プロキシ先検索と設定ファイル作成の自動化によるNginxサーバ構築の簡略化

栗原 尚希[1]　大野 有樹[2]　串田 高幸[1]

概要：Web サーバのアクセス増加に伴い，Nginx ベースのリバースプロキシの導入を試みることがある．その際に導入作業を行うユーザが Nginx に詳しいとは限らない．課題は Nginx ベースのリバースプロキシを Nginx 初学者が構築した場合に，作業時間が長時間となることである．この課題に対してリバースプロキシ構築作業のうち，設定ファイルに書き込む情報の取得，設定ファイルの作成，コンテナの生成，これらの工程を行うソフトウェアを提案した．本提案の評価するために，Nginx ベースのリバースプロキシの構築を手作業の場合と提案ソフトウェアを使用した場合でそれぞれ行い比較した．結果として手作業で構築した作業時間の平均は 3,176 秒，提案ソフトウェアを使用した作業時間は 122 秒となり，3,054 秒の削減となった．

1. はじめに

背景

ユーザからのリクエストを受け取り，それを処理し，ユーザにレスポンスを返すコンピュータのことを Web サーバという．その中でも，Nginx はその高い処理性能を特徴として開発されており，同時に多数の処理を高速に実行することが可能である[*1]．これは大量のユーザからのリクエストを効率的に処理するために必要な機能であり，そのために Nginx は多くの Web サーバの中でシェアを増やしている．

また，Nginx はリバースプロキシ機能も持っている．リバースプロキシはクライアントと Web サーバ間の通信を介在し，Web サーバの応答を代理して通信を中継する機能である．これにより Web サーバの負荷を軽減したり，セキュリティを強化したりすることが可能になる [1, 2]．リバースプロキシの対になるものとしてプロキシと言われるものが存在し，これはクライアント側の Web ブラウザから Web サーバへのリクエストを代理する．プロキシの導入には，クライアントは Web サーバ側に自身の IP アドレスを隠すことによる匿名性の保持や，コンテンツ表示の高速化，アクセスログを取得するといった役割がある．それに対してリバースプロキシは Web サーバ側が行うクライアントへの応答を代理する．リバースプロキシはセキュリティ対策，性能向上，負荷分散，システム構築の自由度向上などの目的で利用される．

本稿では，Nginx を仮想マシン（VM）に直接インストールするのではなく，コンテナ上で稼働させることを前提としている．これは，コンテナ技術が VM と比較して，起動や管理に必要なリソースのオーバーヘッドが少なく，軽量で高速かつ高い移植性を備えた仮想化技術であるからである．さらに，コンテナはアプリケーションとその依存関係をパッケージ化し，異なる環境でも一貫した動作を実現できるため，アジャイル開発によるサービスレベルの向上や DX の推進を目指す企業で広く利用されている [3]．

コンテナはアプリケーションの実行機能を備えているが，コンテナの管理や他のサーバとの連携機能が不足している．この課題を解決するために，オープンソースのソフトウェアである Kubernetes が存在している [4]．Kubernetes はコンテナ化されたアプリケーションを管理し，デプロイ，スケーリング，および運用するためのコンテナオーケストレーションツールの一例である [5]．さらに，Kubernetes の機能を簡素化し，低スペックのコンピュータでも実行可能にした K3s と呼ばれるオープンソースのソフトウェアが存在する [6, 7]．本稿ではこの K3s の環境を想定している．

コンテナを構築する際には，Kubernetes のリソースである ConfigMap を使用することができる．ConfigMap はアプリケーションの構築情報を保存し，それをコンテナ内のアプリケーションに提供するための Kubernetes リソースである[*2]．具体的には，本稿では Nginx の設定ファイルの

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[2] 東京工科大学院バイオ・情報メディア研究科
〒 192–0982 東京都八王子市片倉町 1404-1

[*1] https://atmarkit.itmedia.co.jp/ait/articles/1406/17/news013.html
[*2] https://kubernetes.io/ja/docs/concepts/configuration/configmap/

