hive-builder

リリース *3.7.5*

Mitsuru Nakakawaji

2023 年 09 月 15 日

Contents:

第 1 章　概要 1
- 1.1 サイト . . . 2
- 1.2 構築フェーズ . . . 3
- 1.3 ステージング . . . 4

第 2 章　インストール 5
- 2.1 動作環境 . . . 5
- 2.2 Step 1. python3, docker, sshpass コマンドのインストール . . . 5
- 2.3 Step 2. vagrant のインストール . . . 8
- 2.4 Step 3. 仮想環境の構築 . . . 8
- 2.5 Step 4. hive-builder のインストール . . . 9
- 2.6 Step 5. コレクションのインストール . . . 9
- 2.7 Step 6. プロキシ用の環境変数の設定 . . . 9

第 3 章　Vagrant のインストール 11
- 3.1 Centos 7 の場合 . . . 11
- 3.2 Centos 8 の場合 . . . 12
- 3.3 WSL（Windows Subsystem for Linux）の場合 . . . 14
- 3.4 Mac OS の場合 . . . 14

第 4 章　プロキシ環境下での構築 17
- 4.1 mother マシン側プロキシサーバの利用 . . . 17
- 4.2 vagrant プロバイダの場合 . . . 20
- 4.3 kickstart/prepared プロバイダの場合 . . . 21
- 4.4 プロキシ環境の共通事項 . . . 21

第 5 章　サンプルを構築してみる 23
- 5.1 前提条件と準備 . . . 23
- 5.2 サンプルソースコードの取得 . . . 24
- 5.3 仮想環境の activate . . . 24
- 5.4 hive-builder のインストール . . . 24
- 5.5 パラメータを設定 . . . 25
- 5.6 AWS の設定 . . . 25

5.7 GCP の設定 . . . 25
5.8 ステージの設定 . . . 26
5.9 collection と role のインストール . . . 27
5.10 ドメインの委譲設定 . . . 27
5.11 構築 . . . 29
5.12 テスト . . . 29
5.13 サーバへのログインと zabbix の参照 . . . 30
5.14 サーバの停止と環境の削除 . . . 31
5.15 サンプルのサービス . . . 32

第 6 章 hive 構築ガイド 33
6.1 マザーマシンの構築 . . . 33
6.2 プロジェクトディレクトリの作成 . . . 33
6.3 基盤の構築 . . . 37
6.4 サービスの開発 . . . 38

第 7 章 サービスパターン 47
7.1 負荷分散軸 . . . 47
7.2 ボリューム軸 . . . 48
7.3 負荷分散軸とボリューム軸のパターンの組み合わせの可否 . . . 49
7.4 公開軸 . . . 49
7.5 構成ガイド . . . 52

第 8 章 hive コマンド 69
8.1 Named Arguments . . . 69
8.2 Sub-commands . . . 70
8.3 変数 . . . 75
8.4 ログレベル . . . 76
8.5 .hive ディレクトリ . . . 76
8.6 作業ディレクトリ . . . 76
8.7 マザーマシンからサーバへのアクセス . . . 76
8.8 ステージング . . . 76
8.9 ステージングの切り替え . . . 77
8.10 hive コマンドを使わずに playbook を実行 . . . 77
8.11 hive コマンドを使わずに ssh/scp を実行 . . . 77

第 9 章 インベントリ 79
9.1 hive 定義 . . . 79
9.2 サービス定義 . . . 93

第 10 章 AWS の準備 105
10.1 1. ユーザグループの作成 . . . 105

10.2 2. ユーザの作成 . . . 105
10.3 3. EBS 暗号化用の鍵の設定 . . . 106

第 11 章 フェーズ 107
11.1 build-infra . . . 107
11.2 setup-hosts . . . 108
11.3 build-images . . . 115
11.4 build-networks . . . 117
11.5 build-volumes . . . 117
11.6 deploy-services . . . 117
11.7 initialize-services . . . 118

第 12 章 swarm 拡張機能 119
12.1 仮想 IP 付与機能 . . . 119
12.2 ラベル付与機能 . . . 120
12.3 follow-swarm-service デーモン . . . 121

第 13 章 旧バージョンからの移行 123
13.1 1 系（1.x.x）からの移行 . . . 123
13.2 2.0.x からの移行 . . . 123
13.3 2.0.1 以前からの移行 . . . 124
13.4 2.1.2 以前からの移行 . . . 124

