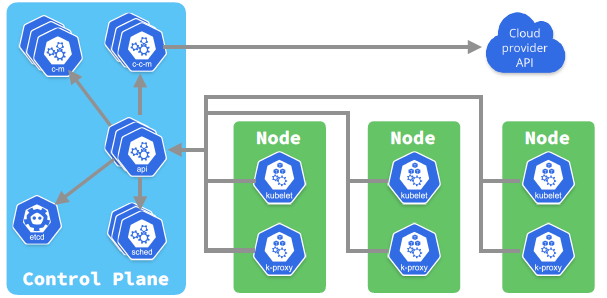
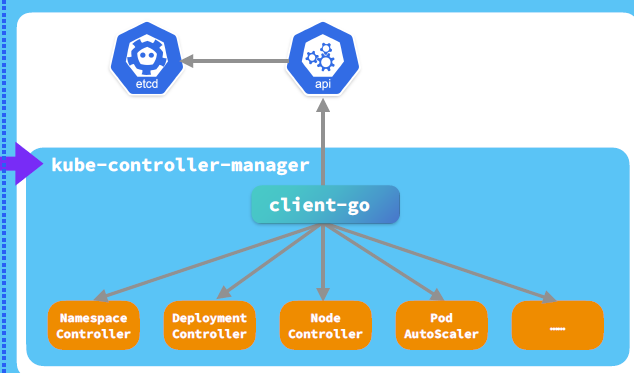
****

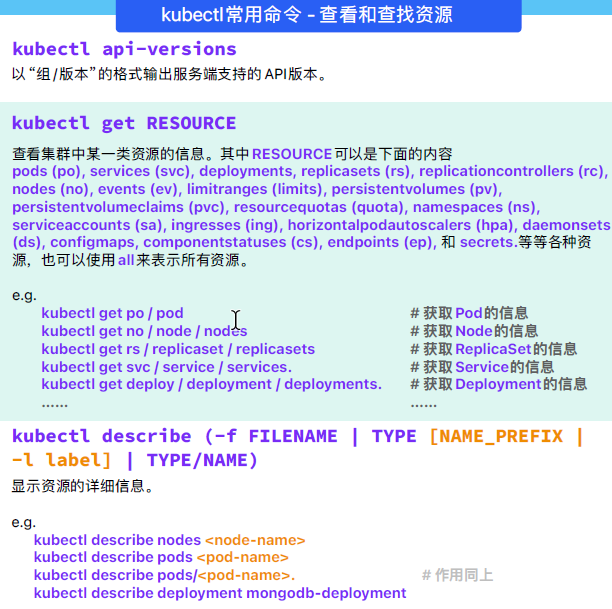
****

****

****

****

****

****

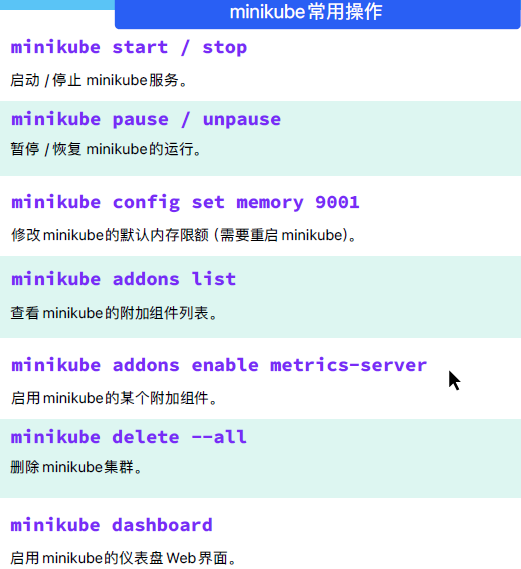
****

****

****

****

****

****

**1. 什么是[Kubernetes](https://kubernetes.io/" \t "_blank)**

[Kubernetes](https://kubernetes.io/)是一个开源的容器编排引擎，  
可以用来管理容器化的应用，  
包括容器的自动化的部署、扩容、缩容、升级、回滚等等，  
它是Google在2014年开源的一个项目，  
它的前身是Google内部的Borg系统。

