最终汇报

董云鹏（组长）　张子谦　冯逸飞

2023.06.02

CONTENTS

1 技术栈与软件开发

1.1 选用技术栈

开发语言：Go

github.com/fatih/color minik8s的分级日志系统
github.com/klauspost/pgzip 用户文件的zip压缩
github.com/gin-gonic/gin APIServer框架

容器运行时：docker

github.com/mholt/archiver Docker镜像打包时用到的tar压缩

分布式存储：ETCD

go.etcd.io/etcd/client/v3 和Etcd存储交互操作的客户端

持续性集成：CI/CD

测试：单元测试：go-test

docker/login-action CICD自动推送镜像到dockerHub
docker/setup-qemu-action CICD交叉编译平台
gotest.tools/v3 项目测试框架

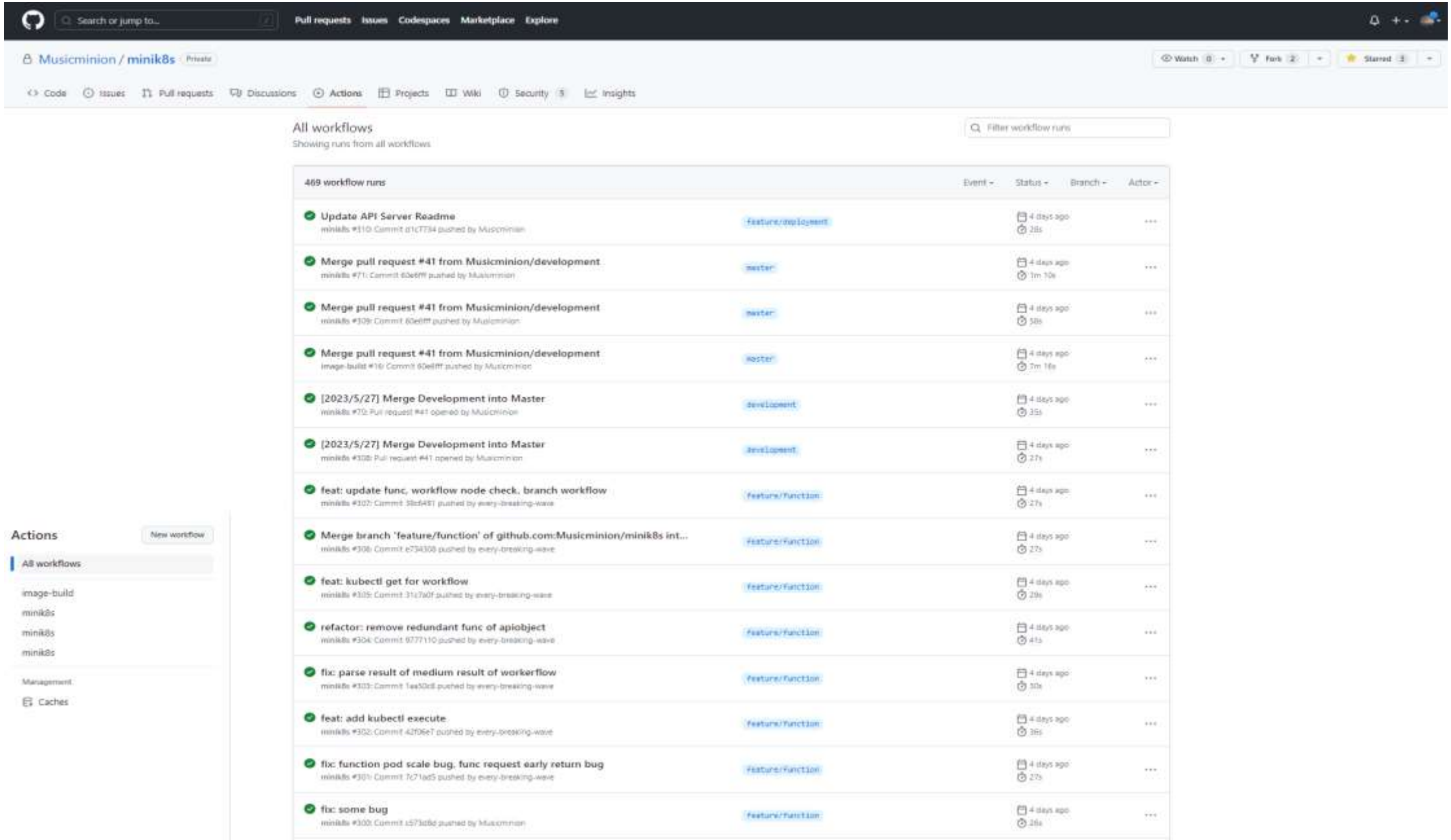
支持应用

github.com/melbahja/goph GPU Job的SSH的客户端
github.com/melbahja/goph GPU Job的SSH的客户端
github.com/pallets/flask Serveless容器内的运行的程序

命令行

gopkg.in/yaml.v3 go的yaml文件解析
spf13/cobra命令行解析
github.com/spf13/cobra Kubectl的命令行工具
github.com/jedib0t/go-pretty/table Kubectl美化

1.2CI/CD git actions进行持续性集成



1.3 版本控制

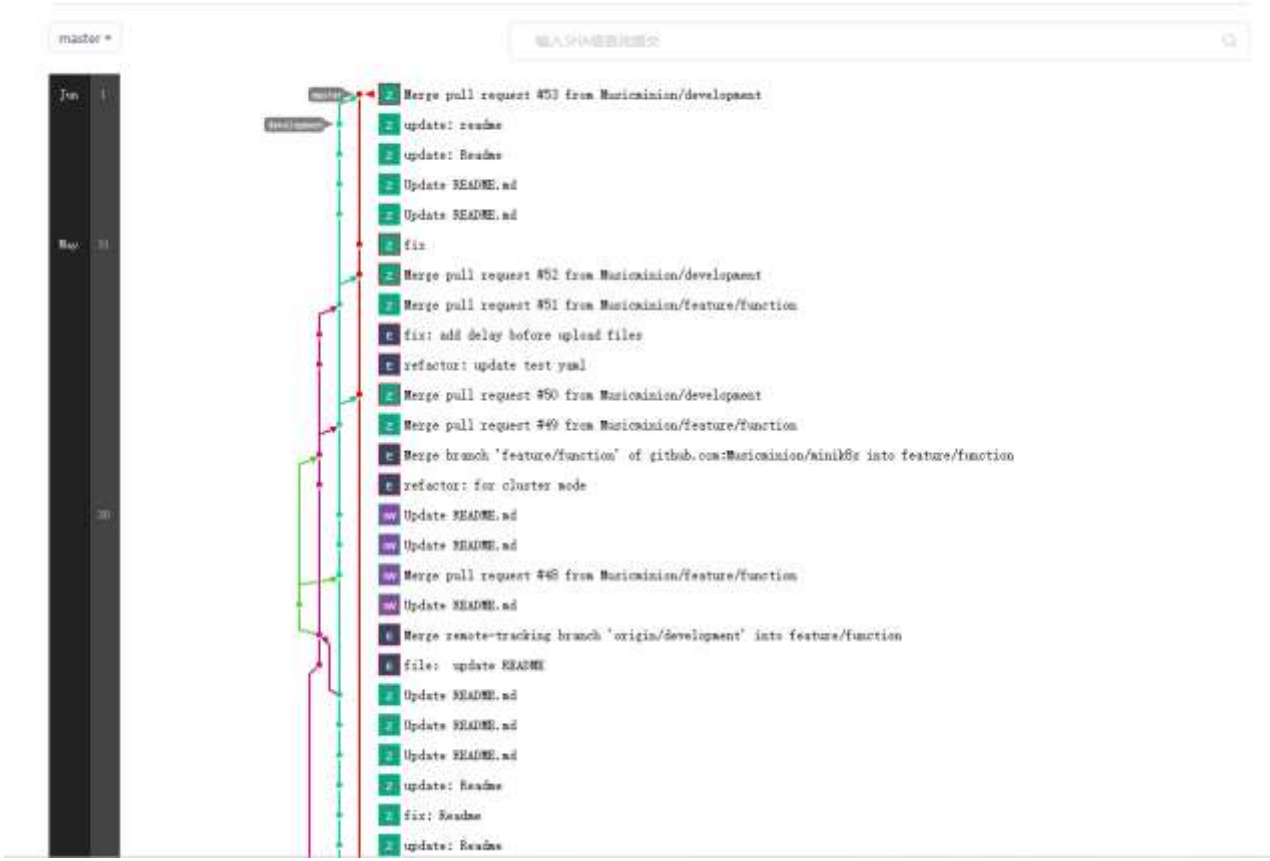
1 代码量较大

Languages

language	files	code	comment	blank	total
Go	172	19,287	0	4,063	23,350
Go Checksum File	1	426	0	1	427
Go Module File	1	102	0	4	106

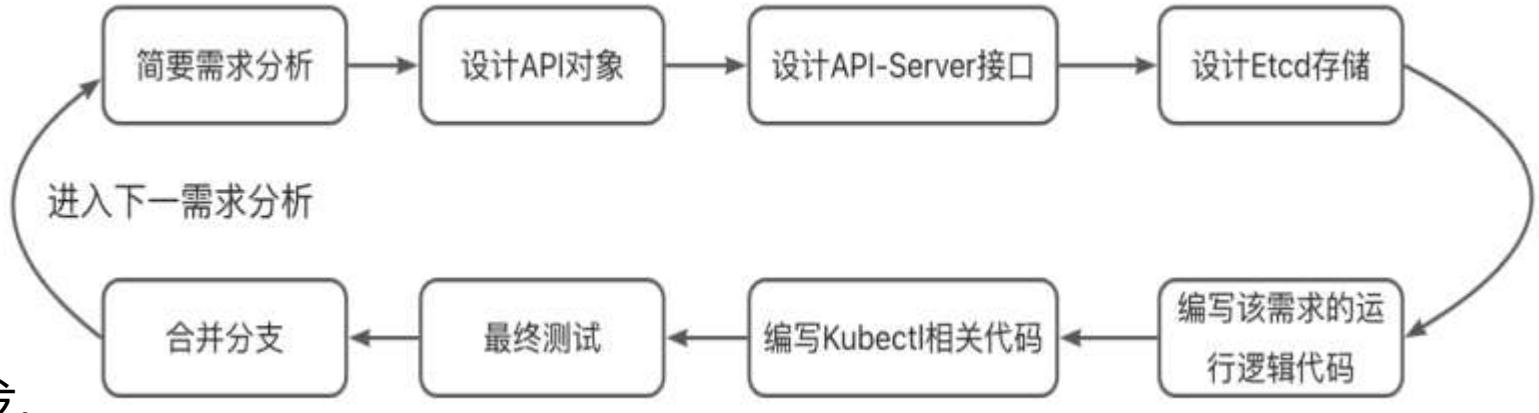
2 每日审查与合并

3 分支管理



1.4 Other Attempts

1 **规范项目开发流程**



•我们的软件开发基于迭代开发、敏捷开发。

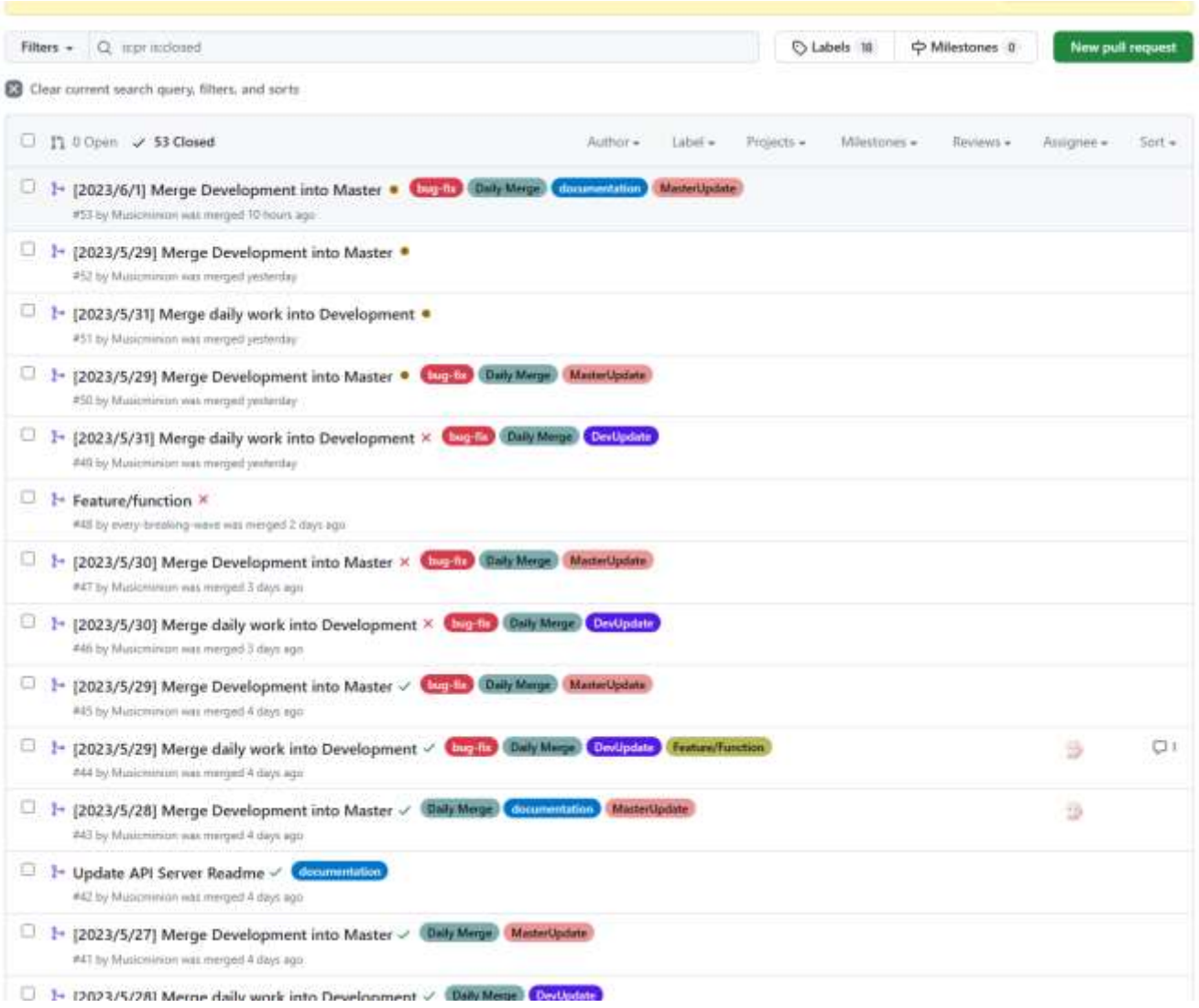
•小组成员每天晚上在软件学院大楼实验室集中进行开发新功能，减少沟通障碍，做到有问题及时解决、沟通，有困难相互请教，这也大大的提高了我们小组的效率。截止15周周末，我们已经完成了所有的功能的开发。基本符合预期进度。

