Dockerbank

Container-basiertes Deployment von biomedizinischen IT-Lösungen

Rahmenbedingungen & Best Practices beim Betrieb von Containerlösungen

Dr. med. Thomas Ganslandt[1]

[1] *Universitätsklinikum Erlangen*

Überblick

Aspekte des technischen Betriebs

- Ablage von Nutzdaten
- Backup von Containern und Nutzdaten
- Debugging & Monitoring laufender Container

Konzeptionelle Aspekte

- Benennung und Versionierung
- Architektur der Dienste

Sicherheits- & rechtliche Aspekte

- Images als "Black Box"?
- Lizenzrechtliche Einschränkungen

Ablage von Nutzdaten (1)

TMF

Begriffsbestimmung "Nutzdaten"

- Daten, die im laufenden Betrieb des Containers entstehen und persistent verfügbar sein sollen
  - Datensätze in Datenbanken
  - Dateien mit Nutzdaten
  - Logdateien, die von dauerhafter Relevanz sind (z.B. Audit-Trails)

- nicht gemeint sind dagegen...
  - die eigentliche Software des Services
  - temporäre Dateien von Services des Containers
  - Konfigurationsdateien, die bei der Generierung des Images erzeugt wurden
  - Logdateien von vorübergehender Relevanz

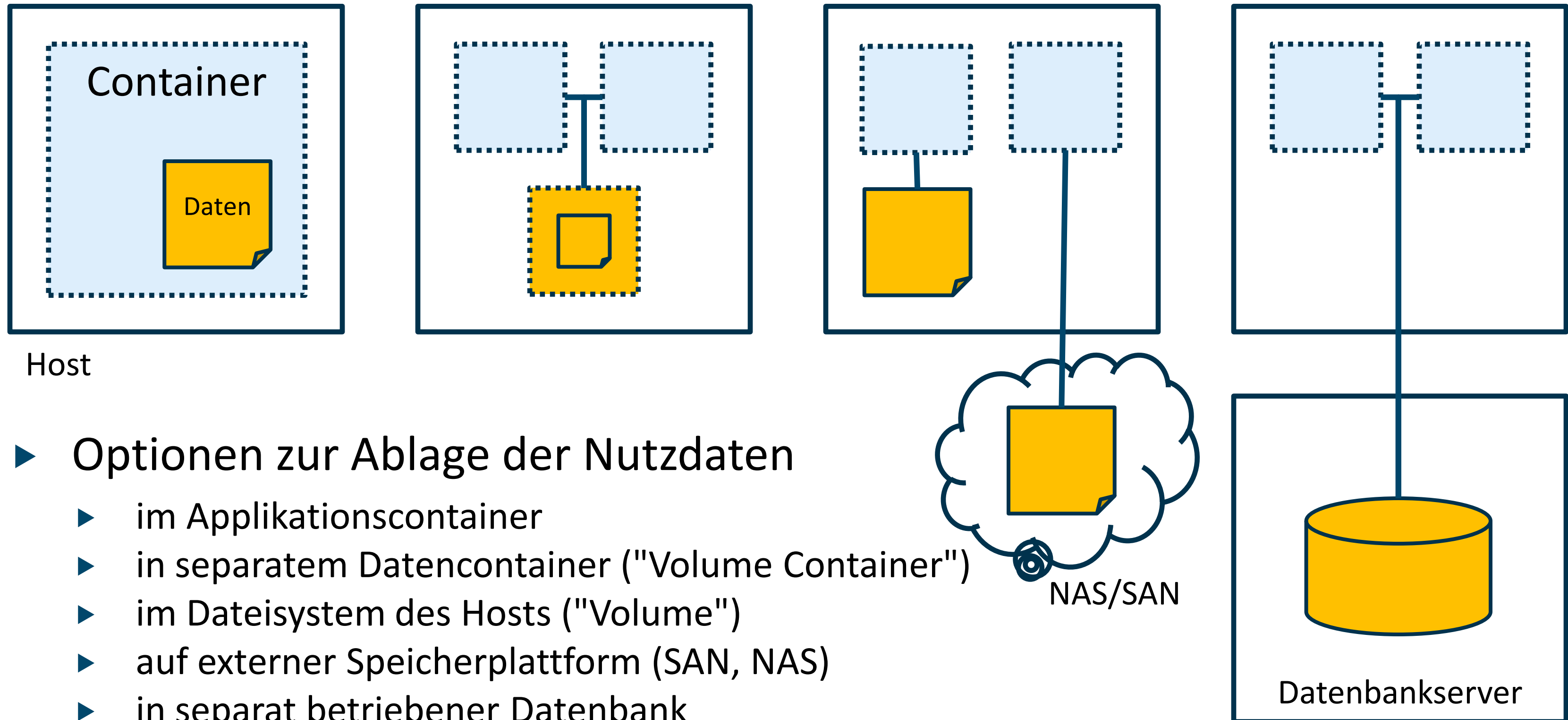
Ablage von Nutzdaten (2)

