云享书库 第7期
bbs.huaweicloud.com

深入理解边缘计算

作者

崔广章

之江实验室助理研究员、浙江大学博士

参与过多个云计算生产项目

在边缘计算领域拥有多年经验

Python图像处理技术全面总结

了解开源项目的源码分析流程

能够对云、边、端主流开源实现进行定制开发

- 华为云开发者社区出品 -

目录

第 1 章 边缘计算入门

本章将从边缘计算系统的组成和概念解析、边缘计算的意义、边缘计算系统的部署与管理、不同应用部署方式的比较 4 个方面对边缘计算系统进行介绍。

1.1 边缘计算系统

本节从组成部分和概念解析两方面来说明边缘计算系统。

1）组成部分：边缘计算系统由云、边、端三部分组成，每部分的解决方案不止一种。本书的云组成部分选择 Kubernetes，边组成部分选择 KubeEdge，端组成部分选择 EdgeX Foundry。

2）概念解析：对组成边缘计算系统的云、边、端三部分涉及的相关概念进行说明。

1.1.1 边缘计算系统组成

1.云——Kubernetes

Kubernetes 是 Google 开源的大规模容器编排解决方案。整套解决方案由核心组件、第三方配套组件和运行时组成，具体如表 1-1 所示。

表 1-1 Kubernetes 组成部分说明

组成部分	组件名称	组件作用	备注
	Kube-apiserver	Kubernetes 内部组件相互通信的消息总线，对外暴露	

核心组件		集群 API 资源的唯一出口	
	Kube-controller	保证集群内部资源的现实状态与期望状态保持一致	
	Kube-scheduler	将需要调度的负载与可用资源最佳匹配	
	Kube-proxy	为节点内的负载访问和节点间的负载访问做代理	
	Kubelet	执行 kube-scheduler 的调度结果，操作相应负载	
第三方组件	etcd	存储集群的元数据和状态数据	
	flannel	集群的跨主机负载网络通信的解决方案	需要对原来的数据包进行额外的封装、解封装，性能

			损耗较大
	Calico	集群的跨主机负载网络通信的解决方案	纯三层网络解决方案，不需要额外的封装、解封装，性能损耗较小
	coredns	负责集群中负载的域名解析	
容器运行时	docker	目前默认容器运行时	
	Containerd	比 docker 轻量，稳定性与 docker 相当容器运行时	
	Cri-o	轻量级容器运行时	目前稳定性没有保证
	frakti	基于 hypervisor 的容器运行时	目前稳定性没有保证

2.边——KubeEdge

KubeEdge 是华为开源的一款基于 Kubernetes 的边缘计算平台，用于将容器化应用的编排功能从云扩展到边缘节点和设备，并为云和边缘之间的网络、应用部署和元数据同步

提供基础架构支持。KubeEdge 使用 Apache 2.0 许可，并且可以免费用于个人或商业用途。KubeEdge 由云部分、边缘部分和运行时组成，具体如表 1-2 所示。

表 1-2 KubeEdge 组成部分说明

组成部分	组件名称	组件作用	备注
云部分	CloudCore	负责将云部分的事件和指令下发到边缘端，同时接受边缘端上报的状态信息和事件信息	
边缘部分	EdgeCore	接收云部分下发的事件和指令，并执行相关指令，同时将边缘的状态信息和事件信息上报到云部分	
运行时	docker	目前，KubeEdge 默认支持 docker	官方表示未来会支持 containerd、cri-o 等容器运行时

3.端——EdgeX Foundry

EdgeX Foundry 是一个 Linux 基金会运营的开源边缘计算物联网软件框架项目。该项目的核心是基于与硬件和操作系统完全无关的参考软件平台建立的互操作框架，构建即插即用的组件生态系统，加速物联网方案的部署。EdgeX Foundry 使有意参与的各方在开放与互操作的物联网方案中自由协作，无论其是使用公开标准还是私有方案。

EdgeX Foundry 微服务集合构成了 4 个微服务层及 2 个增强的基础系统服务。4 个微服务层包含从物理域数据采集到信息域数据处理等一系列的服务，2 个增强的基础系统服务为 4 个微服务层提供服务支撑。

4 个微服务层从物理层到应用层依次为：设备接入服务层（Device Service）、核心服务层（Core Service）、支持服务层（Supporting Service）、导出服务层（Export Service），2 个增强的基础系统服务包括安全和系统管理服务，具体说明如表 1-3 所示。

表 1-3 Edgex Foundry 组成部分说明

组成部分	组件名称	组件作用	备注
设备接入层服务	Device-modbus-go	Golang 实现对接使用 modbus 协议设备的服务	
	Device-camera-go	Golang 实现对接摄像头设备的服务	
	Device-snmp-go	Golang 实现对接 SNMP 服务	

	Device-mqtt-go	Golang 实现对接使用MQTT协议设备的服务	
	Device-sdk-go	Golang 实现对接其他设备的 SDK	SDK 给设备接入提供了较大的灵活性
核心服务层	Core-command	负责向南向设备发送命令	
	Core-metadata	负责设备自身能力描述，提供配置新设备并将它们与其拥有的设备服务配对的功能	
	Core-data	负责采集南向设备层数据，并向北向服务提供数据服务	
	Registry&config	负责服务注册与发现，为其他 EdgeX Foundry 微服务提供关于 EdgeX	

		Foundry 内相关服务的信息，包括微服务配置属性，采用开源 consul 实现	
支持服务层	Support-logging	负责日志记录	
	Support-notifications	负责事件通知	
	Support-scheduler	负责数据调度	
导出服务层	Export-client	导出数据的客户端	
	Export-distro	导出数据的应用	
两个增强基础服务	System-mgmt-agent	提供启动、停止所有微服务的 API	
	Sys-mgmt-executor	负责启动、停止所有微服务的最终执行	

1.1.2 概念解析

组成边缘计算系统的云、边、端三部分的相关概念如下。

- 云：涉及的概念包括 Container、pod、ReplicaSet、Service、Deployment、Daemonset、Job、Volume、ConfigMap、Namespace、Ingress 等。
- 边：目前边缘系统的实现方式是通过对云原有的组件进行裁剪下沉到边缘，所以边涉及的概念是云的子集，而且与云保持一致。
- 端：部署在边上的一套微服务，目前没有引入新的概念。

目前，边和端都在沿用云的概念，所以本节主要是对云的概念进行解析。下面以图解的形式对云涉及的相关概念进行说明。由图 1-1 可知，Container 是在操作系统之上的一种新的环境隔离技术，使用容器隔离出的独立空间包含应用所需的运行时环境和依赖库。在同一台主机上,容器共享操作系统内核。

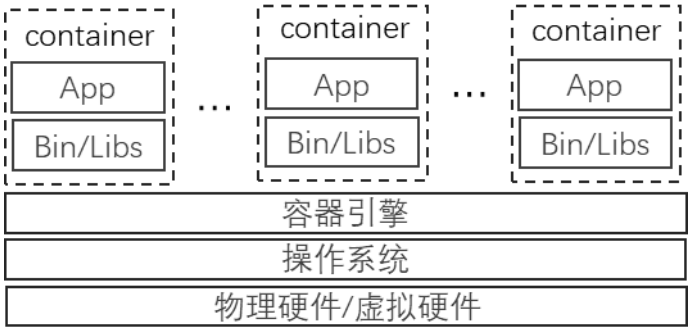

图 1-1 container 解析

由图 1-2 可知，pod 是由一组容器组成的，在同一个 pod 内的容器共享存储和网络命名空间。在边缘计算系统中，pod 是最小的可调度单元，也是应用负载的最终载体。

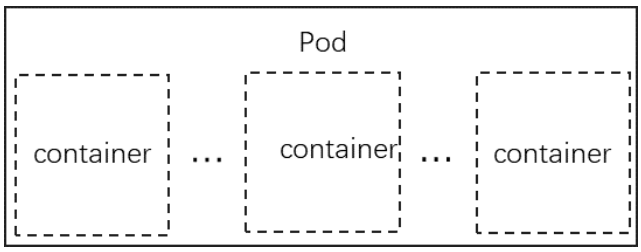

图 1-2 pod 解析

由图 1-3 可知，ReplicaSet 用来管理 pod，负责将 pod 的期望数量与 pod 真实数量保持一致。在边缘计算系统中，ReplicaSet 负责维护应用多实例和故障自愈。

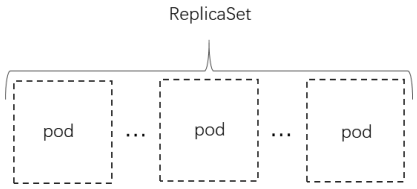

图 1-3 ReplicaSet 解析

由图 1-4 可知，service 做为一组 pod 的访问代理，并在多个 pod 之间做负载均衡。pod 的生命周期相对比较短暂，变更频繁。service 作为一组 pod 固定不变的标记，除了为与之相关的 pod 做访问代理和负载均衡外，还会维护 service 与 pod 的对应关系。

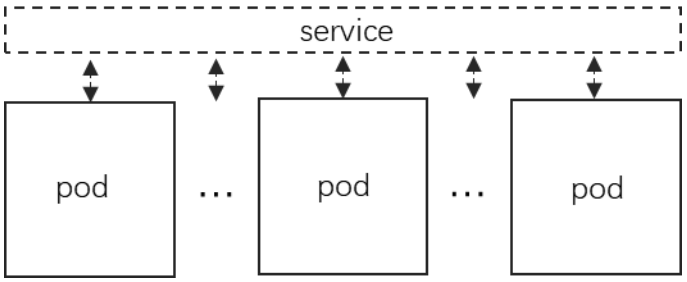

图 1-4 Service 解析

由图 1-5 可知，Deployment 是 Replicaset 的抽象，在 Replicaset 的基础上增加了增加了一些高级功能，功能和应用场景与 Replicaset 相同。

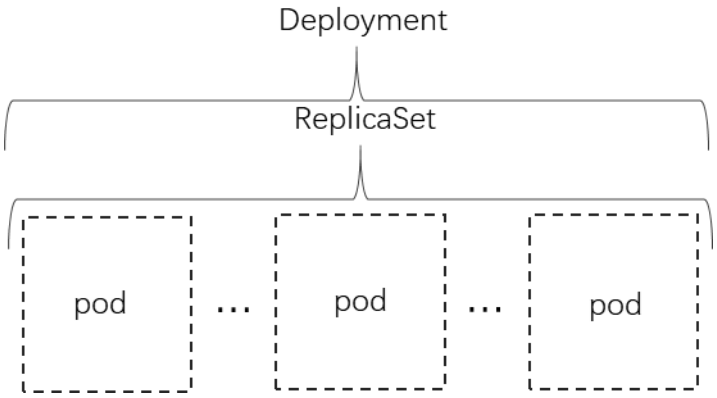

图 1-5 Deployment 解析

由图 1-6 可知，DaemonSet 负责将指定的 pod 在每个节点上都启动一个实例。该功能一般用在部署网络插件、监控插件和日志插件的场景。

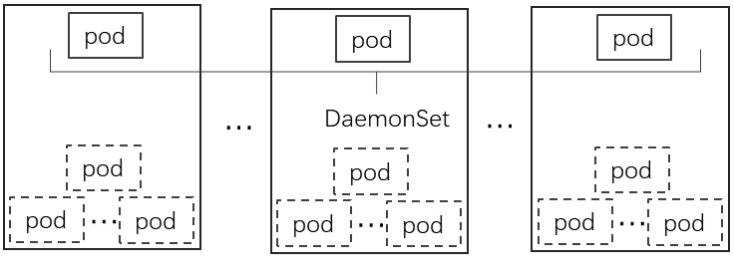

图 1-6 DaemonSet 解析

由图 1-7 可知，Job 用来管理批量运行的 pod，该管理类型的 pod 会被定期批量触发。与 deployment 管理的 pod 的不同，Job 管理的 pod 执行完相应的任务后就退出，不会一直驻留。在边缘计算系统中，一般用 Job 所管理的 pod 来训练 AI 模型。

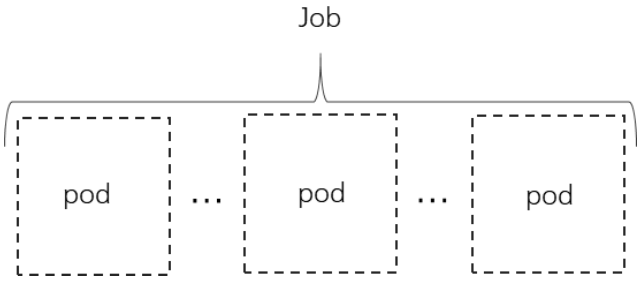

图 1-7 Job 解析

由图 1-8 可知，Volume 是用来给 pod 提供存储的，通过挂载的方式与对应 pod 关联。Volume 分临时存储和持久存储，临时存储类型的 Volume 会随着 pod 的删除被删除，持久存储类型的 Volume 不会随着 pod 的删除被删除。

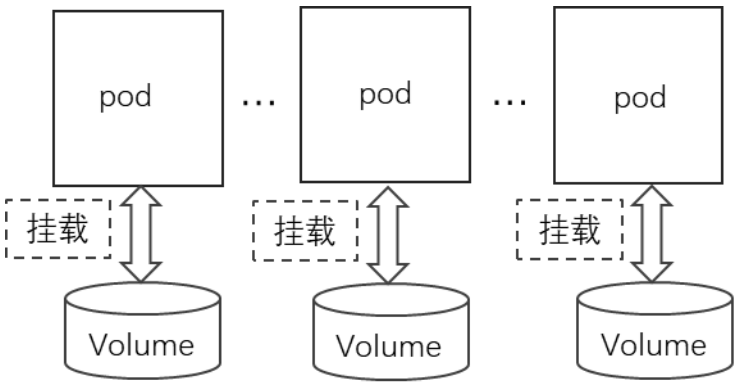

图 1-8 Volume 解析

由图 1-9 可知，ConfigMap 做为 pod 存储配置文件的载体，通过环境变量（env）和文件卷的方式与 pod 进行关联。在边缘计算系统中，以 ConfigMap 方式来管理配置信息会更方便。ConfigMap 还可以对配置中的敏感信息进行加密，使配置信息更安全。

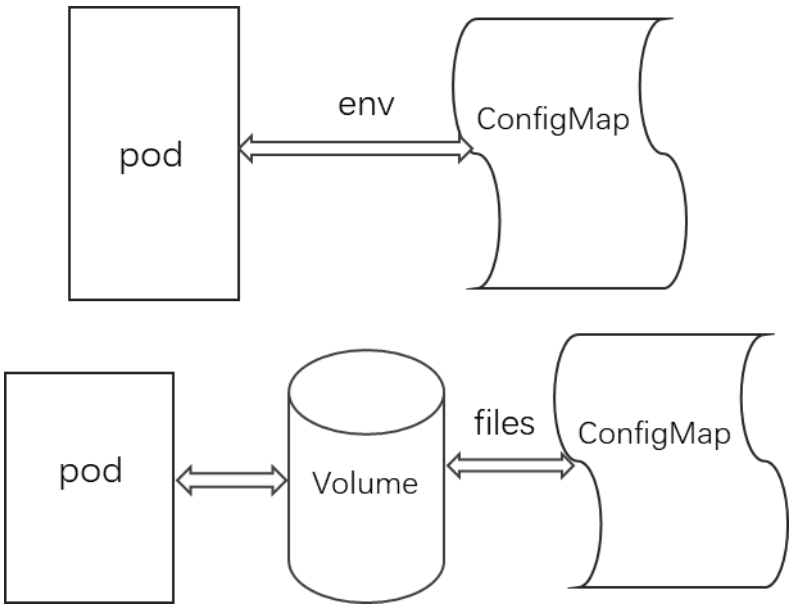

图 1-9 ConfigMap 解析图

由图 1-10 可知，Namespace 是对 pod、service、configmap、deployment、daemonset 等资源进行隔离的一种机制，一般用在同一公司的不同团队隔离资源的场景。在边缘计算系统中，用 Namespace 来对一个团队可以使用的资源（CPU、内存）和创建的负载所需要的资源进行限制。

