



下载APP

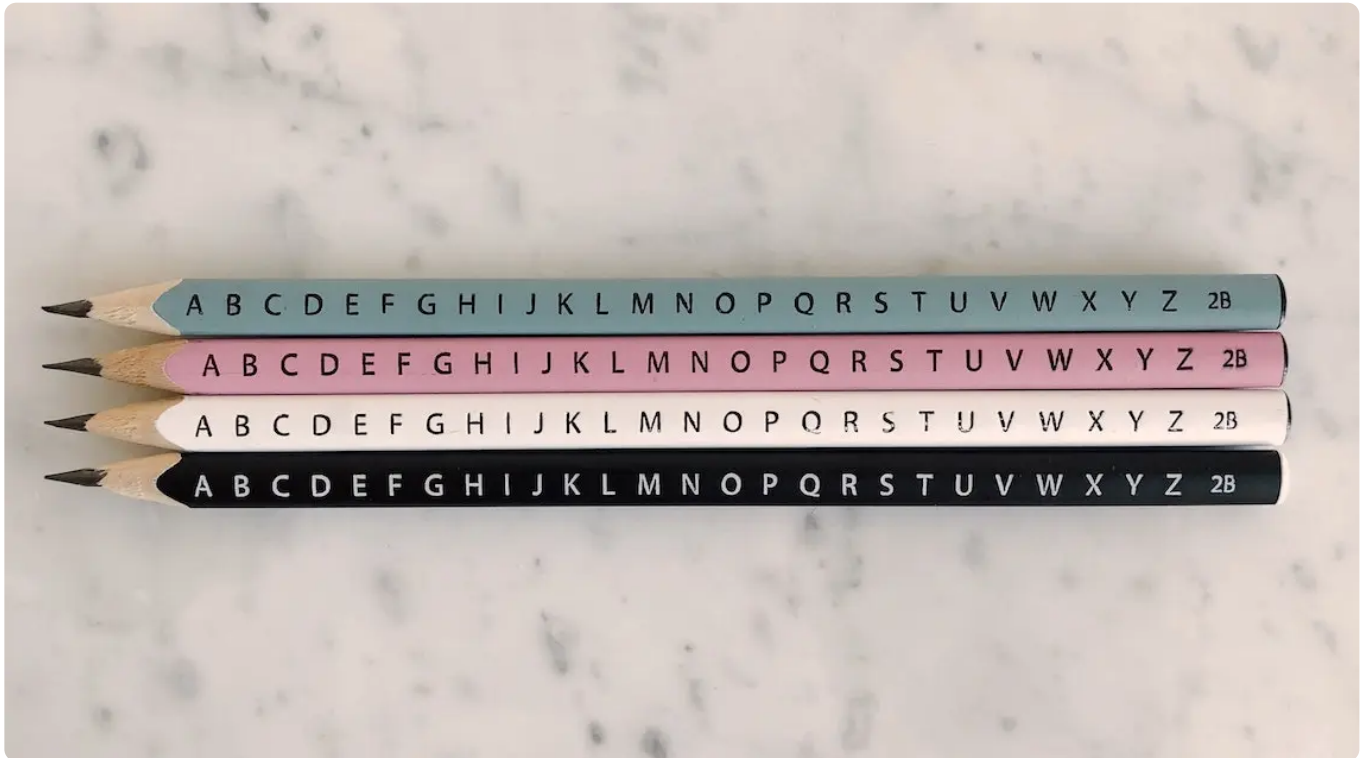


47 | 如何编写Kubernetes资源定义文件？

2021-09-14 孔令飞

《Go 语言项目开发实战》

课程介绍 >



讲述：孔令飞

时长 12:32 大小 11.49M



你好，我是孔令飞。

在接下来的 48 讲，我会介绍如何基于腾讯云 EKS 来部署 IAM 应用。EKS 其实是一个标准的 Kubernetes 集群，在 Kubernetes 集群中部署应用，需要编写 Kubernetes 资源的 YAML (Yet Another Markup Language) 定义文件，例如 Service、Deployment、ConfigMap、Secret、StatefulSet 等。

这些 YAML 定义文件里面有很多配置项需要我们去配置，其中一些也比较难理解。为了你在学习下一讲时更轻松，这一讲我们先学习下如何编写 Kubernetes YAML 文件。



为什么选择 YAML 格式来定义 Kubernetes 资源？

首先解释一下，我们为什么使用 YAML 格式来定义 Kubernetes 的各类资源呢？这是因为 YAML 格式和其他格式（例如 XML、JSON 等）相比，不仅能够支持丰富的数据，而且结构清晰、层次分明、表达性极强、易于维护，非常适合拿来供开发者配置和管理 Kubernetes 资源。

