**加我微信可进群学习交流：**

**微信号：**

luckylucky421302

也可通过扫描下面二维码添加



**课程更新的知识点会通过微信公众号免费分享给大家，可以关注我的公众号**

****

**DaemonSet控制器**

**1.概念**

daemonset控制器能够确保所有的k8s的node节点都运行一个相同的pod副本，当向k8s集群中增加node节点时，这个node节点也会自动创建一个pod副本，当node节点从集群移除，这些pod也会自动删除；删除daemonset也会删除它们创建的pod

**2.使用场景**

（1）在每一个node节点运行一个存储服务，例如gluster，ceph

（2）在每一个node节点运行一个日志收集服务，例如fluentd，logstash

（3）在每一个node节点运行一个监控服务，例如Prometheus Node Exporter，zabbix agent等

一个简单的用法是在所有的节点上都启动一个 DaemonSet， 一个稍微复杂的用法是单独对每种 daemon 类型使用多个 DaemonSet，但具有不同的标志， 并且对不同硬件类型具有不同的内存、CPU 要求。

**3.通过daemonset创建一个pod应用**

**cat daemonset.yaml**

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd-elasticsearch

namespace: kube-system

labels:

k8s-app: fluentd-logging

spec:

selector:

matchLabels:

name: fluentd-elasticsearch

template:

metadata:

labels:

name: fluentd-elasticsearch

spec:

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule

containers:

- name: fluentd-elasticsearch

image: lixiaofan0122/fluentd:v2.5.1

resources:

limits:

memory: 200Mi

requests:

cpu: 100m

memory: 200Mi

volumeMounts:

- name: varlog

mountPath: /var/log

- name: varlibdockercontainers

mountPath: /var/lib/docker/containers

readOnly: true

terminationGracePeriodSeconds: 30

volumes:

- name: varlog

hostPath:

path: /var/log

- name: varlibdockercontainers

hostPath:

path: /var/lib/docker/containers

kubectl apply -f daemonset.yaml

**4.yaml文件必须字段说明**

和其他kubernetes资源对象一样，daemonset需要apiVersion，kind，metadata，spec字段，可以通过**kubectl explain daemonset**查看具体字段信息和配置说明

* **Pod Template**
* .spec 中唯一必需的字段是 .spec.template。
* .spec.template 是一个 [Pod 模板](https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/#pod-templates)。除了它是嵌套的，而且不具有 apiVersion 或 kind 字段，它与 [Pod](https://kubernetes.io/docs/concepts/workloads/pods/pod/) 具有相同的 schema。
* 除了 Pod 必需字段外，在 DaemonSet 中的 Pod 模板必须指定合理的标签（查看 [Pod Selector](https://kubernetes.io/zh/docs/concepts/workloads/controllers/daemonset/#pod-selector)）。
* 在 DaemonSet 中的 Pod 模板必须具有一个值 [RestartPolicy](https://kubernetes.io/docs/user-guide/pod-states)，默认是 Always。
* **Pod Selector**

.spec.selector字段是一个pod选择器。它的工作方式与Job的.spec.selector相同。

从Kubernetes 1.8开始，您必须指定与.spec.template标签匹配的pod选择器。当空闲时，pod选择器将不再默认。选择器默认与kubectl apply不兼容。此外，一旦创建了DaemonSet，其.spec.selector就无法变动。

.spec.selector是一个由两个字段组成的对象：

    matchLabels - 与ReplicationController的.spec.selector相同，匹配拥有哪些标签的pod。

    matchExpressions - 允许通过指定键，值列表以及与键和值相关的运算符来构建更复杂的选择器。

指定两者时，结果为AND。

如果指定了.spec.selector，则它必须与.spec.template.metadata.labels匹配。具有这些不匹配的配置将被API拒绝。

通常不应直接创建任何标签与此选择器匹配的Pod，可以通过另一个DaemonSet或通过其他控制器（如ReplicaSet）创建。否则，DaemonSet控制器会认为这些Pod是由它创建的。

* **Running Pods on Only Some Nodes（仅在某些节点上与运行pod）**

如果指定.spec.template.spec.nodeSelector，则DaemonSet控制器将在与该节点选择器匹配的节点上创建Pod。 同样，如果指定.spec.template.spec.affinity，则DaemonSet控制器将在与该节点关联相匹配的节点上创建Pod。 如果您未指定任何一个，则DaemonSet控制器将在所有节点上创建Pod

* **怎么调度daemonset pod**

通过默认 scheduler 调度，DaemonSet 确保所有符合条件的节点都运行相同的 Pod 副本。通常，运行 Pod 的节点由 Kubernetes 调度器选择。不过，DaemonSet pods 由 DaemonSet 控制器创建和调度。这将引入以下问题：

1.Pod 行为的不一致性：等待调度的正常 Pod 已被创建并处于 Pending 状态，但 DaemonSet pods未在Pending状态下创建。

2.[Pod preemption](https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/)由默认scheduler处理。启用抢占后，DaemonSet控制器将在不考虑 pod优先级和抢占的情况下制定调度决策。

ScheduleDaemonSetPods允许使用默认调度器而不是 DaemonSet 控制器来调度 DaemonSets，方法是将NodeAffinity添加到DaemonSet pods，而不是.spec.nodeName。 然后使用默认调度器将pod绑定到目标主机。 如果 DaemonSet pod 的亲和节点已存在，则替换它。 DaemonSet 控制器仅在创建或修改 DaemonSet pods 时执行这些操作，并且不对 DaemonSet的spec.template进行任何更改。

