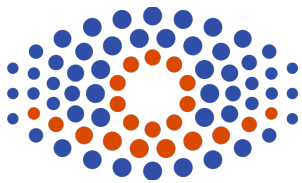
Klein, aber oho – Continuous Delivery von Micro Applications mit Jenkins, Docker & Kubernetes bei Apollo

ContainerConf 2016

15. November 2016

Apollo



Ulrich Häberlein
Teamleitung Backend-Management
Apollo-Optik Holding GmbH & Co KG

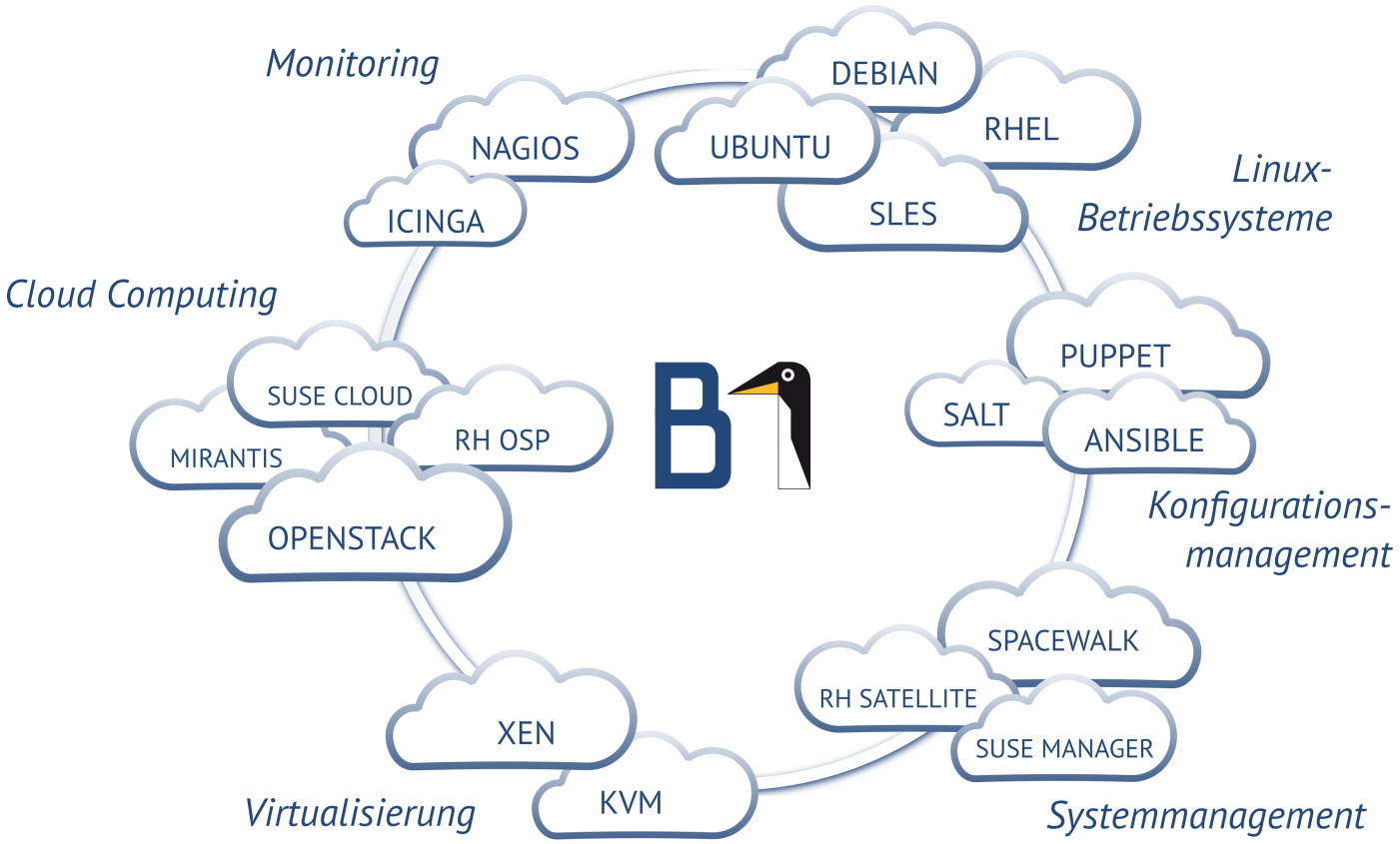
B1 SYSTEMS

Michael Steinfurth
Linux/Unix Consultant & Trainer
B1 Systems GmbH
info@b1-systems.de

Vorstellung B1 Systems

- gegründet 2004
- primär Linux/Open Source-Themen
- national & international tätig
- fast 100 Mitarbeiter
- unabhängig von Soft- und Hardware-Herstellern
- Leistungsangebot:
  - Beratung & Consulting
  - Support
  - Entwicklung
  - Training
  - Betrieb
  - Lösungen
- dezentrale Strukturen

