

G-05

スケーラビリティと高可用性を有するネットワークエミュレータの検討 Consideration of network emulators with scalability and high availability

菅家 悠希†
Yuki Sugaya

井口 信和‡§
Nobukazu Iguchi

1. 序論

ネットワークの仮想化が進んでいる[1]. ネットワークの仮想化により, これまで複数台の物理的なネットワーク機器を用いて構築していたネットワークが, 物理的なサーバを1台用意するだけで構築可能となった. ネットワークの仮想化によりルータ, スイッチ, ファイアウォールといったネットワーク機器やLANケーブルといった物理的な機材に依存することなく, 柔軟に構築することが可能となる.

仮想化による仮想ネットワークは様々な用途に利用可能である. 例えば, インフラ業務においてネットワークの管理運用の時にテストが実施される. このテスト時に仮想ネットワークを用いることで, 実際のネットワーク上で起こる様々な事象が再現可能となり, 有用な運用テストの実施が可能となる. さらに, アプリの試運転による性能評価や, 仮想ネットワークを用いて構築体験を実施するための学習用途などにも利用可能となる.

仮想ネットワークには様々な利点があるが, その一方で課題も存在する. ネットワーク機器の仮想化に伴う仮想ネットワーク機器 (以下, 仮想機器) のリソース消費に伴う物理サーバへの影響は解決すべき課題である. さらに仮想機器へのリソースの割り当てが非適切だと, 仮想機器自体の動作に影響を及ぼすため, 仮想機器の状態を一元管理しリソースを適切に割り当て, 仮想機器の状態に応じた処理を行うことで, 常に稼働状態にしておく必要がある.

そこで, 本研究では上記の課題を解決することで, 仮想機器による物理サーバへの影響を軽減し, さらに仮想機器の状態を一元管理することで, 仮想機器の安定した動作を実現し, システムの可用性を向上させることを目的とし, コンテナオーケストレーションツールである Kubernetes¹ を基盤とするネットワークエミュレータを開発する. 本エミュレータでは Kubernetes により生成したコンテナ (Pod) を利用し仮想機器を実現する. Kubernetes の他に Docker Swarm 等のコンテナオーケストレーションツールの利用も検討したが, 自動スケーリングといった機能面が豊富であることに加え単一コンテナ故障時のフェイルオーバー時間において Kubernetes に優位性があるため本研究では Kubernetes を採用した[2].

Kubernetes を利用することで, サーバの分散管理や拡張性が向上することが期待できる. さらに Kubernetes のコンポーネントにより Pod を常に監視することが可能となり, 安定した仮想ネットワークを実現可能となることから, システムの可用性向上が期待できる.

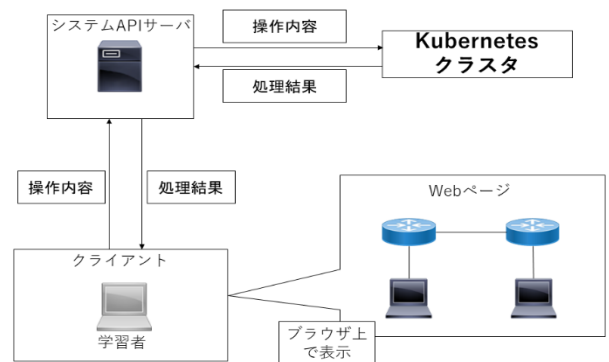


図1 システム構成

2. 関連研究

関連研究として立岩らの研究[3]や新村の研究[4]がある. 立岩らの研究では UML という仮想化技術を利用することで, 仮想ネットワーク上でネットワーク構築学習を実施可能とする. また, 設定したネットワーク上で流れるパケット等を可視化することで, ネットワーク構築の学習を促進する. 新村の研究では Docker² という仮想化技術により仮想ネットワークを構築し, 仮想ネットワーク上で ospf やセグメントルーティング演習などが実施可能である. 仮想化技術として Docker を利用することのメリットとして, コンテナの性質として起動速度やリソース消費に関して仮想マシンよりも優位性がある. さらに, Docker には特定の環境を有する Docker イメージファイルを共有するためのサービスとして Docker Hub というものがあり, 本サービスを通じて, 任意のサービスを実現するための Docker イメージを入手可能となり, 用途に応じてルータやスイッチ, データベースなどといった機能を有するコンテナを容易に構築可能となる.

