作业三 Kubernetes

介绍

Kubernetes 又称 K8s，8 表示中间省略的 8 个字母。

类似作业二中我们接触的 Docker Compose，Kubernetes 用于容器的编排、管理、监控，可以对容器进行批量的、自动化的操作，且比 Docker Compose 更强大。

本次作业中，我们将：

- 通过 Kind 启动一个本地 Kubernetes 集群
- 在集群上部署应用，并体验 Kubernetes 的各项特性

推荐使用本地环境完成本次作业。云服务器资源有限，可能无法正常使用Kind。

启动一个 K8s 集群

安装 Kind

配置真正的 K8s 集群步骤较为繁琐，为了简化环境配置，我们使用 Kind（Kubernetes In Docker），一种使用 Docker 容器作为 node 节点，运行本地 Kubernetes 集群的工具。

以 Ubuntu 为例（其他系统安装参考官方文档：kind - Quik Start），使用以下命令安装 Kind：

```
[ $(uname -m) = x86_64 ] && curl -Lo ./kind
https://kind.sigs.k8s.io/dl/v0.23.0/kind-linux-amd64

chmod +x ./kind

sudo mv ./kind /usr/local/bin/kind
```

拉取节点镜像：

```
docker pull kindest/node:v1.30.0
```

启动 K8s 集群：

```
kind create cluster
```

如果一切顺利，将出现如图所示的输出：

```
(dev) (base) stream@DESKTOP-0N45QHQ:~$ kind create cluster
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.30.0) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s.io/#community 🙂
```

安装 kubectl

为了能通过命令行与集群交互，我们还需要安装 kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
# 如果下载有困难，可在 https://gitee.com/dengzhiling1/kubectl/blob/master/kubectl 手动下载

# 安装
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

通过 kubectl，查看刚刚启动好的 K8s 集群中的节点：

```
kubectl get pods
```

输出如下：

```
NAME                 STATUS   ROLES           AGE   VERSION
kind-control-plane   Ready    control-plane   17m   v1.30.0
```

> **Tips:**
>
> kubectl 有大量的子命令，为了减轻输入负担，可通过以下命令添加自动补全
>
> ```
> source /usr/share/bash-completion/bash_completion
> source <(kubectl completion bash)
> ```

体验 K8s 特性

！ 由于篇幅有限，基础概念将一笔带过或省略，请自行查找相关资料，了解 Deployment, Pod 等 K8s 相关基本概念

我们的 K8s 跑在容器内，容器内和容器外的镜像列表是隔离的。K8s 在部署应用时需要拉取镜像，可能会遇到网络问题。

如果你在以下步骤中因为镜像拉取失败而未达到预期效果：

先在容器外拉取镜像，再通过 `kind load docker-image <镜像名>` 镜像载入到容器内，以供 K8s 直接使用。

以下步骤会用到的镜像：

- nginx:latest
- registry.k8s.io/metrics-server/metrics-server:v0.7.1
- alexeiled/stress-ng:0.12.05

通过创建 Deployment 部署应用

在 Kubernetes 中，通过创建 Deployment，可以创建应用程序（如同 Docker 的 image）的实例（如同 Docker 的容器），这个实例被包含在 Pod 的概念中。Pod 是 Kubernetes 中的最小管理单元。

