KAPITEL 6: SERVICE-INFRASTRUKTUR

LERNZIELE

- Anwendungszweck einer softwareseitigen Lösung für Discovery, Konfiguration und Gateway in Service Infrastrukturen erklären
- Grundaufbau einer Service-Infrastruktur darstellen und erklären
- Möglichkeiten zur Überwachung von Diensten in einer Infrastruktur aufzeigen und erklären
- Strategie zur Umsetzung einer Infrastruktur analysieren und Empfehlungen ableiten

6.1 MOTIVATION

Virtualisierung, Container und Orchestrierungsumgebungen stellen Werkzeuge zur Gestaltung einer Service-Landschaft: Konfigurationen, Netzwerke, Speicherbereiche, Anwendungsisolation und Load-Balancing

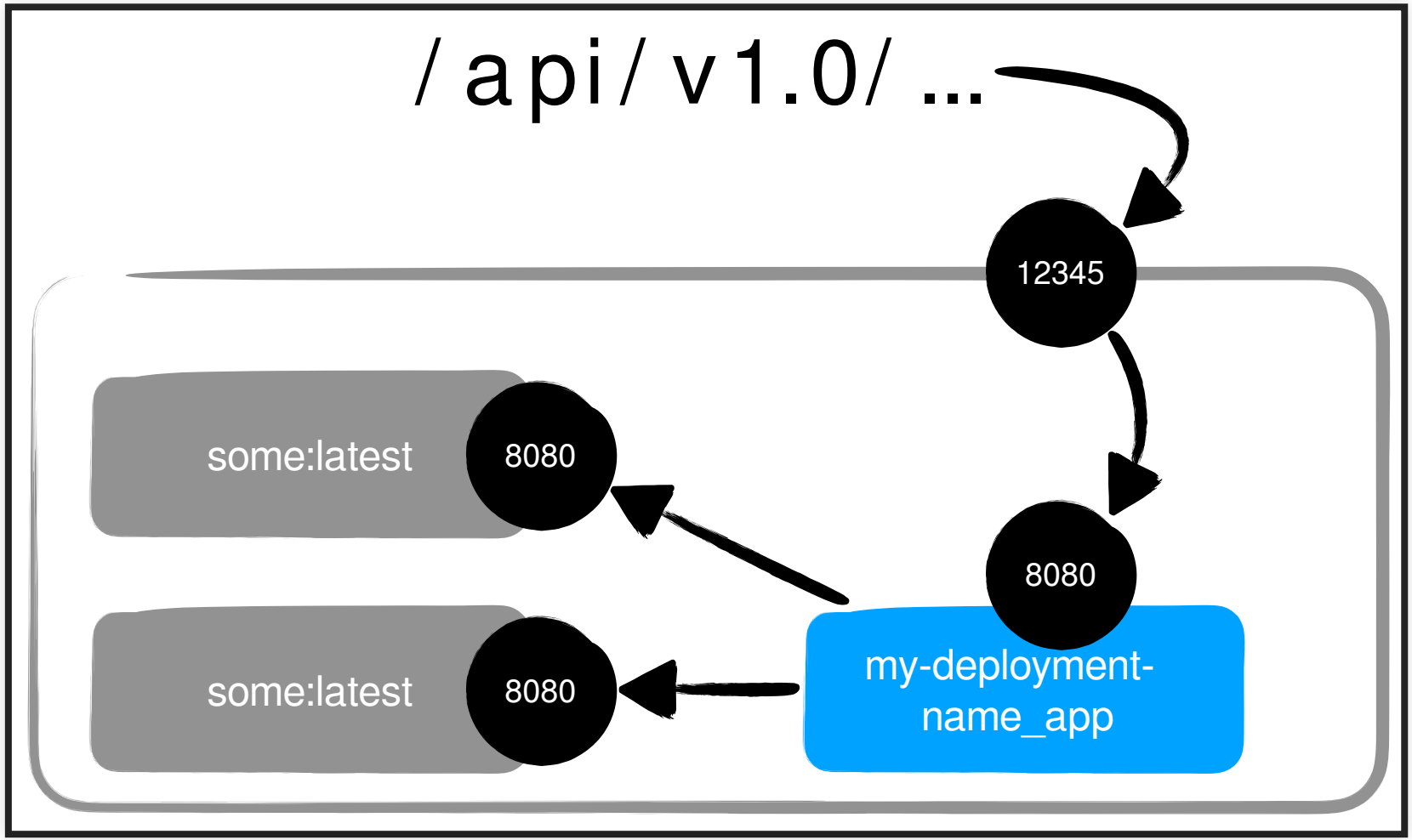
Neben diesen Werkzeugen die über die Ausführungsumgebungen bereitgestellt werden, gibt es Möglichkeiten zur Gestaltung der Infrastruktur mittels individueller Dienste.

Für eine Service-Infrastruktur stellt sich die Frage

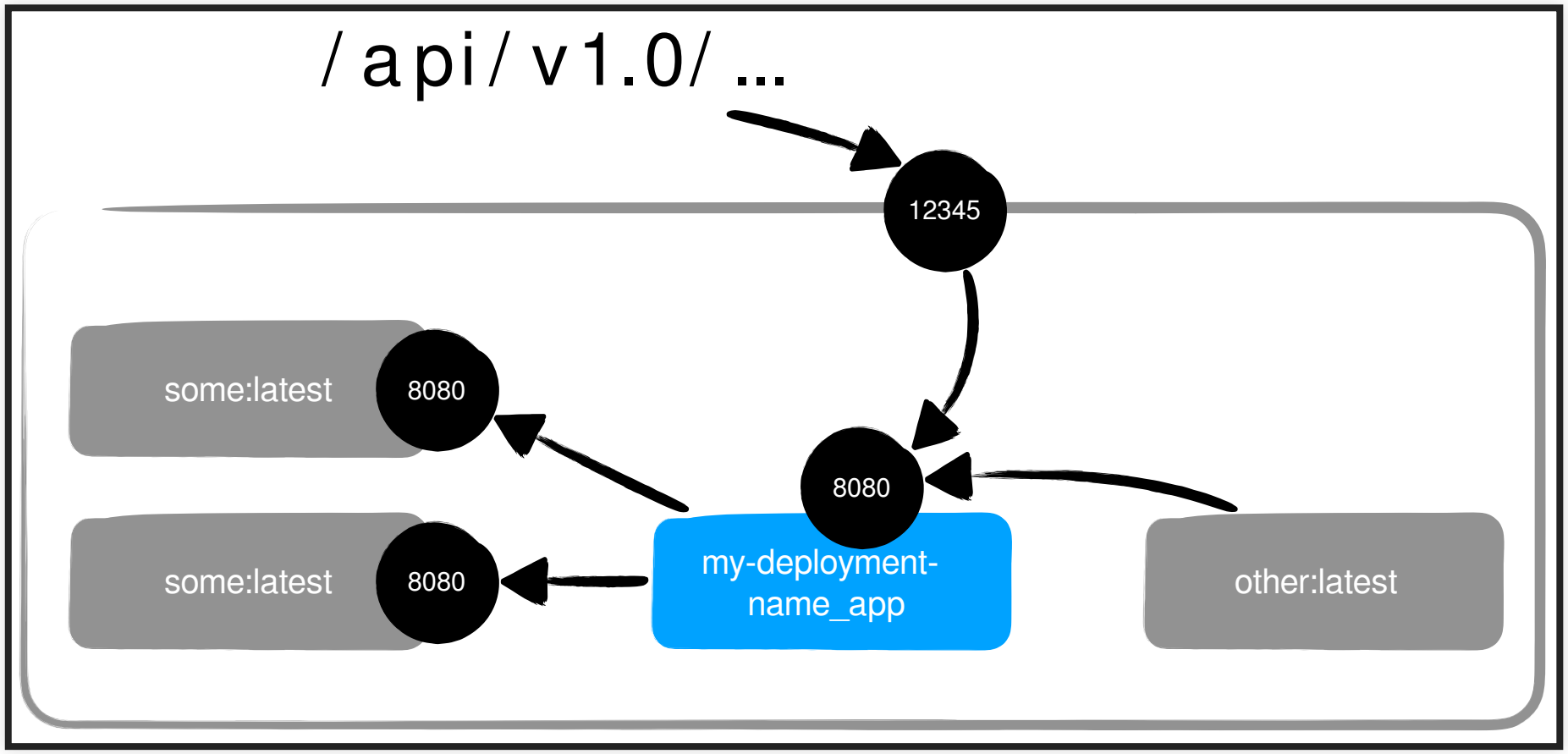
- Wie *erreichen* Services andere Services?
- Wo werden *Konfigurationen* für Services hinterlegt?
- Wie funktioniert der *Zugriff* durch Clientanwendungen auf Services?
- Was passiert mit den *Log-Ausgaben* der Services?
- Wie werden Services *überwacht*?

