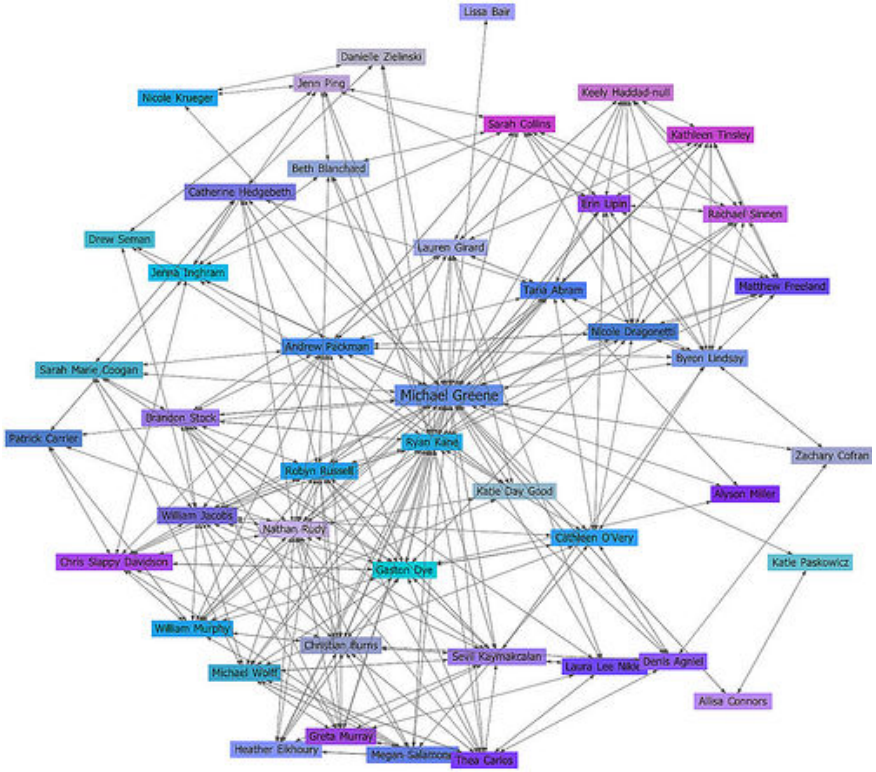
DSSN Praktikum: xodx Simulations-Infrastruktur



