

IT インフラ演習環境 hands-on-base 0.3.0 の 設計と実装

[著] 深町賢一

LPIC 2024/06/08 ウェビナー 新刊
2024 年 6 月 8 日 v0.3.0

■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、TM、[®]、[©]などのマークは省略しています。

まえがき / はじめに

本書は何か？

本書は、docker ベースの IT インフラ演習環境 hands-on-base リポジトリの解説書です。このリポジトリは、元々、2024-06-08 に開催した LPI のウェビナー「ブラウザのフタを開けて HTTP 体験しよう」(<https://lpic-2024q2.demo.fml.org/>) のために作成した環境になります*1。

想定している技術レベル

本ウェビナーを講義する側が、この環境を準備するはずなので、次のような技術レベルが想定されています。講義内容については、リファレンスにあるスライドと動画を参照してください

- HTTP プロトコルを初心者に解説できる
- Unix サーバ管理の基本コマンド操作が分かる
- DNS の設定が出来る
- docker の基本操作が分かる

必要な環境

docker を動かすためのサーバが必要です。本文では、ウェビナーのために VPS を短期間契約する想定となっていますが、自宅サーバでも十分運用できます。問題は DNS まわりです。本文は固定のグローバル IP アドレスを持っている想定で書かれていますが、必須ではありません。工夫すればダイナミック IP でも運用できるはずです。ただ、DNS まわりは環境依存が大きいため、詳細な説明は省略しました

*1 最初の企画案では、(a) 本演習の指導の仕方と (b) その演習環境の作り方の 2 部構成を予定しており、その (b) パート「環境構築」が本書となります ((a) も別途、製作中です)

リファレンス

- <https://github.com/sysbuild-training/hands-on-base>
 - 本書で解説する環境 hands-on-base の github リポジトリ
- <https://youtu.be/y84Asag901o>
 - ウエビナーのアーカイブ動画
- <https://speakerdeck.com/fmlorg/burauzanohutawokai-ketehttpsi-yan-siyou-20240608v1-dot-0-0>
 - ウエビナーのスライド
- <https://lpic-2024q2.demo.fml.org/>
 - ウエビナーで使うスクリプトやワークシートを配布するサイト
 - アーカイブ動画で自習する人のために、そのまま残してあります
- <https://www.lpi.org/ja/japan>
 - Linux Professional Institute 日本支部 LPI
- <https://www.youtube.com/@linuxprofessionalinstitute906>
 - Youtube の LPI チャンネル

目次

まえがき / はじめに	i
本書は何か？	i
想定している技術レベル	i
必要な環境	i
リファレンス	ii
第1章 仕様まとめ	1
1.1 仕様	2
第2章 全体の構成	3
2.1 リポジトリの構成	4
2.2 起動シークエンスの概要	4
2.2.1 起動時に変更可能なグローバルパラメータ	5
2.2.2 起動の様子	5
2.3 DNS	6
第3章 コンテナ群の構成	7
3.1 リポジトリの概要	8
3.2 nginx-ingress (HTTP PROXY)	9
3.3 ssh-gw (SSH PROXY)	10
3.3.1 裏側の理屈	11
3.4 debian-pc コンテナ	12
3.4.1 Dockerfile	12
3.4.2 files/entrypoint.sh	12
第4章 初期化と運用	15
4.1 受講者数を指定して初期化する	15
4.2 make による docker の起動	15
4.2.1 普通に起動する場合	15
4.2.2 デバッグモード	16
4.3 make による docker の停止	16
4.4 実行状況の監視	16

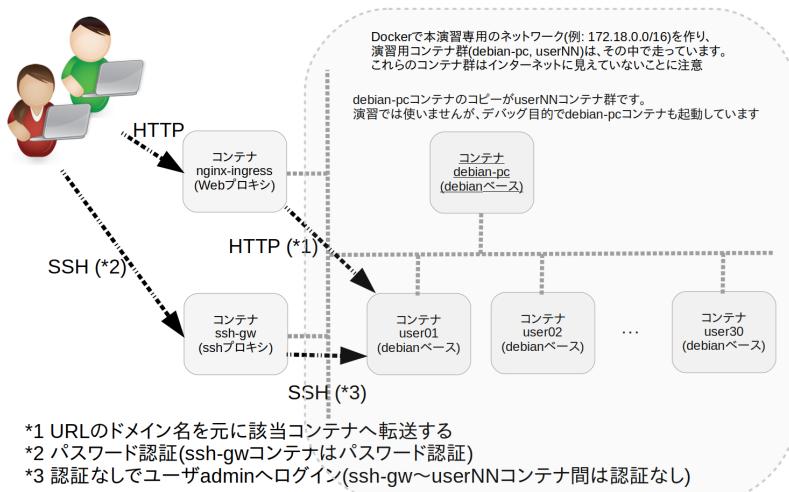
第 5 章 演習環境の負荷評価と調達	17
5.1 演習環境の負荷評価	17
5.2 演習環境の調達	18
あとがき / おわりに	19
この形に落ちつくまでの軌跡	19

