

14 | 深入解析Pod对象（一）：基本概念

2018-09-24 张磊

深入剖析Kubernetes

[进入课程 >](#)



讲述：张磊

时长 11:47 大小 5.41M



你好，我是张磊。今天我和你分享的主题是：深入解析 Pod 对象之基本概念。

在上一篇文章中，我详细介绍了 Pod 这个 Kubernetes 项目中最重要概念。而在今天这篇文章中，我会和你分享 Pod 对象的更多细节。

现在，你已经非常清楚：Pod，而不是容器，才是 Kubernetes 项目中的最小编排单位。将这个设计落实到 API 对象上，容器（Container）就成了 Pod 属性里的一个普通的字段。那么，一个很自然的问题就是：到底哪些属性属于 Pod 对象，而又有哪些属性属于 Container 呢？

要彻底理解这个问题，你就一定要牢记我在上一篇文章中提到的一个结论：Pod 扮演的是传统部署环境里“虚拟机”的角色。这样的设计，是为了使用户从传统环境（虚拟机环境）

向 Kubernetes (容器环境) 的迁移, 更加平滑。


而如果你能把 Pod 看成传统环境里的“机器”、把容器看作是运行在这个“机器”里的“用户程序”, 那么很多关于 Pod 对象的设计就非常容易理解了。

比如, **凡是调度、网络、存储, 以及安全相关的属性, 基本上是 Pod 级别的。**

这些属性的共同特征是, 它们描述的是“机器”这个整体, 而不是里面运行的“程序”。比如, 配置这个“机器”的网卡 (即: Pod 的网络定义), 配置这个“机器”的磁盘 (即: Pod 的存储定义), 配置这个“机器”的防火墙 (即: Pod 的安全定义)。更不用说, 这台“机器”运行在哪个服务器之上 (即: Pod 的调度)。

接下来, 我就先为你介绍 Pod 中几个重要字段的含义和用法。

NodeSelector : 是一个供用户将 Pod 与 Node 进行绑定的字段, 用法如下所示 :


 复制代码

```
1 apiVersion: v1
2 kind: Pod
3 ...
4 spec:
5   nodeSelector:
6     disktype: ssd
```

这样的配置, 意味着这个 Pod 永远只能运行在携带了“disktype: ssd”标签 (Label) 的节点上; 否则, 它将调度失败。

nodeName : 一旦 Pod 的这个字段被赋值, Kubernetes 项目就会被认为这个 Pod 已经经过了调度, 调度的结果就是赋值的节点名字。所以, 这个字段一般由调度器负责设置, 但用户也可以设置它来“骗过”调度器, 当然这个做法一般是在测试或者调试的时候才会用到。

HostAliases : 定义了 Pod 的 hosts 文件 (比如 /etc/hosts) 里的内容, 用法如下 :

 复制代码

```
1 apiVersion: v1
2 kind: Pod
3 ...
4 spec:
5   hostAliases:
6     - ip: "10.1.2.3"
7       hostnames:
8         - "foo.remote"
9         - "bar.remote"
10 ...
```

在这个 Pod 的 YAML 文件中，我设置了一组 IP 和 hostname 的数据。这样，这个 Pod 启动后，/etc/hosts 文件的内容将如下所示：


 复制代码

```
1 cat /etc/hosts
2 # Kubernetes-managed hosts file.
3 127.0.0.1 localhost
4 ...
5 10.244.135.10 hostaliases-pod
6 10.1.2.3 foo.remote
7 10.1.2.3 bar.remote
```

其中，最下面两行记录，就是我通过 HostAliases 字段为 Pod 设置的。需要指出的是，在 Kubernetes 项目中，如果要设置 hosts 文件里的内容，一定要通过这种方法。否则，如果直接修改了 hosts 文件的话，在 Pod 被删除重建之后，kubelet 会自动覆盖掉被修改的内容。

除了上述跟“机器”相关的配置外，你可能也会发现，**凡是跟容器的 Linux Namespace 相关的属性，也一定是 Pod 级别的**。这个原因也很容易理解：Pod 的设计，就是要让它里面的容器尽可能多地共享 Linux Namespace，仅保留必要的隔离和限制能力。这样，Pod 模拟出的效果，就跟虚拟机里程序间的关系非常类似了。

