THE LINUX FOUNDATION
SILVER MEMBER

kubernetes TRAINING PARTNER

kubernetes CERTIFIED SERVICE PROVIDER

Open Telemetry Introduction

August 12th, 2024

whizus

Agenda

- Traces, Logs und Metriken
- Grundlagen Open Telemetry
- Beispiel: Open Telemetry Komponenten
- Deployment-Patterns von Open Telemetry
- Deployment-Patterns im Vergleich
- Beispiel: Deployment-Patterns

Agenda

- Grundlagen Helm-Charts
- Open Telemetry Helm-Charts
- Beispiel: Open Telemetry Helm-Charts
- Grundlagen zu Ingress und Services in Kubernetes
- Best Practices zum Deployment von Open Telemetry
- Absicherung von Open Telemetry Komponenten

Was ist Tracing?

Metriken sind messbare Daten die Informationen über das System geben (zB genutzte CPU, genutzter RAM, Tasks in Queue, …)

Logs sind eine Menge von Events die von der Applikation produziert werden (zB AccessLog, ErrorLog, …). Logs haben normalerweise einen Timestamp.

Ein “**Trace**” ist der Pfad einer Anfrage durch die Applikation, aber was heißt das?

Was ist Tracing?

Jeder **Trace** hat eine eindeutige ID die eine Anfrage zum System darstellt

Jeder Trace besteht aus einer Menge von **Spans**, wobei jeder Span ein Event in der Applikation darstellt. Ein Span kann ein “root” Span sein oder ein Kind (in welchem Fall eine “parent_id” gesetzt ist).

Distributed Traces sind Traces die sich über mehrere Applikationen ziehen. Diese werden mittels “Context Propagation” zwischen den Applikationen übergeben.

Trace VS Log

Ein Span kann als strukturierter Log angesehen werden.

Spans sind Logzeilen mit klarer Struktur und Hierarchie.

Die hierarchische und verteilte Struktur von Traces sind der Mehrwert von Tracing.


```
{
  "name": "/v1/sys/health",
  "context": {
          "trace_id":
"7bba9f33312b3dbb8b2c2c62bb7abe2d",
          "span_id": "086e83747d0e381e"
  },
  "parent_id": "037e83746f0e331e",
  "start_time": "2021-10-22 16:04:01.209458162 +0000
UTC",
  "end_time": "2021-10-22 16:04:01.209514132 +0000
UTC",
  "status_code": "STATUS_CODE_OK",
  "status_message": "",
  "attributes": [..],
  "events": [..]
}
```

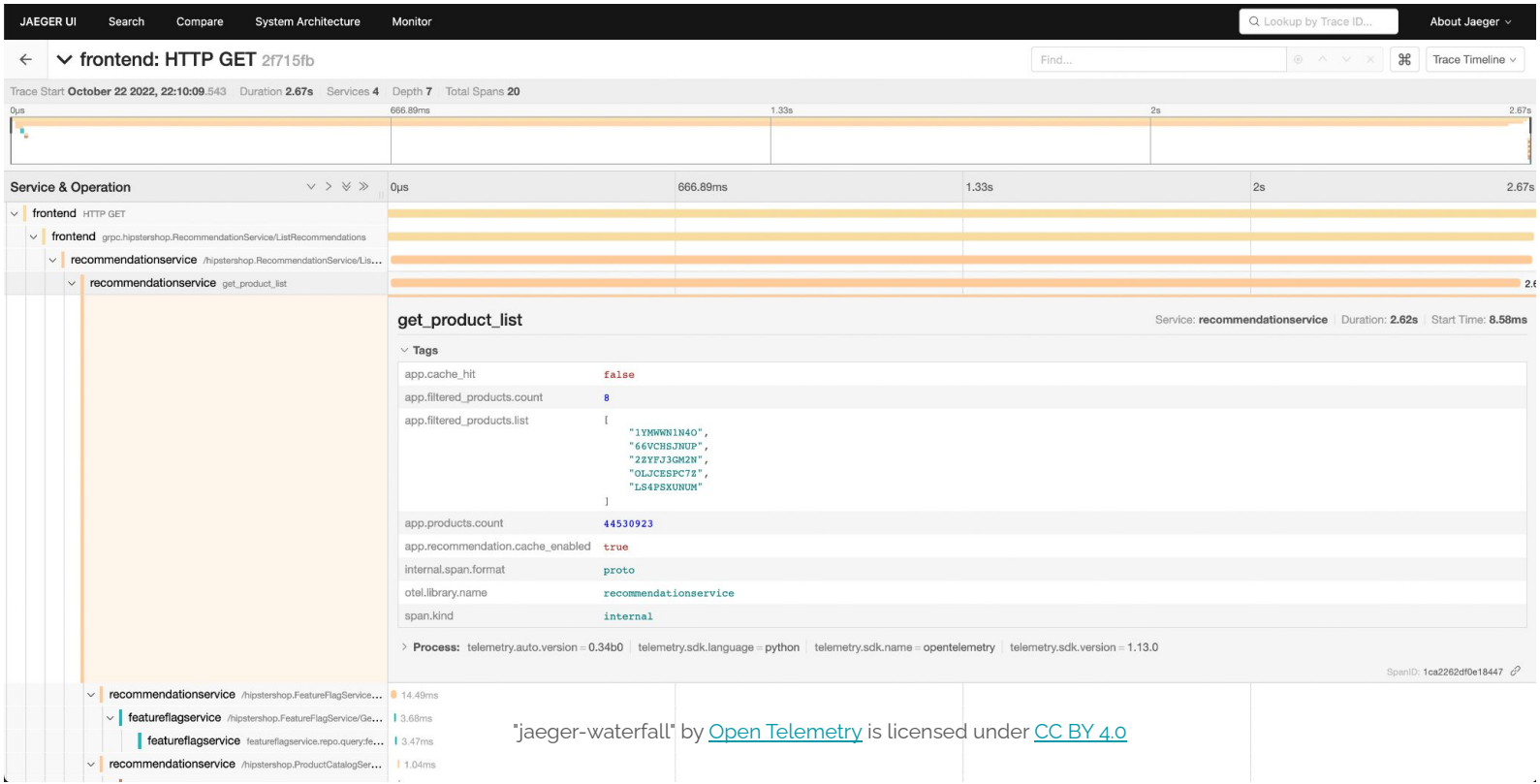
Was ist Tracing?

