云操作系统: Minik8s Lab-2024版

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。所使用的容器技术鼓励同学们使用containerd，也可以使用docker等容器技术。

小组作业

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文件来对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。如果Pod中的容器发生崩溃，Minik8s需要将Pod重启。

   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）
   - volume：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CNI功能，支持Pod间通信

Minik8s需要支持Pod间通信，实现形式可以参考k8s，以插件形式存在(例如Flannel，Weave等)，也可以集成在Minik8s当中，具体应实现如下功能：

- 在Pod启动时为pod分配独立的内网IP
- Pod可以使用分配的IP与同节点上的Pod和不同节点上的Pod通信

3. 实现Service抽象

   Minik8s应当支持Service抽象。Service需要支持多个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，推荐使用IPVS的方式控制流量的转发。Service可以看作一组pod的前端代理。

   Service通过selector筛选包含的pod，并将发往service的流量通过随机/round robin等策略负载均衡到这些Pod上。

   在符合selector筛选条件的Pod更新时（如Pod加入和Pod被删除），service应该动态更新（如将被删除的Pod移出管理和将新启动的Pod纳入管理）。

   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。

   用户可以在集群外通过NodePort或LoadBalancer的方式访问集群内部的服务。

4. 实现Pod ReplicaSet抽象（或者Deployment）

   Minik8s的可以通过Deployment(ReplicaSet)抽象对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，使得ReplicaSet对应的pod数量恢复到replica指定的状态。

   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。

   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。

5. 动态伸缩 (auto-scaling)

   除了ReplicaSet的固定replica数量，Minik8s也可以通过auto-scaling ReplicaSet抽象根据任务的负载对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auto-scaling ReplicaSet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。推荐使用cadvisor(https://github.com/google/cadvisor)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)。

6. DNS与转发

Minik8s需要支持用户通过yaml配置文件对Service的域名和路径进行配置，使得集群内的用户可以直接通过域名与路径的80端口而不是虚拟IP来访问映射至其下的其他Service的特定端口。同时，集群内的Pod可以通过域名与路径的80端口访问到该Service。其中，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，但本要求和kubernetes的Ingress并非一致）：

a. name&kind：DNS配置的名称和类型。类型应该为DNS

b. host：域名主路径，由用户在yaml内指定。

c. 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

i. 例子：若用户先创建了一个Service，那么用户可以使用ServiceIP:Port来访问该Service。在配置DNS和转发时，host为用户定义的域名，假设为example.com，service的名称和端口和该Service对应。设置完之后，用户和Pod内可以通过example.com:80/path来访问该Service，起到和ServiceIP:Port相同的效果。

7. 容错

为minik8s的控制面实现容错，即达到以下目标：

a. minik8s的控制面发生crash，不影响已有pod的运行

b. minik8s的控制面重启后，已部署的service均可以重新访问

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**3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。

自选功能要求

除了基本功能，minik8s还需要实现至少一个自选功能，最终答辩仅考核一个自选功能，如果实现多个，小组只能选择一个功能进行答辩。**注意：**选做功能不是附加分，直接作为100分成绩的组成部分。

MicroService 30.04 class last 10 min

对minik8s的网络功能进行扩展，除开经典的service架构，用户还可以在启动minik8s的控制面时指定改用Service Mesh架构来管理网络，从而在不修改业务逻辑的前提下，提供更强的流量管控功能，以支持microservice架构（可以参照目前主流的Service Mesh开源框架Istio的实现），具体要求如下：

