

# Minik8s结题报告

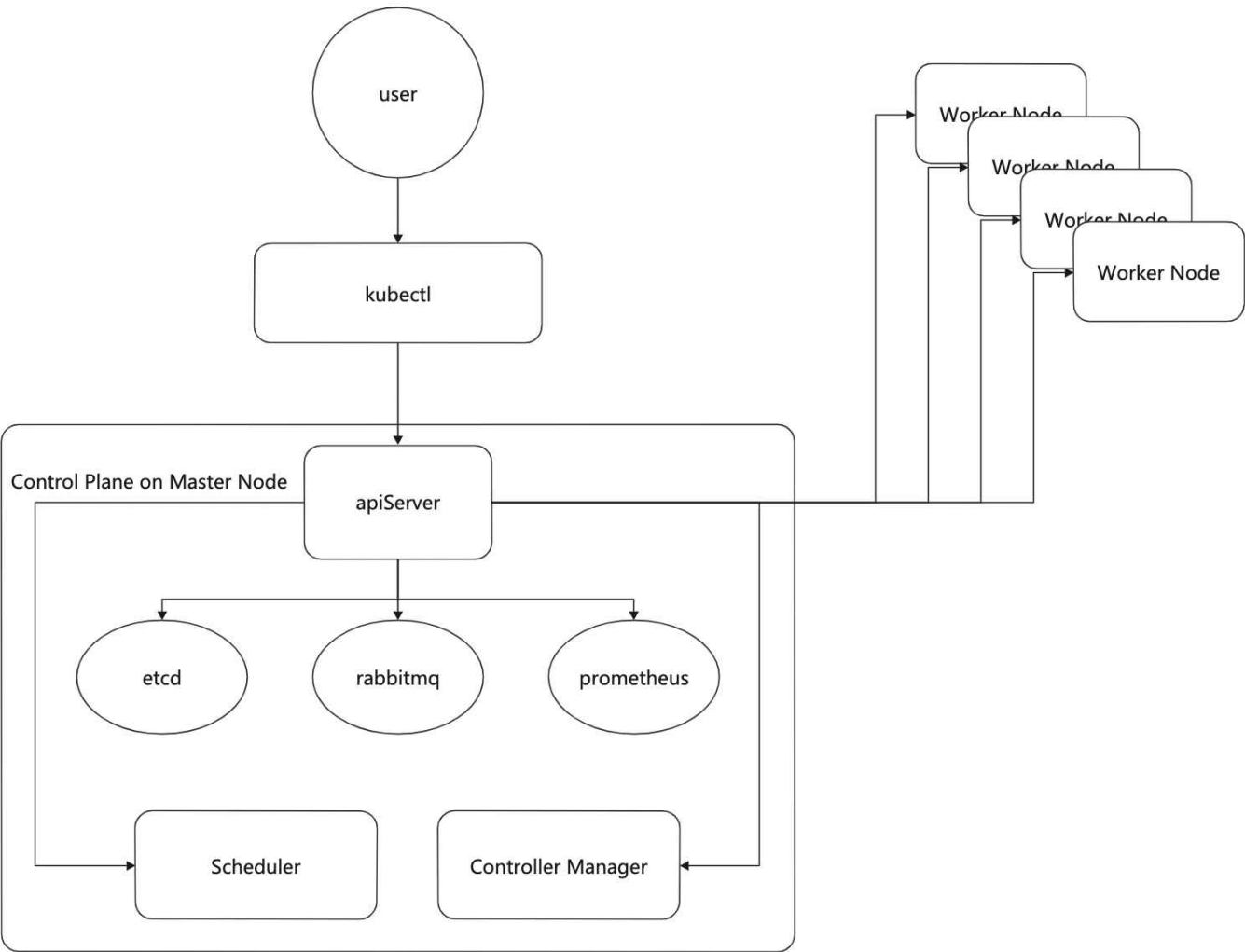
Minik8s第七组

## 1. 人员情况

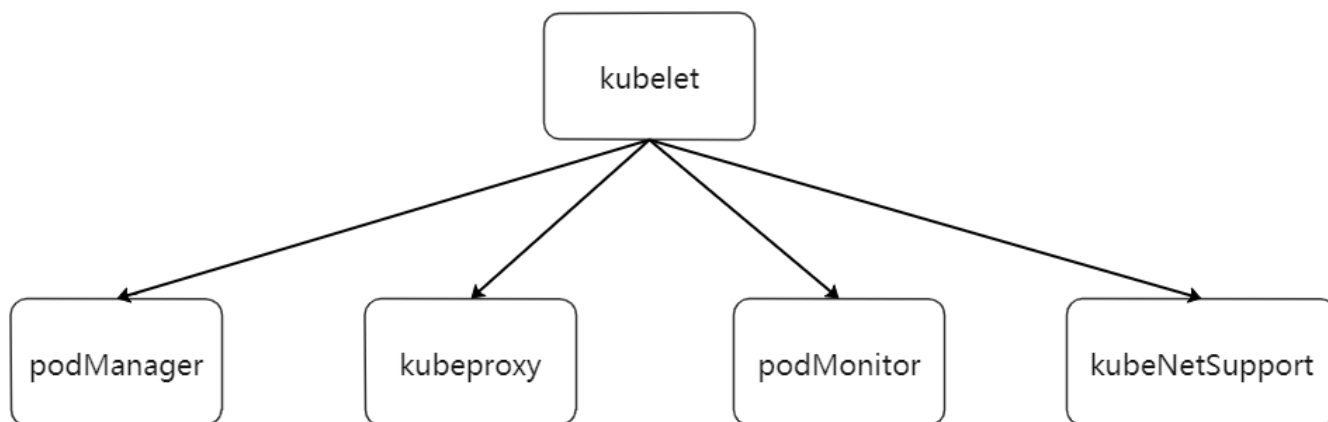
姓名	贡献度	人员组成
王浩天	1/3	组长
周昱宏	1/3	成员
钱博闻	1/3	成员

## 2. 项目架构

Minik8s采取主从架构，Master节点部署apiServer、Scheduler和Controller Manager对整个集群进行控制，使用etcd保存集群状态，使用rabbitmq作为消息中间件实现消息传递，使用prometheus监控Worker Node状态。



Worker Node的核心组件是kubelet，kubelet负责pod的生命周期管理，也负责网络的搭建。



## 3. 功能

### 3.1 Controller介绍

Controller Manager负责管理各种Controller，实现集群状态的维护和特殊功能的实现。在kubernetes的官方实现中，controller的类型多种多样，为了简化设计，我们根据需求选取了一部分controller进行实现。

#### 3.1.1 Replicaset Controller

Replicaset作为k8s中至关重要的对象，控制着pod的副本数量，维护集群状态的关键一环。我们实现了Replicaset Controller来控制Replicaset。

Replicaset Controller运行了一个syncLoop，不间断地向apiServer轮询，获取当前pod的运行状态。当检测到某一个Pod的状态发生变化，会根据Pod中的字段找到其所属的Replicaset，来创建或删除Pod，使当前的replica数量符合用户配置的replica预期数量。

