滴雨科技物联网 -概念篇

最新更新文档请访问 www.microraindrop.com

1 Kaa 概念

1.1 终点

在 Kaa 术语中，端点（EP）代表物联网方程式的*事物*元素。端点是您要通过 Kaa 平台管理的任何终端设备。端点可以是物理设备或其软件仿真。

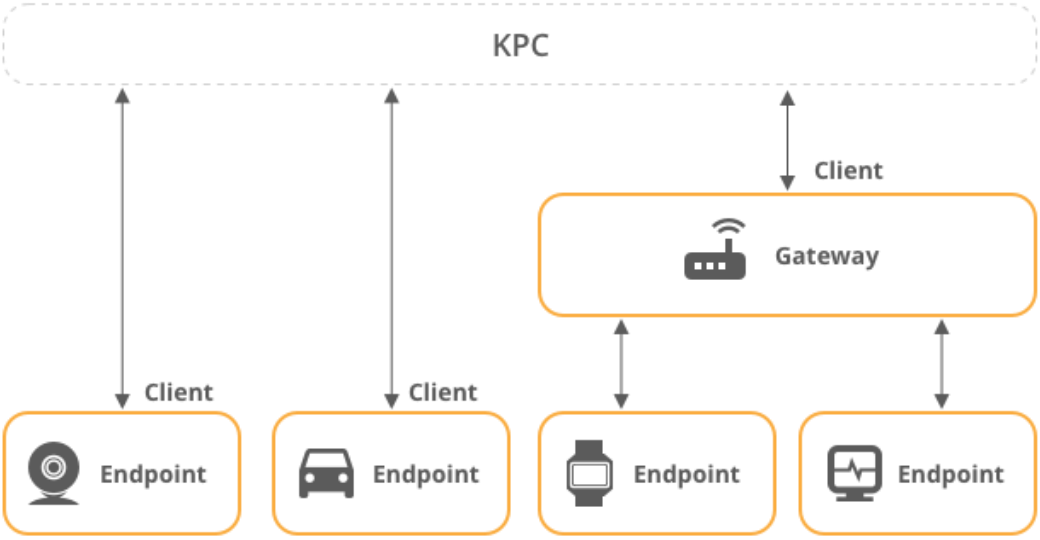
进入平台的所有数据都与端点关联。

“端点”是一个跨领域的概念-从通信层到表示的所有 Kaa 服务都知道端点。

确切地说，端点可以比设备小的单位，这意味着物理设备可以包括多个端点。例如，您要使用 Kaa 平台管理恒温器，以便空调将在特定温度下自动打开和关闭。

您可以通过以下方式之一来管理恒温器：

- 整个恒温器单元充当与 Kaa 服务器交换数据（温度读数等）的单个端点。
- 恒温器组件（温度和湿度传感器，开/关开关等）充当单独的端点。



请注意，我们不使用“设备”或“事物”之类的术语。这是因为单个物理设备可以表示为多个独立的端点。或者您可以从一个代表多个客户端的设备打开多个连接。

1.2 端点编号

端点 ID 用于唯一标识 Kaa 平台实例内的端点。端点 ID 通常是由 Kaa 在创建新端点（例如）时自动生成的 UUIDb1857120-9e72-4886-b3c2-b1bddccbf475。但是，只要它们与^[a-zA-Z0-9._~-]+$正则表达式模式匹配，也可以使用用户定义的端点 ID 。端点 ID 一旦创建就无法更改。

所有端点数据（例如元数据属性，收集的时间序列数据点，命令等）都与特定的端点 ID 相关联。每当您在 Kaa 中（主要通过 REST API 或 NATS）检索或管理与端点相关的数据时，您都会看到端点 ID。

1.3 端点令牌

端点令牌在与 Kaa 平台进行端点通信期间用于端点标识。令牌在应用范围内是唯一的，并且仅分配给一个端点。当消息到达 Kaa 平台时，端点令牌被解析为端点 ID。

发送到平台的每个消息都必须具有终结点令牌，以便平台可以标识发送消息的终结点。对于基于 MQTT 的 Kaa 协议，端点令牌位于 MQTT 主题内。

端点令牌是不包含下列保留字符的任何非空字符串：`+`，`#`，`/`，和`.`。通常，令牌是由 Kaa 自动生成的伪随机字符串（例如 `JTjdbENzHh`），但是为了方便起见，您还可以设置端点令牌（例如设备序列号，MAC 地址等）。

