AGH

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Evaluation of container composition tools for multi-container distributed systems

Michał Orzechowski

CGW Workshops 2016
Kraków, 26.10.2016

Agenda

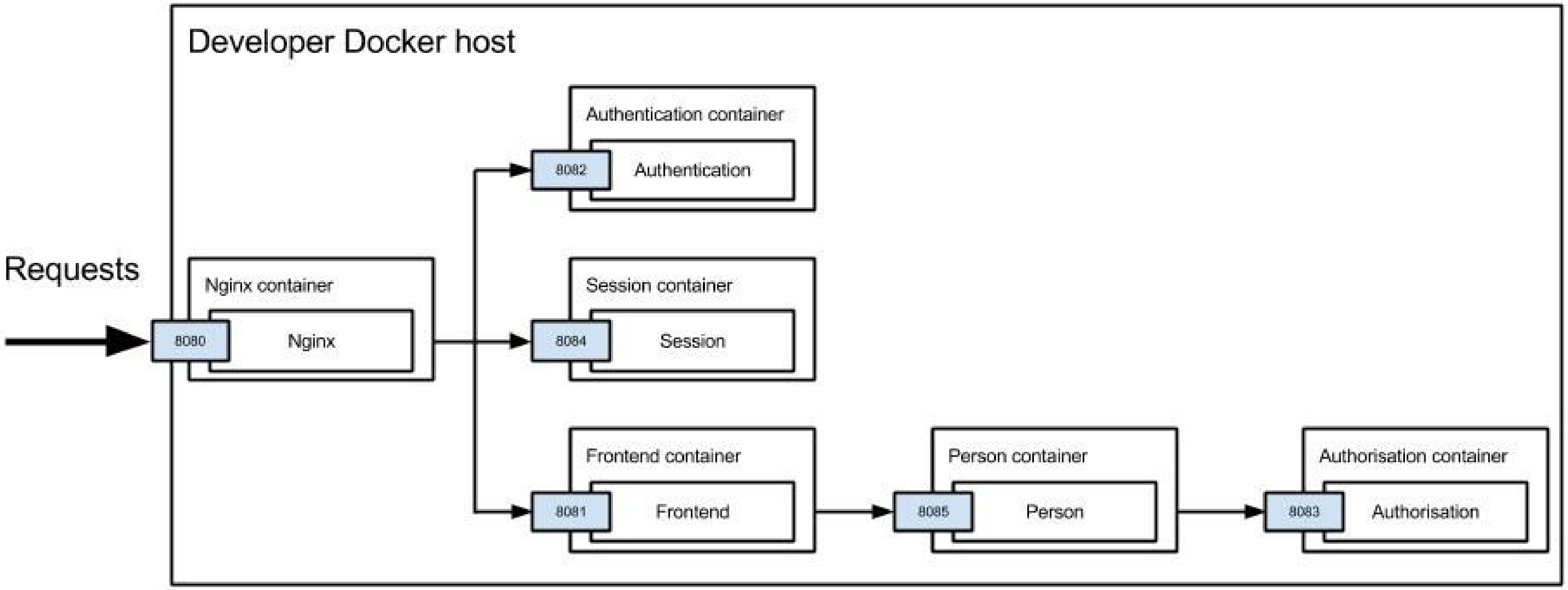
1. Distributed System and Microservices
2. Example Distributed System
3. Container Orchestration Frameworks
4. Container Composition Description
5. Examples of service stacks descriptions
6. Container Composition Description Comparison
7. Limitations of Composition Description
8. Future work

