滴雨科技物联网 -功能和组件篇

最新更新文档请访问 www.microraindrop.com

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。

1 设备管理

- 先决条件
- 基本概念

- 组件

1.1 先决条件

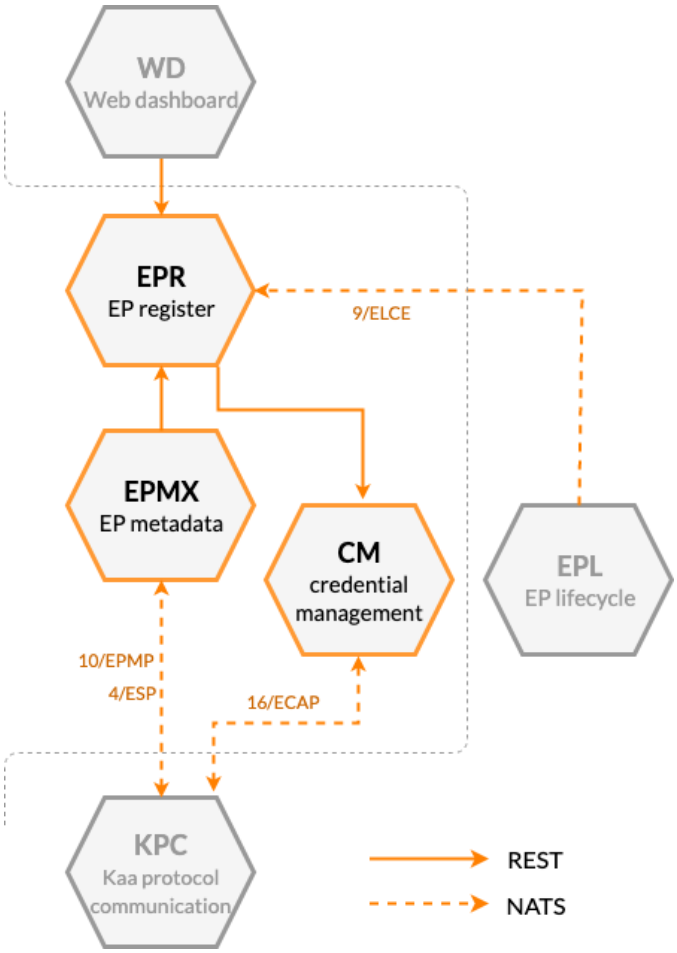
- 您了解 Kaa 平台基于微服务的架构。

1.2 基本概念

设备身份管理是 Kaa IoT 平台的核心功能之一。有了它的帮助，您可以：

- 保留设备数字双胞胎的记录（在 Kaa 术语中也称为“ 端点 ”）。
- 管理设备用于连接到 Kaa 的访问凭据。
- 管理端点元数据属性。
- 通过过滤设备的元数据属性对设备进行分组。

Kaa 中的设备身份管理通过以下几种微服务功能的组合来启用：端点注册服务（EPR），凭据管理服务（CM）和端点元数据扩展服务（EPMX）。



端点注册服务（EPR）维护一个数据库，该数据库包含解决方案中注册的所有端点及其关联的键/值属性（元数据）。您可以使用 EPR REST API 来：

- 在解决方案集群中定义的 Kaa 应用程序之一中设置新的端点。

- 检索端点信息和元数据属性。
- 更新端点元数据属性。
- 配置**端点过滤器**：查询端点数据和元数据属性，使您可以灵活地划分 Kaa 平台实例中的设备。
- 检索与先前设置的过滤器匹配的端点。

EPR 服务也是各种重要状态更新事件的来源。您可以使用 NATS 代理订阅以下内容：

- 端点（取消）注册事件和端点应用程序版本更新符合 9 / ELCE 协议。
- 根据 15 / EME 发送端点元数据更新事件。
- 随 18 / EFE 一起提供的端点过滤器事件可用于监视端点过滤器（去激活）以及与这些过滤器不匹配的端点。

凭据管理服务（CM）管理设备凭据并验证连接的客户端和端点。在您的物联网设备能够连接到 Kaa 之前，必须提供相应的凭据。CM 支持以下凭据类型：

- *端点令牌*用于端点标识。
- 客户端身份验证的*用户名/密码*组合。
- 用于客户端身份验证的客户端 *SSL 证书*。

您可以使用 CM REST API 来管理端点和客户端凭证：

- 设置新的凭据。
- 过渡凭证状态。
- 删除凭据。

CM 支持基于 NATS 的 16 / ECAP 接口，用于通信服务以验证连接设备。

端点元数据扩展服务（EPMX）允许连接的设备检索和更新存储在 EPR 中的端点元数据属性。当您希望设备报告有关其自身的一些描述性信息时（例如序列号，MAC 地址，已安装的软件版本，位置等），此功能很有用。您可以进一步使用这些属性在 EPR，过滤，显示等中进行端点搜索。

EPMX 实现 10 / EPMP 扩展协议，以允许端点检索和管理其元数据。除了实现 10 / EPMP，EPMX 还支持元数据白名单。此功能允许指定允许端点访问的元数据字段的列表。它还允许禁止更新特定字段，使其成为只读。

EPMX 本身不保留元数据，并为此与 Endpoint Register 服务（EPR）集成。

1.3 组件

下表总结了有助于此功能的 Kaa 平台组件的列表：

服务	版
端点寄存器（EPR）	1.1.1
端点元数据扩展（EPMX）	1.1.1
凭证管理（CM）	1.1.1

2 凭据管理服务（CM）

2.1 总览

- 介面
  - 认证方式
  - 凭证管理
  - EP 未注册的事件
  - Tekton 整合

凭据管理服务（CM）用于客户端和终结点端点的身份验证。

CM 支持：

- 通过*端点令牌进行*端点标识。
- 通过*用户名/密码*组合进行客户端身份验证。
- 通过客户端 *SSL 证书*进行客户端身份验证（使用颁发者字段和证书序列号）。

CM 提供 REST API 来管理客户端和端点凭据及其状态。CM 维护一个凭据状态机，如下图所示。

凭据可以处于以下状态之一：

- *非活动*状态是新设置的凭证的初始状态，尚未用于认证端点或客户端。
- *主动*状态是状态凭证在首次用于端点或客户端身份验证后自动移动到的状态。凭证可以从活动状态被挂起或吊销。
- *暂停*状态用于暂时禁用的凭据。CM 服务将拒绝具有暂停凭据的身份验证请求。暂停的凭证可以重新激活。
- *撤消*状态是不再有效的凭据的终端状态。

CM 将所有与凭证相关的数据持久保存到 MariaDB。

2.1.1 介面

CM 支持许多接口来执行其功能角色。下图总结了主要支持的接口。

对于服务间通信，Kaa 服务主要使用在 NATS 消息传递系统上运行的 REST API 和消息传递协议。

2.1.2 认证方式

CM 支持用于通信服务的端点和客户端身份验证协议，以验证端点和客户端提供的凭据。

每当端点或客户端凭据从活动状态转换为挂起或吊销状态或被删除时，CM 都会分别广播端点令牌撤销事件和客户端凭据撤销事件。

2.1.3 凭证管理

CM 提供了一个基于 REST 的界面来管理端点和客户端凭证：

- 供应新凭证
- 过渡凭证状态
- 删除凭证

2.1.4 EP 未注册的事件

CM 侦听端点未注册的事件，以使有效端点凭据的列表保持最新。收到此类事件后，CM 将从数据库中删除相应的终结点令牌。

2.1.5 Tekton 整合

CM 与 Kaa Tekton 集成在一起，用于集中式应用程序配置管理。它通过 17 / SCMP 从 Tekton 接收配置更新消息，并使用 Tekton REST API 检索当前配置。

