Service Discovery 101

Stefan Achtsnit

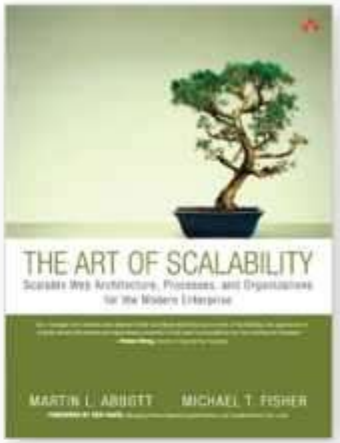
Polyglot Developer Meetup Vienna
2016

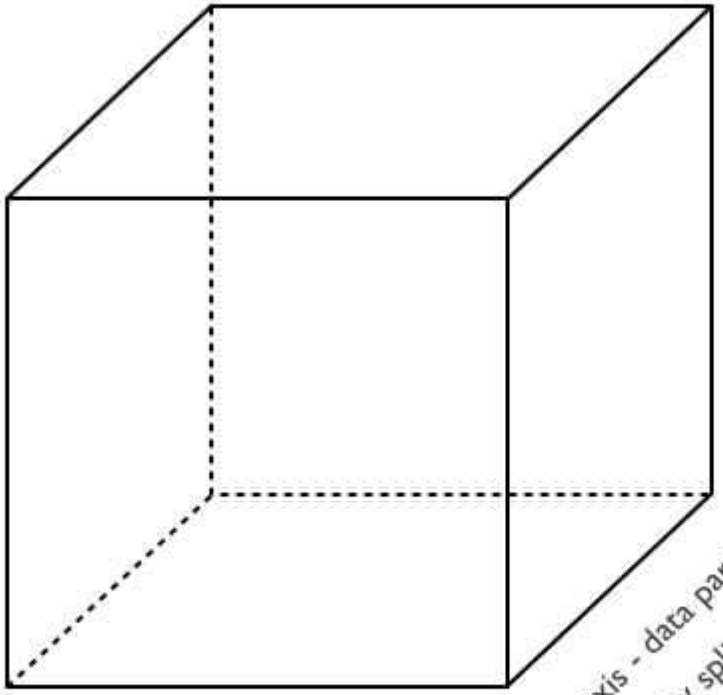
Outline

- The need for Service Discovery
- Service Discovery strategies
- Demo time

Scale Cube

3 dimensions to scaling





Y axis - functional decomposition

Scale by splitting different things

Z axis - data partitioning

Scale by splitting similar things

X axis - horizontal duplication

Scale by cloning

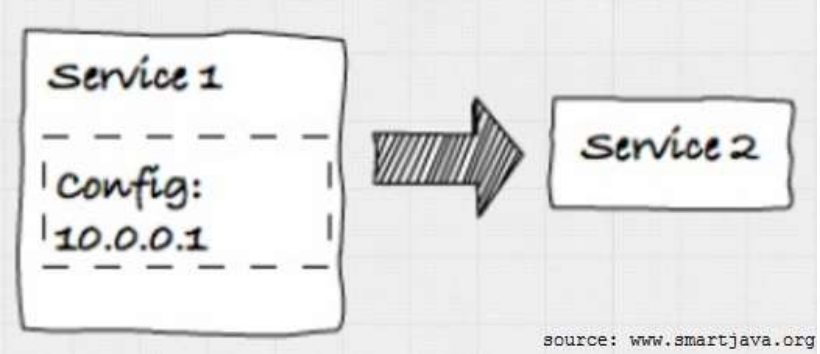
Microservice Architecture

- scale via functional decomposition
- mediate client communication
- design for eventual consistency
- expect services to come and go (this talk!)
- automate deployment
- expose runtime statistics and metrics

Expect services to come and go!

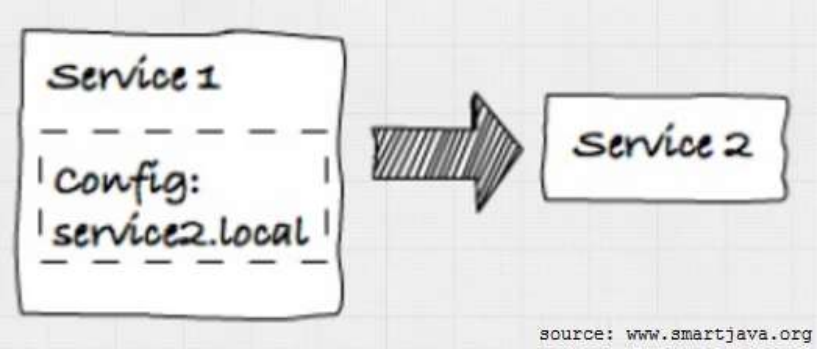
- auto-scaling, i.e. set of service instances may change dynamically
- server failures, i.e. service instances have dynamically assigned network locations
- upgrades/configuration changes