1. 对Pod流量进行劫持：使用sidecar架构实现网络代理，拦截管理所有进出pod的流量。
2. 实现服务注册中心：对外提供Service的虚拟ip。用户能够通过在服务注册中心查询到的虚拟ip访问Service，由service mesh将请求具体转发至对应的pod。
3. 支持自动化服务发现：Service Mesh应当支持自动发现部署的所有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能够正常地得到分发；当服务中的Pod发生ip发生变化时，服务注册中心的信息能自动更新，Pod中的网络代理能做出相应调整，保证服务仍能够被正常访问（用户可以正常访问服务，不同服务之间能够相互调用）
4. 使用sidecar实现高级流量控制功能，包括：
    - 滚动升级：使用自定义的命令行接口，让Service Mesh以合理的方式分发流量，使得某个Service在不停机的情况下完成对内部每个Pod的升级过程
    - 灰度发布：使用自定义的 API，让用户可以通过定义规则的方式来将同一Service的流量定向到不同的 Pod，从而实现灰度发布；规则应支持按配比分配流量和按正则表达式匹配结果（e.g.，匹配http header或者url）分配流量

5. 找一个较为复杂的开源的Microservice应用或者自行实现一个较为复杂的Microservice应用，该应用必须有现实的应用场景(比如真实的web服务)，至少需要包含三个微服务，不能只是简单的hello world。将该应用部署在minik8s上，基于这个应用来展示Microservice相关功能，并在验收报告中**结合该应用的需求和特点**详细分析该应用使用微服务架构的必要性和优势。

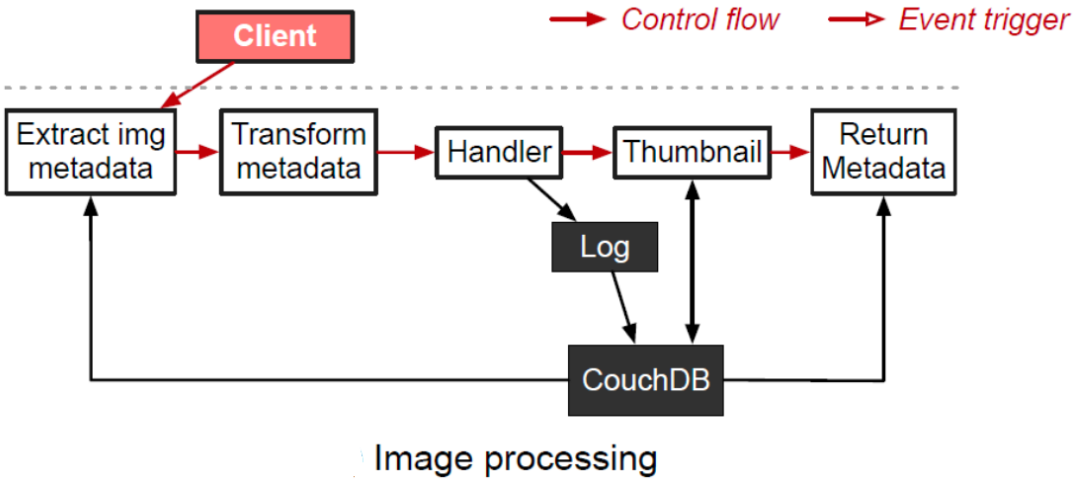
   MicroService应用可参考：GitHub - delimitrou/DeathStarBench: Open-source benchmark suite for cloud microservices 或 GitHub - kubeshark/sock-shop-demo: Sock Shop example with additions

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：

1. 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和event 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指令的格式可以自定义
   - 函数调用的方式：支持用户通过http请求、和绑定事件触发两种方式调用函数。函数可以通过指令指定要绑定的事件源，事件的类型可以是时间计划、文件修改或其他自定义内容
2. 支持Serverless Workflow抽象：用户可以定义Serverless DAG，包括以下几个组成成分：
   - 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。
   - 分支：根据上一个函数的输出，控制面决定接下来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，只要workflow能达到上述要求即可。
3. 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，当请求数量增多时能够自动扩容至>1实例。
   - Serverless应当能够正确处理大量的并发请求(数十并发)，演示时使用wrk2或者Jmeter进行压力测试。
4. 找一个较为复杂的开源的Serverless应用或者自行实现一个较为复杂的Serverless应用，该应用必须有现实的应用场景(比如下图所示的 Image Processing)，不能只是简单的加减乘除或者hello world。将该应用部署在minik8s上，基于这个应用来展示Serverless相关功能，并在验收报告中**结合该应用的需求和特点**详细分析该应用使用Serverless架构的必要性和优势。