6.2 DISCOVERY

Beispiel-Dienst `some` konnte z.B. in einer Swarm-Konfiguration über den Docker-Dienst `my-development-name_app` adressiert werden

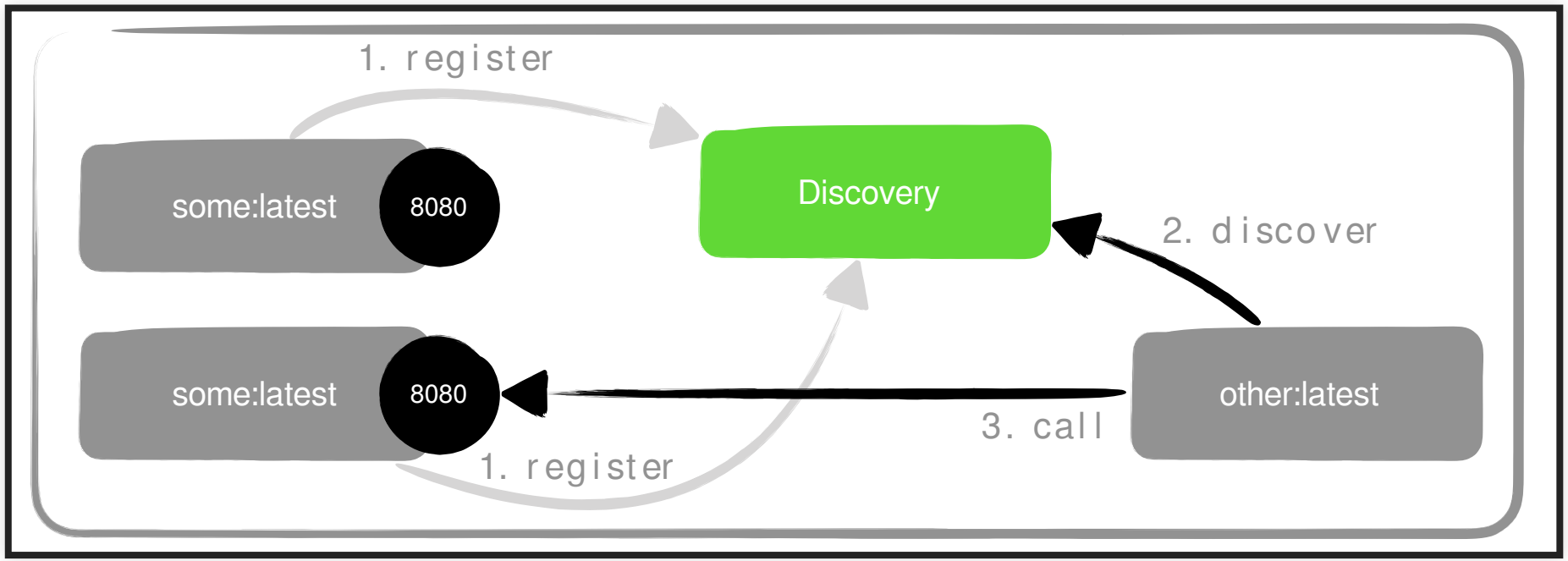


Ein andere Beispiel-Dienst `other` kann den Docker-Dienst (hat IP und Domain-Namen) innerhalb eines Overlay-Netzwerks über den Domain-Namen adressieren.



Alternative

Spezielle Discovery-Dienste als Software-Lösung der Architektur, welche die Aufgabe zur Adressierung übernehmen



1. Startet ein Dienst, registriert sich dieser, z.B. über einen Namen, und hinterlässt zusatzinformationen, damit er aufgefunden werden kann
2. Ein anderer Dienst kann über den Namen die notwendigen Daten abfragen und damit bekannte Instanzen erfahren.
3. Mit den Informationen über verfügbare Instanzen kann der andere Dienst die Instanzen anfragen, z.B. mit Fail-Over-Behandlung.

Vergleich

- Provided Discovery (z. B. Docker Services)
  - Funktioniert Out-of-the-Box
  - Deckt *95%* der Anwendungsfälle ab
  - Abhängig von Ausführungsumgebung (Konfiguration ggf. je nach Deployment notwendig)
- Application-based Discovery
  - Erfordert Entwicklungsarbeit
  - Erlaubt anwendungsspezifisches Load-Balancing
  - Unabhängig von Ausführungsumgebung (ist fester Bestandteil des Deployments)
  - Client-Side Load-Balancing notwendig

