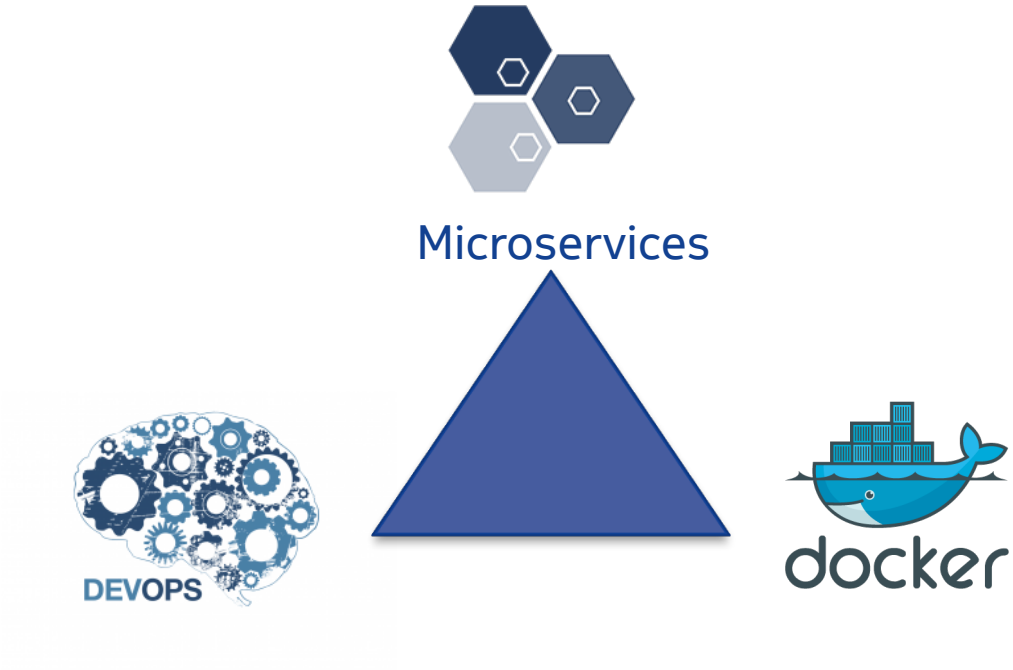
NOKIA Bell Labs

Microservices & docker: from theory to practice

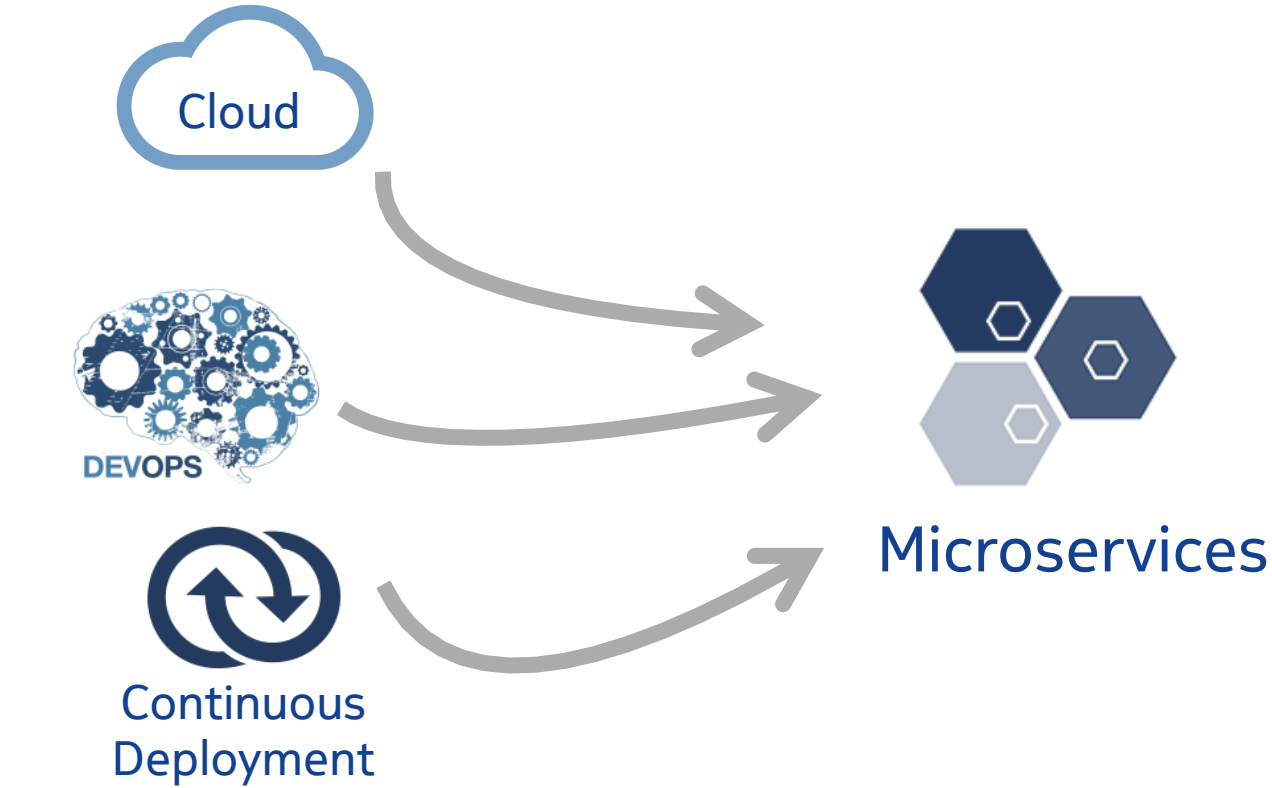
- Tom Van Cutsem, PhD and Nico Janssens, PhD
- Bell Labs, Application platforms and software systems lab

- December 1st, 2016

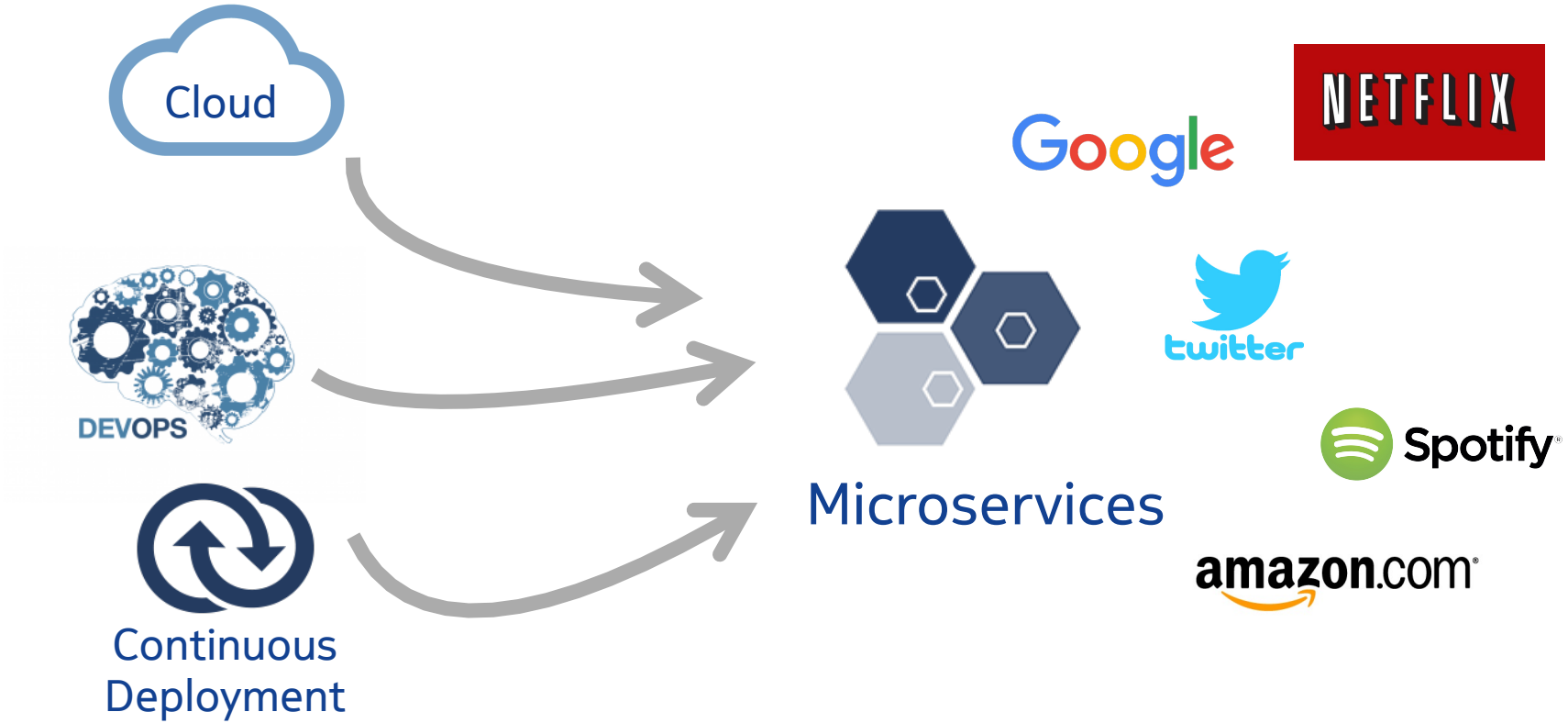
Context



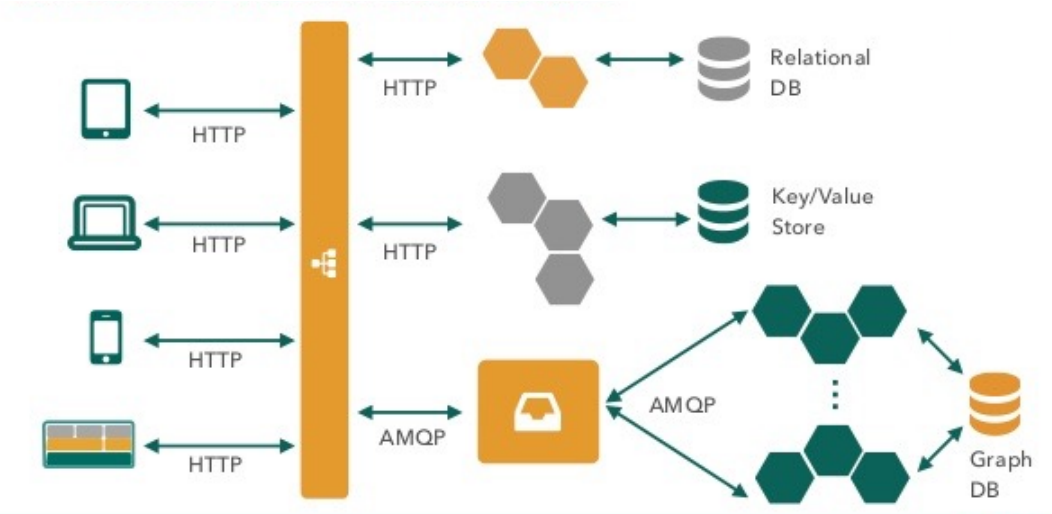
What drives microservices?



You're in good company



What are microservices?

