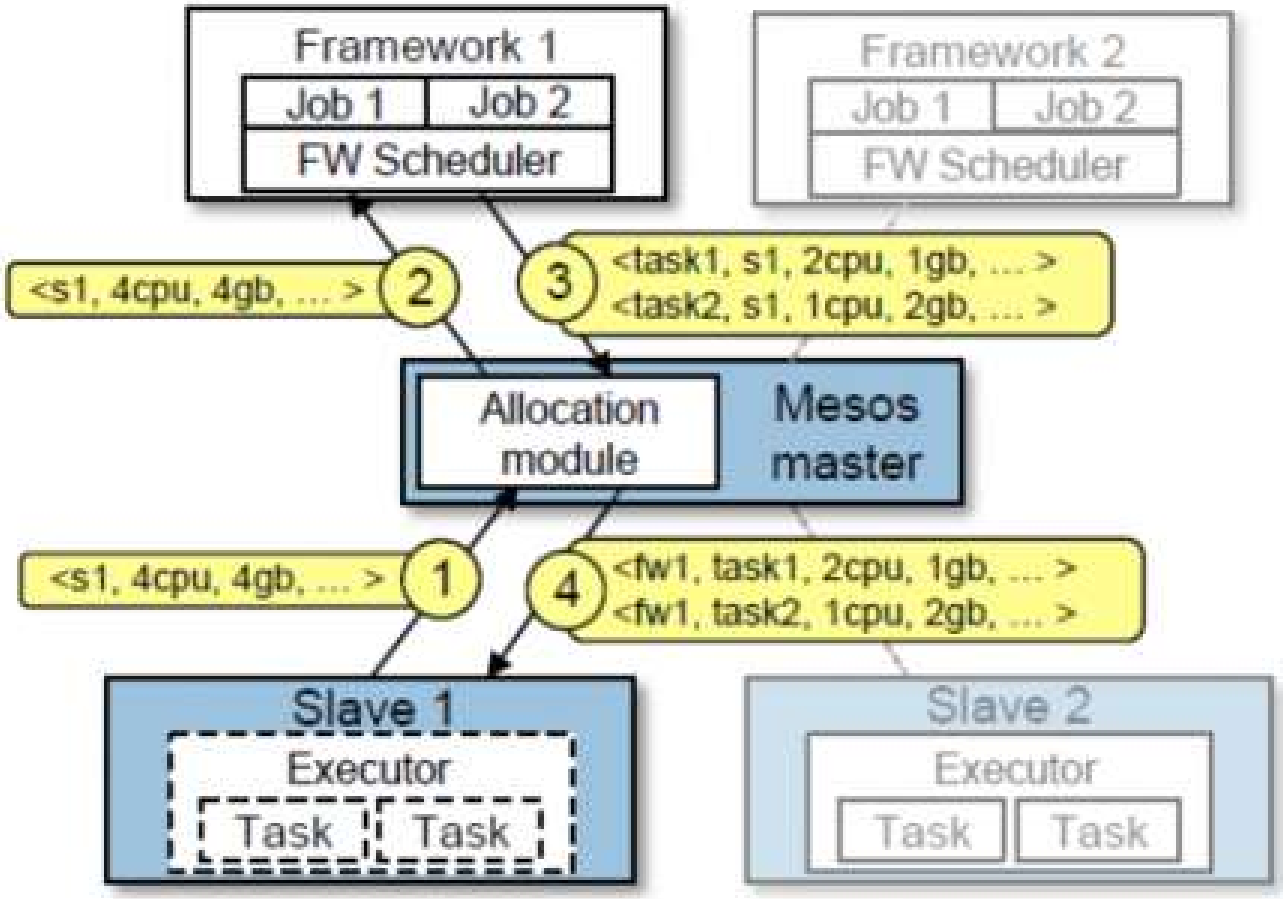
Mesos Architektur



Mesos – 2-level scheduling



Mesos – Beispiel: 2-level scheduling



Marathon – init für Mesos

- Framework für langlaufende Prozesse
- Start, stop, scale, update
- REST-API, WEB-UI zur Übergabe von Parametern
- Hochverfügbar - Leader election über Zookeeper wie Mesos Master
- Docker support, Download der Executables über URIs (tar.gz, zip, etc.)
- Rolling deployments, upgrades, restarts
- Health Checks, Ready Checks
- Artifact Store zur Speicherung von applikationsspezifischen „Secrets“
- Event-Bus für Drittanwendungen z.B. Service Discovery über marathon-lb