端点令牌和端点 ID 的解耦使您可以挂起，吊销，重新激活和重新发行端点令牌，而不会影响端点 ID。要与 Kaa 平台进行通信，您的设备或网关无需知道端点 ID，而只需知道令牌。

1.4 端点元数据

端点元数据是与端点关联的键值属性。它可以是端点的位置，描述，序列号，硬件版本等。元数据存储在 Endpoint Registered 服务上，可以通过两种方式读取或更新：通过通信层或 EPR REST API。

简而言之，端点元数据是与特定端点关联的自由结构的 JSON 文档。

1.5 Kaa 客户

要使设备（端点）与 Kaa 服务器交换数据，您需要一个*客户端*。

Kaa 客户端是一款可识别您的端点并将其数据发送到 Kaa 服务器以及从 Kaa 服务器接收数据的软件。您可以根据需要使用客户端来表示 Kaa 平台的尽可能多的端点。

您的 Kaa 客户端可以是支持 IoT 协议（MQTT，CoAP 等）以与 Kaa 服务器交换数据的任何软件应用程序。

1.6 应用程序和应用程序版本

Kaa 平台旨在同时处理不同类型的设备，并允许它们在单个解决方案范围内共存。为此，我们使用 **Kaa 应用程序**的概念。

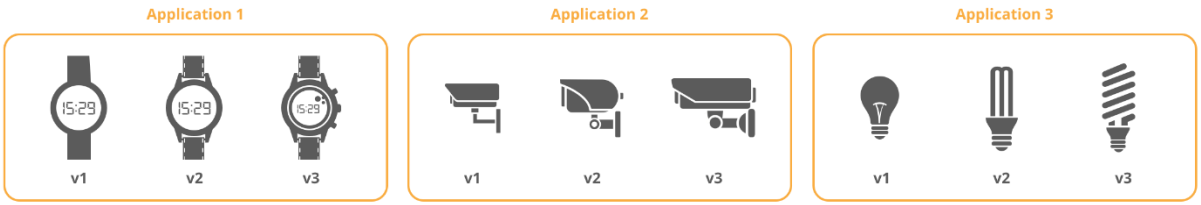
可以将 Kaa 应用程序视为容器，在其中放置取决于设备类型的系统配置。

例如，您要管理一个装有冰箱和咖啡机的智能房屋。在这种情况下，您可以设置两种不同的应用程序：一种用于冰箱，另一种用于咖啡机。这意味着您的所有冰箱将在一个应用程序中表示为一组端点，而所有咖啡机将在另一个应用程序中使用。

应用程序可以具有**版本**。使用版本通过添加或淘汰功能来发展您的设备，同时保持旧版本的正常运行。

Kaa 支持同时运行多个版本。这意味着您可以在不干扰现有端点的情况下为端点开发新功能。

由于各种兼容性原因，应用程序和应用程序版本名称必须限制为小写拉丁字母（`a-z`），数字（`0-9`），破折号（`-`）和下划线（`_`）。



1.7 解决方案

解决方案是 Kaa Web 仪表板中的逻辑命名空间，其中包含特定的用例，例如车队管理。通常，解决方案由一组可视组件（仪表板，小部件等）组成，并与一个或多个 Kaa 应用程序一起运行。

请注意，解决方案概念的当前 WD 实现基于 Kaa 应用程序。因此，解决方案等于应用程序。唯一的区别是合乎逻辑的。

1.8 Kaa 服务

Kaa 平台设计基于微服务。在 Kaa 上下文中，组成 Kaa 平台的微*服务*简称为 *Kaa 服务*，或更简单的服务。

Kaa 服务是打包在 Docker 映像中的一组服务器软件功能。要将服务堆栈用作一个集成平台，您需要将它们部署在群集中。相同的服务可以在具有不同配置（服务实例）的群集中运行。

换句话说，服务是一种软件模块，为实用目的提供一些功能，例如安全性，数据收集，数据传输和存储，设备通信和管理等。将服务视为您堆叠在一起的构建块形成您的 Kaa 解决方案。您不需要一次使用所有类型的块，只需要正确的块即可完成工作。

例如，您希望从简单地从设备收集数据并在仪表板上可视化开始，然后您只需安装 Kaa 服务即可进行数据收集和可视化。假设您随后意识到，您还希望存储数据并具有一些访问管理功能-没问题，只需将相应的 Kaa 服务添加到您的部署中即可。考虑一下您只需根据需要下载并安装的软件插件-与 Kaa 服务相同，您可以将其下载并安装在平台中以获得其他功能。

