开源技术 * IBM微讲堂

Knative Eventing打造标准的云原生事件平台

郭迎春（ Daisy ）
guoyingc@cn.ibm.com

IBM **Developer**

IBM

Knative系列公开课

https://developer.ibm.com/cn/os-academy-knative/

每周四晚8点档

- ◆09月19日 《Knative概览》
- ◆09月26日 《Tekton从源码构建容器镜像》
- ◆10月10日 《Knative Serving让容器从另扩展到无限》
- ◆10月17日 《Knative Eventing打造标准云原生事件平台》
- ◆10月24日 《Knative客户端工具介绍》
- ◆10月31日 《案例分享：在DevOps场景中使用Knative》

讲师介绍

郭迎春
IBM开发中心 开放技术工程院
多年开源社区经验
专注Serverless平台建设
Knative贡献者

微信：daisy-ycguo

日程

- Eventing的目标和实现
- Eventing的资源和使用
- 创建你的事件源类型

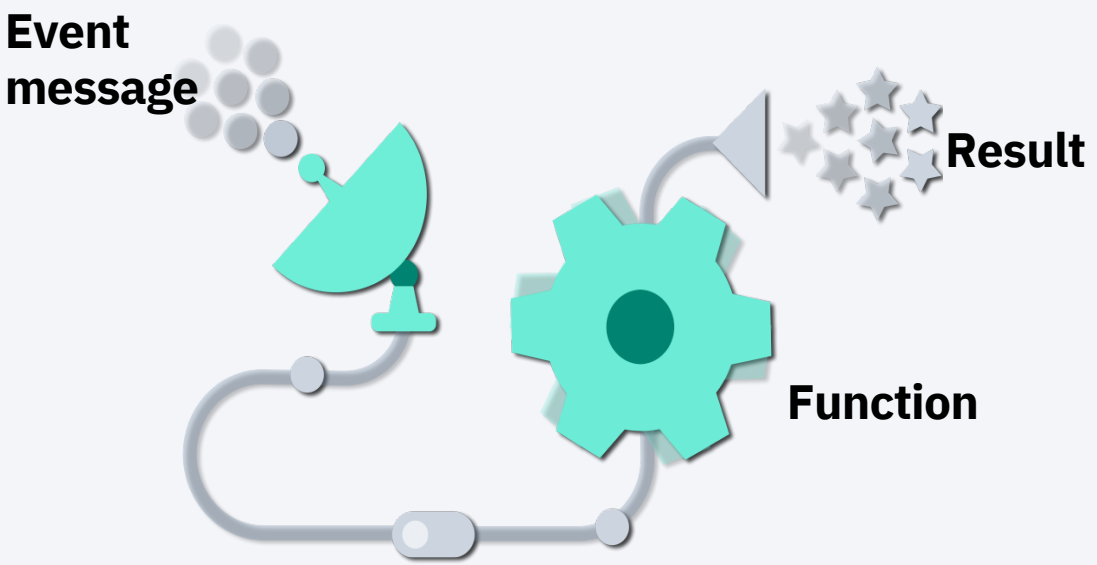
日程

- Eventing的目标和实现
- Eventing的资源和使用
- 创建你的事件源类型

为什么要关心事件？

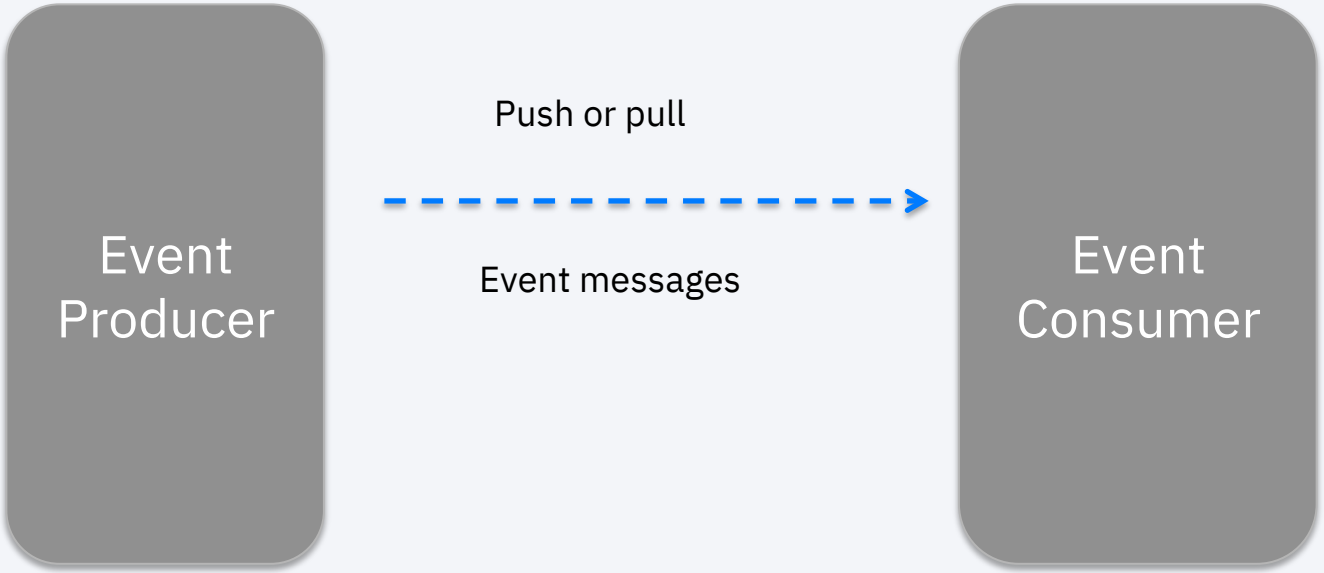
事件无处不在

事件驱动是函数即服务FaaS平台中常见的编程模型。



通用的事件监听模型

- 紧耦合；
- 消息格式互不相同；
- 事件消费者或生产者需要确保通信的可靠；
- 逻辑复杂而且不可重用；
- ……



Knative事件平台的目标

Knative Eventing is a system that is designed to address a common need for cloud native development and provides composable primitives to enable late-binding event sources and event consumers.