構成情報を ConfigMap に保存し，それを基に Nginx コンテナを生成する．

課題

クライアントから Web サーバへのアクセス対応可能な数を増やすには，リバースプロキシを導入することが方法のひとつとして挙げられる [1, 2]．図 1 はリバースプロキシを使用したネットワーク経路を図に示したものである．

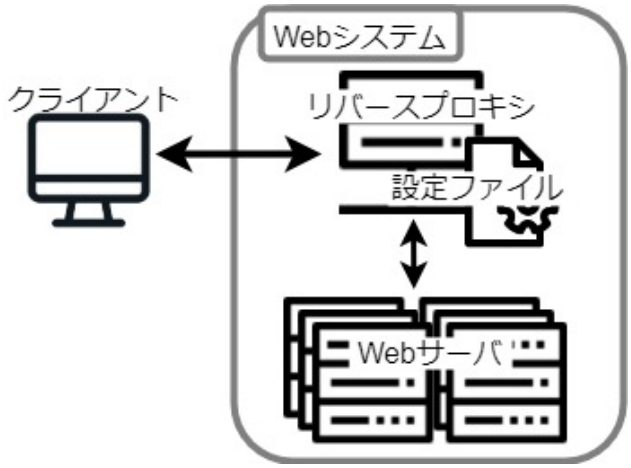

図 1　リバースプロキシの説明

リバースプロキシを導入する場合，Web システム内で Web サーバとクライアントの間に配置する．本稿では，Nginx ベースのリバースプロキシを新たに導入する際の作業時間が長時間となることを課題とし，図 2 に示す．本稿では Kubernetes 上にリバースプロキシを構築することを前提としている．

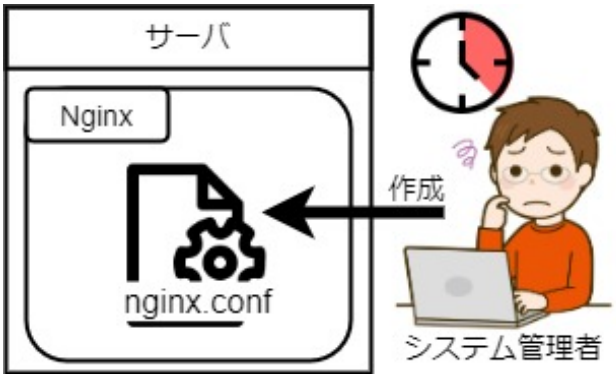

図 2　設定ファイル作成に時間を要す

Nginx をリバースプロキシの Web サーバとして稼働させる場合，設定ファイルである nginx.conf を作成する必要がある．nginx.conf の作成作業は設定項目の記述が主であり，Nginx 特有のドキュメントルールに従って正確に書く必要がある．実際の nginx.conf の記載例をソースコード 1 に記す．

設定ファイルは設定項目と設定値から成り立つディレクティブと，それを種類ごとに括るブロックで構成される．設定項目の例を以下に挙げる．

ソースコード 1　nginx.conf のサンプル

```
1  # ワーカープロセス数
2  worker_processes 1;
3
4  # イベントモジュールの設定
5  events {
6      worker_connections 1024;
7  }
8
9  # HTTP モジュールの設定
10 http {
11     # グローバル設定
12     include mime.types;
13     default_type application/octet-stream;
14
15     # ログの設定
16     log_format main '$remote_addr -
           $remote_user [$time_local] "$request" '
17                     '$status $body_bytes_sent "
                           $http_referer" '
18                     '"$http_user_agent" "
                           $http_x_forwarded_for"';
19     access_log /var/log/nginx/access.log main;
20     error_log /var/log/nginx/error.log;
21
22     # サーバー設定
23     server {
24         listen 80; # ポート番号
25
26         # リバースプロキシの設定
27         location / {
28             proxy_pass http://backend-server; #
                   バックエンドサーバーのアドレス
29             proxy_set_header Host $host;
30             proxy_set_header X-Real-IP
                   $remote_addr;
31             proxy_set_header X-Forwarded-For
                   $proxy_add_x_forwarded_for;
32             proxy_set_header X-Forwarded-Proto
                   $scheme;
33         }
34     }
35 }
```

