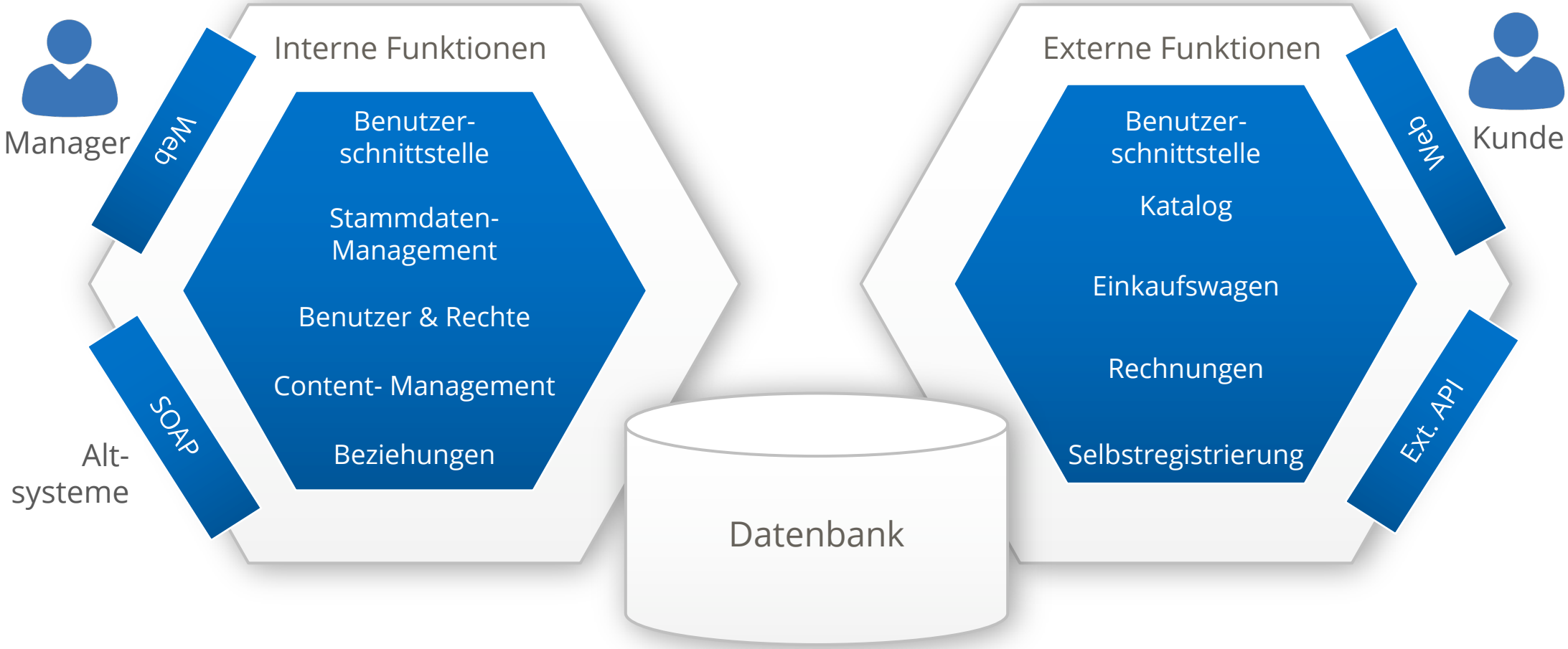
Cross-Cutting Concerns und andere Microservice-Pattern

Dr. Annegret Junker

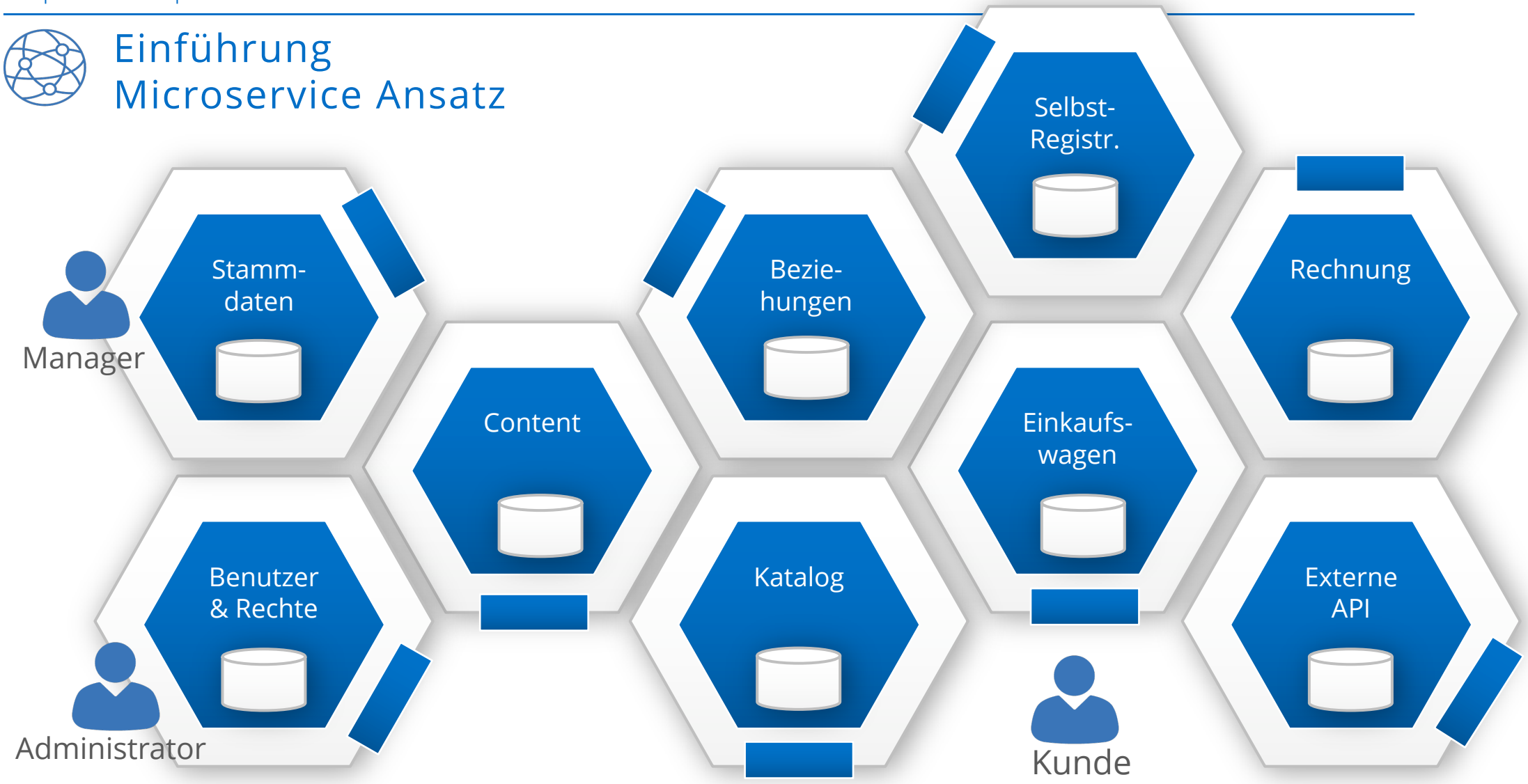
- Senior Software Architekt
- Interessensschwerpunkte:
  - Microservices
  - Software Architekturen und zugehörige Vorgehensweisen
  - Selbstorganisierte Teams
  - Agile Vorgehensweisen und Skalierung
  - UX und mobile Anwendungen