Knative Eventing是一个旨在满足云原生开发的常见需求的事件平台，提供**可组合的元素**以支持**后期绑定**事件生产者和事件消费者。

- 松耦合。这些事件元素都可以独立部署或者跨平台部署。
- 事件生产者与事件消费者是独立的。生产者和消费者可以在不知道对方的情况下独立存在。
- 支持第三方服务的接入。
- 支持跨平台和互操作性，采用CNCF定义的标准数据根式CloudEvents

CloudEvent

- 描述事件的规范
- 简化事件的描述，支持跨服务与跨平台的传递与交互
- CNCF Serverless工作组制定
- 提供多语言的SDK

```
{
    "specversion" : "1.0-rc1",
    "type" : "com.github.pull.create",
    "source" : "https://github.com/cloudevents/spec/pull",
    "subject" : "123",
    "id" : "A234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "text/xml",
    "data" : "<much wow=\"xml\"/>"
}
```

一组Kubernetes CRD 自定义资源类型

- **Event Source**：用于把事件生产者接入Knative事件平台，并把事件传送到消费者；

- **Channel**：实现事件转发和存储，支持不同的技术如Apache Kafka和NATS Streaming；
- **Subscription**：事件的订阅者，即是事件转发的目的地；
- **Parallel**：事件同时转发给多个订阅者的一种机制；
- **Sequence**：事件依次经过多个订阅者的一种机制；

