CKA 题库

1. 创建一个 pod 名称为 nginx，并将其调度到节点为 disk=stat 上

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

2. 创建一个 Pod 名称为 nginx-app，镜像为 nginx，并根据 pod 创建名为 nginx-app 的 Service，type 为 NodePort

```
kubectl run nginx-app --image=nginx
```

之后创建 service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-app
spec:
  selector:
    run: nginx-app
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 9376
  - name: https
    protocol: TCP
    port: 443
    targetPort: 9377
  type: NodePort
```

3. 将 deployment 为 nginx-app 的副本数从 1 变成 4。

```
#方法 1
kubectl scale  --replicas=4 deployment nginx-app
#方法 2，使用 edit 命令将 replicas 改成 4
kubectl edit deploy nginx-app
```

4. 监控 Pod bar 的日志，并提取与错误 file-not-found 相对应的日志行,将这些日志行写入 /opt/dir/bar

```
kubectl logs <podname> | grep Error > /opt/KUCC000xxx/KUCC000xxx.txt
```

考点：Monitor, Log, and Debug

5. 按 capacity 排序列出所有 persistent volumes, 将完整的 kubectl 输出保存到 /opt/dir/volume_list

```
kubectl get pv --sort-by='{spec.capacity.storage}'
```

考点：kubectl 命令熟悉程度

6. 列出环境内所有的 pv 并以 name 字段排序（使用 kubectl 自带排序功能）

```
kubectl get pv --sort-by=.metadata.name
```

考点：kubectl 命令熟悉程度

7. 按如下要求创建一个 Pod：
名称：jenkins
使用 image: jenkins
在名为 website-frontend 的 新 kubernetes namespace 中

```
# 方法一: 通过命令创建
kubectl run jenkins --image=jenkins --namespace=website-frontend --replicas=1 --
generator=run-pod/v1 --labels=app=jenkins --dry-run -o yaml
```

```
# 方法二: 编写 yaml
apiVersion: v1
kind: Pod
nameSpace: website-frontend
metadata:
  name: jenkins
spec:
  containers:
  - name: jenkins
    image: jenkins
```

8. 按如下要求创建一个 Pod：
名称：non-persistent-redis
Container image: redis
Persistent volume name: cache-control
Mount Path: /data/redis
应在 staging namespace 中发布，且该 volume 必须不能是永久的。

```
apiVersion: v1
kind: Pod
metadata:
  name: non-persistent-redis
  namespace: staging
spec:
  containers:
  - name: non-persistent-redis
    image: redis
    volumeMounts:
      - name: cache-control
        mountPath: /data/redis
  volumes:
  - name: cache-control
    emptyDir: {}
```

考点：Volume、emptdir

9. 确保在 Kubernetes cluster 的每个 node 上都运行 pod nginx 的一个实例，此处，nginx 也表示必须使用的 image 名称。切勿覆盖当前存在的任何 taint。

使用 DaemonSet 完成此任务，并使用 ds-kubesc12345 作为 DaemonSet 名称。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ds-kubesc12345
  annotations:
    wocao: wuqing
  labels:
    app: ds-kubesc12345
spec:
  selector:
    matchLabels:
      name: ds-kubesc12345
  template:
    metadata:
      labels:
        name: ds-kubesc12345
    spec:
      containers:
      - name: ds-kubesc12345
        image: busybox
        command:
        - sh
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
```

另外:建议直接先创建一个 deployment 然后修改 yaml 使其成为 daemonset 编排(注意删除 replicas 和其他不必要字段字段)

```
kubectl run ds-kubesc12345 --image nginx --image-pull-policy IfNotPresent --dry-run -o yaml
```

考点：DaemonSet

10. 按如下要求创建一个 Deployment:

Name: nginx-app
Image: nginx
ImageTag: 1.10.2-alpine
Replicas: 3

然后，执行滚动更新，使用新版本 1.11.13-alpine 部署应用并记录此更新。

最后，将此更新回滚至之前版本 1.10.2-alpine

```
kubectl run nginx-app --image=nginx:1.10.2-alpine
kubectl set image deployment nginx-app nginx-app=nginx:1.11.13-alpine --record
# 直接回滚到上一个版本
kubectl rollout undo deployment nginx-app
kubectl rollout status -w deployment nginx-app

# 通过查看历史记录回滚到指定版本
kubectl rollout history deploy/nginx
kubectl rollout undo deploy/nginx  --to-revision=2
```

