云操作系统：Minik8s Lab

本次大作业需要同学们完成一个迷你的容器编排工具minik8s，能够在多机上对满足CRI接口的容器进行管理，支持容器生命周期管理、动态伸缩、自动扩容等基本功能，并且基于minik8s实现两个自选要求，自选要求包括微服务、和Serverless平台集成等。实现过程允许、鼓励同学们使用etcd、zookeeper等现有框架。

基本功能要求

1. 实现Pod抽象，对容器生命周期进行管理

   Minik8s需要支持pod抽象，可以通过yaml来对pod进行配置和启动。

   基于pod抽象，Minik8s应该能够根据用户指令对pod的生命周期进行管理，包括控制pod的启动和终止。

   pod内需要能运行多个容器，它们可以通过localhost互相访问。

   用户能够通过pod的yaml配置文件指定容器的参数，包括：

   - kind: 即配置类型，应该为pod
   - name: pod名称
   - 容器镜像名和版本
   - 容器命令（command）
   - 容器资源用量（如1cpu，128MB内存）
   - volumn：共享卷
   - port：容器暴露的端口

   yaml的格式可以自行设计，包含上述内容即可，可以自行修改或添加新字段和新内容。建议参考kubernetes的写法进行设计。

   Minik8s可以通过get pod, describe pod之类的指令获得pod的运行状态，展示的运行状态信息和kubernetes类似（包括pod名，运行时间、运行状态等）。可以同时运行多个pod。指令格式可以自行设计。

2. 实现Service抽象

   Minik8s应当支持Service抽象，对pod的访问应当通过Service进行。Service需要支持至少两个pod的通信，对外提供service的虚拟ip。用户能够通过虚拟ip访问Service，由minik8s将请求具体转发至对应的pod。此外，Service内的pod也可以通过虚拟ip访问其他Service

   用户能够通过yaml配置文件来创建service，配置文件里应当指定以下内容（同理可以自行设计yaml格式，内容包括要求即可。同样强烈建议参考的kubernetes写法进行设计），包括：

   - kind：即配置类型，应该为service
   - name：service名称
   - selector：筛选包含的pod

- ports：暴露的端口，包括对外暴露的端口（port）和对pod暴露的端口（targetPort）

Minik8s可以通过get svc之类的指令获得service的运行状态。可以同时运行多个service。同样，指令格式可以自行设计。

3. 实现Pod ReplicaSet抽象（或者Deployment）

   Minik8s的Service需要对Pod指定多个replica，并且监控replica的状态。当pod发生crash或者被kill掉，minik8s会自动启动pod并且重新加入service，使得service对应的pod数量恢复到replica指定的状态。

   通过Service的请求以一定的负载均衡策略（随机/Roundrobin）分配到各个pod处。

   Replica配置可以通过类型（kind）为ReplicaSet的yaml配置文件来指定，也可以在启动Pod的时候使用类似于类型为Deployment的yaml配置文件直接指定。配置文件

   Replica对应的Pod可以是无状态的，因此恢复的时候不需要对状态进行同步。

4. 动态伸缩 (auto-scaling)

   除了ReplicaSet的固定replica数量，Minik8s也可以根据任务的负载对Service中Pod的replica数量进行动态扩容和缩容。具体需要有以下要求：

   a. Minik8s需要对Service下的Pod实际资源使用进行定期监控，监控对象需要包括**至少两种**资源类型，其中CPU为必选项，剩下的是自选项，可以是memory，memory bandwidth，I/O，network等。可以使用Prometheus等现有框架进行资源监控。

   b. 用户可以通过yaml配置文件来对动态扩容进行配置。配置文件需要至少包括以下内容：

      - name&kind：扩容配置的名称和类型，类型应该为HorizontalPodAutoscaler
      - 扩容的目标workload，其对象应当是Pod（或者ReplicaSet/Deployment）
      - minReplicas和maxReplicas，表示扩容Pod数量的上下限。显然maxReplicas必须不小于minReplicas。
      - metrics，表示指标类型和目标值，这里主要对资源进行要求。每种资源应当标记资源类型，以及扩缩容的标准（如Utilization。如上文要求，需要支持至少两种资源类型（一个配置文件里不一定要同时写两种资源）。

