**分布式计算环境**

边缘计算与KubeEdge 的演示实例

刘建飞（21023008）、

马昊良（21023064）、

李锐（21026089）

2022年8月15日

目录

[1. 边缘计算研究现状 3](#_Toc10420)

[1.1 计算卸载 3](#_Toc2925)

[1.2 移动性管理 5](#_Toc3004)

[1.4 流量卸载技术 6](#_Toc15695)

[1.5 缓存加速 7](#_Toc19777)

[1.6 网络控制 7](#_Toc15065)

[1.7 边缘计算系统 7](#_Toc14205)

[2. KubeEdge 原理分析 10](#_Toc22585)

[3. 演示应用 12](#_Toc15676)

[3.1 KubeEdge环境搭建 12](#_Toc22275)

[3.2基于KubeEdge的Counter计数器应用搭建与访问 15](#_Toc5097)

[3.2 树莓派操作系统环境配置 19](#_Toc20598)

[4. 分析 34](#_Toc19272)

[5. 人员分工 34](#_Toc27349)

# 边缘计算研究现状

随着数据量的不断增加和对数据处理要求的不断提高，边缘计算应运而生。边缘计算是靠近数据来源的计算，如智能终端。它在网络边缘存储和处理数据。它具有邻近和位置感知功能，并为用户提供近端服务。在数据处理方面，它更快、实时和安全。还可以解决云计算能耗过高的问题，降低成本，降低网络带宽压力。边缘计算应用于生产、能源、智能家居、交通等各个领域。随着物联网的发展，对边缘计算模型的需求日益迫切，已成为研究热点。边缘计算的关键技术主要包括不同层次的计算卸载、移动性管理、流量卸载技术、缓存加速、网络控制等。我们将大致展示这些关键技术研究现状，并总结相关的系统与工程。

## 1.1 计算卸载

计算卸载是指终端设备将部分或全部计算任务卸载到资源丰富的边缘服务器，以解决终端设备在资源存储、计算性能以及能效等方面存在的不足。计算卸载的主要技术是卸载决策。卸载决策主要解决的是移动终端如何卸载计算任务、卸载多少以及卸载什么的问题。根据卸载决策的优化目标将计算卸载分为以降低时延为目标、以降低能量消耗为目标以及权衡能耗和时延为目标的3种类型。

如果任务需要卸载，则完成此任务所花费的时延包括上传任务到移动边缘计算（MEC，Mobile Edge Computing）服务器的传输时延、在MEC服务器上的执行时延和返回执行结果的传输时延。不进行卸载时，时延只包括在本地执行的时间。为保证用户有个很好的体验，需要最大程度地降低任务的完成时延，至少需要满足完成任务所需的最低时延要求。 文献[1]提出了一种高效的一维搜索算法来寻找 最优的任务调度策略。首先利用马尔可夫链理论，根据任务缓冲区的排队状态、本地处理单元的执行状态以及传输单元的状态，分析了在给定的计算任务调度策略下，移动设备上每个任务的平均延迟和平均功耗。然后，建立了功率约束时延最小化问题的数学模型，利用一维搜索算法来寻找最优的调度策略。文献[2]研究了具有带宽和延迟要求的单个应用程序和多个应用程序的应用程序配置问题。对于单一应用和可以并行的多应用场景，提出了一种完全多项式时间近似方案。对于一般的不可并行的多应用场景，提出了一个随机化算法。通过仿真结果表明，与启发式算法相比，所提出的算法可以最大程度地缩短任务的执行时延。在文献[3]中用户将计算任务卸载到多个边缘服务器，并在预定的时隙上从服务器下载结果，通过联合优化任务分配和资源分配，最大限度地减少任务的计算延迟。实验表明，通过此优化方案，可以显著减少任务的计算时延。

如果任务在本地执行只需考虑设备本身能耗，若卸载到MEC服务器，除了设备自身能耗，还包括将任务卸载到MEC服务器上的传输能耗。文献[4]结合5G异构网络的多路访问特性，设计了节能计算卸载方案，在时间约束下共同优化了卸载决策 和无线资源分配，最小化系统能耗。文献[5]考虑了任务可以在移动设备上执行也可以在云服务器上执行两种执行模式，研究在时延约束内如何最佳地配置时钟频率和最佳地调度数据传输，通过求解约束优化问题来最小化设备总能。最后通过实验表明通过将应用程序卸载到云执行，可以节省设备能耗。

不管是能耗和时延都会影响整个系统的性能，所以为了进一步地提升设备的性能，减少整个系统的执行成本，需要在时延和能耗之间寻找平衡，使设备的性能和整个系统的收益达到最好。文献[6]研究了任务卸载调度和资源分配问题。 提出了一种基于交替最小化的低复杂度次优算法，来最小化任务执行延迟和设备能耗的加权总和。仿真表明所提出的算法能有效地缩短任务的执行时延和降低设 备的能耗。文献[7]将任务依赖关系建模为一般拓扑图，使用线性规划的方法解决卸载决策、延迟和能耗联合优化问题。

目前计算卸载技术根据卸载方式的不同，主要集中在粗粒度、细粒度和MEC 与 Device-to-Device（D2D）技术协作三方面进行研究。按照设备上应用任务的划分以及有无 D2D技术 可以将任务卸载方式分为粗粒度的卸载方式、不具有依 赖关系的细粒度的卸载方式、具有依赖关系的细粒度的卸载方式和MEC与D2D技术协作卸载方式。

## 1.2 移动性管理

边缘计算依靠资源在地理上广泛分布的特点来支持应用的移动性，一个边缘计算节点只服务周围的用户。云计算模式对应用移动性的支持则是服务器位置固定，数据通过网络传输到服务器，所以在边缘计算中应用的移动管理是一种新模式。

由于移动用户可能穿越不同的位置单元，设计多单元MEC网络面临的一个挑战是移动性管理，以保证业务的不间断。支持移动性的直接方式是业务迁移，即沿着移动用户的移动路径，将移动用户正在进行的计算业务不断迁移到与其动态关联的服务器/业务上。然而，用户移动性的不确定性使得最优迁移策略难以设计。有三种方法来解决这个问题。第一个是基于短期用户移动性和服务延迟的预测来做出更明智的迁移决策[8]。第二种方法是基于将用户移动建模为马尔可夫过程的在线迁移决策，并应用马尔可夫决策过程理论(MDP)对决策[9]进行优化。这种方法的局限性在于它需要用户流动的统计信息，而在实践中并不总是可以得到这些信息。还有专注于在线迁移设计，没有对未来用户移动性的先验知识。在[10]中提出了学习驱动的迁移方案，[11]采用了深度强化学习方法，用户使用试错法解决先验知识缺乏的问题。另一方面，利用Lyapunov优化技术，在[12]中提出了一种在线迁移策略，该策略平衡了服务延迟、所产生的迁移成本和长期用户迁移。

## 1.4 流量卸载技术

为了实现业务应用在无线网络中的本地化、短距离部署和低延迟、高带宽传输能力，无线网络具有流量卸载能力。在移动边缘计算中，边缘网络流量的卸载是一个非常重要的问题。流量卸载是指将符合特定卸载规则的流量卸载到移动边缘网络(即本地特定网络，可以是Intranet或Internet)，以节省回程带宽、减少延迟，并方便扩展其他MEC业务[13]。[14]提出了一种两层异构无线网络中移动用户能量高效的流量卸载方法。实验结果表明，该方法在典型网络环境下可节省高达34%的能量。

## 1.5 缓存加速

移动边缘缓存技术包括基站缓存、移动内容分发网络和透明缓存。缓存加速技术可以提高内容分发效率，改善用户体验。将内容缓存到移动网络边缘后，用户可以就近获取内容，从而避免了内容的重复传输，减轻了回程网和核心网的压力。同时，边缘缓存可以减少用户请求的网络延迟，从而改善用户的网络体验。此外，边缘缓存还可以开放移动网络资源环境，为用户提供更丰富的服务。[15]提出了一个认知代理(cognitive agent, CA)来帮助用户提前在MEC上缓存和执行任务，并协调通信和缓存来缓解MEC的压力。

## 1.6 网络控制

边缘网络是一个给定的非技术描述的网络(公共电信网络的边缘)。边缘网络包括汇聚层网络和接入层网络的一部分或全部，是接入用户的最后一段网络，是介于现有核心网络和大用户之间的商业网络。在网络控制方面，[16]提出了一种有效的工作负载切片方案，用户使用软件定义网络处理多边缘云环境下的数据密集型应用。

## 1.7 边缘计算系统

现有的边缘计算系统大致可以分为三类，它们在系统架构、编程模型和各种应用方面都有创新：

1)从云推送（Push From Cloud）:在这一类中，云提供商将服务和计算推送到边缘，以利用局部性，减少响应时间，改善用户体验。代表性系统有Cloudlet、Cachier、AirBox和CloudPath。2009年，卡内基梅隆大学(Carnegie Mellon University, CMU)提出了Cloudlet[17]的概念，开放边缘计算倡议也是由Cloudlet项目演变而来的。Cloudlet是一个可信任的、资源丰富的计算机或计算机集群，它与互联网良好连接，并可用于附近的移动设备。CloudPath[18]是多伦多大学提出的边缘计算系统。在这样的系统中，从用户设备到云数据中心的路径提供了计算、存储等多种资源。它支持按需分配和多级体系结构的动态部署。CloudPath的主要思想是实现所谓的“路径计算”，与传统的云计算相比，它可以减少响应时间并提高带宽利用率。

许多传统云计算服务商都在积极推动云服务更贴近用户，缩短客户与云计算的距离，以免市场被移动边缘计算抢走。例如，微软在2017年推出了AzureStack，它可以将云计算能力集成到终端中，数据可以在终端设备上进行处理和分析。