举个例子，在下面这个 Pod 的 YAML 文件中，我定义了 shareProcessNamespace=true：

 复制代码

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nginx
5 spec:
6   shareProcessNamespace: true
7   containers:
8   - name: nginx
9     image: nginx
10  - name: shell
11    image: busybox
12    stdin: true
13    tty: true
```


这就意味着这个 Pod 里的容器要共享 PID Namespace。

而在这个 YAML 文件中，我还定义了两个容器：一个是 nginx 容器，一个是开启了 tty 和 stdin 的 shell 容器。

我在前面介绍容器基础时，曾经讲解过什么是 tty 和 stdin。而在 Pod 的 YAML 文件里声明开启它们俩，其实等同于设置了 docker run 里的 -it (-i 即 stdin , -t 即 tty) 参数。

如果你还是不太理解它们俩的作用的话，可以直接认为 tty 就是 Linux 给用户提供的的一个常驻小程序，用于接收用户的标准输入，返回操作系统的标准输出。当然，为了能够在 tty 中输入信息，你还需要同时开启 stdin (标准输入流)。

于是，这个 Pod 被创建后，你就可以使用 shell 容器的 tty 跟这个容器进行交互了。我们一起实践一下：

 复制代码


```
1 $ kubectl create -f nginx.yaml
```

接下来，我们使用 kubectl attach 命令，连接到 shell 容器的 tty 上：

 复制代码

```
1 $ kubectl attach -it nginx -c shell
```


这样，我们就可以在 shell 容器里执行 ps 指令，查看所有正在运行的进程：

 复制代码

```
1 $ kubectl attach -it nginx -c shell
2 / # ps ax
3 PID    USER      TIME  COMMAND
4      1 root        0:00 /pause
5      8 root        0:00 nginx: master process nginx -g daemon off;
6     14 101        0:00 nginx: worker process
7     15 root        0:00 sh
8     21 root        0:00 ps ax
```

可以看到，在这个容器里，我们不仅可以看到它本身的 ps ax 指令，还可以看到 nginx 容器的进程，以及 Infra 容器的 /pause 进程。这就意味着，整个 Pod 里的每个容器的进程，对于所有容器来说都是可见的：它们共享了同一个 PID Namespace。

类似地，**凡是 Pod 中的容器要共享宿主机的 Namespace，也一定是 Pod 级别的定义，比如：**

 复制代码

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nginx
5 spec:
6   hostNetwork: true
7   hostIPC: true
8   hostPID: true
9   containers:
10  - name: nginx
11    image: nginx
12  - name: shell
13    image: busybox
14    stdin: true
15    tty: true
```

在这个 Pod 中，我定义了共享宿主机的 Network、IPC 和 PID Namespace。这就意味着，这个 Pod 里的所有容器，会直接使用宿主机的网络、直接与宿主机进行 IPC 通信、看到宿主机里正在运行的所有进程。

当然，除了这些属性，Pod 里最重要的字段当属 “Containers” 了。而在上一篇文章中，我还介绍过 “Init Containers”。其实，这两个字段都属于 Pod 对容器的定义，内容也完全相同，只是 Init Containers 的生命周期，会先于所有的 Containers，并且严格按照定义的顺序执行。


Kubernetes 项目中对 Container 的定义，和 Docker 相比并没有什么太大区别。我在前面的容器技术概念入门系列文章中，和你分享的 Image（镜像）、Command（启动命令）、workingDir（容器的工作目录）、Ports（容器要开发的端口），以及 volumeMounts（容器要挂载的 Volume）都是构成 Kubernetes 项目中 Container 的主要字段。不过在这里，还有这么几个属性值得你额外关注。

首先，是 ImagePullPolicy 字段。它定义了镜像拉取的策略。而它之所以是一个 Container 级别的属性，是因为容器镜像本来就是 Container 定义中的一部分。

ImagePullPolicy 的值默认是 Always，即每次创建 Pod 都重新拉取一次镜像。另外，当容器的镜像是类似于 nginx 或者 nginx:latest 这样的名字时，ImagePullPolicy 也会被认为 Always。

