

Cross-Cutting Concerns und andere Microservice-Pattern



Dr. Annegret Junker

- Senior Software Architekt
- Interessensschwerpunkte:
 - Microservices
 - Software Architekturen und zugehörige Vorgehensweisen
 - Selbstorganisierte Teams
 - Agile Vorgehensweisen und Skalierung
 - UX und mobile Anwendungen

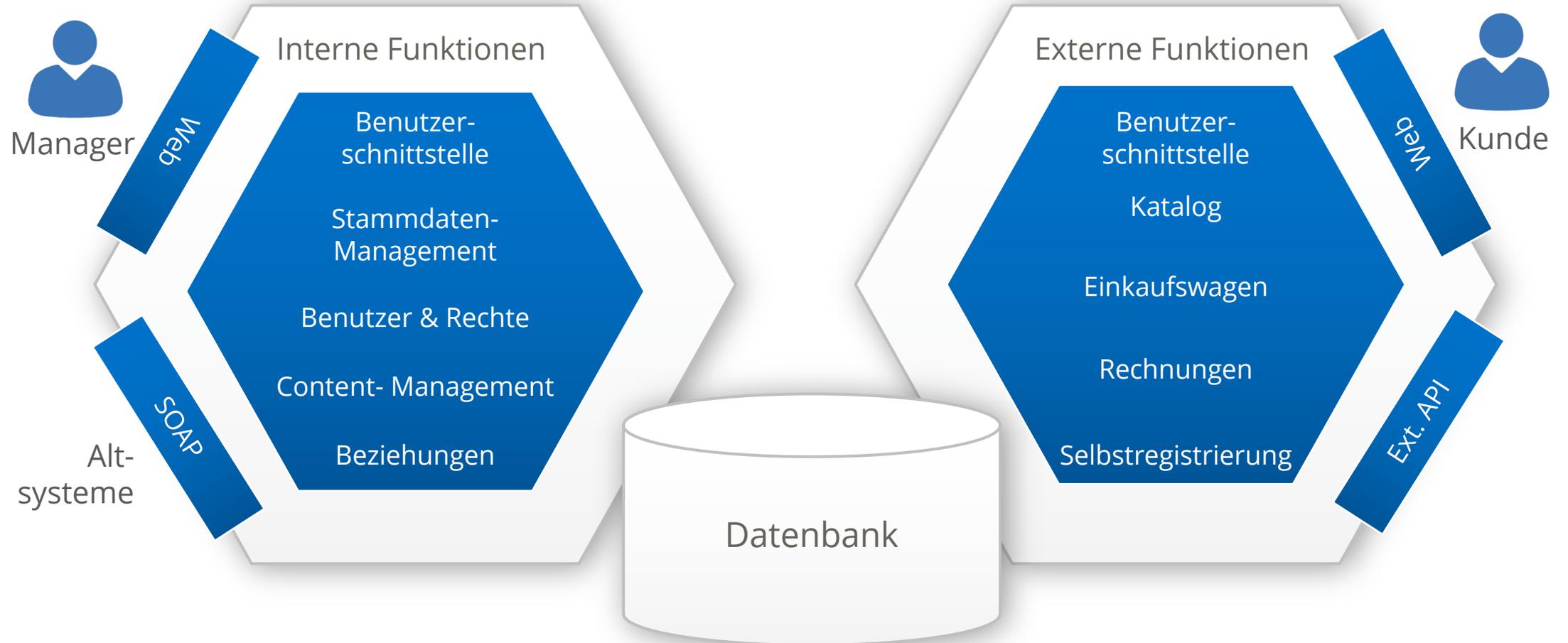
E-Mail: annegret.junker@adesso.de
@Grinsetddy