Da Spans strukturierte und hierarchische Logs sind, können diese durchaus auch als “normale Logs” angesehen/genutzt werden

Da Spans Attribute und Events haben können Metriken in diese verpackt werden und ebenfalls zu Traces/Spans hinzugefügt werden.

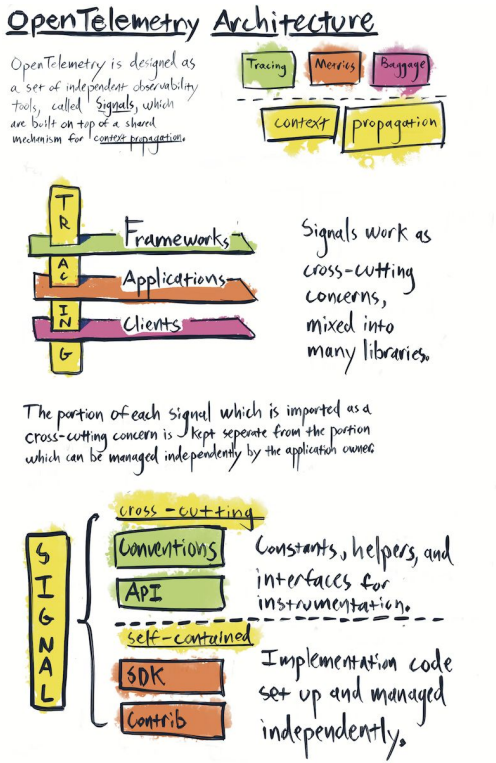
Automatische Instrumentalisierung ist ein weiterer Vorteil von Tracing (mehr Details hierzu später)

Was ist Tracing?



"jaeger-waterfall" by Open Telemetry is licensed under CC BY 4.0

Grundlagen Open Telemetry: Architektur



- **Signals**: Traces, Metrics, Context, Baggage, Propagation, …
- **Conventions, APIs, SDKs, Contrib Packages**
- **Frameworks, Applications, Clients:** Operator, Collector, SDKs

"architecture" by Open Telemetry is licensed under CC BY 4.0

Grundlagen Open Telem.: Komponenten

Traces und **Metrics** haben wir bereits erwähnt

Der “**Baggage**” ist eine Menge von Applikations-Properties die mittels dem Context verfügbar sind. Erlaubt es Traces mit “kontextueller Information” zu erweitern.

Der **Context** ist für die Verteilung der Trace-Informationen notwendig. Mittels diesem werden Informationen geteilt.

Context Propagation ist der Mechanismus mittels welchem der Context verteilt wird.

Grundlagen OTel: Context Propagation

Die folgenden Verbreitungs-Varianten werden von OTEL gewartet:
W3C TraceContext, W3C Baggage, B3, Jaeger

Die Varianten müssen von SDKs/Applikationen implementiert werden
Damit diese mit der Verteilung umgehen können.

Die Varianten nutzen zumeist Header um die Verteilung durchzuführen

https://opentelemetry.io/docs/specs/otel/context/api-propagators/#propagators-distribution

Grundlagen Open Telem.: Komponenten

Conventions und Specifications:
Open Telemetry Spec., Open Telemetry Protocol (OTLP), Open Agent Management Protocol (OpAMP), Semantic Conventions

Multiple **APIs** sind inder Open Telemetry Spec. definiert, zB.:
Context, Baggage, Tracing, Metrics, Logs

Es gibt Open Telemetry SDKs für alle großen Sprachen
(siehe auch die “Contrib” Repositories für die Sprachen)

Es gibt **manuelle** und **automatische Instrumentalisierung** SDKs!
Bei der automatischen Instrumentalisierung muss man keine Code Anpassungen machen.

