CeBIT
Open Source

Der Hype um Docker

Was IT-Leiter und Architekten über agiles Arbeiten, DevOps und Containervirtualisierung wissen sollten.

Nils Magnus

CeBIT Open Source Forum 2015,
Hannover, 16. März 2015

Jump on the Bandwagon

inovex

Docker ist Hype! Doch worum geht’s da eigentlich genau?

In erster Linie ist Docker **keine neue Virtualisierungstechnik.**

So richtig wichtig ist auch nicht, wie performant Docker ist.

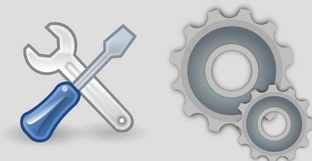
Standardisierung: Docker kann die **Schnittstelle zwischen Development und Operations** vereinfachen.

Unter Umständen Katalysator für echte **DevOps-Prozesse.**

Gretchenfrage: Wie einsatzfähig ist die Technik in der Produktion?

Vorwissen, Zielgruppe:

- IT-Leiter, Systemarchitekten, die schon von Docker gehört haben, aber wissen wollen, ob und wo sich die Technik produktiv einsetzen lässt.



Ziele von IT-Entwicklung und Betrieb

Vier Erfolgsfaktoren beim Einsatz eigener Software:

Schnelligkeit

- Wie schnell lassen sich neue Ideen realisieren (Time to Market)?
- Häufigkeit von Releases

Qualität

- geringe Fehlerrate
- hohe Reaktionsgeschwindigkeit bei Fehlern
- reproduzierbar und nachvollziehbar

Produktorientierung – nicht Technikorientierung

Status auf einen Blick

- Schnelle Rückmeldungen
- Monitoring

DevOps: Anspruch und Wirklichkeit

inovex

Verbesserung in kleinen Schritten und crossfunktionale Teams gibt's leider nicht kostenlos:

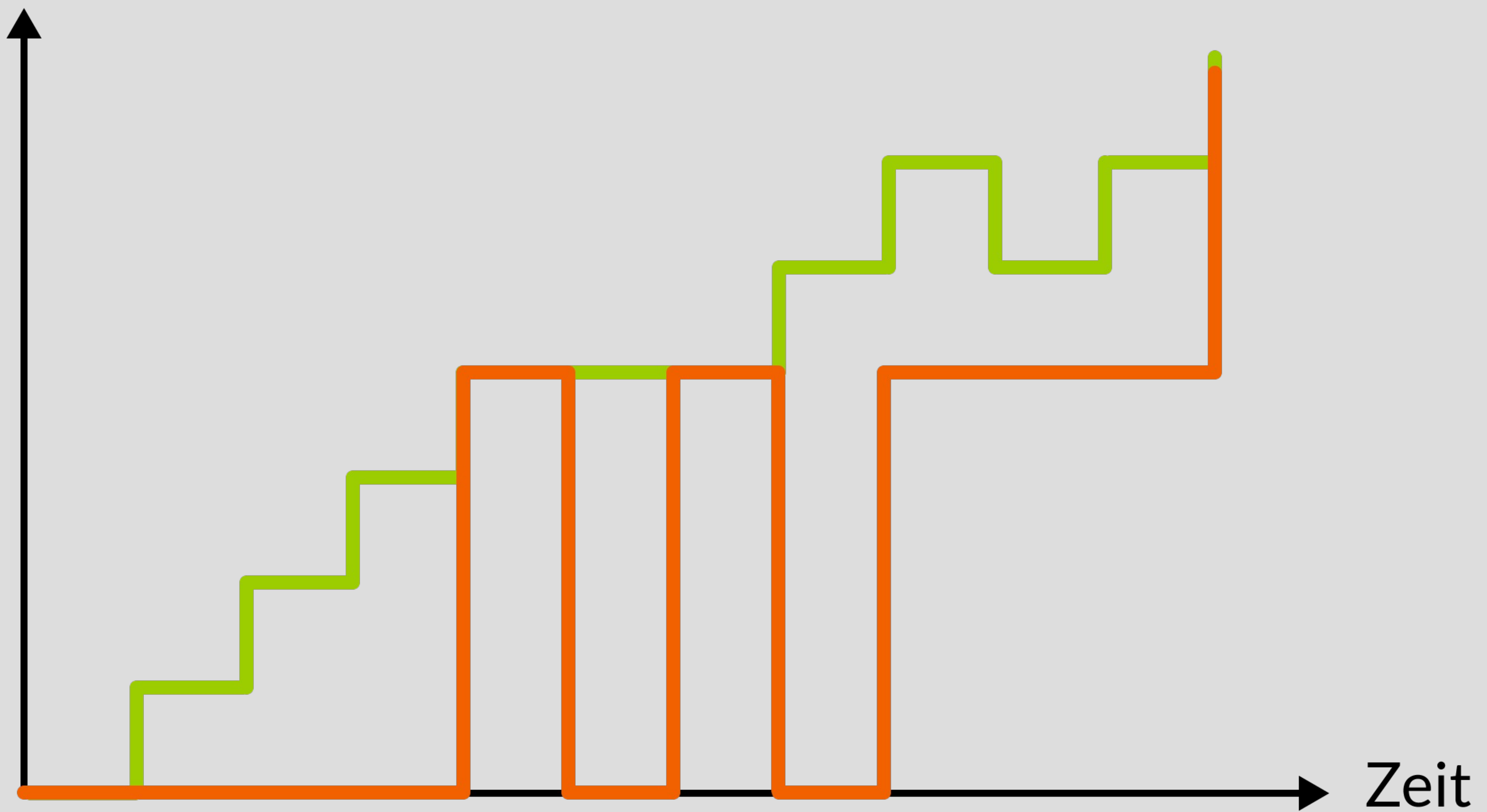
- **Neue Formen der Kommunikation** nötig (Tickets, Boards), Dokumentation unerlässlich.
- **Agile** Zusammenarbeit (zum Beispiel nach Scrum)
- auf **Automatisierung** verlassen (CD und Docker)
- **Kultur** (gemeinsames Commitment notwendig)