Demo Mesos und Marathon

Mesos - Konventionen

- **Ressources** sind z.B. cpu, ram, port und disk und werden vom Mesos Master gemanaged
- **Attribute** sind simple Key-Value Paare, die an die Frameworks übergeben werden
- Ressourcen werden als **Scalars**, **Ranges** und **Sets** gemanaged

```
--resources='cpu:24;mem:24576;disk:409600;ports:[21000-24000];disks:{1,2,3,4,5,6,7,8}'
--attributes='dc:1;floor:2;aisle:6;rack:aa;server:15;os:trusty;jvm:7u68;docker:1.5'
```

- Zuordnung von Ressources zu **Roles**

```
--resources="cpus(*):8; mem(*):15360; disk(*):710534; ports(*):[31000-32000]"
--resources="cpus(prod):8; cpus(stage):2 mem(*):15360; disk(*):710534; ports(*):[31000-32000]"
```

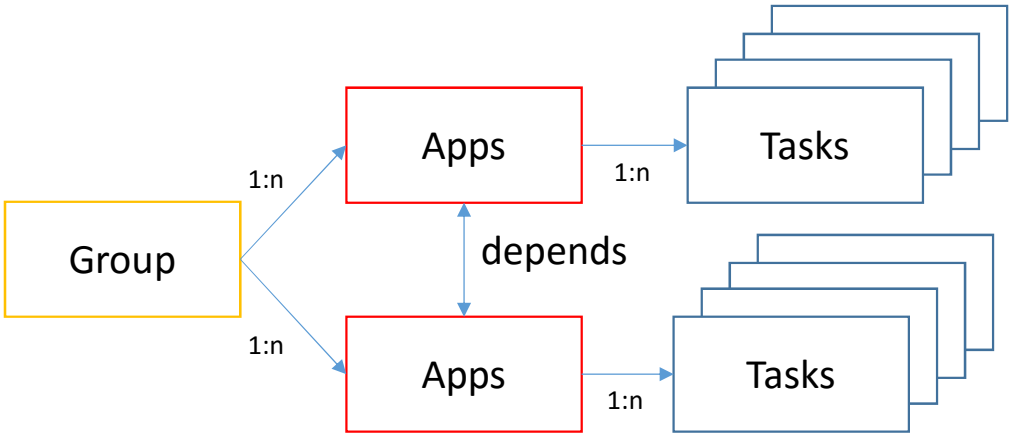
  - Frameworks binden eine bestimmte Rolle oder „any“ → Isolation verschiedener Frameworks möglich
  - Ermöglicht gezielte Überprovisionierung von Ressourcen eines Slaves

Mesos – Advanced Features

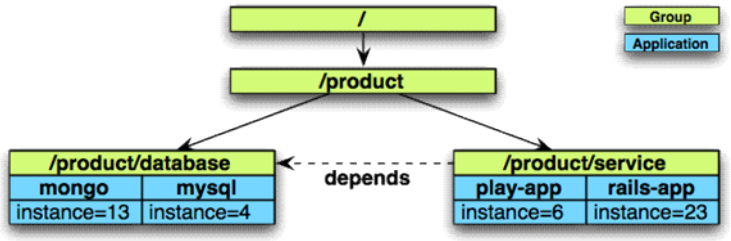
- Constraints: Genau ein Task pro Node, Task auf genau einer Node, …
- Mesos Fetcher – Download von Ressourcen über URIs in die Sandbox eines Tasks (z.B. tar-Archive)
- Oversubscription für „best-effort-Tasks“
- Persistent volumes bzw. multiple disk ressources für Storage
- Quotas für unterschiedliche Frameworks
- Reservations auf Slaves für bestimmte Rollen
- Replicated Logging: Fehlertolerante Log-Replikation des Masters und von Frameworks
- Monitoring: Metriken über REST-API

