Final Presentation - Dockerizing Linked Data

Georges Alkhouri, Tom Neumann

University of Applied Sciences Leipzig

6th Jul. 2015

Problem

Populare knowledge bases faceing **performence/availability** issues through **high request** rates.

Solution ↓

Run a local mirror of the knowledge base with a SPARQL endpoint.

New Problem

To run and maintain a local knowledge base environment is a complex task requiring a lot of effort and is not suitable for domain admins who just want to use the SPARQL interface.

New Solution ↓

Dockerizing Linked Data

Usage Example: Professorenkatalog

The Catalogus Professorum Lipsiensium

- Knowledge base of professors at the Leipzig University
- Includes records from 1409 to presence
- Comprises over 14, 000 entities
- Many interlinked connections in the LOD Cloud
- Curated by **historical researchers** and **interested citizen scientists**

Usage Example: Professorenkatalog

Infrastructure

Professorenkatalogs infrastructure consists of several web applications (Presentation, Storage, Backup, ...).

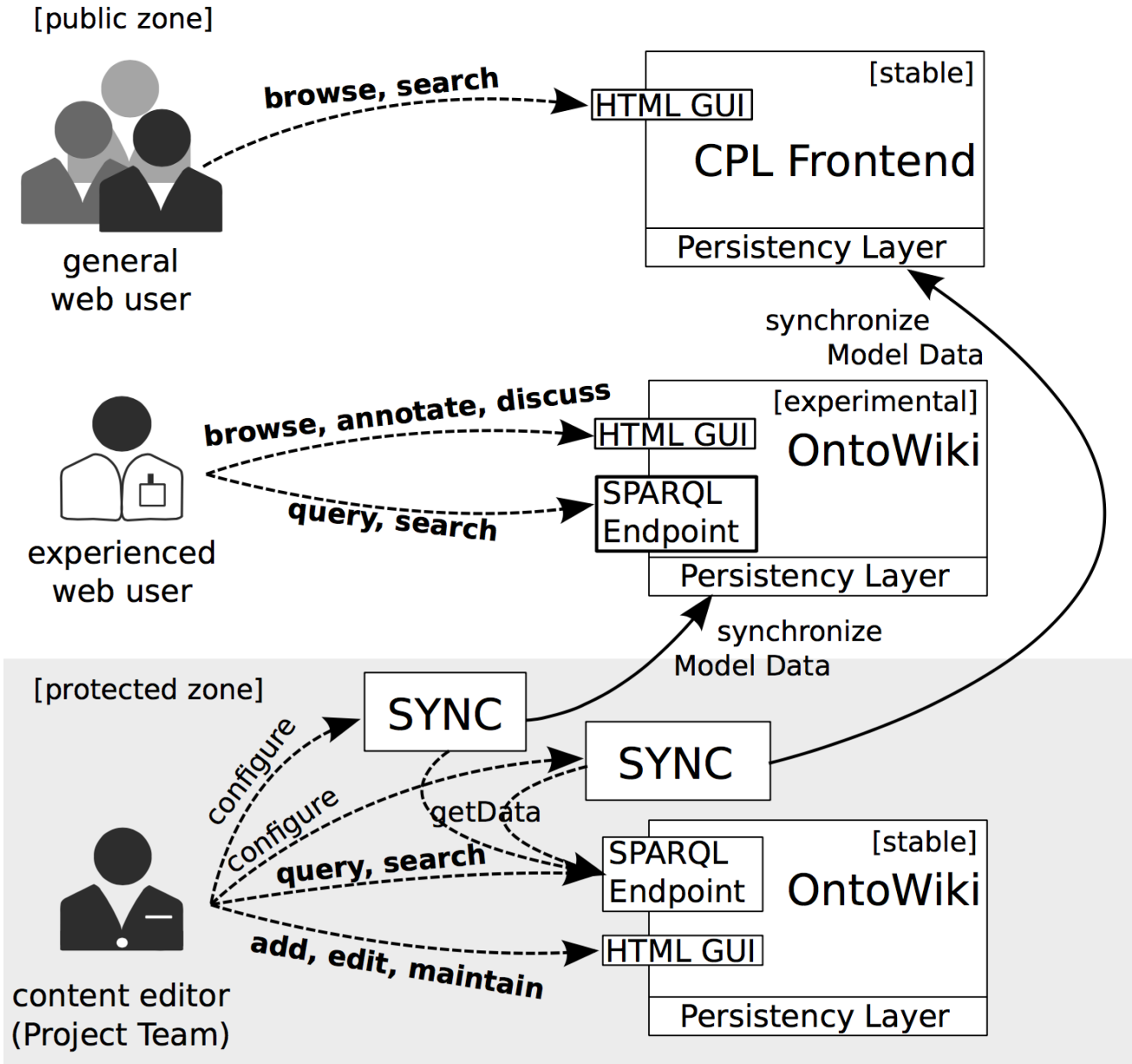

Figure: Architecture of Professorenkatalog[1, p. 6]

What is Docker?

Docker is a free virtualisation technology, which is based on Linux Containers.

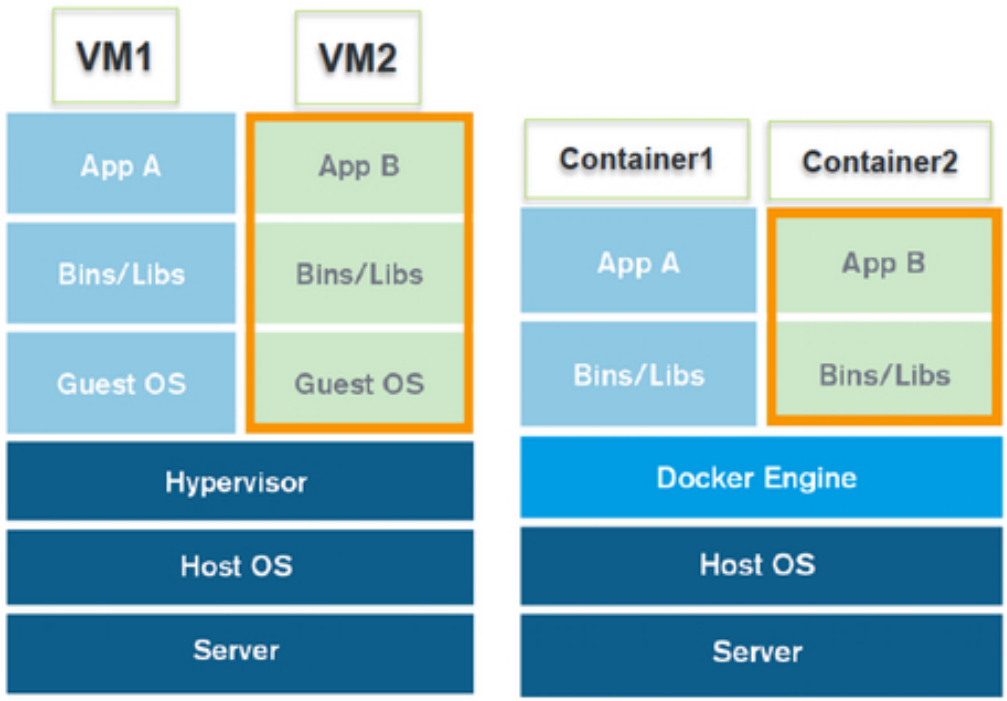

Figure: Virtual Machines vs Docker [2]

What is Docker?

Introduction

- Docker consists of two components: **Docker Engine, Docker Hub**
- **Docker Engine** is managing the containers and deploys the applications on them
- **Docker Hub** is a Docker repository to ship and run your applications anywhere

What is Docker?