Schwerpunkte

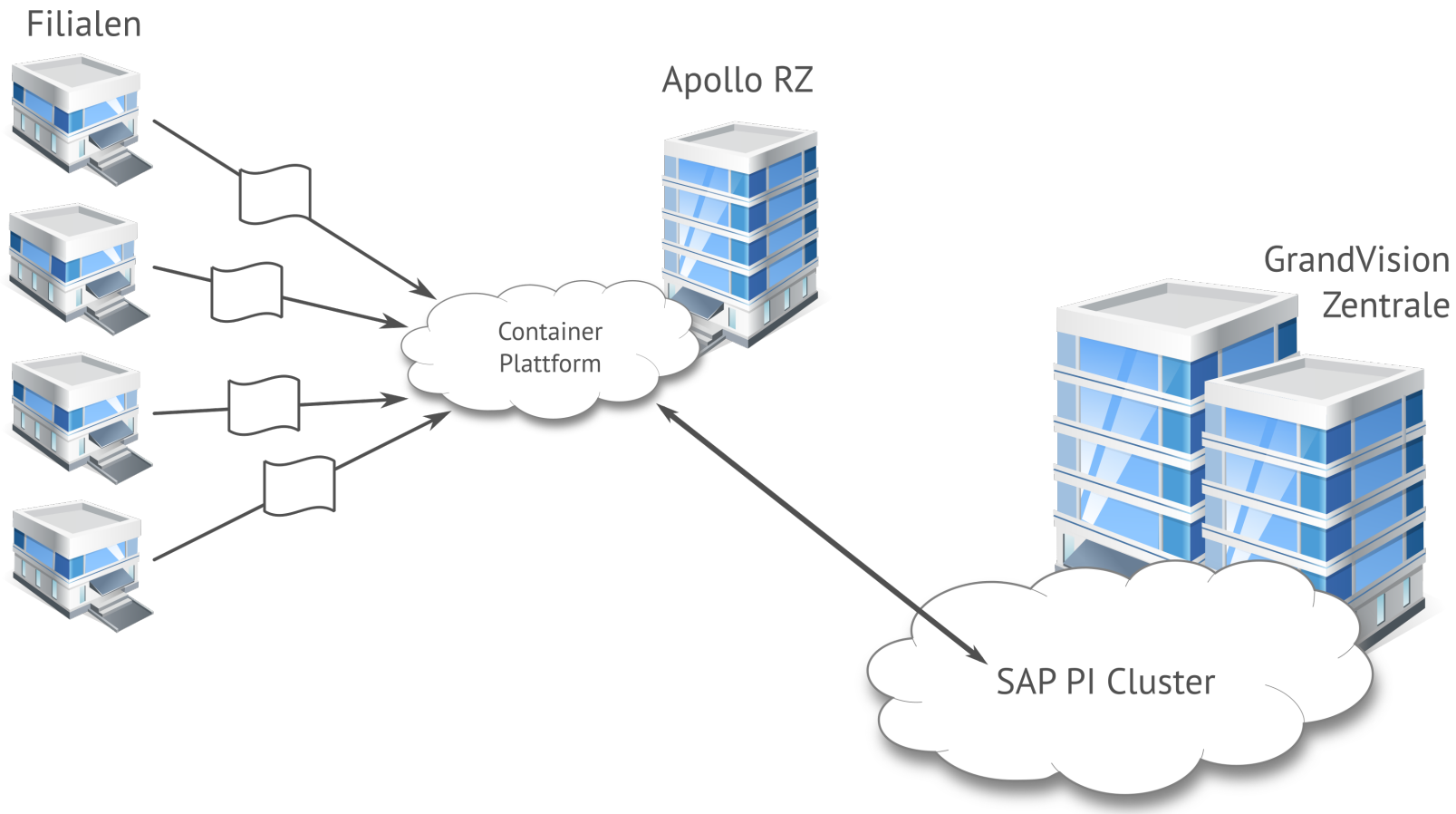


Vorstellung Apollo

- Deutschlands filialstärkster Optiker
  - gegründet 1972
  - über 800 Filialen in Deutschland
  - über 100 Filialen in Österreich
  - mehr als 3.600 Angestellte
- seit 1998 Teil der GrandVision-Gruppe
  - einer der größten Optikkonzerne der Welt
  - über 6000 Filialen in 44 Ländern
  - mehr als 27.000 Angestellte

Business Case

Business Case



Ausgangssituation

