**Kubernetes对象**

**理解Kubernetes对象**

Kubernetes对象是系统中的持久化实体，Kubernetes使用这些实体来表示集群的状态，可以描述的有：

* 什么容器应用在运行，且在哪个节点上。
* 应用可以访问的资源
* 应用行为的政策，如重启，更新和容错机制

一个对象是意图的记录，一旦创建对象，Kubernetes系统会保证该对象存在。通过创建一个对象，你等于告诉Kubernetes你想要你的集群工作负载看起来是什么，这将是集群的期望状态。

与对象协作时，都需要使用Kubernetes API，当使用kubectl命令行接口，会调用所需的Kubernetes API，同时可以在自定义程序中直接使用Kubernetes API。

**Object Spec and Status**

每个Kubernetes对象包含两个嵌套域用于管理对象配置：

Spec，是必须提供的，描述你的对象期望状态，

Status，描述对象的实际状态，由Kubernetes提供并更新。

在任意时间，控制面板都会管理对象实际状态来匹配设置的期望状态。

例如，Kubernetes Deployment是一个对象，代表一个应用在集群中运行。当你创建Deployment时，你可能设置部署的spec需要3个副本运行。Kubernetes识别到spec并启动3个实例，并更新status来匹配spec，如果任何一个实例宕掉，Kubernetes会因为status和spec不统一而响应，在这里，会启动一个替换实例。

**Describing a Kubernetes Object**

除了spec必须提供，其他的一些基本信息也是必须的。当使用Kubernetes API创建对象时，API请求都必须包含将信息以JSON格式封装仅请求体，大部分时间，提供给kubectl的信息放在.yaml文件中，kubectl会转换为JSON格式。

一个yaml文件样例：

**apiVersion: apps/v1 *# for versions before 1.9.0 use apps/v1beta2***

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**spec:**

**selector:**

**matchLabels:**

**app: nginx**

**replicas: 2 *# tells deployment to run 2 pods matching the template* template:**

**metadata:**

**labels:**

**app: nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx:1.7.9**

**ports:**

**- containerPort: 80**

使用yaml文件创建对象可以使用kubectl apply命令，示例如下：

**kubectl apply -f https://k8s.io/examples/application/deployment.yaml --record**

输出如下：

**deployment.apps/nginx-deployment created**

**Required Fields**

在yaml文件中，有一些必要的域需要赋值：

* apiVersion，Kubernetes API的版本号，用于创建对象
* Kind，什么类型的对象
* Metadata，用于辨别唯一性的数据，包括name，UID，和可选性namespace

同时必须提供spec域属性，每个对象的spec准确格式是不一样的，也包含特定于该对象的嵌套域。

**Kubernetes对象管理**

每个对象只能被一种技术管理，对同一对象做混合匹配技术会导致未定义行为。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **技术** | **操作对象** | **推荐环境** | **支持写入** | **学习曲线** |
| 命令行 | 活跃对象 | 开发环境 | 1+ | 低 |
| 预定义对象配置 | 独立文件 | 生产环境 | 1 | 中 |
| 声明对象配置 | 文件路径 | 生产环境 | 1+ | 高 |

**Imperative commands（命名行）**

当使用命令行时，用户直接操作存活对象，提交操作到kubectl命令作为参数或标志。

这是启动或执行一次性任务的最简单的方式，因为直接操作的原因，不会提供之前配置的历史记录。

优点：

* 命令行简单，容易学习和记忆
* 仅一步操作即可修改集群

缺点：

* 命令行不与审核过程集成
* 命令行不提供变更有关的审计跟踪
* 命令行不提供存活资源以外的记录源
* 命令行不提供用于创建新对象的模板。

**Imperative object configuration（预设对象配置）**

在预设配置里，kubectl命令指定了操作，可选项和至少一个文件名。文件必须包含完整的对象定义，以yaml或json格式。

优点：

* 对象配置可以存储在资源控制系统中。
* 对象配置可以与过程集成，比如提交前的变更预览和审计追踪
* 对象配置可以为创建新对象提供模板

缺点：

* 需要对象格式的基本理解
* 需要编写yaml文件这一额外步骤

与声明式对象配置对比的优点：

* 预设对象配置更简单且更易理解
* 在1.5版本中，预设对象配置更成熟

与声明式对象配置对比的缺点：

* 预设对象配置在文件上表现最好，而非目录
* 对存活对象的更新必须体现在配置文件上，不然在下一次替换中会丢失更新

**Declarative object configuration（声明式对象配置）**

使用声明式对象配置时，用户对本地存储的配置文件进行操作，但用户不会定义对文件执行的操作。kubectl会自动检测每个对象的创建，更新和删除操作。 这使得能够处理目录，其中可能需要不同对象的不同操作。