E-Mail: annegret.junker@adesso.de
@Grinseteddy
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 Synchronisationsaufwand

Aufwändigere Integrationstests

Keine einheitliche, konsistente Datenbank

Höherer Kommunikationsaufwand

Keine einheitliche UI

Aufwändigere End-To-End-Tests

Aufwändige Verwaltung von Stammdaten

Höhere Anzahl von zu verwaltenden Betriebseinheiten

Automatisierung notwendig

Höhere Betriebsaufwände

Höhere Hardwarekosten

Uneinheitliches Logging

Pattern
Lösungsmuster

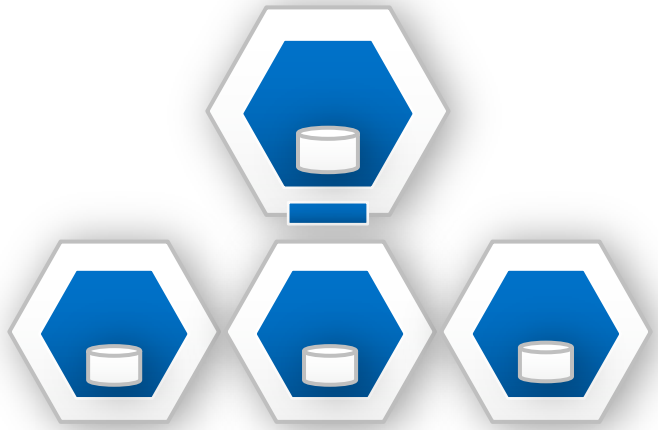
- Pattern sind Muster, die wir immer wieder verwenden können, um ähnliche oder gleichartige Probleme zu adressieren
- Pattern werden gefunden, nicht erfunden

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.

Vergleiche: https://de.wikipedia.org/wiki/Architekturmuster, abgerufen 18.4.2019

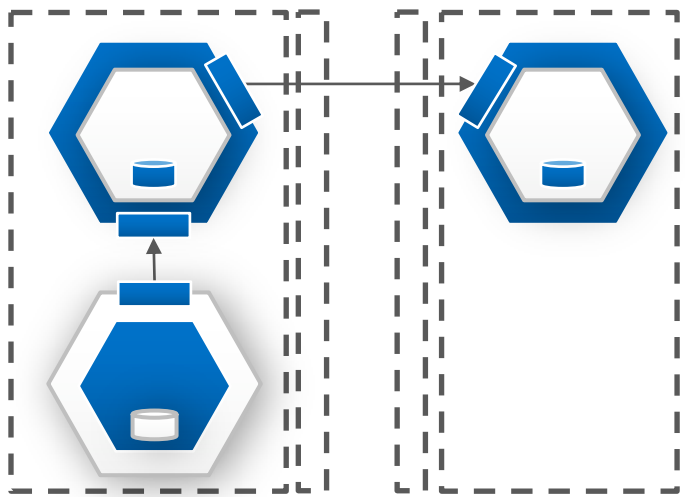
Pattern
4 gängige Lösungsmuster

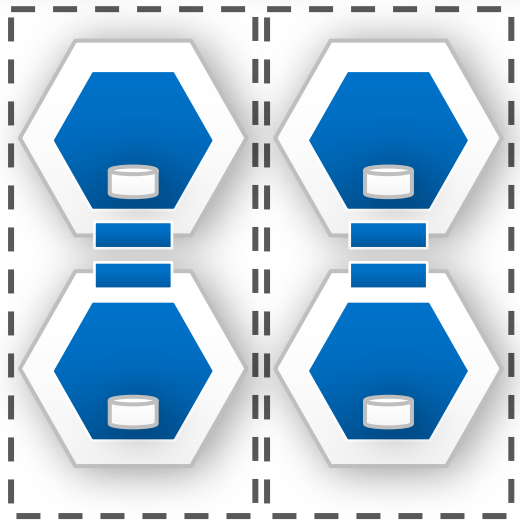
Microservice Chassis
Microservice Unterbau



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

Konfigurations-dienst

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

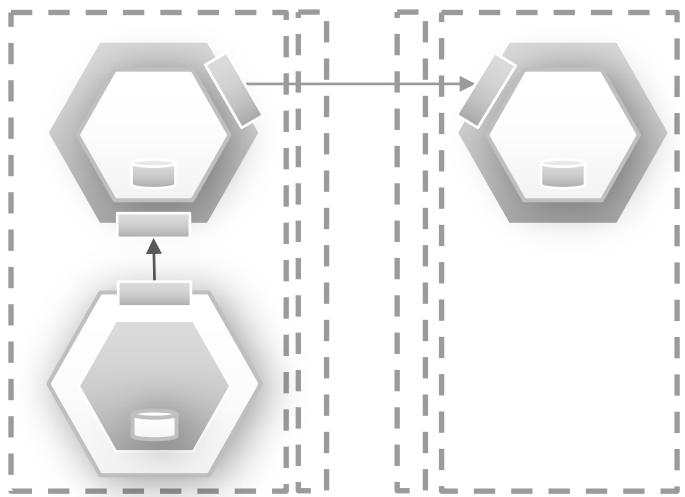
https://github.com/Grinseteddy/CrossCuttingConcernsConfiguration

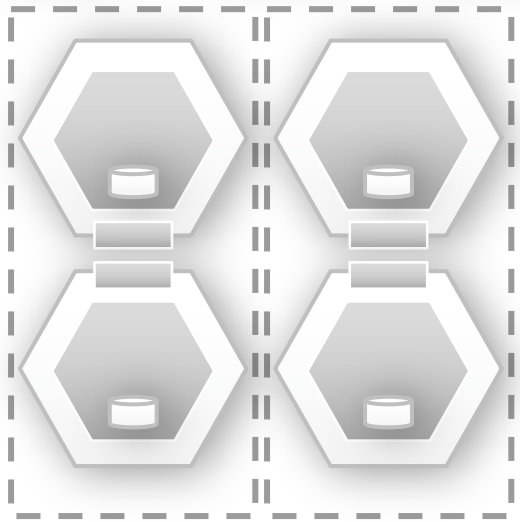
Pattern
Microservice Chassis

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.

