122Kubernetes 系列（一一六）Kubernetes API 实践

介绍

本文解释了 Kubernetes API 的结构和功能。第 1 节概述了 Kubernetes API。第 2 节解释了完成本文所需的要求。此外，第 3 节解释了 Kubernetes API 术语。此外，第 4 节还介绍了一些实际例子。第 5 节介绍了更高级的查询。最后，以一个结论来结束本文。

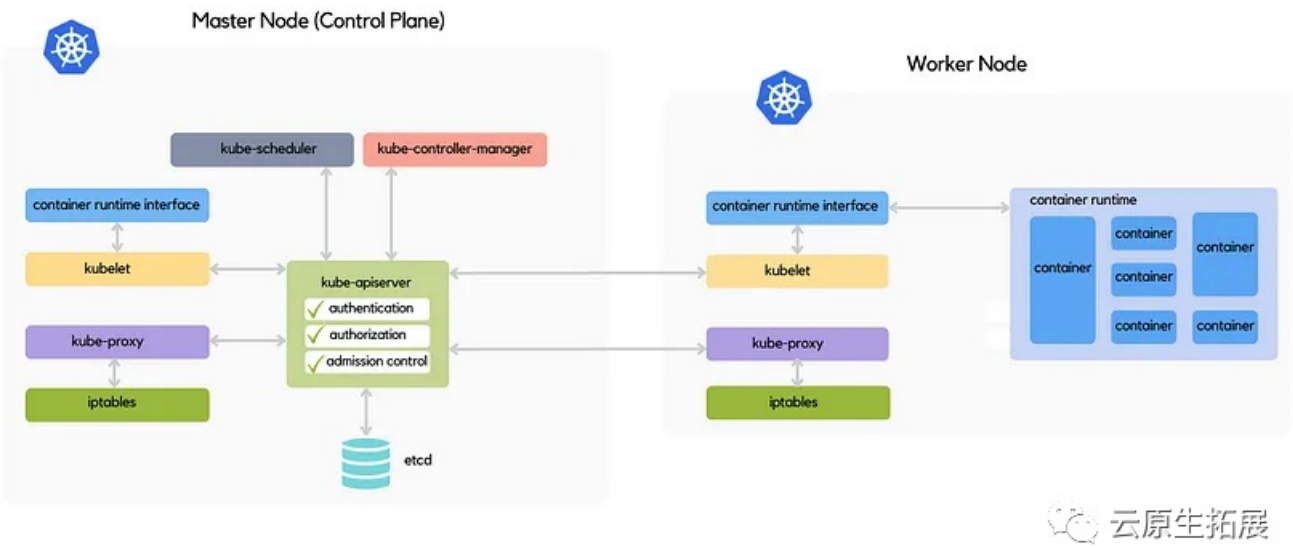
1. API 概述

Kubernetes (k8s) 采用 API 驱动的客户端/服务器架构，其中 API 服务器公开 HTTP API，使最终用户、集群的不同部分和外部组件能够相互通信。

> 为了更好地理解本文，需要具备 k8s 架构的基础知识。

值得记住的是，API 服务器是 k8s 集群的单一入口。它是唯一负责接收客户端查询的组件。并将它们转发给其余的 k8s 组件，然后响应客户端。

API服务器与之通信的k8s组件如下：



- Controller-Manager 包含 k8s 控制器（ deployments 、 statefulsets 等），负责保持 k8s 对象的当前状态与声明的所需状态同步。
- etcd 是一个一致且高可用的 b+tree 键值存储。它记录集群的状态、网络和其他有关集群的重要信息。
- Kube Scheduler 根据 etcd 中存储的信息确定在哪个节点上调度 pod 。

k8s 客户端（用户）可以使用客户端库或通过发出 REST 请求来查询和操作 k8s 中 API 对象的状态。例如，可以使用以下内容：

- kubectl 是k8s提供的命令行工具。它与 API 服务器建立通信并代表我们进行 API 调用，以便我们可以轻松访问和管理 k8s API 资源。

- curl 是一个命令行工具，旨在与服务器交换数据。在k8s中，服务器是k8s API服务器。

> k8s API 调用的默认序列化是 JSON。尽管我们倾向于使用 YAML 格式，但重要的是要知道 YAML 与 JSON 之间的转换。

用于与 API 服务器建立通信。 API 服务器对每个 API 调用执行 `authentication` 、 `authorization` 和 `admission control` 操作。