Eine gemeinsame Scrum-Schulung reicht dabei nicht aus. **Conways Law** lauert!

Prozesse müssen geübt und gelebt werden!

Schneller mit kleinen Schritten

inovex



Kontinuierliche Entwicklung

inovex

Glückspiel Live-Gang: Schritt von Dev nach Prod ist in der Softwareentwicklung oft fehlerbehaftet und unvorhersagbar

Continous Delivery (CD):

- Automatisierte Auslieferung der Applikation sorgt für Nachvollziehbarkeit
- Häufige kleine Deployments, um Fehler einzugrenzen
- schnelles Feedback (für Entwicklung, Produktmanagement, Operations)

Voraussetzungen:

- Geeignete Softwarearchitektur (z. B. Microservices)
- Crossfunktionale Teams, die gemeinsam Probleme verstehen
- Development und Operations nutzen die gleichen Werkzeuge

Delivery-Pipeline

inovex

Auf dem Weg von der Entwicklung zur produktiven Umgebung durchläuft Software eine Kette:

- Versionskontrolle
- Continuous Integration
  - Build
  - Test
  - Package
- Infrastruktur
- Releases
- Test
- Monitoring

Docker und seine Grundlagen

inovex

Leichtgewichtige **Container**virtualisierung

Es baut ursprünglich auf die Kernelfunktionen

- LXC,
- Namespaces,
- Cgroups und
- Layered Filesystems.

Heute lassen sich fast alle Teilkomponenten beliebig austauschen.

Auch wenn sich Docker wie eine Virtualisierung anfühlt, steht es **nicht im Wettbewerb zu KVM, Xen & Co.**

Vorstellbar wie eine aufgebohrte **Chroot-Umgebung**

Isolation von Ressourcen

Gemeinsame **Kernel**nutzung

Exkurs: Sicherheit im Vergleich

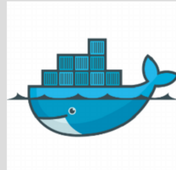
inovex

	Physikalischer Host	Virtuelle Maschine	Container
Gemeinsame Ressourcen	Teilen sich das Netz	Teilen sich die Host-Hardware	Teilen sich den Kernel
Angriffsszenario	Angriff per Netz auf offene Ports etc.	Angriff auf Hypervisor	Angriff per Syscall auf Kernel-Isolation (Namespaces,
Schutzmaß-nahmen	Portfilter, Firewalls, Segmentierung der Netze	Guter Hypervisor	Absicherung im Containermanager, SE-Linux, Capabilitys
Aufwand der Maßnahmen	Einfach, Best Practices	Komplex, aber zentral zu managen	Vielschichtig durch relativ große Angriffsfläche

