第3章

边缘计算实例开发应用

边缘计算平台应用实例

- EdgeX Foundry
- CORD
- AWS IoT Greengrass
- Azure IoT Edge
- Link IoT Edge
- KubeEdge
- Baetyl
- OpenYurt

边缘计算是一个分布式系统范例，通过数据传输路径上的计算、存储与网络资源为用户提供服务，由于这些资源数量众多且在空间上分散，边缘计算平台将对这些资源进行统一的控制与管理，使开发者可以快速地开发与部署应用，成为边缘计算的基础设施。在开发边缘计算平台时，如何实现各个边缘平台间相互协作，提高资源利用率，同时保证网络、数据、应用的安全，这些都是设计平台所考虑的重点问题。本章重点讨论各类边缘计算平台的基本架构、部署方式和应用场景，并给出已有的开源/商用平台实例展示。其中 3.1节介绍边缘计算平台的基本情况和差异分析，3.2节介绍 Linux 基金会、亚马逊、微软、阿里云、华为、百度等不同机构开发部署的 8 个典型的开源边缘计算平台应用案例，3.3节给出可供读者参考使用的虚拟机开发环境，并给出具体开发应用实例 KubeEdge 和 Azure IoT Edge 的搭建和部署细节。通过本章的学习，读者可以自己动手搭建一个边缘计算平台实例，并在该平台上部署不同的应用。

3.1 边缘计算平台

针对不同的问题及应用场景，边缘计算平台呈多样性发展，同时也具有一般性。图 3.1是边缘计算平台的通用功能框架，在该框架中，资源管理功能用于管理网络边缘的计算、网络和存储资源。设备接入和数据采集功能分别用于接入设备和从设备中获取数据。安全管理用于保障来自设备的数据的安全。平台管理功能用于管理设备和监测控制边缘计算应用的运行情况。

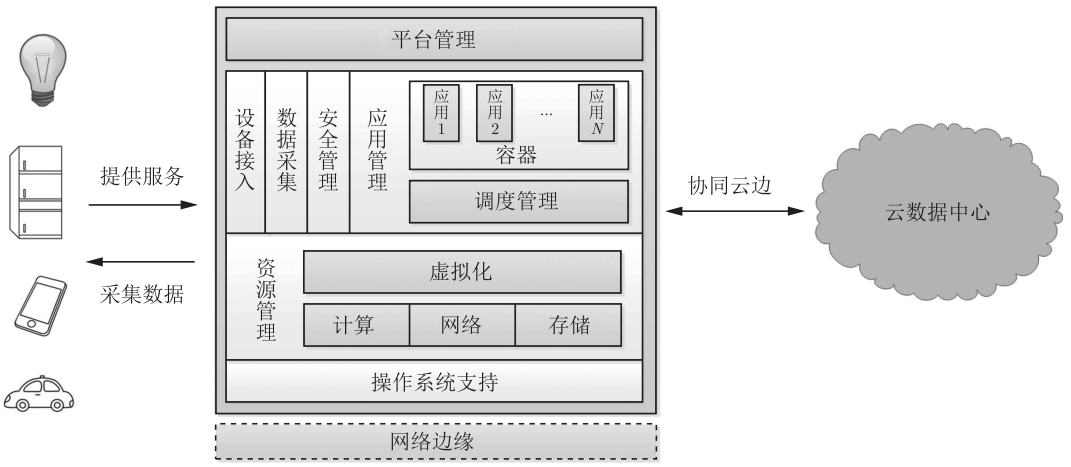

图 3.1　边缘计算平台的通用功能框架（见彩插）

各边缘计算平台的差异可从以下几方面进行对比和分析。

① 设计目标。边缘计算平台的设计目标反映了其针对解决的问题领域，并对平台的系统结构和功能设计有关键性的影响。

② 目标用户。在现有的各种边缘计算平台中，有部分平台是提供给网络运营商以部署边缘云服务，有的边缘计算平台则没有限制，普通用户可以自行在边缘设备上部署使用。

③ 可扩展性。为满足用户应用动态增加和删除的需求，边缘计算平台需要具有良好的可扩展性。目前，虚拟机技术和容器技术常被用于支持可扩展性。

④ 系统特点。面向不同应用领域的边缘计算开源平台具有不同的特点，而这些特点能为不同的边缘计算应用的开发或部署带来方便。

⑤ 应用场景。常见的应用领域包括智能交通、智能工厂和智能家居等多种场景，还有增强现实（AR）/虚拟现实（VR）应用、边缘视频处理和无人车等对响应时延敏感的应用场景。

3.2　开源项目案例

根据边缘计算平台的设计目标和部署方式，目前的边缘计算平台可分为三类：面向物联网端的边缘计算开源平台、面向边缘云服务的边缘计算开源平台和面向云边融合的边缘计算开源平台。面向物联网端的边缘计算开源平台（如 EdgeX Foundry）致力于解决在开发和部署物联网应用的过程中存在的问题；面向边缘云服务的边缘计算平台（如 CORD）着眼于优化或重建网络边缘的基础设施，以实现在网络边缘构建数据中心，并提供类似云中心的服务；面向云边融合的边缘计算开源平台（如 AWS IoT Greengrass、Azure IoT Edge 等）致力于将云服务能力拓展至网络边缘。其中，面向云边融合的边缘计算开源平台应用广泛，成为边缘计算平台的发展趋势，本节着重介绍这类边缘计算

平台。

3.2.1 开源框架 EdgeX Foundry

目前，具有大量设备的物联网产生大量数据，迫切需要结合边缘计算的应用，但物联网的软硬件和接入方式的多样性给数据接入功能带来困难，影响了边缘计算应用的部署。

EdgeX Foundry 是 Linux 基金会主持的一个开源项目，旨在为工业物联网边缘计算构建一个通用的开放框架，其核心是一个标准化互操作性框架。

该框架部署于路由器和交换机等边缘设备上，为各种传感器、设备或其他物联网器件提供即插即用功能并管理它们，进而收集和分析它们的数据，或者导出至边缘计算应用或云计算中心做进一步处理。EdgeX Foundry 针对的是物联网器件的互操作性问题。

