Infrastructure As Microservices

Alternatives to the Monolithic Kubernetes

DevOps Gathering, 10.03.2020

Über uns

Sandra Parsick

- Freelance Software Developer
- Softwerkskammer Ruhrgebiet, Oracle Groundbreaker Ambassador

Schwerpunkte

- Development of Java Enterprise Applications
- Automation von Development Processes


mail@sandra-parsick.de


@SandraParsick


xing.to/sparsick

Über uns

Nils Bokermann

- Freelance Software Developer
- Chemist :-)

Schwerpunkte

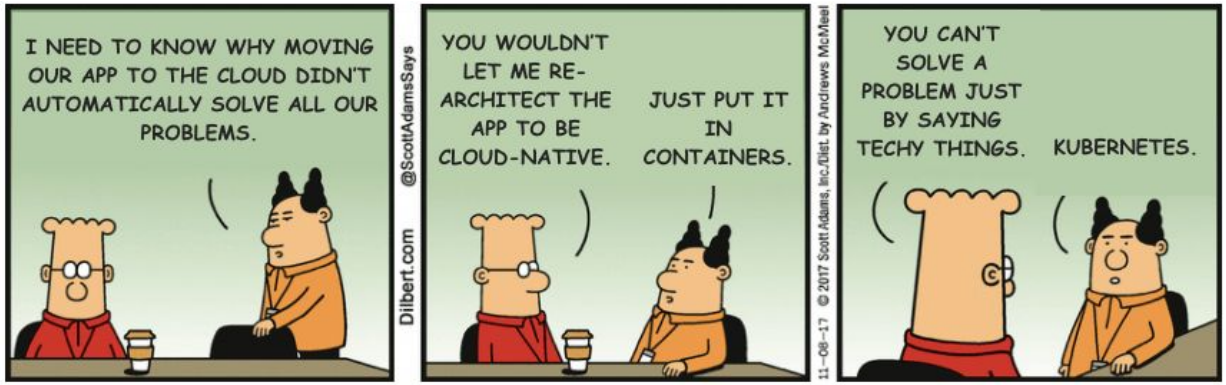
- Development of Java Enterprise Applications
- End-to-End Responsibility


info@bermuda.de


@sanddorn


xing.to/sanddorn

Motivation

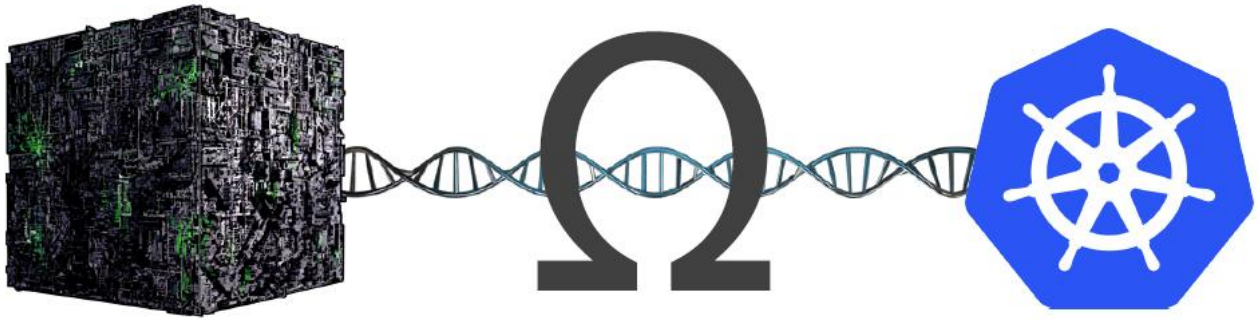

I NEED TO KNOW WHY MOVING OUR APP TO THE CLOUD DIDN'T AUTOMATICALLY SOLVE ALL OUR PROBLEMS.
Dilbert.com
@ScottAdamsSays
YOU WOULDN'T LET ME RE-ARCHITECT THE APP TO BE CLOUD-NATIVE.
JUST PUT IT IN CONTAINERS.
11-06-17 © 2017 Scott Adams, Inc./Dilbert by Andrews McMeel
YOU CAN'T SOLVE A PROBLEM JUST BY SAYING TECHY THINGS.
KUBERNETES.

Professionell Software Development

We think that Professional Software Development shall include the economical success for the business produced by the Software we create.

History of Kubernetes

History of Kubernetes



Bildquelle:
https://blog.risingstack.com/the-history-of-kubernetes/

Exkurs Container

- Container sind „chroot auf Steroiden“
- Es ist keine VM light.
- Nur ein Service (ein Prozess)

Container Historie

1979 Unix V7 implementiert den chroot-system call

1982 BSD übernimmt den chroot-system call

2000 FreeBSD Jails (FreeBSD 4.0)

2001 Linux VServer

2004 Solaris Container

2005 OpenVZ

2006 Process Containers

2008 LXC (LinuX Containers)

2011 CloudFoundry Warden

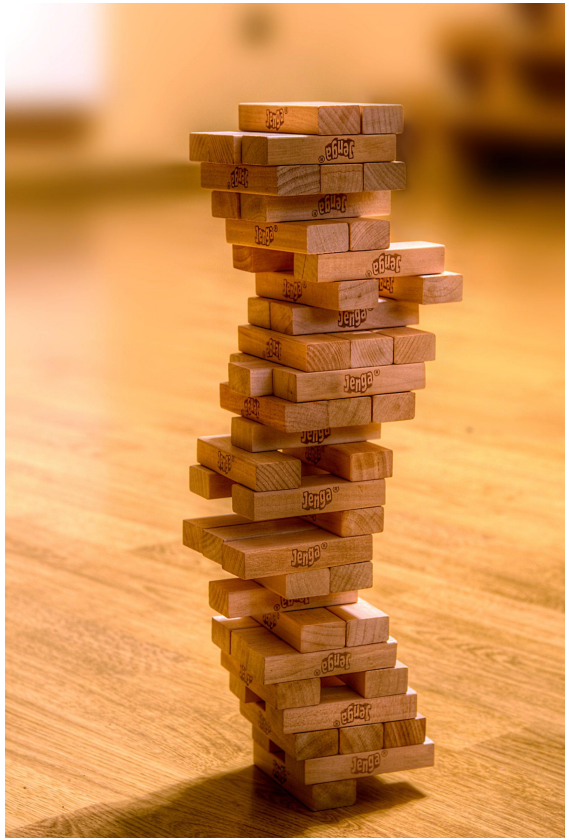
2013 LMCTFY

2013 Docker

2014 kubernetes

Why do we call Kubernetes a Monolith?

