

Infrastructure As Microservices

Alternativen zum Monolithen Kubernetes

Continuous Lifecycle, 14.11.2019

Über uns

Nils Bokermann

- freiberuflicher Softwareentwickler
- Chemiker :-)

Schwerpunkte

- Entwicklung von Java Enterprise Anwendung
- Ende-zu-Ende Verantwortlichkeit



info@bermuda.de



@sanddorn



xing.to/sanddorn

Über uns

Sandra Parsick

- freiberufliche Softwareentwicklerin
- Softwerkskammer Ruhrgebiet, Oracle Groundbreaker Ambassador

Schwerpunkte

- Entwicklung von Java Enterprise Anwendung
- Automatisierung von Entwicklungsprozessen



mail@sandra-parsick.de

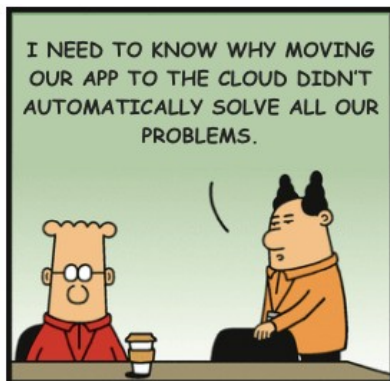


[@SandraParsick](https://twitter.com/SandraParsick)



xing.to/sparsick

Motivation



@ScottAdamsSays
Dilbert.com



11-08-17 © 2017 Scott Adams, Inc. Dist. by Andrews McMeel

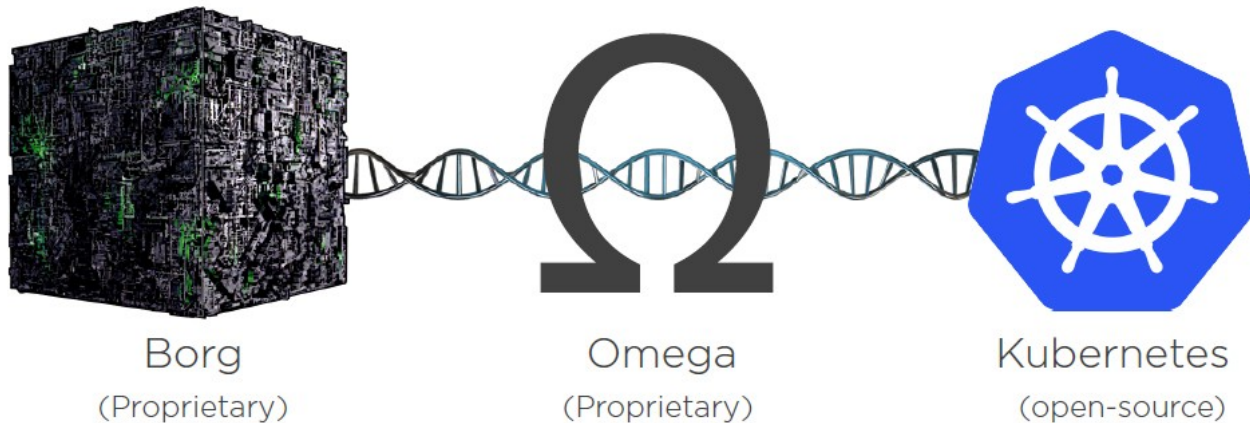


Professionelle Softwareentwicklung

Professionelle Softwareentwicklung definieren wir u.a. dadurch, dass die Systeme, die entwickelt werden, sich auch wirtschaftlich für das Unternehmen lohnen.

Historie Kubernetes

Entstehungsgeschichte Kubernetes



Warum halten wir es für einen
Monolithen?

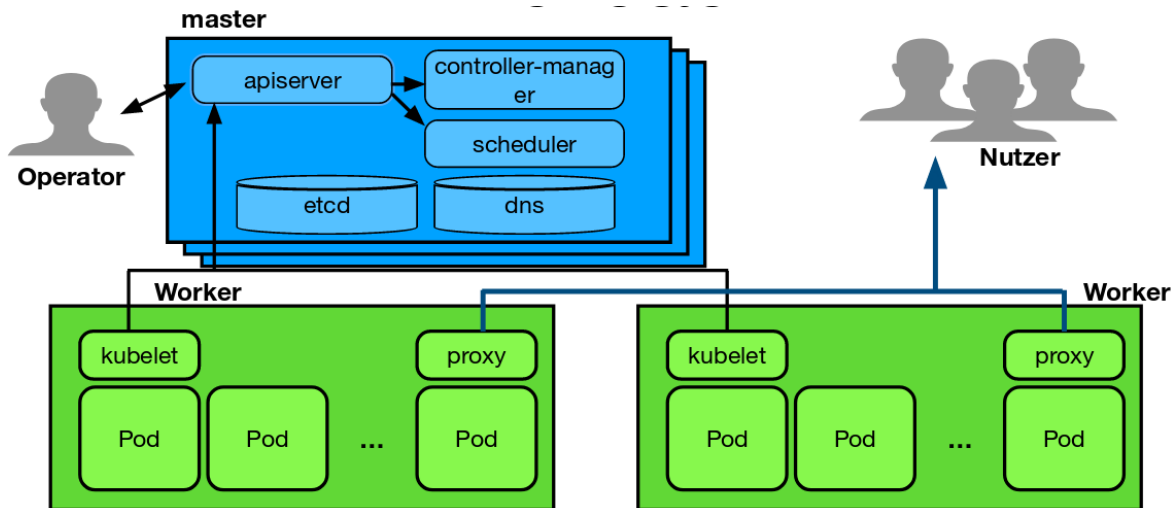
Unsere Definition eines Monolithen



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

Kubernetes: Warum wir es als Monolithen sehen?