创建 Deployment 之后，Deployment 将指示 Kubernetes 集群如何创建和更新应用程序的实例。

我们可以通过 YAML 文件描述 Deployment。例如，下面这个 YAML 文件`deployment-nginx.yaml`描述了一个运行 nginx:latest Docker 镜像的 Deployment：

```yaml
# deployment-nginx.yaml
apiVersion: apps/v1 # 指定 Kubernetes API 的版本，kubectl api-versions 可查看当前集群支持的版本
kind: Deployment    # 定义 Kubernetes 对象的类型，这里是 Deployment。
metadata:           # Deployment 的元数据，即 Deployment 的一些基本属性和信息
  name: nginx       # 为 Deployment 指定一个名称，这里是 nginx。
spec:               # 指定 Deployment 的规范，包括选择器、副本数量、模板等。
  selector:         # 定义如何选择 Pod，以便知道哪些 Pod 属于这个 Deployment。
    matchLabels:    # 使用标签选择器来匹配带有特定标签的 Pod。
      app: nginx    # 匹配所有带有 app=nginx 标签的 Pod。
  replicas: 2       # 设置 Deployment 应该维护的 Pod 副本数量，这里是 2 个。
  template:         # 定义 Pod 模板，这些 Pod 将基于此模板被创建。
    metadata:       # Pod 模板的元数据。
      labels:       # 定义 Pod 模板的标签，用于与 Deployment 的选择器匹配。
        app: nginx  # Pod 模板的标签设置为 app=nginx。
    spec:           # 期望 Pod 实现的功能（即在 Pod 中要部署的容器和其他配置）
      containers:   # 定义 Pod 应该包含的容器列表。
      - name: nginx # 容器的名称，这里是 nginx。
        image: nginx:latest # 指定容器使用的镜像，这里是 nginx 的最新版本。
        ports:              # 定义容器的端口映射。
        - name: http        # 端口的名称，这里是 http。
          containerPort: 80 # 容器内部监听的端口号，这里是 80。
```

在 K8s 集群中应用这个 Deployment：

```
kubectl apply -f deployment-nginx.yaml
```

如此一来，我们在 K8s 集群中创建了 2 个 Pod，每个 Pod 中都包含 1 个 Nginx 容器。

列出该 Deployment 所创建的 Pod：

```
$ kubectl get pods -l app=nginx
NAME                     READY   STATUS    RESTARTS   AGE
nginx-578b6698c8-mzb5p   1/1     Running   0          18m
nginx-578b6698c8-z2p78   1/1     Running   0          18m
```

列出指定 Pod 的信息，如查看 nginx-578b6698c8-mzb5p 这个 Pod：

```
kubectl describe pod nginx-578b6698c8-mzb5p
```

扩缩容与自动复原

“期望状态”（Desired State）原则是 K8s 的重要特性，简单来说，你可以定义你期望各个组件所达到的状态，K8s 会自动调整，将组件始终保持到这个状态。

例如，你期望：你的 Web 服务器始终运行在 4 个容器中，以达到负载均衡的目的；你的数据库复制到 3 个不同的容器中，以达到冗余的目的。在定义这一状态后，如果这 7 个容器中的任何一个出现故障，Kubernetes 引擎会检测到这一点，并自动创建出一个新的容器，以确保维持所需的状态。

在之前的 Deployment 中，我们期望我们定义的 Pod 数量为 2 个，当业务需求发生变化（如访问 Nginx 的流量陡增），我们希望将它扩容为 5 个：

```
kubectl scale deployment nginx --replicas=5
```

再次查看 Pod 的数量，K8s 已经自动增加到 5 个：

```
$ kubectl get pods -l app=nginx
NAME                     READY   STATUS    RESTARTS   AGE
nginx-578b6698c8-kxz2j   1/1     Running   0          65s
nginx-578b6698c8-mzb5p   1/1     Running   0          43m
nginx-578b6698c8-nvkvd   1/1     Running   0          65s
nginx-578b6698c8-tmdf9   1/1     Running   0          65s
nginx-578b6698c8-z2p78   1/1     Running   0          43m
```

为了观察 Pod 数量的变化过程，我们使用以下命令：

```
kubectl get deployment nginx --watch
```

然后新开一个终端，删除一个 Pod：

```
kubectl delete pod nginx-578b6698c8-kxz2j
```

可以观察到，Pod 的数量发生了以下变化。在发现有个 Pod 被删除后，K8s 自动创建了新的 Pod，以保持 Pod 数量始终为 5：

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   5/5     5            5           45m
nginx   4/5     4            4           47m
nginx   4/5     5            4           47m
nginx   5/5     5            5           47m
```

最后，我们将数量缩容为 2 个：

