SPLAB

Zurich University of Applied Sciences

zh aw School of Engineering

InIT Institute of Applied Information Technology

Transactional Migration of Inhomogeneous Composite Cloud Applications

Josef Spillner, Manuel Ramírez López
Service Prototyping Lab (blog.zhaw.ch/splab)

Sep 12, 2018 | 4th CloudWays @ 7th ESOCC

Our research on cloud-native apps

Cloud-Native Design and Architecture

UCC 2015, ESOCC 2017, FGCS 2017, ...

Cloud-Native Software Engineering/TechDebt

... soon!

"Migration"

Design/Code

→ Maturity levels

→ Technologies

"Co-Transformation to Cloud-Native Applications: Development Experiences and Experimental Evaluation" (CLOSER 2018)

"A mixed-method empirical study of Function-as-a-Service software development in industrial practice" (PeerJ Preprints 6:e27005v1)

→ *well-understood*

Location

→ Portability

"Towards Quantifiable Boundaries for Elastic Horizontal Scaling of Microservices" (UCC 2017)

& This paper:

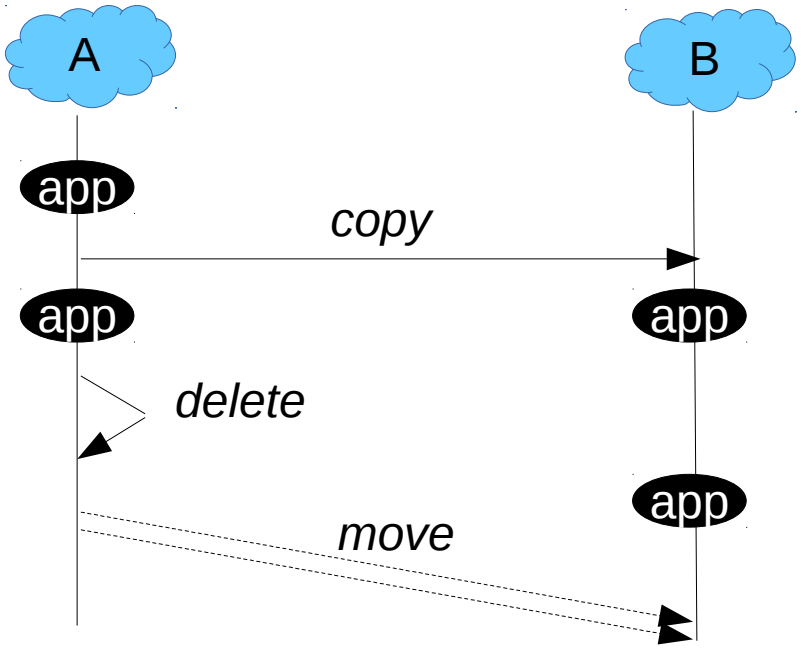
"Transactional Migration of Inhomogeneous Composite Cloud Applications" (CloudWays/ESOCC 2018)

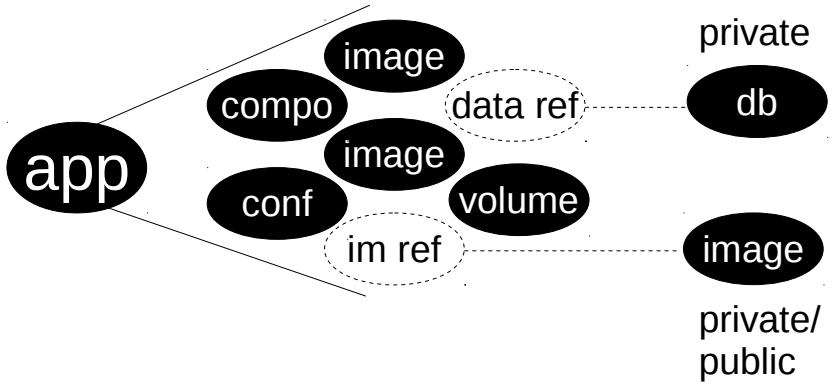
→ *needs research*

Migration semantics

Main differentiation: *copy* vs. *move* at runtime

- applicable to: code (images/image refs), composition/configuration, data
- transactional semantics required, too *

