developerWorks 中国 正在向 IBM Developer 过渡。 我们将为您呈现一个全新的界面和更新的主题领域，并一如既往地提供您希望获得的精彩内容。

Knative 是什么？为什么您需要关注它？

Gregory Haynes
2019 年 3 月 06 日发布

最近有许多关于无服务器、Kubernetes 和 Knative 的讨论。让我们先来解释一下 Knative 在这个生态系统中的适用位之处。Knative 非常适合已经使用 Kubernetes 的应用程序开发者，可为他们提供一些工具，让他们减少对基础架构和注，而将更多精力集中在编写的代码上。这一使命与其他无服务器平台没什么太大差别，但是大多数无服务器平台都下的方法来提供以代码为中心的新界面，而 Knative 则专注于构建这些工具和服务来提升现有的 Kubernetes 体验。长的开发者（Kubernetes 用户）群体带来了即时利益以及轻松的无服务器开发体验。为此，Knative 使用与 Kubern同的模式（控制器）、API (kube-api) 和 Kubernetes 基础架构（Kubernetes 资源）构建而成。Knative 还提供"缩容支持真正零成本使用空闲应用程序，并支持采用蓝/绿部署来测试无服务器应用的新版本。

参考资源

评论

Knative 组件

Knative 包含三个主要子项目：

- **Serving** 提供缩容至零、请求驱动的计算功能。它本质上是无服务器平台的执行和扩展组件。
- **Build** 提供显式"运行至完成"功能，这对创建 CI/CD 工作流程很有用。Serving 使用它将源存储库转换为包含应用
  镜像。
- **Eventing** 提供抽象的交付和订阅机制，允许构建松散耦合和事件驱动的无服务器应用程序。

内容

重要的是，Knative 旨在将这些组件（以及更多子组件）松散耦合。这意味着服务提供商和用户可以根据需要混合、
概览 这些组件。例如，虽然 Serving 组件可以使用 Build 将您的源存储库转换为无服务器应用程序，但技术高超的用户也
己的 Build 机制，该机制使用另一个构建平台，但仍使用 Knative 的 Serving 组件。

Knative 组件

Serving 组件

Serving 组件

Hello, World!

在本文中，我们将重点关注 Serving 子项目，因为它是深入了解 Knative 自然起点。 Knative Serving 用户应该熟悉以
Service（服务） 资源： Service（服务）、 Route（路由）、Configuration（配置）和 Revision（修订）。