Web: www.adesso.de

adesso AG, Tassiloplatz 25, 81541 München

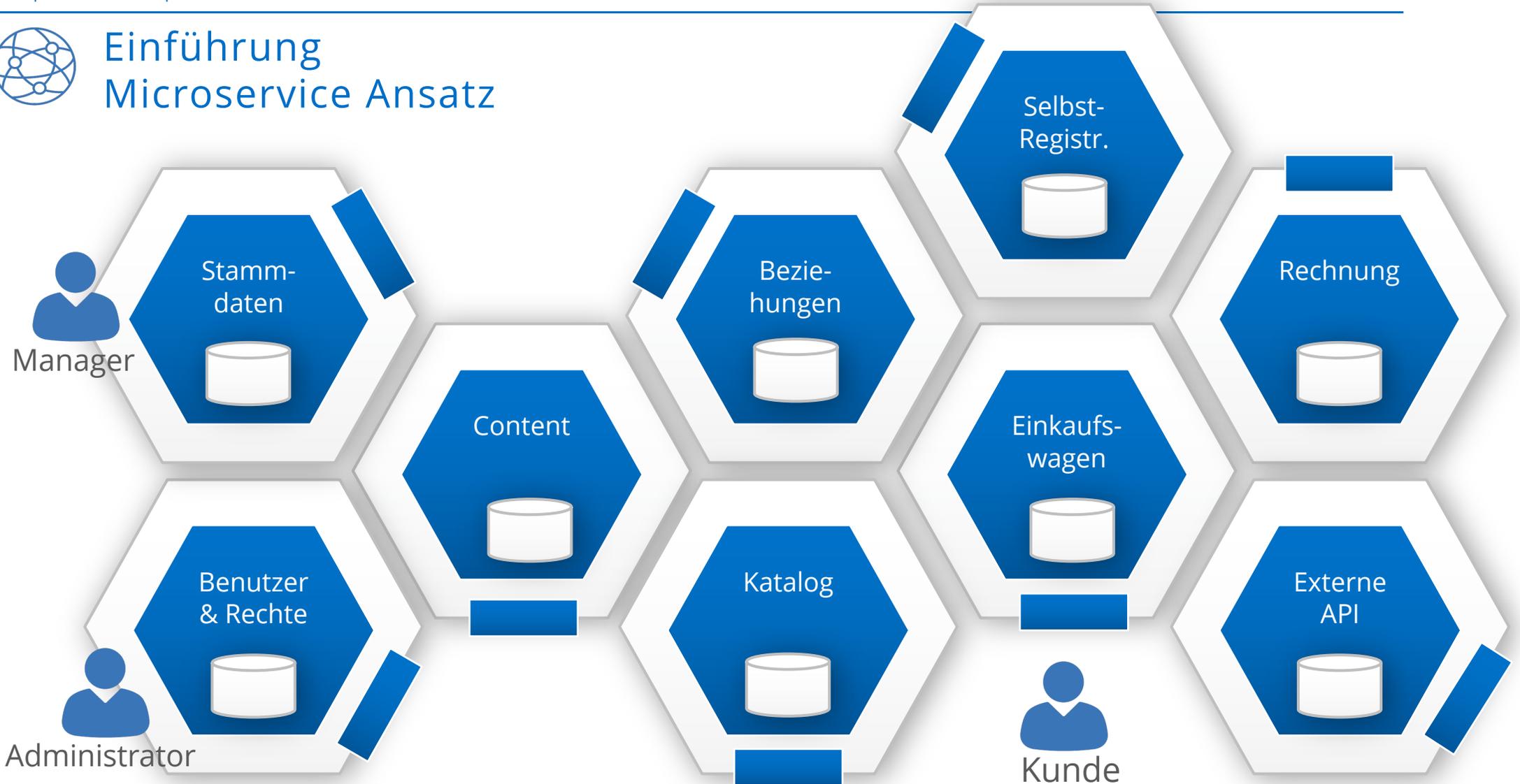


Einführung Klassischer Ansatz





Einführung Microservice Ansatz





Einführung

Neue Lösung: Microservices - Neue Probleme

Höhere Implementierungsaufwände

Kein 2-Phasen-Commit möglich

Mehr Deployments

Höhere Änderungsaufwände

Aufwändige Konfiguration

Höherer

Aufwändigere Integrationstests

Synchronisationsaufwand

Keine einheitliche,
konsistente Datenbank

Aufwändigere
End-To-End-Tests

Höherer

Kommunikationsaufwand

Keine einheitliche UI

Aufwändige Verwaltung von Stammdaten

Höhere Anzahl von zu verwaltenden Betriebseinheiten

Automatisierung notwendig

Höhere Betriebsaufwände

Höhere Hardwarekosten

Uneinheitliches Logging



Pattern Lösungsmuster

- Pattern sind Muster, die wir immer wieder verwenden können, um ähnliche oder gleichartige Probleme zu adressieren
- Pattern werden gefunden, nicht erfunden

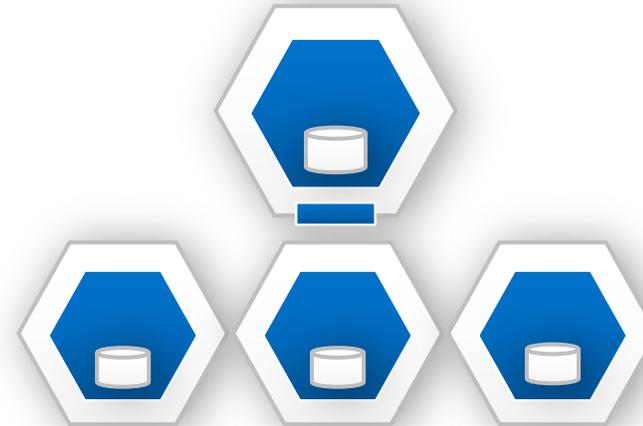
Im Bereich der Softwareentwicklung sind **Architekturmuster** (englisch architectural pattern) in den Arten von Mustern auf oberster Ebene einzuordnen. Im Gegensatz zu Entwurfsmustern (design pattern) oder Idiomen bestimmen sie nicht ein konkretes (meist kleines oder lokales) Teilproblem, sondern die grundlegende Organisation und Interaktion zwischen den Komponenten einer Anwendung.



Pattern

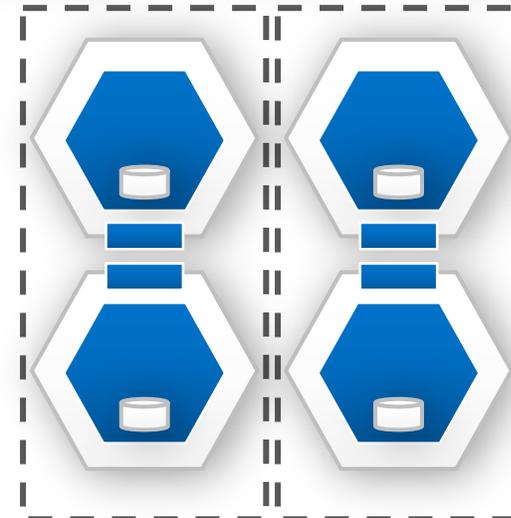
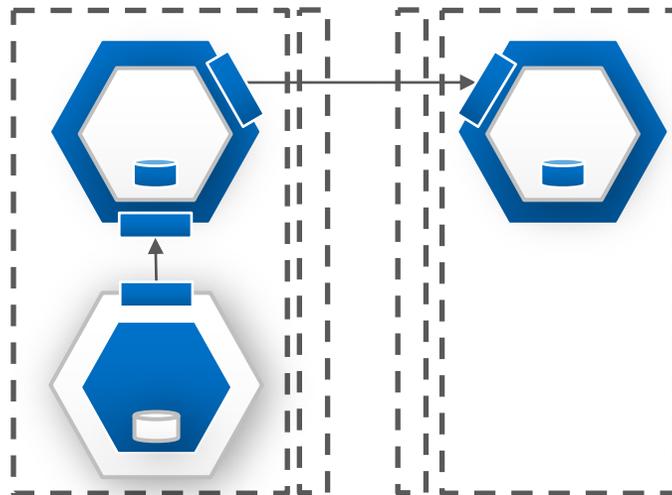
4 gängige Lösungsmuster

Microservice
Chassis
*Microservice
Unterbau*



Externe
Konfiguration

Ambassador
Botschafter



Sidecar
Beiwagen



Pattern

Microservices - Implementierungsbeispiel

Stammdaten Service

Liefert Buchstaben des Alphabets als Stammdaten

<https://github.com/Grinseteddy/CrossCuttingMasterData>

Orchestrierungsdienst

Ruft nacheinander Filter und dann Mapping-Dienst auf
<https://github.com/Grinseteddy/CrossCuttingConcernsOrchestra>

Filter

Filtert einen Buchstaben aus einem gegebenen String

<https://github.com/Grinseteddy/CrossCuttingConcernsFilter>

Mapping-Dienst

Ersetzt einen Buchstaben mit einem anderen

<https://github.com/Grinseteddy/CrossCuttingConcernsMapper>

Konfigurationsdienst

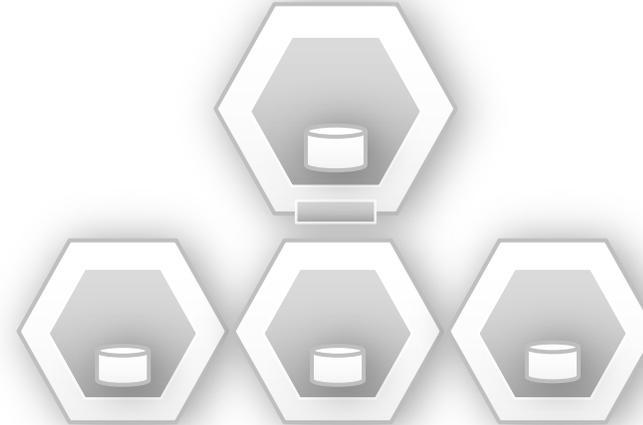
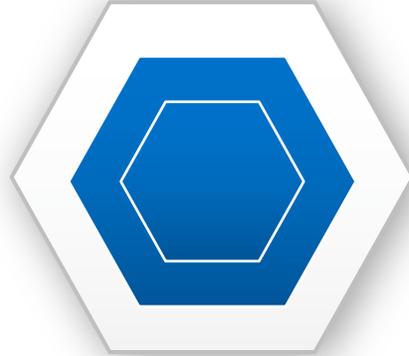
Liefert Indizes für Filter und Mapping Service, Liefert Adressen der Services

