Microservices & docker: from theory to practice

• Tom Van Cutsem, PhD and Nico Janssens, PhD
• Bell Labs, Application platforms and software systems lab
• December 1st, 2016
Context

Microservices

DEVOPS

docker
What drives microservices?

Cloud

DEVOPS

Continuous Deployment

Microservices
You’re in good company
What are microservices?

“SOA done right”

-- James Lewis and Martin Fowler
Microservices: characteristics

Componentization via services

Smart endpoints, dumb pipes
Microservices: characteristics

Products, not projects

“you build it, you run it”

Organized around business capabilities
Microservices: characteristics

Products, not projects

“you build it, you run it”

Organized around business capabilities
Microservices: organize around business services

"Any organization that designs a system ... will inevitably produce a design whose structure is a copy of the organization’s communication structure."
-- Melvin Conway, 1968

(Source: Martin Fowler)
Microservices: characteristics

- Decentralized Governance
- Decentralized Data Management
Large codebases seem to auto-evolve into microservices
War stories from large web companies

1st gen
Monolithic app
Written in one language (e.g., Perl, C++)

2nd gen
Tiered architecture
Java as dominant language

3rd gen
Polyglot microservices
Mix of languages

(Source: highscalability.com, 2015)
Microservices: characteristics

Infrastructure Automation

“IT is an API”

Design for Failure

Evolutionary Design
Microservices & DevOps culture

- Need to be able to provision infrastructure \textit{fast}
- Containerize services (Docker)
- Container orchestration (Swarm, Kubernetes, Rancher, Mesos, …)
- Teams maintain their own services in production
Microservices: risks

**Independent services**
- Service boundaries not easy to change
- End-to-end testing/debugging more difficult
- Distributed systems challenges

**Design for Failure**
- Investment in monitoring tools
- Operational complexity

**Technological Diversity**
- Strong and diverse skill set
Case study: ShowMe
ShowMe: location-based video sharing
Discover or share what’s up near a location of interest
ShowMe: location-based video sharing
Prototype app + experience movie
Functional architecture

- Web server
- Session server
- Checkin server
- Media server
- Analytics DB
- Session storage
- Stored streams

APIs:
- Geocoding API
- Auth API
- Venue API
- Push API
- Checkin API

Components:
- Web client (consumer)
- Web client (producer)
- Foursquare app
Technology stack

Web server

Session server

Checkin server

Web client (consumer)

Web client (producer)

Foursquare app

Geocoding API

Session storage

redis

openstack

Media server

C++

Stored streams

Analytics DB

Neo4j

Foursquare

Foursquare

Node.js

Java
Service interfaces

- Web client (consumer)
- Web client (producer)
- Foursquare app
- Session server
- Checkin server
- Web server
- Geocoding API
- REST/HTTP
- Session storage
- Media server
- Stored streams
- Analytics DB

REST/HTTP connections:
- Web client (consumer) to Web server
- Session server to Checkin server
- Checkin server to Web server
Lessons learned

• Multiple teams working on independent subsystems = highly productive
  – Different goals
  – Different skillsets
  – Different release schedules
  – Less conflicts
• Testing and debugging of the overall system was a pain
• We didn’t sufficiently invest in tooling and automation
  – Manual configuration and set-up
  – Infrastructure not set up to host multiple versions of the app
  – No cross-service unit testing infrastructure
Lessons learned

• Micro-service architecture = distributed system
  – Deal with asynchrony, failure, latency, keeping data consistent across databases
  – Interfaces between services are implicit, not checked by compiler.

• Testing services in isolation is not enough
  – Focus is on monitoring and detecting anomalies more than on thorough testing before deployment

• Deployment is much more complicated
  – Fine-grained orchestration and configuration
  – Each service needs clustering, monitoring, load-balancing, …
  – Variety of runtimes and databases requires larger skill set to tweak, deploy, maintain
  – To do microservices right, should keep old and new versions of the service running side-by-side
Case study: instadash
Instadash app

Real-time fleet tracking

- GPS receiver
- OBD via CAN bus
- Dashcam
- On-board Unit

2 real cars,
10 hours footage
400 virtual cars
Instadash: functional architecture
Instadash: functional architecture

- Web client
- Web gateway
- Websocket server
- Media server
- Resource manager
- Device registry
- Query deployer
- System monitor
- Message Bus
- Workers

Public

Device registry
Instadash: technology stack
Instadash: service interfaces

- **Web client**
- **Web gateway**
- **Websocket server**
- **Media server**
- **Resource manager**
- **Device registry**
- **Query deployer**
- **System monitor**
- **Message Bus**
- **Workers**

**Protocols and Communication:**
- **REST/HTTP**
- **AMQP**
- **JSON**
- **JSON-RPC**
- **Protobuf**

Device registry communicates with:
- **Query deployer**
- **System monitor**

Message Bus communicates with:
- **Query deployer**
- **System monitor**
- **Workers**