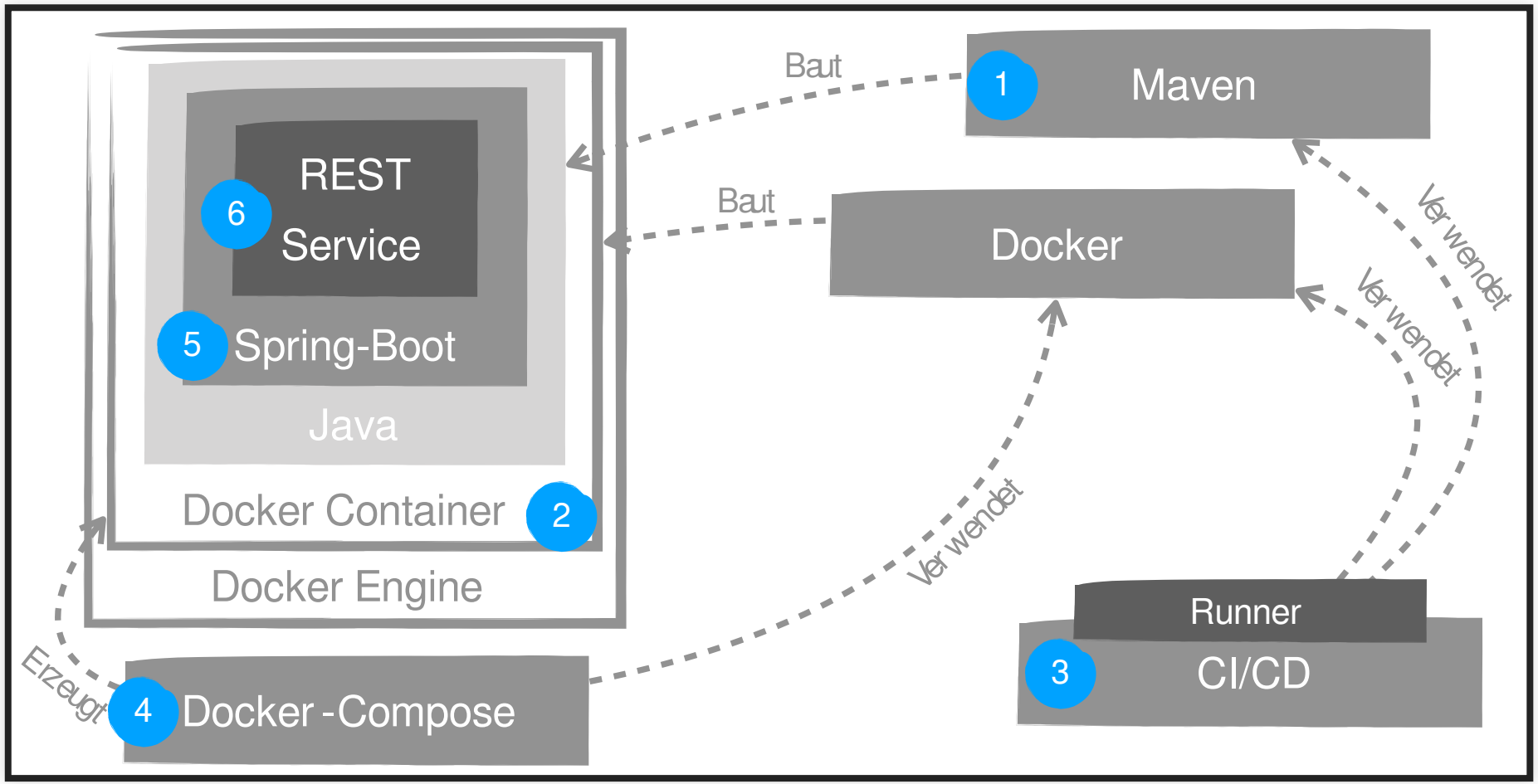
BEISPIEL: DISCOVERY

Im folgenden Paket sind zwei Projekte enthalten. Ein Service und ein Client, der gefunden werden möchte. Mehr zur Nutzung in der README.md.

Nutzen Sie das bereitgestellte Projekt-Archiv spring-eureka.zip, entpacken Sie dieses und importieren Sie den resultierenden Ordner es als "Existing Maven Project".

6.3 KONFIGURATION

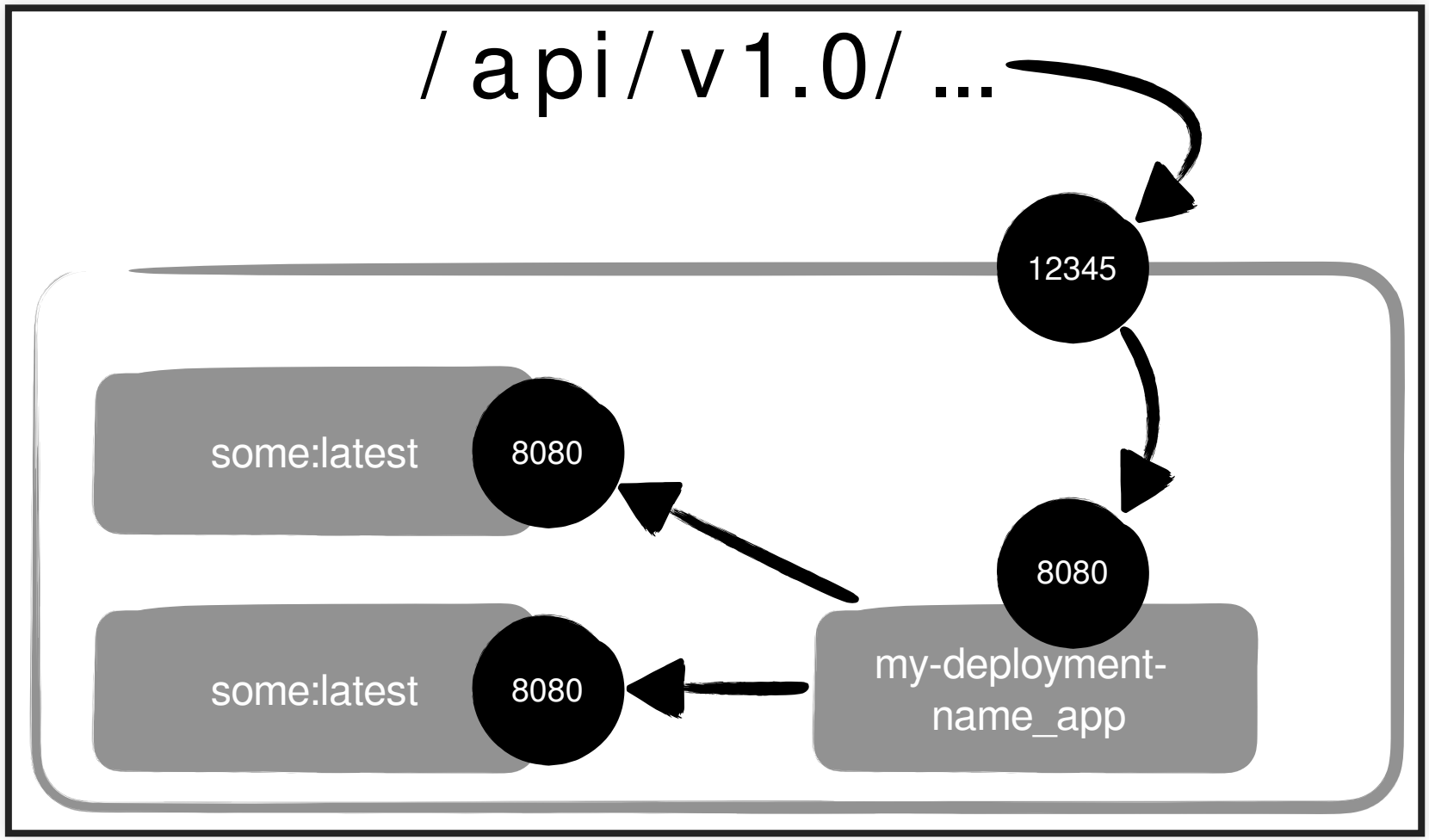
Es verbergen sich Anwendungsspezifische Konfigurationen an verschiedenen Stellen