Docker’s Architecture

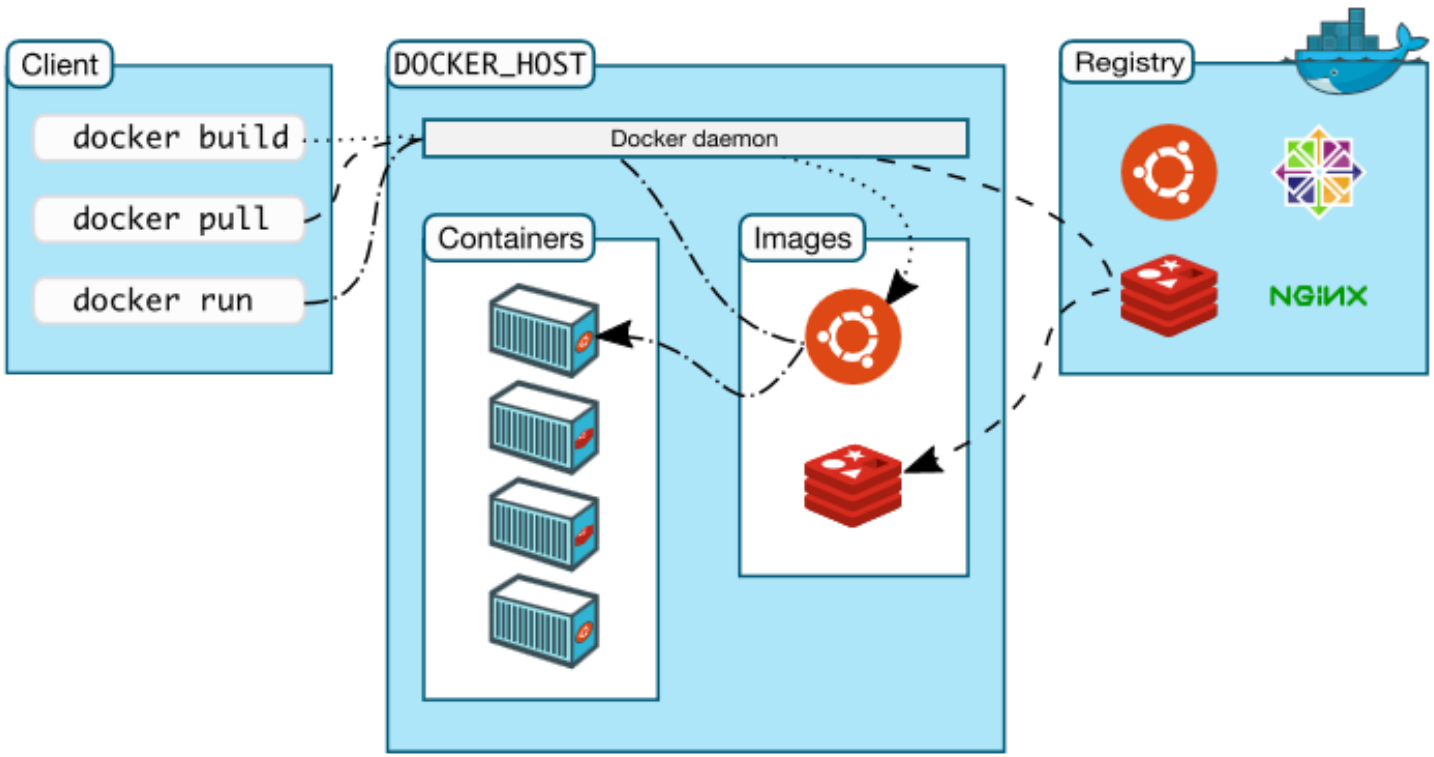

Figure: Architecture of Docker [2]

What is Docker?

Usage

1. Install Docker
2. Pull (and modify) a Docker image from the Docker Hub or create a Dockerfile
3. Run a container by using the Docker image

What is Docker?