**2. 为什么要使用**[**Kubernetes**](https://kubernetes.io/)

在[Kubernetes](https://kubernetes.io/" \t "_blank)出现之前，  
我们一般都是使用Docker来管理容器化的应用，  
但是Docker只是一个单机的容器管理工具，  
它只能管理单个节点上的容器，  
当我们的应用程序需要运行在多个节点上的时候，  
就需要使用一些其他的工具来管理这些节点，  
比如Docker Swarm、Mesos、[Kubernetes](https://kubernetes.io/" \t "_blank)等等，  
这些工具都是容器编排引擎，  
它们可以用来管理多个节点上的容器，  
但是它们之间也有一些区别，  
比如Docker Swarm是Docker官方提供的一个容器编排引擎，  
它的功能比较简单，  
适合于一些小型的、简单的场景，  
而Mesos和[Kubernetes](https://kubernetes.io/" \t "_blank)则是比较复杂的容器编排引擎，  
Mesos是Apache基金会的一个开源项目，  
而[Kubernetes](https://kubernetes.io/)是Google在2014年开源的，  
目前已经成为了CNCF（Cloud Native Computing Foundation）的一个顶级项目，  
基本上已经成为了容器编排引擎的事实标准了。

**3. 使用[minikube](https://minikube.sigs.k8s.io/" \t "_blank)搭建kubernetes集群环境**

[minikube](https://minikube.sigs.k8s.io/)是一个轻量级的kubernetes集群环境，  
可以用来在本地快速搭建一个单节点的kubernetes集群，

**3.1 安装minikube**

[minikube](https://minikube.sigs.k8s.io/)的安装：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | # macOS brew install minikube  # Windows choco install minikube  # Linux curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 sudo install minikube-linux-amd64 /usr/local/bin/minikube |

也可以到官网直接下载安装包来安装：[https://minikube.sigs.k8s.io/docs/start/](https://minikube.sigs.k8s.io/docs/start/" \t "_blank)

**3.2 启动minikube**

|  |  |
| --- | --- |
| 1 2 | # 启动minikube minikube start |

**4. 使用[Multipass](https://multipass.run/" \t "_blank)和[k3s](https://k3s.io/" \t "_blank)搭建kubernetes集群环境**

[minikube](https://minikube.sigs.k8s.io/)只能用来在本地搭建一个单节点的kubernetes集群环境，  
下面介绍如何使用[Multipass](https://multipass.run/)和[k3s](https://k3s.io/" \t "_blank)来搭建一个多节点的kubernetes集群环境，

**4.1 Multipass介绍**

[Multipass](https://multipass.run/)是一个轻量级的虚拟机管理工具，  
可以用来在本地快速创建和管理虚拟机，  
相比于VirtualBox或者VMware这样的虚拟机管理工具，  
[Multipass](https://multipass.run/)更加轻量快速，  
而且它还提供了一些命令行工具来方便我们管理虚拟机。  
官方网址: [https://Multipass.run/](https://multipass.run/)

**4.1.1 安装Multipass**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | # macOS brew install multipass  # Windows choco install multipass  # Linux sudo snap install multipass |

**4.1.2 Multipass常用命令**

关于Multipass的一些常用命令我们可以通过multipass help来查看，  
这里大家只需要记住几个常用的命令就可以了，

|  |  |
| --- | --- |
| 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 | # 查看帮助 multipass help multipass help <command>  # 创建一个名字叫做k3s的虚拟机 multipass launch --name k3s  # 在虚拟机中执行命令 multipass exec k3s -- ls -l  # 进入虚拟机并执行shell multipass shell k3s  # 查看虚拟机的信息 multipass info k3s  # 停止虚拟机 multipass stop k3s  # 启动虚拟机 multipass start k3s  # 删除虚拟机 multipass delete k3s  # 清理虚拟机 multipass purge  # 查看虚拟机列表 multipass list  # 挂载目录（将本地的~/kubernetes/master目录挂载到虚拟机中的~/master目录） multipass mount ~/kubernetes/master master:~/master |