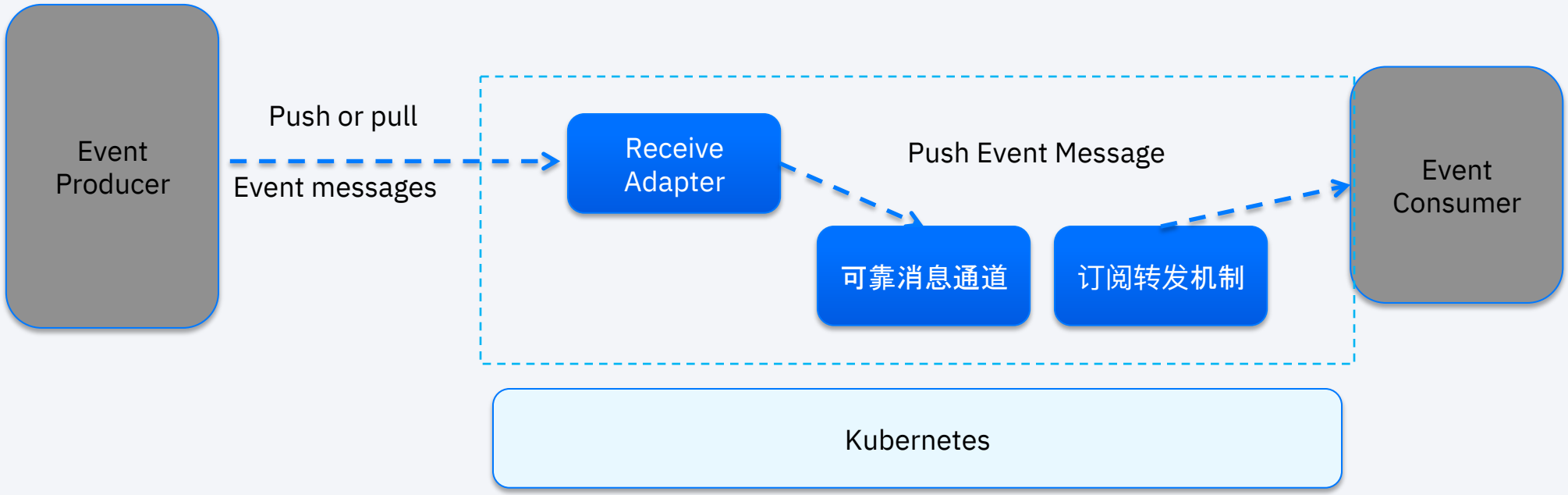
- **Broker**：可以接收事件，并将事件转发到订阅者，相当于 event mesh；
- **Trigger**：对某种事件的订阅；
- **Event Registry**：用于查阅Broker中的事件类型；

Sink

- Sink代表事件消息传送的目的地，暴露一个HTTP端口。
- 在Kubernetes世界中，即带有属性：status.address.hostname
- 有两种类型：
  - Addressable: 暴露一个HTTP端口，接到消息后，返回代码表示成功或者失败。
  - Callable:暴露一个HTTP端口，接到消息后，返回0个或者1个新的事件。
- Knative中的Sink资源：Service，Channel，Broker，Parallel以及Sequence

实现

- 事件生产者(**Event Producer**)产生事件的系统，如Github
- 事件消费者(**Event consumers**)：接收事件的一方，如某个服务或者系统
- 接收适配器(**Receive Adapter**)：数据处理的实体，将外部的事件数据接入，转为CloudEvent格式，并将消息传入事件平台。



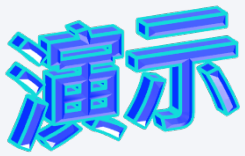
日程

- Eventing的目标和实现
- Eventing的资源和使用
- 创建你的事件源类型

Event Source

- 一个帮手对象，用于把事件生产者接入Knative事件平台，并把事件传送到消费者;
- 是事件生产者在Knative事件平台的抽象;
- 规格说明(Specification)包括:
  - 定义事件源的参数
  - Sink
- 预定义的事件源包括:KubernetesEventSource, GitHubSource, GcpPubSubSource, AwsSqsSource, ContainerSource, CronJobSource

Event Source举例



CronJobSource

```
apiVersion: sources.eventing.knative.dev/v1alpha1
   kind: CronJobSource
   metadata:
   name: cronjobs
   spec:
     schedule: "*/1 * * * *"
     data: "{\"message\": \"Hello world!\"}"
     sink:
       apiVersion: serving.knative.dev/v1alpha1
       kind: Service
       name: event-display
```

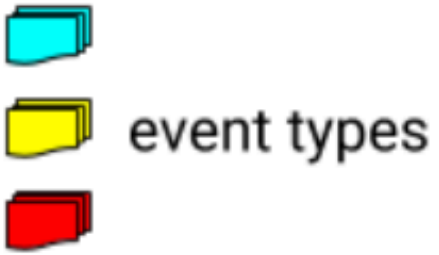
Event Source举例

ContainerSource

```
apiVersion: sources.eventing.knative.dev/v1alpha1
kind: ContainerSource
metadata:
  name: heartbeats-sender
spec:
  image: docker.io/daisyycguo/heartbeats-6790335e994243a8d3f53b967cdd6398
  sink:
    apiVersion: eventing.knative.dev/v1alpha1
    kind: Service
    name: event-display
  args:
    - --period=1
  env:
    - name: POD_NAME
      value: "heartbeats"
    - name: POD_NAMESPACE
      value: "default"
```