EdgeX Foundry 体系架构如图 3.2所示。它由南北两侧、4 个服务层、贯穿整个架构的安全服务和设备，以及系统管理服务构成。

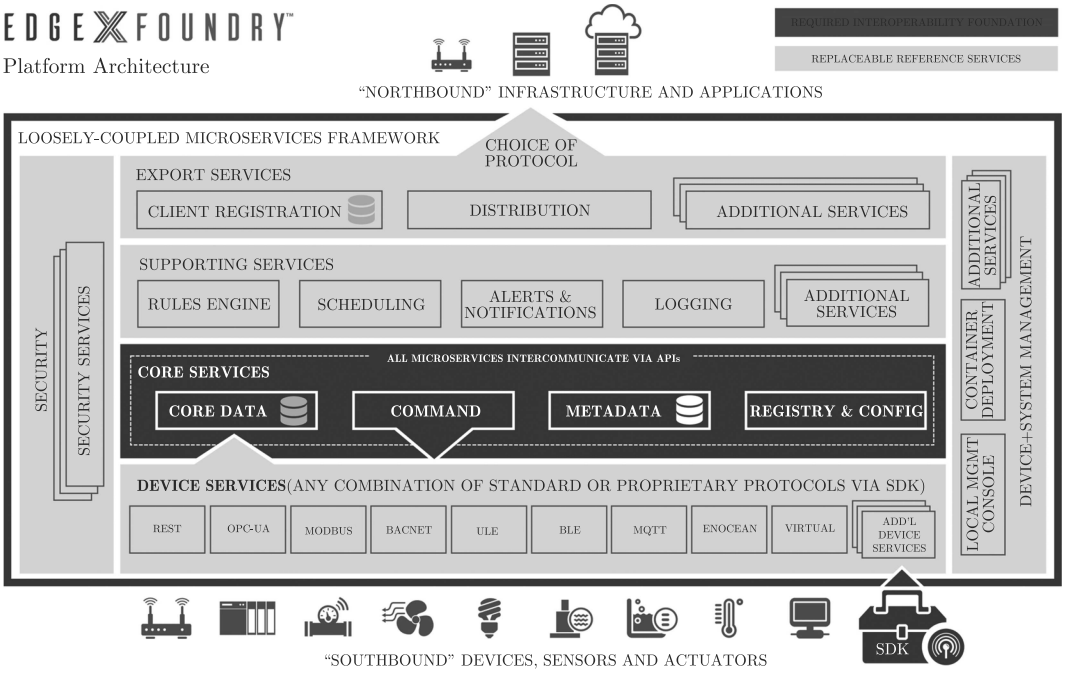

图 3.2 EdgeX Foundry 体系架构（见彩插）

- 南北侧
  - ♦ 南侧：指物理领域内的所有物联网对象，以及与这些设备、传感器或其他物联网器件对象直接通信并收集数据的网络边缘。
  - ♦ 北侧：指数据收集、存储、聚合、分析并转换为信息的云（或企业系统），以及与云通信的网络部分。

- 服务层
  - ♦ 设备服务层：与物理设备直接通信的具体微服务的集合，每个设备微服务可以管理支持对应接口的多个物理设备。
  - ♦ 核心服务层：由核心数据、命令、元数据、注册表和配置组成。
  - ♦ 支持服务层：提供日志、规则引擎、提醒等通用服务。
  - ♦ 输出服务层：将在边缘创建的数据传输到企业（云）系统。

EdgeX Foundry 的主旨是简化和标准化工业物联网边缘计算的架构，实现即插即用组件生态系统，从而加速物联网解决方案的部署。

官网：https://www.edgex-foundry.org。

GitHub：https://github.com/edgexfoundry。

3.2.2　边缘计算开源项目 CORD

CORD（Central Office Re-architected as a Data Center）是开放网络基金会（ONF）推出的开源项目，该项目利用软件定义网络（SDN）、网络功能虚拟化（NFV）和云计算技术重构现有的网络边缘基础设施，并将其打造成可灵活提供计算和网络服务的数据中心。

CORD 计划利用商用硬件和开源软件打造可扩展的边缘网络基础设施，集成多个开源项目为网络运营商提供了一个基于云的、可编程的开放平台，支持用户的自定义应用。

CORD 的硬件架构如图 3.3所示。CORD 利用商用服务器和白盒交换机提供计算、存储和网络资源，并将网络构建为叶脊拓扑架构以支持横向网络的通信带宽需求。此外，CORD 使用专用接入硬件将移动、企业和住宅用户接入网络中。

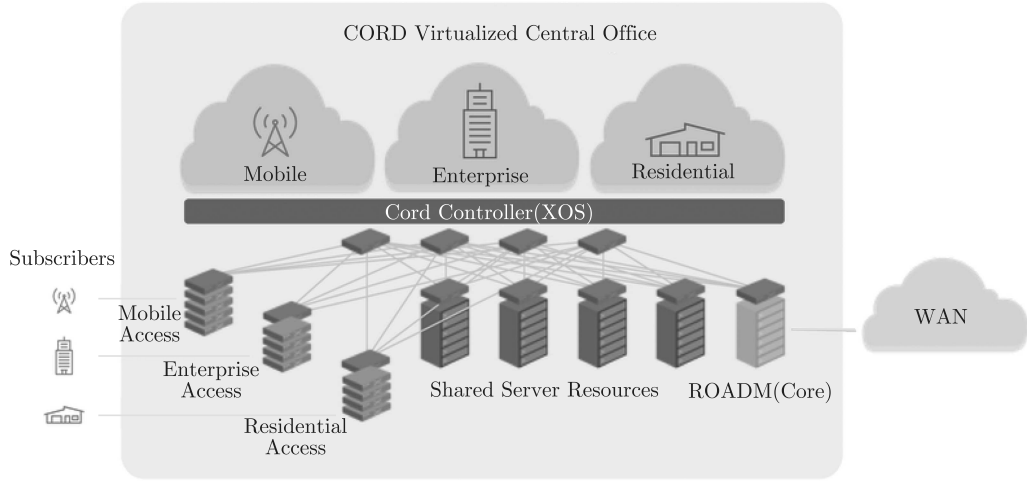

图 3.3　CORD 的硬件架构

从软件架构（图 3.4）上看，CORD 有 4 个子系统：平台、配置文件、工作流和 CI/CD 工具链。

- 平台：公共基础层包括 kubernetes（容器管理系统）和 ONOS（SDN 控制器），每个交换机上都加载 Stratum。
- 配置文件：选择在特定 POD 上运行的特定于部署的微服务和 SDN 控制应用程序集合。
- 工作流：在给定运营商的网络中运行 POD 所需的特定于部署的集成逻辑。
- CI/CD 工具链：用于组装、部署、操作和升级特定的平台/配置文件/工作流组合。

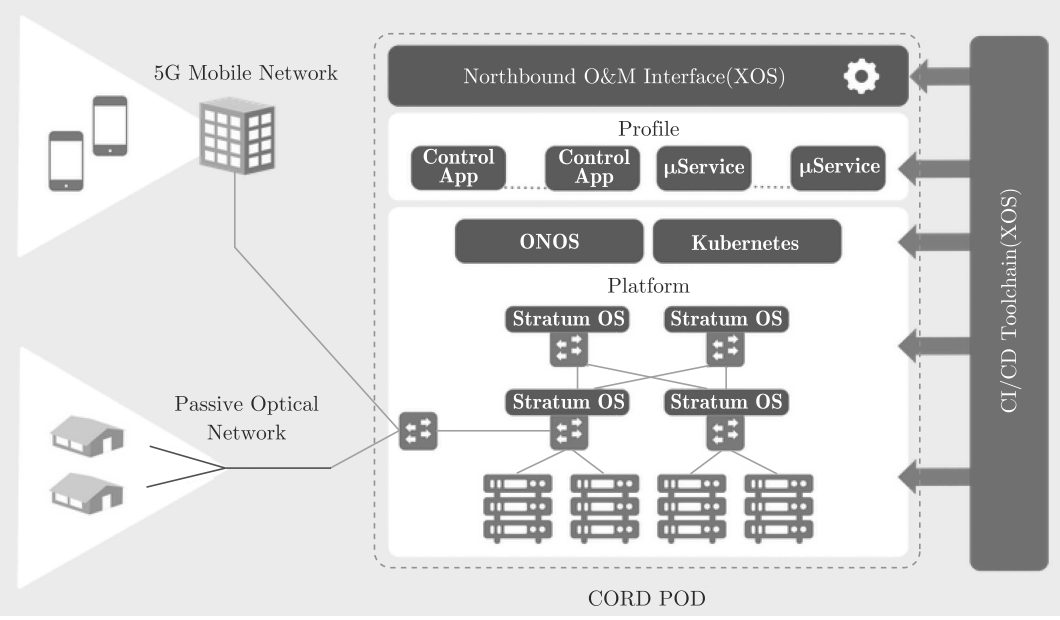

图 3.4 CORD 软件架构（见彩插）

根据用户类型和使用案例的不同，CORD 可被具体实现为 M-CORD、R-CORD 和 E-CORD。M-CORD 是为部署 5G 移动无线网络的运营商提供的开源参考解决方案，是一个基于 SDN、NFV 和云技术的云本地解决方案。R-CORD 是一个基于 CORD 平台的开源解决方案，用于提供超宽带住宅服务，它将运营商的网络边缘转化为一个灵活的服务交付平台，使运营商能够提供最佳的终端用户体验，以及创新的下一代服务。E-CORD 建立在 CORD 基础设施上，以支持企业客户，并允许服务提供商提供企业连接服务（L2 和 L3VPN）。