Richardson, C.: Pattern: Microservice Chassis, https://microservices.io/patterns/microservice-chassis.html, abgerufen 18.4.2019

Microservice Chassis Lösung

Lösung

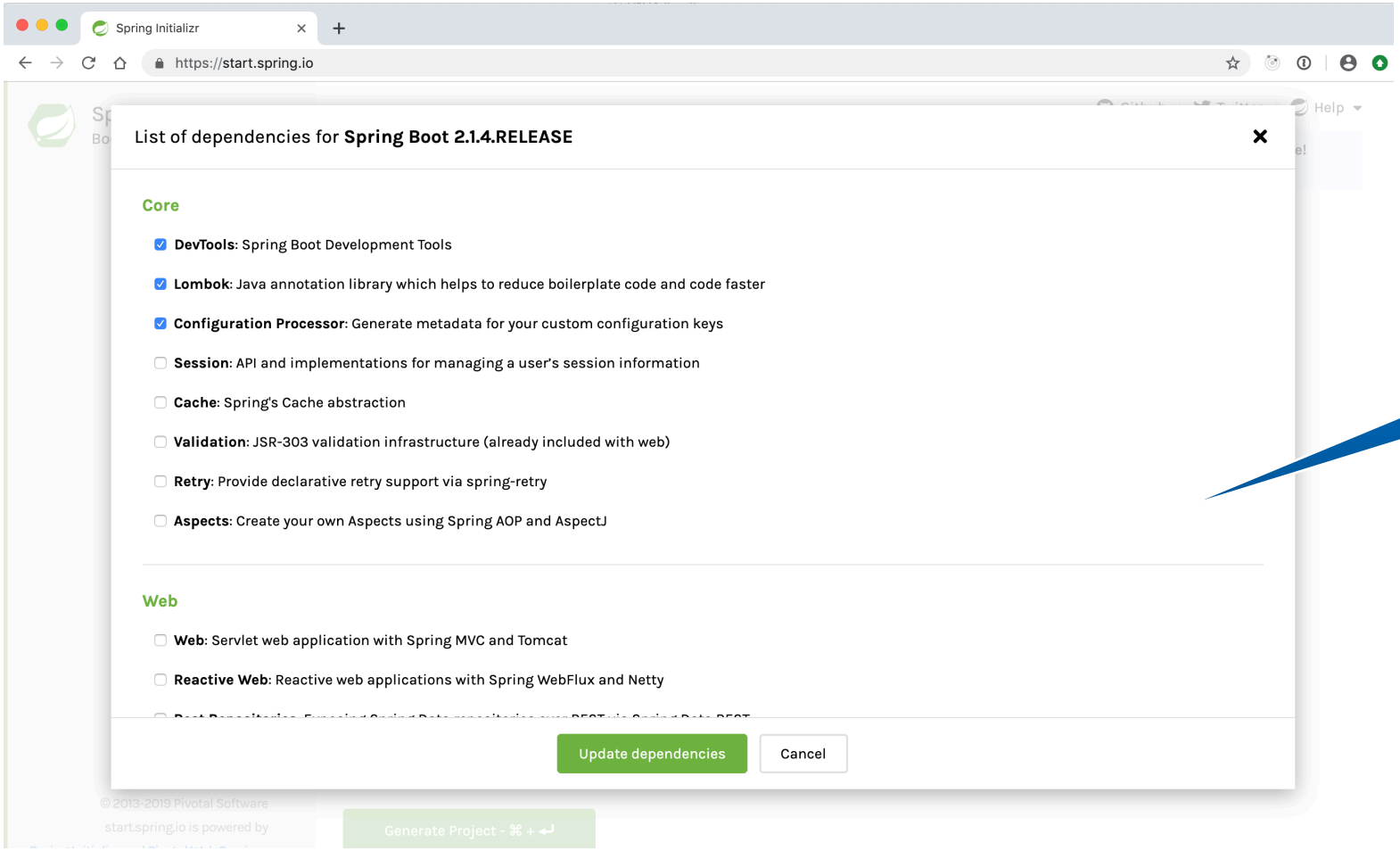
- Benutze Frameworks, die schon entsprechende Umgebungen schaffen
- Z.B. Spring Boot und Spring Cloud
- Benutze Oberflächen um ganze Projekte zu kreieren z.B. Spring Initialzr
- Nutze die Unterstützung Deiner IDE