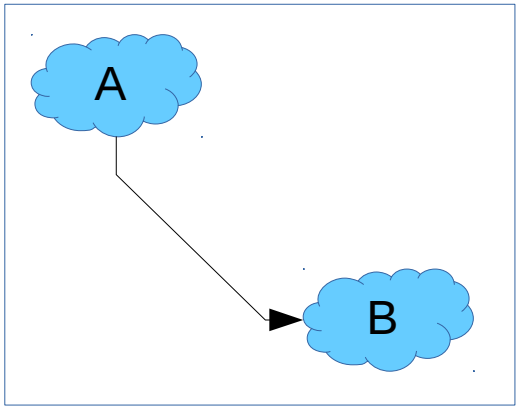




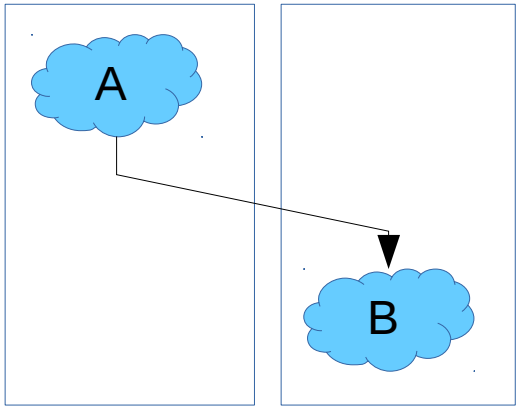
* some resemblance of “ship of Theseus“, Heraclitus‘ puzzle

Migration use cases (industry-defined)

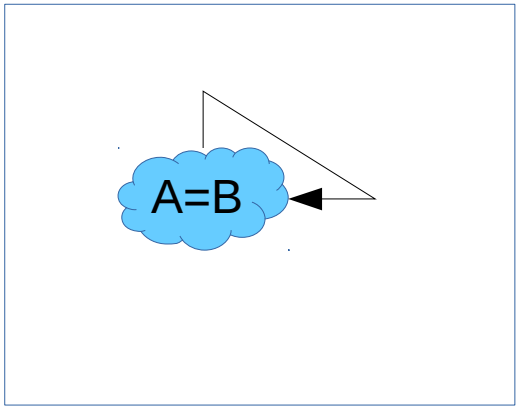
(1) inter-region/zone



(2) across providers



(3) in-situ reconfiguration



- new storage cluster
- new storage class
- composition updates (with immutable deployments)

1. Scale down Deployment or StatefulSet
    - Do this only when needed, probably via a configuration
    - It has to be done for ReadWriteOnce volumes
2. Mount source and destination PVs (via PVCs)
3. Rsync data from source to destination
4. Verify that all data has been copied successfully
5. Patch Deployment or StatefulSet to use the new PV
6. Scale up Deployment or StatefulSet again
7. Stop Pod

Migration technologies

Virtual machines

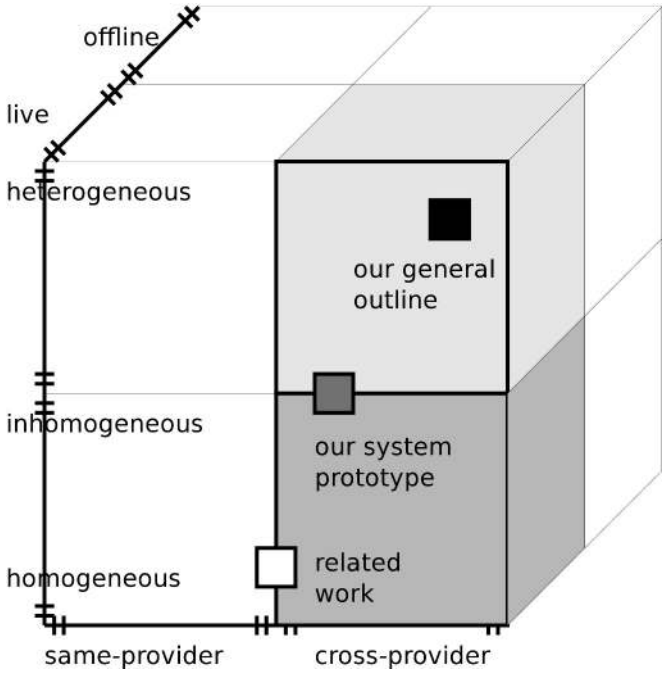
- unidirectional (asymmetric; vendor lock-in by [economic] design)
  - AWS Snowball, Server Migration Service (vSphere, Hyper-V) & similar
- bidirectional (symmetric)
  - vMotion, KVM migrate/savevm/loadvm, XenMotion