#### 3.1.2 Deployment Controller

Deployment Controller负责管理Deployment对象，并且增添对滚动升级的支持。Deployment Controller会watch deployment资源，基于事件驱动进行deployment对象的管理。Deployment并不直接管理pod而是通过对replicaset的操作间接地管理pod，以达到预期pod数量的管理。

Deployment与Replicaset最大的不同点在于，Deployment添加对于RollingUpdate的支持。当Deployment配置文件发生变更，pod需要进行升级时，Deployment Controller并不会骤然增加或减少pod的数量，造成某一时刻服务不可用的情况，而是将pod的数量进行缓慢调整，保证服务的可用性，以用户配置的速率进行新旧pod的替换。

#### 3.1.3 Autoscaler Controller

为了减少资源浪费，同时保证能够在流量尖峰增加pod实例的数量，我们实现了Autoscaler抽象。Autoscaler Controller监听Autoscaler资源的变化来创建和删除Autoscaler，Autoscaler管理的对象是Replicaset或Deployment，通过轮询方式监控Replicaset和Deployment的资源利用率，在资源利用率低时减少Pod实例的数量，而在资源利用率较高、达到用户设定阈值时，创建新的Pod实例。通过匹配资源利用率和Pod实例的数量，我们可以实现对资源的高效利用。