spring BOOT

Spring Initializr
Bootstrap your application

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

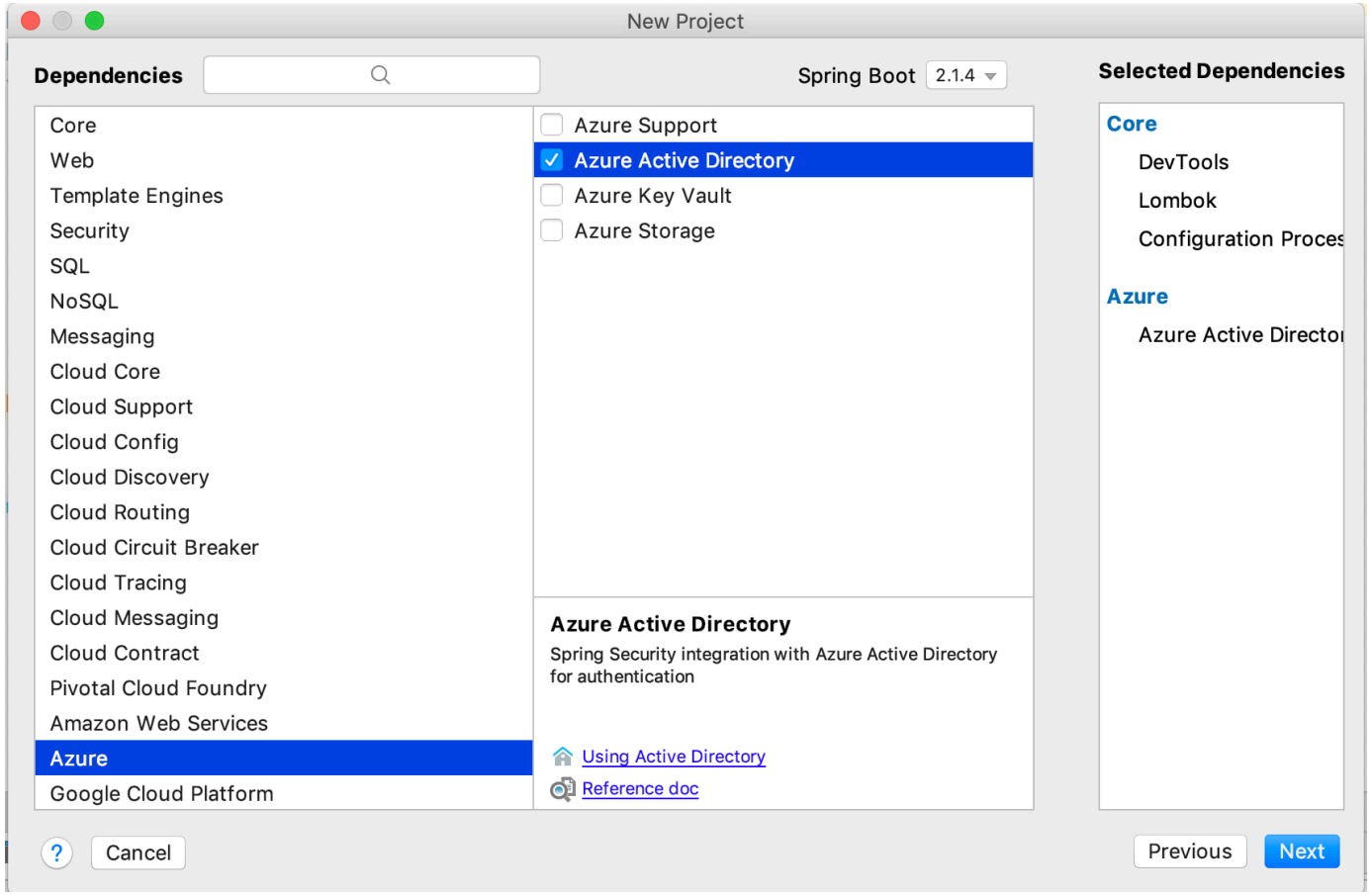


Projekt-Abhängigkeiten mit Gradle oder Maven

Sprachen: Java, Groovy, Kotlin

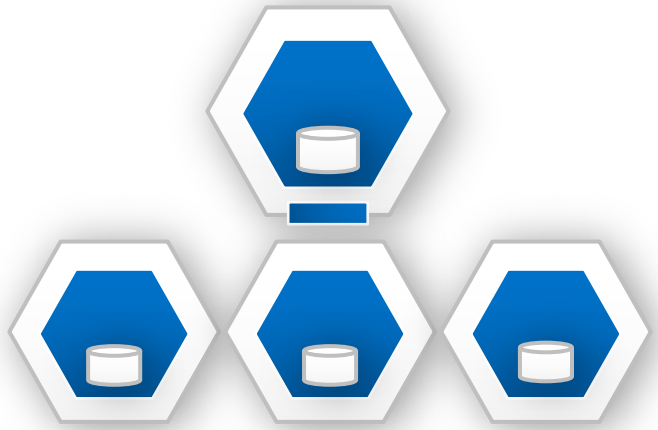
https://start.spring.io/, abgerufen 06.05.2019

Microservice Chassis Beispiel



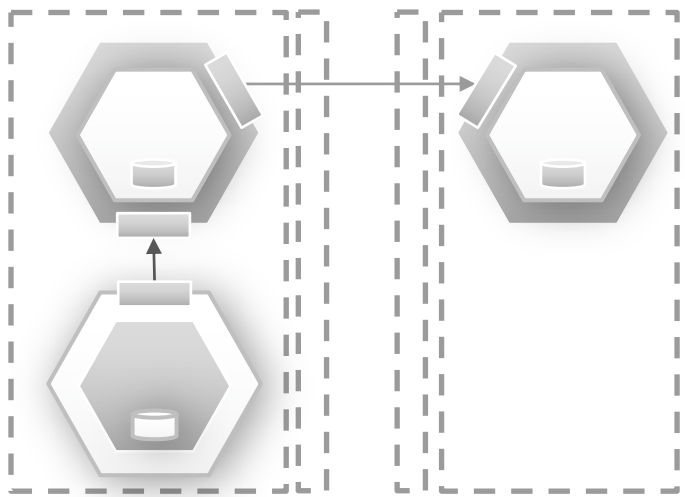
Pattern
Externe Konfiguration

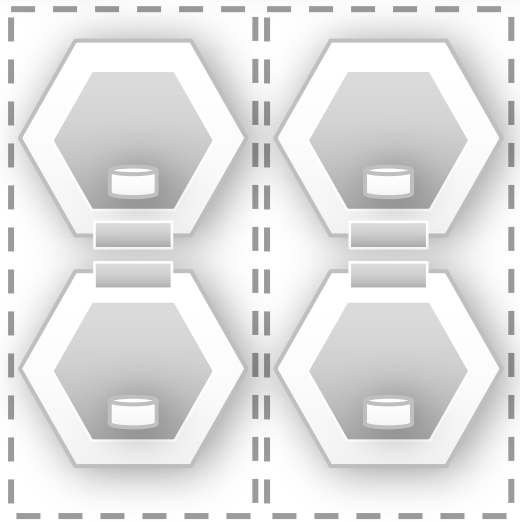
Microservice Chassis
Microservice Unterbau



