Extend cloud execution environment to edge with KubeEdge

Yulin Sun, yulin.sun@huawei.com; Ying Xiong, ying.xiong1@huawei.com; Li Xing, Li.xing1@huawei.com; Ying Huang, ying.huang@huawei.com;

Seattle Cloud Lab, Huawei R&D USA, Bellevue WA

ABSTRACT: In this paper, we introduce an Edge infrastructure (KubeEdge) for the Edge-cloud communication and execution environment, and we consider it as an extension of Cloud infrastructure. This edge infrastructure allows existing Cloud services and Cloud development model to be adopted at Edge and provides seamless communication between edge and cloud. The KubeEdge includes a network protocol stack called KubeBus, a distributed Edge metadata store/synchronization service and an application orchestration service. KubeBus is designed to have its own implementation of OSI layer 2/3/4 protocol, which supports to connect Edge Nodes and VMs in Cloud as one VPN and provides a common multitenant management/data plane for different tenants' edge clusters. Services running in Cloud and at Edge communicate with each other on top of KubeBus with fault tolerance and high availability. The Edge metadata service in KubeEdge provides an edge metadata storage and a service to synchronize metadata between cloud and Edge to supports edge node offline scenario. The KubeEdge includes Kubernetes [1] extension so that Kubernetes could manage Edge Nodes as well as VMs in Cloud and deploys/manages application of Edge Nodes.

KEYWORDS; Edge Computing, Cloud computing, Network protocol stack, Application orchestration

1 INTRODUCTION

With the rapid growing requirement for Edge execution of IOT, AI application, and local stream data analytics, Edge computing, which enable the computation to be "performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services"[2], become more and more important.

In Edge computing, the services which originally running in Cloud are deployed to the Edge environment and collaborated with other Edge services and Cloud services. The Edge services scheduling, life cycle management and monitoring might be done from Cloud. For example, previous study [3] push the computation from Data Center by extend existing platform with optimized algorithm for WAN environment. Beside common data process such data filter and aggregation, AI process is also pushed from data center to Edge, recent research [4] "partition DNN computation between mobile devices and datacenters at the granularity of neural network layers".

"Cloud computing is TCP/IP based high development and integrations of computer technologies such as fast micro-processor, huge memory, high-speed network and reliable system architecture."[The Characteristics of Cloud Computing]. There is mature commercial cloud computing infrastructure to manage the cloud applications. For example, IaaS infrastructure OpenStack, Amazon EC2; containerized application management system Kubernetes [1], docker swarm.

The existing cloud computing infrastructure might not apply to the Edge environment directly. It is because Edge environment is different than the Cloud environment, with respect to network connectivity/topology, network bandwidth and relatively constrained computing resource. For example, one typical Edge scenario is that multiple Edge nodes and Cloud are connected by a Wide-Area Network (WAN). The Edge Nodes are running in private network behind NAT, so that the TCP client running at Cloud can't connect to the TCP service running at Edge; additionally the network between Edge and Cloud might be unstable, the service running at Edge needs to be able to run offline; and the bandwidth between Edge Node and Cloud is less than in Cloud and is more expensive; finally some Edge Node has constrained computation resource such as only 128 MB memory. So dedicate edge computing infrastructure is necessary to address Edge special characters for fast application development and low cost operation on the edge.

The paper presents an Edge infrastructure which support scheduling services to Edge Nodes same as VMs in Cloud with consideration of Edge environment special characters. The Edge infrastructure is based on following 3 components:

1) KubeBus which connects Edge Node and VMs as a VPN and connects one multi-tenant management/data plane to all tenants' Edge clusters;

2) EdgeMetadataService which support metadata storage and Edge and a-synchronization metadata between Cloud and Edge;

3) Kubernetes extension to enable edge application deployment and life-cycle management.

The infrastructure is suitable to extend micro-service based application from Cloud to Edge.

The rest of the paper is organized as below. Section 2 discuss the related work for Edge computing infrastructure; Section 3 introduce the architect and design of the KubeEdge; Section 4 the experiment result for KubeBus; Section 5 discuss the future work; and Section 6 concludes the paper.