   c. Minik8s需要有扩缩容的策略。策略主要包含两个部分，其一为何时进行扩缩容，其二为扩缩容如何进行，即扩缩容的速度是怎样的（如15s增加1个副本)。

5. DNS

   Minik8s需要支持用户通过yaml配置文件对Service的域名进行配置，使得用户可以直接通过域名而不是虚拟IP来访问Service。同时，集群内的Service也可以通过域名而非IP来访问其他Service。此外，还要求支持同一个域名下的多个子路径path对应多个Service。配置文件需要包括以下内容（建议参考kubernetes的Ingress配置）：

   - name&kind：DNS配置的名称和类型。类型应该为DNS
   - host：主路径，由minik8s自动生成
   - paths：子路径，支持path列表。每个path应当包括具体的路径地址，以及对应的service名称和端口

6. 容错

为minik8s的控制面实现容错，即达到以下目标：

- minik8s的控制面发生crash，不影响已有pod的运行
- minik8s的控制面重启后，已部署的service均可以重新访问

7. 支持GPU应用

   本课程将为同学们提供交我算超算平台的访问能力。交我算平台通过Slurm工作负载管理器来调度任务。具体的操作指南如文档所示：

   https://docs.hpc.sjtu.edu.cn/job/slurm.html

   Minik8s能够支持用户编写如CUDA程序的GPU应用，并帮助用户将cuda程序提交至交我算平台编译和运行。

   用户只需要编写cuda程序和编译脚本，并通过yaml配置文件提交给minik8s。minik8s通过内置的server（该server需要minik8s实现）将程序上传至交我算平台编译运行，将结果返回给用户。

   需要保证不同任务之间的隔离性，上传不同名字的任务时应当使用不同的server进程来提交任务给交我算平台。（这里的建议是，可以模拟kubernetes的Job类型，将GPU程序放在pod内，pod内内置用于提交任务的server。）

   配置文件除了基本的name、kind等信息外，还需要包括任务的一些配置信息。这些配置信息和slurm脚本内的配置信息对齐。具体的，配置文件的字段可以自行设计。

   GPU应用需要实现一个cuda编写的矩阵乘法和矩阵加法程序，并且利用硬件的并发能力。在答辩验收的时候，需要从代码层面对如何利用CUDA进行讲解。

8. 多机minik8s

   minik8s最终需要在多机上实现容器编排的功能，即支持>1台机器加入集群。支持多机具体需要支持以下功能：

   a. 支持Node抽象。新的node可以通过配置文件，向现有集群的控制面（如kubernetes中的API-server）注册加入集群。配置文件字段自行设计。同时，可以通过get node等指令获得node的基本信息，包括node名字，node状态等。

   b. 支持scheduler调度pod。pod在启动的时候，minik8s应当首先询问scheduler的调度策略，将pod分配到适合的node上。调度策略的实现可以是和pod配置无关的简单策略，如round robin，也可以是和pod的配置相关的（例如pod A在配置中指定不和pod B运行在同一台机器上，再比如pod A对某种资源有特别要求），实现最简单的调度策略即可拿到该功能>80%的分数。

   c. Service的抽象应该隐藏pod的具体运行位置，即pod无论运行在何处，都可以通过service提供的IP访问到。

   d. Deployment和scale-out的实例均可跨多机部署。

自选功能

除了基本功能，minik8s还需要实现至少一个自选功能。

Microservice

基于minik8s实现简单的Service Mesh（参照Istio），提供更强的流量管控功能，从而无侵入式支撑的microservice架构，具体要求如下：

9. 对Pod流量进行拦截：在Pod的基础上以sidecar的架构实现网络代理，拦截所有进出pod的流量以实现后续流量管控功能
10. 支持自动化服务发现：Service Mesh应当支持利用minik8s提供的API，自动发现部署中所有Service和Pod，并告知每个Pod中的网络代理，使得被劫持后的网络流量仍然能够按照minik8s的定义正常的得到分发
11. 支持高级流量控制功能，包括：
    a. 灰度发布（自定义API，使得用户可以通过制定一系列规则的方式，控制进入同一个Service的流量按照不同规则被定向至不同Pod，从而达到灰度发布的效果。规则应支持：按照一定配比分配流量和按照正则表达式匹配结果分配流量）
    b. 滚动升级（自定义命令行接口，使得某一Service可以在不停机的情况下完成对内部每个Pod的升级过程）
12. 自行实现简单的microservice应用，或部署开源的microservice应用以展示上述功能