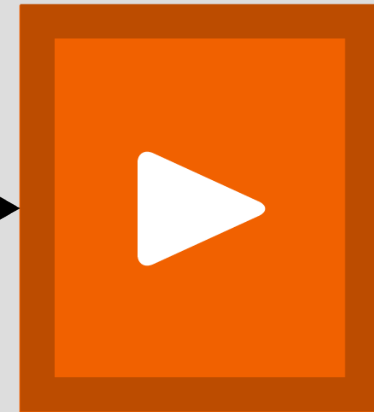
Begriffe im Überblick

inovex

Build nach
Anleitung
aus einem
`Dockerfile`



Starten



Basis-Image
z.B. Ubuntu

Eigenes Image
z.B. Mitglieds-
verwaltung

Container
Laufende
Anwendung

Demo: Image herunterladen und starten

inovex

In der **Docker Registry** liegen schon viele fertige Images.

Einige sind sehr einfach und minimal (**Basisimages**), andere enthalten schon fertig **verpackte Anwendungen**:

```
$ sudo docker search ubuntu
```

Herunterladen eines Images:

```
$ sudo docker pull ubuntu
```

Starten eines Containers:

```
$ sudo docker run -it ubuntu /bin/bash
```

(verlassen mit **exit** oder per Ctrl-p Ctrl-q)

Wie bei jeder Software steht vor dem Einsatz ein kritischer Review, was das Image eigentlich tut (Qualitätssicherung, Security)!

Technische Herausforderungen

inovex

Datentransfer:

- in einen Container hinein- oder herausbekommen
- Mit der Option -v ein Verzeichnis vom Host in den Container hineinmappen

Netzwerk:

- Pragmatischer Default, aber trickreich bei Spezialfällen
- Mapping von Ports des Containers nach außen

Kernel- und Hardwarefunktionen:

- Promiscous Mode, Hostnamen setzen, Device-Zugriff
- Aus Sicherheitsgründen sind per Default nicht aktiv
- Aktivieren ist möglich, aber erfordert tiefes Wissen

Dockerfiles: Bauanleitung für Images

inovex

Je nach Inhalt ist ein typisches **Docker-Image** etwa zwischen 200 MByte und 2 GByte groß.

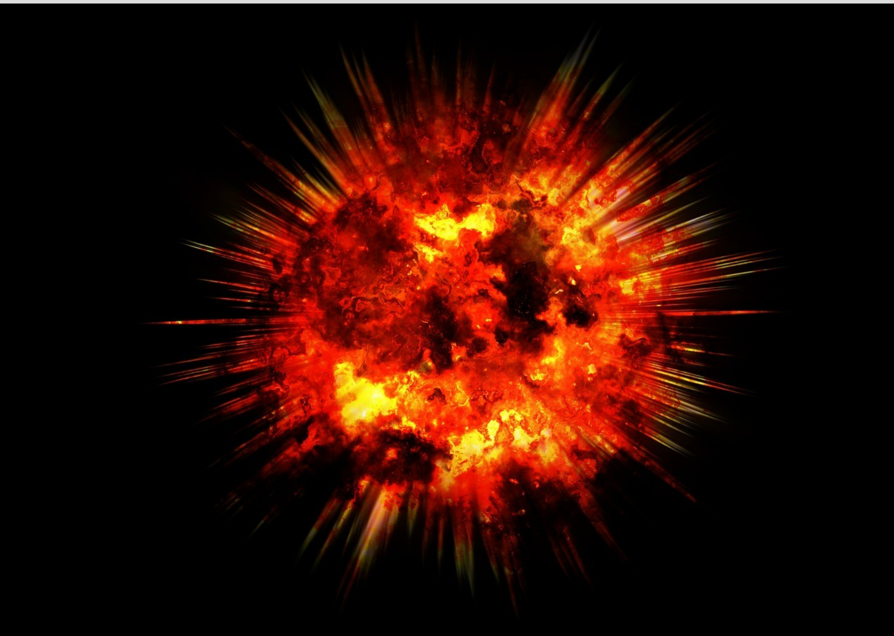
Deutlich kleiner ist die **Anleitung**, wie Docker solche Images selbst baut.