Externe Konfiguration

Ambassador
Botschafter



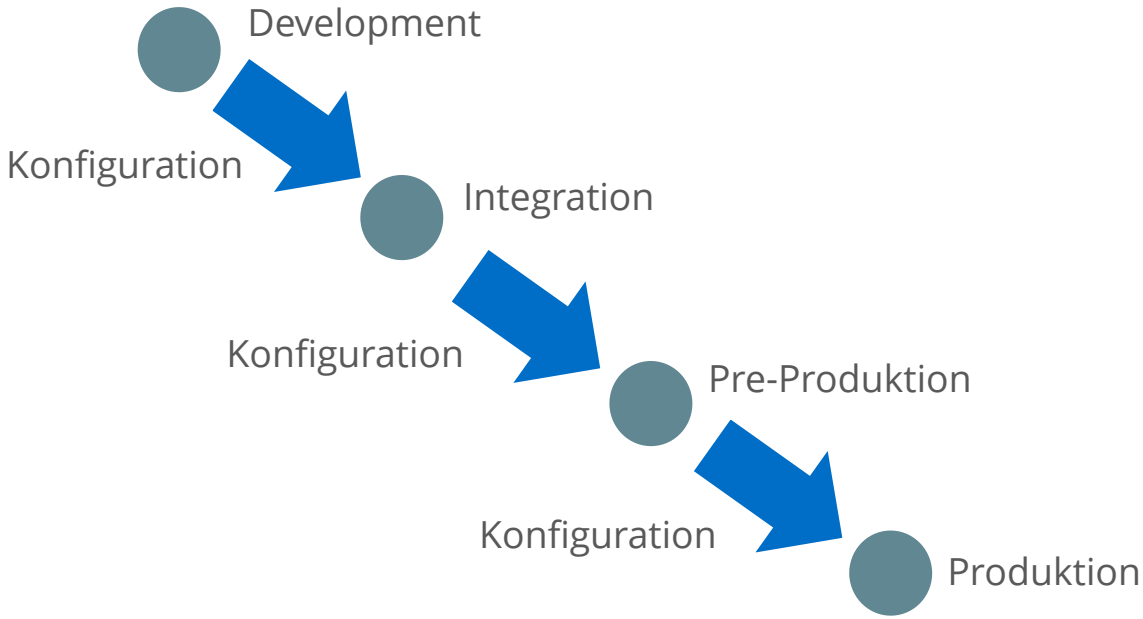


Sidecar
Beiwagen

Externe Konfiguration Problem

Problem

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



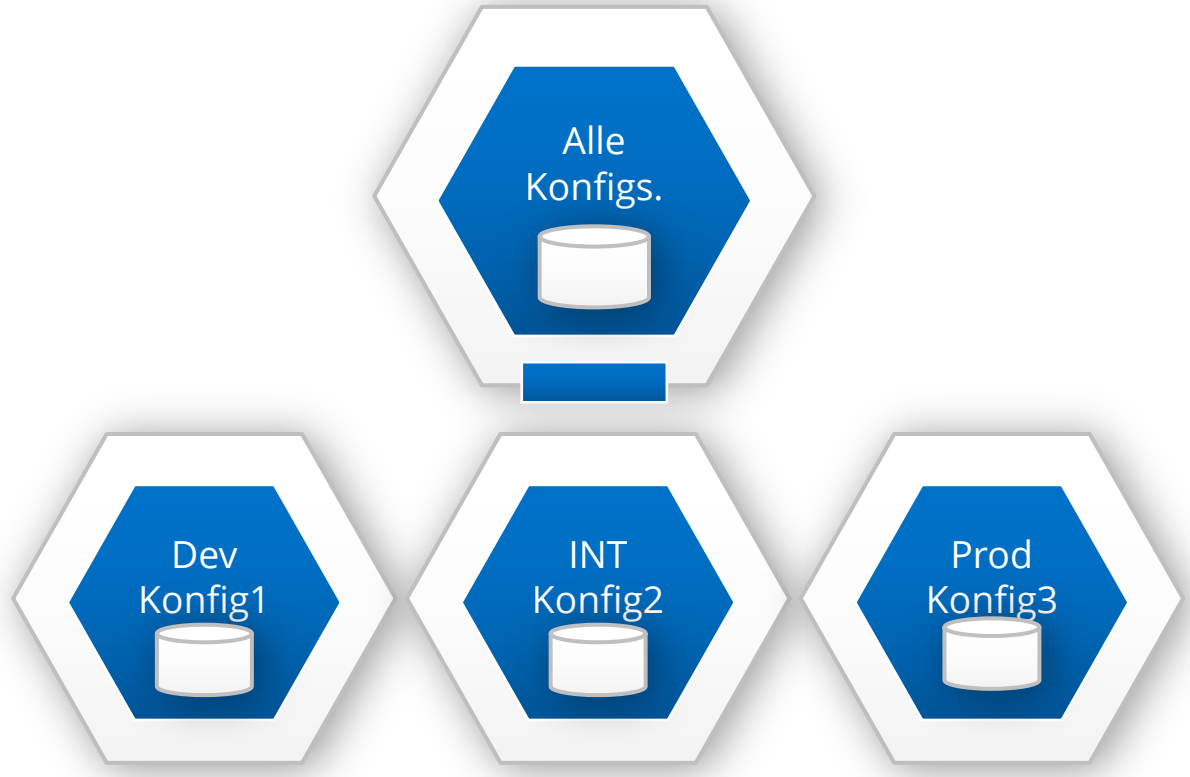
Richardson, C: Pattern: Externalized Configuration, https://microservices.io/patterns/externalized-configuration.html, abgerufen 12.4.2019
External configuration pattern, https://docs.microsoft.com/en-us/azure/architecture/patterns/external-configuration-store, abgerufen 12.4.2019

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
```

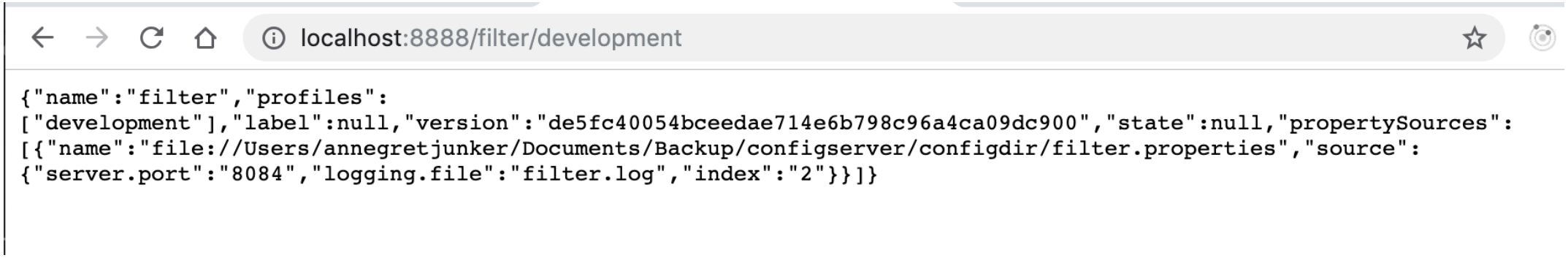
```
@EnableConfigServer
@SpringBootApplication
public class ConfigserverApplication {