官网：https://opencord.org。

GitHub：https://github.com/opencord。

3.2.3 亚马逊边缘计算平台 AWS IoT Greengrass

亚马逊的边缘计算平台 AWS IoT Greengrass 是将云功能扩展到本地设备的软件。它使得设备能够收集和分析更靠近信息源的数据，自主应对本地事件，并在本地网络上相互安全地通信。本地设备也可以安全地与 AWS IoT Core 通信，并将物联网数据导出

到 AWS Cloud。AWS IoT Greengrass 开发人员可以使用 AWS Lambda 函数和预先构建的连接器创建可部署到设备中用于本地执行的无服务器应用程序。

AWS IoT Greengrass 主要由 Greengrass Core 和 IoT Device SDK 两部分组成。由于 AWS IoT Greengrass 提供了预构建的连接器，因此开发人员无须编写代码即可扩展边缘设备功能，同时能够快速连接到边缘的第三方应用程序、本地软件和 AWS 服务。AWS IoT Greengrass 为边缘设备提供信任私有密钥存储的硬件根，可以在使用 AWS IoT Greengrass 功能的同时使用硬件保护的消息加密功能。图 3.5所示为 AWS IoT Greengrass 的基本架构。

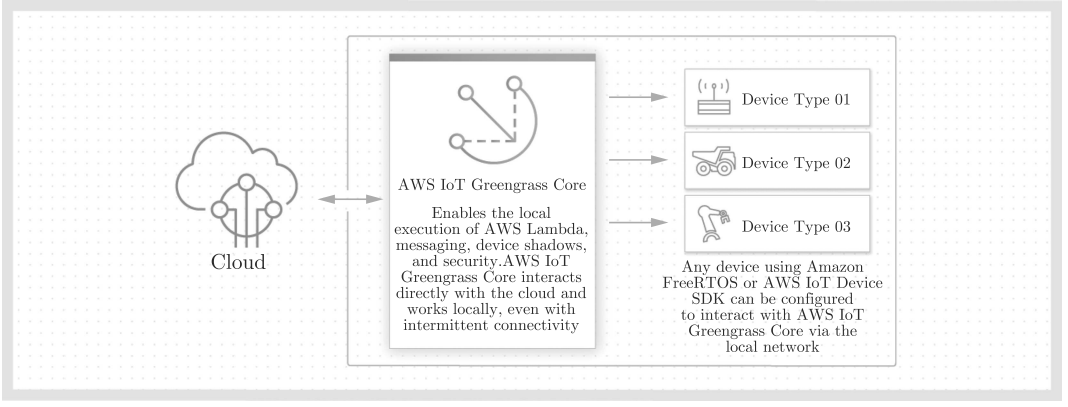

图 3.5　AWS IoT Greengrass 的基本架构

AWS IoT Greengrass 使客户能够构建 IoT 设备和应用程序逻辑。具体来说，AWS IoT Greengrass 对设备上运行的应用程序逻辑提供基于云的管理。在本地部署的 Lambda 函数和连接器通过本地事件及来自云或其他来源的消息触发。在 AWS IoT Greengrass 中，设备可在本地网络上安全地通信并互相交换消息而不必连接到云。AWS IoT Greengrass 提供了一个本地发布/订阅消息管理器，该管理器可在丢失连接的情况下智能地缓冲消息，使云的入站和出站消息得到保留。

AWS IoT Greengrass 将 AWS 无缝扩展到边缘设备中，以便它们可以在本地处理其生成的数据，同时仍使用云进行管理、分析和持久存储。借助 AWS IoT Greengrass，连接的设备可以运行 AWS Lambda 函数、基于机器学习模型执行预测，保持设备数据同步，以及与其他设备安全通信，甚至在没有连接互联网的情况下也可实现这些功能。

利用 AWS IoT Greengrass，可以使用熟悉的语言和编程模型在云中创建和测试设备软件，然后将其部署到设备中。也可以对 AWS IoT Greengrass 进行编程，管理设备上的数据的生命周期，使之可筛选设备数据，并仅将必要信息传输回 AWS。还可以使用 AWS IoT Greengrass 连接器连接到第三方应用程序、本地软件和即时可用的 AWS 服务。连接器还可以用预先构建的协议适配器集成快速启动设备，并允许通过与 AWS Secrets Manager 的集成简化身份验证。

AWS IoT Greengress 的控制台界面如图 3.6 所示。

图 3.6 AWS IoT Greengrass 的控制台界面

AWS IoT Greengrass 保护用户数据的方式包括：

① 通过安全的设备身份验证和授权；

② 通过本地网络中的安全连接建立本地设备与云之间的连接；

③ 设备安全凭证在撤销之前一直在组中有效，即使到云的连接中断，设备仍可以继续在本地安全地进行通信。

AWS IoT Greengrass 还提供了 Lambda 函数的安全、无线的软件更新，具体包括软件分发、AWS IoT Greengrass 核心软件、AWS IoT Greengrass 核心开发工具包、云服务、AWS IoT Greengrass API、Lambda 运行时、影子实施、消息管理器、组管理、发现服务、无线更新代理、本地资源访问、机器学习推理、本地密钥管理器等。

其中，AWS IoT Greengrass 核心软件提供了以下功能：connectors 和 Lambda 函数的部署与本地执行；本地密钥的安全、加密的存储，以及 connectors 和 Lambda 函数进行的受控访问；使用托管订阅通过本地网络在设备、connectors 和 Lambda 函数之间进行的 MQTT 消息传递；使用托管订阅在 AWS IoT 与设备、connectors 和 Lambda 函数之间进行的 MQTT 消息传递；使用设备身份验证和授权确保设备和云之间的安全连接；设备的本地影子同步；影子可配置为与云同步；对本地设备和卷资源的受控访问；用于运行本地推理的云训练机器学习模型的部署；使设备能够发现 Greengrass 核心设备的自动 IP 地址检测；全新的或更新的组配置的集中部署；下载配置数据后，核心设备将自动重启；用户定义的 Lambda 函数的安全、无线的软件更新。

官网：https://amazonaws-china.com/cn/greengrass/。

3.2.4 微软边缘计算平台 Azure IoT Edge

Azure IoT Edge 微软部署的云端融合物联网边缘计算平台，提供在 Azure IoT 中心上构建的完全托管服务。部署云工作负荷（人工智能、Azure 和第三方服务，或自己的业务逻辑）通过标准容器在物联网边缘设备上运行。通过将特定工作负荷迁移到网络

边缘，设备可减少与云的通信时间、加快对本地更改的响应速度，甚至可在较长的离线期内可靠地运行。

Azure IoT Edge 将云分析和自定义业务逻辑迁移到设备，使企业可以专注于业务见解而非数据管理。通过将业务逻辑打包到标准容器中，横向扩展 IoT 解决方案，然后可以将这些容器部署到任何设备，并从云中监视所有的这些设备。微软表示，这些设备现在将能够立即采取实时数据行动。借助开源的 Azure IoT Edge，开发人员可以更灵活地控制自己的边缘解决方案，以及运行时或调试问题。

为了解决 Azure IoT Edge 大规模部署的安全问题，Azure IoT Edge 深入集成了设备调配服务，以安全地配置数以万计的设备和 Azure IoT Edge 安全管理员，这些管理员可以用来保护边缘设备及其组件。自动设备管理（ADM）可以基于设备元数据将大型物联网边缘模块部署到设备。Azure IoT Edge 支持 C、C#、Java、Python 和 Node.js 等编程语言。它还提供 VSCode 模块开发、测试和部署工具，以及带 VSTS 的 CI/CD 管道。