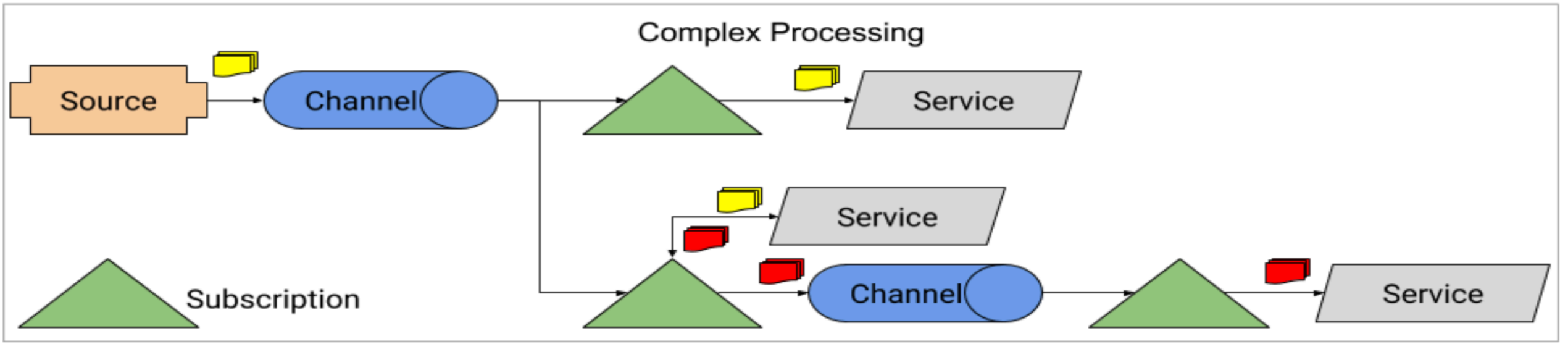
Event Source的使用

- 定义一个事件源
- 通过Sink指定事件消费者的地址
- 适用场景
  - 事件消费者有个接收消息的地址
  - 事件消费者接到消息后，返回代码表示成功或者失败。

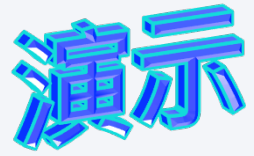


Channel和Subscription

- 通道（Channel）是Kubernetes的自定义资源，代表事件的转发和持久化层。
- 通道（Channel）由 ClusterChannelProvisioner 对象创建，支持不同技术的通道，如 In-memory, Apache Kafka, NATS Streaming 等等.
- 订阅（Subscription）描述通道和事件消费者之间的联系，也可以连接事件消费者和第二个通道，用于处理消费者返回的事件。
- 通道可以由多个订阅，订阅只能连接1个通道和1个事件消费者。



Channel和Subscription举例



Knative Channel

```
apiVersion: messaging.knative.dev/v1alpha1
kind: Channel
metadata:
  name: my-channel
  namespace: default
```

CronJobSource

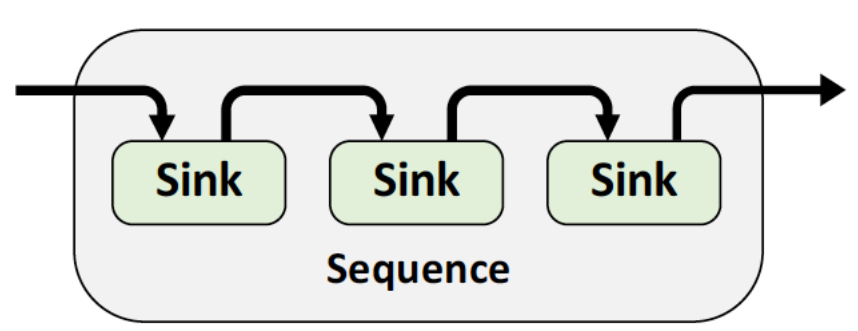
```
apiVersion: sources.eventing.knative.dev/v1alpha1
   kind: CronJobSource
   metadata:
   name: cronjobs
   spec:
     ……
     sink:
       apiVersion: messaging.knative.dev/v1alpha1
       kind: Channel
       name: my-channel
```