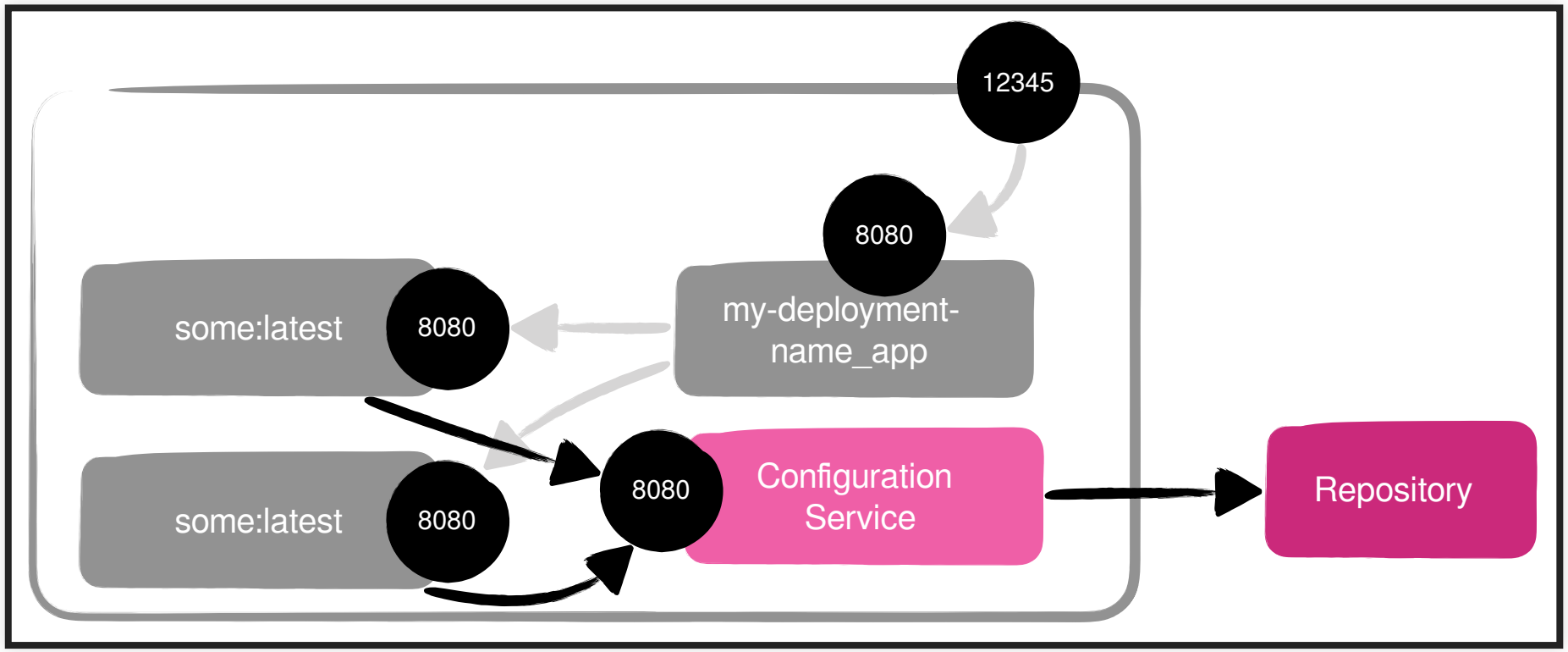
Konfigurationsebenen

- Dependency / Build Konfiguration
- Container Konfiguration
- Build / Deploy Prozess Konfiguration
- Environment Konfiguration (z. B. Netzwerk, Volumes, Port-Mapping)
- Infrastruktur Konfiguration (z. B. Endpunkt für bestimmte Dienste in Abhängigkeit)
- Applikation Konfiguration (z. B. Datenbankzugriff, spezifische Anwendungsparameter)

Rückblick, Beispiel-Dienst `some`: Jeder Dienst-Instanz müsste einzeln konfiguriert werden



Alternative: Ein Konfigurations-Dienst, den jede Instanz für seine eigene Konfiguration anfragt.



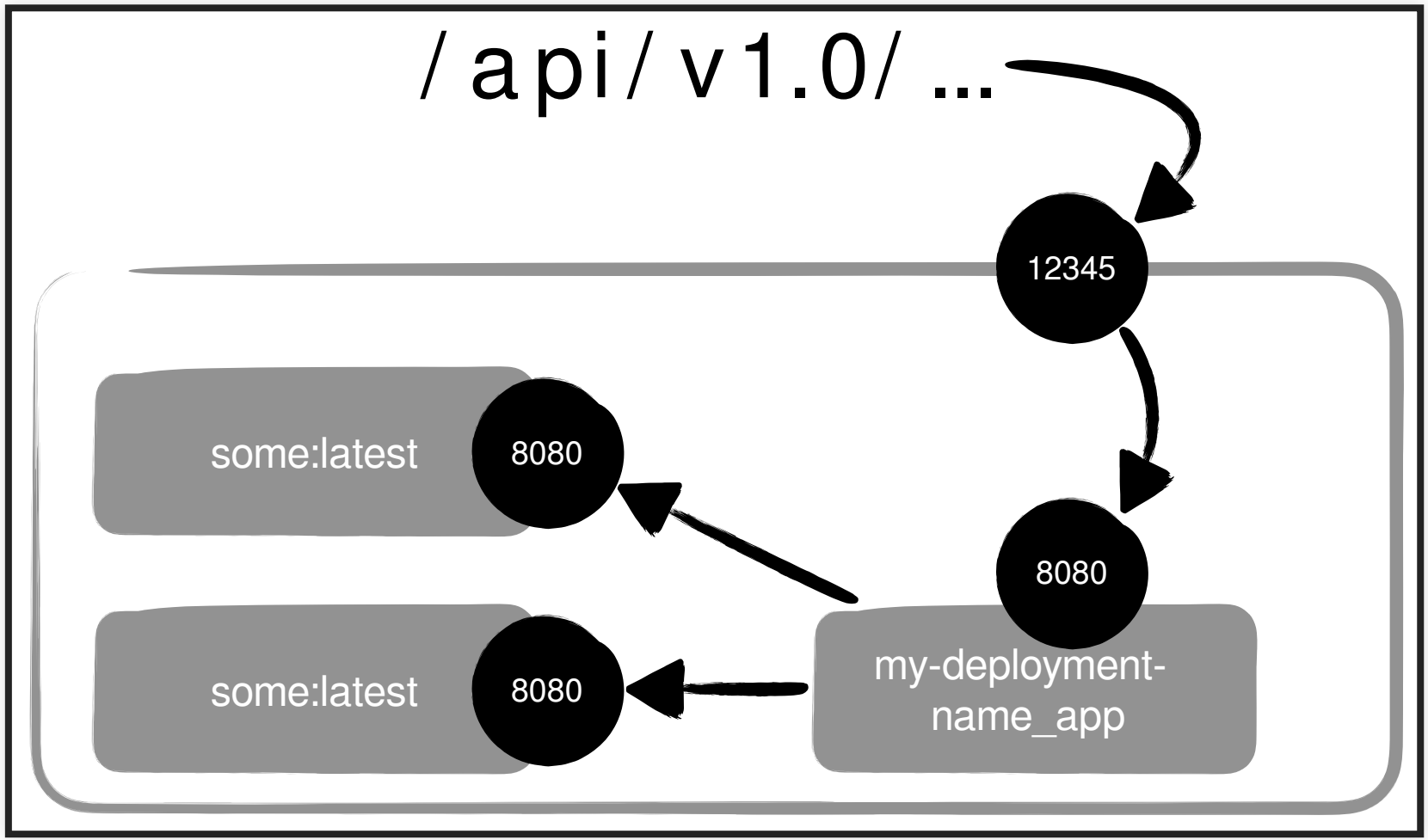
BEISPIEL: CONFIGURATION

Im folgenden Paket sind zwei Projekte enthalten. Ein Configuration Service und ein Client, der Einstellungen vom Service verwendet. Mehr zur Nutzung in der README.md.