•对于新功能开发，我们采用"动态分配"方法，根据进度灵活分配成员的任务。项目框架搭建好之后，基本上在任何时间点小组同时在开发两个或者两个以上的需求。一人开发完成之后，交给另外一个组员完成代码的审查和测试，测试通过之后合并到Master

•功能开发的过程主要是：简要的需求分析->设计API对象->设计API-Server的接口->设计Etcd存储情况->编写该需求的运行逻辑代码->编写Kubectl相关代码->最终测试

1.4 Other Attempts

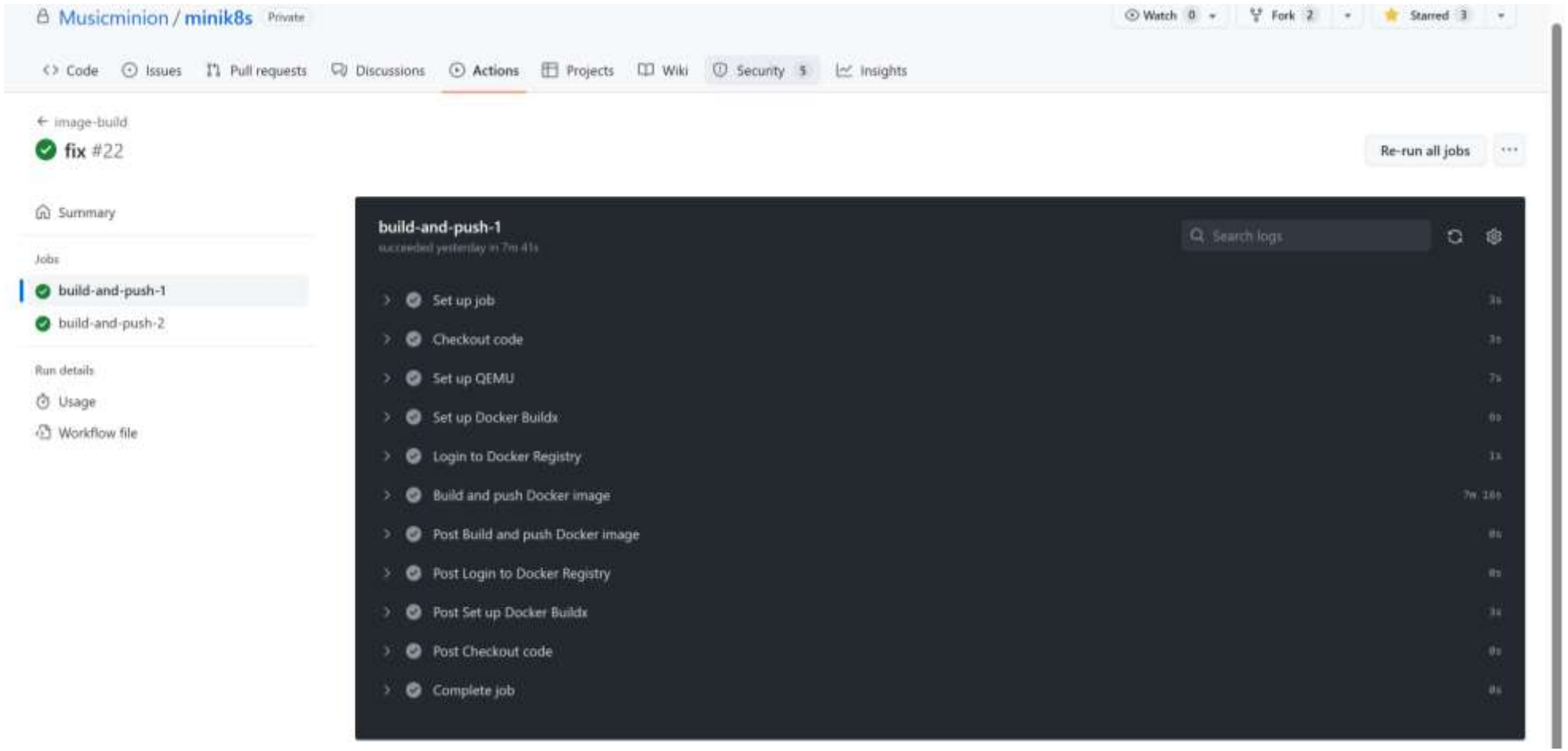
2 **规范PR提交**



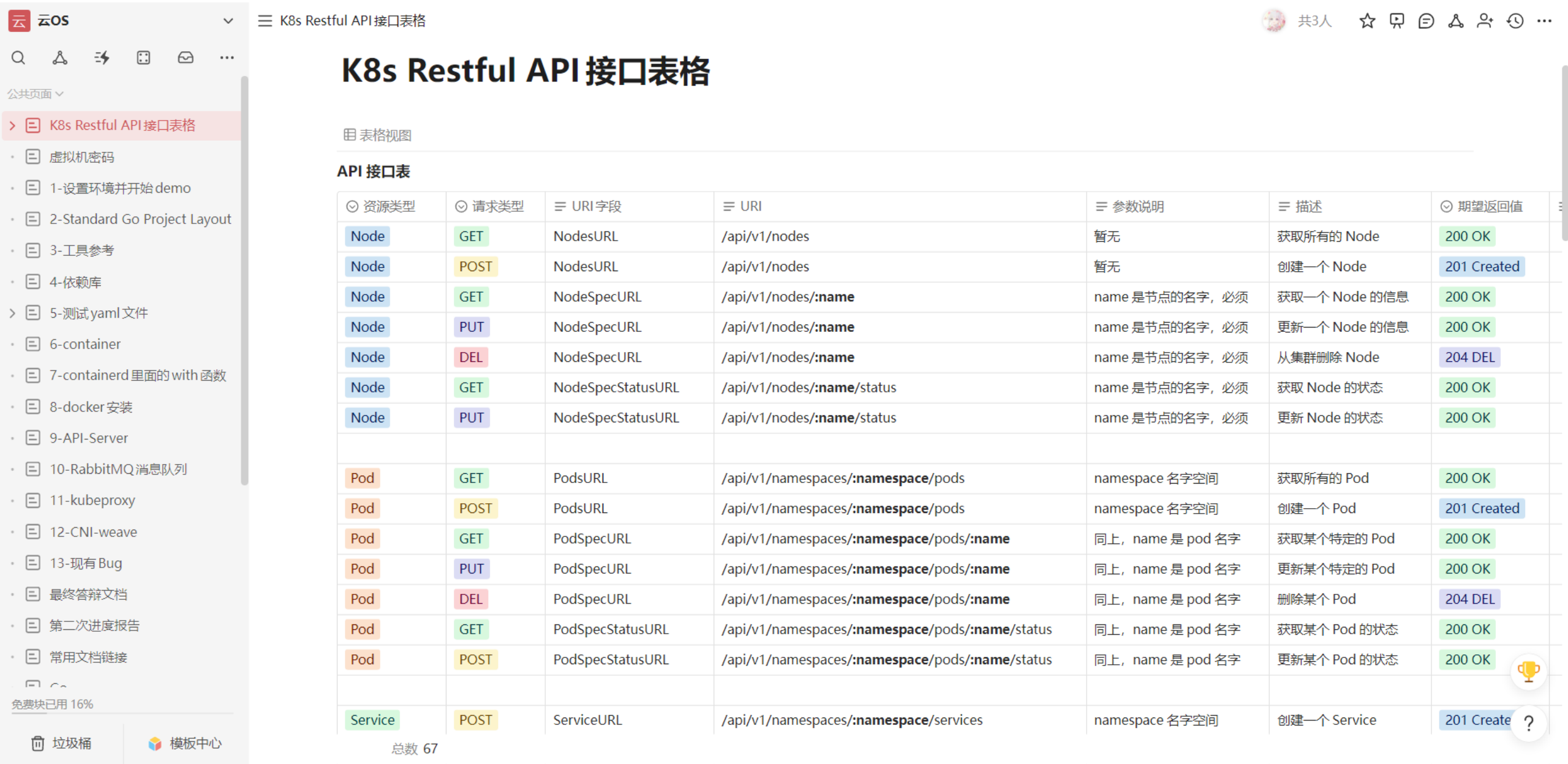
1.4 Other Attempts

master分支收到推送之后,测试后会构建跟项目有关的image，比如func的server，并推送到dockerhub

3 使用Git Actions



1.5 接口文档



K8s Restful API 接口表格

表格视图

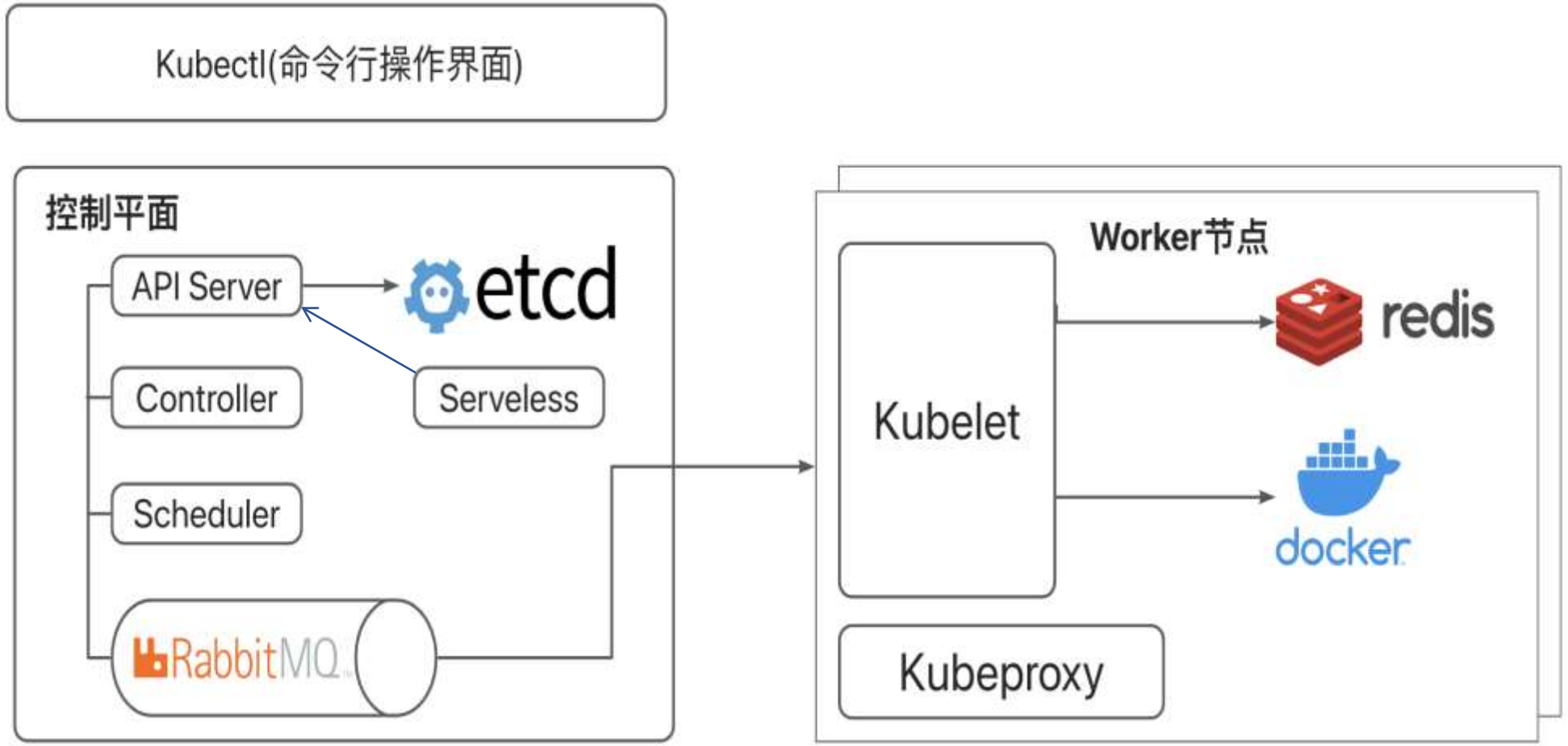
API 接口表

资源类型	请求类型	URI 字段	URI	参数说明	描述	期望返回值
Node	GET	NodesURL	/api/v1/nodes	暂无	获取所有的 Node	200 OK
Node	POST	NodesURL	/api/v1/nodes	暂无	创建一个 Node	201 Created
Node	GET	NodeSpecURL	/api/v1/nodes/**:name**	name 是节点的名字，必须	获取一个 Node 的信息	200 OK
Node	PUT	NodeSpecURL	/api/v1/nodes/**:name**	name 是节点的名字，必须	更新一个 Node 的信息	200 OK
Node	DEL	NodeSpecURL	/api/v1/nodes/**:name**	name 是节点的名字，必须	从集群删除 Node	204 DEL
Node	GET	NodeSpecStatusURL	/api/v1/nodes/**:name**/status	name 是节点的名字，必须	获取 Node 的状态	200 OK
Node	PUT	NodeSpecStatusURL	/api/v1/nodes/**:name**/status	name 是节点的名字，必须	更新 Node 的状态	200 OK
Pod	GET	PodsURL	/api/v1/namespaces/**:namespace**/pods	namespace 名字空间	获取所有的 Pod	200 OK
Pod	POST	PodsURL	/api/v1/namespaces/**:namespace**/pods	namespace 名字空间	创建一个 Pod	201 Created
Pod	GET	PodSpecURL	/api/v1/namespaces/**:namespace**/pods/**:name**	同上，name 是 pod 名字	获取某个特定的 Pod	200 OK