- `authentication` 当客户端提供有效的 TLS 证书时建立（有效的 TLS 证书是由集群 CA 签名并受 API 服务器信任的证书）。
- `authorization` 通过 RBAC （ Role, RoleBinding, ClusterRole, ClusterRoleBinding) ）建立，指定客户端是否可以或不可以对某些k8s对象执行CRUD操作（动词）。在这一步中，客户端身份由 common name field in the subject of the certificate for example CN=Alice 确定。
- 如果客户端对k8s对象执行的 CRUD operation 合法，则 `admission control` 成立。例如，当集群有一个 ResourceQuota 对象时，该对象将命名空间 dev 中可以创建的 pods 数量限制为 2。如果用户尝试创建 3d pod 那么它的请求将被拒绝。

> 值得注意的是，k8s 没有代表客户帐户的对象。无法通过 API 调用将客户端添加到集群。

2. 要求

> 当使用 curl 时。客户端在命令行上传递其 TLS 证书和集群的 CA，如下所示：

```
curl $server/api/v1/ --cacert ./ca.crt --cert ./client.crt --key ./client.key
```

> 当使用 kubectl 时，它使用已由 k8s admin 配置的 config 文件，其中包含客户端 TLS 证书和集群的 CA。该文件应该在 $HOME/.kube/config 下可用

```
cat $HOME/.kube/config
```

为了更好地理解 k8s API，本文将使用 curl 命令。

为了寻求简单性并避免处理 HTTPS 和 TLS 证书进行身份验证。 kubectl proxy 命令将用于创建代理服务器。

> kubectl proxy 命令在本地主机和 Kubernetes API 服务器之间创建代理服务器或应用程序级网关。它还允许通过指定的 HTTP 路径提供静态内容。所有传入数据都通过一个端口进入并转发到远程 Kubernetes API 服务器端口。

- 要创建代理服务器，只需运行：

```
kubectl proxy
```

- 要与 API 服务器通信，只需运行：

```
curl http://127.0.0.1:8001
```

- 为了更好的 JSON 格式和可视化，将使用 jq 命令，如下所示：

```
curl http://127.0.0.1:8001 | jq '.'
```

3. API 术语

3.1 API 资源类型和对象

Kubernetes API 根据资源进行操作。但什么是资源呢？

资源是 k8s 对象的唯一表示。每个资源都有自己的表示形式。这种表示形式称为资源类型 (kind)。

资源类型（kind）可以是 pods, configmaps, services 等。从底层代码中可以看出，不同种类的资源有不同的表示方式。

```yaml
# an example of an nginx pod resource
# kubectl run nginx --image=nginx --dry-run -o yaml > pod.yaml
# pod.yaml

apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  restartPolicy: Always

# an example of an nginx configmap resource
# kubectl create configmap nginx --dry-run -o yaml > configmap.yaml
# configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx
```

资源类型的实例列表称为集合。此外，资源类型的单个实例称为资源，并且通常也代表对象。

> 通过 API 创建的所有对象都有一个唯一的对象名称，以允许幂等创建和检索。

3.2 API 资源 动词

不同的资源类型公开不同的 CRUD 动词（创建、读取、更新、删除），这些动词可以在它们管理的对象上执行。几乎所有对象资源类型都支持标准 HTTP 动词 — GET、POST、PUT、PATCH 和 DELETE。