**nodeAffinity**:

**requiredDuringSchedulingIgnoredDuringExecution**:

**nodeSelectorTerms**:

- **matchFields**:

- **key**: metadata.name

**operator**: In

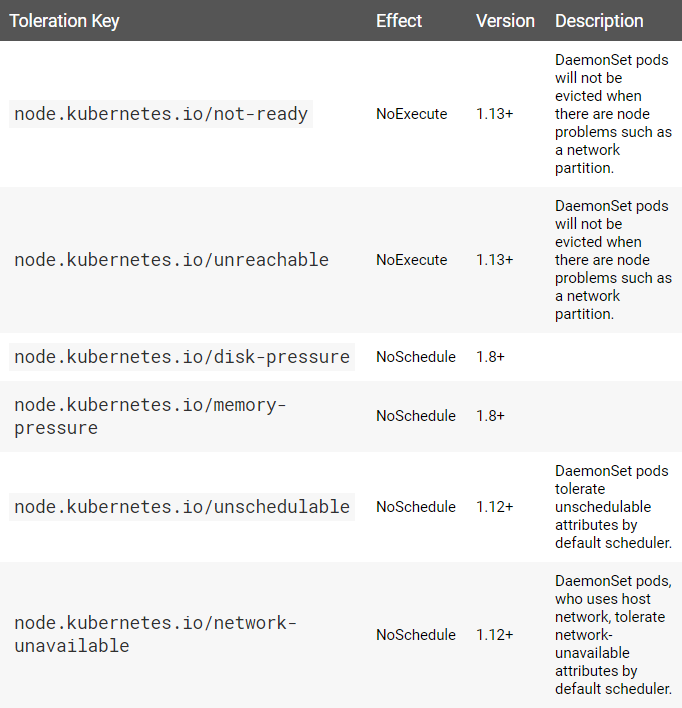
**values**:

- target-host-name

此外，系统会自动添加 node.kubernetes.io/unschedulable：NoSchedule 容忍度到 DaemonSet Pods。 在调度 DaemonSet Pod 时，默认调度器会忽略unschedulable节点。

* **污点和容忍度**

尽管Daemon Pods遵循[污点和容忍度](https://kubernetes.io/docs/concepts/configuration/taint-and-toleration)规则，根据相关特性，会自动将以下容忍度添加到 DaemonSet Pods 中。



* **与 DaemonSet 中的 Pod 进行通信**

与 DaemonSet 中的 Pod 进行通信的几种可能模式如下：

* **Push**：将DaemonSet中的Pod配置为将更新发送到其他服务，例如统计数据库。
* **NodeIP 和已知端口**：DaemonSet 中的 Pod 可以使用 hostPort，从而可以通过节点 IP 访问到 Pod。客户端能通过某种方法获取节点 IP 列表，并且基于此也可以获取到相应的端口。
* **DNS**：创建具有相同 Pod Selector 的 [Headless Service](https://kubernetes.io/docs/concepts/services-networking/service/#headless-services)，然后通过使用 endpoints 资源或从 DNS 中检索到多个 A 记录来发现 DaemonSet。
* **Service**：创建具有相同 Pod Selector 的 Service，并使用该 Service 随机访问到某个节点上的 daemonset pod（没有办法访问到特定节点）。
* **更新 DaemonSet**

如果修改了节点标签，DaemonSet 将立刻向新匹配上的节点添加 Pod，同时删除不能够匹配的节点上的 Pod。可以修改 DaemonSet 创建的 Pod。然而，不允许对 Pod 的所有字段进行更新。当下次 节点（即使具有相同的名称）被创建时，DaemonSet Controller 还会使用最初的模板。你可以删除一个 DaemonSet。如果使用 kubectl 并指定 --cascade=false 选项，则 Pod 将被保留在节点上。然后可以创建具有不同模板的新 DaemonSet。具有不同模板的新 DaemonSet 将能够通过标签匹配并识别所有已经存在的 Pod。 如果有任何 Pod 需要替换，则 DaemonSet 根据它的 updateStrategy 来替换

## DaemonSet 的可替代选择

### init 脚本

我们很可能希望直接在一个节点上启动 daemon 进程（例如，使用 init、upstartd、或 systemd）。这非常好，但基于 DaemonSet 来运行这些进程有如下一些好处：

* 像对待应用程序一样，具备为 daemon 提供监控和管理日志的能力。
* 为 daemon 和应用程序使用相同的配置语言和工具（如 Pod 模板、kubectl）。
* 在资源受限的容器中运行 daemon，能够增加 daemon 和应用容器的隔离性。然而，这也实现了在容器中运行 daemon，但却不能在 Pod 中运行（例如，直接基于 Docker 启动）。

### Deployments和Daemonset区别联系

DaemonSet 与 [Deployments](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) 非常类似，它们都能创建 Pod，这些 Pod 对应的进程都不希望被终止掉（例如，Web 服务器、存储服务器）。 无状态的 Service 使用 Deployments，比如前端 Frontend 服务，实现对副本的数量进行扩缩容、平滑升级，比基于精确控制 Pod 运行在某个主机上要重要得多。 需要 Pod 副本总是运行在全部或特定主机上，并需要先于其他 Pod 启动，当这被认为非常重要时，应该使用 daemonset。