- Es gibt Features, die notwendig sind für den Betrieb von Kubernetes, auch, wenn ich sie nicht aktiv nutze.
- Beispiele:
 - Was passiert, wenn ich
 - etcd wegnahme?
 - kubedns wegnahme?



Auswirkungen dieser monolithischen Struktur

Betrieb von Kubernetes: Hohe Komplexität wie beim Applikationsserver nur auf Rechenzentrumsebene / Betriebsebene

Betrieb will nur Service Discovery um manuelles, statisches Routing zu vermeiden, will aber nicht auf klassisches Deployment-Verfahren verzichten.

-> Geht nicht sinnvoll mit Kubernetes

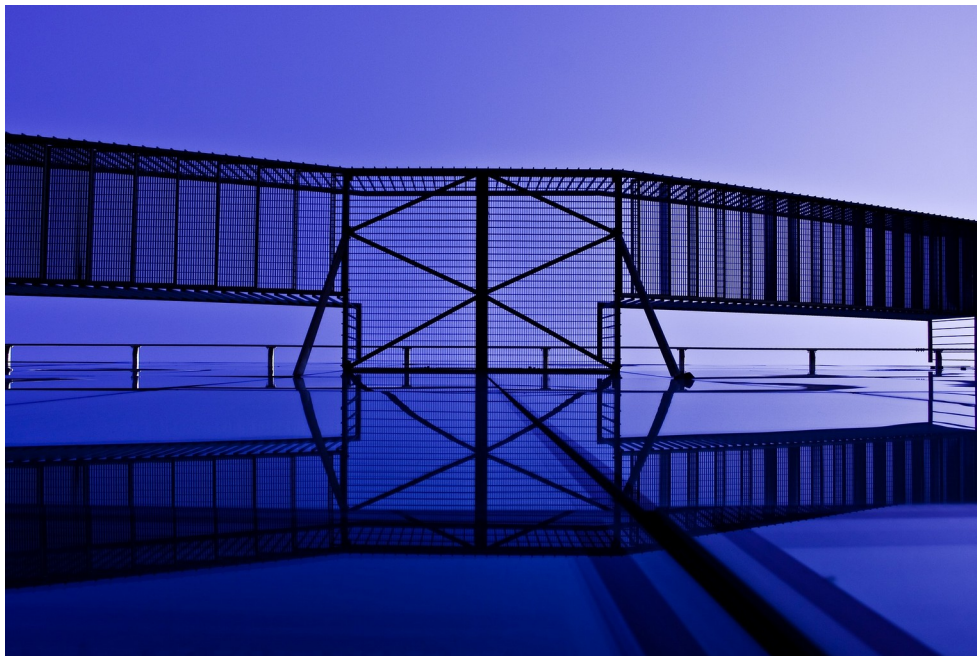
Betrieb will automatische Skalierung, aber will auf seine Hardware-Load Balancer nicht verzichten.

-> Geht nicht sinnvoll mit Kubernetes

Kubernetes aus Sicht eines klassischen Betriebs

Wie sieht klassischer Betrieb aus?

- Klare Domaintrennung: Es gibt Leute, die sich um
 - Firewall
 - Netzwerk
 - Load Balancing
 - Server
 - Applikationen
 - OS
 - kümmern



A still from the movie 'The Thin Red Line' showing two men shaking hands in a forest at night. The man on the left is older, with a weathered face, wearing a dark headband with a circular pattern and a light-colored, fur-trimmed jacket. The man on the right is younger, with dark hair, wearing a dark, patterned jacket. They are both looking at each other with expressions of mutual respect or agreement. The background is a dark, misty forest with tall trees.

Wenn klassischer
Betrieb mit Kubernetes
in Berührung kommt :)

Wenn klassischer Betrieb Kubernetes installiert

- Netzwerk funktioniert auf einmal ganz anders
 - Firewall, Load Balancer, DNS, Namespaces (weiche Netzwerkzonen)
- “Wir brauchen doch eine DMZ”
 - Mehrere Cluster werden aufgebaut und zwischen ihnen wird Hardware-Firewall geschaltet