有关更多信息，请参见配置。

2.2 部署方式

- 在 Kubernetes 上安装 CM 图表
  - 制备
  - 安装
- 图表要求

- 图表值
- 环境变量

所有 Kaa 服务（包括 CM）均以 Helm 图表的形式分发。您可以使用 Kubernetes 运行这些图表。

2.2.1 在 Kubernetes 上安装 CM 图表

2.2.1.1 制备

对于 Kubernetes 中的整个 Kaa 集群，这些步骤应该执行一次。

1. 安装 Kubernetes。
2. 安装 Helm 客户端和 Tiller 服务器。
3. 创建一个 Kaa 许可证密钥（记住要输入您的 Kaa 许可证密钥文件的内容和密码）：

```
export HISTCONTROL=ignorespace # Prevent saving your key password in the shell history; note the leading space in the next line
 cat << EOF > /tmp/kaa-licence.yaml
apiVersion: v1
data:
  file: < your licence key file contents, base64-encoded >
  password: < your licence key password >
kind: Secret
metadata:
  name: license
  type: Opaque
EOF
kubectl create -f /tmp/kaa-licence.yaml
```

16. 为官方的 KaaIoT 泊坞窗注册表指定镜像拉密钥。要定义此机密，请使用您的 KaaID 凭据：

```
export HISTCONTROL=ignorespace  # Prevent saving your credentials in the shell history; note the leading space in the next line
 export KAAID_EMAIL=<your KaaID email, eg. bob@example.com> KAAID_PASSWORD=<your KaaID password>
kubectl create secret docker-registry kaaid --docker-server=hub.kaaiot.net --docker-username=$KAAID_EMAIL --docker-email=$KAAID_EMAIL --docker-password=$KAAID_PASSWORD
```

如果要在 Kaa 群集中启用 API 安全性（推荐），则应为服务创建 auth 服务器后端密码，以便能够请求 PAT 令牌。

```
export HISTCONTROL=ignorespace # Prevent saving your client ID and secret in the shell history; note the leading space in the next line
 cat << EOF > /tmp/keycloak-backend-secret.yaml
apiVersion: v1
data:
  client-id: < your client-id, base64-encoded >
  client-secret: < your client-secret, base64-encoded >
kind: Secret
metadata:
  name: "keycloak-backend"
  labels:
    app.kubernetes.io/name: "kaa-name"
    helm.sh/chart: "kaa-chart-version"
    app.kubernetes.io/instance: "kaa-instance-name"
    app.kubernetes.io/managed-by: "release-service-name"
EOF
kubectl create -f /tmp/keycloak-backend-secret.yaml
```

36. 添加 KaaIoT Helm 存储库：

```
helm repo add kaa-museum https://museum.kaaiot.net/
```

2.2.1.2 安装

完成准备步骤后，一切就绪，可以在您的 Kubernetes 集群上部署 CM。要部署该服务，请运行以下命令（遵守对先前创建的许可证密钥的引用）：

```
helm install --set global.license.secretName=license kaa-museum/cm --name kaa-cm
```

检查 Pod 是否正在运行：

```
kubectl get pods
```

服务初始化完成后，您应该观察到类似于以下内容的输出：

```
NAME                                    READY   STATUS             RESTARTS
AGE

...

kaa-cm                                  1/1     Running            0
2m

...
```

2.2.2 图表要求

资料库	名称	版
@稳定	玛丽亚德	5.7.2
@稳定	ts	2.0.6

2.2.3 图表值

键	类型	默认	描述
亲和力	宾语	`{}`	
注解。部署	宾语	`{}`	
注解.pod	宾语	`{}`	
配置	串	`""`	服务配置映射的内容，作为配置文件自动安装到 Pod 中。
环保	宾语	`{}`	定义 Kubernetes 传递给服务副本的环境变量。
extraPodSpecs.automountServiceAccountToken	布尔	`false`	
fullnameOverride	串	`""`	
global.image.pullSecrets	清单	`[]`	图像拉取秘密名称列表。每个 `name` 字段都必须定义为一条记录。覆盖 `image.pullSecrets`。

键	类型	默认	描述
global.keycloak.backendSecretName	串	`""`	后端身份验证服务器密钥的名称，必须包含 base64 编码的 `client-id` 和 `client-secret`。需要。
global.keycloak.baseUrl	串	`""`	身份验证提供程序的基本 URL。需要。
global.keycloak.enabled	串	`""`	使用 auth 提供程序启用 API 安全性。覆盖 `keycloak.enabled`。
global.keycloak.realm	串	`""`	身份验证提供程序领域。需要。
global.license.secretName	串	`""`	许可证密钥的名称，必须包含两个 base64 编码的字段：`file`（PKCS＃12 中的许可证文件内容）和 `password`。覆盖 `license.secretName`。
global.mariadb.url	串	`""`	Mariadb 网址。
global.nats.url	串	`""`	NATS URL。覆盖 `nats.url`。
global.tekton.enabled	串	`""`	启用 Tekton 集成。禁用后，该服务将期望在配置映射中定义 Kaa 应用程序配置。覆盖 `tekton.enabled`。
global.tekton.url	串	`""`	Tekton URL。覆盖 `tekton.url`。
image.pullPolicy	串	`"IfNotPresent"`	Docker 映像拉取策略。
image.pullSecrets	清单	`[]`	图像拉取秘密名称列表。每个 `name` 字段都必须定义为一条记录。
映像库	串	`"hub.kaaiot.net/core/service/cm/cm"`	Docker 映像存储库映像 URL。
image.tag	串	`""`	Docker 图像标签版本可以拉出并运行。
入口注释	宾语	`{}`	
启用入口	布尔	`false`	

键	类型	默认	描述
ingress.hosts [0]	串	`"chart-example.local"`	
入口路径	清单	`[]`	
ingress.tls	清单	`[]`	
启用 keycloak	串	`""`	禁用用于服务的密钥库集成启用/禁用用于服务的密钥库集成
license.secretName	串	`""`	许可证密钥的名称，必须包含两个 base64 编码的字段：`file`（PKCS＃12 中的许可证文件内容）和 `password`。
mariadb.db.name	串	`"cm"`	
mariadb.db.password	串	`"kaa123"`	
mariadb.db.user	串	`"kaa"`	
mariadb.enabled	布尔	`true`	启用 Mariadb 依赖性。有关详细信息，请参见 https://github.com/helm/charts/tree/master/stable/mariadb
mariadb.master.persistence.enabled	布尔	`false`	
mariadb.replication.enabled	布尔	`false`	
mariadb.rootUser.forcePassword	布尔	`true`	
mariadb.rootUser.password	串	`"9011791362"`	
mariadb.url	串	`"mariadb://-:3306/cm"`	Mariadb 网址。
mariadb.urlOverride	串	`""`	Mariadb 网址。替代 `global.mariadb.url` 和 `mariadb.url`。

键	类型	默认	描述
元数据组件	串	`"backend"`	
metadata.partOf	串	`"kaa"`	
nameOverride	串	`""`	
nats.auth.enabled	布尔	`false`	
nats.enabled	布尔	`true`	启用 NATS 依赖性。有关详细信息，请参阅 https://github.com/helm/charts/tree/master/stable/nats。
nats.url	串	`"nats://-nats-client:4222"`	NATS URL。
nats.urlOverride	串	`""`	NATS URL。替代 `global.nats.url` 和 `nats.url`。
nodeSelector	宾语	`{}`	
probes.enabled	布尔	`true`	为容器启用活动性，就绪性和启动探针。
probes.liveness.initialDelaySeconds	整型	`90`	
probes.liveness.periodSeconds	整型	`3`	
probes.readiness.initialDelaySeconds	整型	`90`	
probes.readiness.periodSeconds	整型	`1`	
复制计数	整型	`1`	要运行的服务实例副本的数量。