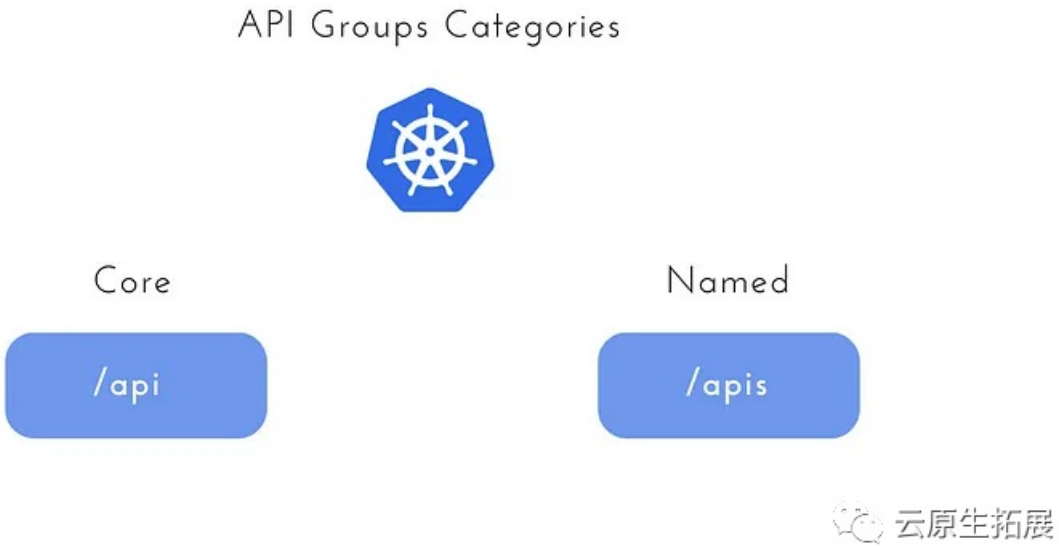
Kubernetes 还使用自己的动词，这些动词通常写成小写，以区别于 HTTP 动词： `list` 、 `get` 、 `watch` 、 `create` 、 `update` 、 `delete` 、 `create` 、 `patch` 。

- list ：返回资源集合。
- get ：返回单个资源
- watch ：实时连续 list 和 get 资源。它就像 Linux watch 命令。因此，在定义 RBAC 动词时，值得注意的是 watch 不能单独放置。它必须与 get 和 list 动词放在一起。如果您发送带有 ?watch 查询参数的 HTTP GET 请求，k8s 会将其称为 watch 而不是 get 。
- 对于 PUT 请求，Kubernetes 根据现有对象的状态在内部将它们分类为 create 或 update 。

3.3 API 组

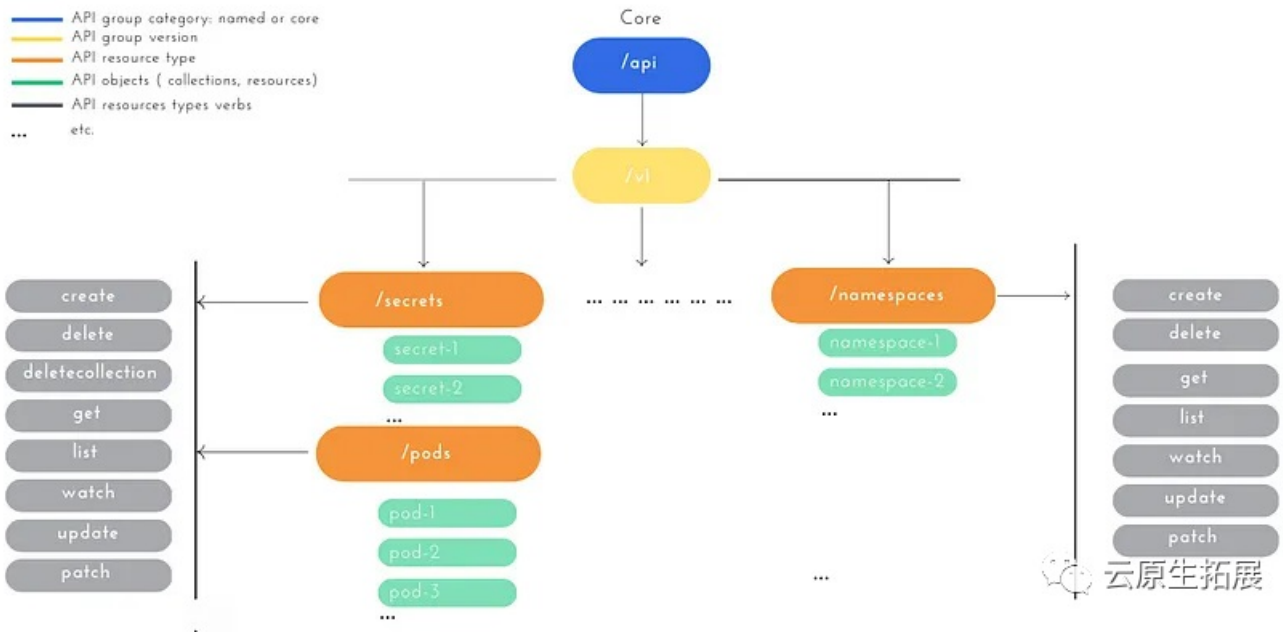
不同的资源类型属于不同的API组。 API 组在 REST 路径和序列化对象的 apiVersion 字段中指定。