    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
```

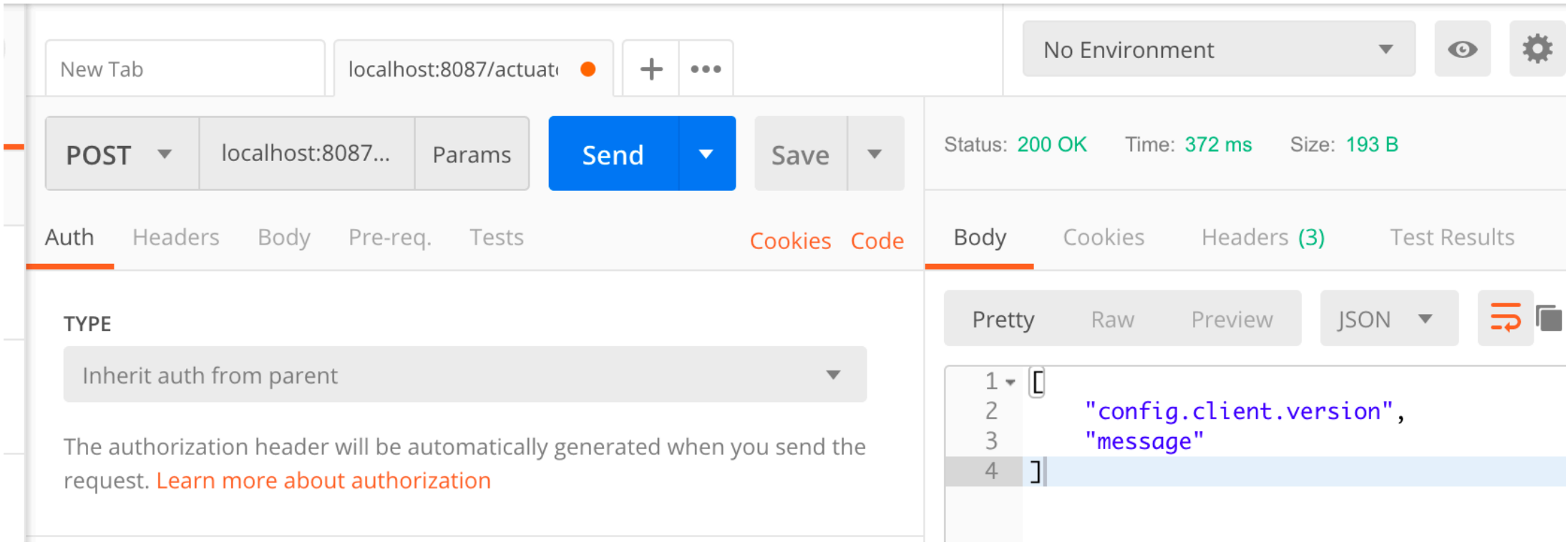
```
C02W51TPHTDD:~ annegretjunker$ cd Documents/Backup/con
figserver/configdir/
C02W51TPHTDD:configdir annegretjunker$ git commit filt
er.properties -mIndexChanged
[master de5fc40] IndexChanged
 1 file changed, 1 insertion(+), 1 deletion(-)
C02W51TPHTDD:configdir annegretjunker$
```

Externe Konfiguration Beispiel



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

Externe Konfiguration Beispiel



Beispiel Konfigurationsserver: https://github.com/Grinseteddy/CrossCuttingConcernsConfiguration
Beispiel Konfigurationsclient: https://github.com/Grinseteddy/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.propeties 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!

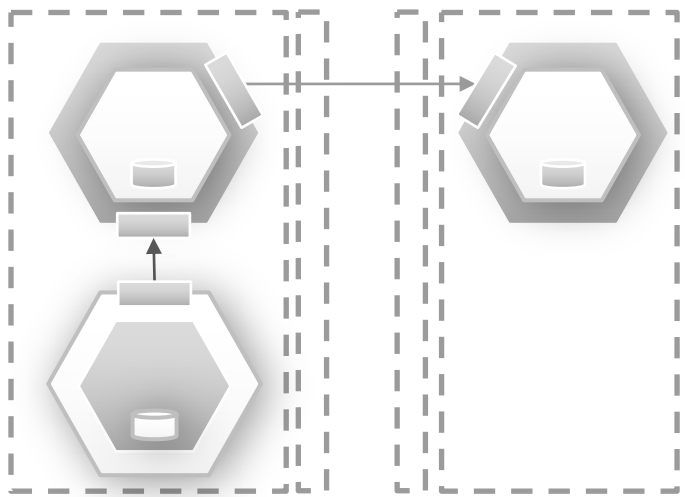
Beispiel Konfigurationsserver: https://github.com/Grinseteddy/CrossCuttingConcernsConfiguration
Beispiel Konfigurationsclient: https://github.com/Grinseteddy/CrossCuttingConcernsConfigClient

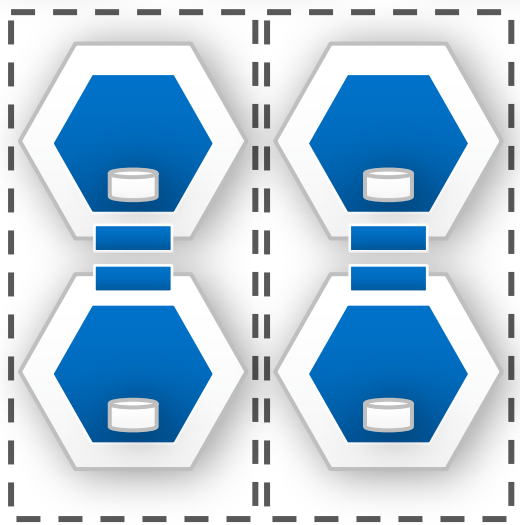
Pattern Sidecar

Microservice Chassis
Microservice Unterbau

Externe Konfiguration

Ambassador
Botschafter



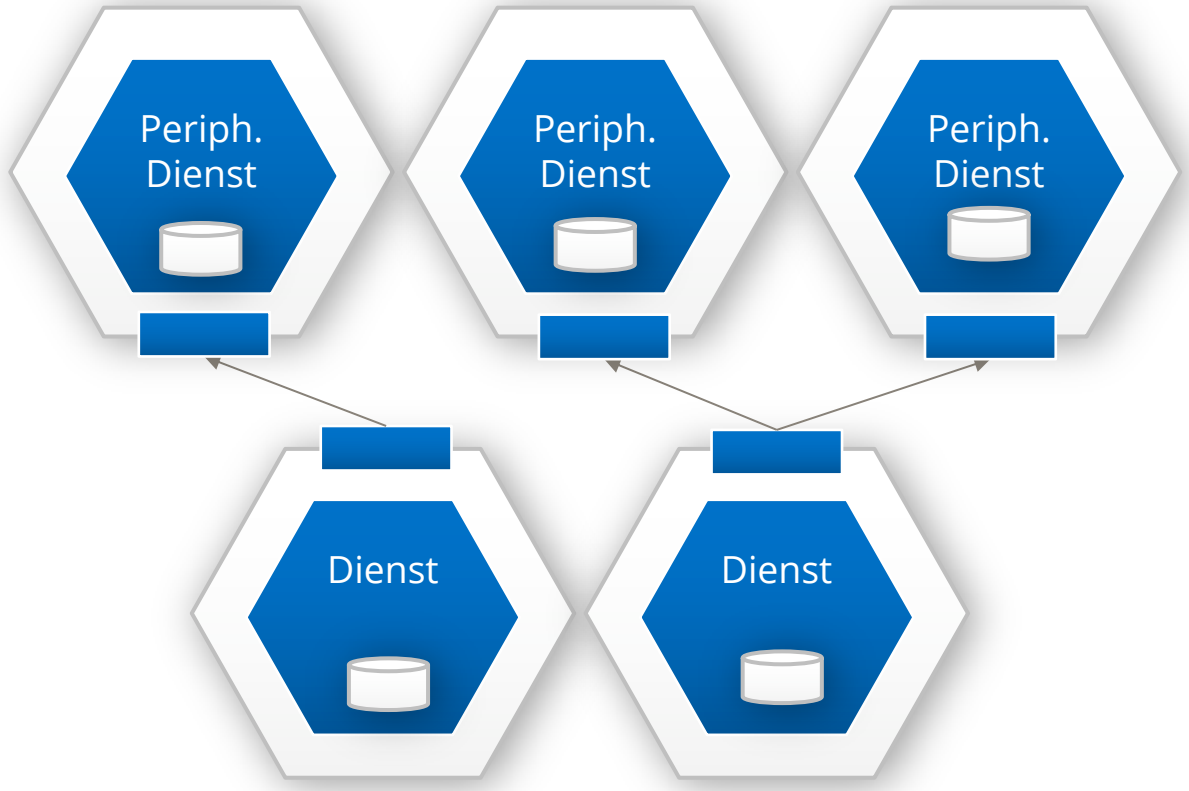


Sidecar
Beiwagen

