基于KubeEdge边缘计算框架的相关研究

杜 易（22020093）

王劲夫（22020056）

刘兴鹏（22020073）

郎清杰（22020136）

2023年9月1日

一、引言

随着物联网的发展，联网的智能设备数量不断增长，每时每刻都在产生海量数据。实际应用环境的变化给云计算提出了新的要求：（1）低延时要求，AR/VR的时延要求是毫秒级，工业控制的时延更是在微秒级；（2）高可靠性，很多应用场景要求具有>99.999%的可用性，响应时间可预测，响应结果可重复等；（3）本地自治，要求边缘侧可适应偶尔断网，或者直接本地自治；（4）解决海量数据和有限带宽的矛盾，设备侧将产生海量数据，而以目前的带宽还无法承载这数据量。另外，很多数据没有全局价值，没有必要浪费带宽上传到云端；（5）信息安全，考虑到商业密码和个人隐私，很多机构和个人并不愿意把数据传输到云端。

传统云计算方法（即中心云）无法很好地解决以上问题，于是边缘计算技术出现了。边缘计算是一种新的计算范式，在网络边缘执行计算。它强调的是接近用户，接近信息源。通过在靠近物或数据源头的网络边缘侧，为应用提供融合计算、存储和网络等资源。同时，边缘计算也是一种使能技术，通过在网络边缘侧提供这些资源，满足行业在敏捷联接、实时业务、数据优化、应用智能、安全与隐私保护等方面的关键需求。边缘计算模型将原有云计算中心的部分或全部计算任务迁移到数据源附近，相比于传统的云计算模型，边缘计算模型具有实时数据处理和分析、安全性高、隐私保护、可扩展性强、位置感知以及低流量的优势。

二、相关研究

为了进一步理解边缘计算的原理、概念、存在问题和发展趋势，我们阅读了一些相关论文，下面对其中三篇进行介绍：

在施巍松等撰写的《边缘计算：现状与展望》一文中，介绍了边缘计算模型具有３个明显的优点：１）在网络边缘处理大量临时数据，不再全部上传云端，这极大地减轻了网络带宽和数据中心功耗的压力；２）在靠近数据生产者处做数据处理，不需要通过网络请求云计算中心的响应，大大减少了系统延迟，增强了服务响应能力；３）边缘计算将用户隐私数据不再上传，而是存储在网络边缘设备上，减少了网络数据泄露的风险，保护了用户数据安全和隐私。把边缘计算的发展分为三个阶段：技术储备期、快速增长期和稳定发展期。文中认为推动边缘计算迅速发展的７项关键技术为：网络、隔离技术、体系结构、边缘操作系统、算法执行框架、数据处理平台以及安全和隐私。文中认为未来几年迫切需要解决的问题：编程模型、软硬件选型、基准程序与标准、动态调度、与垂直行业的紧密结合以及边缘节点的落地。

在Fang Liu等的文章《A Survey on Edge Computing Systems and Tools》中，作者将现有边缘计算模型分为三类，分别是：从云端推送；从云端拉到边缘；混合云边缘。文中认为，现在需要一套全面和协调的基础开源系统/工具来加速物联网和边缘计算解决方案的部署。作者还综述了当前边缘计算系统所采用的能效增强机制，并从应用的角度对边缘计算系统中的新兴技术进行了研究，尤其介绍了基于深度学习的人工智能技术与边缘计算的结合可能性和最近进展。文中认为，在物联网和5G的背景下，边缘计算的愿景在提供更智能的服务和具有更好用户体验的应用方面很有前途。最近提出的关于边缘计算的系统和工具通常减少了数据处理和传输的开销，提高了移动数据分析的效率和功效。同时，边缘计算和深度学习技术的融合进一步促进了基于边缘的智能服务的研究。

在Xiaofei Wang等的文章《Convergence of Edge Computing and Deep Learning: A Comprehensive Survey》中，主要介绍了深度学习的几种网络模型，边缘计算的基础知识，以及二者的结合，特别是介绍了如何利用深度学习来发展边缘计算，如何用边缘计算发展深度学习,怎么在边缘计算的框架下进行深度学习的训练，深度学习在边缘上运行的预测结果怎么样，二者怎么更好融合形成智慧边缘（Intelligent Edge）和边缘智慧(Edge Intelligent)。文中认为，传统的深度学习训练过程需要把数据传到云端进行集中式的训练，会占据网络带宽资源，并且有效率和延时问题。深度学习结合边缘计算，在数据源附近的网络边缘提供深度学习服务，可以解决这样的问题。把深度学习集成到边缘计算的框架，动态地，自适应的进行边缘的管理和维护，是未来的发展方向。文中提出了五个关键技术方向：综合使用边缘计算和深度学习,提供智能服务；深度学习在边缘计算的推理预测；为了发展深度学习的边缘计算,利用边缘计算思想，技术等来服务于深度学习的发展；在边缘计算端使用深度学习来训练模型，以完成特定的任务；使用深度学习技术，更大地促进边缘算力的充分使用。并对深度学习和边缘计算如何融合提出了多个具体框架。

从论文调研的情况看，现在边缘计算正在向实际应用方面发展，尤其是与物联网、5G和深度学习等前沿技术的结合，起到了互相推动的作用。

三、不同边缘计算框架的特征分析以及优劣势

经过调研，发现主流的边缘计算框架主要有Baetyl、K3s、KubeEdge、OpenYurt、SuperEdge等。经过易用性、成熟程度和硬件需求等方面综合考虑，我们决定选用KubeEdge框架进行实验。