第 1 章

仕様まとめ

次のような大方針で仕様を考えました

- できるだけ授業テキスト群と解説や演習内容を揃えたい
- 2時間程度の利用を想定。すぐに捨てるの細かいことは考えなくてよい
- いろいろ決め打ちでOK。かゆいところまで手が届く必要なし



▲図 1.1: ネットワーク図

1.1

仕様

受講者 01 番のユーザは、user01@www.user01.demo.fml.org に ssh してコンテナにログインします。このコンテナ上でサーバ構築の演習を行い、サーバの動作確認は URL <http://www.user01.demo.fml.org/> で行ってもらいます。

もう少し細かい仕様は次のとおりです

- 受講者ひとりひとりにコンテナ環境と URL を一つずつ割りあてます
- URL(www.user01.demo.fml.org) はユーザ名を決め打ちで用意します*¹
- コンテナ環境で WWW サーバを起動し、動作確認は URL で行います
 - 例: <http://www.user01.demo.fml.org/>
- コンテナにはパスワード認証*²で SSH ログインできます
 - ユーザが SSH ログインする先は user01@www.user01.demo.fml.org です
 - * ユーザ名 user01
 - * ドメイン名 www.user01.demo.fml.org
 - * 正確には「SSH の踏み台ごしにコンテナへログインする構成」で、SSH の踏み台でパスワード認証しています。一方「SSH の踏み台～コンテナ」間の SSH はパスワードなしです
- docker compose による環境構築を行います。使うコンテナは次の 3 種類です
 - nginx-ingress
 - * HTTP をリバースプロキシする nginx コンテナ
 - * URL を元に各ユーザのコンテナへプロキシします
 - ssh-gw
 - * 踏み台コンテナで、sshd だけが走っています
 - * ユーザ名を元に各ユーザのコンテナへ SSH をプロキシします
 - debian-pc
 - * これをテンプレートとして、user01 用の user01 コンテナ～user30 用の user30 コンテナとして必要な数だけコピーして起動させます
 - * デバッグ用に別途 debian-pc コンテナも起動しています

*¹ 簡単化のため「グローバル IP アドレス」と「VPS の契約」を想定しますが必須ではありません

*² 初心者に公開鍵暗号と SSH の使い方を説明するには、かなりの時間を要するため、本演習では省略しました。ただ、Unix サーバ運用上は必須の知識なので、本演習後、別途、訓練してください

第 2 章

全体の構成

本章では、コンテナ群以外の要点を解説します。詳細は次の github のリポジトリを参照してください。

<https://github.com/sysbuild-training/hands-on-base>

以下では、この URL をリポジトリと呼びます。なお、コンテナ群については次章で解説します。

▼ リポジトリ構造 (1): コンテナ群以外のもの、ドキュメント類は省略

```
hands-on-base
├── LICENSE
├── Makefile
├── docker-compose.yml
├── examples
│   ├── docker-compose.devel.yml
│   ├── docker-compose.production.yml
│   └── user-containers.devel.yml
└── utils
    ├── config.sh
    ├── nginx-config-setup.sh
    ├── production-yml-setup.sh
    ├── run-test-docker-pc.sh
    ├── run-test-ssh-login-scripts.sh
    ├── sshd-config-setup.sh
    └── user-pwgen.sh
```

2.1 リポジトリの構成

リポジトリ直下にあるファイルは、LICENSE、Makefile そして docker-compose.yml です。

- LICENSE は文字どおりライセンスについて書いてあります
- Unix の伝統芸 make コマンドのルールが書いてあるのが Makefile です。
make は(今風の用語)タスクランナーの御先祖様と言えばよいでしょうか
- docker-compose.yml は docker compose コマンドの基本部分です。実際には、
(a) このファイルと、(b)examples/以下の yml をテンプレートとして work 以下に動的に生成される yml 群を引数に docker compose が起動されます。その詳細は utils/以下のスクリプトを読んでください

これら以外に examples と utils ディレクトリがあります。

- ./examples/
 - docker compose の設定(yml) のテンプレートがあります
 - このファイル群から work/以下に yml の設定ファイルが生成されます
- ./utils/
 - ./work/以下の設定ファイル群を生成するシェルスクリプト群です

2.2 起動シークエンスの概要

コンテナのディレクトリ(nginx-ingress、ssh-gw、debian-pc、testweb)について次章で解説します。また、ドキュメントの README.md と docs ディレクトリも省略します。本節では、これら以外の部分について見ていきます。