Marathon – Orchestrierung von Groups

Hierarchie:



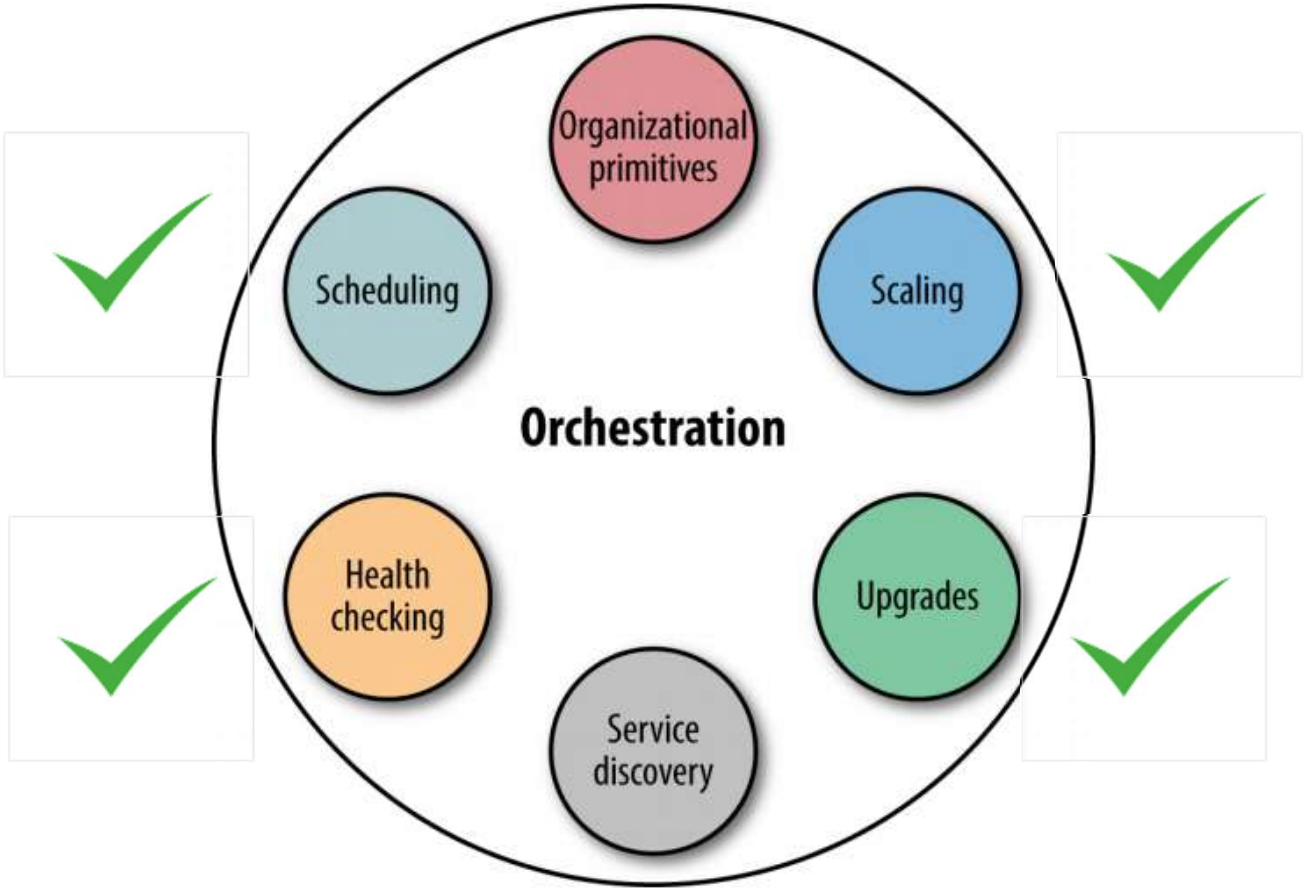
Beispiel:



=

```
{
  "id": "/product",
  "groups": [
    {
      "id": "/product/database",
      "apps": [
        { "id": "/product/mongo", ... },
        { "id": "/product/mysql", ... }
      ]
    },{
      "id": "/product/service",
      "dependencies": ["/product/database"],
      "apps": [
        { "id": "/product/rails-app", ... },
        { "id": "/product/play-app", ... }
      ]
    }
  ]
}
```