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).

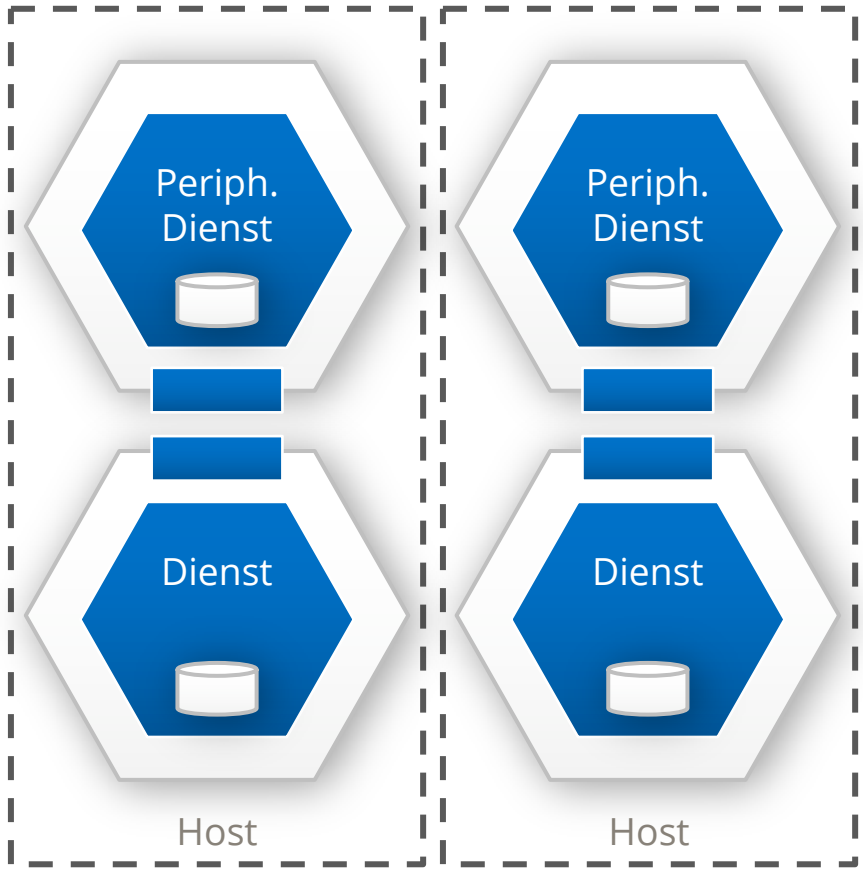


Sidecar pattern, https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar, abgerufen am 11.4.2019

Sidecar Lösung

Lösung

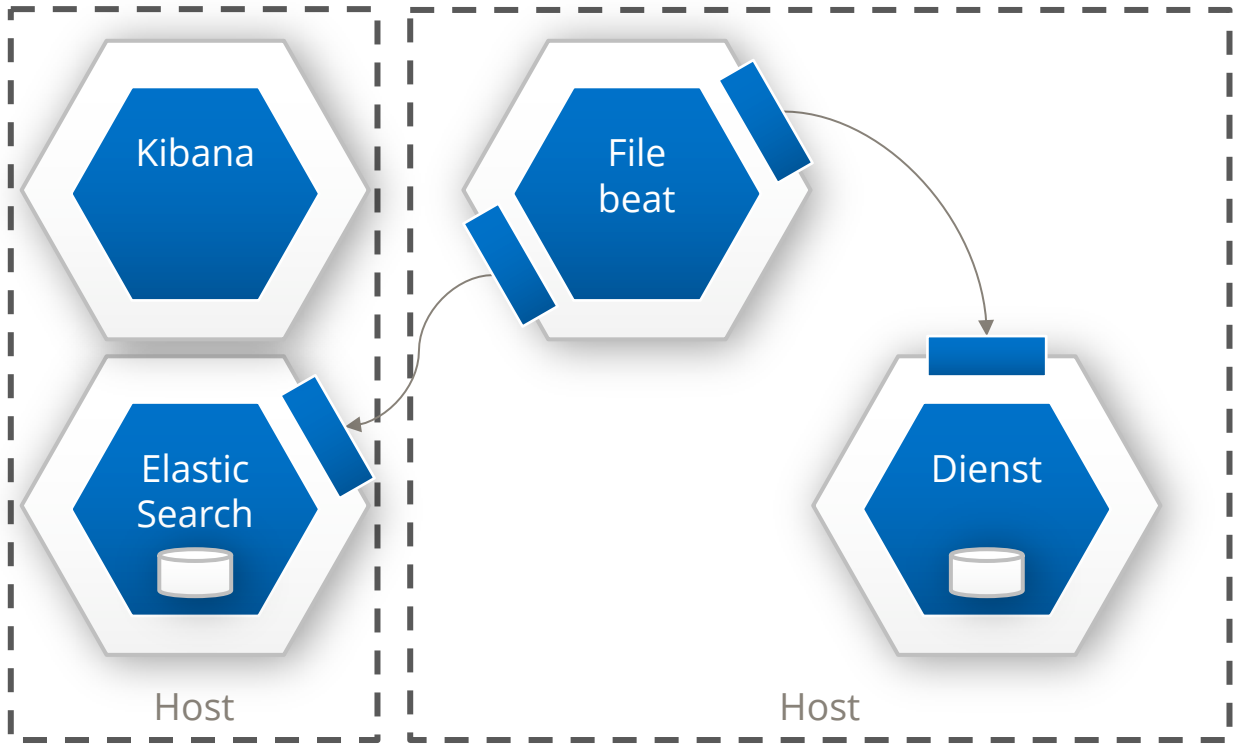
- Getrennte Aufgaben einer Anwendung auf einem gemeinsamen Host



Sidecar pattern, https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar, abgerufen am 11.4.2019

Sidecar Beispiel

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



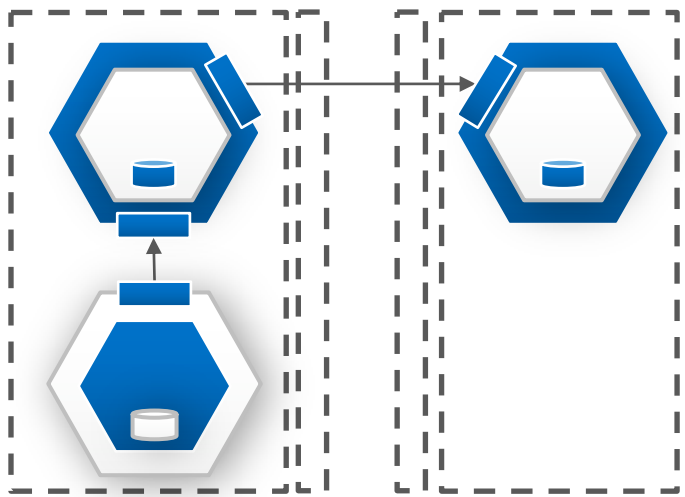
https://www.elastic.co/elk-stack

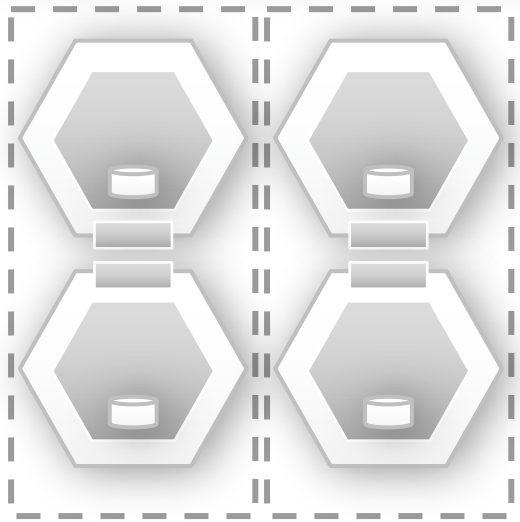
Pattern
Ambassador

Microservice Chassis
Microservice Unterbau

Externe Konfiguration

Ambassador
Botschafter



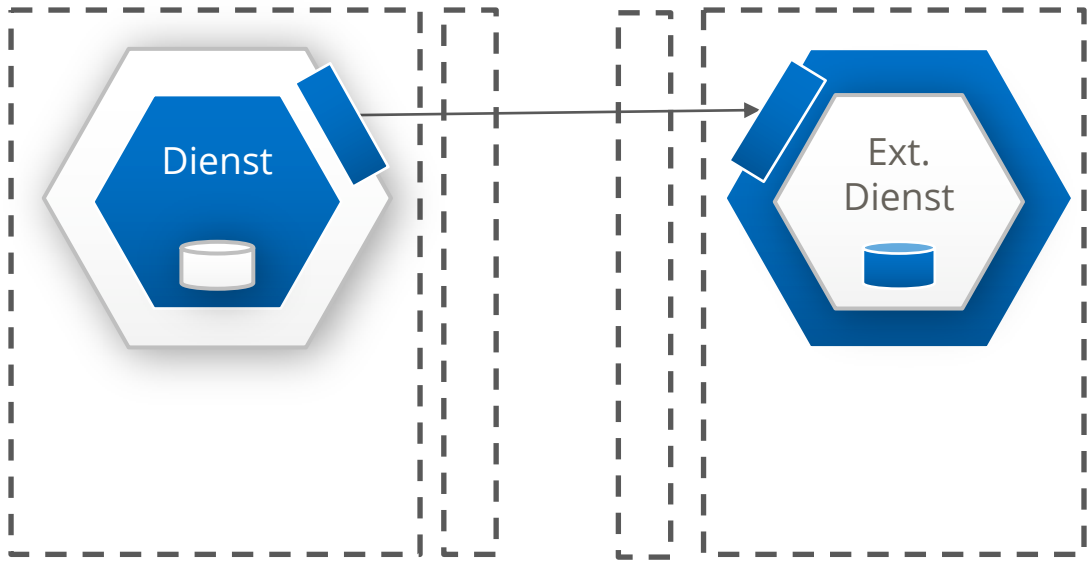


Sidecar
Beiwagen

Ambassador Problem

Problem

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

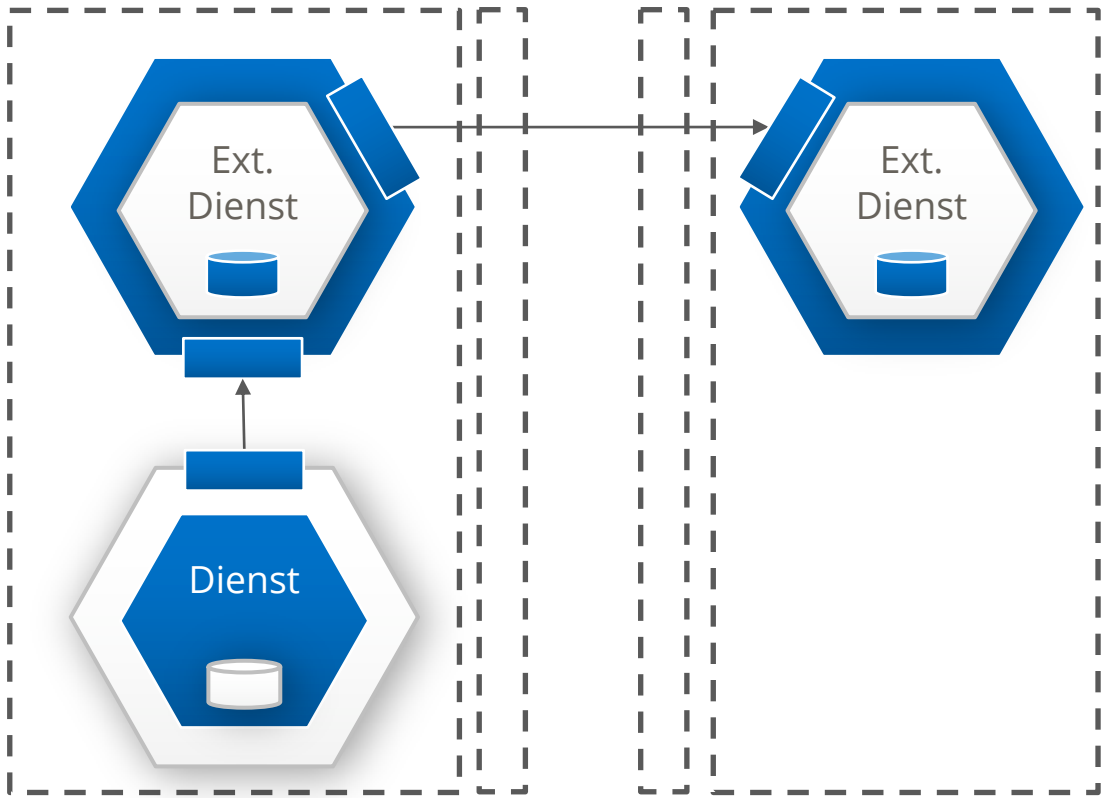


Ambassador Pattern, https://docs.microsoft.com/en-us/azure/architecture/patterns/ambassador, abgerufen 19.04.2019

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 Pattern, https://docs.microsoft.com/en-us/azure/architecture/patterns/ambassador, abgerufen 19.04.2019

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 {
```