2)从物联网拉取（Pull From IoT）:物联网应用将服务和计算从遥远的云端拉到临近的边缘，处理物联网设备产生的海量数据。代表系统包括 PCloud, ParaDrop, FocusStack和SpanEdge。 ParaDrop[19]是由威斯康星大学麦迪逊分校的WiNGS实验室开发的。它是一个边缘计算框架，使计算/存储资源接近移动设备和数据源，供第三方开发人员使用。它的目标是以一种友好的方式将智能带到网络边缘。SpanEdge[20]是瑞典皇家理工学院的一个研究项目。它统一了云中心节点和近边缘中心节点，减少了网络延迟，并提供了允许程序在数据源附近运行的编程环境。另外开发人员可以专注于开发流应用程序，而无需考虑数据源位于何处和分布在何处。嵌入式芯片上系统(soc)的进步已经催生了许多强大到足以运行嵌入式操作系统和复杂算法的物联网设备。许多制造商将机器学习(ML)甚至深度学习功能集成到物联网设备中。利用边缘计算系统和工具，物联网设备可以有效地共享计算、存储和网络资源，同时保持一定的独立性。

3)混合的云-边缘分析(Hybrid Cloud-Edge Analytics):云和边缘优势的集成提供了一种解决方案，在现代高级服务和应用中促进全球最优结果和最小响应时间。代表性系统包括Firework和Cloud-Sea计算系统。这种边缘计算系统利用物联网设备的处理能力来过滤、预处理和聚合物联网数据，同时利用云服务的能力和灵活性对这些数据进行复杂的分析。例如，阿里云2018年推出首款物联网边缘计算产品LinkEdge，将其在云计算、大数据、人工智能(AI)等领域的优势拓展到边缘，构建云/边缘融合的协同计算系统;亚马逊在2017年发布了Amazon Web Services (AWS) Greengrass，可以将AWS无缝扩展到设备上，设备可以对其生成的数据进行本地操作，同时数据被传输到云端进行管理、分析和存储。

除了为特定目的而设计的边缘计算系统，一些开源边缘计算项目也在近期启动。Linux基金会发布了两个项目:2017年的EdgeX Foundry和2018年的Akraino Edge Statck[21]。开放网络基金会(ONF)发起了一个项目，即CORD[22]。Apache软件基金会发布了Apache Edgent。微软于2017年发布了Azure IoT Edge，并于2018年宣布其为开源软件。其中，CORD和Akraino Edge Stack专注于提供边缘云服务;EdgeX Foundry和Apache Edgent专注于物联网，旨在解决给边缘计算在物联网中的实际应用带来困难的问题;Azure IoT Edge提供混合云边缘分析，这有助于将云解决方案迁移到物联网设备。

# KubeEdge 原理分析

KubeEdge提供了一个完整的基于Kubernetes的边缘计算解决方案，KubeEdge是用来构建扩展云的边缘计算解决方案的。控制平面位于云中，但可伸缩和扩展。同时，边缘可以在脱机模式下工作。此外，它是轻量级和容器化的，并且可以在边缘支持异构硬件。通过优化边缘资源的利用，KubeEdge为边缘解决方案节省了大量的设置和运行成本。KubeEdge支持边缘集群的编排和管理，类似于Kubernetes在云中的管理方式。



图1：KubeEdge架构图

KubeEdge架构如图1所示，包括云端和边缘端两部分。

用户通过K8s的API 服务器链接到CloudCore，其中CloudHub是WebSocket服务器，负责监控云端的变化、缓存并发送消息到EdgeHub。

1）EdgeController是扩展的Kubernetes控制器，负责管理边缘节点和Pods 的元数据（Pod是kubernetes中最小的资源管理组件，Pod也是最小化运行容器化应用的资源对象。一个Pod代表着集群中运行的一个进程），EdgeController控制数据发送到指定的边缘节点。

2）DeviceController也是扩展的Kubernetes控制器，负责管理边缘设备, 实现边缘设备元数据/状态数据在云端与边缘端的同步。

3）EdgeHub是WebSocket客户端，负责与云边服务交互实现边缘计算。其中包括将云边资源同步更新到边缘端以及将边端主机、设备状态变化广播至云端。

4）Edged：负责Pod生命周期的管理，可以看成一个简易版的kubelet（kubelet 是 kubernetes 工作节点上的一个代理组件，运行在每个节点上）。

5）EventBus：EventBus 是一个MQTT客户端负责与MQTT服务器Mosquitto的交互，为其他组件提供发布与订阅功能（MQTT(消息队列遥测传输)是ISO 标准(ISO/IEC PRF 20922)下基于发布/订阅范式的消息协议）。

6）ServiceBus：ServiceBus 是一个HTTP客户端与HTTP服务器使用REST进行交互，为云端组件提供HTTP客户端功能，使其请求到达运行在边缘端的HTTP服务器。

7）DeviceTwin：负责存储设备状态，并将设备状态同步到云端，同时也提供了了应用的查询接口。

8）MetaManager：MetaManager 是Edged与Edgehub之间的message 处理器，同时，也负责将元数据存储/查询到/从一个轻量级数据库SQLite。

基于WebSocket 和 消息封装，同时优化了原生Kubernetes 中一些不必要的请求，KubeEdge实现了边缘场景下的云边可靠通信。

# 演示应用

## 3.1 KubeEdge环境搭建

**软件包版本**

* Kernel version: 3.10.0-1160.45.1.el7.x86\_64
* Kubelet version: v1.18.0
* KubeEdge version： v1.7.0
* Golang version: go1.15.3 linux/amd64 for CentOS7.x-86\_x64，go1.15.3 linux/arm64 for Raspberry PI
* Docker version: v19.03

准备好的两台云服务器节点ip地址和主机名分别为：

master 10.101.15.27

node1 10.101.15.28

首先初始化k8s集群主节点

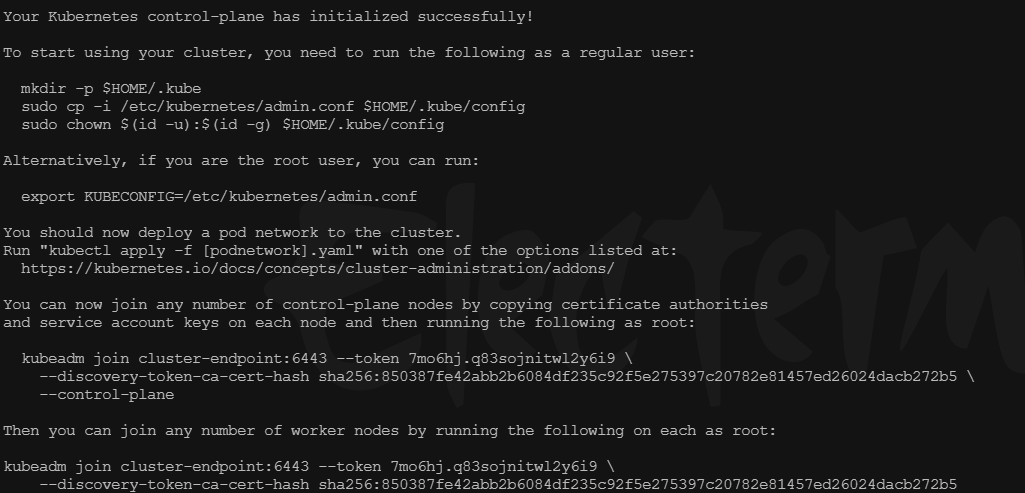


图2 k8s启动成功

KubeEdge由云和边缘组成。它建立在Kubernetes之上，为联网、应用部署和云与边缘之间的元数据同步提供核心基础设施支持。所以如果我们想要设置KubeEdge，我们需要设置Kubernetes集群(可以使用现有的集群)，云端和边缘端。