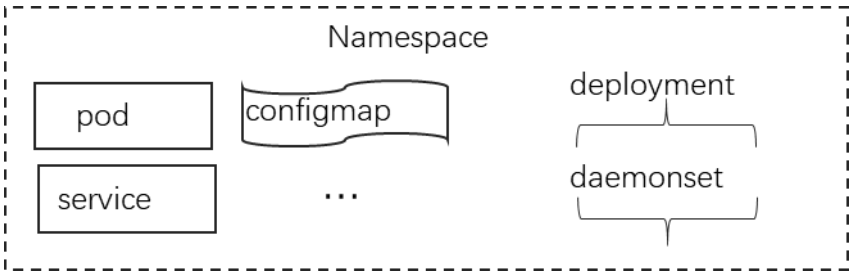

图 1-10 Namespace 解析图

由图 1-11 可知，Ingress 作为集群内与集群外相互通信的桥梁，可以通过 Ingress 将集群内的服务暴露到集群外，同时可以对进入集群内的流量进行合理的管控。在边缘计算系统中，Ingress 是一种资源对象，需要配合 Ingress Controller 和反向代理工作。

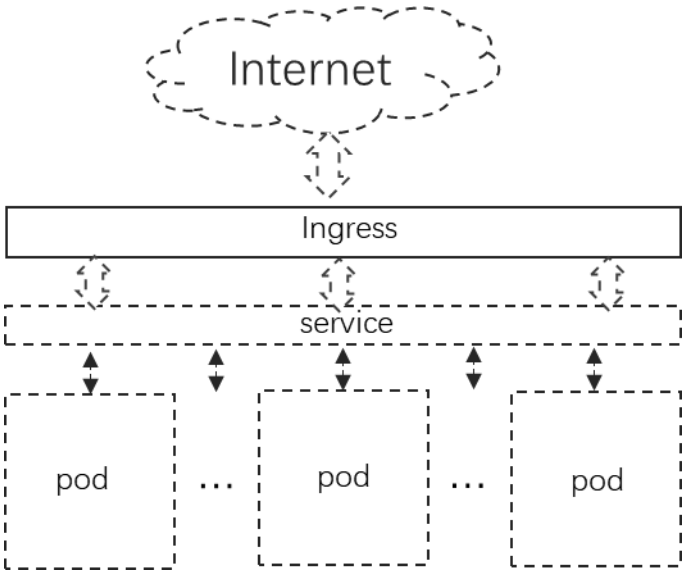

图 1-11 Ingress 解析

1.2 边缘计算的意义

随着移动互联网技术的发展，智能终端设备和各种物联网设备的数量急剧增加。在 5G 和万物互联时代，传统云计算中心集中存储、计算的模式已经无法满足终端设备对低时延、高带宽、强算力的需求。将云数据中心的计算能力下沉到边缘，甚至终端设备，并通过云数据中心进行统一交付、运维、管理，是云计算的趋势，从而催生了边缘计算。

边缘计算为终端用户提供实时、动态和智能的服务计算。边缘计算会将计算推向更接近用户的实际现场，这与需要在云端进行计算的传统云计算有着本质的区别，而这些区别主要表现在带宽负载、资源浪费、安全隐私保护以及异构多源数据处理上。

1.3 边缘计算系统的部署与管理

本节对边缘计算系统的部署采用两个节点的形式，将边缘计算系统的云部分 Kubernetes 部署在云控制节点，边缘部分 KubeEdge 和端部分 EdgeX Foundry 部署在

边缘计算上。这样做的目的是让读者能够快速部署边缘计算系统。通过操作已运行的系统对本书要讲的边缘计算系统有一个感性的认识。

1.3.1 系统部署

本节将对边缘计算系统部署所需要的主机环境和部署 docker、Kubernetes、KubeEdge 和 EdgeX Foundry 的相关步骤进行说明。

1.主机环境

表 4 是部署边缘计算系统的两台主机的详细配置，该环境包含两个节点，即云控制节点和边缘节点。

表 1-4 部署边缘计算系统主机配置

	操作系统	CPU/内存	磁盘	带宽
云控制节点	CentOS 7.7 64 位	2vCPU/8GiB	100GiB	2Mbit/s
边缘节点	CentOS 7.7 64 位	4vCPU/16GiB	40GiB	2Mbit/s

2.部署 docker

本节 docker 的安装步骤适合 CentOS 7.7 64 位操作系统，具体安装步骤如下。其他操作系统请参考 docker 官网相关安装文档。

1）卸载之前安装的老版本 docker（可选），命令如下：

yum remove docker docker-client docker-client-latest docker-common docker-latest docker-latest-logrotate docker-logrotate docker-engine

2）安装 docker Repository，命令如下：

yum install -y yum-utils device-Mapper-persistent-data lvm2

配置安装 docker 时需要的仓库链接：# yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

3）安装 docker Engine-Community(最新版本)，命令如下：

安装 docker：# yum install docker-ce docker-ce-cli containerd.io

4）查看已安装的 docker 相关包，命令如下：

yum list docker-ce --showduplicates | sort -r

5）启动 docker，命令如下：

systemctl start docker

6）确认 docker 已正常运行

查看 docker 运行状态，确认 docker 已经正常运行，命令如下：

systemctl status docker

如果输出类似图 1-12 的信息，说明 docker 已经正常运行。

```
[root@all-in-one ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since Sun 2020-01-19 15:07:51 CST; 1min 13s ago
     Docs: https://docs.docker.com
 Main PID: 2071 (dockerd)
    Tasks: 12
   Memory: 46.7M
   CGroup: /system.slice/docker.service
           └─2071 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

图 1-12 docker 运行状态

3.部署 Kubernetes

下面介绍 Kubernetes16.5 部署。本书的边缘计算系统中只需要部署 Kubernetes 的 master 节点来作为边缘计算系统的云控制中心，为了使部署 Kubernetes 教程更完整，也将部署 Kubernetes 的 node 节点的步骤包含了进来。

（1）安装 Kubernetes master 节点

1）在需要运行 Kubelet 的节点上关闭 swap，命令如下：

```
# swapoff -a
```

2）关闭防火墙，命令如下：

```
# systemctl disable firewalld && systemctl stop firwalld
```

3）关闭 SELinux，命令如下：

```
# setenforce 0
```

4）下载所需的 binary 和 images 压缩包并解压，命令如下：

server binary 是 Kubernetes GitHub 上 release 的编译好的 Kubernetes 版本，包括各组件的二进制和镜像。进入Kubernetes release页面，点击某个 release 的

changelog，如 CHANGELOG-1.16.5.md ，下载其中的 server binary 压缩包。下载完成的安装包如下所示：

```
[root@all-in-one ~]# ls
kubernetes-server-linux-amd64.tar.gz
```

解压安装包命令：# tar -zxvf Kubernetes-server-linux-amd64.tar.gz

通过上述命令解压后，我们会看到类似图 1-13 的内容。

```
[root@all-in-one ~]# tar -zxvf kubernetes-server-linux-amd64.tar.gz
kubernetes/
kubernetes/server/
kubernetes/server/bin/
kubernetes/server/bin/apiextensions-apiserver
kubernetes/server/bin/kube-controller-manager.tar
kubernetes/server/bin/mounter
kubernetes/server/bin/kube-proxy.docker_tag
kubernetes/server/bin/kube-controller-manager.docker_tag
kubernetes/server/bin/kube-proxy.tar
kubernetes/server/bin/kubectl
kubernetes/server/bin/kube-scheduler.tar
kubernetes/server/bin/kube-apiserver.docker_tag
kubernetes/server/bin/kube-scheduler
kubernetes/server/bin/kubeadm
kubernetes/server/bin/kube-controller-manager
kubernetes/server/bin/kube-scheduler.docker_tag
kubernetes/server/bin/kubelet
kubernetes/server/bin/kube-proxy
kubernetes/server/bin/kube-apiserver.tar
kubernetes/server/bin/hyperkube
kubernetes/server/bin/kube-apiserver
kubernetes/LICENSES
kubernetes/kubernetes-src.tar.gz
kubernetes/addons/
```

图 1-13 Kubernetes server binary 压缩包解压

通过图 1-13 可知，压缩包 Kubernetes-server-linux-amd64.tar.gz 解压成文件夹 Kubernetes,所需的 binary 和 image 都在 kuernetes/server/bin 目录下。

5）把 Kubernetes/server/bin 里的 kubeadm、 Kubelet、kubectl 三个 binary 复制到 /usr/bin 下，命令如下：

#cp Kubernetes/server/bin/kubectl Kubernetes/server/bin/kubeadm Kubernetes/server/bin/Kubelet /usr/bin

6）提前加载控制平面镜像。

根据官方文档中 Running kubeadm without an internet connection 小节内容，kubeadm 在执行 init 过程中需要启动控制平面，因此需要在此之前将控制平面的对应版本的镜像准备好，包括 apiserver、 controller manager、scheduler 和 kubeproxy 组件镜像，然后将 Kubernetes/server/bin 中包含的镜像压缩包加载到 master 节点，命令如下：

#docker load -i kube-scheduler.tar

但是，etcd 和 pause 镜像需要通过其他途径（如 docker Hub）来获得。

7）下载 Kubelet 的 systemd unit 定义文件，命令如下：

export RELEASE=v1.16.5

#curl -sSL "https://raw.GitHubusercontent.com/Kubernetes/Kubernetes/${RELEASE}/build/debs/Kubelet.service" > /etc/systemd/system/Kubelet.service

其中，RELEASE 变量需要提前 export 出来，如 v1.16.5。

8）下载 kubeadm 配置文件，命令如下：

#mkdir -p /etc/systemd/system/Kubelet.service.d

#curl -sSL

9）设置 Kubelet 开机自启动，命令如下：

#systemctl enable Kubelet

10）初始化 master 节点，命令如下：

#kubeadm init --Kubernetes-version=v1.16.5

--pod-network-cidr=10.244.0.0/16

其中，Kubernetes-version 告诉 kubeadm 具体需要安装什么版本的 Kubernetes；pod-network-cidr=192.168.0.0/16 flflag 的值跟具体网络方案有关，这个值对应后面的 Calico网络方案。如果安装的是 flflannel，则 pod-network-cidr 的值应该是 10.244.0.0/16。

注意：如果所有镜像就绪，则 kubeadm init 步骤执行时间只需几分钟。如果安装过程遇到错误需要重试，则重试之前运行 kubeadm reset 。

11）配置 kubectl。

由于下面安装 pod网络时使用了 kubectl，因此需要在此之前执行如下配置。

- 如果后续流程使用root 账户，则执行：

#export KUBECONFIG=/etc/Kubernetes/admin.conf

注意：为了方便，我们可以将该命令写到<home>/.profifile 下。

- 如果后续流程使用非root 账户，则执行：

mkdir -p $HOME/.kube

cp -i /etc/Kubernetes/admin.conf $HOME/.kube/config

chown $(id -u):$(id -g) $HOME/.kube/config

12）安装 pod 网络。

这里选择 Calico，按照 Kubernetes 官方安装文档操作即可，命令如下：

#kubectl apply -f

Calico 的 yaml 文件链接为：

https://docs.projectCalico.org/v3.7/manifests/Calico.yaml

13）允许 pod 调度到 master 节点上，否则 master 节点的状态会是 not ready（可选，如果用来做单节点集群则执行此步）。

默认会在执行 kubeadm init 过程中，通过执行以下命令使 pod 调度到 master 节点上：

#kubectl taint nodes --all node-role.Kubernetes.io/master-

14）执行到这步，我们已经有了一个单节点 Kubernetes 集群，可以运行pod。如果需要更多节点加入，可以把其他节点集合到集群。安装就绪的单节点集群如图 1-14 所示。

```
[root@cloud ~]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-778676476b-khwn6      1/1     Running   3          19d
kube-system   calico-node-wlfb9                             1/1     Running   2          19d
kube-system   coredns-6955765f44-24xf5                      1/1     Running   2          19d
kube-system   coredns-6955765f44-jh4mz                      1/1     Running   2          19d
kube-system   etcd-izm5eg14ele2q1188xvph2z                  1/1     Running   2          19d
kube-system   kube-apiserver-izm5eg14ele2q1188xvph2z        1/1     Running   2          19d
kube-system   kube-controller-manager-izm5eg14ele2q1188xvph2z   1/1     Running   3          19d
kube-system   kube-proxy-h5xqg                              1/1     Running   3          19d
kube-system   kube-scheduler-izm5eg14ele2q1188xvph2z        1/1     Running   2          19d
```

图 1-14 单节点集群负载

（2）安装 Kubernetes node 节点（可选）

1）swapoff -a //关闭内存 swap 分区

2）安装 docker。

3）安装 kubeadm、Kubelet。

详细参考安装 Kubernetes master 节点的步骤。

将安装 Kubernetes master 时下载的 server binary 里包含的 kubeadm、Kubelet 两个 binary 复制到 /usr/bin 下。

4）准备镜像。

把安装 Kubernetes master 时下载的 kube-proxy、pause 镜像转移到该节点并加载。

5）为 Kubelet 和 kubeadm 准备配置文件，命令如下：

```
# export RELEASE=v1.16.5

#curl -sSL
"https://raw.GitHubusercontent.com/Kubernetes/Kubernetes/${RELEASE}/build/
debs/Kubelet.service" > /etc/systemd/system/Kubelet.service

#mkdir -p /etc/systemd/system/Kubelet.service.d

#curl -sSL
"https://raw.GitHubusercontent.com/Kubernetes/Kubernetes/${RELEASE}/b
uild/debs/10-kubeadm.conf" >
/etc/systemd/system/Kubelet.service.d/10-kubeadm.conf
```

6）设置 Kubelet 开机启动自动，命令如下：

systemctl enable Kubelet

7）计算节点加入集群，命令如下：

在 node 上执行：

kubeadm join --token : --discovery-token-ca-cert-hash sha256:

注意：这条命令在 master 节点执行 kubeadm init 结束时会在 console 上显示。

4.部署 KubeEdge

在 Kubernetes 已经安装成功的基础上安装 KubeEdge1.1.0，使用 Kubernetes 的 master 节点作为其云控制节点。

（1）安装 Cloud 部分

1）修改 Kubernetes master 节点配置。

cloud 端是 KubeEdge 中与 kube-apiserver 交互的组件，在该教程中 cloud 端与 kube-apiserver 交互使用的是非安全端口，需要在 Kubernetes master 节点上做如下修改：

#vi /etc/Kubernetes/manifests/kube-apiserver.yaml

- --insecure-port=8080

- --insecure-bind-address=0.0.0.0

2）下载安装包。

可以通过两种方式下载安装包：通过 curl 直接下载；在 KubeEdge 的 release 仓库中下载。

第一种方式：通过 curl 直接下载。

VERSION="v1.0.0"

OS="linux"

ARCH="amd64"

curl -L "https://GitHub.com/KubeEdge/KubeEdge/releases/download/${VERSION}/KubeEdge-${VERSION}-${OS}-${ARCH}.tar.gz" --output KubeEdge-${VERSION}-${OS}-${ARCH}.tar.gz && tar -xf KubeEdge-${VERSION}-${OS}-${ARCH}.tar.gz -C /etc

注意：通过 curl 直接下载，由于网速问题一般需要时间比较久，失败的可能较大。

第二种方式：在 KubeEdge 的 release 仓库中下载。

进入 KubeEdge 的 GitHub 仓库中的 KubeEdge v1.0.0 release，下载 KubeEdge-v1.0.0-linux-amd64.tar.gz，将下载的安装包上传到 Kubernetes master 节点的/root 目录下，进行如下操作：

#tar -zxvf KubeEdge-v1.0.0-linux-amd64.tar.gz

mv KubeEdge-v1.0.0-linux-amd64 /etc/KubeEdge

3）在 Kubernetes master 节点生成证书。

生成的证书用来在 KubeEdge 的 edge 与 cloud 端加密通信。证书生成命令如下：

#wget -L

https://raw.GitHubusercontent.com/KubeEdge/KubeEdge/master/build/tool

s/certgen.sh

#chmod +x certgen.sh bash -x ./certgen.sh genCertAndKey edge

注意：上述步骤执行成功之后，会在/etc/KubeEdge 下生成 ca 和 certs 两个目录。

4）创建 device model 和 device CRDs。

在 Kubernetes master 节点上创建 KubeEdge 所需的 device model 和 device CRD。创建步骤如下：

#wget -L

https://raw.GitHubusercontent.com/KubeEdge/KubeEdge/master/build/crds

/devices/devices_v1alpha1_devicemodel.yaml

#chmod +x devices_v1alpha1_devicemodel.yaml

#kubectl create -f devices_v1alpha1_devicemodel.yaml

#wget -L

https://raw.GitHubusercontent.com/KubeEdge/KubeEdge/master/build/crds

/devices/devices_v1alpha1_device.yaml

#chmod +x devices_v1alpha1_device.yaml

#kubectl create -f devices_v1alpha1_device.yaml

5）运行 cloud 端。

在 Kubernetes master 节点上运行 KubeEdge 的 cloud 端，命令如下：

#cd /etc/KubeEdge/cloud

#./CloudCore

注意：本节为了方便查看进程输出采用了前台驻留进程的方式，除了上述方式外，还可以通过 systemd 来查看。

（2）安装 Edge 部分

edge 端是 KubeEdge 运行在边缘设备上的部分，在 edge 端运行之前需要安装合适的容器运行时，包括 docker、containerd 和 cri-o。本节采用的容器运行时是 docker，具体安装步骤可以参考“ 部署 docker”小节。

1）准备 edge 端安装包。

因为证书问题，可以将 Kubernetes master 节点上的/etc/KubeEdge 直接复制到 edge 节点的/etc 下，命令如下：

#scp -r /etc/KubeEdge root@{ edge 节点 ip }:/etc

2）在 Kubernetes master 节点上创建 edge 节点的 node 资源对象，命令如下：

vi node.json