键	类型	默认	描述
资源	宾语	`{}`	
securityContext	宾语	`{}`	
service.externalIPs	清单	`[]`	
service.loadBalancerIP	串	`""`	
服务端口	整型	`80`	
服务类型	串	`"ClusterIP"`	
启用 tekton	串	`"false"`	启用 Tekton 集成。禁用后，该服务将期望在中定义 Kaa 应用程序配置 `config`。
tekton.url	串	`"http://-tekton"`	Tekton URL。
tekton.urlOverride	串	`""`	Tekton URL。替代 `global.tekton.url` 和 `tekton.url`。
TerminationMessagePolicy	串	`"FallbackToLogsOnError"`	Kubernetes 终止消息策略。
宽容	清单	`[]`	
updateStrategy.type	串	`"RollingUpdate"`	部署更新策略。
waitContainers.enabled	布尔	`true`	等待依赖服务。
waitContainers.timeout	整型	`300`	等待依赖性服务超时（以秒为单位）。

没有描述的键是标准的 Kubernetes 值。有关这些的更多信息，请参考官方的 Kubernetes 文档。

2.2.4 环境变量

下表总结了 CM Docker 映像支持的变量，并提供了默认值和说明。

变量名	默认值	描述
INSTANCE_NAME	**厘米**	服务实例名称。
APP_CONFIG_PATH	**/srv/cm/service-config.yml**	容器内服务配置 YAML 文件的路径。如果在 Kubernetes 中运行，请考虑使用 K8s Volumes 进行外部化。
KAA_TEKTON_ENABLED	**假**	启用 Tekton 集成。
KAA_TEKTON_URL	**http：// tekton**	Tekton 服务的 URL。
JMX_ENABLE	**假**	启用 JMX 监视。
JMX_PORT	**10500**	JMX 服务端口。
JMX_MONITOR_USER_PASSWORD		JMX 监视器用户密码。何时需要 JMX_ENABLE=true。
JAVA_OPTIONS	**-Xmx700m**	Java 进程启动的其他参数。
NATS_USERNAME		用于连接到 NATS 消息代理的用户名。
NATS_PASSWORD		连接到 NATS 消息代理的密码。
KAA_LICENSE_CERT_PATH	**/run/license/license.p12**	PKCS # 12 格式的 Kaa 平台许可证证书文件的路径。
KAA_LICENSE_CERT_PASSWORD		许可证证书密码。需要。

2.3 组态

CM 从文件中加载配置数据（通常由 Kubernetes 配置图支持）。如果该服务无法检索配置数据，它将不会启动。有关详细信息，请参考配置。

2.3.1 服务配置结构

推荐的配置格式为 YAML。以下各节描述了影响服务功能各个方面的正式支持的配置选项。该服务可能支持以下列出的选项以外的其他选项，但这些选项不是公共 API 的一部分，可以随时更改或删除。

2.3.1.1 常规服务配置

CM 使用标准的 Spring Boot `server.port` 属性配置端口以公开 REST API。
特定于服务的配置属性位于下 `kaa.cm`。

```
   server:
     port: <unsigned short integer>  # Server port used to expose the REST API a
t


   kaa:
     cm:
       ep-token:
         length: <integer>           # Controls the length of CM-generated endpoi
nt tokens
```

2.3.1.2 Kaa 应用

可以将许多 Kaa 服务配置为具有不同的行为，具体取决于处理的数据所涉及的端点的应用程序版本。这称为*特定于 Appversion 的行为*，并在的服务配置中进行处理 `kaa.applications`。或者，可以从 Kaa Tekton 中获取特定于应用程序的配置。请参阅“ Tekton 配置”部分， 以了解如何配置此类集成。

```
kaa:
  applications:
    <application 1 name>:
      versions:
        <application 1 version 1 name>:
```

2.3.1.3 Tekton

CM 支持与 Kaa Tekton 集成，以进行集中的应用程序配置管理。以下配置选项设置了集成界面。

```
kaa:
  tekton:
    enabled: <boolean>    # Enables Tekton integration. False by default. Als
o can be set with the KAA_TEKTON_ENABLED environment variable.
    url: <string>         # URL of the Tekton service. "http://tekton" by defa
ult. Also can be set with the KAA_TEKTON_URL environment variable.
  scmp.consumer:
    provider.service-instance.name: <string>  # Service instance name of the
Tekton service. "tekton" by default. Also can be set with the KAA_SCMP_CONSUMER_
PROVIDER_SERVICE_INSTANCE_NAME environment variable.
```

2.3.1.4 端点寄存器接口

使用以下参数来配置 CM 用来侦听 EP 未注册事件的基于 REST 的端点注册接口。

```
kaa:
  cm:
    ep-register:
      service-instance:
        name: <service instance name>  # Instance name of the endpoint registe
r
```

2.3.1.5 数据持久化接口

CM 使用 Spring Boot 数据源配置。有关受支持的配置选项的列表，请参阅相应的文档。

注意出于安全原因，用户名和密码必须来自环境变量。

注意 CM 仅使用 MariaDB 数据源进行了测试。

2.3.2 导航键

以下参数配置 CM 与 NATS 的连接。

注意出于安全原因，NATS 用户名和密码均来自环境变量。

```
  nats:
    urls: <comma separated list of URL>  # NATS connection URLs
```

2.3.2.1 认证与授权

CM 的 REST API 安全性是根据带有 UMA 配置文件的 OAuth2 协议实现的。CM 使用 auth-kaa-server-starter 保护其 REST API。CM 使用 auth-kaa-client-starter 来执行对其他服务的经过身份验证的 REST API 调用。

有两个安全配置点：

1）配置 CM 的 HTTP 客户端，使其在对其他安全服务的请求中包括安全标头：

```
  security:
    oauth2:
      client:
        enabled: <boolean>            # Specified if authentication is enabled
for CM REST calls. False by default.
        base-url: <URL>              # Base host url to OAuth provider
        realm: <realm name>           # Realm in OAuth provider
        clientId: <client ID>         # Client ID on whose behalf the requests
will be made
        clientSecret: <client secret>  # Client secret on whose behalf the requ
ests will be made
```

2）通过启用授权和认证来保护 CM 提供的安全 REST API：

```
  security:
    ignored: <value>           # Controls authentication and authorization on R
EST API endpoints.
                              # Possible values: '/**' to disable security or '
none' to enable.
    oauth2:
      client:
        clientId: <clientId>    # Client id on whose behalf the requests will be
 made
        clientSecret: <secret>  # Client secret on whose behalf the requests wi
ll be made
```

```
        accessTokenUri: <URI>   # An OAuth2-compliant Token Endpoint for obtain
ing tokens via the 'Implicit Flow', 'Direct Grants', or 'Client Grants'. E.g. "h
ttps://your-url-to-auth-server/auth/realms/realm-a/protocol/openid-connect/toke
n"

        resourceUri: <URI>      # An UMA-compliant Resource Registration Endpoi
nt which resource servers can use to manage their protected resources and scope
s. E.g. "https://your-url-to-auth-server/auth/realms/realm-a/authz/protection/r
esource_set"

      resource:

        userInfoUri: <URI>      # This is the URL endpoint for the User Info ser
vice described in the OIDC specification. E.g. "https://your-url-to-auth-server
/auth/realms/realm-a/protocol/openid-connect/userinfo"
```

2.3.2.2 管理

CM 监视和管理实现基于 Spring Boot Actuator。有关受支持的配置选项的列表，请参阅相应的文档。

2.3.2.3 记录中

默认情况下，CM 使用 Spring Boot 日志配置和 logback 进行日志记录。有关受支持的配置选项的列表，请参阅相应的文档。

2.3.3 内置配置文件

为了方便起见，CM 随附了默认的内置配置配置文件。您可以控制使用 `spring.profiles.active` 参数启用哪些配置文件。
内置配置文件针对基于 Kubernetes 的生产部署进行了优化。它们没有定义任何 Kaa 应用程序-您必须为基于 Kaa 的特定解决方案配置它们。

