|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 维持指定数量的Pod副本  定义pod    定义replicatset对象：    效果是接管了匹配标签的已有pod：    删除后自动创建：    命令动态扩容：     * kubectl scale 只能用于支持副本数调整的资源类型，如 Deployment, ReplicaSet, StatefulSet。不支持的资源会报错。   配合Service实现负载均衡      删除rs：     |  |  |  | | --- | --- | --- | | **控制器** | **管理已有 Pod** | **行为** | | **Deployment** | ❌ 不会管理已有 Pod | 只会管理自己通过模板创建的 Pod | | **ReplicaSet** | ✅ 可以管理已有 Pod | 只要 Pod 标签匹配，它会接管并确保副本数一致 | |

Deployment

|  |
| --- |
| deployment是一个三级结构，deployment先管理replicaset，replicaset再管理pod  deployment示例：  定义deployment    应用效果:  可以看见deployment和其底层创建的rs：    访问试下：    滚动更新：  命令更新：    可以看到滚动更新：    kubectl set image 是 Kubernetes 中一个非常常用的命令，用于 **更新 Deployment、DaemonSet、StatefulSet 等控制器中容器的镜像版本**，从而触发一次**滚动更新（Rolling Update）**  **验证版本被更新了**    这里的更新不会改变yaml文件的模板版本  修改yaml实现滚动更新：  yaml还是原来的：    直接应用：    效果：    #查看Deployment历史版本    回滚到上一个版本    ubectl rollout undo deployment/nginx-deploy #**默认是回滚到上一个版本（即上一次更新之前的版本）**，而不是最初的第一个版本。  使用指定版本的命令，效果是：    验证回滚结果：    的image那一行    变化的版本指的是？  Template  template: # ← 重点：Pod 模板（变化这里会产生新 revision）  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:1.21.5 # ✅ 镜像版本变化  ports:  - containerPort: 80  env: # ✅ 环境变量变化  - name: ENV  value: dev  livenessProbe: # ✅ 探针变化  httpGet:  path: /healthz  port: 80  readinessProbe: # ✅ 探针变化  httpGet:  path: /ready  port: 80  resources: # ✅ 资源限制变化  requests:  cpu: "100m"  memory: "128Mi"  limits:  cpu: "200m"  memory: "256Mi"  command: ["nginx", "-g", "daemon off;"] # ✅ 启动命令变化  args: ["-T"] # ✅ 启动参数变化  查看更新进程：    查看指定 Deployment（nginx-deploy）的滚动更新状态。  **更新中**  deployment "nginx-deploy" successfully rolled out  表示 Deployment 已经完成滚动更新，所有新 Pod 都已就绪。  **2️⃣ 更新过程中**  Waiting for rollout to finish: 2 of 3 updated replicas are available...   * Deployment 正在滚动更新中。 * 显示已经更新的副本数 vs 期望副本数。   **3️⃣ 更新失败或超时**  deployment "nginx-deploy" failed to rollout: timed out waiting for the condition  或使用：查看实时的pod状态    暂停部署（中断回滚或更新）  更新回滚+立即暂停：    可以看到：      后续操作：        其余操作：    hap自动扩缩容示例：  apiVersion: autoscaling/v2  kind: HorizontalPodAutoscaler  metadata:  name: nginx-hpa # HPA 对象名称  spec:  scaleTargetRef: # 指定要自动扩缩容的目标对象  apiVersion: apps/v1  kind: Deployment  name: nginx-deploy # 目标是名为 nginx-deploy 的 Deployment  minReplicas: 2 # 最少保持 2 个副本（不低于此数量）  maxReplicas: 10 # 最多扩容到 10 个副本（不超过此数量）  metrics:  - type: Resource # 监控的指标类型：资源  resource:  name: cpu # 指标是 CPU 使用率  target:  type: Utilization  averageUtilization: 70 # 当平均 CPU 利用率超过 70% 时扩容  这份 HPA 配置让 nginx-deploy 的 Pod 数量在 2～10 之间自动伸缩，当平均 CPU 使用率超过 70% 时扩容，低于 70% 时缩容。  **HPA 的初始状态副本数**，确实是以 Deployment 里 .spec.replicas 的值为基准。 |