#### 3.1.4 Job Controller

为了支持GPU应用的实现，我们提出了Job抽象，并且基于此实现了Job Controller。这里的Job和Kubernetes的Job概念有所不同，仅支持以交大云计算平台的计算任务，存在一定的局限性。

GPU Job的实现思路是：

- 用户打包计算任务的源代码，编写Job配置文件。在配置文件中，用户需要指定打包文件的绝对路径、Slurm调度器的参数等信息。
- 应用配置文件，系统会自动读取打包后的源码并上传到数据库，Worker Node在硬盘上划分出一块共享区域，实时更新用户的上传文件。
- Job Controller检测到新的Job请求，创建一个Remote-Runner Pod，通过卷映射的方式读取文件内容。Remote-Runner内置一个ssh服务器，可以实现远程运行和scp功能，基于这些功能可以方便地在远程运行GPU应用，并将计算结果传输回Worker Node。

### 3.1.5 Service Controller

Service Controller主要做的工作是根据service的selector，动态选取后端的pod。每隔一段时间，Service Controller会轮询已经被选取的pod，通过apiserver获取其运行时信息（存储在etcd中），如果有pod出现异常，则会调用selector重新选取新的pod。同时，如果service对应pod的数量等于0或者小于上限，每隔一段时间，也会触发selector进行pod的选取。而service的信息一旦更新，worker端的kubeproxy会监听到对应事件，从而修改本地的iptables。

基于上述策略，service能够实现后端pod的自动扩容以及容错（即某个pod出错之后service会自动去除该pod，pod数量为0时service会被标记为error，暂时从iptables中去除）和异常恢复（当前时刻没有选取到pod的话只要后续有符合的pod启动那么service会恢复）。

此外，为了支持dns与转发，service Controller在启动的时候便会创建一个DnsService，service Controller同时进行的工作是会根据dns的配置文件，选取service，同时负责创建gateWayService以支持转发。

## 3.2 Kubelet

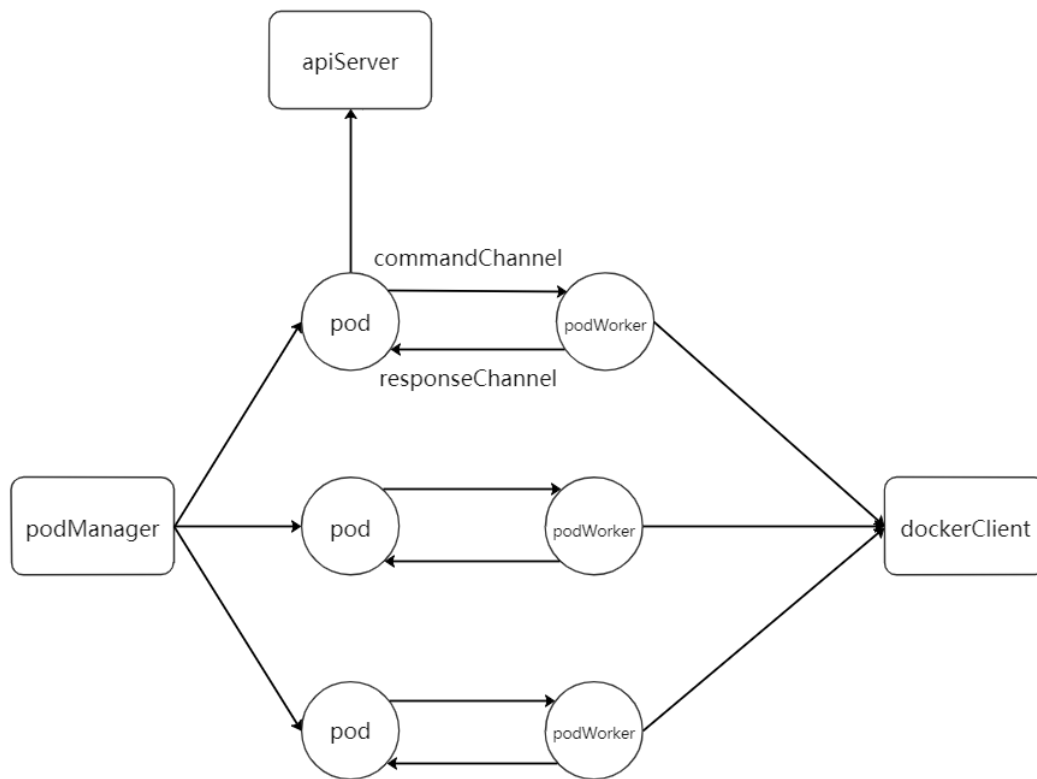
kubelet运行在worker节点，集成了worker节点需要的所有功能，具体来说，包括节点注册、pod管理、service等等，用户仅需要运行一个kubelet即可把节点加入minik8s的集群中。由pod-Maneger，kubeproxy，podMonitor，kubeNetSupport四个组件组成。

