Knative

汪润川

1 Knative

1.1 简述

传统的无服务计算框架存在着厂商绑定、缺乏行业标准的问题，为此，Google 联合 IBM 等公司推出了 Knative 无服务器计算框架，旨在提供一套简单易用的无服务器计算方案，把无服务器计算标准化。它以分布式容器 Kubernetes 为基础，并提供了缩容到零、自动扩缩、集群内构建以及事件框架等功能。Knative 于2018年7月推出，目前仍在快速发展阶段。

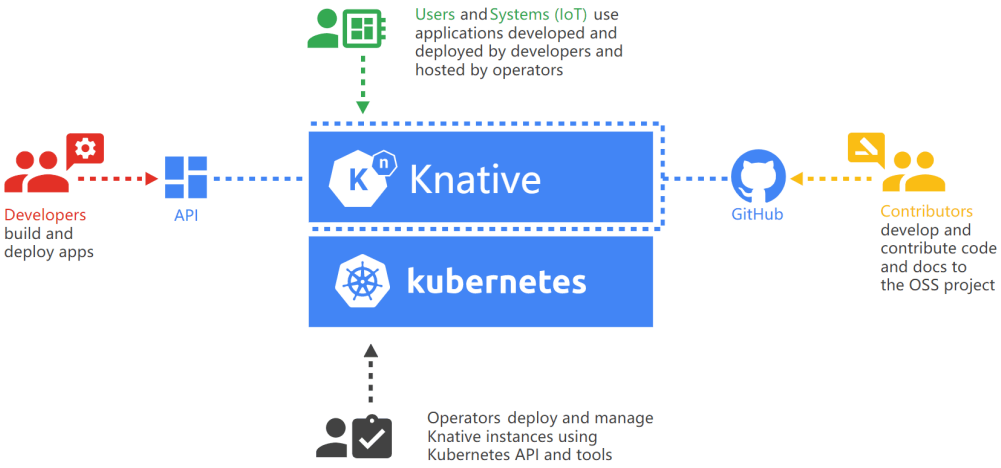

图 1: Knative 是基于 Kubernetes 的无服务计算扩展

Kubernetes（k8s）是自动化容器操作的开源平台，这些操作包括部署，调度和节点集群间扩展。Kubernetes 支持 Docker，Rocket 等容器技术。 Kubernetes 本身非常强大，但它十分复杂，需要理解并管理其丰富的资源，以及学习很多相关工具（yaml, kubectl, Istio...)。Kubernetes 的理念与云原生（Cloud Native）背道而驰。Knative 是一个简化版的 Kubernetes ，使得开发人员可以只专注于业务代码而非基础设施。当然，如果 Knative 不能满足开发人员的需求，Knative 背后的 Kubernetes 功能依然可以被直接使用。

下面简单介绍一些与 Knative 相关的 Kubernetes 概念。

Pod 是 Kubernetes 最小的调度单位，Pod 从属于 Node（物理机或虚拟机），Pod 中可以运行多个 Docker 容器。Pod 的设计理念是为了支持多个容器在一个Pod中共享网络和文件系统，处于一个 Pod 中的多个容器共享PID、IPC、Network 和 UTS 的命名空间。Pod 在创建时会被分配一个IP地址，Pod间的容器可以互相通信。Deployment 是 Pod 版本管理的工具，用来区分不同版本的 Pod。ReplicaSet 是副本控制器，使 Pod 副本数量始终维持在预设的个数。

Kubernetes 中的所有事物都被视为一个 API 对象并且都有一个与之对应的 API 入口。所有的操作和组件间的通信，包括外部的用户命令，都是由 API Server 处理的 REST API 调用。在 Kubernetes 中一切都可视为资源，系统提供了很多默认资源类型，如 Pod、Deployment等。一种资源就是 Kubernetes API 中的一个端点，它存储着某种 API 对象的集合。自定义资源（CRD）是对 Kubernetes API 的扩展，在一个运行中的集群内，自定义资源可以通过动态注册出现和消失，集群管理员可以独立于集群本身更新自定义资源。

Knative 包含2个核心组件： Serving 和 Eventing。Serving 提供无服务器应用或函数的部署能力以及各种服务管理，底层采用 Kubernetes 的 Pods；Eventing 管理进入到环境中的事件，提供事件触发的通道。事件不仅要被送达到相应的服务，也要被持久化到某些队列中去，以适应目标服务当前不可用的情况。同时，Knative 还依赖于 CI/CD 流水线 Tekton 完成从代码到部署的过程。

1.2 核心组件

1.2.1 Tekton

早期 Knative 采用一个专门的 Build 组件解决从源代码构建容器的问题，build 完全基于 Kubernetes 生态，用 Kubernetes CRD 的方式实现。