其实 Kubernetes 支持 YAML 和 JSON 两种格式，JSON 格式通常用来作为接口之间消息传递的数据格式，YAML 格式则用于资源的配置和管理。YAML 和 JSON 这两种格式是可以相互转换的，你可以通过在线工具 [@json2yaml](#)，来自动转换 YAML 和 JSON 数据格式。

例如，下面是一个 YAML 文件中的内容：

[复制代码](#)

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: iam-apiserver
5  spec:
6    clusterIP: 192.168.0.231
7    externalTrafficPolicy: Cluster
8    ports:
9      - name: https
10      nodePort: 30443
11      port: 8443
12      protocol: TCP
13      targetPort: 8443
14    selector:
15      app: iam-apiserver
16    sessionAffinity: None
17    type: NodePort
```

它对应的 JSON 格式的文件内容为：

[复制代码](#)

```
1  {
2    "apiVersion": "v1",
3    "kind": "Service",
4    "metadata": {
5      "name": "iam-apiserver"
6    },
7    "spec": {
8      "clusterIP": "192.168.0.231",
```

```
9     "externalTrafficPolicy": "Cluster",
10     "ports": [
11     {
12         "name": "https",
13         "nodePort": 30443,
14         "port": 8443,
15         "protocol": "TCP",
16         "targetPort": 8443
17     }
18 ],
19     "selector": {
20         "app": "iam-apiserver"
21     },
22     "sessionAffinity": "None",
23     "type": "NodePort"
24 }
25 }
```

我就是通过json2yaml在线工具，来转换 YAML 和 JSON 的，如下图所示：

| JSON to YAML | YAML ✓ | JSON |
|---|---|--|
| Convert YAML to JSON online YAML vs JSON determine which format is right for you JSON stands for javascript object notation <ul style="list-style-type: none">records separated by commaskeys & strings wrapped by double quotesgood choice for data transport YAML stands for YAML ain't markup language and is a superset of JSON <ul style="list-style-type: none">lists begin with a hyphendependent on whitespace / indentationbetter suited for configuration than json privacy notice <ul style="list-style-type: none">analytics and adsense are run on the site which utilize cookieshere is a version of the site without adsense and analyticssubmissions are not stored or loggedcontact: json2yaml@protonmail.com | 1 apiVersion: v1 2 kind: Service 3 metadata: 4 name: iam-apiserver 5 spec: 6 clusterIP: 192.168.0.231 7 externalTrafficPolicy: Cluster 8 ports: 9 - name: https 10 nodePort: 30443 11 port: 8443 12 protocol: TCP 13 targetPort: 8443 14 selector: 15 app: iam-apiserver 16 sessionAffinity: None 17 type: NodePort | 1 { 2 "apiVersion": "v1", 3 "kind": "Service", 4 "metadata": { 5 "name": "iam-apiserver" 6 }, 7 "spec": { 8 "clusterIP": "192.168.0.231", 9 "externalTrafficPolicy": "Cluster", 10 "ports": [11 { 12 "name": "https", 13 "nodePort": 30443, 14 "port": 8443, 15 "protocol": "TCP", 16 "targetPort": 8443 17 } 18], 19 "selector": { 20 "app": "iam-apiserver" 21 }, 22 "sessionAffinity": "None", 23 "type": "NodePort" 24 } 25 } |

在编写 Kubernetes 资源定义文件的过程中，如果因为 YAML 格式文件中的配置项缩进太深，导致不容易判断配置项的层级，那么，你就可以将其转换成 JSON 格式，通过 JSON 格式来判断配置型的层级。

如果想学习更多关于 YAML 的知识，你可以参考 [YAML 1.2 \(3rd Edition\)](#)。这里，可以先看看我整理的 YAML 基本语法：

属性和值都是大小写敏感的。

使用缩进表示层级关系。

禁止使用 Tab 键缩进，只允许使用空格，建议两个空格作为一个层级的缩进。元素左对齐，就说明对齐的两个元素属于同一个级别。

使用 # 进行注释，直到行尾。

key: value格式的定义中，冒号后要有一个空格。

短横线表示列表项，使用一个短横线加一个空格；多个项使用同样的缩进级别作为同一列表。


使用 --- 表示一个新的 YAML 文件开始。

现在你知道了，Kubernetes 支持 YAML 和 JSON 两种格式，它们是可以相互转换的。但鉴于 YAML 格式的各项优点，我建议你使用 YAML 格式来定义 Kubernetes 的各类资源。

Kubernetes 资源定义概述

Kubernetes 中有很多内置的资源，常用的资源有 Deployment、StatefulSet、ConfigMap、Service、Secret、Nodes、Pods、Events、Jobs、DaemonSets 等。除此之外，Kubernetes 还有其他一些资源。如果你觉得 Kubernetes 内置的资源满足不了需求，还可以自定义资源。

Kubernetes 的资源清单可以通过执行以下命令来查看：

 复制代码