Aus unserer Sicht geht es an der Idee von Kubernetes vorbei

Für uns ist Kubernetes ein Rechenzentrum-Verwaltungssystem

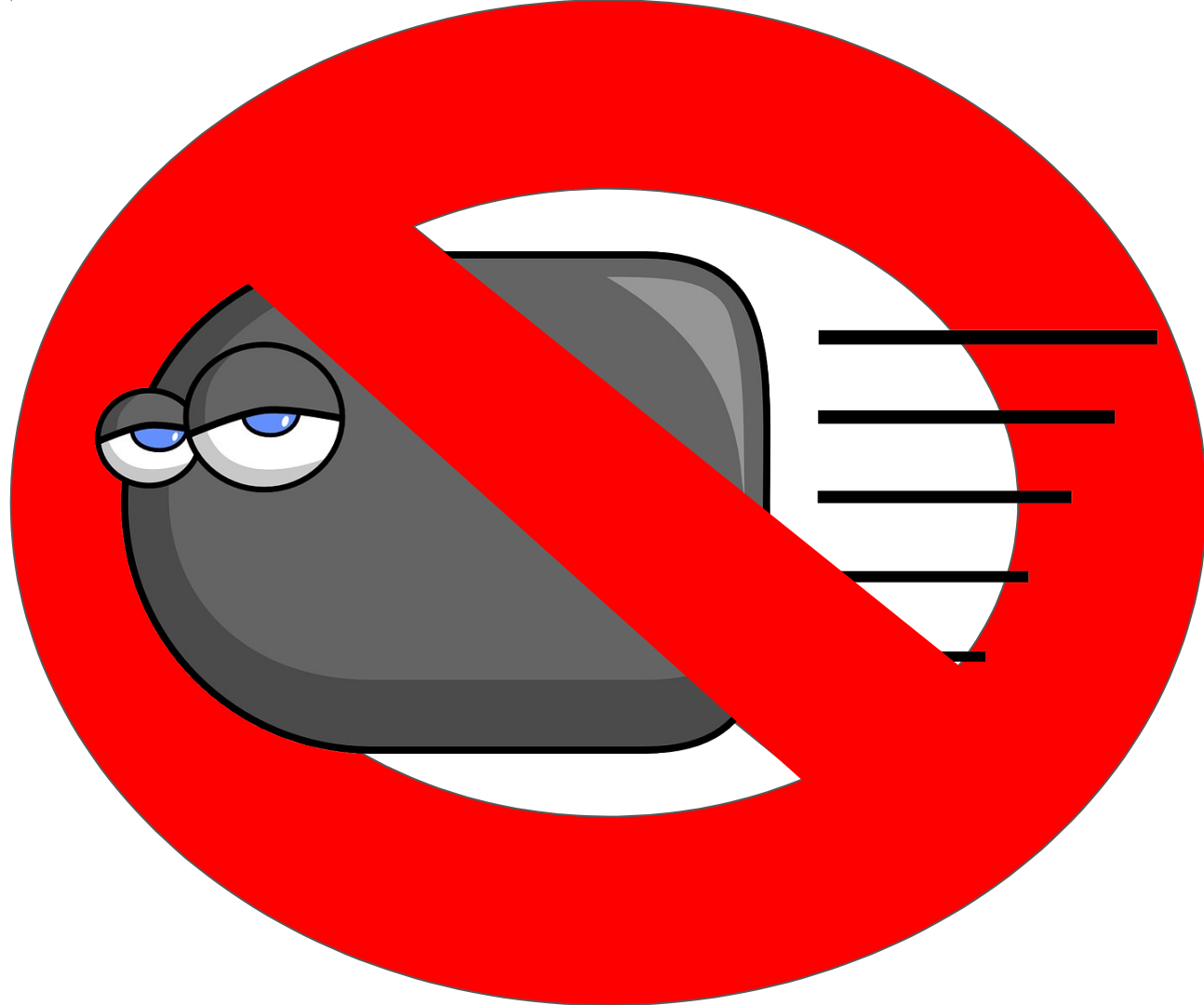
Ist das ein Problem nur von selbst gemanagten Kubernetes?

Nein, man kann es sich auch in der Google Cloud Kubernetes falsch konfigurieren
(virtuelles Rechenzentrum)

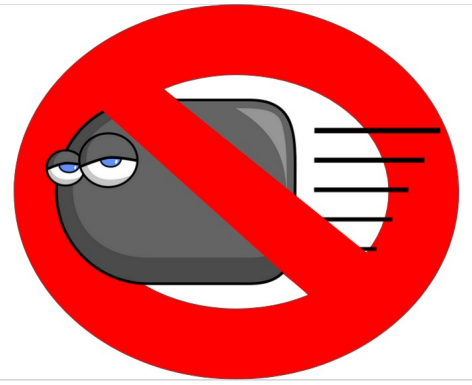
Erklärungsversuch, warum das so ist

Es wird versucht bestehende Betriebskonzepte auf Kubernetes abzubilden

Ja und wie kann eine Alternative
aussehen?