TMF

Ziel: Trennung von Applikation & Nutzdaten

- Persistenz: Erhaltung der Nutzdaten bei Neuaufsetzen des Containers
- Unabhängigkeit: bei Update/Austausch des Applikationscontainers
- Skalierbarkeit: mehrere App-Container mit gemeinsamen Nutzdaten
- Datenschutz: Hochladen eigener Nutzdaten in Images vermeiden

Ablage von Nutzdaten (3)



- Optionen zur Ablage der Nutzdaten
  - im Applikationscontainer
  - in separatem Datencontainer ("Volume Container")
  - im Dateisystem des Hosts ("Volume")
  - auf externer Speicherplattform (SAN, NAS)
  - in separat betriebener Datenbank

- "Aufräumen" nicht vergessen
  - Datencontainer werden (bewusst) nicht automatisch mit dem App-Container gelöscht
  - Datencontainer sollten deshalb sinnvoll benannt werden

Backup von Containern & Nutzdaten (1)

Notwendigkeit

- Nutzdaten sind ohne Frage zu sichern
- bei den Containern dagegen abzuwägen
  - können aus DockerHub/privatem Repository jederzeit wiederhergestellt werden
  - Deployment sollte dazu inklusive nötiger Konfiguration automatisiert sein
- Disaster Recovery
  - vollständige Sicherung des Hosts (z.B. Image) inkl. enthaltener Container & Nutzdaten
  - ermöglicht schnelle Wiederherstellung des Betriebs nach Totalausfall

Backup von Containern & Nutzdaten (2)

Technische Umsetzung: je nach Ablageort

- reguläre Sicherungsmechanismen
  - für Dateien im Host-Filesystem
  - für Dateien auf externen Speichersystemen (SAN, NAS)

- Docker-spezifische Mechanismen
  - Sichern eines aktiven Datencontainers
    - wird in einen zweiten Container "eingehängt" und von dort in ein .tar-Archiv gesichert
  - Export eines Containers in eine Datei
    - zuvor "Fixierung" des aktuellen Dateistands (commit) erforderlich
  - Pushen eines Images nach DockerHub/privates Repository
    - Cave: versehentlicher Einschluss von Nutz-/Konfigurationsdaten möglich

- Datenkonsistenz beachten
  - wie bei jedem Backup
  - z.B. read-only-Mode einer Datenbank während der Sicherung aktivieren
  - z.B. Nutzung eines SQL-Dumps statt Sicherung der Rohdateien

Debugging & Monitoring laufender Container (1)

Debugging primär für Container-Entwicklung & -Konfiguration

- Kapselung im Container erschwert Zugriff auf laufende Prozesse
- Zugriff von "außen"
  - docker logs: zeigt STDOUT-Meldungen der Applikation im Container
  - docker attach: hängt sich live an den Container und zeigt laufende STDOUT-Meldungen
  - docker inspect: zeigt Konfiguration des Containers, insbesondere Umgebungsvariablen und freigegebene Ports
  - docker top: zeigt laufende Prozesse im Container
  - docker stats: zeigt Übersicht aller laufenden Container
- Zugriff von "innen"
  - docker exec: führt beliebige Befehle im laufenden Container aus
    - z.B. auch eine interaktive Shell, mit der das Dateisystem und interne Logdateien betrachtet sowie weitere Befehle ausgeführt werden können
  - nsenter ("namespace enter"): gibt Shell-Zugriff auf laufenden Container

Debugging & Monitoring laufender Container (2)

Monitoring zur Überwachung "fertiger" Installationen

- Laufzeitverhalten, Ressourcenverbrauch, Stabilität
- Herausforderung durch Vielzahl verschiedener oder auch identischer Container, die auf einem oder mehreren Hosts/Clouds laufen
- manuelles Auslesen von Logs & Statistiken unzureichend

