参考

这是 Kubernetes 文档的参考部分。

API 参考

- Kubernetes API 参考 v1.20。
- 使用 Kubernetes API - Kubernetes 的 API 概述

API 客户端库

如果您需要通过编程语言调用 Kubernetes API，您可以使用 客户端库。以下是官方支持的客户端库：

- Kubernetes Go 语言客户端库
- Kubernetes Python 语言客户端库
- Kubernetes Java 语言客户端库
- Kubernetes JavaScript 语言客户端库

CLI 参考

- kubectl - 主要的 CLI 工具，用于运行命令和管理 Kubernetes 集群。
  - JSONPath - 通过 kubectl 使用 JSONPath 表达式 的语法指南。
- kubeadm - 此 CLI 工具可轻松配置安全的 Kubernetes 集群。

组件参考

- kubelet - 在每个节点上运行的主 *节点代理* 。kubelet 采用一组 PodSpecs 并确保所描述的容器健康地运行。
- kube-apiserver - REST API，用于验证和配置 API 对象（如 Pod、服务或副本控制器等）的数据。
- kube-controller-manager - 一个守护进程，它嵌入到了 Kubernetes 的附带的核心控制循环。
- kube-proxy - 可进行简单的 TCP/UDP 流转发或针对一组后端执行轮流 TCP/UDP 转发。
- kube-scheduler - 一个调度程序，用于管理可用性、性能和容量。
  - kube-scheduler 策略
  - kube-scheduler 配置

设计文档

Kubernetes 功能的设计文档归档，不妨考虑从 Kubernetes 架构 和 Kubernetes 设计概述 开始阅读。

Kubernetes 问题和安全

Kubernetes 问题追踪

要报告安全问题，请遵循 Kubernetes 安全问题公开流程。

使用 GitHub Issues 跟踪 Kubernetes 编码工作和公开问题。

- CVE 相关问题

与安全性相关的公告请发送到 kubernetes-security-announce@googlegroups.com 邮件列表。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 04, 2020 at 6:22 PM PST: [zh] Fix links in zh localization (4) (abab517ff)

Kubernetes 安全和信息披露

本页面介绍 Kubernetes 安全和信息披露相关的内容。

安全公告

加入 kubernetes-security-announce 组，以获取关于安全性和主要 API 公告的电子邮件。

你也可以使用此链接 订阅上述的 RSS 反馈。

报告一个漏洞

我们非常感谢向 Kubernetes 开源社区报告漏洞的安全研究人员和用户。 所有的报告都由社区志愿者进行彻底调查。

如需报告，请连同安全细节以及预期的所有 Kubernetes bug 报告 详细信息电子邮件到 security@kubernetes.io列表。

你还可以通过电子邮件向私有 security@kubernetes.io 列表发送电子邮件，邮件中应该包含所有 Kubernetes 错误报告所需的详细信息。

你可以使用产品安全团队成员 的 GPG 密钥加密你的电子邮件到此列表。使用 GPG 加密不需要公开。

我应该在什么时候报告漏洞？

- 你认为在 Kubernetes 中发现了一个潜在的安全漏洞
- 你不确定漏洞如何影响 Kubernetes
- 你认为你在 Kubernetes 依赖的另一个项目中发现了一个漏洞
- 对于具有漏洞报告和披露流程的项目，请直接在该项目处报告

我什么时候不应该报告漏洞？

- 你需要帮助调整 Kubernetes 组件的安全性
- 你需要帮助应用与安全相关的更新
- 你的问题与安全无关

安全漏洞响应

每个报告在 3 个工作日内由产品安全团队成员确认和分析。这将启动安全发布过程。

与产品安全团队共享的任何漏洞信息都保留在 Kubernetes 项目中，除非有必要修复该问题，否则不会传播到其他项目。

随着安全问题从分类、识别修复、发布计划等方面的进展，我们将不断更新报告。

公开披露时间

公开披露日期由 Kubernetes 产品安全团队和 bug 提交者协商。我们倾向于在用户缓解措施可用时尽快完全披露该 bug。

当 bug 或其修复还没有被完全理解，解决方案没有经过良好的测试，或者为了处理供应商协调问题时，延迟披露是合理的。

信息披露的时间范围从即时（尤其是已经公开的）到几周。作为一个基本的约定，我们希望报告日期到披露日期的间隔是 7 天。在设置披露日期时，Kubernetes 产品安全团队拥有最终决定权。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 01, 2020 at 5:02 PM PST: [zh]Sync changes under `content/zh/docs/reference` folder for Issues kubernetes/website#25247. (#25308) (4f9b919a8)

使用 Kubernetes API

本文提供了 Kubernetes API 的参考信息。

REST API 是 Kubernetes 的基本结构。 所有操作和组件之间的通信及外部用户命令都是调用 API 服务器处理的 REST API。 因此，Kubernetes 平台视一切皆为 API 对象， 且它们在 API 中有相应的定义。

Kubernetes API 参考列 出了 Kubernetes v1.20 版本的 API。

如需了解一般背景信息，请查阅 Kubernetes API。 Kubernetes API 控制访问描述了客户端如何 向 Kubernetes API 服务器进行身份认证以及他们的请求如何被鉴权。

API 版本控制

JSON 和 Protobuf 序列化模式遵循相同的模式更改原则。 以下描述涵盖了这两种格式。

API 版本控制和软件版本控制是间接相关的。 API 和发布版本控制提案 描述了 API 版本控制和软件版本控制间的关系。

不同的 API 版本代表着不同的稳定性和支持级别。 你可以在 [API 变更文档]https://git.k8s.io/community/contributors/devel/sig-architecture/api_changes.md#alpha-beta-and-stable-versions) 中查看到更多的不同级别的判定标准。

下面是每个级别的摘要：

<--

- Alpha:
  - The version names contain `alpha` (for example, `v1alpha1`).
  - The software may contain bugs. Enabling a feature may expose bugs. A feature may be disabled by default.
  - The support for a feature may be dropped at any time without notice.
  - The API may change in incompatible ways in a later software release without notice.
  - The software is recommended for use only in short-lived testing clusters, due to increased risk of bugs and lack of long-term support. -->
- Alpha:
  - 版本名称包含 `alpha`（例如，`v1alpha1`）。

    - 软件可能会有 Bug。启用某个特性可能会暴露出 Bug。 某些特性可能默认禁用。
    - 对某个特性的支持可能会随时被删除，恕不另行通知。
    - API 可能在以后的软件版本中以不兼容的方式更改，恕不另行通知。
    - 由于缺陷风险增加和缺乏长期支持，建议该软件仅用于短期测试集群。

- Beta:
    - 版本名称包含 beta （例如， v2beta3 ）。
    - 软件被很好的测试过。启用某个特性被认为是安全的。 特性默认开启。
    - 尽管一些特性会发生细节上的变化，但它们将会被长期支持。

- 在随后的 Beta 版或稳定版中，对象的模式和（或）语义可能以不兼容的方式改变。 当这种情况发生时，将提供迁移说明。 模式更改可能需要删除、编辑和重建 API 对象。 编辑过程可能并不简单。 对于依赖此功能的应用程序，可能需要停机迁移。
- 该版本的软件不建议生产使用。 后续发布版本可能会有不兼容的变动。 如果你有多个集群可以独立升级，可以放宽这一限制。

  说明： 请试用测试版特性时并提供反馈。特性完成 Beta 阶段测试后， 就可能不会有太多的变更了。

- Stable:
    - 版本名称如 vX，其中 X 为整数。
    - 特性的稳定版本会出现在后续很多版本的发布软件中。

API 组

API 组 能够简化对 Kubernetes API 的扩展。 API 组信息出现在REST 路径中，也出现在序列化对象的 apiVersion 字段中。

以下是 Kubernetes 中的几个组：

- *核心*（也叫 *legacy*）组的 REST 路径为 /api/v1。 核心组并不作为 apiVersion 字段的一部分，例如， apiVersion: v1。
- 指定的组位于 REST 路径 /apis/$GROUP_NAME/$VERSION， 并且使用 apiVersion: $GROUP_NAME/$VERSION （例如， apiVersion: batch/v1 ）。 你可以在 Kubernetes API 参考文档 中查看全部的 API 组。

启用或禁用 API 组 {#enabling-or-disabling} 资源和 API 组是在默认情况下被启用的。 你可以通过在 API 服务器上设置 --runtime-config 参数来启用或禁用它们。 --runtime-config 参数接受逗号分隔的 <key>[=<value>] 对， 来描述 API 服务器的运行时配置。如果省略了 =<value> 部分，那么视其指定为 =true。 例如：

- 禁用 batch/v1， 对应参数设置 --runtime-config=batch/v1=false
- 启用 batch/v2alpha1， 对应参数设置 --runtime-config=batch/v2alpha1

  说明： 启用或禁用组或资源时， 你需要重启 API 服务器和控制器管理器来使 --runtime-config 生效。

持久化

Kubernetes 通过 API 资源来将序列化的状态写到 etcd 中存储。

接下来

- 进一步了解 API 惯例
- 阅读 聚合器

Kubernetes API 概念

服务器端应用（Server-Side Apply）

客户端库

Kubernetes 弃用策略

Kubernetes API 健康端点

Kubernetes API 概念

本页描述 Kubernetes API 的通用概念。

Kubernetes API 是基于资源的（RESTful）、通过 HTTP 提供的编程接口。 API 支持通过标准的 HTTP 动词（POST、PUT、PATCH、DELETE 和 GET）检视、创建、更新和删除主要资源，为很多允许细粒度权限控制的对象提供子资源（如将 Pod 绑定到节点上），并且出于便利性或效率考虑，支持并提供这些资源的 不同表示形式。 Kubernetes API 还通过 "watch" 和一致性的列表支持高效的资源变更通知， 从而允许其他组件对资源的状态进行高效的缓存和同步。

标准 API 术语

大多数 Kubernetes API 资源类型都是 对象： 它们代表的是集群中某一概念的具体实例，例如一个 Pod 或名字空间。 为数不多的几个 API 资源类型是"虚拟的" - 它们通常代表的是操作而非对象本身， 例如访问权限检查（使用 POST 请求发送一个 JSON 编码的 SubjectAccessReview 负载到 subjectaccessreviews 资源）。 所有对象都有一个唯一的名字，以便支持幂等的创建和检视操作，不过如果虚拟资源类型 不可检视或者不要求幂等，可以不具有唯一的名字。

Kubernetes 一般会利用标准的 RESTful 术语来描述 API 概念：

- **资源类型（Resource Type）** 是在 URL 中使用的名称（pods、namespaces、services）
- 所有资源类型都有具有一个 JSON 形式（其对象的模式定义）的具体表示，称作**类别（Kind）**
- 某资源类型的实例的列表称作 **集合（Collection）**

- 资源类型的单个实例被称作 **资源（Resource）**

所有资源类型要么是集群作用域的（/apis/GROUP/VERSION/*），要么是名字空间 作用域的（/apis/GROUP/VERSION/namespaces/NAMESPACE/*）。 名字空间作用域的资源类型会在其名字空间被删除时也被删除，并且对该资源类型的 访问是由定义在名字空间域中的授权检查来控制的。 下列路径用来检视集合和资源：

- 集群作用域的资源：
  - GET /apis/GROUP/VERSION/RESOURCETYPE - 返回指定资源类型的资源的集合
  - GET /apis/GROUP/VERSION/RESOURCETYPE/NAME - 返回指定资源类型下名称为 NAME 的资源
- 名字空间作用域的资源：
  - GET /apis/GROUP/VERSION/RESOURCETYPE - 返回所有名字空间中指定资源类型的全部实例的集合
  - GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE - 返回名字空间 NAMESPACE 内给定资源类型的全部实例的集合
  - GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE/NAME - 返回名字空间 NAMESPACE 中给定资源类型的名称为 NAME 的实例

由于名字空间本身是一个集群作用域的资源类型，你可以通过 GET /api/v1/namespaces/ 检视所有名字空间的列表，使用 GET /api/v1/namespaces/NAME 查看特定名字空间的 详细信息。

几乎所有对象资源类型都支持标准的 HTTP 动词 - GET、POST、PUT、PATCH 和 DELETE。 Kubernetes 使用术语 **list** 来描述返回资源集合的操作，以便与返回单个资源的、 通常称作 **get** 的操作相区分。

某些资源类型有一个或多个子资源（Sub-resource），表现为对应资源下面的子路径：

- 集群作用域的子资源：GET /apis/GROUP/VERSION/RESOURCETYPE/NAME/SUBRESOURCE
- 名字空间作用域的子资源：GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE/NAME/SUBRESOURCE

取决于对象是什么，每个子资源所支持的动词有所不同 - 参见 API 文档以了解更多信息。 跨多个资源来访问其子资源是不可能的 - 如果需要这一能力，则通常意味着需要一种 新的虚拟资源类型了。

高效检测变更

为了使客户端能够构造一个模型来表达集群的当前状态，所有 Kubernetes 对象资源类型 都需要支持一致的列表和一个称作 **watch** 的增量变更通知信源（feed）。 每个 Kubernetes 对象都有一个 resourceVersion 字段，代表该资源在下层数据库中 存储的版本。检视资源集合（名字空间作用域或集群作用域）时，服务器返回的响应 中会包含 resourceVersion 值，可用来向服务器发起 watch 请求。 服务器会返回所提供的 resourceVersion 之后发生的所有变更（创建、删除和更新）。 这使得客户端能够取回当前的状态并监视其变更，且不会错过任何变更事件。 客户端的监视连接被断开时，可以从最

后返回的 resourceVersion 重启新的监视连接， 或者执行一个新的集合请求之后从头开始监视操作。 参阅资源版本语义以了解更多细节。

例如：

1. 列举给定名字空间中的所有 Pods：

```
GET /api/v1/namespaces/test/pods
---
200 OK
Content-Type: application/json
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {"resourceVersion":"10245"},
  "items": [...]
}
```

1. 从资源版本 10245 开始，以 JSON 对象的形式接收所有创建、删除或更新操作的通知：

```
GET /api/v1/namespaces/test/pods?watch=1&resourceVersion=10245
---
200 OK
Transfer-Encoding: chunked
Content-Type: application/json

{
  "type": "ADDED",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"resourceVersion":
"10596", ...}, ...}
}
{
  "type": "MODIFIED",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"resourceVersion":
"11020", ...}, ...}
}
...
```

给定的 Kubernetes 服务器只会保留一定的时间内发生的历史变更列表。 使用 etcd3 的集群默认保存过去 5 分钟内发生的变更。 当所请求的 watch 操作因为资源的历史版本不存在而失败，客户端必须能够处理 因此而返回的状态代码 410 Gone，清空其本地的缓存，重新执行 list 操作， 并基于新的 list 操作所返回的 resourceVersion 来开始新的 watch 操作。 大多数客户端库都能够提供某种形式的、包含此逻辑的工具。 （在 Go 语言客户端库中，这一设施称作 Reflector，位于 k8s.io/client-go/cache 包中。）

监视书签

为了处理历史窗口过短的问题，我们引入了 bookmark（书签） 监视事件的概念。 该事件是一种特殊事件，用来标示客户端所请求的、指定的 resourceVersion 之前 的所有变更都以被发送。该事件中返回的对象是所请求的资源类型，但其中仅包含 resourceVersion 字段，例如：

```
GET /api/v1/namespaces/test/pods?
watch=1&resourceVersion=10245&allowWatchBookmarks=true
---
200 OK
Transfer-Encoding: chunked
Content-Type: application/json

{
  "type": "ADDED",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"resourceVersion":
"10596", ...}, ...}
}
...
{
  "type": "BOOKMARK",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"resourceVersion":
"12746"} }
}
```

通过在 watch 请求中设置 allowWatchBookmarks=true 选项，可以请求 bookmark 事件，但是客户端不能假定服务器端会按某特定时间间隔返回书签事件，甚至也不能 假定服务器一定会发送 bookmark 事件。

分块检视大体量结果

FEATURE STATE: Kubernetes v1.9 [beta]

在较大规模的集群中，检视某些资源类型的集合时可能会返回较大体量的响应数据，对服务器和客户端都会造成影响。例如，某集群可能包含数万个 Pod，每个 Pod 的 JSON 编码都有 1-2 KB 的大小。返回所有名字空间的全部 Pod 时，其结果可能体量很大（10-20 MB）且耗用大量的服务器资源。 从 Kubernetes 1.9 开始，服务器支持将单一的大体量集合请求分解成多个小数据块 同时还保证整个请求的一致性的能力。 各个数据块可以按顺序返回，进而降低请求的尺寸，允许面向用户的客户端以增量形式 呈现返回结果，改进系统响应效果。

为了用分块的形式返回一个列表，集合请求上可以设置两个新的参数 limit 和 continue，并且所有 list 操作的返回结果列表的 metadata 字段中会包含一个 新的 continue 字段。 客户端应该将 limit 设置为希望在每个数据块中收到的结果个数上限，而服务器则会在结果中至多返回 limit 个资源并在集合中还有更多资源的时候包含一个 continue 值。客户端在下次请求时则可以将此 continue 值传递给服务器， 告知后者要从何处开始

返回结果的下一个数据块。 通过重复这一操作直到服务器端返回空的 continue 值，客户端可以受到结果的 全集。

与 watch 操作类似，continue 令牌也会在很短的时间（默认为 5 分钟）内过期， 并在无法返回更多结果时返回 410 Gone 代码。 这时，客户端需要从头开始执行上述检视操作或者忽略 limit 参数。

例如，如果集群上有 1253 个 Pods，客户端希望每次收到包含至多 500 个 Pod 的 数据块，它应按下面的步骤来请求数据块：

1. 列举集群中所有 Pod，每次接收至多 500 个 Pods：

```
GET /api/v1/pods?limit=500
---
200 OK
Content-Type: application/json

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"10245",
    "continue": "ENCODED_CONTINUE_TOKEN",
    ...
  },
  "items": [...] // returns pods 1-500
}
```

1. 继续前面的调用，返回下一组 500 个 Pods：

```
GET /api/v1/pods?limit=500&continue=ENCODED_CONTINUE_TOKEN
---
200 OK
Content-Type: application/json

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"10245",
    "continue": "ENCODED_CONTINUE_TOKEN_2",
    ...
  },
  "items": [...] // returns pods 501-1000
}
```

1. 继续前面的调用，返回最后 253 个 Pods：

```
GET /api/v1/pods?limit=500&continue=ENCODED_CONTINUE_TOKEN_2
---
200 OK
Content-Type: application/json

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion":"10245",
    "continue": "", // continue token is empty because we have reached the end of the list
    ...
  },
  "items": [...] // returns pods 1001-1253
}
```

注意 list 操作的 resourceVersion 在每个请求中都设置的是同一个数值， 这表明服务器要向我们展示一个一致的 Pods 快照视图。 在版本 10245 之后创建、更新或删除的 Pods 都不会显示出来，除非用户发出 list 请求时不指定 continue 令牌。 这一设计使得客户端能够将较大的响应切分为较小的数据块，且能够对较大的集合 执行监视动作而不会错失任何更新事件。

以表格形式接收资源

kubectl get 命令的输出是一个包含一个或多个资源的简单表格形式。 过去，客户端需要重复 kubectl 中所实现的表格输出和描述输出逻辑，以执行 简单的对象列表操作。 这一方法在处理某些对象时，需要引入不容忽视的逻辑。 此外，API 聚合 和定制资源 所提供的资源类型都是编译时不可预知的。这意味着，客户端必须针对无法 识别的类型提供通用的实现逻辑。

为了避免上述各种潜在的局限性，客户端可以请求服务器端返回对象的表格（Table） 表现形式，从而将打印输出的特定细节委托给服务器。 Kubernetes API 实现标准的 HTTP 内容类型（Content Type）协商：为 GET 调用 传入一个值为 application/json;as=Table;g=meta.k8s.io;v=v1beta1 的 Accept 头部即可请求服务器以 Table 的内容类型返回对象。

例如，以 Table 格式列举集群中所有 Pods：

```
GET /api/v1/pods
Accept: application/json;as=Table;g=meta.k8s.io;v=v1beta1
---
200 OK
Content-Type: application/json

{
   "kind": "Table",
```

```
  "apiVersion": "meta.k8s.io/v1beta1",
  ...
  "columnDefinitions": [
     ...
  ]
}
```

对于在服务器上不存在定制的 Table 定义的 API 资源类型而言，服务器会返回 一个默认的 Table 响应，其中包含资源的 name 和 creationTimestamp 字段。

```
GET /apis/crd.example.com/v1alpha1/namespaces/default/resources
---
200 OK
Content-Type: application/json
...

{
  "kind": "Table",
  "apiVersion": "meta.k8s.io/v1beta1",
  ...
  "columnDefinitions": [
     {
        "name": "Name",
        "type": "string",
        ...
     },
     {
        "name": "Created At",
        "type": "date",
        ...
     }
  ]
}
```

kube-apiserver 从 1.10 版本开始提供 Table 响应。 因此，并非所有 API 资源类型都支持 Table 响应，尤其是使用客户端访问较老的集群时。 如果客户端需要能够处理所有资源类型，或者有可能需要与较老的集群交互， 则需要在其 Accept 头部设定多个内容类型值，以便可以回退到非表格形式的 JSON 表示。

```
Accept: application/json;as=Table;g=meta.k8s.io;v=v1beta1, application/json
```

资源的其他表示形式

默认情况下，Kubernetes 返回 JSON 序列化的的对象并设定内容类型为 application/json。这是 API 的默认序列化格式。 不过，客户端也可出于大规模环境中更佳性能的需求而请求对象的更为高效的 Protobuf 表现形式。 Kubernetes API 实现了标准的 HTTP 内容类型协商：为 GET 调用传递一个 Accept 头部来请求服务器以所指定的内容

类型返回对象，同时在通过 PUT 或 POST 调用 向服务器发送 Protobuf 格式的对象时提供 Content-Type 头部。 服务器会能够支持所请求的格式时返回 Content-Type 头部，并在所提供的内容类型 不合法时返回 406 Not acceptable（无法接受） 错误。

请参阅 API 文档了解每个 API 所支持的内容类型。

例如：

1. 以 Protobuf 格式列举集群上的所有 Pods：

```
GET /api/v1/pods
Accept: application/vnd.kubernetes.protobuf
---
200 OK
Content-Type: application/vnd.kubernetes.protobuf

... binary encoded PodList object
```

1. 通过向服务器发送 Protobuf 编码的数据创建 Pod，但请求以 JSON 形式接收响应：

```
POST /api/v1/namespaces/test/pods
Content-Type: application/vnd.kubernetes.protobuf
Accept: application/json
... binary encoded Pod object
---
200 OK
Content-Type: application/json

{
  "kind": "Pod",
  "apiVersion": "v1",
  ...
}
```

并非所有 API 资源类型都支持 Protobuf，尤其是那些通过定制资源定义（CRD）或通过 API 扩展而加入的资源。如果客户端必须能够处理所有资源类型，则应在其 Accept 头部指定多种内容类型以便可以回退到 JSON 格式：

```
Accept: application/vnd.kubernetes.protobuf, application/json
```

Protobuf encoding

Kubernetes 使用封套形式来对 Protobuf 响应进行编码。 封套外层由 4 个字节的特殊数字开头，便于从磁盘文件或 etcd 中辨识 Protobuf 格式的（而不是 JSON）数据。 接下来存放的是 Protobuf 编码的封套消息，其中描述下层对象的编码和类型，最后 才是对象本身。

封套格式如下：

```
四个字节的特殊数字前缀：
  字节 0-3: "k8s\x00" [0x6b, 0x38, 0x73, 0x00]

使用下面 IDL 来编码的 Protobuf 消息：
  message Unknown {
    // typeMeta 应该包含 "kind" 和 "apiVersion" 的字符串值，就像
    // 对应的 JSON 对象中所设置的那样
    optional TypeMeta typeMeta = 1;

    // raw 中将保存用 protobuf 序列化的完整对象。
    // 参阅客户端库中为指定 kind 所作的 protobuf 定义
    optional bytes raw = 2;

    // contentEncoding 用于 raw 数据的编码格式。未设置此值意味着没有特殊编码。
    optional string contentEncoding = 3;

    // contentType 包含 raw 数据所采用的序列化方法。
    // 未设置此值意味着  application/vnd.kubernetes.protobuf，且通常被忽略
    optional string contentType = 4;
  }

  message TypeMeta {
    // apiVersion 是 type 对应的组名/版本
    optional string apiVersion = 1;
    // kind 是对象模式定义的名称。此对象应该存在一个 protobuf 定义。
    optional string kind = 2;
  }
```

收到 application/vnd.kubernetes.protobuf 格式响应的客户端在响应与预期的前缀 不匹配时应该拒绝响应，因为将来的版本可能需要以某种不兼容的方式更改序列化格式，并且这种更改是通过变更前缀完成的。

资源删除

资源删除要经过两个阶段：1）终止（finalization），和 2）去除。

```
{
  "kind": "ConfigMap",
  "apiVersion": "v1",
  "metadata": {
    "finalizers": {"url.io/neat-finalization", "other-url.io/my-finalizer"},
    "deletionTimestamp": nil,
  }
}
```

当客户端首先删除某资源时，其 .metadata.deletionTimestamp 会被设置为当前时间。一旦 .metadata.deletionTimestamp 被设置，则对终结器（finalizers）执行动作 的外部控制器就可以在任何时候、以任何顺序执行其清理工作。 这里不强调顺序是因为很可能带来 .metadata.finalizers 被锁定的风险。 .metadata.finalizers 是一个共享的字段，任何具有相关权限的主体都可以对其 执行重排序的操作。如果终结器列表要按顺序处理，则很可能导致负责列表中第一个 终结器的组件要等待负责列表中排序靠后的终结器的组件的信号（可能是字段值变更、 外部系统或者其他形式），从而导致死锁行为。 在不对终结器顺序作强制要求的情况下，终结器可以自行排序，且不会因为其在列表 中的顺序而引入任何不稳定因素。

当最后一个终结器也被移除时，资源才真正从 etcd 中移除。

单个资源 API

API 动词 GET、CREATE、UPDATE、PATCH、DELETE 和 PROXY 仅支持单个资源。 这些支持单一资源的动词不支持以有序或无序列表甚或事务的形式同时提交给 多个资源。 包括 kubectl 在内的客户端将解析资源的列表，并执行单一资源的 API 请求。

API 动词 LIST 和 WATCH 支持获取多个资源，而 DELETECOLLECTION 支持删除多个 资源。

试运行

FEATURE STATE: Kubernetes v1.18 [stable]

修改性质的动词（POST、PUT、PATCH 和 DELETE）可以支持 *试运行（dry run）* 模式的请求。试运行模式可帮助通过典型的请求阶段（准入控制链、合法性 检查、合并冲突）来评估请求，只是最终的对象不会写入存储。请求的响应主体与 非试运行模式下的响应尽可能接近。系统会保证试运行模式的请求不会被写入到存储 中，也不会产生其他副作用。

发起试运行请求

通过设置 dryRun 查询参数可以触发试运行模式。此参数是一个字符串，以枚举值 的形式工作且可接受的值只有：

- All：每个阶段被会正常运行，除了最后的存储阶段。准入控制器会被运行来检查请求 是否合法，变更性（Mutating）控制器会变更请求，PATCH 请求也会触发合并操作， 对象字段的默认值也会被设置，且基于模式定义的合法性检查也会被执行。所生成的变更不会被写入到下层的持久性存储中，但本来会写入到数据库中的最终对象 会和正常的状态代码一起被返回给用户。如果请求会触发准入控制器而该准入控制器 带有一定的副作用，则请求会失败而不是冒险产生不希望的副作用。 所有的内置准入控制器插件都支持试运行模式。此外，准入控制 Webhook 也可在其 配置对象 中通过将 sideEffects 字段设置为 "None" 来声明自身不会产生副作用。 如果某 Webhook 确实会产生副作用，那么 sideEffects 字段应该设置为 "NoneOnDryRun"， 并且 Webhook 应该被更改以支持 AdmissionReview 中的 dryRun 字段，从而避免 在试运行时产生副作用。

空字符串（也即默认值）：保留默认的修改行为。

例如：

```
POST /api/v1/namespaces/test/pods?dryRun=All
Content-Type: application/json
Accept: application/json
```

响应会与非试运行模式请求的响应看起来相同，只是某些生成字段的值可能会不同。

试运行的授权

试运行和非试运行请求的鉴权是完全相同的。因此，要发起一个试运行请求，用户必须被授权执行非试运行请求。

例如，要在 Deployment 对象上试运行 PATCH 操作，你必须具有对 Deployment 执行 PATCH 操作的访问权限，如下面的 RBAC 规则所示：

```yaml
rules:
- apiGroups: ["extensions", "apps"]
  resources: ["deployments"]
  verbs: ["patch"]
```

参阅鉴权概述以了解鉴权细节。

生成的值

对象的某些值通常是在对象被写入数据库之前生成的。很重要的一点是不要依赖试运行请求为这些字段所设置的值，因为试运行模式下所得到的这些值与真实请求所获得的 值很可能不同。这类字段有：

- name：如果设置了 generateName 字段，则 name 会获得一个唯一的随机名称
- creationTimestamp/deletionTimestamp：记录对象的创建/删除时间
- UID：唯一性标识对象，取值随机生成（非确定性）
- resourceVersion： 跟踪对象的持久化（存储）版本
- 变更性准入控制器所设置的字段
- 对于 Service 资源：kube-apiserver 为 v1.Service 对象分配的端口和 IP

服务器端应用

FEATURE STATE: Kubernetes v1.16 [beta]

从 Kubernetes v1.18 开始，可以启用服务器端应用功能 特性，启用该特性后，控制面会跟踪所有新创建的对象的托管字段。服务器端应用提供了一种简洁的模式来管理字段冲突，提供服务器端的 Apply 和 Update 操作，并取代了 kubectl apply 的客户端功能。有关该特性的详细描述，请参见服务器端应用章节

资源版本

资源版本采用字符串来表达，用来标示对象的服务器端内部版本。 客户端可以使用资源版本来判定对象是否被更改，或者在读取、列举或监视资源时 用来表达数据一致性需求。 客户端必需将资源版本视为不透明的对象，将其原封不动地传递回服务器端。 例如，客户端一定不能假定资源版本是某种数值标识，也不可以对两个资源版本值 进行比较看其是否相同（也就是不可以比较两个版本值以判断其中一个比另一个 大或小）。

metadata 中的 resourceVersion

客户端可以在资源中看到资源版本信息，这里的资源包括从服务器返回的 Watch 事件 以及 list 操作响应：

v1.meta/ObjectMeta - 资源 的 metadata.resourceVersion 值标明该实例上次被更改时的资源版本。

v1.meta/ListMeta - 资源集合 （即 list 操作的响应）的 metadata.resourceVersion 所标明的是 list 响应被构造 时的资源版本。

resourceVersion 参数

GET、LIST 和 WATCH 操作都支持 resourceVersion 参数。

参数的具体含义取决于所执行的操作和所给的 resourceVersion 值：

对于 GET 和 LIST 而言，资源版本的语义为：

GET：

resourceVersion 未设置	resourceVersion="0"	resourceVersion="<非零值>"
最新版本	任何版本	不老于给定版本

LIST：

v1.19 及以上版本的 API 服务器支持 resourceVersionMatch 参数，用以确定如何对 LIST 调用应用 resourceVersion 值。 强烈建议在为 LIST 调用设置了 resourceVersion 时也设置 resourceVersionMatch。 如果 resourceVersion 未设置，则 resourceVersionMatch 是不允许设置的。 为了向后兼容，客户端必须能够容忍服务器在某些场景下忽略 resourceVersionMatch 的行为：

- 当设置 resourceVersionMatch=NotOlderThan 且指定了 limit 时，客户端必须能够 处理 HTTP 410 "Gone" 响应。例如，客户端可以使用更新一点的 resourceVersion 来重试，或者回退到 resourceVersion="" （即允许返回任何版本）。
- 当设置了 resourceVersionMatch=Exact 且未指定 limit 时，客户端必须验证 响应数据中 ListMeta 的 resourceVersion 与所请求的 resourceVersion 匹配， 并处理二者可能不匹配的情况。例如，客户端可以重试设置了 limit 的请求。

除非你对一致性有着非常强烈的需求，使用 resourceVersionMatch=NotOlderThan 同时为 resourceVersion 设定一个已知值是优选的交互方式，因为与不设置 resourceVersion 和 resourceVersionMatch 相比，这种配置可以取得更好的 集群性能和可扩缩性。后者需要提供带票选能力的读操作。

resourceVersionMatch 参数	**分页参数**	**resourceVersion 未设置**	**resourceV**
resourceVersionMatch 未设置	limit 未设置	最新版本	任意版本
resourceVersionMatch 未设置	limit=<n>, continue 未设置	最新版本	任意版本
resourceVersionMatch 未设置	limit=<n>, continue=<token>	从 token 开始、精确匹配	非法请求，视 开始、精确匹
resourceVersionMatch=Exact [1]	limit 未设置	非法请求	非法请求
resourceVersionMatch=Exact [1]	limit=<n>, continue 未设置	非法请求	非法请求
resourceVersionMatch=NotOlderThan [1]	limit 未设置	非法请求	任意版本
resourceVersionMatch=NotOlderThan [1]	limit=<n>, continue 未设置	非法请求	任意版本

脚注：

[1] 如果服务器无法正确处理 resourceVersionMatch 参数，其行为与未设置该参数相同。

GET 和 LIST 操作的语义含义如下：

- **最新版本：** 返回资源版本为最新的数据。所返回的数据必须一致（通过票选读操作从 etcd 中取出）。
- **任意版本：** 返回任意资源版本的数据。优选最新可用的资源版本，不过不能保证 强一致性；返回的数据可能是任何资源版本的。请求返回的数据有可能是客户端以前看到过的很老的资源版本。尤其在某些高可用配置环境中，网络分区或者高速缓存未被更新等状态都可能导致这种状况。不能容忍这种不一致性的客户端不应采用此语义。

- **不老于指定版本：** 返回至少比所提供的 resourceVersion 还要新的数据。 优选最新的可用数据，不过最终提供的可能是不老于所给 resourceVersion 的任何版本。对于发给能够正确处理 resourceVersionMatch 参数的服务器的 LIST 请求，此语义 保证 ListMeta 中的 resourceVersion 不老于请求的 resourceVersion，不过 不对列表条目之 ObjectMeta 的 resourceVersion 提供任何保证。 这是因为 ObjectMeta.resourceVersion 所跟踪的是列表条目对象上次更新的时间， 而不是对象被返回时是否是最新。

- **确定版本：** 返回精确匹配所给资源版本的数据。如果所指定的 resourceVersion 的数据不可用，服务器会响应 HTTP 410 "Gone"。 对于发送给能够正确处理 resourceVersionMatch 参数的服务器的 LIST 请求而言， 此语义会保证 ListMeta 中的 resourceVersion 与所请求的 resourceVersion 匹配， 不过不对列表条目之 ObjectMeta 的 resourceVersion 提供任何保证。 这是因为 ObjectMeta.resourc

eVersion 所跟踪的是列表条目对象上次更新的时间， 而不是对象被返回时是否是最新。

- **Continue 令牌、精确匹配：** 返回原先带分页参数的 LIST 调用中指定的资源版本的数据。 在最初的带分页参数的 LIST 调用之后，所有分页式的 LIST 调用都使用所返回的 Continue 令牌来跟踪最初提供的资源版本，

对于 WATCH 操作而言，资源版本的语义如下：

WATCH：

resourceVersion 未设置	resourceVersion="0"	resourceVersion="<非零值>"
读取状态并从最新版本开始	读取状态并从任意版本开始	从指定版本开始

WATCH 操作语义的含义如下：

- **读取状态并从最新版本开始：** 从最新的资源版本开始 WATCH 操作。这里的 最新版本必须是一致的（即通过票选读操作从 etcd 中取出）。为了建立初始状态，WATCH 首先会处理一组合成的 "Added" 事件，这些事件涵盖在初始资源版本中存在 的所有资源实例。 所有后续的 WATCH 事件都是关于 WATCH 开始时所处资源版本之后发生的变更。

- **读取状态并从任意版本开始：** 警告：通过这种方式初始化的 WATCH 操作可能会返回任何状态的停滞数据。请在使用此语义之前执行复核，并在可能的情况下采用其他 语义。此语义会从任意资源版本开始执行 WATCH 操作，优选最新的可用的资源版本， 不过不是必须的；采用任何资源版本作为起始版本都是被允许的。WATCH 操作有可能起始于客户端已经观测到的很老的版本。在高可用配置环境中，因为 网络分裂或者高速缓存未及时更新的原因都会造成此现象。 如果客户端不能容忍这种不一致性，就不要使用此语义来启动 WATCH 操作。 为了建立初始状态，WATCH 首先会处理一组合成的 "Added" 事件，这些事件涵盖在 初始资源版本中存在的所有资源实例。 所有后续的 WATCH 事件都是关于 WATCH 开始时所处资源版本之后发生的变更。

- **从指定版本开始：** 从某确切资源版本开始执行 WATCH 操作。WATCH 事件都是关于 WATCH 开始时所处资源版本之后发生的变更。与前面两种语义不同，WATCH 操作 开始的时候不会生成或处理为所提供资源版本合成的 "Added" 事件。 我们假定客户端既然能够提供确切资源版本，就应该已经拥有了起始资源版本对应的初始状态。

"410 Gone" 响应

服务器不需要提供所有老的资源版本，在客户端请求的是早于服务器端所保留版本的 resourceVersion 时，可以返回 HTTP 410 (Gone) 状态码。 客户端必须能够容忍 410 (Gone) 响应。 参阅高效检测变更以了解如何在监测资源时 处理 410 (Gone) 响应。

如果所请求的 resourceVersion 超出了可应用的 limit，那么取决于请求是否 是通过高速缓存来满足的，API 服务器可能会返回一个 410 Gone HTTP 响应。

不可用的资源版本

服务器不必未无法识别的资源版本提供服务。针对无法识别的资源版本的 LIST 和 GET 请求 可能会短暂等待，以期资源版本可用。如果所给的资源版本在一定的时间段内仍未变得 可用，服务器应该超时并返回 504 (Gateway Timeout)，且可在响应中添加 Retry-After 响应头部字段，标明客户端在再次尝试之前应该等待多少秒钟。 目前，kube-apiserver 也能使用 Too large resource version（资源版本过高） 消息来标识这类响应。针对某无法识别的资源版本的 WATCH 操作可能会无限期 （直到请求超时）地等待下去，直到资源版本可用。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 13, 2021 at 9:01 AM PST: fix typo (33ecc0c09)

服务器端应用（Server-Side Apply）

FEATURE STATE: Kubernetes v1.16 [beta]

简介

服务器端应用协助用户、控制器通过声明式配置的方式管理他们的资源。 它发送完整描述的目标（A fully specified intent）， 声明式地创建和/或修改 对象。

一个完整描述的目标并不是一个完整的对象，仅包括能体现用户意图的字段和值。 该目标（intent）可以用来创建一个新对象， 也可以通过服务器来实现与现有对象的合并。

系统支持多个应用者（appliers）在同一个对象上开展协作。

"字段管理（field management）"机制追踪对象字段的变化。 当一个字段值改变时，其所有权从当前管理器（manager）转移到施加变更的管理器。 当尝试将新配置应用到一个对象时，如果字段有不同的值，且由其他管理器管理， 将会引发冲突。 冲突引发警告信号：此操作可能抹掉其他协作者的修改。 冲突可以被刻意忽略，这种情况下，值将会被改写，所有权也会发生转移。

当你从配置文件中删除一个字段，然后应用这个配置文件， 这将触发服务端应用检查此字段是否还被其他字段管理器拥有。 如果没有，那就从活动对象中删除该字段；如果有，那就重置为默认值。 该规则同样适用于 list 或 map 项目。

服务器端应用既是原有 kubectl apply 的替代品， 也是控制器发布自身变化的一个简化机制。

如果你启用了服务器端应用，控制平面就会跟踪被所有新创建对象管理的字段。

字段管理

相对于通过 kubectl 管理的注解 last-applied， 服务器端应用使用了一种更具声明式特点的方法： 它持续的跟踪用户的字段管理，而不仅仅是最后一次的执行状态。 这就意味着，作为服务器端应用的一个副作用， 关于用哪一个字段管理器负责管理对象中的哪个字段的这类信息，都要对外界开放了。

用户管理字段这件事，在服务器端应用的场景中，意味着用户依赖并期望字段的值不要改变。 最后一次对字段值做出断言的用户将被记录到当前字段管理器。 这可以通过发送 POST、 PUT、 或非应用（non-apply）方式的 PATCH 等命令来修改字段值的方式实现， 或通过把字段放在配置文件中，然后发送到服务器端应用的服务端点的方式实现。 当使用服务器端应用，尝试着去改变一个被其他人管理的字段， 会导致请求被拒绝（在没有设置强制执行时，参见冲突）。

如果两个或以上的应用者均把同一个字段设置为相同值，他们将共享此字段的所有权。 后续任何改变共享字段值的尝试，不管由那个应用者发起，都会导致冲突。 共享字段的所有者可以放弃字段的所有权，这只需从配置文件中删除该字段即可。

字段管理的信息存储在 managedFields 字段中，该字段是对象的 metadata中的一部分。

服务器端应用创建对象的简单示例如下：

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-cm
  namespace: default
  labels:
    test-label: test
  managedFields:
  - manager: kubectl
    operation: Apply
    apiVersion: v1
    time: "2010-10-10T0:00:00Z"
    fieldsType: FieldsV1
    fieldsV1:
      f:metadata:
        f:labels:
          f:test-label: {}
      f:data:
        f:key: {}
data:
  key: some value
```

上述对象在 metadata.managedFields 中包含了唯一的管理器。 管理器由管理实体自身的基本信息组成，比如操作类型、API 版本、以及它管理的字段。

> **说明：** 该字段由 API 服务器管理，用户不应该改动它。

不过，执行 Update 操作修改 metadata.managedFields 也是可实现的。 强烈不鼓励这么做，但当发生如下情况时， 比如 managedFields 进入不一致的状态（显然不应该发生这种情况）， 这么做也是一个合理的尝试。

managedFields 的格式在 API 文档中描述。

冲突

冲突是一种特定的错误状态， 发生在执行 Apply 改变一个字段，而恰巧该字段被其他用户声明过主权时。 这可以防止一个应用者不小心覆盖掉其他用户设置的值。 冲突发生时，应用者有三种办法来解决它：

- **覆盖前值，成为唯一的管理器：** 如果打算覆盖该值（或应用者是一个自动化部件，比如控制器）， 应用者应该设置查询参数 force 为 true，然后再发送一次请求。 这将强制操作成功，改变字段的值，从所有其他管理器的 managedFields 条目中删除指定字段。
- **不覆盖前值，放弃管理权：** 如果应用者不再关注该字段的值， 可以从配置文件中删掉它，再重新发送请求。 这就保持了原值不变，并从 managedFields 的应用者条目中删除该字段。
- **不覆盖前值，成为共享的管理器：** 如果应用者仍然关注字段值，并不想覆盖它， 他们可以在配置文件中把字段的值改为和服务器对象一样，再重新发送请求。 这样在不改变字段值的前提下， 就实现了字段管理被应用者和所有声明了管理权的其他的字段管理器共享。

管理器

管理器识别出正在修改对象的工作流程（在冲突时尤其有用），管理器可以通过修改请求的参数 fieldManager 指定。 虽然 kubectl 默认发往 kubectl 服务端点，但它则请求到应用的服务端点（apply endpoint）。 对于其他的更新，它默认的是从用户代理计算得来。

应用和更新

此特性涉及两类操作，分别是 Apply （内容类型为 application/apply-patch+yaml 的 PATCH 请求）和 Update （所有修改对象的其他操作）。 这两类操作都会更新字段 managedFields，但行为表现有一点不同。

> **说明：**
>
> 不管你提交的是 JSON 数据还是 YAML 数据， 都要使用 application/apply-patch+yaml 作为 Content-Type 的值。

所有的 JSON 文档 都是合法的 YAML。

例如，在冲突发生的时候，只有 apply 操作失败，而 update 则不会。 此外，apply 操作必须通过提供一个 fieldManager 查询参数来标识自身， 而此查询参数对于 update 操作则是可选的。 最后，当使用 apply 命令时，你不能在应用中的对象中持有 managedFields。

一个包含多个管理器的对象，示例如下：

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-cm
  namespace: default
  labels:
    test-label: test
  managedFields:
  - manager: kubectl
    operation: Apply
    apiVersion: v1
    fields:
      f:metadata:
        f:labels:
          f:test-label: {}
  - manager: kube-controller-manager
    operation: Update
    apiVersion: v1
    time: '2019-03-30T16:00:00.000Z'
    fields:
      f:data:
        f:key: {}
data:
  key: new value
```

在这个例子中， 第二个操作被管理器 kube-controller-manager 以 Update 的方式运行。 此 update 更改 data 字段的值， 并使得字段管理器被改为 kube-controller-manager。

如果把 update 操作改为 Apply，那就会因为所有权冲突的原因，导致操作失败。

合并策略

由服务器端应用实现的合并策略，提供了一个总体更稳定的对象生命周期。 服务器端应用试图依据谁管理它们来合并字段，而不只是根据值来否决。 这么做是为了多个参与者可以更简单、更稳定的更新同一个对象，且避免引起意外干扰。

当用户发送一个"完整描述的目标"对象到服务器端应用的服务端点， 服务器会将它和活动对象做一次合并，如果两者中有重复定义的值，那就以配置文件的为准。 如果配置文件中的项目集合不是此用户上一次操作项目的超集， 所有缺少的、没有其他应用者管理的项目会被删除。 关于合并时用来做决策的对象规格的更多信息，参见 sigs.k8s.io/structured-merge-diff.

Kubernetes 1.16 和 1.17 中添加了一些标记， 允许 API 开发人员描述由 list、map、和 structs 支持的合并策略。 这些标记可应用到相应类型的对象，在 Go 文件或在 CRD 的 OpenAPI 的模式中定义：

Golang 标记	OpenAPI extension	可接受的值	描述	引入版本
//+listType	x-kubernetes-list-type	atomic/set/map	适用于 list。 atomic 和 set 适用于只包含标量元素的 list。 map 适用于只包含嵌套类型的 list。 如果配置为 atomic, 合并时整个列表会被替换掉; 任何时候，唯一的管理器都把列表作为一个整体来管理。如果是 set 或 map ，不同的管理器也可以分开管理条目。	1.16
//+listMapKey	x-kubernetes-list-map-keys	用来唯一标识条目的 map keys 切片，例如 ["port", "protocol"]	仅当 +listType=map 时适用。组合值的字符串切片必须唯一标识列表中的条目。尽管有多个 key，listMapKey 是单数的，这是因为 key 需要在 Go 类型中单独的指定。	1.16
//+mapType	x-kubernetes-map-type	atomic/granular	适用于 map。 atomic 指 map 只能被单个的管理器整个的替换。 granular 指 map 支持多个管理器各自更新自己的字段。	1.17
//+structType	x-kubernetes-map-type	atomic/granular	适用于 structs；否则就像 //+mapType 有相同的用法和 openapi 注释.	1.17

自定义资源

默认情况下，服务器端应用把自定义资源看做非结构化数据。 所有的键值（keys）就像 struct 的字段一样被处理， 所有的 list 被认为是原子性的。

如果自定义资源定义（Custom Resource Definition，CRD）定义了一个 模式， 它包含类似以前"合并策略"章节中定义过的注解， 这些注解将在合并此类型的对象时使用。

在控制器中使用服务器端应用

控制器的开发人员可以把服务器端应用作为简化控制器的更新逻辑的方式。 读-改-写 和/或 patch 的主要区别如下所示：

- 应用的对象必须包含控制器关注的所有字段。
- 对于在控制器没有执行过应用操作之前就已经存在的字段，不能删除。 （控制器在这种用例环境下，依然可以发送一个 PATCH/UPDATE）
- 对象不必事先读取，resourceVersion 不必指定。

强烈推荐：设置控制器在冲突时强制执行，这是因为冲突发生时，它们没有其他解决方案或措施。

转移所有权

除了通过冲突解决方案提供的并发控制， 服务器端应用提供了一些协作方式来将字段所有权从用户转移到控制器。

最好通过例子来说明这一点。 让我们来看看，在使用 HorizontalPodAutoscaler 资源和与之配套的控制器， 且开启了 Deployment 的自动水平扩展功能之后， 怎么安全的将 replicas 字段的所有权从用户转移到控制器。

假设用户定义了 Deployment，且 replicas 字段已经设置为期望的值：

application/ssa/nginx-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

并且，用户使用服务器端应用，像这样创建 Deployment：

```
kubectl apply -f https://k8s.io/examples/application/ssa/nginx-deployment.yaml --server-side
```

然后，为 Deployment 启用 HPA，例如：

```
kubectl autoscale deployment nginx-deployment --cpu-percent=50 --min=1 --max=10
```

现在，用户希望从他们的配置中删除 replicas，所以他们总是和 HPA 控制器冲突。 然而，这里存在一个竟态： 在 HPA 需要调整 replicas 之前会有一个时间窗口， 如果在 HPA 写入字段成为所有者之前，用户删除了replicas， 那 API 服务器就会把 replicas 的值设为1， 也就是默认值。 这不是用户希望发生的事情，即使是暂时的。

这里有两个解决方案：

- （容易） 把 replicas 留在配置文件中；当 HPA 最终写入那个字段， 系统基于此事件告诉用户：冲突发生了。在这个时间点，可以安全的删除配置文件。
- （高级）然而，如果用户不想等待，比如他们想为合作伙伴保持集群清晰， 那他们就可以执行以下步骤，安全的从配置文件中删除 replicas。

首先，用户新定义一个只包含 replicas 字段的配置文件：

application/ssa/nginx-deployment-replicas-only.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
```

用户使用名为 handover-to-hpa 的字段管理器，应用此配置文件。

```
kubectl apply -f https://k8s.io/examples/application/ssa/nginx-deployment-replicas-only.yaml \
  --server-side --field-manager=handover-to-hpa \
  --validate=false
```

如果应用操作和 HPA 控制器产生冲突，那什么都不做。 冲突只是表明控制器在更早的流程中已经对字段声明过所有权。

在此时间点，用户可以从配置文件中删除 replicas 。

application/ssa/nginx-deployment-no-replicas.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
```

```yaml
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

注意，只要 HPA 控制器为 replicas 设置了一个新值， 该临时字段管理器将不再拥有任何字段，会被自动删除。 这里不需要执行清理工作。

在用户之间转移所有权

通过在配置文件中把一个字段设置为相同的值，用户可以在他们之间转移字段的所有权，从而共享了字段的所有权。 当用户共享了字段的所有权，任何一个用户可以从他的配置文件中删除该字段， 并应用该变更，从而放弃所有权，并实现了所有权向其他用户的转移。

与客户端应用的对比

由服务器端应用实现的冲突检测和解决方案的一个结果就是， 应用者总是可以在本地状态中得到最新的字段值。 如果得不到最新值，下次执行应用操作时就会发生冲突。 解决冲突三个选项的任意一个都会保证：此应用过的配置文件是服务器上对象字段的最新子集。

这和客户端应用（Client Side Apply） 不同，如果有其他用户覆盖了此值， 过期的值被留在了应用者本地的配置文件中。 除非用户更新了特定字段，此字段才会准确， 应用者没有途径去了解下一次应用操作是否会覆盖其他用户的修改。

另一个区别是使用客户端应用的应用者不能改变他们正在使用的 API 版本，但服务器端应用支持这个场景。

从客户端应用升级到服务器端应用

客户端应用方式时，用户使用 kubectl apply 管理资源， 可以通过使用下面标记切换为使用服务器端应用。

```
kubectl apply --server-side [--dry-run=server]
```

默认情况下，对象的字段管理从客户端应用方式迁移到 kubectl 触发的服务器端应用时，不会发生冲突。

> **注意：**
>
> 保持注解 last-applied-configuration 是最新的。 从注解能推断出字段是由客户端应用管理的。 任何没有被客户端应用管理的字段将引发冲突。
>
> 举例说明，比如你在客户端应用之后， 使用 kubectl scale 去更新 replicas 字段， 可是该字段并没有被客户端应用所拥有， 在执行 kubectl apply --server-side 时就会产生冲突。

此操作以 kubectl 作为字段管理器来应用到服务器端应用。 作为例外，可以指定一个不同的、非默认字段管理器停止的这种行为，如下面的例子所示。 对于 kubectl 触发的服务器端应用，默认的字段管理器是 kubectl。

```
kubectl apply --server-side --field-manager=my-manager [--dry-run=server]
```

从服务器端应用降级到客户端应用

如果你用 kubectl apply --server-side 管理一个资源， 可以直接用 kubectl apply 命令将其降级为客户端应用。

降级之所以可行，这是因为 kubectl server-side apply 会保存最新的 last-applied-configuration 注解。

此操作以 kubectl 作为字段管理器应用到服务器端应用。 作为例外，可以指定一个不同的、非默认字段管理器停止这种行为，如下面的例子所示。 对于 kubectl 触发的服务器端应用，默认的字段管理器是 kubectl。

```
kubectl apply --server-side --field-manager=my-manager [--dry-run=server]
```

API 端点

启用了服务器端应用特性之后， PATCH 服务端点接受额外的内容类型 application/apply-patch+yaml。 服务器端应用的用户就可以把 YAMl 格式的 部分定义对象（partially specified objects）发送到此端点。 当一个配置文件被应用时，它应该包含所有体现你意图的字段。

清除 ManagedFields

可以从对象中剥离所有 managedField， 实现方法是通过使用 MergePatch、 StrategicMergePatch、 JSONPatch、 Update、以及所有的非应用方式的操作来覆盖它。 这可以通过用空条目覆盖 managedFields 字段的方式实现。

```
PATCH /api/v1/namespaces/default/configmaps/example-cm
Content-Type: application/merge-patch+json
```

```
Accept: application/json
Data: {"metadata":{"managedFields": [{}]}}
```

```
PATCH /api/v1/namespaces/default/configmaps/example-cm
Content-Type: application/json-patch+json
Accept: application/json
Data: [{"op": "replace", "path": "/metadata/managedFields", "value": [{}]}]
```

这一操作将用只包含一个空条目的 list 覆写 managedFields， 来实现从对象中整个的去除 managedFields。 注意，只把 managedFields 设置为空 list 并不会重置字段。 这么做是有目的的，所以 managedFields 将永远不会被与该字段无关的客户删除。

在重置操作结合 managedFields 以外其他字段更改的场景中， 将导致 managedFields 首先被重置，其他改变被押后处理。 其结果是，应用者取得了同一个请求中所有字段的所有权。

> **注意：** 对于不接受资源对象类型的子资源（sub-resources）， 服务器端应用不能正确地跟踪其所有权。 如果你对这样的子资源使用服务器端应用，变更的字段将不会被跟踪。

禁用此功能

服务器端应用是一个 beta 版特性，默认启用。 要关闭此特性门控， 你需要在启动 kube-apiserver 时包含参数 `--feature-gates ServerSideApply=false`。 如果你有多个 kube-apiserver 副本，他们都应该有相同的标记设置。

反馈

最后修改 November 30, 2020 at 9:43 PM PST: [zh] Sync using-api (9af43634c)

客户端库

本页面包含基于各种编程语言使用 Kubernetes API 的客户端库概述。

在使用 Kubernetes REST API 编写应用程序时， 您并不需要自己实现 API 调用和 "请求/响应" 类型。 您可以根据自己的编程语言需要选择使用合适的客户端库。

客户端库通常为您处理诸如身份验证之类的常见任务。 如果 API 客户端在 Kubernetes 集群中运行，大多数客户端库可以发现并使用 Kubernetes 服务帐户进行身份验证， 或者能够理解 kubeconfig 文件 格式来读取凭据和 API 服务器地址。

官方支持的 Kubernetes 客户端库

以下客户端库由 Kubernetes SIG API Machinery 正式维护。

语言	客户端库	样例程序
Go	github.com/kubernetes/client-go/	浏览
Python	github.com/kubernetes-client/python/	浏览
Java	github.com/kubernetes-client/java	浏览
dotnet	github.com/kubernetes-client/csharp	浏览
JavaScript	github.com/kubernetes-client/javascript	浏览
Haskell	github.com/kubernetes-client/haskell	浏览

社区维护的客户端库

> **注意：** 本部分链接到提供 Kubernetes 所需功能的第三方项目。Kubernetes 项目作者不负责这些项目。此页面遵循CNCF 网站指南，按字母顺序列出项目。要将项目添加到此列表中，请在提交更改之前阅读内容指南。

以下 Kubernetes API 客户端库是由社区，而非 Kubernetes 团队支持、维护的。

语言	客户端库
Clojure	github.com/yanatan16/clj-kubernetes-api
Go	github.com/ericchiang/k8s
Java (OSGi)	bitbucket.org/amdatulabs/amdatu-kubernetes
Java (Fabric8, OSGi)	github.com/fabric8io/kubernetes-client
Java	github.com/manusa/yakc
Lisp	github.com/brendandburns/cl-k8s
Lisp	github.com/xh4/cube
Node.js (TypeScript)	github.com/Goyoo/node-k8s-client
Node.js	github.com/ajpauwels/easy-k8s
Node.js	github.com/godaddy/kubernetes-client
Node.js	github.com/tenxcloud/node-kubernetes-client
Perl	metacpan.org/pod/Net::Kubernetes
PHP	github.com/allansun/kubernetes-php-client
PHP	github.com/maclof/kubernetes-client
PHP	github.com/travisghansen/kubernetes-client-php
PHP	github.com/renoki-co/php-k8s
Python	github.com/eldarion-gondor/pykube
Python	github.com/fiaas/k8s
Python	github.com/mnubo/kubernetes-py
Python	github.com/tomplus/kubernetes_asyncio
Ruby	github.com/abonas/kubeclient
Ruby	github.com/Ch00k/kuber
Ruby	github.com/kontena/k8s-client

语言	客户端库
Rust	github.com/clux/kube-rs
Rust	github.com/ynqa/kubernetes-rust
Scala	github.com/doriordan/skuber
Scala	github.com/joan38/kubernetes-client
Swift	github.com/swiftkube/client
DotNet	github.com/tonnyeremin/kubernetes_gen
DotNet (RestSharp)	github.com/masroorhasan/Kubernetes.DotNet
Elixir	github.com/obmarg/kazan
Elixir	github.com/coryodaniel/k8s

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 24, 2020 at 9:58 AM PST: add "Swift" entry. (9e149cb47)

Kubernetes 弃用策略

本文档详细解释系统中各个层面的弃用策略（Deprecation Policy）。

Kubernetes 是一个组件众多、贡献者人数众多的大系统。 就像很多类似的软件，所提供的功能特性集合会随着时间推移而自然发生变化， 而且有时候某个功能特性可能需要被去除。被去除的可能是一个 API、一个参数标志或者 甚至某整个功能特性。为了避免影响到现有用户，Kubernetes 对于其中渐次移除 的各个方面规定了一种弃用策略并遵从此策略。

弃用 API 的一部分

由于 Kubernetes 是一个 API 驱动的系统，API 会随着时间推移而演化，以反映 人们对问题共建的认识的变化。Kubernetes API 实际上是一个 API 集合，其中每个 成员称作"API 组（API Group）"，并且每个 API 组都是独立管理版本的。 API 版本会有 三类，每类有不同的废弃策略：

示例	分类
v1	正式发布（Generally available，GA，稳定版本）
v1beta1	Beta （预发布）
v1alpha1	Alpha （试验性）

给定的 Kubernetes 发布版本中可以支持任意数量的 API 组，且每组可以包含 任意个数的版本。

下面的规则负责指导 API 元素的弃用，具体元素包括：

- REST 资源（也即 API 对象）
- REST 资源的字段
- REST 资源的注解，包含"beta"类注解但不包含"alpha"类注解
- 枚举值或者常数值
- 组件配置结构

以下是跨正式发布版本时要实施的规则，不适用于对 master 或发布分支上 不同提交之间的变化。

规则 #1：只能在增加 API 组版本号时删除 API 元素。

一旦在某个特定 API 组版本中添加了 API 元素，则该元素不可从该版本中删除， 且其行为也不能大幅度地变化，无论属于哪一类（GA、Alpha 或 Beta）。

> **说明：** 由于历史原因，Kubernetes 中存在两个"单体式（Monolithic）"API 组 - "core"（无组名）和"extensions"。这两个遗留 API 组中的资源会被逐渐迁移到 更为特定领域的 API 组中。

规则 #2：在给定的发布版本中，API 对象必须能够在不同的 API 版本之间来回 转换且不造成信息丢失，除非整个 REST 资源在某些版本中完全不存在。

例如，一个对象可被用 v1 来写入之后用 v2 来读出并转换为 v1，所得到的 v1 必须 与原来的 v1 对象完全相同。v2 中的表现形式可能与 v1 不同，但系统知道如何 在两个版本之间执行双向转换。 此外，v2 中添加的所有新字段都必须能够转换为 v1 再转换回来。这意味着 v1 必须 添加一个新的等效字段或者将其表现为一个注解。

规则 #3：给定类别的 API 版本在新的、稳定性未降低的 API 版本发布之前不可被废弃。

一个正式发布的（GA）API 版本可替换现有的正式 API 版本或 alpha、beta API 版本。 Beta API 版本 *不可以* 替代正式的 API 版本。

规则 #4a：除了每类 API 版本中的最新版本，旧的 API 版本在其被宣布被废弃之后至少以下时长内仍需被支持：

- **GA：12 个月或者 3 个发布版本（取其较长者）**
- **Beta: 9 个月或者 3 个发布版本（取其较长者）**
- **Alpha: 0 个发布版本**

这里也包含了关于最大支持 2 个发布版本的版本偏差的约定。

> **说明：** 在#52185被解决之前， 已经被保存到持久性存储中的 API 版本都不可以被去除。 你可以禁止这些版本所对应的 REST 末端（在符合本文中弃用时间线的前提下）， 但是 API 服务器必须仍能解析和转换存储中以前写入的数据。

规则 #4b：标记为"preferred（优选的）" API 版本和给定 API 组的 "storage version（存储版本）"在既支持老版本也支持新版本的 Kubernetes 发布 版本出来以前不可以提升其版本号。

用户必须能够升级到 Kubernetes 新的发行版本，之后再回滚到前一个发行版本，且 整个过程中无需针对新的 API 版本做任何转换，也不允许出现功能损坏的情况， 除非用户显式地使用了仅在较新版本中才存在的功能特性。 就对象的存储表示而言，这一点尤其是不言自明的。

以上所有规则最好通过例子来说明。假定现有 Kubernetes 发行版本为 X，其中引入了新的 API 组。大约每隔 3 个月会有一个新的 Kubernetes 版本被发布（每年 4 个版本）。 下面的表格描述了在一系列后续的发布版本中哪些 API 版本是受支持的。

发布版本	**API 版本**	**优选/存储版本**	**注释**
X	v1alpha1	v1alpha1	
X+1	v1alpha2	v1alpha2	• v1alpha1 被去除，发布说明中会包含 "action required（采取行动）" 说明
X+2	v1beta1	v1beta1	• v1alpha2 被去除，发布说明中包含对应的 "action required（采取行动）" 说明
X+3	v1beta2、v1beta1（已弃用）	v1beta1	• v1beta1 被弃用，发布说明中包含对应的 "action required（采取行动）" 说明
X+4	v1beta2、v1beta1（已弃用）	v1beta2	
X+5	v1、v1beta1（已弃用）、v1beta2（已弃用）	v1beta2	• v1beta2 被弃用，发布说明中包含对应的 "action required（采取行动）" 说明
X+6	v1、v1beta2（已弃用）	v1	• v1beta1 被去除，发布说明中包含对应的 "action required（采取行动）" 说明
X+7	v1、v1beta2（已弃用）	v1	
X+8	v2alpha1、v1	v1	• v1beta2 被去除，发布说明中包含对应的 "action required（采取行动）" 说明
X+9	v2alpha2、v1	v1	• v2alpha1 被删除，发布说明中包含对应的 "action required（采取行动）" 说明
X+10	v2beta1、v1	v1	• v2alpha2 被删除，发布说明中包含对应的 "action required（采取行动）" 说明

发布版本	API 版本	优选/存储版本	注释
X+11	v2beta2、v2beta1（已弃用）、v1	v1	• v2beta1 被弃用，发布说明中包含对应的 "action required（采取行动）" 说明
X+12	v2、v2beta2（已弃用）、v2beta1（已弃用）、v1（已弃用）	v1	• v2beta2 已被弃用，发布说明中包含对应的 "action required（采取行动）" 说明 • v1 已被弃用，发布说明中包含对应的 "action required（采取行动）" 说明
X+13	v2、v2beta1（已弃用）、v2beta2（已弃用）、v1（已弃用）	v2	
X+14	v2、v2beta2（已弃用）、v1（已弃用）	v2	• v2beta1 被删除，发布说明中包含对应的 "action required（采取行动）" 说明
X+15	v2、v1（已弃用）	v2	• v2beta2 被删除，发布说明中包含对应的 "action required（采取行动）" 说明
X+16	v2、v1（已弃用）	v2	
X+17	v2	v2	• v1 被删除，发布说明中包含对应的 "action required（采取行动）" 说明

REST 资源（也即 API 对象）

考虑一个假想的名为 Widget 的 REST 资源，在上述时间线中位于 API v1， 而现在打算将其弃用。 我们会在文档和 声明 中与 X+1 版本的发布同步记述此弃用决定。 Wdiget 资源仍会在 API 版本 v1（已弃用）中存在，但不会出现在 v2alpha1 中。 Widget 资源会 X+8 发布版本之前（含 X+8）一直存在并可用。 只有在发布版本 X+9 中，API v1 寿终正寝时，Widget 才彻底消失，相应的资源行为也被移除。

从 Kubernetes v1.19 开始，当 API 请求被发送到一个已弃用的 REST API 末端时：

1. API 响应中会包含一个 Warning 头部字段（如 RFC7234 5.5 节所定义）；
2. 该请求会导致对应的 审计事件 中会增加一个注解 "k8s.io/deprecated":"true"。
3. kube-apiserver 进程的 apiserver_requested_deprecated_apis 度量值会被 设置为 1。 该度量值还附带 group、version、resource 和 subresource 标签（可供添加到度量值 apiserver_request_total 上）， 和一个 removed_release 标签，标明该 API 将消失的 Kubernetes 发布版本。 下面的 Prometheus 查询会返回对 v1.22 中将移除的、已弃用的 API 的请求的信息：

```
apiserver_requested_deprecated_apis{removed_release="1.22"} * on(group,version,resource,subresource) group_right() apiserver_request_total
```

REST 资源的字段

就像整个 REST 资源一样，在 API v1 中曾经存在的各个字段在 API v1 被移除 之前必须一直存在且起作用。 与整个资源上的规定不同，v2 API 可以选择为字段提供不同的表示方式， 只要对应的资源对象可在不同版本之间来回转换即可。 例如，v1 版本中一个名为 "magnitude" 的已弃用字段可能在 API v2 中被命名 为 "deprecatedMagnitude"。当 v1 最终被移除时，废弃的字段也可以从 v2 中移除。

枚举值或常数值

就像前文讲述的 REST 资源及其中的单个字段一样，API v1 中所支持的常数值 必须在 API v1 被移除之前一直存在且起作用。

组件配置结构

组件的配置也是有版本的，并且按 REST 资源的方式来管理。

将来的工作

随着时间推移，Kubernetes 会引入粒度更细的 API 版本。 到那时，这里的规则会根据需要进行调整。

弃用一个标志或 CLI 命令

Kubernetes 系统中包含若干不同的、相互协作的程序。 有时，Kubernetes 可能会删除这些程序的某些标志或 CLI 命令（统称"命令行元素"）。 这些程序可以天然地划分到两个大组中：面向用户的和面向管理员的程序。 二者之间的弃用策略略有不同。 除非某个标志显示地通过前缀或文档来标明其为"alpha"或"beta"， 该标志要被视作正式发布的（GA）。

命令行元素相当于系统的 API 的一部分，不过因为它们并没有采用 REST API 一样的方式来管理版本，其弃用规则规定如下：

规则 #5a：面向用户的命令行元素（例如，kubectl）必须在其宣布被弃用其 在以下时长内仍能使用：

- **GA：12 个月或者 2 个发布版本（取其较长者）**
- **Beta：3 个月或者 1 个发布版本（取其较长者）**
- **Alpha：0 发布版本**

规则 #5b：面向管理员的命令行元素（例如，kubelet）必须在其被宣布弃用 之后以下时长内保持可用：

- **GA：6 个月或 1 个发行版本（取其较长者）**
- **Beta: 3 个月或 1 个发行版本（取其较长者）**
- **Alpha: 0 个发布版本**

规则 #6：被弃用的 CLI 元素在被用到时必须能够产生警告，而警告的 产生是可以被禁止的。

弃用某功能特性或行为

在一些较偶然的情形下，某 Kubernetes 发行版本需要弃用系统的某项功能 特性或者行为，而对应的功能特性或行为并不受 API 或 CLI 控制。在这种情况下， 其弃用规则如下：

规则 #7：被弃用的行为必须在被宣布弃用之后至少 1 年时间内必须保持能用。

这并不意味着对系统的所有更改都受此策略约束。 此规则仅适用于重大的、用户可见的行为；这些行为会影响到在 Kubernetes 中运行的应用的正确性，或者影响到 Kubernetes 集群的管理。 此规则也适用于那些被整个移除的功能特性或行为。

上述规则的一个例外是 *特性门控（Feature Gates）*。特性门控是一些键值偶对， 允许用户启用或禁用一些试验性的功能特性。

特性门控意在覆盖功能特性的整个开发周期，它们无意成为长期的 API。 因此，它们会在某功能特性正式发布或被抛弃之后被弃用和删除。

随着一个功能特性经过不同的成熟阶段，相关的特性门控也会演化。 与功能特性生命周期对应的特性门控状态为：

- Alpha：特性门控默认被禁用，只能由用户显式启用。
- Beta：特性门控默认被弃用，可被用户显式禁用。
- GA: 特性门控被弃用（参见弃用），并且不再起作用。
- GA，弃用窗口期结束：特性门控被删除，不再接受调用。

弃用

功能特性在正式发布之前的生命周期内任何时间点都可被移除。 当未正式发布的功能特性被移除时，它们对应的特性门控也被弃用。

当尝试禁用一个不再起作用的特性门控时，该调用会失败，这样可以避免 毫无迹象地执行一些不受支持的场景。

在某些场合，移除一个即将正式发布的功能特性需要很长时间。特性门控 可以保持其功能，直到对应的功能特性被彻底去除，直到那时特性门控 自身才可被弃用。

由于移除一个已经正式发布的功能特性对应的特性门控也需要一定时间，对特性 门控的调用可能一直被允许，前提是特性门控对功能本身无影响且特性门控不会 引发任何错误。

意在允许用户禁用的功能特性应该包含一个在相关联的特性门控中禁用该功能的机制。

特性门控的版本管理与之前讨论的组件版本管理不同，因此其对应的弃用策略如下：

规则 #8：特性门控所对应的功能特性经历下面所列的成熟性阶段转换时，特性门控 必须被弃用。特性门控弃用时必须在以下时长内保持其功能可用：

- **Beta 特性转为 GA：6 个月或者 2 个发布版本（取其较长者）**
- **Beta 特性转为丢弃：3 个月或者 1 个发布版本（取其较长者）**
- **Alpha 特性转为丢弃：0 个发布版本**

规则 #9：已弃用的特色门控再被使用时必须给出警告回应。当特性门控被 弃用时，必须在发布说明和对应的 CLI 帮助信息中通过文档宣布。 警告信息和文档都要标明是否某特性门控不再起作用。

例外

没有策略可以覆盖所有情况。此策略文档是一个随时被更新的文档，会随着时间 推移演化。在实践中，会有一些情况无法很好地匹配到这里的弃用策略，或者 这里的策略变成了很严重的羁绊。这类情形要与 SIG 和项目牵头人讨论， 寻求对应场景的最佳解决方案。请一直铭记，Kubernetes 承诺要成为一个 稳定的系统，至少会尽力做到不会影响到其用户。此弃用策略的任何例外情况 都会在所有相关的发布说明中公布。

反馈

最后修改 November 30, 2020 at 9:43 PM PST: [zh] Sync using-api (9af43634c)

Kubernetes API 健康端点

Kubernetes API 服务器 提供 API 端点以指示 API 服务器的当前状态。 本文描述了这些 API 端点，并说明如何使用。

API 健康端点

Kubernetes API 服务器提供 3 个 API 端点（healthz、livez 和 readyz）来表明 API 服务器的当前状态。 healthz 端点已被弃用（自 Kubernetes v1.16 起），你应该使用更为明确的 livez 和 readyz 端点。 livez 端点可与 --livez-grace-period 标志一起使用，来指定启动持续时间。 为了正常关机，你可以使用 /readyz 端点并指定 --shutdown-delay-duration 标志。 检查 API 服务器的 health/livez/readyz 端点的机器应依赖于 HTTP 状态代码。 状态码 200 表示 API 服务器是 healthy、live 还是 ready，具体取决于所调用的端点。 以下更详细的选项供操作人员使用，用来调试其集群或专门调试 API 服务器的状态。

以下示例将显示如何与运行状况 API 端点进行交互。

对于所有端点，都可以使用 verbose 参数来打印检查项以及检查状态。 这对于操作人员调试 API 服务器的当前状态很有用，这些不打算给机器使用：

```
curl -k https://localhost:6443/livez?verbose
```

或从具有身份验证的远程主机：

```
kubectl get --raw='/readyz?verbose'
```

输出将如下所示：

```
[+]ping ok
[+]log ok
[+]etcd ok
[+]poststarthook/start-kube-apiserver-admission-initializer ok
[+]poststarthook/generic-apiserver-start-informers ok
[+]poststarthook/start-apiextensions-informers ok
[+]poststarthook/start-apiextensions-controllers ok
[+]poststarthook/crd-informer-synced ok
[+]poststarthook/bootstrap-controller ok
[+]poststarthook/rbac/bootstrap-roles ok
[+]poststarthook/scheduling/bootstrap-system-priority-classes ok
[+]poststarthook/start-cluster-authentication-info-controller ok
[+]poststarthook/start-kube-aggregator-informers ok
[+]poststarthook/apiservice-registration-controller ok
[+]poststarthook/apiservice-status-available-controller ok
[+]poststarthook/kube-apiserver-autoregistration ok
[+]autoregister-completion ok
[+]poststarthook/apiservice-openapi-controller ok
healthz check passed
```

Kubernetes API 服务器也支持排除特定的检查项。 查询参数也可以像以下示例一样进行组合：

```
curl -k 'https://localhost:6443/readyz?verbose&exclude=etcd'
```

输出显示排除了 etcd 检查：

```
[+]ping ok
[+]log ok
[+]etcd excluded: ok
[+]poststarthook/start-kube-apiserver-admission-initializer ok
[+]poststarthook/generic-apiserver-start-informers ok
[+]poststarthook/start-apiextensions-informers ok
[+]poststarthook/start-apiextensions-controllers ok
[+]poststarthook/crd-informer-synced ok
```

```
[+]poststarthook/bootstrap-controller ok
[+]poststarthook/rbac/bootstrap-roles ok
[+]poststarthook/scheduling/bootstrap-system-priority-classes ok
[+]poststarthook/start-cluster-authentication-info-controller ok
[+]poststarthook/start-kube-aggregator-informers ok
[+]poststarthook/apiservice-registration-controller ok
[+]poststarthook/apiservice-status-available-controller ok
[+]poststarthook/kube-apiserver-autoregistration ok
[+]autoregister-completion ok
[+]poststarthook/apiservice-openapi-controller ok
[+]shutdown ok
healthz check passed
```

独立健康检查

FEATURE STATE: Kubernetes v1.20 [alpha]

每个单独的健康检查都会公开一个 http 端点，并且可以单独检查。 单个运行状况检查的模式为 /livez/<healthcheck-name>，其中 livez 和 readyz 表明你要检查的是 API 服务器是否存活或就绪。 <healthcheck-name> 的路径可以通过上面的 verbose 参数发现 ，并采用 [+] 和 ok 之间的路径。 这些单独的健康检查不应由机器使用，但对于操作人员调试系统而言，是有帮助的：

```
curl -k https://localhost:6443/livez/etcd
```

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 24, 2020 at 10:28 AM PST: remove 'shell' form fence. (1962f7dae)

访问 API

关于 Kubernetes 如何实现和控制 API 访问的介绍性材料，可阅读 控制 Kubernetes API 的访问。

参考文档：

- 身份认证
  - 使用启动引导令牌来执行身份认证

- 准入控制器
  - 动态准入控制
- 鉴权与授权
  - 基于角色的访问控制
  - 基于属性的访问控制
  - 节点鉴权
  - Webhook 鉴权
- 证书签名请求
  - 包含 CSR 的批复 和证书签名
- 服务账号
  - 开发者指南
  - 管理文档

用户认证

本页提供身份认证有关的概述。

Kubernetes 中的用户

所有 Kubernetes 集群都有两类用户：由 Kubernetes 管理的服务账号和普通用户。

Kubernetes 假定普通用户是由一个与集群无关的服务通过以下方式之一进行管理的：

- 负责分发私钥的管理员
- 类似 Keystone 或者 Google Accounts 这类用户数据库
- 包含用户名和密码列表的文件

有鉴于此，*Kubernetes 并不包含用来代表普通用户账号的对象*。 普通用户的信息无法通过 API 调用添加到集群中。

尽管无法通过 API 调用来添加普通用户，Kubernetes 仍然认为能够提供由集群的证书机构签名的合法证书的用户是通过身份认证的用户。基于这样的配置，Kubernetes 使用证书中的 'subject' 的通用名称（Common Name）字段（例如，"/CN=bob"）来 确定用户名。接下来，基于角色访问控制（RBAC）子系统会确定用户是否有权针对 某资源执行特定的操作。进一步的细节可参阅 证书请求 下普通用户主题。

与此不同，服务账号是 Kubernetes API 所管理的用户。它们被绑定到特定的名字空间， 或者由 API 服务器自动创建，或者通过 API 调用创建。服务账号与一组以 Secret 保存 的凭据相关，这些凭据会被挂载到 Pod 中，从而允许集群内的进程访问 Kubernetes API。

API 请求则或者与某普通用户相关联，或者与某服务账号相关联，亦或者被视作 匿名请求。这意味着集群内外的每个进程在向 API 服务器发起 请求时都必须通过身份认证，否则会被视作匿名用户。这里的进程可以是在某工作站上 输入 kubectl 命令的操作人员，也可以是节点上的 kubelet 组件，还可以是控制面 的成员。

身份认证策略

Kubernetes 使用身份认证插件利用客户端证书、持有者令牌（Bearer Token）、身份认证代理（Proxy）或者 HTTP 基本认证机制来认证 API 请求的身份。HTTP 请求发给 API 服务器时， 插件会将以下属性关联到请求本身：

- 用户名：用来辨识最终用户的字符串。常见的值可以是 kube-admin 或 jane@example.com。
- 用户 ID：用来辨识最终用户的字符串，旨在比用户名有更好的一致性和唯一性。
- 用户组：取值为一组字符串，其中各个字符串用来标明用户是某个命名的用户逻辑集合的成员。 常见的值可能是 system:masters 或者 devops-team 等。
- 附加字段：一组额外的键-值映射，键是字符串，值是一组字符串；用来保存一些鉴权组件可能 觉得有用的额外信息。

所有（属性）值对于身份认证系统而言都是不透明的，只有被 鉴权组件 解释过之后才有意义。

你可以同时启用多种身份认证方法，并且你通常会至少使用两种方法：

- 针对服务账号使用服务账号令牌
- 至少另外一种方法对用户的身份进行认证

当集群中启用了多个身份认证模块时，第一个成功地对请求完成身份认证的模块会 直接做出评估决定。API 服务器并不保证身份认证模块的运行顺序。

对于所有通过身份认证的用户，system:authenticated 组都会被添加到其组列表中。

与其它身份认证协议（LDAP、SAML、Kerberos、X509 的替代模式等等）都可以通过 使用一个身份认证代理或 身份认证 Webhoook来实现。

X509 客户证书

通过给 API 服务器传递 --client-ca-file=SOMEFILE 选项，就可以启动客户端证书身份认证。 所引用的文件必须包含一个或者多个证书机构，用来验证向 API 服务器提供的客户端证书。 如果提供了客户端证书并且证书被验证通过，则 subject 中的公共名称（Common Name）就被 作为请求的用户名。 自 Kubernetes 1.4 开始，客户端证书还可以通过证书的 organization 字段标明用户的组成员信息。 要包含用户的多个组成员信息，可以在证书种包含多个 organization 字段。

例如，使用 openssl 命令行工具生成一个证书签名请求：

```
openssl req -new -key jbeda.pem -out jbeda-csr.pem -subj "/CN=jbeda/O=app1/O=app2"
```

此命令将使用用户名 jbeda 生成一个证书签名请求（CSR），且该用户属于 "app" 和 "app2" 两个用户组。

参阅管理证书了解如何生成客户端证书。

静态令牌文件

当 API 服务器的命令行设置了 --token-auth-file=SOMEFILE 选项时，会从文件中 读取持有者令牌。目前，令牌会长期有效，并且在不重启 API 服务器的情况下 无法更改令牌列表。

令牌文件是一个 CSV 文件，包含至少 3 个列：令牌、用户名和用户的 UID。 其余列被视为可选的组名。

> **说明：**
>
> 如果要设置的组名不止一个，则对应的列必须用双引号括起来，例如
>
> token,user,uid,"group1,group2,group3"

在请求中放入持有者令牌

当使用持有者令牌来对某 HTTP 客户端执行身份认证时，API 服务器希望看到 一个名为 Authorization 的 HTTP 头，其值格式为 Bearer THETOKEN。 持有者令牌必须是一个可以放入 HTTP 头部值字段的字符序列，至多可使用 HTTP 的编码和引用机制。 例如：如果持有者令牌为 31ada4fd-adec-460c-809a-9e56ceb75269，则其 出现在 HTTP 头部时如下所示：

```
Authorization: Bearer 31ada4fd-adec-460c-809a-9e56ceb75269
```

启动引导令牌

FEATURE STATE: Kubernetes v1.18 [stable]

为了支持平滑地启动引导新的集群，Kubernetes 包含了一种动态管理的持有者令牌类型， 称作 *启动引导令牌（Bootstrap Token）*。 这些令牌以 Secret 的形式保存在 kube-system 名字空间中，可以被动态管理和创建。 控制器管理器包含的 TokenCleaner 控制器能够在启动引导令牌过期时将其删除。

这些令牌的格式为 [a-z0-9]{6}.[a-z0-9]{16}。第一个部分是令牌的 ID；第二个部分 是令牌的 Secret。你可以用如下所示的方式来在 HTTP 头部设置令牌：

```
Authorization: Bearer 781292.db7bc3a58fc5f07e
```

你必须在 API 服务器上设置 --enable-bootstrap-token-auth 标志来启用基于启动 引导令牌的身份认证组件。 你必须通过控制器管理器的 --controllers 标志来启用 TokenCleaner 控制器； 这可以通过类似 --controllers=*,tokencleaner 这种设置来做到。 如果你使用 kubeadm 来启动引导新的集群，该工具会帮你完成这些设置。

身份认证组件的认证结果为 system:bootstrap:<令牌 ID>，该用户属于 system:bootstrappers 用户组。 这里的用户名和组设置都是有意设计成这样，其目的是阻止用户在启动引导集群之后 继续使用这些令牌。 这里的用户名和组名可以用来（并且已经被 kubeadm 用来）构造合适的鉴权 策略，以完成启动引导新集群的工作。

请参阅启动引导令牌 以了解关于启动引导令牌身份认证组件与控制器的更深入的信息，以及如何使用 kubeadm 来管理这些令牌。

服务账号令牌

服务账号（Service Account）是一种自动被启用的用户认证机制，使用经过签名的 持有者令牌来验证请求。该插件可接受两个可选参数：

- --service-account-key-file 一个包含用来为持有者令牌签名的 PEM 编码密钥。若未指定，则使用 API 服务器的 TLS 私钥。
- --service-account-lookup 如果启用，则从 API 删除的令牌会被回收。

服务账号通常由 API 服务器自动创建并通过 ServiceAccount 准入控制器 关联到集群中运行的 Pod 上。 持有者令牌会挂载到 Pod 中可预知的位置，允许集群内进程与 API 服务器通信。 服务账号也可以使用 Pod 规约的 serviceAccountName 字段显式地关联到 Pod 上。

> **说明：** serviceAccountName 通常会被忽略，因为关联关系是自动建立的。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 3
  template:
    metadata:
    # ...
    spec:
      serviceAccountName: bob-the-bot
      containers:
      - name: nginx
        image: nginx:1.14.2
```

在集群外部使用服务账号持有者令牌也是完全合法的，且可用来为长时间运行的、需要与 Kubernetes API 服务器通信的任务创建标识。要手动创建服务账号，可以使用 kubectl create serviceaccount <名称> 命令。此命令会在当前的名字空间中生成一个服务账号和一个与之关联的 Secret。

```
kubectl create serviceaccount jenkins
```

```
serviceaccount/jenkins created
```

查验相关联的 Secret：

```
kubectl get serviceaccounts jenkins -o yaml
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  # ...
secrets:
- name: jenkins-token-1yvwg
```

所创建的 Secret 中会保存 API 服务器的公开的 CA 证书和一个已签名的 JSON Web 令牌（JWT）。

```
kubectl get secret jenkins-token-1yvwg -o yaml
```

```
apiVersion: v1
data:
  ca.crt: <Base64 编码的 API 服务器 CA>
  namespace: ZGVmYXVsdA==
  token: <Base64 编码的持有者令牌>
kind: Secret
metadata:
  # ...
type: kubernetes.io/service-account-token
```

> **说明：** 字段值是按 Base64 编码的，这是因为 Secret 数据总是采用 Base64 编码来存储。

已签名的 JWT 可以用作持有者令牌，并将被认证为所给的服务账号。 关于如何在请求中包含令牌，请参阅前文。 通常，这些 Secret 数据会被挂载到 Pod 中以便集群内访问 API 服务器时使用， 不过也可以在集群外部使用。

服务账号被身份认证后，所确定的用户名为 system:serviceaccount:<名字空间>:<服务账号>， 并被分配到用户组 system:serviceaccounts 和 system:serviceaccounts:<名字空间>。

警告：由于服务账号令牌保存在 Secret 对象中，任何能够读取这些 Secret 的用户 都可以被认证为对应的服务账号。在为用户授予访问服务账号的权限时，以及对 Secret 的读权限时，要格外小心。