```
kubectl scale deployment nginx --replicas=2
```

观察到 K8s 自动删除了 3 个 Pod：

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   5/2     5            5           49m
nginx   5/2     5            5           49m
nginx   2/2     2            2           49m
```

通过 Service 暴露应用

虽然部署了 Nginx，但我们还不能在集群外部访问它们。尽管每个 Pod 都有一个唯一的 IP 地址（可以通过 `kubectl get pods -l app -o wide` 查看），但是如果没有 Service，这些 IP 不会暴露在集群外部。

为了让这些 Pod 对外提供服务，我们需要创建 Service。

Service 是将运行在一个或一组 Pod 上的网络应用程序公开为网络服务的方法。类似于 Deployment，我们同样通过 YAML 文件创建它：

```
# service-nginx.yaml
apiVersion: v1
kind: Service   # 定义 Kubernetes 对象的类型，这里是 Service。
metadata:       # 包含 Service 的元数据。
  name: nginx   # 为 Service 指定一个名称，这里是 nginx。
spec:           # 指定 Service 的规范，包括选择器、端口和类型等。
  selector:     # 定义 Service 如何选择后端 Pod，通过匹配标签。
    app: nginx  # 选择所有带有标签 app=nginx 的 Pod。
  ports:        # 定义 Service 监听的端口配置。
    - protocol: TCP # 指定端口使用 TCP 协议。
      port: 80
      # 定义了从集群外部到 Service 的网络访问点，即如果从集群外部向这个 Service 发送请求，流量会首先到达这个端口。
      targetPort: 80
      # Pod 中容器实际监听的端口。当 Service 接收到流量后，它会将流量转发到匹配的 Pod 上的 targetPort。
  type: ClusterIP       # 指定 Service 的类型，这里是 ClusterIP。
```

应用到 K8s：

```
kubectl apply -f service-nginx.yaml
```

查看刚刚部署的 service：

```
kubectl get services nginx
```

让我们进入 K8s 中的一个节点（事实上，我们也只有一个节点，也就是通过 `docker ps` 所能看见的名为 kind-control-plane 的容器），也即进入了集群内部：

```
docker exec -it kind-control-plane /bin/bash
```

在节点中执行命令：`curl -ks http://$(kubectl get svc nginx -o=jsonpath="{.spec.clusterIP}")`：

```
root@kind-control-plane:/# curl -ks http://$(kubectl get svc nginx -o=jsonpath="{.spec.clusterIP}")

...
<h1>Welcome to nginx!</h1>
```

我们看到了熟悉的 `Welcome to nginx!`，这说明我们成功访问了 Pod 中的 Nginx。

问题一：尝试在容器外直接执行这条命令。它能成功吗，为什么？并解释 `kubectl get svc nginx -o=jsonpath="{.spec.clusterIP}"` 做了什么事情。（注：你无需从 Docker 的角度回答，只需关注 ClusterIP 这个 Service 类型的特性）

接下来，我们将 Service Type 更改为 `NodePort`，并指定对应的端口，如下所示：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30100
  type: NodePort
```

并再次执行 `kubectl apply -f service-nginx.yaml`。

理论上，在此之后，我们可以直接通过 NodeIP:30100 来访问 Nginx。但由于我们的节点并非一台真实的物理机，而是一个 Docker 容器，我们又未对容器配置端口映射，因此在容器外部仍是无法通过 `curl http://NodeIP:30100` 直接访问的。

最后，我们删除这些 Deployment 和 Service：