- DNS まわりには、リポジトリとは独立した設定作業が必要です(後述)
- docker compose でコンテナ群を起動しますが、docker compose の制御は単に docker compose up ではなく make コマンドで行ってください。これは ./work/以下のファイル群を引数に取る必要があるためです。make のルールの詳細はリポジトリ直下にある Makefile を読んでください
- ./work/以下のファイル群は、Makefile 中に定義されているルールにしたがい、./utils/以下のシェルスクリプトで生成されます
- docker compose は ./work/以下に作成される設定ファイル群を使い起動してきます。このファイル群は、(1) ./examples/ にある yml テンプレート (2) コンテナ自体のデフォルト値の二つを元に ./utils のスクリプトが生成したものです

- 各演習環境固有の (Unix) 設定ファイル群が、リポジトリ直下の./work/以下に動的に生成されます。このファイルの中には、コンテナ名/files/の設定ファイルとの合成になっているものがあります。どう合成しているのか？の詳細は./utils/以下のシェルスクリプトを御覧ください
- docker compose を、docker-compose.yml と work/以下に生成される.yml 設定ファイルを引数として起動させます。./work/以下の.yml は./examples/にある.yml をテンプレートとして作成されたものです

♣ 2.2.1 起動時に変更可能なグローバルパラメータ

なんらかのファイル編集なしに変更可能なパラメータは**ユーザ数** (Makefile の先頭にある変数 **MAXUSERS**) だけです。

▼ リスト 2.1: Makefile の先頭部分

```
# global parameters
MAXUSERS = 36
```

環境変数もしくは make の引数で変更できますが、忘れずに実行するために、Makefile の先頭の **MAXUSERS** の値を編集しておくほうがよいと思います。

▼ リスト 2.2: コマンドでユーザ数を指定して実行する例

例: 環境変数で指定する
\$ env MAXUSERS=50 make

例: makeの引数で変数を上書きする
\$ make MAXUSERS=50

♣ 2.2.2 起動の様子

▼ リスト 2.3: make up 実行時に、実際に実行される内容 (デフォルトの開発モード)

```
env MAXUSERS=36 sh utils/nginx-config-setup.sh
env MAXUSERS=36 sh utils/sshd-config-setup.sh
env MAXUSERS=36 sh utils/user-pwgen.sh
env MAXUSERS=36 sh utils/production-yml-setup.sh
docker compose -f docker-compose.yml \
    -f work/docker-compose.devel.yml \
    -f work/user-containers.devel.yml build
docker compose -f docker-compose.yml \
    -f work/docker-compose.devel.yml \
    -f work/user-containers.devel.yml up -d
```

▼ リスト 2.4: make up 実行時に、実際に実行される内容 (本番モード)

```
env MAXUSERS=36 sh utils/nginx-config-setup.sh
env MAXUSERS=36 sh utils/sshd-config-setup.sh
env MAXUSERS=36 sh utils/user-pwgen.sh
```

```
env MAXUSERS=36 sh utils/production-yml-setup.sh
docker compose -f docker-compose.yml \
  -f work/docker-compose.production.yml \
  -f work/user-containers.production.yml build \
  docker compose -f docker-compose.yml \
  -f work/docker-compose.production.yml \
  -f work/user-containers.production.yml up -d
```

2.3 DNS

この演習には自由に設定できる自ドメインがあるという想定がありますが、この部分はハンズオン開催者の環境依存なので、DNS サーバの設定詳細は省略します。

以下では、ドメイン demo.fml.org の例を紹介します。

演習環境を走らせるサーバには、固定 IP と、そこそこのパワーが必要です。とはいえ、月額 1000 円くらいの VPS で大丈夫だと思います（サーバのスペックの見積りについては、第 5 章「演習環境の負荷評価と調達」を参照）。

設定は簡単です。自ドメインに適当なサブドメイン（ここでは demo.fml.org）を作成し、想定するユーザ数の分、VPS の IP アドレス（ここでは 10.20.30.40 とします）でドメインの CNAME なり A レコードなりを設定するだけです（リスト 2.5）。

▼リスト 2.5: ゾーンファイルの例。もちろん IP アドレス (10.20.30.40) は各自の IP に差し替えてください

～省略～

```
$ORIGIN demo.fml.org.

www.user01      IN      A      10.20.30.40
www.user02      IN      A      10.20.30.40
www.user03      IN      A      10.20.30.40
```