2 RELATED WORK

In the era of Internet of Thing (IoT), billions of sensors and actuators are deployed worldwide. To manager the IoT devices and process data from them with Cloud computing resource, Cloud providers provide IoT platforms include Azure IoT platform, AWS IoT and Google IoT solution. The IoT platforms employ a Pub/Sub brokers such as Mqtt [9] or AMQP to handle the communication channel between IoT device and Cloud service, such as Azure IoT hub.

The IoT devices count is huge and the IoT data is too large to fit in the Internet bandwidth to send to Cloud. Cloud providers publish the Edge platform such as AWS GreenGrass [5] and Azure IOT Edge [6] to manage the IoT application execution and IoT data process at Edge side as well as data transfer between Cloud and Edge. For example, GreenGrass extends the AWS Lambda function environment to the Edge, so that Edge application could be deployed to and managed in Edge as a Lambda function. In AWS GreenGrass, the Edge services communicate with each other and Cloud services with Pub/Sub message. Azure IoT Edge has also Edge Hub which extends IoT Hub's functionality to Edge and provides Pub/Sub communication to Edge services. In this model, Edge node communication with central cloud based on Pub/Sub sematic through similar channel designed for IoT device. From Central Cloud point of view, Edge node is taken as a special IoT device.

Pub/Sub protocol such as Mqtt is suitable for communication between Edge devices and Cloud as it is lightweight protocol. As the Edge Nodes might run behind NAT, services running in cloud can't connect to Edge Node directly as a TCP client. Pub/Sub, by nature, is easy to solve the issue. But is it suitable as major communication RPC between Edge services and Cloud? Some of the Edge computation is extension of the Cloud computation and Edge service is deeply integrated with Cloud services such as [3][4]. The Cloud computation programming model is different than Pub/Sub. For example, one popular Cloud programming model is Micro-service, which is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. To extend a Cloud application based on Micro-service to Edge, part of the small services of one application are deployed in the Edge Nodes. It will good that the Edge services shall communicate with each other and the other Cloud services with same RPC as in Cloud. To achieve this goal, Edge platform is required to provide a homogeneous execution environment as Cloud. In the environment, services could talk with each other as in one single cluster no matter they are running in Edge or Cloud. Furthermore, the micro-services shall be freely scheduled between Edge node and Cloud with same deployment infrastructure.

Pub/Sub protocol might not be suitable for extend the micro-services from Cloud to Edge. As inter-micro-service RPC may not be Pub/Sub protocol, the services running in Edge computing infrastructure based on Pub/Sub protocol needs refactoring for satisfy infrastructure requirement.

KubeEdge builds a homogeneous execution environment for Cloud and Edge environment. It addresses the Edge Node connectivity topology issue by KubeBus, which connect Edge Nodes, VMs and container network in the Cloud as a VPN; it also includes a Kubernetes extension to extend Kubernetes functionality to Edge environment so that Edge services could be deployed to Edge as in Cloud; The KubeEdge mitigates the Edge unstable connection to Cloud with EdgeMetadataService, which provides metadata store at Edge and metadata a-synchronization between Cloud and Edge. Next Section will give detail design for them.

3 ARCHITECTURE AND DESIGN

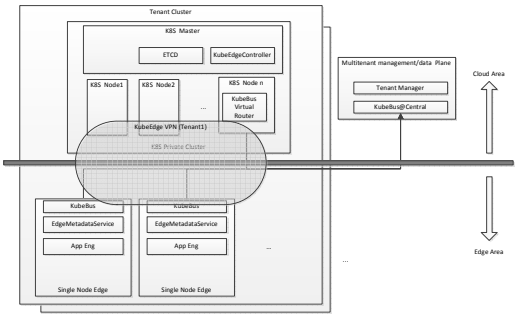

Figure 1: KubeEdge architecture

The KubeEdge provides multi-tenant Edge infrastructure. As figure 1, it includes one multi-tenant management/data plane at Cloud area and multiple tenant clusters. The multi-tenant

management/data plane includes KubeBus@Cloud and the Tenant Manager, which manages the tenants. One tenant cluster includes one or more Edge Nodes running at Edge area and one Kubernetes cluster running in the Cloud. The Kubernetes cluster include Kubernetes master and multiple Virtual Machines (VMs). The Kubernetes master takes both VMs and Edge Nodes as normal Kubernetes Nodes.