See Our Definition of a Monolith



Bildquelle: By Guma89 - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17999924

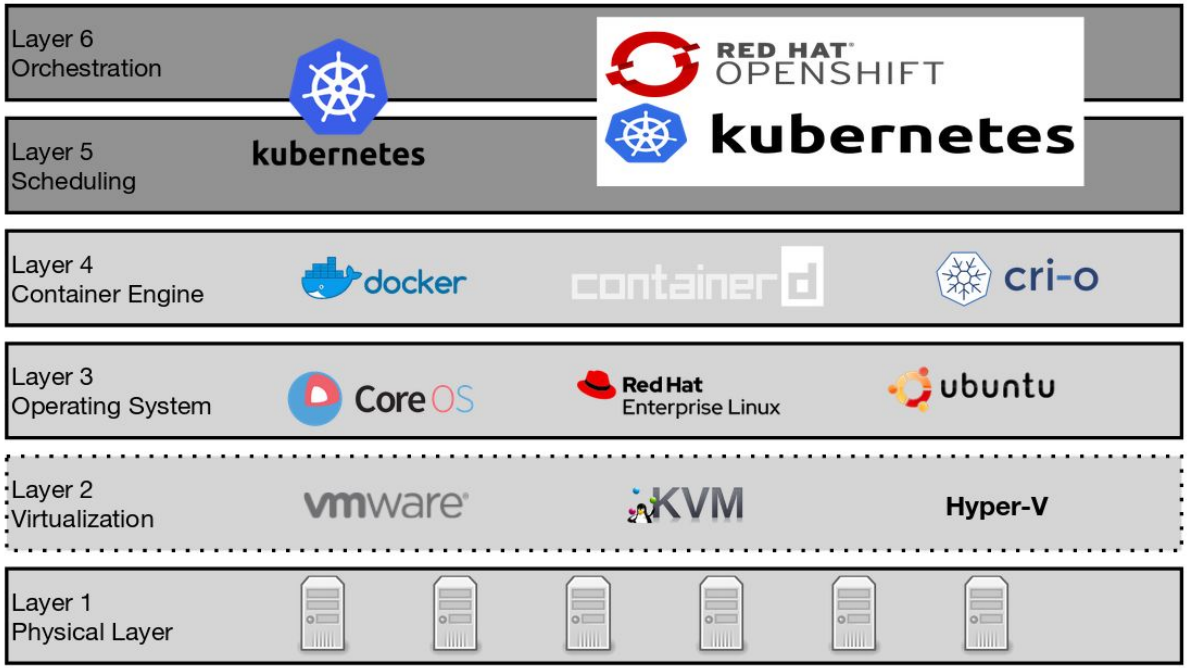
Rückblick in die 2000er: Applikationserver

- War immer All-in-One-Lösung, egal ob Feature genutzt wurde oder nicht
  - Beispiele:
    - Anwendung in Spring, wird aber in einen Applikationsserver deployt
    - java.mail

Auch in Kubernetes habe ich Features, die ich aktiv nicht nutze

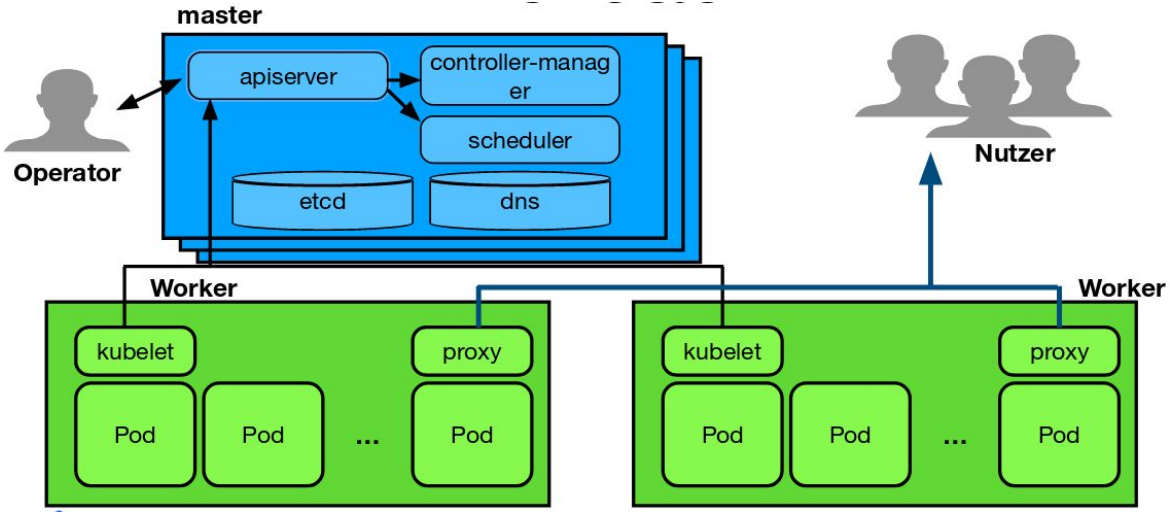
Kubernetes: Full-Stack-Lösung für das Rechenzentrum

- All-in-one-Lösung
- Orchestrierung von Containern
- Skalierung
- Lastverteilung
- Auslastung von Ressourcen



Kubernetes: Warum wir es als Monolithen sehen?

- There are features necessary for the operation of kubernetes, even if my application does not actively use them
- For example:
  - What might happen, if I
    - remove etcd?
    - shut down kubedns?



Impact of the Monolithic Structure

On the operational side of Kubernetes:
High complexity as it was on the application Servers, but this time on data center level

The Operations Department wants Service Discovery to avoid manual static routing, but wants the traditional deployment procedures

-> Not useful with Kubernetes

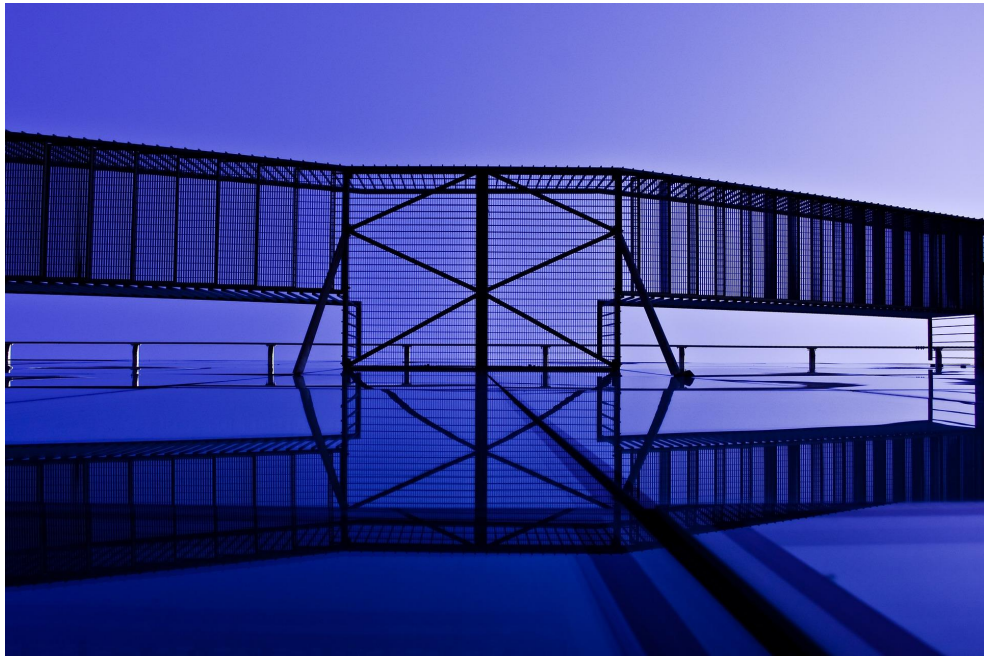
The Operations Department wants automatic scaling, but forces hardware load-balancers to be used.

-> Not useful with Kubernetes

Kubernetes seen from a classic operational perspective

What is classical operations?