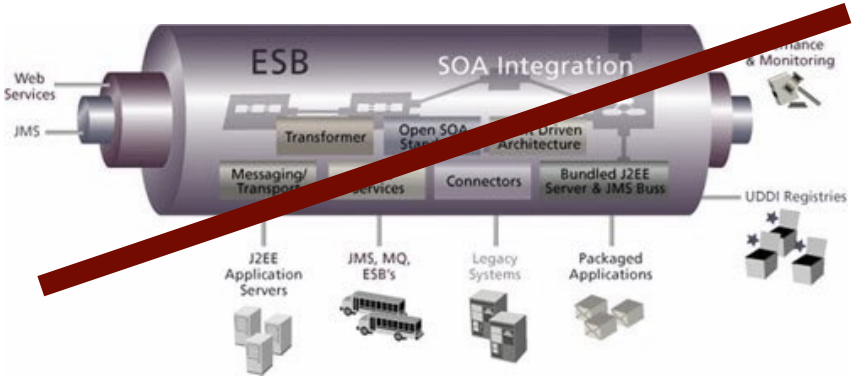


"SOA done right"

-- James Lewis and Martin Fowler

Microservices: characteristics

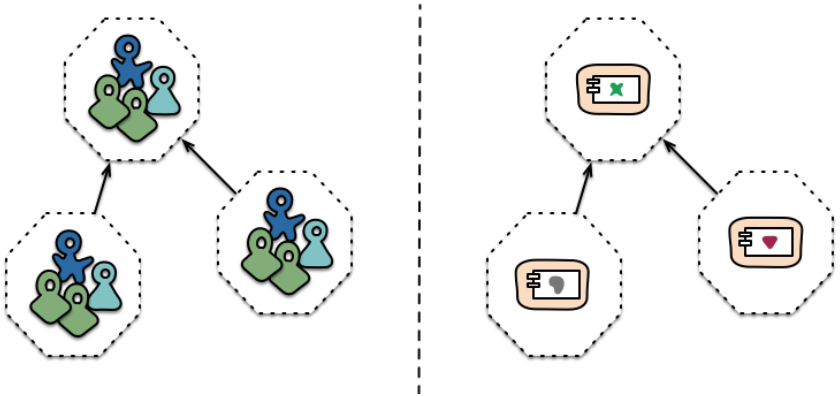
Componentization
via services



Smart endpoints,
dumb pipes

Microservices: characteristics

Products,
not projects

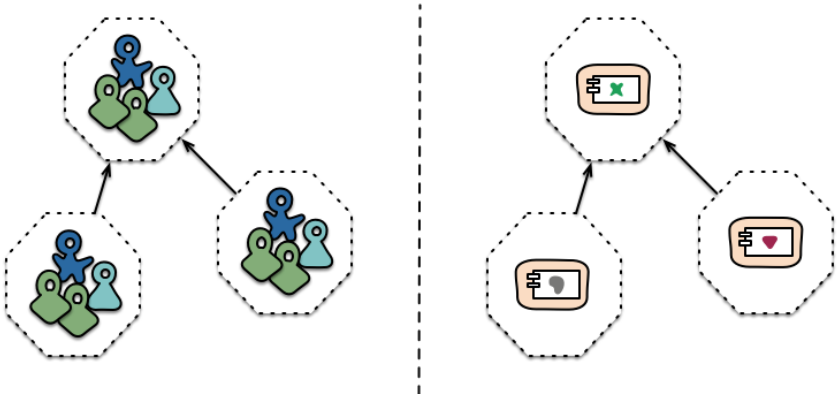


Organized around
business capabilities

“you build it, you run it”

Microservices: characteristics

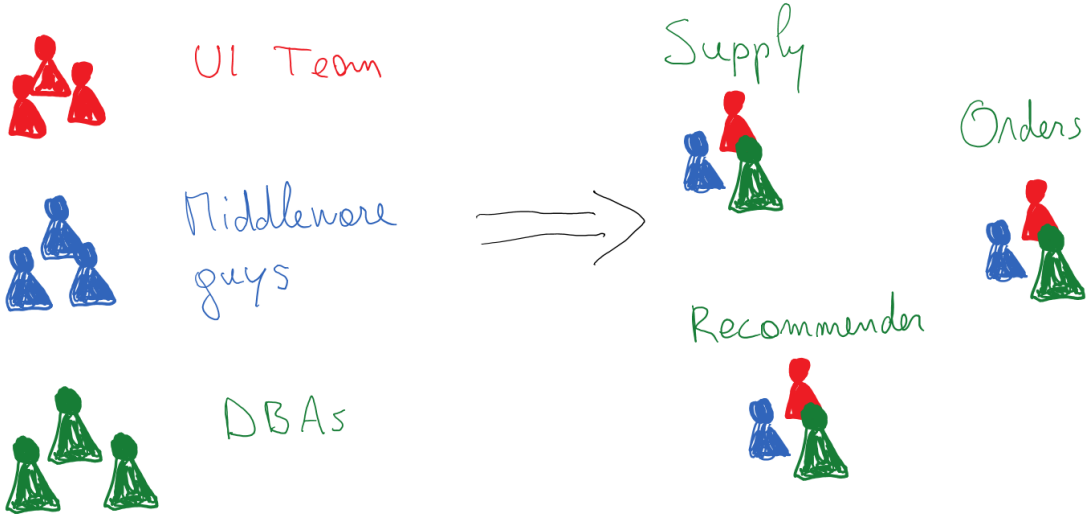
Products,
not projects



Organized around business capabilities

“you build it, you run it”

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)

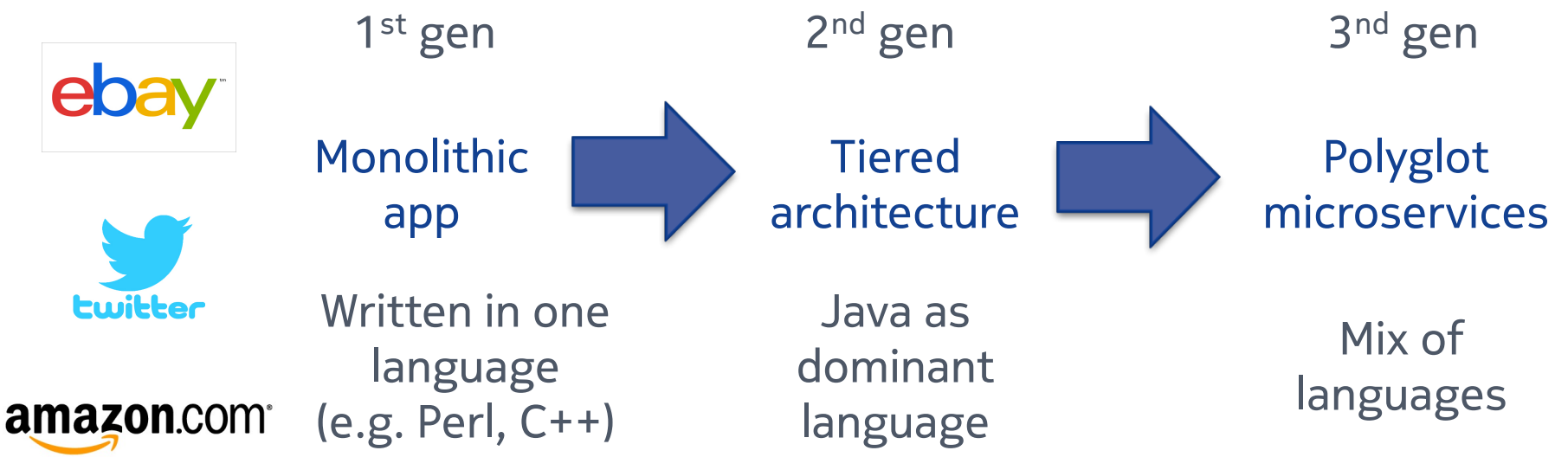
Microservices: characteristics

Decentralized
Governance

Decentralized
Data Management

Large codebases seem to auto-evolve into microservices

War stories from large web companies



(Source: highscalability.com, 2015)

Microservices: characteristics

Infrastructure Automation

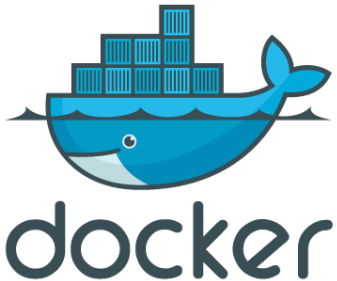
“IT is an API”

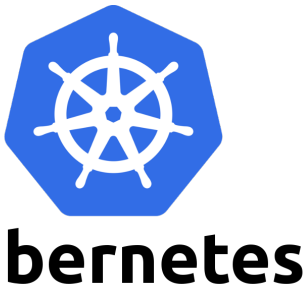
Design for Failure

Evolutionary Design

Microservices & DevOps culture

- Need to be able to provision infrastructure *fast*
- Containerize services (Docker)
- Container orchestration (Swarm, Kubernetes, Rancher, Mesos, …)
- Teams maintain their own services in production