2.3.3.1 默认

```
server:
  port: 80


kaa:
  cm:
    ep-token:
      length: 10
    ep-register:
      base-url: http://epr
```

```
        service-instance:
name: epr

spring:
  datasource:
    url: jdbc:mariadb://mariadb:3306/cm

nats:
  urls: nats://nats:4222

management:
  server:
    port: 8080
  endpoint:
    health:
      enabled: true
      show-details: always
      status:
        http-mapping:
          UNKNOWN: SERVICE_UNAVAILABLE
    shutdown:
      enabled: true
    metrics:
      enabled: true
  endpoints:
    web:
      base-path: /
      exposure:
        include: info,health,metrics

security:
  ignored: /**
```

3 EPMX

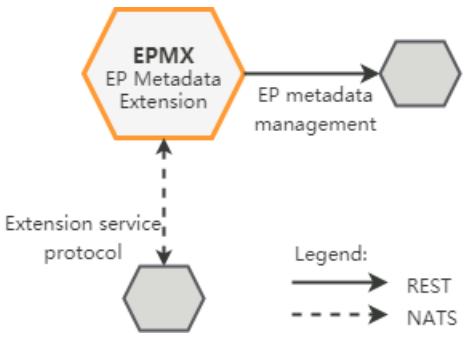
3.1 总览

- 介面
  - 扩展服务协议
  - 端点元数据管理
  - Tekton 整合

端点元数据扩展服务（EPMX）扩展了 Kaa 协议（1 / KP）的通信功能。它实现 10 / EPMP 扩展协议，以允许端点检索和管理其元数据。
除了实现 10 / EPMP，EPMX 还支持元数据白名单。此功能允许指定端点可访问的元数据字段的列表。它还允许禁止更新特定字段，使其成为只读。
EPMX 本身不保留元数据，并为此目的与端点注册服务（EPR）（或其他兼容实现）集成。

3.1.1 介面

EPMX 使用许多接口来执行其功能角色。下图总结了主要支持的接口。



对于服务间通信，Kaa 服务主要使用在 NATS 消息传递系统上运行的 REST API 和消息传递协议。

3.1.2 扩展服务协议

EPMX 通过 4 / ESP 上的 10 / EPMP 接收端点请求并向后发送响应。

3.1.3 端点元数据管理

EPMX 使用 EPR 的 REST API 查询和更新持久化的元数据。

3.1.4 Tekton 整合

EPMX 与 Kaa Tekton 集成在一起，用于集中式应用程序配置管理。它通过 17 / SCMP 从 Tekton 接收配置更新消息，并使用 Tekton REST API 检索当前配置。
有关更多信息，请参见配置。

3.2 部署方式

- 在 Kubernetes 上安装 EPMX 图表
  - 制备
  - 安装
- 图表要求
- 图表值
- 环境变量

所有 Kaa 服务（包括 EPMX）均以 Helm 图表的形式分发。您可以使用 Kubernetes 运行这些图表。

3.2.1 在 Kubernetes 上安装 EPMX 图表

3.2.1.1 **制备**

对于 Kubernetes 中的整个 Kaa 集群，这些步骤应该执行一次。

1. 安装 Kubernetes。
2. 安装 Helm 客户端和 Tiller 服务器。
3. 创建一个 Kaa 许可证密钥（记住要输入您的 Kaa 许可证密钥文件的内容和密码）：

```
4.  export HISTCONTROL=ignorespace # Prevent saving your key passwor
d in the shell history; note the leading space in the next line
5.   cat << EOF > /tmp/kaa-licence.yaml
6.  apiVersion: v1
```

```
data:
  file: < your licence key file contents, base64-encoded >
  password: < your licence key password >
kind: Secret
metadata:
  name: license
  type: Opaque
EOF
kubectl create -f /tmp/kaa-licence.yaml
```

16. 为官方的 KaaIoT 泊坞窗注册表指定镜像拉密钥。要定义此机密，请使用您的 KaaID 凭据：

```
export HISTCONTROL=ignorespace  # Prevent saving your credentials in the shell history; note the leading space in the next line
 export KAAID_EMAIL=<your KaaID email, eg. bob@example.com> KAAID_PASSWORD=<your KaaID password>
kubectl create secret docker-registry kaaid --docker-server=hub.kaaiot.net --docker-username=$KAAID_EMAIL --docker-email=$KAAID_EMAIL --docker-password=$KAAID_PASSWORD
```

20. 如果要在 Kaa 群集中启用 API 安全性（推荐），则应为服务创建 auth 服务器后端密码，以便能够请求 PAT 令牌。

```
export HISTCONTROL=ignorespace # Prevent saving your client ID and secret in the shell history; note the leading space in the next line
 cat << EOF > /tmp/keycloak-backend-secret.yaml
apiVersion: v1
data:
  client-id: < your client-id, base64-encoded >
  client-secret: < your client-secret, base64-encoded >
kind: Secret
metadata:
  name: "keycloak-backend"
  labels:
    app.kubernetes.io/name: "kaa-name"
    helm.sh/chart: "kaa-chart-version"
    app.kubernetes.io/instance: "kaa-instance-name"
    app.kubernetes.io/managed-by: "release-service-name"
EOF
kubectl create -f /tmp/keycloak-backend-secret.yaml
```

37. 添加 KaaIoT Helm 存储库：

```
38.    helm repo add kaa-museum https://museum.kaaiot.net/
```

3.2.1.2 **安装**

完成准备步骤后，一切就绪，即可在 Kubernetes 集群上部署 EPMX。要部署该服务，请运行以下命令（遵守对先前创建的许可证密钥的引用）：

```
   helm install --set global.license.secretName=license kaa-museum/epmx --name
kaa-epmx
```

检查 Pod 是否正在运行：

```
   kubectl get pods
```

服务初始化完成后，您应该观察到类似于以下内容的输出：

```
   NAME                                      READY   STATUS             RESTARTS
   AGE

   ...

   kaa-epmx                                   1/1     Running            0
    2m

   ...
```

3.2.2 图表要求

资料库	名称	版
@稳定	ts	2.0.6

3.2.3 图表值

键	类型	默认	描述
亲和力	宾语	{}	
注解。部署	宾语	{}	
注解.pod	宾语	{}	
配置	串	""	服务配置映射的内容，作为配置文件自动安装到 Pod 中。

键	类型	默认	描述
环保	宾语	`{}`	定义 Kubernetes 传递给服务副本的环境变量。
extraPodSpecs.automountServiceAccountToken	布尔	`false`	
fullnameOverride	串	`""`	
global.image.pullSecrets	清单	`[]`	图像拉取秘密名称列表。每个 `name` 字段都必须定义为一条记录。覆盖 `image.pullSecrets`。
global.keycloak.backendSecretName	串	`""`	后端身份验证服务器密钥的名称，必须包含 base64 编码的 `client-id` 和 `client-secret`。需要。
global.keycloak.baseUrl	串	`""`	身份验证提供程序的基本 URL。需要。
global.keycloak.enabled	串	`""`	使用 auth 提供程序启用 API 安全性。覆盖 `keycloak.enabled`。
global.keycloak.realm	串	`""`	身份验证提供程序领域。需要。
global.license.secretName	串	`""`	许可证密钥的名称，必须包含两个 base64 编码的字段：`file`（PKCS＃12 中的许可证文件内容）和 `password`。覆盖 `license.secretName`。
global.nats.url	串	`""`	NATS URL。覆盖 `nats.url`。
global.tekton.enabled	串	`""`	启用 Tekton 集成。禁用后，该服务将期望在配置映射中定义 Kaa 应用程序配置。覆盖 `tekton.enabled`。
global.tekton.url	串	`""`	Tekton URL。覆盖 `tekton.url`。