- Strict seperation of concernt: There are people who care on
  - Firewall
  - Network
  - Load Balancing
  - Server
  - Applikationen
  - OS




When the classic
Operations Department
first meets Kubernetes

When the Classic Operations runs Kubernetes

- Networking is completly different
  - Firewall, Load Balancer, DNS, Namespaces (Soft Networkzones, Software controlled)
- “We need a DMZ”
  - Different Clusters are installed, with a hardware firewall between the clusters

In our opinion this does not fit the idea of Kubernetes

We think of Kubernetes as a management system for a data-center

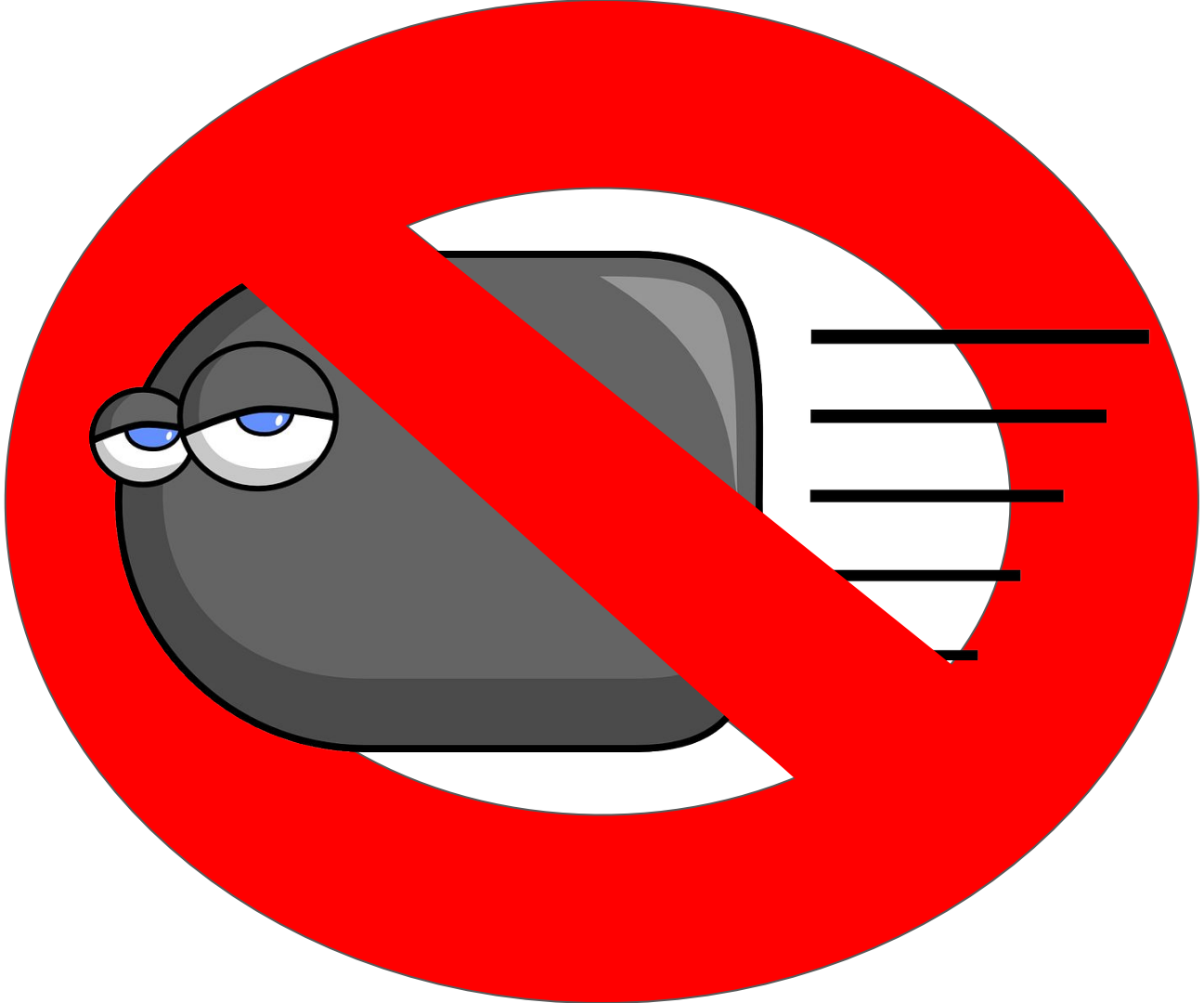
Does this problem exist only on self managed kubernetes?

Not at all, you can misconfigure Kubernetes on a managed platform such as GKE too, as this is just an virtual data center.

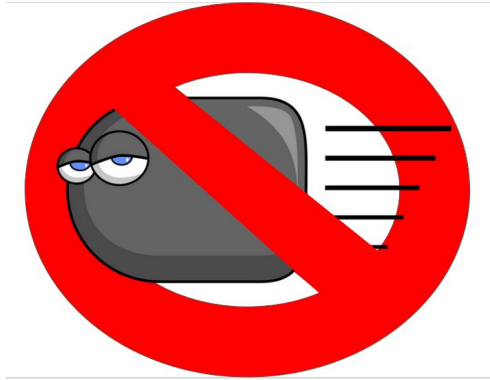
Try an explanation why they do so

They simply try to adopt known operational patterns to kubernetes

So far, but now for an Alternative



Our Proposal



Slow transformation from a manually managed operations to automatically managed operations

Example for a manually managed operation

- The deployment target is fixed before the deployment starts
- Configuration management is done by push principle

Examples for an automatically managed operation

- Deployment target is chosen while deployment takes place.
- Configuration management is done by pull principle

Motivation for a Slow Transformation

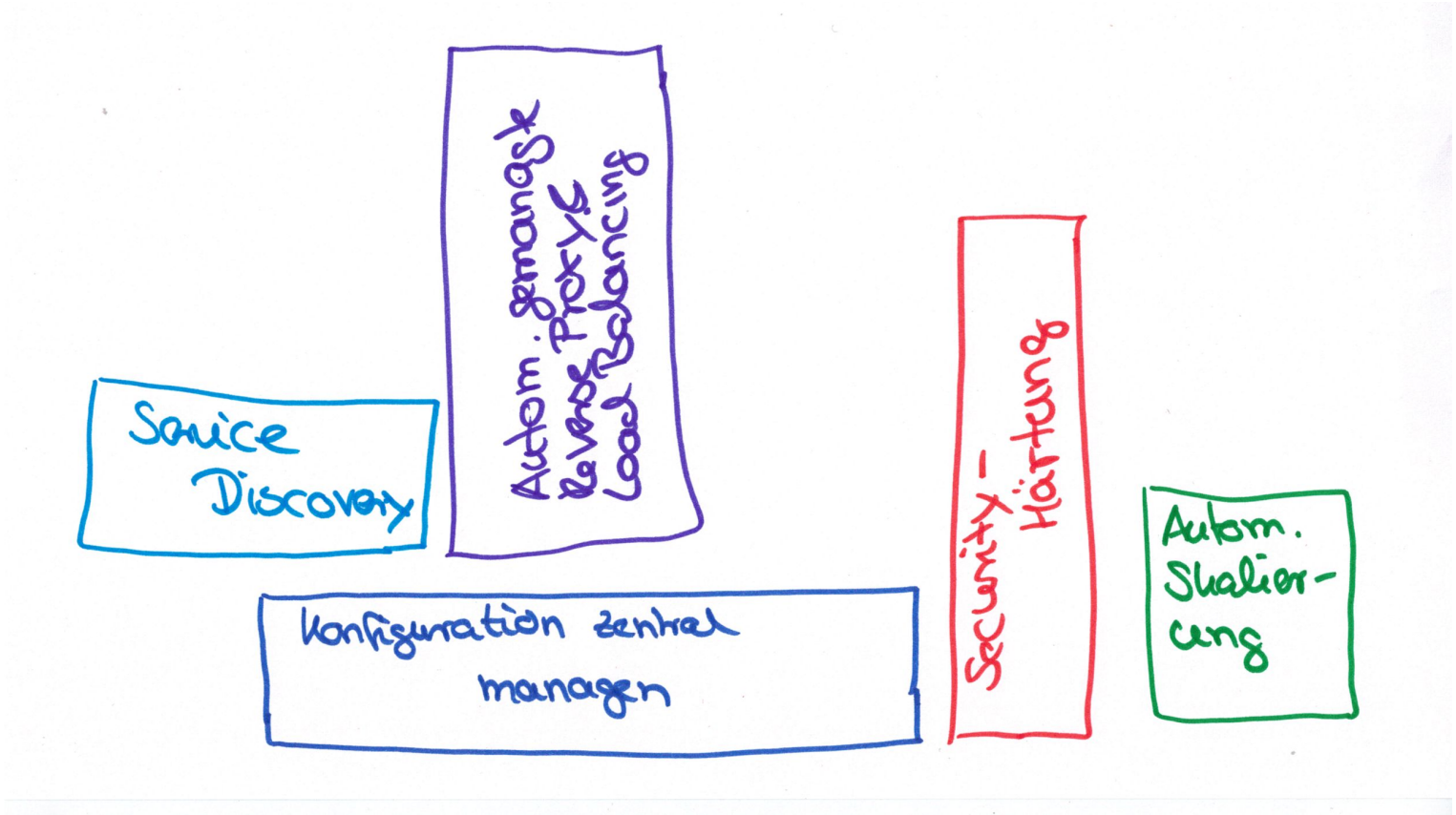
- Big Bang Migration was succesfull only once
- It is unclear if we need the whole transformation
- Keep an eye on the Return On Invest
- Applies for greenfield projects as well
  - Initial learning curve is less steep
  - Infrastructure grows with the requirements