第 14 章 hive 内 zabbix の利用 125
14.1 監視内容 . . . 125
14.2 Slack に通知する . . . 126

第 15 章 バックアップとリストア 129
15.1 バックアップファイルの保存先 . . . 129
15.2 リポジトリと監視データのバックアップ . . . 129
15.3 リストア方法 . . . 130
15.4 個別バックアップの採取 . . . 130

第 16 章 タグマッピング 131
16.1 タグマッピング指定 . . . 131
16.2 タグマッピングの対象 . . . 132
16.3 タグマッピングの運用 . . . 132
16.4 ソフトウェアパッケージのバージョン管理について . . . 135

第 17 章 NFS の利用 137
17.1 高可用性の実装 . . . 137
17.2 事前に用意した NFS サーバを利用する場合 . . . 138
17.3 ファイルサーバの自動構築 . . . 138

17.4 DRBD との比較 142

第 18 章 証明書について 143
18.1 A. コンテナをクライアントとして運用したい場合 143
18.2 B. コンテナをサーバーとして運用したい場合 146
18.3 CA 局証明書の共有 147
18.4 証明書生成ビルトインロール 148
18.5 ルート証明書信頼設定ビルトインロール 149
18.6 OS ごとのデフォルトトラストストア確認コマンド 149

第 19 章 障害からの復旧方法 151
19.1 サーバのディスク残量 151
19.2 コンテナ内のディスク消費量の増加 152
19.3 ホスト内サービスの失敗 155
19.4 DRBD ディスクの同期の失敗 156
19.5 サービスが再起動を繰り返す 158
19.6 サーバーが一台破損した場合 159

第 20 章 よくある質問 161
20.1 build-images で Bad local forwarding specification のエラーになります 161
20.2 リポジトリサーバのログ収集で fluentd を使用しないのはなぜですか 161
20.3 ビルドが止まってしまいます 161
20.4 build-images, initialize-services で fail to create socket のエラーになります 162
20.5 initialize-services で Authentication or permission failure のエラーになります 162
20.6 build-infra で Vagrant command failed のエラーになります 162
20.7 build-images で Release file is not valid yet のエラーが出ます 164
20.8 zabbix で Detect SELinux alert の障害（problem）が残ったままになります。 164
20.9 zabbix の SELinux alert でエラーが出ます 165
20.10 deploy-services で renaming services is not supported のエラーが出ます 165
20.11 build-volumes で modprobe: ERROR: could not insert 'drbd': Required key not available のエラーが出ます 166
20.12 mother 環境構築直後の build-infra フェーズで Unexpected failure during module execution. のエラーが出ます 166
20.13 異常がないのに zabbix で At least one of the services is in a failed state のトリガーがあがります . . 167
20.14 異なるホストに配置されたサービス間の通信ができません 167
20.15 dockerhub からイメージをダウンロードするときにダウンロードでエラーになります 170
20.16 gcp プロバイダを使用している場合に hive build-infra で Permission denied のエラー 171
20.17 サービスが特定のサーバに偏ってしまったようですが、どうしたらいいですか 172
20.18 構築後に internal_cidr の値を変更するにはどうしたらいいですか 172
20.19 hive ssh および ssh コマンドでの ssh 接続ができません 174

第 21 章 bash completion の利用 177

第 1 章

概要

hive-builder は複数台のサーバにまたがって docker コンテナを運用するハイパーコンバージドなサイトを構築するためのツールです。Kubernetes を使用せず docker swarm mode によるクラスタ機能と drbd9 によるディスク冗長化機能を使用することで、シンプルな構成でサイトを構築できます。hive は巣箱の意味で、マイクロサービスの群れ (docker swarm) を収容し、これを運用管理します。

- コントローラを必要とせず、 hive を構成するサーバ（3 台以上）が選挙によってリーダを選出する方式により split brain を防ぎます
- docker swarm クラスタを構築することで、高い可用性を確保できます
- drbd でディスクをミラーリングすることで、データボリュームを持つコンテナもマイグレーションできます
- コマンドを 1 回起動するだけで AWS などの IaaS 上にサイトを構築できます
- ansible の role でコンテナの構築内容を記述できます
- サイトの初回起動時に初期データをロードできます
- サイト内にプライベートなリポジトリサーバを持ち、コンテナイメージを保存します
- サイト内に Zabbix サーバを持ち、稼働を監視します
- 1 個のインベントリから 3 段階のステージングに分けて環境を構築できます