Hardcoded IP



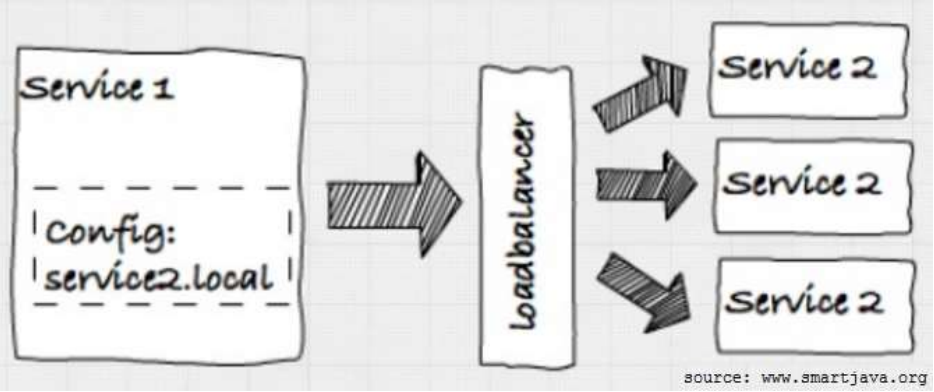
- doesn‘t scale
- not resilient to failures
- config locality

DNS



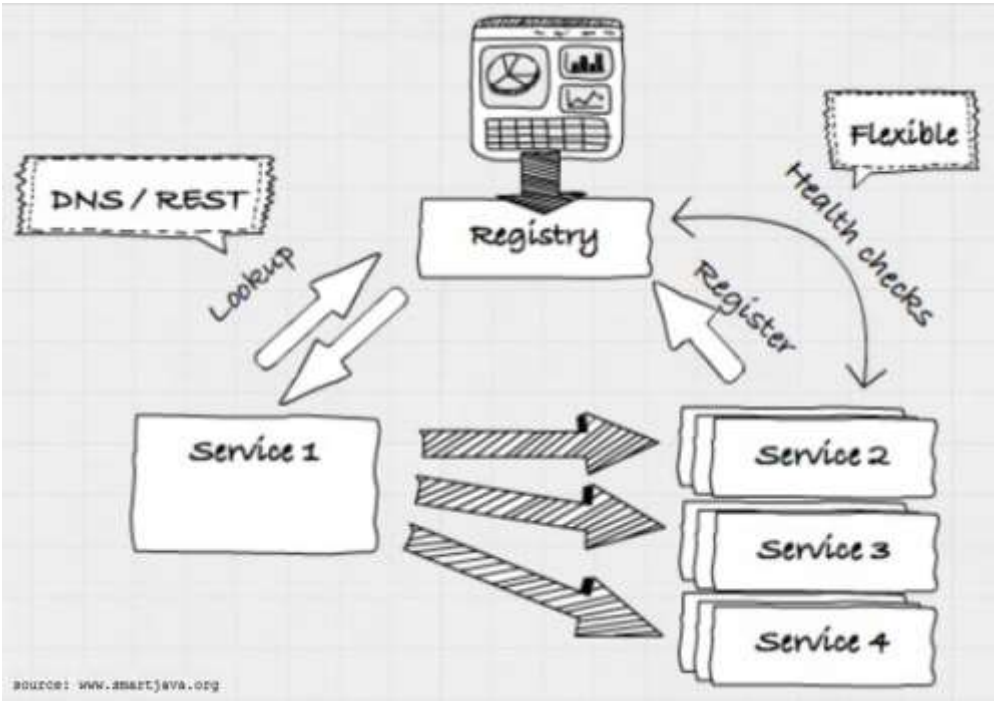
- doesn‘t scale
- not resilient to failures
- DNS server required

DNS with Loadbalancer



- scale out
- failure detection possible (loadbalancer support?)
- loadbalancer new single point of failure
- dynamic load balancer reconfiguration not trivial

Service Registry



- scale out (fully automatable)
- failure detection
- distributed key-value store for configuration

Service Discovery Patterns

- client-side vs server-side discovery
- self-registration vs third-party registration

Check out

- https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/
- http://microservices.io/

Demo time

- consul (dead-man switch)
- consul-template (HAProxy)
- envconsul

Summary

A Service Registry is necessary for robust systems, as

- services may be started and stopped in any order,
- servers may fail/be restarted,
- configuration may be changed!

Recommendation: Consul by HashiCorp

Thank you!