Main Idea

Unix Philosophy

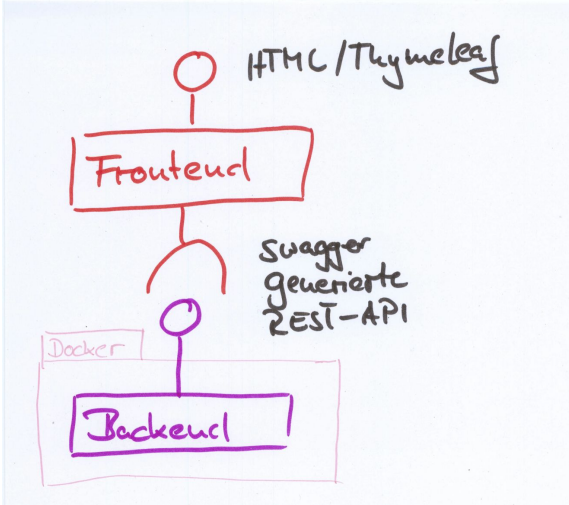
Take the building blocks you need.

Idee für Transformationsschritte



Begin of our journey

- “Ultimative Comic-Hero Applikation”
- Spring Boot application
- Frontend with Thymeleaf and REST-Client
- Backend with RAM based storage
- REST-Services based on swagger
- Configuration:
  - Spring-alike “application.properties”
  - Command line “-D…=...”

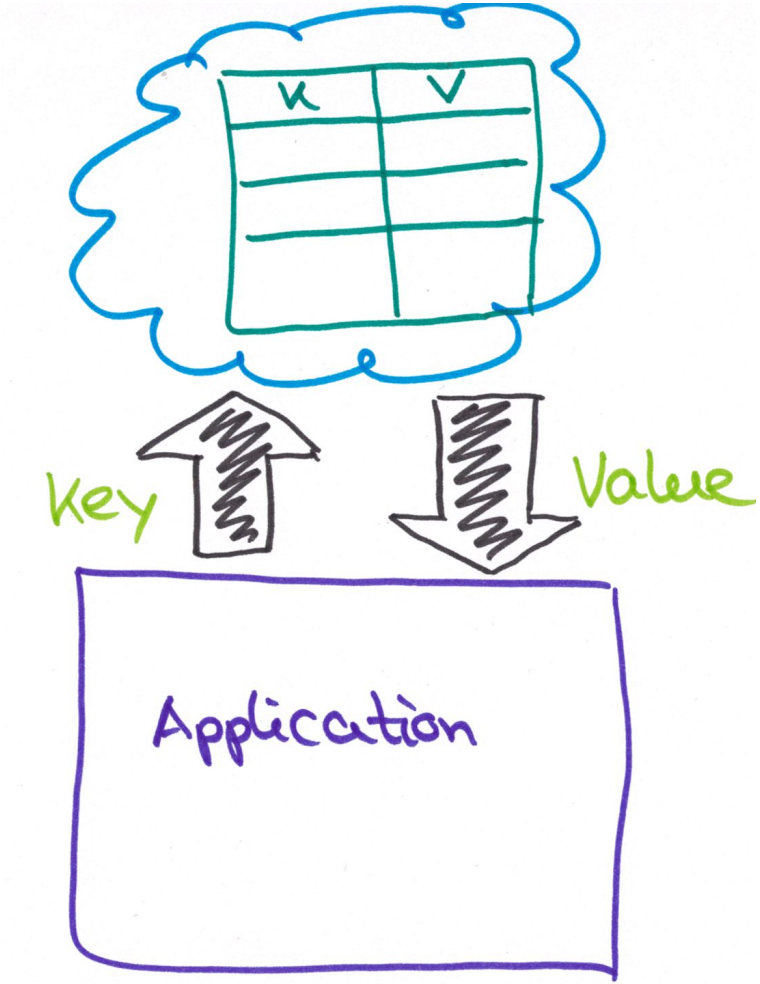


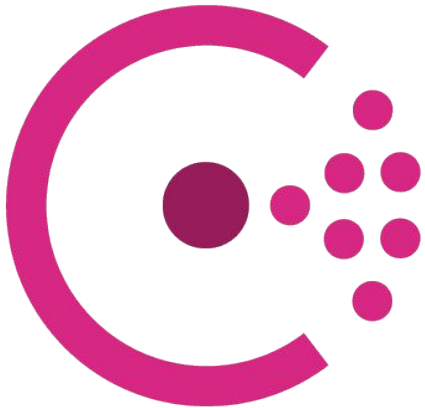
Building block:
Centralize Configuration Management

Centralize Configuration Management

- Configuration such as:
  - Feature-Switch
  - Flags
  - URLs
  - Credentials
- Gives us the basis to the next steps

Key-Value Storage





- K/V Storage and Service Registry by HashiCorp
- Excerpt of the Features:
  - Configuration Management
  - Service Discovery
  - Service Mesh
  - Service Health Check

Effects on Software-Architecture

- Instead of using the (shell) environment, a central source is used.
- All configurations are in one source, independent of the deployment.
- Changed to the Software are needed

```
    public HeroServiceRestClient(@Value("${backend.url}") String backendUrl,
                                 @Value("${backend.username}") String username,
                                 @Value("${backend.password}") String password) {
        heroApi = new HeroApi();
        final ApiClient apiClient = heroApi.getApiClient();
        apiClient.setBasePath(backendUrl);
        apiClient.setPassword(password);
        apiClient.setUsername(username);
        repositoryApi = new RepositoryApi(apiClient);
    }
```