Configuration（配置）

Revision（修订）

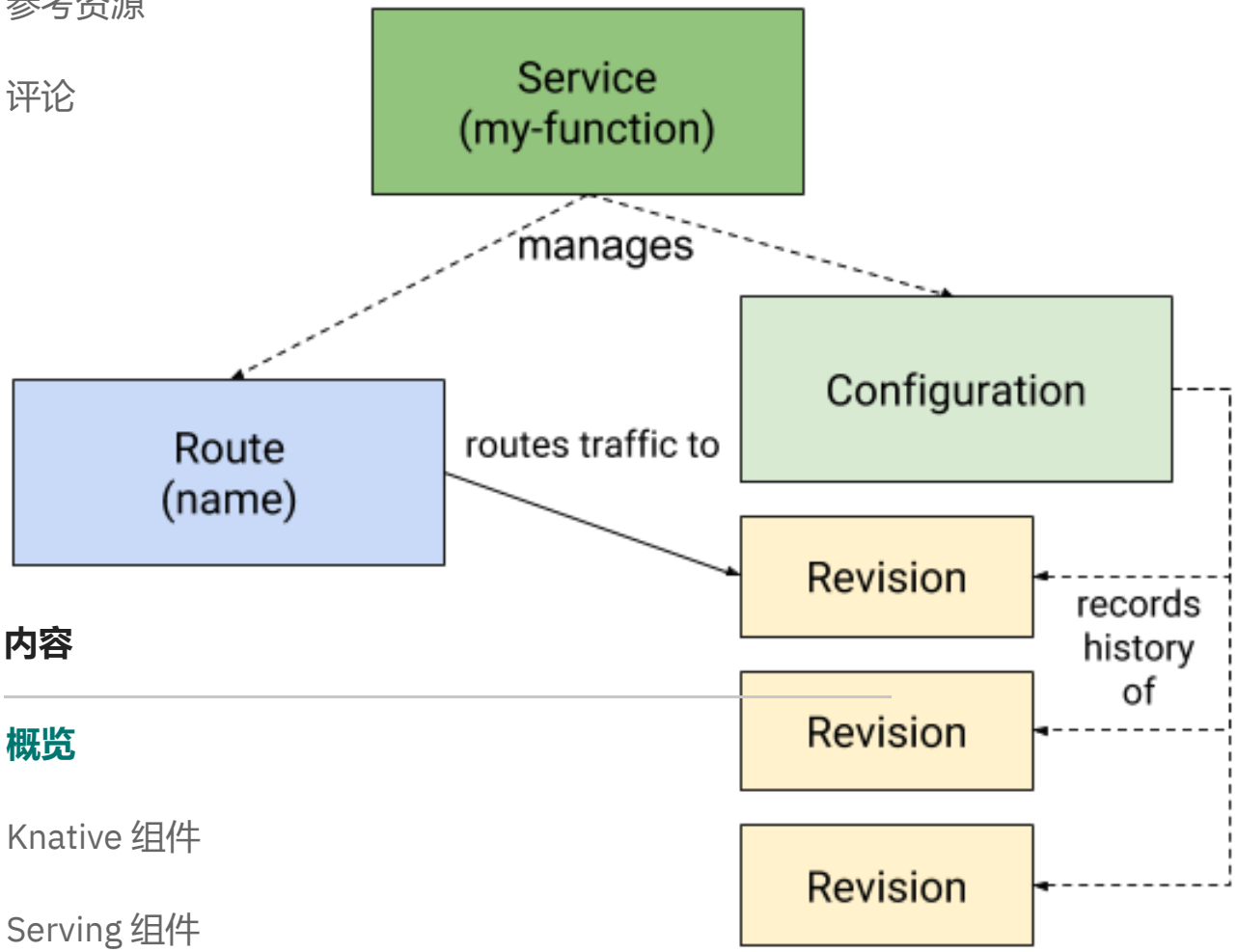
Route（路由）

结束语

参考资源

参考资源

评论



内容

概览

Knative 组件

Serving 组件

Source: https://github.com/knative/docs/tree/master/serving

Hello World

Service（服务）

Configuration（配置）

Revision（修订）

Route（路由）

结束语

参考资源

Revision：这代表一个您的应用程序的单个实例。它以不可变形式包含一个特定的镜像和一组特定的配置选项（如环
它负责管理资源来实现包括运行应用程序的 Kubernetes 部署。

Configuration：这是用于创建修订的界面，因此大多数应用程序生命周期管理都通过此资源进行。将此视为您的部署
负责定义应用程序镜像及其配置，类似于修订，但这些值是可变的。这允许更新配置中的环境变量或镜像标记，以便
本。每当更改这些值时，都会创建一个应用程序的新实例（修订）。因此，给定配置始终至少有一个修订。

参考资源
Route：通过此资源将流量引到一个特定修订。将一个服务用于简单部署时，通常不需要注意此资源，因为它在默认
评论
有流量发送到最新修订。用户还可以使用此资源按百分比指定流量拆分（例如，a/b 部署，其中 10% 的流量将流向
的流量将流向修订 1）。本文不将介绍此话题，但您可以在以下网址中找到此功能的演示：
https://github.com/knative/docs/tree/master/serving/samples/traffic-splitting。

(Knative) Service：不要与 Kubernetes 服务资源混淆，此 Knative 服务是将完整的无服务器应用程序连接在一起的
源。对于简单的用法（例如下面的 hello-world 示例），这是用户在部署其应用程序时需要与之交互的唯一资源。创
还会创建路由和配置，下面将对此详加说明。

Hello, World!

