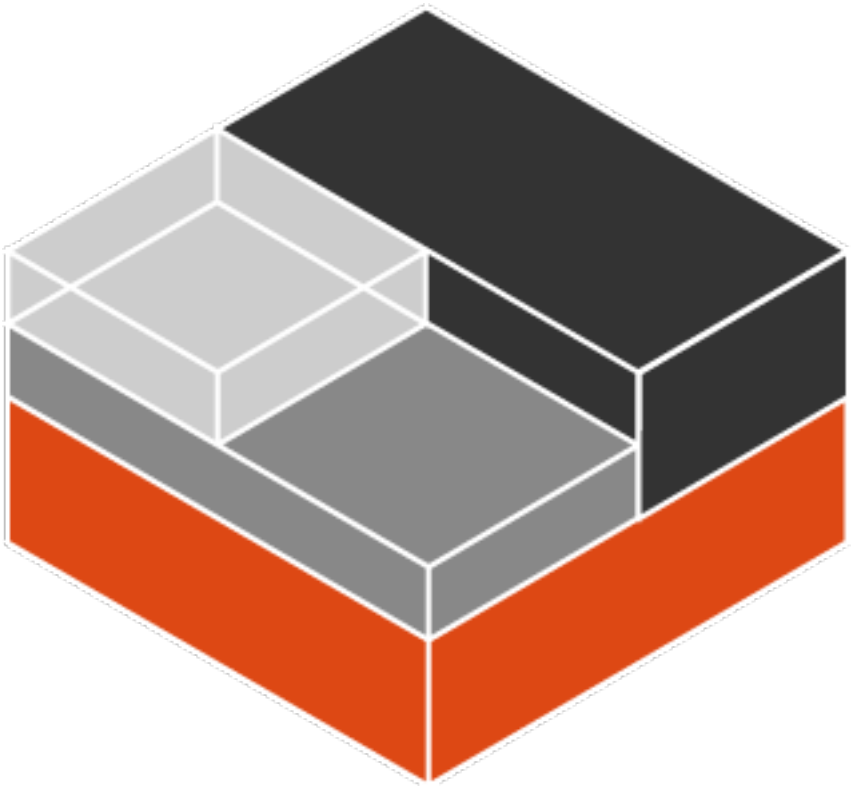
LXD

LXD contributors

2023 年 07 月 07 日

目次



LXD は、次世代のシステムコンテナおよび仮想マシンマネージャです。

コンテナや仮想マシンの中で動作する完全な Linux システムに統一されたユーザーエクスペリエンスを提供します。LXD は 数多くの Linux ディストリビューション のイメージを提供しており、非常にパワフルでありながら、それでいてシンプルな REST API を中心に構築されています。LXD は単一のマシン上の単一のインスタンスからデータセンターのフルラック内のクラスタまでスケールし、開発とプロダクションの両方のワークロードに適しています。

LXD を使えば小さなプライベートクラウドのように感じられるシステムを簡単にセットアップできます。あなたのマシン資源を最適に利用しながら、あらゆるワークロードを効率よく実行できます。

さまざまな環境をコンテナ化したい場合や仮想マシンを稼働させたい場合、あるいは一般にあなたのインフラを費用効率よく稼働および管理したい場合には LXD を使うのを検討するのがお勧めです。

第 1 章

使い始めるには

LXD とは何か、何ができるのか、より良いアイデアを得るためには、オンラインで試すことができます! また、ローカルで動作させたい場合は、LXD を使い始めるにはをご覧ください。

- リリースのアナウンス: https://linuxcontainers.org/ja/lxd/news/
- リリースの tarball: https://linuxcontainers.org/ja/lxd/downloads/
- ドキュメント: https://lxd-ja.readthedocs.io/ja/latest/

第 2 章

セキュリティ

LXD のインストールが安全であることを保証するために、以下の点を考慮してください。

- オペレーティングシステムを最新に保ち、利用可能なすべてのセキュリティパッチをインストールする。
- サポートされている LXD のバージョン（LTS リリースまたは月例機能リリース）のみを使用する。
- LXD デーモンとリモート API へのアクセスを制限すること。
- 必要とされない限り、特権コンテナを使わないこと。特権的なコンテナを使う場合は、適切なセキュリティ対策をしてください。詳細は LXC セキュリティページを参照してください。
- ネットワークインターフェイスを安全に設定してください。

詳細は**セキュリティ**をご覧ください。

重要: UNIX ソケットを介した LXD へのローカルアクセスは、常に LXD へのフルアクセスを許可します。これは、任意のインスタンス上のセキュリティ機能を変更できる能力に加えて、任意のインスタンスにファイルシステムパスやデバイスをアタッチする能力を含みます。

したがって、あなたのシステムへのルートアクセスを信頼できるユーザーにのみ、このようなアクセスを与えるべきです。

第 3 章

プロジェクトとコミュニティ

LXD はフリーソフトウェアであり Apache 2 ライセンス で開発されています。これはコミュニティのプロジェクト、コントリビューション、提案、修正、建設的なフィードバックを温かく迎えるオープンソースプロジェクトです。

LXD プロジェクトは Canonical Ltd にスポンサーされています。

- Code of Conduct
- **プロジェクトに貢献する**
- **サポートを得る**
- YouTube 上のチュートリアルとアナウンスを見る
- IRC で議論する (必要なら Getting started with IRC 参照)
- フォーラムで質問と回答する
- メーリングリストに参加する

3.1 LXD を使い始めるには

このセクションのドキュメントに加えて、ウェブサイト上の LXD を使い始めるには を参照してください。

3.1.1 動作環境

Go

LXD は Go 1.18 以上を必要とし、Golang のコンパイラのみでテストされています。(訳注: 以前は gccgo もサポートされていましたが Golang のみになりました)

ビルドには最低 2GB の RAM を推奨します。

必要なカーネルバージョン

サポートされる最小のカーネルバージョンは 5.4 です。

LXD には以下の機能をサポートするカーネルが必要です。

- Namespaces (`pid`, `net`, `uts`, `ipc` と `mount`)
- Seccomp
- Native Linux AIO (`io_setup(2)` など)

以下のオプションの機能はさらなるカーネルオプションを必要とします。

- Namespaces (`user` と `cgroup`)
- AppArmor (mount mediation に対する Ubuntu パッチを含む)
- Control Groups (`blkio`, `cpuset`, `devices`, `memory`, `pids` と `net_prio`)
- CRIU (正確な詳細は CRIU のアップストリームを参照のこと)

さらに使用している LXC のバージョンで必要とされる他のカーネルの機能も必要です。

LXC

LXD は以下のビルドオプションでビルドされた LXC 4.0.0 以上を必要とします。

- `apparmor` (もし LXD の AppArmor サポートを使用するのであれば)
- `seccomp`

Ubuntu を含む、さまざまなディストリビューションの最近のバージョンを動かすためには、LXCFS もインストールする必要があります。

QEMU

仮想マシンを利用するには QEMU 6.0 以降が必要です。

追加のライブラリー (と開発用のヘッダ)

LXD はデータベースとして `dqlite` を使用しています。ビルドしセットアップするためには `make deps` を実行してください。

LXD は他にもいくつかの (たいていはパッケージ化されている) C ライブラリーを使用しています。

- `libacl1`
- `libcap2`
- `liblz4` (`dqlite` で使用)
- `libuv1` (`dqlite` で使用)
- `libsqlite3` >= 3.25.0 (`dqlite` で使用)

ライブラリーそのものとライブラリーの開発用ヘッダ (`-dev` パッケージ) の全てをインストールしたことを確認してください。

3.1.2 LXD をインストールするには

LXD をインストールする最も簡単な方法は**提供されているパッケージのどれかをインストールする**ことですが、**ソースから *LXD* をインストールする**こともできます。

LXD のインストール後、`lxd` グループがシステム内に存在することを確認してください。このグループ内のユーザが LXD を操作できます。手順は ***LXD* へのアクセスを管理する**を参照してください。

リリースを選択する

LXD は異なるリリースブランチを並行して維持しています。

- 長期サポート (LTS) リリース：現在は LXD 5.0.x と LXD 4.0.x
- 機能リリース：LXD 5.x

本番環境には LTS を推奨します。通常のバグフィクスとセキュリティアップデートの恩恵を受けられるからです。しかし、長期リリースには新しい機能はやどんな種類の挙動の変更も追加されません。

LXD の最新の機能と毎月の更新を得るには、代わりに機能リリースを使ってください。

LXD をパッケージからインストールする

LXD デーモンは Linux でのみ稼働します。クライアントツール (lxc) はほとんどのプラットフォームで利用できます。

Linux

LXD をインストールする最も簡単な方法は *Snap* **パッケージ**をインストールすることです。これはさまざまな Linux ディストリビューションで利用可能です。

この選択肢が使えない場合、**他のインストール方法**を参照してください。

Snap パッケージ

LXD はいくつかの Linux ディストリビューション (例えば、Ubuntu、Arch Linux、Debian、Fedora、そして OpenSUSE) で動作する snap パッケージを公開しテストしています。

snap をインストールするには以下の手順を実行してください。

1. 提供されているディストリビューション一覧を見て、お使いの Linux ディストリビューションで利用可能かを確認してください。利用可能ではない場合、**他のインストール方法**のいずれかで対応してください。
2. snapd をインストールします。Snapcraft ドキュメントインストール手順を参照してください。
3. snap パッケージをインストールします。最新の機能リリースをインストールするには以下のようにします。

   ```
   sudo snap install lxd
   ```

   LXD 5.0 LTS リリースの場合は以下のようにします。

   ```
   sudo snap install lxd --channel=5.0/stable
   ```

LXD の snap パッケージについてより詳細な情報 (上記以外のバージョン、更新の管理など) については Managing the LXD snap を参照してください。

注釈: Ubuntu 18.04 では、もし LXD の deb パッケージを過去にインストールしていた場合、既存の全てのデータを以下のコマンドで移行できます。

```
sudo lxd.migrate
```

他のインストール方法

いくつかの Linux ディストリビューションでは snap パッケージ以外のインストール方法を提供しています。

Alpine Linux

Alpine Linux で機能リリースの LXD をインストールするには、以下のようにします。

```
apk add lxd
```

Arch Linux

Arch Linux で機能リリースの LXD をインストールするには、以下のようにします。

```
pacman -S lxd
```

Fedora

LXC/LXD の Fedora RPM パッケージが COPR レポジトリで利用可能です。

機能リリースの LXD パッケージをインストールするには、以下のようにします。

```
dnf copr enable ganto/lxc4
dnf install lxd
```

インストール手順のより詳細な情報についてはインストールガイドを参照してください。

Gentoo

Gentoo で機能リリースの LXD をインストールするには、以下のようにします。

```
emerge --ask lxd
```

他のオペレーティングシステム

重要: 他のオペレーティングシステム向けのビルドはクライアントのみを含み、サーバーは含みません。

---

macOS

LXD は macOS の LXD クライアントのビルドを Homebrew で公開しています。

機能リリースの LXD をインストールするには、以下のようにします。

```
brew install lxc
```

Windows

Windows 版の LXD クライアントは Chocolatey パッケージとして提供されています。インストールするためには以下のようにします。

1. インストール手順に従って Chocolatey をインストールします。
2. LXD クライアントをインストールします。

   ```
   choco install lxc
   ```

GitHub にも LXD クライアントのネイティブビルドがあります。特定のビルドをダウンロードするには以下のようにします。

1. GitHub アカウントにログインします。
2. 興味のあるブランチやタグ (例えば、最新のリリースタグあるいは `master`) でフィルタリングします。
3. 最新のビルドを選択し、適切なアーティファクトをダウンロードします。

LXD をソースからインストールする

LXD をソースコードからビルドとインストールしたい場合、以下の手順に従ってください。

LXD の開発には `liblxc` の最新バージョン (4.0.0 以上が必要) を使用することをおすすめします。さらに LXD が動作するためには Golang 1.18 以上が必要です。Ubuntu では次のようにインストールできます:

```
sudo apt update
sudo apt install acl attr autoconf automake dnsmasq-base git golang libacl1-dev libcap-
↪dev liblxc1 liblxc-dev libsqlite3-dev libtool libudev-dev liblz4-dev libuv1-dev make␣
↪pkg-config rsync squashfs-tools tar tcl xz-utils ebtables
```

デフォルトのストレージドライバである `dir` ドライバに加えて、LXD ではいくつかのストレージドライバが使えます。これらのツールをインストールすると、initramfs への追加が行われ、ホストのブートが少しだけ遅くなるかもしれませんが、特定のドライバを使いたい場合には必要です:

```
sudo apt install lvm2 thin-provisioning-tools
sudo apt install btrfs-progs
```

テストスイートを実行するには、次のパッケージも必要です:

```
sudo apt install curl gettext jq sqlite3 socat bind9-dnsutils
```

ソースから最新版をビルドする

この方法は LXD の最新版をビルドしたい開発者や Linux ディストリビューションで提供されない LXD の特定のリリースをビルドするためのものです。Linux ディストリビューションへ統合するためのソースからのビルドはここでは説明しません。それは将来、別のドキュメントで取り扱うかもしれません。

```
git clone https://github.com/lxc/lxd
cd lxd
```

これで LXD の現在の開発ツリーをダウンロードしてソースツリー内に移動します。その後下記の手順にしたがって実際に LXD をビルド、インストールしてください。

ソースからリリース版をビルドする

LXD のリリース tarball は完全な依存ツリーと `libraft` と LXD のデータベースのセットアップに使用する `libdqlite` のローカルコピーをバンドルしています。

```
tar zxvf lxd-4.18.tar.gz
cd lxd-4.18
```

これでリリース tarball を解凍し、ソースツリー内に移動します。その後下記の手順にしたがって実際に LXD をビルド、インストールしてください。

ビルドを開始する

実際のビルドは Makefile の 2 回の別々の実行により行われます。1 つは `make deps` でこれは LXD に必要とされるライブラリーをビルドします。もう 1 つは `make` で LXD 自体をビルドします。`make deps` の最後に `make` の実行に必要な環境変数を設定するための手順が表示されます。新しいバージョンの LXD がリリースされたらこれらの環境変数の設定は変わるかもしれませんので、`make deps` の最後に表示された手順を使うようにしてください。下記の手順 (例示のために表示します) はあなたがビルドする LXD のバージョンのものとは一致しないかもしれません。

ビルドには最低 2GB の RAM を搭載することを推奨します。

```
user@host:~$ make deps              ...make[1]:  Leaving directory '/root/go/deps/dqlite'#
environment Please set the following in your environment (possibly ~/.bashrc)# export
CGO_CFLAGS="${CGO_CFLAGS} -I$(go env GOPATH)/deps/dqlite/include/ -I$(go env GOPATH)/
deps/raft/include/"# export CGO_LDFLAGS="${CGO_LDFLAGS} -L$(go env GOPATH)/deps/
dqlite/.libs/ -L$(go env GOPATH)/deps/raft/.libs/"# export LD_LIBRARY_PATH="$(go env
```

GOPATH)/deps/dqlite/.libs/:$(go env GOPATH)/deps/raft/.libs/:${LD_LIBRARY_PATH}"# export CGO_LDFLAGS_ALLOW="(-Wl,-wrap,pthread_create)|(-Wl,-z,now)" user@host:~$ make

ソースからのビルド結果のインストール

ビルドが完了したら、ソースツリーを維持したまま、あなたのお使いのシェルのパスに$(go env GOPATH)/bin を追加し、LD_LIBRARY_PATH 環境変数を make deps で表示された値に設定すれば、 LXD が利用できます。~/.bashrc ファイルの場合は以下のようになります。

```
export PATH="${PATH}:$(go env GOPATH)/bin"
export LD_LIBRARY_PATH="$(go env GOPATH)/deps/dqlite/.libs/:$(go env GOPATH)/deps/raft/.
↪libs/:${LD_LIBRARY_PATH}"
```

これで lxd と lxc コマンドの実行ファイルが利用可能になり LXD をセットアップするのに使用できます。LD_LIBRARY_PATH 環境変数のおかげで実行ファイルは$(go env GOPATH)/deps にビルドされた依存ライブラリーを自動的に見つけて使用します。

マシンセットアップ

LXD が非特権コンテナを作成できるように、root ユーザーに対する sub{u,g}id の設定が必要です。

```
echo "root:1000000:1000000000" | sudo tee -a /etc/subuid /etc/subgid
```

これでデーモンを実行できます (sudo グループに属する全員が LXD とやりとりできるように --group sudo を指定します。別に指定したいグループを作ることもできます)。

```
sudo -E PATH=${PATH} LD_LIBRARY_PATH=${LD_LIBRARY_PATH} $(go env GOPATH)/bin/lxd --group␣
↪sudo
```

注釈: newuidmap/newgidmap ツールがシステムに存在し、/etc/subuid、/etc/subgid が存在する場合は、root ユーザーに少なくとも 10M の UID/GID の連続した範囲を許可するように設定する必要があります。

LXD へのアクセスを管理する

LXD のアクセス制御はグループのメンバーシップに基づいています。root ユーザと `lxd` グループの全てのメンバーはローカルデーモンとやりとりできます。詳細は *LXD* **デーモンへのアクセス**を参照してください。

お使いのシステムに `lxd` グループが存在しない場合は、作成して LXD デーモンを再起動してください。このグループに追加されたメンバーは LXD の完全な制御ができます。

グループのメンバーシップは通常ログイン時にのみ適用されますので、セッションを開き直すか、LXD とやりとりするシェル上で `newgrp lxd` コマンドを実行する必要があります。

重要: UNIX ソケットを介した LXD へのローカルアクセスは、常に LXD へのフルアクセスを許可します。これは、任意のインスタンス上のセキュリティ機能を変更できる能力に加えて、任意のインスタンスにファイルシステムパスやデバイスをアタッチする能力を含みます。

したがって、あなたのシステムへのルートアクセスを信頼できるユーザーにのみ、このようなアクセスを与えるべきです。

LXD をアップグレードする

LXD を新しいバージョンにアップグレードした後、LXD はデータベースを新しいスキーマにアップデートする必要があるかもしれません。このアップデートは LXD のアップグレードの後のデーモン起動時に自動的に実行されます。アップデート前のデータベースのバックアップはアクティブなデータベースと同じ場所 (例えば snap の場合は`/var/snap/lxd/common/lxd/database`) に保存されます。

重要: スキーマのアップデート後は、古いバージョンの LXD はデータベースを無効とみなすかもしれません。これはつまり LXD をダウングレードしてもあなたの LXD の環境は利用不可能と言われるかもしれないということです。

このようなダウングレードが必要な場合は、ダウングレードを行う前にデータベースのバックアップをリストアしてください。

3.1.3 LXD を初期化するには

LXD インスタンスを作成する前に、LXD を設定と初期化する必要があります。

対話式の設定

対話式の設定プロセスを開始するには以下のコマンドを実行します。

```
lxd init
```

注釈: シンプルな設定では、このコマンドは通常ユーザで実行できます。しかし、初期化プロセス中により高度な操作 (例えば、既存のクラスタに参加するなど) を行う際は root 権限が必要な場合があります。この場合は、コマンドを sudo 付きで実行するか root ユーザで実行してください。

このツールは必要な設定を決定するために一連の質問をします。質問はあなたが入力した回答に応じて動的に調整されます。質問は以下の領域をカバーします。

クラスタリング (**クラスタリングについて**と**クラスタを形成するには参照**)
: クラスタは複数の LXD サーバーを結合します。クラスタメンバーは同じ分散データベースを共有し、LXD クライアント (`lxc`) や REST APIT を使って統一的に管理できます。

  デフォルトの回答は no で、クラスタリングは有効化されません。`yes` と回答すると、既存のクラスタに接続するか、クラスタを新規作成できます。

MAAS **サポート** (`maas.io` と MAAS - How to manage VM hosts **参照**)
: MAAS はベアメタルサーバーのデータセンターをビルドできるオープンソースのツールです。

  デフォルトの回答は no で、MAAS サポートは有効化されません。`yes` と回答すると、`name`、`URL`、`API key` を指定して既存の MAAS サーバーに接続できます。

ネットワーク (**ネットワークについて**と**ネットワークデバイス参照**)
: インスタンスにネットワークへのアクセスを提供します。

  LXD に新しいブリッジを作成させる (推奨) こともできますし、既存のネットワークブリッジやインタフェースを使うこともできます。

  後から追加のブリッジを作成して、インスタンスに割り当てることもできます。

ストレージプール (**ストレージプール、ボリューム、バケットについて**と**ストレージドライバ参照**)
: インスタンス (と他のデータ) はストレージプール内に保管されます。

  お試し用にはループバックベースのストレージプールを作ることもできます。しかし、本番環境での利用には、ループバックベースのストレージではなく空のパーティション (または完全なディスク) を使うほうが良いです (ループバックベースのストレージのほうが遅くサイズを縮小できないため)。

  お勧めのバックエンドは `zfs` と `btrfs` です。

  後から追加のストレージプールを作成することもできます。

リモートアクセス (**リモート *API* へのアクセスとリモート *API* 認証参照**) ネットワーク越しにサーバーにリモートアクセスできるようにします。

デフォルトの回答は no で、リモートアクセスは有効化されません。yes と回答すると、ネットワーク越しにサーバーに接続できるようになります。

サーバーにクライアント証明書を追加する (手動にてあるいはトークンを使用して。これが推奨) かトラストパスワードを設定できます。

イメージの自動更新 (**イメージについて参照**) イメージサーバーからイメージをダウンロードできます。この場合、イメージを自動的に更新するようにできます。

デフォルトの回答は yes で、LXD はダウンロードされたイメージを定期的に更新します。

YAML `lxd init` **プリシード** (**非対話式の設定参照**)
yes と回答すると、このコマンドはあなたが選択した設定オプションのサマリをターミナルに表示します。

最小構成のセットアップ

デフォルトのオプションで最小構成のセットアップを作成する場合は、`lxd init` コマンドに`--minimal` フラグを追加することで、設定の行程をスキップできます。

```
lxd init --minimal
```

注釈: 最小構成のセットアップは基本的な設定は提供しますが、設定は速度や機能に最適化されません。特に、デフォルトで使用される ***dir* ストレージドライバ**は他のドライバより遅く、高速なスナップショット、インスタンスのコピーや起動、クォータや最適化されたバックアップを提供しません。

最適化された環境を使いたい場合は、代わりに対話式の設定プロセスを行ってください。

非対話式の設定

`lxd init` コマンドは`--preseed` コマンドラインオプションをサポートし、LXD デーモンの設定、ストレージプール、ネットワークデバイス、プロファイルを YAML プリシードファイルを使って非対話的に設定できます。

例えば、完全に新しく LXD をインストールした状態から始める場合、以下のコマンドで LXD を設定することもできます。

```
    cat <<EOF | lxd init --preseed
config:
  core.https_address: 192.0.2.1:9999
```

(次のページに続く)

(前のページからの続き)

```
  images.auto_update_interval: 15
networks:
- name: lxdbr0
  type: bridge
  config:
    ipv4.address: auto
    ipv6.address: none
EOF
```

このプリシード設定は LXD デーモンを 192.0.2.1 のアドレスの 9999 ポートで HTTPS 接続をリッスンするようにし、15 時間ごとにイメージを自動的に更新し、`lxdbr0` という名前のネットワークブリッジを作成して IPv4 アドレスを自動的に割り当てるようにします。

既存の LXD 環境を再構成する

新しく LXD をインストールして設定する場合は、プリシードファイルを使って指定した設定を適用できます (YAML が適切なキーと値を含む限りは)。指定した設定と矛盾する既存の状態はありません。

しかし、既存の LXD 環境をプリシードファイルで再構成する場合は指定した YAML の設定が既存の設定と矛盾するかもしれません。このような矛盾を回避するために、以下のルールが実施されます。

- 指定された YAML 設定は既存のエンティティを上書きします。これは既存のエンティティを正構成する場合は、変更するキーだけではなくエンティティの設定全体を指定する必要があることを意味します。
- 指定した YAML 設定が存在しないエンティティを含む場合、それらは作成されます。

これは *REST API* の `PUT` リクエストと同様の挙動です。

ロールバック

新しい設定の一部が既存の状態と矛盾する (例えば、ストレージプールのドライバを `dir` から `zfs` に変更しようとするなど) 場合、プリシードを指定したコマンドは失敗し、それ以前に適用した全ての変更をロールバックしようと試みます。

例えば、新しい設定によって作成されたエンティティを削除し、上書きしたエンティティは元の状態に戻します。

ロールバックでエンティティを上書きするときに失敗した場合の挙動は *REST API* の `PUT` と同様です。

注釈: ロールバックの行程は稀ではありますが失敗する可能性があります (たいていはバックエンドのバグか制限によるものです)。そのため、LXD デーモンをプリシードで再構成する際は慎重に行うほうが良いです。

デフォルトプロファイル

対話式の初期化モードと異なり、`lxd init --preseed` コマンドは、指定した YAML ファイルで明示的に指定しない限りは、デフォルトプロファイルを変更しません。

インスタンスに対して、通常はルートディスクデバイスとネットワークインタフェースをデフォルトプロファイルにアタッチしたいでしょう。これの設定例は下記のセクションを参照してください。

設定形式

さまざまなエンティティのサポートされるキーと値は *REST API* のドキュメントに記載されているのと同じですが、利便性のため YAML 形式に変換されています。しかし、YAML は JSON のスーパーセットですので、JSON も使えます。

以下のスニペットは設定可能なほとんどの設定を含むプリシードファイルの例です。これをあなたのプリシードファイルのテンプレートとして使用し、必要な設定を追加、変更、削除して利用できます。

```
# デーモン設定
config:
  core.https_address: 192.0.2.1:9999
  core.trust_password: sekret
  images.auto_update_interval: 6

# ストレージプール
storage_pools:
- name: data
  driver: zfs
  config:
    source: my-zfs-pool/my-zfs-dataset

# ネットワークデバイス
networks:
- name: lxd-my-bridge
  type: bridge
  config:
    ipv4.address: auto
    ipv6.address: none

# プロファイル
profiles:
- name: default
```

(次のページに続く)

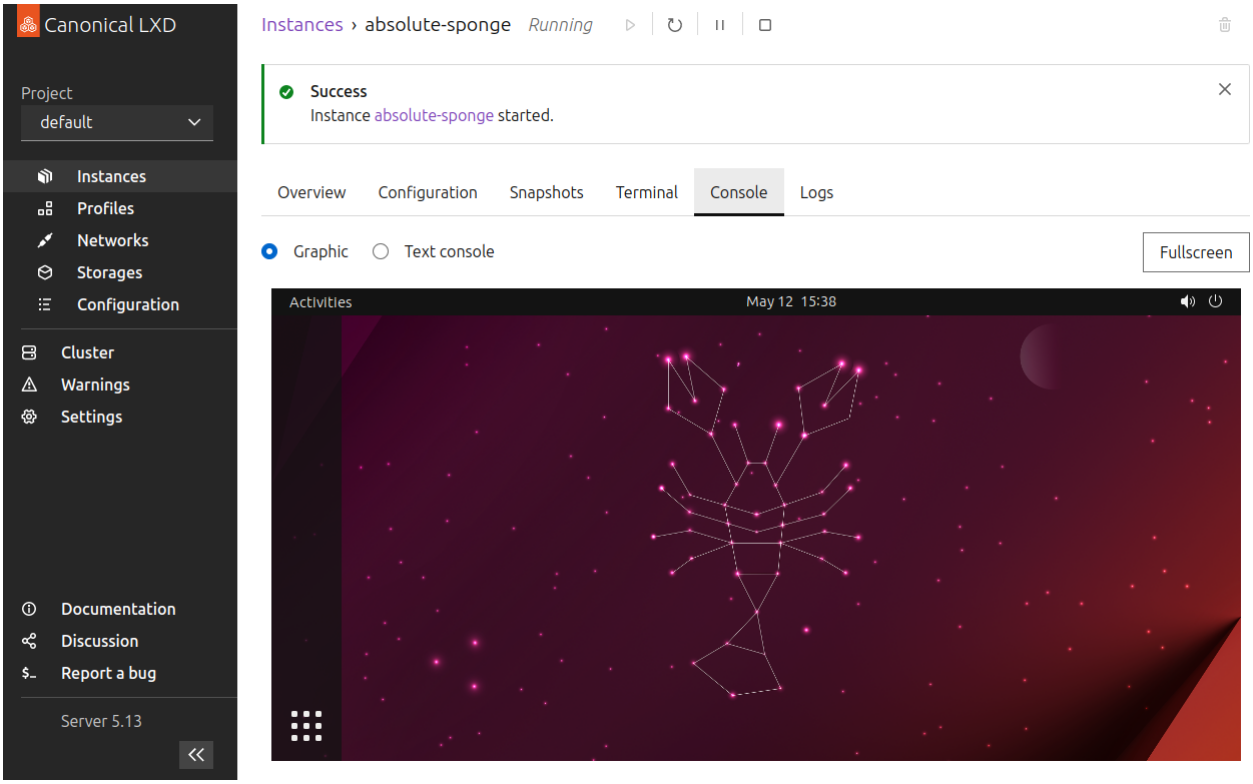
(前のページからの続き)

```
  devices:
    root:
      path: /
      pool: data
      type: disk
- name: test-profile
  description: "Test profile"
  config:
    limits.memory: 2GB
  devices:
    test0:
      name: test0
      nictype: bridged
      parent: lxd-my-bridge
      type: nic
```

3.1.4 LXD ウェブ UI へのアクセス方法

注釈: LXD 5.14 から、LXD ウェブ UI は LXD の snap パッケージに同梱され利用できるようになりました。

ソースコードは LXD-UI の GitHub レポジトリを参照してください。



LXD ウェブ UI は、LXD サーバーとインスタンスを管理するためのグラフィカルインターフェースを提供します。現在、初期段階にありまだ全機能を提供していませんが、最終的には LXD コマンドラインクライアントの代替となるでしょう。

以下の手順を完了して LXD ウェブ UI にアクセスします：

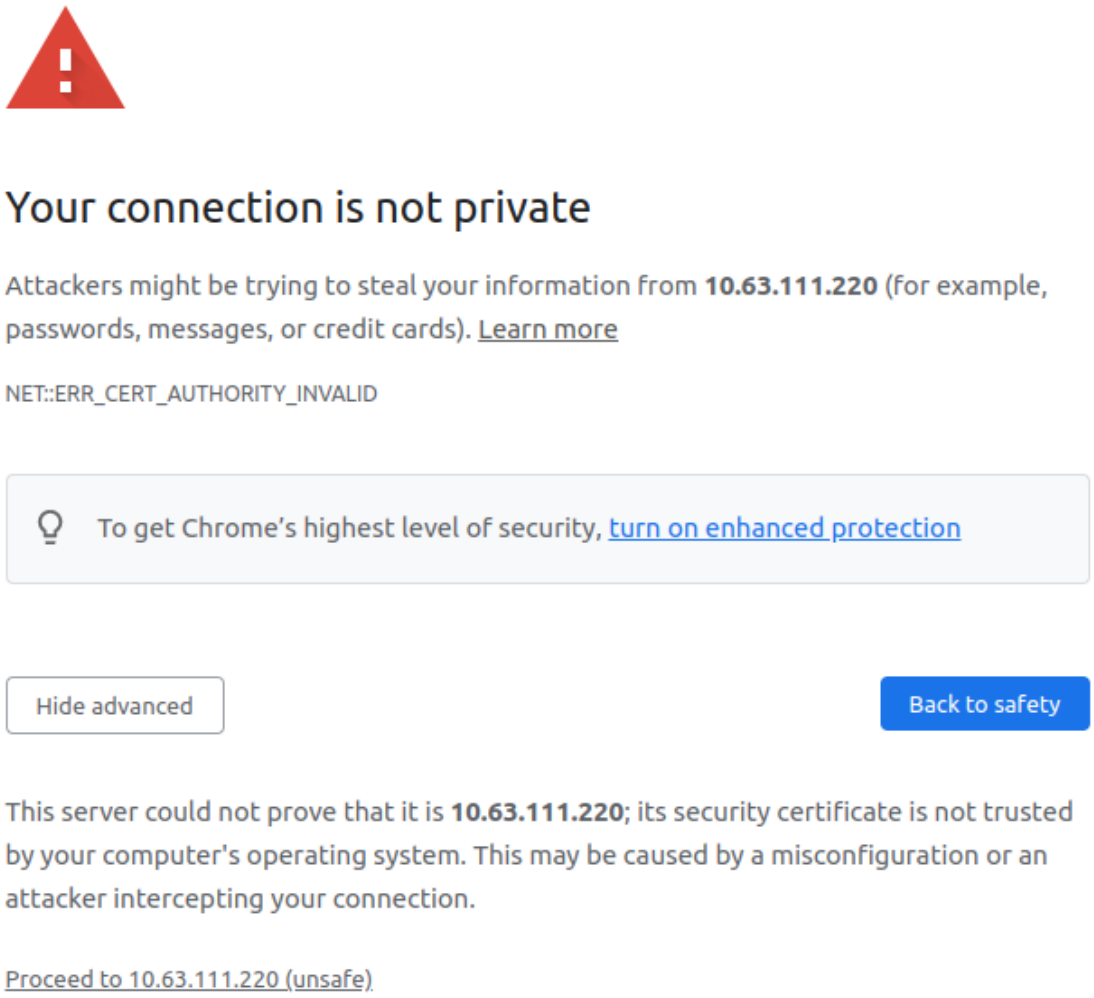
1. snap パッケージ内で UI を有効にします。

```
snap set lxd ui.enable=true
snap restart --reload lxd
```

2. LXD サーバーが**ネットワークに公開されている**ことを確認します。サーバーは**初期化**中に公開させることができるか、それ以降にサーバー設定オプションの *core.https_address* を設定することで公開させることができます。

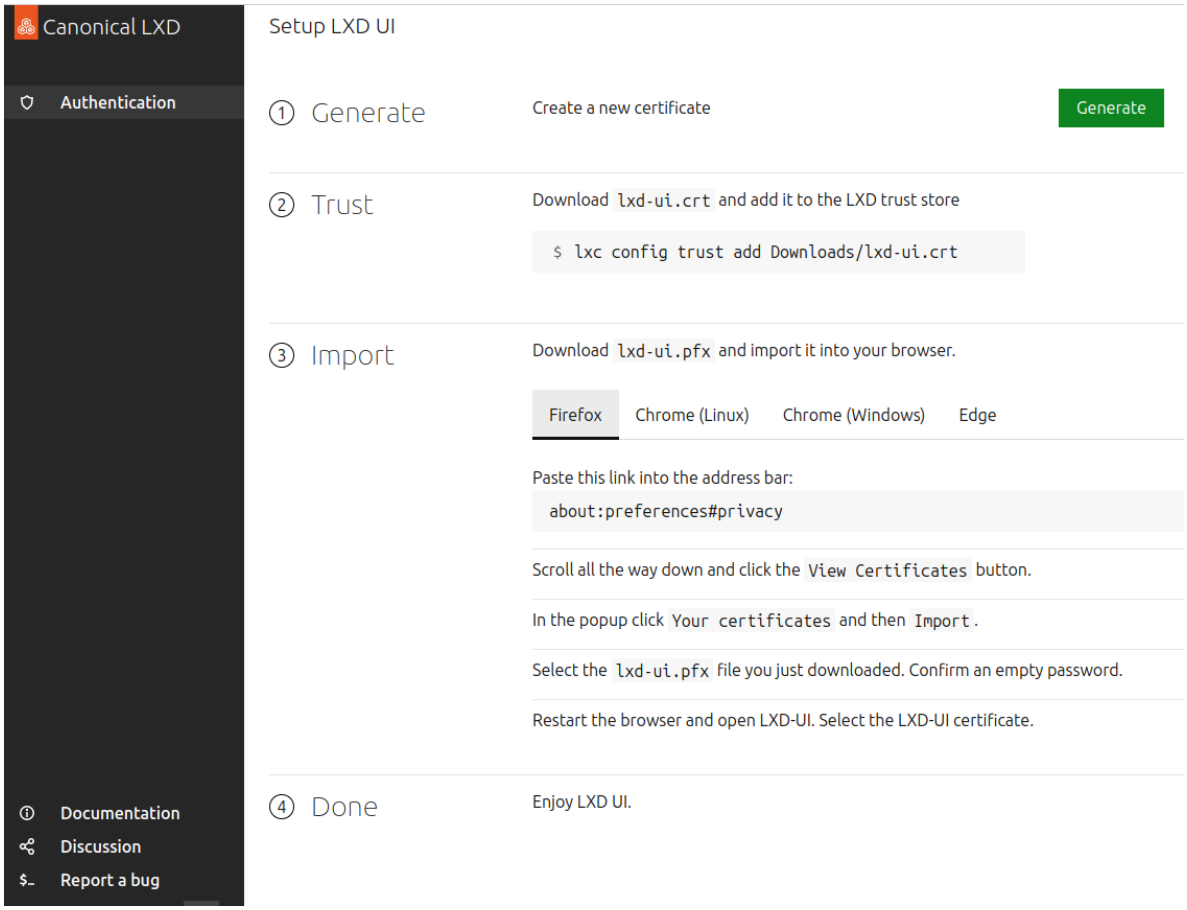
3. サーバーアドレス（例：`https://192.0.2.10:8443`）を入力して、ブラウザから UI にアクセスします。

   セキュアな *TLS* **サーバー証明書**を設定していない場合、LXD は自己署名証明書を使用し、ブラウザにセキュリティ警告が表示されます。ブラウザの機能を使用して、セキュリティ警告が出ても続行してください。



4. UI 内で提示される手順に従って、UI クライアントが LXD サーバーと認証するために必要な証明書を設定します。これらの手順には、証明書のセットの作成、プライベートキーのブラウザへの追加、公開キーのサーバーのトラストストアへの追加が含まれます。

   詳細については**リモート** *API* **認証**を参照してください。



証明書の設定が完了したら、インスタンスの作成、プロファイルの編集、またはサーバーの設定を開始できます。

3.1.5 よく聞かれる質問 (FAQ)

以下のセクションは、よくある質問への回答を提供します。それらは一般的な問題の解決方法を説明し、より詳細な情報へと導きます。

なぜ私のインスタンスはネットワークアクセスがないのですか？

最も可能性が高いのは、あなたのファイアウォールがインスタンスのネットワークアクセスをブロックしているためです。問題とその修正方法についての詳細は **ファイアウォールを設定するには** をご覧ください。

接続問題の別の一般的な原因は、LXD と Docker を同じホスト上で実行していることです。このような問題を修正する方法については *LXD と Docker* **の接続の問題を回避する** を参照してください。

LXD サーバーをリモートアクセス可能にするにはどうすればよいですか？

デフォルトでは、LXD サーバーはネットワークからアクセスできません。なぜなら、それはローカルの Unix ソケットでしかリッスンしていないからです。

リモートアクセスを可能にするためには、 *LXD* **をネットワークに公開するには** の指示に従ってください。

`lxc remote add` を行うと、パスワードまたはトークンを求められるのはなぜですか？

リモート API にアクセスするためには、クライアントは LXD サーバーに対して認証を行わなければなりません。リモートサーバーがどのように設定されているかにより、サーバーが発行したトラストトークンを提供するか、トラストパスワードを指定する必要があります（*core.trust_password* が設定されている場合）。

トラストトークンを使用して認証する方法については *LXD* **サーバーでの認証** を、他の認証方法については **リモート** *API* **認証** を参照してください。

私はなぜ特権コンテナを実行すべきではないのですか？

特権コンテナは、ホスト全体に影響を与えることができます - 例えば、`/sys` 内のものを使ってネットワークカードをリセットすると、ホスト全体のそれがリセットされ、ネットワークが一時的に断線します。詳細は **コンテナのセキュリティ** をご覧ください。

ほとんどのものは非特権コンテナで実行できます。また、NFS ファイルシステムをコンテナ内にマウントしたいなど、通常とは異なる特権を必要とするものの場合、バインドマウントを使用する必要があるかもしれません。

ホームディレクトリをコンテナにバインドマウントすることはできますか？

はい、それは**ディスクデバイス**を使用することで可能です：

```
lxc config device add container-name home disk source=/home/${USER} path=/home/ubuntu
```

非特権コンテナの場合、コンテナ内のユーザーが適切な読み書き権限を持っていることを確認する必要があります。そうでないと、すべてのファイルはオーバーフロー UID/GID（`65536:65536`）として表示され、ワールドリーダブルでないものへのアクセスは失敗します。必要な権限を付与するために以下の方法のいずれかを使用してください：

- `lxc config device add` 呼び出しに `shift=true` を渡します。これは、カーネルとファイルシステムが idmapped マウントまたは shiftfs をサポートしているかどうかに依存します（`lxc info` を参照）。
- `raw.idmap` エントリを追加します（**ユーザー名前空間の** *Idmaps* を参照）。
- ホームディレクトリに再帰的な POSIX ACL を配置します。

特権コンテナはこの問題を持っていません、なぜならコンテナ内のすべての UID/GID は外部と同じだからです。しかし、それが特権コンテナのセキュリティ問題のほとんどの原因でもあります。

LXD コンテナ内で Docker を実行する方法は？

LXD コンテナ内で Docker を実行するためには、コンテナの *security.nesting* プロパティを true に設定します：

```
lxc config set <container> security.nesting true
```

LXD コンテナはカーネルモジュールをロードできないため、Docker の設定によっては、ホストで追加のカーネルモジュールをロードする必要があるかもしれません。コンテナが必要とするカーネルモジュールのカンマ区切りのリストを設定することでこれを行うことができます：

```
lxc config set <container_name> linux.kernel_modules <modules>
```

さらに、コンテナ内に/.dockerenv ファイルを作成すると、Docker がネストした環境で実行されているために発生するいくつかのエラーを無視するのに役立ちます。

LXD クライアント（lxc）は設定をどこに保存しますか？

lxc コマンドはその設定を~/.config/lxc に保存します。Snap ユーザーの場合は~/snap/lxd/common/config に保存します。

様々な設定ファイルがそのディレクトリに保存されます。例えば：

- client.crt：クライアント証明書（要求に応じて生成されます）
- client.key：クライアントキー（要求に応じて生成されます）
- config.yml：設定ファイル（remotes、aliases などの情報）
- servercerts/：remotes に関連するサーバー証明書が保存されているディレクトリ

なぜ私は他のホストから LXD インスタンスに ping を送ることができないのですか？

多くのスイッチは MAC アドレスの変更を許可せず、不正な MAC を持つトラフィックをドロップするか、ポートを完全に無効にするかします。ホストから LXD インスタンスには ping を送ることができますが、異なるホストから ping を送ることができない場合、これが原因である可能性があります。

この問題を診断する方法は、アップリンク上で tcpdump を実行することで、ARP Who has `xx.xx.xx.xx` tell `yy.yy.yy.yy` が表示され、レスポンスを送信しているにもかかわらず確認されていない、または ICMP パケットが成功裏に送受信されているにもかかわらず、他のホストには受け取られていないことを確認することです。

LXD が何をしているかモニターするには?

LXD が何をしているかとどんなプロセスが稼働しているかについての詳細な情報を見るには、lxc monitor コマンドを使用します。

例えば、全てのタイプのメッセージの出力を人間が見やすい形式で表示するには、以下のコマンドを使用します。

```
lxc monitor --pretty
```

全てのオプションについては lxc monitor --help を、より詳しい情報は *LXD* **をデバッグするには**を参照してください。

インスタンス作成時に LXD が止まってしまうのはなぜですか？

ストレージプールの空きが無くなってないか (lxc storage info <pool_name>を実行して) 確認してください。空きが無い場合、LXD はイメージの解凍ができず、作成しようとしているインスタンスは止まったままに見えます。

何が起きているかをより詳しく調べるには lxc monitor を実行し (*LXD* **が何をしているかモニターするには***?*参照)、sudo dmesg で何か I/O エラーが起きていないか確認してください。

3.1.6 コントリビュート

プロジェクトに貢献する前に、以下のガイドラインを確認してください。

プルリクエスト

このプロジェクトへの変更は、Github でプルリクエストとして提案してください。https://github.com/lxc/lxd

そのあと、提案はコードレビューを経て承認され、メインブランチにマージされます。

コミット構成

コミットを次のように分類する必要があります:

- API 拡張 (doc/api-extensions.md と shared/version.api.go を含む変更に対して api: Add XYZ extension)
- ドキュメント (doc/ 内のファイルに対して doc: Update XYZ)
- API 構造 (shared/api/ の変更に対して shared/api: Add XYZ)
- Go クライアントパッケージ (client/ の変更に対して client: Add XYZ)
- CLI (lxc/ の変更に対して lxc/<command>: Change XYZ)

- スクリプト (scripts/ の変更に対して scripts: Update bash completion for XYZ)
- LXD デーモン (lxd/ の変更に対して lxd/<package>: Add support for XYZ)
- テスト (tests/ の変更に対して tests: Add test for XYZ)

同様のパターンが LXD コードツリーの他のツールにも適用されます。そして複雑さによっては、さらに小さな単位に分けられるかもしれません。

CLI ツール (lxc/) 内の文字列を更新する際は、テンプレートを更新してコミットする必要があるでしょう:

```
make i18n
git commit -a -s -m "i18n: Update translation templates" po/
```

このようにすることで、コントリビューションに対するレビューが容易になり、stable ブランチへバックポートするプロセスが大幅に簡素化されます。

ライセンスと著作権

デフォルトで、このプロジェクトに対するいかなる貢献も Apache 2.0 ライセンスの下で行われます。

変更の著者は、そのコードに対する著作権を保持します (著作権の譲渡はありません)。

開発者の起源の証明

このプロジェクトへの貢献の追跡を改善するために、DCO 1.1 を使用しており、ブランチに入るすべての変更に対して「サインオフ」手順を使用しています。

サインオフとは、あなたがそのコミットを書いたことを証明する、そのコミットの説明の最後にある単純な行です。この行は、自分が書いたものであることを証明したり、オープンソースとして渡す権利があることを証明したりします。

> Developer Certificate of Origin Version 1.1
>
> Copyright (C) 2004, 2006 The Linux Foundation and its contributors. 660 York Street, Suite 102, San Francisco, CA 94110 USA
>
> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
>
> Developer's Certificate of Origin 1.1
>
> By making a contribution to this project, I certify that:
>
> (a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or

(b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or

(c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.

(d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

有効なサインオフラインの例は以下の通りです。

```
Signed-off-by: Random J Developer <random@developer.org>
```

実名と有効な電子メールアドレスを使用してください。残念ながら、ペンネームや匿名での投稿はできません。

また、それぞれのコミットには作者が個別に署名する必要があります。大きなセットの一部であってもです。`git commit -s` が役に立つでしょう。

Code of Conduct

コントリビュートする際には、行動規範を遵守しなければなりません。行動規範は、以下のサイトから入手できます。 https://github.com/lxc/lxd/blob/master/CODE_OF_CONDUCT.md

開発を始める

開発環境をセットアップし LXD の新機能に取り組みを開始するには以下の手順に従ってください。

依存ライブラリーのビルド

依存ライブラリーをビルドするには *LXD* **をソースからインストールする** の手順に従ってください。

あなたの fork の remote を追加

依存ライブラリーをビルドし終わったら、 GitHub の fork を remote として追加できます。

```
git remote add myfork git@github.com:<your_username>/lxd.git
git remote update
```

次にこちらに切り替えます。

```
git checkout myfork/master
```

LXD のビルド

最後にレポジトリ内で make を実行すれば LXD のあなたの fork をビルドできます。

この時点であなたが最も行いたいであろうことはあなたの fork 上にあなたの変更のための新しいブランチを作ることです。

```
git checkout -b [name_of_your_new_branch]
git push myfork [name_of_your_new_branch]
```

LXD の新しいコントリビュータのための重要な注意事項

- 永続データは LXD_DIR ディレクトリに保管されます。これは lxd init で作成されます。 LXD_DIR のデフォルトは /var/lib/lxd か snap ユーザーは /var/snap/lxd/common/lxd です。
- 開発中はバージョン衝突を避けるため LXD のあなたの fork 用に LXD_DIR の値を変更すると良いでしょう。
- あなたのソースからコンパイルされる実行ファイルはデフォルトでは $(go env GOPATH)/bin に生成されます。
  - あなたの変更をテストするときはこれらの実行ファイル（インストール済みかもしれないグローバルの lxd ではなく）を明示的に起動する必要があります。
  - これらの実行ファイルを適切なオプションを指定してもっと便利に呼び出せるように ~/.bashrc にエイリアスを作るという選択も良いでしょう。
- 既存のインストール済み LXD のデーモンを実行するための systemd サービスが設定されている場合はバージョン衝突を避けるためにサービスを無効にすると良いでしょう。

3.1.7 サポート

以下のチャネルで情報を得たり、ユーザーサポートを依頼することができます。

サポートとコミュニティ

LXD コミュニティと交流するために以下のチャンネルが用意されています。

バグレポート

バグレポートや機能要求は以下の場所で受け付けています。https://github.com/lxc/lxd/issues/new

フォーラム

フォーラムは以下の場所にあります。https://discuss.linuxcontainers.org

メーリングリスト

開発者やユーザーの議論には LXC のメーリングリストを利用しています。メーリングリストは以下の場所にあります。https://lists.linuxcontainers.org

IRC

ライブの議論がお好みならば、irc.libera.chat の#lxc で私たちを見つけることができます。必要であれば Getting started with IRC を参照してください。

商用サポート

LXD の商用サポートは、Canonical Ltd を通じて受けることができます。

ドキュメント

公式ドキュメントは https://lxd-ja.readthedocs.io/ja/latest/ (原文は https://linuxcontainers.org/lxd/docs/latest/) で入手できます。

その他の資料は、website、YouTube、フォーラムの Tutorials section にあります。

3.2 LXD の管理

3.2.1 lxd と lxc について

LXD はよく LXC と混同されます。そして LXD が `lxd` コマンドと `lxc` コマンドを提供するという事実が混同の一因となっています。

LXD 対 LXC

LXD と LXC は Linux コンテナの 2 つの全く別の実装です。

LXC は Linux カーネルのコンテナ機能に対する低レベルのユーザー空間のインタフェースです。LXC はツール (`lxc-*`コマンド)、テンプレート、そしてライブラリと言語バインディングから構成されます。

LXD は Linux コンテナの取り扱いを簡単にする目的で作られている、より直観的でユーザーフレンドリなツールです。LXD は LXC のツールとディストリビューションのテンプレートシステムの代替となるものであり、ネットワーク上で制御可能なことにより実現される機能も追加されています。内部では、LXD はコンテナの作成と管理に LXC を使用しています。

LXD は LXC がサポートする機能のスーパーセットを提供し、より簡単に使えます。ですので、どちらのツールを使うべきかわからない場合は、LXD を選ぶのが良いでしょう。LXC は LXD をサポートしないディストリビューション上で Linux コンテナを稼働させたい上級ユーザーへの選択肢としてとらえるべきです。

LXD デーモン

LXD の中心的な部分は LXD デーモンです。それは持続的にバックグラウンドで稼働し、インスタンスを管理し、全てのリクエストを処理します。デーモンは REST API を提供し、直接アクセスすることもクライアント経由 (例えば、LXD に同梱のデフォルトのコマンドラインクライアント) でアクセスすることもできます。

LXD デーモンについての詳細な情報は**デーモンの動作**を参照してください。

`lxd` 対 `lxc`

LXD を制御するには、通常 `lxd` と `lxc` という 2 つの異なるコマンドを使います。

LXD **デーモン**
: `lxd` コマンドは LXD デーモンを制御します。通常デーモンは自動的に起動しますので、ほとんどの場合 `lxd` コマンドを使う必要はありません。例外は *LXD* **の初期化**のために `lxd init` サブコマンドを実行するときです。

  またデーモンのデバッグや管理のためのサブコマンドもあります。しかしこれは上級ユーザーのみを想定したものです。全ての利用可能なサブコマンドの概要は `lxd --help` を参照してください。

LXD クライアント

lxc コマンドは LXD のコマンドラインベースのクライアントで、LXD デーモンとやりとりするのに使えます。インスタンス、サーバー設定、そして LXD 内で作成する全てのエンティティを管理するために lxc コマンドを使用できます。全ての利用可能なサブコマンドの概要は lxc --help を参照してください。

lxc ツールは LXD デーモンとのやり取りに使用できる唯一のクライアントではありません。API、UI、あるいは独自の LXD クライアントを使用することもできます。

3.2.2 LXD データベースについて

サーバーの設定と状態を保管するために LXD は分散データーベースを使用します。これにより (例えば、LXC でそうされているように) 設定が各インスタンスのディレクトリ内に保管される場合より高速に問い合わせができます。

利点を理解するため、全てへのインスタンスに「どのインスタンスが br0 を使用しているか？」のような問い合わせをすることを考えてみましょう。データベースが無ければ、この問いに答えるためには、設定を読み込んで解釈し、どのネットワークデバイスが設定上に定義されているかの確認を、ひとつひとつのインスタンスに対して行う必要があります。データベースがあれば、この情報を得るためにデータベースにシンプルな問い合わせを実行できます。

Dqlite

LXD クラスタ内では、クラスタの全てのメンバーが同じデータベースの状態を共有する必要があります。このため、LXD は SQLIte の分散版である Dqlite を使用しています。Dqlite はレプリケーション、冗長性、外部のデータベースプロセスを必要としない自動フェールオーバーの機能を提供します。

LXD をクラスタではなく単一マシンとして使用する場合は、Dqlite のデータベースは実質的には通常の SQLite データベースのように振る舞います。

ファイルの保管場所

データベースは LXD のデータディレクトリ (snap の場合/var/snap/lxd/common/lxd/database/、それ以外は/var/lib/lxd/database/) の database サブディレクトリに保管されます。

LXD を新しいバージョンにアップグレードする際はデータベーススキーマの更新が必要かもしれません。この場合、LXD は自動的にデータベースのバックアップをしてから更新を実行します。詳細は *LXD* **をアップグレードする**を参照してください。

3.2.3 LXD サーバーを設定するには

LXD サーバーで利用可能な全て設定オプションについては**サーバー設定**を参照してください。

LXD サーバーがクラスタの一部の場合、一部のオプションはクラスタに適用され、また別のオプションはローカルサーバー、つまりクラスタメンバーにのみ適用されます。**サーバー設定**オプションの表で、クラスタに適用されるオプションは `global` スコープと表記され、ローカルサーバーのみに適用されるオプションは `local` スコープと表記されます。

サーバーオプションを設定する

以下のコマンドでサーバーオプションを設定できます。

```
lxc config set <key> <value>
```

例えば、ポート 8443 で LXD サーバーにリモートからのアクセスを許可するには、以下のコマンドを入力します。

```
lxc config set core.https_address :8443
```

クラスタ構成では、クラスタメンバーだけにサーバー設定を行うには`--target` フラグを追加してください。例えば、特定のクラスタメンバーでイメージの tarball を保管する場所を設定するには、以下のようなコマンドを入力してください。

```
lxc config set storage.images_volume my-pool/my-volume --target member02
```

サーバー設定を表示する

現在のサーバー設定を表示するには、以下のコマンドを入力します。

```
lxc config show
```

クラスタ構成では、クラスタメンバーだけにサーバー設定を行うには`--target` フラグを追加してください。

サーバー設定全体を編集する

サーバー設定全体を YAML ファイルとして編集するには、以下のコマンドを入力します。

```
lxc config edit
```

クラスタ構成では、クラスタメンバーだけにサーバー設定を行うには`--target` フラグを追加してください。

3.2.4 リモートサーバーを追加するには

リモートサーバーは LXD コマンドラインクライアント内の概念です。デフォルトでは、コマンドラインクライアントはローカルの LXD デーモンとやりとりしますが、他のサーバーやクラスタを追加できます。

リモートサーバーの用途の 1 つはローカルサーバーでインスタンスを作成するのに使えるイメージを配布することです。詳細は**リモートイメージサーバー**を参照してください。

完全な LXD サーバーをお使いのクライアントにリモートサーバーとして追加することもできます。この場合、ローカルのデーモンと同様にリモートサーバーとやりとりできます。例えば、リモートサーバー上のインスタンスを管理したりサーバー設定を更新できます。

認証

LXD サーバーをリモートサーバーとして追加できるようにするには、サーバーの API が公開されている必要があります。それはつまり、*core.https_address* サーバー設定オプションが設定されている必要があることを意味します。

サーバーを追加する際は、**リモート** *API* **認証**の方法で認証する必要があります。

詳細は *LXD* **をネットワークに公開するには**を参照してください。

追加されたリモートを一覧表示する

設定された全てのリモートサーバーを見るには、以下のコマンドを入力します。

```
lxc remote list
```

simple streams 形式を使用するリモートサーバーは純粋なイメージサーバーです。`lxd` 形式を使用するサーバーは LXD サーバーであり、イメージサーバーだけとして稼働しているか、通常の LXD サーバーとして稼働するのに加えて追加のイメージを提供しているかのどちらかです。詳細は**リモートサーバータイプ**を参照してください。

リモートの LXD サーバーを追加する

LXD サーバーをリモートして追加するには、以下のコマンドを入力します。

```
lxc remote add <remote_name> <IP|FQDN|URL> [flags]
```

認証方法によっては固有のフラグが必要です (例えば、Candid 認証では `lxc remote add <remote_name> <IP|FQDN|URL> --auth-type=candid` を使います)。詳細は *LXD* **サーバーでの認証**と**リモート** *API* **認証**を参照してください。

例えば、IP アドレスを指定してリモートを追加するには以下のコマンドを入力します。

```
lxc remote add my-remote 192.0.2.10
```

リモートサーバーのフィンガープリントを確認するプロンプトが表示され、リモートで使用している認証方法によってパスワードまたはトークンの入力を求められます。

デフォルトのリモートを選択する

LXD コマンドラインクライアントは local リモート、つまりローカルの LXD デーモン、に接続する用に初期設定されています。

別のリモートをデフォルトのリモートとして選択するには、以下のように入力します。

```
lxc remote switch <remote_name>
```

どのサーバーがデフォルトのリモートとして設定されているか確認するには、以下のように入力します。

```
lxc remote get-default
```

グローバルのリモートを設定する

グローバルなシステム毎の設定としてリモートを設定できます。これらのリモートは、設定を追加した LXD サーバーの全てのユーザーで利用できます。

ユーザーはこれらのシステムで設定されたリモートを (例えば lxc remote rename または lxc remote set-url を実行することで) オーバーライドできます。その結果、リモートと対応する証明書がユーザー設定にコピーされます。

グローバルリモートを設定するには、以下のいずれかのディレクトリに置かれた config.yml ファイルを編集します。

- (定義されていれば)LXD_GLOBAL_CONF で指定されるディレクトリ
- /var/snap/lxd/common/global-conf/ (snap をお使いの場合)
- /etc/lxd/ (snap 以外の場合)

リモートへの接続用の証明書は同じ場所の servercerts ディレクトリ (例えば、/etc/lxd/servercerts/) に保管する必要があります。証明書はリモート名に対応する (例えば、foo.crt) 必要があります。

以下の設定例を参照してください。

```
remotes:
  foo:
    addr: https://192.0.2.4:8443
```

(次のページに続く)

(前のページからの続き)

```
    auth_type: tls
    project: default
    protocol: lxd
    public: false
  bar:
    addr: https://192.0.2.5:8443
    auth_type: tls
    project: default
    protocol: lxd
    public: false
```

3.2.5 コマンドエイリアスを追加するには

LXD コマンドラインクライアントでは良く使うコマンドのエイリアスを追加できます。長いコマンドのショートカットとして、あるいは既存のコマンドに自動的にフラグを追加するために、エイリアスを使用できます。

コマンドエイリアスを管理するには、`lxc alias` コマンドを使用します。

例えば、インスタンスを削除する際に必ず確認を求めるようにするには `lxc delete` に常に `lxc delete -i` を実行するようにエイリアスを作成します。

```
lxc alias add delete "delete -i"
```

登録された全てののエイリアスを表示するには `lxc alias list` を実行します。全ての利用可能なサブコマンドを表示するには `lxc alias --help` を実行してください。

3.2.6 サーバー設定

LXD サーバーは key/value 設定オプションで設定できます。

key/value 設定は名前空間が分けられています。以下のオプションが利用可能です。

- **コア設定**
- *ACME* **設定**
- *Candid* **と** *RBAC* **設定**
- **クラスタ設定**
- **イメージ設定**
- *Loki* **設定**

- **その他設定**

設定オプションをどのように設定するかについての手順は *LXD* **サーバーを設定するには**を参照してください。

注釈: このページの表で `global` スコープと表記されたオプションは即時に全てのクラスタメンバーに適用されます。`local` スコープと表記されたオプションはメンバーごとに設定する必要があります。

コア設定

以下のサーバーオプションはコアデーモンの設定を制御します。

キー	型	スコープ	デフォルト値	説明
cluster.healing_threshol	integer	global	0	オフラインのクラスタメンバーを退避させるまでの秒数 (無効にするには 0 を設定)
core.bgp_address	string	local	-	BGP サーバーをバインドさせるアドレス (BGP)
core.bgp_asn	string	global	-	ローカルサーバーに使用する BGP の AS 番号 (Autonomous System Number)
core.bgp_routerid	string	local		この BGP サーバーのユニークな ID(IPv4 アドレス形式)
core.debug_address	string	local	-	pprof デバッグサーバーがバインドするアドレス (HTTP)
core.dns_address	string	local	-	権威 DNS サーバーをバインドするアドレス (DNS)
core.https_address	string	local	-	リモート API がバインドするアドレス (HTTPS)
core.https_allowed_cr	bool	global	-	Access-Control-Allow-Credentials HTTP ヘッダの値を true にするかどうか
core.https_allowed_he	string	global	-	Access-Control-Allow-Headers HTTP ヘッダの値
core.https_allowed_me	string	global	-	Access-Control-Allow-Methods HTTP ヘッダの値
core.https_allowed_or	string	global	-	Access-Control-Allow-Origin HTTP ヘッダの値
core.https_trusted_pr	string	global	-	プロキシの connection ヘッダーでクライアントのアドレスを渡す信頼するサーバーの IP アドレスのカンマ区切りリスト
core.metrics_address	string	global	-	メトリクスサーバーをバインドさせるアドレス (HTTPS)
core.metrics_authenti	bool	global	true	メトリクスエンドポイントの認証を強制するかどうか
core.proxy_https	string	global	-	HTTPS プロキシを使用する場合はその URL(未指定の場合は HTTPS_PROXY 環境変数を参照)
core.proxy_http	string	global	-	HTTP プロキシを使用する場合はその URL(未指定の場合は HTTP_PROXY 環境変数を参照)
core.proxy_ignore_hos	string	global	-	プロキシが不要なホスト (NO_PROXY と同様な形式、例えば 1.2.3.4,1.2.3.5 を指定。未指定の場合は NO_PROXY 環境変数を参照)
core.remote_token_exp	string	global	-	リモート追加トークンの有効期限 (デフォルトは有効期限なし)
core.shutdown_timeout	integer	global	5	LXD サーバーがシャットダウンを完了するまでに待つ時間を分で指定

ACME 設定

以下のサーバーオプションは *ACME* 設定を制御します。

キー	型	スコープ	デフォルト値	説明
acme.agree_tos	bool	global	false	ACME の利用規約に同意するか
acme.ca_url	string	global	https://acme-v02.api.letsencrypt.org/directory	ACME サービスのディレクトリリソースの URL
acme.domain	string	global	-	証明書を発行するドメイン
acme.email	string	global	-	アカウント登録に使用する email アドレス

Candid と RBAC 設定

以下のサーバーオプションは、*Candid* **ベースの認証**あるいは**役割ベースのアクセスコントロール** *(RBAC)* を使った外部のユーザ認証を設定します。

キー	型	スコープ	デフォルト値	説明
candid.api.key	string	global	-	Candid サーバーの公開鍵 (HTTP のみのサーバーで必要)
candid.api.url	string	global	-	Candid を使用する外部認証エンドポイントの URL
candid.domains	string	global	-	許可される Candid ドメインのカンマ区切りリスト (空文字は全てのドメインが有効という意味になります)
candid.expiry	integer	global	3600	Candid macaroon の有効期間 (秒で指定)
rbac.agent.private_key	string	global	-	RBAC 登録中に提供される Candid エージェントの秘密鍵
rbac.agent.public_key	string	global	-	RBAC 登録中に提供される Candid エージェントの公開鍵
rbac.agent.url	string	global	-	RBAC 登録中に提供される Candid エージェントの URL
rbac.agent.username	string	global	-	RBAC 登録中に提供される Candid エージェントのユーザー名
rbac.api.expiry	integer	global	-	RBAC の macaroon の有効期限 (秒)
rbac.api.key	string	global	-	RBAC サーバーの公開鍵 (HTTP のみ有効なサーバーで必要)
rbac.api.url	string	global	-	外部の RBAC サーバーの URL

OpenID Connect 設定

キー	型	スコープ	デフォルト値	説明
oidc.client.id	string	global	-	OpenID Connect クライアント ID
oidc.issuer	string	global	-	プロバイダの OpenID Connect Discovery URL
oidc.audience	string	global	-	アプリケーションに期待される audience value (プロバイダによっては必須)

クラスタ設定

以下のサーバーオプションは**クラスタリング**を制御します。

キー	型	スコープ	デフォルト値	説明
cluster.https_address	string	local	-	クラスタのトラフィックに使用するアドレス
cluster.images_minimal_r	integer	global	3	特定のイメージのコピーを持つべきクラスタメンバーの最小数 (リプリケーションなしは 1 を、全メンバーにコピーは-1 を設定)
cluster.join_token_expir	string	global	3H	クラスタジョイントークンの有効期限
cluster.max_standby	integer	global	2	データベースのスタンバイの役割を割り当てられるクラスタメンバーの最大数 (0 から 5 である必要あり)
cluster.max_voters	integer	global	3	データベースの投票者の役割を割り当てられるクラスタメンバーの最大数 (3 以上の奇数である必要あり)
cluster.offline_threshol	integer	global	20	無反応なノードをオフラインとみなす秒数

イメージ設定

以下のサーバーオプションは**イメージ**をどう取り扱うかを制御します。

キー	型	スコープ	デフォルト値	説明
images.auto_update_cached	bool	global	true	LXD がキャッシュしているイメージを自動的に更新するかどうか
images.auto_update_interval	integer	global	6	キャッシュされているイメージが更新されているかチェックする間隔を時間単位で指定
images.compression_algorith	string	global	gzip	新しいイメージに使用する圧縮アルゴリズム (bzip2, gzip, lzma, xz, none のいずれか)
images.default_architecture	string	-	-	アーキテクチャーが混在するクラスタ内で使用するデフォルトのアーキテクチャー
images.remote_cache_expiry	integer	global	10	キャッシュされたが未使用のイメージを破棄するまでの日数

Loki 設定

以下のサーバーオプションは外部ログ集約システムを設定します。

キー	型	スコープ	デフォルト値	説明
loki.api.ca_cert	string	global	-	Loki サーバーの CA 証明書
loki.api.url	string	global	-	Loki サーバーの URL
loki.auth.password	string	global	-	認証に使用するパスワード
loki.auth.username	string	global	-	認証に使用するユーザ名
loki.labels	string	global	-	Loki ログエントリにラベルとして使用する値のカンマ区切りリスト
loki.loglevel	string	global	info	Loki サーバーに送信する最低のログレベル
loki.types	string	global	lifecycle, logging	Loki サーバーに送信するイベント種別 (lifecytle および/または logging)

その他設定

以下のサーバーオプションは**インスタンス**のサーバー固有設定、MAAS 統合、*OVN* 統合、**バックアップ**、**ストレージ**を設定します。

キー	型	スコープ	デフォルト値	説明
backups.compression_al	string	globa	gzip	バックアップに用いる圧縮アルゴリズム (bzip2, gzip, lzma, xz, none のいずれか)
instances.nic.host_name	string	globa	random	random に設定するとランダムなホストインタフェース名を使用し、mac に設定すると lxd<mac_address>の形式 (先頭 2 桁を除いた MAC アドレス) で名前を生成
instances.placement.scriptlet	string	globa	-	カスタムの自動インスタンス配置ロジック用の**インスタンス配置スクリプトレット**を格納
maas.api.key	string	globa	-	MAAS を管理するための API キー
maas.api.url	string	globa	-	MAAS サーバーの URL
maas.machine	string	local	ホスト名	この LXD ホストの MAAS での名前
network.ovn.integration_br	string	globa	br-int	OVN ネットワークに使用する OVN 統合ブリッジ
network.ovn.northbound_cor	string	globa	unix:/var/run/ovn/ovnnb_db.sock	OVN northbound データベース接続文字列
storage.backups_volume	string	local	-	バックアップの tarball を保管するのに使用するボリューム (POOL/VOLUME 形式で指定)
storage.images_volume	string	local	-	イメージの tarball を保管するのに使用するボリューム (POOL/VOLUME 形式で指定)

3.2.7 アーキテクチャ

LXD は Linux カーネルと Go でサポートされるあらゆるアーキテクチャ上で稼働できます。

LXD の一部のエンティティ、例えば、インスタンス、インスタンススナップショット、イメージはアーキテクチャに依存します。

下記のテーブルはサポートされる全てのアーキテクチャを識別子と参照するための名前をリストアップします。アーキテクチャ名は通常は Linux のカーネルアーキテクチャ名と揃えてあります。

ID	名前	注釈	パーソナリティ
1	i686	32bit Intel x86	
2	x86_64	64bit Intel x86	x86
3	armv7l	32bit ARMv7 リトルエンディアン	
4	aarch64	64bit ARMv8 リトルエンディアン	armv7 (省略可能)
5	ppc	32bit PowerPC ビッグエンディアン	
6	ppc64	64bit PowerPC ビッグエンディアン	powerpc
7	ppc64le	64bit PowerPC リトルエンディアン	
8	s390x	64bit ESA/390 ビッグエンディアン	
9	mips	32bit MIPS	
10	mips64	64bit MIPS	mips
11	riscv32	32bit RISC-V リトルエンディアン	
12	riscv64	64bit RISC-V リトルエンディアン	

注釈: LXD はカーネルアーキテクチャのみに影響し、ツールチェインで決定される特定のユーザースペースのフレーバーには影響しません。

これは LXD は ARMv7 hard-float を ARMv7 soft-float と同じとして扱い、両方を armv7 として参照することを意味します。もしユーザーにとって有用であれば、正確なユーザースペースの ABI がイメージとコンテナプロパティとして設定でき、簡単に問い合わせできます。

3.3 セキュリティ

3.3.1 セキュリティについて

LXD のインストールが安全であることを保証するために、以下の点を考慮してください。

- オペレーティングシステムを最新に保ち、利用可能なすべてのセキュリティパッチをインストールする。
- サポートされている LXD のバージョン（LTS リリースまたは月例機能リリース）のみを使用する。

- LXD デーモンとリモート API へのアクセスを制限すること。
- 必要とされない限り、特権コンテナを使わないこと。特権的なコンテナを使う場合は、適切なセキュリティ対策をしてください。詳細は LXC セキュリティページを参照してください。
- ネットワークインターフェイスを安全に設定してください。

詳細な情報は以下のセクションを参照してください。

セキュリティ上の問題を発見した場合、その問題の報告方法については LXD のセキュリティポリシー (原文: LXD security policy) を参照してください。

サポートされているバージョン

サポートされていないバージョンの LXD は実運用環境では絶対に使用しないでください。

LXD には 2 種類のリリースがあります。

- 月次機能リリース
- LTS リリース

フィーチャーリリースでは、最新のものだけがサポートされ、通常はポイントリリースは行いません。次の月次リリースまでユーザーに待ってもらいます。

LTS リリースでは、定期的にバグフィックス・リリースを行います。これは、フィーチャー・リリースに含まれるバグフィックスを集積したもので、新機能は含まれません。

LXD デーモンへのアクセス

LXD は Unix ソケットを介してローカルにアクセスできるデーモンで、設定されていれば TLS (Transport Layer Security) ソケットを介してリモートにアクセスすることもできます。ソケットにアクセスできる人は、ホストデバイスやファイルシステムをアタッチしたり、すべてのインスタンスのセキュリティ機能をいじったりするなど、LXD を完全に制御することができます。

したがって、デーモンへのアクセスを信頼できるユーザに制限するようにしてください。

LXD デーモンへのローカルアクセス

LXD デーモンは root で動作し、ローカル通信用の Unix ソケットを提供します。LXD のアクセス制御は、グループメンバーシップに基づいて行われます。root ユーザーと `lxd` グループのすべてのメンバーがローカルデーモンと対話できます。

重要: UNIX ソケットを介した LXD へのローカルアクセスは、常に LXD へのフルアクセスを許可します。これは、任意のインスタンス上のセキュリティ機能を変更できる能力に加えて、任意のインスタンスにファイルシステ

ムパスやデバイスをアタッチする能力を含みます。

したがって、あなたのシステムへのルートアクセスを信頼できるユーザーにのみ、このようなアクセスを与えるべきです。

リモート API へのアクセス

デフォルトでは、デーモンへのアクセスはローカルでのみ可能です。`core.https_address` という設定オプションを設定することで、同じ API を TLS ソケットでネットワーク上に公開することができます。手順は *LXD* **をネットワークに公開するには** を参照してください。リモートクライアントは、LXD に接続して、公開用にマークされたイメージにアクセスできます。

リモートクライアントが API にアクセスできるように、信頼できるクライアントとして認証する方法がいくつかあります。詳細は**リモート** *API* **認証**を参照してください。

本番環境では、`core.https_address` に、(ホスト上の任意のアドレスではなく) サーバーが利用可能な単一のアドレスを設定する必要があります。さらに、許可されたホスト/サブネットからのみ LXD ポートへのアクセスを許可するファイアウォールルールを設定する必要があります。

コンテナのセキュリティ

LXD コンテナはセキュリティのために幅広い機能を使うことができます。

デフォルトでは、コンテナは **非特権** (*unprivileged*) であり、ユーザーネームスペース内で動作することを意味し、コンテナ内のユーザーの能力を、コンテナが所有するデバイスに対する制限された権限を持つホスト上の通常のユーザーに制限します。

コンテナ間のデータ共有が必要ない場合は、`security.idmap.isolated`(**セキュリティポリシー**参照) を有効にすることで、各コンテナに対して重複しない UID/GID マップを使用し、他のコンテナに対する潜在的な DoS (サービス拒否) 攻撃を防ぐことができます。

LXD はまた、**特権** (*privileged*) コンテナを実行することができます。しかし、これは (訳注: コンテナ内だけで) 安全に root 権限を使えるわけではなく、そのようなコンテナの中でルートアクセスを持つユーザは、閉じ込められた状態から逃れる方法を見つけるだけでなく、ホストを DoS することができてしまう点に注意してください。

コンテナのセキュリティと私たちが使っているカーネルの機能についてのより詳細な情報は LXC セキュリティページにあります。

コンテナ名の漏洩

デフォルトの設定ではシステム上の全ての cgroup と、さらに転じて、全ての実行中のコンテナを一覧表示することが簡単にできてしまいます。

コンテナを開始する前に /sys/kernel/slab と /proc/sched_debug へのアクセスをブロックすることでコンテナ名の漏洩を防げます。このためには以下のコマンドを実行してください。

```
chmod 400 /proc/sched_debug
chmod 700 /sys/kernel/slab/
```

ネットワークセキュリティ

ネットワークインタフェースは必ず安全に設定してください。どのような点を考慮すべきかは、使用するネットワークモードによって異なります。

ブリッジ型 NIC のセキュリティ

LXD のデフォルトのネットワークモードは、各インスタンスが接続する「管理された」プライベートネットワークのブリッジを提供することです。このモードでは、ホスト上に lxdbr0 というインタフェースがあり、それがインスタンスのブリッジとして機能します。

ホストは、管理されたブリッジごとに dnsmasq のインスタンスを実行し、IP アドレスの割り当てと、権威 DNS および再帰 DNS サービスの提供を担当します。

DHCPv4 を使用しているインスタンスには、IPv4 アドレスが割り当てられ、インスタンス名の DNS レコードが作成されます。これにより、インスタンスが DHCP リクエストに偽のホスト名情報を提供して、DNS レコードを偽装することができなくなります。

dnsmasq サービスは、IPv6 のルータ広告機能も提供します。つまり、インスタンスは SLAAC を使って自分の IPv6 アドレスを自動設定するので、dnsmasq による割り当ては行われません。しかし、DHCPv4 を使用しているインスタンスは、SLAAC IPv6 アドレスに相当する AAAA の DNS レコードも取得します。これは、インスタンスが IPv6 アドレスを生成する際に、IPv6 プライバシー拡張を使用していないことを前提としています。

このデフォルト構成では、DNS 名を偽装することはできませんが、インスタンスはイーサネットブリッジに接続されており、希望するレイヤー 2 トラフィックを送信することができます。これは、信頼されていないインスタンスがブリッジ上で MAC または IP の偽装を効果的に行うことができることを意味します。

デフォルトの設定では、ブリッジに接続されたインスタンスがブリッジに (潜在的に悪意のある)IPv6 ルータ広告を送信することで、LXD ホストの IPv6 ルーティングテーブルを修正することも可能です。これは、lxdbr0 インターフェイスが/proc/sys/net/ipv6/conf/lxdbr0/accept_ra を 2 に設定して作成されているためで、forwarding が有効であるにもかかわらず、LXD ホストがルーター広告を受け入れることを意味しています (詳細は/proc/sys/net/ipv4/* Variables を参照してください)。

しかし、LXD はいくつかのブリッジ型 NIC (Network interface controller) セキュリティ機能を提供しており、インスタンスがネットワーク上に送信することを許可されるトラフィックの種類を制御するために使用することができます。これらの NIC 設定は、インスタンスが使用しているプロファイルに追加する必要がありますが、以下のように個々のインスタンスに追加することもできます。

ブリッジ型 NIC には、以下のようなセキュリティ機能があります。

キー	タイプ	デフォルト	必須	説明
security.mac_filtering	bool	false	no	インスタンスが他のインスタンスの MAC アドレスを詐称することを防ぐ。
security.ipv4_filtering	bool	false	no	インスタンスが他のインスタンスの IPv4 アドレスになりすますことを防ぎます (mac_filtering を有効にします)。
security.ipv6_filtering	bool	false	no	インスタンスが他のインスタンスの IPv6 アドレスになりすますことを防ぎます (mac_filtering を有効にします)。

プロファイルで設定されたデフォルトのブリッジ型 NIC の設定は、インスタンスごとに以下の方法で上書きすることができます。

```
lxc config device override <instance> <NIC> security.mac_filtering=true
```

これらの機能を併用することで、ブリッジに接続されているインスタンスが MAC アドレスや IP アドレスを詐称することを防ぐことができます。これらのオプションは、ホスト上で利用可能なものに応じて、xtables(iptables、ip6tables、ebtables) または nftables を使用して実装されます。

これらのオプションは、ネストされたコンテナが異なる MAC アドレスを持つ親ネットワークを使用すること (ブリッジされた NIC や macvlan NIC を使用すること) を効果的に防止することができるのは注目に値します。

IP フィルタリング機能は、スプーフィングされた IP を含む ARP および NDP アドバタイジングをブロックし、スプーフィングされたソースアドレスを含むすべてのパケットをブロックします。

security.ipv4_filtering または security.ipv6_filtering が有効で、(ipvX.address=none またはブリッジで DHCP サービスが有効になっていないため) インスタンスに IP アドレスが割り当てられない場合、そのプロトコルのすべての IP トラフィックがインスタンスからブロックされます。

security.ipv6_filtering が有効な場合、IPv6 のルータ広告がインスタンスからブロックされます。

security.ipv4_filtering または security.ipv6_filtering が有効な場合、ARP、IPv4 または IPv6 ではないイーサネットフレームはすべてドロップされます。これにより、スタックされた VLAN Q-in-Q (802.1ad) フレームが IP フィルタリングをバイパスすることを防ぎます。

ルート化された NIC のセキュリティ

「ルーテッド」と呼ばれる別のネットワークモードがあります。このモードでは、コンテナとホストの間に仮想イーサネットデバイペアを提供します。このネットワークモードでは、LXD ホストがルータとして機能し、コンテナの IP 宛のトラフィックをコンテナの `veth` インターフェイスに誘導するスタティックルートがホストに追加されます。

デフォルトでは、コンテナからのルータ広告が LXD ホスト上の IPv6 ルーティングテーブルを変更するのを防ぐために、ホスト上に作成された `veth` インタフェースは、その `accept_ra` 設定が無効になっています。それに加えて、コンテナが持っていることをホストが知らない IP に対するソースアドレスの偽装を防ぐために、ホスト上の `rp_filter` が 1 に設定されています。

3.3.2 リモート API 認証

LXD デーモンとのリモート通信は、HTTPS 上の JSON を使って行われます。

リモート API にアクセスするためには、クライアントは LXD サーバーとの間で認証を行う必要があります。以下の認証方法がサポートされています。

- *TLS* **クライアント証明書**
- *OpenID Connect* **認証**
- *Candid* **ベースの認証**
- **役割ベースのアクセスコントロール** *(RBAC)*

TLS クライアント証明書

認証に TLS クライアント証明書を使用する場合、クライアントとサーバーの両方が最初に起動したときにキーペアを生成します。サーバーはそのキーペアを LXD ソケットへの全ての HTTPS 接続に使用します。クライアントは、その証明書をクライアント証明書として、あらゆるクライアント・サーバー間の通信に使用します。

証明書を再生成させるには、単に古いものを削除します。次の接続時には、新しい証明書が生成されます。

通信プロトコル

通信プロトコルは TLS1.3 以上に対応しています。すべての通信には完全な前方秘匿を使用し、暗号は強力な楕円曲線（ECDHE-RSA や ECDHE-ECDSA など）に限定してください。

`LXD_INSECURE_TLS` 環境変数をクライアントとサーバーの両方で設定することにより LXD が TLS 1.2 を受け入れるようにすることはできます。しかし、これはサポートされる構成ではなく、時代遅れな企業プロキシを使うために強制される場合にのみ使用すべきです。

生成される鍵は最低でも 4096 ビットの RSA、できれば 384 ビットの ECDSA が望ましいです。署名を使用する場合は、SHA-2 署名のみを信頼すべきです。

我々はクライアントとサーバーの両方を管理しているので、壊れたプロトコルや暗号の下位互換をサポートする理由はありません。

信頼できる TLS クライアント

LXD サーバーが信頼する TLS 証明書のリストは、`lxc config trust list` で取得できます。

信頼できるクライアントは以下のいずれかの方法で追加できます。

- **信頼できる証明書をサーバーに追加する**
- **トラストパスワードを使ったクライアント証明書の追加**
- **トークンを使ったクライアント証明書の追加**

サーバーとの認証を行うワークフローは、SSH の場合と同様で、未知のサーバーへの初回接続時にプロンプトが表示されます。

1. ユーザーが `lxc remote add` でサーバーを追加すると、HTTPS でサーバーに接続され、その証明書がダウンロードされ、フィンガープリントがユーザーに表示されます。
2. ユーザーは、これが本当にサーバーのフィンガープリントであることを確認するよう求められます。これは、サーバーに接続して手動で確認するか、サーバーにアクセスできる人に info コマンドを実行してフィンガープリントを比較してもらうことで確認できます。
3. サーバーはクライアントの認証を試みます。
   - クライアント証明書がサーバーのトラストストアにある場合は、接続が許可されます。
   - クライアント証明書がサーバーのトラストストアにない場合、サーバーはユーザーにトークンまたはトラストパスワードの入力を求めます。提供されたトークンまたはトラストパスワードが一致した場合、クライアント証明書はサーバーのトラストストアに追加され、接続が許可されます。そうでない場合は、接続が拒否されます。

クライアントへの信頼を取り消すには、`lxc config trust remove FINGERPRINT` でそのクライアント証明書をサーバーから削除します。

TLS クライアントを 1 つまたは複数のプロジェクトに制限することが可能です。この場合、クライアントは、グローバルな構成変更の実行や、アクセスを許可されたプロジェクトの構成（制限、制約）の変更もできなくなります。

アクセスを制限するには、`lxc config trust edit FINGERPRINT` を使用します。`restricted` キーを `true` に設定し、クライアントのアクセスを制限するプロジェクトのリストを指定します。プロジェクトのリストが空の場合、クライアントはどのプロジェクトへのアクセスも許可されません。

信頼できる証明書をサーバーに追加する

信頼できるクライアントを追加するには、そのクライアント証明書をサーバーのトラストストアに直接追加するのが望ましい方法です。これを行うには、クライアント証明書をサーバーにコピーし、`lxc config trust add <file>`で登録します。

トラストパスワードを使ったクライアント証明書の追加

クライアント側から新しい信頼関係を確立できるようにするには、サーバーにトラストパスワード (*core.trust_password*) を設定する必要があります。クライアントは、プロンプト時にトラストパスワードを入力することで、自分の証明書をサーバーのトラストストアに追加することができます。

本番環境では、すべてのクライアントが追加された後に、`core.trust_password` の設定を解除してください。これにより、パスワードを推測しようとするブルートフォース攻撃を防ぐことができます。

トークンを使ったクライアント証明書の追加

トークンを使って新しいクライアントを追加することもできます。トークンは調整可能な時間 (*core.remote_token_expiry*) を過ぎるか一度使用すると無効になるため、これはトラストパスワードを使用するよりも安全な方法です。

この方法を使用するには，クライアント名の入力を促す `lxc config trust add` を呼び出して，各クライアント用のトークンを生成します。その後，クライアントは，トラストパスワードの入力を求められたときに生成されたトークンを提供することで，自分の証明書をサーバーのトラストストアに追加することができます。

注釈: LXD サーバーが NAT の後ろ側にいる場合、クライアント用のリモートを追加する際には外部のパブリックアドレスを指定する必要があります。

```
lxc remote add <name> <IP_address>
```

admin パスワードのプロンプトが表示されたら、生成されたトークンを入力してください。

サーバーでトークンを生成する際、 LXD はクライアントがサーバーにアクセスするために使える IP アドレスのリストを含めます。しかし、サーバーが NAT の後ろ側にいる場合、これらのアドレスはクライアントが接続できないローカルアドレスの場合があります。その場合、手動で外部アドレスを指定する必要があります。

あるいは、クライアントはリモートの追加時にトークンを直接提供することもできます。`lxc remote add <name> <token>`.