您可以根据用例和您的期望在平台中包含不同数量的不同类型的服务。这意味着您的 Kaa 平台部署的规模，复杂性和功能会根据您的需求和案例的具体情况而变化。

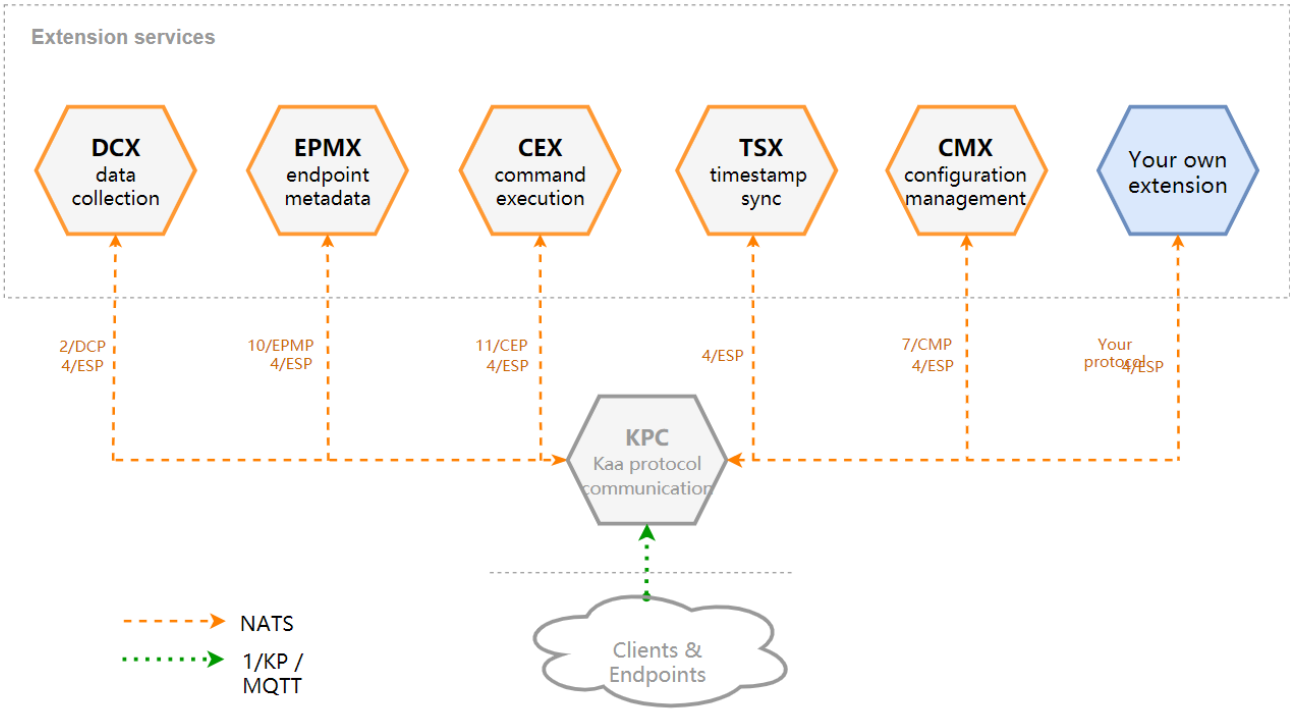
Kaa 团队不断开发大量服务供您选择。另外，您可以开发自己的服务并将其集成到平台中。

有关每个 Kaa 服务的详细文档，请使用页面顶部的“**组件**”下拉列表。

1.9 推广服务

扩展（扩展服务）是特殊的 Kaa 平台服务，实现了平台为端点提供的各种 IoT 功能，例如数据收集，配置管理，元数据属性同步等。根据经验，当您看到一个服务简称时，以“X”结尾-可能是扩展服务。Kaa 平台扩展的示例包括数据收集扩展（DCX），端点元数据扩展（EPMX），配置管理扩展（CMX）等。

所有扩展都位于平台通信层后面，并与通信服务进行互操作（请参阅 KPC）。通过仅在通信层后面添加新的扩展，这种架构即可让您轻松地通过新的 IoT 功能扩展 Kaa 平台：