Build 现已不再维护和推荐使用，当前推荐的构建组件是使用持续集成和持续部署（CI/CD）流水线 Tekton Pipelines。Tekton 以自定义资源的形式提供了一组 Kubernetes 扩展，用于定义流水线。图2展示了 Tekton Pipelines 的 CRD，包括：

- PipelineResource 定义了一个对象，该对象是流水线的输入（例如 git 存储库）或输出（例如 docker 镜像）。
- PipelineRun 定义了流水线的执行。它引用要运行的 Pipeline 以及要用作输入和输出的 PipelineResources。
- Pipeline 定义了构成流水线的 Tasks。
- Task 定义了一组构建步骤，如编译代码、运行测试以及构建和部署镜像，相当于一系列模板。

图3展示了将一个应用从源码构建成容器镜像，然后再部署到 Knative 环境上的过程。这个过程包括 Build 和 Deploy 两个任务，这两个任务通过一个 Pipeline 封装。 PipelineResource 包括了应用的源码，以及如何构建和部署改应用。 Build 阶段， kaniko 会根据 Dockerfile 将源码构建成一个镜像，并上传到 Docker Registry 上。容器构建完成后， Build 会将

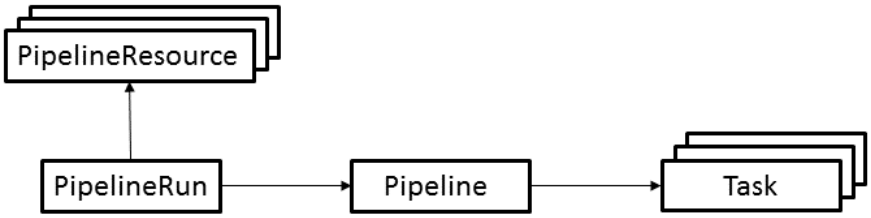

图 2: Tekton Pipeline 的CRD

镜像名、版本等信息传给 Pipeline 下游的任务。第二个任务同样从 PipelineResource 中获取部署的文件，利用 kuberctl 完成部署。

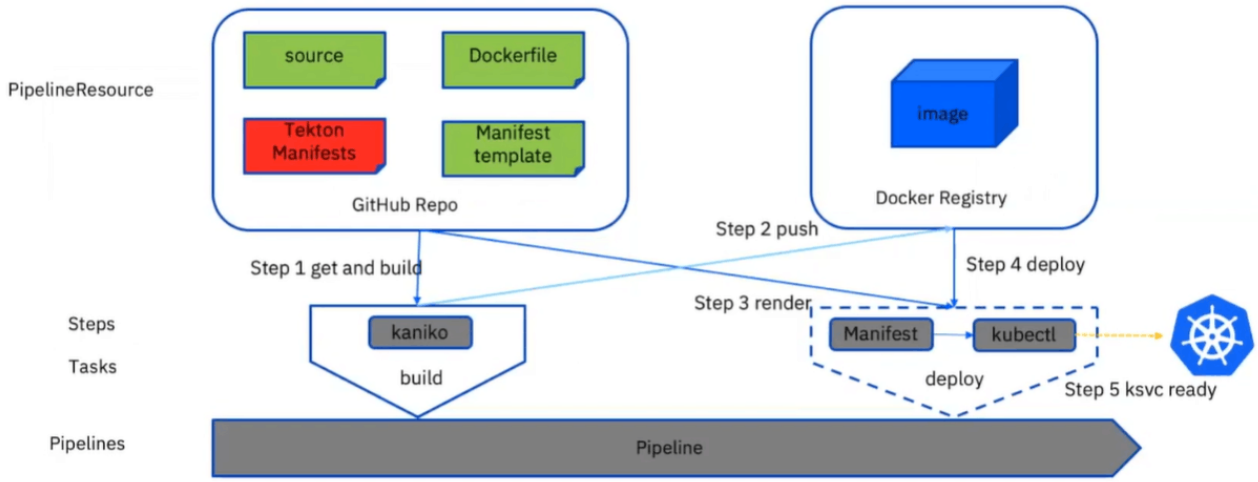

图 3: 从源码构建容器镜像并部署

1.2.2 Serving

Serving 提供了无服务器应用或函数的部署能力，以及自动缩扩容、版本管理、流量控制、滚动升级等功能。Serving 的基本结构如图4所示，它提供四个主要的 API：

- Route: 定义网络端口，映射一个或多个 Revision，将流量按比例导入不同的 Revision。
- Revision: 应用的旧版本，每次修改代码或配置的快照，可以根据进入的流量自动扩缩容。
- Configuration: 维护应用的最新配置，每次修改 Configuration 会产生一个新的 Revision。
- Service: 管理应用的整个生命周期，确保应用拥有 Configuration 和 Route。Service 可以被看作是正在部署的应用或者函数。