```
{
    "kind": "Node",
    "apiVersion": "v1",
    "metadata": {
      "name": "edge-node",
```

```
        "labels": {
        "name": "edge-node",
          "node-role.Kubernetes.io/edge": ""
        }
        }
    }
```

```
# kubectl create -f node.json
```

3）修改 edge 部分的配置.

修改两部分内容：edge 端连接 cloud 端的 ip；edge 端的 name 与在 Kubernetes master 上创建的 node 相对应。

- edge 端连接 cloud 端的 IP

edgehub.websocket.url：IP 修改成 Kubernetes master IP 端口不变。

edgehub.quic.url：IP 修改成 Kubernetes master IP 端口不变。

- edge 端的 name 与在 Kubernetes master 上创建的 node 相对应。

controller:node-id 与在 Kubernetes master 上创建的 node 的 name 保持一致。

edged:hostname-override 与在 Kubernetes master 上创建的 node 的 name 保持一致。

4）运行 edge 端，命令如下：

#cd /etc/KubeEdge/edge

#./EdgeCore

注意：本节为了方便查看进程输出采用了前台驻留进程的方式，除了上述方式外，还可以通过 systemd 来查看。

（3）验证 KubeEdge 是否正常运行。

KubeEdge 部署成功后，在 Kubernetes master 节点通过 kubectl 工具查看其运行状态，具体如图 1-15 所示。

```
[root@cloud ~]# kubectl get nodes
NAME        STATUS   ROLES    AGE   VERSION
edge-node   Ready    edge     75d   v1.10.9-kubeedge-v1.0.0
master01    Ready    master   75d   v1.16.2
```

图 1-15 集群节点运行状态

5.部署 EdgeX Foundry

EdgeX Foundry 是一套可以用 KubeEdge 部署到边缘上的 IoT SaaS 平台。它可以采集、存储 IoT 设备的数据并将其导出到云数据中心，同时通过向终端设备下发指令对终端设备进行控制。

(1)准备镜像

本节以容器的形式部署 EdgeX Foundry，需要在 KubeEdge 管理的边缘计算节点上准备 edgex-ui-go、edgex-vault、edgex-mongo、support-scheduler-go、support-notifications-go、support-logging-go、core-command-go、core-metadata-go、core-data-go、export-distro-go、export-client-go、edgex-vault-worker-go、edgex-vault 和 edgex-volume 共 14 个镜像。

有两种方法获取这些镜像：

1）直接在 dockerhub 上下载这些镜像；

2）根据 EdgeX Foundry 源码仓库中的 Makefile 文件构建这些镜像。

（2）准备部署 EdgeXFdoundry 组件所需的 yaml 文件

需要在前面部署的 Kubernetes 的 master 节点上准备与每个镜像对应的 yaml 文件对其进行部署。绝大多数镜像需要通过 deployment 进行部署,个别镜像需要 Job 进行部署，有些镜像还需要 service 对外暴露服务，这些 yaml 文件没有固定的标准。目前，EdgeX Foundry 官方还没有提供相关 yaml 文件，建议根据具体场景进行编写。

（3）通过 yaml 文件部署 EdgeX Foundry。

至此，我们已经拥有了 Kubernetes 的 master 节点，并将 master 节点作为云端控制节点，KubeEdge 管理的节点作为边缘计算节点的云、边协同的集群。同时，在 KubeEdge 管理的节点上准备好了部署 EdgeX Foundry 所需的镜像，在 Kubernetes 的 master 节点上准备好了运行 EdgeX Foundry 镜像所需的 yaml 文件。接下来，只需在 Kubernetes 的 master 节点上通过 kubectl 命令创建 yaml 文件中描述的资源对象即可，具体命令如下：

#kubectl create -f {文件名}.yaml

yaml 文件中描述的资源对象都创建好，意味着 EdgeX Foundry 的部署结束。至于，EdgeX Foundry 是否部署成功，可以通过如下命令进行验证：

#kubectl get pods - all-namespace

从图 1-16 可知，部署的 EdgeX Foundry 相关组件都已正常运行。

```
root@cloud:~# kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY   STATUS              RESTARTS   AGE
default       bluetooth-device-mapper-deployment-59c9d8c6-pdzhk   0/1     ContainerCreating   0          13h
default       edgex-config-seed-77f45747b5-rktjf            0/1     Completed           1079       3d23h
default       edgex-core-command-c8c8b877-pdzcc             1/1     Running             0          3d23h
default       edgex-core-consul-58f96f6dc6-vpgkp            1/1     Running             0          3d23h
default       edgex-core-data-7fcfc56696-s8pfz              1/1     Running             0          3d23h
default       edgex-core-metadata-db8648998-ch6lk           1/1     Running             0          3d23h
default       edgex-device-mqtt-64747cdd9-4sx84             1/1     Running             2          3d22h
default       edgex-export-client-775b9d96cc-bgb79          1/1     Running             0          3d23h
default       edgex-export-distro-5fc47f9cf5-8hpxq          1/1     Running             0          3d23h
default       edgex-files-75594c9b64-2hdth                  1/1     Running             0          3d23h
default       edgex-mongo-7bb7d86bc6-79g7c                  1/1     Running             0          3d23h
default       edgex-support-logging-5548fbdcc6-ndkjj        1/1     Running             0          3d23h
default       edgex-support-notifications-c5489d45c-w7968   1/1     Running             0          3d23h
default       edgex-support-rulesengine-66d7d977cd-86bvn    1/1     Running             0          3d23h
default       edgex-support-scheduler-c966f876b-vm9zc       1/1     Running             0          3d23h
default       edgex-ui-go-5d786d5565-52z7f                  1/1     Running             0          3d23h
```

图 1-16 EdgeXFoundry 组件运行状态

最后，通过在浏览器里访问 edgex-ui-go（即在浏览器访问 http://{EdgeX Foundry 所运行主机的 ip}:4000）进入 EdgeX Foundry 的登录页面，具体如图 1-17 所示。

图 1-17 EdgeX Foundry 的登录页面

在图 1-17 中输入对应的 Name/Password，就可以成功进入 EdgeX Foundry 的控制台，具体如图 1-18 所示。

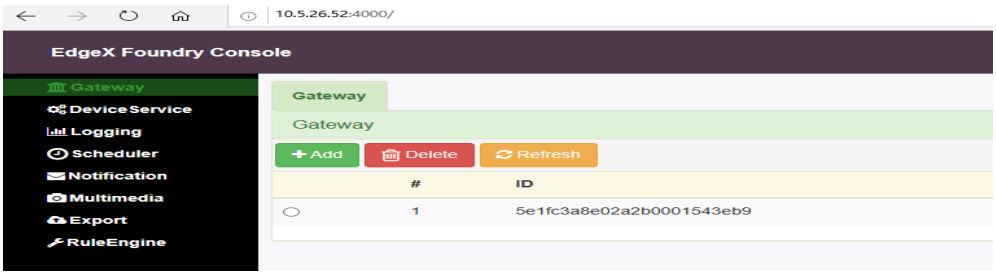

图 1-18 EdgeX Foundry 控制台

至此，我们已经拥有了由两个节点组成的，包含云、边、端的完整边缘计算系统。接下来介绍边缘计算系统的管理和在该边缘计算系统上部署应用。

1.3.2 系统管理

通过以上对云、边、端三部分的梳理，我们了解到边缘计算系统的管理可分为集群管理和应用管理。

1.集群管理

集群管理是对集群级别的资源进行管理，这些资源主要包括 node、namespace。下面通过对上述对象增、删、改、查进行说明。

（1）对 node 的操作

1）创建 node，命令如下：

kubectl create -f {node 定义文件}.ymal

2）删除 node，命令如下：

kubectl delete -f {node 定义文件}.ymal

#kubectl delete node {node 名字}

3）修改 node，命令如下：

#kubectl apply -f {修改过的 node 定义文件}.yaml

#kubectl edit node {node 名字}

4）查看 node，命令如下：

查看集群的 node 列表：#kubectl get nodes

查看指定 node 的具体定义：#kubectl describe node {node 名字}

（2）对 namespace 的操作

1）创建 Namespace，命令如下：

kubectl create -f {namespace 定义文件}.ymal

kubectl create namespace {namespace 名字}

2）删除 Namespace，命令如下：

kubectl delete -f {namespace 定义文件}.ymal

#kubectl delete namespace {namespace 名字}

3）修改 Namespace，命令如下：

#kubectl apply -f {修改过的 namespace 定义文件}.yaml

#kubectl edit namespace {namespace 名字}

4）查看 Namespace。

查看集群的 namespace 列表，命令如下：

#kubectl get namespace

查看指定 namespace 的具体定义，命令如下：

#kubectl describe namespace {namespace 名字}

集群级别的资源一般不需要用户对其进行创建、修改或者删除，只是在用户需要时对其进行查看。

2.应用管理

应用管理主要是对应用相关的资源进行管理，这些资源包括 deployment、replicaset、pod、service endpoint、service acount、secret、persistent volume、persistent volume claim。对这些应用相关资源的操作，与集群相关资源的操作比较相似，我们可以参考集群管理对指定资源进行增、删、改、查的操作。

需要说明一点，应用相关的资源一般需要用户创建和管理，也就是说掌握对应用相关的资源的增、删、改、查是有必要的。

1.4 不同应用部署方式的比较

本节主要对在云、边协同的集群上以及在传统的云平台上应用部署的架构、适用场景进行说明。

1. 在边缘计算系统上部署应用

图 1-19 是在云、边协同集群上部署应用的架构图。由架构图可知，云控制中心作为集群的控制平面，边缘计算节点作为应用负载最终运行的节点，运行在边缘计算节点上的应用与终端设备进行交互，负责终端设备的数据采集、存储和处理，并通过下发指令控制终端设备。

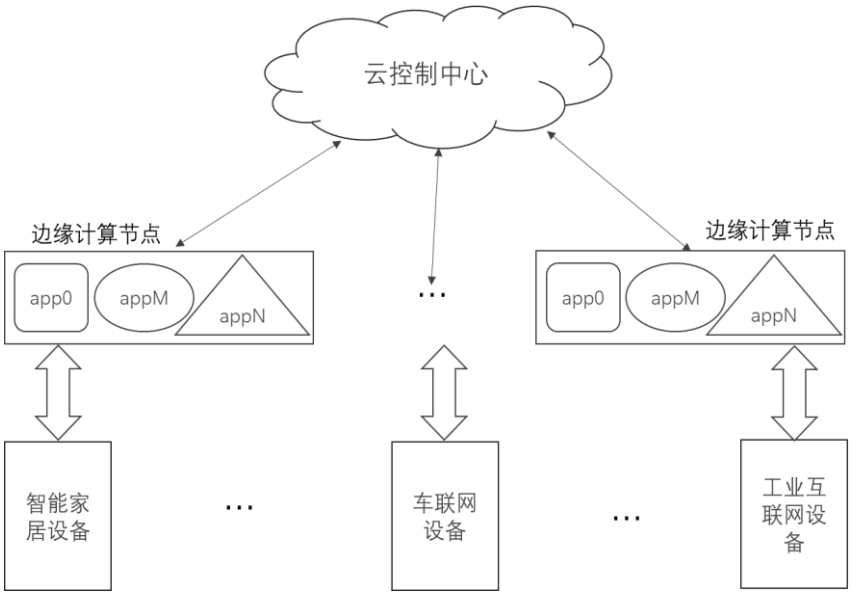

图 1-20 在边缘计算系统上部署应用

2. 在云上部署应用

图 1-21 是在传统云平台上部署应用的架构图。由架构图可知，云既作为控制平面，也最终承载应用负载。运行在云上的应用负载类型是多种多样的，既有面向互联网企业互联网企业的 Web 服务，也有面向智能家居、工业互联网、车联网等行业的 IoT SaaS 平台。

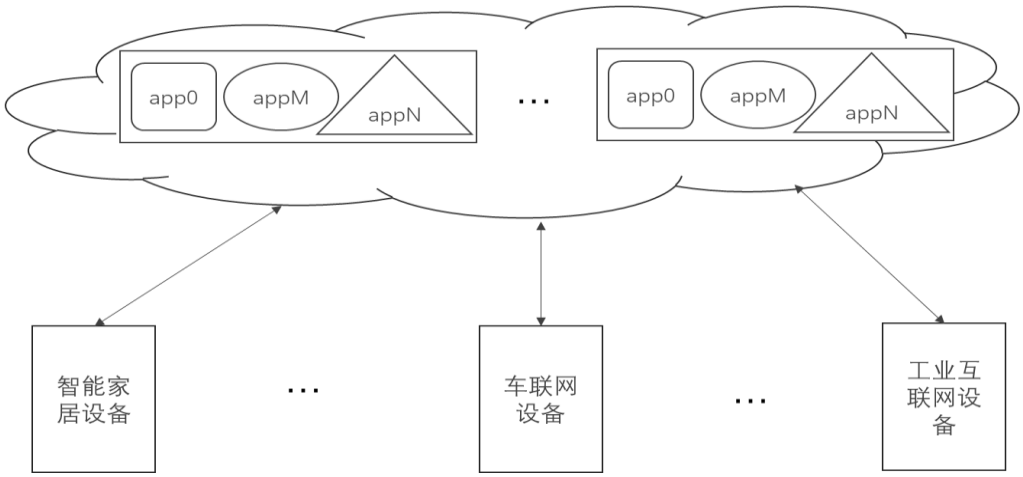

图 1-21 在云上部署应用架构图

3. 不同部署方式的适用场景

通过对比在边缘计算系统上部署应用和在云上部署应用的架构可知：

1）在边缘计算系统上部署应用的架构适合实时性要求较高、比较重视隐私、与云连接的网络质量没有保障、网络带宽受限的场景，比如 5G、无人驾驶、车联网、智能家居、工业互联网、医疗互联网、AR/VR 等。

2）在云上部署应用的架构适合实时性要求不高、计算密集型、I/O 密集型的场景，比如面向互联网的各种各样的 Web 服务、AI 模型训练、离线大数据处理等。

1.5 本章小结

本章对边缘计算系统的组成、边缘计算的意义、边缘计算系统的部署与管理、不同应用部署方式的比较进行了介绍。下一章将从整体架构切入介绍从云、边、端的部署与配置。

第 2 章 云、边、端的部署与配置

本章将从云、边、端协同的边缘计算系统的整体架构切入，罗列云、边、端各部分包含的组件的技术栈，然后分别对云、边、端各部分的部署方式和注意事项进行系统梳理和详细说明。

2.1 边缘计算整体架构

本节将对云、边、端协同的边缘计算系统的整体架构进行梳理和分析。边缘计算系统整体分为云、边、端三部分，具体如图 2-1 所示。

1）云： CPU 支持 X86 和 ARM 架构；操作系统支持 Linux、Windows 和 MacOS；容器运行时支持 docker、containerd 和 cri-o；集群编排使用 Kubernetes，包括控制节点、计算节点和集群存储。控制节点核心组件包括 kube-apiserver、kube-controller-manager 和 kube-scheduler，计算节点组件包括 Kubelet 和 kube-proxy，集群存储组件包括 etcd。云上的负载以 pod 形式运行，pod 是 container 组，container 是基于操作系统 namespace 和 cgroup 隔离出来的独立空间。

2）边：CPU 支持 X86 和 ARM 架构；操作系统支持 Linux；容器运行时支持 docker；边缘集群编排使用 KubeEdge，包括云部分的 CloudCore、边缘部分的 EdgeCore 和边缘集群存储 sqlite。边缘的负载以 pod 形式运行。

3）端：由运行在边缘集群上的管理端设备的服务框架 EdgeX Foundry 和端设备组成，EdgeX Foundry 从下向上依次为设备服务层、核心服务层、支持服务层、开放服务层，这也是物理域到信息域的数据处理顺序。设备服务层负责与南向设备交互；核心服务层介于北

向与南向之间作为消息管道和负责数据存储；支持服务层包含广泛的微服务，主要提供边缘分析服务和智能分析服务；开放服务层整个 EdgeX Foundry 服务框架的网关层。

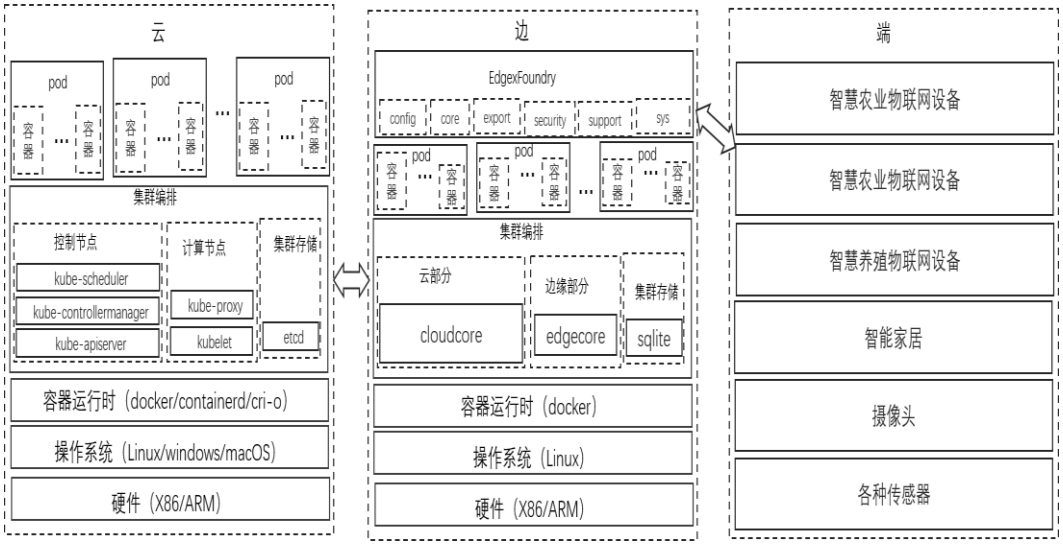

图 2-1 边缘计算整体架构

2.2 部署云部分——Kubernetes

Kubernetes 是一个全新的基于容器技术的分布式架构的云部署方案，是 Google 开源的容器集群管理系，为部署容器化的应用提供资源调度、服务发现和动态伸缩等一系列完整功能，提高了大规模容器集群管理的便捷性。本书将 Kubernetes 作为边缘计算系统的云部分解决方案。

本节会对 Kubernetes 的部署方式进行梳理，主要从 Kubernetes 相关的容器运行时部署、Kubernetes 的学习环境部署、Kubernetes 的生产环境部署三方面进行梳理。

2.2.1 Kubernetes 相关的容器运行时部署

Kubernetes 通过容器运行时在 pod 里运行容器，官方默认支持 docker 容器运行时。除此之外，Kubernetes 支持的容器运行时还包括 CRI-O、Containerd、Frakti 等，具体如表 2-1 所示。

表 2-1 Kubernetes 支持的容器运行时

容器运行时	原理说明	备注
docker	基于 Linux Namespace、Control Groups 和 Union filesystem 的一种容器运行时实现方式	同时支持 Linux 和 Windows 操作系统，目前是 Kubernetes 默认的容器运行时
Containerd	与 docker 相比减少了 dockerd 这一层富功能组件，其底层实现原理与 docker 相同	Containerd 的可配制性比较强，可以通过插件的方式替换具体实现
CRI-O	RedHat 发布的容器运行时，在同时满足 CRI 标准和 OCI 标准的前提下，将 CRI 和 OCI 的实现都进行了轻量化	还没有在大规模运行环境的验证，稳定性没有保障
Frakti	基于 Hypervisors 的容器	还没有在大规模运行环境的

	运行时，具有独立 kernel，比基于 Linux Namespace 的容器运行时的隔离性要好	验证，稳定性没有保障

从 Kubernetes 支持的容器运行时列表，我们可知：

1）docker 和 Containerd 在实现原理上是相同的，只是 Containerd 裁剪了 docker 原有的一些富功能；

2）CRI-O 为了追求轻量级和简洁，对 CRI 和 OCI 重新进行实现；

3）Frakti 的目的是实现容器运行时的强隔离性，基于 Hypervisors 实现容器运行时，使每个容器具有独立的 kernel。

目前，业界普遍使用的容器运行时是 docker。本节详细说明一下部署 docker 的相关步骤和注意事项。

1.部署 docker

本书环境使用的操作系统都是 CentOS7+，所以部署 docker 的步骤也是针对 CentOS7+操作环境。

1）安装需要的依赖包，命令如下：