API 组可分为核心组或命名组。



3.3.1 核心组

核心组位于 REST 路径 /api/v1 处。

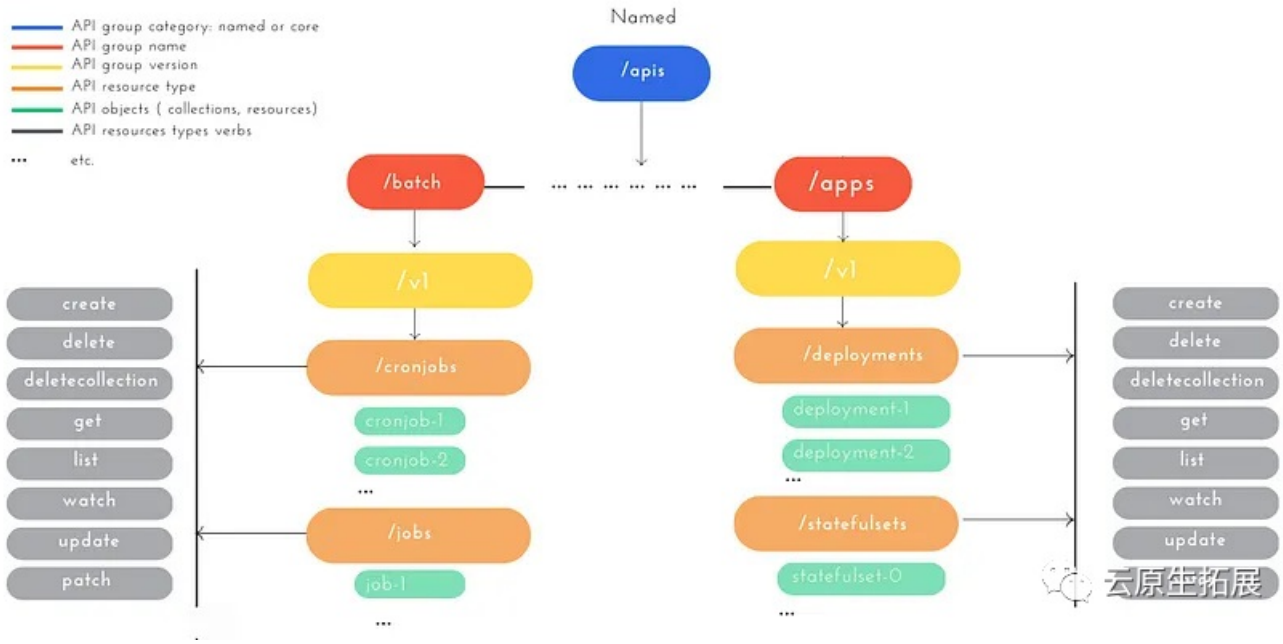


核心组被指定为 apiVersion 字段的一部分。例如： apiVersion: v1 。

```yaml
# an example of an nginx configmap resource
apiVersion: v1  # here
kind: ConfigMap
metadata:
  name: nginx
```