- "worker_processes":
  サーバが使用するワーカープロセス（処理単位）の数を指定する．同時に処理できるリクエスト数を変更できる．
- "error_log":
  エラーログの出力先やフォーマットを指定する．
- "location":
  リクエスト URI に基づいてリクエストの処理方法を指定する．静的なコンテンツの提供やリバースプロキシの設定，リクエストの制御，複数の目的で使用する．
- "proxy_pass":
  リバースプロキシのプロキシ先のアドレスを指定する．

これら以外にも，リバースプロキシサーバと Web サーバの間でセキュアな通信を行うための”proxy_ssl”や，リクエストヘッダーの情報を Web サーバに転送するための”proxy_set_header”の，無数の設定項目が存在している．Nginx ベースのリバースプロキシを構築するユーザは，Nginx 独自のドキュメントルールに従いつつ，公式ドキュメント[*3]から必要とする設定項目から探し，設定ファイルを作成する必要がある．

各章の概要

本稿は以下の構成を持つ．第 1 章では背景とその課題について述べ，第 2 章では関連研究について述べる．第 3 章では，課題解決のための提案方式を説明し，第 4 章ではその実装と実験方法を述べる．第 5 章では基礎実験の内容とその評価について論じ，第 6 章では提案方式に関する議論を行う．最後に，第 7 章では論文のまとめと達成した成果について述べる．

2. 関連研究

教育行政システムの負荷分散を行い，最適なクラスター運用状態を手動で比較・適用する研究がある [8]．提案として，Nginx のサーバクラスター環境に基づいて最適なクラスター運用状態を比較し複数同時アクセスが来た際，より安定したパフォーマンスを発揮する設定を適用する． 本稿の提案方式で構築されるリバースプロキシの設定は全てにおいて最適とは言えず，更にパフォーマンス向上の余地がある．その点を改善する際に，この研究は活用することができる．

サーバの負荷情報をリアルタイムで収集し，Nginx の設定を動的に調整する研究がある [9]．本稿の提案ではリバースプロキシの構築時に Nginx の設定ファイルを作成して，それ以降に書き換えることは想定していない．そのためその後クラスター内の環境に変化があっても以前の設定を使用し続けることになる．リバースプロキシ構築後にプロキシ先が増えるといったクラスター内の環境の変化があっても自動でプロキシ設定が行われるような機能を追加する場合に，サーバ内のリアルタイム監視を行うこの研究は活動できる．

アプリケーションのパフォーマンスを最適化するための，自動チューニングツールを提案する研究がある [10]．この研究では実際に Nginx にチューニングを実行し，Nginx サーバのパフォーマンス向上に成功している．この研究の提案はチューニングを目的としているため，本稿の課題を解決することは出来ない．

[*3] http://nginx.org/en/docs/

3. 提案

提案の目的は，Nginx の設定ファイル作成とその記載情報の取得の作業時間を削減することで，リバースプロキシの構築時間全体を短縮することである．提案方式では，設定ファイルの作成と記載情報の取得作業に掛かる時間を削減するために，これらの工程と apply 作業の自動化を図る．提案では設定ファイルの記載情報を取得して記述する．具体的には設定ファイルの”proxy_pass”に記載する情報として，クラスター内に存在する Service を自動で取得する．その際，取得した情報を全てそのまま設定ファイルの書き込むと，プロキシの必要がない Service にも接続されてしまう．それを踏まえて提案方式の前提条件として，以下の 5 つを設定する．

- K3s は事前にインストールされている．
- クラスター内にはプロキシの対象となる HTTP サーバは事前に用意されている．
- クラスター内の Service のうち，HTTP サーバ以外にはプロキシしない．
- クラスター内の HTTP サーバには全てプロキシする．
- ポートが設定されていない Service は無視する．

