

# Knative: 重新定义Serverless

敖小剑 @ 蚂蚁金服 中间件





敖小剑 / Sky Ao

资深码农，十六年软件开发经验，微服务专家，Service Mesh布道师，Servicemesh社区联合创始人。专注于基础架构，Cloud Native 拥护者，敏捷实践者，坚守开发一线打磨匠艺的架构师。

曾在亚信、爱立信、唯品会等任职，对基础架构和微服务有过深入研究和实践。

目前就职蚂蚁金服，在中间件团队从事Service Mesh、Serverless等云原生产品开发。

# 1

1

Knative是什么?

2

Knative主要组件

3

Knative分析与探讨

4

Knative后续发展

5

总结



Knative 是谷歌牵头发起的 Serverless 项目，希望通过提供一套简单易用的 Serverless 开源方案，把 Serverless 标准化。

项目地址：<https://github.com/knative>




**Kubernetes**-based platform to **build**, **deploy**, and **manage** modern **serverless** workloads

基于**Kubernetes**平台，用于**构建**，**部署**和**管理**现代**serverless**工作负载



Google



IBM



Pivotal



redhat.



## ✓ 目前Release情况

- 2018-11-07 v0.2.1 版本发布
- 2018-10-31 v0.2.0 版本发布
- 2018-08-14 v0.1.1 版本发布
- 2018-07-19 v0.1.0 版本发布

非常新  
处于早期发展阶段

## ✓ 云端Serverless

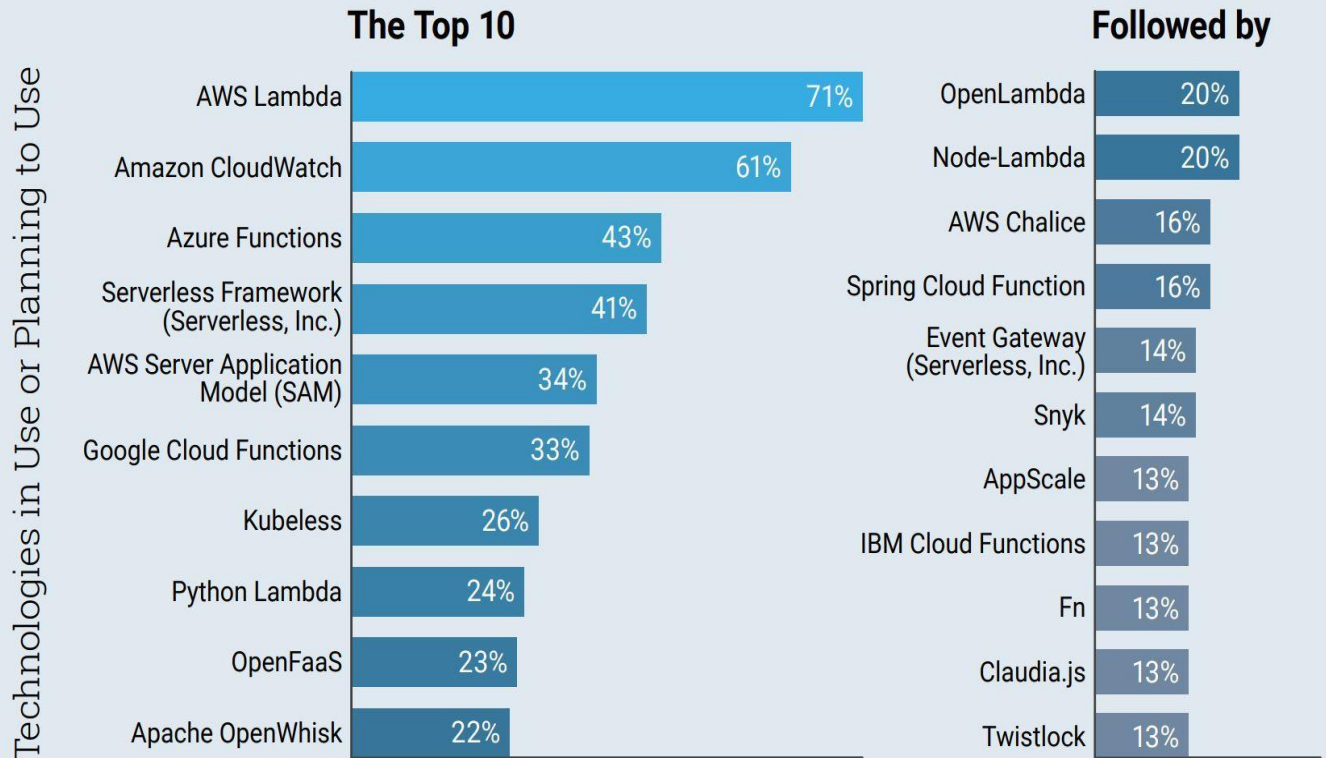
- AWS Lambda
- Google Cloud Functions
- Microsoft Azure Functions
- IBM Cloud Functions
- .....