```
kubectl delete -f service-nginx.yaml
kubectl delete -f deployment-nginx.yaml
```

资源分配与监控

到目前为止，我们已经学习了如何在 Kubernetes 上简单地调度 Pod，并假设集群总是能够承载我们的应用程序。然而，当 Kubernetes 调度一个 Pod 时，需要确保容器拥有足够的资源来运行。如果在一个资源有限的节点上调度一个大型应用程序，该节点可能会耗尽资源，导致出现意外行为。

Metrics Server

Metrics Server 是 Kubernetes 集群中的一个轻量级组件，用于为集群提供资源使用情况的监控和性能指标。它通过收集和聚合集群中各个节点的资源使用数据（如 CPU、内存、磁盘和网络），使得用户能够方便地查看和管理 Kubernetes 集群的性能。

部署 Metrics Server：

```
kubectl apply -f
https://gitee.com/dengzhiling1/kubectl/releases/download/v1.0/components.yaml
```

查看 Metrics Server 部署状态：

```
$ kubectl get pods -n kube-system -o wide | grep metrics
metrics-server-59576d77b-p27tb               1/1     Running   0          112m
 10.244.0.13   kind-control-plane   <none>           <none>
```

等待一分钟左右，让 metrics server 收集系统指标。然后通过以下命令查看集群中各个 node 和 pod 的资源使用情况：

```
# 查看 pod 的资源使用情况
kubectl top pod -A
# 查看 node 的资源使用情况
kubectl top node
```

在定义 Pod 模板时，可以指定每个容器需要多少资源。主要有两种类型的资源：CPU 和内存。Kubernetes 调度器使用这些信息来决定在哪里运行你的 Pods。我们可以为容器指定两种类型的资源数量：`request` 和 `limit` 。

当为 Pod 中的容器指定 `request` 时：

- Kubernetes 调度器使用这些信息来决定将 Pod 放置在哪个节点上。
- Kubernetes 为容器预留所 request（请求） 数量的系统资源，供其使用。

当为容器指定 `limit` 时：

- Kubernetes 强制执行这些 `limit`，以确保运行中的容器使用的资源不超过你设定的限制。

简而言之，`request` 是 Kubernetes 调度决策和资源预留的基础，而 `limit` 是用来防止容器过度消耗资源的机制。正确配置这些参数对于确保应用程序的稳定运行和集群资源的有效利用至关重要。

关于更多细节，参考 Kubernetes - 为 Pod 和容器管理资源

查看可用资源

查看各个 node 的资源使用情况：

```
kubectl top node
```

查看指定 node 的总体可用资源（我们只有一个节点，即 kind-control-plane）：

```
kubectl get nodes kind-control-plane -o yaml
```

问题二：在 `kubectl get nodes kind-control-plane -o yaml` 所输出的内容中，哪些字段在节点资源中表示可用资源？（有两个）通过利用 `kubectl explain` 工具，解释这些字段的含义。

为 Pod 限制资源

为了直观感受资源限制对 Pod 运行的影响，我们使用压力测试工具 stress-ng。

以下命令将在 K8s 中创建一个名为 stress 的 Deployment，它将使用镜像 alexeiled/stress-ng:0.12.05 创建一个 stress-ng 容器，并执行 `/stress-ng --cpu=1` 命令，运行压力测试：

```
kubectl create deploy stress --image=alexeiled/stress-ng:0.12.05 -- /stress-ng --cpu=1
```

稍等一会儿，再查看 pod 的资源使用情况：`kubectl top pod`

问题三：通过`kubectl edit deploy stress`命令修改配置，在合适的地方指定`limits`，将`cpu`限制为`100m`。稍等一会儿，再次查看 pod 的资源使用情况。在修改前和修改后，pod 分别使用了多少 CPU?

最后，我们删除这个 Deployment：

```
kubectl delete deploy stress
```

作业要求

回答前文中的**问题一（35分）、问题二（35分）、问题三（30分）**，并在你认为有必要时附上相关截图。

提交 DDL 及提交方式：

请于【**2024.06.10 23:59**】前，将作业 PDF 文件（文件命名为 **姓名-学号-第三次作业**，如 张三-23232323-第三次作业.pdf）发送至邮箱 【**dengzl11@163.com**】，邮件标题格式： **姓名-学号-第三次作业**