Unser Vorschlag



Schleichende Transformation von einem manuell gemanagten Betrieb zu
automatisch gemanagten Betrieb

Beispiel für manuell gemanagter Betrieb

- Das Deployment-Ziel steht vor dem Deployment fest.
- Konfigurationsmanagement nach Push-Prinzip

Beispiel für automatisch gemanagten Betrieb

- Das Deployment-Ziel ist vor dem Deployment beliebig, aber definiert nach dem Deployment.
- Konfigurationsmanagement nach Pull-Prinzip

Motivation für eine schleichende Transformation

- Big Bang Migration hat noch nie funktioniert
- Ich weiss garnicht, ob ich die komplette Transformation brauche
- Return On Invest im Auge
- Gilt auch für Greenfield-Projekte
 - Da Einstiegshürde ist geringer
 - Infrastruktur wächst mit den Anforderungen

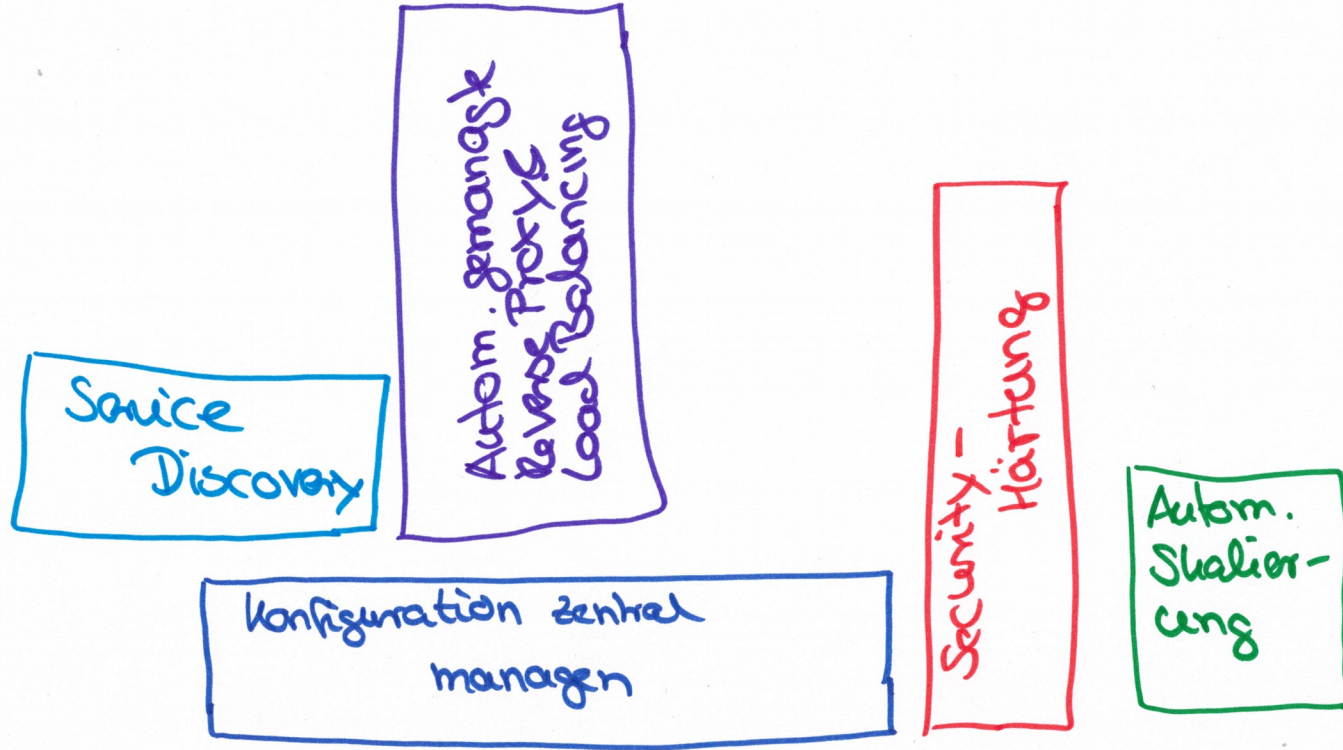
Kernidee

Unix Philosophie

Jeder nimmt sich die Bausteine, die er/sie braucht

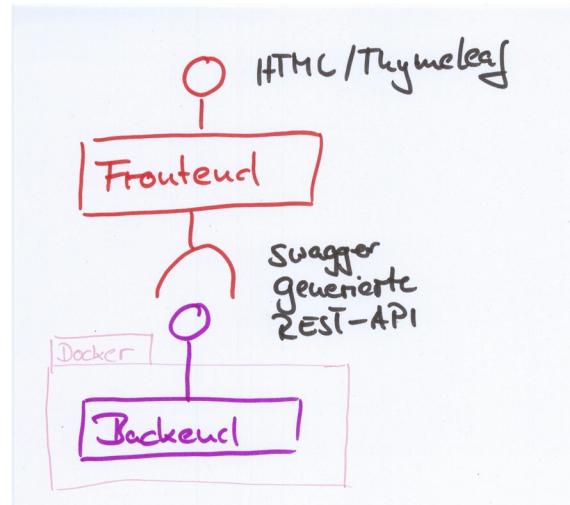


Idee für Transformationsschritte



Ausgangspunkt

- “Ultimative Comic-Helden Applikation”
- Spring Boot Anwendung
- Frontend mit Thymeleaf und REST-Client
- Backend mit Storage im RAM
- REST-Service mit Swagger generiert
- Konfiguration:
 - Spring-alike “application.properties”
 - Kommando-Zeile “-D...=...”