kubelet的运行可以指定节点的配置文件也可以缺省执行，用法如下

```
1 ./kubelet 或者 ./kubelet node.yaml
```

### 3.2.1 podManager

podManager的功能在于管理所有的pod（这里的pod指内存中的pod抽象，与实际运行的pod一一对应，主要用于方便管理实际运行的pod），通过维持一个podName到pod的map来实现，提供删除pod，创建pod的接口，其架构图如下：



pod会将命令放入commandChannel，podWorker接收到command之后会调用dockerClient进行对应的容器操作，然后将结果通过responseChannel返回给pod。pod如果有状态的更新也会上传到apiserver。

### 3.2.2 kubeProxy

kubeProxy的主要功能有两部分，一是在监听到etcd中service的配置发生变化时，调整当前节点的iptables以支持对service的访问。二是在监听到Dns及转发的配置信息时，本机做相应的处理。

#### 1. service的支持：

通过修改本机的iptables实现，主要是在nat表中设置一些规则，如Svc链，Sep链等等。这里的实现与k8s相同，不再细述。同时，根据iptables中链以及规则的关系，在代码中也建立了对应的链抽象，与实际的iptables链一一对应，从而便于管理以及修改iptables。

#### 2. dns与转发的支持

首先要了解实现dns与转发的具体方案。在minik8s中，为了达到能够根据域名已经路径访问具体服务的效果，minik8s会在系统启动并且有节点注册上之后，部署一个dns service，用于域名解析，且该服务的clusterIp 固定为10.10.10.10，所有pod的DNS服务器指定为该地址。之后，每当用户配置一个dns与转发。均会生成一个gateWay service，同时在dns服务器中增加一条hostName到gateWay service的clusterIp的记录。

由于部署的服务背后需要pod的支持，且pod会分散到不同节点，因此kubeProxy在这里的主要职责是，检查本地是否有dnsService或者gateWayService的endpoint（pod），并修改其配置信息以支持service层面的修改。这里配置信息的修改通过配置文件的mount支持，使得worker节点只需要更改本机的文件便可以同步到pod中。