```
1 $ kubectl api-resources
2 NAME                                SHORTNAMES  APIVERSION
3 bindings                            v1
4 componentstatuses                  cs          v1
5 configmaps                        cm          v1
6 endpoints                         ep          v1
7 events                            ev          v1
```

上述输出中，各列的含义如下。

NAME：资源名称。

SHORTNAMES：资源名称简写。

APIVERSION：资源的 API 版本，也称为 group。

NAMESPACED：资源是否具有 Namespace 属性。

KIND：资源类别。

这些资源有一些共同的配置，也有一些特有的配置。这里，我们先来看下这些资源共同的配置。

下面这些配置是 Kubernetes 各类资源都具备的：

[复制代码](#)

```
1 ---
2 apiVersion: <string> # string类型，指定group的名称，默认为core。可以使用 `kubectl ap
3 kind: <string> # string类型，资源类别。
4 metadata: <Object> # 资源的元数据。
5   name: <string> # string类型，资源名称。
6   namespace: <string> # string类型，资源所属的命名空间。
7   labels: < map[string]string> # map类型，资源的标签。
8   annotations: < map[string]string> # map类型，资源的标注。
9   selfLink: <string> # 资源的 REST API路径，格式为：/api/<group>/namespaces/<name:
10 spec: <Object> # 定义用户期望的资源状态 (disired state)。
11 status: <Object> # 资源当前的状态，以只读的方式显示资源的最近状态。这个字段由kubernetes
```

你可以通过 `kubectl explain <object>` 命令来查看 Object 资源对象介绍，并通过 `kubectl explain <object1>.<object2>` 来查看<object1>的子对象<object2>的资源介绍，例如：

[复制代码](#)

```
1 $ kubectl explain service
2 $ kubectl explain service.spec
3 $ kubectl explain service.spec.ports
```

Kubernetes 资源定义 YAML 文件，支持以下数据类型：

string，表示字符串类型。

object，表示一个对象，需要嵌套多层字段。

map[string]string，表示由 key:value 组成的映射。

[]string，表示字符串列表。

[]object , 表示对象列表。

boolean , 表示布尔类型。


integer , 表示整型。

常用的 Kubernetes 资源定义

上面说了, Kubernetes 中有很多资源, 其中 Pod、Deployment、Service、ConfigMap 这 4 类是比较常用的资源, 我来一个个介绍下。

Pod 资源定义

下面是一个 Pod 的 YAML 定义:

 复制代码

```
1 apiVersion: v1    # 必须 版本号, 常用v1  apps/v1
2 kind: Pod         # 必须
3 metadata:         # 必须, 元数据
4   name: string    # 必须, 名称
5   namespace: string # 必须, 命名空间, 默认上default, 生产环境为了安全性建议新建命名空间分
6   labels:         # 非必须, 标签, 列表值
7     - name: string
8   annotations:    # 非必须, 注解, 列表值
9     - name: string
10 spec:             # 必须, 容器的详细定义
11   containers:     #必须, 容器列表,
12     - name: string      #必须, 容器1的名称
13       image: string     #必须, 容器1所用的镜像
14       imagePullPolicy: [Always|Never|IfNotPresent] #非必须, 镜像拉取策略, 默认是Al
15       command: [string] # 非必须 列表值, 如果不指定, 则是一镜像打包时使用的启动命令
16       args: [string] # 非必须, 启动参数
17       workingDir: string # 非必须, 容器内的工作目录
18       volumeMounts: # 非必须, 挂载到容器内的存储卷配置
19         - name: string # 非必须, 存储卷名字, 需与【@1】处定义的名字一致
20           readOnly: boolean #非必须, 定义读写模式, 默认是读写
21   ports: # 非必须, 需要暴露的端口
22     - name: string # 非必须 端口名称
23       containerPort: int # 非必须 端口号
24       hostPort: int # 非必须 宿主机需要监听的端口号, 设置此值时, 同一台宿主机不能存z
25       proctocol: [tcp|udp] # 非必须 端口使用的协议, 默认是tcp
26   env: # 非必须 环境变量
27     - name: string # 非必须 , 环境变量名称
28       value: string # 非必须, 环境变量键值对
29   resources: # 非必须, 资源限制
30     limits: # 非必须, 限制的容器使用资源的最大值, 超过此值容器会推出
31     cpu: string # 非必须, cpu资源, 单位是core, 从0.1开始
```