Containers

- Docker image save/load, rsync...
- Docker engine 1.7 live migration (demo Jun‘15, not generally available)
- Docker with CRI-O (checkpointing - no longer active since Dec‘15)
- Flocker (portable containers - no longer active since Dec‘16)
- Kubernetes: Helm charts, recent, no data; otherwise focus on pods
- Virtuozzo - works, but technology differences...
- Jelastic - commercial solution

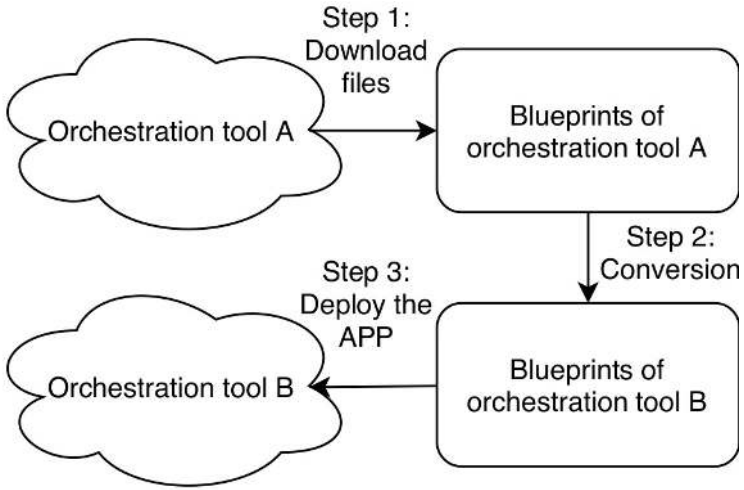
again: → *needs research*

Migration categories and dimensions

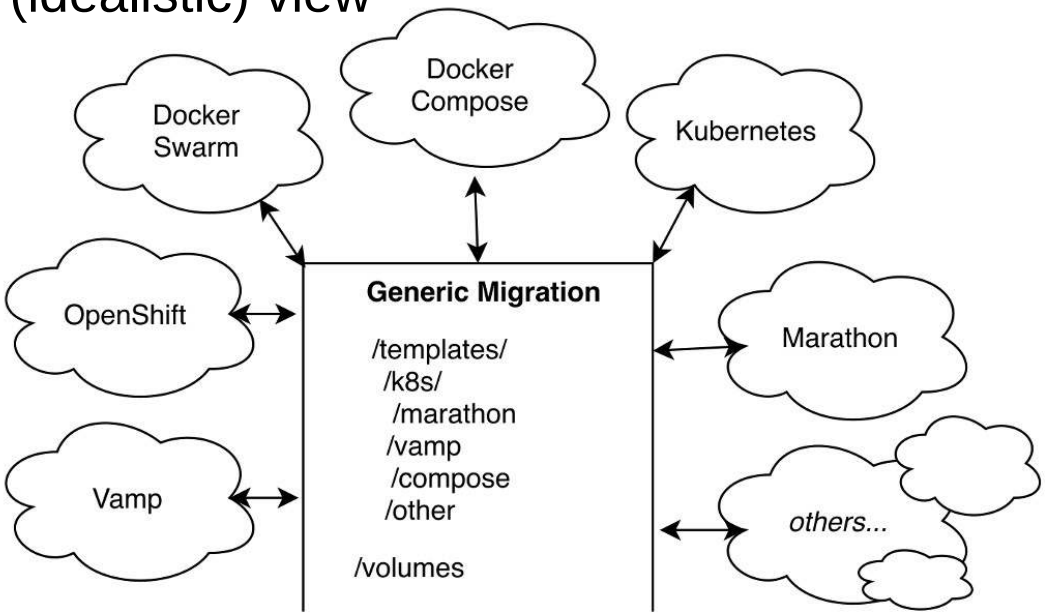


Design considerations

Migration workflow

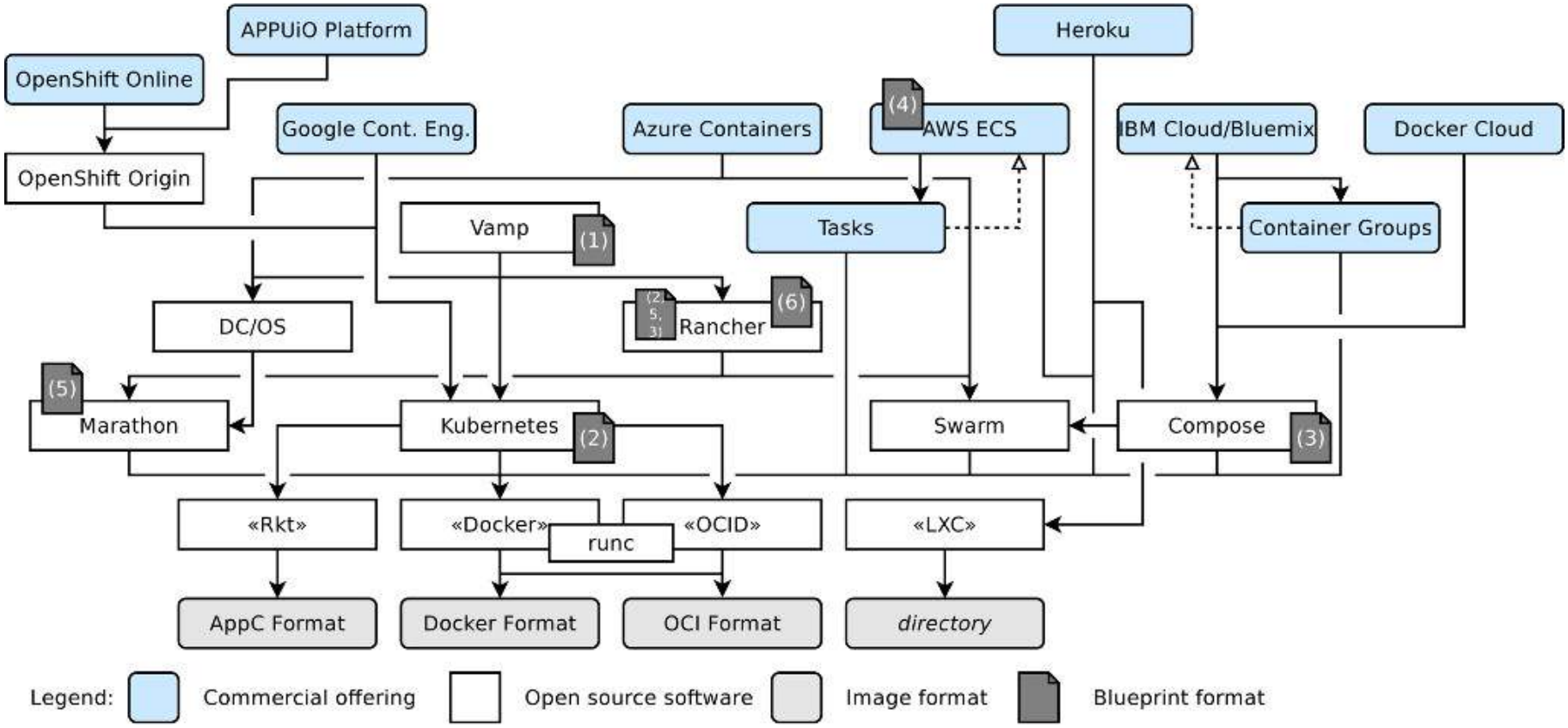


Heterogeneous (idealistic) view



Design considerations

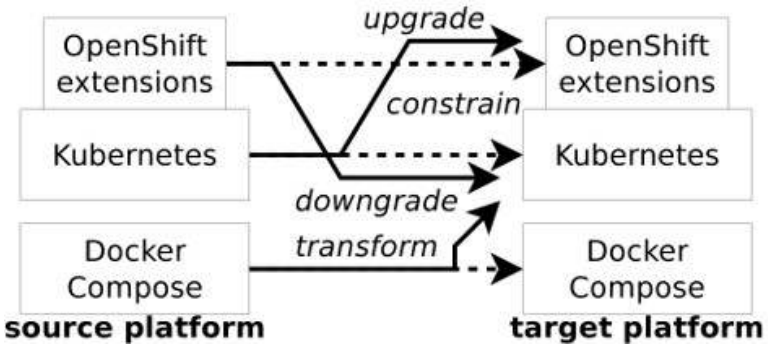
Yet... the unsurmountable reality fueled by lots of VC investment...



