Essence of the **Cloud**

from **Monolith** to **µServices**

Damian Adamowicz, © Gigaset 2016

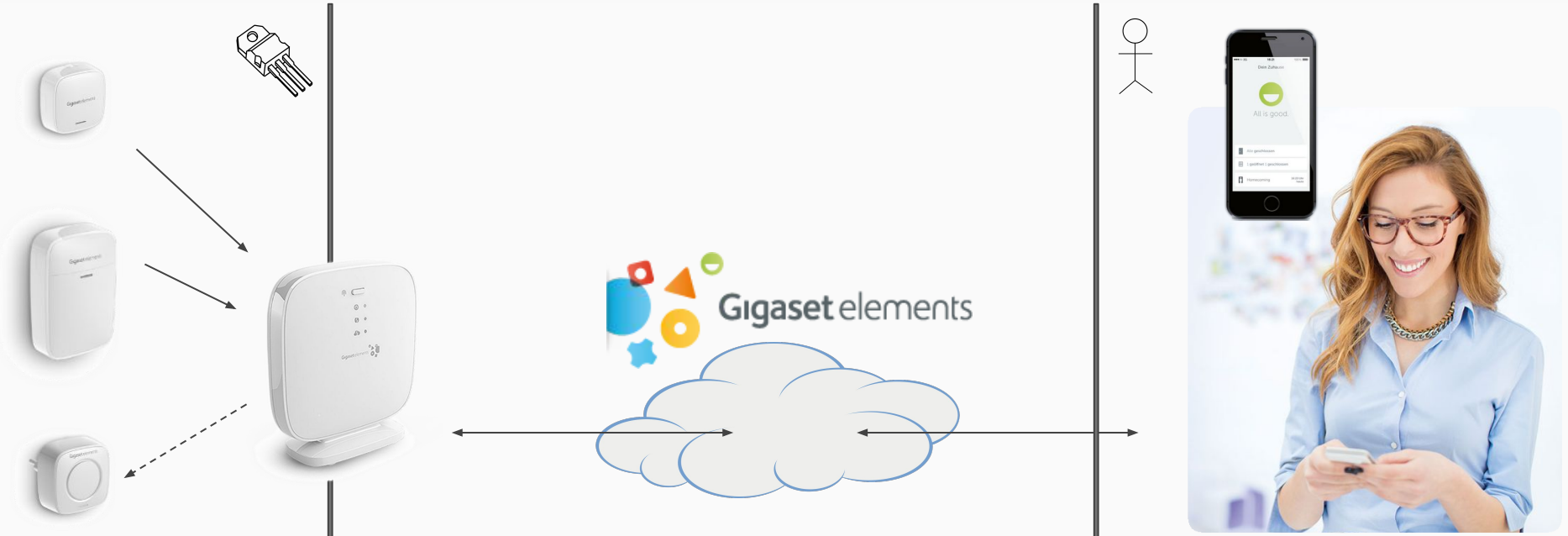
ver 20160830

Preface

- History based on **Gigaset Cloud**
- **Stripped** from sensitive or classified information
- **Simplified** for sake of presentation
- **Focused** on cloud part