## ✓ 开源项目

- Iron.io
- kubeless
- Riff
- Fission
- OpenFaaS
- Apache OpenWhisk
- Spring Cloud Functions
- Lambda Framework
- WebTask
- .....



# Top Technologies on Serverless Roadmaps for Next 18 Months

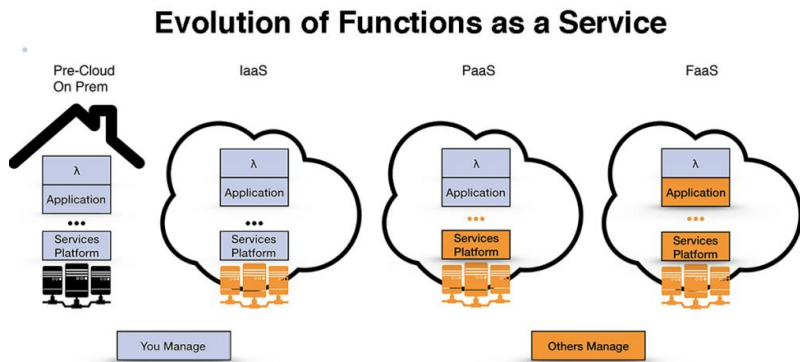


% of Respondents Using or Planning to Use Serverless Architecture

# 风险：提供商绑定！

✓ 每个云厂商，每个开源项目，都各不相同：

- 代码到容器的构建 (Build)
  - 函数的定义和编写方式
  - 构建镜像，部署函数的方式
- 函数触发的方式 (Eventing)
  - 事件格式
  - 事件和函数的绑定方式
  - 订阅/发布机制
- 运行时管理能力 (Serving)
  - 网络路由
  - 流量控制
  - 升级策略
  - 自动伸缩



缺乏标准，市场呈现碎片化

- ✓ 提供通用型工具以帮助开发人员在Kubernetes上构建Function
- ✓ 帮助云服务供应商及企业平台运营商为任何云环境中的开发人员提供Serverless体验
- ✓ 提供整合的平台，将Kubernetes、Serverless和ServiceMesh结合在一起
- ✓ 多云战略，不会被某个云提供商锁定，可在不同FaaS平台之间移植

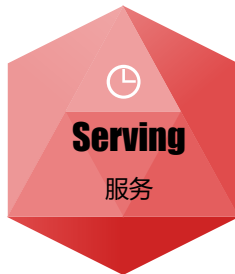
## 构建容器和部署负载

- 在 kubernetes 上编排 source-to-url 的工作流程
- 提供标准化可移植的方法
- 定义和运行集群上的容器镜像构建



## 为工作负载提供服务

- 请求驱动计算，可以扩展到零
- 根据需求自动伸缩和调整工作负载大小
- 使用蓝绿部署路由和管理流量



## 事件驱动

- 管理和交付事件
- 将服务绑定到事件
- 对发布/订阅细节进行抽象
- 帮助开发人员摆脱相关负担



# 2

1

Knative是什么?