これらの研究に対して本研究では, Kubernetes により生成した Pod を利用することで実現した仮想機器を用いて仮想的なネットワークを構築し, インフラ業務におけるネットワーク運用テスト等を実施可能な環境を利用者に提供する.

3. 検討システム

本システムの構成を図1に示す. 本システムはクライアント, システム API サーバによって構成される. クライアントは Web アプリケーションとして実装しており, 利用者は Web アプリケーションを通じてシステムを利用することが可能である. システム API サーバはクライアントからの操作を受け付け仮想ネットワークに対して設定を発行するために利用される. 仮想ネットワークは Kubernetes クラスタ上で Pod を展開することで実現する.

† 近畿大学大学院総合理工学研究科, Graduate School of Science and Engineering Research, Kindai University

‡ 近畿大学情報学部, Faculty of Informatics, Kindai University

§ 近畿大学情報学研究所, Cyber Informatics Research Institute, Kindai University

本システムにおける Kubernetes クラスターの構成を含む詳細なシステム構成を図2に示す。本システムの Kubernetes クラスターは、3 台のサーバ (ノード) からなり、システム全体を管理するための Master ノード 1 台と、配置された仮想機器を動作させるための Worker ノード 2 台からなる。システム API サーバは Kubernetes の api を通じて様々な操作を発行する。図2で示した Kubernetes のコントロールプレーンコンポーネントは Kubernetes 公式ドキュメントに基づき設計した。

本システムにおける GUI を図3に示す。GUI は機器選択パネル、ネットワークキャンパス、コンソールに分かれている。機器選択パネルで追加したい機器を選択し、ネットワークキャンパスにドラッグ&ドロップすることで機器が追加される。追加した機器間の結線処理は結線ボタンを押下し、接続元機器と接続先機器を選択することで可能となる。追加した機器に対する設定はコンソールを利用することで CLI ベースで設定が可能である。

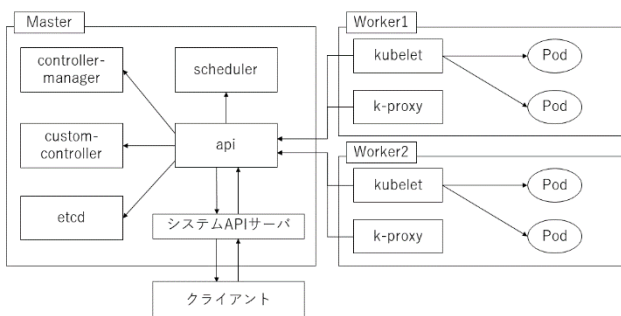


図2 システム構成 (詳細)

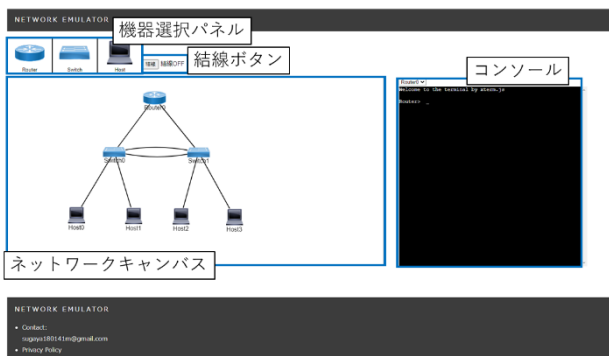


図3 システム GUI

3.1. サーバへのリソース問題への対処

仮想ネットワークを実現する上での課題である、仮想機器のリソース消費に伴う物理サーバへの影響の解決策として Kubernetes を利用しサーバの分散管理を可能とすることでサーバ 1 台への集中的な負荷を軽減する。本設計では図2のように、Master ノード 1 台と Worker ノード 2 台でクラスター構築しているが、リソースの消費に応じてサーバの拡張が求められる、その場合図4のようにクラスターに新たなノードを追加し、Worker ノードとして稼働させることで容易に水平スケーリングが可能となる。さらにシステム API サーバと仮想機器を動作させるための Worker ノードを分離することで、仮想機器のリソース消費による影響がシステム全体に生じない設計とした。例えば Worker1 が仮想機器による膨大なリソース消費によって停止し、応答不能状態になったとしても、Master ノードには影響を与えないためシステ

