パブリックコンテナサービスを用いた超分散テストベッドの構築

董 允治[2,1]　中田 秀基[1,2,a)]　谷村 勇輔[1,2,b)]

概要：IoT センサの普及に伴いセンサデータの爆発的増大が想定される。このような環境ではエッジにおいて前処理を行うことでデータ量を低減するとともにクラウドでの処理を軽減するアプローチが有効であると考えられる。このような環境で動作するミドルウェアの負荷に対する特性を評価するには大規模なテストベッドが必要だが、実機でこのようなテストベッドを用意するのはさまざまな観点から現実的ではない。シミュレータを使用する方法も考えられるが、各モジュールへの負荷を検証することはできない。我々はクラウド上のコンテナサービスを利用することで、テストベッドを構築する方法を提案する。オーケストレーションサービスを用いることで容易に短時間で大規模なテスト環境を構築できることを確認した。

キーワード：Kubernetes, パブリッククラウド, 大規模テストベッド

A Prototype Implementation of Computing Continuum Testbed using Public Cloud Container Service

Yunzhi Dong[2,1]　Hidemoto Nakada[1,2,a)]　Yusuke Tanimura[1,2,b)]

Abstract: With the proliferation of IoT sensors, an explosive increase in sensor data is expected. In such an environment, an approach that reduces the amount of data by pre-processing at the edge and reduces processing in the cloud is considered promissing. To evaluate the characteristics and performance of the middleware which works in such an environment requires large-scale testbeds. Howerver, preparing such a testbed with actual computers and networks is not feasible in terms of cost. We propose a method to build a testbed by using container services in the public cloud. We have confirmed that a large-scale test environment can be easily built in relatively short time by using an orchestration service.

Keywords: Kubernetes, public cloud, large-scale testbed

1. はじめに

IoT センサの普及に伴いセンサデータの爆発的増大が起こりつつある。このような状況においては、センサからクラウドへ直接情報を送信するとクラウドに過負荷がかかることが予想される。これに対してセンサとクラウドの中間にエッジと呼ばれる層を追加し、エッジとクラウドで適切に負荷を分散することで、クラウドに負荷が集中するのを抑制しようという試みが提案されている [1][2]。我々は、このような試みの一貫として、センサからのデータをエッジで集約することでクラウドへの負荷を低減するシステムを提案し、その実現性を検討してきた [3]。

しかしこのようなシステムの大規模な環境での評価は容易ではない。多数のノードから構成される実験環境を確立することはそれ自体技術的にも経済的にも困難である。Grid5000[4] のような例はあるが、維持管理のコストは膨大で、持続可能ではない。SimGrid[5] や GridSim[6] など

[1] 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
[2] 筑波大学
University of Tsukuba
[a)] hide-nakada@aist.go.jp
[b)] yusuke.tanimura@aist.go.jp

のシミュレータを利用する方法も考えられるが、多くのシミュレータはネットワークのみに着目しており、個々のノード上で動作するモジュールの過負荷を評価することはできない。

これに対して、我々はクラウド上のコンテナオーケストレーションサービスを利用することで、テストベッドを構築する方法を提案した [7][8]。本稿では、このアプローチをさらに進め、テストベッドの構築と破棄を自動化し、Jupyter Notebook[9] から制御する方法を提案する。提案システムでは、パブリッククラウド上へのテスト環境を構築し、実験を実行し、テスト環境を破棄する作業を手元のPC 環境から容易に行う事ができる。

本稿の構成は以下のとおりである。2 節では、本稿で用いる Kubernetes や Amazon EKS、テスト対象となる MQTT に関して説明する。3 節では、提案システム上で構成するテストベッドについて概説する。4 節で、提案システムの構成について説明し、5 節で本稿をまとめ、将来の課題を述べる。

2. 背景

2.1 Kubernetes

2.1.1 Kubernetes の概要

Kubernetes[10] は、複数のコンテナを管理するコンテナオーケストレータの一つで、デファクトスタンダードとして広く用いられている。コンテナオーケストレータは、個別の機能を持つ多数の小さいサービスを組み合わせることで一つのアプリケーションを構成するマイクロサービスが普及とともに一般化した。コンテナとはアプリケーションもしくはサービスを、その依存する環境とともにパッケージしたものだ。コンテナオーケストレータは、このようなコンテナの集合を一体として管理監視し、起動終了を行う。たとえば、故障したコンテナがあれば、同じ機能を持つ別のコンテナを自動的に起動するセルフヒーリング機能を持つ。