2

**Knative主要组件**

3

Knative分析与探讨

4

Knative后续发展

5

总结



蚂蚁金服  
ANT FINANCIAL

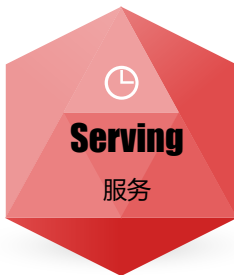
## Build

- 在 kubernetes 上编排 source-to-url 的工作流程
- 提供标准化可移植的方法
- 定义和运行集群上的容器镜像构建



## Serving

- 请求驱动计算，可以扩展到零
- 根据需求自动伸缩和调整工作负载大小
- 使用蓝绿部署路由和管理流量



## Eventing

- 管理和交付事件
- 将服务绑定到事件
- 对发布/订阅细节进行抽象
- 帮助开发人员摆脱相关负担



标准化，可替代，松散组合，不绑定

## ✓ Why not dockerfile?

### 1. 目标不同

- Source-to-image
- Source-to-url

### 2. 构建环境

- Knative的构建是在k8s中进行，和k8s生态结合
- 扩展了Kubernetes并利用现有的Kubernetes原语

### 3. 高度不同

- Knative Build的目标是提供标准的，可移植的，可重用的而且性能优化的方法，用于定义和运行集群上的容器镜像构建。
- 可以作为更大系统中的一部分

## ✓ Build是Knative中的自定义资源(CRD)

- 可以通过 yaml 文件定义构建过程

## ✓ 关键特性

- Build 可以包括多个步骤，而每个步骤指定一个 Builder.
- Builder 是一种容器镜像，可以创建该镜像来完成任何任务
- Build中的步骤可以推送到仓库。
- BuildTemplate可用于定义可重用的模板。
- 可以定义Build中的source以装载数据到 Kubernetes Volume，支持git仓库
- 通过ServiceAccount来使用Kubernetes Secrets进行身份验证。

```
apiVersion: build.knative.dev/v1alpha1
kind: Build
metadata:
  name: example-build
spec:
  serviceAccountName: build-auth-example
  source:
    git:
      url: https://github.com/example/build-example.git
      revision: master
  steps:
    - name: ubuntu-example
      image: ubuntu
      args: ["ubuntu-build-example", "SECRETS-example.md"]
  steps:
    - image: gcr.io/example-builders/build-example
      args: ['echo', 'hello-example', 'build']
```



- ✓ 定义为: Kubernetes-based, scale-to-zero, request-driven compute
- ✓ 以Kubernetes和Istio为基础, 支持serverless应用程序和功能的部署与服务
- ✓ Knative Serving项目提供了中间件原语:
  - Serverless容器的快速部署
  - 自动伸缩, 支持扩容到零
  - Istio组件的路由和网络编程
  - 已部署代码和配置的时间点快照

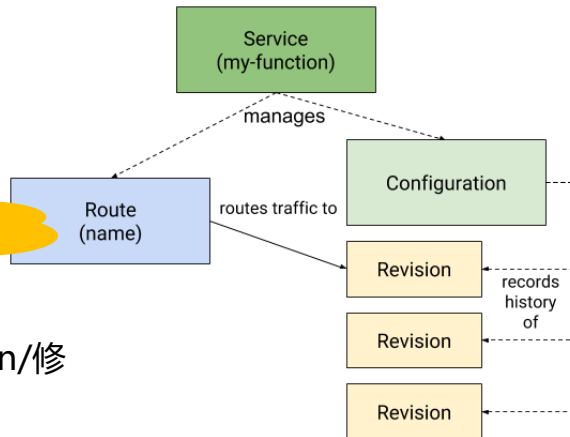
## ✓ 背景

- kubernetes 和 istio 本身的概念非常多
- 理解和管理，比较困难
- knative 提供了更高一层的抽象
- 基于 kubernetes 的 CRD 实现

## ✓ 抽象概念

- Service: 自动管理工作负载的整个生命周期
- Route: 将网络端点映射到一个或多个revision/修订版本
- Configuration: 维护部署所需的状况
- Revision: 每次对工作负载进行代码和配置修改的时间点快照

service.serving.knative.dev  
不是k8s的service



## ✓ 伸缩界限

- `autoscaling.knative.dev/minScale: "2"` # 默认为0
- `autoscaling.knative.dev/maxScale: "10"` # 默认没有上限

## ✓ Autoscaler

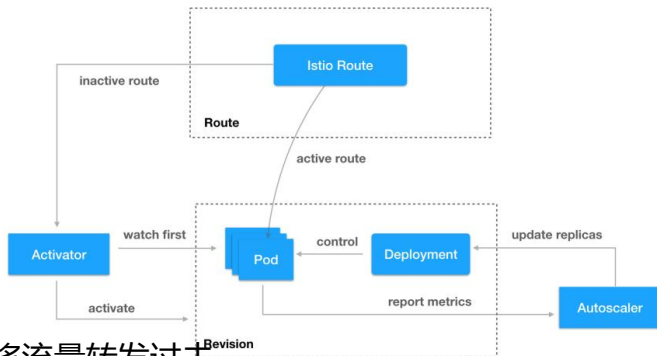
- Revision对应一组pod, 由Deployment管理
- Pod上报metrics到autoscaler
- Autoscaler分析判断, 修改replicas数量

