Meet Up

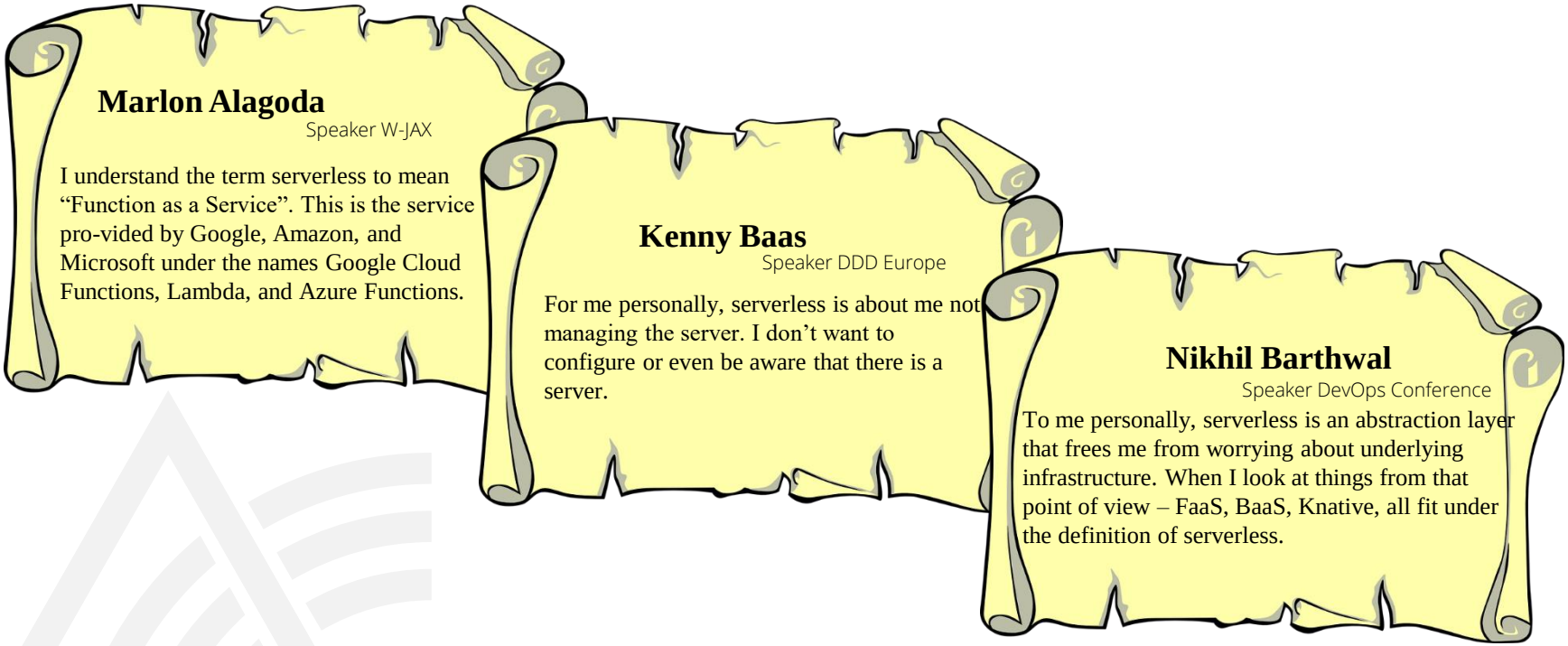
Serverless mit Knative

Stefan Kühnlein – Senior Solution Architekt

Agenda

1. FaaS - Serverless
2. Knative
3. Knative - Eventing
4. Knative - Serving

Function as a Service / Serverless



Function as a Service / Serverless

- Cloud-Anbieter bieten die Möglichkeit „kleine“ Funktionen in einem PaaS Service auszuführen
- Funktionen werden in der Regel in Containern ausgeführt.
- Management der Virtualisierung, Server und Container übernimmt der Cloud-Anbieter
- Entwickler können sich ausschließlich auf zu implementierende Business Logik konzentrieren
- Kostengünstige Abrechnungsmodelle

Übersicht von FaaS bzw. Serverless Frameworks

- Frameworks auf Basis von Docker
  - nuclio
  - OpenWhisk
  - OpenFaaS
  - Fn Project
- Frameworks auf Basis von Kubernetes
  - nuclio
  - Fission
  - OpenFaaS
  - Kubeless
  - Knative