Multipass有个问题，  
每次M1芯片的Mac升级之后Multipass的虚拟机都会被删除，  
不知道大家有没有遇到类似的问题。

|  |  |
| --- | --- |
| 1 2 3 4 | # 镜像位置 /var/root/Library/Application Support/multipassd/qemu/vault/instances # 配置文件 /var/root/Library/Application Support/multipassd/qemu/multipassd-vm-instances.json |

**4.2** [**k3s**](https://k3s.io/)**介绍**

[k3s](https://k3s.io/) 是一个轻量级的[Kubernetes](https://kubernetes.io/" \t "_blank)发行版，它是 [Rancher Labs](https://www.rancher.com/) 推出的一个开源项目，  
旨在简化[Kubernetes](https://kubernetes.io/)的安装和维护，同时它还是CNCF认证的[Kubernetes](https://kubernetes.io/" \t "_blank)发行版。

**4.2.1 创建和配置master节点**

首先我们需要使用multipass创建一个名字叫做k3s的虚拟机，

|  |  |
| --- | --- |
| 1 | multipass launch --name k3s --cpus 2 --memory 8G --disk 10G |

虚拟机创建完成之后，  
可以配置SSH密钥登录，  
不过这一步并不是必须的，  
即使不配置也可以通过multipass exec或者multipass shell命令来进入虚拟机，  
然后我们需要在master节点上安装[k3s](https://k3s.io/" \t "_blank)，

使用[k3s](https://k3s.io/)搭建kubernetes集群非常简单，  
只需要执行一条命令就可以在当前节点上安装[k3s](https://k3s.io/)，  
打开刚刚创建的k3s虚拟机，  
执行下面的命令就可以安装一个[k3s](https://k3s.io/)的master节点，

|  |  |
| --- | --- |
| 1 2 | # 安装k3s的master节点 curl -sfL https://get.k3s.io | sh - |

国内用户可以换成下面的命令，使用ranher的镜像源来安装：

|  |  |
| --- | --- |
| 1 | curl -sfL https://rancher-mirror.rancher.cn/k3s/k3s-install.sh | INSTALL\_K3S\_MIRROR=cn sh - |

安装完成之后，可以通过kubectl命令来查看集群的状态，

|  |  |
| --- | --- |
| 1 | sudo kubectl get nodes |

**4.2.2 创建和配置worker节点**

接下来需要在这个master节点上获取一个token，  
用来作为创建worker节点时的一个认证凭证，  
它保存在/var/lib/rancher/k3s/server/node-token这个文件里面，  
我们可以使用sudo cat命令来查看一下这个文件中的内容，

|  |  |
| --- | --- |
| 1 | sudo cat /var/lib/rancher/k3s/server/node-token |

将TOKEN保存到一个环境变量中

|  |  |
| --- | --- |
| 1 | TOKEN=$(multipass exec k3s sudo cat /var/lib/rancher/k3s/server/node-token) |

保存master节点的IP地址

|  |  |
| --- | --- |
| 1 | MASTER\_IP=$(multipass info k3s | grep IPv4 | awk '{print $2}') |

确认：

|  |  |
| --- | --- |
| 1 | echo $MASTER\_IP |

使用刚刚的TOKEN和MASTER\_IP来创建两个worker节点  
并把它们加入到集群中

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | # 创建两个worker节点的虚拟机 multipass launch --name worker1 --cpus 2 --memory 8G --disk 10G multipass launch --name worker2 --cpus 2 --memory 8G --disk 10G  # 在worker节点虚拟机上安装k3s  for f in 1 2; do  multipass exec worker$f -- bash -c "curl -sfL https://rancher-mirror.rancher.cn/k3s/k3s-install.sh | INSTALL\_K3S\_MIRROR=cn K3S\_URL=\"https://$MASTER\_IP:6443\" K3S\_TOKEN=\"$TOKEN\" sh -"  done |