而如果它的值被定义为 Never 或者 IfNotPresent，则意味着 Pod 永远不会主动拉取这个镜像，或者只在宿主机上不存在这个镜像时才拉取。

其次，是 Lifecycle 字段。它定义的是 Container Lifecycle Hooks。顾名思义，Container Lifecycle Hooks 的作用，是在容器状态发生变化时触发一系列“钩子”。我们来看这样一个例子：

 复制代码

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: lifecycle-demo
5 spec:
6   containers:
7   - name: lifecycle-demo-container
8     image: nginx
```

```
9     lifecycle:
10       postStart:
11         exec:
12           command: ["/bin/sh", "-c", "echo Hello from the postStart handler > /usr/share/nginx/html/message"]
13       preStop:
14         exec:
15           command: ["/usr/sbin/nginx", "-s", "quit"]
```

这是一个来自 Kubernetes 官方文档的 Pod 的 YAML 文件。它其实非常简单，只是定义了一个 nginx 镜像的容器。不过，在这个 YAML 文件的容器（Containers）部分，你会看到这个容器分别设置了一个 postStart 和 preStop 参数。这是什么意思呢？

先说 postStart 吧。它指的是，在容器启动后，立刻执行一个指定的操作。需要明确的是，postStart 定义的操作，虽然是在 Docker 容器 ENTRYPOINT 执行之后，但它并不严格保证顺序。也就是说，在 postStart 启动时，ENTRYPOINT 有可能还没有结束。

当然，如果 postStart 执行超时或者错误，Kubernetes 会在该 Pod 的 Events 中报出该容器启动失败的错误信息，导致 Pod 也处于失败的状态。

而类似地，preStop 发生的时机，则是容器被杀死之前（比如，收到了 SIGKILL 信号）。而需要明确的是，preStop 操作的执行，是同步的。所以，它会阻塞当前的容器杀死流程，直到这个 Hook 定义操作完成之后，才允许容器被杀死，这跟 postStart 不一样。

所以，在这个例子中，我们在容器成功启动之后，在 /usr/share/nginx/html/message 里写入了一句“欢迎信息”（即 postStart 定义的操作）。而在这个容器被删除之前，我们则先调用了 nginx 的退出指令（即 preStop 定义的操作），从而实现了容器的“优雅退出”。

在熟悉了 Pod 以及它的 Container 部分的主要字段之后，我再和你分享一下**这样一个 Pod 对象在 Kubernetes 中的生命周期**。

Pod 生命周期的变化，主要体现在 Pod API 对象的**Status 部分**，这是它除了 Metadata 和 Spec 之外的第三个重要字段。其中，pod.status.phase，就是 Pod 的当前状态，它有如下几种可能的情况：

1. Pending。这个状态意味着，Pod 的 YAML 文件已经提交给了 Kubernetes，API 对象已经被创建并保存在 Etcd 当中。但是，这个 Pod 里有些容器因为某种原因而不能被顺

利创建。比如，调度不成功。

2. Running。这个状态下，Pod 已经调度成功，跟一个具体的节点绑定。它包含的容器都已经创建成功，并且至少有一个正在运行中。
3. Succeeded。这个状态意味着，Pod 里的所有容器都正常运行完毕，并且已经退出了。这种情况在运行一次性任务时最为常见。