Gigaset elements

Gigaset elements - Business Assumptions



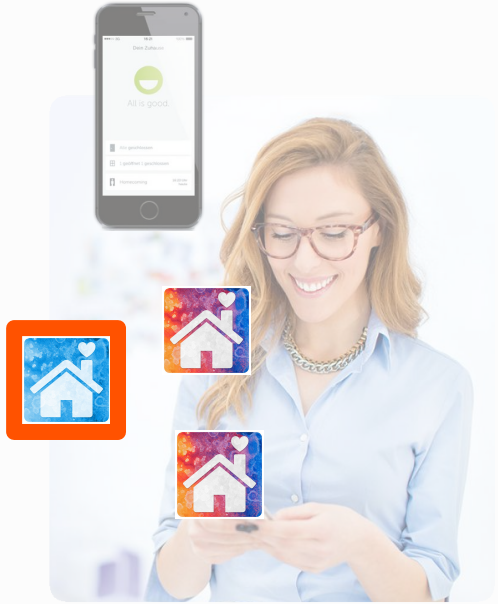
6LoWPAN @
DECT-ULE

GSM / WiFi

Gigaset elements - Organisation

separate independent company

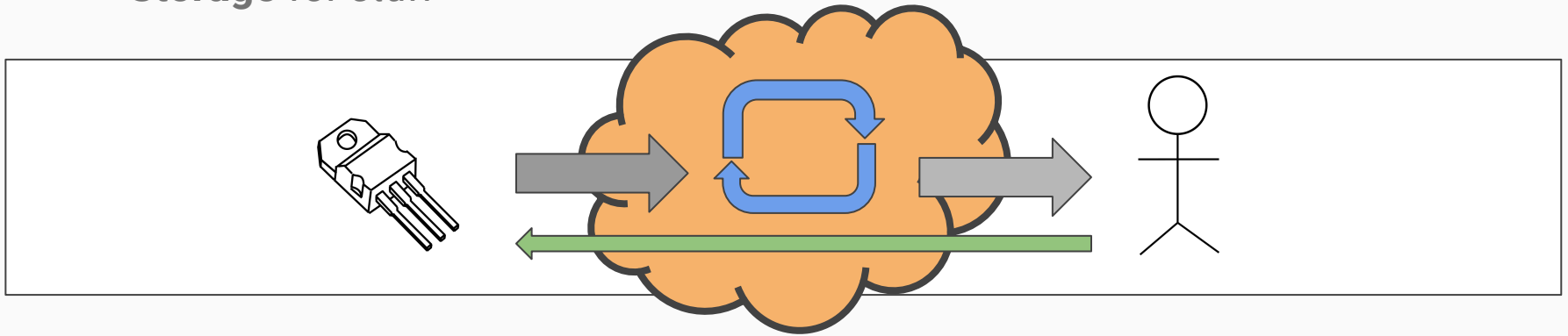
6LoWPAN @
DECT-ULE



GSM / WiFi

Requirements of **IoT Cloud**

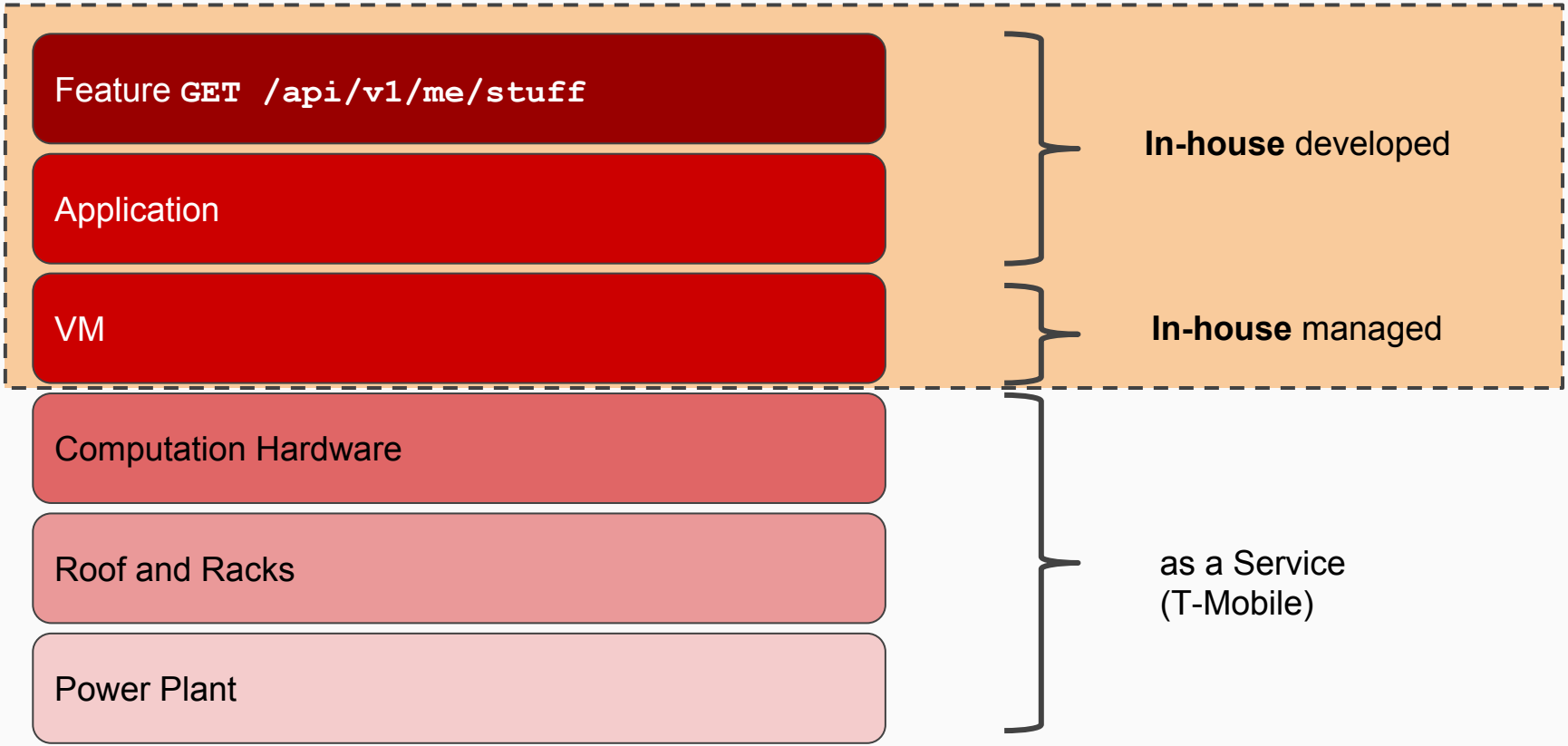
- **API** for frontend applications (because **User** wants to see what is going on)
- **API** for *things* (as in **IoT**; because we want input from stuff - to make other things to happen, for the **User**)
- *Fast* internal **event processing** (because **User** expects consequences at once)
- **Storage** for *stuff*