不同的边缘计算框架的特点及其区别如下：

**（1）Kubeedge**

Kubeedge 是首个基于K8s扩展的，提供云边协同能力的开放式边缘计算框架， 作为边缘计算框架的先驱者，其也是在CNCF中在该领域首个开源的项目。其重点解决了当前的几个问题：云边协同、资源异构、大规模、轻量化、一致的设备管理和接入体验。

在项目中我们经常能看见“docker+kubeedge”的组合在DevOps中发挥高效率，当前许多的k8s运行在边缘。

Kubeedge的优势在于：比较适合部署边缘较复杂的应用;Kubeedge的轻量化+可移植性非常适合作为边缘计算的场景，而且Kubeedge具有良好的扩展性；当前围绕Kubeedge已经形成了一个强大的云原生技术生态圈。

Kubeedge的缺点在于：其主要是为云数据中心设计的，如果需要在边缘进行部署该类应用，则需要面对一下几项挑战：当前的边缘端的ARM处理器具有低功耗等特点且使用广泛，但是当前k8s并不支持ARM架构;许多边缘设别的资源有限，无法部署完整的k8s。

**（2）Superedge**

Superedge的架构设计比较简洁，是对K8s的一种增强，在其云端增加了Application-Grid Controller, Edge-Health Admission等组件。在边缘端增加了Lite-Apsiserver等组件。其功能和OpenYurt基本一致，该项目相对于OpenYurt增加了边缘计算

**(3) OpenYurt**

OpenYurt 开源通过命令将Kubeedge转换为OpenYurt项目。且OpenYurt完整的保留了Kubelet等重新开发的一些轻量化组件。实现了Pod, Volume, Node等K8s资源对象的生命周期管理。所以OpenYurt对边缘的资源要求较高，这也加剧了其与低资源的边缘节点的矛盾，但是对于当前的手机以及树莓派都是开源满足该要求的。OpenYurt相对于KubeEdge 跟随 Kubernetes 版本升级零负担，OpenYurt相比Kueedge扩展出了更多的能力。

**(4) K3s**

该项目是轻量化的k8s, 开源不需要中心云，完全部署在边缘节点。和OpenYurt， Kubeedge相比缺了边缘计算的云边协同，边缘自治等可行， 其主要的目的是轻量化，便于在边缘节点进行部署。该项目意在用于在资源较少的设备边缘场景。但是当前在其上叠加了一层集群管理方案，该方案当前尚不成熟。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 项目 | Kubeedge | OpenYurt | SuperEdge | K3s |
| 边缘自治 | 有 | 有 | 有 | 无 |
| 部署复杂度 | 简单 | 简单 | 简单 | 简单 |
| 轻量化 | 是 | 否 | 否 | 否 |
| 生态兼容 | 部分兼容 | 完整兼容 | 完整兼容 | 部分兼容 |
| 去中心化 | 是 | 是 | 是 | 否 |
| 边缘单元化 | 无 | 是 | 是 | 无 |
| 原生支持k8s | 是 | 是 | 是 | 是 |
| 是否支持MQTT | 支持 | 支持 | 支持 | 支持 |
| 容器化编排 | 支持 | 支持 | 支持 | 支持 |

四、KubeEdg**e框架原理分析**

本实验，我们采用Kubeedge作为实验框架，下面对该框架进行介绍：

**1.简介**

KubeEdge框架是由华为公司于2018年11月15日宣布开源，是一个Kubernetes Native的边缘计算管理框架，其设计初衷是：云边协同，计算下沉，让云端更加容易地管理边缘节点和设备。可将本机容器化应用编排和管理扩展到边缘端设备。它构建在Kubernetes之上，为网络和应用程序提供核心基础架构支持，并在云端和边缘端部署应用，同步元数据。Kubernets 是以应用为中心的技术架构，以一套技术体系支持任意负载，运行于任意基础设施之上，向下屏蔽基础设施差异，实现底层资源的统一调度，向上通过容器经过向标准化应用，实现负载自动化部署，向外突破中心云计算的边界，将云计算能力拓展到边缘以及现场，构建云边一体化基础设施。KubeEdge的主要改进是把 Kubernetes 从云端延伸到边缘。

KubeEdge的优势有：

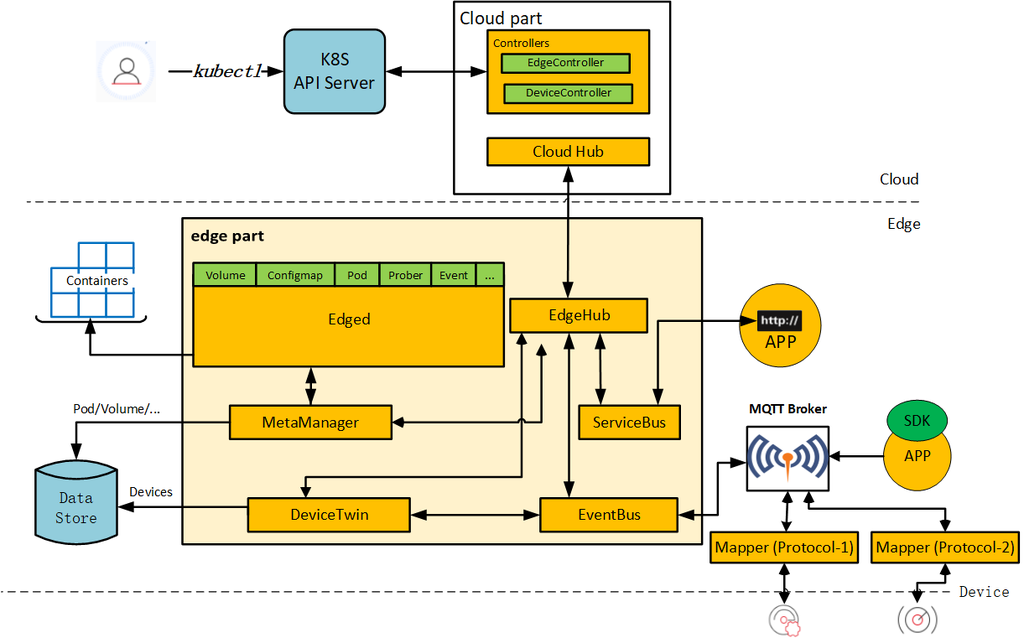
边缘计算 - 通过在Edge上运行的业务逻辑，可以在生成数据的本地保护和处理大量数据。这减少了网络带宽需求以及边缘和云之间的消耗。这样可以提高响应速度，降低成本并保护客户的数据隐私；