Beispiel für einfachen Webserver auf Basis von Ubuntu:

```
FROM ubuntu:lastest
MAINTAINER Nils Magnus
RUN apt-get update && apt-get-upgrade -yq
RUN apt-get install apache2
CMD service apache2 start && /bin/bash
```

Standardisiert und nachvollziehbar

- Kein Silo-Wissen in den konkreten Installationen (zum Beispiel Handarbeit, manuelle Anpassungen)
- Stattdessen codifiziertes Wissen (Configuration as Code)



Kompletten Server

- schnell
- nachvollziehbar
- von Scratch

aufsetzen (Infrastruktur, Software, Konfiguration)

Container selbst sind „Wegwerfware“

Pet-vs-Cattle-Pattern

inovex

Gerade, wenn es um viele Instanzen (Server, VMs, Container) geht, ist ein Umdenken im Umgang mit Anwendungen nötig:

Pet: Jeder Admin hegt und pflegt seine Server von Hand, kennt jede Besonderheit

Cattle: Der DevOp treibt eine Herde von sich gleichendem Vieh über die Prärie. Keine emotionale Bindung an ein einzelnes Tier/Server

Bei Konfigurationsupdates, Softwareupdates oder Rollbacks:

- nur das Dockerfile konfiguriert den Container
- kein aufwändiges Configdeployment in Container
- evtl. nicht mal ein Konfigurationsmanagement

Alles in Docker?

inovex

Ergibt es Sinn, alle Aufgaben in Docker-Containern abzulegen?

Dockerisieren ist zwar eine gute Übung, aber kein Selbstzweck.

Welchen Sinn hat Infrastruktur in Containern?

- Git-Repository,
- DNS/DHCP,
- Zeitserver, …

Datenbanken in Docker sind umstritten

- Hinreichend I/O zum Teilen vorhanden?
- Wie Backup, Datentransfer und -sharing realisieren?

Docker für Entwickler

inovex

Diesmal **wirklich** eine produktionsnahe Umgebung, die beim Entwickler auf dem Notebook laufen kann.

- Ressourcensparende Umgebung,
- Nicht nur die Anwendung, sondern auch Datenbanken und Webserver sind so abbildbar,
- Ablauf des Deployments wird einfacher,
- Entwickler können selbst Deployments anstoßen und womögliche Probleme selbst erkennen.

Microservices

inovex

Neue Sicht auf eine Anwendungsarchitektur.

Alter Wein (SOA) in neuen Schläuchen, aber immerhin sind die Schläuche neu!

- Anwendungen in deutlich kleinere, autonome Subanwendungen teilen.
- APIs müssen stabil bleiben.
- Sehr schnelles Deployment eines Service ist möglich.
- Das Deployment von Service A ist ohne Auswirkung auf den Service B. Sollte zumindest so sein.

Container unterstützen dieses Pattern optimal.

Docker im Kontext für Services

inovex

Allein ist Docker nur ein kleiner (wenn auch wichtiger) Baustein.

Zu Debuggingzwecken ist ein Container in Sekunden mit einem frischen Sandbox-Betriebssystem bereit.

Als Ablaufumgebung im Produktivbetrieb bleiben viele weitere Fragen:

- **Wo** sind die **Dockerfiles** hinterlegt?
- Welche **Instanz** aktiviert die Container, die zu einer Anwendung gehören?
- Wer kümmert sich um das **Datenrouting** (Loadbalancer)?
- Wer verwaltet **gemeinsame Daten** in einem Cluster?
- Woher weiß ein Container in dieser dynamischen Umgebung, wo er **andere Dienste** findet (Service Discovery)?