Service Orchestrierung



Service Orchestrierung im Reality Check

- Dependency Management
- Service Discovery und Loadbalancing
- Running at scale
- Stateful Services
- Monitoring & Logging

- Aktuell: wenig Standards, viele heterogene (Teil-)Lösungen

Dependency Management – Server

- Die unter Mesos liegenden Maschinen müssen nach wie vor provisioniert und konfiguriert werden
- Allerdings stark vereinfacht, da nur noch wenige Rollen
  - Grundsätzlich: Mesos Master und Mesos Slave Maschinen
  - Zusätzlich: Admin-Nodes, Edge-Nodes, Nodes f. spezielle Mesos-Rollen, …
  - Definition von Port-Ranges zwingend
- Nicht jeder hat die Infrastruktur der „Hyperscaler“
  - Hyperconverged Systems sind ideale Partner z.B. Nutanix, Cisco FlexPod
  - Realität ist anders: heterogene Hardware, vorhandener SAN-Storage, etc.
- Konfigurationsmanagement mit z.B. Ansible, Chef oder Puppet
  - Module für Chef und Puppet existieren

Dependency Management - Applikationen

- Entwickler sollen keine Annahmen über die darunterliegende Infrastruktur machen
  - 12-factor apps
- Geeignete Deploymentformate:
  - Container erleichtern dies wesentlich, das Docker-Format ist etabliert und sehr gut geeignet
  - Lösung für Nicht-Docker-Applikationen: Deployment z.B. als tarball über URIs und Mesos Fetcher in Verbindung mit MesosFetcher Cache
- Lösungen für „secrets“
  - Environment-Variable ???, Shared-FS (URIs!), Marathon Artifacts, Hashicorp Vault, …

Service Discovery und Load-Balancing

- Service Discovery Mechanismen behandeln nur die Erreichbarkeit von Services untereinander, sorgen aber nicht für ein intelligentes Routing
- Service Discovery und Load-Balancing hängen aber eng miteinander zusammen