```
# yum install yum-utils device-Mapper-persistent-data lvm2
```

2）增加安装 docker 所需的 repository，命令如下：

```
# yum-config-manager --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo
```

3）安装制定版本的 docker，命令如下：

yum update && yum install containerd.io-1.2.10 docker-ce-19.03.4 docker-ce-cli-19.03.4，命令如下：

4）安装制定版本的 docker，命令如下：

yum update && yum install containerd.io-1.2.10 docker-ce-19.03.4 docker-ce-cli-19.03.4

5）设置 docker 的配置文件。

创建配置文件目录：#mkdir /etc/docker

设置 docker 配置文件：# cat > /etc/docker/daemon.json <<EOF

```
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
```

}

EOF

6）启动 docker，命令如下：

mkdir -p /etc/systemd/system/docker.service.d

systemctl daemon-reload

systemctl restart docker

至此，docker 容器运行时就安装成功了。接下来，分析 docker 配置相关的注意事项。

7）docker 配置。

docker 的相关配置在/etc/docker/daemon.json 文件中进行设置，在 daemon.json 文件可以设置私有仓库、DNS 解析服务器、docker 运行时使用的路径、镜像加速地址、日志输出、Cgroup Driver、docker 主机的标签等。本节重点介绍 Cgroup Driver 的设置。

在 Linux 操作系统发行版本中使用 systemd 作为其初始化系统，初始化进程生成并使用一个 root 控制组件（cgroup），并将其当作 cgroup 管理器。systemd 与 cgroup 集成紧密，为每个进程分配 cgroup。docker 容器运行时默认的 cgroup 管理器是 cgroupfs，也就是 Kubelet 使用 cgroupfs 来管理 cgroup。这样就造成在同一台主机上同时使用 systemd 和 cgroupfs 两种 cgroup 管理器来对 cgroup 进行管理。

cgroup 用来约束分配给进程的资源。单个 cgroup 管理器能够简化分配资源的视图，并且默认情况下在管理可用资源和使用中的资源时使用一致的视图。当有两个 cgroup 管理器时，最终产生两种视图。我们已经看到某些案例中的节点配置让使 Kubelet 和 docker

使用 cgroupfs 管理器，而节点上运行的其余进程则使用 systemd，这类节点在资源压力下会变得不稳定。

更改设置，令容器运行时和 Kubelet 使用 systemd 作为 cgroup 驱动，以便系统更稳定。 请注意在/etc/docker/daemon.json 文件中设置 native.cgroupdriver=systemd 选项。具体如下：

```
# vi /etc/docker/daemon.json

{
   ...
   "exec-opts": ["native.cgroupdriver=systemd"],
   ...
}
```

2.2.2 Kubernetes 的学习环境部署

本节对部署 Kubernetes 学习环境的相关工具进行梳理，并对不同工具从安装方法、使用方法、相应原理的维度进行横向和纵向对比，如表 2-2 所示。

表 2-2 搭建 Kubernetes 学习环境的工具

部署工具	依赖	原理	备注
Minikube	操作系统对虚拟化的	创建一台虚拟机，	只适用于学习和

	支持、一款 Hypervisor KVM/VirtualBox、安装并配置了 kubectl	在虚拟机里运行 Kubernetes 集群	测试 Kubernete 的场景
Kind （Kubernetes in docker）	docker 容器运行时、在容器里运行 Kubernetes 集群的基础镜像 node image	创建一个 docker 容器，并在该容器里运行一个 Kubernetes 集群	只适用于学习和测试 Kubernete 的场景

从搭建 Kubernetes 学习环境的工具列表可知，Minikube、Kind（Kubernetes in docker）都可以搭建 Kubernetes 的学习环境，但两者所需要的依赖和原理各不相同。由于 Minikube 和 Kind（Kubernetes in docker）都是用于搭建 Kubernetes 学习环境的工具，所以两者的安装步骤和使用方法相对比较简单。接下来，笔者对两者的安装步骤和使用方法进行详细说明。

1.Minikube 的安装与使用

Minikube 是一种可以轻松在本地运行 Kubernetes 的工具。其首先通过在物理服务器或计算机上创建虚拟机，然后在虚拟机（VM）内运行一个单节点 Kubernetes 集群。该 Kubernetes 集群可以用于开发和测试 Kubernetes 的最新版本。

下面对 Minikube 的安装和使用进行说明。本书使用的操作系统都是 CentOS7+，所以本节安装 Minikube 的步骤也是针对 CentOS7+操作环境。

（1）安装 Minikube

1）检查对虚拟化的支持，命令如下：

```
# grep -E --color 'vmx|svm' /proc/cpuinf
```

2）安装 kubectl。

因为网络问题，使用操作系统原生包管理工具安装会遇到很多问题，所以推荐在 Kubernetes 的 GitHub 仓库上的 release 下载 pre-built 的 binary 进行安装。

进入 Kubernetes 的 GitHub 仓库上的 release 主页，找到需要下载的 Kubernetes 版本，比如本节要下载的版本是 v1.16.6，如图 2-2 所示。

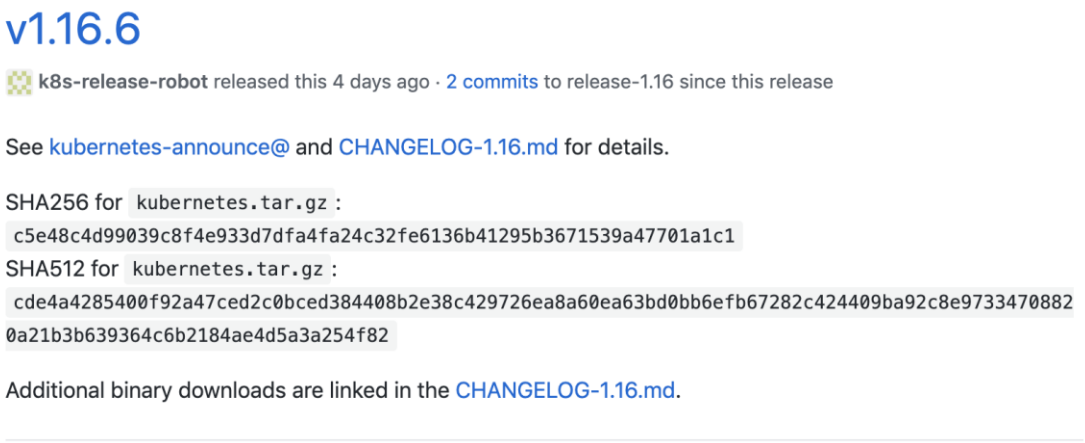

图 2-2 Kubernetes release 版本

点击 CHANGELOG-1.16.md 进入 binary 下载列表，复制 server binary 下载地址，使用 wget 下载 server binary 压缩包，命令如下：

```
# wget https://dl.k8s.io/v1.16.6/Kubernetes-server-linux-amd64.tar.gz
```

下载 Kubernetes 具体如图 2-3 所示。

图 2-3 下载 Kubernetes

如图 2-4 所示，解压 Kubernetes-server-linux-amd64.tar.gz，命令如下：

tar -zxvf Kubernetes-server-linux-amd64.tar.gz

```
[root@edge cuitest]# tar -zxvf kubernetes-server-linux-amd64.tar.gz
kubernetes/
kubernetes/server/
kubernetes/server/bin/
kubernetes/server/bin/apiextensions-apiserver
kubernetes/server/bin/kube-controller-manager.tar
kubernetes/server/bin/mounter
kubernetes/server/bin/kube-proxy.docker_tag
kubernetes/server/bin/kube-controller-manager.docker_tag
kubernetes/server/bin/kube-proxy.tar
kubernetes/server/bin/kubectl
kubernetes/server/bin/kube-scheduler.tar
kubernetes/server/bin/kube-apiserver.docker_tag
kubernetes/server/bin/kube-scheduler
kubernetes/server/bin/kubeadm
kubernetes/server/bin/kube-controller-manager
kubernetes/server/bin/kube-scheduler.docker_tag
kubernetes/server/bin/kubelet
kubernetes/server/bin/kube-proxy
kubernetes/server/bin/kube-apiserver.tar
kubernetes/server/bin/hyperkube
kubernetes/server/bin/kube-apiserver
kubernetes/LICENSES
kubernetes/kubernetes-src.tar.gz
kubernetes/addons/
```

图 2-4 解压 Kubernetes

由图 2-4 可知，kubectl 在 Kubernetes/server/bin 下，只需将其放入/usr/bin 下即可：

#cp Kubernetes/server/bin/kubectl /usr/bin

3）安装 KVM。

在确认所在的操作系统支持虚拟的前提下，通过如下步骤安装 KVM 及相关依赖。

更新安装 KVM 所需的源，命令如下：

#yum -y update && # yum install epel-release

安装 KVM 及其所需的依赖包，命令如下：

yum install qemu-kvm libvirt libvirt-python libguestfs-tools virt-install

设置 libvirtd 开机自启动并将其启动，命令如下：

systemctl enable libvirtd && systemctl start libvirtd

4）安装 Minikube。

curl -LO

https://storage.googleapis.com/minikube/releases/latest/minikube-1.6.2.rpm \

&& rpm -ivh minikube-1.6.2.rpm

（2）使用 Minikube

1）启动一个本地单节点集群，命令如下：

minikube start --vm-driver=<driver_name>

2）检查集群状态，命令如下：

minikube status

3）使用集群部署应用，命令如下：

kubectl create deployment hello-minikube --image={image-name}

至此，我们已经成功安装了 Minikube，并通过 Minikube 创建了一个本地单节点的 Kubernetes 集群。

2.Kind 的安装与使用

Kind（Kubernetes in docker）是一种使用 docker 容器节点（该容器可用于运行嵌套容器，在该容器里可以使用 systemd 运行、管理 Kubernetes 的组件）运行本地 Kubernetes 集群的工具。Kind 主要是为了测试 Kubernetes 本身而设计的，也可用于本地开发或持续集成。

下面对 Kind 的安装和使用进行说明。本书环境使用的操作系统都是 CentOS7+，所以本节安装 Kind 的步骤也是针对 CentOS7+操作环境。

（1）安装 Kind

由于安装 Kind 需要 Golang 语言环境，使用 Kind 运行本地 Kubernetes 集群需要 docker 容器运行时，因此在安装 Kind 之前需要先安装 Golang 和 docker。

1）安装 Golang 和 docker，命令如下：