OpenID Connect（OIDC）令牌

OpenID Connect 是一种 OAuth2 认证方式， 被某些 OAuth2 提供者支持，例如 Azure 活动目录、Salesforce 和 Google。 协议对 OAuth2 的主要扩充体现在有一个附加字段会和访问令牌一起返回， 这一字段称作 ID Token（ID 令牌）。 ID 令牌是一种由服务器签名的 JSON Web 令牌（JWT），其中包含一些可预知的字段， 例如用户的邮箱地址，

要识别用户，身份认证组件使用 OAuth2 令牌响应 中的 id_token（而非 access_token）作为持有者令牌。 关于如何在请求中设置令牌，可参见前文。

[JavaScript must be enabled to view content]

1. 登录到你的身份服务（Identity Provider）
2. 你的身份服务将为你提供 access_token、id_token 和 refresh_token
3. 在使用 kubectl 时，将 id_token 设置为 --token 标志值，或者将其直接添加到 kubeconfig 中
4. kubectl 将你的 id_token 放到一个称作 Authorization 的头部，发送给 API 服务器
5. API 服务器将负责通过检查配置中引用的证书来确认 JWT 的签名是合法的
6. 检查确认 id_token 尚未过期
7. 确认用户有权限执行操作
8. 鉴权成功之后，API 服务器向 kubectl 返回响应
9. kubectl 向用户提供反馈信息

由于用来验证你是谁的所有数据都在 id_token 中，Kubernetes 不需要再去联系 身份服务。在一个所有请求都是无状态请求的模型中，这一工作方式可以使得身份认证 的解决方案更容易处理大规模请求。不过，此访问也有一些挑战：

1. Kubernetes 没有提供用来触发身份认证过程的 "Web 界面"。 因为不存在用来收集用户凭据的浏览器或用户接口，你必须自己先行完成 对身份服务的认证过程。
2. id_token 令牌不可收回。因其属性类似于证书，其生命期一般很短（只有几分钟）， 所以，每隔几分钟就要获得一个新的令牌这件事可能很让人头疼。
3. 如果不使用 kubectl proxy 命令或者一个能够注入 id_token 的反向代理， 向 Kubernetes 控制面板执行身份认证是很困难的。

配置 API 服务器

要启用此插件，须在 API 服务器上配置以下标志：

参数	描述	示例	必需?
--oidc-issuer-url	允许 API 服务器发现公开的签名密钥的服务的 URL。只接受模式为 https:// 的 URL。此值通常设置为服务的发现 URL，不含路径。例如："https://accounts.google.com" 或 "https://login.salesforce.com"。此 URL 应指向 .well-known/openid-configuration 下一层的路径。	如果发现 URL 是 https://accounts.google.com/.well-known/openid-configuration，则此值应为 https://accounts.google.com	是
--oidc-client-id	所有令牌都应发放给此客户 ID。	kubernetes	是

参数	描述	示例	必需？
--oidc-username-claim	用作用户名的 JWT 申领（JWT Claim）。默认情况下使用 sub 值，即最终用户的一个唯一的标识符。管理员也可以选择其他申领，例如 email 或者 name，取决于所用的身份服务。不过，除了 email 之外的申领都会被添加令牌发放者的 URL 作为前缀，以免与其他插件产生命名冲突。	sub	否
--oidc-username-prefix	要添加到用户名申领之前的前缀，用来避免与现有用户名发生冲突（例如：system: 用户）。例如，此标志值为 oidc: 时将创建形如 oidc:jane.doe 的用户名。如果此标志未设置，且 --oidc-username-claim 标志值不是 email，则默认前缀为 <令牌发放者的 URL># ，其中 <令牌发放者 URL > 的值取自 --oidc-issuer-url 标志的设定。此标志值为 - 时，意味着禁止添加用户名前缀。	oidc:	否
--oidc-groups-claim	用作用户组名的 JWT 申领。如果所指定的申领确实存在，则其值必须是一个字符串数组。	groups	否
--oidc-groups-prefix	添加到组申领的前缀，用来避免与现有用户组名（如：system: 组）发生冲突。例如，此标志值为 oidc: 时，所得到的用户组名形如 oidc:engineering 和 oidc:infra。	oidc:	否
--oidc-required-claim	取值为一个 key=value 偶对，意为 ID 令牌中必须存在的申领。如果设置了此标志，则 ID 令牌会被检查以确定是否包含取值匹配的申领。此标志可多次重复，以指定多个申领。	claim=value	否
--oidc-ca-file	指向一个 CA 证书的路径，该 CA 负责对你的身份服务的 Web 证书提供签名。默认值为宿主系统的根 CA。	/etc/kubernetes/ssl/kc-ca.pem	否

很重要的一点是，API 服务器并非一个 OAuth2 客户端，相反，它只能被配置为 信任某一个令牌发放者。这使得使用公共服务（如 Google）的用户可以不信任发放给 第三方的凭据。 如果管理员希望使用多个 OAuth 客户端，他们应该研究一下那些支持 azp（Authorized Party，被授权方）申领的服务。 azp 是一种允许某客户端代替另一客户端发放令牌的机制。

Kubernetes 并未提供 OpenID Connect 的身份服务。 你可以使用现有的公共的 OpenID Connect 身份服务（例如 Google 或者 其他服务）。 或者，你也可以选择自

已运行一个身份服务，例如 CoreOS dex、 Keycloak、 CloudFoundry UAA 或者 Tremolo Security 的 OpenUnison。

要在 Kubernetes 环境中使用某身份服务，该服务必须：

1. 支持 OpenID connect 发现； 但事实上并非所有服务都具备此能力
2. 运行 TLS 协议且所使用的加密组件都未过时
3. 拥有由 CA 签名的证书（即使 CA 不是商业 CA 或者是自签名的 CA 也可以）

关于上述第三条需求，即要求具备 CA 签名的证书，有一些额外的注意事项。 如果你部署了自己的身份服务，而不是使用云厂商（如 Google 或 Microsoft）所提供的服务，你必须对身份服务的 Web 服务器证书进行签名，签名所用证书的 CA 标志要设置为 TRUE，即使用的是自签名证书。这是因为 GoLang 的 TLS 客户端实现对证书验证 标准方面有非常严格的要求。如果你手头没有现成的 CA 证书，可以使用 CoreOS 团队所开发的这个脚本 来创建一个简单的 CA 和被签了名的证书与密钥对。 或者你也可以使用 这个类似的脚本， 生成一个合法期更长、密钥尺寸更大的 SHA256 证书。

特定系统的安装指令：

- UAA
- Dex
- OpenUnison

使用 kubectl

选项一 - OIDC 身份认证组件

第一种方案是使用 kubectl 的 oidc 身份认证组件，该组件将 id_token 设置 为所有请求的持有者令牌，并且在令牌过期时自动刷新。在你登录到你的身份服务之后， 可以使用 kubectl 来添加你的 id_token、refresh_token、client_id 和 client_secret，以配置该插件。

如果服务在其刷新令牌响应中不包含 id_token，则此插件无法支持该服务。 这时你应该考虑下面的选项二。

```
kubectl config set-credentials USER_NAME \
  --auth-provider=oidc \
  --auth-provider-arg=idp-issuer-url=( issuer url ) \
  --auth-provider-arg=client-id=( your client id ) \
  --auth-provider-arg=client-secret=( your client secret ) \
  --auth-provider-arg=refresh-token=( your refresh token ) \
  --auth-provider-arg=idp-certificate-authority=( path to your ca certificate ) \
  --auth-provider-arg=id-token=( your id_token )
```

作为示例，在完成对你的身份服务的身份认证之后，运行下面的命令：

```
kubectl config set-credentials mmosley  \
      --auth-provider=oidc  \
      --auth-provider-arg=idp-issuer-url=https://oidcidp.tremolo.lan:8443/auth/
idp/OidcIdP  \
```

```
      --auth-provider-arg=client-id=kubernetes  \
      --auth-provider-arg=client-secret=1db158f6-177d-4d9c-8a8b-
d36869918ec5  \
      --auth-provider-arg=refresh-token=q1bKLFOyUiosTfawzA93TzZIDzH2TNa2S
Mm0zEiPKTUwME6BkEo6Sql5yUWVBSWpKUGphaWpxSVAfekBOZbBhaEW+VlFUe
VRGcluyVF5JT4+haZmPsluFoFu5XkpXk5BXqHega4GAXlF+ma+vmYpFcHe5eZR+sl
BFpZKtQA= \
      --auth-provider-arg=idp-certificate-authority=/root/ca.pem \
      --auth-provider-arg=id-token=eyJraWQiOiJDTj1vaWRjaWRwLnRyZW1vbG8u
bGFuLCBPVT1EZW1vLCBPPVRybWVvbG8gU2VjdXJpdHksIEw9QXJsaW5ndG9uLCB
TVD1WaXJnaW5pYSwgQz1VUy1DTj1rdWJlLWNhLTEyMDIxNDc5MjEwMzYwNzMy
MTUyIiwiYWxnIjoiUlMyNTYifQ.eyJpc3MiOiJodHRwczovL29pZGNpZHAudHJlbW9s
by5sYW46ODQ0My9hdXRoL2lkcC9PaWRjSWRQIiwiYXVkIjoia3ViZXJuZXRlcyIsImV
4cCI6MTQ4MzU0OTUxMSwianRpIjoiMm96US15TXdFcHV4WDlHZUhQdy1hZyIsIm
lhdCI6MTQ4MzU0OTQ1MSwibmJmIjoxNDgzNTQ5MzMxLCJzdWIiOiI0YWViMzdiY
S1iNjQ1LTQ4ZmQtYWIzMC0xYTAxZWU0MWUyMTgifQ.w6p4J_6qQ1HzTG9nrEOru
bxIMb9K5hzcMPxc9IxPx2K4xO9l-
oFiUw93daH3m5pluP6K7eOE6txBuRVfEcpJSwlelsOsW8gb8VJcnzMS9EnZpeA0tW_
p-mnkFc3VcfyXuhe5R3G7aa5d8uHv70yJ9Y3-
UhjiN9EhpMdfPAoEB9fYKKkJRzF7utTTIPGrSaSU6d2pcpfYKaxIwePzEkT4DfcQthoZ
dy9ucNvvLoi1DIC-
UocFD8HLs8LYKEqSxQvOcvnThbObJ9af71EwmuE21fO5KzMW20KtAeget1gnldOo
sPtz1G5EwvaQ401-RPQzPGMVBld0_zMCAwZttJ4knw
```

此操作会生成以下配置：

```
users:
- name: mmosley
  user:
    auth-provider:
      config:
        client-id: kubernetes
        client-secret: 1db158f6-177d-4d9c-8a8b-d36869918ec5
        id-token: eyJraWQiOiJDTj1vaWRjaWRwLnRyZW1vbG8ubGFuLCBPVT1EZW1v
LCBPPVRybWVvbG8gU2VjdXJpdHksIEw9QXJsaW5ndG9uLCBTVD1WaXJnaW5pYS
wgQz1VUy1DTj1rdWJlLWNhLTEyMDIxNDc5MjEwMzYwNzMyMTUyIiwiYWxnIjoiUl
MyNTYifQ.eyJpc3MiOiJodHRwczovL29pZGNpZHAudHJlbW9sby5sYW46ODQ0My
9hdXRoL2lkcC9PaWRjSWRQIiwiYXVkIjoia3ViZXJuZXRlcyIsImV4cCI6MTQ4MzU0OT
UxMSwianRpIjoiMm96US15TXdFcHV4WDlHZUhQdy1hZyIsImlhdCI6MTQ4MzU0O
TQ1MSwibmJmIjoxNDgzNTQ5MzMxLCJzdWIiOiI0YWViMzdiYS1iNjQ1LTQ4ZmQtY
WIzMC0xYTAxZWU0MWUyMTgifQ.w6p4J_6qQ1HzTG9nrEOrubxIMb9K5hzcMPxc9
IxPx2K4xO9l-
oFiUw93daH3m5pluP6K7eOE6txBuRVfEcpJSwlelsOsW8gb8VJcnzMS9EnZpeA0tW_
p-mnkFc3VcfyXuhe5R3G7aa5d8uHv70yJ9Y3-
UhjiN9EhpMdfPAoEB9fYKKkJRzF7utTTIPGrSaSU6d2pcpfYKaxIwePzEkT4DfcQthoZ
dy9ucNvvLoi1DIC-
```

```
UocFD8HLs8LYKEqSxQvOcvnThbObJ9af71EwmuE21fO5KzMW20KtAeget1gnldOo
sPtz1G5EwvaQ401-RPQzPGMVBld0_zMCAwZttJ4knw
      idp-certificate-authority: /root/ca.pem
      idp-issuer-url: https://oidcidp.tremolo.lan:8443/auth/idp/OidcIdP
      refresh-token: q1bKLFOyUiosTfawzA93TzZIDzH2TNa2SMm0zEiPKTUwME6Bk
Eo6Sql5yUWVBSWpKUGphaWpxSVAfekBOZbBhaEW+VlFUeVRGcluyVF5JT4+haZ
mPsluFoFu5XkpXk5BXq
    name: oidc
```

当你的 id_token 过期时，kubectl 会尝试使用你的 refresh_token 来刷新你的 id_token ，并且在 client_secret 中存放 refresh_token 的新值，同时把 id_token 的新值写入到 .kube/config 文件中。

选项二 - 使用 --token 选项

kubectl 命令允许你使用 --token 选项传递一个令牌。 你可以将 id_token 的内容复制粘贴过来，作为此标志的取值：

```
kubectl --token=eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL21sYi50cmVtb2x
vLmxhbjo4MDQzL2F1dGgvaWRwL29pZGMiLCJhdWQiOiJrdWJlcm5ldGVzIiwiZXhw
IjoxNDc0NTk2NjY5LCJqdGkiOiI2RDUzNXoxUEpFNjJOR3QxaWVyYm9RIiwiaWF0Ijo
xNDc0NTk2MzY5LCJuYmYiOjE0NzQ1OTYyNDksInN1YiI6Im13aW5kdSIsInVzZXJfc
m9sZSI6WyJ1c2VycyIsIm5ldy1uYW1lc3BhY2Utdmlld2VyIl0sImVtYWlsIjoibXdpbm
R1QG5vbW9yZWplZGkuY29tIn0.f2As579n9VNoaKzoF-dOQGmXkFKf1FMyNV0-
va_B63jn-_n9LGSCca_6IVMP8pO-Zb4KvRqGyTP0r3HkHxYy5c81AnIh8ijarruczl-
TK_yF5akjSTHFZD-0gRzlevBDiH8Q79NAr-
ky0P4iIXS8lY9Vnjch5MF74Zx0c3alKJHJUnnpjIACByfF2SCaYzbWFMUNat-K1PaUk5-
ujMBG7yYnr95xD-63n8CO8teGUAAEMx6zRjzfhnhbzX-
ajwZLGwGUBT4WqjMs70-6a7_8gZmLZb2az1cZynkFRj2BaCkVT3A2RrjeEwZEtGXlM
qKJ1_I2ulrOVsYx01_yD35-rw get nodes
```

Webhook 令牌身份认证

Webhook 身份认证是一种用来验证持有者令牌的回调机制。

- --authentication-token-webhook-config-file 指向一个配置文件，其中描述 如何访问远程的 Webhook 服务。
- --authentication-token-webhook-cache-ttl 用来设定身份认证决定的缓存时间。默认时长为 2 分钟。

配置文件使用 kubeconfig 文件的格式。文件中，clusters 指代远程服务，users 指代远程 API 服务 Webhook。下面是一个例子：

```
# Kubernetes API 版本
apiVersion: v1
# API 对象类别
kind: Config
```

```yaml
# clusters 指代远程服务
clusters:
  - name: name-of-remote-authn-service
    cluster:
      certificate-authority: /path/to/ca.pem         # 用来验证远程服务的 CA
      server: https://authn.example.com/authenticate # 要查询的远程服务 URL。必须使用 'https'。

# users 指代 API 服务的 Webhook 配置
users:
  - name: name-of-api-server
    user:
      client-certificate: /path/to/cert.pem # Webhook 插件要使用的证书
      client-key: /path/to/key.pem          # 与证书匹配的密钥

# kubeconfig 文件需要一个上下文（Context），此上下文用于本 API 服务器
current-context: webhook
contexts:
- context:
    cluster: name-of-remote-authn-service
    user: name-of-api-sever
  name: webhook
```

当客户端尝试在 API 服务器上使用持有者令牌完成身份认证（ 如前所述）时， 身份认证 Webhook 会用 POST 请求发送一个 JSON 序列化的对象到远程服务。 该对象是 authentication.k8s.io/v1beta1 组的 TokenReview 对象， 其中包含持有者令牌。 Kubernetes 不会强制请求提供此 HTTP 头部。

要注意的是，Webhook API 对象和其他 Kubernetes API 对象一样，也要受到同一 版本兼容规则约束。 实现者要了解对 Beta 阶段对象的兼容性承诺，并检查请求的 apiVersion 字段， 以确保数据结构能够正常反序列化解析。此外，API 服务器必须启用 authentication.k8s.io/v1beta1 API 扩展组 （ --runtime-config=authentication.k8s.io/v1beta1=true ）。

POST 请求的 Body 部分将是如下格式：

```json
{
  "apiVersion": "authentication.k8s.io/v1beta1",
  "kind": "TokenReview",
  "spec": {
    "token": "<持有者令牌>"
  }
}
```

远程服务应该会填充请求的 status 字段，以标明登录操作是否成功。 响应的 Body 中的 spec 字段会被忽略，因此可以省略。 如果持有者令牌验证成功，应该返回如下所示的响应：

```
{
  "apiVersion": "authentication.k8s.io/v1beta1",
  "kind": "TokenReview",
  "status": {
    "authenticated": true,
    "user": {
      "username": "janedoe@example.com",
      "uid": "42",
      "groups": [
        "developers",
        "qa"
      ],
      "extra": {
        "extrafield1": [
          "extravalue1",
          "extravalue2"
        ]
      }
    }
  }
}
```

而不成功的请求会返回：

```
{
  "apiVersion": "authentication.k8s.io/v1beta1",
  "kind": "TokenReview",
  "status": {
    "authenticated": false
  }
}
```

HTTP 状态码可用来提供进一步的错误语境信息。

身份认证代理

API 服务器可以配置成从请求的头部字段值（如 X-Remote-User）中辩识用户。 这一设计是用来与某身份认证代理一起使用 API 服务器，代理负责设置请求的头部字段值。

- --requestheader-username-headers 必需字段，大小写不敏感。用来设置要获得用户身份所要检查的头部字段名称列表（有序）。第一个包含数值的字段会被用来提取用户名。
- --requestheader-group-headers 可选字段，在 Kubernetes 1.6 版本以后支持，大小写不敏感。 建议设置为 "X-Remote-Group"。用来指定一组头部字段名称列表，以供检查用户所属的组名称。 所找到的全部头部字段的取值都会被用作用户组名。

- --requestheader-extra-headers-prefix 可选字段，在 Kubernetes 1.6 版本以后支持，大小写不敏感。 建议设置为 "X-Remote-Extra-"。用来设置一个头部字段的前缀字符串，API 服务器会基于所给 前缀来查找与用户有关的一些额外信息。这些额外信息通常用于所配置的鉴权插件。 API 服务器会将与所给前缀匹配的头部字段过滤出来，去掉其前缀部分，将剩余部分 转换为小写字符串并在必要时执行百分号解码 后，构造新的附加信息字段键名。原来的头部字段值直接作为附加信息字段的值。

  说明： 在 1.13.3 版本之前（包括 1.10.7、1.9.11），附加字段的键名只能包含 HTTP 头部标签的合法字符。

例如，使用下面的配置：

```
--requestheader-username-headers=X-Remote-User
--requestheader-group-headers=X-Remote-Group
--requestheader-extra-headers-prefix=X-Remote-Extra-
```

针对所收到的如下请求：

```
GET / HTTP/1.1
X-Remote-User: fido
X-Remote-Group: dogs
X-Remote-Group: dachshunds
X-Remote-Extra-Acme.com%2Fproject: some-project
X-Remote-Extra-Scopes: openid
X-Remote-Extra-Scopes: profile
```

会生成下面的用户信息：

```
name: fido
groups:
- dogs
- dachshunds
extra:
  acme.com/project:
  - some-project
  scopes:
  - openid
  - profile
```

为了防范头部信息侦听，在请求中的头部字段被检视之前， 身份认证代理需要向 API 服务器提供一份合法的客户端证书， 供后者使用所给的 CA 来执行验证。 警告：*不要* 在不同的上下文中复用 CA 证书，除非你清楚这样做的风险是什么以及 应如何保护 CA 用法的机制。

- --requestheader-client-ca-file 必需字段，给出 PEM 编码的证书包。 在检查请求的头部字段以提取用户名信息之前，必须提供一个合法的客户端证书， 且该证书要能够被所给文件中的机构所验证。

- --requestheader-allowed-names 可选字段，用来给出一组公共名称（CN）。 如果此标志被设置，则在检视请求中的头部以提取用户信息之前，必须提供 包含此列表中所给的 CN 名的、合法的客户端证书。

匿名请求

启用匿名请求支持之后，如果请求没有被已配置的其他身份认证方法拒绝，则被视作 匿名请求（Anonymous Requests）。这类请求获得用户名 system:anonymous 和 对应的用户组 system:unauthenticated。

例如，在一个配置了令牌身份认证且启用了匿名访问的服务器上，如果请求提供了非法的持有者令牌，则会返回 401 Unauthorized 错误。 如果请求没有提供持有者令牌，则被视为匿名请求。

在 1.5.1-1.5.x 版本中，匿名访问默认情况下是被禁用的，可以通过为 API 服务器设定 --anonymous-auth=true 来启用。

在 1.6 及之后版本中，如果所使用的鉴权模式不是 AlwaysAllow，则匿名访问默认是被启用的。 从 1.6 版本开始，ABAC 和 RBAC 鉴权模块要求对 system:anonymous 用户或者 system:unauthenticated 用户组执行显式的权限判定，所以之前的为 * 用户或 * 用户组赋予访问权限的策略规则都不再包含匿名用户。

用户伪装

一个用户可以通过伪装（Impersonation）头部字段来以另一个用户的身份执行操作。 使用这一能力，你可以手动重载请求被身份认证所识别出来的用户信息。 例如，管理员可以使用这一功能特性来临时伪装成另一个用户，查看请求是否被拒绝， 从而调试鉴权策略中的问题，

带伪装的请求首先会被身份认证识别为发出请求的用户，之后会切换到使用被伪装的用户的用户信息。

- 用户发起 API 调用时 *同时* 提供自身的凭据和伪装头部字段信息
- API 服务器对用户执行身份认证
- API 服务器确认通过认证的用户具有伪装特权
- 请求用户的信息被替换成伪装字段的值
- 评估请求，鉴权组件针对所伪装的用户信息执行操作

以下 HTTP 头部字段可用来执行伪装请求：

- Impersonate-User：要伪装成的用户名
- Impersonate-Group：要伪装成的用户组名。可以多次指定以设置多个用户组。可选字段；要求 "Impersonate-User" 必须被设置。
- Impersonate-Extra-<附加名称>：一个动态的头部字段，用来设置与用户相关的附加字段。 此字段可选；要求 "Impersonate-User" 被设置。为了能够以一致的形式保留， <附加名称>部分必须是小写字符，如果有任何字符不是 合法的 HTTP 头部标签字符， 则必须是 utf8 字符，且转换为百分号编码。

说明： 在 1.11.3 版本之前（以及 1.10.7、1.9.11），<附加名称> 只能包含 合法的 HTTP 标签字符。

头部字段集合的示例：

```
Impersonate-User: jane.doe@example.com
Impersonate-Group: developers
Impersonate-Group: admins
Impersonate-Extra-dn: cn=jane,ou=engineers,dc=example,dc=com
Impersonate-Extra-acme.com%2Fproject: some-project
Impersonate-Extra-scopes: view
Impersonate-Extra-scopes: development
```

在使用 kubectl 时，可以使用 --as 标志来配置 Impersonate-User 头部字段值， 使用 --as-group 标志配置 Impersonate-Group 头部字段值。

```
kubectl drain mynode
```

```
Error from server (Forbidden): User "clark" cannot get nodes at the cluster scope. (get nodes mynode)
```

设置 --as 和 --as-group 标志：

```
kubectl drain mynode --as=superman --as-group=system:masters
```

```
node/mynode cordoned
node/mynode drained
```

要伪装成某个用户、某个组或者设置附加字段，执行伪装操作的用户必须具有对所伪装的类别（"user"、"group" 等）执行 "impersonate" 动词操作的能力。 对于启用了 RBAC 鉴权插件的集群，下面的 ClusterRole 封装了设置用户和组伪装字段 所需的规则：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: impersonator
rules:
- apiGroups: [""]
  resources: ["users", "groups", "serviceaccounts"]
  verbs: ["impersonate"]
```

附加字段会被作为 userextras 资源的子资源来执行权限评估。 如果要允许用户为附加字段 "scopes" 设置伪装头部，该用户需要被授予以下规则：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: scopes-impersonator
```

```yaml
rules:
# 可以设置 "Impersonate-Extra-scopes" 头部
- apiGroups: ["authentication.k8s.io"]
  resources: ["userextras/scopes"]
  verbs: ["impersonate"]
```

你也可以通过约束资源可能对应的 resourceNames 限制伪装头部的取值：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: limited-impersonator
rules:
# 可以伪装成用户 "jane.doe@example.com"
- apiGroups: [""]
  resources: ["users"]
  verbs: ["impersonate"]
  resourceNames: ["jane.doe@example.com"]

# 可以伪装成用户组 "developers" 和 "admins"
- apiGroups: [""]
  resources: ["groups"]
  verbs: ["impersonate"]
  resourceNames: ["developers","admins"]

# 可以将附加字段 "scopes" 伪装成 "view" 和 "development"
- apiGroups: ["authentication.k8s.io"]
  resources: ["userextras/scopes"]
  verbs: ["impersonate"]
  resourceNames: ["view", "development"]
```

client-go 凭据插件

FEATURE STATE: Kubernetes v1.11 [beta]

k8s.io/client-go 及使用它的工具（如 kubectl 和 kubelet）可以执行某个外部 命令来获得用户的凭据信息。

这一特性的目的是便于客户端与 k8s.io/client-go 并不支持的身份认证协议（LDAP、Kerberos、OAuth2、SAML 等）继承。 插件实现特定于协议的逻辑，之后返回不透明的凭据以供使用。 几乎所有的凭据插件使用场景中都需要在服务器端存在一个支持 Webhook 令牌身份认证组件的模块， 负责解析客户端插件所生成的凭据格式。

示例应用场景

在一个假想的应用场景中，某组织运行这一个外部的服务，能够将特定用户的已签名的令牌转换成 LDAP 凭据。此服务还能够对 Webhook 令牌身份认证组件的请求做出响应以 验证所提供的令牌。用户需要在自己的工作站上安装一个凭据插件。

要对 API 服务器认证身份时：

- 用户发出 kubectl 命令。
- 凭据插件提示用户输入 LDAP 凭据，并与外部服务交互，获得令牌。
- 凭据插件将令牌返回该 client-go，后者将其用作持有者令牌提交给 API 服务器。
- API 服务器使用Webhook 令牌身份认证组件向 外部服务发出 TokenReview 请求。
- 外部服务检查令牌上的签名，返回用户的用户名和用户组信息。

配置

凭据插件通过 kubectl 配置文件 来作为 user 字段的一部分设置。

```yaml
apiVersion: v1
kind: Config
users:
- name: my-user
  user:
    exec:
      # 要执行的命令。必需。
      command: "example-client-go-exec-plugin"

      # 解析 ExecCredentials 资源时使用的 API 版本。必需。
      #
      # 插件返回的 API 版本必需与这里列出的版本匹配。
      #
      # 要与支持多个版本的工具（如 client.authentication.k8sio/v1alpha1）集成，
      # 可以设置一个环境变量或者向工具传递一个参数标明 exec 插件所期望的版本。
      apiVersion: "client.authentication.k8s.io/v1beta1"

      # 执行此插件时要设置的环境变量。可选字段。
      env:
      - name: "FOO"
        value: "bar"

      # 执行插件时要传递的参数。可选字段。
      args:
      - "arg1"
      - "arg2"

      # 当可执行文件不存在时显示给用户的文本。可选的。
```

```
    installHint: |
      需要 example-client-go-exec-plugin 来在当前集群上执行身份认证。可以通过以下命令安装：

      MacOS: brew install example-client-go-exec-plugin

      Ubuntu: apt-get install example-client-go-exec-plugin

      Fedora: dnf install example-client-go-exec-plugin

      ...

    # 是否使用 KUBERNETES_EXEC_INFO 环境变量的一部分向这个 exec 插件
    # 提供集群信息（可能包含非常大的 CA 数据）
    provideClusterInfo: true
clusters:
- name: my-cluster
  cluster:
    server: "https://172.17.4.100:6443"
    certificate-authority: "/etc/kubernetes/ca.pem"
    extensions:
    - name: client.authentication.k8s.io/exec # 为每个集群 exec 配置保留的扩展名
      extension:
        arbitrary: config
        this: 在设置 provideClusterInfo 时可通过环境变量 KUBERNETES_EXEC_INFO 指定
        you: ["can", "put", "anything", "here"]
contexts:
- name: my-cluster
  context:
    cluster: my-cluster
    user: my-user
current-context: my-cluster
```

解析相对命令路径时，kubectl 将其视为与配置文件比较而言的相对路径。 如果 KUBECONFIG 被设置为 /home/jane/kubeconfig，而 exec 命令为 ./bin/example-client-go-exec-plugin，则要执行的可执行文件为 /home/jane/bin/example-client-go-exec-plugin。

```
- name: my-user
  user:
    exec:
      # 对 kubeconfig 目录而言的相对路径
      command: "./bin/example-client-go-exec-plugin"
      apiVersion: "client.authentication.k8s.io/v1beta1"
```

输出和输出格式

所执行的命令会在 stdout 打印 ExecCredential 对象。 k8s.io/client-go 使用 status 中返回的凭据信息向 Kubernetes API 服务器 执行身份认证。

在交互式会话中运行时，stdin 是直接暴露给插件使用的。 插件应该使用 TTY check 来确定是否适合用交互方式请求用户输入。

与使用持有者令牌凭据，插件在 ExecCredential 的状态中返回一个令牌：

```
{
  "apiVersion": "client.authentication.k8s.io/v1beta1",
  "kind": "ExecCredential",
  "status": {
    "token": "my-bearer-token"
  }
}
```

另一种方案是，返回 PEM 编码的客户端证书和密钥，以便执行 TLS 客户端身份认证。如果插件在后续调用中返回了不同的证书或密钥，k8s.io/client-go 会终止其与服务器的连接，从而强制执行新的 TLS 握手过程。

如果指定了这种方式，则 clientKeyData 和 clientCertificateData 字段都必需存在。

clientCertificateData 字段可能包含一些要发送给服务器的中间证书（Intermediate Certificates）。

```
{
  "apiVersion": "client.authentication.k8s.io/v1beta1",
  "kind": "ExecCredential",
  "status": {
    "clientCertificateData": "-----BEGIN CERTIFICATE-----\n...\n-----END
CERTIFICATE-----",
    "clientKeyData": "-----BEGIN RSA PRIVATE KEY-----\n...\n-----END RSA
PRIVATE KEY-----"
  }
}
```

作为一种可选方案，响应中还可以包含以 RFC3339 时间戳格式给出的证书到期时间。证书到期时间的有无会有如下影响：

- 如果响应中包含了到期时间，持有者令牌和 TLS 凭据会被缓存，直到到期期限到来、 或者服务器返回 401 HTTP 状态码，或者进程退出。
- 如果未指定到期时间，则持有者令牌和 TLS 凭据会被缓存，直到服务器返回 401 HTTP 状态码或者进程退出。

```
{
  "apiVersion": "client.authentication.k8s.io/v1beta1",
  "kind": "ExecCredential",
```

```
  "status": {
    "token": "my-bearer-token",
    "expirationTimestamp": "2018-03-05T17:30:20-08:00"
  }
}
```

调用此插件时可以选择性地设置环境变量 KUBERNETES_EXEC_INFO。 该变量包含了此插件获取凭据所针对的集群信息。此信息可用于执行群集特定的凭据获取逻辑。 为了启用此行为，必须在 kubeconfig 中的 exec user 字段上设置provideClusterInfo字段。下面是上述 KUBERNETES_EXEC_INFO 环境变量的示例。

```
{
  "apiVersion": "client.authentication.k8s.io/v1beta1",
  "kind": "ExecCredential",
  "spec": {
    "cluster": {
      "server": "https://172.17.4.100:6443",
      "certificate-authority-data": "LS0t...",
      "config": {
        "arbitrary": "config",
        "this": "在设置 provideClusterInfo 时可通过环境变量 KUBERNETES_EXEC_INFO 指定",
        "you": ["can", "put", "anything", "here"]
      }
    }
  }
}
```

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 15, 2021 at 1:39 PM PST: Fix typo and enhance description (80471044f)

使用启动引导令牌（Bootstrap Tokens）认证

FEATURE STATE: Kubernetes v1.18 [stable]

启动引导令牌是一种简单的持有者令牌（Bearer Token），这种令牌是在新建集群 或者在现有集群中添加新节点时使用的。 它被设计成能够支持 kubeadm， 但是也可以被用在其他的案例中以便用户在不使用 kubeadm 的情况下启动集群。 它也被设计成可以通过 RBAC 策略，结合 Kubelet TLS 启动引导 系统进行工作。

启动引导令牌被定义成一个特定类型的 Secret （bootstrap.kubernetes.io/token），并存在于 kube-system 名字空间中。 这些 Secret 会被 API 服务器上的启动引导认证组件（Bootstrap Authenticator）读取。 控制器管理器中的控制器 TokenCleaner 能够删除过期的令牌。 这些令牌也被用来在节点发现的过程中会使用的一个特殊的 ConfigMap 对象。 BootstrapSigner 控制器也会使用这一 ConfigMap。

令牌格式

启动引导令牌使用 abcdef.0123456789abcdef 的形式。 更加规范地说，它们必须符合正则表达式 [a-z0-9]{6}\.[a-z0-9]{16}。

令牌的第一部分是 "Token ID"，它是一种公开信息，用于引用令牌并确保不会 泄露认证所使用的秘密信息。 第二部分是"令牌秘密（Token Secret）"，它应该被共享给受信的第三方。

启用启动引导令牌

启用启动引导令牌身份认证

启动引导令牌认证组件可以通过 API 服务器上的如下标志启用：

```
--enable-bootstrap-token-auth
```

启动引导令牌被启用后，可以作为持有者令牌的凭据，用于 API 服务器请求的身份认证。

```
Authorization: Bearer 07401b.f395accd246ae52d
```

令牌认证为用户名 system:bootstrap:<token id> 并且是组 system:bootstrappers 的成员。额外的组信息可以通过令牌的 Secret 来设置。

过期的令牌可以通过启用控制器管理器中的 tokencleaner 控制器来删除。

启动引导令牌的 Secret 格式

每个合法的令牌背后对应着 kube-system 名字空间中的某个 Secret 对象。 你可以从这里. 找到完整设计文档。

这是 Secret 看起来的样子。

```
apiVersion: v1
kind: Secret
metadata:
```

```yaml
  # name 必须是 "bootstrap-token-<token id>" 格式的
  name: bootstrap-token-07401b
  namespace: kube-system

# type 必须是 'bootstrap.kubernetes.io/token'
type: bootstrap.kubernetes.io/token
stringData:
  # 供人阅读的描述，可选。
  description: "The default bootstrap token generated by 'kubeadm init'."

  # 令牌 ID 和秘密信息，必需。
  token-id: 07401b
  token-secret: base64(f395accd246ae52d)

  # 可选的过期时间字段
  expiration: "2017-03-10T03:22:11Z"
  # 允许的用法
  usage-bootstrap-authentication: "true"
  usage-bootstrap-signing: "true"

  # 令牌要认证为的额外组，必须以 "system:bootstrappers:" 开头
  auth-extra-groups: system:bootstrappers:worker,system:bootstrappers:ingress
```

Secret 的类型必须是 bootstrap.kubernetes.io/token，而且名字必须是 bootstrap-token-<token id>。 令牌必须存在于 kube-system 名字空间中。

usage-bootstrap-* 成员表明这个 Secret 的用途。启用时，值必须设置为 true。

- usage-bootstrap-authentication 表示令牌可以作为持有者令牌用于 API 服务器的身份认证。
- usage-bootstrap-signing 表示令牌可被用于 cluster-info ConfigMap 的签名，就像下面描述的那样。

expiration 字段控制令牌的失效期。过期的令牌在用于身份认证时会被拒绝，在用于 ConfigMap 签名时会被忽略。 过期时间值是遵循 RFC3339 进行编码的 UTC 时间。启用 TokenCleaner 控制器会自动删除过期的令牌。

使用 kubeadm 管理令牌

你可以使用 kubeadm 工具管理运行中集群上的令牌。 参见 kubeadm token 文档 以了解详细信息。

ConfigMap 签名

除了身份认证，令牌还可以用于签名 ConfigMap。 这一用法发生在集群启动过程的早期，在客户端信任 API 服务器之前。 被签名的 ConfigMap 可以被共享令牌完成身份认证。

通过在控制器管理器上启用 bootstrapsigner 控制器可以启用 ConfigMap 签名特性。

```
--controllers=*,bootstrapsigner
```

被签名的 ConfigMap 是 kube-public 名字空间中的 cluster-info。 典型的工作流中，客户端在未经认证和忽略 TLS 报错的状态下读取这个 ConfigMap。 通过检查 ConfigMap 中嵌入的签名校验 ConfigMap 的载荷。

ConfigMap 会是这个样子的：

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-info
  namespace: kube-public
data:
  jws-kubeconfig-07401b: eyJhbGciOiJIUzI1NiIsImtpZCI6IjA3NDAxYiJ9..tYEfbo6zDNo40MQE07aZcQX2m3EB2rO3NuXtxVMYm9U
  kubeconfig: |
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority-data: <非常长的证书数据>
        server: https://10.138.0.2:6443
      name: ""
    contexts: []
    current-context: ""
    kind: Config
    preferences: {}
    users: []
```

ConfigMap 的 kubeconfig 成员是一个填好了集群信息的配置文件。 这里主要交换的信息是 certificate-authority-data。在将来可能会有扩展。

签名是一个使用 "detached" 模式生成的 JWS 签名。 为了检验签名，用户应该按照 JWS 规则（base64 编码且丢掉结尾的 = ）对 kubeconfig 的载荷进行编码。完成编码的载荷会被插入到两个句点中间，形成完整的 JWS。你可以使用完整的令牌（比如 07401b.f395accd246ae52d）作为共享密钥， 通过 HS256 方式 (HMAC-SHA256) 对 JWS 进行校验。 用户 *必须* 确保使用了 HS256。

> **警告：**
>
> 任何拥有了启动引导令牌的主体都可以为该令牌生成一个合法的签名。 当使用 ConfigMap 签名时，非常不建议针对很多客户使用相同的令牌，因为某个被攻击的 客户可能对另一个一来签名来开启 TLS 信任的客户发起中间人攻击。

参考 [kubeadm 实现细节] 了解更多信息。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 28, 2020 at 10:57 PM PST: [zh] Sync English site changes for authentication ref (b8c9b2fa3)

证书签名请求

FEATURE STATE: Kubernetes v1.19 [stable]

证书 API 支持 X.509 的自动化配置， 它为 Kubernetes API 的客户端提供一个编程接口， 用于从证书颁发机构（CA）请求并获取 X.509 证书。

CertificateSigningRequest（CSR）资源用来向指定的签名者申请证书签名， 在最终签名之前，申请可能被批准，也可能被拒绝。

请求签名流程

CertificateSigningRequest 资源类型允许客户使用它申请发放 X.509 证书。 CertificateSigningRequest 对象 在 spec.request 中包含一个 PEM 编码的 PKCS#10 签名请求。 CertificateSigningRequest 使用 spec.signerName 字段标示 *签名者*（请求的接收方）。 注意，spec.signerName 在 certificates.k8s.io/v1 之后的 API 版本是必填项。

创建完成的 CertificateSigningRequest，要先通过批准，然后才能签名。 根据所选的签名者，CertificateSigningRequest 可能会被 控制器自动批准。 否则，就必须人工批准， 人工批准可以使用 REST API（或 go 客户端），也可以执行 kubectl certificate approve 命令。 同样，CertificateSigningRequest 也可能被驳回， 这就相当于通知了指定的签名者，这个证书不能签名。

对于已批准的证书，下一步是签名。 对应的签名控制器首先验证签名条件是否满足，然后才创建证书。 签名控制器然后更新 CertificateSigningRequest， 将新证书保存到现有 CertificateSigningRequest 对象的 status.certificate 字段中。 此时，字段 status.certificate 要么为空，要么包含一个用 PEM 编码的 X.509 证书。 直到签名完成前，CertificateSigningRequest 的字段 status.certificate 都为空。

一旦 status.certificate 字段完成填充，请求既算完成， 客户端现在可以从 CertificateSigningRequest 资源中获取已签名的证书的 PEM 数据。 当然如果不满足签名条件，签名者可以拒签。

为了减少集群中遗留的过时的 CertificateSigningRequest 资源的数量， 一个垃圾收集控制器将会周期性地运行。 此垃圾收集器会清除在一段时间内没有改变过状态的 CertificateSigningRequests：

- 已批准的请求：1小时后自动删除
- 已拒绝的请求：1小时后自动删除
- 挂起的请求：1小时后自动删除

签名者

所有签名者都应该提供自己工作方式的信息， 以便客户端可以预期到他们的 CSR 将发生什么。 此类信息包括：

1. **信任分发**：信任（CA 证书包）是如何分发的。
2. **许可的主体**：当一个受限制的主体（subject）发送请求时，相应的限制和应对手段。
3. **许可的 x509 扩展**：包括 IP subjectAltNames、DNS subjectAltNames、Email subjectAltNames、URI subjectAltNames 等，请求一个受限制的扩展项时的应对手段。
4. **许可的密钥用途/扩展的密钥用途**：当用途和签名者在 CSR 中指定的用途不同时，相应的限制和应对手段。
5. **过期时间/证书有效期**：过期时间由签名者确定、由管理员配置，还是由 CSR 对象指定等， 以及过期时间与签名者在 CSR 中指定过期时间不同时的应对手段。
6. **允许/不允许 CA 位**：当 CSR 包含一个签名者并不允许的 CA 证书的请求时，相应的应对手段。

一般来说，当 CSR 被批准通过，且证书被签名后，status.certificate 字段 将包含一个 PEM 编码的 X.509 证书。 有些签名者在 status.certificate 字段中存储多个证书。 在这种情况下，签名者的说明文档应当指明附加证书的含义。 例如，这是要在 TLS 握手时提供的证书和中继证书。

PKCS#10 签名请求格式不允许设置证书的过期时间或者生命期。因此，证书的过期 时间或者生命期必须通过类似 CSR 对象的注解字段这种形式来设置。 尽管让签名者使用过期日期从理论上来讲也是可行的，目前还不存在哪个实现这样做了。 （内置的签名者都是用相同的 ClusterSigningDuration 配置选项，而该选项 中将生命期的默认值设为 1 年，且可通过 kube-controller-manager 的命令行选项 --cluster-signing-duration 来更改。）

Kubernetes 签名者

Kubernetes提供了内置的签名者，每个签名者都有一个众所周知的 signerName:

1. kubernetes.io/kube-apiserver-client：签名的证书将被 API 服务器视为客户证书。 kube-controller-manager 不会自动批准它。
    1. 信任分发：签名的证书将被 API 服务器视为客户端证书。CA 证书包不通过任何其他方式分发。
    2. 许可的主体：没有主体限制，但审核人和签名者可以选择不批准或不签署。某些主体，比如集群管理员级别的用户或组因部署和安装方式不同而不同，所以批准和签署之前需要进行额外仔细审查。 用来限制 system:masters 的

CertificateSubjectRestriction 准入插件默认处于启用状态， 但它通常不是集群中唯一的集群管理员主体。
3. 许可的 x509 扩展：允许 subjectAltName 和 key usage 扩展，弃用其他扩展。
4. 许可的密钥用途：必须包含 ["client auth"]，但不能包含 ["digital signature", "key encipherment", "client auth"] 之外的键。
5. 过期时间/证书有效期：通过 kube-controller-manager 中 --cluster-signing-duration 标志来设置，由其中的签名者实施。
6. 允许/不允许 CA 位：不允许。

1. kubernetes.io/kube-apiserver-client-kubelet: 签名的证书将被 kube-apiserver 视为客户证书。 kube-controller-manager 可以自动批准它。

    1. 信任分发：签名的证书将被 API 服务器视为客户端证书。CA 证书包不通过任何其他方式分发。
    2. 许可的主体：组织名必须是 ["system:nodes"]，用户名以 "system:node:" 开头
    3. 许可的 x509 扩展：允许 key usage 扩展，禁用 subjectAltName 扩展，并删除其他扩展。
    4. 许可的密钥用途：必须是 ["key encipherment", "digital signature", "client auth"]
    5. 过期时间/证书有效期：通过 kube-controller-manager 中签名者的实现所对应的标志 --cluster-signing-duration 来设置。
    6. 允许/不允许 CA 位：不允许。

1. kubernetes.io/kubelet-serving: 签名服务证书，该服务证书被 API 服务器视为有效的 kubelet 服务证书， 但没有其他保证。kube-controller-manager 不会自动批准它。
    1. 信任分发：签名的证书必须被 kube-apiserver 认可，可有效的中止 kubelet 连接。CA 证书包不通过任何其他方式分发。
    2. 许可的主体：组织名必须是 ["system:nodes"]，用户名以 "system:node:" 开头
    3. 许可的 x509 扩展：允许 key usage、DNSName/IPAddress subjectAltName 等扩展， 禁止 EmailAddress、URI subjectAltName 等扩展，并丢弃其他扩展。 至少有一个 DNS 或 IP 的 SubjectAltName 存在。
    4. 许可的密钥用途：必须是 ["key encipherment", "digital signature", "client auth"]
    5. 过期日期/证书生命期：通过 kube-controller-manager 中签名者的实现所对应的标志 --cluster-signing-duration 来设置。
    6. 允许/不允许 CA 位：不允许。

1. kubernetes.io/legacy-unknown: 不保证信任。Kubernetes 的一些第三方发行版可能会使用它签署的客户端证书。 稳定版的 CertificateSigningRequest API（certificates.k8s.io/v1 以及之后的版本）不允许将 signerName 设置为 kubernetes.io/legacy-unknown。 kube-controller-manager 不会自动批准这类请求。
    1. 信任分发：没有。这个签名者在 Kubernetes 集群中没有标准的信任或分发。
    2. 许可的主体：全部。

3. 许可的 x509 扩展：允许 subjectAltName 和 key usage 等扩展，并弃用其他扩展。
4. 许可的密钥用途：全部。
5. 过期日期/证书生命期：通过 kube-controller-manager 中签名者的实现所对应的标志 --cluster-signing-duration 来设置。
6. 允许/不允许 CA 位 - 不允许。

说明：

注意：所有这些故障仅在 kube-controller-manager 日志中报告。

对于这些签名者，信任的分发发生在带外（out of band）。上述信任之外的任何信任都是完全巧合的。 例如，一些发行版可能会将 kubernetes.io/legacy-unknown 作为 kube-apiserver 的客户端证书， 但这个做法并不标准。 这些用途都没有以任何方式涉及到 ServiceAccount 中的 Secrets .data[ca.crt]。 此 CA 证书包只保证使用默认的服务（kubernetes.default.svc）来验证到 API 服务器的连接。

鉴权

授权创建 CertificateSigningRequest 和检索 CertificateSigningRequest:

- verbs（动词）：create、get、list、watch, group（组）：certificates.k8s.io，resources：certificatesigningrequests

例如：

access/certificate-signing-request/clusterrole-create.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csr-creator
rules:
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - create
  - get
  - list
  - watch
```

授权批准 CertificateSigningRequest：

- verbs（动词）：get、list、watch， group（组）：certificates.k8s.io，resources（资源）：certificatesigningrequests

- verbs（动词）：update， group（组）：certificates.k8s.io， resources（资源）：certificatesigningrequests/approval
- verbs（动词）：approve， group（组）：certificates.k8s.io， resources（资源）：signers， resourceName：<signerNameDomain>/<signerNamePath> 或 <signerNameDomain>/*

例如：

access/certificate-signing-request/clusterrole-approve.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csr-approver
rules:
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests/approval
  verbs:
  - update
- apiGroups:
  - certificates.k8s.io
  resources:
  - signers
  resourceNames:
  - example.com/my-signer-name # example.com/* can be used to authorize for all signers in the 'example.com' domain
  verbs:
  - approve
```

授权签名 CertificateSigningRequest：

- verbs（动词）：get、list、watch， group（组）：certificates.k8s.io，resources（资源）：certificatesigningrequests
- verbs（动词）：update， group（组）：certificates.k8s.io， resources（资源）：certificatesigningrequests/status

- verbs（动词）：sign， group（组）：certificates.k8s.io， resources（资源）：signers， resourceName：<signerNameDomain>/<signerNamePath> 或 <signerNameDomain>/*

access/certificate-signing-request/clusterrole-sign.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csr-signer
rules:
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests/status
  verbs:
  - update
- apiGroups:
  - certificates.k8s.io
  resources:
  - signers
  resourceName:
  - example.com/my-signer-name # example.com/* can be used to authorize for all signers in the 'example.com' domain
  verbs:
  - sign
```

普通用户

为了让普通用户能够通过认证并调用 API，需要执行几个步骤。 首先，该用户必须拥有 Kubernetes 集群签发的证书， 然后将该证书作为 API 调用的 Certificate 头或通过 kubectl 提供。

创建私钥

下面的脚本展示了如何生成 PKI 私钥和 CSR。 设置 CSR 的 CN 和 O 属性很重要。 CN 是用户名，O 是该用户归属的组。 你可以参考 RBAC 了解标准组的信息。

```
openssl genrsa -out john.key 2048
openssl req -new -key john.key -out john.csr
```

创建 CertificateSigningRequest

创建一个 CertificateSigningRequest，并通过 kubectl 将其提交到 Kubernetes 集群。 下面是生成 CertificateSigningRequest 的脚本。

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: john
spec:
  groups:
  - system:authenticated
  request:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ1ZqQ0NBVDRD
QVFBd0VURVBNQTBHQTFVRUF3d0dZVzVuWld4aE1JSUJJakFOQmdrcWhraUc5dz
BCQVFFRgpBQU9DQVE4QU1JSUJDZ0tDQVFFQTByczhJTHRHdTYxakx2dHhWTTJS
VlRWMDNHWlJTWWw0dWluVWo4RElaWjBOCnR2MUZtRVFSd3VoaUZsOFEzcWl0
Qm0wMUFSMkNJVXBGd2ZzSjZ4MXF3ckJzVkhZbGlBNVhwRVpZM3ExcGswSDQK
M3Z3aGJlK1o2MVNrVHF5SVBYUUwrTWM5T1Nsbm0xb0R2N0NtSkZNMUlMRVI3
QTVGZnZKOEdFRjJ6dHBoaUlFMwpub1dtdHNZb3JuT2wzc2lHQ2ZGZzR4Zmd4eW
8ybmlneFNVekl1bXNnVm9PM2ttT0x1RVF6cXpkakJ3TFJXbWlECklmMXBMWnoyal
Vnald4UkhCM1gyWnVVV1d1T09PZnpXM01LaE8ybHEvZi9DdS8wYk83c0x0MCt3U
2ZMSU91TFcKcW90blZtRmxMMytqTy82WDNDKzBERHk5aUtwbXJjVDBnWGZLem
E1dHJRSURBUUFCb0FBd0RRWUpLb1pJaHZjTgpBUUVMQlFBRGdnRUJBR05WdmV
IOGR4ZzNvK21VeVRkbmFjVmQ1N24zSkExdnZEU1JWREkyQTZ1eXN3ZFp1L1BVCk
kwZXpZWFV0RVNnSk1IRmQycVVNMjNuNVJsSXJ3R0xuUXFISUh5VStWWHhsdnZs
RnpNOVpEWllSTmU3QlJvYXgKQVlEdUI5STZXT3FYbkFvczFqRmxNUG5NbFpqdU5k
SGxpT1BjTU1oNndLaTZzZFhpVStHYTJ2RUVLY01jSVUyRgpvU2djUWdMYTk0aEpac
Gk3ZnNMdm1OQUxoT045UHdNMGM1dVJVejV4T0dGMUtCbWRSeEgvbUNOS2JK
YjFRQm1HCkkwYitEUEdaTktXTU0xMzhIQXdoV0tkNjVoVHdYOWl4V3ZHMkh4TG1
WQzg0L1BHT0tWQW9FNkpsYWFHdTlQVmkKdjlOSjVaZlZrcXdCd0hKbzZXdk9xVlA
3SVFjZmg3d0drWm89Ci0tLS0tRU5EIENFUlRJRklDQVRFIFJFUVVFU1QtLS0tLQo=
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - client auth
EOF
```

需要注意的几点:

- usage 字段必须是 'client auth'
- request 字段是 CSR 文件内容的 base64 编码值。 要得到该值，可以执行命令 cat john.csr | base64 | tr -d "\n"。

批准证书签名请求

使用 kubectl 创建 CSR 并批准。

获取 CSR 列表：

```
kubectl get csr
```

批准 CSR：

```
kubectl certificate approve john
```

取得证书

从 CSR 取得证书：

```
kubectl get csr/john -o yaml
```

证书的内容使用 base64 编码，存放在字段 status.certificate。

创建角色和角色绑定

创建了证书之后，为了让这个用户能访问 Kubernetes 集群资源，现在就要创建 Role 和 RoleBinding 了。

下面是为这个新用户创建 Role 的示例脚本：

```
kubectl create role developer --verb=create --verb=get --verb=list --
verb=update --verb=delete --resource=pods
```

下面是为这个新用户创建 RoleBinding 的示例命令：

```
kubectl create rolebinding developer-binding-john --role=developer --user=john
```

添加到 kubeconfig

最后一步是将这个用户添加到 kubeconfig 文件。 我们假设私钥和证书文件存放在 "/home/vagrant/work/" 目录中。

首先，我们需要添加新的凭据：

```
kubectl config set-credentials john --client-key=/home/vagrant/work/john.key --
client-certificate=/home/vagrant/work/john.crt --embed-certs=true
```

然后，你需要添加上下文：

```
kubectl config set-context john --cluster=kubernetes --user=john
```

来测试一下，把上下文切换为 john：

```
kubectl config use-context john
```

批准和驳回

控制平面的自动化批准

kube-controller-manager 内建了一个证书批准者，其 signerName 为 kubernetes.io/kube-apiserver-client-kubelet， 该批准者将 CSR 上用于节点凭据的各种权限委托给权威认证机构。 kube-controller-manager 将 SubjectAccessReview 资源发送（POST）到 API 服务器， 以便检验批准证书的授权。

使用 kubectl 批准或驳回

Kubernetes 管理员（拥有足够的权限）可以手工批准（或驳回） CertificateSigningRequests， 此操作使用 kubectl certificate approve 和 kubectl certificate deny 命令实现。

使用 kubectl 批准一个 CSR：

```
kubectl certificate approve <certificate-signing-request-name>
```

同样地，驳回一个 CSR：

```
kubectl certificate deny <certificate-signing-request-name>
```

使用 Kubernetes API 批准或驳回

REST API 的用户可以通过向待批准的 CSR 的 approval 子资源提交更新请求来批准 CSR。 例如，你可以编写一个 operator 来监视特定类型的 CSR，然后发送一个更新来批准它。

当你发出批准或驳回的指令时，根据你期望的状态来选择设置 Approved 或 Denied。

批准（Approved） 的 CSR：

```yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
...
status:
  conditions:
  - lastUpdateTime: "2020-02-08T11:37:35Z"
    lastTransitionTime: "2020-02-08T11:37:35Z"
    message: Approved by my custom approver controller
    reason: ApprovedByMyPolicy # You can set this to any string
    type: Approved
```

驳回（Denied）的 CRS：

```yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
...
status:
  conditions:
  - lastUpdateTime: "2020-02-08T11:37:35Z"
    lastTransitionTime: "2020-02-08T11:37:35Z"
    message: Denied by my custom approver controller
    reason: DeniedByMyPolicy # You can set this to any string
    type: Denied
```

status.conditions.reason 字段通常设置为一个首字母大写的对机器友好的原因码; 这是一个命名约定，但你也可以随你的个人喜好设置。 如果你想添加一个仅供人类使用的注释，那就用 status.conditions.message 字段。

签名

控制平面签名者

Kubernetes 控制平面实现了每一个 Kubernetes 签名者， 每个签名者的实现都是 kube-controller-manager 的一部分。

> **说明：** 在Kubernetes v1.18 之前， kube-controller-manager 签名所有标记为 approved 的 CSR。

基于 API 的签名者

REST API 的用户可以通过向待签名的 CSR 的 status 子资源提交更新请求来对 CSR 进行签名。

作为这个请求的一部分， status.certificate 字段应设置为已签名的证书。 此字段可包含一个或多个 PEM 编码的证书。

所有的 PEM 块必须具备 "CERTIFICATE" 标签，且不包含文件头，且编码的数据必须是 RFC5280 第 4 节 中描述的 BER 编码的 ASN.1 证书结构。

```
-----BEGIN CERTIFICATE-----
MIIDgjCCAmqgAwIBAgIUC1N1EJ4Qnsd322BhDPRwmg3b/
oAwDQYJKoZIhvcNAQEL
BQAwXDELMAkGA1UEBhMCeHgxCjAIBgNVBAgMAXgxCjAIBgNVBAcMAXgxCjAIB
gNV
BAoMAXgxCjAIBgNVBAsMAXgxCzAJBgNVBAMMAmNhMRAwDgYJKoZIhvcNAQk
BFgF4
MB4XDTIwMDcwNjIyMDcwMFoXDTI1MDcwNTIyMDcwMFowNzEVMBMGA1UECh
MMc3lz
dGVtOm5vZGVzMR4wHAYDVQQDExVzeXN0ZW06bm9kZToxMjcuMC4wLjEwggEi
MA0G
```

```
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDne5X2eQ1JcLZkKvhzCR4Hxl9+ZmU3
+e1zfOywLdoQxrPi+o4hVsUH3q0y52BMa7u1yehHDRSaq9u62cmi5ekgXhXHzGmm
kmW5n0itRECv3SFsSm2DSghRKf0mm6iTYHWDHzUXKdm9lPPWoSOxoR5oqOsm3JEh
Q7Et13wrvTJqBMJo1GTwQuF+HYOku0NF/DLqbZIcpI08yQKyrBgYz2uO51/oNp8a
sTCsV4OUfyHhx2BBLUo4g4SptHFySTBwlpRWBnSjZPOhmN74JcpTLB4J5f4iEeA7
2QytZfADckG4wVkhH3C2EJUmRtFIBVirwDn39GXkSGlnvnMgF3uLZ6zNAgMBAAGj
YTBfMA4GA1UdDwEB/
wQEAwIFoDATBgNVHSUEDDAKBggrBgEFBQcDAjAMBgNVHRMB
Af8EAjAAMB0GA1UdDgQWBBTREl2hW54lkQBDeVCcd2f2VSlB1DALBgNVHREEBDAC
ggAwDQYJKoZIhvcNAQELBQADggEBABpZjuIKTq8pCaX8dMEGPWtAykgLsTcD2jYr
L0/TCrqmuaaliUa42jQTt2OVsVP/L8ofFunj/KjpQU0bvKJPLMRKtmxbhXuQCQi1
qCRkp8o93mHvEz3mTUN+D1cfQ2fpsBENLnpS0F4G/JyY2Vrh19/
X8+mImMEK5eOy
o0BMby7byUj98WmcUvNCiXbC6F45QTmkwEhMqWns0JZQY+/
XeDhEcg+lJvz9Eyo2
aGgPsye1o3DpyXnyfJWAWMhOz7cikS5X2adesbgI86PhEHBXPIJ1v13ZdfCExmdd
M1fLPhLyR54fGaY+7/X8P9AZzPefAkwizeXwe9ii6/a08vWoiE4=
-----END CERTIFICATE-----
```

非 PEM 内容可能会出现在证书 PEM 块前后的位置，且未经验证， 以允许使用 RFC7468 第5.2节 中描述的解释性文本。

当使用 JSON 或 YAML 格式时，此字段是 base-64 编码。 包含上述示例证书的 CertificateSigningRequest 如下所示：

```yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
...
status:
  certificate: "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JS..."
```

接下来

- 参阅 管理集群中的 TLS 认证
- 查看 kube-controller-manager 中签名者部分的源代码
- 查看 kube-controller-manager 中批准者部分的源代码
- 有关 X.509 本身的详细信息，请参阅 RFC 5280 第3.1节
- 有关 PKCS#10 证书签名请求语法的信息，请参阅 RFC 2986

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 28, 2020 at 5:07 PM PST: [zh] Sync changes from English (auth references) (bab256b78)

使用准入控制器

此页面概述了准入控制器。

什么是准入控制插件？

准入控制器是一段代码，它会在请求通过认证和授权之后、对象被持久化之前拦截到达 API 服务器的请求。控制器由下面的列表组成， 并编译进 kube-apiserver 二进制文件，并且只能由集群管理员配置。 在该列表中，有两个特殊的控制器：MutatingAdmissionWebhook 和 ValidatingAdmissionWebhook。 它们根据 API 中的配置，分别执行变更和验证 准入控制 webhook。

准入控制器可以执行 "验证（Validating）" 和/或 "变更（Mutating）" 操作。 变更（mutating）控制器可以修改被其接受的对象；验证（validating）控制器则不行。

准入控制过程分为两个阶段。第一阶段，运行变更准入控制器。第二阶段，运行验证准入控制器。 再次提醒，某些控制器既是变更准入控制器又是验证准入控制器。

如果任何一个阶段的任何控制器拒绝了该请求，则整个请求将立即被拒绝，并向终端用户返回一个错误。

最后，除了对对象进行变更外，准入控制器还可以有其它作用：将相关资源作为请求处理的一部分进行变更。 增加使用配额就是一个典型的示例，说明了这样做的必要性。 此类用法都需要相应的回收或回调过程，因为任一准入控制器都无法确定某个请求能否通过所有其它准入控制器。

为什么需要准入控制器？

Kubernetes 的许多高级功能都要求启用一个准入控制器，以便正确地支持该特性。 因此，没有正确配置准入控制器的 Kubernetes API 服务器是不完整的，它无法支持你期望的所有特性。

如何启用一个准入控制器？

Kubernetes API 服务器的 enable-admission-plugins 标志接受一个用于在集群修改对象之前 调用的（以逗号分隔的）准入控制插件顺序列表。

例如，下面的命令就启用了 NamespaceLifecycle 和 LimitRanger 准入控制插件：

```
kube-apiserver --enable-admission-plugins=NamespaceLifecycle,LimitRanger ...
```

说明：

根据你 Kubernetes 集群的部署方式以及 API 服务器的启动方式的不同，你可能需要以不同的方式应用设置。 例如，如果将 API 服务器部署为 systemd 服务，你可能需要修改 systemd 单元文件； 如果以自托管方式部署 Kubernetes，你可能需要修改 API 服务器的清单文件。

怎么关闭准入控制器？

Kubernetes API 服务器的 disable-admission-plugins 标志，会将传入的（以逗号分隔的）准入控制插件列表禁用，即使是默认启用的插件也会被禁用。

```
kube-apiserver --disable-admission-plugins=PodNodeSelector,AlwaysDeny ...
```

哪些插件是默认启用的？

下面的命令可以查看哪些插件是默认启用的：

```
kube-apiserver -h | grep enable-admission-plugins
```

在目前版本中，它们是：

```
NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition,
Priority, DefaultTolerationSeconds, DefaultStorageClass,
StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass,
CertificateApproval, CertificateSigning, CertificateSubjectRestriction,
DefaultIngressClass, MutatingAdmissionWebhook,
ValidatingAdmissionWebhook, ResourceQuota
```

每个准入控制器的作用是什么？

AlwaysAdmit

FEATURE STATE: Kubernetes v1.13 [deprecated]

该准入控制器会允许所有的 pod 接入集群。已废弃，因为它的行为根本就和没有准入控制器一样。

AlwaysPullImages

该准入控制器会修改每一个新创建的 Pod 的镜像拉取策略为 Always 。 这在多租户集群中是有用的，这样用户就可以放心，他们的私有镜像只能被那些有凭证的人使用。 如果没有这个准入控制器，一旦镜像被拉取到节点上，任何用户的 Pod 都可以通过已了解到的镜像 的名称（假设 Pod 被调度到正确的节点上）来使用它，而不需要对镜像进行任何授权检查。 当启用这个准入控制器时，总是在启动容器之前拉取镜像，这意味着需要有效的凭证。

AlwaysDeny

FEATURE STATE: Kubernetes v1.13 [deprecated]

拒绝所有的请求。由于没有实际意义，已废弃。

CertificateApproval

此准入控制器获取"审批" CertificateSigningRequest 资源的请求并执行额外的授权检查， 以确保审批请求的用户有权限审批 spec.signerName 请求 CertificateSigningRequest 资源的证书请求。

有关对证书签名请求资源执行不同操作所需权限的详细信息， 请参阅证书签名请求

CertificateSigning

此准入控制器获取 CertificateSigningRequest 资源的 status.certificate 字段更新请求并执行额外的授权检查， 以确保签发证书的用户有权限为 spec.signerName 请求 CertificateSigningRequest 资源的证书请求签发证书。

有关对证书签名请求资源执行不同操作所需权限的详细信息， 请参阅证书签名请求

CertificateSubjectRestrictions

此准入控制器获取具有 kubernetes.io/kube-apiserver-client 的 spec.signerName 的 CertificateSigningRequest 资源创建请求， 它拒绝任何包含了 system:masters 一个"组"（或者"组织"）的请求。

DefaultStorageClass

该准入控制器监测没有请求任何特定存储类的 PersistentVolumeClaim 对象的创建， 并自动向其添加默认存储类。 这样，没有任何特殊存储类需求的用户根本不需要关心它们，它们将获得默认存储类。

当未配置默认存储类时，此准入控制器不执行任何操作。如果将多个存储类标记为默认存储类， 它将拒绝任何创建 PersistentVolumeClaim 的操作，并显示错误。 要修复此错误，管理员必须重新访问其 StorageClass 对象，并仅将其中一个标记为默认。 此准入控制器会忽略所有 PersistentVolumeClaim 更新操作，仅响应创建操作。

关于持久化卷和存储类，以及如何将存储类标记为默认，请参见 持久化卷。

DefaultTolerationSeconds

该准入控制器为 Pod 设置默认的容忍度，在 5 分钟内容忍 notready:NoExecute 和 unreachable:NoExecute 污点。 （如果 Pod 尚未容忍 node.kubernetes.io/not-ready：NoExecute 和 node.kubernetes.io/unreachable：NoExecute 污点的话）

DenyExecOnPrivileged

FEATURE STATE: Kubernetes v1.13 [deprecated]

如果一个 pod 拥有一个特权容器，该准入控制器将拦截所有在该 pod 中执行 exec 命令的请求。

此功能已合并至 DenyEscalatingExec。 而 DenyExecOnPrivileged 准入插件已被废弃，并将在 v1.18 被移除。

建议使用基于策略的准入插件（例如 PodSecurityPolicy 和自定义准入插件），该插件可以针对特定用户或名字空间，还可以防止创建权限过高的 Pod。

DenyEscalatingExec

FEATURE STATE: Kubernetes v1.13 [deprecated]

该准入控制器将拒绝在由于拥有升级特权，而具备访问宿主机能力的 Pod 中执行 exec 和 attach 命令。这包括在特权模式运行的 Pod，可以访问主机 IPC 名字空间的 Pod，和访问主机 PID 名字空间的 Pod 。

DenyExecOnPrivileged 准入插件已被废弃，并将在 v1.18 被移除。

建议使用基于策略的准入插件（例如 PodSecurityPolicy 和自定义准入插件），该插件可以针对特定用户或名字空间，还可以防止创建权限过高的 Pod。

EventRateLimit

FEATURE STATE: Kubernetes v1.13 [alpha]

该准入控制器缓解了事件请求淹没 API 服务器的问题。集群管理员可以通过以下方式指定事件速率限制：

- 启用 EventRateLimit 准入控制器；
- 从文件中引用 EventRateLimit 配置文件，并提供给 API 服务器命令的 `--admission-control-config-file` 标志：

- apiserver.config.k8s.io/v1
- apiserver.k8s.io/v1alpha1

```yaml
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: EventRateLimit
  path: eventconfig.yaml
...
```

```yaml
# Deprecated in v1.17 in favor of apiserver.config.k8s.io/v1
apiVersion: apiserver.k8s.io/v1alpha1
kind: AdmissionConfiguration
```

```yaml
plugins:
- name: EventRateLimit
  path: eventconfig.yaml
...
```

可以在配置中指定四种类型的限制：

- Server: API 服务器收到的所有事件请求共享一个桶。
- Namespace: 每个名字空间都有一个专用的桶。
- User: 给每个用户都分配一个桶。
- SourceAndObject: 根据事件的源和涉及对象的每种组合分配桶。

下面是一个配置示例 eventconfig.yaml：

```yaml
apiVersion: eventratelimit.admission.k8s.io/v1alpha1
kind: Configuration
limits:
- type: Namespace
  qps: 50
  burst: 100
  cacheSize: 2000
- type: User
  qps: 10
  burst: 50
```

详情请参见 事件速率限制提案。

ExtendedResourceToleration

该插件有助于创建可扩展资源的专用节点。 如果运营商想创建可扩展资源的专用节点（如 GPU、FPGA 等），那他们应该以扩展资源名称作为键名， 为节点设置污点。 如果启用了该准入控制器，会将此类污点的容忍自动添加到请求扩展资源的 Pod 中， 用户不必再手动添加这些容忍。

ImagePolicyWebhook

ImagePolicyWebhook 准入控制器允许使用一个后端的 webhook 做出准入决策。

配置文件格式

ImagePolicyWebhook 使用配置文件来为后端行为设置配置选项。该文件可以是 JSON 或 YAML， 并具有以下格式:

```yaml
imagePolicy:
  kubeConfigFile: /path/to/kubeconfig/for/backend
  # 以秒计的时长，控制批准请求的缓存时间
  allowTTL: 50
  # 以秒计的时长，控制批准请求的缓存时间
```

```
  denyTTL: 50
  # 以毫秒计的时长，控制重试间隔
  retryBackoff: 500
  # 确定 Webhook 后端失效时的行为
  defaultAllow: true
```

从文件中引用 ImagePolicyWebhook 的配置文件，并将其提供给 API 服务器命令标志 --admission-control-config-file：

- apiserver.config.k8s.io/v1
- apiserver.k8s.io/v1alpha1

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: ImagePolicyWebhook
  path: imagepolicyconfig.yaml
...
```

```
# v1.17 中已废弃以鼓励使用 apiserver.config.k8s.io/v1
apiVersion: apiserver.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
- name: ImagePolicyWebhook
  path: imagepolicyconfig.yaml
...
```

或者，你也可以直接将配置嵌入到文件中：

- apiserver.config.k8s.io/v1
- apiserver.k8s.io/v1alpha1

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: ImagePolicyWebhook
  configuration:
    imagePolicy:
      kubeConfigFile: <kubeconfig 文件路径>
      allowTTL: 50
      denyTTL: 50
      retryBackoff: 500
      defaultAllow: true
```

```
# v1.17 中已废弃以鼓励使用 apiserver.config.k8s.io/v1
apiVersion: apiserver.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
```

```yaml
- name: ImagePolicyWebhook
  configuration:
    imagePolicy:
      kubeConfigFile: <kubeconfig 文件路径>
      allowTTL: 50
      denyTTL: 50
      retryBackoff: 500
      defaultAllow: true
```

ImagePolicyWebhook 的配置文件必须引用 kubeconfig 格式的文件；该文件设置了到后端的连接参数。 要求后端使用 TLS 进行通信。

kubeconfig 文件的 cluster 字段需要指向远端服务，user 字段需要包含已返回的授权者。

```yaml
# clusters 指的是远程服务。
clusters:
- name: name-of-remote-imagepolicy-service
  cluster:
    certificate-authority: /path/to/ca.pem    # CA 用于验证远程服务
    server: https://images.example.com/policy # 要查询的远程服务的 URL。必须是 'https' 。

# users 指的是 API 服务器的 Webhook 配置。
users:
- name: name-of-api-server
  user:
    client-certificate: /path/to/cert.pem # webhook 准入控制器使用的证书
    client-key: /path/to/key.pem          # 证书匹配的密钥
```

关于 HTTP 配置的更多信息，请参阅 kubeconfig 文档。

请求载荷

当面对一个准入决策时，API 服务器发送一个描述操作的 JSON 序列化的 imagepolicy.k8s.io/v1alpha1 ImageReview 对象。 该对象包含描述被审核容器的字段，以及所有匹配 *.image-policy.k8s.io/* 的 Pod 注解。

注意，Webhook API 对象与其他 Kubernetes API 对象一样受制于相同的版本控制兼容性规则。 实现者应该知道对 alpha 对象的更宽松的兼容性，并检查请求的 "apiVersion" 字段， 以确保正确的反序列化。 此外，API 服务器必须启用 imagepolicy.k8s.io/v1alpha1 API 扩展组（ --runtime-config=imagepolicy.k8s.io/v1alpha1=true ）。

请求载荷示例：

```
{
  "apiVersion":"imagepolicy.k8s.io/v1alpha1",
```

```
  "kind":"ImageReview",
  "spec":{
    "containers":[
      {
        "image":"myrepo/myimage:v1"
      },
      {
        "image":"myrepo/
myimage@sha256:beb6bd6a68f114c1dc2ea4b28db81bdf91de202a9014972bec5
e4d9171d90ed"
      }
    ],
    "annotations":{
      "mycluster.image-policy.k8s.io/ticket-1234": "break-glass"
    },
    "namespace":"mynamespace"
  }
}
```

远程服务将填充请求的 ImageReviewStatus 字段，并返回允许或不允许访问的响应。 响应体的 "spec" 字段会被忽略，并且可以省略。一个允许访问应答会返回：

```
{
  "apiVersion": "imagepolicy.k8s.io/v1alpha1",
  "kind": "ImageReview",
  "status": {
    "allowed": true
  }
}
```

若不允许访问，服务将返回：

```
{
  "apiVersion": "imagepolicy.k8s.io/v1alpha1",
  "kind": "ImageReview",
  "status": {
    "allowed": false,
    "reason": "image currently blacklisted"
  }
}
```

更多的文档，请参阅 imagepolicy.v1alpha1 API 对象和 plugin/pkg/admission/imagepolicy/admission.go。

使用注解进行扩展

一个 Pod 中匹配 *.image-policy.k8s.io/* 的注解都会被发送给 Webhook。 这样做使得了解后端镜像策略的用户可以向它发送额外的信息，并为不同的后端实现 接收不同的信息。

你可以在这里输入的信息有：

- 在紧急情况下，请求 "break glass" 覆盖一个策略。
- 从一个记录了 break-glass 的请求的 ticket 系统得到的一个 ticket 号码。
- 向策略服务器提供一个提示，用于提供镜像的 imageID，以方便它进行查找。

在任何情况下，注解都是由用户提供的，并不会被 Kubernetes 以任何方式进行验证。在将来，如果一个注解确定将被广泛使用，它可能会被提升为 ImageReviewSpec 的一个命名字段。

LimitPodHardAntiAffinityTopology

该准入控制器拒绝（定义了 AntiAffinity 拓扑键的）任何 Pod （requiredDuringSchedulingRequiredDuringExecution 中的 kubernetes.io/hostname 除外）。

LimitRanger

该准入控制器会观察传入的请求，并确保它不会违反 Namespace 中 LimitRange 对象枚举的任何约束。 如果你在 Kubernetes 部署中使用了 LimitRange 对象，则必须使用此准入控制器来 执行这些约束。 LimitRanger 还可以用于将默认资源请求应用到没有指定任何内容的 Pod； 当前，默认的 LimitRanger 对 default 名字空间中的所有 Pod 都应用了 0.1 CPU 的需求。

请查看 limitRange 设计文档 和 LimitRange 例子 以了解更多细节。

MutatingAdmissionWebhook

FEATURE STATE: Kubernetes v1.13 [beta]

该准入控制器调用任何与请求匹配的变更 Webhook。匹配的 Webhook 将被串行调用。每一个 Webhook 都可以根据需要修改对象。

MutatingAdmissionWebhook，顾名思义，仅在变更阶段运行。

如果由此准入控制器调用的 Webhook 有副作用（如降低配额）， 则它 *必须* 具有协调系统，因为不能保证后续的 Webhook 和验证准入控制器都会允许完成请求。

如果你禁用了 MutatingAdmissionWebhook，那么还必须使用 --runtime-config 标志禁止 admissionregistration.k8s.io/v1beta1 组/版本中的 MutatingWebhookConfiguration 对象（版本 >=1.9 时，这两个对象都是默认启用的）。

谨慎编写和安装变更 webhook

- 当用户尝试创建的对象与返回的对象不同时，用户可能会感到困惑。

- 当它们回读的对象与尝试创建的对象不同，内建的控制环可能会出问题。
  - 与覆盖原始请求中设置的字段相比，使用原始请求未设置的字段会引起问题的可能性较小。 应尽量避免前面那种方式。
- 这是一个 beta 特性。Kubernetes 未来的版本可能会限制这些 Webhook 可以进行的变更类型。
- 内建资源和第三方资源的控制回路未来可能会受到破坏性的更改，使现在运行良好的 Webhook 无法再正常运行。即使完成了 Webhook API 安装，也不代表会为该 webhook 提供无限期的支持。

NamespaceAutoProvision

该准入控制器会检查名字空间资源上的所有传入请求，并检查所引用的名字空间是否确实存在。 如果找不到，它将创建一个名字空间。 此准入控制器对于不想要求名字空间必须先创建后使用的集群部署中很有用。

NamespaceExists

该准入控制器检查除 Namespace 以外的名字空间作用域资源上的所有请求。 如果请求引用的名字空间不存在，则拒绝该请求。

NamespaceLifecycle

该准入控制器禁止在一个正在被终止的 Namespace 中创建新对象，并确保 使用不存在的 Namespace 的请求被拒绝。 该准入控制器还会禁止删除三个系统保留的名字空间，即 default、 kube-system 和 kube-public。

删除 Namespace 会触发删除该名字空间中所有对象（Pod、Service 等）的一系列操作。 为了确保这个过程的完整性，我们强烈建议启用这个准入控制器。

NodeRestriction

该准入控制器限制了 kubelet 可以修改的 Node 和 Pod 对象。 为了受到这个准入控制器的限制，kubelet 必须使用在 system:nodes 组中的凭证， 并使用 system:node:<nodeName> 形式的用户名。 这样，kubelet 只可修改自己的 Node API 对象，只能修改绑定到节点本身的 Pod 对象。

在 Kubernetes 1.11+ 的版本中，不允许 kubelet 从 Node API 对象中更新或删除污点。

在 Kubernetes 1.13+ 的版本中，NodeRestriction 准入插件可防止 kubelet 删除 Node API 对象，并对 kubernetes.io/ 或 k8s.io/ 前缀标签的 kubelet 强制进行如下修改：

- **防止** kubelet 添加/删除/更新带有 node-restriction.kubernetes.io/ 前缀的标签。保留此前缀的标签，供管理员用来标记 Node 对象以隔离工作负载，并且不允许 kubelet 修改带有该前缀的标签。
- **允许** kubelet 添加/删除/更新这些和这些前缀的标签：
  - kubernetes.io/hostname
  - kubernetes.io/arch

- kubernetes.io/os
- beta.kubernetes.io/instance-type
- node.kubernetes.io/instance-type
- failure-domain.beta.kubernetes.io/region （已弃用）
- failure-domain.beta.kubernetes.io/zone（已弃用）
- topology.kubernetes.io/region
- topology.kubernetes.io/zone
- kubelet.kubernetes.io/-prefixed labels
- node.kubernetes.io/-prefixed labels

kubelet 保留 kubernetes.io 或 k8s.io 前缀的所有标签，并且将来可能会被 NodeRestriction 准入插件允许或禁止。

将来的版本可能会增加其他限制，以确保 kubelet 具有正常运行所需的最小权限集。

OwnerReferencesPermissionEnforcement

该准入控制器保护对 metadata.ownerReferences 对象的访问，以便只有对该对象具有 "删除" 权限的用户才能对其进行更改。 该准入控制器还保护对 metadata.ownerReferences[x].blockOwnerDeletion 对象的访问， 以便只有对所引用的 **属主（owner）** 的 finalizers 子资源具有 "更新" 权限的用户才能对其进行更改。

PersistentVolumeLabel

FEATURE STATE: Kubernetes v1.13 [deprecated]

该准入控制器会自动将区（region）或区域（zone）标签附加到由云提供商（如 GCE、AWS）定义的 PersistentVolume。这有助于确保 Pod 和 PersistentVolume 位于相同的区或区域。 如果准入控制器不支持为 PersistentVolumes 自动添加标签，那你可能需要手动添加标签， 以防止 Pod 挂载其他区域的卷。
PersistentVolumeLabel 已被废弃，标记持久卷已由 云管理控制器接管。 从 1.11 开始，默认情况下禁用此准入控制器。

PodNodeSelector

该准入控制器通过读取名字空间注解和全局配置，来为名字空间中可以可以使用的节点选择器 设置默认值并实施限制。

配置文件格式

PodNodeSelector 使用配置文件来设置后端行为的选项。 请注意，配置文件格式将在将来某个版本中改为版本化文件。 该文件可以是 JSON 或 YAML，格式如下：

```
podNodeSelectorPluginConfig:
 clusterDefaultNodeSelector: name-of-node-selector
 namespace1: name-of-node-selector
 namespace2: name-of-node-selector
```

基于提供给 API 服务器命令行标志 --admission-control-config-file 的文件名， 从文件中引用 PodNodeSelector 配置文件：

- apiserver.config.k8s.io/v1
- apiserver.k8s.io/v1alpha1

```yaml
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: PodNodeSelector
  path: podnodeselector.yaml
...
```

```yaml
# 在 v1.17 中废弃，以鼓励使用 apiserver.config.k8s.io/v1
apiVersion: apiserver.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
- name: PodNodeSelector
  path: podnodeselector.yaml
...
```

配置注解格式

PodNodeSelector 使用键为 scheduler.alpha.kubernetes.io/node-selector 的注解 为名字空间设置节点选择算符。

```yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/node-selector: name-of-node-selector
  name: namespace3
```

内部行为

该准入控制器行为如下：

1. 如果 Namespace 的注解带有键 scheduler.alpha.kubernetes.io/node-selector，则将其值用作节点选择算符。
2. 如果名字空间缺少此类注解，则使用 PodNodeSelector 插件配置文件中定义的 clusterDefaultNodeSelector 作为节点选择算符。
3. 评估 Pod 节点选择算符和名字空间节点选择算符是否存在冲突。存在冲突将导致拒绝。
4. 评估 pod 节点选择算符和名字空间的白名单定义的插件配置文件是否存在冲突。存在冲突将导致拒绝。

   说明：

PodNodeSelector 允许 Pod 强制在特定标签的节点上运行。 另请参阅 PodTolerationRestriction 准入插件，该插件可防止 Pod 在特定污点的节点上运行。

PersistentVolumeClaimResize

该准入控制器检查传入的 PersistentVolumeClaim 调整大小请求，对其执行额外的验证操作。

说明：

对调整卷大小的支持是一种 Alpha 特性。管理员必须将特性门控 ExpandPersistentVolumes 设置为 true 才能启用调整大小。

启用 ExpandPersistentVolumes 特性门控之后，建议将 PersistentVolumeClaimResize 准入控制器也启用。除非 PVC 的 StorageClass 明确地将 allowVolumeExpansion 设置为 true 来显式启用调整大小。否则，默认情况下该准入控制器会阻止所有对 PVC 大小的调整。

例如：由以下 StorageClass 创建的所有 PersistentVolumeClaim 都支持卷容量扩充：

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gluster-vol-default
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://192.168.10.100:8080"
  restuser: ""
  secretNamespace: ""
  secretName: ""
allowVolumeExpansion: true
```

关于持久化卷申领的更多信息，请参见 PersistentVolumeClaims。

PodSecurityPolicy

此准入控制器负责在创建和修改 Pod 时根据请求的安全上下文和可用的 Pod 安全策略确定是否可以执行请求。

查看 Pod 安全策略文档 了解更多细节。

PodTolerationRestriction

准入控制器 PodTolerationRestriction 检查 Pod 的容忍度与其名字空间的容忍度之间是否存在冲突。如果存在冲突，则拒绝 Pod 请求。 然后，它将名字空间的容忍度合并到 Pod 的容忍度中，之后根据名字空间的容忍度 白名单检查所得到的容忍度结果。如果检查成功，则将接受 Pod 请求，否则拒绝该请求。

如果 Pod 的名字空间没有任何关联的默认容忍度或容忍度白名单，则使用集群级别的 默认容忍度或容忍度白名单（如果有的话）。

名字空间的容忍度通过注解健 scheduler.alpha.kubernetes.io/defaultTolerations 来设置。可接受的容忍度可以通过 scheduler.alpha.kubernetes.io/tolerationsWhitelist 注解键来添加。

名字空间注解的示例：