Charakteristik einer Funktion

- Die Funktion besteht aus Sicht des Entwicklers nur aus dem Quellcode
- Die Funktion muss zustandslos sein
- Die Funktion wird durch einen Trigger gestartet
  - Messages
  - Timer
  - HTTP
  - Kubernetes Events
- Die Funktion hat für alle Trigger stets dasselbe Interface

Entwicklungsumgebung für FaaS

- Funktionen sind nur Quellcode
- Es wird kein gesonderter Build-Schritt implementiert
- Es müssen i.R. keine Container gebaut oder durch eine Registry verwaltet werden

Laufzeitumgebung für FaaS

- Die Verwaltung der Funktionen und Laufzeitumgebung im Cluster erfolgt durch das verwendete Framework
  - Keine Erzeugung von zusätzlichen Kubernetes Ressourcen wie Services, Ingress
  - Wichtige Teile der Infrastruktur wie Logging oder Skalierung werden ebenfalls durch das Framework zur Verfügung gestellt
  - Auf andere Pods oder Services des Clusters können wie gewohnt zugegriffen werden
  - Skalierung erfolgt automatisch
  - Lebenszeit der Container der Laufzeitumgebung ist länger als die der Funktion

Typische Anwendungsfälle für FaaS

- Kleine CRUD Services
- Simple Web Hooks
- Kleine WebServer zum Ausspielen einzelner HTML Seiten
- Endpunkte für IOT
- Timer Actions, geplante Tasks oder Jobs
- Reaktion auf Kubernetes Events

Knative

Knative erweitert Kubernetes um die Möglichkeit Serverless Workloads ohne herstellerspezifische Mechanismen abzubilden (z.B. AWS Lamda, Azure functions)

Einführung in Knative

- Knative (kay-nay-tive) erweitert Kubernetes um einen Satz von Middleware Komponenten die für die Erstellung von containerbasierten Anwendungen notwendig sind
- Highlights
  - Fokussierte API mit einem höheren Abstraktionslevel für die gängigsten Anwendungsfälle
  - Bereitstellung von skalierbaren, sicheren und stateless Services in wenigen Sekunden
  - Lose Kopplung der Funktionen
  - Pluggable Komponenten für Logging, Monitoring und Service Mesh
  - Knative kann auf allen Kubernetes Plattformen ausgeführt werden
  - Unterstützt die gängigsten Entwickler Experiences wie GitOps, DockerOps, ManualOps
  - Knative kann mit den gängigen Tools und Frameworks wie Django, Ruby on Rails, Spring und vielen weiteren verwendet werden.