～省略～

```
www.user30      IN      A      10.20.30.40
```

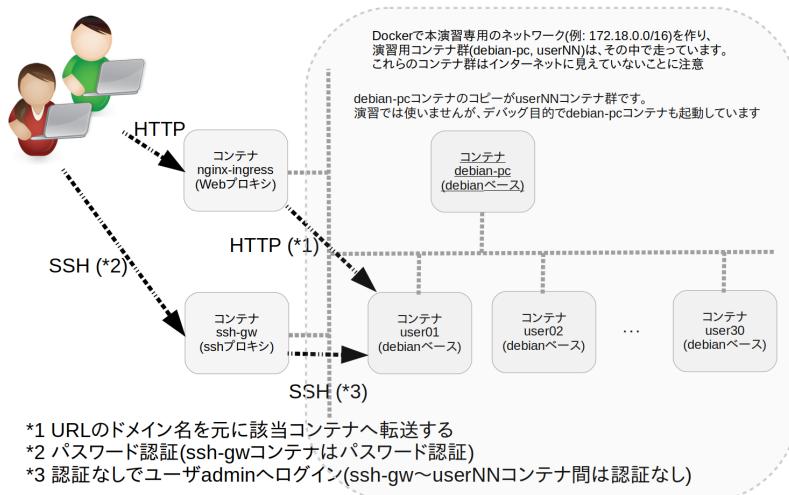
第3章

コンテナ群の構成

本章では、コンテナ群の要点を解説します。

▼ リポジトリ構造 (2): コンテナ群 (演習で直接は使わない testweb は省略)

```
hands-on-base
├── debian-pc
│   ├── Dockerfile
│   └── files
│       └── entrypoint.sh
├── nginx-ingress
│   ├── Dockerfile
│   └── files
│       ├── conf.d
│       │   ├── debian-pc.conf
│       │   └── testweb.conf
│       └── nginx.conf
└── ssh-gw
    ├── Dockerfile
    ├── dist
    │   ├── 00-src-info
    │   ├── moduli
    │   ├── ssh_config
    │   └── sshd_config
    └── files
        ├── entrypoint.sh
        ├── list.users
        └── sshd.conf.patch
```



▲図 3.1: ネットワーク図 (再掲)

3.1 リポジトリの概要

- リポジトリには次の4種類のコンテナがあり、同名のディレクトリ以下に docker コンテナの作成ファイルがあります
 - nginx-ingress
 - ssh-gw
 - debian-pc
 - testweb コンテナはテスト用です。演習では明示的には使いませんが一応 うごかしてあります

まずは、フロントエンドにあたる nginx-ingress (HTTP PROXY) と ssh-gw (SSH PROXY) コンテナをし、そのあと演習環境の実体である debian-pc コンテナについて解説します。

なお、説明で使う IP アドレスは、DNS 節の例で使った 10.20.30.40 を使うことにします。

3.2 nginx-ingress (HTTP PROXY)

ユーザ **user01** のホームページの URL は <http://www.user01.demo.fml.org/> になります。この URL は、各自の演習環境 (debian-pc コンテナのコピー) で動かす WWW サーバ (実体は www.py) の動作確認用です。

ユーザからの (つまりインターネット側からの) HTTP は、10.20.30.40 の 80/tcp で動いている **nginx-ingress** コンテナ (リバースプロキシ) が、いったん受け止め、再度、裏側のコンテナの 80/tcp へ HTTP します。リレーする先は、**user01** という名のコンテナで、コンテナの元イメージは **debian-pc** (後述) です。

裏側ではユーザ数だけユーザ名の **debian-pc** コンテナのコピーが起動しています。ユーザ ID が user01 ならコンテナ名も user01 です。受講者数が 30 人なら user01 から user30 の合計 30 個のコンテナがが起動しています。

docker compose up するとプロジェクト独自の bridge ネットワークが構築され、コンテナ群は、その bridge に接続される仕組みです。独自 bridge ネットワークでは 127.0.0.11 で名前解決できる (docker が用意した resolver が動いている) ので、**user01** などのコンテナ名を、そのままリバースプロキシする先のサーバ名として使えます。

なお、インターネット側へ EXPOSE する必要がある 80/tcp は **nginx-ingress** コンテナ (リバースプロキシ) だけです。user01 から user30 コンテナは bridge ネットワークでだけ 80/tcp を listen します。つまり、インターネット側からの HTTP は、一度 nginx が受け、nginx が裏側ネットワーク (bridge ネットワーク) の中で、再度 HTTP を張り、コンテナと中継しているわけです。

設定がどうなっているか？ ですが、まず、**nginx-ingress/files/conf.d** 以下に各コンテナごとのリバースプロキシの雛形の設定があります。演習環境のコンテナへプロキシする設定は、**nginx-ingress/files/conf.d/debian-pc.conf** をテンプレートとして、**./work/etc/nginx/conf.d** 以下に各コンテナの設定ファイル (**user01.conf** ~ **user30.conf**) を生成します。**nginx-ingress** コンテナでは、これらを **/etc/nginx/conf.d** 以下にマウント (volume mount) しているので、起動時に、これらの設定ファイル群を読んでリバースプロキシを実行しています。

つまり、初期化時に、**nginx-ingress/files/conf.d/debian-pc.conf** を元に次のような設定ファイル群が作成されているわけです (リスト 3.1)。