* 在cloud side, 需要安装

- Docker

- Kubernetes cluster

- cloudcore

* 在 edge side, 需要安装

- Docker

- MQTT (We can also use internal MQTT broker) （配置可以选用，不是一定需要）

- edgecore

安装有两种方式，一种源码编译手动安装，还有一种是使用kubeedge提供的工具-keadm。

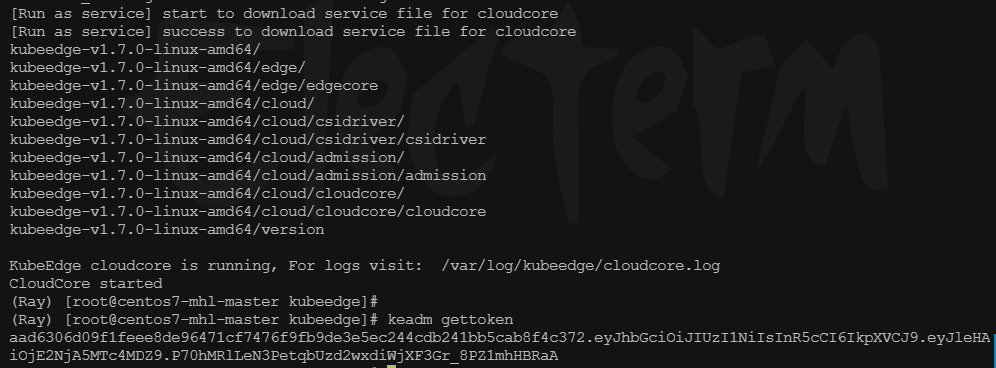


图3 CloudCore启动成功

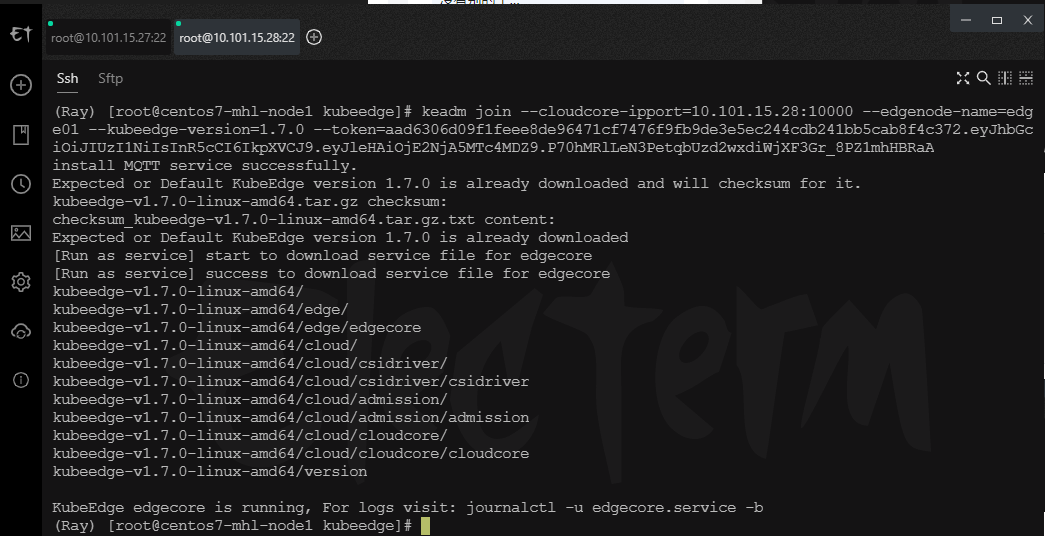


图4 KubeEdge EdgeCore启动成功

通过在CloudCore节点获取Token，将EdgeCore节点介入到集群中，

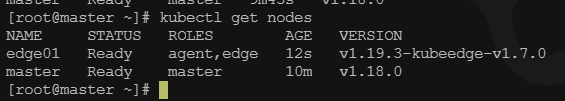


图5 边缘集群查看

可以看到edge01加入成功。

## 3.2基于KubeEdge的Counter计数器应用搭建与访问

KubeEdge Counter Demo 计数器是一个伪设备，用户无需任何额外的物理设备即可运行此演示。计数器在边缘侧运行，用户可以从云侧在 Web 中对其进行控制，也可以从云侧在 Web 中获得计数器值,原理图如下:

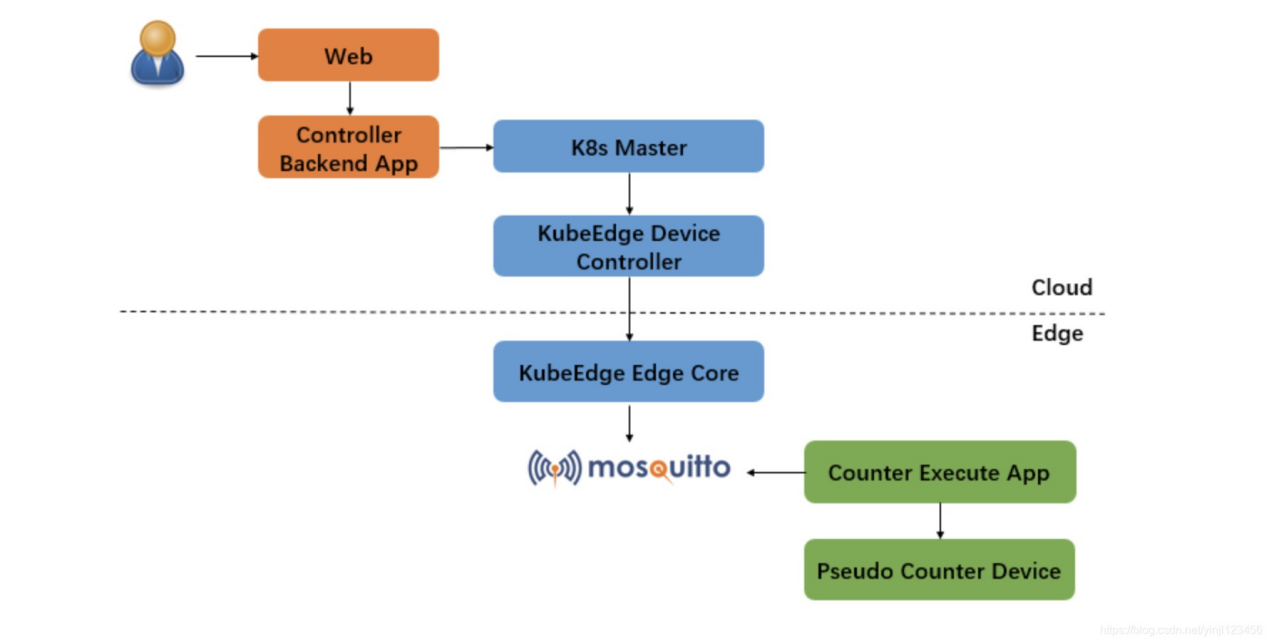


图6 Counter计数器原理图

本计数器使用云服务器模拟各类型节点，通过kubernetes搭建集群并初始化主节点，集群规模为一主一从，在 k8s 集群基础上使用keadm工具在主节点master上部署 Kubeedge cloudcore服务，使主节点充当云端控制节点，在从节点node1上使用keadm加入集群并部署KubeEdge edgecore服务，使从节点充当边缘节点，编写业务逻辑实现计数器功能，将应用打包成容器通过KubeEdge调度机制将计数器容器调度到边缘节点node1上运行，在云端控制节点运行Webcontroler APP 并开放8089端口，实现云端master节点控制计数器开关，并实时获取边缘节点计数器状态。