There is one KubeEdge agent running in each Edge Node. The KubeEdge agent includes KubeBus which provides Edge network access, EdgeMetadataService provides the edge applications metadata storage and synchronization service; and AppEngine which manages the Edge application's life cycle. The AppEngine is the lightweight Kubelet running at Edge.

In the Cloud area, for each tenant's Kubernetes cluster, there is one KubeBus virtual router running in one VM nodes, it routes the traffic between Edge Node subnet and VM subnet/container network subnet; there is also an EdgeController running in KubeMaster, which exchange the edge services configuration and status between KubeMaster and Edge Nodes.

3.1 KubeBus

In the KubeEdge environment, one service could be scheduled to run either in Cloud VM or in Edge Node and communicates with other services with same RPC as in Cloud environment. The KubeBus is designed to address the connectivity issue between edge services, which running in Edge Node and the cloud services, which running in container network in Cloud environment. KubeBus also supports one single management/data plane for multiple tenants and provides Http WebService publish from Edge Nodes.

3.1.1 Edge Node VPN

The first KubeBus's functionality is to link the Edge Nodes in an Edge Virtual Private Network. In one typical Edge environment, different Edge Node running in different private network without public IP address and it could only connect to Cloud with NAT. There is no direct network connection between 2 Edge Nodes. The 2 Edge Nodes' communication needs to be routed through the KubeBus@Cloud. KubeBus is L3 overlay network. As Figure 2, the KubeBus implements OSI network model's L2 and L3 over TCP Connection. The KubeBus Data Link Layer creates one or more long run TCP connections between Edge Node and KubeBus@Cloud. The KubeBus also includes one TUN interface which accepts the IP packet from OS kernel. The IP packet is passed through the L3 and L2 and send to the KubeBus@Cloud and then routed by the KubeBus@Cloud to target Edge Node through the long run TCP connection.

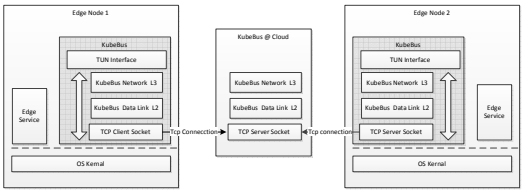

Figure 2: KubeBus Edge Node VPN

3.1.2 Connect Edge Node VPN with Container Network

KubeBus's second functionality is to make the Edge services running in same network as the Cloud application running in VM network or container network, such as flannel.

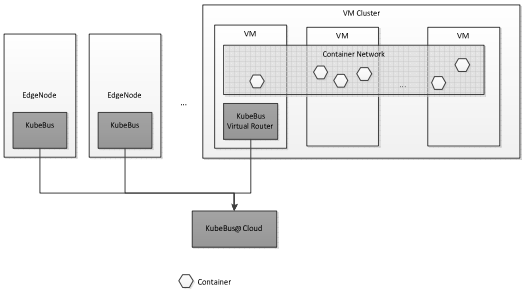

Figure 3: KubeBus Edge/VM/Container network VPN

In KubeEdge Environment, there are 3 subnets. All the Edge Nodes of one tenants belong to one Edge Node subnet, which is a VPN network created by KubeBus as in 3.1.1; in the VM cluster, all the VMs belong to VM subnet; all the containers running in the VM cluster belong to Container subnet. KubeBus will connect them as single VPN.

As figure 3, one KubeBus virtual router agent is installed on one of the VMs in the VM Cluster. The KubeBus virtual router agent includes the KubeBus network protocol stack and acting as router below Edge Node subnet and VM subnet. As the VM subnet could access the Container subnet, when the correct routes are configured in each VM node and Edge Node, the Edge Node subnet finally connects to the Container subnet (Note: for the cluster with address filtering policies, extra overlay is needed).