- Statische Ports
- Dynamische Ports
- Service-Registries

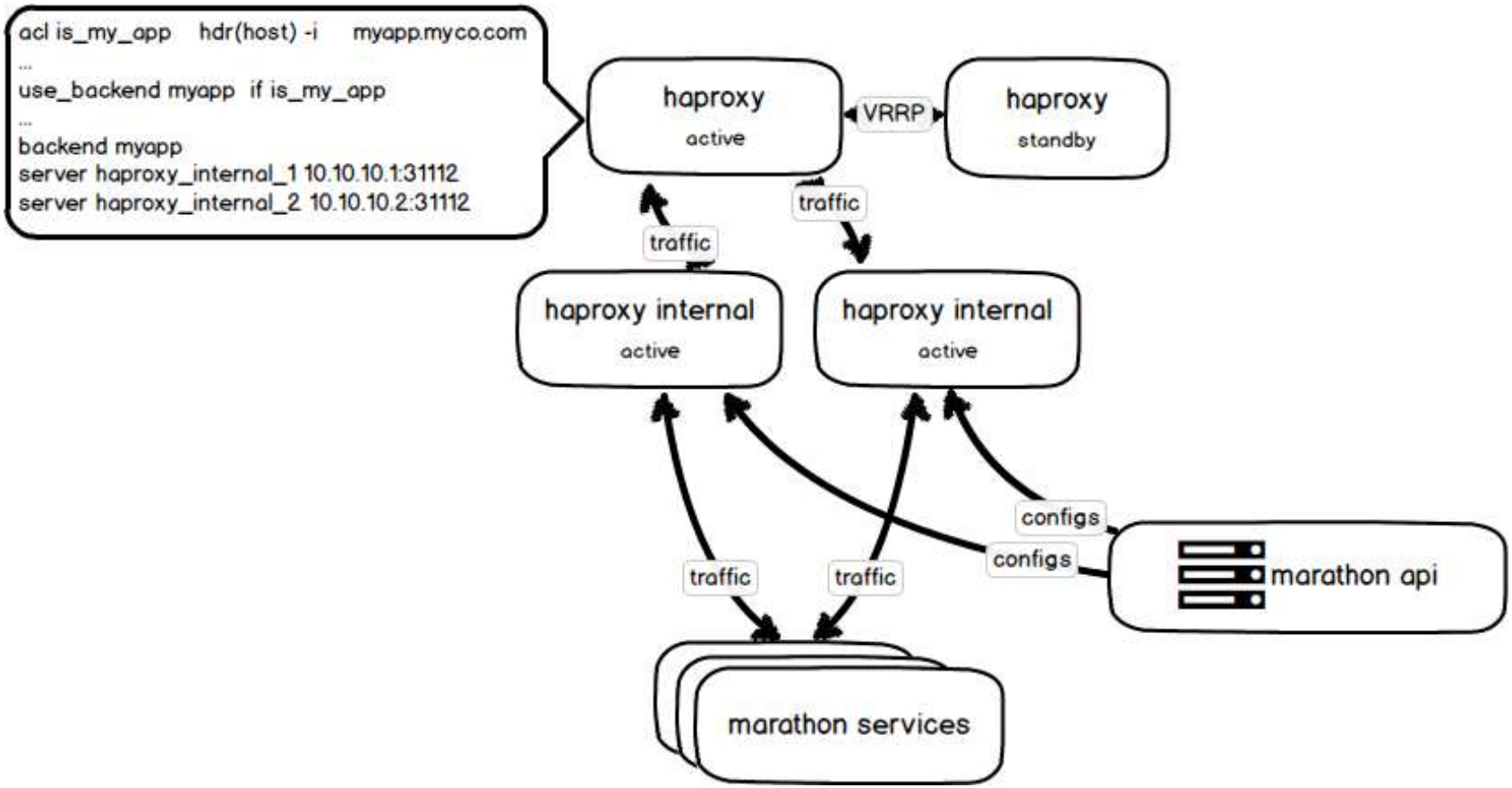
Service Discovery - statische Service Ports

- Jede Instanz eines Services läuft auf einem eigenen Host mit „well-known-Port“
- Sobald mehrere Instanzen auf demselben Host laufen, müssen diese eine eigene IP erhalten
  - Software-Defined Networks (SDNs) wie Flannel, Calico, Weave, …
- Discovery erfolgt typischerweise über DNS-A Records / Service Registry
- Docker, Kubernetes (Flannel) und Mesos (Calico) entwickeln z.Zt. Lösungen in Richtung VIPs
  - Ziel: Hostübergreifende Kommunikation ohne Proxies mit Overlay-Netzwerken

Service Discovery - dynamische Service Ports

- Bridged Network ist in einem Mesos-Cluster ohne SDN die einzig sinnvolle Alternative
  - Ausnahme: Legacy-Applikationen, die feste Host-Ports voraussetzen
- DNS Load-Balancing?
  - SRV Records, langsamer Refresh, DNS-Caches
- Typische Lösungen: Reverse Proxy, dass well-known ports oder Pfade auf dynamische Ports mapped
  - Nginx, HAProxy, …
  - Dynamische Konfiguration mit den aktuellen Daten von z.B. Marathon über dessen Event-Bus: Bamboo, marathon-lb, AirBnb Synapse/Nerve, …
  - Herausforderung: reload des Proxies ohne Downtime bei vielen Routen