提案方式

提案方式は提案の内容を自動的に行うソフトウェアを作成し，提案ソフトウェアの名前は NR-make とする．図 3 に提案のシーケンス図を示す．

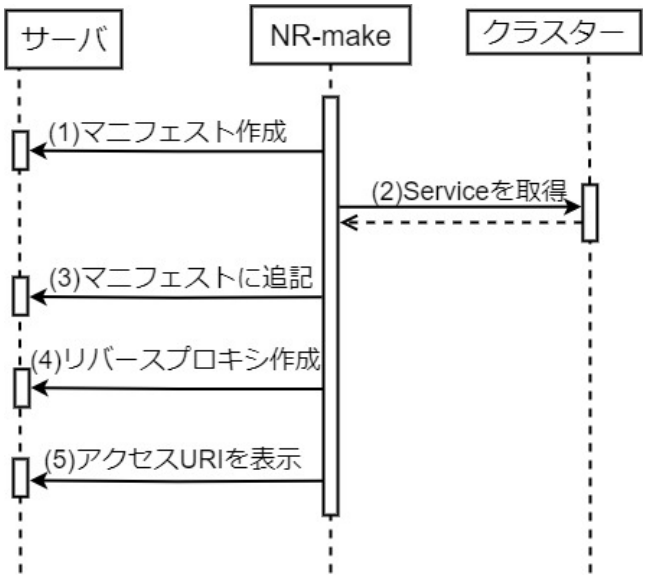

図 3 提案のシーケンス図

本来 Nginx ベースのリバースプロキシを Kubernetes クラスター上に構築する場合，ユーザ自身がプロキシ先のアドレスや Service 名を調べ，その情報を含めて Nginx の設定ファイルとなるマニフェストファイルに書き込む必要

がある．提案方式はこれらを自動的に行うソフトウェアを作成することである．ユーザがプロキシ先の情報を調べたり，マニフェストファイルの間違い修正に割く時間を削減する．図 4 は提案の概要である．

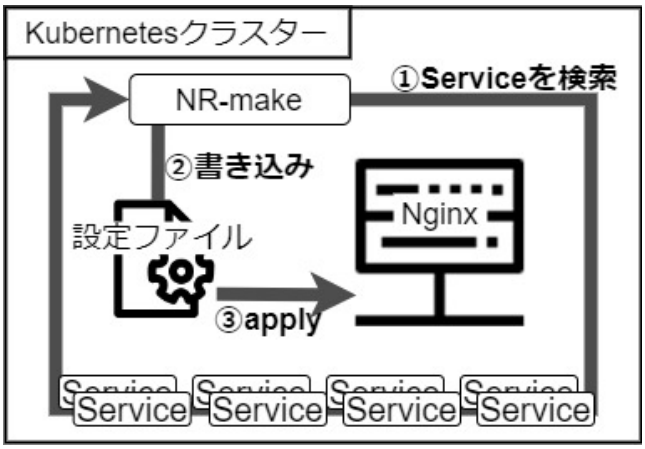

図 4 提案の概要

提案手法の手順と目的について詳しく述べる．提案手法の手順は以下の 5 つである．

(1) ConfigMap, Deployment, Service マニフェストを作成
(2) Kubernetes クラスター内の Service 情報を取得
(3) 取得した Service 情報を編集し ConfigMap マニフェストに追記
(4) 3 つのマニフェストを基にコンテナを作成
(5) アクセス URI を画面出力

(1)ConfigMap，Deployment，Service のマニフェストを作成では，リバースプロキシの設定を決めるファイルと，生成したコンテナ管理用のオブジェクトの設定を決めるファイルを作成する．この時点で作成される ConfigMap マニフェストは未完成の状態であり，以降の手順で追記される．

(2)Kubernetes クラスター内の Service 情報を取得では，次の手順で必要となるプロキシ先の Service の情報を取得する．この手順では NR-make を実行したクラスター内に存在する全ての Service 情報が一時保存される．この手順は図 4 の①である．ソースコード 2 は Service 名を取得するために使用するコマンドである．

ソースコード 2 Service 取得コマンドの例

```
1 services=$(kubectl get services --all-
      namespaces -o jsonpath='{.items[*].metadata
      .name}' | tr ' ' '\n')
```