键	类型	默认	描述
image.pullPolicy	串	`"IfNotPresent"`	Docker 映像拉取策略。
image.pullSecrets	清单	`[]`	图像拉取秘密名称列表。每个 `name` 字段都必须定义为一条记录。
映像库	串	`"hub.kaaiot.net/core/service/epmx/epmx"`	Docker 映像存储库映像 URL。
image.tag	串	`""`	Docker 图像标签版本可以拉出并运行。
启用 keycloak	串	`""`	使用 auth 提供程序启用 API 安全性。覆盖 `global.keycloak.enabled`。
license.secretName	串	`""`	许可证密钥的名称，必须包含两个 base64 编码的字段：`file`（PKCS＃12 中的许可证文件内容）和 `password`。
元数据组件	串	`"backend"`	
metadata.partOf	串	`"kaa"`	
nameOverride	串	`""`	
nats.auth.enabled	布尔	`false`	
nats.enabled	布尔	`true`	启用 NATS 依赖性。有关详细信息，请参阅 https://github.com/helm/charts/tree/master/stable/nats。
nats.url	串	`"nats://-nats-client:4222"`	NATS URL。
nats.urlOverride	串	`""`	NATS URL。替代 `global.nats.url` 和 `nats.url`。
nodeSelector	宾语	`{}`	

键	类型	默认	描述
probes.enabled	布尔	`true`	为容器启用活动性，就绪性和启动探针。
probes.liveness.initialDelaySeconds	整型	`90`	
probes.liveness.periodSeconds	整型	`3`	
probes.readiness.initialDelaySeconds	整型	`90`	
probes.readiness.periodSeconds	整型	`1`	
复制计数	整型	`1`	要运行的服务实例副本的数量。
资源	宾语	`{}`	
securityContext	宾语	`{}`	
启用 tekton	串	`"false"`	启用 Tekton 集成。禁用后，该服务将期望在中定义 Kaa 应用程序配置 `config`。
tekton.url	串	`"http://-tekton"`	Tekton URL。
tekton.urlOverride	串	`""`	Tekton URL。替代 `global.tekton.url` 和 `tekton.url`。
TerminationMessagePolicy	串	`"FallbackToLogsOnError"`	Kubernetes 终止消息策略。
宽容	清单	`[]`	
updateStrategy.type	串	`"RollingUpdate"`	部署更新策略。
waitContainers.enabled	布尔	`true`	等待依赖服务。

键	类型	默认	描述
waitContainers.timeout	整型	300	等待依赖性服务超时（以秒为单位）。

没有描述的键是标准的 Kubernetes 值。有关这些的更多信息，请参考官方的 Kubernetes 文档。

3.2.4 环境变量

下表总结了 EPMX Docker 映像支持的变量，并提供了默认值和说明。

变量名	默认值	描述
INSTANCE_NAME	**epmx**	服务实例名称。
APP_CONFIG_PATH	**/srv/epmx/service-config.yml**	容器内服务配置 YAML 文件的路径。如果在 Kubernetes 中运行，请考虑使用 K8s Volumes 进行外部化。
KAA_TEKTON_ENABLED	**假**	启用 Tekton 集成。
KAA_TEKTON_URL	**http：// tekton**	Tekton 服务的 URL。
JMX_ENABLE	**假**	启用 JMX 监视。
JMX_PORT	**10500**	JMX 服务端口。
JMX_MONITOR_USER_PASSWORD		JMX 监视器用户密码。何时需要 JMX_ENABLE=true。
JAVA_OPTIONS	**-Xmx700m**	Java 进程启动的其他参数。

变量名	默认值	描述
NATS_USERNAME		用于连接到 NATS 消息代理的用户名。
NATS_PASSWORD		连接到 NATS 消息代理的密码。
KAA_LICENSE_CERT_PATH	**/run/license/license.p12**	PKCS # 12 格式的 Kaa 平台许可证证书文件的路径。
KAA_LICENSE_CERT_PASSWORD		许可证证书密码。需要。

3.3 组态

- 服务配置结构
  - Kaa 应用
  - Tekton
  - 端点寄存器接口
  - 导航键
  - 管理
  - 认证方式
  - 记录中
- 内置配置文件
  - 默认

EPMX 从文件（通常由 Kubernetes 配置图支持）加载配置数据。如果该服务无法检索配置数据，它将不会启动。有关详细信息，请参考配置。

3.3.1 服务配置结构

推荐的配置格式为 YAML。以下各节描述了影响服务功能各个方面的正式支持的配置选项。该服务可能支持以下列出的选项以外的其他选项，但这些选项不是公共 API 的一部分，可以随时更改或删除。

3.3.1.1 **Kaa 应用**

可以将许多 Kaa 服务配置为具有不同的行为，具体取决于处理的数据所涉及的端点的应用程序版本。这称为*特定于 Appversion 的行为*，并在的服务配置中进行处理 `kaa.applications`。或者，可以从 Kaa Tekton 中获取特定于应用程序的配置。请参阅“ Tekton 配置”部分，以了解如何配置此类集成。

```
kaa:
  applications:
    <application 1 name>:
      versions:
        <application 1 version 1 name>:
          # List of metadata keys that are allowed for endpoint to read. Optional. When
not specified, all metadata keys
          # are allowed for both reading and writing.
          metadata-keys:
            <metadata-key-name-1>:      # Name of the metadata key
              write-enabled: <boolean>  # Indicates whether writing is allowed for endpo
int. Defaults to true.
            <metadata-key-name-2>:
            ...
```

例如：

```
kaa:
  applications:
    smart_kettle:
      versions:
        kettle_v1:                # "kettle_v1" application version is configure
d with no key filtering
        kettle_v2:                # "kettle_v2" allows only "location" and "owne
r" metadata fields
          metadata-keys:
            location:             # "location" field is write-enabled
            owner:
              write-enabled: false  # "owner" field is read-only
```

```
      kettle_v3:                # "kettle_v3" explicitly specifies no filterin
g
        metadata-keys: null
```

3.3.1.2 Tekton

EPMX 支持与 Kaa Tekton 集成，以进行集中的应用程序配置管理。以下配置选项设置了集成界面。

```
kaa:
  tekton:
    enabled: <boolean>    # Enables Tekton integration. False by default. Also c
an be set with the KAA_TEKTON_ENABLED environment variable.
    url: <string>         # URL of the Tekton service. "http://tekton" by defaul
t. Also can be set with the KAA_TEKTON_URL environment variable.
  scmp.consumer:
    provider.service-instance.name: <string>  # Service instance name of the Tek
ton service. "tekton" by default. Also can be set with the KAA_SCMP_CONSUMER_PRO
VIDER_SERVICE_INSTANCE_NAME environment variable.
```

3.3.1.3 端点寄存器接口

使用以下参数来配置 EPMX 用来检索和管理端点元数据的基于 REST 的端点寄存器接口。

```
kaa:
  epmx:
    ep-register:
      base-url: <URL>  # Base URL of the endpoint register REST API
```

3.3.1.4 导航键

以下参数配置 EPMX 与 NATS 的连接。请注意，出于安全原因，NATS 用户名和密码均来自环境变量。

```
nats:
  urls: <comma separated list of URL>  # NATS connection URLs
```

3.3.1.5 管理

EPMX 监视和管理实现基于 Spring Boot Actuator。有关受支持的配置选项的列表，请参阅相应的文档。

3.3.1.6 认证方式

EPMX 使用 auth-kaa-client-starter 进行经过身份验证的 REST API 调用。根据 oAuth 协议实施的身份验证。

```
security:
  oauth2:
    client:
```

```
      enabled: <boolean>            # Is authentication enable for EPMX REST cal
ls. False by default
      base-url: <URL>              # Base host url to oAuth provider (e.g. Keycl
oak)
      realm: <realm name>           # Realm in oAuth provider
      clientId: <client ID>         # Client ID on whose behalf the requests wil
l be made
      clientSecret: <client secret>  # Client secret on whose behalf the request
s will be made
```

3.3.1.7 记录中

默认情况下，EPMX 使用 Spring Boot 日志配置和 logback 进行日志记录。有关受支持的配置选项的列表，请参阅相应的文档。

3.3.2 内置配置文件

为方便起见，EPMX 带有默认的内置配置文件。
内置配置文件针对基于 Kubernetes 的生产部署进行了优化。它们没有定义任何 Kaa 应用程序-您必须为基于 Kaa 的特定解决方案配置它们。