Serverless

基于minik8s实现Serverless平台，能够提供按函数粒度运行程序，并且支持自动扩容（scale-to-0），并且能够支持函数链的构建，并且支持函数链间通信。强烈建议参考现有开源平台，如Knative、OpenFaaS的实现方式。具体要求如下：

13. 支持Function抽象。用户可以通过单个文件（zip包或代码文件）定义函数内容，通过指令上传给minik8s，并且通过http trigger调用函数。

    函数需要至少支持Python语言

    （函数的格式，return的格式，update、invoke指令的格式通过knative或者openwhisk来演示）
14. 支持Serverless Workflow抽象：

    用户可以定义Serverless DAG，包括以下几个组成成分：
    a. 函数调用链：在调用函数时传参给第一个函数，之后依次调用各个函数，前一个函数的输出作为后一个函数的输入，最终输出结果。
    b. 分支：根据上一个函数的输出字段，控制面决定运行哪一个分支的函数。

    Serverless Workflow可以通过配置文件来定义，参考AWS StepFunction或Knative的做法。除此之外，同学们也可以自行定义编程模型来构建Serverless Workflow，仅需要workflow能达到上述要求即可。
15. Serverless的自动扩容（Scale-to-0）

    Serverless的实例应当在函数请求首次到来时被创建（冷启动），并且在长时间没有函数请求再次到来时被删除（scale-to-0）。同时，Serverless能够监控请求数变化，当请求数量增多时根据相应Policy能够自动扩容至>1实例。

16. 实现Serverless应用（Function Workflow）以展示上述功能

考核方式

该Lab自由度较高，minik8s的实现不对编程语言、实现方式等进行限制（虽然还是强烈建议参考kubernetes原本的实现方式），主要通过答辩进行考核，对项目的功能进行验证。

阶段性考核

该lab要求分多次迭代完成，并且会阶段性组织答辩考核，一共包括**一次过程答辩**和**一次最终答辩**。

过程答辩主要是通过助教判断一下流程进度，对流程进度进行评价。过程答辩之后，需要提交一个中期文档，汇报完成进度。

最终答辩需要完成所有功能。

评分标准：功能要求 80% + 工程要求 20%。

组织/工程要求

- **禁止抄袭！抄袭者严格0分处理！**
- 该lab为3人小组合作完成，并且指定一人为组长。自由组队
- 中期DDL暂定五月初，结题DDL暂定六月初。
- 组员内部必须明确分工。在项目开始时，开题需要指定每一次迭代的任务内容划分，人员的分工安排。中期答辩需要介绍每次迭代的完成情况，介绍实际的人员分工，以及对进度进行评估。每一个迭代结束后，按需对下一个迭代的计划进行微调。
- 按照开源社区的标准流程开发：每个功能需要通过git分支单独构建，实现完成后通过PR的方式融入主分支中。
- 使用gitee private仓库来存放代码，把两位助教加入协作者中。
  - 助教柳清源：邮箱lqyuan980413@163.com
  - 助教赵子铭：邮箱dumplings_ming@sjtu.edu.cn
- CI/CD：git push需要通过CI/CD中的测试。CI/CD可以任选框架（travis，Jenkins等均可）
- Best practice参考文档例子：
  - 开源项目管理，Javascript项目最佳实践：https://github.com/elsewhencode/project-guidelines
  - Go项目Layout：https://github.com/golang-standards/project-layout

开题文档要求

开题报告通过pdf格式提交，需要包括以下内容:

- 人员组成：组员的姓名、学号信息，组长指定。
- 选定的可选题目内容
- 任务的时间安排，将时间分成几次迭代，指定每次迭代需要完成的任务有哪些。

- 人员分工
- gitee目录