Kubernetes はコントロールプレーンとワーカーノードから構成される。コントロールプレーンには、ユーザからの入力を受け付ける API サーバや状態を管理するデータベースに相当する etcd、クラスタの情報を収集するコントロールマネージャ、コンテナを実行するノードを決定するスケジューラが存在する。

個々のワーカノードには複数のポッドを動作させる事ができる。ポッドは個別の IP アドレスを持つ単位で、ポッドの中にさらに複数のコンテナを持つ事ができる。この様子を図 1 に示す。

Kubernetes はさまざまなコンテナエンジンをサポートするが、本稿では Docker[11] を用いる。

2.1.2 クライアントツール

Kubernetes ユーザはコントロールプレーンの API サーバに接続する何らかのクライアントツールを用いて、Kubernetes クラスタを制御する。コマンドラインツールの `kubectl` がよく用いられるが、API ライブラリや UI ツールも存在する。

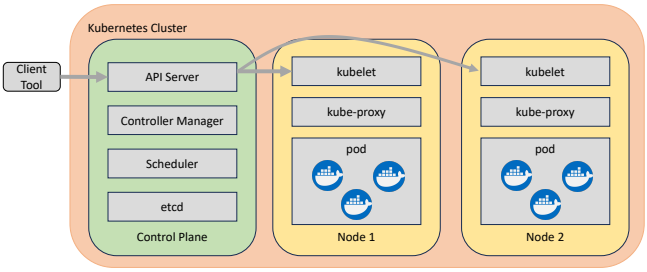

図 1 Kubernetes の概要

2.1.3 ConfigMap と Secret

例えばアクセス先の IP アドレスや、認証情報をコンテナイメージに組み込むと、これらが変更されるたびにコンテナイメージのリビルドが必要になり非効率である。Kubernetes では、ConfigMap と Secret と呼ばれる機能を用いることで、コンテナに対して起動時に外部から設定情報を与えることができる。

2.1.4 DaemonSet と Deployment

DaemonSet を用いると、すべて (もしくは一部) のノードで自動的に実行するポッドを指定することができる。例えばログ収集やノードを監視するデーモンなどはすべてのノードで動作することが望ましい。このようなデーモンを実行するポッドを自動起動するものが DaemonSet である。

Deployment は同一のサービスのレプリカを複数起動するために用いられる。複数のレプリカをノードにまたがって起動することで負荷分散を行うことができる。また、耐故障性のためにも重要な機能である。

2.2 Amazon EKS

Kubernetes はオンプレミス環境でも広く使用されているが、小規模な組織で運用するのはそれほど容易ではない。これに対してパブリッククラウド上で Kubernetes をサポートするサービスが登場している。Amazon EKS(Elastic Kubernetes Service)[12] はその一つで、Kubernetes を Amazon クラウド上で実行するサービスである。同様に、Google Computing Services には Google Kubernetes Engine(GKE)[13] が、Azure には Azure Kubernetes Service(AKS)[14] が存在するが、本稿では Amazon EKS を用いた。

EKS ではノードを通常の EC2 上の仮想計算機もしくは Fargate[15] 上に構築する。EKS を用いることで、非常に大規模な実験環境を容易に管理運用することができる。EKS の課金は 1 クラスタあたり 1 時間 0.1 ドルと安価である (もちろんワーカノードには別途課金される)。