3.3.2.1 默认

```
kaa:
  epmx:
    ep-register:
      base-url: http://epr


nats:
  urls: nats://nats:4222


management:
  port: 8080
Show more
```

4 EPR

4.1 总览

- 介面
  - EP 和元数据管理
  - EP 过滤器
  - EP 生命周期和元数据事件
  - 端点连接事件
  - 凭证管理
  - EP 过滤器事件
  - Tekton 整合

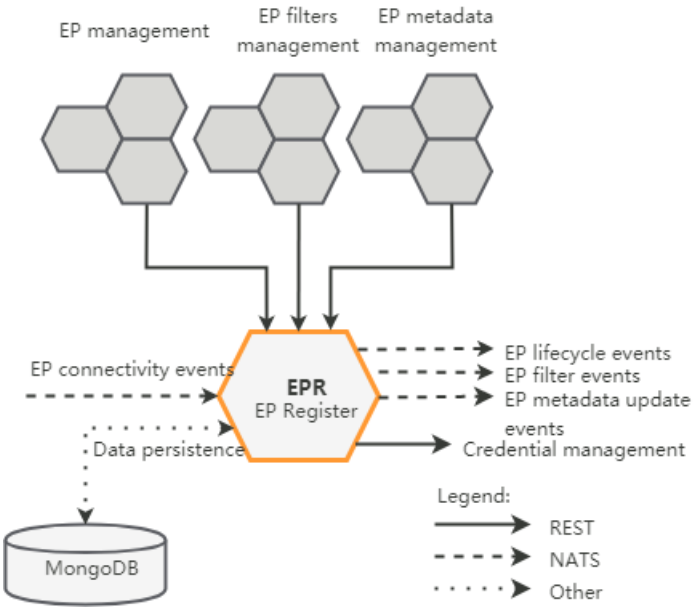
端点注册服务（EPR）是 Kaa 平台组件，用于记录解决方案中所有端点注册及其关联的键/值属性（称为元数据）。此服务提供 REST API 接口来管理端点和端点元数据密钥。

EPR 广播端点（EP）生命周期事件，例如端点的注册或删除以及任何 EP 元数据更新。

EPR 将所有与 EP 相关的数据持久保存到 MongoDB 中。

4.1.1 介面

EPR 支持许多接口来执行其功能角色。下图总结了主要支持的接口。



对于服务间通信，Kaa 服务主要使用在 NATS 消息传递系统上运行的 REST API 和消息传递协议。

4.1.2 EP 和元数据管理

EPR 提供了一个基于 REST 的界面来管理端点及其元数据。该接口允许注册端点，更新 EP 应用程序版本和 EP 元数据等等。

4.1.3 EP 过滤器

有关过滤 EP 的详细信息，请参阅 EP 过滤器文档。

4.1.4 EP 生命周期和元数据事件

EPR 广播以下事件：

- 端点已注册/未注册，EP 应用程序版本已更新。参见 9 / ELCE。
- 端点元数据已更新。参见 15 / EME。

4.1.5 端点连接事件

EPR 侦听 9 / ELCE 中定义的端点连接事件。当端点连接到平台时，EPR 将其报告的应用程序版本与存储的版本进行比较，如果有更新，则在数据库中对其进行更新，并广播端点应用程序版本已更新事件。

4.1.6 凭证管理

EPR 使用基于 REST 的凭据管理 API 为新创建的端点设置初始令牌。

4.1.7 EP 过滤器事件

EPR 广播 18 / EFE 中定义的以下 EP 过滤器事件：

- 端点过滤器已激活/已停用
- 端点匹配/不匹配过滤器

4.1.8 Tekton 整合

EPR 与 Kaa Tekton 集成在一起，用于集中式应用程序配置管理。它通过 17 / SCMP 从 Tekton 接收配置更新消息，并使用 Tekton REST API 检索当前配置。

有关更多信息，请参见配置。

4.2 REST API

4.2.1 端点注册 REST API 文档版本 v1

{schema}：// {host} / api / {version}

- **模式**：*必填（http，https 之一-默认：http）*
- **host**：*必填（字符串-默认值：localhost）*
- **版本**：*必填（v1）*

- **终点**
- **筛选器**
- **应用领域**

4.2.2 终点

端点上的操作，元数据管理。

/端点发布得到
/ endpoints / {endpointId}获取
/ endpoints / {endpointId} / app-version 获取放
/ endpoints / {endpointId} /元数据获取放 补丁
/ endpoints / {endpointId} / metadata / {metadataKey}获取放 删除
/ endpoints / {endpointId} /元数据键

4.3 部署方式

所有 Kaa 服务（包括 EPR）均以 Helm 图表的形式分发。您可以使用 Kubernetes 运行这些图表。

4.3.1 在 Kubernetes 上安装 EPR 图表

4.3.1.1 制备

对于 Kubernetes 中的整个 Kaa 集群，这些步骤应该执行一次。

1. 安装 Kubernetes。
2. 安装 Helm 客户端和 Tiller 服务器。
3. 创建一个 Kaa 许可证密钥（记住要输入您的 Kaa 许可证密钥文件的内容和密码）：

```
4.  export HISTCONTROL=ignorespace # Prevent saving your key password in the
    shell history; note the leading space in the next line
5.   cat << EOF > /tmp/kaa-licence.yaml
```

```
apiVersion: v1
data:
  file: < your licence key file contents, base64-encoded >
  password: < your licence key password >
kind: Secret
metadata:
  name: license
  type: Opaque
EOF
kubectl create -f /tmp/kaa-licence.yaml
```

16. 为官方的 KaaIoT 泊坞窗注册表指定镜像拉密钥。要定义此机密，请使用您的 KaaID 凭据：

```
export HISTCONTROL=ignorespace  # Prevent saving your credentials in the shell history; note the leading space in the next line
 export KAAID_EMAIL=<your KaaID email, eg. bob@example.com> KAAID_PASSWORD=<your KaaID password>
kubectl create secret docker-registry kaaid --docker-server=hub.kaaiot.net --docker-username=$KAAID_EMAIL --docker-email=$KAAID_EMAIL --docker-password=$KAAID_PASSWORD
```

20. 如果要在 Kaa 群集中启用 API 安全性（推荐），则应为服务创建 auth 服务器后端密码，以便能够请求 PAT 令牌。

```
export HISTCONTROL=ignorespace # Prevent saving your client ID amd secret in the shell history; note the leading space in the next line
 cat << EOF > /tmp/keycloak-backend-secret.yaml
apiVersion: v1
data:
  client-id: < your client-id, base64-encoded >
  client-secret: < your client-secret, base64-encoded >
kind: Secret
metadata:
  name: "keycloak-backend"
  labels:
    app.kubernetes.io/name: "kaa-name"
    helm.sh/chart: "kaa-chart-version"
    app.kubernetes.io/instance: "kaa-instance-name"
    app.kubernetes.io/managed-by: "release-service-name"
EOF
kubectl create -f /tmp/keycloak-backend-secret.yaml
```

37. 添加 KaaIoT Helm 存储库：

```
38.  helm repo add kaa-museum https://museum.kaaiot.net/
```

4.3.1.2 **安装**

完成准备步骤后，一切就绪，可以在您的 Kubernetes 集群上部署 EPR。要部署该服务，请运行以下命令（遵守对先前创建的许可证密钥的引用）：

```
helm install --set global.license.secretName=license kaa-museum/epr --name kaa-epr
```

检查 Pod 是否正在运行：

```
kubectl get pods
```

服务初始化完成后，您应该观察到类似于以下内容的输出：

```
NAME                                    READY   STATUS             RESTARTS
AGE

...

kaa-epr                                 1/1     Running            0          2
m

...
```