内容

概览
假设您正常安装了 Knative，并可使用 kubectl 来访问 Knative 集群。（访问
https://github.com/knative/docs/blob/master/install/README.md，获取有关安装 Knative 的更多信息。）
Knative 组件
让我们使用 kubectl，通过将以下内容写入文件 (service.yaml) 并运行，定义以下 Knative 服务：
Serving 组件

```
$ kubectl -f service.yaml

apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: helloworld-go
  namespace: default
spec:
  runLatest:
    configuration:
      revisionTemplate:
        spec:
          container:
            image: <helloworld-go docker image>
            env:
```

```
            - name: TARGET
              value: "Go Sample v1"
```

这就是我们创建一个 helloworld-go 应用程序的请求驱动实例所需的全部内容。

通过从以下命令中复制 EXTERNAL IP 字段，查找可以访问您的应用程序的 IP 地址：

```
$ kubectl get svc knative-ingressgateway --namespace istio-system

NAME                     TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
knative-ingressgateway   LoadBalancer   10.23.247.74   35.203.155.229   80:32380/TCP,443:32390/TC
```

然后使用以下命令查找应用程序的域，复制 DOMAIN 字段：

```
$ kubectl get ksvc helloworld-go

NAME            DOMAIN                                LATESTCREATED         LATESTREADY           R
helloworld-go   helloworld-go.default.example.com   helloworld-go-00001   helloworld-go-00001   T
```

Service（服务）

接着，可以使用以下命令向应用程序发出请求：

Configuration（配置）

```
$ curl -H "Host: {DOMAIN}" http://{IP_ADDRESS}

Hello World: Go Sample v1!
```

结束语

恭喜！您刚刚成功地部署了第一个 Knative 无服务器应用程序！

参考资源

参考资源

Service（服务）

评论

现在，让我们来看看应用程序的构成资源。先查看定义时所使用的服务：

```
$ kubectl get ksvc helloworld-go

NAME            DOMAIN                                  LATESTCREATED         LATESTREADY           R
helloworld-go   helloworld-go.default.example.com   helloworld-go-00001   helloworld-go-00001   T
```

这向我们展示了应用程序的域，以及应用程序最新创建的修订和已有的修订。由于这是我们最初定义的确切资源，在

内容

内容并不多，因此让我们深入了解一下 Knative 为我们创建的一些其他资源。

概览

Knative 组件

Configuration（配置）

Serving 组件

在我们定义 Knative 服务时，会自动为我们的应用程序创建一个配置。我们可以使用以下命令查看此配置：

Hello, World!

```
$ kubectl get configuration helloworld-go -o yaml

apiVersion: serving.knative.dev/v1alpha1
kind: Configuration
metadata:
  name: helloworld-go
  namespace: default
  <other_metadata>
spec:
  generation: 1
  revisionTemplate:
    metadata:
      annotations:
```

```
        sidecar.istio.io/inject: "false"
      creationTimestamp: null
    spec:
      container:
        env:
        - name: TARGET
          value: Go Sample v1
        image: <image_url>/helloworld-go
        name: ""
        resources: {}
      containerConcurrency: 1
      timeoutSeconds: 1m0s
status:
  conditions:
  <conditions>
  latestCreatedRevisionName: helloworld-go-00001
  latestReadyRevisionName: helloworld-go-00001
  observedGeneration: 1
```

概览

为便于阅读，已删除了部分输出。如您所见，根据我们发出的命令，此配置的名称 (helloworld-go) 与我们所定义的
Knative 组件
配 - 定义服务时总是如此。最有趣的组件是 `spec.revisionTemplate` 部分。