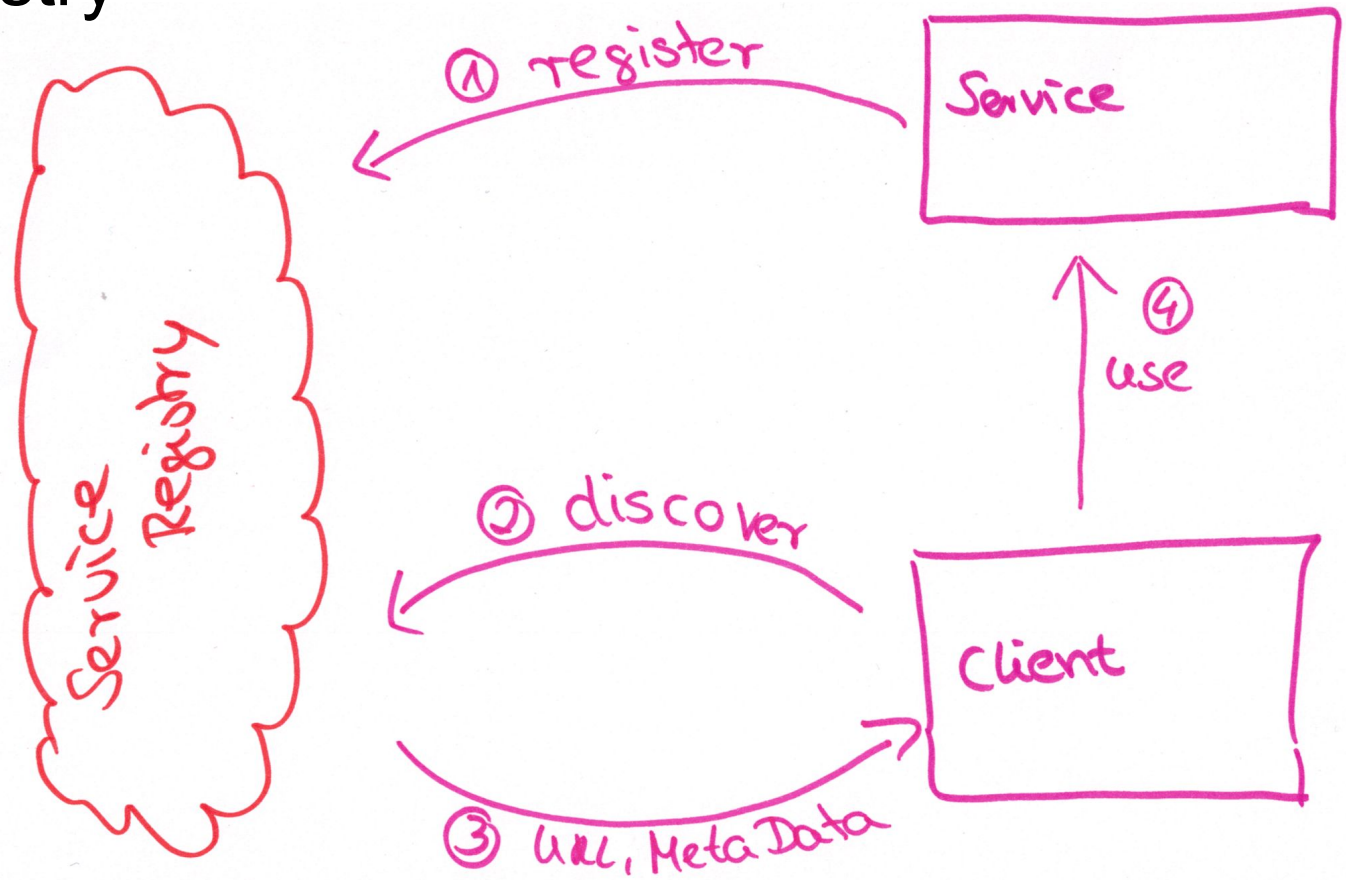
```java
public HeroServiceRestClient(ConfigurableEnvironment configurableEnvironment) {
    String backendUrl =  configurableEnvironment.getRequiredProperty("backend.url");
    String username = configurableEnvironment.getProperty("backend.username");
    String password = configurableEnvironment.getProperty("backend.password");
    heroApi = new HeroApi();
    final ApiClient apiClient = heroApi.getApiClient();
    apiClient.setBasePath(backendUrl);
    if (username != null ) {
        apiClient.setPassword(password);
        apiClient.setUsername(username);
    }
    repositoryApi = new RepositoryApi(apiClient);
}
```

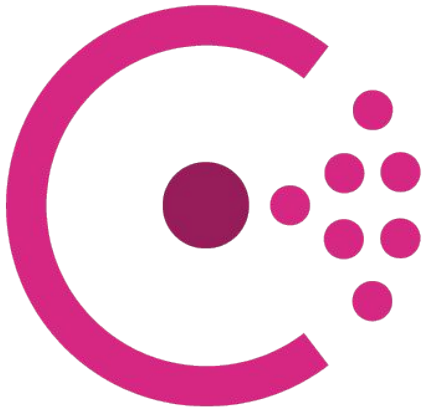
Building Block:
Service Discovery

Definition of a Service

- Database
- Classic application-server
- FatJar
- Docker Container
- Shell-Script :)
- Exec (Some Binary)
- Each of them may provide different services

Service Registry

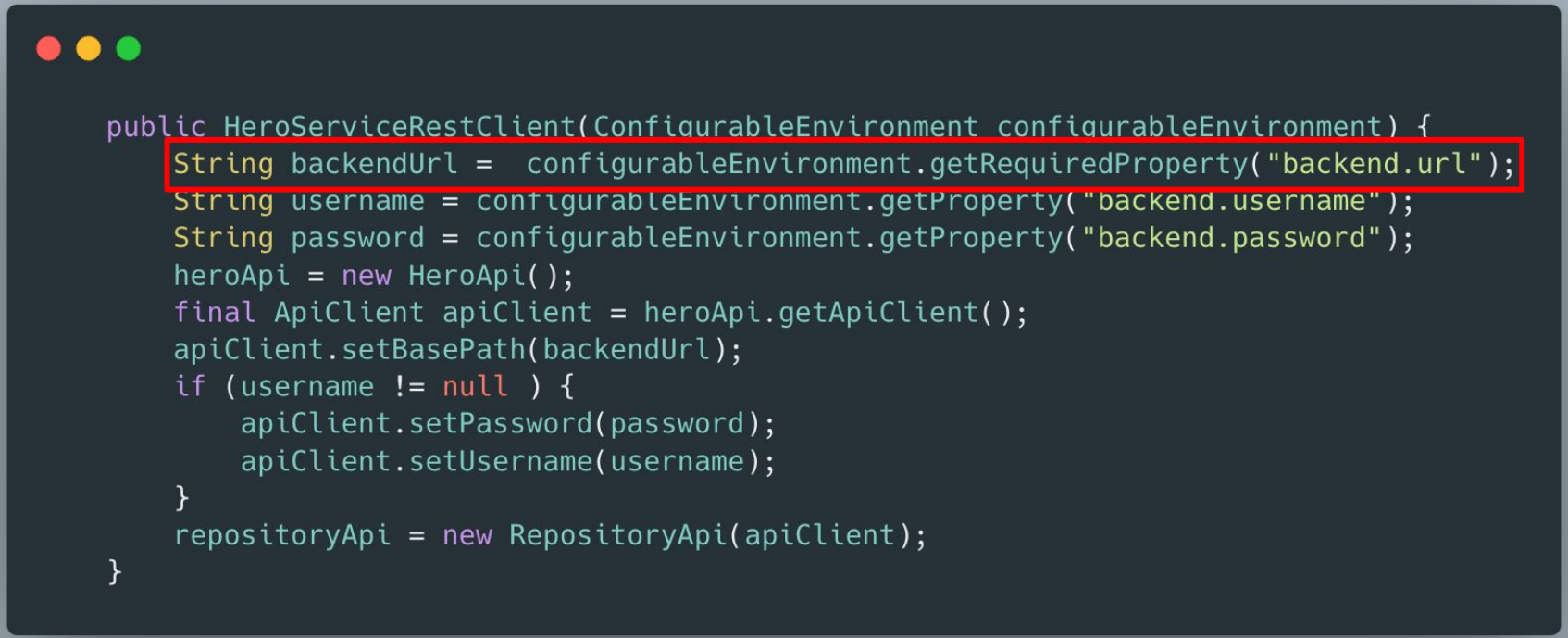




- K/V Storage and Service Registry by HashiCorp
- Excerpt of the Features:
  - Configuration Management
  - Service Discovery
  - Service Mesh
  - Service Health Check