<https://github.com/Grinseteddy/CrossCuttingConcernsConfiguration>



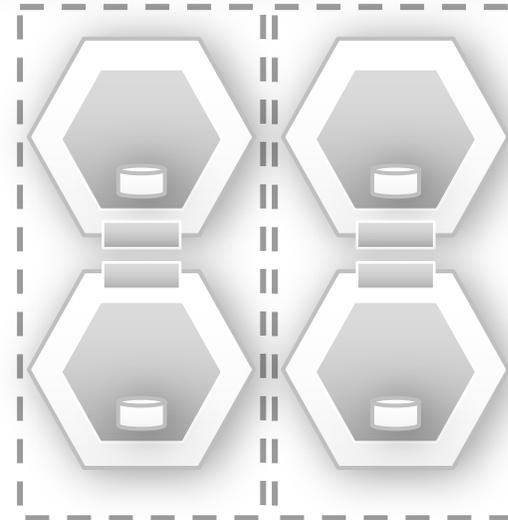
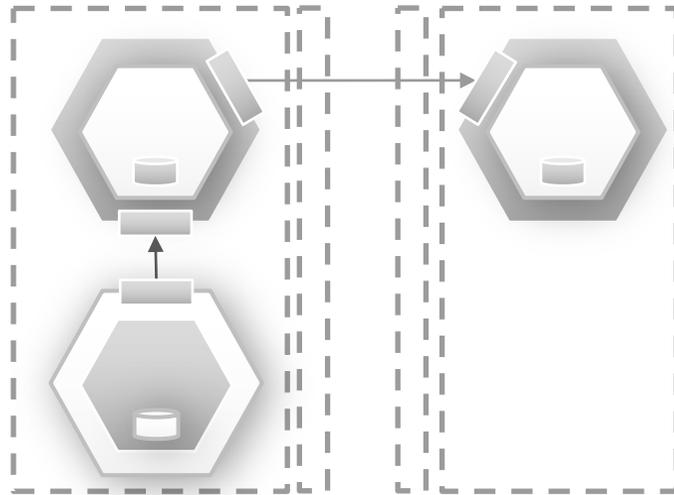
Pattern Microservice Chassis

Microservice
Chassis
*Microservice
Unterbau*



Externe
Konfiguration

Ambassador
Botschafter



Sidecar
Beiwagen



Microservice Chassis Problem

Problem

- Dienste müssen leicht und unkompliziert erzeugbar sein.
- Alle Dienste müssen die gleichen Probleme wie Logging, Serviceregistration, usw. lösen.
- Solche Probleme sind normalerweise spezifisch zu der verwendeten Technologie.





Microservice Chassis Lösung

Lösung

- Benutze Frameworks, die schon entsprechende Umgebungen schaffen
- Z.B. Spring Boot und Spring Cloud
- Benutze Oberflächen um ganze Projekte zu kreieren z.B. Spring Initializr
- Nutze die Unterstützung Deiner IDE



Walls, C.: Initializing a Spring Boot project with Spring Initializr, <https://freecontent.manning.com/wp-content/uploads/initializing-a-spring-boot-project-with-spring-initializr.pdf>, abgerufen 18.4.2019

<https://start.spring.io/>, abgerufen 18.4.2019

IntelliJ Help, <https://www.jetbrains.com/help/idea/spring-boot.html>, abgerufen 18.4.2019



Microservice Chassis Beispiel

Spring Initializr

https://start.spring.io

List of dependencies for **Spring Boot 2.1.4.RELEASE**

Core

- DevTools:** Spring Boot Development Tools
- Lombok:** Java annotation library which helps to reduce boilerplate code and code faster
- Configuration Processor:** Generate metadata for your custom configuration keys
- Session:** API and implementations for managing a user's session information
- Cache:** Spring's Cache abstraction
- Validation:** JSR-303 validation infrastructure (already included with web)
- Retry:** Provide declarative retry support via spring-retry
- Aspects:** Create your own Aspects using Spring AOP and AspectJ

Web

- Web:** Servlet web application with Spring MVC and Tomcat
- Reactive Web:** Reactive web applications with Spring WebFlux and Netty

Update dependencies Cancel

© 2013-2019 Pivotal Software
start.spring.io is powered by

Generate Project

Projekt-Abhängigkeiten mit
Gradle oder Maven

Sprachen: Java, Groovy,
Kotlin



Microservice Chassis Beispiel

The screenshot shows the 'New Project' dialog in an IDE. The 'Spring Boot' version is set to 2.1.4. The 'Dependencies' list on the left includes categories like Core, Web, Template Engines, Security, SQL, NoSQL, Messaging, Cloud Core, Cloud Support, Cloud Config, Cloud Discovery, Cloud Routing, Cloud Circuit Breaker, Cloud Tracing, Cloud Messaging, Cloud Contract, Pivotal Cloud Foundry, Amazon Web Services, Azure (highlighted), and Google Cloud Platform. The 'Selected Dependencies' list on the right includes Core (DevTools, Lombok, Configuration Process) and Azure (Azure Active Directory). The 'Azure Active Directory' dependency is selected in the middle list, with a description: 'Spring Security integration with Azure Active Directory for authentication'. Below the description are links for 'Using Active Directory' and 'Reference doc'. At the bottom, there are 'Cancel', 'Previous', and 'Next' buttons.

Dependencies

Spring Boot 2.1.4

Selected Dependencies

Core

- DevTools
- Lombok
- Configuration Process

Azure

- Azure Active Directory

Azure Active Directory

Spring Security integration with Azure Active Directory for authentication

[Using Active Directory](#)

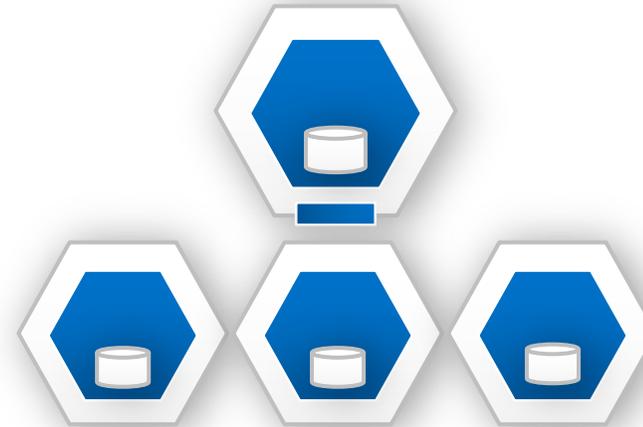
[Reference doc](#)