Distributed Systems and Microservices

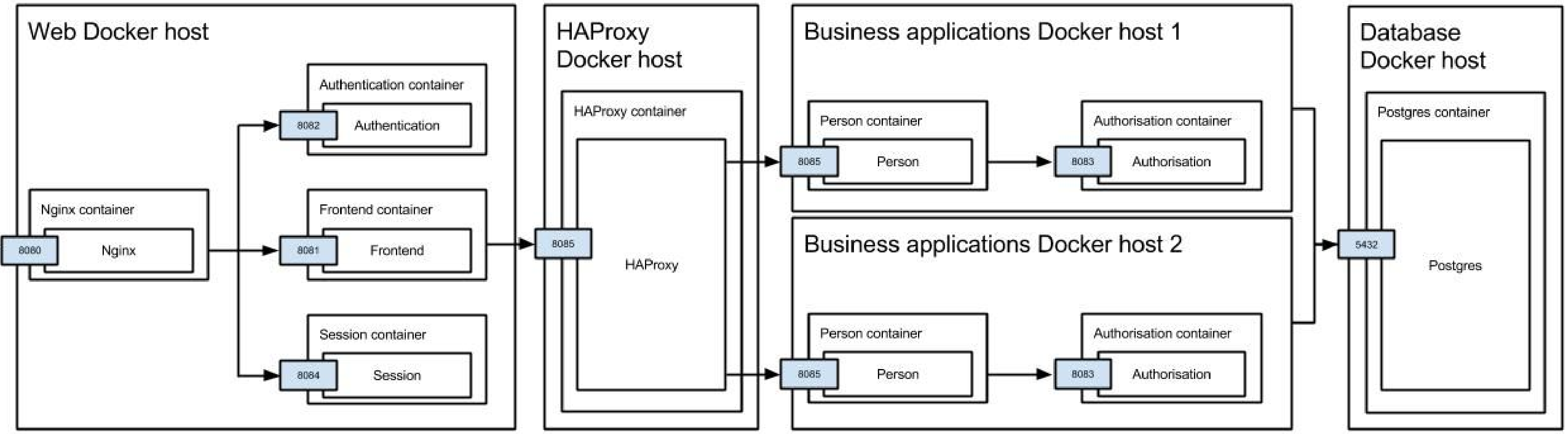
- microservices are production proven alternative to SOA
- promote good practices of well designed distributed systems
- work well at scale eg. moderately complex web shop can easily constitute of 450 microservices
- good fit for virtual machine and container deployments

Example distributed system build from microservices

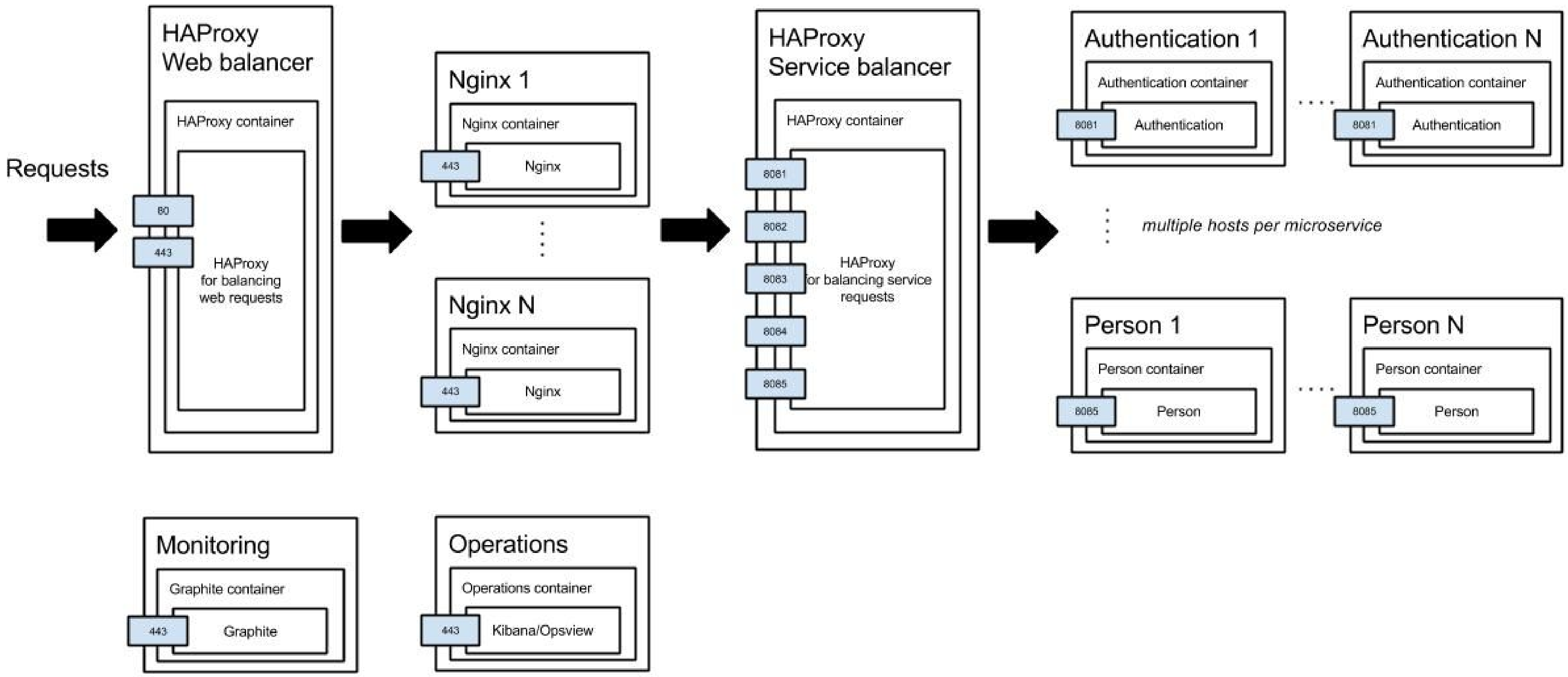
Simple Local Architecture Example



Simple Scaled Architecture Example



Large Scaled Architecture Example



How to deploy and manage it at scale?