### 3.2.3 kubeNetSupport

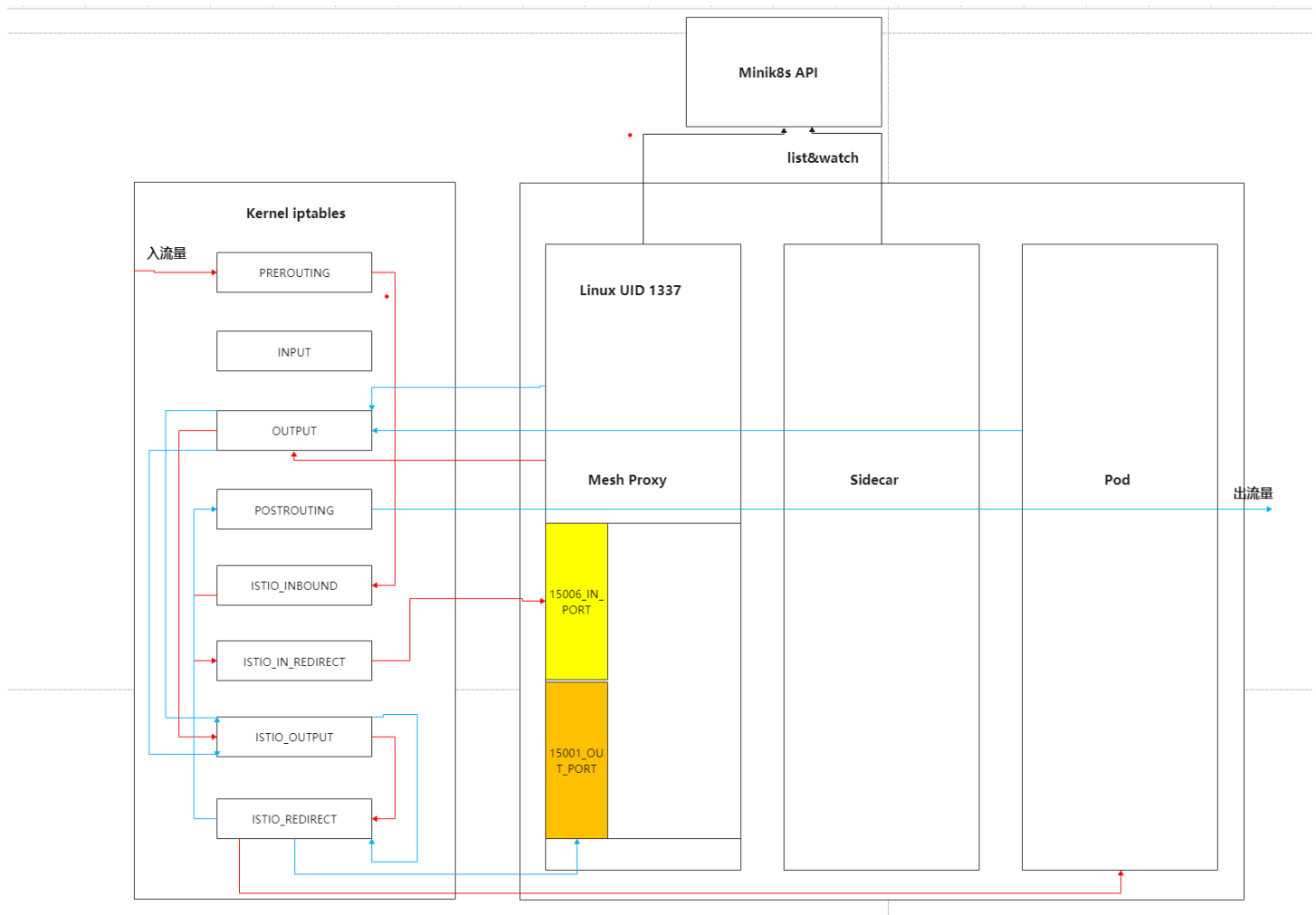
kubeNetSupport的功能比较简单，即实现节点的注册操作。同时获取必要的配置信息来运行flannel插件，实现集群节点注册以及网络互通的自动化操作。

### 3.2.4 PodMonitor

PodMonitor主要实现对于Pod占用内存及CPU资源的监控。通过Docker的client获取基础的信息，利用  $cpuPercent = (cpuDelta/systemDelta) * onlineCPUs * 100.0$  这一公式，计算CPU占用率。并在Node上打点记录。在Master节点，使用Prometheus，获取Pod资源的打点数值。

### 3.3 微服务

微服务主要分为Sidecar和Mesh Proxy两个模块，均运行在Node节点。Sidecar主要监听Master节点是否要求开启Sidecar模式，并更改iptables里的规则。Mesh Proxy主要监听Master节点的规则信息和service信息，配置灰度发布逻辑，接管Pod所有的出入流量，截获网络包并完成转发。架构图如下所示，红色线条为入流量，蓝色线条为出流量。



#### 3.3.1 Sidecar

Sidecar模块实现更改iptables的功能。可以通过配置文件，新增或清空iptables的修改。

- **ISTIO\_INBOUND**: tcp流量进入**ISTIO\_IN\_REDIRECT**链，排除ssh端口的流量
- **ISTIO\_IN\_REDIRECT**: tcp入流量重定向到15006端口
- **ISTIO\_REDIRECT**: tcp流量定向到15001端口
- **ISTIO\_OUTPUT**: 对于目的地非localhost的流量，跳转到**ISTIO\_IN\_REDIRECT**，对于mesh proxy (UID 1337) 以及目的地localhost的流量，不进行重定向。对于其他流量，跳转到**ISTIO\_REDIRECT**