ム自体は稼働し続ける、さらに Kubernetes のコンポーネントによってノードの停止を検知することで、別のノードへ Pod の再割り当てが可能となる。

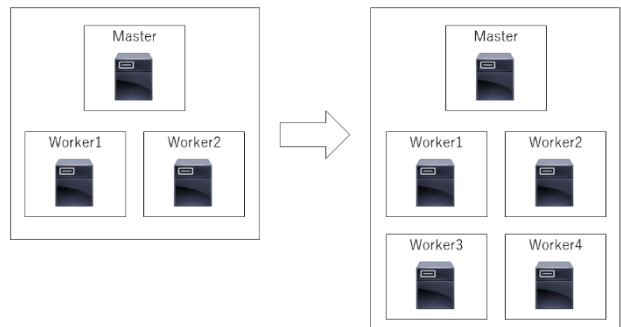


図4 ノードの水平スケーリング

本設計における課題として異なるノード上に存在する Pod 同士の疎通問題がある。本来は 1 つのノード上に仮想機器を動作させるため、各機器間の通信は Linux Bridge 等を利用しローカル通信により疎通が可能であった。しかしながら、別ノード間の疎通に関してはローカル通信を利用できないことから、トンネリングといった、別のアプローチが必要となる。本システムでは別ノード間の疎通を実現するために l2tp を利用することで実現する。

本システムにおける別ノード間の疎通について図5を用いて説明する。図5のシナリオでは Worker1 上にある Pod1 の eth0 インタフェースと Worker2 上にある Pod2 の eth0 インタフェースの接続を示している。まず Pod に対してインタフェースを複数生成するために Multus CNI という CNI プラグインを使用する。今回は Multus CNI により各 Pod に 3 つのインタフェースが生成されている。そして Multus CNI によって生成されたインタフェースと Linux Bridge を接続し、その Bridge 同士を l2tp で接続することで、別ノード間の疎通が実現可能となる。

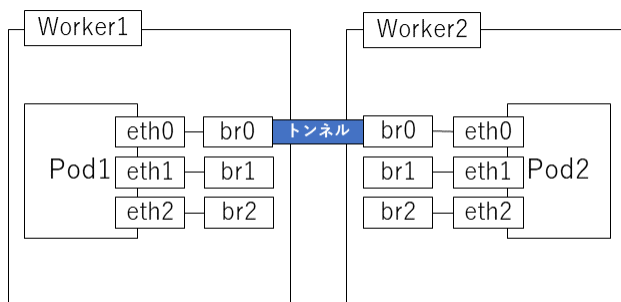


図5 別ノード間の疎通について

3.2. 仮想ネットワークの可用性向上への方針

仮想機器の可用性を向上させるための方法として Kubernetes のコントローラを利用することで、仮想機器へのリソース制御に加え仮想機器が故障したときに即座に検知し Pod の再割り当てが可能となる。図6に Pod が故障したときにおける Pod の再割り当てのイメージを示す。Kubernetes クラスター上で Pod1 と Pod2 が動作しているとする。これらの Pod は Kubernetes のコンポーネントである controller によって監視される。例えば Pod1 が故障した時、controller によって故障が検知され、即座に既存のものに代わる Pod1 が割り

当てられる。仮想機器の故障時に即座に検知し対処することで仮想ネットワークにおける可用性の向上が期待できる。

さらに、Kubernetesのカスタムリソースを利用したBPS[5]を利用することでバックアップ用のPodを作成することが可能である。図7にPodのバックアップ手順について示す。図7のように障害発生時にプライマリPodをバックアップPodを即座に代えることで可用性の向上が期待できる。