Knative Subscription

```
apiVersion: eventing.knative.dev/v1alpha1
  kind: Subscription
  metadata:
    name: mysubscription
    namespace: default
  spec:
    channel:
      apiVersion: messaging.knative.dev/v1alpha1
      kind: Channel
      name: my-channel
    subscriber:
      ref:
        apiVersion: serving.knative.dev/v1alpha1
        kind: Service
        name: event-display
```

Sequence

- Sequence 定义了一系列依次被执行的服务。
- 事件发送给每个服务
- 每个服务都可以修改，过滤或者创建新的事件。



Event Source

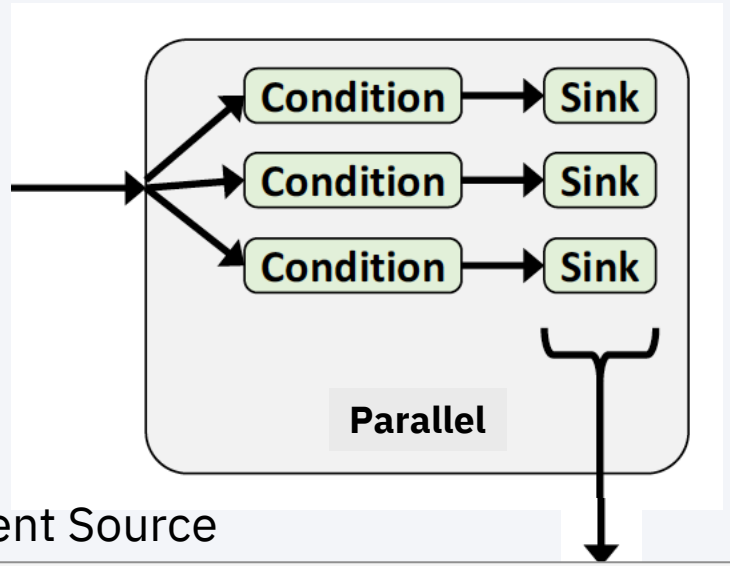
```
spec:
   ……
   sink:
     apiVersion: messaging.knative.dev/v1alpha1
     kind: Sequence
     name: my-sequence
```

Knative Sequence

```
apiVersion: messaging.knative.dev/v1alpha1
kind: Sequence
metadata:
  name: my-sequence
spec:
  steps:
    - ref:
        apiVersion: serving.knative.dev/v1alpha1
        kind: Service
        name: first
    - ref:
        apiVersion: serving.knative.dev/v1alpha1
        kind: Service
        name: second
    - ref:
        apiVersion: serving.knative.dev/v1alpha1
        kind: Service
        name: third
```

Parallel

- Parallel定义了一系列并行执行的服务
- 每个服务接收到同一个事件
- 每个分支可以定义filter



Event Source

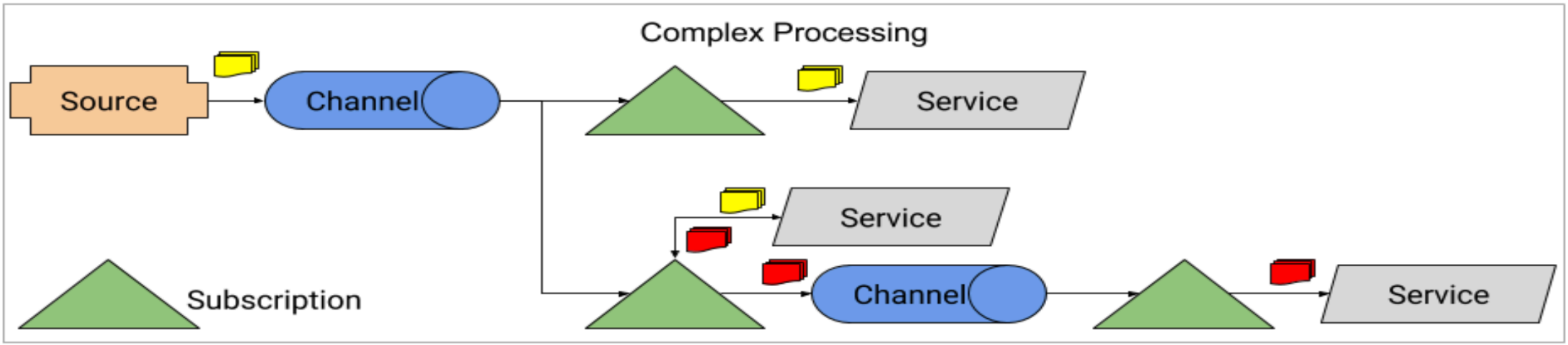
```
spec:
   ……
   sink:
     apiVersion: messaging.knative.dev/v1alpha1
     kind: Parallel
     name: odd-even-parallel
```