4.3.2 图表要求

资料库	名称	版
@稳定	mongodb	4.0.5
@稳定	ts	2.0.6

4.3.3 图表值

键	类型	默认	描述
亲和力	宾语	{}	

键	类型	默认	描述
注解。部署	宾语	`{}`	
注解.pod	宾语	`{}`	
配置	串	`""`	服务配置映射的内容，作为配置文件自动安装到 Pod 中。
环保	宾语	`{}`	定义 Kubernetes 传递给服务副本的环境变量。
extraPodSpecs.automountServiceAccountToken	布尔	`false`	
fullnameOverride	串	`""`	
global.image.pullSecrets	清单	`[]`	图像拉取秘密名称列表。每个 `name` 字段都必须定义为一条记录。覆盖 `image.pullSecrets`。
global.keycloak.backendSecretName	串	`""`	后端身份验证服务器密钥的名称，必须包含 base64 编码的 `client-id` 和 `client-secret`。需要。
global.keycloak.baseUrl	串	`""`	身份验证提供程序的基本 URL。需要。
global.keycloak.enabled	串	`""`	使用 auth 提供程序启用 API 安全性。覆盖 `keycloak.enabled`。
global.keycloak.realm	串	`""`	身份验证提供程序领域。需要。

键	类型	默认	描述
global.license.secretName	串	`""`	许可证密钥的名称，必须包含两个 base64 编码的字段：`file`（PKCS＃12 中的许可证文件内容）和 `password`。覆盖 `license.secretName`。
global.mongodb.url	串	`""`	Mongodb URL。覆盖 `mongodb.url`。
global.nats.url	串	`""`	NATS URL。覆盖 `nats.url`。
global.tekton.enabled	串	`""`	启用 Tekton 集成。禁用后，该服务将期望在配置映射中定义 Kaa 应用程序配置。覆盖 `tekton.enabled`。
global.tekton.url	串	`""`	Tekton URL。覆盖 `tekton.url`。
image.pullPolicy	串	`"IfNotPresent"`	Docker 映像拉取策略。
image.pullSecrets	清单	`[]`	图像拉取秘密名称列表。每个 `name` 字段都必须定义为一条记录。
映像库	串	`"hub.kaaiot.net/core/service/epr/epr"`	Docker 映像存储库映像 URL。
image.tag	串	`""`	Docker 图像标签版本可以拉出并运行。
入口注释	宾语	`{}`	
启用入口	布尔	`false`	

键	类型	默认	描述
ingress.hosts [0]	串	`"chart-example.local"`	
入口路径	清单	`[]`	
ingress.tls	清单	`[]`	
启用 keycloak	串	`""`	使用 auth 提供程序启用 API 安全性。覆盖 `global.keycloak.enabled`。
license.secretName	串	`""`	许可证密钥的名称，必须包含两个 base64 编码的字段：`file`（PKCS＃12 中的许可证文件内容）和 `password`。
元数据组件	串	`"backend"`	
metadata.partOf	串	`"kaa"`	
mongodb.enabled	布尔	`true`	有关详细信息，请查看 https://github.com/helm/charts/tree/master/stable/mongodb
mongodb.image.pullPolicy	串	`"IfNotPresent"`	
mongodb.image.tag	串	`"3.6.10"`	
mongodb.persistence.enabled	布尔	`false`	
mongodb.url	串	`"mongodb://-:27017"`	Mongodb URL。

键	类型	默认	描述
mongodb.urlOverride	串	`""`	Mongodb URL。替代 `global.mongodb.url` 和 `mongodb.url`。
mongodb.usePassword	布尔	`false`	
nameOverride	串	`""`	
nats.auth.enabled	布尔	`false`	
nats.enabled	布尔	`true`	启用 NATS 依赖性。有关详细信息，请参阅 https://github.com/helm/charts/tree/master/stable/nats。
nats.url	串	`"nats://-nats-client:4222"`	NATS URL。
nats.urlOverride	串	`""`	NATS URL。替代 `global.nats.url` 和 `nats.url`。
nodeSelector	宾语	`{}`	
probes.enabled	布尔	`true`	为容器启用活动性，就绪性和启动探针。
probes.liveness.initialDelaySeconds	整型	`90`	
probes.liveness.periodSeconds	整型	`3`	

键	类型	默认	描述
probes.readiness.initialDelaySeconds	整型	`90`	
probes.readiness.periodSeconds	整型	`1`	
复制计数	整型	`1`	要运行的服务实例副本的数量。
资源	宾语	`{}`	
securityContext	宾语	`{}`	
service.externalIPs	清单	`[]`	
service.loadBalancerIP	串	`""`	
服务端口	整型	`80`	
服务类型	串	`"ClusterIP"`	
启用 tekton	串	`"false"`	启用 Tekton 集成。禁用后，该服务将期望在中定义 Kaa 应用程序配置 `config`。
tekton.url	串	`"http://-tekton"`	Tekton URL。
tekton.urlOverride	串	`""`	Tekton URL。替代 `global.tekton.url` 和 `tekton.url`。

键	类型	默认	描述
TerminationMessagePolicy	串	`"FallbackToLogsOnError"`	Kubernetes 终止消息策略。
宽容	清单	`[]`	
updateStrategy.type	串	`"RollingUpdate"`	部署更新策略
waitContainers.enabled	布尔	`true`	等待依赖服务。
waitContainers.timeout	整型	`300`	等待依赖性服务超时（以秒为单位）。

没有描述的键是标准的 Kubernetes 值。有关这些的更多信息，请参考官方的 Kubernetes 文档。

4.3.4 环境变量

下表总结了 EPR Docker 映像支持的变量，并提供了默认值和说明。

变量名	默认值	描述
`INSTANCE_NAME`	**epr**	服务实例名称。
`APP_CONFIG_PATH`	**/srv/epr/service-config.yml**	容器内服务配置 YAML 文件的路径。如果在 Kubernetes 中运行，请考虑使用 K8s Volumes 进行外部化。
`KAA_TEKTON_ENABLED`	**假**	启用 Tekton 集成。
`KAA_TEKTON_URL`	**http：// tekton**	Tekton 服务的 URL。

变量名	默认值	描述
`JMX_ENABLE`	**假**	启用 JMX 监视。
`JMX_PORT`	**10500**	JMX 服务端口。
`JMX_MONITOR_USER_PASSWORD`		JMX 监视器用户密码。何时需要 `JMX_ENABLE=true`。
`JAVA_OPTIONS`	**-Xmx700m**	Java 进程启动的其他参数。
`NATS_USERNAME`		用于连接到 NATS 消息代理的用户名。
`NATS_PASSWORD`		连接到 NATS 消息代理的密码。
`KAA_LICENSE_CERT_PATH`	**/run/license/license.p12**	PKCS＃12 格式的 Kaa 平台许可证证书文件的路径。
`KAA_LICENSE_CERT_PASSWORD`		许可证证书密码。需要。

4.4 组态

EPR 从文件（通常由 Kubernetes 配置图支持）加载配置数据。如果该服务无法检索配置数据，它将不会启动。有关详细信息，请参考配置。

4.4.1 服务配置结构

推荐的配置格式为 YAML。以下各节描述了影响服务功能各个方面的正式支持的配置选项。该服务可能支持以下列出的选项以外的其他选项，但这些选项不是公共 API 的一部分，可以随时更改或删除。

4.4.1.1 常规服务配置

EPR 使用标准的 Spring Boot `server.port` 属性配置端口以公开 REST API。

```
server:
  port: <unsigned short integer>  # Server port used to expose the REST API at
```