Pod	PUT	PodSpecURL	/api/v1/namespaces/**:namespace**/pods/**:name**	同上，name 是 pod 名字	更新某个特定的 Pod	200 OK
Pod	DEL	PodSpecURL	/api/v1/namespaces/**:namespace**/pods/**:name**	同上，name 是 pod 名字	删除某个 Pod	204 DEL
Pod	GET	PodSpecStatusURL	/api/v1/namespaces/**:namespace**/pods/**:name**/status	同上，name 是 pod 名字	获取某个 Pod 的状态	200 OK
Pod	POST	PodSpecStatusURL	/api/v1/namespaces/**:namespace**/pods/**:name**/status	同上，name 是 pod 名字	更新某个 Pod 的状态	200 OK
Service	POST	ServiceURL	/api/v1/namespaces/**:namespace**/services	namespace 名字空间	创建一个 Service	201 Create

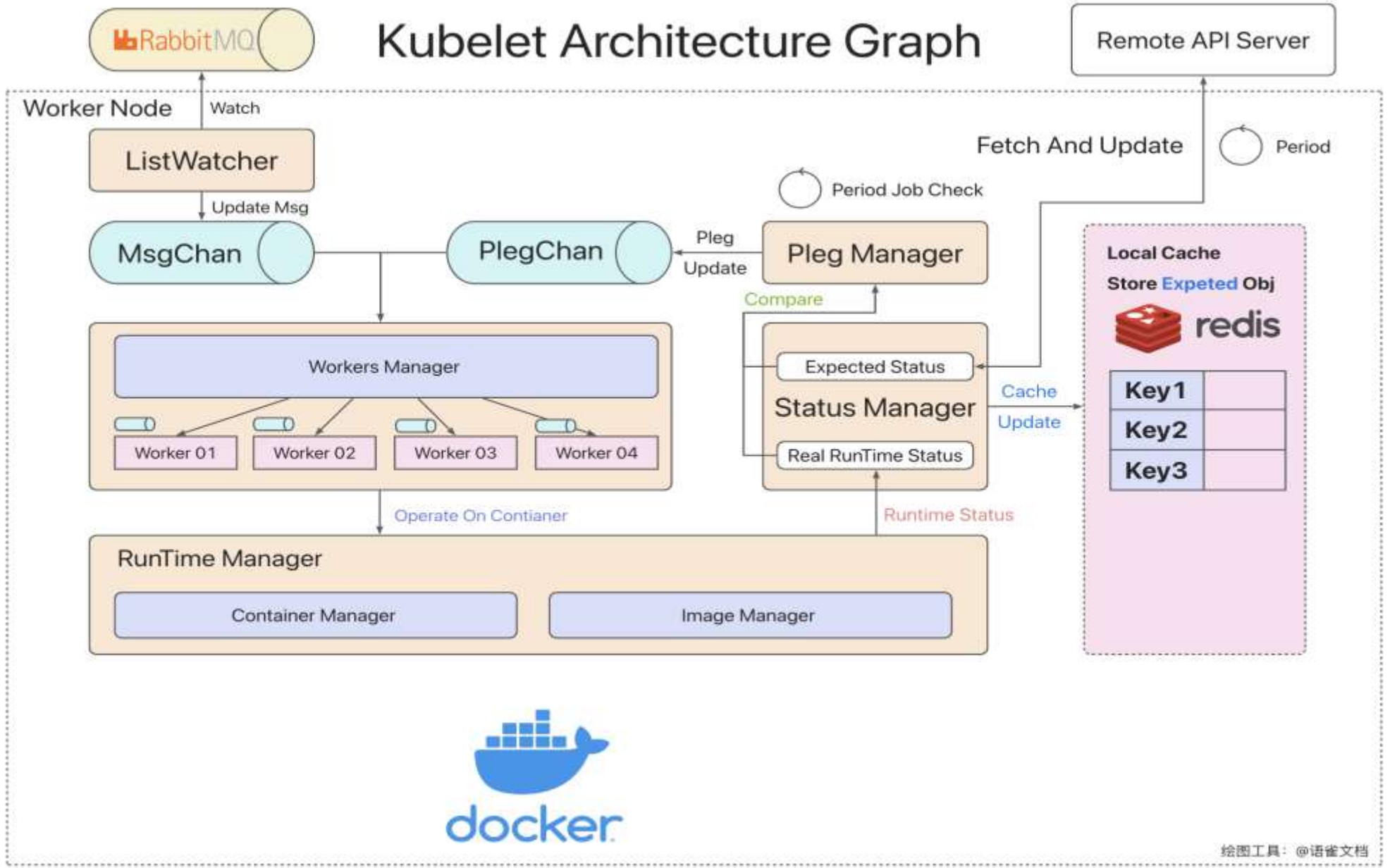
总数 67

2 系统结构及组件功能

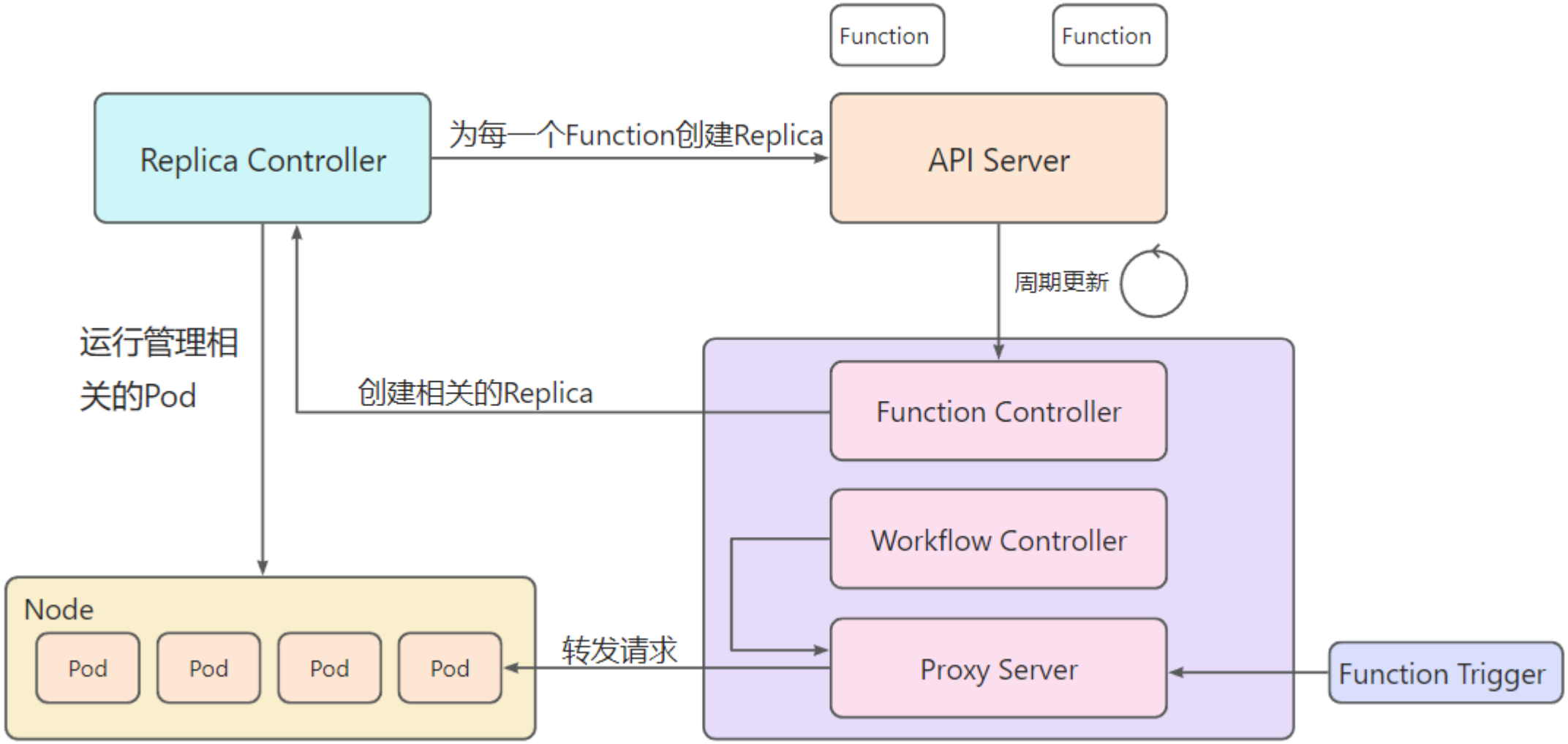
2.1 系统结构


Kubectl(命令行操作界面)
控制平面
API Server
etcd
Controller
Serveless
Scheduler
RabbitMQ
Worker节点
Kubelet
redis
docker
Kubeproxy

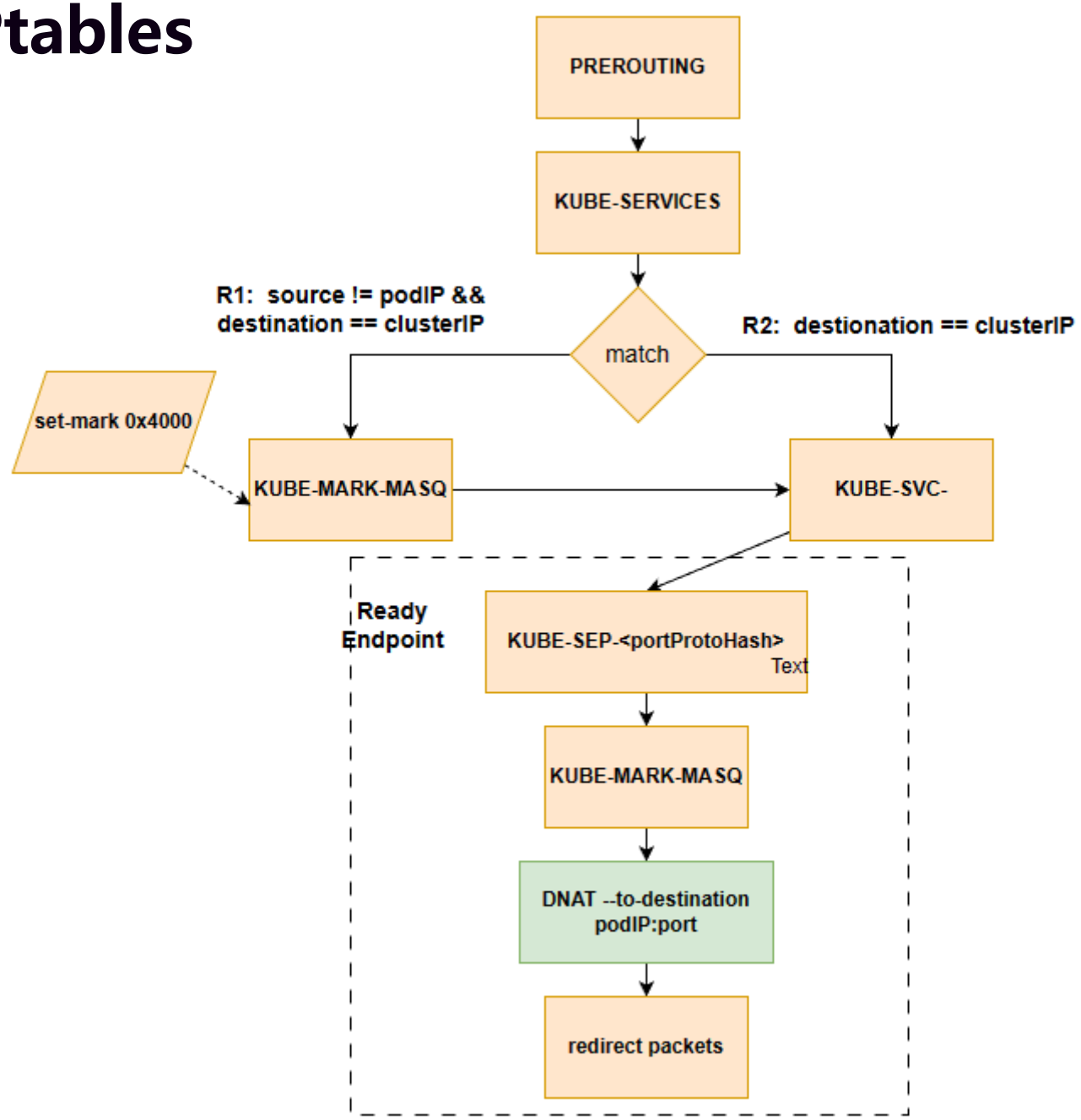
2.2 kubelet


Kubelet Architecture Graph
RabbitMQ
Remote API Server
Worker Node
Watch
ListWatcher
Update Msg
MsgChan
PlegChan
Pleg
Update
Pleg Manager
Period Job Check
Fetch And Update
Period
Compare
Local Cache
Store Expeted Obj
redis
Key1
Key2
Key3
Workers Manager
Worker 01
Worker 02
Worker 03
Worker 04
Expected Status
Status Manager
Real RunTime Status
Cache
Update
Operate On Contianer
Runtime Status
RunTime Manager
Container Manager
Image Manager
docker
绘图工具：@语雀文档