Universität Leipzig, Institut für Informatik

Agile Knowledge and Semantic Web Working Group

Markus Ackermann

2. Februar 2014

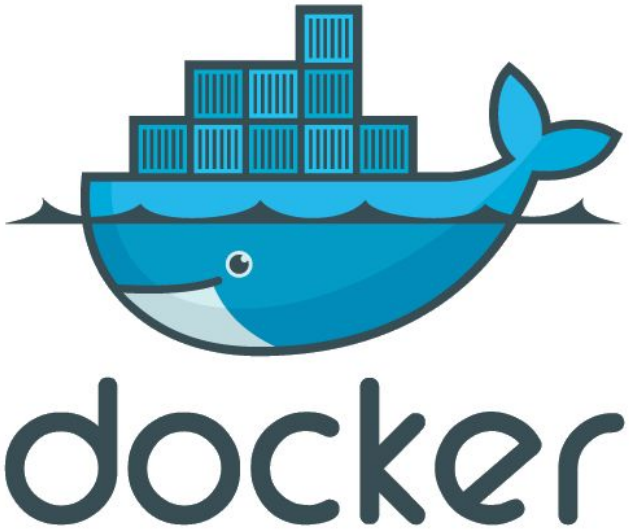
1. Vorhaben und Anforderungen

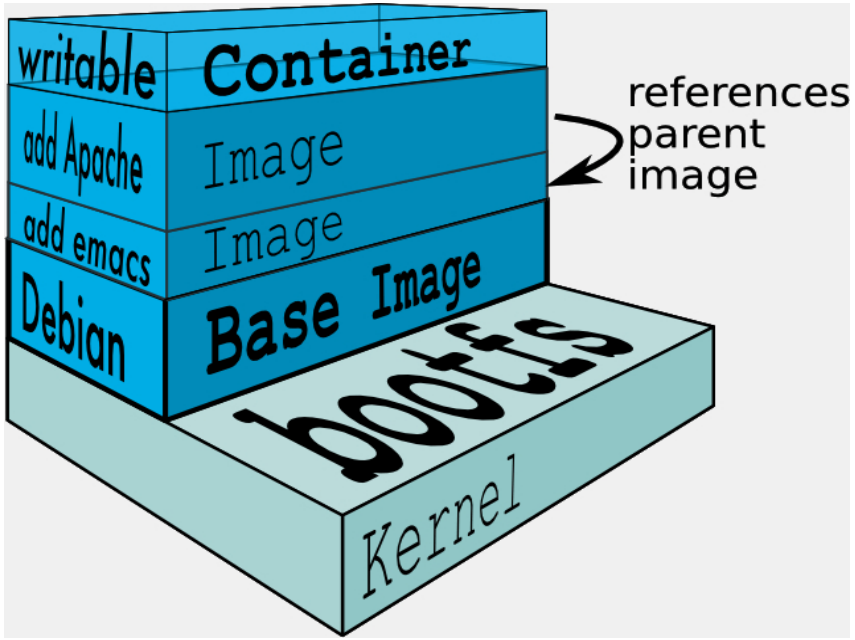
- Simulation des Verhaltens eines DSSN aus xodx-Knoten (und PubSubHubs)
  - Abbilden eigenständiger Netzwerkteilnehmer durch Docker
  - Performance, Lastverträglichkeit und Stabilität gegenwärtiger xodx Implementierung testen
  - ggf. konzeptionelle Probleme beim Betreiben eines größeren DSSN aufdecken

- Anforderungen:
  - möglichst realistisches Nutzerverhalten simulieren (Event Streams von echtem sozialen Netzwerk)
  - Koordination des Fortschreiten der Zeit in Netzwerkteilnehmern (global clock)
  - Logging von Irregularitäten / Programmfehlern in Teilen des DSSN
  - Microservice-Architektur soweit sinnvoll

Docker

- Open Source Projekt zur leichtgewichtigen Simulation Services in isolierten Linux-Umgebungen (Containern)
- eigenes rootfs und eigenständige (virtualisierte) Netzwerkadapter für jeden Container
- alle Container werden auf der Kernel-Instanz des Host Systems ausgeführt (geringerer Overhead)
- Isolation der Container durch LXC (Linux Containers)
  - control groups (resource management)
  - namespace isolation (Prozess-Hierarchie, Netzwerk, eingebundene Dateisysteme)
- Anwendungsgebiete:
  - Sicherstellen einheitlicher Arbeitsumgebungen für Entwickler
  - einfacheres, plattformunabhängieres Deployment
  - verbesserte Fehlertoleranz gegen Fehlfunktion einzelner Services eines Hosts





Docker Konzepte und Paradigma

Host System: vollwertiges Linux-System (mit installiertem Docker Daemon)

- stellt Speicherplatz, Arbeitsspeicher und CPU-Zeit für Container bereit
- verwaltet persistiert die Images und Container und die darin laufenden Prozesse

Image: Abbild eines vollständigen Linux-Dateisystems, einer Umgebungs-konfiguration (Netzwerk-Apater, Umgebungsvariablen,...) und eines Entry-Points

- werden inkrementell entwickelt (Docker Repository für Basis-Images)
- Systemzustand bei der Initialisierung eines Containers

Dockerfile: Skipt-ähnliche Spezifikation zur Generierung eines neuen Images

- Basis-Image, Systemaufrufe, Hinzufügen von externen Dateien, ...

Container: isolierte simulierte Systemumgebung für einen oder mehrere Prozesse

- mehrfache Starts und Stops möglich
- versioniertes Persistieren aller Änderungen an Dateisystem und Umgebung

Volume: spezieller Teilbaum der Dateisystem-Hierarchie in einem Container

- keine Versionierung, gemeinsame Nutzung über verschiedene Container hinweg
- Docker-verwaltete Volumes persistieren, bis kein Container sie nutzt
- alternativ: Teile des Host-FS können unmittelbar gemountet werden

Image Spezifikation (Dockerfile)

```
FROM debian:7.7

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update
RUN apt-get install -y virtuoso-opensource

ENTRYPOINT ["/bin/bash"]

EXPOSE 1111 8990

COPY start.sh /
RUN chmod +x /start.sh
CMD ["/start.sh"]

COPY virtuoso.ini /etc/virtuoso-opensource-6.1/virtuoso.ini
COPY init_virtuoso.sh /
RUN bash /init_virtuoso.sh
```