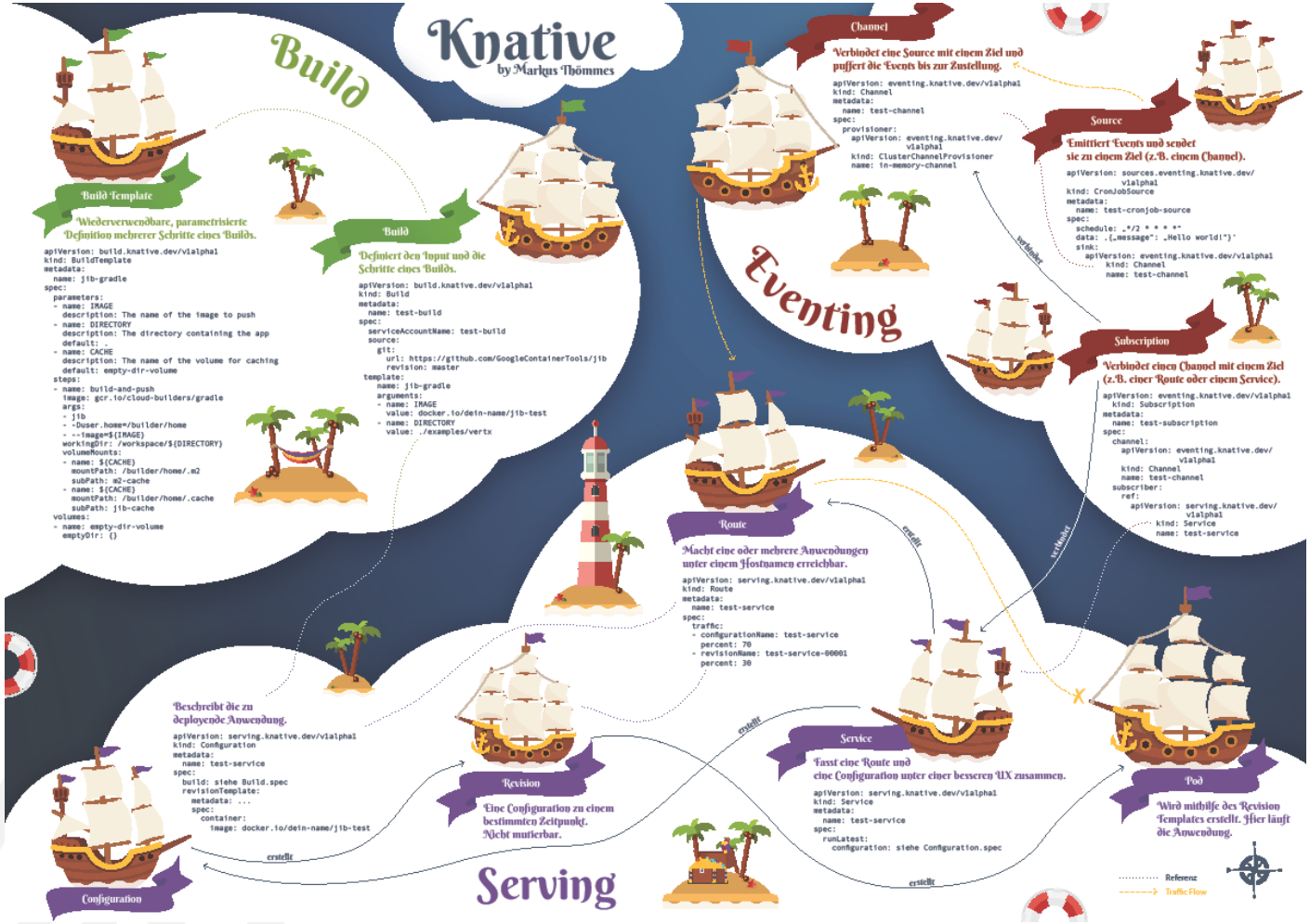
Einführung in Knative

- Knative ist ein Open Source Projekt initiiert von Google und wird von vielen Unternehmen wie Pivotal, IBM, Red Hat und SAP unterstützt
- Knative Komponenten konzentrieren sich auf die Lösung alltäglicher, aber schwieriger Aufgaben wie:
  - Bereitstellung von Containern
  - Routing und Traffic Management mit blue/green deployment
  - Automatische Skalierung der Worksload on demand
  - Bindung von laufenden Services an eventbasierte Systeme