Typische Loadbalancing und Service Discovery Topologie mit Marathon und HAProxy

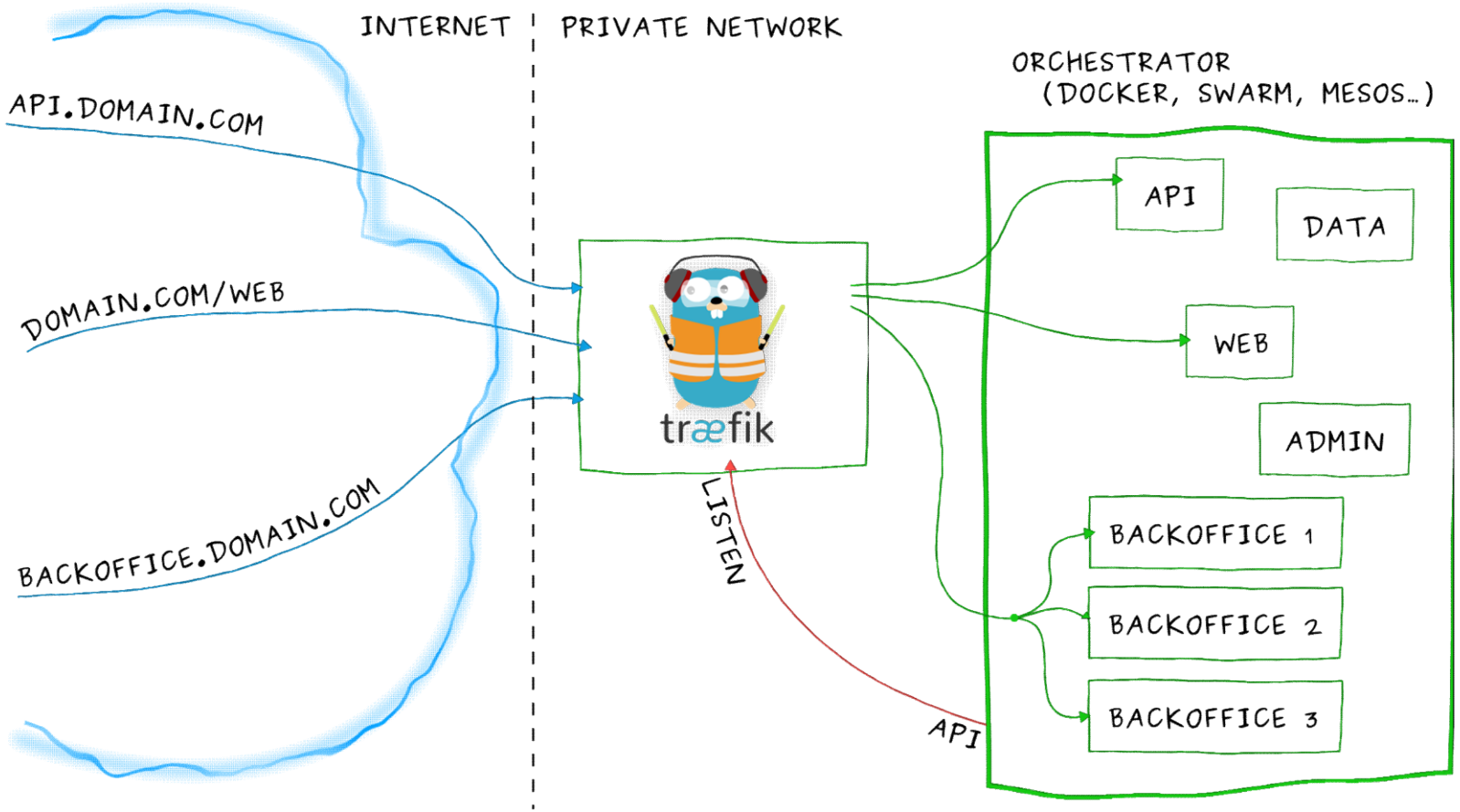


Service Discovery - Service-Registries

- Registries liefern kein Routing!
- Teilweise sehr spezielle Lösungen wie WeaveDNS oder Eureka
- Vielversprechend ist eine Kombination von DNS und Load-Balancing / Routing