(3) 取得した Service 情報を編集し ConfigMap マニフェストに追記では，リバースプロキシの処理に関係する部分の設定をマニフェストに書き込む．また取得した Service がプロキシの対象の HTTP サーバであるか判別の選定も行う．この手順で書き込まれるテキストは，Kubernetes クラスター内に存在する Service の個数や各情報の状況によって変化する．ここでの各情報は Service 名や Service のポートである．図 4 の②当たる部分である．ソースコード 3 はこの手順の処理を簡潔に表した例である．ソースコード 3 では，取得した Service の数だけループしその中で，プロキシ先として設定する必要があるか 5-7 行目で判別してから 10-12 行目で ConfigMap に追記する．

ソースコード 3 Service 書き込みコマンドの例

```
1  # 取得したService の数だけループ
2  for service in $services; do
3
4    # 現ループの
         Service がプロキシの対象の HTTP サーバか確認
5    if [ プロキシ先のステータスコード != "200" ]; then
6      continue
7    fi
8
9    # ConfigMap に追記
10   echo " location パス名 {
11         proxy_pass http://$service.default.svc.
               cluster.local:プロキシ先のポート/;
12       }" >> ~/nginx_proxy/nginx-configmap.yaml
13 done
```

(4)3 つのマニフェストを基にコンテナを作成では，実際にリバースプロキシが動作する状態にするためにコンテナを起動させ，図 4 の③のである．この時点でリバースプロキシとプロキシ先が接続され，リバースプロキシ構築自体は完了する．

(5) アクセス URI を画面出力では，リバースプロキシ経由で Web ページにアクセスする URI をターミナル上に表示する．リバースプロキシ構築後の動作確認をスムーズにユーザが行うために表示する．

ユースケース・シナリオ

本稿のユースケースは，自社の保有サーバ上で Web サイトを管理する食品製造会社である．図 5 はユースケース・シナリオを表したものである．

システム管理者は自社の Web サーバ管理業務の設計・構築・運用・保守を行う．自社 Web サーバ上には会社ホームページが運用されており，このホームページには会社概要や取り扱い商品の紹介，お問い合わせフォーム，採用情報が Web 公開されている．クライアントはインターネットを通してこのホームページにアクセスする．

現代ではソーシャルネットワーキングサービス，通称 SNS が普及してる．この SNS の拡散がきっかけで突然特定の商品やサービスの知名度が上がり，それに付随する Web ページへのアクセス数が増加することがある[*4]．この

[*4] https://twitter.com/AJIMAI3/status/1600350358005764096

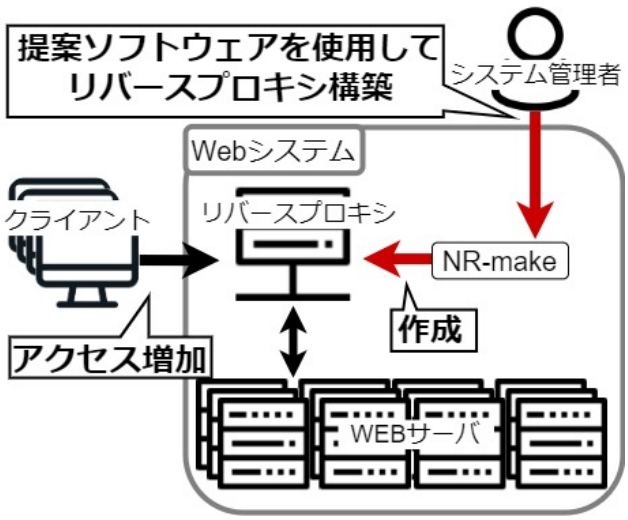

図 5 ユースケース・シナリオの概要図

際のアクセス数の増加に伴い，Web ページを管理する Web サーバの負荷が上昇する．システム管理者はこれに対処すべく，サーバ負荷耐性の向上を目的としたリバースプロキシ導入を，本稿のシナリオとする．

リバースプロキシを導入する過程で，それを行うシステム管理者が Nginx や Kubernetes の知識に乏しい場合，作業に長時間が必要とされる．本稿で作成したソフトウェアは，リバースプロキシ構築における設定ファイル作成と適用を自動で行う．そのため Nginx に関係する知識に乏しい場合でも，作業に長時間をかけずに，リバースプロキシ構築を行うことができる．