```
apiVersion: v1
kind: Namespace
metadata:
  name: apps-that-need-nodes-exclusively
  annotations:
    scheduler.alpha.kubernetes.io/defaultTolerations: '[{"operator": "Exists",
"effect": "NoSchedule", "key": "dedicated-node"}]'
    scheduler.alpha.kubernetes.io/tolerationsWhitelist: '[{"operator": "Exists",
"effect": "NoSchedule", "key": "dedicated-node"}]'
```

优先级

优先级准入控制器使用 priorityClassName 字段并用整型值填充优先级。 如果找不到优先级，则拒绝 Pod。

ResourceQuota

该准入控制器会监测传入的请求，并确保它不违反任何一个 Namespace 中的 Resource Quota 对象中枚举出来的约束。 如果你在 Kubernetes 部署中使用了 ResourceQuota ，你必须使用这个准入控制器来强制 执行配额限制。

请查看 resourceQuota 设计文档和 Resource Quota 例子 了解更多细节。

RuntimeClass

+

FEATURE STATE: Kubernetes v1.20 [stable]

如果你开启 PodOverhead 特性门控, 并且通过 Pod 开销 配置来定义一个 RuntimeClass，这个准入控制器会检查新的 Pod。 当启用的时候，这个准入控制器会拒绝任何 overhead 字段已经设置的 Pod。 对于配置了 RuntimeClass 并在其 .spec 中选定 RuntimeClass 的 Pod， 此准入控制器会根据相应 RuntimeClass 中定义的值为 Pod 设置 .spec.overhead。

> **说明：** Pod 的 .spec.overhead 字段和 RuntimeClass 的 .overhead 字段均为处于 beta 版本。 如果你未启用 PodOverhead 特性门控，则所有 Pod 均被视为未设置 .spec.overhead。

详情请参见 Pod 开销。

SecurityContextDeny

该准入控制器将拒绝任何试图设置特定提升 SecurityContext 字段的 Pod，正如任务为 Pod 或 Container 配置安全上下文 中所展示的那样。 如果集群没有使用 Pod 安全策略 来限制安全上下文所能获取的值集，那么应该启用这个功能。

ServiceAccount

此准入控制器实现了 ServiceAccount 的自动化。 如果你打算使用 Kubernetes 的 ServiceAccount 对象，我们强烈建议你使用这个准入控制器。

StorageObjectInUseProtection

StorageObjectInUseProtection 插件将 kubernetes.io/pvc-protection 或 kubernetes.io/pv-protection finalizers 添加到新创建的持久化卷声明（PVC） 或持久化卷（PV）中。 如果用户尝试删除 PVC/PV，除非 PVC/PV 的保护控制器移除 finalizers，否则 PVC/PV 不会被删除。 有关更多详细信息，请参考 保护使用中的存储对象。

TaintNodesByCondition

FEATURE STATE: Kubernetes v1.12 [beta]

该准入控制器为新创建的节点添加 NotReady 和 NoSchedule 污点。 这些污点能够避免一些竞态条件的发生，这类静态条件可能导致 Pod 在更新节点污点以准确 反映其所报告状况之前，就被调度到新节点上。

ValidatingAdmissionWebhook

FEATURE STATE: Kubernetes v1.13 [beta]

该准入控制器调用与请求匹配的所有验证 Webhook。 匹配的 Webhook 将被并行调用。如果其中任何一个拒绝请求，则整个请求将失败。 该准入控制器仅在验证（Validating）阶段运行；与 MutatingAdmissionWebhook 准入控制器 所调用的 Webhook 相反，它调用的 Webhook 应该不会使对象出现变更。

如果以此方式调用的 Webhook 有其它作用（如，降低配额），则它必须具有协调机制。 这是因为无法保证后续的 Webhook 或其他有效的准入控制器都允许请求完成。

如果你禁用了 ValidatingAdmissionWebhook，还必须通过 --runtime-config 标志来禁用 admissionregistration.k8s.io/v1beta1 组/版本中的 ValidatingWebhookConfiguration 对象（默认情况下在 1.9 版和更高版本中均处于启用状态）。

有推荐的准入控制器吗？

有。推荐使用的准入控制器默认情况下都处于启用状态 （请查看这里）。 因此，你无需显式指定它们。 你可以使用 --enable-admission-plugins 标志（ **顺序不重要** ）来启用默认设置以外的其他准入控制器。

说明：

`--admission-control` 在 1.10 中已废弃，由 `--enable-admission-plugins` 取代。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 18, 2021 at 10:22 AM PST: [zh] Clean PodPreset related pages/examples (b9265d377)

动态准入控制

除了内置的 admission 插件， 准入插件可以作为扩展独立开发，并以运行时所配置的 Webhook 的形式运行。 此页面描述了如何构建、配置、使用和监视准入 Webhook。

什么是准入 Webhook？

准入 Webhook 是一种用于接收准入请求并对其进行处理的 HTTP 回调机制。 可以定义两种类型的准入 webhook，即 验证性质的准入 Webhook 和 修改性质的准入 Webhook。 修改性质的准入 Webhook 会先被调用。它们可以更改发送到 API 服务器的对象以执行自定义的设置默认值操作。

在完成了所有对象修改并且 API 服务器也验证了所传入的对象之后， 验证性质的 Webhook 会被调用，并通过拒绝请求的方式来强制实施自定义的策略。

> **说明：** 如果准入 Webhook 需要保证它们所看到的是对象的最终状态以实施某种策略。 则应使用验证性质的准入 Webhook，因为对象被修改性质 Webhook 看到之后仍然可能被修改。

尝试准入 Webhook

准入 Webhook 本质上是集群控制平面的一部分。你应该非常谨慎地编写和部署它们。 如果你打算编写或者部署生产级准入 webhook，请阅读用户指南以获取相关说明。 在下文中，我们将介绍如何快速试验准入 Webhook。

先决条件

- 确保 Kubernetes 集群版本至少为 v1.16（以便使用 `admissionregistration.k8s.io/v1` API）或者 v1.9 （以便使用 `admissionregistration.k8s.io/v1beta1` API）。

- 确保启用 MutatingAdmissionWebhook 和 ValidatingAdmissionWebhook 控制器。 这里 是一组推荐的 admission 控制器，通常可以启用。
- 确保启用了 admissionregistration.k8s.io/v1beta1 API。

编写一个准入 Webhook 服务器

请参阅 Kubernetes e2e 测试中的 admission webhook 服务器 的实现。webhook 处理由 apiserver 发送的 AdmissionReview 请求，并且将其决定作为 AdmissionReview 对象以相同版本发送回去。

有关发送到 webhook 的数据的详细信息，请参阅 webhook 请求。

要获取来自 webhook 的预期数据，请参阅 webhook 响应。

示例准入 Webhook 服务器置 ClientAuth 字段为空，默认为 NoClientCert 。这意味着 webhook 服务器不会验证客户端的身份，认为其是 apiservers。 如果你需要双向 TLS 或其他方式来验证客户端，请参阅如何对 apiservers 进行身份认证。

部署准入 Webhook 服务

e2e 测试中的 webhook 服务器通过 deployment API 部署在 Kubernetes 集群中。该测试还将创建一个 service 作为 webhook 服务器的前端。参见相关代码。

你也可以在集群外部署 webhook。这样做需要相应地更新你的 webhook 配置。

即时配置准入 Webhook

你可以通过 ValidatingWebhookConfiguration 或者 MutatingWebhookConfiguration 动态配置哪些资源要被哪些准入 Webhook 处理。

以下是一个 ValidatingWebhookConfiguration 示例，mutating webhook 配置与此类似。有关每个配置字段的详细信息，请参阅 webhook 配置 部分。

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: "pod-policy.example.com"
webhooks:
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  ["CREATE"]
    resources:   ["pods"]
    scope:       "Namespaced"
```

```yaml
  clientConfig:
    service:
      namespace: "example-namespace"
      name: "example-service"
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM bundle containing the CA that signed the webhook's serving certificate>...tLS0K"
  admissionReviewVersions: ["v1", "v1beta1"]
  sideEffects: None
  timeoutSeconds: 5
```

```yaml
# 1.16 中被淘汰，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: "pod-policy.example.com"
webhooks:
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  ["CREATE"]
    resources:   ["pods"]
    scope:       "Namespaced"
  clientConfig:
    service:
      namespace: "example-namespace"
      name: "example-service"
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM bundle containing the CA that signed the webhook's serving certificate>...tLS0K"
  admissionReviewVersions: ["v1beta1"]
  timeoutSeconds: 5
```

scope 字段指定是仅集群范围的资源（Cluster）还是名字空间范围的资源资源（Namespaced）将与此规则匹配。* 表示没有范围限制。

> **说明：** 当使用 clientConfig.service 时，服务器证书必须对 <svc_name>.<svc_namespace>.svc 有效。

> **说明：** 对于使用 admissionregistration.k8s.io/v1 创建的 webhook 而言，其 webhook 调用的默认超时是 10 秒； 对于使用 admissionregistration.k8s.io/v1beta1 创建的 webhook 而言，其默认超时是 30 秒。 从 kubernetes 1.14 开始，可以设置超时。建议对 webhooks 设置较短的超时时间。 如果 webhook 调用超时，则根据 webhook 的失败策略处理请求。

当 apiserver 收到与 rules 相匹配的请求时，apiserver 按照 clientConfig 中指定的方式向 webhook 发送一个 admissionReview 请求。

创建 webhook 配置后，系统将花费几秒钟使新配置生效。

对 apiservers 进行身份认证

如果你的 webhook 需要身份验证，则可以将 apiserver 配置为使用基本身份验证、持有者令牌或证书来向 webhook 提供身份证明。完成此配置需要三个步骤。

- 启动 apiserver 时，通过 `--admission-control-config-file` 参数指定准入控制配置文件的位置。
- 在准入控制配置文件中，指定 MutatingAdmissionWebhook 控制器和 ValidatingAdmissionWebhook 控制器应该读取凭据的位置。 凭证存储在 kubeConfig 文件中（是的，与 kubectl 使用的模式相同），因此字段名称为 kubeConfigFile。 以下是一个准入控制配置文件示例：
- apiserver.config.k8s.io/v1
- apiserver.k8s.io/v1alpha1

```yaml
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: ValidatingAdmissionWebhook
  configuration:
    apiVersion: apiserver.config.k8s.io/v1
    kind: WebhookAdmissionConfiguration
    kubeConfigFile: "<path-to-kubeconfig-file>"
- name: MutatingAdmissionWebhook
  configuration:
    apiVersion: apiserver.config.k8s.io/v1
    kind: WebhookAdmissionConfiguration
    kubeConfigFile: "<path-to-kubeconfig-file>"
```

```yaml
# 1.17 中被淘汰，推荐使用 apiserver.config.k8s.io/v1
apiVersion: apiserver.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
- name: ValidatingAdmissionWebhook
  configuration:
    # 1.17 中被淘汰，推荐使用 apiserver.config.k8s.io/v1，kind = WebhookAdmissionConfiguration
    apiVersion: apiserver.config.k8s.io/v1alpha1
    kind: WebhookAdmission
    kubeConfigFile: "<path-to-kubeconfig-file>"
- name: MutatingAdmissionWebhook
  configuration:
    # 1.17 中被淘汰，推荐使用 apiserver.config.k8s.io/v1，kind = WebhookAdmissionConfiguration
    apiVersion: apiserver.config.k8s.io/v1alpha1
```

```
  kind: WebhookAdmission
  kubeConfigFile: "<path-to-kubeconfig-file>"
```

有关 AdmissionConfiguration 的更多信息，请参见 AdmissionConfiguration schema。 有关每个配置字段的详细信息，请参见 webhook 配置部分。

- 在 kubeConfig 文件中，提供证书凭据：

```
apiVersion: v1
kind: Config
users:
# 名称应设置为服务的 DNS 名称或配置了 Webhook 的 URL 的主机名（包括端口）。
# 如果将非 443 端口用于服务，则在配置 1.16+ API 服务器时，该端口必须包含在名称中。
#
# 对于配置在默认端口（443）上与服务对话的 Webhook，请指定服务的 DNS 名称：
# - name: webhook1.ns1.svc
#   user: ...
#
# 对于配置在非默认端口（例如 8443）上与服务对话的 Webhook，请在 1.16+ 中指定服务的 DNS 名称和端口：
# - name: webhook1.ns1.svc:8443
#   user: ...
# 并可以选择仅使用服务的 DNS 名称来创建第二节，以与 1.15 API 服务器版本兼容：
# - name: webhook1.ns1.svc
#   user: ...
#
# 对于配置为使用 URL 的 webhook，请匹配在 webhook 的 URL 中指定的主机（和端口）。
# 带有 `url: https://www.example.com` 的 webhook：
# - name: www.example.com
#   user: ...
#
# 带有 `url: https://www.example.com:443` 的 webhook：
# - name: www.example.com:443
#   user: ...
#
# 带有 `url: https://www.example.com:8443` 的 webhook：
# - name: www.example.com:8443
#   user: ...
#
- name: 'webhook1.ns1.svc'
  user:
```

```
    client-certificate-data: "<pem encoded certificate>"
    client-key-data: "<pem encoded key>"
# `name` 支持使用 * 通配符匹配前缀段。
- name: '*.webhook-company.org'
  user:
    password: "<password>"
    username: "<name>"
# '*' 是默认匹配项。
- name: '*'
  user:
    token: "<token>"
```

当然，你需要设置 webhook 服务器来处理这些身份验证。

请求

向 Webhook 发送 POST 请求时，请设置 Content-Type: application/json 并对 admission.k8s.io API 组中的 AdmissionReview 对象进行序列化，将所得到的 JSON 作为请求的主体。

Webhook 可以在配置中的 admissionReviewVersions 字段指定可接受的 AdmissionReview 对象版本：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  admissionReviewVersions: ["v1", "v1beta1"]
  ...
```

创建 admissionregistration.k8s.io/v1 webhook 配置时，admissionReviewVersions 是必填字段。 Webhook 必须支持至少一个当前和以前的 apiserver 都可以解析的 AdmissionReview 版本。

```
# v1.16 中被淘汰，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  admissionReviewVersions: ["v1beta1"]
  ...
```

如果未指定 admissionReviewVersions，则创建 admissionregistration.k8s.io/v1beta1 Webhook 配置时的默认值为 v1beta1。

API 服务器将发送的是 admissionReviewVersions 列表中所支持的第一个 AdmissionReview 版本。如果 API 服务器不支持列表中的任何版本，则不允许创建配置。

如果 API 服务器遇到以前创建的 Webhook 配置，并且不支持该 API 服务器知道如何发送的任何 AdmissionReview 版本，则调用 Webhook 的尝试将失败，并依据失败策略进行处理。

此示例显示了 AdmissionReview 对象中包含的数据，该数据用于请求更新 apps/v1 Deployment 的 scale 子资源：

- admission.k8s.io/v1
- admission.k8s.io/v1beta1

```
{
  "apiVersion": "admission.k8s.io/v1",
  "kind": "AdmissionReview",
  "request": {
    # 唯一标识此准入回调的随机 uid
    "uid": "705ab4f5-6393-11e8-b7cc-42010a800002",

    # 传入完全正确的 group/version/kind 对象
    "kind": {"group":"autoscaling","version":"v1","kind":"Scale"},
    # 修改 resource 的完全正确的的 group/version/kind
    "resource": {"group":"apps","version":"v1","resource":"deployments"},
    # subResource（如果请求是针对 subResource 的）
    "subResource": "scale",

    # 在对 API 服务器的原始请求中，传入对象的标准 group/version/kind
    # 仅当 webhook 指定 `matchPolicy: Equivalent` 且将对 API 服务器的原始请求转换为 webhook 注册的版本时，这才与 `kind` 不同。
    "requestKind": {"group":"autoscaling","version":"v1","kind":"Scale"},
    # 在对 API 服务器的原始请求中正在修改的资源的标准 group/version/kind
    # 仅当 webhook 指定了 `matchPolicy：Equivalent` 并且将对 API 服务器的原始请求转换为 webhook 注册的版本时，这才与 `resource` 不同。
    "requestResource": {"group":"apps","version":"v1","resource":"deployments"},
    # subResource（如果请求是针对 subResource 的）
    # 仅当 webhook 指定了 `matchPolicy：Equivalent` 并且将对 API 服务器的原始请求转换为该 webhook 注册的版本时，这才与 `subResource` 不同。
    "requestSubResource": "scale",

    # 被修改资源的名称
    "name": "my-deployment",
    # 如果资源是属于名字空间（或者是名字空间对象），则这是被修改的资源的名字空间
    "namespace": "my-namespace",
```

```
    # 操作可以是 CREATE、UPDATE、DELETE 或 CONNECT
    "operation": "UPDATE",

    "userInfo": {
      # 向 API 服务器发出请求的经过身份验证的用户的用户名
      "username": "admin",
      # 向 API 服务器发出请求的经过身份验证的用户的 UID
      "uid": "014fbff9a07c",
      # 向 API 服务器发出请求的经过身份验证的用户的组成员身份
      "groups": ["system:authenticated","my-admin-group"],
      # 向 API 服务器发出请求的用户相关的任意附加信息
      # 该字段由 API 服务器身份验证层填充，并且如果 webhook 执行了任何 SubjectAccessReview 检查，则应将其包括在内。
      "extra": {
        "some-key":["some-value1", "some-value2"]
      }
    },

    # object 是被接纳的新对象。
    # 对于 DELETE 操作，它为 null。
    "object": {"apiVersion":"autoscaling/v1","kind":"Scale",...},
    # oldObject 是现有对象。
    # 对于 CREATE 和 CONNECT 操作，它为 null。
    "oldObject": {"apiVersion":"autoscaling/v1","kind":"Scale",...},
    # options 包含要接受的操作的选项，例如 meta.k8s.io/v CreateOptions、UpdateOptions 或 DeleteOptions。
    # 对于 CONNECT 操作，它为 null。
    "options": {"apiVersion":"meta.k8s.io/v1","kind":"UpdateOptions",...},

    # dryRun 表示 API 请求正在以 `dryrun` 模式运行，并且将不会保留。
    # 带有副作用的 Webhook 应该避免在 dryRun 为 true 时激活这些副作用。
    # 有关更多详细信息，请参见 http://k8s.io/docs/reference/using-api/api-concepts/#make-a-dry-run-request
    "dryRun": false
  }
}
```

```
{
  # v1.16 中被废弃，推荐使用 admission.k8s.io/v1
  "apiVersion": "admission.k8s.io/v1beta1",
  "kind": "AdmissionReview",
  "request": {
    # 唯一标识此准入回调的随机 uid
    "uid": "705ab4f5-6393-11e8-b7cc-42010a800002",
```

```
  # 传入完全正确的 group/version/kind 对象
  "kind": {"group":"autoscaling","version":"v1","kind":"Scale"},
  # 修改 resource 的完全正确的的 group/version/kind
  "resource": {"group":"apps","version":"v1","resource":"deployments"},
  # subResource（如果请求是针对 subResource 的）
  "subResource": "scale",

  # 在对 API 服务器的原始请求中，传入对象的标准 group/version/kind。
  # 仅当 Webhook 指定了 `matchPolicy：Equivalent` 并且将对 API 服务器的原始请求转换为该 Webhook 注册的版本时，这与 `kind` 不同。
  # 仅由 v1.15+ API 服务器发送。
  "requestKind": {"group":"autoscaling","version":"v1","kind":"Scale"},
  # 在对 API 服务器的原始请求中正在修改的资源的标准 group/version/kind
  # 仅当 webhook 指定了 `matchPolicy：Equivalent` 并且将对 API 服务器的原始请求转换为 webhook 注册的版本时，这才与 `resource` 不同。
  # 仅由 v1.15+ API 服务器发送。
  "requestResource": {"group":"apps","version":"v1","resource":"deployments"},
  # subResource（如果请求是针对 subResource 的）
  # 仅当 webhook 指定了 `matchPolicy：Equivalent` 并且将对 API 服务器的原始请求转换为该 webhook 注册的版本时，这才与 `subResource` 不同。
  # 仅由 v1.15+ API 服务器发送。
  "requestSubResource": "scale",

  # 被修改资源的名称
  "name": "my-deployment",
  # 如果资源是属于名字空间（或者是名字空间对象），则这是被修改的资源的名字空间
  "namespace": "my-namespace",

  # 操作可以是 CREATE、UPDATE、DELETE 或 CONNECT
  "operation": "UPDATE",

  "userInfo": {
    # 向 API 服务器发出请求的经过身份验证的用户的用户名
    "username": "admin",
    # 向 API 服务器发出请求的经过身份验证的用户的 UID
    "uid": "014fbff9a07c",
    # 向 API 服务器发出请求的经过身份验证的用户的组成员身份
    "groups": ["system:authenticated","my-admin-group"],
    # 向 API 服务器发出请求的用户相关的任意附加信息
    # 该字段由 API 服务器身份验证层填充，并且如果 webhook 执行了任何 SubjectAccessReview 检查，则应将其包括在内。
    "extra": {
      "some-key":["some-value1", "some-value2"]
    }
  },
```

```
    # object 是被接纳的新对象。
    # 对于 DELETE 操作，它为 null。
    "object": {"apiVersion":"autoscaling/v1","kind":"Scale",...},
    # oldObject 是现有对象。
    # 对于 CREATE 和 CONNECT 操作（对于 v1.15.0 之前版本的 API 服务器中的 DELETE 操作），它为 null。
    "oldObject": {"apiVersion":"autoscaling/v1","kind":"Scale",...},
    # options 包含要接受的操作的选项，例如 meta.k8s.io/v CreateOptions、UpdateOptions 或 DeleteOptions。
    # 对于 CONNECT 操作，它为 null。
    # 仅由 v1.15+ API 服务器发送。
    "options": {"apiVersion":"meta.k8s.io/v1","kind":"UpdateOptions",...},

    # dryRun 表示 API 请求正在以 `dryrun` 模式运行，并且将不会保留。
    # 带有副作用的 Webhook 应该避免在 dryRun 为 true 时激活这些副作用。
    # 有关更多详细信息，请参见 http://k8s.io/docs/reference/using-api/api-concepts/#make-a-dry-run-request
    "dryRun": false
  }
}
```

响应

Webhook 使用 HTTP 200 状态码、Content-Type: application/json 和一个包含 AdmissionReview 对象的 JSON 序列化格式来发送响应。该 AdmissionReview 对象与发送的版本相同，且其中包含的 response 字段已被有效填充。

response 至少必须包含以下字段：

- uid，从发送到 webhook 的 request.uid 中复制而来
- allowed，设置为 true 或 false

Webhook 允许请求的最简单响应示例：

- admission.k8s.io/v1
- admission.k8s.io/v1beta1

```
{
  "apiVersion": "admission.k8s.io/v1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": true
  }
}
```

```
{
  "apiVersion": "admission.k8s.io/v1beta1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": true
  }
}
```

Webhook 禁止请求的最简单响应示例：

- admission.k8s.io/v1
- admission.k8s.io/v1beta1

```
{
  "apiVersion": "admission.k8s.io/v1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": false
  }
}
```

```
{
  "apiVersion": "admission.k8s.io/v1beta1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": false
  }
}
```

当拒绝请求时，Webhook 可以使用 status 字段自定义 http 响应码和返回给用户的消息。 有关状态类型的详细信息，请参见 API 文档。 禁止请求的响应示例，它定制了向用户显示的 HTTP 状态码和消息：

- admission.k8s.io/v1
- admission.k8s.io/v1beta1

```
{
  "apiVersion": "admission.k8s.io/v1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": false,
    "status": {
      "code": 403,
      "message": "You cannot do this because it is Tuesday and your name starts
```

```
with A"
    }
  }
}
```

```
{
  "apiVersion": "admission.k8s.io/v1beta1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": false,
    "status": {
      "code": 403,
      "message": "You cannot do this because it is Tuesday and your name starts
with A"
    }
  }
}
```

当允许请求时，mutating准入 Webhook 也可以选择修改传入的对象。 这是通过在响应中使用 patch 和 patchType 字段来完成的。 当前唯一支持的 patchType 是 JSONPatch。 有关更多详细信息，请参见 JSON patch。 对于 patchType: JSONPatch，patch 字段包含一个以 base64 编码的 JSON patch 操作数组。

例如，设置 spec.replicas 的单个补丁操作将是 [{"op": "add", "path": "/spec/replicas", "value": 3}]。

如果以 Base64 形式编码，结果将是 W3sib3AiOiAiYWRkIiwgInBhdGgiOiAiL3NwZWMvcmVwbGljYXMiLCAidmFsdWUiOiAzfV0=

因此，添加该标签的 webhook 响应为：

- admission.k8s.io/v1
- admission.k8s.io/v1beta1

```
{
  "apiVersion": "admission.k8s.io/v1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": true,
    "patchType": "JSONPatch",
    "patch": "W3sib3AiOiAiYWRkIiwgInBhdGgiOiAiL3NwZWMvcmVwbGljYXMiLCAidmFsdWUiOiAzfV0="
  }
}
```

```
{
  "apiVersion": "admission.k8s.io/v1beta1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": true,
    "patchType": "JSONPatch",
    "patch": "W3sib3AiOiAiYWRkIiwgInBhdGgiOiAiL3NwZWMvcmVwbGljYXMiLCAidmFsdWUiOiAzfV0="
  }
}
```


Webhook 配置

要注册准入 Webhook，请创建 MutatingWebhookConfiguration 或 ValidatingWebhookConfiguration API 对象。

每种配置可以包含一个或多个 Webhook。如果在单个配置中指定了多个 Webhook，则应为每个 webhook 赋予一个唯一的名称。 这在 admissionregistration.k8s.io/v1 中是必需的，但是在使用 admissionregistration.k8s.io/v1beta1 时强烈建议使用， 以使生成的审核日志和指标更易于与活动配置相匹配。

每个 Webhook 定义以下内容。

匹配请求-规则

每个 webhook 必须指定用于确定是否应将对 apiserver 的请求发送到 webhook 的规则列表。 每个规则都指定一个或多个 operations、apiGroups、apiVersions 和 resources 以及资源的 scope：

- operations 列出一个或多个要匹配的操作。 可以是 CREATE、UPDATE、DELETE、CONNECT 或 * 以匹配所有内容。
- apiGroups 列出了一个或多个要匹配的 API 组。"" 是核心 API 组。"*" 匹配所有 API 组。
- apiVersions 列出了一个或多个要匹配的 API 版本。"*" 匹配所有 API 版本。
- resources 列出了一个或多个要匹配的资源。
  - "*" 匹配所有资源，但不包括子资源。
  - "*/*" 匹配所有资源，包括子资源。
  - "pods/*" 匹配 pod 的所有子资源。
  - "*/status" 匹配所有 status 子资源。
- scope 指定要匹配的范围。有效值为 "Cluster"、"Namespaced" 和 "*"。 子资源匹配其父资源的范围。在 Kubernetes v1.14+ 版本中才被支持。 默认值为 "*"，对应 1.14 版本之前的行为。
  - "Cluster" 表示只有集群作用域的资源才能匹配此规则（API 对象 Namespace 是集群作用域的）。
  - "Namespaced" 意味着仅具有名字空间的资源才符合此规则。
  - "*" 表示没有范围限制。

如果传入请求与任何 Webhook 规则的指定操作、组、版本、资源和范围匹配，则该请求将发送到 Webhook。

以下是可用于指定应拦截哪些资源的规则的其他示例。

匹配针对 apps/v1 和 apps/v1beta1 组中 deployments 和 replicasets 资源的 CREATE 或 UPDATE 请求：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  rules:
  - operations: ["CREATE", "UPDATE"]
    apiGroups: ["apps"]
    apiVersions: ["v1", "v1beta1"]
    resources: ["deployments", "replicasets"]
    scope: "Namespaced"
  ...
```

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  rules:
  - operations: ["CREATE", "UPDATE"]
    apiGroups: ["apps"]
    apiVersions: ["v1", "v1beta1"]
    resources: ["deployments", "replicasets"]
    scope: "Namespaced"
  ...
```

匹配所有 API 组和版本中的所有资源（但不包括子资源）的创建请求：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
```

```
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
  ...
```

```
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
  ...
```

匹配所有 API 组和版本中所有 status 子资源的更新请求：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
  ...
```

```
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  rules:
```

```
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
  ...
```

匹配请求：objectSelector

在版本 v1.15+ 中，通过指定 objectSelector，Webhook 能够根据 可能发送的对象的标签来限制哪些请求被拦截。 如果指定，则将对 objectSelector 和可能发送到 Webhook 的 object 和 oldObject 进行评估。如果两个对象之一与选择器匹配，则认为该请求已匹配。

空对象（对于创建操作而言为 oldObject，对于删除操作而言为 newObject）， 或不能带标签的对象（例如 DeploymentRollback 或 PodProxyOptions 对象） 被认为不匹配。

仅当选择使用 webhook 时才使用对象选择器，因为最终用户可以通过设置标签来 跳过准入 Webhook。

这个例子展示了一个 mutating webhook，它将匹配带有标签 foo:bar 的任何资源的 CREATE 的操作：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  objectSelector:
    matchLabels:
      foo: bar
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
  ...
```

```
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
```

```
- name: my-webhook.example.com
  objectSelector:
    matchLabels:
      foo: bar
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
  ...
```

有关标签选择器的更多示例，请参见标签。

匹配请求：namespaceSelector

通过指定 namespaceSelector，Webhook 可以根据具有名字空间的资源所处的 名字空间的标签来选择拦截哪些资源的操作。

namespaceSelector 根据名字空间的标签是否匹配选择器，决定是否针对具名字空间的资源（或 Namespace 对象）的请求运行 webhook。 如果对象是除 Namespace 以外的集群范围的资源，则 namespaceSelector 标签无效。

本例给出的修改性质的 Webhook 将匹配到对名字空间中具名字空间的资源的 CREATE 请求， 前提是这些资源不含值为 "0" 或 "1" 的 "runlevel" 标签：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  namespaceSelector:
    matchExpressions:
    - key: runlevel
      operator: NotIn
      values: ["0","1"]
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "Namespaced"
  ...
```

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  namespaceSelector:
    matchExpressions:
    - key: runlevel
      operator: NotIn
      values: ["0","1"]
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "Namespaced"
  ...
```

此示例显示了一个验证性质的 Webhook，它将匹配到对某名字空间中的任何具名字空间的资源的 CREATE 请求，前提是该名字空间具有值为 "prod" 或 "staging" 的 "environment" 标签：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  namespaceSelector:
    matchExpressions:
    - key: environment
      operator: In
      values: ["prod","staging"]
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "Namespaced"
  ...
```

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
```

```yaml
...
webhooks:
- name: my-webhook.example.com
  namespaceSelector:
    matchExpressions:
    - key: environment
      operator: In
      values: ["prod","staging"]
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "Namespaced"
...
```

有关标签选择器的更多示例，请参见 标签。

匹配请求：matchPolicy

API 服务器可以通过多个 API 组或版本来提供对象。 例如，Kubernetes API 服务器允许通过 extensions/v1beta1、apps/v1beta1、 apps/v1beta2 和 apps/v1 API 创建和修改 Deployment 对象。

例如，如果一个 webhook 仅为某些 API 组/版本指定了规则（例如 apiGroups:["apps"], apiVersions:["v1","v1beta1"]），而修改资源的请求 是通过另一个 API 组/版本（例如 extensions/v1beta1）发出的， 该请求将不会被发送到 Webhook。

在 v1.15+ 中，matchPolicy 允许 webhook 定义如何使用其 rules 匹配传入的请求。允许的值为 Exact 或 Equivalent。

- Exact 表示仅当请求与指定规则完全匹配时才应拦截该请求。
- Equivalent 表示如果某个请求意在修改 rules 中列出的资源， 即使该请求是通过其他 API 组或版本发起，也应拦截该请求。

在上面给出的示例中，仅为 apps/v1 注册的 webhook 可以使用 matchPolicy：

- matchPolicy: Exact 表示不会将 extensions/v1beta1 请求发送到 Webhook
- matchPolicy:Equivalent 表示将 extensions/v1beta1 请求发送到 webhook （将对象转换为 webhook 指定的版本：apps/v1）

建议指定 Equivalent，确保升级后启用 API 服务器中资源的新版本时， Webhook 继续拦截他们期望的资源。

当 API 服务器停止提供某资源时，该资源不再被视为等同于该资源的其他仍在提供服务的版本。 例如，extensions/v1beta1 中的 Deployment 已被废弃，计划在 v1.16 中默认停止使用。 在这种情况下，带有 apiGroups:["extensions"], apiVersions:["v1beta1"], resources: ["deployments"] 规则的 Webhook 将不再拦截通过 apps/v1

API 来创建 Deployment 的请求。 ["deployments"] 规则将不再拦截通过 apps/v1 API 创建的部署。

此示例显示了一个验证性质的 Webhook，该 Webhook 拦截对 Deployment 的修改（无论 API 组或版本是什么），始终会发送一个 apps/v1 版本的 Deployment 对象：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  matchPolicy: Equivalent
  rules:
  - operations: ["CREATE","UPDATE","DELETE"]
    apiGroups: ["apps"]
    apiVersions: ["v1"]
    resources: ["deployments"]
    scope: "Namespaced"
  ...
```

使用 admissionregistration.k8s.io/v1 创建的 admission webhhok 默认为 Equivalent。

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  matchPolicy: Equivalent
  rules:
  - operations: ["CREATE","UPDATE","DELETE"]
    apiGroups: ["apps"]
    apiVersions: ["v1"]
    resources: ["deployments"]
    scope: "Namespaced"
  ...
```

使用 admissionregistration.k8s.io/v1beta1 创建的准入 Webhook 默认为 Exact。

调用 Webhook

API 服务器确定请求应发送到 webhook 后，它需要知道如何调用 webhook。 此信息在 webhook 配置的 clientConfig 节中指定。

Webhook 可以通过 URL 或服务引用来调用，并且可以选择包含自定义 CA 包，以用于验证 TLS 连接。

URL

`url` 以标准 URL 形式给出 webhook 的位置（`scheme://host:port/path`）。

`host` 不应引用集群中运行的服务；通过指定 `service` 字段来使用服务引用。 主机可以通过某些 apiserver 中的外部 DNS 进行解析。 （例如，kube-apiserver 无法解析集群内 DNS，因为这将违反分层规则）。`host` 也可以是 IP 地址。

请注意，将 localhost 或 127.0.0.1 用作 host 是有风险的， 除非你非常小心地在所有运行 apiserver 的、可能需要对此 webhook 进行调用的主机上运行。这样的安装可能不具有可移植性，即很难在新集群中启用。

scheme 必须为 "https"；URL 必须以 "https://" 开头。

使用用户或基本身份验证（例如："user:password@"）是不允许的。 使用片段（"#..."）和查询参数（"?..."）也是不允许的。

这是配置为调用 URL 的修改性质的 Webhook 的示例 （并且期望使用系统信任根证书来验证 TLS 证书，因此不指定 caBundle）：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  clientConfig:
    url: "https://my-webhook.example.com:9443/my-webhook-path"
  ...
```

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  clientConfig:
    url: "https://my-webhook.example.com:9443/my-webhook-path"
  ...
```

服务引用

clientConfig 内部的 Service 是对该 Webhook 服务的引用。 如果 Webhook 在集群中运行，则应使用 service 而不是 url。 服务的 namespace 和 name 是必需的。 port 是可选的，默认值为 443。path 是可选的，默认为 "/"。

这是一个 mutating Webhook 的示例，该 mutating Webhook 配置为在子路径 "/my-path" 端口 "1234" 上调用服务，并使用自定义 CA 包针对 ServerName my-service-name.my-service-namespace.svc 验证 TLS 连接：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  clientConfig:
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM bundle containing the CA that signed the webhook's serving certificate>...tLS0K"
    service:
      namespace: my-service-namespace
      name: my-service-name
      path: /my-path
      port: 1234
  ...
```

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  clientConfig:
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM bundle containing the CA that signed the webhook's serving certificate>...tLS0K"
    service:
      namespace: my-service-namespace
      name: my-service-name
      path: /my-path
      port: 1234
  ...
```

副作用

Webhook 通常仅对发送给他们的 AdmissionReview 内容进行操作。 但是，某些 Webhook 在处理 admission 请求时会进行带外更改。

进行带外更改的（产生"副作用"的） Webhook 必须具有协调机制（如控制器），该机制定期确定事物的实际状态，并调整由准入 Webhook 修改的带外数据以反映现实情况。 这是因为对准入 Webhook 的调用不能保证所准入的对象将原样保留，或根本不保留。 以后，webhook 可以修改对象的内容，在写入存储时可能会发生冲突，或者 服务器可以在持久保存对象之前关闭电源。

此外，处理 dryRun: true admission 请求时，具有副作用的 Webhook 必须避免产生副作用。 一个 Webhook 必须明确指出在使用 dryRun 运行时不会有副作用， 否则 dry-run 请求将不会发送到该 Webhook，而 API 请求将会失败。

Webhook 使用 webhook 配置中的 sideEffects 字段显示它们是否有副作用：

- Unknown：有关调用 Webhook 的副作用的信息是不可知的。 如果带有 dryRun：true 的请求将触发对该 Webhook 的调用，则该请求将失败，并且不会调用该 Webhook。
- None：调用 webhook 没有副作用。
- Some：调用 webhook 可能会有副作用。 如果请求具有 dry-run 属性将触发对此 Webhook 的调用， 则该请求将会失败，并且不会调用该 Webhook。
- NoneOnDryRun：调用 webhook 可能会有副作用，但是如果将带有 dryRun: true 属性的请求发送到 webhook，则 webhook 将抑制副作用（该 webhook 可识别 dryRun）。

允许值：

- 在 admissionregistration.k8s.io/v1beta1 中，sideEffects 可以设置为 Unknown、None、Some 或者 NoneOnDryRun，并且默认值为 Unknown。
- 在 admissionregistration.k8s.io/v1 中，sideEffects 必须设置为 None 或者 NoneOnDryRun。

这是一个 validating webhook 的示例，表明它对 dryRun: true 请求没有副作用：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  sideEffects: NoneOnDryRun
  ...
```

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
```

```yaml
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  sideEffects: NoneOnDryRun
  ...
```

超时

由于 Webhook 会增加 API 请求的延迟，因此应尽快完成自身的操作。 timeoutSeconds 用来配置在将调用视为失败之前，允许 API 服务器等待 Webhook 响应的时间长度。

如果超时在 Webhook 响应之前被触发，则基于失败策略，将忽略 Webhook 调用或拒绝 API 调用。

超时值必须设置在 1 到 30 秒之间。

这是一个自定义超时设置为 2 秒的 validating Webhook 的示例：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  timeoutSeconds: 2
  ...
```

使用 admissionregistration.k8s.io/v1 创建的准入 Webhook 默认超时为 10 秒。

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  timeoutSeconds: 2
  ...
```

使用 admissionregistration.k8s.io/v1beta1 创建的准入 Webhook 默认超时为 30 秒。

再调用策略

修改性质的准入插件（包括 Webhook）的任何一种排序方式都不会适用于所有情况。 (参见 https://issue.k8s.io/64333 示例)。 修改性质的 Webhook 可以向对象中添加新

的子结构（例如向 pod 中添加 container），已经运行的其他修改插件可能会对这些新结构有影响（就像在所有容器上设置 imagePullPolicy 一样）。

在 v1.15+ 中，允许修改性质的准入插件感应到其他插件所做的更改，如果修改性质的 Webhook 修改了一个对象，则会重新运行内置的修改性质的准入插件，并且修改性质的 Webhook 可以指定 reinvocationPolicy 来控制是否也重新调用它们。

可以将 reinvocationPolicy 设置为 Never 或 IfNeeded。默认为 Never。

- Never: 在一次准入测试中，不得多次调用 Webhook。
- IfNeeded: 如果在最初的 Webhook 调用之后被其他对象的插件修改了被接纳的对象，则可以作为准入测试的一部分再次调用该 webhook。

要注意的重要因素有：

- 不能保证附加调用的次数恰好是一。
- 如果其他调用导致对该对象的进一步修改，则不能保证再次调用 Webhook。
- 使用此选项的 Webhook 可能会重新排序，以最大程度地减少额外调用的次数。
- 要在确保所有修改都完成后验证对象，请改用验证性质的 Webhook（推荐用于有副作用的 Webhook）。

这是一个修改性质的 Webhook 的示例，该 Webhook 在以后的准入插件修改对象时被重新调用：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  reinvocationPolicy: IfNeeded
  ...
```

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  reinvocationPolicy: IfNeeded
  ...
```

修改性质的 Webhook 必须具有幂等性，并且能够成功处理 已被接纳并可能被修改的对象的修改性质的 Webhook。对于所有修改性质的准入 Webhook 都是如此，因为它们可以在对象中进行的 任何更改可能已经存在于用户提供的对象中，但是对于选择重新调用的 webhook 来说是必不可少的。

失败策略

failurePolicy 定义了如何处理准入 webhook 中无法识别的错误和超时错误。允许的值为 Ignore 或 Fail。

- Ignore 表示调用 webhook 的错误将被忽略并且允许 API 请求继续。
- Fail 表示调用 webhook 的错误导致准入失败并且 API 请求被拒绝。

这是一个修改性质的 webhook，配置为在调用准入 Webhook 遇到错误时拒绝 API 请求：

- admissionregistration.k8s.io/v1
- admissionregistration.k8s.io/v1beta1

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  failurePolicy: Fail
  ...
```

使用 admissionregistration.k8s.io/v1beta1 创建的准入 Webhook 将 failurePolicy 默认设置为 Ignore。

```yaml
# v1.16 中被废弃，推荐使用 admissionregistration.k8s.io/v1
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  failurePolicy: Fail
  ...
```

使用 admissionregistration.k8s.io/v1beta1 创建的准入 Webhook 将 failurePolicy 默认设置为 Ignore。

监控 Admission Webhook

API 服务器提供了监视准入 Webhook 行为的方法。这些监视机制可帮助集群管理员 回答以下问题：

1. 哪个修改性质的 webhook 改变了 API 请求中的对象？
2. 修改性质的 Webhook 对对象做了哪些更改？
3. 哪些 webhook 经常拒绝 API 请求？是什么原因拒绝？

Mutating Webhook 审计注解

有时，了解 API 请求中的哪个修改性质的 Webhook 使对象改变以及该 Webhook 应用了哪些更改很有用。

在 v1.16+ 中，kube-apiserver 针对每个修改性质的 Webhook 调用执行 审计操作。每个调用都会生成一个审计注解，记述请求对象是否发生改变， 可选地还可以根据 webhook 的准入响应生成一个注解，记述所应用的修补。 针对给定请求的给定执行阶段，注解被添加到审计事件中， 然后根据特定策略进行预处理并写入后端。

事件的审计级别决定了要记录哪些注解：

在 Metadata 或更高审计级别上，将使用 JSON 负载记录带有键名 mutation.webhook.admission.k8s.io/round_{round idx}_index_{order idx} 的注解， 该注解表示针对给定请求调用了 Webhook，以及该 Webhook 是否更改了对象。

例如，对于正在被重新调用的某 Webhook，所记录的注解如下。 Webhook 在 mutating Webhook 链中排在第三个位置，并且在调用期间未改变请求对象。

```
# 审计事件相关记录
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
    "annotations": {
        "mutation.webhook.admission.k8s.io/round_1_index_2": "{\"configuration\":\"my-mutating-webhook-configuration.example.com\",\"webhook\":\"my-webhook.example.com\",\"mutated\": false}"
        # 其他注解
        ...
    }
    # 其他字段
    ...
}
```

```
# 反序列化的注解值
{
    "configuration": "my-mutating-webhook-configuration.example.com",
    "webhook": "my-webhook.example.com",
    "mutated": false
}
```

对于在第一轮中调用的 Webhook，所记录的注解如下。 Webhook 在 mutating Webhook 链中排在第一位，并在调用期间改变了请求对象。

```
# 审计事件相关记录
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
```

```
    "annotations": {
        "mutation.webhook.admission.k8s.io/round_0_index_0": "{\"configuration\":
\"my-mutating-webhook-configuration.example.com\",\"webhook\":\"my-
webhook-always-mutate.example.com\",\"mutated\": true}"
        # 其他注解
        ...
    }
    # 其他字段
    ...
}
```

```
# 反序列化的注解值
{
    "configuration": "my-mutating-webhook-configuration.example.com",
    "webhook": "my-webhook-always-mutate.example.com",
    "mutated": true
}
```

在 Request 或更高审计级别上，将使用 JSON 负载记录带有键名为 patch.webhook.admission.k8s.io/round_{round idx}_index_{order idx} 的注解， 该注解表明针对给定请求调用了 Webhook 以及应用于请求对象之上的修改。

例如，以下是针对正在被重新调用的某 Webhook 所记录的注解。 Webhook 在修改性质的 Webhook 链中排在第四，并在其响应中包含一个 JSON 补丁， 该补丁已被应用于请求对象。

```
# 审计事件相关记录
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
    "annotations": {
        "patch.webhook.admission.k8s.io/round_1_index_3": "{\"configuration\":
\"my-other-mutating-webhook-configuration.example.com\",\"webhook\":\"my-
webhook-always-mutate.example.com\",\"patch\":[{\"op\":\"add\",\"path\":\"/
data/mutation-stage\",\"value\":\"yes\"}],\"patchType\":\"JSONPatch\"}"
        # 其他注解
        ...
    }
    # 其他字段
    ...
}
```

```
# 反序列化的注解值
{
    "configuration": "my-other-mutating-webhook-configuration.example.com",
    "webhook": "my-webhook-always-mutate.example.com",
    "patchType": "JSONPatch",
```

```
  "patch": [
    {
      "op": "add",
      "path": "/data/mutation-stage",
      "value": "yes"
    }
  ]
}
```

准入 Webhook 度量值

Kube-apiserver 从 /metrics 端点公开 Prometheus 指标，这些指标可用于监控和诊断 apiserver 状态。以下指标记录了与准入 Webhook 相关的状态。

apiserver 准入 Webhook 拒绝次数

有时，了解哪些准入 Webhook 经常拒绝 API 请求以及拒绝的原因是很有用的。

在 v1.16+ 中，kube-apiserver 提供了 Prometheus 计数器度量值，记录 准入 Webhook 的拒绝次数。 度量值的标签给出了 Webhook 拒绝该请求的原因：

- name：拒绝请求 Webhook 的名称。
- operation：请求的操作类型可以是 CREATE、UPDATE、DELETE 和 CONNECT 其中之一。
- type：Admission webhook 类型，可以是 admit 和 validating 其中之一。
- error_type：标识在 webhook 调用期间是否发生了错误并且导致了拒绝。其值可以是以下之一：
  - calling_webhook_error：发生了来自准入 Webhook 的无法识别的错误或超时错误， 并且 webhook 的 失败策略 设置为 Fail。
  - no_error：未发生错误。Webhook 在准入响应中以 allowed: false 值拒绝了请求。 度量标签 rejection_code 记录了在准入响应中设置的 .status.code。
  - apiserver_internal_error：apiserver 发生内部错误。
- rejection_code：当 Webhook 拒绝请求时，在准入响应中设置的 HTTP 状态码。

拒绝计数指标示例：

```
# HELP apiserver_admission_webhook_rejection_count [ALPHA] Admission webhook rejection count, identified by name and broken out for each admission type (validating or admit) and operation. Additional labels specify an error type (calling_webhook_error or apiserver_internal_error if an error occurred; no_error otherwise) and optionally a non-zero rejection code if the webhook rejects the request with an HTTP status code (honored by the apiserver when the code is greater or equal to 400). Codes greater than 600 are truncated to 600, to keep the metrics cardinality bounded.
# TYPE apiserver_admission_webhook_rejection_count counter
apiserver_admission_webhook_rejection_count{error_type="calling_webhook_erro
```

```
r",name="always-timeout-
webhook.example.com",operation="CREATE",rejection_code="0",type="validatin
g"} 1
apiserver_admission_webhook_rejection_count{error_type="calling_webhook_erro
r",name="invalid-admission-response-
webhook.example.com",operation="CREATE",rejection_code="0",type="validatin
g"} 1
apiserver_admission_webhook_rejection_count{error_type="no_error",name="den
y-unwanted-configmap-
data.example.com",operation="CREATE",rejection_code="400",type="validating"}
13
```

最佳实践和警告

幂等性

幂等的修改性质的准入 Webhook 能够成功处理已经被它接纳甚或修改的对象。 即使多次执行该准入测试，也不会产生与初次执行结果相异的结果。

幂等 mutating admission Webhook 的示例：

1. 对于 CREATE Pod 请求，将 Pod 的字段 .spec.securityContext.runAsNonRoot 设置为 true，以实施安全最佳实践。
2. 对于 CREATE Pod 请求，如果未设置容器的字段 .spec.containers[].resources.limits，设置默认资源限制值。
3. 对于 CREATE pod 请求，如果 Pod 中不存在名为 foo-sidecar 的边车容器， 向 Pod 注入一个 foo-sidecar 容器。

在上述情况下，可以安全地重新调用 Webhook，或接受已经设置了字段的对象。

非幂等 mutating admission Webhook 的示例：

1. 对于 CREATE pod 请求，注入名称为 foo-sidecar 并带有当前时间戳的 边车容器（例如 foo-sidecar-19700101-000000）。
2. 对于 CREATE/UPDATE pod 请求，如果容器已设置标签 "env" 则拒绝， 否则将 "env": "prod" 标签添加到容器。
3. 对于 CREATE pod 请求，盲目地添加一个名为 foo-sidecar 的边车容器， 而未查看 Pod 中是否已经有 foo-sidecar 容器。

在上述第一种情况下，重新调用该 Webhook 可能导致同一个 Sidecar 容器 多次注入到 Pod 中，而且每次使用不同的容器名称。 类似地，如果 Sidecar 已存在于用户提供的 Pod 中，则 Webhook 可能注入重复的容器。

在上述第二种情况下，重新调用 Webhook 将导致 Webhook 自身输出失败。

在上述第三种情况下，重新调用 Webhook 将导致 Pod 规范中的容器重复， 从而使请求无效并被 API 服务器拒绝。

拦截对象的所有版本

建议通过将 .webhooks[].matchPolicy 设置为 Equivalent， 以确保准入 Webhooks 始终拦截对象的所有版本。 建议准入 Webhooks 应该更偏向注册资源的稳定版本。 如果无法拦截对象的所有版本，可能会导致准入策略未再某些版本的请求上执行。 有关示例，请参见匹配请求：matchPolicy。

可用性

建议准入 webhook 尽快完成执行（时长通常是毫秒级），因为它们会增加 API 请求的延迟。 建议对 Webhook 使用较小的超时值。有关更多详细信息，请参见超时。

建议 Admission Webhook 应该采用某种形式的负载均衡机制，以提供高可用性和高性能。 如果集群中正在运行 Webhook，则可以在服务后面运行多个 Webhook 后端，以利用该服务支持的负载均衡。

确保看到对象的最终状态

如果某准入 Webhook 需要保证自己能够看到对象的最终状态以实施策略， 则应该使用一个验证性质的 webhook， 因为可以通过 mutating Webhook 看到对象后对其进行修改。

例如，一个修改性质的准入Webhook 被配置为在每个 CREATE Pod 请求中 注入一个名称为 "foo-sidecar" 的 sidecar 容器。

如果*必须*存在边车容器，则还应配置一个验证性质的准入 Webhook 以拦截 CREATE Pod 请求，并验证要创建的对象中是否存在具有预期配置的名称为 "foo-sidecar" 的容器。

避免自托管的 Webhooks 中出现死锁

如果集群内的 Webhook 配置能够拦截启动其自己的 Pod 所需的资源， 则该 Webhook 可能导致其自身部署时发生死锁。

例如，某修改性质的准入 Webhook 配置为仅当 Pod 中设置了某个标签 （例如 "env": "prod"）时，才接受 CREATE Pod 请求。 Webhook 服务器在未设置 "env" 标签的 Deployment 中运行。当运行 Webhook 服务器的 容器的节点运行不正常时， Webhook 部署尝试将容器重新调度到另一个节点。 但是，由于未设置 "env" 标签，因此请求将被现有的 Webhook 服务器拒绝，并且调度迁移不会发生。

建议使用 namespaceSelector 排除 Webhook 所在的名字空间。

副作用

建议准入 Webhook 应尽可能避免副作用，这意味着该准入 webhook 仅对发送给他们的 AdmissionReview 的内容起作用，并且不要进行额外更改。 如果 Webhook 没有任何副作用，则 .webhooks[].sideEffects 字段应设置为 None。

如果在准入执行期间存在副作用，则应在处理 dryRun 为 true 的 AdmissionReview 对象时避免产生副作用，并且其 .webhooks[].sideEffects 字段应设置为 NoneOnDryRun 。更多详细信息，请参见副作用。

避免对 kube-system 名字空间进行操作

kube-system 名字空间包含由 Kubernetes 系统创建的对象， 例如用于控制平面组件的服务账号，诸如 kube-dns 之类的 Pod 等。 意外更改或拒绝 kube-system 名字空间中的请求可能会导致控制平面组件 停止运行或者导致未知行为发生。 如果你的准入 Webhook 不想修改 Kubernetes 控制平面的行为，请使用 namespaceSelector 避免拦截 kube-system 名字空间。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 04, 2020 at 6:22 PM PST: [zh] Fix links in zh localization (4) (abab517ff)

管理 Service Accounts

这是一篇针对服务账号的集群管理员指南。你应该熟悉 配置 Kubernetes 服务账号。

对鉴权和用户账号的支持已在规划中，当前并不完备。 为了更好地描述服务账号，有时这些不完善的特性也会被提及。

用户账号与服务账号

Kubernetes 区分用户账号和服务账号的概念，主要基于以下原因：

- 用户账号是针对人而言的。 服务账号是针对运行在 Pod 中的进程而言的。
- 用户账号是全局性的。其名称跨集群中名字空间唯一的。服务账号是名字空间作用域的。
- 通常情况下，集群的用户账号可能会从企业数据库进行同步，其创建需要特殊权限， 并且涉及到复杂的业务流程。 服务账号创建有意做得更轻量，允许集群用户为了具体的任务创建服务账号 以遵从权限最小化原则。
- 对人员和服务账号审计所考虑的因素可能不同。
- 针对复杂系统的配置包可能包含系统组件相关的各种服务账号的定义。因为服务账号 的创建约束不多并且有名字空间域的名称，这种配置是很轻量的。

服务账号的自动化

三个独立组件协作完成服务账号相关的自动化：

- ServiceAccount 准入控制器
- Token 控制器
- ServiceAccount 控制器

ServiceAccount 准入控制器

对 Pod 的改动通过一个被称为 准入控制器 的插件来实现。它是 API 服务器的一部分。当 Pod 被创建或更新时，它会同步地修改 Pod。 如果该插件处于激活状态（在大多数发行版中都是默认激活的），当 Pod 被创建 或更新时它会进行以下操作：

1. 如果该 Pod 没有设置 serviceAccountName，将其 serviceAccountName 设为 default。
2. 保证 Pod 所引用的 serviceAccountName 确实存在，否则拒绝该 Pod。
3. 如果 Pod 不包含 imagePullSecrets 设置，将 serviceAccountName 所引用 的服务账号中的 imagePullSecrets 信息添加到 Pod 中。
4. 如果服务账号的 automountServiceAccountToken 或 Pod 的 automountServiceAccountToken 都为设置为 false，则为 Pod 创建一个 volume，在其中包含用来访问 API 的令牌。
5. 如果前一步中为服务账号令牌创建了卷，则为 Pod 中的每个容器添加一个 volumeSource，挂载在其 /var/run/secrets/kubernetes.io/serviceaccount 目录下。

当 BoundServiceAccountTokenVolume 特性门控被启用时，你可以将服务账号卷迁移到投射卷。 服务账号令牌会在 1 小时后或者 Pod 被删除之后过期。 更多信息可参阅投射卷。

Token 控制器

TokenController 作为 kube-controller-manager 的一部分运行，以异步的形式工作。其职责包括：

- 监测 ServiceAccount 的创建并创建相应的服务账号令牌 Secret 以允许访问 API。
- 监测 ServiceAccount 的删除并删除所有相应的服务账号令牌 Secret。
- 监测服务账号令牌 Secret 的添加，保证相应的 ServiceAccount 存在，如有需要， 向 Secret 中添加令牌。
- 监测服务账号令牌 Secret 的删除，如有需要，从相应的 ServiceAccount 中移除引用。

你必须通过 --service-account-private-key-file 标志为 kube-controller-manager 的令牌控制器传入一个服务账号私钥文件。该私钥用于为所生成的服务账号令牌签名。 同样地，你需要通过 --service-account-key-file 标志将对应的公钥通知给 kube-apiserver。公钥用于在身份认证过程中校验令牌。

创建额外的 API 令牌

控制器中有专门的循环来保证每个 ServiceAccount 都存在对应的包含 API 令牌的 Secret。 当需要为 ServiceAccount 创建额外的 API 令牌时，可以创建一个类型为 kubernetes.io/service-account-token 的 Secret，并在其注解中引用对应的 ServiceAccount。控制器会生成令牌并更新该 Secret：

下面是这种 Secret 的一个示例配置：

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecretname
  annotations:
    kubernetes.io/service-account.name: myserviceaccount
type: kubernetes.io/service-account-token
```

```
kubectl create -f ./secret.json
kubectl describe secret mysecretname
```

删除/废止服务账号令牌 Secret

```
kubectl delete secret mysecretname
```

服务账号控制器

服务账号控制器管理各名字空间下的 ServiceAccount 对象，并且保证每个活跃的 名字空间下存在一个名为 "default" 的 ServiceAccount。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 28, 2020 at 10:57 PM PST: [zh] Sync English site changes for authentication ref (b8c9b2fa3)

鉴权概述

了解有关 Kubernetes 鉴权的更多信息，包括使用支持的鉴权模块创建策略的详细信息。

在 Kubernetes 中，你必须在鉴权（授予访问权限）之前进行身份验证（登录），有关身份验证的信息， 请参阅访问控制概述.

Kubernetes 期望请求中存在 REST API 常见的属性。 这意味着 Kubernetes 鉴权适用于现有的组织范围或云提供商范围的访问控制系统， 除了 Kubernetes API 之外，它还可以处理其他 API。

确定是允许还是拒绝请求

Kubernetes 使用 API 服务器对 API 请求进行鉴权。 它根据所有策略评估所有请求属性来决定允许或拒绝请求。 一个 API 请求的所有部分都必须被某些策略允许才能继续。这意味着默认情况下拒绝权限。

（尽管 Kubernetes 使用 API 服务器，但是依赖于特定对象种类的特定字段的访问控制和策略由准入控制器处理。 ）

当系统配置了多个鉴权模块时，Kubernetes 将按顺序使用每个模块。 如果任何鉴权模块批准或拒绝请求，则立即返回该决定，并且不会与其他鉴权模块协商。 如果所有模块对请求没有意见，则拒绝该请求。 被拒绝响应返回 HTTP 状态代码 403。

审查你的请求属性

Kubernetes 仅审查以下 API 请求属性：

- **用户** - 身份验证期间提供的 user 字符串。
- **组** - 经过身份验证的用户所属的组名列表。
- **额外信息** - 由身份验证层提供的任意字符串键到字符串值的映射。
- **API** - 指示请求是否针对 API 资源。
- **请求路径** - 各种非资源端点的路径，如 /api 或 /healthz。
- **API 请求动词** - API 动词 get、list、create、update、patch、watch、 proxy、redirect、delete 和 deletecollection 用于资源请求。 要确定资源 API 端点的请求动词，请参阅 确定请求动词。
- **HTTP 请求动词** - HTTP 动词 get、post、put 和 delete 用于非资源请求。
- **Resource** - 正在访问的资源的 ID 或名称（仅限资源请求）- 对于使用 get、update、patch 和 delete 动词的资源请求，你必须提供资源名称。
- **子资源** - 正在访问的子资源（仅限资源请求）。
- **名字空间** - 正在访问的对象的名称空间（仅适用于名字空间资源请求）。
- **API 组** - 正在访问的 API 组 （仅限资源请求）。空字符串表示核心 API 组。

确定请求动词

非资源请求

对于 /api/v1/... 或 /apis/<group>/<version>/... 之外的端点的请求被 视为"非资源请求（Non-Resource Requests）"，并使用该请求的 HTTP 方法的 小写形式作为其请求动词。 例如，对 /api 或 /healthz 这类端点的 GET 请求将使用 get 作为其动词。

资源请求

要确定对资源 API 端点的请求动词，需要查看所使用的 HTTP 动词以及该请求是针对单个资源还是一组资源：

HTTP 动词	请求动词
POST	create
GET, HEAD	get（针对单个资源）、list（针对集合）
PUT	update
PATCH	patch
DELETE	delete（针对单个资源）、deletecollection（针对集合）

Kubernetes 有时使用专门的动词以对额外的权限进行鉴权。例如：

- PodSecurityPolicy
  - policy API 组中 podsecuritypolicies 资源使用 use 动词
- RBAC
  - 对 rbac.authorization.k8s.io API 组中 roles 和 clusterroles 资源的 bind 和 escalate 动词
- 身份认证
  - 对核心 API 组中 users、groups 和 serviceaccounts 以及 authentication.k8s.io API 组中的 userextras 所使用的 impersonate 动词。

鉴权模块

- **Node** - 一个专用鉴权组件，根据调度到 kubelet 上运行的 Pod 为 kubelet 授予权限。 了解有关使用节点鉴权模式的更多信息，请参阅节点鉴权。
- **ABAC** - 基于属性的访问控制（ABAC）定义了一种访问控制范型，通过使用将属性组合 在一起的策略，将访问权限授予用户。策略可以使用任何类型的属性（用户属性、资源属性、 对象，环境属性等）。要了解有关使用 ABAC 模式的更多信息，请参阅 ABAC 模式。
- **RBAC** - 基于角色的访问控制（RBAC）是一种基于企业内个人用户的角色来管理对 计算机或网络资源的访问的方法。在此上下文中，权限是单个用户执行特定任务的能力， 例如查看、创建或修改文件。要了解有关使用 RBAC 模式的更多信息，请参阅 RBAC 模式。
  - 被启用之后，RBAC（基于角色的访问控制）使用 rbac.authorization.k8s.io API 组来 驱动鉴权决策，从而允许管理员通过 Kubernetes API 动态配置权限策略。
  - 要启用 RBAC，请使用 --authorization-mode = RBAC 启动 API 服务器。
- **Webhook** - WebHook 是一个 HTTP 回调：发生某些事情时调用的 HTTP POST； 通过 HTTP POST 进行简单的事件通知。实现 WebHook 的 Web 应用程序会在发生某些事情时 将消息发布到 URL。要了解有关使用 Webhook 模式的更多信息，请参阅 Webhook 模式。

检查 API 访问

kubectl 提供 auth can-i 子命令，用于快速查询 API 鉴权。 该命令使用 SelfSubjectAccessReview API 来确定当前用户是否可以执行给定操作， 无论使用何种鉴权模式该命令都可以工作。

```
kubectl auth can-i create deployments --namespace dev
```

```
yes
```

```
kubectl auth can-i create deployments --namespace prod
```

```
no
```

管理员可以将此与 用户扮演 结合使用，以确定其他用户可以执行的操作。

```
kubectl auth can-i list secrets --namespace dev --as dave
```

```
no
```

SelfSubjectAccessReview 是 authorization.k8s.io API 组的一部分，它将 API 服务器鉴权公开给外部服务。该组中的其他资源包括：

- SubjectAccessReview - 对任意用户的访问进行评估，而不仅仅是当前用户。 当鉴权决策被委派给 API 服务器时很有用。例如，kubelet 和扩展 API 服务器使用 它来确定用户对自己的 API 的访问权限。
- LocalSubjectAccessReview - 与 SubjectAccessReview 类似，但仅限于特定的 名字空间。
- SelfSubjectRulesReview - 返回用户可在名字空间内执行的操作集的审阅。 用户可以快速汇总自己的访问权限，或者用于 UI 中的隐藏/显示动作。

可以通过创建普通的 Kubernetes 资源来查询这些 API，其中返回对象的响应 "status" 字段是查询的结果。

```
kubectl create -f - -o yaml << EOF
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
spec:
  resourceAttributes:
    group: apps
    name: deployments
    verb: create
    namespace: dev
EOF
```

生成的 SelfSubjectAccessReview 为：

```
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
metadata:
  creationTimestamp: null
spec:
  resourceAttributes:
    group: apps
    name: deployments
    namespace: dev
    verb: create
status:
```

```
allowed: true
denied: false
```

为你的鉴权模块设置参数

你必须在策略中包含一个参数标志，以指明你的策略包含哪个鉴权模块：

可以使用的参数有：

- --authorization-mode=ABAC 基于属性的访问控制（ABAC）模式允许你 使用本地文件配置策略。
- --authorization-mode=RBAC 基于角色的访问控制（RBAC）模式允许你使用 Kubernetes API 创建和存储策略。
- --authorization-mode=Webhook WebHook 是一种 HTTP 回调模式，允许你使用远程 REST 端点管理鉴权。
- --authorization-mode=Node 节点鉴权是一种特殊用途的鉴权模式，专门对 kubelet 发出的 API 请求执行鉴权。
- --authorization-mode=AlwaysDeny 该标志阻止所有请求。仅将此标志用于测试。
- --authorization-mode=AlwaysAllow 此标志允许所有请求。仅在你不需要 API 请求 的鉴权时才使用此标志。

你可以选择多个鉴权模块。模块按顺序检查，以便较靠前的模块具有更高的优先级来允许或拒绝请求。

通过创建 Pod 提升权限

能够在名字空间中创建 Pod 的用户可能会提升其在该名字空间内的权限。 他们可以创建在该名字空间内访问其权限的 Pod。 他们可以创建 Pod 访问用户自己无法读取的 Secret，或者在具有不同/更高权限的 服务帐户下运行的 Pod 。

> **注意：**
>
> 系统管理员在授予对 Pod 创建的访问权限时要小心。 被授予在名字空间中创建 Pod（或创建 Pod 的控制器）的权限的用户可以： 读取名字空间中的所有 Secret；读取名字空间中的所有 ConfigMap； 并模拟名字空间中的任意服务账号并执行账号可以执行的任何操作。 无论采用何种鉴权方式，这都适用。

接下来

- 要了解有关身份验证的更多信息，请参阅 控制对 Kubernetes API 的访问 中的 **身份验证** 部分。
- 要了解有关准入控制的更多信息，请参阅 使用准入控制器。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 28, 2020 at 10:57 PM PST: [zh] Sync English site changes for authentication ref (b8c9b2fa3)

使用 RBAC 鉴权

基于角色（Role）的访问控制（RBAC）是一种基于组织中用户的角色来调节控制对 计算机或网络资源的访问的方法。

RBAC 鉴权机制使用 rbac.authorization.k8s.io API 组 来驱动鉴权决定，允许你通过 Kubernetes API 动态配置策略。

要启用 RBAC，在启动 API 服务器 时将 --authorization-mode 参数设置为一个逗号分隔的列表并确保其中包含 RBAC。

```
kube-apiserver --authorization-mode=Example,RBAC --<其他选项> --<其他选项>
```

API 对象

RBAC API 声明了四种 Kubernetes 对象：*Role*、*ClusterRole*、*RoleBinding* 和 *ClusterRoleBinding*。你可以像使用其他 Kubernetes 对象一样， 通过类似 kubectl 这类工具 描述对象, 或修补对象。

> **注意：**
>
> 这些对象在设计时即实施了一些访问限制。如果你在学习过程中对集群做了更改，请参考 避免特权提升和引导 一节，以了解这些限制会以怎样的方式阻止你做出修改。

Role 和 ClusterRole

RBAC 的 *Role* 或 *ClusterRole* 中包含一组代表相关权限的规则。 这些权限是纯粹累加的（不存在拒绝某操作的规则）。

Role 总是用来在某个名字空间 内设置访问权限；在你创建 Role 时，你必须指定该 Role 所属的名字空间。

与之相对，ClusterRole 则是一个集群作用域的资源。这两种资源的名字不同（Role 和 ClusterRole）是因为 Kubernetes 对象要么是名字空间作用域的，要么是集群作用域的， 不可两者兼具。

ClusterRole 有若干用法。你可以用它来：

1. 定义对某名字空间域对象的访问权限，并将在各个名字空间内完成授权；
2. 为名字空间作用域的对象设置访问权限，并跨所有名字空间执行授权；
3. 为集群作用域的资源定义访问权限。

如果你希望在名字空间内定义角色，应该使用 Role； 如果你希望定义集群范围的角色，应该使用 ClusterRole。

Role 示例

下面是一个位于 "default" 名字空间的 Role 的示例，可用来授予对 pods 的读访问权限：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" 标明 core API 组
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

ClusterRole 示例

ClusterRole 可以和 Role 相同完成授权。 因为 ClusterRole 属于集群范围，所以它也可以为以下资源授予访问权限：

- 集群范围资源（比如 节点（Node））
- 非资源端点（比如 `/healthz`）
- 跨名字空间访问的名字空间作用域的资源（如 Pods），比如，你可以使用 ClusterRole 来允许某特定用户执行 `kubectl get pods --all-namespaces`

下面是一个 ClusterRole 的示例，可用来为任一特定名字空间中的 Secret 授予读访问权限， 或者跨名字空间的访问权限（取决于该角色是如何绑定的）：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" 被忽略，因为 ClusterRoles 不受名字空间限制
  name: secret-reader
rules:
- apiGroups: [""]
  # 在 HTTP 层面，用来访问 Secret 对象的资源的名称为 "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

Role 或 ClusterRole 对象的名称必须是合法的 路径区段名称。

RoleBinding 和 ClusterRoleBinding

角色绑定（Role Binding）是将角色中定义的权限赋予一个或者一组用户。 它包含若干 **主体**（用户、组或服务账户）的列表和对这些主体所获得的角色的引用。 RoleBinding 在指定的名字空间中执行授权，而 ClusterRoleBinding 在集群范围执行授权。

一个 RoleBinding 可以引用同一的名字空间中的任何 Role。 或者，一个 RoleBinding 可以引用某 ClusterRole 并将该 ClusterRole 绑定到 RoleBinding 所在的名字空间。 如果你希望将某 ClusterRole 绑定到集群中所有名字空间，你要使用 ClusterRoleBinding。

RoleBinding 或 ClusterRoleBinding 对象的名称必须是合法的 路径区段名称。

RoleBinding 示例

下面的例子中的 RoleBinding 将 "pod-reader" Role 授予在 "default" 名字空间中的用户 "jane"。 这样，用户 "jane" 就具有了读取 "default" 名字空间中 pods 的权限。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# 此角色绑定允许 "jane" 读取 "default" 名字空间中的 Pods
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# 你可以指定不止一个"subject（主体）"
- kind: User
  name: jane # "name" 是不区分大小写的
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" 指定与某 Role 或 ClusterRole 的绑定关系
  kind: Role # 此字段必须是 Role 或 ClusterRole
  name: pod-reader      # 此字段必须与你要绑定的 Role 或 ClusterRole 的名称匹配
  apiGroup: rbac.authorization.k8s.io
```

RoleBinding 也可以引用 ClusterRole，以将对应 ClusterRole 中定义的访问权限授予 RoleBinding 所在名字空间的资源。这种引用使得你可以跨整个集群定义一组通用的角色， 之后在多个名字空间中复用。

例如，尽管下面的 RoleBinding 引用的是一个 ClusterRole，"dave"（这里的主体，不区分大小写）只能访问 "development" 名字空间中的 Secrets 对象，因为 RoleBinding 所在的名字空间（由其 metadata 决定）是 "development"。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# 此角色绑定使得用户 "dave" 能够读取 "default" 名字空间中的 Secrets
# 你需要一个名为 "secret-reader" 的 ClusterRole
kind: RoleBinding
metadata:
```

```
  name: read-secrets
  # RoleBinding 的名字空间决定了访问权限的授予范围。
  # 这里仅授权在 "development" 名字空间内的访问权限。
  namespace: development
subjects:
- kind: User
  name: dave # 'name' 是不区分大小写的
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

ClusterRoleBinding 示例

要跨整个集群完成访问权限的授予，你可以使用一个 ClusterRoleBinding。 下面的 ClusterRoleBinding 允许 "manager" 组内的所有用户访问任何名字空间中的 Secrets。

```
apiVersion: rbac.authorization.k8s.io/v1
# 此集群角色绑定允许 "manager" 组中的任何人访问任何名字空间中的 secrets
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # 'name' 是不区分大小写的
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

创建了绑定之后，你不能再修改绑定对象所引用的 Role 或 ClusterRole。 试图改变绑定对象的 roleRef 将导致合法性检查错误。 如果你想要改变现有绑定对象中 roleRef 字段的内容，必须删除重新创建绑定对象。

这种限制有两个主要原因：

1. 针对不同角色的绑定是完全不一样的绑定。要求通过删除/重建绑定来更改 roleRef，这样可以确保要赋予绑定的所有主体会被授予新的角色（而不是在允许修改 roleRef 的情况下导致所有现有主体未经验证即被授予新角色对应的权限）。
2. 将 roleRef 设置为不可以改变，这使得可以为用户授予对现有绑定对象的 update 权限， 这样可以让他们管理主体列表，同时不能更改被授予这些主体的角色。

命令 kubectl auth reconcile 可以创建或者更新包含 RBAC 对象的清单文件， 并且在必要的情况下删除和重新创建绑定对象，以改变所引用的角色。 更多相关信息请参照命令用法和示例

对资源的引用

在 Kubernetes API 中，大多数资源都是使用对象名称的字符串表示来呈现与访问的。例如，对于 Pod 应使用 "pods"。 RBAC 使用对应 API 端点的 URL 中呈现的名字来引用资源。 有一些 Kubernetes API 涉及 **子资源（subresource）**，例如 Pod 的日志。 对 Pod 日志的请求看起来像这样：

```
GET /api/v1/namespaces/{namespace}/pods/{name}/log
```

在这里，pods 对应名字空间作用域的 Pod 资源，而 log 是 pods 的子资源。 在 RBAC 角色表达子资源时，使用斜线（/）来分隔资源和子资源。 要允许某主体读取 pods 同时访问这些 Pod 的 log 子资源，你可以这么写：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-and-pod-logs-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list"]
```

对于某些请求，也可以通过 resourceNames 列表按名称引用资源。 在指定时，可以将请求限定为资源的单个实例。 下面的例子中限制可以 "get" 和 "update" 一个名为 my-configmap 的 ConfigMap：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: configmap-updater
rules:
- apiGroups: [""]
  # 在 HTTP 层面，用来访问 ConfigMap 的资源的名称为 "configmaps"
  resources: ["configmaps"]
  resourceNames: ["my-configmap"]
  verbs: ["update", "get"]
```

> **说明：**
>
> 你不能针对 create 或者 deletecollection 请求来实施 resourceName 限制。 对于 create 操作而言，这是因为在鉴权时还不知道对象名称。

聚合的 ClusterRole

你可以将若干 ClusterRole **聚合（Aggregate）** 起来，形成一个复合的 ClusterRole。 某个控制器作为集群控制面的一部分会监视带有 aggregationRule 的

ClusterRole 对象集合。aggregationRule 为控制器定义一个标签 选择算符供后者匹配应该组合到当前 ClusterRole 的 roles 字段中的 ClusterRole 对象。

下面是一个聚合 ClusterRole 的示例：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      rbac.example.com/aggregate-to-monitoring: "true"
rules: [] # 控制面自动填充这里的规则
```

如果你创建一个与某现有聚合 ClusterRole 的标签选择算符匹配的 ClusterRole， 这一变化会触发新的规则被添加到聚合 ClusterRole 的操作。 下面的例子中，通过创建一个标签同样为 rbac.example.com/aggregate-to-monitoring: true 的 ClusterRole，新的规则可被添加到 "monitoring" ClusterRole 中。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring-endpoints
  labels:
    rbac.example.com/aggregate-to-monitoring: "true"
# 当你创建 "monitoring-endpoints" ClusterRole 时，
# 下面的规则会被添加到 "monitoring" ClusterRole 中
rules:
- apiGroups: [""]
  resources: ["services", "endpoints", "pods"]
  verbs: ["get", "list", "watch"]
```

默认的面向用户的角色 使用 ClusterRole 聚合。 这使得作为集群管理员的你可以为扩展默认规则，包括为定制资源设置规则， 比如通过 CustomResourceDefinitions 或聚合 API 服务器提供的定制资源。

例如，下面的 ClusterRoles 让默认角色 "admin" 和 "edit" 拥有管理自定义资源 "CronTabs" 的权限， "view" 角色对 CronTab 资源拥有读操作权限。 你可以假定 CronTab 对象在 API 服务器所看到的 URL 中被命名为 "crontabs"。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: aggregate-cron-tabs-edit
  labels:
    # 添加以下权限到默认角色 "admin" 和 "edit" 中
    rbac.authorization.k8s.io/aggregate-to-admin: "true"
```

```yaml
    rbac.authorization.k8s.io/aggregate-to-edit: "true"
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: aggregate-cron-tabs-view
  labels:
    # 添加以下权限到 "view" 默认角色中
    rbac.authorization.k8s.io/aggregate-to-view: "true"
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch"]
```

Role 示例

以下示例均为从 Role 或 CLusterRole 对象中截取出来，我们仅展示其 `rules` 部分。

允许读取在核心 API 组下的 `"Pods"`：

```yaml
rules:
- apiGroups: [""]
  # 在 HTTP 层面，用来访问 Pod 的资源的名称为 "pods"
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

允许读/写在 "extensions" 和 "apps" API 组中的 Deployment（在 HTTP 层面，对应 URL 中资源部分为 "deployments"）：

```yaml
rules:
- apiGroups: ["extensions", "apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

允许读取核心 API 组中的 "pods" 和读/写 `"batch"` 或 `"extensions"` API 组中的 "jobs"：

```yaml
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch", "extensions"]
```

```
  resources: ["jobs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

允许读取名称为 "my-config" 的 ConfigMap（需要通过 RoleBinding 绑定以 限制为某名字空间中特定的 ConfigMap）：

```
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["my-config"]
  verbs: ["get"]
```

允许读取在核心组中的 "nodes" 资源（因为 Node 是集群作用域的，所以需要 ClusterRole 绑定到 ClusterRoleBinding 才生效）：

```
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "list", "watch"]
```

允许针对非资源端点 /healthz 和其子路径上发起 GET 和 POST 请求（必须在 ClusterRole 绑定 ClusterRoleBinding 才生效）：

```
rules:
  - nonResourceURLs: ["/healthz", "/healthz/*"] # nonResourceURL 中的 '*' 是一个全局通配符
    verbs: ["get", "post"]
```

对主体的引用

RoleBinding 或者 ClusterRoleBinding 可绑定角色到某 *主体（Subject）*上。 主体可以是组，用户或者 服务账户。

Kubernetes 用字符串来表示用户名。 用户名可以是普通的用户名，像 "alice"；或者是邮件风格的名称，如 "bob@example.com"， 或者是以字符串形式表达的数字 ID。 你作为 Kubernetes 管理员负责配置 身份认证模块 以便后者能够生成你所期望的格式的用户名。

> **注意：**
>
> 前缀 system: 是 Kubernetes 系统保留的，所以你要确保 所配置的用户名或者组名不能出现上述 system: 前缀。 除了对前缀的限制之外，RBAC 鉴权系统不对用户名格式作任何要求。

在 Kubernetes 中，鉴权模块提供用户组信息。 与用户名一样，用户组名也用字符串来表示，而且对该字符串没有格式要求， 只是不能使用保留的前缀 system:。

服务账户 的用户名前缀为 system:serviceaccount:，属于前缀为 system:serviceaccounts: 的用户组。

说明：

- system:serviceaccount:（单数）是用于服务账户用户名的前缀；
- system:serviceaccounts:（复数）是用于服务账户组名的前缀。

RoleBinding 示例

下面示例是 RoleBinding 中的片段，仅展示其 subjects 的部分。

对于名称为 alice@example.com 的用户：

```yaml
subjects:
- kind: User
  name: "alice@example.com"
  apiGroup: rbac.authorization.k8s.io
```

对于名称为 frontend-admins 的用户组：

```yaml
subjects:
- kind: Group
  name: "frontend-admins"
  apiGroup: rbac.authorization.k8s.io
```

对于 kube-system 名字空间中的默认服务账户：

```yaml
subjects:
- kind: ServiceAccount
  name: default
  namespace: kube-system
```

对于任何名称空间中的 "qa" 组中所有的服务账户：

```yaml
subjects:
- kind: Group
  name: system:serviceaccounts:qa
  apiGroup: rbac.authorization.k8s.io
```

对于 "dev" 名称空间中 "development" 组中的所有服务帐户：

```yaml
subjects:
- kind: Group
  name: system:serviceaccounts:dev
  apiGroup: rbac.authorization.k8s.io
  namespace: development
```

对于在任何名字空间中的服务账户：

```yaml
subjects:
- kind: Group
```

```
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

对于所有已经过认证的用户：

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

对于所有未通过认证的用户：

```
subjects:
- kind: Group
  name: system:unauthenticated
  apiGroup: rbac.authorization.k8s.io
```

对于所有用户：

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
- kind: Group
  name: system:unauthenticated
  apiGroup: rbac.authorization.k8s.io
```

默认 Roles 和 Role Bindings

API 服务器创建一组默认的 ClusterRole 和 ClusterRoleBinding 对象。 这其中许多是以 system: 为前缀的，用以标识对应资源是直接由集群控制面管理的。 所有的默认 ClusterRole 和 ClusterRoleBinding 都有 kubernetes.io/bootstrapping=rbac-defaults 标签。

> **注意：**
>
> 在修改名称包含 system: 前缀的 ClusterRole 和 ClusterRoleBinding 时要格外小心。 对这些资源的更改可能导致集群无法继续工作。

自动协商

在每次启动时，API 服务器都会更新默认 ClusterRole 以添加缺失的各种权限，并更新默认的 ClusterRoleBinding 以增加缺失的的各类主体。 这种自动协商机制允许集群去修复一些不小心发生的修改，并且有助于保证角色和角色绑定 在新的发行版本中有权限或主体变更时仍然保持最新。

如果要禁止此功能，请将默认 ClusterRole 以及 ClusterRoleBinding 的 rbac.authorization.kubernetes.io/autoupdate 注解设置成 false。 注意，缺少默认权限和角色绑定主体可能会导致集群无法正常工作。

如果基于 RBAC 的鉴权机制被启用，则自动协商功能默认是被启用的。

API 发现角色

无论是经过身份验证的还是未经过身份验证的用户，默认的角色绑定都授权他们读取被认为 是可安全地公开访问的 API（ 包括 CustomResourceDefinitions ）。 如果要禁用匿名的未经过身份验证的用户访问，请在 API 服务器配置中中添加 `--anonymous-auth=false` 的配置选项。

通过运行命令 kubectl 可以查看这些角色的配置信息:

```
kubectl get clusterroles system:discovery -o yaml
```

> **说明：**
>
> 如果你编辑该 ClusterRole，你所作的变更会被 API 服务器在重启时自动覆盖，这是通过 自动协商机制完成的。要避免这类覆盖操作， 要么不要手动编辑这些角色，要么禁止自动协商机制。

Kubernetes RBAC API 发现角色

默认 ClusterRole	默认 ClusterRoleBinding	描述
system:basic-user	**system:authenticated** 组	允许用户以只读的方式去访问他们自己的基本信息。在 1.14 版本之前，这个角色在默认情况下也绑定在 `system:unauthenticated` 上。
system:discovery	**system:authenticated** 组	允许以只读方式访问 API 发现端点，这些端点用来发现和协商 API 级别。 在 1.14 版本之前，这个角色在默认情况下绑定在 `system:unauthenticated` 上。
system:public-info-viewer	**system:authenticated** 和 **system:unauthenticated** 组	允许对集群的非敏感信息进行只读访问，它是在 1.14 版本中引入的。

面向用户的角色

一些默认的 ClusterRole 不是以前缀 `system:` 开头的。这些是面向用户的角色。 它们包括超级用户（ Super-User ）角色（ `cluster-admin` ）、 使用 ClusterRoleBinding 在集群范围内完成授权的角色（ `cluster-status` ）、 以及使用 RoleBinding 在特定名字空间中授予的角色（ `admin`、`edit`、`view` ）。

面向用户的 ClusterRole 使用 ClusterRole 聚合以允许管理员在 这些 ClusterRole 上添加用于定制资源的规则。如果想要添加规则到 admin、edit 或者 view， 可以创建带有以下一个或多个标签的 ClusterRole：

```yaml
metadata:
  labels:
    rbac.authorization.k8s.io/aggregate-to-admin: "true"
    rbac.authorization.k8s.io/aggregate-to-edit: "true"
    rbac.authorization.k8s.io/aggregate-to-view: "true"
```

默认 ClusterRole	默认 ClusterRoleBinding	描述
cluster-admin	**system:masters** 组	允许超级用户在平台上的任何资源上执行所有操作。 当在 **ClusterRoleBinding** 中使用时，可以授权对集群中以及所有名字空间中的全部资源进行完全控制。 当在 **RoleBinding** 中使用时，可以授权控制 RoleBinding 所在名字空间中的所有资源，包括名字空间本身。
admin	无	允许管理员访问权限，旨在使用 **RoleBinding** 在名字空间内执行授权。 如果在 **RoleBinding** 中使用，则可授予对名字空间中的大多数资源的读/写权限， 包括创建角色和角色绑定的能力。 但是它不允许对资源配额或者名字空间本身进行写操作。
edit	无	允许对名字空间的大多数对象进行读/写操作。它不允许查看或者修改角色或者角色绑定。 不过，此角色可以访问 Secret，以名字空间中任何 ServiceAccount 的身份运行 Pods，所以可以用来了解名字空间内所有服务账户的 API 访问级别。
view	无	允许对名字空间的大多数对象有只读权限。 它不允许查看角色或角色绑定。 此角色不允许查看 Secrets，因为读取 Secret 的内容意味着可以访问名字空间中 ServiceAccount 的凭据信息，进而允许利用名字空间中任何 ServiceAccount 的 身份访问 API（这是一种特权提升）。

核心组件角色

默认 ClusterRole	默认 ClusterRoleBinding	描述
system:kube-scheduler	**system:kube-scheduler** 用户	允许访问 scheduler 组件所需要的资源。

默认 ClusterRole	默认 ClusterRoleBinding	描述
system:volume-scheduler	**system:kube-scheduler** 用户	允许访问 kube-scheduler 组件所需要的卷资源。
system:kube-controller-manager	**system:kube-controller-manager** 用户	允许访问控制器管理器 组件所需要的资源。 各个控制回路所需要的权限在控制器角色 详述。
system:node	无	允许访问 kubelet 所需要的资源，**包括对所有 Secret 的读操作和对所有 Pod 状态对象的写操作。** 你应该使用 Node 鉴权组件 和 NodeRestriction 准入插件 而不是 `system:node` 角色。同时基于 kubelet 上调度执行的 Pod 来授权 kubelet 对 API 的访问。 `system:node` 角色的意义仅是为了与从 v1.8 之前版本升级而来的集群兼容。
system:node-proxier	**system:kube-proxy** 用户	允许访问 kube-proxy 组件所需要的资源。

其他组件角色

默认 ClusterRole	默认 ClusterRoleBinding	描述
system:auth-delegator	无	允许将身份认证和鉴权检查操作外包出去。 这种角色通常用在插件式 API 服务器上，以实现统一的身份认证和鉴权。
system:heapster	无	为 Heapster 组件（已弃用）定义的角色。
system:kube-aggregator	无	为 kube-aggregator 组件定义的角色。
system:kube-dns	在 **kube-system** 名字空间中的 **kube-dns** 服务账户	为 kube-dns 组件定义的角色。
system:kubelet-api-admin	无	允许 kubelet API 的完全访问权限。
system:node-bootstrapper	无	允许访问执行 kubelet TLS 启动引导 所需要的资源。
system:node-problem-detector	无	为 node-problem-detector 组件定义的角色。
system:persistent-volume-provisioner	无	允许访问大部分 动态卷驱动 所需要的资源。

