Vol.2019-HPC-171 No.3
2019/9/20

ユーザ権限におけるDockerオーケストレーションの構築と撤去

畑中 智之[1,a)]　建部 修見[2]

概要：コンテナ技術は，プログラムとその実行環境をコンテナとしてまとめて実行する技術である．Dockerはデファクトスタンダードなコンテナソフトウェアであるが，実行にルート権限が必要で共有計算機環境では向かないという問題があった．Docker 19.03で実験的に導入されたRootlessモードは，デーモンをユーザ権限で動作させることを可能とする．共有計算機環境における利用が期待されているが，未だ研究段階である．オーケストレーションは，コンテナの配備を自動化することを指す．コンテナの管理や，オーバレイネットワークの構築によるコンテナ名での名前解決が可能となるため，HPCにおいても有用である．そこで本研究では，共有計算機環境でDockerのRootlessモードを使用したオーケストレーションの構築と撤去が実現可能であることを示した．また，オーケストレーションの構築，撤去を行うツールの提案を行った．

1. はじめに

コンテナ技術は，プログラムとその実行環境をコンテナとしてまとめて実行する技術である．開発環境と実行環境で同じ環境を用意できることや，環境をコードとして表現できることから，活用が進んでいる．産業界で用いられているコンテナソフトウェアにDocker[1]がある．Dockerはすでにデファクトスタンダードとなっているが，デーモンの実行にルート権限が必要であるため，HPCのような共有計算機環境では向かないという問題がある．

共有計算機環境でもコンテナ技術を利用するための研究に，次のようなものがある．Singularity[2], [3]，Charliecloud[4]は，いずれもユーザ権限で動作するDocker互換のコンテナソフトウェアである．また，Dockerコンテナ上のMPIアプリケーションを，dockerコマンドのラッパーを介して実行することでセキュリティを確保しようとする手法が提案されている[5]．TSUBAME3.0にはDockerが導入されているが，セキュリティを確保するため，ユーザが任意のイメージを実行することを許可していない[6]．いずれも，代替のコンテナソフトウェアを用いたり，Dockerの機能を制限したりすることでコンテナの実行を実現しており，Dockerそのものを動かすには至っていない．

DockerのRootlessモード（Rootless Docker）は，Docker 19.03で実験的に導入された，Dockerデーモンをユーザ権限で動作させるモードである[7]．ユーザ権限で動作させることで，セキュリティ上の問題を回避することが可能となる．そのため，共有計算機環境におけるDockerの利用が期待されているが，未だ研究段階である．

オーケストレーションは，コンテナの配備を自動化することを指す．Swarm mode[8]，Kubernetes[9]は，オーケストレーションツールの一例である．オーケストレーションを用いることで，コンテナの管理や，オーバレイネットワークの構築によるコンテナ名での名前解決が可能となるため，HPCにおいても有用である．

そこで本研究では，Rootless Dockerを使用したオーケストレーションの構築と撤去が可能であることを実証する．さらに，複数ノード間での設定が必要で複雑になりがちなオーケストレーションの構築と撤去を自動化するツールを提案する．

本論文は5章で構成される．まず，2章で関連研究について述べる．次に，3章でHPCにおいてオーケストレーションを使用したときに解決する問題と，その実現に関する問題を示し，4章で問題解決に向けたオーケストレーションツールの設計を行う．最後に，5章でまとめと課題について述べる．

2. 関連研究

2.1 modules

modules[10]は，必要なソフトウェアとそのバージョンを指定してロードする仕組みである．バージョンによる依

[1] 筑波大学大学院システム情報工学研究科
[2] 筑波大学計算科学研究センター
[a)] hatanaka@hpcs.cs.tsukuba.ac.jp

存関係を解決することができる．HPC では，複数のユーザで，同一のソフトウェアで異なるバージョンが必要となるケースがあるため，よく使用されている．必要なソフトウェアや，必要なバージョンがインストールされていない場合は，管理者にインストールを依頼する必要がある．

2.2 Singularity

Singularity[2], [3] は，Sylabs が提供しているコンテナソフトウェアである．HPC のような共有計算機環境で実行するため，ユーザ権限で動作するように設計されている．また，Docker のイメージを実行することや，Dockerfile からビルドした Singualrity コンテナをユーザ権限で実行することができる．