声明性对象配置保留其他编写者所做的更改，即使更改未合并回对象配置文件也是如此。 这可以通过使用patch API操作来仅写入观察到的差异，而不是使用replace API操作来替换整个对象配置。

对比预设对象配置的优点：

* 对存活对象的修改会被保留，即使没有整合到配置文件中。
* 对操作目录有更好的支持且会自动检测每个对象的操作类型。

对比预设对象配置的缺点：

* 更难以定位问题和理解非预期结果。
* 使用diff的部分更新会创建复杂的合并和修补操作。

**Names（名称）**

所有Kubernetes REST API中的对象，都有唯一性的Name和UID。

对于非唯一，用户提供的属性，Kubernetes提供有labels（标签）和annotation（注解）

**Names**

由客户端提供的在资源URL中代表一个对象，比如/api/v1/pods/some-name。

仅提供的类型对象才可以有名称，若删除该对象，则可以创建一个拥有同样名称的新对象。

按照惯例，Kubernetes资源的名称最大长度为253，有小写字母，-，.，但某些资源限制更具体。

**UIDs**

由Kubernetes生成的唯一字符串。

在Kubernetes集群的整个生命周期中创建的每个对象都具有不同的UID。 它旨在区分类似实体的历史事件

**Namespaces（命名空间）**

Kubernetes支持在同一个物理集群上使用多个虚拟集群，这些虚拟集群被称为命名空间。

**什么时候使用命名空间**

命名空间旨在用于多个用户分布在多个团队或项目中的环境中。 对于具有几个到几十个用户的集群，您根本不需要创建或考虑名称空间。 当您需要它们提供的功能时，请开始使用命名空间。

命名空间提供名称范围。 资源名称在名称空间中必须是唯一的，而不是跨名称空间。 命名空间不能彼此嵌套，并且每个Kubernetes资源只能位于一个命名空间中。

命名空间是一种在多个用户之间划分群集资源的方法（通过资源配额）。

在Kubernetes的未来版本中，默认情况下，同一名称空间中的对象将具有相同的访问控制策略。

没有必要使用多个名称空间来分隔略有不同的资源，例如同一软件的不同版本：使用标签来区分同一名称空间中的资源

**使用命名空间**

创建和删除命名空间的描述见[Admin Guide documentation for namespaces](https://kubernetes.io/docs/admin/namespaces).

**查看命名空间**

可以列出当前集群使用的命名空间：kubectl get namespace

Kubernetes会以三个初始化命名空间启动：

Default，未指明命名空间对象的默认命名空间

Kube-system，由Kubernetes系统创建的对象所属空间

Kube-public，自动创建且所有用户可读，此命名空间主要用于集群使用，用于某些资源在整个集群中需要公开可见且公开可读取。 此命名空间的公共方面只是一个约定，而不是一个要求。

**设置请求的命名空间**

为当前请求设置命名空间，使用--namespace标签

**设置偏好的命名空间**

可以在该上下文中为所有后续kubectl命令永久保存命名空间。

**命名空间和DNS**

创建服务时，它会创建相应的DNS条目。 此条目的格式为<service-name>。<namespace-name> .svc.cluster.local，这意味着如果容器只使用<service-name>，它将解析为名称空间本地的服务。 这对于在多个名称空间（如开发，分段和生产）中使用相同的配置非常有用。 如果要跨命名空间访问，则需要使用完全限定的域名（FQDN）

**不是所有对象都在命名空间**

命名空间资源本身不在命名空间中，此外，低阶资源如node和persistentVolumes，也不在任何命名空间。使用如下指令查询不在命名空间中的资源：

***# In a namespace***

**kubectl api-resources --namespaced=true**

***# Not in a namespace***

**kubectl api-resources --namespaced=false**

**Labels and Selectors（标签和选择器）**

标签是附加到对象（例如pods）的ley/value。

标签旨在用于指定对用户有意义且相关的对象的标识属性，但不直接暗示核心系统的语义。

标签可用于组织和选择对象的子集。

标签可以在创建时附加到对象，随后可以随时添加和修改。

每个对象都可以定义一组key/value标签。 每个Key对于给定对象必须是唯一的。

标签允许高效的查询和监视，非常适合在UI和CLI中使用。非识别信息应使用注解记录.