PKI システムの使用

PKI (Public key infrastructure) の設定では、システム管理者が中央の PKI を管理し、すべての LXD クライアント用のクライアント証明書とすべての LXD デーモン用のサーバー証明書を発行します。

PKI モードを有効にするには、以下の手順を実行します。

1. すべてのマシンに CA（認証局）の証明書を追加します。
   - クライアントの設定ディレクトリ (`~/.config/lxc` または Snap ユーザーの場合は`~/snap/lxd/common/config`) に `client.ca` ファイルを配置する。
   - `server.ca` ファイルをサーバーの設定ディレクトリ (`/var/lib/lxd` または Snap ユーザーの場合は`/var/snap/lxd/common/lxd`) に置く。
2. CA から発行された証明書をクライアントとサーバーに配置し、自動生成された証明書を置き換える。
3. サーバーを再起動します。

このモードでは、LXD デーモンへの接続はすべて、事前に発行された CA 証明書を使って行われます。

もしサーバー証明書が CA によって署名されていなければ、接続は単に通常の認証メカニズムを通過します。サーバー証明書が有効で CA によって署名されていれば、ユーザに証明書を求めるプロンプトを出さずに接続を続行します。

生成された証明書は自動的には信頼されないことに注意してください。そのため、**信頼できる *TLS* クライアント**で説明している方法のいずれかで、サーバーに追加する必要があります。

OpenID Connect 認証

LXD は OpenID Connect を使用して、OIDC（OpenID Connect）アイデンティティプロバイダーを通じてユーザーを認証することをサポートしています。

注釈: OpenID Connect 認証は現在開発中です。LXD 5.13 から、OpenID Connect を通じた認証がサポートされていますが、まだユーザーロールの取り扱いはありません。設定された OIDC アイデンティティプロバイダーを通じて認証するすべてのユーザーは、LXD へのフルアクセスを得ます。

LXD を OIDC 認証を使用するように設定するには、*`oidc.*`*サーバー設定オプションを設定します。

OIDC 認証で設定された LXD サーバーを指すリモートを追加するには、`lxc remote add <remote_name> <remote_address>`を実行します。その後、ウェブブラウザで認証を求められ、LXD が使用するデバイスコードを確認する必要があります。LXD クライアントはその後、アクセストークンとリフレッシュトークンを取得し保存し、それらを LXD とのすべてのやりとりに使用します。

Candid ベースの認証

*candid.** サーバーオプションにより Candid 認証を使うように LXD を設定できます。この場合、サーバーで認証を行おうとするクライアントは、*candid.api.url* で指定された認証サーバーからディスチャージトークンを取得しなければなりません。

認証サーバーの証明書は、LXD サーバーから信頼されていなければなりません。

Candid/Macaroon 認証を設定した LXD サーバーにリモートポインティングを追加するには、`lxc remote add REMOTE ENDPOINT --auth-type=candid` を実行します。ユーザーを確認するために、クライアントは認証サーバーが要求する認証情報の入力を求められます。認証が成功した場合、クライアントは LXD サーバーに接続し、認証サーバーから受け取ったトークンを提示します。LXD サーバーはトークンを検証し、リクエストを認証します。トークンはクッキーとして保存され、クライアントが LXD にリクエストするたびに提示されます。

Candid ベースの認証を設定する方法については、チュートリアルの Candid authentication for LXD を参照してください。

役割ベースのアクセスコントロール (RBAC)

LXD は Canonical の RBAC サービスとの連携をサポートしています。RBAC は Ubuntu Pro サブスクリプションに含まれています。Candid ベースの認証と組み合わせることで、RBAC (Role Based Access Control) は、API クライアントが LXD 上でできることを制限するために使うことができます。

このような設定では、認証は Candid を通して行われ、RBAC サービスはユーザー/グループの関係に役割を維持します。ロールは個々のプロジェクトにも、すべてのプロジェクトにも、あるいは LXD インスタンス全体にも割り当てることができます。

プロジェクトに適用された場合のロールの意味は以下の通りです。

- 監査役: プロジェクトへの読み取り専用のアクセス権
- ユーザー: 通常のライフサイクルアクション（開始、停止、...）を実行する能力。インスタンスでのコマンド実行、コンソールへのアタッチ、スナップショットの管理など。
- オペレーター: 上記のすべての機能に加え、インスタンスやイメージの作成、再設定、削除を行う機能インスタンスとイメージの作成、再設定、削除
- 管理者: 上記の機能に加えて、プロジェクト自体を再構成する機能を持つ

重要: 制限のないプロジェクトでは、`auditor` と `user` のロールだけが、ホストへのルートアクセスを任せられないユーザーに適しています。

また、**制限付きプロジェクト** では、適切に設定されていれば、`operator` ロールも安全に使用することができます。

LXD サーバーで RBAC を有効にするには *rbac.** サーバーオプションを設定してください。これは candid.* オプションのスーパーセットで、LXD を RBAC サービスに統合できます。

TLS サーバー証明書

LXD は ACME (Automatic Certificate Management Environment) サービス (例えば Let's Encrypt) を使ったサーバー証明書の発行をサポートします。

この機能を有効にするには以下の**サーバー設定**をしてください。

- acme.domain: 証明書を発行するドメイン。
- acme.email: ACME サービスのアカウントに使用する email アドレス。
- acme.agree_tos: ACME サービスの利用規約に同意するためには true に設定する必要あり。
- acme.ca_url: ACME サービスのディレクトリ URL。デフォルトでは LXD は "Let's Encrypt" を使用。

この機能を利用するには、 LXD は 80 番ポートを開放する必要があります。これは HAProxy のようなリバースプロキシを使用することで実現できます。

以下は lxd.example.net をドメインとして使用する HAProxy の最小限の設定です。証明書が発行された後、LXD は https://lxd.example.net/ でアクセスできます。

```
# Global configuration
global
  log /dev/log local0
  chroot /var/lib/haproxy
  stats socket /run/haproxy/admin.sock mode 660 level admin
  stats timeout 30s
  user haproxy
  group haproxy
  daemon
  ssl-default-bind-options ssl-min-ver TLSv1.2
  tune.ssl.default-dh-param 2048
  maxconn 100000

# Default settings
defaults
  mode tcp
  timeout connect 5s
  timeout client 30s
  timeout client-fin 30s
  timeout server 120s
```

(次のページに続く)

(前のページからの続き)

```
  timeout tunnel 6h
  timeout http-request 5s
  maxconn 80000

# Default backend - Return HTTP 301 (TLS upgrade)
backend http-301
  mode http
  redirect scheme https code 301

# Default backend - Return HTTP 403
backend http-403
  mode http
  http-request deny deny_status 403

# HTTP dispatcher
frontend http-dispatcher
  bind :80
  mode http

  # Backend selection
  tcp-request inspect-delay 5s

  # Dispatch
  default_backend http-403
  use_backend http-301 if { hdr(host) -i lxd.example.net }

# SNI dispatcher
frontend sni-dispatcher
  bind :443
  mode tcp

  # Backend selection
  tcp-request inspect-delay 5s

  # require TLS
  tcp-request content reject unless { req.ssl_hello_type 1 }

  # Dispatch
  default_backend http-403
```

(次のページに続く)

(前のページからの続き)

```
  use_backend lxd-nodes if { req.ssl_sni -i lxd.example.net }

# LXD nodes
backend lxd-nodes
  mode tcp

  option tcp-check

  # Multiple servers should be listed when running a cluster
  server lxd-node01 1.2.3.4:8443 check
  server lxd-node02 1.2.3.5:8443 check
  server lxd-node03 1.2.3.6:8443 check
```

失敗のシナリオ

以下のシナリオでは認証は失敗します。

サーバー証明書が変更された場合

サーバー証明書は以下の場合に変更されるかも知れません。

- サーバーが完全に再インストールされたため新しい証明書に変わった。
- 接続がインターセプトされた (MITM (Machine in the middle))。

このような場合、このリモートの設定内のフィンガープリントと証明書のフィンガープリントが一致しないため、クライアントはサーバーへの接続を拒否します。

この場合サーバー管理者に連絡して証明書が実際に変更されたのかを確認するのはユーザ次第です。実際に変更されたのであれば、証明書を新しいものに置き換えるか、リモートを削除して追加し直すことができます。

サーバーとの信頼関係が取り消された場合

別の信頼されたクライアントまたはローカルのサーバー管理者がサーバー上で対象のクライアントの信頼エントリを削除した場合、そのクライアントに対するサーバーの信頼関係は取り消されます。

この場合、サーバーは引き続き同じ証明書を使用していますが、全ての API 呼び出しは対象のクライアントが信頼されていないことを示す 403 のステータスコードを返します。

3.3.3 LXD をネットワークに公開するには

デフォルトでは、LXD は Unix ソケットを介してローカルユーザーからのみ使用でき、ネットワーク経由でアクセスすることはできません。

LXD をネットワークに公開するには、ローカル Unix ソケット以外のアドレスをリッスンするように設定する必要があります。これを行うには、*core.https_address* サーバー設定オプションを設定します。

例えば、LXD サーバーをポート 8443 でアクセスできるようにするには、以下のコマンドを入力します。

```
lxc config set core.https_address :8443
```

特定の IP アドレスからのアクセスを許可するには、ip addr を使用して利用可能なアドレスを見つけ、それを設定します。例えば：

user@host:~$ ip addr 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo valid_lft forever preferred_lft forever inet6 ::1/ 128 scope host valid_lft forever preferred_lft forever2: enp5s0: <BROADCAST, MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000 link/ether 00:16:3e:e3:f3:3f brd ff:ff:ff:ff:ff:ff inet 10.68.216.12/24 metric 100 brd 10.68. 216.255 scope global dynamic enp5s0 valid_lft 3028sec preferred_lft 3028sec inet6 fd42:e819:7a51:5a7b:216:3eff:fee3:f33f/64 scope global mngtmpaddr noprefixroute valid_lft forever preferred_lft forever inet6 fe80::216:3eff:fee3:f33f/64 scope link valid_lft forever preferred_lft forever3: lxdbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000 link/ether 00:16:3e:8d:f3:72 brd ff:ff:ff:ff:ff:ff inet 10.64.82.1/24 scope global lxdbr0 valid_lft forever preferred_lft forever inet6 fd42:f4ab:4399:e6eb::1/64 scope global valid_lft forever preferred_lft forever user@host:~$ lxc config set core.https_address 10.68.216.12 全てのリモートクライアントは LXD に接続して公開利用とマークされた任意のイメージにアクセスできます。

LXD サーバーでの認証

リモート API にアクセスできるようにするには、クライアントは LXD サーバーに認証しなければなりません。いくつかの認証方法があります。詳細は**リモート *API* 認証**を参照してください。

お勧めの方法はクライアントの TLS 証明書をトラストトークンを使ってサーバーのトラストストアに追加することです。トラストトークンを使ってクライアントを認証するには、以下の手順を実行します。

1. サーバーで、以下のコマンドを入力します。

   ```
   lxc config trust add
   ```

追加したいクライアントの名前を入力します。クライアント証明書を追加するのに使用できるトークンをコマンドが生成し表示します。

2. クライアントで、以下のコマンドでサーバーを追加します。

```
lxc remote add <remote_name> <token>
```

注釈: LXD サーバーが NAT の後ろ側にいる場合、クライアント用のリモートを追加する際には外部のパブリックアドレスを指定する必要があります。

```
lxc remote add <name> <IP_address>
```

admin パスワードのプロンプトが表示されたら、生成されたトークンを入力してください。

サーバーでトークンを生成する際、 LXD はクライアントがサーバーにアクセスするために使える IP アドレスのリストを含めます。しかし、サーバーが NAT の後ろ側にいる場合、これらのアドレスはクライアントが接続できないローカルアドレスの場合があります。その場合、手動で外部アドレスを指定する必要があります。

詳細や他の認証方法については**リモート *API* 認証**を参照してください。

3.4 インスタンス

3.4.1 インスタンスについて

LXD 以下のインスタンスタイプをサポートします。

コンテナ コ

ンテナはデフォルトのインスタンスタイプです。コンテナは現状 LXD インスタンスの最も完全な実装であり、仮想マシンよりも多くの機能をサポートしています。

コンテナは `liblxc` (LXC) を使って実装されています。

仮想マシン

Virtual machines (VMs) は LXD バージョン 4.0 以降ネイティブにサポートされています。ビルトインのエージェントのおかげで、ほぼコンテナと同様に使えます。

LXD は仮想マシンの機能を提供するために `qemu` を使用しています。

注釈: 現状、仮想マシンはコンテナよりサポートする機能が少ないですが、将来には両方のインスタンスタイプで同じ機能セットをサポートする計画です。

仮想マシンでどの機能が利用可能かを見るには、**インスタンスオプション** ドキュメントの条件のカラムを確認してください。

3.4.2 インスタンスを作成するには

インスタンスを作成するには、`lxc init` か `lxc launch` コマンドを使用できます。`lxc init` コマンドはインスタンスを作成だけしますが、`lxc launch` コマンドは作成して起動します。

使い方

コンテナを作成するには以下のコマンドを入力します。

```
lxc launch|init <image_server>:<image_name> <instance_name> [flags]
```

イメージ イ

メージは必要最小限のオペレーティングシステム (例えば、Linux ディストリビューション) と LXD 関連の情報を含みます。さまざまなオペレーティングシステムのイメージがビルトインのリモートイメージサーバーで利用できます。詳細は **イメージ** を参照してください。

イメージがローカルにない場合、イメージサーバーとイメージの名前を指定 (例えば、LXD のビルトインのイメージサーバー上の 22.04 Ubuntu イメージなら `images:ubuntu/22.04`) する必要があります。

インスタンス名 イ

ンスタンス名は LXD の運用環境 (そしてクラスタ内) でユニークである必要があります。追加の要件については **インスタンスプロパティ** を参照してください。

フラグ フ

ラグの完全なリストについては `lxc launch --help` か `lxc init --help` を参照してください。よく使うフラグは以下のとおりです。

- `--config` は新しいインスタンスの設定オプションを指定します
- `--device` はプロファイルを通して提供されるデバイスの **デバイスオプション** を上書きします
- `--profile` は新しいインスタンスに使用する **プロファイル** を指定します
- `--network` や `--storage` は新しいインスタンスに指定のネットワークやストレージプールを使用させます
- `--target` は指定のクラスタメンバー上にインスタンスを作成します
- `--vm` はコンテナではなく仮想マシンを作成します

設定ファイルを渡す

インスタンス設定をフラグとして指定する代わりに、YAML ファイルでコマンドに渡すことができます。

例えば、config.yaml の設定でコンテナを起動するには、以下のコマンドを入力します。

```
lxc launch images:ubuntu/22.04 ubuntu-config < config.yaml
```

Tip: YAML ファイルの必要な文法を見るには既存のインスタンス設定の中身を確認 (lxc config show <instance_name> -e) してください。

例

以下の例では lxc launch を使用しますが、同じように lxc init も使用できます。

コンテナを起動する

images サーバーの Ubuntu 22.04 のイメージで ubuntu-container というインスタンス名でコンテナを起動するには、以下のコマンドを入力します。

```
lxc launch images:ubuntu/22.04 ubuntu-container
```

仮想マシンを起動する

images サーバーの Ubuntu 22.04 のイメージで ubuntu-vm というインスタンス名で仮想マシンを起動するには、以下のコマンドを入力します。

```
lxc launch images:ubuntu/22.04 ubuntu-vm --vm
```

コンテナを指定の設定で起動する

コンテナを起動しリソースを 1 つの vCPU と 192MiB の RAM に限定するには、以下のコマンドを入力します。

```
lxc launch images:ubuntu/22.04 ubuntu-limited --config limits.cpu=1 --config limits.
↪memory=192MiB
```

指定のクラスタメンバー上で仮想マシンを起動する

クラスタメンバー server2 上で仮想マシンを起動するには、以下のコマンドを入力します。

```
lxc launch images:ubuntu/22.04 ubuntu-container --vm --target server2
```

指定のインスタンスタイプでコンテナを起動する

LXD ではクラウドのシンプルなインスタンスタイプが使えます。これは、インスタンスの作成時に指定できる文字列で表されます。

3 つの指定方法があります:

- <instance type>
- <cloud>:<instance type>
- c<CPU>-m<RAM in GB>

例えば、次の 3 つは同じです:

- t2.micro
- aws:t2.micro
- c1-m1

コマンドラインでは、インスタンスタイプは次のように指定します:

```
lxc launch ubuntu:22.04 my-instance -t t2.micro
```

使えるクラウドとインスタンスタイプのリストは https://github.com/dustinkirkland/instance-type で確認できます。

3.4.3 インスタンスを管理するには

全てのインスタンスを一覧表示するには以下のコマンドを入力します。

```
lxc list
```

表示するインスタンスをフィルターできます。例えば、インスタンスタイプ、状態、またはインスタンスが配置されているクラスタメンバーでフィルターできます。

```
lxc list type=container
lxc list status=running
lxc list location=server1
```

インスタンス名でフィルターもできます。複数のインスタンスを一覧表示するには、名前の正規表現を使います。例えば以下のようにします。

```
lxc list ubuntu.*
```

全てのフィルターオプションを見るには `lxc list --help` と入力します。

インスタンスの情報を表示する

インスタンスについての詳細情報を表示するには以下のコマンドを入力します。

```
lxc info <instance_name>
```

インスタンスの最新のログの行を表示するにはコマンドに `--show-log` を追加します。

```
lxc info <instance_name> --show-log
```

インスタンスを起動する

インスタンスを起動するには以下のコマンドを入力します。

```
lxc start <instance_name>
```

インスタンスが存在しないか既に稼働中の場合はエラーになります。

起動する際にコンソールにすぐにアタッチするには `--console` フラグを渡します。例えば以下のようにします。

```
lxc start <instance_name> --console
```

詳細は **コンソールにアクセスするには** を参照してください。

インスタンスを停止する

インスタンスを停止するには以下のコマンドを入力します。

```
lxc stop <instance_name>
```

インスタンスが存在しないか稼働中ではない場合はエラーになります。

インスタンスを削除する

インスタンスがもう不要な場合、削除できます。削除する前にインスタンスを停止する必要があります。

インスタンスを削除するには以下のコマンドを入力します。

```
lxc delete <instance_name>
```

> **注意:** このコマンドはインスタンスとそのスナップショットを永久的に削除します。

間違ってインスタンスを削除するのを防ぐ

間違ってインスタンスを削除するのを防ぐには 2 つの方法があります。

- `lxc delete` コマンドを使うたびに承認のプロンプトを表示するには、エイリアスを作成します。

  ```
  lxc alias add delete "delete -i"
  ```

- 特定のインスタンスが削除されることを防ぐためには、そのインスタンスの *security.protection.delete* を `true` に設定します。手順は **インスタンスを設定するには** を参照してください。

インスタンスを再構築する

インスタンスの root ディスクを一掃して再初期化したいがインスタンスの設定は維持したい場合、インスタンスを再構築できます。

再構築はスナップショットが 1 つも存在しないインスタンスでのみ可能です。

再構築の前にインスタンスを停止します。そして、以下のコマンドのいずれかを入力します。

- 別のイメージでインスタンスを再構築する。

  ```
  lxc rebuild <image_name> <instance_name>
  ```

- 空のルートディスクでインスタンスを再構築する。

```
lxc rebuild <instance_name> --empty
```

rebuild コマンドについてのより詳細な情報は lxc rebuild --help を参照してください。

3.4.4 インスタンスを設定するには

インスタンスオプション を設定するか **デバイス** を設定することでインスタンスを設定できます。

設定方法は以下の項を参照してください。

注釈: 異なるインスタンス設定を保管し再利用するには、**プロファイル** を使用してください。

インスタンスオプションを設定する

インスタンスを作成する 際にインスタンスオプションを指定できます。

インスタンスが作成された後にインスタンスオプションを更新するには、lxc config set コマンドを使います。インスタンス名とインスタンスオプションのキーとバリューを指定します。

```
lxc config set <instance_name> <option_key>=<option_value> <option_key>=<option_value> ..
↪.
```

利用可能なオプションの一覧とどのオプションがどのインスタンスタイプで利用可能かの情報は **インスタンスオプション** を参照してください。

例えば、コンテナのメモリーリミットを変更するには、以下のコマンドを入力します。

```
lxc config set my-container limits.memory=128MiB
```

注釈: 一部のインスタンスオプションはインスタンスが稼働中に即座に更新されます。他のインスタンスオプションはインスタンスの再起動後に更新されます。

どのオプションがインスタンス稼働中に即座に反映されるかの情報は **インスタンスオプション** の "ライブアップデート" 列を参照してください。

デバイスを設定する

インスタンスにインスタンスデバイスを追加や設定するには、`lxc config device add` コマンドを使います。一般的に、デバイスはコンテナの稼働中に追加または削除できます。VM はいくつかのデバイスタイプではホットプラグをサポートしますが、全てではありません。

インスタンス名、デバイス名、デバイスタイプと (**デバイスタイプ** ごとに) 必要に応じてデバイスオプションを指定します。

```
lxc config device add <instance_name> <device_name> <device_type> <device_option_key>=
↪<device_option_value> <device_option_key>=<device_option_value> ...
```

利用可能なデバイスタイプとそのオプションについては **デバイス** を参照してください。

注釈: 各デバイスのエントリはインスタンスごとにユニークな名前により識別します。

プロファイルに定義されたデバイスは、プロファイルがインスタンスに割り当てられる順番にインスタンスに適用されます。インスタンス設定内に直接定義されたデバイスは最後に適用されます。各ステージで、より以前のステージに同じ名前のデバイスがある場合は、デバイスエントリ全体が最後の定義により上書きされます。

デバイス名は最大 64 文字です。

例えば、ホストシステムの `/share/c1` 上のストレージをインスタンスのパス `/opt` に追加するには、以下のコマンドを入力します。

```
lxc config device add my-container disk-storage-device disk source=/share/c1 path=/opt
```

以前追加したデバイスのインスタンスデバイスオプションを設定するには、以下のコマンドを入力します。

```
lxc config device set <instance_name> <device_name> <device_option_key>=<device_option_
↪value> <device_option_key>=<device_option_value> ...
```

注釈: デバイスオプションは **インスタンスの作成** 時に `--device` フラグを使って指定することもできます。これは **プロファイル** を通して提供されるデバイスのデバイスオプションを上書きしたい場合に有用です。

デバイスを除去するには、`lxc config device remove` コマンドを使います。利用可能なコマンドの完全なリストは `lxc config device --help` を参照してください。

インスタンス設定を表示する

書き込み可能なインスタンスプロパティ、インスタンスオプション、デバイスとデバイスオプションを含むインスタンスの現在の設定を表示するには、以下のコマンドを入力します。

```
lxc config show <instance_name> --expanded
```

インスタンス設定全体を編集する

書き込み可能なインスタンスプロパティ、インスタンスオプション、デバイスとデバイスオプションを含むインスタンス設定全体を編集するには、以下のコマンドを入力します。

```
lxc config edit <instance_name>
```

注釈: 利便性のため、`lxc config edit` コマンドは読み取り専用のインスタンスプロパティを含む設定全体を表示します。しかし、これらのプロパティは変更できません。変更しても無視されます。

3.4.5 インスタンススナップショットを作成するには

インスタンスのスナップショットを作成することでその時点のインスタンスを保存でき、これによりインスタンスを元の状態に簡単に復元できます。

インスタンススナップショットはインスタンスボリュームと同じストレージプールに保管されます。

スナップショットを作成する

インスタンスのスナップショットを作成するには以下のコマンドを使用します。

```
lxc snapshot <instance_name> [<snapshot name>]
```

既存のスナップショットを置き換えるにはスナップショット名に `--reuse` フラグを追加します。

`snapshots.expiry` 設定オプションが設定されていない限り、デフォルトではスナップショットは永遠に保持されます。特定のスナップショットをこの期限が切れても維持するには `--no-expiry` フラグを使用します。

仮想マシンの場合は、`--stateful` フラグを追加すると、インスタンスボリュームに含まれるデータだけでなくインスタンスの稼働状態もキャプチャーします。この機能は CRIU の制限により、コンテナでは完全にはサポートされていないことに注意してください。

スナップショットを表示、編集、削除する

インスタンスのスナップショットを表示するには以下のコマンドを使用します。

```
lxc info <instance_name>
```

スナップショットを <instance_name>/<snapshot_name> で参照することで、インスタンスと同様にスナップショットを表示や変更できます。

スナップショットについての設定情報を表示するには、以下のコマンドを使用します。

```
lxc config show <instance_name>/<snapshot_name>
```

スナップショットの有効期限を変更するには、以下のコマンドを使用します。

```
lxc config edit <instance_name>/<snapshot_name>
```

注釈: 一般には、スナップショットはインスタンスの状態を保存しているので、編集できません。唯一の例外が有効期限です。設定に対する他の変更は黙って無視されます。

スナップショットを削除するには、以下のコマンドを使用します。

```
lxc delete <instance_name>/<snapshot_name>
```

インスタンススナップショットをスケジュールする

指定の日時 (最大で毎分 1 回) にスナップショットを自動的に作成するようにインスタンスを設定できます。そうするには *snapshots.schedule* インスタンスオプションを設定します。

例えば、日次スナップショットを設定するには、以下のコマンドを使用します。

```
lxc config set <instance_name> snapshots.schedule @daily
```

毎日午前 6 時にスナップショットを取得するように設定するには、以下のコマンドを使用します。

```
lxc config set <instance_name> snapshots.schedule "0 6 * * *"
```

定期的なスナップショットをスケジュールする際は、自動的な削除 (*snapshots.expiry*) とスナップショットの命名パターン (*snapshots.pattern*) の設定を検討してください。また稼働中でないインスタンスのスナップショットを取得するかどうか (*snapshots.schedule.stopped*) も設定すると良いかもしれません。

インスタンススナップショットを復元する

任意のスナップショットにインスタンスを復元できます。

そうするには、以下のコマンドを使用します。

```
lxc restore <instance_name> <snapshot_name>
```

スナップショットがステートフル (インスタンスの稼働状態についての情報を含むという意味) な場合、--stateful フラグを追加すると状態も復元できます。

3.4.6 プロファイルを使用するには

プロファイルは一組の設定オプションを保持します。プロファイルにはインスタンスオプション、デバイスとデバイスオプションを含められます。

1 つのインスタンスには任意の数のプロファイルを適用できます。プロファイルは指定された順番に適用され、その結果最後に指定したプロファイルが特定のキーを上書きします。どのような場合でも、インスタンス固有の設定はプロファイル由来のものを上書きします。

注釈: プロファイルはコンテナと仮想マシンに適用できます。ですので、どちらのタイプに有効なオプションとデバイスを含めることができます。

インスタンスタイプに適用できない設定を含むプロファイルを適用すると、この設定は無視されエラーにはなりません。

新しいインスタンスを起動する際にプロファイルを指定しない場合は、自動的には default プロファイルが適用されます。このプロファイルはネットワークインタフェースとルートディスクを定義します。default プロファイルはリネームや削除はできません。

プロファイルを表示する

全ての利用可能なプロファイルを一覧表示するには以下のコマンドを入力します。

```
lxc profile list
```

プロファイルの内容を表示するには以下のコマンドを入力します。

```
lxc profile show <profile_name>
```

空のプロファイルを作成する

空のプロファイルを作成するには以下のコマンドを入力します。

```
lxc profile create <profile_name>
```

プロファイルを編集する

プロファイルの特定の設定オプションを設定するか、あるいは YAML 形式でプロファイル全体を編集できます。

プロファイルの特定の設定オプションを設定する

プロファイルのインスタンスオプションを設定するには、`lxc profile set` コマンドを使います。プロファイル名とインスタンスオプションのキーとバリューを指定します。

```
lxc profile set <profile_name> <option_key>=<option_value> <option_key>=<option_value> ..
↪.
```

プロファイルのインスタンスデバイスを追加と変更するには、`lxc profile device add` コマンドを使います。プロファイル名、デバイス名、デバイスタイプと (**デバイスタイプ**ごとの) 必要に応じてデバイスオプションを指定します。

```
lxc profile device add <instance_name> <device_name> <device_type> <device_option_key>=
↪<device_option_value> <device_option_key>=<device_option_value> ...
```

以前にプロファイルに追加したデバイスのインスタンスデバイスオプションを設定するには、`lxc profile device set` コマンドを使います。

```
lxc profile device set <instance_name> <device_name> <device_option_key>=<device_option_
↪value> <device_option_key>=<device_option_value> ...
```

プロファイル全体を編集する

個々の設定オプションを別々に設定する代わりに、YAML 形式で一度にすべてのオプションを提供できます。

既存のプロファイルまたはインスタンス設定の中身で必要なマークアップをチェックします。例えば、`default` プロファイルは以下のようになっているかもしれません。

```
config: {}
description: Default LXD profile
devices:
```

(次のページに続く)

(前のページからの続き)

```
  eth0:
    name: eth0
    network: lxdbr0
    type: nic
  root:
    path: /
    pool: default
    type: disk
name: default
used_by:
```

インスタンスオプションは config の下に提供されます。インスタンスデバイスとインスタンスデバイスオプションは devices の下に提供されます。

ターミナルの標準エディタを使ってプロファイルを編集するには、以下のコマンドを入力します。

```
lxc profile edit <profile_name>
```

別の方法として、設定を含む YAML ファイル (例えば、profile.yaml) を作成して、以下のコマンドで設定をプロファイルに書き込めます。

```
lxc profile edit <profile_name> < profile.yaml
```

インスタンスにプロファイルを適用する

インスタンスにプロファイルを適用するには以下のコマンドを入力します。

```
lxc profile add <instance_name> <profile_name>
```

Tip: プロファイル追加後に lxc config show <instance_name>を実行して設定を確認します。

プロファイルが profiles の下に一覧表示されます。しかし、プロファイルからの設定オプションは config の下には表示されません (--expanded フラグを追加しない限り)。この挙動の理由はこれらの設定はプロファイルからは取得されインスタンス設定から取得されるわけではないからです。

これはプロファイルを編集する場合、変更はプロファイルを使用している全てのインスタンスに自動的に適用されることを意味します。

インスタンスの起動時に--profile フラグを追加してプロファイルを指定することもできます。

```
lxc launch <image> <instance_name> --profile <profile> --profile <profile> ...
```

インスタンスからプロファイルを削除する

インスタンスからプロファイルを削除するには以下のコマンドを入力します。

```
lxc profile remove <instance_name> <profile_name>
```

3.4.7 cloud-init を使用するには

cloud-init は Linux ディストリビューションのインスタンスの自動的な初期化とカスタマイズのためのツールです。

インスタンスに cloud-init 設定を追加することで、インスタンスの最初の起動時に cloud-init に特定のアクションを実行させることができます。可能なアクションには、例えば以下のようなものがあります：

- パッケージの更新とインストール
- 特定の設定の適用
- ユーザーの追加
- サービスの有効化
- コマンドやスクリプトの実行
- VM のファイルシステムをディスクのサイズに自動的に拡張する

詳細な情報は Cloud-init documentation を参照してください。

注釈: cloud-init アクションはインスタンスの最初の起動時に一度だけ実行されます。インスタンスの再起動ではアクションは再実行されません。

イメージ内の cloud-init サポート

cloud-init を使用するには、cloud-init がインストールされたイメージをベースにインスタンスを作る必要があります。

- ubuntu および ubuntu-daily **イメージサーバー**からのすべてのイメージは cloud-init をサポートしています。

- images リモートからのイメージには cloud-init が有効化されたバリアントがあり、通常デフォルトバリアントよりもサイズが大きくなります。クラウドバリアントは/cloud 接尾辞を使用します。例えば、images:ubuntu/22.04/cloud。

設定オプション

LXD は、cloud-init の設定に対して cloud-init.*と user.*の 2 つの異なる設定オプションセットをサポートしています。どちらのセットを使用する必要があるかは、使用するイメージの cloud-init サポートによって異なります。一般的には、新しいイメージは cloud-init.*設定オプションをサポートし、古いイメージは user.*をサポートしていますが、例外も存在する可能性があります。

以下の設定オプションがサポートされています。

- cloud-init.vendor-data または user.vendor-data (Vendor data を参照)
- cloud-init.user-data または user.user-data (User data formats を参照)
- cloud-init.network-config または user.network-config (Network configuration を参照)

設定オプションの詳細については、*cloud-init* **インスタンスオプション**と、cloud-init ドキュメント内の LXD データソースを参照してください。

ベンダーデータとユーザーデータ

vendor-data と user-data の両方が、cloud-init にクラウド構成データを提供するために使用されます。

主な考え方は、vendor-data は一般的なデフォルト構成に使用され、user-data はインスタンス固有の構成に使用されることです。これは、プロファイルで vendor-data を指定し、インスタンス構成で user-data を指定する必要があることを意味します。LXD はこの方法を強制しませんが、プロファイルとインスタンス構成の両方で vendor-data と user-data を使用することができます。

インスタンスに対して vendor-data と user-data の両方が提供される場合、cloud-init は 2 つの構成をマージします。しかし、両方の設定で同じキーを使った場合、マージは不可能になるかもしれません。この場合、指定されたデータをどのようにマージするべきかを clout-init に指定してください。Merging user data sections を参照して手順を確認してください。

cloud-init の設定方法

インスタンスの cloud-init を設定するには、対応する設定オプションをインスタンスが使用する**プロファイル**または**インスタンス構成**に直接追加します。

インスタンスに直接 cloud-init を設定する場合、cloud-init はインスタンスの最初の起動時にのみ実行されることに注意してください。つまり、インスタンスを起動する前に cloud-init を設定する必要があります。これを行うには、lxc launch の代わりに lxc init でインスタンスを作成し、設定が完了した後に起動します。

cloud-init 設定の YAML フォーマット

cloud-init のオプションでは、YAML の literal スタイルフォーマットが必要です。パイプ記号 (|) を使用して、パイプの後にインデントされたテキスト全体を、改行とインデントを保持したまま cloud-init に単一の文字列として渡すことを示します。

vendor-data および user-data のオプションは通常、#cloud-config で始まります。

例：

```
config:
  cloud-init.user-data: |
    #cloud-config
    package_upgrade: true
    packages:
      - package1
      - package2
```

Tip: 構文が正しいかどうかを確認する方法については、How to debug user data を参照してください。

cloud-init のステータスを確認する方法

cloud-init はインスタンスの最初の起動時に自動的に実行されます。設定されたアクションによっては、完了するまでに時間がかかる場合があります。

cloud-init のステータスを確認するには、インスタンスにログインして以下のコマンドを入力します。

```
cloud-init status
```

結果が status: running の場合、cloud-init はまだ実行中です。結果が status: done の場合、完了しています。

また、--wait フラグを使用して、cloud-init が完了したときにのみ通知を受け取ることができます：

```
root@instance:~# cloud-init status --wait     .....................................status:
done
```

ユーザーデータやベンダーデータを指定する方法

user-data と vendor-data の設定は、例えば、パッケージのアップグレードやインストール、ユーザーの追加、コマンドの実行などに使用することができます。

提供される値は、最初の行で cloud-init に渡されるユーザーデータ形式のタイプを示す必要があります。パッケージのアップグレードやユーザーの設定などのアクティビティには、#cloud-config が使用するデータ形式です。

構成データは、インスタンスのルートファイルシステム内の以下のファイルに保存されます：

- /var/lib/cloud/instance/cloud-config.txt
- /var/lib/cloud/instance/user-data.txt

例

以下のセクションでは、さまざまな例のユースケースに対するユーザーデータ（またはベンダーデータ）の設定を参照してください。

より高度な例は、cloud-init ドキュメントで見つけることができます。

パッケージのアップグレード

インスタンスが作成された直後に、インスタンスのリポジトリからパッケージをアップグレードするためには、package_upgrade キーを使用します：

```
config:
  cloud-init.user-data: |
    #cloud-config
    package_upgrade: true
```

パッケージのインストール

インスタンスのセットアップ時に特定のパッケージをインストールするには、packages キーを使用し、パッケージ名をリストとして指定します：

```
config:
  cloud-init.user-data: |
    #cloud-config
    packages:
      - git
      - openssh-server
```

タイムゾーンの設定

インスタンス作成時にインスタンスのタイムゾーンを設定するには、timezone キーを使用します：

```
config:
  cloud-init.user-data: |
    #cloud-config
    timezone: Europe/Rome
```

コマンドの実行

コマンド（マーカーファイルの書き込みなど）を実行するには、runcmd キーを使用し、コマンドをリストとして指定します：

```
config:
  cloud-init.user-data: |
    #cloud-config
    runcmd:
      - [touch, /run/cloud.init.ran]
```

ユーザーアカウントの追加

ユーザーアカウントを追加するには、user キーを使用します。デフォルトユーザーやサポートされているキーに関する詳細は、cloud-init ドキュメント内の Including users and groups の例を参照してください。

```
config:
  cloud-init.user-data: |
    #cloud-config
    user:
      - name: documentation_example
```

ネットワーク構成データを指定する方法

デフォルトでは、cloud-init はインスタンスの eth0 インターフェイスに DHCP クライアントを設定します。デフォルトの構成を上書きするために、network-config オプションを使用して独自のネットワーク構成を定義することができます（これはテンプレートの構造によるものです）。

その後、cloud-init は Ubuntu リリースに応じて ifupdown か netplan を使用して、システム上の関連するネットワーク構成をレンダリングします。

構成データは、インスタンスのルートファイルシステム内の以下のファイルに保存されます：

- /var/lib/cloud/seed/nocloud-net/network-config
- /etc/network/interfaces.d/50-cloud-init.cfg (ifupdown を使用している場合)
- /etc/netplan/50-cloud-init.yaml (netplan を使用している場合)

例

特定のネットワークインターフェースに静的な IPv4 アドレスを設定し、カスタム名前サーバーを使用するための次の設定を使用します：

```
config:
  cloud-init.network-config: |
    version: 1
    config:
      - type: physical
        name: eth1
        subnets:
          - type: static
            ipv4: true
            address: 10.10.101.20
            netmask: 255.255.255.0
            gateway: 10.10.101.1
            control: auto
      - type: nameserver
        address: 10.10.10.254
```

3.4.8 インスタンス内でコマンドを実行するには

LXD では、ネットワークを経由してインスタンスにアクセスする必要なしに、LXD クライアントを使用してインスタンス内でコマンドを実行できます。

コンテナでは、これは常に機能し、LXD によって直接処理されます。仮想マシンでは、これが機能するには、仮想マシン内 lxd-agent プロセスが稼働している必要があります。

インスタンス内部でコマンドを実行するには lxc exec コマンドを使います。シェルコマンド (例えば、 /bin/bash) を実行することで、インスタンスにシェルアクセスできます。

インスタンス内部でコマンドを実行する

ホストマシンの端末から単一のコマンドを実行するには、`lxc exec` コマンドを使います。

```
lxc exec <instance_name> -- <command>
```

例えば、コンテナ上のパッケージリストを更新するには以下のコマンドを入力します。

```
lxc exec ubuntu-container -- apt-get update
```

実行モード

インタラクティブ・モードでは、入力 (stdin) と出力 (stdout, stderr) を扱うために疑似端末装置 (PTS) が使用されます。これは、ターミナル・エミュレータに接続されている場合（スクリプトから実行されていない場合)、CLI によって自動的に選択されます。インタラクティブ・モードを強制するには、`--force-interactive` か`--mode interactive` をコマンドに追加します。

非インタラクティブ・モードでは、代わりにパイプが (stdin、stdout、stderr のそれぞれに 1 つずつ) 割り当てられます。これにより、多くのスクリプトで必要とされるように、コマンドを実行しながら、stdin、stdout、stderr を別々に適切に取得することができます。非インタラクティブ・モードを強制するには、`--force-noninteractive` か`--mode non-interactive` をコマンドに追加します。

ユーザー、グループ、作業ディレクトリ

LXD はインスタンス内のデータを読まない、あるいはその中にあるものを信用しないというポリシーを持っています。これは、LXD がユーザーやグループの解決を処理するために、`/etc/passwd`、`/etc/group` や`/etc/nsswitch.conf` のようなものを解析しないことを意味しています。

結果として、LXD はユーザのホームディレクトリがどこにあるか、あるいはどのような補助的なグループがあるかを知りません。

デフォルトでは、LXD は root (UID 0)、デフォルトのグループ (GID 0) としてコマンドを実行し、作業ディレクトリは`/root` に設定されています。ユーザー、グループ、作業ディレクトリは以下のフラグによって上書きできます。

- `--user` - コマンドを実行するユーザ ID
- `--group` - コマンドを実行するグループ ID
- `--cwd` - コマンドを実行するディレクトリ

環境

以下の 2 つの方法で exec セッションに環境変数を渡せます。

インスタンスオプションとして環境変数を渡す イ

ンスタンス内で ENVVAR 環境変数を VALUE に設定するには、environment.ENVVAR **インスタンスオプション**を設定します。

```
lxc config set <instance_name> environment.ENVVAR=VALUE
```

exec コマンドに環境変数を渡す

exec コマンドに環境変数を渡すには、--env フラグを使います。例えば以下のようにします。

```
lxc exec <instance_name> --env ENVVAR=VALUE -- <command>
```

さらに、LXD は (上記のいずれかの方法で渡されない限り) 以下のデフォルト値を設定します。

変数名	条件	値
PATH	-	以下を連結したもの • /usr/local/sbin • /usr/local/bin • /usr/sbin • /usr/bin • /sbin • /bin • /snap (適切な場合) • /etc/NIXOS (適切な場合)
LANG	-	C.UTF-8
HOME	root で実行する場合 (UID 0)	/root
USER	root で実行する場合 (UID 0)	root

インスタンスにシェルアクセスする

インスタンス内で直接コマンドを実行したい場合、インスタンス内でシェルコマンドを実行します。例えば、以下のコマンド (インスタンス内に/bin/bash コマンドが存在する想定) を入力します。

```
lxc exec <instance_name> -- /bin/bash
```

デフォルトでは root ユーザでログインします。別のユーザでログインしたい場合は、以下のコマンドを入力します。

```
lxc exec <instance_name> -- su --login <user_name>
```

注釈: インスタンス内で稼働しているオペレーティングシステムによっては、先にユーザを作成する必要があるかもしれません。

インスタンスシェルを終了するには、exit を入力するか Ctrl+d を押します。

3.4.9 コンソールにアクセスするには

インスタンスのコンソールにアタッチするには lxc console コマンドを使います。コンソールは起動時に既に利用可能になり、必要なら、ブートメッセージを見て、コンテナや仮想マシンの起動時の問題をデバッグするのに使えます。

インタラクティブなコンソールに接続するには、以下のコマンドを入力します。

```
lxc console <instance_name>
```

ログ出力を見るには --show-log フラグを渡します。

```
lxc console <instance_name> --show-log
```

インスタンスが起動したらすぐにコンソールにアタッチできます。

```
lxc start <instance_name> --console
lxc start <instance_name> --console=vga
```

グラフィカルなコンソールにアタッチする (仮想マシンの場合)

仮想マシンでは、コンソールにログオンしてグラフィカルな出力を見ることができます。コンソールを使えば、例えば、グラフィカルなインタフェースを使ってオペレーティングシステムをインストールしたりデスクトップ環境を実行できます。

さらなる利点は lxd-agent プロセスが実行していなくても、コンソールは利用可能です。これは lxd-agent が起動する前や lxd-agent が全く利用可能でない場合にもコンソール経由で仮想マシンにアクセスできることを意味します。

仮想マシンにグラフィカルなアウトプットを持つ VGA コンソールを開始するには、 SPICE クライアント (例えば、virt-viewer または spice-gtk-client) をインストールする必要があります。次に以下のコマンドを入力します。

```
lxc console <vm_name> --type vga
```

3.4.10 インスタンス内のファイルにアクセスするには

LXD クライアントを使って、ネットワーク経由でインスタンスにアクセスする必要なしに、インスタンス内部のファイルを管理できます。ファイルを個別に編集、削除したり、ローカルマシンからプッシュ、ローカルマシンにプルできます。あるいは、インスタンスのファイルシステムをローカルマシン上にマウントできます。

コンテナでは、これらの操作は必ず機能し LXD で直接処理されます。仮想マシンでは、これらの操作が機能するためには `lxd-agent` プロセスが仮想マシン内部で稼働している必要があります。

インスタンスのファイルを編集する

ローカルマシンからインスタンスのファイルを編集するには、以下のコマンドを入力します。

```
lxc file edit <instance_name>/<path_to_file>
```

例えば、インスタンス内の `/etc/hosts` ファイルを編集するには、以下のコマンドを入力します。

```
lxc file edit my-container/etc/hosts
```

注釈: ファイルはインスタンス上に既に存在している必要があります。インスタンス上にファイルを作成するのに `edit` コマンドは使えません。

インスタンスからファイルを削除する

インスタンスからファイルを削除するには、以下のコマンドを入力します。

```
lxc file delete <instance_name>/<path_to_file>
```

インスタンスからローカルマシンにファイルをプルする

インスタンスからローカルマシンにファイルをプルするには、以下のコマンドを入力します。

```
lxc file pull <instance_name>/<path_to_file> <local_file_path>
```

例えば `/etc/hosts` ファイルをカレントディレクトリにプルするには、以下のコマンドを入力します。

```
lxc file pull my-instance/etc/hosts .
```

インスタンスのファイルをローカルマシンにプルする代わりに、標準出力にプルして別のコマンドの標準入力にパイプすることもできます。これは、例えば、ログファイルをチェックするのに便利です。

```
lxc file pull my-instance/var/log/syslog - | less
```

ディレクトリの全ての中身をプルするには、以下のコマンドを入力します。

```
lxc file pull -r <instance_name>/<path_to_directory> <local_location>
```

ローカルマシンからインスタンスにファイルをプッシュする

ローカルマシンからインスタンスにファイルをプッシュするには、以下のコマンドを入力します。

```
lxc file push <local_file_path> <instance_name>/<path_to_file>
```

ディレクトリの全ての中身をプッシュするには、以下のコマンドを入力します。

```
lxc file push -r <local_location> <instance_name>/<path_to_directory>
```

インスタンスのファイルシステムをマウントする

インスタンスのファイルシステムをクライアントのローカルパスにマウントできます。

そうするためには、`sshfs` がインストールされていることを確認してください。次に以下のコマンドを入力します (snap をお使いの場合はコマンドは root 権限を必要とします)。

```
lxc file mount <instance_name>/<path_to_directory> <local_location>
```

これでローカルマシンからファイルにアクセスできます。

SSH SFTP リスナーをセットアップする

別の方法として、SSH SFTP リスナーをセットアップすることもできます。この方法では任意の SFTP クライアントで専用のユーザ名で接続できます。また、snap をお使いの場合、 root 権限を必要としません。

そうするには、まず以下のコマンドを入力してリスナーをセットアップします。

```
lxc file mount <instance_name> [--listen <address>:<port>]
```

例えば、ローカルマシン上のランダムなポート (例えば、127.0.0.1:45467) にリスナーをセットアップするには以下のようにします。

```
lxc file mount my-instance
```

ローカルネットワークの外側からインスタンスのファイルにアクセスしたい場合、特定のアドレスとポートを渡せます。

```
lxc file mount my-instance --listen 192.0.2.50:2222
```

> **注意:** あなたのインスタンスをリモートに公開することになるので、これを実行する際には注意してください。

特定のアドレスとランダムなポートでリスナーをセットアップするには以下のようにします。

```
lxc file mount my-instance --listen 192.0.2.50:0
```

コマンドは割り当てられたポートと接続に使用するユーザ名とパスワードを出力します。

Tip: --auth-user フラグを渡すとユーザ名を指定できます。

この情報を使ってファイルシステムにアクセスします。例えば sshfs で接続するには、以下のコマンドを入力します。

```
sshfs <user_name>@<address>:<path_to_directory> <local_location> -p <port>
```

例えば以下のようにします。

```
sshfs xFn8ai8c@127.0.0.1:/home my-instance-files -p 35147
```

これでインスタンスのファイルシステムをローカルマシン上の指定の場所でアクセスできます。

3.4.11 インスタンスの起動に失敗する問題のトラブルシューティング方法

インスタンスの起動に失敗し、エラー状態になる場合、これは通常、インスタンスの作成に使用したイメージまたはサーバー設定に関連する大きな問題を示しています。

問題のトラブルシューティングを行うには、以下の手順を完了してください：

1. 関連するログファイルとデバッグ情報を保存します：

   インスタンスログ イ

   ンスタンスログを表示するには、次のコマンドを入力します：

   ```
   lxc info <instance_name> --show-log
   ```

   コンソールログ コ

   ンソールログを表示するには、次のコマンドを入力します：

   ```
   lxc console <instance_name> --show-log
   ```

   詳細なサーバー情報

   LXD の snap パッケージには、デバッグ用の関連サーバー情報を収集するツールが含まれています。それを実行するには、次のコマンドを入力します：

   ```
   sudo lxd.buginfo
   ```

2. LXD サーバーを実行しているマシンを再起動します。

3. インスタンスをもう一度起動してみてください。エラーが再発した場合は、ログを比較して同じエラーかどうか確認します。

   同じエラーであり、ログ情報からエラーの原因を特定できない場合は、フォーラムで質問を投稿してください。収集したログファイルを含めるようにしてください。

トラブルシューティングの例

この例では、systemd が起動できない RHEL 7 システムを調査しましょう。

`user@host:~$ lxc console --show-log systemd Console log: Failed to insert module 'autofs4'Failed to insert module 'unix'Failed to mount sysfs at /sys: Operation not permittedFailed to mount proc at /proc: Operation not permitted[!!!!!!] Failed to mount API filesystems, freezing.` ここでのエラーは、 /sys と /proc がマウントできないと言っています - これは、非特権コンテナでは正しいです。しかし、LXD は可能であればこれらのファイルシステムを自動的にマウントします。

コンテナの要件では、すべてのコンテナには空の /dev、/proc、/sys ディレクトリが必要であり、/sbin/init が存在していなければなりません。これらのディレクトリが存在しない場合、LXD はそれらをマウントできず、systemd がその後それを試みます。これは非特権コンテナなので、systemd はこれを行う能力がなく、コンテナはフリーズします。

したがって、何も変更される前の環境を確認でき、raw.lxc 設定パラメータを使用してコンテナ内の init システムを明示的に変更できます。これは、Linux カーネルのコマンドラインで init=/bin/bash を設定するのと同等です。

```
lxc config set systemd raw.lxc 'lxc.init.cmd = /bin/bash'
```

これがどのように見えるかを示します：

user@host:~$ lxc config set systemd raw.lxc 'lxc.init.cmd = /bin/bash' user@host:~$ lxc start systemd user@host:~$ lxc console --show-log systemd Console log: [root@systemd /]# これでコンテナが起動したので、確認して期待通りにうまく動作していないことがわかります：

user@host:~$ lxc exec systemd bash [root@systemd ~]# ls[root@systemd ~]# mountmount: failed to read mtab: No such file or directory[root@systemd ~]# cd /[root@systemd /]# ls /proc/sys[root@systemd /]# exit LXD は自動的に回復しようとするため、起動時にいくつかのディレクトリが作成されました。コンテナをシャットダウンして再起動すると問題が解決しますが、元の原因はまだ残っています - テンプレートには必要なファイルが含まれていません。

3.4.12 インスタンスの設定

インスタンス設定は以下の異なるカテゴリから構成されます。

インスタンスプロパティ イ

ンスタンスプロパティはインスタンスが作成されるときに指定されます。これには、例えば、インスタンス名やアーキテクチャが含まれます。いくつかのプロパティは読み取り専用で作成後は変更できませんが、他のプロパティは **インスタンス設定全体を編集する** 際に更新できます。

YAML 設定内では、プロパティはトップレベルにあります。

利用可能なインスタンスプロパティのレファレンスは **インスタンスプロパティ** を参照してください。

インスタンスオプション イ

ンスタンスオプションはインスタンスに直接関連する設定オプションです。これには、例えば、起動時のオプション、セキュリティ設定、ハードウェアのリミット、カーネルモジュール、スナップショット、そしてユーザの鍵を含みます。これらのオプションはインスタンスの作成時に (--config key=value フラグを使って) キー/バリューペアで指定できます。作成後は lxc config set や lxc config unset コマンドで変更できます。

YAML 設定内では、オプションは config エントリの下に配置されます。

利用可能なインスタンスオプションのレファレンスは **インスタンスオプション**、オプションをどのように設定するかの手順は **インスタンスオプションを設定する** を参照してください。

インスタンスデバイス インスタンスデバイスはインスタンスにアタッチされます。これらは、例えば、ネットワークインタフェース、マウントポイント、USB そして GPU デバイスが含まれます。通常、デバイスはインスタンスを作成した後に `lxc config device add` コマンドで追加しますが、プロファイルやインスタンスを作成するのに使用する YAML 設定ファイルに追加することもできます。

各デバイスタイプには固有のオプションのセットがあり、**インスタンスデバイスオプション**として参照されます。

YAML 設定内では、デバイスは `devices` エントリの下に配置されます。

利用可能なデバイスと対応するインスタンスデバイスオプションのレファレンスについては **デバイス**、インスタンスデバイスをどのように追加し設定するかの手順は **デバイスを設定する** を参照してください。

インスタンスプロパティ

インスタンスプロパティはインスタンスが作成されたときに設定されます。これらは**プロファイル**の一部にはできません。

以下のインスタンスプロパティが利用可能です。

プロパティ	読み取り専用	説明
`name`	yes	インスタンス名 (**インスタンス名の要件**参照)
`architecture`	no	インスタンスアーキテクチャ

インスタンス名の要件

インスタンス名は `lxc rename` コマンドでインスタンスをリネームすることでのみ変更できます。

有効なインスタンス名は次の要件を満たさなければなりません。

- 名前は 1〜63 文字である必要があります。
- 名前は ASCII テーブルの文字、数字、ダッシュのみを含む必要があります。
- 名前は数字またはダッシュで始まってはいけません。
- 名前はダッシュで終わってはいけません。

これらの要件は、インスタンス名が DNS レコードとして、ファイルシステム上で、色々なセキュリティプロファイル、そしてインスタンス自身のホスト名として使えるように定められています。

インスタンスオプション

インスタンスオプションはインスタンスに直接関係する設定オプションです。

インスタンスオプションをどのように設定するかの手順は**インスタンスオプションを設定する**を参照してください。

key/value 形式の設定は、名前空間で分けられています。以下のオプションが利用できます。

- **その他のオプション**
- **ブート関連のオプション**
- *cloud-init* **設定**
- **リソース制限**
- **マイグレーションオプション**
- *NVIDIA* **と** *CUDA* **の設定**
- *raw* **インスタンス設定のオーバーライド**
- **セキュリティポリシー**
- **スナップショットのスケジュールと設定**
- **揮発性の内部データ**

各オプションに型が定義されていますが、全ての値は文字列として保管され、REST API で文字列としてエクスポートされる (こうすることで後方互換性を壊すことなく任意の追加の値をサポートできます) ことに注意してください。

その他のオプション

以下のセクションに一覧表示される設定オプションに加えて、以下のインスタンスオプションがサポートされます。

キー	型	デフォルト値	ライブアップデート	条件	説明
agent.nic_config	bool	false	no	仮想マシン	デフォルトのネットワークインタフェースの名前と MTU をインスタンスデバイスと同じにするかどうかを制御 (これはコンテナでは自動でそうなります)
cluster.evacuate	string	auto	no	-	インスタンス退避時に何をするか (auto, migrate, live-migrate, stop)
environment.*	string	-	yes (exec)	-	インスタンス実行時に設定される key/value 形式の環境変数
linux.kernel_modules	string	-	yes	コンテナ	インスタンスを起動する前にロードするカーネルモジュールのカンマ区切りのリスト
linux.sysctl.*	string	-	no	コンテナ	コンテナ内の対応する sysctl 設定を上書きする値
user.*	string	-	no	-	自由形式のユーザー定義の key/value の設定の組 (検索に使えます)

ブート関連のオプション

以下のインスタンスオプションはインスタンスのブート関連の挙動を制御します。

キー	型	デフォルト値	ライブアップデート	条件	説明
boot.autostart	bool	-	n/a	-	LXD 起動時に常にインスタンスを起動するかどうかを制御 (設定しない場合、最後の状態がリストアされます)
boot.autostart.delay	integer	0	n/a	-	インスタンスが起動した後に次のインスタンスが起動するまで待つ秒数
boot.autostart.priority	integer	0	n/a	-	インスタンスを起動させる順番 (高いほど早く起動します)
boot.host_shutdown_	integer	30	yes	-	強制停止前にインスタンスが停止するのを待つ秒数
boot.stop.priority	integer	0	n/a	-	インスタンスの停止順 (高いほど早く停止します)

cloud-init 設定

以下のインスタンスオプションはインスタンスの *cloud-init* 設定を制御します。

キー	型	デフォルト値	ライブアップデート	条件	説明
cloud-init.network-config	string	DHCP on eth0	no	イメージでサポートされている場合	cloud-init のネットワーク設定 (設定はシード値として使用)
cloud-init.user-data	string	#cloud-c	no	イメージでサポートされている場合	cloud-init のユーザデータ (設定はシード値として使用)
cloud-init.vendor-data	string	#cloud-c	no	イメージでサポートされている場合	cloud-init のベンダーデータ (設定はシード値として使用)
user.network-config	string	DHCP on eth0	no	イメージでサポートされている場合	cloud-init.network-config のレガシーバージョン
user.user-data	string	#cloud-c	no	イメージでサポートされている場合	cloud-init.user-data のレガシーバージョン
user.vendor-data	string	#cloud-c	no	イメージでサポートされている場合	cloud-init.vendor-data のレガシーバージョン

これらのオプションのサポートは使用するイメージに依存し、保証はされません。

cloud-init.user-data と cloud-init.vendor-data の両方を指定すると、両方のオプションの設定がマージされます。このため、これらのオプションに設定する cloud-init 設定が同じキーを含まないようにしてください。

リソース制限

以下のインスタンスオプションはインスタンスのリソース制限を指定します。

キー	型	デフォルト値	ライブアップデート	条件	説明
limits.cpu	string	仮想マシンでは 1 CPU	yes	-	インスタンスに割り当てる CPU 番号、もしくは番号の範囲。*CPU* **ピンニング**参照
limits.cpu.allowance	string	100%	yes	コンテナ	どれくらい CPU を使えるかを制御。ソフトリミットとしてパーセント指定 (50%) か固定値として単位時間内に使える時間 (25ms/100ms) を指定できます。**割り当てと優先度 *(コンテナのみ)*** 参照
limits.cpu.nodes	string	-	yes	-	インスタンスの CPU を配置する NUMA ノード ID あるいは ID の範囲のカンマ区切りリスト。**割り当てと優先度 *(コンテナのみ)*** 参照。
limits.cpu.priority	integer	10 (最大値)	yes	コンテナ	リソースをオーバーコミットする際に同じ CPU をシェアする他のインスタンスと比較した CPU スケジューリングの優先度 (0〜10 の整数)。**割り当てと優先度 *(コンテナのみ)*** 参照
limits.disk.priority	integer	5 (中央値)	yes	-	高負荷時に、インスタンスの I/O リクエストに割り当てる優先度を制御 (0〜10 の整数)
limits.hugepages.64KB	string	-	yes	コンテナ	64 KB huge pages の数を制限するためのバイト数 (さまざまな単位が指定可能、**ストレージとネットワーク制限の単位**参照) の固定値。*huge page* **の制限** 参照
limits.hugepages.1MB	string	-	yes	コンテナ	1 MB huge pages の数を制限するためのバイト数 (さまざまな単位が指定可能、**ストレージとネットワーク制限の単位**参照) の固定値。*huge page* **の制限** 参照
limits.hugepages.2MB	string	-	yes	コンテナ	2 MB huge pages の数を制限するためのバイト数 (さまざまな単位が指定可能、**ストレージとネットワーク制限の単位**参照) の固定値。*huge page* **の制限** 参照
limits.hugepages.1GB	string	-	yes	コンテナ	1 GB huge pages の数を制限するためのバイト数 (さまざまな単位が指定可能、**ストレージとネットワーク制限の単位**参照) の固定値。*huge page* **の制限** 参照
limits.kernel.*	string	-	no	コンテナ	インスタンスごとのカーネルリソースの制限 (例、オープンできるファイルの数)。**カーネルリソース制限**参照

CPU 制限

CPU 使用率を制限するための異なるオプションがあります：

- `limits.cpu` を設定して、インスタンスが見ることができ、使用することができる CPU を制限します。このオプションの設定方法は、*CPU* **ピンニング**を参照してください。
- `limits.cpu.allowance` を設定して、インスタンスが利用可能な CPU にかける負荷を制限します。このオプションはコンテナのみで利用可能です。このオプションの設定方法は、**割り当てと優先度 (コンテナのみ)** を参照してください。

これらのオプションは同時に設定して、インスタンスが見ることができる CPU とそれらのインスタンスの許可される使用量の両方を制限することが可能です。しかし、`limits.cpu.allowance` を時間制限と共に使用する場合、スケジューラーに多くの制約をかけ、効率的な割り当てが難しくなる可能性があるため、`limits.cpu` の追加使用は避けるべきです。

CPU 制限は cgroup コントローラの `cpuset` と `cpu` を組み合わせて実装しています。

CPU ピンニング

`limits.cpu` は `cpuset` コントローラを使って、CPU を固定 (ピンニング) します。どの CPU を、またはどれぐらいの数の CPU を、インスタンスから見えるようにし、使えるようにするかを指定できます。

- どの CPU を使うかを指定するには、`limits.cpu` を CPU の組み合わせ (例:`1,2,3`) あるいは CPU の範囲 (例:`0-3`) で指定できます。

  単一の CPU にピンニングするためには、CPU の個数との区別をつけるために、範囲を指定する文法 (例:`1-1`) を使う必要があります。

- CPU の個数を指定した場合 (例:`4`)、LXD は特定の CPU にピンニングされていない全てのインスタンスをダイナミックに負荷分散し、マシン上の負荷を分散しようとします。インスタンスが起動したり停止するたびに、またシステムに CPU が追加されるたびに、インスタンスはリバランスされます。

仮想マシンの CPU リミット

注釈: LXD は `limits.cpu` オプションのライブアップデートをサポートします。しかし、仮想マシンの場合は、対応する CPU がホットプラグされるだけです。ゲストのオペレーティングシステムによって、新しい CPU をオンラインにするためには、インスタンスを再起動するか、なんらかの手動の操作を実行する必要があります。

LXD の仮想マシンはデフォルトでは 1 つの vCPU だけを割り当てられ、ホストの CPU のベンダーとタイプとマッチした CPU として現れますが、シングルコアでスレッドなしになります。

`limits.cpu` を単一の整数に設定する場合、LXD は複数の vCPU を割り当ててゲストにはフルなコアとして公開します。これらの vCPU はホスト上の特定の物理コアにはピンニングされません。vCPU の個数は VM の稼働中

に変更できます。

limits.cpu を CPU ID(lxc info --resources で表示されます) の範囲またはカンマ区切りリストの組に設定する場合、vCPU は物理コアにピンニングされます。このシナリオでは、LXD は CPU 設定が現実のハードウェアトポロジーとぴったり合うかチェックし、合う場合はそのトポロジーをゲスト内に複製します。CPU ピンニングを行う場合、VM の稼働中に設定を変更することはできません。

例えば、ピンニング設定が 8 個のスレッド、同じコアのスレッドの各ペアと 2 個の CPU に散在する偶数のコアを持つ場合、ゲストは 2 個の CPU、各 CPU に 2 個のコア、各コアに 2 個のスレッドを持ちます。NUMA レイアウトは同様に複製され、このシナリオでは、ゲストではほとんどの場合、2 個の NUMA ノード、各 CPU ソケットに 1 個のノードを持つことになるでしょう。

複数の NUMA ノードを持つような環境では、メモリは同様に NUMA ノードで分割され、ホスト上で適切にピンニングされ、その後ゲストに公開されます。

これら全てにより、ゲストスケジューラはソケット、コア、スレッドを適切に判断し、メモリを共有したり NUMA ノード間でプロセスを移動する際に NUMA トポロジーを考慮できるので、ゲスト内で非常に高パフォーマンスな操作を可能にします。

割り当てと優先度 (コンテナのみ)

limits.cpu.allowance は、時間の制限を与えたときは CFS スケジューラのクォータを、パーセント指定をした場合は全体的な CPU シェアの仕組みを使います。

- 時間制限 (例:20ms/50ms) はハードリミットです。例えば、コンテナが最大で 1 つの CPU を使用することを許可する場合は、limits.cpu.allowance を 100ms/100ms のような値に設定します。この値は 1 つの CPU に相当する時間に対する相対値なので、2 つの CPU の時間を制限するには、100ms/50ms あるいは 200ms/100ms のような値を使用します。
- パーセント指定を使う場合は、制限は負荷状態にある場合のみに適用されるソフトリミットです。設定は、同じ CPU(もしくは CPU の組) を使う他のインスタンスとの比較で、インスタンスに対するスケジューラの優先度を計算するのに使われます。例えば、負荷時のコンテナの CPU 使用率を 1 つの CPU に制限するためには、limits.cpu.allowance を 100% に設定します。

limits.cpu.nodes はインスタンスが使用する CPU を特定の NUMA ノードに限定するのに使えます。

- どの NUMA ノードを使用するか指定するには、limits.cpu.nodes に NUMA ノード ID の組 (例えば 0,1) または NUMA ノードの範囲 (例えば、0-1,2-4) のどちらかを設定します。

limits.cpu.priority は、CPU の組を共有する複数のインスタンスに割り当てられた CPU の割合が同じ場合に、スケジューラの優先度スコアを計算するために使われる別の因子です。

huge page の制限

LXD では `limits.hugepage.[size]` キーを使ってコンテナが利用できる huge page の数を制限できます。

アーキテクチャはしばしば huge page のサイズを公開しています。利用可能な huge page サイズはアーキテクチャによって異なります。

huge page の制限は非特権コンテナ内で `hugetlbfs` ファイルシステムの `mount` システムコールをインターセプトするように LXD を設定しているときには特に有用です。LXD が `hugetlbfs mount` システムコールをインターセプトすると LXD は正しい `uid` と `gid` の値を `mount` オプションに指定して `hugetblfs` ファイルシステムをコンテナにマウントします。これにより非特権コンテナからも huge page が利用可能となります。しかし、ホストで利用可能な huge page をコンテナが使い切ってしまうのを防ぐため、`limits.hugepages.[size]` を使ってコンテナが利用可能な huge page の数を制限することを推奨します。

huge page の制限は `hugetlb` cgroup コントローラによって実行されます。これはこれらの制限を適用するために、ホストシステムが `hugetlb` コントローラをレガシーあるいは cgroup の単一階層構造 (訳注:cgroup v2) に公開する必要があることを意味します。

カーネルリソース制限

LXD は、インスタンスのリソース制限を設定するのに使用できる一般の名前空間キー `limits.kernel.*`を公開しています。

`limits.kernel.*`接頭辞に続いて指定されるリソースについて LXD が全く検証を行わないという意味でこれは汎用です。LXD は対象のカーネルがサポートする全ての利用可能なリソースについて知ることはできません。代わりに、LXD は `limits.kernel.*`接頭辞の後の対応するリソースキーとその値をカーネルに単に渡します。カーネルが適切な検証を行います。これによりユーザーはシステム上でサポートされる任意の制限を指定できます。

よくある制限のいくつかは以下のとおりです。

キー	リソース	説明
limits.kernel.as	RLIMIT_AS	プロセスの仮想メモリーの最大サイズ
limits.kernel.core	RLIMIT_CORE	プロセスのコアダンプファイルの最大サイズ
limits.kernel.cpu	RLIMIT_CPU	プロセスが使える CPU 時間の秒単位の制限
limits.kernel.data	RLIMIT_DATA	プロセスのデータセグメントの最大サイズ
limits.kernel.fsize	RLIMIT_FSIZE	プロセスが作成できるファイルの最大サイズ
limits.kernel.locks	RLIMIT_LOCKS	プロセスが確立できるファイルロック数の制限
limits.kernel.memlock	RLIMIT_MEMLOCK	プロセスが RAM 上でロックできるメモリのバイト数の制限
limits.kernel.nice	RLIMIT_NICE	引き上げることができるプロセスの nice 値の最大値
limits.kernel.nofile	RLIMIT_NOFILE	プロセスがオープンできるファイルの最大値
limits.kernel.nproc	RLIMIT_NPROC	呼び出し元プロセスのユーザーが作れるプロセスの最大数
limits.kernel.rtprio	RLIMIT_RTPRIO	プロセスに対して設定できるリアルタイム優先度の最大値
limits.kernel.sigpending	RLIMIT_SIGPENDIN	呼び出し元プロセスのユーザーがキューに入れられるシグナルの最大数

指定できる制限の完全なリストは getrlimit(2)/setrlimit(2) システムコールの man ページで確認できます。

limits.kernel.*名前空間内で制限を指定するには、RLIMIT_を付けずに、リソース名を小文字で指定します。例えば、RLIMIT_NOFILE は nofile と指定します。

制限は、コロン区切りのふたつの数字もしくは unlimited という文字列で指定します (例:limits.kernel.nofile=1000:2000)。単一の値を使って、ソフトリミットとハードリミットを同じ値に設定できます (例:limits.kernel.nofile=3000)。

明示的に設定されないリソースは、インスタンスを起動したプロセスから継承されます。この継承は LXD でなく、カーネルによって強制されることに注意してください。

マイグレーションオプション

以下のインスタンスオプションはインスタンスが**ある *LXD* サーバーから別のサーバーに移動される**場合の挙動を制御します。

キー	型	デフォルト値	ライブアップデート	条件	説明
migration.incremental.memory	bool	false	yes	コンテナ	インスタンスのダウンタイムを短くするためにインスタンスのメモリを増分転送するかどうかを制御
migration.incremental.memory.goal	integer	70	yes	コンテナ	インスタンスを停止させる前に同期するメモリの割合 (%)
migration.incremental.memory.iterations	integer	10	yes	コンテナ	インスタンスを停止させる前に完了させるメモリ転送処理の最大数
migration.stateful	bool	false	no	仮想マシン	ステートフルな停止/開始とスナップショットを許可するかどうかを制御 (有効にするとこれと非互換ないくつかの機能は使えなくなります)

NVIDIA と CUDA の設定

以下のインスタンスオプションはインスタンスの NVIDIA と CUDA の設定を指定します。

キー	型	デフォルト値	ライブアップデート	条件	説明
nvidia.driver.capabilities	string	compute,utility	no	コンテナ	インスタンスに必要なドライバケーパビリティ (libnvidia-container に環境変数 NVIDIA_DRIVER_CAPABILITIES を設定)
nvidia.runtime	bool	false	no	コンテナ	ホストの NVIDIA と CUDA ランタイムライブラリをインスタンス内でも使えるようにする
nvidia.require.cuda	string	-	no	コンテナ	必要となる CUDA バージョンのバージョン表記 (libnvidia-container に環境変数 NVIDIA_REQUIRE_CUDA を設定)
nvidia.require.driver	string	-	no	コンテナ	必要となるドライババージョンのバージョン表記 (libnvidia-container に環境変数 NVIDIA_REQUIRE_DRIVER を設定)

raw インスタンス設定のオーバーライド

以下のインスタンスオプションは LXD 自身が使用するバックエンド機能に直接制御できるようにします。

キー	型	デフォルト値	ライブアップデート	条件	説明
raw.apparmor	blob	-	yes	-	生成されたプロファイルに追加する AppArmor プロファイルエントリ
raw.idmap	blob	-	no	非特権コンテナ	生 (raw) の idmap 設定 (例:both 1000 1000)
raw.lxc	blob	-	no	コンテナ	生成された設定に追加する生 (raw) の LXC 設定
raw.qemu	blob	-	no	仮想マシン	生成されたコマンドラインに追加される生 (raw) の QEMU 設定
raw.qemu.conf	blob	-	no	仮想マシン	生成された qemu.conf に追加/オーバーライドする (*QEMU* **設定のオーバーライド**参照)
raw.seccomp	blob	-	no	コンテナ	生 (raw) の Seccomp 設定

重要: これらの raw.*キーを設定すると LXD を予期せぬ形で壊してしまうかもしれません。このため、これらのキーを設定するのは避けるほうが良いです。

QEMU 設定のオーバーライド

VM インスタンスに対しては、LXD は-readconfig コマンドラインオプションで QEMU に渡す設定ファイルを使って QEMU を設定します。この設定ファイルは各インスタンスの起動前に生成されます。設定ファイルは/var/log/lxd/<instance_name>/qemu.conf に作られます。

デフォルトの設定はほとんどの典型的な利用ケース、VirtIO デバイスを持つモダンな UEFI ゲスト、では正常に動作します。しかし、いくつかの状況では、生成された設定をオーバーライドする必要があります。例えば以下のような場合です。

- UEFI をサポートしない古いゲスト OS を実行する。
- VirtIO がゲスト OS でサポートされない場合にカスタムな仮想デバイスを指定する。
- マシンの起動前に LXD でサポートされないデバイスを追加する。
- ゲスト OS と衝突するデバイスを削除する。

設定をオーバーライドするには、raw.qemu.conf オプションを設定します。これは qemu.conf と似たような形式ですが、いくつか拡張した形式をサポートします。これは複数行の設定オプションですので、複数のセクションやキーを変更するのに使えます。

- 生成された設定ファイルのセクションやキーを置き換えるには、別の値を持つセクションを追加します。

  例えば、デフォルトの virtio-gpu-pci GPU ドライバをオーバーライドするには以下のセクションを使います。

```
raw.qemu.conf: |-
    [device "qemu_gpu"]
    driver = "qxl-vga"
```

- セクションを削除するには、キー無しのセクションを指定します。例えば以下のようにします。

```
raw.qemu.conf: |-
    [device "qemu_gpu"]
```

- キーを削除するには、空の文字列を値として指定します。例えば以下のようにします。

```
raw.qemu.conf: |-
    [device "qemu_gpu"]
```

(次のページに続く)

(前のページからの続き)

```
    driver = ""
```

- 新規のセクションを追加するには、設定ファイル内に存在しないセクション名を指定します。

QEMU で使用される設定ファイル形式は同じ名前の複数のセクションを許可します。以下は LXD で生成される設定の抜粋です。

```
[global]
driver = "ICH9-LPC"
property = "disable_s3"
value = "1"

[global]
driver = "ICH9-LPC"
property = "disable_s4"
value = "1"
```

オーバーライドするセクションを指定するには、インデクスを指定します。例えば以下のようにします。

```
raw.qemu.conf: |-
    [global][1]
    value = "0"
```

セクションのインデクスは 0(指定しない場合のデフォルト値) から始まりますので、上の例は以下の設定を生成します。

```
[global]
driver = "ICH9-LPC"
property = "disable_s3"
value = "1"

[global]
driver = "ICH9-LPC"
property = "disable_s4"
value = "0"
```

セキュリティポリシー

以下のインスタンスオプションはインスタンスのセキュリティポリシーを制御します。

キー	型	デフォルト値	ライブアップデート	条件	説
security.csm	bool	false	no	仮想マシン	U
security.devlxd	bool	true	no	-	イ
security.devlxd.images	bool	false	no	コンテナ	d
security.idmap.base	integer	-	no	非特権コンテナ	害
security.idmap.isolated	bool	false	no	非特権コンテナ	イ
security.idmap.size	integer	-	no	非特権コンテナ	使
security.nesting	bool	false	yes	コンテナ	イ
security.privileged	bool	false	no	コンテナ	特
security.protection.delete	bool	false	yes	-	イ
security.protection.shift	bool	false	yes	コンテナ	イ
security.agent.metrics	bool	true	no	仮想マシン	状
security.secureboot	bool	true	no	仮想マシン	U
security.sev	bool	false	no	仮想マシン	A
security.sev.policy.es	bool	false	no	仮想マシン	A
security.sev.session.dh	string	true	no	仮想マシン	ゲ
security.sev.session.data	string	true	no	仮想マシン	ゲ
security.syscalls.allow	string	-	no	コンテナ	\
security.syscalls.deny	string	-	no	コンテナ	\
security.syscalls.deny_compat	bool	false	no	コンテナ	x
security.syscalls.deny_default	bool	true	no	コンテナ	デ
security.syscalls.intercept.bpf	bool	false	no	コンテナ	b
security.syscalls.intercept.bpf.devices	bool	false	no	コンテナ	c
security.syscalls.intercept.mknod	bool	false	no	コンテナ	m
security.syscalls.intercept.mount	bool	false	no	コンテナ	m
security.syscalls.intercept.mount.allowed	string	-	yes	コンテナ	イ
security.syscalls.intercept.mount.fuse	string	-	yes	コンテナ	F
security.syscalls.intercept.mount.shift	bool	false	yes	コンテナ	m
security.syscalls.intercept.sched_setscheduler	bool	false	no	コンテナ	s
security.syscalls.intercept.setxattr	bool	false	no	コンテナ	s
security.syscalls.intercept.sysinfo	bool	false	no	コンテナ	s

スナップショットのスケジュールと設定

以下のインスタンスオプションは *instance snapshots* の作成と削除を制御します。

キー	型	デフォルト値	ライブアップデート	条件	説明
snapshots.schedule	string	-	no	-	Cron 表記 (<minute> <hour> <dom> <month> <dow>)、またはスケジュールエイリアスのカンマ区切りリスト (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly)、または自動スナップショットを無効にする場合は空文字 (デフォルト)
snapshots.schedule.stopped	bool	false	no	-	停止したインスタンスのスナップショットを自動的に作成するかどうかを制御
snapshots.pattern	string	snap%	no	-	スナップショットの名前を表す Pongo2 テンプレート文字列 (スケジュールされたスナップショットと名前無しのスナップショットで使用)。**スナップショットの自動命名**参照
snapshots.expiry	string	-	no	-	スナップショットをいつ削除するかを制御 (1M 2H 3d 4w 5m 6y のような式を期待)

スナップショットの自動命名

snapshots.pattern オプションはスナップショット名をフォーマットする Pongo2 テンプレート文字列です。

スナップショット名にタイムスタンプを追加するには、Pongo2 コンテキスト変数 creation_date を使用します。スナップショット名に使用できない文字を含まないようにテンプレート文字列をフォーマットするようにしてください。例えば、 snapshots.pattern を {{ creation_date|date:'2006-01-02_15-04-05' }} に設定し、作成日時を秒の制度まで落として、スナップショットを命名するようにします。

名前の衝突を防ぐ別の方法はパターン内に %d プレースホルダを使うことです。最初のスナップショットでは、プレースホルダは 0 に置換されます。後続のスナップショットでは、既存のスナップショットが考慮され、プレースホルダの位置の最大の数を見つけます。この数が 1 増加されて新しい名前に使用されます。

揮発性の内部データ

以下の揮発性のキーはインスタンスに固有な内部データを保管するため LXD で現在内部的に使用されています。

キー	型	説明
volatile.apply_template	string	次の起動時にトリガーされるテンプレートフックの名前
volatile.apply_nvram	string	次の起動時に仮想マシンの NVRAM を再生成するかどうか
volatile.base_image	string	インスタンスを作成したイメージのハッシュ (存在する場合)
volatile.cloud-init.instance-id	string	cloud-init に公開する instance-id(UUID)
volatile.evacuate.origin	string	退避したインスタンスのオリジン (クラスタメンバー)
volatile.idmap.base	integer	インスタンスの主 idmap の範囲の最初の ID
volatile.idmap.current	string	インスタンスで現在使用中の idmap
volatile.idmap.next	string	次にインスタンスが起動する際に使う idmap
volatile.last_state.idmap	string	シリアライズ化したインスタンスの UID/GID マップ
volatile.last_state.power	string	最後にホストがシャットダウンした時点のインスタンスの状態
volatile.vsock_id	string	最後の起動時に使用されたインスタンスの vsock ID
volatile.uuid	string	インスタンスの UUID(全サーバーとプロジェクト内でグローバルにユニーク)
volatile.uuid.generation	string	インスタンスの時間の位置が後退するたびに変わるインスタンス generation UUID (全サーバーとプロジェクト内でグローバルにユニーク)
volatile.<name>.apply_quota	string	次回のインスタンス起動時に適用されるディスククォータ
volatile.<name>.ceph_rbd	string	Ceph のディスクデバイスの RBD デバイスパス
volatile.<name>.host_name	string	ホスト上のネットワークデバイス名
volatile.<name>.hwaddr	string	ネットワークデバイスの MAC アドレス (hwaddr プロパティがデバイスに設定されていない場合)
volatile.<name>.last_state.created	string	物理デバイスのネットワークデバイスが作られたかどうか (true または false)
volatile.<name>.last_state.mtu	string	物理デバイスをインスタンスに移動したときに使われていたネットワークデバイスの元の MTU
volatile.<name>.last_state.hwaddr	string	物理デバイスをインスタンスに移動したときに使われていたネットワークデバイスの元の MAC
volatile.<name>.last_state.ip_addresses	string	ネットワークデバイスで最後に使用されていた IP アドレスのカンマ区切りリスト
volatile.<name>.last_state.vdpa.name	string	VDPA デバイスファイルディスクリプタをインスタンスに移動させる際に使用される VDPA デバイス名
volatile.<name>.	string	SR-IOV の仮想ファンクション (VF) をインスタンスに移動したときに使わ

注釈: 揮発性のキーはユーザは設定できません。

デバイス

デバイスはインスタンス (**デバイスを設定する** 参照) またはプロファイル (**プロファイルを編集する** 参照) にアタッチされます。

デバイスには、例えば、ネットワークインタフェース、マウントポイント、USB そして GPU デバイスがあります。これらのデバイスはインスタンスデバイスの種別に応じてインスタンスデバイスオプションを持つことができます。

LXD では次のデバイスタイプが使えます。

ID (データベース)	名前	条件	説明
0	*none*	-	継承ブロッカー
1	*nic*	-	ネットワークインタフェース
2	*disk*	-	インスタンス内のマウントポイント
3	*unix-char*	コンテナ	Unix キャラクタデバイス
4	*unix-block*	コンテナ	Unix ブロックデバイス
5	*usb*	-	USB デバイス
6	*gpu*	-	GPU デバイス
7	*infiniband*	コンテナ	インフィニバンドデバイス
8	*proxy*	コンテナ	プロキシデバイス
9	*unix-hotplug*	コンテナ	Unix ホットプラグデバイス
10	*tpm*	-	TPM デバイス
11	*pci*	仮想マシン	PCI デバイス

各インスタンスには一組の **標準デバイス** が付属します。

標準デバイス

LXD は、標準の POSIX システムが動作するのに必要な基本的なデバイスを常にインスタンスに提供します。これらはインスタンスやプロファイルの設定では見えず、上書きもできません。

標準デバイスは次のようなデバイスが含まれます。

デバイス	デバイスのタイプ
/dev/null	キャラクタデバイス
/dev/zero	キャラクタデバイス
/dev/full	キャラクタデバイス
/dev/console	キャラクタデバイス
/dev/tty	キャラクタデバイス
/dev/random	キャラクタデバイス
/dev/urandom	キャラクタデバイス
/dev/net/tun	キャラクタデバイス
/dev/fuse	キャラクタデバイス
lo	ネットワークインタフェース

これ以外に関しては、インスタンスの設定もしくはインスタンスで使われるいずれかのプロファイルで定義する必要があります。デフォルトのプロファイルには、インスタンス内で eth0 になるネットワークインタフェースが通常は含まれます。

タイプ: none

注釈: none デバイスタイプはコンテナと VM の両方でサポートされます。

none タイプのデバイスはプロパティを一切持たず、インスタンス内に何も作成しません。

これはプロファイルからのデバイスの継承を止めるためだけに存在します。そうするには、継承をスキップしたいデバイスと同じ名前の none タイプのデバイスを追加してください。

デバイスは、元のデバイスを含むプロファイルより後に適用するプロファイルに追加するか、インスタンス上に直接追加できます。

タイプ: nic

注釈: nic デバイスタイプはコンテナと VM の両方でサポートされます。

(ipvlan NIC タイプを除いて)NIC はコンテナと VM の両方でホットプラグをサポートします。

ネットワークデバイス (**ネットワークインタフェースコントローラー**や *NIC* とも呼びます) はネットワークへの接続を提供します。LXD はさまざまな異なるタイプのネットワークデバイス (*NIC* **タイプ**) をサポートします。

nictype 対 network

インスタンスにネットワークデバイスを追加する際には、追加したいデバイスのタイプを選択するのに 2 つの方法があります。nictype プロパティを指定するか network プロパティを使うかです。

これらの 2 つのデバイスオプションは相互排他であり、デバイスを作成時にどちらか 1 つのみ指定可能です。しかし、network オプションを指定する際には、nictype オプションはネットワークタイプから自動的に導出されることに注意してください。

nictype
: nictype デバイスオプションを使用する際は、LXD に管理されていないネットワークインタフェースを指定できます。このため、LXD がネットワークインタフェースを使用するために必要な全ての情報を指定する必要があります。

  この方法を使用する際は、nictype オプションはデバイス作成時に指定する必要があり、作成後は変更できません。

network
: network デバイスオプションを使用する際は、NIC は既存の**管理されたネットワーク**にリンクされます。この場合、LXD はネットワークについて必要な情報を全て持っているので、デバイス追加時にはネットワーク名を指定するだけでよいです。

  この方法を使用する際は、nictype オプションは LXD が自動的に導出します。値は読み取り専用で変更できません。

  ネットワークから継承される他のデバイスオプションは NIC 固有のデバイスオプションの「管理」カラムで「yes」と記載されています。network の方法を使う場合、NIC のこれらのオプションを直接カスタマイズはできません。

詳細な情報は**ネットワークについて**を参照してください。

利用可能な NIC

次の NIC は nictype か network オプションを使って追加できます。

- *bridged*: ホスト上に存在する既存のブリッジを使い、ホストのブリッジをインスタンスに接続する仮想デバイスペアを作成します。
- *macvlan*: 既存のネットワークデバイスをベースに MAC アドレスが異なる新しいネットワークデバイスを作成します。
- *sriov*: SR-IOV が有効な物理ネットワークデバイスの仮想ファンクション (virtual function) をインスタンスにパススルーします。
- *physical*: ホストの物理デバイスをインスタンスにパススルーします。対象のデバイスはホスト上では見えなくなり、インスタンス内に出現します。

次の NIC は network オプションでのみ追加できます。

- *ovn*: 既存の OVN ネットワークを使用し、インスタンスが接続する仮想デバイスペアを作成します。

次の NIC は nictype オプションでのみ追加できます。

- *ipvlan*: 既存のネットワークデバイスをベースに MAC アドレスは同じですが IP アドレスが異なる新しいネットワークデバイスを作成します。
- *p2p*: 仮想デバイスペアを作成し、片方をインスタンス内に置き、残りの片方をホスト上に残します。
- *routed*: 仮想デバイスペアを作成し、ホストからインスタンスに繋いで静的ルートとプロキシ ARP/NDP エントリーを作成します。これにより指定された親インタフェースのネットワークにインスタンスが参加できるようになります。

利用可能なデバイスオプションは NIC タイプによって異なり、以下のセクションの表に一覧表示されます。

nictype: bridged

注釈: この NIC タイプは nictype オプションか network オプションで選択できます（管理された bridge ネットワークの情報については**ブリッジネットワーク**参照）。