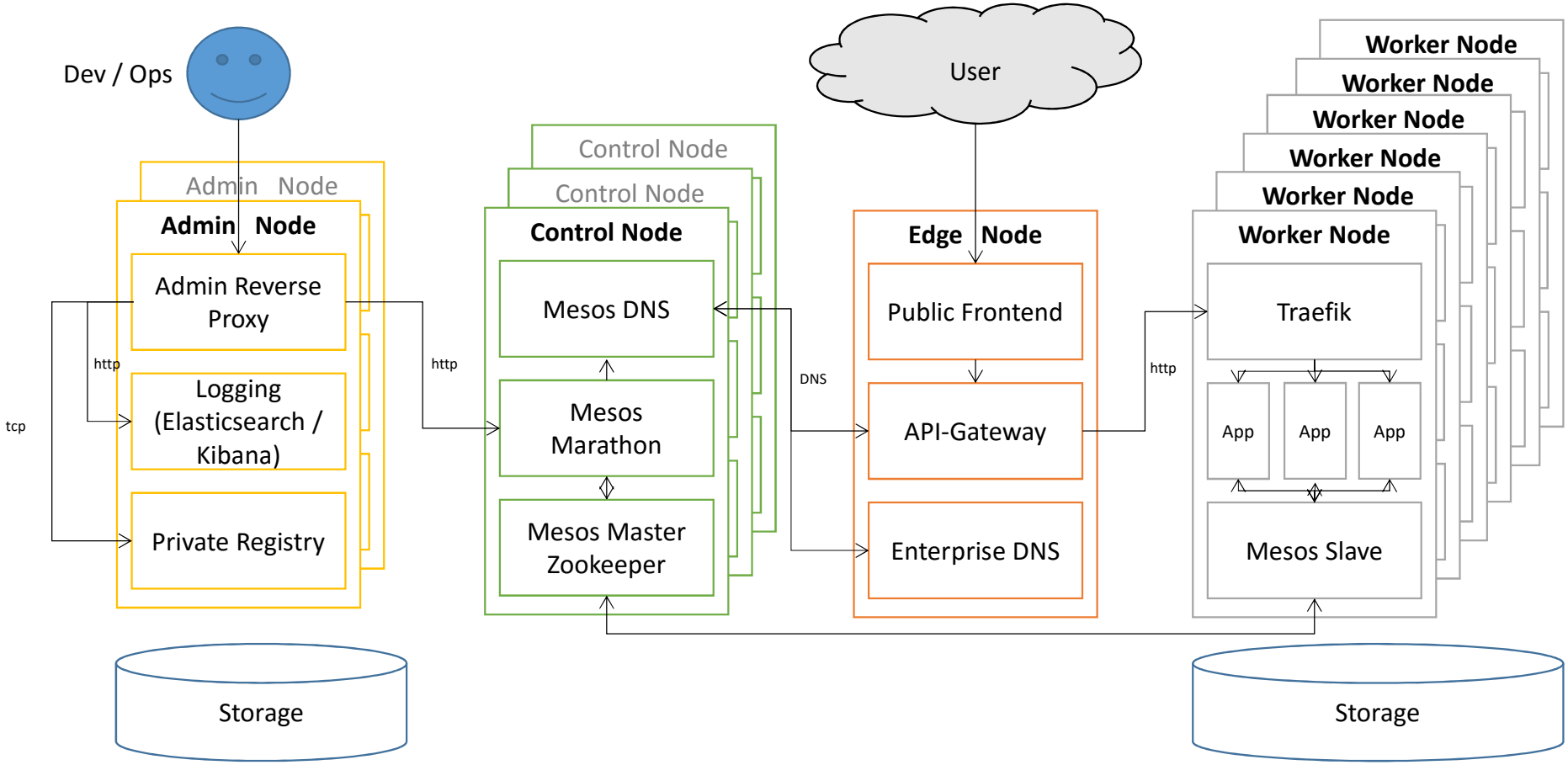
Name	Consistency	Language	Registration	Lookup
ZooKeeper	Strong	Java	Client	Bespoke clients
etcd	Strong	Go	Sidekick+client	HTTP API
Consul	Strong	Go	Automatic and through traefik (Consul backend)	DNS + HTTP/JSON API
Mesos-DNS	Strong	Go	Automatic and through traefik (Marathon back-end)	DNS + HTTP/JSON API
SkyDNS	Strong	Go	Client registration	DNS
WeaveDNS	Strong	Go	Auto	DNS
SmartStack	Strong	Java	Client registration	Automatic through HAProxy config
Eureka	Eventual	Java	Client registration	Bespoke clients