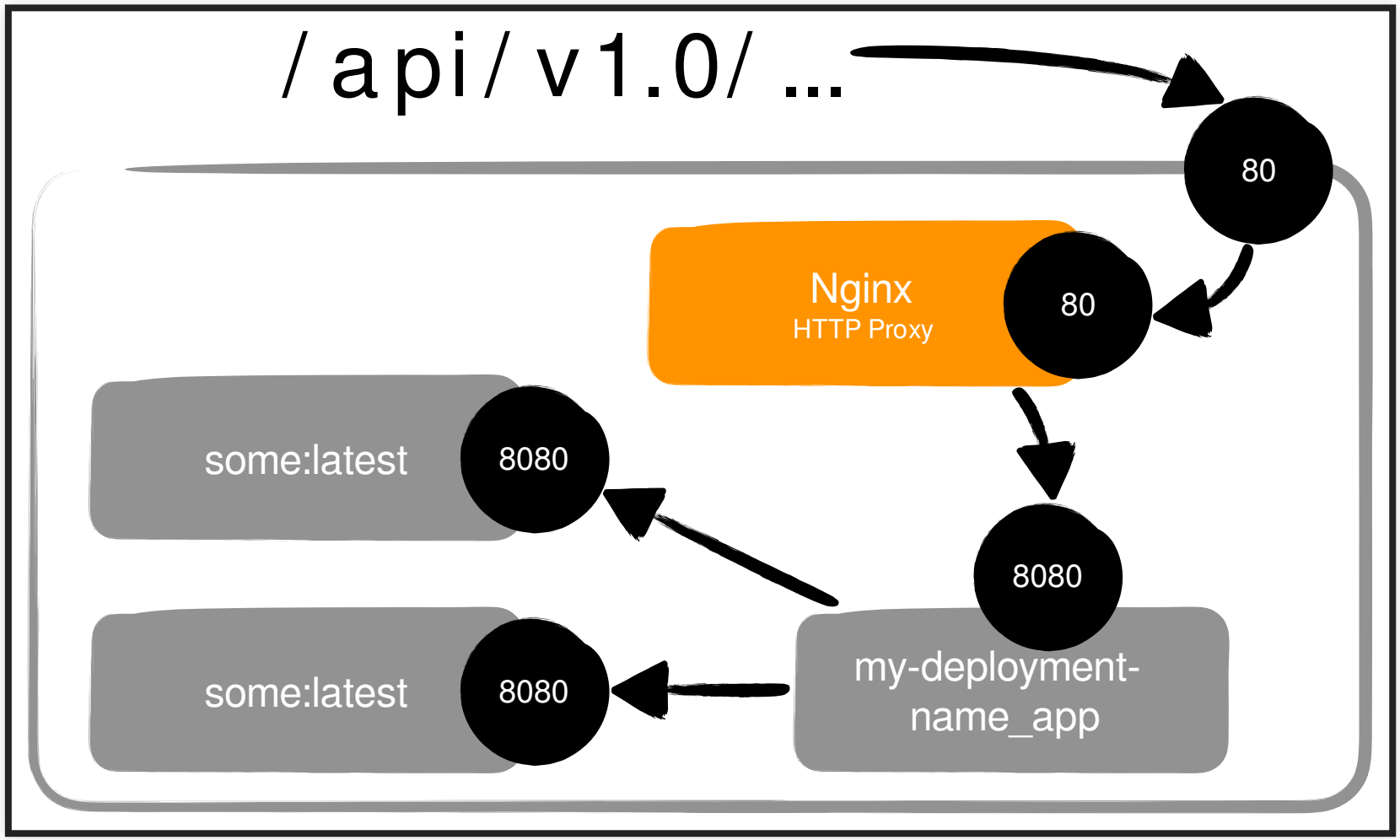
Nutzen Sie das bereitgestellte Projekt-Archiv spring-config.zip, entpacken Sie dieses und importieren Sie den resultierenden Ordner es als "Existing Maven Project".

6.4 GATEWAY

Rückblick, Beispiel-Dienst `some`: Die Endpunkte im Dienst könnte über den Host und den veröffentlichten Ports erreicht werden



Anstatt individueller Ports bietet sich die Möglichkeit zur Nutzung eines HTTP-Proxy für die Kommunikation an. Dieser kann dann Anfragen verteilt. Er muss jedoch (i.R.) manuell konfiguriert werden.

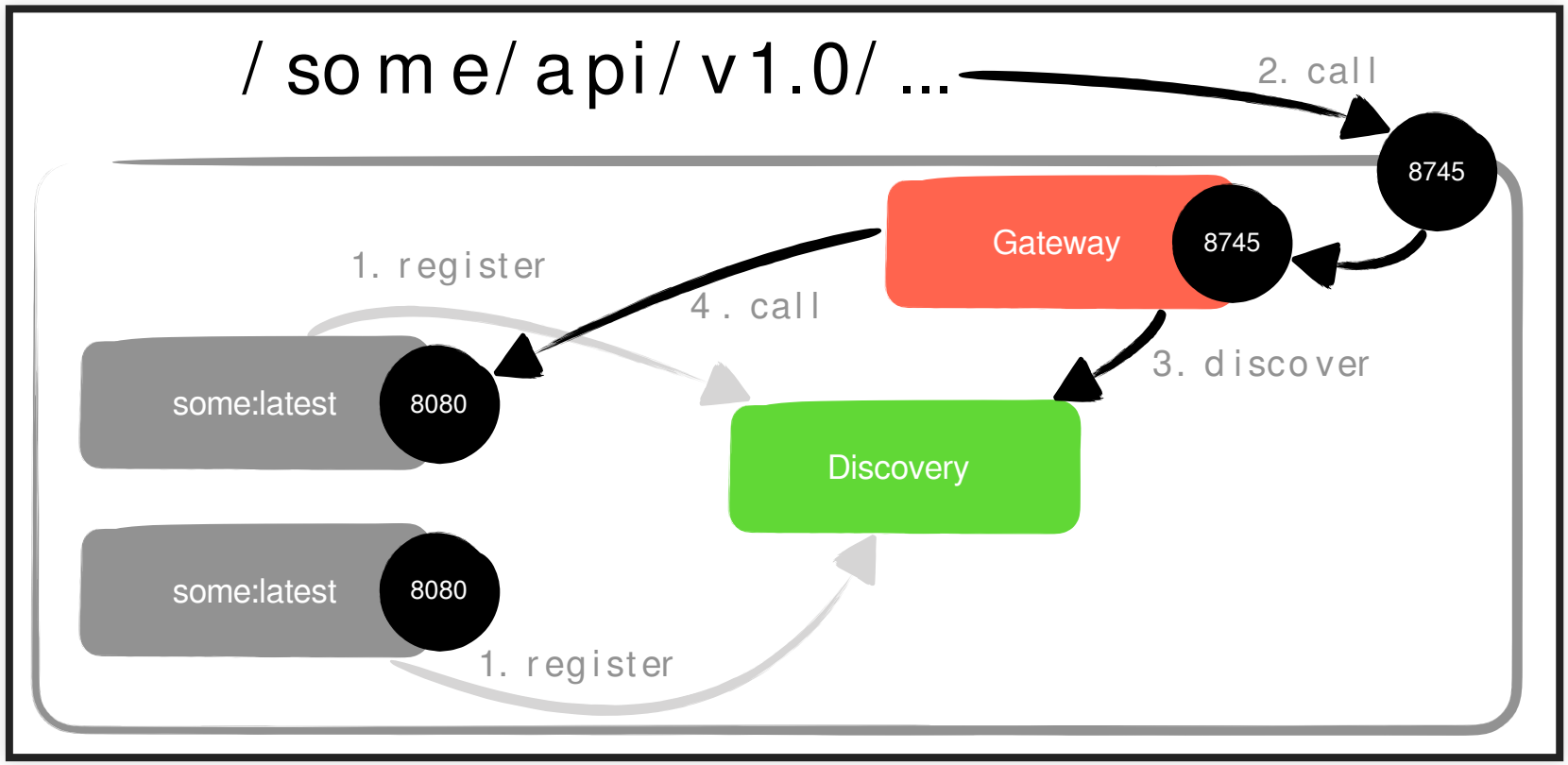


Beispiel Nginx-Konfiguration

```
server { # simple reverse-proxy
    listen 80;
    server_name localhost;
    location /api/ {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_pass http://some:8080;
    }
}
```

- Bereitstellung eines HTTP-Endpunktes über `localhost:80`
- Alle Anfragen mit dem Pfad-Prefix `/api/` speziell behandeln
- Anfragen weiterleiten an einen internen Dienst mit dem Domainnamen `some`

Alternativ bietet sich der Einsatz von Gateways an, insbesondere jene, die mit Discovery-Diensten zusammenarbeiten und damit automatisiert Anfragen gegen Dienste auflösen kann (nach festzulegenden Mustern).



