

Minik8s Lab-验收指南

验收文档要求

由于答辩时间受限，因此，本次Minik8s Lab需要编写验收文档，以便充分展示与介绍整个项目。项目验收文档的不以文档长度和字数评分，但项目验收文档需清晰地描述以下内容：

1. 项目的总体架构与各个组件的功能和实现的软件栈
2. 项目各组员分工和贡献度占比
3. 项目的gitee目录地址、项目的各分支介绍、CI/CD介绍、软件测试方法介绍、以及项目所遵循的新功能开发流程介绍
4. 项目**所有实现的功能**和简要使用方法（特别是项目验收答辩演示过程中时间原因未演示到的功能）

文档内容着重补充演示时未能演示到的部分，如边界情况处理，鲁棒性扩展等，作为功能完善性考量的重要补充。例如，能否在节点中删除Node？Service的selector能否指定多个Pod？Scheduler是否支持多种调度策略？Service对应的Pod发生了重启后如何处理等。

验收文档的截止时间为6月5日23:59，提交方式为通过canvas提交

验收答辩要求

本次Minik8s Lab主要通过演示功能要求实现情况进行评分。功能演示总时间为**15min**，方法可以是现场演示或者录屏。若演示基于录屏，需要准备好环境以便随时现场演示，同时我们在演示过程中随时会对实现方式等提问。功能演示的流程请参考后文所介绍流程。

功能演示开始之前，需要严格在3分钟时间内简单介绍项目架构，主要介绍项目包括了哪些组件，软件栈是什么，用了哪些库等。之后开始演示完成的功能情况。

除后文所述功能演示流程外，本次Lab鼓励大家根据实现过程中发现的需求和问题，自行添加附加功能（如命令行错误输入检测、各种配置的更新及处理方式、容器不可启动情况下的处理、各种边界情况、跨子网的多机集群等），如有实现，也将面对的问题和具体功能在答辩过程中演示。

答辩时间为6月3日全天，每小组依次答辩，具体答辩时间安排详见后续时间安排表

基本功能

1. 部署多机minik8s
 - a. 演示对Node抽象进行配置和操作的流程与运行情况
 - 演示配置文件时，需演示自行设计的Node配置文件相关接口，并简述各字段含义
 - 演示运行状况时，需演示如何通过minik8s的命令行接口（后文中用minikubectl代称）的相关命令，将全新的计算节点添加到minik8s的集群中，并演示如何利用minikubectl的接

口获得Node的状态

- 整个演示集群中应至少包含两台计算节点，一台中运行minik8s的管理程序和minik8s所部署的实际容器（即既作为master节点，又作为worker节点），另一台中则只运行minik8s所部署的实际容器（即仅作为worker节点）

2. 实现Pod抽象，对容器生命周期进行管理

- a. 演示利用配置文件创建包含多容器Pod（若未实现多容器Pod，则演示单容器Pod）的配置文件与运行状况（若所有功能均已实现，可将此项与后续b项演示所需的配置文件合并）。
 - 演示的Pod配置文件中需包含：配置种类（kind）、Pod名称（name）、容器的镜像名称与版本、容器镜像所执行的命令、对容器资源用量的限制、容器所暴露的端口，且单Pod内应包含多个容器
 - 演示运行状况时可采用minikubectl、docker等指令展示创建的Pod、容器以及对容器各种参数的配置，以演示和每个参数的效果
- b. 演示在同一Pod的多个容器间利用localhost进行相互通信的运行情况
 - 演示需创建包含多容器的Pod，并自行设计场景演示同Pod内跨容器利用localhost进行本地网络访问
- c. 演示对Pod进行多机调度时的运行情况，并介绍调度策略
 - 演示时应利用minikubectl在多机环境下创建新的Pod，利用minikubectl的接口展示所创建的Pod被部署在的节点，并介绍该节点分配所基于的调度策略（调度策略应至少有一种）
- d. 演示利用volume接口对同一Pod内的多个容器实现文件共享的配置文件与运行状况
 - 演示的配置文件中需包含利用volume接口实现共享文件的接口设计
 - 演示运行状况需自行设计场景，并利用命令行进行演示（如容器A中创建修改文件可被同一Pod下的容器B所访问等）

3. 实现Service抽象

- a. 演示利用配置文件创建Service的配置文件及运行情况
 - 演示的Service配置文件中需包括：配置种类（kind）、Service名称（name）、Service对Pod的筛选（selector）、Service暴露的端口（ports）
 - 通过minikubectl之类的命令行工具创建Service完成后，可以通过minikubectl get/describe svc之类的指令展示创建的Service的运行状态（IP，端口等）
 - 被演示Service底层的Pod应当是一个可以被访问的应用，如Nginx或简单的echo server之类，便于后续演示。