> 警告**:** まだ、ドキュメントに未執筆部分がたくさんあることと、今後、非互換となる変更を行う可能性があることから、ビジネス用途の本番環境でご利用になるのは難しいかもしれません。本番環境でご利用になる場合はご一報ください。

> 警告**:** 2.0.1 からホストの OS は CentOS 8 となり、1.2.3 以前のバージョンで構築された CentOS 7 によるサイトとは、互換性がなくなりました。サイトを再構築するか、1 系の最新版（1.2.3）を利用してください。

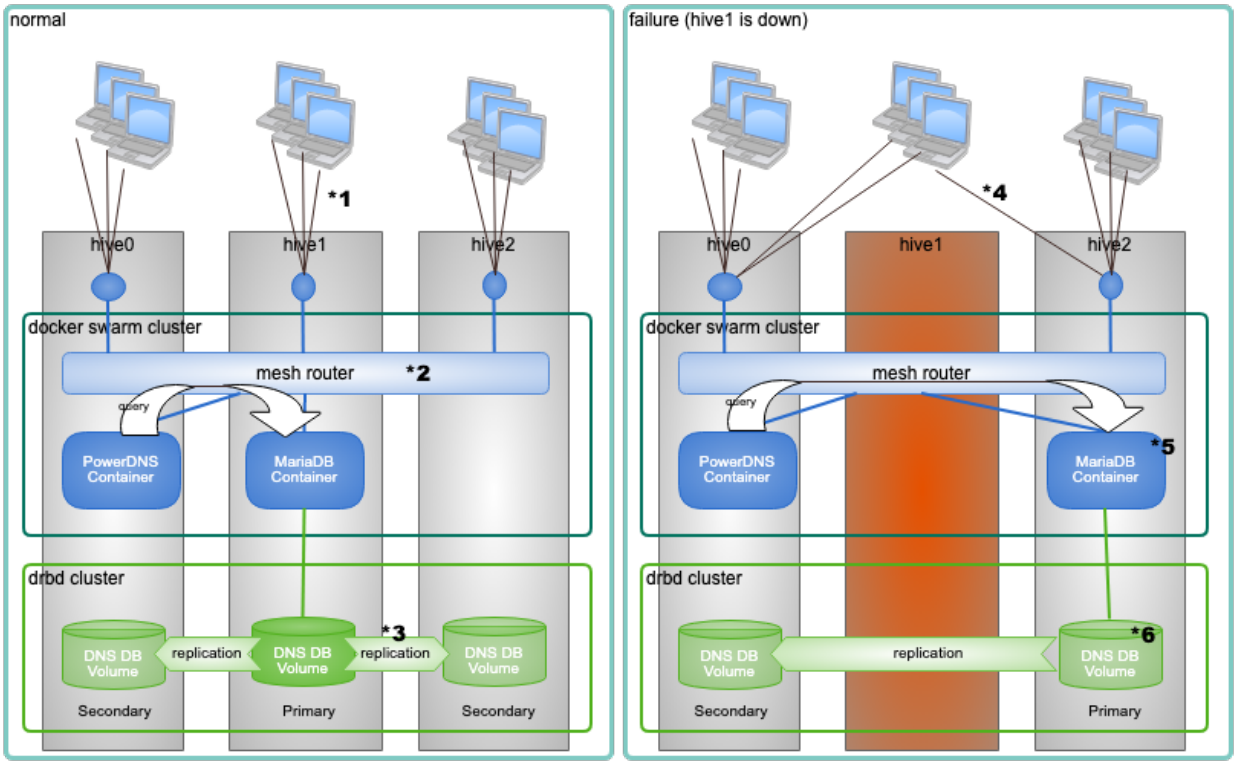
警告: 3.0.1 から ansible 4.0 が対象となり、2.2.14 以前のバージョンで構築された ansible 2.9 によるサイトとは、互換性がなくなりました。サイトを再構築するか、2 系の最新版（2.2.14）を利用してください。