Grundlagen Open Telem.: Komponenten

Es gibt Open Telemetry SDKs/Libraries für den Framework und Client-Support.

Es gibt den Stand-Alone Open Telemetry Collector zur Verarbeitung und Handhabung von Trace-Informationen

Es gibt den Kubernetes Operator und Helm-Charts für das Deployment von Collector Instanzen und zur Injektion von Auto-Instrumentalisierung.

Es gibt auch ein “Contrib” Repository für den Collector.
Viele Receiver, Exporter und Processor (wie Sampling) werden dort gewartet!

Grundlagen Open Telem.: Komponenten

Es gibt Open Telemetry SDKs/Libraries für den Framework und Client-Support.

Es gibt den Stand-Alone Open Telemetry Collector zur Verarbeitung und Handhabung von Trace-Informationen

Es gibt den Kubernetes Operator und Helm-Charts für das Deployment von Collector Instanzen und zur Injektion von Auto-Instrumentalisierung.

Grundlagen OTel: Collector

"otel-collector" by Open Telemetry is licensed under CC BY 4.0

Grundlagen OTel: Collector Konfiguration

Receivers: Konfiguration der Datenquellen des Collectors.
Es gibt Standard und “Contrib” Quellen

Exporters: Konfiguration der Ziele des Collectors
Es gibt Standard und “Contrib” Ziele

Processors: Modifikation und Transformation der Daten von Receivers bevor diese zu den Exporters weitergegeben wird

Connector: Erlaubt es zwei “Pipelines” miteinander zu verknüpfen (ist gleichzeitig ein Receiver und Exporter)

Grundlagen OTel: Collector Konfiguration

Service.Extensions: Erweiterte Funktionalität die aktiviert werden soll (zB HealthChecks)

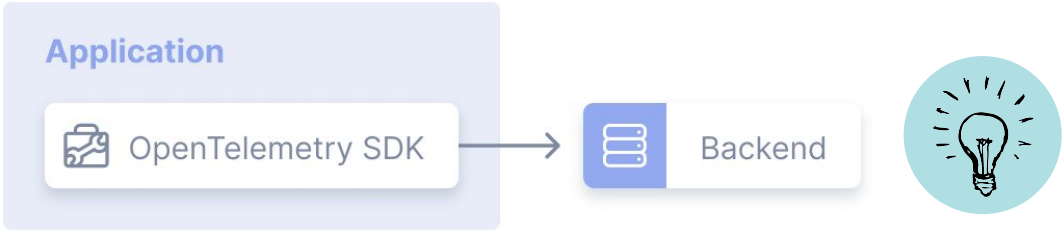
Service.Pipelines: Definiert den Flow von Daten durch den Collector. Receivers -> Processors -> Exporters

Service.Telemetry: Konfiguriere Metriken und Logs des Collector selbst.

Für Authentication bei Receivers und Exporters müssen diese mittels Extensions zuerst aktiviert werden!

Beispiel:
Open Telemetry Komponenten

Deployment-Patterns: NoCollector



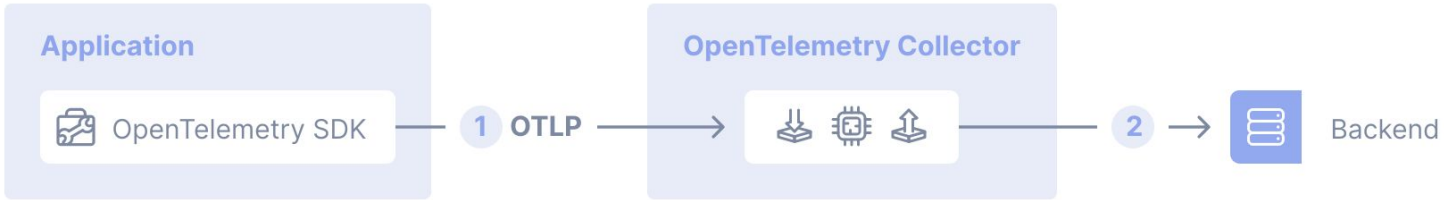
"otel-sdk" by Open Telemetry is licensed under CC BY 4.0

Applikation sollte SDK nutzen!

Es müssen keine zusätzlichen Komponenten genutzt werden

Jeder Service muss das Backend kennen und Daten schicken können.
Oft problematisch aufgrund von Authentication und Network-Requirements.