Microservices: risks

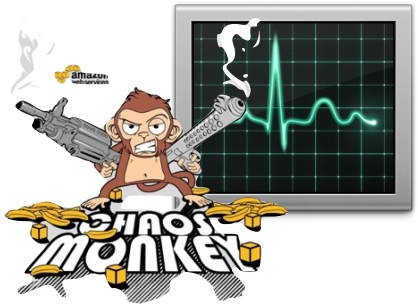
Independent services

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

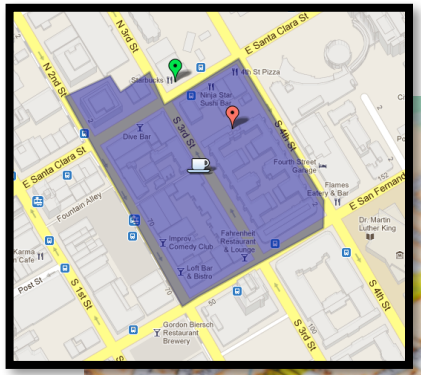
Technological Diversity

Strong and diverse skill set

Case study: ShowMe

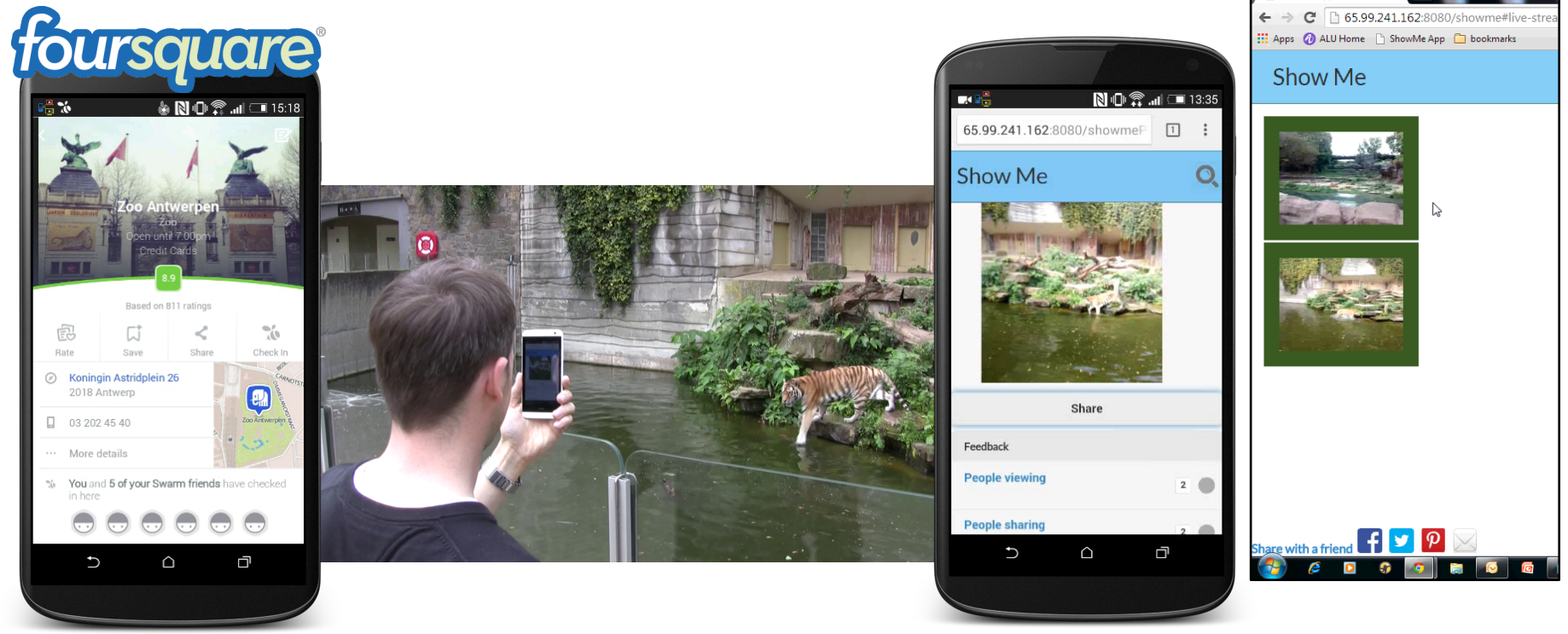
ShowMe: location-based video sharing

Discover or share what's up near a location of interest



ShowMe: location-based video sharing

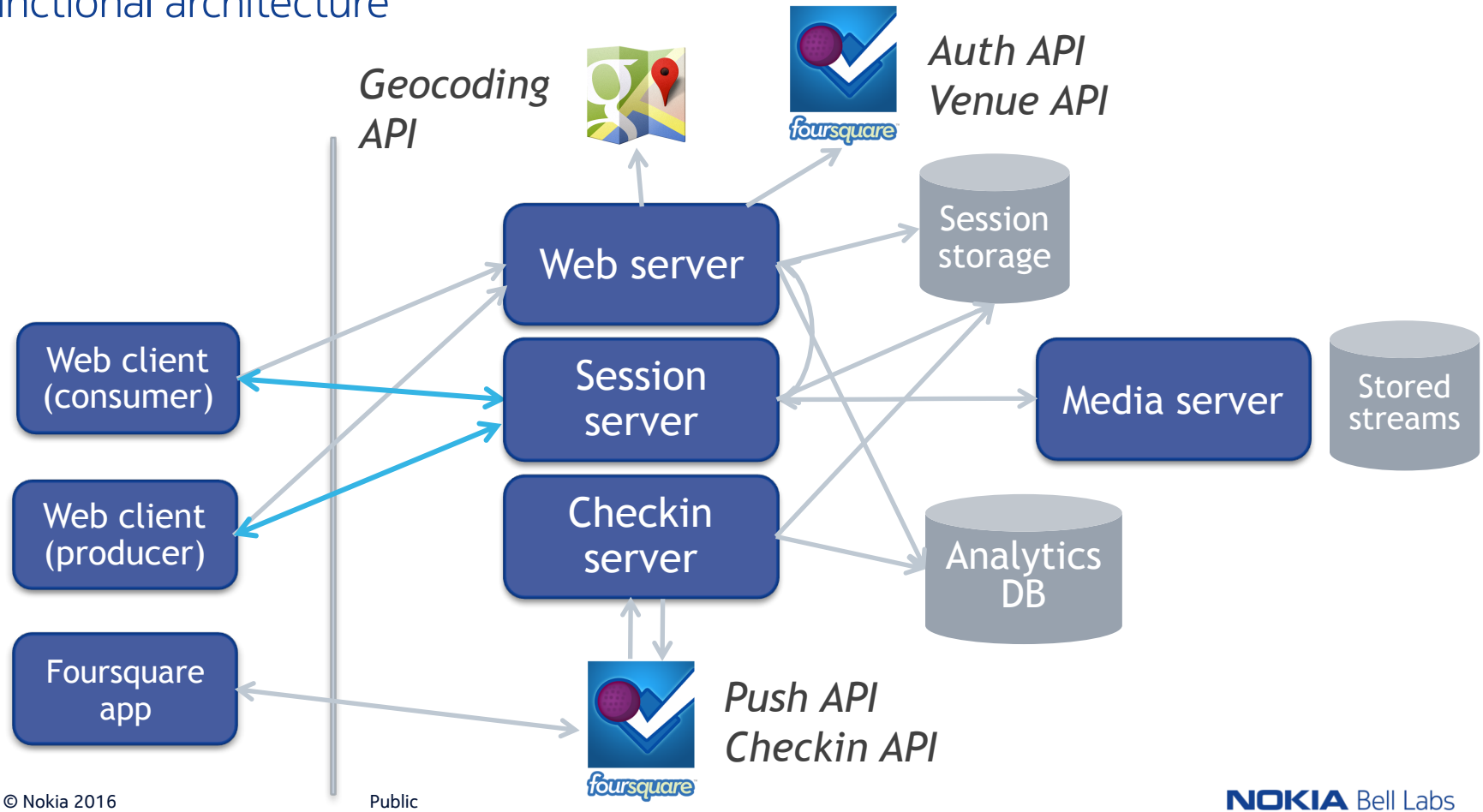
Prototype app + experience movie



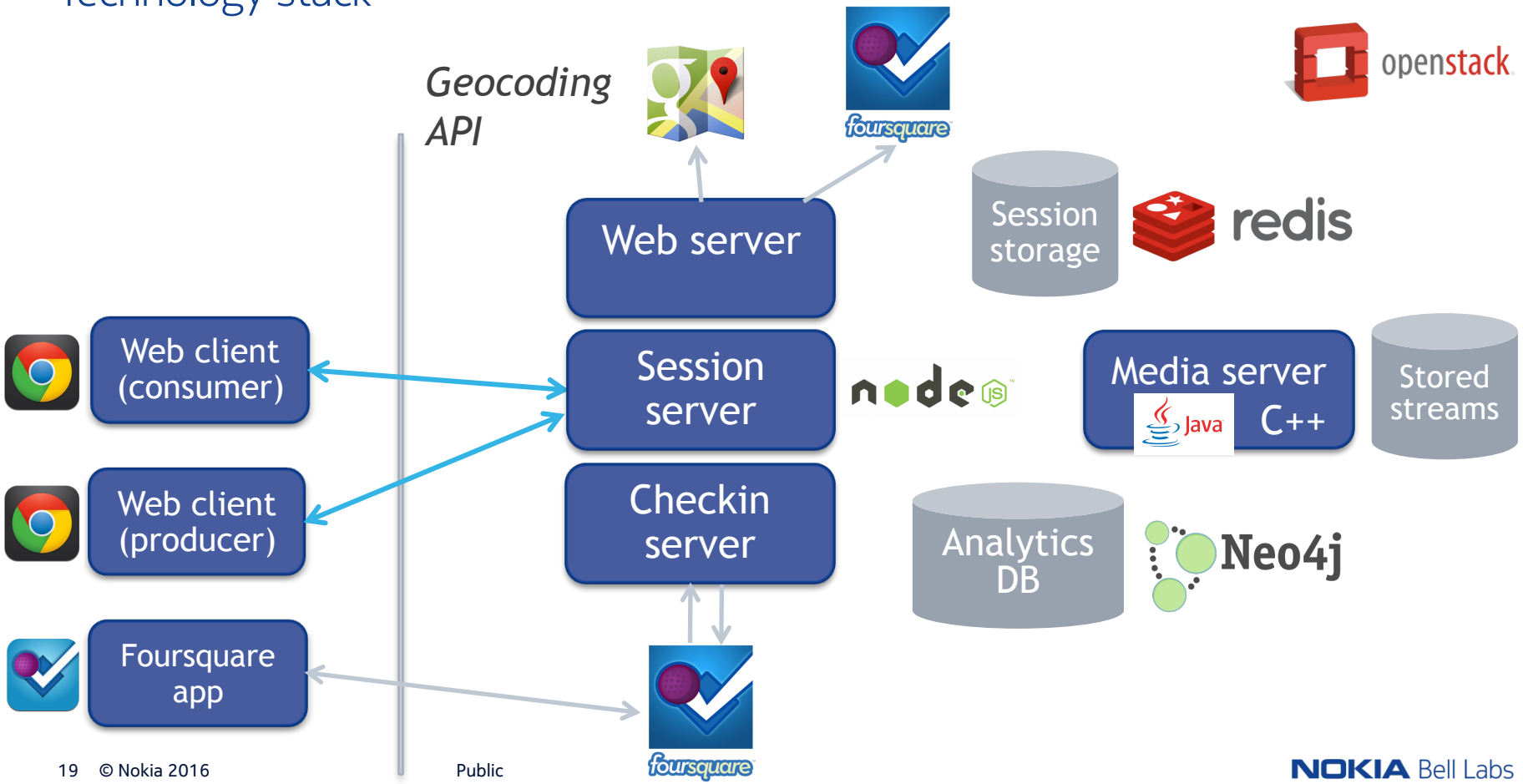
  Public

NOKIA Bell Labs

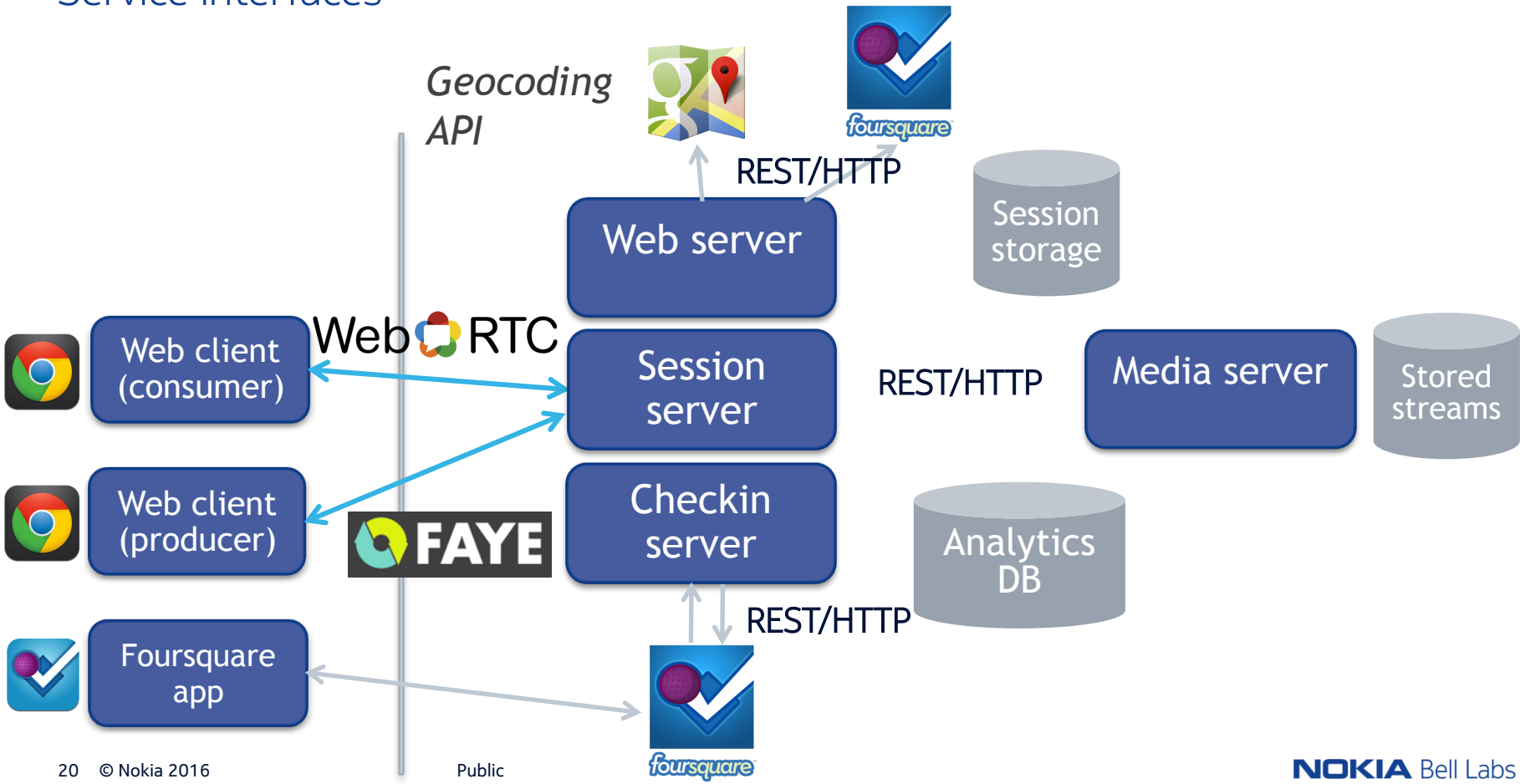
Functional architecture



Technology stack



Service interfaces



NOKIA Bell Labs

Lessons learned

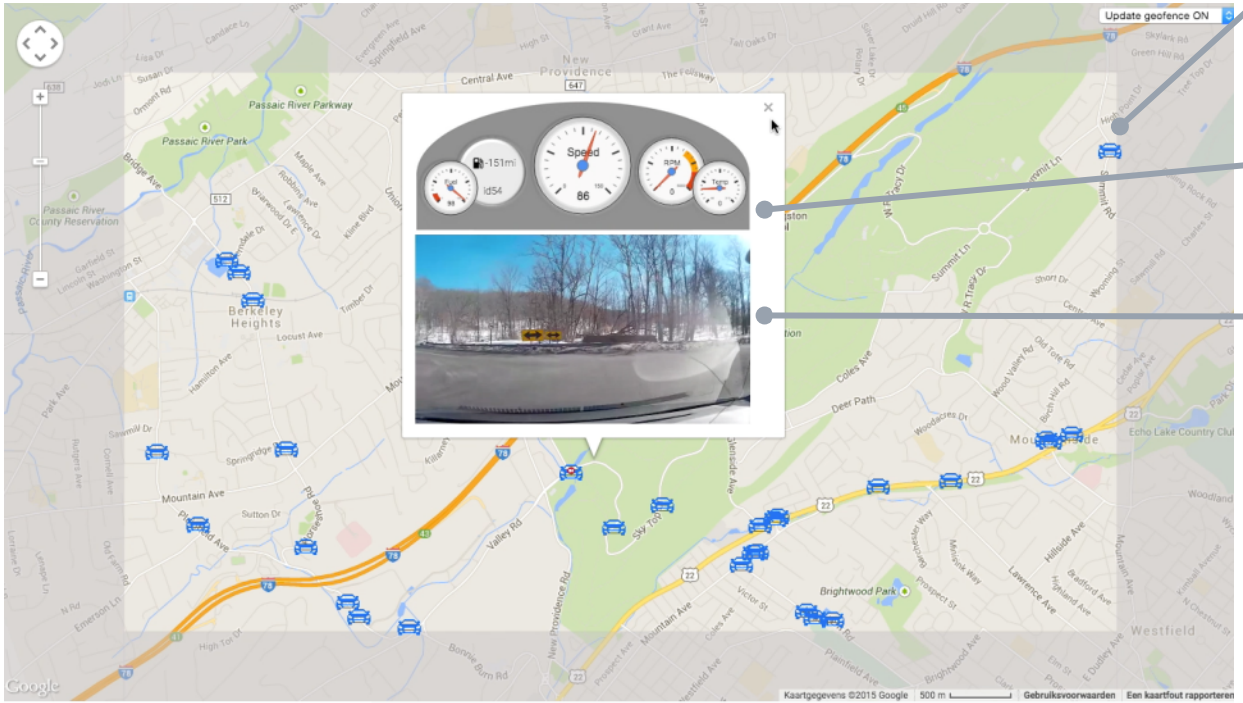
- Multiple teams working on independent subsystems = highly productive
  - Different goals
  - Different skillsets
  - Different release schedules
  - Less conflicts
- Testing and debugging of the overall system was a pain
- We didn’t sufficiently invest in tooling and automation
  - Manual configuration 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