? Cancel Previous Next



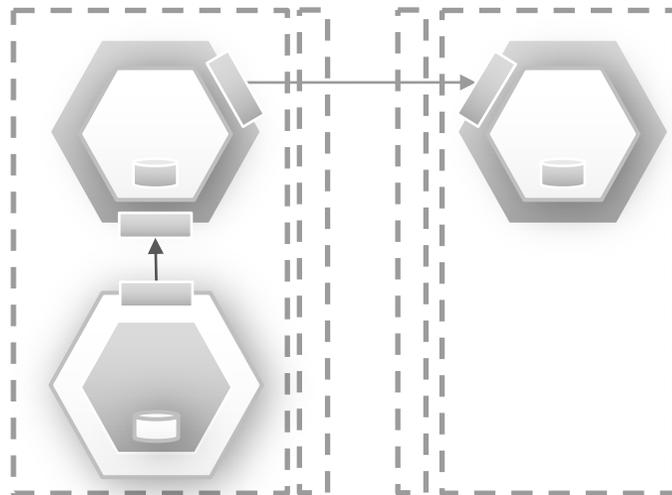
Pattern Externe Konfiguration

Microservice
Chassis
*Microservice
Unterbau*

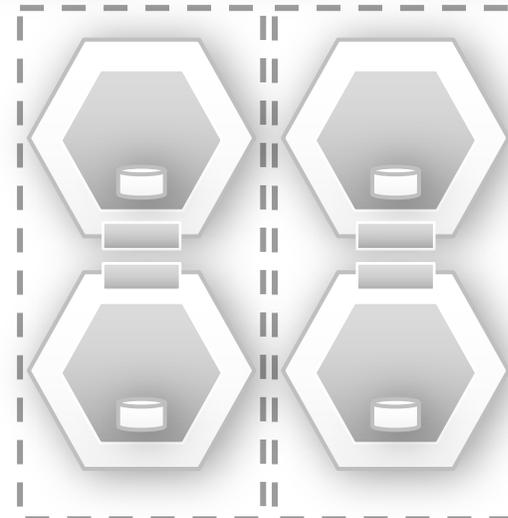


Externe
Konfiguration

Ambassador
Botschafter



Sidecar
Beiwagen

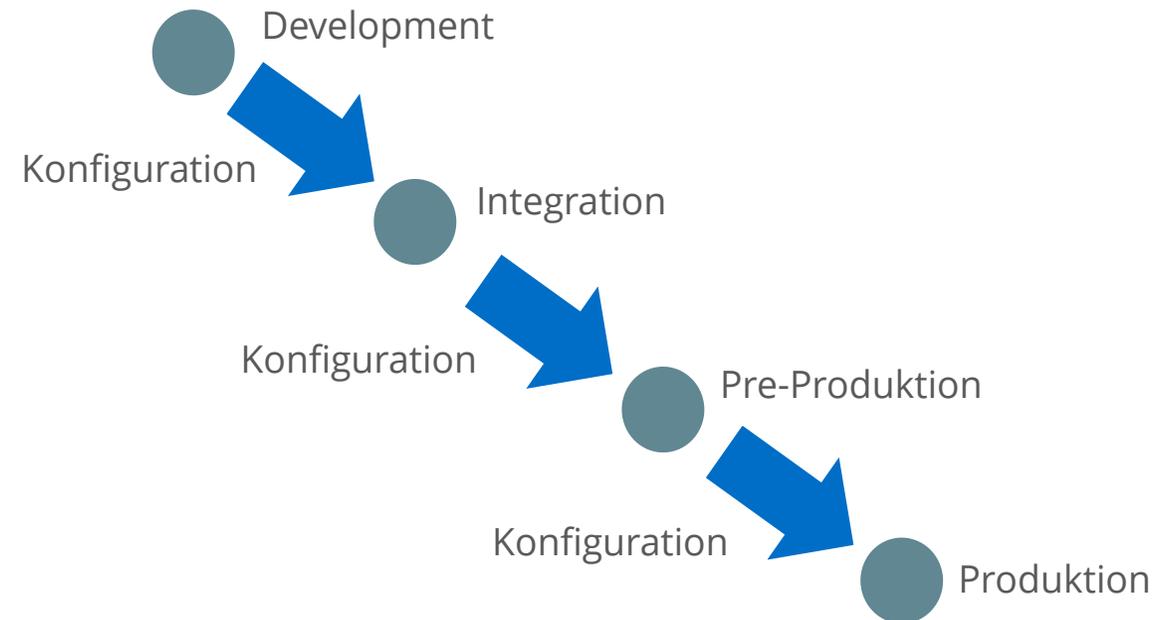




Externe Konfiguration Problem

Problem

- Dienste müssen in unterschiedlichen Umgebungen laufen und brauchen hierfür angepasste Parameter



Richardson, C: Pattern: Externalized Configuration, <https://microservices.io/patterns/externalized-configuration.html>,
abgerufen 12.4.2019

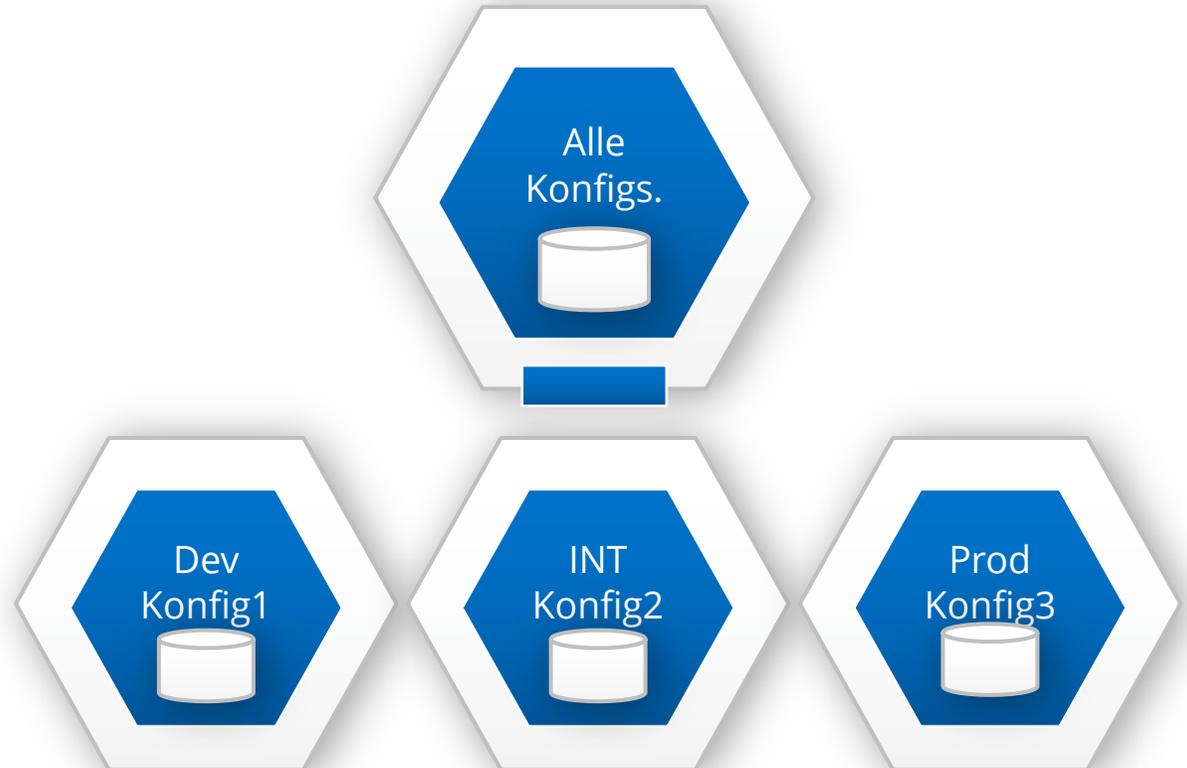
External configuration pattern, <https://docs.microsoft.com/en-us/azure/architecture/patterns/external-configuration-store>,
abgerufen 12.4.2019



Externe Konfiguration Lösung

Lösung

- Alle Konfigurationen werden außerhalb der Applikation gehalten. Beim Start der Applikation werden die Konfiguration in Abhängigkeit ihrer Umgebung geladen.
- Z.B.
/{ConfigServer}:8888/Filter/Development



Richardson, C: Pattern: Externalized Configuration, <https://microservices.io/patterns/externalized-configuration.html>,
abgerufen 12.4.2019

External configuration pattern, <https://docs.microsoft.com/en-us/azure/architecture/patterns/external-configuration-store>,
abgerufen 12.4.2019