Deployment-Patterns: Agent

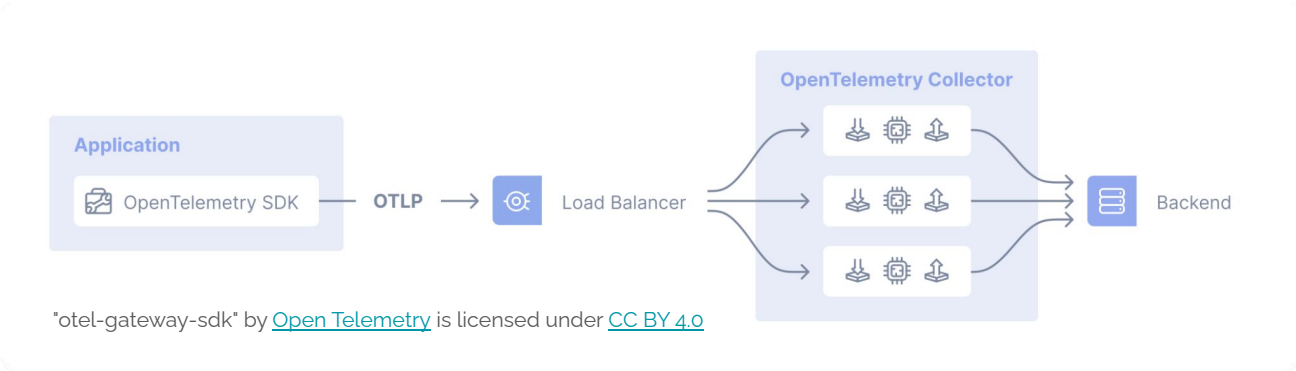


"otel-agent-sdk" by Open Telemetry is licensed under CC BY 4.0

Konfiguration für Backend Kommunikation kann ausgelagert werden.
Sidecar-Injection mittels Operator möglich.

Einzelne “Collector” Instanz skaliert nicht zum Support von vielen Applikationen.

Deployment-Patterns: Gateway

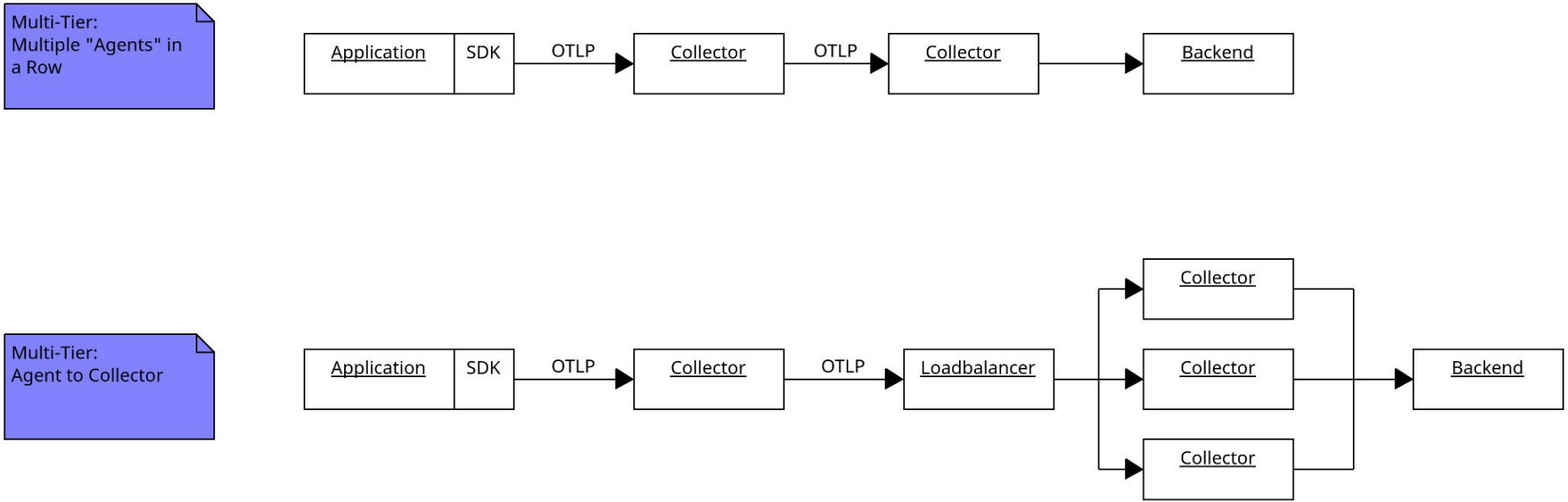


"otel-gateway-sdk" by Open Telemetry is licensed under CC BY 4.0

Zentrale Konfiguration und Weiterleitung.
Skalierbar.