Azure IoT Edge 包含以下三个组件。

① IoT Edge 模块：可以运行 Azure 服务、第三方服务或者本地代码的容器。这些模块部署到 IoT Edge 设备，在设备上以本地方式执行。IoT Edge 模块是执行单位，以 docker 兼容容器的方式实现，在边缘运行业务逻辑。可以将多个模块配置为互相通信，创建一个数据处理管道，也可以开发自定义模块，或者将某些 Azure 服务打包到模块中，以脱机方式在边缘提供服务。

② IoT Edge 运行时：如图 3.7所示，IoT Edge 包含 Moby 引擎，在每个 IoT Edge 设备上运行，并管理部署到每个设备的模块。允许在 IoT Edge 设备上使用自定义逻辑和云逻辑。运行时位于 IoT Edge 设备上，执行管理和通信操作，并执行多个功能，具体包括：在设备上安装和更新工作负荷；维护设备上的 Azure IoT Edge 安全标准；确保 IoT Edge 模块始终运行；将模块运行状况报告给云以进行远程监控；管理下游设备与 IoT Edge 设备之间、IoT Edge 设备上的模块之间，以及 IoT Edge 设备与云之间的通信。Azure IoT Edge 运行时可以通过各种方式在各种大型 IoT 设备上运行。它支持 Linux 和 Windows 操作系统，并可提取硬件详细信息。如果要处理的数据不多，可使用比 Raspberry Pi 3 小的设备；如果要运行资源密集型工作负荷，也可使用工业服务器。

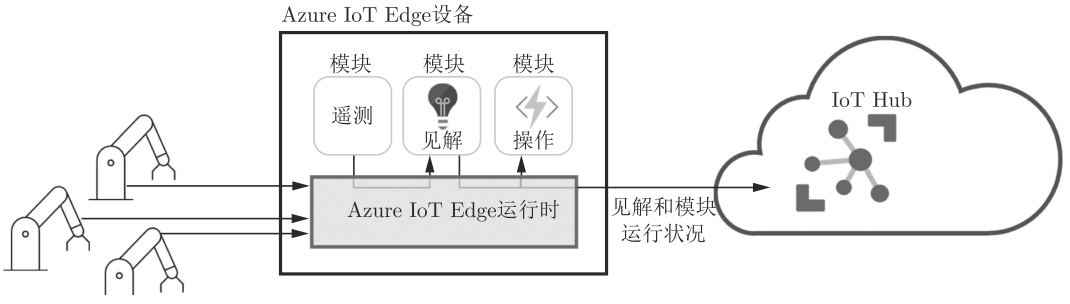

图 3.7　IoT Edge 运行时框架

Azure IoT Edge 运行时是免费且开源的，但客户必须使用付费的 Azure IoT Hub 实例进行扩展。边缘设备的管理和部署也将基于 Azure 服务或客户使用的 Edge 模块。

③ IoT Edge 云接口：如图 3.8所示。管理数百万台 IoT 设备的软件生命周期很困难，这些设备通常具有不同的品牌且型号各异，或者地理位置分散。需要为特定类型的设备创建和配置工作负荷，部署到所有设备，并监视以捕获任何行为异常的设备。这些活动不能逐个设备地完成，必须大规模地进行操作。因此，可以通过基于云的接口界面远程监视和管理 IoT Edge 设备。Azure IoT Edge 提供了一个符合上述解决方案需要的控制平面。

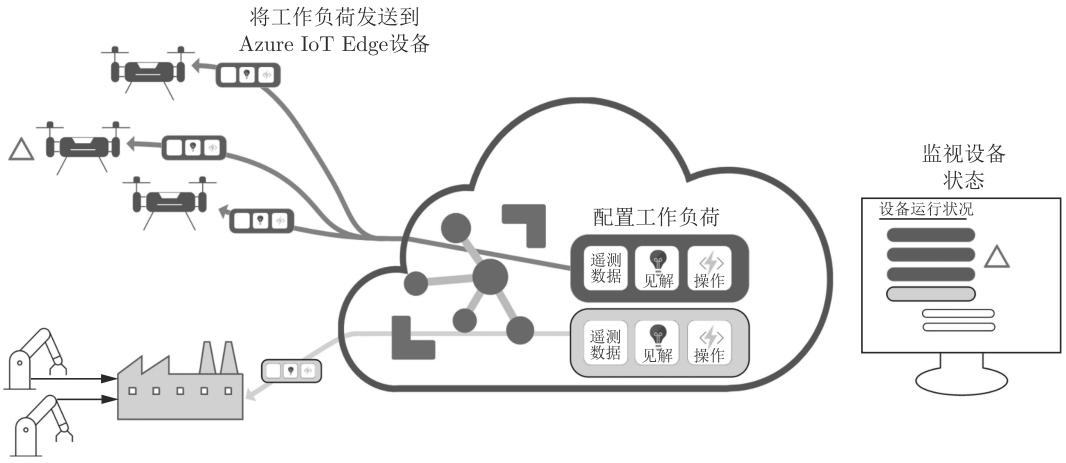

图 3.8 IoT Edge 云接口框架

官网：https://azure.microsoft.com/zh-cn/services/iot-edge/。

3.2.5 阿里云边缘计算平台 Link Edge

在 2018 年云栖大会·深圳峰会上，阿里云推出首个 IoT 边缘计算产品——Link Edge。Link Edge 是阿里云的一款云边一体的 PaaS 层软件产品，将云端的能力下沉到边缘侧，解决边缘实时性、可靠性、运维经济性等方面遇到的问题。南向提供通信协议框架为软硬件开发者提供便捷的通信协议开发能力，北向通过 Open API 为 SaaS 开发者提供快速构建云端应用的能力。对于运维，云端提供一体化的运维工具，可以在云端集中运维，降低运维成本，提升运维效率。

Link Edge 继承了阿里云安全、存储、计算、人工智能的能力，可部署于不同量级的智能设备和计算节点中，通过定义物模型连接不同协议、不同数据格式的设备，提供安全可靠、低延时、低成本、易扩展、弱依赖的本地计算服务。同时，物联网边缘计算可以结合阿里云的大数据、AI 学习、语音、视频等能力，打造出“云—边—端”三位一体的计算体系。其核心功能主要有边缘实例、设备接入、场景联动、边缘应用、流数据分析和消息路由。Link Edge 架构如图 3.9所示，主要涉及设备端、边缘计算端和云端三部分。

① 设备端：提供设备接入工具，将各厂商提供的协议及数据格式的设备转换为标

准统一的设备模型，开发者使用设备接入 SDK，将非标设备转换成标准物模型，就近接入网关，从而实现设备的管理和控制。

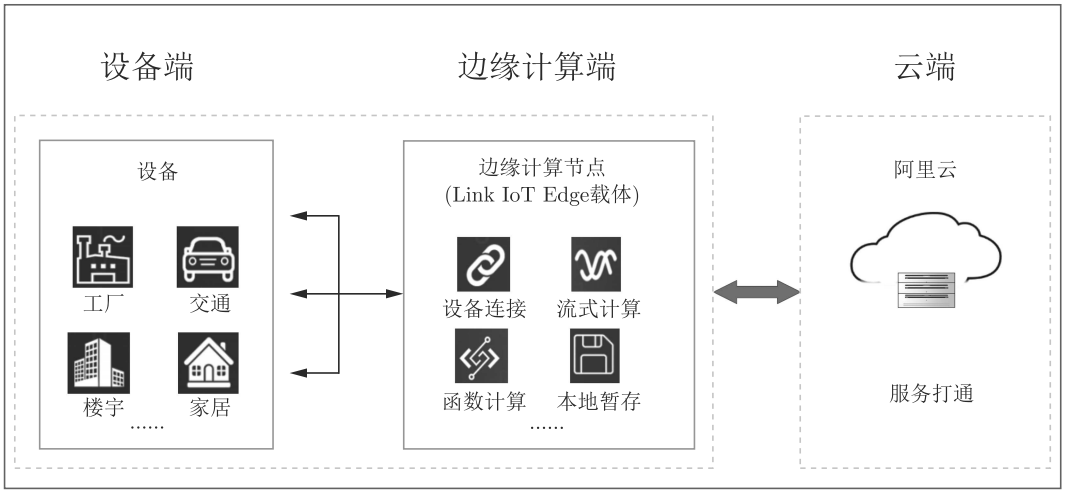

图 3.9　Link Edge 体系架构

② 边缘计算端：一定资源要求下的计算在边缘就近计算，设备与云端断开连接的情况下计算不被中止；设备连接到网关后，网关可以实现设备数据的采集、流转、存储、分析和上报设备数据至云端，同时网关提供规则引擎、函数计算引擎，方便场景编排和业务扩展。