Workers communicate with:
- **System monitor**

Websocket server communicates with:
- **Media server**
- **Resource manager**

Resource manager communicates with:
- **Device registry**

Device registry communicates with:
- **Web client**
- **Web gateway**
- **Websocket server**
- **Media server**
- **Resource manager**
- **Query deployer**
- **System monitor**
- **Workers**

Web client communicates with:
- **FAYE**
- **RTC**

Web gateway communicates with:
- **FAYE**
- **RTC**

FAYE communicates with:
- **RTC**

RTC communicates with:
- **Devices**
Microservice communication patterns

REST over HTTP

Microservice A

Microservice B

REST/HTTP is well-understood
HTTP support is ubiquitous
JSON as data model is a natural fit

Text-based protocol overheads
Microservice communication patterns

RPC

Microservice A

Microservice B

Fast, often binary encoding
Built-in schema support

Firewall issues, less ubiquitous
Need an additional discovery service
Microservice communication patterns

Decoupling between components (bus handles both discovery and routing)

More complex, beware bottleneck
Communication patterns

- Web client
  - Web gateway
  - Websocket server
  - Media server
  - Resource manager
- Query deployer
- System monitor
- Message Bus
- Workers
- Device registry

Open Source Enterprise Messaging

- Open source
- Large community
- Many client libraries
Communication patterns: point-to-point

- Web client
- Web gateway
- WebSocket server
- Media server
- Resource manager
- Device registry
- Query deployer
- System monitor
- Message Bus
- Workers

- JSON-RPC over AMQP
- Messages represent requests or commands
Communication patterns: publish-subscribe

- Use AMQP's rich routing semantics via topic exchanges
- Messages represent events (JSON payload)
Communication patterns: work queueing

- A single queue served by multiple workers.
- Goal is to spread tasks over multiple instances of the same service.
- Messages represent tasks (work to be done)
Monitoring: our approach

- Used Riemann as central dashboard and event monitoring server
- Client libraries for a variety of programming languages (remember: polyglot)
- Each microservice regularly reports service-specific statistics
- Each host machine also reports generic resource statistics
Monitoring

Web client

- Web gateway
- Websocket server
- Media server
- Resource manager

Websocket server

- Query deployer
- System monitor

Message Bus

- Protobufs over UDP or TCP
- Device registry

Workers

Device registry

Protobufs over UDP or TCP

RIEMANN
Monitoring: dashboards
Lessons learned

- Message bus as **central broker** had many advantages
  - Solved service discovery (all components need to know the broker, not each other)
  - Queueing makes services more robust to failover
  - Message bus dashboard gave a wealth of system information about communication patterns, message rates, etc.
  - But: can quickly become a bottleneck: proper configuration and tuning was key
  - Also: all components needed hardening to e.g. auto-reconnect when broker went down

- Use external **configuration files** that can be generated or templated from a central place
- Use **schema validation** to catch bugs faster (e.g. JSON-Schema, Protobufs, AVRO, …)
- **Monitoring** was essential to see what’s going on
- **Dockerizing** services was key to getting this system going (20+ processes)
PART II

On MicroServices, Docker and DevOps
Context

Microservices

DEVOPS

docker
Docker Containers
Efficiency

Lightweight application isolation ➔ very low performance overhead

source: https://blog.jayway.com/2015/03/21/a-not-very-short-introduction-to-docker/
Container programming ➔ Dockerfile

FROM ubuntu:16.04
MAINTAINER Sven Dowideit <SvenDowideit@docker.com>

RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:screencast' | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/sshd

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
Docker Containers
Active eco-system

source: http://slidedeck.io/dpdornseifer/reveal_docker
Docker Containers
Portability

The matrix from hell

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

source: http://www.slideshare.net/Docker/docker-lpc-2014cristian
Docker Containers
Flexibility

Build, ship and run any app, anywhere [docker]

source: http://blog.terranillius.com/post/docker_testing/
Docker Containers
Demo
DevOps
@10k feet

DevOps
CALMS

**Culture**
- Promotes collaborative and open culture between Dev and Ops
- Embrace change and experimentation

**Automation**
- Automate whenever possible
- CI/CD, Infrastructure as Code, ...

**Lean**
- Focus on producing value for the end-user
- Small size batches, higher release cycles

**Measurement**
- Measure everything all the time and use this info to improve/refine cycles
- Show the improvement

**Sharing**
- Open information sharing – experiences, successes, failures, etc.
- Collaboration & communication – learn from each other (Kanban board, IM, wiki)
Moving away from traditional telco service design

Operational costs pressures push Telcos to virtualize environments while preserving **non-functional requirements**

- 5 nines availability
- Reliability
- Performance and response times
Moving away from traditional telco service design

Additional **non-functional requirements** to take into account

- Scalability
- Elasticity
- Agility
- Operability and portability

Low overhead
Portability
Micro-service architectures
Active eco-system + public image registries
Facilitates DevOps methodology
Bell Labs Projects: New Home/IoT Service Platform