这样就完成了一个多节点的kubernetes集群的搭建。

**5. 在线实验环境**

[Killercoda](https://killercoda.com/)

[Play-With-K8s](https://labs.play-with-k8s.com/)

**6. kubectl常用命令**

**6.1 基础使用**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | # 查看帮助 kubectl --help  # 查看API版本 kubectl api-versions  # 查看集群信息 kubectl cluster-info |

**6.2 资源的创建和运行**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | # 创建并运行一个指定的镜像 kubectl run NAME --image=image [params...] # e.g. 创建并运行一个名字为nginx的Pod kubectl run nginx --image=nginx  # 根据YAML配置文件或者标准输入创建资源 kubectl create RESOURCE # e.g. # 根据nginx.yaml配置文件创建资源 kubectl create -f nginx.yaml # 根据URL创建资源 kubectl create -f https://k8s.io/examples/application/deployment.yaml # 根据目录下的所有配置文件创建资源 kubectl create -f ./dir  # 通过文件名或标准输入配置资源 kubectl apply -f (-k DIRECTORY | -f FILENAME | stdin) # e.g. # 根据nginx.yaml配置文件创建资源 kubectl apply -f nginx.yaml |

**6.3 查看资源信息**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | # 查看集群中某一类型的资源 kubectl get RESOURCE # 其中，RESOURCE可以是以下类型： kubectl get pods / po # 查看Pod kubectl get svc # 查看Service kubectl get deploy # 查看Deployment kubectl get rs # 查看ReplicaSet kubectl get cm # 查看ConfigMap kubectl get secret # 查看Secret kubectl get ing # 查看Ingress kubectl get pv # 查看PersistentVolume kubectl get pvc # 查看PersistentVolumeClaim kubectl get ns # 查看Namespace kubectl get node # 查看Node kubectl get all # 查看所有资源  # 后面还可以加上 -o wide 参数来查看更多信息 kubectl get pods -o wide  # 查看某一类型资源的详细信息 kubectl describe RESOURCE NAME # e.g. 查看名字为nginx的Pod的详细信息 kubectl describe pod nginx |

**6.4 资源的修改、删除和清理**

|  |  |
| --- | --- |
| 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 | # 更新某个资源的标签 kubectl label RESOURCE NAME KEY\_1=VALUE\_1 ... KEY\_N=VALUE\_N # e.g. 更新名字为nginx的Pod的标签 kubectl label pod nginx app=nginx  # 删除某个资源 kubectl delete RESOURCE NAME # e.g. 删除名字为nginx的Pod kubectl delete pod nginx  # 删除某个资源的所有实例 kubectl delete RESOURCE --all # e.g. 删除所有Pod kubectl delete pod --all  # 根据YAML配置文件删除资源 kubectl delete -f FILENAME # e.g. 根据nginx.yaml配置文件删除资源 kubectl delete -f nginx.yaml  # 设置某个资源的副本数 kubectl scale --replicas=COUNT RESOURCE NAME # e.g. 设置名字为nginx的Deployment的副本数为3 kubectl scale --replicas=3 deployment/nginx  # 根据配置文件或者标准输入替换某个资源 kubectl replace -f FILENAME # e.g. 根据nginx.yaml配置文件替换名字为nginx的Deployment kubectl replace -f nginx.yaml |