3.1.3 Multitenant management/data plane and service publish

KubeEdge is a multi-tenant infrastructure. The Edge nodes of different tenants are managed by the multitenant management/data plane. On one hand, there shall be communication between the management/data plane services and Edge Nodes of all the tenants; on the other hand, the Edge Nodes of one tenants shall be isolated from Edge Nodes of other tenant. So not as existing VPN solution [7], KubeBus builds an Http layer over KubeBus stack to enable the communication between Edge Node and multitenant management/data plane, based on which WebService running in Edge Node could be accessed from manage/data plane services and vice versa. In addition to that, the Http WebService running in Edge Node could also be published to Internet via the multitenant management/data plane. It is useful for some customer scenarios. For example, one Edge Node with video camera could publish the Http video stream as an Http WebService to KubeBus and expose that on the multitenant management/data plane, then the Http Client in Internet such as Web browser could get the video stream from the Http WebService running on the Edge Node.

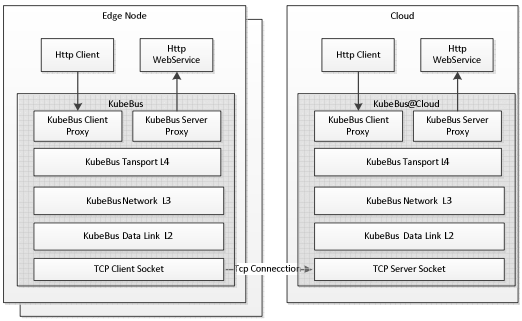

Figure 4: KubeBus Http Protocol Stack

As Figure 4, over the KubeBus's unreliable network layer 3, KubeBus transport layer 4 with reliable connection is built. The KubeBus L4 provides same API interface as TCP, such as listen/accept/connect/disconnect. Over the KubeBus L4, there are 2 "Http reverse proxies" running: KubeBus Client Proxy and KubeBus Server Proxy.

1) The KubeBus Client proxy listen on TCP port and when gets the Http client request, it will forward the request to the KubeBus Server proxy through KubeBus L4 reliable connection;

2) the KubeBus Server proxy listens on the KubeBus L4, and when gets the Http request forwarded by KubeBus Client Proxy, it will in turn forward the request to the target local Http WebService. After that, the Http response will be returned to Http client through Server proxy and Client proxy.

With the KubeBus Client and Server proxy, another "Http Layer over KubeBus" is built. Http WebService running at Cloud or Edge Node could be registered to KubeBus and the Http Client could access the WebService from Cloud, Edge Node or Internet.

Http WebServices of different tenants from Cloud or different Edge Nodes could be registered to KubeBus. One WebService is identified with <TenantId, Edge Node name, WebServiceName>. The TenantId is global identifier for a tenant; the Edge Node name is the tenant scope unique Edge Node name; the Web Service name is Node level unique service name. KubeBus uses a global identifier URL schema to identify the Http WebServices figure 5.

```
Http(s)://{hostname}/{Tenant Name}/{Edge Node Name}/{Http WebService Name}/…
```

Figure 5: KubeBus global identifier URL schema

The hostname is the KubeBus Client Reverse Proxy host name which could be accessed by the Http Client. In the Edge Node, the hostname is "localhost" and in the Cloud or Internet, the hostname is the public IP which host KubeBus service.

3.2 EdgeMetadataService

As Edge Node to Cloud is WAN, the connection might be not reliable and the bandwidth is low and expensive.

1) To address the WAN reliable issue, Edge services are required to running autonomously. The Edge service needs to re-run successfully even the Edge Node is offline after reboot. So the configurations for Edge services needs be persistent in Edge Node and the configuration could be synchronized from Cloud to Edge after the network connection restored;

2) To address the WAN's low bandwidth issue, the data volume transferred for configuration synchronized shall be minimized, so the incremental synchronization is preferred than the full snapshot synchronization. Not only the configuration needs to transfer from Cloud to Edge, the Edge services status also needs transfer from Edge to Cloud.

The Edge Metadata service is built for these 2 requirements. The Edge Metadata Service includes a metadata storage and synchronization service (SyncService) running in Edge side; there are another metadata storage are running in the Cloud side. The Edge service's configuration is written to the metadata storage in Cloud side to enable the configuration change process as a-sync process, i.e. even when the Edge is offline, the write will still succeed; after the network connection between Edge Node and Cloud restored, the Sync Service requests the configuration changes since last success synchronization from Cloud metadata storage and write to the local metadata storage. The synchronization shall be eventually consistent. So metadata storage needs to support atomic write and delta retrieval.