- Business-Plattform mit Datenbanken
- 900 Filialen
- Verarbeitung von Bestellungen, Lagerprozessen und Stapelverarbeitung aus den Schnittstellen der POS Systeme
- zentraler SAP Betrieb der GrandVision

Warum Middleware in Microapplication I

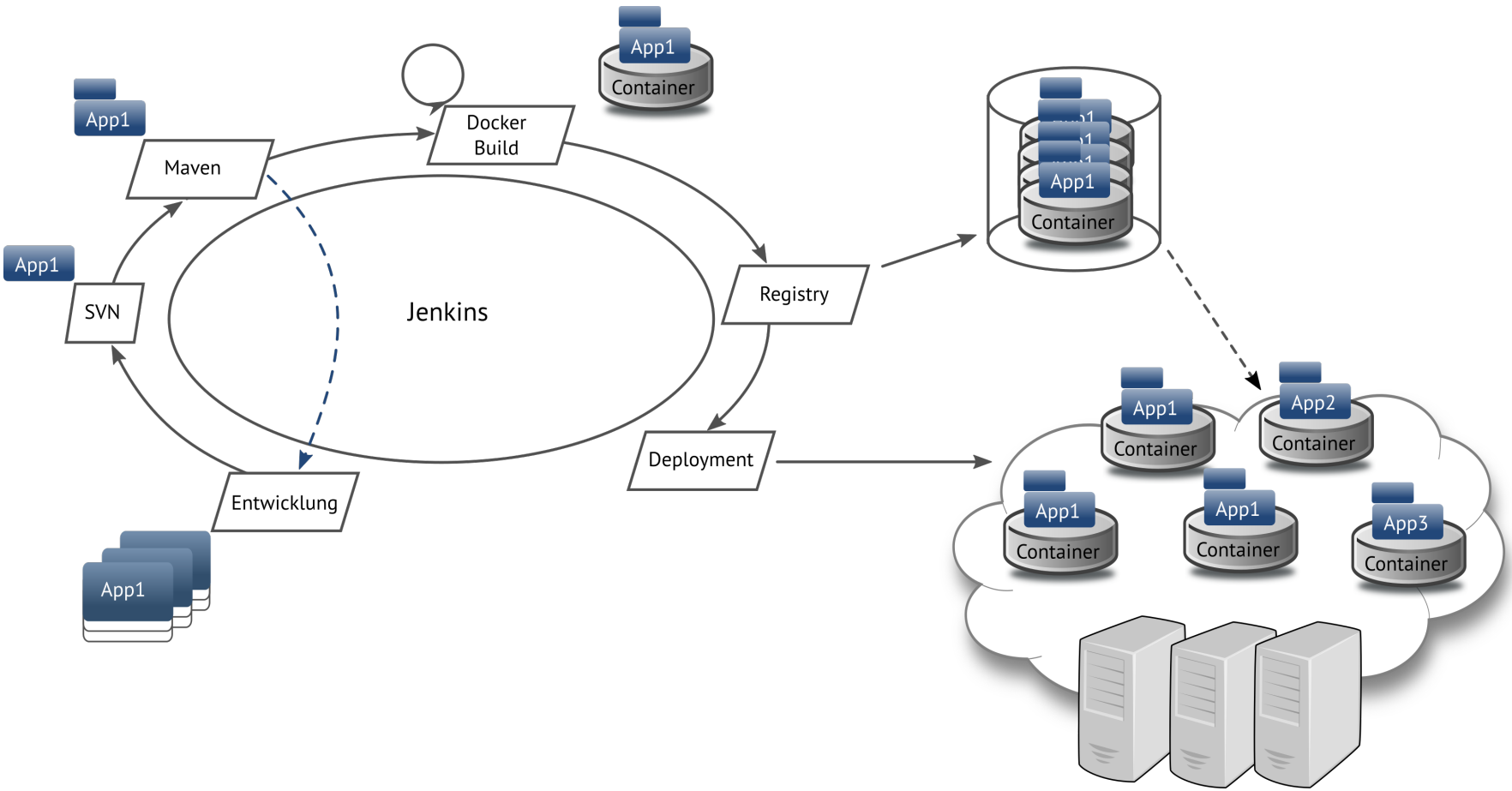
- agile Entwicklung
- schnelle, änderbare Geschäftsfälle
- einfach skalier- und erweiterbar
- kontinuierliche, automatische Updates
- standardisiertes Testmanagement