Cloud v1.0

Start - Nov 2012

Cloud 1.0 Layers

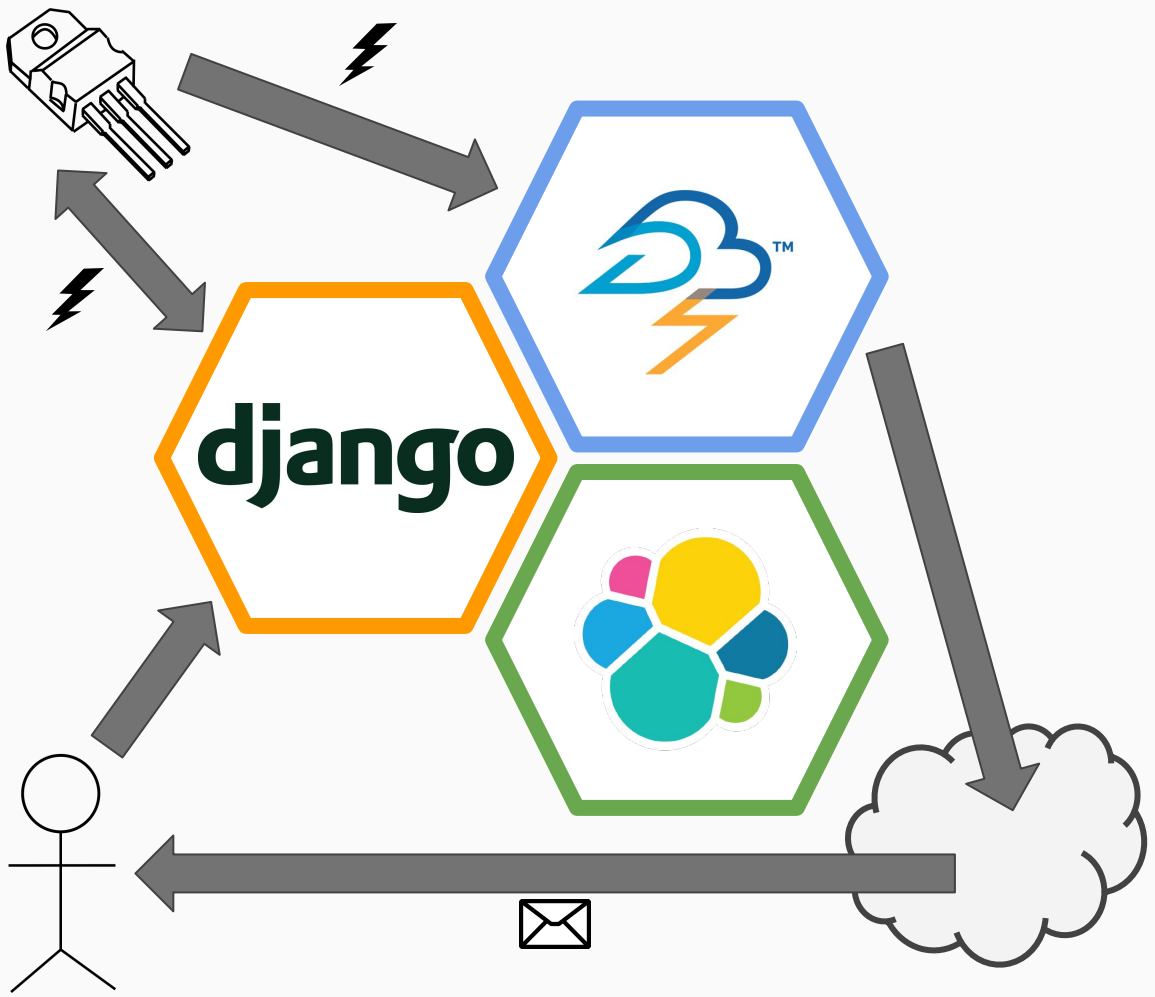


Compact Building Blocks

Event processing
Apache Storm

API & Management
django

Storage
elasticsearch



Cloud 1.0

a.k.a - compact disaster

Results

- **distributed** logic (django/storm)
- **mixed** languages (python/java)
- **elasticsearch** as **multipurpose DB** (r/w vs u/r)
- **hidden dependencies**
- **manual deployment**, limited HA and stability

Vigeland monolith, Oslo

Why it does **not scale**?

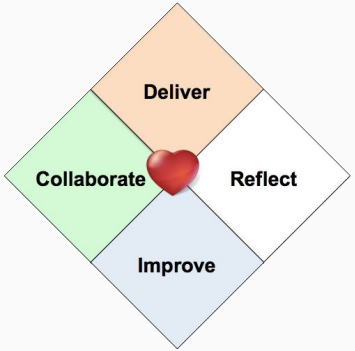
Technology

- **Architecture** principles **went bad**
- **Hard** to **integrate** deliveries
- **Hard** to **monitor** and diagnose
- Rare **deployments** lead to **big-bang** and **clash**

Why it does **not scale**?

Organisation

- Agile/**SCRUM** adaptation **went bad**
- **Ops** Team introduced **too late**
- **Knowledge transfer** transferred “knowledge”
- **Additional resources** did **decreased velocity**