3.3.2 命名组

命名组位于 REST 路径 `/apis/$GROUP_NAME/$VERSION/$PLURALNAME`



命名组被指定为 apiVersion 字段的一部分，即 apiVersion: *GROUP_NAME*/ VERSION 。例如： apiVersion: batch/v1

```
- apiVersion: batch/v1 #here
  kind: CronJob
  metadata:
```

命名组中的每个 API 组可以有不同的版本。

- Alpha ：（例如 v1alpha1 ）此版本存在许多错误，因为尚未经过测试。默认情况下它是禁用的。 Alpha 功能随时可能更改或消失。
- Beta ：（例如 v1beta1 ）默认情况下启用此功能，因为它已经过测试。但它仍然存在一些错误。
- Stable ：（例如 v1 ）此版本是稳定的。默认情况下它是启用的。

> 核心组的资源类型始终在 v1 版本下。确保始终使用属于 stable API 组的对象。例如：我们有一个资源类型 CronJob 的对象，属于 API 组 batch ，分类在指定组下，并且有两个版本可用： v1 和 v1beta1 （如果可用）（默认情况下，k8s 将 v1 视为其首选版本）。

```
- apiVersion: batch/v1        # this is the perefered version
  kind: CronJob
  metadata:

...
- apiVersion: batch/v1alpha1  # this is an available version but not the most
                              # perefered
  kind: CronJob
  metadata:
```

API 调用检索

让我们列出 k8s 集群中可用的 k8s API

```
curl http://127.0.0.1:8001 | jq '.'


# OUTPUT :
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1",
    ...
    ...

# REMEMBER :
# core group:  /api/v1
# named group: /apis/$GROUP_NAME/$VERSION
# EXPLANATION :
# We can see that we have different resource types grouped into different
# groups, Some of these groups are made available under the core group and
# others under the named group and are available under a specific version.
# EXAMPLE :
# The resource group: batch is available under the named group (apis). it
# groups a certain resource types of version v1.
```

4.1 核心组

- 让我们探索一下 API core group

```
curl http://127.0.0.1:8001/api | jq '.'

# OUTPUT
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "X.X.X.X/x",
      "serverAddress": "X.X.X.X/X"
    }
  ]
}
# EXPLANATION
# we can see that the only version available for the resource types under
# this API GROUP is v1. Thus, all resources of these group are of version v1.
```

- 让我们在核心 API 组下列出版本 v1 的资源类型。

```
curl http://127.0.0.1:8001/api/v1 | jq '.resources[].name'

# OUTPUT :
...
"configmaps"
...
"pods"
"namespaces"
...
```

- 让我们列出资源类型 pods 下可用的集合或对象或资源

```
# get pods
curl http://127.0.0.1:8001/api/v1/pods | jq  .items[].metadata.name

# get information about a specific pod using its name
curl http://127.0.0.1:8001/api/v1/pods/ | jq  '.items[].metadata |  select(.name=="nginx-a1Gv0")'
```

- 让我们列出资源类型 namespace 下可用的集合或对象或资源

```
# get namespaces
curl http://127.0.0.1:8001/api/v1/namespaces | jq  .items[].metadata.name

# get information about a specific namespace using its name
 curl http://127.0.0.1:8001/api/v1/namespaces/proxy| jq .
# get a list of existing pods within the selected namespace: proxy
# you can see that we've used a combination of the namespaces and pods resource types
 curl http://127.0.0.1:8001/api/v1/namespaces/proxy/pods | jq .items[].metadata.name
# get information of a specific pod within the selected namespace : proxy
 curl http://127.0.0.1:8001/api/v1/namespaces/proxy/pods | jq  '.items[].metadata |  select(.name=="nginx-a1Gv0")'
```

从上面的示例可以看出，有关 pod nginx-a1Gv0 的信息是使用不同的 REST 路径获取的：

```
# get information of a specific pod within the selected namespace : proxy
curl http://127.0.0.1:8001/api/v1/namespaces/proxy/pods | jq  '.items[].metadata |  select(.name=="nginx-a1Gv0")'
# get information about a specific pod using its name
curl http://127.0.0.1:8001/api/v1/pods/ | jq  '.items[].metadata |  select(.name=="nginx-a1Gv0")'
```