Basic Commands

docker ...

build	*dockerfile*: build an image from a Dockerfile
run	*image*: run a command in a new container
start	*name*|*id*: start a stopped container
stop	*name*|*id*: stop a running container
rm	*name*|*id*: remove a container
rmi	*name*|*id*: remove an image

Docker Example: Virtuoso 7

Virtuoso is a SQL-ORDBMS and Web Application Server (Universal Server). The Server provides SQL, XML, RDF data mangement. Access to the Triple Store is available in many ways, for example via SPARQL, ODBC, JDBC.

Docker example: Virtuoso 7

Listing 1: Vituoso 7 Dockerfile

```
FROM debian:jessie
MAINTAINER Natanael Arndt ...
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update
# install some basic packages
RUN apt-get install -y libldap-2.4-2 libssl1.0.0 unixodbc
ADD virtuoso-minimal_7.2_all.deb \
virtuoso-opensource-7-bin_7.2_amd64.deb \
libvirtodbc0_7.2_amd64.deb
RUN dpkg -i virtuoso-minimal_7.2_all.deb \
virtuoso-opensource-7-bin_7.2_amd64.deb \
libvirtodbc0_7.2_amd64.deb
ADD virtuoso.ini.dist /
ADD run.sh /
# expose the ODBC and management ports to the outer world
EXPOSE 1111
EXPOSE 8890
ENV PWDDBA="dba"
VOLUME "/var/lib/virtuoso/db"
VOLUME "/import_store"
WORKDIR /var/lib/virtuoso/db
CMD ["/run.sh"]
```

Simple Docker Demo

Virtuoso Container

Dockerizing project is hosting an own virtuoso image at:

`https://registry.hub.docker.com/u/aksw/dld-store-virtuoso7/`

Simple Docker Demo

Run Container

Start and run a docker container through:

```
docker run -d
           --name="virtuoso"
           -p <host port>:8890 //SPARQL
           -p <host port>:1111 //ODBC
           -e PWDDBA="super secret"
           -v <host virtuoso directory>:/var/lib/
              ↪ virtuoso/db
            aksw/dld-store-virtuoso7
```

Simple Docker Demo

What is going on?

run	Run a command in a new container
-d	Run container in background and print container ID
- -name	Assign a name to the container
-p	Publish a container's port to the host
-e	Set environment variables into container
-v	Bind mount a volume

"aksw/dld-store-virtuoso7" is the image name, local or on docker hub

Simple Docker Demo

Setup Virtuoso

The virtuoso.ini file is injected into the container through `-v` which mounts the datebase folder from the host system into the container.

If not specified the container provides a fallback file.

Simple Docker Demo

Access Virtuoso Container

After `docker run` docker provides an access to the container through the exposed port (`-p 8890:8890`) on localhost.