考点：资源的更新

11. 创建一个 Deployment 的 spec 文件：

Image: nginx
Replicas: 4
Label: app_env_stage=prod
Name: kual12345
将此 spec 文件的副本保存至 /opt/dir/deployment_spec.yaml (或 .json)。

完成后，清理(删除)执行此任务时生成的任何新 Kubernetes API 对象。

```
kubectl run kual12345 --image nginx --replicas 4 --labels app_env_stage=prod --dry-run -o yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app_env_stage: prod
  name: kual12345
spec:
  replicas: 4
  selector:
    matchLabels:
      app_env_stage: prod
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app_env_stage: prod
    spec:
      containers:
      - image: nginx
        name: kual12345
        resources: {}
status: {}
```

12. 添加一个 init container 至 bumpy-llama (已在 spec 文件 /opt/dir/Pod-spec-12345.yaml 中定义)

init container 应该：创建一个名为 /workdir/faithful.txt 的 空文件

如果未检测到 /workdir/failthful.txt , Pod 应退出

一旦使用 init container 定义更新 spec 文件，则应创建 Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: hungry-bear
spec:
  volumes:
  - name: workdir
    emptyDir: {}
  containers:
  - name: checker
    image: busybox
    command: ["/bin/sh", "-c", "if [ -f /workdir/faithful.txt ] ; then sleep 100000; else exit
1; fi"]
    volumeMounts:
    - name: workdir
      mountPath: /workdir
  initContainers:
  - name: init-c
    image: busybox
    command: ["/bin/sh", "-c", "touch /workdir/faithful.txt"]
    volumeMounts:
    - name: workdir
      mountPath: /workdir
```

考点：init Container。一开始审题不仔细，以为要用到 livenessProbes

13.创建一个 Secret:

Name: super-secret
Credential: alice or username:bob

使用镜像 redis，创建一个名为 pod-secrets-via-file 的 Pod， 在挂载点/secrets 挂载 secret super-secret

创建第二个名为 pod-secrets-via-env 的 Pod，也适用镜像 redis,将 secret 的 credential/username 用于环境变量 TOPSECRET/CREDENTIALS

生成 secret 参考命令

```
kubectl create secret generic super-secret --from-literal=username=bob  --from-
literal=credential=alice
```

生成 yaml 文件的命令，再在此基础上改写(可以在 https://kubernetes.io/docs/页面搜索框输入 volumes 查询 example)

```
kubectl run pod-secrets-via-file --image=redis --generator=run-pod/v1 --dry-run -o
yaml >12-1.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod-secrets-via-file
  name: pod-secrets-via-file
spec:
  volumes:
  - name: super-secret
    secret:
      secretName: super-secret
  containers:
  - image: redis
    name: pod-secrets-via-file
    resources: {}
    volumeMounts:
    - name: super-secret
      mountPath: /secrets
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod-secrets-via-env
  name: pod-secrets-via-env
spec:
  volumes:
  - name: super-secret
    secret:
      secretName: super-secret
  containers:
  - image: redis
    name: pod-secrets-via-env
    resources: {}
    env:
    - name: CREDENTIALS
      valueFrom:
        secretKeyRef:
          name: super-secret
          key: username
    - name: TOPSECRET
      valueFrom:
        secretKeyRef:
          name: super-secret
          key: credential
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

14. 创建和配置 front-end-service service, 以便可通过 NodePort 访问该 service 并将其路由到名为 front-end 的现有 Pod

```
kubectl expose pod front-end --name=front-end-service --type='NodePort' --port=80
```

考点：Service

15. 创建一个名为 kucc3 的 Pod，在 Pod 里面分别为以下每个 image 单独运行一个 app container（可能会有 1 ~ 4 个 images）：

nginx + redis + memcached + consul

```
# 先用命令行创建一个示例
kubectl run kucc3 --image=nginx --image-pull-policy=IfNotPresent --replicas=1 --
generator=run-pod/v1 --dry-run -o yaml
```

```
# 而后修改示例
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: kucc4
  name: kucc4
spec:
  containers:
  - image: nginx
    name: nginx
  - image: redis
    name: redis
  - image: memcached
    name: memcached
  - image: consul
    name: consul
status: {}
```

考点：kubectl 命令熟悉程度、多个容器的 pod 的创建

16. 列出服务 foo 在命名空间 production 对应的所有 pods

```
[root@vms31 KUCC00302]# kubectl get svc --show-labels -n production
[root@vms31 KUCC00302]# kubectl get pods -l name=haha -n production |grep -v
NAME|awk '{print $1}' > /opt/KUCC00302/kucc00302.txt
```