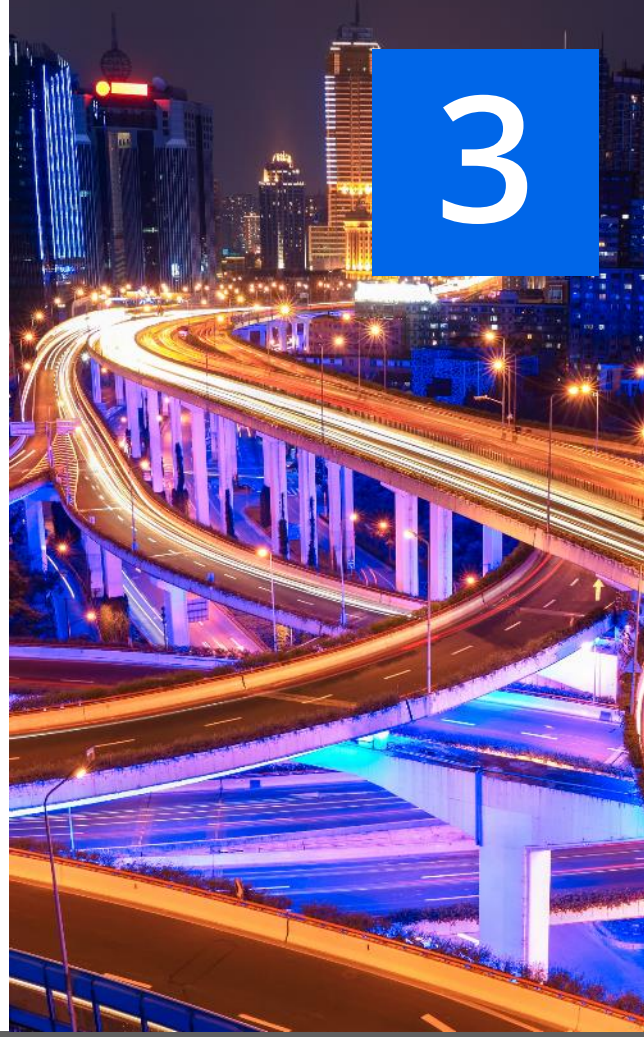
Einführung in Knative

- Releases von Knative
  - v0.10.0 vom 30. Oktober 2019
  - v0.9.0 vom 17. September 2019
  - v0.8.1 vom 28. August 2019
  - V0.8.0 vom 6. August 2019 (aka „v1rc1“)
- Komponenten von Knative
  - Serving - Request-driven compute that can scale to zero
  - Eventing - Management and delivery of events
  - Build – Source to Container Orchestration bis einschließlich V0.7

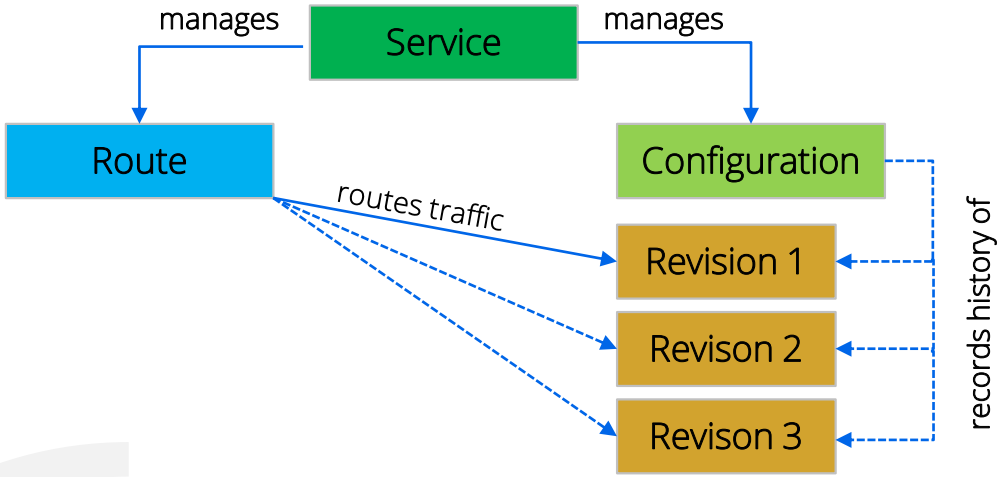
Architekturübersicht Knative


Knative
by Markus Thömmes
Build
Build Template
Wiederverwendbare, parametrisierte Definition mehrerer Schritte eines Builds.
apiVersion: build.knative.dev/v1alpha1
kind: BuildTemplate
metadata:
name: jib-gradle
spec:
parameters:
- name: IMAGE
description: The name of the image to push
- name: DIRECTORY
description: The directory containing the app
default: .
- name: CACHE
description: The name of the volume for caching
default: empty-dir-volume
steps:
- name: build-and-push
image: gcr.io/cloud-builders/gradle
args:
- jib
- -Duser.home=/builder/home
- --image=${IMAGE}
workingDir: /workspace/${DIRECTORY}
volumeMounts:
- name: ${CACHE}
mountPath: /builder/home/.m2
subPath: m2-cache
- name: ${CACHE}
mountPath: /builder/home/.cache
subPath: jib-cache
volumes:
- name: empty-dir-volume
emptyDir: {}
Build
Definiert den Input und die Schritte eines Builds.
apiVersion: build.knative.dev/v1alpha1
kind: Build
metadata:
name: test-build
spec:
serviceAccountName: test-build
source:
git:
url: https://github.com/GoogleContainerTools/jib
revision: master
template:
name: jib-gradle
arguments:
- name: IMAGE
value: docker.io/dein-name/jib-test
- name: DIRECTORY
value: ./examples/vertx
Eventing
Channel
Verbindet eine Source mit einem Ziel und puffert die Events bis zur Zustellung.
apiVersion: eventing.knative.dev/v1alpha1
kind: Channel
metadata:
name: test-channel
spec:
provisioner:
apiVersion: eventing.knative.dev/
v1alpha1
kind: ClusterChannelProvisioner
name: in-memory-channel
Source
Emittiert Events und sendet sie zu einem Ziel (z.B. einem Channel).
apiVersion: sources.eventing.knative.dev/
v1alpha1
kind: CronJobSource
metadata:
name: test-cronjob-source
spec:
schedule: „*/2 * * * *"
data: ‚{„message": „Hello world!"}'
sink:
apiVersion: eventing.knative.dev/v1alpha1
kind: Channel
name: test-channel
verbindet
Subscription
Verbindet einen Channel mit einem Ziel (z.B. einer Route oder einem Service).
apiVersion: eventing.knative.dev/v1alpha1
kind: Subscription
metadata:
name: test-subscription
spec:
channel:
apiVersion: eventing.knative.dev/
v1alpha1
kind: Channel
name: test-channel
subscriber:
ref:
apiVersion: serving.knative.dev/
v1alpha1
kind: Service
name: test-service
verbindet
Route
Macht eine oder mehrere Anwendungen unter einem Hostnamen erreichbar.
apiVersion: serving.knative.dev/v1alpha1
kind: Route
metadata:
name: test-service
spec:
traffic:
- configurationName: test-service
percent: 70
- revisionName: test-service-00001
percent: 30
erstellt
Beschreibt die zu deployende Anwendung.
apiVersion: serving.knative.dev/v1alpha1
kind: Configuration
metadata:
name: test-service
spec:
build: siehe Build.spec
revisionTemplate:
metadata: ...
spec:
container:
image: docker.io/dein-name/jib-test
Configuration
erstellt
Revision
Eine Configuration zu einem bestimmten Zeitpunkt. Nicht mutierbar.
erstellt
Service
Fasst eine Route und eine Configuration unter einer besseren UX zusammen.
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
name: test-service
spec:
runLatest:
configuration: siehe Configuration.spec
erstellt
Pod
Wird mithilfe des Revision Templates erstellt. Hier läuft die Anwendung.
Referenz
Traffic Flow
Serving