Knative Sequence

```
apiVersion: messaging.knative.dev/v1alpha1
kind: Parallel
metadata:
  name: odd-even-parallel
spec:
  branches:
    - filter:
        ref:
          apiVersion: serving.knative.dev/v1alpha1
          kind: Service
          name: even-filter
      subscriber:
        ref:
          apiVersion: serving.knative.dev/v1alpha1
          kind: Service
          name: even-transformer
    - filter:
        ref:
          apiVersion: serving.knative.dev/v1alpha1
          kind: Service
          name: odd-filter
      subscriber:
        ref:
          apiVersion: serving.knative.dev/v1alpha1
          kind: Service
          name: odd-transformer
```

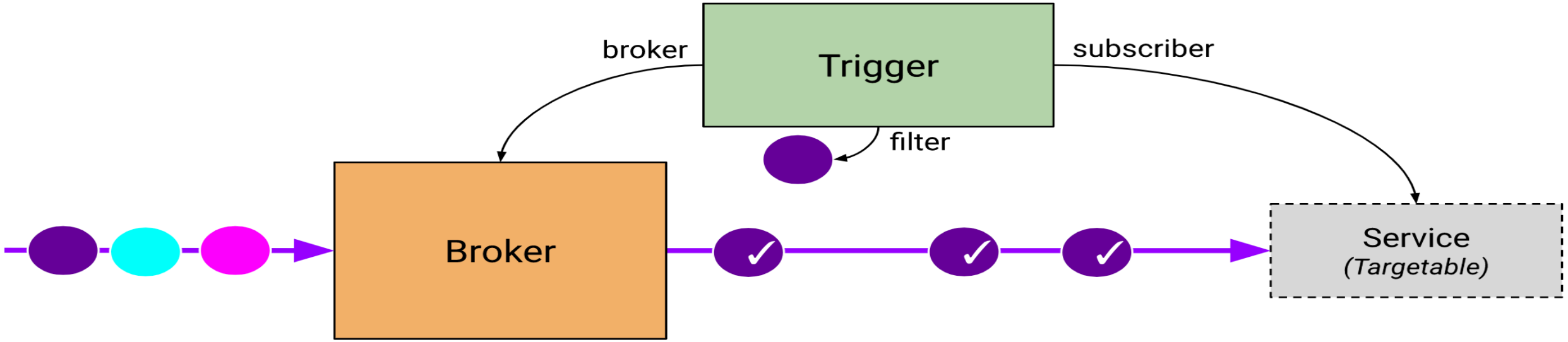
Channel和Subscription的使用

- 创建Channel对象，创建Subscription对象指定通道的订阅者。
- 事件源将事件消息发送到Channel，通道转发给由Subscription指定的事件消费者。
- 适用场景：
  - 单个事件源，多个订阅者
  - 利用Sequence和Parallel处理事件的流程比较复杂，需要多个订阅者协同工作，如同工作流。



Broker和Trigger

- 引入Broker和Trigger的目的，是为了搭建一个黑盒子，将具体的实现隐藏起来，最终用户不用关心。
- Broker如同事件桶，接入各种不同的事件，这些事件可以通过属性来过滤。
- Trigger描述了一个过滤器，只有通过了过滤器选择的事件，可以被传送给事件消费者。