## ✓ Activator

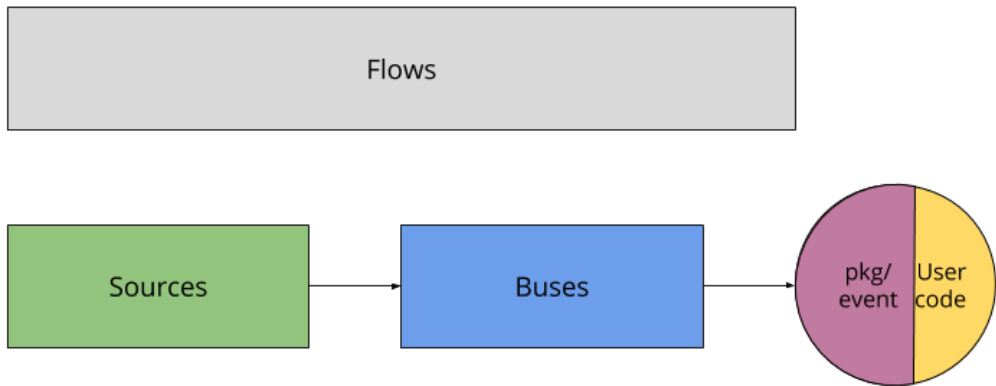
- 处理scale to zero场景
- 缩容到0时, Route流量切向Activator
- 有新请求时, Activator拉起pod, 然后将流量转发过去

## ✓ Route

- 对应Istio的路由规则, 如DestinationRoute 和 VirtualService
- 决定流量的路由方式



# Knative Eventing: 事件绑定和发送



Eventing的核心功能：对发布/订阅细节进行抽象处理，帮助开发人员摆脱相关负担。

## ✓ 抽象

- 总线通过 NATS 或 Kafka 等消息总线提供k8s原生抽象
- **Channel** 是网络终端，它使用特定于总线的实现来接收（并可选地持久化）事件
- **Subscription** 将在channel上收到的事件连接到感兴趣的目标，表示为DNS名称。
- **Bus** 定义了使用特定持久化策略实现channel和subscription所需的适配层

## ✓ 目前实现了3个Bus

- Stub 提供无依赖的内存传输
- Kafka 使用现有（用户提供的）Kafka集群来实现持久性
- GCP PubSub 使用Google Cloud PubSub来实现消息持久性

## ✓ 抽象

- **Source** 是抽象的数据源
- **Feed** 是一个原始对象，用于定义 EventType 和操作之间的连接
- **EventType** 和 **ClusterEventType** 描述了一组具有由 EventSource 发出的通用模式的特定事件
- **EventSource** 和 **ClusterEventSource** 描述了可能产生一个或多个 EventTypes 的外部系统

## ✓ 目前实现了3个Source

- K8sevents 收集Kubernetes Events并将它们呈现为CloudEvents。
- GitHub 收集 pull request 通知并将其表示为CloudEvents。
- GCP PubSub 收集发布到 GCP PubSub topic的事件，并将它们表示为CloudEvents

## ✓ 背景

- serverless 平台和产品众多
- 支持的事件来源和事件格式定义五花八门
- Knative和CNCFT试图对事件进行标准化

## ✓ CloudEvents 介绍

- CloudEvents是一种以通用方式描述事件数据的规范。
- CloudEvents旨在简化跨服务，平台及其他方面的事件声明和发送
- CloudEvents 最初由 CNCF Serverless 工作组提出。

## ✓ CloudEvents 状态

- <https://cloudevents.io/>
- <https://github.com/cloudevents/spec>
- 2018年4月，发布了v0.1版本
- 从刚刚结束的 Kubeconf 上海站得知，v0.2即将发布



cloudevents

# 3

1

Knative是什么?

2

Knative主要组件

3

**Knative分析与探讨**

4

Knative后续发展

5

总结



蚂蚁金服  
ANT FINANCIAL





**Knative 不是一个Serverless实现，而是一个Serverless平台。**

**Implement → Platform**



## 工作负载

---

和标准化的 FaaS 不同, knative 期望能够运行所有的工作负载:

- Function
- Microservice
- Traditional Application
- Container

## 平台支撑

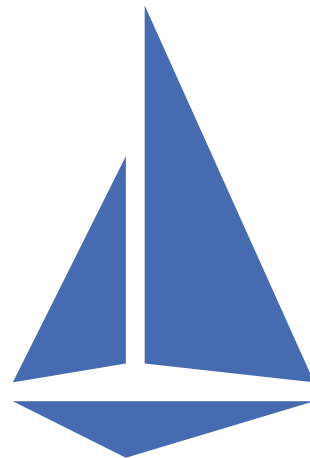
---

Knative 建立在 kubernetes 和 istio 之上

- 使用 kubernetes 提供的容器管理能力
  - Deployment
  - Replicaset
  - Pods
- 使用 istio 提供的网络管理功能
  - Ingress
  - Load balance
  - Dynamic Route

这两点, 是knative最吸引我们的地方

- ✓ 背景
  - 基于 kubernetes 的 serverless 产品非常多
  - 基于同时又基于 istio, knative 是第一个
- ✓ 存在普遍质疑
  - 真的有必要基于 istio 来做吗?
  - Kubernetes很复杂, knative也很复杂
  - Istio的复杂度会让整个系统的复杂度再上升一个台阶
- ✓ 我们的分析
  - Istio 的地位已定
  - Serverless + Servicemesh on Kubernetes 组合很强大



## ✓ 复杂度很高

- Kubernetes 复杂度
- Istio 的复杂度
- Knative 的复杂度

## ✓ 挑战

- 学习掌握、构建维护、运维调试很复杂
- 需要了解的概念和抽象非常多（上百个）
- 落地过程中会遇到的各种问题
- 对开发团队，运维团队挑战很大

# 4

1

Knative是什么?

2

Knative主要组件

3

Knative分析与探讨

4

**Knative后续发展**

5

总结



蚂蚁金服  
ANT FINANCIAL

## ✓ 背景

- 性能问题一直是 Serverless 被人诟病的重点
- 也是目前应用不够广泛的决定性因素之一
- Serverless 整个网络链路偏长
- 扩容时容器拉起需要时间，尤其从0到1会很明显

## ✓ 改进方向

- 1到N 的自动伸缩：如fast forking技术
- 0到1：目前的最大难点

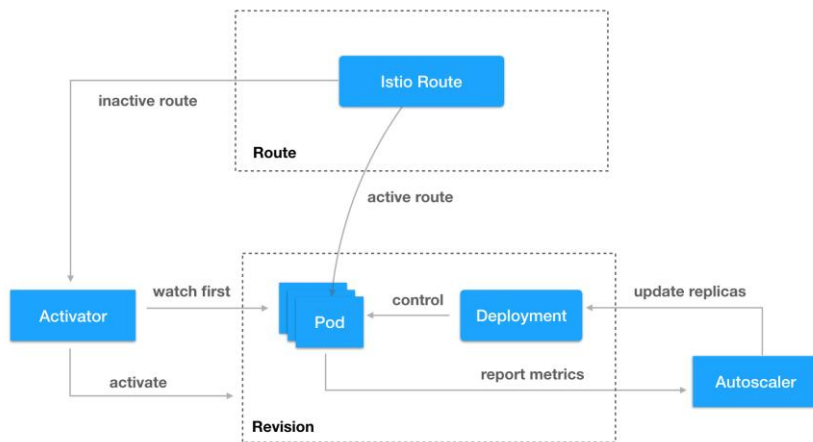
## ✓ 背景

- 为了实现 autoscaling, 在 Knative Serving 的每个 pods中有一个代理 (queue-proxy)
- 负责执行请求队列参数 (单线程或者多线程), 并向Autoscaler报告并发客户端指标
- 后果: 调用链路上又多了一层, 对整个性能势必会有影响

## ✓ Knative的解决方案

- 去掉Queue Proxy
- 计划直接使用 Istio 的 sidecar ( Envoy ) 来替换掉 queue proxy

- ✓ 目前 autoscaler 是knative自行实现的
- ✓ 计划转向采用 k8s 的原生能力
  - HPA (Horizontal Pod Autoscaler)
  - Custom Metrics





## ✓ Fast Brain

- 维持每个Pod所需的并发请求级别
  - 不好评估
  - 不够准确
- 刚开始是hard code的，最近修改为可配置

## ✓ Slow Brain

- 根据CPU，内存和延迟统计信息提出所需的级别
- 目前尚未实现

## ✓ 支持的事件源和消息系统远不完善

- 外部事件源只支持 github、kubernetes 和 Google PubSub
- 消息系统 (bus) 只支持内存/kafka/Google Cloud PubSub

