|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DaemonSet** 和 **Deployment** 的区别；   |  |  | | --- | --- | | **资源类型** | **定义** | | **Deployment** | 用于管理 **一组副本 Pod**，确保指定数量的 Pod 在集群中运行，支持滚动更新。 | | **DaemonSet** | 用于在 **每个（或指定的）节点上运行一个 Pod**，确保每个节点都有一个 Pod 实例，通常用于节点级别的服务。 |   区别：   |  |  |  | | --- | --- | --- | | **特性** | **Deployment** | **DaemonSet** | | **Pod 副本控制** | 指定 replicas 数量 | 每个节点一个 Pod（可通过节点选择器过滤节点） | | **调度方式** | Scheduler 自由调度到任意节点，只要满足资源即可 | 自动在每个节点（符合选择条件）上创建 Pod | | **更新策略** | 支持 RollingUpdate 或 Recreate | 一般更新所有节点的 Pod（可用 RollingUpdate） | | **用途** | 应用服务、无状态服务 | 节点级别服务，如日志采集（Fluentd）、监控（Prometheus node exporter）、网络插件（CNI） | | **生命周期** | 可以随意扩缩容 | Pod 会随节点添加/删除而自动增减 | | **副本数管理** | 用户控制，   |  | | --- | |  |  |  | | --- | | 可自定义 replicas | | 自动管理，等于节点数 |   **调度对比**   * **Deployment**   + Pod 由 Scheduler 调度到任意符合条件的节点   + 副本数量固定（replicas），Pod 可能集中在部分节点 * **DaemonSet**   + Kubernetes 自动在每个节点创建一个 Pod   + Pod 会随着节点增加自动创建   + 可以结合 **NodeSelector / Taints / Affinity** 控制部署哪些节点   总结：  **Deployment = 指定副本，调度灵活 → 应用服务**   **DaemonSet = 每节点一个 → 节点服务/守护进程**  **RollingUpdate 关键字段**  strategy:  type: RollingUpdate  rollingUpdate:  maxSurge: 1 # 更新时可多出 1 个 Pod  maxUnavailable: 0 # 更新期间最多不可用 0 个 Pod  **命名空间作用**   * DaemonSet 必须在某个 **namespace** 下创建，例如 default、kube-system 或自定义 namespace * **只能在该 namespace 内被管理和查询**： * kubectl get daemonset -n kube-system * kubectl delete daemonset my-daemonset -n default * 不同 namespace 可以有 **同名 DaemonSet**，互不影响   **对比 ClusterScope 资源**   * **DaemonSet 是 Namespaced** * **ClusterRole / ClusterRoleBinding**、**Node / PersistentVolume** 等是 **ClusterScope**（集群级） * 也就是说，DaemonSet 的生命周期、查询、删除都受 namespace 限制，但它创建的 Pod 可以调度到集群中的任意节点（满足节点选择器或亲和性条件）   **Namespaced 资源**   * **作用域**：限制在某个命名空间内进行创建、管理和查询 * **不限制 Pod 调度的节点**，除非你自己设置节点约束（NodeSelector、NodeAffinity、Taints/Tolerations）   调度限制：  **NodeSelector、NodeAffinity 和 Taints/Tolerations**  **NodeSelector**   * **类型**：最简单的节点选择方式 * **作用**：Pod 只能调度到 **标签完全匹配的节点** * **语法**：   nodeSelector:  disktype: ssd   * **特点**：   + 简单粗暴，**只能做等于匹配**   + 不支持逻辑运算（>、<、in、notin 等） * **适用场景**：简单的节点标记选择   **NodeAffinity（节点亲和性）**   * **类型**：比 NodeSelector 更灵活的节点选择策略 * **作用**：定义 **Pod 与节点标签的亲和规则** * **语法示例**：   affinity:  nodeAffinity:  requiredDuringSchedulingIgnoredDuringExecution:  nodeSelectorTerms:  - matchExpressions:  - key: disktype  operator: In  values:  - ssd   * **特点**：   + 支持 **逻辑运算**（In、NotIn、Gt、Lt）   + 支持 **硬性约束（Required）** 或 **软性偏好（Preferred）**   + 可与 Pod 调度器结合，更灵活控制 Pod 分布 * **适用场景**：复杂调度需求，例如高性能存储节点、GPU 节点等   **Taints / Tolerations（污点 / 容忍）**   * **类型**：从节点角度控制哪些 Pod 可以调度到节点 * **作用**：   + **节点打上污点（Taint）** → 默认阻止所有 Pod 调度   + **Pod 设置 Toleration** → 可以容忍污点，允许调度到该节点  |  |  |  |  | | --- | --- | --- | --- | | **特性** | **NodeSelector** | **NodeAffinity** | **Taints/Tolerations** | | **方向** | Pod → 节点 | Pod → 节点 | 节点 → Pod | | **复杂度** | 简单 | 灵活，可逻辑运算 | 灵活，节点主动控制 | | **硬性/软性** | 硬性约束 | 可硬可软 | 污点硬性阻止，Pod 可容忍 | | **使用场景** | 简单节点选择 | 高级调度策略 | 节点隔离、专用节点控制 |   NodeSelector 简单粗暴，NodeAffinity 灵活亲和，Taints/Tolerations 节点说不，你能容忍才行   DaemonSet 的目标是 **在集群中的每个节点上都运行一个 Pod**。   所以默认副本数 = 集群中符合条件的节点数，而不是 1。  **updateStrategy 的作用**  updateStrategy:  type: RollingUpdate  rollingUpdate:  maxUnavailable: 1  **DaemonSet 的滚动更新**   * **没有 replicas**，Pod 数由 **节点数量** 决定 * **没有 maxSurge** → 相当于 maxSurge=0 * **只支持 maxUnavailable** * **可用 Pod 数范围**：   更新期间不会临时多创建 Pod，每次只替换部分节点的 Pod，保证可用性。   * **type: RollingUpdate**：滚动更新 Pod（逐个替换节点上的 Pod） * **maxUnavailable: 1**：更新过程中最多允许 **1 个节点的 Pod 不可用**   + 举例：     - 集群有 5 个节点     - 更新镜像版本     - Kubernetes 会先删除 1 个节点的旧 Pod，再创建新 Pod     - 其他 4 个节点的 Pod 保持运行 * 这样可以保证服务 **不中断或可用性尽量高**    **DaemonSet 的 RollingUpdate 只允许设置 maxUnavailable**   没有单独的 minAvailable 配置（这是 Deployment 的概念）   你通过设置 maxUnavailable 控制可用 Pod 最少数量：   你的例子：5 - 1 = 4   所以更新时可用节点数范围：4 ~ 5 |

示例

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 四个镜像：  Nginx两个版本  Fluentd  Tomcat  示例1：    默认行为：  **Node 与 Namespace 的关系**   * **Node（节点）**：集群中的物理或虚拟机器，不属于任何 namespace，它是全局资源。 * **Namespace（命名空间）**：Kubernetes 用来隔离资源的逻辑空间，只影响 Pod、Service、ConfigMap 等 namespaced 对象。   **2️⃣ DaemonSet 的行为**   * DaemonSet 会在 **每个节点**（或者符合 nodeSelector/tolerations 的节点）上运行一个 Pod。 * 这个 Pod **属于 DaemonSet 所在的命名空间**。   + 例如你在 namespace: default 下创建 DaemonSet   + 它会在集群每个节点上创建 Pod   + 每个 Pod 的命名空间都是 default * 所以即使节点是全局的，DaemonSet 生成的 Pod 仍然受命名空间隔离。   应用后效果：    示例二，指定节点调度：  定义daemonset    给指定node设标签：    应用效果：    示例三，部署日志收集器：fluentd  EFK  编写daemonset：    污点容忍  容器定义{基本  {资源管理resources  {挂载节点日志目录volumemounts **readOnly: true**：挂载只读，避免 Pod  定义卷  **tolerations 的匹配方式：**  **olerations 的基本结构**  tolerations:  - key: node-role.kubernetes.io/master  operator: Exists # 或者 Equal  effect: NoSchedule  value: "" # 只有 operator: Equal 时才需要   * **key**：对应节点污点的 key * **operator**（可选，默认 Equal）：匹配方式   + Equal（默认）：key 和 value 都要匹配   + Exists：只匹配 key，不关心 value * **value**：污点的 value，仅当 operator=Equal 时才有用 * **effect**：对应污点的 effect（NoSchedule、PreferNoSchedule、NoExecute）   **你的例子分析**  tolerations:  - key: node-role.kubernetes.io/master  effect: NoSchedule   * 没有写 operator → 默认 Equal * 没有写 value → 默认为空字符串 * 匹配逻辑：只要节点污点 **key=node-role.kubernetes.io/master, value=""** 并且 effect=NoSchedule，Pod 就可以调度 * 通常 master 节点的污点写法是： * key=node-role.kubernetes.io/master, effect=NoSchedule   value 通常为空，所以这个 toleration 可以匹配 master 节点   |  |  |  | | --- | --- | --- | | **effect** | **调度新 Pod** | **已存在 Pod** | | NoSchedule | 不允许 | 不影响 | | PreferNoSchedule | 尽量避免 | 不影响 | | NoExecute | 不允许 | 会驱逐 |   最严格的是：noexcute  创建命名空间：    应用daemonset：    Pod名： **Deployment/ReplicaSet 创建的 Pod**   * Pod 名 = **Deployment 名或 ReplicaSet 名 + 随机 hash**   **DaemonSet 创建的 Pod**   * Pod 名 = **DaemonSet 名 + PodTemplate hash + 节点名**   Kubectl describe po poname -n logging 看到各种信息：    而且进入也看一看到目录是被绑定了    滚动更新示例  定义daemonset：滚动更新    应用：    访问：    **镜像自带端口**   * 一些镜像（如 nginx、redis 等）在 Dockerfile 中 **通过 EXPOSE 声明了端口** * 例如 nginx 官方镜像： * EXPOSE 80 * **作用**：   + EXPOSE 只是告诉别人这个容器通常会使用这个端口   + 并不会自动对外开放网络访问   + 容器内部应用仍然监听 80 端口   **2️⃣ Kubernetes Pod 的 containerPort**  containers:  - name: nginx  image: nginx:1.21  ports:  - containerPort: 80   * **containerPort** 只是一个元数据（声明端口） * Kubernetes 用它做：   + 对 Service 创建端口映射   + 对一些工具（比如 kubectl port-forward、网络策略）有提示作用 * **不会实际改变容器内部应用监听的端口**   + 例如 nginx 镜像本身监听 80，即使不写 containerPort: 80，Pod 内 nginx 也在 80 端口运行   + 只是 Kubernetes 不知道这个端口信息   进行更新：    升级效果：    Ondelete跟新策略：  定义daemonset：    应用：    访问正常：    此时更新：    删除一个以后： |