1.1 サイト

サイトは複数のコンテナ収容サーバと 1 台のリポジトリサーバから構成されます。

1.1.1 高可用性

コンテナ収容サーバはリーダ選出選挙で高い可用性を確保するために 3 台以上配備する必要があります。docker swarm mode と drbd9 の機能で高可用性を実現します。コンテナ収容サーバで構成されるクラスタを hive と呼びます。以下にその仕組を説明します。



1. 利用者端末は GSLB により、hive0-2 の 3 個のグローバル IP アドレスに対してラウンドロビンで均等に割り当てられる
2. メッシュルータにより、各サービスがどのサーバで稼働しているかにかかわらず適切にルーティングされる
3. 常時 Primary から Secondary に更新内容が複製される
4. GSLB が死活監視によって hive1 の停止を検知し、利用者端末に対して hive0,hive2 のグローバル IP のみを

返すようになる

5. swarm クラスタが hive1 の機能停止を検知して MariaDB コンテナを hive2 にマイグレーション
6. MariaDB コンテナのマイグレーションに連動して、hive2 のボリュームが Primary に自動的に昇格

ただし、GSLB の機能は hive-builder に含まれていません。hive-builder のサンプルとして Powerdns を使った GSLB を添付していますので、これを構築して使っていただくか、route53 などの GSLB サービスを利用していただくことで、上記の高可用性サイトを構築していただけます。

1.1.2 リポジトリサーバ

リポジトリサーバは以下の 4 つの機能を提供します。

リポジトリサービス コ
ンテナイメージを保持し、コンテナ収容サーバに配信する

稼働監視サービス コ
ンテナ収容サーバの健常性を監視する

バックアップサービス 日
次バッチでコンテナからバックアップを採取する

ログ収集サービス コ
ンテナからログを収集してログファイルを作る

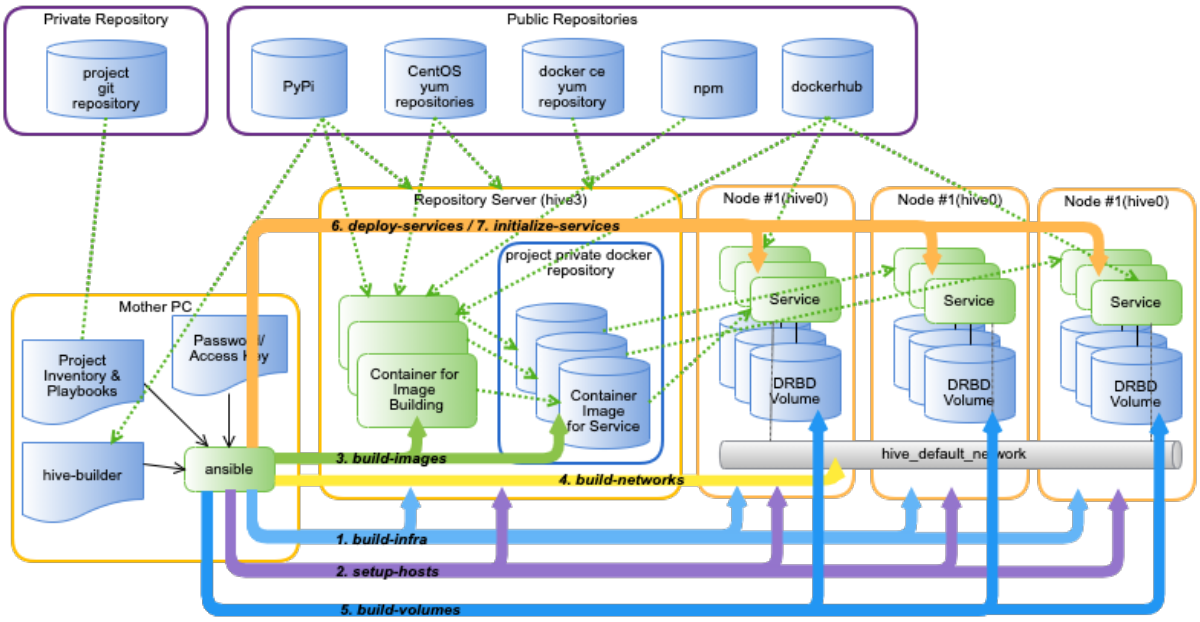
リポジトリサーバ自身が機能停止しても、hive はサービスを提供し続けることができます。また、hive-builder で短時間で再構築できるため、バックアップを取る必要もありません。