4.4.1.2 Kaa 应用

可以将许多 Kaa 服务配置为具有不同的行为，具体取决于处理的数据所涉及的端点的应用程序版本。这称为*特定于 Appversion 的行为*，并在的服务配置中进行处理 `kaa.applications`。或者，可以从 Kaa Tekton 中获取特定于应用程序的配置。请参阅“ Tekton 配置”部分，以了解如何配置此类集成。
尽管 EPR 当前不支持任何特定于 Appversion 的配置，但它使用应用程序和 Appversion 名称来验证 API 调用和协议数据。

```
kaa:
  applications:
    <application 1 name>:               # Kaa application name
      versions:
        <application 1 version 1 name>:  # Kaa application version name
```

4.4.1.3 Tekton

EPR 支持与 Kaa Tekton 集成，以进行集中的应用程序配置管理。以下配置选项设置了集成界面。

```
kaa:
  tekton:
```

```
    enabled: <boolean>    # Enables Tekton integration. False by default. Also c
an be set with the KAA_TEKTON_ENABLED environment variable.
    url: <string>         # URL of the Tekton service. "http://tekton" by defaul
t. Also can be set with the KAA_TEKTON_URL environment variable.
  scmp.consumer:
    provider.service-instance.name: <string>  # Service instance name of the Tek
ton service. "tekton" by default. Also can be set with the KAA_SCMP_CONSUMER_PRO
VIDER_SERVICE_INSTANCE_NAME environment variable.
```

4.4.1.4 数据持久化接口

EPR 使用 Spring Data MongoDB 配置。有关受支持的配置选项的列表，请参阅相应的文档。

注意出于安全原因，用户名和密码必须来自环境变量。

4.4.1.5 导航键

以下参数配置 EPR 与 NATS 的连接。请注意，出于安全原因，NATS 用户名和密码均来自环境变量。

```
nats:
  urls: <comma separated list of URL>  # NATS connection URLs
```

4.4.1.6 通讯服务接口

```
kaa:
  epr:
    communication-service:
      service-instance:
        name: <service instance name>  # Instance name of the communication servi
ce
```

4.4.1.7 认证与授权

EPR REST API 安全性是根据带有 UMA 配置文件的 OAuth2 协议实现的。

通过以下配置选项控制保护：

```
security:
  ignored: <value>          # Controls authentication and authorization on REST
API endpoints.
                          # Possible values: '/**' to disable security or 'none
' to enable.
  oauth2:
    client:
```

```
      clientId: <clientId>   # Client ID on whose behalf the requests will be mad
e.
      clientSecret: <secret> # Client secret on whose behalf the requests will b
e made.
      accessTokenUri: <URI>  # An OAuth2-compliant Token Endpoint for obtaining
tokens via the 'Implicit Flow',
                             # 'Direct Grants', or 'Client Grants'.
                             # E.g. "https://your-url-to-auth-server/auth/realms/r
ealm-a/protocol/openid-connect/token"
      resourceUri: <URI>     # An UMA-compliant Resource Registration Endpoint w
hich resource servers can use
                             # to manage their protected resources and scopes.
                             # E.g. "https://your-url-to-auth-server/auth/realms/r
ealm-a/authz/protection/resource_set"
    resource:
      userInfoUri: <URI>     # URL endpoint for the User Info service described i
n the OIDC specification.
                             # E.g. "https://your-url-to-auth-server/auth/realms/r
ealm-a/protocol/openid-connect/userinfo"
```

4.4.1.8 **管理**

EPR 监视和管理实现基于 Spring Boot Actuator。有关受支持的配置选项的列表，请参阅相应的文档。

4.4.1.9 **记录中**

默认情况下，EPR 使用 Spring Boot 日志记录配置和 logback 进行日志记录。有关受支持的配置选项的列表，请参阅相应的文档。

4.4.2 内置配置文件

为了您的方便，EPR 带有默认的内置配置配置文件。
内置配置文件针对基于 Kubernetes 的生产部署进行了优化。它们没有定义任何 Kaa 应用程序-您必须为基于 Kaa 的特定解决方案配置它们。

4.4.2.1 **默认**

```
server:
  port: 80
  compression:
```

```
    enabled: true
    mime-types: application/json

nats:
  urls: nats://nats:4222

kaa:
```

4.5 关键服务功能

本指南介绍了 EPR 的主要功能，以及如何在 IoT 应用程序中有效使用它们。

特征	描述
EP 过滤器	当查询 EPR REST API 以获得 EP 时，EP 过滤器用于过滤 EP。

4.5.1 EP 过滤器

- 过滤状态
- 筛选器限制
- MongoDB 查询示例

当查询 EPR REST API 以获得 EP 时，EP 过滤器用于过滤 EP。EP 过滤器是专用的。每个过滤器都有一个在应用程序范围内由 EPR 在过滤器配置时分配的唯一 ID。对于每个 EP 过滤器，都会创建相应的 MongoDB 视图 `filter-{filterId}`。
可以使用功能强大的查询来指定过滤器，这些查询可以引用以下值：

- 端点元数据；
- 端点应用程序版本名称。

每当创建 EP 过滤器时，EPR 都会对应用程序的所有 EP 进行过滤器匹配的异步重新计算。每当更新 EP 应用程序版本或 EP 元数据时，都会重新评估该 EP 的过滤器匹配/不匹配。
EPR 提供 REST API，用于管理端点过滤器和接收匹配端点的列表。

对于每个命名的过滤器，EPR 会将端点过滤器事件广播到不同的主题。端点过滤器事件指示端点何时匹配或不匹配过滤器（通常是由于元数据或应用程序版本名称更新）。请参阅 EP 过滤器事件。

4.5.1.1 过滤状态

由于异步特性，过滤器具有两种状态：`ACTIVE` 和 `PENDING`。
创建 EP 过滤器后，它们将处于 `PENDING` 状态，直到 EP 匹配过程完成。如果过滤器处于此状态，则将禁用过滤器的 EP 搜索。
EP 匹配完成后，过滤器会将状态更改为 `ACTIVE`。此状态表示过滤器已被激活并且可以正常使用。

4.5.1.2 筛选器限制

与 EP 过滤过程相关的一些限制：

- 不能在同一谓词条件中使用多个元数据键值，因为 MongoDB 不支持此类查询。`$ where` 运算符可用于此类过滤。例如，以下谓词无效：metadata.foo> metadata.bar
- 在谓词的一侧，单个条件必须是一个常数。另一方面-元数据密钥或应用程序版本
- mongo 查询引擎本机不支持加，减，除，乘等算术运算
- 仅支持 MongoDB 查询操作和函数

4.5.1.3 MongoDB 查询示例

假设 EP 已在应用版本中注册，`app1-1` 并具有如下所述的元数据：

```
{
  "key1":1,
  "key2":"value",
  "key3":true,
  "key4":{
    "v1":1,
    "v2":[1,2,3]
  },
  "key5":[
    {
      "val":5
    },
    100,
    500,
    ["val1"]
  ],
  "key6":null
}
```

仅按应用程序版本过滤

MongoDB 查询：

```
db.endpoint.find({
    application_version: 'app1-1'
});
```

按元数据键和应用程序版本过滤

MongoDB 查询：

```
db.endpoint.find({
    $and: [
        { metadata: { $elemMatch: { key: 'key4', "value.v1": { $gt: 0 } } } },
        { metadata: { $elemMatch: { key: 'key5', value: { $size: 4 } } } },
        { metadata: { $elemMatch: { key: 'key5', value: { $in: [100] } } } },
        { metadata: { $elemMatch: { key: 'key5', value: { $elemMatch: { $gt: 499 } } } } }
 },
        { metadata: { $elemMatch: { key: 'key5', "value.0.val": { $ne: 4 } } } },
        { $or: [
            { metadata: { $elemMatch: { key: 'key1', value: { $gt: 0 } } } },
            { metadata: { $elemMatch: { key: 'key3', value: true } } }] },
        { metadata: { $elemMatch: { key: 'key6', value: null } } },
        { metadata: { $elemMatch: { key: 'key2', value: { $type: 'string' } } } },
        { metadata: { $elemMatch: { key: 'key2', value: { $regex: /val.*/i } } } },
        { application_version: 'app1-1' }]
});
```