```


32     memory: string 内存限制, 单位为MiB, GiB
33     requests: # 非必须, 启动时分配的资源
34     cpu: string
35     memory: string
36     livenessProbe: # 非必须, 容器健康检查的探针探测方式
37     exec: # 探测命令
38     command: [string] # 探测命令或者脚本
39     httpGet: # httpGet方式
40     path: string # 探测路径, 例如 http://ip:port/path
41     port: number
42     host: string
43     scheme: string
44     httpHeaders:
45     - name: string
46     value: string
47     tcpSocket: # tcpSocket方式, 检查端口是否存在
48     port: number
49     initialDelaySeconds: 0 #容器启动完成多少秒后的再进行首次探测, 单位为s
50     timeoutSeconds: 0 #探测响应超时的时间, 默认是1s, 如果失败, 则认为容器不健康, 4
51     periodSeconds: 0 # 探测间隔时间, 默认是10s
52     successThreshold: 0 #
53     failureThreshold: 0
54     securityContext:
55     privileged: false
56     restartPolicy: [Always|Never|OnFailure] # 容器重启的策略,
57     nodeSelector: object # 指定运行的宿主机
58     imagePullSecrets: # 容器下载时使用的Secrets名称, 需要与volumes.secret中定义
59     - name: string
60     hostNetwork: false
61     volumes: ## 挂载的共享存储卷类型
62     - name: string # 非必须, 【@1】
63     emptyDir: {}
64     hostPath:
65     path: string
66     secret: # 类型为secret的存储卷, 使用内部的secret内的items值作为环境变量
67     secretName: string
68     items:
69     - key: string
70     path: string
71     configMap: ## 类型为configMap的存储卷
72     name: string
73     items:
74     - key: string
75     path: string

```

Pod 是 Kubernetes 中最重要的资源, 我们可以通过 Pod YAML 定义来创建一个 Pod, 也可以通过 DaemonSet、Deployment、ReplicaSet、StatefulSet、Job、CronJob 来创建 Pod。

Deployment 资源定义

Deployment 资源定义 YAML 文件如下：

 复制代码

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    labels: # 设定资源的标签
5      app: iam-apiserver
6    name: iam-apiserver
7    namespace: default
8  spec:
9    progressDeadlineSeconds: 10 # 指定多少时间内不能完成滚动升级就视为失败，滚动升级自动回滚
10   replicas: 1 # 声明副本数，建议 >= 2
11   revisionHistoryLimit: 5 # 设置保留的历史版本个数，默认是10
12   selector: # 选择器
13     matchLabels: # 匹配标签
14       app: iam-apiserver # 标签格式为key: value对
15   strategy: # 指定部署策略
16     rollingUpdate:
17       maxSurge: 1 # 最大额外可以存在的副本数，可以为百分比，也可以为整数
18       maxUnavailable: 1 # 表示在更新过程中能够进入不可用状态的 Pod 的最大值，可以为百分比
19       type: RollingUpdate # 更新策略，包括：重建(Recreate)、RollingUpdate(滚动更新)
20   template: # 指定Pod创建模板。注意：以下定义为Pod的资源定义
21     metadata: # 指定Pod的元数据
22       labels: # 指定Pod的标签
23         app: iam-apiserver
24     spec:
25       affinity:
26         podAntiAffinity: # Pod反亲和性，尽量避免同一个应用调度到相同Node
27         preferredDuringSchedulingIgnoredDuringExecution: # 软需求
28         - podAffinityTerm:
29             labelSelector:
30               matchExpressions: # 有多个选项，只有同时满足这些条件的节点才能运行 Pod
31                 - key: app
32                   operator: In # 设定标签键与一组值的关系，In、NotIn、Exists、DoesNotExist
33                 values:
34                   - iam-apiserver
35             topologyKey: kubernetes.io/hostname
36             weight: 100 # weight 字段值的范围是1-100。
37     containers:
38       - command: # 指定运行命令
39         - /opt/iam/bin/iam-apiserver # 运行参数
40         - --config=/etc/iam/iam-apiserver.yaml
41         image: ccr.ccs.tencentyun.com/lkccc/iam-apiserver-amd64:v1.0.6 # 镜像名
42         imagePullPolicy: Always # 镜像拉取策略。IfNotPresent：优先使用本地镜像；Never：从不使用本地镜像
43         # lifecycle: # kubernetes支持postStart和preStop事件。当一个容器启动后，Kubernetes会执行postStart事件，容器启动失败会执行preStop事件
44         name: iam-apiserver # 容器名称，与应用名称保持一致
45         ports: # 端口设置