- Home
  - Small footprints
  - Device
  - Hub

- Sensor network
- WiFi network
- Mobile 5G network

- Distributed Cloud Platform
  - For Heterogeneous HW infrastructure

- High Performance & Predictability

- Edge Cloud
- Central Cloud
Bell Labs Projects: Bandwidth Optimized Streaming Analytics

Bell Labs Projects: New Communication Service

**Key Goal:** Simplify interactions among people, machines, and their environments

- From transaction-oriented Web model to persistent conversations
- Uniform interaction model for people, machines, and objects
- Rich context-based communications and collaboration
Micro-service chat architecture

Public load balancer

WS server

message broker

TURN/STUN

Media Server

WS server

HTTP server

web storage

distributed object store and FS

load balancer

coTURN

mobile client

web browser

machine buddies

coTURN

location and geo-fencing service

RT + batch analytics

Stateless Workers

Chat-Thread Mgr

User View Mgr

User Mgr

Group Mgr

User Presence Mgr

Push Notification Adapter

Call Mgr

...

SDK

web browser

machine buddies

mobile client

Web storage

distributed object store and FS

node

redis

COMCAST CMB

Java

mongoDB

key-value database

document-oriented db
Initial production design
20 node cluster with RANCHER and DOCKER

Rancher Cluster Manager
- Native Docker support
- Re-usage of Docker Compose files
- Low entry hurdle
- Dashboard
- Redundancy
- IPSec support
Evaluation
MicroServices

Rapid and independent evolution (lifecycle management) ✔
Use the right tool for the job ✔
Decentralized governance and data management ✔
Evaluation

Docker

Low overhead ✔

Portability ✔

Micro-service architectures ✔

Active eco-system with public image registries ✔

Facilitates DevOps methodology ✔
Docker lifecycle management
• Don’t forget to clean old containers and dangling images
• For non-trivial lifecycle mgmt and production environments, rely on other tools
  – compose, swarm, kubernetes, mesos+marathon/chronos, saltstack, terraform, etc.

Dockerfiles
• Think carefully how to structure your Dockerfiles (across Dockerfiles)
  – Each line in a Docker file is a separate image layer, which by default will be cached (exceptions!)
• Order from generic/stable commands to specific/unstable commands
  – Use explicit version tagging for all installed packages (consistency across future builds)
  – Avoid unnecessary layers & packages → smartly combine commands

Performance when sharing host resources (e.g. when using bridge network)
No need to dockerize all your services ...
Evaluation
Docker

Application packaging → KISS!
   – Containers are not VMs, but application environments
   – Don’t try to stuff too many background services inside each container (sshd, logging, etc.)
   – Don’t install build tools (e.g. gcc) without good reason → use build containers for that!

Data storage
   – Try to avoid storing (all) data inside the application containers
     • Containers should be as much as possible easily replacable
   – Use key-value stores (etcd), DBs (mysql), data containers or host-volumes (-v)

Security
Networking
Background reading and references

- Martin Fowler’s article (must read): http://martinfowler.com/articles/microservices.html
- Community site: http://microservices.io/
- Insightful blogs:
  - http://www.tigerteam.dk/2014/micro-services-its-not-only-the-size-that-matters-its-also-how-you-use-them-part-1/
  - A critical note: http://contino.co.uk/microservices-not-a-free-lunch/
Copyright and confidentiality

The contents of this document are proprietary and confidential property of Nokia. This document is provided subject to confidentiality obligations of the applicable agreement(s).

This document is intended for use of Nokia’s customers and collaborators only for the purpose for which this document is submitted by Nokia. No part of this document may be reproduced or made available to the public or to any third party in any form or means without the prior written permission of Nokia. This document is to be used by properly trained professional personnel. Any use of the contents in this document is limited strictly to the use(s) specifically created in the applicable agreement(s) under which the document is submitted. The user of this document may voluntarily provide suggestions, comments or other feedback to Nokia in respect of the contents of this document ("Feedback").

Such Feedback may be used in Nokia products and related specifications or other documentation. Accordingly, if the user of this document gives Nokia Feedback on the contents of this document, Nokia may freely use, disclose, reproduce, license, distribute and otherwise commercialize the feedback in any Nokia product, technology, service, specification or other documentation.

Nokia operates a policy of ongoing development. Nokia reserves the right to make changes and improvements to any of the products and/or services described in this document or withdraw this document at any time without prior notice.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability or contents of this document. NOKIA SHALL NOT BE RESPONSIBLE IN ANY EVENT FOR ERRORS IN THIS DOCUMENT or for any loss of data or income or any special, incidental, consequential, indirect or direct damages howsoever caused, that might arise from the use of this document or any contents of this document.

This document and the product(s) it describes are protected by copyright according to the applicable laws.

Nokia is a registered trademark of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.