```
# yum -y install Golang
```

参考“部署 docker”步骤来部署 docker 容器运行时。

2）安装 kind，命令如下：

```
#GO111MODULE="on" go get sigs.k8s.io/kind@v0.7.0
```

上述步骤会将 kind 安装到 GOPATH/bin 目录下。为了使用方便，建议将其在 /etc/profile 中进行追加设置，命令如下：

```
# vi /etc/profile
```

export PATH=$GOPATH/bin:$PATH

使在/etc/profile 中设置的环境变量立即生效，命令如下：

#source /etc/profile

（2）使用 Kind

1）使用 Kind 创建 Kubernetes 集群（如图 2-5 所示），命令如下：

kind create cluster。

```
[root@edge cuitest]# kind create cluster
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.17.0) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Not sure what to do next? 😅 Check out https://kind.sigs.k8s.io/docs/user/quick-start/
```

图 2-5 使用 Kind 创建集群

2）检查、使用 Kind 部署的集群(如图 2-6 所示),命令如下：

#kubectl get pods --all-namespaces

```
[root@edge cuitest]# kubectl get pods --all-namespaces
NAMESPACE            NAME                                          READY   STATUS    RESTARTS   AGE
kube-system          coredns-6955765f44-8l5sh                      1/1     Running   0          2m19s
kube-system          coredns-6955765f44-nmws4                      1/1     Running   0          2m19s
kube-system          etcd-kind-control-plane                       1/1     Running   0          2m32s
kube-system          kindnet-2m2gq                                 1/1     Running   0          2m18s
kube-system          kube-apiserver-kind-control-plane             1/1     Running   0          2m32s
kube-system          kube-controller-manager-kind-control-plane    1/1     Running   0          2m32s
kube-system          kube-proxy-q7l2k                              1/1     Running   0          2m18s
kube-system          kube-scheduler-kind-control-plane             1/1     Running   0          2m32s
local-path-storage   local-path-provisioner-7745554f7f-x28fl       1/1     Running   0          2m19s
```

图 2-6 检查使用 Kind 部署的集群状态

至此，我们已经成功安装了 Kind，并通过 Kind 创建了一个本地单节点的 Kubernetes 集群。

2.2.3 Kubernetes 的生产环境部署

本节对部署 Kubernetes 生产环境的相关工具进行梳理，并对不同工具从安装方法、使用方法、相应原理的维度进行横向和纵向对比，具体如表 2-3 所示。

表 2-3 搭建 Kubernetes 生产环境的工具

部署工具	依赖	原理	备注
kubeadm	docker 容器运行时、Kubelet	将安装 Kubernetes 过程的环境检查、下载镜像、生成证书、配置 Kubelet、准备 yaml 文件等步骤自动化	Kubeadm 所需的镜像默认以 k8s.gcr.io 开头，而且是硬编码的，国内无法直接下载，需要提前准备好
kops	Kubectl	在 kubeadm 的基础上，使搭建 Kubernetes 集群更方便	主要在 AWS 上进行自动化部署 Kubernetes 集群
KRIB	Golang	Digital Rebar Provision 专有的在裸机上搭建 Kubernetes 集群的工具	Digital Rebar Provision 还可以调用 Kubespray
Kubespray	ansible	用 ansible	

		playbooks、inventory 自动化部署 Kubernetes 集群	

从表 2-3 可知，kops、KRIB 有明显局限性，因为 kops 主要在 AWS 上进行自动化部署 Kubernetes 集群；KRIB 主要在裸机上进行自动化部署 Kubernetes 集群。kubeadm 和 Kubespray 可以在多种平台上搭建 Kubernetes 的生产环境，kubespray 从 v2.3 开始支持 kubeadm，也就意味着 kubespray 最终还是通过 kubeadm 进行自动化部署 Kubernetes 集群。

基于上述原因，本节将只对 kubeadm 进行自动化部署 Kubernetes 集群的步骤进行说明。

本节将对使用 kubeadm 的注意事项以及安装、使用 kubeadm 进行梳理和分析。kubeadm 支持的平台和资源要求如表 2-4 所示。

表 2-4 kubeadm 支持的平台和资源要求

kubeadm	支持的平台	Ubuntu 16.04+、Debian 9+ CentOS 7、Red Hat Enterprise Linux (RHEL) 7、Fedora 25+、HyprloTOS v1.0.1+、Container Linux (tested with 1800.6.0)

	资源要求	至少 2CPUs、2GB 内存
	注意事项	集群中主机之间的网络必须是相互可达的，在集群中每台主机的 hostname、MAC address 和 product_uuid 必须是唯一的，iptables 后端不用 nftable，在集群中每台主机上 Kubernetes 需要的一些端口必须是打开的，而且 swap 要关闭

接下来介绍对 kubeadm 的注意事项注意说明。

1）确保集群中所有主机之间网络可达。

在集群中不同主机间通过 ping 命令进行检测，命令如下：

ping {被检测主机 ip}

2）确保集群中所有主机的 hostname、MAC address 和 product_uuid 唯一。

查看主机 hostname 命令：#hostname

查看 MAC address 命令：#ip link 或者 #ifconfig -a

查看 product_uuid 命令：#/sys/class/dmi/id/product_uuid

3）iptables 后端不用 nftable，命令如下：

update-alternatives --set iptables /usr/sbin/iptables-legacy

4）Kubernetes 集群中主机需要打开的端口，如表 2-5 所示。

表 2-5 Kubernetes 集群中主机需要打开的端口

节点	协议	流量方向	端口	用途	使用组件
控制节点	TCP	进入集群	6443	Kubernetes-api-server	集群中所有组件都会使用这个端口
			2379、2380	etcd server client API	kube-api、etcd
			10250	Kubelet API	Kubelet、控制平面
			10251	kube-scheduler	kube-scheduler
			10252	kube-controller-manager	kube-controller-manager
计算节点	TCP	进入集群	10250	Kubelet API	Kubelet、控制平面
			30000-32767	NodePort Services	集群中所有组件都会使用这个端口

由表 2-5 可知，上述需要打开的端口都是 Kubernetes 默认需要打开的端口。我们也可以根据需要对一些端口进行单独指定，比如 Kubernetes-api-server 默认打开的端口是 6443，也可以指定打开其他与现有端口不冲突的端口。

5）在 Kubernetes 集群中所有节点上关闭 swap，命令如下：

#swapoff -a

（2）安装 kubeadm

安装 kubeadm 有两种方式，即通过操作系统的包管理工具进行安装和从 Kubernetes GitHub 仓库的 release 上下载 pre-build 的 binary 进行安装。下面对这两种安装方式进行详细说明。

1）通过操作系统的包管理工具安装 kubeadm。

在需要安装 kubeadm 的节点上设置安装 kubeadm 需要的仓库，命令如下：

```
#cat <<EOF > /etc/yum.repos.d/Kubernetes.repo
[Kubernetes]
name=Kubernetes
 baseurl=https://packages.cloud.google.com/yum/repos/Kubernetes-el7-x8
 6_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
```

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg

https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

将 SELinux 设置为 permissive，命令如下：

#setenforce 0

#sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

安装 kubeadm、Kubelet、kubectl，命令如下：

#yum install -y Kubelet kubeadm kubectl --disableexcludes=Kubernetes

将 Kubelet 设置为开机自启动，命令如下：

#systemctl enable --now Kubelet

注意：由于网络问题，通过操作系统包管理工具安装 kubeadm 很可能会失败。

2）通过在 Kubernetes GitHub 仓库的 release 上下载 pre-build 的 binary 安装 kubeadm。

进入 Kubernetes 的 GitHub 仓库上的 release 主页，找到需要下载的 Kubernetes 版本，比如要下载的版本是 v1.16.6，如图 2-7 所示。

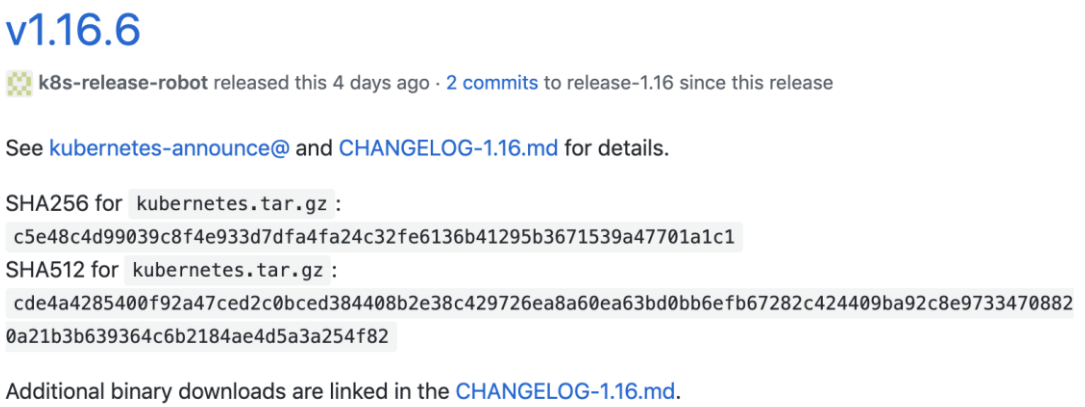

图 2-7 Kubernetes release 版本

点击 CHANGELOG-1.16.md 进入 binary 下载列表，复制 server binary 下载地址，使用 wget 下载 server binary 压缩包，命令如下：

```
# wget https://dl.k8s.io/v1.16.6/Kubernetes-server-linux-amd64.tar.gz
```

下载 Kubernetes binary 具体如图 2-8 所示。

图 2-8 下载 Kubernetes binary

解压 Kubernetes-server-linux-amd64.tar.gz，命令如下：

```
# tar -zxvf Kubernetes-server-linux-amd64.tar.gz
```

解压 Kubernetes binary,具体如图 2-9 所示。

```
[root@edge cuitest]# tar -zxvf kubernetes-server-linux-amd64.tar.gz
kubernetes/
kubernetes/server/
kubernetes/server/bin/
kubernetes/server/bin/apiextensions-apiserver
kubernetes/server/bin/kube-controller-manager.tar
kubernetes/server/bin/mounter
kubernetes/server/bin/kube-proxy.docker_tag
kubernetes/server/bin/kube-controller-manager.docker_tag
kubernetes/server/bin/kube-proxy.tar
kubernetes/server/bin/kubectl
kubernetes/server/bin/kube-scheduler.tar
kubernetes/server/bin/kube-apiserver.docker_tag
kubernetes/server/bin/kube-scheduler
kubernetes/server/bin/kubeadm
kubernetes/server/bin/kube-controller-manager
kubernetes/server/bin/kube-scheduler.docker_tag
kubernetes/server/bin/kubelet
kubernetes/server/bin/kube-proxy
kubernetes/server/bin/kube-apiserver.tar
kubernetes/server/bin/hyperkube
kubernetes/server/bin/kube-apiserver
kubernetes/LICENSES
kubernetes/kubernetes-src.tar.gz
kubernetes/addons/
```

图 2-9 解压 Kubernetes binary

由图 2-9 可知，kubeadm 在 Kubernetes/server/bin 只需将其放入/usr/bin 下即可，命令如下：#cp Kubernetes/server/bin/kubeadm /usr/bin

（3）使用 kubeadm

使用 kubeadm 可以部署 Kubernetes 单节点集群、Kubernetes 单控制节点集群和 Kubernetes 高可用集群。下面将详细说明部署 3 种集群的具体步骤和注意事项。

1）部署 Kubernetes 单节点集群。

使用 kubeadm 部署 Kubernetes 单节点集群，其实是在一个节点使用 kubeadm 部署 Kubernetes 的控制平面，然后对该节点进行设置，使其能够运行应用负载。

查看使用 kubeadm 部署 Kubernetes 集群时所需的镜像，命令如下：

#kubeadm config images list

所需镜像如图 2-10 所示。

```
[root@edge ~]# kubeadm config images list
I0126 15:23:09.585888   25816 version.go:251] remote version
k8s.gcr.io/kube-apiserver:v1.16.6
k8s.gcr.io/kube-controller-manager:v1.16.6
k8s.gcr.io/kube-scheduler:v1.16.6
k8s.gcr.io/kube-proxy:v1.16.6
k8s.gcr.io/pause:3.1
k8s.gcr.io/etcd:3.3.15-0
k8s.gcr.io/coredns:1.6.2
```

图 2-10 kubeadm 所需镜像

由于这些镜像都是以 k8s.gcr.io*开头，一般情况下，kubeadm 无法正常下载这些镜像，需要提前准备好。获取这些镜像的方法不止一种，笔者建议通过 dockerhub 获得。

使用 kubeadm 创建 Kubernetes 集群，在创建集群的过程中会用到图 2-10 中列出的所有镜像，命令为#kubeadm init {args}。

在 args 中一般只需指定--control-plane-endpoint、--pod-network-cidr、--cri-socket、--apiserver-advertise-address 参数。这些参数的具体作用如下。

1）--control-plane-endpoint：在搭建高可用 Kubernetes 集群时，为多个控制平面共用的域名或负载均衡 IP；

2）--pod-network-cidr Kubernetes：集群中 pod 所用的 IP 池；

3）--cri-socket ： 指定 Kubernetes 集群使用的容器运行时；

4）--apiserver-advertise-address ： 指定 kube-api-server 绑定的 IP 地址，既可以是 IPv4 也可以是 IPv6 。

我们可以根据具体情况具体指定以上参数。

根据 non-root 用户和 root 用户，设置 kubectl 使用的配置文件。

non-root 用户设置命令入如下：

```
$mkdir -p $HOME/.kube
$sudo cp -i /etc/Kubernetes/admin.conf $HOME/.kube/config
$sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

root 用户设置命令如下：

```
export KUBECONFIG=/etc/Kubernetes/admin.conf
```

为了方便，也可以将 KUBECONFIG 设置成自动生效的系统环境变量，命令如下：

```
# vim /etc/profile

export KUBECONFIG=/etc/Kubernetes/admin.conf
```

安装 pod network 所需的网络插件，命令如下：

```
# kubectl apply -f https://docs.projectCalico.org/v3.8/manifests/Calico.yaml
```

本节使用的网络插件是 Calico，我们也可以根据具体需求选择其他的网络插件，比如 flannel、Weave Net、kube-router 等。

至此，一个完整的 Kubernetes 控制节点就搭建完成了，但还不能算一个完整单节点集群，因为该控制节点默认不接受负载调度。要使其能够接受负载调度，需要进行如下设置：

kubectl taint nodes --all node-role.Kubernetes.io/master-

2）部署 Kubernetes 单控制节点集群。

Kubernetes 的单控制节点集群，是指该 Kubernetes 集群只有一个控制节点，但可以有不止一个的计算节点。部署该集群只需在部署 Kubernetes 单节点集群中安装 pod network 所需的网络插件之后，计算节点加入该控制节点即可，具体命令如下：

ubeadm join <control-plane-host>:<control-plane-port> --token <token> --discovery-token-ca-cert-hash sha256:<hash>

使用 kubeadm join 可以将多个计算节点加入到已经部署成功的控制节点，与控制节点组成一个单控制节点的 Kubernetes 集群。

3）部署 Kubernetes 高可用集群。

kubeadm 除了可以部署单节点 Kubernetes 集群和单控制节点 Kubernetes 集群外，还可以部署高可用 Kubernetes 集群。Kubeadm 部署 Kubernetes 高可用集群的架构包含两种，即 ectd 集群与 Kubernetes 控制节点集群一起部署的 Kubernetes 高可用集群，