▼ リスト 3.1:

```
./work/etc/nginx/conf.d/user01.conf
./work/etc/nginx/conf.d/user02.conf
```

```
～ 省略 ～
```

```
./work/etc/nginx/conf.d/user30.conf
```

3.3 ssh-gw (SSH PROXY)

SSH でログインするには

ユーザ user01 には、あらかじめユーザ名とパスワードを通知し、次のように ssh コマンドを実行するように指示してください。

▼ リスト 3.2: ユーザが各自の PC のターミナルで実行するコマンド

```
ssh user01@www.user01.demo.fml.org
```

ログインに成功すると、プロンプトは **admin@user01\$** のように表示されます。

▼ リスト 3.3: 実行例: user01 コンテナにログインしたところ

```
$ ssh user01@www.user01.demo.fml.org
～ 省略 ～
Are you sure you want to continue connecting (yes/no)? yes
user01@www.user01.demo.fml.org's password:
Linux f86830571f83 4.19.0-26-amd64 #1 SMP Debian 4.19.304-1 (2024-01-09) x86_64
```

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

```
Last login: Mon Mar 11 10:50:50 2024 from 172.23.0.4
admin@user01$
```

プロキシの理屈は nginx の場合と同様です。インターネット側からの SSH を **ssh-gw** コンテナが 22/tcp で待ち受けています。いちど、**このコンテナにパスワード認証でログイン**してもらい、SSH コネクションを受け止めた sshd から、再度（bridge ネットワークにある）裏側のコンテナへ ssh を張る動作です。前述のとおりコンテナ名で名前解決できるため、SSH 先のサーバ名はコンテナ名 user01 になります（リスト 3.4、詳細はソースコードを参照）。

このプロキシ動作分の設定は、**ssh-gw** コンテナの設定ファイル /etc/ssh/sshd_config に埋め込んであります。

そういうわけで、**ssh-gw** コンテナには、(1) user01 ～ user30 のユーザとパスワードの作成 (2) /etc/ssh/sshd_config に全ユーザ分のプロキシ、という二種類の設

定が必要です。初期化の際に、これらの準備をしています。

ssh-gw コンテナの sshd から、再度、裏側のコンテナへ ssh しますが、このときは、どのコンテナ (user01 ~ user30) へもユーザは **admin** で、**パスワードは無し**です。

▼ リスト 3.4: sshd_config に追加するプロキシ設定例。これは debian-pc コンテナ行きの分。本来 ForceCommand 行は 1 行だが、ここでは折り返していることに注意

```
Match User user01
  ForceCommand /usr/bin/ssh \
    -o StrictHostKeyChecking=no \
    -o PasswordAuthentication=yes \
    -l admin user01
```

♣ 3.3.1 裏側の理屈

リポジトリの./work/docker-compose.production.yml を見てください。リポジトリ上の./work/etc/ssh/をコンテナの/etc/ssh/に volume mount しています。よってコンテナの/etc/ssh/sshd_config ファイルの実体は、リポジトリの./work/etc/ssh/sshd_config です。注意点として、./work/etc/ssh/sshd_config は、起動時に毎回./utils/sshd-config-setup.sh が再生成しています。

コンテナ (user01 ~ user30) は、すべて **debian-pc** コンテナイメージと同じであることに注意してください。コンテナのカスタマイズは、**./work/docker-compose.production.yml** に書かれています。初期化の際に、このファイルに設定が追加されます。

SSH プロキシの品定め

プロキシの実装は、少し考えあぐねていました。

プロキシの定番 **squid** で SSH のプロキシも出来るそうですが、それ以外にも **sshpiper**^{*1}、**sshmuxd**^{*2}、**sshr**^{*3} などが見つかります。ただし、今回は運用を考えるわけではないので、プラグイン拡張出来る出来ないとか関係ないですし、設定ファイルのフォーマットを覚えるのが面倒とか、ドキュメントが口吻に無いとか、学習コストが大きいし、無駄に高機能のように思いました。

たしか **sshmuxd** のドキュメントに、「sshd_config でもいいけど、その設定を考えるのが煩わしい」旨が書いてあったと思いますが、2 時間一本勝負のハン

*1 <https://github.com/tg123/sshpiper/>

*2 <https://github.com/kennylevinsen/sshmuxd>

*3 鶴田博文、松本亮介、"ユーザに変更を要求せずにシステム変化に追従可能な SSH プロキシサーバ sshr の開発"、トランザクションデジタルプラクティス Vol.2 No.4 (2021), <https://www.ipsj.or.jp/dp/contents/publication/48/TR0204-06.html> (accessed 2024-03-10)

ズオンでは、ユーザごとに設定が異なる可能性など考慮不要です。今回は、テンプレートを一つ作りさえすれば、そこから機械的に生成するだけなので、そんなに手間ではありません。

そういうわけでも、少し PROXY まわりも試してみたのですが、すなおに **openssh** の設定ファイルを書くことにしたというわけです。

.....