#### 3.3.2 MeshProxy

主要实现流量管控和按规则转发的功能

## 1. 流量管控

获取socket的文件描述符fd，从socket文件中获取原目的地地址，Mesh Proxy再与真正的目的地建立连接，进行双向转发。

- a. 15006端口，管控TCP入流量，Mesh Proxy会将请求的地址转为localhost进行转发。
- b. 15001端口，管控TCP出流量，Mesh Proxy会将使用ClusterIP的请求，根据规则选择一个Endpoint Pod转发请求。

## 2. 规则注入与灰度发布

MeshProxy监听用户的规则配置，和service及pod的变化。不断更新service的pod对应的权重。对于每一个Pod，按照权重的份额，实现类似彩票调度的公平共享的流量分发，实现灰度发布的效果。

### 3.3.3 滚动升级

此部分为Deployment的功能，只需Apply一个新的yaml格式配置文件即可实现滚动升级。详细信息见Deployment Controller的功能介绍。

## 3.4 kubectl

kubectl 是用户的命令行交互工具，用法如下

pod的使用方式：

```
1 kubectl apply pod.yaml      #部署pod
2 kubectl del pod podName     #删除pod
3 kubectl get pod podName     #获取单个pod信息
4 kubectl get pod             #获取所有pod信息
```

service的使用方式：

```
1 kubectl apply service.yaml   #部署service
2 kubectl del service serviceName #删除service
3 kubectl get service serviceName #获取某个service信息
4 kubectl get service          #获取所有service信息
```

replicaset的使用方式：

```
1 kubectl apply replicaset.yaml #部署replicaset
2 kubectl del replicaset replicasetName #删除replicaset
3 kubectl get replicaset replicasetName #获取某个replicaset信息
4 kubectl get replicaset         #获取所有replicaset信息
```

Deployment Controller的使用方式为：

```
1 kubectl apply deployment.yaml
```

dns与转发的使用方式为：

```
1 kubectl apply dnsAndTrans.yaml      #部署dns与转发
2 kubectl del dns dnsName             #删除dns与转发
3 kubectl get dns dnsName             #获取某个dns与转发的信息
4 kubectl get dns                     #获取所有dns与转发的信息
```

Job Controller的使用方式为：

```
1 kubectl apply job.yaml
2 kubectl get job
3 kubectl get job jobName
```

Autoscaler Controller的使用方式为：

```
1 kubectl apply autoscaler.yaml
```

## 4. 补充说明

以下部分为在答辩过程中涉及较少部分的补充说明。

### 4.1 跨子网的多机支持

选取flannel网络插件辅助进行网络配置。在必选功能中，由于我们需要实现多节点的跨子网通信，选择了vxlan模式，在配置Node之间的通道时将public-ip字段设置为自己的公网ip，并且通过公网访问位于Master节点的etcd。这样我们实现了集群的跨子网通信。

### 4.2 Scheduler调度策略

scheduler用于pod的调度，即将pod调度到某个节点上。支持三种调度，第一种是随机调度；第二种是round robin调度；第三种是根据pod的标签和节点的标签进行匹配调度。运行scheduler带上对应数字或名称做参数即可，如

```
1 ./scheduler round-robin    #以rr作为调度策略启动调度器
```

### 4.3 GPU应用的实现

GPU应用通过Job提交，具体流程已经在Job Controller部分介绍过，这里着重介绍一下创建Job Pod时使用到的docker image。



我们使用了[chn1234wanghaotian/remote-runner:6.0](#)，这是我们自己打包的一个docker image，主要运行remote\_runner服务。它内置了一个http服务器监听用户请求，返回任务执行的状态。在启动时指定local path和remote path以及远程服务的认证信息，应用会将本地目录拷贝到远端，然后执行预设的脚本；当收到http请求时，将会执行一次同步操作，将远程目录同步到本地目录。通过这个remote\_runner服务，我们可以轻松地将本地任务提交到远程，并且实时同步任务执行情况。