**Motivation（动机）**

标签使用户能够以松散耦合的方式将他们自己的组织结构映射到系统对象，而无需客户端存储这些映射。

服务部署和批处理流水线通常是多维实体（例如，多个分区或部署，多个释放轨道，多个层，每层多个微服务）。 管理通常需要交叉操作，这打破了严格的层次表示的封装，特别是由基础设施而不是用户确定的严格的层次结构。

**Syntax and character set（语法和字符设置）**

标签是key/value。有效标签key有两个部分：可选前缀和名称，用斜杠（/）分隔。名称是必需的，必须是63个字符或更少，以字母数字字符（[a-z0-9A-Z]）开头和结尾，带有破折号（ - ），下划线（\_），点（.）和字母数字之间。前缀是可选的。如果指定，前缀必须是DNS子域：由点（.）分隔的一系列DNS标签，总共不超过253个字符，后跟斜杠（/）。

如果省略前缀，则假定标签Key对用户是私有的。向最终用户对象添加标签的自动系统组件（例如，kube-scheduler，kube-controller-manager，kube-apiserver，kubectl或其他第三方自动化）必须指定前缀。

kubernetes.io/和k8s.io/前缀保留给Kubernetes核心组件。

有效标签值必须为63个字符或更少，并且必须为空或者以字母（ - ），下划线（\_），点（。）和字母数之间。

**Label Selectors（标签选择器）**

与名称和UID不同，标签不提供唯一性。 通常，我们希望许多对象携带相同的标签。

通过标签选择器，客户端/用户可以识别一组对象。 标签选择器是Kubernetes中的核心分组原语。

API目前支持两种类型的选择器：基于等式和基于集合。 标签选择器可以由逗号分隔的多个要求组成。 在多个要求的情况下，必须满足所有要求，因此逗号分隔符充当逻辑AND（&&）运算符。

空或非指定选择器的语义取决于上下文，使用选择器的API类型应记录它们的有效性和含义。

对于某些API类型（例如ReplicaSet），两个实例的标签选择器不能在命名空间内重叠，或者控制器可以将其视为冲突的指令，并且无法确定应该存在多少副本.

**基于平等的要求**

基于平等或不平等的要求允许按标签键和值进行过滤。 匹配对象必须满足所有指定的标签约束，尽管它们也可能有其他标签。 允许三种运算符=，==，！=。 前两个代表平等（简单地说是同义词），而后者代表不平等。

**基于集合的要求**

基于集合的标签要求允许根据一组值过滤密钥。 支持三种运算符：in，notin和exists（仅密钥标识符）。

**API（略，暂时不记录）**

**Annotations（注解）**

可以使用Kubernetes注释将任意非标识元数据附加到对象。 工具和库等客户端可以检索此元数据。

**Attaching metadata to objects（附加元数据）**

可以使用标签或注解将元数据附加到Kubernetes对象。 标签可用于选择对象和查找满足特定条件的对象集合。 相反，注解不用于识别和选择对象。 注解中的元数据可以是小的或大的，结构化的或非结构化的，并且可以包括标签不允许的字符。注解和标签一致，是key/value结构。

**Syntax and character set（语法和字符设置）**

注解是key/value。 有效的注解key有两个部分：可选的前缀和名称，用斜杠（/）分隔。 名称是必需的，必须是63个字符或更少，以字母数字字符（[a-z0-9A-Z]）开头和结尾，带有（ - ），下划线（\_），点（.）和字母数字之间。 前缀是可选的。 如果指定，前缀必须是DNS子域：由点（.）分隔的一系列DNS标签，总共不超过253个字符，后跟斜杠（/）。

如果省略前缀，则假定注解key对用户是私有的。 向最终用户对象添加注解的自动系统组件（例如，kube-scheduler，kube-controller-manager，kube-apiserver，kubectl或其他第三方自动化）必须指定前缀。

kubernetes.io/和k8s.io/前缀保留给Kubernetes核心组件。

**Field Selectors（字段选择器）**

字段选择器允许您根据一个或多个资源字段的值选择Kubernetes资源。

字段选择器本质上是资源过滤器。 默认情况下，不应用选择器/过滤器，这意味着将选择指定类型的所有资源。

**Supported fields（支持的字段）**

支持的字段选择器因Kubernetes资源类型而异。 所有资源类型都支持metadata.name和metadata.namespace字段。 使用不受支持的字段选择器会产生错误。

**Supported Operators（支持的操作）**

可以将=，==和！=运算符与字段选择器一起使用（=和==表示相同的事情）。

**Chained selectors（链式选择器）**

与标签和其他选择器一样，字段选择器可以作为逗号分隔列表链接在一起。

**Multiple resource types（多种资源类型）**

可以跨多种资源类型使用字段选择器。