Warum Middleware in Microapplication II

- Sicherstellung der Deployment-Qualität
- hohe Verfügbarkeit
- betriebssystemunabhängig
- Configuration as Code möglich

Workflow

Workflow


App1
Container
App1
Maven
Docker
Build
App1
SVN
Jenkins
Registry
App1
Container
Deployment
Entwicklung
App1
App1
Container
App2
Container
App1
Container
App1
Container
App3
Container

Jenkinsübernahme

1. Java-Anwendungen werden entwickelt und landen im SVN
2. Jenkins bekommt SVN Änderungen mit, übernimmt Bauprozess
3. maven $\rightarrow$ Bauen mit Abhängigkeiten und Test
4. Übergabe an Microapp-Job

Jenkins-MicroappContainer bauen

1. Applikation verpacken in generischen Container
2. Konfigurationsdateien befüllen
3. Docker Image bauen
4. Metainformationen in Dockerfile (Service $\leftrightarrow$ Ports)
5. Verschieben in die Registry

Jenkins Deployment

1. Jobübergabe
2. Image herunterladen aus Registry (Tag)
3. Erstellen der Metainformationen
4. Anhand von Informationen aus Deployment-NFS und Dockerfile
5. Anlegen Struktur im NFS
6. Anpassen der Softwarezustandsdatei (Konfigurations-NFS)
7. Weiter mit Deployment

Deployment Entwicklungsstufe

- Dockerfile Labels + Konfigurationsinformationen = YAML
- z. B. TAG und Häufigkeit konfiguriert in Konfiguration
- direktes Deployment in Kubernetes Platform pro Container vom Jenkins-Slave

Deployment YAML

Beispiel deployment.yaml Beispiel gekürzt

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: testmicroapplication-deployment
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
  template:
    spec:
      restartPolicy: Always
      containers:
      - name: testmicroapplication
        image: registry/apps/testmicroapplication:B123
        env:
```

Test Staging

- Softwarezustandsdatei in Entwicklung anhand von config-Datei im SVN
- Container Name & Version, Min & Max Häufigkeit, Downtime-Wert
- Jenkins-Job kopiert Zustanddatei in Teststufenbereich im SVN
  ↪ automatisches Erkennen und Herstellen aller Pods (Container)

Beispiel Zustandsdatei