## 4.4 etcd中相关信息的存储

所有组件对于资源的共享通过etcd实现，同时，为了把控制流和信息流进行一个解耦，在实际信息的存储过程中，做了配置信息和运行时信息的区分。以pod为例，etcd中存储的pod信息，分为pod配置信息以及pod运行时信息。修改pod配置信息意味着修改实际部署的pod（如调度器调度节点需要在pod配置信息中加入节点信息，删除pod需要设置pod配置信息为deleted），kubelet会监听pod配置信息进行对应操作。

pod的运行时信息只能由kubelet进行更新，其他组件只能查看运行时信息。类似的处理同时用在了service和replicaset等存储上。

## 5. 外部依赖

### 5.1 网络配置flannel

选取flannel网络插件辅助进行网络配置。flannel有两种工作模式，分别是vxlan模式和host-gw模式，vxlan模式可以跨子网通信，host-gw速度最快单只能在子网内部通信。我们适配了这两种配置模式，使得minik8s既可以使用vxlan也可以使用host-gw。

在必选功能中，由于我们需要实现多节点的跨子网通信，选择了vxlan模式，在配置Node之间的通道时将public-ip字段设置为自己的公网ip，并且通过公网访问位于Master节点的etcd，这样就可以配置好跨子网的Node直接通信。

在微服务功能中，为了性能，我们使用host-gw模式，将微服务部署的范围限定在同一子网中，这样做的好处是减少微服务之间调用开销，同时限制在同一子网也可以保证服务的安全性，不易暴露给外界遭到attack。

### 5.2 Control-Plane

位于Master节点的Control Plane部署了etcd、rabbitmq和prometheus。

- etcd：高可用的分布式一致性kv store，也是minik8s的核心之一，它可以保证集群状态被持久化存储，也可以在发生部分crash时保证kv一致性。我们用etcd来存储配置文件信息，也会做运行时状态的更新。
- rabbitmq：apiServer的一个重要机制是watch。官方了实现是使用http长连接，我们在这并没有采用http长连接实现watch的机制，而是使用了rabbitmq的发布订阅模式实现的，它可以很好地取代原生kubernetes的SharedInformer和watch，是一种高效的实现。
- prometheus：成熟的监控方案，本项目中用于监控Worker Node Pod的资源利用率。

## 6. 开发流程

### 6.1 gitee介绍

gitee仓库地址：<https://gitee.com/organizations/minik8s/projects>

分支介绍：



<input type="checkbox"/> develop	周昱宏 b017e45 !60 1 5天前	保护分支 ▾	
<input type="checkbox"/> feature-reboot	周昱宏 d5a7ae0 增加对容错的支持 5天前	常规分支 ▾	
<input type="checkbox"/> test/replicaset	王浩天 7a22f72 replicaset test 5天前	常规分支 ▾	
<input type="checkbox"/> test/gpu	王浩天 1ada727 modified job controller 5天前	常规分支 ▾	
<input type="checkbox"/> feature-DNS	周昱宏 31e2f9b mark 6天前	常规分支 ▾	
<input type="checkbox"/> test/autoscaler	王浩天 c0488c2 finish autoscaler test 6天前	常规分支 ▾	
<input type="checkbox"/> patch/kubectrl	王浩天 9f27862 kubectrl 6天前	常规分支 ▾	
<input type="checkbox"/> feature/service-controller	王浩天 10a852e merge service manag... 6天前	常规分支 ▾	
<input type="checkbox"/> patch/control-plane	王浩天 e84ccbf update control plane 7天前	常规分支 ▾	
<input type="checkbox"/> feature/autoscaler	王浩天 33626a7 finish autoscaler test ... 8天前	常规分支 ▾	
<input type="checkbox"/> feature/kubectrl	王浩天 ac84b81 resolve conflicts 8天前	常规分支 ▾	
<input type="checkbox"/> feature/deployment	王浩天 77ed688 [fix] deploymet 9天前	常规分支 ▾	
<input type="checkbox"/> zyh	周昱宏 ccb3507 finish service 12天前	常规分支 ▾	
<input type="checkbox"/> feature/gpu	王浩天 74ee191 tested gpu 15天前	常规分支 ▾	
<input type="checkbox"/> feature/replicaTest	钱博闻 54d39b9 fix: add rs config prefix 17天前	常规分支 ▾	
<input type="checkbox"/> feature/configSplit	钱博闻 9948754 fix: make multi pods 18天前	常规分支 ▾	
<input type="checkbox"/> feature-pod-unity	周昱宏 3e0e908 打通用户到调度器到k... 20天前	常规分支 ▾	