- b. 演示Service的单机可访问性

时间原因，如果能够达到多机可访问性，则可以跳过这一步。

- 在集群内通过curl之类的方式，通过IP访问本机的Service，演示需要展示出能够访问到应用并获得相应的返回
- 在集群内的其他Pod，通过虚拟IP，能够发送请求并访问集群内的其他本机Service，演示场景可以自行设计

c. 演示Service的多机可访问性

- 在集群内通过curl之类的方式，通过IP访问其他节点上的Service，演示需要展示出能够访问到应用并获得相应的返回
- 在集群内的其他Pod，通过虚拟IP，能够发送请求并访问集群内的其他节点上的Service，演示场景可以自行设计

d. 演示Service对多个Pod的映射

- 创建新的Service，selector的设置使得Service能够映射多个Pod
- 能够通过Service的虚拟IP访问到多个Pod。区分Pod的方式，可以用不同的端口，或者端口相同的情况下有默认的负载均衡工具，二者取其一即可。

4. 实现Pod ReplicaSet抽象（或者Deployment）

a. 演示利用配置文件创建ReplicaSet或Deployment（后文采用Deployment作为统称）的配置文件与运行状况

- 演示的Deployment的配置文件中需至少包含：Deployment的唯一标识符、Deployment对应的Pod、Replica的数目
- 演示运行状况时需演示如何使用minikubectl创建Deployment，并利用minikubectl指令展示创建的Deployment与Pod，其中，创建的Pod应能够均匀分布在多个节点上

b. 演示将Deployment绑定至Service的配置文件和运行状况

- 演示的配置文件需包含如何将Deployment绑定至Service上的配置
- 演示运行状况时需自行设计场景，展示如何使用minikubectl将Service与Deployment映射至一起，从而使得访问Service的流量能够以一定负载均衡策略被分配到同一Deployment内的位于多个节点的不同Pod中

c. 演示Deployment中Pod停止运行时，Deployment进行恢复的运行状况

- 演示时需自行设计场景（如某一Pod中的某一容器由于内存用量超出资源上限而导致被终止），展示正在运行的Pod的数目的变化过程

5. 动态伸缩 (auto-scaling)

a. 演示利用配置文件创建水平扩容（后文简写为HPA）配置，能够对Service中的Pod进行动态扩容

- 演示的HPA配置文件中需包含：HPA的唯一标识符（name），kind，扩容的目标workload（对象是Service中的Pod或ReplicaSet或Deployment，后面将HPA目标统称为Pod），扩容的minReplicas和maxReplicas，以及扩容的metrics（至少包括CPU）。

- 配置完成后能够通过minikubectl get之类的指令查看到配置的HPA
- 自行设计测试场景，使得HPA的目标Pod能够增加或减少负载，从而能够触发扩缩容条件
- 扩缩容所新创建的Pod应能够分布在不同节点中

b. 演示扩缩容策略之一：扩缩容时机

- 扩容：给HPA的目标Pod增加负载，当负载达到扩容策略metrics规定的值时，增加Pod数量直到maxReplicas
- 缩容：给HPA的目标Pod降低负载，当负载降低至规定值时，减少Pod数量直到minReplicas
- 对能够支持的metrics，简单介绍minik8s是如何对metrics进行监控的，并且简单介绍minik8s如何通过监控的metrics执行扩缩容命令的

c. 演示扩缩容策略之一：扩缩容速度

- 在演示扩缩容时机时，如果扩缩容策略包括了扩缩容速度，那么一同演示，即在部署的时候在配置文件中配置好扩容速度的标准，在扩容现象发生的时候简单衡量扩容的速度，说明扩缩容的速度符合规定

d. 演示扩缩容后访问目标Pod：

- 当Pod发生扩容后，演示通过Pod的Service的IP能够通过一定的负载均衡策略访问到扩容后的所有Pod，不可以只增加Pod的数量但是扩缩容后的Pod无法访问

6. DNS与转发

a. 演示利用配置文件对DNS与转发规则进行配置的配置文件与运行状况

- 演示的配置文件中需包含：配置名称（name）、配置类型（kind）、主路径（host）、子路径（path），以及转发的目标Service，且配置文件中应包含多个对应不同Service的子路径
- 演示运行状况时需自行设计场景，展示利用minikubectl创建/获取DNS与转发配置；展示分别在集群中的Pod内部和集群中的宿主机上，通过DNS和转发功能访问域名所映射的Service提供的服务；