http://alistair.cockburn.us/Rediscovering+the+Heart+of+Agile

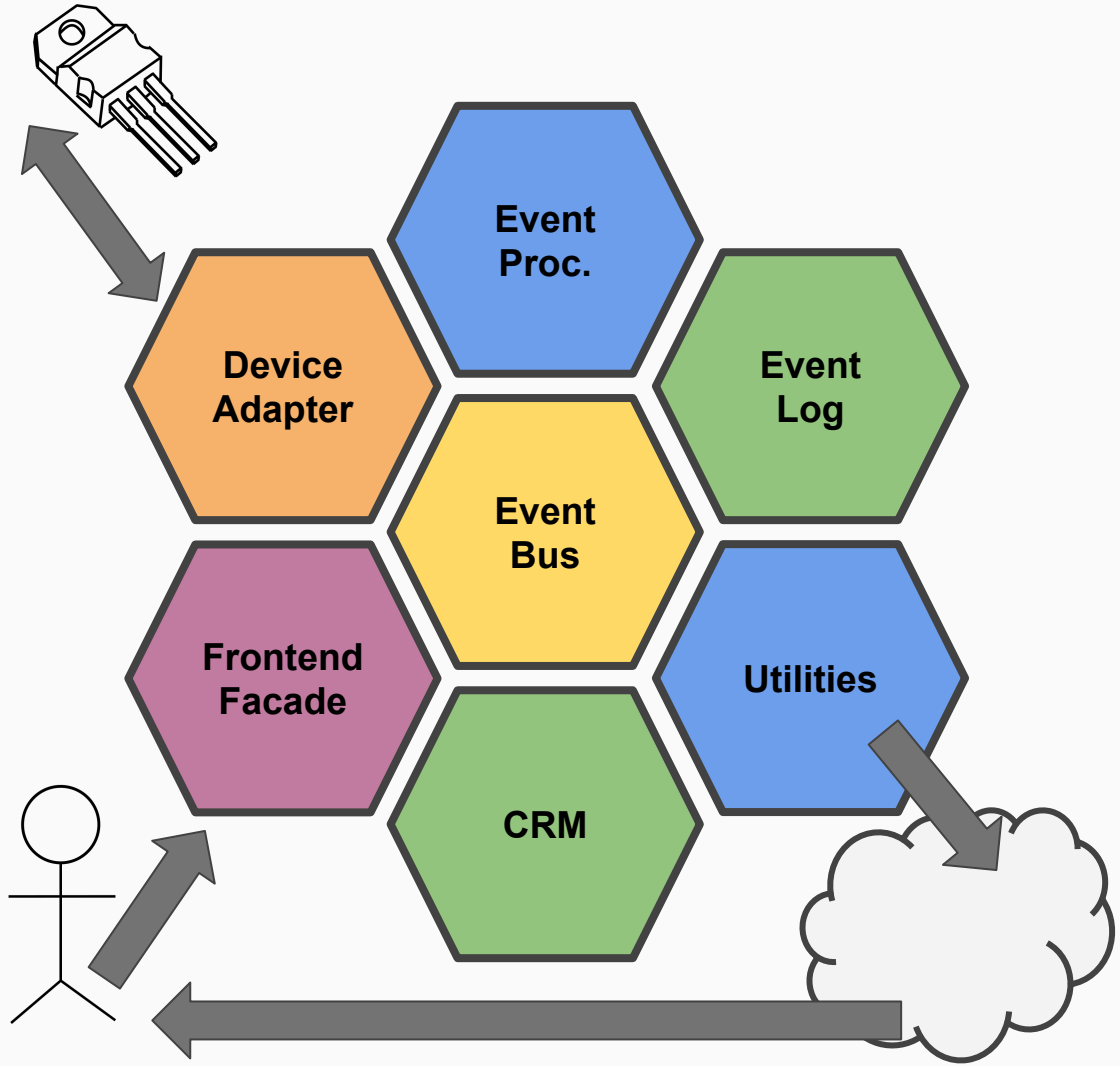
Cloud 2.0

Mar 2015

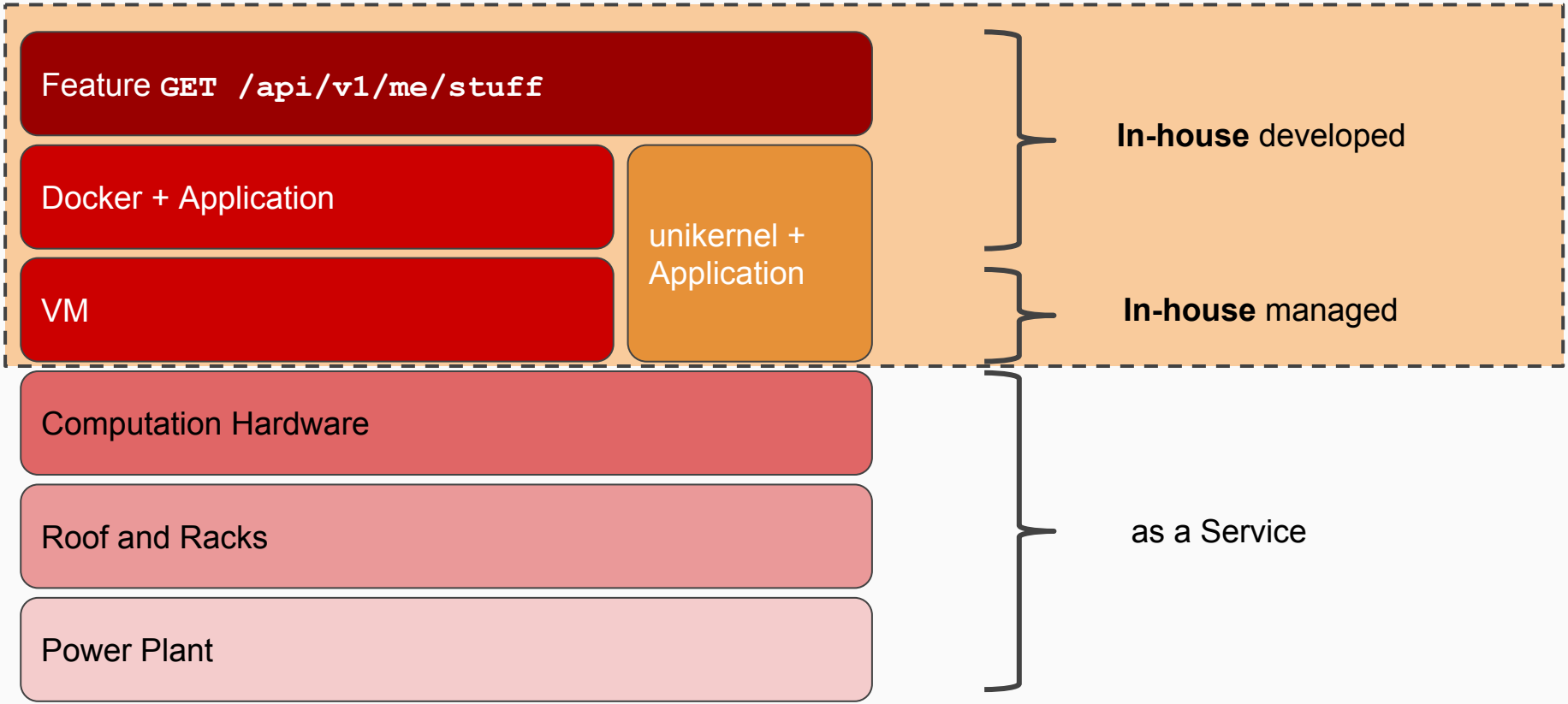
Abstract Building Blocks

Classic **SOA**

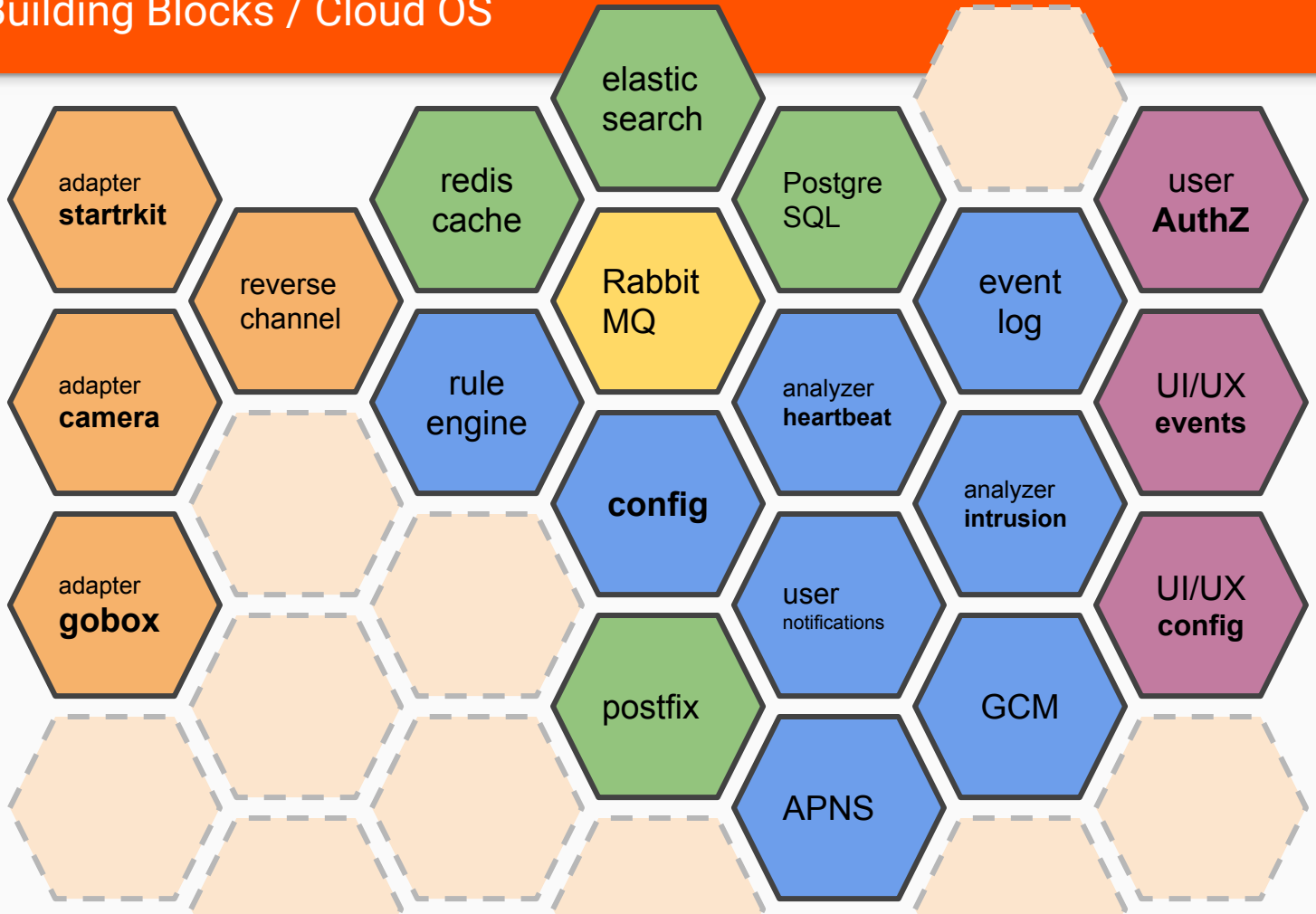
- isolated **microservices**
- agreed `HTTP REST` contract
- **utility** services (i.e. **DB**)



Cloud 2.0 Layers



Building Blocks / Cloud OS



play akka

elastic

RabbitMQ

redis

PostgreSQL

NGINX

docker

CentOS

Cloud 2.0 - µServices Swarm

- semi-**automatic** docker-ized **deployment**
- **feature**-to-**service** mapping
- isolation makes enhancement easier
- even low-level changes (i.e. glibc security update) work smoothly

- but … **Continuous Versioning** is still not there ;-)

Why it does **scale**?