**6.5 调试和交互**

|  |  |
| --- | --- |
| 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 | # 进入某个Pod的容器中 kubectl exec [-it] POD [-c CONTAINER] -- COMMAND [args...] # e.g. 进入名字为nginx的Pod的容器中，并执行/bin/bash命令 kubectl exec -it nginx -- /bin/bash  # 查看某个Pod的日志 kubectl logs [-f] [-p] [-c CONTAINER] POD [-n NAMESPACE] # e.g. 查看名字为nginx的Pod的日志 kubectl logs nginx  # 将某个Pod的端口转发到本地 kubectl port-forward POD [LOCAL\_PORT:]REMOTE\_PORT [...[LOCAL\_PORT\_N:]REMOTE\_PORT\_N] # e.g. 将名字为nginx的Pod的80端口转发到本地的8080端口 kubectl port-forward nginx 8080:80  # 连接到现有的某个Pod（将某个Pod的标准输入输出转发到本地） kubectl attach POD -c CONTAINER # e.g. 将名字为nginx的Pod的标准输入输出转发到本地 kubectl attach nginx  # 运行某个Pod的命令 kubectl run NAME --image=image -- COMMAND [args...] # e.g. 运行名字为nginx的Pod kubectl run nginx --image=nginx -- /bin/bash |

**7. Portainer的安装和使用**

[Portainer](https://www.portainer.io/) 是一个轻量级的容器管理工具，  
可以用来管理Docker和Kubernetes，  
它提供了一个Web界面来方便我们管理容器，  
官方网址: <https://www.portainer.io/>

**7.1 安装Portainer**

|  |  |
| --- | --- |
| 1 2 | # 创建一个名字叫做portainer的虚拟机 multipass launch --name portainer --cpus 2 --memory 8G --disk 10G |

当然也可以直接安装在我们刚刚创建的master节点上，

|  |  |
| --- | --- |
| 1 2 | # 在master节点上安装portainer，并将其暴露在NodePort 30777上 kubectl apply -n portainer -f https://downloads.portainer.io/ce2-19/portainer.yaml |

或者使用Helm安装

|  |  |
| --- | --- |
| 1 2 | # 使用Helm安装Portainer helm upgrade --install --create-namespace -n portainer portainer portainer/portainer --set tls.force=true |

然后直接访问 https://localhost:30779/ 或者 http://localhost:30777/ 就可以了，

**8. Helm的安装和使用**

[Helm](https://helm.sh/) 是一个Kubernetes的包管理工具，  
可以用来管理Kubernetes的应用，  
它提供了一个命令行工具来方便我们管理Kubernetes的应用，  
官方网址: <https://helm.sh/>

**8.1 安装Helm**

使用包管理器安装：

|  |  |
| --- | --- |
| 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 | # macOS brew install helm  # Windows choco install kubernetes-helm # 或者 scoop install helm  # Linux（Debian/Ubuntu） curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null sudo apt-get install apt-transport-https --yes echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list sudo apt-get update sudo apt-get install helm  # Linux（CentOS/Fedora） sudo dnf install helm  # Linux（Snap） sudo snap install helm --classic  # Linux（FreeBSD） pkg install helm |

使用脚本安装

|  |  |
| --- | --- |
| 1 2 3 | $ curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 $ chmod 700 get\_helm.sh $ ./get\_helm.sh |

或者

|  |  |
| --- | --- |
| 1 | curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash |

Share

* [Kubernetes](https://geekhour.net/tags/Kubernetes/)

[2023-11-01](https://geekhour.net/2023/11/01/mac-m1-pwn/)

[Security](https://geekhour.net/categories/Security/)

[**M系列Mac搭建Pwn环境**](https://geekhour.net/2023/11/01/mac-m1-pwn/)

**M系列Mac搭建Pwn环境**

虚拟机使用UTM，系统为Ubuntu 20.04  
其他虚拟机还试过Multipass、Parallels Desktop，  
还有Docker，但是都会遇到一些问题。

而UTM是基于QEMU的，可以模拟ARM、x86、x86\_64等架构，

**1. 安装UTM和Ubuntu**

UTM官方地址：<https://github.com/utmapp/UTM>

Mac版本地址：<https://mac.getutm.app/>