`http://localhost:8890/sparql`

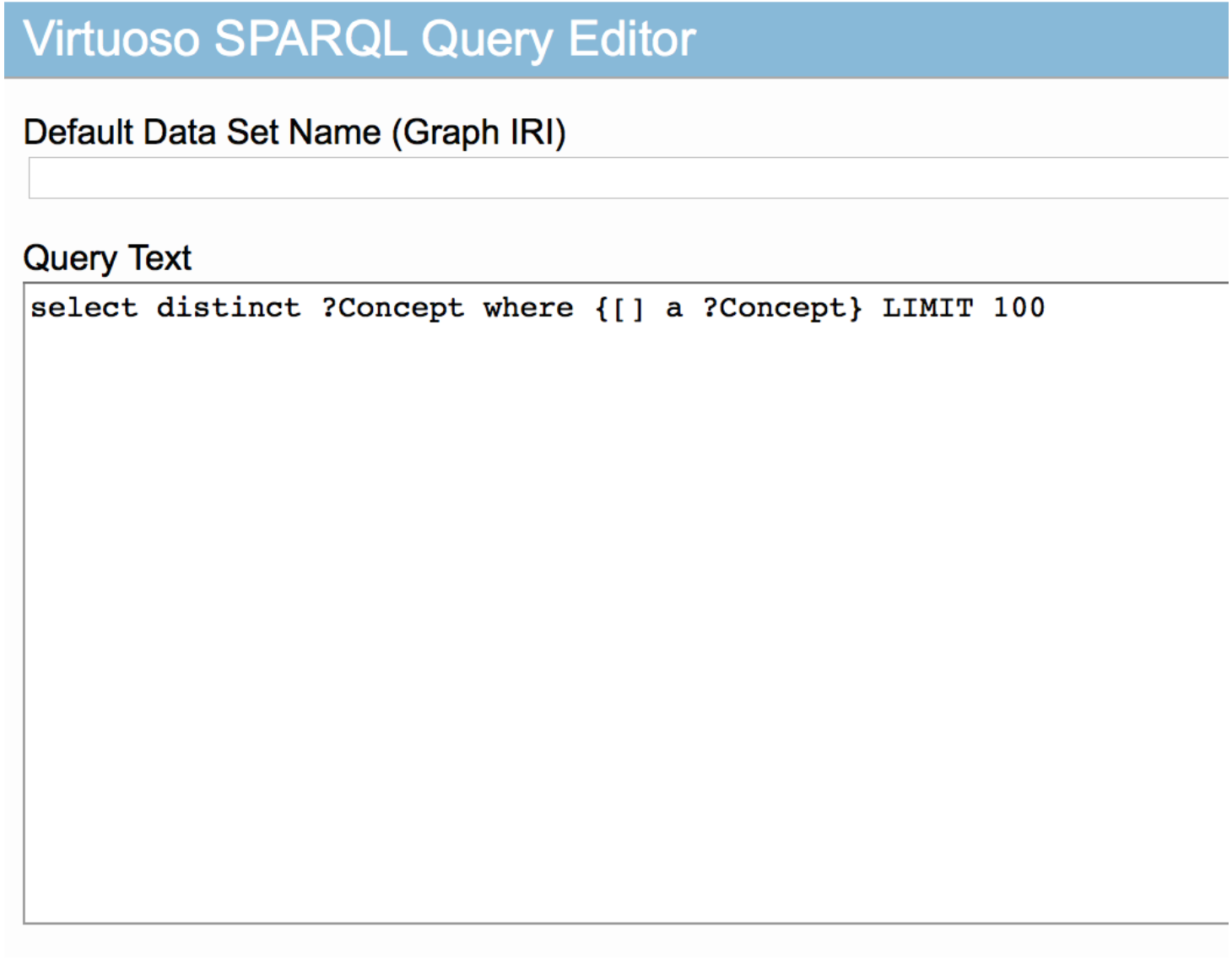

Figure: Virtuoso SPARQL Endpoint provided by a docker container

Multiple Containers

Communication

Containers can connect and expose information with each other they are not necessarily isolated.

Multiple Containers

Communication Approaches

Network port mapping

Maps a port inside the container to a port on the host (`docker run ... -p 8890:8890 ...`).

Linking System

Source containers information can be sent to a recipient container by naming the source

```
docker run --name="db" ...
```

and linking it to a recipient

`docker run --link="db" ... webserver`.