---

bridged NIC はホストの既存のブリッジを使用し、ホストのブリッジをインスタンスに接続するための仮想デバイスのペアを作成します。

デバイスオプション

bridged タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	管理	説明
boot.priority	integer	-	no	VM のブート優先度 (高いほうが先にブート)
host_name	string	ランダムに割り当て	no	ホスト内でのインタフェースの名前
hwaddr	string	ランダムに割り当て	no	新しいインタフェースの MAC アドレス
ipv4.address	string	-	no	DHCP でインスタンスに割り当てる IPv4 アドレス (security.ipv4_filtering 設定時に全ての IPv4 トラフィックを制限するには none と設定可能)
ipv4.routes	string	-	no	ホスト上で NIC に追加する IPv4 静的ルートのカンマ区切りリスト
ipv4.routes.external	string	-	no	NIC にルーティングしアップリンクのネットワーク (BGP) で公開する IPv4 静的ルートのカンマ区切りリスト
ipv6.address	string	-	no	DHCP でインスタンスに割り当てる IPv6 アドレス (security.ipv6_filtering 設定時に全ての IPv6 トラフィックを制限するには none と設定可能)
ipv6.routes	string	-	no	ホスト上で NIC に追加する IPv6 静的ルートのカンマ区切りリスト
ipv6.routes.external	string	-	no	NIC にルーティングしアップリンクのネットワーク (BGP) で公開する IPv6 静的ルートのカンマ区切りリスト
limits.egress	string	-	no	外向きトラフィックの I/O 制限値 (さまざまな単位が使用可能、**ストレージとネットワーク制限の単位**参照)
limits.ingress	string	-	no	内向きトラフィックの I/O 制限値 (さまざまな単位が使用可能、**ストレージとネットワーク制限の単位**参照)
limits.max	string	-	no	内向きと外向きの両方のトラフィック I/O 制限値 (limits.ingress と limits.egress の両方を設定するのと同じ)
maas.subnet.ipv4	string	-	yes	インスタンスを登録する MAAS IPv4 サブネット
maas.subnet.ipv6	string	-	yes	インスタンスを登録する MAAS IPv6 サブネット
mtu	integer	親の MTU	yes	新しいインタフェースの MTU
name	string	カーネルが割り当	no	インスタンス内でのインタフェースの名前

nictype: macvlan

注釈: この NIC タイプは nictype オプションか network オプションで選択できます（管理された macvlan ネットワークの情報については *macvlan* **ネットワーク**参照)。

macvlan NIC は既存の NIC をベースにしますが、MAC アドレスが異なる新しいネットワークデバイスをセットアップします。

macvlan NIC を使う場合、LXD ホストとインスタンス間の通信はできません。ホストとインスタンスの両方がゲートウェイと通信できますが、それらが直接通信はできません。

デバイスオプション

macvlan タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	管理	説明
boot.priority	integer	-	no	VM のブート優先度 (高いほうが先にブート)
gvrp	bool	false	no	GARP VLAN Registration Protocol を使って VLAN を登録する
hwaddr	string	ランダムに割り当て	no	新しいインタフェースの MAC アドレス
maas.subnet.ipv4	string	-	yes	インスタンスを登録する MAAS IPv4 サブネット
maas.subnet.ipv6	string	-	yes	インスタンスを登録する MAAS IPv6 サブネット
mtu	integer	親の MTU	yes	新しいインタフェースの MTU
name	string	カーネルが割り当て	no	インスタンス内部でのインタフェース名
network	string	-	no	(nictype を直接設定する代わりに) デバイスをリンクする先の管理されたネットワーク
parent	string	-	yes	ホストデバイスの名前 (nictype を直接設定する場合は必須)
vlan	integer	-	no	アタッチ先の VLAN ID

`nictype: sriov`

注釈: この NIC タイプは `nictype` オプションか `network` オプションで選択できます（管理された `sriov` ネットワークの情報については *SR-IOV* **ネットワーク**参照）。

`sriov` NIC は SR-IOV を有効にした物理ネットワークデバイスの仮想ファンクションをインスタンスにパススルーします。

SR-IOV を有効にしたネットワークデバイスは一組の仮想ファンクション (VF) をネットワークデバイスの単一の物理ファンクション (PF) に関連付けます。PF は標準的な PCIe 関数です。一方、VF はデータの移動に最適化された非常に軽量な PCIe 関数です。PF のプロパティを変えるのを防ぐため、VF の構成機能は限定されています。

VF はシステムには通常の PCIe デバイスのように見えますので、通常の物理デバイスと全く同じようにインスタンスにパススルーできます。

VF の割り当て

`sriov` インタフェースタイプは `parent` プロパティを通してシステム上の SR-IOV を有効にしたネットワークデバイスの名前を渡されることを想定しています。すると LXD はシステム上の任意の利用可能な VF をチェックします。

デフォルトでは、LXD は見つけた最初の未使用な VF を割り当てます。有効になっているものが 1 つもないか、有効な VF が全て使用中の場合、サポートされている VF の数を最大に上げて最初の未使用な VF を使用します。全ての利用可能な VF が使用中か、カーネルまたはカードが VF の数の増加をサポートしない場合は、LXD はエラーを返します。

注釈: LXD に特定の VF を使わせたい場合、`sriov` NIC の代わりに `physical` NIC を使用し、`parent` オプションを VF 名に設定してください。

デバイスオプション

`sriov` タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	管理	説明
boot.priority	integer	-	no	VM のブート優先度 (高いほうが先にブート)
hwaddr	string	ランダムに割り当て	no	新しいインタフェースの MAC アドレス
maas.subnet.ipv4	string	-	yes	インスタンスを登録する MAAS IPv4 サブネット
maas.subnet.ipv6	string	-	yes	インスタンスを登録する MAAS IPv6 サブネット
mtu	integer	カーネルが割り当て	yes	新しいインタフェースの MTU
name	string	カーネルが割り当て	no	インスタンス内部でのインタフェース名
network	string	-	no	(nictype を直接設定する代わりに) デバイスをリンクする先の管理されたネットワーク
parent	string	-	yes	ホストデバイスの名前 (nictype を直接設定する場合は必須)
security.mac_filtering	bool	false	no	インスタンスが他のインスタンスの MAC アドレスになりすますのを防ぐ
vlan	integer	-	no	アタッチ先の VLAN ID

nictype: ovn

注釈: この NIC タイプは network オプションでのみ選択できます (管理された ovn ネットワークの情報については *OVN* **ネットワーク**参照)。

---

ovn NIC は既存の OVN ネットワークを使用し、それにインスタンスが接続する仮想デバイスペアを作成します。

SR-IOV ハードウェアアクセラレーション

acceleration=sriov を使用するには、LXD ホスト内の Ethernet スイッチデバイスのドライバモデル (switchdev) をサポートする互換性のある SR-IOV 物理 NIC を持っている必要があります。LXD は物理 NIC(PF) が switchdev モードに設定され、OVN 統合 OVS ブリッジに接続され、1 つ以上の仮想ファンクション (VF) がアクティブになっていることを前提とします。

これを実現するには、基本的な前提条件となる以下のセットアップ手順に従ってください。

1. PF と VF をセットアップする

1. PF 上でいくつかの VF をアクティベートし (以下の例では enp9s0f0np0 とし、PCI アドレスは 0000:09:00.0 とします)、アンバインドします。
2. switchdev モードと PF 上の hw-tc-offload を有効にします。
3. VF をリバインドします。

```
echo 4 > /sys/bus/pci/devices/0000:09:00.0/sriov_numvfs
for i in $(lspci -nnn | grep "Virtual Function" | cut -d' ' -f1); do echo 0000:
↪$i > /sys/bus/pci/drivers/mlx5_core/unbind; done
devlink dev eswitch set pci/0000:09:00.0 mode switchdev
ethtool -K enp9s0f0np0 hw-tc-offload on
for i in $(lspci -nnn | grep "Virtual Function" | cut -d' ' -f1); do echo 0000:
↪$i > /sys/bus/pci/drivers/mlx5_core/bind; done
```

2. ハードウェアオフロードを有効にし、統合ブリッジ (通常 br-int と呼ばれます) に PF NIC を追加して OVS をセットアップします。

```
ovs-vsctl set open_vswitch . other_config:hw-offload=true
systemctl restart openvswitch-switch
ovs-vsctl add-port br-int enp9s0f0np0
ip link set enp9s0f0np0 up
```

VDPA ハードウェアアクセラレーション

acceleration=vdpa を使用するには互換性のある VDPA 物理 NIC が必要です。セットアップ手順は SR-IOV ハードウェアアクセラレーションと同様ですが、さらに vhost_vdpa モジュールをセットアップし、利用可能な VDPA 管理デバイスがあることを確認する必要があります:

```
modprobe vhost_vdpa && vdpa mgmtdev show
```

デバイスオプション

sriov タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	管理	説明
acceleration	string	none	no	ハードウェアオフローディングを有効にする (none か sriov か vdpa、*SR-IOV* **ハードウェアアクセラレーション**参照)
boot.priority	integer	-	no	VM のブート優先度 (高いほうが先にブート)
host_name	string	ランダムに割り当て	no	ホスト内部でのインタフェース名
hwaddr	string	ランダムに割り当て	no	新しいインタフェースの MAC アドレス
ipv4.address	string	-	no	DHCP でインスタンスに割り当てる IPv4 アドレス
ipv4.routes	string	-	no	NIC へルーティングする IPv4 静的ルートのカンマ区切りリスト
ipv4.routes.external	string	-	no	NIC へのルーティングとアップリンクネットワークでの公開に使用する IPv4 静的ルートのカンマ区切りリスト
ipv6.address	string	-	no	DHCP でインスタンスに割り当てる IPv6 アドレス
ipv6.routes	string	-	no	NIC へルーティングする IPv6 静的ルートのカンマ区切りリスト
ipv6.routes.external	string	-	no	NIC へのルーティングとアップリンクネットワークでの公開に使用する IPv6 静的ルートのカンマ区切りリスト
name	string	カーネルが割り当て	no	インスタンス内部でのインタフェース名
nested	string	-	no	この NIC をどの親 NIC の下にネストするか (vlan も参照)
network	string	-	yes	デバイスの接続先の管理されたネットワーク (必須)
security.acls	string	-	no	適用するネットワーク ACL のカンマ区切りリスト
security.acls.default.egress.action	string	reject	no	どの ACL ルールにもマッチしない外向きトラフィックに使うアクション
security.acls.default.egress.logged	bool	false	no	どの ACL ルールにもマッチしない外向きトラフィックをログ出力するかどうか
security.acls.default.ingress.action	string	reject	no	どの ACL ルールにもマッチしない内向きトラフィックに使うアクション
security.acls.default.ingress.logged	bool	false	no	どの ACL ルールにもマッチしない内向きトラフィックをログ出力するかどうか
vlan	integer	-	no	ネストする際に使用する VLAN ID (nested も参照)

nictype: physical

注釈:

- この NIC タイプは nictype オプションまたは network オプションで選択できます（管理された physical ネットワークの情報については**物理ネットワーク**参照)。
- それぞれの親デバイスに対して physical NIC は 1 つだけ持つことができます。

physical NIC はホストからパススルーされるそのままの物理デバイスを提供します。対象のデバイスはホストから消失し、インスタンス内に出現します (これは各ターゲットデバイスに physical NIC は 1 つだけ持つことができることを意味します)。

デバイスオプション

physical タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	管理	説明
boot.priority	integer	-	no	VM のブート優先度 (高いほうが先にブート)
gvrp	bool	false	no	GARP VLAN Registration Protocol を使って VLAN を登録する
hwaddr	string	ランダムに割り当て	no	新しいインタフェースの MAC アドレス
maas.subnet.ipv4	string	-	no	インスタンスを登録する MAAS IPv4 サブネット
maas.subnet.ipv6	string	-	no	インスタンスを登録する MAAS IPv6 サブネット
mtu	integer	親の MTU	no	新しいインタフェースの MTU
name	string	カーネルが割り当て	no	インスタンス内部でのインタフェース名
network	string	-	no	デバイスのリンク先 (nictype を直接指定する代わりに) の管理ネットワーク
parent	string	-	yes	ホストデバイスの名前 (必須)
vlan	integer	-	no	アタッチ先の VLAN ID

nictype: ipvlan

注釈:

- この NIC タイプはコンテナのみで利用でき、仮想マシンでは利用できません。
- この NIC タイプは nictype オプションでのみ選択できます。
- この NIC タイプはホットプラグをサポートしません。

---

ipvlan NIC は既存のネットワークデバイスを元に、同じ MAC アドレスですが IP アドレスは異なるような新しいネットワークデバイスをセットアップします。

ipvlan NIC を使う場合、LXD ホストとインスタンス間の通信はできません。ホストとインスタンスの両方がゲートウェイと通信できますが、それらが直接通信はできません。

LXD は現状 L2 と L3S モードで IPVLAN をサポートします。このモードでは、ゲートウェイは LXD により自動的に設定されますが、コンテナが起動する前に ipv4.address と ipv6.address の設定の 1 つあるいは両方を使うことにより IP アドレスを手動で指定する必要があります。

DNS

ネームサーバーは自動的には設定されないので、コンテナ内部で設定する必要があります。このためには、以下の sysctl の設定をしてください。

- IPv4 アドレスを使用する場合

```
net.ipv4.conf.<parent>.forwarding=1
```

- IPv6 アドレスを使用する場合

```
net.ipv6.conf.<parent>.forwarding=1
net.ipv6.conf.<parent>.proxy_ndp=1
```

デバイスオプション

ipvlan タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
gvrp	bool	false	GARP VLAN Registration Protocol を使って VLAN を登録する
hwaddr	string	ランダムに割り当て	新しいインタフェースの MAC アドレス
ipv4.address	string	-	インスタンスに追加する IPv4 静的アドレスのカンマ区切りリスト (l2 モードでは、CIDR 形式か/24 のサブネットの単一アドレスで指定可能)
ipv4.gateway	string	auto (l3s), - (l2)	l3s モードでは、デフォルト IPv4 ゲートウェイを自動的に追加するかどうか (auto か none を指定可能)。l2 モードでは、ゲートウェイの IPv4 アドレス。
ipv4.host_tabl	integer	-	(メインのルーティングテーブルに加えて)IPv4 の静的ルートを追加する先のカスタムポリシー・ルーティングテーブル ID
ipv6.address	string	-	インスタンスに追加する IPv6 静的アドレスのカンマ区切りリスト (l2 モードでは、CIDR 形式か/64 のサブネットの単一アドレスで指定可能)
ipv6.gateway	string	auto (l3s), - (l2)	l3s モードでは、デフォルト IPv6 ゲートウェイを自動的に追加するかどうか (auto か none を指定可能)。l2 モードで、はゲートウェイの IPv6 アドレス。
ipv6.host_tabl	integer	-	(メインのルーティングテーブルに加えて)IPv6 の静的ルートを追加する先のカスタムポリシー・ルーティングテーブル ID
mode	string	l3s	IPVLAN のモード (l2 か l3s のいずれか)
mtu	integer	親の MTU	新しいインタフェースの MTU
name	string	カーネルが割り当て	インスタンス内部でのインタフェース名
queue.tx.length	integer	-	NIC の送信キューの長さ
parent	string	-	ホストデバイスの名前 (必須)
vlan	integer	-	アタッチ先の VLAN ID

nictype: p2p

注釈: この NIC タイプは nictype オプションでのみ選択できます。

p2p NIC は仮想デバイスペアを作成し、片方はインスタンス内に配置し、もう片方はホストに残します。

デバイスオプション

p2p タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
boot.priority	integer	-	VM のブート優先度 (高いほうが先にブート)
host_name	string	ランダムに割り当て	ホスト内でのインタフェースの名前
hwaddr	string	ランダムに割り当て	新しいインタフェースの MAC アドレス
ipv4.routes	string	-	ホスト上で NIC に追加する IPv4 静的ルートのカンマ区切りリスト
ipv6.routes	string	-	ホスト上で NIC に追加する IPv6 静的ルートのカンマ区切りリスト
limits.egress	string	-	外向きトラフィックの I/O 制限値 (さまざまな単位が使用可能、**ストレージとネットワーク制限の単位**参照)
limits.ingress	string	-	内向きトラフィックの I/O 制限値 (さまざまな単位が使用可能、**ストレージとネットワーク制限の単位**参照)
limits.max	string	-	内向きと外向きの両方のトラフィック I/O 制限値 (limits.ingress と limits.egress の両方を設定するのと同じ)
mtu	integer	カーネルが割り当て	新しいインタフェースの MTU
name	string	カーネルが割り当て	インスタンス内部でのインタフェース名

`nictype: routed`

注釈: この NIC タイプは `nictype` オプションでのみ選択できます。

`routed` NIC タイプはホストをインスタンスに接続する仮想デバイスペアを作成し、インスタンスが指定された親インタフェースのネットワークに参加できるように、静的ルートとプロキシ ARP/NDP エントリをセットアップします。コンテナでは仮想イーサネットデバイスペアを使用し、VM では TAP デバイスを使用します。

この NIC タイプは運用上は IPVLAN に似ていて、ブリッジを設定することなくホストの MAC アドレスを共用して、インスタンスが外部ネットワークに参加できるようにします。しかし、カーネルに IPVLAN サポートを必要としないことと、ホストとインスタンスが互いに通信できることが `ipvlan` とは異なります。

この NIC タイプは `netfilter` のルールを尊重し、ホストのルーティングテーブルを使ってパケットをルーティングしますので、ホストが複数のネットワークに接続している場合に役立ちます。

IP アドレス、ゲートウェイ、ルーティング イ

ンスタンスが起動する前に IP アドレスを (`ipv4.address` と `ipv6.address` の設定のいずれかあるいは両方を使って) 手動で指定する必要があります。

コンテナでは、NIC はホスト上に下記のリンクローカルゲートウェイ IP アドレスを設定し、それらをコンテナの NIC インタフェースのデフォルトゲートウェイに設定します。

169.254.0.1 fe80::1

VM では、ゲートウェイは手動か `cloud-init` のような仕組みを使って設定する必要があります。

注釈: お使いのコンテナイメージがインタフェースに対して DHCP を使うように設定されている場合、上記の自動的に追加される設定は削除される可能性が高いです。この場合、IP アドレスとゲートウェイを手動か `cloud-init` のような仕組みを使って設定する必要があります。

この NIC タイプはインスタンスの IP アドレス全てをインスタンスの `veth` インタフェースに向ける静的ルートをホスト上に設定します。

複数の IP アドレス そ

れぞれの NIC デバイスに複数の IP アドレスを追加できます。

しかし、代わりに複数の `routed` NIC インタフェースを使うほうが望ましいかもしれません。この場合、`ipv4.gateway` と `ipv6.gateway` の値を `none` に設定し、後続のインタフェースがデフォルトゲートウェイの衝突を避けるようにします。さらに、これらの後続のインタフェースに `ipv4.host_address` と `ipv6.host_address` を使って異なるホスト側のアドレスを指定することを検討してください。

親のインタフェース こ

の NIC は parent のネットワークインタフェースのセットがあってもなくても利用できます。

parent ネットワークインタフェースのセットがある場合、インスタンスの IP のプロキシ ARP/NDP エントリが親のインタフェースに追加され、インスタンスが親のインタフェースのネットワークにレイヤ 2 で参加できるようにします。

DNS

ネームサーバーは自動的には設定されないので、インスタンス内で設定する必要があります。このためには、以下の sysctl の設定をしてください。

- IPv4 アドレスを使用する場合

```
net.ipv4.conf.<parent>.forwarding=1
```

- IPv6 アドレスを使用する場合

```
net.ipv6.conf.all.forwarding=1
net.ipv6.conf.<parent>.forwarding=1
net.ipv6.conf.all.proxy_ndp=1
net.ipv6.conf.<parent>.proxy_ndp=1
```

デバイスオプション

routed タイプの NIC デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
gvrp	bool	false	GARP VLAN Registration Protocol を使って VLAN を登録する
host_name	string	ランダムに割り当て	ホスト内でのインタフェース名
hwaddr	string	ランダムに割り当て	新しいインタフェースの MAC アドレス
ipv4.address	string	-	インスタンスに追加する IPv4 静的アドレスのカンマ区切りリスト
ipv4.gateway	string	auto	自動的に IPv4 デフォルトゲートウェイを追加するかどうか (auto か none を指定可能)
ipv4.host_address	string	169.254.0.1	ホスト側の veth インタフェースに追加する IPv4 アドレス
ipv4.host_table	integer	-	(メインのルーティングテーブルに加えて)IPv4 の静的ルートを追加する先のカスタムポリシー・ルーティングテーブル ID
ipv4.neighbor_prol	bool	true	IP アドレスが利用可能か知るために親のネットワークを調べるかどうか
ipv4.routes	string	-	ホスト上で NIC に追加する IPv4 静的ルートのカンマ区切りリスト (L2 ARP/NDP プロキシを除く)
ipv6.address	string	-	インスタンスに追加する IPv6 静的アドレスのカンマ区切りリスト
ipv6.gateway	string	auto	自動的に IPv6 のデフォルトゲートウェイを追加するかどうか (auto か none を指定可能)
ipv6.host_address	string	fe80::1	ホスト側の veth インタフェースに追加する IPv6 アドレス
ipv6.host_table	integer	-	(メインのルーティングテーブルに加えて)IPv6 の静的ルートを追加する先のカスタムポリシー・ルーティングテーブル ID
ipv6.neighbor_prol	bool	true	IP アドレスが利用可能か知るために親のネットワークを調べるかどうか
ipv6.routes	string	-	ホスト上で NIC に追加する IPv6 静的ルートのカンマ区切りリスト (L2 ARP/NDP プロキシを除く)
limits.ingress	string	-	内向きトラフィックに対する bit/s での I/O 制限値 (さまざまな単位をサポート、**ストレージとネットワーク制限の単位**参照)
limits.egress	string	-	外向きトラフィックに対する bit/s での I/O 制限値 (さまざまな単位をサポート、**ストレージとネットワーク制限の単位**参照)
limits.max	string	-	内向きと外向き両方のトラフィックの I/O 制限値 (limits.ingress と limits.egress の両方を設定するのと同じ)
mtu	integer	親の MTU	新しいインタフェースの MTU
name	string	カーネルが割り当て	インスタンス内でのインタフェース名

bridge、macvlan、ipvlan を使った物理ネットワークへの接続

bridged、macvlan、ipvlan インタフェースタイプのいずれも、既存の物理ネットワークへ接続するために使用できます。

macvlan は、物理 NIC を効率的に分岐できます。つまり、物理 NIC からインスタンスで使える第 2 のインタフェースを取得できます。この方法はブリッジデバイスと仮想イーサネットデバイスペアの作成を不要にしますし、通常はブリッジよりも良いパフォーマンスが得られます。

macvlan の欠点は、macvlan はインスタンス自身と外部との間で通信はできますが、親デバイスとは通信できないことです。つまりインスタンスとホストが通信する必要がある場合は macvlan は使えません。

そのような場合は、bridge デバイスを選ぶのが良いでしょう。macvlan では使えない MAC フィルタリングと I/O 制限も使えます。

ipvlan は macvlan と同様ですが、フォークされたデバイスが静的に割り当てられた IP アドレスを持ち、ネットワーク上の親の MAC アドレスを受け継ぐ点が異なります。

MAAS を使った統合管理

もし、LXD ホストが接続されている物理ネットワークを MAAS を使って管理している場合で、インスタンスを直接 MAAS が管理するネットワークにアタッチしたい場合は、MAAS とやりとりをしてインスタンスをトラッキングするように LXD を設定できます。

そのためには、デーモンに対して、maas.api.url と maas.api.key を設定しなければなりません。そして、maas.subnet.ipv4 と maas.subnet.ipv6 の両方またはどちらかを、インスタンスもしくはプロファイルの nic エントリーに設定します。

これで、LXD は全てのインスタンスを MAAS に登録し、適切な DHCP リースと DNS レコードをインスタンスに与えます。

ipv4.address もしくは ipv6.address キーを NIC に設定した場合は、MAAS 上で静的な割り当てとして登録されます。

タイプ: disk

注釈: disk デバイスタイプはコンテナと VM の両方でサポートされます。コンテナと VM の両方でホットプラグをサポートします。

ディスクデバイスはインスタンスに追加のストレージを提供します。

コンテナにとっては、それらはインスタンス内の実質的なマウントポイントです (ホスト上の既存のファイルまたはディレクトリのバインドマウントとしてか、あるいは、ソースがブロックデバイスの場合は通常のマウントのマ

ウントポイント)。仮想マシンは 9p または virtiofs (使用可能な場合) を通してホスト側のマウントまたはディレクトリを共有するか、あるいはブロックベースのディスクに対する VirtIO ディスクとして共有します。

ディスクデバイスの種類

さまざまなソースからディスクデバイスを作成できます。
source オプションに指定する値によって、追加されるディスクデバイスのタイプが決まります:

ストレージボリューム 最

も一般的なタイプのディスクデバイスはストレージボリュームです。
ストレージボリュームを追加するには、デバイスの source としてその名前を指定します：

```
lxc config device add <instance_name> <device_name> disk pool=<pool_name> source=
↪<volume_name> [path=<path_in_instance>]
```

path はファイルシステムボリュームには必要ですが、ブロックボリュームには必要ありません。

また、lxc storage volume attach コマンドを使用して**インスタンスにカスタムストレージボリュームをアタッチする**することもできます。
どちらのコマンドも、ストレージボリュームをディスクデバイスとして追加するための同じメカニズムを使用します。

ホスト上のパス ホ

ストのパス（ファイルシステムまたはブロックデバイスのいずれか）をインスタンスと共有するには、ディスクデバイスとして追加し、source としてホストパスを指定します：

```
lxc config device add <instance_name> <device_name> disk source=<path_on_host>␣
↪[path=<path_in_instance>]
```

path はファイルシステムボリュームには必要ですが、ブロックデバイスには必要ありません。

Ceph RBD

LXD は、インスタンスの内部ファイルシステムを管理するために Ceph を使用できますが、既存の外部管理 Ceph RBD をインスタンスに使用したい場合は、次のコマンドで追加できます：

```
lxc config device add <instance_name> <device_name> disk source=ceph:<pool_name>/
↪<volume_name> ceph.user_name=<user_name> ceph.cluster_name=<cluster_name> [path=
↪<path_in_instance>]
```

path はファイルシステムボリュームには必要ですが、ブロックデバイスには必要ありません。

CephFS

LXD はインスタンスで内部のファイルシステムの管理に Ceph を使えますが、既存の外部で管理されている Ceph ファイルシステムをインスタンスで使用したい場合は、以下のコマンドで追加できます。

```
lxc config device add <instance_name> <device_name> disk source=cephfs:<fs_name>/
↪<path> ceph.user_name=<user_name> ceph.cluster_name=<cluster_name> path=<path_in_
↪instance>
```

ISO file
: 仮想マシンには ISO ファイルをディスクデバイスとして追加できます。ISO ファイルは VM 内部の ROM デバイスとして追加されます。

  このソースタイプは VM でのみ利用可能です。

  ISO ファイルを追加するには、そのファイルパスを `source` として指定します。

```
lxc config device add <instance_name> <device_name> disk source=<file_path_on_host>
```

VM cloud-init
: `cloud-init.vendor-data`、`cloud-init.user-data` 設定キー (**インスタンスオプション**参照) から `cloud-init` 設定の ISO イメージを生成し、仮想マシンにアタッチできます。

  このソースタイプは VM でのみ利用可能です。

  そのようなデバイスを追加するには、以下のコマンドを使用します。

```
lxc config device add <instance_name> <device_name> disk source=cloud-init:config
```

デバイスオプション

`disk` デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	必須	説明
boot.priorit	integer	-	no	VM のブート優先度 (高いほうが先にブート)
ceph.cluster	string	ceph	no	Ceph クラスタのクラスタ名 (Ceph か CephFS のソースには必須)
ceph.user_na	string	admir	no	Ceph クラスタのユーザ名 (Ceph か CephFS のソースには必須)
io.cache	string	none	no	VM のみ: デバイスのキャッシュモードを上書きする (none, writeback または unsafe)
limits.max	string	-	no	読み取りと書き込み両方の byte/s か IOPS による I/O 制限 (limits.read と limits.write の両方を設定するのと同じ)
limits.read	string	-	no	byte/s(さまざまな単位が使用可能、**ストレージとネットワーク制限の単位**参照) もしくは IOPS(あとに iops と付けなければなりません) で指定する読み込みの I/O 制限値 - *I/O* **制限値の設定** も参照
limits.write	string	-	no	byte/s(さまざまな単位が使用可能、**ストレージとネットワーク制限の単位**参照) もしくは IOPS(あとに iops と付けなければなりません) で指定する書き込みの I/O 制限値 - *I/O* **制限値の設定** も参照
path	string	-	yes	ディスクをマウントするインスタンス内のパス (コンテナのみ)
pool	string	-	no	ディスクデバイスが属するストレージプール (LXD が管理するストレージボリュームにのみ適用可能)
propaga	string	-	no	バインドマウントをインスタンスとホストでどのように共有するかを管理する (private (デフォルト), shared, slave, unbindable, rshared, rslave, runbindable, rprivate のいずれか。完全な説明は Linux Kernel の文書 shared subtree をご覧ください)
raw.mount.options	string	-	no	ファイルシステム固有のマウントオプション
readonl	bool	false	no	マウントを読み込み専用とするかどうかを制御
recursi	bool	false	no	ソースパスを再帰的にマウントするかどうかを制御
require	bool	true	no	ソースが存在しないときに失敗とするかどうかを制御
shift	bool	false	no	ソースの UID/GID をインスタンスにマッチするように変換させるためにオーバーレイの shift を設定するか (コンテナのみ)
size	string	-	no	byte(さまざまな単位が使用可能、 **ストレージとネットワーク制限の単位** 参照) で指定するディスクサイズ。rootfs (/) でのみサポートされます
size.state	string	-	no	上の size と同じですが、仮想マシン内のランタイム状態を保存するために使われるファイルシステムボリュームに適用されます
source	string	-	yes	ファイルシステムまたはブロックデバイスのソース (詳細は**ディスクデバイスの種類**参照)

タイプ: unix-char

注釈: unix-char デバイスタイプはコンテナでサポートされます。ホットプラグをサポートします。

Unix キャラクタデバイスは、指定したキャラクタデバイスをインスタンス内の (/dev 以下の) デバイスとして出現させます。そのデバイスから読み取りやデバイスへ書き込みができます。

デバイスオプション

unix-char デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
gid	int	0	インスタンス内のデバイス所有者の GID
major	int	ホスト上のデバイス	デバイスのメジャー番号
minor	int	ホスト上のデバイス	デバイスのマイナー番号
mode	int	0660	インスタンス内のデバイスのモード
path	string	-	インスタンス内のパス (source と path のどちらかを設定しなければいけません)
required	bool	true	このデバイスがインスタンスの起動に必要かどうか (ホットプラグ参照)
source	string	-	ホスト上のパス (source と path のどちらかを設定しなければいけません)
uid	int	0	インスタンス内のデバイス所有者の UID

ホットプラグ

をサポートします。

```
Unix ブロックデバイスは、指定したブロックデバイスをインスタンス内の (`/dev`以下の) デバイスとして出現させます。
そのデバイスから読み取りやデバイスへ書き込みができます。

## デバイスオプション

`unix-block`デバイスには以下のデバイスオプションがあります。
```

(次のページに続く)

(前のページからの続き)

キー	型	デフォルト値	説明
`gid`	int	`0`	インスタンス内のデバイス所有者の GID
`major`	int	ホスト上のデバイス	デバイスのメジャー番号
`minor`	int	ホスト上のデバイス	デバイスのマイナー番号
`mode`	int	`0660`	インスタンス内のデバイスのモード
`path`	string	-	インスタンス内のパス (`source` と `path` のどちらかを設定しなければいけません)
`required`	bool	`true`	このデバイスがインスタンスの起動に必要かどうか (↪{ref}`devices-unix-block-hotplugging` 参照)
`source`	string	-	ホスト上のパス (`source` と `path` のどちらかを設定しなければいけません)
`uid`	int	`0`	インスタンス内のデバイス所有者の UID

(devices-unix-block-hotplugging)=
ホットプラグ

ホットプラグは `required=false` を設定しデバイスの `source` オプションを指定した場合に有効になります。

この場合、デバイスはホスト上で出現したときに、コンテナの起動後であっても、自動的にコンテナにパススルーされます。
ホストシステムからデバイスが消えると、コンテナからも消えます。

タイプ: unix-block

注釈: unix-block デバイスタイプはコンテナでサポートされます。ホットプラグをサポートします。

Unix ブロックデバイスは、指定したブロックデバイスをインスタンス内の (/dev 以下の) デバイスとして出現させます。そのデバイスから読み取りやデバイスへ書き込みができます。

デバイスオプション

unix-block デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
gid	int	0	インスタンス内のデバイス所有者の GID
major	int	ホスト上のデバイス	デバイスのメジャー番号
minor	int	ホスト上のデバイス	デバイスのマイナー番号
mode	int	0660	インスタンス内のデバイスのモード
path	string	-	インスタンス内のパス (source と path のどちらかを設定しなければいけません)
required	bool	true	このデバイスがインスタンスの起動に必要かどうか (**ホットプラグ**参照)
source	string	-	ホスト上のパス (source と path のどちらかを設定しなければいけません)
uid	int	0	インスタンス内のデバイス所有者の UID

ホットプラグ

ホットプラグは required=false を設定しデバイスの source オプションを指定した場合に有効になります。

この場合、デバイスはホスト上で出現したときに、コンテナの起動後であっても、自動的にコンテナにパススルーされます。ホストシステムからデバイスが消えると、コンテナからも消えます。

タイプ: usb

注釈: usb デバイスタイプはコンテナと VM の両方でサポートされます。コンテナと VM の両方でホットプラグをサポートします。

USB デバイスは、指定された USB デバイスをインスタンスに出現させます。パフォーマンスの問題のため、高スループットまたは低レイテンシを要求するデバイスの使用は避けてください。

コンテナでは、(/dev/bus/usb にある)libusb デバイスのみがインスタンスに渡されます。この方法はユーザスペースのドライバを持つデバイスで機能します。専用のカーネルドライバを必要とするデバイスは、代わりに *unix-char* **デバイス**か *unix-hotplug* **デバイス**を使用してください。

仮想マシンでは、USB デバイス全体がパススルーされますので、あらゆる USB デバイスがサポートされます。デバイスがインスタンスに渡されると、ホストからは消失します。

デバイスオプション

usb デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
gid	int	0	コンテナのみ: インスタンス内のデバイス所有者の GID
mode	int	0660	コンテナのみ: インスタンス内のデバイスのモード
product	string	-	USB デバイスのプロダクト ID
required	bool	false	このデバイスがインスタンスの起動に必要かどうか (デフォルトは false で、すべてのデバイスがホットプラグ可能です)
uid	int	0	コンテナのみ: インスタンス内のデバイス所有者の UID
vendorid	string	-	USB デバイスのベンダー ID

タイプ: gpu

GPU デバイスは、指定の GPU デバイスをインスタンス内に出現させます。

注釈: コンテナでは、gpu デバイスは同時に複数の GPU にマッチングさせることができます。VM では、各デバイスは 1 つの GPU にしかマッチできません。

以下のタイプの GPU が gputype デバイスオプションを使って追加できます。

- *physical* (コンテナと VM): GPU 全体をインスタンスにパススルーします。gputype が指定されない場合これがデフォルトです。
- *mdev* (VM のみ): 仮想 GPU を作成しインスタンスにパススルーします。
- *mig* (コンテナのみ): MIG(Multi-Instance GPU) を作成しインスタンスにパススルーします。
- *sriov* (VM のみ): SR-IOV を有効にした GPU の仮想ファンクション (virtual function) をインスタンスに与えます。

利用可能なデバイスオプションは GPU タイプごとに異なり、以下のセクションの表に一覧表示されます。

gputype: physical

注釈: physical GPU タイプはコンテナと VM の両方でサポートされます。ホットプラグはコンテナのみでサポートし、VM ではサポートしません。

physical GPU デバイスは GPU 全体をインスタンスにパススルーします。

デバイスオプション

physical タイプのデバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
gid	int	0	インスタンス (コンテナのみ) 内のデバイス所有者の GID
id	string	-	GPU デバイスの DRM カード ID
mode	int	0660	インスタンス (コンテナのみ) 内のデバイスのモード
pci	string	-	GPU デバイスの PCI アドレス
productid	string	-	GPU デバイスのプロダクト ID
uid	int	0	インスタンス (コンテナのみ) 内のデバイス所有者の UID
vendorid	string	-	GPU デバイスのベンダー ID

gputype: mdev

注釈: mdev GPU タイプは VM でのみサポートされます。ホットプラグはサポートしていません。

mdev GPU デバイスは仮想 GPU を作成しインスタンスにパススルーします。利用可能な mdev プロファイルの一覧は lxc info --resources を実行すると確認できます。

デバイスオプション

mdev タイプのデバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
id	string	-	GPU デバイスの DRM カード ID
mdev	string	-	使用する mdev プロファイル (必須 - 例:i915-GVTg_V5_4)
pci	string	-	GPU デバイスの PCI アドレス
productid	string	-	GPU デバイスのプロダクト ID
vendorid	string	-	GPU デバイスのベンダー ID

gputype: mig

注釈: mig GPU タイプはコンテナでのみサポートされます。ホットプラグはサポートしていません。

mig GPU デバイスは MIG コンピュートインスタンスを作成しインスタンスにパススルーします。現状これは NVIDIA MIG を事前に作成しておく必要があります。

デバイスオプション

mig タイプのデバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
id	string	-	GPU デバイスの DRM カード ID
mig.ci	int	-	既存の MIG コンピュートインスタンス ID
mig.gi	int	-	既存の MIG GPU インスタンス ID
mig.uuid	string	-	既存の MIG デバイス UUID(MIG-接頭辞は省略可)
pci	string	-	GPU デバイスの PCI アドレス
productid	string	-	GPU デバイスのプロダクト ID
vendorid	string	-	GPU デバイスのベンダー ID

mig.uuid(NVIDIA drivers 470+) か、mig.ci と mig.gi(古い NVIDIA ドライバ) の両方を設定する必要があります。

gputype: sriov

注釈: sriov GPU タイプは VM でのみサポートされます。ホットプラグはサポートしていません。

sriov GPU デバイスは SR-IOV が有効な GPU の仮想ファンクション (virtual function) をインスタンスにパススルーします。

デバイスオプション

sriov タイプのデバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
id	string	-	GPU デバイスの DRM カード ID
pci	string	-	親 GPU デバイスの PCI アドレス
productid	string	-	親 GPU デバイスのプロダクト ID
vendorid	string	-	親 GPU デバイスのベンダー ID

タイプ: infiniband

注釈: infiniband デバイスタイプはコンテナと VM の両方でサポートされます。ホットプラグはコンテナのみでサポートし、VM ではサポートしません。

LXD では、InfiniBand デバイスに対する 2 種類の異なったネットワークタイプが使えます。

- physical: ホストの物理デバイスをインスタンスにパススルーします。対象のデバイスはホスト上では見えなくなり、インスタンス内に出現します。
- sriov: SR-IOV が有効な物理ネットワークデバイスの仮想ファンクション (virtual function) をインスタンスにパススルーします。

  注釈: InfiniBand デバイスは SR-IOV をサポートしますが、他の SR-IOV が有効なデバイスと異なり、InfiniBand は SR-IOV モードの動的なデバイスの作成をサポートしません。このため、対応するカーネルモジュールを設定することで仮想ファンクションの数を事前に設定する必要があります。

physical な infiniband デバイスを作成するには、以下のコマンドを使用します。

```
lxc config device add <instance_name> <device_name> infiniband nictype=physical parent=
↪<device>
```

sriov の infiniband デバイスを作成するには、以下のコマンドを使用します。

```
lxc config device add <instance_name> <device_name> infiniband nictype=sriov parent=
↪<sriov_enabled_device>
```

デバイスオプション

infiniband デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	必須	説明
hwadc	string	ランダムに割り当て	no	新しいインタフェースの MAC アドレス。20 バイト全てを指定するか短い 8 バイト (この場合親デバイスの最後の 8 バイトだけを変更) のどちらかを設定可能
mtu	integer	親の MTU	no	新しいインタフェースの MTU
name	string	カーネルが割り当て	no	インスタンス内部でのインタフェース名
nicty	string	-	yes	デバイスタイプ (physical か sriov のいずれか)
parer	string	-	yes	ホスト上のデバイスまたはブリッジの名前

タイプ: proxy

注釈: proxy デバイスタイプはコンテナ (NAT と非 NAT モード) と VM(NAT モードのみ) でサポートされます。コンテナと VM の両方でホットプラグをサポートします。

プロキシデバイスにより、ホストとインスタンス間のネットワーク接続を転送できます。この方法で、ホストのアドレスの一つに到達したトラフィックをインスタンス内のアドレスに転送したり、その逆でインスタンス内にアドレスを持ちホストを通して接続することができます。

NAT **モード**では、プロキシデバイスを TCP と UDP のプロキシに使用することができます。NAT モードではない場合、Unix ソケット間のトラフィックをプロキシすることもできます（これは例えば、コンテナからホストシステムへのグラフィカルな GUI やオーディオトラフィックを転送するのに便利です)。また、プロトコル間でもプロ

キシすることができます（例えば、ホストシステム上に TCP リスナーを設置し、そのトラフィックをコンテナ内の Unix ソケットに転送することができます）。

利用できる接続タイプは次の通りです。

- tcp <-> tcp
- udp <-> udp
- unix <-> unix
- tcp <-> unix
- unix <-> tcp
- udp <-> tcp
- tcp <-> udp
- udp <-> unix
- unix <-> udp

proxy デバイスを追加するには、以下のコマンドを使用します。

```
lxc config device add <instance_name> <device_name> proxy listen=<type>:<addr>:<port>[-
↪<port>][,<port>] connect=<type>:<addr>:<port> bind=<host/instance_name>
```

NAT モード

プロキシデバイスは NAT モード (nat=true) もサポートします。NAT モードではパケットは別の接続を通してプロキシされるのではなく NAT を使ってフォワードされます。これはターゲットの送り先が HAProxy の PROXY プロトコル (非 NAT モードでプロキシデバイスを使う場合はこれはクライアントアドレスを渡す唯一の方法です) をサポートする必要なく、クライアントのアドレスを維持できるという利点があります。

しかし、NAT モードはインスタンスが稼働しているホストがゲートウェイの場合（例えば lxdbr0 を使用しているケース）のみサポートされます。

NAT モードでサポートされる接続のタイプは以下の通りです。

- tcp <-> tcp
- udp <-> udp

プロキシデバイスを nat=true に設定する際は、以下のようにターゲットのインスタンスが NIC デバイス上に静的 IP を持つようにする必要があります。

IP アドレスを指定する

インスタンス NIC に静的 IP を設定するには、以下のコマンドを使用します。

```
lxc config device set <instance_name> <nic_name> ipv4.address=<ipv4_address> ipv6.
↪address=<ipv6_address>
```

静的な IPv6 アドレスを設定するためには、親のマネージドネットワークは `ipv6.dhcp.stateful` を有効にする必要があります。

IPv6 アドレスを設定する場合は以下のような角括弧の記法を使います。例えば以下のようにします。

```
connect=tcp:[2001:db8::1]:80
```

connect のアドレスをワイルドカード (IPv4 では 0.0.0.0、IPv6 では [::] にします) に設定することで、接続アドレスをインスタンスの IP アドレスになるように指定できます。

注釈: listen のアドレスも非 NAT モードではワイルドカードのアドレスが使用できます。しかし、NAT モードを使う際は LXD ホスト上の IP アドレスを指定する必要があります。

デバイスオプション

`proxy` デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	必須	説明
bind	string	host	no	どちら側にバインドするか (host/instance)
connect	string	-	yes	接続するアドレスとポート (<type>:<addr>:<port>[-<port>][,<port>])
gid	int	0	no	listen する Unix ソケットの所有者の GID
listen	string	-	yes	バインドし、接続を待ち受けるアドレスとポート (<type>:<addr>:<port>[-<port>][,<port>])
mode	int	0644	no	listen する Unix ソケットのモード
nat	bool	false	no	NAT 経由でプロキシを最適化するかどうか (インスタンスの NIC が静的 IP を持つ必要あり)
proxy_proto	bool	false	no	送信者情報を送信するのに HAProxy の PROXY プロトコルを使用するかどうか
security.gid	int	0	no	特権を落とす GID
security.uid	int	0	no	特権を落とす UID
uid	int	0	no	listen する Unix ソケットの所有者の UID

タイプ: unix-hotplug

注釈: unix-hotplug デバイスタイプはコンテナでサポートされます。ホットプラグをサポートします。

Unix ホットプラグデバイスは、指定した Unix デバイスをインスタンス内の (/dev 以下の) デバイスとして出現させます。デバイスがホストシステム上にある場合は、デバイスから読み取りやデバイスへ書き込みができます。

実装はホスト上で稼働する systemd-udev に依存します。

デバイスオプション

unix-hotplug デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	説明
gid	int	0	インスタンス内でのデバイスオーナーの GID
mode	int	0660	インスタンス内でのデバイスのモード
product	string	-	Unix デバイスの製品 ID
require	bool	false	このデバイスがインスタンスを起動するのに必要かどうか (デフォルトは false で、全てのデバイスはホットプラグ可能です)
uid	int	0	インスタンス内でのデバイスオーナーの UID
vendori	string	-	Unix デバイスのベンダー ID

タイプ: tpm

注釈: tpm デバイスタイプは、コンテナと VM の両方でサポートされています。ただし、コンテナではホットプラグがサポートされていますが、VM ではサポートされていません。

TPM デバイスは、TPM（Trusted Platform Module）エミュレータへのアクセスを有効にします。

TPM デバイスは、ブートプロセスを検証し、ブートチェーンのステップが改ざんされていないことを確認するために使用できます。また、暗号化キーを安全に生成および保存することもできます。

LXD は、TPM 2.0 をサポートするソフトウェア TPM を使用します。
コンテナの主な使用例は、証明書のシールで、これによりキーがコンテナの外部に保存され、攻撃者がそれらを取得することがほぼ不可能になります。
仮想マシンでは、TPM を証明書のシールに使用するだけでなく、ブートプロセスの検証にも使用できます。これにより、例えば、Windows BitLocker と互換性のあるフルディスク暗号化が可能になります。

デバイスオプション

tpm デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	必須	説明
path	string	-	コンテナでは必須	コンテナのみ: インスタンス内でのパス (例:/dev/tpm0)
pathrm	string	-	コンテナでは必須	コンテナのみ: インスタンス内でのリソースマネージャのパス (例:dev/tpmrm0)

タイプ: pci

注釈: pci デバイスタイプは VM でサポートされます。ホットプラグはサポートされません。

PCI デバイスは生の PCI デバイスをホストから仮想マシンにパススルーするために使用されます。

これらや主にサウンドカードやビデオキャプチャーカードのような特別な単一機能の PCI カードに使われることを意図しています。理論上は、GPU やネットワークカードなどより高度な PCI デバイスも使用できますが、通常はそれらのデバイスのために LXD が提供する個別のデバイスタイプ (*gpu* **デバイス**や *nic* **デバイス**) を使うほうがより便利です。

デバイスオプション

pci デバイスには以下のデバイスオプションがあります。

キー	型	デフォルト値	必須	説明
address	string	-	yes	デバイスの PCI アドレス

ストレージとネットワーク制限の単位

バイト数とビット数を表す値は全ていくつかの単位を使用し特定の制限がどういう値かをより理解しやすいようにできます。

10 進と 2 進 (kibi) の単位の両方がサポートされており、後者は主にストレージの制限に有用です。

現在サポートされているビットの単位の完全なリストは以下の通りです。

- bit (1)
- kbit (1000)
- Mbit (1000^2)
- Gbit (1000^3)
- Tbit (1000^4)
- Pbit (1000^5)
- Ebit (1000^6)
- Kibit (1024)
- Mibit (1024^2)

- Gibit (1024^3)
- Tibit (1024^4)
- Pibit (1024^5)
- Eibit (1024^6)

現在サポートされているバイトの単位の完全なリストは以下の通りです。

- B または bytes (1)
- kB (1000)
- MB (1000^2)
- GB (1000^3)
- TB (1000^4)
- PB (1000^5)
- EB (1000^6)
- KiB (1024)
- MiB (1024^2)
- GiB (1024^3)
- TiB (1024^4)
- PiB (1024^5)
- EiB (1024^6)

3.4.13 コンテナ実行環境

LXD は実行する全てのコンテナに一貫性のある環境を提供しようとします。

正確な環境はカーネルの機能やユーザーの設定によって若干異なりますが、それ以外は全てのコンテナに対して同一です。

ファイルシステム

LXD は使用するどのイメージから生成する新規のコンテナは少なくとも以下のルートレベルのディレクトリが存在することを前提とします。

- /dev (空のディレクトリ)
- /proc (空のディレクトリ)
- /sbin/init (実行ファイル)
- /sys (空のディレクトリ)

デバイス

LXD のコンテナは tmpfs ファイルシステムをベースとする最低限で一時的な/dev を持ちます。これは tmpfs であって devtmpfs ファイルシステムではないので、デバイスノードは手動で作成されたときのみ現れます。

デバイスノードの標準セットでは以下のデバイスが自動的にセットアップされます。

- /dev/console
- /dev/fd
- /dev/full
- /dev/log
- /dev/null
- /dev/ptmx
- /dev/random
- /dev/stdin
- /dev/stderr
- /dev/stdout
- /dev/tty
- /dev/urandom
- /dev/zero

標準セットのデバイスに加えて、以下のデバイスも利便性のためにセットアップされます。

- /dev/fuse
- /dev/net/tun

- /dev/mqueue

ネットワーク

LXD コンテナはネットワークデバイスをいくつでもアタッチできます。これらの名前はユーザーにオーバーライドされない限りは ethX で X は連番です。

コンテナからホストへのコミュニケーション

LXD は/dev/lxd/sock にソケットをセットアップし、コンテナ内の root ユーザーはこれを使ってホストの LXD とコミュニケーションできます。

API ドキュメントは**インスタンス～ホスト間の通信**を参照してください。

マウント

以下のマウントがデフォルトでセットアップされます。

- /proc (proc)
- /sys (sysfs)
- /sys/fs/cgroup/* (cgroupfs) (cgroup namespace サポートを欠くカーネルの場合のみ)

以下のパスがホスト上に存在する場合は自動的にマウントされます。

- /proc/sys/fs/binfmt_misc
- /sys/firmware/efi/efivars
- /sys/fs/fuse/connections
- /sys/fs/pstore
- /sys/kernel/debug
- /sys/kernel/security

これらのパスを引き渡す理由は、これらがマウントされているか、コンテナ内でマウント可能であることが必要とされているレガシーな init システムのためです。

これらのパスほとんどは非特権コンテナ内からは書き込み可能ではなく (あるいは読み取り可能ですらなく)、特権コンテナ内では LXD の AppArmor ポリシーによってブロックされます。

LXCFS

ホストに LXCFS がある場合は、コンテナ用に自動的にセットアップされます。

これは通常いくつかの/proc ファイルになり、それらは bind mount を通してオーバーライドされます。古いカーネルでは/sys/fs/cgroup の仮想バージョンも LXCFS によりセットアップされるかもしれません。

PID1

LXD は何であれ/sbin/init に置かれているものをコンテナの初期プロセス (PID 1) として起動します。このバイナリは親が変更されたプロセス (訳注: ゾンビプロセスなど) の処理を含めて適切な init システムとして振る舞う必要があります。

LXD がコンテナの PID1 とコミュニケーションするのは以下の 2 つのシグナルだけです。

- SIGINT コンテナのリブートをトリガーする
- SIGPWR (かあるいは SIGRTMIN+3) コンテナのクリーンなシャットダウンをトリガーする

PID1 の初期環境は container=lxc 以外は空です。init システムは container=lxc をランタイムの検出 (訳注: lxc で動いていることを知る) に使用できます。

デフォルトの 3 個 (訳注: stdin, stdout, stderr) より上の全てのファイルディスクリプタは PID1 が起動される前に閉じられます。

3.5 イメージ

3.5.1 イメージについて

LXD はイメージをベースとしたワークフローを使用します。各インスタンスはイメージをベースとしています。イメージは基礎となるオペレーティングシステム (例えば、Linux ディストリビューション) と LXD に関連するいくつかの情報を含みます。

イメージはリモートのイメージストア (概要は**リモートイメージサーバー**参照) から利用可能ですが、既存のインスタンスや rootfs イメージをベースにして、独自のイメージを作成できます。

リモートサーバーからローカルのイメージストアにイメージをコピーしたり、ローカルのイメージをリモートサーバーにコピーできます。ローカルのイメージをリモートのインスタンスを作るのに使うこともできます。

各イメージはフィンガープリント (SHA256) で識別されます。イメージを管理しやすくするために、LXD では各イメージに 1 つ以上のエイリアスを定義できます。

キャッシュ

リモートのイメージからインスタンスを作成する際、LXD はイメージをダウンロードしローカルにキャッシュします。イメージはローカルのイメージストアに cached フラグをセットして保管されます。イメージは以下のいずれかが発生するまでは非公開のイメージとしてローカルに保持されます。

- *images.remote_cache_expiry* で指定された日数の間新しいインスタンスを作成するのにイメージが使われなかった。
- イメージの有効期限 (イメージのプロパティの 1 つ。どのように変更するかの情報は**イメージのプロパティを編集する**参照) に達した。

LXD はイメージから新しいインスタンスが起動される度にイメージの `last_used_at` プロパティを更新することで、イメージの利用状況を記録しています。

自動更新

LXD はリモートサーバーからのイメージを自動的に最新に更新します。

注釈: エイリアスを指定して取得したイメージだけが更新されます。フィンガープリントを指定してイメージを取得した場合は、その特定のイメージバージョンを要求したことになります。

自動更新が有効になるかどうかはイメージをどのようにダウンロードしたかに依存します。

- インスタンス作成時にイメージがダウンロードとキャッシュされた場合は、ダウンロード時に *images.auto_update_cached* が `true` に設定されていれば、自動的に更新されます。
- イメージがリモートサーバーから `lxc image copy` コマンドでコピーされた場合は、`--auto-update` フラグが指定されていた場合のみ自動的に更新されます。

イメージのこの挙動は *auto_update* **プロパティを編集**することで変更できます。

起動時と *images.auto_update_interval* の間隔 (デフォルトでは 6 時間ごと) を過ぎるたびに、LXD デーモンは自動更新とマークされコピー元のサーバーが記録されたストア内の全てのイメージについてより新しいバージョンがあるかをチェックします。

新しいイメージが見つかったら、イメージ・ストアにダウンロードされます。その後古いイメージを指していたエイリアスは新しいイメージを指すように変更され、古いイメージはストアから削除されます。

インスタンスの生成が遅くならないようにするため、LXD はキャッシュされたイメージからインスタンスを作成する際に新しいバージョンが利用可能かをチェックしません。これはイメージが次の更新期間で更新されるまでの間は、新しく作成するインスタンスにイメージの古いバージョンが使われるかもしれないことを意味します。

特別なイメージプロパティ

プレフィックス `requirements` で始まるイメージプロパティ (例:`requirements.XYZ`) は LXD がホストシステムと当該イメージで生成されるインスタンスの互換性を判断するために使用されます。これらの互換性がない場合には、LXD はそのインスタンスを起動しません。

以下の要件がサポートされています。

キー	タイプ	既定値	説明
`requirements.secureboot`	string	-	`false` に設定すると、イメージがセキュアブートで起動しないことを示します。
`requirements.cgroup`	string	-	`v1` に設定されている場合、ホストで `CGroupV1` が実行されている必要があることを示します。

3.5.2 リモートイメージを使用するには

`lxc` CLI コマンドはいくつかのリモートイメージサーバーを初期設定されています。概要は**リモートイメージサーバー**を参照してください。

設定されたリモートを一覧表示する

設定された全てのリモートサーバーを見るには、以下のコマンドを入力します。

```
lxc remote list
```

simple streams 形式を使用するリモートサーバーは純粋なイメージサーバーです。`lxd` 形式を使用するサーバーは LXD サーバーであり、イメージサーバーだけとして稼働しているか、通常の LXD サーバーとして稼働するのに加えて追加のイメージを提供しているかのどちらかです。詳細は**リモートサーバータイプ**を参照してください。

リモート上の利用可能なイメージを一覧表示する

サーバー上の全てのリモートイメージを一覧表示するには、以下のコマンドを入力します。

```
lxc image list <remote>:
```

結果をフィルタできます。手順は**利用可能なイメージをフィルタする**を参照してください。

リモートサーバーを追加する

どのようにリモートを追加するかはサーバーが使用しているプロトコルに依存します。

simple streams サーバーを追加する

simple streams サーバーをリモートとして追加するには、以下のコマンドを入力します。

```
lxc remote add <remote_name> <URL> --protocol=simplestreams
```

URL は HTTPS でなければなりません。

リモートの LXD サーバーを追加する

LXD サーバーをリモートして追加するには、以下のコマンドを入力します。

```
lxc remote add <remote_name> <IP|FQDN|URL> [flags]
```

認証方法によっては固有のフラグが必要です (例えば、Candid 認証では `lxc remote add <remote_name> <IP|FQDN|URL> --auth-type=candid` を使います)。詳細は *LXD* **サーバーでの認証**と**リモート** *API* **認証**を参照してください。

例えば、IP アドレスを指定してリモートを追加するには以下のコマンドを入力します。

```
lxc remote add my-remote 192.0.2.10
```

リモートサーバーのフィンガープリントを確認するプロンプトが表示され、リモートで使用している認証方法によってパスワードまたはトークンの入力を求められます。

イメージを参照する

イメージを参照するには、リモートとイメージのエイリアスまたはフィンガープリントをコロンで区切って指定します。例:

```
images:ubuntu/22.04
ubuntu:22.04
local:ed7509d7e83f
```

デフォルトのリモートを選択する

リモート名前を指定せずにイメージ名だけ指定すると、デフォルトのイメージサーバーが使用されます。

どのサーバーがデフォルトのイメージサーバーと設定されているか表示するには、以下のコマンドを入力します。

```
lxc remote get-default
```

別のリモートをデフォルトのイメージサーバーに選択するには、以下のコマンドを入力します。

```
lxc remote switch <remote_name>
```

3.5.3 イメージを管理するには

イメージを扱う際は、利用可能なイメージについての情報を調べたりプロパティを表示・編集したり、特定のイメージを参照するエイリアスを設定したりできます。またイメージをファイルにエクスポートすることもできます。これは他のマシンに**イメージをコピーしたりインポート**するのに便利です。

利用可能なイメージを一覧表示する

サーバー上の全てのイメージを一覧表示するには、以下のコマンドを入力します。

```
lxc image list [<remote>:]
```

リモートを指定しない場合は、**デフォルトリモート**が使用されます。

利用可能なイメージをフィルタする

表示される結果をフィルタするには、コマンドの後にエイリアスかフィンガープリントの一部を指定します。例えば、全ての Debian のイメージを表示するには、以下のコマンドを入力します。

```
lxc image list images: debian
```

複数のフィルタも指定可能です。例えば、全ての 64-bit の Debian のイメージを表示するには、以下のコマンドを入力します。

```
lxc image list images: debian amd64
```

エイリアスとフィンガープリント以外のプロパティをフィルタするには、<key>=<value>形式でフィルタを指定します。例えば以下のようにします。

```
lxc image list images: debian architecture=x86_64
```

イメージの情報を表示する

イメージの情報を表示するには、以下のコマンドを入力します。

```
lxc image info <image_ID>
```

イメージの ID としては、イメージのエイリアスかフィンガープリントを指定できます。リモートのイメージでは、リモートサーバーを忘れずに含めてください (例えば `images:ubuntu/22.04`)。

イメージのプロパティだけを表示するには、以下のコマンドを入力します。

```
lxc image show <image_ID>
```

また、イメージの (`properties` キー配下にある) 特定のプロパティは以下のコマンドで表示できます。

```
lxc image get-property <image_ID> <key>
```

例えば、公式の Ubuntu 22.04 イメージのリリース名を表示するには、以下のコマンドを入力します。

```
lxc image get-property ubuntu:22.04 release
```

イメージのプロパティを編集する

`properties` キー配下にあるイメージの特定のプロパティを設定するには、以下のコマンドを入力します。

```
lxc image set-property <image_ID> <key>
```

注釈: これらのプロパティはイメージについての情報を伝えるために使用できます。これらはいかなる方法によっても LXD の挙動を変更はしません。

トップレベルのプロパティを含むイメージのプロパティ全体を編集するには、以下のコマンドを入力します。

```
lxc image edit <image_ID>
```

イメージのエイリアスを設定する

イメージのエイリアスを設定すると、イメージを参照するのがより容易になり便利かもしれません。というのはエイリアスを覚えるほうがフィンガープリントを覚えるより通常容易だからです。しかし、もっとも重要なのは、エイリアスは別のイメージを指すように変更できるので、常に最新のイメージ (例えば最新のリリースバージョン) を提供するエイリアスを作れることです。

イメージ一覧の中に既存のエイリアスの例を見ることができます。完全なリストを見るには、以下のコマンドを入力します。

```
lxc image alias list
```

イメージを**コピーやインポート**または**発行**する際には、直接イメージにエイリアスを割り当てることができます。代わりに、以下のコマンドを入力してエイリアスを作成することもできます。

```
lxc image alias create <alias_name> <image_fingerprint>
```

エイリアスは削除することもできます。

```
lxc image alias delete <alias_name>
```

エイリアスをリネームするには、以下のコマンドを入力します。

```
lxc image alias rename <alias_name> <new_alias_name>
```

エイリアスの名前を維持したまま、別の (例えば、より新しいバージョンの) イメージにエイリアスを向けたい場合は、既存のエイリアスを削除して新しいエイリアスを作成する必要があります。

イメージをファイルにエクスポートする

イメージはローカルサーバーかリモートの LXD サーバーのイメージストアに保管されます。ですが、イメージをファイルにエクスポートすることができます。この方法はイメージファイルをバックアップしたり、エアギャップされた (訳注: セキュリティを高めるために外部ネットワークから遮断された) 環境にイメージを持っていくのに便利です。

コンテナイメージをファイルにエクスポートするには、以下のコマンドを入力します。

```
lxc image export [<remote>:]<image> [<output_directory_path>]
```

仮想マシンイメージをファイルにエクスポートするには、`--vm` フラグを追加します。

```
lxc image export [<remote>:]<image> [<output_directory_path>] --vm
```

イメージに使用されるファイル構造の説明については**イメージ形式**を参照してください。

3.5.4 イメージをコピーやインポートするには

イメージをイメージストアに追加するには、他のサーバーからコピーすることもできますし、ファイル (ローカルのファイルまたはウェブサーバー上のファイル) からインポートすることもできます。

リモートからイメージをコピーする

あるサーバーから別のサーバーにイメージをコピーするには、以下のコマンドを入力します。

```
lxc image copy [<source_remote>:]<image> <target_remote>:
```

注釈: イメージをローカルのイメージストアにコピーするには、コピー先のリモートに `local:` と指定します。

全ての利用可能なフラグの一覧は `lxc image copy --help` を参照してください。最も重要なものは以下の通りです。

`--alias`
: イメージのコピーに割り当てるエイリアス。

`--copy-aliases`
: コピー元のイメージが持つエイリアスをコピーする。

`--auto-update`
: 元のイメージが更新されたらコピーも更新する。

`--vm`
: エイリアスからコピーする際、仮想マシンを作成するのに使えるイメージをコピーする。

ファイルからイメージをインポートする

要求される**イメージ形式**を使用するイメージファイルを持っていれば、イメージストアにインポートできます。

そのようなイメージファイルを取得する方法はいくつかあります。

- 既存のイメージをエクスポートする (**イメージをファイルにエクスポートする**参照)
- `distrobuilder` でイメージを生成する (**イメージをビルドする**参照)
- image server からイメージファイルをダウンロードする (イメージをファイルにダウンロードしてインポートするより、**リモートのイメージを使用する**ほうが通常は簡単なことに注意してください)

ローカルファイルシステムからインポートする

ローカルファイルシステムからイメージをインポートするには、`lxc image import` コマンドを使用します。このコマンドは**統合イメージ** (圧縮されたファイルまたはディレクトリ) と**分離イメージ** (2 つのファイル) の両方をサポートします。

1 つのファイルまたはディレクトリから統合イメージをインポートするには、以下のコマンドを入力します。

```
lxc image import <image_file_or_directory_path> [<target_remote>:]
```

分離イメージをインポートするには、以下のコマンドを入力します。

```
lxc image import <metadata_tarball_path> <rootfs_tarball_path> [<target_remote>:]
```

どちらの場合も、`--alias` フラグでエイリアスを割り当てられます。利用可能なすべてのフラグは `lxc image import --help` を参照してください。

リモートウェブサーバーからファイルをインポートする

URL を指定してリモートウェブサーバーからイメージファイルをインポートできます。この方法はイメージをユーザに配布するためだけに LXD サーバーを稼働させる代わりに使用できます。必要なのはカスタムヘッダ (**カスタム** *HTTP* **ヘッダ**参照) をサポートする基本的なウェブサーバーだけです。

イメージファイルは統合イメージ (**統合** *tarball* 参照) として提供される必要があります。

リモートウェブサーバーからイメージをインポートするには、以下のコマンドを入力します。

```
lxc image import <URL>
```

`--alias` フラグでローカルのイメージにエイリアスを割り当てられます。

カスタム HTTP ヘッダ

LXD では以下のカスタム HTTP ヘッダをウェブサーバーで設定する必要があります。

`LXD-Image-Hash` ダ
: ウンロードされるイメージの SHA256 ハッシュ値。

`LXD-Image-URL` イ
: メージをダウンロードする URL。

LXD はサーバーに問い合わせる際に以下のヘッダを設定します。

`LXD-Server-Architectures` ク
: ライアントがサポートするアーキテクチャのカンマ区切りリスト。

LXD-Server-Version 使用している LXD のバージョン

3.5.5 イメージを作成するには

独自のイメージを作成し共有したい場合、既存のインスタンスやスナップショットをベースにすることもできますし、一から独自のイメージを作ることもできます。

インスタンスやスナップショットからイメージを発行する

インスタンスやインスタンススナップショットを新しいインスタンスのベースとして使いたい場合、それらからイメージを作成し発行するのが良いです。

インスタンスからイメージを発行するには、インスタンスが停止されていることを確認してください。次に以下のコマンドを入力します。

```
lxc publish <instance_name> [<remote>:]
```

スナップショットからイメージを発行するには、以下のコマンドを入力します。

```
lxc publish <instance_name>/<snapshot_name> [<remote>:]
```

どちらの場合も`--alias` フラグで新しいイメージにエイリアスを設定し、`--expire` で有効期限を設定し、`--public` でイメージを公開状態にすることができます。同じ名前のイメージがすでに存在する場合は、`--reuse` フラグを追加して上書きします。利用可能な全てのフラグ一覧は `lxc publish --help` を参照してください。

発行のプロセスはインスタンスやスナップショットから tarball を生成した後圧縮するため、かなりの時間がかかるかもしれません。特に I/O と CPU の負荷が高いため、発行の操作は LXD で直列化 (訳注:1 つずつ順に実行) されます。

発行用にインスタンスを準備する

インスタンスからイメージを発行する前に、イメージに含めるべきでない全てのデータをクリーンアップしてください。通常、これは以下のデータを含みます。

- インスタンスメタデータ (編集には `lxc config metadata` を使ってください)
- ファイルテンプレート (編集には `lxc config template` を使ってください)
- インスタンス自身の内部のインスタンスに特有なデータ (例えば、ホストの SSH 鍵と `dbus/systemd machine-id`)

イメージをビルドする

独自イメージをビルドするには、distrobuilder が使用できます。

インストール手順とツールの使い方は distrobuilder のドキュメントを参照してください。

3.5.6 イメージにプロファイルを関連付けるには

特定のイメージに 1 つ以上のプロファイルを関連付けできます。イメージから作成されたインスタンスは関連付けられたプロファイルが指定された順に自動的に適用されます。

イメージにプロファイルのリストを関連付けるには、lxc image edit コマンドを使って profiles: セクションを編集します。

```
profiles:
- default
```

提供されているほとんどのイメージは default プロファイルだけを含むプロファイルリストが設定されています。(default プロファイルを含む) 全てのプロファイルがイメージに関連付けられないようにするには、空のリストを渡します。

注釈: 空のリストを渡すのは nil を渡すのとは異なります。プロファイルリストに nil を渡すとイメージに default プロファイルだけが関連付けられます。

---

launch または init コマンドに--profile または--no-profiles フラグを追加することで、イメージに関連付けられたプロファイルをインスタンス作成時にオーバーライドできます。

3.5.7 リモートイメージサーバー

lxc CLI コマンドは下記のデフォルトリモートイメージサーバーが初期設定されています。

images: このサーバーはさまざまな Linux ディストリビューションの非公式イメージを提供します。イメージは LXD チームによりメンテナンスされ、コンパクトで最小限にビルドされています。

利用可能なイメージの概要については images.linuxcontainers.org を参照してください。

ubuntu: このサーバーは公式の安定版の Ubuntu イメージを提供します。全てのイメージは cloud イメージです。これは cloud-init と lxd-agent の両方を含んでいることを意味します。

利用可能なイメージの概要については cloud-images.ubuntu.com/releases を参照してください。

ubuntu-daily: このサーバーは公式のデイリービルド版の Ubuntu イメージを提供します。全てのイメージは cloud イメージです。これは `cloud-init` と `lxd-agent` の両方を含んでいることを意味します。

利用可能なイメージの概要については cloud-images.ubuntu.com/daily を参照してください。

リモートサーバータイプ

LXD は下記のタイプのリモートイメージサーバーをサポートします。

simple streams **サーバー**
: simple streams 形式を使う純粋なイメージサーバー。デフォルトのイメージサーバーは simple streams サーバーです。

公開 LXD **サーバー** イメージを配布するためだけに稼働し、このサーバー自身ではインスタンスを稼働しない LXD サーバー。

LXD サーバーをポート 8443 で公開で利用可能にするには、*core.https_address* 設定オプションを`:8443`に設定し、認証方法をなにも設定しないようにします (詳細は *LXD* **をネットワークに公開するには**参照)。そして共有したいイメージを `public` にセットします。

LXD **サーバー** ネットワーク越しに管理できる通常の LXD サーバー、イメージサーバーとしても利用可能。

セキュリティ上の理由により、リモート API へのアクセスを制限し、アクセス制御のための認証方法を設定するほうが良いです。詳細な情報は *LXD* **をネットワークに公開するには**と**リモート** *API* **認証**を参照してください。

3.5.8 イメージ形式

イメージはルートファイルシステムとイメージを記述するメタデータファイルを含みます。またイメージを使用するインスタンス内部でファイルを生成するためのテンプレートも含められます。

イメージは統合イメージ (単一ファイル) か分離イメージ (2 つのファイル) としてパッケージできます。

中身

コンテナのイメージは以下のディレクトリ構造を持ちます。

```
metadata.yaml
rootfs/
templates/
```

VM のイメージは以下のディレクトリ構造を持ちます。

```
metadata.yaml
rootfs.img
templates/
```

どちらのインスタンスタイプでも、templates/ディレクトリは省略可能です。

メタデータ

metadata.yaml ファイルはイメージが LXD 内で稼働するために関連する情報を含みます。以下の情報を含んでいます。

```
architecture: x86_64
creation_date: 1424284563
properties:
  description: Ubuntu 22.04 LTS Intel 64bit
  os: Ubuntu
  release: jammy 22.04
templates:
  ...
```

architecture と creation_date フィールドは必須です。properties フィールドはイメージのデフォルトプロパティのセットを含みます。os, release, name, description フィールドはよく使われますが、必須ではありません。

templates フィールドは省略可能です。テンプレートをどのように設定するかの情報は**テンプレート *(省略可能)*** を参照してください。

ルートファイルシステム

コンテナでは、rootfs/ディレクトリがコンテナ内のルートディレクトリ (/) の完全なファイルシステムツリーを含みます。

仮想マシンは rootfs/ディレクトリの代わりに rootfs.img qcow2 ファイルを使います。このファイルはメインのディスクデバイスになります。

テンプレート (省略可能)

インスタンス内部でファイルを動的に作成するのにテンプレートを使用できます。そのためには、metadata.yaml ファイル内でテンプレートルールを設定し、templates/ディレクトリ内にテンプレートファイルを配置します。

一般的なルールとして、パッケージに所有されるファイルはテンプレート化は決してするべきではないです。そうでないとインスタンスの通常のオペレーションで上書きされてしまうでしょう。

テンプレートルール

生成すべき各ファイルに対して、metadata.yaml ファイル内でルールを作成します。例:

```
templates:
  /etc/hosts:
    when:
      - create
      - rename
    template: hosts.tpl
    properties:
      foo: bar
  /etc/hostname:
    when:
      - start
    template: hostname.tpl
  /etc/network/interfaces:
    when:
      - create
    template: interfaces.tpl
    create_only: true
```

when キーは以下の 1 つ以上を指定できます。

- create - 新規インスタンスがイメージから作成された時に実行
- copy - 既存インスタンスからインスタンスが作成されたときに実行
- start - インスタンスが開始する度に毎回実行

template キーは templates/ディレクトリ内のテンプレートファイルを指します。

properties キーでユーザ定義のテンプレートプロパティをテンプレートファイルに渡せます。

ファイルが存在しない場合にのみ LXD にファイルを作らせ、ファイルが存在する場合は上書きしてほしくない場合は、create_only キーをセットします。

テンプレートファイル

テンプレートファイルは Pongo2 形式を使います。

テンプレートファイルは常に以下のコンテキストを受け取ります。

変数	型	説明
trigger	string	テンプレートをトリガーしたイベント名
path	string	テンプレートを使用するファイルのパス
instance	map[string]string	インスタンスプロパティのキー/値マップ (名前、アーキテクチャ、特権、一時的)
config	map[string]string	インスタンス設定のキー/値マップ
devices	map[string]map[string]st	インスタンスに割り当てられたデバイスのキー/値マップ
propertie	map[string]string	metadata.yaml で指定されたテンプレートプロパティのキー/値マップ

利便性のため、以下の関数が Pongo2 テンプレートにエクスポートされます。

- config_get("user.foo", "bar") - user.foo の値か、未設定の場合は"bar"を返します。

イメージの tarball

LXD は 2 種類の LXD 固有のイメージ形式、統合 tarball と分離 tarball をサポートします。

これらの tarball は圧縮されていても構いません。LXD は tarball の広範囲の圧縮アルゴリズムをサポートします。しかし、互換性のためには gzip または xz を使うのが良いです。

統合 tarball

統合 tarball は単一の tarball(通常*.tar.xz) で、イメージの完全な中身を含みます。それにはメタデータ、ルートファイルシステムと省略可能なテンプレートファイルが含まれます。

これが LXD 自身がイメージを公開する際に内部的に使用している形式です。通常こちらのほうが扱いやすいので、LXD 固有のイメージを作る際は統合形式を使うのが良いです。

この形式のイメージの識別子は tarball の SHA-256 ハッシュ値です。

分離 tarball

分離イメージは 2 つの分離した tarball から構成されます。1 つの tarball(通常`*.tar.xz`) はメタデータと省略可能なテンプレートファイルを含み、もう 1 つ (通常、コンテナでは`*.squashfs` で仮想マシンでは`*.qcow2`) はルートファイルシステムを含みます。

コンテナでは、ルートファイルシステムの tarball は SquashFS でフォーマットされていても構いません。仮想マシンでは、`rootfs.img` ファイルは常に `qcow2` 形式を使用します。任意で `qcow2` のネイティブ圧縮を使って圧縮しても構いません。

この形式は既に利用可能である既存の LXD 以外の rootfs tarball から簡単にイメージをビルドできるように設計されています。LXD と他のツールの両方で使用するイメージを作りたい場合もこの形式を使うのが良いです。

この形式のイメージの識別子はメタデータとルートファイルシステム tarball を (この順で) 結合したものの SHA-256 ハッシュ値です。

3.6 ストレージ

3.6.1 ストレージプール、ボリューム、バケットについて

LXD はデータを（イメージやインスタンスのように）コンテントタイプに応じて別のストレージボリュームに分けてストレージプールに保管します。ストレージプールはデータを保管するためのディスクであり、ストレージボリュームは特定の目的に使用されるディスク上の別々のパーティションであると考えることも出来るでしょう。

ストレージボリュームに加えて、ストレージバケットというものもあります。これは Amazon S3 (Simple Storage Service) プロトコルを使用します。ストレージボリュームと同様に、ストレージバケットはストレージプールの一部です。

ストレージプール

初期化時に、 LXD は最初のストレージプールを作成するためのプロンプトを表示します。必要であれば、後からストレージプールを追加できます (**ストレージプールを作成する** 参照)。

それぞれのストレージプールはストレージドライバを使用します。次のストレージドライバが利用できます。

- **ディレクトリ** - *`dir`*
- *Btrfs* - *`btrfs`*
- *LVM* - *`lvm`*
- *ZFS* - *`zfs`*
- *Ceph RBD* - *`ceph`*

- *CephFS - `cephfs`*
- *Ceph Object - `cephobject`*

さらなる情報については以下の how-to ガイドを参照してください。

- **ストレージプールを管理するには**
- **特定のストレージプール内にインスタンスを作成するには**

データストレージのロケーション

LXD のデータをどこに保管するかは設定と選択したストレージドライバによって異なります。使用されるストレージドライバによって、 LXD はファイルシステムをホストと共有することもできますし、データを別にしておくこともできます。

ストレージロケーション	Directory	Btrfs	LVM	ZFS	Ceph (全て)
ホストと共有	✓	✓	-	✓	-
専用のディスク/パーティション	-	✓	✓	✓	-
ループディスク	-	✓	✓	✓	-
リモートストレージ	-	-	-	-	✓

ホストと共有

ファイルシステムをホストと共有するのは LXD を実行する上で通常もっとも空間効率が良い方法です。ほとんどの場合、もっとも管理が楽な方法でもあります。

この選択肢は `dir` ドライバ、 `btrfs` ドライバ (ホストが Btrfs で LXD に専用のサブボリュームを使用する場合)、 `zfs` ドライバ (ホストが ZFS で zpool 上で LXD に専用のデータセットを使用する場合) でサポートされます。

専用のディスク/パーティション

メインのディスク上で LXD に空のパーティションを使用するか、ホストから完全に独立したストレージを保管する完全な専用のディスクを使用します。

この選択肢は `btrfs` ドライバ、 `lvm` ドライバ、 `zfs` ドライバでサポートされます。

ループディスク

LXD ではメインドライブ上にループファイルを作成して選択したストレージドライバでそれを使用できます。この方法はディスクやパーティションを使用する方法と機能的には似ていますが、大きな 1 つのファイルをメインドライブとして使用する点が異なります。これはそれぞれの書き込みがストレージドライバとメインドライブのファイルシステムを通過することを意味し、パフォーマンスは低くなります。

ループファイルは snap を使用している場合は `/var/snap/lxd/common/lxd/disks/`、それ以外の場合は `/var/lib/lxd/disks/` に作られます。

ループファイルは通常縮小できません。最大で指定した限界まで拡大しますが、インスタンスやイメージを削除してもファイルが縮小することはありません。しかしサイズを増やすことはできます。 **ストレージプールをリサイズする** を参照してください。

リモートストレージ

`ceph`, `cephfs`, `cephobject` ドライバはデータを完全に独立な Ceph ストレージクラスタに保管します。これは別途セットアップが必要です。

デフォルトストレージプール

LXD にはデフォルトストレージプールという概念はありません。

ストレージボリュームを作成する時は、使用するストレージプールを指定する必要があります。

インスタンスの作成時に LXD が自動的にストレージボリュームを作成する際は、インスタンスに設定されたストレージプールを使用します。この設定は以下のいずれかの方法でできます。

- 直接インスタンスに指定: `lxc launch <image> <instance_name> --storage <storage_pool>`
- プロファイル経由: `lxc profile device add <profile_name> root disk path=/ pool=<storage_pool>` and `lxc launch <image> <instance_name> --profile <profile_name>`
- デフォルトプロファイル経由

プロファイルでは使用するストレージプールはルートディスクデバイスのプールで定義されます。

```
root:
  type: disk
  path: /
  pool: default
```

デフォルトプロファイルではこのプールは (訳注: LXD の) 初期化時に作られたストレージプールに設定されています。

ストレージボリューム

インスタンスを作成する際、 LXD は必要なストレージボリュームを自動的に作成します。追加のストレージボリュームを作成することもできます。

さらなる情報については以下の how-to ガイドを参照してください。

- **ストレージボリュームを管理するには**
- **ストレージボリュームを移動やコピーするには**
- **カスタムストレージボリュームをバックアップするには**

ストレージボリュームタイプ

ストレージボリュームは以下の種別があります。

`container/vm`
: LXD はインスタンスを起動する際にこのどちらかのストレージボリュームを自動的に作成します。それはインスタンスのルートディスクとして使用され、インスタンスが削除される際に破棄されます。

  このストレージボリュームはインスタンス起動時に使用されたプロファイル (あるいはプロファイルが指定されない場合はデフォルトプロファイル) に指定されたストレージプール内に作成されます。起動のコマンドに `--storage` フラグを渡してストレージプールを明示的に指定することもできます。

`image`
: LXD はイメージから 1 つあるいは複数のインスタンスを起動するためにイメージを解凍する際にこれらのストレージボリュームを自動的に作成します。インスタンスが作成された後は削除できます。手動で削除しない場合、インスタンス起動の 10 日後に自動的に削除されます。

  イメージのストレージボリュームはインスタンスのストレージボリュームと同じストレージプール内に作成されます。それは最適化されたイメージのストレージをサポートする **ストレージドライバ** を使用するストレージプールだけです。

`custom`
: インスタンスから分離して保管したいデータを保持する 1 つあるいは複数のカスタムストレージボリュームを追加できます。カスタムストレージボリュームはインスタンス間で共有でき、インスタンスが削除されても残ります。

  バックアップやイメージを保管するためにカスタムストレージボリュームを使用することもできます。

  カスタムボリュームの作成時は使用するストレージプールを指定する必要があります。

コンテントタイプ

それぞれのストレージボリュームは以下のコンテントタイプのどれかを使用します。

`filesystem`
: このコンテントタイプはコンテナとコンテナイメージに使用されます。これはカスタムストレージボリュームのデフォルトのコンテントタイプです。

  コンテントタイプが `filesystem` のカスタムストレージボリュームはコンテナと仮想マシンの両方にアタッチでき、インスタンス間で共有できます。

`block`
: このコンテントタイプは仮想マシンと仮想マシンイメージで使用されます。コンテントタイプ `block` のカスタムストレージボリュームは `--type=block` フラグを使って作成できます。

  コンテントタイプが `block` のカスタムストレージボリュームは仮想マシンのみにアタッチできます。これらはインスタンス間では共有すべきではありません。同時アクセスはデータ破壊を引き起こすからです。

ストレージバケット

ストレージバケットは S3 プロトコルを使用してオブジェクトストレージの機能を提供します。

これはカスタムストレージボリュームと同様の方法で使用されます。しかし、ストレージボリュームとは異なり、ストレージバケットはインスタンスに紐付けされません。代わりに、アプリケーションはストレージバケットにその URL を使って直接アクセスできます。

それぞれのストレージバケットには 1 つまたは複数のアクセスキーが割り当てられ、アプリケーションはアクセスの際にこれを使う必要があります。

ストレージバケットはローカルのストレージ (`dir`, `btrfs`, `lvm` あるいは `zfs` プールの場合) あるいはリモートストレージ (`cephobject` プールの場合) 上に配置できます。

ローカルストレージドライバでストレージバケットを有効にし、 S3 プロトコル経由でアプリケーションがバケットにアクセスできるようにするには、`core.storage_buckets_address` サーバー設定 (**コア設定**参照) を調整する必要があります。

さらなる情報については以下の how-to ガイドを参照してください。

- **ストレージバケットとキーを管理するには**

3.6.2 ストレージプールを管理するには

ストレージプール を作成、設定、表示、リサイズするための手順については以下のセクションを参照してください。

ストレージプールを作成する

LXD は初期化中にストレージプールを作成します。同じドライバあるいは別のドライバを使用して、後からさらにストレージプールを追加できます。

ストレージプールを作成するには以下のコマンドを使用します。

```
lxc storage create <pool_name> <driver> [configuration_options...]
```

別途指定しない場合は、 LXD は実用的なデフォルトのサイズ (空きディスクスペースの 20%、しかし最低 5GiB で最大 30GIB) でループベースのストレージをセットアップします。

それぞれのドライバで利用可能な設定オプションの一覧は **ストレージドライバ** ドキュメントを参照してください。

例

それぞれのストレージドライバでストレージプールを作成する例は以下を参照してください。

ディレクトリ

`pool1` という名前のディレクトリプールを作成する。

```
lxc storage create pool1 dir
```

`/data/lxd` という既存のディレクトリを使って `pool2` を作成する。

```
lxc storage create pool2 dir source=/data/lxd
```

Btrfs

`pool1` という名前のループバックプールを作成する。

```
lxc storage create pool1 btrfs
```

`/some/path` にある既存の Btrfs ファイルシステムを使って `pool2` を作成する。

```
lxc storage create pool2 btrfs source=/some/path
```

`/dev/sdX` 上に `pool3` という名前のプールを作成する。

```
lxc storage create pool3 btrfs source=/dev/sdX
```

LVM

pool1 という名前のループバックのプールを作成する (LVM ボリュームグループ名も pool1 になります)。

```
lxc storage create pool1 lvm
```

my-pool という既存の LVM ボリュームグループを使って pool2 を作成する。

```
lxc storage create pool2 lvm source=my-pool
```

my-vg というボリュームグループ内の my-pool という既存の LVM thin-pool を使って pool3 を作成する。

```
lxc storage create pool3 lvm source=my-vg lvm.thinpool_name=my-pool
```

/dev/sdX 上に pool4 という名前のプールを作成する (LVM ボリュームグループ名も pool4 になります)。

```
lxc storage create pool4 lvm source=/dev/sdX
```

/dev/sdX 上に my-pool という LVM ボリュームグループ名で pool5 という名前のプールを作成する。

```
lxc storage create pool5 lvm source=/dev/sdX lvm.vg_name=my-pool
```

ZFS

pool1 という名前のループバックプールを作成する (ZFS zpool 名も pool1 になります)。

```
lxc storage create pool1 zfs
```

pool2 という名前のループバックプールを my-tank という ZFS zpool 名で作成する。

```
lxc storage create pool2 zfs zfs.pool_name=my-tank
```

my-tank という既存の ZFS zpool を使用して pool3 を作成する。

```
lxc storage create pool3 zfs source=my-tank
```

my-tank/slice という既存の ZFS データセットを使用して pool4 を作成する。

```
lxc storage create pool4 zfs source=my-tank/slice
```