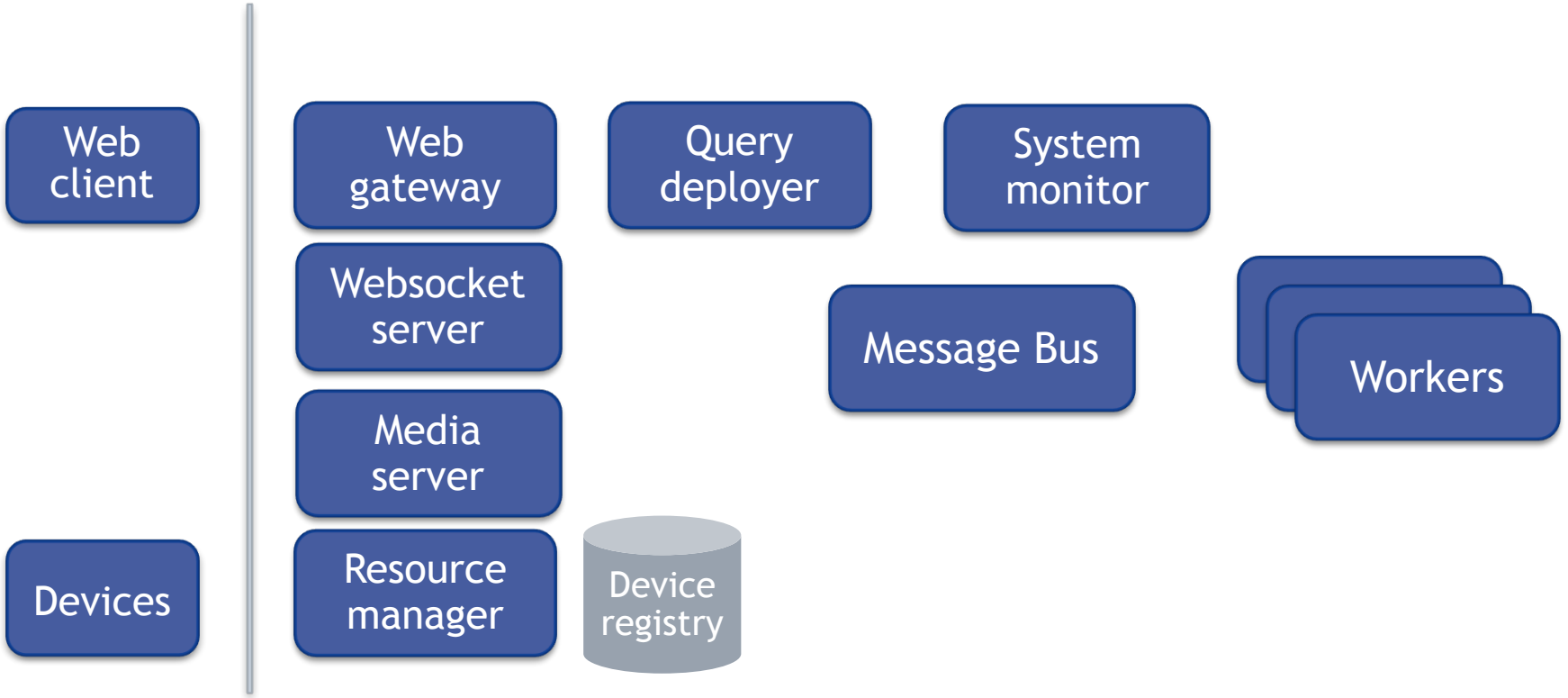
On-board Unit

2 real cars,
10 hours footage
400 virtual cars

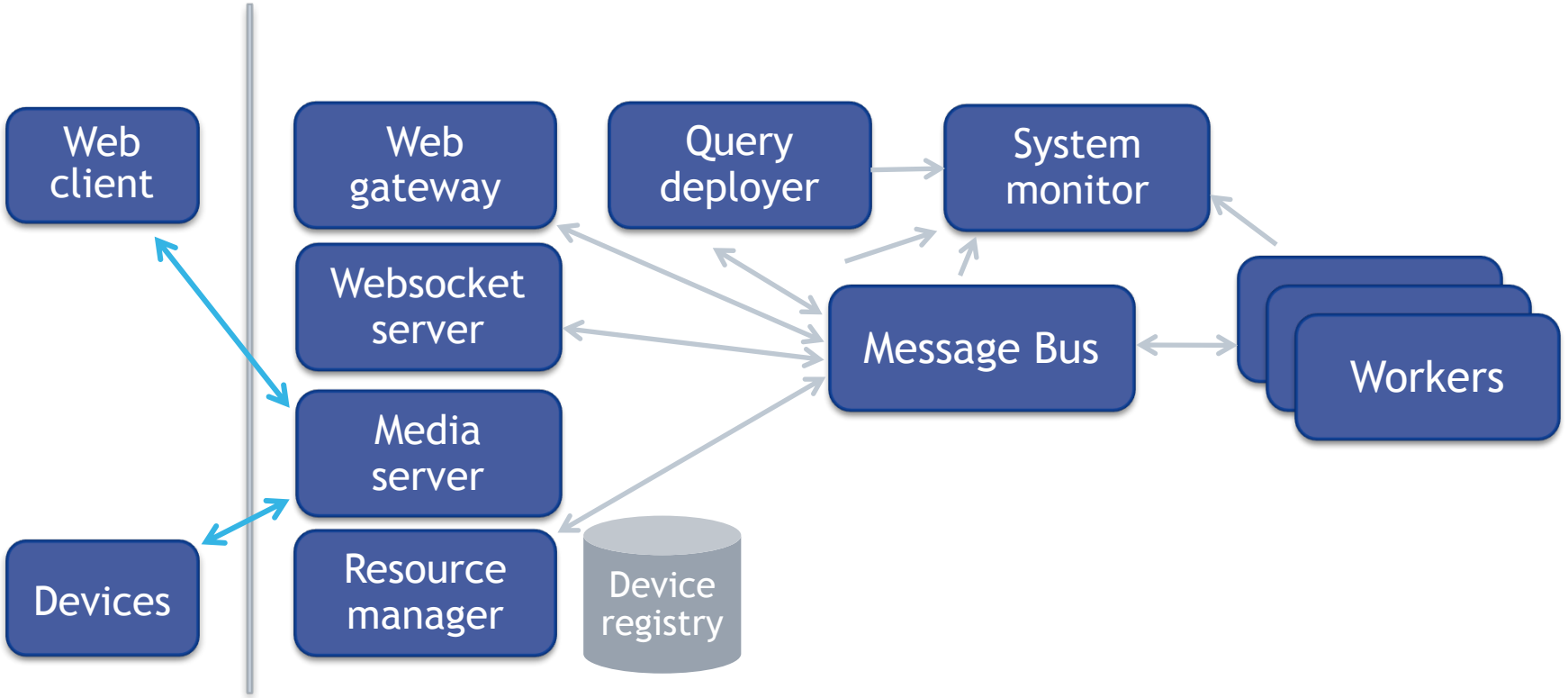
  Public

NOKIA Bell Labs

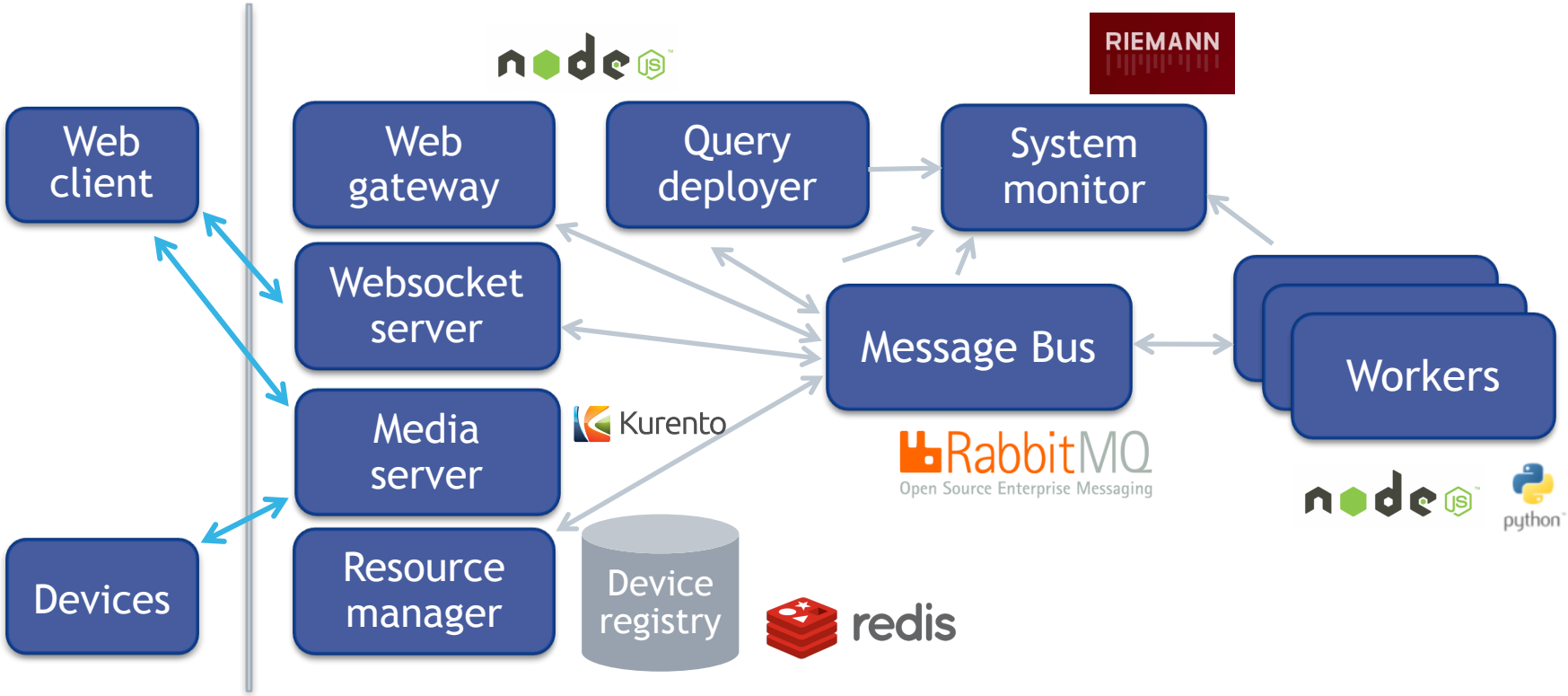
Instadash: functional architecture



Instadash: functional architecture

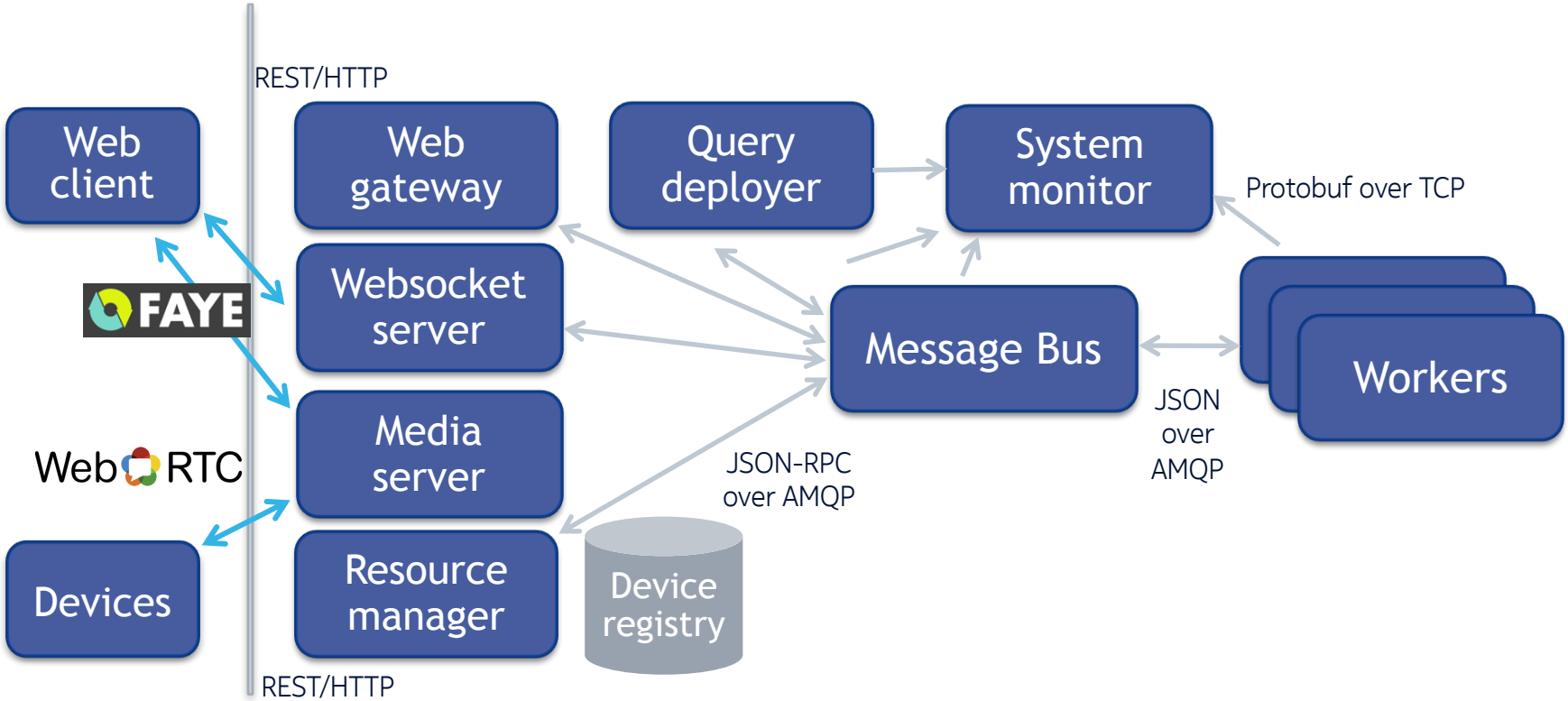


Instadash: technology stack

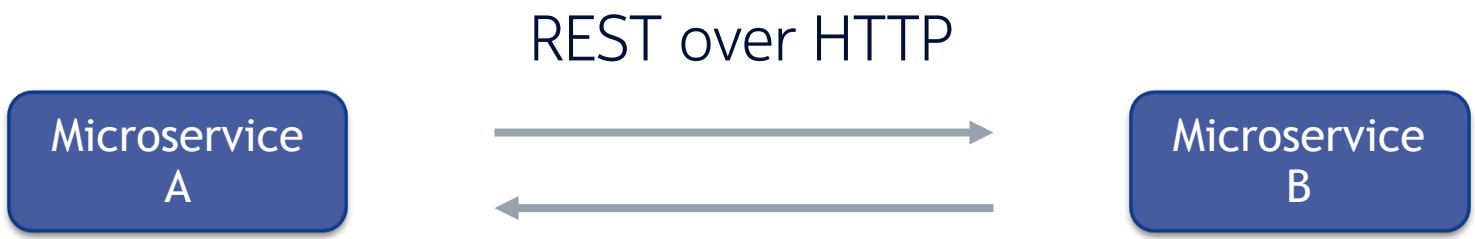


NOKIA Bell Labs

Instadash: service interfaces



Microservice communication patterns



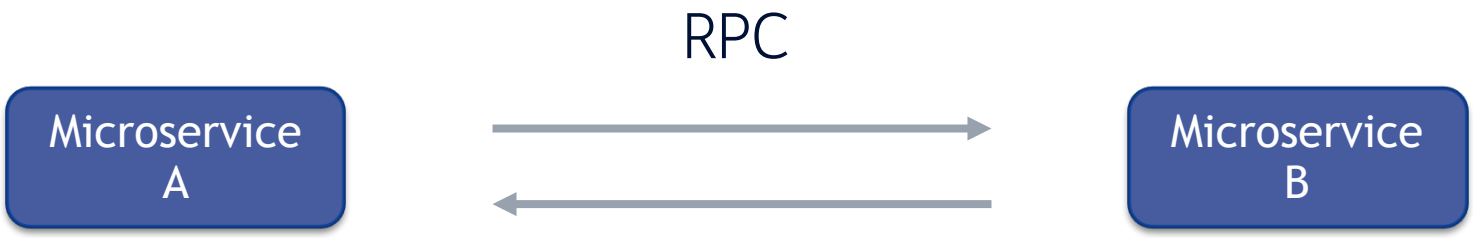