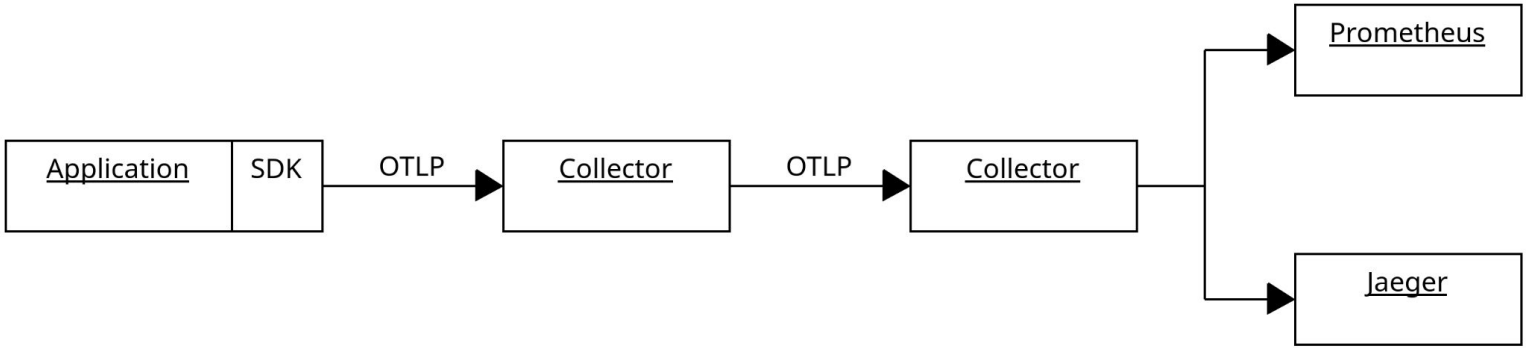
Multi-Tier Collector Setups können notwendig sein für manche Use-Cases.
Multi-Tier Collector Setups erhöhen Latency
Ein weiterer Service der gewartet werden muss.

Deployment-Patterns: Gateway



Benutze den [Loabalancing Exporter]!

Deployment-Patterns: Fan-Out



Agents (oder Gateways) können genutzt werden um Daten zu mehreren Backends zu senden

Braucht mehr Konfigurationsaufwand und Wartung.
Daten können dupliziert werden und mehr Speicher verbrauchen.

Deployment-Patterns: Vergleich

No Collector: “Nur” Open Telemetry SDK und Backends

Agent: Collector Instanzen als Sidecar oder Stand-Alone

Gateway: mehrere Collector Instanzen und Loadbalancer als Open Telemetry Service

Fan-Out: Nutzen von Collector Instanzen um mehrere Ziele anzureichern (weitere Collector Instanzen oder Backends)

Deployment-Patterns: Vergleich

Patterns bauen aufeinander auf und können einzeln oder gemeinsam eingesetzt werden (zB Agent und Gateway)

Umso mehr Collector Instanzen im Einsatz umso mehr Konfiguration und Komponenten müssen zusätzlich überwacht und gewartet werden

Nutze und Verbinde die Patterns dem Use-Case entsprechend.

Helm-Charts und Operator unterstützen bei der Konfiguration und Wartung von Collector Instanzen (mehr dazu später)

Beispiel:
Deployment Patterns

Grundlagen: Helm-Charts Komponenten

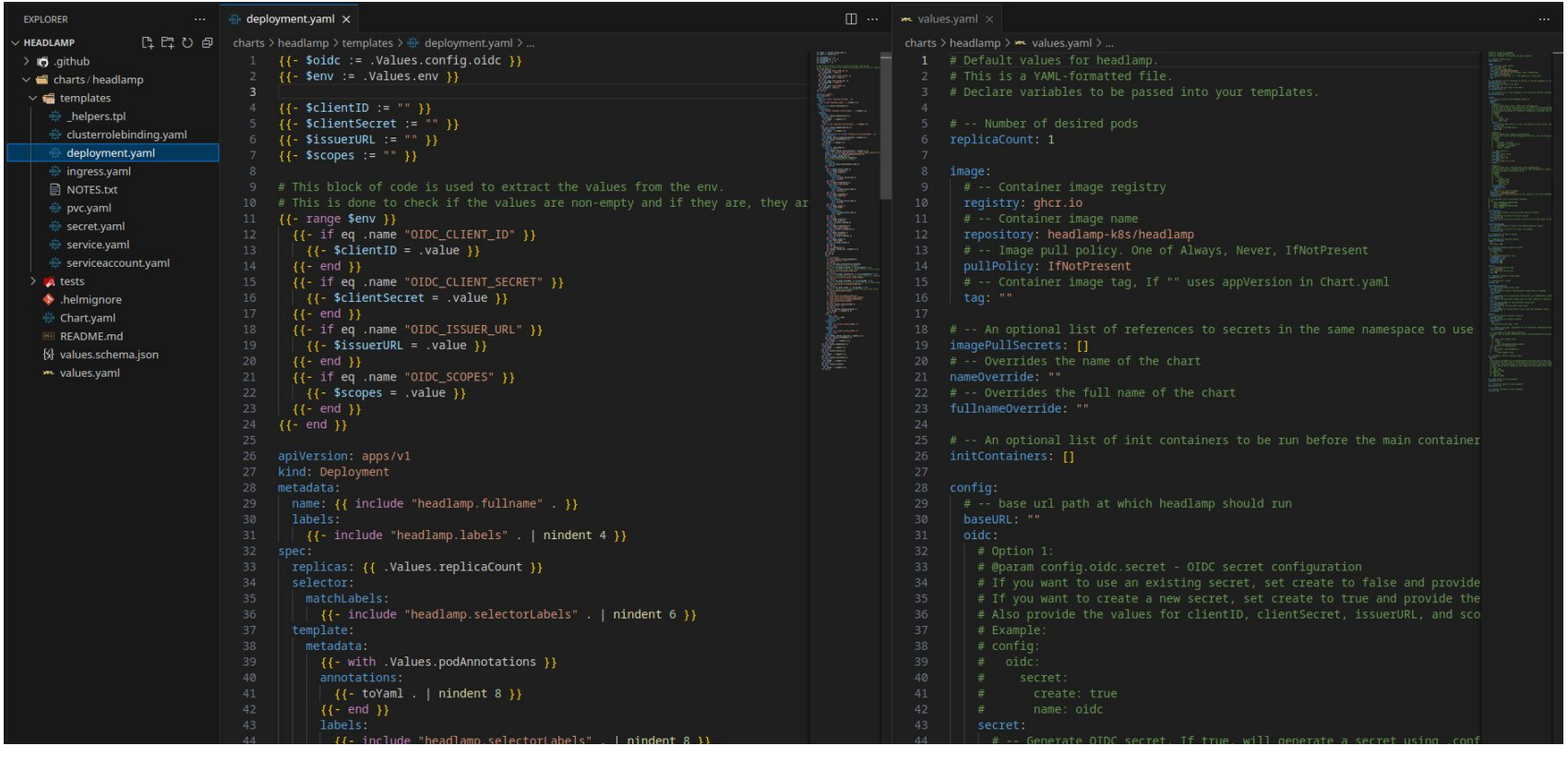
Charts: Die jeweiligen “packages” die installiert werden können.
Bestehen aus einer Menge von Kubernetes Resource Templates.