默认 ClusterRole	默认 ClusterRoleBinding	描述
system:monitoring	**system:monitoring** 组	允许对控制平面监控端点的读取访问（例如：kube-apiserver 存活和就绪端点（/healthz、/livez、/readyz），各个健康检查端点（/healthz/*、/livez/* 和 /readyz/*）和 /metrics）。 请注意，各个运行状况检查端点和度量标准端点可能会公开敏感信息。

内置控制器的角色

Kubernetes 控制器管理器 运行内建于 Kubernetes 控制面的控制器。 当使用 --use-service-account-credentials 参数启动时，kube-controller-manager 使用单独的服务账户来启动每个控制器。 每个内置控制器都有相应的、前缀为 system:controller: 的角色。 如果控制管理器启动时未设置 --use-service-account-credentials，它使用自己的身份凭据来运行所有的控制器，该身份必须被授予所有相关的角色。 这些角色包括:

- system:controller:attachdetach-controller
- system:controller:certificate-controller
- system:controller:clusterrole-aggregation-controller
- system:controller:cronjob-controller
- system:controller:daemon-set-controller
- system:controller:deployment-controller
- system:controller:disruption-controller
- system:controller:endpoint-controller
- system:controller:expand-controller
- system:controller:generic-garbage-collector
- system:controller:horizontal-pod-autoscaler
- system:controller:job-controller
- system:controller:namespace-controller
- system:controller:node-controller
- system:controller:persistent-volume-binder
- system:controller:pod-garbage-collector
- system:controller:pv-protection-controller
- system:controller:pvc-protection-controller
- system:controller:replicaset-controller
- system:controller:replication-controller
- system:controller:resourcequota-controller
- system:controller:root-ca-cert-publisher
- system:controller:route-controller
- system:controller:service-account-controller
- system:controller:service-controller
- system:controller:statefulset-controller
- system:controller:ttl-controller

初始化与预防权限提升

RBAC API 会阻止用户通过编辑角色或者角色绑定来提升权限。 由于这一点是在 API 级别实现的，所以在 RBAC 鉴权组件未启用的状态下依然可以正常工作。

对角色创建或更新的限制

只有在符合下列条件之一的情况下，你才能创建/更新角色:

1. 你已经拥有角色中包含的所有权限，且其作用域与正被修改的对象作用域相同。（对 ClusterRole 而言意味着集群范围，对 Role 而言意味着相同名字空间或者集群范围）。
2. 你被显式授权在 rbac.authorization.k8s.io API 组中的 roles 或 clusterroles 资源使用 escalate 动词。

例如，如果 user-1 没有列举集群范围所有 Secret 的权限，他将不能创建包含该权限的 ClusterRole。 若要允许用户创建/更新角色：

1. 根据需要赋予他们一个角色，允许他们根据需要创建/更新 Role 或者 ClusterRole 对象。
2. 授予他们在所创建/更新角色中包含特殊权限的权限:
    - 隐式地为他们授权（如果它们试图创建或者更改 Role 或 ClusterRole 的权限， 但自身没有被授予相应权限，API 请求将被禁止）。
    - 通过允许他们在 Role 或 ClusterRole 资源上执行 escalate 动作显式完成授权。 这里的 roles 和 clusterroles 资源包含在 rbac.authorization.k8s.io API 组中。

对角色绑定创建或更新的限制

只有你已经具有了所引用的角色中包含的全部权限时，或者你被授权在所引用的角色上执行 bind 动词时，你才可以创建或更新角色绑定。这里的权限与角色绑定的作用域相同。例如，如果用户 user-1 没有列举集群范围所有 Secret 的能力，则他不可以创建 ClusterRoleBinding 引用授予该许可权限的角色。 如要允许用户创建或更新角色绑定：

1. 赋予他们一个角色，使得他们能够根据需要创建或更新 RoleBinding 或 ClusterRoleBinding 对象。
2. 授予他们绑定某特定角色所需要的许可权限：
    - 隐式授权下，可以将角色中包含的许可权限授予他们；
    - 显式授权下，可以授权他们在特定 Role （或 ClusterRole）上执行 bind 动词的权限。

例如，下面的 ClusterRole 和 RoleBinding 将允许用户 user-1 把名字空间 user-1-namespace 中的 admin、edit 和 view 角色赋予其他用户：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: role-grantor
```

```yaml
rules:
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["rolebindings"]
  verbs: ["create"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["clusterroles"]
  verbs: ["bind"]
  # 忽略 resourceNames 意味着允许绑定任何 ClusterRole
  resourceNames: ["admin","edit","view"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: role-grantor-binding
  namespace: user-1-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: role-grantor
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: user-1
```

当启动引导第一个角色和角色绑定时，需要为初始用户授予他们尚未拥有的权限。 对初始角色和角色绑定进行初始化时需要：

- 使用用户组为 system:masters 的凭据，该用户组由默认绑定关联到 cluster-admin 这个超级用户角色。
- 如果你的 API 服务器启动时启用了不安全端口（使用 --insecure-port），你也可以通过 该端口调用 API ，这样的操作会绕过身份验证或鉴权。

一些命令行工具

kubectl create role

创建 Role 对象，定义在某一名字空间中的权限。例如:

- 创建名称为 "pod-reader" 的 Role 对象，允许用户对 Pods 执行 get、watch 和 list 操作：

  ```
  kubectl create role pod-reader --verb=get --verb=list --verb=watch --resource=pods
  ```

- 创建名称为 "pod-reader" 的 Role 对象并指定 resourceNames：

```
kubectl create role pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod
```

- 创建名为 "foo" 的 Role 对象并指定 apiGroups：

```
kubectl create role foo --verb=get,list,watch --resource=replicasets.apps
```

- 创建名为 "foo" 的 Role 对象并指定子资源权限:

```
kubectl create role foo --verb=get,list,watch --resource=pods,pods/status
```

- 创建名为 "my-component-lease-holder" 的 Role 对象，使其具有对特定名称的资源执行 get/update 的权限：

```
kubectl create role my-component-lease-holder --verb=get,list,watch,update --resource=lease --resource-name=my-component
```

kubectl create clusterrole

创建 ClusterRole 对象。例如：

- 创建名称为 "pod-reader" 的 ClusterRole对象，允许用户对 Pods 对象执行 get、watch和list` 操作：

```
kubectl create clusterrole pod-reader --verb=get,list,watch --resource=pods
```

- 创建名为 "pod-reader" 的 ClusterRole 对象并指定 resourceNames：

```
kubectl create clusterrole pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod
```

- 创建名为 "foo" 的 ClusterRole 对象并指定 apiGroups：

```
kubectl create clusterrole foo --verb=get,list,watch --resource=replicasets.apps
```

- 创建名为 "foo" 的 ClusterRole 对象并指定子资源:

```
kubectl create clusterrole foo --verb=get,list,watch --resource=pods,pods/status
```

- 创建名为 "foo" 的 ClusterRole 对象并指定 nonResourceURL：

```
kubectl create clusterrole "foo" --verb=get --non-resource-url=/logs/*
```

- 创建名为 "monitoring" 的 ClusterRole 对象并指定 aggregationRule：

```
kubectl create clusterrole monitoring --aggregation-rule="rbac.example.com/aggregate-to-monitoring=true"
```

kubectl create rolebinding

在特定的名字空间中对 Role 或 ClusterRole 授权。例如：

- 在名字空间 "acme" 中，将名为 admin 的 ClusterRole 中的权限授予名称 "bob" 的用户:

  ```
  kubectl create rolebinding bob-admin-binding --clusterrole=admin --user=bob --namespace=acme
  ```

- 在名字空间 "acme" 中，将名为 view 的 ClusterRole 中的权限授予名字空间 "acme" 中名为 myapp 的服务账户：

  ```
  kubectl create rolebinding myapp-view-binding --clusterrole=view --serviceaccount=acme:myapp --namespace=acme
  ```

- 在名字空间 "acme" 中，将名为 view 的 ClusterRole 对象中的权限授予名字空间 "myappnamespace" 中名称为 myapp 的服务账户：

  ```
  kubectl create rolebinding myappnamespace-myapp-view-binding --clusterrole=view --serviceaccount=myappnamespace:myapp --namespace=acme
  ```

kubectl create clusterrolebinding

在整个集群（所有名字空间）中用 ClusterRole 授权。例如：

- 在整个集群范围，将名为 cluster-admin 的 ClusterRole 中定义的权限授予名为 "root" 用户：

  ```
  kubectl create clusterrolebinding root-cluster-admin-binding --clusterrole=cluster-admin --user=root
  ```

- 在整个集群范围内，将名为 system:node-proxier 的 ClusterRole 的权限授予名为 "system:kube-proxy" 的用户：

  ```
  kubectl create clusterrolebinding kube-proxy-binding --clusterrole=system:node-proxier --user=system:kube-proxy
  ```

- 在整个集群范围内，将名为 view 的 ClusterRole 中定义的权限授予 "acme" 名字空间中 名为 "myapp" 的服务账户：

  ```
  kubectl create clusterrolebinding myapp-view-binding --clusterrole=view --serviceaccount=acme:myapp
  ```

kubectl auth reconcile

使用清单文件来创建或者更新 rbac.authorization.k8s.io/v1 API 对象。

尚不存在的对象会被创建，如果对应的名字空间也不存在，必要的话也会被创建。 已经存在的角色会被更新，使之包含输入对象中所给的权限。如果指定了 --remove-extra-permissions，可以删除额外的权限。

已经存在的绑定也会被更新，使之包含输入对象中所给的主体。如果指定了 --remove-extra-permissions，则可以删除多余的主体。

例如:

- 测试应用 RBAC 对象的清单文件，显示将要进行的更改：

  ```
  kubectl auth reconcile -f my-rbac-rules.yaml --dry-run
  ```

- 应用 RBAC 对象的清单文件，保留角色中的额外权限和绑定中的其他主体：

  ```
  kubectl auth reconcile -f my-rbac-rules.yaml
  ```

- 应用 RBAC 对象的清单文件, 删除角色中的额外权限和绑定中的其他主体：

  ```
  kubectl auth reconcile -f my-rbac-rules.yaml --remove-extra-subjects --remove-extra-permissions
  ```

查看 CLI 帮助获取详细的用法。

服务账户权限

默认的 RBAC 策略为控制面组件、节点和控制器授予权限。 但是不会对 kube-system 名字空间之外的服务账户授予权限。 （除了授予所有已认证用户的发现权限）

这使得你可以根据需要向特定服务账户授予特定权限。 细粒度的角色绑定可带来更好的安全性，但需要更多精力管理。 粗粒度的授权可能导致服务账户被授予不必要的 API 访问权限（甚至导致潜在的权限提升）， 但更易于管理。

按从最安全到最不安全的顺序，存在以下方法：

1. 为特定应用的服务账户授予角色（最佳实践）

   这要求应用在其 Pod 规约中指定 serviceAccountName， 并额外创建服务账户（包括通过 API、应用程序清单、kubectl create serviceaccount 等）。

   例如，在名字空间 "my-namespace" 中授予服务账户 "my-sa" 只读权限：

   ```
   kubectl create rolebinding my-sa-view \
     --clusterrole=view \
     --serviceaccount=my-namespace:my-sa \
     --namespace=my-namespace
   ```

1. 将角色授予某名字空间中的 "default" 服务账户

   如果某应用没有指定 serviceAccountName，那么它将使用 "default" 服务账户。

> **说明：** "default" 服务账户所具有的权限会被授予给名字空间中所有未指定 serviceAccountName 的 Pod。

例如，在名字空间 "my-namespace" 中授予服务账户 "default" 只读权限：

```
kubectl create rolebinding default-view \
  --clusterrole=view \
  --serviceaccount=my-namespace:default \
  --namespace=my-namespace
```

许多插件组件 在 kube-system 名字空间以 "default" 服务账户运行。 要允许这些插件组件以超级用户权限运行，需要将集群的 cluster-admin 权限授予 kube-system 名字空间中的 "default" 服务账户。

> **说明：** 启用这一配置意味着在 kube-system 名字空间中包含以超级用户账号来访问 API 的 Secrets。

```
kubectl create clusterrolebinding add-on-cluster-admin \
  --clusterrole=cluster-admin \
  --serviceaccount=kube-system:default
```

1. 将角色授予名字空间中所有服务账户

   如果你想要名字空间中所有应用都具有某角色，无论它们使用的什么服务账户， 可以将角色授予该名字空间的服务账户组。

   例如，在名字空间 "my-namespace" 中的只读权限授予该名字空间中的所有服务账户：

   ```
   kubectl create rolebinding serviceaccounts-view \
     --clusterrole=view \
     --group=system:serviceaccounts:my-namespace \
     --namespace=my-namespace
   ```

1. 在集群范围内为所有服务账户授予一个受限角色（不鼓励）

   如果你不想管理每一个名字空间的权限，你可以向所有的服务账户授予集群范围的角色。

   例如，为集群范围的所有服务账户授予跨所有名字空间的只读权限：

   ```
   kubectl create clusterrolebinding serviceaccounts-view \
     --clusterrole=view \
     --group=system:serviceaccounts
   ```

1. 授予超级用户访问权限给集群范围内的所有服务帐户（强烈不鼓励）

   如果你不关心如何区分权限，你可以将超级用户访问权限授予所有服务账户。

> **警告：** 这样做会允许所有应用都对你的集群拥有完全的访问权限，并将允许所有能够读取 Secret（或创建 Pod）的用户对你的集群有完全的访问权限。

```
kubectl create clusterrolebinding serviceaccounts-cluster-admin \
  --clusterrole=cluster-admin \
  --group=system:serviceaccounts
```

从 ABAC 升级

原来运行较老版本 Kubernetes 的集群通常会使用限制宽松的 ABAC 策略， 包括授予所有服务帐户全权访问 API 的能力。

默认的 RBAC 策略为控制面组件、节点和控制器等授予有限的权限，但不会为 kube-system 名字空间外的服务账户授权 （除了授予所有认证用户的发现权限之外）。

这样做虽然安全得多，但可能会干扰期望自动获得 API 权限的现有工作负载。 这里有两种方法来完成这种转换:

并行鉴权

同时运行 RBAC 和 ABAC 鉴权模式, 并指定包含 现有的 ABAC 策略 的策略文件：

```
--authorization-mode=RBAC,ABAC --authorization-policy-file=mypolicy.json
```

关于命令行中的第一个选项：如果早期的鉴权组件，例如 Node，拒绝了某个请求，则 RBAC 鉴权组件尝试对该 API 请求鉴权。如果 RBAC 也拒绝了该 API 请求，则运行 ABAC 鉴权组件。这意味着被 RBAC 或 ABAC 策略所允许的任何请求都是被允许的请求。

如果 API 服务器启动时，RBAC 组件的日志级别为 5 或更高（--vmodule=rbac*=5 或 --v=5）， 你可以在 API 服务器的日志中看到 RBAC 的细节 （前缀 RBAC:）你可以使用这些信息来确定需要将哪些角色授予哪些用户、组或服务帐户。

一旦你将角色授予服务账户 ，工作负载运行时 在服务器日志中没有出现 RBAC 拒绝消息，就可以删除 ABAC 鉴权器。

宽松的 RBAC 权限

你可以使用 RBAC 角色绑定在多个场合使用宽松的策略。

> **警告：**
>
> 下面的策略允许 **所有** 服务帐户充当集群管理员。 容器中运行的所有应用程序都会自动收到服务帐户的凭据，可以对 API 执行任何操作， 包括查看 Secrets 和修改权限。这一策略是不被推荐的。

```
kubectl create clusterrolebinding permissive-binding \
  --clusterrole=cluster-admin \
```

```
--user=admin \
--user=kubelet \
--group=system:serviceaccounts
```

在你完成到 RBAC 的迁移后，应该调整集群的访问控制，确保相关的策略满足你的 信息安全需求。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 20, 2021 at 4:23 PM PST: Fix Typo in rbac.md (f84a0898e)

使用 Node 鉴权

节点鉴权是一种特殊用途的鉴权模式，专门对 kubelet 发出的 API 请求进行鉴权。

概述

节点鉴权器允许 kubelet 执行 API 操作。包括：

读取操作：

- services
- endpoints
- nodes
- pods
- secrets、configmaps、pvcs 以及绑定到 kubelet 节点的与 pod 相关的持久卷

写入操作：

- 节点和节点状态（启用 NodeRestriction 准入插件以限制 kubelet 只能修改自己的节点）
- Pod 和 Pod 状态 (启用 NodeRestriction 准入插件以限制 kubelet 只能修改绑定到自身的 Pod)
- 事件

鉴权相关操作：

- 对于基于 TLS 的启动引导过程时使用的 certificationsigningrequests API 的读/写权限
- 为委派的身份验证/授权检查创建 tokenreviews 和 subjectaccessreviews 的能力

在将来的版本中，节点鉴权器可能会添加或删除权限，以确保 kubelet 具有正确操作所需的最小权限集。

为了获得节点鉴权器的授权，kubelet 必须使用一个凭证以表示它在 system:nodes 组中，用户名为 system:node:<nodeName>。 上述的组名和用户名格式要与 kubelet TLS 启动引导过程中为每个 kubelet 创建的标识相匹配。

要启用节点授权器，请使用 --authorization-mode = Node 启动 apiserver。

要限制 kubelet 具有写入权限的 API 对象，请使用 --enable-admission-plugins=...,NodeRestriction,... 启动 apiserver，从而启用 NodeRestriction 准入插件。

迁移考虑因素

在 system:nodes 组之外的 Kubelet

system:nodes 组之外的 kubelet 不会被 Node 鉴权模式授权，并且需要继续通过当前授权它们的机制来授权。 节点准入插件不会限制来自这些 kubelet 的请求。

具有无差别用户名的 Kubelet

在一些部署中，kubelet 具有 system:nodes 组的凭证，但是无法给出它们所关联的节点的标识，因为它们没有 system:node:... 格式的用户名。 这些 kubelet 不会被 Node 授权模式授权，并且需要继续通过当前授权它们的任何机制来授权。

因为默认的节点标识符实现不会把它当作节点身份标识，NodeRestriction 准入插件会忽略来自这些 kubelet 的请求。

相对于以前使用 RBAC 的版本的更新

升级的 1.7 之前的使用 RBAC 的集群将继续按原样运行，因为 system:nodes 组绑定已经存在。

如果集群管理员希望开始使用 Node 鉴权器和 NodeRestriction 准入插件来限制节点对 API 的访问，这一需求可以通过下列操作来完成且不会影响已部署的应用：

1. 启用 Node 鉴权模式 (--authorization-mode=Node,RBAC) 和 NodeRestriction 准入插件
2. 确保所有 kubelet 的凭据符合组/用户名要求
3. 审核 apiserver 日志以确保 Node 鉴权器不会拒绝来自 kubelet 的请求（日志中没有持续的 NODE DENY 消息）
4. 删除 system:node 集群角色绑定

RBAC 节点权限

在 1.6 版本中，当使用 RBAC 鉴权模式 时，system:nodes 集群角色会被自动绑定到 system:node 组。

在 1.7 版本中，不再推荐将 system:nodes 组自动绑定到 system:node 角色，因为节点鉴权器通过对 secret 和 configmap 访问的额外限制完成了相同的任务。 如果同时启用了 Node 和 RBAC 授权模式，1.7 版本则不会创建 system:nodes 组到 system:node 角色的自动绑定。

在 1.8 版本中，绑定将根本不会被创建。

使用 RBAC 时，将继续创建 system:node 集群角色，以便与将其他用户或组绑定到该角色的部署方法兼容。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 September 26, 2020 at 7:37 PM PST: Update more links (a8542542a)

Webhook 模式

WebHook 是一种 HTTP 回调：某些条件下触发的 HTTP POST 请求；通过 HTTP POST 发送的简单事件通知。一个基于 web 应用实现的 WebHook 会在特定事件发生时把消息发送给特定的 URL。

具体来说，当在判断用户权限时，Webhook 模式会使 Kubernetes 查询外部的 REST 服务。

配置文件格式

Webhook 模式需要一个 HTTP 配置文件，通过 --authorization-webhook-config-file=SOME_FILENAME 的参数声明。

配置文件的格式使用 kubeconfig。在文件中，"users" 代表着 API 服务器的 webhook，而 "cluster" 代表着远程服务。

使用 HTTPS 客户端认证的配置例子：

```yaml
# Kubernetes API 版本
apiVersion: v1
# API 对象种类
kind: Config
# clusters 代表远程服务。
clusters:
  - name: name-of-remote-authz-service
```

```yaml
    cluster:
      # 对远程服务进行身份认证的 CA。
      certificate-authority: /path/to/ca.pem
      # 远程服务的查询 URL。必须使用 'https'。
      server: https://authz.example.com/authorize

# users 代表 API 服务器的 webhook 配置
users:
  - name: name-of-api-server
    user:
      client-certificate: /path/to/cert.pem # webhook plugin 使用 cert
      client-key: /path/to/key.pem          # cert 所对应的 key

# kubeconfig 文件必须有 context。需要提供一个给 API 服务器。
current-context: webhook
contexts:
- context:
    cluster: name-of-remote-authz-service
    user: name-of-api-server
  name: webhook
```

请求载荷

在做认证决策时，API 服务器会 POST 一个 JSON 序列化的 authorization.k8s.io/v1beta1 SubjectAccessReview 对象来描述这个动作。这个对象包含了描述用户请求的字段，同时也包含了需要被访问资源或请求特征的具体信息。

需要注意的是 webhook API 对象与其他 Kubernetes API 对象一样都同样都服从版本兼容规则。实施人员应该了解 beta 对象的更宽松的兼容性承诺，同时确认请求的 "apiVersion" 字段能被正确地反序列化。此外，API 服务器还必须启用 authorization.k8s.io/v1beta1 API 扩展组 (--runtime-config=authorization.k8s.io/v1beta1=true)。

一个请求内容的例子：

```yaml
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "kittensandponies",
      "verb": "get",
      "group": "unicorn.example.org",
      "resource": "pods"
    },
    "user": "jane",
    "group": [
```

```
    "group1",
    "group2"
  ]
 }
}
```

期待远程服务填充请求的 status 字段并响应允许或禁止访问。响应主体的 spec 字段被忽略，可以省略。允许的响应将返回:

```
{
 "apiVersion": "authorization.k8s.io/v1beta1",
 "kind": "SubjectAccessReview",
 "status": {
  "allowed": true
 }
}
```

为了禁止访问，有两种方法。

在大多数情况下，第一种方法是首选方法，它指示授权 webhook 不允许或对请求"无意见"，但是，如果配置了其他授权者，则可以给他们机会允许请求。如果没有其他授权者，或者没有一个授权者，则该请求被禁止。webhook 将返回:

```
{
 "apiVersion": "authorization.k8s.io/v1beta1",
 "kind": "SubjectAccessReview",
 "status": {
  "allowed": false,
  "reason": "user does not have read access to the namespace"
 }
}
```

第二种方法立即拒绝其他配置的授权者进行短路评估。仅应由对集群的完整授权者配置有详细了解的 webhook 使用。webhook 将返回:

```
{
 "apiVersion": "authorization.k8s.io/v1beta1",
 "kind": "SubjectAccessReview",
 "status": {
  "allowed": false,
  "denied": true,
  "reason": "user does not have read access to the namespace"
 }
}
```

对于非资源的路径访问是这么发送的:

```
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "nonResourceAttributes": {
      "path": "/debug",
      "verb": "get"
    },
    "user": "jane",
    "group": [
      "group1",
      "group2"
    ]
  }
}
```

非资源类的路径包括：/api, /apis, /metrics, /resetMetrics, /logs, /debug, /healthz, /swagger-ui/, /swaggerapi/, /ui, 和 /version。客户端需要访问 /api, /api/*, /apis, /apis/*, 和 /version 以便 能发现服务器上有什么资源和版本。对于其他非资源类的路径访问在没有 REST API 访问限制的情况下拒绝。

更多信息可以参考 authorization.v1beta1 API 对象和webhook.go。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 September 26, 2020 at 7:37 PM PST: Update more links (a8542542a)

使用 ABAC 鉴权

基于属性的访问控制（Attribute-based access control - ABAC）定义了访问控制范例，其中通过使用将属性组合在一起的策略来向用户授予访问权限。

策略文件格式

基于 ABAC 模式，可以这样指定策略文件 --authorization-policy-file=SOME_FILENAME。

此文件格式是 JSON Lines，不应存在封闭的列表或映射，每行一个映射。

每一行都是一个策略对象，策略对象是具有以下属性的映射：

- 版本控制属性：
  - apiVersion，字符串类型：有效值为abac.authorization.kubernetes.io/v1beta1，允许对策略格式进行版本控制和转换。
  - kind，字符串类型：有效值为 Policy，允许对策略格式进行版本控制和转换。
- spec 配置为具有以下映射的属性：
  - 主体匹配属性：
    - user，字符串类型；来自 --token-auth-file 的用户字符串，如果你指定 user，它必须与验证用户的用户名匹配。
    - group，字符串类型；如果指定 group，它必须与经过身份验证的用户的一个组匹配，system:authenticated匹配所有经过身份验证的请求。system:unauthenticated匹配所有未经过身份验证的请求。
- 资源匹配属性：
  - apiGroup，字符串类型；一个 API 组。
    - 例： extensions
    - 通配符：*匹配所有 API 组。
  - namespace，字符串类型；一个命名空间。
    - 例如：kube-system
    - 通配符：*匹配所有资源请求。
  - resource，字符串类型；资源类型。
    - 例：pods
    - 通配符：*匹配所有资源请求。
- 非资源匹配属性：
  - nonResourcePath，字符串类型；非资源请求路径。
    - 例如：/version或 /apis
    - 通配符：
      - * 匹配所有非资源请求。
      - /foo/* 匹配 /foo/ 的所有子路径。
- readonly，键入布尔值，如果为 true，则表示该策略仅适用于 get、list 和 watch 操作。

  说明：

  属性未设置等效于属性被设置为对应类型的零值(例如空字符串、0、false)，然而，出于可读性考虑，应尽量选择不设置这类属性。

  在将来，策略可能以 JSON 格式表示，并通过 REST 界面进行管理。

鉴权算法

请求具有与策略对象的属性对应的属性。

当接收到请求时，确定属性。未知属性设置为其类型的零值（例如：空字符串，0，false）。

设置为 "*" 的属性将匹配相应属性的任何值。

检查属性的元组，以匹配策略文件中的每个策略。如果至少有一行匹配请求属性，则请求被鉴权（但仍可能无法通过稍后的合法性检查）。

要允许任何经过身份验证的用户执行某些操作，请将策略组属性设置为 "system:authenticated"。

要允许任何未经身份验证的用户执行某些操作，请将策略组属性设置为 "system:authentication"。

要允许用户执行任何操作，请使用 apiGroup，命名空间， 资源和 nonResourcePath 属性设置为 "*" 的策略。

要允许用户执行任何操作，请使用设置为 "*" 的 apiGroup，namespace，resource 和 nonResourcePath 属性编写策略。

Kubectl

Kubectl 使用 api-server 的 /api 和 /apis 端点来发现服务资源类型，并使用位于 /openapi/v2 的模式信息来验证通过创建/更新操作发送到 API 的对象。

当使用 ABAC 鉴权时，这些特殊资源必须显式地通过策略中的 nonResourcePath 属性暴露出来（参见下面的 示例）：

- /api，/api/*，/apis和 /apis/* 用于 API 版本协商。
- /version 通过 kubectl version 检索服务器版本。
- /swaggerapi/* 用于创建 / 更新操作。

要检查涉及到特定 kubectl 操作的 HTTP 调用，您可以调整详细程度： kubectl --v=8 version

例子

1. Alice 可以对所有资源做任何事情：

   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "alice", "namespace": "*", "resource": "*", "apiGroup": "*"}}
   ```

2. Kubelet 可以读取任何 pod：

   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "pods", "readonly": true}}
   ```

3. Kubelet 可以读写事件：

   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "events"}}
   ```

1. Bob 可以在命名空间 projectCaribou 中读取 pod：

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "bob", "namespace": "projectCaribou", "resource": "pods", "readonly": true}}
```

2. 任何人都可以对所有非资源路径进行只读请求：

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:authenticated", "readonly": true, "nonResourcePath": "*"}}
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:unauthenticated", "readonly": true, "nonResourcePath": "*"}}
```

完整文件示例

服务帐户的快速说明

服务帐户自动生成用户。用户名是根据命名约定生成的：

```
system:serviceaccount:<namespace>:<serviceaccountname>
```

创建新的命名空间也会导致创建一个新的服务帐户：

```
system:serviceaccount:<namespace>:default
```

例如，如果要将 API 的 kube-system 完整权限中的默认服务帐户授予，则可以将此行添加到策略文件中：

```
{"apiVersion":"abac.authorization.kubernetes.io/v1beta1","kind":"Policy","spec":{"user":"system:serviceaccount:kube-system:default","namespace":"*","resource":"*","apiGroup":"*"}}
```

需要重新启动 apiserver 以获取新的策略行。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 September 26, 2020 at 7:37 PM PST: Update more links (a8542542a)

API 参考

v1.20

Kubernetes API v1.20

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 28, 2020 at 3:00 AM PST: versioning... (8ebfea611)

知名标签（Label）、注解（Annotation）和 污点（Taint）

Kubernetes 保留了 kubernetes.io 命名空间下的所有标签和注解。

本文档提供这些标签、注解和污点的参考，也可用来协调对这类标签、注解和污点的设置。

kubernetes.io/arch

示例：kubernetes.io/arch=amd64

用于：Node

Kubelet 用 Go 中定义的 runtime.GOARCH 值来填充该标签。这在诸如混用 arm 和 x86 节点的情况下很有用。

kubernetes.io/os

示例：kubernetes.io/os=linux

用于：Node

Kubelet 用该 Go 中定义的 runtime.GOOS 值来填充该标签。这在集群中存在不同操作系统的节点时很有用（例如：混合 Linux 和 Windows 操作系统的节点）。

beta.kubernetes.io/arch (已弃用)

该标签已被弃用。请使用 kubernetes.io/arch。

beta.kubernetes.io/os (已弃用)

该标签已被弃用。请使用 kubernetes.io/os。

kubernetes.io/hostname

示例：kubernetes.io/hostname=ip-172-20-114-199.ec2.internal

用于：Node

Kubelet 用 hostname 值来填充该标签。注意：可以通过向 kubelet 传入 --hostname-override 参数对 "真正的" hostname 进行修改。

此标签还用作拓扑层次结构的一部分。有关更多信息，请参见 topology.kubernetes.io/zone。

beta.kubernetes.io/instance-type (已弃用)

> **说明：**
>
> 从 kubernetes 1.17 版本开始，不推荐使用此标签，而推荐使用 node.kubernetes.io/instance-type。

node.kubernetes.io/instance-type

示例：node.kubernetes.io/instance-type=m3.medium

用于：Node

Kubelet 用 cloudprovider 中定义的实例类型来填充该标签。未使用 cloudprovider 时不会设置该标签。 该标签在想要将某些负载定向到特定实例类型的节点上时会很有用，但通常用户更希望依赖 Kubernetes 调度器来执行基于资源的调度， 所以用户应该致力于基于属性而不是实例类型来进行调度（例如：需要一个 GPU，而不是 g2.2xlarge）。

failure-domain.beta.kubernetes.io/region (已弃用)

参考 topology.kubernetes.io/region。

> **说明：**
>
> 从 kubernetes 1.17 版本开始，不推荐使用此标签，而推荐使用 topology.kubernetes.io/region。

failure-domain.beta.kubernetes.io/zone (已弃用)

参考 topology.kubernetes.io/zone。

> **说明：**
>
> 从 kubernetes 1.17 版本开始，不推荐使用此标签，而推荐使用 topology.kubernetes.io/zone。

topology.kubernetes.io/region

示例：

`topology.kubernetes.io/region=us-east-1`

参考 topology.kubernetes.io/zone。

topology.kubernetes.io/zone

示例：

`topology.kubernetes.io/zone=us-east-1c`

用于：Node、PersistentVolume

对于 Node： Kubelet 或外部 cloud-controller-manager 用 cloudprovider 中定义的区域信息来填充该标签。 未使用 cloudprovider 时不会设置该标签，但如果该标签在你的拓扑中有意义的话，应该考虑设置。

对于 PersistentVolume：可感知拓扑的卷制备程序将自动在 PersistentVolumes 上设置节点亲和性约束。

区域代表逻辑故障域。Kubernetes 集群通常跨越多个区域以提高可用性。 虽然区域的确切定义留给基础架构实现，但是区域的常见属性包括区域内的网络延迟非常低，区域内的免费网络流量以及与其他区域的故障独立性。 例如，一个区域内的节点可能共享一个网络交换机，但不同区域内的节点则不应共享。

地区代表一个更大的域，由一个或多个区域组成。 Kubernetes 集群跨越多个地域是不常见的，而地域或区域的确切定义则留给基础设施实现， 地域的共同属性包括它们之间的网络延迟比它们内部更高，它们之间的网络流量成本不为零，故障独立于其他区域或域。 例如，一个地域内的节点可能共享电力基础设施（例如 UPS 或发电机），但不同地域的节点通常不会共享。

Kubernetes 对区域和区域的结构做了一些假设:

1. 地域和区域是分层的:区域是地域的严格子集，任何区域都不能位于两个地域中。
2. 区域名称在地域之间是唯一的；例如，地域 "africa-east-1" 可能包含区域 "africa-east-1a" 和 "africa-east-1b"。

标签的使用者可以安全地假设拓扑标签不变。 即使标签是严格可变的，标签的使用者也可以认为节点只能通过被销毁并重建才能从一个区域迁移到另一个区域。

Kubernetes 可以以各种方式使用这些信息。 例如，调度器自动尝试将 ReplicaSet 中的多个 Pods 分布到单区域集群中的多个节点上（为了减少节点故障的影响， 请参阅 kubernetesiohostname）。 对于多区域集群，这种分布行为也被应用到区域上（以减少区域故障的影响）。 这是通过 *SelectorSpreadPriority* 实现的。

SelectorSpreadPriority 是一种尽力而为（best-effort）的处理方式，如果集群中的区域是异构的 (例如：不同区域之间的节点数量、节点类型或 Pod 资源需求不同），可能使得 Pod 在各区域间无法均匀分布。 如有需要，用户可以使用同质的区域(节点数量和类型相同) 来减小 Pod 分布不均的可能性。

由于卷不能跨区域挂载（Attach），调度器（通过 *VolumeZonePredicate* 预选）也会保证需要特定卷的 Pod 被调度到卷所在的区域中。

如果 PersistentVolumeLabel 准入控制器不支持自动为 PersistentVolume 打标签，且用户希望防止 pod 跨区域进行卷的挂载， 应考虑手动打标签 (或增加对 PersistentVolumeLabel 的支持）。如果用户的基础设施没有这种约束，则不需要为卷添加区域标签。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 01, 2020 at 5:02 PM PST: [zh]Sync changes under `content/zh/docs/reference` folder for Issues kubernetes/website#25247. (#25308) (4f9b919a8)

安装工具

Kubeadm

Kubeadm

Kubeadm 是一个提供了 kubeadm init 和 kubeadm join 的工具，作为创建 Kubernetes 集群的 "快捷途径" 的最佳实践。

kubeadm 通过执行必要的操作来启动和运行最小可用集群。按照设计，它只关注启动引导，而非配置机器。同样的，安装各种 "锦上添花" 的扩展，例如 Kubernetes Dashboard, 监控方案，以及特定云平台的扩展，都不在讨论范围内。

相反，我们希望在 kubeadm 之上构建更高级别以及更加合规的工具，理想情况下，使用 kubeadm 作为所有部署工作的基准将会更加易于创建一致性集群。

如何安装

要安装 kubeadm, 请查阅安装指南.

接下来

- kubeadm init 用于搭建控制平面节点
- kubeadm join 用于搭建工作节点并将其加入到集群中
- kubeadm upgrade 用于升级 Kubernetes 集群到新版本
- kubeadm config 如果你使用了 v1.7.x 或更低版本的 kubeadm 版本初始化你的集群，则使用 kubeadm upgrade 来配置你的集群
- kubeadm token 用于管理 kubeadm join 使用的令牌
- kubeadm reset 用于恢复通过 kubeadm init 或者 kubeadm join 命令对节点进行的任何变更
- kubeadm version 用于打印 kubeadm 的版本信息
- kubeadm alpha 用于预览一组可用于收集社区反馈的特性

kubeadm init

此命令初始化一个 Kubernetes 控制平面节点。

概要

运行此命令来搭建 Kubernetes 控制平面节点。

"init" 命令执行以下阶段：

```
preflight                  Run pre-flight checks
certs                      Certificate generation
  /ca                      Generate the self-signed Kubernetes CA to provision
identities for other Kubernetes components
  /apiserver               Generate the certificate for serving the Kubernetes API
  /apiserver-kubelet-client    Generate the certificate for the API server to connect
to kubelet
  /front-proxy-ca          Generate the self-signed CA to provision identities for
front proxy
  /front-proxy-client      Generate the certificate for the front proxy client
  /etcd-ca                 Generate the self-signed CA to provision identities for etcd
  /etcd-server             Generate the certificate for serving etcd
  /etcd-peer               Generate the certificate for etcd nodes to communicate
with each other
  /etcd-healthcheck-client     Generate the certificate for liveness probes to
healthcheck etcd
  /apiserver-etcd-client       Generate the certificate the apiserver uses to access
etcd
  /sa                      Generate a private key for signing service account tokens
```

```
along with its public key
kubeconfig                  Generate all kubeconfig files necessary to establish the
control plane and the admin kubeconfig file
  /admin                    Generate a kubeconfig file for the admin to use and for
kubeadm itself
  /kubelet                  Generate a kubeconfig file for the kubelet to use *only* for
cluster bootstrapping purposes
  /controller-manager       Generate a kubeconfig file for the controller manager
to use
  /scheduler                Generate a kubeconfig file for the scheduler to use
kubelet-start               Write kubelet settings and (re)start the kubelet
control-plane               Generate all static Pod manifest files necessary to
establish the control plane
  /apiserver                Generates the kube-apiserver static Pod manifest
  /controller-manager       Generates the kube-controller-manager static Pod
manifest
  /scheduler                Generates the kube-scheduler static Pod manifest
etcd                        Generate static Pod manifest file for local etcd
  /local                    Generate the static Pod manifest file for a local, single-node
local etcd instance
upload-config               Upload the kubeadm and kubelet configuration to a
ConfigMap
  /kubeadm                  Upload the kubeadm ClusterConfiguration to a
ConfigMap
  /kubelet                  Upload the kubelet component config to a ConfigMap
upload-certs                Upload certificates to kubeadm-certs
mark-control-plane          Mark a node as a control-plane
bootstrap-token             Generates bootstrap tokens used to join a node to a
cluster
kubelet-finalize            Updates settings relevant to the kubelet after TLS
bootstrap
  /experimental-cert-rotation  Enable kubelet client certificate rotation
addon                       Install required addons for passing Conformance tests
  /coredns                  Install the CoreDNS addon to a Kubernetes cluster
  /kube-proxy               Install the kube-proxy addon to a Kubernetes cluster
```

```
kubeadm init [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，则使用默认网络接口。
--apiserver-bind-port int32　默认值：6443
API 服务器绑定的端口。
--apiserver-cert-extra-sans stringSlice

	用于 API Server 服务证书的可选附加主题备用名称（SAN）。可以是 IP 地址和 DNS 名称。
--cert-dir string　默认值："/etc/kubernetes/pki"	
	保存和存储证书的路径。
--certificate-key string	
	用于加密 kubeadm-certs Secret 中的控制平面证书的密钥。
--config string	
	kubeadm 配置文件的路径。
--control-plane-endpoint string	
	为控制平面指定一个稳定的 IP 地址或 DNS 名称。
--cri-socket string	
	要连接的 CRI 套接字的路径。如果为空，则 kubeadm 将尝试自动检测此值；仅当安装了多个 CRI 或具有非标准 CRI 插槽时，才使用此选项。
--dry-run	
	不要应用任何更改；只是输出将要执行的操作。
--experimental-patches string	
	包含名为 "target[suffix][+patchtype].extension" 的文件的目录路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
--feature-gates string	
	一组用来描述各种功能特性的键值（key=value）对。选项是：IPv6DualStack=true|false (ALPHA - default=false)
-h, --help	
	init 操作的帮助命令
--ignore-preflight-errors stringSlice	
	错误将显示为警告的检查列表；例如：'IsPrivilegedUser,Swap'。取值为 'all' 时将忽略检查中的所有错误。
--image-repository string　默认值："k8s.gcr.io"	
	选择用于拉取控制平面镜像的容器仓库
--kubernetes-version string　默认值："stable-1"	
	为控制平面选择一个特定的 Kubernetes 版本。
--node-name string	
	指定节点的名称。
--pod-network-cidr string	
	指明 pod 网络可以使用的 IP 地址段。如果设置了这个参数，控制平面将会为每一个节点自动分配 CIDRs。
--service-cidr string　默认值："10.96.0.0/12"	
	为服务的虚拟 IP 地址另外指定 IP 地址段
--service-dns-domain string　默认值："cluster.local"	
	为服务另外指定域名，例如："myorg.internal"。

--skip-certificate-key-print	
	不要打印用于加密控制平面证书的密钥。
--skip-phases stringSlice	
	要跳过的阶段列表
--skip-token-print	
	跳过打印 'kubeadm init' 生成的默认引导令牌。
--token string	
	这个令牌用于建立控制平面节点与工作节点间的双向通信。格式为 [a-z0-9]{6}\.[a-z0-9]{16} - 示例：abcdef.0123456789abcdef
--token-ttl duration　默认值：24h0m0s	
	令牌被自动删除之前的持续时间（例如 1 s，2 m，3 h）。如果设置为 '0'，则令牌将永不过期
--upload-certs	
	将控制平面证书上传到 kubeadm-certs Secret。

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

Init 命令的工作流程

kubeadm init 命令通过执行下列步骤来启动一个 Kubernetes 控制平面节点。

1. 在做出变更前运行一系列的预检项来验证系统状态。一些检查项目仅仅触发警告，其它的则会被视为错误并且退出 kubeadm，除非问题得到解决或者用户指定了 --ignore-preflight-errors=<错误列表> 参数。

1. 生成一个自签名的 CA 证书来为集群中的每一个组件建立身份标识。 用户可以通过将其放入 --cert-dir 配置的证书目录中（默认为 /etc/kubernetes/pki） 来提供他们自己的 CA 证书以及/或者密钥。 APIServer 证书将为任何 --apiserver-cert-extra-sans 参数值提供附加的 SAN 条目，必要时将其小写。

1. 将 kubeconfig 文件写入 /etc/kubernetes/ 目录以便 kubelet、控制器管理器和调度器用来连接到 API 服务器，它们每一个都有自己的身份标识，同时生成一个名为 admin.conf 的独立的 kubeconfig 文件，用于管理操作。

1. 为 API 服务器、控制器管理器和调度器生成静态 Pod 的清单文件。假使没有提供一个外部的 etcd 服务的话，也会为 etcd 生成一份额外的静态 Pod 清单文件。

   静态 Pod 的清单文件被写入到 /etc/kubernetes/manifests 目录; kubelet 会监视这个目录以便在系统启动的时候创建 Pod。

一旦控制平面的 Pod 都运行起来， kubeadm init 的工作流程就继续往下执行。

1. 对控制平面节点应用标签和污点标记以便不会在它上面运行其它的工作负载。

1. 生成令牌，将来其他节点可使用该令牌向控制平面注册自己。 如 kubeadm token 文档所述， 用户可以选择通过 --token 提供令牌。

1. 为了使得节点能够遵照启动引导令牌 和 TLS 启动引导 这两份文档中描述的机制加入到集群中，kubeadm 会执行所有的必要配置：

    - 创建一个 ConfigMap 提供添加集群节点所需的信息，并为该 ConfigMap 设置相关的 RBAC 访问规则。
    - 允许启动引导令牌访问 CSR 签名 API。
    - 配置自动签发新的 CSR 请求。

    更多相关信息，请查看 kubeadm join。

1. 通过 API 服务器安装一个 DNS 服务器 (CoreDNS) 和 kube-proxy 附加组件。 在 Kubernetes 版本 1.11 和更高版本中，CoreDNS 是默认的 DNS 服务器。 要安装 kube-dns 而不是 CoreDNS，必须在 kubeadm ClusterConfiguration 中配置 DNS 插件。 有关配置的更多信息，请参见下面的"带配置文件使用 kubeadm init" 一节。 请注意，尽管已部署 DNS 服务器，但直到安装 CNI 时才调度它。

    > **警告：** 从 v1.18 开始，在 kubeadm 中使用 kube-dns 已废弃，并将在以后的版本中将其删除。

在 kubeadm 中使用 init phases

Kubeadm 允许你使用 kubeadm init phase 命令分阶段创建控制平面节点。

要查看阶段和子阶段的有序列表，可以调用 kubeadm init --help。 该列表将位于帮助屏幕的顶部，每个阶段旁边都有一个描述。 注意，通过调用 kubeadm init，所有阶段和子阶段都将按照此确切顺序执行。

某些阶段具有唯一的标志，因此，如果要查看可用选项的列表，请添加 --help，例如：

```
sudo kubeadm init phase control-plane controller-manager --help
```

你也可以使用 --help 查看特定父阶段的子阶段列表：

```
sudo kubeadm init phase control-plane --help
```

kubeadm init 还公开了一个名为 --skip-phases 的参数，该参数可用于跳过某些阶段。 参数接受阶段名称列表，并且这些名称可以从上面的有序列表中获取。

例如：

```
sudo kubeadm init phase control-plane all --config=configfile.yaml
sudo kubeadm init phase etcd local --config=configfile.yaml
# 你现在可以修改控制平面和 etcd 清单文件
sudo kubeadm init --skip-phases=control-plane,etcd --config=configfile.yaml
```

该示例将执行的操作是基于 configfile.yaml 中的配置在 /etc/kubernetes/manifests 中写入控制平面和 etcd 的清单文件。 这允许你修改文件，然后使用 --skip-phases 跳过这些阶段。 通过调用最后一个命令，你将使用自定义清单文件创建一个控制平面节点。

结合一份配置文件来使用 kubeadm init

> **注意：** 配置文件的功能仍然处于 alpha 状态并且在将来的版本中可能会改变。

通过一份配置文件而不是使用命令行参数来配置 kubeadm init 命令是可能的， 但是一些更加高级的功能只能够通过配置文件设定。 这份配置文件通过 --config 选项参数指定的， 它必须包含 ClusterConfiguration 结构，并可能包含更多由 ---\n 分隔的结构。 在某些情况下，可能不允许将 --config 与其他标志混合使用。

可以使用 kubeadm config print 命令打印出默认配置。

如果你的配置没有使用最新版本， **推荐**使用 kubeadm config migrate 命令进行迁移。

有关配置的字段和用法的更多信息， 你可以访问 API 参考页面并从 列表 中选择一个版本。

添加 kube-proxy 参数

kubeadm 配置中有关 kube-proxy 的说明请查看：

- kube-proxy

使用 kubeadm 启用 IPVS 模式的说明请查看：

- IPVS

向控制平面组件传递自定义的命令行参数

有关向控制平面组件传递命令行参数的说明请查看： 控制平面命令行参数

使用自定义的镜像

默认情况下, kubeadm 会从 k8s.gcr.io 仓库拉取镜像。如果请求的 Kubernetes 版本是 CI 标签 （例如 ci/latest ），则使用 gcr.io/kubernetes-ci-images。

你可以通过使用带有配置文件的 kubeadm来重写此操作。

允许的自定义功能有：

- 使用其他的 imageRepository 来代替 k8s.gcr.io。
- 将 useHyperKubeImage 设置为 true，使用 HyperKube 镜像。
- 为 etcd 或 DNS 附件提供特定的 imageRepository 和 imageTag。

请注意配置文件中的配置项 kubernetesVersion 或者命令行参数 --kubernetes-version 会影响到镜像的版本。

将控制平面证书上传到集群

通过将参数 --upload-certs 添加到 kubeadm init，你可以将控制平面证书临时上传到集群中的 Secret。 请注意，此 Secret 将在 2 小时后自动过期。证书使用 32 字节密钥加密，可以使用 --certificate-key 指定。 通过将 --control-plane 和 --certificate-key 传递给 kubeadm join， 可以在添加其他控制平面节点时使用相同的密钥下载证书。

以下阶段命令可用于证书到期后重新上传证书：

```
kubeadm init phase upload-certs --upload-certs --certificate-key=SOME_VALUE --config=SOME_YAML_FILE
```

如果未将参数 --certificate-key 传递给 kubeadm init 和 kubeadm init phase upload-certs， 则会自动生成一个新密钥。

以下命令可用于按需生成新密钥：

```
kubeadm certs certificate-key
```

使用 kubeadm 管理证书

有关使用 kubeadm 进行证书管理的详细信息，请参阅 使用 kubeadm 进行证书管理。该文档包括有关使用外部 CA，自定义证书和证书更新的信息。

管理 kubeadm 为 kubelet 提供的 systemd 配置文件

kubeadm 包自带了关于 systemd 如何运行 kubelet 的配置文件。 请注意 kubeadm 客户端命令行工具永远不会修改这份 systemd 配置文件。 这份 systemd 配置文件属于 kubeadm DEB/RPM 包。

有关更多信息，请阅读 管理 systemd 的 kubeadm 内嵌文件。

结合 CRI 运行时使用 kubeadm

默认情况下，kubeadm 尝试检测你的容器运行环境。有关此检测的更多详细信息，请参见 kubeadm CRI 安装指南。

设置节点的名称

默认情况下, kubeadm 基于机器的主机地址分配一个节点名称。你可以使用 --node-name 参数覆盖此设置。 此标识将合适的 --hostname-override 值传递给 kubelet。

在没有互联网连接的情况下运行 kubeadm

要在没有互联网连接的情况下运行 kubeadm，你必须提前拉取所需的控制平面镜像。

你可以使用 kubeadm config images 子命令列出并拉取镜像：

```
kubeadm config images list
kubeadm config images pull
```

kubeadm 需要的所有镜像，例如 k8s.gcr.io/kube-*、k8s.gcr.io/etcd 和 k8s.gcr.io/pause 都支持多种架构。

kubeadm 自动化

除了像文档 kubeadm 基础教程 中所描述的那样，将从 kubeadm init 取得的令牌复制到每个节点， 你还可以并行地分发令牌以实现简单自动化。 要实现自动化，你必须知道控制平面节点启动后将拥有的 IP 地址，或使用 DNS 名称或负载均衡器的地址。

1. 生成一个令牌。这个令牌必须具有以下格式：< 6 个字符的字符串>.< 16 个字符的字符串>。 更加正式的说法是，它必须符合以下正则表达式：[a-z0-9]{6}\.[a-z0-9]{16}。

   kubeadm 可以为你生成一个令牌：

   ```
   kubeadm token generate
   ```

1. 使用这个令牌同时启动控制平面节点和工作节点。它们一旦运行起来应该就会互相寻找对方并且建立集群。 同样的 --token 参数可以同时用于 kubeadm init 和 kubeadm join 命令。

1. 当加入其他控制平面节点时，可以对 --certificate-key 执行类似的操作。可以使用以下方式生成密钥：

   ```
   kubeadm certs certificate-key
   ```

一旦集群启动起来，你就可以从控制平面节点的 /etc/kubernetes/admin.conf 文件获取管理凭证， 并使用这个凭证同集群通信。

注意这种搭建集群的方式在安全保证上会有一些宽松，因为这种方式不允许使用 --discovery-token-ca-cert-hash 来验证根 CA 的哈希值（因为当配置节点的时候，它还没有被生成）。 更多信息请参阅 kubeadm join文档。

接下来

- 进一步阅读了解 kubeadm init phase
- kubeadm join 启动一个 Kubernetes 工作节点并且将其加入到集群
- kubeadm upgrade 将 Kubernetes 集群升级到新版本
- kubeadm reset 恢复 kubeadm init 或 kubeadm join 命令对节点所作的变更

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 14, 2020 at 4:10 PM PST: [zh] Sync changes to kubeadm reference (e944fb147)

kubeadm join

此命令用来初始化 Kubernetes 工作节点并将其加入集群。

摘要

当节点加入 kubeadm 初始化的集群时，我们需要建立双向信任。 这个过程可以分解为发现（让待加入节点信任 Kubernetes 控制平面节点）和 TLS 引导（让Kubernetes 控制平面节点信任待加入节点）两个部分。

有两种主要的发现方案。 第一种方法是使用共享令牌和 API 服务器的 IP 地址。 第二种是提供一个文件 - 标准 kubeconfig 文件的一个子集。 该文件可以是本地文件，也可以通过 HTTPS URL 下载。 格式是 kubeadm join --discovery-token abcdef.1234567890abcdef 1.2.3.4:6443、kubeadm join--discovery-file path/to/file.conf 或者kubeadm join --discovery-file https://url/file.conf。 只能使用其中一种。 如果发现信息是从 URL 加载的，必须使用 HTTPS。 此外，在这种情况下，主机安装的 CA 包用于验证连接。

如果使用共享令牌进行发现，还应该传递 --discovery-token-ca-cert-hash 参数来验证 Kubernetes 控制平面节点提供的根证书颁发机构（CA）的公钥。 此参数的值指定为 "<hash-type>:<hex-encoded-value>"，其中支持的哈希类型为 "sha256"。哈希是通过 Subject Public Key Info（SPKI）对象的字节计算的（如 RFC7469）。 这个值可以从 "kubeadm init" 的输出中获得，或者可以使用标准工具进行计算。 可以多次重复 --discovery-token-ca-cert-hash 参数以允许多个公钥。

如果无法提前知道 CA 公钥哈希，则可以通过 --discovery-token-unsafe-skip-ca-verification 参数禁用此验证。 这削弱了kubeadm 安全模型，因为其他节点可能会模仿 Kubernetes 控制平面节点。

TLS 引导机制也通过共享令牌驱动。 这用于向 Kubernetes 控制平面节点进行临时的身份验证，以提交本地创建的密钥对的证书签名请求（CSR）。 默认情况下，kubeadm 将设置 Kubernetes 控制平面节点自动批准这些签名请求。 这个令牌通过 --tls-bootstrap-token abcdef.1234567890abcdef 参数传入。

通常两个部分会使用相同的令牌。 在这种情况下可以使用 --token 参数，而不是单独指定每个令牌。

"join [api-server-endpoint]" 命令执行下列阶段：

```
preflight              Run join pre-flight checks
control-plane-prepare  Prepare the machine for serving a control plane
  /download-certs        [EXPERIMENTAL] Download certificates shared among control-plane nodes from the kubeadm-certs Secret
  /certs                 Generate the certificates for the new control plane components
  /kubeconfig            Generate the kubeconfig for the new control plane components
  /control-plane         Generate the manifests for the new control plane components
kubelet-start          Write kubelet settings, certificates and (re)start the kubelet
control-plane-join     Join a machine as a control plane instance
  /etcd                  Add a new local etcd member
  /update-status         Register the new control-plane node into the ClusterStatus maintained in the kubeadm-config ConfigMap
  /mark-control-plane    Mark a node as a control-plane
```

```
kubeadm join [api-server-endpoint] [flags]
```

选项

--apiserver-advertise-address string	
	如果该节点托管一个新的控制平面实例，则 API 服务器将公布其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--apiserver-bind-port int32　　默认值: 6443	
	如果节点应该托管新的控制平面实例，则为 API 服务器要绑定的端口。
--certificate-key string	
	使用此密钥可以解密由 init 上传的证书 secret。
--config string	
	kubeadm 配置文件的路径。
--control-plane	
	在此节点上创建一个新的控制平面实例
--cri-socket string	
	要连接的 CRI 套接字的路径。如果为空，则 kubeadm 将尝试自动检测此值；仅当安装了多个 CRI 或具有非标准 CRI 插槽时，才使用此选项。
--discovery-file string	
	对于基于文件的发现，给出用于加载集群信息的文件或者 URL。

--discovery-token string	
	对于基于令牌的发现，该令牌用于验证从 API 服务器获取的集群信息。
--discovery-token-ca-cert-hash stringSlice	
	对基于令牌的发现，验证根 CA 公钥是否与此哈希匹配 (格式: "<type>:<value>")。
--discovery-token-unsafe-skip-ca-verification	
	对于基于令牌的发现，允许在未关联 --discovery-token-ca-cert-hash 参数的情况下添加节点。
--experimental-patches string	
	包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
-h, --help	
	join 操作的帮助命令
--ignore-preflight-errors stringSlice	
	错误将显示为警告的检查列表；例如：'IsPrivilegedUser,Swap'。取值为 'all' 时将忽略检查中的所有错误。
--node-name string	
	指定节点的名称
--skip-phases stringSlice	
	要跳过的阶段列表
--tls-bootstrap-token string	
	指定在加入节点时用于临时通过 Kubernetes 控制平面进行身份验证的令牌。
--token string	
	如果未提供这些值，则将它们用于 discovery-token 令牌和 tls-bootstrap 令牌。

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

join 工作流

kubeadm join 初始化 Kubernetes 工作节点并将其加入集群。 该操作过程包含下面几个步骤：

1. kubeadm 从 API 服务器下载必要的集群信息。 默认情况下，它使用引导令牌和 CA 密钥哈希来验证数据的真实性。 也可以通过文件或 URL 直接发现根 CA。

1. 一旦知道集群信息，kubelet 就可以开始 TLS 引导过程。

TLS 引导程序使用共享令牌与 Kubernetes API 服务器进行临时的身份验证，以提交证书签名请求 (CSR)； 默认情况下，控制平面自动对该 CSR 请求进行签名。

1. 最后，kubeadm 配置本地 kubelet 使用分配给节点的确定标识连接到 API 服务器。

对于控制平面节点，执行额外的步骤：

1. 从集群下载控制平面节点之间共享的证书（如果用户明确要求）。
2. 生成控制平面组件清单、证书和 kubeconfig。
3. 添加新的本地 etcd 成员。
4. 将此节点添加到 kubeadm 集群的 ClusterStatus。

使用 kubeadm 的 join phase 命令

Kubeadm 允许你使用 kubeadm join phase 分阶段将节点加入集群。

要查看阶段和子阶段的有序列表，可以调用 kubeadm join --help。 该列表将位于帮助屏幕的顶部，每个阶段旁边都有一个描述。 注意，通过调用 kubeadm join，所有阶段和子阶段都将按照此确切顺序执行。

有些阶段具有唯一的标志，因此，如果要查看可用选项列表，请添加 --help，例如：

```
kubeadm join phase kubelet-start --help
```

类似于 kubeadm init phase命令， kubeadm join phase 允许你使用 --skip-phases 标志跳过阶段列表。

例如：

```
sudo kubeadm join --skip-phases=preflight --config=config.yaml
```

发现要信任的集群 CA

Kubeadm 的发现有几个选项，每个选项都有安全性上的优缺点。 适合你的环境的正确方法取决于节点是如何准备的以及你对网络的安全性期望和节点的生命周期特点。

带 CA 锁定模式的基于令牌的发现

这是 Kubernetes 1.8 及以上版本中的默认模式。 在这种模式下，kubeadm 下载集群配置（包括根CA）并使用令牌验证它， 并且会验证根 CA 的公钥与所提供的哈希是否匹配， 以及 API 服务器证书在根 CA 下是否有效。

CA 键哈希格式为 sha256:<hex_encoded_hash>。 默认情况下，在 kubeadm init 最后打印的 kubeadm join 命令 或者 kubeadm token create --print-join-command 的输出信息中返回哈希值。 它使用标准格式 (请参考 RFC7469) 并且也能通过第三方工具或者制备系统进行计算。 例如，使用 OpenSSL CLI：

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* //'
```

kubeadm join 命令示例

对于工作节点：

```
kubeadm join --discovery-token abcdef.1234567890abcdef --discovery-token-ca-cert-hash sha256:1234..cdef 1.2.3.4:6443
```

对于控制面节点：

```
kubeadm join --discovery-token abcdef.1234567890abcdef --discovery-token-ca-cert-hash sha256:1234..cdef --control-plane 1.2.3.4:6443
```

如果使用 `--upload-certs` 调用 kubeadm init 命令， 你也可以对控制平面节点调用带 `--certificate-key` 参数的 join 命令， 将证书复制到该节点。

优势：

- 允许引导节点安全地发现主节点的信任根，即使其他工作节点或网络受到损害。
- 方便手动执行，因为所需的所有信息都适合于易于复制和粘贴的单个 kubeadm join 命令。

劣势：

- CA 哈希通常在主节点被提供之前是不知道的，这使得构建使用 kubeadm 的自动化配置工具更加困难。 通过预先生成CA，你可以解除这个限制。

无 CA 锁定模式的基于令牌的发现

_这是 Kubernetes 1.7 和早期版本_中的默认设置；使用时要注意一些重要的补充说明。 此模式仅依赖于对称令牌来签名(HMAC-SHA256)发现信息，这些发现信息为主节点建立信任根。 在 Kubernetes 1.8 及以上版本中仍然可以使用 `--discovery-token-unsafe-skip-ca-verification` 参数，但是如果可能的话，你应该考虑使用一种其他模式。

kubeadm join 命令示例

```
kubeadm join --token abcdef.1234567890abcdef --discovery-token-unsafe-skip-ca-verification 1.2.3.4:6443
```

优势

- 仍然可以防止许多网络级攻击。
- 可以提前生成令牌并与主节点和工作节点共享，这样主节点和工作节点就可以并行引导而无需协调。 这允许它在许多配置场景中使用。

劣势

- 如果攻击者能够通过某些漏洞窃取引导令牌，那么他们可以使用该令牌（连同网络级访问）为其它处于引导过程中的节点提供假冒的主节点。 在你的环境中，这可能是一个适当的折衷方法，也可能不是。

基于 HTTPS 或文件发现

这种方案提供了一种带外方式在主节点和引导节点之间建立信任根。 如果使用 kubeadm 构建自动配置，请考虑使用此模式。 发现文件的格式为常规的 Kubernetes kubeconfig 文件。

如果发现文件不包含凭据，则将使用 TLS 发现令牌。

kubeadm join 命令示例：

- kubeadm join --discovery-file path/to/file.conf （本地文件）
- kubeadm join --discovery-file https://url/file.conf (远程 HTTPS URL)

优势：

- 允许引导节点安全地发现主节点的信任根，即使网络或其他工作节点受到损害。

劣势：

- 要求你有某种方法将发现信息从主节点传送到引导节点。 例如，这可以通过云提供商或驱动工具实现。 该文件中的信息不是加密的，而是需要 HTTPS 或等效文件来保证其完整性。

确保你的安装更加安全

Kubeadm 的默认值可能不适用于所有人。 本节说明如何以牺牲可用性为代价来加强 kubeadm 安装。

关闭节点客户端证书的自动批准

默认情况下，Kubernetes 启用了 CSR 自动批准器，如果在身份验证时使用 Bootstrap Token，它会批准对 kubelet 的任何客户端证书的请求。 如果不希望集群自动批准kubelet客户端证书，可以通过执行以下命令关闭它：

```
kubectl delete clusterrolebinding kubeadm:node-autoapprove-bootstrap
```

关闭后，kubeadm join 操作将会被阻断，直到管理员已经手动批准了在途中的 CSR 才会继续：

```
kubectl get csr
```

输出类似于：

```
NAME                                                   AGE     REQUESTOR                 CONDITION
node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ   18s     system:bootstrap:878f07   Pending
```

```
kubectl certificate approve node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ
```

输出类似于：

```
certificatesigningrequest "node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ" approved
```

```
kubectl get csr
```

输出类似于：

```
NAME                                                   AGE     REQUESTOR                 CONDITION
node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ   1m      system:bootstrap:878f07   Approved,Issued
```

这迫使工作流只有在运行了 kubectl 证书批准后，kubeadm join 才能成功。

关闭对集群信息 ConfigMap 的公开访问

为了实现使用令牌作为唯一验证信息的加入工作流，默认情况下会公开带有验证主节点标识所需数据的 ConfigMap。 虽然此 ConfigMap 中没有私有数据，但一些用户可能希望无论如何都关闭它。 这样做需要禁用 kubeadm join 工作流的 --discovery-token 参数。 以下是实现步骤：

- 从 API 服务器获取 cluster-info 文件：

```
kubectl -n kube-public get cm cluster-info -o yaml | grep "kubeconfig:" -A11 | grep "apiVersion" -A10 | sed "s/    //" | tee cluster-info.yaml
```

输出类似于：

```
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority-data: <ca-cert>
    server: https://<ip>:<port>
  name: ""
contexts: []
current-context: ""
preferences: {}
users: []
```

- 使用 cluster-info.yaml 文件作为 kubeadm join --discovery-file 参数。

- 关闭 cluster-info ConfigMap 的公开访问：

```
kubectl -n kube-public delete rolebinding kubeadm:bootstrap-signer-clusterinfo
```

这些命令应该在执行 kubeadm init 之后、在kubeadm join 之前执行。

使用带有配置文件的 kubeadm join

> **注意：**
>
> 配置文件目前是 alpha 功能，在将来的版本中可能会变动。

可以用配置文件替代命令行参数的方法配置 kubeadm join，一些高级功能也只有在使用配置文件时才可选用。 该文件通过 --config 参数来传递，并且文件中必须包含 JoinConfiguration 结构。 在某些情况下，不允许将 --config 与其他标志混合使用。

使用 kubeadm config print 命令可以打印默认配置。

如果你的配置没有使用最新版本， **推荐**使用 kubeadm config migrate 命令转换。

有关配置的字段和用法的更多信息，你可以导航到我们的 API 参考页 并从[列表]中选择一个版本(https://godoc.org/k8s.io/kubernetes/cmd/kubeadm/app/apis/kubeadm#pkg-subdirectories)。

接下来

- kubeadm init 初始化 Kubernetes 主节点
- kubeadm token 管理 kubeadm join 的令牌
- kubeadm reset 将 kubeadm init 或 kubeadm join 对主机的更改恢复到之前状态

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 27, 2020 at 5:10 PM PST: sync changes in docs/reference/setup-tools/kubeadm/ directory (679b45e2c)

kubeadm upgrade

kubeadm upgrade 是一个对用户友好的命令，它将复杂的升级逻辑包装在一个命令后面，支持升级的规划和实际执行。

kubeadm 升级指南

本文档概述了使用 kubeadm 执行升级的步骤。 有关 kubeadm 旧版本，请参阅 Kubernetes 网站的旧版文档。

你可以使用 kubeadm upgrade diff 来查看将应用于静态 pod 清单的更改。

要在 Kubernetes v1.13.0 及更高版本中使用 kube-dns 进行升级，请遵循本指南。

在 Kubernetes v1.15.0 和更高版本中，kubeadm upgrade apply 和 kubeadm upgrade node 也将自动续订该节点上的 kubeadm 托管证书，包括存储在 kubeconfig 文件中的证书。 要选择退出，可以传递参数 --certificate-renewal=false。有关证书续订的更多详细信息请参见证书管理文档。

> **说明：**
>
> kubeadm upgrade apply 和 kubeadm upgrade plan 命令都具有遗留的 --config 标志， 可以在执行特定控制平面节点的规划或升级时重新配置集群。请注意，升级工作流不是为这种情况而设计的，并且有意外结果的报告。

kubeadm upgrade plan

概述

检查可升级到哪些版本，并验证您当前的集群是否可升级。 要跳过互联网检查，请传递可选的 [version] 参数

```
kubeadm upgrade plan [version] [flags]
```

选项

--allow-experimental-upgrades	
	显示不稳定版本的 Kubernetes 作为升级替代方案，并允许升级到 Kubernetes 的 Alpha/Beta/发行候选版本。
--allow-release-candidate-upgrades	
	显示 Kubernetes 的发行候选版本作为升级选择，并允许升级到 Kubernetes 的发行候选版本。
--config string	
	配置文件的路径。
--feature-gates string	
	一组描述各种特征特性门控的键值对。选项有：IPv6DualStack=true|false (ALPHA - default=false) PublicKeysECDSA=true|false (ALPHA - default=false)
-h, --help	
	帮助
--ignore-preflight-errors stringSlice	

	检查清单，其错误将显示为警告。 例如："IsPrivilegedUser，Swap"。 值 "all" 忽略所有检查的错误。
--kubeconfig string Default: "/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。 如果标志为未设置，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--print-config	
	指定是否打印将在升级中使用的配置文件。

从父命令继承的选项

--rootfs string	
	[EXPERIMENTAL] "真实"主机根文件系统的路径。

kubeadm upgrade apply

概要

将 Kubernetes 集群升级到指定版本

```
kubeadm upgrade apply [version]
```

选项

--allow-experimental-upgrades	
	显示 Kubernetes 的不稳定版本作为升级替代方案，并允许升级到 Kubernetes 的 alpha/beta 或 RC 版本。
--allow-release-candidate-upgrades	
	显示 Kubernetes 的候选版本作为升级替代方案，并允许升级到 Kubernetes 的 RC 版本。
--certificate-renewal Default: true	
	执行升级期间更改的组件所使用的证书的更新。
--config string	
	kubeadm 配置文件的路径。
--dry-run	
	不要更改任何状态，只输出要执行的操作。
--etcd-upgrade 默认值: true	
	执行 etcd 的升级。
--experimental-patches string	
	包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
--feature-gates string	

	一组键值对，用于描述各种功能。选项包括： IPv6DualStack=true|false (ALPHA - 默认=false) PublicKeysECDSA=true|false (ALPHA - 默认=false)
-f, --force	
	强制升级，但可能无法满足某些要求。这也意味着非交互模式。
-h, --help	
	apply 操作的帮助命令
--ignore-preflight-errors stringSlice	
	错误将显示为警告的检查列表；例如：'IsPrivilegedUser,Swap'。取值为 'all' 时将忽略检查中的所有错误。
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。如果未设置标志，则在相关目录下搜索以查找现有 kubeconfig 文件。
--print-config	
	指定是否应打印将在升级中使用的配置文件。
-y, --yes	
	执行升级，不提示确认（非交互模式）。

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm upgrade diff

概述

显示哪些差异将被应用于现有的静态 pod 资源清单。参考: kubeadm upgrade apply --dry-run

```
kubeadm upgrade diff [version] [flags]
```

选项

--api-server-manifest string 默认值："/etc/kubernetes/manifests/kube-apiserver.yaml"
API服务器清单的路径
--config string
kubeadm 配置文件的路径
-c, --context-lines int 默认值：3
差异中有多少行上下文
--controller-manager-manifest string 默认值： "/etc/kubernetes/manifests/kube-controller-manager.yaml"
控制器清单的路径

-h, --help
帮助
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件，如果标志是未设置，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--scheduler-manifest string 默认值："/etc/kubernetes/manifests/kube-scheduler.yaml"
调度程序清单的路径

从父命令继承的选项

--rootfs string
[EXPERIMENTAL] "真实"主机根文件系统的路径。

kubeadm upgrade node

概要

升级集群中某个节点的命令

"node" 命令执行以下阶段：

```
preflight       执行节点升级前检查
control-plane   如果存在的话，升级部署在该节点上的管理面实例
kubelet-config  更新该节点上的 kubelet 配置
```

```
kubeadm upgrade node [flags]
```

选项

--certificate-renewal
对升级期间变化的组件所使用的证书执行更新。
--dry-run
不更改任何状态，只输出将要执行的操作。
--etcd-upgrade 默认值: true
执行 etcd 的升级。
--experimental-patches string
包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
-h, --help
node 操作的帮助命令
--kubeconfig string 默认值: "/etc/kubernetes/admin.conf"

	用于与集群交互的 kubeconfig 文件。如果参数未指定，将从一系列标准位置检索存在的 kubeconfig 文件。
--kubelet-version string	
	升级后 *期望的* kubelet 配置版本。如未指定，将使用 kubeadm-config ConfigMap 中的 KubernetesVersion
--skip-phases stringSlice	
	要跳过的阶段的列表

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

接下来

- 如果你使用 kubeadm v1.7.x 或更低版本初始化集群，则可以参考kubeadm 配置配置集群用于 kubeadm upgrade。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 27, 2020 at 5:10 PM PST: sync changes in docs/reference/setup-tools/kubeadm/ directory (679b45e2c)

kubeadm config

在 kubeadm init 执行期间，kubeadm 将 ClusterConfiguration 对象上传到你的集群的 kube-system 名字空间下 名为 kubeadm-config 的 ConfigMap 对象中。 然后在 kubeadm join、kubeadm reset 和 kubeadm upgrade 执行期间读取此配置。 要查看此 ConfigMap，请调用 kubeadm config view。

你可以使用 kubeadm config print 命令打印默认配置， 并使用 kubeadm config migrate 命令将旧版本的配置转化成新版本。 kubeadm config images list 和 kubeadm config images pull 命令可以用来列出并拉取 kubeadm 所需的镜像。

更多信息请浏览使用带配置文件的 kubeadm init 或使用带配置文件的 kubeadm join.

在 Kubernetes v1.13.0 及更高版本中，要列出/拉取 kube-dns 镜像而不是 CoreDNS 镜像， 必须使用这里所描述的 --config 方法。

kubeadm config upload from-file

kubeadm config view

概要

使用此命令，可以查看 kubeadm 配置的集群中的 ConfigMap。 该配置位于 "kube-system" 命名空间中的名为 "kubeadm-config" 的 ConfigMap 中。

```
kubeadm config view [flags]
```

选项

-h, --help
view 操作的帮助命令

继承于父命令的选项

--kubeconfig string　　默认值：/etc/kubernetes/admin.conf"
用于和集群通信的 KubeConfig 文件。如果未设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 KubeConfig 文件。
--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

kubeadm config print init-defaults

概要

此命令打印对象，例如用于 'kubeadm init' 的默认 init 配置对象。

请注意，Bootstrap Token 字段之类的敏感值已替换为 {"abcdef.0123456789abcdef" "" "nil" <nil> [] []} 之类的占位符值以通过验证，但不执行创建令牌的实际计算。

```
kubeadm config print init-defaults [flags]
```

选项

--component-configs stringSlice
组件配置 API 对象的逗号分隔列表，打印其默认值。可用值：[KubeProxyConfiguration KubeletConfiguration]。如果未设置此参数，则不会打印任何组件配置。
-h, --help
init-defaults 操作的帮助命令

从父命令继承的选项

--kubeconfig string 默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

kubeadm config print join-defaults

概要

此命令打印对象，例如用于 'kubeadm join' 的默认 join 配置对象。

请注意，诸如启动引导令牌字段之类的敏感值已替换为 {"abcdef.0123456789abcdef" "" "nil" <nil> [] []} 之类的占位符值以通过验证，但不执行创建令牌的实际计算。

```
kubeadm config print join-defaults [flags]
```

选项

--component-configs stringSlice
组件配置 API 对象的逗号分隔列表，打印其默认值。可用值：[KubeProxyConfiguration KubeletConfiguration]。如果未设置此参数，则不会打印任何组件配置。
-h, --help
join-defaults 操作的帮助命令

从父命令继承的选项

--kubeconfig string 默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

kubeadm config migrate

概要

此命令允许您在 CLI 工具中将本地旧版本的配置对象转换为最新支持的版本，而无需变更集群中的任何内容。在此版本的 kubeadm 中，支持以下 API 版本：

- kubeadm.k8s.io/v1beta2

因此，无论您在此处传递 --old-config 参数的版本是什么，当写入到 stdout 或 --new-config （如果已指定）时， 都会读取、反序列化、默认、转换、验证和重新序列化 API 对象。

换句话说，如果您将此文件传递给 "kubeadm init"，则该命令的输出就是 kubeadm 实际上在内部读取的内容。

```
kubeadm config migrate [flags]
```

选项

-h, --help	
	migrate 操作的帮助信息
--new-config string	
	使用新的 API 版本生成的 kubeadm 配置文件的路径。这个路径是可选的。如果没有指定，输出将被写到 stdout。
--old-config string	
	使用旧 API 版本且应转换的 kubeadm 配置文件的路径。此参数是必需的。

从父命令继承的选项

--kubeconfig string 默认值："/etc/kubernetes/admin.conf"	
	用于和集群通信的 kubeconfig 文件。如果未设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 kubeconfig 文件。
--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm config images list

概要

打印 kubeadm 要使用的镜像列表。配置文件用于自定义任何镜像或镜像存储库。

```
kubeadm config images list [flags]
```

选项

--allow-missing-template-keys 默认值：true	
	如果设置为 true，则在模板中缺少字段或哈希表的键时忽略模板中的任何错误。 仅适用于 golang 和 jsonpath 输出格式。
-o, --experimental-output string 默认值："text"	
	输出格式：text|json|yaml|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file 其中之一
--config string	
	kubeadm 配置文件的路径。

--feature-gates string
一组键值对（key=value），用于描述各种特征。选项是： Auditing=true|false (ALPHA - 默认=false) CoreDNS=true|false (默认=true) DynamicKubeletConfig=true|false (BETA - 默认=false)
-h, --help
list 操作的帮助命令
--image-repository string　默认值："k8s.gcr.io"
选择要从中拉取控制平面镜像的容器仓库
--kubernetes-version string　默认值："stable-1"
为控制平面选择一个特定的 Kubernetes 版本

从父命令继承的选项

--kubeconfig string　默认值："/etc/kubernetes/admin.conf"
用于和集群通信的 kubeconfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 kubeconfig 文件。
--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

kubeadm config images pull

概要

拉取 kubeadm 使用的镜像。

```
kubeadm config images pull [flags]
```

选项

--config string
kubeadm 配置文件的路径。
--cri-socket string
要连接的 CRI 套接字的路径。如果为空，则 kubeadm 将尝试自动检测此值；仅当安装了多个 CRI 或具有非标准 CRI 插槽时，才使用此选项。
--feature-gates string
一系列键值对（key=value），用于描述各种特征。可选项是： IPv6DualStack=true|false (ALPHA - 默认值=false)
-h, --help
pull 操作的帮助命令
--image-repository string　默认值："k8s.gcr.io"
选择用于拉取控制平面镜像的容器仓库
--kubernetes-version string　默认值："stable-1"

	为控制平面选择一个特定的 Kubernetes 版本。

从父命令继承的选项

--kubeconfig string　默认值："/etc/kubernetes/admin.conf"
用于和集群通信的 kubeconfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 kubeconfig 文件。
--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

接下来

- kubeadm upgrade 将 Kubernetes 集群升级到更新版本 [kubeadm upgrade]

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 27, 2020 at 5:10 PM PST: sync changes in docs/reference/setup-tools/kubeadm/ directory (679b45e2c)

kubeadm reset

该命令尽力还原由 kubeadm init 或 kubeadm join 所做的更改。

概要

尽最大努力还原通过 'kubeadm init' 或者 'kubeadm join' 操作对主机所做的更改

"reset" 命令执行以下阶段：

```
preflight              Run reset pre-flight checks
update-cluster-status  Remove this node from the ClusterStatus object.
remove-etcd-member     Remove a local etcd member.
cleanup-node           Run cleanup node.
```

```
kubeadm reset [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"

	存储证书的目录路径。如果已指定，则需要清空此目录。
--cri-socket string	
	要连接的 CRI 套接字的路径。如果为空，则 kubeadm 将尝试自动检测此值；仅当安装了多个CRI 或具有非标准 CRI 插槽时，才使用此选项。
-f, --force	
	在不提示确认的情况下重置节点。
-h, --help	
	reset 操作的帮助命令
--ignore-preflight-errors stringSlice	
	错误将显示为警告的检查列表；例如：'IsPrivilegedUser,Swap'。取值为 'all' 时将忽略检查中的所有错误。
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。如果未设置该标志，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--skip-phases stringSlice	
	要跳过的阶段列表

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

Reset 工作流程

kubeadm reset 负责从使用 kubeadm init 或 kubeadm join 命令创建的文件中清除节点本地文件系统。对于控制平面节点，reset 还从 etcd 集群中删除该节点的本地 etcd 堆成员，还从 kubeadm ClusterStatus 对象中删除该节点的信息。 ClusterStatus 是一个 kubeadm 管理的 Kubernetes API 对象，该对象包含 kube-apiserver 端点列表。

kubeadm reset phase 可用于执行上述工作流程的各个阶段。 要跳过阶段列表，你可以使用 --skip-phases 参数，该参数的工作方式类似于 kubeadm join 和 kubeadm init 阶段运行器。

外部 etcd 清理

如果使用了外部 etcd，kubeadm reset 将不会删除任何 etcd 中的数据。这意味着，如果再次使用相同的 etcd 端点运行 kubeadm init，你将看到先前集群的状态。

要清理 etcd 中的数据，建议你使用 etcdctl 这样的客户端，例如：

```
etcdctl del "" --prefix
```

更多详情请参考 [etcd 文档]。

接下来

- 参考 kubeadm init 来初始化 Kubernetes 主节点。
- 参考 kubeadm join 来初始化 Kubernetes 工作节点并加入集群。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 27, 2020 at 5:10 PM PST: sync changes in docs/reference/setup-tools/kubeadm/ directory (679b45e2c)

kubeadm token

如使用引导令牌进行身份验证所描述的，引导令牌用于在即将加入集群的节点和主节点间建立双向认证。

kubeadm init 创建了一个有效期为 24 小时的令牌，下面的命令允许你管理令牌，也可以创建和管理新的令牌。

kubeadm token create

概要

这个命令将为你创建一个引导令牌。 您可以设置此令牌的用途，"有效时间" 和可选的人性化的描述。

这里的 [token] 是指将要生成的实际令牌。 该令牌应该是一个通过安全机制生成的随机令牌，形式为 "[a-z0-9]{6}.[a-z0-9]{16}"。 如果没有给出 [token]，kubeadm 将生成一个随机令牌。

```
kubeadm token create [token]
```

选项

--config string	
	kubeadm 配置文件的路径。
--description string	
	针对令牌用途的人性化的描述。
--groups stringSlice	默认值：[system:bootstrappers:kubeadm:default-node-token]

	此令牌用于身份验证时将进行身份验证的其他组。必须匹配 "\\system:bootstrappers:[a-z0-9:-]{0,255}[a-z0-9]\\z"
-h, --help	
	create 操作的帮助命令
--print-join-command	
	不仅仅打印令牌，而是打印使用令牌加入集群所需的完整 'kubeadm join' 参数。
--ttl duration 默认值：24h0m0s	
	令牌有效时间，超过该时间令牌被自动删除。(例如： 1s, 2m, 3h)。如果设置为 '0'，令牌将永远不过期。
--usages stringSlice 默认值：[signing,authentication]	
	描述可以使用此令牌的方式。你可以多次使用 `--usages` 或者提供一个以逗号分隔的选项列表。合法选项有: [signing,authentication]

从父命令继承的选项

--dry-run
是否启用 `dry-run` 运行模式
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"
用于和集群通信的 KubeConfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 KubeConfig 文件。
--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm token delete

概要

这个命令将为你删除指定的引导令牌列表。

[token-value] 是要删除的 "[a-z0-9]{6}.[a-z0-9]{16}" 形式的完整令牌或者是 "[a-z0-9]{6}" 形式的的令牌 ID。

```
kubeadm token delete [token-value] ...
```

选项

-h, --help
delete 操作的帮助命令

从父命令继承的选项

--dry-run
是否启用 `dry-run` 运行模式
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"

	用于和集群通信的 KubeConfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 KubeConfig 文件。
--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm token generate

概要

此命令将打印一个随机生成的可以被 "init" 和 "join" 命令使用的引导令牌。 您不必使用此命令来生成令牌。你可以自己设定，只要格式符合 "[a-z0-9]{6}.[a-z0-9]{16}"。这个命令提供是为了方便生成规定格式的令牌。 您也可以使用 "kubeadm init" 并且不指定令牌，该命令会生成一个令牌并打印出来。

```
kubeadm token generate [flags]
```

选项

-h, --help	
	generate 操作的帮助命令

从父命令继承的选项

--dry-run	
	是否启用 `dry-run` 运行模式
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"	
	用于和集群通信的 KubeConfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 KubeConfig 文件。
--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm token list

概要

此命令将为您列出所有的引导令牌。

```
kubeadm token list [flags]
```

选项

--allow-missing-template-keys 默认值：true	
	如果设置为 true，则在模板中缺少字段或哈希表的键时忽略模板中的任何错误。 仅适用于 golang 和 jsonpath 输出格式。
-o, --experimental-output string 默认值："text"	

	输出格式：text|json|yaml|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file 其中之一
-h, --help	
	list 操作的帮助命令

从父命令继承的选项

--dry-run	
	是否启用 `dry-run` 模式
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	用于和集群通信的 kubeconfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 kubeconfig 文件。
--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

接下来

- kubeadm join 引导 Kubernetes 工作节点并将其加入集群

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 27, 2020 at 5:10 PM PST: sync changes in docs/reference/setup-tools/kubeadm/ directory (679b45e2c)

kubeadm version

此命令用来输出 kubeadm 的版本。

概要

打印 kubeadm 的版本

```
kubeadm version [flags]
```

选项

-h, --help	
	version 操作的帮助命令

-o, --output string
输出格式；可用的选项有 'yaml', 'json' 和 'short'

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 14, 2020 at 4:10 PM PST: [zh] Sync changes to kubeadm reference (e944fb147)

kubeadm alpha

> **注意：**
>
> kubeadm alpha 提供了一组可用于收集社区反馈的预览性质功能。 请试用这些功能并给我们提供反馈！

kubeadm alpha kubeconfig user

使用子命令 user 为其他用户创建 kubeconfig 文件。

- kubeconfig
- user

概要

kubeconfig 文件应用程序。

Alpha 免责声明：此命令当前为 alpha 功能。

选项

-h, --help
kubeconfig 操作的帮助命令

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

为其他用户输出 kubeconfig 文件。

Alpha 免责声明：此命令当前为 Alpha 功能。

```
kubeadm alpha kubeconfig user [flags]
```

示例

```
# 使用名为 bar 的 kubeadm 配置文件为名为 foo 的另一用户输出 kubeconfig 文件
kubeadm alpha kubeconfig user --client-name=foo --config=bar
```

选项

--client-name string
用户名。如果生成客户端证书，则用作其 CN。
--config string
指向 kubeadm 配置文件的路径
-h, --help
user 操作的帮助命令
--org stringSlice
客户端证书的组织。如果创建客户端证书，此值将用作其 O 字段值。
--token string
应该用此 kubeconfig 的身份验证机制的令牌，而不是客户端证书

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机的根目录。

kubeadm alpha kubelet config

使用以下命令启用 DynamicKubeletConfiguration 功能。

- kubelet
- enable-dynamic

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

选项

-h, --help
kubelet 操作的帮助命令

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机的根目录。

概要

针对集群中的 kubelet-config-1.X ConfigMap 启用或更新节点的动态 kubelet 配置，其中 X 是所需 kubelet 版本的次要版本。

警告：此功能仍处于试验阶段，默认情况下处于禁用状态。仅当知道自己在做什么时才启用它，因为在此阶段它可能会产生令人惊讶的副作用。

Alpha 免责声明：此命令当前为 Alpha 功能。