1.2 構築フェーズ

サイトの構築は、以下の 7 つのフェーズを順に実行することで行われます。

フェーズ名	対象	内容
build-infra	ホスト, ネットワーク	ホストとネットワークを作成し、環境を構築する
setup-hosts	ホスト	ホストを設定する
build-images	コンテナイメージ	コンテナイメージをビルドする
build-networks	内部ネットワーク	内部ネットワークを構築する
build-volumes	ボリューム	ボリュームを構築する
deploy-services	サービス	サービスを配備する
initialize-services	サービス	サービスを初期化する

以下にフェーズごとの動作を図示します。



1.3 ステージング

ステージングには以下の 3 つがあります。

ステージ名	説明
private	開発者個人のテスト環境です
staging	結合テストを実施する検証用の環境です
production	本番環境です

すべてのステージを定義する必要はなく、必要に応じてインベントリにステージごとのインフラを定義していただけます。

第 2 章

インストール

ここでは、hive-builder を mother マシンにインストールする方法について説明します。

2.1 動作環境

インターネット上の各種リポジトリに http, https でアクセスできる必要があります。プロキシ経由でのみアクセスできる環境の場合、「プロキシ環境下での構築」を参照して事前準備を行ってください。

mother マシンの OS は、CentOS, Windows Subsystem for Linux, Mac OS, Ubuntu などの環境で以下を満たしている必要があります。

- openssl コマンドが利用できること
- pip が利用できること
- python 3.8 以上が利用できること
- git コマンドが利用できること
- docker コマンドが利用できること

インストールの手順を示します。OS ごとにインストール方法が異なる手順もありますので注意してください。

2.2 Step 1. python3, docker, sshpass コマンドのインストール

hive-builder では、python3, docker, sshpass コマンドが必要です。python3 は python 3.8 以上である必要があります。各 OS ごとの手順に従ってインストールしてください。

注釈: Cent OS でサポートしているのは CentOS 7/8 です。CentOS 6 は非サポートです。

2.2.1 1-A. Centos 7 の場合

以下のコマンドを root ユーザで実行して python3.8, docker コマンドをインストールしてください。Centos 7 の場合は sshpass コマンドは不要です。

```
yum install -y centos-release-scl
yum install -y rh-python38 which docker-client
scl enable rh-python38 bash
```

2.2.2 1-B. Centos 8 の場合

以下のコマンドを root ユーザで実行して python3.9, docker コマンドをインストールしてください。Centos 8 の場合は sshpass コマンドは不要です。

```
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install -y python39 docker-ce-cli
```

注釈: prepared プロバイダを使用し、その対象サーバ内に mother 環境を作成する場合は、docker コマンドをインストールする必要はありません。その場合はこのステップをスキップしてください。逆に CentOS 標準の docker-client パッケージがインストールされていると、hive-builder がインストールする docker-ce と競合して構築に失敗しますので、注意してください。

2.2.3 1-C. Windows の場合

Windows 利用する場合は WSL(Windows Subsystem for Linux) 上で実行する必要があります。WSL を公式サイト https://docs.microsoft.com/ja-jp/windows/wsl/install-win10 に従ってインストールしてください。その後、以下のコマンドを WSL の root ユーザで実行して python3, docker, sshpass をインストールしてください。

```
apt update
apt upgrade
apt install -y libffi-dev libbz2-dev liblzma-dev libsqlite3-dev libncurses5-dev libgdbm-
↪dev zlib1g-dev libreadline-dev libssl-dev tk-dev build-essential libncursesw5-dev␣
↪libc6-dev openssl git
wget https://www.python.org/ftp/python/3.9.5/Python-3.9.5.tgz
tar -zxf Python-3.9.5.tgz
cd Python-3.9.5
./configure --enable-optimizations
```

(次のページに続く)