17. 按如下要求调度一个 Pod：

名称：nginx-kucc2345
Image： nginx
Node Selector: disk=spinning

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    run: nginx-kucc2345
  name: nginx-kucc2345
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx-kucc2345
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: nginx-kucc2345
    spec:
      containers:
      - image: nginx
        name: nginx-kucc2345
        resources: {}
      nodeSelector:
        disk: spinning
status: {}
```

考点：pod 的调度。

参考：assign-pod-node

18. 创建名为 app-config 的 Persistent Volume，容量为 1Gi, 访问模式为 ReadWriteOnce。Volume 类型为 hostPath，位于 /srv/app-config。

```
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: app-config
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /srv/app-config
```

考点：创建 PV
参考：persistent volumes

19. 提供一个非完全正常运行的 Kubernetes Cluster, 在该 Cluster 中找出故障征兆。

确定 node 和出现故障的服务，采取措施修复故障服务，使 Cluster 恢复正常。确保所有更改永久有效。

提示：
可使用 ssh node-name 通过 ssh 命令连接到相应 node
可使用 sudo -i 命令在该 cluster 的任何 node 上获取更高权限

情况一
kubectl get cs 能看到 controller manager 没有启动
登陆到 master 上
找到相关服务重启
systemctl start kube-manager-controller.service

情况二
kubectl get node 显示 connection refuse，估计是 apiserver 的故障。
具体配置文件参考 static pod 那一题,主要解题思路如下:
1.为什么 api-server 没启动 (跟其他题的集群对比发现 api-server 配置文件相同。但是不生效)
2.为什么 kubelet 没有拉起来 api-server (跟其他题的集群对比 比如 static pod 那一题，可以发现 kubelet 没有指定静态 pod 目录)
3.kubelet 配置文件中加入 (--pod-manifest-path=xxxxxxx)，重启 kubelet systelctl restart kubelet

参考：Troubleshoot Clusters

20. 名为 wk8s-node-0 的 Kubernetes worker node 处于 NotReady 状态。

调查发生这种情况的原因，并采取相应措施将 Node 恢复为 Ready 状态，确保所做的任何更改永久有效。

提示：

可使用命令 ssh wk8s-node-0 连接到故障节点
可使用命令 sudo -i 在该节点上获取更高权限

```
#查看集群状态
kubectl get nodes
#查看故障节点信息
kubectl describe node node1

#Message 显示 kubelet 无法访问
#进入故障节点
ssh node1

#查看节点中的 kubelet 进程
ps -aux | grep kubelete
#没找到 kubelet 进程，查看 kubelet 服务状态
systemctl status kubelet.service
#kubelet 服务没启动，启动服务并观察
systemctl start kubelet.service
#启动正常，enable 服务
systemctl enable kubelet.service

#回到考试节点并查看状态
exit

kubectl get nodes #正常
```

21. 在本任务中，将配置一个新 Node ik8s-node-0 并将其加入一个 Kubernetes Cluster，方法如下：

配置 kubelet 以便自动轮换证书，且通过使用 RBAC 确保服务器和客户端的 CSRs 能够得到自动批准和签署

确保创建合适的 cluster-infok ConfigMap，并在正确的 Namespace 中进行相应的配置，以便后续的 Nodes 能够轻松加入该集群

用于引导的 kubeconfig 应创建在新 Node 的 /etc/kubernetes/bootstrap-kubelet.conf 上(切勿在 Node 成功加入集群后移除此文件)

相应的集群级 CA 证书位于 Node 的 /etc/kubernetes/pki/ca.crt 上，应确保所有自动签发的证书都安装到 Node 的 /var/lib/kubelet/pki 目录上，并且成功引导后，将在 /etc/kubernetes/kubelet.conf 渲染 kubelet 的 kubeconfig 文件

使用额外的组引导尝试加入集群的 Nodes，组的名称应为 system:bootstrappers:cka:default-node-token

解决方案应在系统启动时随着 kubelet 的 systemd service unit 文件(可在 /etc/systemd/system/kubelet.service 中找到) 一起自动启动

要测试解决方案，应通过位于 /opt/dir/kube-flannel.yaml 的 spec 文件创建相应的资源。该过程将创建必要的资源和 kube-flannel-ds DaemonSet。应确保将此 DaemonSet 正确部署到集群的单个(应该是每个)Node。

提示：

对于此任务，未在 ik8s-master-0 上配置或运行 kubelet, 请勿尝试配置

您将使用 TLS 引导来完成此任务

可通过以下命令获取 Kubernetes API 服务器的 IP 地址： ssh ik8s-node-0 getent hosts ik8s-master-0

API 服务器正在侦听常用端口 6443/tcp，且只会处理 TLS 请求

kubelet 二进制文件已安装到 ik8s-node-0 的 /usr/bin/kubelet 上。执行此任务期间，无需将 kube-proxy 部署到集群

可以使用 ssh ik8s-node-0 来连接到 Worker Node

可以使用 ssh ik8s-master-0 来连接到 Master Node

无需进一步配置在 ik8s-master-0 上运行的 Control Plane 服务

可使用 sudo -i 在这两个 Nodes 上获取更高权限

已在 ik8s-node-0 上安装并运行 Docker

另外:可能题目有另外一种说法,就是需要你去配置 worker 节点上 kube-proxy、kubelet 的 tls

参考：TLS Bootstrapping

22. 将标签为 ek8s-node-1 的 Node 设置为不可用，并重新调度该 Node 上所有运行的 Pods。