```
kubeadm alpha kubelet config enable-dynamic [flags]
```

示例

```
  # 为节点启用动态 kubelet 配置。
  kubeadm alpha phase kubelet enable-dynamic-config --node-name node-1 --
kubelet-version 1.16.0

  WARNING: This feature is still experimental, and disabled by default. Enable
only if you know what you are doing, as it
  may have surprising side-effects at this stage.
```

选项

-h, --help
enable-dynamic 操作的帮助命令
--kubeconfig string　　默认值: "/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。如果未设置该标志，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--kubelet-version string
kubelet 所需版本
--node-name string
应该启用动态 kubelet 配置节点的名称

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机的根目录。

kubeadm alpha selfhosting pivot

子命令 pivot 可用于将 Pod 托管的静态控制平面转换为自托管的控制平面。 有关 pivot 更多信息，请参见 文档。

- selfhosting
- pivot

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

选项

-h, --help
selfhosting 操作的帮助命令

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机的根目录。

概要

将用于控制平面组件的静态 Pod 文件转换为通过 Kubernetes API 配置的自托管 DaemonSet。

有关自托管的限制，请参阅相关文档。

Alpha 免责声明：此命令当前为 alpha 功能。

```
kubeadm alpha selfhosting pivot [flags]
```

示例

```
# 将静态 Pod 托管的控制平面转换为自托管的控制平面。

kubeadm alpha phase self-hosting convert-from-staticpods
```

选项

--cert-dir string　　默认值："/etc/kubernetes/pki"

	证书存储的路径
--config string	
	kubeadm 配置文件的路径。
-f, --force	
	在不提示确认的情况下转换集群
-h, --help	
	pivot 操作的帮助命令
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
-s, --store-certs-in-secrets	
	启用 secret 存储证书

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

接下来

- 用来启动引导 Kubernetes 控制平面节点的 kubeadm init 命令
- 用来将节点连接到集群的 kubeadm join 命令
- 用来还原 kubeadm init 或 kubeadm join 操作对主机所做的任何更改的 kubeadm reset 命令

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 14, 2020 at 1:25 PM PST: [zh] Sync kubeadm certs command reference (9c2547068)

kubeadm certs

kubeadm certs 提供管理证书的工具。关于如何使用这些命令的细节，可参见 使用 kubeadm 管理证书。

kubeadm certs

用来操作 Kubernetes 证书的一组命令。

- 概览

概要

与处理 kubernetes 证书相关的命令

选项

-h, --help
certs 命令的帮助

继承于父命令的选项

--rootfs string
[实验] 到'真实'主机根文件系统的路径。

kubeadm certs renew

你可以使用 all 子命令来续订所有 Kubernetes 证书，也可以选择性地续订部分证书。更多的相关细节，可参见 手动续订证书。

- renew
- all
- admin.conf
- apiserver-etcd-client
- apiserver-kubelet-client
- apiserver
- controller-manager.conf
- etcd-healthcheck-client
- etcd-peer
- etcd-server
- front-proxy-client
- scheduler.conf

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

```
kubeadm certs renew [flags]
```

选项

-h, --help
renew 操作的帮助命令

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

续订运行控制平面所需的所有已知证书。续订是无条件进行的，与到期日期无关。续订也可以单独运行以进行更多控制。

```
kubeadm certs renew all [flags]
```

选项

--cert-dir string 默认值："/etc/kubernetes/pki"
存储证书的路径。
--config string
kubeadm 配置文件的路径。
--csr-dir string
输出 CSR 和私钥的路径
--csr-only
创建 CSR 而不是生成证书
-h, --help
all 操作的帮助命令
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api
使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

续订 kubeconfig 文件中嵌入的证书，供管理员 和 kubeadm 自身使用。

无论证书的到期日期如何，续订都是无条件进行的；SAN 等额外属性将基于现有文件/证书，因此无需重新提供它们。

默认情况下，续订会尝试使用由 kubeadm 管理的本地 PKI 中的证书机构；作为替代方案， 也可以使用 K8s 证书 API 进行证书续订，或者（作为最后一种选择）生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防证书文件在其他地方使用。

```
kubeadm certs renew admin.conf [flags]
```

选项

--cert-dir string	默认值："/etc/kubernetes/pki"
	保存证书的路径。
--config string	
	kubeadm 配置文件的路径。
--csr-dir string	
	CSR 和私钥的输出路径
--csr-only	
	创建 CSR 而不是生成证书
-h, --help	
	admin.conf 子操作的帮助命令
--kubeconfig string	Default: "/etc/kubernetes/admin.conf"
	与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api	
	使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

续订 apiserver 用于访问 etcd 的证书。

无论证书的到期日期如何，续订都会无条件地进行；SAN 等额外属性将基于现有文件/证书，因此无需重新提供它们。

默认情况下，续订尝试使用在 kubeadm 所管理的本地 PKI 中的证书颁发机构；作为替代方案， 可以使用 K8s 证书 API 进行证书更新，或者作为最后一个选择来生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防文件在其他地方使用。

```
kubeadm certs renew apiserver-etcd-client [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"	
	存储证书的路径。
--config string	
	kubeadm 配置文件的路径。
--csr-dir string	
	输出 CSR 和私钥的路径
--csr-only	
	创建 CSR 而不是生成证书
-h, --help	
	apiserver-etcd-client 操作的帮助命令
--kubeconfig string　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api	
	使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

续订 apiserver 用于连接 kubelet 的证书。

无论证书的到期日期如何，续订都会无条件地进行；SAN 等额外属性将基于现有文件/证书，因此无需重新提供它们。

默认情况下，续订尝试使用位于 kubeadm 所管理的本地 PKI 中的证书颁发机构；作为替代方案， 也可能调用 K8s 证书 API 进行证书更新；亦或者，作为最后一个选择，生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防文件在其他地方使用。

```
kubeadm certs renew apiserver-kubelet-client [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"	
	存储证书的路径。
--config string	
	kubeadm 配置文件的路径。

--csr-dir string
输出 CSR 和私钥的路径
--csr-only
创建 CSR 而不是生成证书
-h, --help
apiserver-kubelet-client 操作的帮助命令
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api
使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

续订用于提供 Kubernetes API 的证书。

无论证书的到期日期如何，续订都会无条件地进行；SAN 等额外属性将基于现有文件/证书，因此无需重新提供它们。

默认情况下，续订尝试在 kubeadm 管理的本地 PKI 中使用证书颁发机构；作为替代方案， 可以使用 K8s 证书 API 进行证书更新，或者作为最后一个选择来生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防文件在其他地方使用。

```
kubeadm certs renew apiserver [flags]
```

选项

--cert-dir string 默认值："/etc/kubernetes/pki"
保存证书的路径。
--config string
kubeadm 配置文件的路径。
--csr-dir string
CSR 和私钥的输出路径
--csr-only
创建 CSR 而不是生成证书
-h, --help
apiserver 子操作的帮助命令
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"

	与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api	
	使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

续订 kubeconfig 文件中嵌入的证书，以供控制器管理器（Controller Manager）使用。

续订无条件地进行，与证书的到期日期无关；SAN 等额外属性将基于现有的文件/证书，因此无需重新提供它们。

默认情况下，续订会尝试使用 kubeadm 管理的本地 PKI 中的证书颁发机构；作为替代方案， 可以使用 K8s 证书 API 进行证书续订；亦或者，作为最后一种选择，生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防文件在其他地方使用。

```
kubeadm alpha renew controller-manager.conf [flags]
```

选项

--cert-dir string　　默认值："/etc/kubernetes/pki"	
	保存证书的路径。
--config string	
	kubeadm 配置文件的路径。
--csr-dir string	
	CSR 和私钥的输出路径
--csr-only	
	创建 CSR 而不是生成证书
-h, --help	
	controller-manager.conf 操作的帮助命令
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api	
	使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

续订存活态探针的证书，用于对 etcd 执行健康检查。

无论证书的到期日期如何，续订都是无条件进行的；SAN 等额外属性将基于现有文件/证书，因此无需重新提供它们。

默认情况下，续订会尝试使用由 kubeadm 管理的本地 PKI 中的证书机构；作为替代方案， 也可以使用 K8s certificate API 进行证书续订，或者（作为最后一种选择）生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防证书文件在其他地方使用。

```
kubeadm certs renew etcd-healthcheck-client [flags]
```

选项

--cert-dir string 默认值："/etc/kubernetes/pki"
保存证书的路径。
--config string
kubeadm 配置文件的路径。
--csr-dir string
CSR 和私钥的输出路径
--csr-only
创建 CSR 而不是生成证书
-h, --help
etcd-healthcheck-client 操作的帮助命令
--kubeconfig string 默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api
使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

续订 etcd 节点间用来相互通信的证书。

无论证书的到期日期如何，续订都是无条件进行的；SAN 等额外属性将基于现有文件/证书，因此无需重新提供它们。

默认情况下，续订会尝试使用由 kubeadm 管理的本地 PKI 中的证书机构； 作为替代方案，也可以使用 K8s certificate API 进行证书续订，或者（作为最后一种选择）生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防证书文件在其他地方使用。

```
kubeadm certs renew etcd-peer [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"	
	保存证书的路径。
--config string	
	kubeadm 配置文件的路径。
--csr-dir string	
	CSR 和私钥的输出路径
--csr-only	
	创建 CSR 而不是生成证书
-h, --help	
	etcd-peer 操作的帮助命令
--kubeconfig string　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api	
	使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

续订用于提供 etcd 服务的证书。

续订无条件地进行，与证书的到期日期无关；SAN 等额外属性将基于现有的文件/证书，因此无需重新提供它们。

默认情况下，续订会尝试在 kubeadm 管理的本地 PKI 中使用证书颁发机构；作为替代方案， 可以使用 K8s 证书 API 进行证书续订，或者作为最后一种选择来生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防文件在其他地方使用。

```
kubeadm certs renew etcd-server [flags]
```

选项

--cert-dir string	默认值："/etc/kubernetes/pki"
	保存证书的路径。
--config string	
	kubeadm 配置文件的路径。
--csr-dir string	
	CSR 和私钥的输出路径
--csr-only	
	创建 CSR 而不是生成证书
-h, --help	
	etcd-server 操作的帮助命令
--kubeconfig string	默认值："/etc/kubernetes/admin.conf"
	与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api	
	使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

为前端代理客户端续订证书。

无论证书的到期日期如何，续订都会无条件地进行；SAN 等额外属性将基于现有文件/证书，因此无需重新提供它们。

默认情况下，续订尝试使用位于 kubeadm 所管理的本地 PKI 中的证书颁发机构；作为替代方案， 也可以使用 K8s 证书 API 进行证书续订；亦或者，作为最后一种方案，生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防文件在其他地方使用。

```
kubeadm certs renew front-proxy-client [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"
存储证书的路径。
--config string
kubeadm 配置文件的路径。
--csr-dir string
输出 CSR 和私钥的路径
--csr-only
创建 CSR 而不是生成证书
-h, --help
front-proxy-client 操作的帮助命令
--kubeconfig string　默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api
使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

续订 kubeconfig 文件中嵌入的证书，以供调度管理器使用。

续订无条件地进行，与证书的到期日期无关；SAN 等额外属性将基于现有的文件/证书，因此无需重新提供它们。

默认情况下，续订会尝试使用在 kubeadm 所管理的本地 PKI 中的证书颁发机构；作为替代方案， 也可以使用 K8s 证书 API 进行证书续订；亦或者，作为最后一种选择，生成 CSR 请求。

续订后，为了使更改生效，需要重新启动控制平面组件，并最终重新分发更新的证书，以防文件在其他地方使用。

```
kubeadm certs renew scheduler.conf [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"
保存证书的路径。
--config string
kubeadm 配置文件的路径。

--csr-dir string	
	CSR 和私钥的输出路径
--csr-only	
	创建 CSR 而不是生成证书
-h, --help	
	scheduler.conf 操作的帮助命令
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。 如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--use-api	
	使用 Kubernetes 证书 API 续订证书

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm certs certificate-key

此命令可用来生成一个新的控制面证书密钥。密钥可以作为 `--certificate-key` 标志的取值传递给 kubeadm init 和 kubeadm join 命令，从而在添加新的控制面节点时能够自动完成证书复制。

- certificate-key

概要

该命令将打印出可以与 "init" 命令一起使用的安全的随机生成的证书密钥。

你也可以使用 kubeadm init `--upload-certs` 而无需指定证书密钥； 命令将为你生成并打印一个证书密钥。

```
kubeadm certs certificate-key [flags]
```

选项

-h, --help	
	certificate-key 操作的帮助命令

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm certs check-expiration

此命令检查 kubeadm 所管理的本地 PKI 中的证书是否以及何时过期。 更多的相关细节，可参见 检查证书过期。

- check-expiration

概要

检查 kubeadm 管理的本地 PKI 中证书的到期时间。

```
kubeadm certs check-expiration [flags]
```

选项

--cert-dir string	默认值: "/etc/kubernetes/pki"
	保存证书的路径
--config string	
	kubeadm 配置文件的路径
-h, --help	
	check-expiration 的帮助命令

继承于父命令的选项

--rootfs string	
	[实验] 到'真实'主机根文件系统的路径。

kubeadm certs generate-csr

此命令可用来为所有控制面证书和 kubeconfig 文件生成密钥和 CSR（签名请求）。 用户可以根据自身需要选择 CA 为 CSR 签名。

- generate-csr

为运行控制平面所需的所有证书生成密钥和证书签名请求（CSR）。该命令会生成部分 kubeconfig 文件， 其中 "users > user > client-key-data" 字段包含私钥数据，并为每个 kubeconfig 文件创建一个随附的 ".csr" 文件。

此命令设计用于 Kubeadm 外部 CA 模式。 它生成你可以提交给外部证书颁发机构进行签名的 CSR。

应使用 ".crt" 作为文件扩展名将 PEM 编码的签名证书与密钥文件一起保存。 或者，对于 kubeconfig 文件，PEM 编码的签名证书应使用 base64 编码， 并添加到 "users > user > client-certificate-data" 字段。

```
kubeadm certs generate-csr [flags]
```

示例

```
# 以下命令将为所有控制平面证书和 kubeconfig 文件生成密钥和 CSR：
kubeadm certs generate-csr --kubeconfig-dir /tmp/etc-k8s --cert-dir /tmp/etc-k8s/pki
```

选项

--cert-dir string	
	保存证书的路径
--config string	
	kubeadm 配置文件的路径。
-h, --help	
	generate-csr 命令的帮助
--kubeconfig-dir string	默认值："/etc/kubernetes"
	保存 kubeconfig 文件的路径。

继承于父命令的选项

--rootfs string	
	[实验] 到'真实'主机根文件系统的路径。

接下来

- 用来启动引导 Kubernetes 控制面节点的 kubeadm init 命令
- 用来将节点连接到集群的 kubeadm join 命令
- 用来回滚 kubeadm init 或 kubeadm join 对当前主机所做修改的 kubeadm reset 命令

反馈

最后修改 December 14, 2020 at 1:25 PM PST: [zh] Sync kubeadm certs command reference (9c2547068)

kubeadm init phase

kubeadm init phase 能确保调用引导过程的原子步骤。因此，如果希望自定义应用，则可以让 kubeadm 做一些工作，然后填补空白。

kubeadm init phase 与 kubeadm init 工作流一致，后台都使用相同的代码。

kubeadm init phase preflight

使用此命令可以在控制平面节点上执行启动前检查。

- preflight

概要

运行 kubeadm init 前的启动检查。

```
kubeadm init phase preflight [flags]
```

案例

```
# 使用配置文件对 kubeadm init 进行启动检查。
kubeadm init phase preflight --config kubeadm-config.yml
```

选项

--config string	
	kubeadm 配置文件的路径。
-h, --help	
	preflight 操作的帮助命令
--ignore-preflight-errors stringSlice	
	错误将显示为警告的检查列表：例如：'IsPrivilegedUser,Swap'。取值为 'all' 时将忽略检查中的所有错误。

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase kubelet-start

此阶段将检查 kubelet 配置文件和环境文件，然后启动 kubelet。

- kubelet-start

概要

使用 kubelet 配置文件编写一个文件，并使用特定节点的 kubelet 设置编写一个环境文件，然后（重新）启动 kubelet。

```
kubeadm init phase kubelet-start [flags]
```

示例

```
# 从 InitConfiguration 文件中写入带有 kubelet 参数的动态环境文件。
kubeadm init phase kubelet-start --config config.yaml
```

选项

--config string	
	kubeadm 配置文件的路径。
--cri-socket string	
	连接到 CRI 套接字的路径。如果为空，则 kubeadm 将尝试自动检测该值；仅当安装了多个 CRI 或具有非标准 CRI 套接字时，才使用此选项。
-h, --help	
	kubelet-start 操作的帮助命令
--node-name string	
	指定节点名称。

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase certs

该阶段可用于创建 kubeadm 所需的所有证书。

- certs
- all
- ca
- apiserver
- apiserver-kubelet-client
- front-proxy-ca
- front-proxy-client
- etcd-ca
- etcd-server
- etcd-peer
- healthcheck-client
- apiserver-etcd-client
- sa

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

```
kubeadm init phase certs [flags]
```

选项

-h, --help
certs 操作的帮助命令

从父指令中继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成所有证书

```
kubeadm init phase certs all [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，将使用默认网络接口。
--apiserver-cert-extra-sans stringSlice
用于 API 服务器服务证书的可选额外替代名称（SAN）。可以同时使用 IP 地址和 DNS 名称。
--cert-dir string 默认值："/etc/kubernetes/pki"
证书的存储路径。
--config string
kubeadm 配置文件的路径。
--control-plane-endpoint string
为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help
all 操作的帮助命令
--kubernetes-version string 默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。
--service-cidr string 默认值："10.96.0.0/12"
VIP 服务使用其它的 IP 地址范围。
--service-dns-domain string 默认值："cluster.local"
服务使用其它的域名，例如："myorg.internal"。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成自签名的 Kubernetes CA 以提供其他 Kubernetes 组件的身份，并将其保存到 ca.cert 和 ca.key 文件中。

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 Alpha 功能。

```
kubeadm init phase certs ca [flags]
```

选项

--cert-dir string　　默认值："/etc/kubernetes/pki"
证书的存储路径。
--config string
kubeadm 配置文件的路径。
-h, --help
ca 操作的帮助命令
--kubernetes-version string　　默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成用于服务 Kubernetes API 的证书，并将其保存到 apiserver.cert 和 apiserver.key 文件中。

默认 SAN 是 kubernetes、kubernetes.default、kubernetes.default.svc、kubernetes.default.svc.cluster.local、10.96.0.1、127.0.0.1。

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 Alpha 功能。

```
kubeadm init phase certs apiserver [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，则使用默认的网络接口。
--apiserver-cert-extra-sans stringSlice

	用于 API Server 服务证书的可选附加主体备用名称（SAN）。可以是 IP 地址和 DNS 名称。
--cert-dir string　默认值："/etc/kubernetes/pki"	
	证书的存储路径。
--config string	
	kubeadm 配置文件的路径。
--control-plane-endpoint string	
	为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help	
	apiserver 操作的帮助命令
--kubernetes-version string　默认值："stable-1"	
	为控制平面指定特定的 Kubernetes 版本。
--service-cidr string　默认值："10.96.0.0/12"	
	指定服务 VIP 可使用的其他 IP 地址段。
--service-dns-domain string　默认值："cluster.local"	
	为服务使用其他域名，例如 "myorg.internal"。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成供 API 服务器连接 kubelet 的证书，并将其保存到 apiserver-kubelet-client.cert 和 apiserver-kubelet-client.key 文件中。

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 Alpha 功能。

```
kubeadm init phase certs apiserver-kubelet-client [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"
存储证书的路径。
--config string
kubeadm 配置文件路径。
-h, --help
apiserver-kubelet-client 操作的帮助命令
--kubernetes-version string　默认值："stable-1"
为控制平面指定特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string
[实验] 指向宿主机上的 '实际' 根文件系统的路径。

概要

生成自签名 CA 来提供前端代理的身份，并将其保存到 front-proxy-ca.cert 和 front-proxy-ca.key 文件中。

如果两个文件都已存在，kubeadm 将跳过生成步骤并将使用现有文件。

Alpha 免责声明：此命令目前是 alpha 阶段。

```
kubeadm init phase certs front-proxy-ca [flags]
```

选项

--cert-dir string 默认值："/etc/kubernetes/pki"
存储证书的路径。
--config string
kubeadm 配置文件的路径。
-h, --help
front-proxy-ca 操作的帮助命令
--kubernetes-version string 默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

为前端代理客户端生成证书，并将其保存到 front-proxy-client.cert 和 front-proxy-client.key 文件中。 如果两个文件都已存在，kubeadm 将跳过生成步骤并将使用现有文件。 Alpha 免责声明：此命令目前是 alpha 阶段。

```
kubeadm init phase certs front-proxy-client [flags]
```

选项

--cert-dir string 默认："/etc/kubernetes/pki"
存储证书的路径。
--config string
kubeadm 配置文件的路径。

-h, --help
front-proxy-client 操作的帮助命令
--kubernetes-version string 默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成用于为 etcd 设置身份的自签名 CA，并将其保存到 etcd/ca.cert 和 etcd/ca.key 文件中。

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 Alpha 功能。

```
kubeadm init phase certs etcd-ca [flags]
```

选项

--cert-dir string 默认值："/etc/kubernetes/pki"
证书的存储路径。
--config string
kubeadm 配置文件的路径。
-h, --help
etcd-ca 操作的帮助命令
--kubernetes-version string 默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成用于提供 etcd 服务的证书，并将其保存到 etcd/server.cert 和 etcd/server.key 文件中。

默认 SAN 为 localhost、127.0.0.1、127.0.0.1、:: 1

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 alpha 功能。

```
kubeadm init phase certs etcd-server [flags]
```

选项

--cert-dir string	默认值：/etc/kubernetes/pki"
	保存和存储证书的路径。
--config string	
	kubeadm 配置文件的路径。
-h, --help	
	etcd-server 操作的帮助命令
--kubernetes-version string	默认值：stable-1"
	为控制平面指定特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

生成 etcd 节点相互通信的证书，并将其保存到 etcd/peer.cert 和 etcd/peer.key 文件中。

默认 SAN 为 localhost、127.0.0.1、127.0.0.1、:: 1

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 alpha 功能。

```
kubeadm init phase certs etcd-peer [flags]
```

选项

--cert-dir string	默认值：/etc/kubernetes/pki"
	保存和存储证书的路径。
--config string	
	kubeadm 配置文件的路径。
-h, --help	
	etcd-peer 操作的帮助命令
--kubernetes-version string	默认值：stable-1"
	为控制平面指定特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string

	[实验] 到 '真实' 主机根文件系统的路径。

概要

生成用于 etcd 健康检查的活跃性探针的证书，并将其保存到 healthcheck-client.cert 和 etcd/healthcheck-client.key 文件中。

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 alpha 功能。

```
kubeadm init phase certs etcd-healthcheck-client [flags]
```

选项

--cert-dir string	默认值："/etc/kubernetes/pki"
	证书存储的路径。
--config string	
	kubeadm 配置文件的路径。
-h, --help	
	etcd-healthcheck-client 操作的帮助命令
--kubernetes-version string	默认值："stable-1"
	为控制平面选择特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

生成 apiserver 用于访问 etcd 的证书，并将其保存到 apiserver-etcd-client.cert 和 apiserver-etcd-client.key 文件中。

如果两个文件都已存在，则 kubeadm 将跳过生成步骤，使用现有文件。

Alpha 免责声明：此命令当前为 Alpha 功能。

```
kubeadm init phase certs apiserver-etcd-client [flags]
```

选项

--cert-dir string	默认值："/etc/kubernetes/pki"
	证书的存储路径。
--config string	
	kubeadm 配置文件的路径。
-h, --help	

	apiserver-etcd-client 操作的帮助命令
--kubernetes-version string　　默认值："stable-1"	
	为控制平面指定特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

生成用于签名 service account 令牌的私钥及其公钥，并将其保存到 sa.key 和 sa.pub 文件中。如果两个文件都已存在，则 kubeadm 会跳过生成步骤，而将使用现有文件。

Alpha 免责声明：此命令当前为 alpha 阶段。

```
kubeadm init phase certs sa [flags]
```

选项

--cert-dir string　　默认值："/etc/kubernetes/pki"	
	保存和存储证书的路径。
-h, --help	
	sa 操作的帮助命令

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase kubeconfig

可以通过调用 all 子命令来创建所有必需的 kubeconfig 文件，或者分别调用它们。

- kubeconfig
- all
- admin
- kubelet
- controller-manager
- scheduler

概要

此命令并非设计用来单独运行。请阅读可用子命令列表。

```
kubeadm init phase kubeconfig [flags]
```

选项

-h, --help
kubeconfig 操作的帮助命令

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成所有 kubeconfig 文件

```
kubeadm init phase kubeconfig all [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果没有设置，将使用默认的网络接口。
--apiserver-bind-port int32　默认值：6443
要绑定到 API 服务器的端口。
--cert-dir string　默认值："/etc/kubernetes/pki"
保存和存储证书的路径。
--config string
kubeadm 配置文件的路径。
--control-plane-endpoint string
为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help
all 操作的帮助命令
--kubeconfig-dir string　默认值："/etc/kubernetes"
kubeconfig 文件的保存路径。
--kubernetes-version string　默认值："stable-1"
为控制平面指定特定的 Kubernetes 版本。
--node-name string
指定节点名称。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

为管理员和 kubeadm 本身生成 kubeconfig 文件，并将其保存到 admin.conf 文件中。

```
kubeadm init phase kubeconfig admin [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，则使用默认的网络接口。
--apiserver-bind-port int32　默认值：6443
要绑定到 API 服务器的端口。
--cert-dir string　默认值："/etc/kubernetes/pki"
保存和存储证书的路径。
--config string
kubeadm 配置文件的路径。
--control-plane-endpoint string
为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help
admin 操作的帮助命令
--kubeconfig-dir string　默认值："/etc/kubernetes"
kubeconfig 文件的保存路径。
--kubernetes-version string　默认值："stable-1"
为控制平面指定特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成 kubelet 要使用的 kubeconfig 文件，并将其保存到 kubelet.conf 文件。

请注意，该操作目的是仅应用于引导集群。在控制平面启动之后，应该从 CSR API 请求所有 kubelet 凭据。

```
kubeadm init phase kubeconfig kubelet [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，则使用默认的网络接口。
--apiserver-bind-port int32　默认值：6443

	要绑定到 API 服务器的端口。
--cert-dir string　默认值：/etc/kubernetes/pki"	
	保存和存储证书的路径。
--config string	
	kubeadm 配置文件的路径。
--control-plane-endpoint string	
	为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help	
	kubelet 操作的帮助命令
--kubeconfig-dir string　默认值：/etc/kubernetes"	
	kubeconfig 文件的保存路径。
--kubernetes-version string　默认值："stable-1"	
	为控制平面选择特定的 Kubernetes 版本。
--node-name string	
	指定节点的名称。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成控制器管理器要使用的 kubeconfig 文件，并保存到 controller-manager.conf 文件中。

```
kubeadm init phase kubeconfig controller-manager [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，则使用默认的网络接口。
--apiserver-bind-port int32　默认值：6443
要绑定到 API 服务器的端口。
--cert-dir string　默认值：/etc/kubernetes/pki"
保存和存储证书的路径。
--config string
kubeadm 配置文件的路径。
--control-plane-endpoint string
为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help
controller-manager 操作的帮助命令
--kubeconfig-dir string　默认值：/etc/kubernetes"

	kubeconfig 文件的保存路径。
--kubernetes-version string　默认值：\"stable-1\"	
	为控制平面指定特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs 字符串
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成调度器（scheduler）要使用的 kubeconfig 文件，并保存到 scheduler.conf 文件中。

```
kubeadm init phase kubeconfig scheduler [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，则使用默认的网络接口。
--apiserver-bind-port int32　默认值：6443
要绑定到 API 服务器的端口。
--cert-dir string　默认值：\"/etc/kubernetes/pki\"
保存和存储证书的路径。
--config string
kubeadm 配置文件的路径。
--control-plane-endpoint string
为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help
scheduler 操作的帮助命令
--kubeconfig-dir string　默认值：\"/etc/kubernetes\"
kubeconfig 文件的保存路径。
--kubernetes-version string　默认值：\"stable-1\"
为控制平面指定特定的 Kubernetes 版本。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase control-plane

使用此阶段，可以为控制平面组件创建所有必需的静态 Pod 文件。

- control-plane
- all
- apiserver
- controller-manager
- scheduler

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

```
kubeadm init phase control-plane [flags]
```

选项

-h, --help
control-plane 操作的帮助命令

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成所有的静态 Pod 清单文件

```
kubeadm init phase control-plane all [flags]
```

示例

```
# 为控制平面组件生成静态 Pod 清单文件，其功能等效于 kubeadm init 生成的文件。
kubeadm init phase control-plane all

# 使用从某配置文件中读取的选项为生成静态 Pod 清单文件。
kubeadm init phase control-plane all --config config.yaml
```

选项

--apiserver-advertise-address string	
API 服务器所公布的其正在监听的 IP 地址。如果未设置，将使用默认的网络接口。	
--apiserver-bind-port int32	默认值：6443
API 服务器要绑定的端口。	
--apiserver-extra-args mapStringString	

	形式为 <flagname>=<value> 的一组额外参数，用来传递给 API 服务器， 或者覆盖其默认配置值
--cert-dir string　默认值："/etc/kubernetes/pki"	
	存储证书的路径。
--config string	
	kubeadm 配置文件的路径。
--control-plane-endpoint string	
	为控制平面选择一个稳定的 IP 地址或者 DNS 名称。
--controller-manager-extra-args mapStringString	
	一组形式为 <flagname>=<value> 的额外参数，用来传递给控制管理器（Controller Manager）或覆盖其默认设置值
--experimental-patches string	
	包含名为 "target[suffix][+patchtype].extension" 的文件的目录。 例如，"kube-apiserver0+merge.yaml" 或者 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，分别与 kubectl 所支持的 patch 格式相匹配。默认的 "patchtype" 是 "strategic"。 "extension" 必须是 "json" 或 "yaml"。 "suffix" 是一个可选的字符串，用来确定按字母顺序排序时首先应用哪些 patch。
--feature-gates string	
	一组用来描述各种特性门控的键值（key=value）对。选项是： IPv6DualStack=true|false (ALPHA - 默认=false) PublicKeysECDSA=true|false (ALPHA - 默认=false)
-h, --help	
	all 操作的帮助命令
--image-repository string　默认值："k8s.gcr.io"	
	选择用于拉取控制平面镜像的容器仓库
--kubernetes-version string　默认值："stable-1"	
	为控制平面选择指定的 Kubernetes 版本。
--pod-network-cidr string	
	指定 Pod 网络的 IP 地址范围。如果设置了此标志，控制平面将自动地为每个节点分配 CIDR。
--scheduler-extra-args mapStringString	
	一组形式为 <flagname>=<value> 的额外参数，用来传递给调度器（Scheduler）或覆盖其默认设置值 传递给调度器（scheduler）一组额外的参数或者以 <flagname>=<value> 形式覆盖其默认值。
--service-cidr string　默认值："10.96.0.0/12"	
	为服务 VIP 选择 IP 地址范围。

从父指令继承的选项

--rootfs string

	[实验] 指向 '真实' 宿主机的根文件系统的路径。

概要

生成 kube-apiserver 静态 Pod 清单

```
kubeadm init phase control-plane apiserver [flags]
```

选项

--apiserver-advertise-address string	
	API 服务器所公布的其正在监听的 IP 地址。如果未设置，将使用默认网络接口。
--apiserver-bind-port int32　　默认值： 6443	
	要绑定到 API 服务器的端口。
--apiserver-extra-args mapStringString	
	一组 <flagname>=<value> 形式的额外参数，用来传递给 API 服务器 或者覆盖其默认参数配置
--cert-dir string　　默认值："/etc/kubernetes/pki"	
	保存和存储证书的路径。
--config string	
	kubeadm 配置文件的路径。
--control-plane-endpoint string	
	为控制平面指定一个稳定的 IP 地址或 DNS 名称。
	包含名为 "target[suffix][+patchtype].extension" 的文件的目录。 例如，"kube-apiserver0+merge.yaml" 或者 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，分别与 kubectl 所支持的 patch 格式相匹配。默认的 "patchtype" 是 "strategic"。 "extension" 必须是 "json" 或 "yaml"。 "suffix" 是一个可选的字符串，用来确定按字母顺序排序时首先应用哪些 patch。
--feature-gates string	
	一组键值对，用于描述各种功能特性的特性门控。选项是： IPv6DualStack=true|false (ALPHA - 默认=false) PublicKeysECDSA=true|false (ALPHA - 默认=false)
-h, --help	
	apiserver 操作的帮助命令
--image-repository string　　默认值："k8s.gcr.io"	
	选择要从中拉取控制平面镜像的容器仓库
--kubernetes-version string　　默认值："stable-1"	
	为控制平面选择特定的 Kubernetes 版本
--service-cidr string　　默认值："10.96.0.0/12"	
	指定服务 VIP 使用 IP 地址的其他范围。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统路径。

概要

生成 kube-controller-manager 静态 Pod 清单

```
kubeadm init phase control-plane controller-manager [flags]
```

选项

--cert-dir string　默认值："/etc/kubernetes/pki"
存储证书的路径。
--config string
kubeadm 配置文件的路径。
--controller-manager-extra-args mapStringString
一组 <flagname>=< 形式的额外参数，传递给控制器管理器（Controller Manager）或者覆盖其默认配置值
包含名为 "target[suffix][+patchtype].extension" 的文件的目录。 例如，"kube-apiserver0+merge.yaml" 或者 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，分别与 kubectl 所支持的 patch 格式相匹配。默认的 "patchtype" 是 "strategic"。 "extension" 必须是 "json" 或 "yaml"。 "suffix" 是一个可选的字符串，用来确定按字母顺序排序时首先应用哪些 patch。
-h, --help
controller-manager 操作的帮助命令
--image-repository string　默认值："k8s.gcr.io"
选择要从中拉取控制平面镜像的容器仓库
--kubernetes-version string　默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。
--pod-network-cidr string
指定 Pod 网络的 IP 地址范围。如果设置，控制平面将自动为每个节点分配 CIDR。

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

生成 kube-scheduler 静态 Pod 清单

```
kubeadm init phase control-plane scheduler [flags]
```

选项

--cert-dir string　默认值：/etc/kubernetes/pki	
	存储证书的路径。
--config string	
	kubeadm 配置文件的路径。
--experimental-patches string	
	包含名为 "target[suffix][+patchtype].extension" 的文件的目录。 例如，"kube-apiserver0+merge.yaml" 或者 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，分别与 kubectl 所支持的 patch 格式相匹配。默认的 "patchtype" 是 "strategic"。 "extension" 必须是 "json" 或 "yaml"。 "suffix" 是一个可选的字符串，用来确定按字母顺序排序时首先应用哪些 patch。
-h, --help	
	scheduler 操作的帮助命令
--image-repository string　默认值:"k8s.gcr.io"	
	选择要从中拉取控制平面镜像的容器仓库
--kubernetes-version string　默认值：stable-1"	
	为控制平面选择特定的 Kubernetes 版本。
--scheduler-extra-args mapStringString	
	一组 <flagname>=<value> 形式的额外参数，用来传递给调度器 或者覆盖其默认参数配置

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase etcd

根据静态 Pod 文件，使用以下阶段创建本地 etcd 实例。

- etcd
- local

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

```
kubeadm init phase etcd [flags]
```

选项

-h, --help	
	etcd 操作的帮助命令

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

为本地单节点 etcd 实例生成静态 Pod 清单文件

```
kubeadm init phase etcd local [flags]
```

示例

```
# 为 etcd 生成静态 Pod 清单文件，其功能等效于 kubeadm init 生成的文件。
kubeadm init phase etcd local

# 使用从配置文件读取的选项为 etcd 生成静态 Pod 清单文件。
kubeadm init phase etcd local --config config.yaml
```

选项

--cert-dir string　　默认值："/etc/kubernetes/pki"
存储证书的路径。
--config string
kubeadm 配置文件的路径。
--experimental-patches string
包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
-h, --help
local 操作的帮助命令
--image-repository string　　默认值："k8s.gcr.io"
选择要从中拉取控制平面镜像的容器仓库

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase upload-config

可以使用此命令将 kubeadm 配置文件上传到集群。或者使用 kubeadm config。

- upload-config
- all
- kubeadm
- kubelet

概要

此命令并非设计用来单独运行。请参阅可用的子命令列表。

```
kubeadm init phase upload-config [flags]
```

选项

-h, --help	
	upload-config 操作的帮助命令

从父命令中继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

将所有配置上传到 ConfigMap

```
kubeadm init phase upload-config all [flags]
```

选项

--config string	
	kubeadm 配置文件的路径。
-h, --help	
	all 操作的帮助命令
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

将 kubeadm ClusterConfiguration 上传到 kube-system 命名空间中名为 kubeadm-config 的 ConfigMap 中。 这样就可以正确配置系统组件，并在升级时提供无缝的用户体验。

另外，可以使用 kubeadm 配置。

```
kubeadm init phase upload-config kubeadm [flags]
```

示例

```
# 上传集群配置
kubeadm init phase upload-config --config=myConfig.yaml
```

选项

--config string	
	kubeadm 配置文件的路径。
-h, --help	
	kubeadm 操作的帮助命令
--kubeconfig string　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

将从 kubeadm InitConfiguration 对象提取的 kubelet 配置上传到集群中 kubelet-config-1.X 形式的 ConfigMap，其中 X 是当前（API 服务器）Kubernetes 版本的次要版本。

```
kubeadm init phase upload-config kubelet [flags]
```

示例

```
# 将 kubelet 配置从 kubeadm 配置文件上传到集群中的 ConfigMap。
kubeadm init phase upload-config kubelet --config kubeadm.yaml
```

选项

--config string	
	到 kubeadm 配置文件的路径。

-h, --help	
	kubelet 操作的帮助命令
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	与集群通信时使用的 kubeconfig 文件。如果未设置该标签，则可以通过一组标准路径来寻找已有的 kubeconfig 文件。

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase upload-certs

使用以下阶段将控制平面证书上传到集群。默认情况下，证书和加密密钥会在两个小时后过期。

- upload-certs

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

```
kubeadm init phase upload-certs [flags]
```

选项

--certificate-key string	
	用于加密 kubeadm-certs Secret 中的控制平面证书的密钥。
--config string	
	kubeadm 配置文件的路径。
-h, --help	
	upload-certs 操作的帮助命令
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	用来与集群通信的 kubeconfig 文件。 如果此标志未设置，则可以在一组标准的位置搜索现有的 kubeconfig 文件。
--skip-certificate-key-print	
	不要打印输出用于加密控制平面证书的密钥。
--upload-certs	
	将控制平面证书上传到 kubeadm-certs Secret。

从父命令继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase mark-control-plane

使用以下阶段来给具有 node-role.kubernetes.io/master="" 键值对的节点打标签（label）和记录污点（taint）。

- mark-control-plane

概要

标记 Node 节点为控制平面节点

```
kubeadm init phase mark-control-plane [flags]
```

示例

```
# 将控制平面标签和污点应用于当前节点，其功能等效于 kubeadm init执行的操作。
kubeadm init phase mark-control-plane --config config.yml

# 将控制平面标签和污点应用于特定节点
kubeadm init phase mark-control-plane --node-name myNode
```

选项

--config string	
	kubeadm 配置文件的路径。
-h, --help	
	mark-control-plane 操作的帮助命令
--node-name string	
	指定节点名称。

从父命令继承的选项

--rootfs 字符串	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase bootstrap-token

使用以下阶段来配置引导令牌。

- bootstrap-token

概要

启动引导令牌（bootstrap token）用于在即将加入集群的节点和控制平面节点之间建立双向信任。

该命令使启动引导令牌（bootstrap token）所需的所有配置生效，然后创建初始令牌。

```
kubeadm init phase bootstrap-token [flags]
```

示例

```
# 进行所有引导令牌配置，并创建一个初始令牌，功能上与 kubeadm init 生成的令牌等效。
kubeadm init phase bootstrap-token
```

选项

--config string	
	kubeadm 配置文件的路径。
-h, --help	
	bootstrap-token 操作的帮助命令
--kubeconfig string　　默认值："/etc/kubernetes/admin.conf"	
	用于和集群通信的 kubeconfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 kubeconfig 文件。
--skip-token-print	
	跳过打印 'kubeadm init' 生成的默认引导令牌。

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

kubeadm init phase kubelet-finialize

使用以下阶段在 TLS 引导后更新与 kubelet 相关的设置。 你可以使用 all 子命令来运行所有 kubelet-finalize 阶段。

- kublet-finalize
- kublet-finalize-all
- kublet-finalize-cert-rotation

TLS 引导后更新与 kubelet 相关的设置

```
kubeadm init phase kubelet-finalize [flags]
```

示例

```
# 在 TLS 引导后更新与 kubelet 相关的设置
kubeadm init phase kubelet-finalize all --config
```

选项

-h, --help

	kubelet-finalize 操作的帮助命令

继承于父命令的选项

--rootfs string
[实验] 到'真实'主机根文件系统的路径。

运行所有 kubelet-finalize 阶段

```
kubeadm init phase kubelet-finalize all [flags]
```

示例

```
# 在 TLS 引导后更新与 kubelet 相关的设置
kubeadm init phase kubelet-finalize all --config
```

选项

--cert-dir string 默认值: "/etc/kubernetes/pki"
保存和存储证书的路径。
--config string
kubeadm 配置文件的路径。
-h, --help
all 操作的帮助命令

继承于父命令的选项

--rootfs string
[实验] 到'真实'主机根文件系统的路径。

启用 kubelet 客户端证书轮换

```
kubeadm init phase kubelet-finalize experimental-cert-rotation [flags]
```

选项

--cert-dir string Default: "/etc/kubernetes/pki"
保存和存储证书的路径。
--config string
kubeadm 配置文件的路径。
-h, --help
experimental-cert-rotation 操作的帮助命令

继承于父命令的选项

--rootfs string

	[实验] 到'真实'主机根文件系统的路径。

kubeadm init phase addon

可以使用 all 子命令安装所有可用的插件，或者有选择性地安装它们。

- addon
- all
- coredns
- kube-proxy

概要

此命令并非设计用来单独运行。请参阅可用子命令列表。

```
kubeadm init phase addon [flags]
```

选项

-h, --help	
	addon 操作的帮助命令

继承于父命令的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

安装所有插件（ addon ）

```
kubeadm init phase addon all [flags]
```

选项

--apiserver-advertise-address string	
	API 服务器所公布的其正在监听的 IP 地址。如果未设置，则将使用默认网络接口。
--apiserver-bind-port int32　　默认值：6443	
	API 服务器绑定的端口。
--config string	
	kubeadm 配置文件的路径。
--control-plane-endpoint string	
	为控制平面指定一个稳定的 IP 地址或 DNS 名称。
--feature-gates string	
	一组键值对（ key=value ），描述了各种特征。选项包括： IPv6DualStack=true|false (ALPHA - 默认值=false)

-h, --help
--image-repository string　默认值："k8s.gcr.io"
选择用于拉取控制平面镜像的容器仓库
--kubeconfig string　默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--kubernetes-version string　默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。
--pod-network-cidr string
指定 Pod 网络的 IP 地址范围。如果已设置，控制平面将自动为每个节点分配 CIDR。
--service-cidr string　默认值："10.96.0.0/12"
为服务 VIP 使用 IP 地址的其他范围。
--service-dns-domain string　默认值："cluster.local"
为服务使用其他域名，例如 "myorg.internal"。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

通过 API 服务器安装 CoreDNS 附加组件。请注意，即使 DNS 服务器已部署，在安装 CNI 之前 DNS 服务器不会被调度执行。

```
kubeadm init phase addon coredns [flags]
```

选项

--config string
kubeadm 配置文件的路径。
--feature-gates string
一组用来描述各种功能特性的键值（key=value）对。选项是：IPv6DualStack=true|false (ALPHA - 默认值=false)
-h, --help
coredns 操作的帮助命令
--image-repository string　默认值："k8s.gcr.io"
选择用于拉取控制平面镜像的容器仓库
--kubeconfig string　默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--kubernetes-version string　默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。

--service-cidr string　默认值："10.96.0.0/12"
为服务 VIP 选择 IP 地址范围。
--service-dns-domain string　默认值："cluster.local"
服务使用其它的域名，例如："myorg.internal"。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

通过 API 服务器安装 kube-proxy 附加组件。

```
kubeadm init phase addon kube-proxy [flags]
```

选项

--apiserver-advertise-address string
API 服务器所公布的其正在监听的 IP 地址。如果未设置，则将使用默认网络接口。
--apiserver-bind-port int32　默认值: 6443
API 服务器绑定的端口。
--config string
kubeadm 配置文件的路径。
--control-plane-endpoint string
为控制平面指定一个稳定的 IP 地址或 DNS 名称。
-h, --help
kube-proxy 操作的帮助命令
--image-repository string　默认值："k8s.gcr.io"
选择用于拉取控制平面镜像的容器仓库
--kubeconfig string　默认值："/etc/kubernetes/admin.conf"
与集群通信时使用的 kubeconfig 文件。如果未设置该参数，则可以在一组标准位置中搜索现有的 kubeconfig 文件。
--kubernetes-version string　默认值："stable-1"
为控制平面选择特定的 Kubernetes 版本。
--pod-network-cidr string
指定 Pod 网络的 IP 地址范围。如果已设置，控制平面将自动为每个节点分配 CIDR。

继承于父命令的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

要使用 kube-dns 代替 CoreDNS，必须传递一个配置文件：

```
# 仅用于安装 DNS 插件
kubeadm init phase addon coredns --config=someconfig.yaml
# 用于创建完整的控制平面节点
kubeadm init --config=someconfig.yaml
# 用于列出或者拉取镜像
kubeadm config images list/pull --config=someconfig.yaml
# 升级
kubeadm upgrade apply --config=someconfig.yaml
```

该文件必须在 ClusterConfiguration 中包含一个 DNS 字段，以及包含一个插件的类型 - kube-dns（默认值为 CoreDNS）。

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
dns:
  type: "kube-dns"
```

有关 v1beta2 配置中每个字段的更多详细信息，可以访问 API。

接下来

- kubeadm init 引导 Kubernetes 控制平面节点
- kubeadm join 将节点连接到集群
- kubeadm reset 恢复通过 kubeadm init 或 kubeadm join 操作对主机所做的任何更改
- kubeadm alpha 尝试实验性功能

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 27, 2020 at 5:10 PM PST: sync changes in docs/ reference/setup-tools/kubeadm/ directory (679b45e2c)

kubeadm join phase

kubeadm join phase 使你能够调用 join 过程的基本原子步骤。 因此，如果希望执行自定义操作，可以让 kubeadm 做一些工作，然后由用户来补足剩余操作。

kubeadm join phase 与 kubeadm join 工作流程 一致，后台都使用相同的代码。

kubeadm join phase

- phase

概要

使用此命令来调用 join 工作流程的某个阶段

选项

-h, --help
phase 操作的帮助命令

从父命令中继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm join phase preflight

使用此命令可以在即将加入集群的节点上执行启动前检查。

- preflight

概要

运行 kubeadm join 命令添加节点前检查。

```
kubeadm join phase preflight [api-server-endpoint] [flags]
```

示例

```
# 使用配置文件运行 kubeadm join 命令添加节点前检查。
kubeadm join phase preflight --config kubeadm-config.yml
```

选项

--apiserver-advertise-address string
对于将要托管新的控制平面实例的节点，指定 API 服务器将公布的其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--apiserver-bind-port int32　　默认值：6443
针对将要托管新的控制平面实例的节点，设置 API 服务器要绑定的端口。
--certificate-key string
使用此密钥可以解密由 `init` 操作上传的证书 secret。
--config string

	kubeadm 配置文件的路径。
--control-plane	
	在此节点上创建一个新的控制平面实例
--cri-socket string	
	提供给 CRI 套接字建立连接的路径。如果为空，则 kubeadm 将尝试自动检测该值；仅当安装了多个 CRI 或具有非标准 CRI 套接字时，才使用此选项。
--discovery-file string	
	对于基于文件的发现，给出用于加载集群信息的文件或者 URL。
--discovery-token string	
	对于基于令牌的发现，该令牌用于验证从 API 服务器获取的集群信息。
--discovery-token-ca-cert-hash stringSlice	
	对于基于令牌的发现，验证根 CA 公钥是否匹配此哈希值（格式："<type>:<value>"）。
--discovery-token-unsafe-skip-ca-verification	
	对于基于令牌的发现，允许在未关联 --discovery-token-ca-cert-hash 参数的情况下添加节点。
-h, --help	
	preflight 操作的帮助命令
--ignore-preflight-errors stringSlice	
	错误将显示为警告的检查列表；例如：'IsPrivilegedUser,Swap'。取值为 'all' 时将忽略检查中的所有错误。
--node-name string	
	指定节点名称。
--tls-bootstrap-token string	
	指定在加入节点时用于临时通过 Kubernetes 控制平面进行身份验证的令牌。
--token string	
	如果未提供这些值，则将它们用于 discovery-token 令牌和 tls-bootstrap 令牌。

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm join phase control-plane-prepare

使用此阶段，你可以准备一个作为控制平面的节点。

- control-plane-prepare
- all
- download-certs
- certs
- kubeconfig
- control-plane

概要

准备为控制平面服务的机器

```
kubeadm join phase control-plane-prepare [flags]
```

示例

```
# 准备为控制平面服务的机器
kubeadm join phase control-plane-prepare all
```

选项

-h, --help
control-plane-prepare 操作的帮助命令

从父命令中继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

概要

准备为控制平面服务的机器

```
kubeadm join phase control-plane-prepare all [api-server-endpoint] [flags]
```

选项

--apiserver-advertise-address string
如果该节点托管一个新的控制平面实例，则 API 服务器将公布其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--apiserver-bind-port int32　　默认值：6443
如果该节点托管一个新的控制平面实例，则为 API 服务器要绑定的端口。
--certificate-key string
使用此密钥解密由 init 上传的证书 secrets。
--config string
kubeadm 配置文件的路径。
--control-plane
在此节点上创建一个新的控制平面实例
--discovery-file string
对于基于文件的发现，给出用于加载集群信息的文件或者 URL。
--discovery-token string
对于基于令牌的发现，该令牌用于验证从 API 服务器获取的集群信息。
--discovery-token-ca-cert-hash stringSlice

	对于基于令牌的发现，请验证根 CA 公钥是否匹配此哈希值（格式："<type>:<value>"）。
--discovery-token-unsafe-skip-ca-verification	
	对于基于令牌的发现，允许在未关联 --discovery-token-ca-cert-hash 参数的情况下添加节点。
--experimental-patches string	
	包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
-h, --help	
	all 操作的帮助命令
--node-name string	
	指定节点名称。
--tls-bootstrap-token string	
	指定在加入节点时用于临时通过 Kubernetes 控制平面进行身份验证的令牌。
--token string	
	如果未提供这些值，则将它们用于 discovery-token 令牌和 tls-bootstrap 令牌。

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

概要

[实验]从 kubeadm-certs Secret 下载控制平面节点之间共享的证书

```
kubeadm join phase control-plane-prepare download-certs [api-server-endpoint] [flags]
```

选项

--certificate-key string	
	使用此密钥可以解密由 init 上传的证书 secret。
--config string	
	kubeadm 配置文件的路径。
--control-plane	
	在此节点上创建一个新的控制平面实例
--discovery-file string	
	对于基于文件的发现，给出用于加载集群信息的文件或者 URL。
--discovery-token string	
	对于基于令牌的发现，该令牌用于验证从 API 服务器获取的集群信息。

--discovery-token-ca-cert-hash stringSlice	
	对于基于令牌的发现，请验证根 CA 公钥是否匹配此哈希值（格式："<type>:<value>"）。
--discovery-token-unsafe-skip-ca-verification	
	对于基于令牌的发现，允许在未关联 --discovery-token-ca-cert-hash 参数的情况下添加节点。
-h, --help	
	kubeconfig 操作的帮助命令
--tls-bootstrap-token string	
	指定在加入节点时用于临时通过 Kubernetes 控制平面进行身份验证的令牌。
--token string	
	如果未提供这些值，则将它们用于 discovery-token 令牌和 tls-bootstrap 令牌。

从父命令中继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

概要

为新的控制平面组件生成证书

```
kubeadm join phase control-plane-prepare certs [api-server-endpoint] [flags]
```

选项

--apiserver-advertise-address string	
	如果该节点托管一个新的控制平面实例，则 API 服务器将公布其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--config string	
	kubeadm 配置文件的路径。
--control-plane	
	在此节点上创建一个新的控制平面实例
--discovery-file string	
	对于基于文件的发现，给出用于加载集群信息的文件或者 URL。
--discovery-token string	
	对于基于令牌的发现，该令牌用于验证从 API 服务器获取的集群信息。
--discovery-token-ca-cert-hash stringSlice	
	对于基于令牌的发现，请验证根 CA 公钥是否匹配此哈希值（格式："<type>:<value>"）。
--discovery-token-unsafe-skip-ca-verification	
	对于基于令牌的发现，允许在未关联 --discovery-token-ca-cert-hash 参数的情况下添加节点。
-h, --help	

	certs 操作的帮助命令
--node-name string	
	指定节点名称。
--tls-bootstrap-token string	
	指定在加入节点时用于临时通过 Kubernetes 控制平面进行身份验证的令牌。
--token string	
	如果未提供这些值，则将它们用于 discovery-token 令牌和 tls-bootstrap 令牌。

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

概要

为新的控制平面组件生成 kubeconfig

```
kubeadm join phase control-plane-prepare kubeconfig [api-server-endpoint] [flags]
```

选项

--certificate-key string	
	使用此密钥可以解密由 init 上传的证书 secret。
--config string	
	kubeadm 配置文件的路径。
--control-plane	
	在此节点上创建一个新的控制平面实例
--discovery-file string	
	对于基于文件的发现，给出用于加载集群信息的文件或者 URL。
--discovery-token string	
	对于基于令牌的发现，该令牌用于验证从 API 服务器获取的集群信息。
--discovery-token-ca-cert-hash stringSlice	
	对于基于令牌的发现，请验证根 CA 公钥是否匹配此哈希值（格式："<type>:<value>"）。
--discovery-token-unsafe-skip-ca-verification	
	对于基于令牌的发现，允许在未关联 --discovery-token-ca-cert-hash 参数的情况下添加节点。
-h, --help	
	kubeconfig 操作的帮助命令
--tls-bootstrap-token string	
	指定在加入节点时用于临时通过 Kubernetes 控制平面进行身份验证的令牌。
--token string	

	如果未提供这些值，则将它们用于 discovery-token 令牌和 tls-bootstrap 令牌。

从父命令中继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

概要

为新的控制平面组件生成清单（manifest）

```
kubeadm join phase control-plane-prepare control-plane [flags]
```

选项

--apiserver-advertise-address string
对于将要托管新的控制平面实例的节点，指定 API 服务器将公布的其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--apiserver-bind-port int32　默认值：6443
针对将要托管新的控制平面实例的节点，设置 API 服务器要绑定的端口。
--config string
kubeadm 配置文件的路径。
--control-plane
在此节点上创建一个新的控制平面实例
--experimental-patches string
包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
-h, --help
control-plane 操作的帮助命令

从父命令中继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm join phase kubelet-start

使用此阶段，你可以配置 kubelet 设置、证书和（重新）启动 kubelet。

- kubelet-start

概要

生成一个包含 KubeletConfiguration 的文件和一个包含特定于节点的 kubelet 配置的环境文件，然后（重新）启动 kubelet。

```
kubeadm join phase kubelet-start [api-server-endpoint] [flags]
```

选项

--config string	
	kubeadm 配置文件的路径。
--cri-socket string	
	提供给 CRI 套接字建立连接的路径。如果为空，则 kubeadm 将尝试自动检测该值；仅当安装了多个 CRI 或具有非标准 CRI 套接字时，才使用此选项。
--discovery-file string	
	For file-based discovery, a file or URL from which to load cluster information. 对于基于文件的发现，给出用于加载集群信息的文件或者 URL。
--discovery-token string	
	对于基于令牌的发现，该令牌用于验证从 API 服务器获取的集群信息。
--discovery-token-ca-cert-hash stringSlice	
	对于基于令牌的发现，验证根 CA 公钥是否匹配此哈希值（格式："<type>:<value>"）。
--discovery-token-unsafe-skip-ca-verification	
	对于基于令牌的发现，允许在未关联 --discovery-token-ca-cert-hash 参数的情况下添加节点。
-h, --help	
	kubelet-start 操作的帮助命令
--node-name string	
	指定节点名称。
--tls-bootstrap-token string	
	指定在加入节点时用于临时通过 Kubernetes 控制平面进行身份验证的令牌。
--token string	
	如果未提供这些值，则将它们用于 discovery-token 令牌和 tls-bootstrap 令牌。

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm join phase control-plane-join

使用此阶段，你可以将节点作为控制平面实例加入。

- control-plane-join

- all
- etcd
- update-status
- mark-control-plane

概要

添加作为控制平面实例的机器

```
kubeadm join phase control-plane-join [flags]
```

示例

```
# 将机器作为控制平面实例加入
kubeadm join phase control-plane-join all
```

选项

-h, --help	
	control-plane-join 操作的帮助命令

从父命令中继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

添加作为控制平面实例的机器

```
kubeadm join phase control-plane-join all [flags]
```

选项

--apiserver-advertise-address string	
	如果该节点托管一个新的控制平面实例，则 API 服务器将公布其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--config string	
	kubeadm 配置文件的路径。
--experimental-control-plane	
	在此节点上创建一个新的控制平面实例
-h, --help	
	all 操作的帮助命令
--node-name string	
	指定节点名称。

从父命令继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

添加新的本地 etcd 成员

```
kubeadm join phase control-plane-join etcd [flags]
```

选项

--apiserver-advertise-address string
如果该节点托管一个新的控制平面实例，则 API 服务器将公布其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--config string
kubeadm 配置文件的路径。
--control-plane
在此节点上创建一个新的控制平面实例
--experimental-patches string
包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是"strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
-h, --help
etcd 操作的帮助命令
--node-name string
指定节点的名称

从父命令中继承的选项

--rootfs string
[实验] 到 '真实' 主机根文件系统的路径。

概要

将新的控制平面节点注册到 kubeadm-config ConfigMap 维护的 ClusterStatus 中

```
kubeadm join phase control-plane-join update-status [flags]
```

选项

--apiserver-advertise-address string

	如果该节点托管一个新的控制平面实例，则 API 服务器将公布其正在侦听的 IP 地址。如果未设置，则使用默认网络接口。
--config string	
	kubeadm 配置文件的路径。
--control-plane	
	在此节点上创建一个新的控制平面实例
-h, --help	
	update-status 操作的帮助命令
--node-name string	
	指定节点名称。

从父命令中继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

概要

将 Node 节点标记为控制平面节点

```
kubeadm join phase control-plane-join mark-control-plane [flags]
```

选项

--config string	
	kubeadm 配置文件的路径。
--control-plane	
	在此节点上创建一个新的控制平面实例
-h, --help	
	mark-control-plane 操作的帮助命令
--node-name string	
	指定节点的名称

从父命令中继承的选项

--rootfs string	
	[实验] 到 '真实' 主机根文件系统的路径。

接下来

- kubeadm init 引导 Kubernetes 控制平面节点
- kubeadm join 将节点添加到集群
- kubeadm reset 恢复通过 kubeadm init 或 kubeadm join 操作对主机所做的任何更改

- kubeadm alpha 尝试实验性功能

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 04, 2020 at 6:22 PM PST: [zh] Fix links in zh localization (4) (abab517ff)

kubeadm reset phase

kubeadm reset phase 使你能够调用 reset 过程的基本原子步骤。 因此，如果希望执行自定义操作，可以让 kubeadm 做一些工作，然后由用户来补足剩余操作。

kubeadm reset phase 与 kubeadm reset 工作流程 一致，后台都使用相同的代码。

kubeadm reset phase

- phase

概要

使用此命令来调用 reset 工作流程的某个阶段

选项

-h, --help	
	phase 操作的帮助命令

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm reset phase preflight

使用此阶段，你可以在要重置的节点上执行启动前检查阶段。

- preflight

概要

kubeadm reset（重置）前运行启动前检查。

```
kubeadm reset phase preflight [flags]
```

选项

-f, --force	
	在不提示确认的情况下重置节点。
-h, --help	
	preflight 操作的帮助命令
--ignore-preflight-errors stringSlice	
	错误将显示为警告的检查列表；例如：'IsPrivilegedUser,Swap'。取值为 'all' 时将忽略检查中的所有错误。

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm reset phase update-cluster-status

使用此阶段，你可以从 ClusterStatus 对象中删除此控制平面节点。

- update-cluster-status

概要

如果该节点是控制平面节点，从 ClusterStatus 对象中删除该节点。

```
kubeadm reset phase update-cluster-status [flags]
```

选项

-h, --help	
	update-cluster-status 操作的帮助命令

从父命令继承的选项

--rootfs string	
	[实验] 指向 '真实' 宿主机根文件系统的路径。

kubeadm reset phase remove-etcd-member

使用此阶段，你可以从 etcd 集群中删除此控制平面节点的 etcd 成员。

- remove-etcd-member

概要

上传关于当前状态的配置，以便 'kubeadm upgrade' 以后可以知道如何配置升级后的集群。

```
kubeadm config upload [flags]
```

选项

-h, --help
upload 操作的帮助信息

从父命令继承的选项

--kubeconfig string　　默认值： "/etc/kubernetes/admin.conf"
用于和集群通信的 KubeConfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 KubeConfig 文件。
--rootfs string
[实验] 到'真实'主机根文件系统的路径。

kubeadm reset phase cleanup-node

使用此阶段，你可以在此节点上执行清理工作。

- cleanup-node

概要

执行 cleanup node（清理节点）操作。

```
kubeadm reset phase cleanup-node [flags]
```

选项

--cert-dir string　　默认值："/etc/kubernetes/pki"
存储证书的目录路径。如果已指定，则需要清空此目录。
--cri-socket string
要连接的 CRI 套接字的路径。如果为空，则 kubeadm 将尝试自动检测此值；仅当安装了多个CRI 或具有非标准 CRI 插槽时，才使用此选项。
-h, --help

	cleanup-node 操作的帮助命令

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

接下来

- kubeadm init 引导 Kubernetes 控制平面节点
- kubeadm join 将节点添加到集群
- kubeadm reset 恢复通过 kubeadm init 或 kubeadm join 操作对主机所做的任何更改
- kubeadm alpha 尝试实验性功能

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 04, 2020 at 6:22 PM PST: [zh] Fix links in zh localization (4) (abab517ff)

kubeadm upgrade phase

在 Kubernetes v1.15.0 版本中，kubeadm 引入了对 kubeadm upgrade node 阶段的初步支持。其他 kubeadm upgrade 子命令如 apply 等阶段将在未来发行版中添加。

kubeadm upgrade node phase

使用此阶段，可以选择执行辅助控制平面或工作节点升级的单独步骤。请注意，kubeadm upgrade apply 命令仍然必须在主控制平面节点上调用。

- phase
- preflight
- control-plane
- kubelet-config

概要

使用此命令调用 node 工作流的某个阶段

选项

-h, --help
phase 操作的帮助命令

从父命令继承的选项

--rootfs string
[实验] 指向 '真实' 宿主机根文件系统的路径。

执行 kubeadm 升级节点的预检。

```
kubeadm upgrade node phase preflight [flags]
```

选项

-h, --help
preflight 操作的帮助命令
--ignore-preflight-errors stringSlice
错误将显示为警告的检查清单。示例：'IsPrivilegedUser,Swap'。值为'all'表示忽略所有检查的错误。

继承于父命令的选项

--rootfs string
[实验] 到'真实'主机根文件系统的路径。

概要

升级部署在此节点上的控制平面实例，如果有的话

```
kubeadm upgrade node phase control-plane [flags]
```

选项

--certificate-renewal
更新在升级期间变更的组件使用的证书。
--dry-run
不改变任何状态，只输出将要执行的动作。
--etcd-upgrade 默认值: true
执行 etcd 的升级。
--experimental-patches string

	包含名为 "target[suffix][+patchtype].extension" 的文件的目录的路径。 例如，"kube-apiserver0+merge.yaml" 或仅仅是 "etcd.json"。 "patchtype" 可以是 "strategic"、"merge" 或 "json" 之一，并且它们与 kubectl 支持的补丁格式匹配。 默认的 "patchtype" 为 "strategic"。 "extension" 必须为 "json" 或 "yaml"。 "suffix" 是一个可选字符串，可用于确定首先按字母顺序应用哪些补丁。
-h, --help	
	control-plane 的帮助信息
--kubeconfig string 默认值: "/etc/kubernetes/admin.conf"	
	用于和集群通信的 KubeConfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 KubeConfig 文件。

从父命令继承的选项

--rootfs string	
	[实验] 到'真实'主机根文件系统的路径。

从群集中 "kubelet-config-1.X" 的 ConfigMap 下载 kubelet 配置，其中 X 是 kubelet 的次要版本。 kubeadm 使用 --kubelet-version 参数来确定所需的 kubelet 版本。

```
kubeadm upgrade node phase kubelet-config [flags]
```

选项

--dry-run	
	不改变任何状态，只输出将要执行的操作
-h, --help	
	配置操作的帮助信息
--kubeconfig string 默认值： "/etc/kubernetes/kubelet.conf"	
	用于和集群通信的 KubeConfig 文件。如果它没有被设置，那么 kubeadm 将会搜索一个已经存在于标准路径的 KubeConfig 文件。
--kubelet-version string	
	升级后的 kubelet 的*期望*版本。

从父命令继承的选项

--rootfs string	
	[实验] 到'真实'主机根文件系统的路径。

接下来

- kubeadm init 引导一个 Kubernetes 控制平面节点
- kubeadm join 将节点加入到集群
- kubeadm reset 还原 kubeadm init 或 kubeadm join 命令对主机所做的任何更改

- kubeadm upgrade 升级 kubeadm 节点
- kubeadm alpha 尝试实验性功能

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 November 27, 2020 at 5:10 PM PST: sync changes in docs/reference/setup-tools/kubeadm/ directory (679b45e2c)

实现细节

FEATURE STATE: Kubernetes v1.10 [stable]

kubeadm init 和 kubeadm join 结合在一起提供了良好的用户体验，因为从头开始创建实践最佳而配置最基本的 Kubernetes 集群。 但是，kubeadm *如何* 做到这一点可能并不明显。

本文档提供了更多幕后的详细信息，旨在分享有关 Kubernetes 集群最佳实践的知识。

核心设计原则

kubeadm init 和 kubeadm join 设置的集群该是：

- **安全的**：它应采用最新的最佳实践，例如：
  - 实施 RBAC 访问控制
  - 使用节点鉴权机制（Node Authorizer）
  - 在控制平面组件之间使用安全通信
  - 在 API 服务器和 kubelet 之间使用安全通信
  - 锁定 kubelet API
  - 锁定对系统组件（例如 kube-proxy 和 CoreDNS）的 API 的访问
  - 锁定启动引导令牌（Bootstrap Token）可以访问的内容
- **易用的**：用户只需要运行几个命令即可：
  - kubeadm init
  - export KUBECONFIG=/etc/kubernetes/admin.conf
  - kubectl apply -f <所选网络.yaml>
  - kubeadm join --token <令牌> <端点>:<端口>
- **可扩展的**：
  - *不* 应偏向任何特定的网络提供商。不涉及配置集群网络
  - 应该可以使用配置文件来自定义各种参数

常量以及众所周知的值和路径

为了降低复杂性并简化基于 kubeadm 的高级工具的开发，对于众所周知的路径和文件名， kubeadm 使用了一组有限的常量值。

Kubernetes 目录 /etc/kubernetes 在应用程序中是一个常量，因为在大多数情况下 它显然是给定的路径，并且是最直观的位置；其他路径常量和文件名有：

- /etc/kubernetes/manifests 作为 kubelet 查找静态 Pod 清单的路径。静态 Pod 清单的名称为：
  - etcd.yaml
  - kube-apiserver.yaml
  - kube-controller-manager.yaml
  - kube-scheduler.yaml
- /etc/kubernetes/ 作为带有控制平面组件身份标识的 kubeconfig 文件的路径。kubeconfig 文件的名称为：
  - kubelet.conf (在 TLS 引导时名称为 bootstrap-kubelet.conf)
  - controller-manager.conf
  - scheduler.conf
  - admin.conf 用于集群管理员和 kubeadm 本身
- 证书和密钥文件的名称：
  - ca.crt, ca.key 用于 Kubernetes 证书颁发机构
  - apiserver.crt, apiserver.key 用于 API 服务器证书
  - apiserver-kubelet-client.crt, apiserver-kubelet-client.key 用于 API 服务器安全地连接到 kubelet 的客户端证书
  - sa.pub, sa.key 用于控制器管理器签署 ServiceAccount 时使用的密钥
  - front-proxy-ca.crt, front-proxy-ca.key 用于前端代理证书颁发机构
  - front-proxy-client.crt, front-proxy-client.key 用于前端代理客户端

kubeadm init 工作流程内部设计

kubeadm init 内部工作流程 包含一系列要执行的原子性工作任务，如 kubeadm init 中所述。

kubeadm init phase 命令允许用户分别调用每个任务，并最终提供可重用且可组合的 API 或工具箱， 其他 Kubernetes 引导工具、任何 IT 自动化工具和高级用户都可以使用它来 创建自定义集群。

预检

Kubeadm 在启动 init 之前执行一组预检，目的是验证先决条件并避免常见的集群启动问题。 用户可以使用 --ignore-preflight-errors 选项跳过特定的预检查或全部检查。

- [警告] 如果要使用的 Kubernetes 版本（由 --kubernetes-version 标志指定）比 kubeadm CLI 版本至少高一个小版本。

- Kubernetes 系统要求：
  - 如果在 linux上运行：
    - [错误] 如果内核早于最低要求的版本
    - [错误] 如果未设置所需的 cgroups 子系统
  - 如果使用 docker：
    - [警告/错误] 如果 Docker 服务不存在、被禁用或未激活。
    - [错误] 如果 Docker 端点不存在或不起作用
    - [警告] 如果 docker 版本不在经过验证的 docker 版本列表中
  - 如果使用其他 cri 引擎：
    - [错误] 如果 crictl 套接字未应答

- [错误] 如果用户不是 root 用户
- [错误] 如果机器主机名不是有效的 DNS 子域
- [警告] 如果通过网络查找无法访问主机名
- [错误] 如果 kubelet 版本低于 kubeadm 支持的最低 kubelet 版本（当前小版本 -1）
- [错误] 如果 kubelet 版本比所需的控制平面板版本至少高一个小（不支持的版本偏斜）
- [警告] 如果 kubelet 服务不存在或已被禁用
- [警告] 如果 firewalld 处于活动状态
- [错误] 如果 API 服务器绑定的端口或 10250/10251/10252 端口已被占用
- [错误] 如果 /etc/kubernetes/manifest 文件夹已经存在并且不为空
- [错误] 如果 /proc/sys/net/bridge/bridge-nf-call-iptables 文件不存在或不包含 1
- [错误] 如果建议地址是 ipv6，并且 /proc/sys/net/bridge/bridge-nf-call-ip6tables 不存在或不包含 1
- [错误] 如果启用了交换分区
- [错误] 如果命令路径中没有 conntrack、ip、iptables、mount、nsenter 命令
- [警告] 如果命令路径中没有 ebtables、ethtool、socat、tc、touch、crictl 命令
- [警告] 如果 API 服务器、控制器管理器、调度程序的其他参数标志包含一些无效选项
- [警告] 如果与 https://API.AdvertiseAddress:API.BindPort 的连接通过代理
- [警告] 如果服务子网的连接通过代理（仅检查第一个地址）
- [警告] 如果 Pod 子网的连接通过代理（仅检查第一个地址）

- 如果提供了外部 etcd：
  - [错误] 如果 etcd 版本低于最低要求版本
  - [错误] 如果指定了 etcd 证书或密钥，但无法找到
- 如果未提供外部 etcd（因此将安装本地 etcd）：
  - [错误] 如果端口 2379 已被占用
  - [错误] 如果 Etcd.DataDir 文件夹已经存在并且不为空
- 如果授权模式为 ABAC：
  - [错误] 如果 abac_policy.json 不存在
- 如果授权方式为 Webhook
  - [错误] 如果 webhook_authz.conf 不存在

请注意：

1. 可以使用 kubeadm init phase preflight 命令单独触发预检。

生成必要的证书

Kubeadm 生成用于不同目的的证书和私钥对：

- Kubernetes 集群的自签名证书颁发机构会保存到 ca.crt 文件和 ca.key 私钥文件中
- 用于 API 服务器的服务证书，使用 ca.crt 作为 CA 生成，并将证书保存到 apiserver.crt 文件中，私钥保存到 apiserver.key 文件中 该证书应包含以下备用名称：
    - Kubernetes 服务的内部 clusterIP（服务 CIDR 的第一个地址。 例如：如果服务的子网是 10.96.0.0/12，则为 10.96.0.1）
    - Kubernetes DNS 名称，例如：如果 --service-dns-domain 标志值是 cluster.local， 则为 kubernetes.default.svc.cluster.local； 加上默认的 DNS 名称 kubernetes.default.svc、kubernetes.default 和 kubernetes，
    - 节点名称
    - --apiserver-advertise-address
    - 用户指定的其他备用名称
- 用于 API 服务器安全连接到 kubelet 的客户端证书，使用 ca.crt 作为 CA 生成，并保存到 apiserver-kubelet-client.crt，私钥保存到 apiserver-kubelet-client.key 文件中。该证书应该在 system:masters 组织中。
- 用于签名 ServiceAccount 令牌的私钥保存到 sa.key 文件中，公钥保存到 sa.pub 文件中
- 用于前端代理的证书颁发机构保存到 front-proxy-ca.crt 文件中，私钥保存到 front-proxy-ca.key 文件中
- 前端代理客户端的客户端证书，使用 front-proxy-ca.crt 作为 CA 生成，并保存到 front-proxy-client.crt 文件中，私钥保存到 front-proxy-client.key 文件中

证书默认情况下存储在 /etc/kubernetes/pki 中，但是该目录可以使用 --cert-dir 标志进行配置。

请注意：

1. 如果证书和私钥对都存在，并且其内容经过评估符合上述规范，将使用现有文件，并且跳过给定证书的生成阶段。 这意味着用户可以将现有的 CA 复制到 /etc/kubernetes/pki/ca.{crt,key}， kubeadm 将使用这些文件对其余证书进行签名。请参阅使用自定义证书。
2. 仅对 CA 来说，如果所有其他证书和 kubeconfig 文件都已就位，则可以只提供 ca.crt 文件， 而不提供 ca.key 文件。 kubeadm 能够识别出这种情况并启用 ExternalCA，这也意味着了控制器管理器中的 csrsigner 控制器将不会启动
3. 如果 kubeadm 在 外部 CA 模式 下运行，所有证书必须由用户提供，因为 kubeadm 无法自行生成它们。
4. 如果在 --dry-run 模式下执行 kubeadm，证书文件将写入一个临时文件夹中
5. 可以使用 kubeadm init phase certs all 命令单独生成证书。

为控制平面组件生成 kubeconfig 文件

Kubeadm 生成具有用于控制平面组件身份标识的 kubeconfig 文件：

- 供 kubelet 在 TLS 引导期间使用的 kubeconfig 文件 —— /etc/kubernetes/bootstrap-kubelet.conf。 在此文件中，有一个引导令牌或内嵌的客户端证书，向集群表明此节点身份。 此客户端证书应：
  - 根据节点鉴权模块的要求，属于 system:nodes 组织
  - 具有通用名称（CN）：system:node:<小写主机名>
- 控制器管理器的 kubeconfig 文件 —— /etc/kubernetes/controller-manager.conf； 在此文件中嵌入了一个具有控制器管理器身份标识的客户端证书。 此客户端证书应具有 CN：system:kube-controller-manager， 该 CN 由 RBAC 核心组件角色 默认定义的。
- 调度器的 kubeconfig 文件 —— /etc/kubernetes/scheduler.conf； 此文件中嵌入了具有调度器身份标识的客户端证书。此客户端证书应具有 CN：system:kube-scheduler， 该 CN 由 RBAC 核心组件角色 默认定义的。

另外，用于 kubeadm 本身和 admin 的 kubeconfig 文件也被生成并保存到 /etc/kubernetes/admin.conf 文件中。 此处的 admin 定义为正在管理集群并希望完全控制集群（**root**）的实际人员。 内嵌的 admin 客户端证书应是 system:masters 组织的成员， 这一组织名由默认的 RBAC 面向用户的角色绑定 定义。它还应包括一个 CN。 kubeadm 使用 kubernetes-admin CN。

请注意：

1. ca.crt 证书内嵌在所有 kubeconfig 文件中。
2. 如果给定的 kubeconfig 文件存在且其内容经过评估符合上述规范，则 kubeadm 将使用现有文件， 并跳过给定 kubeconfig 的生成阶段
3. 如果 kubeadm 以 ExternalCA 模式 运行，则所有必需的 kubeconfig 也必须由用户提供，因为 kubeadm 不能自己生成
4. 如果在 --dry-run 模式下执行 kubeadm，则 kubeconfig 文件将写入一个临时文件夹中
5. 可以使用 kubeadm init phase kubeconfig all 命令分别生成 kubeconfig 文件。

为控制平面组件生成静态 Pod 清单

Kubeadm 将用于控制平面组件的静态 Pod 清单文件写入 /etc/kubernetes/manifests 目录。 Kubelet 启动后会监视这个目录以便创建 Pod。

静态 Pod 清单有一些共同的属性：

- 所有静态 Pod 都部署在 kube-system 名字空间
- 所有静态 Pod 都打上 tier:ontrol-plane 和 component:{组件名称} 标签
- 所有静态 Pod 均使用 system-node-critical 优先级

所有静态 Pod 都设置了 hostNetwork:true，使得控制平面在配置网络之前启动；
- 结果导致：
  - 控制器管理器和调度器用来调用 API 服务器的地址为 127.0.0.1。
  - 如果使用本地 etcd 服务器，则 etcd-servers 地址将设置为 127.0.0.1:2379
- 同时为控制器管理器和调度器启用了领导者选举
- 控制器管理器和调度器将引用 kubeconfig 文件及其各自的唯一标识
- 如将自定义参数传递给控制平面组件 中所述，所有静态 Pod 都会获得用户指定的额外标志
- 所有静态 Pod 都会获得用户指定的额外卷（主机路径）

请注意：

1. 所有镜像默认从 k8s.gcr.io 拉取。 关于自定义镜像仓库，请参阅 使用自定义镜像 。
2. 如果在 --dry-run 模式下执行 kubeadm，则静态 Pod 文件写入一个临时文件夹中。
3. 可以使用 kubeadm init phase control-plane all 命令分别生成主控组件的静态 Pod 清单。

API 服务器

API 服务器的静态 Pod 清单会受到用户提供的以下参数的影响:

- 要绑定的 apiserver-advertise-address 和 apiserver-bind-port； 如果未提供，则这些值默认为机器上默认网络接口的 IP 地址和 6443 端口。
- service-cluster-ip-range 给 service 使用
- 如果指定了外部 etcd 服务器，则应指定 etcd-servers 地址和相关的 TLS 设置（etcd-cafile，etcd-certfile，etcd-keyfile）； 如果未提供外部 etcd 服务器，则将使用本地 etcd（通过主机网络）
- 如果指定了云提供商，则配置相应的 --cloud-provider，如果该路径存在，则配置 --cloud-config （这是实验性的，是 Alpha 版本，将在以后的版本中删除）

无条件设置的其他 API 服务器标志有：

- --insecure-port=0 禁止到 API 服务器不安全的连接
- --enable-bootstrap-token-auth=true 启用 BootstrapTokenAuthenticator 身份验证模块。 更多细节请参见 TLS 引导。
- --allow-privileged 设为 true（诸如 kube-proxy 这些组件有此要求）
- --requestheader-client-ca-file 设为 front-proxy-ca.crt

- --enable-admission-plugins 设为：
  - NamespaceLifecycle 例如，避免删除系统保留的名字空间
  - LimitRanger 和 ResourceQuota 对名字空间实施限制
  - ServiceAccount 实施服务账户自动化

    - PersistentVolumeLabel 将区域（Region）或区（Zone）标签附加到由云提供商定义的 PersistentVolumes （此准入控制器已被弃用并将在以后的版本中删除）。 如果未明确选择使用 gce 或 aws 作为云提供商，则默认情况下，v1.9 以后的版本 kubeadm 都不会部署。
    - DefaultStorageClass 在 PersistentVolumeClaim 对象上强制使用默认存储类型
    - DefaultTolerationSeconds
    - NodeRestriction 限制 kubelet 可以修改的内容（例如，仅此节点上的 pod）

- --kubelet-preferred-address-types 设为 InternalIP,ExternalIP,Hostname; 这使得在节点的主机名无法解析的环境中，kubectl log 和 API 服务器与 kubelet 的其他通信可以工作

- 使用在前面步骤中生成的证书的标志：

    - --client-ca-file 设为 ca.crt
    - --tls-cert-file 设为 apiserver.crt
    - --tls-private-key-file 设为 apiserver.key
    - --kubelet-client-certificate 设为 apiserver-kubelet-client.crt
    - --kubelet-client-key 设为 apiserver-kubelet-client.key
    - --service-account-key-file 设为 sa.pub
    - --requestheader-client-ca-file 设为 front-proxy-ca.crt
    - --proxy-client-cert-file 设为 front-proxy-client.crt
    - --proxy-client-key-file 设为 front-proxy-client.key

- 其他用于保护前端代理（ API 聚合层 ）通信的标志:

    - --requestheader-username-headers=X-Remote-User
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-allowed-names=front-proxy-client

控制器管理器

控制器管理器的静态 Pod 清单受用户提供的以下参数的影响:

- 如果调用 kubeadm 时指定了 --pod-network-cidr 参数，则可以通过以下方式启用 某些 CNI 网络插件所需的子网管理器功能：
    - 设置 --allocate-node-cidrs=true
    - 根据给定 CIDR 设置 --cluster-cidr 和 --node-cidr-mask-size 标志
- 如果指定了云提供商，则指定相应的 --cloud-provider，如果存在这样的配置文件， 则指定 --cloud-config 路径（此为试验性功能，是 Alpha 版本，将在以后的版本中删除）。

其他无条件设置的标志包括：

- --controllers 为 TLS 引导程序启用所有默认控制器以及 BootstrapSigner 和 TokenCleaner 控制器。详细信息请参阅 TLS 引导
- --use-service-account-credentials 设为 true
- 使用先前步骤中生成的证书的标志：

  ---root-ca-file 设为 ca.crt

  - 如果禁用了 External CA 模式，则 --cluster-signing-cert-file 设为 ca.crt，否则设为 ""
  - 如果禁用了 External CA 模式，则 --cluster-signing-key-file 设为 ca.key，否则设为 ""
  - --service-account-private-key-file 设为 sa.key

调度器

调度器的静态 Pod 清单不受用户提供的参数的影响。

为本地 etcd 生成静态 Pod 清单

如果用户指定了外部 etcd，则将跳过此步骤，否则 kubeadm 会生成静态 Pod 清单文件， 以创建在 Pod 中运行的具有以下属性的本地 etcd 实例：

- 在 localhost:2379 上监听并使用 HostNetwork=true
- 将 hostPath 从 dataDir 挂载到主机的文件系统
- 用户指定的任何其他标志

请注意：

1. etcd 镜像默认从 k8s.gcr.io 拉取。有关自定义镜像仓库，请参阅 使用自定义镜像。
2. 如果 kubeadm 以 --dry-run 模式执行，etcd 静态 Pod 清单将写入一个临时文件夹。
3. 可以使用 'kubeadm init phase etcd local' 命令单独为本地 etcd 生成静态 Pod 清单

可选的动态 Kubelet 配置

要使用这个功能，请执行 kubeadm alpha kubelet config enable-dynamic。 它将 kubelet 的初始化配置写入 /var/lib/kubelet/config/init/kubelet 文件。

初始化配置用于在这个特定节点上启动 kubelet，从而为 kubelet 插件文件提供了 一种替代方法。如以下步骤中所述，这种配置将由 kubelet 基本配置所替代。 请参阅通过配置文件设置 Kubelet 参数 了解更多信息。

请注意：

1. 要使动态 kubelet 配置生效，应在 /etc/systemd/system/kubelet.service.d/10-kubeadm.conf 中指定 --dynamic-config-dir=/var/lib/kubelet/config/dynamic 标志。
2. 通过使用配置文件 --config some-file.yaml 将 KubeletConfiguration 对象传递给 kubeadm init 或 kubeadm join 来更改 kubelet 配置。 可以使用 --- 分隔符将 KubeletConfiguration 对象与其他对象（例如 InitConfiguration ） 分开。更多的详细信息，请查看 kubeadm config print-default 命令。

等待控制平面启动

kubeadm 等待（最多 4m0s ），直到 localhost:6443/healthz（kube-apiserver 存活）返回 ok。 但是为了检测死锁条件，如果 localhost:10255/healthz（kubelet 存活）或 localhost:10255/healthz/syncloop（kubelet 就绪）未能在 40s 和 60s 内未返回 ok， 则 kubeadm 会快速失败。

kubeadm 依靠 kubelet 拉取控制平面镜像并将其作为静态 Pod 正确运行。 控制平面启动后，kubeadm 将完成以下段落中描述的任务。

（可选）编写基本 kubelet 配置

FEATURE STATE: Kubernetes v1.9 [alpha]

如果带 --feature-gates=DynamicKubeletConfig 参数调用 kubeadm，则 kubeadm：

1. 将 kubelet 基本配置写入 kube-system 名字空间的 kubelet-base-config-v1.9 ConfigMap 中。
2. 创建 RBAC 规则，以授予对所有引导令牌和所有 kubelet 实例对该 ConfigMap 的读取访问权限 （即 system:bootstrappers:kubeadm:default-node-token 组和 system:nodes 组）
3. 通过将 Node.spec.configSource 指向新创建的 ConfigMap，为初始控制平面节点启用动态 kubelet 配置功能。

将 kubeadm ClusterConfiguration 保存在 ConfigMap 中以供以后参考

kubeadm 将传递给 kubeadm init 的配置保存在 kube-system 名字空间下名为 kubeadm-config 的 ConfigMap 中。

这将确保将来执行的 kubeadm 操作（例如 kubeadm upgrade ）将能够确定实际/当前集群状态， 并根据该数据做出新的决策。

请注意：

1. 在保存 ClusterConfiguration 之前，从配置中删除令牌等敏感信息。
2. 可以使用 kubeadm init phase upload-config 命令单独上传主控节点配置。

将节点标记为控制平面

一旦控制平面可用，kubeadm 将执行以下操作：

- 给节点打上 node-role.kubernetes.io/master="" 标签，标记其为控制平面
- 给节点打上 node-role.kubernetes.io/master:NoSchedule 污点

请注意：

1. 可以使用 kubeadm init phase mark-control-plane 命令单独触发控制平面标记

为即将加入的节点加入 TLS 启动引导

Kubeadm 使用引导令牌认证 将新节点连接到现有集群； 更多的详细信息，请参见 设计提案。

kubeadm init 确保为该过程正确配置了所有内容，这包括以下步骤以及设置 API 服务器和控制器标志，如前几段所述。

请注意：

1. 可以使用 kubeadm init phase bootstrap-token 命令配置节点的 TLS 引导，执行以下段落中描述的所有配置步骤； 或者每个步骤都单独触发。

创建引导令牌

kubeadm init 创建第一个引导令牌，该令牌是自动生成的或由用户提供的 --token 标志的值；如引导令牌规范中记录的那样， 令牌应保存在 kube-system 名字空间下名为 bootstrap-token-<令牌-id> 的 Secret 中。

请注意：

1. 由 kubeadm init 创建的默认令牌将用于在 TLS 引导过程中验证临时用户； 这些用户会成为 system:bootstrappers:kubeadm:default-node-token 组的成员。
2. 令牌的有效期有限，默认为 24 小时（间隔可以通过 -token-ttl 标志进行更改）
3. 可以使用 kubeadm token 命令创建其他令牌，这些令牌还提供其他有用的令牌管理功能

允许加入的节点调用 CSR API

Kubeadm 确保 system:bootstrappers:kubeadm:default-node-token 组中的用户 能够访问证书签名 API。

这是通过在上述组与默认 RBAC 角色 system:node-bootstrapper 之间创建名为 kubeadm:kubelet-bootstrap 的 ClusterRoleBinding 来实现的。

为新的引导令牌设置自动批准

Kubeadm 确保 csrapprover 控制器自动批准引导令牌的 CSR 请求。

这是通过在 system:bootstrappers:kubeadm:default-node-token 用户组和 system:certificates.k8s.io:certificatesigningrequests:nodeclient 默认角色之间 创建名为 kubeadm:node-autoapprove-bootstrap 的 ClusterRoleBinding 来实现的。

还应创建 system:certificates.k8s.io:certificatesigningrequests:nodeclient 角色， 授予对 /apis/certificates.k8s.io/certificatesigningrequests/nodeclient 执行 POST 的权限。

通过自动批准设置节点证书轮换

Kubeadm 确保节点启用了证书轮换，csrapprover 控制器将自动批准节点的 新证书的 CSR 请求。

这是通过在 system:nodes 组和 system:certificates.k8s.io:certificatesigningrequests:selfnodeclient 默认角色之间创建名为 kubeadm:node-autoapprove-certificate-rotation 的 ClusterRoleBinding 来实现的。

创建公共 cluster-info ConfigMap

本步骤在 kube-public 名字空间中创建名为 cluster-info 的 ConfigMap。

另外，它创建一个 Role 和一个 RoleBinding，为未经身份验证的用户授予对 ConfigMap 的访问权限（即 RBAC 组 system:unauthenticated 中的用户）。

请注意：

1. 对 cluster-info ConfigMap 的访问 *不受* 速率限制。 如果你把 API 服务器暴露到外网，这可能是一个问题，也可能不是； 这里最坏的情况是 DoS 攻击，攻击者使用 kube-apiserver 能够处理的所有动态请求 来为 cluster-info ConfigMap 提供服务。

安装插件

Kubeadm 通过 API 服务器安装内部 DNS 服务器和 kube-proxy 插件。

请注意：

1. 此步骤可以调用 'kubeadm init phase addon all' 命令单独执行。

代理

在 kube-system 名字空间中创建一个用于 kube-proxy 的 ServiceAccount； 然后以 DaemonSet 的方式部署 kube-proxy：

- 主控节点凭据（ca.crt 和 token）来自 ServiceAccount
- API 服务器节点的位置（URL）来自 ConfigMap
- kube-proxy 的 ServiceAccount 绑定了 system:node-proxier ClusterRole 中的特权

DNS

- 在 Kubernetes 1.18 版本中，通过 kubeadm 部署 kube-dns 这一操作已经弃用， 将在未来的版本中删除。
- CoreDNS 服务的名称为 kube-dns。这样做是为了防止当用户将集群 DNS 从 kube-dns 切换到 CoreDNS 或者反过来时，出现服务中断。--config 方法在 这里 有描述。
- 在 kube-system 名字空间中创建 CoreDNS/kube-dns 的 ServiceAccount
- kube-dns 的 ServiceAccount 绑定了 system:kube-dns ClusterRole 中的特权

kubeadm join 步骤内部设计

与 kubeadm init 类似，kubeadm join 内部工作流由一系列待执行的原子工作任务组成。

这分为发现（让该节点信任 Kubernetes 的主控节点）和 TLS 引导 （让 Kubernetes 的主控节点信任该节点）。

请参阅使用引导令牌进行身份验证 或相应的设计提案。

预检

kubeadm 在开始执行之前执行一组预检，目的是验证先决条件，避免常见的集群启动问题。

请注意：

1. kubeadm join 预检基本上是 kubeadm init 预检的一个子集
2. 从 1.9 开始，kubeadm 为 CRI 通用的功能提供了更好的支持；在这种情况下，Docker 特定的控制参数将跳过或替换为 crictl 中与之相似的控制参数。
3. 从 1.9 开始，kubeadm 支持加入在 Windows 上运行的节点；在这种情况下，将跳过 Linux 特定的控制参数。
4. 在任何情况下，用户都可以通过 --ignore-preflight-errors 选项跳过 特定的预检（或者进而跳过所有预检）。

发现 cluster-info

主要有两种发现方案。第一种是使用一个共享令牌以及 API 服务器的 IP 地址。 第二种是提供一个文件（它是标准 kubeconfig 文件的子集）。

共享令牌发现

如果带 --discovery-token 参数调用 kubeadm join，则使用了令牌发现功能； 在这种情况下，节点基本上从 kube-public 名字空间中的 cluster-info ConfigMap 中检索集群 CA 证书。

为了防止"中间人"攻击，采取了以下步骤：

- 首先，通过不安全连接检索 CA 证书（这是可能的，因为 kubeadm init 授予 system:unauthenticated 的用户对 cluster-info 访问权限）
- 然后 CA 证书通过以下验证步骤：
  - 基本验证：使用令牌 ID 而不是 JWT 签名
  - 公钥验证：使用提供的 --discovery-token-ca-cert-hash。这个值来自 kubeadm init 的输出， 或者可以使用标准工具计算（哈希值是按 RFC7469 中主体公钥信息（SPKI）对象的字节计算的） --discovery-token-ca-cert-hash 标志可以重复多次，以允许多个公钥。
  - 作为附加验证，通过安全连接检索 CA 证书，然后与初始检索的 CA 进行比较

请注意：

1. 通过 --discovery-token-unsafe-skip-ca-verification 标志可以跳过公钥验证； 这削弱了 kubeadm 安全模型，因为其他人可能冒充 Kubernetes 主控节点。

文件/HTTPS 发现

如果带 --discovery-file 参数调用 kubeadm join，则使用文件发现功能； 该文件可以是本地文件或通过 HTTPS URL 下载；对于 HTTPS，主机安装的 CA 包 用于验证连接。

通过文件发现，集群 CA 证书是文件本身提供；事实上，这个发现文件是一个 kubeconfig 文件， 只设置了 server 和 certificate-authority-data 属性， 如 kubeadm join 参考文档中所述，当与集群建立连接时，kubeadm 尝试访问 cluster-info ConfigMap， 如果可用，就使用它。

TLS 引导

知道集群信息后，将写入文件 bootstrap-kubelet.conf，从而允许 kubelet 执行 TLS 引导（相反，在 v1.7 之前 TLS 引导都是由 kubeadm 管理）。

TLS 引导机制使用共享令牌对 Kubernetes 主控节点进行临时身份验证，以便 为本地创建的密钥对提交证书签名请求（CSR）。

该请求会被自动批准，并且该操作保存 ca.crt 文件和 kubelet.conf 文件，用于 kubelet 加入集群，同时删除 bootstrap-kubelet.conf。

请注意：

- 临时身份验证根据 kubeadm init 过程中保存的令牌进行验证（或者使用 kubeadm token 创建的其他令牌）
- 临时身份验证解析到 system:bootstrappers:kubeadm:default-node-token 组的一个用户成员， 该成员在 kubeadm init 过程中被授予对 CSR API 的访问权
- 根据 kubeadm init 过程的配置，自动 CSR 审批由 csrapprover 控制器管理

（可选）编写 init kubelet 配置

FEATURE STATE: Kubernetes v1.9 [alpha]

如果带 --feature-gates=DynamicKubeletConfig 参数调用 kubeadm，则 kubeadm：

1. 使用引导令牌凭证从 kube-system 名字空间中 ConfigMap kubelet-base-config-v1.9 中读取 kubelet 基本配置， 并将其作为 kubelet init 配置文件 /var/lib/kubelet/config/init/kubelet 写入磁盘。
2. 一旦 kubelet 开始使用节点自己的凭据（/etc/kubernetes/kubelet.conf）， 就更新当前节点配置，指定该节点或 kubelet 配置来自上述 ConfigMap。

请注意：

1. 要使动态 kubelet 配置生效，应在 /etc/systemd/system/kubelet.service.d/10-kubeadm.conf 中指定 --dynamic-config-dir=/var/lib/kubelet/config/dynamic 标志。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 14, 2020 at 4:10 PM PST: [zh] Sync changes to kubeadm reference (e944fb147)

kubectl 命令行界面

---

kubectl 概述

你可以使用 Kubectl 命令行工具管理 Kubernetes 集群。 kubectl 在 $HOME/.kube 目录中查找一个名为 config 的配置文件。 你可以通过设置 KUBECONFIG 环境变量或设置 --kubeconfig 参数来指定其它 kubeconfig 文件。

本文概述了 kubectl 语法和命令操作描述，并提供了常见的示例。 有关每个命令的详细信息，包括所有受支持的参数和子命令， 请参阅 kubectl 参考文档。 有关安装说明，请参见安装 kubectl 。

语法

使用以下语法 kubectl 从终端窗口运行命令：