KubeEdge chooses Etcd [8] as metadata storage. It is because Etcd supports transactional write and has Multiple Version Concurrent Control API interface to retrieve the delta change, i.e. Get/Watch based on Revision.

```
Func SyncFromCloudToEdge(CloudStore, EdgeStore) {
        while true {
                LastSyncRevision = ReadLastSyncRevision(EdgeStore)
                Changes, NewRevision = ReadDelta(CloudStore, LastSyncRevision)
                TransactionalWrite(EdgeStore, Changes, NewRevision)
        }
}
```

Figure 6: Configuration Synchronization Algorithm

The algorithm of configuration synchronization from Cloud to Edge is as figure 6. The synchronization process is a loop. The loop has 3 steps. At first, the Last Sync Revision (LSR) of last successful synchronization is read from Edge Node local Etcd Store; second, the changes and the New Revision after the LSR is from Cloud Etcd Store; at last, both the changes and the New Revision is written to the Edge Node local Etcd Store. The New Revision in the write will become the LSR in next loop iteration. The algorithm can achieve the eventual consistent and atomic when the synchronization process crash anytime. Because only the changed data needs to be synchronized, the bandwidth consumed by synchronization is minimized. The algorithm of service status synchronization from Edge to Cloud is similar.

3.3 Kubernetes Edge Extension

In Kubernetes, there is one agent (Kubelet) running at each Node. It gets application configuration from KubeMaster, manages application life cycle and reports application's status to the KubeMaster. The Kubernetes Edge Extension is designed to address the WAN network connection specific character in Edge environment. It splits the Kubelet to 2 parts: EdgeController running in Cloud and AppEngine running at Edge Node. The EdgeController retrieves application configuration from KubeMaster with representation of Edge Node; then it will pass the changed configuration through the EdgeMetadataService to the AppEngine; after that, the AppEngine will handle the application life cycle based on the application configuration stored in the EdgeMetadataService. The AppEngine will also report the applications status to KubuMaster through EdgeMetadataService and the EdgeController. Because the data exchange between Cloud and Edge is built on the EdgeMetadataService, it could be done in a-synchronization model and only the change is passed to save the bandwidth.

4. EXPERIMENT

We evaluated the KubeEdge performance in the test environment as figure 7. The KubeMaster and VM1/VM2 is running in one subnet, the Edge1 and Edge2 are running in different subnets. All of them connect to KubeBus@Cloud through Internet. The KubueBus@Cloud is running in one CentralVM in AWS.

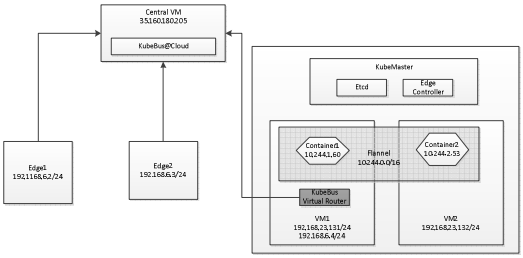

Figure 7: performance evaluation environment

We have tested the network latency between them with Ping. The test result is as figure 8.

	Latency (ms)
e1 --> Cloud	30.66
e1 --> e2	61.24
e1 --> VM1	56.86
e1 --> VM2	55.88
e1 --> Container1	60.68
e1 --> Container2	66.78

Figure 8: Ping latency in Edge environment

From the result, we can see the latency between the edges and VMs are mainly from the internet latency. The KubeBus contributes very little for the latency.

Based on that, we compares the deployment latency between to Edge Node and VM. We only measure the latency between the KubeMaster gets deployment request and the Kubelet/AppEngine get the result. The latency for VM is about 2 seconds and for the edge, the Edge is 3 seconds. We can see the latency of Edge's impact for the deployment delay is acceptable.