```
kubetl get node --show-labels |grep name=ek8s-node-1
kubectl drain ek8s-node-1
#如果直接 drain 会出错，需要添加--ignore-daemonsets --delete-local-data 参数
kubectl drain node node1  --ignore-daemonsets --delete-local-data
另外:使用 cordon 可以先暂停调度
kubectl cordon
```

考点：节点调度、维护

23. 为在 https://127.0.0.1:2379 运行的 etcd 实例创建快照，并将快照保存至文件路径 /data/backup/etcd-snapshot.db。

Etcd 实例运行的 etcd 版本为 3.3.10

以下 TLS 证书密钥用于通过 etcdctl 连接服务器：

CA 证书：/opt/dir/ca.crt
客户端证书：/opt/dir/etcd-client.crt
客户端密钥：/opt/dir/etcd-client.key

```
export ETCDCTL_API=3
etcdctl help 再 etcdctl snapshot save --help
etcdctl --endpoints=127.0.0.1:2379 --cacert=/opt/dir/ca.crt --cert=//opt/dir/etcd-client.crt --key=/opt/dir/etcd-client.key snapshot save /data/backup/etcd-snapshot.db
```

参考：backing up an etcd cluster

24. 按如下要求创建一个 Deployment：

名称：nginx-random
通过 Service 暴露：nginx-random
确保 Service 和 Pod 可通过各自的 DNS 记录访问
在此 Deployment 中运行的任何 Pod 内的 Container 都应使用 nginx image
接下来，使用实用工具 nslookup 查询该 Service 和 Pod 的 DNS 记录，并将输出结果分别写入 /opt/dir/service.dns 和 /opt/dir/Pod.dns

确保在任何测试中使用 busybox

```
kubectl run nginx-dns --image=nginx
kubectl expose deployment nginx-dns --port=80
kubectl get pods - o wide 获取 pod 的 IP
kubectl run busybox -it --rm --image=busybox:1.28 sh
      nslookup nginx-dns
      nslookup 获取 pod 的 IP
```

考点：网络相关，DNS 解析

参考：Debugging DNS Resolution

25. 通过 Pod Label name=cpu-user，找到运行时占用大量 CPU 的 Pod，并将占用 CPU 最高的 Pod 名称写入文件 /opt/dir/1234.txt (已存在)。

```
kubectl top pod |grep -iv name|sort -k 2 -n
```

列出 Service 名为 test 下的 pod 并找出使用 CPU 使用率最高的一个，将 pod 名称写入文件中

```
#使用-o wide 获取 service test 的 SELECTOR
kubectl get svc test -o wide
##获取结果我就随便造了
NAME  TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE      SELECTOR
test  ClusterIP   None       <none>      3306/TCP   50d      app=wordpress,tier=mysql

#获取对应 SELECTOR 的 pod 使用率，找到最大那个写入文件中
kubectl top test -l 'app=wordpress,tier=mysql'
```

考点：获取 service selector，kubectl top 监控 pod 资源

26. 检查有多少 Nodes 已准备就绪(不包括被打上 Taint: NoSchedule 的节点)，并将数量写入 /opt/dir/nodes.txt。

```
kubectl get nodes --show-labels|grep -vi schedule |grep -vi version |grep -i ready |wc -
l
```

```
#CheatSheet 方法，应该还能优化 JSONPATH
JSONPATH='{range .items[*]}{@.metadata.name}:{range
@.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"
```

考点：kubectl 命令熟悉程度

27. 扩容 deployment webserver 到 6 个 pods