/dev/sdX 上に pool5 という名前のプールを作成する (ZFS zpool 名も pool5 になります)。

```
lxc storage create pool5 zfs source=/dev/sdX
```

/dev/sdX 上に my-tank という ZFS zpool 名で pool6 という名前のプールを作成する。

```
lxc storage create pool6 zfs source=/dev/sdX zfs.pool_name=my-tank
```

Ceph RBD

デフォルトの Ceph クラスタ (名前は ceph) 内に pool1 という名前の OSD ストレージプールを作成する。

```
lxc storage create pool1 ceph
```

my-cluster という Ceph クラスタ内に pool2 という名前の OSD ストレージプールを作成する。

```
lxc storage create pool2 ceph ceph.cluster_name=my-cluster
```

デフォルトの Ceph クラスタ内に my-osd という on-disk 名で pool3 という名前の OSD ストレージプールを作成する。

```
lxc storage create pool3 ceph ceph.osd.pool_name=my-osd
```

my-already-existing-osd という既存の OSD ストレージプールを使って pool4 を作成する。

```
lxc storage create pool4 ceph source=my-already-existing-osd
```

ecpool という既存の OSD ストレージプールと rpl-pool という OSD リプリケーテッドプールを使って pool5 を作成する。

```
lxc storage create pool5 ceph source=rpl-pool ceph.osd.data_pool_name=ecpool
```

CephFS

注釈: CephFS ドライバを使用する際は、事前に CephFS ファイルシステムを作成する必要があります。このファイルシステムは 2 つの OSD ストレージプールからなります。そのうち 1 つは実際のデータ、もう 1 つはファイルメタデータに使用されます。

既存の CephFS ファイルシステム my-filesystem を使って pool1 を作成する。

```
lxc storage create pool1 cephfs source=my-filesystem
```

my-filesystem ファイルシステムからサブディレクトリ my-directory を使って pool2 を作成する。

```
lxc storage create pool2 cephfs source=my-filesystem/my-directory
```

Ceph Object

注釈: Ceph Object ドライバを使用する場合、事前に稼働中の Ceph Object Gateway `radosgw` の URL を用意しておく必要があります。

既存の Ceph Object Gateway `https://www.example.com/radosgw` を使用して `pool1` を作成する。

```
lxc storage create pool1 cephobject cephobject.radosgw.endpoint=https://www.example.com/
↪radosgw
```

クラスタ内にストレージプールを作成する

LXD クラスタを稼働していてストレージプールを追加したい場合、それぞれのクラスタメンバー内にストレージを別々に作る必要があります。この理由は、設定、例えばストレージのロケーションやプールのサイズがクラスタメンバー間で異なるかもしれないからです。

このため、 `--target=<cluster_member>` フラグを指定してストレージプールをペンディング状態でまず作成し、メンバーごとに適切な設定を行う必要があります。全てのメンバーで同じストレージプール名を使用しているか確認してください。次に `--target` フラグなしでストレージプールを作成し、実際にセットアップします。

例えば、以下の一連のコマンドは 3 つのクラスタメンバー上で異なるロケーションと異なるサイズで `my-pool` という名前のストレージプールをセットアップします。

```
user@host:~$ lxc storage create my-pool zfs source=/dev/sdX size=10GB --target=vm01
Storage pool my-pool pending on member vm01        user@host:~$ lxc storage create my-pool
zfs source=/dev/sdX size=15GB --target=vm02    Storage pool my-pool pending on member vm02
user@host:~$ lxc storage create my-pool zfs source=/dev/sdY size=10GB --target=vm03
Storage pool my-pool pending on member vm03    user@host:~$ lxc storage create my-pool zfs
Storage pool my-pool created
```

クラスタのストレージを設定するにはも参照してください。

注釈: ほとんどのストレージドライバでは、ストレージプールは各クラスタメンバー上にローカルに存在します。これは 1 つのメンバー上のストレージプール内にストレージボリュームを作成しても、別のクラスタメンバー上では利用可能にはならないことを意味します。

この挙動は Ceph ベースのストレージプール (`ceph`、 `cephfs`、 `cephobject`) では異なります。これらではストレージプールは 1 つの中央のロケーション上に存在し、全てのクラスタメンバーが同じストレージボリュームを持つ同じストレージプールにアクセスします。

ストレージプールを設定する

各ストレージドライバで利用可能な設定オプションについては **ストレージドライバ** ドキュメントを参照してください。

(source のような) ストレージプールの一般的なキーはトップレベルです。ドライバ固有のキーはドライバ名で名前空間が分けられています。

ストレージプールに設定オプションを設定するには以下のコマンドを使用します。

```
lxc storage set <pool_name> <key> <value>
```

例えば、 dir ストレージプールでストレージプールのマイグレーション中に圧縮をオフにするには以下のコマンドを使用します。

```
lxc storage set my-dir-pool rsync.compression false
```

ストレージプールの設定を編集するには以下のコマンドを使用します。

```
lxc storage edit <pool_name>
```

ストレージプールを表示する

全ての利用可能なストレージプールの一覧を表示し設定を確認できます。

以下のコマンドで全ての利用可能なストレージプールを一覧表示できます。

```
lxc storage list
```

出力結果の表には (訳注: LXD の) 初期化時に作成した (通常 default や local と呼ばれる) ストレージプールとあなたが追加したあらゆるストレージプールが含まれます。

特定のプールに関する詳細情報を表示するには、以下のコマンドを使用します。

```
lxc storage show <pool_name>
```

特定のプールに関する使用量を表示するには、以下のコマンドを使用します。

```
lxc storage info <pool_name>
```

ストレージプールをリサイズする

ストレージがもっと必要な場合、size 設定キーを変更することでストレージプールのサイズを拡大できます。

```
lxc storage set <pool_name> size=<new_size>
```

これはループファイルをバックエンドとし LXD で管理されているストレージプールでのみ機能します。プールは拡張（サイズを増やす）のみが可能で、縮小はできません。

3.6.3 特定のストレージプール内にインスタンスを作成するには

インスタンスストレージボリュームはインスタンスのルートディスクデバイスにより指定されたストレージプール内に作成されます。通常この設定はインスタンスに適用されるプロファイルで提供されます。詳細な情報は **デフォルトストレージプール** を参照してください。

インスタンスを作成または起動する際に別のストレージプールを使用するには --storage フラグを追加します。このフラグはプロファイルからのルートディスクデバイスをオーバーライドします。例えば

```
lxc launch <image> <instance_name> --storage <storage_pool>
```

インスタンスストレージプールを別のプールに移動する

インスタンスストレージプールを別のプールに移動するにはまずインスタンスが停止されていることを確認してください。次に以下のコマンドでインスタンスを別のプールに移動します。

```
lxc move <instance_name> --storage <target_pool_name>
```

3.6.4 ストレージボリュームを管理するには

ストレージボリューム を作成、設定、表示、リサイズするための手順については以下のセクションを参照してください。

カスタムストレージボリュームを作成する

インスタンスを作成する際に、 LXD はインスタンスのルートディスクとして使用するストレージボリュームを自動的に作成します。

インスタンスにカスタムストレージボリュームを追加できます。このカスタムストレージボリュームはインスタンスから独立しています。これは別にバックアップできたり、カスタムストレージボリュームを削除するまで残っていることを意味します。コンテントタイプが filesystem のカスタムストレージボリュームは異なるインスタンス間で共有もできます。

詳細な情報は **ストレージボリューム** を参照してください。

ボリュームを作成する

ストレージプール内にカスタムストレージボリュームを作成するには以下のコマンドを使用します。

```
lxc storage volume create <pool_name> <volume_name> [configuration_options...]
```

各ドライバで利用可能なストレージボリューム設定オプションについては **ストレージドライバ** ドキュメントを参照してください。

デフォルトではカスタムストレージボリュームは `filesystem` **コンテントタイプ** を使用します。`block` コンテントタイプのカスタムストレージボリュームを作成するには `--type` フラグを追加してください。

```
lxc storage volume create <pool_name> <volume_name> --type=block [configuration_options..
↪.]
```

クラスタメンバー上にカスタムストレージボリュームを追加するには `--target` フラグを追加してください。

```
lxc storage volume create <pool_name> <volume_name> --target=<cluster_member>␣
↪[configuration_options...]
```

注釈: ほとんどのストレージドライバではカスタムストレージボリュームはクラスタ間で同期されず作成されたメンバー上にのみ存在します。この挙動は Ceph ベースのストレージプール (`ceph` and `cephfs`) では異なり、ボリュームはどのクラスタメンバーでも利用可能です。

インスタンスにカスタムストレージボリュームをアタッチする

カスタムストレージボリュームを作成したら、それを 1 つあるいは複数のインスタンスに **ディスクデバイス** として追加できます。

以下の制限があります。

- **コンテントタイプ** `block` のカスタムストレージボリュームはコンテナにはアタッチできず、仮想マシンのみにアタッチできます。
- データ破壊を防ぐため、 **コンテントタイプ** `block` のカスタムストレージボリュームは同時に複数の仮想マシンには決してアタッチするべきではありません。

コンテントタイプ `filesystem` のカスタムストレージボリュームは以下のコマンドを使用します。ここで `<location>` はインスタンス内でストレージボリュームにアクセスするためのパス (例: `/data`) です。

```
lxc storage volume attach <pool_name> <filesystem_volume_name> <instance_name> <location>
```

コンテントタイプ block のカスタムストレージボリュームは <location> を指定しません。

```
lxc storage volume attach <pool_name> <block_volume_name> <instance_name>
```

デフォルトではカスタムストレージボリュームはインスタンスに **デバイス** の名前でボリュームが追加されます。異なるデバイス名を使用したい場合は、コマンドにデバイス名を追加できます。

```
lxc storage volume attach <pool_name> <filesystem_volume_name> <instance_name> <device_
↪name> <location>
lxc storage volume attach <pool_name> <block_volume_name> <instance_name> <device_name>
```

ボリュームをデバイスとしてアタッチする

`lxc storage volume attach` コマンドは、インスタンスにディスクデバイスを追加するためのショートカットです。もしくは、通常の方法でストレージボリュームのディスクデバイスを追加することもできます：

```
lxc config device add <instance_name> <device_name> disk pool=<pool_name> source=<volume_
↪name> [path=<location>]
```

この方法を使用すると、必要に応じてコマンドに更なる設定を追加することができます。利用可能なすべてのデバイスオプションについては **ディスクデバイス** を参照してください。

I/O 制限値の設定

ストレージボリュームをインスタンスに **ディスクデバイス** としてアタッチする際に、 I/O 制限値を設定できます。そのためには `limits.read`, `limits.write`, `limits.max` に対応する制限値を設定します。詳細な情報は **タイプ**: *disk* レファレンスを参照してください。

制限値は Linux の `blkio` cgroup コントローラー経由で適用されます。これによりディスクのレベルで I/O を制限することができます (しかしそれより細かい単位では制限できません)。

注釈: 制限値はパーティションやパスではなく物理ディスク全体に適用されるため、以下の制約があります。

- 仮想デバイス (例えば device mapper) 上に存在するファイルシステムには制限値は適用されません
- ファイルシステムが複数のブロックデバイス上に存在する場合、各デバイスは同じ制限を受けます。
- 同じディスク上に存在する 2 つのディスクデバイスが同じインスタンスにアタッチされた場合は、 2 つのデバイスの制限値は平均されます

全ての I/O 制限値は実際のブロックデバイスアクセスにのみ適用されます。そのため、制限値を設定する際はファイルシステム自体のオーバーヘッドを考慮してください。キャッシュされたデータへのアクセスはこの制限値に影響されません。

バックアップやイメージにボリュームを使用する

カスタムボリュームをディスクデバイスとしてインスタンスにアタッチする代わりに、**バックアップ** あるいは **イメージ** を格納する特別な種類のボリュームとして使うこともできます。

このためには、対応する**サーバー設定**を設定する必要があります。

- バックアップ tarball を保管するためにカスタムボリュームを使用する。

```
lxc config set storage.backups_volume <pool_name>/<volume_name>
```

- イメージ tarball を保管するためにカスタムボリュームを使用する。

```
lxc config set storage.images_volume <pool_name>/<volume_name>
```

ストレージボリュームを設定する

各ストレージドライバで利用可能な設定オプションについては **ストレージドライバ** ドキュメントを参照してください。

ストレージボリュームの設定オプションを設定するには以下のコマンドを使用します。

```
lxc storage volume set <pool_name> <volume_name> <key> <value>
```

例えば、スナップショットの破棄期限を 1 ヶ月に設定するには以下のコマンドを使用します。

```
lxc storage volume set my-pool my-volume snapshots.expiry 1M
```

インスタンスのストレージボリュームを設定するには、 **ストレージボリュームタイプ** を含めたボリューム名を指定します。例えば

```
lxc storage volume set my-pool container/my-container-volume user.XXX value
```

ストレージボリューム設定を編集するには以下のコマンドを使用します。

```
lxc storage volume edit <pool_name> <volume_name>
```

ストレージボリュームのデフォルト値を変更する

ストレージプールのデフォルトのボリューム設定を定義できます。そのためには、 volume 接頭辞をつけたストレージプール設定 volume.<VOLUME_CONFIGURATION>=<VALUE> をセットします。

新しいストレージボリュームまたはインスタンスに明示的に設定されない限り、この値はプール内の全ての新しいストレージボリュームに使用されます。一般的に、ストレージプールのレベルに設定されたデフォルト値は (ボリュームが作成される前であれば) ボリューム設定でオーバーライドでき、ボリューム設定はインスタンス設定 (**タイプ** が container か vm のストレージボリュームについて) でオーバーライドできます。

例えば、ストレージプールにデフォルトのボリュームサイズを設定するには以下のコマンドを使用します。

```
lxc storage set [<remote>:]<pool_name> volume.size <value>
```

ストレージボリュームを表示する

ストレージプール内の全ての利用可能なストレージボリュームを一覧表示しそれらの設定を確認できます。

あるストレージプール内の全ての利用可能なストレージボリュームを一覧表示するには以下のコマンドを使用します。

```
lxc storage volume list <pool_name>
```

全てのプロジェクト (デフォルトのプロジェクトだけでなく) ストレージボリュームを表示するには、--all-projects フラグを追加してください。

結果の表にはそのプール内の各ストレージボリュームについて **ストレージボリュームタイプ** と **コンテントタイプ** が含まれます。

注釈: カスタムストレージボリュームはインスタンスボリュームと同じ名前を使うこともできます (例えば c1 という名前のコンテナストレージボリュームと c1 という名前のカスタムストレージボリュームを持つ c1 という名前のコンテナを作成することもできます)。このため、インスタンスストレージボリュームとカスタムストレージボリュームを区別するには、全てのインタンスストレージボリュームは <volume_type>/<volume_name> (例えば container/c1 または virtual-machine/vm) のようにコマンド内で指定する必要があります。

特定のカスタムボリュームについて詳細な情報を表示するには以下のコマンドを使用します。

```
lxc storage volume show <pool_name> <volume_name>
```

特定のインスタンスボリュームについて詳細な情報を表示するには以下のコマンドを使用します。

```
lxc storage volume show <pool_name> <volume_type>/<volume_name>
```

ストレージボリュームをリサイズする

ボリュームにもっとストレージが必要な場合、ストレージボリュームのサイズを拡大できます。場合によっては、ストレージボリュームのサイズを縮小することもできます。

ストレージボリュームをリサイズするにはサイズ設定を設定します。

```
lxc storage volume set <pool_name> <volume_name> size <new_size>
```

重要:

- ストレージボリュームの拡大は通常は正常に動作します (ストレージプールが十分なストレージを持つ場合)。
- ストレージボリュームの縮小はコンテントタイプ `filesystem` のストレージボリュームでのみ可能です。ただし現在使用しているサイズより小さく縮小はできないので、縮小が保証されているわけではありません。
- コンテントタイプ `block` のストレージボリュームの縮小は不可能です。

3.6.5 ストレージボリュームを移動やコピーするには

カスタムストレージボリュームはあるストレージプールから別のストレージプールに **コピー** や **移動** したり、同じストレージプール内でコピーやリネームできます。

インスタンスストレージボリュームをあるストレージプールから別のストレージプールに移動するには、別のストレージプールに **対応するインスタンスを移動** します。

別のドライバを使用するストレージプール間でボリュームをコピーまたは移動する際はボリュームは自動的に変換されます。

カスタムストレージボリュームをコピーする

カスタムストレージボリュームをコピーするには以下のコマンドを使用します。

```
lxc storage volume copy <source_pool_name>/<source_volume_name> <target_pool_name>/
↪<target_volume_name>
```

ボリュームに存在するかもしれないスナップショットをスキップしてボリュームだけをコピーするには --volume-only フラグを追加してください。コピー先のロケーションにボリュームが既に存在する場合は、コピーを更新するために --refresh フラグを使ってください。

同じストレージプール内でボリュームをコピーするにはコピー元とコピー先に同じプールを指定します。この場合コピー元とコピー先で別のボリューム名を指定する必要があります。

ボリュームをあるストレージプールから別のストレージプールにコピーする場合は、同じボリューム名を使うことも新しいボリューム名にリネームすることもできます。

カスタムストレージボリュームを移動またはリネームする

カスタムストレージボリュームを移動またはリネームする前に、それを利用している全てのインスタンスを **停止** する必要があります。

ストレージボリュームを移動またはリネームするには以下のコマンドを使用します。

```
lxc storage volume move <source_pool_name>/<source_volume_name> <target_pool_name>/
↪<target_volume_name>
```

変更前のプールと変更後のプールに同じプールを指定すると、同じストレージプールに置いたままボリューム名を変更できます。この場合変更前と変更後で別のボリューム名を指定する必要があります。

ボリュームをあるストレージプールから別のストレージプールに移動する場合は、同じボリューム名を使うことも新しいボリューム名にリネームすることもできます。

クラスタメンバー間でコピーまたは移動する

(ceph と ceph-fs を除く) ほとんどのストレージドライバではストレージボリュームはそれらが作成されたクラスタメンバー上だけに存在します。

あるクラスタメンバーから別のメンバーにカスタムストレージボリュームをコピーまたは移動するには --target と --destination-target フラグでそれぞれコピー/移動元のクラスタメンバーとコピー/移動先のクラスタメンバーを指定します。

プロジェクト間でコピーまたは移動する

別のプロジェクトにカスタムストレージボリュームをコピーまたは移動するには --target-project を追加してください。

LXD サーバー間でコピーまたは移動する

対象の各プールにリモートを指定することで異なる LXD サーバー間でカスタムストレージボリュームをコピーまたは移動できます。

```
lxc storage volume copy <source_remote>:<source_pool_name>/<source_volume_name> <target_
↪remote>:<target_pool_name>/<target_volume_name>
lxc storage volume move <source_remote>:<source_pool_name>/<source_volume_name> <target_
↪remote>:<target_pool_name>/<target_volume_name>
```

ネットワークのセットアップに応じて `--mode` フラグを追加して転送モードを選べます。

`pull` (**デフォルト**)
: コピー/移動先のサーバーに指定のストレージボリュームをプルするように指示します。

`push`
: コピー/移動元のサーバーからストレージボリュームをコピー/移動先のサーバーにプッシュします。

`relay`
: コピー/移動元のサーバーからストレージボリュームをローカルクライアントにプルし、それからコピー/移動先のサーバーにプッシュします。

インスタンスストレージプールを別のプールに移動する

インスタンスストレージプールを別のプールに移動するにはまずインスタンスが停止されていることを確認してください。次に以下のコマンドでインスタンスを別のプールに移動します。

```
lxc move <instance_name> --storage <target_pool_name>
```

3.6.6 カスタムストレージボリュームをバックアップするには

カスタムストレージボリュームをバックアップするには以下の方法があります。

- **スナップショットをバックアップに使う**
- **エクスポートファイルをバックアップに使用する**
- **カスタムストレージボリュームをコピーする**

どの方法を選ぶかはユースケースと使用しているストレージドライバによります。

一般的に、スナップショットは高速で空間効率がよい (ストレージドライバによります) ですが、ボリュームと同じストレージプール内に置かれますので信頼性はそれほど高くはありません。エクスポートファイルは別のディスクに保管できますので、より信頼性が高いです。これはボリュームを別のストレージプールに復元するのにも使えます。もし別のネットワークで接続された LXD サーバーがある場合、ボリュームをこの別のサーバーに定期的にコ

ピーすることで信頼性を高められますし、この方法はボリュームのスナップショットをバックアップするのにも使用できます。

注釈: カスタムストレージボリュームはインスタンスにアタッチされるかもしれませんが、インスタンスの一部ではありません。そのため、カスタムストレージボリュームの内容はインスタンスをバックアップする際に保存されません。ストレージボリュームのバックアップは別途行う必要があります。

スナップショットをバックアップに使う

スナップショットは指定した日時のストレージボリュームの状態を保存します。ボリュームを元の状態に簡単に戻せるようになります。スナップショットはボリュームと同じストレージプールに保存されます。

ほとんどのストレージドライバは最適化されたスナップショット作成をサポートします (**機能比較** 参照)。これらのドライバではスナップショットの作成は高速で空間効率も良いです。`dir` ドライバでは、スナップショット機能は利用できますが、効率的ではありません。`lvm` ドライバでは、スナップショットの作成は高速ですが、スナップショットの復元は thin-pool モードを使っているときのみ効率的です。

カスタムストレージボリュームのスナップショットを作成する

カスタムストレージボリュームのスナップショットを作成するには以下のコマンドを使用します。

```
lxc storage volume snapshot <pool_name> <volume_name> [<snapshot_name>]
```

既存のスナップショットを置き換えるにはスナップショット名に `--reuse` フラグを追加します。

`snapshots.expiry` 設定オプションが設定されていない限り、デフォルトではスナップショットは永遠に保持されます。特定のスナップショットをこの期限が切れても維持するには `--no-expiry` フラグを使用します。

スナップショットを閲覧、編集、削除する

ストレージのスナップショットを表示するには以下のコマンドを使用します。

```
lxc storage volume info <pool_name> <volume_name>
```

スナップショットの閲覧と編集はスナップショットを `<volume_name>/<snapshot_name>` で参照することでカスタムストレージボリュームと同様に行うことができます。

スナップショットの情報を表示するには以下のコマンドを使用します。

```
lxc storage volume show <pool_name> <volume_name>/<snapshot_name>
```

スナップショットを編集する (例えば、説明を追加したり保管期限を変更する) には以下のコマンドを使用します。

```
lxc storage volume edit <pool_name> <volume_name>/<snapshot_name>
```

スナップショットを削除するには以下のコマンドを使用します。

```
lxc storage volume delete <pool_name> <volume_name>/<snapshot_name>
```

カスタムストレージボリュームのスナップショットをスケジュールする

特定の日時にスナップショットを自動で作成するようにカスタムストレージボリュームを設定できます。そのためにはストレージボリュームに snapshots.schedule 設定オプションを設定してください (**ストレージボリュームを設定する** 参照)。

例えば、日次スナップショットを設定するには以下のコマンドを使用します。

```
lxc storage volume set <pool_name> <volume_name> snapshots.schedule @daily
```

毎日午前 6 時にスナップショットをとるように設定するには以下のコマンドを使用します。

```
lxc storage volume set <pool_name> <volume_name> snapshots.schedule "0 6 * * *"
```

定期的にスナップショット作成をスケジュールする場合は、自動破棄 (snapshots.expiry) とスナップショットの命名規則 (snapshots.pattern) の設定も検討してください。これらの設定オプションについてより詳しい情報は **ストレージドライバ** ドキュメントを参照してください。

カスタムストレージボリュームのスナップショットを復元する

カスタムストレージボリュームを任意のスナップショットの状態に復元できます。

このために、まずカスタムストレージボリュームを使用している全てのインスタンスを停止する必要があります。次に以下のコマンドを使用します。

```
lxc storage volume restore <pool_name> <volume_name> <snapshot_name>
```

スナップショットを同じストレージプール内あるいは別のストレージプール内 (リモートのストレージプールでも可) の新しいカスタムストレージボリュームに復元することもできます。このためには以下のコマンドを使用します。

```
lxc storage volume copy <source_pool_name>/<source_volume_name>/<source_snapshot_name>
↪<target_pool_name>/<target_volume_name>
```

エクスポートファイルをバックアップに使用する

カスタムストレージボリュームの完全な内容をスタンドアロンのファイルにエクスポートできます。このファイルは任意の場所に保管できます。最大の信頼性のためには、バックアップが失われたり破損されたりしないよう別のファイルシステム上にバックアップファイルを保管してください。

カスタムストレージボリュームをエクスポートする

圧縮されたファイル (例: /path/to/my-backup.tgz) にカスタムストレージボリュームをエクスポートするには以下のコマンドを使用します。

```
lxc storage volume export <pool_name> <volume_name> [<file_path>]
```

ファイルパスを指定しない場合、エクスポートされたファイルは作業ディレクトリに <instance name>.<extension> という名前（たとえば、my-container.tar.gz）で保存されます。

警告: 出力ファイル (<instance name>.<extension>, <instance name>.backup あるいは指定したファイルパス) が既に存在する場合、このコマンドは既存のファイルを警告なしに上書きします。

コマンドに以下のフラグを追加できます。

--compression
: デフォルトでは出力ファイルは gzip 圧縮を使います。別の圧縮アリゴリズム (例えば bzip2) を指定することもできますし --compression=none で圧縮をオフにすることもできます。

--optimized-storage
: ストレージプールが btrfs か zfs ドライバを使用している場合、 --optimized-storage フラグを指定すると個別のファイルのアーカイブの代わりにドライバ固有のバイナリー blob としてデータを保存できます。この場合、エクスポートされたファイルは同じストレージドライバを使うプールでのみ利用できます。

  ボリュームを最適化されたモードでエクスポートするのは個別のファイルをエクスポートするより通常速いです。スナップショットはメインボリュームからの差分としてエクスポートされるので、サイズが小さくなりアクセスが容易になります。

--volume-only
: デフォルトではエクスポートファイルはストレージボリュームの全てのスナップショットを含みます。このフラグを追加するとスナップショット無しでボリュームをエクスポートします。

エクスポートファイルからカスタムストレージボリュームを復元する

エクスポートファイル (例えば /path/to/my-backup.tgz) を新しいカスタムストレージボリュームとしてインポートできます。このためには以下のコマンドを使用します。

```
lxc storage volume import <pool_name> <file_path> [<volume_name>]
```

ボリューム名を指定しない場合、エクスポートしたストレージボリュームの元の名前が新しいボリュームの名前として使われます。指定したストレージプールにその名前のボリュームが既に (あるいは引き続き) 存在する場合、コマンドはエラーを返します。この場合、バックアップをインポートする前に既存のボリュームを削除するかインポート用に別のボリューム名を指定してください。

3.6.7 ストレージバケットとキーを管理するには

ストレージバケット を作成、設定、表示、リサイズするための手順およびストレージバケットキーを管理する方法については以下のセクションを参照してください。

S3 アドレスを設定する

S3 アドレスを設定することにより、ローカルストレージ（dir、btrfs、lvm、または zfs プール）上のストレージバケットを使用することが可能になります。これにより、S3 プロトコルを通じてバケットにアクセスできるようになります。

S3 アドレスを設定するには、*core.storage_buckets_address* サーバー設定オプションを設定します。例えば：

```
lxc config set core.storage_buckets_address :8555
```

ストレージバケットを管理する

ストレージバケットは S3 プロトコルを使って公開されるオブジェクトストレージを提供します。

カスタムストレージボリュームとは異なり、ストレージバケットはインスタンスに追加されるのではなく、それらの URL を通してアプリケーションから直接アクセスされます。

詳細は **ストレージバケット** を参照してください。

ストレージバケットを作成する

ストレージプール内にストレージバケットを作成するには、以下のコマンドを使用します。

```
lxc storage bucket create <pool_name> <bucket_name> [configuration_options...]
```

それぞれのドライバで利用可能なストレージバケット設定オプションの一覧については **ストレージドライバ** を参照してください。

クラスタメンバーにストレージバケットを追加するには `--target` フラグを追加してください。

```
lxc storage bucket create <pool_name> <bucket_name> --target=<cluster_member>␣
↪[configuration_options...]
```

注釈: ほとんどのストレージドライバでは、ストレージバケットはクラスタ間でリプリケートされず、作成されたメンバー上にのみ存在します。この挙動は `cephobject` ストレージプールでは異なります。 `cephobject` ではバケットはどのクラスタメンバーからも利用できます。

ストレージバケットを設定するには

各ストレージドライバで利用可能な設定オプションについては **ストレージドライバ** ドキュメントを参照してください。

ストレージバケットの設定オプションを設定するには以下のコマンドを使用します。

```
lxc storage bucket set <pool_name> <bucket_name> <key> <value>
```

例えば、バケットにクォータサイズを設定するには、以下のコマンドを使用します。

```
lxc storage bucket set my-pool my-bucket size 1MiB
```

以下のコマンドでストレージバケットの設定を編集することもできます。

```
lxc storage bucket edit <pool_name> <bucket_name>
```

ストレージバケットとそのキーを削除するには以下のコマンドを使用します。

```
lxc storage bucket delete <pool_name> <bucket_name>
```

ストレージバケットを表示するには

ストレージプール内の全ての利用可能なストレージバケットの一覧を表示し設定を確認できます。

ストレージプール内の全ての利用可能なストレージバケットを一覧表示するには、以下のコマンドを使用します。

```
lxc storage bucket list <pool_name>
```

特定のバケットの詳細情報を表示するには、以下のコマンドを使用します。

```
lxc storage bucket show <pool_name> <bucket_name>
```

ストレージバケットをリサイズするには

デフォルトではストレージバケットにはクォータは適用されません。

ストレージバケットクォータを設定するには、サイズを設定します。

```
lxc storage bucket set <pool_name> <bucket_name> size <new_size>
```

重要:

- ストレージバケットの拡大は通常は正常に動作します (ストレージプールが十分なストレージを持つ場合)。
- ストレージバケットを現在の使用量より縮小することはできません。

ストレージバケットキーを管理する

アプリケーションがストレージバケットにアクセスするためには **アクセスキー** と **シークレットキー** からなる S3 クレデンシャルを使う必要があります。特定のバケットに対して複数のセットのクレデンシャルを作成できます。

それぞれのクレデンシャルのセットにはキー名を設定します。キー名は参照のためだけに用いられ、アプリケーションがクレデンシャルを使用する際に提供する必要はありません。

それぞれのクレデンシャルのセットには **ロール** が設定されます。それはバケットにどの操作を実行できるかを指定します。

使用可能なロールは以下のとおりです。

- `admin` - バケットへのフルアクセス。
- `read-only` - バケットへの読み取り専用アクセス (一覧とファイルの取得のみ)。

バケットキー作成時にロールが指定されない場合、使用されるロールは `read-only` になります。

ストレージバケットキーを作成する

ストレージバケットにクレデンシャルのセットを作成するには、以下のコマンドを使用します。

```
lxc storage bucket key create <pool_name> <bucket_name> <key_name> [configuration_
↪options...]
```

ストレージバケットに特定のロールを持つクレデンシャルのセットを作成するには、以下のコマンドを使用します。

```
lxc storage bucket key create <pool_name> <bucket_name> <key_name> --role=admin␣
↪[configuration_options...]
```

これらのコマンドはランダムなクレデンシャルキーのセットを生成し表示します。

ストレージバケットキーを編集または削除するには

既存のバケットキーを編集するには以下のコマンドを使用します。

```
lxc storage bucket edit <pool_name> <bucket_name> <key_name>
```

既存のバケットキーを削除するには以下のコマンドを使用します。

```
lxc storage bucket key delete <pool_name> <bucket_name> <key_name>
```

ストレージバケットのキーを表示するには

既存のバケットに定義されているキーを表示するには以下のコマンドを使用します。

```
lxc storage bucket key list <pool_name> <bucket_name>
```

特定のバケットキーを表示するには以下のコマンドを使用します。

```
lxc storage bucket key show <pool_name> <bucket_name> <key_name>
```

3.6.8 ストレージドライバ

LXD はイメージ、インスタンスとカスタムボリュームを保管するのに以下のストレージドライバをサポートします。

ディレクトリ - dir

ディレクトリストレージドライバは基本的なバックエンドで通常のファイルとディレクトリ構造にデータを保管します。このドライバは素早くセットアップできディスク上のファイルを直接見ることができるので、テストには便利かもしれません。しかし、 LXD の操作はこのドライバ用には **最適化されていません**。

LXD の dir ドライバ

LXD の dir ドライバは完全に機能し、他のドライバと同じ機能セットを提供します。しかし、他のドライバよりは圧倒的に遅いです。これはインスタンス、スナップ、ショットを一瞬でコピーする代わりにイメージの解凍を行う必要があるためです。

作成時に (source 設定オプションを使って) 別途指定されてない限り、データは /var/snap/lxd/common/lxd/storage-pools/ (snap でインストールした場合) または /var/lib/lxd/storage-pools/ ディレクトリに保管されます。

クォータ

dir ドライバは ext4 か XFS 上で動作しファイルシステムレベルでプロジェクトのクォータが有効な場合にストレージのクォータをサポートします。

設定オプション

dir ドライバを使うストレージプールとこれらのプール内のストレージボリュームには以下の設定オプションが利用できます。

ストレージプール設定

キー	型	デフォルト値	説明
rsync.bwlimit	string	0 (no limit)	ストレージエンティティの転送に rsync を使う必要があるときにソケット I/O に指定する上限を設定
rsync.compression	bool	true	ストレージプールのマイグレーションの際に圧縮を使うかどうか
source	string	-	ブロックデバイスかループファイルかファイルシステムエントリのパス

Tip: これらの設定に加えて、ストレージボリューム設定のデフォルト値を設定できます。**ストレージボリュームのデフォルト値を変更する** を参照してください。

ストレージボリューム設定

キー	型	条件	デフォルト値	説明
security.shifted	bool	カスタムボリューム	volume.security.shifted と同じか false	ID シフトオーバーレイを有効にする (複数の分離されたインスタンスによるアタッチを許可する)
security.unmapped	bool	カスタムボリューム	volume.security.unmapped と同じか false	ボリュームの ID マッピングを無効にする
size	string	適切なドライバ	volume.size と同じ	ストレージボリュームのサイズ/クォータ
snapshots.expiry	string	カスタムボリューム	volume.snapshots.expiry と同じ	スナップショットをいつ削除するかを制御 (1M 2H 3d 4w 5m 6y のような式を期待)
snapshots.pattern	string	カスタムボリューム	volume.snapshots.pattern と同じか snap%d	スナップショットの名前を表す Pongo2 テンプレート文字列 (スケジュールされたスナップショットと名前無しのスナップショットで使用)[*1]
snapshots.schedule	string	カスタムボリューム	volume.snapshots.schedule と同じ	Cron 表記 (<minute> <hour> <dom> <month> <dow>)、またはスケジュールエイリアスのカンマ区切りリスト (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly)、または自動スナップショットを無効にする場合は空文字 (デフォルト)

ストレージバケット設定

ローカルのストレージプールドライバでストレージバケットを有効にし、 S3 プロトコル経由でアプリケーションがバケットにアクセスできるようにするには *core.storage_buckets_address* サーバー設定を調整する必要があります。

ストレージバケットは `dir` プール用の設定はありません。他のストレージプールドライバとは異なり、 `dir` ドライバは `size` 設定によるバケットクォータのサポートはありません。

Btrfs - `btrfs`

Btrfs (B-tree file system) は COW (copy-on-write) 原則に基づいたローカルファイルシステムです。COW はデータが修正された後に既存のデータを上書きするのではなく別のブロックに保管され、データ破壊のリスクが低くなることを意味します。他のファイルシステムと異なり、Btrfs はエクステントベースです。これはデータを連続したメモリ領域に保管することを意味します。

基本的なファイルシステムの機能に加えて、Btrfs は RAID、ボリューム管理、プーリング、スナップショット、チェックサム、圧縮、その他の機能を提供します。

Btrfs を使うにはマシンに `btrfs-progs` がインストールされているか確認してください。

用語

Btrfs ファイルシステムは**サブボリューム**を持つことができます。これはファイルシステムのメインツリーの名前をつけられたバイナリサブツリーでそれ自身の独立したファイルとディレクトリ階層を持ちます。*Btrfs* **スナップショット**は特殊なタイプのサブボリュームで別のサブボリュームの特定の状態をキャプチャーします。スナップショットは読み書き可または読み取り専用にできます。

[*1] `snapshots.pattern` オプションはスナップショット名をフォーマットする Pongo2 テンプレート文字列です。
スナップショット名にタイムスタンプを追加するには、Pongo2 コンテキスト変数 `creation_date` を使用します。スナップショット名に使用できない文字を含まないようにテンプレート文字列をフォーマットするようにしてください。例えば、 `snapshots.pattern` を `{{ creation_date|date:'2006-01-02_15-04-05' }}` に設定し、作成日時を秒の制度まで落として、スナップショットを命名するようにします。
名前の衝突を防ぐ別の方法はパターン内に `%d` プレースホルダを使うことです。最初のスナップショットでは、プレースホルダは `0` に置換されます。後続のスナップショットでは、既存のスナップショットが考慮され、プレースホルダの位置の最大の数を見つけます。この数が 1 増加されて新しい名前に使用されます。

LXD の btrfs ドライバ

LXD の btrfs ドライバはインスタンス、イメージ、スナップショットごとにサブボリュームを使用します。新しいエンティティを作成する際 (例えば、新しいインスタンスを起動する)、 Btrfs スナップショットを作成します。

Btrfs はブロックデバイスの保管をネイティブにはサポートしていません。このため、仮想マシンに Btrfs を使用する場合、 LXD は仮想マシンを格納するディスク上に巨大なファイルを作成します。このアプローチはあまり効率的ではなく、スナップショット作成時に問題を引き起こすかもしれません。

Btrfs はネストした LXD 環境内のコンテナ内部でストレージバックエンドとして使用できます。この場合、親のコンテナ自体は Btrfs を使う必要があります。しかし、ネストした LXD のセットアップは親から Btrfs のクォータは引き継がないことに注意してください (以下の **クォータ** 参照)。

クォータ

Btrfs は qgroups 経由でストレージクォータをサポートします。Btrfs qgroups は階層的ですが、新しいサブボリュームは親のサブボリュームの qgroups に自動的に追加されるわけではありません。これはユーザが設定されたクォータから逃れることができることは自明であることを意味します。このため、厳密なクォータが必要な場合は、別のストレージドライバを検討すべきです (例えば、refquotas ありの ZFS や LVM 上の Btrfs)。

クォータを使用する際は、 Btrfs のエクステントはイミュータブルであることを考慮に入れる必要があります。ブロックが書かれると、それらは新しいエクステントに現れます。古いエクステントはその上の全てのデータが参照されなくなるか上書きされるまで残ります。これはサブボリューム内で現在存在するファイルで使用されている合計容量がクォータより小さい場合でもクォータに達することがあり得ることを意味します。

注釈: この問題は Btrfs 上で仮想マシンを使用する際にもっともよく発生します。これは Btrfs サブボリューム上に生のディスクイメージを使用する際のランダムな I/O の性質のためです。

このため、仮想マシンには Btrfs ストレージプールは決して使うべきではありません。

どうしても仮想マシンに Btrfs ストレージプールを使う必要がある場合、インスタンスのルートディスクの *size.state* をルートディスクのサイズの 2 倍に設定してください。この設定により、ディスクイメージファイルの全てのブロックが qgroup クォータに達すること無しに上書きできるようになります。*btrfs.mount_options=compress-force* ストレージプールオプションでもこのシナリオを回避できます。圧縮を有効にすることの副作用で最大のエクステントサイズを縮小しブロックの再書き込みが 2 倍のストレージを消費しないようになるからです。しかし、これはストレージプールのオプションなので、プール上の全てのボリュームに影響します。

設定オプション

btrfs ドライバを使うストレージプールとこれらのプール内のストレージボリュームには以下の設定オプションが利用できます。

ストレージプール設定

キー	型	デフォルト値	説明
btrfs.mount_opt	string	user_subvol_rm_allowed	ブロックデバイスのマウントオプション
size	string	自動 (空きディスクスペースの 20%, >= 5 GiB and <= 30 GiB)	ループベースのプールを作成する際のストレージプールのサイズ (バイト単位、接尾辞のサポートあり、増やすとストレージプールのサイズを拡大)
source	string	-	既存のブロックデバイス、ループファイル、あるいは Btrfs サブボリュームのパス
source.wipe	bool	false	ストレージプールを作成する前に source で指定されたブロックデバイスの中身を消去する

Tip: これらの設定に加えて、ストレージボリューム設定のデフォルト値を設定できます。 **ストレージボリュームのデフォルト値を変更する** を参照してください。

ストレージボリューム設定

キー	型	条件	デフォルト値	説明
securi shifte	bool	カスタムボリューム	volume. security. shifted と同じか false	ID シフトオーバーレイを有効にする (複数の分離されたインスタンスによるアタッチを許可する)
securi unmapp	bool	カスタムボリューム	volume. security. unmapped と同じか false	ボリュームへの id マッピングを無効にする
size	string	適切なドライバ	volume. size と同じ	ストレージボリュームのサイズ/クォータ
snapsh expiry	string	カスタムボリューム	volume. snapshots. expiry と同じ	スナップショットをいつ削除するかを制御 (1M 2H 3d 4w 5m 6y のような式を期待)
snapsh patter	string	カスタムボリューム	volume. snapshots. pattern と同じか snap%d	スナップショットの名前を表す Pongo2 テンプレート文字列 (スケジュールされたスナップショットと名前無しのスナップショットで使用)[*1]
snapsh schedu	string	カスタムボリューム	volume. snapshots. schedule と同じ	Cron 表記 (<minute> <hour> <dom> <month> <dow>)、またはスケジュールエイリアスのカンマ区切りリスト (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly)、または自動スナップショットを無効にする場合は空文字 (デフォルト)

[*1] snapshots.pattern オプションはスナップショット名をフォーマットする Pongo2 テンプレート文字列です。

スナップショット名にタイムスタンプを追加するには、Pongo2 コンテキスト変数 creation_date を使用します。スナップショット名に使用できない文字を含まないようにテンプレート文字列をフォーマットするようにしてください。例えば、 snapshots.pattern を {{ creation_date|date:'2006-01-02_15-04-05' }} に設定し、作成日時を秒の制度まで落として、スナップショットを命名するようにします。

名前の衝突を防ぐ別の方法はパターン内に %d プレースホルダを使うことです。最初のスナップショットでは、プレースホルダは 0 に置換されます。後続のスナップショットでは、既存のスナップショットが考慮され、プレースホルダの位置の最大の数を見つけます。こ

ストレージバケット設定

ローカルのストレージプールドライバでストレージバケットを有効にし、 S3 プロトコル経由でアプリケーションがバケットにアクセスできるようにするには *core.storage_buckets_address* サーバー設定を調整する必要があります。

キー	型	条件	デフォルト値	説明
size	string	適切なドライバ	volume.size と同じ	ストレージバケットのサイズ/クォータ

LVM - lvm

LVM (Logical Volume Manager) はファイルシステムというよりストレージマネージメントフレームワークです。これは物理ストレージデバイスを管理するのに使用され、複数のストレージボリュームを作成し、配下の物理ストレージデバイスを使用し仮想化できるようにします。

この過程で物理ストレージをオーバーコミットすることが可能で、全ての利用可能なストレージが同時に使用されるわけではないシナリオに対して柔軟性を提供できることに注意してください。

LVM を使用するにはマシン上に lvm2 がインストールされていることを確認してください。

用語

LVM は複数の物理ストレージデバイスを組み合わせて **ボリュームグループ** にすることができます。その後このボリュームグループから異なるタイプの **論理ボリューム** を割り当てることができます。

サポートされるボリュームタイプの 1 つに *thin pool* があります。これは許可された最大サイズの合計は利用可能な物理ストレージより大きいような薄くプロビジョンされたボリュームを作成することでリソースをオーバーコミットすることを可能にします。別のタイプは **ボリュームスナップショット** でこれは論理ボリュームの特定の状態をキャプチャーします。

の数が 1 増加されて新しい名前に使用されます。

LXD の lvm ドライバ

LXD の lvm ドライバはイメージに論理ボリュームを、インスタンスとスナップショットにボリュームスナップショットを使用します。

LXD はボリュームグループを完全制御できると想定しています。このため、LXD が所有しないファイルシステムエンティティは LXD が消してしまうかもしれないので、LVM ボリュームグループ内に置くべきではありません。しかし、既存のボリュームグループを再利用する必要がある場合 (例えば、あなたの環境ではボリュームグループが 1 つしかない場合)、*lvm.vg.force_reuse* を設定することでこれは可能です。

デフォルトでは LVM ストレージプールは LVM thin pool を使用しその中に全ての LXD ストレージエンティティ (イメージ、インスタンス、カスタムボリューム) の論理ボリュームを作成します。この挙動はプール作成時に *lvm.use_thinpool* を false に設定することで変更できます。この場合、LXD はスナップショットでない全てのストレージエンティティに "通常の" 論理ボリュームを使用します。これは深刻なパフォーマンスの低下とディスクの空き容量の低下を lvm ドライバに必然的にもたらすことに注意してください (スピードとストレージ使用量の両面で dir ドライバに近くなります)。この理由は thin pool でない論理ボリュームがスナップショットのスナップショットをサポートしないため、ほとんどのストレージ操作が rsync の使用にフォールバックするためです。さらに、 thin でないスナップショットは作成時に最大のサイズのストレージを予約しなければならないため、 thin スナップショットよりもはるかに大容量のストレージを使用するからです。このため、このオプションはどうしても必要なユースケースの場合にのみ選択すべきです。

インスタンスの入れ替わりが激しい環境 (例えば、継続的インテグレーション) では、LXD の操作が遅くなるのを回避するため /etc/lvm/lvm.conf 内のバックアップの retain_min と retain_days 設定を調整すべきです。

設定オプション

lvm ドライバを使うストレージプールとこれらのプール内のストレージボリュームには以下の設定オプションが利用できます。

ストレージプール設定

キー	型	デフォルト値	説明
lvm.thinpool_nam	string	LXDThinPool	ボリュームが作成される thin pool
lvm.thinpool_met	string	0 (auto)	thin pool メタデータボリュームのサイズ (デフォルトは LVM が適切なサイズを計算)
lvm.use_thinpool	bool	true	ストレージプールは論理ボリュームに thin pool を使うかどうか
lvm.vg.force_reuse	bool	false	既存の空でないボリュームグループの使用を強制
lvm.vg_name	string	プールの名前	作成するボリュームグループ名
rsync.bwlimit	string	0 (no limit)	ストレージエンティティーの転送に rsync を使う場合、ソケット I/O に設定する上限を指定
rsync.compression	bool	true	ストレージプールをマイグレートする際に圧縮を使用するかどうか
size	string	自動 (空きディスクスペースの 20%, >= 5 GiB and <= 30 GiB)	ループベースのプールを作成する際のストレージプールのサイズ (バイト単位、接尾辞のサポートあり、増やすとストレージプールのサイズを拡大)
source	string	-	既存のブロックデバイスかループファイルか LVM ボリュームグループのパス
source.wipe	bool	false	ストレージプールを作成する前に source で指定されたブロックデバイスの中身を消去する

Tip: これらの設定に加えて、ストレージボリューム設定のデフォルト値を設定できます。 **ストレージボリュームのデフォルト値を変更する** を参照してください。

ストレージボリューム設定

キー	型	条件	デフォルト値	説明
block.filesy	string		volume.block.filesystem と同じ	ストレージボリュームのファイルシステム: btrfs, ext4 または xfs (未指定の場合 ext4)
block.mount_	string		volume.block.mount_optior と同じ	block-backed なファイルシステムボリュームのマウントオプション
lvm.stripes	string		volume.lvm.stripes と同じ	新しいボリューム (あるいは thin pool ボリューム) に使用するストライプ数
lvm.stripes.size	string		volume.lvm.stripes.size と同じ	使用するストライプのサイズ (最低 4096 バイトで 512 バイトの倍数を指定)
securi shifte	bool	カスタムボリューム	volume.security.shifted と同じか false	ID シフトオーバーレイを有効にする (複数の分離されたインスタンスによるアタッチを許可する)
securi unmapp	bool	カスタムボリューム	volume.security.unmapped と同じか false	ボリュームへの ID マッピングを無効にする
size	string		volume.size と同じ	ストレージボリュームのサイズ/クォータ
snapsh expiry	string	カスタムボリューム	volume.snapshots.expiry と同じ	スナップショットをいつ削除するかを制御 (1M 2H 3d 4w 5m 6y のような式を期待)
snapsh patter	string	カスタムボリューム	volume.snapshots.pattern と同じか snap%d	スナップショットの名前を表す Pongo2 テンプレート文字列 (スケジュールされたスナップショットと名前無しのスナップショットで使用)[*1]
snapsh schedu	string	カスタム	volume.	Cron 表記 (<minute> <hour> <dom> <month> <dow>)、またはスケジュールエイリアスのカンマ区切りリスト (@hourly, @daily,

ストレージバケット設定

ローカルのストレージプールドライバでストレージバケットを有効にし、 S3 プロトコル経由でアプリケーションがバケットにアクセスできるようにするには *core.storage_buckets_address* サーバー設定を調整する必要があります。

キー	型	条件	デフォルト値	説明
size	string	適切なドライバ	volume.size と同じ	ストレージバケットのサイズ/クォータ

ZFS - zfs

ZFS (Zettabyte file system) は物理ボリューム管理とファイルシステムを兼ね備えています。ZFS のインストールは一連のストレージデバイスに広がることができ非常にスケーラブルで、ディスクを追加してストレージプールの空き容量を即座に拡大できます。

ZFS はブロックベースのファイルシステムで、あらゆる操作を検証、確認、訂正するのにチェックサムを使用することでデータ破壊から守ります。十分な速度で動作するためには、この機構には強力な環境と大量の RAM が必要です。

さらに、 ZFS はスナップショット、リプリケーション、RAID 管理、コピー・オン・ライトのクローン、圧縮、その他の機能を提供します。

ZFS を使用するにはマシンに zfsutils-linux をインストールしていることを確認してください。

用語

ZFS は物理ストレージデバイスに基づいた論理ユニットを作成します。これらの論理ユニットは *ZFS pools* または *zpools* と呼ばれます。さらにそれぞれの zpool は複数の **データセット** に分割されます。これらの**データセット**は以下の異なるタイプがあります。

- *ZFS* **ファイルシステム** はパーティションまたはマウントされたファイルシステムとして扱えます。
- *ZFS* **ボリューム** はブロックデバイスを表します。

[*1] snapshots.pattern オプションはスナップショット名をフォーマットする Pongo2 テンプレート文字列です。
スナップショット名にタイムスタンプを追加するには、Pongo2 コンテキスト変数 creation_date を使用します。スナップショット名に使用できない文字を含まないようにテンプレート文字列をフォーマットするようにしてください。例えば、 snapshots.pattern を {{ creation_date|date:'2006-01-02_15-04-05' }} に設定し、作成日時を秒の制度まで落として、スナップショットを命名するようにします。
名前の衝突を防ぐ別の方法はパターン内に %d プレースホルダを使うことです。最初のスナップショットでは、プレースホルダは 0 に置換されます。後続のスナップショットでは、既存のスナップショットが考慮され、プレースホルダの位置の最大の数を見つけます。この数が 1 増加されて新しい名前に使用されます。

- *ZFS* **スナップショット** は ZFS ファイルシステムまたは ZFS ボリュームの特定の状態をキャプチャーします。ZFS スナップショットは読み取り専用です。
- *ZFS* **クローン** は ZFS スナップショットの書き込み可能なコピーです。

LXD の zfs ドライバ

LXD の zfs ドライバは ZFS **ファイルシステム**と ZFS ボリュームをイメージとカスタムストレージボリュームに使用し、ZFS スナップショットとクローンをイメージからのインスタンス作成とインスタンスとカスタムボリュームスナップショットに使用します。デフォルトでは LXD は ZFS プール作成時に圧縮を有効にします。

LXD は ZFS プールと**データセット**を完全制御できると想定します。このため、ZFS プールまたは**データセット**内に、LXD が所有しないファイルシステムエンティティは、決して置くべきではありません。LXD が消してしまうかもしれないからです。

ZFS のコピー・オン・ライトが動作する仕組みのため、親の ZFS ファイルシステムは全ての子供がいなくなるまで削除できません。その結果、LXD は削除されたがまだ参照されている全てのオブジェクトを自動的にリネームします。それらのオブジェクトは全ての参照がいなくなりオブジェクトが安全に削除できるようになるまでランダムな deleted/ パスに保管されます。この方法はスナップショットの復元に予期しない結果をもたらすかもしれないことに注意してください。下記の **制限** を参照してください。

ZFS 0.8 以降の上で新しく作成された全てのプールで LXD はトリミングサポートを自動的に有効化します。これはコントローラによるより良いブロックの再利用を可能にすることで SSD の寿命を伸ばし、ループバックの ZFS プールを使用する際にはルートファイルシステム上の容量を解放できるようにもします。ZFS の 0.8 より前のバージョンを稼働していてトリミングを有効にしたい場合は、少なくともバージョン 0.8 にアップグレードしてください。そして以下のコマンドを実行し、今後作成されるプールにトリミングが自動的に有効化され、現在未使用のスペースの全てがトリムされることを確認してください。

```
zpool upgrade ZPOOL-NAME
zpool set autotrim=on ZPOOL-NAME
zpool trim ZPOOL-NAME
```

制限

zfs ドライバには以下の制限があります。

プールの一部を委譲する
: ZFS はプールの一部をコンテナユーザに委譲することをサポートしていません。ZFS のアップストリームではこの機能を提供すべくアクティブに作業中です。

古いスナップショットからの復元
: ZFS は最新ではないスナップショットからの復元をサポートしていません。ですが、古いスナップショットから新しいインスタンスを作成することはできます。この方法は特定のスナップショットが必要なものを含

んでいるかを確認することを可能にします。正しいスナップショットを決定したら **指定より新しいスナップショットを削除** して必要なスナップショットが最新になるようにして復元できるようにします。

別の方法として、復元中により新しいスナップショットを自動的に破棄するように LXD を設定することもできます。そのためにはボリュームの *zfs.remove_snapshots* (あるいはプール内の全てのボリュームのストレージプールの対応する volume.zfs.remove_snapshots 設定) を設定します。

しかし、 *zfs.clone_copy* が true に設定される場合は、インスタンスのコピーは ZFS のスナップショットも使用することに注意してください。この場合は、スナップショットの全ての子孫を削除すること無しに、インスタンスを最後のコピーの前に取られたスナップショットに復元できません。この選択肢が選べない場合、欲しいスナップショットを新しいインスタンスにコピーしてから古いインスタンスを削除することはできます。しかし、インスタンスが持っていたであろう他の全てのスナップショットは失うことになります。

I/O クォータを観測する

I/O クォータは ZFS ファイルシステムに大きな影響は与えません。これは ZFS は (SPL を使用した) Solaris モジュールの移植でありネイティブな Linux ファイルシステムではないためで、 I/O の制限はネイティブ Linux ファイルシステムに適用されるからです。

クォータ

ZFS は quota と refquota という 2 種類の異なるクォータのプロパティを提供します。quota はスナップショットとクローンを含むデータセットの合計サイズを制限します。refquota はスナップショットとクローンは含まずデータセット内のデータのサイズだけを制限します。

デフォルトでは、ストレージボリュームにクォータを設定する際は LXD は quota プロパティを使用します。代わりに refquota プロパティを使用したい場合はボリュームの *zfs.use_refquota* 設定 (あるいはプール内の全てのボリュームのストレージプールの対応する volume.zfs.use_refquota 設定) を設定します。

また *zfs.use_reserve_space* (または volume.zfs.use_reserve_space) 設定を to use ZFS の reservation または refreservation を quota または refquota と使用するために設定することもできます。

設定オプション

zfs ドライバを使うストレージプールとこれらのプール内のストレージボリュームには以下の設定オプションが利用できます。

ストレージプール設定

キー	型	デフォルト値	説明
size	string	自動 (空きディスクスペースの 20%, >= 5 GiB and <= 30 GiB)	ループベースのプールを作成する際のストレージプールのサイズ (バイト単位、接尾辞のサポートあり、増やすとストレージプールのサイズを拡大)
source	string	-	既存のブロックデバイスかループファイルか ZFS データセット/プールのパス
source.wipe	bool	false	ストレージプールを作成する前に source で指定されたブロックデバイスの中身を消去する
zfs.clone_	string	true	Boolean の文字列を指定した場合は ZFS のフル **データセット**コピーの代わりに軽量なクローンを使うかどうかを制御し、 rebase という文字列を指定した場合は初期イメージをベースにコピーします。
zfs.export	bool	true	アンマウントの実行中に zpool のエクスポートを無効にする
zfs.pool_n	string	プールの名前	zpool 名

Tip: これらの設定に加えて、ストレージボリューム設定のデフォルト値を設定できます。 **ストレージボリュームのデフォルト値を変更する** を参照してください。

ストレージボリューム設定

キー	型	条件	デフォルト値	説明
block.filesys	string	zfs.block_m が有効	volume.block.filesystem と同じ	ストレージボリュームのファイルシステム: btrfs, ext4 または xfs (未指定の場合 ext4)
block.mount_c	string	zfs.block_m が有効	volume.block.mount_option と同じ	block-backed なファイルシステムボリュームのマウントオプション
securit shifte	bool	カスタムボリューム	volume.security.shifted と同じか false	ID シフトオーバーレイを有効にする (複数の分離されたインスタンスによるアタッチを許可する)
securit unmappe	bool	カスタムボリューム	volume.security.unmapped と同じか false	ボリュームの ID マッピングを無効にする
size	string		volume.size と同じ	ストレージボリュームのサイズ/クォータ
snapsho expiry	string	カスタムボリューム	volume.snapshots.expiry と同じ	スナップショットをいつ削除するかを制御 (1M 2H 3d 4w 5m 6y のような式を期待)
snapsho pattern	string	カスタムボリューム	volume.snapshots.pattern と同じか snap%d	スナップショットの名前を表す Pongo2 テンプレート文字列 (スケジュールされたスナップショットと名前無しのスナップショットで使用)[*1]
snapsho schedul	string	カスタムボリューム	snapshots.schedule と同じ	Cron 表記 (<minute> <hour> <dom> <month> <dow>)、またはスケジュールエイリアスのカンマ区切りリスト (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly)、または自動スナップショットを無効にする場合は空文字 (デフォルト)
zfs.blocksi	string		volume.zfs.blocksize と同じ	ZFS ブロックのサイズを 512〜16MiB の範囲で指定します (2 の累乗でなければなりません)。ブロックボリュームでは、より大きな値が設定されていても、最大値の 128KiB が使用されます。
zfs.block_m	bool		volume.zfs.	dataset よりもフォーマットした zvol を使うかどうか

ストレージバケット設定

ローカルのストレージプールドライバでストレージバケットを有効にし、 S3 プロトコル経由でアプリケーションがバケットにアクセスできるようにするには *core.storage_buckets_address* サーバー設定を調整する必要があります。

キー	型	条件	デフォルト値	説明
size	string	適切なドライバ	volume.size と同じ	ストレージバケットのサイズ/クォータ

Ceph RBD - ceph

Ceph はオープンソースのストレージプラットフォームで、データを RADOS (Reliable Autonomic Distributed Object Store) に基づいたストレージクラスタ内に保管します。非常にスケーラブルで、単一障害点がない分散システムであり非常に信頼性が高いです。

Tip: ベーシックな Ceph クラスタを素早く構築したい場合、MicroCeph をチェックしてみてください。

Ceph はブロックストレージ用とファイルシステム用に異なるコンポーネントを提供します。

Ceph RBD (RADOS Block Device) はデータとワークロードを Ceph クラスタに分散する Ceph のブロックストレージコンポーネントです。これは薄いプロビジョニングを使用し、リソースをオーバーコミットできることを意味します。

用語

Ceph は保管するデータに **オブジェクト** という用語を使用します。データを保存と管理する責任を持つデーモンは *Ceph OSD (Object Storage Daemon)* です。Ceph のストレージは **プール** に分割されます。これはオブジェクトを保管する論理的なパーティションです。これらは **データプール**, **ストレージプール**, *OSD* **プール** とも呼ばれます。

Ceph ブロックデバイスは *RBD* **イメージ** とも呼ばれ、これらの RBD イメージの **スナップショット** と **クローン** を作成できます。

[*1] snapshots.pattern オプションはスナップショット名をフォーマットする Pongo2 テンプレート文字列です。
スナップショット名にタイムスタンプを追加するには、Pongo2 コンテキスト変数 creation_date を使用します。スナップショット名に使用できない文字を含まないようにテンプレート文字列をフォーマットするようにしてください。例えば、 snapshots.pattern を {{ creation_date|date:'2006-01-02_15-04-05' }} に設定し、作成日時を秒の制度まで落として、スナップショットを命名するようにします。
名前の衝突を防ぐ別の方法はパターン内に %d プレースホルダを使うことです。最初のスナップショットでは、プレースホルダは 0 に置換されます。後続のスナップショットでは、既存のスナップショットが考慮され、プレースホルダの位置の最大の数を見つけます。この数が 1 増加されて新しい名前に使用されます。

LXD の ceph ドライバ

注釈: Ceph RBD ドライバを使用するには `ceph` と指定する必要があります。これは少し誤解を招く恐れがあります。 Ceph の全ての機能ではなく Ceph RBD (ブロックストレージ) の機能しか使わないからです。コンテントタイプ `filesystem` (イメージ、コンテナとカスタムファイルシステムボリューム) のストレージボリュームには `ceph` ドライバは Ceph RDB イメージをその上にファイルシステムがある状態で使用します (*`block.filesystem`* 参照)。

別の方法として、コンテントタイプ `filesystem` でストレージボリュームを作成するのに *CephFS* を使用することもできます。

他のストレージドライバとは異なり、このドライバはストレージシステムをセットアップはせず、既に Ceph クラスタをインストール済みであると想定します。

このドライバはリモートのストレージを提供するという意味でも他のドライバとは異なる振る舞いをします。結果として、内部ネットワークに依存し、ストレージへのアクセスはローカルのストレージより少し遅くなるかもしれません。一方で、リモートのストレージを使うことはクラスタ構成では大きな利点があります。これはストレージプールを同期する必要なしに、全てのクラスタメンバーが同じ内容を持つ同じストレージプールにアクセスできるからです。

LXD 内の `ceph` ドライバはイメージ、スナップショットに RBD イメージを使用し、インスタンスとスナップショットを作成するのにクローンを使用します。

LXD は OSD ストレージプールに対して完全制御できることを想定します。このため、LXD OSD ストレージプール内に LXD が所有しないファイルシステムエンティティは LXD が消してしまうかもしれないので決して置くべきではありません。

Ceph RBD 内で copy-on-write が動作する方法のため、親の RBD イメージは全ての子がいなくなるまで削除できません。結果として LXD は削除されたがまだ参照されているオブジェクトを自動的にリネームします。そのようなオブジェクトは全ての参照がいなくなりオブジェクトが安全に削除できるようになるまで `zombie_` 接頭辞をつけて維持されます。

制限

`ceph` ドライバには以下の制限があります。

インスタンス間でのカスタムボリュームの共有 コ

ンテントタイプ `filesystem` のカスタムストレージボリュームは異なるクラスタメンバーの複数のインスタンス間で通常は共有できます。しかし、 Ceph RBD ドライバは RBD イメージ上にファイルシステムを置くことでコンテントタイプ `filesystem` のボリュームを「シミュレート」しているため、カスタムストレージボリュームは一度に 1 つのインスタンスにしか割り当てできません。コンテントタイプ `filesystem` のカスタムボリュームを共有する必要がある場合は代わりに *CephFS - cephfs* ドライバを使用してください。

複数インストールされた LXD 間で OSD ストレージプールの共有 複数インストールされた LXD 間で同じ OSD ストレージプールを共有することはサポートされていません。

タイプ "erasure" の OSD プールの使用 タイプ "erasure" の OSD プールを使用するには事前に OSD プールを作成する必要があります。さらにタイプ "replicated" の別の OSD プールを作成する必要もあります。これはメタデータを保管するのに使用されます。これは Ceph RBD が omap をサポートしないために必要となります。どのプールが "erasure coded" であるかを指定するために *ceph.osd.data_pool_name* 設定オプションをイレージャーコーディングされたプールの名前に設定し *source* 設定オプションをリプリケートされたプールの名前に設定します。

設定オプション

ceph ドライバを使うストレージプールとこれらのプール内のストレージボリュームには以下の設定オプションが利用できます。

ストレージプール設定

キー	型	デフォルト値	説明
ceph.cluster_name	string	ceph	新しいストレージプールを作成する Ceph クラスタの名前
ceph.osd.data_pool_name	string	-	OSD data pool の名前
ceph.osd.pg_num	string	32	OSD ストレージプール用の placement グループの数
ceph.osd.pool_name	string	プールの名前	OSD ストレージプールの名前
ceph.rbd.clone_copy	bool	true	フルのデータセットコピーではなく RBD のライトウェイトクローンを使うかどうか
ceph.rbd.du	bool	true	停止したインスタンスのディスク使用データを取得するのに RBD du を使用するかどうか
ceph.rbd.features	string	layering	ボリュームで有効にする RBD の機能のカンマ区切りリスト
ceph.user.name	string	admin	ストレージプールとボリュームの作成に使用する Ceph ユーザー
source	string	-	使用する既存の OSD ストレージプール
volatile.pool.pristine	string	true	プールが作成時に空かどうか

Tip: これらの設定に加えて、ストレージボリューム設定のデフォルト値を設定できます。 **ストレージボリュームのデフォルト値を変更する** を参照してください。

ストレージボリューム設定

キー	型	条件	デフォルト値	説明
block.filesy	string		volume.block.filesystem と同じ	ストレージボリュームのファイルシステム: btrfs, ext4 または xfs (未指定の場合 ext4)
block.mount_	string		volume.block.mount_optio と同じ	block-backed なファイルシステムボリュームのマウントオプション
securi shifte	bool	カスタムボリューム	volume.security.shifted と同じか false	ID シフトオーバーレイを有効にする (複数の分離されたインスタンスによるアタッチを許可する)
securi unmapp	bool	カスタムボリューム	volume.security.unmapped と同じか false	ボリュームの ID マッピングを無効にする
size	string		volume.size と同じ	ストレージボリュームのサイズ/クォータ
snapsh expiry	string	カスタムボリューム	volume.snapshots.expiry と同じ	スナップショットをいつ削除するかを制御 (1M 2H 3d 4w 5m 6y のような式を期待)
snapsh patter	string	カスタムボリューム	volume.snapshots.pattern と同じか snap%d	スナップショットの名前を表す Pongo2 テンプレート文字列 (スケジュールされたスナップショットと名前無しのスナップショットで使用)[*1]
snapsh schedu	string	カスタムボリューム	volume.snapshots.schedule と同じ	Cron 表記 (<minute> <hour> <dom> <month> <dow>)、またはスケジュールエイリアスのカンマ区切りリスト (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly)、または自動スナップショットを無効にする場合は空文字 (デフォルト)

CephFS - cephfs

Ceph はオープンソースのストレージプラットフォームで、データを RADOS に基づいたストレージクラスタ内に保管します。非常にスケーラブルで、単一障害点がない分散システムであり非常に信頼性が高いです。

Tip: ベーシックな Ceph クラスタを素早く構築したい場合、MicroCeph をチェックしてみてください。

Ceph はブロックストレージ用とファイルシステム用に異なるコンポーネントを提供します。

CephFS (Ceph File System) は堅牢でフル機能の POSIX 互換の分散ファイルシステムを提供する Ceph のファイルシステムコンポーネントです。内部的には ファイルを Ceph オブジェクトにマップし、ファイルのメタデータ (例えば、ファイルの所有権、ディレクトリパス、アクセス権限) を別のデータプールに保管します。

用語

Ceph は保管するデータに **オブジェクト** という用語を使用します。データを保存と管理する責任を持つデーモンは *Ceph OSD* です。Ceph のストレージは **プール** に分割されます。これはオブジェクトを保管する論理的なパーティションです。これらは **データプール**, **ストレージプール**, *OSD* **プール** とも呼ばれます。

CephFS **ファイルシステム** は 2 つの OSD ストレージプールから構成され、ひとつは実際のデータ、もうひとつはファイルメタデータに使用されます。

LXD の cephfs ドライバ

注釈: `cephfs` ドライバはコンテントタイプ `filesystem` のカスタムストレージボリュームにのみ使用できます。

他のストレージボリュームには *Ceph* ドライバを使用してください。そのドライバはコンテントタイプ `filesystem` のカスタムストレージボリュームにも使用できますが、 Ceph RBD イメージを使って実装しています。

[*1] `snapshots.pattern` オプションはスナップショット名をフォーマットする Pongo2 テンプレート文字列です。
スナップショット名にタイムスタンプを追加するには、Pongo2 コンテキスト変数 `creation_date` を使用します。スナップショット名に使用できない文字を含まないようにテンプレート文字列をフォーマットするようにしてください。例えば、 `snapshots.pattern` を `{{ creation_date|date:'2006-01-02_15-04-05' }}` に設定し、作成日時を秒の制度まで落として、スナップショットを命名するようにします。
名前の衝突を防ぐ別の方法はパターン内に `%d` プレースホルダを使うことです。最初のスナップショットでは、プレースホルダは `0` に置換されます。後続のスナップショットでは、既存のスナップショットが考慮され、プレースホルダの位置の最大の数を見つけます。この数が 1 増加されて新しい名前に使用されます。

他のストレージドライバとは異なり、このドライバはストレージシステムをセットアップはせず、既に Ceph クラスタをインストール済みであると想定します。

使用したい CephFS ファイルシステムは事前に作成する必要があり *source* オプションで指定する必要があります。

このドライバはリモートのストレージを提供するという意味でも他のドライバとは異なる振る舞いをします。結果として、内部ネットワークに依存し、ストレージへのアクセスはローカルのストレージより少し遅くなるかもしれません。一方で、リモートのストレージを使うことはクラスタ構成では大きな利点があります。これはストレージプールを同期する必要なしに、全てのクラスタメンバーが同じ内容を持つ同じストレージプールにアクセスできるからです。

LXD は OSD ストレージプールに対して完全制御できることを想定します。このため、LXD OSD ストレージプール内に LXD が所有しないファイルシステムエンティティは LXD が消してしまうかもしれないので決して置くべきではありません。

LXD の `cephfs` ドライバはサーバー側でスナップショットが有効な場合はスナップショットをサポートします。

設定オプション

`cephfs` ドライバを使うストレージプールとこれらのプール内のストレージボリュームには以下の設定オプションが利用できます。

ストレージプール設定

キー	型	デフォルト値	説明
`cephfs.cluster_name`	string	`ceph`	CephFS ファイルシステムを含む Ceph クラスタの名前
`cephfs.fscache`	bool	`false`	カーネルの `fscache` と `cachefilesd` を使用するか
`cephfs.path`	string	`/`	CephFS をマウントするベースのパス
`cephfs.user.name`	string	`admin`	使用する Ceph のユーザー
`source`	string	-	使用する既存の CephFS ファイルシステムかファイルシステムパス
`volatile.pool.pristine`	string	`true`	作成時に CephFS ファイルシステムが空だったか

Tip: これらの設定に加えて、ストレージボリューム設定のデフォルト値を設定できます。 **ストレージボリュームのデフォルト値を変更する** を参照してください。

ストレージボリューム設定

キー	型	条件	デフォルト値	説明
securi shifte	bool	カスタムボリューム	volume. security. shifted と同じか false	ID シフトオーバーレイを有効にする (複数の分離されたインスタンスによるアタッチを許可する)
securi unmapp	bool	カスタムボリューム	volume. security. unmapped と同じか false	ボリュームの ID マッピングを無効にする
size	string	適切なドライバ	volume. size と同じ	ストレージボリュームのサイズ/クォータ
snapsh expiry	string	カスタムボリューム	volume. snapshots. expiry と同じ	スナップショットをいつ削除するかを制御 (1M 2H 3d 4w 5m 6y のような式を期待)
snapsh patter	string	カスタムボリューム	volume. snapshots. pattern と同じか snap%d	スナップショットの名前を表す Pongo2 テンプレート文字列 (スケジュールされたスナップショットと名前無しのスナップショットで使用)[*1]
snapsh schedu	string	カスタムボリューム	volume. snapshots. schedule と同じ	Cron 表記 (<minute> <hour> <dom> <month> <dow>)、またはスケジュールエイリアスのカンマ区切りリスト (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly)、または自動スナップショットを無効にする場合は空文字 (デフォルト)

[*1] snapshots.pattern オプションはスナップショット名をフォーマットする Pongo2 テンプレート文字列です。

スナップショット名にタイムスタンプを追加するには、Pongo2 コンテキスト変数 creation_date を使用します。スナップショット名に使用できない文字を含まないようにテンプレート文字列をフォーマットするようにしてください。例えば、 snapshots.pattern を {{ creation_date|date:'2006-01-02_15-04-05' }} に設定し、作成日時を秒の制度まで落として、スナップショットを命名するようにします。

名前の衝突を防ぐ別の方法はパターン内に %d プレースホルダを使うことです。最初のスナップショットでは、プレースホルダは 0 に置換されます。後続のスナップショットでは、既存のスナップショットが考慮され、プレースホルダの位置の最大の数を見つけます。こ

Ceph Object - cephobject

Ceph はオープンソースのストレージプラットフォームで、データを RADOS に基づいたストレージクラスタ内に保管します。非常にスケーラブルで、単一障害点がない分散システムであり非常に信頼性が高いです。

Tip: ベーシックな Ceph クラスタを素早く構築したい場合、MicroCeph をチェックしてみてください。

Ceph はブロックストレージ用とファイルシステム用に異なるコンポーネントを提供します。

Ceph Object Gateway は `librados` 上に構築されたオブジェクトストレージインタフェースであり Ceph Storage Clusters への RESTful ゲートウェイを持つアプリケーションを提供します。Amazon S3 RESTful API の大きなサブセットと互換性を持つオブジェクトストレージの機能を提供します。

用語

Ceph は保管するデータに **オブジェクト** という用語を使用します。データを保存と管理する責任を持つデーモンは *Ceph OSD* です。Ceph のストレージは **プール** に分割されます。これはオブジェクトを保管する論理的なパーティションです。これらは **データプール**, **ストレージプール**, *OSD* **プール** とも呼ばれます。

Ceph Object Gateway は複数の OSD プールとゲートウェイの機能を提供する 1 つ以上の *Ceph Object Gateway daemon* (`radosgw`) プロセスから構成されます。

LXD の cephobject ドライバ

注釈: `cephobject` ドライバはバケットのみに使用できます。

ストレージボリュームには *Ceph* または *CephFS* ドライバを使用してください。

他のストレージドライバとは異なり、このドライバはストレージシステムをセットアップはせず、既に Ceph クラスタをインストール済みであると想定します。

事前に `radosgw` 環境をセットアップし、HTTP/HTTPS エンドポイント URL が LXD サーバーからアクセス可能なことを確認してください。Ceph クラスタをどのようにセットアップするかの情報については Manual Deployment を、そして `radosgw` 環境をどのようにセットアップするかについては Ceph Object Gateway を参照してください。

の数が 1 増加されて新しい名前に使用されます。

radosgw URL はプールの作成時に *cephobject.radosgw.endpoint* オプションを使って指定できます。また LXD はバケットの管理に radosgw-admin コマンドを使用しています。ですのでこのコマンドが LXD サーバー上で利用可能で操作可能である必要があります。

このドライバはリモートのストレージを提供するという意味でも他のドライバとは異なる振る舞いをします。結果として、内部ネットワークに依存し、ストレージへのアクセスはローカルのストレージより少し遅くなるかもしれません。一方で、リモートのストレージを使うことはクラスタ構成では大きな利点があります。これはストレージプールを同期する必要なしに、全てのクラスタメンバーが同じ内容を持つ同じストレージプールにアクセスできるからです。

LXD は OSD ストレージプールに対して完全制御できることを想定します。このため、LXD OSD ストレージプール内に LXD が所有しないファイルシステムエンティティは LXD が消してしまうかもしれないので決して置くべきではありません。

設定オプション

cephobject ドライバを使うストレージプールとこれらのプール内のストレージボリュームには以下の設定オプションが利用できます。

ストレージプール設定

キー	型	デフォルト値	説明
cephobject.bucket.name_prefix	string	-	Ceph 内のバケット名に追加する接頭辞
cephobject.cluster_name	string	ceph	使用する Ceph クラスタ
cephobject.radosgw.endpoint	string	-	radosgw ゲートウェイプロセスの URL
cephobject.radosgw.endpoint_cert_file	string	-	エンドポイント通信に使用する TLS クライアント証明書を含むファイルへのパス
cephobject.user.name	string	admin	使用する Ceph ユーザ
volatile.pool.pristine	string	true	作成時に radosgw lxd-admin ユーザが存在したかどうか

ストレージバケット設定

キー	型	デフォルト値	説明
size	string	-	ストレージバケットのクォータ

ドライバ固有の情報と設定オプションについては対応するページを参照してください。

機能比較

可能であれば、各システムの高度な機能を使って、LXD は操作を最適化しようとします。

機能	ディレクトリ	Btrfs	LVM	ZFS	Ceph RBD	CephFS	Ceph Object
最適化されたイメージストレージ	no	yes	yes	yes	yes	n/a	n/a
最適化されたインスタンスの作成	no	yes	yes	yes	yes	n/a	n/a
最適化されたスナップショットの作成	no	yes	yes	yes	yes	yes	n/a
最適化されたイメージの転送	no	yes	no	yes	yes	n/a	n/a
最適化されたボリュームの転送	no	yes	no	yes	yes	n/a	n/a
コピーオンライト	no	yes	yes	yes	yes	yes	n/a
ブロックデバイスベース	no	no	yes	no	yes	no	n/a
インスタントクローン	no	yes	yes	yes	yes	yes	n/a
コンテナ内でストレージドライバの使用	yes	yes	no	no	no	n/a	n/a
古い（最新ではない）スナップショットからのリストア	yes	yes	yes	no	yes	yes	n/a
ストレージクオータ	yes*	yes	yes	yes	yes	yes	yes
`lxd init` で利用可能	yes	yes	yes	yes	yes	no	no
オブジェクトストレージ	yes	yes	yes	yes	no	no	yes

最適化されたイメージストレージ

ディレクトリドライバを除く全てのストレージドライバはなんらかの種類の最適化されたイメージ保管フォーマットがあります。インスタンスの作成をほぼ瞬時に行うため、 LXD はインスタンスの作成時にイメージの tarball を一から解凍するのではなく事前に作成したイメージボリュームを複製します。

全く使われないかもしれないイメージのためにストレージプール上にそのようなボリュームを準備するのを避けるため、ボリュームはオンデマンドで生成されます。このため、最初のインスタンスの作成は後続のインスタンスより時間がかかります。

最適化されたボリュームの転送

Btrfs, ZFS と Ceph RBD は内部で送信/受信の機構を持ち最適化されたボリューム転送を行えます。

同じストレージドライバを使うストレージプール間でボリュームを転送する場合、ストレージドライバが最適化された転送をサポートしている場合は、LXD はこの最適化された転送を使用し、最適化された転送のほうが速いです。そうでない場合は、LXD はコンテナとファイルシステムボリュームを転送するのに `rsync` を使用するか、仮想マシンとカスタムボリュームブロックを転送するのに raw ブロック転送を使います。

最適化された転送は下層のストレージドライバのデータ転送のネイティブの機能を使い、`rsync` を使うより通常は速いです。しかし、最適化された転送のフルのポテンシャルが明らかになるのは、インスタンスや定期的なスナップショットを使用するカスタムボリュームのコピーを更新するときです。

- 初回のスナップショットを作成しコピーを更新する際、転送はほぼフルコピーと同じ時間がかかります。LXD は新しいスナップショットとスナップショットとメインボリュームの差分を転送します。
- 後続のスナップショットでは、転送は大幅に速くなります。LXD はフルの新規のスナップショットは転送せず、新規のスナップショットとターゲット上に存在する最新のスナップショットとの差分のみを転送します。
- 新規のスナップショット無しで更新する場合、LXD はメインボリュームとターゲット上に存在する最新のスナップショットとの差分のみを転送します。この転送は `rsync` を使うより通常は速いです (最新のスナップショットがあまりにも古すぎない限りは)。

一方で、スナップショットなしのインスタンスのコピーを更新する際は `rsync` を使うよりも (インスタンスがスナップショットを 1 つも持っていないか、リフレッシュが`--instance-only` フラグを使うことのために) 遅くなるでしょう。そのような場合には最適化された転送であれば (存在しない) 最新のスナップショットとメインボリュームの差分、つまりフルのボリュームを転送したでしょう。ですので、LXD はスナップショットがないリフレッシュには最適化された転送ではなく `rsync` を使用します。

おすすめのセットアップ

LXD で使う場合のベストな 2 つのオプションは ZFS と Btrfs です。このふたつは同様の機能を持ちますが、ZFS のほうがより信頼性が上です。

可能であれば、LXD のストレージプールにディスク全体かパーティションを専用で使用させるべきです。LXD で loop ベースのストレージを作れますが、プロダクション環境ではおすすめしません。詳細は **データストレージのロケーション** を参照してください。

ディレクトリーバックエンドは最後の手段の選択肢と捉えるべきです。LXD の全てのメインの機能をサポートしますが、インスタントコピーやスナップショットを実行できないため遅く非効率です。そのため、絶えずインスタンスのストレージ全体をコピーすることになります。

セキュリティの考慮

現在、 Linux Kernel はブロックベースのファイルシステム（例: `ext4`）が別のオプションでマウント済みの場合マウントオプションは黙って無視し適用しません。これは専用ディスクデバイスが異なるストレージプール間で異なるマウントオプションで共有されている時、2 つめのマウントは期待しているマウントオプションにならないかもしれないことを意味します。これは例えば 1 つめのストレージプールが `acl` サポートを提供する想定で、2 つめのストレージプールが `acl` サポートを提供しない想定であるようなときにセキュリティ上の問題になります。

この理由により、現状はストレージプールごとに専用のディスクデバイスを持つか、同じ専用ディスクを共有する全てのストレージプールで確実に同じマウントオプションを使うことを推奨します。

3.7 ネットワーク

3.7.1 ネットワークについて

あなたのインスタンスをインターネットに接続するにはいろいろな方法があります。最も簡単な方法は LXD の初期化時にネットワークブリッジを作って全てのインスタンスでこのブリッジを使うことですが、 LXD はネットワークに関するさまざまな高度な設定をサポートします。

ネットワークデバイス

インスタンスへの直接のネットワークアクセスを許可するには、 NIC とも呼ばれるネットワークデバイスを最低 1 つ割り当てる必要がありますネットワークデバイスは以下のどれかの方法で設定できます。

- LXD の初期化中にセットアップしたデフォルトのネットワークブリッジを使用する。デフォルトの設定を表示するにはデフォルトのプロファイルを確認します。

  ```
  lxc profile show default
  ```

  この方法はインスタンスのネットワークを指定しない場合に使用します。

- 既存のネットワークインタフェースをインスタンスにネットワークデバイスとして追加して使用する。このネットワークインタフェースは LXD の制御外です。そのため、ネットワークインタフェースを使用するために必要な全ての情報を LXD に指定する必要があります。

  以下のようなコマンドを使用します。

  ```
  lxc config device add <インスタンス名> <デバイス名> nic nictype=<NIC タイプ> ...
  ```

  指定可能な NIC タイプの一覧とそれらの設定プロパティについては **タイプ**: *nic* を参照してください。

  例えば、既存の Linux ブリッジ (`br0`) を追加するには以下のコマンドを使えます。

```
lxc config device add <instance_name> eth0 nic nictype=bridged parent=br0
```

- **マネージドネットワークを作成** し、それをインスタンスにネットワークデバイスとして追加する。この方法では LXD は設定されるネットワークについての全ての必要な情報を持っていますので、デバイスとしてインスタンスに直接アタッチできます。

```
lxc network attach <network_name> <instance_name> <device_name>
```

詳細は**インスタンスにネットワークをアタッチする**を参照してください。

マネージドネットワーク

LXD でマネージドネットワークは `lxc network [create|edit|set]` コマンドで作成と設定をします。

ネットワークタイプによって、 LXD はネットワークを完全に制御するか、単に外部のネットワークインタフェースを管理するかのどちらかになります。

全ての *NIC* **タイプ** がネットワークタイプとしてサポートされているわけではないことに注意してください。LXD はいくつかのタイプのみマネージドネットワークとしてセットアップできます。

完全に制御されるネットワーク

完全に制御されるネットワークではネットワークインタフェースを作成し、例えば IP を管理する機能を含むほとんどの機能を提供します。

LXD は以下のネットワークタイプをサポートします。

ブリッジネットワーク

ネットワークブリッジはインスタンス NIC が接続できる仮想的な L2 イーサネットスイッチを作成し、インスタンスが他のインスタンスやホストと通信できるようにします。LXD のブリッジは下層のネイティブな Linux のブリッジと Open vSwitch を利用できます。

LXD の文脈では、 `bridge` ネットワークタイプは、ブリッジを共用するインスタンスを同一の L2 ネットワークセグメントに接続するような L2 ブリッジを作成します。これによりインスタンス間のトラフィックを通すことができます。ブリッジはさらにローカルの DHCP と DNS を提供することもできます。

これがデフォルトのネットワークタイプです。

***OVN* ネットワーク**

OVN (Open Virtual Network) は仮想ネットワーク抽象化をサポートするソフトウェアで定義されたネットワークシステムです。あなた自身のプライベートクラウドを構築するのに使用できます。詳細は `www.ovn.org` をご参照ください。

LXD の文脈では、 `ovn` ネットワークタイプは論理ネットワークを作成します。セットアップするには OVN

ツールをインストールし設定する必要があります。さらに、OVN にネットワーク接続を提供するアップリンクのネットワークを作成する必要があります。アップリンクのネットワークとして、外部ネットワークタイプの 1 つかマネージドな LXD ブリッジを使う必要があります。

Tip: 他のネットワークタイプと違って、 OVN ネットワークは **プロジェクト** 内に作成・管理できます。これは、制限されたプロジェクトであっても、非管理者ユーザとして自身の OVN ネットワークを作成できることを意味します。

外部ネットワーク

外部ネットワークは既に存在するネットワークを使用します。そのため、 LXD がそれらを制御するには限界があるため、ネットワーク ACL、ネットワークフォワードやネットワークゾーンのような LXD の機能はサポートされません。

外部ネットワークを使用する主な目的は親インタフェースによるアップリンクのネットワークを提供することです。この外部ネットワークはインスタンスや他のネットワークを親のインタフェースに接続する際のプリセットを指定します。