Singularity の Rootless モード [11] は，コンテナ内でルート権限を持つことができる．user_namespaces(7)[12] を活用することで実現している．なお，uid, gid のマッピングは，Rootless Docker でも用いられている．

2.3 Charliecloud

Charliecloud[4] は，Singularity と同様に，ユーザ権限で動作するコンテナソフトウェアである．ルート権限で動作する Docker よりもセキュアである．コンテナイメージは tarball として保存される．実行は，コンテナイメージを展開し，chroot(2)[13] を用いてファイルシステムのルートをコンテナ内に変更することで実現している．

2.4 Docker

Docker[1] は，デファクトスタンダードなコンテナソフトウェアである．dockerd と呼ばれるデーモンが常に起動しており，コンテナ管理を行う．コマンドラインからの命令は dockerd に伝えられ，dockerd がコンテナの起動や停止などの処理を行う．

dockerd の動作にルート権限が必要であるため，複数のユーザが使用する共有計算機環境での導入にはセキュリティ上の問題があった．Docker 19.03 で実験的に導入された Rootless モード（Rootless Docker）は，ユーザ権限で dockerd を動作させることができる [7]．そのため，HPC においても Docker の利用が期待されている．

2.5 Rootless Docker

Rootless Docker では，rootlesskit[14] により dockerd をユーザ権限で動作させ，さらにユーザ権限で操作できるネットワークにアタッチすることで動作する．uid のマッピングは user_namespaces(7)[12] によって行われ，ネットワークの分離は slirp4netns[15] や vpnkit[16] によって行われる．

2.5.1 rootlesskit

rootlesskit は，プロセスがあたかもルート権限で動作しているかのように見せるためのソフトウェアである [14]．具体的には，user_namespaces(7), mount_namespaces(7)[17] で namespaces(7)[18] を作成し，newuidmap(1)[19], newgidmap(1)[20] を，subuid(5)[21], subgid(5)[22] とともに実行することで実現している．rootlesskit を介することで，本来ルート権限がないと実行できない dockerd を，ユーザ権限で実行できるようになる．

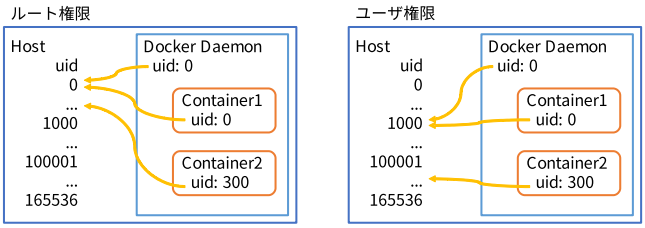


図 1　user_namespaces(7) によるマッピングの概略

2.5.2 user_namespaces(7) によるマッピング

Rootless Docker では，user_namespaces(7) によって uid, gid のマッピングを行う．図 1 は，user_namespaces(7) による uid のマッピングの概略である．ルート権限で動作する dockerd は，コンテナ内の uid とホストの uid が一致する（図 1 左）．ユーザ権限で動作する dockerd は，コンテナ内の uid が 0 のとき，dockerd を起動したユーザの uid と一致し，コンテナ内の uid が 0 でないとき，dockerd を起動したユーザが持つマッピング先の uid に一致する（図 1 右）．

ユーザが使用できる uid, gid の範囲は，subuid(5), subgid(5) によって設定される．この設定のみ，管理者による設定が必要である．

2.5.3 slirp4netns によるネットワークの分離

Docker は，コンテナとの通信や，仮想ネットワークを構築するため，ネットワークアダプタを作成する必要がある．一方，ユーザ権限ではこのような操作はできない．

slirp4netns[15] は，ユーザ権限のネットワークを構築することを可能とする．slirp によってネットワークを分離し，ユーザ権限でネットワークアダプタの作成，削除等の操作が行えるようにする．また，ホストと NAT で接続することで，ホスト上のネットワークと通信できるようにする．

slirp4netns によって仮想化されたネットワーク内であれば，ユーザ権限でもネットワークの操作ができるため，dockerd の実行が可能となる．

2.6 docker-compose

docker-compose は，単一ホスト上で，複数のコンテナの起動を自動化するソフトウェアである [23]．動作させるコンテナを yaml ファイルとして定義でき，その定義に従いコンテナの立ち上げを行うことができる．ただし，複数ノードにわたってのコンテナの起動はできない．