滚动更新策略

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rollingupdate资源对象    滚动更新时，Kubernetes 会 **同时存在旧版本 Pod 和新版本 Pod**。 这两个参数就是用来控制「更新期间」Pod 数量变化范围的。  **maxUnavailable —— 最大不可用数**  意思是：**在更新过程中，最多有多少个 Pod 可以处于不可用状态。**   * 类型：整数 或 百分比 * 表示：更新时允许“临时停掉”的旧 Pod 数量 * 向下取整   例如：  maxUnavailable: 1  假设原本有 5 个副本，那么在滚动更新中，最多有 1 个旧 Pod 可以“被删除但新 Pod 还没 ready”。  **maxSurge —— 最大超量数**  意思是：**在更新过程中，最多可以额外创建多少个新 Pod。**   * 类型：整数 或 百分比 * 表示：更新时允许“临时超出副本数上限”的 Pod 数量 * 向上取整   例如：  maxSurge: 1  原本 5 个副本，更新时最多可以同时运行 6 个 Pod（1 个超额的新 Pod）。  **生产推荐配置：「一上一下，先上后下」**  strategy:  type: RollingUpdate  rollingUpdate:  maxUnavailable: 0  maxSurge: 1   * 含义：更新时**先启动 1 个新版本 Pod**； * 等它通过健康检查（readinessProbe）； * 再优雅地删除 1 个旧版本 Pod； * 如此循环直到全部替换完成。   🟢 优点：   * 无中断（服务持续可用） * 负载平滑 * 容易回滚   🔴 缺点：   * 更新速度较慢（特别是副本多时）     自定义更新策略示例  是在deployment里strategy字段指定的  创建deployment对象：  --record 记录到回滚  查看回滚记录：    改镜像版本，重新应用，触发回滚，看更新策略：    看到版本变了，但是没看到-o wide策略  再修改滚动更新策略  改为maxsurge=1  Maxunaviliable=0    **动态修改 Deployment 的滚动更新策略**的   |  |  | | --- | --- | | **部分** | **含义** | | kubectl patch | 表示对已有资源做“部分修改（补丁更新）” | | deployment myapp-v1 | 指定要修改的资源类型和名称（这里是名为 myapp-v1 的 Deployment） | | -p | 表示后面跟的是一段 JSON 格式的修改内容（patch 内容） | | '{"spec":{...}}' | JSON 补丁，用来修改 .spec 字段下的配置 |   **效果说明**   |  |  |  | | --- | --- | --- | | **参数** | **含义** | **说明** | | **maxSurge: 1** | 滚动更新时，最多可以“临时多建”1个 Pod | 更新时 Pod 总数 = replicas + 1 | | **maxUnavailable: 0** | 滚动更新时，不允许任何 Pod 不可用 | 意味着必须先启动新 Pod，确认可用后再删除旧 Pod（“一上一下”最平滑策略） |   查看修改后的策略，验证策略被修改：      命令总结：    Rs和deployment地异同点：   |  |  |  | | --- | --- | --- | | **项目** | **ReplicaSet（RS）** | **Deployment** | | **定义** | 确保指定数量的 Pod 副本始终处于运行状态。 | 在 ReplicaSet 之上实现版本管理、滚动更新和回滚控制。 | | **作用层级** | 直接控制 Pod。 | 控制多个 ReplicaSet 的生命周期。 | | **典型用途** | 保证 Pod 数量恒定。 | 应用的持续部署与更新（CI/CD 场景）。 |   **Deployment 会维护多个历史版本的 ReplicaSet**，而每个 ReplicaSet 代表一个 **Pod 模板（Template）版本**。 当你更新 Deployment 的 Pod 模板（例如镜像版本、环境变量、标签），Deployment 会创建新的 ReplicaSet 并逐步用新 Pod 替换旧 Pod。  **但是问题：Pod 会被替换，数据岂不是丢了？**  👉 **没错，Pod 是临时的（Ephemeral）**， 每次更新或调度迁移时，Pod 都可能被销毁并重建。 所以——**要想保证数据不丢失，就必须“脱离 Pod 自身”，用持久化卷（PVC）保存。**  **✅ 三、Kubernetes 中的数据持久化机制**   |  |  |  | | --- | --- | --- | | **名称** | **英文名** | **作用** | | **Volume** | 临时存储卷 | Pod 内多个容器共享数据，但随 Pod 删除而消失 | | **PersistentVolume（PV）** | 持久卷 | 集群级的实际存储资源（NFS、云盘等） | | **PersistentVolumeClaim（PVC）** | 持久卷声明 | Pod 通过它“申请”持久卷 | | **StorageClass** | 存储类 | 定义如何动态创建 PV（云环境中常用） |   总结一句话：   |  |  | | --- | --- | | **层级** | **说明** | | **Deployment** | 负责滚动更新和版本控制 | | **ReplicaSet** | 负责某一版本 Pod 的数量一致性 | | **Pod** | 临时运行单元，可随时销毁重建 | | **PVC/PV** | **保存数据**，与 Pod 生命周期解耦 |     一个 Deployment 管多个 ReplicaSet（每个代表一个 Pod 模板版本）   Pod 是临时的，会被替换   要持久化数据，必须使用 PVC（持久卷声明）挂载外部存储   数据保存在 PV，不随 Pod 销毁而丢失 |