DNS und Loadbalancing - Traefik



Demo Mesos DNS und traefik

Beispiel für eine System- / Deployment-Architektur



Running at scale

- Private Registry

- Isolierung verschiedener Projekte voneinander
  - Pragmatische Lösung: verschiedene Registries…
- Hochverfügbar auslegen!
- Hohe Anforderungen an Netzwerk und das darunterliegende File-System
  - HDFS, Clustered File-Systeme
- Base Images standardisieren, kleine Images verwenden

Running at scale

- Performance

- Container müssen adäquat gesized werden
  - Ressource-Limit überschritten: Container wird gekilled
- Ressourcen sind nicht homogen
  - 1 CPU = 1 Share aber nicht jeder Core ist gleich! Gilt auch für RAM
- Noisy neighbours
  - Schlechte Netzwerk-Isolation, viel Kommunikation, Interferenzen
  - Definition von Rollen für Maschinen, um besonders ressourcenhungrige Tasks zu isolieren

Running at scale - Hochverfügbarkeit

- Mesos ist eine gute Grundlage für hochverfügbare Applikationen
- Mesos und Marathon handeln rolling upgrades, canaries, auto-scaling
- Eine Container-basierte Architektur macht eine Applikation aber nicht automatisch resilient
  - Die Verfügbarkeit von Applikationen wird vom Entwickler bestimmt
  - Weiterhin existieren SPOFs: z.B. Docker Dämon
- Failover und Persistierung von Zuständen muss nach wie vor durch die Applikation selbst geschehen
  - Zookeeper ist hierfür ein gutes Werkzeug…

Stateful Services

- Stateful Services werden von Mesos und Marathon unterstützt
- Mesos Frameworks für ArangoDB, Elasticsearch, Cassandra, Hadoop, …
- Local Disk
  - **Persistent Volumes** auf einem Slave bleiben auch nach Beendigung eines Containers erhalten
  - über Metadaten wird ein Container immer wieder auf der selben „richtigen“ Node gestartet
- Shared-FS oder besser Clustered FS
- SAN (Block Storage) wird zunehmend unterstützt (EMC, Netapps, …)
  - Beispiel: EMC rexray Treiber für Block Storage mit Docker
  - Für EMC SAN-Familie, aber z.B. auch für Open Stack Cinder
  - Unterstützung in Mesos: DVDI (Docker Volume Driver Isolator module)

Demo Mesos Persistent Volumes

Monitoring

- Viele Facetten

- Alarmierung, Resilienz
- Fehleranalyse – Logging
- Metriken von Services und Applikationen
- Kapazitätsplanung des gesamten Hardware Clusters

- Pragmatisch bleiben!

Alarmierung, Resilienz

- Gute Unterstützung von Healthchecks und Resilienz durch Marathon
- Automatisierte Restarts
- Neuprovisionierung bei Hardware-Ausfällen
- Healthchecks
  - verpflichtend
  - Je mehr, je besser
  - Ready-state

Logging

- Log-Aggregation zwingend
- Standardisiertes Log-Format
- Getrennte Hardware für Logs, niemals lokale Platten
- Logs sollten Prozess-Ids, ECIDs, UserIds o.ä. enthalten
- Logging nach stdout / stderr
  - Konfiguration der Log-Files durch Ops
  - Typischerweise Umleitung mit logstash
- ELK-Stack zur Auswertung

Metriken von Services, Applikationen, Hardware

- Allocation vs. Utilization
- Monitoring von CPU und RAM unter Last
  - Auswertung der Metriken von Mesos
  - Mesos Konsole, Mesos-UI
- End2End Monitoring
  - Logging von TimeStamps via Javascript
- Regelmäßiges Reporting auf Basis dieser Daten

- Tools: sysdig Monitoring, Prometheus und viele, viele mehr

Demo scale-up und Mesos-UI

Zusammenfassung

- Service-Orchestrierung mit Mesos ist gut möglich und „im großen Stil“ produktionserprobt
- DC/OS liefert hier eine schlüsselfertige Komplettlösung
  - Etablierte Partner wie HP, MS oder Cisco
- Mesos bleibt dabei ein offeneres System als andere Lösungen
- Aktuell sehr hohe Dynamik (Anzahl Releases, etc.)
- Viele Fragen sind offen, viele Lösungen existieren, best-practices bilden sich erst langsam heraus