Image Spezifikation (Startprozess)

```
#!/bin/bash

# shutdown allowing for clean shutdown of daemons
cleanup () {
  echo "stopping nginx..."
  service virtuoso-opensource-6.1 stop

  echo "stopping tail of log..."
  [[ -n $tailpid ]] && kill "$tailpid"

  exit 0
}

trap 'cleanup' INT TERM

service virtuoso-opensource-6.1 start

echo "start finshed, tailing VOS log (press Ctrl + C to shut down this node):"
tail -f /var/lib/virtuoso-opensource-6.1/db/virtuoso.log &
tailpid=$!

wait #on tail
```

Kopplung und Orchestrierung

- Microservices als begünstigtes Architekturmodell
  - isoliere Teilaufgaben eines Informationssystems in möglichst kleine, eigenständige (Web-)Services
  - unabhängige Wahlmöglichkeiten für Plattformen/Technologien pro Microservice
  - unabhängige Aktualisierung einzelner Microservices möglich (außer Schnittstelle) -> auch bei laufenden Betrieb

- zwei prinzipielle Kopplungsmöglichkeiten (Orchestrierung) für Container
  - via Netzwerk-Kommunikation
    - *linking* von virt. Netzwerk-Hosts pro Container (Aliase in `/etc/hosts`)

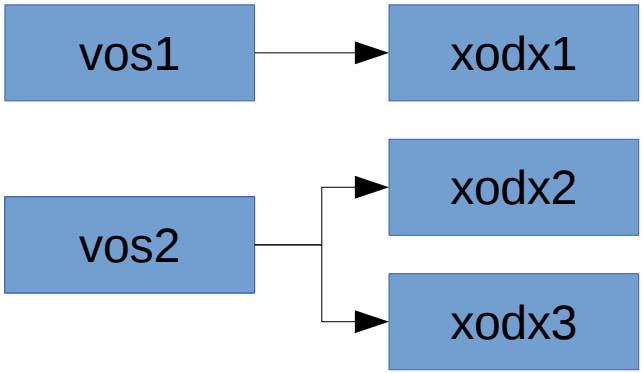
  - über geteilte Volumes
    - effizienter bei großen persistenten Datenmengen (Synchronisation erfoderlich)

Docker Compose (ehem. Fig): Python-Tool zur Orchestrieung von Containern zu grob-granulareren Services
  - Container=Service, Aggregat von Services = Service
  - deklarative Beschreibung von Services und ihrer Verknüpfung in YAML
  - gruppierte Überwachung und Administration aller Container eines Services

Spezifikation aggr. Services (Fig)

```
xodx1:
  image: 'xodx:default'
  links:
    - vos1:vos
  ports:
    - "8001:80"

vos1:
  image: 'vos:lite-True_mem-64'
  mem_limit: 64m
  ports:
    - "1111"
```

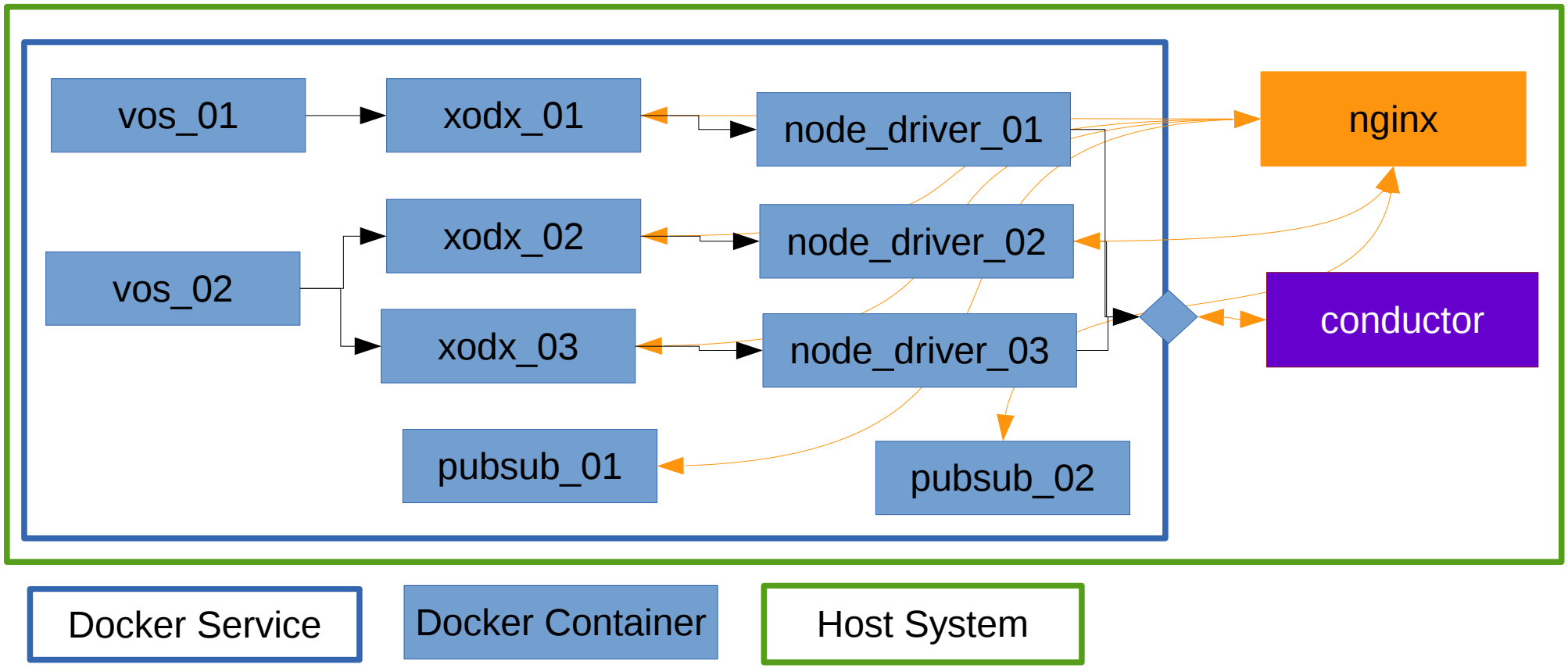



```
xodx2:
  image: 'xodx:default'
  links:
    - vos2:vos
  ports:
    - "8002:80"

xodx3:
  image: 'xodx:default'
  links:
    - vos2:vos
  ports:
    - "8003:80"

vos2:
  image: 'vos:lite-True_mem-128'
  mem_limit: 128m
  ports:
    - "1111"
```