以及 ectd 集群与 Kubernetes 控制节点集群分开部署的 Kubernetes 高可用集群，具体架构如图 2-11 和图 2-12 所示。

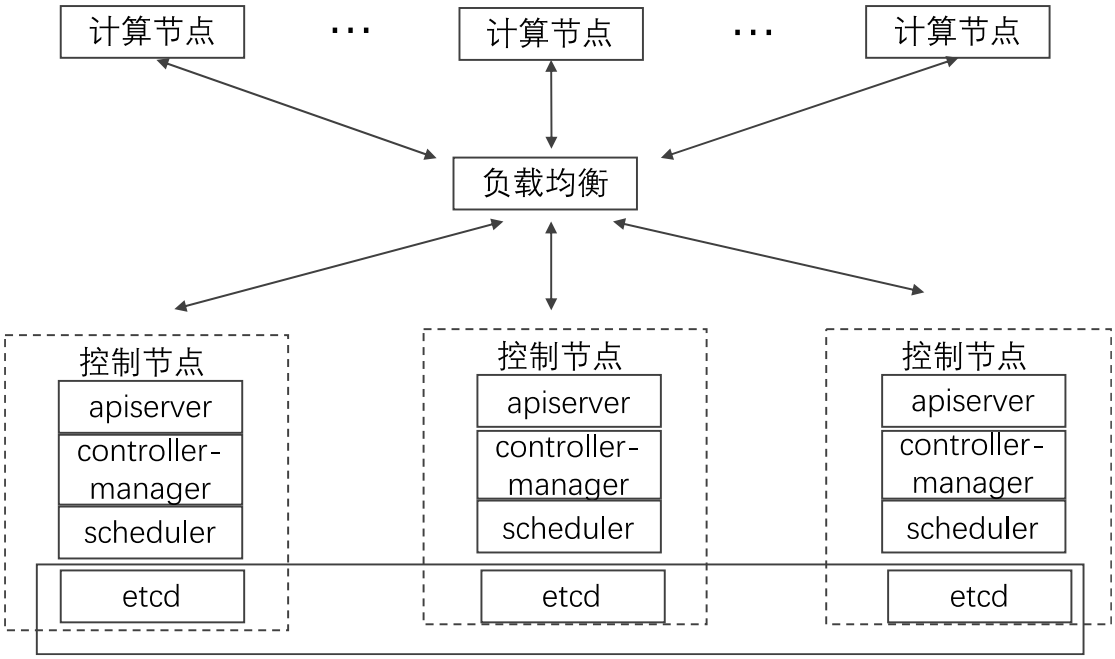

图 2-11 ectd 集群与 Kubernetes 控制节点集群一起部署

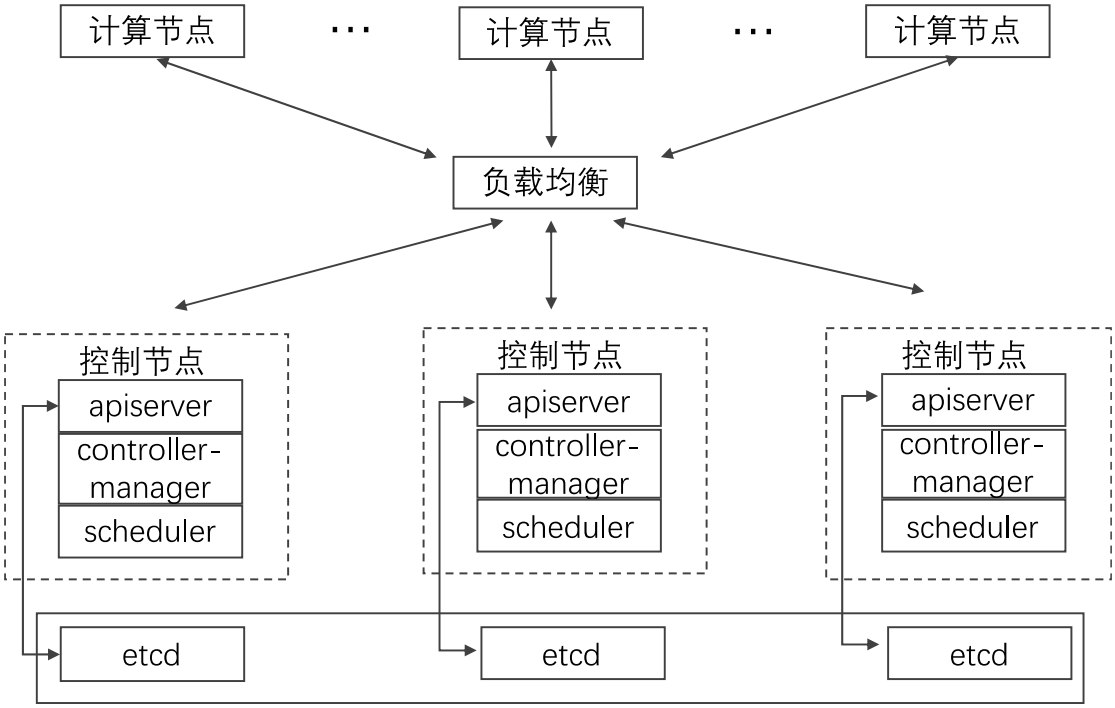

图 2-12 ectd 集群与 Kubernetes 控制节点集群分开部署

由图 2-11 和图 2-12 可知，Kubernetes 集群高可用即 Kubernetes 集群中 master 节点和 etcd 集群的高可用。etcd 集群又有两种高可用部署方式，即 ectd 集群与 Kubernetes 控制节点集群一起部署和 ectd 集群与 Kubernetes 控制节点集群分开部署。

部署 Kubernetes 高可用集群是面向生产环境的，需要的资源比较多，部署步骤也相对比较复杂，限于篇幅本书就不展开说明了，感兴趣的读者可以参考 Kubernetes 官网进行实践。

2.3 部署边缘部分——KubeEdge

KubeEdge 是一个基于 Kubernetes 构建的开放平台，能够将 Kubernetes 拥有的编排容器化应用的能力扩展到边缘的节点和设备，并为云和边缘之间的网络、应用部署和元数据同步提供基础架构支持。本书将 KubeEdge 作为边缘计算系统边部分的解决方案。

本节会对 KubeEdge 的部署方式进行梳理。KubeEdge 可以通过系统进程的方式、容器化的方式和使用工具进行部署。

2.3.1 以系统进程的方式部署 KubeEdge

以系统进程的方式部署 KubeEdge，即以系统进程的方式部署 KubeEdge 的云组件和边缘组件。下面将对部署过程中需要的依赖、配置等进行详细说明。

（1） 安装 KubeEdge 的云组件

获取 KubeEdge 云组件的方法有两种，即通过编译 KubeEdge 的云组件源码和从 KubeEdge GitHub 仓库的 release 下载。本节只说明通过编译 KubeEdge 的云组件源码获得 KubeEdge 云组件可执行文件的方式。

1）编译 KubeEdge 的云组件源码。

下载 KubeEdge 源码命令如下：

```
#git clone https://GitHub.com/KubeEdge/KubeEdge.git KubeEdge
```

在编译之前确保 gcc 已经安装，命令如下：

```
#gcc --version
```

通过编译源码，获得 KubeEdge 云组件，命令如下：

```
#cd KubeEdge
#make all WHAT=CloudCore
```

编译成功之后，会在./cloud 下生成可执行文件 CloudCore，将其复制到/usr/bin 下即可。

2）创建 device model 和 device CRDs，命令如下：

#cd ../KubeEdge/build/crds/devices

#kubectl create -f devices_v1alpha1_devicemodel.yaml

#kubectl create -f devices_v1alpha1_device.yaml

3）生成 Certificates，命令如下：

#cd KubeEdge/build/tools

#./certgen.sh genCertAndKey edge

执行上述命令后，会在/etc/KubeEdge/ca 下生成 rootCA.crt，在 etc/KubeEdge/certs 下生成 edge.crt 、edge.key。生成的这些证书在 KubeEdge 的云组件和边缘组件中共用。

4）生成和设置 KubeEdge 云组件的配置文件。

使用 CloudCore 可以生成最小化配置文件和默认配置文件。

创建配置文件目录命令如下：

#mkdir -p /etc/KubeEdge/config/

生成最小化配置文件命令如下：

#CloudCore --minconfig > /etc/KubeEdge/config/CloudCore.yaml

生成默认配置文件命令如下：

CloudCore --defaultconfig > /etc/KubeEdge/config/CloudCore.yaml

执行上述命令后，会在/etc/KubeEdge/config 下生成 CloudCore.yaml。下面对执行 CloudCore 生成的默认配置文件 CloudCore.yaml 进行说明，具体如下所示。

```
apiVersion: CloudCore.config.KubeEdge.io/v1alpha1
kind: CloudCore
kubeAPIConfig:
  kubeConfig: /root/.kube/config # kubeconfig 文件的绝对路径
  master: "" # kube-apiserver address (比如:http://localhost:8080)
modules:
  cloudhub:
    nodeLimit: 10
    tlsCAFile: /etc/KubeEdge/ca/rootCA.crt
    tlsCertFile: /etc/KubeEdge/certs/edge.crt
    tlsPrivateKeyFile: /etc/KubeEdge/certs/edge.key
    unixsocket:
      address: unix:///var/lib/KubeEdge/KubeEdge.sock # unix domain socket
address
      enable: true # enable unix domain socket protocol
    websocket:
      address: 0.0.0.0
```

```
    enable: true # enable websocket protocol
    port: 10000 # open port for websocket server
```

5）运行 KubeEdge 云组件，命令如下：

#nohup ./CloudCore &

除了上述形式，还可以通过 systemd 以后台进程的形式运行 KubeEdge 云组件，命令如下：

#ln KubeEdge/build/tools/CloudCore.service

/etc/systemd/system/CloudCore.service

systemctl daemon-reload

systemctl start CloudCore

将 KubeEdge 云组件设置为开机自启动，命令如下：

#systemctl enable CloudCore

（2）安装 KubeEdge 的边缘组件

1）编译 KubeEdge 的边缘组件源码。

下载 KubeEdge 源码，命令如下：

#git clone https://GitHub.com/KubeEdge/KubeEdge.git KubeEdge

在编译之前确保 gcc 已经安装，命令如下：

#gcc --version

通过编译源码获得 KubeEdge 的边缘组件，命令如下：

#cd KubeEdge

#make all WHAT=EdgeCore

编译成功之后，会在./edge 下生成可执行文件 EdgeCore，将其复制到/usr/bin 下即可。

2）从 KubeEdge 的云组件节点复制 Certificates，命令如下：

#scp -r /etc/KubeEdge root@{KubeEdge edge 节点 IP}:/etc/KubeEdge

3）在 KubeEdge 的云组件节点为边缘节点创建 node 对象资源，命令如下：

#kubectl create -f KubeEdge/build/node.json

node.json 具体内容如下所示。

```
{
  "kind": "Node",
  "apiVersion": "v1",
  "metadata": {
    "name": "edge-node",
    "labels": {
      "name": "edge-node",
      "node-role.Kubernetes.io/edge": ""
    }
```

```
  }
}
```

4)生成和设置 KubeEdge 边缘组件的配置文件。

使用 EdgeCore 可以生成最小化配置文件和默认配置文件。

创建配置文件目录，命令如下：

#mkdir -p /etc/KubeEdge/config/

生成最小化配置文件，命令如下：

#EdgeCore - minconfig > /etc/KubeEdge/config/EdgeCore.yaml

生成默认配置文件，命令如下：

EdgeCore - defaultconfig > /etc/KubeEdge/config/EdgeCore.yaml

执行上述命令后，会在/etc/KubeEdge/config 下生成 EdgeCore.yaml 文件。下面对执行 EdgeCore 生成的默认配置文件 EdgeCore.yaml 进行说明，具体如下所示。

```
apiVersion: EdgeCore.config.KubeEdge.io/v1alpha1
database:
  dataSource: /var/lib/KubeEdge/EdgeCore.db
kind: EdgeCore
modules:
  edged:
    cgroupDriver: cgroupfs
    clusterDNS: ""
```

```
    clusterDomain: ""
    devicePluginEnabled: false
    dockerAddress: unix:///var/run/docker.sock
    gpuPluginEnabled: false
    hostnameOverride: $your_hostname
    interfaceName: eth0
    nodeIP: $your_ip_address
    podSandboxImage: KubeEdge/pause:3.1    # KubeEdge/pause:3.1 for x86
arch , KubeEdge/pause-arm:3.1 for arm arch, KubeEdge/pause-arm64 for
arm64 arch
    remoteImageEndpoint: unix:///var/run/dockershim.sock
    remoteRuntimeEndpoint: unix:///var/run/dockershim.sock
    runtimeType: docker
  edgehub:
    heartbeat: 15    # second
    tlsCaFile: /etc/KubeEdge/ca/rootCA.crt
    tlsCertFile: /etc/KubeEdge/certs/edge.crt
    tlsPrivateKeyFile: /etc/KubeEdge/certs/edge.key
    websocket:
      enable: true
      handshakeTimeout: 30    # second
      readDeadline: 15    # second
```

```
        server: 127.0.0.1:10000    # CloudCore address
        writeDeadline: 15    # second
    eventbus:
      mqttMode: 2    # 0: internal mqtt broker enable only. 1: internal and external mqtt broker enable. 2: external mqtt broker
      mqttQOS: 0    # 0: QOSAtMostOnce, 1: QOSAtLeastOnce, 2: QOSExactlyOnce.
      mqttRetain: false    # if the flag set true, server will store the message and can be delivered to future subscribers.
      mqttServerExternal: tcp://127.0.0.1:1883    # external mqtt broker url.
      mqttServerInternal: tcp://127.0.0.1:1884    # internal mqtt broker url.
```

其中，Modules.edged.hostnameOverride 与 node.json 里的 metadata.name 保持一致；Modules.edged.nodeIP 是 KubeEdge 边缘节点的 IP；Modules.edgehub.websocket.server 是 KubeEdge 云节点的 IP。

5)运行 KubeEdge 云组件，命令如下：

```
#nohup ./EdgeCore &
```

除了上述形式，我们还可以通过 systemd 以后台进程的形式运行 KubeEdge 云组件，命令如下：

#ln KubeEdge/build/tools/EdgeCore.service

/etc/systemd/system/EdgeCore.service

systemctl daemon-reload

systemctl start EdgeCore

将 KubeEdge 云组件设置为开机自启动，命令如下：

#systemctl enable EdgeCore

至此，以系统进程的方式部署 KubeEdge 的云组件和边缘组件都已经完成了，接下来检查 KubeEdge 的状态，并基于 KubeEdge 部署应用。

（3）检查 KubeEdge 节点状态

在 KubeEdge 云节点执行如下命令，检查 KubeEdge 边缘节点的状态，命令如下：

#kubectl get nodes

（4）基于 KubeEdge 部署应用

基于 KubeEdge 部署应用的命令如下：

#kubectl apply -f KubeEdge/build/deployment.yaml

2.3.2 以容器化的方式部署 KubeEdge

本节以容器的方式部署 KubeEdge，即以容器的方式部署 KubeEdge 的云组件和边缘组件。下面将对部署过程的步骤和相关配置等进行详细说明。

（1）以容器的方式部署 KubeEdge 的云组件

1)下载部署 KubeEdge 的云组件所需的资源文件，命令如下：

#git clone https://GitHub.com/KubeEdge/KubeEdge.git KubeEdge

2)构建部署 KubeEdge 的云组件所需的镜像，命令如下：

#cd KubeEdge

make cloudimage

3)生成部署 KubeEdge 的云组件所需的 06-secret.yaml，命令如下：

#cd build/cloud

#../tools/certgen.sh buildSecret | tee ./06-secret.yaml

4)以容器的方式部署 KubeEdge 的云组件，命令如下：

#for resource in $(ls *.yaml); do kubectl create -f $resource; done

（2）以容器的方式部署 KubeEdge 的边缘组件

1）下载部署 KubeEdge 的边缘组件所需的资源文件，命令如下：

#git clone https://GitHub.com/KubeEdge/KubeEdge.git KubeEdge

2）检查 container runtime 环境，命令如下：

```
# cd ./KubeEdge/build/edge/run_daemon.sh prepare
```

3）设置容器参数，命令如下：

```
# ./KubeEdge/build/edge /run_daemon.sh set \
      cloudhub=0.0.0.0:10000 \
      edgename=edge-node \
      EdgeCore_image="KubeEdge/EdgeCore:latest" \
      arch=amd64 \
      qemu_arch=x86_64 \
      certpath=/etc/KubeEdge/certs \
      certfile=/etc/KubeEdge/certs/edge.crt \
      keyfile=/etc/KubeEdge/certs/edge.key
```

4）构建部署 KubeEdge 的边缘组件所需的镜像，命令如下：

```
#./KubeEdge/build/edge /run_daemon.sh build
```