Chart-Repository: Ein Chart-Repository beinhält ein oder mehrere Charts.
Charts können aber auch direkt von ihrem Quellcode aus installiert werden.

Helm CLI: Notwendig zur Durchführung von Helm Befehlen

Release: Eine laufende Instanz eines Charts in einem Cluster.

Grundlagen: Helm-Charts Komponenten



Grundlagen: Helm-Charts

Keine Installation am Cluster notwendig. Helm funktioniert sofort.

Release Informationen werden von Helm als Kubernetes Secrets gespeichert:

```
marcel@marcel-ThinkPad-P53:~/Documents/code/github|⇒  helm list -n kube-system
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
my-headlamp     kube-system     3               2024-07-26 16:22:53.742253327 +0200 CEST        deployed        headlamp-0.18.2 0.22.0
marcel@marcel-ThinkPad-P53:~/Documents/code/github|⇒  kubectl get -n kube-system secret
NAME                                TYPE                                  DATA   AGE
bootstrap-token-abcdef              bootstrap.kubernetes.io/token         6      21m
oidc                                Opaque                                0      17m
sh.helm.release.v1.my-headlamp.v1   helm.sh/release.v1                    1      17m
sh.helm.release.v1.my-headlamp.v2   helm.sh/release.v1                    1      21s
sh.helm.release.v1.my-headlamp.v3   helm.sh/release.v1                    1      8s
marcel@marcel-ThinkPad-P53:~/Documents/code/github|⇒  helm history -n kube-system my-headlamp
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Fri Jul 26 16:05:18 2024        superseded      headlamp-0.18.2 0.22.0          Install complete
2               Fri Jul 26 16:22:41 2024        superseded      headlamp-0.18.2 0.22.0          Upgrade complete
3               Fri Jul 26 16:22:53 2024        deployed        headlamp-0.18.2 0.22.0          Upgrade complete
marcel@marcel-ThinkPad-P53:~/Documents/code/github|⇒
```

Grundlagen: Helm-Charts ff.

```
marcel@marcel-ThinkPad-P53:~/Documents/code/github|⇒  kubectl get -n kube-system all
NAME                                             READY   STATUS    RESTARTS   AGE
pod/coredns-76f75df574-ldwxk                     1/1     Running   0          35m
pod/coredns-76f75df574-v5msh                     1/1     Running   0          35m
pod/etcd-kind-control-plane                      1/1     Running   0          35m
pod/kindnet-n6474                                1/1     Running   0          35m
pod/kube-apiserver-kind-control-plane            1/1     Running   0          35m
pod/kube-controller-manager-kind-control-plane   1/1     Running   0          35m
pod/kube-proxy-h8mq9                             1/1     Running   0          35m
pod/kube-scheduler-kind-control-plane            1/1     Running   0          35m
pod/my-headlamp-6d574b55df-rkt9w                 1/1     Running   0          32m

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   35m
service/my-headlamp  ClusterIP   10.96.84.101   <none>        80/TCP                   32m

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/kindnet      1         1         1       1            1           kubernetes.io/os=linux   35m
daemonset.apps/kube-proxy   1         1         1       1            1           kubernetes.io/os=linux   35m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns        2/2     2            2           35m
deployment.apps/my-headlamp    1/1     1            1           32m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-76f75df574         2         2         2       35m
replicaset.apps/my-headlamp-6d574b55df     1         1         1       32m
marcel@marcel-ThinkPad-P53:~/Documents/code/github|⇒
```

Open Telemetry Helm-Charts

Open Telemetry Demo: Helm-Chart das versucht Open Telemetry in einer realitätsnahem Umgebung herzuzeigen

Open Telemetry Collector: Helm-Chart für das Aufsetzen von Collector Instanzen in Kubernetes

Open Telemetry Operator: Ein Kubernetes Operator der dabei Unterstützt Collector Instanzen laufen zu lassen.
Unterstützt Sidecar-Injection.