Knative Eventing



Knative Komponenten – Serving



Knative Komponenten - Serving

- Service - *service.serving.knative.dev*

  Die Komponente Service steuert die Erstellung weiterer Objekte, um sicherzustellen, dass die App für jede Aktualisierung des Dienstes über eine Route, Konfiguration und eine neue Revision verfügt. Der Dienst kann so definiert werden, dass der Datenverkehr immer zur neusten Version oder zu einer gepinnten Version weitergeleitet wird.

- Route – *route.service.knative.dev*

  Diese Komponente ordnet einen Netzwerkendpunkt einer oder mehreren Revisionen zu. Sie können den Verkehr auf verschiedene Arten verwalten, einschließlich Teilverkehr und benannten Routen.

- Configuration - *configuration.serving.knative.dev*

  Mit Hilfe dieser Komponte erfolgt eine saubere Trennung zwischen Code und Konfiguration gemäß den Vorgaben der Twelve-Factor-App Methode. Durch die Änderung an einer Konfiguration wird eine neue Revision erstellt.

- Revision - *revision.serving.knative.dev*

  Die Ressource ist eine Momentaufnahme des Codes und der Konfiguration für jede an der Workload vorgenommene Änderung. Revision sind unveränderliche Objekte und können so lange aufbewahrt werden, wie es sinnvoll ist.

Knative Serving - Trafficmanagment

- Knative ist in der Lage den eingehenden Traffic auf verschieden Revisionen zu verteilen.

Definition of Revision 1

```
apiVersion: serving.knative.dev/v1alpha1
kind: Configuration
metadata:
  name: meetup
spec:
  template:
    metadata:
      name: meetup-v1
      sidecar.istio.io/inject: "false"
    spec:
      containers:
        - image: <image_url>:v_1
          env:
            - name: service.hostname
              value: "Guten Abend"
```

Definition of Revision 2

```
apiVersion: serving.knative.dev/v1alpha1
kind: Configuration
metadata:
  name: meetup
spec:
 template:
    metadata:
      name: meetup-v2
      sidecar.istio.io/inject: "false"
    spec:
      containers:
        - image: <image_url>:v_1
          env:
            - name: service.hostname
              value: "Willkommen beim Meetup"
```

Definition of Route

```
apiVersion:serving.knative.dev/v1alpha1
kind: Route
metadata:
  name: meetup
spec:
  traffic:
    - revisionName: meetup_v1_<pod>
      percent: 25
    - revisionName: meetup_v2_<Pod>
      percent: 75
```

Knative Serving – Autoscaler

- Scale-to-zero

  Ist die Haupteigenschaft, die Knative zu einer Serverless Platform macht.
  Die Konfiguration erfolgt in einer ConfigMap namens *config-autoscaler* im Namespace knative-serving.