3.4 debian-pc コンテナ

▼ debian-pc コンテナのファイル群

```
└── debian-pc
    ├── Dockerfile
    └── files
        └── entrypoint.sh
```

3.4.1 Dockerfile

Dockerfile はシンプルです。

debian の小さなコンテナモデル (debian:12-slim) をベースに、演習で必要なパッケージのみを apt コマンドで追加しています。

そして、**files/entrypoint.sh** ファイルをコンテナ内の**/entrypoint.sh** にコピーし、コンテナは、起動シークエンスの最後に、この**/entrypoint.sh** を実行します。

さまざまな仕込みは**/entrypoint.sh** の中に書かれています。

3.4.2 files/entrypoint.sh

コンテナの起動時に、コンテナ環境のカスタマイズを行う約 100 行のシェルスクリプトです。大きく分けて処理が 5 ステップあります。ステップ 3 の「コマンド履歴」部分は説明が難しいので付録として github のほうにでも書いておきます

(1) admin アカウントの作成

最初の 2 行はユーザの作成です。useradd 行でユーザ admin を作成します。次の sed の行では、作成したユーザのパスワードを削除しています。

後半の 2 行は sudo の設定です。ログインしたら、パスワードなしで sudo コマンドが使えるように設定を仕込みます。admin グループのユーザであればパスワードなしで実行出来ます。ただ、admin グループには admin ユーザしかいないので、実質的にはユーザ admin だけの設定です (注: 最近の debian では、ユーザ X を作成す

る際に、グループ X も作成し、そのグループ X がデフォルトのグループになっています)

```
#  
# (1) create a user "admin" without the password :-)  
# ~コメント群は省略~  
#  
useradd -m -u 1000 -s /bin/bash admin  
sed -i '/admin/s/!//' /etc/shadow  
  
test -d /etc/sudoers.d || mkdir -p /etc/sudoers.d  
echo '%admin    ALL=(ALL:ALL) NOPASSWD: ALL' >> /etc/sudoers.d/admin
```

(2) ホスト名をつける

環境変数 HOSTNAME を.yml で設定しているため、この環境変数をつうじて各コンテナにホスト名を割り振っています。~/bashrc(/home/admin/.bashrc) に設定を仕込むことでシェルのプロンプトのホスト名を設定します。シェルの設定ファイルに仕込みつつ、利便性のため、環境変数 XHOSTNAME にも HOSTNAME の値をいれ export しておいています (HOSTNAME は、ありがちな変数名なので、どこかで上書きされる可能性を考え、別途 XHOSTNAME も使っているというわけです)

それ以外にも、白黒のプロンプトにする (color_prompt=no) 設定もしています。

```
#  
# (2) hostname, bash prompt  
# ~コメント群は省略~  
#  
if [ -n "$HOSTNAME" ];then  
    echo "color_prompt=no" >> /home/admin/.bashrc  
    echo "PS1='admin@${HOSTNAME}\$ '" >> /home/admin/.bashrc  
    echo "export XHOSTNAME=${HOSTNAME}" >> /home/admin/.bashrc  
    export HOSTNAME=${HOSTNAME}  
    export XHOSTNAME=${HOSTNAME}  
fi
```

(3) コマンド履歴をとる

```
#  
# (3) send the bash history to the docker daemon  
# ~コメント群は省略~  
#
```

ユーザが入力したコマンド履歴を二重に取っています。この設定を仕込んでいるので、docker を動かすホスト (いわゆる母艦側 (で通じますか?)) の syslog を見張っていると、リアルタイムに全ユーザの操作履歴を見ることが出来ます

例: dockerを動かすホスト側で全ユーザの演習の様子を監視する

```
$ sudo tail -F /var/log/syslog
~省略~
2024-06-06T12:38:28.748843+00:00 ip-172-26-14-159 debian-pc/hands-on-base-debian-pc-1/79dce64lee19[711]: debian-pc      admin 2024-06-06T12:38:28 171767750
>8 --- df [0]
~省略~
```

(4) 環境変数で指定された URL のスクリプトを実行する

環境変数 RC_LOCAL_SCRIPT_URL を .yml で指定していれば、そのファイルをダウンロードして、シェルスクリプトとして実行します。ちょっとしたコンテナのカスタマイズであれば、この方法でカスタマイズすることができます。perl の行は、30 個のコンテナが同時にダウンロードしないよう、少しランダムにスリープするための呪文です (えらく適当な乱数生成ですけど)。

```
#
# (4)
#
url="${RC_LOCAL_SCRIPT_URL}"
if [ -n "$url" ];then
  printf "# run the startup script defined by URL: %s\n" "$url"
  printf "# script starts (%s)\n" "`date`"
  /usr/bin/perl -e 'rand(time|$); sleep(rand($$)/60000); sleep(rand(3));'
  curl -o /rc.local "$url"
  /bin/sh /rc.local
  printf "# script ends (%s)\n" "`date`"
fi
```