Multiple Containers

Linking System - Shared Information

Environment variables

Docker creates Environment variables in the target container,

```
...
DB_NAME=db
DB_PORT=tcp://172.17.0.5:5432
DB_PORT_5432_TCP=tcp://172.17.0.5:5432
DB_PORT_5432_PROTO=tcp
DB_PORT_5432_PORT=5432
DB_PORT_5432_ADDR=172.17.0.5
...
```

Multiple Containers

Linking System - Shared Information

Updating the /etc/hosts file

- Docker adds a host entry for the source container
- Automatically updates hosts file with new IP when source container restarts

Dockerizing Linked Data

The Project wants to improve the setup of linked data environments and make the replacement of components more easier.

through ↓

Applying **micro service architecture** with Docker

Dockerizing Linked Data

Containerised Knowledge Base

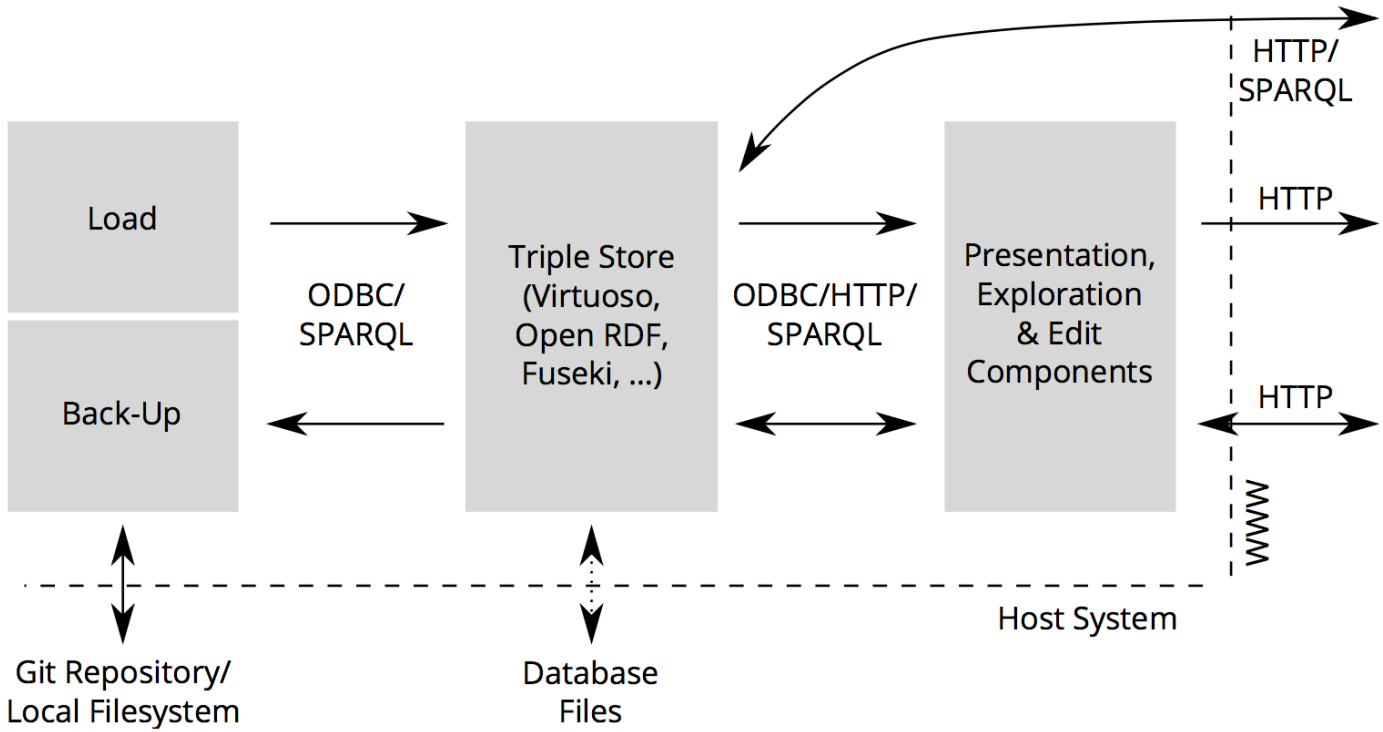

Figure: Architecture and data-flow of the containerized micro services[1, p. 3]

Dockerizing Linked Data

Docker Compose

The Dockerizing application works with Docker Compose.