简化开发 - 开发人员可以编写基于常规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；

Kubernetes原生支持 - 借助KubeEdge，用户可以在Edge节点上编排应用，管理设备并监视应用和设备状态，就像云中的传统Kubernetes集群一样；

大量的应用 - 可以轻松地将现有的复杂机器学习，图像识别，事件处理和其他高级应用程序部署和部署到Edge。

**2.架构**



上图为KubeEdge架构示意图，由图可以看出该框架由以下部分构成:

**2.1云边通信**

CloudHub:一个 Web Socket 服务端，用于大量的 edge 端基于 websocket 或者 quic 协议连接上来。负责监听云端的变化, 缓存并发送消息到 EdgeHub。

EdgeHub:一个 Web Socket 客户端，负责将接收到的信息转发到各edge端的模块处理；同时将来自个edge端模块的消息通过隧道发送到cloud端。提供可靠和高效的云边信息同步。

**2.2云上部分（cloud part）**

EdgeController: 用于控制 Kubernetes API Server 与边缘的节点、应用和配置的状态同步。

DeviceController:一个扩展的 Kubernetes 控制器，管理边缘设备，确保设备信息、设备状态的云边同步。

**2.3边缘部分（edge part）**

MetaManager:该模块后端对应一个本地的数据库（sqlLite），所有其他模块需要与 cloud 端通信的内容都会被保存到本地 DB 种一份，当需要查询数据时，如果本地 DB 中存在该数据，就会从本地获取，这样就避免了与 cloud 端之间频繁的网络交互；同时，在网络中断的情况下，本地的缓存的数据也能够保障其稳定运行，在通信恢复之后，重新同步数据，是边缘节点自治能力的关键。

Edged: 是运行在边缘节点的代理，用于管理容器化的应用程序。算是个重新开发的轻量化 Kubelet，实现 Pod，Volume，Node 等 Kubernetes 资源对象的生命周期管理。

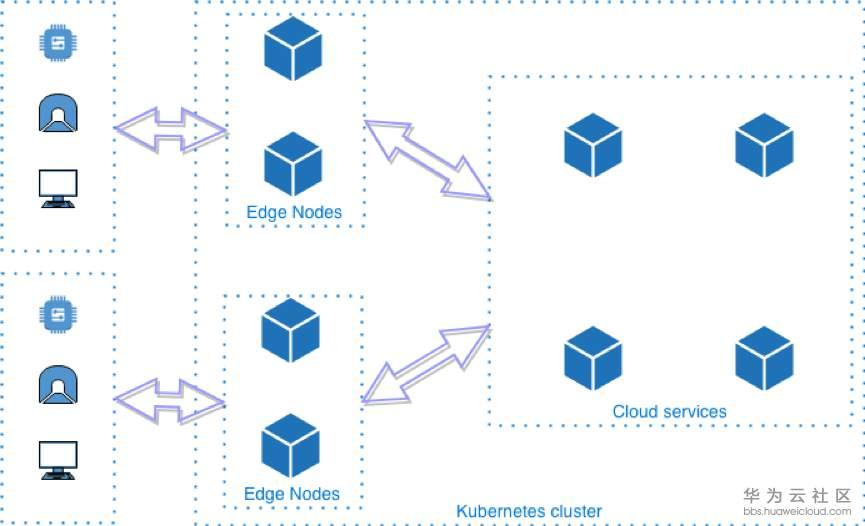
EventBus:一个与 MQTT 服务器（mosquitto）交互的 MQTT 客户端，为其他组件提供订阅和发布功能。

ServiceBus:一个运行在边缘的HTTP客户端，接受来自云上服务的请求，与运行在边缘端的HTTP服务器交互，提供了云上服务通过HTTP协议访问边缘端HTTP服务器的能力。

DeviceTwin:负责存储设备状态并将设备状态同步到云，它还为应用程序提供查询接口。

**3.部署模型**

KubeEdge 遵循的是以下部署模型:



KubeEdge 是一种完全去中心化的部署模式，管理面部署在云端，边缘节点无需太多资源就能运行 Kubernetes 的 agent，云边通过消息协同。从 Kubernetes 的角度看，边缘节点 + 云端才是一个完整的 Kubernetes 集群。这种部署模型能够同时满足“设备边缘”和“基础设施边缘”场景的部署要求。