https://github.com/Grinseteddy/CrossCuttingMasterDataClient

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

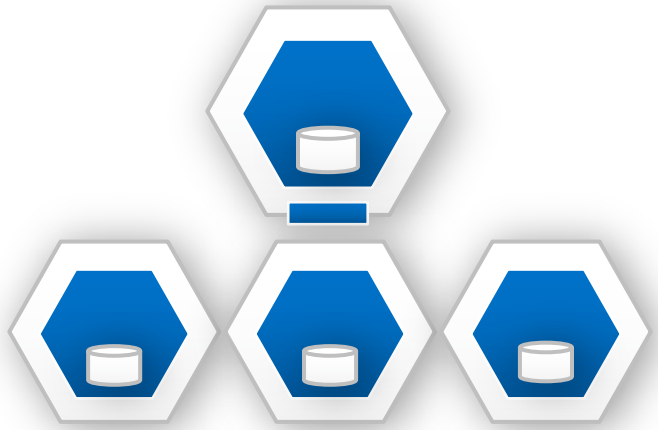
Pattern
4 gängige Lösungsmuster

Baue schnell, einfach und verlässlich Dienste

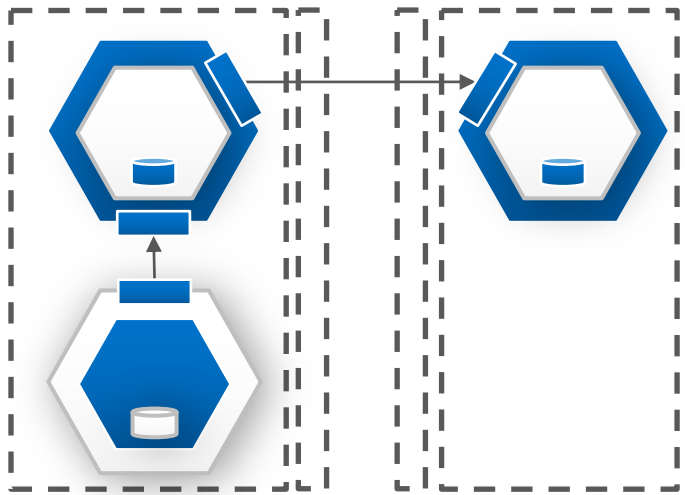
Microservice Chassis
Microservice Unterbau

Vermeide unnötigen Aufwand unterschiedlichen Umgebungen

Externe Konfiguration

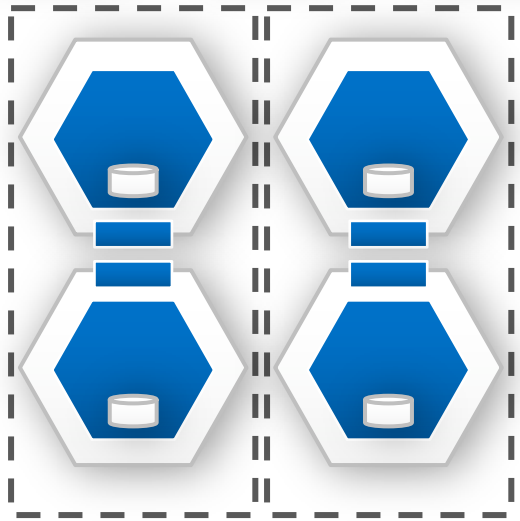


Ambassador
Botschafter



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