Effects on Software-Architecture

- Service-URLs are no longer a part of manual configuration.
- Services must register at the Service Registry.
- Clients geht the URL from the Service Registry.


public HeroServiceRestClient(ConfigurableEnvironment configurableEnvironment) {
String backendUrl = configurableEnvironment.getRequiredProperty("backend.url");
String username = configurableEnvironment.getProperty("backend.username");
String password = configurableEnvironment.getProperty("backend.password");
heroApi = new HeroApi();
final ApiClient apiClient = heroApi.getApiClient();
apiClient.setBasePath(backendUrl);
if (username != null) {
apiClient.setPassword(password);
apiClient.setUsername(username);
}
repositoryApi = new RepositoryApi(apiClient);
}

```
public HeroServiceRestClient(DiscoveryClient discoveryClient,
                             ConfigurableEnvironment configurableEnvironment) {
    this.discoveryClient = discoveryClient;
    String username = configurableEnvironment.getProperty("backend.username");
    String password = configurableEnvironment.getProperty("backend.password");
    heroApi = new HeroApi();
    final ApiClient apiClient = heroApi.getApiClient();
    apiClient.setBasePath(retrieveURL());

    if (username != null ) {
        apiClient.setPassword(password);
        apiClient.setUsername(username);
    }
    repositoryApi = new RepositoryApi(apiClient);
}
```

```java
    private String retrieveURL() {
      Optional<ServiceInstance> serviceOptional =
discoveryClient.getInstances(backendInstanceName).stream().findAny();
        if (serviceOptional.isPresent()) {
            ServiceInstance serviceInstance = serviceOptional.get();
            LOGGER.info(serviceInstance.getUri().toString());
            URI uri = serviceInstance.getUri();
            String baseUrl = "/api";
            String fullURL = uri.toString() + baseUrl;
            LOGGER.info("ServicePath: {}", fullURL);
            return fullURL;
        }
        return backendUrl;
    }
```

Building Block:
Automatically Managed Reverse Proxy and Load Balancing

Automatically Managed Reverse Proxy and Load Balancing

- Classically an admin configures the reverse proxy and the load balancer.
- It is dependent on the Service Discovery
- Reverse Proxies and Load Balancer configure themselves by the information they read from the Service Discovery

- Reverse Proxy and Load Balancer for HTTP(S) and TCP-based Application
- Excerpt from the Features:
  - Autoconfigurable
  - Cluster-ready

PROVIDERS HEALTH V1.7.18 / MAROILLES DOCUMENTATION

5 FRONTENDS

frontend-backend-dockerbe-backend

Main | Details

Route Rule

```
Host:backend-dockerbe-backend.consul.localhost
```

Entry Points http

Backend backend-backend-dockerbe-backend

5 BACKENDS

backend-backend-dockerbe-backend

Main | Details

Server	Weight
http://127.0.0.1:20823	1
http://127.0.0.1:26357	1
http://127.0.0.1:29497	1

PROVIDERS HEALTH V1.7.18 / MAROILLES DOCUMENTATION

Backend backend-consul

frontend-frontend-webs-frontendnomad

Main | Details

Route Rule

```
Host:frontend-webs-frontendnomad.consul.localhost
```

Entry Points http

Backend backend-frontend-webs-frontendnomad

backend-frontend-webs-frontendnomad

Main | Details

Server	Weight
http://127.0.0.1:29680	1

backend-nomad

Main | Details

```
# Enable Consul Catalog Provider.
[consulCatalog]

# Consul server endpoint.
endpoint = "127.0.0.1:8500"

# Expose Consul catalog services by default in Traefik.
exposedByDefault = true

# Default base domain used for the frontend rules.
domain = "consul.localhost"

# Keep a Consul node only if all checks status are passing
# If true, only the Consul nodes with checks status 'passing' will be kept.
# if false, only the Consul nodes with checks status 'passing' or 'warning' will be kept.
strictChecks = true
```

Effects on the Software Architecture

None 😉

Building Block: Automatic Scaling

Automatic Scaling

- Services must be propagated based on rules
  - Monitoring
  - Configuration
- Resilience

HashiCorp

Nomad

- Service Orchestrator by HashiCorp
- Excerpt from Features:
  - Clustering
  - multiple Deployment-Formats
    - Fat-Jar
    - Docker
    - any executable

```
job "frontend" {
  region = "global"
  datacenters = ["dc1"]
  type = "service"

  update {
    stagger      = "30s"
    max_parallel = 1
  }

  # A group defines a series of tasks that should be co-located
  # on the same client (host). All tasks within a group will be
  # placed on the same host.
  group "webs" {
    # Specify the number of these tasks we want.
    count = 1

    # Create an individual task (unit of work). This particular
    # task utilizes a Docker container to front a web application.
    task "frontend" {
      driver = "java"

      # Configuration is specific to each driver.
      config {
        jar_path    = "local/frontend-1.0.0.jar"
        args        = ["--server.port=${NOMAD_PORT_http}", "--server.address=${NOMAD_IP_http}"]
      }

      artifact {
        source      = "https://github.com/repository/maven-public/com/github/sparsick/frontend/1.0.0
/frontend-1.0.0.jar"
        options {
          checksum = "md5:b239ba8eaa890f45689a77e2145dabdf"
        }
      }
```

```
      # The service block tells Nomad how to register this service
      # with Consul for service discovery and monitoring.
      service {
        # This tells Consul to monitor the service on the port
        # labelled "http". Since Nomad allocates high dynamic port
        # numbers, we use labels to refer to them.
        port = "http"

        check {
          type     = "http"
          path     = "/actuator/health"
          interval = "30s"
          timeout  = "2s"
        }
      }

      # Specify the maximum resources required to run the task,
      # include CPU, memory, and bandwidth.
      resources {
        cpu    = 500 # MHz
        memory = 512 # MB

        network {
          mbits = 100

          # This requests a dynamic port named "http". This will
          # be something like "46283", but we refer to it via the
          # label "http".
          port "http" {
          }
        }
      }
    }
  }
}
```