REST/HTTP is well-understood

HTTP support is ubiquitous

JSON as data model is a natural fit

Text-based protocol overheads

Microservice communication patterns



Fast, often binary encoding

Built-in schema support

Firewall issues, less ubiquitous

Need an additional discovery service

GRPC JSON-RPC AVRO ØMQ

Microservice communication patterns

Decoupling between components
(bus handles both discovery and routing)

More complex, beware bottleneck

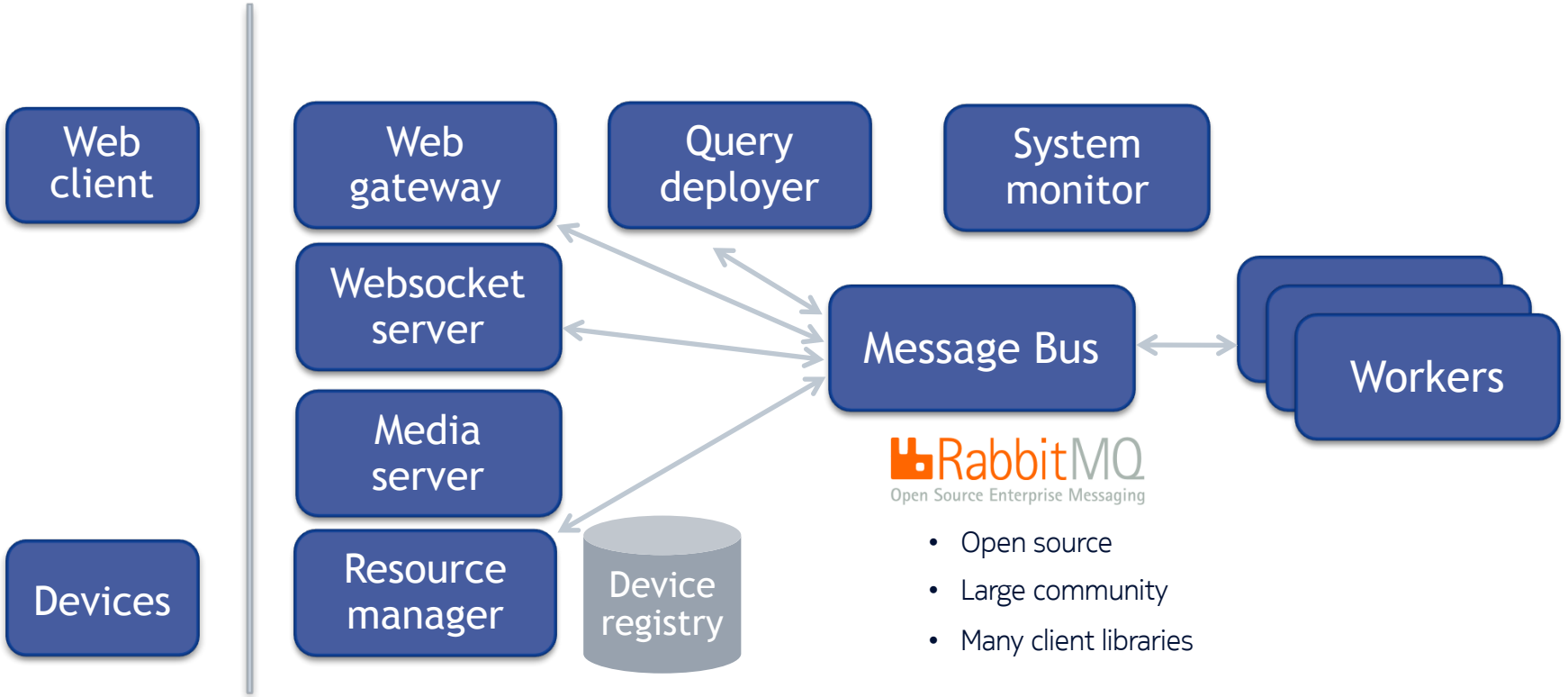
RabbitMQ
Open Source Enterprise Messaging