Trigger举例

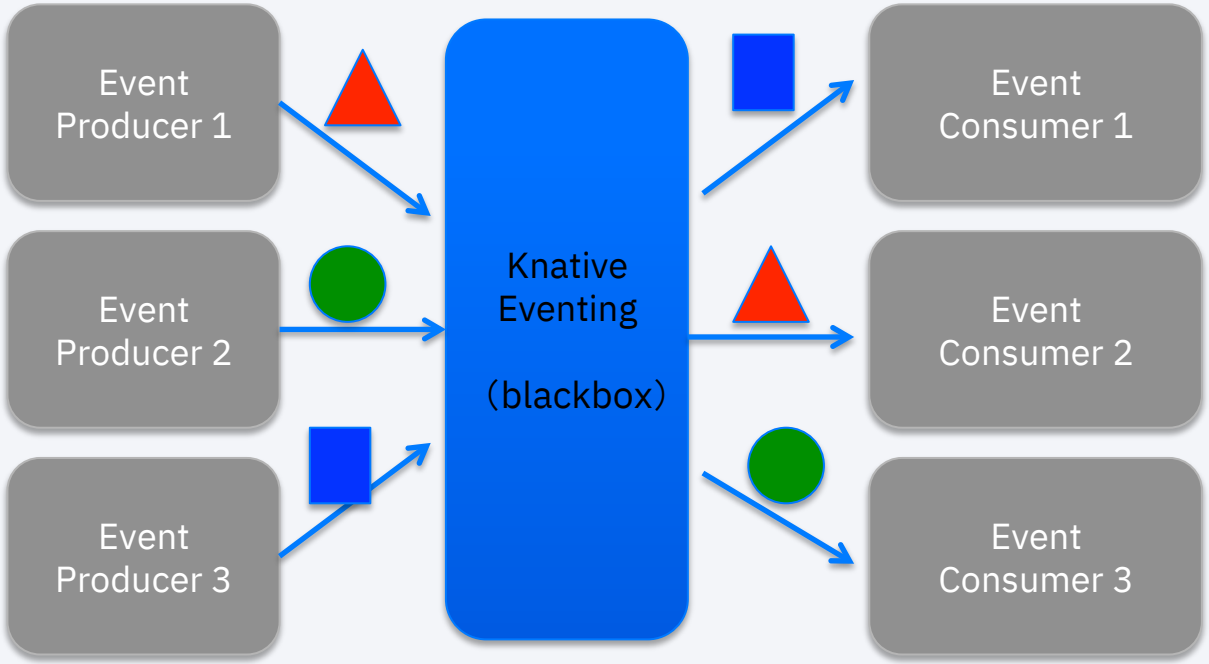
演示

Trigger

```
apiVersion: eventing.knative.dev/v1alpha1
kind: Trigger
metadata:
  name: mytrigger
spec:
  filter:
    sourceAndType:
      type: dev.knative.cronjob.event
  subscriber:
    ref:
      apiVersion: serving.knative.dev/v1alpha1
      kind: Service
      name: event-display
```