Default configuration auto-scaler

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-autoscaler
  namespace: knative-serving
data:
  container-concurrency-target-default: 100
  container-concurrency-target-percentage: 1.0
  enable-scale-to-zero: true
  enable-vertical-pod-autoscaling: false
  max-scale-up-rate: 10
  panic-window: 6s
  scale-to-zero-grace-period: 30s
  stable-window: 60s
  tick-interval: 2s
```

Knative Serving – Scaling

- Scale-to-zero
  - enable-scale-to-zero
    aktiviert oder deaktiviert dieses Feature
  - scale-to-zero-grace-period
    Ein Zeitwert mit einen minimalen Wert von 6 Sekunden.
    Ist diese Periode abgelaufen, so werden alle Pods gestoppt und es werden keine Ressourcen mehr verbraucht
    Eingehende Anfragen werden an den Activator weitergeleitet, der die Erstellung neuer Pods veranlasst.
  - stable-window
    Ein Zeitwert mit einen minimalen Wert von 6 Sekunden.

Wann die Pods beendet werden, ergibt die Summe der Parameter *scale-to-zero-grace-period* und *stable-window*

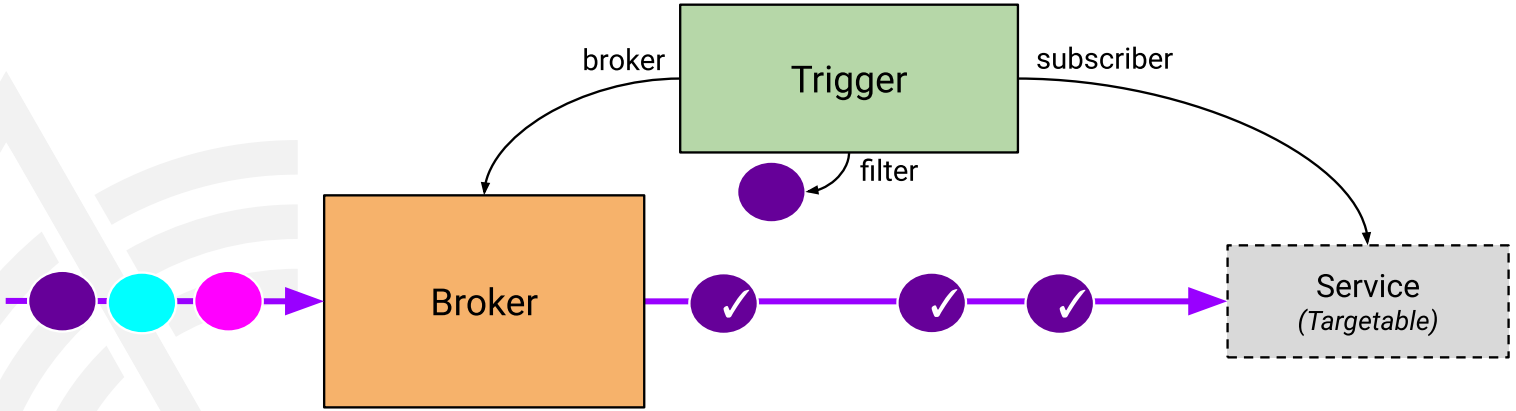
Übung -> Pizza & Beer

- Checkout Git-Repository - https://github.com/opitzconsulting/oc-expertcamp-knative
- Install necessary tools
- Login to Openshift Cluster
  oc login https://api.meetup.openshift.opitz.cloud:6443 –p r3dh4t1! –u userXX
- Build your own Container
- Deploy your first Knative Service
- See what you have deployed

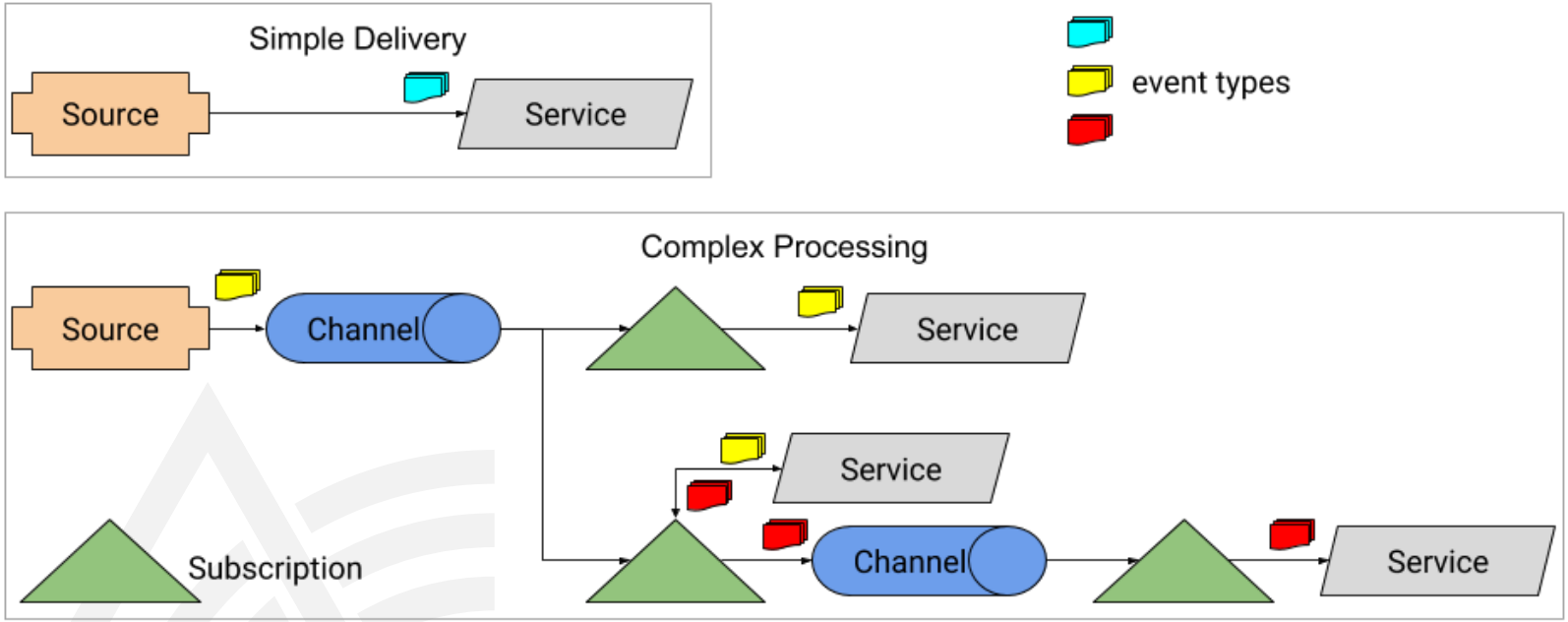
Knative Eventing

Knative Eventing – Triggers und Broker