5）启动 KubeEdge 的边缘组件容器，命令如下：

```
#./KubeEdge/build/edge /run_daemon.sh up
```

至此，以容器的方式部署 KubeEdge 的云组件和边缘组件都已经完成了。关于 KubeEdge 的状态查看以及基于 KubeEdge 部署应用部分，读者可以参考“以系统进程的方式部署 KubeEdge 部分”。

2.4 部署端部分——EdgeX Foundry

EdgeX Foundry 是一个由 Linux Foundation 托管的、供应商中立的开源项目，用于为 IoT 边缘计算系统构建通用的开放框架。该项目的核心是一个互操作性框架。该框架可以托管在与硬件和操作系统无关的平台上，以实现即插即用组件的生态系统，从而加速 IoT 解决方案的部署。本节将对 EdgeX Foundry 的部署方式进行系统梳理，并对部署方式中的相关注意事项进行详细说明，具体如表 2-9 所示。

表 2-9 KubeEdge 的部署方式和注意事项

部署方式	部署平台	部署原理	备注
容器化部署	docker-compose	将 EdgeX Foundry 的各组件容器化，并通过 docker-compose 对其进行部署、编排	目前，该部署方式是官方提供的部署方式之一
	Kubernetes	将 EdgeX Foundry 的各组件容器化，并通过 Kubernetes 对其	目前，官方不提供该部署方式的相关说明

		进行部署、编排	
	KubeEdge	将 EdgeX Foundry 的各组件容器化，并通过 KubeEdge 对其进行部署、编排	目前，官方不提供该部署方式的相关说明
系统进程部署	支持 EdgeX Foundry 运行的各种操作系统	将 EdgeX Foundry 的各组件以系统进程的方式进行部署	目前，该部署方式是官方提供的部署方式之一

需要说明的是，在本书云、边、端协同的边缘计算系统中，作为端解决方案的 EdgeX Foundry 是通过 KubeEdge 对其进行容器化部署的。但是，目前官方没有提供通过 KubeEdge 对其进行容器化部署的相关说明，所以笔者根据本书的部署环境针对 KubeEdge 开发了一套 yaml 文件。

1）该 yaml 文件托管在 GitHub 上（https://GitHub.com/WormOn/edgecomputing/tree/master/end），可作为读者学习参考资料。

2）通过 Kubernetes 对 EdgeX Foundry 进行容器化部署，与通过 KubeEdge 对其进行容器化部署原理相同，读者也可以参考该 yaml 文件完成通过 Kubernetes 对 EdgeX Foundry 的部署。

2.4.1 系统进程部署 EdgeX Foundry

以系统进程部署 EdgeX Foundry，即将 EdgeX Foundry 的各组件以系统进程的方式进行部署。本节对该方式进行展开说明。

（1） 获取 EdgeX Foundry 源码，命令如下：

#git clone https://GitHub.com/EdgeX Foundry/edgex-go.git

（2） 基于源码构建 EdgeX Foundry 各组件的 binary。

进入 edgex-go 源码根目录，命令如下：

#cd edgex-go

源码编译 edgex-go，命令如下：

#make build

构建 EdgeX Foundry 各组件的 binary，具体如图 2-13 所示。

```
[root@edge edgex-go]# make build
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/config-seed/config-seed ./cmd/config-seed
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/core-metadata/core-metadata ./cmd/core-metadata
CGO_ENABLED=1 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/core-data/core-data ./cmd/core-data
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/core-command/core-command ./cmd/core-command
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/support-logging/support-logging ./cmd/support-logging
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/support-notifications/support-notifications ./cmd/support-notifications
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/sys-mgmt-executor/sys-mgmt-executor ./cmd/sys-mgmt-executor
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/sys-mgmt-agent/sys-mgmt-agent ./cmd/sys-mgmt-agent
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/support-scheduler/support-scheduler ./cmd/support-scheduler
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-secrets-setup/security-secrets-setup ./cmd/security-secrets-setup
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-proxy-setup/security-proxy-setup ./cmd/security-proxy-setup
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-secretstore-setup/security-secretstore-setup ./cmd/security-secretstore-setup
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-file-token-provider/security-file-token-provider ./cmd/security-file-token-provider
```

图 2-13 构建 EdgeX Foundry 各组件的 binary

由图 2-13 可知，会在./cmd 下各组件子目录里生成相应的可执行文件，比如 config-seed 的可执行文件会在./cmd/config-seed 目录下，具体如图 2-14 所示。

```
[root@edge edgex-go]# ls cmd/
config-seed   core-data      security-file-token-provider  security-secrets-setup      support-logging       support-scheduler  sys-mgmt-executor
core-command  core-metadata  security-proxy-setup          security-secretstore-setup  support-notifications  sys-mgmt-agent
[root@edge edgex-go]# ls cmd/config-seed/
Attribution.txt  config-seed  Dockerfile  main.go  res
```

图 2-14 源码编译 edgex-go 生成的可执行文件

（3）运行 EdgeX Foundry 的各组件

通过 make 一键运行 edgex，命令如下：

#make run

edgex-go 一键启动命令和输出结果如图 2-15 所示。

图 2-15 edgex-go 一键启动

由 2-15 可知，make run 是通过执行#cd bin && ./edgex-launch.sh 将 EdgeX Foundry 的各组件以系统进程的方式运行起来的。下面看一下 edgex-launch.sh 的具体内容。

打开 edgex-launch.sh：#vim edgex-go/bin/edgex-launch.sh，具体如下所示。

```
#!/bin/bash
#
```

```
# Copyright (c) 2018
# Mainflux
#
# SPDX-License-Identifier: Apache-2.0
#


###
# Launches all EdgeX Go binaries (must be previously built).
#
# Expects that Consul and MongoDB are already installed and running.
#
###

DIR=$PWD
CMD=../cmd

# Kill all edgex-* stuff
function cleanup {
  pkill edgex
}

# disable secret-store integration
```

```
export EDGEX_SECURITY_SECRET_STORE=false


###
# Support logging
###
cd $CMD/support-logging
# Add `edgex-` prefix on start, so we can find the process family
exec -a edgex-support-logging ./support-logging &
cd $DIR


###
# Core Command
###
cd $CMD/core-command
# Add `edgex-` prefix on start, so we can find the process family
exec -a edgex-core-command ./core-command &
cd $DIR


###
# Core Data
###
cd $CMD/core-data
```

```
exec -a edgex-core-data ./core-data &
cd $DIR

###
# Core Meta Data
###
cd $CMD/core-metadata
exec -a edgex-core-metadata ./core-metadata &
cd $DIR

###
# Support Notifications
###
cd $CMD/support-notifications
# Add `edgex-` prefix on start, so we can find the process family
exec -a edgex-support-notifications ./support-notifications &
cd $DIR

###
# System Management Agent
###
cd $CMD/sys-mgmt-agent
```

```
# Add `edgex-` prefix on start, so we can find the process family
exec -a edgex-sys-mgmt-agent ./sys-mgmt-agent &
cd $DIR

# Support Scheduler
###
cd $CMD/support-scheduler
# Add `edgex-` prefix on start, so we can find the process family
exec -a edgex-support-scheduler ./support-scheduler &
cd $DIR

trap cleanup EXIT

while : ; do sleep 1 ; done
```

edgex-launch.sh 主要做了 3 件事。

1）通过 shell 的内置命令 exec 将 EdgeX Foundry 的各组件以系统进程的方式运行起来；

2）通过一个 while 死循环将 edgex-launch.sh 以前台驻留进程的方式驻留在前台；

3）通过 trap 命令监听 EXIT 信号，并在监听到 EXIT 信号之后调用 clean 函数杀掉 EdgeX Foundry 的各组件以系统进程的方式运行起来的进程。

2.4.2 容器化部署 EdgeX Foundry

以容器化方式部署 EdgeX Foundry，即使用 docker-compose、Kubernetes 和 KubeEdge 对 EdgeX Foundry 进行容器化部署。本节对使用 docker-compose 部署 EdgeX Foundry 的步骤和注意事项进行详细说明。

1）获取 EdgeX Foundry 源码，命令如下：

#git clone https://GitHub.com/EdgeX Foundry/edgex-go.git

2）基于源码构建 EdgeX Foundry 各组件的 binary。

进入 edgex-go 源码根目录，命令如下：

#cd edgex-go

源码编译 edgex-go 命令如下：

#make build

源码编译 edgex-go 具体如图 2-16 所示。

```
[root@edge edgex-go]# make build
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/config-seed/config-seed ./cmd/config-seed
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/core-metadata/core-metadata ./cmd/core-metadata
CGO_ENABLED=1 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/core-data/core-data ./cmd/core-data
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/core-command/core-command ./cmd/core-command
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/support-logging/support-logging ./cmd/support-logging
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/support-notifications/support-notifications ./cmd/support-notifications
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/sys-mgmt-executor/sys-mgmt-executor ./cmd/sys-mgmt-executor
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/sys-mgmt-agent/sys-mgmt-agent ./cmd/sys-mgmt-agent
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o cmd/support-scheduler/support-scheduler ./cmd/support-scheduler
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-secrets-setup/security-secrets-setup ./cmd/security-secrets-setup
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-proxy-setup/security-proxy-setup ./cmd/security-proxy-setup
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-secretstore-setup/security-secretstore-setup ./cmd/security-secretstore-setup
CGO_ENABLED=0 GO111MODULE=on go build -ldflags "-X github.com/edgexfoundry/edgex-go.Version=master" -o ./cmd/security-file-token-provider/security-file-token-provider ./cmd/security-file-token-provider
```

图 2-16 源码编译 edgex-go

由图 2-16 可知，会在./cmd 下各组件子目录里生成相应的可执行文件，比如 config-seed 的可执行文件会在./cmd/config-seed 目录下，具体如图 2-17 所示。

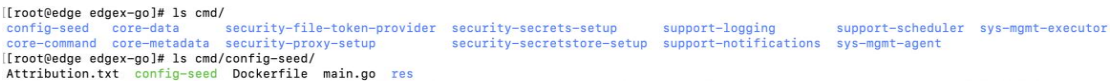

图 2-17 edgex-go 源码编译结果

3）构建 EdgeX Foundry 各组件的 image。

使用 docker 容器对 edgex-go 进行源码编译的命令如下：

make docker

使用 docker 容器对 edgex-go 进行源码编译具体如图 2-18 所示。

```
[root@edge edgex-go]# make docker
docker build \
        -f cmd/config-seed/Dockerfile \
        --label "git_sha=7ca3df59c01c7f56d3c83a840c5995cdb7718c20" \
        -t edgexfoundry/docker-core-config-seed-go:7ca3df59c01c7f56d3c83a840c5995cdb7718c20 \
        -t edgexfoundry/docker-core-config-seed-go:master-dev \
        .
Sending build context to Docker daemon  384.2MB
Step 1/38 : FROM golang:1.12-alpine AS build-env
1.12-alpine: Pulling from library/golang
c9b1b535fdd9: Pull complete
cbb0d8da1b30: Pull complete
d909eff28200: Pull complete
5a53565f2368: Pull complete
70debc116c7f: Pull complete
Digest: sha256:820a1f62d83b65d3ebf3d38afe08ac378ff27a7f5d5fd7f60ccd1601fd03060f
Status: Downloaded newer image for golang:1.12-alpine
 ---> b889ec575585
Step 2/38 : ENV GO111MODULE=on
```

图 2-18 使用 docker 容器对 edgex-go 进行源码编译

4）获取运行 EdgeX Foundry 各组件的 docker-compose.yml 文件，命令如下：

#curl -s -o docker-compose.yml

https://raw.GitHubusercontent.com/EdgeX

Foundry/developer-scripts/master/releases/edinburgh/compose-files//docker-compose-edinburgh-no-secty-1.0.1.yml

将 docker-compose.yml 文件的相关镜像替换为构建的最新镜像，命令如下：

vim docker-compose.yml

替换镜像具体如图 2-19 所示。

```
version: '3'
volumes:
  db-data:
  log-data:
  consul-config:
  consul-data:
  portainer_data:

services:
  volume:
    image: edgexfoundry/docker-edgex-volume:1.0.0
    container_name: edgex-files
    networks:
      edgex-network:
        aliases:
            - edgex-files
    volumes:
      - db-data:/data/db
      - log-data:/edgex/logs
      - consul-config:/consul/config
      - consul-data:/consul/data
```

图 2-19 替换镜像

5）运行 EdgeX Foundry。

使用 docker-compose 启动 edgex，命令如下：

docker-compose up -d

使用 docker-compose 启动 edgex-go 具体如图 2-20 所示。

```
[root@edge tmp]# docker-compose up -d
WARNING: Found orphan containers (edgex-device-
up.
Creating edgex-files ... done
Creating tmp_portainer_1   ... done
Creating edgex-mongo       ... done
Creating edgex-core-consul ... done
Creating edgex-config-seed ... done
Creating edgex-support-logging ... done
Creating edgex-core-data              ... done
Creating edgex-core-metadata          ... done
Creating edgex-support-notifications ... done
Creating edgex-sys-mgmt-agent         ... done
Creating edgex-support-scheduler      ... done
Creating edgex-core-command           ... done
Creating edgex-export-client          ... done
Creating edgex-ui-go                  ... done
Creating edgex-device-virtual         ... done
Creating edgex-export-distro          ... done
Creating edgex-support-rulesengine    ... done
```

图 2-20 使用 docker-compose 启动 edgex-go

至此，通过 docker-compose 以容器的方式运行 EdgeX Foundry 的相关步骤和注意事项也就结束了。

2.5 本章小结

本章梳理了云、边、端协同的边缘计算系统的整体架构，对云、边、端各部分包含的组件的技术栈进行了罗列，还分别对云、边、端各部分的部署方式和注意事项进行了系统梳理和详细说明。下一章将对整个边缘计算系统的逻辑架构及云、边、端之间的逻辑关系进行系统梳理。

第 3 章 边缘计算系统逻辑架构

通过第 1 章和第 2 章的学习，我们对边缘计算系统有了一个感性认识。本章将对整个边缘计算系统的逻辑架构及云、边、端之间的逻辑关系进行系统梳理。

3.1 边缘计算系统逻辑架构

由图 3-1 可知，逻辑架构侧重边缘计算系统云、边、端各部分之间的交互和协同关系，包括云、边协同，边、端协同和云、边、端协同 3 个部分。

1)云边协同：通过云部分 Kubernetes 的控制节点和边部分 KubeEdge 所运行的节点共同实现的。

2)边端协同：通过边部分 KubeEdge 和端部分 EdgeX Foundry 共同实现的。

3)云边端协同：通过云解决方案 Kubernetes 控制节点、边缘解决方案 KubeEdge 和端解决方案 EdgeX Foundry 共同实现的。

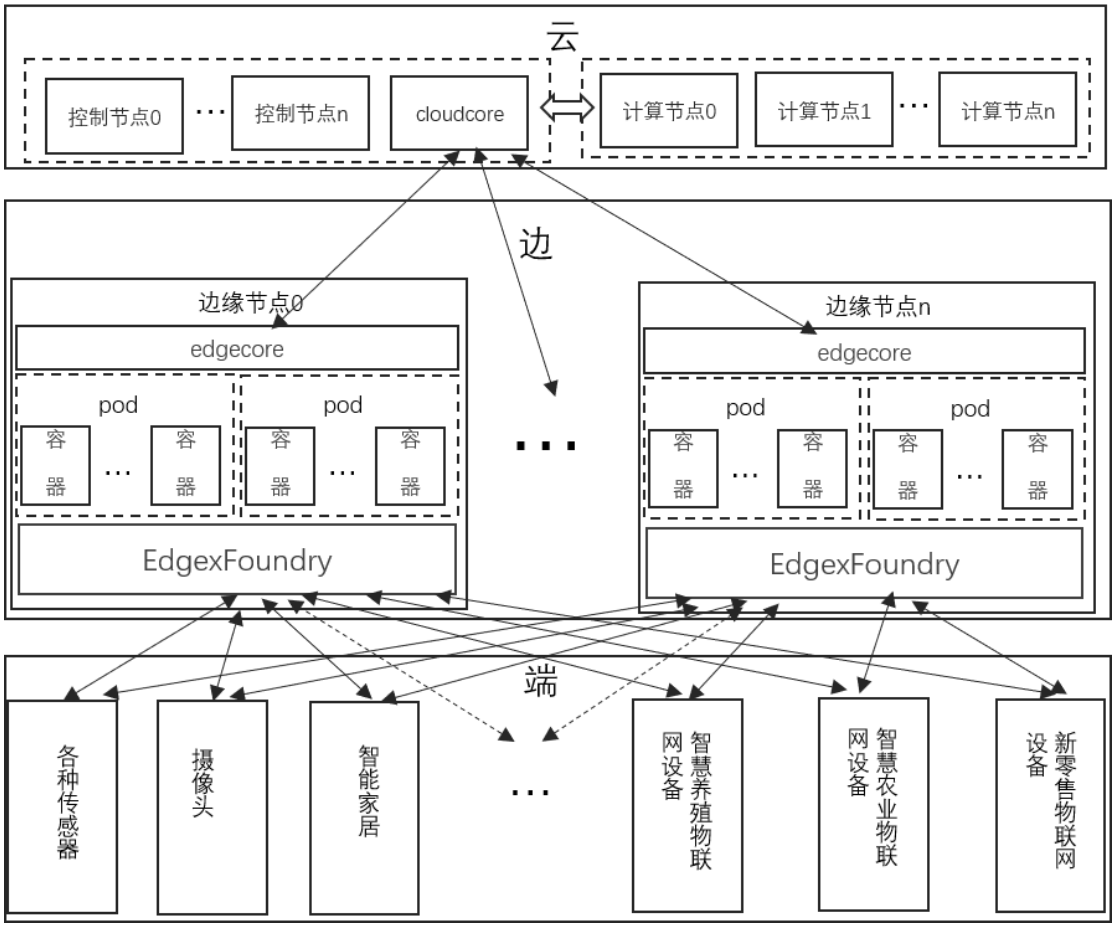

图 3-1 边缘计算系统逻辑架构

3.2 云边协同

云边协同的具体实现如图 3-2 所示。

1）Kubernetes 控制节点沿用云计算原有的数据模型，保持原有的控制、数据流程不变，即 KubeEdge 所运行的节点在 Kubernetes 上呈现出来的是 Kubernetes 的一个普通节点，Kubernetes 可以像管理普通节点一样管理 KubeEdge 所运行的节点。