Werkzeuge zur Aggregation & Visualisierung

- CAdvisor: sammelt & visualisiert docker stats-Angaben
- Graylog: fragt STDOUT/-ERR ab & bietet aggregierte Sicht
- Prometheus: Abfrage & Visualisierung verschiedener Daten, Alerting
- Scout, DataDog: kommerzielle Angebote (zentral gehosted)

Größere Management-Plattformen: z.B. Kubernetes

Benennung und Versionierung

Images

- Vergabe von "Tags" zur Kennzeichnung von Versionen
  - z.B. Postgres mit Tag 9.1, 9.2, ... 9.6
  - spezielles Tag "latest" für jeweils aktuellste Version
- erlaubt Einbindung ausgewählter Versionsstände
  - z.B. einer älteren Version, wenn dies für eine Applikation benötigt wird
  - z.B. immer der aktuellsten ("latest") Version
- teilweise auch zur Kennzeichnung von Varianten (z.B. php-apache, php-cli)

Container

- Vergabe eines Namens beim Instanziieren aus dem Image
  - hilft, Container
- Vergabe von "Labels" zur Kennzeichnung von Eigenschaften
  - z.B. für alle Datencontainer, Applikationsserver, ...
  - hilft, Log- & Sensordaten beim Monitoring zu strukturieren

Architektur der Dienste

Kernkonzept Microservices

- Container sollen im Idealfall nur genau einen Dienst (=1 Prozess) anbieten
- Wie soll mit der nötigen Infrastruktur umgegangen werden?
  - z.B. Datenbankserver - würde bei klassischer VM ggf. im gleich Gast installiert
- Nutzung separater "Infrastruktur"-Container
  - z.B. für die Datenbank
  - Infrastruktur kann von mehreren Diensten gemeinsam genutzt oder getrennt je Dienst betrieben werden

Orchestrierung

- Nutzung von docker compose, um alle für einen Dienst nötigen Container parallel aufzusetzen und inkl. Konfiguration zu starten
- Bündelung von Containern zu "pods" in Kubernetes, um komplexe Dienste gemeinsam managen zu können

Sicherheit: Container als "Black Box"?

TMF

Images als Summe vieler Layers

- unklar, wer dazu beigetragen hat
  - wurden aktuelle Releasestände verwendet?
  - wurden Empfehlungen für die sichere Konfiguration beachtet?
  - wurde ggf. absichtlich oder versehentlich Schadsoftware eingeschleust?
- aber: wo liegt der Unterschied zu regulären Softwaredistributionen?
  - werden von größeren Teams zusammengestellt
  - nutzen verschiedenste Infrastrukturpakete und Bibliotheken

Vorteile des Container-Ansatzes

- Kapselung der Dienste innerhalb des Hosts sowie untereinander
  - nicht explizit freigegebene Ports sind automatisch gesperrt
  - kein Zugriff aus dem Container auf Dateien des Hosts oder anderer Container
- zunehmende Verfügbarkeit von "official" Images verbreiteter Tools
- Möglichkeit einer "Kuratierung" biomedizinischer Container

Lizenrechtliche Einschränkungen

Containerisierung kann im Konflikt zu Lizenzen stehen

- z.B. darf der Sourcecode von REDCap nur direkt an Lizenznehmer weitergegeben werden
  - die Veröffentlichung eines REDCap-Images auf DockerHub ist damit unzulässig
  - ein Dockerfile, dass den Container auf Basis lokal vorliegenden Sourcecodes generiert, ist dagegen möglich
- z.B. darf die Testversion der Oracle-Datenbank (XE) nur direkt über die Oracle-Homepage distribuiert werden
  - Images, die offenbar von Privatpersonen erstellt wurden, sind trotzdem auf DockerHub zu finden

Empfehlung

- Lizenzvorgaben bei der Containerisierung einer Applikation beachten
  - betrifft ggf. auch genutzte Infrastrukturdienste
  - Alternativen wie die Bereitstellung von Dockerfiles prüfen
  - ggf. muss auf Produkte verzichtet werden, deren Lizenz inkompatibel ist

TMF

Vielen Dank für Ihre Aufmerksamkeit!

thomas.ganslandt@uk-erlangen.de