1. Der Dienst registriert sich beim Start am Discovery-Dienst (vgl. Discovery)
2. Eine Anfrage vom Client wird an einem Gateway-Dienst entgegen genommen, zum Beispiel `/some/api/v1.0/...`
3. Der Gateway-Dienst nutzt einen Bestandteil der angefragten Resourcen-URL, zum Beispiel einen Pfad-Prefix, um damit den Discovery-Dienst nach verfügbaren Dienst-Instanzen zu dem Namen zu finden. Im Beispiel könnte dies `some` sein
4. Mit den gefundenen Instanzen führt der Gateway-Dienst die eigentliche Anfrage aus, modifiziert hier die URL, im Beispiel auf die ursprüngliche `/api/v1.0` und gibt das Ergebnis an den Client zurück

HTTP Proxy

- Zentraler Proxy zur Steuerung von HTTP-Traffic
- Behandlung von SSL/TLS, ggf. unverschlüsselte Kommunikation im internen Netzwerk

Service Gateway

- Arbeitet mit Application-based Service Discovery zusammen
- Löst unter einer einheitlichen URL-Struktur Anfragen gegen Dienste im internen Service-Netzwerk auf
- Schließt einen zusätzlichen HTTP-Proxy nicht aus, ist in der Regel empfehlenswert

BEISPIEL: GATEWAY

Im folgenden Paket sind vier Projekte enthalten. Ein Discovery Service, zwei Clients und ein Gateway. Mehr zur Nutzung in der README.md.

Nutzen Sie das bereitgestellte Projekt-Archiv spring-gateway.zip, entpacken Sie dieses und importieren Sie den resultierenden Ordner es als "Existing Maven Project".

6.5 LOGGING

Anwendungen liefern über Log-Ausgaben verschiedene Informationen

```
2018-06-28 12:18:37.510  INFO 38670 --- [           main] o.s.c.support.DefaultLifecycleProcessor  : Starting beans in phase 0
2018-06-28 12:18:37.514  INFO 38670 --- [           main] o.s.c.n.eureka.InstanceInfoFactory       : Setting initial instance status as: STARTING
2018-06-28 12:18:37.531  INFO 38670 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using JSON encoding codec LegacyJacksonJson
2018-06-28 12:18:37.531  INFO 38670 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using JSON decoding codec LegacyJacksonJson
2018-06-28 12:18:37.531  INFO 38670 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using XML encoding codec XStreamXml
2018-06-28 12:18:37.531  INFO 38670 --- [           main] c.n.d.provider.DiscoveryJerseyProvider   : Using XML decoding codec XStreamXml
2018-06-28 12:18:37.585  INFO 38670 --- [           main] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2018-06-28 12:18:37.586  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Disable delta property : false
2018-06-28 12:18:37.587  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Single vip registry refresh property : null
2018-06-28 12:18:37.588  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Force full registry fetch : false
2018-06-28 12:18:37.588  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Application is null : false
2018-06-28 12:18:37.588  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Registered Applications size is zero : true
2018-06-28 12:18:37.588  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Application version is -1: true
2018-06-28 12:18:37.588  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Getting all instance registry info from the eureka server
2018-06-28 12:18:37.594  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : The response status is 200
2018-06-28 12:18:37.595  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Starting heartbeat executor: renew interval is: 30
2018-06-28 12:18:37.617  INFO 38670 --- [           main] c.n.discovery.InstanceInfoReplicator     : InstanceInfoReplicator onDemand update allowed rate per min is 4
2018-06-28 12:18:37.620  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Discovery Client initialized at timestamp 1530181117620 with initial instances count: 1
2018-06-28 12:18:37.652  INFO 38670 --- [           main] c.n.e.EurekaDiscoveryClientConfiguration : Registering application actionitemlist-service with eureka with status UP
2018-06-28 12:18:37.654  INFO 38670 --- [           main] com.netflix.discovery.DiscoveryClient    : Saw local status change event StatusChangeEvent [timestamp=1530181117653, current=UP, previous=STA...]
2018-06-28 12:18:37.655  INFO 38670 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_ACTIONITEMLIST-SERVICE/10.232.27.163:actionitemlist-service: registering service...
2018-06-28 12:18:37.691  INFO 38670 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_ACTIONITEMLIST-SERVICE/10.232.27.163:actionitemlist-service - registration status: 204
2018-06-28 12:18:37.709  INFO 38670 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2018-06-28 12:18:37.710  INFO 38670 --- [           main] c.n.e.EurekaDiscoveryClientConfiguration : Updating port to 8080
2018-06-28 12:18:37.710  INFO 38670 --- [           main] c.n.e.EurekaDiscoveryClientConfiguration : Updating port to 8080
2018-06-28 12:18:37.714  INFO 38670 --- [           main] d.f.i.s.d.A.ActionItemListApplication    : Started ActionItemListApplication in 9.242 seconds (JVM running for 12.524)
2018-06-28 12:19:54.031  INFO 38670 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring FrameworkServlet 'dispatcherServlet'
2018-06-28 12:19:54.031  INFO 38670 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization started
2018-06-28 12:19:54.059  INFO 38670 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization completed in 27 ms
```

Diese landen nicht selten als Konsolenausgabe eines Containers auf dem Host oder in Log-Dateien im Container.

Strategie 1: Log-Dateien

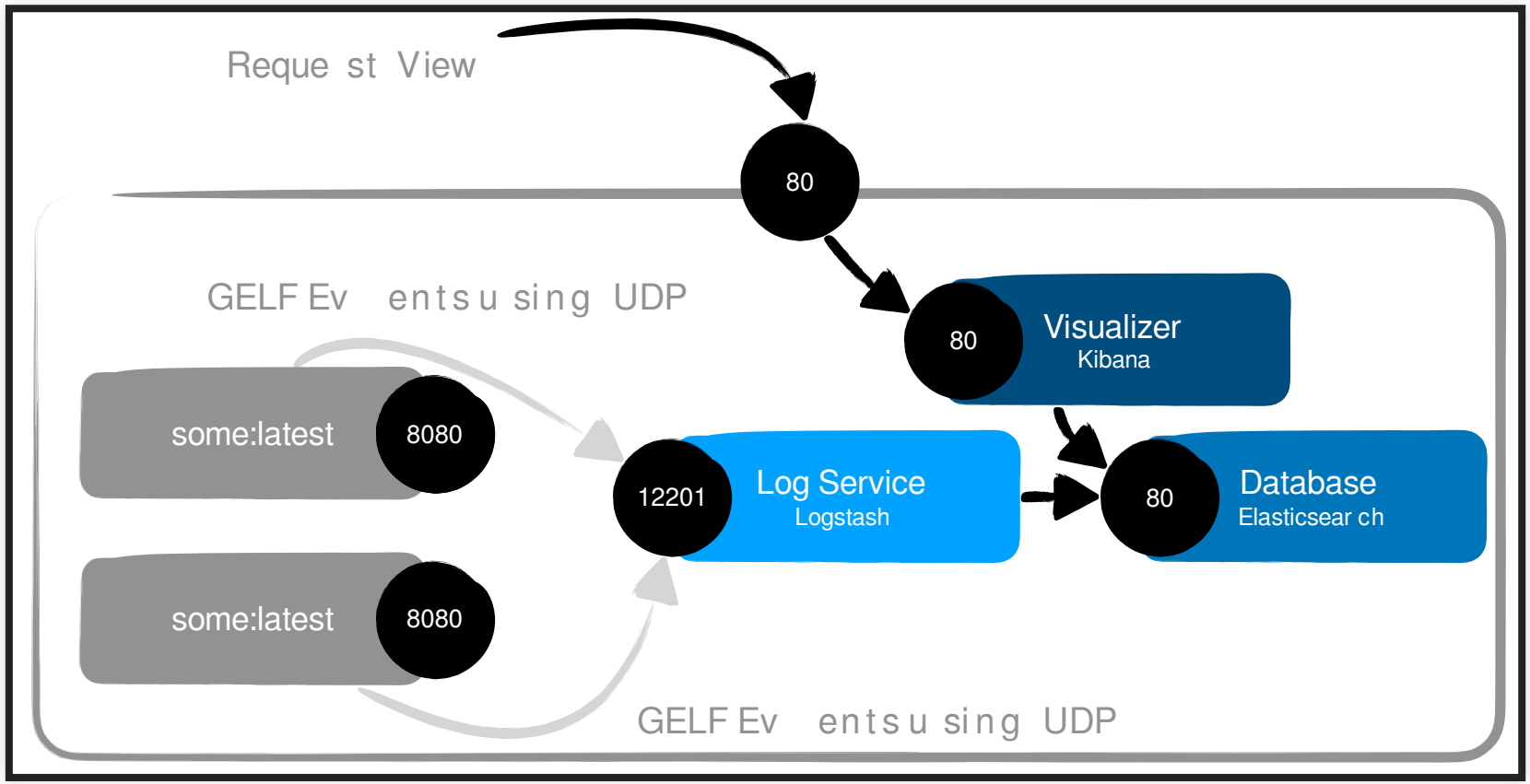
- Die Anwendung loggt alles in eine definierte Datei, z.B. konfiguriert über log4j
- Alternativ können mittels Mount-Points Log-Dateien aus dem Container bewegt werden; Log-Datei könnte anschließend in einem Netzwerk-Datei-System landen um ext. zugänglich gemacht zu werden