2.2 Serverless 架构

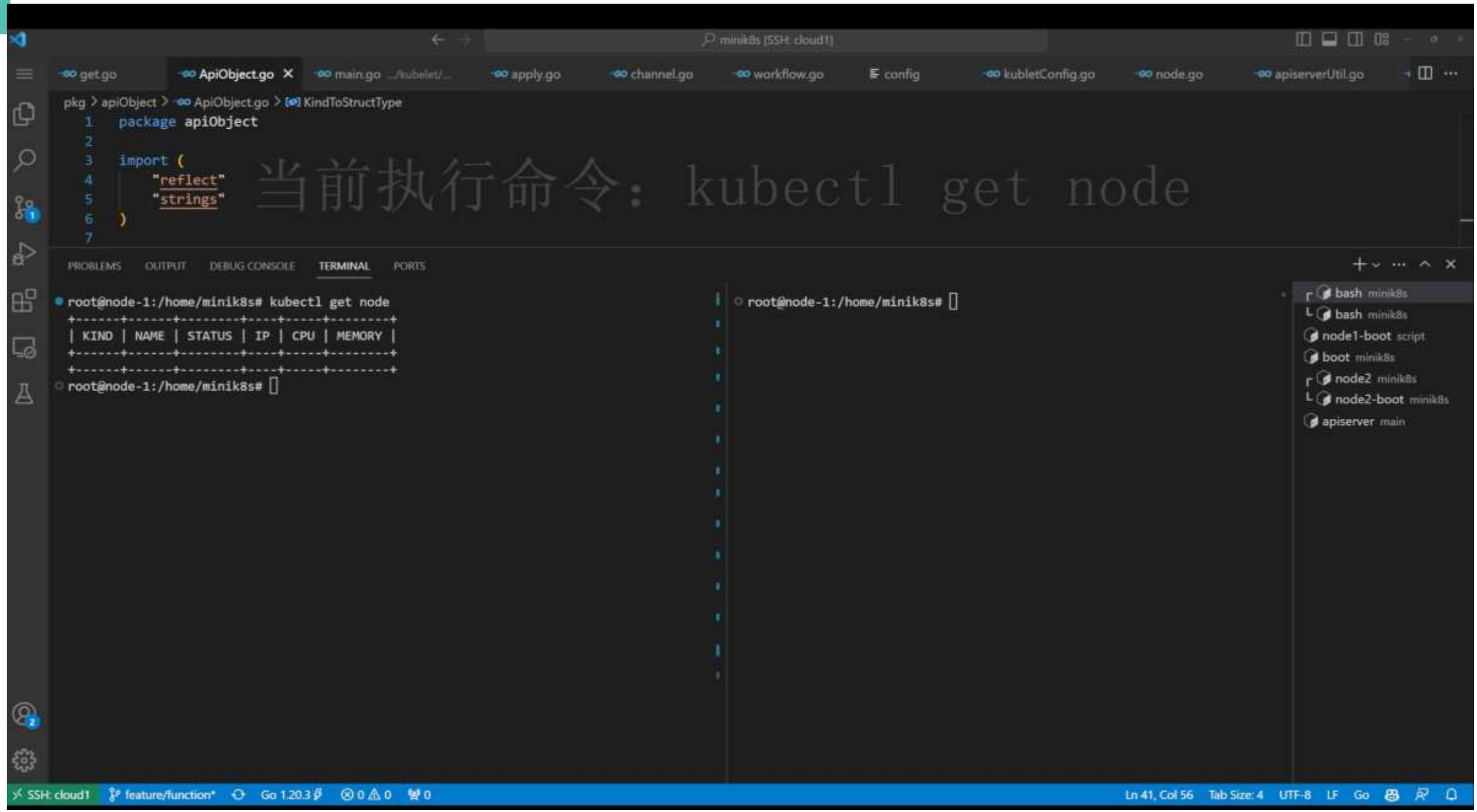


2.3 KubeProxy IPtables

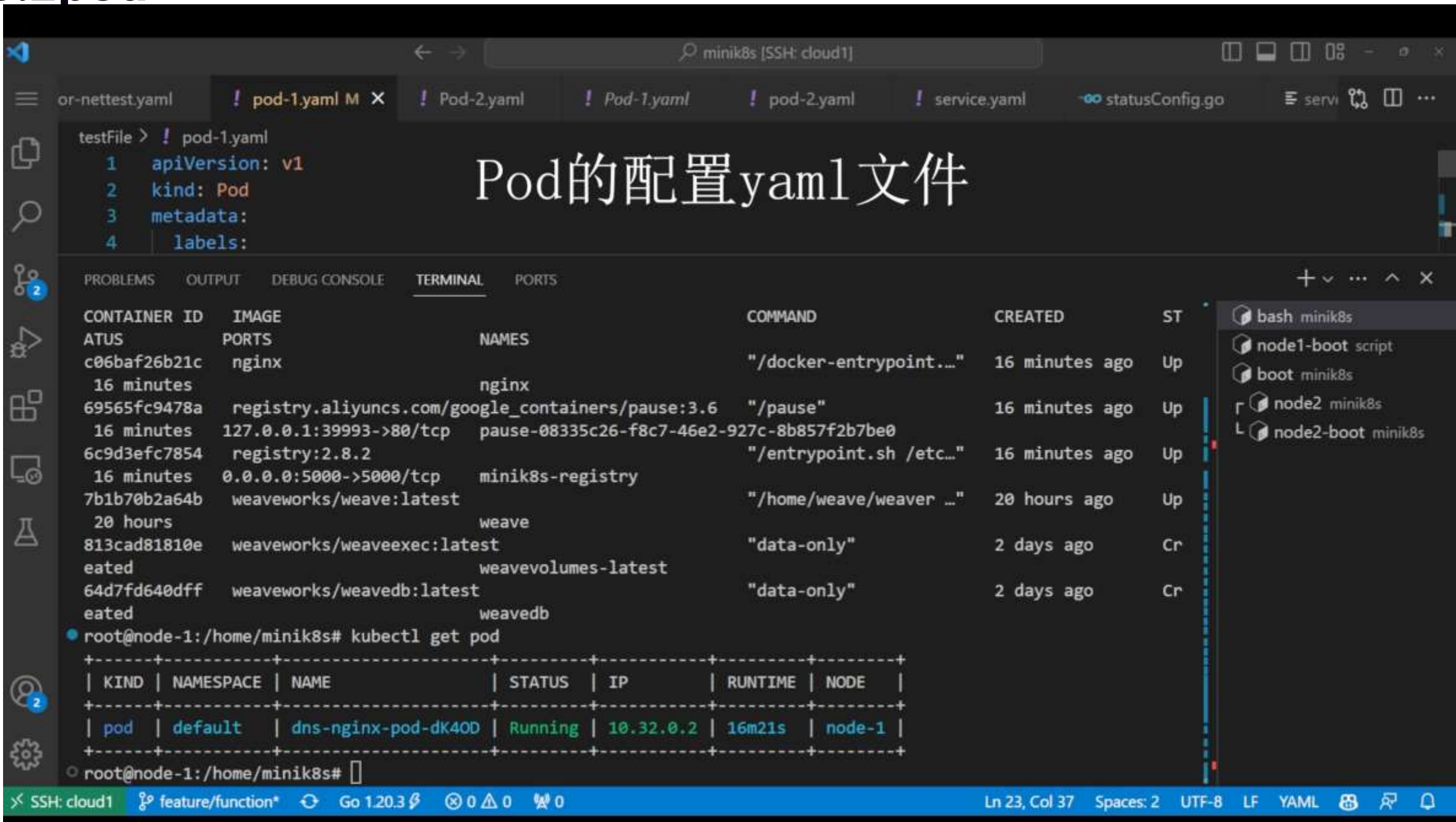


3 演示视频

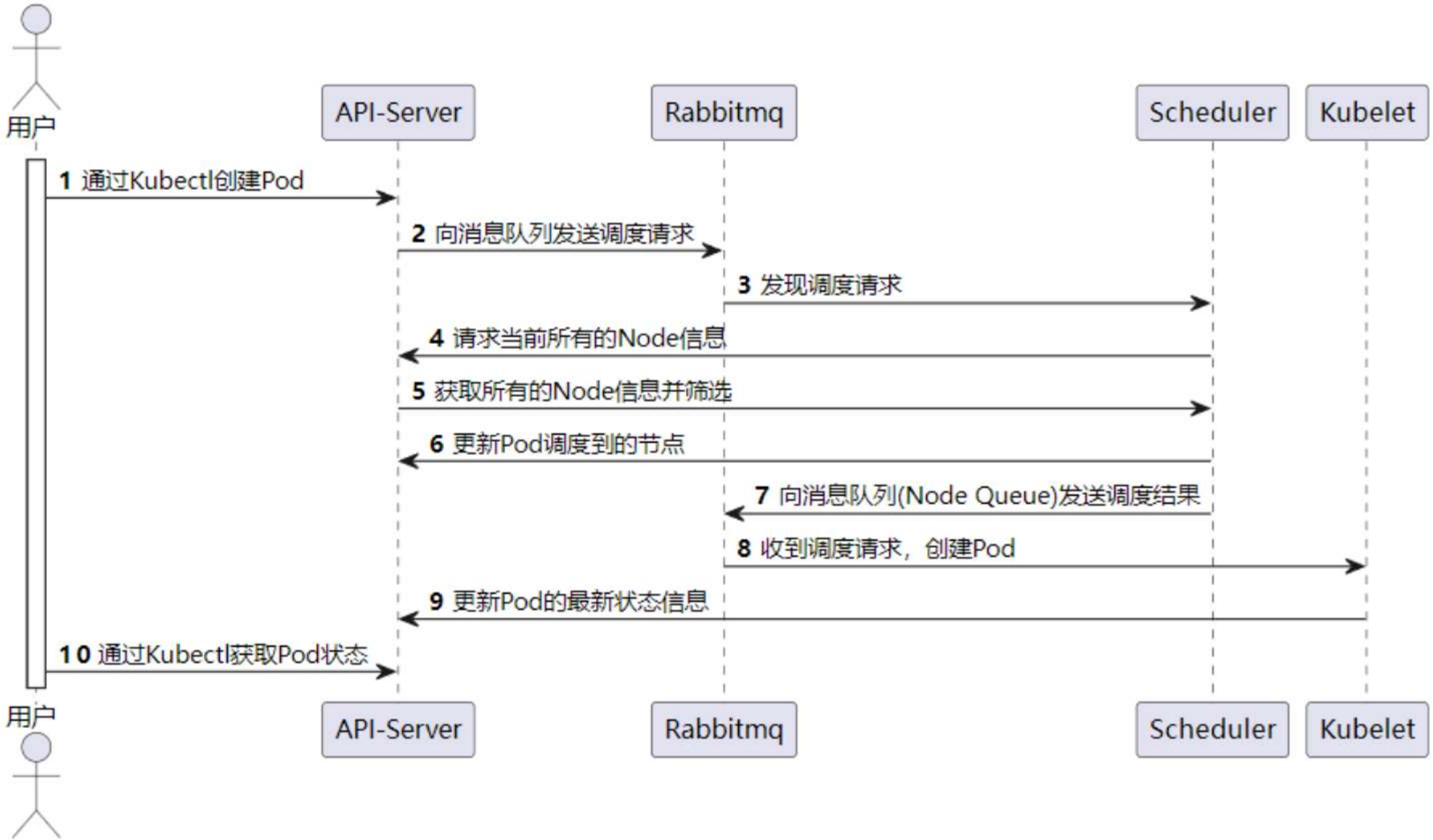
3.1node


minik8s [SSH: cloud1]
get.go
ApiObject.go
main.go .../kubelet/...
apply.go
channel.go
workflow.go
config
kubletConfig.go
node.go
apiserverUtil.go
pkg > apiObject > ApiObject.go > KindToStructType
1 package apiObject
2
3 import (
4 "reflect"
5 "strings"
6)
7
当前执行命令：kubectl get node
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
root@node-1:/home/minik8s# kubectl get node
+------+------+--------+----+-----+--------+
| KIND | NAME | STATUS | IP | CPU | MEMORY |
+------+------+--------+----+-----+--------+
+------+------+--------+----+-----+--------+
root@node-1:/home/minik8s#
root@node-1:/home/minik8s#
bash minik8s
bash minik8s
node1-boot script
boot minik8s
node2 minik8s
node2-boot minik8s
apiserver main
SSH: cloud1
feature/function*
Go 1.20.3
Ln 41, Col 56
Tab Size: 4
UTF-8
LF
Go