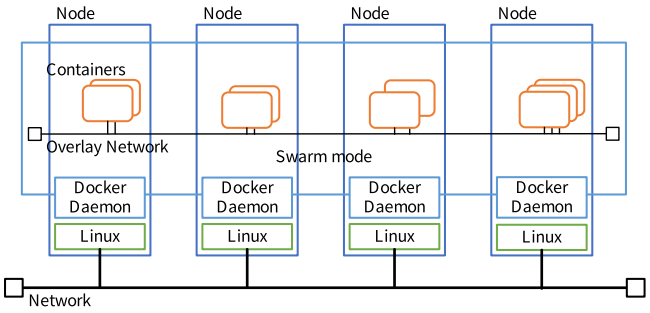


図 2　オーケストレーションの構成例

2.7　オーケストレーション

オーケストレーションは，コンテナの配備を自動化することを指す．オーケストレーションの構成例を図 2 に示す．オーケストレーションを導入することで，ノードの管理，コンテナの管理，スケジューリングの自動化が可能となる．また，ノード間でオーバレイネットワークを構築し，その上でコンテナを動作させるため，複数ノード間でもコンテナ名での名前解決が可能となる．

オーケストレーションツールに，Swarm mode[8], Kubernetes[9] がある．Swarm mode は，Docker がデフォルトでサポートしているオーケストレーションツールである．docker-compose.yml を使用して，docker stack コマンドでデプロイを行うことができるため，docker-compose と相性が良い．Kubernetes は，Google が設計したオーケストレーションツールである．エンタープライズでのオーケストレーションではデファクトスタンダードとなっている．

3.　HPC におけるオーケストレーション

HPC において Rootless Docker とオーケストレーションを利用することで，任意のコンテナを，任意のノードに分配することが可能となる．さらに，オーバレイネットワークの構築によって，各コンテナは任意のホスト名で名前解決を行うことが可能となる．

一方，HPC でオーケストレーションを実行するには，2 点課題がある．まず，HPC においてオーケストレーションの利用が可能であるかが明らかになっていない点である．もう一つは，オーケストレーションの構築，撤去には煩雑な処理が必要となる点である．

HPC でオーケストレーションを実行するには，これら 2 点の課題の解決が必要となる．

4.　オーケストレーションの設計

HPC において Rootless Docker とオーケストレーションを用いることは有益だが，課題がある．そこで，その課題の解消を目的に，HPC におけるオーケストレーションの実現可能性を実証し，その構築と撤去を行うツールを提案する．

オーケストレーションには，Docker がデフォルトでサポートしている Swarm mode を採用する．産業界では Kubernetes がデファクトスタンダードとなっているが，学習コストが高い．一方，Swarm mode が有する機能は Kubernetes よりも少ないが，コンテナの配備，オーバレイネットワークの構築等，HPC で使用するには十分な機能を有している．また，docker-compose との相性もよいため，Swarm mode を用いたオーケストレーションを使用する．

4.1 節，4.2 節でオーケストレーションの構築と撤去が可能であることを示し，4.3 節で構築と撤去を行うツールを提案する．

4.1　オーケストレーションの構築

共有計算機環境上でオーケストレーションの実現が可能であることを実証するため，各ノードで手動で操作可能な状態において，次の手順で，Rootless Docker を用いた Swarm クラスタの構築ができることを確認した．図 3 は，Swarm クラスタを構築する大まかな流れである．なお，subuid(5), subgid(5) はそれぞれ設定済みで，ホスト間はそれぞれ通信できるように設定されている．

(1) rootlesskit を使用して，Rootless Docker デーモンを立ち上げる．このとき，クラスタを管理するために 2377/tcp を，ノード間で通信するために 7946/tcp と 7946/udp を，オーバレイネットワークの通信のために 4789/udp を，ホストに公開する設定をする．また，/etc, /run, /var/lib をコピーして，ユーザ権限での書き込みを可能にする．
(2) Swarm クラスタを初期化する前に，オプションに com.docker .network.bridge.enable_icc=true を指定した上で docker_gwbridge を作成する．
(3) マネージャーノードで docker swarm init を実行し，Swarm クラスタを初期化する．
(4) マネージャーノードで docker swarn join-token を実行し，Swarm クラスタに参加するためのトークンを得る．
(5) 各ワーカーノードで docker swarm join を実行し，Swarm クラスタに参加する．
(6) マネージャーノードで，Swarm クラスタに参加するノードが揃ったことを確認する．