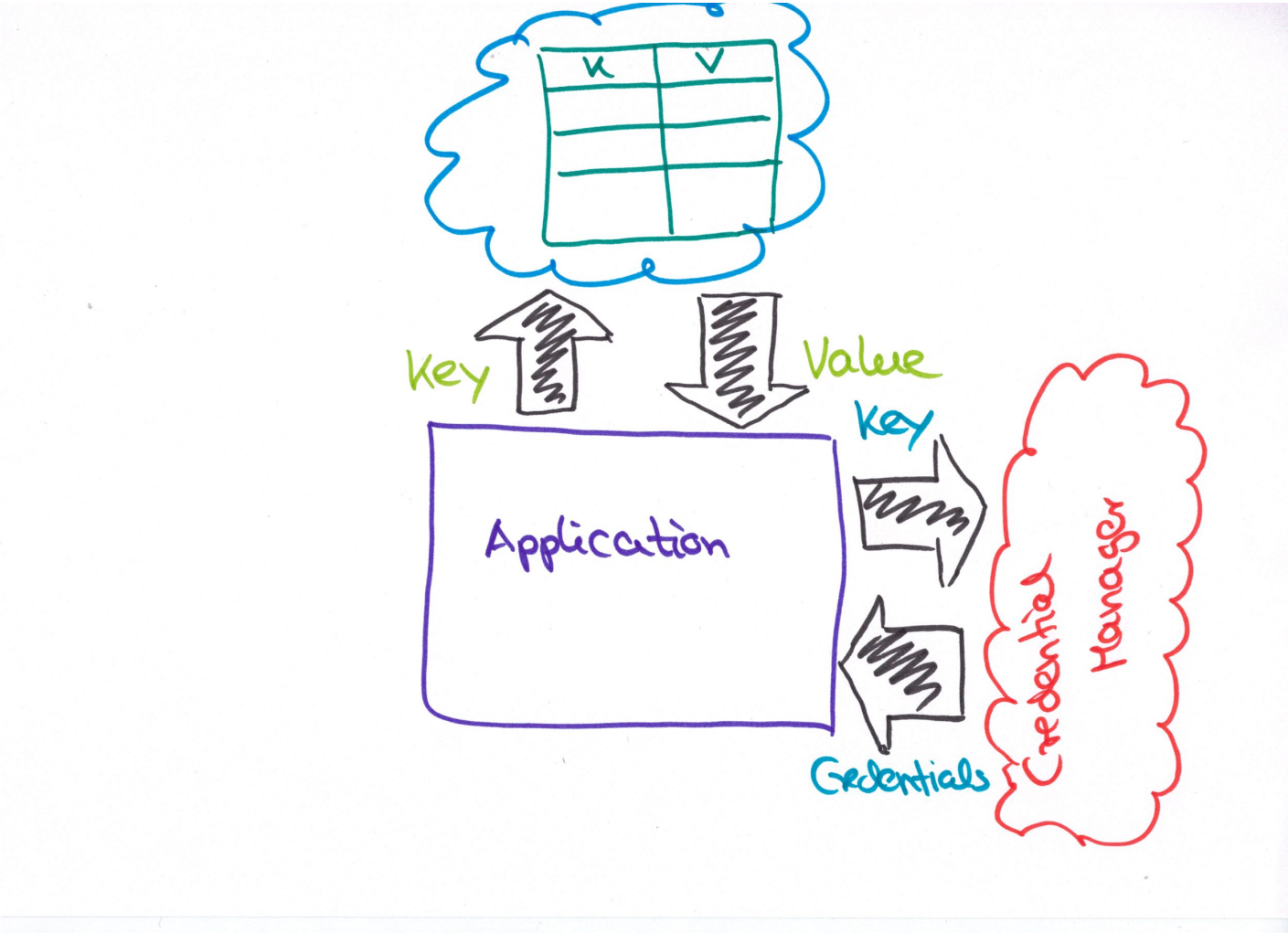
Effects on the Software Architecture

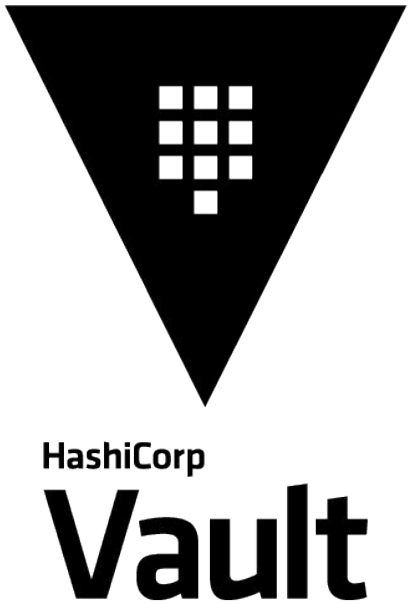
- Each service must be stateless
- Each service must be able to be started several times
- Every service must be terminated at any time
- Clients must handle missing services
- Interfaces must be idempotent

Building Block: Security Hardening

As a Reminder:
Centralize Configuration Management

- Configuration as:
  - Feature-Switch ✅
  - Flags ✅
  - URLs ✅
  - Credentials ⚠️
- Requirements to Credentials
  - Secure Credentials
  - Do not store Credentials unprotected (in clear text)