## ✓ 后续改进

- Knative 本身会慢慢扩展
- 更多的还是需要用户自行实现

## ✓ Knative目前还没有函数的 pipeline 管理

- 类似 AWS Step Functions
- 在官方文档中没有看到相关的 roadmap,
- 但是这个功能是必不可少的
- CNCF serverless WG 正在制定workflow标准
- knative 如果不做, 就只能社区来做补充
  - 有待和knative官方进一步沟通

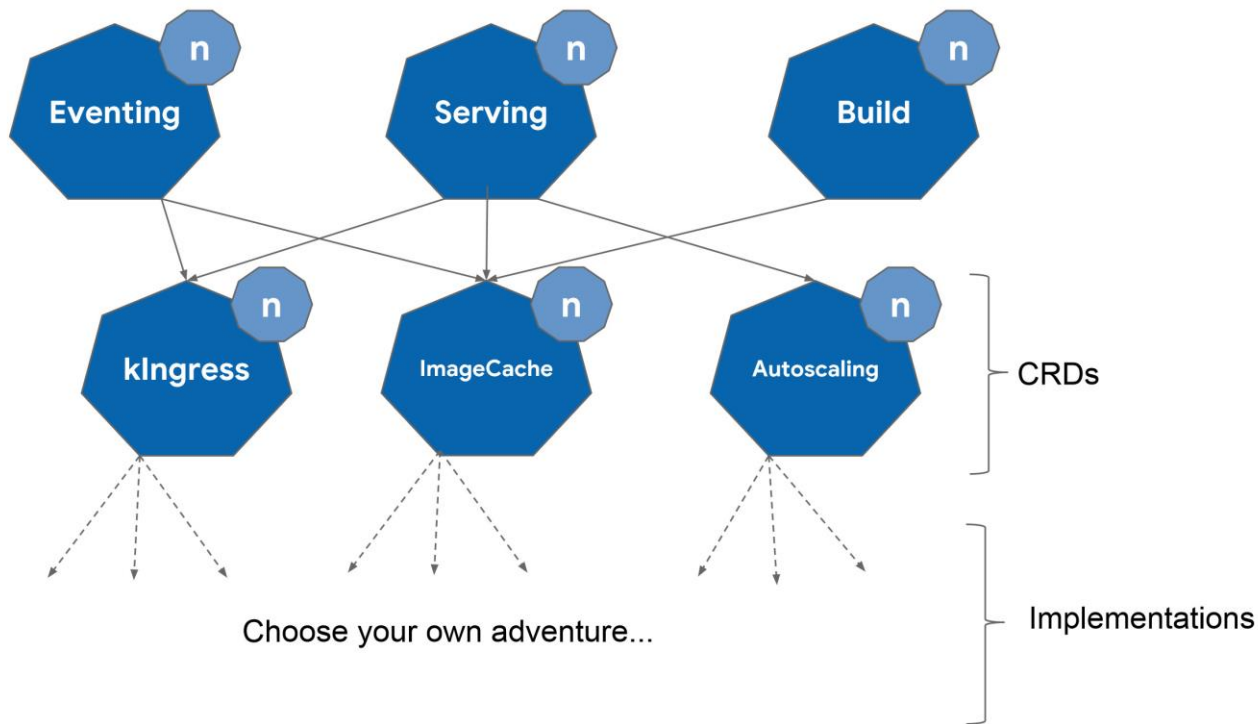


**AWS Step Functions**

- ✓ Percentage Splits
- ✓ Cross-namespace backend references
- ✓ Shared IP ingress (host : routing)
- ✓ Ability to rewrite HTTP requests
- ✓ Metrics collection (telemetry)
- ✓ Mutual TLS / unified authentication
- ✓ Access Control policy / authorization
- ✓ Container queueing
- ✓ Fast reprogramming
- ✓ Status reporting of config propagation

# Knative的可拔插设计 (Pluggability)

Loosely coupled at the top, and pluggable at the bottom



# 5

1

Knative是什么?

2

Knative主要组件

3

Knative分析与探讨

4

Knative后续发展

5

总结



## ✓ Knative的优势

- 产品定位准确，技术方向明确，推出时机精准
- Kubernetes + Service mesh + Serverless 组合威力
- 不拘泥于FaaS，支持BaaS和传统应用，适用性更广泛
- 平台化，标准化

## ✓ 存在的问题

- 太早期，不够成熟，太多东西进行中
- 系统复杂度高

# 欢迎加入ServiceMesher社区



---

<http://www.servicemesh.com>

ServiceMesh中国技术社区



---

微信公众号

servicemesh