③ 云端：设备数据上传至云端后，可以结合阿里云功能，如大数据、AI 学习等，通过标准 API 实现更多的功能和应用。

Link Edge 的应用领域越来越广泛，通过通用链接框架安全、快速地将设备连接至边缘核心软件，可以在本地实时处理设备数据，进行设备之间的数据转发和暂存，并通过边缘核心软件连接至云端，打通云端能力。Link Edge 可应用于未来酒店、工业生产、风力发电等场景。例如，在未来酒店，边缘网关快速集成本地设备后，作为本地节点快速响应本地事件，可实现本地 M2M 的智能联动和室内外一体化的语音智能；在风力发电的应用场景中，部署边缘计算网关，实时采集机组数据，在本地处理采集的数据后，先将数据上传至阿里云 MaxCompute，再使用大数据训练模型对发电参数（如风向灵敏度、启动延时参数等）进行优化，将模型转化为算法或者规则导入本地边缘节点，自动调整风电机组参数，提高机组的发电性能。

官网：https://www.aliyun.com/product/iotedge。

3.2.6　华为开源边缘计算平台 KubeEdge

KubeEdge 是华为提供的开源的边缘计算平台，用于将本机容器化的应用程序编排功能扩展到 Edge 上的主机，它在 kubernetes 原生的容器编排和调度能力之上实现了云边协同、计算下沉、海量边缘设备管理、边缘自治等能力。在追求边缘极致轻量

化的同时，结合云原生生态的众多优势，解决当前智能边缘领域面临的挑战。它基于 kubernetes 构建，并为网络应用程序提供基础架构支持，为云和边缘之间的部署和元数据实现同步。KubeEdge 使用 Apache 2.0 许可，并且绝对可以免费用于个人或商业用途。

如图 3.10所示，KubeEdge 分为 CloudCore 和 EdgeCore 两个可执行程序，它们分别包含以下模块。

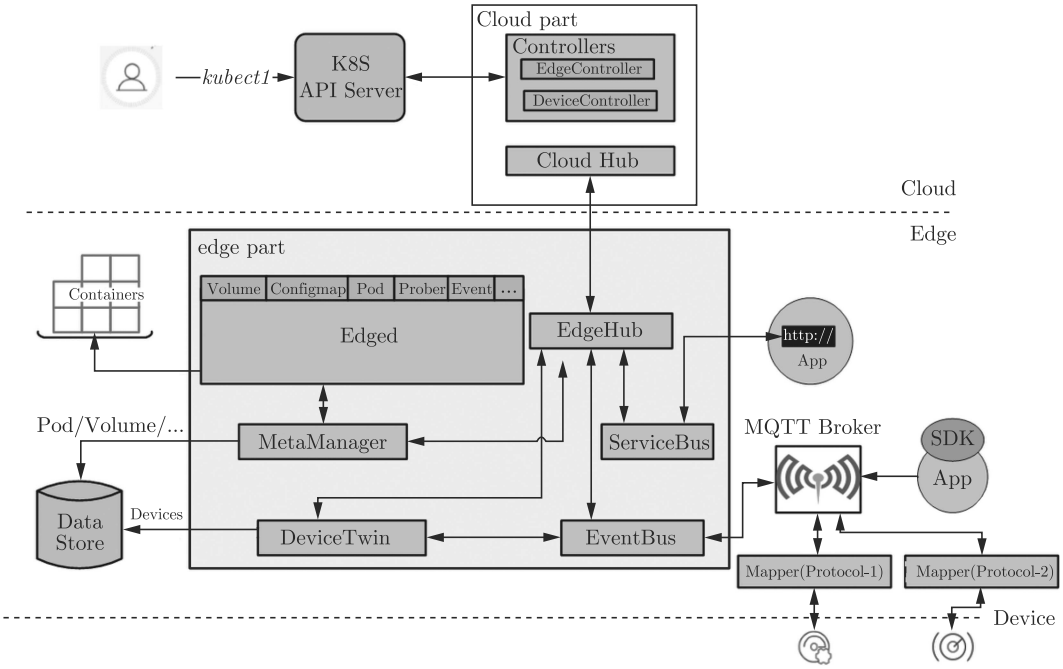

图 3.10 KubeEdge 体系架构

- CloudCore
  - Cloud Hub：云中的通信接口模块；
  - EdgeController：管理 Edge 节点；
  - DeviceController：负责设备管理。
- EdgeCore
  - Edged：在边缘管理容器化的应用程序；
  - EdgeHub：Edge 上的通信接口模块；
  - EventBus：使用 MQTT 处理内部边缘通信；
  - DeviceTwin：用于处理设备元数据的设备的软件镜像；
  - MetaManager：管理边缘节点上的元数据；
  - ServiceBus：接收云上服务请求和边缘应用进行 http 交互。

KubeEdge 的特点包括：完全开放，EdgeCore 和 CloudCore 都是开源的；离线模式，即使与云断开连接，Edge 也可以运行；基于 kubernetes，包括节点、群集、应用程序和设备管理；可扩展、容器化、微服务；资源优化，可在资源不足的情况下运行，可实

现边缘云上资源的优化利用；跨平台，用户无感知；可以在私有、公共和混合云中工作；支持数据管理和数据分析管道引擎；支持异构，包括 x86、ARM 等；简化开发，基于 SDK 的设备加成、应用程序部署等开发；易于维护，包括升级、回滚、监视、警报等。

基于 KubeEdge 的上述特点，将在 3.3节重点阐述 KubeEdge 的架构搭建和实例开发过程。

除此之外，华为还提供了物联网边缘计算解决方案 Intelligent EdgeFabric，它通过管理用户的边缘节点，提供将云上应用延伸到边缘的能力，联动边缘和云端的数据，同时，在云端提供统一的设备/应用监控、日志采集等运维能力，为企业提供完整的边缘计算解决方案。Intelligent EdgeFabric 体系架构如图 3.11所示。