当需要进行滚动升级时，Route 会将所有流量逐渐路由到最新的版本中，而旧版本最终会因为没有流量而缩容至0。

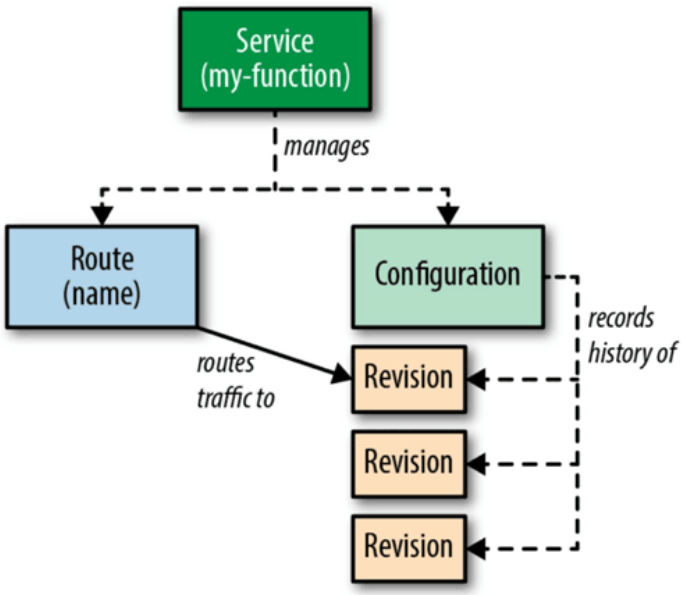

图 4: Serving 基本结构

Knative 的自动缩容扩容功能需要 Autoscaler 和 Activator 两个组件完成，如图5所示。Reversion 中的 pod 会自动汇报 metrics 数据到 autoscaler，autoscaler 会根据请求量和资源

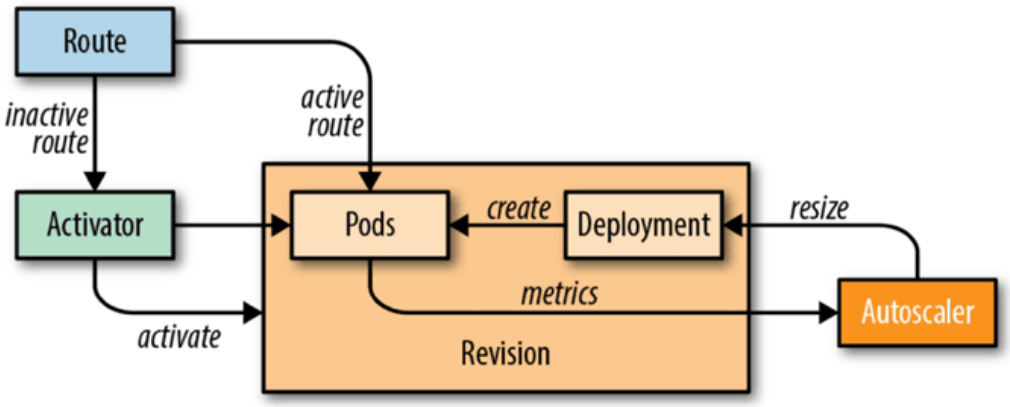

图 5: Autoscaler 实现自动缩容扩容

使用情况修改 deployment 的副本（replicas）数量，从而实现自动扩缩容。

Revision 处于激活状态才接受请求。当一个 Revision 停止接受请求时，Autoscaler 将其置为待命状态。处于待命状态下，一个 Revision 底层部署缩容至零并且所有到它的流量均路由至 Activator。Activator 是一个共享组件，其捕获所有到待命 Revision 的流量。当它收到某一待命 Revision 的激活请求后，它转变 Revision 状态至激活，然后代理请求至合适的 Pod。

1.2.3 Eventing

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。特征是松耦合、事件生产者和消费者互相独立、支持第三方服务接入、支持跨平台和互操作性。

Eventing 定义了一组 Kubernetes CRD，包括

- Event Source: 用于把事件生产者接入 Knative 事件平台，并把事件传送给消费者。
- Channel: 实现事件的转发和存储。底层实现可以有 Kafka, NATS Streaming 等。
- Subscription: 事件的订阅者，即事件转发的目的地。
- Parallel: 事件同时转发给多个订阅者，每个订阅者接收到同一个事件。
- Sequence: 事件依次经过多个订阅者，每个订阅者都可以修改，过滤或者创建新的事件。
- Broker: 可以接收事件，并将事件根据过滤条件转发给订阅者。
- Trigger: 根据订阅者的要求对事件设置过滤条件过滤。
- Event Registry: 用于查阅 Broker 中的事件类型。