首先根据yaml文件分别创建模型和实例，

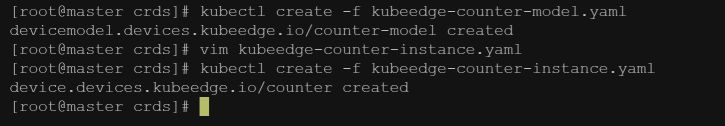


图7创建模型和实例

下一步分别创建云端应用和边缘端应用，云端应用web-controller-app用来控制边缘端的pi-counter-app应用，修改端口为8089。

部署云端应用web-controller-app和边缘端应用pi-counter-app。

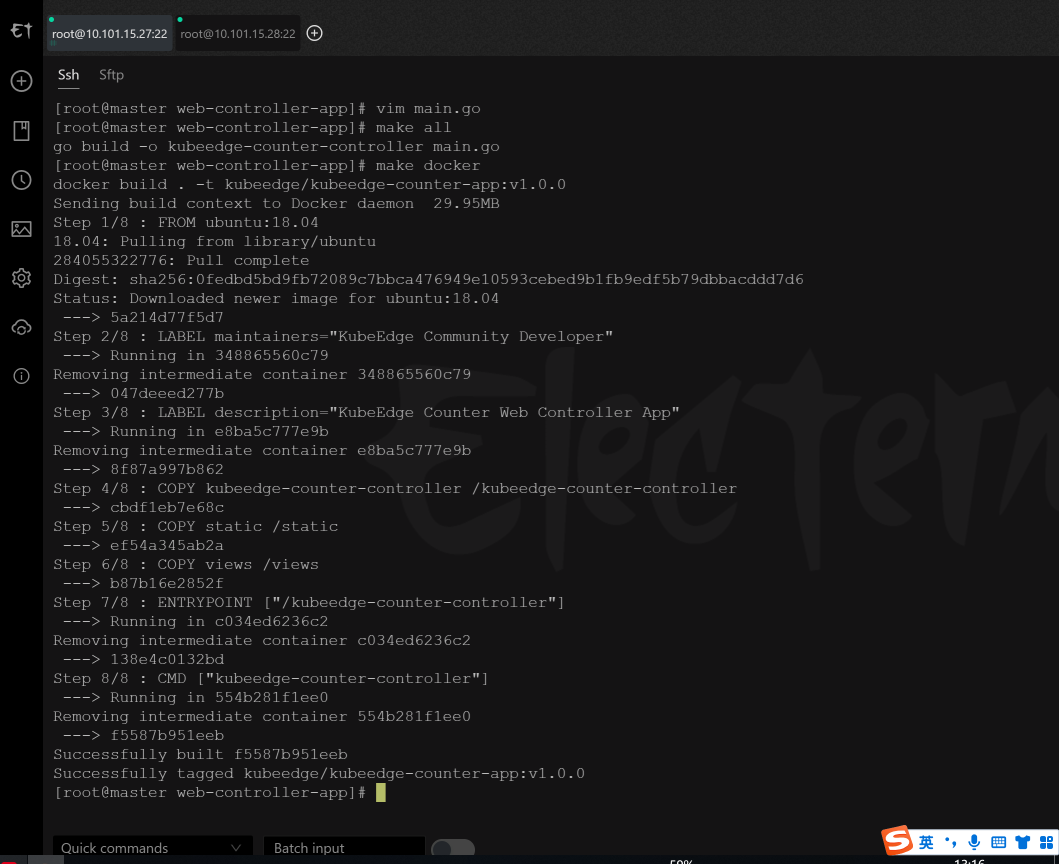


图8 云端应用web-controller-app:v1.0.0

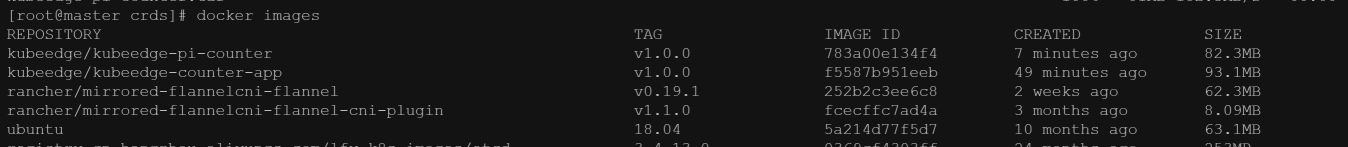


图9 生成云端和边缘节点镜像

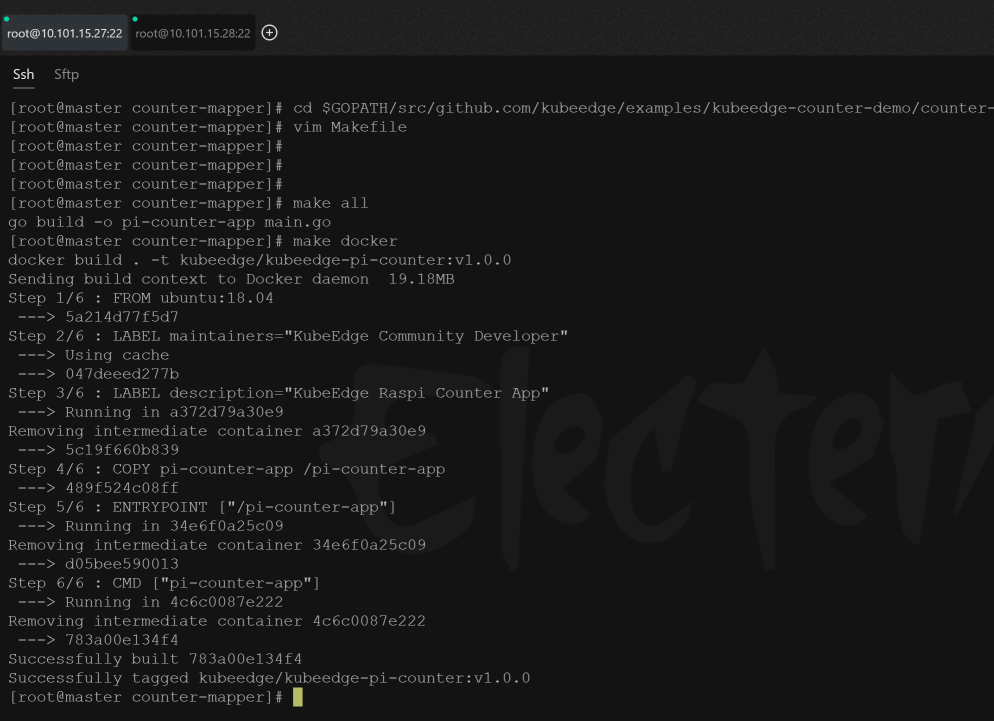


图10 边缘端pi-counter-app:v1.0.0镜像

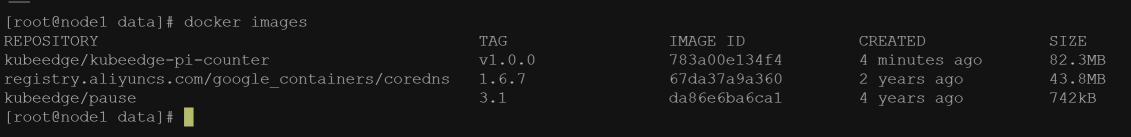


图11 node节点的镜像创建结果

至此，KubeEdge Demo的云端部分和边缘端的部分部署完成。

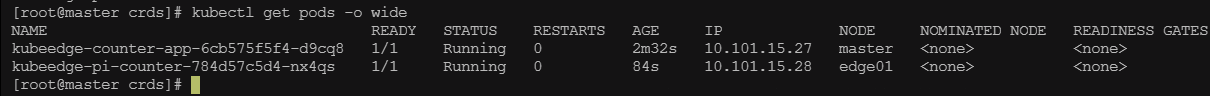


图12 在master节点查询容器

直接访问<http://10.101.15.27:8089/>进入Web欢迎界面，点击Execute按钮，应用开始执行，可以在edge边缘节点上通过以下命令查看执行结果。