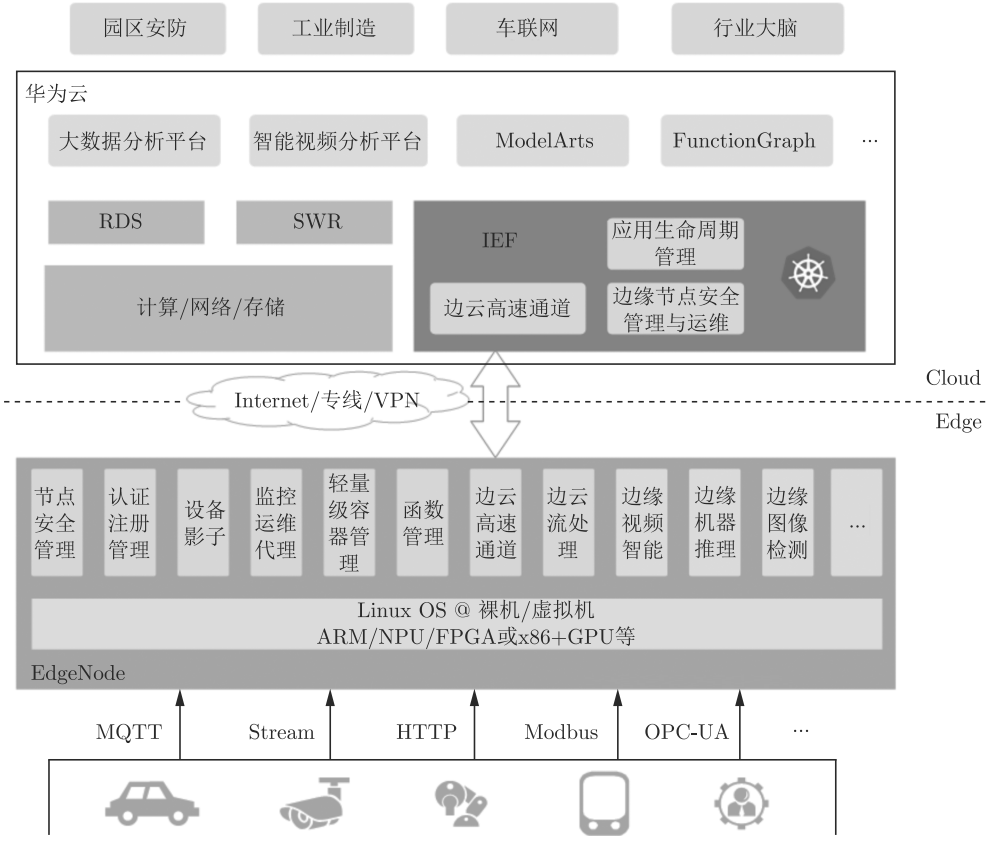

图 3.11　Intelligent EdgeFabric 体系架构

官网：https://kubeedge.io/zh/。

GitHub：https://github.com/kubeedge/kubeedge。

3.2.7　百度边缘计算平台 Baetyl

百度的边缘计算平台 Baetyl 的前身是开源边缘计算平台 OpenEdge，后来百度智能云宣布将百度智能边缘计算框架 Baetyl 捐赠给 Linux Foundation Edge 社区，成为

其旗下项目。Baetyl 旨在将云计算能力拓展至用户现场，提供临时离线、低延时的计算服务，包括设备接入、消息路由、消息远程同步、函数计算、设备信息上报、配置下发等功能。Baetyl 和智能边缘（Baidu-IntelliEdge，BIE）云端管理套件配合使用，通过在云端进行智能边缘核心设备的建立、存储卷创建、服务创建、函数编写，然后生成配置文件下发至 Baetyl 本地运行包，整体可达到边缘计算、云端管理、边云协同的效果，满足各种边缘计算场景。目前，Baetyl v2 提供了一个全新的边云融合平台，采用云端管理、边缘运行的方案，分成边缘计算框架和云端管理套件两部分，支持多种部署方式。可在云端管理所有资源，如节点、应用、配置等，自动部署应用到边缘节点，满足各种边缘计算场景，特别适合新兴的强边缘设备，如 AI 一体机、5G 路侧盒子等。

Baetyl 体系架构如图 3.12所示。在架构设计上，Baetyl 一方面推行模块化，拆分各项主要功能，确保每一项功能都是一个独立的模块，整体由主程序控制启动、退出，确保各项子功能模块运行互不依赖、互不影响；总体上来说，推行模块化的设计模式，可以充分满足用户按需使用、按需部署的切实要求；另一方面，Baetyl 在设计上还采用全面容器化的设计思路，基于各模块的镜像可以在支持 docker 的各类操作系统上进行一键式构建，依托 docker 跨平台支持的特性，确保 Baetyl 在各系统、平台的环境一致；此外，Baetyl 还针对 docker 容器化模式赋予其资源隔离与限制能力，精确分配各运行实例的 CPU、内存等资源，提升资源利用效率。

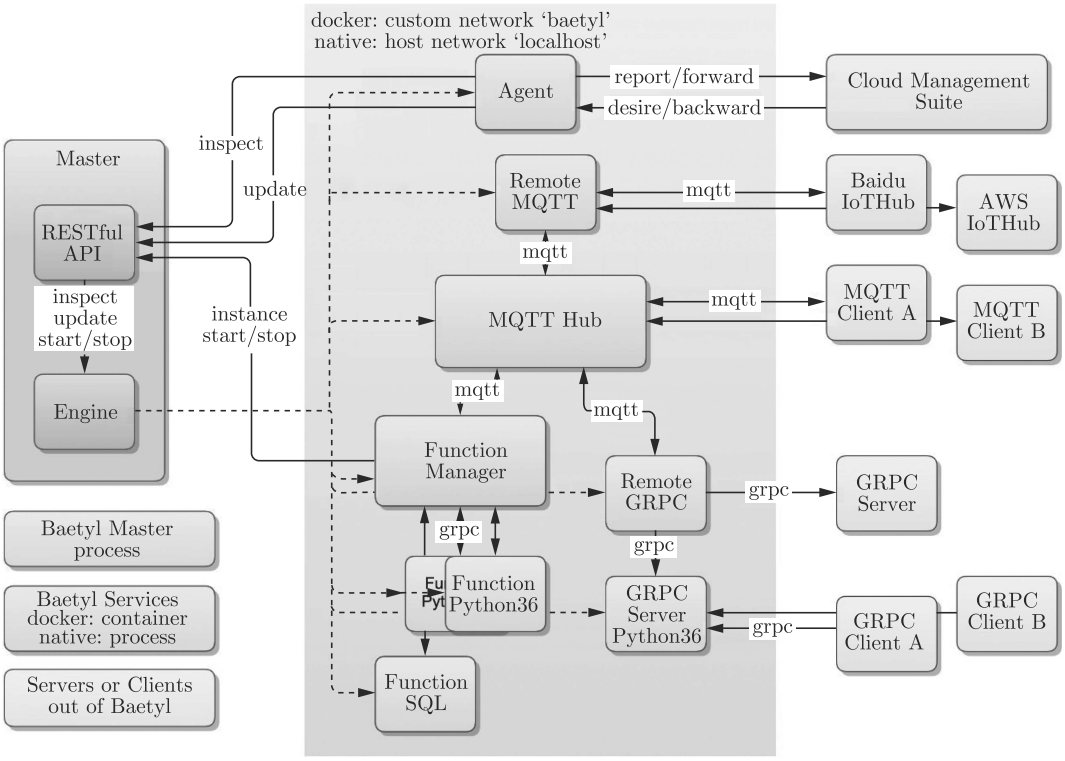

图 3.12　Baetyl 体系架构（见彩插）

官网：https://www.baetyl.io/zh/。

Github：https://github.com/baetyl/baet-yl。

3.2.8　阿里巴巴边缘计算云原生项目 OpenYurt

OpenYurt 是阿里巴巴基于 ACK@Edge（边缘集群托管服务）推出的开源云原生边缘计算框架。该框架依托 kubernetes 强大的容器应用编排、调度能力，实现完全边缘计算云原生基础设施架构，帮助开发者轻松完成在海量边、端资源上的大规模应用的交付、运维和管控。OpenYurt 适用于常见的边缘计算用例，其需求包括使设备和工作负载之间的长距离网络流量最小化、克服网络带宽或可靠性限制、远程处理数据以减少延迟、提供更好的安全模型来处理敏感数据。