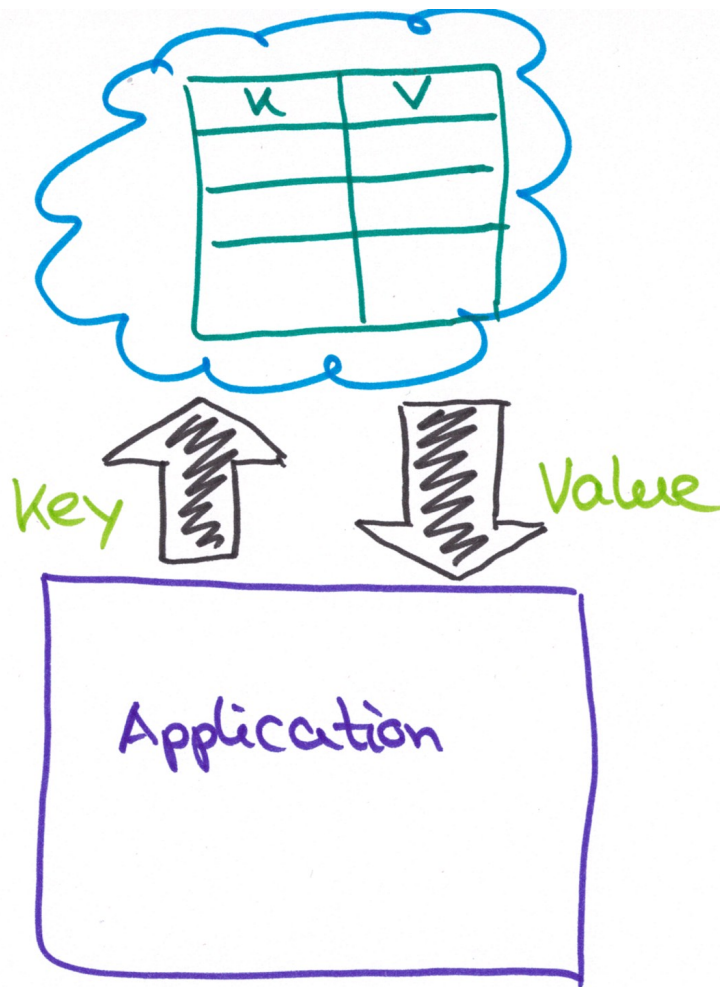


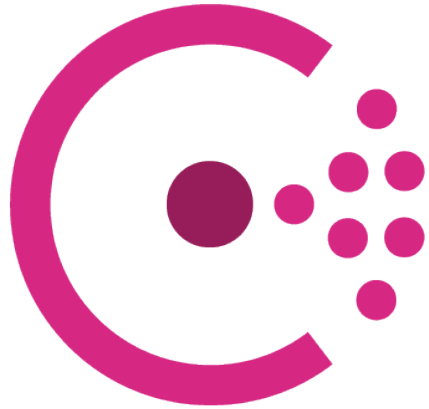
Baustein: Konfiguration zentral managen

Konfiguration zentral managen

- Konfigurationen wie:
 - Feature-Switch
 - Flags
 - URLs
 - Credentials
- Als Basis für alle weiteren Schritte notwendig

Key-Value Storage





HashiCorp

Consul

- K/V Storage und Service Registry von HashiCorp
- Auszug aus der Featureliste:
 - Configuration Management
 - Service Discovery
 - Service Mesh
 - Service Health Check

Auswirkungen auf die Software-Architektur

- Anstelle des Environments wird jetzt eine zentral gewartete Quelle genutzt
- Alle Konfiguration liegt in einer Quelle, unabhängig vom Rechner
- Änderungen an der Software sind nötig.



```
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);  
}
```