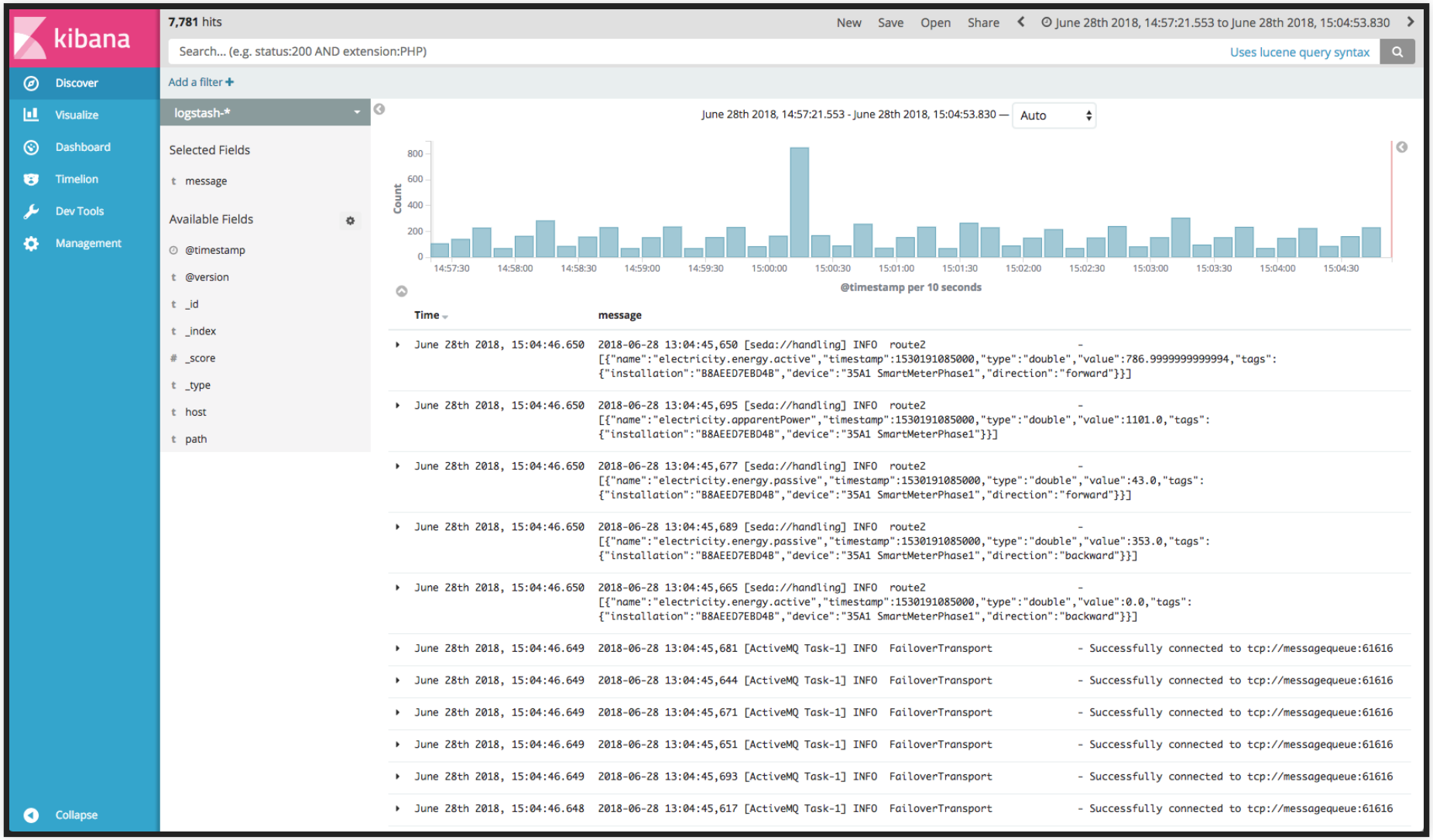
Strategie 2: StdOut

- Ausgaben werden in die Standardausgabe gesendet
- Was mit den Ausgaben passiert ist abhängig von der Ausführungsumgebung
- Im Fall von Docker werden alle Standardausgaben je Container in einer Log-Datei verwaltet
- Anzeige der Logs mittels `docker logs <containerName>` möglich
- Zugriff von ext. über Webfrontends wie Portainer.io möglich