阅读有关 Kaa 协议 MQTT 主题结构的信息，以了解如何将端点消息路由到扩展。

1.10 服务接口协议

不同的服务旨在处理不同的数据。与不同服务一起使用的数据的结构和格式记录在接口协议中，该协议以 Kaa RFC 列表的形式归档。除了规范 Kaa 服务之间的数据交换外，Kaa RFC 还为几乎所有平台实体定义了规则。

1.11 蓝图

Kaa 蓝图是对某些 Kaa 解决方案部署及其所有设置和配置参数的详细说明性描述。

为 Kaa 解决方案定义一个蓝图后，您就可以在继续开发该蓝图的同时使该解决方案正常运行：您可以存储，缩放，调整，测试，将其部署到其他地方，等等。

蓝图包含有关在 Kaa 解决方案的应用程序中使用的服务的名称，版本和所有配置参数的信息。它还包含有关用于设置解决方案集群的环境和部署参数的信息。

1.12 通讯协议

1.12.1 1 / KP（Kaa 协议）

Kaa 的主要通信协议是 1 / KP，它基于 MQTT。该协议非常通用，不会对客户端施加任何其他格式限制。

它旨在允许多个端点通过单个连接进行通信，并通过 MQTT 网关和代理进行遍历。

考虑到将来实现 CoAP 绑定的实现也是经过深思熟虑的。

1 / KP 并未定义客户端可用的所有服务器功能。相反，它定义了 MQTT 主题格式和一般准则，并依靠协议扩展来处理其余内容。

1 / KP 扩展包括 2 / DCP，7 / CMP 和 10 / EPMP。扩展定义了特定的有效负载格式以及服务器应如何处理消息。

1.12.2 4 / ESP（扩展服务协议）

虽然 1 / KP 及其扩展名似乎很重要，因为它定义了与客户端的所有通信，但实际上，它对平台体系结构的影响非常有限。只有一个微服务知道或关心 1 / KP-KPC（Kaa 协议通信服务）。