五、实验

**1.KubeEdge环境部署**

**1.1系统及软件版本**

·3台ubuntu16.04分别作为k8s集群主节点即云端节点（ubuntu）、集群从节点（node1）和边缘端节点（edge1）

·Docker version 20.10.7

·Kubernetes v1.19.0

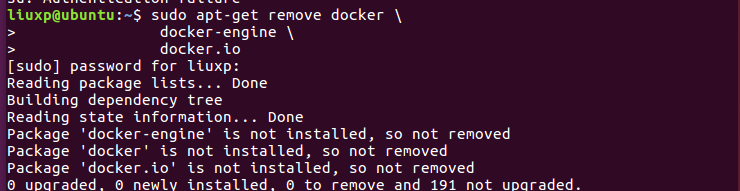
·go version go1.15 linux/amd64

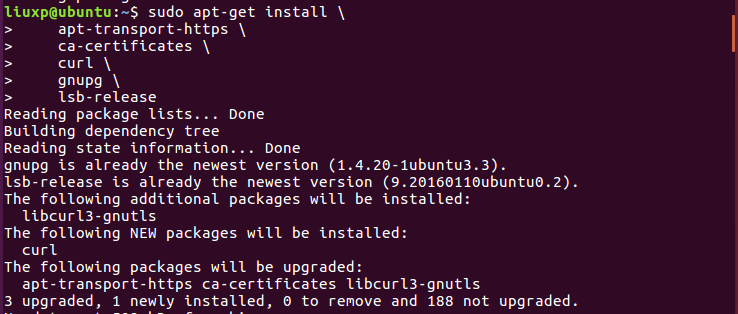
·kubeedge v1.10.2 linux/amd64

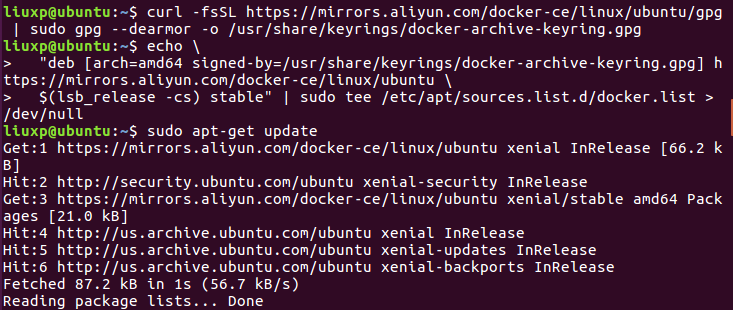
**1.2 k8s安装部署**

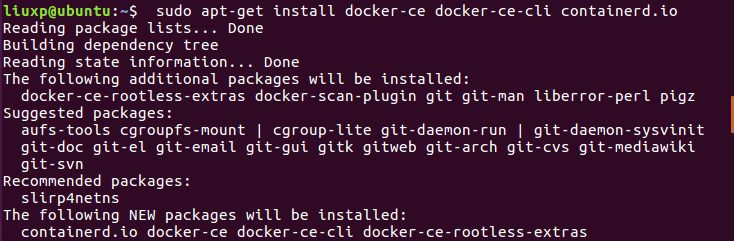
**1.2.1安装docker**

卸载旧版本，旧版本的 Docker 称为 docker 或者 docker-engine，使用以下命令卸载旧版本：

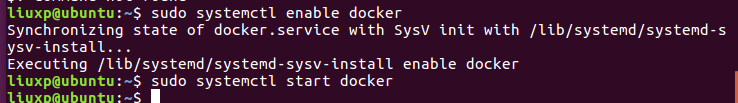


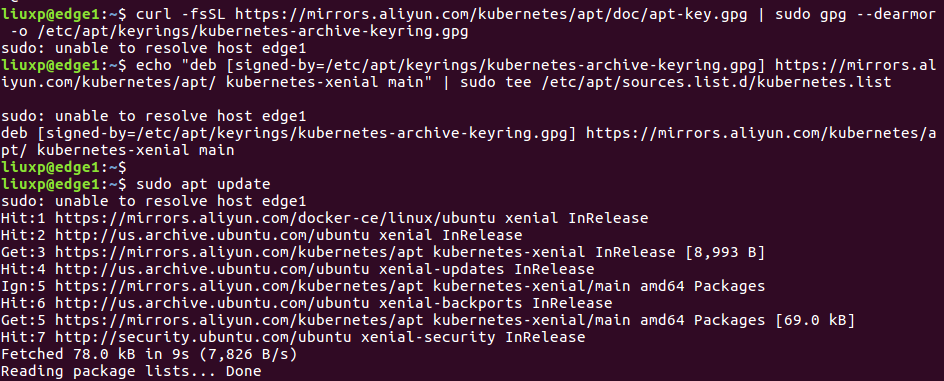
由于 apt 源使用 HTTPS 以确保软件下载过程中不被篡改。因此，我们首先需要添加使用 HTTPS 传输的软件包以及 CA 证书：