LXD は以下の外部ネットワークタイプをサポートします。

macvlan ネットワーク

macvlan は仮想的な LAN (Local Area Network) で同じネットワークインタフェースに複数の IP アドレスを割り当てたい場合に使用できます。基本的にはネットワークインタフェースをそれぞれの IP アドレスを持つ複数のサブインタフェースに分割することになります。その後ランダムに生成された MAC アドレスに基づいて IP アドレスを設定できます。

LXD の文脈では、 `macvlan` ネットワークタイプは親の macvlan インタフェースへインスタンスを接続する際に使用するプリセット設定を提供します。

SR-IOV ネットワーク

SR-IOV (Single root I/O virtualization) は仮想環境内で単一のネットワークポートを複数の仮想ネットワークインタフェースのように見せるように出来るハードウェア標準です。

LXD の文脈では、 `sriov` ネットワークタイプは親の SR-IOV インタフェースへインスタンスを接続する際に使用するプリセット設定を提供します。

物理ネットワーク

物理 (`physical`) ネットワークタイプは既存のネットワークに接続します。これはネットワークインタフェースまたはブリッジになることができ、OVN のためのアップリンクネットワークとしての役目を果たします。

OVN ネットワークを親インタフェースに接続する際のプリセット設定を提供します。

お勧めの設定

一般に、マネージドネットワークは設定が容易で設定を繰り返すこと無く複数のインスタンスで同じネットワークを再利用できるので、マネージドネットワークが使用できる場合はこれを使用すべきです。

どのネットワークタイプを使用すべきかはあなたの固有の使い方によります。完全に制御されたネットワークを選ぶと、ネットワークデバイスを使用するのに比べてより多くの機能を提供します。

一般的なお勧めとしては

- LXD を単一のシステム上かパブリッククラウドで動かしている場合は、 **ブリッジネットワーク** を使用してください。場合によっては Ubuntu Fan と共に使用するのが良いかもしれません。
- あなた自身のプライベートクラウドで LXD を動かしている場合は、 *OVN* **ネットワーク** を使用してください。

注釈: OVN は適切な運用には共有された L2 のアップリンクネットワークが必要です。このため、パブリッククラウドで LXD を動かしている場合は通常 OVN は使用できません。

- インスタンス NIC をマネージドネットワークに接続するためには、可能であれば `parent` プロパティより `network` プロパティを使用してください。こうすることで、 NIC はネットワークの設定を引き継ぎ、`nictype` を指定する必要がなくなります。

3.7.2 ネットワークを作成し設定するには

マネージドネットワークを作成し設定するには、 `lxc network` コマンドとそのサブコマンドを使用します。どのコマンドでも `--help` を追加すると使用方法と利用可能なフラグについてより詳細な情報を表示できます。

ネットワークタイプ

以下のネットワークタイプが利用できます。

ネットワークタイプ	ドキュメント	設定オプション
bridge	ブリッジネットワーク	設定オプション
ovn	*OVN* ネットワーク	設定オプション
macvlan	*macvlan* ネットワーク	設定オプション
sriov	*SR-IOV* ネットワーク	設定オプション
physical	物理ネットワーク	設定オプション

ネットワークを作成する

ネットワークを作成するには以下のコマンドを実行します。

```
lxc network create <name> --type=<network_type> [configuration_options...]
```

利用可能なネットワークタイプ一覧と設定オプションへのリンクは **ネットワークタイプ** を参照してください。

--type 引数を指定しない場合、デフォルトのタイプ bridge が使用されます。

クラスタ内にネットワークを作成する

LXD クラスタを実行していてネットワークを作成したい場合、各クラスタメンバーに別々にネットワークを作成する必要があります。この理由はネットワーク設定は、例えば親ネットワークインタフェースの名前のように、クラスタメンバー間で異なるかもしれないからです。

このため、まず --target=<cluster_member> フラグとメンバー用の適切な設定を指定して保留中のネットワークを作成する必要があります。全てのメンバーで同じネットワーク名を使うようにしてください。次に実際にセットアップするために --target フラグなしでネットワークを作成してください。

例えば、以下の一連のコマンドで 3 つのクラスタメンバー上に UPLINK という名前の物理ネットワークをセットアップします。

user@host:~$ lxc network create UPLINK --type=physical parent=br0 --target=vm01 Network UPLINK pending on member vm01 user@host:~$ lxc network create UPLINK --type=physical parent=br0 --target=vm02 Network UPLINK pending on member vm02 user@host:~$ lxc network create UPLINK --type=physical parent=br0 --target=vm03 Network UPLINK pending on member vm03 user@host:~$ lxc network create UPLINK --type=physical Network UPLINK created **クラスタのネットワークを設定するには**も参照してください。

インスタンスにネットワークをアタッチする

マネージドネットワークを作成後、それをインスタンスに *NIC* **デバイス**としてアタッチできます。

そのためには、以下のコマンドを使います。

```
lxc network attach <network_name> <instance_name> [<device_name>] [<interface_name>]
```

デバイス名とインタフェース名は省略可能ですが、少なくともデバイス名は指定することをお勧めします。指定しない場合、LXD はネットワーク名をデバイス名として使用しますが、紛らわしく問題を起こすかもしれません。例えば、LXD イメージは eth0 インタフェースに IP 自動設定を行いますが、インタフェースの名前が違うと機能しません。

例えば、my-network というネットワークを my-instance というインタンスに eth0 デバイスとしてアタッチするには、以下のコマンドを入力します。

```
lxc network attach my-network my-instance eth0
```

NIC デバイスを追加する

`lxc network attach` コマンドはインスタンスに NIC デバイスを追加するショートカットです。別の方法として、通常通りネットワーク設定で NIC デバイスを追加できます。

```
lxc config device add <instance_name> <device_name> nic network=<network_name>
```

この方法を使う場合、必要に応じてネットワークのデフォルト設定をオーバーライドするように追加の設定をコマンドに追加できます。全ての利用可能なデバイスオプションについては *NIC* **デバイス**を参照してください。

ネットワークを設定する

既存のネットワークを設定するには、 `lxc network set` と `lxc network unset` コマンド (単一の設定項目を設定する場合) または `lxc network edit` コマンド (設定全体を編集する場合) のどちらかを使います。特定のクラスタメンバーの設定を変更するには、 `--target` フラグを追加してください。

例えば、以下のコマンドは物理ネットワークの DNS サーバーを設定します。

```
lxc network set UPLINK dns.nameservers=8.8.8.8
```

利用可能な設定オプションはネットワークタイプによって異なります。各ネットワークタイプの設定オプションへのリンクは **ネットワークタイプ** を参照してください。

高度なネットワーク機能を設定するためには別のコマンドがあります。以下のドキュメントを参照してください。

- **ネットワーク** *ACL* **を設定するには**
- **ネットワークフォワードを設定するには**
- **ネットワークロードバランサーを設定するには**
- **ネットワークゾーンを設定するには**
- **ピアルーティング関係を作成するには** (OVN のみ)

3.7.3 ネットワーク ACL を設定するには

注釈: ネットワーク ACL は *OVN NIC* **タイプ**、*OVN* **ネットワーク** と **ブリッジネットワーク** (いくつか制限あり、**ブリッジの制限** 参照) で利用できます。

ネットワーク ACL (Access Control Lists; アクセス制御リスト) は同じネットワークに接続された異なるインスタンス間のネットワークアクセスや、他のネットワークとのアクセスを制御するトラフィクルールを定義します。

ネットワーク ACL は インスタンスの NIC やネットワークに直接適用できます。ネットワークに適用するときは、ネットワークに接続された全ての NIC に ACL が適用されます。

特定の ACL を (明示的にあるいはネットワークから暗黙的に) 適用したインスタンス NIC は論理的なグループを形成し、他のルールから送信元あるいは送信先として参照できます。より詳細な情報は *ACL* **グループ** を参照してください。

ACL を作成する

ACL を作成するには以下のコマンドを使用します。

```
lxc network acl create <ACL_name> [configuration_options...]
```

このコマンドはルール無しの ACL を作成します。次のステップとして ACL に **ルールを追加します**。

有効なネットワーク ACL の名前は以下のルールに従う必要があります。

- 名前は 1 文字から 63 文字の間である
- 名前は ASCII の文字、数字、ハイフンからのみなる
- 名前は数字やハイフンから始まらない
- 名前はハイフンで終わらない

ACL のプロパティ

ACL のプロパティには次のものがあります。

プロパティ	型	必須	説明
name	string	yes	プロジェクト内でユニークなネットワーク ACL の名前
description	string	no	ネットワーク ACL の説明
ingress	rule list	no	内向きのトラフィックルールのリスト
egress	rule list	no	外向きのトラフィックルールのリスト
config	string set	no	キー・バリューペア形式での設定オプション (user.* カスタムキーのみサポート)

ルールの追加と削除

それぞれの ACL はルールの 2 つのリストを含みます。

- **イングレス** *(ingress)* ルールは NIC に向かう内向きのトラフィックに適用されます。
- **イーグレス** *(egress)* ルールは NIC から出ていく外向きのトラフィックに適用されます。

ACL にルールを追加するには、以下のコマンドを使います。 <direction> には ingress か egress のどちらかを指定します。

```
lxc network acl rule add <ACL_name> <direction> [properties...]
```

このコマンドは指定した方向 (direction) に対応するリストにルールを追加します。

(*ACL* **全体を編集する** 場合を除き) ルールを編集することはできませんが、以下のコマンドでルールを削除はできます。

```
lxc network acl rule remove <ACL_name> <direction> [properties...]
```

ユニークにルールを特定するのに必要な全てのプロパティを指定するか、またはマッチした全てのルールを削除するためコマンドに --force を追加する必要があります。

ルールの順番と優先度

ルールはリストとして提供されます。しかしリスト内のルールの順番は重要ではなくフィルタリングには影響しません。

LXD は以下のように action プロパティに基づいてルールの順番を自動的に決めます。

- drop
- reject

- allow
- 上記の全てにマッチしなかったトラフィックに対する自動のデフォルトアクション (デフォルトでは reject、**デフォルトアクションの設定** 参照)。

これは NIC に複数の ACL を適用する際、組み合わせたルールの順番を指定する必要がないことを意味します。ACL 内のあるルールがマッチすれば、そのルールが採用され、他のルールは考慮されません。

ルールのプロパティ

ACL ルールには次のプロパティがあります。

プロパティ	型	必須	説明
action	string	yes	マッチしたトラフィックに適用するアクション (allow, reject または drop)
state	string	yes	ルールの状態 (enabled, disabled または logged)、未設定の場合のデフォルト値は enabled
descripti	string	no	ルールの説明
source	string	no	CIDR か IP の範囲、送信元の ACL の名前、あるいは (ingress ルールに対しての) ソースサブジェクト名セレクターのカンマ区切りリスト、または any の場合は空を指定
destinati	string	no	CIDR か IP の範囲、送信先の ACL の名前、あるいは (egress ルールに対しての) デスティネーションサブジェクト名セレクターのカンマ区切りリスト、または any の場合は空を指定
protocol	string	no	マッチ対象のプロトコル (icmp4, icmp6, tcp, udp)、または any の場合は空を指定
source_pc	string	no	protocol が udp か tcp の場合はポートかポートの範囲 (開始-終了で両端含む) のカンマ区切りリスト、または any の場合は空を指定
destinati	string	no	protocol が udp か tcp の場合はポートかポートの範囲 (開始-終了で両端含む) のカンマ区切りリスト、または any の場合は空を指定
icmp_type	string	no	protocol が icmp4 か icmp6 の場合は ICMP の Type 番号、または any の場合は空を指定
icmp_code	string	no	protocol が icmp4 か icmp6 の場合は ICMP の Code 番号、または any の場合は空を指定

ルール内でセレクタを使う

注釈: この機能は *OVN NIC* **タイプ** と *OVN* **ネットワーク** でのみサポートされます。

(ingress ルールの) source フィールドと (egress ルールの) destination フィールドは CIDR や IP の範囲の代わりにセレクタの使用をサポートします。

この機能を使えば、 IP のリストを管理したり追加のサブネットを作ることなしに、 ACL グループかネットワークセレクタを使ってインスタンスのグループに対するルールを定義できます。

ACL グループ

(明示的にあるいはネットワーク経由で暗黙的に) 特定の ACL を適用されたインスタンス NIC は論理的なポートグループを形成します。

そのような ACL グループは **サブジェクト名セレクタ** と呼ばれ、他の ACL グループ内で ACL 名を用いて参照できます。

例えば `foo` という名前の ACL がある場合、この ACL が適用されたインスタンス NIC のグループを `source=foo` で参照できます。

ネットワークセレクタ

ネットワークサブジェクトセレクタ を用いて、ネットワーク上の外向きと内向きのトラフィックにルールを定義できます。

`@internal` と `@external` という 2 つの特別なネットワークサブジェクトセレクタがあります。これらはそれぞれネットワークのローカルと外向きのトラフィックを示します。例:

```
source=@internal
```

ネットワークが **ネットワークピア** をサポートする場合、ピア接続間のトラフィックを@<network_name>/<peer_name> という形式のネットワークサブジェクトセレクタで参照できます。例:

```
source=@ovn1/mypeer
```

ネットワークサブジェクトセレクターを使用する際は、 ACL 適用先のネットワークは指定されたピア接続を持っていなければなりません。持っていない場合 ACL は適用できません。

トラフィックのログ

一般的には ACL はインスタンスとネットワーク間のネットワークトラフィックを制御するためのものです。しかし、特定のネットワークトラフィックをログ出力するためにルールを使うこともできます。これはモニタリングや、ルールを実際に有効にする前にテストするのに役立ちます。

ログのためにルールを追加するには `state=logged` プロパティ付きでルールを作成してください。ACL 内の全てのログのルールに対するログ出力は以下のコマンドで表示できます。

```
lxc network acl show-log <ACL_name>
```

ACL を編集する

ACL を編集するには以下のコマンドを使用します。

```
lxc network acl edit <ACL_name>
```

このコマンドは ACL を編集用に YAML 形式でオープンします。ACL 設定とルールの両方を編集できます。

ACL の適用

ACL の設定が終わったらネットワークかインスタンス NIC に適用する必要があります。

そのためにはネットワークか NIC の設定の `security.acls` リストに ACL を追加してください。ネットワークの場合は、以下のコマンドを使います。

```
lxc network set <network_name> security.acls="<ACL_name>"
```

インスタンス NIC の場合は、以下のコマンドを使います。

```
lxc config device set <instance_name> <device_name> security.acls="<ACL_name>"
```

デフォルトアクションの設定

1 つ以上の ACL が NIC に (明示的にあるいはネットワーク経由で暗黙的に) 適用されると、 NIC にデフォルトの reject ルールが追加されます。このルールは適用された ACL 内のどのルールにもマッチしない全てのトラフィックを拒否 (reject) します。

この挙動はネットワークと NIC レベルの `security.acls.default.ingress.action` と `security.acls.default.egress.action` 設定で変更できます。NIC レベルの設定はネットワークレベルの設定を上書きします。

例えば、ネットワークに接続された全てのインスタンスの内向きのトラフィックを許可 (`allow`) するには以下のコマンドを使用します。

```
lxc config device set <instance_name> <device_name> security.acls.default.ingress.
↪action=allow
```

インスタンス NIC に同じデフォルトアクションを設定するには以下のコマンドを使用します。

```
lxc config device set <instance_name> <device_name> security.acls.default.ingress.
↪action=allow
```

ブリッジの制限

ブリッジネットワークにネットワーク ACL を使用する場合は以下の制限に気を付けてください。

- OVN ACL とは違い、ブリッジ ACL はブリッジと LXD ホストの間の境界のみに適用されます。これは外部へと外部からのトラフィックにネットワークポリシーを適用するために使うことしかできないことを意味します。ブリッジ間のファイアウォール、つまり同じブリッジに接続されたインスタンス間のトラフィックを制御するファイアウォールには使えません。
- *ACL* **グループとネットワークセレクタ** はサポートされません。
- `iptables` ファイアウォールドライバを使う際は、 IP レンジサブジェクト（例：`192.0.2.1-192.0.2.10`）は使用できません。
- ベースラインのネットワークサービスルールが（対応する INPUT/OUTPUT チェイン内の） ACL ルールの前に適用されます。これは一旦 ACL チェインに入ってしまうと INPUT/OUTPUT と FORWARD トラフィックを区別できないからです。このため ACL ルールはベースラインのサービスルールをブロックするのには使えません。

3.7.4 ネットワークフォワードを設定するには

注釈: ネットワークフォワードは *OVN* **ネットワーク** と **ブリッジネットワーク** で利用できます。

ネットワークフォワードは外部 IP アドレス（あるいは外部 IP アドレスの特定のポート）をフォワード設定が属するネットワーク内の内部 IP アドレス（あるいは内部 IP アドレスの特定のポート）にフォワードする機能です。

この機能は外部 IP アドレスが限定されていて 1 つの外部アドレスを複数のインスタンスで共有したい場合に有用です。この場合にネットワークフォワードを 2 つの異なる方法で利用できます。

- 外部アドレスからの全てのトラフィックを 1 つのインスタンスの内部アドレスにフォワードします。この方法はネットワークフォワードを単に再設定することで外部アドレスに向けられたトラフィックを別のインスタンスに簡単に移動できます。
- 外部アドレスの異なるポートからのトラフィックを異なるインスタンスにフォワードします (さらにこれらのインスタンスの異なるポートにフォワードもできます)。この方法は外部 IP アドレスを「共有」し、複数のインスタンスを一度に公開できます。

ネットワークフォワードを作成する

ネットワークフォワードを作成するには以下のコマンドを使用します。

```
lxc network forward create <network_name> <listen_address> [configuration_options...]
```

それぞれのフォワードはネットワークに割り当てられます。フォワードには単一の外部リッスンアドレスが必要です (使用しているネットワークに応じてどのアドレスがフォワードできるかについて詳細は **リッスンアドレスの要件** を参照してください)。

さらに target_address=<IP_address> 設定オプションを追加することでデフォルトのターゲットアドレスを追加することもできます。こうするとポート指定にマッチしないトラフィックは全てこのアドレスにフォワードします。このターゲットアドレスはフォワードが関連付けられるネットワークと同じサブネット内でなければならないことに注意してください。

フォワードのプロパティ

ネットワークフォワードのプロパティには以下のものがあります。

プロパティ	型	必須	説明
listen_addre	string	yes	リッスンする IP アドレス
description	string	no	ネットワークフォワードの説明
config	string set	no	キー/バリューペア形式の設定オプション （target_address と user.* カスタムキーのみサポート）
ports	port list	no	**ポート指定** のリスト

リッスンアドレスの要件

有効なリッスンアドレスの要件はフォワードがどのネットワークタイプに割り当てられるかに応じて異なります。

ブリッジネットワーク

- 任意の衝突しないリッスンアドレスが使用できます。
- リッスンアドレスは他のネットワークで使用中のサブネットとオーバーラップはできません。

OVN ネットワーク

- 利用可能なリッスンアドレスはアップリンクネットワークの ipv{n}.routes 設定か (設定されている場合は) プロジェクトの restricted.networks.subnets 設定で定義されていなければなりません。
- リッスンアドレスは他のネットワークで使用中のサブネットとオーバーラップはできません。

ポートを設定する

リッスンアドレスの特定のポートからターゲットアドレスの特定のポートにトラフィックをフォワードするためにネットワークフォワードにポート指定を追加できます。このターゲットアドレスはデフォルトターゲットアドレスとは異なるものである必要があります。またフォワードを割り当てるネットワークと同じサブネットである必要があります。

ポート指定を追加するには以下のコマンドを使用します。

```
lxc network forward port add <network_name> <listen_address> <protocol> <listen_ports>
↪<target_address> [<target_ports>]
```

単一のポートか一組のポートを指定できます。異なるポートにトラフィックをフォワードしたい場合、 2 つの選択肢があります。

- 単一のターゲットポートを指定し、全てのリッスンポートからのトラフィックをこのターゲットポートにフォワードします。
- リッスンポートと同じ数の一組のターゲットポートを指定し、最初のリッスンポートを最初のターゲットポートに、 2 番目のリッスンポートを 2 番目のターゲットポートに、以下同様というようにフォワードします。

ポートのプロパティ

ネットワークフォワードポートのプロパティには以下のものがあります。

プロパティ	型	必須	説明
protocol	string	yes	ポートのプロトコル (tcp or udp)
listen_port	string	yes	リッスンするポート (例 80,90-100)
target_address	string	yes	フォワード先の IP アドレス
target_port	string	no	ターゲットのポート (例 70,80-90 or 90)、 空の場合は listen_port と同じ
description	string	no	ポートの説明

ネットワークフォワードを編集する

ネットワークフォワードを編集するには以下のコマンドを使用します。

```
lxc network forward edit <network_name> <listen_address>
```

このコマンドはネットワークフォワードを編集用に YAML 形式でオープンします。全般の設定とポート指定の両方を編集できます。

ネットワークフォワードを削除する

ネットワークフォワードを削除するには以下のコマンドを使用します。

```
lxc network forward delete <network_name> <listen_address>
```

3.7.5 ネットワークゾーンを設定するには

注釈: ネットワークゾーンは *OVN* **ネットワーク** と **ブリッジネットワーク** で利用できます。

ネットワークゾーンは LXD のネットワークの DNS レコードを保持するのに使用します。

ネットワークゾーンを使うと全てのインスタンスの有効な正引きと逆引きのレコードを自動的に維持できます。多くのネットワークにまたがる複数のインスタンスからなる LXD クラスタを運用する際に有用です。

各インスタンスに DNS レコードを持つとインスタンス上のネットワークサービスにアクセスするのがより簡単になります。また例えば外部への SMTP サービスをホストする際にも重要です。インスタンスに正しい正引きと逆引きの DNS エントリがないと、送信されたメールが潜在的なスパムと判定されてしまうかもしれません。

各ネットワークは異なるゾーンに関連します。

- 正引き DNS レコード - カンマ区切りの複数のゾーン (プロジェクトごとに最大 1 つ)
- IPv4 逆引き DNS レコード - 単一のゾーン
- IPv6 逆引き DNS レコード - 単一のゾーン

LXD は全てのインスタンス、ネットワークゲートウェイ、ダウンストリーム (下流) のネットワークポートの全てに対して正引きと逆引きのレコードを自動で管理し、オペレータのプロダクションの DNS サーバーへのゾーン転送のためのこれらのゾーンを提供します。

プロジェクトビュー

プロジェクトには features.networks.zones 機能があります。デフォルトでは無効です。これは新しいネットワークゾーンがどのプロジェクト内に作成されるかを制御します。この機能を有効にすると新しいゾーンはプロジェクト内に作成されますが、無効の場合はデフォルトプロジェクト内に作成されます。

これにより、複数のプロジェクトがデフォルトプロジェクト (すなわち features.networks=false と設定されたプロジェクト) 内のネットワークを共有できるようになり、共有されたネットワークに対してプロジェクト指向の (プロジェクト内のインスタンスのアドレスのみを含むような)「ビュー」を提供するプロジェクト固有の DNS ゾーンを持てるようになります。

生成されるレコード

正引きレコード

例えば、あなたのネットワークで lxd.example.net の正引き DNS レコードのゾーンを設定した場合、以下の DNS 名を解決するレコードを生成します。

- ネットワーク内の全てのインスタンスに対して: <instance_name>.lxd.example.net
- ネットワークゲートウェイに対して: <network_name>.gw.lxd.example.net
- ダウンストリームネットワークポートに対して (ダウンストリーム OVN ネットワークを持つアップリンクのネットワーク上に設定されれうネットワークゾーンに対して): <project_name>-<downstream_network_name>.uplink.lxd.example.net
- ゾーンに手動で追加されたレコード

ゾーン設定に対して生成されたレコードは dig コマンドで確認できます。これは core.dns_address が<DNS_server_IP>:<DNS_server_PORT>に設定されていることを前提としています。(その設定オプションを設定すると、バックエンドはすぐにそのアドレスでサービスを開始します。)

特定のゾーンに対して dig リクエストが許可されるようにするためには、そのゾーンの peers.NAME.address 設定オプションを設定する必要があります。NAME はランダムなもので構いません。値は、dig が呼び出される IP アドレスと一致しなければなりません。同じランダムな NAME の peers.NAME.key は未設定のままにしておく必要があります。

例: lxc network zone set lxd.example.net peers.whatever.address=192.0.2.1

注釈: dig が呼び出し元の同じマシンのアドレスであるだけでは十分ではありません。それは、lxd 内の DNS サーバーが正確なリモートアドレスと考えるものと文字列で一致する必要があります。dig は 0.0.0.0 にバインドするため、必要なアドレスはおそらく、あなたが core.dns_address に提供したものと同じです。

例えば、dig @<DNS_server_IP> -p <DNS_server_PORT> axfr lxd.example.net と実行すると以下のような出力がでるかもしれません。

```
user@host:~$ dig @192.0.2.200 -p 1053 axfr lxd.example.net        lxd.example.net. 3600 IN
SOA lxd.example.net. ns1.lxd.example.net. 1669736788 120 60 86400 30lxd.example.
net. 300 IN NS ns1.lxd.example.net.lxdtest.gw.lxd.example.net. 300 IN A 192.0.2.
1lxdtest.gw.lxd.example.net. 300 IN AAAA fd42:4131:a53c:7211::1default-ovntest.
uplink.lxd.example.net. 300 IN A 192.0.2.20default-ovntest.uplink.lxd.example.net.
300 IN AAAA fd42:4131:a53c:7211:216:3eff:fe4e:b794c1.lxd.example.net. 300 IN AAAA
fd42:4131:a53c:7211:216:3eff:fe19:6edec1.lxd.example.net. 300 IN A 192.0.2.125manualtest.
lxd.example.net. 300 IN A 8.8.8.8lxd.example.net. 3600 IN SOA lxd.example.net. ns1.lxd.
example.net. 1669736788 120 60 86400 30
```

逆引きレコード

192.0.2.0/24 を使用するネットワークに 2.0.192.in-addr.arpa の IPv4 逆引き DNS レコードのゾーンを設定すると、正引きゾーンの 1 つを経由してネットワークを参照する全てのプロジェクトからのアドレスの逆引き PTR DNS レコードを生成します。

例えば dig @<DNS_server_IP> -p <DNS_server_PORT> axfr 2.0.192.in-addr.arpa を実行すると以下のような出力が得られるかもしれません。

```
user@host:~$ dig @192.0.2.200 -p 1053 axfr 2.0.192.in-addr.arpa 2.0.192.in-addr.arpa. 3600
IN SOA 2.0.192.in-addr.arpa. ns1.2.0.192.in-addr.arpa. 1669736828 120 60 86400 302.0.
192.in-addr.arpa. 300 IN NS ns1.2.0.192.in-addr.arpa.1.2.0.192.in-addr.arpa. 300 IN PTR
lxdtest.gw.lxd.example.net.20.2.0.192.in-addr.arpa. 300 IN PTR default-ovntest.uplink.
lxd.example.net.125.2.0.192.in-addr.arpa. 300 IN PTR c1.lxd.example.net.2.0.192.in-addr.
arpa. 3600 IN SOA 2.0.192.in-addr.arpa. ns1.2.0.192.in-addr.arpa. 1669736828 120 60 86400
30
```

組み込みの DNS サーバーを有効にする

ネットワークゾーンを使用するには、組み込みの DNS サーバーを有効にする必要があります。

そのためには、 LXD サーバーのローカルアドレスに core.dns_address 設定オプション (**コア設定**参照) を設定してください。既存の DNS との衝突を避けるためポート 53 を使用しないことをお勧めします。これは DNS サーバーがリッスンするアドレスです。LXD クラスタの場合、アドレスは各クラスタメンバーによって異なるかもしれないことに注意してください。

注釈: 組み込みの DNS サーバーは AXFR 経由でのゾーン転送のみをサポートしており、DNS レコードへの直接の問い合わせはできません。つまりこの機能は外部の DNS サーバー (bind9, nsd, ...) の使用を前提としています。外部の DNS サーバーが LXD からの全体のゾーンを転送し、有効期限を過ぎたら更新し、DNS 問い合わせに対す

る管理権限を持つ応答 (authoritative answers) を提供します。

ゾーン転送の認証はゾーン毎に設定され、各ゾーンでピアごとに IP アドレスと TSIG キーを設定して、TSIG キーベースの認証を行います。

ネットワークゾーンの作成と設定

ネットワークゾーンの作成には以下のコマンドを使用します。

```
lxc network zone create <network_zone> [configuration_options...]
```

以下の例は正引き DNS レコードのゾーン、IPv4 逆引き DNS レコードのゾーン、IPv6 逆引き DNS レコードのゾーンを作成する方法を示しています。

```
lxc network zone create lxd.example.net
lxc network zone create 2.0.192.in-addr.arpa
lxc network zone create 1.0.0.0.1.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa
```

注釈: ゾーン名は複数のプロジェクトをまたいでグローバルにユニークでなければなりません。そのため、別のプロジェクト内の既存のゾーンのせいでゾーンの作成がエラーになることがありえます。

ネットワークを作成するときに設定オプションを指定できますし、後から以下のコマンドで設定もできます。

```
lxc network zone set <network_zone> <key>=<value>
```

YAML 形式でネットワークゾーンを編集するには以下のコマンドを使用します。

```
lxc network zone edit <network_zone>
```

設定オプション

ネットワークゾーンで利用可能な設定オプションは下記のとおりです。

キー	型	必須	デフォルト値	説明
peers.NAME.address	string	no	-	DNS サーバーの IP アドレス
peers.NAME.key	string	no	-	サーバーの TSIG キー
dns.nameservers	string set	no	-	(NS レコード用の) DNS サーバーの FQDN のカンマ区切りリスト
network.nat	bool	no	true	NAT されたサブネットのレコードを生成するかどうか
user.*	*	no	-	ユーザー提供の自由形式のキー・バリューペア

注釈: tsig-keygen を使用して TSIG キーを生成するとき、キー名は<zone_name>_<peer_name>. というフォーマットに従わなければなりません。たとえば、ゾーン名が lxd.example.net でピア名が bind9 の場合、キー名は lxd.example.net_bind9. でなければなりません。この形式に従わない場合、ゾーン転送が失敗する可能性があります。

ネットワークにネットワークゾーンを追加する

ネットワークにゾーンを追加するにはネットワーク設定内に対応する設定オプションを設定します。

- 正引き DNS レコードには: dns.zone.forward
- IPv4 逆引き DNS レコードには: dns.zone.reverse.ipv4
- IPv6 逆引き DNS レコードには: dns.zone.reverse.ipv6

例えば

```
lxc network set <network_name> dns.zone.forward="lxd.example.net"
```

ゾーンはプロジェクトに属し、プロジェクトの networks 機能に紐づきます。プロジェクトの restricted.networks.zones 設定キーを使ってプロジェクトを指定のドメインとサブドメインに制限できます。

カスタムレコードを追加する

ネットワークゾーンは、全てのインスタンス、ネットワークゲートウェイ、ダウンストリームネットワークポートに対して正引きと逆引きレコードを自動的に生成します。

そのためには `lxc network zone record` コマンドを使用します。

レコードを作成する

レコードを作成するには以下のコマンドを使用します。

```
lxc network zone record create <network_zone> <record_name>
```

このコマンドはエントリ無しの空のレコードを作成しネットワークゾーンに追加します。

レコードのプロパティ

レコードは以下のプロパティを持ちます。

プロパティ	型	必須	説明
`name`	string	yes	レコードのユニークな名前
`description`	string	no	レコードの説明
`entries`	entry list	no	DNS エントリのリスト
`config`	string set	no	キー／バリュー形式の設定オプション (`user.*` カスタムキーのみサポート)

エントリを追加または削除する

レコードにエントリを追加するには以下のコマンドを使います。

```
lxc network zone record entry add <network_zone> <record_name> <type> <value> [--ttl
↪<TTL>]
```

このコマンドはレコードに指定した型と値を持つ DNS エントリを追加します。

例えば、デュアルスタックのウェブサーバーを作成するには以下のような 2 つのエントリを持つレコードを追加します。

```
lxc network zone record entry add <network_zone> <record_name> A 1.2.3.4
lxc network zone record entry add <network_zone> <record_name> AAAA 1234::1234
```

エントリにカスタムの time-to-live (秒で指定) を設定するには `--ttl` フラグが使えます。指定しない場合、デフォルトの 300 秒になります。

(lxc network zone record edit でレコード全体を編集するのを除いて) エントリを編集は出来ませんが、以下のコマンドでエントリを削除できます。

```
lxc network zone record entry remove <network_zone> <record_name> <type> <value>
```

3.7.6 LXD を BGP サーバーとして設定するには

注釈: BGP サーバー機能は **ブリッジネットワーク** と **物理ネットワーク** で利用できます。

BGP (Border Gateway Protocol) は自律システム間でルーティング情報を交換できるプロトコルです。

外部アドレスを特定の LXD サーバーやインスタンスに直接ルーティングしたい場合は、 LXD を BGP サーバーとして設定できます。すると LXD は BGP ピアとして振る舞い、関連するルートとネクストホップを外部のルータ、例えばあなたのネットワークルータに広告します。アップストリームの BGP ルータとセッションを自動的に確立し、使用中のアドレスとサブネットを広告します。

BGP サーバー機能は LXD サーバーやクラスタが正しいホストへルーティングされる特定のサブネットやアドレスを取得することで内部／外部アドレス空間を直接使用できるようにします。こうすることで、トラフィックを対象のインスタンスにフォワードできます。

ブリッジネットワークについては、以下のアドレスとネットワークが広告されます。

- ipv4.address または ipv6.address サブネットのネットワーク (対応する nat プロパティが true に設定されていない場合)
- ipv4.nat.address または ipv6.nat.address サブネットのネットワーク (対応する nat プロパティが true に設定されていない場合)
- ネットワークフォワードアドレス
- ブリッジネットワークに接続されているインスタンス NIC 上の ipv4.routes.external または ipv6.routes.external で指定されているアドレスまたはサブネット

サブネットを対応する設定オプションに確実に追加してください。さもなければ、広告されません。

物理ネットワークについては、物理ネットワークのレベルに直接広告されるアドレスはありません。代わりに、全てのダウンストリームネットワーク (network オプションで物理ネットワークをアップリンクネットワークとして指定するネットワーク) のネットワーク、フォワードとルートがブリッジネットワークに対するのと同じように広告されます。

注釈: 現時点では、特定のピアに一部の特定のルート／アドレスのみを広告することはできません。これが必要な

場合はアップストリームルータでプリフィクスをフィルタしてください。

BGP サーバーを設定する

LXD を BGP サーバーとして設定するには、以下のサーバー設定オプションを全てのクラスタメンバーで設定してください (**コア設定**参照)。

- `core.bgp_address` - BGP サーバーの IP アドレス
- `core.bgp_asn` - ローカルサーバーの ASN (Autonomous System Number)
- `core.bgp_routerid` - BGP サーバーのユニークな識別子

例えば、以下のような値を設定します。

```
lxc config set core.bgp_address=192.0.2.50:179
lxc config set core.bgp_asn=65536
lxc config set core.bgp_routerid=192.0.2.50
```

これらの設定オプションが一旦設定されると、 LXD は BGP セッションのリッスンを始めます。

ネクストホップを設定する (bridge のみ)

ブリッジネットワークについては、ネクストホップ設定をオーバーライドできます。デフォルトでは、ネクストホップは BGP セッションに使用されるアドレスに設定されます。

別のアドレスに設定するには、 `bgp.ipv4.nexthop` または `bgp.ipv6.nexthop` を設定してください。

OVN ネットワークに BGP ピアを設定する

OVN ネットワークをアップリンクネットワーク (`physical` または `bridge`) と使用する場合、アップリンクネットワークは許可されるサブネット一覧と BGP 設定を持つネットワークです。このため、 BGP サーバーに接続するのに必要な情報を含むアップリンクネットワーク上に BGP ピアを設定する必要があります。

アップリンクネットワークに対して以下の設定オプションを設定してください。

- `bgp.peers.<name>.address` - ダウンストリームネットワークで使用されるピアアドレス
- `bgp.peers.<name>.asn` - ローカルサーバーの ASN
- `bgp.peers.<name>.password` - ピアセッションに対するオプショナルなパスワード
- `bgp.peers.<name>.holdtime` - ピアセッションに対する省略可能なホールドタイム (秒で指定)

アップリンクネットワークが一旦設定されると、ダウンストリームの OVN ネットワークは BGP で広告される外部のサブネットとアドレスを取得します。ネクストホップはアップリンクネットワークの OVN ルータのアドレスに設定されます。

3.7.7 ブリッジネットワーク

LXD でのネットワークの設定タイプの 1 つとして、LXD はネットワークブリッジの作成と管理をサポートしています。

ネットワークブリッジはインスタンス NIC が接続できる仮想的な L2 イーサネットスイッチを作成し、インスタンスが他のインスタンスやホストと通信できるようにします。LXD のブリッジは下層のネイティブな Linux のブリッジと Open vSwitch を利用できます。

`bridge` ネットワークはそれを利用する複数のインスタンスを接続する L2 ブリッジを作成しそれらのインスタンスを単一の L2 ネットワークセグメントにします。LXD で作成されたブリッジは"managed"です。つまり、ブリッジインタフェース自体を作成するのに加えて、LXD さらに DHCP、IPv6 ルート広告と DNS サービスを提供するローカルの `dnsmasq` プロセスをセットアップします。デフォルトではブリッジに対して NAT も行います。

LXD ブリッジネットワークでファイアウォールを設定するための手順については**ファイアウォールを設定するには**を参照してください。

注釈: 静的な DHCP 割当は MAC アドレスを DHCP 識別子として使用するクライアントに依存します。この方法はインスタンスをコピーする際に衝突するリースを回避し、静的に割り当てられたリースが正しく動くようにします。

IPv6 プリフィクスサイズ

ブリッジネットワークで IPv6 を使用する場合、64 のプリフィクスサイズを使用するべきです。

より大きなサブネット (つまり 64 より小さいプリフィクスを使用する) も正常に動くはずですが、通常それらは SLAAC (Stateless Address Auto-configuration) には役立ちません。

より小さなサブネットも (IPv6 の割当にはステートフル DHCPv6 を使用する場合) 理論上は可能ですが、`dnsmasq` に適切にサポートされていないので問題が起きるかもしれません。より小さなサブネットを作らなければならない場合は、静的割当を使うか別のルータ広告デーモンを使用してください。

設定オプション

bridge ネットワークタイプでは現在以下の設定キーネームスペースがサポートされています。

- bgp (BGP ピア設定)
- bridge (L2 インタフェースの設定)
- dns (DNS サーバーと名前解決の設定)
- fan (Ubuntu FAN overlay に特有な設定)
- ipv4 (L3 IPv4 設定)
- ipv6 (L3 IPv6 設定)
- maas (MAAS ネットワーク識別)
- security (ネットワーク ACL 設定)
- raw (raw の設定のファイルの内容)
- tunnel (ホスト間のトンネリングの設定)
- user (key/value の自由形式のユーザメタデータ)

注釈: ネットワークのサブネット情報を指定する箇所では LXD は CIDR 表記 (例えば 192.0.2.0/24 や 2001:db8::/32) を使用します。これは単一のアドレスが必要なケース (例えば、トンネルのローカル／リモートアドレス、インスタンスに適用する NAT アドレスや特定のアドレス) では適用されません。

---

bridge ネットワークタイプには以下の設定オプションがあります。

キー	型	条件	デフォルト
bgp.peers.NAME.address	string	BGP サーバー	-
bgp.peers.NAME.asn	integer	BGP サーバー	-
bgp.peers.NAME.password	string	BGP サーバー	- (パスワード無し)
bgp.peers.NAME.holdtime	integer	BGP サーバー	180
bgp.ipv4.nexthop	string	BGP サーバー	ローカルアドレス
bgp.ipv6.nexthop	string	BGP サーバー	ローカルアドレス
bridge.driver	string	-	native
bridge.external_interfaces	string	-	-
bridge.hwaddr	string	-	-
bridge.mode	string	-	standard
bridge.mtu	integer	-	1500

キー	型	条件	デフォルト
dns.domain	string	-	lxd
dns.mode	string	-	managed
dns.search	string	-	-
dns.zone.forward	string	-	managed
dns.zone.reverse.ipv4	string	-	managed
dns.zone.reverse.ipv6	string	-	managed
fan.overlay_subnet	string	ファンモード	240.0.0.0/8
fan.type	string	ファンモード	vxlan
fan.underlay_subnet	string	ファンモード	auto(作成時のみ)
ipv4.address	string	標準モード	- (作成時の初期値: auto)
ipv4.dhcp	bool	IPv4 アドレス	true
ipv4.dhcp.expiry	string	IPv4 DHCP	1h
ipv4.dhcp.gateway	string	IPv4 DHCP	IPv4 アドレス
ipv4.dhcp.ranges	string	IPv4 DHCP	全てのアドレス
ipv4.firewall	bool	IPv4 アドレス	true
ipv4.nat	bool	IPv4 アドレス	false (ipv4.address が auto の場合の作
ipv4.nat.address	string	IPv4 アドレス	-
ipv4.nat.order	string	IPv4 アドレス	before
ipv4.ovn.ranges	string	-	-
ipv4.routes	string	IPv4 アドレス	-
ipv4.routing	bool	IPv4 アドレス	true
ipv6.address	string	標準モード	- (作成時の初期値: auto)
ipv6.dhcp	bool	IPv6 アドレス	true
ipv6.dhcp.expiry	string	IPv6 DHCP	1h
ipv6.dhcp.ranges	string	IPv6 ステートフル DHCP	全てのアドレス
ipv6.dhcp.stateful	bool	IPv6 DHCP	false
ipv6.firewall	bool	IPv6 アドレス	true
ipv6.nat	bool	IPv6 アドレス	false (ipv6.address が auto の場合の作
ipv6.nat.address	string	IPv6 アドレス	-
ipv6.nat.order	string	IPv6 アドレス	before
ipv6.ovn.ranges	string	-	-
ipv6.routes	string	IPv6 アドレス	-
ipv6.routing	bool	IPv6 アドレス	true
maas.subnet.ipv4	string	IPv4 アドレス	-
maas.subnet.ipv6	string	IPv6 アドレス	-
raw.dnsmasq	string	-	-
security.acls	string	-	-

キー	型	条件	デフォルト
security.acls.default.egress.action	string	security.acls	reject
security.acls.default.egress.logged	bool	security.acls	false
security.acls.default.ingress.action	string	security.acls	reject
security.acls.default.ingress.logged	bool	security.acls	false
tunnel.NAME.group	string	vxlan	239.0.0.1
tunnel.NAME.id	integer	vxlan	0
tunnel.NAME.interface	string	vxlan	-
tunnel.NAME.local	string	gre か vxlan	-
tunnel.NAME.port	integer	vxlan	0
tunnel.NAME.protocol	string	標準モード	-
tunnel.NAME.remote	string	gre か vxlan	-
tunnel.NAME.ttl	integer	vxlan	1
user.*	string	-	-

サポートされている機能

bridge ネットワークタイプでは以下の機能がサポートされています。

- **ネットワーク *ACL* を設定するには**
- **ネットワークフォワードを設定するには**
- **ネットワークゾーンを設定するには**
- ***LXD* を *BGP* サーバーとして設定するには**
- ***systemd-resolved* と統合するには**

systemd-resolved と統合するには

LXD を実行するシステムが DNS ルックアップの実行に systemd-resolved を使用する場合、resolved に LXD が名前解決できるドメインを通知するべきです。そうするには、 LXD ネットワークブリッジにより提供される DNS サーバーとドメインを resolved の設定に追加してください。

注釈: この機能を使いたい場合、 dns.mode オプション (**設定オプション** 参照) を managed か dynamic に設定する必要があります。

dns.domain の設定によっては、 DNS 名前解決を許可するため resolved の DNSSEC を無効化する必要があるかもしれません。これは resolved.conf 内の DNSSEC オプションにより実現できます。

---

resolved を設定する

ネットワークブリッジを resolved 設定に追加するには、対応するブリッジの DNS アドレスとドメインを指定します。

DNS アドレス
: IPv4 アドレス、 IPv6 アドレス、あるいは両方を使用できます。アドレスはサブネットのネットマスク無しで指定する必要があります。

  ブリッジの IPv4 アドレスを取得するには以下のコマンドを使用します。

```
lxc network get <network_bridge> ipv4.address
```

  ブリッジの IPv6 アドレスを取得するには以下のコマンドを使用します。

```
lxc network get <network_bridge> ipv6.address
```

DNS ドメイン
: ブリッジの DNS ドメイン名を取得するには以下のコマンドを使用します。

```
lxc network get <network_bridge> dns.domain
```

  このオプションが設定されていない場合、デフォルトのドメイン名は lxd です。

resolved を設定するには以下のコマンドを使用します。

```
resolvectl dns <network_bridge> <dns_address>
resolvectl domain <network_bridge> ~<dns_domain>
```

注釈: resolved で DNS ドメインを指定する場合、ドメイン名に ~ の接頭辞をつけてください。~ により resolved がこのドメインをルックアップするためだけに対応するネームサーバーを使うようになります。

ご利用のシェルによっては ~ が展開されるのを防ぐために DNS ドメインを引用符で囲む必要があるかもしれません。

例えば以下のようにします。

```
resolvectl dns lxdbr0 192.0.2.10
resolvectl domain lxdbr0 '~lxd'
```

注釈: 別の方法として、 systemd-resolve コマンドを使用することもできます。このコマンドは systemd の新しいリリースでは廃止予定となっていますが、後方互換性のため引き続き提供されています。

```
systemd-resolve --interface <network_bridge> --set-domain ~<dns_domain> --set-dns <dns_
↪address>
```

resolved の設定はブリッジが存在する限り残ります。リブートのたびに LXD が再起動した後に上記のコマンドを実行するか、下記のように設定を永続的にする必要があります。

resolved の設定を永続的にする

システムの起動時に適用され LXD がネットワークインタフェースを作成したときに有効になるように systemd-resolved の DNS 設定を自動化できます。

そうするには、 /etc/systemd/system/lxd-dns-<network_bridge>.service という名前の systemd ユニットファイルを以下の内容で作成してください。

```
[Unit]
Description=LXD per-link DNS configuration for <network_bridge>
BindsTo=sys-subsystem-net-devices-<network_bridge>.device
After=sys-subsystem-net-devices-<network_bridge>.device

[Service]
Type=oneshot
ExecStart=/usr/bin/resolvectl dns <network_bridge> <dns_address>
ExecStart=/usr/bin/resolvectl domain <network_bridge> <dns_domain>
ExecStopPost=/usr/bin/resolvectl revert <network_bridge>
RemainAfterExit=yes

[Install]
WantedBy=sys-subsystem-net-devices-<network_bridge>.device
```

ファイル名と内容で <network_bridge> をブリッジの名前 (例えば lxdbr0) に置き換えてください。さらに <dns_address> と <dns_domain> を *resolved* を設定する に書かれているように置き換えてください。

次に以下のコマンドでサービスの自動起動を有効にし起動します。

```
sudo systemctl daemon-reload
sudo systemctl enable --now lxd-dns-<network_bridge>
```

(LXD が既に実行中のため) 対応するブリッジが既に存在する場合、以下のコマンドでサービスが起動したかを確認できます。

```
sudo systemctl status lxd-dns-<network_bridge>.service
```

以下のような出力になるはずです。

```
user@host:~$ sudo systemctl status lxd-dns-lxdbr0.service        ● lxd-dns-lxdbr0.service
- LXD per-link DNS configuration for lxdbr0 Loaded:  loaded (/etc/systemd/system/
lxd-dns-lxdbr0.service; enabled; vendor preset:  enabled) Active:  inactive (dead)
since Mon 2021-06-14 17:03:12 BST; 1min 2s ago Process:  9433 ExecStart=/usr/bin/
resolvectl dns lxdbr0 n.n.n.n (code=exited, status=0/SUCCESS) Process:  9434 ExecStart=/
usr/bin/resolvectl domain lxdbr0 ~lxd (code=exited, status=0/SUCCESS) Main PID: 9434
(code=exited, status=0/SUCCESS)
```

resolved に設定が反映されたか確認するには、 resolvectl status <network_bridge> を実行します。

```
user@host:~$ resolvectl status lxdbr0     Link 6 (lxdbr0) Current Scopes:  DNSDefaultRoute
setting:  no LLMNR setting:  yesMulticastDNS setting:  no DNSOverTLS setting:  no DNSSEC
setting:  no DNSSEC supported:  no Current DNS Server:  n.n.n.n DNS Servers:  n.n.n.n DNS
Domain:  ~lxd
```

ファイアウォールを設定するには

Linux のファイアウォールは netfilter をベースにしています。LXD は同じサブシステムを使用しているため、接続に問題を引き起こすことがありえます。

ファイアウォールを動かしている場合、 LXD が管理しているブリッジとホストの間のネットワークトラフィックを許可するように設定する必要があるかもしれません。そうしないと、一部のネットワークの機能 (DHCP、DNS と外部ネットワークへのアクセス) が期待通り動かないかもしれません。

ファイアウォール (あるいは他のアプリケーション) に設定されたルールと LXD が追加するファイアウォールのルールが衝突するケースがあります。例えば、ファイアウォールが LXD デーモンより後に起動した場合ファイアウォールが LXD のルールを削除するかもしれず、そうするとインスタンスへのネットワーク接続を妨げるかもしれません。

xtables 対 nftables

netfilter にルールを追加するには xtables (IPv4 には iptables と IPv6 には ip6tables) と nftables という異なるユーザースペースのコマンドがあります。

xtables は順序ありのルールのリストを提供しますが、そのため複数のシステムがルールの追加や削除を行うと問題が起きるかもしれません。nftables は分離されたルールを別々のネームスペースに追加することができますので、異なるアプリケーションからのルールを分離するのに役立ちます。しかし、パケットが 1 つのネームスペースでブロックされる場合、他のネームスペースがそれを許可することはできません。そのため、 1 つのネームスペースが他のネームスペースのルールへ影響することは依然としてあり、ファイアウォールのアプリケーションが LXD のネットワーク機能に影響することがありえます。

システムで nftables を利用可能な場合、 LXD はそれを検知して nftables モードにスイッチします。このモードでは LXD は自身の nftables のネームスペースを用いてルールを nftables に追加します。

LXD のファイアウォールを使用する

デフォルトでは LXD が管理するブリッジはフル機能を使えるようにするためファイアウォールにルールを追加します。システムで他のファイアウォールを使用していない場合は LXD にファイアウォールのルールを管理させることができます。

これを有効または無効にするには ipv4.firewall または ipv6.firewall **設定オプション** を使用してください。

別のファイアウォールを使用する

別のアプリケーションが追加するファイアウォールのルールは LXD が追加するファイアウォールルールと干渉するかもしれません。このため、別のファイアウォールを使用する場合は LXD のファイアウォールルールを無効にするべきです。また LXD のインスタンスがホスト上で LXD が動かしている DHCP と DNS サーバーにアクセスできるようにするため、インスタンスと LXD ブリッジ間のネットワークトラフィックを許可するように設定しなければなりません。

LXD のファイアウォールルールをどのように無効化し、 firewalld と UFW をどのように適切に設定するかは以下を参照してください。

LXD のファイアウォールルールを無効化する

指定のネットワークブリッジ (例えば lxdbr0) に LXD がファイアウォールルールを設定しないようにするためには以下のコマンドを実行してください。

```
lxc network set <network_bridge> ipv6.firewall false
lxc network set <network_bridge> ipv4.firewall false
```

firewalld で信頼されたゾーンにブリッジを追加する

firewalld で LXD ブリッジへとブリッジからのトラフィックを許可するには、ブリッジインタフェースを trusted ゾーンに追加してください。(再起動後も設定が残るように) 恒久的にこれを行うには以下のコマンドを実行してください。

```
sudo firewall-cmd --zone=trusted --change-interface=<network_bridge> --permanent
sudo firewall-cmd --reload
```

例えば

```
sudo firewall-cmd --zone=trusted --change-interface=lxdbr0 --permanent
sudo firewall-cmd --reload
```

警告: 上に示したコマンドはシンプルな例です。あなたの使い方に応じて、より高度なルールが必要な場合があり、その場合上の例をそのまま実行するとうっかりセキュリティリスクを引き起こす可能性があります。

UFW でブリッジにルールを追加する

UFW で認識不能なトラフィックを全てドロップするルールを入れていると、 LXD ブリッジへとブリッジからのトラフィックをブロックしてしまいます。この場合ブリッジへとブリッジからのトラフィックを許可し、さらにブリッジへフォワードされるトラフィックを許可するルールを追加する必要があります。

そのためには次のコマンドを実行します。

```
sudo ufw allow in on <network_bridge>
sudo ufw route allow in on <network_bridge>
sudo ufw route allow out on <network_bridge>
```

例えば

```
sudo ufw allow in on lxdbr0
sudo ufw route allow in on lxdbr0
sudo ufw route allow out on lxdbr0
```

警告: 上に示したコマンドはシンプルな例です。あなたの使い方に応じて、より高度なルールが必要な場合があり、その場合上の例をそのまま実行するとうっかりセキュリティリスクを引き起こす可能性があります。

LXD と Docker の接続の問題を回避する

同じホストで LXD と Docker を動かすと接続の問題を引き起こします。この問題のよくある理由は Docker はグローバルの FOWARD のポリシーを drop に設定するので、それが LXD がトラフィックをフォワードすることを妨げインスタンスのネットワーク接続を失わせるということです。詳細は Docker on a router を参照してください。

この問題を回避するためのさまざまな方法があります：

Docker をアンインストールする こ
: のような問題を防ぐ最も簡単な方法は、LXD を実行しているシステムから Docker をアンインストールしてシステムを再起動することです。代わりに、LXD のコンテナや仮想マシンの中で Docker を実行できます。

  詳細情報については、LXD のコンテナの中で Docker を実行するを参照してください。

IPv4 の転送を有効にする
: Docker をアンインストールすることができない場合、Docker サービスが開始する前に IPv4 転送を有効にすることで、Docker がグローバル FORWARD ポリシーを変更するのを防ぐことができます。LXD ブリッ

ジネットワークは通常、この設定を有効にします。ただし、LXD が Docker の後に起動すると、Docker は既にグローバル FORWARD ポリシーを変更している可能性があります。

> **警告:** IPv4 の転送を有効にすると、Docker のコンテナポートがローカルネットワーク上の任意のマシンからアクセス可能になる可能性があります。環境によりますが、これは望ましくない場合があります。詳細については、ローカルネットワークのコンテナアクセス問題を参照してください。

Docker が開始する前に IPv4 転送を有効にするためには、次の `sysctl` 設定が有効になっていることを確認します：

```
net.ipv4.conf.all.forwarding=1
```

重要: この設定はホストの再起動時にも保持されるようにする必要があります。

これを行う一つの方法は、次のコマンドを使用して`/etc/sysctl.d/`ディレクトリにファイルを追加することです：

```
echo "net.ipv4.conf.all.forwarding=1" > /etc/sysctl.d/99-forwarding.conf
systemctl restart systemd-sysctl
```

外向きネットワークトラフィックフローを許可する

Docker のコンテナポートがローカルネットワーク上の任意のマシンからアクセス可能になる可能性を避けたい場合、Docker が提供するより複雑なソリューションを適用できます。

次のコマンドを使用して、LXD 管理ブリッジインターフェースからの外向きネットワークトラフィックフローを明示的に許可します：

```
iptables -I DOCKER-USER -i <network_bridge> -j ACCEPT
iptables -I DOCKER-USER -o <network_bridge> -m conntrack --ctstate RELATED,
↪ESTABLISHED -j ACCEPT
```

例えば、LXD 管理ブリッジが `lxdbr0` と呼ばれている場合、次のコマンドを使用して外向きトラフィックのフローを許可できます：

```
iptables -I DOCKER-USER -i lxdbr0 -j ACCEPT
iptables -I DOCKER-USER -o lxdbr0 -m conntrack --ctstate RELATED,ESTABLISHED -j␣
↪ACCEPT
```

重要: これらのファイアウォールルールは、ホストの再起動時にも保持されるようにする必要があります。

これを行う方法は Linux ディストリビューションによります。

3.7.8 OVN ネットワーク

OVN は仮想ネットワーク抽象化をサポートするソフトウェアで定義されたネットワークシステムです。あなた自身のプライベートクラウドを構築するのに使用できます。詳細は `www.ovn.org` をご参照ください。

ovn ネットワークタイプは OVNSDN (software-defined networking) を使って論理的なネットワークの作成を可能にします。この種のネットワークは複数の個別のネットワーク内で同じ論理ネットワークのサブネットを使うような検証環境やマルチテナントの環境で便利です。

LXD の OVN ネットワークはより広いネットワークへの外向きのアクセスを可能にするため既存の管理された**ブリッジネットワーク**や**物理ネットワーク**に接続できます。デフォルトでは、OVN 論理ネットワークからの全ての接続はアップリンクのネットワークによって割り当てられた IP に NAT されます。

OVN ネットワークをセットアップする基本的な手順については *LXD で OVN をセットアップするには*をご参照ください。

注釈: 静的な DHCP 割当は MAC アドレスを DHCP 識別子として使用するクライアントに依存します。この方法はインスタンスをコピーする際に衝突するリースを回避し、静的に割り当てられたリースが正しく動くようにします。

設定オプション

ovn ネットワークタイプでは現在以下の設定キーネームスペースがサポートされています。

- `bridge` (L2 インタフェースの設定)
- `dns` (DNS サーバーと名前解決の設定)
- `ipv4` (L3 IPv4 設定)
- `ipv6` (L3 IPv6 設定)
- `security` (ネットワーク ACL 設定)
- `user` (key/value の自由形式のユーザメタデータ)

注釈: ネットワークのサブネット情報を指定する箇所では LXD は CIDR 表記 (例えば `192.0.2.0/24` や `2001:db8::/32`) を使用します。これは単一のアドレスが必要なケース (例えば、トンネルのローカル／リモート

アドレス、インスタンスに適用する NAT アドレスや特定のアドレス) では適用されません。

ovn ネットワークタイプには以下の設定オプションがあります。

キー	型	条件	デフォルト	説明
network	string	-	-	外部ネットワークへのアクセスに使うアップリンクのネットワーク
bridge.hwaddr	string	-	-	ブリッジの MAC アドレス
bridge.mtu	integer	-	1442	ブリッジの MTU(デフォルトではホストからホストへの Geneve トンネルを許可します)
dns.domain	string	-	lxd	DHCP のクライアントに広告し DNS の名前解決に使用するドメイン
dns.search	string	-	-	完全なドメインサーチのカンマ区切りリスト (デフォルトは dns.domain の値)
dns.zone.forward	string	-	-	正引き DNS レコード用の DNS ゾーン名のカンマ区切りリスト
dns.zone.reverse.ipv4	string	-	-	IPv4 逆引き DNS レコード用の DNS ゾーン名
dns.zone.reverse.ipv6	string	-	-	IPv6 逆引き DNS レコード用の DNS ゾーン名
ipv4.address	string	標準モード	- (作成時の初期値: auto)	ブリッジの IPv4 アドレス (CIDR 形式)。IPv4 をオフにするには none、新しいランダムな未使用のサブネットを生成するには auto を指定。
ipv4.dhcp	bool	IPv4 アドレス	true	DHCP を使ってアドレスを割り当てるかどうか
ipv4.l3only	bool	IPv4 address	false	layer 3 only モード を有効にするかどうか
ipv4.nat	bool	IPv4 アドレス	false(ipv4.address が auto の場合の作成時の初期値: true)	NAT するかどうか
ipv4.nat.address	string	IPv4 アドレス	-	ネットワークからの外向きトラフィックに使用されるソースアドレス (アップリンクに ovn.ingress_mode=routed が必要)
ipv6.address	string	標準モード	- (作成時の初期値: auto)	ブリッジの IPv6 アドレス (CIDR 形式)。IPv6 をオフにするには none、新しいランダムな未使用のサブネットを生成するには auto を指定。
ipv6.dhcp	bool	IPv6 アドレス	true	DHCP 上に追加のネットワーク設定を提供するかどうか
ipv6.dhcp.stateful	bool	IPv6 DHCP	false	DHCP を使ってアドレスを割り当てるかどうか
ipv6.l3only	bool	IPv6 DHCP state-	false	layer 3 only モード を有効にするかどうか

サポートされている機能

ovn ネットワークタイプでは以下の機能がサポートされています。

- ネットワーク *ACL* を設定するには
- ネットワークフォワードを設定するには
- ネットワークゾーンを設定するには
- ピアルーティング関係を作成するには
- ネットワークロードバランサーを設定するには

LXD で OVN をセットアップするには

スタンドアロンのネットワークとしてまたは小さな LXD クラスタとして基本的な OVN ネットワークをセットアップするには以下の項を参照してください。

スタンドアロンの OVN ネットワークをセットアップする

外向きの接続のために LXD が管理する親のブリッジネットワーク (例: `lxdbr0`) に接続するスタンドアロンの OVN ネットワークを作成するには以下の手順を実行してください。

1. ローカルサーバーに OVN ツールをインストールします。

```
sudo apt install ovn-host ovn-central
```

2. OVN の統合ブリッジを設定します。

```
sudo ovs-vsctl set open_vswitch . \
  external_ids:ovn-remote=unix:/var/run/ovn/ovnsb_db.sock \
  external_ids:ovn-encap-type=geneve \
  external_ids:ovn-encap-ip=127.0.0.1
```

3. OVN ネットワークを作成します。

```
lxc network set <parent_network> ipv4.dhcp.ranges=<IP_range> ipv4.ovn.ranges=<IP_
↪range>
lxc network create ovntest --type=ovn network=<parent_network>
```

4. `ovntest` ネットワークを使用するインスタンスを作成します。

```
lxc init ubuntu:22.04 c1
lxc config device override c1 eth0 network=ovntest
lxc start c1
```

5. lxc list を実行してインスタンスの情報を表示します。

```
user@host:~$ lxc list+------+---------+----------------------+----------------------------------
NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |+------+---------+----------------------+----------
c1 | RUNNING | 192.0.2.2 (eth0) | 2001:db8:cff3:5089:216:3eff:fef0:549f (eth0) |
CONTAINER | 0 |+------+---------+----------------------+----------------------------------------
```

OVN 上に LXD クラスタをセットアップする

OVN ネットワークを使用する LXD クラスタをセットアップするには以下の手順を実行してください。

LXD と同様に、OVN の分散データベースは奇数のメンバーで構成されるクラスタ上で動かす必要があります。以下の手順は最小構成の 3 台のサーバーを使います。 3 台のサーバーでは OVN の分散データベースと OVN コントローラの両方を動かします。さらに LXD クラスタに OVN コントローラのみを動かすサーバーを任意の台数追加できます。4 台のマシンを使う完全なチュートリアルは上にリンクした YouTube の動画を参照してください。

1. OVN の分散データベースを動かしたい 3 台のマシンで次の手順を実行してください。
   1. OVN ツールをインストールします。

      ```
      sudo apt install ovn-central ovn-host
      ```

   2. マシンの起動時に OVN サービスが起動されるように自動起動を有効にします。

      ```
      systemctl enable ovn-central
      systemctl enable ovn-host
      ```

   3. OVN を停止します。

      ```
      systemctl stop ovn-central
      ```

   4. マシンの IP アドレスをメモします。

      ```
      ip -4 a
      ```

   5. /etc/default/ovn-central を編集します。
   6. 以下の設定をペーストします (<server_1>, <server_2> and <server_3> をそれぞれのマシンの IP アドレスに、 <local> をあなたがいるマシンの IP アドレスに置き換えてください)。

- 最初のマシン

```
OVN_CTL_OPTS=" \
     --db-nb-addr=<local> \
     --db-nb-create-insecure-remote=yes \
     --db-sb-addr=<local> \
     --db-sb-create-insecure-remote=yes \
     --db-nb-cluster-local-addr=<local> \
     --db-sb-cluster-local-addr=<local> \
     --ovn-northd-nb-db=tcp:<server_1>:6641,tcp:<server_2>:6641,tcp:<server_
↪3>:6641 \
     --ovn-northd-sb-db=tcp:<server_1>:6642,tcp:<server_2>:6642,tcp:<server_
↪3>:6642"
```

- 2 番目と 3 番目のマシン

```
OVN_CTL_OPTS=" \
      --db-nb-addr=<local> \
     --db-nb-cluster-remote-addr=<server_1> \
     --db-nb-create-insecure-remote=yes \
     --db-sb-addr=<local> \
     --db-sb-cluster-remote-addr=<server_1> \
     --db-sb-create-insecure-remote=yes \
     --db-nb-cluster-local-addr=<local> \
     --db-sb-cluster-local-addr=<local> \
     --ovn-northd-nb-db=tcp:<server_1>:6641,tcp:<server_2>:6641,tcp:<server_
↪3>:6641 \
     --ovn-northd-sb-db=tcp:<server_1>:6642,tcp:<server_2>:6642,tcp:<server_
↪3>:6642"
```

7. OVN を起動します。

```
systemctl start ovn-central
```

2. 残りのマシンでは ovn-host のみインストールし、自動起動を有効にしてください。

```
sudo apt install ovn-host
systemctl enable ovn-host
```

3. 全てのマシンで Open vSwitch (変数は上記の通りに置き換えてください) を設定します。

```
sudo ovs-vsctl set open_vswitch . \
   external_ids:ovn-remote=tcp:<server_1>:6642,tcp:<server_2>:6642,tcp:<server_3>
↪:6642 \
   external_ids:ovn-encap-type=geneve \
   external_ids:ovn-encap-ip=<local>
```

4. 全てのマシンで `lxd init` を実行して LXD クラスタを作成してください。最初のマシンでクラスタを作成します。次に最初のマシンで `lxc cluster add <machine_name>` を実行してトークンを出力し、他のマシンで LXD を初期化する際にトークンを指定して他のマシンをクラスタに参加させます。

5. 最初のマシンでアップリンクネットワークを作成し設定します。

```
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
↪<machine_name_1>
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
↪<machine_name_2>
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
↪<machine_name_3>
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
↪<machine_name_4>
lxc network create UPLINK --type=physical \
   ipv4.ovn.ranges=<IP_range> \
   ipv6.ovn.ranges=<IP_range> \
   ipv4.gateway=<gateway> \
   ipv6.gateway=<gateway> \
   dns.nameservers=<name_server>
```

必要な値を決定します。

アップリンクネットワーク ア

クティブな OVN シャーシがクラスタメンバー間で移動できるようにするため、ハイアベイラビリティな OVN クラスタには共有されたレイヤー 2 ネットワークが必須です (これにより OVN のルータの外部 IP が実質的に別のホストから到達可能にできます)。

そのため管理されていないブリッジインタフェースまたは使用されていない物理インタフェースを OVN アップリンクで使用される物理ネットワークの親として指定する必要があります。以下の手順は手動で作成した管理されていないブリッジを使用する想定です。このブリッジをセットアップする手順は ネットワークブリッジの設定 を参照してください。

ゲートウェイ

`ip -4 route show default` と `ip -6 route show default` を実行してください。

ネームサーバー

resolvectl を実行してください。

IP の範囲 割り当てられた IP を元に適切な IP の範囲を使用してください。

6. 引き続き最初のマシンで LXD を OVN DB クラスタと通信できるように設定します。そのためには /etc/default/ovn-central 内の ovn-northd-nb-db の値を確認し、以下のコマンドで LXD に指定します。

```
lxc config set network.ovn.northbound_connection <ovn-northd-nb-db>
```

7. 最後に (最初のマシンで) 実際の OVN ネットワークを作成します。

```
lxc network create my-ovn --type=ovn
```

8. OVN ネットワークをテストするには、インスタンスを作成してネットワークが接続できるか確認します。

```
lxc launch images:ubuntu/22.04 c1 --network my-ovn
lxc launch images:ubuntu/22.04 c2 --network my-ovn
lxc launch images:ubuntu/22.04 c3 --network my-ovn
lxc launch images:ubuntu/22.04 c4 --network my-ovn
lxc list
lxc exec c4 bash
ping <IP of c1>
ping <nameserver>
ping6 -n www.linuxcontainers.org
```

ピアルーティング関係を作成するには

デフォルトでは、 2 つの OVN ネットワーク間のトラフィックはアップリンクのネットワークを経由します。しかし、パケットが OVN サブシステムから出てホストのネットワークスタック (そして、場合によっては外部ネットワーク) を通過し対象ネットサークの OVN サブシステムに戻る必要があるため、この経路は非効率です。ホストのネットワークの構成によっては、利用できる帯域幅が制限される場合があります（ホストの外部ネットワークより OVN のオーバーレイネットワークが広帯域幅のネットワークにある場合)。

このため、 LXD では 2 つの OVN ネットワーク間でルーティング関係を作成できます。この方法を使うと 2 つのネットワーク間での通信がアップリンクのネットワーク経由ではなく OVN サブシステム内で完結できます。

ネットワーク間にルーティング関係を作成する

2 つのネットワーク間にルーティング関係を作成するには、両方のネットワークにネットワークピアを作成する必要があります。関係は双方向でなくてはなりません。1 のネットワークだけセットアップした場合、ルーティング関係はペンディング状態になり、アクディブにはなりません。

ピアのルーティング関係を作成する際は、対象のネットワークとの関係を特定するピアの名前を指定します。名前は自由に選ぶことができ、後で関係を編集または削除する際に使用します。

同じプロジェクト内のネットワーク間でピアのルーティング関係を作成するには次のコマンドを使います。

```
lxc network peer create <network1> <peering_name> <network2> [configuration_options]
lxc network peer create <network2> <peering_name> <network1> [configuration_options]
```

別のプロジェクトの OVN ネットワーク間でピアのルーティング関係を作成することもできます。

```
lxc network peer create <network1> <peering_name> <project2/network2> [configuration_
↪options] --project=<project1>
lxc network peer create <network2> <peering_name> <project1/network1> [configuration_
↪options] --project=<project2>
```

重要: プロジェクトまたはネットワークの名前が間違っている場合、コマンドは対応するプロジェクトやネットワークが存在しないというエラーは出さず、ルーティング関係はペンディング状態のままになります。これは他のプロジェクトのユーザがプロジェクトやネットワークが存在するか調べられないようにするための (訳注: セキュリティ上の) 仕様です。

ピアのプロパティ

ピアのルーティング関係には以下のプロパティがあります。

プロパティ	型	必須	説明
name	string	yes	ローカルネットワーク上のネットワークピアの名前
description	string	no	ネットワークピアの説明
config	string set	no	設定のキーバリューペアー (user.* のカスタムキーのみサポート)
target_projec	string	yes	対象のネットワークがどのプロジェクト内に存在するか (作成時に必須)
target_networ	string	yes	どのネットワークとピアを作成するか (作成時に必須)
status	string	--	作成中か作成完了 (対象のネットワークと相互にピアリングした状態) かを示すステータス

ルーティング関係の一覧を表示する

ネットワークのネットワークピア全ての一覧を表示するには次のコマンドを実行します。

```
lxc network peer list <network>
```

ルーティング関係を編集する

ネットワークピアを編集するには次のコマンドを実行します。

```
lxc network peer edit <network> <peering_name>
```

このコマンドは YAML 形式のネットワークピアを編集モードで開きます。

ネットワークロードバランサーを設定するには

注釈: ネットワークロードバランサーは現状では *OVN* **ネットワーク** でのみ利用できます.

ネットワークロードバランサーは、外部 IP アドレス上の特定のポートを、ロードバランサーが属するネットワークの内部 IP アドレス上の特定のポートにフォワードできるという点で、ネットワークフォワードに似ています。ネットワークロードバランサーとネットワークフォワードの違いは、ロードバランサーは内向きのトラフィックを複数の内部のバックエンドアドレスで共有するのに使えることです。

この機能は外部 IP アドレスの数に限りがあったり、複数のインスタンスで単一の外部アドレスとそのアドレス上のポートを共有したい場合に有用です。

ロードバランサーは以下の要素で構成されます。

- 単一の外部リッスン IP アドレス。
- 内部 IP アドレスとオプショナルなポートレンジからなる単一あるいは複数の名前付きバックエンド。
- 単一または複数の名前付きバックエンドにフォワードされるように設定された単一または複数のリッスンポートレンジ。

ネットワークロードバランサーを作成する

ネットワークロードバランサーを作成するには以下のコマンドを使用します。

```
lxc network load-balancer create <network_name> <listen_address> [configuration_options..
↪.]
```

それぞれのロードバランサーはネットワークに割り当てられます。ロードバランサーには単一の外部リッスンアドレスが必要です (どのアドレスがロードバランス可能かについてのさらなる情報は **リッスンアドレスの要件** 参照)。

ロードバランサーのプロパティ

ネットワークロードバランサーは以下のプロパティを持ちます。

プロパティ	型	必須	説明
listen_addres	string	yes	リッスンする IP アドレス
description	string	no	ネットワークロードバランサーの説明
config	string set	no	キー/バリュー形式の設定オプション (user.* カスタムキーのみがサポートされます)
backends	backend list	no	**バックエンド仕様** のリスト
ports	port list	no	**ポート仕様** のリスト

リッスンアドレスの要件

有効なリッスンアドレスは以下の要件を満たす必要があります。

- 許可されるリッスンアドレスはアップリンクのネットワークの ipv{n}.routes 設定またはプロジェクトの restricted.networks.subnets 設定 (設定されている場合) に定義されている必要があります。
- リッスンアドレスは他のネットワークやネットワーク内のエンティティで使用されているサブネットと重なってはいけません。

バックエンドを設定する

ターゲットのアドレス (と省略可能なポート) をネットワークロードバランサーに定義するためにバックエンド仕様を追加できます。バックエンドのターゲットアドレスはロードバランサーが関連付けられているネットワークと同じサブネット内である必要があります。

バックエンド仕様を追加するには以下のコマンドを使用します。

```
lxc network load-balancer backend add <network_name> <listen_address> <backend_name>
↪<listen_ports> <target_address> [<target_ports>]
```

ターゲットポートは省略可能です。省略した場合、ロードバランサーはバックエンドのリッスンポートをバックエンドのターゲットポートとして使用します。

トラフィックを別のポートにフォワードしたい場合、2 つの選択肢があります。

- 単一のターゲットポートを指定し、全てのリッスンポートへのトラフィックをこのターゲットポートにフォワードする。
- 一組のターゲットポートをリッスンポートと同じ数のポートで指定し、リッスンポートの最初のポートをターゲットポートの最初のポート、リッスンポートの 2 番目のポートをターゲットポートの 2 番目のポート、というようにトラフィックをフォワードする。

バックエンドのプロパティ

ネットワークロードバランサーのバックエンドは以下のプロパティを持ちます。

プロパティ	型	必須	説明
name	string	yes	バックエンドの名前
target_address	string	yes	フォワード先の IP アドレス
target_port	string	no	ターゲットポート (例 70,80-90 や 90)、空の場合 **ポート** の listen_port と同じ
description	string	no	バックエンドの説明

ポートを設定する

ネットワークロードバランサーにポート指定を追加し、リッスンアドレスの特定のポートから、単一または複数のバックエンド上の特定のポートにトラフィックを転送できます。

ポート仕様を追加するには以下のコマンドを使用します。

```
lxc network load-balancer port add <network_name> <listen_address> <protocol> <listen_
↪ports> <backend_name>[,<backend_name>...]
```

単一のリッスンポートまたは一組のポートを指定できます。指定されたバックエンドはポートのリッスンポート設定と互換性があるターゲットポート設定を持たなければなりません。

ポートのプロパティ

ネットワークロードバランサーのポートは以下のプロパティを持ちます。

プロパティ	型	必須	説明
protocol	string	yes	ポートのプロトコル (tcp または udp)
listen_port	string	yes	リッスンポート (例 80,90-100)
target_backend	backend list	yes	フォワード先のバックエンドの名前
description	string	no	ポートの説明

ネットワークロードバランサーを編集する

ネットワークロードバランサーを編集するには以下のコマンドを使用します。

```
lxc network load-balancer edit <network_name> <listen_address>
```

このコマンドは YAML 形式のネットワークロードバランサーの設定を編集用に開きます。一般の設定、バックエンド、ポート仕様を編集できます。

ネットワークロードバランサーを削除する

ネットワークロードバランサーを削除するには以下のコマンドを使用します。

```
lxc network load-balancer delete <network_name> <listen_address>
```

3.7.9 外部ネットワーク

外部ネットワークは既に存在するネットワークを使用します。そのため、 LXD がそれらを制御するには限界があるため、ネットワーク ACL、ネットワークフォワードやネットワークゾーンのような LXD の機能はサポートされません。

外部ネットワークを使用する主な目的は親インタフェースによるアップリンクのネットワークを提供することです。この外部ネットワークはインスタンスや他のネットワークを親のインタフェースに接続する際のプリセットを指定します。

LXD は以下の外部ネットワークタイプをサポートします。

macvlan ネットワーク

macvlan は仮想的な LAN で同じネットワークインタフェースに複数の IP アドレスを割り当てたい場合に使用できます。基本的にはネットワークインタフェースをそれぞれの IP アドレスを持つ複数のサブインタフェースに分割することになります。その後ランダムに生成された MAC アドレスに基づいて IP アドレスを設定できます。

`macvlan` ネットワークタイプは親のインタフェースにインスタンスを接続する際に使用するプリセットを指定できます。この場合、接続先のネットワークについて基本的な設定詳細を一切知る必要なしに単に `network` オプションをインスタンス NIC に設定できます。

注釈: `macvlan` NIC を使う場合、LXD ホストとインスタンス間の通信はできません。ホストとインスタンスの両方がゲートウェイと通信できますが、それらが直接通信はできません。

設定オプション

macvlan ネットワークタイプでは現在以下の設定キーネームスペースがサポートされています。

- maas (MAAS ネットワーク識別)
- user (key/value の自由形式のユーザメタデータ)

注釈: ネットワークのサブネット情報を指定する箇所では LXD は CIDR 表記 (例えば 192.0.2.0/24 や 2001:db8::/32) を使用します。これは単一のアドレスが必要なケース (例えば、トンネルのローカル／リモートアドレス、インスタンスに適用する NAT アドレスや特定のアドレス) では適用されません。

---

macvlan ネットワークタイプでは以下の設定オプションが使用できます。

キー	型	条件	デフォルト	説明
gvrp	bool	-	false	GARP VLAN Registration Protocol を使って VLAN を登録する
mtu	integer	-	-	作成するインタフェースの MTU
parent	string	-	-	macvlan NIC を作成する親のインタフェース
vlan	integer	-	-	アタッチする先の VLAN ID
maas.subnet.ipv4	string	IPv4 アドレス	-	インスタンスを登録する MAAS IPv4 サブネット（NIC の network プロパティを使用する場合）
maas.subnet.ipv6	string	IPv6 アドレス	-	インスタンスを登録する MAAS IPv6 サブネット（NIC の network プロパティを使用する場合）
user.*	string	-	-	ユーザ指定の自由形式のキー／バリューペア

SR-IOV ネットワーク

SR-IOV は仮想環境内で単一のネットワークポートを複数の仮想ネットワークインタフェースのように見せるように出来るハードウェア標準です。

sriov ネットワークタイプは親のインタフェースに接続する際に使用するプリセットを指定できるようにします。この場合接続先の設定詳細を一切知ること無くインスタンス NIC に単に network オプションを設定できます。

設定オプション

sriov ネットワークでは現在以下の設定キーネームスペースがサポートされています。

- maas (MAAS ネットワーク識別)
- user (key/value の自由形式のユーザメタデータ)

注釈: ネットワークのサブネット情報を指定する箇所では LXD は CIDR 表記 (例えば 192.0.2.0/24 や 2001:db8::/32) を使用します。これは単一のアドレスが必要なケース (例えば、トンネルのローカル／リモートアドレス、インスタンスに適用する NAT アドレスや特定のアドレス) では適用されません。

---

sriov ネットワークタイプには以下の設定オプションがあります。

キー	型	条件	デフォルト	説明
mtu	integer	-	-	作成するインタフェースの MTU
parent	string	-	-	sriov NIC を作成する親のインタフェース
vlan	integer	-	-	アタッチする先の VLAN ID
maas.subnet.ipv4	string	IPv4 アドレス	-	インスタンスを登録する MAAS IPv4 サブネット（NIC の network プロパティを使用する場合）
maas.subnet.ipv6	string	IPv6 アドレス	-	インスタンスを登録する MAAS IPv6 サブネット（NIC の network プロパティを使用する場合）
user.*	string	-	-	ユーザ指定の自由形式のキー／バリューペア

物理ネットワーク

物理 (physical) ネットワークタイプは既存のネットワークに接続します。これはネットワークインタフェースまたはブリッジになることができ、OVN のためのアップリンクネットワークとしての役目を果たします。

このネットワークタイプは OVN ネットワークを親インタフェースに接続する際に使用するプリセットの設定を提供したり、インスタンスが物理インタフェースを NIC として使用できるようにします。この場合、インスタンス NIC は接続先の設定詳細を知ること無く単に network オプションを設定できるようにします。

設定オプション

物理ネットワークでは現在以下の設定キーネームスペースがサポートされています。

- `bgp` (BGP ピア設定)
- `dns` (DNS サーバーと名前解決の設定)
- `ipv4` (L3 IPv4 設定)
- `ipv6` (L3 IPv6 設定)
- `maas` (MAAS ネットワーク識別)
- `ovn` (OVN 設定)
- `user` (key/value の自由形式のユーザメタデータ)

注釈: ネットワークのサブネット情報を指定する箇所では LXD は CIDR 表記 (例えば `192.0.2.0/24` や `2001:db8::/32`) を使用します。これは単一のアドレスが必要なケース (例えば、トンネルのローカル／リモートアドレス、インスタンスに適用する NAT アドレスや特定のアドレス) では適用されません。

物理ネットワークタイプには以下の設定オプションがあります。

キー	型	条件	デフォルト	説明
gvrp	bool	-	false	GARP VLAN Registration Protocol を使って VLAN を登録する
mtu	integer	-	-	作成するインタフェースの MTU
parent	string	-	-	ネットワークで使う既存のインタフェース
vlan	integer	-	-	アタッチする先の VLAN ID
bgp.peers.NAME.address	string	BGP server	-	ovn ダウンストリームネットワークで使用するピアアドレス (IPv4 か IPv6)
bgp.peers.NAME.asn	integer	BGP server	-	ovn ダウンストリームネットワークで使用する AS 番号
bgp.peers.NAME.password	string	BGP server	- (パスワード無し)	ovn ダウンストリームネットワークで使用するピアのセッションパスワード（省略可能）
bgp.peers.NAME.holdtime	integer	BGP server	180	ピアセッションホールドタイム (秒で指定、省略可能)
dns.nameservers	string	標準モード	-	物理 (physical) ネットワークの DNS サーバー IP のリスト
ipv4.gateway	string	標準モード	-	ゲートウェイとネットワークの IPv4 アドレス（CIDR 表記）
ipv4.ovn.ranges	string	-	-	子供の OVN ネットワークルーターに使用する IPv4 アドレスの範囲（開始-終了 形式) のカンマ区切りリスト
ipv4.routes	string	IPv4 アドレス	-	子供の OVN ネットワークの ipv4.routes.external 設定で利用可能な追加の IPv4 CIDR サブネットのカンマ区切りリスト
ipv4.routes.anycast	bool	IPv4 アドレス	false	複数のネットワーク／ NIC で同時にオーバーラップするルートが使われることを許可するかどうか
ipv6.gateway	string	標準モード	-	ゲートウェイとネットワークの IPv6 アドレス（CIDR 表記）
ipv6.ovn.ranges	string	-	-	子供の OVN ネットワークルーターに使用する IPv6 アドレスの範囲（開始-終了 形式) のカンマ区切りリスト
ipv6.routes	string	IPv6 アドレス	-	子供の OVN ネットワークの ipv6.routes.external 設定で利用可能な追加の IPv6 CIDR サブネットのカンマ区切りリスト
ipv6.routes.anycast	bool	IPv6 アドレス	false	複数のネットワーク／ NIC で同時にオーバーラップするルートが使われることを許可するかどうか

サポートされている機能

物理ネットワークタイプでは以下の機能がサポートされています。

- *LXD を BGP サーバーとして設定するには*

3.8 プロジェクト

3.8.1 プロジェクトについて

LXD サーバーを整理して、関連するインスタンスをまとめるためにプロジェクトを使用できます。隔離されたインスタンスに加えて、各プロジェクトは特定のイメージ、プロファイル、ネットワーク、およびストレージを持つことができます。

例えば、プロジェクトは以下のシナリオで役立ちます:

- 異なる目的のために非常に多くのインスタンスを実行している場合、例えば、異なる顧客プロジェクトのために。これらのインスタンスを別々にして、それらを見つけやすく管理しやすくすることを望みます。また、整合性のために各顧客プロジェクトで同じインスタンス名を再利用することができます。顧客プロジェクト内の各インスタンスは、同じ基本構成（例えば、ネットワークやストレージ）を使用する必要がありますが、顧客プロジェクト間で構成が異なる場合があります。

  この場合、各顧客プロジェクト（つまり、各インスタンスのグループ）ごとに LXD プロジェクトを作成し、各 LXD プロジェクトで異なるプロファイル、ネットワーク、およびストレージを使用できます。

- LXD サーバーが複数のユーザー間で共有されている場合。各ユーザーは自分のインスタンスを実行し、自分のプロファイルを設定するかもしれません。ユーザーインスタンスを制限し、各ユーザーが自分のインスタンスのみとやり取りでき、他のユーザーが作成したインスタンスを見ることができないようにしたいです。さらに、各ユーザーのリソースを制限し、異なるユーザーのインスタンスが互いに干渉しないようにしたいです。

  この場合、制限されたプロジェクトを持つマルチユーザー環境を設定できます。

LXD には `default` プロジェクトが付属しています。プロジェクトを追加する方法については、**プロジェクトの作成と設定方法**を参照してください。

プロジェクトの隔離

プロジェクトは常にそれらが含むインスタンスをカプセル化します。これは、インスタンスがプロジェクト間で共有されず、インスタンス名が複数のプロジェクトで重複して使用できることを意味します。特定のプロジェクト内にいる場合、そのプロジェクトに属するインスタンスのみが表示されます。

他のエンティティ（イメージ、プロファイル、ネットワーク、およびストレージ）は、プロジェクト内で隔離されるか、`default` プロジェクトから継承されます。どのエンティティが隔離されているかを設定するには、プロジェクト内で対応する**機能**を有効または無効にします。機能が有効になっている場合、対応するエンティティはプロジェクト内で隔離されます。機能が無効になっている場合、`default` プロジェクトから継承されます。

例えば、プロジェクトに `features.networks` を有効にすると、プロジェクトは別のネットワークのセットを使用し、`default` プロジェクトで定義されたネットワークは使用しません。`features.images` を無効にすると、プロジェクトは `default` プロジェクトで定義されたイメージにアクセスでき、プロジェクトを使用している間に追加したイメージも `default` プロジェクトに追加されます。