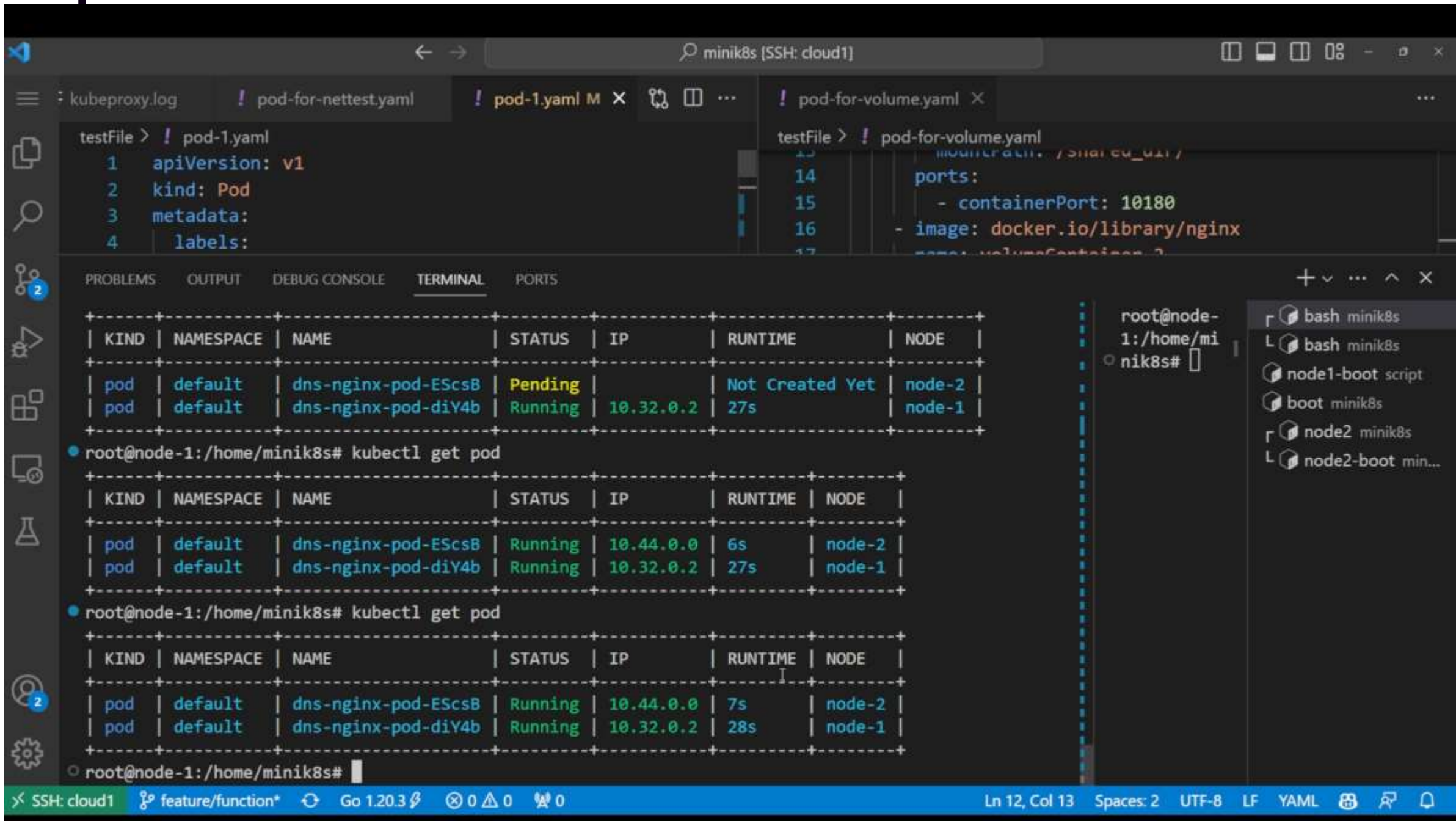
3.2pod


minik8s [SSH: cloud1]
or-nettest.yaml
pod-1.yaml M
Pod-2.yaml
Pod-1.yaml
pod-2.yaml
service.yaml
statusConfig.go
testFile > pod-1.yaml
1 apiVersion: v1
2 kind: Pod
3 metadata:
4 labels:
Pod的配置yaml文件
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
CONTAINER ID IMAGE COMMAND CREATED ST
ATUS PORTS NAMES
c06baf26b21c nginx "/docker-entrypoint.…" 16 minutes ago Up
16 minutes nginx
69565fc9478a registry.aliyuncs.com/google_containers/pause:3.6 "/pause" 16 minutes ago Up
16 minutes 127.0.0.1:39993->80/tcp pause-08335c26-f8c7-46e2-927c-8b857f2b7be0
6c9d3efc7854 registry:2.8.2 "/entrypoint.sh /etc…" 16 minutes ago Up
16 minutes 0.0.0.0:5000->5000/tcp minik8s-registry
7b1b70b2a64b weaveworks/weave:latest "/home/weave/weaver …" 20 hours ago Up
20 hours weave
813cad81810e weaveworks/weaveexec:latest "data-only" 2 days ago Cr
eated weavevolumes-latest
64d7fd640dff weaveworks/weavedb:latest "data-only" 2 days ago Cr
eated weavedb
root@node-1:/home/minik8s# kubectl get pod
| KIND | NAMESPACE | NAME | STATUS | IP | RUNTIME | NODE |
| pod | default | dns-nginx-pod-dK4OD | Running | 10.32.0.2 | 16m21s | node-1 |
root@node-1:/home/minik8s#
bash minik8s
node1-boot script
boot minik8s
node2 minik8s
node2-boot minik8s
SSH: cloud1
feature/function*
Go 1.20.3
Ln 23, Col 37
Spaces: 2
UTF-8
LF
YAML

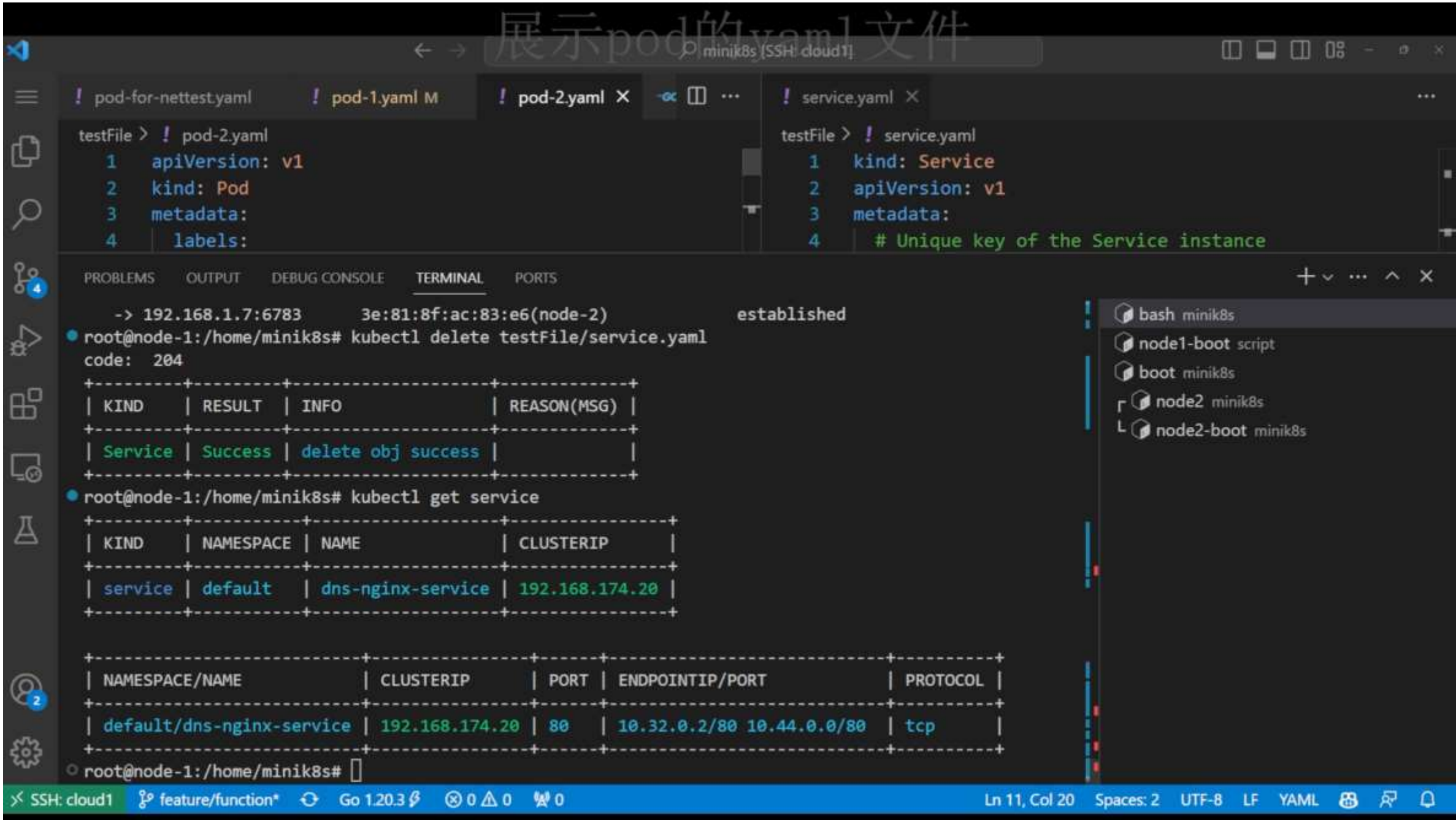
3.2pod


用户
API-Server
Rabbitmq
Scheduler
Kubelet
1 通过Kubectl创建Pod
2 向消息队列发送调度请求
3 发现调度请求
4 请求当前所有的Node信息
5 获取所有的Node信息并筛选
6 更新Pod调度到的节点
7 向消息队列(Node Queue)发送调度结果
8 收到调度请求，创建Pod
9 更新Pod的最新状态信息
10 通过Kubectl获取Pod状态
用户
API-Server
Rabbitmq
Scheduler
Kubelet