4.2 命名组

- 让我们列出指定组下可用的 API 组

```
# REMEMBER :
# core group:  /api/v1
# named group: /apis/$GROUP_NAME/$VERSION
# Let's list the groups available under the named group.
curl http://127.0.0.1:8001/apis/| jq .groups[].name

# OUTPUT :
...
"apps"
...
"batch"
...
"storage.k8s.io"
...
# EXPLANATION :
# We can see that we have different API groups available under the named group.
```

API 组在特定版本下或在一组不同版本下可用，其中一个版本被指定为首选版本。因此，默认情况下在对象的 apiVersion 字段中使用。让我们探索一下 API 组的版本

```
curl http://127.0.0.1:8001/apis/batch/ | jq .

# OUTPUT
{
  "kind": "APIGroup",
  "apiVersion": "v1",
  "name": "batch",
  "versions": [
    {
      "groupVersion": "batch/v1",
      "version": "v1"
    }
  ],
  "preferredVersion": {
    "groupVersion": "batch/v1",
    "version": "v1"
  }
}
# EXPLANATION
# The api group batch avaialble under the named group has version v1 which
# is also the preferred version. Thus, resources types avaiable under the api
# Group batch are of version v1.
```

- 让我们列出版本 v1 的 API 组 batch 下可用的资源类型。

```
curl http://127.0.0.1:8001/apis/batch/v1 | jq .resources[].name


# OUTPUT
"cronjobs"
"cronjobs/status"
"jobs"
"jobs/status"
# EXPLANATION
# resources types available under the API group batch of type v1 are cronjobs
# and jobs. These resources can be accessed using their REST PATH /cronjobs
# and can be explored further using their REST PATH and REST SUBPATH
# /cronjobs/status
```

- 让我们列出在 v1 版本的 API 组 batch 的资源类型 cronjobs 下可用的集合、资源和对象。

```
curl http://127.0.0.1:8001/apis/batch/v1/cronjobs | jq .items[].metadata.name


# OUTPUT
"daily-cronjob-database-backuper"
"..."
"..."
# Explanation
# These are the list of running cronjobs objects
```

4.3 资源类型定义

让我们探讨一下 API 资源类型是如何定义的。

```
# Let's explore the api resources type available under the core group


# List the resources type
 curl http://127.0.0.1:8001/api/v1/| jq .
# Select only resources of type pods
 curl http://127.0.0.1:8001/api/v1/ | jq '.resources[] | select(.name=="pods")'
# OUTPUT
{
  "name": "pods",
  "singularName": "",
  "namespaced": true,
  "kind": "Pod",
  "verbs": [
    "create",
    "delete",
    "deletecollection",
    "get",
    "list",
    "patch",
    "update",
    "watch"
  ],
  "shortNames": [
    "po"
  ],
  "categories": [
    "all"
  ],
  "storageVersionHash": "xxxxxx"
}
```

从上面的输出我们可以得出每个资源类型的定义如下：

- a plural name 在 name 字段中定义。 plural name 可用作 HTTP 端点。我们通过使用可用的 k8s 资源类型 plural names 进行 API 调用。
- a kind 在 kind 字段中定义。它定义了资源类型
- a list of verbs 在 verbs 字段中定义。它定义了可以对该资源类型管理的集合、对象或资源执行的操作列表。
- a list of short names 在 shotNames 字段中定义。它允许使用短名称查询资源类型。例如：

```
kubectl get po # instead of: kubectl get pods
kubectl get deploy # intead of : kubectl get deployments
```

最后， nameSpaced 字段定义资源类型是命名空间还是集群范围。

> 例如：资源类型 pod 的对象是命名空间的。要创建或管理 pods 对象，我们应该指定一个命名空间，否则 k8s 会将它们视为 default 命名空间的一部分。

```
资源类型 pod 的对象是命名空间的。要创建或管理 pods 对象，我们应该指定一个命名空间，否则 k8s 会将它们视为 default 命名空间的一部分。
```