Externe Konfiguration Beispiel

Konfigurations-Server

```
server.port=8888
spring.cloud.config.server.git.uri=file://Users/annegretjunker/Documents/Backup/configserver/configdir

@EnableConfigServer
@SpringBootApplication
public class ConfigserverApplication {

    public static void main(String[] args) { SpringApplication.run(ConfigserverApplication.class, args); }

}
```

Konfigurations-Client

```
spring.application.name=filter
# N.B. this is the default:
spring.cloud.config.uri=http://localhost:8888
```

```
[C02W51TPHTDD:~ annegretjunker$ cd Documents/Backup/con]
figserver/configdir/
[C02W51TPHTDD:configdir annegretjunker$ git commit filt]
er.properties -mIndexChanged
[master de5fc40] IndexChanged
 1 file changed, 1 insertion(+), 1 deletion(-)
C02W51TPHTDD:configdir annegretjunker$
```



Externe Konfiguration Beispiel

← → ↻ 🏠 ⓘ localhost:8888/filter/development



```
{ "name": "filter", "profiles":  
  [ "development", "label": null, "version": "de5fc40054bceadae714e6b798c96a4ca09dc900", "state": null, "propertySources":  
    [ { "name": "file://Users/annegretjunker/Documents/Backup/configserver/configdir/filter.properties", "source":  
      { "server.port": "8084", "logging.file": "filter.log", "index": "2" } } ] }
```

Empfohlenes Tutorial: <https://spring.io/guides/gs/centralized-configuration/>



Externe Konfiguration Beispiel

The screenshot shows a web browser's developer tools interface. At the top, there's a browser tab labeled 'New Tab' and the address bar shows 'localhost:8087/actuati'. The environment is set to 'No Environment'. Below the address bar, the request method is 'POST', the URL is 'localhost:8087...', and the parameters are 'Params'. A blue 'Send' button is visible. The status bar shows 'Status: 200 OK', 'Time: 372 ms', and 'Size: 193 B'. The response body is displayed in JSON format:

```
[  
  1  [   
  2    "config.client.version",   
  3    "message"   
  4  ]
```

. The 'Body' tab is selected, and the response is shown in 'Pretty' JSON format. The 'Auth' tab is also visible, showing 'Inherit auth from parent' and a note: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'.

Beispiel Konfigurationsserver: <https://github.com/Grinsetddy/CrossCuttingConcernsConfiguration>

Beispiel Konfigurationsclient: <https://github.com/Grinsetddy/CrossCuttingConcernsConfigClient>



Externe Konfiguration Beispiel

HANDS ON

- Starte Configserver
- Starte Configclient
- Prüfe Einstellungen <http://localhost:8888/configclient/default> → Nachricht: Hallo Annegret!
- Prüfe Nachricht: <http://localhost:8087/message> → Nachricht: Hallo Annegret!
- Verändere Nachricht in configclient.properties (~/.Documents/Backup/configserver/configdir/: Hallo Jax!
- Prüfe die Einstellungen für GIT in application.properties in ConfigServer
- Git commit for configclient.properties (`git commit configclient.properties -mMessageChanged`)
- Rufe <http://localhost:8087/message> → Nachricht: Hallo Annegret!
- Sende Refresh zum Client: Postman `localhost:8087/actuator/refresh`
- Prüfe Nachricht: <http://localhost:8087/message> → Nachricht: Hallo Jax!

Beispiel Konfigurationsserver: <https://github.com/Grinseteddy/CrossCuttingConcernsConfiguration>

Beispiel Konfigurationsclient: <https://github.com/Grinseteddy/CrossCuttingConcernsConfigClient>



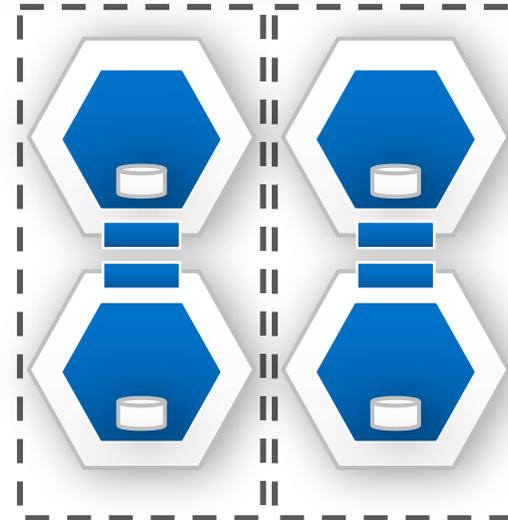
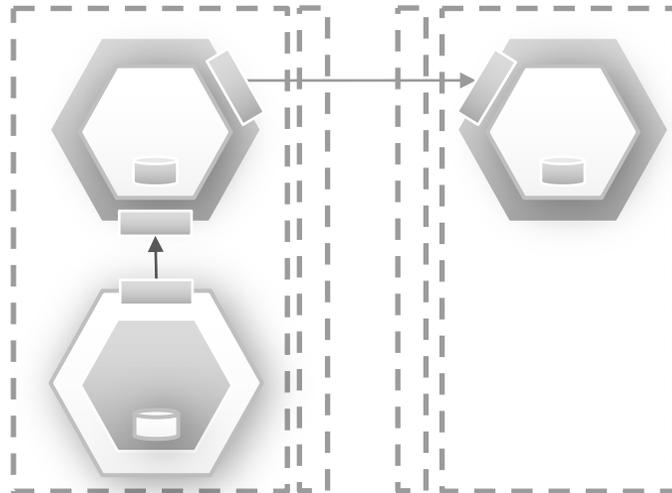
Pattern Sidecar