Technology

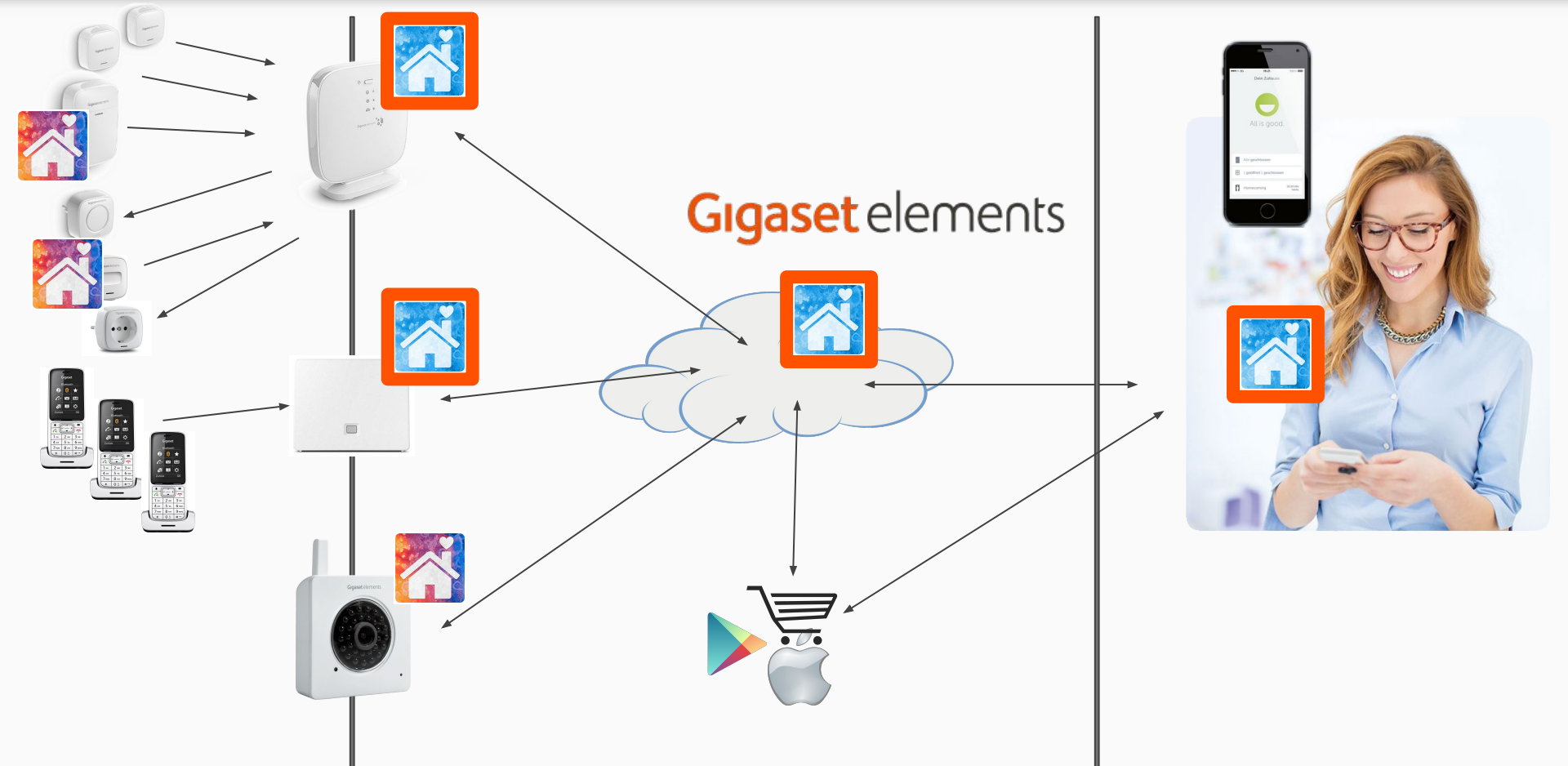
- Solid overall system **Architecture** with hints from **Ops**
- **µServices make** feature delivery **less complex**
- **parallel** feature **development** and **deployment**
  - **80** deployments/month (prev: **0.3 - 1**)

Why it does **scale**?

Organisation

- Redesigned **Teams** (**domains** vs **features**)
- Recognized **DevOps** initiative
- but … communication channel is still the limit

Gigaset elements



Summary

- **Monolith** can hit you from any angle
- **Scaling teams** is equally important to **scaling backend services**

Cloud 3.0

__future__

Cloud Layers 3.0

Feature `GET /api/me/stuff`

In-house developed

Really, do not care ;P

http://look-cuter.myshopify.com/products/no-fox-given

Cloud 3.0 - **Features** Only

- build **features**, not **infrastructure**
- out-of-the-box tools (**aws** lambda, sns / **google** cloud functions, pubsub)
- **scaling** as a service
- maybe even **Continuous Versioning**

Drawbacks

- vendor **lock**
- custom **utility** services

Q&A

damian.adamowicz@gigaset.com

https://www.linkedin.com/in/damianadamowicz

http://www.gigaset-elements.com/en/

https://www.gigaset.com/pl_pl/cms/gigaset/kariera.html

Gigaset