4. Failed。这个状态下，Pod 里至少有一个容器以不正常的状态（非 0 的返回码）退出。这个状态的出现，意味着你得想办法 Debug 这个容器的应用，比如查看 Pod 的 Events 和日志。
5. Unknown。这是一个异常状态，意味着 Pod 的状态不能持续地被 kubelet 汇报给 kube-apiserver，这很有可能是主从节点（Master 和 Kubelet）间的通信出现了问题。

更进一步地，Pod 对象的 Status 字段，还可以再细分出一组 Conditions。这些细分状态的值包括：PodScheduled、Ready、Initialized，以及 Unschedulable。它们主要用于描述造成当前 Status 的具体原因是什么。

比如，Pod 当前的 Status 是 Pending，对应的 Condition 是 Unschedulable，这就意味着它的调度出现了问题。

而其中，Ready 这个细分状态非常值得我们关注：它意味着 Pod 不仅已经正常启动（Running 状态），而且已经可以对外提供服务了。这两者之间（Running 和 Ready）是有区别的，你不妨仔细思考一下。

Pod 的这些状态信息，是我们判断应用运行情况的重要标准，尤其是 Pod 进入了非“Running”状态后，你一定要能迅速做出反应，根据它所代表的异常情况开始跟踪和定位，而不是去手忙脚乱地查阅文档。

总结

在今天这篇文章中，我详细讲解了 Pod API 对象，介绍了 Pod 的核心使用方法，并分析了 Pod 和 Container 在字段上的异同。希望这些讲解能够帮你更好地理解 and 记忆 Pod YAML 中的核心字段，以及这些字段的准确含义。

实际上，Pod API 对象是整个 Kubernetes 体系中最核心的一个概念，也是后面我讲解各种控制器时都要用到的。

在学习完这篇文章后，我希望你能仔细阅读

`$GOPATH/src/k8s.io/kubernetes/vendor/k8s.io/api/core/v1/types.go` 里，`type Pod struct`，尤其是 `PodSpec` 部分的内容。争取做到下次看到一个 Pod 的 YAML 文件时，不再需要查阅文档，就能做到把常用字段及其作用信手拈来。

而在下一篇文章中，我会通过大量的实践，帮助你巩固和进阶关于 Pod API 对象核心字段的使用方法，敬请期待吧。

思考题

你能否举出一些 Pod（即容器）的状态是 Running，但是应用其实已经停止服务的例子？相信 Java Web 开发者的亲身体会会比较多吧。

感谢你的收听，欢迎你给我留言，也欢迎分享给更多的朋友一起阅读。



深入剖析 Kubernetes

Kubernetes 原来可以如此简单

张磊
Kubernetes 社区
资深成员与项目维护者



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 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 13 | 为什么我们需要Pod？

下一篇 15 | 深入解析Pod对象（二）：使用进阶

精选留言 (50)

写留言



blackpikle...

2018-09-24

42

对于 Pod 状态是 Ready，实际上不能提供服务的情况能想到几个例子：

1. 程序本身有 bug，本来应该返回 200，但因为代码问题，返回的是 500；
2. 程序因为内存问题，已经僵死，但进程还在，但无响应；
3. Dockerfile 写的不规范，应用程序不是主进程，那么主进程出了什么问题都无法发现；
4. 程序出现死循环。

展开

作者回复: 课代表来了



龙坤

2018-09-26

34

你好，我进入 shell 容器，然后执行 ps ax，跟例子的结果不一样。例子代码也一样，添加了 shareProcessNamespace: true 了，为什么不行呢，请问可能出现的原因在哪里