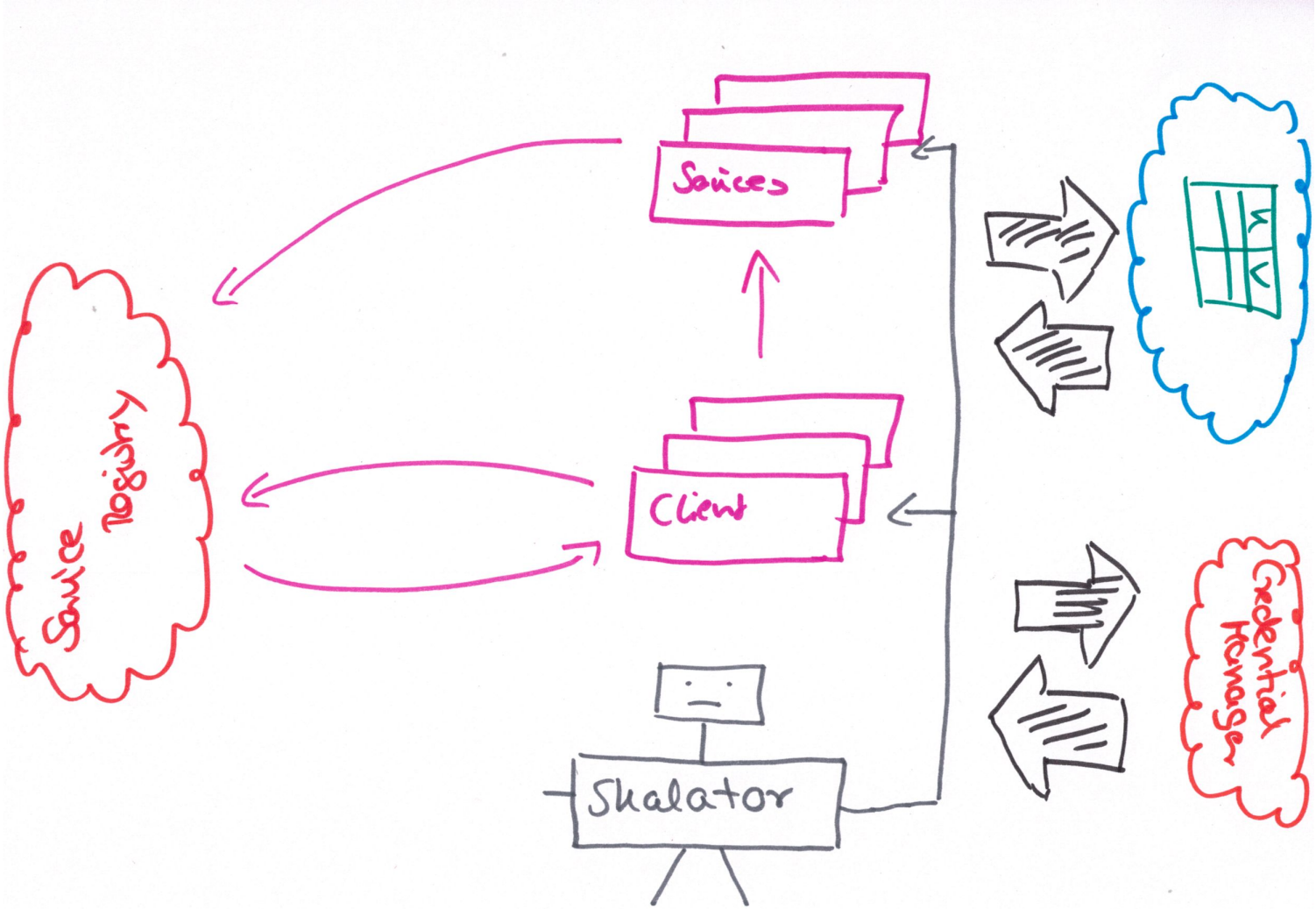

K
V
Key
Value
Application
Key
Credentials
Credential Manager



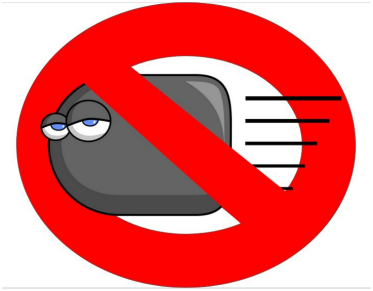
- Secret Management by HashiCorp
- Excerpt of Features:
  - tokens
  - passwords
  - certificates
  - encryption keys
  - UI, CLI, HTTP API

Effects on the Software Architecture

- Using the Credential Manager instead of the K/V Storage.


Sources
Source Registry
Client
Skalator
KV
Credential Manager

Hashicorp-Stack, die neue Silver Bullet?

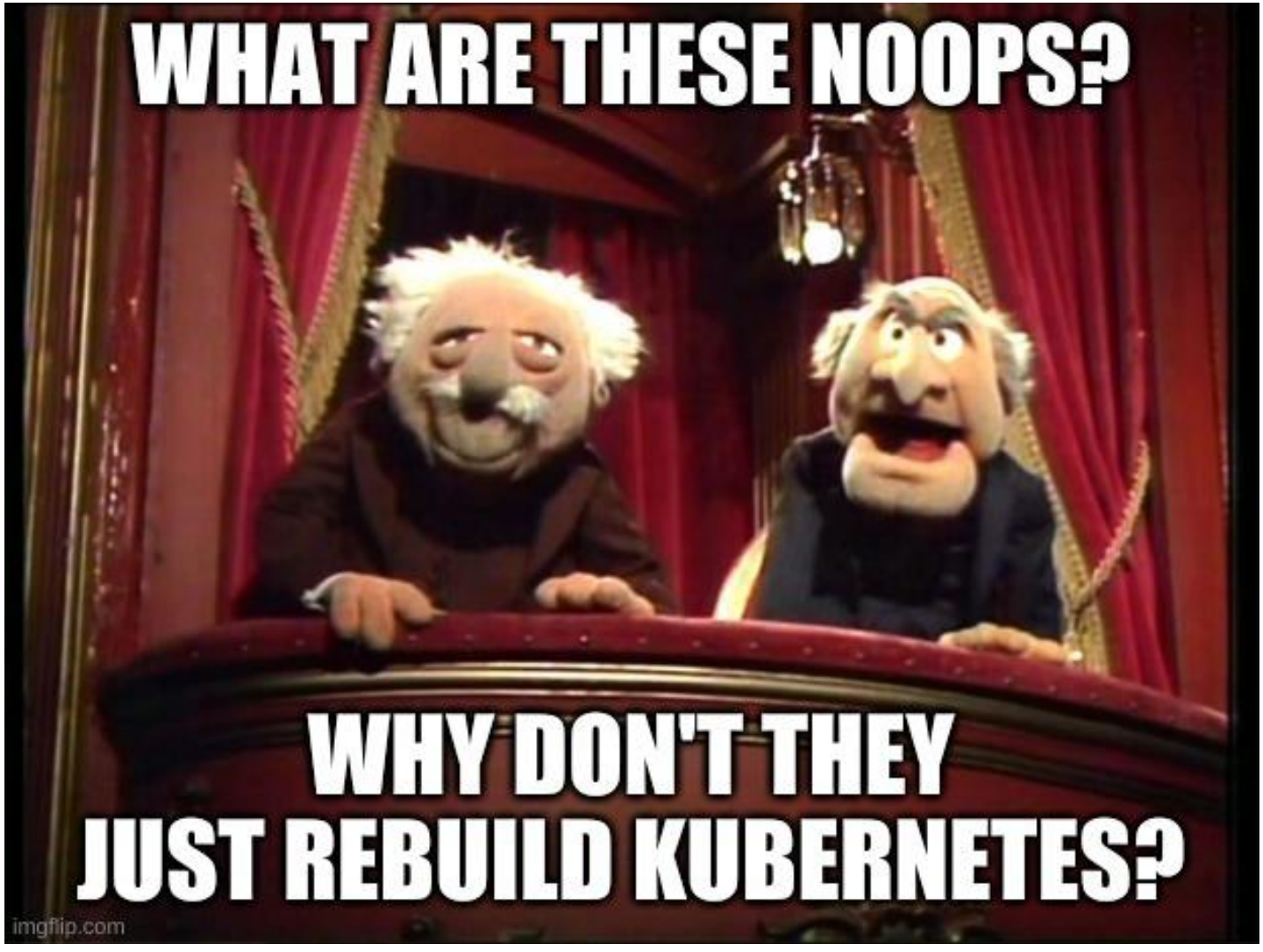


Netflix Eureka

etcd

CLOUDFOUNDRY

NGINX


WHAT ARE THESE NOOPS?
WHY DON'T THEY
JUST REBUILD KUBERNETES?
imgflip.com

When to Choose Kubernetes?

When to Choose Kubernetes

⚠️ In our opinion, kubernetes is a management system for a data-center. ⚠️

- Whenever I know I need the complete feature-set
- When one has to use his data-center up to the technical barrier
  - For economic reasons
  - Reseource hungry application, that must be automatically scaled
- In the end, the Controlling Department decides on economic reasons


We, as techincians, only provide data as a resource of decisions, which techological solution will be economic useful.

Conclusion

Unix Philosophy

Use the pieces you need

Don’t use technology because of itself.

Economics and function must be the focus.

Questions?

https://github.com/sanddorn/InfrastructureAsMicroservice


mail@sandra-parsick.de


@SandraParsick


xing.to/sparsick


info@bermuda.de


@sanddorn


xing.to/sanddorn

Further Information (Partially German)

- https://blog.risingstack.com/the-history-of-kubernetes/
- Kubernetes in Action von Marko Lukša; 1.Auflage 2017 Manning
- Skalierbare Container-Infrastrukturen - Das Handbuch für Administratoren von Oliver Liebel; 2., aktualisierte und erweiterte Auflage 2018 Rheinwerk Computing

Bildnachweis

- Icons made by Dave Gandy from www.flaticon.com
- Image by Marc Vanduffel from Pixabay
- Image by OpenClipart-Vectors from Pixabay
- Image by Francis Ray from Pixabay