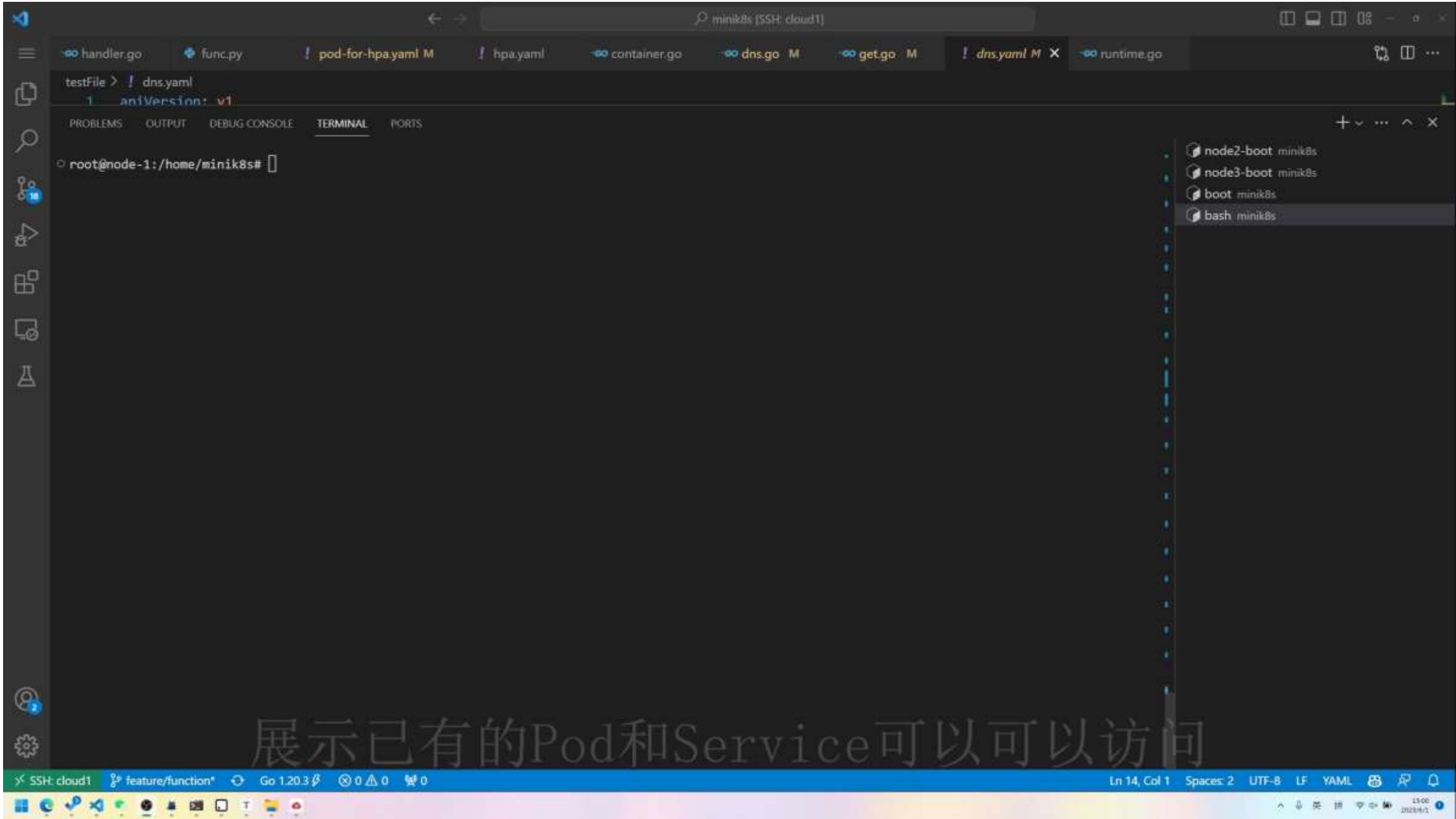
3.2pod


minik8s [SSH: cloud1]
kubeproxy.log
pod-for-nettest.yaml
pod-1.yaml M
pod-for-volume.yaml
testFile > pod-1.yaml
1 apiVersion: v1
2 kind: Pod
3 metadata:
4 labels:
testFile > pod-for-volume.yaml
14 ports:
15 - containerPort: 10180
16 - image: docker.io/library/nginx
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
KIND	NAMESPACE	NAME	STATUS	IP	RUNTIME	NODE
pod	default	dns-nginx-pod-EScsB	Pending		Not Created Yet	node-2
pod	default	dns-nginx-pod-diY4b	Running	10.32.0.2	27s	node-1
root@node-1:/home/minik8s# kubectl get pod
KIND	NAMESPACE	NAME	STATUS	IP	RUNTIME	NODE
pod	default	dns-nginx-pod-EScsB	Running	10.44.0.0	6s	node-2
pod	default	dns-nginx-pod-diY4b	Running	10.32.0.2	27s	node-1
root@node-1:/home/minik8s# kubectl get pod
KIND	NAMESPACE	NAME	STATUS	IP	RUNTIME	NODE
pod	default	dns-nginx-pod-EScsB	Running	10.44.0.0	7s	node-2
pod	default	dns-nginx-pod-diY4b	Running	10.32.0.2	28s	node-1
root@node-1:/home/minik8s#
root@node-1:/home/minik8s#
bash minik8s
bash minik8s
node1-boot script
boot minik8s
node2 minik8s
node2-boot min...
SSH: cloud1
feature/function*
Go 1.20.3
Ln 12, Col 13
Spaces: 2
UTF-8
LF
YAML

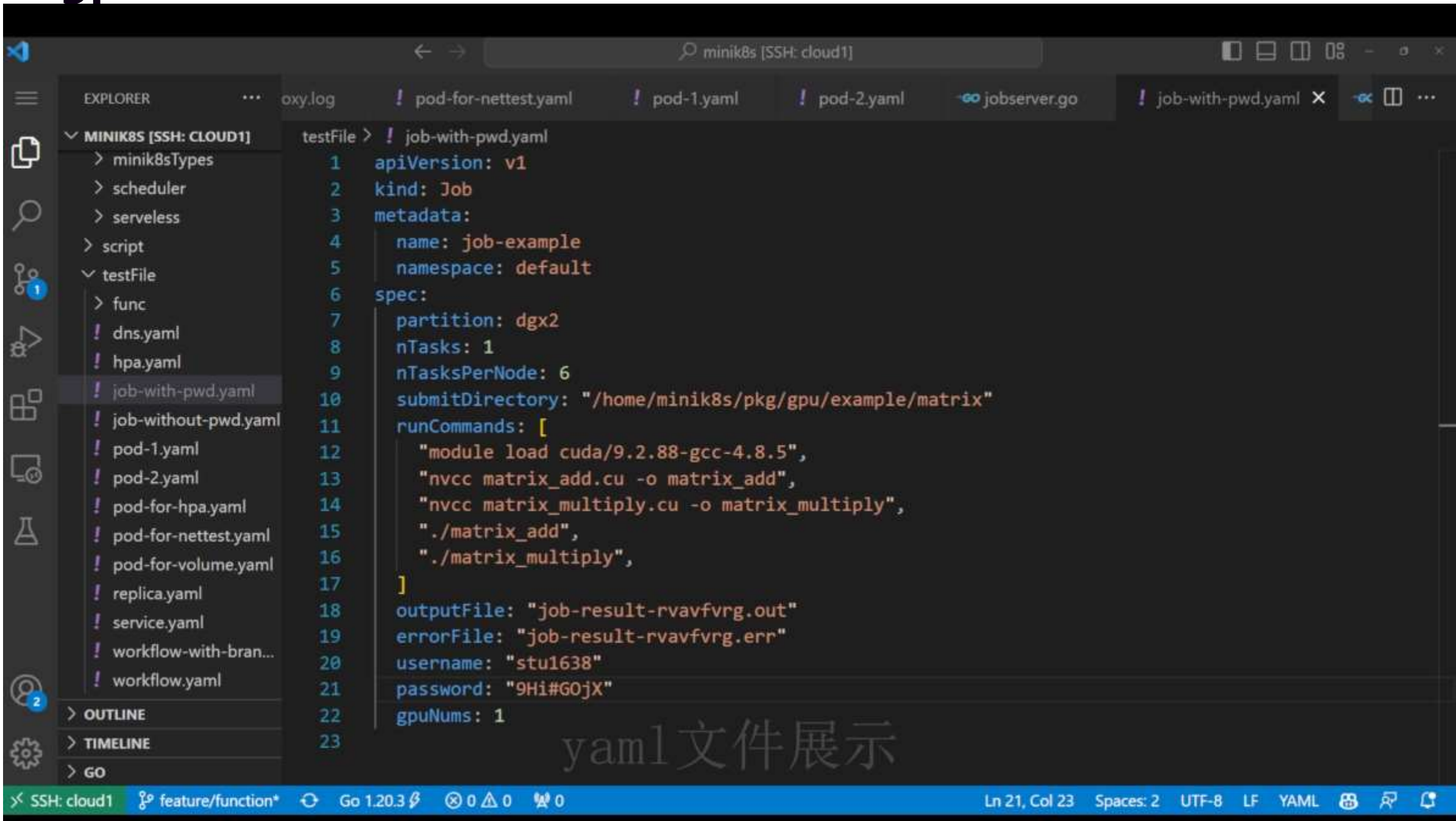
3.3service


展示pod的yaml文件
minik8s [SSH: cloud1]
pod-for-nettest.yaml
pod-1.yaml M
pod-2.yaml
service.yaml
testFile > pod-2.yaml
1 apiVersion: v1
2 kind: Pod
3 metadata:
4 labels:
testFile > service.yaml
1 kind: Service
2 apiVersion: v1
3 metadata:
4 # Unique key of the Service instance
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
-> 192.168.1.7:6783 3e:81:8f:ac:83:e6(node-2) established
root@node-1:/home/minik8s# kubectl delete testFile/service.yaml
code: 204
| KIND | RESULT | INFO | REASON(MSG) |
| Service | Success | delete obj success | |
root@node-1:/home/minik8s# kubectl get service
KIND	NAMESPACE	NAME	CLUSTERIP
service	default	dns-nginx-service	192.168.174.20
NAMESPACE/NAME	CLUSTERIP	PORT	ENDPOINTIP/PORT	PROTOCOL
default/dns-nginx-service	192.168.174.20	80	10.32.0.2/80 10.44.0.0/80	tcp
root@node-1:/home/minik8s#
bash minik8s
node1-boot script
boot minik8s
node2 minik8s
node2-boot minik8s
SSH: cloud1
feature/function*
Go 1.20.3
Ln 11, Col 20
Spaces: 2
UTF-8
LF
YAML

3.4DNS


handler.go
func.py
pod-for-hpa.yaml M
hpa.yaml
container.go
dns.go M
get.go M
dns.yaml M
runtime.go
testFile > dns.yaml
1 apiVersion: v1
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
root@node-1:/home/minik8s#
node2-boot minik8s
node3-boot minik8s
boot minik8s
bash minik8s
展示已有的Pod和Service可以可以访问
SSH: cloud1
feature/function*
Go 1.20.3
Ln 14, Col 1
Spaces: 2
UTF-8
LF
YAML

3.5gpu


minik8s [SSH: cloud1]
EXPLORER
MINIK8S [SSH: CLOUD1]
minik8sTypes
scheduler
serveless
script
testFile
func
dns.yaml
hpa.yaml
job-with-pwd.yaml
job-without-pwd.yaml
pod-1.yaml
pod-2.yaml
pod-for-hpa.yaml
pod-for-nettest.yaml
pod-for-volume.yaml
replica.yaml
service.yaml
workflow-with-bran...
workflow.yaml
OUTLINE
TIMELINE
GO
oxy.log
pod-for-nettest.yaml
pod-1.yaml
pod-2.yaml
jobserver.go
job-with-pwd.yaml
testFile > job-with-pwd.yaml
1 apiVersion: v1
2 kind: Job
3 metadata:
4 name: job-example
5 namespace: default
6 spec:
7 partition: dgx2
8 nTasks: 1
9 nTasksPerNode: 6
10 submitDirectory: "/home/minik8s/pkg/gpu/example/matrix"
11 runCommands: [
12 "module load cuda/9.2.88-gcc-4.8.5",
13 "nvcc matrix_add.cu -o matrix_add",
14 "nvcc matrix_multiply.cu -o matrix_multiply",
15 "./matrix_add",
16 "./matrix_multiply",
17]
18 outputFile: "job-result-rvavfvrg.out"
19 errorFile: "job-result-rvavfvrg.err"
20 username: "stu1638"
21 password: "9Hi#GOjX"
22 gpuNums: 1
23
yaml文件展示
SSH: cloud1
feature/function*
Go 1.20.3
0 0 0
Ln 21, Col 23
Spaces: 2
UTF-8
LF
YAML

3.5gpu

cudaError_t cudaMalloc (void devPtr, size_t size);**

CUDA 中的一个内存分配函数，用于分配 device 内存。

cudaMalloc 函数可以分配指定大小的连续内存块，并返回指向此内存块的指针给 devPtr。注意这里的 devPtr 应该是存放在 host 中的。

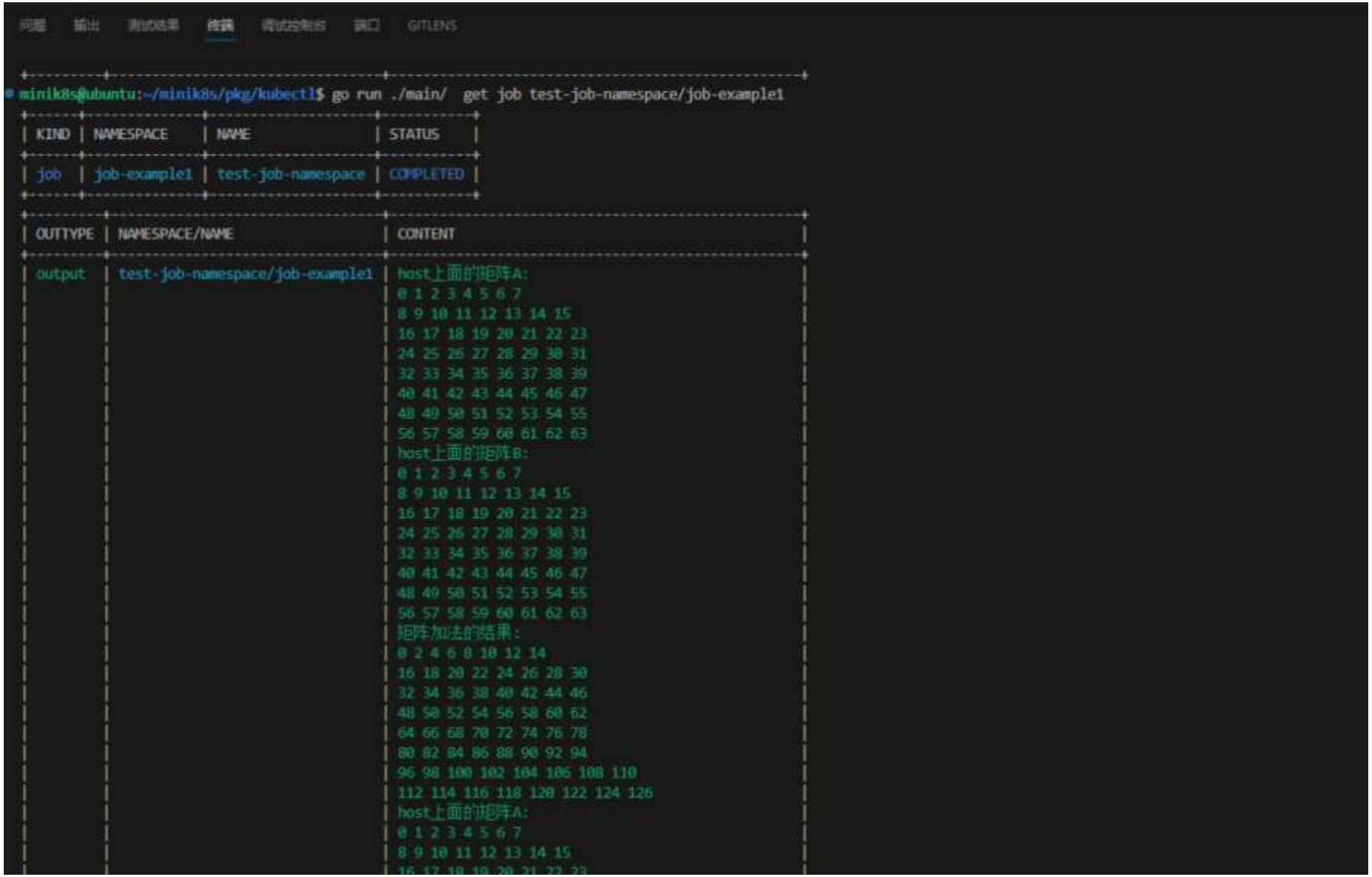
C++

```
//allocate GPU mem
  cudaMalloc((void **)&dev_A, sizeof(int *) * M);
  cudaMalloc((void **)&dev_B, sizeof(int *) * M);
  cudaMalloc((void **)&dev_C, sizeof(int *) * M);
```