```
/ # ps
```

```
PID USER TIME COMMAND
```

```
1 root 0:00 sh...
```

展开



Jeff.W

2018-10-12

25

POD 的直译是豆荚，豆荚中的一个或者多个豆属于同一个家庭，共享一个物理豆荚（可以共享调度、网络、存储，以及安全），每个豆虽然有自己的空间，但是由于之间的缝隙，可以近距离无缝沟通（Linux Namespace 相关的属性）。

展开



安排

2018-10-05

4

你好，我进入 shell 容器，然后执行 ps ax，跟例子的结果不一样。例子代码也一样，添加了 shareProcessNamespace: true 了，为什么不行呢，请问可能出现的原因在哪里

```
/ # ps
```

PID USER TIME COMMAND

1 root 0:00 sh...

展开 ▾

作者回复: apiserver 加一feature-gates=PodShareProcessNamespace=true 1.11后已经默认开启了



asdf100

2018-09-30

👍 4

本地测试的进入shell容器后执行ps ax，跟例子的结果不一样。没有看到nginx窗口里的nginx进程信息，为什么，我看也有人遇到这个问题。

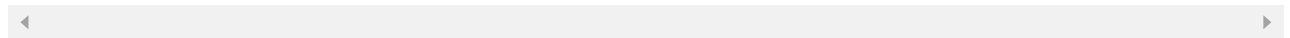
/ # ps

PID USER TIME COMMAND

1 root 0:00 sh...

展开 ▾

作者回复: 你看看sharepid功能开启了没



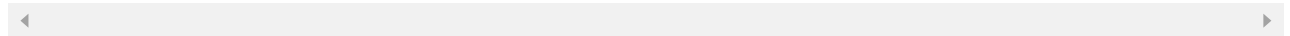
杨孔来

2018-09-25

👍 3

老师，如果pod中的image更新了（比如通过jenkins发布了新版本），我想通过重启pod，获取最新的image，有什么命令，可以优雅的重启pod，而不影响当前pod提供的业务吗

作者回复: 不是讲了prestop hook了？



追风

2019-01-01

👍 2

作者的那条评论。先让子弹飞一会，让我看出了作者决胜千里之外的眼界。哈哈哈



两两

2018-09-26

👍 2

pod runing好理解，但k8s怎么知道容器runing呢，通过什么标准判断？应用死循环，k8s怎么能感知？

作者回复: liveness和readiness啥区别？

◀ ▶



姜戈

2018-09-25

👍 2

通过node selector将任务调度到了woker1 成功运行之后 再修改worker1的label, 任务会重新调度吗？

作者回复: 不会

◀ ▶



hexinzhe

2018-09-25

👍 2

比较想要知道优雅停机方面的更详细内容，比如说terminationgraceperiodseconds与prestop之间的关系，两者怎么用

展开 ▾

作者回复: 前者就是sig term的超时时间。后者是要你自己编写逻辑处理的。

◀ ▶



混沌渺无极

2018-09-24

👍 2

各位，中秋节好。

如果entrypoint是一个一直运行的命令，那postStart会执行吗？还是启动一个协程成执行entrypoint，然后再运行一个协程执行这个postStart，所以这两个命令的执行状态是独立的，没有真正的先后关系。

展开 ▾

作者回复: 文中不是已经解释了？当然会执行，不管entrypoint。

◀ ▶



pytimer

2018-09-24

👍 2

老师，问一下，我看pod.status.phase是running，但是Ready是false，如果我想判断pod状态，要怎么做

作者回复: 看events



XThunderin...

2019-01-17

👍 1

文章中有处有问题："ImagePullPolicy 的值默认是 Always，即..." 这部分和官方文档与实际情况不一致。

在官方文档中提到"The default pull policy is IfNotPresent"，我这边在使用中发现的也是这样子的~

附一下官方文档链接：<https://kubernetes.io/docs/concepts/containers/images/>

展开 ▾

作者回复: 最新版本才改的呢



橄榄树

2018-12-22

👍 1

老师 和你同样的版本为什么shareProcessNamespace不起作用呢？



王天明

2018-12-12

👍 1

张老师，有一个问题，在使用kubeadm安装后，我查看etc与apiserver的信息，发现他们俩的ip都是宿主机内网IP，同时在apiserver中有关etc的地址写的是127.0.0.1,很是纳闷，又重要找到这节课，是因为作了如下设置了吗？

hostNetwork: true

hostIPC: true...

展开 ▾



细雨

2018-12-03

👍 1

问一下老师，infra 网络的镜像为什么取名字叫 pause 呀，难道它一直处于“暂停状态”吗？

作者回复: 对啊



短笛

2018-11-21

👍 1

Pod 的意思我理解应该是指 a small herd or school of marine animals, especially whales 而不是豆荚，为什么是鲸群呢？因为 Docker 的 Logo 啊 🐳

展开 ∨



陆培尔

2018-10-28

👍 1

关于pod还有一事请教，之前老师说过pod所有进出流量都会经pause这个根容器，那么是否可以这样理解，实现service mesh的最佳方式是扩展这个根容器的功能来做流量控制和路由，这比再注入一个envoy要更加底层，更加原生？

展开 ∨

作者回复: 当然不是。pause和sidecar没有任何区别。



北卡

2018-09-27

👍 1

只要容器没有down掉，pod就会处于running状态。pod只会监控到容器的状态，但不会监控容器里面程序的运行状态。如果程序处于死循环，或者其他bug状态，但并没有异常退出，此刻容器还是会处于存活状态，但实际上程序已经不能工作了。

日常使用的感受是这样的，不知道对不。

展开 ∨



Vincen

2018-09-26

👍 1

pod对象中的有些字段不能够在deployment中使用比如文章中提到的HostAliases，我们通常使用deployment来部署pod。这种情况怎么办？

展开 ∨

作者回复: 这个字段为啥不能用？