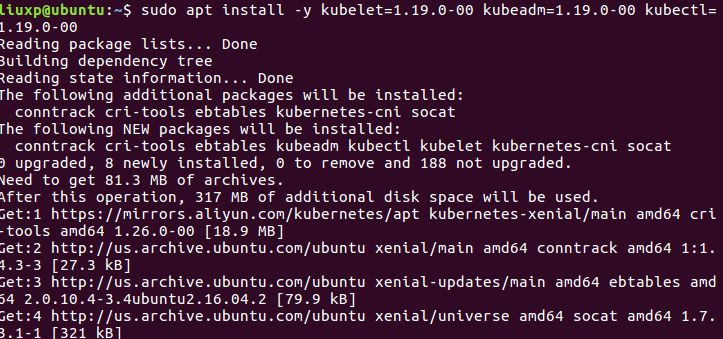
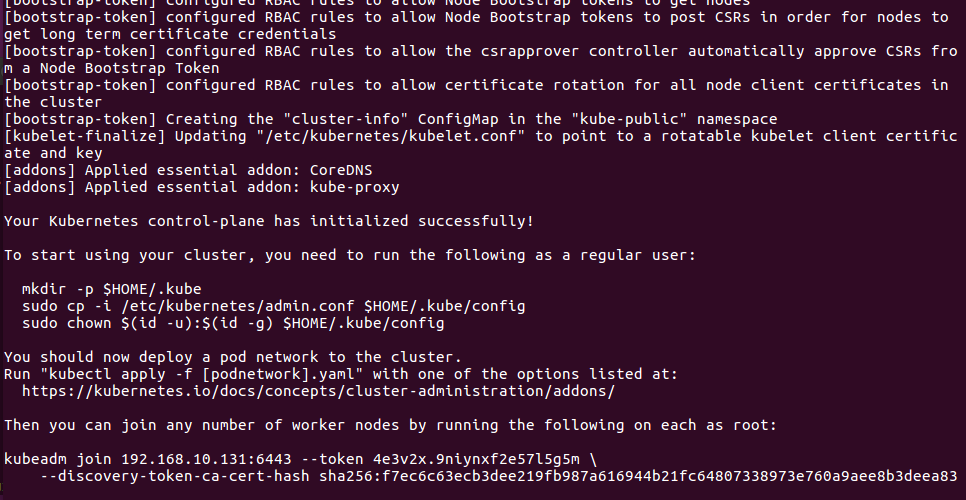
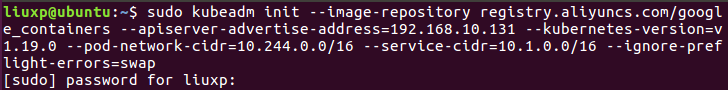
更换及添加国内源：

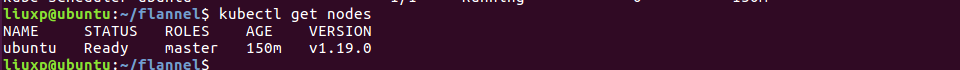
安装docker：

启动docker：

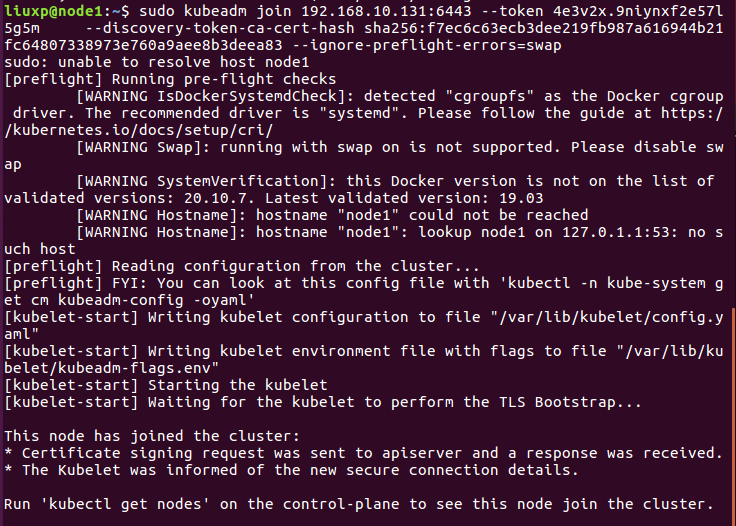
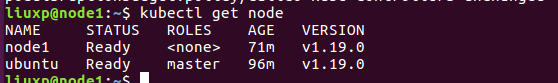
**1.2.2安装kubernetes**

下载[阿里云镜像](https://so.csdn.net/so/search?q=%E9%98%BF%E9%87%8C%E4%BA%91%E9%95%9C%E5%83%8F&spm=1001.2101.3001.7020" \t "_blank) Google Cloud 公开签名秘钥并添加Kubernets apt仓库：

安装 kubelet、kubeadm 和 [kubectl](https://so.csdn.net/so/search?q=kubectl&spm=1001.2101.3001.7020" \t "_blank)，并锁定其版本： 初始化k8s集群Master主节点：