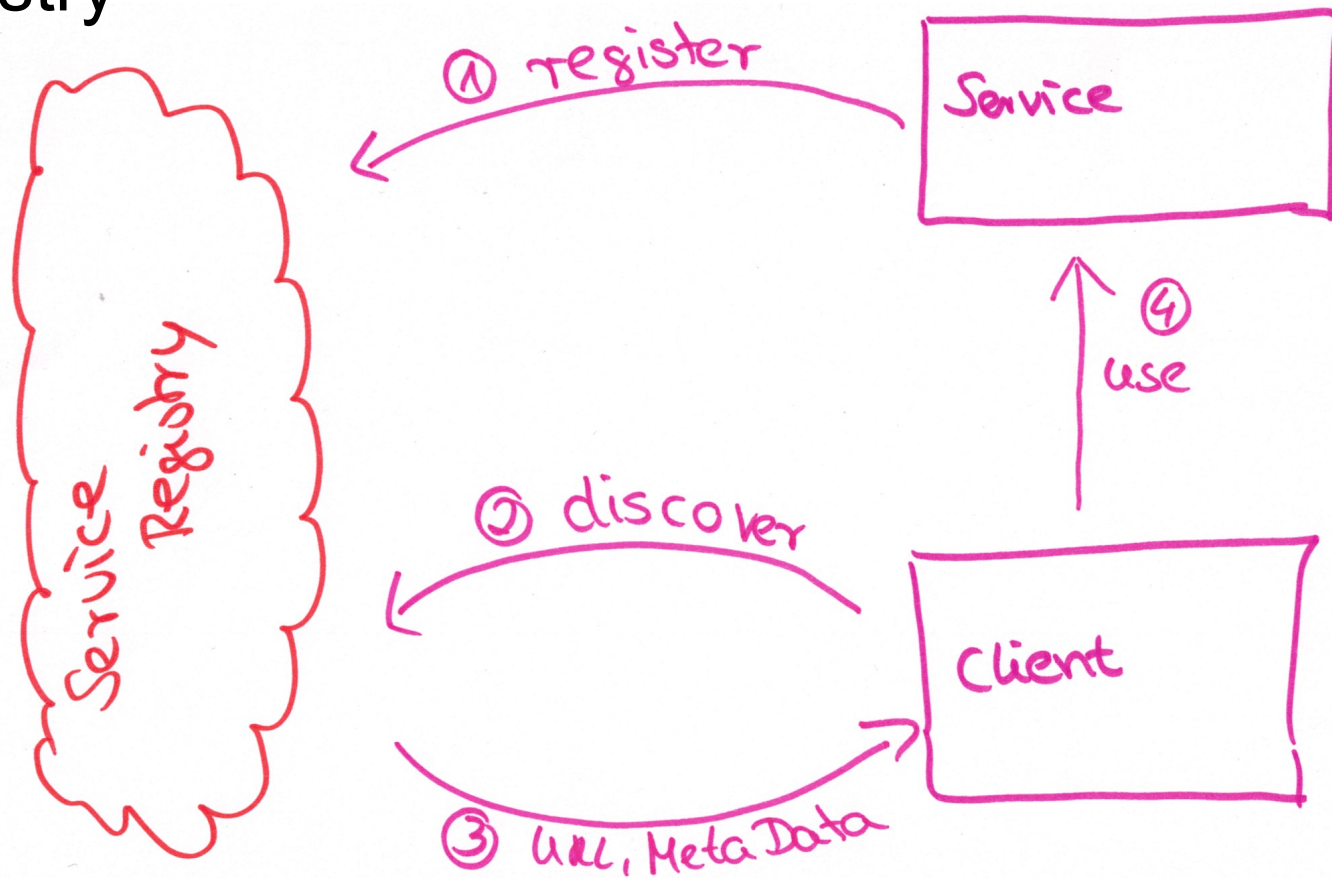
```
public HeroServiceRestClient(ConfigurableEnvironment configurableEnvironment) {  
    String backendUrl = configurableEnvironment.getRequiredProperty("backend.url");  
    String username = configurableEnvironment.getProperty("backend.username");  
    String password = configurableEnvironment.getProperty("backend.password");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

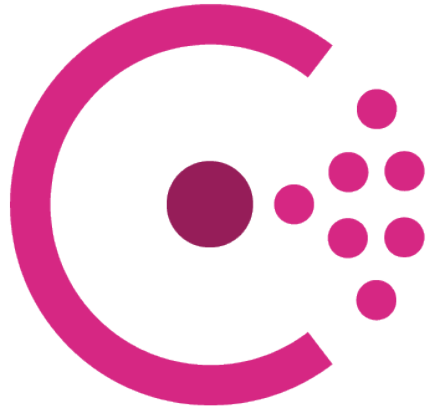
Baustein: Service Discovery

Definition Service

- Datenbank
- klassischer Applicationserver
- FatJar
- Docker Container
- Shell-Script :)
- Exec (Irgendein Binary)
- natürlich können die Obigen mehrere Services anbieten

Service Registry





HashiCorp

Consul

- K/V Storage und Service Registry von HashiCorp
- Auszug aus der Featureliste:
 - Configuration Management
 - Service Discovery
 - Service Mesh
 - Service Health Check

Auswirkungen auf die Software-Architektur

- Service-URLs werden nicht mehr als Konfiguration gespeichert.
- Services müssen sich an Service Registry anmelden.
- Clients sollen sich die URLs aus der Service Registry holen.



```
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);
}
```



```
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;  
}
```

Baustein: Automatisch gemanagtes
Reverse Proxy und Load Balancing

Automatisiertes Reverse Proxy und Load Balancing

- In der alten Welt pflegt ein Admin Reverse Proxy und Load Balancing manuell
- Ist von Service Discovery abhängig
- Reverse Proxies + Load Balancer lesen die Service Registry aus und konfiguriert sich aus dieser Information selbst



- Reverse Proxy und Load Balancer für HTTP und TCP-basierte Anwendungen
- Auszug aus der Featureliste:
 - Autokonfigurierbar
 - Cluster-ready



5 FRONTENDS



frontend-backend-dockerbe-backend

Main

Details

Route Rule

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

Entry Points

http

Backend



backend-backend-dockerbe-backend



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

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



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
```