(前のページからの続き)

```
make install
apt install docker sshpass
apt install docker.io
```

ansible でサーバへのログインに使用する ssh 鍵のファイルについて、owner は自分で mode は 0400 となっていて、他人から参照できない状態である必要があります。Windows 10 WSL 環境で、例えば、 /mnt/c/Users/lucy のように Windows から見えるディレクトリに hive のルートディレクトリを作成すると、ssh 鍵の mode が 0777 となってしまい、 ssh ログイン時にエラーになります。その場合、context_dir を ~/hive-context などに設定することで回避できます。以下のコマンドを実行してください。

```
mkdir -p ~/.hive/private
hive set context_dir ~/.hive/private
```

この操作はステージごとに必要であり、context_dir はステージごとに異なる必要があります。

2.2.4 1-D. Mac OS の場合

Python と sshpass コマンドのインストールには Homebrew が必要です。Homebrew が未インストールの場合は公式ページ https://brew.sh/index_ja に従って Homebrew をインストールしてください。

以下のコマンドで Python3.9 と sshpass コマンドをインストールしてください。

```
brew install python@3.9 sshpass
```

後述の仮想環境を作成するときの python3 コマンドには /usr/local/Cellar/python@3.9/3.9.5/bin/python3.9 を使用してください。

docker コマンドのインストールの手順は以下のページに従ってください。https://docs.docker.com/docker-for-mac/install/ インストール後、一度は docker アプリケーションを起動しないと docker コマンドがインストールされません。デスクトップから docker アプリケーションを起動して、docker コマンドが使えるようになったことを確認した後、ステータスバーの docker のアイコンをクリックして docker を終了しても構いません。hive-builder は docker コマンドを必要としますが、端末の docker デーモンにアクセスしません。docker desktop for mac は VM を起動しますので、リソースを消費します。他に docker を必要とすることがなければ、落としておいてください。

2.2.5 1-E. raspbian の場合

以下のコマンドを root ユーザで実行して python3, docker, sshpass をインストールしてください。

```
apt-get update
apt-get upgrade
apt install -y libffi-dev libbz2-dev liblzma-dev libsqlite3-dev libncurses5-dev libgdbm-
↪dev zlib1g-dev libreadline-dev libssl-dev tk-dev build-essential libncursesw5-dev␣
↪libc6-dev openssl git sshpass
wget https://www.python.org/ftp/python/3.9.5/Python-3.9.5.tgz
tar -zxf Python-3.9.5.tgz
cd Python-3.9.5
./configure --enable-optimizations
make install
curl -sSL https://get.docker.com | sh
usermod -aG docker pi
```

2.3 Step 2. vagrant のインストール

vagrant プロバイダを使用する場合は 次の *Vagrant* **のインストール** を参照して vagrant をインストールしてください。

注釈: raspbian では vagrant プロバイダは利用できません。vagrant プロバイダを使用しない場合はこのステップは不要ですので、スキップしてください。

2.4 Step 3. 仮想環境の構築

hive-builder をインストールするための仮想環境を作成したほうが良いでしょう。仮想環境の作成は pyenv, conda, pipenv など、他のツールを用いることもできますし、もともと hive-builder 専用に用意された OS であれば、仮想環境を作成せずに利用することも可能です。以下に Python3 の venv モジュールを用いて作成する場合のコマンド例を示します。

```
cd ~
python3 -m venv hive
echo source ~/hive/bin/activate >> .bashrc
source ~/hive/bin/activate
pip install --upgrade pip wheel
```

注釈: python3 の複数のバージョンがインストールされている場合は、上記の「python3」の部分では明示的に最新バージョンを指定してください。例えば、 CentOS 8 の場合は、python39 コマンドを使用してください。Mac OS の場合は、/usr/local/Cellar/python@3.9/3.9.5/bin/python3.9 を使用してください。

2.5 Step 4. hive-builder のインストール

以下のコマンドでインストールしてください。

```
pip install hive_builder
```

2.6 Step 5. コレクションのインストール

hive-builder をインストールすると、 ansible-core がインストールされます。続いて、 hive-builder が使用する ansible コレクション, ansible ロールをインストールする必要があります。ansible コレクション, ansible ロールはプロジェクト/ステージのコンテキストディレクトリにインストールされるため、プロエジェクトのディレクトリを作成し、hive set stage コマンドでステージを設定後に以下のコマンドを実行して ansible コレクション, ansible ロールをインストールしてください。

```
hive install-collection
```