Microservice
Chassis
*Microservice
Unterbau*



Externe
Konfiguration

Ambassador
Botschafter



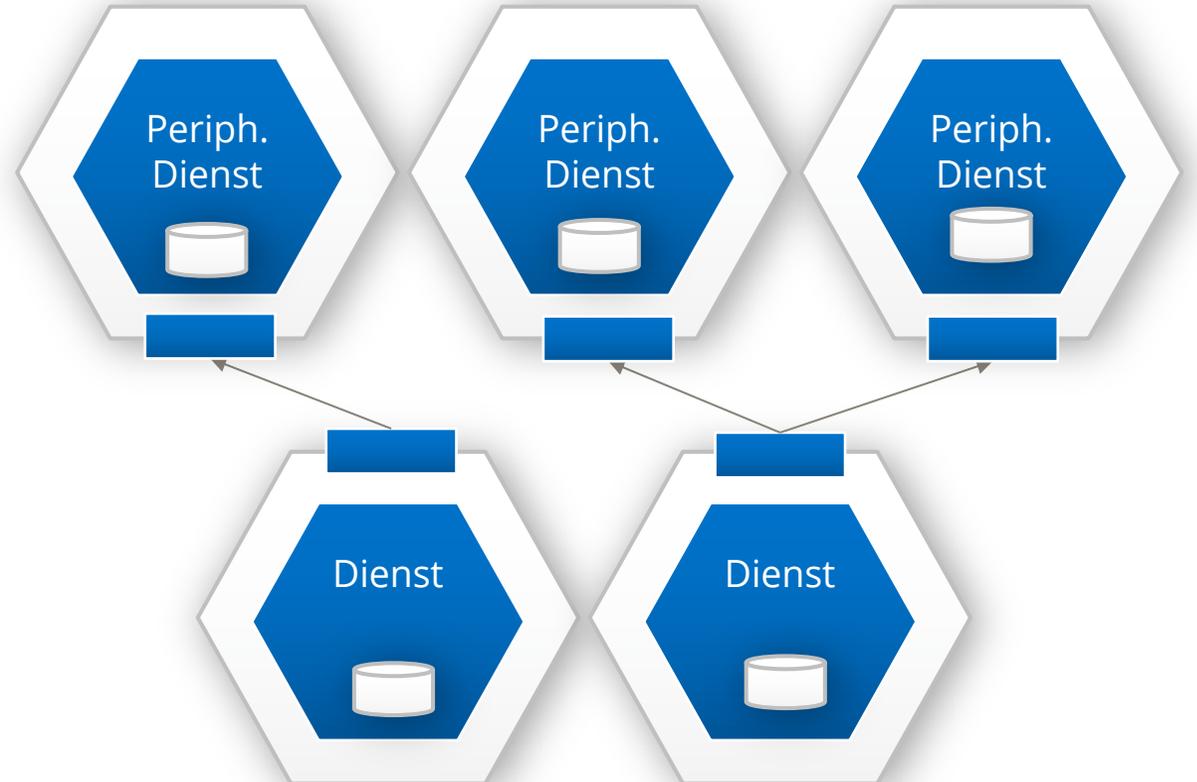
Sidecar
Beiwagen



Sidecar Problem

Problem

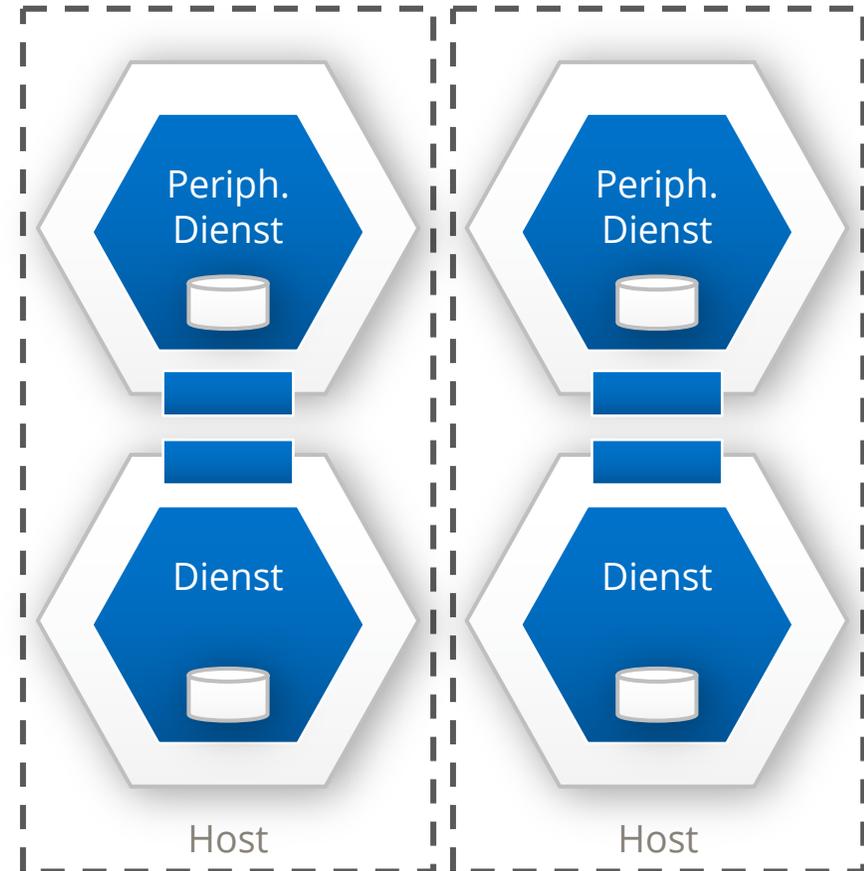
- Dienste brauchen oft begleitende Funktionalität wie Monitoring oder Logging.
- Als vollständig separierte Dienste brauchen sie dann extremen Verwaltungsaufwand für jede einzelne Funktionalität.
- Nicht separiert werden häufig gleiche Aufgaben mehrfach (und dann unterschiedlich implementiert).