4. 実装

実装では提案方式を基に，リバースプロキシの自動構築を行うソフトウェアを新たに作成する．使用するプログラミング言語はシェルスクリプトを使用する．以下に具体的な処理を記す．

まず”echo”コマンドで設定が書かれたマニフェストを作成する．この際書き込まれる情報は Deployment と Service マニフェストの全てと ConfigMap の途中までであり，NR-make 実行毎に同じ情報である．”kubectl get”コマンドでクラスター内の Service の情報を取得する．この取得した情報は変数に保存され，複数 Service が存在した場合は改行文字で区切られる．この取得した情報をループで一つずつ取り出し，”echo”コマンドで ConfgMap マニフェストの”proxy_pass”の設定値として書き込む．この際 if 文で，指定した名前の Service，ポートが null の Service，”curl”コマンドで 200 番が返ってこなかった Service はループをスキップする．作成した 3 つのマニフェストファイルを基にコンテナを生成するために，マニフェストファイルを指定して”kubectl apply”コマンドを実行する．この際生成されるコンテナ内の Nginx のバージョンは基本的に最新のものが使用される．実装時点では，バージョン 1.25.3 である．”kubectl get”コマンドでリバースプロキシ自体の URI を取得する．更に NR-make の処理内で指定したパスと組み合わせ，”echo”コマンドでサービス名と共にターミナル上に表示する．この処理は接続したプロキシの数だけループする．

5. 評価実験

本稿で作成した NR-make を使用することにより，リバースプロキシ構築時間が短縮できているか実験を行い確かめる．図 6 は評価実験の概要を表した図である．手動での構築作業時間を比較対象とし，NR-make を使用してリバースプロキシを構築した作業時間を計測する．手動の構築は Nginx の知識がない情報系学生が行う．構築作業の開始は Kubernetes 環境とプロキシ先となる HTTP サーバが用意され，リバースプロキシサーバは存在しない状態とする．作業の終了はリバースプロキシ 1 つが作成され，HTTP サーバに正常にプロキシされていることが確認出来た時とする．

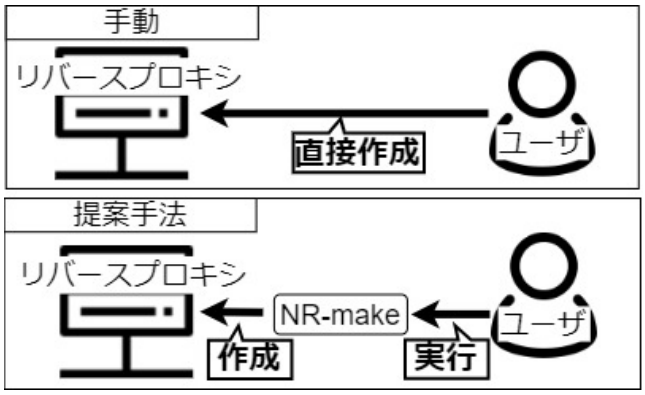

図 6 評価実験の概要図

実験環境

実験環境には 4 つの VM を用いる．今回の実験でのリバースプロキシ構築完了条件は，リバースプロキシのプロキシが機能するまでとし，その他の追加の機能は考えないものとする．時間の計測は作業の開始時間と終了時間を記録し，その差を作業時間とする．

NR-make を使用した構築時間の計測する際，NR-make を実行する Kubernetes クラスター内には事前にプロキシ先となる Service を 2 つを用意する．手動の構築時間を計測する際は，プロキシ先となる HTTP サーバを事前に別の VM 上に 2 つ用意する．更に実際に構築を行う VM は Kubernetes がインストールされたのみの状態とし，ユーザそれぞれに 1 つずつ用意した．手動での構築は，作業を行うユーザの能力で作業時間が大きく変動する．そのため 3 人がそれぞれ作業を行い，その作業時間を実験データと