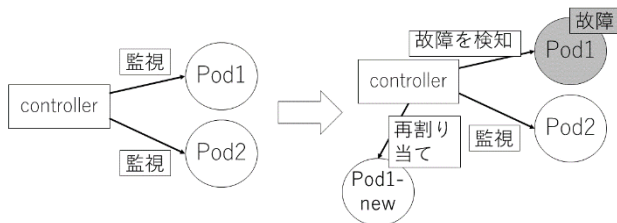


図6 Pod再割り当てのイメージ

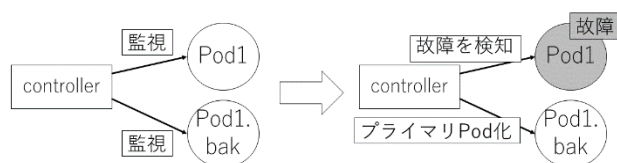


図7 Podのバックアップについて

今後、仮想機器の実装を進めることで、各インタフェースの状態やIPアドレス、結線情報などの独自のフィールドを管理することが予想される。例えば仮想ネットワーク上のルータが1台故障してしまった場合、即座に故障を検知し、Podを再割り当てした後、ルータに設定されていた内容を復元し、さらには結線状況を復元する必要がある。ルータが持つべきフィールドの一例について図8を用いて説明する。図8はルータを実現するPodの内容を定義したマニフェストファイルの一例である。ルータを実現するためにFRRoutingを利用する。Kubernetesのカスタムリソースを定義することでルータ独自のフィールドを持つことが可能となる。specフィールドがルータ独自のフィールドとなっており、インタフェースeth0及びeth1に対応する情報として、ipアドレスやインターフェースの状態、結線情報として対向インターフェース情報などを持つ。このようにして、今後、KubernetesのCustom Controllerを作成し独自のフィールドに対するふるまいを定義することで、仮想機器に対してより柔軟な制御が期待できるため今後検討を進めていく。

4. 実験

実験では性能評価実験を予定している。リソース消費問題に関する実験として、仮想機器の台数が増加したときのサーバの負荷を計測し、サーバの分散によりどれだけサーバ全体のパフォーマンス向上に繋がったかどうかを評価する。可用性に関する実験として、任意のPodが故障したときに、Podが正しく復元できるかどうかを確認する。さらに復元が可能であった場合は復元にかかった時間を計測し、本システムにおける仮想ネットワークの可用性を確認する。

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-router
spec:
  ###独自のフィールドを定義###
  router:
    int:
      eth0:
        ip_address
        status
        target
      eth1:
        ip_address
        status
        target
  containers:
  - name: frr
    image: frrouting/frr:v8.1.0
    securityContext:
      privileged: true
```

図8 ルータに定義するマニフェスト例

5. 結論

本研究では、ネットワークエミュレータを開発するにあたっての、仮想ネットワークの有用性及び課題について問題提起し、課題に対する解決方法について検討した。Kubernetesを基盤とした仮想ネットワークを構築することで、仮想ネットワークの構築時に生じる課題を解決することが可能となる。本ネットワークエミュレータを通じて、インフラ運用テスト等の実施が容易となることが期待できる。

参考文献

- [1] 総務省 情報通信審議会,情報通信技術分科会, I P ネットワーク設備委員会: 第三次報告 (案)
—IoTの普及に対応した電気通信設備に係る技術的条件—, 入手先<https://www.soumu.go.jp/main_content/000674158.pdf>.
- [2] I. M. A. Jawarneh et al., "Container Orchestration Engines: A Thorough Functional and Performance Comparison," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1-6, doi: 10.1109/ICC.2019.8762053.
- [3] 立岩佑一郎, 安田孝美, 横井茂樹: 仮想環境ソフトウェアに基づくLAN構築技能とTCP/IP理論の関連付け学習のためのネットワーク動作可視化システムの開発, 情報処理学会論文誌, Vol.48, No.4, pp1684-1694 (2007).
- [4] 新村 正明, 大規模ネットワーク経路制御技術演習のためのオンライン演習システムの提案, 教育システム情報学会誌, 39 巻, 2 号, p. 303-308, (2022)
- [5] M. Zhu, R. Kang, F. He and E. Oki, "Implementation of Backup Resource Management Controller for Reliable Function Allocation in Kubernetes," 2021 IEEE 7th International Conference on Network Softwarization (NetSoft), 2021, pp. 360-362, doi: 10.1109/NetSoft51509.2021.9492724.¹

¹ Kubernetes, 入手先<<https://kubernetes.io/ja/>>, (参照:2022-07-17) .

² Docker, 入手先 <<https://www.docker.com/>>, (参照:2022-07-17)