Sidecar Lösung

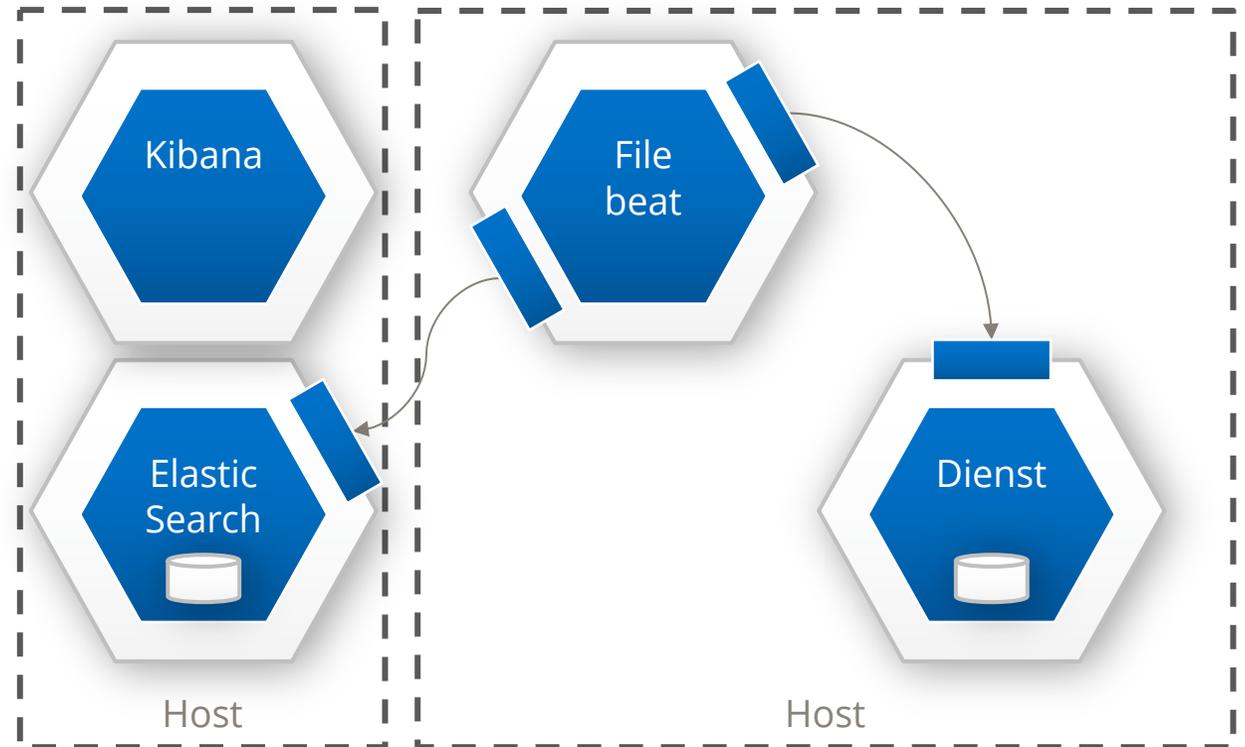
Lösung

- Getrennte Aufgaben einer Anwendung auf einem gemeinsamen Host



Sidecar Beispiel

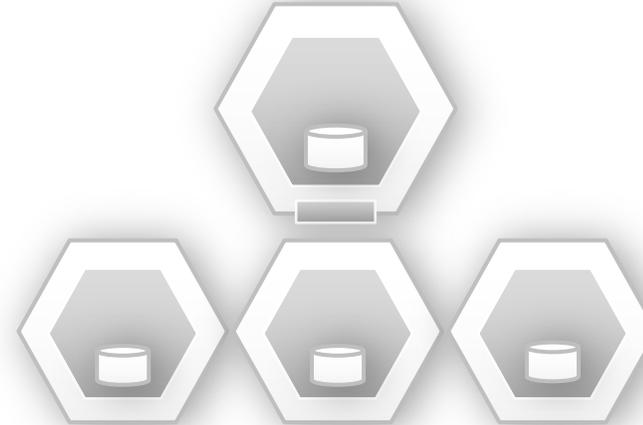
- Filebeat wird zusammen mit Applikation auf einem Host installiert
- ElasticSearch wird durch Filebeat versorgt
- → Standard „ELK stack“ Lösung





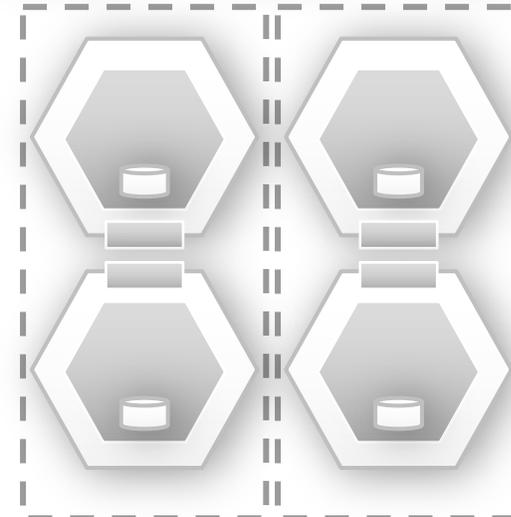
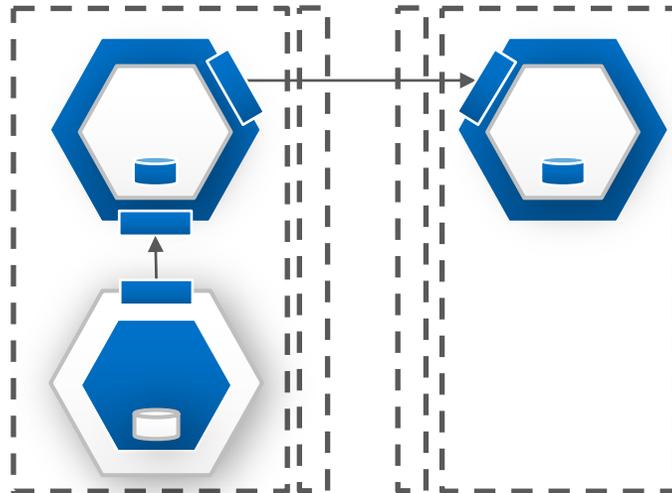
Pattern Ambassador

Microservice
Chassis
*Microservice
Unterbau*



Externe
Konfiguration

Ambassador
Botschafter



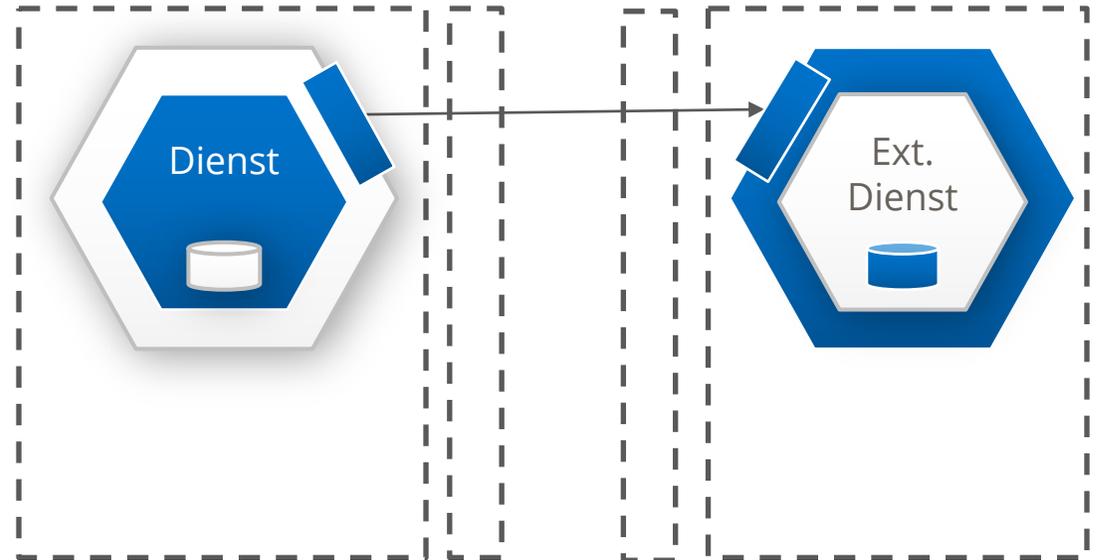
Sidecar
Beiwagen



Ambassador Problem

Problem

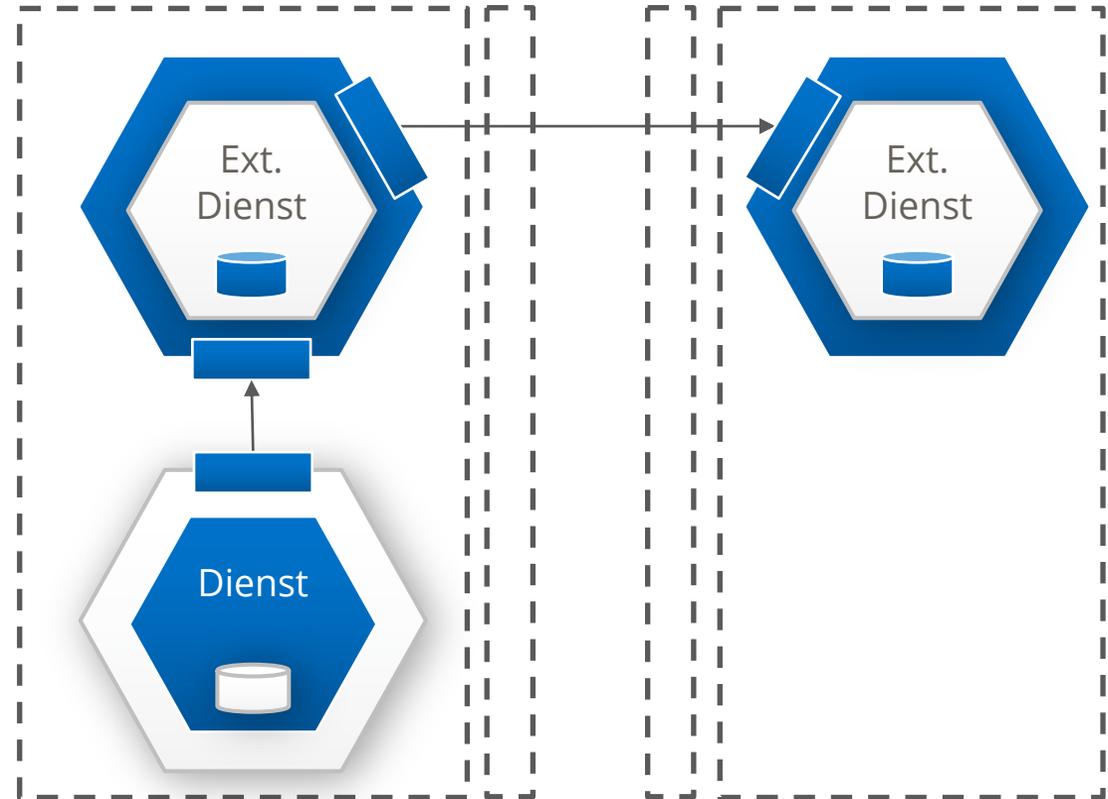
- Netzwerkaufrufe können teuer und unzuverlässig sein
- Insbesondere über mehrere Instanzen hinweg werden sie fehleranfällig und langsam