```
kubectl [command] [TYPE] [NAME] [flags]
```

其中 command、TYPE、NAME 和 flags 分别是：

- command：指定要对一个或多个资源执行的操作，例如 create、get、describe、delete。
- TYPE：指定资源类型。资源类型不区分大小写， 可以指定单数、复数或缩写形式。例如，以下命令输出相同的结果:

  ```
  kubectl get pod pod1
  kubectl get pods pod1
  kubectl get po pod1
  ```

- NAME：指定资源的名称。名称区分大小写。 如果省略名称，则显示所有资源的详细信息 kubectl get pods。

  在对多个资源执行操作时，你可以按类型和名称指定每个资源，或指定一个或多个文件：

  - 要按类型和名称指定资源：
    - 要对所有类型相同的资源进行分组，请执行以下操作：TYPE1 name1 name2 name<#>。

      例子：kubectl get pod example-pod1 example-pod2
    - 分别指定多个资源类型：TYPE1/name1 TYPE1/name2 TYPE2/name3 TYPE<#>/name<#>。

      例子：kubectl get pod/example-pod1 replicationcontroller/example-rc1

    - 用一个或多个文件指定资源：-f file1 -f file2 -f file<#>
        - 使用 YAML 而不是 JSON 因为 YAML 更容易使用，特别是用于配置文件时。 例子：kubectl get -f ./pod.yaml

- flags: 指定可选的参数。例如，可以使用 -s 或 -server 参数指定 Kubernetes API 服务器的地址和端口。

  注意：

  从命令行指定的参数会覆盖默认值和任何相应的环境变量。

如果你需要帮助，只需从终端窗口运行 kubectl help 即可。

操作

下表包含所有 kubectl 操作的简短描述和普通语法：

操作	语法	描述
alpha	kubectl alpha SUBCOMMAND [flags]	列出与 alpha 特性对应的可用命令，这些特性在 Kubernetes 集群中默认情况下是不启用的。
annotate	kubectl annotate (-f FILENAME | TYPE NAME | TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]	添加或更新一个或多个资源的注解。
api-resources	kubectl api-resources [flags]	列出可用的 API 资源。
api-versions	kubectl api-versions [flags]	列出可用的 API 版本。
apply	kubectl apply -f FILENAME [flags]	从文件或 stdin 对资源应用配置更改。
attach	kubectl attach POD -c CONTAINER [-i] [-t] [flags]	附加到正在运行的容器，查看输出流或与容器（stdin）交互。
auth	kubectl auth [flags] [options]	检查授权。
autoscale	kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu-percent=CPU] [flags]	自动伸缩由副本控制器管理的一组 pod。
certificate	kubectl certificate SUBCOMMAND [options]	修改证书资源。
cluster-info	kubectl cluster-info [flags]	显示有关集群中主服务器和服务的端口信息。

操作	语法	描述
completion	kubectl completion SHELL [options]	为指定的 shell（bash 或 zsh）输出 shell 补齐代码。
config	kubectl config SUBCOMMAND [flags]	修改 kubeconfig 文件。有关详细信息，请参阅各个子命令。
convert	kubectl convert -f FILENAME [options]	在不同的 API 版本之间转换配置文件。配置文件可以是 YAML 或 JSON 格式。
cordon	kubectl cordon NODE [options]	将节点标记为不可调度。
cp	kubectl cp <file-spec-src> <file-spec-dest> [options]	在容器之间复制文件和目录。
create	kubectl create -f FILENAME [flags]	从文件或 stdin 创建一个或多个资源。
delete	kubectl delete (-f FILENAME | TYPE [NAME | /NAME | -l label | --all]) [flags]	从文件、标准输入或指定标签选择器、名称、资源选择器或资源中删除资源。
describe	kubectl describe (-f FILENAME | TYPE [NAME_PREFIX | /NAME | -l label]) [flags]	显示一个或多个资源的详细状态。
diff	kubectl diff -f FILENAME [flags]	将 live 配置和文件或标准输入做对比 (**BETA**)
drain	kubectl drain NODE [options]	腾空节点以准备维护。
edit	kubectl edit (-f FILENAME | TYPE NAME | TYPE/NAME) [flags]	使用默认编辑器编辑和更新服务器上一个或多个资源的定义。
exec	kubectl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]]	对 pod 中的容器执行命令。
explain	kubectl explain [--recursive=false] [flags]	获取多种资源的文档。例如 pod, node, service 等。
expose	kubectl expose (-f FILENAME | TYPE NAME | TYPE/NAME) [--port=port] [--protocol=TCP|UDP] [--target-port=number-or-name] [--name=name] [--external-ip=external-ip-of-service] [--type=type] [flags]	将副本控制器、服务或 pod 作为新的 Kubernetes 服务暴露。
get	kubectl get (-f FILENAME | TYPE [NAME | /NAME | -l label]) [--watch] [--sort-by=FIELD] [[-o | --output]=OUTPUT_FORMAT] [flags]	列出一个或多个资源。

操作	语法	描述
kustomize	kubectl kustomize <dir> [flags] [options]	列出从 kustomization.yaml 文件中的指令生成的一组 API 资源。参数必须是包含文件的目录的路径，或者是 git 存储库 URL，其路径后缀相对于存储库根目录指定了相同的路径。
label	kubectl label (-f FILENAME | TYPE NAME | TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]	添加或更新一个或多个资源的标签。
logs	kubectl logs POD [-c CONTAINER] [--follow] [flags]	在 pod 中打印容器的日志。
options	kubectl options	全局命令行选项列表，适用于所有命令。
patch	kubectl patch (-f FILENAME | TYPE NAME | TYPE/NAME) --patch PATCH [flags]	使用策略合并 patch 程序更新资源的一个或多个字段。
plugin	kubectl plugin [flags] [options]	提供用于与插件交互的实用程序。
port-forward	kubectl port-forward POD [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N] [flags]	将一个或多个本地端口转发到一个 pod。
proxy	kubectl proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [flags]	运行 Kubernetes API 服务器的代理。
replace	kubectl replace -f FILENAME	从文件或标准输入中替换资源。
rollout	kubectl rollout SUBCOMMAND [options]	管理资源的部署。有效的资源类型包括：Deployments, DaemonSets 和 StatefulSets。
run	kubectl run NAME --image=image [--env="key=value"] [--port=port] [--dry-run=server | client | none] [--overrides=inline-json] [flags]	在集群上运行指定的镜像。
scale	kubectl scale (-f FILENAME | TYPE NAME | TYPE/NAME) --replicas=COUNT [--resource-version=version] [--current-replicas=count] [flags]	更新指定副本控制器的大小。
set	kubectl set SUBCOMMAND [options]	配置应用程序资源。

操作	语法	描述
taint	kubectl taint NODE NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N [options]	更新一个或多个节点上的污点。
top	kubectl top [flags] [options]	显示资源（CPU/内存/存储）的使用情况。
uncordon	kubectl uncordon NODE [options]	将节点标记为可调度。
version	kubectl version [--client] [flags]	显示运行在客户端和服务器上的 Kubernetes 版本。
wait	kubectl wait ([-f FILENAME] | resource.group/resource.name | resource.group [(-l label | --all)]) [--for=delete|--for condition=available] [options]	实验性：等待一种或多种资源的特定条件。

了解更多有关命令操作的信息，请参阅 kubectl 参考文档。

资源类型

下表列出所有受支持的资源类型及其缩写别名：

(以下输出可以通过 kubectl api-resources 获取，内容以 Kubernetes 1.19.1 版本为准。)

资源名	缩写名	API 分组	按命名空间	资源类型
bindings			true	Binding
componentstatuses	cs		false	ComponentSt
configmaps	cm		true	ConfigMap
endpoints	ep		true	Endpoints
events	ev		true	Event
limitranges	limits		true	LimitRange
namespaces	ns		false	Namespace
nodes	no		false	Node
persistentvolumeclaims	pvc		true	PersistentVolu
persistentvolumes	pv		false	PersistentVolu
pods	po		true	Pod
podtemplates			true	PodTemplate
replicationcontrollers	rc		true	ReplicationCo
resourcequotas	quota		true	ResourceQuot
secrets			true	Secret
serviceaccounts	sa		true	ServiceAccou

资源名	缩写名	API 分组	按命名空间	资源类型
services	svc		true	Service
mutatingwebhookconfigurations		admissionregistration.k8s.io	false	MutatingWebh
validatingwebhookconfigurations		admissionregistration.k8s.io	false	ValidatingWeb
customresourcedefinitions	crd,crds	apiextensions.k8s.io	false	CustomResou
apiservices		apiregistration.k8s.io	false	APIService
controllerrevisions		apps	true	ControllerRev
daemonsets	ds	apps	true	DaemonSet
deployments	deploy	apps	true	Deployment
replicasets	rs	apps	true	ReplicaSet
statefulsets	sts	apps	true	StatefulSet
tokenreviews		authentication.k8s.io	false	TokenReview
localsubjectaccessreviews		authorization.k8s.io	true	LocalSubjectA
selfsubjectaccessreviews		authorization.k8s.io	false	SelfSubjectAc
selfsubjectrulesreviews		authorization.k8s.io	false	SelfSubjectRu
subjectaccessreviews		authorization.k8s.io	false	SubjectAccess
horizontalpodautoscalers	hpa	autoscaling	true	HorizontalPod
cronjobs	cj	batch	true	CronJob
jobs		batch	true	Job
certificatesigningrequests	csr	certificates.k8s.io	false	CertificateSig
leases		coordination.k8s.io	true	Lease
endpointslices		discovery.k8s.io	true	EndpointSlice
events	ev	events.k8s.io	true	Event
ingresses	ing	extensions	true	Ingress
flowschemas		flowcontrol.apiserver.k8s.io	false	FlowSchema
prioritylevelconfigurations		flowcontrol.apiserver.k8s.io	false	PriorityLevelC
ingressclasses		networking.k8s.io	false	IngressClass
ingresses	ing	networking.k8s.io	true	Ingress
networkpolicies	netpol	networking.k8s.io	true	NetworkPolicy
runtimeclasses		node.k8s.io	false	RuntimeClass
poddisruptionbudgets	pdb	policy	true	PodDisruption
podsecuritypolicies	psp	policy	false	PodSecurityPo
clusterrolebindings		rbac.authorization.k8s.io	false	ClusterRoleBi
clusterroles		rbac.authorization.k8s.io	false	ClusterRole
rolebindings		rbac.authorization.k8s.io	true	RoleBinding
roles		rbac.authorization.k8s.io	true	Role
priorityclasses	pc	scheduling.k8s.io	false	PriorityClass
csidrivers		storage.k8s.io	false	CSIDriver

资源名	缩写名	API 分组	按命名空间	资源类型
csinodes		storage.k8s.io	false	CSINode
storageclasses	sc	storage.k8s.io	false	StorageClass
volumeattachments		storage.k8s.io	false	VolumeAttach

输出选项

有关如何格式化或排序某些命令的输出的信息，请使用以下部分。有关哪些命令支持各种输出选项的详细信息，请参阅kubectl 参考文档。

格式化输出

所有 kubectl 命令的默认输出格式都是人类可读的纯文本格式。要以特定格式向终端窗口输出详细信息，可以将 -o 或 --output 参数添加到受支持的 kubectl 命令中。

语法

```
kubectl [command] [TYPE] [NAME] -o=<output_format>
```

根据 kubectl 操作，支持以下输出格式：

Output format	Description
-o custom-columns=<spec>	使用逗号分隔的自定义列列表打印表。
-o custom-columns-file=<filename>	使用 <filename> 文件中的自定义列模板打印表。
-o json	输出 JSON 格式的 API 对象
-o jsonpath=<template>	打印 jsonpath 表达式定义的字段
-o jsonpath-file=<filename>	打印 <filename> 文件中 jsonpath 表达式定义的字段。
-o name	仅打印资源名称而不打印任何其他内容。
-o wide	以纯文本格式输出，包含任何附加信息。对于 pod 包含节点名。
-o yaml	输出 YAML 格式的 API 对象。

示例

在此示例中，以下命令将单个 pod 的详细信息输出为 YAML 格式的对象：

```
kubectl get pod web-pod-13je7 -o yaml
```

请记住：有关每个命令支持哪种输出格式的详细信息，请参阅 kubectl 参考文档。

自定义列

要定义自定义列并仅将所需的详细信息输出到表中，可以使用该 custom-columns 选项。你可以选择内联定义自定义列或使用模板文件：`-o=custom-columns=<spec>` 或 `-o=custom-columns-file=<filename>`。

示例

内联：

```
kubectl get pods <pod-name> -o custom-columns=NAME:.metadata.name,RSRC:.metadata.resourceVersion
```

模板文件：

```
kubectl get pods <pod-name> -o custom-columns-file=template.txt
```

其中，template.txt 文件包含：

```
NAME          RSRC
metadata.name metadata.resourceVersion
```

运行任何一个命令的结果类似于:

```
NAME          RSRC
submit-queue   610995
```

Server-side 列

kubectl 支持从服务器接收关于对象的特定列信息。 这意味着对于任何给定的资源，服务器将返回与该资源相关的列和行，以便客户端打印。 通过让服务器封装打印的细节，这允许在针对同一集群使用的客户端之间提供一致的人类可读输出。

此功能默认启用。要禁用它，请将该 `--server-print=false` 参数添加到 kubectl get 命令中。

例子：

要打印有关 pod 状态的信息，请使用如下命令：

```
kubectl get pods <pod-name> --server-print=false
```

输出类似于：

```
NAME       AGE
pod-name   1m
```

排序列表对象

要将对象排序后输出到终端窗口，可以将 --sort-by 参数添加到支持的 kubectl 命令。通过使用 --sort-by 参数指定任何数字或字符串字段来对对象进行排序。要指定字段，请使用 jsonpath 表达式。

语法

```
kubectl [command] [TYPE] [NAME] --sort-by=<jsonpath_exp>
```

示例

要打印按名称排序的 pod 列表，请运行：

```
kubectl get pods --sort-by=.metadata.name
```

示例：常用操作

使用以下示例集来帮助你熟悉运行常用 kubectl 操作：

kubectl apply - 以文件或标准输入为准应用或更新资源。

```
# 使用 example-service.yaml 中的定义创建服务。
kubectl apply -f example-service.yaml

# 使用 example-controller.yaml 中的定义创建 replication controller。
kubectl apply -f example-controller.yaml

# 使用 <directory> 路径下的任意 .yaml, .yml, 或 .json 文件 创建对象。
kubectl apply -f <directory>
```

kubectl get - 列出一个或多个资源。

```
# 以纯文本输出格式列出所有 pod。
kubectl get pods

# 以纯文本输出格式列出所有 pod，并包含附加信息(如节点名)。
kubectl get pods -o wide

# 以纯文本输出格式列出具有指定名称的副本控制器。提示：你可以使用别名 'rc' 缩短和替换 'replicationcontroller' 资源类型。
kubectl get replicationcontroller <rc-name>

# 以纯文本输出格式列出所有副本控制器和服务。
kubectl get rc,services

# 以纯文本输出格式列出所有守护程序集，包括未初始化的守护程序集。
```

```
kubectl get ds --include-uninitialized

# 列出在节点 server01 上运行的所有 pod
kubectl get pods --field-selector=spec.nodeName=server01
```

kubectl describe - 显示一个或多个资源的详细状态，默认情况下包括未初始化的资源。

```
# 显示名称为 <node-name> 的节点的详细信息。
kubectl describe nodes <node-name>

# 显示名为 <pod-name> 的 pod 的详细信息。
kubectl describe pods/<pod-name>

# 显示由名为 <rc-name> 的副本控制器管理的所有 pod 的详细信息。
# 记住：副本控制器创建的任何 pod 都以复制控制器的名称为前缀。
kubectl describe pods <rc-name>

# 描述所有的 pod，不包括未初始化的 pod
kubectl describe pods
```

> **说明：**
>
> kubectl get 命令通常用于检索同一资源类型的一个或多个资源。 它具有丰富的参数，允许你使用 -o 或 --output 参数自定义输出格式。你可以指定 -w 或 --watch 参数以开始观察特定对象的更新。 kubectl describe 命令更侧重于描述指定资源的许多相关方面。它可以调用对 API 服务器 的多个 API 调用来为用户构建视图。 例如，该 kubectl describe node 命令不仅检索有关节点的信息，还检索在其上运行的 pod 的摘要，为节点生成的事件等。

kubectl delete - 从文件、stdin 或指定标签选择器、名称、资源选择器或资源中删除资源。

```
# 使用 pod.yaml 文件中指定的类型和名称删除 pod。
kubectl delete -f pod.yaml

# 删除所有带有 '<label-key>=<label-value>' 标签的 Pod 和服务。
kubectl delete pods,services -l <label-key>=<label-value>

# 删除所有 pod，包括未初始化的 pod。
kubectl delete pods --all
```

kubectl exec - 对 pod 中的容器执行命令。

```
# 从 pod <pod-name> 中获取运行 'date' 的输出。默认情况下，输出来自第一个容器。
kubectl exec <pod-name> -- date

# 运行输出 'date' 获取在容器的 <container-name> 中 pod <pod-name> 的输出。
```

```
kubectl exec <pod-name> -c <container-name> -- date

# 获取一个交互 TTY 并运行 /bin/bash <pod-name >。默认情况下，输出来自第一个容器。
kubectl exec -ti <pod-name> -- /bin/bash
```

kubectl logs - 打印 Pod 中容器的日志。

```
# 从 pod 返回日志快照。
kubectl logs <pod-name>

# 从 pod <pod-name> 开始流式传输日志。这类似于 'tail -f' Linux 命令。
kubectl logs -f <pod-name>
```

示例：创建和使用插件

使用以下示例来帮助你熟悉编写和使用 kubectl 插件：

```
# 用任何语言创建一个简单的插件，并为生成的可执行文件命名
# 以前缀 "kubectl-" 开始
cat ./kubectl-hello
```

```
#!/bin/sh

# 这个插件打印单词 "hello world"
echo "hello world"
```

这个插件写好了，把它变成可执行的：

```
sudo chmod a+x ./kubectl-hello

# 并将其移动到路径中的某个位置
sudo mv ./kubectl-hello /usr/local/bin
sudo chown root:root /usr/local/bin

# 你现在已经创建并"安装了"一个 kubectl 插件。
# 你可以开始使用这个插件，从 kubectl 调用它，就像它是一个常规命令一样
kubectl hello
```

```
hello world
```

```
# 你可以"卸载"一个插件，只需从你的路径中删除它
sudo rm /usr/local/bin/kubectl-hello
```

为了查看可用的所有 kubectl 插件，你可以使用 kubectl plugin list 子命令：

```
kubectl plugin list
```

输出类似于：

```
The following kubectl-compatible plugins are available:

/usr/local/bin/kubectl-hello
/usr/local/bin/kubectl-foo
/usr/local/bin/kubectl-bar
```

kubectl plugin list指令也可以向你告警哪些插件被运行，或是被其它插件覆盖了,例如:

```
sudo chmod -x /usr/local/bin/kubectl-foo # 删除执行权限
kubectl plugin list
```

```
The following kubectl-compatible plugins are available:

/usr/local/bin/kubectl-hello
/usr/local/bin/kubectl-foo
  - warning: /usr/local/bin/kubectl-foo identified as a plugin, but it is not
executable
/usr/local/bin/kubectl-bar

error: one plugin warning was found
```

你可以将插件视为在现有 kubectl 命令之上构建更复杂功能的一种方法：

```
cat ./kubectl-whoami
```

接下来的几个示例假设你已经将 kubectl-whoami 设置为以下内容:

```
#!/bin/bash

#这个插件利用 `kubectl config` 命令基于当前所选上下文输出当前用户的信息
kubectl config view --template='{{ range .contexts }}{{ if eq .name "'$(kubectl
config current-context)'" }}Current user: {{ printf "%s\n" .context.user }}{{ end }}
{{ end }}'
```

运行以上命令将为你提供一个输出，其中包含 KUBECONFIG 文件中当前上下文的用户:

```
#!/bin/bash
# 使文件成为可执行的
sudo chmod +x ./kubectl-whoami

# 然后移动到你的路径中
sudo mv ./kubectl-whoami /usr/local/bin

kubectl whoami
Current user: plugins-user
```

要了解关于插件的更多信息，请查看示例 cli 插件。

接下来

- 开始使用 kubectl 命令。
- 查看更多示例 cli 插件。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 04, 2020 at 6:22 PM PST: [zh] Fix links in zh localization (4) (abab517ff)

JSONPath 支持

Kubectl 支持 JSONPath 模板。

JSONPath 模板由 {} 包起来的 JSONPath 表达式组成。Kubectl 使用 JSONPath 表达式来过滤 JSON 对象中的特定字段并格式化输出。除了原始的 JSONPath 模板语法，以下函数和语法也是有效的:

1. 使用双引号将 JSONPath 表达式内的文本引起来。
2. 使用 range，end 运算符来迭代列表。
3. 使用负片索引后退列表。负索引不会"环绕"列表，并且只要 `-index + listLength> = 0` 就有效。

   说明：

   - $ 运算符是可选的，因为默认情况下表达式总是从根对象开始。
   - 结果对象将作为其 String() 函数输出。

给定 JSON 输入:

```
{
  "kind": "List",
  "items":[
    {
      "kind":"None",
      "metadata":{"name":"127.0.0.1"},
      "status":{
```

```
      "capacity":{"cpu":"4"},
      "addresses":[{"type": "LegacyHostIP", "address":"127.0.0.1"}]
     }
    },
    {
     "kind":"None",
     "metadata":{"name":"127.0.0.2"},
     "status":{
      "capacity":{"cpu":"8"},
      "addresses":[
       {"type": "LegacyHostIP", "address":"127.0.0.2"},
       {"type": "another", "address":"127.0.0.3"}
      ]
     }
    }
  ],
  "users":[
    {
     "name": "myself",
     "user": {}
    },
    {
     "name": "e2e",
     "user": {"username": "admin", "password": "secret"}
    }
  ]
}
```

函数	描述	示例	结果
text	纯文本	kind is {.kind}	kind is List
@	当前对象	{@}	与输入相同
. or []	子运算符	{.kind}, {['kind']} or {['name\.type']}	List
..	递归下降	{..name}	127.0.0.1 127.0.0.2 myself e2e
*	通配符。获取所有对象	{.items[*].metadata.name}	[127.0.0.1 127.0.0.2]
[start:end :step]	下标运算符	{.users[0].name}	myself

函数	描述	示例	结果
[,]	并集运算符	{.items[*]['metadata.name', 'status.capacity']}	127.0.0.1 127.0.0.2 map[cpu:4] map[cpu:8]
?()	过滤	{.users[?(@.name=="e2e")].user.password}	secret
range, end	迭代列表	{range .items[*]}[{.metadata.name}, {.status.capacity}] {end}	[127.0.0.1, map[cpu:4]] [127.0.0.2, map[cpu:8]]
''	引用解释执行字符串	{range .items[*]}{.metadata.name}{'\t'}{end}	127.0.0.1 127.0.0.2

使用 kubectl 和 JSONPath 表达式的示例:

```
kubectl get pods -o json
kubectl get pods -o=jsonpath='{@}'
kubectl get pods -o=jsonpath='{.items[0]}'
kubectl get pods -o=jsonpath='{.items[0].metadata.name}'
kubectl get pods -o=jsonpath="{.items[*]['metadata.name', 'status.capacity']}"
kubectl get pods -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}
{.status.startTime}{"\n"}{end}'
```

说明：

在 Windows 上，您必须用双引号把任何包含空格的 JSONPath 模板（不是上面 bash 所示的单引号）。 反过来，这意味着您必须在模板中的所有文字周围使用单引号或转义的双引号。 例如:

```
C:\> kubectl get pods -o=jsonpath="{range .items[*]}{.metadata.name}
{'\t'}{.status.startTime}{'\n'}{end}"
C:\> kubectl get pods -o=jsonpath="{range .items[*]}{.metadata.name}
{\"\t\"}{.status.startTime}{\"\n\"}{end}"
```

说明：

不支持 JSONPath 正则表达式。如需使用正则表达式进行匹配操作，您可以使用如 jq 之类的工具。

```
# kubectl 的 JSONpath 输出不支持正则表达式
# 下面的命令不会生效
kubectl get pods -o jsonpath='{.items[?(@.metadata.name=~/
^test$/)].metadata.name}'

# 下面的命令可以获得所需的结果
```

```
kubectl get pods -o json | jq -r '.items[] | select(.metadata.name | test("test-")).spec.containers[].image'
```

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 01, 2020 at 5:10 PM PST: [zh] Sync change from English (Issues 25247) (#25322) (ec4d60aa8)

kubectl

简介

kubectl 管理控制 Kubernetes 集群。

获取更多信息，请访问 kubectl 概述。

```
kubectl [flags]
```

选项

--add-dir-header	
	设置为 true 表示添加文件目录到日志信息头中
--alsologtostderr	
	表示将日志输出到文件的同时输出到 stderr
--as string	
	以指定用户的身份执行操作
--as-group stringArray	
	模拟指定的组来执行操作，可以使用这个标志来指定多个组。
--azure-container-registry-config string	
	包含 Azure 容器仓库配置信息的文件的路径。
--cache-dir string 默认值: "$HOME/.kube/cache"	
	默认缓存目录
--certificate-authority string	
	指向证书机构的 cert 文件路径
--client-certificate string	
	TLS 使用的客户端证书路径
--client-key string	

	TLS 使用的客户端密钥文件路径
--cloud-provider-gce-l7lb-src-cidrs cidrs　默认值: 130.211.0.0/22,35.191.0.0/16	
	在 GCE 防火墙中开放的 CIDR，用来进行 L7 LB 流量代理和健康检查。
--cloud-provider-gce-lb-src-cidrs cidrs　默认值: 130.211.0.0/22,209.85.152.0/22,209.85.204.0/22,35.191.0.0/16	
	在 GCE 防火墙中开放的 CIDR，用来进行 L4 LB 流量代理和健康检查。
--cluster string	
	要使用的 kubeconfig 集群的名称
--context string	
	要使用的 kubeconfig 上下文的名称
--default-not-ready-toleration-seconds int　默认值: 300	
	表示 `notReady` 状态的容忍度秒数：默认情况下，`NoExecute` 被添加到尚未具有此容忍度的每个 Pod 中。
--default-unreachable-toleration-seconds int　默认值: 300	
	表示 `unreachable` 状态的容忍度秒数：默认情况下，`NoExecute` 被添加到尚未具有此容忍度的每个 Pod 中。
-h, --help	
	kubectl 操作的帮助命令
--insecure-skip-tls-verify	
	设置为 true，则表示不会检查服务器证书的有效性。这样会导致您的 HTTPS 连接不安全。
--kubeconfig string	
	CLI 请求使用的 kubeconfig 配置文件的路径。
--log-backtrace-at traceLocation　默认值: 0	
	当日志机制运行到指定文件的指定行（file:N）时，打印调用堆栈信息
--log-dir string	
	如果不为空，则将日志文件写入此目录
--log-file string	
	如果不为空，则将使用此日志文件
--log-file-max-size uint　默认值: 1800	
	定义日志文件的最大尺寸。单位为兆字节。如果值设置为 0，则表示日志文件大小不受限制。
--log-flush-frequency duration　默认值: 5s	
	两次日志刷新操作之间的最长时间（秒）
--logtostderr　默认值: true	
	日志输出到 stderr 而不是文件中
--match-server-version	
	要求客户端版本和服务端版本相匹配
-n, --namespace string	
	如果存在，CLI 请求将使用此命名空间
--one-output	

	如果为 true，则只将日志写入初始严重级别（而不是同时写入所有较低的严重级别）。
--password string	
	API 服务器进行基本身份验证的密码
--profile string　默认值: "none"	
	要记录的性能指标的名称。可取 (none|cpu|heap|goroutine|threadcreate|block|mutex) 其中之一。
--profile-output string　默认值: "profile.pprof"	
	用于转储所记录的性能信息的文件名
--request-timeout string　默认值: "0"	
	放弃单个服务器请求之前的等待时间，非零值需要包含相应时间单位（例如：1s、2m、3h）。零值则表示不做超时要求。
-s, --server string	
	Kubernetes API 服务器的地址和端口
--skip-headers	
	设置为 true 则表示跳过在日志消息中出现 header 前缀信息
--skip-log-headers	
	设置为 true 则表示在打开日志文件时跳过 header 信息
--stderrthreshold severity　默认值: 2	
	等于或高于此阈值的日志将输出到标准错误输出（stderr）
--token string	
	用于对 API 服务器进行身份认证的持有者令牌
--user string	
	指定使用 kubeconfig 配置文件中的用户名
--username string	
	用于 API 服务器的基本身份验证的用户名
-v, --v Level	
	指定输出日志的日志详细级别
--version version[=true]	
	打印 kubectl 版本信息并退出
--vmodule moduleSpec	
	以逗号分隔的 pattern=N 设置列表，用于过滤文件的日志记录

另请参见

- kubectl annotate - 更新资源所关联的注解
- kubectl api-resources - 打印服务器上所支持的 API 资源
- kubectl api-versions - 以"组/版本"的格式输出服务端所支持的 API 版本
- kubectl apply - 基于文件名或标准输入，将新的配置应用到资源上
- kubectl attach - 连接到一个正在运行的容器

- kubectl auth - 检查授权信息

- kubectl autoscale - 对一个资源对象（Deployment、ReplicaSet 或 ReplicationController ）进行扩缩
- kubectl certificate - 修改证书资源
- kubectl cluster-info - 显示集群信息
- kubectl completion - 根据已经给出的 Shell（bash 或 zsh），输出 Shell 补全后的代码
- kubectl config - 修改 kubeconfig 配置文件

- kubectl convert - 在不同的 API 版本之间转换配置文件
- kubectl cordon - 标记节点为不可调度的
- kubectl cp - 将文件和目录拷入/拷出容器
- kubectl create - 通过文件或标准输入来创建资源
- kubectl debug - 创建用于排查工作负载和节点故障的调试会话
- kubectl delete - 通过文件名、标准输入、资源和名字删除资源，或者通过资源和标签选择器来删除资源

- kubectl describe - 显示某个资源或某组资源的详细信息
- kubectl diff - 显示目前版本与将要应用的版本之间的差异
- kubectl drain - 腾空节点，准备维护
- kubectl edit - 修改服务器上的某资源
- kubectl exec - 在容器中执行相关命令
- kubectl explain - 显示资源文档说明
- kubectl expose - 给定副本控制器、服务、Deployment 或 Pod，将其暴露为新的 kubernetes Service

- kubectl get - 显示一个或者多个资源信息
- kubectl kustomize - 从目录或远程 URL 中构建 kustomization
- kubectl label - 更新资源的标签
- kubectl logs - 输出 pod 中某容器的日志
- kubectl options - 打印所有命令都支持的共有参数列表
- kubectl patch - 基于策略性合并修补（Stategic Merge Patch）规则更新某资源中的字段

- kubectl plugin - 运行命令行插件
- kubectl port-forward - 将一个或者多个本地端口转发到 pod
- kubectl proxy - 运行一个 kubernetes API 服务器代理
- kubectl replace - 基于文件名或标准输入替换资源
- kubectl rollout - 管理资源的上线
- kubectl run - 在集群中使用指定镜像启动容器

- kubectl scale - 为一个 Deployment、ReplicaSet 或 ReplicationController 设置一个新的规模尺寸值
- kubectl set - 为对象设置功能特性
- kubectl taint - 在一个或者多个节点上更新污点配置
- kubectl top - 显示资源（CPU /内存/存储）使用率
- kubectl uncordon - 标记节点为可调度的
- kubectl version - 打印客户端和服务器的版本信息
- kubectl wait - 实验性：等待一个或多个资源达到某种状态

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 18, 2020 at 10:04 AM PST: update content/zh/docs/reference/kubectl/kubectl.md (9e876d557)

kubectl 命令

kubectl 命令参考

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 25, 2019 at 8:51 AM PST: ZH-trans: merge release1.16-temporary to master (#18217) (b8b2cc7c7)

kubectl 备忘单

本页列举了常用的 "kubectl" 命令和标志

Kubectl 自动补全

BASH

```
source <(kubectl completion bash) # 在 bash 中设置当前 shell 的自动补全，要先安装 bash-completion 包。
echo "source <(kubectl completion bash)" >> ~/.bashrc # 在您的 bash shell 中永久的添加自动补全
```

您还可以为 kubectl 使用一个速记别名，该别名也可以与 completion 一起使用：

```
alias k=kubectl
complete -F __start_kubectl k
```

ZSH

```
source <(kubectl completion zsh)  # 在 zsh 中设置当前 shell 的自动补全
echo "[[ $commands[kubectl] ]] && source <(kubectl completion zsh)" >>
~/.zshrc # 在您的 zsh shell 中永久的添加自动补全
```

Kubectl 上下文和配置

设置 kubectl 与哪个 Kubernetes 集群进行通信并修改配置信息。 查看使用 kubeconfig 跨集群授权访问 文档获取配置文件详细信息。

```
kubectl config view # 显示合并的 kubeconfig 配置。

# 同时使用多个 kubeconfig 文件并查看合并的配置
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2 kubectl config view

# 获取 e2e 用户的密码
kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'

kubectl config view -o jsonpath='{.users[].name}'    # 显示第一个用户
kubectl config view -o jsonpath='{.users[*].name}'   # 获取用户列表
kubectl config get-contexts                          # 显示上下文列表
kubectl config current-context                       # 展示当前所处的上下文
kubectl config use-context my-cluster-name           # 设置默认的上下文为 my-
cluster-name

# 添加新的用户配置到 kubeconf 中，使用 basic auth 进行身份认证
kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeus
er --password=kubepassword

# 在指定上下文中持久性地保存名字空间，供所有后续 kubectl 命令使用
kubectl config set-context --current --namespace=ggckad-s2

# 使用特定的用户名和名字空间设置上下文
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # 删除用户 foo
```

Kubectl apply

apply 通过定义 Kubernetes 资源的文件来管理应用。 它通过运行 kubectl apply 在集群中创建和更新资源。 这是在生产中管理 Kubernetes 应用的推荐方法。 参见 Kubectl 文档。

创建对象

Kubernetes 配置可以用 YAML 或 JSON 定义。可以使用的文件扩展名有 .yaml、.yml 和 .json。

```
kubectl apply -f ./my-manifest.yaml           # 创建资源
kubectl apply -f ./my1.yaml -f ./my2.yaml     # 使用多个文件创建
kubectl apply -f ./dir                        # 基于目录下的所有清单文件创建资源
kubectl apply -f https://git.io/vPieo         # 从 URL 中创建资源
kubectl create deployment nginx --image=nginx # 启动单实例 nginx

# 创建一个打印 "Hello World" 的 Job
kubectl create job hello --image=busybox -- echo "Hello World"

# 创建一个打印 "Hello World" 间隔1分钟的 CronJob
kubectl create cronjob hello --image=busybox   --schedule="*/1 * * * *" -- echo "
Hello World"

kubectl explain pods                          # 获取 pod 清单的文档说明

# 从标准输入创建多个 YAML 对象
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000"
```

```
EOF

# 创建有多个 key 的 Secret
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
EOF
```

查看和查找资源

```
# get 命令的基本输出
kubectl get services                          # 列出当前命名空间下的所有 services
kubectl get pods --all-namespaces             # 列出所有命名空间下的全部的 Pods
kubectl get pods -o wide                      # 列出当前命名空间下的全部 Pods，并显示更详细的信息
kubectl get deployment my-dep                 # 列出某个特定的 Deployment
kubectl get pods                              # 列出当前命名空间下的全部 Pods
kubectl get pod my-pod -o yaml                # 获取一个 pod 的 YAML

# describe 命令的详细输出
kubectl describe nodes my-node
kubectl describe pods my-pod

# 列出当前名字空间下所有 Services，按名称排序
kubectl get services --sort-by=.metadata.name

# 列出 Pods，按重启次数排序
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

# 列举所有 PV 持久卷，按容量排序
kubectl get pv --sort-by=.spec.capacity.storage

# 获取包含 app=cassandra 标签的所有 Pods 的 version 标签
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# 检索带有 "." 键值，例： 'ca.crt'
kubectl get configmap myconfig \
  -o jsonpath='{.data.ca\.crt}'
```

```
# 获取所有工作节点（使用选择器以排除标签名称为 'node-role.kubernetes.io/master' 的结果）
kubectl get node --selector='!node-role.kubernetes.io/master'

# 获取当前命名空间中正在运行的 Pods
kubectl get pods --field-selector=status.phase=Running

# 获取全部节点的 ExternalIP 地址
kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'

# 列出属于某个特定 RC 的 Pods 的名称
# 在转换对于 jsonpath 过于复杂的场合，"jq" 命令很有用；可以在 https://stedolan.github.io/jq/ 找到它。
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entries | .[] | "\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})

# 显示所有 Pods 的标签（或任何其他支持标签的 Kubernetes 对象）
kubectl get pods --show-labels

# 检查哪些节点处于就绪状态
JSONPATH='{range .items[*]}{@.metadata.name}:{range @.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# 列出被一个 Pod 使用的全部 Secret
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq

# 列举所有 Pods 中初始化容器的容器 ID（containerID）
# Helpful when cleaning up stopped containers, while avoiding removal of initContainers.
kubectl get pods --all-namespaces -o jsonpath='{range .items[*].status.initContainerStatuses[*]}{.containerID}{"\n"}{end}' | cut -d/ -f3

# 列出事件（Events），按时间戳排序
kubectl get events --sort-by=.metadata.creationTimestamp

# 比较当前的集群状态和假定某清单被应用之后的集群状态
kubectl diff -f ./my-manifest.yaml

# 生成一个句点分隔的树，其中包含为节点返回的所有键
# 在复杂的嵌套JSON结构中定位键时非常有用
```

```
kubectl get nodes -o json | jq -c 'path(..)|[.[]|tostring]|join(".")'

# 生成一个句点分隔的树，其中包含为pod等返回的所有键
kubectl get pods -o json | jq -c 'path(..)|[.[]|tostring]|join(".")'
```

更新资源

```
kubectl set image deployment/frontend www=image:v2               # 滚动更新 "frontend" Deployment 的 "www" 容器镜像
kubectl rollout history deployment/frontend                      # 检查 Deployment 的历史记录，包括版本
kubectl rollout undo deployment/frontend                         # 回滚到上次部署版本
kubectl rollout undo deployment/frontend --to-revision=2         # 回滚到特定部署版本
kubectl rollout status -w deployment/frontend                    # 监视 "frontend" Deployment 的滚动升级状态直到完成
kubectl rollout restart deployment/frontend                      # 轮替重启 "frontend" Deployment

cat pod.json | kubectl replace -f -                              # 通过传入到标准输入的 JSON 来替换 Pod

# 强制替换，删除后重建资源。会导致服务不可用。
kubectl replace --force -f ./pod.json

# 为多副本的 nginx 创建服务，使用 80 端口提供服务，连接到容器的 8000 端口。
kubectl expose rc nginx --port=80 --target-port=8000

# 将某单容器 Pod 的镜像版本（标签）更新到 v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | kubectl replace -f -

kubectl label pods my-pod new-label=awesome                      # 添加标签
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq       # 添加注解
kubectl autoscale deployment foo --min=2 --max=10                # 对 "foo" Deployment 自动伸缩容
```

部分更新资源

```
# 部分更新某节点
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

# 更新容器的镜像；spec.containers[*].name 是必须的。因为它是一个合并性质的主键。
```

```
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'

# 使用带位置数组的 JSON patch 更新容器的镜像
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image", "value":"new image"}]'

# 使用带位置数组的 JSON patch 禁用某 Deployment 的 livenessProbe
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove", "path": "/spec/template/spec/containers/0/livenessProbe"}]'

# 在带位置数组中添加元素
kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1", "value": {"name": "whatever" } }]'
```

编辑资源

使用你偏爱的编辑器编辑 API 资源。

```
kubectl edit svc/docker-registry                      # 编辑名为 docker-registry 的服务
KUBE_EDITOR="nano" kubectl edit svc/docker-registry   # 使用其他编辑器
```

对资源进行伸缩

```
kubectl scale --replicas=3 rs/foo                                 # 将名为 'foo' 的副本集伸缩到 3 副本
kubectl scale --replicas=3 -f foo.yaml                            # 将在 "foo.yaml" 中的特定资源伸缩到 3 个副本
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql  # 如果名为 mysql 的 Deployment 的副本当前是 2，那么将它伸缩到 3
kubectl scale --replicas=5 rc/foo rc/bar rc/baz                   # 伸缩多个副本控制器
```

删除资源

```
kubectl delete -f ./pod.json                                      # 删除在 pod.json 中指定的类型和名称的 Pod
kubectl delete pod,service baz foo                                # 删除名称为 "baz" 和 "foo" 的 Pod 和服务
kubectl delete pods,services -l name=myLabel                      # 删除包含 name=myLabel 标签的 pods 和服务
kubectl -n my-ns delete pod,svc --all                             # 删除在 my-ns 名字空间中全部的 Pods 和服务
# 删除所有与 pattern1 或 pattern2 awk 模式匹配的 Pods
kubectl get pods  -n mynamespace --no-headers=true | awk '/pattern1|pattern2/{print $1}' | xargs  kubectl delete -n mynamespace pod
```

与运行中的 Pods 进行交互

```
kubectl logs my-pod                                 # 获取 pod 日志（标准输出）
kubectl logs -l name=myLabel                        # 获取含 name=myLabel 标签的 Pods 的日志（标准输出）
kubectl logs my-pod --previous                      # 获取上个容器实例的 pod 日志（标准输出）
kubectl logs my-pod -c my-container                 # 获取 Pod 容器的日志（标准输出，多容器场景）
kubectl logs -l name=myLabel -c my-container        # 获取含 name=myLabel 标签的 Pod 容器日志（标准输出，多容器场景）
kubectl logs my-pod -c my-container --previous      # 获取 Pod 中某容器的上个实例的日志（标准输出，多容器场景）
kubectl logs -f my-pod                              # 流式输出 Pod 的日志（标准输出）
kubectl logs -f my-pod -c my-container              # 流式输出 Pod 容器的日志（标准输出，多容器场景）
kubectl logs -f -l name=myLabel --all-containers    # 流式输出含 name=myLabel 标签的 Pod 的所有日志（标准输出）
kubectl run -i --tty busybox --image=busybox -- sh  # 以交互式 Shell 运行 Pod
kubectl run nginx --image=nginx -n mynamespace      # 在指定名字空间中运行 nginx Pod
kubectl run nginx --image=nginx                     # 运行 ngins Pod 并将其规约写入到名为 pod.yaml 的文件
  --dry-run=client -o yaml > pod.yaml

kubectl attach my-pod -i                            # 挂接到一个运行的容器中
kubectl port-forward my-pod 5000:6000               # 在本地计算机上侦听端口 5000 并转发到 my-pod 上的端口 6000
kubectl exec my-pod -- ls /                         # 在已有的 Pod 中运行命令（单容器场景）
kubectl exec --stdin --tty my-pod -- /bin/sh        # 使用交互 shell 访问正在运行的 Pod (一个容器场景)
kubectl exec my-pod -c my-container -- ls /         # 在已有的 Pod 中运行命令（多容器场景）
kubectl top pod POD_NAME --containers               # 显示给定 Pod 和其中容器的监控数据
```

与节点和集群进行交互

```
kubectl cordon my-node                                # 标记 my-node 节点为不可调度
kubectl drain my-node                                 # 对 my-node 节点进行清空操作，为节点维护做准备
kubectl uncordon my-node                              # 标记 my-node 节点为可以调度
```

```
kubectl top node my-node                                  # 显示给定节点的度量值
kubectl cluster-info                                      # 显示主控节点和服务的地址
kubectl cluster-info dump                                 # 将当前集群状态转储到标准输出
kubectl cluster-info dump --output-directory=/path/to/cluster-state   # 将当前集群状态输出到 /path/to/cluster-state

# 如果已存在具有指定键和效果的污点，则替换其值为指定值。
kubectl taint nodes foo dedicated=special-user:NoSchedule
```

资源类型

列出所支持的全部资源类型和它们的简称、API 组, 是否是名字空间作用域 和 Kind。

```
kubectl api-resources
```

用于探索 API 资源的其他操作：

```
kubectl api-resources --namespaced=true      # 所有命名空间作用域的资源
kubectl api-resources --namespaced=false     # 所有非命名空间作用域的资源
kubectl api-resources -o name                # 用简单格式列举所有资源（仅显示资源名称）
kubectl api-resources -o wide                # 用扩展格式列举所有资源（又称 "wide" 格式）
kubectl api-resources --verbs=list,get       # 支持 "list" 和 "get" 请求动词的所有资源
kubectl api-resources --api-group=extensions # "extensions" API 组中的所有资源
```

格式化输出

要以特定格式将详细信息输出到终端窗口，将 -o（或者 --output）参数添加到支持的 kubectl 命令中。

输出格式	描述
-o=custom-columns=<spec>	使用逗号分隔的自定义列来打印表格
-o=custom-columns-file=<filename>	使用 <filename> 文件中的自定义列模板打印表格
-o=json	输出 JSON 格式的 API 对象
-o=jsonpath=<template>	打印 jsonpath 表达式中定义的字段
-o=jsonpath-file=<filename>	打印在 <filename> 文件中定义的 jsonpath 表达式所指定的字段。
-o=name	仅打印资源名称而不打印其他内容
-o=wide	以纯文本格式输出额外信息，对于 Pod 来说，输出中包含了节点名称
-o=yaml	输出 YAML 格式的 API 对象

使用 -o=custom-columns 的示例：

```
# 集群中运行着的所有镜像
kubectl get pods -A -o=custom-columns='DATA:spec.containers[*].image'

 # 除 "k8s.gcr.io/coredns:1.6.2" 之外的所有镜像
kubectl get pods -A -o=custom-columns='DATA:spec.containers[?(@.image!
="k8s.gcr.io/coredns:1.6.2")].image'

# 输出 metadata 下面的所有字段，无论 Pod 名字为何
kubectl get pods -A -o=custom-columns='DATA:metadata.*'
```

有关更多示例，请参看 kubectl 参考文档。

Kubectl 日志输出详细程度和调试

Kubectl 日志输出详细程度是通过 -v 或者 --v 来控制的，参数后跟一个数字表示日志的级别。 Kubernetes 通用的日志习惯和相关的日志级别在 这里 有相应的描述。

详细程度	描述
--v=0	用于那些应该 *始终* 对运维人员可见的信息，因为这些信息一般很有用。
--v=1	如果您不想要看到冗余信息，此值是一个合理的默认日志级别。
--v=2	输出有关服务的稳定状态的信息以及重要的日志消息，这些信息可能与系统中的重大变化有关。这是建议大多数系统设置的默认日志级别。
--v=3	包含有关系统状态变化的扩展信息。
--v=4	包含调试级别的冗余信息。
--v=5	跟踪级别的详细程度。
--v=6	显示所请求的资源。
--v=7	显示 HTTP 请求头。
--v=8	显示 HTTP 请求内容。
--v=9	显示 HTTP 请求内容而且不截断内容。

接下来

- 参阅 kubectl 概述，进一步了解JsonPath。
- 参阅 kubectl 选项。
- 参阅 kubectl 使用约定来理解如何在可复用的脚本中使用它。
- 查看社区中其他的 kubectl 备忘单。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 06, 2021 at 1:33 AM PST: [zh] update cheatsheet.md (d8d2c89bf)

kubectl 的用法约定

kubectl 的推荐用法约定。

在可重用脚本中使用 kubectl

对于脚本中的稳定输出：

- 请求一个面向机器的输出格式，例如 -o name、-o json、-o yaml、-o go template 或 -o jsonpath。
- 完全限定版本。例如 jobs.v1.batch/myjob。这将确保 kubectl 不会使用其默认版本，该版本会随着时间的推移而更改。
- 不要依赖上下文、首选项或其他隐式状态。

最佳实践

kubectl run

若希望 kubectl run 满足基础设施即代码的要求：

- 使用特定版本的标签标记镜像，不要将该标签移动到新版本。例如，使用 :v1234、v1.2.3、r03062016-1-4，而不是 :latest（有关详细信息，请参阅配置的最佳实践）。
- 使用基于版本控制的脚本来运行包含大量参数的镜像。
- 对于无法通过 kubectl run 参数来表示的功能特性，使用基于源码控制的配置文件，以记录要使用的功能特性。

你可以使用 --dry-run=client 参数来预览而不真正提交即将下发到集群的对象实例：

> **说明：**
>
> 所有的 kubectl run 生成器已弃用。 查阅 Kubernetes v1.17 文档中的生成器列表以及它们的用法。

生成器

你可以使用 kubectl 命令生成以下资源， kubectl create --dry-run=client -o yaml：

```
clusterrole         创建 ClusterRole。
clusterrolebinding  为特定的 ClusterRole 创建 ClusterRoleBinding。
configmap           使用本地文件、目录或文本值创建 Configmap。
cronjob             使用指定的名称创建 Cronjob。
deployment          使用指定的名称创建 Deployment。
```

```
job                 使用指定的名称创建 Job。
namespace           使用指定的名称创建名称空间。
poddisruptionbudget 使用指定名称创建 Pod 干扰预算。
priorityclass       使用指定的名称创建 Priorityclass。
quota               使用指定的名称创建配额。
role                使用单一规则创建角色。
rolebinding         为特定角色或 ClusterRole 创建 RoleBinding。
secret              使用指定的子命令创建 Secret。
service             使用指定的子命令创建服务。
serviceaccount      使用指定的名称创建服务帐户。
```

kubectl apply

- 您可以使用 kubectl apply 命令创建或更新资源。有关使用 kubectl apply 更新资源的详细信息，请参阅 Kubectl 文档。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 18, 2020 at 10:04 AM PST: sync en. (cd8ba3956)

适用于 Docker 用户的 kubectl

您可以使用 Kubernetes 命令行工具 kubectl 与 API 服务器进行交互。如果您熟悉 Docker 命令行工具，则使用 kubectl 非常简单。但是，docker 命令和 kubectl 命令之间有一些区别。以下显示了 docker 子命令，并描述了等效的 kubectl 命令。

docker run

要运行 nginx 部署并将其暴露，请参见kubectl create deployment docker:

```
docker run -d --restart=always -e DOMAIN=cluster --name nginx-app -p 80:80 nginx
```

```
55c103fa129692154a7652490236fee9be47d70a8dd562281ae7d2f9a339a6db
```

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS               NAMES
```

```
55c103fa1296        nginx               "nginx -g 'daemon of..."   9 seconds ago       Up 9 seconds        0.0.0.0:80->80/tcp   nginx-app
```

kubectl:

```
# 启动运行 nginx 的 Pod
kubectl create deployment --image=nginx nginx-app
```

```
deployment.apps/nginx-app created
```

```
# add env to nginx-app
kubectl set env deployment/nginx-app  DOMAIN=cluster
```

```
deployment.apps/nginx-app env updated
```

> **说明：**
>
> kubectl 命令打印创建或突变资源的类型和名称，然后可以在后续命令中使用。部署后，您可以公开新服务。

```
# 通过服务公开端口
kubectl expose deployment nginx-app --port=80 --name=nginx-http
```

```
service "nginx-http" exposed
```

在 kubectl 命令中，我们创建了一个 Deployment，这将保证有 N 个运行 nginx 的 pod(N 代表 spec 中声明的 replica 数，默认为 1)。我们还创建了一个 service，其选择器与容器标签匹配。查看使用服务访问群集中的应用程序 获取更多信息。

默认情况下镜像会在后台运行，与 docker run -d ... 类似，如果您想在前台运行，使用 kubectl run 在前台运行 Pod:

```
kubectl run [-i] [--tty] --attach <name> --image=<image>
```

与 docker run ... 不同的是，如果指定了 --attach ，我们将连接到 stdin，stdout 和 stderr，而不能控制具体连接到哪个输出流（docker -a ...）。要从容器中退出，可以输入 Ctrl + P，然后按 Ctrl + Q。

因为我们使用 Deployment 启动了容器，如果您终止连接到的进程（例如 ctrl-c），容器将会重启，这跟 docker run -it 不同。 如果想销毁该 Deployment（和它的 pod），您需要运行 kubectl delete deployment <name>。

docker ps

如何列出哪些正在运行？查看 kubectl get。

使用 docker 命令：

```
docker ps -a
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES
14636241935f        ubuntu:16.04        "echo test"              5 seconds ago
Exited (0) 5 seconds ago                        cocky_fermi
55c103fa1296        nginx               "nginx -g 'daemon of..."   About a minute ago
Up About a minute           0.0.0.0:80->80/tcp   nginx-app
```

使用 kubectl 命令：

```
kubectl get po
```

```
NAME                         READY     STATUS      RESTARTS   AGE
nginx-app-8df569cb7-4gd89    1/1       Running     0          3m
ubuntu                       0/1       Completed   0          20s
```

docker attach

如何连接到已经运行在容器中的进程？查看 kubectl attach。

使用 docker 命令：

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS               NAMES
55c103fa1296        nginx               "nginx -g 'daemon of..."   5 minutes ago       Up
5 minutes           0.0.0.0:80->80/tcp   nginx-app
```

```
docker attach 55c103fa1296
...
```

kubectl:

```
kubectl get pods
```

```
NAME              READY     STATUS    RESTARTS   AGE
nginx-app-5jyvm   1/1       Running   0          10m
```

```
kubectl attach -it nginx-app-5jyvm
...
```

要从容器中分离，可以输入 Ctrl + P，然后按 Ctrl + Q。

docker exec

如何在容器中执行命令？查看 kubectl exec。

使用 docker 命令：

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS               NAMES
55c103fa1296        nginx               "nginx -g 'daemon of..."   6 minutes ago       Up
6 minutes       0.0.0.0:80->80/tcp   nginx-app
```

```
docker exec 55c103fa1296 cat /etc/hostname
```

```
55c103fa1296
```

使用 kubectl 命令：

```
kubectl get po
```

```
NAME              READY     STATUS    RESTARTS   AGE
nginx-app-5jyvm   1/1       Running   0          10m
```

```
kubectl exec nginx-app-5jyvm -- cat /etc/hostname
```

```
nginx-app-5jyvm
```

执行交互式命令怎么办？

使用 docker 命令：

```
docker exec -ti 55c103fa1296 /bin/sh
# exit
```

kubectl:

```
kubectl exec -ti nginx-app-5jyvm -- /bin/sh
# exit
```

更多信息请查看获取运行中容器的 Shell 环境。

docker logs

如何查看运行中进程的 stdout/stderr？查看 kubectl logs。

使用 docker 命令：

```
docker logs -f a9e
```

```
192.168.9.1 - - [14/Jul/2015:01:04:02 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/
7.35.0" "-"
192.168.9.1 - - [14/Jul/2015:01:04:03 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/
7.35.0" "-"
```

使用 kubectl 命令：

```
kubectl logs -f nginx-app-zibvs
```

```
10.240.63.110 - - [14/Jul/2015:01:09:01 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.26.0" "-"
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.26.0" "-"
```

现在是时候提一下 pod 和容器之间的细微差别了；默认情况下如果 pod 中的进程退出 pod 也不会终止，相反它将会重启该进程。这类似于 docker run 时的 `--restart=always` 选项， 这是主要差别。在 docker 中，进程的每个调用的输出都是被连接起来的，但是对于 kubernetes，每个调用都是分开的。要查看以前在 kubernetes 中执行的输出，请执行以下操作：

```
kubectl logs --previous nginx-app-zibvs
```

```
10.240.63.110 - - [14/Jul/2015:01:09:01 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.26.0" "-"
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.26.0" "-"
```

查看日志架构获取更多信息。

docker stop and docker rm

如何停止和删除运行中的进程？查看 kubectl delete。

使用 docker 命令：

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                         NAMES
a9ec34d98787        nginx               "nginx -g 'daemon of"   22 hours ago        Up
22 hours        0.0.0.0:80->80/tcp, 443/tcp   nginx-app
```

```
docker stop a9ec34d98787
```

```
a9ec34d98787
```

```
docker rm a9ec34d98787
```

```
a9ec34d98787
```

使用 kubectl 命令：

```
kubectl get deployment nginx-app
```

```
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-app   1         1         1            1           2m
```

```
kubectl get po -l run=nginx-app
```

```
NAME                     READY     STATUS    RESTARTS   AGE
nginx-app-2883164633-aklf7   1/1       Running   0          2m
```

```
kubectl delete deployment nginx-app
```

```
deployment "nginx-app" deleted
```

```
kubectl get po -l run=nginx-app
# Return nothing
```

> **说明：**
>
> 请注意，我们不直接删除 pod。使用 kubectl 命令，我们要删除拥有该 pod 的 Deployment。如果我们直接删除 pod，Deployment 将会重新创建该 pod。

docker login

在 kubectl 中没有对 docker login 的直接模拟。如果您有兴趣在私有镜像仓库中使用 Kubernetes，请参阅使用私有镜像仓库。

docker version

如何查看客户端和服务端的版本？查看 kubectl version。

使用 docker 命令：

```
docker version
```

```
Client version: 1.7.0
Client API version: 1.19
Go version (client): go1.4.2
Git commit (client): 0baf609
OS/Arch (client): linux/amd64
Server version: 1.7.0
Server API version: 1.19
Go version (server): go1.4.2
Git commit (server): 0baf609
OS/Arch (server): linux/amd64
```

使用 kubectl 命令：

```
kubectl version
```

```
Client Version: version.Info{Major:"1", Minor:"6",
GitVersion:"v1.6.9+a3d1dfa6f4335",
GitCommit:"9b77fed11a9843ce3780f70dd251e92901c43072",
```

```
GitTreeState:"dirty", BuildDate:"2017-08-29T20:32:58Z",
OpenPaasKubernetesVersion:"v1.03.02", GoVersion:"go1.7.5", Compiler:"gc",
Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"6",
GitVersion:"v1.6.9+a3d1dfa6f4335",
GitCommit:"9b77fed11a9843ce3780f70dd251e92901c43072",
GitTreeState:"dirty", BuildDate:"2017-08-29T20:32:58Z",
OpenPaasKubernetesVersion:"v1.03.02", GoVersion:"go1.7.5", Compiler:"gc",
Platform:"linux/amd64"}
```

docker info

如何获取有关环境和配置的各种信息？查看 kubectl cluster-info。

使用 docker 命令：

```
docker info
```

```
Containers: 40
Images: 168
Storage Driver: aufs
 Root Dir: /usr/local/google/docker/aufs
 Backing Filesystem: extfs
 Dirs: 248
 Dirperm1 Supported: false
Execution Driver: native-0.2
Logging Driver: json-file
Kernel Version: 3.13.0-53-generic
Operating System: Ubuntu 14.04.2 LTS
CPUs: 12
Total Memory: 31.32 GiB
Name: k8s-is-fun.mtv.corp.google.com
ID: ADUV:GCYR:B3VJ:HMPO:LNPQ:KD5S:YKFQ:76VN:IANZ:7TFV:ZBF4:BYJO
WARNING: No swap limit support
```

使用 kubectl 命令：

```
kubectl cluster-info
```

```
Kubernetes master is running at https://108.59.85.141
KubeDNS is running at https://108.59.85.141/api/v1/namespaces/kube-system/
services/kube-dns/proxy
kubernetes-dashboard is running at https://108.59.85.141/api/v1/namespaces/
kube-system/services/kubernetes-dashboard/proxy
Grafana is running at https://108.59.85.141/api/v1/namespaces/kube-system/
services/monitoring-grafana/proxy
Heapster is running at https://108.59.85.141/api/v1/namespaces/kube-system/
```

```
services/monitoring-heapster/proxy
InfluxDB is running at https://108.59.85.141/api/v1/namespaces/kube-system/
services/monitoring-influxdb/proxy
```

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 October 27, 2020 at 12:17 AM PST: Update links in page docker-cli-to-kubectl.md (dcf5fd415)

命令行工具参考

特性门控

本页详述了管理员可以在不同的 Kubernetes 组件上指定的各种特性门控。

关于特性各个阶段的说明，请参见特性阶段。

概述

特性门控是描述 Kubernetes 特性的一组键值对。您可以在 Kubernetes 的每一个组件中使用 `--feature-gates` flag 来启用或禁用这些特性。

每个 Kubernetes 组件都支持启用或禁用与该组件相关的一组特性门控。 使用 -h 参数来查看所有组件支持的完整特性门控。 要为诸如 kubelet 之类的组件设置特性门控，请使用 --feature-gates 参数，并向其传递一组特性：