- Broker
  - Stellt eine Reihe von Ereignissen bereit
  - Empfängt Ereignisse und leitet diese an Abonnenten weitere, die von einem oder mehreren Triggern definiert wurden
- Trigger
  - Beschreibt einen Filter für Ereignisattribute, die an einem Empfänger gesendet werden sollen



Knative Eventing - Architektur



Knative Eventing - Sources

- KubernetesEventSource
- GitHubSource
- GcpPubSubSource
- AwsSqsSource
- ContainerSource
- CronJobSource
- KafkaSource
- CamelSource

Knative Eventing – Source to Service

- Event Source

  Die Event Source lauscht auf ein externes Event. Diese ist Verantwortlich, die Daten zu sammeln und an den Service – sink – weiterzuleiten

Beispiel einer CronJobSource mit einem Service als Sink

```
apiVersion: serving.knative.dev/v1alpha1
kind: CronJobSource
metadata:
  name: meetup-cronjob-source
spec:
  template:
    schedule: "* * * * *"
    data: '{"message": Hello Meetup!"}'
    sink:
    # Events werden direkt an den angegebenen Service geschickt.
    apiVersion: serving.knative.dev/v1alpha1
    kind: Service
    name: helloworld
```

… überraschend mehr Möglichkeiten

Stefan Kühnlein

Senior Solution Architekt

Weltenburger Str. 4
81677 München

stefan.kuehnlein@opitz-consulting.com
+49 173 7279307

WWW.OPITZ-CONSULTING.COM

@OC_WIRE

OPITZCONSULTING

opitzconsulting

opitz-consulting-bcb8-1009116

Knative Komponenten – Serving

- Route

  Die Ressource *route.serving,knative.dev* ordnet einen Netzwerkpunkt eine oder mehrere Ressourcen zu.