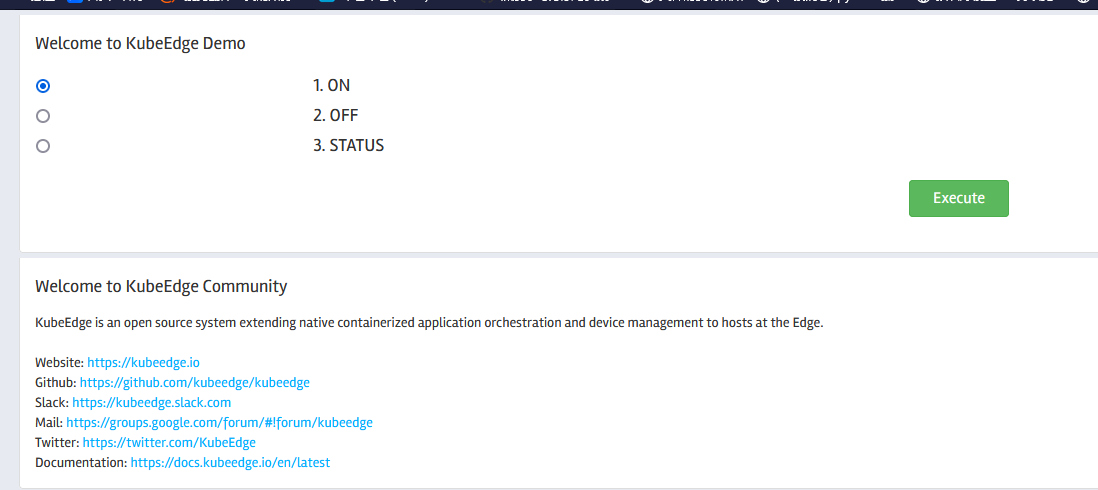


图13 Web界面

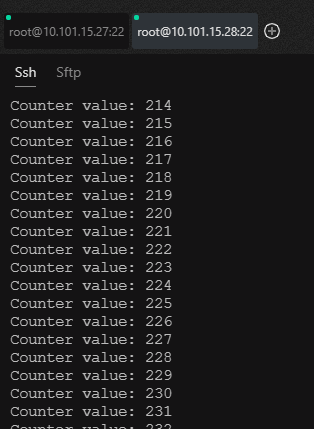


图14 边缘端Counter计数器

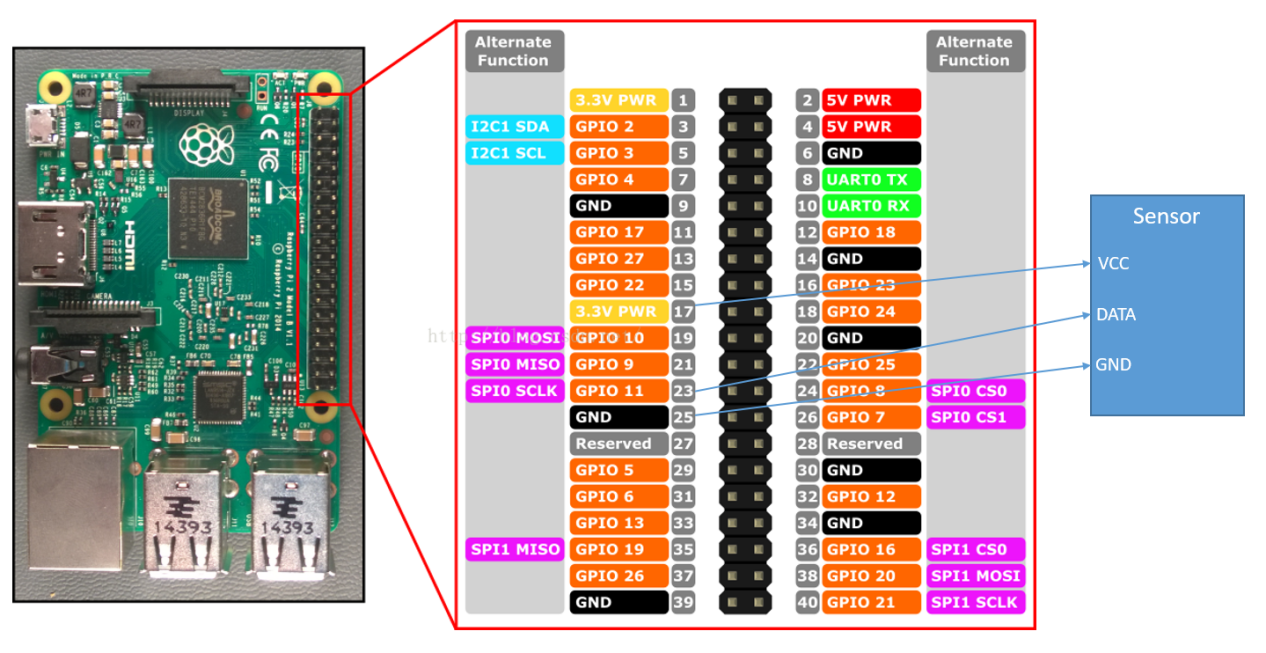
表示测试成功。

## 3.2 树莓派操作系统环境配置

### 3.2.1.硬件配置

所需硬件：树莓派4B、杜邦线、温度传感器、SD卡、读卡器





树莓派配置：

Raspberry PI 4B

CPU(s): 4

Memory: 4G

Disk: 16G

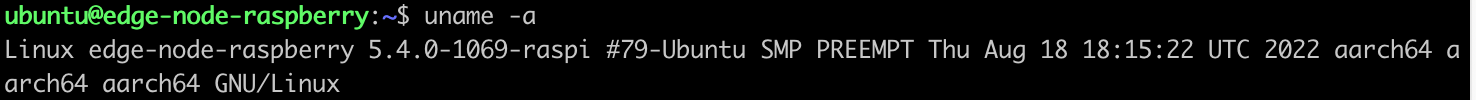
Vendor ID: ARM

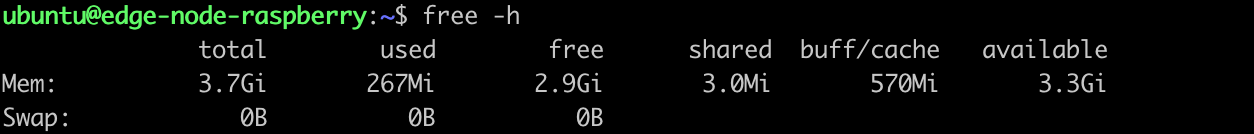
Model name: Cortex-A72

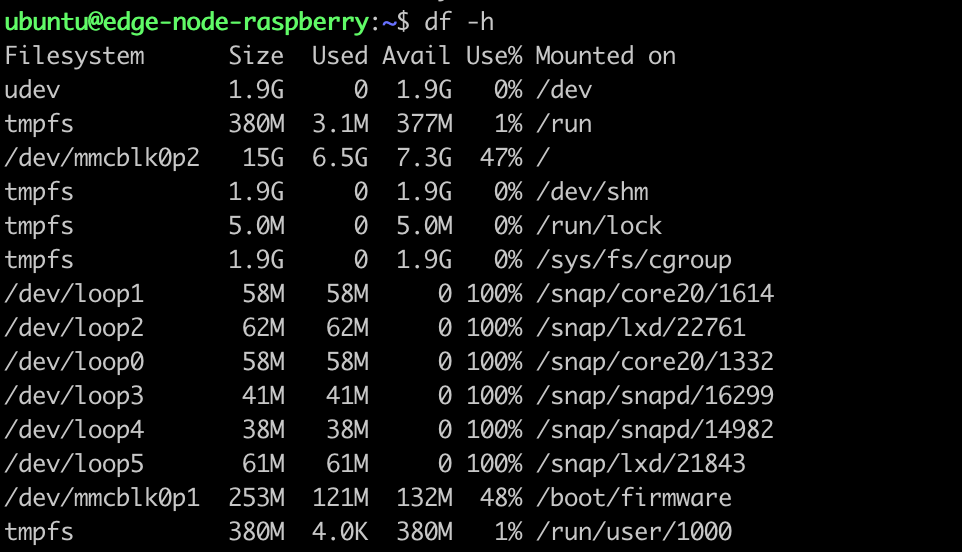
CPU max MHz: 1500.0000

CPU min MHz: 600.0000

温度传感器: DHT11







### 3.2.2.软件环境

**3.2.2.1. 操作系统**

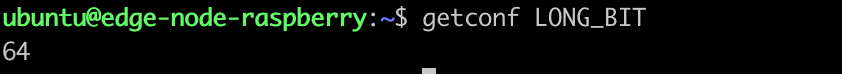
本实验为树莓派采用的操作系统是Ubuntu20.04.4 LTS（Liunx 5.4.0-raspi aarch64）。树莓派于2022年官方支持安装64位操作系统和软件，并且树莓派4B系列拥有支持64位的硬件架构，我们采用的也是最新支持的64位ubuntu系统。

通过树莓派官网提供的系统烧录软件在SD卡烧录系统。