```
--feature-gates="...,DynamicKubeletConfig=true"
```

下表总结了在不同的 Kubernetes 组件上可以设置的特性门控。

- 引入特性或更改其发布阶段后，"Since" 列将包含 Kubernetes 版本。
- "Until" 列（如果不为空）包含最后一个 Kubernetes 版本，您仍可以在其中使用特性门控。
- 如果某个特性处于 Alpha 或 Beta 状态，您可以在 Alpha 和 Beta 特性门控表中找到该特性。
- 如果某个特性处于稳定状态，您可以在毕业和废弃特性门控表.中找到该特性的所有阶段。
- 毕业和废弃特性门控表 还列出了废弃的和已被移除的特性。

Alpha 和 Beta 的特性门控

特性	默认值	状态	开始 (Since)	结束 (Until)
AnyVolumeDataSource	false	Alpha	1.18	
APIListChunking	false	Alpha	1.8	1.8
APIListChunking	true	Beta	1.9	
APIPriorityAndFairness	false	Alpha	1.17	
APIResponseCompression	false	Alpha	1.7	
AppArmor	true	Beta	1.4	
BalanceAttachedNodeVolumes	false	Alpha	1.11	
BoundServiceAccountTokenVolume	false	Alpha	1.13	
CPUManager	false	Alpha	1.8	1.9
CPUManager	true	Beta	1.10	
CRIContainerLogRotation	false	Alpha	1.10	1.10
CRIContainerLogRotation	true	Beta	1.11	
CSIInlineVolume	false	Alpha	1.15	1.15
CSIInlineVolume	true	Beta	1.16	-
CSIMigration	false	Alpha	1.14	1.16
CSIMigration	true	Beta	1.17	
CSIMigrationAWS	false	Alpha	1.14	
CSIMigrationAWS	false	Beta	1.17	
CSIMigrationAWSComplete	false	Alpha	1.17	
CSIMigrationAzureDisk	false	Alpha	1.15	1.18
CSIMigrationAzureDisk	false	Beta	1.19	
CSIMigrationAzureDiskComplete	false	Alpha	1.17	
CSIMigrationAzureFile	false	Alpha	1.15	

特性	默认值	状态	开始 (Since)	结束 (Until)
CSIMigrationAzureFileComplete	false	Alpha	1.17	
CSIMigrationGCE	false	Alpha	1.14	1.16
CSIMigrationGCE	false	Beta	1.17	
CSIMigrationGCEComplete	false	Alpha	1.17	
CSIMigrationOpenStack	false	Alpha	1.14	
CSIMigrationOpenStackComplete	false	Alpha	1.17	
CSIMigrationvSphere	false	Beta	1.19	
CSIMigrationvSphereComplete	false	Beta	1.19	
CSIStorageCapacity	false	Alpha	1.19	
CSIVolumeFSGroupPolicy	false	Alpha	1.19	
ConfigurableFSGroupPolicy	false	Alpha	1.18	
CustomCPUCFSQuotaPeriod	false	Alpha	1.12	
DefaultPodTopologySpread	false	Alpha	1.19	
DevicePlugins	false	Alpha	1.8	1.9
DevicePlugins	true	Beta	1.10	
DisableAcceleratorUsageMetrics	false	Alpha	1.19	1.20
DryRun	false	Alpha	1.12	1.12
DryRun	true	Beta	1.13	
DynamicKubeletConfig	false	Alpha	1.4	1.10
DynamicKubeletConfig	true	Beta	1.11	
EndpointSlice	false	Alpha	1.16	1.16
EndpointSlice	false	Beta	1.17	
EndpointSlice	true	Beta	1.18	
EndpointSliceProxying	false	Alpha	1.18	1.18
EndpointSliceProxying	true	Beta	1.19	
EphemeralContainers	false	Alpha	1.16	
ExpandCSIVolumes	false	Alpha	1.14	1.15
ExpandCSIVolumes	true	Beta	1.16	
ExpandInUsePersistentVolumes	false	Alpha	1.11	1.14
ExpandInUsePersistentVolumes	true	Beta	1.15	
ExpandPersistentVolumes	false	Alpha	1.8	1.10
ExpandPersistentVolumes	true	Beta	1.11	
ExperimentalHostUserNamespaceDefaulting	false	Beta	1.5	
GenericEphemeralVolume	false	Alpha	1.19	
HPAScaleToZero	false	Alpha	1.16	
HugePageStorageMediumSize	false	Alpha	1.18	1.18
HugePageStorageMediumSize	true	Beta	1.19	
HyperVContainer	false	Alpha	1.10	

特性	默认值	状态	开始 (Since)	结束 (Until)
ImmutableEphemeralVolumes	false	Alpha	1.18	1.18
ImmutableEphemeralVolumes	true	Beta	1.19	
IPv6DualStack	false	Alpha	1.16	
KubeletPodResources	false	Alpha	1.13	1.14
KubeletPodResources	true	Beta	1.15	
LegacyNodeRoleBehavior	true	Alpha	1.16	
LocalStorageCapacityIsolation	false	Alpha	1.7	1.9
LocalStorageCapacityIsolation	true	Beta	1.10	
LocalStorageCapacityIsolationFSQuotaMonitoring	false	Alpha	1.15	
MountContainers	false	Alpha	1.9	
NodeDisruptionExclusion	false	Alpha	1.16	1.18
NodeDisruptionExclusion	true	Beta	1.19	
NonPreemptingPriority	false	Alpha	1.15	1.18
NonPreemptingPriority	true	Beta	1.19	
PodDisruptionBudget	false	Alpha	1.3	1.4
PodDisruptionBudget	true	Beta	1.5	
PodOverhead	false	Alpha	1.16	-
ProcMountType	false	Alpha	1.12	
QOSReserved	false	Alpha	1.11	
RemainingItemCount	false	Alpha	1.15	
RotateKubeletServerCertificate	false	Alpha	1.7	1.11
RotateKubeletServerCertificate	true	Beta	1.12	
RunAsGroup	true	Beta	1.14	
RuntimeClass	false	Alpha	1.12	1.13
RuntimeClass	true	Beta	1.14	
SCTPSupport	false	Alpha	1.12	1.18
SCTPSupport	true	Beta	1.19	
ServerSideApply	false	Alpha	1.14	1.15
ServerSideApply	true	Beta	1.16	
ServiceAccountIssuerDiscovery	false	Alpha	1.18	
ServiceAppProtocol	false	Alpha	1.18	1.18
ServiceAppProtocol	true	Beta	1.19	
ServiceNodeExclusion	false	Alpha	1.8	1.18
ServiceNodeExclusion	true	Beta	1.19	
ServiceTopology	false	Alpha	1.17	
SetHostnameAsFQDN	false	Alpha	1.19	
StartupProbe	false	Alpha	1.16	1.17
StartupProbe	true	Beta	1.18	

特性	默认值	状态	开始 (Since)	结束 (Until)
StorageVersionHash	false	Alpha	1.14	1.14
StorageVersionHash	true	Beta	1.15	
SupportNodePidsLimit	false	Alpha	1.14	1.14
SupportNodePidsLimit	true	Beta	1.15	
SupportPodPidsLimit	false	Alpha	1.10	1.13
SupportPodPidsLimit	true	Beta	1.14	
Sysctls	true	Beta	1.11	
TokenRequest	false	Alpha	1.10	1.11
TokenRequest	true	Beta	1.12	
TokenRequestProjection	false	Alpha	1.11	1.11
TokenRequestProjection	true	Beta	1.12	
TTLAfterFinished	false	Alpha	1.12	
TopologyManager	false	Alpha	1.16	
ValidateProxyRedirects	false	Alpha	1.12	1.13
ValidateProxyRedirects	true	Beta	1.14	
VolumeSnapshotDataSource	false	Alpha	1.12	1.16
VolumeSnapshotDataSource	true	Beta	1.17	-
WindowsEndpointSliceProxying	false	Alpha	1.19	
WindowsGMSA	false	Alpha	1.14	
WindowsGMSA	true	Beta	1.16	
WinDSR	false	Alpha	1.14	
WinOverlay	false	Alpha	1.14	

已毕业和不推荐使用的特性门控

特性	默认值	状态	开始 (Since)	结束 (Until)
Accelerators	false	Alpha	1.6	1.10
Accelerators	-	Deprecated	1.11	-
AdvancedAuditing	false	Alpha	1.7	1.7
AdvancedAuditing	true	Beta	1.8	1.11
AdvancedAuditing	true	GA	1.12	-
AffinityInAnnotations	false	Alpha	1.6	1.7
AffinityInAnnotations	-	Deprecated	1.8	-
AllowExtTrafficLocalEndpoints	false	Beta	1.4	1.6
AllowExtTrafficLocalEndpoints	true	GA	1.7	-
BlockVolume	false	Alpha	1.9	1.12
BlockVolume	true	Beta	1.13	1.17
BlockVolume	true	GA	1.18	-

特性	默认值	状态	开始 (Since)	结束 (Until)
CSIBlockVolume	false	Alpha	1.11	1.13
CSIBlockVolume	true	Beta	1.14	1.17
CSIBlockVolume	true	GA	1.18	-
CSIDriverRegistry	false	Alpha	1.12	1.13
CSIDriverRegistry	true	Beta	1.14	1.17
CSIDriverRegistry	true	GA	1.18	
CSINodeInfo	false	Alpha	1.12	1.13
CSINodeInfo	true	Beta	1.14	1.16
CSINodeInfo	true	GA	1.17	
AttachVolumeLimit	false	Alpha	1.11	1.11
AttachVolumeLimit	true	Beta	1.12	1.16
AttachVolumeLimit	true	GA	1.17	-
CSIPersistentVolume	false	Alpha	1.9	1.9
CSIPersistentVolume	true	Beta	1.10	1.12
CSIPersistentVolume	true	GA	1.13	-
CustomPodDNS	false	Alpha	1.9	1.9
CustomPodDNS	true	Beta	1.10	1.13
CustomPodDNS	true	GA	1.14	-
CustomResourceDefaulting	false	Alpha	1.15	1.15
CustomResourceDefaulting	true	Beta	1.16	1.16
CustomResourceDefaulting	true	GA	1.17	-
CustomResourcePublishOpenAPI	false	Alpha	1.14	1.14
CustomResourcePublishOpenAPI	true	Beta	1.15	1.15
CustomResourcePublishOpenAPI	true	GA	1.16	-
CustomResourceSubresources	false	Alpha	1.10	1.10
CustomResourceSubresources	true	Beta	1.11	1.15
CustomResourceSubresources	true	GA	1.16	-
CustomResourceValidation	false	Alpha	1.8	1.8
CustomResourceValidation	true	Beta	1.9	1.15
CustomResourceValidation	true	GA	1.16	-
CustomResourceWebhookConversion	false	Alpha	1.13	1.14
CustomResourceWebhookConversion	true	Beta	1.15	1.15
CustomResourceWebhookConversion	true	GA	1.16	-
DynamicAuditing	false	Alpha	1.13	1.18
DynamicAuditing	-	Deprecated	1.19	-
DynamicProvisioningScheduling	false	Alpha	1.11	1.11
DynamicProvisioningScheduling	-	Deprecated	1.12	-
DynamicVolumeProvisioning	true	Alpha	1.3	1.7

特性	默认值	状态	开始 (Since)	结束 (Until)
DynamicVolumeProvisioning	true	GA	1.8	-
EnableEquivalenceClassCache	false	Alpha	1.8	1.14
EnableEquivalenceClassCache	-	Deprecated	1.15	-
ExperimentalCriticalPodAnnotation	false	Alpha	1.5	1.12
ExperimentalCriticalPodAnnotation	false	Deprecated	1.13	-
EvenPodsSpread	false	Alpha	1.16	1.17
EvenPodsSpread	true	Beta	1.18	1.18
EvenPodsSpread	true	GA	1.19	-
GCERegionalPersistentDisk	true	Beta	1.10	1.12
GCERegionalPersistentDisk	true	GA	1.13	-
HugePages	false	Alpha	1.8	1.9
HugePages	true	Beta	1.10	1.13
HugePages	true	GA	1.14	-
Initializers	false	Alpha	1.7	1.13
Initializers	-	Deprecated	1.14	-
KubeletConfigFile	false	Alpha	1.8	1.9
KubeletConfigFile	-	Deprecated	1.10	-
KubeletPluginsWatcher	false	Alpha	1.11	1.11
KubeletPluginsWatcher	true	Beta	1.12	1.12
KubeletPluginsWatcher	true	GA	1.13	-
MountPropagation	false	Alpha	1.8	1.9
MountPropagation	true	Beta	1.10	1.11
MountPropagation	true	GA	1.12	-
NodeLease	false	Alpha	1.12	1.13
NodeLease	true	Beta	1.14	1.16
NodeLease	true	GA	1.17	-
PersistentLocalVolumes	false	Alpha	1.7	1.9
PersistentLocalVolumes	true	Beta	1.10	1.13
PersistentLocalVolumes	true	GA	1.14	-
PodPriority	false	Alpha	1.8	1.10
PodPriority	true	Beta	1.11	1.13
PodPriority	true	GA	1.14	-
PodReadinessGates	false	Alpha	1.11	1.11
PodReadinessGates	true	Beta	1.12	1.13
PodReadinessGates	true	GA	1.14	-
PodShareProcessNamespace	false	Alpha	1.10	1.11
PodShareProcessNamespace	true	Beta	1.12	1.16
PodShareProcessNamespace	true	GA	1.17	-

特性	默认值	状态	开始 (Since)	结束 (Until)
PVCProtection	false	Alpha	1.9	1.9
PVCProtection	-	Deprecated	1.10	-
RequestManagement	false	Alpha	1.15	1.16
ResourceLimitsPriorityFunction	false	Alpha	1.9	1.18
ResourceLimitsPriorityFunction	-	Deprecated	1.19	-
ResourceQuotaScopeSelectors	false	Alpha	1.11	1.11
ResourceQuotaScopeSelectors	true	Beta	1.12	1.16
ResourceQuotaScopeSelectors	true	GA	1.17	-
RotateKubeletClientCertificate	true	Beta	1.8	1.18
RotateKubeletClientCertificate	true	GA	1.19	-
ScheduleDaemonSetPods	false	Alpha	1.11	1.11
ScheduleDaemonSetPods	true	Beta	1.12	1.16
ScheduleDaemonSetPods	true	GA	1.17	-
ServiceLoadBalancerFinalizer	false	Alpha	1.15	1.15
ServiceLoadBalancerFinalizer	true	Beta	1.16	1.16
ServiceLoadBalancerFinalizer	true	GA	1.17	-
StorageObjectInUseProtection	true	Beta	1.10	1.10
StorageObjectInUseProtection	true	GA	1.11	-
StreamingProxyRedirects	false	Beta	1.5	1.5
StreamingProxyRedirects	true	Beta	1.6	1.18
StreamingProxyRedirects	-	Deprecated	1.19	-
SupportIPVSProxyMode	false	Alpha	1.8	1.8
SupportIPVSProxyMode	false	Beta	1.9	1.9
SupportIPVSProxyMode	true	Beta	1.10	1.10
SupportIPVSProxyMode	true	GA	1.11	-
TaintBasedEvictions	false	Alpha	1.6	1.12
TaintBasedEvictions	true	Beta	1.13	1.17
TaintBasedEvictions	true	GA	1.18	-
TaintNodesByCondition	false	Alpha	1.8	1.11
TaintNodesByCondition	true	Beta	1.12	1.16
TaintNodesByCondition	true	GA	1.17	-
VolumePVCDataSource	false	Alpha	1.15	1.15
VolumePVCDataSource	true	Beta	1.16	1.17
VolumePVCDataSource	true	GA	1.18	-
VolumeScheduling	false	Alpha	1.9	1.9
VolumeScheduling	true	Beta	1.10	1.12
VolumeScheduling	true	GA	1.13	-
VolumeSubpath	true	GA	1.13	-

特性	默认值	状态	开始 (Since)	结束 (Until)
VolumeSubpathEnvExpansion	false	Alpha	1.14	1.14
VolumeSubpathEnvExpansion	true	Beta	1.15	1.16
VolumeSubpathEnvExpansion	true	GA	1.17	-
WatchBookmark	false	Alpha	1.15	1.15
WatchBookmark	true	Beta	1.16	1.16
WatchBookmark	true	GA	1.17	-
WindowsGMSA	false	Alpha	1.14	1.15
WindowsGMSA	true	Beta	1.16	1.17
WindowsGMSA	true	GA	1.18	-
WindowsRunAsUserName	false	Alpha	1.16	1.16
WindowsRunAsUserName	true	Beta	1.17	1.17
WindowsRunAsUserName	true	GA	1.18	-

使用特性

特性阶段

处于 *Alpha* 、*Beta* 、 *GA* 阶段的特性。

Alpha 特性代表：

- 默认禁用。
- 可能有错误，启用此特性可能会导致错误。
- 随时可能删除对此特性的支持，恕不另行通知。
- 在以后的软件版本中，API 可能会以不兼容的方式更改，恕不另行通知。
- 建议将其仅用于短期测试中，因为开启特性会增加错误的风险，并且缺乏长期支持。

Beta 特性代表：

- 默认禁用。
- 该特性已经经过良好测试。启用该特性是安全的。
- 尽管详细信息可能会更改，但不会放弃对整体特性的支持。
- 对象的架构或语义可能会在随后的 Beta 或稳定版本中以不兼容的方式更改。当发生这种情况时，我们将提供迁移到下一版本的说明。此特性可能需要删除、编辑和重新创建 API 对象。编辑过程可能需要慎重操作，因为这可能会导致依赖该特性的应用程序停机。
- 推荐仅用于非关键业务用途，因为在后续版本中可能会发生不兼容的更改。如果您具有多个可以独立升级的，则可以放宽此限制。

  说明：

  请试用 *Beta* 特性并提供相关反馈！ 一旦特性结束 Beta 状态，我们就不太可能再对特性进行大幅修改。

General Availability (GA) 特性也称为 *稳定* 特性，*GA* 特性代表着：

- 此特性会一直启用；你不能禁用它。
- 不再需要相应的特性门控。
- 对于许多后续版本，特性的稳定版本将出现在发行的软件中。

特性门控列表

每个特性门控均用于启用或禁用某个特定的特性：

- Accelerators：使用 Docker 时启用 Nvidia GPU 支持。
- AdvancedAuditing：启用高级审计功能。
- AffinityInAnnotations（ *已弃用* ）：启用 Pod 亲和或反亲和。
- AllowExtTrafficLocalEndpoints：启用服务用于将外部请求路由到节点本地终端。
- AnyVolumeDataSource: 允许使用任何自定义的资源来做作为 PVC 中的 DataSource.
- APIListChunking：启用 API 客户端以块的形式从 API 服务器检索（"LIST" 或 "GET"）资源。
- APIPriorityAndFairness: Enable managing request concurrency with prioritization and fairness at each server. (Renamed from RequestManagement)
- APIPriorityAndFairness: 在每个服务器上启用优先级和公平性来管理请求并发。（由 RequestManagement 重命名而来）
- APIResponseCompression：压缩 "LIST" 或 "GET" 请求的 API 响应。
- AppArmor：使用 Docker 时，在 Linux 节点上启用基于 AppArmor 机制的强制访问控制。请参见 AppArmor 教程 获取详细信息。

- AttachVolumeLimit：启用卷插件用于报告可连接到节点的卷数限制。有关更多详细信息，请参见 动态卷限制。
- BalanceAttachedNodeVolumes：包括要在调度时进行平衡资源分配的节点上的卷数。 调度器在决策时会优先考虑 CPU、内存利用率和卷数更近的节点。
- BlockVolume：在 Pod 中启用原始块设备的定义和使用。有关更多详细信息，请参见 原始块卷支持。
- BoundServiceAccountTokenVolume：迁移 ServiceAccount 卷以使用由 ServiceAccountTokenVolumeProjection 组成的投射卷。集群管理员可以使用 serviceaccount_stale_tokens_total 度量值来监控依赖于扩展令牌的负载。如果没有这种类型的负载，你可以在启动 kube-apiserver 时 添加 --service-account-extend-token-expiration=false 参数关闭扩展令牌。查看 绑定服务账号令牌 获取更多详细信息。
- ConfigurableFSGroupPolicy：在 Pod 中挂载卷时，允许用户为 fsGroup 配置卷访问权限和属主变更策略。请参见 为 Pod 配置卷访问权限和属主变更策略。
- CPUManager：启用容器级别的 CPU 亲和性支持，有关更多详细信息，请参见 CPU 管理策略。

- CRIContainerLogRotation：为 cri 容器运行时启用容器日志轮换。
- CSIBlockVolume：启用外部 CSI 卷驱动程序用于支持块存储。有关更多详细信息，请参见 csi 原始块卷支持。

- CSIDriverRegistry：在 csi.storage.k8s.io 中启用与 CSIDriver API 对象有关的所有逻辑。
- CSIInlineVolume：为 Pod 启用 CSI 内联卷支持。
- CSIMigration：确保填充和转换逻辑能够将卷操作从内嵌插件路由到相应的预安装 CSI 插件。
- CSIMigrationAWS：确保填充和转换逻辑能够将卷操作从 AWS-EBS 内嵌插件路由到 EBS CSI 插件。 如果节点未安装和配置 EBS CSI 插件，则支持回退到内嵌 EBS 插件。 这需要启用 CSIMigration 特性标志。
- CSIMigrationAWSComplete：停止在 kubelet 和卷控制器中注册 EBS 内嵌插件， 并启用 shims 和转换逻辑将卷操作从AWS-EBS 内嵌插件路由到 EBS CSI 插件。 这需要启用 CSIMigration 和 CSIMigrationAWS 特性标志，并在集群中的所有节点上安装和配置 EBS CSI 插件。
- CSIMigrationAzureDisk：确保填充和转换逻辑能够将卷操作从 Azure 磁盘内嵌插件路由到 Azure 磁盘 CSI 插件。如果节点未安装和配置 AzureDisk CSI 插件，支持回退到内建 AzureDisk 插件。这需要启用 CSIMigration 特性标志。
- CSIMigrationAzureDiskComplete：停止在 kubelet 和卷控制器中注册 Azure 磁盘内嵌插件， 并启用 shims 和转换逻辑以将卷操作从 Azure 磁盘内嵌插件路由到 AzureDisk CSI 插件。 这需要启用 CSIMigration 和 CSIMigrationAzureDisk 特性标志， 并在集群中的所有节点上安装和配置 AzureDisk CSI 插件。
- CSIMigrationAzureFile：确保填充和转换逻辑能够将卷操作从 Azure 文件内嵌插件路由到 Azure 文件 CSI 插件。如果节点未安装和配置 AzureFile CSI 插件，支持回退到内嵌 AzureFile 插件。这需要启用 CSIMigration 特性标志。
- CSIMigrationAzureFileComplete：停止在 kubelet 和卷控制器中注册 Azure-File 内嵌插件， 并启用 shims 和转换逻辑以将卷操作从 Azure-File 内嵌插件路由到 AzureFile CSI 插件。 这需要启用 CSIMigration 和 CSIMigrationAzureFile 特性标志， 并在集群中的所有节点上安装和配置 AzureFile CSI 插件。

- CSIMigrationGCE：启用 shims 和转换逻辑，将卷操作从 GCE-PD 内嵌插件路由到 PD CSI 插件。如果节点未安装和配置 PD CSI 插件，支持回退到内嵌 GCE 插件。 这需要启用 CSIMigration 特性标志。
- CSIMigrationGCEComplete：停止在 kubelet 和卷控制器中注册 GCE-PD 内嵌插件， 并启用 shims 和转换逻辑以将卷操作从 GCE-PD 内嵌插件路由到 PD CSI 插件。 这需要启用 CSIMigration 和 CSIMigrationGCE 特性标志，并在集群中的所有节点上 安装和配置 PD CSI 插件。
- CSIMigrationOpenStack：确保填充和转换逻辑能够将卷操作从 Cinder 内嵌插件路由到 Cinder CSI 插件。如果节点未安装和配置 Cinder CSI 插件，支持回退到内嵌 Cinder 插件。 这需要启用 CSIMigration 特性标志。
- CSIMigrationOpenStackComplete：停止在 kubelet 和卷控制器中注册 Cinder 内嵌插件， 并启用 shims 和转换逻辑将卷操作从 Cinder 内嵌插件路由到 Cinder CSI 插件。 这需要启用 CSIMigration 和 CSIMigrationOpenStack 特性标志，并在集群中的所有节点上 安装和配置 Cinder CSI 插件。
- CSIMigrationvSphere: 启用 shims 和转换逻辑，将卷操作从 vSphere 内嵌插件路由到 vSphere CSI 插件。 如果节点未安装和配置 vSphere CSI 插件，则支持回退到 vSphere 内嵌插件。 这需要启用 CSIMigration 特性标志。
- CSIMigrationvSphereComplete: 停止在 kubelet 和卷控制器中注册 vSphere 内嵌插件， 并启用 shims 和转换逻辑以将卷操作从 vSphere 内嵌插件路由到

vSphere CSI 插件。 这需要启用 CSIMigration 和 CSIMigrationvSphere 特性标志，并在集群中的所有节点上 安装和配置 vSphere CSI 插件。
- CSINodeInfo：在 csi.storage.k8s.io 中启用与 CSINodeInfo API 对象有关的所有逻辑。
- CSIPersistentVolume：启用发现和挂载通过 CSI（容器存储接口）兼容卷插件配置的卷。
- CSIStorageCapacity: 使 CSI 驱动程序可以发布存储容量信息，并使 Kubernetes 调度程序在调度 Pod 时使用该信息。参见 存储容量。 详情请参见 csi 卷类型。
- CSIVolumeFSGroupPolicy: 允许 CSIDrivers 使用 fsGroupPolicy 字段. 该字段能控制由 CSIDriver 创建的卷在挂载这些卷时是否支持卷所有权和权限修改。

- CustomCPUCFSQuotaPeriod：使节点能够更改 CPUCFSQuotaPeriod。
- CustomPodDNS：使用其 dnsConfig 属性启用 Pod 的自定义 DNS 设置。 更多详细信息，请参见 Pod 的 DNS 配置。
- CustomResourceDefaulting：为 OpenAPI v3 验证架构中的默认值启用 CRD 支持。
- CustomResourcePublishOpenAPI：启用 CRD OpenAPI 规范的发布。
- CustomResourceSubresources：对于用 CustomResourceDefinition 创建的资源启用 /status 和 /scale 子资源。
- CustomResourceValidation：对于用 CustomResourceDefinition 创建的资源启用基于模式的验证。
- CustomResourceWebhookConversion：对于用 CustomResourceDefinition 创建的资源启用基于 Webhook 的转换。 对正在运行的 Pod 进行故障排除。

- DisableAcceleratorUsageMetrics: 禁用 kubelet 收集加速器指标.
- DevicePlugins：在节点上启用基于 设备插件 资源供应。
- DefaultPodTopologySpread: 启用 PodTopologySpread 调度插件来做 默认的调度传播.
- DryRun：启用服务器端 dry run 请求， 以便无需提交即可测试验证、合并和差异化。
- DynamicAuditing（ *已弃用* ）：在 v1.19 版本前用于启用动态审计。
- DynamicKubeletConfig：启用 kubelet 的动态配置。请参阅 重新配置 kubelet 。
- DynamicProvisioningScheduling：扩展默认调度器以了解卷拓扑并处理 PV 配置。 此特性已在 v1.12 中完全被 VolumeScheduling 特性取代。
- DynamicVolumeProvisioning（ *已弃用* ）：启用持久化卷到 Pod 的 动态预配置 。

- EnableAggregatedDiscoveryTimeout（ *已弃用* ）：对聚集的发现调用启用五秒钟超时设置。
- EnableEquivalenceClassCache：调度 Pod 时，使 scheduler 缓存节点的等效项。
- EphemeralContainers：启用添加 临时容器 到正在运行的 Pod 的特性。
- EvenPodsSpread：使 Pod 能够在拓扑域之间平衡调度。请参阅 Pod 拓扑扩展约束。
- ExpandInUsePersistentVolumes：启用扩展使用中的 PVC。请查阅 调整使用中的 PersistentVolumeClaim 的大小。
- ExpandPersistentVolumes：启用持久卷的扩展。请查阅 扩展永久卷声明。

- ExperimentalCriticalPodAnnotation：启用将特定 Pod 注解为 *critical* 的方式，用于 确保其调度。 从 v1.13 开始，Pod 优先级和抢占功能已弃用此特性。

- ExperimentalHostUserNamespaceDefaultingGate：启用主机默认的用户名字空间。 这适用于使用其他主机名字空间、主机安装的容器，或具有特权或使用特定的非名字空间功能（例如 MKNODE、SYS_MODULE 等）的容器。 如果在 Docker 守护程序中启用了用户名字空间重新映射，则启用此选项。
- EndpointSlice：启用 EndpointSlice 以实现更多可扩展的网络端点。 需要启用相应的 API 和控制器，请参阅 启用 EndpointSlice。
- EndpointSliceProxying：启用此特性门控后，Linux 上运行的 kube-proxy 将使用 EndpointSlices 取代 Endpoints 作为主要数据源，可以提高扩展性和性能。请参见 启用 EndpointSlice。
- WindowsEndpointSliceProxying：启用此特性门控后，Windows 上运行的 kube-proxy 将使用 EndpointSlices 取代 Endpoints 作为主要数据源，可以提高扩展性和性能。 请参见 启用 EndpointSlice。
- GCERegionalPersistentDisk：在 GCE 上启用区域 PD 特性。
- GenericEphemeralVolume：启用支持临时卷和内联卷的（可以由第三方存储供应商提供、存储容量跟踪、从快照还原等等）所有功能。请参见 临时卷。
- HugePages：启用分配和使用预分配的 巨页资源。
- HugePageStorageMediumSize：启用支持多种大小的预分配 巨页资源。

- HyperVContainer：为 Windows 容器启用 Hyper-V 隔离。
- HPAScaleToZero：使用自定义指标或外部指标时，可将 HorizontalPodAutoscaler 资源的 minReplicas 设置为 0。
- ImmutableEphemeralVolumes：允许将各个 Secret 和 ConfigMap 标记为不可变更的， 以提高安全性和性能。
- KubeletConfigFile：启用从使用配置文件指定的文件中加载 kubelet 配置。 有关更多详细信息，请参见 通过配置文件设置 kubelet 参数。
- KubeletPluginsWatcher：启用基于探针的插件监视应用程序，使 kubelet 能够发现插件， 例如 CSI 卷驱动程序。
- KubeletPodResources：启用 kubelet 的 pod 资源 grpc 端点。更多详细信息，请参见 支持设备监控。
- LegacyNodeRoleBehavior：禁用此选项后，服务负载均衡中的传统行为和节点中断将忽略 node-role.kubernetes.io/master 标签，而使用 NodeDisruptionExclusion 和 ServiceNodeExclusion 提供的特性指定的标签。

- LocalStorageCapacityIsolation：允许使用 本地临时存储以及 emptyDir 卷 的 sizeLimit 属性。
- LocalStorageCapacityIsolationFSQuotaMonitoring：如果 本地临时存储 启用了 LocalStorageCapacityIsolation，并且 emptyDir 卷 的后备文件系统支持项目配额，并且启用了这些配额，请使用项目配额来监视 emptyDir 卷的存储消耗 而不是遍历文件系统，以此获得更好的性能和准确性。
- MountContainers：在主机上启用将应用程序容器用作卷安装程序。
- MountPropagation：启用将一个容器安装的共享卷共享到其他容器或 Pod。 更多详细信息，请参见 挂载传播。

- NodeDisruptionExclusion：启用节点标签 node.kubernetes.io/exclude-disruption， 以防止在区域故障期间驱逐节点。

- NodeLease：启用新的租赁 API 以报告节点心跳，可用作节点运行状况信号。
- NonPreemptingPriority：为 PriorityClass 和 Pod 启用 NonPreempting 选项。
- PersistentLocalVolumes：在 Pod 中启用 "本地" 卷类型的使用。 如果请求 "本地" 卷，则必须指定 Pod 亲和力。
- PodDisruptionBudget：启用 PodDisruptionBudget 特性。
- PodOverhead：启用 PodOverhead 特性以考虑 Pod 开销。
- PodPriority：根据优先级 启用 Pod 的调度和抢占。
- PodReadinessGates：启用 PodReadinessGate 字段的设置以扩展 Pod 准备状态评估。 有关更多详细信息，请参见 Pod 就绪状态判别。

- PodShareProcessNamespace：在 Pod 中启用 shareProcessNamespace 的设置， 以便在 Pod 中运行的容器之间共享同一进程名字空间。更多详细信息，请参见 在 Pod 中的容器间共享同一进程名字空间。
- ProcMountType：启用对容器的 ProcMountType 的控制。
- PVCProtection：启用防止任何 Pod 仍使用 PersistentVolumeClaim(PVC) 删除的特性。
- QOSReserved：允许在 QoS 级别进行资源预留，以防止处于较低 QoS 级别的 Pod 突发进入处于较高 QoS 级别的请求资源（仅适用于内存）。
- ResourceLimitsPriorityFunction （ *已弃用* ）：启用某调度器优先级函数， 该函数将最低得分 1 指派给至少满足输入 Pod 的 CPU 和内存限制之一的节点， 目的是打破得分相同的节点之间的关联。

- ResourceQuotaScopeSelectors：启用资源配额范围选择器。
- RotateKubeletClientCertificate：在 kubelet 上启用客户端 TLS 证书的轮换。更多详细信息，请参见 kubelet 配置。
- RotateKubeletServerCertificate：在 kubelet 上启用服务器 TLS 证书的轮换。更多详细信息，请参见 kubelet 配置。
- RunAsGroup：启用对容器初始化过程中设置的主要组 ID 的控制。
- RuntimeClass：启用 RuntimeClass 特性用于选择容器运行时配置。
- ScheduleDaemonSetPods：启用 DaemonSet Pods 由默认调度程序而不是 DaemonSet 控制器进行调度。

- SCTPSupport：在 Service、Endpoints、NetworkPolicy 和 Pod 定义中， 允许将 *SCTP* 用作 protocol 值。
- ServerSideApply：在 API 服务器上启用 服务器端应用（SSA） 路径。
- ServiceAccountIssuerDiscovery：在 API 服务器中为服务帐户颁发者启用 OIDC 发现端点。 颁发者和 JWKS URL）。详情请参见 为 Pod 配置服务账户 。
- ServiceAppProtocol：为 Service 和 Endpoints 启用 AppProtocol 字段。
- ServiceLoadBalancerFinalizer：为服务负载均衡启用终结器保护。
- ServiceNodeExclusion：启用从云提供商创建的负载均衡中排除节点。 如果节点标记有 alpha.service-controller.kubernetes.io/exclude-balancer 键或 node.kubernetes.io/exclude-from-external-load-balancers， 则可以排除节点。
- ServiceTopology：启用服务拓扑可以让一个服务基于集群的节点拓扑进行流量路由。 有关更多详细信息，请参见 服务拓扑。

- SetHostnameAsFQDN：启用将全限定域名（FQDN）设置为 Pod 主机名的功能。 请参见给 Pod 设置 setHostnameAsFQDN 字段。
- StartupProbe：在 kubelet 中启用 启动探针。
- StorageObjectInUseProtection：如果仍在使用 PersistentVolume 或 PersistentVolumeClaim 对象，则将其推迟。

- StorageVersionHash：允许 apiserver 在发现中公开存储版本的哈希值。
- StreamingProxyRedirects：指示 API 服务器拦截（并遵循）从后端（kubelet）进行重定向以处理流请求。 流请求的例子包括 exec、attach 和 port-forward 请求。
- SupportIPVSProxyMode：启用使用 IPVS 提供内服务负载平衡。更多详细信息，请参见 服务代理。
- SupportPodPidsLimit：启用支持限制 Pod 中的进程 PID。
- Sysctls：启用对可以为每个 Pod 设置的名字空间内核参数（sysctls）的支持。 更多详细信息，请参见 sysctls。

- TaintBasedEvictions：根据节点上的污点和 Pod 上的容忍度启用从节点驱逐 Pod 的特性。 有关更多详细信息，请参见污点和容忍度。
- TaintNodesByCondition：根据节点状况 启用自动在节点标记污点。
- TokenRequest：在服务帐户资源上启用 TokenRequest 端点。
- TokenRequestProjection：启用通过 projected 卷 将服务帐号令牌注入到 Pod 中的特性。
- TopologyManager：启用一种机制来协调 Kubernetes 不同组件的细粒度硬件资源分配。 详见控制节点上的拓扑管理策略。
- TTLAfterFinished：完成执行后，允许 TTL 控制器清理资源。

- VolumePVCDataSource：启用对将现有 PVC 指定数据源的支持。
- VolumeScheduling：启用卷拓扑感知调度，并使 PersistentVolumeClaim（PVC） 绑定调度决策；当与 PersistentLocalVolumes 特性门控一起使用时， 还可以使用 PersistentLocalVolumes 卷类型。
- VolumeSnapshotDataSource：启用卷快照数据源支持。

- VolumeSubpathEnvExpansion：启用 subPathExpr 字段用于将环境变量扩展为 subPath。
- WatchBookmark：启用对监测 bookmark 事件的支持。
- WindowsGMSA：允许将 GMSA 凭据规范从 Pod 传递到容器运行时。
- WindowsRunAsUserName：提供使用非默认用户在 Windows 容器中运行应用程序的支持。 详情请参见配置 RunAsUserName。
- WinDSR：允许 kube-proxy 为 Windows 创建 DSR 负载均衡。
- WinOverlay：允许 kube-proxy 在 Windows 的 overlay 模式下运行。

接下来

- Kubernetes 的弃用策略 介绍了项目已移除的特性部件和组件的方法。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 16, 2020 at 9:34 AM PST: [zh] update content to match en master (acaa24115)

kube-scheduler

简介

Kubernetes 调度器是一个策略丰富、拓扑感知、工作负载特定的功能，调度器显著影响可用性、性能和容量。调度器需要考虑个人和集体的资源要求、服务质量要求、硬件/软件/政策约束、亲和力和反亲和力规范、数据局部性、负载间干扰、完成期限等。工作负载特定的要求必要时将通过 API 暴露。

```
kube-scheduler [flags]
```

选项

--add-dir-header
如果为 true，则将文件目录添加到标题中
--address string 默认: "0.0.0.0"
弃用: 要监听 --port 端口的 IP 地址（对于所有 IPv4 接口设置为 0.0.0.0，对于所有 IPv6 接口设置为 ::）。 请参阅 --bind-address。
--algorithm-provider string
弃用: 要使用的调度算法，可选值：ClusterAutoscalerProvider | DefaultProvider
--alsologtostderr
--authentication-kubeconfig string
--authentication-skip-lookup
如果为 false，则 authentication-kubeconfig 将用于从集群中查找缺少的身份验证配置。
--authentication-token-webhook-cache-ttl duration 默认: 10s
缓存来自 Webhook 令牌身份验证器的响应的持续时间。
--authentication-tolerate-lookup-failure 默认: true
如果为 true，则无法从集群中查找缺少的身份验证配置是致命的。请注意，这可能导致身份验证将所有请求视为匿名。
--authorization-always-allow-paths stringSlice 默认: [/healthz]

	在授权过程中跳过的 HTTP 路径列表，即在不联系 'core' kubernetes 服务器的情况下被授权的 HTTP 路径。
--authorization-kubeconfig string	
	指向具有足够权限以创建 subjectaccessreviews.authorization.k8s.io 的 'core' kubernetes 服务器的 kubeconfig 文件。这是可选的。如果为空，则禁止所有未经授权跳过的请求。
--authorization-webhook-cache-authorized-ttl duration 默认: 10s	
	缓存来自 Webhook 授权者的 'authorized' 响应的持续时间。
--authorization-webhook-cache-unauthorized-ttl duration 默认: 10s	
	缓存来自 Webhook 授权者的 'unauthorized' 响应的持续时间。
--azure-container-registry-config string	
	包含 Azure 容器仓库配置信息的文件的路径。
--bind-address ip 默认: 0.0.0.0	
	侦听 --secure-port 端口的 IP 地址。集群的其余部分以及 CLI/ Web 客户端必须可以访问关联的接口。如果为空，将使用所有接口（所有 IPv4 接口使用 0.0.0.0，所有 IPv6 接口使用 ::）。
--cert-dir string	
	TLS 证书所在的目录。如果提供了--tls-cert-file 和 --tls private-key-file，则将忽略此参数。
--client-ca-file string	
	如果已设置，由 client-ca-file 中的授权机构签名的客户端证书的任何请求都将使用与客户端证书的 CommonName 对应的身份进行身份验证。
--config string	
	配置文件的路径。标志会覆盖此文件中的值。
--contention-profiling	
	弃用: 如果启用了性能分析，则启用锁竞争分析
--feature-gates mapStringBool	

一组 key=value 对，描述了 alpha/experimental 特征开关。选项包括：
APIListChunking=true|false (BETA - 默认值=true)
APIResponseCompression=true|false (BETA - 默认值=true)
AllAlpha=true|false (ALPHA - 默认值=false)
AppArmor=true|false (BETA - 默认值=true)
AttachVolumeLimit=true|false (BETA - 默认值=true)
BalanceAttachedNodeVolumes=true|false (ALPHA - 默认值=false)
BlockVolume=true|false (BETA - 默认值=true)
BoundServiceAccountTokenVolume=true|false (ALPHA - 默认值=false)
CPUManager=true|false (BETA - 默认值=true)
CRIContainerLogRotation=true|false (BETA - 默认值=true)
CSIBlockVolume=true|false (BETA - 默认值=true)
CSIDriverRegistry=true|false (BETA - 默认值=true)
CSIInlineVolume=true|false (BETA - 默认值=true)
CSIMigration=true|false (ALPHA - 默认值=false)
CSIMigrationAWS=true|false (ALPHA - 默认值=false)
CSIMigrationAzureDisk=true|false (ALPHA - 默认值=false)
CSIMigrationAzureFile=true|false (ALPHA - 默认值=false)
CSIMigrationGCE=true|false (ALPHA - 默认值=false)
CSIMigrationOpenStack=true|false (ALPHA - 默认值=false)
CSINodeInfo=true|false (BETA - 默认值=true)
CustomCPUCFSQuotaPeriod=true|false (ALPHA - 默认值=false)
CustomResourceDefaulting=true|false (BETA - 默认值=true)
DevicePlugins=true|false (BETA - 默认值=true)
DryRun=true|false (BETA - 默认值=true)
DynamicAuditing=true|false (ALPHA - 默认值=false)
DynamicKubeletConfig=true|false (BETA - 默认值=true)
EndpointSlice=true|false (ALPHA - 默认值=false)
EphemeralContainers=true|false (ALPHA - 默认值=false)
EvenPodsSpread=true|false (ALPHA - 默认值=false)
ExpandCSIVolumes=true|false (BETA - 默认值=true)
ExpandInUsePersistentVolumes=true|false (BETA - 默认值=true)
ExpandPersistentVolumes=true|false (BETA - 默认值=true)
ExperimentalHostUserNamespaceDefaulting=true|false (BETA - 默认值=false)
HPAScaleToZero=true|false (ALPHA - 默认值=false)
HyperVContainer=true|false (ALPHA - 默认值=false)
IPv6DualStack=true|false (ALPHA - 默认值=false)
KubeletPodResources=true|false (BETA - 默认值=true)
LegacyNodeRoleBehavior=true|false (ALPHA - 默认值=true)
LocalStorageCapacityIsolation=true|false (BETA - 默认值=true)
LocalStorageCapacityIsolationFSQuotaMonitoring=true|false (ALPHA - 默认值=false)
MountContainers=true|false (ALPHA - 默认值=false)
NodeDisruptionExclusion=true|false (ALPHA - 默认值=false)
NodeLease=true|false (BETA - 默认值=true)
NonPreemptingPriority=true|false (ALPHA - 默认值=false)
PodOverhead=true|false (ALPHA - 默认值=false)

--hard-pod-affinity-symmetric-weight int32 默认: 1
弃用: RequiredDuringScheduling 亲和力不是对称的，但是存在与每个 RequiredDuringScheduling 关联性规则相对应的隐式 PreferredDuringScheduling 关联性规则 --hard-pod-affinity-symmetric-weight 代表隐式 PreferredDuringScheduling 关联性规则的权重。权重必须在 0-100 范围内。此选项已移至策略配置文件。
-h, --help
kube-scheduler 帮助命令
--http2-max-streams-per-connection int
服务器为客户端提供的 HTTP/2 连接最大限制。零表示使用 golang 的默认值。
--kube-api-burst int32 默认: 100
弃用: 与 kubernetes apiserver 通信时使用
--kube-api-content-type string 默认: "application/vnd.kubernetes.protobuf"
弃用: 发送到 apiserver 的请求的内容类型。
--kube-api-qps float32 默认: 50
弃用: 与 kubernetes apiserver 通信时要使用的 QPS
--kubeconfig string
弃用: 具有授权和主节点位置信息的 kubeconfig 文件的路径。
--leader-elect 默认: true
在执行主循环之前，开始领导者选举并选出领导者。为实现高可用性，运行多副本的组件并选出领导者。
--leader-elect-lease-duration duration 默认: 15s
非领导者候选人在观察到领导者更新后将等待直到试图获得领导但未更新的领导者职位的等待时间。这实际上是领导者在被另一位候选人替代之前可以停止的最大持续时间。该情况仅在启用了领导者选举的情况下才适用。
--leader-elect-renew-deadline duration 默认: 10s
领导者尝试在停止领导之前更新领导职位的间隔时间。该时间必须小于或等于租赁期限。仅在启用了领导者选举的情况下才适用。
--leader-elect-resource-lock endpoints 默认: "endpoints"
在领导者选举期间用于锁定的资源对象的类型。支持的选项是端点（默认）和 `configmaps`
--leader-elect-resource-name string 默认: "kube-scheduler"
在领导者选举期间用于锁定的资源对象的名称。
--leader-elect-resource-namespace string 默认: "kube-system"
在领导者选举期间用于锁定的资源对象的命名空间。
--leader-elect-retry-period duration 默认: 2s
客户应在尝试获取和更新领导之间等待的时间。仅在启用了领导者选举的情况下才适用。
--lock-object-name string 默认: "kube-scheduler"
弃用: 定义锁对象的名称。将被删除以便使用 Leader-elect-resource-name
--lock-object-namespace string 默认: "kube-system"

	弃用: 定义锁对象的命名空间。将被删除以便使用 leader-elect-resource-namespace。
--log-backtrace-at traceLocation 默认: :0	
	当记录命中行文件：N 时发出堆栈跟踪
--log-dir string	
	如果为非空，则在此目录中写入日志文件
--log-file string	
	如果为非空，请使用此日志文件
--log-file-max-size uint 默认: 1800	
	定义日志文件可以增长到的最大值。单位为兆字节。如果值为0，则最大文件大小为无限制。
--log-flush-frequency duration 默认: 5s	
	两次日志刷新之间的最大秒数
--logtostderr 默认: true	
	日志记录到标准错误而不是文件
--master string	
	Kubernetes API 服务器的地址（覆盖 kubeconfig 中的任何值）
--policy-config-file string	
	弃用：具有调度程序策略配置的文件。如果未提供 policy ConfigMap 或 --use-legacy-policy-config = true，则使用此文件
--policy-configmap string	
	弃用: 包含调度程序策略配置的 ConfigMap 对象的名称。如果 --use-legacy-policy-config = false，则它必须在调度程序初始化之前存在于系统命名空间中。必须将配置作为键为 'policy.cfg' 的 'Data' 映射中元素的值提供
--policy-configmap-namespace string 默认: "kube-system"	
	弃用: 策略 ConfigMap 所在的命名空间。如果未提供或为空，则将使用 kube-system 命名空间。
--port int 默认: 10251	
	弃用: 在没有身份验证和授权的情况下不安全地为 HTTP 服务的端口。如果为0，则根本不提供 HTTP。请参见--secure-port。
--profiling	
	弃用: 通过 Web 界面主机启用配置文件：port/debug/pprof/
--requestheader-allowed-names stringSlice	
	客户端证书通用名称列表允许在 --requestheader-username-headers 指定的头部中提供用户名。如果为空，则允许任何由权威机构 --requestheader-client-ca-file 验证的客户端证书。
--requestheader-client-ca-file string	
	在信任 --requestheader-username-headers 指定的头部中的用户名之前用于验证传入请求上的客户端证书的根证书包。警告：通常不依赖于传入请求已经完成的授权。
--requestheader-extra-headers-prefix stringSlice 默认: [x-remote-extra-]	
	要检查请求头部前缀列表。建议使用 X-Remote-Extra-
--requestheader-group-headers stringSlice 默认: [x-remote-group]	

	用于检查组的请求头部列表。建议使用 X-Remote-Group。
--requestheader-username-headers stringSlice　默认: [x-remote-user]	
	用于检查用户名的请求头部列表。 X-Remote-User 很常见。
--scheduler-name string　默认: "default-scheduler"	
	弃用: 调度程序名称用于根据 Pod 的 "spec.schedulerName" 选择此调度程序将处理的 Pod。
--secure-port int　默认: 10259	
	通过身份验证和授权为 HTTPS 服务的端口。如果为 0，则根本不提供 HTTPS。
--skip-headers	
	如果为 true，请在日志消息中避免头部前缀
--skip-log-headers	
	如果为true，则在打开日志文件时避免头部
--stderrthreshold severity　默认: 2	
	达到或超过此阈值的日志转到 stderr
--tls-cert-file string	
	包含默认的 HTTPS x509 证书的文件。（CA证书（如果有）在服务器证书之后并置）。如果启用了 HTTPS 服务，并且未提供 --tls-cert-file 和 --tls-private-key-file，则会为公共地址生成一个自签名证书和密钥，并将其保存到 --cert-dir 指定的目录中。
--tls-cipher-suites stringSlice	
	服务器的密码套件列表，以逗号分隔。如果省略，将使用默认的 Go 密码套件。可能的值： TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_RC4_128_SHA
--tls-min-version string	
	支持的最低 TLS 版本。可能的值：VersionTLS10, VersionTLS11, VersionTLS12, VersionTLS13
--tls-private-key-file string	
	包含与 --tls-cert-file 匹配的默认 x509 私钥的文件。
--tls-sni-cert-key namedCertKey　默认: []	

	一对 x509 证书和私钥文件路径，可选地后缀为完全限定域名的域模式列表，并可能带有前缀的通配符段。如果未提供域模式，则获取证书名称。非通配符匹配胜过通配符匹配，显式域模式胜过获取名称。 对于多个密钥/证书对，请多次使用 --tls-sni-cert-key。例如: "example.crt,example.key" 或者 "foo.crt,foo.key:*.foo.com,foo.com"。
--use-legacy-policy-config	
	弃用: 设置为 true 时，调度程序将忽略策略 ConfigMap 并使用策略配置文件
-v, --v Level	
	日志级别详细程度的数字
--version version[=true]	
	打印版本信息并退出
--vmodule moduleSpec	
	以逗号分隔的 pattern = N 设置列表，用于文件过滤的日志记录
--write-config-to string	
	如果已设置，请将配置值写入此文件并退出。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 June 29, 2020 at 6:50 PM PST: Fix page front matter (zh localization) (30b8110ee)

kubelet

简介

kubelet 是在每个 Node 节点上运行的主要 "节点代理"。它可以使用以下之一向 apiserver 注册： 主机名（hostname）；覆盖主机名的参数；某云驱动的特定逻辑。

kubelet 是基于 PodSpec 来工作的。每个 PodSpec 是一个描述 Pod 的 YAML 或 JSON 对象。 kubelet 接受通过各种机制（主要是通过 apiserver）提供的一组 PodSpec，并确保这些 PodSpec 中描述的容器处于运行状态且运行状况良好。 kubelet 不管理不是由 Kubernetes 创建的容器。

除了来自 apiserver 的 PodSpec 之外，还可以通过以下三种方式将容器清单（manifest）提供给 kubelet。

文件（File）：利用命令行参数传递路径。kubelet 周期性地监视此路径下的文件是否有更新。 监视周期默认为 20s，且可通过参数进行配置。

HTTP 端点（HTTP endpoint）：利用命令行参数指定 HTTP 端点。 此端点的监视周期默认为 20 秒，也可以使用参数进行配置。

HTTP 服务器（HTTP server）：kubelet 还可以侦听 HTTP 并响应简单的 API （目前没有完整规范）来提交新的清单。

```
kubelet [flags]
```

选项

--add-dir-header
设置为 true 表示将文件目录添加到日志消息的头部
--address ip　默认值：0.0.0.0
kubelet 用来提供服务的 IP 地址（设置为0.0.0.0 表示使用所有 IPv4 接口， 设置为 :: 表示使用所有 IPv6 接口）。已弃用：应在 --config 所给的 配置文件中进行设置。（进一步了解）
--allowed-unsafe-sysctls strings
用逗号分隔的字符串序列设置允许使用的非安全的 sysctls 或 sysctl 模式（以 * 结尾）。 使用此参数时风险自担。已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）.
--alsologtostderr
设置为 true 表示将日志输出到文件的同时输出到 stderr
--anonymous-auth　默认值：true
设置为 true 表示 kubelet 服务器可以接受匿名请求。未被任何认证组件拒绝的请求将被视为匿名请求。 匿名请求的用户名为 system:anonymous，用户组为 system:unauthenticated。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--authentication-token-webhook
使用 TokenReview API 对持有者令牌进行身份认证。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--authentication-token-webhook-cache-ttl duration　默认值：2m0s
对 Webhook 令牌认证组件所返回的响应的缓存时间。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--authorization-mode string
kubelet 服务器的鉴权模式。可选值包括：AlwaysAllow、Webhook。Webhook 模式使用 SubjectAccessReview API 鉴权。 当 --config 参数未被设置时，默认值为 AlwaysAllow，当使用了 --config 时，默认值为 Webhook。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--authorization-webhook-cache-authorized-ttl duration　默认值：5m0s
对 Webhook 认证组件所返回的 "Authorized（已授权）" 应答的缓存时间。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--authorization-webhook-cache-unauthorized-ttl duration　默认值：30s

	对 Webhook 认证组件所返回的 "Unauthorized（未授权）" 应答的缓存时间。 --config 时，默认值为 Webhook。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--azure-container-registry-config string	
	包含 Azure 容器镜像库配置信息的文件的路径。
--bootstrap-kubeconfig string	
	某 kubeconfig 文件的路径，该文件将用于获取 kubelet 的客户端证书。 如果 --kubeconfig 所指定的文件不存在，则使用引导所用 kubeconfig 从 API 服务器请求客户端证书。成功后，将引用生成的客户端证书和密钥的 kubeconfig 写入 --kubeconfig 所指定的路径。客户端证书和密钥文件将存储在 --cert-dir 所指的目录。
--cert-dir string 默认值：/var/lib/kubelet/pki	
	TLS 证书所在的目录。如果设置了 --tls-cert-file 和 --tls-private-key-file， 则此标志将被忽略。
--cgroup-driver string 默认值：cgroupfs	
	kubelet 用来操作本机 cgroup 时使用的驱动程序。支持的选项包括 cgroupfs 和 systemd。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解） /td>
--cgroup-root string 默认值：""	
	可选的选项，为 Pod 设置根 cgroup。容器运行时会尽可能使用此配置。 默认值 "" 意味着将使用容器运行时的默认设置。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cgroups-per-qos 默认值：true	
	启用创建 QoS cgroup 层次结构。此值为 true 时 kubelet 为 QoS 和 Pod 创建顶级的 cgroup。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--chaos-chance float	
	如果此值大于 0.0，则引入随机客户端错误和延迟。用于测试。 已启用：将在未来版本中移除。
--client-ca-file string	
	如果设置了此参数，则使用对应文件中机构之一检查请求中所携带的客户端证书。 若客户端证书通过身份认证，则其对应身份为其证书中所设置的 CommonName。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cloud-config string	
	云驱动配置文件的路径。空字符串表示没有配置文件。 已弃用：将在 1.23 版本中移除，以便于从 kubelet 中去除云驱动代码。
--cloud-provider string	
	云服务的提供者。设置为空字符串表示在没有云驱动的情况下运行。 如果设置了此标志，则云驱动负责确定节点的名称（参考云提供商文档以确定是否以及如何使用主机名）。 已弃用：将在 1.23 版本中移除，以便于从 kubelet 中去除云驱动代码。
--cluster-dns strings	

DNS 服务器的 IP 地址，以逗号分隔。此标志值用于 Pod 中设置了 "dnsPolicy=ClusterFirst" 时为容器提供 DNS 服务。注意：列表中出现的所有 DNS 服务器必须包含相同的记录组， 否则集群中的名称解析可能无法正常工作。至于名称解析过程中会牵涉到哪些 DNS 服务器， 这一点无法保证。 --config 时，默认值为 Webhook。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cluster-domain string
集群的域名。如果设置了此值，kubelet 除了将主机的搜索域配置到所有容器之外，还会为其 配置所搜这里指定的域名。 --config 时，默认值为 Webhook。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cni-bin-dir string 默认值：/opt/cni/bin
<警告：alpha 特性> 此值为以逗号分隔的完整路径列表。 kubelet 将在所指定路径中搜索 CNI 插件的可执行文件。 仅当容器运行环境设置为 docker 时，此特定于 docker 的参数才有效。
--cni-cache-dir string 默认值：/var/lib/cni/cache
<警告：alpha 特性> 此值为一个目录的全路径名。CNI 将在其中缓存文件。 仅当容器运行环境设置为 docker 时，此特定于 docker 的参数才有效。
--cni-conf-dir string 默认值：/etc/cni/net.d
<警告：alpha 特性> 此值为某目录的全路径名。kubelet 将在其中搜索 CNI 配置文件。 仅当容器运行环境设置为 docker 时，此特定于 docker 的参数才有效。
--config string
kubelet 将从此标志所指的文件中加载其初始配置。此路径可以是绝对路径，也可以是相对路径。 相对路径按 kubelet 的当前工作目录起计。省略此参数时 kubelet 会使用内置的默认配置值。 命令行参数会覆盖此文件中的配置。
--container-log-max-files int32 默认值：5
<警告：beta 特性> 设置容器的日志文件个数上限。此值必须不小于 2。 此标志只能与 --container-runtime=remote 标志一起使用。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--container-log-max-size string 默认值：10Mi
<警告：beta 特性> 设置容器日志文件在轮换生成新文件时之前的最大值（例如，10Mi）。 此标志只能与 --container-runtime=remote 标志一起使用。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--container-runtime string 默认值：docker
要使用的容器运行时。目前支持 docker、remote。
--container-runtime-endpoint string 默认值：unix:///var/run/dockershim.sock
[实验性特性] 远程运行时服务的端点。目前支持 Linux 系统上的 UNIX 套接字和 Windows 系统上的 npipe 和 TCP 端点。例如： unix:///var/run/dockershim.sock、 npipe:////./pipe/dockershim。
--contention-profiling
当启用了性能分析时，启用锁竞争分析。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cpu-cfs-quota 默认值：true

	为设置了 CPU 限制的容器启用 CPU CFS 配额保障。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cpu-cfs-quota-period duration	默认值：100ms
	设置 CPU CFS 配额周期 cpu.cfs_period_us。默认使用 Linux 内核所设置的默认值。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cpu-manager-policy string	默认值：none
	要使用的 CPU 管理器策略。可选值包括：none 和 static。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--cpu-manager-reconcile-period duration	默认值：10s
	<警告：alpha 特性> 设置 CPU 管理器的调和时间。例如：10s 或者 1m。 如果未设置，默认使用节点状态更新频率。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--docker-endpoint string	默认值：unix:///var/run/docker.sock
	使用这里的端点与 docker 端点通信。 仅当容器运行环境设置为 docker 时，此特定于 docker 的参数才有效。
--dynamic-config-dir string	
	kubelet 使用此目录来保存所下载的配置，跟踪配置运行状况。 如果目录不存在，则 kubelet 创建该目录。此路径可以是绝对路径，也可以是相对路径。 相对路径从 kubelet 的当前工作目录计算。 设置此参数将启用动态 kubelet 配置。必须启用 DynamicKubeletConfig 特性门控之后才能设置此标志；由于此特性为 beta 阶段，对应的特性门控当前默认为 true。
--enable-cadvisor-json-endpoints	Default: `false`
	启用 cAdvisor JSON 数据的 /spec 和 /stats/* 端点。 已弃用：未来版本将会移除此标志。
--enable-controller-attach-detach	默认值：true
	启用 Attach/Detach 控制器来挂接和摘除调度到该节点的卷，同时禁用 kubelet 执行挂接和摘除操作。
--enable-debugging-handlers	Default: `true`
	启用服务器上用于日志收集和在本地运行容器和命令的端点。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--enable-server	
	启用 kubelet 服务器。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--enforce-node-allocatable strings	Default: `pods`
	用逗号分隔的列表，包含由 kubelet 强制执行的节点可分配资源级别。 可选配置为：none、pods、system-reserved 和 kube-reserved。 在设置 system-reserved 和 kube-reserved 这两个值时，同时要求设置 --system-reserved-cgroup 和 --kube-reserved-cgroup 这两个参数。 如果设置为 none，则不需要设置其他参数。 参考相关文档。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--event-burst int32	默认值：10

事件记录的个数的突发峰值上限，在遵从 --event-qps 阈值约束的前提下 临时允许事件记录达到此数目。仅在 --event-qps 大于 0 时使用。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--event-qps int32 Default: 5
设置大于 0 的值表示限制每秒可生成的事件数量。设置为 0 表示不限制。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--eviction-hard string 默认值：imagefs.available<15%,memory.available<100Mi,nodefs.available<10%
触发 Pod 驱逐操作的一组硬性门限（例如：memory.available<1Gi （内存可用值小于 1 G））设置。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--eviction-max-pod-grace-period int32
响应满足软性驱逐阈值（Soft Eviction Threshold）而终止 Pod 时使用的最长宽限期（以秒为单位）。 如果设置为负数，则遵循 Pod 的指定值。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--eviction-minimum-reclaim mapStringString
当某资源压力过大时，kubelet 将执行 Pod 驱逐操作。 此参数设置软性驱逐操作需要回收的资源的最小数量（例如：imagefs.available=2Gi）。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--eviction-pressure-transition-period duration 默认值：5m0s
kubelet 在驱逐压力状况解除之前的最长等待时间。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--eviction-soft mapStringString
设置一组驱逐阈值（例如：memory.available<1.5Gi）。 如果在相应的宽限期内达到该阈值，则会触发 Pod 驱逐操作。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--eviction-soft-grace-period mapStringString
设置一组驱逐宽限期（例如，memory.available=1m30s），对应于触发软性 Pod 驱逐操作之前软性驱逐阈值所需持续的时间长短。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--exit-on-lock-contention
设置为 true 表示当发生锁文件竞争时 kubelet 可以退出。
--experimental-allocatable-ignore-eviction 默认值：false
设置为 true 表示在计算节点可分配资源数量时忽略硬性逐出阈值设置。 参考 相关文档。 已启用：将在 1.23 版本中移除。
--experimental-bootstrap-kubeconfig string
已弃用：应使用 --bootstrap-kubeconfig 标志
--experimental-check-node-capabilities-before-mount
[实验性特性] 设置为 true 表示 kubelet 在进行挂载卷操作之前要 在本节点上检查所需的组件（如可执行文件等）是否存在。 已弃用：将在 1.23 版本中移除，以便使用 CSI。
--experimental-kernel-memcg-notification

	设置为 true 表示 kubelet 将会集成内核的 memcg 通知机制而不是使用轮询机制来判断是否达到了内存驱逐阈值。 此标志将在 1.23 版本移除。 已弃用：应在 `--config` 所给的配置文件中进行设置。 （进一步了解）
--experimental-log-sanitization	
	[试验性功能] 启用此标志之后，kubelet 会避免将标记为敏感的字段（密码、密钥、令牌等）写入日志中。运行时的日志清理可能会带来相当的计算开销，因此不应该在 产品环境中启用。 已弃用：应在 `--config` 所给的配置文件中进行设置。 （进一步了解）
--experimental-mounter-path string　默认值：mount	
	[实验性特性] 卷挂载器（mounter）的可执行文件的路径。设置为空表示使用默认挂载器 mount。 已弃用：将在 1.23 版本移除以支持 CSI。
--fail-swap-on　默认值：true	
	设置为 true 表示如果主机启用了交换分区，kubelet 将直接失败。 已弃用：应在 `--config` 所给的配置文件中进行设置。 （进一步了解）
--feature-gates mapStringBool	

用于 alpha 实验性质的特性开关组，每个开关以 key=value 形式表示。当前可用开关包括： APIListChunking=true|false (BETA - 默认值为 true)
APIPriorityAndFairness=true|false (BETA - 默认值为 true)
APIResponseCompression=true|false (BETA - 默认值为 true)
APIServerIdentity=true|false (ALPHA - 默认值为 false)
AllAlpha=true|false (ALPHA - 默认值为 false)
AllBeta=true|false (BETA - 默认值为 false)
AllowInsecureBackendProxy=true|false (BETA - 默认值为 true)
AnyVolumeDataSource=true|false (ALPHA - 默认值为 false)
AppArmor=true|false (BETA - 默认值为 true)
BalanceAttachedNodeVolumes=true|false (ALPHA - 默认值为 false)
BoundServiceAccountTokenVolume=true|false (ALPHA - 默认值为 false)
CPUManager=true|false (BETA - 默认值为 true)
CRIContainerLogRotation=true|false (BETA - 默认值为 true)
CSIInlineVolume=true|false (BETA - 默认值为 true)
CSIMigration=true|false (BETA - 默认值为 true)
CSIMigrationAWS=true|false (BETA - 默认值为 false)
CSIMigrationAWSComplete=true|false (ALPHA - 默认值为 false)
CSIMigrationAzureDisk=true|false (BETA - 默认值为 false)
CSIMigrationAzureDiskComplete=true|false (ALPHA - 默认值为 false)
CSIMigrationAzureFile=true|false (ALPHA - 默认值为 false)
CSIMigrationAzureFileComplete=true|false (ALPHA - 默认值为 false)
CSIMigrationGCE=true|false (BETA - 默认值为 false)
CSIMigrationGCEComplete=true|false (ALPHA - 默认值为 false)
CSIMigrationOpenStack=true|false (BETA - 默认值为 false)
CSIMigrationOpenStackComplete=true|false (ALPHA - 默认值为 false)
CSIMigrationvSphere=true|false (BETA - 默认值为 false)
CSIMigrationvSphereComplete=true|false (BETA - 默认值为 false)
CSIServiceAccountToken=true|false (ALPHA - 默认值为 false)
CSIStorageCapacity=true|false (ALPHA - 默认值为 false)
CSIVolumeFSGroupPolicy=true|false (BETA - 默认值为 true)
ConfigurableFSGroupPolicy=true|false (BETA - 默认值为 true)
CronJobControllerV2=true|false (ALPHA - 默认值为 false)
CustomCPUCFSQuotaPeriod=true|false (ALPHA - 默认值为 false)
DefaultPodTopologySpread=true|false (BETA - 默认值为 true)
DevicePlugins=true|false (BETA - 默认值为 true)
DisableAcceleratorUsageMetrics=true|false (BETA - 默认值为 true)
DownwardAPIHugePages=true|false (ALPHA - 默认值为 false)
DynamicKubeletConfig=true|false (BETA - 默认值为 true)
EfficientWatchResumption=true|false (ALPHA - 默认值为 false)
EndpointSlice=true|false (BETA - 默认值为 true)
EndpointSliceNodeName=true|false (ALPHA - 默认值为 false)
EndpointSliceProxying=true|false (BETA - 默认值为 true)
EndpointSliceTerminatingCondition=true|false (ALPHA - 默认值为 false)
EphemeralContainers=true|false (ALPHA - 默认值为 false)
ExpandCSIVolumes=true|false (BETA - 默认值为 true)
ExpandInUsePersistentVolumes=true|false (BETA - 默认值为 true)
ExpandPersistentVolumes=true|false (BETA - 默认值为 true)

--file-check-frequency duration　默认值：20s
检查配置文件中新数据的时间间隔。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--hairpin-mode string　默认值：promiscuous-bridge
设置 kubelet 执行发夹模式（hairpin）网络地址转译的方式。 该模式允许后端端点对其自身服务的访问能够再次经由负载均衡转发回自身。 可选项包括 "promiscuous-bridge"、"hairpin-veth" 和 "none"。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--healthz-bind-address ip　默认值：127.0.0.1
用于运行 healthz 服务器的 IP 地址（设置为 0.0.0.0 表示使用所有 IPv4 接口， 设置为 :: 表示使用所有 IPv6 接口。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--healthz-port int32　默认值：10248
本地 healthz 端点使用的端口（设置为 0 表示禁用）。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
-h, --help
kubelet 操作的帮助命令
--hostname-override string
如果为非空，将使用此字符串而不是实际的主机名作为节点标识。如果设置了 --cloud-provider，则云驱动将确定节点的名称（请查阅云服务商文档以确定是否以及如何使用主机名）。
--housekeeping-interval duration　默认值：10s
清理容器操作的时间间隔。
--http-check-frequency duration　默认值：20s
HTTP 服务以获取新数据的时间间隔。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--image-credential-provider-bin-dir string
指向凭据提供组件可执行文件所在目录的路径。
--image-credential-provider-config string
指向凭据提供插件配置文件所在目录的路径。
--image-gc-high-threshold int32　默认值：85
镜像垃圾回收上限。磁盘使用空间达到该百分比时，镜像垃圾回收将持续工作。 值必须在 [0，100] 范围内。要禁用镜像垃圾回收，请设置为 100。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--image-gc-low-threshold int32　默认值：80
镜像垃圾回收下限。磁盘使用空间在达到该百分比之前，镜像垃圾回收操作不会运行。值必须在 [0，100] 范围内，并且不得大于 --image-gc-high-threshold的值。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--image-pull-progress-deadline duration　默认值：1m0s
如果在该参数值所设置的期限之前没有拉取镜像的进展，镜像拉取操作将被取消。 仅当容器运行环境设置为 docker 时，此特定于 docker 的参数才有效。
--image-service-endpoint string

	[实验性特性] 远程镜像服务的端点。若未设定则默认情况下使用 --container-runtime-endpoint 的值。目前支持的类型包括在 Linux 系统上的 UNIX 套接字端点和 Windows 系统上的 npipe 和 TCP 端点。 例如：unix:///var/run/dockershim.sock、npipe:////./pipe/dockershim。
--iptables-drop-bit int32　　默认值：15	
	标记数据包将被丢弃的 fwmark 位设置。必须在 [0，31] 范围内。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--iptables-masquerade-bit int32　　默认值：14	
	标记数据包将进行 SNAT 的 fwmark 空间位设置。必须在 [0，31] 范围内。 请将此参数与 kube-proxy 中的相应参数匹配。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--keep-terminated-pod-volumes	
	设置为 true 表示 Pod 终止后仍然保留之前挂载过的卷，常用于调试与卷有关的问题。已弃用：将未来版本中移除。
--kernel-memcg-notification	
	若启用，则 kubelet 将与内核中的 memcg 通知机制集成，不再使用轮询的方式来判定 是否 Pod 达到内存驱逐阈值。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--kube-api-burst int32　　默认值：10	
	每秒发送到 apiserver 的突发请求数量上限。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--kube-api-content-type string　　默认值：application/vnd.kubernetes.protobuf	
	发送到 apiserver 的请求的内容类型。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--kube-api-qps int32　　默认值：5	
	与 apiserver 通信的每秒查询个数（QPS）。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--kube-reserved mapStringString　　默认值：<None>	
	kubernetes 系统预留的资源配置，以一组 资源名称=资源数量 格式表示。（例如：cpu=200m,memory=500Mi,ephemeral-storage=1Gi）。 当前支持 cpu、memory 和用于根文件系统的 ephemeral-storage。 请参阅相关文档获取更多信息。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--kube-reserved-cgroup string　　默认值：""	
	给出某个顶层 cgroup 绝对名称，该 cgroup 用于管理通过标志 --kube-reserved 为 kubernetes 组件所预留的计算资源。例如："/kube-reserved"。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--kubeconfig string	
	kubeconfig 配置文件的路径，指定如何连接到 API 服务器。 提供 --kubeconfig 将启用 API 服务器模式，而省略 --kubeconfig 将启用独立模式。
--kubelet-cgroups string	
	用于创建和运行 kubelet 的 cgroup 的绝对名称。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）

--lock-file string
<警告：alpha 特性> kubelet 使用的锁文件的路径。
--log-backtrace-at traceLocation 默认值：:0
形式为 <file>:<N>。 当日志逻辑执行到命中 <file> 的第 <N> 行时，转储调用堆栈。
--log-dir string
如果此值为非空，则在所指定的目录中写入日志文件。
--log-file string
如果此值非空，使用所给字符串作为日志文件名。
--log-file-max-size uint 默认值：1800
设置日志文件的最大值。单位为兆字节（M）。如果值为 0，则表示文件大小无限制。
--log-flush-frequency duration 默认值：5s
两次日志刷新之间的最大秒数（默认值为 5s）。
--logging-format string 默认值："text"
设置日志文件格式。可以设置的格式有："text"、"json"。 非默认的格式不会使用以下标志的配置：--add-dir-header, --alsologtostderr, --log-backtrace-at, --log-dir, --log-file, --log-file-max-size, --logtostderr, --skip-headers, --skip-log-headers, --stderrthreshold, --log-flush-frequency。 非默认选项的其它值都应视为 Alpha 特性，将来出现更改时不会额外警告。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--logtostderr 默认值：true
日志输出到 stderr 而不是文件。
--make-iptables-util-chains 默认值：true
设置为 true 表示 kubelet 将确保 iptables 规则在主机上存在。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--manifest-url string
用于访问要运行的其他 Pod 规范的 URL。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--manifest-url-header string
取值为由 HTTP 头部组成的逗号分隔列表，在访问 --manifest-url 所给出的 URL 时使用。 名称相同的多个头部将按所列的顺序添加。该参数可以多次使用。例如： --manifest-url-header 'a:hello,b:again,c:world' --manifest-url-header 'b:beautiful'。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--master-service-namespace string 默认值：default
kubelet 向 Pod 注入 Kubernetes 主控服务信息时使用的命名空间。 已弃用：此标志将在未来的版本中删除。
--max-open-files int 默认值：1000000
kubelet 进程可以打开的最大文件数量。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--max-pods int32 默认值：110
此 kubelet 能运行的 Pod 最大数量。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）

--maximum-dead-containers int32　默认值：-1
设置全局可保留的已停止容器实例个数上限。 每个实例会占用一些磁盘空间。要禁用，请设置为负数。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--maximum-dead-containers-per-container int32　默认值：1
每个已停止容器可以保留的的最大实例数量。每个容器占用一些磁盘空间。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--minimum-container-ttl-duration duration
已结束的容器在被垃圾回收清理之前的最少存活时间。 例如：300ms、10s 或者 2h45m。 已弃用：请改用 --eviction-hard 或者 --eviction-soft。 此标志将在未来的版本中删除。
--minimum-image-ttl-duration duration　默认值：2m0s
不再使用的镜像在被垃圾回收清理之前的最少存活时间。 例如：300ms、10s 或者 2h45m。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--network-plugin string
<警告：alpha 特性> 设置 kubelet/Pod 生命周期中各种事件调用的网络插件的名称。 仅当容器运行环境设置为 docker 时，此特定于 docker 的参数才有效。
--network-plugin-mtu int32
<警告：alpha 特性> 传递给网络插件的 MTU 值，将覆盖默认值。 设置为 0 则使用默认的 MTU 1460。仅当容器运行环境设置为 docker 时， 此特定于 docker 的参数才有效。
--node-ip string
节点的 IP 地址。如果设置，kubelet 将使用该 IP 地址作为节点的 IP 地址。
--node-labels mapStringString
<警告：alpha 特性> kubelet 在集群中注册本节点时设置的标签。标签以 key=value 的格式表示，多个标签以逗号分隔。名字空间 kubernetes.io 中的标签必须以 kubelet.kubernetes.io 或 node.kubernetes.io 为前缀， 或者在以下明确允许范围内： beta.kubernetes.io/arch, beta.kubernetes.io/instance-type, beta.kubernetes.io/os, failure-domain.beta.kubernetes.io/region, failure-domain.beta.kubernetes.io/zone, kubernetes.io/arch, kubernetes.io/hostname, kubernetes.io/os, node.kubernetes.io/instance-type, topology.kubernetes.io/region, topology.kubernetes.io/zone。
--node-status-max-images int32　默认值：50
在 node.status.images 中可以报告的最大镜像数量。如果指定为 -1，则不设上限。已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--node-status-update-frequency duration　默认值：10s
指定 kubelet 向主控节点汇报节点状态的时间间隔。注意：更改此常量时请务必谨慎， 它必须与节点控制器中的 nodeMonitorGracePeriod 一起使用。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--non-masquerade-cidr string　默认值：10.0.0.0/8
kubelet 向该 IP 段之外的 IP 地址发送的流量将使用 IP 伪装技术。 设置为 0.0.0.0/0 则不使用伪装。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--one-output

如果设置此标志为 true，则仅将日志写入其原来的严重性级别中， 而不是同时将其写入更低严重性级别中。
--oom-score-adj int32　默认值：-999
kubelet 进程的 oom-score-adj 参数值。有效范围为 [-1000，1000]。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--pod-cidr string
用于给 Pod 分配 IP 地址的 CIDR 地址池，仅在独立运行模式下使用。 在集群模式下，CIDR 设置是从主服务器获取的。对于 IPv6，分配的 IP 的最大数量为 65536。已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--pod-infra-container-image string　默认值：k8s.gcr.io/pause:3.2
指定基础设施镜像，Pod 内所有容器与其共享网络和 IPC 命名空间。 仅当容器运行环境设置为 docker 时，此特定于 docker 的参数才有效。
--pod-manifest-path string
设置包含要运行的静态 Pod 的文件的路径，或单个静态 Pod 文件的路径。以点（.）开头的文件将被忽略。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--pod-max-pids int　默认值：-1
设置每个 Pod 中的最大进程数目。如果为 -1，则 kubelet 使用节点可分配的 PID 容量作为默认值。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--pods-per-core int32
kubelet 在每个处理器核上可运行的 Pod 数量。此 kubelet 上的 Pod 总数不能超过 --max-pods 标志值。因此，如果此计算结果导致在 kubelet 上允许更多数量的 Pod，则使用 --max-pods 值。值为 0 表示不作限制。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--port int32　默认值：10250
kubelet 服务监听的本机端口号。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--protect-kernel-defaults
设置 kubelet 的默认内核调整行为。如果已设置该参数，当任何内核可调参数与 kubelet 默认值不同时，kubelet 都会出错。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--provider-id string
设置主机数据库（即，云驱动）中用来标识节点的唯一标识。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--qos-reserved mapStringString
<警告：alpha 特性> 设置在指定的 QoS 级别预留的 Pod 资源请求，以一组 "资源名称=百分比" 的形式进行设置，例如 memory=50%。 当前仅支持内存（memory）。要求启用 QOSReserved 特性门控。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--read-only-port int32　默认值：10255
kubelet 可以在没有身份验证/鉴权的情况下提供只读服务的端口（设置为 0 表示禁用）。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--really-crash-for-testing

设置为 true 表示发生失效时立即崩溃。仅用于测试。 已弃用：将在未来版本中移除。
--redirect-container-streaming
启用容器流数据重定向。如果设置为 false，则 kubelet 将在 apiserver 和容器运行时 之间转发容器流数据；如果设置为 true，则 kubelet 将返回指向 apiserver 的 HTTP 重定向信息， 而 apiserver 将直接访问容器运行时。代理方法更安全，但会带来一些开销。 重定向方法性能更高，但安全性较低，因为 apiserver 和容器运行时之间的连接可能未通过身份验证。 已弃用：容器流数据重定向会在 v1.20 中从 kubelet 中移除，此标志会在 v1.22 中移除。 相关信息可参见改进说明。
--register-node 默认值：true
将本节点注册到 API 服务器。如果未提供 --kubeconfig 标志设置， 则此参数无关紧要，因为 kubelet 将没有要注册的 API 服务器。
--register-schedulable 默认值：true
注册本节点为可调度的节点。当 --register-node标志为 false 时此设置无效。 已弃用：此参数将在未来的版本中删除。
--register-with-taints mapStringString
设置本节点的污点标记，格式为 <key>=<value>:<effect>， 以逗号分隔。当 --register-node 为 false 时此标志无效。 已弃用：将在未来版本中移除。
--registry-burst int32 默认值：10
设置突发性镜像拉取的个数上限，在不超过 --registration-qps 设置值的前提下 暂时允许此参数所给的镜像拉取个数。仅在 --registry-qps 大于 0 时使用。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--registry-qps int32 Default: 5
如此值大于 0，可用来限制镜像仓库的 QPS 上限。设置为 0，表示不受限制。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--reserved-cpus string
用逗号分隔的一组 CPU 或 CPU 范围列表，给出为系统和 Kubernetes 保留使用的 CPU。 此列表所给出的设置优先于通过 --system-reserved 和 --kube-reskube-reserved 所保留的 CPU 个数配置。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--resolv-conf string 默认值：/etc/resolv.conf
名字解析服务的配置文件名，用作容器 DNS 解析配置的基础。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--root-dir string 默认值：/var/lib/kubelet
设置用于管理 kubelet 文件的根目录（例如挂载卷的相关文件等）。
--rotate-certificates
<警告：Beta 特性> 设置当客户端证书即将过期时 kubelet 自动从 kube-apiserver 请求新的证书进行轮换。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--rotate-server-certificates

	当 kubelet 的服务证书即将过期时自动从 kube-apiserver 请求新的证书进行轮换。要求启用 RotateKubeletServerCertificate 特性门控，以及对提交的 CertificateSigningRequest 对象进行批复（Approve）操作。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--runonce	
	设置为 true 表示从本地清单或远程 URL 创建完 Pod 后立即退出 kubelet 进程。 与 --enable-server 标志互斥。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--runtime-cgroups string	
	设置用于创建和运行容器运行时的 cgroup 的绝对名称。
--runtime-request-timeout duration　默认值：2m0s	
	设置除了长时间运行的请求（包括 pull、logs、exec 和 attach 等操作）之外的其他运行时请求的超时时间。 到达超时时间时，请求会被取消，抛出一个错误并会等待重试。已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--seccomp-profile-root string　默认值：/var/lib/kubelet/seccomp	
	<警告：alpha 特性> seccomp 配置文件目录。 已弃用：将在 1.23 版本中移除，以使用 <root-dir>/seccomp 目录。
--serialize-image-pulls　默认值：true	
	逐一拉取镜像。建议 *不要* 在 docker 守护进程版本低于 1.9 或启用了 Aufs 存储后端的节点上 更改默认值。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--skip-headers	
	设置为 true 时在日志消息中去掉标头前缀。
--skip-log-headers	
	设置为 true，打开日志文件时去掉标头。
--stderrthreshold int　默认值：2	
	设置严重程度达到或超过此阈值的日志输出到标准错误输出。
--streaming-connection-idle-timeout duration　默认值：4h0m0s	
	设置流连接在自动关闭之前可以空闲的最长时间。0 表示没有超时限制。 例如：5m。已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--sync-frequency duration　默认值：1m0s	
	在运行中的容器与其配置之间执行同步操作的最长时间间隔。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--system-cgroups /	
	此标志值为一个 cgroup 的绝对名称，用于所有尚未放置在根目录下某 cgroup 内的非内核进程。 空值表示不指定 cgroup。回滚该参数需要重启机器。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--system-reserved mapStringString　默认值：无	
	系统预留的资源配置，以一组 资源名称=资源数量 的格式表示，（例如：cpu=200m,memory=500Mi,ephemeral-storage=1Gi）。 目前仅支持 cpu 和 memory 的设置。 更多细节可参考 相关文档。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）

--system-reserved-cgroup string　默认值：""
此标志给出一个顶层 cgroup 绝对名称，该 cgroup 用于管理非 kubernetes 组件，这些组件的计算资源通过 --system-reserved 标志进行预留。 例如 "/system-reserved"。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--tls-cert-file string
包含 x509 证书的文件路径，用于 HTTPS 认证。 如果有中间证书，则中间证书要串接在在服务器证书之后。 如果未提供 --tls-cert-file 和 --tls-private-key-file，kubelet 会为公开地址生成自签名证书和密钥，并将其保存到通过 --cert-dir 指定的目录中。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--tls-cipher-suites string
服务器端加密算法列表，以逗号分隔。如果不设置，则使用 Go 语言加密包的默认算法列表。 可选加密算法包括： TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_RC4_128_SHA 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--tls-min-version string
设置支持的最小 TLS 版本号，可选的版本号包括：VersionTLS10、 VersionTLS11、VersionTLS12 和 VersionTLS13。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--tls-private-key-file string
包含与 --tls-cert-file 对应的 x509 私钥文件路径。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--topology-manager-policy string　默认值：none
设置拓扑管理策略（Topology Manager policy）。可选值包括：none、 best-effort、restricted 和 single-numa-node。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
--topology-manager-scope string　默认值：container
拓扑提示信息使用范围。拓扑管理器从提示提供者（Hints Providers）处收集提示信息， 并将其应用到所定义的范围以确保 Pod 准入。 可选值包括：container（默认）、pod。 已弃用：应在 --config 所给的配置文件中进行设置。（进一步了解）
-v, --v Level

	设置 kubelet 日志级别详细程度的数值。
--version version[=true]	
	打印 kubelet 版本信息并退出。
--vmodule moduleSpec	
	以逗号分隔的 pattern=N 设置列表，用于文件过滤的日志记录
--volume-plugin-dir string 默认值：/usr/libexec/kubernetes/kubelet-plugins/volume/exec/	
	用来搜索第三方存储卷插件的目录。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）
--volume-stats-agg-period duration 默认值：1m0s	
	指定 kubelet 计算和缓存所有 Pod 和卷的磁盘用量总值的时间间隔。要禁用磁盘用量计算， 请设置为 0。 已弃用：应在 --config 所给的配置文件中进行设置。 （进一步了解）

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 25, 2020 at 5:57 PM PST: [zh] Resync kubelet configuration (709aabf20)

kube-apiserver

简介

Kubernetes API 服务器验证并配置 API 对象的数据， 这些对象包括 pods、services、replicationcontrollers 等。 API 服务器为 REST 操作提供服务，并为集群的共享状态提供前端， 所有其他组件都通过该前端进行交互。