Container Orchestration Frameworks

- Cattle (part of Racher)
- Docker Swarm
- Kubernetes
- Marathon (part of Mesos)
- Fleet (part of CoreOS)

Container Composition Description

Stack files defined with:

- Docker Compose
- Rancher Compose
- Kubernetes Objects (Pods, Services…)
- Crowdr (orchestration on single node, but using script)

Docker Compose and Kubernetes

```
frontend:
  image: java:8-jre
  links:
    - person
  ports:
   - "8081:8081"
  volumes:
   - docker/volume-frontend:/frontend
   - docker/volume-log:/log
  command: "run_frontend.sh"

person:
  image: java:8-jre
  links:
     - authorisation
  ports:
   - "8085:8085"
  volumes:
   - docker/volume-person:/person
   - docker/volume-log:/log
  command: "run_person.sh"
```

```
kind: Pod
apiVersion: v1beta1
id: person-mysql
desiredState:
  manifest:
    version: v1beta1
    id: mysql
    containers:
      - name: person-mysql
        image: mysql
        cpu: 100
        ports:
          - containerPort: 3306
        volumeMounts:
          - name: mysql-persistent-storage
            mountPath: /var/lib/mysql
    volumes:
      - name: mysql-persistent-storage
        source:
          persistentDisk:
            pdName: replicated-person-mysql-disk
            fsType: ext4
```

Racher Compose and Crowdr

```
frontend:
  image: java:8-jre
  links:
    - person
  ports:
   - "8081:8081"
  volumes:
   - docker/volume-frontend:/frontend
   - docker/volume-log:/log
  command: "run_frontend.sh"
  scale: 2
  load_balancer_config:
    haproxy_config: {}
  health_check:
    port: 42
    interval: 2000
    unhealthy_threshold: 3
    healthy_threshold: 2
    response_timeout: 2000
```

```
#!/bin/bash

crowdr_project="example"

# frontend
frontend image mysql:5.7.10
frontend before.run create_network
frontend net overlay
frontend volume volume-frontend:/var/lib/mysql

# person
person image wordpress:4.3.1
person net overlay
person volume person-frontend:/var/lib/mysql
person publish 8085:8085
```

Container Composition Description Comparison

	Docker Compose	Racher Compose	Kubernetes
Types of Workloads	**Cloud Native applications**	**Cloud Native applications**	**Cloud Native applications**
Application Definition	**Kubernetes Objects (Pods, Services, Controllers) YAML, JSON**	**docker-compose.yml (services, volumes, networks) YAML**	**racher-compose.yml (services, volumes, networks) YAML**
Application Scalability constructs	**Manual or automated scaling of Pods**	**Manual scaling of individual services**	**Manual scaling of individual services**
Logging and monitoring	**Liveness, readiness**	**Liveness**	**Liveness, readiness**
Distributed Storage	**Storage backends (e.g. NFS, AWS EBS)**	**Single host volumes, extendable with Flocker**	**Single host volumes, extendable with Flocker**

Standard Release Process Model

1. Development and local testing
2. Compile and fast tests
3. Slow tests
4. User Acceptance Testing
5. Performance testing
6. Production

A need to tailor service composition to **environment** and **configuration**.

Limitations of Composition Description

- limited static syntax (YAML, JSON)
- cannot mix multiple stack files
- cannot inherit from a individual service
- cannot inherit from a base stack file
- cannot respond to changes in configuration or environment
- no information about version constraints
- no notion of horizontal scaling or performance constraints

Maven vs. Gradle

Future work

- Further evaluation of quickly number of container related tools
- Comparison of presented composition tools with model-driven approaches eg. CAMEL or TOSCA
- Development of service stack syntax using a script based, statically typed dynamic language (eg. TypeJs) with support for Docker Engine
- Development of

Thank you!

Michał Orzechowski

PhD Student, Department of Computer Science

AGH University of Science and Technology