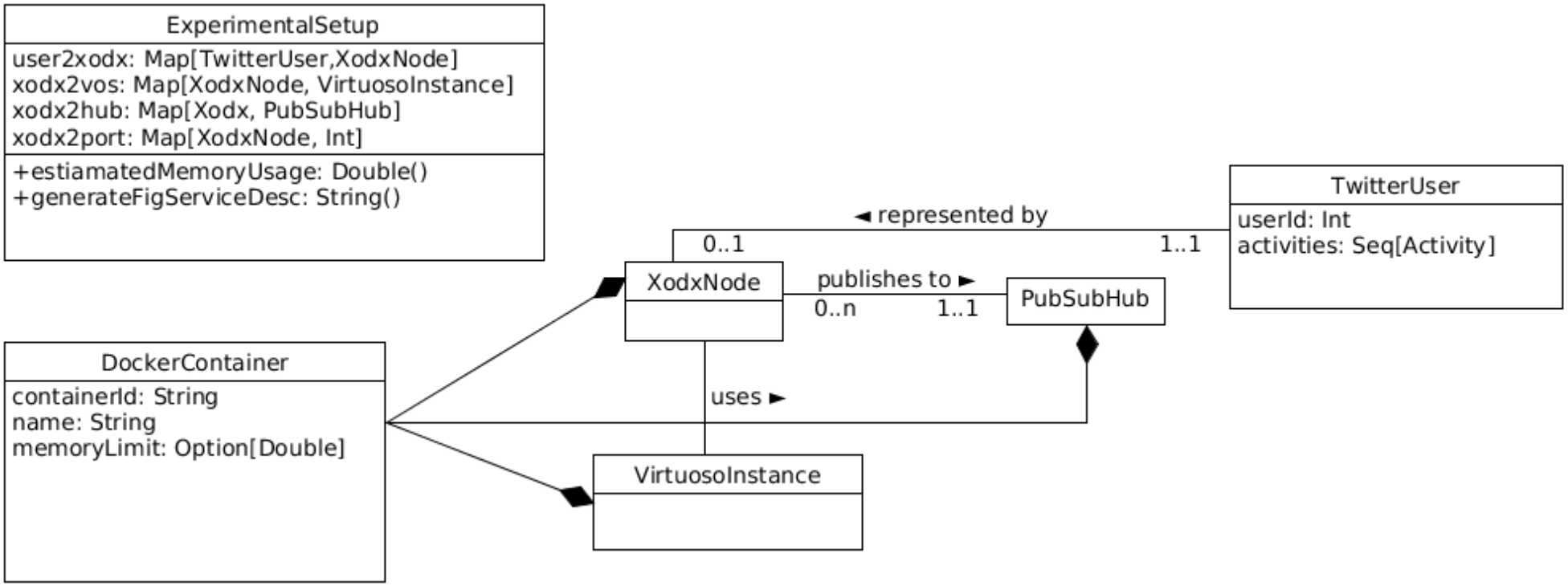
Aufbau des Simulations-Service



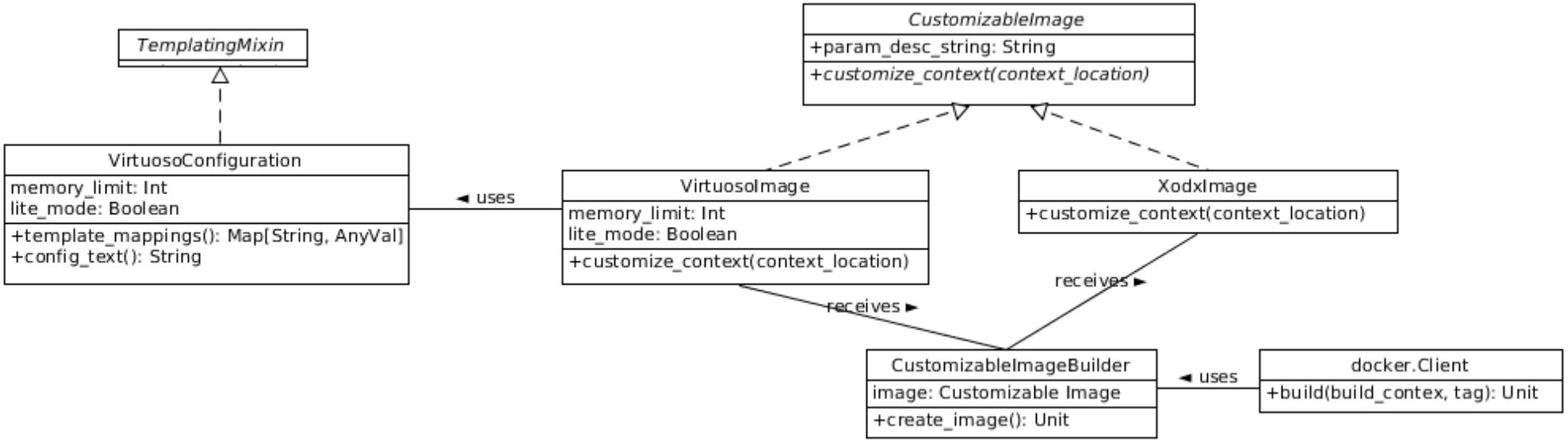
node driver: liest Ereignisse/Aktionen aus Event-Log (mounted volume) und alle Ereignisse Zeitscheibe sequentiell aus; meldet Beendigung einer Zeitscheibe zurück zum conductor

conductor: Monitoring der Prozesse und allg. Netzwerkauslastung gibt Startsignal für Zeitscheiben, sammelt Bestätigungen für der node driver ein (Synchronisationspunkte)

Design: Netzwerk-Topologie



Design: Image-Erstellung



Konfigurations-Templating

```
[Parameters]
ServerPort                = 1111
LiteMode                  = {{LiteMode}}
DisableUnixSocket         = 1
                    DisableTcpSocket          = 0

                                  [...]

NumberOfBuffers           = {{NumberOfBuffers}}
MaxDirtyBuffers           = {{MaxDirtyBuffers}}
```

- Mustache Templating Engine ('logic-less' templates)
- binding mit `['LiteMode': 'true', 'NumberOfBuffers': 800, 'MaxDirtyBuffers': 600]`

```
[Parameters]
ServerPort                = 1111
LiteMode                  = true
DisableUnixSocket         = 1
                    DisableTcpSocket          = 0

                                  [...]

NumberOfBuffers           = 800
MaxDirtyBuffers           = 600
```

ToDos und offene Fragen

- CLI Interface für Generieren der Topologie (envtl. mit Task-Management durch *Invoke/Fabric*)
- Reverse Proxy Konfiguration für xodx-Knoten in nginx
- Dockerized PubSub-Hub (bisher noch offizielle Hubs im WAN)
- conductor und node driver
- xodx-Fix: beachte und übernehme ursprüngliche URL des Requests
- Berechnen und Überwachen Summe des max. erwarteten Speicherverbrauchs (ggf. abhängiges Konfigurieren)

- Was ist die maximale Länge von Zeitscheiben?
- Welches Kommunikationsmittel zw. conductor und node driver? (REST, RabbitMQ, ZeroMQ)