Auswirkungen auf die Software-Architektur

Keine 😊

Baustein: Automatisierte Skalierung

Automatisierte Skalierung

- Service muss regelbasiert verteilt werden:
 - Monitoring
 - Konfiguration
- Resilience



HashiCorp

Nomad

- Service Orchestrator von HashiCorp
- Auszug aus der Feature-Liste:
 - Clustering
 - div. Deployment-Formate
 - 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" {
```

```
    }
```

```
  }
```

```
}
```

```
}
```

```
}
```

```
}
```

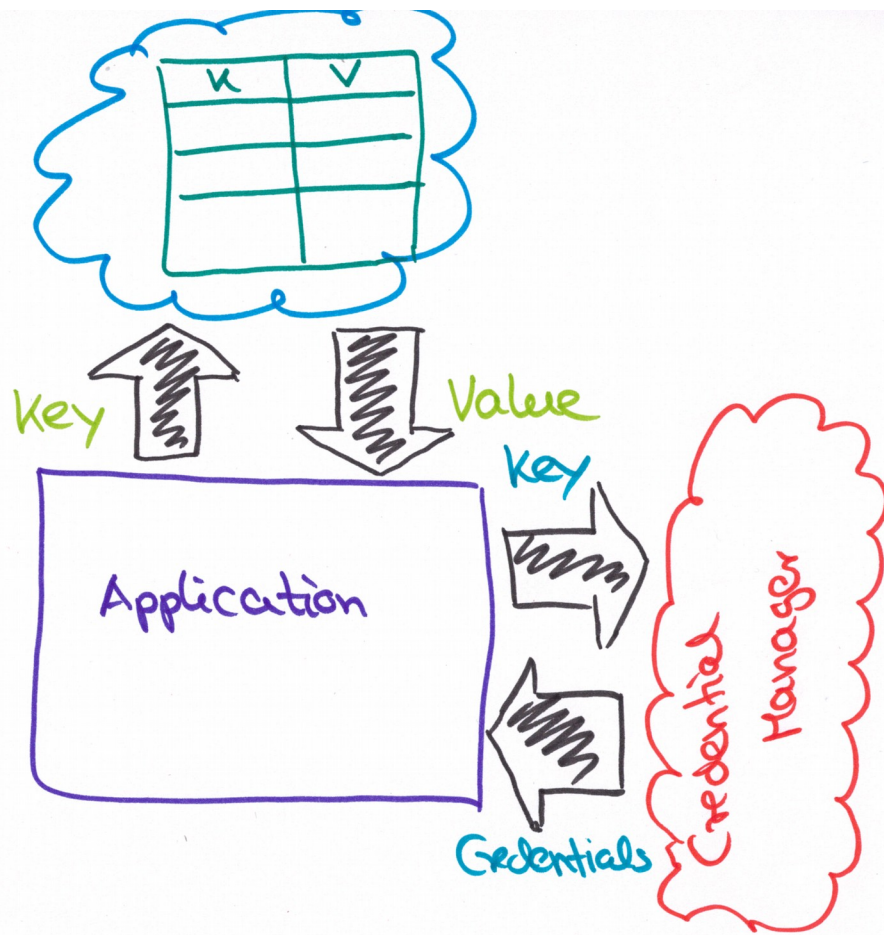
Auswirkungen auf die Software-Architektur

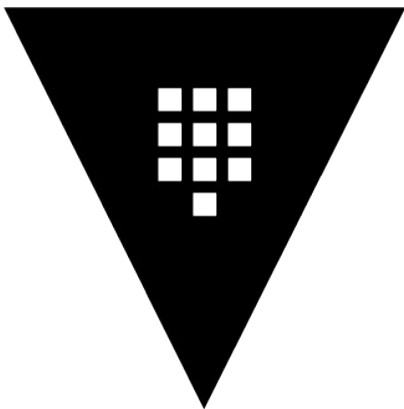
- einzelne Services müssen zustandslos sein.
- Services müssen mehrmals gestartet werden können.
- Services müssen jederzeit beendet werden können.
- Clients müssen mit ausfallenden Services klar kommen.
- Schnittstellen müssen idempotent sein.

Baustein: Security-Härtung

Zur Erinnerung: Konfiguration zentral managen

- Konfigurationen wie:
 - Feature-Switch ✓
 - Flags ✓
 - URLs ✓
 - Credentials ⚠
- Anforderung an Credentials:
 - Credentials absichern
 - Credentials nicht im Klartext speichern





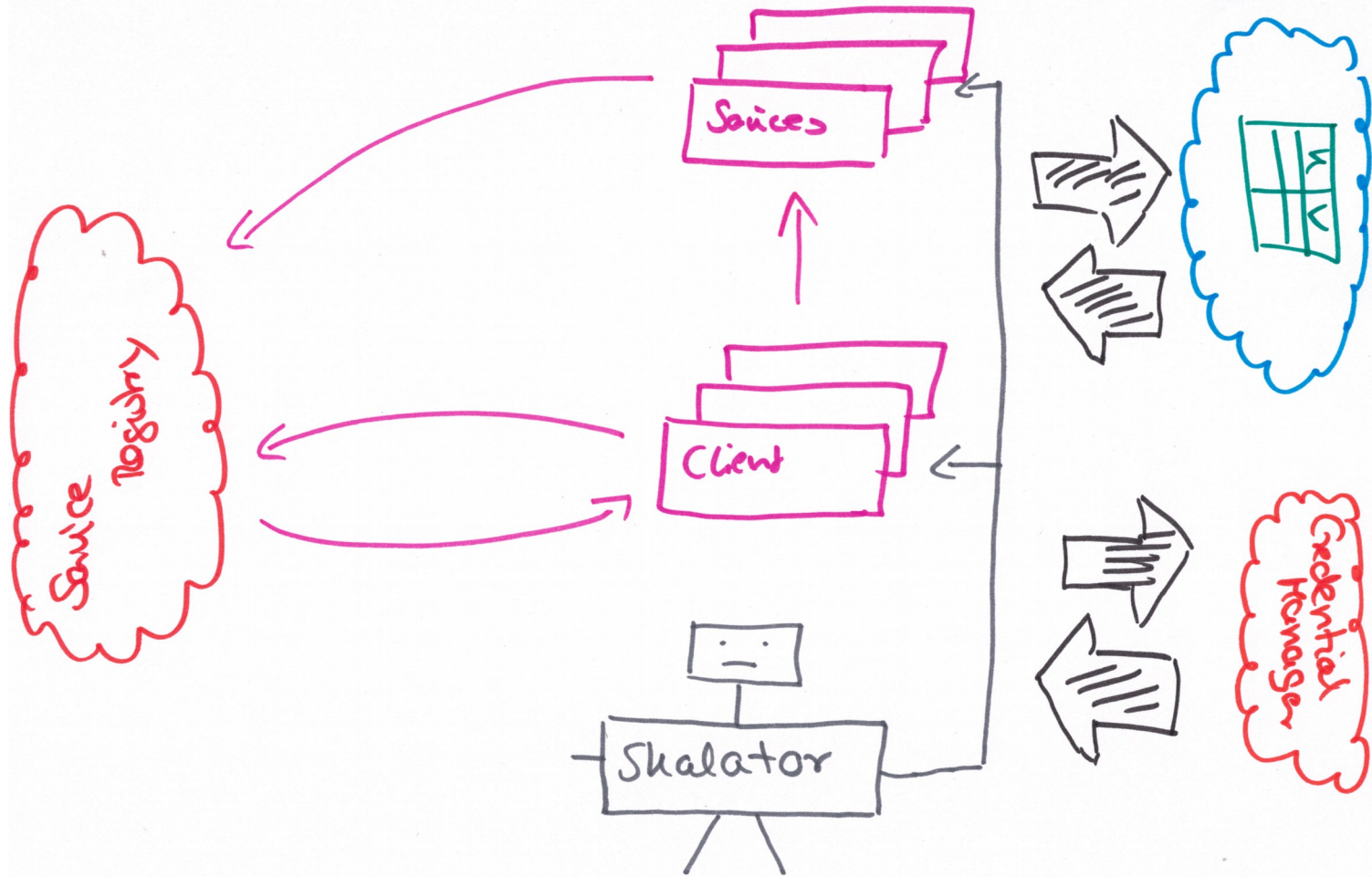
HashiCorp

Vault

- Secret Management von HashiCorp
- Auszug aus der Feature-Liste:
 - tokens
 - passwords
 - certificates
 - encryption keys
 - UI, CLI, HTTP API

Auswirkungen auf die Software-Architektur