Serverless样例可参考Serverlessbench中的Alexa：ServerlessBench/Testcase4-Application-breakdown/alexa at master · SJTU-IPADS/ServerlessBench



个人作业

1. 持久化存储

   Minik8s需要实现持久化存储，即当Pod（例如，运行Mysql）发生crash或被kill掉时，pod中的应用数据仍然存在，并且能够迁移至新起的pod当中。具体需要支持以下功能：

   a. 支持PV和PVC抽象。可以通过动态和静态两种方式创建PV。静态方式：集群管理员可以预先手动创建PV；动态方式：Minik8s能够根据PVC自动创建符合要求的PV。

   b. 创建Pod时能够使用yaml文件创建PVC，实现Pod与PV代表的真实存储资源之间的绑定。Pod被删除之后即与PV解绑，但对应的数据仍然存在，重启一个新的Pod与该PV绑定后仍能访问原来的数据。

   c. 持久化存储也需要支持多机，即在其他节点上重启一个新的Pod与原先Pod的PV绑定后仍能访问原来的数据。

2. 支持GPU应用

   a. 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

   b. 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。

   c. 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。

   d. 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）

e. 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。

f. 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。

3. 日志与监控 (可观测性)

   Minik8s需要利用Promethus实现简单的日志和监控功能，以提高系统的可观测性。具体的功能要求如下：

   a. 实现对集群中各节点的资源的监控。Minik8s应当能够采集集群中每个节点(主机)的运行指标，如CPU、内存、网络等信息，并通过Promethus UI显示各节点的资源。

   b. 实现对用户程序的日志监控。用户可以通过Promethus Client Library在自己的程序中自定义需要采集的指标，主动暴露相应的信息。创建一个Pod运行该程序，集群中的Promethus应当能够监控到这些用户自定义的指标，并显示在Promethus UI。实现基于Pull的模型即可。

   c. 实现集群中Promethus的自动服务发现。基于Pull的模型需要知道所有被监控对象的访问信息，从被监控的对象那里拉取监控数据。为了更好地实现上述两个功能，需要让集群中的Promethus自动发现Node的加入和退出、Pod的创建和删除，在不重启Prometheus服务的情况下动态地发现需要监控的Target实例信息。

      i. 当新的节点加入集群时，集群中的Promethus可以自动监控新加入的节点。当节点退出集群时，自动停止对该节点的监控。

      ii. 当用户创建一个Pod，且其中的程序主动暴露自定义的Promethus指标时，集群中的Promethus可以自动监控Pod程序中自定义的指标。当Pod被删除时，自动停止对这些用户自定义指标的监控。

   d. 利用Grafana为集群监控提供更丰富的可视化展示方式。

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（但还是强烈建议参考kubernetes原本的实现方式），主要通过答辩进行考核，对项目的功能、性能、可靠性等进行验证。

项目要求的功能分为小组作业和个人作业，**小组作业**分工由各个小组自行决定，共同答辩，结合小组集体得分和最终贡献度给分。**个人作业**要求小组成员每人实现一个功能，最终答辩时进行单人答辩，单独赋分。

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！（说明：所有最终的代码会经过查重软件审核，并且会比对所有往届的提交代码。只要检测到有抄袭，不管是被抄袭者还是抄袭者，本次课程0分）。**
- 该lab为3人小组合作完成，并且指定一人为组长。自由组队
- 开题DDL为4/17 23:59，中期DDL暂定劳动节后，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。最终提交的验收报告中需要指定每个成员的贡献度，助教还会根据代码仓库的修改记录修正每个人的贡献度(**git commit时请使用自己的名字**)。
- 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，把对应助教加入协作者中。
  - 助教马骁骞（1-5组）：邮箱maxiaoqian2023@outlook.com
  - 助教李昱翰（6-10组）：邮箱7143192@sjtu.edu.cn
  - 助教刘容川（11-15组）：邮箱rongchuan_liu@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，也可以使用gitLab进行CI/CD）
- 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:

- 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