Repository:
https://github.com/open-telemetry/opentelemetry-helm-charts/tree/main

Beispiel:
Open Telemetry Helm-Charts

Grundlagen: Ingress

Nur HTTP/HTTPS
(für andere Typen nutze Service.Type=NodePort und Service.Type=LoadBalancer mehr später)

Verarbeitet vom Ingress-Controller

Der Ingress-Controller benutzt entweder einen internen Loadbalancer oder konfigurierte externe. Für eine Liste von Controllern siehe: https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

Ziel: interne Services (HTTP/HTTPS) externen Applikationen verfügbar machen

Grundlagen: Ingress

Controllers die von Kubernetes gewartet werden:
AWS, GCE, nginx

Community Controllers:
AKS, Traefik, Haproxy, Cilium

Zum Erinnern:
Sie nutzen entweder Loadbalancer Pods oder externe Loabalancer

Multiple Ingress-Controller können in parallel existieren
(allerdings benötigt es mehr Aufwand andere als den “default” zu nutzen)

Grundlagen: Kubernetes Service

TCP und UDP

Können über IP oder DNS genutzt werden.
DNS wird vom Cluster DNS Service zur Verfügung gestellt (zumeist CoreDNS)

Kann nur für interne Services (ClusterIP) oder externe Services konfiguriert werden(NodePort bzw. LoadBalancer)

Ziel: Service Discovery und zentraler Endpunkt für Applikationszugriff

Ingress VS Service

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
        nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
        paths:
        - path: /testpath
          pathType: Prefix
          backend:
            service:
              name: test
              port:
                number: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
        app.kubernetes.io/name: MyApp
  ports:
        - name: http
          protocol: TCP
          port: 80
          targetPort: 9376
        - name: https
          protocol: TCP
          port: 443
          targetPort: 9377
```

Grundlagen: Kubernetes Service

Ein Kubernetes Service mit “selector” Konfiguration erstellt automatisch **EndpointSlice** Objekte, jedoch können diese auch manuell angelegt werden.

Service.Type=NodePort:
Es wird ein Port am Worker-Node für den Service geöffnet. Externe clients müssen auf den Worker-Node Port zugreifen (sie wissen vom Service nichts)

Service.Type=LoadBalancer:
Ein cloud-controller-manager konfiguriert einen Loadbalancer basierend auf dem Service

Service.Type=ExternalName:
Erstellt einen Service als “Endpunkt” für externe Services (zB. DB)

Grundlagen: Kubernetes Service

```
apiVersion: discovery.k8s.io/v1
kind: EndpointSlice
metadata:
  name: example-abc
  labels:
    kubernetes.io/service-name: example
addressType: IPv4
ports:
  - name: http
    protocol: TCP
    port: 80
endpoints:
  - addresses:
      - "10.1.2.3"
    conditions:
        ready: true
    hostname: pod-1
    nodeName: node-1
    zone: us-west2-a
```