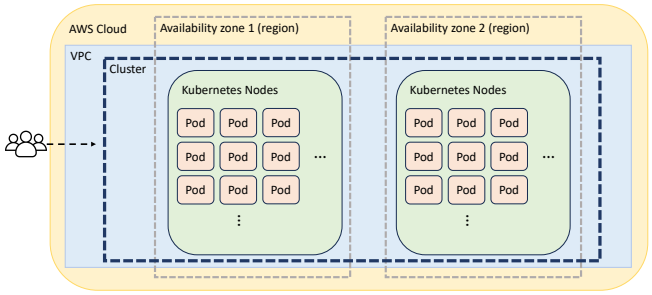

図 2 Amazon EKS の概要

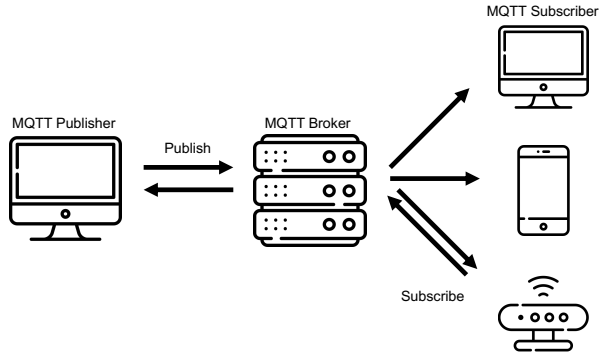

図 3 MQTT の概要

2.3 MQTT

MQTT[16] は、Publish/Subscribe モデルに基づく軽量な通信プロトコルであり、広く普及している。MQTT の通信には、Publisher, Subscriber, Broker の 3 者が関与し、Broker を中心としたスター構造の通信トポロジをとる図 3。Publisher はデータの送信者であり、特定のトピックを指定して Broker にメッセージを送信する。Subscriber はデータの受信者であり、Broker に対して特定のトピックに対する受信希望を行う。Broker は、Publisher からのメッセージを受信すると、そのメッセージで指定されているトピックに対して受診希望を行っている Subscriber に、そのメッセージを送信する。Broker を介することで多対多の柔軟な通信が可能となる。

MQTT には 3 レベルの QoS(Quality of Service) が用意されている。Publisher は送信時に QoS を指定することができる。QoS 0 はベストエフォートによる送信で、到達性は保証されない。つまりメッセージは途中で破棄される可能性がある。QoS 1 は At-least-once(少なくとも 1 回) の到達を保証する。再送を行うため受信通知が失われた場合には複数回メッセージが配送される可能性がある。QoS 2 は exactly-once(厳密に 1 回) の到達を保証する。送信側が受信通知に反応するまでライブラリがメッセージをユーザに引き渡さないことによって、複数回の配送を抑制する。

3. 超分散テストベッド

本節では、本稿で構築する超分散テストベッドを概説する。我々は膨大の数のセンサから得られる情報をクラウドに集積することに興味を持っている。これをナイーブに実装するとクラウドの負荷が非常に高くなることが予想される。これに対応するために、センサ群を地理的に局所性を持つグループに分割し、グループ内で一旦集積および集約してからクラウドに送信する方法が考えられる (図 4)。

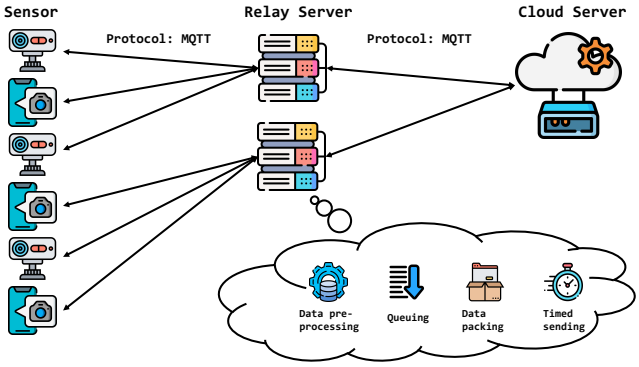

図 4 超分散テストベッドの概要

この中間で集積するサーバを中継局が設置されるエッジに配置し、デバイス、エッジ、クラウドの 3 階層で情報を収集する。エッジで行う集約としては、単純にメッセージを集約してメッセージ数を減らすことや、平均値や最大最小値としてサマライズするなど、アプリケーションに応じてさまざまな方法が考えられる。