Ambassador Lösung

Lösung

- Der externe Dienst bekommt ein Client auf der Dienste-Seite und kann ohne teure Netzwerk-Kommunikation aufgerufen werden.
- Der Aufruf ist vollkommen transparent für den Dienst.





Ambassador Beispiel

```
public class MasterDataController {  
  
    static Logger logger=LoggerFactory.getLogger(MasterDataController.class);  
  
    @GetMapping(value = "/Character/{index}")  
    @ResponseBody  
    public MasterData masterDataAt(@PathVariable("index") String index) throws Exception {
```

```
@RestController  
public class MasterDataClientController {  
  
    static Logger logger=LoggerFactory.getLogger(MasterDataClientController.class);  
  
    private MasterData answer;  
  
    public void MasterDataClientController() {  
    }  
  
    @GetMapping(value = "/Character/{index}")  
    @ResponseBody  
    public MasterData masterDataAt(@PathVariable("index") String index) throws Exception {
```



Externe Konfiguration Beispiel

HANDS ON

- Starte Configserver
- Starte Masterdata Server
- Prüfe aktives Alphabet am Server <http://localhost:8083/small>, <http://localhost:8083/capital> → Standardalphabet
- Starte Masterdata Client
- Prüfe aktives Alphabet am Client <http://localhost:8088/Character/2> → Standardalphabet
- Ersetze Buchstaben am Server `PUT localhost:8083/capital/2/X`
- Prüfe aktives Alphabet am Server <http://localhost:8083/capital> → Großbuchstaben mit X statt D
- Prüfe aktives Alphabet am Client <http://localhost:8088/Character/2> → Standardalphabet
- Sende Refresh zum Client `PUT localhost:8088/refresh/1`
- Prüfe aktives Alphabet am Client <http://localhost:8088/Character/2> → Großbuchstaben mit X statt D

Beispiel Master Data Client: <https://github.com/Grinseteddy/CrossCuttingMasterDataClient>

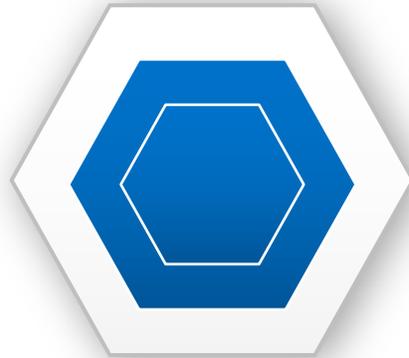
Beispiel Master Data Service: <https://github.com/Grinseteddy/CrossCuttingMasterData>



Pattern 4 gängige Lösungsmuster

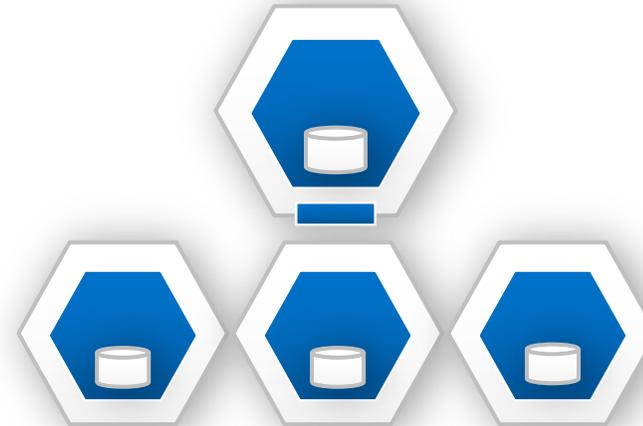
Baue schnell,
einfach und
verlässlich Dienste

Microservice
Chassis
*Microservice
Unterbau*



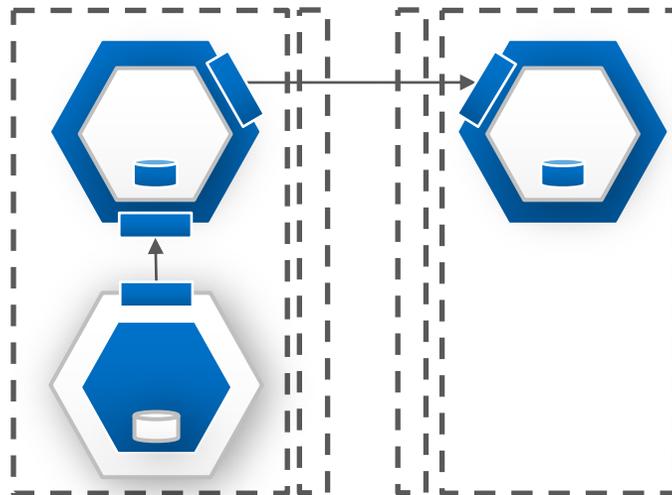
Vermeide
unnötigen
Aufwand
unterschiedlichen
Umgebungen

Externe
Konfiguration



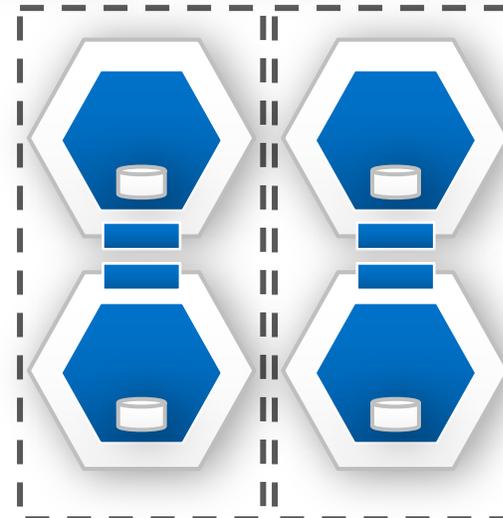
Ambassador
Botschafter

Vermeide
aufwändige
Netzwerk-
Kommunikation



Sidecar
Beiwagen

Vermeide
Verwaltungs- und
Konfigurations-
aufwand





Zusammenfassung Pattern sind hilfreich

- Lösungsmuster sind insbesondere hilfreich um immer wiederkehrende Probleme zu lösen
- Muster können in ein Dienste-Chassis gepackt werden und stehen damit verlässlich und einheitlich zur Verfügung
- Insbesondere unterstützende Dienste wie Logging, Konfiguration, Stammdatenservices werden so verlässlich und stabil