```



```
46 - containerPort: 8443 # 容器暴露的端口
47   name: secure # 端口名称
48   protocol: TCP # 协议, TCP和UDP
49 livenessProbe: # 存活检查, 检查容器是否正常, 不正常则重启实例
50   httpGet: # HTTP请求检查方法
51     path: /healthz # 请求路径
52     port: 8080 # 检查端口
53     scheme: HTTP # 检查协议
54   initialDelaySeconds: 5 # 启动延时, 容器延时启动健康检查的时间
55   periodSeconds: 10 # 间隔时间, 进行健康检查的时间间隔
56   successThreshold: 1 # 健康阈值, 表示后端容器从失败到成功的连续健康检查成功次数
57   failureThreshold: 1 # 不健康阈值, 表示后端容器从成功到失败的连续健康检查成功次数
58   timeoutSeconds: 3 # 响应超时, 每次健康检查响应的最大超时时间
59 readinessProbe: # 就绪检查, 检查容器是否就绪, 不就绪则停止转发流量到当前实例
60   httpGet: # HTTP请求检查方法
61     path: /healthz # 请求路径
62     port: 8080 # 检查端口
63     scheme: HTTP # 检查协议
64   initialDelaySeconds: 5 # 启动延时, 容器延时启动健康检查的时间
65   periodSeconds: 10 # 间隔时间, 进行健康检查的时间间隔
66   successThreshold: 1 # 健康阈值, 表示后端容器从失败到成功的连续健康检查成功次数
67   failureThreshold: 1 # 不健康阈值, 表示后端容器从成功到失败的连续健康检查成功次数
68   timeoutSeconds: 3 # 响应超时, 每次健康检查响应的最大超时时间
69 startupProbe: # 启动探针, 可以知道应用程序容器什么时候启动了
70   failureThreshold: 10
71   httpGet:
72     path: /healthz
73     port: 8080
74     scheme: HTTP
75   initialDelaySeconds: 5
76   periodSeconds: 10
77   successThreshold: 1
78   timeoutSeconds: 3
79 resources: # 资源管理
80   limits: # limits用于设置容器使用资源的最大上限, 避免异常情况下节点资源消耗过多
81     cpu: "1" # 设置cpu limit, 1核心 = 1000m
82     memory: 1Gi # 设置memory limit, 1G = 1024Mi
83   requests: # requests用于预分配资源, 当集群中的节点没有request所要求的资源数量
84     cpu: 250m # 设置cpu request
85     memory: 500Mi # 设置memory request
86 terminationMessagePath: /dev/termination-log # 容器终止时消息保存路径
87 terminationMessagePolicy: File # 仅从终止消息文件中检索终止消息
88 volumeMounts: # 挂载日志卷
89 - mountPath: /etc/iam/iam-apiserver.yaml # 容器内挂载镜像路径
90   name: iam # 引用的卷名称
91   subPath: iam-apiserver.yaml # 指定所引用的卷内的子路径, 而不是其根路径。
92 - mountPath: /etc/iam/cert
93   name: iam-cert
94 dnsPolicy: ClusterFirst
95 restartPolicy: Always # 重启策略, Always、OnFailure、Never
96 schedulerName: default-scheduler # 指定调度器的名字
97 imagePullSecrets: # 在Pod中设置ImagePullSecrets只有提供自己密钥的Pod才能访问私
```

```
98     - name: ccr-registry # 镜像仓库的Secrets需要在集群中手动创建
99 securityContext: {} # 指定安全上下文
100 terminationGracePeriodSeconds: 5 # 优雅关闭时间, 这个时间内优雅关闭未结束, k8s
101 volumes: # 配置数据卷, 类型详见https://kubernetes.io/zh/docs/concepts/stora
102 - configMap: # configMap 类型的数据卷
103     defaultMode: 420 #权限设置0~0777, 默认0664
104     items:
105     - key: iam-apiserver.yaml
106       path: iam-apiserver.yaml
107       name: iam # configmap名称
108     name: iam # 设置卷名称, 与volumeMounts名称对应
109 - configMap:
110     defaultMode: 420
111     name: iam-cert
112     name: iam-cert
```

在部署时, 你可以根据需要来配置相应的字段, 常见的需要配置的字段为: labels、name、namespace、replicas、command、imagePullPolicy、container.name、livenessProbe、readinessProbe、resources、volumeMounts、volumes、imagePullSecrets等。

另外, 在部署应用时, 经常需要提供配置文件, 供容器内的进程加载使用。最常用的方法是挂载 ConfigMap 到应用容器中。那么, 如何挂载 ConfigMap 到容器中呢?