redis

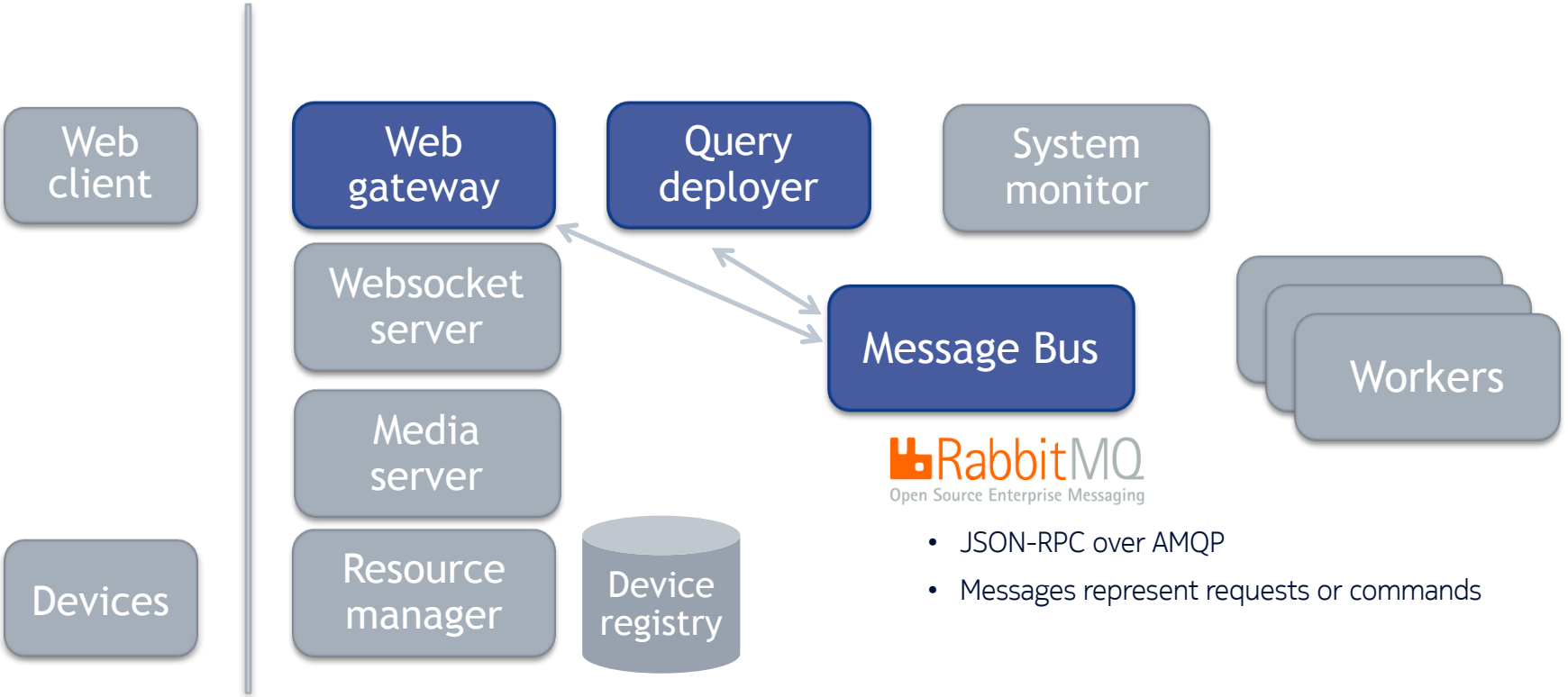
kafka

NATS

Communication patterns

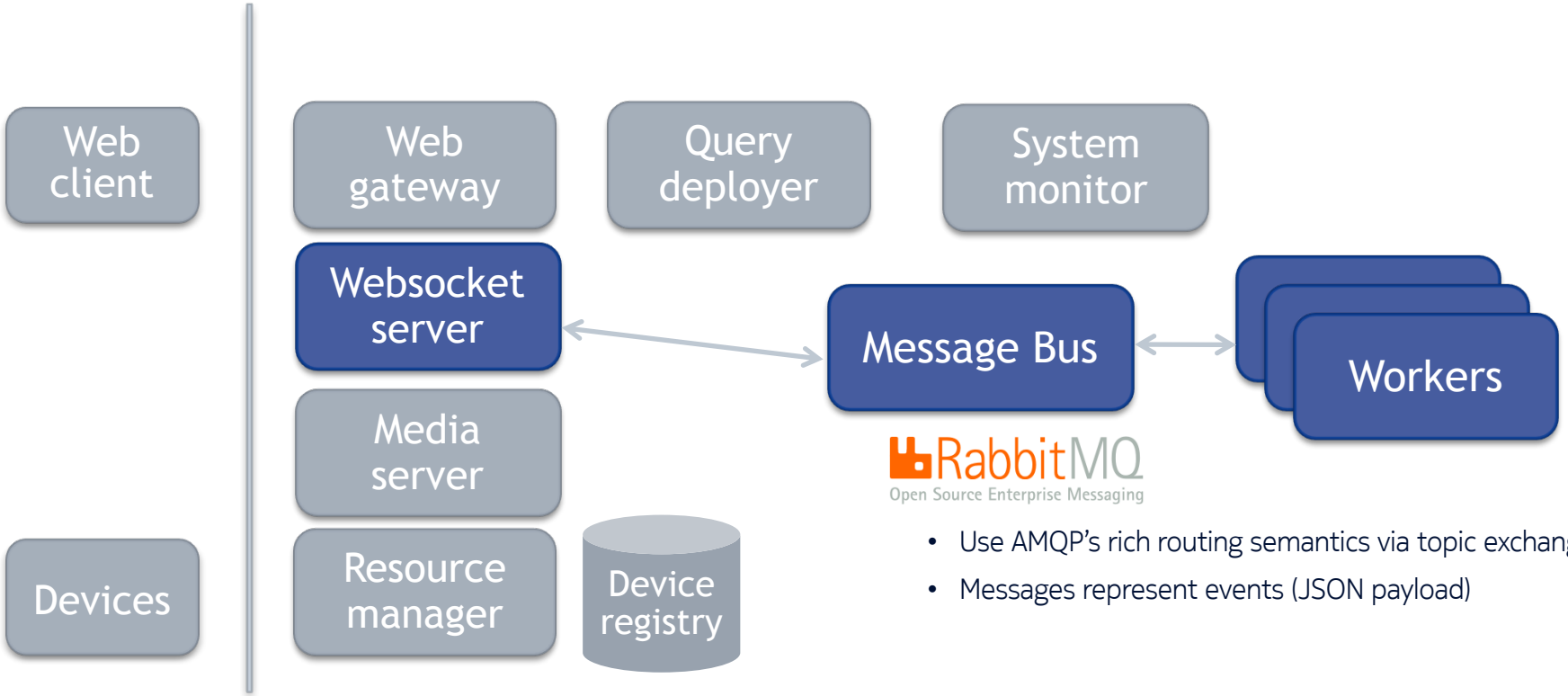


Communication patterns: point-to-point



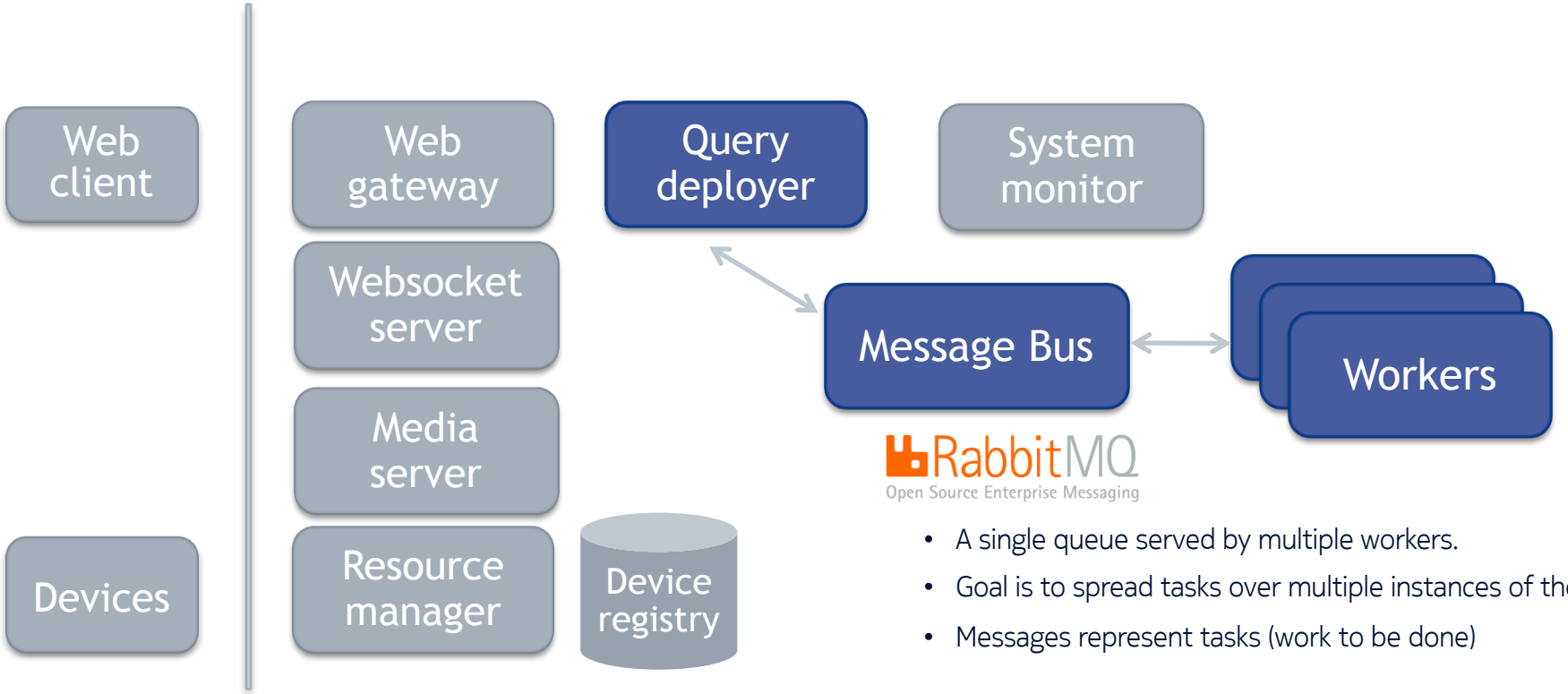
- JSON-RPC over AMQP
- Messages represent requests or commands

Communication patterns: publish-subscribe



- Use AMQP's rich routing semantics via topic exchanges
- Messages represent events (JSON payload)

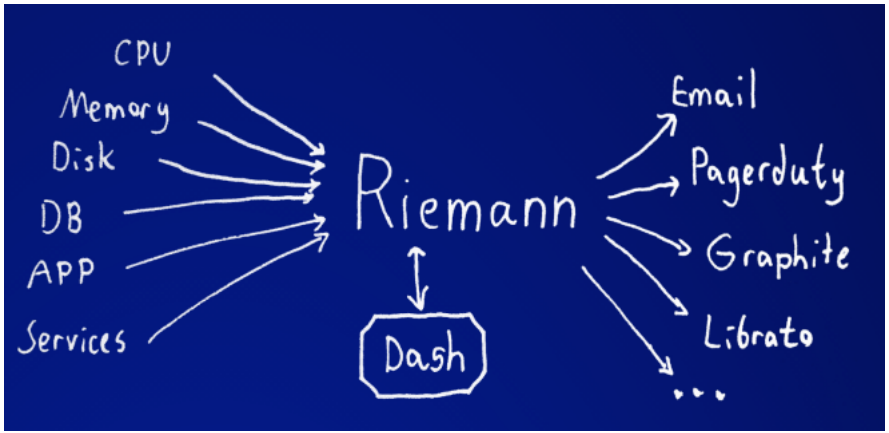
Communication patterns: work queueing



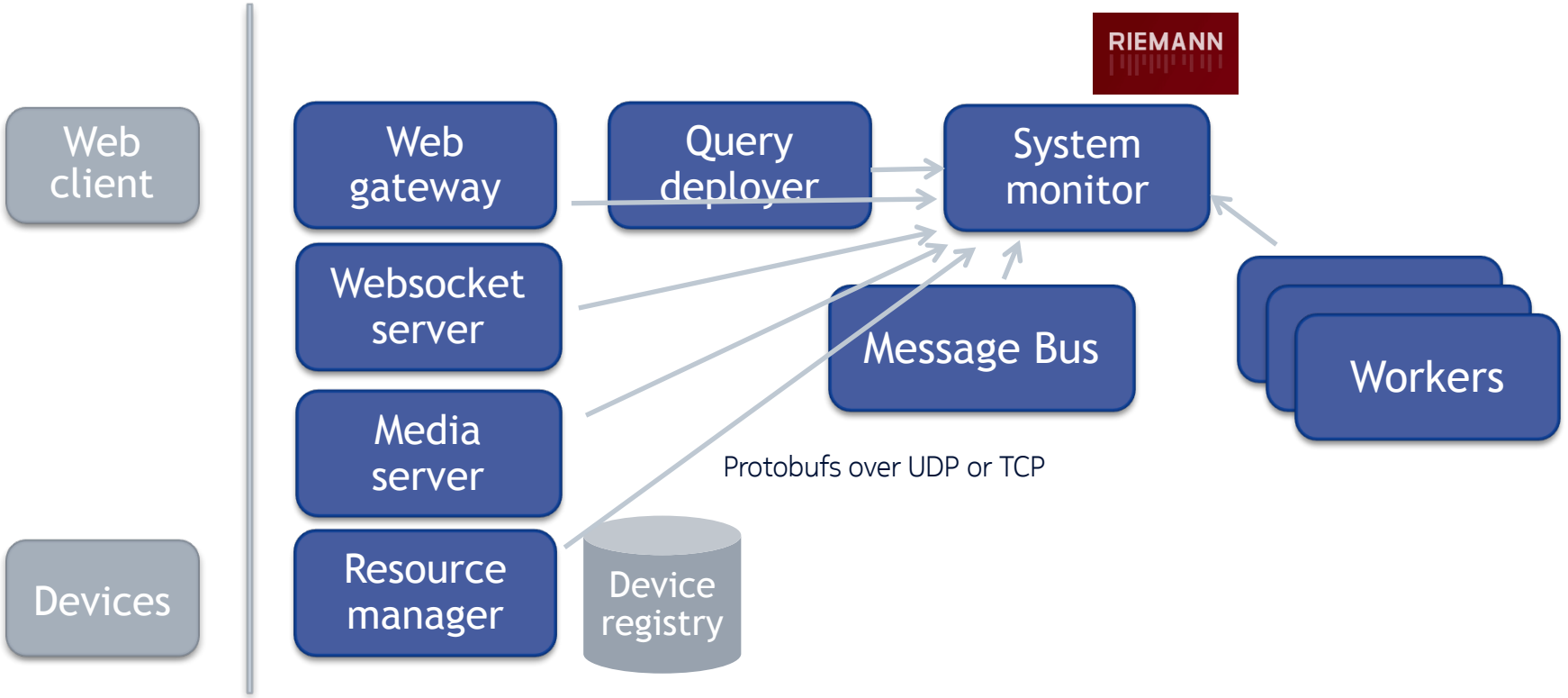
- A single queue served by multiple workers.
- Goal is to spread tasks over multiple instances of the same serv
- Messages represent tasks (work to be done)