安装flannel网络插件并查看节点状态为Ready：

**1.2.3从节点部署并添加至k8s集群**

同样方法安装docker和kubernetes然后执行添加命令： 查看节点状态：

显示从节点添加成功

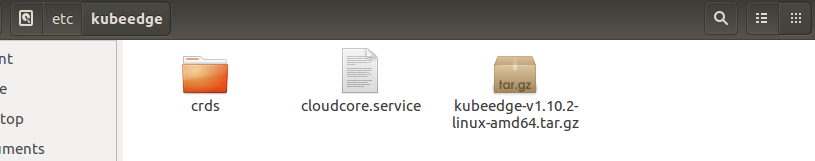
**1.3Kubeedge安装部署**

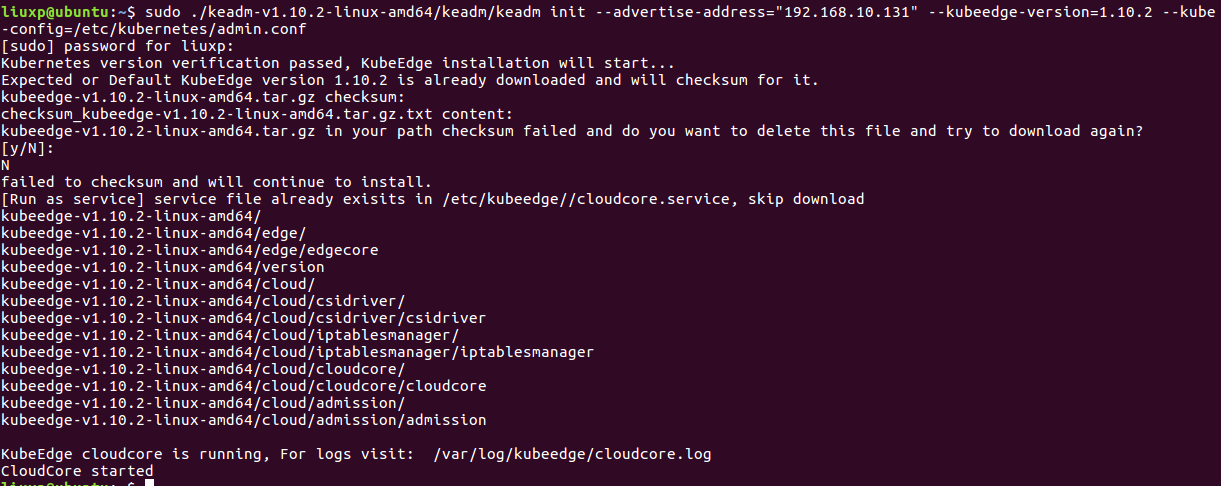
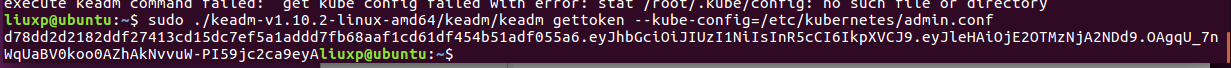
Kubeedge由云和边缘组成。

**1.3.1云端部署**

使用前面部署的master主机作为云端，使用kubeedge提供的keadm工具，版本为v1.10.2-linux/amd64。

手动下载好需要的文件并放至/etc/kubeedge/下：

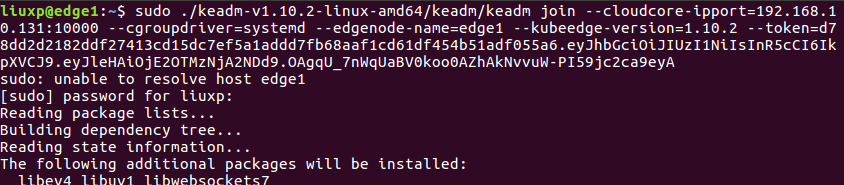
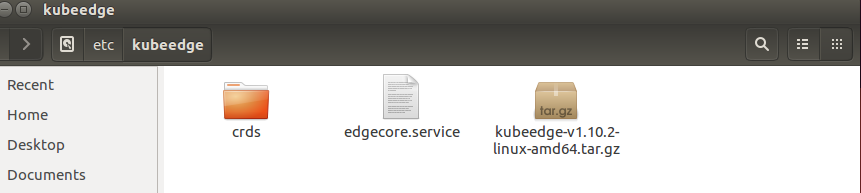


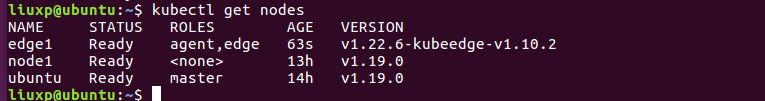
使用keadm工具初始化kubeedge的cloudcore，--advertise-address="192.168.10.131"为masterIP。 获取kubeedge的token