OpenYurt 具有强大的边缘自治、边缘运维、集群转换能力。其技术方案有如下几个特点。

- 保持原生的 kubernetes：保证对原生 kubernetes API 的完全兼容，通过对 kubernetes 节点应用生命周期管理添加一层新的封装，提供边缘计算所需要的核心管控能力。
- 无缝转换：提供了一个工具，可以轻松地将本机 kubernetes 转换为“edge”就绪，而且 OpenYurt 组件的额外资源和维护成本非常低。
- 节点自治：提供了云边网络容忍机制，即使网络中断，运行在边缘节点中的应用程序也不会受到影响。
- 云平台无关：OpenYurt 可以很容易地部署在任何公共云 kubernetes 服务中。

OpenYurt 遵循经典的边缘应用程序架构设计，即云端的一个 kubernetes 主节点管理驻留在边缘端的多个边缘节点。每个边缘节点都有适当的计算资源，允许运行多个边缘应用程序和 kubernetes 节点守护进程。OpenYurt 架构如图 3.13所示，其主要组件包括以下几种。

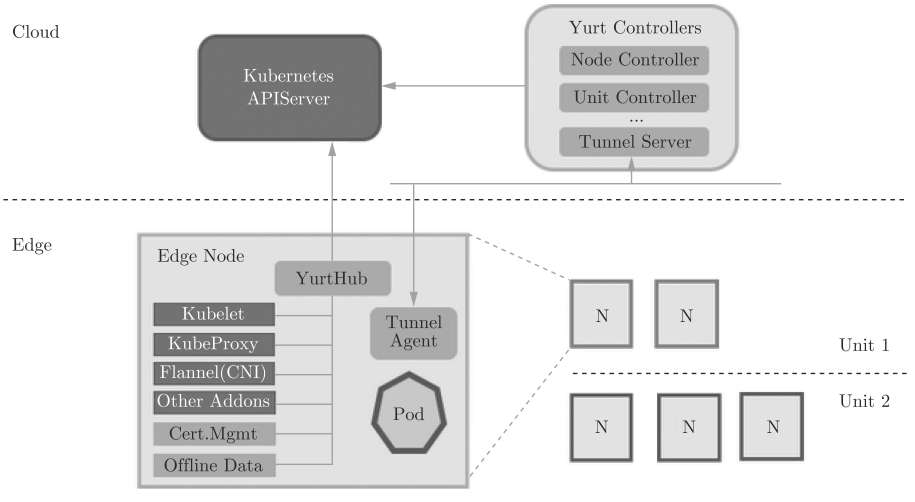

图 3.13　OpenYurt 架构

① YurtHub：一个节点守护进程，作为 kubernetes 节点守护进程（如 Kubelet、KubeProxy、CNI 插件等）的出站流量的代理。它在边缘节点的本地存储中缓存 kubernetes 节点守护进程可能访问的所有资源的状态。如果边缘节点是离线状态，这些守护进程可以在节点重启时恢复状态。

② Yurt 控制器管理器：针对不同的边缘计算用例，管理一些控制器，如节点控制器和单元控制器（即将发布）。

③ Yurt 隧道服务器：通过反向代理连接在每个边缘节点上运行的 TunnelAgent 守护进程，实现在云端控制平面和连接到 Intranet（待发布）的边缘节点之间建立安全的网络访问。

GitHub：https://github.com/alibaba/openyurt。

3.3 开发应用实例

3.3.1 虚拟机开发环境

为方便读者复现和动手实操，本节的开发实例均提供了虚拟机开发环境。搭建 KubeEdge 需要配置云端主机和两个边缘端设备，其环境配置见表 3.1。由于云端主机需要编译源码，因此配置了较大的内存和硬盘空间。

表 3.1 KubeEdge 开发环境

	OS 版本	CPU 型号	内存/GB	硬盘容量/GB
云端	Ubuntu 16.04 64 bit	Intel®Core™ i7-8700	8	20
边缘端 1	Ubuntu 16.04 64 bit	Intel®Core™ i7-8700	2	10
边缘端 2	Ubuntu 16.04 64 bit	Intel®Core™ i7-8700	2	10

目前支持运行 Azure IoT Edge 的操作系统分为两类：第一类系统由微软进行过严格的 Azure IoT Edge Runtime 测试，完全兼容，并提供了安装程序包；第二类是理论上可兼容，或者微软合作伙伴已经成功运行了的系统。Azure IoT Edge 在虚拟机中运行时，主机操作系统必须与模块容器内部使用的客户操作系统相匹配。本书运行 Azure IoT Edge 的操作系统是 Ubuntu Server 16.04，内存为 4GB，硬盘存储容量为 20GB。

3.3.2 KubeEdge 搭建与实例开发

KubeEdge 架构分为三部分，分别是云、边、端三侧。云端负责云上应用和配置的校验、下发，边缘侧则负责运行边缘应用和管理接入设备，设备端运行各种边缘设备，由此完成云、边、端的协同。KubeEdge 组件分为云端组件和边缘侧组件，云端组件包括 CloudCore、Admission Webhook，它们构建在 kubernetes 的调度能力之上，可以运行在任何 kubernetes 集群中；边缘侧组件包括 EdgeCore 及接入设备的 Mappers，其中，Mappers 负责接入边缘设备，EdgeCore 负责边缘应用与设备管理。目前，KubeEdge 的版本已更新到 v1.3。本节以 KubeEdge v1.1 为例，详细讲解 KubeEdge 的搭建过程，并给出一个运行示例，供读者参考。

环境配置如下。

- 云端 master：Ubuntu 16.04 64 bit，用户名为 k8smaster，IP 为 192.168.137.131。
- 边缘端 slave1：Ubuntu 16.04 64 bit，用户名为 k8slave1，IP 为 192.168.137.132。
- 边缘端 edge：Ubuntu 16.04 64 bit，用户名为 edge-node，IP 为 192.168.137.133。

KubeEdge 部署需要如下组件。

- 云端：docker、kubernetes 集群和 KubeEdge 云端核心模块。
- 边缘端：docker、mqtt 和 KubeEdge 边缘端核心模块。

1. 搭建 kubernetes 集群

1）安装 docker

docker 是一个用于开发、发布和运行应用程序的开放平台。利用 docker 快速交付、测试和部署代码，可以显著减少代码的编写和在运行代码之间的延迟。

```
# apt-get install docker.io
```

新建/etc/docker/daemon.json 文件：

```
# cat > /etc/docker/daemon.json <<-EOF
{
    "registry-mirrors": [
        "https://a8qh6yqv.mirror.aliyuncs.com",
        "http://hub-mirror.c.163.com"
    ],
    "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
```

其中，registry-mirrors 为镜像加速器地址。

重启 docker，查看 cgroup：

```
# systemctl restart docker
# docker info | grep -i cgroup
Cgroup Driver: systemd
```

2）master 主机部署 kubernetes

（1）关闭 swap。

编辑 /etc/fstab 文件，注释 swap 分区挂载的行，再执行以下命令:

```
# swapoff -a
```

（2）添加 kubernetes 国内源并更新。

```
# apt-get update && apt-get install -y apt-transport-https
# curl https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg | apt-key add -
```

```
# cat << EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://mirrors.aliyun.com/kubernetes/apt/ kubernetes-xenial main
EOF
# apt-get update
```

（3）安装 kubeadm 等工具。

```
# apt-get install -y kubeadm kubectl kubelet kubernetes-cni
安装指定版本：
# apt-get install kubeadm=1.17.0-00 kubectl=1.17.0-00 kubelet=1.17.0-00
```

其中，kubeadm 用作引导集群的命令，kubectl 用作命令行工具，kubelet 作为在集群中的所有机器上运行的组件，执行诸如启动 pods 和容器之类的操作。安装之后，kubelet 每隔几秒就会重新启动一次，它在 crashloop 中等待 kubeadm 的指令。