引用 ConfigMap 对象时, 你可以在 volume 中通过它的名称来引用。你可以自定义 ConfigMap 中特定条目所要使用的路径。下面的配置就显示了如何将名为 log-config 的 ConfigMap 挂载到名为 configmap-pod 的 Pod 中:

[复制代码](#)

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: configmap-pod
5  spec:
6    containers:
7      - name: test
8        image: busybox
9        volumeMounts:
10         - name: config-vol
11           mountPath: /etc/config
12    volumes:
13      - name: config-vol
14        configMap:
15          name: log-config
```

```
16         items:
17             - key: log_level
18               path: log_level
```

log-config ConfigMap 以卷的形式挂载，并且存储在 log_level 条目中的所有内容都被挂载到 Pod 的/etc/config/log_level 路径下。请注意，这个路径来源于卷的 mountPath 和 log_level 键对应的path。

这里需要注意，在使用 ConfigMap 之前，你首先要创建它。接下来，我们来看下 ConfigMap 定义。

ConfigMap 资源定义

下面是一个 ConfigMap YAML 示例：

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4      name: test-config4
5  data: # 存储配置内容
6      db.host: 172.168.10.1 # 存储格式为key: value
7      db.port: 3306
```

[复制代码](#)

可以看到，ConfigMap 的 YAML 定义相对简单些。假设我们将上述 YAML 文件保存在了 iam-configmap.yaml文件中，我们可以执行以下命令，来创建 ConfigMap：


```
1 $ kubectl create -f iam-configmap.yaml
```

[复制代码](#)

除此之外，kubectl 命令行工具还提供了 3 种创建 ConfigMap 的方式。我来分别介绍下。

1) 通过--from-literal参数创建


创建命令如下：

 复制代码

```
1 $ kubectl create configmap iam-configmap --from-literal=db.host=172.168.10.1 -
```

2) 通过--from-file=<文件>参数创建

创建命令如下：


 复制代码

```
1 $ echo -n 172.168.10.1 > ./db.host
2 $ echo -n 3306 > ./db.port
3 $ kubectl create cm iam-configmap --from-file=./db.host --from-file=./db.port
```

--from-file的值也可以是一个目录。当值是目录时，目录中的文件名为 key，目录的内容为 value。

3) 通过--from-env-file参数创建


创建命令如下：

 复制代码

```
1 $ cat << EOF > env.txt
2 db.host=172.168.10.1
3 db.port=3306
4 EOF
5 $ kubectl create cm iam-configmap --from-env-file=env.txt
```

Service 资源定义

Service 是 Kubernetes 另一个核心资源。通过创建 Service，可以为一组具有相同功能的容器应用提供一个统一的入口地址，并且将请求负载到后端的各个容器上。Service 资源定义 YAML 文件如下：

 复制代码

```
1 apiVersion: v1
2 kind: Service
3 metadata:
```

```
4   labels:
5     app: iam-apiserver
6     name: iam-apiserver
7     namespace: default
8 spec:
9   clusterIP: 192.168.0.231 # 虚拟服务地址
10  externalTrafficPolicy: Cluster # 表示此服务是否希望将外部流量路由到节点本地或集群范围
11  ports: # service需要暴露的端口列表
12    - name: https #端口名称
13      nodePort: 30443 # 当type = NodePort时, 指定映射到物理机的端口号
14      port: 8443 # 服务监听的端口号
15      protocol: TCP # 端口协议, 支持TCP和UDP, 默认TCP
16      targetPort: 8443 # 需要转发到后端Pod的端口号
17  selector: # label selector配置, 将选择具有label标签的Pod作为其后端RS
18    app: iam-apiserver
19  sessionAffinity: None # 是否支持session
20  type: NodePort # service的类型, 指定service的访问方式, 默认为clusterIp
```

上面, 我介绍了常用的 Kubernetes YAML 的内容。我们在部署应用的时候, 是需要手动编写这些文件的。接下来, 我就讲解一些在编写过程中常用的编写技巧。

YAML 文件编写技巧

这里我主要介绍三个技巧。

1) 使用在线的工具来自动生成模板 YAML 文件。

YAML 文件很复杂, 完全从 0 开始编写一个 YAML 定义文件, 工作量大、容易出错, 也没必要。我比较推荐的方式是, 使用一些工具来自动生成所需的 YAML。