**1.3.2边缘端部署**

使用第三台ubuntu16.04，以相同方法配置docker、k8s以及keadm工具。

手动下载好需要的文件并放至/etc/kubeedge/下：

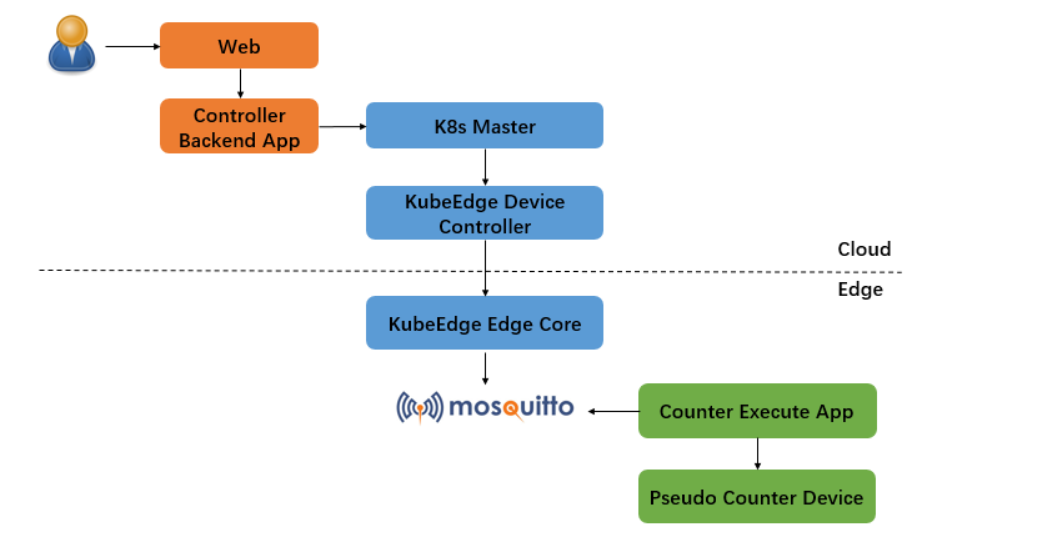
使用cloudcore获取的token将edgecore节点加入到集群中：

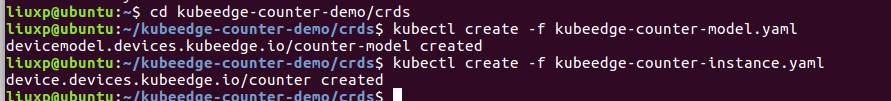
查看节点状态：

显示edge1加入成功。至此，Kubeedge部署成功。

**2.基于KubeEdge的Counter运行实例**

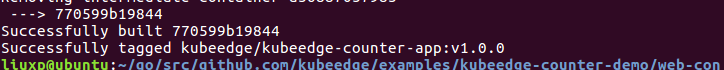
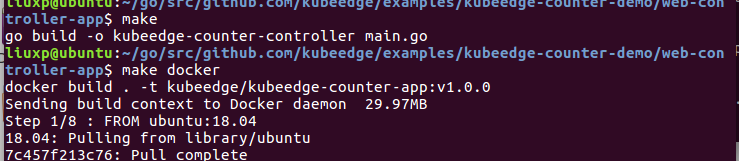
Counter是一个伪设备，用户无需任何额外的物理设备即可运行此演示。计数器在边缘端运行，用户可以从云端通过网络对其进行控制，也可以从云端通过网络获取计数器的值。原理图如下：

**2.1为计数器创建设备模型和设备实例**

现在在群集中安装Device CRD API，使用 yaml 文件为计数器创建设备模型和实例：

**2.2部署云端应用即运行Kubeedge Web App**

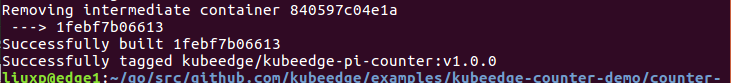
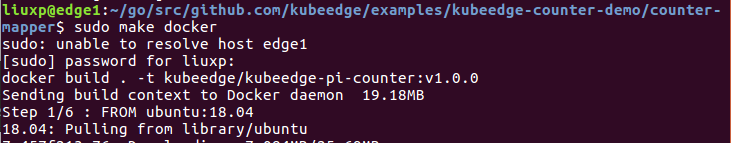
云端应用web-controller-app用来控制边缘端的pi-counter-app应用，该程序默认监听的端口号为80，此处修改为8089。构建镜像：



查看节点部署web-controller-app：

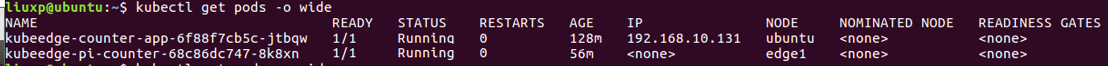
**2.3部署边缘端应用即运行Kubeedge Pi Counter App**