The hardware configuration is as

	Configuration
VM1 VM2	Amd64, 2 core, 4G memory, Ubuntue 16.04
Edge1	Respberry Pi 3 Model B, 1G memory, Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, Linux raspberrypi 4.14.0-v7+
Edge2	Amd64, 2 core, 4G memory, Ubuntue 16.04
CentralVM	Amd64, 1 core, 1G memory, Ubuntue 16.04

Figure 9: hardware configuration

KubeEdge builds a basic Edge infrastructure as an extension of Cloud. It has assumption that the Edge Nodes need to communicate with each other through Cloud. In reality, there might be direct connection between Edge Nodes and Edge Nodes could work as a cluster even when Cloud is offline. Following features are planned to enhance the functionality enable edge cluster.

1) Edge mesh network: The KubeBus create a VPN for Edge Nodes and Container network in Cloud. In current model, the inter-Edge-Node connection is going through KubeBus@Cloud. There might be direct connection between 2 Edge Nodes. For example, Edge Node might has public IP address so that another Edge Node could create a TCP connection to it directly; 2 Edge Nodes could also set up direct connection based on TCP hole punching technology; or there might be dedicated connection between 2 Edge Nodes. With the direct connection, the Edge network topology will be a de-centralized mesh network. KubeBus needs to support the Edge mesh network creation such as TCP hole punching and efficient packet routing in Edge mesh network.

2) De-centralized EdgeMetadataService: The Edge network will be a decentralized Edge mesh network. When the connection to the Cloud is not available, the connection between Edge Nodes might still work and the Edge Nodes shall still collaborate with each other and work efficiently. The De-centralized EdgeMetadataService is needed to enable the metadata exchange between Edge Nodes in Edge mesh network. For example, an Edge Node might join Edge mesh network by connecting to another Edge Node. The route to the new Edge Node shall be propagated to other Edge Nodes when the Cloud is offline with support of de-centralized EdgeMetadataService.

3) De-centralized Edge cluster: Based on the Edge mesh network, the Edge Nodes shall work autonomously when the central Cloud is offline as a cluster. The services running in one Edge Node could be re-scheduled to other Node for load balance, reliability with consideration of network topology, resource limitation and pre-configured task priority. De-centralized Edge cluster is needed to achieve this goal.

6. CONCLUSION

With more data is generated from Edge side, more computation which originally executes in Cloud will be push to Edge side. The Edge environment is different than Cloud with respect of network topology, connectivity and performance. Edge infrastructure is critical for push Cloud computation to Edge efficiently. This paper presents KubeEdge as one Edge infrastructure. The infrastructure is considered as an extension of Cloud infrastructure. It builds a similar Edge execution environment as Cloud by linking Edge Nodes and VM in Cloud as a single cluster. It could an alternative of the existing Pub/Sub based Edge infrastructure.

References

[1] "Kubernetes: Production-Grade Container Orchestration", https://kubernetes.io/, [Online; accessed Oct.1st, 2018].

[2]Quan Zhang, Xiaohong Zhang, Weisong Shi, "Firework: Big Data Processing in Collaborative Edge Environment", 2016 IEEE/ACM Symposium on Edge Computing (SEC).

[3] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, “Low latency geo-distributed data analytics,” in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. ACM, 2015, pp. 421–434.

[4] Y Kang, J Hauswald, C Gao, A Rovinski, T Mudge, J Mars, L Tang, “Collaborative Intelligence Between the Cloud and Mobile Edge”, CM SIGPLAN Notices 52 (4), 615-629.

[5] “AWS Greengrass Document”, https://docs.aws.amazon.com/greengrass, [Online; accessed Oct.1st, 2018].

[6] “Azure IoT Edge”, https://azure.microsoft.com/en-us/services/iot-edge/, [Online; accessed Oct.1st, 2018].

[7] “OpenVPN: Your private path to access network resource and service securely”, https://openvpn.net/, [Online; accessed Oct.1st, 2018].

[8] “etcd: A distributed, reliable key-value store for the most critical data of a distributed system”, https://coreos.com/etcd/, [Online; accessed Oct.1st, 2018].

[9] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, “Mqtt-s - a publish/subscribe protocol for wireless sensor networks,” in Communication systems software and middleware and workshops, 2008.