7. 容错

a. 演示Pod和Service的容错：

- 启动一个Pod和Service，然后对minik8s的控制面（api-server, controller, scheduler, 该节点的kubelet, 不包括etcd）全部进行重启，重启过程中Pod能够正常运行
- 重启之后，仍然能通过minikubectl get之类的指令查看到在所有节点上已部署的Service和Pod
- 重启之后，仍然能正常访问上述已部署的Service

8. 支持GPU应用

a. 演示任务的提交：

- 准备cuda程序和编译脚本，通过yaml配置文件提交给minik8s。
- yaml文件的格式比较灵活，必须字段只有name、kind，任务的配置信息可以自行设计，包括slurm脚本要求的一些配置信息。
- 提交之后，能够通过minikubectl get之类的命令得到任务的提交情况
- CUDA程序需要是矩阵乘法和矩阵加法程序，需要展示代码并简单介绍如何利用GPU的并发能力

b. 演示获取GPU任务的返回结果

- 演示minik8s的用户如何通过minikubectl的相关命令来得到返回结果。如果演示时任务一直处于pending状态，则推迟到演示的最后查看GPU上cuda程序的返回结果。如果仍然pending，那么就跳过这一步。

自选功能

Microservice

1. 演示如何将现有的微服务应用或自行编写的微服务应用部署在minik8s中

- 演示时应利用架构图，简要介绍微服务应用总体架构（包含整个应用面向的场景、服务数目、服务间通信关系，服务与Pod和Service的对应关系）
- 演示时应利用minikubectl，展示部署完成后的Pod与Service运行情况
- 后续所有演示均基于这一微服务应用进行

2. 演示利用Sidecar架构的代理进行流量拦截的运行状况

- 演示时需展示如何将Sidecar模式的代理利用配置文件或命令行接口，注入到现有Pod中的配置方式，并利用代理程序的输出或log文件等方式展示所有流量确实已被Sidecar代理程序劫持和处理

3. 演示流量拦截后，和现有minik8s功能（Pod、Service）的兼容性

- 演示时应展示注入sidecar代理后，现有的Pod间通信和利用service的通信仍然能够正常运作

4. 演示灰度发布配置和运行状况

- 演示时应展示灰度发布的配置文件，简要介绍对配置文件API的设计，并展示应用配置文件的方法。配置文件的API中，应能够支持：按照比例分配流量、按照对网络包内容进行正则匹配的结果分配流量
- 演示时应自行设计场景，展示可以通过可基于制定的规则，达到正确分配流量、灰度发布的效果

5. 演示滚动升级配置和运行状况

- 演示时应展示对滚动升级命令行接口的配置，配置文件中应至少包含对滚动升级过程中可用性（即在滚动升级过程中应至少保证可用的Pod数目）的配置接口
- 演示时应展示整个对滚动升级的过程，并简要介绍滚动升级时，创建、删除Pod的策略

Serverless

1. 演示Function的定义和运行流程

- 用户定义函数内容，通过指令上传给minik8s，上传之后，minik8s能够通过minikubectl get之类的指令查看到函数。函数至少支持Python语言
- 上传之后，用户能够对函数进行调用（invoke），传入参数，并得到返回结果。函数的逻辑要能够反映出参数信息。
- 用户上传的每一个函数需要在独立的Pod内运行保证隔离性

2. 演示Workflow定义和运行流程

- 用户展示Workflow的定义文件（如yaml配置文件或其他方式），展示Workflow的定义和上传过程。展示Workflow内的各个函数，函数的逻辑可以非常简单，建议设置辨识度以区分每个函数在workflow中的位置，同时也要体现出输入参数值对返回结果的影响（不可以出现“无论传入参数是多少，返回结果都是一样”的情况）。
- 介绍Workflow支持怎样的分支/条件
- 运行Workflow，演示workflow如何运行一条分支上的所有函数的。要求前一个函数的输出作为后一个函数的输入，体现在下游函数的计算过程中。

3. 演示函数的自动扩容和scale-to-0

- 演示当一段时间内没有新的请求到来时（如30s或1min），函数实例会被清除。
- 演示当不存在函数实例时，首次调用函数会自动生成新的实例
- 演示当请求并发数（RPS）增多时，serverless能够扩容至多个实例，并且请求可以发送至这些实例中的任意一个进行处理。简要介绍扩容策略。

4. 演示Function的更新和删除

- 对某一个函数（以函数名或者id来唯一标识），如果有新的函数代码，可以通过update之类的指令对函数进行更新。
- 可以通过delete之类的指令对函数进行删除