(5) sshd の起動

最後に、「パスワードなしでも OK」と設定して、sshd (ssh daemon) を起動します。-D オプションをついているので、この最後の行の sshd を実行しつづける形になります (より専門的な解説: -D オプションにより sshd は (tty を detach せず) ターミナルを握りしめたままになります。そして exec sshd するので、entrypoint.sh プロセスは sshd プロセスに成り代わります)

```
#
# (5) sshd
# ~コメント群は省略~
#
mkdir -p /run/sshd

sed -i /PermitEmptyPasswords/d      /etc/ssh/sshd_config
printf "PermitEmptyPasswords yes" >> /etc/ssh/sshd_config

exec /usr/sbin/sshd -D
```

第4章

初期化と運用

4.1 受講者数を指定して初期化する

受講者数をみつもり、Makefile の MAXUSERS の数字を編集してください。その後、make コマンドで docker を起動してください（後述）。自動的に初期化プロセスが走り、./work/以下に設定ファイル群を構築します

4.2 make による docker の起動

Makefile には STAGE という変数があります。STAGE に設定可能な値は devel(開発) と production(本番) です。STAGE を指定しない場合は、開発環境（いわゆるステージング環境、STAGE=devel）となり、起動するポート番号も起動するコンテナの数も異なります。

開発モードが不要であれば、Makefile の上の方にある STAGE という行を production と書き換えてしまうとよいでしょう

そのうえで、以下のようにすれば、適切な設定ファイルを引数に docker compose up が実行されます

♣ 4.2.1 普通に起動する場合

普通に起動する場合は、単に make up を実行してください。docker はターミナルから切り離されます。docker の実行状況は syslog を見てもらうことになります

```
$ make up
```

♣ 4.2.2 デバッグモード

デバッグモードで起動するなら `make debug` を実行してください。これはターミナルにログが出続ける状態です。

```
$ make debug
```

開発用に用意したものです。これを使うことは、あまり無いとおもいます。

動作テスト確認用なので、本番と異なるポート番号だったり、コンテナが1ユーザ分しか起動しないなどの相違点があります。相違点は、ファイル名に`.devel.yml`ファイルと`.production.devel`の差を見てください。

4.3 make による docker の停止

```
$ make down
```

4.4 実行状況の監視

ユーザが演習している様子をホスト側でリアルタイムに観察できるようになっています。演習上、これをどう活かすか？は難しいところですが、質問された際に、ユーザの環境を見せてもらわなくとも、ログから、「何を実行して、何に困っているか？」を推定できるので便利だと思います（少なくとも、うちのTAチームは、そういう考えているようです）。

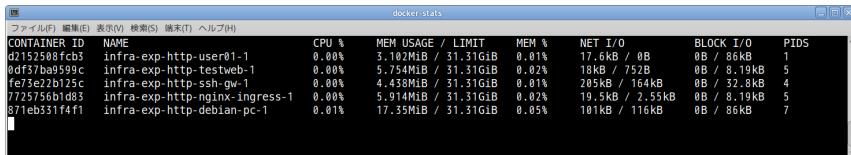
例: dockerを動かすホスト側で全ユーザの演習の様子を監視する

```
$ sudo tail -F /var/log/syslog
~省略~
2024-06-06T12:38:28.748843+00:00 ip-172-26-14-159 debian-pc/hands-on-base-debi>
>an-pc-1/79dce641ee19[711]:  debian-pc      admin 2024-06-06T12:38:28 171767750>
>8 --- df [0]
~省略~
```

第 5 章

演習環境の負荷評価と調達

5.1 演習環境の負荷評価



CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
d2152508fc03	infra-exp-user01-1	0.00%	3.182MB / 31.316GB	0.01%	17.5KB / 0B	0B / 86KB	1
0df31ba84599	infra-exp-http-testweb-1	0.00%	5.754MB / 31.316GB	0.02%	18KB / 752B	0B / 8.19KB	5
4e73e21b125c	infra-exp-http-ssh-gw-1	0.00%	4.438MB / 31.316GB	0.01%	205KB / 164KB	0B / 32.3KB	4
7725756b1d83	infra-exp-nginx-ingress-1	0.00%	5.914MB / 31.316GB	0.02%	19.5KB / 2.55KB	0B / 8.19KB	5
871eb531f44f	infra-exp-http-debian-pc-1	0.01%	17.359MB / 31.316GB	0.05%	181KB / 116KB	0B / 86KB	7