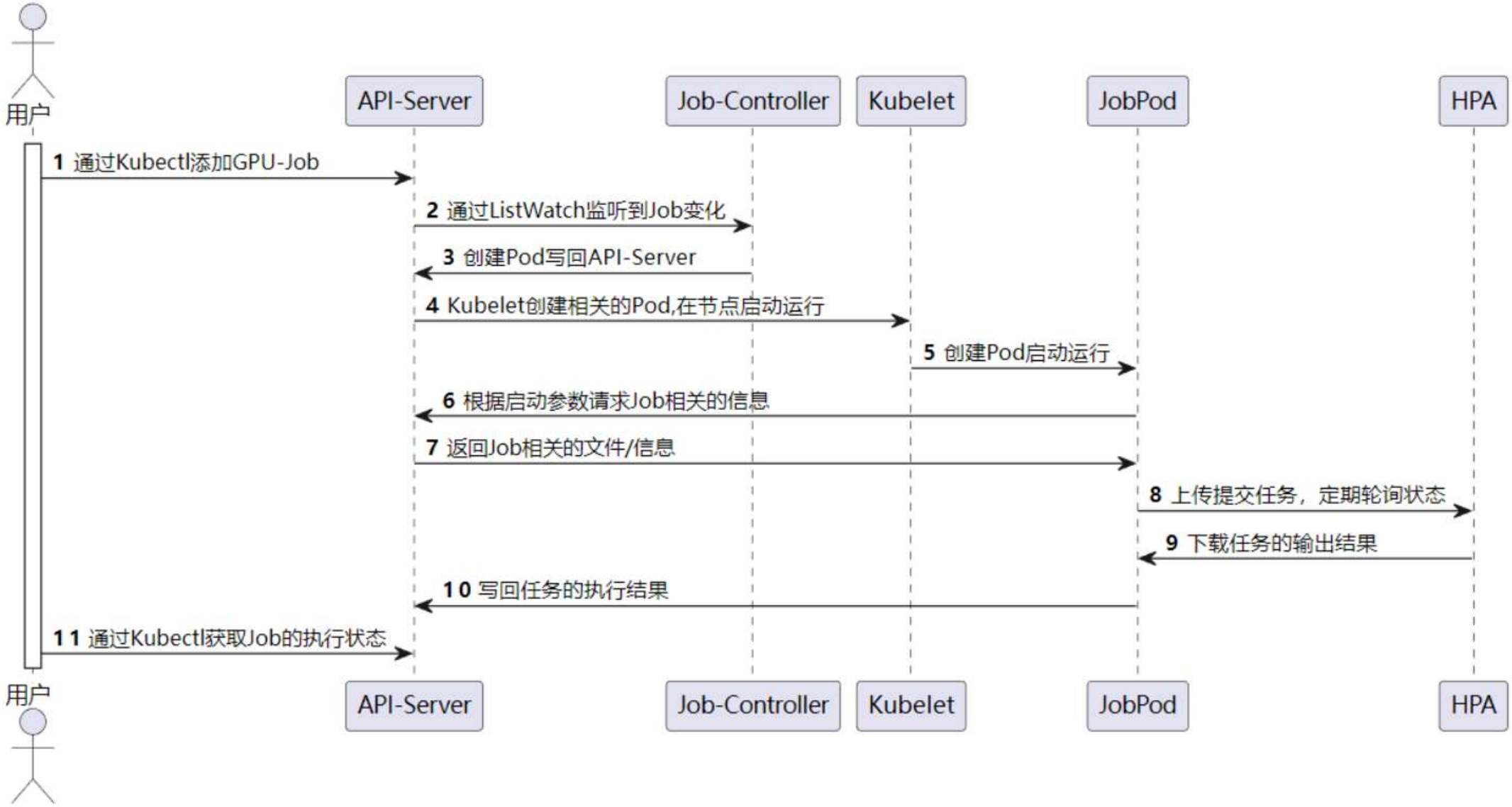
我们编写了简易的并行矩阵加法和乘法函数。

使用CUDA编程的情况下，我们首先定义了matrix_add和matrix_multiply两个使用__global__标记的核函数。
CUDA的风格类C，所以在处理矩阵这样的二维数组时需要两重指针，在进行host内存和device内存之间的数据转移，这里的cudaMemcpy函数作用是拷贝一段连续的内存，所以无法处理二重指针，需要辅助指针dev_A和dev_B，也就是说我们共需要四类指针，host上的二重和一重指针，device上的二重和一重指针。

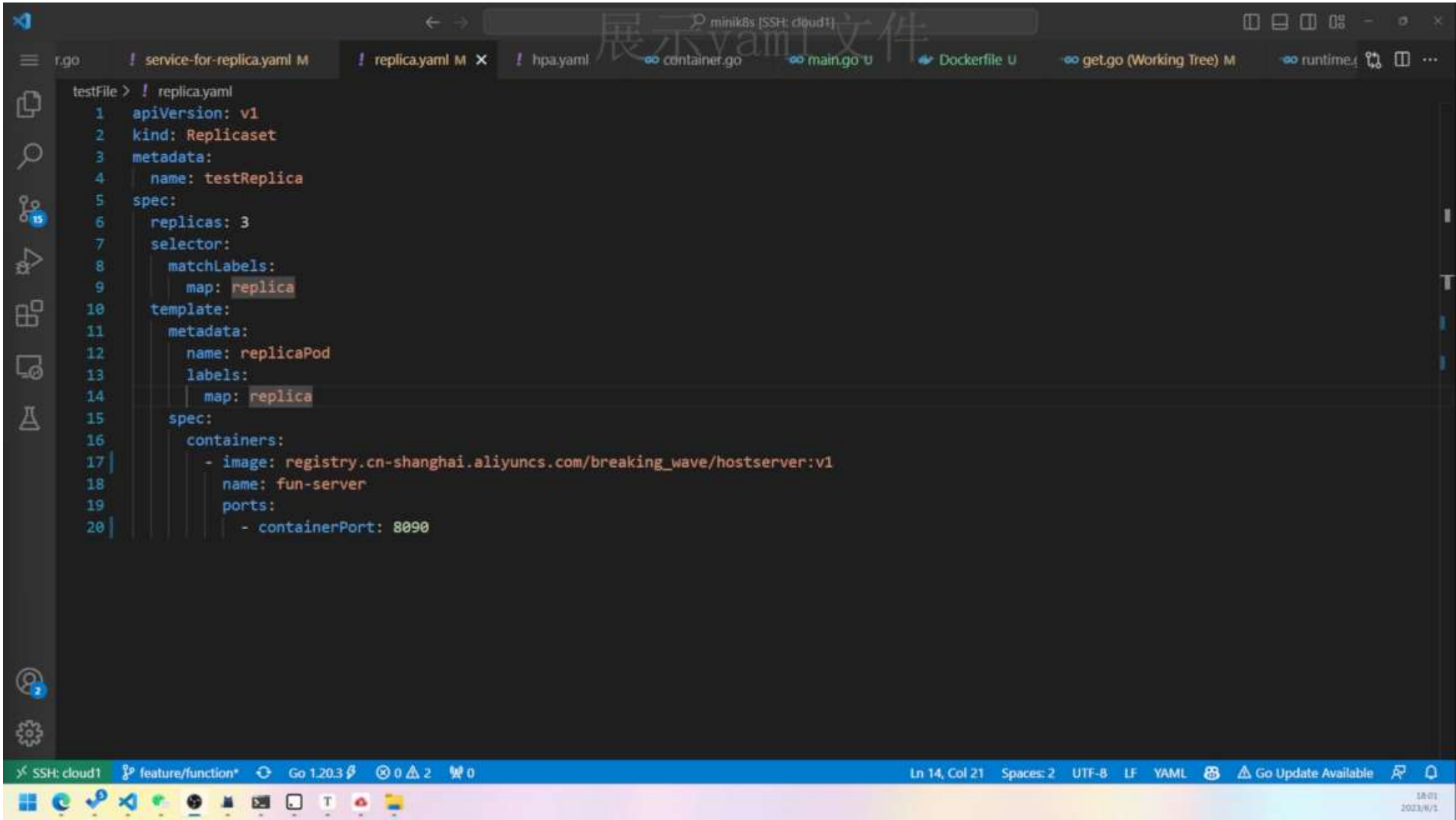
3.5gpu


问题 输出 测试结果 终端 调试控制台 端口 GITLENS
minik8s@ubuntu:~/minik8s/pkg/kubectl$ go run ./main/ get job test-job-namespace/job-example1
KIND | NAMESPACE | NAME | STATUS
job | job-example1 | test-job-namespace | COMPLETED
OUTTYPE | NAMESPACE/NAME | CONTENT
output | test-job-namespace/job-example1
host上面的矩阵A:
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
host上面的矩阵B:
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
矩阵加法的结果:
0 2 4 6 8 10 12 14
16 18 20 22 24 26 28 30
32 34 36 38 40 42 44 46
48 50 52 54 56 58 60 62
64 66 68 70 72 74 76 78
80 82 84 86 88 90 92 94
96 98 100 102 104 106 108 110
112 114 116 118 120 122 124 126
host上面的矩阵A:
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15

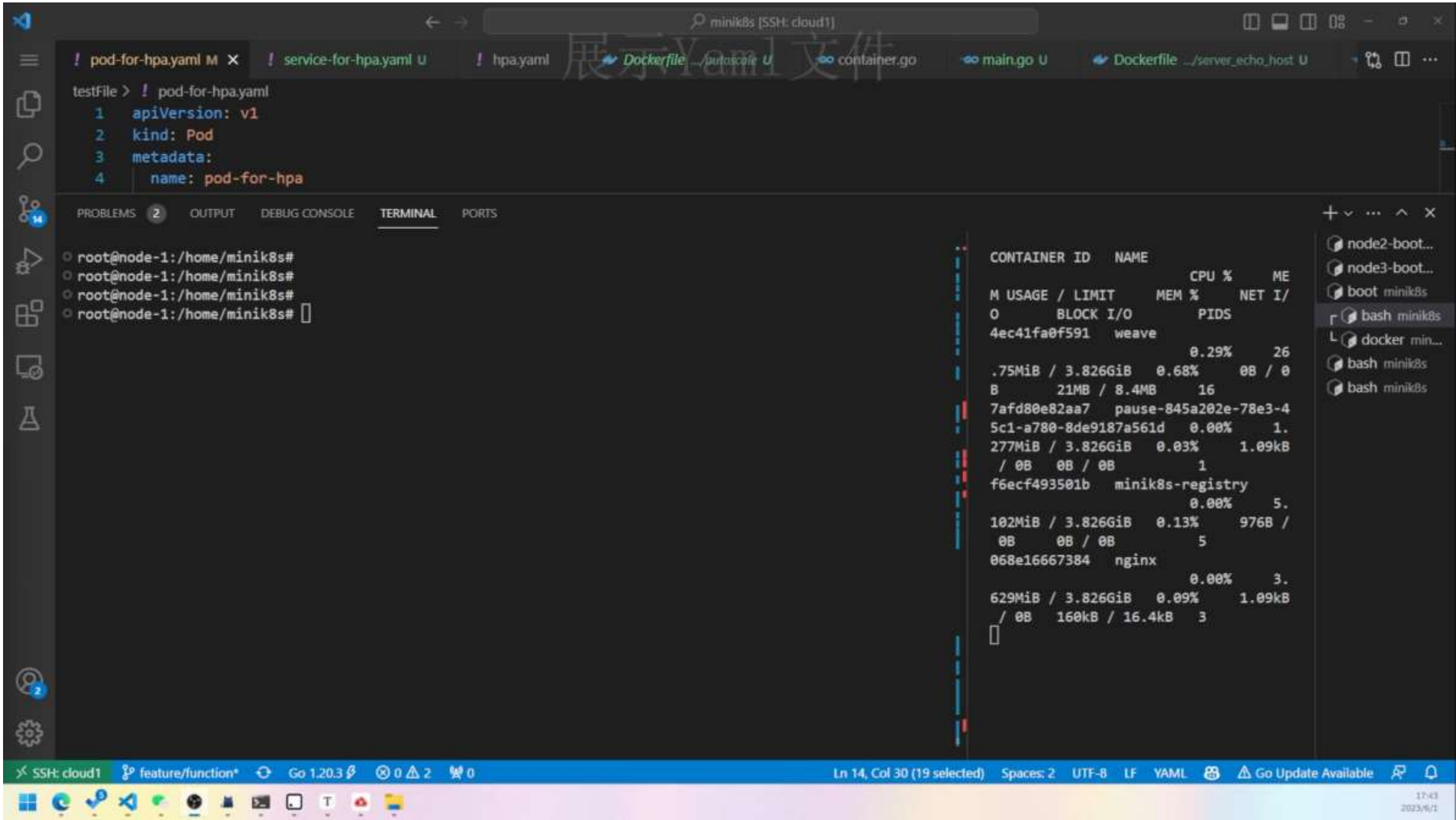
3.5gpu


用户
API-Server
Job-Controller
Kubelet
JobPod
HPA
1 通过Kubectl添加GPU-Job
2 通过ListWatch监听到Job变化
3 创建Pod写回API-Server
4 Kubelet创建相关的Pod,在节点启动运行
5 创建Pod启动运行
6 根据启动参数请求Job相关的信息
7 返回Job相关的文件/信息
8 上传提交任务，定期轮询状态
9 下载任务的输出结果
10 写回任务的执行结果
11 通过Kubectl获取Job的执行状态
用户
API-Server
Job-Controller
Kubelet
JobPod
HPA