```
addressType: IPv4
apiVersion: discovery.k8s.io/v1
endpoints:
- addresses:
  - 172.18.0.2
  conditions:
    ready: true
kind: EndpointSlice
metadata:
  labels:
    kubernetes.io/service-name: kubernetes
  name: kubernetes
  namespace: default
ports:
- name: https
  port: 6443
  protocol: TCP
```

Open Telemetry: Best Practices

Initialize Early: SDKs müssen früh initialisiert werden um alle Informationen zu sammeln

Use Collector: Benutze einen Collector für die Sammlung von Daten für Batching, Sampling und externe Export Konfiguration

Use Sampling: Verwende Sampling um Datenmengen zu reduzieren

Capture critical data: HTTP requests, DB requests, ...

Open Telemetry: Best Practices

Export data: Daten müssen auch wirklich zu externen Applikationen (Jaeger, Grafana, Splunk, …) exportiert werden!

Secure Data: Stelle sicher das Authentication und Encryption aktiviert sind

Start with Automatic Instrumentation: Beginne mit der automatischen Instrumentalisierung von Applikationen und gehe zu manueller über wo notwendig

Use Semantic information: Stelle sicher das es eine semantische Konvention für Meta-Informationen gibt

Open Telemetry: Best Practices

https://uptrace.dev/opentelemetry/distributed-tracing.html#best-practices

https://www.honeycomb.io/blog/opentelemetry-best-practices

https://github.com/open-telemetry/opentelemetry-collector/blob/main/docs/security-best-practices.md

https://grafana.com/blog/2023/12/18/opentelemetry-best-practices-a-users-guide-to-getting-started-with-opentelemetry/

Open Telemetry: Best Practices

https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/best-practices/metrics

https://docs.datadoghq.com/opentelemetry/ingestion_sampling_with_opentelemetry/

https://www.elastic.co/observability-labs/blog/best-practices-instrumenting-opentelemetry

Open Telemetry: Sampling

https://opentelemetry.io/docs/concepts/sampling/

https://opentelemetry.io/blog/2022/tail-sampling/

Open Telemetry: Security

Verwende Authentication Extensions bei Receivers und Exporters

Anonymisiere Daten bereits so früh als möglich (zB Sidecar Collector)

Aktiviere empfohlene Processor

Verwende verschlüsselte Verbindungen (zB HTTPS statt HTTP)

Happy Tracing!

whizus

Ab hier folien für kopieren (sind die vorlagen)

Some words about me …

This is me

This is me too

This is me

Few points

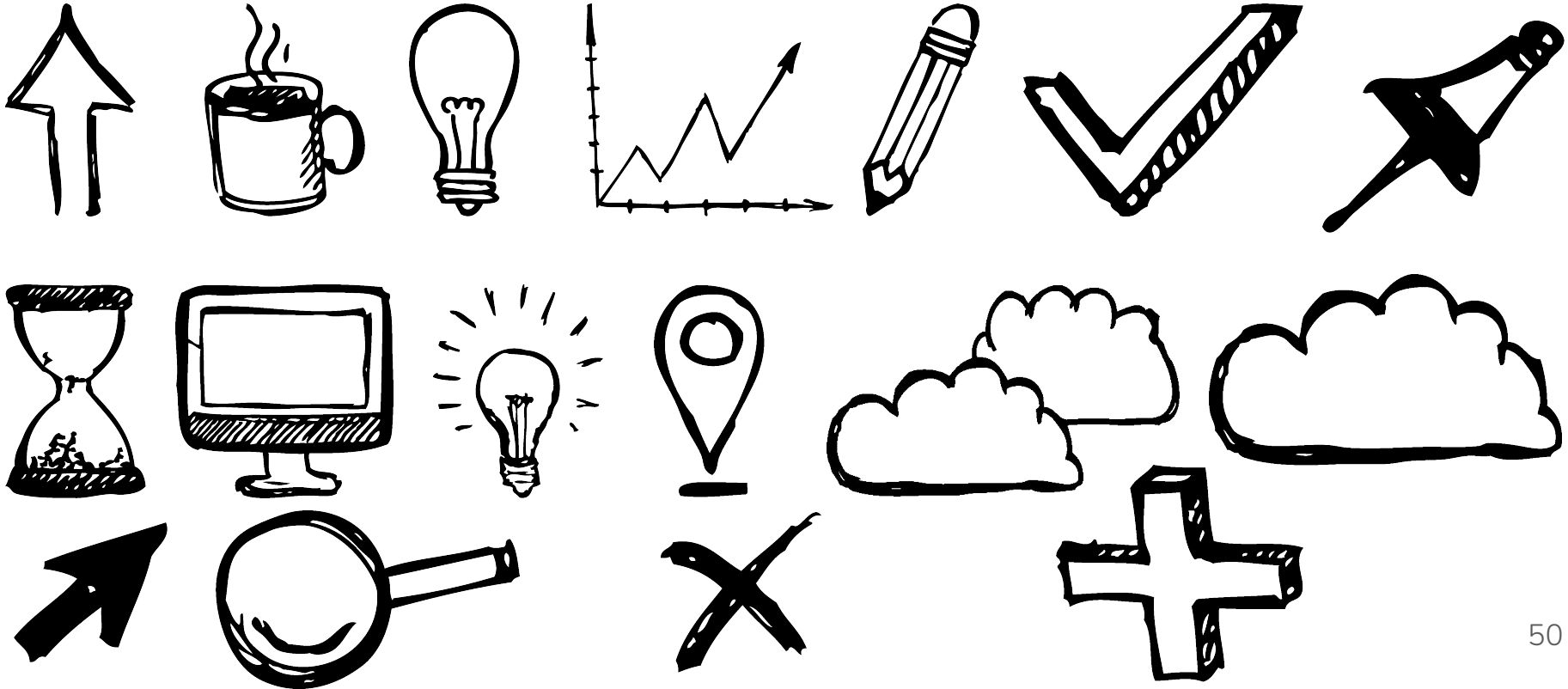
One point

Two points

3 makes a charm


What a wonderful world

Icon Pack



Agenda

- Point to talk about
- Another thing to talk about
- The greatest thing of all
- Kidding, this is really the best thing of all time
- Well, but you can't miss this one either
- Wrapping it all up

Buzzword