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



NOKIA Bell Labs

Monitoring: dashboards

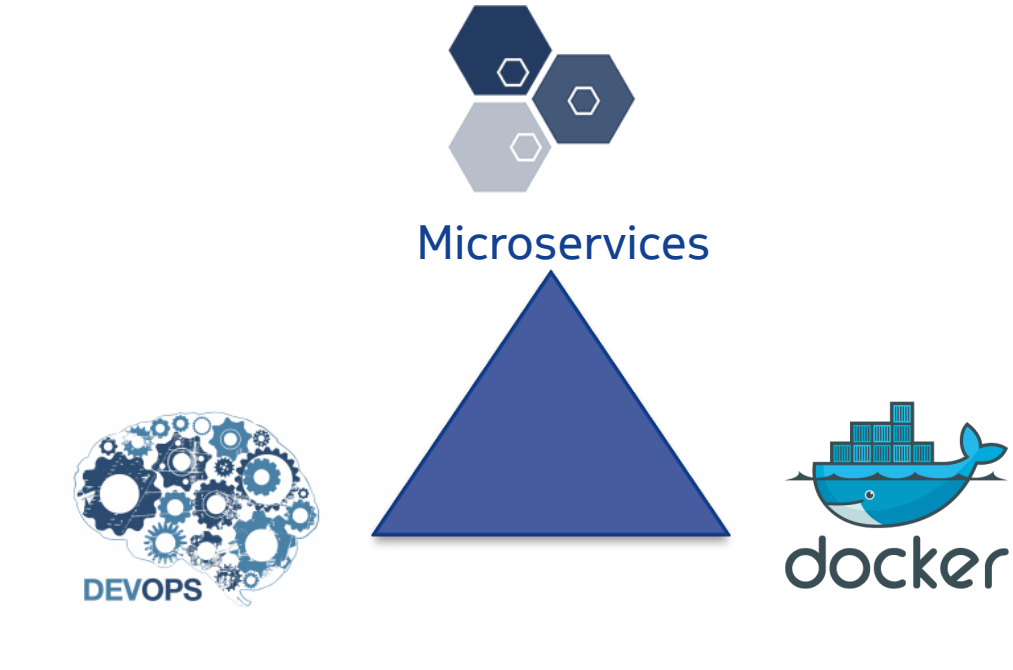
Lessons learned

- 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

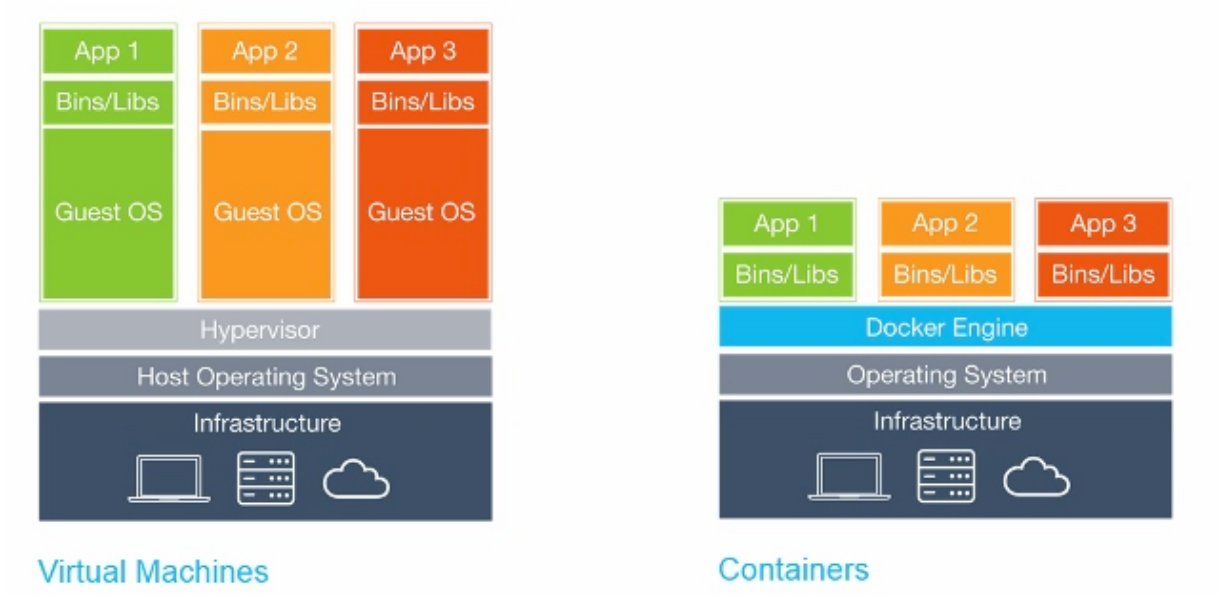
On MicroServices, Docker and DevOps

Context



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/

Docker Containers
Programmability

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

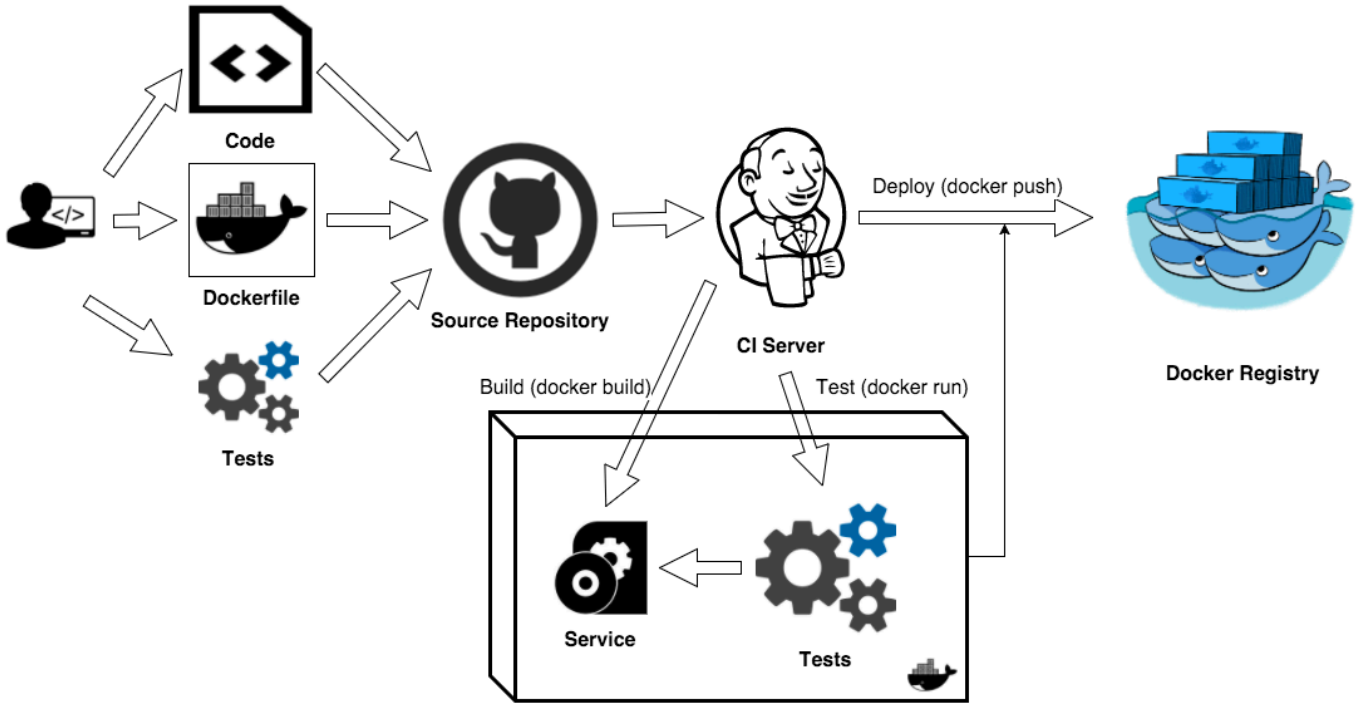
The matrix from hell

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

Docker Containers

Flexibility

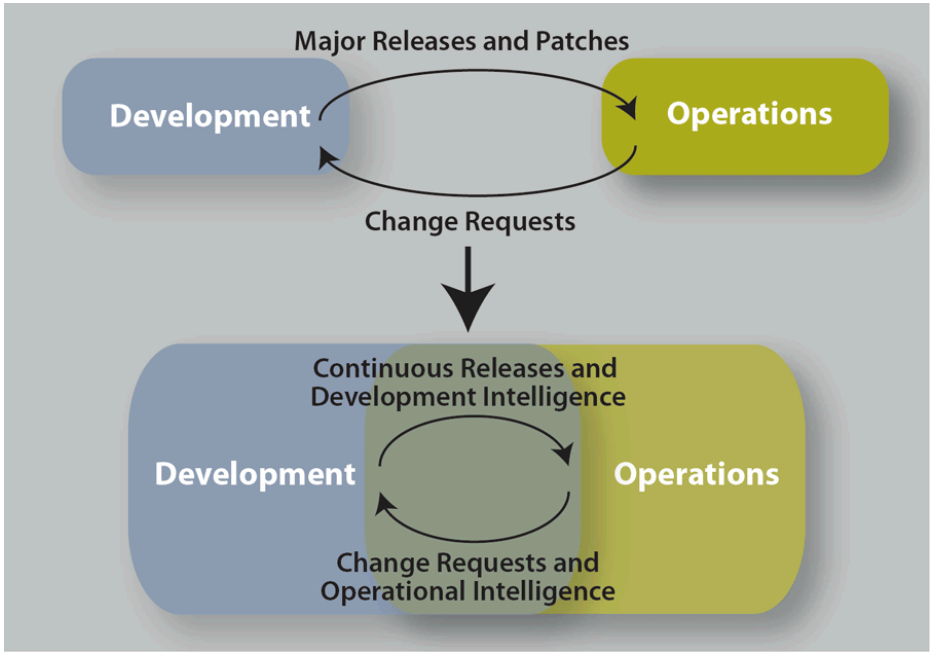
Build, ship and run any app, anyware [docker]



source: http://blog.terranillius.com/post/docker_testing/

Docker Containers
Demo

DevOps
@10k feet



source: http://www.drdobbs.com/architecture-and-design/getting-devops-right-the-lay-of-the-land/240062639