Swarm クラスタの構築の確認は，図 4 の docker-compose.yml を使用して docker stack deploy コマンドを実行することで行った．docker-compose.yml には，正しくコンテナが分配されていることを確認するための client を 4 つと，通信を確認するためにサービスを提供する httpd を 1 つ展開する設定がされている．この docker-compose.yml をもとに，スタックをデプロイした．デプロイ後，まず，各ノードで docker ps コマンドを実行することで，指定した数だけコンテナが分配されていることを確認した．次に，

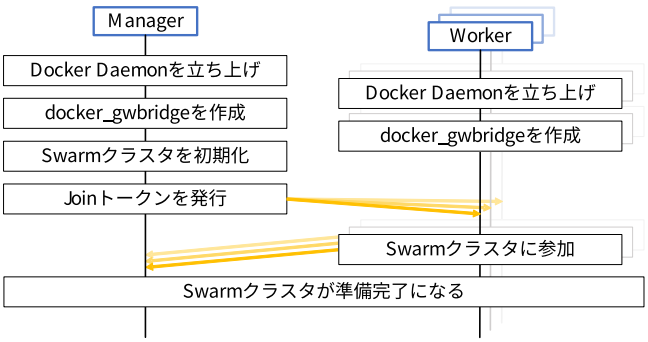


図 3　Swarm クラスタ構築の手順

```
1  version: "3"
2  services:
3    client:
4      command: "tail -f /dev/null"
5      deploy:
6        replicas: 4
7      hostname: "client"
8      image: "ubuntu"
9    httpd:
10     deploy:
11       replicas: 1
12     hostname: "httpd"
13     image: "httpd"
```

図 4　Swarm クラスタ構築の検証に用いた docker-compose.yml

```
1  $ docker exec -it {CONTAINER} /bin/bash
2  root@client:/# apt update && apt
       install -y curl
3  root@client:/# curl httpd
4  <html><body><h1>It
       works!</h1></body></html>
5  root@client:/#
```

図 5　httpd と curl による疎通確認（apt の出力は省略している）

client のうちひとつに docker exec で入り，client 内で curl をインストールした上で，http://httpd:80 を取得できることを確認した（図 5）．

これにより，ここで示した手順でオーケストレーションを構築することが可能できること，docker stack コマンドによりデプロイできること，オーバレイネットワークによるコンテナ間の通信ができることが実証されたといえる．

4.2　オーケストレーションの撤去

次の手順で，Rootless Docker を用いた Swarm クラスタを撤去できることを確認した．

(1) Rootless Docker デーモンを停止する．
(2) Rootless Docker が使用したディレクトリを削除する．

本来ならば，ワーカーノードによる Swarm クラスタからの離脱，マネージャーノードによる Swarm クラスタからの離脱，Docker デーモンの停止が必要である．しかし，Docker のディレクトリをそのまま削除しても，Docker が使用したイメージや，作成したネットワーク等の設定が削除されるのみである．使用される物理ノードがその都度異なる HPC では，残しておいた場合でも再び使えるとは限らないため，Docker のディレクトリをそのまま削除する手法を取った．

4.3　構築ツールの提案

4.1 節，4.2 節で確認したオーケストレーションの構築と撤去の手順は，各ノードで実行状態を確認しながら構築する方法であった．そのため，自動での構築には適用できないという問題がある．そこで，先の手順をもとに，Swarm クラスタの構築と撤去を自動で実行する構築ツールを提案する．

具体的には，図 6 に従い，構築と撤去を行う．構築は，まず，マネージャーノードで起動した構築ツールが，各ノードで RPC サーバを立ち上げる．次に，Dockerd の起動コマンドを基点に，RPC サーバを経由して，各ノードで dockerd を立ち上げる．最後に，Swarm の構築コマンドを基点に，立ち上げた dockerd に対して Swarm クラスタの初期化などの必要な処理を行っていく．これらの処理によって，Swarm クラスタの構築が完了する．なお，処理に失敗した場合は，一定時間待ったのち再度実行する．撤去は，構築ツールに Ctrl+C（SIGINT）を送ることで，デーモンの停止と，使用したディレクトリの削除を行う．つまり，プロセスに kill コマンドを用いて SIGINT を送ることで停止が可能である．