- Statt K/V Storage wird jetzt ein Credential Manager angesprochen



Hashicorp-Stack, die neue Silver Bullet?



Netflix Eureka



CLOUD **FOUNDRY**

NGINX

Wann Kubernetes?

Wann Kubernetes?

⚠ Aus unserer Sicht ist Kubernetes ein Rechenzentrumverwaltungssystem. ⚠

- Wenn ich weiß, dass ich alle Feature brauche.
- Wenn du dein Rechenzentrum bis an die technisch mögliche Grenzen auslasten musst
 - Betriebswirtschaftlich
 - Ressourcenfressende Anwendung, das automatisiert gemanagt werden muss
- Schlussendlich muss das Controlling entscheiden, ob Kubernetes oder nicht



Wir Techniker liefern Zahlen als
Entscheidungsgrundlage, welche
Technologie **wirtschaftlich**
Sinn macht.

Fazit

Unix Philosophie

Jeder nimmt sich die Bausteine, die er/sie braucht

Macht Technik nicht zum Selbstzweck

Wirtschaftlichkeit und Fachlichkeit
müssen im Fokus stehen



Fragen?

<https://github.com/sanddorn/InfrastructureAsMicroservice>



mail@sandra-parsick.de



@SandraParsick



xing.to/sparsick



info@bermuda.de



@sanddorn



xing.to/sanddorn

Weitere Informationen

- <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](https://www.flaticon.com) from www.flaticon.com
- Image by [Marc Vanduffel](#) from [Pixabay](#)
- Image by [OpenClipart-Vectors](#) from [Pixabay](#)
- Image by [Francis Ray](#) from [Pixabay](#)