## 6.2 分支介绍

以下为部分分支，主要有feature、patch和test三类。

- feature：功能分支，每一个功能点对应一个feature分支
- patch：补丁分支，当遇到bug需要紧急提交补丁进行修复，会通过patch分支创建PR并进行合并
- test：测试分支，测试功能点（集成测试）时使用到的分支。

关于测试情况：现在，主分支test的目录下有很多测试文件，这些是我们进行答辩验收以及进行单元测试、集成测试的测试项。

王浩天 add recover arguments 530f93f 5天前	227 次提交	
...		
autoscaler	!56 完成autoscaler的测试	6天前
deployment	!47 解决了deployment控制rs异常的bug	9天前
fault-tolerance	add recover arguments	5天前
gpu	!58 更新了remote runner版本	5天前
iptable	rebase conflict	10天前
pod	small fix	17天前
replicaset	!59 测试replicaset并且录制视频	5天前
service	mark	6天前
DnsTransConfig.yaml	mark	9天前
dockerCpu.json	!28 完成对pod的cpu和内存的监控	23天前
myPodConfig.yaml	更改pod和podmanager的架构	10天前
pod.yaml	fix: message send to kubelet	1个月前
pomdModel.yaml	feat: add file marshal	1个月前




## 6.3 Pull Request情况

Pull Request情况：每次合并我们都采用Pull Request扁平化分支的合并方式，将feature、patch或test分支合并进develop分支，这样做便于进行责任定位，同时也便于在出现异常情况时进行回滚。

测试replicaset并且录制视频	K8sGroup:test/replicaset	K8sGroup:develop	审查：钱 王	测试：钱 王	5天前
更新了remote runner版本	K8sGroup:test/gpu	K8sGroup:develop	审查：钱 王	测试：钱 王	5天前
完成autoscaler的测试	K8sGroup:test/autoscaler	K8sGroup:develop	审查：钱 王	测试：钱 王	6天前
更新了control-plane的配置信息	K8sGroup:patch/control-plane	K8sGroup:develop	审查：钱 王	测试：钱	7天前
修改了bug，并且经过了测试。现在auto scaler可以正确处理普罗米修斯抖动导致的异常扩缩容	K8sGroup:feature/autoscaler	K8sGroup:develop	审查：钱 王	测试：钱	8天前
新增了deployment和replicaset的get命令	K8sGroup:feature/kubectl	K8sGroup:develop	审查：钱 王	测试：钱	8天前

## 6.4 敏捷开发

我们采取敏捷开发，定期进行小组例会，并且编写会议纪要，方便进行进度同步和任务分配，这种工作模式使得我们的开发效率大大提升。

- 5.28 sync  
5月28日 10:14
- 5.25 sync  
5月25日 21:14
- 5.22 sync  
5月22日 10:55
- 5.20 sync  
5月20日 15:08
- 周昱宏的视频会议  
5月17日 22:13
- 5.16 sync  
5月16日 21:19
- 5.15 sync  
5月15日 22:54
- 5.11 例会  
5月11日 12:56
- 5.8 例会  
5月8日 21:52 | 
- 5.5 sync  
5月5日 15:42 | 
- K8sGroup的视频会议  
5月2日 17:51 | 

使用飞书进行团队协作和文档共享，并且进行通过细粒度的任务分配。

K8sGroup-Wiki 公开

🔗 <≡

🔍 搜索

空间目录 +

📄 首页

▼ 📄 报告

📄 Minik8s开题报告

📄 Minik8s结题报告

▶ 📄 开发流程

▶ 📄 技术预研

▼ 📄 外部文档

▶ 📄 电子书

📄 云操作系统：Minik8s Lab.pdf

▶ 📄 资源

📄 TODO List

▼ 📄 会议记录及工作计划

📄 0404工作计划

📄 0405 项目架构与开题报告

📄 0408 会议记录

📄 0417 会议记录

📄 0424 会议记录

📄 0429 会议记录

📄 0506 中期检查

会议记录及工作计划 > 0408 会议记录 ☆

已经保存到云端

🔗 分享

0408 会议记录

• 目标：能跑就行，功能模块分开一点

TODO

☑️ ReplicaSet @钱博闻

☑️ Kubelet @周昱宏

☑️ Controller-Manager @王浩天

0408 会议记录

TODO