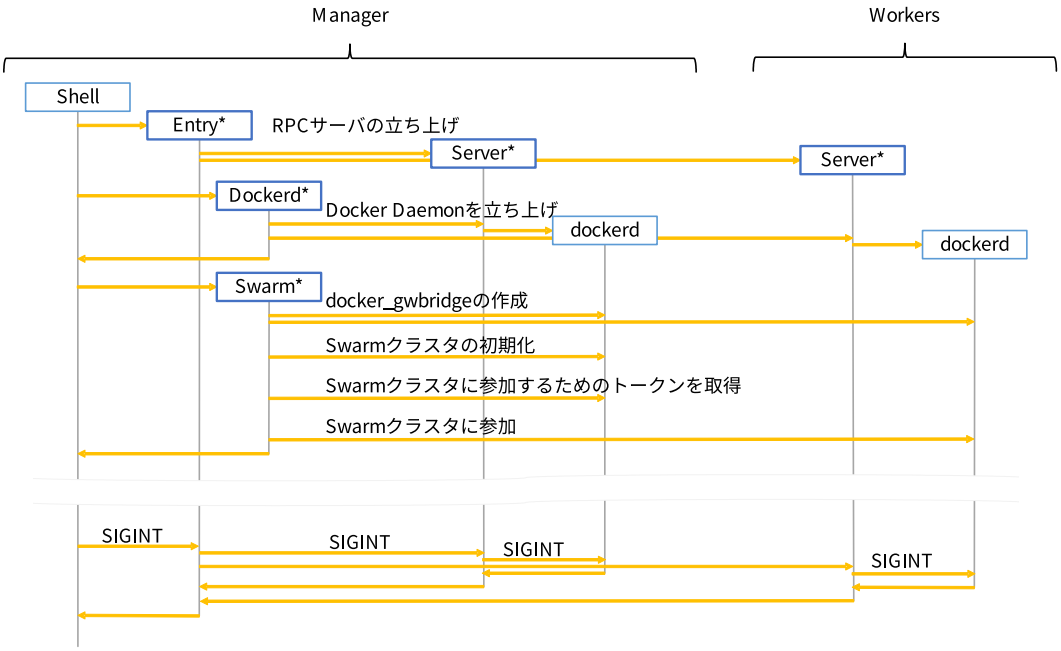
この構築ツールによって，Rootless Docker を用いた Swarm クラスタの構築と撤去が容易に行うことが可能となる．これにより，HPC においてもオーケストレーションが容易に実現できるようになる．

5.　まとめと課題

本研究では，Rootless Docker を使用したオーケストレーションの構築と撤去が実現可能であることを示した．また，オーケストレーションの構築，撤去を行うツールの提案を行った．

一方，Rootless Docker とオーケストレーションを用いて計算したときの性能評価は未実施である．デーモンを起動し，クラスタを構築するまでの時間の評価や，Docker を介する場合と介さない場合の性能評価が必要となるため，今後の課題としたい．

謝辞　本研究の一部は，JSPS 科研費 17H01748，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），JST CREST JPMJCR1414 および富士通研究所との共同



注: *印は，構築ツールが提供するコマンド

図 6 オーケストレーションツールの動作

研究による．

参考文献

[1] Docker: Enterprise Container Platform | Docker, Docker Inc. (online), available from ⟨https://www.docker.com/⟩ (accessed 2019-07-26).

[2] Kurtzer, G. M., Sochat, V. and Bauer, M. W.: Singularity: Scientific containers for mobility of compute, *PLOS ONE*, Vol. 12, No. 5, pp. 1–20 (online), DOI: 10.1371/journal.pone.0177459 (2017).

[3] Sochat, V. V., Prybol, C. J. and Kurtzer, G. M.: Enhancing reproducibility in scientific computing: Metrics and registry for Singularity containers, *PLOS ONE*, Vol. 12, No. 11, pp. 1–24 (online), DOI: 10.1371/journal.pone.0188511 (2017).