プロジェクトを作成するときに有効または無効になっている機能についての情報は、利用可能な**プロジェクトの機能**のリストを参照してください。

注釈: 新しいプロジェクトを使用する前に、有効にしたい機能を選択する必要があります。プロジェクトにインスタンスが含まれている場合、機能はロックされます。それらを編集するには、まずすべてのインスタンスを削除する必要があります。

アップグレードで追加された新しい機能は、既存のプロジェクトでは無効になっています。

マルチユーザー環境での制限されたプロジェクト

LXD サーバーが複数のユーザーによって使用される場合（例えば、ラボ環境で）、プロジェクトを使用して各ユーザーの活動を制限できます。この方法は、**プロジェクトの隔離**で説明されているように、インスタンスや他のエンティティ（機能設定に依存）を隔離します。また、ユーザーを自分のユーザースペースに制限し、他のユーザーのインスタンスやデータにアクセスできないようにします。LXD サーバーやその設定に影響を与える変更、例えばストレージの追加や削除は許可されません。

さらに、この方法では、`lxd` グループのメンバーでなくても LXD を使用できます（*LXD* **デーモンへのアクセス**を参照）。`lxd` グループのメンバーは、LXD への完全なアクセス権を持っており、ファイルシステムのパスをアタッチしたり、インスタンスのセキュリティ機能を調整したりすることができます。これにより、ホストシステムへのルートアクセスが可能になります。制限されたプロジェクトを使用することで、ユーザーが LXD でできることを制限しますが、同時にユーザーがルートアクセスを得ることも防ぎます。

プロジェクトの認証方法

特定のユーザーにプロジェクトを制限するために使用できるさまざまな認証方法があります。

クライアント証明書

TLS **クライアント証明書**を制限して、特定のプロジェクトのみへのアクセスを許可することができます。アクセスを制限する前に、プロジェクトが存在している必要があります。制限された証明書を使用して接続するクライアントは、クライアントがアクセスが許可されているプロジェクトのみを見ることができます。

RBAC **認証** **役**

割ベースのアクセスコントロール *(RBAC)* を使用する場合、ロールをグローバルおよびプロジェクト単位でユーザーやグループに割り当てることができます。ロールは、プロジェクト内でユーザーが許可されている操作を定義します。この方法で、各プロジェクトを表示、使用、または管理できるユーザーを設定できます。

マルチユーザー LXD **デーモン**

LXD snap には、ユーザーごとの動的なプロジェクト作成を可能にするマルチユーザー LXD デーモンが含まれています。`lxd` グループ以外の特定のユーザーグループを設定して、グループ内のすべてのユーザーに制限された LXD アクセスを提供できます。

このグループのメンバーであるユーザーが LXD を使用し始めると、LXD は自動的にこのユーザーの制限されたプロジェクトを作成します。

snap を使用していない場合でも、ディストリビューションがサポートしていれば、この機能を使用できます。

さまざまな認証方法を有効にし、設定する方法については、**特定のユーザーにプロジェクトを制限する方法**を参照してください。

3.8.2 プロジェクトの作成と設定方法

プロジェクトは作成時または後で設定することができます。ただし、プロジェクトにインスタンスが含まれている場合、有効になっている機能を変更することはできません。

プロジェクトを作成する

プロジェクトを作成するには、`lxc project create` コマンドを使用します。

`--config` フラグを使用して設定オプションを指定できます。利用可能な設定オプションについては、**プロジェクトの設定**を参照してください。

例えば、インスタンスを分離し、デフォルトプロジェクトのイメージとプロファイルにアクセスを許可する `my-project` というプロジェクトを作成するには、次のコマンドを入力します：

```
lxc project create my-project --config features.images=false --config features.
↪profiles=false
```

セキュリティに関する機能（例えば、コンテナのネスト）へのアクセスをブロックし、バックアップを許可する my-restricted-project というプロジェクトを作成するには、次のコマンドを入力します：

```
lxc project create my-restricted-project --config restricted=true --config restricted.
↪backups=allow
```

プロジェクトの設定

プロジェクトを設定するには、特定の設定オプションを設定するか、プロジェクト全体を編集できます。

いくつかの設定オプションは、インスタンスが含まれていないプロジェクトに対してのみ設定できます。

特定の設定オプションを設定する

特定の設定オプションを設定するには、lxc project set コマンドを使用します。

例えば、my-project で作成できるコンテナの数を 5 つに制限するには、次のコマンドを入力します：

```
lxc project set my-project limits.containers=5
```

特定の設定オプションを解除するには、lxc project unset コマンドを使用します。

注釈: 設定オプションを解除すると、デフォルト値に設定されます。このデフォルト値は、プロジェクトが作成されたときに設定される初期値と異なる場合があります。

プロジェクトを編集する

プロジェクトの設定全体を編集するには、lxc project edit コマンドを使用します。

例：

```
lxc project edit my-project
```

3.8.3 異なるプロジェクトで作業する方法

default プロファイル以外にもプロジェクトがある場合、LXD で作業する際に正しいプロジェクトを使用するか、対象となるプロジェクトを確認する必要があります。

注釈: **特定のユーザーに制限されたプロジェクト**がある場合、すべてのプロジェクトを表示できるのは、LXD へのフルアクセス権を持つユーザーのみです。

フルアクセス権を持たないユーザーは、アクセス権があるプロジェクトの情報のみを表示できます。

プロジェクトの一覧表示

すべてのプロジェクト（閲覧許可があるもの）を一覧表示するには、次のコマンドを入力します：

```
lxc project list
```

デフォルトでは、出力はリスト形式で表示されます：

```
user@host:~$ lxc project list+----------------------+--------+----------+-----------------+------------
NAME | IMAGES | PROFILES | STORAGE VOLUMES | STORAGE BUCKETS | NETWORKS | NETWORK ZONES |
DESCRIPTION | USED BY |+----------------------+--------+----------+-----------------+----------------+-
default | YES | YES | YES | YES | YES | Default LXD project | 19
|+----------------------+--------+----------+-----------------+----------------+----------+-----------
my-project (current) | YES | NO | NO | NO | YES | YES | | 0 |+----------------------+--------+----------
```

異なる出力形式を要求するには、--format フラグを追加します。詳細については、lxc project list --help を参照してください。

プロジェクトの切り替え

デフォルトでは、LXD で実行するすべてのコマンドは、現在使用しているプロジェクトに影響します。どのプロジェクトを使用しているかを確認するには、lxc project list コマンドを使用します。

別のプロジェクトに切り替えるには、次のコマンドを入力します：

```
lxc project switch <project_name>
```

プロジェクトをターゲットにする

別のプロジェクトに切り替える代わりに、コマンドを実行する際に特定のプロジェクトをターゲットにすることができます。多くの LXD コマンドは、`--project` フラグをサポートしており、異なるプロジェクトでアクションを実行できます。

注釈: 許可があるプロジェクトだけをターゲットにすることができます。

以下のセクションでは、プロジェクトを切り替える代わりにターゲットにする典型的な例をいくつか紹介します。

特定のプロジェクト内のインスタンスをリストする

特定のプロジェクト内のインスタンスをリストするには、`lxc list` コマンドに`--project` フラグを追加します。

例：

```
lxc list --project my-project
```

インスタンスを別のプロジェクトに移動する

インスタンスを 1 つのプロジェクトから別のプロジェクトに移動するには、次のコマンドを入力します：

```
lxc move <instance_name> <new_instance_name> --project <source_project> --target-project
↪<target_project>
```

ターゲットプロジェクトにその名前のインスタンスが存在しない場合、同じインスタンス名を維持できます。

例えば、インスタンス `my-instance` を `default` プロジェクトから `my-project` に移動し、インスタンス名を維持するには、次のコマンドを入力します：

```
lxc move my-instance my-instance --project default --target-project my-project
```

プロファイルを別のプロジェクトにコピーする

デフォルトの設定でプロジェクトを作成すると、プロファイルはプロジェクト内で隔離されます（*features.profiles* が `true` に設定されています）。そのため、プロジェクトはデフォルトのプロファイル（`default` プロジェクトの一部）にアクセスできず、インスタンスを作成しようとすると次のようなエラーが表示されます。

`user@host:~$ lxc launch images:ubuntu/22.04 my-instance Creating my-instanceError: Failed instance creation: Failed creating instance record: Failed initialising instance: Failed getting root disk: No root device could be found` これを修正するには、`default` プロ

ジェクトのデフォルトプロファイルの内容を現在のプロジェクトのデフォルトプロファイルにコピーします。そのためには、次のコマンドを入力してください：

```
lxc profile show default --project default | lxc profile edit default
```

3.8.4 特定のユーザーにプロジェクトを制限する方法

プロジェクトを使用して、異なるユーザーまたはクライアントの活動を制限できます。詳細については、**マルチユーザー環境での制限されたプロジェクト**を参照してください。

特定のユーザーにプロジェクトを制限する方法は、選択した認証方法によって異なります。

特定の TLS クライアントにプロジェクトを制限する

LXD サーバーへの接続に使用される TLS クライアント証明書を制限することで、特定のプロジェクトへのアクセスを制限できます。詳細については、*TLS* **クライアント証明書**を参照してください。

クライアント証明書が追加された時点からアクセスを制限するには、トークン認証を使用するか、クライアント証明書をサーバーに直接追加する必要があります。パスワード認証を使用する場合、クライアント証明書が追加された後にのみ制限できます。

制限されたクライアント証明書を追加するには、次のコマンドを使用します:

トークン認証

```
lxc config trust add --projects <project_name> --restricted
```

クライアント証明書を追加

```
lxc config trust add <certificate_file> --projects <project_name> --restricted
```

クライアントは、通常の方法でサーバーをリモートに追加できます（`lxc remote add <server_name> <token>` または `lxc remote add <server_name> <server_address>`）。そして、指定されたプロジェクトのみにアクセスできます。

既存の証明書のアクセスを制限するには（アクセス制限が変更されるか、証明書が信頼パスワードで追加されたため）、次のコマンドを使用します：

```
lxc config trust edit <fingerprint>
```

`restricted` が `true` に設定されていることを確認し、`projects` の下に証明書がアクセスを許可するプロジェクトを指定してください。

注釈: リモートを追加するときに`--project` フラグを指定できます。この設定では、指定されたプロジェクトが事前に選択されます。ただし、これによってクライアントがこのプロジェクトに制限されるわけではありません。

特定の RBAC ロールにプロジェクトを制限する

Canonical RBAC サービスを使用している場合、RBAC ロールはそのロールを持つユーザーが実行できる操作を定義します。詳細については、**役割ベースのアクセスコントロール** *(RBAC)* を参照してください。

RBAC を使用してプロジェクトを制限するには、RBAC インターフェイスで対象のプロジェクトに移動し、必要に応じて異なるユーザーやグループに RBAC ロールを割り当てます。

特定の LXD ユーザーにプロジェクトを制限する

LXD snap を使用する場合、snap に含まれるマルチユーザー LXD デーモンを設定して、特定のユーザーグループ内のすべてのユーザーのために動的にプロジェクトを作成できます。

そのためには、`daemon.user.group` 設定オプションを対応するユーザーグループに設定します：

```
sudo snap set lxd daemon.user.group=<user_group>
```

LXD を使用できるようにしたいすべてのユーザーアカウントがこのグループのメンバーであることを確認してください。

グループのメンバーが LXD コマンドを発行すると、LXD はこのユーザーのために制限されたプロジェクトを作成し、このプロジェクトに切り替えます。この時点で LXD が**初期化**されていない場合、自動的に初期化されます（デフォルト設定で）。

プロジェクトの設定をカスタマイズしたい場合（例えば、制限や制約を課すために）、プロジェクトが作成された後で行うことができます。プロジェクト設定を変更するには、LXD への完全なアクセスが必要です。つまり、設定した LXD ユーザグループの一員であるだけでなく、lxd グループの一員である必要があります。

3.8.5 プロジェクトの設定

プロジェクトは、キー/値の設定オプションのセットを通じて設定することができます。これらのオプションを設定する方法については、**プロジェクトの設定** を参照してください。

キー/値の設定は名前空間化されています。次のオプションが利用可能です。

- **プロジェクトの機能**
- **プロジェクトの制限**

- プロジェクトの制約
- プロジェクト固有の設定

プロジェクトの機能

プロジェクトの機能は、プロジェクト内でどのエンティティが隔離され、どのエンティティが default プロジェクトから継承されるかを定義します。

feature.* オプションが true に設定されている場合、対応するエンティティはプロジェクト内で隔離されます。

注釈: 特定のオプションを明示的に設定せずにプロジェクトを作成すると、このオプションは以下の表で与えられた初期値に設定されます。

ただし、feature.* オプションのいずれかを解除すると、初期値に戻るのではなく、デフォルト値に戻ります。すべての feature.* オプションのデフォルト値は false です。

キー	タイプ	デフォルト	初期値	説明
features.images	bool	false	true	プロジェクト用に独立したイメージとイメージエイリアスのセットを使用するかどうか
features.networks	bool	false	false	プロジェクト用に独立したネットワークのセットを使用するかどうか
features.networks.zones	bool	false	false	プロジェクト用に独立したネットワークゾーンのセットを使用するかどうか
features.profiles	bool	false	true	プロジェクト用に独立したプロファイルのセットを使用するかどうか
features.storage.buckets	bool	false	true	プロジェクト用に独立したストレージバケットのセットを使用するかどうか
features.storage.volumes	bool	false	true	プロジェクト用に独立したストレージボリュームのセットを使用するかどうか

プロジェクトの制限

プロジェクトの制限は、プロジェクトに属するコンテナや VM が使用できるリソースの上限を定義します。

`limits.*` オプションによっては、プロジェクト内で許可されるエンティティの数に制限が適用されることがあります（例： `limits.containers` や `limits.networks`）。また、プロジェクト内のすべてのインスタンスのリソース使用量の合計値に制限が適用されることもあります（例： `limits.cpu` や `limits.processes`）。後者の場合、制限は通常、各インスタンスに設定されている **リソース制限** に適用されます（直接またはプロファイル経由で設定されている場合）、実際に使用されているリソースではありません。

例えば、プロジェクトの `limits.memory` 設定を `50GB` に設定した場合、プロジェクトのインスタンスで定義されたすべての `limits.memory` 設定キーの個別の値の合計が 50GB 未満に保たれます。`limits.memory` 設定の合計が 50GB を超えるインスタンスを作成しようとすると、エラーが発生します。

同様に、プロジェクトの `limits.cpu` 設定キーを `100` に設定すると、個々の `limits.cpu` 値の合計が 100 未満に保たれます。

プロジェクトの制限を使用する場合、以下の条件を満たす必要があります。

- `limits.*` 設定のいずれかを設定し、インスタンスに対応する設定がある場合、プロジェクト内のすべてのインスタンスに対応する設定が定義されている必要があります（直接またはプロファイル経由で設定）。インスタンスの設定オプションについては **リソース制限** を参照してください。
- *CPU* **ピンニング** が有効になっている場合、`limits.cpu` 設定は使用できません。これは、プロジェクトで `limits.cpu` を使用するためには、プロジェクト内の各インスタンスの `limits.cpu` 設定を CPU の数、または CPU のセットや範囲ではなく、数値に設定する必要があることを意味します。
- `limits.memory` 設定は、パーセンテージではなく絶対値で設定する必要があります。

キー	タイプ	デフォルト	説明
limits.containers	integer	-	プロジェクトで作成できるコンテナの最大数
limits.cpu	integer	-	プロジェクトのインスタンスで設定された個々の limits.cpu 設定の合計の最大値
limits.disk	string	-	プロジェクトのすべてのインスタンスボリューム、カスタムボリューム、およびイメージが使用するディスク容量の合計の最大値
limits.instances	integer	-	プロジェクトで作成できるインスタンスの合計数の最大値
limits.memory	string	-	プロジェクトのインスタンスで設定された個々の limits.memory 設定の合計の最大値
limits.networks	integer	-	プロジェクトが持つことのできるネットワークの最大数
limits.processes	integer	-	プロジェクトのインスタンスで設定された個々の limits.processes 設定の合計の最大値
limits.virtual-machine	integer	-	プロジェクトで作成できる VM の最大数

プロジェクトの制約

プロジェクトのインスタンスがセキュリティに関連する機能（コンテナのネストや raw LXC 設定など）にアクセスできないようにするには、restricted 設定オプションを true に設定します。その後、さまざまな restricted.* オプションを使用して、通常は restricted によってブロックされる個々の機能を選択し、プロジェクトのインスタンスで使用できるように許可できます。

例えば、プロジェクトを制限し、すべてのセキュリティ関連機能をブロックしつつ、コンテナのネストを許可するには、次のコマンドを入力します:

```
lxc project set <project_name> restricted=true
lxc project set <project_name> restricted.containers.nesting=allow
```

セキュリティに関連する各機能には、関連する restricted.* プロジェクト設定オプションがあります。機能の

使用を許可する場合は、その restricted.* オプションの値を変更してください。ほとんどの restricted.* 設定は、block（デフォルト）または allow に設定できる二値スイッチです。ただし、一部のオプションは、より細かい制御のために他の値をサポートしています。

注釈: restricted.* オプションを有効にするには、restricted 設定を true に設定する必要があります。restricted が false に設定されている場合、restricted.* オプションを変更しても効果はありません。

すべての restricted.* キーを allow に設定することは、restricted 自体を false に設定することと同等です。

キー	タイプ	デフォルト	説明
restricted	bool	false	セキュリティに敏感な機能へのアクセスをブロックするかどうか - restricted.* キーが有効になるためには、この設定を有効にする必要があります（必要に応じて一時的に無効にできるように、関連するキーをクリアせずに有効にする）
restricted.backups	string	block	インスタンスやボリュームのバックアップを作成できないようにする
restricted.cluster.groups	string	-	与えられたもの以外のクラスタグループをターゲットにすることを防ぐ
restricted.cluster.target	string	block	インスタンスの作成や移動時にクラスタメンバーを直接ターゲットにすることを防ぐ
restricted.containers.lowlevel	string	block	raw.lxc、raw.idmap、volatile などの低レベルなコンテナオプションを使用できないようにする
restricted.containers.nesting	string	block	security.nesting=true を設定できないようにする
restricted.containers.privilege	string	unpri	特権コンテナの設定を制限する（unprivileged は security.privileged=true を設定できないようにする、isolated は security.privileged=true と security.idmap.isolated=true の設定を制限する、allow は制限がない）
restricted.containers.interception	string	block	システムコールインターセプトオプションの使用を制限する - allow に設定されている場合、通常は安全なインターセプトオプションが許可される（ファイルシステムのマウントはブロックされたまま）
restricted.devices.disk	string	manag	ディスクデバイスの使用を制限する（block はルートデバイス以外のディスクデバイスの使用を制限する、managed は pool= が設定されている場合にのみディスクデバイスを使用できるようにする、allow は制限がない）
restricted.devices.disk.paths	string	-	restricted.devices.disk が allow に設定されている場合：disk デバイスの source 設定に制限を加える、カンマ区切りのパスプレフィックスのリスト（空の場合、すべてのパスが許可されます）
restricted.devices.gpu	string	block	gpu タイプのデバイスの使用を制限する
restricted.devices.infiniband	string	block	infiniband タイプのデバイスの使用を制限する
restricted.devices.nic	string	manag	ネットワークデバイスの使用を制限し、ネットワークへのアクセスを制御する（block はすべてのネットワークデバイスの使用を制限する、managed は network= が設定されている場合にのみネットワークデバイスを使用できるようにする、allow は制限がない）
restricted.	string	block	pci タイプのデバイスの使用を制限する

プロジェクト固有の設定

プロジェクトに対していくつかの **サーバー設定** オプションを上書きできます。また、プロジェクトにユーザーメタデータを追加することができます。

キー	タイプ	デフォルト	説明
backups.compression_algorith	string	-	プロジェクト内のバックアップに使用する圧縮アルゴリズム（bzip2、gzip、lzma、xz、または none）
images.auto_update_cached	bool	-	LXD がキャッシュするイメージを自動的に更新するかどうか
images.auto_update_interval	integer	-	キャッシュされたイメージの更新を検索する間隔（時間単位）（無効にするには 0）
images.compression_algorith	string	-	プロジェクト内の新しいイメージに使用する圧縮アルゴリズム（bzip2、gzip、lzma、xz、または none）
images.default_architecture	string	-	混在アーキテクチャクラスタで使用するデフォルトアーキテクチャ
images.remote_cache_expiry	integer	-	プロジェクト内で未使用のキャッシュされたリモートイメージが削除されるまでの日数
user.*	string	-	ユーザーが提供する自由形式のキー/値ペア

3.9 クラスタリング

3.9.1 クラスタリングについて

全体のワークロードを複数のサーバーに分散するため、 LXD はクラスタリングモードで動かせます。このシナリオでは、クラスタメンバーとそのインスタンスの設定を保持する同じ分散データベースを任意の台数の LXD サーバーで共有します。LXD クラスタは lxc クライアントまたは REST API を使って管理できます。

この機能は *clustering* API 拡張の一部として導入され、 LXD 3.0 以降で利用可能です。

Tip: ベーシックな LXD クラスタを素早くセットアップしたい場合、MicroCloud をチェックしてみてください。

クラスタメンバー

LXD クラスタは 1 台のブートストラップサーバーと少なくともさらに 2 台のクラスタメンバーから構成されます。クラスタは状態を **分散データベース** に保管します。これは Raft アルゴリズムを使用して複製される Dqlite データベースです。

2 台のメンバーだけでもクラスタを作成することは出来なくはないですが、少なくとも 3 台のクラスタメンバーを強く推奨します。このセットアップでは、クラスタは少なくとも 1 台のメンバーの消失に耐えることができ、分散状態の過半数を確立できます。

クラスタを作成する際、Dqlite データベースは 3 番目のメンバーがクラスタにジョインするまではブートストラップサーバー上でのみ稼働します。そして 2 番目と 3 番目のサーバーはデータベースの複製を受信します。

詳細は **クラスタを形成するには** を参照してください。

メンバーロール

3 台のメンバーのクラスタでは、全てのメンバーがクラスタの状態を保管する分散データベースを複製します。クラスタのメンバーがさらに増えると、一部のメンバーだけがデータベースを複製します。残りのメンバーはデータベースへアクセスしますが、複製はしません。

任意の時点で、選出されたリーダーが 1 つ存在し、他のメンバーの健康状態をモニターします。

データベースを複製する各メンバーは *voter* か *stand-by* のロールを持ちます。クラスタリーダーがオフラインになると voter の 1 つが新しいリーダーに選出されます。voter のメンバーがオフラインになると stand-by メンバーが自動的に voter に昇格します。データベース (そしてクラスタ) は voter の過半数がオンラインである限り利用可能です。

以下のロールが LXD クラスタメンバーに割り当て可能です。自動のロールは LXD 自身によって割り当てられユーザによる変更は出来ません。

ロール	自動	説明
`database`	yes	分散データベースの voter メンバー
`database-leader`	yes	分散データベースの現在のリーダー
`database-standby`	yes	分散データベースの stand-by (voter ではない) メンバー
`event-hub`	no	内部 LXD イベントへの交換ポイント (hub) (最低 2 つは必要)
`ovn-chassis`	no	OVN ネットワークのアップリンクゲートウェイの候補

voter メンバーのデフォルトの数 (*`cluster.max_voters`*) は 3 です。stand-by メンバーのデフォルトの数 (*`cluster.max_standby`*) は 2 です。この設定では、クラスタを稼働したまま一度に最大で 1 つの voter メンバーの電源を切ることができます。

詳細は **クラスタを管理するには** を参照してください。

オフラインメンバーと障害耐性

クラスタメンバーがダウンして設定されたオフラインの閾値を超えると、ステータスはオフラインと記録されます。この場合、このメンバーに対する操作はできなくなり、全てのメンバーの状態変更を必要とする操作もできなくなります。

オフラインのメンバーがオンラインに戻るとすぐに操作が再びできるようになります。

オフラインになったメンバーがリーダーそのものだった場合、他のメンバーは新しいリーダーを選出します。

サーバーを再びオンラインに復旧できないあるいはしたくない場合、**クラスタからメンバーを削除** できます。

応答しないメンバーをオフラインと判断する秒数は *cluster.offline_threshold* 設定で調整できます。デフォルト値は 20 秒です。最小値は 10 秒です。

オフラインのメンバーからインスタンスを自動的に**退避**するには、*cluster.healing_threshold* 設定をゼロでない値に設定してください。

詳細は**クラスタを復旧するには**を参照してください。

failure domain

オフラインになったメンバーにロールを割り当てる際に、どのクラスタメンバーを優先するかを指示するために failure domain を使用できます。例えば、現在データベースロールを持つクラスタメンバーがシャットダウンした場合、 LXD はデータベースロールを同じ failure domain 内の別のクラスタメンバーがあればそれに割り当てようとします。

クラスタメンバーの failure domain を更新するには、 `lxc cluster edit <member>` コマンドを使って `failure_domain` プロパティを `default` から他の文字列に変更します。

メンバー設定

LXD クラスタメンバーは一般的に同一のシステムと想定されています。それはクラスタにジョインする全ての LXD サーバーはブートストラップサーバーとストレージプールとネットワークについて同一の設定を持つ必要があるということです。

少し異なるディスクの順序やネットワークインタフェースの名前付けのようなことに対応するため、ストレージとネットワークに関連してメンバー固有のいくつかの設定が例外的に用意されています。

クラスタ内にそのような設定が存在する場合、追加するサーバーにはそれらの設定に対する値を提供する必要があります。たいていの場合、これはインタラクティブな `lxd init` コマンドで実行され、ユーザにストレージやネットワークに関連する設定の値の入力を求めます。

通常これらの設定には以下のものが含まれます。

- ストレージプールのソースデバイスとサイズ

- ZFS プール、 LVM thin pool、または LVM ボリュームグループの名前
- ブリッジネットワークの外部インタフェースと BGP の next-hop
- 管理された `physical` または `macvlan` ネットワークの親のネットワークデバイス名

詳細は **クラスタのストレージを設定するには** と **クラスタのネットワークを設定するには** を参照してください。

事前に質問を調べたい (スクリプトでの自動化に有用) 場合、 `/1.0/cluster` API エンドポイントをクエリしてください。これは `lxc query /1.0/cluster` あるいは他の API クライアントを使って実行できます。

イメージ

デフォルトでは、 LXD はデータベースメンバーと同じ数のクラスタメンバーにイメージを複製します。通常これはクラスタ内で最大 3 つのコピーを持つことを意味します。

障害耐性とイメージがローカルで利用できる確率を改善するためこの数を増やすことができます。そのためには、*`cluster.images_minimal_replica`* 設定を変更してください。すべてのクラスタメンバーにイメージをコピーするには`-1` という特別な値を使用できます。

クラスタグループ

LXD のクラスタではクラスタグループにメンバーを追加できます。これらのクラスタグループは、全ての利用可能なメンバーのサブセットに属するクラスタメンバー上で、インスタンスを起動するのに使用できます。例えば、GPU を持つ全てのメンバーからなるクラスタメンバーを作って、GPU が必要な全てのインスタンスをこのクラスタグループ上で起動できます。

デフォルトでは、全てのクラスタメンバーは `default` グループに属します。

詳細は **クラスタグループをセットアップするには** と **特定のクラスタメンバー上でインスタンスを起動する** を参照してください。

インスタンスの自動配置

クラスタのセットアップでは各インスタンスはクラスタメンバーの 1 つの上で稼働します。インスタンスを起動する際、特定のクラスタメンバー、クラスタグループをターゲットにするか、あるいは LXD に自動的にどれかのクラスタメンバーに割り当てさせることもできます。

デフォルトでは、自動的な割り当てはインスタンス数が一番少ないクラスタメンバーを選択します。複数のメンバーが同じインスタンス数の場合は、それらの 1 つがランダムで選ばれます。

しかし、この挙動を *`scheduler.instance`* 設定で制御することもできます。

- クラスタメンバーの `scheduler.instance` が `all` に設定されると、以下の条件でこのクラスタメンバーが選ばれます。

  - インスタンスが `--target` を指定せずに作成され、かつクラスタメンバーのインスタンス数が最小である。
  - インスタンスがこのクラスタメンバー上で稼働するようにターゲットされた。
  - インスタンスがこのクラスタメンバーが所属するクラスタグループのメンバー上で稼働するようにターゲットされ、かつクラスタメンバーがそのクラスタグループの他のメンバーと比べてインスタンス数が最小である。

- クラスタメンバーの `scheduler.instance` が `manual` に設定されると、以下の条件でこのクラスタメンバーが選ばれます。
  - インスタンスがこのクラスタメンバー上で稼働するようにターゲットされた。

- クラスタメンバーの `scheduler.instance` が `group` に設定されると、以下の条件でこのクラスタメンバーが選ばれます。
  - インスタンスがこのクラスタメンバー上で稼働するようにターゲットされた。
  - インスタンスがこのクラスタメンバーが所属するクラスタグループのメンバー上で稼働するようにターゲットされ、かつクラスタメンバーがそのクラスタグループの他のメンバーと比べてインスタンス数が最小である。

インスタンス配置スクリプトレット

LXD では埋め込まれたスクリプト (スクリプトレット) を使って自動的なインスタンス配置を制御するカスタムロジックを使用できます。この方法は、組み込みのインスタンス配置機能よりも柔軟性が高いです。

インスタンス配置スクリプトレットは Starlark 言語 (Python のサブセット) で記述する必要があります。スクリプトレットは、LXD がインスタンスをどこに配置するかを知る必要があるたびに呼び出されます。スクリプトレットは、配置されるインスタンスに関する情報と、インスタンスをホストできる候補のクラスタメンバーに関する情報を受け取ります。スクリプトレットからクラスタメンバー候補の状態と利用可能なハードウェアリソースについての情報を要求することもできます。

インスタンス配置スクリプトレットは `instance_placement` 関数を以下のシグネチャで実装する必要があります。

`instance_placement(request, candidate_members)`:

- `request` は、`scriptlet.InstancePlacement` の展開された表現を含むオブジェクトです。このリクエストには、`project` および `reason` フィールドが含まれています。`reason` は、`new`、`evacuation`、または `relocation` のいずれかです。
- `candidate_members` は、`api.ClusterMember` エントリを表すクラスタメンバーオブジェクトの `list` です。

例:

```
def instance_placement(request, candidate_members):
    # 情報ログ出力の例。これは LXD のログに出力されます。
    log_info("instance placement started: ", request)

    # インスタンスのリクエストに基づいてロジックを適用する例。
    if request.name == "foo":
        # エラーログ出力の例。これは LXD のログに出力されます。
        log_error("Invalid name supplied: ", request.name)

        fail("Invalid name") # エラーで終了してインスタンス配置を拒否します。

    # 提供された第 1 候補のサーバーにインスタンスを配置する。
    set_target(candidate_members[0].server_name)

    return # インスタンス配置を進めるために空を返す。
```

スクリプトレットは LXD に適用するためには instances.placement.scriptlet グローバル設定に設定する必要があります。

例えばスクリプトレットが instance_placement.star というファイルに保存されている場合、LXD には以下のように適用できます。

```
cat instance_placement.star | lxc config set instances.placement.scriptlet=-
```

LXD に現在適用されているスクリプトレットを見るには lxc config get instances.placement.scriptlet コマンドを使用してください。

スクリプトレットでは (Starlark で提供される関数に加えて) 以下の関数が利用できます。

- log_info(*messages): info レベルで LXD のログにログエントリを追加します。messages は 1 つ以上のメッセージの引数です。
- log_warn(*messages): warn レベルで LXD のログにログエントリを追加します。messages は 1 つ以上のメッセージの引数です。
- log_error(*messages): error レベルで LXD のログにログエントリを追加します。messages は 1 つ以上のメッセージの引数です。
- set_cluster_member_target(member_name): インスタンスが作成されるべきクラスタメンバーを設定します。member_name はインスタンスが作成されるべきクラスタメンバーの名前です。この関数が呼ばれなければ、LXD は組み込みのインスタンス配置ロジックを使用します。
- get_cluster_member_state(member_name): クラスタメンバーの状態を取得します。api.ClusterMemberState の形式でクラスタメンバーの状態を含むオブジェクトを返します。member_name は

状態を取得する対象のクラスタメンバーの名前です。

- `get_cluster_member_resources(member_name)`: クラスタメンバーのリソースについての情報を取得します。`api.Resources` の形式でリソースについての情報を含むオブジェクトを返します。`member_name` はリソース情報を取得する対象のクラスタメンバーの名前です。
- `get_instance_resources()`: インスタンスが必要とするリソースについての情報を取得します。`scriptlet.InstanceResources` の形式でリソース情報を含むオブジェクトを返します。

注釈: オブジェクト内のフィールド名は対応する Go の型の JSON フィールド名と同じです。

3.9.2 クラスタを形成するには

LXD クラスタを形成するときはブートストラップサーバーから始めます。このブートストラップサーバーは既存の LXD サーバーでもよいですし新しくインストールしたものでもよいです。

ブートストラップサーバーを初期化した後、クラスタに追加のサーバーをジョインできます。詳細は **クラスタメンバー** を参照してください。

LXD クラスタを形成するために初期化プロセス中に設定をインタラクティブに指定することもできますし、完全な設定を含むプリシードファイルを使うこともできます。

素早く自動的にベーシックな LXD クラスタをセットアップするには MicroCloud が使えます。ただし、このプロジェクトはまだ初期段階なことに注意してください。

クラスタをインタラクティブに設定する

クラスタを形成するには、まずブートストラップサーバー上で `lxd init` を実行する必要があります。その後クラスタにジョインさせたい他のサーバー上でもそのコマンドを実行します。

クラスタをインタラクティブに形成する際、クラスタを設定するために `lxd init` のプロンプトの質問に回答します。

ブートストラップサーバーを初期化する

ブートストラップサーバーを初期化するには、 `lxd init` を実行して希望の設定に応じて質問に回答します。

ほとんどの質問はデフォルト値を受け入れることができますが、以下の質問には適切に答えるようにしてください。

- `Would you like to use LXD clustering?`

  yes を選択。

- What IP address or DNS name should be used to reach this server?

  他のサーバーがアクセスできる IP または DNS のアドレスを確実に使用してください。

- Are you joining an existing cluster?

  no を選択。

- Setup password authentication on the cluster?

  認証トークン (推奨) を使う場合 **no** を、**トラストパスワード** を使う場合 **yes** を選択。

```
user@host:~$ lxd init         Would you like to use LXD clustering? (yes/no) [default=no]:
yesWhat IP address or DNS name should be used to reach this server? [default=192.0.2.
101]:Are you joining an existing cluster? (yes/no) [default=no]:  noWhat member name
should be used to identify this server in the cluster? [default=server1]:Setup password
authentication on the cluster? (yes/no) [default=no]:  noDo you want to configure a
new local storage pool? (yes/no) [default=yes]:Name of the storage backend to use
(btrfs, dir, lvm, zfs) [default=zfs]:Create a new ZFS pool? (yes/no) [default=yes]:Would
you like to use an existing empty block device (e.g. a disk or partition)? (yes/no)
[default=no]:Size in GiB of the new loop device (1GiB minimum) [default=9GiB]:Do you
want to configure a new remote storage pool? (yes/no) [default=no]:Would you like to
connect to a MAAS server? (yes/no) [default=no]:Would you like to configure LXD to use
an existing bridge or host interface? (yes/no) [default=no]:Would you like to create a
new Fan overlay network? (yes/no) [default=yes]:What subnet should be used as the Fan
underlay? [default=auto]:Would you like stale cached images to be updated automatically?
(yes/no) [default=yes]:Would you like a YAML "lxd init" preseed to be printed? (yes/no)
[default=no]:
```

初期化プロセスが終了したら、最初のクラスタメンバーが起動してネットワーク上で利用可能になるはずです。これは `lxc cluster list` で確認できます。

追加のサーバーをジョインさせる

これでクラスタに追加のサーバーをジョインできるようになりました。

注釈: 追加するサーバーは新規にインストールした LXD サーバーにしたほうがよいです。既存のサーバーを使う場合、既存のデータは消失するので、ジョインする前にデータを確実にクリアしてください。

クラスタにサーバーをジョインさせるには、クラスタ上で `lxd init` を実行します。既存のクラスタにジョインするには root 権限が必要ですので、コマンドを root で実行するか sudo をつけて実行するのを忘れないでください。

基本的に、初期化プロセスは以下のステップからなります。

1. 既存のクラスタにジョインをリクエストする。

   `lxd init` の最初の質問に適切に回答します。

   - `Would you like to use LXD clustering?`

     yes を選択。

   - `What IP address or DNS name should be used to reach this server?`

     他のサーバーがアクセスできる IP または DNS のアドレスを確実に使用してください。

   - `Are you joining an existing cluster?`

     yes を選択。

   - `Do you have a join token?`

     ブートストラップサーバーを **認証トークン** (推奨) を使うように設定した場合 **yes** を、**トラストパスワード** を使うように設定した場合 **no** を選択。

2. クラスタで認証する。

   ブートスラップサーバーを設定する際に選んだ認証方法に応じて 2 つの方法があります。

   認証トークン (推奨)

   認証トークン を使うようにクラスタを設定した場合、新メンバーごとにジョイントークンを生成する必要があります。そのためには、既存のクラスタメンバー (例えばブートストラップサーバー) で以下のコマンドを実行します。

   ```
   lxc cluster add <new_member_name>
   ```

   このコマンドは設定時に有効な (*cluster.join_token_expiry* 参照) 一回限りのジョイントークンを返します。`lxd init` のプロンプトでジョイントークンを求められたときにこのトークンを入力してください。

   ジョイントークンは既存のオンラインメンバーのアドレス、一回限りのシークレットとクラスタ証明書のフィンガープリントを含みます。ジョイントークンがこれらの質問に自動で回答できるので、`lxd init` 中に回答が必要な質問の量を減らすことができます。

   トラストパスワード

   トラストパスワード を使うようにクラスタを設定した場合、認証プロセスを開始できるまでに `lxd init` はより多くの情報を必要とします。

   1. 新しいクラスタメンバーの名前を指定します。
   2. 既存のクラスタメンバーのアドレスを提供します (ブートストラップサーバーまたはすでに追加済みの他のサーバー)。

3. クラスタのフィンガープリントを検証します。
4. フィンガープリントが正しければ、クラスタで認証するトラストパスワードを入力します。

3. クラスタにジョインする際サーバーの全てのローカルデータが消失することを確認します。
4. サーバー固有の設定を行います (詳細は **メンバー設定** を参照)。

   デフォルト値を受け入れることもできますし、各サーバーにカスタム値を指定することもできます。

認証トークン (推奨)

```
user@host:~$ sudo lxd init    Would you like to use LXD clustering? (yes/no) [default=no]:
yesWhat IP address or DNS name should be used to reach this server? [default=192.
0.2.102]:Are you joining an existing cluster? (yes/no) [default=no]:  yesDo you
have a join token? (yes/no/[token]) [default=no]:  yesPlease provide join token:
eyJzZXJ2ZXJfbmFtZSI6InJwaTAxIiwiZmluZ2VycHJpbnQiOiIyNjZjZmExZDk0ZDZiMjk2Nzk0YjU0YzJlYzdjOTMwNDA5ZjIzNjdr
existing data is lost when joining a cluster, continue? (yes/no) [default=no] yesChoose
"size" property for storage pool "local":Choose "source" property for storage pool
"local":Choose "zfs.pool_name" property for storage pool "local":Would you like a YAML
"lxd init" preseed to be printed? (yes/no) [default=no]: トラストパスワード
```

```
user@host:~$ sudo lxd init    Would you like to use LXD clustering? (yes/no) [default=no]:
yesWhat IP address or DNS name should be used to reach this server? [default=192.0.
2.102]:Are you joining an existing cluster? (yes/no) [default=no]:  yesDo you have
a join token? (yes/no/[token]) [default=no]:  noWhat member name should be used to
identify this server in the cluster? [default=server2]:IP address or FQDN of an
existing cluster member (may include port):  192.0.2.101:8443Cluster fingerprint:
2915dafdf5c159681a9086f732644fb70680533b0fb9005b8c6e9bca51533113You can validate this
fingerprint by running "lxc info" locally on an existing cluster member.Is this the
correct fingerprint? (yes/no/[fingerprint]) [default=no]:  yesCluster trust password:All
existing data is lost when joining a cluster, continue? (yes/no) [default=no] yesChoose
"size" property for storage pool "local":Choose "source" property for storage pool
"local":Choose "zfs.pool_name" property for storage pool "local":Would you like a YAML
"lxd init" preseed to be printed? (yes/no) [default=no]:
```

初期化プロセスが終わった後、サーバーが新しいクラスタメンバーとして追加されます。これは `lxc cluster list` で確認できます。

クラスタをプリシードファイルで設定する

クラスタを形成するには、まずブートストラップサーバー上で `lxd init` を実行します。その後、クラスタにジョインさせたい他のサーバーでもこのコマンドを実行します。

`lxd init` の質問にインタラクティブに回答する代わりに、プリシードファイルを使って必要な情報を提供できます。以下のコマンドを使って `lxd init` にファイルをフィードできます。

```
cat <preseed-file> | lxd init --preseed
```

サーバーごとに異なるプリシードファイルが必要です。

ブートストラップサーバーを初期化する

プリシードファイルの必要な中身は認証に **認証トークン** (推奨) を使うか **トラストパスワード** を使うかに応じて異なります。

認証トークン (推奨)

クラスタリングを有効にするには、ブートストラップサーバー用のプリシードファイルは以下のフィールドを含む必要があります。

```
config:
  core.https_address: <IP_address_and_port>
cluster:
  server_name: <server_name>
  enabled: true
```

ブートストラップサーバー用のプリシードファイルの例を以下に示します。

```
config:
  core.https_address: 192.0.2.101:8443
  images.auto_update_interval: 15
storage_pools:
- name: default
  driver: dir
networks:
- name: lxdbr0
  type: bridge
profiles:
- name: default
  devices:
    root:
```

(次のページに続く)

(前のページからの続き)

```
      path: /
      pool: default
      type: disk
    eth0:
      name: eth0
      nictype: bridged
      parent: lxdbr0
      type: nic
cluster:
  server_name: server1
  enabled: true
```

トラストパスワード

クラスタリングを有効にするには、ブートストラップサーバー用のプリシードファイルは以下のフィールドを含む必要があります。

```
config:
  core.https_address: <IP_address_and_port>
  core.trust_password: <trust_password>
cluster:
  server_name: <server_name>
  enabled: true
```

ブートストラップサーバー用のプリシードファイルの例を以下に示します。

```
config:
  core.trust_password: the_password
  core.https_address: 192.0.2.101:8443
  images.auto_update_interval: 15
storage_pools:
- name: default
  driver: dir
networks:
- name: lxdbr0
  type: bridge
profiles:
- name: default
  devices:
    root:
```

(次のページに続く)

(前のページからの続き)

```
      path: /
      pool: default
      type: disk
    eth0:
      name: eth0
      nictype: bridged
      parent: lxdbr0
      type: nic
cluster:
  server_name: server1
  enabled: true
```

追加のサーバーをジョインさせる

プリシードファイルの必要な中身は認証に **認証トークン** (推奨) を使うか **トラストパスワード** を使うかに応じて異なります。

新しいクラスタメンバー用のプリシードファイルは参加するサーバーに固有のデータと設定値を含む `cluster` セクションのみが必要です。

認証トークン (推奨)

追加のサーバーのプリシードファイルは以下の項目を含む必要があります。

```
cluster:
  enabled: true
  server_address: <IP_address_of_server>
  cluster_token: <join_token>
```

新しいクラスタメンバー用のプリシードファイルの例を以下に示します。

```
cluster:
  enabled: true
  server_address: 192.0.2.102:8443
  cluster_token:␣
↪eyJzZXJ2ZXJfbmFtZSI6Im5vZGUyIiwiZmluZ2VycHJpbnQiOiJjZjlmNmVhMWIzYjhiNjgxNzQ1YTY1NTY2YjM3ZGUwOTUzNjRmM2
  member_config:
  - entity: storage-pool
    name: default
    key: source
    value: ""
```

トラストパスワード

追加のサーバーのプリシードファイルは以下の項目を含む必要があります。

```
cluster:
  server_name: <server_name>
  enabled: true
  cluster_address: <IP_address_of_bootstrap_server>
  server_address: <IP_address_of_server>
  cluster_password: <trust_password>
  cluster_certificate: <certificate> # これか cluster_certificate_path を使用します
  cluster_certificate_path: <path_to-certificate_file> # これか cluster_certificate を使用
します
```

YAML 互換の cluster_certificate キーを作成するにはブートストラップサーバー上で以下のどちらかのコマンドを実行してください。

- snap を使用している場合: sed ':a;N;$!ba;s/\n/\n\n/g' /var/snap/lxd/common/lxd/cluster.crt
- そうでない場合: sed ':a;N;$!ba;s/\n/\n\n/g' /var/lib/lxd/cluster.crt

あるいは、cluster.crt ファイルをブートストラップサーバーからジョインさせたいサーバーにコピーして cluster_certificate_path キーにそのパスを指定します。

新しいクラスタメンバー用のプリシードファイルの例を以下に示します。

```
cluster:
  server_name: server2
  enabled: true
  server_address: 192.0.2.102:8443
  cluster_address: 192.0.2.101:8443
  cluster_certificate: "-----BEGIN CERTIFICATE-----

opyQ1VRpAg2sV2C4W8irbNqeUsTeZZxhLqp4vNOXXBBrSqUCdPu1JXADV0kavg1l

2sXYoMobyV3K+RaJgsr1OiHjacGiGCQT3YyNGGY/n5zgT/8xI0Dquvja0bNkaf6f

...

-----END CERTIFICATE-----
"
  cluster_password: the_password
```

(次のページに続く)

(前のページからの続き)

```
member_config:
- entity: storage-pool
  name: default
  key: source
  value: ""
```

MicroCloud を使う

LXD クラスタを手動でセットアップする代わりに、MicroCloud を使ってすぐに使える LXD クラスタと Ceph ストレージの環境を作ることができます。

これに必要な snap パッケージをインストールするには、以下のコマンドを実行します。

```
snap install lxd microceph microcloud
```

次に以下のコマンドでブートストラッププロセスを開始します。

```
microcloud init
```

初期化の行程中に、MicroCloud は他のサーバーを検出、クラスタをセットアップし、Ceph に追加するディスクを尋ねるプロンプトを表示します。

初期化が完了したら、Ceph と LXD クラスタの両方が作られ、LXD 自体はネットワークとクラスタ内で使用するのに適したストレージが設定された状態になります。

3.9.3 クラスタを管理するには

クラスタを形成した後、メンバーの一覧と状態を見るには `lxc cluster list` を使用します。

```
user@host:~$ lxc cluster list+---------+----------------------------+------------------+--------------+
NAME | URL | ROLES | ARCHITECTURE | FAILURE DOMAIN | DESCRIPTION | STATE | MESSAGE
|+---------+----------------------------+------------------+--------------+---------------+-----------
server1 | https://192.0.2.101:8443 | database-leader | x86_64 | default | | ONLINE |
Fully operational || | | database | | | | |+---------+----------------------------+------------------
server2 | https://192.0.2.102:8443 | database-standby | aarch64 | default | | ONLINE |
Fully operational |+---------+----------------------------+------------------+-------------+----------
server3 | https://192.0.2.103:8443 | database-standby | aarch64 | default | | ONLINE |
Fully operational |+---------+----------------------------+------------------+-------------+----------
```

ここのクラスタメンバーについてより詳細な情報を見るには、以下のコマンドを実行します。

```
lxc cluster show <member_name>
```

クラスタメンバーの状態と使用状況を見るには、以下のコマンドを実行します。

```
lxc cluster info <member_name>
```

クラスタを設定するには

クラスタを設定するには、`lxc config` を使用します。例えば以下のようにします。

```
lxc config set cluster.max_voters 5
```

いくつかの **サーバー設定** はグローバルで他はローカルであることに注意してください。グローバル設定はどのクラスタメンバー上でも実行でき、変更は分散データベースを通して他のクラスタメンバーにも伝搬されます。ローカル設定は設定したサーバー上でのみ (あるいは `--target` で指定したサーバー上でのみ) 変更されます。

サーバー設定に加えて、各クラスタメンバーに固有ないくつかのクラスタ設定があります。利用可能な設定の全てについては **クラスタメンバーの設定** を参照してください。

これらの設定を変更するには、`lxc cluster set` か `lxc cluster edit` を使用します。例えば以下のようにします。

```
lxc cluster set server1 scheduler.instance manual
```

メンバーロールを割り当てる

クラスタメンバーに **メンバーロール** を追加または削除するには `lxc cluster role` コマンドを使用します。例えば以下のようにします。

```
lxc cluster role add server1 event-hub
```

注釈: LXD で自動で割り当てられないロールのみが追加または削除できます。

クラスタメンバー設定を編集する

メンバー固有設定、メンバーロール、failure domain とクラスタグループを含むクラスタメンバーのプロパティを編集するには `lxc cluster edit` コマンドを使用します。

クラスタメンバーの退避と復元

既存のクラスタメンバーの全てのインスタンスを空にしたい (例えば再起動が必要なシステムのアップデートを適用するような日常のメンテナンスやハードウェアの変更を行う場合など) ようなケースがあります。

このためには `lxc cluster evacuate` コマンドを使用します。このコマンドは指定したサーバー上の全てのインスタンスを他のクラスタメンバーに移動します。退避したクラスタメンバーは "evacuated" 状態になり、このメンバー上でインスタンスの作成はできなくなります。

各インスタンスがどのように移動するかは *cluster.evacuate* インスタンス設定で制御できます。インスタンスは `boot.host_shutdown_timeout` 設定にしたがってクリーンにシャットダウンされます。

退避したサーバーが再び利用可能になったときに、サーバーを通常の稼働状態に戻すには `lxc cluster restore` コマンドを使用します。このコマンドは退避したインスタンスを一時的に保持していたサーバーから戻します。

自動での退避

cluster.healing_threshold 設定をゼロでない値に設定すると、クラスタメンバーがオフラインになったら、インスタンスは自動的に退避されます。

退避されたサーバーが再び利用可能になったら、手動で復元する必要があります。

クラスタメンバーを削除する

クラスタからメンバーをクリーンに削除するには以下のコマンドを使用します。

```
lxc cluster remove <member_name>
```

オンラインでインスタンスが 1 つも存在しないメンバーだけがクリーンに削除できます。

オフラインのクラスタメンバーの強制削除

クラスタメンバーが恒久的にオフラインになった場合、クラスタメンバーをクラスタから強制削除できます。メンバーを復旧できないと気づいたらすぐに削除するようにしてください。クラスタにオフラインのメンバーを残しておくと、クラスタを新しいバージョンにアップグレードする際に問題が起きるかもしれません。

クラスタメンバーを強制削除するには、引き続きオンラインになっているクラスタメンバーの 1 つで以下のコマンドを入力します。

```
lxc cluster remove --force <member_name>
```

> **注意:** クラスタメンバーを強制削除するとメンバーのデータベースが不整合な状態 (例えば、メンバー上のストレージプールが削除されないなど) のままになります。結果として、後で LXD を再び初期化できなくなり、サーバーを完全に再インストールするしかなくなります。

クラスタメンバーをアップグレードする

クラスタをアップグレードするには、全てのメンバーをアップグレードする必要があります。全てのメンバーを同じ LXD のバージョンにアップグレードしなければなりません。

> **注意:** オフラインのメンバーがいる場合はクラスタをアップグレードしないでください。オフラインのメンバーはアップグレードできず、クラスタがブロックした状態になってしまいます。
>
> さらに snap をお使いの場合、アップグレードは自動で実行されますので、問題を避けるためには、オフラインメンバーを直ちに復旧するか削除するほうが良いです。

単一のメンバーをアップグレードするには、単にそのホスト上で LXD パッケージをアップグレードして LXD デーモンを再起動します。例えば、snap を使用していれば、LXD を現在のチャンネルの最新バージョンとコホートにリフレッシュ (さらに LXD をリロード) します。

```
sudo snap refresh lxd --cohort="+"
```

新しいバージョンのデーモンがデータベーススキーマまたは API に変更がある場合、アップグレードされたメンバーは "blocked" 状態に遷移するかも知れません。この場合、メンバーは LXD API リクエストに応答しなくなります (これはこのメンバー上では `lxc` コマンドがもはや使用きなくなることを意味します) が、稼働中のインスタンスは引き続き稼働します。

これはアップグレードされておらず古いバージョンを稼働している他のクラスタメンバーがある場合に発生します。ブロックされているメンバーがあるかを確認するには、ブロックされていないクラスタメンバー上で `lxc cluster list` を実行します。

残りのクラスタメンバーのアップグレードを進めると、それら全てのメンバーが "blocked" 状態になります。最後のメンバーをアップグレードすると、ブロックされたメンバーは全てのサーバーが最新になったことを検知し、ブロックされたメンバーが再び使用可能になります。

クラスタ証明書をアップグレードする

LXD クラスタ内で全てのサーバー上の API は同じ共有された証明書で応答します。これは通常有効期限が 10 年に設定されたごく普通の自己署名証明書です。

証明書は `/var/snap/lxd/common/lxd/cluster.crt` (snap を使用している場合) か `/var/lib/lxd/cluster.crt` (それ以外の場合) に保管され、全てのクラスタメンバー上で同じです。

この自己署名証明書を別の証明書、例えば、 ACME サービス経由で得られた有効な証明書 (詳細は *TLS* **サーバー証明書** を参照) に置き換えることができます。そのためには `lxc cluster update-certificate` コマンドを使用します。このコマンドはクラスタ内の全てのサーバーの証明書を置き換えます。

3.9.4 クラスタを復旧するには

クラスタの 1 つまたは複数のメンバーがオフラインまたは到達不能になるかもしれません。この場合、このメンバー上での操作と全てのメンバーにまたがる状態変更が必要な操作は不可能になります。詳細は**オフラインメンバーと障害耐性**と**自動での退避**を参照してください。

オフラインのクラスタメンバーを復旧させるかクラスタから削除すると、通常通り操作が可能となります。これができない場合、故障の原因となったケースに応じて、クラスタを復旧させるいくつかの方法があります。詳細は以下のセクションを参照してください。

注釈: 復旧が必要な状態のクラスタにいる場合、LXD クライアントが LXD デーモンに接続できないため、ほとんどの `lxc` コマンドは動きません。

このため、クラスタを復旧するコマンドは LXD デーモン (`lxd`) が直接提供します。全ての利用可能なコマンドの概要を見るには `lxd cluster --help` を実行してください。

過半数割れからの復旧

各 LXD クラスタは (*`cluster.max_voters`* で設定した) 特定のメンバー数を持ち、これが分散データベースの voter メンバーの数を決定します。クラスタメンバーの過半数を恒久的に失った場合 (例えば、 3 つのメンバーのクラスタで 2 つのメンバーを消失した場合)、クラスタは過半数を失い利用不可能となります。しかし、最低 1 つのデータベースメンバーが生き残っていれば、クラスタを復旧できます。

このためには、以下のステップを実行してください。

1. クラスタ内の生き残っているどれかのメンバーにログオンし以下のコマンドを実行します。

   ```
   sudo lxd cluster list-database
   ```

   このコマンドはデータベースロールの 1 つを持つクラスタメンバーを表示します。

2. 一覧表示されたデータベースメンバーの 1 つを新しいリーダーとして選択します。そのマシンにログオンします (すでにログオンしたマシンと異なる場合)。

3. そのマシンで LXD デーモンが実行中でないことを確認します。例えば、snap を使用している場合は以下のようにします。

```
sudo snap stop lxd
```

4. まだオンラインである他の全てのクラスタメンバーにログオンし LXD デーモンを停止します。

5. 新しいリーダーとして選択したサーバーで以下のコマンドを実行します。

```
sudo lxd cluster recover-from-quorum-loss
```

6. 全てのマシンで再び LXD デーモンを開始し、新しいリーダーを始動させます。例えば、snap を使用している場合は以下のようにします。

```
sudo snap start lxd
```

これでデータベースはオンラインに戻るはずです。データベースから情報が削除されることはありません。失ったクラスタメンバーについての情報は、そのメンバーのインスタンスについてのメタデータも含めて、全て残っています。失ったインスタンスを再び作成する必要がある場合に、この情報がさらに復旧を進める上で役に立ちます。

失ったクラスタメンバーを恒久的に削除するには、強制削除します。**クラスタメンバーを削除する** を参照してください。

アドレス変更からクラスタメンバーを復旧する

クラスタのいくつかのメンバーがもう到達不能な場合、あるいはクラスタ自体が IP アドレスまたはリッスニングポート番号の変更のために到達不能な場合、クラスタを再設定できます。

そうするためには、クラスタの各メンバーでクラスタ設定を編集し、IP アドレスとリッスニンググポート番号を必要に応じて変更します。この操作中はメンバーは削除できません。クラスタ設定は完全なクラスタの記述を含む必要がありますので、全てのクラスタメンバー上で全てのクラスタメンバーを変更する必要があります。

異なるメンバーの **メンバーロール** を編集できますが、以下の制限があります。

- `database*` ロールを持たないクラスタメンバーは、グローバルデータベースがないため、 voter になれません。
- 少なくとも 2 つのメンバー (2 つのメンバーからなるクラスタの場合を除く、この場合は 1 つで十分) が voter にとどまる必要があります。そうでなければ過半数が成り立ちません。

各クラスタメンバーにログオンして以下のステップを実行します。

1. LXD デーモンを停止します。例えば、snap を使用していれば以下のようにします。

```
sudo snap stop lxd
```

2. 以下のコマンドを実行します。

```
sudo lxd cluster edit
```

3. クラスタメンバーがクラスタの他のメンバーについて持っている情報の YAML 表現を編集します。

```
# Latest dqlite segment ID: 1234

members:
  - id: 1              # メンバーの内部 ID（読み取り専用）
    name: server1      # クラスタメンバー名（読み取り専用）
    address: 192.0.2.10:8443 # メンバーの最終の既知のアドレス（変更可能）
    role: voter              # メンバーの最終の既知のロール（変更可能）
  - id: 2              # メンバーの内部 ID（読み取り専用）
    name: server2      # クラスタメンバー名（読み取り専用）
    address: 192.0.2.11:8443 # メンバーの最終の既知のアドレス（変更可能）
    role: stand-by           # メンバーの最終の既知のロール（変更可能）
  - id: 3              # メンバーの内部 ID（読み取り専用）
    name: server3      # クラスタメンバー名（読み取り専用）
    address: 192.0.2.12:8443 # メンバーの最終の既知のアドレス（変更可能）
    role: spare              # メンバーの最終の既知のロール（変更可能）
```

アドレスとロールを編集できます。

全てのクラスタメンバー上でこの変更をした後、全てのメンバーで LXD デーモンを再び開始します。例えば、snap を使用していれば以下のようにします。

```
sudo snap start lxd
```

クラスタは全てのメンバーが入った状態で再び完全に利用可能になるはずです。データベースから情報が削除されることはありません。クラスタメンバーとそれらのインスタンスについての情報は全て残っています。

Raft のメンバーシップを手動で変更する

場合によっては、なんらかの予期せぬ挙動のために Raft のメンバーシップ設定を手動で変更する必要があるかもしれません。

例えば、クラスタメンバーをクリーンでない状態で削除した場合、 `lxc cluster list` では表示されないが Raft 設定の一部として残るという状態になるかも知れません。Raft の設定を見るには以下のコマンドを使用します。

```
lxd sql local "SELECT * FROM raft_nodes"
```

この場合、残ったノードを削除するには以下のコマンドを使用します。

```
lxd cluster remove-raft-node <address>
```

3.9.5 クラスタ内のインスタンスを管理するには

クラスタのセットアップでは、各インスタンスはどれかのクラスタメンバー上で稼働します。どのクラスタメンバーからでも各インスタンスを操作できるので、インスタンスが配置されているクラスタメンバーにログオンする必要はありません。

特定のクラスタメンバー上でインスタンスを起動する

インスタンスを起動する際、特定のクラスタメンバー上で稼働するようにターゲットできます。これはどのクラスタメンバーからでも実行できます。

例えば、`c1` という名前のインスタンスを `server2` クラスタメンバー上で起動するには、以下のコマンドを使用します。

```
lxc launch images:ubuntu/22.04 c1 --target server2
```

インスタンスを特定のクラスタメンバーあるいは特定の **クラスタグループ** 上で実行できます。

ターゲットを指定しなかった場合、インスタンスはクラスタメンバーに自動的に割り当てられます。詳細は**インスタンスの自動配置**を参照してください。

インスタンスの配置を確認する

インスタンスがどのメンバーに配置されているかを確認するには、クラスタ内の全てのインスタンス一覧を表示します。

```
lxc list
```

location カラムに各インスタンスが稼働しているメンバーが表示されます。

インスタンスを移動する

既存のインスタンスを他のクラスタメンバーに移動できます。例えば、 c1 インスタンスを server1 クラスタメンバーに移動するには以下のコマンドを使用します。

```
lxc stop c1
lxc move c1 --target server1
lxc start c1
```

詳細は **サーバー間で既存の *LXD* インスタンスを移動するには** を参照してください。

3.9.6 クラスタのストレージを設定するには

クラスタの全てのメンバーは同一のストレージプール設定を持つ必要があります。メンバーごとに異なる設定が可能なのは *source*、*size*、*zfs.pool_name*、*lvm.thinpool_name* と *lvm.vg_name* だけです。詳細は **メンバー設定** を参照してください。

LXD は初期化時にデフォルトの local ストレージプールを各クラスタメンバーに作成します。

追加のストレージプールを作成するのは以下の 2 ステップで行います。

1. 全てのクラスタメンバーで新しいストレージプールを定義し設定します。例えば、 3 つのメンバーを持つクラスタでは以下のようにします。

   ```
   lxc storage create --target server1 data zfs source=/dev/vdb1
   lxc storage create --target server2 data zfs source=/dev/vdc1
   lxc storage create --target server3 data zfs source=/dev/vdb1 size=10GiB
   ```

   注釈: メンバー固有の設定キーは source、size、zfs.pool_name、lvm.thinpool_name と lvm.vg_name だけを渡せます。他の設定キーを渡すとエラーになります。

これらのコマンドはストレージプールを定義しますが作成はしません。`lxc storage list` を実行するとこのストレージプールは "pending" と表示されます。

2. 全てのクラスタメンバーでストレージプールを実在化させるには以下のコマンドを実行します。

```
lxc storage create data zfs
```

注釈: このコマンドにメンバー固有ではない設定キーを追加できます。

ストレージプールを定義した際のクラスタメンバーがいない、あるいはクラスタメンバーがダウンしている場合はエラーになります。

クラスタ内にストレージプールを作成する も参照してください

メンバー固有のプール設定を参照する

ストレージプールのクラスタ全体の設定を表示するには `lxc storage show <pool_name>` を実行します。

メンバー固有の設定を参照するには `--target` フラグを使用してください。例えば以下のようにします。

```
lxc storage show data --target server2
```

ストレージボリュームを作成する

ほとんどのストレージドライバ (Ceph ベースのストレージドライバを除いて)、ストレージボリュームはクラスタ内で複製されず、ストレージを作成したメンバー上にのみ存在します。特定のボリュームがどのメンバー上にあるのかを見るには `lxc storage volume list <pool_name>` を実行してください。

ストレージボリュームを作成する際に `--target` フラグを使用すると特定のクラスタメンバー上にストレージボリュームを作成できます。フラグを指定しない場合、ボリュームはコマンドを実行したクラスタメンバー上に作成されます。例えば、 `server1` というクラスタメンバー上でボリュームを作成するには以下のようにします。

```
lxc storage volume create local vol1
```

他のクラスタメンバー上で同じ名前のボリュームを作成するには以下のようにします。

```
lxc storage volume create local vol1 --target server2
```

別のボリュームも別のクラスタメンバー上にある限り同じ名前を持つことができます。典型的な例はイメージボリュームです。

クラスタ内のストレージボリュームは、指定した名前のボリュームを複数のクラスタメンバーが持つ場合は --target フラグを指定する必要があるという点を除けば、クラスタではない LXD 環境と同じように管理できます。例えば、ストレージボリュームの情報を表示するには以下のようにします。

```
lxc storage volume show local vol1 --target server1
lxc storage volume show local vol1 --target server2
```

3.9.7 クラスタのネットワークを設定するには

クラスタの全てのメンバーは同一のネットワーク設定を持つ必要があります。メンバーごとに異なってもよい設定は *bridge.external_interfaces*、*parent*、*bgp.ipv4.nexthop* と *bgp.ipv6.nexthop* だけです。詳細は **メンバー設定** を参照してください。

追加のネットワークを作成する際は以下の 2 ステップで行います。

1. 全てのクラスタメンバー上で新しいネットワークを定義し設定します。例えば、3 つのメンバーを持つクラスタでは以下のようにします。

   ```
   lxc network create --target server1 my-network
   lxc network create --target server2 my-network
   lxc network create --target server3 my-network
   ```

   注釈: メンバー固有の設定キーは bridge.external_interfaces、parent、bgp.ipv4.nexthop と bgp.ipv6.nexthop だけを渡せます。他の設定キーを渡すとエラーになります。

   これらのコマンドはネットワークを定義しますが作成はしません。lxc network list を実行するとこのネットワークは "pending" と表示されます。

2. 全てのクラスタメンバーでネットワークを実在化させるには以下のコマンドを実行します。

   ```
   lxc network create my-network
   ```

   注釈: このコマンドにメンバー固有ではない設定キーを追加できます。

   ネットワークを定義した際のクラスタメンバーがいない、あるいはクラスタメンバーがダウンしている場合はエラーになります。

クラスタ内にネットワークを作成する も参照してください。

個別の REST API とクラスタネットワーク

クライアントの REST API エンドポイント用とクラスタメンバー間の内部トラフィック用で別のネットワークを設定できます。例えば、DNS ラウンドロビンで REST API に仮想アドレスを使う場合にこの分離は役立ちます。

そうするためには、*cluster.https_address* (クラスタ内部トラフィック用のアドレス) と *core.https_address* (REST API のアドレス) に異なるアドレスを指定する必要があります。

1. 通常通りクラスタを作成し、クラスタ内部トラフィックに使うクラスタのアドレスを忘れずに使用する。このアドレスは cluster.https_address で設定します。
2. メンバーがジョインした後、 REST API のアドレスを core.https_address で設定する。例えば以下のようにします。

```
lxc config set core.https_address 0.0.0.0:8443
```

注釈: core.https_address はクラスタメンバーに固有ですので、異なるメンバーに異なるアドレスを設定できます。メンバーに複数のインタフェースでリッスンするようにワイルドカードアドレスを使用することもできます。

3.9.8 クラスタグループをセットアップするには

クラスタメンバーは **クラスタグループ** にアサインできます。デフォルトでは、全てのクラスタメンバーは default グループに属しています。

クラスタグループを作成するには、lxc cluster group create コマンドを使用します。例えば以下のようにします。

```
lxc cluster group create gpu
```

クラスタメンバーを 1 つまたは複数のグループに割り当てるには、lxc cluster group assign コマンドを使用します。このコマンドは、指定したクラスタメンバーを現在所属しているすべてのクラスタグループから削除し、その後、指定したグループまたはグループに追加します。

たとえば、server1 を gpu グループのみに割り当てるには、次のコマンドを使用します:

```
lxc cluster group assign server1 gpu
```

server1 を gpu グループに割り当てるとともに、default グループにも保持させるためには、以下のコマンドを使用します：

```
lxc cluster group assign server1 default,gpu
```

クラスタグループメンバー上でインスタンスを起動する

クラスタグループがある場合、インスタンスを、特定のメンバー上で動かすようにターゲットする代わりに、クラスタグループのいずれかのメンバー上で動かすようにターゲットできます。

注釈: クラスタグループにインスタンスをターゲットできるようにするには *scheduler.instance* は `all` (デフォルト) または `group` に設定する必要があります。

詳細は**インスタンスの自動配置**を参照してください。

クラスタグループのメンバー上でインスタンスを起動するには、**特定のクラスタメンバー上でインスタンスを起動する** の指示に従ってください。ただし `--target` フラグではグループ名の前に `@` をつけて指定してください。例えば以下のようにします。

```
lxc launch images:ubuntu/22.04 c1 --target=@gpu
```

3.9.9 クラスタメンバーの設定

各クラスタメンバーは以下のサポートされるネームスペース内で独自のキー・バリュー設定を持てます。

- `user` (ユーザーのメタデータ用に自由形式のキー・バリュー)
- `scheduler` (メンバーが自クラスタによりどのように動的にターゲットされるかに関連するオプション)

現状サポートされるキーは以下の通りです。

キー	型	デフォルト値	説明
`scheduler.instance`	string	`all`	指定可能な値は `all`、`manual`、`group`。詳細は**インスタンスの自動配置**参照。
`user.*`	string	-	自由形式のユーザーのキー・バリュー・ストレージ (検索で使用可能)。

3.10 本番環境のセットアップ

3.10.1 パフォーマンスチューニングについて

お使いの LXD 環境を本番稼働に移行する準備が出来たら、システムのパフォーマンスを最適化するためにいくらか時間を取るほうが良いです。パフォーマンスに影響を与えるいくつかの視点があります。お使いの LXD 環境を改善するためにチューニングするべき選択肢と設定を決定するのに以下の手順が役立ちます。

ベンチマークを実行する

LXD はシステムのパフォーマンスを評価するためにベンチマークツールを提供しています。このツールを使って複数のコンテナを初期化・起動し、システムがコンテナを作成するのに必要な時間を計測できます。異なる LXD の設定、システム設定、さらにはハードウェア構成に対して繰り返しツールを実行することで、パフォーマンスを比較し、どの設定が理想的か評価できます。

ツールを実行する手順については **パフォーマンスをベンチマークするには** を参照してください。

インスタンスのメトリクスをモニターする

LXD は全ての実行中のインスタンスについてのメトリクスといくつかの内部メトリクスを収集します。これは CPU、メモリー、ネットワーク、ディスク、プロセスの使用量を含みます。Prometheus で読み取って Grafana でグラフを表示するのに使うことを想定しています。利用可能なメトリクスの一覧は**提供されるメトリクス**を参照してください。

あなたのインスタンスが使用しているリソースを見積もるために定期的にメトリクスをモニターするほうが良いです。スパイクやボトルネックがある場合や、使用量のパターンが変化したり、設定を見直す必要がある場合に、これらの数値が役立ちます。

メトリクス収集についての詳細な情報は **メトリクスを監視するには** を参照してください。

サーバー設定をチューニングする

ほとんどの Linux ディストリビューションのデフォルトのカーネル設定は大量のコンテナや仮想マシンを稼働させるのに最適化されていません。ですので、デフォルトの設定で引き起こされる制限にひっかかるのを避けるため、関連する設定を確認、変更するほうが良いです。

これらの制限にひっかかかった場合の典型的なエラーは以下のようなものです。

- `Failed to allocate directory watch: Too many open files`
- `<Error> <Error>: Too many open files`
- `failed to open stream: Too many open files in...`

- neighbour: ndisc_cache: neighbor table overflow!

関連するサーバー設定と提案される値の一覧は *LXD* **プロダクション環境のサーバー設定** を参照してください。

ネットワーク帯域幅をチューニングする

インスタンス間あるいは LXD ホストとインスタンス間で大量のローカルなアクティビティがある場合、あるいは高速なインターネット接続をお持ちの場合、 LXD のセットアップのネットワーク帯域幅を増やすことを検討すると良いです。これは送信と受信のキューの長さを拡張することで実現できます。

手順については **ネットワーク帯域幅を拡大するには** を参照してください。

3.10.2 パフォーマンスをベンチマークするには

LXD サーバーあるいはクラスタのパフォーマンスは、ハードウェア、サーバー設定、選択されたストレージドライバ、ネットワーク帯域幅から全体的な利用パターンに至るまでの多数の異なる因子によって変わります。

最適な設定を見つけるには、異なるセットアップを評価するためベンチマークテストを実行するほうが良いです。

LXD ではこの目的のためベンチマークツールを提供しています。このツールを使って複数のコンテナを初期化・起動し、システムがコンテナを作成するのに必要な時間を計測できます。異なる LXD の設定、システム設定、さらにはハードウェア構成に対して繰り返しツールを実行することで、パフォーマンスを比較し、どの設定が理想的か評価できます。

ツールを取得する

snap をご利用の場合、ベンチマークツールは自動でインストールされます。`lxd.benchmark` で利用できます。

それ以外、 LXD をディストリビューションのパッケージマネージャからインストールしたかソースからビルドした場合は、ツールは `lxd-benchmark` で利用できます。もし存在しない場合は、 go (バージョン 1.18 以降) がインストールされていることを確認の上、以下のコマンドでツールをインストールしてください。

```
go install github.com/lxc/lxd/lxd-benchmark@latest
```

ツールを実行する

あなたの LXD のパフォーマンスを測定するには `lxd.benchmark [action]` を実行してください。(このコマンドはあなたが snap を使用していると想定しています。そうでない場合 `lxd.benchmark` を `lxd-benchmark` と読み替えてください。以下の例でも同様です)

ベンチマークは現在の LXD 設定を使用します。別のプロジェクトを使用したい場合は、`--project` で指定してください。

全てのアクションについて、使用する並列スレッド数 (デフォルトはダイナミックなバッチサイズを使用) を指定できます。また結果を CSV のレポートファイルに追加し、一定の方法でラベル付けすることもできます。

利用可能なアクションとフラグについては lxd.benchmark help を参照してください。

イメージを選択する

ベンチマークを実行する前に、使用したいイメージの種別を選んでください。

ローカルイメージ コンテナの作成にかかる時間を計測し、イメージをダウンロードするのにかかる時間を無視したい場合は、ベンチマークツールを実行する前にイメージをローカルのイメージストアにコピーするのが良いです。

そうするには、以下のようなコマンドを実行し、 lxd.benchmark の実行時にはイメージのフィンガープリント (例えば 2d21da400963) を指定します。

```
lxc image copy images:ubuntu/22.04 local:
```

またイメージにエイリアスを割り当てて、lxd.benchmark の実行時にエイリアス (例えば ubuntu) を指定することもできます。

```
lxc image copy images:ubuntu/22.04 local: --alias ubuntu
```

リモートイメージ 全体の結果にダウンロード時間も含めたい場合は、リモートイメージ (例えば images:ubuntu/22.04) を指定します。lxd.benchmark が使用するデフォルトのイメージは最新の Ubuntu イメージ (ubuntu:) ですので、このイメージを使用したい場合は、ツール実行時にイメージ名を省略できます。

コンテナを作成、起動する

指定した数のコンテナを作成するには以下のコマンドを実行します。

```
lxd.benchmark init --count <number> <image>
```

特権コンテナを作成するにはコマンドに --privileged を追加します。

例

コマンド	説明
`lxd.benchmark init --count 10 --privileged`	最新の Ubuntu イメージを使用して 10 個の特権コンテナを作成する。
`lxd.benchmark init --count 20 --parallel 4 images:alpine/edge`	Alpine Edge イメージを使用する 20 個のコンテナを 4 つのパラレルスレッドを使用して作成する。
`lxd.benchmark init 2d21da400963`	フィンガープリントが `2d21da400963` のローカルイメージを使用して 1 個のコンテナを作成する。
`lxd.benchmark init --count 10 ubuntu`	`ubuntu` のエイリアスを設定されたイメージを使用して 10 個のコンテナを作成する。

`init` アクションを使用するとコンテナを作成するが起動はせずにベンチマークを実行します。作成したコンテナを起動するには、以下のコマンドを実行します。

```
lxd.benchmark start
```

あるいは `launch` アクションを使用してコンテナを作成し起動します。

```
lxd.benchmark launch --count 10 <image>
```

このアクションでは、`--freeze` フラグを追加するとコンテナの起動直後に凍結できます。コンテナを凍結するとプロセスは一時停止しますので、このフラグは各コンテナ内でプロセスが起動後に干渉するのを回避して純粋な起動時間を計測できます。

コンテナを削除する

作成したベンチマーク用のコンテナを削除するには、以下のコマンドを実行します。

```
lxd.benchmark delete
```

注釈: 新しいベンチマークを実行する前には既存のベンチマーク用コンテナを全て削除する必要があります。

3.10.3 メトリクスを監視するには

LXD は全ての実行中のインスタンスについてのメトリクスといくつかの内部メトリクスを収集します。これは CPU、メモリー、ネットワーク、ディスク、プロセスの使用量を含みます。Prometheus で読み取って Grafana でグラフを表示するのに使うことを想定しています。利用可能なメトリクスの一覧は**提供されるメトリクス**を参照してください。

クラスタ環境では、 LXD はアクセスされているサーバー上で稼働中のインスタンスの値だけを返します。ですので、各クラスタメンバーから別々にデータを取得する必要があります。。

インスタンスメトリクスは `/1.0/metrics` エンドポイントを呼ぶと更新されます。複数のスクレイパーに対応するためメトリクスは 8 秒キャッシュします。メトリクスの取得は比較的重い処理ですので、影響が大きすぎるようならデフォルトの間隔より長い間隔でスクレイピングすることを検討してください。

生データを取得する

LXD が収集した生データを見るには、`1.0/metrics` エンドポイントに `lxc query` コマンドで問い合わせてください。

```
user@host:~$ lxc query /1.0/metrics                          # HELP lxd_cpu_seconds_total The
total number of CPU time used in seconds.# TYPE lxd_cpu_seconds_total
counterlxd_cpu_seconds_total{cpu="0",mode="system",name="u1",project="default",
type="container"} 60.304517lxd_cpu_seconds_total{cpu="0",mode="user",name="u1",
project="default",type="container"} 145.647502lxd_cpu_seconds_total{cpu="0",
mode="iowait",name="vm",project="default",type="virtual-machine"} 4614.
78lxd_cpu_seconds_total{cpu="0",mode="irq",name="vm",project="default",
type="virtual-machine"} 0lxd_cpu_seconds_total{cpu="0",mode="idle",name="vm",
project="default",type="virtual-machine"} 412762lxd_cpu_seconds_total{cpu="0",
mode="nice",name="vm",project="default",type="virtual-machine"} 35.
06lxd_cpu_seconds_total{cpu="0",mode="softirq",name="vm",project="default",
type="virtual-machine"} 2.41lxd_cpu_seconds_total{cpu="0",mode="steal",name="vm",
project="default",type="virtual-machine"} 9.84lxd_cpu_seconds_total{cpu="0",
mode="system",name="vm",project="default",type="virtual-machine"} 340.
84lxd_cpu_seconds_total{cpu="0",mode="user",name="vm",project="default",
type="virtual-machine"} 261.25# HELP lxd_cpu_effective_total The total number of
effective CPUs.# TYPE lxd_cpu_effective_total gaugelxd_cpu_effective_total{name="u1",
project="default",type="container"} 4lxd_cpu_effective_total{name="vm",project="default",
type="virtual-machine"} 0# HELP lxd_disk_read_bytes_total The total number of bytes
read.# TYPE lxd_disk_read_bytes_total counterlxd_disk_read_bytes_total{device="loop5",
name="u1",project="default",type="container"} 2048lxd_disk_read_bytes_total{device="loop3",
name="vm",project="default",type="virtual-machine"} 353280...
```

Prometheus をセットアップする

生のメトリクスを収集し保管するには、Prometheus をセットアップするのが良いです。メトリクス API エンドポイントを使ってメトリクスを収集するように設定できます。

メトリクスエンドポイントを公開する

/1.0/metrics API エンドポイントを公開するには、利用可能にするアドレスを設定する必要があります。

そのためには、*core.metrics_address* サーバー設定オプションか *core.https_address* サーバー設定オプションのいずれかを設定できます。core.metrics_address オプションはメトリクスのみを公開し、core.https_address は完全な API を公開します。ですので、完全な API とメトリクスの API で別のアドレスを使いたい場合、あるいはメトリクスの API のみ公開し完全な API は公開したくない場合は core.metrics_address オプションを設定するのが良いです。

例えば、完全な API を 8443 ポートで公開するには、次のコマンドを入力します:

```
lxc config set core.https_address ":8443"
```

メトリクス API エンドポイントのみを 8444 ポートで公開するには、次のコマンドを入力します:

```
lxc config set core.metrics_address ":8444"
```

メトリクス API エンドポイントのみを指定した IP アドレスとポートで公開するには、次のようなコマンドを入力します:

```
lxc config set core.metrics_address "192.0.2.101:8444"
```

メトリクス用証明書の追加

1.0/metrics エンドポイントは他の証明書に加えて metrics タイプの証明書を受け付けるという点で特別なエンドポイントです。このタイプの証明書はメトリクス専用で、インスタンスや他の LXD のエンティティの操作には使用できません。

新しい証明書は以下のように作成します（この手順はメトリクス用の証明書に限ったものではありません）。

```
openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -sha384 -keyout␣
↪metrics.key -nodes -out metrics.crt -days 3650 -subj "/CN=metrics.local"
```

注釈: 上のコマンドは OpenSSL 1.1.0 またはそれ以降が必要です。

作成後、証明書を信頼済みクライアントのリストに metrics というタイプを指定して追加する必要があります。

```
lxc config trust add metrics.crt --type=metrics
```

メトリクス用証明書を Prometheus で利用可能にする

Prometheus を LXD サーバーと別のマシンで稼働させる場合、必要な証明書を Prometheus のマシンにコピーする必要があります。

- 作成したメトリクス用証明書 (`metrics.crt`) と鍵 (`metrics.key`)
- `/var/snap/lxd/common/lxd/` (snap を使用している場合) あるいは`/var/lib/lxd/` (それ以外) に置かれている LXD サーバー証明書 (`server.crt`)

これらのファイルを Prometheus にアクセスできる `tls` ディレクトリ、例えば、`/var/snap/prometheus/common/tls`(snap を使用している場合) あるいは`/etc/prometheus/tls` (それ以外) にコピーしてください。次の例のコマンドを参照してください:

```
# tls ディレクトリーを作成
mkdir /var/snap/prometheus/common/tls

# 新規に作成された証明書と鍵を tls ディレクトリーにコピー
cp metrics.crt metrics.key /var/snap/prometheus/common/tls/

# LXD サーバー証明書を tls ディレクトリーにコピー
cp /var/snap/lxd/common/lxd/server.crt /var/snap/prometheus/common/tls/
```

Snap を使っていない場合、さらに Prometheus がこれらのファイルを読めるように (通常、Prometheus は `prometheus` ユーザで稼働しています) する必要があります。

```
chown -R prometheus:prometheus /etc/prometheus/tls
```

Prometheus を LXD からデータ収集できるように設定する

最後に、 LXD をターゲットとして Prometheus の設定に追加する必要があります。

そのためには、`/var/snap/prometheus/current/prometheus.yml` (snap を使用している場合) あるいは`/etc/prometheus/prometheus.yaml` (それ以外) を編集し、LXD にジョブを追加します。

必要な設定は以下のようになります:

```
scrape_configs:
  - job_name: lxd
    metrics_path: '/1.0/metrics'
```

(次のページに続く)

(前のページからの続き)

```
    scheme: 'https'
    static_configs:
      - targets: ['foo.example.com:8443']
    tls_config:
      ca_file: 'tls/server.crt'
      cert_file: 'tls/metrics.crt'
      key_file: 'tls/metrics.key'
      # XXX: server_name は targets のホスト名が証明書でカバーされない
      #      (証明書の SAN リストに含まれない) 場合は必須です
      server_name: 'foo'
```

注釈: LXD 証明書が targets リスト内で使用するのと同じホスト名を含まない場合は server_name の指定は必須です。これを確認するには、server.crt を開いて Subject Alternative Name (SAN) セクションを確認してください。

例えば、server.crt が以下の内容を持つとします:

```
user@host:~$ openssl x509 -noout -text -in /var/snap/prometheus/common/tls/server.
crt                ... X509v3 Subject Alternative Name:  DNS:foo, IP Address:127.0.0.1, IP
Address:0:0:0:0:0:0:0:1...
```

Subject Alternative Name (SAN) リストが targets リスト (foo.example.com) のホスト名を含んでいないので、 server_name ディレクティブを使用して比較に使用する名前を上書きする必要があります。

以下は複数の LXD サーバーのメトリックを収集するために複数のジョブを使用する prometheus.yaml の設定例です。

```
scrape_configs:
  # abydos, langara, orilla は最初に abydos からブートストラップした単一クラスタで
  # (ここでは `hdc`と呼びます)、このため 3 ノードで `ca_file`と `server_name`を共有しています。
  # `ca_file`は LXD クラスタの各メンバー上に存在する `/var/snap/lxd/common/lxd/cluster.crt`
  # ファイルに対応しています。
  #
  # 注意: `project`パラメータは `default`プロジェクトを使用しないか複数のプロジェクトを
  #       使用する場合に提供されます。
  #
  # 注意: クラスタの各メンバーはローカルで稼働するインスタンスのメトリクスだけを提供します。
  #       これが `lxd-hdc`クラスタが 3 つのターゲットを一覧表示している理由です。
  - job_name: "lxd-hdc"
    metrics_path: '/1.0/metrics'
```

(次のページに続く)

(前のページからの続き)

```
    params:
      project: ['jdoe']
    scheme: 'https'
    static_configs:
      - targets:
        - 'abydos.hosts.example.net:8444'
        - 'langara.hosts.example.net:8444'
        - 'orilla.hosts.example.net:8444'
    tls_config:
      ca_file: 'tls/abydos.crt'
      cert_file: 'tls/metrics.crt'
      key_file: 'tls/metrics.key'
      server_name: 'abydos'

  # jupiter, mars, saturn は 3 つのスタンドアロンの LXD サーバーです。
  # 注意: これらでは `default`プロジェクトのみが使用されているため、プロジェクトの設定は省略しています。
  - job_name: "lxd-jupiter"
    metrics_path: '/1.0/metrics'
    scheme: 'https'
    static_configs:
      - targets: ['jupiter.example.com:9101']
    tls_config:
      ca_file: 'tls/jupiter.crt'
      cert_file: 'tls/metrics.crt'
      key_file: 'tls/metrics.key'
      server_name: 'jupiter'

  - job_name: "lxd-mars"
    metrics_path: '/1.0/metrics'
    scheme: 'https'
    static_configs:
      - targets: ['mars.example.com:9101']
    tls_config:
      ca_file: 'tls/mars.crt'
      cert_file: 'tls/metrics.crt'
      key_file: 'tls/metrics.key'
      server_name: 'mars'
```

(次のページに続く)

(前のページからの続き)

```
- job_name: "lxd-saturn"
  metrics_path: '/1.0/metrics'
  scheme: 'https'
  static_configs:
    - targets: ['saturn.example.com:9101']
  tls_config:
    ca_file: 'tls/saturn.crt'
    cert_file: 'tls/metrics.crt'
    key_file: 'tls/metrics.key'
    server_name: 'saturn'
```