2）KubeEdge 之所以能够运行在资源受限、网络质量不可控的边缘节点上，是因为 KubeEdge 在 Kubernetes 控制节点的基础上通过云部分的 CloudCore 和边缘部分的 EdgeCore 实现了对 Kubernetes 云计算编排容器化应用能力的下沉。云部分的

CloudCore 负责监听 Kubernetes 控制节点的指令和事件下发到边缘部分的 EdgeCore，同时将边缘部分 EdgeCore 上报的状态信息和事件信息提交给 Kubernetes 的控制节点；边缘部分的 EdgeCore 负责接收云部分 CloudCore 的指令和事件信息，并执行相关指令和维护边缘负载的生命周期，同时将边缘的状态信息和事件上报给云部分的 CloudCore。除此之外，EdgeCore 是在 Kubernetes 的 Kubelet 组件的基础上进行裁剪、定制而成的，将 Kubelet 在边缘上用不到的富功能进行了裁剪，针对边缘上资源受限、网络质量不佳的现状在 Kubelet 的基础上增加了离线计算的功能，使 EdgeCore 能够很好地适应边缘的环境。

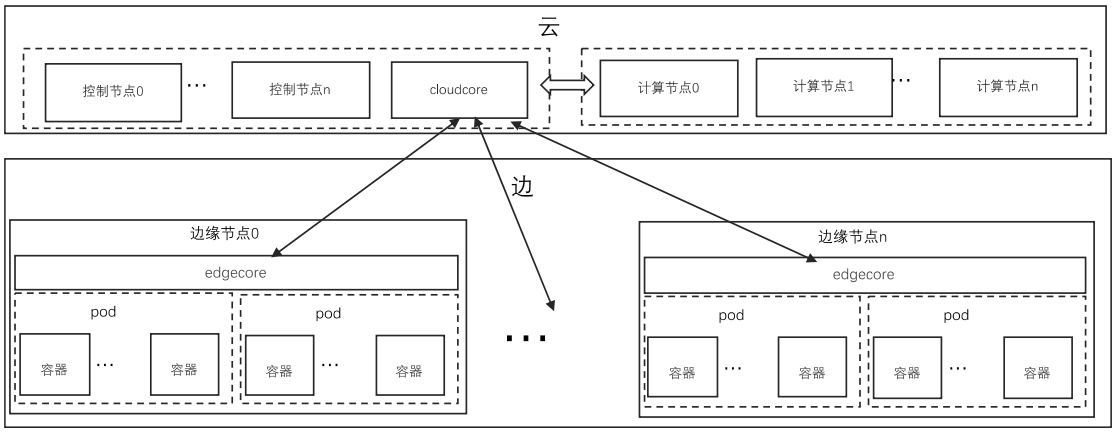

图 3-2 边缘计算系统云、边协同逻辑架构

3.3 边端协同

边端协同的具体实现如图 3-3 所示。

1）KubeEdge 作为运行在边缘节点管理程序，负责运行在边缘节点应用负载的资源、运行状态和故障自愈等。在本书的边缘计算系统中，KubeEdge 为 EdgeX Foundry 服务提供所需的计算资源，同时负责管理 EdgeX Foundry 服务的整个生命周期。

2）EdgeX Foundry 是由 KubeEdge 管理的一套 IoT SaaS 平台，该平台将以微服

务的形式管理多种物联网终端设备。同时，EdgeX Foundry 能够通过所管理的微服务采集、过滤、存储和挖掘分析多种物联网终端设备的数据，也可以通过所管理的微服务向多种物联网终端设备下发指令来对终端设备进行控制和监控设备的运行状态。

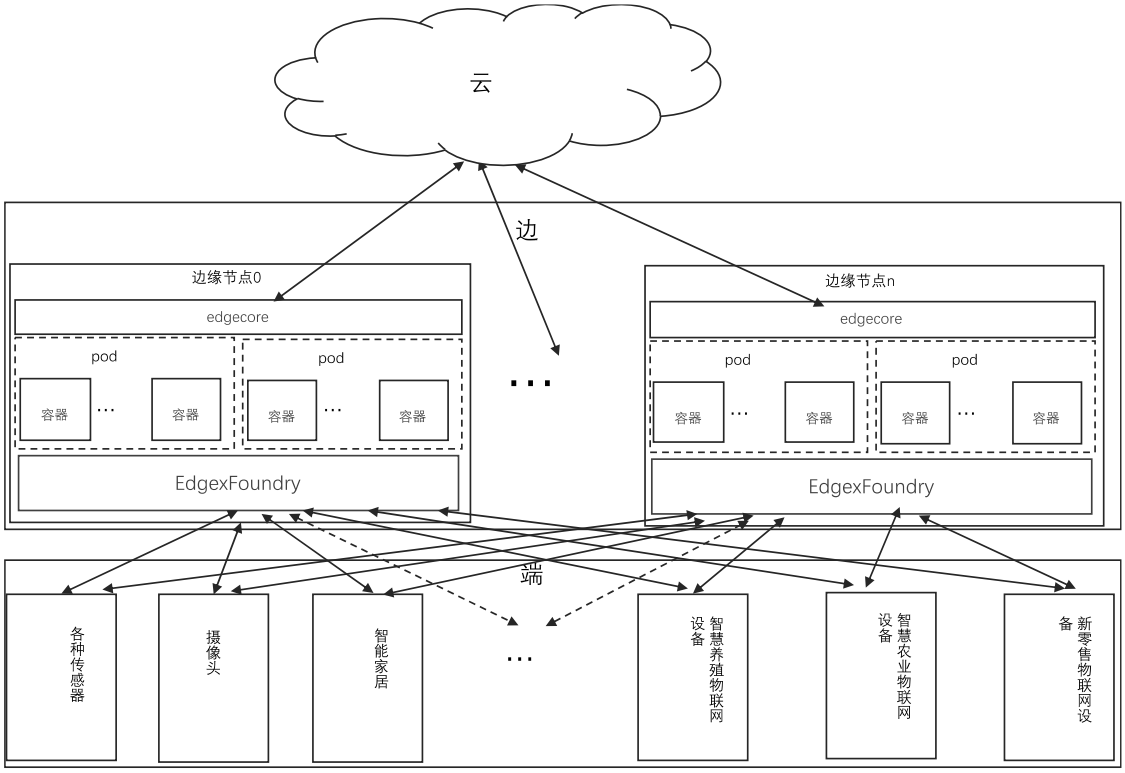

图 3-3 边缘计算系统边端协同逻辑架构

由图 3-4 可知，KubeEdge 自己的端解决方案由 MQTT 代理和对接支持各种协议设备的服务组成。

1）MQTT 代理作为各种物联网终端设备和 KubeEdge 节点之间的一个通信管道，负责接收终端设备发送的数据，并将接收到的数据发送到已经订阅了 MQTT 代理的 KubeEdge 节点上；

2）对接支持各种协议设备的服务 负责与支持相应协议的设备进行交互，能够采集设备的数据并发送给 MQTT 代理，并能够从 MQTT 代理接收相关指令下发到设备，但目前只支持 Bluetooth 和 Modbus 两种协议。

通过上述分析可知，KubeEdge 的端解决方案还比较初级。

1）KubeEdge 的端解决方案支持的负载类型还比较单一，目前只能通过 MQTT 代理支持一些物联网终端设备，对视频处理和使用 AI 模型进行推理的应用负载的类型还不支持。

2）对接支持各种协议设备的服务目前还比较少，只支持使用 Bluetooth 和 Modbus 两种协议的设备。

基于上述原因，本书的边缘计算系统的端解决方案没有使用 KubeEdge 自己的端解决方案，而是使用 EdgeX Foundry 这款功能相对比较完整的 IoT SaaS 平台作为端解决方案。

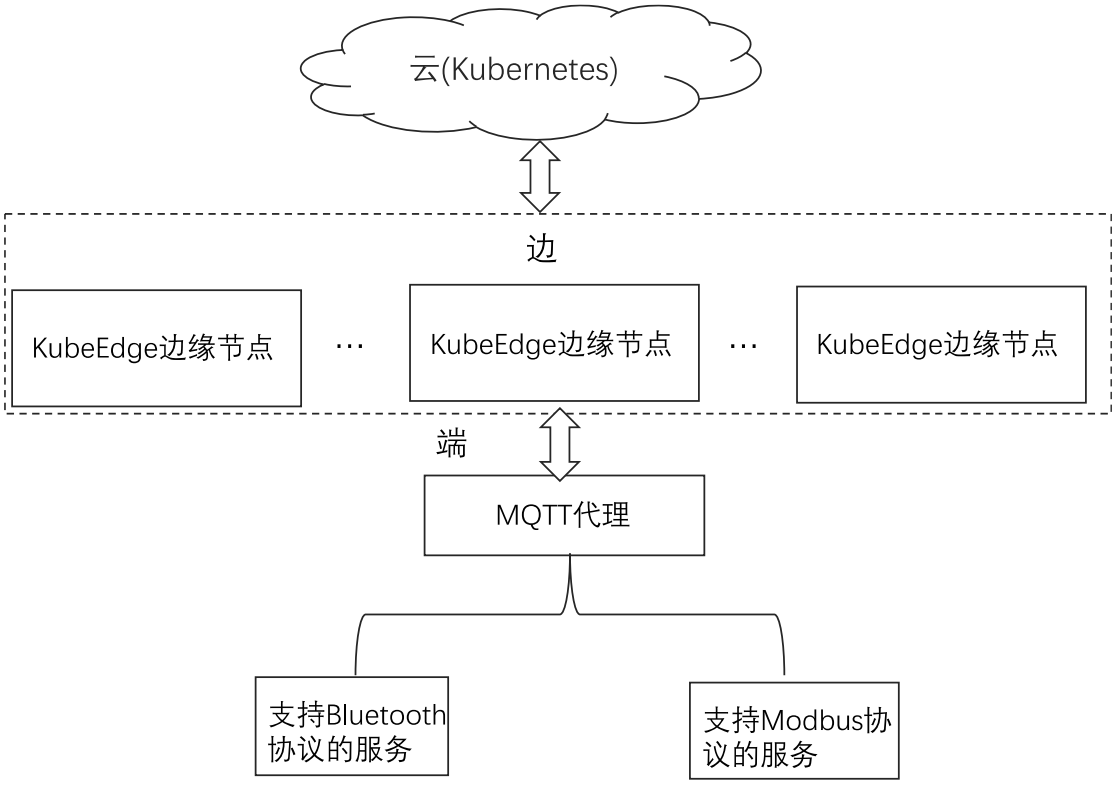

图 3-4 KubeEdge 端解决方案逻辑架构

3.4 云、边、端协同

边缘计算系统协同的理想效果如图 3-5 所示。

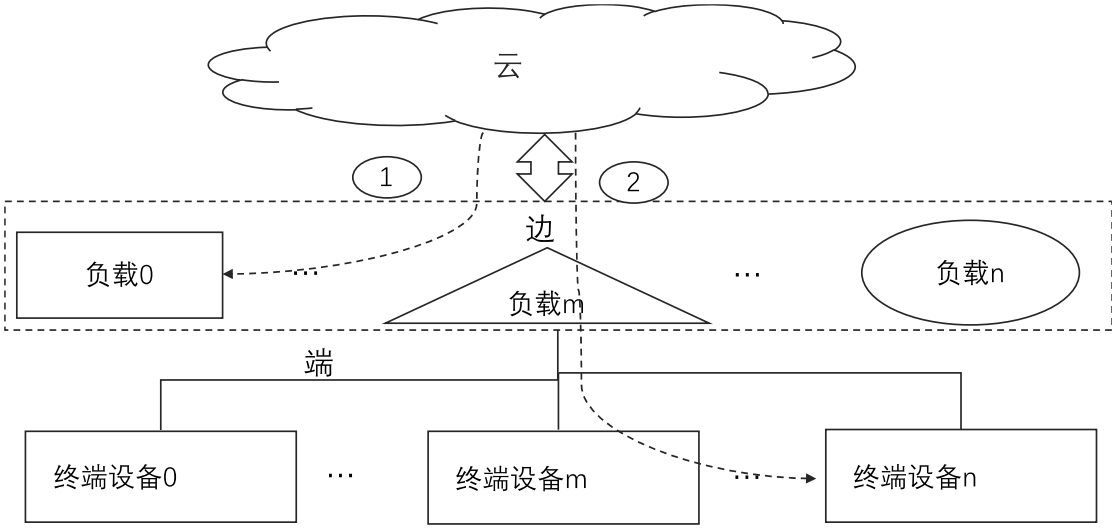

图 3-5 KubeEdge 云、边、端协同的理想效果

由图 3-5 可知，KubeEdge 云、边、端协同包括两层，即云、边协同和云、边、端协同。

1）云、边协同：云作为控制平面，边作为计算平台；

2）云、边、端协同：在云、边协同的基础上，管理终端设备的服务作为边上的负载，云可以通过控制边来影响端，从而实现云、边、端协同。

云、边、端协同是通过 Kubernetes 控制节点、KubeEdge 和 EdgeX Foundry 共同实现的，Kubernetes 控制节点下发指令到 KubeEdge 的边缘集群，操作 EdgeX Foundry 的服务，从而影响终端设备。目前，我们还不能通过 Kubernetes 的控制节点与终端设备直接交互。

3.4 本章小结

本章对整个边缘计算系统的逻辑架构及云、边、端之间的逻辑关系和现状进行了系统梳理。

1）从云、边协同的架构切入，对实现云、边协同的原理进行了梳理，同时对边解决方案的一些特性进行了说明。

2）从边、端协同的架构切入，对目前边、端协同的架构和原理进行了系统梳理，并对KubeEdge 自有的端解决方案的架构、原理和现状进行进行了说明。

3）从云、边、端协同的架构切入，对云、边、端协同的理想效果和边缘计算系统目前云、边、端协同的现状进行了梳理和对比。

HUAWEI

华为云开发者社区

更多精彩内容，扫码关注交流讨论

华为云开发者社区

华为云开发者联盟公众号