[4] Priedhorsky, R. and Randles, T.: Charliecloud: Unprivileged Containers for User-defined Software Stacks in HPC, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, ACM, pp. 36:1–36:10 (online), DOI: 10.1145/3126908.3126925 (2017).

[5] Bayser, M. and Cerqueira, R.: Integrating MPI with Docker for HPC, *2017 IEEE International Conference on Cloud Engineering (IC2E)*, Los Alamitos, CA, USA, IEEE Computer Society, pp. 259–265 (online), DOI: 10.1109/IC2E.2017.40 (2017).

[6] Nomura, A.: Introducing Container Technology to TSUBAME3.0 Supercomputer (2019). Presentation in Containers for Acceleration and Accessibility in HPC and Cloud Ecosystems.

[7] Docker: Docker Engine release notes | Docker Documentation, Docker Inc. (online), available from ⟨https://docs.docker.com/engine/release-notes/#19030⟩ (accessed 2019-07-30).

[8] Docker: Docker Swarm overview | Docker Documentation, Docker Inc. (online), available from ⟨https://docs.docker.com/swarm/overview/⟩ (accessed 2019-07-26).

[9] Kubernetes: Production-Grade Container Orchestration - Kubernetes, The Kubernetes Authors (online), available from ⟨https://kubernetes.io/⟩ (accessed 2019-07-30).

[10] Furlani, J. L. and Osel, P. W.: Abstract Yourself With Modules, *Proceedings of the 10th USENIX Conference on System Administration*, LISA '96, Berkeley, CA, USA, USENIX Association, pp. 193–204 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1029824.1029858⟩ (1996).

[11] Singularity: Fakeroot feature - Singularity container 3.3 documentation, Sylabs Inc. (online), available from ⟨https://sylabs.io/guides/3.3/user-guide/fakeroot.html⟩ (accessed 2019-08-13).

[12] The Linux man-pages project: user_namespaces(7), (online), available from ⟨http://man7.org/linux/man-pages/man7/user_namespaces.7.html⟩ (accessed 2019-08-13).

[13] The Linux man-pages project: chroot(2), (online), available from ⟨http://man7.org/linux/man-pages/man2/chroot.2.html⟩ (accessed 2019-08-13).

[14] rootlesskit: GitHub - rootless-containers/rootlesskit: kind of Linux-native "fake root" utility, made for mainly running Docker and Kubernetes as an unprivileged user, rootless-containers (online), available from ⟨https://github.com/rootless-containers/rootlesskit⟩ (accessed 2019-07-30).

[15] slirp4netns: GitHub - rootless-containers/slirp4netns: User-mode networking for unprivileged network namespaces, rootless-containers (online), available from ⟨https://github.com/rootless-containers/slirp4netns⟩ (accessed 2019-07-30).

[16] vpnkit: GitHub - moby/vpnkit: A toolkit for embedding VPN capabilities in your application, moby (online), available from ⟨https://github.com/moby/vpnkit⟩

(accessed 2019-08-03).
[17] The Linux man-pages project: mount_namespaces(7), (online), available from ⟨http://man7.org/linux/man-pages/man7/mount_namespaces.7.html⟩ (accessed 2019-08-13).
[18] The Linux man-pages project: namespaces(7), (online), available from ⟨http://man7.org/linux/man-pages/man7/namespaces.7.html⟩ (accessed 2019-08-13).
[19] The Linux man-pages project: newuidmap(1), (online), available from ⟨http://man7.org/linux/man-pages/man1/newuidmap.1.html⟩ (accessed 2019-08-13).
[20] The Linux man-pages project: newgidmap(1), (online), available from ⟨http://man7.org/linux/man-pages/man1/newgidmap.1.html⟩ (accessed 2019-08-13).
[21] The Linux man-pages project: subuid(5), (online), available from ⟨http://man7.org/linux/man-pages/man5/subuid.5.html⟩ (accessed 2019-08-13).
[22] The Linux man-pages project: subgid(5), (online), available from ⟨http://man7.org/linux/man-pages/man5/subgid.5.html⟩ (accessed 2019-08-13).
[23] Docker: Overview of Docker Compose | Docker Documentation, Docker Inc. (online), available from ⟨https://docs.docker.com/compose/⟩ (accessed 2019-08-03).