Parametrisierung und Orchestrierung

inovex

- Reicht ein Dockerfile alleine aus, um einen realen Dienst vollständig zu beschreiben?
- Kaum, denn Dockerfiles als alleinige Beschreibung eines Softwaredienstes sind natürlich idealisiert.

Parametrisierungen und **Konfiguration** von ...

- Betriebsumgebung (Stage, Prod, …),
- Kontakt zu anderen Komponenten (Datenbank-Connect, externe Dienste, …),
- Softwareversionen,
- anwendungsspezifische Konfigurationen,
- Applikationsdaten (Datenbanken, Dateiverzeichnisse, …) und
- Passwörter sowie Credentials

... sollten nicht innerhalb des Dockerfile stehen.

Ökosystem

inovex

Hostbetriebsystem für Docker:

- CoreOS

Verteilte CMDB:

- etcd

Service-Deployment:

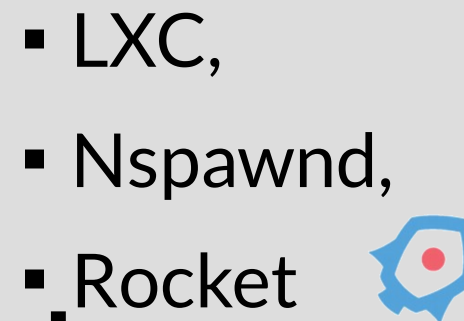
- Systemd
- Fleet
- Swarm

Integrierte Verwaltung:

- Kubernetes,
- Mesos,
- OpenShift,
- Cockpit, …

Alternativen zu Docker:

- LXC,
- Nspawnd,
- Rocket



u. v. a. m.!

Docker im IT-Betrieb

inovex

Technische Reife:

- Funktional (als Container) fast vollständig
- Stabilität und Sicherheit: gut, aber im Fluss
- Benutzbarkeit: sehr gut

Sehr dynamisches **Ökosystem:**

- Unübersichtlich: für viele Aufgaben alternative Tools
- Unklar, welche Ansätze sich am Markt durchsetzen, welche wieder verschwinden oder sich mergen
- Aufwände für Schulung und Einführung entstehen

Wie produktionsreif ist Docker also?

Noch bestehender Nachholbedarf

inovex

Der Einsatzzweck bestimmt die Tauglichkeit:

- ✗ **Abtrennung hochkritischer Daten:**
  Konzeptionell problematisch wegen geteiltem Kernel

- ✗ **Viele gleiche Nodes auf einem Docker-Host (HA):**
  Ergibt nur begrenzt Sinn, weil identische Hardware

- ✗ **Drop-in-Replacement bestehender Anwendungen**

- **Komplexe Applikationen, mehrere Schichten/Netze:**
  im Prinzip ok, wenn Orchestrierung etc. gelöst ist

Einsatzempfehlungen

inovex

Heute schon einsatzfähig in diesen Szenarien:

- ✔ **Testumgebung für Proof-of-Concept-Tests:** problemlose und realitätsnahe Spielwiese
- ✔ **Austauschformat für die Applikationsentwicklung:** mit sanftem Druck zum Continuous Deployment!
- ✔ **Moderne Anwendungen, die neu konzipiert werden**
- ✔ **Geplante, flexible Microservice-Strukturen**

Vielen Dank für Ihre Aufmerksamkeit!

Kontakt

Nils Magnus
Senior Systems Engineer

inovex GmbH
Office München
Valentin-Linhof Str. 2
81829 München

Mobil: +49-173-3181-057
E-Mail: nils.magnus@inovex.de

Sprechen Sie uns an auf unsere Referenzprojekte!

Medienreferenzen

inovex

- „Container 01 KMJ“ von KMJ aus der deutschsprachigen Wikipedia. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons
- inovex GmbH
- „Kuhkuscheln“ am Bergwaldhof
- „Wolpertinger“ von Rainer Zenz - Rainer Zenz. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons
- OpenClips, Pixabay
- Vielen Dank an Alexander Pacnik für seine Anregungen zum Thema Continuous Delivery
- Vielen Dank an Max Wippert für Anmerkungen zum Thema DevOps
- Vielen Dank an Tobias Bayer für redaktionelle Hinweise
- Vielen Dank an Lukas Funk für gestalterische Hinweise