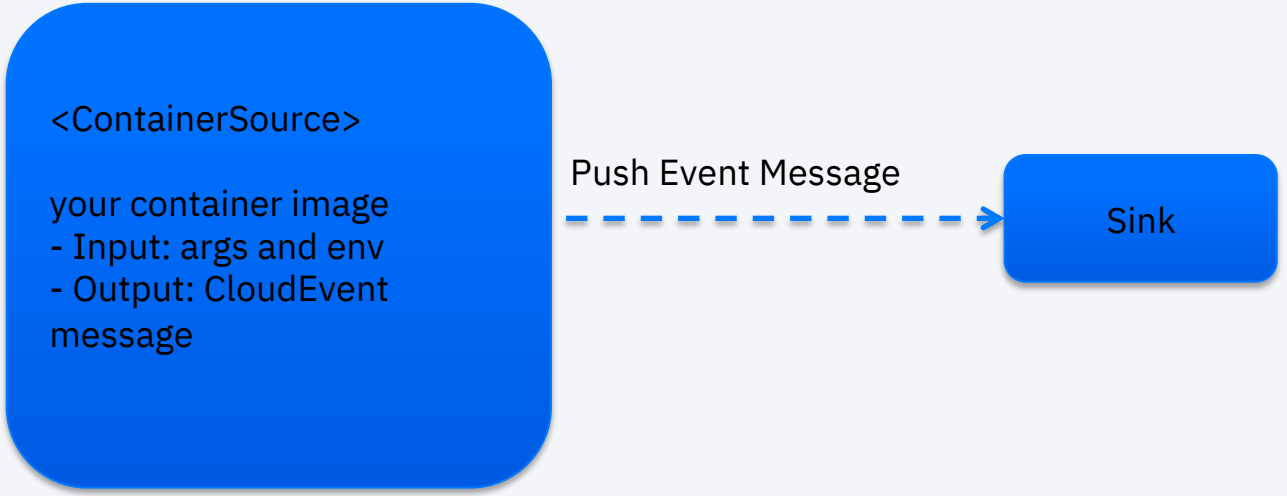
Broker和Trigger的使用

- Broker可以由管理员创建，设为默认。
- 最终用户只需要创建Trigger，通过”subscriber”属性指定事件消费者
- 最终用户通过”Filter”属性来指定事件消费者感兴趣的某种属性值。
- 事件生产者被屏蔽了，无需指定。
- 适用场景：用户只想关心高层次的抽象概念，而不想关心底层的实现细节。



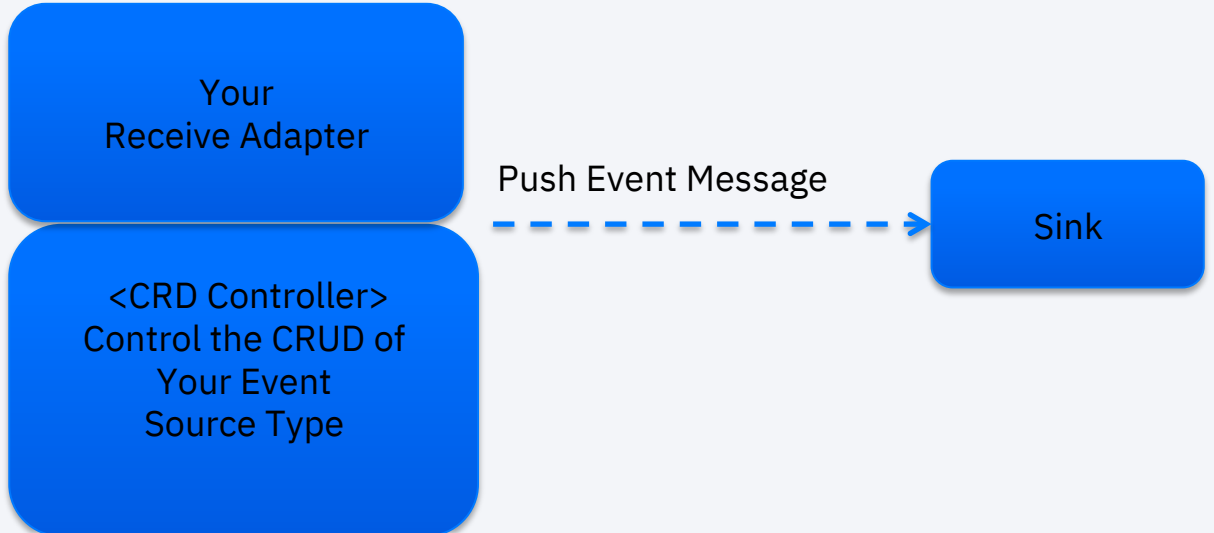
创建你的事件源 #1：使用ContainerSource

- 创建容器镜像，监听事件生产者，事件产生时把事件消息转化为CloudEvent格式发送给Knative事件平台。
- 将容器镜像上传到镜像库。
- 创建ContainerSource，指定使用该镜像。
- Knative事件平台会按照ContainerSource对象的生命周期来管理，创建对象时初始化容器镜像，删除对象时删除容器镜像。
- 例如监控某个FTP服务器，或者定时任务。



创建你的事件源 #2：使用Kubernetes CRD

- 创建接收适配器（Receive Adapter）的镜像，它需要监听事件生产者，在事件产生时把事件消息转化为CloudEvent格式发送给Knative事件平台。
- 按照Kubernetes自定义资源的框架，定义你的事件源对象。你需要创建 Kubernetes Resource 和 Kubernetes Controller来管理事件源对象的增删改查。
- 这是一种复杂的方式，利用了Kubernetes的扩展能力，事件源类型为K8s原生对象，用力处理复杂的事件场景。开发者需要了解Kubernetes CRD的框架。



回顾

- Eventing的目标和实现
- Eventing的资源和使用
- 创建你的事件源类型