する．構築を行うユーザには手順として，ConfigMap と Deployment と Service のマニフェストファイルを作成・記述することと apply することを伝えた．更に情報として，それぞれマニフェストファイルの入力例と，プロキシ先の URI を伝えた．この構築作業の過程でユーザが行き詰まった場合は，その都度リバースプロキシの構築経験のある者が助言を行った．以下に VM の構成要素を示す．

- VM 構成情報
  OS: Ubuntu-22.04
  vCPU: 2 コア
  RAM: 2GB
  HDD: 25GB

実験結果と分析

実験結果のグラフを図 7 に示す．この際，Nginx ベースのリバースプロキシを 3 人が手動で構築した作業時間を「ユーザ A」「ユーザ B」「ユーザ C」とした．更に比較対象として，提案ソフトウェアを使用して構築した際の作業時間を「提案使用」とした．この「提案使用」には，NR-make 実行時間，その他の NR-make のインストールや実行コマンド入力も含まれている．

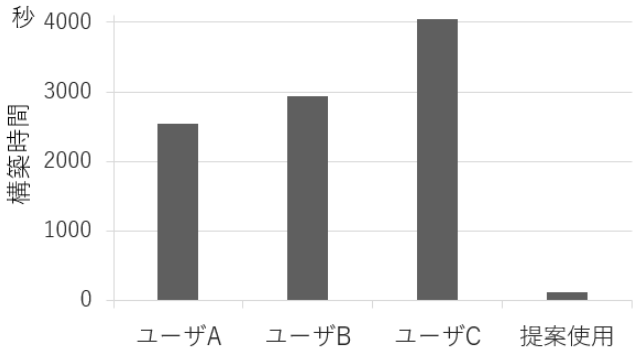

図 7 実験結果のグラフ

更に手動構築の詳しい値を表 1 に示す．ユーザがそれぞれのマニフェストの作成と apply に掛かった時間とその合計を「ConfigMap」「Deployment」「Service」「合計」とした．更に作業項目ごとの平均を小数第二位以下四捨五入で示した．

表 1 手動構築の詳細

ユーザ	ConfigMap[秒]	Deployment[秒]	Service[秒]	合計 [秒]
A	667	1,297	590	2,544
B	756	1,479	697	2,932
C	2,361	969	722	4,052
平均	1,258	1,248.3	669.7	3,176

実験結果として手動構築を行った場合，ユーザ A が 2,544 秒，ユーザ B が 2,932 秒，ユーザ C が 4,052 秒掛かり，これらを平均すると 3,176 秒となる．提案手法を活用した構築の作業時間は 122 秒となった．これらを比較すると 3,054 秒の差があり，96.16%の削減となった．ここまでの時間削減となった理由としては，手動の構築にはタイピングで設定を書かなければならないことや，構文ミスをした場合そのミスの位置発見に時間が掛かることがある．

手動構築の時間の内訳として Service の作成にかかる時間が他に比べ短い．これは Service のマニフェストファイルに記述する内容が他のマニフェストファイルに比べ少なく，誤字をしにくことがある．

6. 議論

提案方式では，リバースプロキシのプロキシ先となる Service を検索，その Service にプロキシされるように設定ファイルを記載，コンテナの作成を自動で行う．この処理でリバースプロキシの構築自体は完了し，プロキシされている状態になると想定している．しかし実際に正常にプロキシされているかの確認は，NR-make の処理内では行われない．つまりユーザ自身がリバースプロキシ経由で Web ページにアクセスして，正常にプロキシされているかの確認する必要がある．この工程も NR-make に組み込む場合，リバースプロキシ経由でアクセスしたステータスコードを判別して，プロキシできているかの確認を行う．さらにリバースプロキシ経由でアクセスして取得される HTML データと，プロキシ先自体に格納されている HTML データを比較し，想定通りのプロキシ先に接続されているかの処理を追加する方法が取れる．

また, 本提案方式の最終工程ではアクセスするための URI とプロキシ先のサービス名を表示するが，他にも Namespace 名の表示もすることでさらなるユーザの利便性向上が図れる．これは対応する Service 名を基に Namespace を取得し変数に保存・出力することで行える．