Observations:
- consolidation does occur, but:
- differences remain (configuration, extensions, distributions)

Design considerations

Inhomogeneous (realistic) view



Paths:
- *upgrade*
  e.g. generate DeploymentConfig from Deployment
- *downgrade*
  e.g. remove ImageStream
- *constrain*
  e.g. change replicas and reduce # of pods
- *transform*
  e.g. turning compose file into deployment descriptors

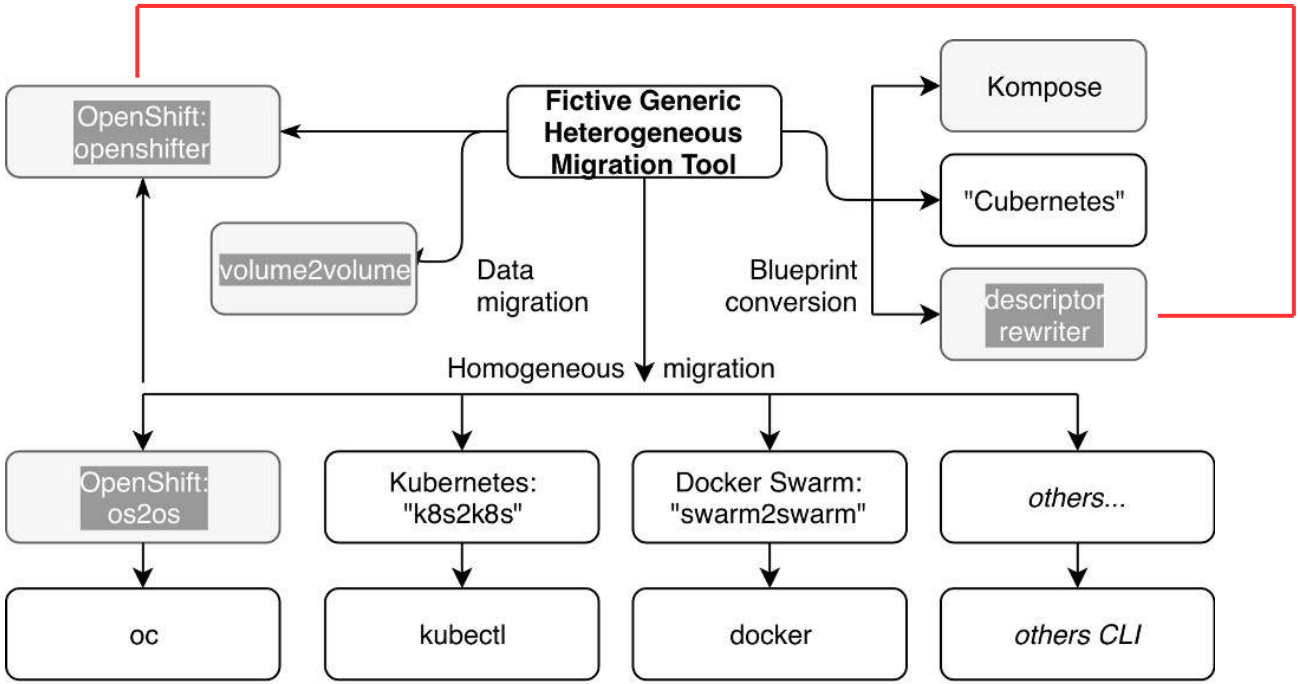
Iterative experience buildup by:
- multiple alternative (competing) implementations

Representation of applications eventually as:
- auto-generated Helm charts based on Kubernetes/OpenShift deployments (except for Docker Compose)
- “fat charts“ concept for fully self-contained stateful snapshots

Transaction guarantees
- ability to cancel in-flight + rollback, along with prediction
- (pre-copy/post-copy differential state transfer sequence)