```
testmicroapplication:B123:1:1:1
integration-sample:B42:2:3:1
```

Temporäre Besonderheit

- Anwendungen werden direkt nach Test und Entwicklung ausgerollt
  ↪ schnellerer Entwicklungsfortschritt, da gleichzeitig Weiterentwicklung und Fehlerbehebung aus Tests
- späterer Normalzustand: Herstellung eines Gesamtzustands der Container

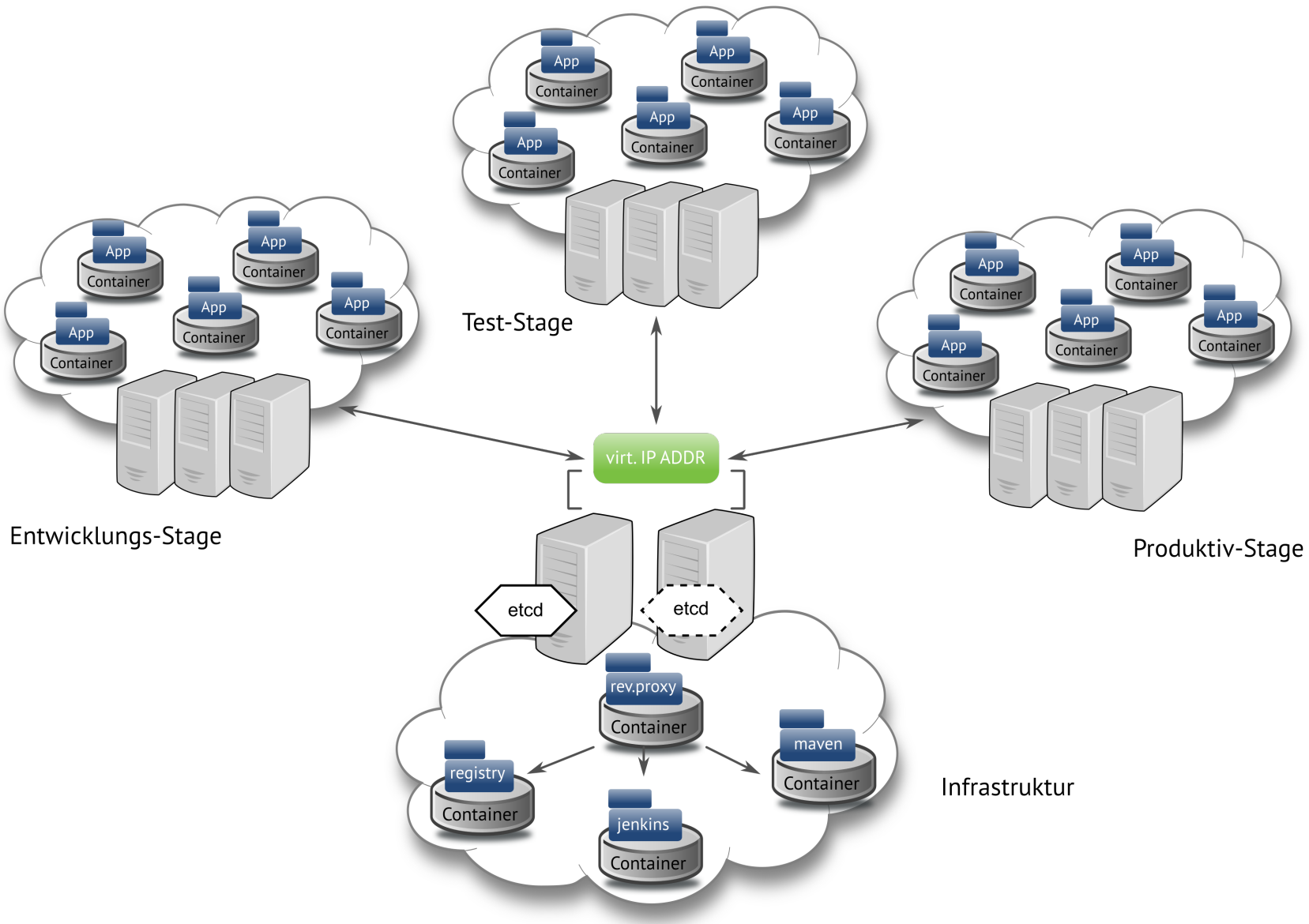
Infrastruktur

Hostsysteme

- VMware virtuelle Maschinen mit SLES 12
- 3 VLANs für jede Stageing Area jeweils /16
- +1 VLAN Cluster Service IP-Adressbereich
- pro Stageing Area geteiltes NFS-Volume
- SUSE Manager Deployment
- Systeme einheitlich $\rightarrow$ Konfiguration eines neuen Systems in fünf Minuten

Kubernetes

- flanneld für virtuelles Netz
- kubernetes Pakete (Master & Worker)
- Master- und Worker-Knoten einheitlich + bessere Verfügbarkeit
- Docker als Backend
- infrastrukturelle Abhängigkeiten ausgelagert (kein Henne-Ei-Problem)

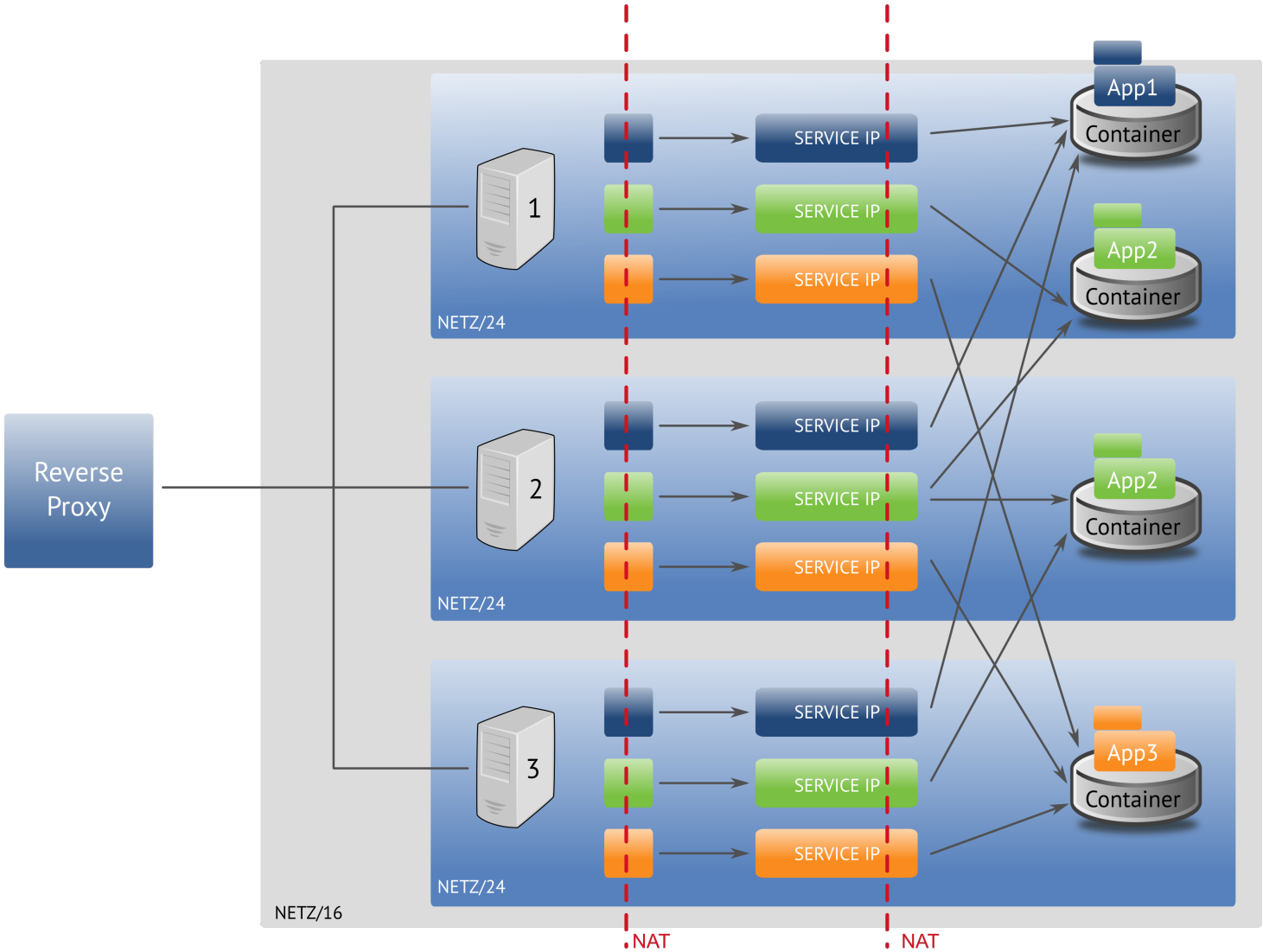

App
Container
Test-Stage