ConfigMap についての議論もある．本提案方式では ConfigMap に書き込む際，プロキシする Service ごとに URI を分けるためにパスを割り当てている．ソースコード 4 の 10 行目の location 以降のパスがこれにあたる．

この割り当てるパスは”Service 番号”(番号は 1 から連番で増加させていった自然数) としている．実際に運用する場合，この割り当て方法は視認性・管理性の面で問題となる．これを解決する手段として，パスをディレクトリ構造に変更することや，パス名を Service 名または HTML ページにある情報を基に決定することが挙げられる．

7. おわりに

Web サーバの負荷耐性の向上には，Nginx ベースのリバースプロキシ導入が選択肢のひとつである．本稿での課題は，その際の構築作業を Nginx 初学者が行うと作業時間が長時間になることである．課題に対しての提案として構

ソースコード 4　プロキシの location パスの割り当て

```
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: nginx-app-proxy
  name: nginx-conf
data:
  nginx-service.conf: |-
    server{
      location /パス {
        proxy_pass プロキシ先;
      }
    }
```

築の作業工程の，設定ファイルに記載する情報の取得と記述，コンテナの作成，アクセス URI の表示，これらの自動化をする．評価では提案手法を用いたリバースプロキシ構築と，用いない手動構築を実際に行い，作業時間を計測し比較した．結果として，提案手法を用いないで構築する作業時間から 3,054 秒の時間削減を行うことができた．

参考文献

[1] Ma, C. and Chi, Y.: Evaluation Test and Improvement of Load Balancing Algorithms of Nginx, *IEEE Access*, Vol. 10, pp. 14311–14324 (online), DOI: 10.1109/ACCESS.2022.3146422 (2022).

[2] Al-Saydali, J. and Al-Saydali, M.: Performance comparison between Apache and NGINX under slow rate DoS attacks (2021).

[3] Potdar, A., Narayan, D., Kengond, S. and Mulla, M.: Performance Evaluation of Docker Container and Virtual Machine, *Procedia Computer Science*, Vol. 171, pp. 1419–1428 (online), DOI: 10.1016/j.procs.2020.04.152 (2020).

[4] Shamim, S. I.: Mitigating Security Attacks in Kubernetes Manifests for Security Best Practices Violation, *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, New York, NY, USA, Association for Computing Machinery, p. 1689–1690 (online), DOI: 10.1145/3468264.3473495 (2021).

[5] Truyen, E., Kratzke, N., Van Landuyt, D., Lagaisse, B. and Joosen, W.: Managing Feature Compatibility in Kubernetes: Vendor Comparison and Analysis, *IEEE Access*, Vol. 8, pp. 228420–228439 (online), DOI: 10.1109/ACCESS.2020.3045768 (2020).

[6] Telenyk, S., Sopov, O., Zharikov, E. and Nowakowski, G.: A Comparison of Kubernetes and Kubernetes-Compatible Platforms, *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Vol. 1, pp. 313–317 (online), DOI: 10.1109/IDAACS53288.2021.9660392 (2021).

[7] Koziolek, H. and Eskandani, N.: Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, K3s, K0s, and Microshift, *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23, New York, NY, USA, Association for Computing Machinery, p. 17–29 (online), DOI: 10.1145/3578244.3583737 (2023).

[8] Han, J. and Ye, Y.: Research on the Application of Load Balancing in Educational Administration System., *Journal of Information Processing Systems*, Vol. 19, No. 5 (2023).

[9] E, Q., Wang, Y., Yuan, L. and Zhong, Y.: Research on Nginx Dynamic Load Balancing Algorithm, (online), DOI: 10.1109/icmtma50254.2020.00138 (2020).

[10] Wang, Q., Wang, R., Hu, Y., Shi, X., Liu, Z., Ma, T., Song, H. and Shi, H.: KeenTune: Automated Tuning Tool for Cloud Application Performance Testing and Optimization, *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, New York, NY, USA, Association for Computing Machinery, p. 1487–1490 (online), DOI: 10.1145/3597926.3604920 (2023).