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

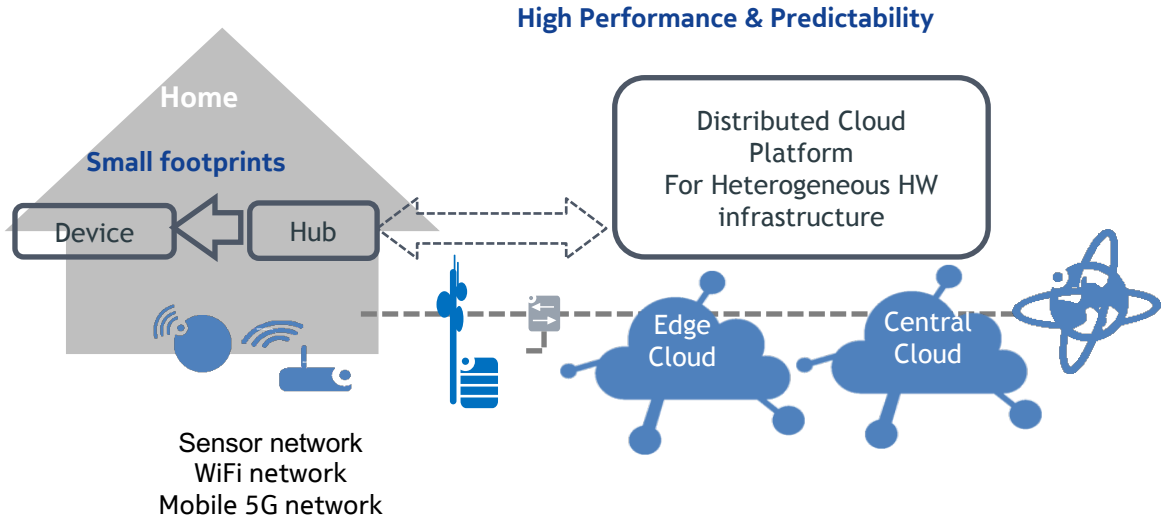
Portability

Micro-service architectures

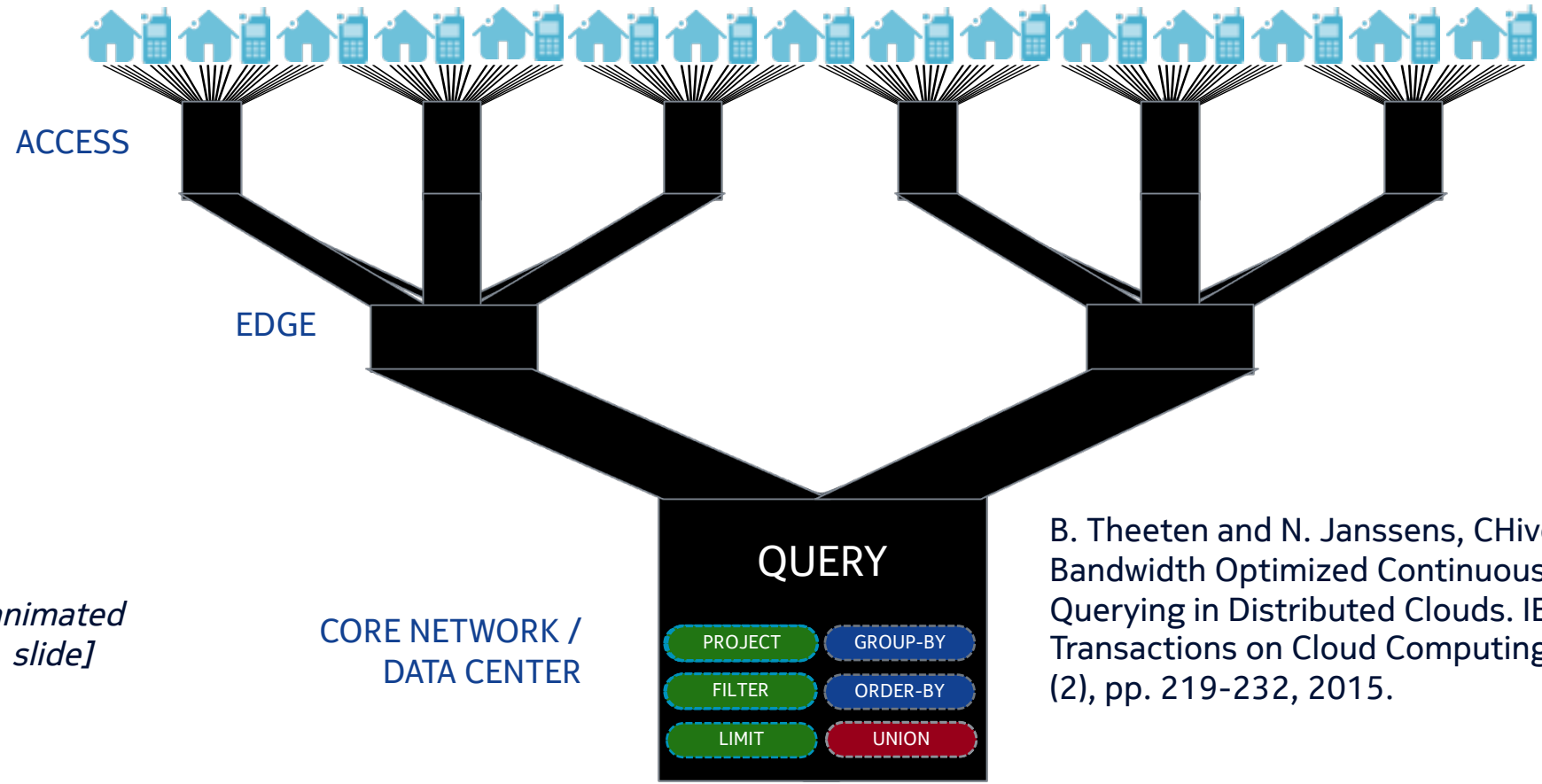
Active eco-system + public image registries

Facilitates DevOps methodology

Bell Labs Projects: New Home/IoT Service Platform



Bell Labs Projects: Bandwidth Optimized Streaming Analytics



[animated slide]

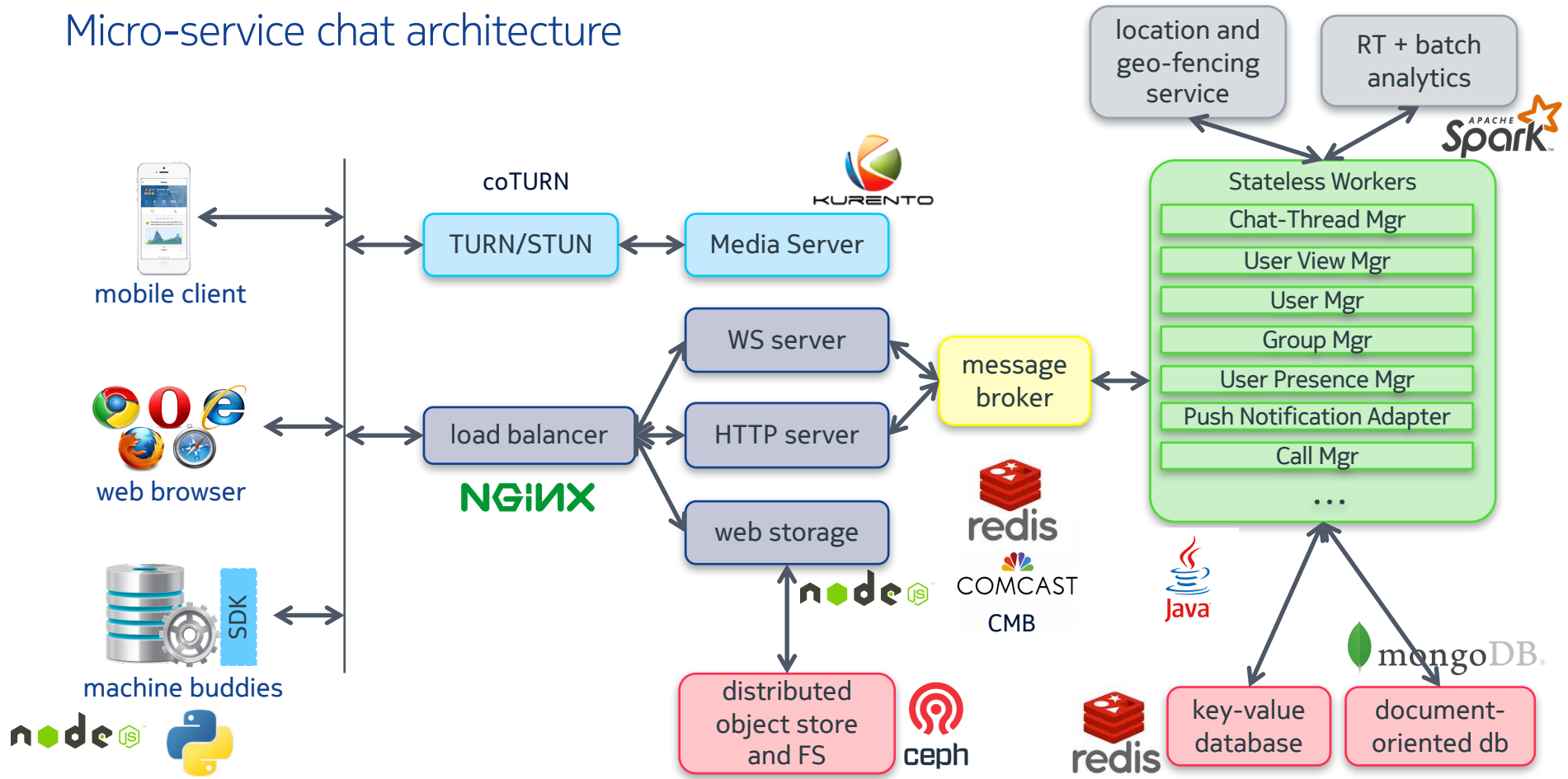
B. Theeten and N. Janssens, CHive: Bandwidth Optimized Continuous Querying in Distributed Clouds. IEEE Transactions on Cloud Computing 3 (2), pp. 219-232, 2015.

Bell Labs Projects: New Communication Service

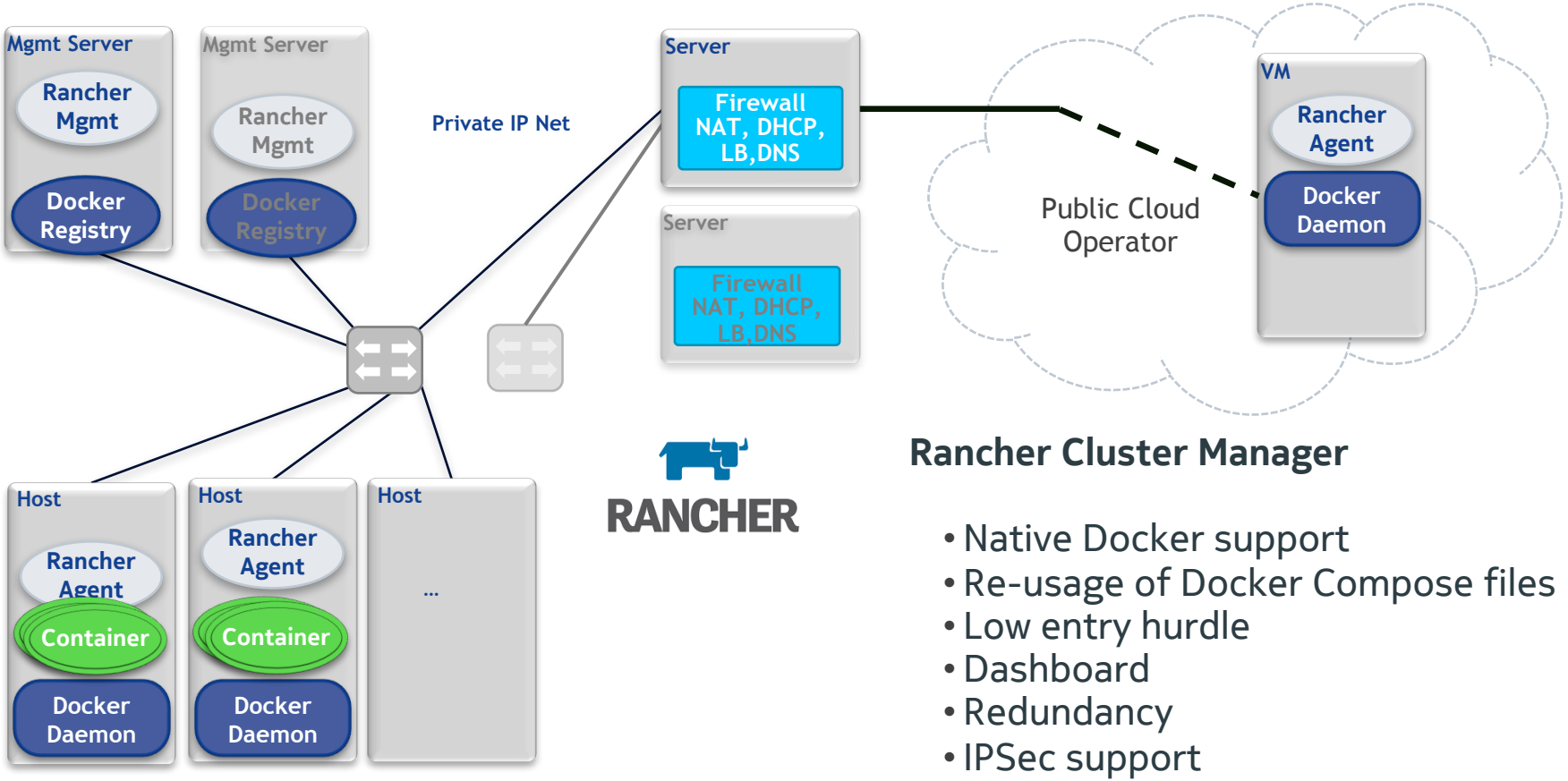
Key Goal: Simplify interactions among people, machines, and their environments

- 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



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

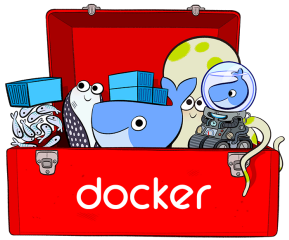
Low overhead ✔

Portability ✔

Micro-service architectures ✔

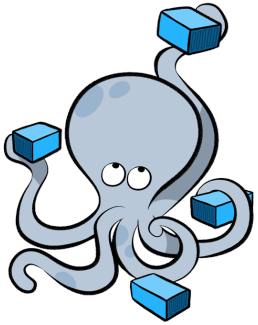
Active eco-system with public image registries ✔

Facilitates DevOps methodology ✔



toolbox

RANCHER


compose


registry

Evaluation
Docker

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/
- A. Cockcroft (prev. Netflix lead engineer) on migrating to micro-services: http://www.infoq.com/presentations/migration-cloud-native
- Insightful blogs:
  - http://www.tigerteam.dk/2014/micro-services-its-not-only-the-size-that-matters-its-also-how-you-use-them-part-1/
  - http://gomorpheus.com/blog/2014-10-24-the-new-reality-microservices-apply-the-internet-model-to-app-development
  - A critical note: http://contino.co.uk/microservices-not-a-free-lunch/
  - http://highscalability.com/blog/2015/12/1/deep-lessons-from-google-and-ebay-on-building-ecosystems-of.html
- Colossus (Tumblr Engineering Blog): http://engineering.tumblr.com/post/102906359034/colossus-a-new-service-framework-from-tumblr
- Finagle (Twitter Engineering Blog): https://blog.twitter.com/2011/finagle-a-protocol-agnostic-rpc-system

NOKIA

Copyright and confidentiality