4. 提案システム

4.1 本稿で構築する環境の概要

本稿で提案するシステムの目的は、3 節で述べた環境を擬似的にパブリッククラウド内に構築することである。

具体的には、センサを模した Publisher(以降 Sensor と呼ぶ) からの送信を中間層の Broker でローカルに集約し、さらにクラウドを模した最終層の Broker に集約する 2 段階の構成を考える一段目の Broker を EdgeBroker、2 段目の Broker を CloudBroker と呼ぶ。1 つの EdgeBroker が集約する Sensor の数を M、EdgeBroker の数を N とする。系全体としては $M \times N$ 個の Sensor が存在することになる。EdgeBroker の背後には、中継と集約を行うポッド (以下 Relay と呼ぶ) を配置する。この Relay は EdgeBroker に対しては Subscriber として振る舞い、CloudBroker に対しては Publisher として振る舞う。つまり、EdgeBroker からメッセージを受け取り、集約を行った後に CloudBroker に送信する。CloudBroker に対する Subscriber(Receiver と呼ぶ) が最終的にメッセージを受け取る。

4.2 Kubernetes へのマッピング

負荷分散のために、各エッジ環境を 1 つのノードを割り当てる。またクラウド環境にも 1 つのノードを割り当てる。エッジ環境は N 個あるので、ノード数は $N+1$ となる。

コンテナの種類としては、下記の 4 種類を用意した。

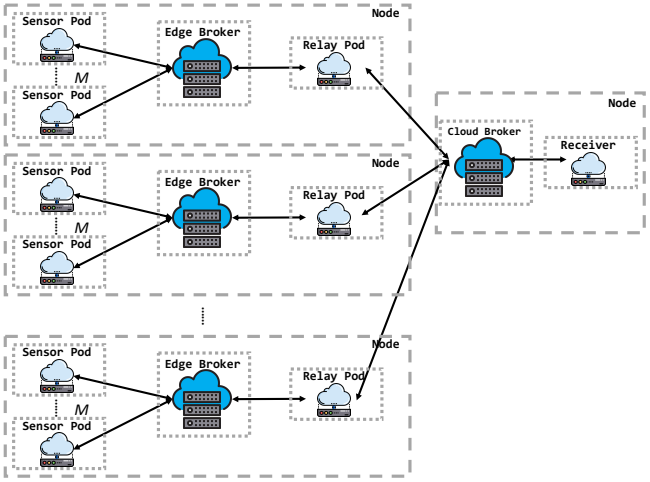

図 5 実験環境の構成

- EdgeBroker および CloudBroker
- Sensor
- Relay
- Receiver

この様子を図 5 に示す。エッジ環境のノードには、M 個の Sensor ポッドと EdgeBroker ポッド、および Relay ポッドが動作する。クラウド環境ノードでは、CloudBroker ポッドと Receiver ポッドが動作する。

すべてのノードで EdgeBroker もしくは CloudBroker を動作させるので、ここでは DaemonSet を用いている。これによってノードを起動させるだけで自動的に Broker が立ち上がる (EdgeBroker と CloudBroker は同じイメージで作成している)。

また、Sensor の起動には Deployment を用いた。レプリカ数を指定することで同一のノード内に指定した数の Sensor を起動することができる。このような利用方法は、Deployment が本来意図したものではないが、管理は容易になる。

4.2.1 証明書の取り扱い

MQTT は TLS 証明書に基づく認証と暗号化をサポートしており、インターネット環境で使用する際にはこれがデフォルトとなる。今回の実験ではクライアント認証は行わずサーバ (Broker) 側のみで認証を行った。

サーバ認証を行うには、サーバ側には CA(Certificate Authority) の証明書とサーバの秘密鍵ファイルを、クライアント側には CA の証明書を配置する必要がある。本来であれば秘密鍵ファイルをサーバから動かすことはできないため、外部に CA を設置し、サーバで CSR を作成して CA に送付し、CA で作成した証明書をサーバに送り返す、という手順が必要だが、これは煩雑である。