Entwicklungs-Stage
Produktiv-Stage
virt. IP ADDR
etcd
etcd
rev.proxy
Container
registry
Container
maven
Container
jenkins
Container
Infrastruktur

Infrastrukturkomponenten

- Infrastrukturkomponenten auf ausgelagerten Kubernetes-Plattformen
- *Registry*, *Jenkins*, *Maven* durch Container höhere Verfügbarkeit
- *etcd* mit virtueller IP-Adresse und *keepalived* hochverfügbar
- Reverse Proxy mit dynamischer Konfiguration aus Kubernetes-Services für vhosts
- alle Komponenten mit Jenkins bau- und deploybar

Rolling Update & Lastverteilung & Skalierung

- Rolling Updates Replicasets in Deployments
  ↪ keine Downtimes der Applikationen
- Verteilung der Netzwerklast durch Kubernetes-Proxy
  ↪ Iptables-Regeln verteilen unabhängig von eingehender IP-Adresse
- Horizontal Pod Autoscaler
  ↪ konfigurierbare Skalierung bei Leistungsengpässen


Reverse Proxy
1
2
3
SERVICE IP
SERVICE IP
SERVICE IP
SERVICE IP
SERVICE IP
SERVICE IP
SERVICE IP
SERVICE IP
SERVICE IP
App1
Container
App2
Container
App2
Container
App3
Container
NETZ/24
NETZ/24
NETZ/24
NETZ/16
NAT
NAT

Logging

- Logging vorbereitet auf *ELK*-Stack
- Stack selber in Kubernetes mit verteiltem Speicher
- *fluentd* zur Erfassung der Containerlogs
- Applikationen schreiben selbständig Logdateien
- Ausbau der Applikationen zu *ELK*-Anbindung

Monitoring

- Nagios für die grundlegende Systemüberwachung
  Speicher & Basisdienste
- Grafana mit InfluxDB für die Performancegraphen von Clustern und Containern

Erfahrungen & Lessons Learned

Erfahrungen

- Infrastruktur schneller als erwartet
- Faktor 20 schneller zur SAP-Infrastruktur
- Infrastruktur selbst zur Entlastung durch dynamische Jenkins Slaves genutzt
- Umgebung sehr stabil und quasi selbstheilend

Lessons Learned

- *etcd* hochverfügbar machen ohne Henne-Ei-Problem und weniger als drei Container
- *rabbitmq* Message Broker im Cluster anfällig für Aktualisierungen (Erlang Version)

Vielen Dank für Ihre Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de
oder +49 (0)8457 - 931096