Serving 组件

Revision（修订）

Hello World
Service（服务）

运行以下命令来查看由我们的配置所创建的修订：

Configuration（配置）

```
$ kubectl get revision helloworld-go-00001 -o yaml

apiVersion: serving.knative.dev/v1alpha1
kind: Revision
metadata:
  name: helloworld-go-00001
  namespace: default
  <metadata>
```

```
  spec:
    container:
      env:
      - name: TARGET
        value: Go Sample v1
      image: <image_url>
      name: ""
      resources: {}
    containerConcurrency: 1
    generation: 1
    timeoutSeconds: 1m0s
  status:
    conditions:
    <conditions>
    serviceName: helloworld-go-00001-service
```

内容

与上文配置一样，为便于阅读，我删除了一些修订输出。如前所述，每次修改配置时，都会创建一个新的修订，同时配置创建初始修订。由于这是第一个修订，因此它的修订号为 00001。您应该也注意到，修订规范与上述配置中的 revisionTemplate 相匹配。

现在让我们尝试更改配置（通过我们的服务），并观察新修订的创建。为此，我们可以将 `service.yaml` 文件中的行 `value: "Go Sample v1"` 更改为 `value: "Go Sample v2"`。

然后使用以下命令来更新服务：`kubectl apply -f service.yaml`。

让我们使用以下命令来查看当前修订：

```
$ kubectl get revision

NAME                   SERVICE NAME                   READY     REASON
helloworld-go-00001    helloworld-go-00001-service    True
helloworld-go-00002    helloworld-go-00002-service    True
```

参考资源

如您所见，我们创建了一个新修订 (helloworld-go-00002)。重新运行 curl 命令我们还可以看到应用程序的响应已发

评论

```
$ curl -H "Host: {DOMAIN}" http://{IP_ADDRESS}

Hello World: Go Sample v2!
```

Route（路由）

现在运行以下命令查看 Knative 为我们的服务创建的路由：

```
$ kubectl get route helloworld-go -oyaml

apiVersion: serving.knative.dev/v1alpha1
kind: Route
metadata:
  name: helloworld-go
  namespace: default
  <other metadata>
spec:
  generation: 1
  traffic:
  - configurationName: helloworld-go
    percent: 100
status:
  address:
    hostname: helloworld-go.default.svc.cluster.local
  <conditions<
  domain: helloworld-go.default.example.com
  domainInternal: helloworld-go.default.svc.cluster.local
  traffic:
  - percent: 100
    revisionName: helloworld-go-00002
```

参考资源

如您所见，与我们的其他资源相比，此资源的 spec 部分相当简略，因为它只包含一个 generation 和 traffic 部分。用 traffic 部分指定配置（就像我们此时的配置一样），这始终会将流量引到最新就绪的修订或一个特定修订。在定位一时，值得注意的是，您可以指定修订列表 - 每个修订都有各自的目标百分比，然后允许用户逐步更改，最初仅将一小送到一个新修订。

结束语

在本文中，您了解了如何部署一个简单的 helloworld 无服务器应用程序，该应用程序完全根据请求负载进行扩展。您成 Knative Serving 项目的组件，以及如何使用 `kubectl` 检查基本构件。本文仅对 Knative 进行了初步的探讨，我鼓励 Knative docs 存储库中查询更多信息：https://github.com/knative/docs。 如果您有兴趣施展自己的 Knative 新技能，

内容

在云中自动化 Knative 安装。

概览

参考资源

- GitHub 上的 Knative
- IBM Cloud 与 Google 及开放社区合作，帮助构建 Knative 并扩展无服务器的强大功能
- 在 IBM Cloud 上使用 Istio 安装 Knative
- Knative 101：通过几个练习来了解 Knative：本教程简要概述了 Knative 及其各种功能和组件，您可以通过实验来深入了解这个开源平台。

本文翻译自：Knative: What is it and why should you care? （2019-01-28）