这里我推荐使用 [k8syaml](#) 工具。k8syaml 是一个在线的 YAML 生成工具, 当前能够生成 Deployment、StatefulSet、DaemonSet 类型的 YAML 文件。k8syaml 具有默认值, 并且有对各字段详细的说明, 可以供我们填参时参考。

2) 使用 `kubectl run` 命令获取 YAML 模板:

 复制代码


```
1 $ kubectl run --dry-run=client --image=nginx nginx -o yaml > my-nginx.yaml
2 $ cat my-nginx.yaml
3 apiVersion: v1
4 kind: Pod
5 metadata:
```

```
6   creationTimestamp: null
7   labels:
8     run: nginx
9   name: nginx
10 spec:
11   containers:
12   - image: nginx
13     name: nginx
14     resources: {}
15   dnsPolicy: ClusterFirst
16   restartPolicy: Always
17 status: {}
```

然后，我们可以基于这个模板，来修改配置，形成最终的 YAML 文件。

3) 导出集群中已有的资源描述。

有时候，如果我们想创建一个 Kubernetes 资源，并且发现该资源跟集群中已经创建的资源描述相近或者一致的时候，可以选择导出集群中已经创建资源的 YAML 描述，并基于导出的 YAML 文件进行修改，获得所需的 YAML。例如：

 复制代码

```
1 $ kubectl get deployment iam-apiserver -o yaml > iam-authz-server.yaml
```

接着，修改iam-authz-server.yaml。通常，我们需要删除 Kubernetes 自动添加的字段，例如ubectl.kubernetes.io/last-applied-configuration、deployment.kubernetes.io/revision、creationTimestamp、generation、resourceVersion、selfLink、uid、status。

这些技巧可以帮助我们更好地编写和使用 Kubernetes YAML。

使用 Kubernetes YAML 时的一些推荐工具

接下来，我再介绍一些比较流行的工具，你可以根据自己的需要进行选择。

kubeval

 kubeval可以用来验证 Kubernetes YAML 是否符合 Kubernetes API 模式。

安装方法如下：

[复制代码](#)

```
1 $ wget https://github.com/instrumenta/kubeval/releases/latest/download/kubeval
2 $ tar xf kubeval-linux-amd64.tar.gz
3 $ mv kubeval $HOME/bin
```

安装完成后，我们对 Kubernetes YAML 文件进行验证：

[复制代码](#)

```
1 $ kubeval deployments/iam.invalid.yaml
2 ERR - iam/templates/iam-configmap.yaml: Duplicate 'ConfigMap' resource 'iam'
```

根据提示，查看iam.yaml，发现在iam.yaml文件中，我们定义了两个同名的iam ConfigMap：

[复制代码](#)

```
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: iam
5 data:
6   {}
7 ---
8 # Source: iam/templates/iam-configmap.yaml
9 apiVersion: v1
10 kind: ConfigMap
11 metadata:
12   name: iam
13 data:
14   iam: ""
15   iam-apiserver.yaml: |
16     ...
```

可以看到，使用kubeval之类的工具，能让我们在部署的早期，不用访问集群就能发现YAML 文件的错误。

kube-score

🔗 **kube-score**能够对 Kubernetes YAML 进行分析，并根据内置的检查对其评分，这些检查是根据安全建议和最佳实践而选择的，例如：

以非 Root 用户启动容器。

为 Pods 设置健康检查。

定义资源请求和限制。

你可以按照这个方法安装：

```
1 $ go get github.com/zegl/kube-score/cmd/kube-score
```

[📄 复制代码](#)

然后，我们对 Kubernetes YAML 进行评分：

```
1 $ kube-score score -o ci deployments/iam.invalid.yaml
2 [OK] iam-apiserver apps/v1/Deployment
3 [OK] iam-apiserver apps/v1/Deployment
4 [OK] iam-apiserver apps/v1/Deployment
5 [OK] iam-apiserver apps/v1/Deployment
6 [CRITICAL] iam-apiserver apps/v1/Deployment: The pod does not have a matching
7 [CRITICAL] iam-apiserver apps/v1/Deployment: Container has the same readiness
8 [CRITICAL] iam-apiserver apps/v1/Deployment: (iam-apiserver) The pod has a con
9 [CRITICAL] iam-apiserver apps/v1/Deployment: (iam-apiserver) The container is
10 [CRITICAL] iam-apiserver apps/v1/Deployment: (iam-apiserver) The container run
11 [OK] iam-apiserver apps/v1/Deployment
12 ...
```

[📄 复制代码](#)