Docker Compose:

- Tool for defining and running **multi-container** applications
- Define a multi-container application in a single file

Dockerizing Linked Data

Docker Compose how it works

1. Write some Dockerfiles for reproducing your images
2. Define the services that make up your app in **docker-compose.yml**
3. Run `docker-compose up` and Compose will start and run all services

Dockerizing Linked Data

docker-compose.yml file

Listing 2: Compose file example from Docker

```
web:
    build: .
    ports:
        - "5000:5000"
    volumes:
        - .:/code
    links:
        - redis
redis:
    image: redis
```

Previous example is equal to following docker commands:

```
docker run --name="redis" redis

docker build -t web .
docker run --link="redis" -p 5000:5000 -v .:/
    ↪ code --name="web" web
```

Dockerizing Linked Data

Linking

Compose connects containers and shares volumes, IP adresses or environment variables to multiple containers with the `link` or `volumes_from` tag.

Dockerizing Linked Data

Converting

Dockerizings dld.py Script converts a project custom YAML config file to a Docker Compose config file

Listing 3: Dockerizing Config File

```
datasets:
    dbpedia-homepages:
        graph_name: "http://dbpedia.org"
        file: "sample-data/homepages_en.ttl.gz"

    dbpedia-inter-language-links-old:
        file: "sample-data/old_interlanguage_links_en.
            ↪ nt.gz"

components:
    store:
        image: aksw/dld-store-virtuoso7
        environment:
            PWDDBA: herakiel
    load: aksw/dld-load-virtuoso
    present:
        - {
```

```
            image: aksw/dld-present-ontowiki,
            ports: ["88:80"],
        }
        #- image: aksw/dld-present-pubby

settings:
    default_graph: "http://dbpedia.org"
```

Listing 4: Converted Docker Compose File

```
load:
    environment: {DEFAULT_GRAPH: 'http://dbpedia.org'}
    image: aksw/dld-load-virtuoso
    links: [store]
    volumes: ['<absolute path>/wd-dld/models:/import']
    volumes_from: [store]
presentontowiki:
    environment: {DEFAULT_GRAPH: 'http://dbpedia.org'}
    image: aksw/dld-present-ontowiki
    links: [store]
    ports: ['88:80']
store:
    environment: {DEFAULT_GRAPH: 'http://dbpedia.org',
        ↪ PWDDBA: herakiel}
    image: aksw/dld-store-virtuoso7
```

Dockerizing Linked Data

Services

There are 4 kinds of services in the setup area of Dockerizing composer files:

store

- the store service defines a Triple Store
- needs an image (e.g. aksw/dld-store-virtuoso7)
- needs a volume for persistent data storage

load

- the load service defines a load image (e.g. aksw/dld-load-virtuosoload)
- it is needed to load data into the store

Dockerizing Linked Data

Services

backup

- defines a backup component (e.g. aksw/dld-backup-virtuoso)
- this component should be used for a backup of the Triple Store data

present

- defines one or more presentation images (e.g. aksw/dld-present-ontowiki)
- the component is used to explore the Triple Store data

Summary

The projects result is a collection of Dockerfiles / images and packages. The collection is consisting of semantic web images (e.g. Virtuoso7) and utility images (e.g. backup).

Advanteges

- docker images are simple to ship, use and modify
- many ready to use images on Docker Hub
- multi container applications

Disadvantege

- security doubts (http://www.golem.de/news/studie-docker-images-oft-mit-sicherheitsluecken-1505-114310.html)

Links

Dockerizing Web Site

http://dockerizing.github.io

Dockerizing @Github

http://github.com/dockerizing

Dockerizing Images @ Dockerhub

https://registry.hub.docker.com/repos/aksw/

References

[1] Knowledge Base Shipping to the Linked Open Data Cloud
Natanael Arndt, Markus Ackermann, Martin Brümmer, Thomas Riechert
Jul. 2015

[2] Docker documentation
https://docs.docker.com/
Jul. 2015