設定を編集後、Prometheus を再起動する（例えば、`snap restart prometheus`）とデータ収集を開始します。

Grafana ダッシュボードをセットアップする

メトリクスデータを可視化するには、Grafana を設定します。LXD は、Prometheus によって収集された LXD メトリクスを表示するように設定された Grafana ダッシュボードを提供します。

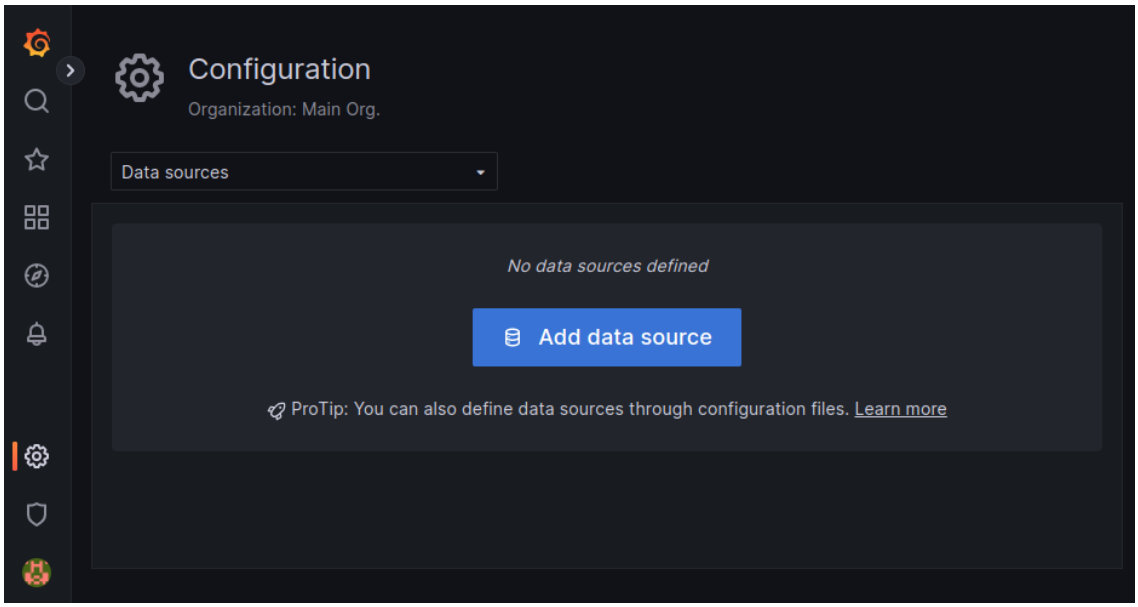
注釈: このダッシュボードは Grafana 8.4 以降が必要です。

Grafana のドキュメントを参照して、インストールとサインインの手順を確認してください:

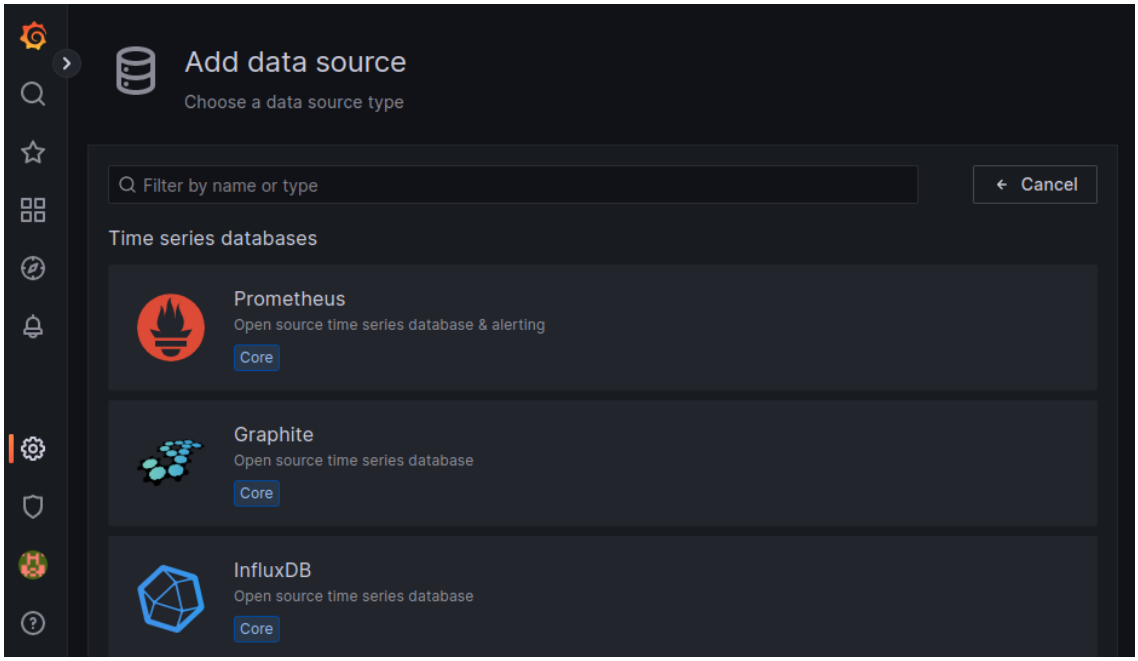
- Grafana をインストールする
- Grafana にサインインする

次の手順で LXD ダッシュボードをインポートします:

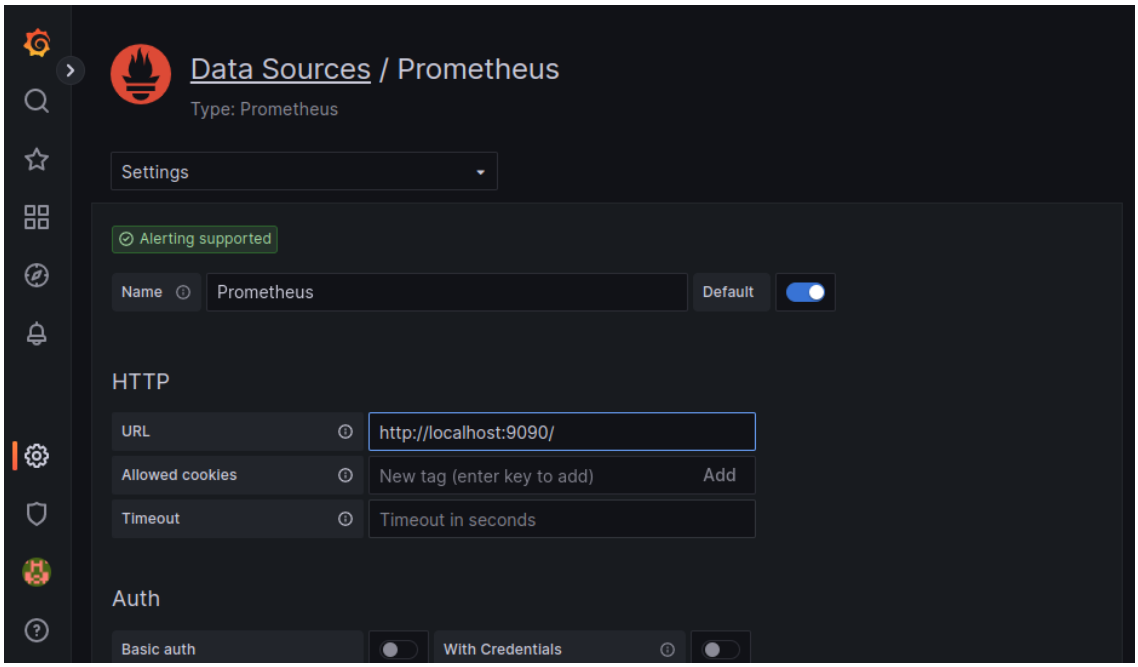
1. Prometheus をデータソースとして設定します:
    1. *Configuration* > *Data sources* に移動します。
    2. *Add data source* をクリックします。



3. *Prometheus* を選択します。



4. *URL* フィールドに `http://localhost:9090/` を入力します。

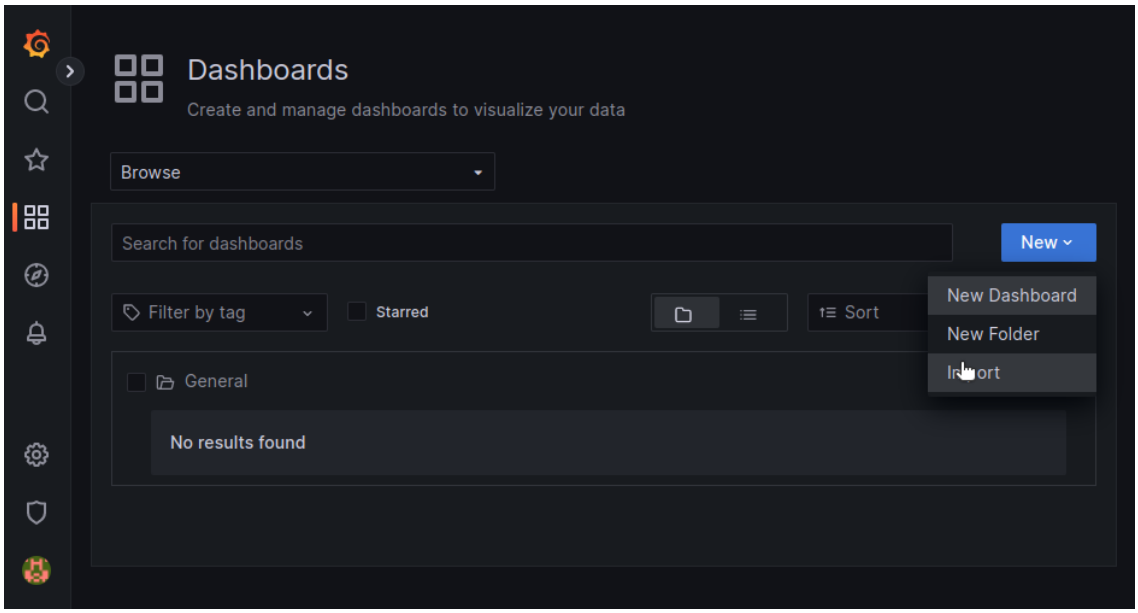


5. 他のフィールドはデフォルトの設定のままにし、**保存＆テスト**をクリックします。

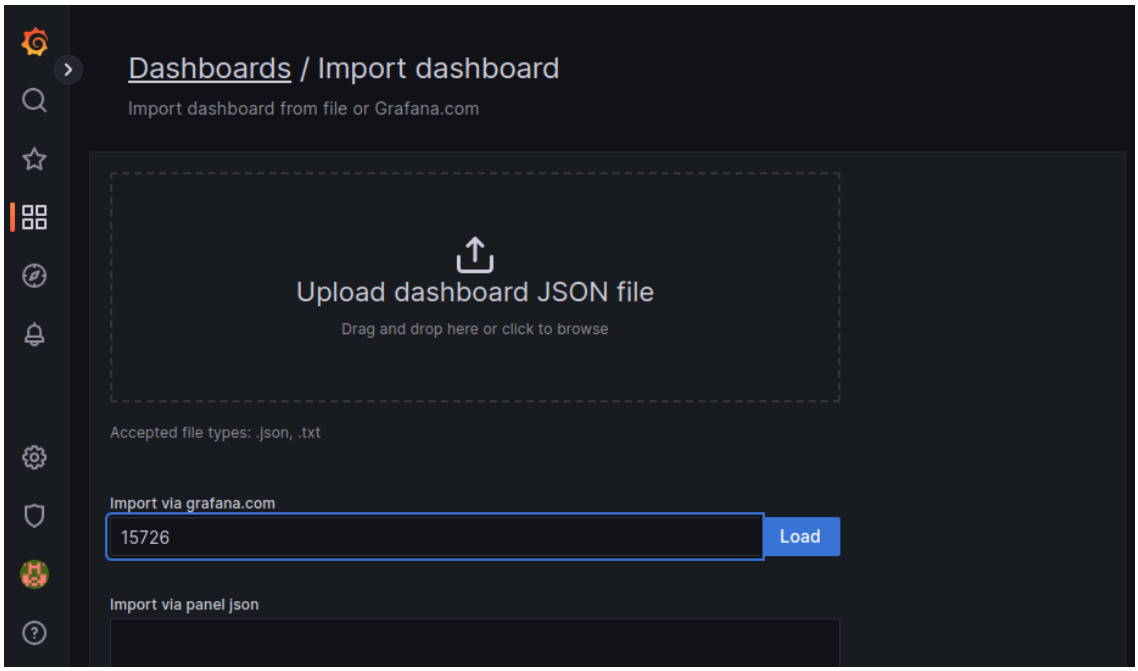
2. LXD ダッシュボードをインポートします:

   1. *Dashboards* > *Browse* に移動します。

   2. *New* をクリックし、*Import* を選択します。

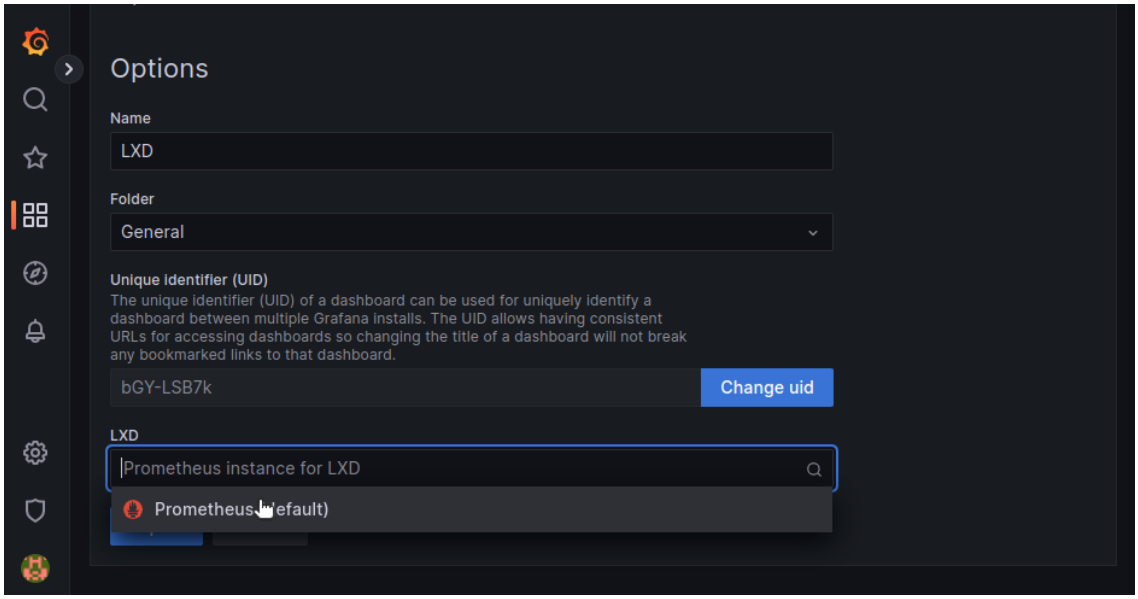


   3. *Import via grafana.com* フィールドにダッシュボード ID 15726 を入力します。



4. *Load* をクリックします。

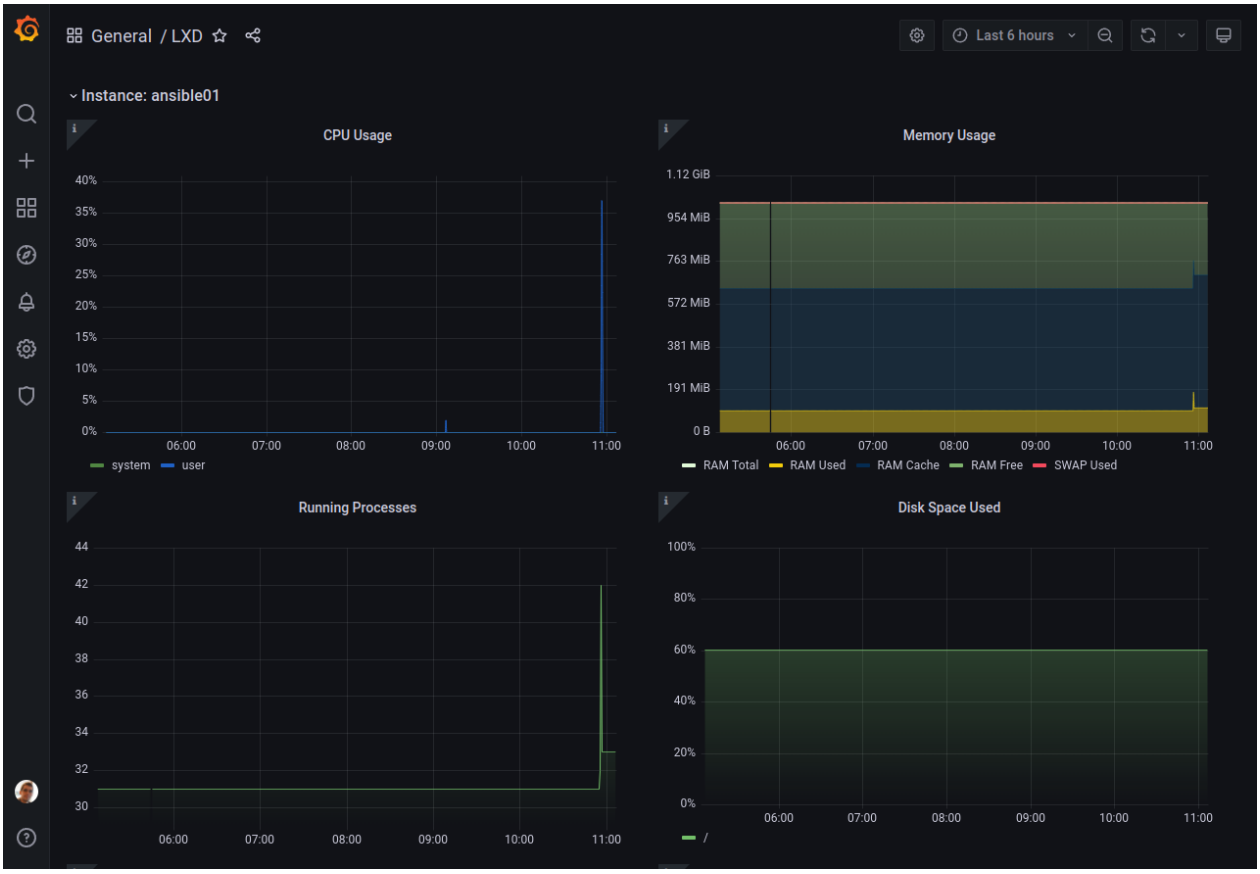
5. *LXD* のドロップダウンメニューから、設定した Prometheus のデータソースを選択します。



6. *Import* をクリックします。

これで LXD ダッシュボードが表示されるはずです。プロジェクトを選択し、インスタンスによってフィルタリングすることができます。

ページの下部で、各インスタンスのデータを見ることができます。



3.10.4 ネットワーク帯域幅を拡大するには

あなたの LXD 環境のネットワーク帯域幅を送信キューの長さ (txqueuelen) を調整することで拡大できます。以下のようなシナリオでは適しているでしょう。

- 大量のローカルアクティビティ (インスタンス間接続あるいはホストあるいはインスタンス間の接続) がある LXD ホスト上に 1 GbE あるいはそれ以上の NIC 1 GbE あるいはそれ以上の NIC がある場合
- LXD ホストで 1 GbE あるいはそれ以上のインターネット接続がある場合

使用するインスタンス数が多いほど、この設定変更の利益があります。

注釈: 以下の手順では txqueuelen の値として 10000 (10GbE NIC でよく使用されます) を、net.core.netdev_max_backlog の値として 182757 を使用しています。ネットワークによっては、異なる値を使用する必要があるかもしれません。

一般的に、低速なデバイスでレイテンシが高い場合は小さい txqueuelen の値を、レイテンシが低いデバイスでは大きな txqueuelen の値を使用するのが良いです。net.core.netdev_max_backlog の値について、良い指標は net.ipv4.tcp_mem 設定の最小値を使用することです。

LXD ホスト上のネットワーク帯域幅を拡大する

LXD ホスト上のネットワーク帯域幅を拡大するには以下の手順を実行してください。

1. 実 NIC と LXD NIC (例: lxdbr0) の両方で送信キューの長さ (txqueuelen) を拡大します。テストのために一時的にこれを行うには以下のコマンドが使用できます。

   ```
   ifconfig <interface> txqueuelen 10000
   ```

   変更を恒久的にするには /etc/network/interfaces 内のインタフェース設定に以下のコマンドを追加します。

   ```
   up ip link set eth0 txqueuelen 10000
   ```

2. 受信キューの長さ (net.core.netdev_max_backlog) を拡大します。テストのために一時的にこれを行うには以下のコマンドが使用できます。

   ```
   echo 182757 > /proc/sys/net/core/netdev_max_backlog
   ```

   変更を恒久的にするには /etc/sysctl.conf に以下の設定を追加します。

```
net.core.netdev_max_backlog = 182757
```

インスタンス上の送信キューの長さを拡大する

インスタンス上の全ての Ethernet インタフェースの `txqueulen` の値も変更する必要があります。このためには、以下の方法のいずれかを使います:

- 上述の LXD ホストへの変更と同じ変更を適用する。
- インスタンスのプロファイルあるいは設定で `queue.tx.length` デバイスオプションを設定する。

3.10.5 LXD サーバーをバックアップする

何をバックアップするか

LXD サーバーのバックアップを計画する際は、 LXD に保管／管理されている全ての異なるエンティティについて考慮してください。

- インスタンス (データベースのレコードとファイルシステム)
- イメージ (データベースのレコード、イメージファイル、そしてファイルシステム)
- ネットワーク (データベースのレコードと状態ファイル)
- プロファイル (データベースのレコード)
- ストレージボリューム (データベースのレコードとファイルシステム)

データベースだけをバックアップあるいはインスタンスだけをバックアップしても完全に機能するバックアップにはなりません。

ディザスターリカバリのシナリオによっては、上記のようなバックアップも妥当かもしれませんが、素早くオンラインに復帰することが目標なら、使用している LXD の全ての異なるピースを考慮してください。

フルバックアップ

フルバックアップは `/var/lib/lxd` あるいは snap ユーザーの場合は `/var/snap/lxd/common/lxd` の全体を含みます。

LXD が外部ストレージを使用している場合はそれらも適切にバックアップする必要があります。これは LVM ボリュームグループや ZFS プールなど LXD に直接含まれていないあらゆる外部のリソースです。

リストアにはリストア先のサーバー上の LXD の停止、 LXD ディレクトリの削除、そしてバックアップと必要な外部リソースのリストアを含みます。

snap パッケージを使っておらず、かつシステムに /etc/subuid と /etc/subgid ファイルがある場合、 lxd と root ユーザーの両方についてこれらのファイルあるいは少なくともこれらのファイル内のエントリーを復元することも良い考えです（コンテナのファイルシステムの不要なシフトを防ぎます）。

その後再び LXD を起動し、全てが正常に動作するか確認してください。

LXD サーバーのセカンダリバックアップ

LXD は 2 つのホスト間でインスタンスとストレージボリュームのコピーと移動をサポートしています。

ですので予備のサーバーがあれば、インスタンスとストレージボリュームを時々そのセカンダリサーバーにコピーしておき、オフラインの予備あるいは単なるストレージサーバーとして稼働させることが可能です。そして必要ならばそこからインスタンスをコピーして戻すことができます。

インスタンスのバックアップ

lxc export コマンドがインスタンスをバックアップの tarball にエクスポートするのに使えます。これらの tarball はデフォルトで全てのスナップショットを含みますが、同じストレージプールバックエンドを使っている LXD サーバーにリストアすることがわかっていれば「最適化」された tarball を取得することもできます。

サーバー上にインストールされたどんな圧縮ツールでも --compression を指定することで利用可能です。LXD 側でのバリデーションはなく、 LXD から実行可能で -c オプションで標準出力への出力をサポートしているコマンドであれば動作します。

これらの tarball はあなたが望むどんなファイルシステム上にどのようにでも保存することができ、 lxc import コマンドを使って LXD にインポートして戻すことができます。

ディザスタリカバリ

LXD は lxd recover コマンドを提供しています（通常の lxc コマンドではなく lxd コマンドであることに注意）。これはインタラクティブな CLI ツールでデータベース内に存在する全てのストレージプールをスキャンしリカバリー可能な焼失したボリュームを探します。また（ディスク上には存在するがデータベース内には存在しない）任意の未知のストレージプールの詳細をユーザーが指定してそれらに対してもスキャンを試みることもできます。

指定されたインスタンスを復元するのに必要な全ての（インスタンス設定、アタッチしたデバイス、ストレージボリューム、プール設定も含めた）情報を含む各インスタンスのストレージボリューム内の backup.yaml ファイルを LXD は保管しているため、それをインスタンス、ストレージボリューム、ストレージプールのデータベースレコードをリビルドするのに使用できます。

lxd recover ツールはストレージプールを（まだマウントされていなければ）マウントし、 LXD に関係すると思われる未知のボリュームをスキャンしようと試みます。各インスタンスボリュームについては LXD はマウントして backup.yaml ファイルにアクセスしようと試みます。その後 backup.yaml ファイルの内容と（対応するス

ナップショットなど）ディスク上に実際に存在するものとを比較してある程度の整合性チェックを行い、問題なければデータベースのレコードを再生成します。

ストレージプールのデータベースレコードも作成が必要な場合、ディスカバリーフェーズにユーザーが入力した情報よりも、インスタンスの backup.yaml ファイルを設定のベースとして優先して使用します。ただし、それが無い場合はユーザーが入力した情報をもとにプールのデータベースレコードを復元するようにフォールバックします。

3.10.6 LXD プロダクション環境のサーバー設定

あなたの LXD サーバーで多数のインタンスを稼働できるようにするには、サーバーの制限値にひっかからないよう以下の設定を調整してください。

値 のカラムに各パラメータの推奨値を記載しています。

/etc/security/limits.conf

注釈: snap ユーザーの場合はこれらの制限は自動的に上げられます。

ドメイン	種別	項目	値	デフォルト	説明
*	soft	nofile	1048576	未設定	オープンするファイルの最大数
*	hard	nofile	1048576	未設定	オープンするファイルの最大数
root	soft	nofile	1048576	未設定	オープンするファイルの最大数
root	hard	nofile	1048576	未設定	オープンするファイルの最大数
*	soft	memloc	unlimite	未設定	ロックされたメモリ内の最大のアドレス空間 (KB)
*	hard	memloc	unlimite	未設定	ロックされたメモリ内の最大のアドレス空間 (KB)
root	soft	memloc	unlimite	未設定	ロックされたメモリ内の最大のアドレス空間 (KB) (bpf システムコール監視にのみ必要)
root	hard	memloc	unlimite	未設定	ロックされたメモリ内の最大のアドレス空間 (KB) (bpf システムコール監視にのみ必要)

/etc/sysctl.conf

注釈: これらのパラメータを変更した後、サーバーを再起動してください。

パラメータ	値	デフォルト	説明
fs.aio-max-nu	5242	6553	これは並行に実行される非同期 I/O 操作の最大数です。 AIO サブシステムを使うワークロードが大量にある場合（例: MySQL ）、これを増やす必要があるかもしれません
fs.inotify.max_queued	1048	1638	これは対応する inotify のインスタンスにキューイングされるイベント数の上限を指定します (inotify 参照)
fs.inotify.max_user_i	1048	128	これは実ユーザー ID ごとに作成可能な inotify のインスタンス数の上限を指定します (inotify 参照)
fs.inotify.max_user_w	1048	8192	これは実ユーザー ID ごとに作成可能な watch 数の上限を指定します (inotify 参照)
kernel.dmesg_rest	1	0	この設定を有効にするとコンテナがカーネルのリングバッファー内のメッセージにアクセスするのを拒否します。この設定はホスト・システム上の非 root ユーザーへのアクセスも拒否することに注意してください
kernel.keys.maxbytes	2000	2000	非 root ユーザーが使用できる key ring の最大サイズ
kernel.keys.maxkeys	2000	200	非 root ユーザーが使用できるキーの最大数で、コンテナ数より大きくなければなりません
net.core.bpf_jit_li	1000	varies	eBPF JIT 割り当てのサイズ制限 (CONFIG_BPF_JIT_ALWAYS_ON=y でコンパイルされた< 5.15 のカーネル上では、この値は作成できるインスタンスの量を制限するかもしれません)
net.ipv4.neigh.default.gc_thresh	8192	1024	これは ARP テーブル (IPv4) 内のエントリーの最大数です。1024 個を超えるコンテナを作成するなら増やすべきです。増やさなければ ARP テーブルがフルになったときに neighbour: ndisc_cache: neighbor table overflow! というエラーが発生し、コンテナがネットワーク設定を取得できなくなります (ip-sysctl 参照)
net.ipv6.neigh.default.gc_thresh	8192	1024	これは ARP テーブル (IPv6) 内のエントリーの最大数です。1024 個を超えるコンテナを作成するなら増やすべきです。増やさなければ ARP テーブルがフルになったときに neighbour: ndisc_cache: neighbor table overflow! というエラーが発生し、コンテナがネットワーク設定を取得できなくなります (ip-sysctl 参照)
vm.max_map_co	2621	6553	このファイルはプロセスが持つメモリマップ領域の最大数を含みます。malloc の呼び出しの副作用として、直接的には mmap と mprotect によって、また、共有ライブラリーをロードすることによって、メモリマップ領域を使います

3.10.7 提供されるメトリクス

LXD は、数々のインスタンスメトリクスと内部メトリクスを提供します。これらのメトリクスの取り扱い方については、**メトリクスを監視するには**を参照してください。

インスタンスメトリクス

以下のインスタンスメトリクスが提供されています：

メトリック	説明
lxd_cpu_effective_total	使用可能な CPU の総数
lxd_cpu_seconds_total{cpu="<cpu>", mode="<mode>"}	使用された CPU 時間の合計 (秒単位)
lxd_disk_read_bytes_total{device="<dev>"}	読み出されたバイト数合計
lxd_disk_reads_completed_total{device="<dev>"}	完了した読み取り回数合計
lxd_disk_written_bytes_total{device="<dev>"}	書き込まれたバイト数合計
lxd_disk_writes_completed_total{device="<dev>"}	完了した書き込み回数合計
lxd_filesystem_avail_bytes{device="<dev>",fstype="<type>"}	利用可能な領域 (バイト単位)
lxd_filesystem_free_bytes{device="<dev>",fstype="<type>"}	空き領域 (バイト単位)
lxd_filesystem_size_bytes{device="<dev>",fstype="<type>"}	ファイルシステムのサイズ (バイト単位)
lxd_memory_Active_anon_bytes	アクティブ LRU リスト上のアノニマスメモリの量
lxd_memory_Active_bytes	アクティブ LRU リスト上のメモリの量
lxd_memory_Active_file_bytes	アクティブ LRU リスト上のディスク同期して解放で
lxd_memory_Cached_bytes	キャッシュメモリの量
lxd_memory_Dirty_bytes	ディスクへの書き戻し待ちのメモリの量
lxd_memory_HugepagesFree_bytes	hugetlb の空きメモリの量
lxd_memory_HugepagesTotal_bytes	hugetlb の使用メモリの量
lxd_memory_Inactive_anon_bytes	非アクティブ LRU リスト上のアノニマスメモリの量
lxd_memory_Inactive_bytes	非アクティブ LRU リスト上のメモリの量
lxd_memory_Inactive_file_bytes	非アクティブ LRU リスト上のディスク同期して解放
lxd_memory_Mapped_bytes	マップされたメモリの量
lxd_memory_MemAvailable_bytes	利用可能なメモリの量
lxd_memory_MemFree_bytes	空きメモリの量
lxd_memory_MemTotal_bytes	使用中メモリの量
lxd_memory_OOM_kills_total	out-of-memory で kill された回数
lxd_memory_RSS_bytes	アノニマスと swap キャッシュメモリの量
lxd_memory_Shmem_bytes	スワップアウトして解放できる (swap-backed) キャッ
lxd_memory_Swap_bytes	使用中のスワップメモリの量
lxd_memory_Unevictable_bytes	回収不可のメモリの使用量
lxd_memory_Writeback_bytes	ディスクへの同期のためにキューに入れられているメ

表 3 – 前のページからの続き

メトリック	説明
lxd_network_receive_bytes_total{device="<dev>"}	指定のインタフェース上の受信したバイト数
lxd_network_receive_drop_total{device="<dev>"}	指定のインタフェース上の受信でドロップしたバイト
lxd_network_receive_errs_total{device="<dev>"}	指定のインタフェース上の受信エラー数
lxd_network_receive_packets_total{device="<dev>"}	指定のインタフェース上の受信パケット数
lxd_network_transmit_bytes_total{device="<dev>"}	指定のインタフェース上の送信したバイト数
lxd_network_transmit_drop_total{device="<dev>"}	指定のインタフェース上の送信でドロップしたバイト
lxd_network_transmit_errs_total{device="<dev>"}	指定のインタフェース上の送信エラー数
lxd_network_transmit_packets_total{device="<dev>"}	指定のインタフェース上の送信パケット数
lxd_procs_total	稼働中のプロセス数

内部メトリクス

以下の内部メトリクスが提供されています：

メトリック	説明
lxd_go_alloc_bytes_total	割り当てられた (その後の解放された分も含む) バイト数累計
lxd_go_alloc_bytes	割り当てられ使用中のバイト数
lxd_go_buck_hash_sys_bytes	プロファイルのバケットハッシュテーブルで使用されたバイト数
lxd_go_frees_total	解放の合計回数
lxd_go_gc_sys_bytes	システムメタデータのガベージコレクションで使用されたバイト数
lxd_go_goroutines	現在存在する goroutine 数
lxd_go_heap_alloc_bytes	割り当てられ使用中のヒープのバイト数
lxd_go_heap_idle_bytes	使用を待っているヒープのバイト数
lxd_go_heap_inuse_bytes	使用中のヒープのバイト数
lxd_go_heap_objects	割り当てられたオブジェクト数
lxd_go_heap_released_bytes	OS に開放されたヒープのバイト数
lxd_go_heap_sys_bytes	システムから取得されたヒープのバイト数
lxd_go_lookups_total	ポインタルックアップの合計回数
lxd_go_mallocs_total	mallocs の合計回数
lxd_go_mcache_inuse_bytes	mcache 構造で使用されるバイト数
lxd_go_mcache_sys_bytes	システムから取得された mcache 構造で使用されるバイト数
lxd_go_mspan_inuse_bytes	mspan 構造で使用されるバイト数
lxd_go_mspan_sys_bytes	システムから取得された mspan 構造で使用されるバイト数
lxd_go_next_gc_bytes	次のガベージコレクションが発生する際のヒープのバイト数
lxd_go_other_sys_bytes	他のシステム割当に使用されるバイト数
lxd_go_stack_inuse_bytes	スタックアロケータに使用されるバイト数
lxd_go_stack_sys_bytes	スタックアロケータ用にシステムから取得されたバイト数
lxd_go_sys_bytes	システムから取得されたバイト数
lxd_operations_total	実行中の処理の数
lxd_uptime_seconds	デーモンの uptime(秒単位)
lxd_warnings_total	アクティブな警告の数

3.11 マイグレーション

LXD は異なる状況でインスタンスをマイグレートするためのツールと機能を提供します。

サーバー間で既存の LXD インスタンスをマイグレートする 最も基本的な種類のマイグレーションは 1 つのサーバー上に LXD インスタンスがあり、それを別の LXD サーバーに移動したいというものです。仮想マシンでは、ライブマイグレーションを行えます。これは稼働中にダウンタイムなしで VM をマイグレートできることを意味します。

詳細は **サーバー間で既存の *LXD* インスタンスを移動するには** を参照してくだい。

物理または仮想マシンを LXD インスタンスにマイグレートする

物理または仮想 (VM またはコンテナ) の既存のマシンがある場合、既存のマシン上に LXD インスタンスを作成するために `lxd-migrate` ツールが使えます。このツールは提供されたパーティション、ディスクやイメージを LXD サーバーの LXD ストレージプールにコピーし、そのストレージを使ってインスタンスをセットアップします。新しいインスタンスの追加の設定を行うこともできます。

詳細は **物理または仮想マシンを *LXD* インスタンスにインポートするには** を参照してください。

LXD から LXD へインスタンスをマイグレートする

LXC を使っていて全てまたは一部の LXC コンテナを同じマシン上の LXD に移動したい場合、`lxc-to-lxd` ツールが使えます。このツールは LXC 設定を解析し、既存の LXC コンテナのデータと設定を新しい LXD コンテナにコピーします。

詳細は ***LXC* から *LXD* にコンテナをマイグレートするには** を参照してください。

3.11.1 サーバー間で既存の LXD インスタンスを移動するには

ある LXD サーバーから別のサーバーへインスタンスを移動するには `lxc move` コマンドを使います。

```
lxc move [<source_remote>:]<source_instance_name> <target_remote>:[<target_instance_name>
↪]
```

注釈: コンテナを移動する際にはまず停止する必要があります。詳細は **コンテナのライブマイグレーション** を参照してください。

仮想マシンを移動する際は、**仮想マシンのライブマイグレーション** を有効にするか、まず仮想マシンを停止する必要があります。

移動元のリモートがデフォルトのリモートの場合は省略可能で、移動先でも同じインスタンス名を使用する場合は移動先インスタンス名は省略できます。インスタンスを特定のクラスタメンバーに移動したい場合は、`--target` フラグを指定してください。この場合、移動元と移動先のリモートは指定を省略してください。

ネットワークのセットアップに応じて、`--mode` フラグを追加して転送モードを選択できます。

`pull` (**デフォルト**)

移動先のサーバーに、移動元のサーバーへ接続させ該当のインスタンスをプルするように指示します。

`push`

移動元のサーバーに、移動先のサーバーへ接続させインスタンスをプッシュするように指示します。

`relay`

クライアントに移動元と移動先の両方に接続させデータをクライアント経由で転送するよう指示します。

移動先のサーバー上でインスタンスを動かすように設定を調整する必要がある場合、(--config, --device, --storage, --target-project を使用して) 設定を直接指定するか、(--no-profiles か --profile を使って) プロファイルを経由して指定できます。全ての利用可能なフラグについては lxc move --help を参照してください。

ライブマイグレーション

ライブマイグレーションとはインスタンスの稼働中にマイグレートするという意味です。仮想マシンではフルにサポートされています。コンテナでは限定的にサポートされています。

仮想マシンのライブマイグレーション

仮想マシンは稼働したまま、つまり一切のダウンタイムなしで、別のサーバーに移動できます。

ライブマイグレーションを可能にするには、ステートフルマイグレーションのサポートを有効にする必要があります。そのためには、以下の設定を確認してください。

- インスタンスの *migration.stateful* を true に設定する。
- 仮想マシンのルートディスクデバイスの *size.state* を少なくとも仮想マシンの *limits.memory* 設定のサイズに設定する。

コンテナのライブマイグレーション

コンテナについては CRIU (Checkpoint/Restore in Userspace) を使用したライブマイグレーションが限定的にサポートされています。しかし、広範囲に及ぶカーネルへの依存のため、非常にベーシックなコンテナ (ネットワークデバイスなしの非 systemd コンテナ) のみが安定してマイグレートできます。ほとんどの実世界でのシナリオでは、コンテナを停止、移動してその後起動するのが良いです。

コンテナのライブマイグレーションを使用したい場合、マイグレーション元と先の両方のサーバーで CRIU を有効にする必要があります。snap をお使いの場合、以下のコマンドを使用して CRIU を有効にしてください。

```
snap set lxd criu.enable=true
systemctl reload snap.lxd.daemon
```

それ以外の場合、両方のシステムに CRIU がインストールされていることを確認してください。

コンテナのメモリ転送を最適化するには *migration.incremental.memory* プロパティを true に設定して CRIU の事前コピー機能を使用してください。この設定では LXD はコンテナの一連のメモリダンプを実行するよう CRIU に指示します。それぞれのダンプの後、 LXD はメモリダンプを指定されたリモートに送信します。理想的なシナリオでは、各メモリダンプを前のメモリダンプとの差分にまで減らし、それによりすでに同期されたメモリの割合を増やします。同期されたメモリの割合が *migration.incremental.memory.goal* で設定した閾値と等

しいか超えた場合、あるいは *migration.incremental.memory.iterations* で指定された許容される繰り返し回数の最大値に達した場合、 LXD は CRIU に最終的なメモリダンプを実行し、転送するように要求します。

3.11.2 物理または仮想マシンを LXD インスタンスにインポートするには

LXD は既存のディスクやイメージに基づく LXD インスタンスを作成するツール (`lxd-migrate`) を提供しています。

このツールは Linux マシン上で実行できます。まず LXD サーバーに接続して空のインスタンスを作成します。このインスタンスはマイグレーション中またはマイグレーション後に設定を変更できます。次にこのツールはあなたが用意したディスクまたはイメージからインスタンスにデータをコピーします。

このツールはコンテナと仮想マシンの両方を作成できます。

- コンテナを作成する際は、コンテナのルートファイルシステムを含むディスクまたはパーティションを用意する必要があります。例えば、これはあなたがツールを実行しているマシンまたはコンテナの `/` ルートディスクかもしれません。
- 仮想マシンを作成する際は、起動可能なディスク、パーティション、またはイメージを用意する必要があります。これは単にファイルシステムを用意するだけでは不十分であり、実行中のコンテナから仮想マシンを作成することはできないことを意味します。また使用中の物理マシンから仮想マシンを作成することもできません。これはマイグレーションツールがコピーしようとするディスクを使用するからです。代わりに、起動可能なディスク、起動可能なパーティション、または現在使用中でないディスクを用意してください。

既存のマシンを LXD インスタンスにマイグレートするには以下の手順を実行してください。

1. 最新の LXD release の **Assets** セクションから `bin.linux.lxd-migrate` ツールをダウンロードしてください。
2. ツールをインスタンスを作成したいマシン上に配置して (通常 `chmod u+x bin.linux.lxd-migrate` を実行して) 実行可能にしてください。
3. マシンに `rsync` がインストールされているか確認してください。インストールされていない場合は (例えば `sudo apt install rsync` で) インストールしてください。
4. 以下のようにツールを実行します。

   ```
   ./bin.linux.lxd-migrate
   ```

   ツールはマイグレーソンに必要な情報を入力するようプロンプトを出します。

   Tip: ツールをインタラクティブに実行する代わりの方法として、設定をパラメータでコマンドに指定することもできます。詳細は `./bin.linux.lxd-migrate --help` を参照してください。

1. LXD サーバーの URL を、 IP アドレスまたは DNS 名で指定してください。
2. 証明書のフィンガープリントを確認してください。
3. 認証の方法を選択してください (**リモート** *API* **認証** 参照)。

   例えば、証明書トークンを選ぶ場合、 LXD サーバーにログオンしてマイグレーションツールを実行中のマシン用のトークンを `lxc config trust add` で作成してください。次に生成されたトークンを、ツールを認証するのに使用してください。
4. コンテナと仮想マシンのどちらを作成するか選択してください。
5. 作成するインスタンスの名前を指定してください。
6. ルートファイルシステム (コンテナの場合)、起動可能なディスク、パーティションまたはイメージファイル (仮想マシンの場合) のパスを指定します。
7. コンテナの場合、必要に応じてファイルシステムのマウントを追加します。
8. 仮想マシンの場合、セキュアブートがサポートされているかを指定します。
9. 任意で、新しいインスタンスを設定します。プロファイルを指定するか、オプションやストレージを変更したりネットワークを設定する設定オプションを直接指定できます。

   あるいは、マイグレーション後に新しいインスタンスを設定することもできます。
10. マイグレーションの設定が完了したら、マイグレーションプロセスを開始します。

```
user@host:~$ ./bin.linux.lxd-migrate   Please provide LXD server URL: https://192.0.
2.7:8443Certificate fingerprint:  xxxxxxxxxxxxxxxxok (y/n)? y 1) Use a certificate
token2) Use an existing TLS authentication certificate3) Generate a temporary TLS
authentication certificatePlease pick an authentication mechanism above:  1Please
provide the certificate token:  xxxxxxxxxxxxxxxx Remote LXD server:  Hostname:  bar
Version:  5.4 Would you like to create a container (1) or virtual-machine (2)?:
1Name of the new instance:  fooPlease provide the path to a root filesystem:  /Do
you want to add additional filesystem mounts? [default=no]:  Instance to be created:
Name:  foo Project:  default Type:  container Source:  /  Additional overrides can
be applied at this stage:1) Begin the migration with the above configuration2)
Override profile list3) Set additional configuration options4) Change instance
storage pool or volume size5) Change instance network Please pick one of the options
above [default=1]:  3Please specify config keys and values (key=value ...):  limits.
cpu=2 Instance to be created:  Name:  foo Project:  default Type:  container Source:
/  Config:  limits.cpu:  "2" Additional overrides can be applied at this stage:1)
Begin the migration with the above configuration2) Override profile list3) Set
additional configuration options4) Change instance storage pool or volume size5)
Change instance network Please pick one of the options above [default=1]:  4Please
provide the storage pool to use:  defaultDo you want to change the storage size?
```

```
[default=no]:  yesPlease specify the storage size:  20GiB Instance to be created:
Name:  foo Project:  default Type:  container Source:  /  Storage pool:  default
Storage pool size:  20GiB Config:  limits.cpu:  "2" Additional overrides can be
applied at this stage:1) Begin the migration with the above configuration2) Override
profile list3) Set additional configuration options4) Change instance storage
pool or volume size5) Change instance network Please pick one of the options above
[default=1]:  5Please specify the network to use for the instance:  lxdbr0 Instance
to be created:  Name:  foo Project:  default Type:  container Source:  /  Storage
pool:  default Storage pool size:  20GiB Network name:  lxdbr0 Config:  limits.
cpu:  "2" Additional overrides can be applied at this stage:1) Begin the migration
with the above configuration2) Override profile list3) Set additional configuration
options4) Change instance storage pool or volume size5) Change instance network
Please pick one of the options above [default=1]:  1Instance foo successfully
created
```

5. マイグレーションが完了したら、新しいインスタンスをチェックし、設定を新しい環境にあわせて更新してください。通常は、少なくともストレージ設定 (`/etc/fstab`) とネットワーク設定を更新する必要があります。

3.11.3 LXC から LXD にコンテナをマイグレートするには

LXD は LXC のコンテナを LXD サーバーにインポートするためのツール (`lxc-to-lxd`) を提供しています。LXC コンテナは LXD サーバーと同じマシン上に存在する必要があります。

このツールは LXC コンテナを分析し、データと設定の両方を新しい LXD コンテナにマイグレートします。

注釈: あるいは LXC コンテナ内で `lxd-migrate` ツールを使用して LXD にマイグレートすることもできます (**物理または仮想マシンを *LXD* インスタンスにインポートするには** 参照)。しかし、このツールは LXC コンテナの設定は一切マイグレートしません。

ツールを取得する

snap をお使いの場合、`lxc-to-lxd` は自動でインストールされます。`lxd.lxc-to-lxd` で利用できます。

そうでない場合、 `go` (バージョン 1.18 以降) がインストールされていることを確認の上、以下のコマンドでツールをインストールしてください。

```
go install github.com/lxc/lxd/lxc-to-lxd@latest
```

LXC コンテナを用意する

一度に 1 つのコンテナをマイグレートすることもできますし、同時にあなたの全ての LXC コンテナをマイグレートすることもできます。

注釈: マイグレートされたコンテナは元のコンテナと同じ名前を使用します。LXD にインスタンス名としてすでに存在する名前を持つコンテナをマイグレートすることはできません。

このため、マイグレーションプロセスを開始する前に名前の衝突を引き起こす可能性のある LXC コンテナはリネームしてください。

マイグレーションプロセスを開始する前に、マイグレートしたいコンテナを停止してください。

マイグレーションプロセスを開始する

コンテナをマイグレートするには `sudo lxd.lxc-to-lxd [flags]` と実行してください。(このコマンドはあなたが snap を使用していると想定しています。そうでない場合 `lxd.lxc-to-lxd` を `lxc-to-lxd` と読み替えてください。以下の例でも同様です)

例えば、全てのコンテナをマイグレートするには

```
sudo lxd.lxc-to-lxd --all
```

`lxc1` コンテナだけをマイグレートするには

```
sudo lxd.lxc-to-lxd --containers lxc1
```

2 つのコンテナ (`lxc1` と `lxc2`) をマイグレートし LXD 内の `my-storage` ストレージプールを使用するには

```
sudo lxd.lxc-to-lxd --containers lxc1,lxc2 --storage my-storage
```

実際に実行せずに全てのコンテナのマイグレートをテストするには

```
sudo lxd.lxc-to-lxd --all --dry-run
```

全てのコンテナをマイグレートするが、`rsync` の帯域幅を 5000 KB/s に限定するには

```
sudo lxd.lxc-to-lxd --all --rsync-args --bwlimit=5000
```

全ての利用可能なフラグを確認するには `sudo lxd.lxc-to-lxd --help` と実行してください。

注釈: `linux64` アーキテクチャがサポートされない (`linux64` architecture isn't supported) というエラーが出る場

合、ツールを最新版にアップデートするか LXC コンテナ内のアーキテクチャを `linux64` から `amd64` か `x86_64` に変更してください。

設定を確認する

このツールは LXC の設定と (1 つまたは複数の) コンテナの設定を分析し、可能な限りの範囲で設定をマイグレートします。以下のような実行結果が出力されます。

```
user@host:~$ sudo lxd.lxc-to-lxd --containers lxc1       Parsing LXC configurationChecking
for unsupported LXC configuration keysChecking for existing containersChecking whether
container has already been migratedValidating whether incomplete AppArmor support
is enabledValidating whether mounting a minimal /dev is enabledValidating container
rootfsProcessing network configurationProcessing storage configurationProcessing
environment configurationProcessing container boot configurationProcessing container
apparmor configurationProcessing container seccomp configurationProcessing container
SELinux configurationProcessing container capabilities configurationProcessing container
architecture configurationCreating containerTransferring container:  lxc1:  ...Container
'lxc1' successfully created
```

マイグレーションプロセスが完了したら、設定を確認し、必要に応じて、マイグレートした LXD コンテナを起動する前に LXD 内の設定を更新してください。

3.12 REST API

3.12.1 REST API

LXD とクライアント間の全ての通信は HTTP 上の RESTful API を使用します。この API や (リモートの通信では)TLS あるいは (ローカルの操作では)Unix ソケットで通信します。

どのようにリモートの API にアクセスするかについての情報は**リモート** *API* **認証**を参照してください。

Tip:

- どのように API が使われるかの例を見るには LXD クライアント (`lxc`) で`--debug` フラグを追加してコマンドを実行してください。デバッグ情報が API の呼び出しと戻り値を表示します。
- 手軽にに API を呼び出せるように、LXD クライアントは `lxc query` コマンドを提供しています。

API のバージョニング

サポートされている API のメジャーバージョンのリストは `GET /` を使って取得できます。

後方互換性を壊す場合は API のメジャーバージョンが上がります。

後方互換性を壊さずに追加される機能は `api_extensions` の追加という形になり、特定の機能がサーバーでサポートされているかクライアントがチェックすることで利用できます。

戻り値

次の 3 つの標準的な戻り値の型があります。

- 標準の戻り値
- バックグラウンド操作
- エラー

標準の戻り値

標準の同期的な操作に対しては以下のような dict が返されます。

```
{
    "type": "sync",
    "status": "Success",
    "status_code": 200,
    "metadata": {}                          // リソースやアクションに固有な追加のメタデータ
}
```

HTTP ステータスコードは必ず 200 です。

バックグラウンド操作

リクエストの結果がバックグラウンド操作になる場合、 HTTP ステータスコードは 202 (Accepted) になり、操作の URL を指す HTTP の Location ヘッダが返されます。

レスポンスボディは以下のような構造を持つ dict です。

```
{
    "type": "async",
    "status": "OK",
    "status_code": 100,
    "operation": "/1.0/instances/<id>",                    // バックグラウンド操作の URL
```

(次のページに続く)

(前のページからの続き)

```
    "metadata": {}                                          // 操作のメタデータ (下記参照)
}
```

操作のメタデータの構造は以下のようになります。

```
{
    "id": "a40f5541-5e98-454f-b3b6-8a51ef5dbd3c",           // 操作の UUID
    "class": "websocket",                                   // 操作の種別 (task, websocket,
 token のいずれか)
    "created_at": "2015-11-17T22:32:02.226176091-05:00",    // 操作の作成日時
    "updated_at": "2015-11-17T22:32:02.226176091-05:00",    // 操作の最終更新日時
    "status": "Running",                                    // 文字列表記での操作の状態
    "status_code": 103,                                     // 整数表記での操作の状態␣
↪(status ではなくこちらを利用してください。訳注: 詳しくは下記のステータスコードの項を参照)
    "resources": {                                          // リソース種別 (container,␣
↪snapshots, images のいずれか) の dict を影響を受けるリソース
      "containers": [
        "/1.0/instances/test"
      ]
    },
    "metadata": {                                           // 対象となっている (この例では␣
↪exec) 操作に固有なメタデータ
      "fds": {
        "0": "2a4a97af81529f6608dca31f03a7b7e47acc0b8dc6514496eb25e325f9e4fa6a",
        "control": "5b64c661ef313b423b5317ba9cb6410e40b705806c28255f601c0ef603f079a7"
      }
    },
    "may_cancel": false,                                    // (REST で DELETE を使用して)␣
↪操作がキャンセル可能かどうか
    "err": ""                                               // 操作が失敗した場合にエラー文字
列が設定されます
}
```

対象の操作に対して追加のリクエストを送って情報を取り出さなくても、何が起こっているかユーザーにとってわかりやすい形でボディは構成されています。ボディに含まれる全ての情報はバックグラウンド操作の URL から取得することもできます。

エラー

さまざまな状況によっては操作を行う前に直ぐに問題が起きる場合があり、そういう場合には以下のような値が返されます。

```
{
    "type": "error",
    "error": "Failure",
    "error_code": 400,
    "metadata": {}                    // エラーについてのさらなる詳細
}
```

HTTP ステータスコードは 400, 401, 403, 404, 409, 412, 500 のいずれかです。

ステータスコード

LXD REST API はステータス情報を返す必要があります。それはエラーの理由だったり、操作の現在の状態だったり、 LXD が提供する様々なリソースの状態だったりします。

デバッグをシンプルにするため、ステータスは常に文字列表記と整数表記で重複して返されます。ステータスの整数表記の値は将来に渡って不変なので API クライアントが個々の値に依存できます。文字列表記のステータスは人間が API を手動で実行したときに何が起きているかをより簡単に判断できるように用意されています。

ほとんどのケースでこれらは `status` と `status_code` と呼ばれ、前者はユーザーフレンドリーな文字列表記で後者は固定の数値です。

整数表記のコードは常に 3 桁の数字で以下の範囲の値となっています。

- 100 to 199: リソースの状態 (started, stopped, ready, ...)
- 200 to 399: 成功したアクションの結果
- 400 to 599: 失敗したアクションの結果
- 600 to 999: 将来使用するために予約されている番号の範囲

現在使用されているステータスコード一覧

コード	意味
100	操作が作成された
101	開始された
102	停止された
103	実行中
104	キャンセル中
105	ペンディング
106	開始中
107	停止中
108	中断中
109	凍結中
110	凍結された
111	解凍された
112	エラー
113	準備完了
200	成功
400	失敗
401	キャンセルされた

再帰

巨大な一覧のクエリを最適化するために、コレクションに対して再帰が実装されています。コレクションに対するクエリの GET リクエストに `recursion` パラメータを指定できます。

デフォルト値は 0 でコレクションのメンバーの URL が返されることを意味します。1 を指定するとこれらの URL がそれが指すオブジェクト (通常は dict 形式) で置き換えられます。

再帰はジョブへのポインタ (URL) をオブジェクトそのもので単に置き換えるように実装されています。

フィルタ

検索結果をある値でフィルタするために、コレクションにフィルタが実装されています。コレクションに対する GET クエリに `filter` 引数を渡せます。

フィルタはインスタンス、イメージ、ストレージボリュームのエンドポイントに提供されています。

フィルタにはデフォルト値はありません。これは見つかった全ての結果が返されることを意味します。フィルタの引数には以下のような言語を設定します。

```
?filter=field_name eq desired_field_assignment
```

この言語は REST API のフィルタロジックを構成するための OData の慣習に従います。フィルタは下記の論理演算子もサポートします。not(not), equals(eq), not equals(ne), and(and), or(or) フィルタは左結合で評価されます。空白を含む値はクォートで囲むことができます。ネストしたフィルタもサポートされます。例えば設定内のフィールドに対してフィルタするには以下のように指定します。

```
?filter=config.field_name eq desired_field_assignment
```

device の属性についてフィルタするには以下のように指定します。

```
?filter=devices.device_name.field_name eq desired_field_assignment
```

以下に上記の異なるフィルタの方法を含む GET クエリをいくつか示します。

```
containers?filter=name eq "my container" and status eq Running

containers?filter=config.image.os eq ubuntu or devices.eth0.nictype eq bridged

images?filter=Properties.os eq Centos and not UpdateSource.Protocol eq simplestreams
```

非同期操作

完了までに 1 秒以上かかるかもしれない操作はバックグラウンドで実行しなければなりません。そしてクライアントにはバックグラウンド操作 ID を返します。

クライアントは操作のステータス更新をポーリングするか long-poll API を使って通知を待つことが出来ます。

通知

通知のために WebSocket ベースの API が利用できます。クライアントへ送られるトラフィックを制限するためにいくつかの異なる通知種別が存在します。

リモート操作の状態をポーリングしなくて済むように、リモート操作を開始する前に操作の通知をクライアントが常に購読しておくのがお勧めです。

PUT と PATCH の使い分け

LXD API は既存のオブジェクトを変更するのに PUT と PATCH の両方をサポートします。

PUT はオブジェクト全体を新しい定義で置き換えます。典型的には GET で現在のオブジェクトの状態を取得した後に PUT が呼ばれます。

レースコンディションを避けるため、 GET のレスポンスから ETag ヘッダを読み取り PUT リクエストの If-Match ヘッダに設定するべきです。こうしておけば GET と PUT の間にオブジェクトが他から変更されていた場合は更新が失敗するようになります。

PATCH は変更したいプロパティだけを指定することでオブジェクト内の単一のフィールドを変更するのに用いられます。キーを削除するには通常は空の値を設定すれば良いようになっていますが、 PATCH ではキーの削除は出来ず、代わりに PUT を使う必要がある場合もあります。

インスタンス、コンテナと仮想マシン

このドキュメントでは`/1.0/instances/...` のようなパスを示します。これらはかなり新しく、仮想マシンがサポートされた LXD 3.19 で導入されました。コンテナのみをサポートし仮想マシンはサポートしない古いリリースでは全く同じ API を`/1.0/containers/...` で利用します。

後方互換性の理由で LXD は `/1.0/containers` API を引き続き公開しサポートしますが、簡潔さのため以下では両方をドキュメントはしないことにしました。

`/1.0/virtual-machines` に追加のエンドポイントも存在し、 `/1.0/containers` とほぼ同様ですが、仮想マシンのタイプのインスタンスのみを表示します。

API 構造

LXD は API エンドポイントを記述する Swagger 仕様を自動生成しています。この API 仕様の YAML 版が `rest-api.yaml` にあります。手軽にウェブで見る場合は `https://linuxcontainers.org/lxd/api/master/` を参照してください。

3.12.2 API 仕様

3.12.3 API 拡張

それらの変更は全て後方互換であり、 `GET /1.0` の `api_extensions` を見ることでクライアントツールにより検出可能です。

storage_zfs_remove_snapshots

storage.zfs_remove_snapshots というデーモン設定キーが導入されました。

値の型は Boolean でデフォルトは false です。 true にセットすると、スナップショットを復元しようとするときに必要なスナップショットを全て削除するように LXD に指示します。

ZFS でスナップショットの復元が出来るのは最新のスナップショットに限られるので、この対応が必要になります。

container_host_shutdown_timeout

boot.host_shutdown_timeout というコンテナ設定キーが導入されました。

値の型は integer でコンテナを停止しようとした後 kill するまでどれだけ待つかを LXD に指示します。

この値は LXD デーモンのクリーンなシャットダウンのときにのみ使用されます。デフォルトは 30s です。

container_stop_priority

boot.stop.priority というコンテナ設定キーが導入されました。

値の型は integer でシャットダウン時のコンテナの優先度を指示します。

コンテナは優先度レベルの高いものからシャットダウンを開始します。

同じ優先度のコンテナは並列にシャットダウンします。デフォルトは 0 です。

container_syscall_filtering

コンテナ設定キーに関するいくつかの新しい syscall が導入されました。

- security.syscalls.blacklist_default
- security.syscalls.blacklist_compat
- security.syscalls.blacklist
- security.syscalls.whitelist

使い方は **インスタンスの設定** を参照してください。

auth_pki

これは PKI 認証モードのサポートを指示します。

このモードではクライアントとサーバーは同じ PKI によって発行された証明書を使わなければなりません。

詳細は **セキュリティ** を参照してください。

container_last_used_at

`GET /1.0/containers/<name>` エンドポイントに `last_used_at` フィールドが追加されました。

これはコンテナが開始した最新の時刻のタイムスタンプです。

コンテナが作成されたが開始はされていない場合は `last_used_at` フィールドは `1970-01-01T00:00:00Z` になります。

etag

関連性のある全てのエンドポイントに ETag ヘッダのサポートが追加されました。

この変更により GET のレスポンスに次の HTTP ヘッダが追加されます。

- ETag (ユーザーが変更可能なコンテンツの SHA-256)

また PUT リクエストに次の HTTP ヘッダのサポートが追加されます。

- If-Match (前回の GET で得られた ETag の値を指定)

これにより GET で LXD のオブジェクトを取得して PUT で変更する際に、レースコンディションになったり、途中で別のクライアントがオブジェクトを変更していた (訳注: のを上書きしてしまう) というリスク無しに PUT で変更できるようになります。

patch

HTTP の PATCH メソッドのサポートを追加します。

PUT の代わりに PATCH を使うとオブジェクトの部分的な変更が出来ます。

usb_devices

USB ホットプラグのサポートを追加します。

https_allowed_credentials

LXD API を全てのウェブブラウザで (SPA 経由で) 使用するには、 XHR の度に認証情報を送る必要があります。それぞれの XHR リクエストで `withCredentials=true` とセットします。

Firefox や Safari などいくつかのブラウザは `Access-Control-Allow-Credentials: true` ヘッダがないレスポンスを受け入れることができません。サーバーがこのヘッダ付きのレスポンスを返すことを保証するには `core.https_allowed_credentials=true` と設定してください。

image_compression_algorithm

この変更はイメージを作成する時 (`POST /1.0/images`) に `compression_algorithm` というプロパティのサポートを追加します。

このプロパティを設定するとサーバーのデフォルト値 (`images.compression_algorithm`) をオーバーライドします。

directory_manipulation

LXD API 経由でディレクトリを作成したり一覧したりでき、ファイルタイプを X-LXD-type ヘッダに付与するようになります。現状はファイルタイプは `file` か `directory` のいずれかです。

container_cpu_time

この拡張により実行中のコンテナの CPU 時間を取得できます。

storage_zfs_use_refquota

この拡張により新しいサーバープロパティ `storage.zfs_use_refquota` が追加されます。これはコンテナにサイズ制限を設定する際に `quota` の代わりに `refquota` を設定するように LXD に指示します。また LXD はディスク使用量を調べる際に `used` の代わりに `usedbydataset` を使うようになります。

これはスナップショットによるディスク消費をコンテナのディスク利用の一部とみなすかどうかを実質的に切り替えることになります。

storage_lvm_mount_options

この拡張は storage.lvm_mount_options という新しいデーモン設定オプションを追加します。デフォルト値は discard で、このオプションにより LVM LV で使用するファイルシステムの追加マウントオプションをユーザーが指定できるようになります。

network

LXD のネットワーク管理 API 。

次のものを含みます。

- /1.0/networks エントリーに managed プロパティを追加
- ネットワーク設定オプションの全て (詳細は **ネットワーク設定** を参照)
- POST /1.0/networks (詳細は *RESTful API* を参照)
- PUT /1.0/networks/<entry> (詳細は *RESTful API* を参照)
- PATCH /1.0/networks/<entry> (詳細は *RESTful API* を参照)
- DELETE /1.0/networks/<entry> (詳細は *RESTful API* を参照)
- nic タイプのデバイスの ipv4.address プロパティ (nictype が bridged の場合)
- nic タイプのデバイスの ipv6.address プロパティ (nictype が bridged の場合)
- nic タイプのデバイスの security.mac_filtering プロパティ (nictype が bridged の場合)

profile_usedby

プロファイルを使用しているコンテナをプロファイルエントリーの一覧の used_by フィールドとして新たに追加します。

container_push

コンテナが push モードで作成される時、クライアントは作成元と作成先のサーバー間のプロキシとして機能します。作成先のサーバーが NAT やファイアウォールの後ろにいて作成元のサーバーと直接通信できず pull モードで作成できないときにこれは便利です。

container_exec_recording

新しい Boolean 型の record-output を導入します。これは /1.0/containers/<name>/exec のパラメータでこれを true に設定し wait-for-websocket を false に設定すると標準出力と標準エラー出力をディスクに保存し logs インタフェース経由で利用可能にします。

記録された出力の URL はコマンドが実行完了したら操作のメタデータに含まれます。

出力は他のログファイルと同様に、通常は 48 時間後に期限切れになります。

certificate_update

REST API に次のものを追加します。

- 証明書の GET に ETag ヘッダ
- 証明書エントリーの PUT
- 証明書エントリーの PATCH

container_exec_signal_handling

クライアントに送られたシグナルをコンテナ内で実行中のプロセスにフォワーディングするサポートを /1.0/containers/<name>/exec に追加します。現状では SIGTERM と SIGHUP がフォワードされます。フォワード出来るシグナルは今後さらに追加されるかもしれません。

gpu_devices

コンテナに GPU を追加できるようにします。

container_image_properties

設定キー空間に新しく image を導入します。これは読み取り専用で、親のイメージのプロパティを含みます。

migration_progress

転送の進捗が操作の一部として送信側と受信側の両方に公開されます。これは操作のメタデータの fs_progress 属性として現れます。

id_map

security.idmap.isolated, security.idmap.isolated, security.idmap.size, raw.id_map のフィールドを設定できるようにします。

network_firewall_filtering

ipv4.firewall と ipv6.firewall という 2 つのキーを追加します。false に設置すると iptables の FORWARDING ルールの生成をしないようになります。 NAT ルールは対応する ipv4.nat や ipv6.nat キーが true に設定されている限り引き続き追加されます。

ブリッジに対して dnsmasq が有効な場合、 dnsmasq が機能する (DHCP/DNS) ために必要なルールは常に適用されます。

network_routes

ipv4.routes と ipv6.routes を導入します。これらは LXD ブリッジに追加のサブネットをルーティングできるようにします。

storage

LXD のストレージ管理 API 。

これは次のものを含みます。

- GET /1.0/storage-pools
- POST /1.0/storage-pools (詳細は *RESTful API* を参照)
- GET /1.0/storage-pools/<name> (詳細は *RESTful API* を参照)
- POST /1.0/storage-pools/<name> (詳細は *RESTful API* を参照)
- PUT /1.0/storage-pools/<name> (詳細は *RESTful API* を参照)
- PATCH /1.0/storage-pools/<name> (詳細は *RESTful API* を参照)
- DELETE /1.0/storage-pools/<name> (詳細は *RESTful API* を参照)
- GET /1.0/storage-pools/<name>/volumes (詳細は *RESTful API* を参照)
- GET /1.0/storage-pools/<name>/volumes/<volume_type> (詳細は *RESTful API* を参照)
- POST /1.0/storage-pools/<name>/volumes/<volume_type> (詳細は *RESTful API* を参照)
- GET /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (詳細は *RESTful API* を参照)
- POST /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (詳細は *RESTful API* を参照)

- PUT /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (詳細は *RESTful API* を参照)
- PATCH /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (詳細は *RESTful API* を参照)
- DELETE /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (詳細は *RESTful API* を参照)
- 全てのストレージ設定オプション (詳細は **ストレージの設定** を参照)

file_delete

/1.0/containers/<name>/files の DELETE メソッドを実装

file_append

X-LXD-write ヘッダを実装しました。値は overwrite か append のいずれかです。

network_dhcp_expiry

ipv4.dhcp.expiry と ipv6.dhcp.expiry を導入します。 DHCP のリース期限を設定できるようにします。

storage_lvm_vg_rename

storage.lvm.vg_name を設定することでボリュームグループをリネームできるようにします。

storage_lvm_thinpool_rename

storage.thinpool_name を設定することで thin pool をリネームできるようにします。

network_vlan

macvlan ネットワークデバイスに vlan プロパティを新たに追加します。

これを設定すると、指定した VLAN にアタッチするように LXD に指示します。LXD はホスト上でその VLAN を持つ既存のインタフェースを探します。もし見つからない場合は LXD がインタフェースを作成して macvlan の親として使用します。

image_create_aliases

POST /1.0/images に aliases フィールドを新たに追加します。イメージの作成／インポート時にエイリアスを設定できるようになります。

container_stateless_copy

POST /1.0/containers/<name> に live という属性を新たに導入します。false に設定すると、実行状態を転送しようとしないように LXD に伝えます。

container_only_migration

container_only という Boolean 型の属性を導入します。 true に設定するとコンテナだけがコピーや移動されるようになります。

storage_zfs_clone_copy

ZFS ストレージプールに storage_zfs_clone_copy という Boolean 型のプロパティを導入します。false に設定すると、コンテナのコピーは zfs send と receive 経由で行われるようになります。これにより作成先のコンテナは作成元のコンテナに依存しないようになり、 ZFS プールに依存するスナップショットを維持する必要がなくなります。しかし、これは影響するプールのストレージの使用状況が以前より非効率的になるという結果を伴います。このプロパティのデフォルト値は true です。つまり明示的に false に設定しない限り、空間効率の良いスナップショットが使われます。

unix_device_rename

path を設定することによりコンテナ内部で unix-block/unix-char デバイスをリネームできるようにし、ホスト上のデバイスを指定する source 属性が追加されます。path を設定せずに source を設定すると、 path は source と同じものとして扱います。 source や major/minor を設定せずに path を設定すると source は path と同じものとして扱います。ですので、最低どちらか 1 つは設定しなければなりません。

storage_rsync_bwlimit

ストレージエンティティを転送するために rsync が起動される場合に rsync.bwlimit を設定すると使用できるソケット I/O の量に上限を設定します。

network_vxlan_interface

ネットワークに tunnel.NAME.interface オプションを新たに導入します。

このキーは VXLAN トンネルにホストのどのネットワークインタフェースを使うかを制御します。

storage_btrfs_mount_options

Btrfs ストレージプールに btrfs.mount_options プロパティを導入します。

このキーは Btrfs ストレージプールに使われるマウントオプションを制御します。

entity_description

これはエンティティにコンテナ、スナップショット、ストレージプール、ボリュームのような説明を追加します。

image_force_refresh

既存のイメージを強制的にリフレッシュできます。

storage_lvm_lv_resizing

これはコンテナの root ディスクデバイス内に size プロパティを設定することで論理ボリュームをリサイズできるようにします。

id_map_base

これは security.idmap.base を新しく導入します。これにより分離されたコンテナに map auto-selection するプロセスをスキップし、ホストのどの UID/GID をベースとして使うかをユーザーが指定できるようにします。

file_symlinks

これは file API 経由でシンボリックリンクを転送するサポートを追加します。X-LXD-type に symlink を指定できるようになり、リクエストの内容はターゲットのパスを指定します。

container_push_target

POST /1.0/containers/<name> に target フィールドを新たに追加します。これはマイグレーション中に作成元の LXD ホストが作成先に接続するために利用可能です。

network_vlan_physical

physical ネットワークデバイスで vlan プロパティが使用できるようにします。

設定すると、 parent インタフェース上で指定された VLAN にアタッチするように LXD に指示します。 LXD はホスト上でその parent と VLAN を既存のインタフェースで探します。見つからない場合は作成します。その後コンテナにこのインタフェースを直接アタッチします。

storage_images_delete

これは指定したストレージプールからイメージのストレージボリュームをストレージ API で削除できるようにします。

container_edit_metadata

これはコンテナの metadata.yaml と関連するテンプレートを/1.0/containers/<name>/metadata 配下の URL にアクセスすることにより API で編集できるようにします。コンテナからイメージを発行する前にコンテナを編集できるようになります。

container_snapshot_stateful_migration

これは stateful なコンテナのスナップショットを新しいコンテナにマイグレートできるようにします。

storage_driver_ceph

これは Ceph ストレージドライバを追加します。

storage_ceph_user_name

これは Ceph ユーザーを指定できるようにします。

instance_types

これはコンテナの作成リクエストに instance_type フィールドを追加します。値は LXD のリソース制限に展開されます。

storage_volatile_initial_source

これはストレージプール作成中に LXD に渡された実際の作成元を記録します。

storage_ceph_force_osd_reuse

これは Ceph ストレージドライバに ceph.osd.force_reuse プロパティを導入します。 true に設定すると LXD は別の LXD インスタンスで既に使用中の OSD ストレージプールを再利用するようになります。

storage_block_filesystem_btrfs

これは ext4 と xfs に加えて Btrfs をストレージボリュームファイルシステムとしてサポートするようになります。

resources

これは LXD が利用可能なシステムリソースを LXD デーモンに問い合わせできるようにします。

kernel_limits

これは nofile でコンテナがオープンできるファイルの最大数といったプロセスのリミットを設定できるようにします。形式は limits.kernel.[**リミット名**] です。

storage_api_volume_rename

これはカスタムストレージボリュームをリネームできるようにします。

external_authentication

これは Macaroons での外部認証をできるようにします。

network_sriov

これは SR-IOV を有効にしたネットワークデバイスのサポートを追加します。

console

これはコンテナのコンソールデバイスとコンソールログを利用可能にします。

restrict_devlxd

`security.devlxd` コンテナ設定キーを新たに導入します。このキーは `/dev/lxd` インタフェースがコンテナで利用可能になるかを制御します。`false` に設定すると、コンテナが LXD デーモンと連携するのを実質無効にすることになります。

migration_pre_copy

これはライブマイグレーション中に最適化されたメモリ転送をできるようにします。

infiniband

これは InfiniBand ネットワークデバイスを使用できるようにします。

maas_network

これは MAAS ネットワーク統合をできるようにします。

デーモンレベルで設定すると、 `nic` デバイスを特定の MAAS サブネットにアタッチできるようになります。

devlxd_events

これは `devlxd` ソケットに Websocket API を追加します。

`devlxd` ソケット上で `/1.0/events` に接続すると、 Websocket 上でイベントのストリームを受け取れるようになります。

proxy

これはコンテナに proxy という新しいデバイスタイプを追加します。これによりホストとコンテナ間で接続をフォワーディングできるようになります。

network_dhcp_gateway

代替のゲートウェイを設定するための ipv4.dhcp.gateway ネットワーク設定キーを新たに追加します。

file_get_symlink

これは file API を使ってシンボリックリンクを取得できるようにします。

network_leases

/1.0/networks/NAME/leases API エンドポイントを追加します。 LXD が管理する DHCP サーバーが稼働するブリッジ上のリースデータベースに問い合わせできるようになります。

unix_device_hotplug

これは Unix デバイスに required プロパティのサポートを追加します。

storage_api_local_volume_handling

これはカスタムストレージボリュームを同じあるいは異なるストレージプール間でコピーしたり移動したりできるようにします。

operation_description

全ての操作に description フィールドを追加します。

clustering

LXD のクラスタリング API 。

これは次の新しいエンドポイントを含みます (詳細は *RESTful API* を参照)。

- GET /1.0/cluster
- UPDATE /1.0/cluster
- GET /1.0/cluster/members
- GET /1.0/cluster/members/<name>

- POST /1.0/cluster/members/<name>
- DELETE /1.0/cluster/members/<name>

次の既存のエンドポイントは以下のように変更されます。

- POST /1.0/containers 新しい target クエリパラメータを受け付けるようになります。
- POST /1.0/storage-pools 新しい target クエリパラメータを受け付けるようになります
- GET /1.0/storage-pool/<name> 新しい target クエリパラメータを受け付けるようになります
- POST /1.0/storage-pool/<pool>/volumes/<type> 新しい target クエリパラメータを受け付けるようになります
- GET /1.0/storage-pool/<pool>/volumes/<type>/<name> 新しい target クエリパラメータを受け付けるようになります
- POST /1.0/storage-pool/<pool>/volumes/<type>/<name> 新しい target クエリパラメータを受け付けるようになります
- PUT /1.0/storage-pool/<pool>/volumes/<type>/<name> 新しい target クエリパラメータを受け付けるようになります
- PATCH /1.0/storage-pool/<pool>/volumes/<type>/<name> 新しい target クエリパラメータを受け付けるようになります
- DELETE /1.0/storage-pool/<pool>/volumes/<type>/<name> 新しい target クエリパラメータを受け付けるようになります
- POST /1.0/networks 新しい target クエリパラメータを受け付けるようになります
- GET /1.0/networks/<name> 新しい target クエリパラメータを受け付けるようになります

event_lifecycle

これはイベント API に lifecycle メッセージ種別を新たに追加します。

storage_api_remote_volume_handling

これはリモート間でカスタムストレージボリュームをコピーや移動できるようにします。

nvidia_runtime

コンテナに nvidia_runtime という設定オプションを追加します。これを true に設定すると NVIDIA ランタイムと CUDA ライブラリーがコンテナに渡されます。

container_mount_propagation

これはディスクデバイスタイプに propagation オプションを新たに追加します。これによりカーネルのマウントプロパゲーションの設定ができるようになります。

container_backup

コンテナのバックアップサポートを追加します。

これは次のエンドポイントを新たに追加します (詳細は *RESTful API* を参照)。

- GET /1.0/containers/<name>/backups
- POST /1.0/containers/<name>/backups
- GET /1.0/containers/<name>/backups/<name>
- POST /1.0/containers/<name>/backups/<name>
- DELETE /1.0/containers/<name>/backups/<name>
- GET /1.0/containers/<name>/backups/<name>/export

次の既存のエンドポイントは以下のように変更されます。

- POST /1.0/containers 新たな作成元の種別 backup を受け付けるようになります

devlxd_images

コンテナに security.devlxd.images 設定オプションを追加します。これにより devlxd 上で /1.0/images/FINGERPRINT/export API が利用可能になります。 nested LXD を動かすコンテナがホストから生のイメージを取得するためにこれは利用できます。

container_local_cross_pool_handling

これは同じ LXD インスタンス上のストレージプール間でコンテナをコピー・移動できるようにします。

proxy_unix

proxy デバイスで Unix ソケットと abstract Unix ソケットの両方のサポートを追加します。これらは `unix:/path/to/unix.sock` (通常のソケット) あるいは `unix:@/tmp/unix.sock` (abstract ソケット) のようにアドレスを指定して利用可能です。

現状サポートされている接続は次のとおりです。

- `TCP <-> TCP`
- `UNIX <-> UNIX`
- `TCP <-> UNIX`
- `UNIX <-> TCP`

proxy_udp

proxy デバイスで UDP のサポートを追加します。

現状サポートされている接続は次のとおりです。

- `TCP <-> TCP`
- `UNIX <-> UNIX`
- `TCP <-> UNIX`
- `UNIX <-> TCP`
- `UDP <-> UDP`
- `TCP <-> UDP`
- `UNIX <-> UDP`

clustering_join

これにより GET /1.0/cluster がノードに参加する際にどのようなストレージプールとネットワークを作成する必要があるかについての情報を返します。また、それらを作成する際にどのノード固有の設定キーを使う必要があるかについての情報も返します。同様に PUT /1.0/cluster エンドポイントも同じ形式でストレージプールとネットワークについての情報を受け付け、クラスタに参加する前にこれらが自動的に作成されるようになります。

proxy_tcp_udp_multi_port_handling

複数のポートにトラフィックをフォワーディングできるようにします。フォワーディングはポートの範囲が転送元と転送先で同じ (例えば 1.2.3.4 0-1000 -> 5.6.7.8 1000-2000) 場合か転送元で範囲を指定し転送先で単一のポートを指定する (例えば 1.2.3.4 0-1000 -> 5.6.7.8 1000) 場合に可能です。

network_state

ネットワークの状態を取得できるようになります。

これは次のエンドポイントを新たに追加します (詳細は *RESTful API* を参照)。

- GET /1.0/networks/<name>/state

proxy_unix_dac_properties

これは抽象的 Unix ソケットではない Unix ソケットに GID, UID, パーミションのプロパティを追加します。

container_protection_delete

security.protection.delete フィールドを設定できるようにします。 true に設定するとコンテナが削除されるのを防ぎます。スナップショットはこの設定により影響を受けません。

proxy_priv_drop

proxy デバイスに security.uid と security.gid を追加します。これは root 権限を落とし (訳注: 非 root 権限で動作させるという意味です)、 Unix ソケットに接続する際に用いられる UID/GID も変更します。

pprof_http

これはデバッグ用の HTTP サーバーを起動するために、新たに core.debug_address オプションを追加します。

このサーバーは現在 pprof API を含んでおり、従来の cpu-profile, memory-profile と print-goroutines デバッグオプションを置き換えるものです。

proxy_haproxy_protocol

proxy デバイスに proxy_protocol キーを追加します。これは HAProxy PROXY プロトコルヘッダの使用を制御します。

network_hwaddr

ブリッジの MAC アドレスを制御する bridge.hwaddr キーを追加します。

proxy_nat

これは最適化された UDP/TCP プロキシを追加します。設定上可能であればプロキシ処理は proxy デバイスの代わりに iptables 経由で行われるようになります。

network_nat_order

LXD ブリッジに ipv4.nat.order と ipv6.nat.order 設定キーを導入します。これらのキーは LXD のルールをチェイン内の既存のルールの前に置くか後に置くかを制御します。

container_full

これは GET /1.0/containers に recursion=2 という新しいモードを導入します。これにより状態、スナップショットとバックアップの構造を含むコンテナの全ての構造を取得できるようになります。

この結果 lxc list は必要な全ての情報を 1 つのクエリで取得できるようになります。

candid_authentication

これは新たに candid.api.url 設定キーを導入し core.macaroon.endpoint を削除します。

backup_compression

これは新たに backups.compression_algorithm 設定キーを導入します。これによりバックアップの圧縮の設定が可能になります。

candid_config

これは candid.domains と candid.expiry 設定キーを導入します。前者は許可された／有効な Candid ドメインを指定することを可能にし、後者は macaroon の有効期限を設定可能にします。 lxc remote add コマンドに新たに --domain フラグが追加され、これにより Candid ドメインを指定可能になります。

nvidia_runtime_config

これは nvidia.runtime と libnvidia-container ライブラリーを使用する際に追加のいくつかの設定キーを導入します。これらのキーは NVIDIA container の対応する環境変数にほぼそのまま置き換えられます。

- nvidia.driver.capabilities => NVIDIA_DRIVER_CAPABILITIES
- nvidia.require.cuda => NVIDIA_REQUIRE_CUDA
- nvidia.require.driver => NVIDIA_REQUIRE_DRIVER

storage_api_volume_snapshots

ストレージボリュームスナップショットのサポートを追加します。これらはコンテナスナップショットのように振る舞いますが、ボリュームに対してのみ作成できます。

これにより次の新しいエンドポイントが追加されます (詳細は *RESTful API* を参照)。

- GET /1.0/storage-pools/<pool>/volumes/<type>/<name>/snapshots
- POST /1.0/storage-pools/<pool>/volumes/<type>/<name>/snapshots
- GET /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>
- PUT /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>
- POST /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>
- DELETE /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>

storage_unmapped

ストレージボリュームに新たに security.unmapped という Boolean 設定を導入します。

true に設定するとボリューム上の現在のマップをフラッシュし、以降の idmap のトラッキングとボリューム上のリマッピングを防ぎます。

これは隔離されたコンテナ間でデータを共有するために使用できます。この際コンテナを書き込みアクセスを要求するコンテナにアタッチした後にデータを共有します。

projects

新たに project API を追加します。プロジェクトの作成、更新、削除ができます。

現時点では、プロジェクトは、コンテナ、プロファイル、イメージを保持できます。そして、プロジェクトを切り替えることで、独立した LXD リソースのビューを見せられます。

candid_config_key

新たに candid.api.key オプションが使えるようになります。これにより、エンドポイントが期待する公開鍵を設定でき、HTTP のみの Candid サーバーを安全に利用できます。

network_vxlan_ttl

新たにネットワークの設定に tunnel.NAME.ttl が指定できるようになります。これにより、VXLAN トンネルの TTL を増加させることができます。

container_incremental_copy

新たにコンテナのインクリメンタルコピーができるようになります。--refresh オプションを指定してコンテナをコピーすると、見つからないファイルや、更新されたファイルのみをコピーします。コンテナが存在しない場合は、通常のコピーを実行します。

usb_optional_vendorid

名前が暗示しているように、コンテナにアタッチされた USB デバイスの vendorid フィールドが省略可能になります。これにより全ての USB デバイスがコンテナに渡されます (GPU に対してなされたのと同様)。

snapshot_scheduling

これはスナップショットのスケジューリングのサポートを追加します。これにより 3 つの新しい設定キーが導入されます。 snapshots.schedule, snapshots.schedule.stopped, そして snapshots.pattern です。スナップショットは最短で 1 分間隔で自動的に作成されます。

snapshots_schedule_aliases

スナップショットのスケジュールはスケジュールエイリアスのカンマ区切りリストで設定できます。インスタンスには <@hourly> <@daily> <@midnight> <@weekly> <@monthly> <@annually> <@yearly> <@startup>、ストレージボリュームには <@hourly> <@daily> <@midnight> <@weekly> <@monthly> <@annually> <@yearly> のエイリアスが利用できます。

container_copy_project

コピー元のコンテナの dict に project フィールドを導入します。これによりプロジェクト間でコンテナをコピーあるいは移動できるようになります。

clustering_server_address

これはサーバーのネットワークアドレスを REST API のクライアントネットワークアドレスと異なる値に設定することのサポートを追加します。クライアントは新しい cluster.https_address 設定キーを初期のサーバーのアドレスを指定するためにに設定できます。新しいサーバーが参加する際、クライアントは参加するサーバーの core.https_address 設定キーを参加するサーバーがリッスンすべきアドレスに設定でき、PUT /1.0/cluster API の server_address キーを参加するサーバーがクラスタリングトラフィックに使用すべきアドレスに設定できます (server_address の値は自動的に参加するサーバーの cluster.https_address 設定キーにコピーされます)。

clustering_image_replication

クラスタ内のノードをまたいだイメージのレプリケーションを可能にします。新しい cluster.images_minimal_replica 設定キーが導入され、イメージのリプリケーションに対するノードの最小数を指定するのに使用できます。

container_protection_shift

security.protection.shift の設定を可能にします。これによりコンテナのファイルシステム上で UID/GID をシフト (再マッピング) させることを防ぎます。

snapshot_expiry

これはスナップショットの有効期限のサポートを追加します。タスクは 1 分おきに実行されます。snapshots.expiry 設定オプションは、1M 2H 3d 4w 5m 6y（それぞれ 1 分、2 時間、3 日、4 週間、5 ヶ月、6 年）といった形式を取ります。この指定ではすべての部分を使う必要はありません。