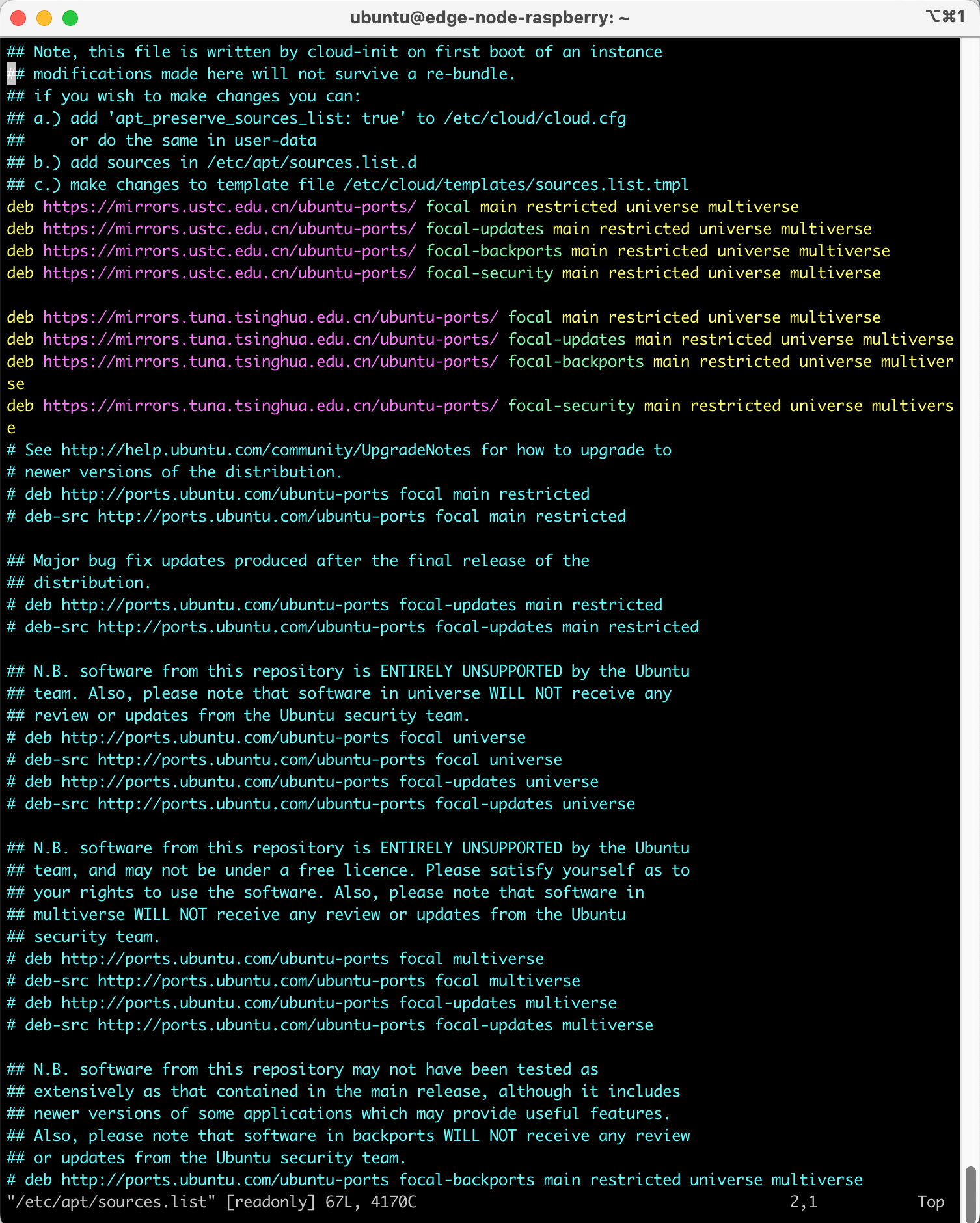
通过路由器找到树莓派所在的内网ip并进行设置





首先修改软件镜像源，删除原先的镜像源，添加中科大和清华镜像源：





Ubuntu20.04自动休眠禁止

sudo systemctl mask sleep.target suspend.target hibernate.target hybrid-sleep.target

根据情况需要配置VNC

首先安装桌面环境：

sudo apt update

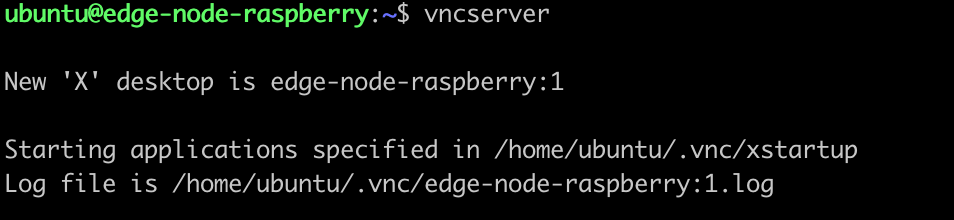
sudo apt upgrade

sudo apt install xfce4 xfce4-goodies

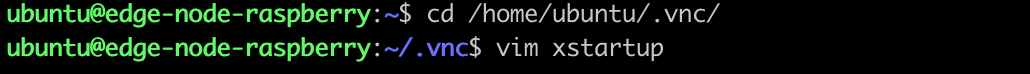
接下来安装VNC

sudo apt install tightvncserver

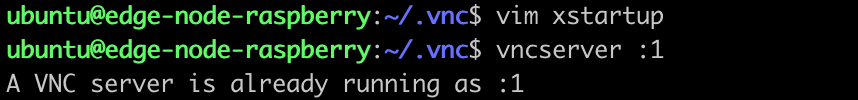
启动VNC，第一次配置时需要设置密码，默认端口1

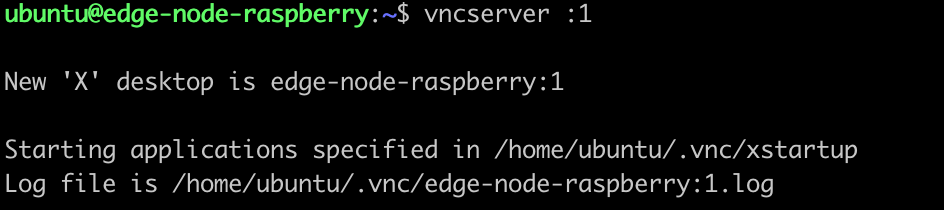


修改vnc配置文件









**安装docker**

sudo apt update

sudo apt install docker.io

sudo systemctl start docker

sudo systemctl enable docker

**为docker配置镜像加速**

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["https://0y4nh6yd.mirror.aliyuncs.com"]