Sink 是 Eventing 中一个重要概念，表示事件消息传送的目的地，也就是事件接收者的一个 HTTP 端口。

一个简单的事件订阅流程如图6所示。 Event Source 定义了事件源，通过 Sink 指定事件消费者的地址。 Knative 规定传输的事件必须符合 CloudEvents 格式，因此在 Event Source 后还有一个 Adapter（图中未画出）进行格式转换。为了适应单个事件源对多个订阅者的场景，需要使用 Channel 和 Subscription。 Channel 具有存储功能，可以避免消费者因暂时不在线而导致的消息丢失。 Subscription 连接 Channel 和消费者。消费者可以通过 Subscription 接收消息，也可以向 Subscription 发送回复，并由 Subscription 向其他 Channel 转发。基于 Channel 和 Subscription ，可以实现 Parallel 和 Sequence。

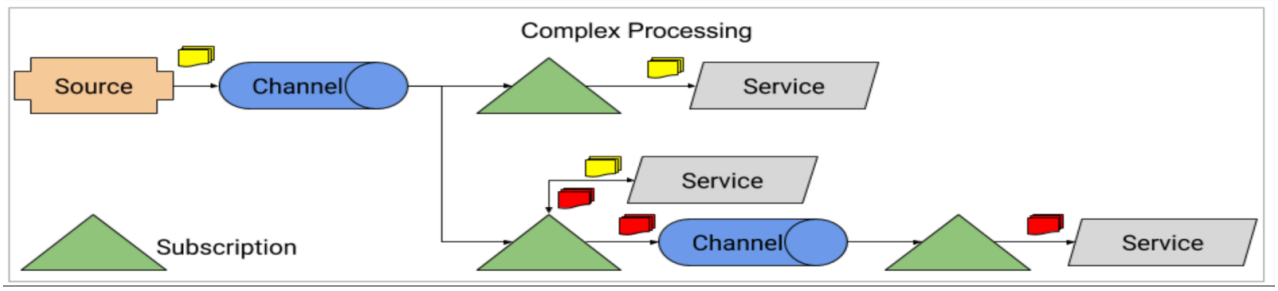

图 6: 通过 Channel 和 Subscription 实现事件的转发和订阅

在较新的 Knative 版本中，Eventing 增加了 Broker 和 Trigger 对象，目的是搭建一个黑盒，将具体的实现隐藏起来，减少需要用户添加的 Channel 和 Subscription。同时， Broker

和 Trigger 可以满足消费者对信息过滤的需求，消费者可以选择订阅自己感兴趣的事件，而非接收 Channel 中的所有信息。目前，Eventing 支持的过滤指标包括 Type（事件类型）和 Source （事件来源），将来还会支持更多的筛选规则。（TODO：找一下最新的筛选规则）

消费者此时不需要创建 Subscription 对象，而是创建一个 Trigger 对象，描述自己感兴趣的事件。Broker 是一个事件桶，接收各种事件，根据 Trigger 中定义的规则，对事件进行过滤，过滤之后发送给事件的订阅方。事件的转发和订阅结构如图7所示。

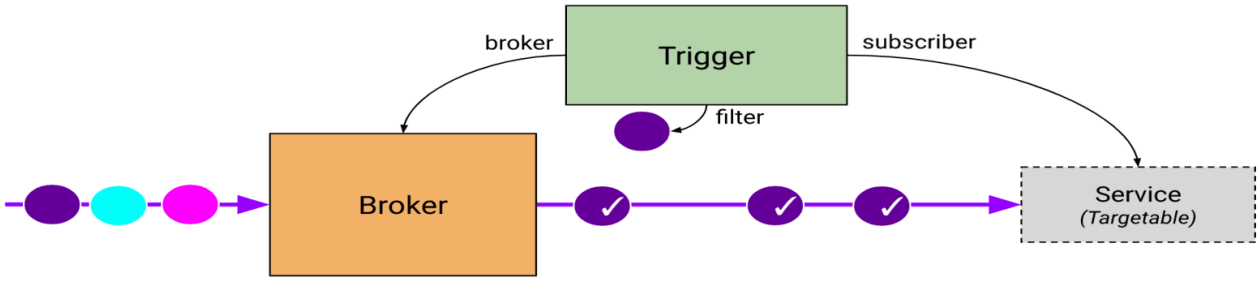

图 7: 通过 Broker 和 Trigger 实现事件的转发和订阅。订阅者告诉 Trigger 只要紫色的消息。

2 参考

Kubernetes 文档
Knative 文档
IBM开源技术微讲堂 KNative系列（视频）
IBM开源技术微讲堂 KNative系列（讲义）
IBM Tekton Pipelines 文档
Knative入门——构建基于 Kubernetes 的现代化Serverless应用