资源类型 pv [持久存储] 的对象属于集群范围。要创建或管理 pvs ， 我们不需要指定命名空间。 k8s 将它们视为集群范围的资源。

```
kubectl get pv
```

检查 API 资源类型名称、短名称、版本、种类以及它们是否具有命名空间的简单方法。您可以使用以下命令。它代表我们与 API 服务器执行 API 调用以获取这些信息。

```
kubectl api-resources

# watch the API Calls the kubectl establish with the API-server
kubectl api-resources -v 10
```

5. API调用高级查询

5.1 将 kubectl 与 jsonPath 和 jq 一起使用

```
# Brief Resume of kubectl JsonPath and jq Operators: more can be found in [5]

- {"\n"}          : the escape sequence
- {"\t"}          : the space sequence
- range, end      : to iterate a list of elements
- [,]             : the union operator. for example, get two elements [a,b]
- [start:end:step]: subscript operator
- *               : wildcard. Get all objects
- ..              : recursive descent
- . or []         : child operator
- select ()       : to filter elements
- keys            : to list json file keys for further queries
```

- 获取可能的json属性来查询pod k8s资源

```
kubectl get pod -o json | jq 'keys'

# OUTPUT
[
  "apiVersion",
  "items",
  "kind",
  "metadata"
]
# Explanation
# We can see that we can query the above attributes, for example:
# If we want to list pods within the default namespace:

kubectl get pod -o json | jq '.items[]'

# Let's get the possible attributes to query for each pod within the
# list of items

kubectl get pod -o json | jq '.items[]'  | jq 'keys'
kubectl get pod -o json | jq '.items[].metadata' | jq 'keys'
kubectl get pod -o json | jq '.items[].metadata.name'
```

- 获取默认命名空间内 pod 的名称

```
# option 1
kubectl get pod -o=jsonpath='{.items[*].metadata.name}'

# option 2
kubectl get pod -o=jsonpath='{range .items[*]}{.metadata.name}{"\n"}{end}'
```

- 获取默认命名空间内 nginx pod 的容器和镜像名称

```
kubectl get pods -o json | jq '.items[] | select (.metadata.name == "nginx") | [.spec.containers[].name, .spec.containe
```

- 获取默认命名空间中名称中包含 nginx 模式的 pod 的容器和镜像名称。

```
kubectl get pods -o json | jq '.items[] | select (.metadata.name | test ("nginx")) | [.spec.containers[].name, .spec.cont
```

- 获取默认命名空间内的 Pod 名称及其相应的开始日期

```
kubectl get pod -o json | jq '.items[] | [.metadata.name, .status.startTime]'
```

- 根据开始日期排序 pods

```
kubectl get pod -o json | jq '.items[] | .status.startTime | sort_by(.startTime)'

 # ERROR
 Cannot iterate over string.


 # COMMENT
 The idea here is just to show the sort_by function usage.
```

- 对于默认命名空间中的每个 Pod，获取其名称、CPU、RAM 要求，并得出其服务质量 (qos)

```
kubectl get pod -o json | jq '.items[] | [.metadata.name, .spec.containers[].resources, .status.qosClass]'
```

```
# Additional Queries

kubectl get pod -o json | jq .items[].spec.containers[].resources
kubectl get pod -o json | jq .items[].spec.containers[].resources.requests
kubectl get pod -o json | jq .items[].spec.containers[].resources.requests.cpu
kubectl get pod -o json | jq .items[].spec.containers[].resources.requests.memory
kubectl get pod -o json | jq .items[].spec.containers[].resources.limits
kubectl get pod -o json | jq .items[].spec.containers[].resources.limits.cpu
kubectl get pod -o json | jq .items[].spec.containers[].resources.limits.memory
kubectl get pod -o json | jq .items[].status.qosClass
```

总结

本文的主要目标是通过实际示例更深入地了解 Kubernetes API 功能。这是通过对 k8s API 进行总体概述，然后通过使用 curl 和 kubectl 命令的不同示例获得更多实用信息来实现的。