（4）获取部署所需的镜像版本。

执行如下命令可获取镜像列表：

```
# kubeadm config images list
```

kubeadm 配置镜像列表如图 3.14所示。

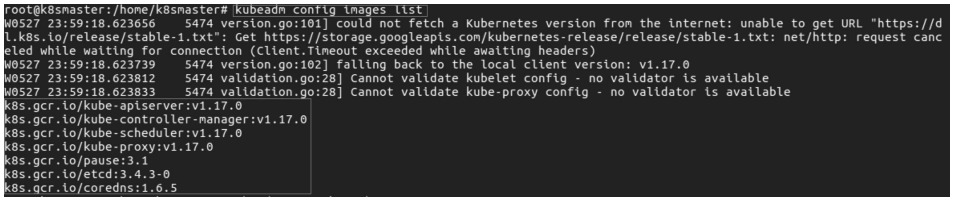

图 3.14 kubeadm 配置镜像列表

（5）拉取镜像文件。

初始化 kubernetes 时，需要配置阿里云的镜像仓库，编辑脚本 pullk8s.sh：

```
images=(
kube-apiserver:v1.17.0
kube-controller-manager:v1.17.0
kube-scheduler:v1.17.0
kube-proxy:v1.17.0
pause:3.1
etcd:3.4.3-0
coredns:1.6.5
)

for imageName in ${images[@]} ; do
docker pull registry.cn-hangzhou.aliyuncs.com/google_containers/$imageName
docker tag registry.cn-hangzhou.aliyuncs.com/google_containers/$imageName k8s.gcr.io/$imageName
docker rmi registry.cn-hangzhou.aliyuncs.com/google_containers/$imageName
done
```

执行脚本拉取镜像：

```
# chmod +x pullk8s.sh
# bash pullk8s.sh
```

（6）下载 flannel 镜像。

flannel 是一个专为 kubernetes 定制的三层网络解决方案，在以太网的基础上再封装的一个包含容器 IP 地址的虚拟网络，主要用于解决容器的跨主机通信问题。

```
# docker pull quay.io/coreos/flannel:v0.11.0-amd64
```

（7）初始化 kubernetes。

```
# kubeadm init --pod-network-cidr=10.244.0.0/16
```

选择一个 Pod 网络附加组件，并验证它是否需要向 kubeadm init 传递任何参数。根据选择的第三方提供商，需要将--pot-network-cidr 设置为特定于提供商的值。k8s 初始化成功如图 3.15所示。

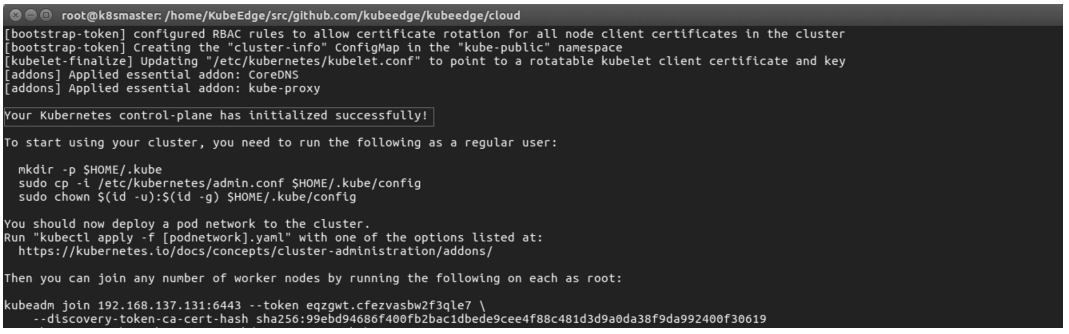

图 3.15　k8s 初始化成功

kubeadm init 首先执行一系列预检查，以确保机器已准备好运行 kubernetes。这些预检查会报出警告并在出现错误时退出。然后，kubeadm init 下载并安装 cluster control plane 的组件。

根据提示，复制 admin.conf 文件到当前用户的相应目录下，以支持非 root 用户运行 kubectl。

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

token 用于 control-plane 节点和 joining 节点之间的相互身份验证。token 的内容应该保密，因为任何拥有 token 的人都可以向集群添加经过验证的节点。kubeadm token 命令可以列出、创建和删除 token。

（8）部署 flannel。

安装完 master 节点后，查看节点信息（kubectl get nodes），发现节点的状态为 Noready，此时用 flannel 进行网络配置。

```
# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
      flannel.yml
```

3）配置 node 节点

（1）前期准备。

前期准备工作包括：

- 安装 kubeadm；
- 下载 flannel 镜像；
- 将主机的/etc/kubernetes/admin.conf 文件复制到 node 节点的/etc/kubernetes/目录。

（2）加入集群。

执行命令加入节点，结果如图 3.16所示。

```
root@k8slave1:/home# kubeadm join 192.168.137.131:6443 --token eqzgwt.cfezvasbw2f3qle7 \
>     --discovery-token-ca-cert-hash sha256:99ebd94686f400fb2bac1dbede9cee4f88c481d3d9a0da38f9da992400f30619
W0603 01:30:33.416651  110000 join.go:346] [preflight] WARNING: JoinControlPane.controlPlane settings will be ignored when control-pl
ane flag is not set.
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.17" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

图 3.16　node 节点加入成功

4）验证 kubernetes 集群

在 master 节点执行 kubectl get nodes 验证节点状态。

如图 3.17所示，两台机器已为 Ready 状态，kubernetes 集群部署成功。

```
root@k8smaster:/home/KubeEdge# kubectl get nodes
NAME        STATUS   ROLES    AGE     VERSION
k8slave1    Ready    <none>   34m     v1.17.0
k8smaster   Ready    master   5h53m   v1.17.0
```

图 3.17　k8s 部署成功

2. 安装 KubeEdge

1）KubeEdge 环境准备

（1）创建部署文件目录。

```
# mkdir /home/KubeEdge
# cd /home/KubeEdge
# mkdir cloud edge certs yamls src
```

cloud 和 edge 文件夹分别存放编译云端和边缘端生成的文件，certs 存放生成的证书，yamls 存放配置文件，src 存放源码。

（2）golang 环境搭建。

- 下载 golang，并解压到/usr/local。
- 添加环境变量。

在 ~/.bashrc 文件末尾添加：

```
# vim ~/.bashrc
export GOPATH=/home/KubeEdge
export PATH=$PATH:/usr/local/go/bin
```

保存后执行 source ~/.bashrc 命令生效，执行 go version 命令验证。

（3）下载 kubeedge 源码。

```
# git clone https://github.com/kubeedge/kubeedge.git $GOPATH/src/github.com/kubeedge/kubeedge
```

（4）检测 gcc 是否安装。

确保主机上已经安装了 gcc。用以下命令检查：

```
# gcc --version
```

（5）选择 kubeedge 版本。

kubeedge v1.1.0 和 kubeedge v1.2.0 以后的版本编译方式有一些区别，clone 下来的版本已经不是 v1.1.0 了，此处需要把版本切回到 v1.1.0。

```
# cd $GOPATH/src/github.com/kubeedge/kubeedge/
# git tag
# git checkout v1.1.0
```

（6）编译云端。

```
# cd $GOPATH/src/github.com/kubeedge/kubeedge/
# make all WHAT=cloudcore
```

生成的二进制 cloudcore 文件位于 cloud 目录。复制 cloudcore 和同一目录的配置文件（conf 目录）到部署工程目录：

```
# cp -a cloud/cloudcore $GOPATH/cloud/
# cp -a cloud/conf/ $GOPATH/cloud/
```

（7）编译边缘端。

```
# cd $GOPATH/src/github.com/kubeedge/kubeedge/
# make all WHAT=edgecore
```