该微服务处理客户端连接，并与 4 / ESP（扩展服务协议）之间来回转换消息，该协议是平台其余部分用来与端点对话的协议。

这种方法允许将所有设备通信隔离到一个易于扩展或替换的单独层中。

2 架构概述

- 微服务抽象
- 服务组成和服务间通信
- 可扩展性
- 部署方式
- 组态
  - 自动配置推出

2.1 微服务抽象

Kaa 平台是云原生的物联网平台。

平台的体系结构基于**微服务方法，**并充分利用了它。每个 Kaa 微服务都是一个独立的构建块。您可以混合并匹配这些块以创建一致的解决方案。

在整个平台的规模上，Kaa 微服务只是一堆黑匣子。这意味着任何单个微服务的体系结构对于整个 Kaa 平台的体系结构都不重要。

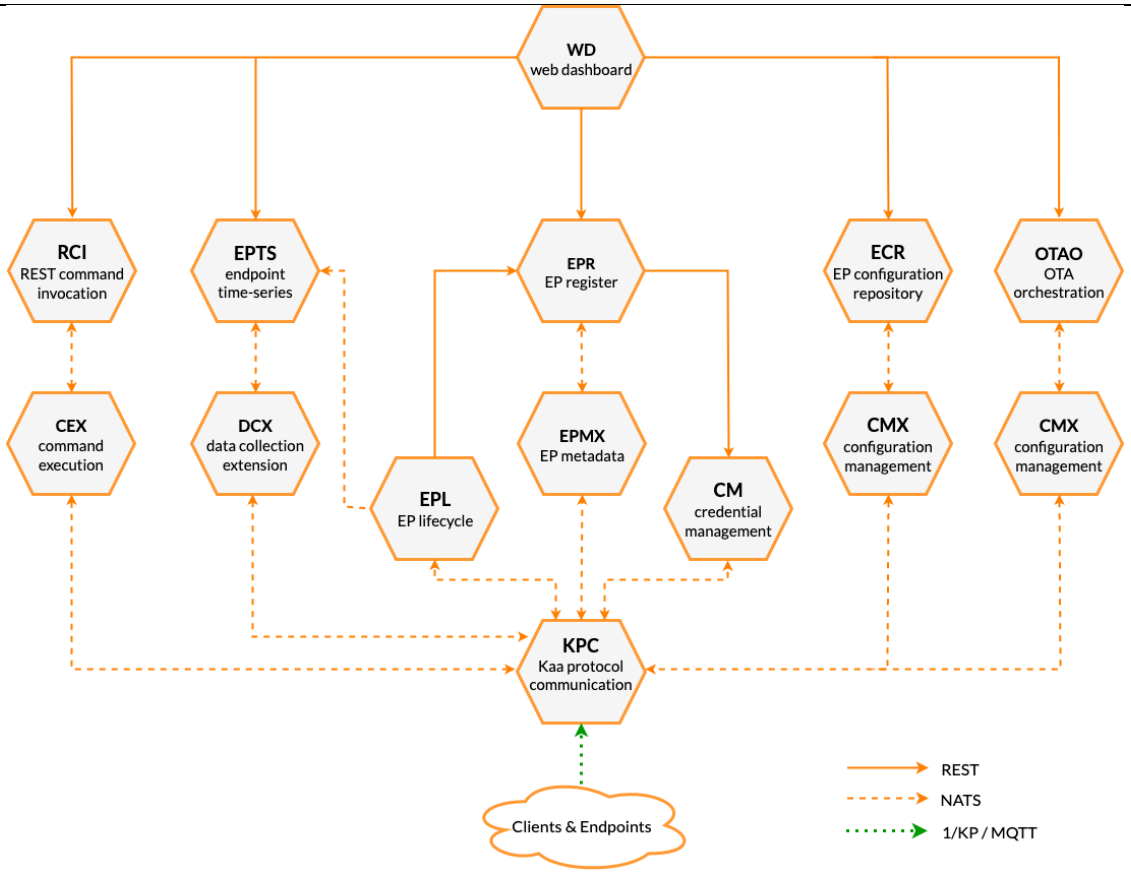
为了实现这种微服务抽象，Kaa 工程师使用了多种技术。

首先，**所有服务间通信协议都使用 HTTP 和 NATS 传输消息**，并使用 **JSON 和 Avro 对其进行编码。**所有这些技术都是定义明确的，并且对所有主流编程语言都有多种实现，因此**我们不受任何实现语言的束缚。**目前，大多数 Kaa 微服务都是用 **Java，Go 和 TypeScript（NodeJS）**编写的，但是平台用户也可以使用 **Python，Rust，Scala 等**实现其兼容和集成的微服务。

其次，所有 Kaa 微服务都**作为 Docker 映像分发。**Docker 有效地抽象了所有微服务设置和运行时依赖关系-使用 Java 运行 Docker 容器与运行 Go 驱动的 Docker 容器没有什么不同。这可以帮助运营团队部署 Kaa 解决方案，而无需设置依赖项。

第三，在 Docker 之上，我们使用容器编排系统，Kubernetes 是首选系统。大量的生态系统项目，例如 Helm，NGINX，NATS，Prometheus，Fluentd，Grafana 等等，都改善了 Kaa 用户体验，并帮助以任何规模和复杂性运行基于 Kaa 的解决方案。

以下是 **Kaa 微服务的典型组成图：**



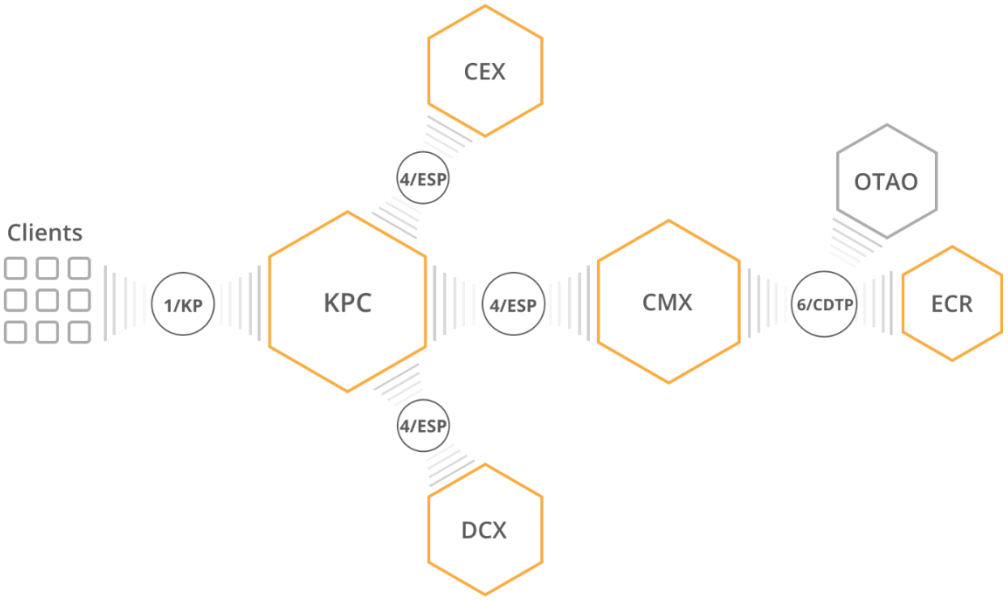
这些技术与定义明确并记录在案的接口相结合，使我们**无需任何人就**可以**交换整个微服务实现**。