作成されるスナップショットには、指定した式に基づいて有効期限が設定されます。expires_at で定義される有効期限は、API や lxc config edit コマンドを使って手動で編集できます。有効な有効期限が設定されたスナップショットはタスク実行時に削除されます。有効期限は expires_at に空文字列や 0001-01-01T00:00:00Z（zero time）を設定することで無効化できます。snapshots.expiry が設定されていない場合はこれがデフォルトです。

これは次のような新しいエンドポイントを追加します（詳しくは *RESTful API* をご覧ください）：

- PUT /1.0/containers/<name>/snapshots/<name>

snapshot_expiry_creation

コンテナ作成に expires_at を追加し、作成時にスナップショットの有効期限を上書きできます。

network_leases_location

ネットワークのリースリストに Location フィールドを導入します。これは、特定のリースがどのノードに存在するかを問い合わせるときに使います。

resources_cpu_socket

ソケットの情報が入れ替わる場合に備えて CPU リソースにソケットフィールドを追加します。

resources_gpu

サーバーリソースに新規に GPU 構造を追加し、システム上で利用可能な全ての GPU を一覧表示します。

resources_numa

全ての CPU と GPU に対する NUMA ノードを表示します。

kernel_features

サーバーの環境からオプショナルなカーネル機能の使用可否状態を取得します。

id_map_current

内部的な `volatile.idmap.current` キーを新規に導入します。これはコンテナに対する現在のマッピングを追跡するのに使われます。

実質的には以下が利用可能になります。

- `volatile.last_state.idmap` => ディスク上の idmap
- `volatile.idmap.current` => 現在のカーネルマップ
- `volatile.idmap.next` => 次のディスク上の idmap

これはディスク上の map が変更されていないがカーネルマップは変更されている (例: `shiftfs`) ような環境を実装するために必要です。

event_location

API イベントの世代の場所を公開します。

storage_api_remote_volume_snapshots

ストレージボリュームをそれらのスナップショットを含んで移行できます。

network_nat_address

これは LXD ブリッジに `ipv4.nat.address` と `ipv6.nat.address` 設定キーを導入します。これらのキーはブリッジからの送信トラフィックに使うソースアドレスを制御します。

container_nic_routes

これは nic タイプのデバイスに ipv4.routes と ipv6.routes プロパティを導入します。ホストからコンテナへの NIC への静的ルートが追加できます。

rbac

RBAC (role based access control; ロールベースのアクセス制御) のサポートを追加します。これは以下の設定キーを新規に導入します。

- rbac.api.url
- rbac.api.key
- rbac.api.expiry
- rbac.agent.url
- rbac.agent.username
- rbac.agent.private_key
- rbac.agent.public_key

cluster_internal_copy

これは通常の POST /1.0/containers を実行することでクラスタノード間でコンテナをコピーすることを可能にします。この際 LXD はマイグレーションが必要かどうかを内部的に判定します。

seccomp_notify

カーネルが seccomp ベースの syscall インターセプトをサポートする場合に登録された syscall が実行されたことをコンテナから LXD に通知することができます。 LXD はそれを受けて様々なアクションをトリガーするかを決定します。

lxc_features

これは GET /1.0 ルート経由で lxc info コマンドの出力に lxc_features セクションを導入します。配下の LXC ライブラリーに存在するキー・フィーチャーに対するチェックの結果を出力します。

container_nic_ipvlan

これは nic デバイスに ipvlan のタイプを導入します。

network_vlan_sriov

これは SR-IOV デバイスに VLAN (vlan) と MAC フィルタリング (security.mac_filtering) のサポートを導入します。

storage_cephfs

ストレージプールドライバとして CephFS のサポートを追加します。これはカスタムボリュームとしての利用のみが可能になり、イメージとコンテナは CephFS ではなく Ceph (RBD) 上に構築する必要があります。

container_nic_ipfilter

これは bridged の NIC デバイスに対してコンテナの IP フィルタリング (security.ipv4_filtering and security.ipv6_filtering) を導入します。

resources_v2

/1.0/resources のリソース API を見直しました。主な変更は以下の通りです。

- CPU
  - ソケット、コア、スレッドのトラッキングのレポートを修正しました
  - コア毎の NUMA ノードのトラッキング
  - ソケット毎のベースとターボの周波数のトラッキング
  - コア毎の現在の周波数のトラッキング
  - CPU のキャッシュ情報の追加
  - CPU アーキテクチャをエクスポート
  - スレッドのオンライン／オフライン状態を表示
- メモリ
  - HugePages のトラッキングを追加
  - NUMA ノード毎でもメモリ消費を追跡
- GPU
  - DRM 情報を別の構造体に分離

- DRM 構造体内にデバイスの名前とノードを公開
- NVIDIA 構造体内にデバイスの名前とノードを公開
- SR-IOV VF のトラッキングを追加

container_exec_user_group_cwd

`POST /1.0/containers/NAME/exec` の実行時に `User`, `Group` と `Cwd` を指定するサポートを追加

container_syscall_intercept

`security.syscalls.intercept.*` 設定キーを追加します。これはどのシステムコールを LXD がインターセプトし昇格された権限で処理するかを制御します。

container_disk_shift

`disk` デバイスに `shift` プロパティを追加します。これは `shiftfs` のオーバーレイの使用を制御します。

storage_shifted

ストレージボリュームに新しく `security.shifted` という Boolean の設定を導入します。

これを `true` に設定すると複数の隔離されたコンテナが、それら全てがファイルシステムに書き込み可能にしたまま、同じストレージボリュームにアタッチするのを許可します。

これは `shiftfs` をオーバーレイファイルシステムとして使用します。

resources_infiniband

リソース API の一部として InfiniBand キャラクタデバイス (`issm`, `umad`, `uverb`) の情報を公開します。

daemon_storage

これは `storage.images_volume` と `storage.backups_volume` という 2 つの新しい設定項目を導入します。これらは既存のプール上のストレージボリュームがデーモン全体のイメージとバックアップを保管するのに使えるようにします。

instances

これはインスタンスの概念を導入します。現状ではインスタンスの唯一の種別は container です。

image_types

これはイメージに新しく Type フィールドのサポートを導入します。 Type フィールドはイメージがどういう種別かを示します。

resources_disk_sata

ディスクリソース API の構造体を次の項目を含むように拡張します。

- SATA デバイス (種別) の適切な検出
- デバイスパス
- ドライブの RPM
- ブロックサイズ
- ファームウェアバージョン
- シリアルナンバー

clustering_roles

これはクラスタのエントリーに roles という新しい属性を追加し、クラスタ内のメンバーが提供する role の一覧を公開します。

images_expiry

イメージの有効期限を設定できます。

resources_network_firmware

ネットワークカードのエントリーに FirmwareVersion フィールドを追加します。

backup_compression_algorithm

バックアップを作成する (POST /1.0/containers/<name>/backups) 際に compression_algorithm プロパティのサポートを追加します。

このプロパティを設定するとデフォルト値 (backups.compression_algorithm) をオーバーライドすることができます。

ceph_data_pool_name

Ceph RBD を使ってストレージプールを作成する際にオプショナルな引数 (ceph.osd.data_pool_name) のサポートを追加します。この引数が指定されると、プールはメタデータは pool_name で指定されたプールに保持しつつ実際のデータは data_pool_name で指定されたプールに保管するようになります。

container_syscall_intercept_mount

security.syscalls.intercept.mount, security.syscalls.intercept.mount.allowed, security.syscalls.intercept.mount.shift 設定キーを追加します。これらは mount システムコールを LXD にインターセプトさせるかどうか、昇格されたパーミションでどのように処理させるかを制御します。

compression_squashfs

イメージやバックアップを SquashFS ファイルシステムの形式でインポート／エクスポートするサポートを追加します。

container_raw_mount

ディスクデバイスに raw mount オプションを渡すサポートを追加します。

container_nic_routed

routed nic デバイスタイプを導入します。

container_syscall_intercept_mount_fuse

security.syscalls.intercept.mount.fuse キーを追加します。これはファイルシステムのマウントを fuse 実装にリダイレクトするのに使えます。このためには例えば security.syscalls.intercept.mount.fuse=ext4=fuse2fs のように設定します。

container_disk_ceph

既存の Ceph RBD もしくは CephFS を直接 LXD コンテナに接続できます。

virtual_machines

仮想マシンサポートが追加されます。

image_profiles

新しいコンテナを起動するときに、イメージに適用するプロファイルのリストが指定できます。

clustering_architecture

クラスタメンバーに `architecture` 属性を追加します。この属性はクラスタメンバーのアーキテクチャを示します。

resources_disk_id

リソース API のディスクのエントリーに `device_id` フィールドを追加します。

storage_lvm_stripes

通常のボリュームと thin pool ボリューム上で LVM ストライプを使う機能を追加します。

vm_boot_priority

ブートの順序を制御するため NIC とディスクデバイスに `boot.priority` プロパティを追加します。

unix_hotplug_devices

Unix のキャラクタデバイスとブロックデバイスのホットプラグのサポートを追加します。

api_filtering

インスタンスとイメージに対する GET リクエストの結果をフィルタリングする機能を追加します。

instance_nic_network

NIC デバイスの network プロパティのサポートを追加し、管理されたネットワークへ NIC をリンクできるようにします。これによりネットワーク設定の一部を引き継ぎ、 IP 設定のより良い検証を行うことができます。

clustering_sizing

データベースの投票者とスタンバイに対してカスタムの値を指定するサポートです。cluster.max_voters と cluster.max_standby という新しい設定キーが導入され、データベースの投票者とスタンバイの理想的な数を指定できます。

firewall_driver

ServerEnvironment 構造体にファイアーウォールのドライバが使用されていることを示す Firewall プロパティを追加します。

storage_lvm_vg_force_reuse

既存の空でないボリュームグループからストレージボリュームを作成する機能を追加します。このオプションの使用には注意が必要です。というのは、同じボリュームグループ内に LXD 以外で作成されたボリュームとボリューム名が衝突しないことを LXD が保証できないからです。このことはもし名前の衝突が起きたときは LXD 以外で作成されたボリュームを LXD が削除してしまう可能性があることを意味します。

container_syscall_intercept_hugetlbfs

mount システムコール・インターセプションが有効にされ hugetlbfs が許可されたファイルシステムとして指定された場合、 LXD は別の hugetlbfs インスタンスを UID と GID をコンテナの root の UID と GID に設定するマウントオプションを指定してコンテナにマウントします。これによりコンテナ内のプロセスが huge page を確実に利用できるようにします。

limits_hugepages

コンテナが使用できる huge page の数を hugetlb cgroup を使って制限できるようにします。この機能を使用するには hugetlb cgroup が利用可能になっている必要があります。注意: hugetlbfs ファイルシステムの mount システムコールをインターセプトするときは、ホストの huge page のリソースをコンテナが使い切ってしまわないように huge page を制限することを推奨します。

container_nic_routed_gateway

この拡張は ipv4.gateway と ipv6.gateway という NIC の設定キーを追加します。指定可能な値は auto か none のいずれかです。値を指定しない場合のデフォルト値は auto です。auto に設定した場合は、デフォルトゲートウェイがコンテナ内部に追加され、ホスト側のインタフェースにも同じゲートウェイアドレスが追加されるという現在の挙動と同じになります。none に設定すると、デフォルトゲートウェイもアドレスもホスト側のインタフェースには追加されません。これにより複数のルートを持つ NIC デバイスをコンテナに追加できます。

projects_restrictions

この拡張はプロジェクトに restricted という設定キーを追加します。これによりプロジェクト内でセキュリティセンシティブな機能を使うのを防ぐことができます。

custom_volume_snapshot_expiry

この拡張はカスタムボリュームのスナップショットに有効期限を設定できるようにします。有効期限は snapshots.expiry 設定キーにより個別に設定することも出来ますし、親のカスタムボリュームに設定してそこから作成された全てのスナップショットに自動的にその有効期限を適用することも出来ます。

volume_snapshot_scheduling

この拡張はカスタムボリュームのスナップショットにスケジュール機能を追加します。snapshots.schedule と snapshots.pattern という 2 つの設定キーが新たに追加されます。スナップショットは最短で 1 分毎に作成可能です。

trust_ca_certificates

この拡張により提供された CA (server.ca) によって信頼されたクライアント証明書のチェックが可能になります。core.trust_ca_certificates を true に設定すると有効にできます。有効な場合、クライアント証明書のチェックを行い、チェックが OK であれば信頼されたパスワードの要求はスキップします。ただし、提供された CRL (ca.crl) に接続してきたクライアント証明書が含まれる場合は例外です。この場合は、パスワードが求められます。

snapshot_disk_usage

この拡張はスナップショットのディスク使用量を示す /1.0/instances/<name>/snapshots/<snapshot> の出力に size フィールドを新たに追加します。

clustering_edit_roles

この拡張はクラスタメンバーに書き込み可能なエンドポイントを追加し、ロールの編集を可能にします。

container_nic_routed_host_address

この拡張は NIC の設定キーに ipv4.host_address と ipv6.host_address を追加し、ホスト側の veth インタフェースの IP アドレスを制御できるようにします。これは同時に複数の routed NIC を使用し、予測可能な next-hop のアドレスを使用したい場合に有用です。

さらにこの拡張は ipv4.gateway と ipv6.gateway の NIC 設定キーの振る舞いを変更します。auto に設定するとコンテナはデフォルトゲートウェイをそれぞれ ipv4.host_address と ipv6.host_address で指定した値にします。

デフォルト値は次の通りです。

ipv4.host_address: 169.254.0.1 ipv6.host_address: fe80::1

これは以前のデフォルトの挙動と後方互換性があります。

container_nic_ipvlan_gateway

この拡張は ipv4.gateway と ipv6.gateway の NIC 設定キーを追加し auto か none の値を指定できます。指定しない場合のデフォルト値は auto です。この場合は従来同様の挙動になりコンテナ内部に追加されるデフォルトゲートウェイと同じアドレスがホスト側のインタフェースにも追加されます。none に設定された場合、ホスト側のインタフェースにはデフォルトゲートウェイもアドレスも追加されません。これによりコンテナに IPVLAN の NIC デバイスを複数追加することができます。

resources_usb_pci

この拡張は /1.0/resources の出力に USB と PC デバイスを追加します。

resources_cpu_threads_numa

この拡張は numa_node フィールドをコアごとではなくスレッドごとに記録するように変更します。これは一部のハードウェアでスレッドを異なる NUMA ドメインに入れる場合があるようなのでそれに対応するためのものです。

resources_cpu_core_die

それぞれのコアごとに die_id 情報を公開します。

api_os

この拡張は /1.0 内に os と os_version の 2 つのフィールドを追加します。

これらの値はシステム上の OS-release のデータから取得されます。

container_nic_routed_host_table

この拡張は ipv4.host_table と ipv6.host_table という NIC の設定キーを導入します。これで指定した ID のカスタムポリシーのルーティングテーブルにインスタンスの IP のための静的ルートを追加できます。

container_nic_ipvlan_host_table

この拡張は ipv4.host_table と ipv6.host_table という NIC の設定キーを導入します。これで指定した ID のカスタムポリシーのルーティングテーブルにインスタンスの IP のための静的ルートを追加できます。

container_nic_ipvlan_mode

この拡張は mode という NIC の設定キーを導入します。これにより ipvlan モードを l2 か l3s のいずれかに切り替えられます。指定しない場合、デフォルトは l3s （従来の挙動）です。

l2 モードでは ipv4.address と ipv6.address キーは CIDR か単一アドレスの形式を受け付けます。単一アドレスの形式を使う場合、デフォルトのサブネットのサイズは IPv4 では /24 、 IPv6 では /64 となります。

l2 モードでは ipv4.gateway と ipv6.gateway キーは単一の IP アドレスのみを受け付けます。

resources_system

この拡張は /1.0/resources の出力にシステム情報を追加します。

images_push_relay

この拡張はイメージのコピーに push と relay モードを追加します。また以下の新しいエンドポイントも追加します。

- POST 1.0/images/<fingerprint>/export

network_dns_search

この拡張はネットワークに dns.search という設定オプションを追加します。

container_nic_routed_limits

この拡張は routed NIC に limits.ingress, limits.egress, limits.max を追加します。

instance_nic_bridged_vlan

この拡張は bridged NIC に vlan と vlan.tagged の設定を追加します。

vlan には参加するタグなし VLAN を指定し、 vlan.tagged は参加するタグ VLAN のカンマ区切りリストを指定します。

network_state_bond_bridge

この拡張は /1.0/networks/NAME/state API に bridge と bond のセクションを追加します。

これらはそれぞれの特定のタイプに関連する追加の状態の情報を含みます。

Bond:

- Mode
- Transmit hash
- Up delay
- Down delay
- MII frequency
- MII state
- Lower devices

Bridge:

- ID
- Forward delay
- STP mode
- Default VLAN
- VLAN filtering
- Upper devices

resources_cpu_isolated

この拡張は CPU スレッドに Isolated プロパティを追加します。これはスレッドが物理的には Online ですがタスクを受け付けないように設定しているかを示します。

usedby_consistency

この拡張により、可能な時は UsedBy が適切な ?project= と ?target= に対して一貫性があるようになるはずです。

UsedBy を持つ 5 つのエンティティーは以下の通りです。

- プロファイル
- プロジェクト
- ネットワーク
- ストレージプール
- ストレージボリューム

custom_block_volumes

この拡張によりカスタムブロックボリュームを作成しインスタンスにアタッチできるようになります。カスタムストレージボリュームの作成時に --type フラグが新規追加され、 fs と block の値を受け付けます。

clustering_failure_domains

この拡張は PUT /1.0/cluster/<node> API に failure_domain フィールドを追加します。これはノードの failure domain を設定するのに使えます。

container_syscall_filtering_allow_deny_syntax

いくつかのシステムコールに関連したコンテナの設定キーが更新されました。

- security.syscalls.deny_default
- security.syscalls.deny_compat
- security.syscalls.deny
- security.syscalls.allow

resources_gpu_mdev

/1.0/resources の利用可能な媒介デバイス (mediated device) のプロファイルとデバイスを公開します。

console_vga_type

この拡張は /1.0/console エンドポイントが ?type= 引数を取るように拡張します。これは console (デフォルト) か vga (この拡張で追加される新しいタイプ) を指定可能です。

/1.0/<instance name>/console?type=vga に POST する際はメタデータフィールド内の操作の結果ウェブソケットにより返されるデータはターゲットの仮想マシンの SPICE Unix ソケットにアタッチされた双方向のプロキシになります。

projects_limits_disk

利用可能なプロジェクトの設定キーに limits.disk を追加します。これが設定されるとプロジェクト内でインスタンスボリューム、カスタムボリューム、イメージボリュームが使用できるディスクスペースの合計の量を制限できます。

network_type_macvlan

ネットワークタイプ macvlan のサポートを追加し、このネットワークタイプに parent 設定キーを追加します。これは NIC デバイスインタフェースを作る際にどの親インタフェースを使用するべきかを指定します。

さらに macvlan の NIC に network 設定キーを追加します。これは NIC デバイスの設定の基礎として使う network を指定します。

network_type_sriov

ネットワークタイプ sriov のサポートを追加し、このネットワークタイプに parent 設定キーを追加します。これは NIC デバイスインタフェースを作る際にどの親インタフェースを使用するべきかを指定します。

さらに sriov の NIC に network 設定キーを追加します。これは NIC デバイスの設定の基礎として使う network を指定します。

container_syscall_intercept_bpf_devices

この拡張はコンテナ内で bpf のシステムコールをインターセプトする機能を提供します。具体的には device cgroup の bpf のプログラムを管理できるようにします。

network_type_ovn

ネットワークタイプ ovn のサポートを追加し、 bridge タイプのネットワークを parent として設定できるようにします。

ovn という新しい NIC のデバイスタイプを追加します。これにより network 設定キーにどの ovn のタイプのネットワークに接続すべきかを指定できます。

さらに全ての ovn ネットワークと NIC デバイスに適用される 2 つのグローバルの設定キーを追加します。

- network.ovn.integration_bridge - 使用する OVS 統合ブリッジ
- network.ovn.northbound_connection - OVN northbound データベース接続文字列

projects_networks

プロジェクトに features.networks 設定キーを追加し、プロジェクトがネットワークを保持できるようにします。

projects_networks_restricted_uplinks

プロジェクトに restricted.networks.uplinks 設定キーを追加し、プロジェクト内で作られたネットワークがそのアップリンクのネットワークとしてどのネットワークが使えるかを（カンマ区切りリストで）指定します。

custom_volume_backup

カスタムボリュームのバックアップサポートを追加します。

この拡張は以下の新しい API エンドポイント （詳細は *RESTful API* を参照）を含みます。

- GET /1.0/storage-pools/<pool>/<type>/<volume>/backups
- POST /1.0/storage-pools/<pool>/<type>/<volume>/backups
- GET /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>
- POST /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>
- DELETE /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>
- GET /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>/export

以下の既存のエンドポイントが変更されます。

- POST /1.0/storage-pools/<pool>/<type>/<volume> が新しいソースタイプとして backup を受け付けます

backup_override_name

InstanceBackupArgs に Name フィールドを追加し、バックアップをリストアする際に別のインスタンス名を指定できるようにします。

StoragePoolVolumeBackupArgs に Name と PoolName フィールドを追加し、カスタムボリュームのバックアップをリストアする際に別のボリューム名を指定できるようにします。

storage_rsync_compression

ストレージプールに rsync.compression 設定キーを追加します。このキーはストレージプールをマイグレートする際に rsync での圧縮を無効にするために使うことができます。

network_type_physical

新たに physical というネットワークタイプのサポートを追加し、 ovn ネットワークのアップリンクとして使用できるようにします。

physical ネットワークの parent で指定するインタフェースは ovn ネットワークのゲートウェイに接続されます。

network_ovn_external_subnets

ovn ネットワークがアップリンクネットワークの外部のサブネットを使用できるようにします。

physical ネットワークに ipv4.routes と ipv6.routes の設定を追加します。これは子供の OVN ネットワークで ipv4.routes.external と ipv6.routes.external の設定で使用可能な外部のルートを指定します。

プロジェクトに restricted.networks.subnets 設定を追加します。これはプロジェクト内の OVN ネットワークで使用可能な外部のサブネットを指定します（未設定の場合はアップリンクネットワークで定義される全てのルートが使用可能です）。

network_ovn_nat

ovn ネットワークに ipv4.nat と ipv6.nat の設定を追加します。

これらの設定 (訳注: ipv4.nat や ipv6.nat) を未設定でネットワークを作成する際、(訳注: ipv4.address や ipv6.address が未設定あるいは auto の場合に) 対応するアドレス (訳注: ipv4.nat であれば ipv4.address、ipv6.nat であれば ipv6.address) がサブネット用に生成される場合は適切な NAT が生成され、ipv4.nat や ipv6.nat は true に設定されます。

この設定がない場合は値は false として扱われます。

network_ovn_external_routes_remove

ovn ネットワークから ipv4.routes.external と ipv6.routes.external の設定を削除します。

ネットワークと NIC レベルの両方で指定するのではなく、 ovn NIC タイプ上で等価な設定を使えます。

tpm_device_type

tpm デバイスタイプを導入します。

storage_zfs_clone_copy_rebase

zfs.clone_copy に rebase という値を導入します。この設定で LXD は先祖の系列上の image データセットを追跡し、その最上位に対して send/receive を実行します。

gpu_mdev

これは仮想 CPU のサポートを追加します。GPU デバイスに mdev 設定キーを追加し、i915-GVTg_V5_4 のようなサポートされる mdev のタイプを指定します。

resources_pci_iommu

これはリソース API の PCI エントリーに IOMMUGroup フィールドを追加します。

resources_network_usb

リソース API のネットワークカードエントリーに usb_address フィールドを追加します。

resources_disk_address

リソース API のディスクエントリーに usb_address と pci_address フィールドを追加します。

network_physical_ovn_ingress_mode

physical ネットワークに ovn.ingress_mode 設定を追加します。

OVN NIC ネットワークの外部 IP アドレスがアップリンクネットワークにどのように広告されるかの方法を設定します。

l2proxy (proxy ARP/NDP) か routed のいずれかを指定します。

network_ovn_dhcp

ovn ネットワークに ipv4.dhcp と ipv6.dhcp の設定を追加します。

DHCP (と IPv6 の RA) を無効にできます。デフォルトはオンです。

network_physical_routes_anycast

physical ネットワークに ipv4.routes.anycast と ipv6.routes.anycast の Boolean の設定を追加します。デフォルトは false です。

ovn.ingress_mode=routed と共に使うと physical ネットワークをアップリンクとして使う OVN ネットワークでサブネット／ルートのオーバーラップ検出を緩和できます。

projects_limits_instances

limits.instances を利用可能なプロジェクトの設定キーに追加します。設定するとプロジェクト内で使われるインスタンス（VM とコンテナ）の合計数を制限します。

network_state_vlan

これは /1.0/networks/NAME/state API に vlan セクションを追加します。

これらは VLAN インタフェースに関連する追加の状態の情報を含みます。

- lower_device
- vid

instance_nic_bridged_port_isolation

これは bridged NIC に security.port_isolation のフィールドを追加します。

instance_bulk_state_change

一括状態変更（詳細は *REST API* を参照）のために次のエンドポイントを追加します。

- PUT /1.0/instances

network_gvrp

これはオプショナルな gvrp プロパティを macvlan と physical ネットワークに追加し、さらに ipvlan, macvlan, routed, physical NIC デバイスにも追加します。

設定された場合は、これは VLAN が GARP VLAN Registration Protocol を使って登録すべきかどうかを指定します。デフォルトは false です。

instance_pool_move

これは POST /1.0/instances/NAME API に pool フィールドを追加し、プール間でインスタンスのルートディスクを簡単に移動できるようにします。

gpu_sriov

これは SR-IOV を有効にした GPU のサポートを追加します。これにより sriov という GPU タイプのプロパティが追加されます。

pci_device_type

これは pci デバイスタイプを追加します。

storage_volume_state

/1.0/storage-pools/POOL/volumes/VOLUME/state API エンドポイントを新規追加しボリュームの使用量を取得できるようにします。

network_acl

これは /1.0/network-acls の API エンドポイントプリフィクス以下の API にネットワークの ACL のコンセプトを追加します。

migration_stateful

migration.stateful という設定キーを追加します。

disk_state_quota

これは disk デバイスに size.state というデバイス設定キーを追加します。

storage_ceph_features

ストレージプールに ceph.rbd.features 設定キーを追加し、新規ボリュームに使用する RBD の機能を制御します。

projects_compression

backups.compression_algorithm と images.compression_algorithm 設定キーを追加します。これらによりプロジェクトごとのバックアップとイメージの圧縮の設定が出来るようになります。

projects_images_remote_cache_expiry

プロジェクトに images.remote_cache_expiry 設定キーを追加します。これを設定するとキャッシュされたリモートのイメージが指定の日数使われない場合は削除されるようになります。

certificate_project

API 内の証明書に restricted と projects プロパティを追加します。projects は証明書がアクセスしたプロジェクト名の一覧を保持します。

network_ovn_acl

OVN ネットワークと OVN NIC に security.acls プロパティを追加します。これにより ネットワークに ACL をかけられるようになります。

projects_images_auto_update

images.auto_update_cached と images.auto_update_interval 設定キーを追加します。これらによりプロジェクト内のイメージの自動更新を設定できるようになります。

projects_restricted_cluster_target

プロジェクトに restricted.cluster.target 設定キーを追加します。これによりどのクラスタメンバーにワークロードを配置するかやメンバー間のワークロードを移動する能力を指定する --target オプションをユーザーに使わせないように出来ます。

images_default_architecture

images.default_architecture をグローバルの設定キーとプロジェクトごとの設定キーとして追加します。これはイメージリクエストの一部として明示的に指定しなかった場合にどのアーキテクチャーを使用するかを LXD に指定します。

network_ovn_acl_defaults

OVN ネットワークと NIC に security.acls.default.{in,e}gress.action と security.acls.default.{in,e}gress.logged 設定キーを追加します。これは削除された ACL の default.action と default.logged キーの代わりになるものです。

gpu_mig

これは NVIDIA MIG のサポートを追加します。mig GPU type と関連する設定キーを追加します。

project_usage

プロジェクトに現在のリソース割り当ての情報を取得する API エンドポイントを追加します。API の GET /1.0/projects/<name>/state で利用できます。

network_bridge_acl

bridge ネットワークに security.acls 設定キーを追加し、ネットワーク ACL を適用できるようにします。

さらにマッチしなかったトラフィックに対するデフォルトの振る舞いを指定する security.acls.default.{in,e}gress.action と security.acls.default.{in,e}gress.logged 設定キーを追加します。

warnings

LXD の警告 API です。

この拡張は次のエンドポイントを含みます（詳細は *Restful API* 参照）。

- GET /1.0/warnings
- GET /1.0/warnings/<uuid>
- PUT /1.0/warnings/<uuid>
- DELETE /1.0/warnings/<uuid>

projects_restricted_backups_and_snapshots

プロジェクトに restricted.backups と restricted.snapshots 設定キーを追加し、ユーザーがバックアップやスナップショットを作成できないようにします。

clustering_join_token

トラスト・パスワードを使わずに新しいクラスタメンバーを追加する際に使用する参加トークンをリクエストするための POST /1.0/cluster/members API エンドポイントを追加します。

clustering_description

クラスタメンバーに編集可能な説明を追加します。

server_trusted_proxy

core.https_trusted_proxy のサポートを追加します。 この設定は、LXD が HAProxy スタイルの connection ヘッダーをパースし、そのような（HAProxy などのリバースプロキシサーバーが LXD の前面に存在するような）接続の場合でヘッダーが存在する場合は、プロキシサーバーが（ヘッダーで）提供するリクエストの（実際のクライアントの）ソースアドレスへ（LXD が）ソースアドレスを書き換え（て、LXD の管理するクラスタにリクエストを送出し）ます。（LXD のログにもオリジナルのアドレスを記録します）

clustering_update_cert

クラスタ全体に適用されるクラスタ証明書を更新するための PUT /1.0/cluster/certificate エンドポイントを追加します。

storage_api_project

これはプロジェクト間でカスタムストレージボリュームをコピー／移動できるようにします。

server_instance_driver_operational

これは /1.0 エンドポイントの driver の出力をサーバー上で実際にサポートされ利用可能であるドライバのみを含めるように修正します（LXD に含まれるがサーバー上では利用不可なドライバも含めるのとは違って）。

server_supported_storage_drivers

これはサーバーの環境情報にサポートされているストレージドライバの情報を追加します。

event_lifecycle_requestor_address

lifecycle requestor に address のフィールドを追加します。

resources_gpu_usb

リソース API 内の ResourcesGPUCard (GPU エントリ) に USBAddress (usb_address) を追加します。

clustering_evacuation

クラスタメンバーを退避と復元するための POST /1.0/cluster/members/<name>/state エンドポイントを追加します。また設定キー cluster.evacuate と volatile.evacuate.origin も追加します。これらはそれぞれ退避の方法 (auto, stop or migrate) と移動したインスタンスのオリジンを設定します。

network_ovn_nat_address

これは LXD の ovn ネットワークに ipv4.nat.address と ipv6.nat.address 設定キーを追加します。これらのキーで OVN 仮想ネットワークからの外向きトラフィックのソースアドレスを制御します。これらのキーは OVN ネットワークのアップリンクネットワークが ovn.ingress_mode=routed という設定を持つ場合にのみ指定可能です。

network_bgp

これは LXD を BGP ルーターとして振る舞わせルートを bridge と ovn ネットワークに広告するようにします。

以下のグローバル設定が追加されます。

- core.bgp_address
- core.bgp_asn
- core.bgp_routerid

以下のネットワーク設定キーが追加されます（bridge と physical）。

- bgp.peers.<name>.address
- bgp.peers.<name>.asn
- bgp.peers.<name>.password
- bgp.ipv4.nexthop
- bgp.ipv6.nexthop

そして下記の NIC 特有な設定が追加されます（NIC type が bridged の場合）。

- ipv4.routes.external
- ipv6.routes.external

network_forward

これはネットワークアドレスのフォワード機能を追加します。bridge と ovn ネットワークで外部 IP アドレスを定義して対応するネットワーク内の内部 IP アドレス (複数指定可能) にフォワード出来ます。

custom_volume_refresh

ボリュームマイグレーションに refresh オプションのサポートを追加します。

network_counters_errors_dropped

これはネットワークカウンターに受信エラー数、送信エラー数とインバウンドとアウトバウンドのドロップしたパケット数を追加します。

metrics

これは LXD にメトリクスを追加します。実行中のインスタンスのメトリクスを OpenMetrics 形式で返します。

この拡張は次のエンドポイントを含みます。

- GET /1.0/metrics

image_source_project

POST /1.0/images に project フィールドを追加し、イメージコピー時にコピー元プロジェクトを設定できるようにします。

clustering_config

クラスタメンバーに config プロパティを追加し、キー・バリュー・ペアを設定可能にします。

network_peer

ネットワークピアリングを追加し、 OVN ネットワーク間のトラフィックが OVN サブシステムの外に出ずに通信できるようにします。

linux_sysctl

linux.sysctl.* 設定キーを追加し、ユーザーが一コンテナ内の一部のカーネルパラメータを変更できるようにします。

network_dns

組み込みの DNS サーバーとゾーン API を追加し、 LXD インスタンスに DNS レコードを提供します。

以下のサーバー設定キーが追加されます。

- core.dns_address

以下のネットワーク設定キーが追加されます。

- dns.zone.forward
- dns.zone.reverse.ipv4
- dns.zone.reverse.ipv6

以下のプロジェクト設定キーが追加されます。

- restricted.networks.zones

DNS ゾーンを管理するために下記の REST API が追加されます。

- /1.0/network-zones (GET, POST)
- /1.0/network-zones/<name> (GET, PUT, PATCH, DELETE)

ovn_nic_acceleration

OVN NIC に acceleration 設定キーを追加し、ハードウェアオフロードを有効にするのに使用できます。設定値は none または sriov です。

certificate_self_renewal

これはクライアント自身の信頼証明書の更新のサポートを追加します。

instance_project_move

これは POST /1.0/instances/NAME API に project フィールドを追加し、インスタンスをプロジェクト間で簡単に移動できるようにします。

storage_volume_project_move

これはストレージボリュームのプロジェクト間での移動のサポートを追加します。

cloud_init

これは以下のキーを含む project 設定キー名前空間を追加します。

- cloud-init.vendor-data
- cloud-init.user-data
- cloud-init.network-config

これはまた devlxd にインスタンスのデバイスを表示する /1.0/devices エンドポイントを追加します。

network_dns_nat

これはネットワークゾーン (DNS) に network.nat を設定オプションとして追加します。

デフォルトでは全てのインスタンスの NIC のレコードを生成するという現状の挙動になりますが、false に設定すると外部から到達可能なアドレスのレコードのみを生成するよう LXD に指示します。

database_leader

クラスタリーダーに設定される database-leader ロールを追加します。

instance_all_projects

全てのプロジェクトのインスタンス表示のサポートを追加します。

clustering_groups

クラスタメンバーのグループ化のサポートを追加します。

これは以下の新しいエンドポイントを追加します。

- /1.0/cluster/groups (GET, POST)
- /1.0/cluster/groups/<name> (GET, POST, PUT, PATCH, DELETE)

以下のプロジェクトの制限が追加されます。

- restricted.cluster.groups

ceph_rbd_du

Ceph ストレージプールに ceph.rbd.du という Boolean の設定を追加します。実行に時間がかかるかもしれない rbd du の呼び出しの使用を無効化できます。

instance_get_full

これは GET /1.0/instances/{name} に recursion=1 のモードを追加します。これは状態、スナップショット、バックアップの構造体を含む全てのインスタンスの構造体が取得できます。

qemu_metrics

これは security.agent.metrics という Boolean 値を追加します。デフォルト値は true です。false に設定するとメトリクスや他の状態の取得のために lxd-agent に接続することはせず、 QEMU からの統計情報に頼ります。

gpu_mig_uuid

NVIDIA 470+ ドライバ (例. MIG-74c6a31a-fde5-5c61-973b-70e12346c202) で使用される MIG UUID 形式のサポートを追加します。MIG- の接頭辞は省略できます。

この拡張が古い mig.gi と mig.ci パラメーターに取って代わります。これらは古いドライバとの互換性のため残されますが、同時には設定できません。

event_project

イベントの API にイベントが属するプロジェクトを公開します。

clustering_evacuation_live

`cluster.evacuate` への設定値 `live-migrate` を追加します。これはクラスタ退避の際にインスタンスのライブマイグレーションを強制します。

instance_allow_inconsistent_copy

`POST /1.0/instances` のインスタンスソースに `allow_inconsistent` フィールドを追加します。`true` の場合、`rsync` はコピーからインスタンスを生成するときに `Partial transfer due to vanished source files` (code 24) エラーを無視します。

network_state_ovn

これにより、`/1.0/networks/NAME/state` API に `ovn` セクションが追加されます。これには OVN ネットワークに関連する追加の状態情報が含まれます:

- chassis (シャーシ)

storage_volume_api_filtering

ストレージボリュームの GET リクエストの結果をフィルタリングする機能を追加します。

image_restrictions

この拡張機能は、イメージのプロパティに、イメージの制限やホストの要件を追加します。これらの要件はインスタンスとホストシステムとの互換性を決定するのに役立ちます。

storage_zfs_export

`zfs.export` を設定することで、プールのアンマウント時に zpool のエクスポートを無効にする機能を導入しました。

network_dns_records

network zones (DNS) API を拡張し、カスタムレコードの作成と管理機能を追加します。

これにより、以下が追加されます。

- `GET /1.0/network-zones/ZONE/records`
- `POST /1.0/network-zones/ZONE/records`
- `GET /1.0/network-zones/ZONE/records/RECORD`
- `PUT /1.0/network-zones/ZONE/records/RECORD`
- `PATCH /1.0/network-zones/ZONE/records/RECORD`
- `DELETE /1.0/network-zones/ZONE/records/RECORD`

storage_zfs_reserve_space

`quota/refquota` に加えて、ZFS プロパティの `reservation/refreservation` を設定する機能を追加します。

network_acl_log

ACL ファイアウォールのログを取得するための API `GET /1.0/networks-acls/NAME/log` を追加します。

storage_zfs_blocksize

ZFS ストレージボリュームに新しい `zfs.blocksize` プロパティを導入し、ボリュームのブロックサイズを設定できるようになります。

metrics_cpu_seconds

LXD が使用する CPU 時間をミリ秒ではなく秒単位で出力するように修正されたかどうかを検出するために使用されます。

instance_snapshot_never

`snapshots.schedule` に`@never` オプションを追加し、継承を無効にすることができます。

certificate_token

トラストストアに、トラストパスワードに代わる安全な手段として、トークンベースの証明書を追加します。

これは POST /1.0/certificates に token フィールドを追加します。

instance_nic_routed_neighbor_probe

これは routed NIC が親のネットワークが利用可能かを調べるために IP 近傍探索するのを無効化できるようにします。

ipv4.neighbor_probe と ipv6.neighbor_probe の NIC 設定を追加します。未指定の場合のデフォルト値は true です。

event_hub

これは event-hub というクラスタメンバーの役割と ServerEventMode 環境フィールドを追加します。

agent_nic_config

これを true に設定すると、仮想マシンの起動時に lxd-agent がインスタンスの NIC デバイスの名前と MTU を変更するための NIC 設定を適用します。

projects_restricted_intercept

restricted.container.intercept という設定キーを追加し通常は安全なシステムコールのインターセプションオプションを可能にします。

metrics_authentication

core.metrics_authentication というサーバー設定オプションを追加し /1.0/metrics のエンドポイントをクライアント認証無しでアクセスすることを可能にします。

images_target_project

コピー元とは異なるプロジェクトにイメージをコピーできるようにします。

cluster_migration_inconsistent_copy

POST /1.0/instances/<name> に allow_inconsistent フィールドを追加します。 true に設定するとクラスタメンバー間で不整合なコピーを許します。

cluster_ovn_chassis

ovn-chassis というクラスタロールを追加します。これはクラスタメンバーが OVN シャーシとしてどう振る舞うかを指定できるようにします。

container_syscall_intercept_sched_setscheduler

security.syscalls.intercept.sched_setscheduler を追加し、コンテナ内の高度なプロセス優先度管理を可能にします。

storage_lvm_thinpool_metadata_size

storage.thinpool_metadata_size により thin pool のメタデータボリュームサイズを指定できるようにします。

指定しない場合のデフォルトは LVM が適切な thin pool のメタデータボリュームサイズを選択します。

storage_volume_state_total

これは GET /1.0/storage-pools/{name}/volumes/{type}/{volume}/state API に total フィールドを追加します。

instance_file_head

/1.0/instances/NAME/file に HEAD を実装します。

instances_nic_host_name

instances.nic.host_name サーバー設定キーを追加します。これは random か mac を指定できます。指定しない場合のデフォルト値は random です。random に設定するとランダムなホストインタフェース名を使用します。mac に設定すると lxd1122334455 の形式で名前を生成します。

image_copy_profile

イメージをコピーする際にプロファイルの組を修正できるようにします。

container_syscall_intercept_sysinfo

`security.syscalls.intercept.sysinfo` を追加し `sysinfo` システムコールで cgroup ベースのリソース使用状況を追加できるようにします。

clustering_evacuation_mode

退避リクエストに `mode` フィールドを追加します。これにより従来 `cluster.evacuate` で設定されていた退避モードをオーバーライドできます。

resources_pci_vpd

PCI リソースエントリに VPS 構造体を追加します。この構造体には完全な製品名と追加の設定キーバリューペアを含むベンダー提供のデータが含まれます。

qemu_raw_conf

生成された qemu.conf の指定したセクションをオーバライドするための `raw.qemu.conf` 設定キーを追加します。

storage_cephfs_fscache

CephFS プール上の `fscache/cachefilesd` をサポートするための `cephfs.fscache` 設定オプションを追加します。

network_load_balancer

これはネットワークのロードバランサー機能を追加します。ovn ネットワークで外部 IP アドレス上にポートを定義し、ポートから対応するネットワーク内部の単一または複数の内部 IP にトラフィックをフォワードできます。

vsock_api

これは双方向の vsock インタフェースを導入し、 lxd-agent と LXD サーバーがよりよく連携できるようにします。

instance_ready_state

インスタンスに新しく Ready 状態を追加します。これは devlxd を使って設定できます。

network_bgp_holdtime

特定のピアの BGP ホールドタイムを制御するために bgp.peers.<name>.holdtime キーを追加します。

storage_volumes_all_projects

全てのプロジェクトのストレージボリュームを一覧表示できるようにします。

metrics_memory_oom_total

/1.0/metrics API に lxd_memory_OOM_kills_total メトリックを追加します。メモリーキラー (OOM) が発動された回数を報告します。

storage_buckets

storage bucket API を追加します。ストレージプールのために S3 オブジェクトストレージのバケットの管理をできるようにします。

storage_buckets_create_credentials

これはバケット作成時に管理者の初期クレデンシャルを返すようにストレージバケット API を更新します。

metrics_cpu_effective_total

これは lxd_cpu_effective_total メトリックを /1.0/metrics API に追加します。有効な CPU の総数を返します。

projects_networks_restricted_access

プロジェクト内でアクセスできるネットワークを (カンマ区切りリストで) 示す restricted.networks.access プロジェクト設定キーを追加します。指定しない場合は、全てのネットワークがアクセスできます (後述の restricted.devices.nic 設定でも許可されている場合)。

これはまたプロジェクトの restricted.devices.nic 設定で制御されるネットワークアクセスにも変更をもたらします。

- restricted.devices.nic が managed に設定される場合 (未設定時のデフォルト), マネージドネットワークのみがアクセスできます。

- restricted.devices.nic が allow に設定される場合、全てのネットワークがアクセスできます (restricted.networks.access 設定に依存)。
- restricted.devices.nic が block に設定される場合、どのネットワークにもアクセスできません。

storage_buckets_local

これは core.storage_buckets_address グローバル設定を指定することでローカルストレージプール上のストレージバケットを使用できるようにします。

loki

これはライフサイクルとロギングのイベントを Loki サーバーに送れるようにします。

以下のグローバル設定キーを追加します。

- loki.api.ca_cert: イベントを Loki サーバーに送る際に使用する CA 証明書。
- loki.api.url: Loki サーバーの URL。
- loki.auth.username と loki.auth.password: Loki が BASIC 二症を有効にしたリバースプロキシの背後にいる場合に使用。
- loki.labels: Loki イベントのラベルに使用されるカンマ区切りリストの値。
- loki.loglevel: Loki サーバーに送るイベントの最低のログレベル。
- loki.types: Loki サーバーに送られるイベントのタイプ (lifecycle および/または logging)。

acme

ACME サポートを追加します。これにより Let's Encrypt や他の ACME サービスを使って証明書を発行できます。

以下のグローバルの設定キーを追加します。

- acme.domain: 証明書を発行するドメイン。
- acme.email: ACME サービスのアカウントに使用する email アドレス。
- acme.ca_url: ACME サービスのディレクトリ URL、デフォルトは https://acme-v02.api.letsencrypt.org/directory。

また以下のエンドポイントを追加します。これは HTTP-01 チャレンジで必要です。

- /.well-known/acme-challenge/<token>

internal_metrics

これはメトリクスのリストに内部メトリクスを追加します。以下を含みます。

- 実行したオペレーションの総数
- アクティブな警告の総数
- デーモンの uptime (秒数)
- Go のメモリ統計
- goroutine の数

cluster_join_token_expiry

クラスタジョイントークンに有効期限を追加します。デフォルトは 3 時間ですが、cluster.join_token_expiry 設定キーで変更できます。

remote_token_expiry

リモートの追加ジョイントークンに有効期限を追加します。core.remote_token_expiry 設定キーで変更できます。デフォルトは無期限です。

storage_volumes_created_at

この変更によりストレージボリュームとそのスナップショットの作成日時を保管するようになります。

これは StorageVolume と StorageVolumeSnapshot API タイプに CreatedAt フィールドを追加します。

cpu_hotplug

これは VM に CPU ホットプラグを追加します。CPU ピンニング使用時はホットプラグは無効になります。CPU ピンニングには NUMA デバイスのホットプラグも必要ですが、これはできないためです。

projects_networks_zones

プロジェクトに features.networks.zones の機能を追加します。これはネットワークゾーンが作成されたときにどのプロジェクトに関連付けされるかを変更します。以前はネットワークゾーンは features.networks の値に従っており、つまりネットワークと同じプロジェクト内に作られていました。

この拡張により features.networks から切り離され、複数のプロジェクトがデフォルトプロジェクト (例えば features.networks=false と設定) 内のネットワークを共有できるようになり、共有されたネットワークに対してプロジェクト指向の (プロジェクト内のインスタンスのアドレスのみを含むような)「ビュー」を提供するプロジェクト固有の DNS ゾーンを持てるようになりました。

これはまたネットワークの dns.zone.forward にも変更を及ぼし、複数のゾーンに共有ネットワークを関連付けるために (プロジェクト毎に最大 1 つの)DNS ゾーン名のカンマ区切りリストを指定できるようになりました。

dns.zone.reverse.*設定は変更無しで、引き続き単一の DNS ゾーンのみが設定できます。しかし結果として生成されるゾーンの内容は、正引きゾーンの 1 つを経由してネットワークを参照する全てのプロジェクトからのアドレスをカバーするような PTR レコードを含むようになりました。

features.networks=true の設定を持つ既存のプロジェクトは自動的に features.networks.zones=true が設定されますが、新規のプロジェクトは明示的に設定する必要があります。

instance_nic_txqueuelength

NIC デバイスの txqueuelen パラメータを制御する txqueuelen キーを追加します。

cluster_member_state

GET /1.0/cluster/members/<member>/state API エンドポイントとそれに関連する ClusterMemberState API レスポンスタイプを追加します。

instances_placement_scriptlet

Starlark スクリプトレットを LXD に提供し、クラスタ内の新規インスタンスの配置を制御するカスタムロジックを使えるようにします。

Starlark スクリプトレットは新しいグローバル設定オプション instances.placement.scriptlet により LXD に提供されます。

storage_pool_source_wipe

ストレージプールに source.wipe ブール値を追加し、LXD は要求されたディスクのパーティションヘッダーを消去する必要があることを示します。これにより、既存のファイルシステムがあることによる潜在的な失敗を回避できるようになります。

zfs_block_mode

これにより、ZFS ブロック filesystem ボリュームを使用して、ZFS の上に異なるファイルシステムを使用することができるようになります。

これにより、ZFS ストレージプールに以下の新しい設定オプションが追加されます：

- volume.zfs.block_mode
- volume.block.mount_options

- volume.block.filesystem

instance_generation_id

インスタンスの generation ID のサポートが追加されます。VM またはコンテナの generation ID は、インスタンスの時間的な世代が後ろに移動するたびに変更されます。現時点では、 generation ID は VM タイプのインスタンスを通じてのみ公開されています。これにより、VM ゲスト OS は、既に発生した可能性のある状態の複製を回避するために必要な状態を再初期化できます：

- volatile.uuid.generation

disk_io_cache

これは、ディスクデバイスに新しい io.cache プロパティを導入し、VM のキャッシング動作を上書きするために使用できます。

amd_sev

AMD SEV (Secure Encrypted Virtualization) のサポートを追加します。ゲスト VM のメモリを暗号化するのに使用できます。

これは SEV 暗号化に以下の新しい設定オプションを追加します。

- security.sev : (bool) この VM で SEV を有効化するか
- security.sev.policy.es : (bool) この VM で SEV-ES を有効化するか
- security.sev.session.dh : (string) ゲストオーナーの base64 エンコードされた Diffie-Hellman キー
- security.sev.session.data : (string) ゲストオーナーの base64 エンコードされた session blob

storage_pool_loop_resize

これはループファイルをバックエンドとするストレージプールを、プールの size 設定を変更することでサイズを拡大できるようにします。

migration_vm_live

これは VM で QEMU から QEMU へのライブマイグレーションを (Ceph を含む) 共有されたストレージと共有されないストレージプールの両方で実行するサポートを追加します。

これはさらにマイグレーションの CRIUType protobuf フィールドに 3 という CRIUType_VM_QEMU の値を追加します。

ovn_nic_nesting

これは同じインスタンスの ovn NIC を別の ovn NIC 内にネストするサポートを追加します。これは OVN ロジカルスイッチポートを VLAN タグを使用する別の OVN NIC の内側にトンネルできるようにします。

この機能は nested プロパティを使って親の NIC 名を指定し、vlan プロパティでトンネルに使用する VLAN ID を指定することで設定できます。

oidc

OpenID Connect (OIDC) 認証のサポートを追加します。

これは以下の設定キーを追加します:

- oidc.issuer
- oidc.client.id
- oidc.audience

network_ovn_l3only

これは ovn ネットワークを "layer 3 only" モードに設定する能力を追加します。このモードは ipv4.l3only と ipv6.l3only 設定オプションを使うことで IPv4 または IPv6 のレベルで有効化できます。

このモードを有効にすると、ネットワークは以下のように変更されます:

- 仮想ルータの内部ポートアドレスが単一ホストのネットマスクで設定されます (例 IPv4 では /32 あるいは IPv6 では /128)。
- アクティブなインスタンスの NIC アドレスへの静的ルートが仮想ルータに追加されます。

- アクティブでないアドレス向けのパケットがアップリンクのネットワークからエスケープされるのを防ぐために、内部のサブネット全体への破棄ルートが仮想ルータに追加されます。
- 255.255.255.255 のネットマスクがインスタンス設定で使用されるように DHCPv4 サーバーが設定されます。

ovn_nic_acceleration_vdpa

これは ovn_nic_acceleration API 拡張をアップデートします。OVN NIC の acceleration 設定キーが Virtual Data Path Acceleration (VDPA) をサポートするための vdpa という値を受け付けられるようになります。

cluster_healing

これにより、オフラインのクラスタメンバーを自動的に退避させるクラスタヒーリングが追加されます。

次の新しい設定キーが追加されます：

- cluster.healing_threshold

この設定キーは整数を取り、0（デフォルト）に設定することで無効化できます。設定された場合、その値はオフラインのクラスタメンバーが退避させられる閾値を表します。もし値が cluster.offline_threshold よりも低い場合、その値が代わりに使用されます。

オフラインのクラスタメンバーが退避させられると、リモートバックアップされたインスタンスのみが移行されます。ローカルインスタンスは、クラスタメンバーがオフラインになった際にそれらを移行する方法がないため、無視されます。

instances_state_total

この拡張は、インスタンスの状態 API の一部である InstanceStateDisk と InstanceStateMemory に新しい total フィールドを追加します。

auth_user

メイン API エンドポイントに現在のユーザーの詳細情報を追加します。

以下の項目を追加します。

- auth_user_name
- auth_user_method

security_csm

CSM (Compatibility Support Module) の使用を制御する security.csm 設定キーを追加し、レガシーなオペレーティングシステムを LXD VM 内で稼働できるようにします。

instances_rebuild

この拡張はインスタンスを同じイメージ、別のイメージ、あるいは空のイメージで再構築できるようにします。POST /1.0/instances/<name>/rebuild?project=<project> API エンドポイントと lxc rebuild CLI コマンドを新しく追加します。

numa_cpu_placement

これは CPU の組を指定の NUMA ノードの組内に配置できるようにします。

以下の設定キーを追加します。

- limits.cpu.nodes : (string) (limits.cpu の動的な値により選ばれた)CPU を配置する NUMA ノード ID または NUMA ノード ID の範囲のカンマ区切りリスト。

3.12.4 インスタンス〜ホスト間の通信

ホストされているワークロード (インスタンス) とそのホストのコミュニケーションは厳密には必要とされているわけではないですが、とても便利な機能です。

LXD ではこの機能は /dev/lxd/sock というノードを通して実装されており、このノードは全ての LXD のインスタンスに対して作成、セットアップされます。

このファイルはインスタンス内部のプロセスが接続できる Unix ソケットです。マルチスレッドで動いているので複数のクライアントが同時に接続できます。

注釈: インスタンスのソケットへのアクセスを許可するには *security.devlxd* を true (これがデフォルトです) に設定する必要があります。

実装詳細

ホストでは LXD は /var/lib/lxd/devlxd/sock をバインドして新しいコネクションのリッスンを開始します。

このソケットは、LXD が開始させたすべてのインスタンス内の /dev/lxd/sock に公開されます。

4096 を超えるインスタンスを扱うのに単一のソケットが必要です。そうでなければ、LXD は各々のインスタンスに異なるソケットをバインドする必要があり、ファイルディスクリプタ数の上限にすぐ到達してしまいます。

認証

/dev/lxd/sock への問い合わせは依頼するインスタンスに関連した情報のみを返します。リクエストがどこから来たかを知るために、 LXD は初期のソケットのユーザクレデンシャルを取り出し、 LXD が管理しているインスタンスのリストと比較します。

プロトコル

/dev/lxd/sock のプロトコルは JSON メッセージを用いたプレーンテキストの HTTP であり、 LXD プロトコルのローカル版に非常に似ています。

メインの LXD API とは異なり、 /dev/lxd/sock API にはバックグラウンド処理と認証サポートはありません。

REST-API

API の構造

- /
  - /1.0
    - /1.0/config
      - /1.0/config/{key}
    - /1.0/devices
    - /1.0/events
    - /1.0/images/{fingerprint}/export
    - /1.0/meta-data

API の詳細

/

GET

- 説明: サポートされている API のリスト
- 出力: サポートされている API エンドポイント URL のリスト (デフォルトでは ['/1.0']`)

戻り値:

```
[
    "/1.0"
]
```

/1.0

GET

- 説明: 1.0 API についての情報
- 出力: dict 形式のオブジェクト

戻り値:

```
{
    "api_version": "1.0",
    "location": "foo.example.com",
    "instance_type": "container",
    "state": "Started",
}
```

PATCH

- 説明: インスタンスの状態を更新する (有効な状態は `Ready` と `Started`)
- 戻り値: 無し

入力:

```
{
   "state": "Ready"
}
```

/1.0/config

GET

- 説明: 設定キーの一覧
- 出力: 設定キー URL のリスト

設定キーの名前はインスタンスの設定の名前と一致するようにしています。しかし、設定の namespace の全てが /dev/lxd/sock にエクスポートされているわけではありません。現在は cloud-init.* と user.* キーのみがインスタンスにアクセス可能となっています。

現時点ではインスタンスが書き込み可能な名前空間はありません。

戻り値:

```
[
    "/1.0/config/user.a"
]
```

/1.0/config/<KEY>

GET

- 説明: そのキーの値
- 出力: プレーンテキストの値

戻り値:

```
blah
```

/1.0/devices

GET

- 説明: インスタンスのデバイスのマップ
- 出力: dict

戻り値:

```
{
    "eth0": {
        "name": "eth0",
        "network": "lxdbr0",
        "type": "nic"
    },
    "root": {
        "path": "/",
        "pool": "default",
        "type": "disk"
    }
}
```

/1.0/events

GET

- 説明: この API ではプロトコルが WebSocket にアップグレードされます。
- 出力: 無し (イベントのフローが終わることがなくずっと続く)

サポートされる引数は以下の通りです。

- type: 購読する通知の種別のカンマ区切りリスト (デフォルトは all)

通知の種別には以下のものがあります。

- `config` (あらゆる `user.*` 設定キーの変更)
- `device` (あらゆるデバイスの追加、変更、削除)

この API は決して終了しません。それぞれの通知は別々の JSON の dict として送られます。

```
{
    "timestamp": "2017-12-21T18:28:26.846603815-05:00",
    "type": "device",
    "metadata": {
        "name": "kvm",
        "action": "added",
        "config": {
            "type": "unix-char",
            "path": "/dev/kvm"
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
}
```

```
{
    "timestamp": "2017-12-21T18:28:26.846603815-05:00",
    "type": "config",
    "metadata": {
        "key": "user.foo",
        "old_value": "",
        "value": "bar"
    }
}
```

/1.0/images/<FINGERPRINT>/export

GET

- 説明: 公開されたあるいはキャッシュされたイメージをホストからダウンロードする
- 出力: 生のイメージあるいはエラー
- アクセス権: `security.devlxd.images` を `true` に設定する必要があります

戻り値:

```
LXD デーモン API の /1.0/images/<FINGERPRINT>/export を参照してください。
```

/1.0/meta-data

GET

- 説明: cloud-init と互換性のあるコンテナのメタデータ
- 出力: cloud-init のメタデータ

戻り値:

```
#cloud-config
instance-id: af6a01c7-f847-4688-a2a4-37fddd744625
local-hostname: abc
```

3.12.5 イベント

イントロダクション

イベントとは LXD 上で発生したアクションに関するメッセージです。 `/1.0/events` の API エンドポイントを直接使うか `lxc monitor` コマンドを使うことでウェブソケットに接続しログとライフサイクルメッセージがストリーム出力されます。

イベント種別

LXD は現在 3 つのイベント種別をサポートします。

- `logging`: サーバーのログレベルに関係なく全てのログメッセージを表示します。
- `operation`: 作成から完了までの（状態と進捗メタデータの更新を含む）全ての実行中のオペレーションを表示します。
- `lifecycle`: LXD 上で発生する特定のアクションの監査証跡を表示します。

イベント構造

例:

```
location: cluster_name
metadata:
  action: network-updated
  requestor:
    protocol: unix
    username: root
  source: /1.0/networks/lxdbr0
timestamp: "2021-03-14T00:00:00Z"
type: lifecycle
```

- `location`: クラスタメンバー名（クラスタであれば）。
- `timestamp`: RFC3339 形式のイベント発生時刻。
- `type`: イベント種別（`logging`, `operation`, `lifecycle` のいずれか）。
- `metadata`: 特定のイベント種別に関する情報。

logging イベントの構造

- message: ログメッセージ。
- level: ログのログレベル。
- context: イベントに含まれる追加情報。

operation イベントの構造

- id: オペレーションの UUID
- class: オペレーション種別（task, token, websocket のいずれか）。
- description: オペレーションの説明。
- created_at: オペレーションの作成日時。
- updated_at: オペレーションの更新日時。
- status: オペレーションの現在の状態。
- status_code: オペレーションのステータスコード。
- resources: このオペレーションで影響を受けるリソース。
- metadata: オペレーション特有のメタデータ。
- may_cancel: オペレーションがキャンセル可能か。
- err: オペレーションのエラーメッセージ。
- location: クラスタメンバー名（クラスタであれば）。

ライフサイクルイベントの構造

- action: 発生したライフサイクルアクション。
- requestor: 誰がリクエストを作成したかの情報（該当するものがあれば）。
- source: アクションの対象のパス。
- context: イベントに含まれる追加情報。

サポートされるライフサイクルイベント

名前	説明
certificate-created	サーバーのトラスト・ストアに新しい証明書が追加された。
certificate-deleted	トラスト・ストアから証明書が削除された。
certificate-updated	証明書の設定が更新された。
cluster-certificate-updated	クラスタ全体の証明書が変更された。
cluster-disabled	このマシンに対してクラスタリングが無効化された。
cluster-enabled	このマシンに対してクラスタリングが有効化された。
cluster-group-created	新しいクラスタグループが作成された。
cluster-group-deleted	クラスタグループが削除された。
cluster-group-renamed	クラスタグループがリネームされた。
cluster-group-updated	クラスタグループが更新された。
cluster-member-added	新しいマシンがクラスタに参加した。
cluster-member-removed	クラスタからクラスタメンバーが削除された。
cluster-member-renamed	クラスタメンバーがリネームされた。
cluster-member-updated	クラスタメンバーの設定が変更された。
cluster-token-created	クラスタメンバー追加のための参加トークンが作成された。
config-updated	サーバーの設定が変更された。
image-alias-created	既存イメージのエイリアスが作成された。
image-alias-deleted	既存イメージのエイリアスが削除された。
image-alias-renamed	既存イメージのエイリアスがリネームされた。
image-alias-updated	イメージ・エイリアスの設定が変更された。
image-created	イメージ・ストアに新しいイメージが追加された。
image-deleted	イメージ・ストアからイメージが削除された。
image-refreshed	ローカルのイメージコピーが現在のソースイメージのバージョンに更新された。
image-retrieved	raw イメージファイルがサーバーからダウンロードされた。
image-secret-created	イメージ取得用のワンタイム・キーが作成された。
image-updated	イメージの設定が変更された。
instance-backup-created	インスタンスのバックアップが作成された。
instance-backup-deleted	インスタンスのバックアップが削除された。
instance-backup-renamed	インスタンスのバックアップがリネームされた。
instance-backup-retrieved	raw インスタンス・バックアップ・ファイルがダウンロードされた。
instance-console	インスタンスのコンソールに接続された。
instance-console-reset	コンソール・バッファーがリセットされた。
instance-console-retrieved	コンソール・ログがダウンロードされた。
instance-created	新しいインスタンスが作成された。
instance-deleted	インスタンスが削除された。
instance-exec	インスタンス上でコマンドが実行された。

表 4 – 前のページか

名前	説明
instance-file-deleted	インスタンス上のファイルが削除された。
instance-file-pushed	インスタンスにファイルがアップロードされた。
instance-file-retrieved	インスタンスからファイルがダウンロードされた。
instance-log-deleted	インスタンスの指定のログファイルが削除された。
instance-log-retrieved	インスタンスの指定のログファイルがダウンロードされた。
instance-metadata-retrieved	インスタンスのイメージメタデータがダウンロードされた。
instance-metadata-template-created	インスタンスの新しいイメージテンプレートファイルが作成された。
instance-metadata-template-deleted	インスタンスのイメージテンプレートファイルが削除された。
instance-metadata-template-retrieved	インスタンスのイメージテンプレートファイルがダウンロードされた。
instance-metadata-updated	インスタンスのイメージメタデータが変更された。
instance-paused	インスタンスが休止状態にされた。
instance-ready	インスタンスが準備完了になった。
instance-renamed	インスタンスがリネームされた。
instance-restarted	インスタンスが再起動された。
instance-restored	インスタンスがスナップショットから復元された。
instance-resumed	インスタンスが休止状態から復帰された。
instance-shutdown	インスタンスがシャットダウンされた。
instance-snapshot-created	インスタンスのスナップショットが作成された。
instance-snapshot-deleted	インスタンスのスナップショットが削除された。
instance-snapshot-renamed	インスタンスのスナップショットがリネームされた。
instance-snapshot-updated	インスタンス・スナップショットの設定が変更された。
instance-started	インスタンスが起動された。
instance-stopped	インスタンスが停止された。
instance-updated	インスタンスの設定が変更された。
network-acl-created	新しいネットワーク ACL が作成された。
network-acl-deleted	ネットワーク ACL が削除された。
network-acl-renamed	ネットワーク ACL がリネームされた。
network-acl-updated	ネットワーク ACL の設定が変更された。
network-created	ネットワークデバイスが作成された。
network-deleted	ネットワークデバイスが削除された。
network-forward-created	ネットワークフォワードが作成された。
network-forward-deleted	ネットワークフォワードが削除された。
network-forward-updated	ネットワークフォワードが更新された。
network-peer-created	ネットワークピアが作成された。
network-peer-deleted	ネットワークピアが削除された。
network-peer-updated	ネットワークピアがリネームされた。
network-renamed	ネットワークデバイスがリネームされた。

表 4 – 前のページか

名前	説明
network-updated	ネットワークデバイスの設定が変更された。
network-zone-created	ネットワークゾーンが作成された。
network-zone-deleted	ネットワークゾーンが削除された。
network-zone-record-created	ネットワークゾーンレコードが作成された。
network-zone-record-deleted	ネットワークゾーンレコードが削除された。
network-zone-record-updated	ネットワークゾーンレコードが更新された。
network-zone-updated	ネットワークゾーンが更新された。
operation-cancelled	オペレーションがキャンセルされた。
profile-created	新しいプロファイルが作成された。
profile-deleted	プロファイルが削除された。
profile-renamed	プロファイルがリネームされた。
profile-updated	プロファイルの設定が変更された。
project-created	新しいプロジェクトが作成された。
project-deleted	プロジェクトが削除された。
project-renamed	プロジェクトがリネームされた。
project-updated	プロジェクトの設定が変更された。
storage-pool-created	新しいストレージプールが作成された。
storage-pool-deleted	ストレージプールが削除された。
storage-pool-updated	ストレージプールの設定が変更された。
storage-volume-backup-created	ストレージボリュームの新しいバックアップが作成された。
storage-volume-backup-deleted	ストレージボリュームのバックアップが削除された。
storage-volume-backup-renamed	ストレージボリュームのバックアップがリネームされた。
storage-volume-backup-retrieved	ストレージボリュームのバックアップがダウンロードされた。
storage-volume-created	新しいストレージボリュームが作成された。
storage-volume-deleted	ストレージボリュームが削除された。
storage-volume-renamed	ストレージボリュームがリネームされた。
storage-volume-restored	ストレージボリュームがスナップショットから復元された。
storage-volume-snapshot-created	新しいストレージボリュームスナップショットが作成された。
storage-volume-snapshot-deleted	ストレージボリュームのスナップショットが削除された。
storage-volume-snapshot-renamed	ストレージボリュームのスナップショットがリネームされた。
storage-volume-snapshot-updated	ストレージボリュームのスナップショットの設定が変更された。
storage-volume-updated	ストレージボリュームの設定が変更された。
warning-acknowledged	警告の状態が "acknowledged" （確認済み）に設定された。
warning-deleted	警告が削除された。
warning-reset	警告の状態が "new" （新規）に設定された。

3.13 内部動作とデバッグ

3.13.1 デーモンの動作

この仕様書は *LXD* **デーモン**の振る舞いの一部を取り扱います。

起動

起動する度に LXD はディレクトリ構造が存在することをチェックします。もし存在しない場合は、必要なディレクトリを作成し、キーペアを生成し、データベースを初期化します。

ひとたびデーモンが動作の準備が出来ると、 LXD はデータベース内のインスタンスのテーブルから対象のテーブルを検索し、電源状態が実際の状態と異なっていないかを確認します。もしインスタンスの電源状態が稼働中と記録されているのにインスタンスが稼働していない場合は LXD はそのインスタンスを開始します。

シグナル処理

SIGINT, SIGQUIT, SIGTERM

これらのシグナルについては LXD は一時的に停止し、後に再開してインスタンスの処理を継続することを想定しています。

インスタンスは稼働し続けて LXD は全ての接続を閉じ、クリーンな状態で終了するでしょう。

SIGPWR

LXD にホストがシャットダウンしようとしていることを伝えます。

LXD は全てのインスタンスをクリーンにシャットダウンしようと試みます。30 秒後、LXD は残りのインスタンスを kill します。

ホストがリブート完了後に LXD がインスタンスを元の状態に戻せるように、データベース内のインスタンスのテーブルの `power_state` カラムにインスタンスの元の電源状態を記録しておきます。

SIGUSR1

メモリプロファイルを`--memprofile` で指定したファイルにダンプします。

3.13.2 LXD をデバッグするには

インスタンスの問題をデバッグする際の情報については、**インスタンスの起動に失敗する問題のトラブルシューティング方法**を参照してください。

lxc と lxd のデバッグ

lxc と lxd のコードをトラブルシューティングするのに役立ついくつかの異なる方法を説明します。

lxc --debug

クライアントのどのコマンドにも --debug フラグを追加することで内部についての追加情報を出力することができます。もし有用な情報がない場合はログ出力の呼び出しで追加することができます。

```
logger.Debugf("Hello: %s", "Debug")
```

lxc monitor

このコマンドはメッセージがリモートのサーバーに現れるのをモニターします。

ローカルソケット経由での REST API

サーバーサイドで LXD とやりとりするのに最も簡単な方法はローカルソケットを経由することです。以下のコマンドは GET /1.0 にアクセスし、jq ユーティリティを使って JSON を人間が読みやすいように整形します。

```
curl --unix-socket /var/lib/lxd/unix.socket lxd/1.0 | jq .
```

あるいは snap ユーザーの場合は

```
curl --unix-socket /var/snap/lxd/common/lxd/unix.socket lxd/1.0 | jq .
```

利用可能な API については *RESTful API* をご参照ください。

HTTPS 経由での REST API

LXD **への** *HTTPS* **接続**には、有効なクライアント証明書が必要で、最初の lxc remote add で生成されます。この証明書は、認証と暗号化のための接続ツールに渡す必要があります。

必要に応じて、openssl を使って証明書 (~/.config/lxc/client.crt または Snap ユーザーの場合~/snap/lxd/common/config/client.crt) を調べることができます：

```
openssl x509 -text -noout -in client.crt
```

表示される行の中に以下のようなものがあるはずです：

```
Certificate purposes:
SSL client : Yes
```

コマンドラインツールを使う

```
wget --no-check-certificate --certificate=$HOME/.config/lxc/client.crt --private-key=
↪$HOME/.config/lxc/client.key -q0 - https://127.0.0.1:8443/1.0

# または snap ユーザーの場合
wget --no-check-certificate --certificate=$HOME/snap/lxd/common/config/client.crt --
↪private-key=$HOME/snap/lxd/common/config/client.key -q0 - https://127.0.0.1:8443/1.0
```

ブラウザを使う

いくつかのブラウザ拡張はウェブのリクエストを作成、修正、リプレイするための便利なインタフェースを提供しています。LXD サーバーに対して認証するには `lxc` のクライアント証明書をインポート可能な形式に変換しブラウザにインポートしてください。

例えば Windows で利用可能な形式の `client.pfx` を生成するには以下のようにします。

```
openssl pkcs12 -clcerts -inkey client.key -in client.crt -export -out client.pfx
```

上記のコマンドを実行し、(訳注：変換後の証明書をインポートしてから) ブラウザで `https://127.0.0.1:8443/1.0` を開けば期待通り動くはずです。

LXD データベースをデバッグ

グローバル**データベース**のファイルは LXD のデータディレクトリ (例 `/var/lib/lxd/database/global` か snap ユーザーは `/var/snap/lxd/common/lxd/database/global`) の `./database/global` サブディレクトリの下に格納されます。

クラスタの各メンバーもそのメンバー固有の何らかのデータを保持する必要があるため、LXD は通常の SQLite のデータベース (「ローカル」データベース) も使用します。これは `./database/local.db` に置かれます。

アップグレードの前にはグローバルデータベースのディレクトリとローカルデータベースのファイルのバックアップが作成され、 `.bak` のサフィックス付きでタグ付けされます。アップグレード前の状態に戻す必要がある場合は、このバックアップを使うことができます。

データベースのデータとスキーマをダンプする

データベースのデータまたはスキーマの SQL テキスト形式でのダンプを取得したい場合は、`lxd sql <local|global> [.dump|.schema]` コマンドを使ってください。これにより sqlite3 コマンドラインツールの `.dump` または `.schema` ディレクティブと同じ出力を生成できます。

コンソールからカスタムクエリを実行する

ローカルまたはグローバルデータベースに SQL クエリ (例 `SELECT`, `INSERT`, `UPDATE`) を実行する必要がある場合、`lxd sql` コマンドを使うことができます (詳細は `lxd sql --help` を実行してください)。

ただ、これが必要になるのは壊れたアップデートかバグからリカバーするときだけでしょう。その場合、まず LXD チームに相談してみてください (GitHub のイシューを作成するかフォーラムに投稿)。

LXD デーモン起動時にカスタムクエリを実行する

SQL のデータマイグレーションのバグあるいは関連する問題のためにアップグレード後に LXD デーモンが起動に失敗する場合、壊れたアップデートを修復するクエリを含んだ `.sql` ファイルを作成することで、その状況からリカバーできる可能性があります。

ローカルデータベースに対して修復を実行するには、修復に必要なクエリを含む `./database/patch.local.sql` というファイルを作成してください。同様にグローバルデータベースの修復には `./database/patch.global.sql` というファイルを作成してください。

これらのファイルはデーモンの起動シーケンスの非常に早い段階で読み込まれ、クエリが成功したときは削除されます (クエリは SQL トランザクション内で実行されるので、クエリが失敗したときにデータベースの状態が変更されることはありません)。

上記の通り、まず LXD チームに相談してみてください。

クラスタデータベースをディスクに同期

クラスタデータベースの内容をディスクにフラッシュしたいなら、`lxd sql global .sync` コマンドを使ってください。これは通常の SQLite 形式のデータベースのファイルを`./database/global/db.bin` に書き込みます。その後 `sqlite3` コマンドラインツールを使って中身を見ることができます。

3.13.3 環境変数

以下の環境変数を設定することで、LXD のクライアントとデーモンをユーザーの環境に適合させることができ、いくつかの高度な機能を有効または無効にすることができます。

注釈: snap 版の LXD をお使いの場合はこれらの環境変数は利用できません。

クライアントとサーバー共通の環境変数

名前	説明
`LXD_DIR`	LXD のデータディレクトリ
`PATH`	実行ファイルの検索対象のパスのリスト
`http_proxy`	HTTP 用のプロキシサーバーの URL
`https_proxy`	HTTPS 用のプロキシサーバーの URL
`no_proxy`	プロキシが不要なドメイン、IP アドレスあるいは CIDR レンジのリスト

クライアントの環境変数

名前	説明
`EDITOR`	使用するテキストエディタ
`VISUAL`	(`EDITOR` が設定されてないときに) 使用するテキストエディタ
`LXD_CONF`	LXC 設定ディレクトリーのパス
`LXD_GLOBAL_CONF`	LXC グローバル設定ディレクトリーのパス
`LXC_REMOTE`	使用するリモートの名前（設定されたデフォルトのリモートよりも優先されます）

サーバーの環境変数

名前	説明
LXD_EXEC_P	(サブコマンド実行時に使用される)LXD 実行ファイルのフルパス
LXD_LXC_TE	LXC テンプレート設定ディレクトリ
LXD_SECURI	false に設定すると AppArmor を無効にします
LXD_UNPRIV	true に設定すると非特権コンテナしか作れなくなるように強制します。LXD_UNPRIVILEGED_ONLY を設定する前に作られた特権コンテナは引き続き特権を持つことに注意してください。このオプションを LXD デーモンを最初にセットアップするときに設定するのが実用的です。
LXD_OVMF_P	OVMF_CODE.fd と OVMF_VARS.ms.fd を含む OVMF ビルドへのパス
LXD_SHIFTF	shiftfs のサポートを無効にする (従来の UID シフトを試す際に有用です)
LXD_IDMAPP	idmap を使ったマウントを無効にする (従来の UID シフトを試す際に有用です)
LXD_DEVMON	デバイスモニターでモニターするパス。主にテスト用。

3.13.4 システムコールのインターセプション

LXD では非特権コンテナで、いくつか特定のシステムコールをインターセプトできます。もし、それが安全であると見なせるのであれば、ホスト上で特権を昇格させて実行します。

これを行うことで、対象のシステムコールではパフォーマンスに影響があり、LXD ではリクエストを評価するための作業が必要となり、もし許可されれば昇格した特権で実行されます。

特定のシステムコールインターセプションのオプションの有効化はコンテナの設定オプションを使ってコンテナ単位で行われます。

利用できるシステムコール

mknod / mknodat

mknod と mknodat システムコールを使用して、色々なスペシャルファイルを作成できます。

もっとも一般的にはコンテナ内部で、ブロックデバイスやキャラクターデバイスを作成するために呼び出されます。このようなデバイスを作成することは、非特権コンテナ内では許可されません。これは、ディスクやメモリのようなリソースに直接書き込みのアクセスを許可することになり、特権を昇格するのに非常に簡単な方法であるためです。

しかし、作成しても安全であるファイルもあります。このような場合に、システムコールをインターセプトすることで、特定の処理のブロックが解除され、非特権コンテナ内部で実行できるようになります。

現時点で許可されているデバイスは次のものです:

- overlayfs whiteout (char 0:0)
- /dev/console (char 5:1)
- /dev/full (char 1:7)
- /dev/null (char 1:3)
- /dev/random (char 1:8)
- /dev/tty (char 5:0)
- /dev/urandom (char 1:9)
- /dev/zero (char 1:5)

キャラクターデバイス以外のすべてのファイルタイプは、現時点では通常通りカーネルに送られるので、この機能を有効にしても動作は全く変わりません。

この機能は security.syscalls.intercept.mknod を true に設定することで有効に出来ます。

bpf

カーネル内の eBPF プログラムを管理するために bpf システムコールを使用します。これらは様々なカーネルサブシステムにアタッチされます。

一般に、信頼していない eBPF プログラムをロードするのはタイミングベースの攻撃を容易にするので問題です。

LXD の eBPF サポートは現在のところデバイスの cgroup エントリを管理するプログラムに限定しています。有効にするには security.syscalls.intercept.bpf と security.syscalls.intercept.bpf.devices の両方を true に設定する必要があります。

mount

mount システムコールは物理と仮想ファイルシステムの両方のマウントを可能にします。デフォルトでは、非特権コンテナはカーネルにより制限され、いくつかの仮想とネットワークファイルシステムのみに限定されています。

物理ファイルシステムをマウントできるようにするにはシステムコールインターセプションが使えます。LXD はこれを取り扱うために様々な選択肢を提供しています。

security.syscalls.intercept.mount は全体の機能を制御するのに使用され、他のいずれかの選択肢を機能させるためには有効にする必要があります。

security.syscalls.intercept.mount.allowed はコンテナ内に直接マウント可能なファイルシステムのリストを指定できます。これはユーザが信頼できないデータをカーネルに送り込むことを許すため最も危険な選択肢です。これにより簡単にホストシステムをクラッシュさせたり攻撃が出来てしまいます。そのため信頼された環境でのみ使うようにすべきです。

security.syscalls.intercept.mount.shift は上記に加えてコンテナで使用される UID/GID マップでマウントした結果をシフトさせるのに使用できます。これは非特権コンテナ内で全てが nobody/nogroup として表示されるのを回避するために必要です。

これらよりもっと安全な代替は security.syscalls.intercept.mount.fuse でファイルシステムの名前と FUSE ハンドラの組を指定できます。これが設定されると指定されたファイルシステムのどれかをマウントしようとするとそのファイルシステムに対応する FUSE ハンドラ呼び出しにリダイレクトされます。

これは全て呼び出し側として実行されるので、カーネルのアタックサーフェスにまつわる全ての問題を回避し、そのため一般的に安全と考えられます。しかし、あらゆる種類のシステムコールインターセプションはホストシステムに過大な負荷をかける簡単な方法になることを留意しておくべきです。

sched_setscheduler

sched_setscheduler システムコールはプロセスの優先度を管理するのに使えます。

これを許可するとユーザが自分のプロセスの優先度を著しく上げることを許すため、潜在的に多くのシステムリソースを使われることになります。

これはまた SCHED_FIFO のようなスケジューラへのアクセスを許すので、一般的には欠陥と考えられ、システム全体の安定性に著しく影響を与える可能性があります。このため通常の状況下では、真の root ユーザ (あるいはグローバルの CAP_SYS_NICE) のみがこれの使用を許すべきです。

setxattr

setxattr システムコールは、拡張ファイル属性を設定するのに使われます。

現時点で、これにより処理される属性は次のものです:

- trusted.overlay.opaque (overlayfs directory whiteout)

この介入は多数の文字列で行う必要があるため、現在のところ、対象の少数の属性のみインターセプトする簡単な方法がありません。上記の属性のみを許可しているため、カーネルが以前に許可していた他の属性を破損する可能性があります。

この機能は security.syscalls.intercept.setxattr を true に設定することで有効にできます。

sysinfo

sysinfo システムコールはいくつかのディストリビューションでリソース使用量を報告するのに /proc/ エントリーの代わりに使用されます。

システム全体ではなく個々のコンテナのリソース使用情報を提供するために、このシステムコール・インターセプションモードはシステムコールのレスポンスに cgroup ベースのリソース使用情報を設定します。

3.13.5 ユーザー名前空間 (user namespace) 用の ID のマッピング

LXD は安全なコンテナを実行します。これは主にユーザー・ネームスペースの使用によって実現されています。ユーザー・ネームスペースはコンテナを非特権で実行することを可能にし、攻撃対象を大幅に限定します。

ユーザー・ネームスペースはコンテナの UID と GID の組をホストの UID と GID の組にマッピングすることで機能します。

例えば、 100000 から 165535 までのホストの UID と GID を LXD が使用できるようにし、コンテナで 0 から 65535 までの UID/GID にマッピングするように設定できます。

この結果、コンテナ内で 0 の UID で動くプロセスが実際には UID 100000 で動くことになります。

root (0) と nobody (65534) の POSIX の範囲をカバーするため、割当は必ず最低 65535 個の UID と GID であるべきです。

カーネルのサポート

ユーザー・ネームスペースの使用にはカーネル 3.12 以上が必要です。 LXD は古いカーネルでも起動しますが、コンテナを起動するのは拒否します。

使用可能な範囲

ほとんどのホストでは、 LXD は初回起動時に lxd ユーザーの割当のために/etc/subuid と /etc/subgid をチェックし、そこで指定されている範囲の最初の 65536 個の UID と GID をデフォルト・プロファイルで使用するように設定します。

範囲が 65536 より小さい場合 (範囲が全く無い場合を含む)、これが修正されるまで LXD はコンテナの作成と起動に失敗します。

/etc/subuid 、 /etc/subgid 、 newuidmap (パスを検索)、 newgidmap (パスを検索) のいくつか (ただし全部ではない) がシステムに存在する場合、これは shadow の設定が間違っていることを示しているので、これが修正されるまで LXD はコンテナの起動に失敗します。

これらのファイルが 1 つも無い場合、 LXD は 1000000 の基点の UID/GID から開始する 1000000000 の UID/GID の範囲を想定します。

これは最もよくあるケースであり、完全に非特権なコンテナをホストするシステム上で稼働するのではない場合（コンテナランタイム自身はユーザー権限で実行するような場合）に、通常は推奨される設定です。

ホスト間で異なる範囲の使用

ホスト間でコンテナを移動する時、送信側のマッピングが送られるので、受信側のホストで異なる範囲にマッピング可能です。

コンテナ毎に異なる ID マッピング

コンテナを他のコンテナからより一層隔離するために、 LXD はコンテナ毎に異なる ID マッピングを使用することをサポートしています。これはコンテナ毎に `security.idmap.isolated` と `security.idmap.size` という 2 つの設定項目で制御できます。

`security.idmap.isolated` が設定されたコンテナは `security.idmap.isolated` が設定された他のコンテナと衝突しないユニークな ID の範囲を持つように設定されます (もしそのようなコンテナが 1 つも存在しない場合、このキーを設定しようとしても失敗します)。

`security.idmap.size` が設定されたコンテナはこのサイズに ID の範囲が設定されます。このプロパティが設定されていない隔離されたコンテナは ID の範囲がデフォルトのサイズ 65536 に設定されます。これにより POSIX に準拠し、コンテナ内で `nobody` ユーザーが使用できます。

特定のマッピングを選択するには `security.idmap.base` を設定すると自動検出機構をオーバーライドし、コンテナでベースとして使用したいホストの UID/GID を LXD に伝えることができます。

これらのプロパティを反映するにはコンテナの再起動が必要です。

カスタムの ID マッピング

さらに LXD は ID マッピングの一部をカスタマイズすることをサポートします。例えば、UID を変更するファイルシステムを必要とせずに、ホストのファイルシステムの一部をコンテナにバインドマウントすることをユーザーに許可できます。このためのコンテナ毎の設定項目は `raw.idmap` で、設定例は以下のようになります。

```
both 1000 1000
uid 50-60 500-510
gid 100000-110000 10000-20000
```

1 行目は、ホストの UID と GID 1000 の両方をコンテナ内の UID 1000 にマッピングする設定です (これは例えばユーザーのホームディレクトリをコンテナ内にバインドマウントするのに使用できます)。

2 行目と 3 行目は UID または GID のどちらかだけをコンテナ内にマッピングする設定です。行の中の 2 番目のエントリーはソース ID 、 つまりホスト上の ID で、 3 番目のエントリーはコンテナ内部での範囲です。これらの範囲は同じサイズでなければなりません。

このプロパティを反映するにはコンテナの再起動が必要です。

3.14 外部リソース

Configuration options

instance

agent.nic_config, ??
cluster.evacuate, ??
migration.incremental.memory.iterations, ??

server

backups.compression_algorithm, ??
instances.nic.host_name, ??
instances.placement.scriptlet, ??
maas.api.key, ??

something

test1, ??
test2, ??
test3, ??