检查的结果有OK、SKIPPED、WARNING和CRITICAL。CRITICAL是需要你修复的；WARNING是需要你关注的；SKIPPED是因为某些原因略过的检查；OK是验证通过的。

如果你想查看详细的错误原因和解决方案，可以使用-o human选项，例如：

```
1 $ kube-score score -o human deployments/iam.invalid.yaml
```

[📄 复制代码](#)

上述命令会检查 YAML 资源定义文件，如果有不合规的地方会报告级别、类别以及错误详情，如下图所示：

```
[colin@dev iam]$ kube-score score -o human deployments/templates/*.yaml
apps/v1/Deployment iam-apiserver
[CRITICAL] Pod NetworkPolicy
  · The pod does not have a matching NetworkPolicy
    Create a NetworkPolicy that targets this pod to control who/what can communicate with this pod. Note,
    this feature needs to be supported by the CNI implementation used in the Kubernetes cluster to have an
    effect.
[CRITICAL] Pod Probes
  · Container has the same readiness and liveness probe
    Using the same probe for liveness and readiness is very likely dangerous. Generally it's better to
    avoid the livenessProbe than re-using the readinessProbe.
    More information: https://github.com/zegl/kube-score/blob/master/README_PROBES.md
[CRITICAL] Container Security Context
  · iam-apiserver -> The pod has a container with a writable root filesystem
    Set securityContext.readOnlyRootFilesystem to true
  · iam-apiserver -> The container is running with a low user ID
    A userid above 10 000 is recommended to avoid conflicts with the host. Set securityContext.runAsUser
    to a value > 10000
  · iam-apiserver -> The container running with a low group ID
    A groupid above 10 000 is recommended to avoid conflicts with the host. Set
    securityContext.runAsGroup to a value > 10000
[CRITICAL] Pod NetworkPolicy
  · The pod does not have a matching NetworkPolicy
    Create a NetworkPolicy that targets this pod to control who/what can communicate with this pod. Note,
    this feature needs to be supported by the CNI implementation used in the Kubernetes cluster to have an
    effect.
```

当然，除了 kubeval、kube-score 这两个工具，业界还有其他一些 Kubernetes 检查工具，例如 [config-lint](#)、[copper](#)、[conftest](#)、[polaris](#) 等。

这些工具，我推荐你这么来选择：首先，使用 kubeval 工具做最基本的 YAML 文件验证。验证通过之后，我们就可以进行更多的测试。如果你没有特别复杂的 YAML 验证要求，只需要用到一些最常见的检查策略，这时候可以使用 kube-score。如果你有复杂的验证要求，并且希望能够自定义验证策略，则可以考虑使用 copper。当然，polaris、config-lint、copper 也值得你去尝试下。

总结

今天，我主要讲了如何编写 Kubernetes YAML 文件。

YAML 格式具有丰富的数据表达能力、清晰的结构和层次，因此被用于 Kubernetes 资源的定义文件中。如果你要把应用部署在 Kubernetes 集群中，就要创建多个关联的 K8s 资源，如果要创建 K8s 资源，目前比较多的方式还是编写 YAML 格式的定义文件。

这一讲我介绍了 K8s 中最常用的四种资源（Pod、Deployment、Service、ConfigMap）的 YAML 定义的写法，你可以常来温习。

另外，在编写 YAML 文件时，也有一些技巧。比如，可以通过在线工具 [k8syaml](#) 来自动生成初版的 YAML 文件，再基于此 YAML 文件进行二次修改，从而形成终版。

最后，我还给你分享了编写和使用 Kubernetes YAML 时，社区提供的多种工具。比如，kubeval 可以校验 YAML，kube-score 可以给 YAML 文件打分。了解了如何编写 Kubernetes YAML 文件，下一讲的学习相信你会进行得更顺利。

课后练习

1. 思考一下，如何将 ConfigMap 中的 Key 挂载到同一个目录中，文件名为 Key 名？
2. 使用 kubeval 检查你正在或之前从事过的项目的 K8s YAML 定义文件，查看报错，并修改和优化。

欢迎你在留言区和我交流讨论，我们下一讲见。

分享给需要的人，Ta订阅后你可得 **24** 元现金奖励

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 46 | 如何制作Docker镜像？

下一篇 48 | 基于腾讯云 EKS 的容器化部署实战

更多学习推荐

175 道 Go 工程师 大厂常考面试题

限量免费领取 



精选留言 (1)

 写留言



随风而过

2021-09-14

覆盖掉挂载的整个目录，使用volumeMount.subPath来声明我们只是挂载单个文件，而不是整个目录，只需要在subPath后面加上我们挂载的单个文件名即可