Competing implementations

Envisioned and realised prototypes



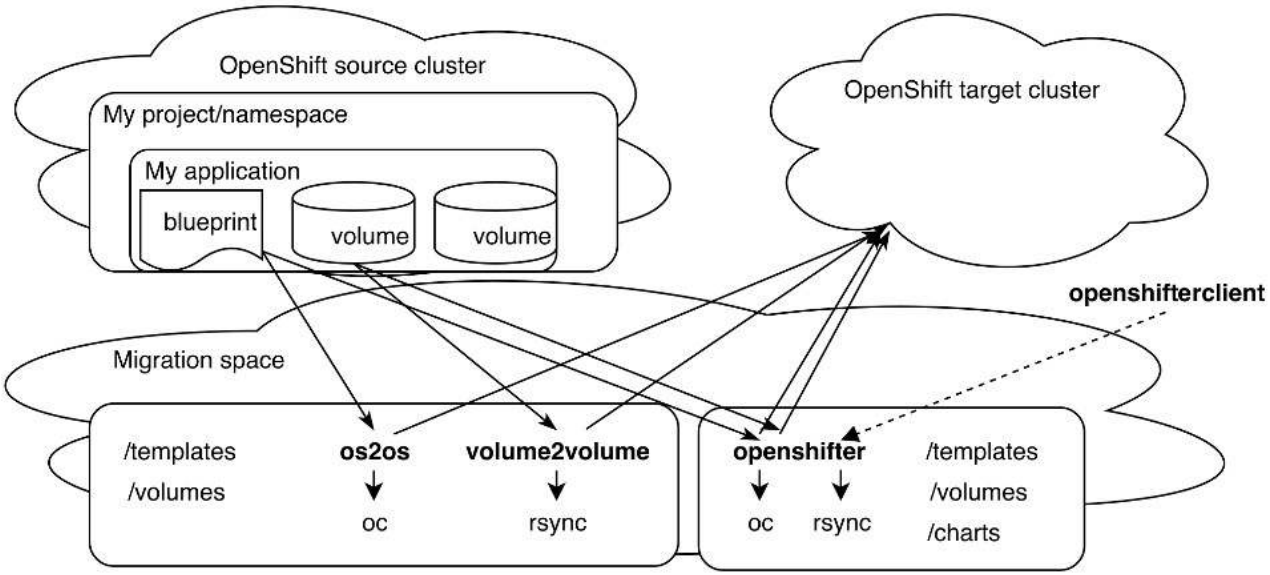
Takeaway:

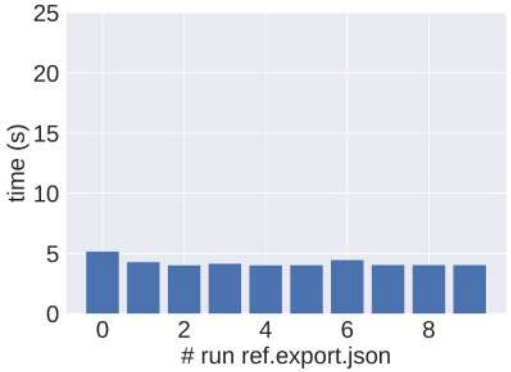
- fully developed os2os/volume2volume with testing, CI/CD integration, ...
- continuing evolvement of openshifter as more promising design
  - deployable as scalable service
  - interwoven service and state handling
  - integration of constraints via descriptor rewriting

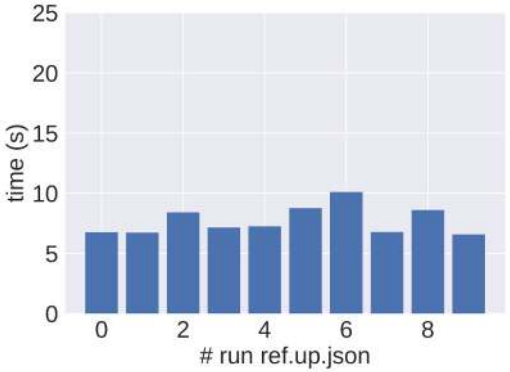
Evaluation

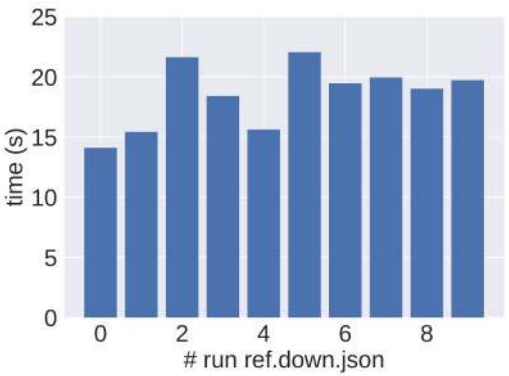
Disclaimer: only first couple of experiments

- focus on OpenShift instances (clusters) within data centre









Conclusions

Achievements
- study of feasibility of portable, take-where-you-go cloud applications
- initial concept for stateful application migration (in the ‘transfer‘ sense)

Limitations
- concept not fully implemented yet, lack of autodiscovery and failure provocation

Applied research in industry context
- requirements changing with customer requests + technological evolution
- prototypes available as open source via our research lab repository
  - **http://github.com/serviceprototypinglab/**
- automated testbed setup scheduled to arrive
  - allows for better reproducible research
- long-term perspective (i.e. support by Kubernetes ecosystem vendors) not yet clear