边缘端的pi-counter-app应用受云端应用控制，主要与mqtt服务器通信，进行简单的计数功能。构建镜像：



部署Pi Counter App：

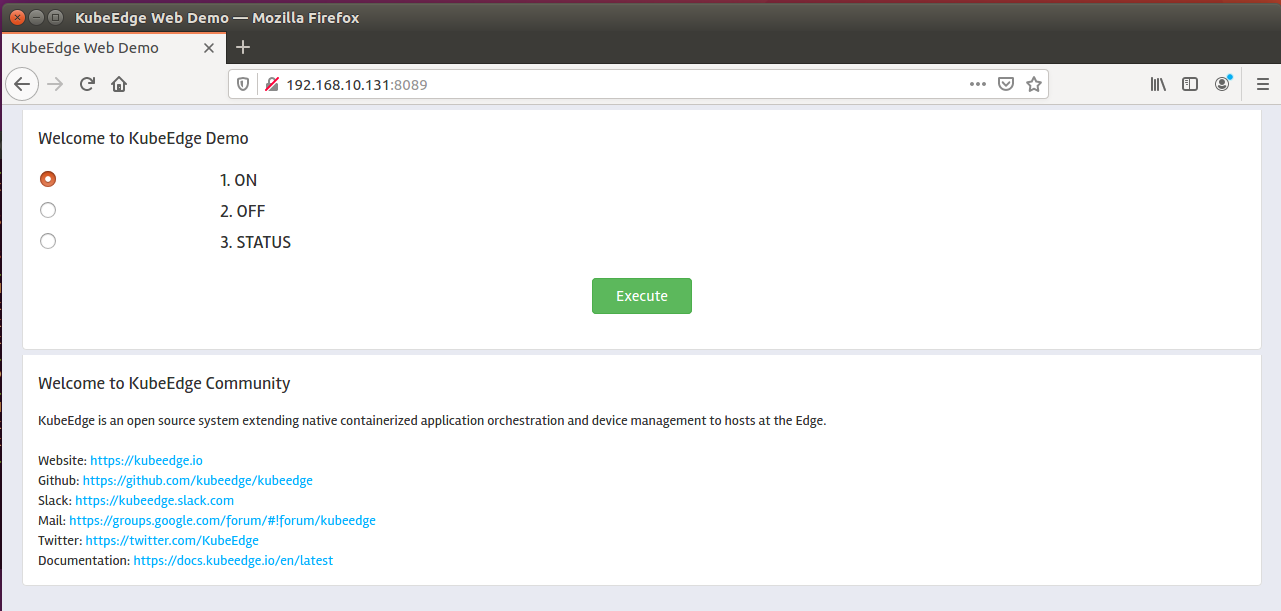
查看pod状态：



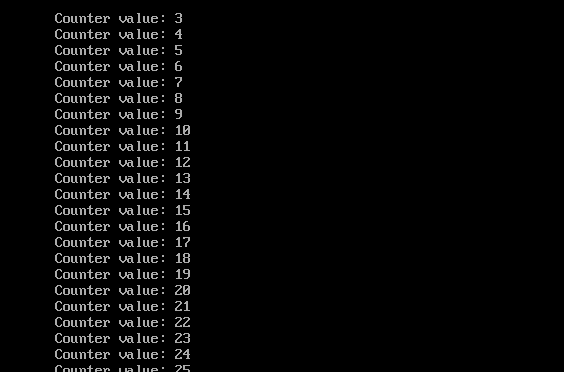
至此，Counter运行实例的云端部分和边缘端的部分都已经部署完毕。

**3.3实例运行**

浏览器访问192.168.10.131:8089进入web界面：

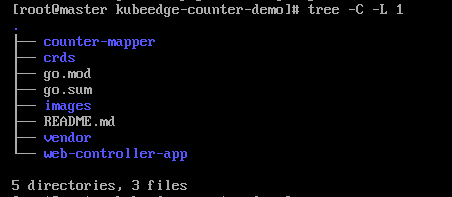


选择ON，并点击Execute，可以在edge边缘节点上查看执行结果。



**3.4 counter代码解析**

Couner的一级目录如图所示，couunter-mapper对应边缘设备；crds对应部署时使用yaml文件；web-controller-app(云端控制)。



Web-controller-app中的main.go文件使用了Beego这样的一个应用开发框架，使用beego.Router()来注册路由，改方法会调用自定义的Index函数。 在Index()函数中对主要是html文件，用于初始化网页View(即加载模板Layout, 个性化页面Tplname)

|  |
| --- |
| func main() {  beego.Router("/", new(controllers.TrackController), "get:Index")  beego.Router("/track/control/:trackId", new(controllers.TrackController), "get,post:ControlTrack")  beego.Run(":80")  } |

第二个beego.Router()函数则会注册trackID并应映射到TrackController处理器，在收到Get或者Post请求时，则会执行ControlTrack()函数。ControlTrack()函数主要用于获取输入Key为：TrackID的值并赋值道道params结构体的TrackID字段。这里的TrackID一共有三个值，分别为ON,OFF,STATUS.如果TrackID为ON或者OFF,则会执行UpdateDeviceTwinWithDesiredTrack函数来对result结果进行修改。如果是STATUS,则会执行UpdateStatus函数。Get的方式获取当前的状态，将每个Twin内的Desired与reported存储并返回。

Init()函数，该函数会在执行main()函数之前执行。主要用于初始化CRDClient从而可以与K8s的API进行一些列的通讯。在init()函数中首先创建KubeConfig(). 在KubeConfig()中会对KubeQPS, KubeBurst等进行配置和初始化。

|  |
| --- |
| func init() {  kubeConfig, err := utils.KubeConfig()  if err != nil {log.Fatalf("Failed to create KubeConfig, error : %v", err)}  log.Println("Get kubeConfig successfully")  crdClient, err = utils.NewCRDClient(kubeConfig)  if err != nil {log.Fatalf("Failed to create device crd client , error : %v", err)}  log.Println("Get crdClient successfully")  } |

之后执行NewCRDClient创建restClient， 在其中会首先创建k8s资源对象的元素据Scheme，紧接着创建SchemeBuilder进行管理。之后使用schemeBuilder.Addtoscheme().之后创建config,设置APIPath,直到使用scheme创建一个新的CodecFactory, 配置部分即完成了。

五、总结

总的来说，KubeEdge是一个强大的边缘计算的开放平台。它通过基于kubernetes的容器编排功能、云和边缘之间的部署和元数据同步以及完全开源的模式，为边缘计算提供了可能性。在这个平台上，用户可以管理和调度在边缘运行的工作负载，将边缘设备连接到云端，实现远程控制和自动化控制。

未来，随着云计算和物联网技术的不断发展，我们相信KubeEdge等边缘计算框架将会在更多的领域发挥重要的作用有着广阔的前景和巨大的潜力等待我们去探索和挖掘。

# 人员分工及项目地址

本次任务由杜易，王劲夫，刘兴鹏，郎清杰共同完成，分工如下：

杜 易（22020093）：调研近年来边缘计算领域论文，阅读了若干篇综述了解当前边缘计算的发展前沿，撰写相关报告。

王劲夫（22020056）：比较百度Batyl, K3s, KubeEdge, OenYurt, SuperEdge框架的异同。详细分析本实验所用框架KubeEdge的原理，并撰写报告。

刘兴鹏（22020073）： 基于ubuntu系统，先配置了K8s环境，然后再在上面部署kubeedge相关内容，并解决一些列环境不适配问题，完成环境部署。

郎清杰（22020136）：分析分布式应用counter的代码，并在其上进行适配修改，达到在该Kubeedge集群中进行部署。

项目地址：

https://github.com/liuxingpeng520521/kubeedge\_demo\_succ