▲図 5.1: docker stats の様子

図 5.1 は開発環境で docker stats を実行している様子です。debian-pc コンテナを実行して、その上で www.py をダウンロード後、www.py を起動し、すでに何回かアクセスしてみたところです。だいたい何度やっても、この 17MB 前後の値になります。nano による編集や telnet を実行しても変わりません。

共通部分の nginx-ingress(リバースプロキシ) や ssh-gw(SSH の踏み台) などのコンテナ群は、これらのメモリ使用量を合計しても 20MB でおつりが来るでしょう。

いちおう余裕を見て約 2 倍、ひとつのコンテナあたり 30MB のメモリが必要と見積もることにします。受講者数が 30 人とすると、演習環境で必要とするメモリは 1GB 程度です。docker を動かすホスト側の分を入れても 2GB でなんとかなりそうです。

5.2

演習環境の調達

動かす環境としては、適当な VPS を想定しています。

うちで利用している、さくらの VPS (「3 仮想コア、メモリ 2GB、200GB SSD」の年額払いで月あたり 1480 円 (これは契約当時の価格で、いま新規契約すると二割ほど高めの模様)) あたりで十分のようです。AWS には Lightsail という VPS サービスがありますが、これには「2vCPU、メモリ 2GB、60GB SSD」という月額 12USD の品目があります (注:この 12USD は月額の請求上限値)。

ただしスポットのハンズオンといった臨時利用の場合、最低利用期間が問題です。さくらの VPS は 3 ヶ月ですが、Lightsail はサーバを削除すれば課金されないので、イベントの準備～本番までの契約分だけ (たとえば 1 週間分 3USD) の請求に出来ます。このくらいの仕様と価格が選定の基準になりそうです。

VPS のベンチマーク

価格だけでは実際の速度が分かりません。ベンチマークが必要です。いちおう、過去 30 年以上、さまざまな機種で、途中からはクラウドサービスも含めて、同じソフトウェア (Unix *bytebench*) でベンチマークし続けてきたデータがあります。それによれば、1 仮想コア (vCPU) の速度は、さくらの VPS と AWS Lightsail で、だいだい同じくらいです。

詳細は、次の URL で、ご堪能ください。

<https://technotes.fml.org/tech/benchmark/unixbench/>

EC2 で良かったのでは?

もちろん EC2 を使う案でも大丈夫です。むしろ作業手順に慣れていれば EC2 でセミナーの時間だけ起動するほうがよいでしょう。

今回は (a)(請求されるという意味で) 初めて本物の AWS を使ったということと (b) このあと IT インフラ部 (という名の 3 年生の課外活動?) が 3 ヶ月ほど使う予定があった (c) 単に使ってみたかった:-) ので、Lightsail にしました。きっと次の機会があれば EC2 でやるでしょう。

あとがき / おわりに

いかがだったでしょうか。感想や質問は隨時受け付けています。

この形に落ちつくまでの軌跡

(軌跡というわりに半日くらいの出来事ですけれども:-)

非常に基本的な演習内容なので、これなら docker でイケるのではないか？ と考え、1.5 人日くらい頑張って作りました。本書は、その備忘録兼解説のドキュメントです。

ことの発端は？ と言うと、LPIC のウェビナーでハンズオンをやることになったことでした。

ふだんの授業は AWS Academy でやっているのですが、一般のイベントに AWS Academy は利用できません。この際なので、AWS Cloud Formation あたりを真面目に勉強するという案も一瞬あたまをよぎりましたが、お金かかるから止めることにしました（正確には「設定を間違えると青天井な AWS が怖い症候群」という弱々な根性です;-）。しかしながら、非常に基本的な演習内容なので、大真面目に AWS EC2 を人数分用意しよう！ といった必要は無さそうに見えました。

そして、いちど作ったからには、何度でも、だれにでも利用できるべきです。長年フリーソフトウェア運動をやってますから、作るものは、すべてこのポリシーに従っています。だから、コマンド一発で「何度でも」「同じ環境を構築できる」ように作りこんでおきましたし、システムも github に出しておきました。システムにはドキュメントが必要です。正確には、まずドキュメントがあって、システムがあるべきなのです。そういうわけで、これが、そのドキュメント（解説書）になります。

♣ 著者紹介



深町賢一 (mailto:fukachan@fml.org)

"一文にもならんことを一生懸命やるのが本当の文化だ -- 梅棹忠夫"

- <https://github.com/fmlorg/>
- <https://www.fml.org/>
- <https://technotes.fml.org/>
- https://twitter.com/fukachan_fmlorg

IT インフラ演習環境 hands-on-base 0.3.0 の設計と実装

2024 年 6 月 8 日 v0.3.0

著 者 深町賢一

発行者 深町賢一

連絡先 fukachan@fml.org

<https://www.fml.org/>

@fukachan_fmlorg (https://twitter.com/fukachan_fmlorg)

印刷所 ○○印刷所

© 2024 深町賢一

(powered by Re:VIEW Starter)