2.2 服务组成和服务间通信

为了使微服务可组合，**Kaa 平台使用了定义良好的基于 NATS 的协议**。我们使用轻量级变更管理程序来开发这些协议，并与微服务实现分开跟踪它们。这允许单个协议的多种实现在单个解决方案中共存和合作。

主要的服务间通信指南在 3 / ISM（服务间消息传递） RFC 中定义。所有其他服务间协议均基于 3 / ISM，通常定义一个或两个角色。例如，4 / ESP（扩展服务协议）定义“通信服务”和“扩展服务”角色，而 6 / CDTP（配置数据传输协议）定义“配置数据提供者”和“配置数据使用者”角色。

这非常有用，因为它允许每个角色具有多种不同的实现。



例如，**我们可以有多个“通信服务”实现，每个实现一个不同的面向客户端的协议：MQTT，CoAP，HTTP，专有的基于 UDP 的协议**-该服务的唯一要求是实现 4 的“通信服务”端/ ESP。这样可以轻松交换客户端通信层，而不会影响任何其他服务-这是完全透明的。此外，您可以部署多个通信服务实现来处理在单个解决方案中通过不同协议进行通信的客户端。

另一个示例是 ECR（端点配置存储库）和 OTAO（空中编排）服务，它们均实现 6 / CDTP 的“配置数据提供者”侧。因此，所有的微服务-CMX，KPC-均可与任何实现一起使用。