Strategie 3: Log-Service

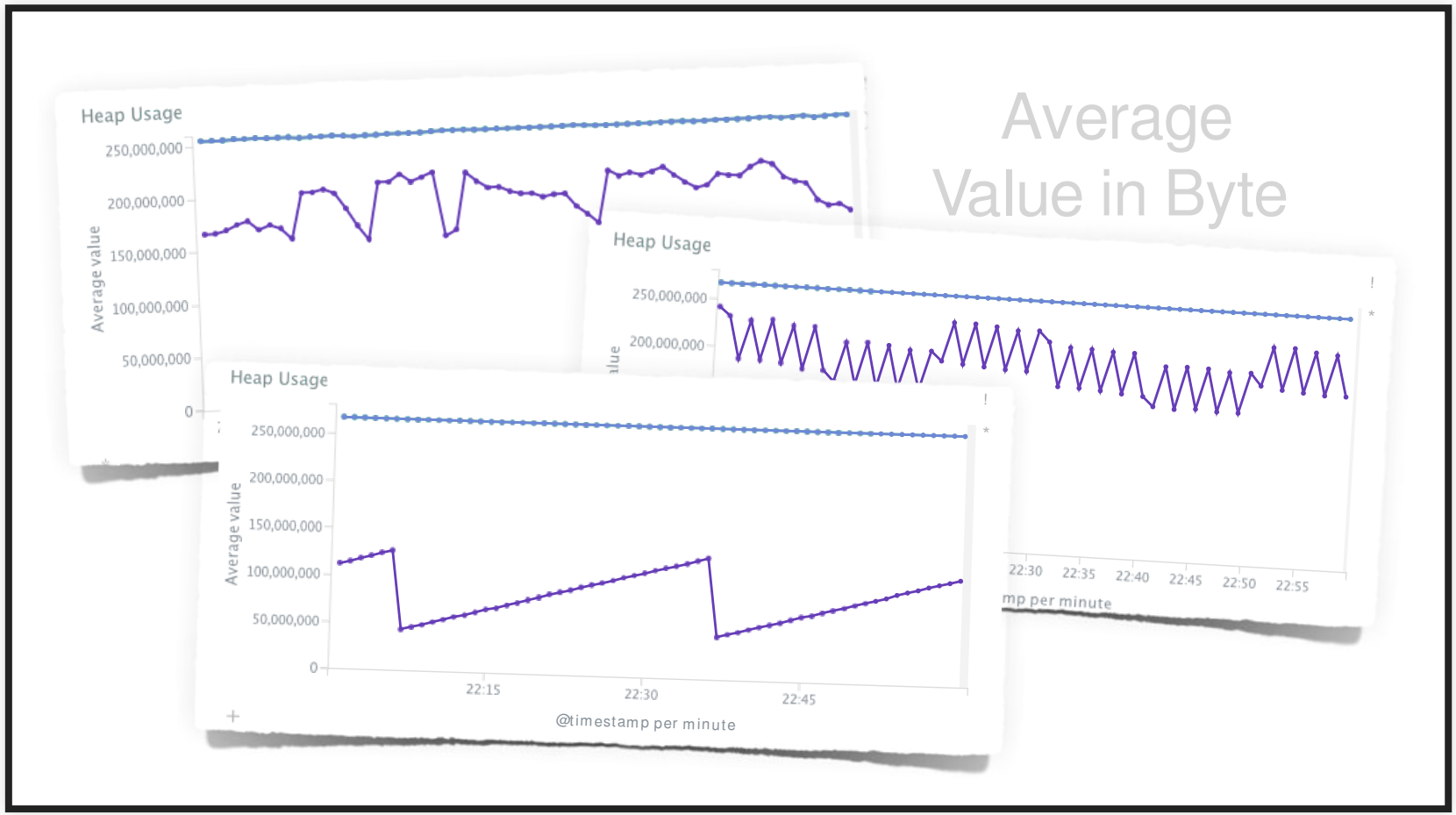
- Service zum Erfassen von Log-Ereignissen
- Speichert Ereignisse in einer Datenbank für spätere Auswertung
- Kann Dateien mitlesen und Ereignisse an Datenbank übergeben
- Kann Schnittstelle bereitstellen, an die Anwendungen ihre Log-Ereignisse senden kann
- Docker unterstützt hier z. B. Graylog Extended Log Format (GELF)


Reque st View
80
GELF Ev entsu sing UDP
80
Visualizer
Kibana
some:latest
8080
12201
Log Service
Logstash
80
Database
Elasticsear ch
some:latest
8080
GELF Ev entsu sing UDP


kibana
7,781 hits
New
Save
Open
Share
June 28th 2018, 14:57:21.553 to June 28th 2018, 15:04:53.830
Search... (e.g. status:200 AND extension:PHP)
Uses lucene query syntax
Add a filter +
Discover
Visualize
Dashboard
Timelion
Dev Tools
Management
Collapse
logstash-*
Selected Fields
t message
Available Fields
@timestamp
t @version
t _id
t _index
_score
t _type
t host
t path
June 28th 2018, 14:57:21.553 - June 28th 2018, 15:04:53.830 —
Auto
Count
800
600
400
200
0
14:57:30
14:58:00
14:58:30
14:59:00
14:59:30
15:00:00
15:00:30
15:01:00
15:01:30
15:02:00
15:02:30
15:03:00
15:03:30
15:04:00
15:04:30
@timestamp per 10 seconds
Time
message
June 28th 2018, 15:04:46.650 2018-06-28 13:04:45,650 [seda://handling] INFO route2 - [{"name":"electricity.energy.active","timestamp":1530191085000,"type":"double","value":786.9999999999994,"tags":{"installation":"B8AEED7EBD4B","device":"35A1 SmartMeterPhase1","direction":"forward"}}]
June 28th 2018, 15:04:46.650 2018-06-28 13:04:45,695 [seda://handling] INFO route2 - [{"name":"electricity.apparentPower","timestamp":1530191085000,"type":"double","value":1101.0,"tags":{"installation":"B8AEED7EBD4B","device":"35A1 SmartMeterPhase1"}}]
June 28th 2018, 15:04:46.650 2018-06-28 13:04:45,677 [seda://handling] INFO route2 - [{"name":"electricity.energy.passive","timestamp":1530191085000,"type":"double","value":43.0,"tags":{"installation":"B8AEED7EBD4B","device":"35A1 SmartMeterPhase1","direction":"forward"}}]
June 28th 2018, 15:04:46.650 2018-06-28 13:04:45,689 [seda://handling] INFO route2 - [{"name":"electricity.energy.passive","timestamp":1530191085000,"type":"double","value":353.0,"tags":{"installation":"B8AEED7EBD4B","device":"35A1 SmartMeterPhase1","direction":"backward"}}]
June 28th 2018, 15:04:46.650 2018-06-28 13:04:45,665 [seda://handling] INFO route2 - [{"name":"electricity.energy.active","timestamp":1530191085000,"type":"double","value":0.0,"tags":{"installation":"B8AEED7EBD4B","device":"35A1 SmartMeterPhase1","direction":"backward"}}]
June 28th 2018, 15:04:46.649 2018-06-28 13:04:45,681 [ActiveMQ Task-1] INFO FailoverTransport - Successfully connected to tcp://messagequeue:61616
June 28th 2018, 15:04:46.649 2018-06-28 13:04:45,644 [ActiveMQ Task-1] INFO FailoverTransport - Successfully connected to tcp://messagequeue:61616
June 28th 2018, 15:04:46.649 2018-06-28 13:04:45,671 [ActiveMQ Task-1] INFO FailoverTransport - Successfully connected to tcp://messagequeue:61616
June 28th 2018, 15:04:46.649 2018-06-28 13:04:45,651 [ActiveMQ Task-1] INFO FailoverTransport - Successfully connected to tcp://messagequeue:61616
June 28th 2018, 15:04:46.649 2018-06-28 13:04:45,693 [ActiveMQ Task-1] INFO FailoverTransport - Successfully connected to tcp://messagequeue:61616
June 28th 2018, 15:04:46.648 2018-06-28 13:04:45,617 [ActiveMQ Task-1] INFO FailoverTransport - Successfully connected to tcp://messagequeue:61616

6.6 MONITORING

Neben Log-Daten laufen in Anwendungen auch verschiedene Kennzahlen auf



Zum Beispiel Kennzahlen zur Speicher- und CPU-Nutzung.

Diese Informationen lassen sich auf mehreren Ebenen identifizieren:

- Host-Informationen
- Container-Informationen
- Anwendungs-Informationen

Für skalierbare Anwendungen sind diese Informationen wichtig. Deren Aggregation und Analyse kann genutzt werden, um die Skalierung des Systems selbst zu steuern als auch Anomalitäten zu identifizieren die auf Defekte hinweisen könnten.

Typische Probleme zur Laufzeit:

- Zu hohe Speicherlast (RAM)
- Festplattenspeicher voll
- CPU-Auslastung zu hoch

Innerhalb von Kubernetes können z.B. mittels Metric Server CPU und RAM von Pods überwacht werden und diese über einen Horizontal Pod Autoscaler genutzt werden.

Auf Betriebssystemebene und für die Überwachung von Containern gibt es verschiedene Lösungen (hier nicht im Fokus), z.B. Nagios.

Spannend sind darüber hinaus Ausführungsumgebungen wie für Java, die selbst eine weitere Virtualisierungsschicht mitbringen und erneut den Bedarf der Überwachung mitbringen.

Hierfür könnte die Anwendung im Container überwacht werden (vgl. Java-Werkzeuge) oder Bibliotheken zum Einsatz kommen, die während der Ausführung Informationen auslesen und zentral sammeln