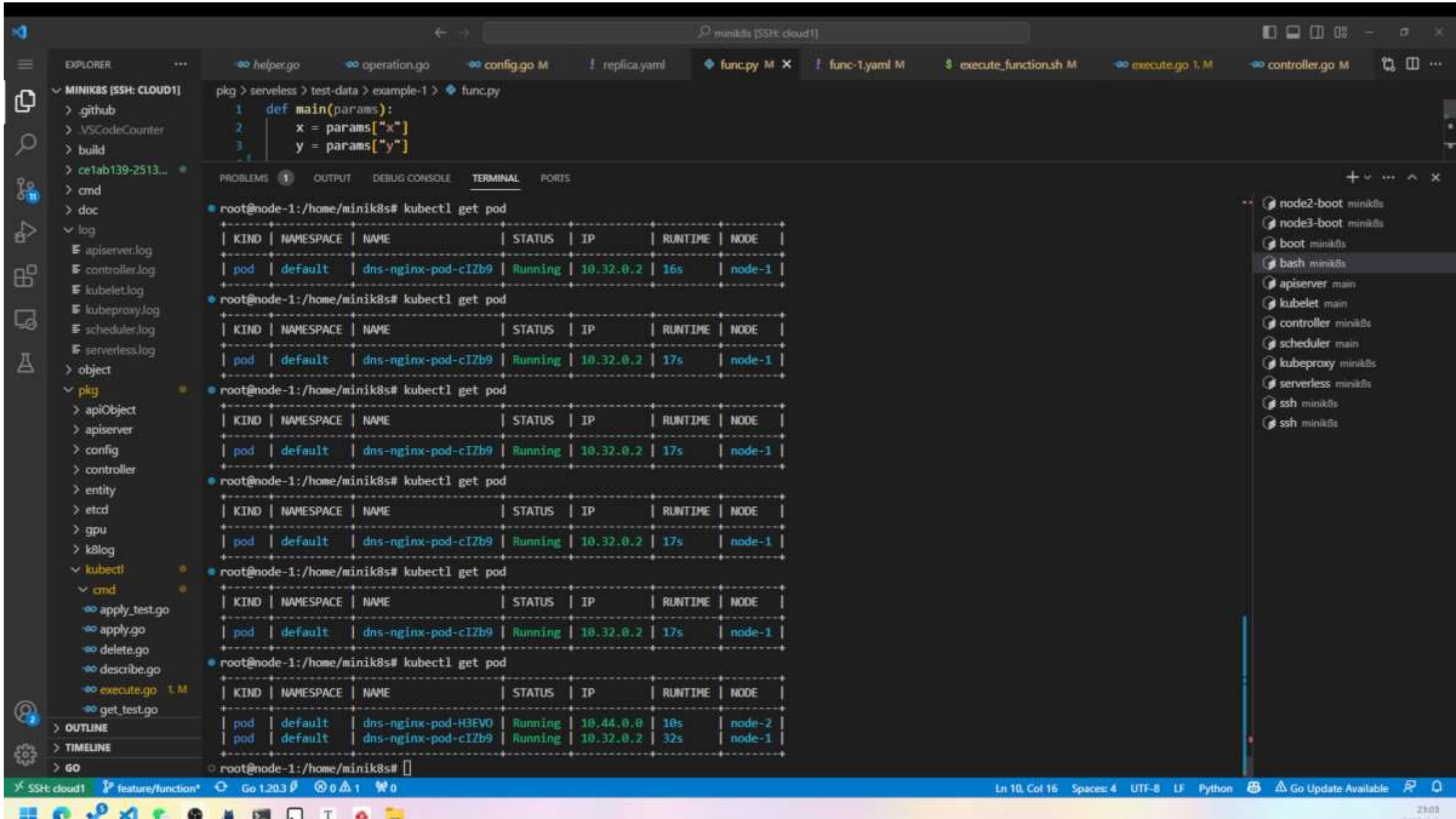
3.6replicaset



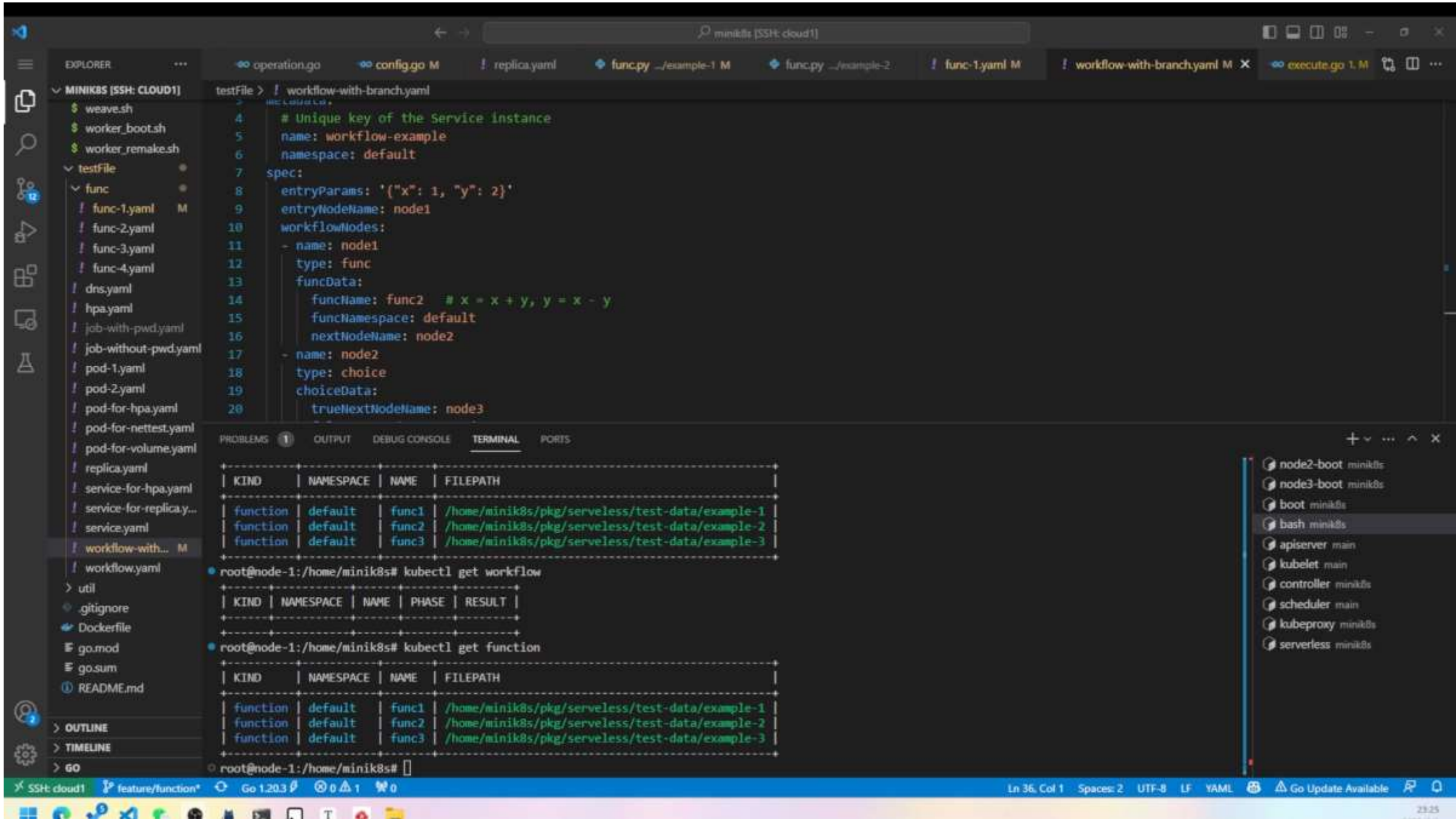
3.7auto scale



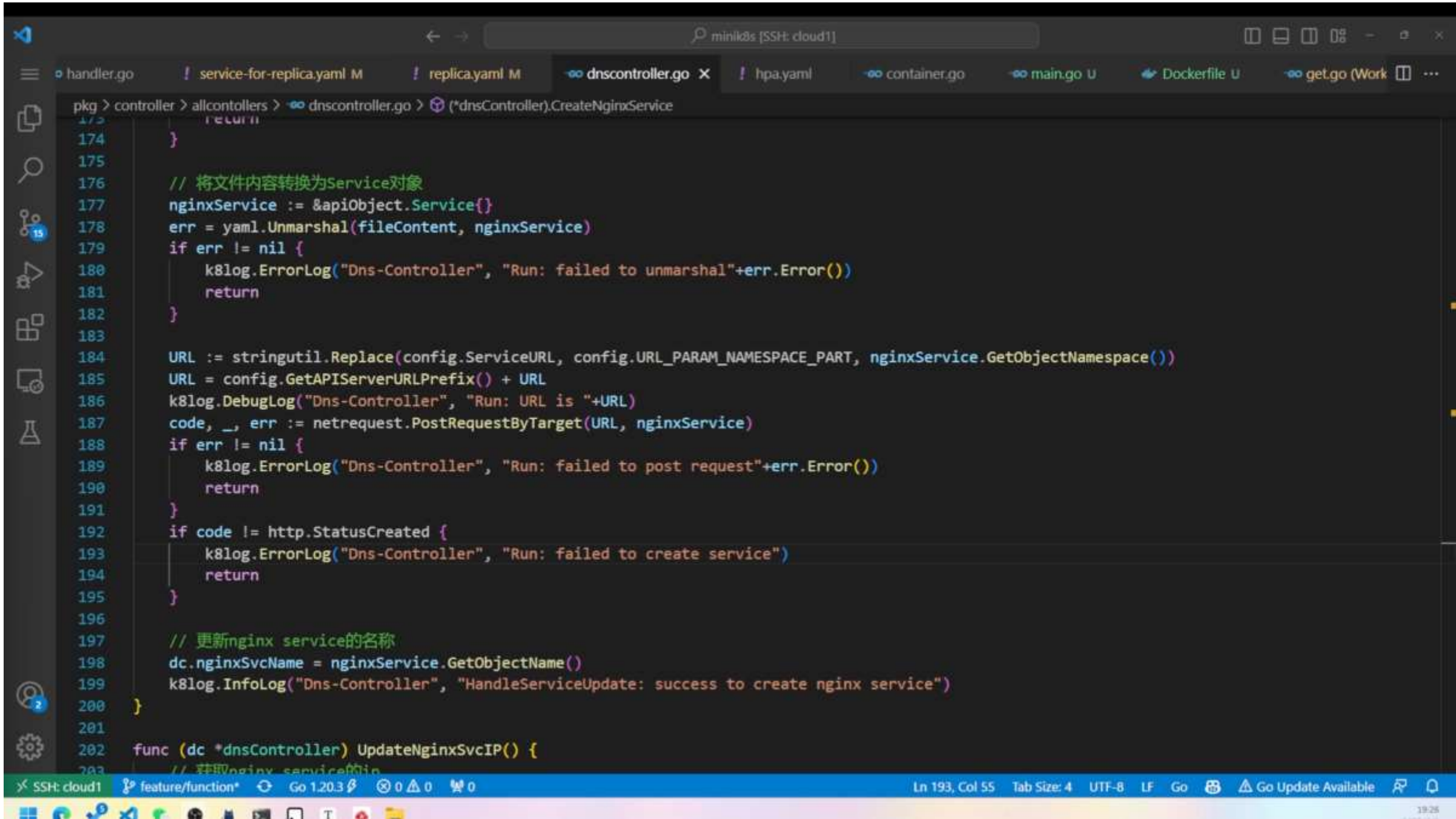
3.8serverless：function



3.8serverless：workflow



3.9fault tolerance



4 Thanks & QA