クラウドに閉じた環境では実際にアクセスされる危険性はなく、TLS による負荷の上昇だけを検証することが TLS を使用する目的であるため、今回は単純化した手続きを採用した。具体的には、手元の PC 上に CA を作成し、その CA を用いてサーバ証明書と秘密鍵のペアを 1 つだけ作り、それらをすべてのポッドで共有した。ポッドでの共有には前述した Secret を使用した。

```
1  apiVersion: apps/v1
2  kind: DaemonSet
3  metadata:
4    name: broker-pod
5  spec:
6    selector:
7      matchLabels:
8        app: broker
9    template:
10     metadata:
11       labels:
12         app: broker
13     spec:
14       volumes:
15       - name: secret-volume
16         secret:
17           secretName: broker-certs
18       containers:
19       - name: broker-pod
20         image: XXXXX/YYYY:latest
21         volumeMounts:
22         - name: secret-volume
23           mountPath: /etc/secrets/broker-certs
24         command:
25           - /bin/sh
26           - -c
27           - >
28             中略
29             /usr/sbin/mosquitto -c \
30             /etc/mosquitto/mosquitto.conf
31         ports:
32         - containerPort: 8883
33           protocol: TCP
34
```

図 6 EdgeBroker 兼 CloudBroker の yaml ファイル

4.3 Broker の IP アドレスの解決

Broker にアクセスするためには、Broker コンテナをホストしているポッドの IP アドレスを取得する必要がある。ポッドの IP アドレスは自動的に採番されるため、ユーザが制御する事はできない。

このため本システムでは Broker ポッドを起動した後で、Kubernetes に問い合わせてポッドの IP アドレスを取得し、その値を設定したポッド yaml ファイルを動的に生成し、Sensor ポッド等を起動している。

4.4 Broker の yaml ファイル

例として、Broker に用いた yaml ファイルの一部を図 6 に示す。2 行目で DaemonSet を用いることを宣言している。20 行目で使用するコンテナイメージを指定している。こ

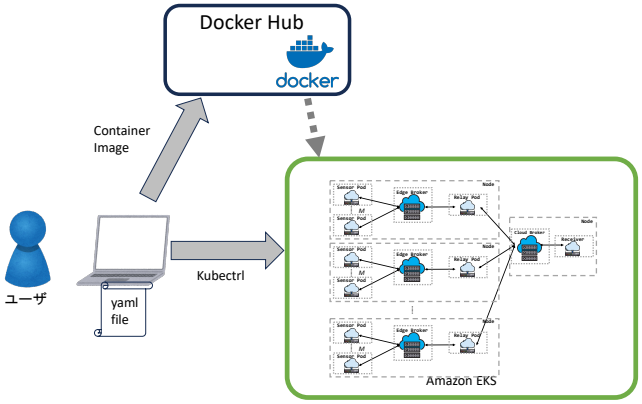

図 7 実験の実行

こで指定するイメージはあらかじめ Docker Hub にアップロードしておく必要がある。証明書を Secret に登録しているので、14-17 行目で Secret のボリュームを指定し、21-23 行目でそれを特定のパス (`/etc/secrets/broker-certs`) にマウントしている。24-30 行目でコンテナ内で起動するコマンドを指定している。ここでは MQTT のブローカ実装である Mosquitto[17] を起動している。32-33 行目で、外部からアクセスを受け付けるポートを指定している。8883 は MQTT が SSL/TLS を用いた暗号化通信に用いる標準のポート番号である。

4.5 Amazon EKS での実現

Amazon EKS に対する操作には Amazon の提供する Python API である Boto3[18] を使用した。Amazon EKS のワーカノードを作成するには、EC2 を用いる方法と fargate を用いる方法がある。今回は EC2 を用いたため、事前にノードグループと呼ばれるものを定義し、最大ノード数と最小ノード数を設定する必要がある。EKS が必要に応じて EC2 インスタンスを起動してくれる。

Kubernetes の操作は、Kubernetes の Python 用 API[19] を使用して実装した。コンテナイメージの保存には Docker Hub[20] を用いる。実験環境の概要を図 7 に示す。

4.6 実験の手順

実験の過程は以下のようになる。

- 実験を実行するコードを記述する。
  例えば Sensor ではメッセージを特定のトピックで送出し、Receiver は特定のトピックから受信する。Relay は EdgeBroker からメッセージを受け取りそれを集積して、CloudBroker に送出する。
- 上記の実験コードを含むコンテナイメージを作成し Docker Hub に登録する。
- 必要な数のノードを持つ EKS 環境を作成する。
- Jupyter Notebook に書かれた手順にしたがって、EKS 上に実験環境をセットアップする。
- 実験が完了したらログを取得する。
- EKS 上の実験環境を破棄する。
- EKS 環境を破棄する。