```
kube-apiserver [flags]
```

选项

--add-dir-header	
	如果为 true，则将文件目录添加到日志消息的标题中
--admission-control-config-file string	
	包含准入控制配置的文件。
--advertise-address ip	

	向集群成员通知 apiserver 消息的 IP 地址。 这个地址必须能够被集群中其他成员访问。 如果 IP 地址为空，将会使用 --bind-address， 如果未指定 --bind-address，将会使用主机的默认接口地址。
--allow-privileged	
	如果为 true, 将允许特权容器。[默认值=false]
--alsologtostderr	
	在向文件输出日志的同时，也将日志写到标准输出。
--anonymous-auth 默认值：true	
	启用到 API server 的安全端口的匿名请求。 未被其他认证方法拒绝的请求被当做匿名请求。 匿名请求的用户名为 system:anonymous， 用户组名为 system:unauthenticated。
--api-audiences stringSlice	
	API 的标识符。 服务帐户令牌验证者将验证针对 API 使用的令牌是否已绑定到这些受众中的至少一个。 如果配置了 --service-account-issuer 标志，但未配置此标志， 则此字段默认为包含发行者 URL 的单个元素列表。
--apiserver-count int 默认值：1	
	集群中运行的 apiserver 数量，必须为正数。（在启用 --endpoint-reconciler-type=master-count 时使用。）
--audit-log-batch-buffer-size int 默认值：10000	
	批处理和写入之前用于存储事件的缓冲区大小。 仅在批处理模式下使用。
--audit-log-batch-max-size int 默认值：1	
	批处理的最大大小。 仅在批处理模式下使用。
--audit-log-batch-max-wait duration	
	强制写入尚未达到最大大小的批处理之前要等待的时间。 仅在批处理模式下使用。
--audit-log-batch-throttle-burst int	
	如果之前未使用 ThrottleQPS，则同时发送的最大请求数。 仅在批处理模式下使用。
--audit-log-batch-throttle-enable	
	是否启用了批量限制。 仅在批处理模式下使用。
--audit-log-batch-throttle-qps float32	
	每秒的最大平均批处理数。 仅在批处理模式下使用。
--audit-log-compress	
	若设置了此标志，则轮换的日志文件会使用 gzip 压缩。
--audit-log-format string 默认值："json"	
	所保存的审计格式。 "legacy" 表示每行一个事件的文本格式。"json" 表示结构化的 JSON 格式。 已知格式为 legacy，json。
--audit-log-maxage int	
	根据文件名中编码的时间戳保留旧审计日志文件的最大天数。
--audit-log-maxbackup int	
	保留的旧审计日志文件的最大数量。
--audit-log-maxsize int	
	轮换之前，审计日志文件的最大大小（以兆字节为单位）。
--audit-log-mode string 默认值："blocking"	

	发送审计事件的策略。 阻塞（blocking）表示发送事件应阻止服务器响应。 批处理导致后端异步缓冲和写入事件。 已知的模式是批处理（batch），阻塞（blocking），严格阻塞（blocking-strict）。
--audit-log-path string	
	如果设置，则所有到达 apiserver 的请求都将记录到该文件中。 "-" 表示标准输出。
--audit-log-truncate-enabled	
	是否启用事件和批次截断。
--audit-log-truncate-max-batch-size int 默认值：10485760	
	发送到下层后端的批次的最大数据量。 实际的序列化大小可能会增加数百个字节。 如果一个批次超出此限制，则将其分成几个较小的批次。
--audit-log-truncate-max-event-size int 默认值：102400	
	发送到下层后端的批次的最大数据量。 如果事件的大小大于此数字，则将删除第一个请求和响应， 并且没有减小足够大的程度，则将丢弃事件。
--audit-log-version string 默认值："audit.k8s.io/v1"	
	用于序列化写入日志的审计事件的 API 组和版本。
--audit-policy-file string	
	定义审计策略配置的文件的路径。
--audit-webhook-batch-buffer-size int 默认值：10000	
	划分批次和写入之前用于存储事件的缓冲区大小。 仅在批处理模式下使用。
--audit-webhook-batch-max-size int 默认值：400	
	批次的最大大小。 仅在批处理模式下使用。
--audit-webhook-batch-max-wait duration 默认值：30s	
	强制写入尚未达到最大大小的批处理之前要等待的时间。 仅在批处理模式下使用。
--audit-webhook-batch-throttle-burst int 默认值：15	
	如果之前未使用 ThrottleQPS，则同时发送的最大请求数。 仅在批处理模式下使用。
--audit-webhook-batch-throttle-enable 默认值：true	
	是否启用了批量限制。 仅在批处理模式下使用。
--audit-webhook-batch-throttle-qps float32 默认值：10	
	每秒的最大平均批次数。 仅在批处理模式下使用。
--audit-webhook-config-file string	
	定义审计 webhook 配置的 kubeconfig 格式文件的路径。
--audit-webhook-initial-backoff duration 默认值：10s	
	重试第一个失败的请求之前要等待的时间。
--audit-webhook-mode string 默认值："batch"	
	发送审计事件的策略。 阻止（Blocking）表示发送事件应阻止服务器响应。 批处理导致后端异步缓冲和写入事件。 已知的模式是批处理（batch），阻塞（blocking），严格阻塞（blocking-strict）。
--audit-webhook-truncate-enabled	
	是否启用事件和批处理截断。
--audit-webhook-truncate-max-batch-size int 默认值：10485760	

	发送到下层后端的批次的最大数据量。 实际的序列化大小可能会增加数百个字节。 如果一个批次超出此限制，则将其分成几个较小的批次。
--audit-webhook-truncate-max-event-size int　　默认值：102400	
	发送到下层后端的批次的最大数据量。 如果事件的大小大于此数字，则将删除第一个请求和响应， 并且如果事件和事件的大小没有足够减小，则将丢弃事件。
--audit-webhook-version string　　默认值："audit.k8s.io/v1"	
	用于序列化写入 Webhook 的审计事件的 API 组和版本。
--authentication-token-webhook-cache-ttl duration　　默认值：2m0s	
	来自 Webhook 令牌身份验证器的缓存响应的持续时间。
--authentication-token-webhook-config-file string	
	包含 Webhook 配置的文件，用于以 kubeconfig 格式进行令牌认证。 API 服务器将查询远程服务，以对持有者令牌进行身份验证。
--authentication-token-webhook-version string　　默认值："v1beta1"	
	与 Webhook 之间交换 authentication.k8s.io TokenReview 时使用的 API 版本。
--authorization-mode stringSlice　　默认值：[AlwaysAllow]	
	在安全端口上进行鉴权的插件的顺序列表。 逗号分隔的列表：AlwaysAllow,AlwaysDeny,ABAC,Webhook,RBAC,Node。
--authorization-policy-file string	
	包含安全策略的文件，其内容为分行 JSON 格式， 在安全端口上与 --authorization-mode=ABAC 一起使用。
--authorization-webhook-cache-authorized-ttl duration　　默认值：5m0s	
	缓存来自 Webhook 鉴权组件的 "授权（authorized）" 响应的持续时间。
--authorization-webhook-cache-unauthorized-ttl duration　　默认值：30s	
	缓存来自 Webhook 鉴权模块的 "未授权（unauthorized）" 响应的持续时间。
--authorization-webhook-config-file string	
	包含 Webhook 配置的文件，其格式为 kubeconfig， 与 --authorization-mode=Webhook 一起使用。 API 服务器将查询远程服务，以对 API 服务器的安全端口的访问执行鉴权。
--authorization-webhook-version string　　默认值："v1beta1"	
	与 Webhook 之间交换 authorization.k8s.io SubjectAccessReview 时使用的 API 版本。
--azure-container-registry-config string	
	包含 Azure 容器仓库配置信息的文件的路径。
--bind-address ip　　默认值：0.0.0.0	
	监听 --secure-port 端口的 IP 地址。 集群的其余部分以及 CLI/web 客户端必须可以访问关联的接口。 如果为空白或未指定地址（0.0.0.0 或 ::），则将使用所有接口。
--cert-dir string　　默认值："/var/run/kubernetes"	
	TLS 证书所在的目录。 如果提供了 --tls-cert-file 和 --tls-private-key-file，则将忽略此标志。
--client-ca-file string	
	如果已设置，则使用与客户端证书的 CommonName 对应的标识对任何出示由 client-ca 文件中的授权机构之一签名的客户端证书的请求进行身份验证。

--cloud-config string
云厂商配置文件的路径。 空字符串表示无配置文件。
--cloud-provider string
云服务提供商。 空字符串表示没有云厂商。
--cloud-provider-gce-l7lb-src-cidrs cidrs 默认值：130.211.0.0/22,35.191.0.0/16
在 GCE 防火墙中打开 CIDR，以进行 L7 LB 流量代理和运行状况检查
--contention-profiling
如果启用了性能分析，则启用锁争用性能分析
--cors-allowed-origins stringSlice
CORS 允许的来源清单，以逗号分隔。 允许的来源可以是支持子域匹配的正则表达式。 如果此列表为空，则不会启用 CORS。
--default-not-ready-toleration-seconds int 默认值：300
标明 notReady:NoExecute 的 tolerationSeconds， 默认情况下将其添加到尚未具有此容忍度的每个 pod 中。
--default-unreachable-toleration-seconds int 默认值：300
标明 unreachable:NoExecute 的 tolerationSeconds， 默认情况下将其添加到尚未具有此容忍度的每个 pod 中。
--default-watch-cache-size int 默认值：100
默认监听（watch）缓存大小。 如果为零，则将为没有设置默认监视大小的资源禁用监视缓存。
--delete-collection-workers int 默认值：1
为 DeleteCollection 调用而产生的工作程序数。 这些用于加速名字空间清理。
--disable-admission-plugins stringSlice

	尽管位于默认启用的插件列表中（NamespaceLifecycle、LimitRanger、ServiceAccount、TaintNodesByCondition、Priority、DefaultTolerationSeconds、DefaultStorageClass、StorageObjectInUseProtection、PersistentVolumeClaimResize、RuntimeClass、CertificateApproval、CertificateSigning、CertificateSubjectRestriction、DefaultIngressClass、MutatingAdmissionWebhook、ValidatingAdmissionWebhook、ResourceQuota）仍须被禁用的插件。 取值为逗号分隔的准入插件列表：AlwaysAdmit、AlwaysDeny、AlwaysPullImages、CertificateApproval、CertificateSigning、CertificateSubjectRestriction、DefaultIngressClass、DefaultStorageClass、DefaultTolerationSeconds、DenyEscalatingExec、DenyExecOnPrivileged、EventRateLimit、ExtendedResourceToleration、ImagePolicyWebhook、LimitPodHardAntiAffinityTopology、LimitRanger、MutatingAdmissionWebhook、NamespaceAutoProvision、NamespaceExists、NamespaceLifecycle、NodeRestriction、OwnerReferencesPermissionEnforcement、PersistentVolumeClaimResize、PersistentVolumeLabel、PodNodeSelector、PodSecurityPolicy、PodTolerationRestriction、Priority、ResourceQuota、RuntimeClass、SecurityContextDeny、ServiceAccount、StorageObjectInUseProtection、TaintNodesByCondition、ValidatingAdmissionWebhook。 该标志中插件的顺序无关紧要。
--egress-selector-config-file string	
	带有 apiserver 出站选择器配置的文件。
--enable-admission-plugins stringSlice	

	除了默认启用的插件（NamespaceLifecycle、LimitRanger、ServiceAccount、TaintNodesByCondition、Priority、DefaultTolerationSeconds、DefaultStorageClass、StorageObjectInUseProtection、PersistentVolumeClaimResize、RuntimeClass、CertificateApproval、CertificateSigning、CertificateSubjectRestriction、DefaultIngressClass、MutatingAdmissionWebhook、ValidatingAdmissionWebhook、ResourceQuota）之外要启用的插件取值为逗号分隔的准入插件列表：AlwaysAdmit、AlwaysDeny、AlwaysPullImages、CertificateApproval、CertificateSigning、CertificateSubjectRestriction、DefaultIngressClass、DefaultStorageClass、DefaultTolerationSeconds、DenyEscalatingExec、DenyExecOnPrivileged、EventRateLimit、ExtendedResourceToleration、ImagePolicyWebhook、LimitPodHardAntiAffinityTopology、LimitRanger、MutatingAdmissionWebhook、NamespaceAutoProvision、NamespaceExists、NamespaceLifecycle、NodeRestriction、OwnerReferencesPermissionEnforcement、PersistentVolumeClaimResize、PersistentVolumeLabel、PodNodeSelector、PodSecurityPolicy、PodTolerationRestriction、Priority、ResourceQuota、RuntimeClass、SecurityContextDeny、ServiceAccount、StorageObjectInUseProtection、TaintNodesByCondition、ValidatingAdmissionWebhook 该标志中插件的顺序无关紧要。
--enable-aggregator-routing	
	允许聚合器将请求路由到端点 IP 而非集群 IP。
--enable-bootstrap-token-auth	
	启用以允许将 "kube-system" 名字空间中类型为 "bootstrap.kubernetes.io/token" 的 Secret 用于 TLS 引导身份验证。
--enable-garbage-collector　默认值：true	
	启用通用垃圾收集器。 必须与 kube-controller-manager 的相应标志同步。
--enable-priority-and-fairness　默认值：true	
	如果为 true 且启用了 APIPriorityAndFairness 特性门控， 请使用增强的处理程序替换 max-in-flight 处理程序， 以便根据优先级和公平性完成排队和调度。
--encryption-provider-config string	
	包含加密提供程序配置信息的文件，用在 etcd 中所存储的 Secret 上。
--endpoint-reconciler-type string　默认值："lease"	
	使用端点协调器（master-count, lease, none）
--etcd-cafile string	
	用于保护 etcd 通信的 SSL 证书颁发机构文件。
--etcd-certfile string	
	用于保护 etcd 通信的 SSL 证书文件。
--etcd-compaction-interval duration　默认值：5m0s	
	压缩请求的间隔。 如果为0，则禁用来自 apiserver 的压缩请求。
--etcd-count-metric-poll-period duration　默认值：1m0s	
	针对每种类型的资源数量轮询 etcd 的频率。 0 禁用度量值收集。
--etcd-db-metric-poll-interval duration　默认值：30s	

	轮询 etcd 和更新度量值的请求间隔。 0 禁用度量值收集
--etcd-healthcheck-timeout duration	检查 etcd 健康状况时使用的超时时长。
--etcd-keyfile string	
	用于保护 etcd 通信的 SSL 密钥文件。
--etcd-prefix string	默认值："/registry"
	要在 etcd 中所有资源路径之前添加的前缀。
--etcd-servers stringSlice	
	要连接的 etcd 服务器列表（scheme://ip:port），以逗号分隔。
--etcd-servers-overrides stringSlice	
	etcd 服务器针对每个资源的重载设置，以逗号分隔。 单个替代格式：组/资源#服务器（group/resource#servers），其中服务器是 URL，以分号分隔。
--event-ttl duration	默认值：1h0m0s
	事件的保留时长。
--experimental-logging-sanitization	
	[试验性功能] 启用此标志时，被标记为敏感的字段（密码、密钥、令牌）都不会被日志输出。 运行时的日志清理可能会引入相当程度的计算开销，因此不应该在产品环境中启用。
--external-hostname string	
	为此主机生成外部化 UR L时要使用的主机名（例如 Swagger API 文档或 OpenID 发现）。
--feature-gates mapStringBool	

一组 key=value 对，用来描述测试性/试验性功能的特性门控。可选项有：
APIListChunking=true|false (BETA - 默认值=true)
APIPriorityAndFairness=true|false (BETA - 默认值=true)
APIResponseCompression=true|false (BETA - 默认值=true)
APIServerIdentity=true|false (ALPHA - 默认值=false)
AllAlpha=true|false (ALPHA - 默认值=false)
AllBeta=true|false (BETA - 默认值=false)
AllowInsecureBackendProxy=true|false (BETA - 默认值=true)
AnyVolumeDataSource=true|false (ALPHA - 默认值=false)
AppArmor=true|false (BETA - 默认值=true)
BalanceAttachedNodeVolumes=true|false (ALPHA - 默认值=false)
BoundServiceAccountTokenVolume=true|false (ALPHA - 默认值=false)
CPUManager=true|false (BETA - 默认值=true)
CRIContainerLogRotation=true|false (BETA - 默认值=true)
CSIInlineVolume=true|false (BETA - 默认值=true)
CSIMigration=true|false (BETA - 默认值=true)
CSIMigrationAWS=true|false (BETA - 默认值=false)
CSIMigrationAWSComplete=true|false (ALPHA - 默认值=false)
CSIMigrationAzureDisk=true|false (BETA - 默认值=false)
CSIMigrationAzureDiskComplete=true|false (ALPHA - 默认值=false)
CSIMigrationAzureFile=true|false (ALPHA - 默认值=false)
CSIMigrationAzureFileComplete=true|false (ALPHA - 默认值=false)
CSIMigrationGCE=true|false (BETA - 默认值=false)
CSIMigrationGCEComplete=true|false (ALPHA - 默认值=false)
CSIMigrationOpenStack=true|false (BETA - 默认值=false)
CSIMigrationOpenStackComplete=true|false (ALPHA - 默认值=false)
CSIMigrationvSphere=true|false (BETA - 默认值=false)
CSIMigrationvSphereComplete=true|false (BETA - 默认值=false)
CSIServiceAccountToken=true|false (ALPHA - 默认值=false)
CSIStorageCapacity=true|false (ALPHA - 默认值=false)
CSIVolumeFSGroupPolicy=true|false (BETA - 默认值=true)
ConfigurableFSGroupPolicy=true|false (BETA - 默认值=true)
CronJobControllerV2=true|false (ALPHA - 默认值=false)
CustomCPUCFSQuotaPeriod=true|false (ALPHA - 默认值=false)
DefaultPodTopologySpread=true|false (BETA - 默认值=true)
DevicePlugins=true|false (BETA - 默认值=true)
DisableAcceleratorUsageMetrics=true|false (BETA - 默认值=true)
DownwardAPIHugePages=true|false (ALPHA - default=false)
DynamicKubeletConfig=true|false (BETA - 默认值=true)
EfficientWatchResumption=true|false (ALPHA - 默认值=false)
EndpointSlice=true|false (BETA - 默认值=true)
EndpointSliceNodeName=true|false (ALPHA - 默认值=false)
EndpointSliceProxying=true|false (BETA - 默认值=true)
EndpointSliceTerminatingCondition=true|false (ALPHA - 默认值=false)
EphemeralContainers=true|false (ALPHA - 默认值=false)
ExpandCSIVolumes=true|false (BETA - 默认值=true)
ExpandInUsePersistentVolumes=true|false (BETA - 默认值=true)
ExpandPersistentVolumes=true|false (BETA - 默认值=true)

--goaway-chance float
为防止 HTTP/2 客户端卡在单个 apiserver 上，可启用随机关闭连接（GOAWAY）。 客户端的其他运行中请求将不会受到影响，并且客户端将重新连接，可能会在再次通过负载平衡器后登陆到其他 apiserver 上。 此参数设置将发送 GOAWAY 的请求的比例。 具有单个 apiserver 或不使用负载平衡器的群集不应启用此功能。 最小值为0（关闭），最大值为 .02（1/50 请求）； 建议使用 .001（1/1000）。
-h, --help
kube-apiserver 的帮助命令
--http2-max-streams-per-connection int
服务器为客户端提供的 HTTP/2 连接中最大流数的限制。 零表示使用 golang 的默认值。
--identity-lease-duration-seconds int 默认值：3600
kube-apiserver 租约时长（按秒计），必须是正数。（当 APIServerIdentity 特性门控被启用时使用此标志值）
--identity-lease-renew-interval-seconds int 默认值：10
kube-apiserver 对其租约进行续期的时间间隔（按秒计），必须是正数。（当 APIServerIdentity 特性门控被启用时使用此标志值）
--kubelet-certificate-authority string
证书颁发机构的证书文件的路径。
--kubelet-client-certificate string
TLS 的客户端证书文件的路径。
--kubelet-client-key string
TLS 客户端密钥文件的路径。
--kubelet-preferred-address-types stringSlice 默认值：[Hostname,InternalDNS,InternalIP,ExternalDNS,ExternalIP]
用于 kubelet 连接的首选 NodeAddressTypes 列表。
--kubelet-timeout duration 默认值：5s
kubelet 操作超时时间。
--kubernetes-service-node-port int
如果非零，那么 Kubernetes 主服务（由 apiserver 创建/维护）将是 NodePort 类型，使用它作为端口的值。 如果为零，则 Kubernetes 主服务将为 ClusterIP 类型。
--livez-grace-period duration
此选项代表 apiserver 完成启动序列并生效所需的最长时间。 从 apiserver 的启动时间到这段时间为止， /livez 将假定未完成的启动后钩子将成功完成，因此返回 true。
当日志机制执行到'文件 :N'时，生成堆栈跟踪
--log-dir string
如果为非空，则在此目录中写入日志文件
--log-file string
如果为非空，使用此日志文件
--log-file-max-size uint 默认值：1800

	定义日志文件可以增长到的最大大小。单位为兆字节。 如果值为 0，则最大文件大小为无限制。
--log-flush-frequency duration　默认值：5s	
	两次日志刷新之间的最大秒数
--logging-format string　默认值："text"	
	设置日志格式。允许的格式："json"，"json"。 非默认格式不支持以下标志：--add_dir_header、--alsologtostderr、--log_backtrace_at、--log_dir、--log_file、--log_file_max_size、--logtostderr、--one_output、-skip_headers、-skip_log_headers、--stderrthreshold、-vmodule 和 --log-flush-frequency。 当前非默认选择为 alpha，会随时更改而不会发出警告。
c --logtostderr　默认值：true	
	在标准错误而不是文件中输出日志记录
--master-service-namespace string　默认值："default"	
	已废弃：应该从其中将 Kubernetes 主服务注入到 Pod 中的名字空间。
--max-connection-bytes-per-sec int	
	如果不为零，则将每个用户连接限制为该数（字节数/秒）。 当前仅适用于长时间运行的请求。
--max-mutating-requests-inflight int　默认值：200	
	在给定时间内进行中变更类型请求的最大个数。 当超过该值时，服务将拒绝所有请求。零表示无限制。
--max-requests-inflight int　默认值：400	
	在给定时间内进行中非变更类型请求的最大数量。 当超过该值时，服务将拒绝所有请求。 零表示无限制。
--min-request-timeout int　默认值：1800	
	可选字段，表示处理程序在请求超时前，必须保持其处于打开状态的最小秒数。 当前只对监听（Watch）请求的处理程序有效，它基于这个值选择一个随机数作为连接超时值，以达到分散负载的目的。
--oidc-ca-file string	
	如果设置该值，将会使用 oidc-ca-file 中的机构之一对 OpenID 服务的证书进行验证， 否则将会使用主机的根 CA 对其进行验证。
--oidc-client-id string	
	OpenID 连接客户端的要使用的客户 ID，如果设置了 oidc-issuer-url，则必须设置这个值。
--oidc-groups-claim string	
	如果提供该值，这个自定义 OpenID 连接声明将被用来设定用户组。 该声明值需要是一个字符串或字符串数组。 此标志为实验性的，请查阅身份认证相关文档进一步了解详细信息。
--oidc-groups-prefix string	
	如果提供，则所有组都将以该值作为前缀，以防止与其他身份认证策略冲突。
--oidc-issuer-url string	

	OpenID 颁发者 URL，只接受 HTTPS 方案。 如果设置该值，它将被用于验证 OIDC JSON Web Token(JWT)。
--oidc-required-claim mapStringString	
	描述 ID 令牌中必需声明的键值对。 如果设置此值，则会验证 ID 令牌中存在与该声明匹配的值。 重复此标志以指定多个声明。
--oidc-signing-algs stringSlice 默认值：[RS256]	
	允许的 JOSE 非对称签名算法的逗号分隔列表。 若 JWT 所带的 "alg" 标头值不在列表中，则该 JWT 将被拒绝。 取值依据 RFC 7518 https://tools.ietf.org/html/rfc7518#section-3.1 定义。
--oidc-username-claim string 默认值："sub"	
	要用作用户名的 OpenID 声明。 请注意，除默认声明（"sub"）以外的其他声明不能保证是唯一且不可变的。 此标志是实验性的，请参阅身份认证文档以获取更多详细信息。
--oidc-username-prefix string	
	如果提供，则所有用户名都将以该值作为前缀。 如果未提供，则除 "email" 之外的用户名声明都会添加颁发者 URL 作为前缀，以避免冲突。 要略过添加前缀处理，请设置值为 "-"。
--one-output	
	此标志为真时，日志只会被写入到其原生的严重性级别中（而不是同时写到所有较低 严重性级别中）。
--permit-port-sharing	
	如果为 true，则在绑定端口时将使用 SO_REUSEPORT， 这样多个实例可以绑定到同一地址和端口上。[默认值 = false]
--profiling 默认值：true	
	通过 Web 界面启用性能分析 host:port/debug/pprof/
--proxy-client-cert-file string	
	当必须调用外部程序以处理请求时，用于证明聚合器或者 kube-apiserver 的身份的客户端证书。 包括代理转发到用户 api-server 的请求和调用 Webhook 准入控制插件的请求。 Kubernetes 期望此证书包含来自于 --requestheader-client-ca-file 标志中所给 CA 的签名。 该 CA 在 kube-system 命名空间的 "extension-apiserver-authentication" ConfigMap 中公开。 从 kube-aggregator 收到调用的组件应该使用该 CA 进行各自的双向 TLS 验证。
--proxy-client-key-file string	
	当必须调用外部程序来处理请求时，用来证明聚合器或者 kube-apiserver 的身份的客户端私钥。 这包括代理转发给用户 api-server 的请求和调用 Webhook 准入控制插件的请求。
--request-timeout duration 默认值：1m0s	
	可选字段，指示处理程序在超时之前必须保持打开请求的持续时间。 这是请求的默认请求超时，但对于特定类型的请求，可能会被 `--min-request-timeout`等标志覆盖。
--requestheader-allowed-names stringSlice	

此值为客户端证书通用名称（Common Name）的列表；表中所列的表项可以用来提供用户名， 方式是使用 --requestheader-username-headers 所指定的头部。 如果为空，能够通过 --requestheader-client-ca-file 中机构认证的客户端证书都是被允许的。
--requestheader-client-ca-file string
在信任请求头中以 --requestheader-username-headers 指示的用户名之前， 用于验证接入请求中客户端证书的根证书包。 警告：一般不要假定传入请求已被授权。
--requestheader-extra-headers-prefix stringSlice
用于查验请求头部的前缀列表。建议使用 X-Remote-Extra-。
--requestheader-group-headers stringSlice
用于查验用户组的请求头部列表。建议使用 X-Remote-Group。
--requestheader-username-headers stringSlice
用于查验用户名的请求头头列表。建议使用 X-Remote-User。
--runtime-config mapStringString
一组启用或禁用内置 API 的键值对。支持的选项包括： v1=true|false（针对核心 API 组） <group>/<version>=true|false（针对特定 API 组和版本，例如：apps/v1=true） api/all=true|false 控制所有 API 版本 api/ga=true|false 控制所有 v[0-9]+ API 版本 api/beta=true|false 控制所有 v[0-9]+beta[0-9]+ API 版本 api/alpha=true|false 控制所有 v[0-9]+alpha[0-9]+ API 版本 api/legacy 已弃用，并将在以后的版本中删除
--secure-port int　默认值：6443
带身份验证和鉴权机制的 HTTPS 服务端口。 不能用 0 关闭。
--service-account-extend-token-expiration　默认值：true
在生成令牌时，启用投射服务帐户到期时间扩展， 这有助于从旧版令牌安全地过渡到绑定的服务帐户令牌功能。 如果启用此标志，则准入插件注入的令牌的过期时间将延长至 1 年，以防止过渡期间发生意外故障， 并忽略 service-account-max-token-expiration 的值。
--service-account-issuer string
服务帐号令牌颁发者的标识符。 颁发者将在已办法令牌的 "iss" 声明中检查此标识符。 此值为字符串或 URI。 如果根据 OpenID Discovery 1.0 规范检查此选项不是有效的 URI，则即使特性门控设置为 true， ServiceAccountIssuerDiscovery 功能也将保持禁用状态。 强烈建议该值符合 OpenID 规范：https://openid.net/specs/openid-connect-discovery-1_0.html。 实践中，这意味着 service-account-issuer 取值必须是 HTTPS URL。 还强烈建议此 URL 能够在 {service-account-issuer}/.well-known/openid-configuration 处提供 OpenID 发现文档。
--service-account-jwks-uri string
覆盖 /.well-known/openid-configuration 提供的发现文档中 JSON Web 密钥集的 URI。 如果发现文档和密钥集是通过 API 服务器外部（而非自动检测到或被外部主机名覆盖）之外的 URL 提供给依赖方的，则此标志很有用。 仅在启用 ServiceAccountIssuerDiscovery 特性门控的情况下有效。

--service-account-key-file stringArray
包含 PEM 编码的 x509 RSA 或 ECDSA 私钥或公钥的文件，用于验证 ServiceAccount 令牌。 指定的文件可以包含多个键，并且可以使用不同的文件多次指定标志。 如果未指定，则使用 --tls-private-key-file。 提供 --service-account-signing-key 时必须指定。
--service-account-lookup 默认值：true
如果为 true，则在身份认证时验证 etcd 中是否存在 ServiceAccount 令牌。
--service-account-max-token-expiration duration
服务帐户令牌发布者创建的令牌的最长有效期。 如果请求有效期大于此值的有效令牌请求，将使用此值的有效期颁发令牌。
--service-account-signing-key-file string
包含服务帐户令牌颁发者当前私钥的文件的路径。 颁发者将使用此私钥签署所颁发的 ID 令牌。
--service-cluster-ip-range string
CIDR 表示的 IP 范围用来为服务分配集群 IP。 此地址不得与指定给节点或 Pod 的任何 IP 范围重叠。
--service-node-port-range portRange 默认值：30000-32767
保留给具有 NodePort 可见性的服务的端口范围。 例如："30000-32767"。范围的两端都包括在内。
--show-hidden-metrics-for-version string
你要显示隐藏指标的先前版本。仅先前的次要版本有意义，不允许其他值。 格式为 <major>.<minor>，例如："1.16"。 这种格式的目的是确保您有机会注意到下一个版本是否隐藏了其他指标， 而不是在此之后将它们从发行版中永久删除时感到惊讶。
--shutdown-delay-duration duration
延迟终止时间。在此期间，服务器将继续正常处理请求。 端点 /healthz 和 /livez 将返回成功，但是 /readyz 立即返回失败。 在此延迟过去之后，将开始正常终止。 这可用于允许负载平衡器停止向该服务器发送流量。
--skip-headers
如果为 true，日志消息中避免标题前缀
--skip-log-headers
如果为 true，则在打开日志文件时避免标题
--stderrthreshold severity 默认值：2
将达到或超过此阈值的日志写到标准错误输出
--storage-backend string
持久化存储后端。选项："etcd3"（默认）。
--storage-media-type string 默认值："application/vnd.kubernetes.protobuf"
用于在存储中存储对象的媒体类型。 某些资源或存储后端可能仅支持特定的媒体类型，并且将忽略此设置。
--tls-cert-file string

	包含用于 HTTPS 的默认 x509 证书的文件。（CA 证书（如果有）在服务器证书之后并置）。 如果启用了 HTTPS 服务，并且未提供 --tls-cert-file 和 --tls-private-key-file， 为公共地址生成一个自签名证书和密钥，并将其保存到 --cert-dir 指定的目录中。
--tls-cipher-suites stringSlice	
	服务器的密码套件的列表，以逗号分隔。如果省略，将使用默认的 Go 密码套件。 首选值：TLS_AES_128_GCM_SHA256、TLS_AES_256_GCM_SHA384、 TLS_CHACHA20_POLY1305_SHA256、 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA、 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256、 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA、 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384、 TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305、 TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256、 TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA、 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA、 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256、 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA、 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384、 TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305、 TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256、 TLS_RSA_WITH_3DES_EDE_CBC_SHA、 TLS_RSA_WITH_AES_128_CBC_SHA、 TLS_RSA_WITH_AES_128_GCM_SHA256、 TLS_RSA_WTLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256、 TLS_ECDHE_ECDSA_WITH_RC4_128_SHA、 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256、 TLS_ECDHE_RSA_WITH_RC4_128_SHA、 TLS_RSA_WITH_AES_128_CBC_SHA256、 TLS_RSA_WITH_RC4_128_SHA。
--tls-min-version string	
	支持的最低 TLS 版本。可能的值：VersionTLS10，VersionTLS11， VersionTLS12，VersionTLS13
--tls-private-key-file string	
	包含匹配 --tls-cert-file 的 x509 证书私钥的文件。
--tls-sni-cert-key namedCertKey　默认值：[]	
	一对 x509 证书和私钥文件路径，（可选）后缀为全限定域名的域名模式列表，可以使用带有通配符的前缀。 域模式也允许使用 IP 地址，但仅当 apiserver 对客户端请求的IP地址具有可见性时，才应使用 IP。 如果未提供域模式，则提取证书的名称。 非通配符匹配优先于通配符匹配，显式域模式优先于提取出的名称。 对于多个密钥/证书对，请多次使用 --tls-sni-cert-key。 示例："example.crt,example.key" 或 "foo.crt,foo.key:*.foo.com,foo.com"。
--token-auth-file string	
	如果设置该值，这个文件将被用于通过令牌认证来保护 API 服务的安全端口。

-v, --v Level	
	日志级别详细程度的数字
--version version[=true]	
	打印版本信息并退出
--vmodule moduleSpec	
	以逗号分隔的 pattern=N 设置列表，用于文件过滤的日志记录
--watch-cache 默认值：true	
	在 apiserver 中启用监视缓存
--watch-cache-sizes stringSlice	
	某些资源（pods、nodes 等）的监视缓存大小设置，以逗号分隔。 每个资源对应的设置格式：resource[.group]#size，其中 resource 为小写复数（无版本）， 对于 apiVersion v1（旧版核心 API）的资源要省略 group， 对其它资源要给出 group，size 为一个数字。 启用 watch-cache 时，此功能生效。 某些资源（replicationcontrollers、endpoints、nodes、pods、services、apiservices.apiregistration.k8s.io）具有通过启发式设置的系统默认值，其他资源默认为 default-watch-cache-size

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 18, 2021 at 9:15 PM PST: [zh] Resync kube-apiserver reference (ee81eeccf)

kube-controller-manager

简介

Kubernetes 控制器管理器是一个守护进程，内嵌随 Kubernetes 一起发布的核心控制回路。 在机器人和自动化的应用中，控制回路是一个永不休止的循环，用于调节系统状态。 在 Kubernetes 中，每个控制器是一个控制回路，通过 API 服务器监视集群的共享状态， 并尝试进行更改以将当前状态转为期望状态。 目前，Kubernetes 自带的控制器例子包括副本控制器、节点控制器、命名空间控制器和服务账号控制器等。

```
kube-controller-manager [flags]
```

选项

--add-dir-header

	若为 true，将文件目录添加到头部。
--allocate-node-cidrs	
	基于云驱动来为 Pod 分配和设置子网掩码。
--alsologtostderr	
	在向文件输出日志的同时，也将日志写到标准输出。
--attach-detach-reconcile-sync-period duration　默认值：1m0s	
	协调器（reconciler）在相邻两次对存储卷进行挂载和解除挂载操作之间的等待时间。此时长必须长于 1 秒钟。此值设置为大于默认值时，可能导致存储卷无法与 Pods 匹配。
--authentication-kubeconfig string	
	此标志值为一个 kubeconfig 文件的路径名。该文件中包含与某 Kubernetes "核心" 服务器相关的信息，并支持足够的权限以创建 tokenreviews.authentication.k8s.io。 此选项是可选的。如果设置为空值，所有令牌请求都会被认作匿名请求， Kubernetes 也不再在集群中查找客户端的 CA 证书信息。
--authentication-skip-lookup	
	此值为 false 时，通过 authentication-kubeconfig 参数所指定的文件会被用来 检索集群中缺失的身份认证配置信息。
--authentication-token-webhook-cache-ttl duration　默认值：10s	
	对 Webhook 令牌认证设施返回结果的缓存时长。
--authentication-tolerate-lookup-failure	
	此值为 true 时，即使无法从集群中检索到缺失的身份认证配置信息也无大碍。 需要注意的是，这样设置可能导致所有请求都被视作匿名请求。
--authorization-always-allow-paths stringSlice　默认值：[/healthz]	
	鉴权过程中会忽略的一个 HTTP 路径列表。 换言之，控制器管理器会对列表中路径的访问进行授权，并且无须征得 Kubernetes "核心" 服务器同意。
--authorization-kubeconfig string	
	包含 Kubernetes "核心" 服务器信息的 kubeconfig 文件路径， 所包含信息具有创建 subjectaccessreviews.authorization.k8s.io 的足够权限。 此参数是可选的。如果配置为空字符串，未被鉴权模块所忽略的请求都会被禁止。
--authorization-webhook-cache-authorized-ttl duration　默认值：10s	
	对 Webhook 形式鉴权组件所返回的"已授权（Authorized）"响应的缓存时长。
--authorization-webhook-cache-unauthorized-ttl duration　默认值：10s	
	对 Webhook 形式鉴权组件所返回的"未授权（Unauthorized）"响应的缓存时长。
--azure-container-registry-config string	
	指向包含 Azure 容器仓库配置信息的文件的路径名。
--bind-address ip　默认值：0.0.0.0	
	针对 --secure-port 端口上请求执行监听操作的 IP 地址。 所对应的网络接口必须从集群中其它位置可访问（含命令行及 Web 客户端）。 如果此值为空或者设定为非特定地址（0.0.0.0 或 ::），意味着所有网络接口都在监听范围。
--cert-dir string	

	TLS 证书所在的目录。如果提供了 --tls-cert-file 和 --tls-private-key-file，此标志会被忽略。
--cidr-allocator-type string　默认值："RangeAllocator"	
	要使用的 CIDR 分配器类型。
--client-ca-file string	
	如果设置了此标志，对于所有能够提供客户端证书的请求，若该证书由 client-ca-file 中所给机构之一签署，则该请求会被成功认证为客户端证书中 CommonName 所给的实体。
--cloud-config string	
	云驱动程序配置文件的路径。空字符串表示没有配置文件。
--cloud-provider string	
	云服务的提供者。空字符串表示没有对应的提供者（驱动）。
--cluster-cidr string	
	集群中 Pods 的 CIDR 范围。要求 --allocate-node-cidrs 标志为 true。
--cluster-name string　默认值："kubernetes"	
	集群实例的前缀。
--cluster-signing-cert-file string　默认值："/etc/kubernetes/ca/ca.pem"	
	包含 PEM 编码格式的 X509 CA 证书的文件名。该证书用来发放集群范围的证书。如果设置了此标志，则不需要锦衣设置 --cluster-signing-* 标志。
--cluster-signing-duration duration　默认值：8760h0m0s	
	所签名证书的有效期限。
--cluster-signing-key-file string　默认值："/etc/kubernetes/ca/ca.key"	
	包含 PEM 编码的 RSA 或 ECDSA 私钥的文件名。该私钥用来对集群范围证书签名。
--cluster-signing-kube-apiserver-client-cert-file string	
	包含 PEM 编码的 X509 CA 证书的文件名， 该证书用于为 kubernetes.io/kube-apiserver-client 签署者颁发证书。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--cluster-signing-kube-apiserver-client-key-file string	
	包含 PEM 编码的 RSA 或 ECDSA 私钥的文件名， 该私钥用于为 kubernetes.io/kube-apiserver-client 签署者签名证书。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--cluster-signing-kubelet-client-cert-file string	
	包含 PEM 编码的 X509 CA 证书的文件名， 该证书用于为 kubernetes.io/kube-apiserver-client-kubelet 签署者颁发证书。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--cluster-signing-kubelet-client-key-file string	
	包含 PEM 编码的 RSA 或 ECDSA 私钥的文件名， 该私钥用于为 kubernetes.io/kube-apiserver-client-kubelet 签署者签名证书。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--cluster-signing-kubelet-serving-cert-file string	

	包含 PEM 编码的 X509 CA 证书的文件名， 该证书用于为 kubernetes.io/kubelet-serving 签署者颁发证书。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--cluster-signing-kubelet-serving-key-file string	
	包含 PEM 编码的 RSA或ECDSA 私钥的文件名， 该私钥用于对 kubernetes.io/kubelet-serving 签署者的证书进行签名。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--cluster-signing-legacy-unknown-cert-file string	
	包含 PEM 编码的 X509 CA 证书的文件名， 用于为 kubernetes.io/legacy-unknown 签署者颁发证书。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--cluster-signing-legacy-unknown-key-file string	
	包含 PEM 编码的 RSA 或 ECDSA 私钥的文件名， 用于为 kubernetes.io/legacy-unknown 签署者签名证书。 如果指定，则不得设置 --cluster-signing-{cert,key}-file。
--concurrent-deployment-syncs int32　默认值：5	
	可以并发同步的 Deployment 对象个数。数值越大意味着对 Deployment 的响应越及时， 同时也意味着更大的 CPU（和网络带宽）压力。
--concurrent-endpoint-syncs int32　默认值：5	
	可以并发执行的 Endpoints 同步操作个数。数值越大意味着更快的 Endpoints 更新操作， 同时也意味着更大的 CPU （和网络）压力。
--concurrent-gc-syncs int32　默认值：20	
	可以并发同步的垃圾收集工作线程个数。
--concurrent-namespace-syncs int32　默认值：10	
	可以并发同步的 Namespace 对象个数。较大的数值意味着更快的名字空间终结操作， 不过也意味着更多的 CPU （和网络）占用。
--concurrent-replicaset-syncs int32　默认值：5	
	可以并发同步的 ReplicaSet 个数。数值越大意味着副本管理的响应速度越快， 同时也意味着更多的 CPU （和网络）占用。
--concurrent-resource-quota-syncs int32　默认值：5	
	可以并发同步的 ResourceQuota 对象个数。数值越大，配额管理的响应速度越快，不过对 CPU （和网络）的占用也越高。
--concurrent-service-endpoint-syncs int32　默认值：5	
	可以并发执行的服务端点同步操作个数。数值越大，端点片段（Endpoint Slice） 的更新速度越快，不过对 CPU （和网络）的占用也越高。默认值为 5。
--concurrent-service-syncs int32　默认值：1	
	可以并发同步的 Service 对象个数。数值越大，服务管理的响应速度越快， 不过对 CPU （和网络）的占用也越高。
--concurrent-serviceaccount-token-syncs int32　默认值：5	
	可以并发同步的服务账号令牌对象个数。数值越大，令牌生成的速度越快， 不过对 CPU （和网络）的占用也越高。
--concurrent-statefulset-syncs int32　默认值：5	

可以并发同步的 StatefulSet 对象个数。数值越大，StatefulSet 管理的响应速度越快， 不过对 CPU （和网络）的占用也越高。
--concurrent-ttl-after-finished-syncs int32　默认值：5
可以并发同步的 TTL-after-finished 控制器线程个数。
--concurrent_rc_syncs int32　默认值：5
可以并发同步的 ReplicationController 对象个数。数值越大，副本管理的响应速度越快， 不过对 CPU （和网络）的占用也越高。
--configure-cloud-routes　默认值：true
决定是否由 --allocate-node-cidrs 所分配的 CIDR 要通过云驱动程序来配置。
--contention-profiling
在启用了性能分析（profiling）时，也启用锁竞争情况分析。
--controller-start-interval duration
在两次启动控制器管理器之间的时间间隔。
--controllers stringSlice　默认值：[*]
要启用的控制器列表。* 表示启用所有默认启用的控制器；foo 启用名为 foo 的控制器； -foo 表示禁用名为 foo 的控制器。 控制器的全集：attachdetach、bootstrapsigner、cloud-node-lifecycle、clusterrole-aggregation、cronjob、csrapproving、csrcleaner、csrsigning、daemonset、deployment、disruption、endpoint、endpointslice、endpointslicemirroring、ephemeral-volume、garbagecollector、horizontalpodautoscaling、job、namespace、nodeipam、nodelifecycle、persistentvolume-binder、persistentvolume-expander、podgc、pv-protection、pvc-protection、replicaset、replicationcontroller、resourcequota、root-ca-cert-publisher、route、service、serviceaccount、serviceaccount-token、statefulset、tokencleaner、ttl、ttl-after-finished 默认禁用的控制器有：bootstrapsigner 和 tokencleaner。
--deployment-controller-sync-period duration　默认值：30s
Deployment 资源的同步周期。
--disable-attach-detach-reconcile-sync
禁用卷挂接/解挂调节器的同步。禁用此同步可能导致卷存储与 Pod 之间出现错位。请小心使用。
--enable-dynamic-provisioning　默认值：true
在环境允许的情况下启用动态卷制备。
--enable-garbage-collector　默认值：true
启用通用垃圾收集器。必须与 kube-apiserver 中对应的标志一致。
--enable-hostpath-provisioner
在没有云驱动程序的情况下，启用 HostPath 持久卷的制备。 此参数便于对卷供应功能进行开发和测试。HostPath 卷的制备并非受支持的功能特性， 在多节点的集群中也无法工作，因此除了开发和测试环境中不应使用。
--enable-taint-manager　默认值：true
警告：此为Beta 阶段特性。设置为 true 时会启用 NoExecute 污点， 并在所有标记了此污点的节点上逐出所有无法忍受该污点的 Pods。

--endpoint-updates-batch-period duration	
	端点（Endpoint）批量更新周期时长。对 Pods 变更的处理会被延迟， 以便将其与即将到来的更新操作合并，从而减少端点更新操作次数。 较大的数值意味着端点更新的迟滞时间会增长，也意味着所生成的端点版本个数会变少。
--endpointslice-updates-batch-period duration	
	端点片段（Endpoint Slice）批量更新周期时长。对 Pods 变更的处理会被延迟， 以便将其与即将到来的更新操作合并，从而减少端点更新操作次数。 较大的数值意味着端点更新的迟滞时间会增长，也意味着所生成的端点版本个数会变少。
--experimental-logging-sanitization	
	[试验性功能] 当启用此标志时，被标记为敏感的字段（密码、密钥、令牌）不会被日志输出。 运行时的日志清理操作可能会引入相当程度的计算开销，因此不应在生产环境中启用。
--external-cloud-volume-plugin string	
	当云驱动程序设置为 external 时要使用的插件名称。此字符串可以为空。 只能在云驱动程序为 external 时设置。目前用来保证节点控制器和卷控制器能够 在三种云驱动上正常工作。
--feature-gates mapStringBool	

一组 key=value 对，用来描述测试性/试验性功能的特性门控（Feature Gate）。可选项有：
APIListChunking=true|false (BETA - 默认值=true)
APIPriorityAndFairness=true|false (BETA - 默认值=true)
APIResponseCompression=true|false (BETA - 默认值=true)
APIServerIdentity=true|false (ALPHA - 默认值=false)
AllAlpha=true|false (ALPHA - 默认值=false)
AllBeta=true|false (BETA - 默认值=false)
AllowInsecureBackendProxy=true|false (BETA - 默认值=true)
AnyVolumeDataSource=true|false (ALPHA - 默认值=false)
AppArmor=true|false (BETA - 默认值=true)
BalanceAttachedNodeVolumes=true|false (ALPHA - 默认值=false)
BoundServiceAccountTokenVolume=true|false (ALPHA - 默认值=false)
CPUManager=true|false (BETA - 默认值=true)
CRIContainerLogRotation=true|false (BETA - 默认值=true)
CSIInlineVolume=true|false (BETA - 默认值=true)
CSIMigration=true|false (BETA - 默认值=true)
CSIMigrationAWS=true|false (BETA - 默认值=false)
CSIMigrationAWSComplete=true|false (ALPHA - 默认值=false)
CSIMigrationAzureDisk=true|false (BETA - 默认值=false)
CSIMigrationAzureDiskComplete=true|false (ALPHA - 默认值=false)
CSIMigrationAzureFile=true|false (ALPHA - 默认值=false)
CSIMigrationAzureFileComplete=true|false (ALPHA - 默认值=false)
CSIMigrationGCE=true|false (BETA - 默认值=false)
CSIMigrationGCEComplete=true|false (ALPHA - 默认值=false)
CSIMigrationOpenStack=true|false (BETA - 默认值=false)
CSIMigrationOpenStackComplete=true|false (ALPHA - 默认值=false)
CSIMigrationvSphere=true|false (BETA - 默认值=false)
CSIMigrationvSphereComplete=true|false (BETA - 默认值=false)
CSIServiceAccountToken=true|false (ALPHA - 默认值=false)
CSIStorageCapacity=true|false (ALPHA - 默认值=false)
CSIVolumeFSGroupPolicy=true|false (BETA - 默认值=true)
ConfigurableFSGroupPolicy=true|false (BETA - 默认值=true)
CronJobControllerV2=true|false (ALPHA - 默认值=false)
CustomCPUCFSQuotaPeriod=true|false (ALPHA - 默认值=false)
DefaultPodTopologySpread=true|false (BETA - 默认值=true)
DevicePlugins=true|false (BETA - 默认值=true)
DisableAcceleratorUsageMetrics=true|false (BETA - 默认值=true)
DownwardAPIHugePages=true|false (ALPHA - 默认值=false)
DynamicKubeletConfig=true|false (BETA - 默认值=true)
EfficientWatchResumption=true|false (ALPHA - 默认值=false)
EndpointSlice=true|false (BETA - 默认值=true)
EndpointSliceNodeName=true|false (ALPHA - 默认值=false)
EndpointSliceProxying=true|false (BETA - 默认值=true)
EndpointSliceTerminatingCondition=true|false (ALPHA - 默认值=false)
EphemeralContainers=true|false (ALPHA - 默认值=false)
ExpandCSIVolumes=true|false (BETA - 默认值=true)
ExpandInUsePersistentVolumes=true|false (BETA - 默认值=true)

--flex-volume-plugin-dir string　默认值："/usr/libexec/kubernetes/kubelet-plugins/volume/exec/"	
	FlexVolume 插件要搜索第三方卷插件的目录路径。
-h, --help	
	kube-controller-manager 的帮助信息。
--horizontal-pod-autoscaler-cpu-initialization-period duration　默认值：5m0s	
	Pod 启动之后可以忽略 CPU 采样值的时长。
--horizontal-pod-autoscaler-downscale-stabilization duration　默认值：5m0s	
	自动扩缩程序的回溯时长。自动扩缩器不会基于在给定的时长内所建议的规模对负载执行规模缩小的操作。
--horizontal-pod-autoscaler-initial-readiness-delay duration　默认值：30s	
	Pod 启动之后，在此值所给定的时长内，就绪状态的变化都不会作为初始的就绪状态。
--horizontal-pod-autoscaler-sync-period duration　默认值：15s	
	水平 Pod 扩缩器对 Pods 数目执行同步操作的周期。
--horizontal-pod-autoscaler-tolerance float　默认值：0.1	
	此值为目标值与实际值的比值与 1.0 的差值。只有超过此标志所设的阈值时，HPA 才会考虑执行缩放操作。
--http2-max-streams-per-connection int	
	服务器为客户端所设置的 HTTP/2 连接中流式连接个数上限。 此值为 0 表示采用 Go 语言库所设置的默认值。
--kube-api-burst int32　默认值：30	
	与 Kubernetes API 服务器通信时突发峰值请求个数上限。
--kube-api-content-type string　默认值："application/vnd.kubernetes.protobuf"	
	向 API 服务器发送请求时使用的内容类型（Content-Type）。
--kube-api-qps float32　默认值：20	
	与 API 服务器通信时每秒请求数（QPS）限制。
--kubeconfig string	
	指向 kubeconfig 文件的路径。该文件中包含主控节点位置以及鉴权凭据信息。
--large-cluster-size-threshold int32　默认值：50	
	节点控制器在执行 Pod 逐出操作逻辑时，基于此标志所设置的节点个数阈值来判断 所在集群是否为大规模集群。当集群规模小于等于此规模时， `--secondary-node-eviction-rate` 会被隐式重设为 0。
--leader-elect　默认值：true	
	在执行主循环之前，启动领导选举（Leader Election）客户端，并尝试获得领导者身份。 在运行多副本组件时启用此标志有助于提高可用性。
--leader-elect-lease-duration duration　默认值：15s	
	对于未获得领导者身份的节点，在探测到领导者身份需要更迭时需要等待 此标志所设置的时长，才能尝试去获得曾经是领导者但尚未续约的席位。 本质上，这个时长也是现有领导者节点在被其他候选节点替代之前可以停止 的最长时长。只有集群启用了领导者选举机制时，此标志才起作用。

--leader-elect-renew-deadline duration　默认值：10s
当前执行领导者角色的节点在被停止履行领导职责之前可多次尝试续约领导者身份； 此标志给出相邻两次尝试之间的间歇时长。 此值必须小于或等于租期时长（Lease Duration）。仅在集群启用了领导者选举时有效。
--leader-elect-resource-lock string　默认值："endpointsleases"
在领导者选举期间用于锁定的资源对象的类型。 支持的选项为 "endpoints"、"configmaps"、"leases"、"endpointsleases" 和 "configmapsleases"。
--leader-elect-resource-name string　默认值："kube-controller-manager"
在领导者选举期间，用来执行锁操作的资源对象名称。
--leader-elect-resource-namespace string　默认值："kube-system"
在领导者选举期间，用来执行锁操作的资源对象的名字空间。
--leader-elect-retry-period duration　默认值：2s
尝试获得领导者身份时，客户端在相邻两次尝试之间要等待的时长。此标志仅在启用了领导者选举的集群中起作用。
--log-backtrace-at traceLocation　默认值：:0
当执行到 file:N 所给的文件和代码行时，日志机制会生成一个调用栈快照。
--log-dir string
此标志为非空字符串时，日志文件会写入到所给的目录中。
--log-file string
此标志为非空字符串时，意味着日志会写入到所给的文件中。
--log-file-max-size uint　默认值：1800
定义日志文件大小的上限。单位是兆字节（MB）。 若此值为 0，则不对日志文件尺寸进行约束。
--log-flush-frequency duration　默认值：5s
将内存中日志数据清除到日志文件中时，相邻两次清除操作之间最大间隔秒数。
--logging-format string　默认值："text"
设置日志格式。允许的格式："text"，"json"。 非默认格式不支持以下标志：--add_dir_header、--alsologtostderr、--log_backtrace_at、--log_dir、--log_file、--log_file_max_size、--logtostderr、--one_output、--skip_headers、--skip_log_headers、--stderrthreshold、--vmodule、--log-flush-frequency。 当前非默认选项为 Alpha，如有更改，恕不另行通知。
--logtostderr　默认值：true
将日志写出到标准错误输出（stderr）而不是写入到日志文件。
--master string
Kubernetes API 服务器的地址。此值会覆盖 kubeconfig 文件中所给的地址。
--max-endpoints-per-slice int32　默认值：100
每个 EndpointSlice 中可以添加的端点个数上限。每个片段中端点个数越多， 得到的片段个数越少，但是片段的规模会变得更大。默认值为 100。
--min-resync-period duration　默认值：12h0m0s
自省程序的重新同步时隔下限。实际时隔长度会在 min-resync-period 和 2 * min-resync-period 之间。

--mirroring-concurrent-service-endpoint-syncs int32　默认值：5
EndpointSliceMirroring 控制器将同时执行的服务端点同步操作数。 较大的数量 = 更快的端点切片更新，但 CPU（和网络）负载更多。 默认为 5。
--mirroring-endpointslice-updates-batch-period duration
EndpointSlice 的长度更新了 EndpointSliceMirroring 控制器的批处理周期。 EndpointSlice 更改的处理将延迟此持续时间， 以使它们与潜在的即将进行的更新结合在一起，并减少 EndpointSlice 更新的总数。 较大的数量 = 较高的端点编程延迟，但是生成的端点修订版本数量较少
--mirroring-max-endpoints-per-subset int32　默认值：1000
EndpointSliceMirroring 控制器将添加到 EndpointSlice 的最大端点数。 每个分片的端点越多，端点分片越少，但资源越大。 默认为 100。
--namespace-sync-period duration　默认值：5m0s
对名字空间对象进行同步的周期。
--node-cidr-mask-size int32
集群中节点 CIDR 的掩码长度。对 IPv4 而言默认为 24；对 IPv6 而言默认为 64。
--node-cidr-mask-size-ipv4 int32
在双堆栈（同时支持 IPv4 和 IPv6）的集群中，节点 IPV4 CIDR 掩码长度。默认为 24。
--node-cidr-mask-size-ipv6 int32
在双堆栈（同时支持 IPv4 和 IPv6）的集群中，节点 IPv6 CIDR 掩码长度。默认为 64。
--node-eviction-rate float32　默认值：0.1
当某区域变得不健康，节点失效时，每秒钟可以从此标志所设定的节点 个数上删除 Pods。请参阅 --unhealthy-zone-threshold 以了解"健康"的判定标准。这里的区域（zone）在集群并不跨多个区域时 指的是整个集群。
--node-monitor-grace-period duration　默认值：40s
在将一个 Node 标记为不健康之前允许其无响应的时长上限。 必须比 kubelet 的 nodeStatusUpdateFrequency 大 N 倍； 这里 N 指的是 kubelet 发送节点状态的重试次数。
--node-monitor-period duration　默认值：5s
节点控制器对节点状态进行同步的重复周期。
--node-startup-grace-period duration　默认值：1m0s
在节点启动期间，节点可以处于无响应状态； 但超出此标志所设置的时长仍然无响应则该节点被标记为不健康。
--one-output
如果此标志为 true，则仅将日志写入其自身的严重性级别（而不是同时写入更低的严重性级别中）。
--permit-port-sharing
如果为 true，则在绑定端口时将使用 SO_REUSEPORT， 这允许多个实例在同一地址和端口上进行绑定。 [默认值 = false]
--pod-eviction-timeout duration　默认值：5m0s
在失效的节点上删除 Pods 时为其预留的宽限期。

--profiling 默认值：true
通过位于 host:port/debug/pprof/ 的 Web 接口启用性能分析。
--pv-recycler-increment-timeout-nfs int32 默认值：30
NFS 清洗 Pod 在清洗用过的卷时，根据此标志所设置的秒数，为每清洗 1 GiB 数据增加对应超时时长，作为 activeDeadlineSeconds。
--pv-recycler-minimum-timeout-hostpath int32 默认值：60
对于 HostPath 回收器 Pod，设置其 activeDeadlineSeconds 参数下限。 此参数仅用于开发和测试目的，不适合在多节点集群中使用。
--pv-recycler-minimum-timeout-nfs int32 默认值：300
NFS 回收器 Pod 要使用的 activeDeadlineSeconds 参数下限。
--pv-recycler-pod-template-filepath-hostpath string
对 HostPath 持久卷进行回收利用时，用作模版的 Pod 定义文件所在路径。 此标志仅用于开发和测试目的，不适合多节点集群中使用。
--pv-recycler-pod-template-filepath-nfs string
对 NFS 卷执行回收利用时，用作模版的 Pod 定义文件所在路径。
--pv-recycler-timeout-increment-hostpath int32 默认值：30
HostPath 清洗器 Pod 在清洗对应类型持久卷时，为每 GiB 数据增加此标志所设置的秒数， 作为其 activeDeadlineSeconds 参数。此标志仅用于开发和测试环境，不适合多节点集群环境。
--pvclaimbinder-sync-period duration 默认值：15s
持久卷（PV）和持久卷申领（PVC）对象的同步周期。
--requestheader-allowed-names stringSlice
标志值是客户端证书中的 Common Names 列表。其中所列的名称可以通过 --requestheader-username-headers 所设置的 HTTP 头部来提供用户名。 如果此标志值为空表，则被 --requestheader-client-ca-file 中机构所验证过的所有客户端证书都是允许的。
--requestheader-client-ca-file string
根证书包文件名。在信任通过 --requestheader-username-headers 所指定的任何用户名之前，要使用这里的证书来检查请求中的客户证书。 警告：一般不要依赖对请求所作的鉴权结果。
--requestheader-extra-headers-prefix stringSlice 默认值：[x-remote-extra-]
要插入的请求头部前缀。建议使用 X-Remote-Exra-。
--requestheader-group-headers stringSlice 默认值：[x-remote-group]
用来检查用户组名的请求头部名称列表。建议使用 X-Remote-Group。
--requestheader-username-headers stringSlice 默认值：[x-remote-user]
用来检查用户名的请求头部名称列表。建议使用 X-Remote-User。
--resource-quota-sync-period duration 默认值：5m0s
对系统中配额用量信息进行同步的周期。
--root-ca-file string
如果此标志非空，则在服务账号的令牌 Secret 中会包含此根证书机构。 所指定标志值必须是一个合法的 PEM 编码的 CA 证书包。
--route-reconciliation-period duration 默认值：10s

	对云驱动为节点所创建的路由信息进行调解的周期。
--secondary-node-eviction-rate float32　默认值：0.01	
	当区域不健康，节点失效时，每秒钟从此标志所给的节点个数上删除 Pods。 参见 --unhealthy-zone-threshold 以了解"健康与否"的判定标准。 在只有一个区域的集群中，区域指的是整个集群。如果集群规模小于 --large-cluster-size-threshold 所设置的节点个数时，此值被隐式地重设为 0。
--secure-port int　默认值：10257	
	在此端口上提供 HTTPS 身份认证和鉴权操作。若此标志值为 0，则不提供 HTTPS 服务。
--service-account-private-key-file string	
	包含 PEM 编码的 RSA 或 ECDSA 私钥数据的文件名，这些私钥用来对服务账号令牌签名。
--service-cluster-ip-range string	
	集群中 Service 对象的 CIDR 范围。要求 `--allocate-node-cidrs` 标志为 true。
--show-hidden-metrics-for-version string	
	你希望展示隐藏度量值的上一个版本。只有上一个次版本号有意义，其他值都是不允许的。 字符串格式为 "<major>.<minor>"。例如："1.16"。 此格式的目的是确保你能够有机会注意到下一个版本隐藏了一些额外的度量值， 而不是在更新版本中某些度量值被彻底删除时措手不及。
--skip-headers	
	若此标志为 true，则在日志消息中避免写入头部前缀信息。
--skip-log-headers	
	若此标志为 true，则在写入日志文件时避免写入头部信息。
--stderrthreshold severity　默认值：2	
	等于或大于此阈值的日志信息会被写入到标准错误输出（stderr）。
--terminated-pod-gc-threshold int32　默认值：12500	
	在已终止 Pods 垃圾收集器删除已终止 Pods 之前，可以保留的已删除 Pods 的个数上限。若此值小于等于 0，则相当于禁止垃圾回收已终止的 Pods。
--tls-cert-file string	
	包含 HTTPS 所用的默认 X509 证书的文件。如果有 CA 证书，会被串接在服务器证书之后。 若启用了 HTTPS 服务且 --tls-cert-file 和 --tls-private-key-file 标志未设置， 则为节点的公开地址生成自签名的证书和密钥，并保存到 --cert-dir 所给的目录中。
--tls-cipher-suites stringSlice	

	供服务器使用的加密包的逗号分隔列表。若忽略此标志，则使用 Go 语言默认的加密包。 可选值包括：TLS_AES_128_GCM_SHA256、TLS_AES_256_GCM_SHA384、 TLS_CHACHA20_POLY1305_SHA256、 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA、 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256、 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA、 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384、 TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305、 TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256、 TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA、 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA、 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256、 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA、 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384、 TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305、 TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256、 TLS_RSA_WITH_3DES_EDE_CBC_SHA、 TLS_RSA_WITH_AES_128_CBC_SHA、 TLS_RSA_WITH_AES_128_GCM_SHA256、 TLS_RSA_WITH_AES_256_CBC_SHA、 TLS_RSA_WITH_AES_256_GCM_SHA384. 不安全的值: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256、 TLS_ECDHE_ECDSA_WITH_RC4_128_SHA、 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256、 TLS_ECDHE_RSA_WITH_RC4_128_SHA、 TLS_RSA_WITH_AES_128_CBC_SHA256、TLS_RSA_WITH_RC4_128_SHA
--tls-min-version string	
	可支持的最低 TLS 版本。可选值包括： "VersionTLS10"、"VersionTLS11"、"VersionTLS12"、"VersionTLS13"。
--tls-private-key-file string	
	包含与 --tls-cert-file 对应的默认 X509 私钥的文件。
--tls-sni-cert-key namedCertKey　默认值：[]	
	X509 证书和私钥文件路径的耦对。作为可选项，可以添加域名模式的列表， 其中每个域名模式都是可以带通配片段前缀的全限定域名（FQDN）。 域名模式也可以使用 IP 地址字符串，不过只有 API 服务器在所给 IP 地址上 对客户端可见时才可以使用 IP 地址。在未提供域名模式时，从证书中提取域名。 如果有非通配方式的匹配，则优先于通配方式的匹配；显式的域名模式优先于提取的域名。 当存在多个密钥/证书耦对时，可以多次使用 --tls-sni-cert-key 标志。 例如：example.crt,example.key 或 foo.crt,foo.key:*.foo.com,foo.com。
--unhealthy-zone-threshold float32　默认值：0.55	
	仅当给定区域中处于非就绪状态的节点（最少 3 个）的占比高于此值时，才将该区域视为不健康。
--use-service-account-credentials	

	当此标志为 true 时，为每个控制器单独使用服务账号凭据。
-v, --v Level	
	日志级别详细程度取值
--version version[=true]	
	打印版本信息之后退出
--vmodule moduleSpec	
	由逗号分隔的列表，每一项都是 pattern=N 格式，用来执行根据文件过滤的日志行为。
--volume-host-allow-local-loopback 默认值：true	
	此标志为 false 时，禁止本地回路 IP 地址和 --volume-host-cidr-denylist 中所指定的 CIDR 范围。
--volume-host-cidr-denylist stringSlice	
	用逗号分隔的一个 CIDR 范围列表，禁止使用这些地址上的卷插件。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 20, 2021 at 9:49 AM PST: Resync kcm reference (9e2873bbd)

kube-proxy

简介

Kubernetes 网络代理在每个节点上运行。网络代理反映了每个节点上 Kubernetes API 中定义的服务，并且可以执行简单的 TCP、UDP 和 SCTP 流转发，或者在一组后端进行循环 TCP、UDP 和 SCTP 转发。当前可通过 Docker-links-compatible 环境变量找到服务集群 IP 和端口，这些环境变量指定了服务代理打开的端口。有一个可选的插件，可以为这些集群 IP 提供集群 DNS。用户必须使用 apiserver API 创建服务才能配置代理。

```
kube-proxy [flags]
```

选项

--azure-container-registry-config string
包含 Azure 容器仓库配置信息的文件的路径。
--bind-address 0.0.0.0 默认值: 0.0.0.0

	代理服务器要使用的 IP 地址（对于所有 IPv4 接口设置为 0.0.0.0，对于所有 IPv6 接口设置为 ::）
--cleanup	
	如果为 true，清理 iptables 和 ipvs 规则并退出。
--cleanup-ipvs　默认值: true	
	如果设置为 true 并指定了 --cleanup，则 kube-proxy 除了常规清理外，还将刷新 IPVS 规则。
--cluster-cidr string	
	集群中 Pod 的 CIDR 范围。配置后，将从该范围之外发送到服务集群 IP 的流量被伪装，从 Pod 发送到外部 LoadBalancer IP 的流量将被重定向到相应的集群 IP。
--config string	
	配置文件的路径。
--config-sync-period duration　默认值: 15m0s	
	来自 apiserver 的配置的刷新频率。必须大于 0。
--conntrack-max-per-core int32　默认值: 32768	
	每个 CPU 核跟踪的最大 NAT 连接数（0 表示保留原样限制并忽略 conntrack-min）。
--conntrack-min int32　默认值: 131072	
	无论 conntrack-max-per-core 多少，要分配的 conntrack 条目的最小数量（将 conntrack-max-per-core 设置为 0 即可保持原样的限制）。
--conntrack-tcp-timeout-close-wait duration　默认值: 1h0m0s	
	处于 CLOSE_WAIT 状态的 TCP 连接的 NAT 超时
--conntrack-tcp-timeout-established duration　默认值: 24h0m0s	
	已建立的 TCP 连接的空闲超时（0 保持原样）
--detect-local-mode LocalMode	
	用于检测本地流量的模式
--feature-gates mapStringBool	

一组键=值（key=value）对，描述了 alpha/experimental 的特征。可选项有：
APIListChunking=true|false (BETA - 默认值=true)
APIPriorityAndFairness=true|false (ALPHA - 默认值=false)
APIResponseCompression=true|false (BETA - 默认值=true)
AllAlpha=true|false (ALPHA - 默认值=false)
AllBeta=true|false (BETA - 默认值=false)
AllowInsecureBackendProxy=true|false (BETA - 默认值=true)
AnyVolumeDataSource=true|false (ALPHA - 默认值=false)
AppArmor=true|false (BETA - 默认值=true)
BalanceAttachedNodeVolumes=true|false (ALPHA - 默认值=false)
BoundServiceAccountTokenVolume=true|false (ALPHA - 默认值=false)
CPUManager=true|false (BETA - 默认值=true)
CRIContainerLogRotation=true|false (BETA - 默认值=true)
CSIInlineVolume=true|false (BETA - 默认值=true)
CSIMigration=true|false (BETA - 默认值=true)
CSIMigrationAWS=true|false (BETA - 默认值=false)
CSIMigrationAWSComplete=true|false (ALPHA - 默认值=false)
CSIMigrationAzureDisk=true|false (BETA - 默认值=false)
CSIMigrationAzureDiskComplete=true|false (ALPHA - 默认值=false)
CSIMigrationAzureFile=true|false (ALPHA - 默认值=false)
CSIMigrationAzureFileComplete=true|false (ALPHA - 默认值=false)
CSIMigrationGCE=true|false (BETA - 默认值=false)
CSIMigrationGCEComplete=true|false (ALPHA - 默认值=false)
CSIMigrationOpenStack=true|false (BETA - 默认值=false)
CSIMigrationOpenStackComplete=true|false (ALPHA - 默认值=false)
CSIMigrationvSphere=true|false (BETA - 默认值=false)
CSIMigrationvSphereComplete=true|false (BETA - 默认值=false)
CSIStorageCapacity=true|false (ALPHA - 默认值=false)
CSIVolumeFSGroupPolicy=true|false (ALPHA - 默认值=false)
ConfigurableFSGroupPolicy=true|false (ALPHA - 默认值=false)
CustomCPUCFSQuotaPeriod=true|false (ALPHA - 默认值=false)
DefaultPodTopologySpread=true|false (ALPHA - 默认值=false)
DevicePlugins=true|false (BETA - 默认值=true)
DisableAcceleratorUsageMetrics=true|false (ALPHA - 默认值=false)
DynamicKubeletConfig=true|false (BETA - 默认值=true)
EndpointSlice=true|false (BETA - 默认值=true)
EndpointSliceProxying=true|false (BETA - 默认值=true)
EphemeralContainers=true|false (ALPHA - 默认值=false)
ExpandCSIVolumes=true|false (BETA - 默认值=true)
ExpandInUsePersistentVolumes=true|false (BETA - 默认值=true)
ExpandPersistentVolumes=true|false (BETA - 默认值=true)
ExperimentalHostUserNamespace默认值ing=true|false (BETA - 默认值=false)
GenericEphemeralVolume=true|false (ALPHA - 默认值=false)
HPAScaleToZero=true|false (ALPHA - 默认值=false)
HugePageStorageMediumSize=true|false (BETA - 默认值=true)
HyperVContainer=true|false (ALPHA - 默认值=false)
IPv6DualStack=true|false (ALPHA - 默认值=false)

--healthz-bind-address 0.0.0.0　默认值: 0.0.0.0:10256
服务健康检查的 IP 地址和端口（对于所有 IPv4 接口设置为 '0.0.0.0:10256'，对于所有 IPv6 接口设置为 '[::]:10256'） 设置为空则禁用。
--healthz-bind-address 0.0.0.0　默认值: 0.0.0.0:10256
服务健康检查的 IP 地址和端口（设置为 0.0.0.0 表示使用所有 IPv4 接口，设置为 :: 表示使用所有 IPv6 接口）
-h, --help
kube-proxy 操作的帮助命令
--hostname-override string
如果非空，将使用此字符串作为标识而不是实际的主机名。
--iptables-masquerade-bit int32　默认值: 14
如果使用纯 iptables 代理，则 fwmark 空间的 bit 用于标记需要 SNAT 的数据包。必须在 [0,31] 范围内。
--iptables-min-sync-period duration　默认值：1s
iptables 规则可以随着端点和服务的更改而刷新的最小间隔（例如 '5s'、'1m'、'2h22m'）。
--iptables-sync-period duration　默认值: 30s
刷新 iptables 规则的最大间隔（例如 '5s'、'1m'、'2h22m'）。必须大于 0。
--ipvs-exclude-cidrs stringSlice
逗号分隔的 CIDR 列表，ipvs 代理在清理 IPVS 规则时不应使用此列表。
--ipvs-min-sync-period duration
ipvs 规则可以随着端点和服务的更改而刷新的最小间隔（例如 '5s'、'1m'、'2h22m'）。
--ipvs-scheduler string
代理模式为 ipvs 时的 ipvs 调度器类型
--ipvs-strict-arp
通过将 arp_ignore 设置为 1 并将 arp_announce 设置为 2 启用严格的 ARP
--ipvs-sync-period duration　默认值: 30s
刷新 ipvs 规则的最大间隔（例如 '5s'、'1m'、'2h22m'）。必须大于 0。
--ipvs-tcp-timeout duration
空闲 IPVS TCP 连接的超时时间，0 保持连接（例如 '5s'、'1m'、'2h22m'）。
--ipvs-tcpfin-timeout duration
收到 FIN 数据包后，IPVS TCP 连接的超时，0 保持连接不变（例如 '5s'、'1m'、'2h22m'）。
--ipvs-udp-timeout duration
IPVS UDP 数据包的超时，0 保持连接不动（例如 '5s'、'1m'、'2h22m'）。
--kube-api-burst int32　默认值: 10
与 kubernetes apiserver 通信的数量
--kube-api-content-type string　默认值: "application/vnd.kubernetes.protobuf"
发送到 apiserver 的请求的内容类型。
--kube-api-qps float32　默认值: 5

	与 kubernetes apiserver 交互时使用的 QPS
--kubeconfig string	
	包含授权信息的 kubeconfig 文件的路径（master 位置由 master 标志设置）。
--log-flush-frequency duration　默认值: 5s	
	两次日志刷新之间的最大秒数
--masquerade-all	
	如果使用纯 iptables 代理，则对通过服务集群 IP 发送的所有流量进行 SNAT（通常不需要）
--master string	
	Kubernetes API 服务器的地址（覆盖 kubeconfig 中的任何值）
--metrics-bind-address ipport 0.0.0.0　默认值: 127.0.0.1:10249	
	metrics 服务器要使用的 IP 地址和端口 （设置为 '0.0.0.0:10249' 则使用 IPv4 接口，设置为 '[::]:10249' 则使用所有 IPv6 接口） 设置为空则禁用。
--metrics-port int32　默认值: 10249	
	绑定 metrics 服务器的端口。使用 0 表示禁用。
--nodeport-addresses stringSlice	
	一个字符串值，指定用于 NodePorts 的地址。值可以是有效的 IP 块（例如 1.2.3.0/24, 1.2.3.4/32）。默认的空字符串切片（[]）表示使用所有本地地址。
--oom-score-adj int32　默认值: -999	
	kube-proxy 进程中的 oom-score-adj 值必须在 [-1000,1000] 范围内
--profiling	
	如果为 true，则通过 Web 接口 /debug/pprof 启用性能分析。
--proxy-mode ProxyMode	
	使用哪种代理模式：'userspace'（较旧）或 'iptables'（较快）或 'ipvs'（实验）。如果为空，使用最佳可用代理（当前为 iptables）。如果选择了 iptables 代理，无论如何，但系统的内核或 iptables 版本较低，这总是会回退到用户空间代理。
--proxy-port-range port-range	
	可以使用代理服务流量的主机端口（包括 beginPort-endPort、single port、beginPort+offset）的范围。如果（未指定，0 或 0-0）则随机选择端口。
--show-hidden-metrics-for-version string	
	你要显示隐藏指标的先前版本。 仅先前的次要版本有意义，不允许其他值。 格式为 <major>.<minor> ，例如：'1.16'。 这种格式的目的是确保你有机会注意到下一个发行版是否隐藏了其他指标， 而不是在之后将其永久删除时感到惊讶。
--udp-timeout duration　默认值: 250ms	
	空闲 UDP 连接将保持打开的时长（例如 '250ms'，'2s'）。必须大于 0。仅适用于 proxy-mode=userspace
--version version[=true]	
	打印版本信息并退出
--write-config-to string	
	如果设置，将配置值写入此文件并退出。

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 02, 2020 at 3:19 PM PST: sync kube-controller-manager, kube-proxy and kubelet-authentication-authorization (05381f607)

Kubelet 认证/鉴权

概述

kubelet 的 HTTPS 端点公开了 API， 这些 API 可以访问敏感度不同的数据， 并允许你在节点上和容器内以不同级别的权限执行操作。

本文档介绍了如何对 kubelet 的 HTTPS 端点的访问进行认证和鉴权。

Kubelet 身份认证

默认情况下，未被已配置的其他身份认证方法拒绝的对 kubelet 的 HTTPS 端点的请求会被视为匿名请求， 并被赋予 system:anonymous 用户名和 system:unauthenticated 组。

要禁用匿名访问并向未经身份认证的请求发送 401 Unauthorized 响应，请执行以下操作：

- 带 --anonymous-auth=false 标志启动 kubelet

要对 kubelet 的 HTTPS 端点启用 X509 客户端证书认证：

- 带 --client-ca-file 标志启动 kubelet，提供一个 CA 证书包以供验证客户端证书
- 带 --kubelet-client-certificate 和 --kubelet-client-key 标志启动 apiserver
- 有关更多详细信息，请参见 apiserver 身份验证文档

要启用 API 持有者令牌（包括服务帐户令牌）以对 kubelet 的 HTTPS 端点进行身份验证，请执行以下操作：

- 确保在 API 服务器中启用了 authentication.k8s.io/v1beta1 API 组
- 带 --authentication-token-webhook 和 --kubeconfig 标志启动 kubelet
- kubelet 调用已配置的 API 服务器上的 TokenReview API，以根据持有者令牌确定用户信息

Kubelet 鉴权

任何成功通过身份验证的请求（包括匿名请求）之后都会被鉴权。 默认的鉴权模式为 AlwaysAllow，它允许所有请求。

细分对 kubelet API 的访问权限可能有多种原因：

- 启用了匿名身份验证，但是应限制匿名用户调用 kubelet API 的能力
- 启用了持有者令牌认证，但应限制任意 API 用户（如服务帐户）调用 kubelet API 的能力
- 启用了客户端证书身份验证，但仅应允许已配置的 CA 签名的某些客户端证书使用 kubelet API

要细分对 kubelet API 的访问权限，请将鉴权委派给 API 服务器：

- 确保在 API 服务器中启用了 authorization.k8s.io/v1beta1 API 组
- 带 --authorization-mode=Webhook 和 --kubeconfig 标志启动 kubelet
- kubelet 调用已配置的 API 服务器上的 SubjectAccessReview API， 以确定每个请求是否得到鉴权

kubelet 使用与 apiserver 相同的 请求属性 方法对 API 请求执行鉴权。

请求的动词根据传入请求的 HTTP 动词确定：

HTTP 动词	请求动词
POST	create
GET, HEAD	get
PUT	update
PATCH	patch
DELETE	delete

资源和子资源是根据传入请求的路径确定的：

Kubelet API	资源	子资源
/stats/*	nodes	stats
/metrics/*	nodes	metrics
/logs/*	nodes	log
/spec/*	nodes	spec
其它所有	nodes	proxy

名字空间和 API 组属性始终是空字符串， 资源名称始终是 kubelet 的 Node API 对象的名称。

在此模式下运行时，请确保传递给 apiserver 的由 --kubelet-client-certificate 和 --kubelet-client-key 标志标识的用户具有以下属性的鉴权：

- verb=*, resource=nodes, subresource=proxy
- verb=*, resource=nodes, subresource=stats
- verb=*, resource=nodes, subresource=log

- verb=*, resource=nodes, subresource=spec
- verb=*, resource=nodes, subresource=metrics

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 04, 2020 at 6:22 PM PST: [zh] Fix links in zh localization (4) (abab517ff)

TLS bootstrapping

Overview

This document describes how to set up TLS client certificate bootstrapping for kubelets. Kubernetes 1.4 introduced an API for requesting certificates from a cluster-level Certificate Authority (CA). The original intent of this API is to enable provisioning of TLS client certificates for kubelets. The proposal can be found here and progress on the feature is being tracked as feature #43.

kube-apiserver configuration

The API server should be configured with an authenticator that can authenticate tokens as a user in the `system:bootstrappers` group.

This group will later be used in the controller-manager configuration to scope approvals in the default approval controller. As this feature matures, you should ensure tokens are bound to a Role-Based Access Control (RBAC) policy which limits requests (using the bootstrap token) strictly to client requests related to certificate provisioning. With RBAC in place, scoping the tokens to a group allows for great flexibility (e.g. you could disable a particular bootstrap group's access when you are done provisioning the nodes).

While any authentication strategy can be used for the kubelet's initial bootstrap credentials, the following two authenticators are recommended for ease of provisioning.

1. Bootstrap Tokens - **alpha**
2. Token authentication file

Using bootstrap tokens is currently **alpha** and will simplify the management of bootstrap token management especially in a HA scenario.

Token authentication file

Tokens are arbitrary but should represent at least 128 bits of entropy derived from a secure random number generator (such as /dev/urandom on most modern systems). There are multiple ways you can generate a token. For example:

`head -c 16 /dev/urandom | od -An -t x | tr -d ' '`

will generate tokens that look like `02b50b05283e98dd0fd71db496ef01e8`

The token file should look like the following example, where the first three values can be anything and the quoted group name should be as depicted:

```
02b50b05283e98dd0fd71db496ef01e8,kubelet-bootstrap,
10001,"system:bootstrappers"
```

Add the `--token-auth-file=FILENAME` flag to the kube-apiserver command (in your systemd unit file perhaps) to enable the token file. See docs here for further details.

Client certificate CA bundle

Add the `--client-ca-file=FILENAME` flag to the kube-apiserver command to enable client certificate authentication, referencing a certificate authority bundle containing the signing certificate (e.g. `--client-ca-file=/var/lib/kubernetes/ca.pem`).

kube-controller-manager configuration

The API for requesting certificates adds a certificate-issuing control loop to the Kubernetes Controller Manager. This takes the form of a cfssl local signer using assets on disk. Currently, all certificates issued have one year validity and a default set of key usages.

Signing assets

You must provide a Certificate Authority in order to provide the cryptographic materials necessary to issue certificates. This CA should be trusted by kube-apiserver for authentication with the `--client-ca-`

file=FILENAME flag. The management of the CA is beyond the scope of this document but it is recommended that you generate a dedicated CA for Kubernetes. Both certificate and key are assumed to be PEM-encoded.

The kube-controller-manager flags are:

```
--cluster-signing-cert-file="/etc/path/to/kubernetes/ca/ca.crt" --cluster-signing-key-file="/etc/path/to/kubernetes/ca/ca.key"
```

Approval controller

In 1.7 the experimental "group auto approver" controller is dropped in favor of the new csrapproving controller that ships as part of kube-controller-manager and is enabled by default. The controller uses the SubjectAccessReview API to determine if a given user is authorized to request a CSR, then approves based on the authorization outcome. To prevent conflicts with other approvers, the builtin approver doesn't explicitly deny CSRs, only ignoring unauthorized requests.

The controller categorizes CSRs into three subresources:

1. nodeclient - a request by a user for a client certificate with O=system:nodes and CN=system:node:(node name).
2. selfnodeclient - a node renewing a client certificate with the same O and CN.
3. selfnodeserver - a node renewing a serving certificate. (ALPHA, requires feature gate)

The checks to determine if a CSR is a selfnodeserver request is currently tied to the kubelet's credential rotation implementation, an **alpha** feature. As such, the definition of selfnodeserver will likely change in a future and requires the RotateKubeletServerCertificate feature gate on the controller manager. The feature progress can be tracked at kubernetes/features#267.

```
--feature-gates=RotateKubeletServerCertificate=true
```

The following RBAC ClusterRoles represent the nodeclient, selfnodeclient, and selfnodeserver capabilities. Similar roles may be automatically created in future releases.

```
# A ClusterRole which instructs the CSR approver to approve a user requesting
# node client credentials.
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-client-csr
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/nodeclient"]
```

```
  verbs: ["create"]
---
# A ClusterRole which instructs the CSR approver to approve a node renewing its
# own client credentials.
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-client-renewal-csr
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/selfnodeclient"]
  verbs: ["create"]
---
# A ClusterRole which instructs the CSR approver to approve a node requesting a
# serving cert matching its client cert.
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-server-renewal-csr
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/selfnodeserver"]
  verbs: ["create"]
```

These powers can be granted to credentials, such as bootstrapping tokens. For example, to replicate the behavior provided by the removed auto-approval flag, of approving all CSRs by a single group:

```
# REMOVED: This flag no longer works as of 1.7.
--insecure-experimental-approve-all-kubelet-csrs-for-
group="system:bootstrappers"
```

An admin would create a `ClusterRoleBinding` targeting that group.

```
# Approve all CSRs for the group "system:bootstrappers"
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: auto-approve-csrs-for-group
subjects:
- kind: Group
  name: system:bootstrappers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: approve-node-client-csr
  apiGroup: rbac.authorization.k8s.io
```

To let a node renew its own credentials, an admin can construct a ClusterRoleBinding targeting that node's credentials:

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: node1-client-cert-renewal
subjects:
- kind: User
  name: system:node:node-1 # Let "node-1" renew its client certificate.
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: approve-node-client-renewal-csr
  apiGroup: rbac.authorization.k8s.io
```

Deleting the binding will prevent the node from renewing its client credentials, effectively removing it from the cluster once its certificate expires.

kubelet configuration

To request a client certificate from kube-apiserver, the kubelet first needs a path to a kubeconfig file that contains the bootstrap authentication token. You can use kubectl config set-cluster, set-credentials, and set-context to build this kubeconfig. Provide the name kubelet-bootstrap to kubectl config set-credentials and include --token=<token-value> as follows:

```
kubectl config set-credentials kubelet-bootstrap --token=${BOOTSTRAP_TOKEN}
--kubeconfig=bootstrap.kubeconfig
```

When starting the kubelet, if the file specified by --kubeconfig does not exist, the bootstrap kubeconfig is used to request a client certificate from the API server. On approval of the certificate request and receipt back by the kubelet, a kubeconfig file referencing the generated key and obtained certificate is written to the path specified by --kubeconfig. The certificate and key file will be placed in the directory specified by --cert-dir.

> **说明：**
>
> The following flags are required to enable this bootstrapping when starting the kubelet:
>
> ```
> --bootstrap-kubeconfig="/path/to/bootstrap/kubeconfig"
> ```

Additionally, in 1.7 the kubelet implements **alpha** features for enabling rotation of both its client and/or serving certs. These can be enabled through the respective RotateKubeletClientCertificate and RotateKubeletServerCertificate

feature flags on the kubelet, but may change in backward incompatible ways in future releases.

```
--feature-gates=RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true
```

RotateKubeletClientCertificate causes the kubelet to rotate its client certificates by creating new CSRs as its existing credentials expire. RotateKubeletServerCertificate causes the kubelet to both request a serving certificate after bootstrapping its client credentials and rotate the certificate. The serving cert currently does not request DNS or IP SANs.

kubectl approval

The signing controller does not immediately sign all certificate requests. Instead, it waits until they have been flagged with an "Approved" status by an appropriately-privileged user. This is intended to eventually be an automated process handled by an external approval controller, but for the alpha version of the API it can be done manually by a cluster administrator using kubectl. An administrator can list CSRs with `kubectl get csr` and describe one in detail with `kubectl describe csr <name>`. Before the 1.6 release there were no direct approve/deny commands so an approver had to update the Status field directly (rough how-to). Later versions of Kubernetes offer `kubectl certificate approve <name>` and `kubectl certificate deny <name>` commands.

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 04, 2020 at 6:22 PM PST: [zh] Fix links in zh localization (4) (abab517ff)

调度

调度策略

调度器配置

调度策略

kube-scheduler 根据调度策略指定的*断言（predicates）和优先级（priorities）*分别对节点进行过滤和打分。

你可以通过执行 kube-scheduler --policy-config-file <filename> 或 kube-scheduler --policy-configmap <ConfigMap> 设置并使用调度策略。

断言

以下*断言*实现了过滤接口：

- PodFitsHostPorts：检查 Pod 请求的端口（网络协议类型）在节点上是否可用。
- PodFitsHost：检查 Pod 是否通过主机名指定了 Node。
- PodFitsResources：检查节点的空闲资源（例如，CPU和内存）是否满足 Pod 的要求。
- MatchNodeSelector：检查 Pod 的节点选择算符 和节点的 标签 是否匹配。
- NoVolumeZoneConflict：给定该存储的故障区域限制， 评估 Pod 请求的卷在节点上是否可用。
- NoDiskConflict：根据 Pod 请求的卷是否在节点上已经挂载，评估 Pod 和节点是否匹配。
- MaxCSIVolumeCount：决定附加 CSI 卷的数量，判断是否超过配置的限制。
- CheckNodeMemoryPressure：如果节点正上报内存压力，并且没有异常配置，则不会把 Pod 调度到此节点上。
- CheckNodePIDPressure：如果节点正上报进程 ID 稀缺，并且没有异常配置，则不会把 Pod 调度到此节点上。
- CheckNodeDiskPressure：如果节点正上报存储压力（文件系统已满或几乎已满），并且没有异常配置，则不会把 Pod 调度到此节点上。
- CheckNodeCondition：节点可用上报自己的文件系统已满，网络不可用或者 kubelet 尚未准备好运行 Pod。 如果节点上设置了这样的状况，并且没有异常配置，则不会把 Pod 调度到此节点上。
- PodToleratesNodeTaints：检查 Pod 的容忍 是否能容忍节点的污点。
- CheckVolumeBinding：基于 Pod 的卷请求，评估 Pod 是否适合节点，这里的卷包括绑定的和未绑定的 PVCs 都适用。

优先级

以下*优先级*实现了打分接口：

- SelectorSpreadPriority：属于同一 Service、 StatefulSet 或 ReplicaSet 的 Pod，跨主机部署。
- InterPodAffinityPriority：实现了 Pod 间亲和性与反亲和性的优先级。
- LeastRequestedPriority：偏向最少请求资源的节点。 换句话说，节点上的 Pod 越多，使用的资源就越多，此策略给出的排名就越低。
- MostRequestedPriority：支持最多请求资源的节点。 该策略将 Pod 调度到整体工作负载所需的最少的一组节点上。
- RequestedToCapacityRatioPriority：使用默认的打分方法模型，创建基于 ResourceAllocationPriority 的 requestedToCapacity。
- BalancedResourceAllocation：偏向平衡资源使用的节点。
- NodePreferAvoidPodsPriority：根据节点的注解 scheduler.alpha.kubernetes.io/preferAvoidPods 对节点进行优先级排序。 你可以使用它来暗示两个不同的 Pod 不应在同一节点上运行。
- NodeAffinityPriority：根据节点亲和中 PreferredDuringSchedulingIgnoredDuringExecution 字段对节点进行优先级排序。 你可以在将 Pod 分配给节点中了解更多。
- TaintTolerationPriority：根据节点上无法忍受的污点数量，给所有节点进行优先级排序。 此策略会根据排序结果调整节点的等级。
- ImageLocalityPriority：偏向已在本地缓存 Pod 所需容器镜像的节点。
- ServiceSpreadingPriority：对于给定的 Service，此策略旨在确保该 Service 关联的 Pod 在不同的节点上运行。 它偏向把 Pod 调度到没有该服务的节点。 整体来看，Service 对于单个节点故障变得更具弹性。
- EqualPriority：给予所有节点相等的权重。
- EvenPodsSpreadPriority：实现了 Pod 拓扑扩展约束的优先级排序。

接下来

- 了解调度
- 了解 kube-scheduler 配置

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 September 22, 2020 at 10:32 PM PST: translate content/zh/docs/reference/scheduling/policies.md (22ddf4f82)

调度器配置

FEATURE STATE: Kubernetes v1.19 [beta]

你可以通过编写配置文件，并将其路径传给 kube-scheduler 的命令行参数，定制 kube-scheduler 的行为。

调度模板（Profile）允许你配置 kube-scheduler 中的不同调度阶段。每个阶段都暴露于某个扩展点中。插件通过实现一个或多个扩展点来提供调度行为。

你可以通过运行 kube-scheduler --config <filename> 来设置调度模板， 配置文件使用组件配置的 API (v1alpha1)。

最简单的配置如下：

```
apiVersion: kubescheduler.config.k8s.io/v1beta1
kind: KubeSchedulerConfiguration
clientConnection:
  kubeconfig: /etc/srv/kubernetes/kube-scheduler/kubeconfig
```

配置文件

通过调度配置文件，你可以配置 kube-scheduler 在不同阶段的调度行为。 每个阶段都在一个扩展点中公开。 调度插件通过实现一个或多个扩展点，来提供调度行为。

你可以配置同一 kube-scheduler 实例使用多个配置文件。

扩展点

调度行为发生在一系列阶段中，这些阶段是通过以下扩展点公开的：

1. QueueSort：这些插件对调度队列中的悬决的 Pod 排序。 一次只能启用一个队列排序插件。
1. PreFilter：这些插件用于在过滤之前预处理或检查 Pod 或集群的信息。 它们可以将 Pod 标记为不可调度。
1. Filter：这些插件相当于调度策略中的断言（Predicates），用于过滤不能运行 Pod 的节点。 过滤器的调用顺序是可配置的。 如果没有一个节点通过所有过滤器的筛选，Pod 将会被标记为不可调度。
1. PreScore：这是一个信息扩展点，可用于预打分工作。
1. Score：这些插件给通过筛选阶段的节点打分。调度器会选择得分最高的节点。
1. Reserve：这是一个信息扩展点，当资源已经预留给 Pod 时，会通知插件。 这些插件还实现了 Unreserve 接口，在 Reserve 期间或之后出现故障时调用。
1. Permit：这些插件可以阻止或延迟 Pod 绑定。
1. PreBind：这些插件在 Pod 绑定节点之前执行。
1. Bind：这个插件将 Pod 与节点绑定。绑定插件是按顺序调用的，只要有一个插件完成了绑定，其余插件都会跳过。绑定插件至少需要一个。
1. PostBind：这是一个信息扩展点，在 Pod 绑定了节点之后调用。

对每个扩展点，你可以禁用默认插件或者是启用自己的插件，例如：

```yaml
apiVersion: kubescheduler.config.k8s.io/v1beta1
kind: KubeSchedulerConfiguration
profiles:
  - plugins:
      score:
        disabled:
        - name: NodeResourcesLeastAllocated
        enabled:
        - name: MyCustomPluginA
          weight: 2
        - name: MyCustomPluginB
          weight: 1
```

你可以在 disabled 数组中使用 * 禁用该扩展点的所有默认插件。 如果需要，这个字段也可以用来对插件重新顺序。

调度插件

1. UnReserve：这是一个信息扩展点，如果一个 Pod 在预留后被拒绝，并且被 Permit 插件搁置，它就会被调用。

调度插件

下面默认启用的插件实现了一个或多个扩展点：

- SelectorSpread：对于属于 Services、 ReplicaSets 和 StatefulSets 的 Pod，偏好跨多个节点部署。

  实现的扩展点：PreScore，Score。

- ImageLocality：选择已经存在 Pod 运行所需容器镜像的节点。

  实现的扩展点：Score。

- TaintToleration：实现了污点和容忍。

  实现的扩展点：Filter，Prescore，Score。

- NodeName：检查 Pod 指定的节点名称与当前节点是否匹配。

  实现的扩展点：Filter。

- NodePorts：检查 Pod 请求的端口在节点上是否可用。

  实现的扩展点：PreFilter，Filter。

- NodePreferAvoidPods：基于节点的 注解 scheduler.alpha.kubernetes.io/preferAvoidPods 打分。

  实现的扩展点：Score。

- NodeAffinity：实现了节点选择器 和节点亲和性。

  实现的扩展点：Filter，Score.

- PodTopologySpread：实现了 Pod 拓扑分布。

  实现的扩展点：PreFilter，Filter，PreScore，Score。

- NodeUnschedulable：过滤 .spec.unschedulable 值为 true 的节点。

  实现的扩展点：Filter。

- NodeResourcesFit：检查节点是否拥有 Pod 请求的所有资源。

实现的扩展点：PreFilter，Filter。

- NodeResourcesBalancedAllocation：调度 Pod 时，选择资源使用更为均衡的节点。

  实现的扩展点：Score。

- NodeResourcesLeastAllocated：选择资源分配较少的节点。

  实现的扩展点：Score。

- VolumeBinding：检查节点是否有请求的卷，或是否可以绑定请求的卷。

  实现的扩展点: PreFilter，Filter，Reserve，PreBind。

- VolumeRestrictions：检查挂载到节点上的卷是否满足卷提供程序的限制。

  实现的扩展点：Filter。

- VolumeZone：检查请求的卷是否在任何区域都满足。

  实现的扩展点：Filter。

- NodeVolumeLimits：检查该节点是否满足 CSI 卷限制。

  实现的扩展点：Filter。

- EBSLimits：检查节点是否满足 AWS EBS 卷限制。

  实现的扩展点：Filter。

- GCEPDLimits：检查该节点是否满足 GCP-PD 卷限制。

  实现的扩展点：Filter。

- AzureDiskLimits：检查该节点是否满足 Azure 卷限制。

  实现的扩展点：Filter。

- InterPodAffinity：实现 Pod 间亲和性与反亲和性。

  实现的扩展点：PreFilter，Filter，PreScore，Score。

- PrioritySort：提供默认的基于优先级的排序。

  实现的扩展点：QueueSort。

- DefaultBinder：提供默认的绑定机制。

  实现的扩展点：Bind。

- DefaultPreemption：提供默认的抢占机制。

实现的扩展点：PostFilter。

你也可以通过组件配置 API 启用以下插件（默认不启用）：

- NodeResourcesMostAllocated：选择已分配资源多的节点。

  实现的扩展点：Score。

- RequestedToCapacityRatio：根据已分配资源的某函数设置选择节点。

  实现的扩展点：Score。

- NodeResourceLimits：选择满足 Pod 资源限制的节点。

  实现的扩展点：PreScore，Score。

- CinderVolume：检查该节点是否满足 OpenStack Cinder 卷限制。

  实现的扩展点：Filter。

- NodeLabel：根据配置的 标签 过滤节点和/或给节点打分。

  实现的扩展点：Filter，Score。

- ServiceAffinity：检查属于某个 服务（Service） 的 Pod 与配置的标签所定义的节点集是否适配。 这个插件还支持将属于某个 Service 的 Pod 分散到各个节点。

  实现的扩展点：PreFilter，Filter，Score。

多配置文件

你可以配置 kube-scheduler 运行多个配置文件。 每个配置文件都有一个关联的调度器名称，并且可以在其扩展点中配置一组不同的插件。

使用下面的配置样例，调度器将运行两个配置文件：一个使用默认插件，另一个禁用所有打分插件。

```yaml
apiVersion: kubescheduler.config.k8s.io/v1beta1
kind: KubeSchedulerConfiguration
profiles:
  - schedulerName: default-scheduler
  - schedulerName: no-scoring-scheduler
    plugins:
      preScore:
        disabled:
        - name: '*'
      score:
        disabled:
        - name: '*'
```

希望根据特定配置文件调度的 Pod，可以在 .spec.schedulerName 字段指定相应的调度器名称。

默认情况下，将创建一个名为 default-scheduler 的配置文件。 这个配置文件包括上面描述的所有默认插件。 声明多个配置文件时，每个配置文件中调度器名称必须唯一。

如果 Pod 未指定调度器名称，kube-apiserver 将会把它设置为 default-scheduler。 因此，应该存在一个名为 default-scheduler 的配置文件来调度这些 Pod。

> **说明：**
>
> Pod 的调度事件把 .spec.schedulerName 字段值作为 ReportingController。 领导者选择事件使用列表中第一个配置文件的调度器名称。

> **说明：**
>
> 所有配置文件必须在 QueueSort 扩展点使用相同的插件，并具有相同的配置参数（如果适用）。 这是因为调度器只有一个的队列保存悬决的 Pod。

接下来

- 阅读 kube-scheduler 参考
- 了解调度

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 January 08, 2021 at 1:52 AM PST: [zh] fix typo for clarity (c231dfcf6)

工具

Kubernetes 包含一些内置工具，可以帮助用户更好的使用 Kubernetes 系统。

Kubectl

kubectl 是 Kubernetes 命令行工具， 可以用来操控 Kubernetes 集群。

Kubeadm

kubeadm 是一个命令行工具， 可以用来在物理机、云服务器或虚拟机（目前处于 alpha 阶段） 上轻松部署一个安全可靠的 Kubernetes 集群。

Minikube

minikube 是一个可以方便用户 在其工作站点本地部署一个单节点 Kubernetes 集群的工具，用于开发和测试。

Dashboard

Dashboard 是 Kubernetes 基于 Web 的用户管理界面，允许用户部署容器化应用到 Kubernetes 集群，进行故障排查以及管理集群和集群资源。

Helm

Kubernetes Helm 是一个管理 预先配置完毕的 Kubernetes 资源包的工具，这里的资源在 Helm 中也被称作 Kubernetes charts。

使用 Helm：

- 查找并使用已经打包为 Kubernetes charts 的流行软件
- 分享您自己的应用作为 Kubernetes charts
- 为 Kubernetes 应用创建可重复执行的构建
- 为您的 Kubernetes 清单文件提供更智能化的管理
- 管理 Helm 软件包的发布

Kompose

Kompose 一个转换工具， 用来帮助 Docker Compose 用户迁移至 Kubernetes。

使用 Kompose:

- 将一个 Docker Compose 文件解释成 Kubernetes 对象
- 将本地 Docker 开发 转变成通过 Kubernetes 来管理
- 转换 v1 或 v2 Docker Compose yaml 文件 或 已发布的应用程序包

反馈

此页是否对您有帮助？

是 否

感谢反馈。如果您有一个关于如何使用 Kubernetes 的特定的、需要答案的问题，可以访问 Stack Overflow. 在 GitHub 仓库上登记新的问题 报告问题 或者 提出改进建议.

最后修改 December 17, 2020 at 4:23 PM PST: Update tools.md (4bb411324)