2.3 可扩展性

在描述 Kaa 平台的可伸缩性功能之前，让我们先定义一些术语。

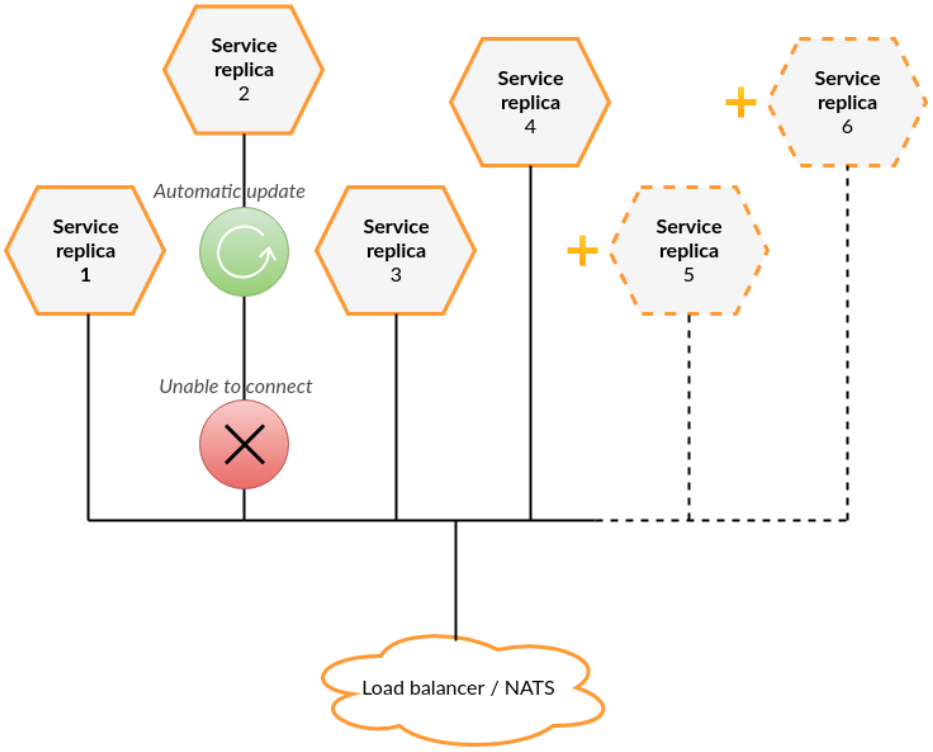
Kaa 服务是打包在 Docker 映像中的微**服务**。

服务实例是 Kaa 服务及其配置。

为了使服务做一些有用的事情，您需要至少部署一个**服务实例副本**（或简称为**副本**）——一个正在运行的 Docker 容器。

每个服务实例可以具有处理负载所需的多个副本。 大多数服务副本是独立的，并且不会相互通信。他们既没有主从关系，也没有主人与主人的关系。

实例副本利用 NATS 队列组，因此任何指向服务实例的请求都可以由任何副本处理。**没有单点故障。**



所有 Kaa 服务都可以水平缩放。 许多服务根本不共享副本之间的任何数据。其他人使用 Redis 或其他数据存储共享状态。在所有情况下，每个服务都内部处理水平可伸缩性，因此您需要查看特定于服务的文档以了解扩展细节。在大多数情况下，它归结为扩展数据存储。并且我们已经注意选择可扩展的数据存储。

2.4 部署方式

Kaa 平台利用 Kubernetes 作为所有解决方案**的企业级编排平台。**它使您可以从管理容器的生命周期，减轻节点故障，联网等方面进行抽象，而仅关注业务领域。

Kubernetes 是建立在声明描述，这意味着你定义*什么*你需要和 Kubernetes 计算出*如何*得到它自己的。**声明性方法使您可以灵活地运行群集，而无需更改**应用程序中**的任何代码行。**

这样一来，您几乎可以在任何地方运行 Kaa：在**私有裸机群集上，在 Amazon AWS 或 Google 的 GCP 之类的公共云中，甚至在笔记本电脑上。** 您只需要安装 Kubernetes 和 Kaa Cluster 蓝图。

就 Kaa 而言，集群蓝图是 Kubernetes 资源定义和 Kaa 微服务配置的集合。两者都是基于文本的，预计将使用 VCS（例如 Git）进行版本控制。

蓝图完全定义了群集状态，但存储的数据除外。换句话说，**您可以轻松地恢复或复制集群（例如，出于测试或开发目的）。**此外，可以直接从 VCS 提供服务的配置，让您仅需按下一次提交即可更改正在运行的群集的行为！

2.5 组态

Kaa 平台的所有组件都可以从文件系统读取配置。配置文件的默认路径为 `/srv/<service-name>/service-config.yml`（例如`/srv/kpc/service-config.yml`）。

从文件系统读取文件使我们可以轻松地在 Kubernetes 集群上运行时管理配置。我们建议将每个服务的配置存储在单独的 Kubernetes `ConfigMap` 对象中，并将其安装到容器的文件系统中。KPC `Pod` 及其定义的示例定义 `ConfigMap` 可以在下面找到：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kpc-config
data:
  service-config.yml: |
    kaa:
      applications:
        demo_application:
          versions:
Show more
```

配置性能和 Kubernetes 定义的例子的列表可以在每个卡阿微服务的文档中找到*配置*和*部署*分别段。

2.5.1 自动配置推出

为了自动应用配置更新，我们建议使用名为 Reloader 的第三方工具。当检测到 `ConfigMap` 或其更新时 `Secret`，Reloader 会利用 Kubernetes 的本机功能对受影响的服务进行滚动更新。这使您能够安全地推出配置更改，而不必担心破坏群集。

3 功能和组件

Kaa 是一个**模块化的物联网平台**，利用**微服务架构**将关注点，可扩展性和可扩展性明确分离。本文档部分讨论了 **Kaa 平台**最重要的**核心功能，其架构以及如何使用它们。** 浏览嵌套的小节，以详细了解 Kaa 的各种功能以及支持这些功能的组件。

特征	描述
设备管理	记录物理设备的**数字孪生。设备访问凭据，元数据属性，过滤和分组。**
通讯	**设备和网关通信**支持。**身份验证，访问授权，数据交换和多路复用。**
数据采集	设备**遥测数据收集**和存储。**时间序列数据，设备日志，警报。连接外部系统。**
配置管理	**设备配置数据的**管理和**分发**：单独或批量。
命令调用	**远程控制**连接的设备。立即和延迟的**命令调用。**
软件更新	**软件版本管理**和升级流程。有针对性的，逐步的和计划的软件推出。
可视化	用于数据可视化，设备管理，平台管理等的 **Web 界面。**
基础设施	Kaa 平台的**基础结构**组件将**操作和管理集群化。**
杂项	Kaa 平台的**其他**附加功能。