}

EOF

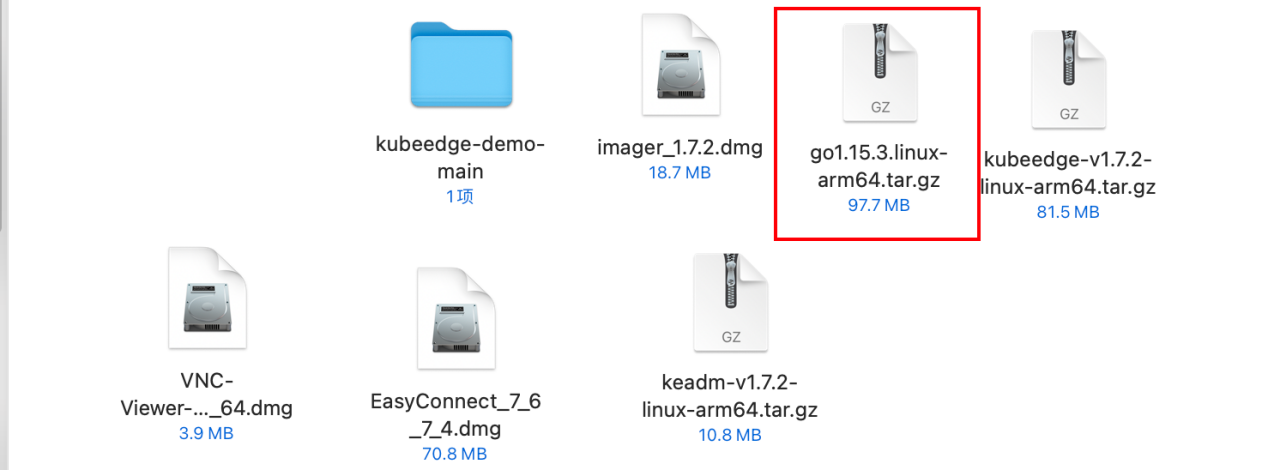
sudo systemctl daemon-reload

sudo systemctl restart docker



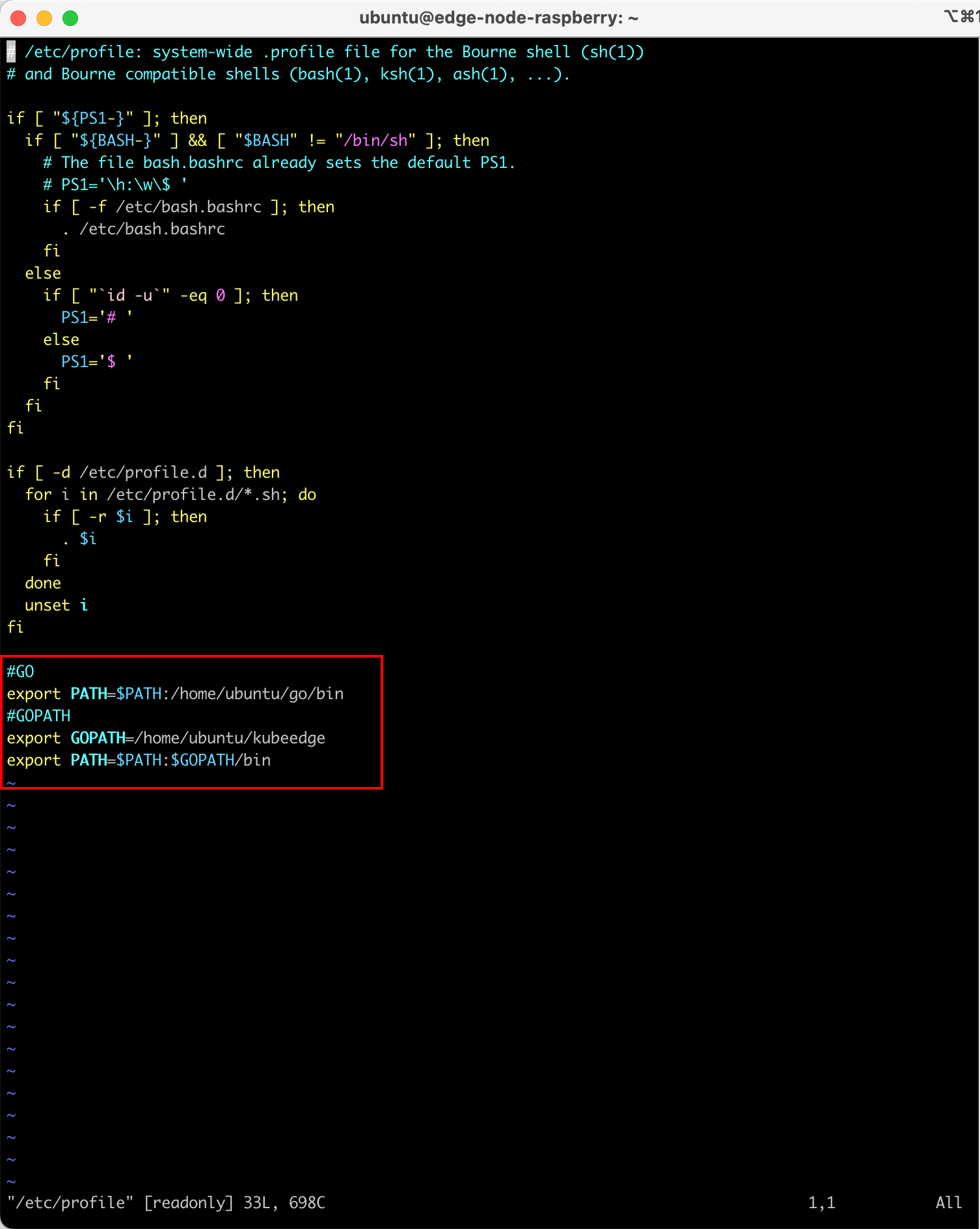
安装go语言环境

官网下载tar软件包通过sftp传输到树莓派上安装



tar -zxvf go1.15.3.linux-arm64.tar.gz

#在/etc/profile里加入环境变量，以便一次部署失败不用再次source

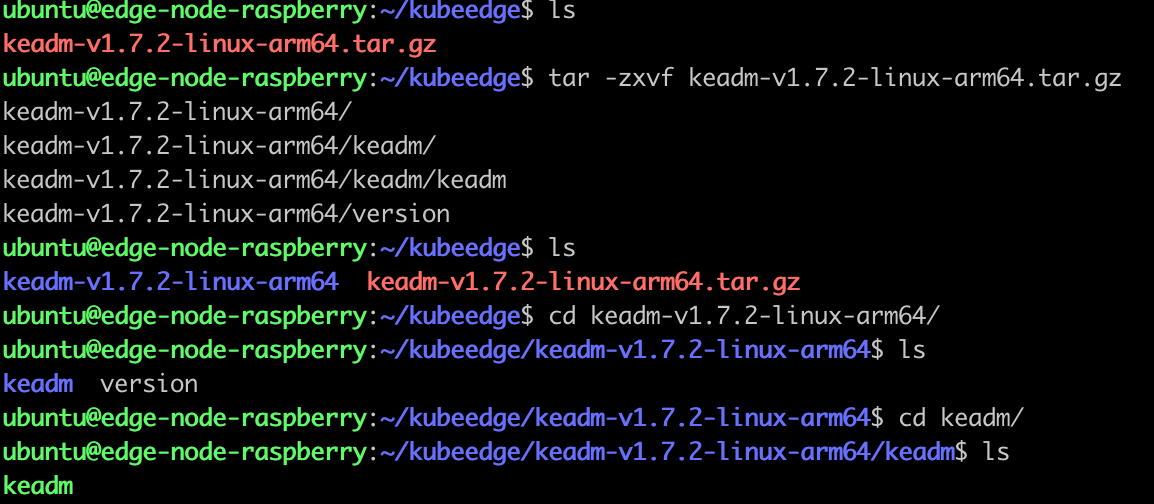


**3.2.2.2. 安装kubeedge**

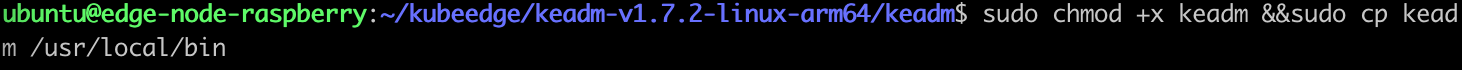
**1）keadm**

由于kubeedge版本更新，现在不再需要手动自己编译需要的可执行文件，在github下载后的压缩包内存在可执行文件。

树莓派边端不需要自己手动编译edgecore，只需要安装keadm进行管理，在加入云端时自动拉取edgecore并安装运行。

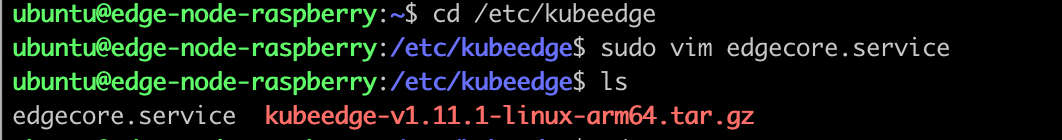


提高keadm权限并设置系统变量里的可执行文件

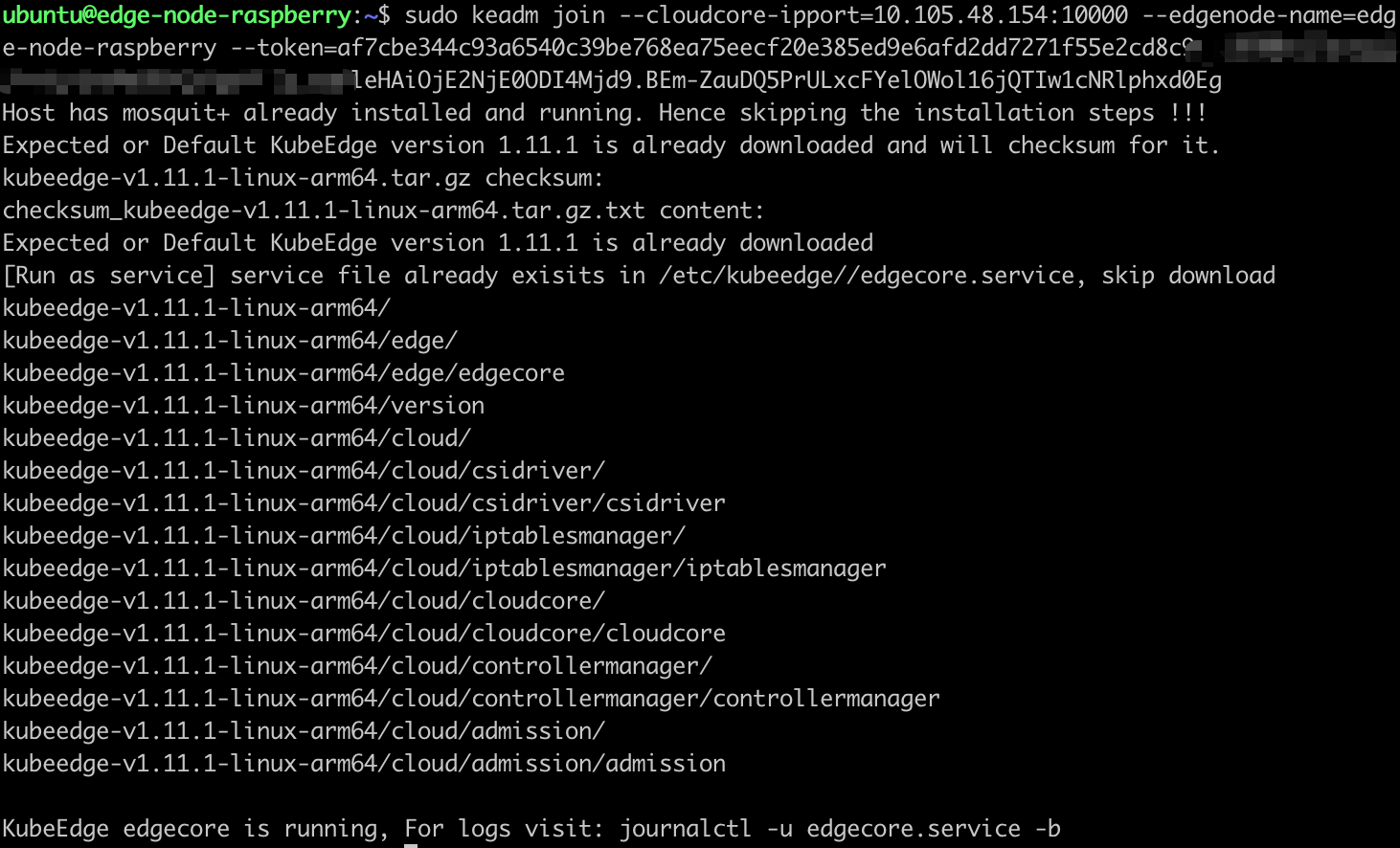


**2） 加入云端并安装运行edgecore**

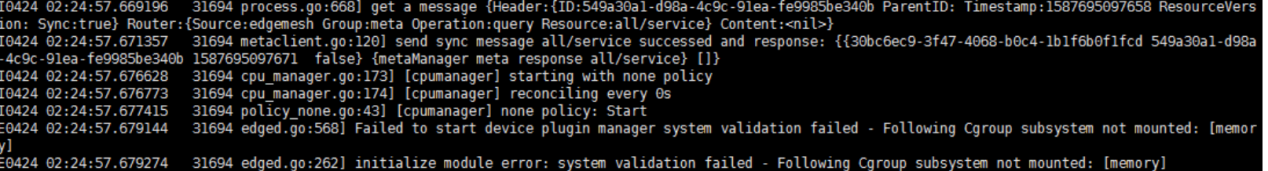
加入云端，但是因为github被墙了所以edgecore.service文件无法下载。解决方案自己创建该文件并修改其中的内容为网页中的内容。



再次用keadm申请加入云端。



使用systemctl status edgecore.service命令查看服务日志会发现服务没有跑起来，查看日志发现在启动初始化过程中失败了，原因是树莓派的Ubuntu20系统设置存在问题，这个问题广泛出现在linux内核版本5以上的树莓派系统中。



日志显示的是内存资源的cgroups机制没有开启。查看可用的cgroup:

cat /proc/cgroups

#subsys\_name hierarchy num\_cgroups enabled

cpuset 3 1 1

cpu 5 38 1

cpuacct 5 38 1

blkio 6 38 1

memory 0 53 0

devices 8 38 1

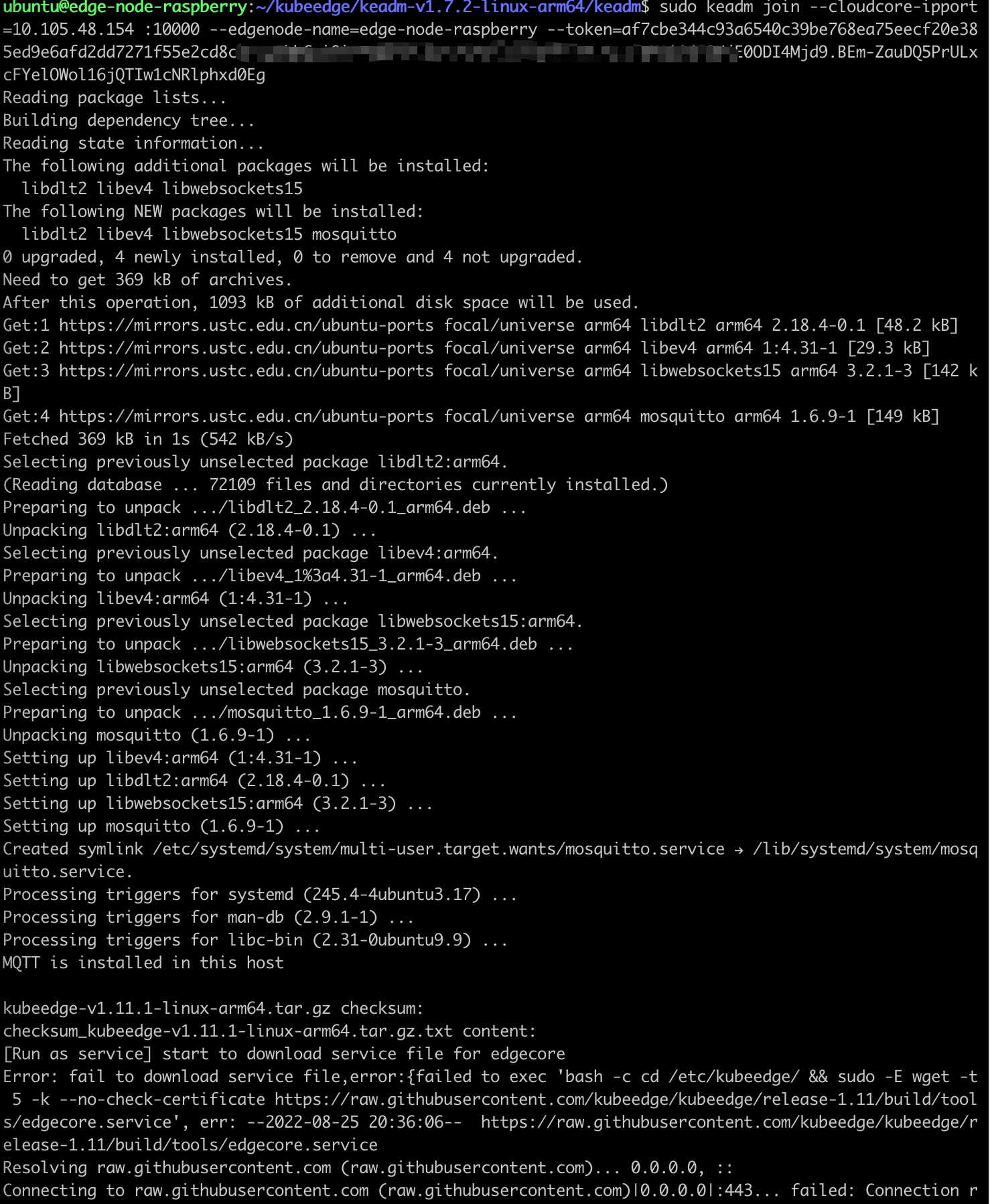
freezer 7 1 1

net\_cls 4 1 1

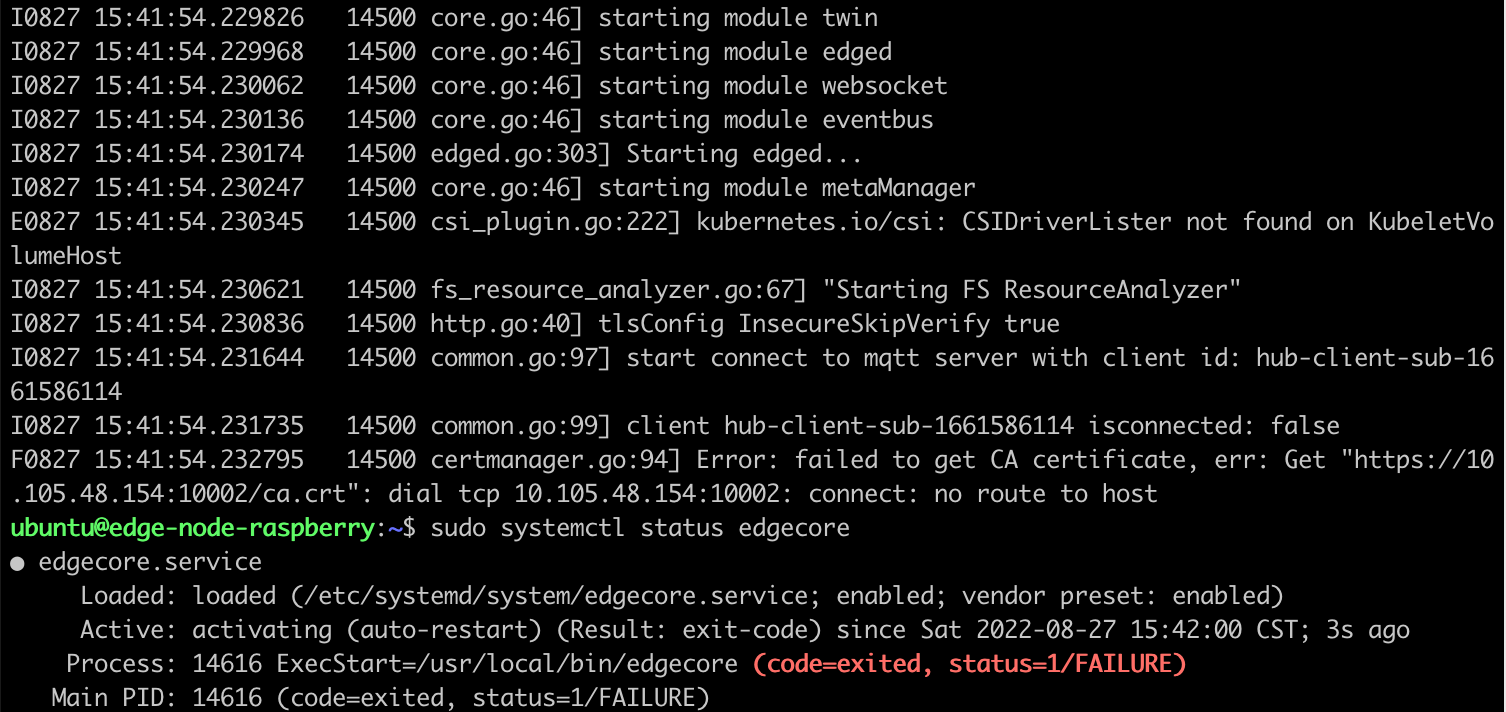
pids 2 43 1

通过修改启动文件增加cgroup的内存资源。





但是由于cloudcore在之江服务器，只有内网ip暂时无法访问加入集群。



# 分析

引入KubeEdge框架后，借助在Edge上运行的业务逻辑，可以在生成数据的本地保护和处理大量数据。这样可以减少边缘和云之间的网络带宽需求和消耗。提高响应速度，降低成本并保护隐私。

其次KubeEdge简化了开发，我们可以编写基于常规http或mqtt的应用程序，对其进行容器化，然后在Edge或Cloud中的任何位置运行它们中的更合适的一个。KubeEdge 是从Kubernetes改造而来，因此使用KubeEdge管理节点，就像以前Kubernetes管理节点一样，在Kubernetes基础不需要学习太多额外的知识。

# 人员分工及项目地址

本次任务由马昊良，刘建飞，李锐共同完成，分工如下：

马昊良（21023064）：并在云上搭建集群以及 KubeEdge 环境进行实验部署。

刘建飞（21023008）：在 Github 上新建项目，使用树莓派在已搭建的KubeEdege环境下进行应用演示。

李锐（21026089）：完成边缘计算研究现状分析和KubeEdge的原理，主要包括论文搜集、阅读并分析整理形成报告的第 1、2、4、5部分，并负责最终实验报告和相关材料的汇总整理。

项目地址 https://github.com/xxlbl/kubeedge-demo

参考文献

[1] LIU J，MAO Y Y，ZHANG J，et al.Delay-optimal computation task scheduling for mobile-edge computing system[C]//IEEE International Symposium on Information Theory，2016.

[2] YU R Z，XUE G L，ZHANG X.Application provision in fog computing- enabled internet- of-things：a network perspective[C]//IEEE INFOCOM 2018- IEEE Conference on Computer Communications，2018：783-791.

[3] XING H，LIU L，XU J，et al.Joint task assignment and wireless resource allocation for cooperative mobile-edge computing[C]//IEEE International Conference on Communications，2018.

[4] ZHANG K，MAO Y，LENG S，et al.Energy efficient offloading for mobile edge computing in 5G heterogeneous networks[J].IEEE Access，2016，4：5896-5907.

[5] WEN Y，ZHANG W，LUO H.Energy optimal mobile application execution：taming resource poor mobile devices with cloud clones[C]//IEEE INFOCOM，2012：2716-2720.

[6] MAO Y，ZHANG J，LETAIEF K B.Joint task offloading scheduling and transmit power allocation for mobile edge computing systems[C]//IEEE Wireless Communications and Networking Conference，2017：1-6.

[7] MAHMOODI S E，UMA R N，SUBBALAKSHMI K P. Optimal joint scheduling and cloud offloading for mobile applications[J].IEEE Trans on Cloud Computing，2019，7：301-313.

[8] H. Ma, Z. Zhou, and X. Chen, “Leveraging the power of prediction: Predictive service placement for latency-sensitive mobile edge computing,” IEEE Trans. Wireless Commun., vol. 19, no. 10, pp. 6454–6468, Oct. 2020.

[9] A. Ksentini, T. Taleb, and M. Chen, “A markov decision process-based service migration procedure for follow me cloud,” in Proc. IEEE Int. Conf. Commun. (ICC), Sydney, NSW, Australia, Jun. 10-14, 2014, pp. 1350–1354.

[10] Y. Sun, S. Zhou, and J. Xu, “EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks,” IEEE J. Sel. Areas Commun., vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[11] J. Wang, J. Hu, and G. Min, “Online service migration in edge computing with incomplete information: A deep recurrent actor-critic method.” [Online]. Available: https://arxiv.org/pdf/2012.08679.pdf

[12] T. Ouyang, Z. Zhou, and X. Chen, “Follow me at the edge: Mobilityaware dynamic service placement for mobile edge computing,” IEEE J. Sel. Areas Commun., vol. 36, no. 10, pp. 2333–2345, Oct. 2018.

[13] L. I. N. Kaiqun and C. T. C. F. Branch, ‘‘Pilot study on application of MEC local shunting service,’’ Modern Inf. Technol., vol. 1, no. 3, pp. 65–67, 2017.

[14] F. Lu, J. Hu, L. T. Yang, Z. Tang, P. Li, Z. Shi, and H. Jin, ‘‘Energyefficient traffic offloading for mobile users in two-tier heterogeneous wireless networks,’’ Future Gener. Comput. Syst., vol. 105, pp. 855–863, Apr. 2020.

[15] R. Wang, M. Li, L. Peng, Y. Hu, M. M. Hassan, and A. Alelaiwi, ‘‘Cognitive multi-agent empowering mobile edge computing for resource caching and collaboration,’’ Future Gener. Comput. Syst., vol. 102, pp. 66–74, Jan. 2020.

[16] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Ranjan, ‘‘Optimal decision making for big data processing at edge-cloud environment: An SDN perspective,’’ IEEE Trans. Ind. Informat., vol. 14, no. 2, pp. 778–789, Feb. 2018.

[17] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for VM-based cloudlets in mobile computing,” IEEE Pervasive Comput., vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[18] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara, “Cloudpath: A multi-tier cloud computing framework,” in Proc. 2nd ACM/IEEE Symp. Edge Comput., 2017, p. 20.

[19] P. Liu, D. Willis, and S. Banerjee, “ParaDrop: Enabling lightweight multi-tenancy at the network’s extreme edge,” in Proc. IEEE/ACM Symp. Edge Comput. (SEC), Oct. 2016, pp. 1–13.

[20] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, “SpanEdge: Towards unifying stream processing over central and near-the-edge data centers,” in Proc. IEEE/ACM Symp. Edge Comput. (SEC), Oct. 2016, pp. 168–178.

[21] (2018). Akraino Edge Stack. [Online]. Available: https://www.akraino.org

[22] (2018). Cord. [Online]. Available: <https://www.opennetworking.org/cord>

参考资料

* [https://www.yuque.com/leifengyang/oncloud/ghnb83#6lMCM](https://www.yuque.com/leifengyang/oncloud/ghnb83" \l "6lMCM)
* <https://gitee.com/liu_hu_wei/examples>
* <https://www.cnblogs.com/ltaodream/p/15135365.html>
* <https://blog.csdn.net/weixin_38159695/article/details/118486461>
* <https://www.cnblogs.com/cptao/p/10912709.html>
* <https://blog.csdn.net/yinjl123456/article/details/119300034>
* <https://github.com/kubeedge>
* <https://github.com/kubeedge/kubeedge/releases/>