```
[root@vms31 opt]# kubectl get deployments.
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
nginx-app  3       3        3           3          11h
webserver  1       1        1           1          174d
[root@vms31 opt]# kubectl scale --replicas=6 deployment/webserver
deployment.extensions/webserver scaled
[root@vms31 opt]# kubectl get deployments.
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
nginx-app  3       3        3           3          11h
webserver  6       6        6           1          174d
```

考点：deployment 的 Scaling，搜索 Scaling

参考：Scaling the application by increasing the replica count

28. configure the kubelet systemed managed service, on the node labelled with name=wk8s-node-1,to launch a pod containing a single container of image nginx named

myservice automatically.Any spec file requried should be placed in the /etc/kuberneteds/mainfests directory on the node

提示:

可 ssh 到错误节点$ ssh wk8s-node-0

可以通过命令获取权限$ sudo -i

```
vi /etc/systemd/system/kubelet.service
添加下面参数
--pod-manifest-path=/etc/kubernetes/manifests
使用下面的参考命令生成 pod 文件
kubectl run myservice --image=nginx --generator=run-pod/v1 --dry-run -o yaml >
/etc/kubernetes/manifests/24.yml
重启服务
systemctl start kubelet
```

考点：Static Pod

29. 创建 nginx-app 的 deployment， 使用镜像为 nginx:1.11.0-alpine ,修改镜像为 1.11.3-alpine，并记录升级，再使用回滚，将镜像回滚至 nginx:1.11.0-alpine

```
# 创建 nginx-app 的 deployment
kubectl run nginx-app --image=nginx:1.11.0-alpine --record
# 修改镜像， nginx-app 为 container 的名字
kubectl set image deployment nginx-app nginx-app=nginx:1.11.3-alipne
# 回滚
kubectl rollout undo deployment nginx-app
```

30. 使 node1 节点不可调度，并重新分配该节点上的 pod

```
#直接 drain 会出错，需要添加--ignore-daemonsets --delete-local-data 参数
kubectl drain node node1  --ignore-daemonsets --delete-local-data
```

31. 部署 ingress controller，创建 ingress，域名为***，使得 curl 该域名，返回 200 ok

```
# 部署 ingres controller
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/mandatory.yaml

#创建规则
[root@dce-bj-master-21 ingress]# cat ingress-rule.yml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: a2048
  labels:
    dce.daocloud.io/app: dao-2048
spec:
  rules:
  - host: 2048.fuck.you
    http:
      paths:
      - path: /1
        backend:
          serviceName: dao-2048
          servicePort: 80
      - path: /2
        backend:
          serviceName: dao-2049
          servicePort: 80
```

32. 创建一个固定结束次数的并行 Job，共 2 个 pod，运行 45 completion，Job pod 打印“Beijing”。镜像自选

```
cat job.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: busybox
spec:
  parallelism: 2
  completions: 45
  template:
    metadata:
      name: busybox
    spec:
      containers:
      - name: busybox
        image: busybox
        command: ["echo", "Beijing"]
      restartPolicy: Never
[root@conn01 ~]# kubectl describe job busybox
......
Pods Statuses:  0 Running / 45 Succeeded / 0 Failed
......
[root@conn01 ~]# kubectl logs  busybox-cj86j
Beijing
```

33. namespace secure 中存在一个 redis 的 pod，配置该 namespace 的 network policy，拒绝其它 namespace 中的 pod 的访问，指定某一个 namespace 中的 pod 可以访问 redis pod 的 6379 端口。集群的 k8s 版本为 1.6.2。参考：https://v1-6.docs.kubernetes.io/docs/concepts/services-networking/network-policies/

创建 NameSpace 的 Policy

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: secure
  name: deny-other-namespaces
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

创建 POD 和 POD 的 Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: secure
spec:
  podSelector:
    matchLabels:
      role: redis
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```

34. 创建一个 deployment，该 deployment 具有 app=***的 annotation，将 deployment 的 yaml 文件保存至指定目录

```
kubectl annotate deploy test-alpine app='ckaexam'
```

考点：Annotation

35. 使用 rbac 资源，包括 role、rolebinding 等，具体可参考官方文档

```
读取 ns default 下的 Pod 的权限和列出 Pod 的权限
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] #"" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

给用户 jane 读取 ns default 下的 Pod 的权限和列出 Pod 的权限
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

考点：RBAC

36. 登陆另一集群，该集群未配置网络，选择 calico 配置集群网络。创建 pod，配置 network policy。可参考 k8s 官网 network policy

考点：Network