5. おわりに

本稿では、大量のセンサデータを収集する超分散システムの評価を行うテストベッドをパブリッククラウド上のコンテナオーケストレーションサービスを用いることで自動化するシステムを提案した。本システムを用いることで、大規模な環境の実験が比較的容易に行うことができることを示した。

今後の課題としては、以下が挙げられる。

- より大規模な環境の実験
  これまでに 100 センサーからなる環境の構築をテストしている。より大規模な環境に対してもスケールすることを確認する。
- Kubernetes クラスタ構成手法の見直し
  現在の IP アドレスの設定手法は Kubernetes の外部で作業を行っており、望ましくない。Kubernetes 内部で完結する手法を検討する必要がある。
- ネットワークのエミュレーション
  提案システムのコンテナ間のネットワーク接続は、ネイティブの速度で動作するが、より詳細な評価を行うにはネットワークのエミュレーションが必要になる。例えば帯域制約やレイテンシのインジェクションなどを検討する。

謝辞 本成果の一部は、国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の「ポスト 5 G情報通信システム基盤強化研究開発事業」（JPNP20017）の委託事業の結果得られたものである。

参考文献

[1] Yousef, A. and Fung, C.: All one needs to know about fog computing and related edge computing paradigms: A complete survey, *Journal of Systems Architecture*, Vol. 98, pp. 289–330 (2019).
[2] Lopez, P. G. and Montresor, A.: Edge-centric Computing: Vision and Challenges, *ACM SIGCOMM Computer Communication Review*, Vol. 45, p. 37– (2015).
[3] 董 允治, 中田秀基, 谷村勇輔: ネットワークエッジを活用したデータ収集システムに向けた MQTT の性能計測, 情報処理学会研究報告, Vol. 2022-OS-158(12) (2023).
[4] Cappello, F., Caron, E., Dayde, M., Desprez, F., Jegou, Y., Primet, P., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quetier, B. and Richard, O.: Grid'5000: a large scale and highly reconfigurable grid experimental testbed, *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.*, pp. 8 pp.– (2005).
[5] Casanova, H., Giersch, A., Legrand, A., Quinson, M. and Suter, F.: Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms, *Journal of Parallel and Distributed Computing*,

Vol. 74, No. 10, pp. 2899–2917 (online), available from ⟨http://hal.inria.fr/hal-01017319⟩ (2014).
[6] Buyya, R. and Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, *Concurrency and Computation: Practice and Experience*, Vol. 14 (online), DOI: 10.1002/cpe.710 (2002).
[7] 董　允治, 中田秀基, 谷村勇輔：ネットワークエッジを活用した大規模データ収集システムのテスト環境構築と最適化の検討，情報処理学会研究報告，Vol. 2022-HPC-191(11) (2023).
[8] Dong, Y., Nakada, H. and Tanimura, Y.: Cloud-based Testbed for Large-scale data Collection System with Network-Edge, *Proceedings of IMCOM 2024, the Annual International Conference on Ubiquitous Information Management and Communication* (2024).
[9] : Jupyter Project, `https://jupyter.org/`.
[10] : Kubernetes: Production-Grade Container Orchestration, `https://kubernetes.io/`.
[11] : Docker, `https://www.docker.com/`.
[12] : Amazon Elastic Kubernetes Service, `https://aws.amazon.com/eks/`.
[13] : Google Kubernetes Engine (GKE): The most scalable and fully automated Kubernetes service, `https://cloud.google.com/kubernetes-engine`.
[14] : Azure Kubernetes Service (AKS), `https://azure.microsoft.com/en-us/products/kubernetes-service/`.
[15] : AWS Fargate, `https://aws.amazon.com/fargate/`.
[16] : MQTT: The Standard for IoT Messaging, `https://mqtt.org/`.
[17] : Eclipse Mosquitto, `https://mosquitto.org/`.
[18] : Boto3 - The AWS SDK for Python, `https://github.com/boto/boto3`.
[19] : Kubernetes Python Client, `https://github.com/kubernetes-client/python`.
[20] : Docker Hub: Develop faster. Run anywhere., `https://hub.docker.com/`.