2.7 Step 6. プロキシ用の環境変数の設定

コンテナ収容サーバ、リポジトリサーバが、 yum, docker, pip, npm などインターネット上のリポジトリアクセスするときにプロキシ経由でアクセスする必要がある場合は、setup-hosts を実行前に各サーバにプロキシ用の設定を行う必要があります。その場合は、**プロキシ環境下での構築** を参照して設定してください。

注釈: サーバから直接リポジトリにアクセスできる場合は、このステップは不要ですので、スキップしてください。

第 3 章

Vagrant のインストール

vagrant プロバイダを使用する場合は vagrant をインストールしてください。また、 vagrant-disksize プラグインもインストールしてください。

注釈: raspbian では vagrant プロバイダは利用できません。vagrant プロバイダを使用しない場合は vagrant をインストールする必要はありません。

注釈: OS ごとにインストール手順が異なりますので、以下の該当する手順のみを実行してください。使用している OS と異なる OS の手順を実行しないように注意してください。

3.1 Centos 7 の場合

Centos 7 では vagrant の libvirt プロバイダを使用します。libvert, qemu-kvm と Vagrant がインストールされている必要があります。以下のコマンドを root ユーザで実行して vagrant をインストールしてください。

```
yum install qemu-kvm qemu-img libvirt virt-install git libvirt-devel gcc
systemctl enable --now libvirtd
yum install https://releases.hashicorp.com/vagrant/2.2.14/vagrant_2.2.14_x86_64.rpm
sh -c 'echo echo^^c2^^a05.2.30r130521 > /usr/bin/VBoxManage'
chmod +x^^c2^^a0/usr/bin/VBoxManage
```

注釈: 最後の 2 行は vagrant up で「Vagrant could not detect VirtualBox! Make sure VirtualBox is properly installed.」のエラーが出る場合に必要となる回避策です。Vagrant のバージョンによっては不要になる可能性があります。

注釈: 2 行目で「ファイルが開けません: https://releases.hashicorp.com/vagrant/2.2.10/vagrant_2.2.10_x86_64.rpm を飛ばします。」というエラーが出る場合は、yum update nss curl コマンドを実行して、 nss と curl をアップデートしてください。

上記の後、構築を実行するユーザで以下のコマンドを実行して仮想環境を構築してください。

```
vagrant plugin install vagrant-libvirt
vagrant plugin install vagrant-disksize
vagrant box add centos/8 --provider=libvirt
sudo usermod --append --groups libvirt `whoami`
```

注釈: vagrant plugin install で「usr/bin/ld: 認識できないオプション '--compress-debug-sections=zlib' です」というエラーが出る場合は、以下のコマンドを実行して、vagrant にパッチをあててください。

```
sudo sed -i -e 's/-Wl,--compress-debug-sections=zlib //' /opt/vagrant/embedded/lib/ruby/
↪2.6.0/x86_64-linux/rbconfig.rb
```

3.2 Centos 8 の場合

Centos 8 では vagrant の libvirt プロバイダを使用します。libvert, qemu-kvm と Vagrant がインストールされている必要があります。また、 vagrant-disksize プラグインがインストールされている必要があります。以下のコマンドを root ユーザで実行して vagrant をインストールしてください。

```
yum install -y --enablerepo=powertools dnsmasq qemu-kvm qemu-img git gcc ruby ruby-devel␣
↪cmake libcmocka-devel \
  libcmocka wget make gcc-c++ rpcgen python3-docutils ninja-build glib2-devel gnutls-
↪devel \
  libxslt-devel libtirpc-devel yajl-devel byacc
python3 -m venv ~/meson
source ~/meson/bin/activate
pip install meson
wget https://github.com/libvirt/libvirt/archive/v6.10.0.tar.gz
wget https://gitlab.com/keycodemap/keycodemapdb/-/archive/master/keycodemapdb-master.tar.
↪gz
tar xzf v6.10.0.tar.gz
tar xzf keycodemapdb-master.tar.gz
```

(次のページに続く)

(前のページからの続き)

```
ln -s  ~/keycodemapdb-master/* libvirt-6.10.0/src/keycodemapdb/
cd libvirt-6.10.0/
groupadd libvirt
chgrp -R libvirt /var/log/libvirt
sed -i -e "s/^SELINUX=enforcing$/SELINUX=disabled/g" /etc/selinux/config
setenforce 0
meson --prefix=/usr --localstatedir=/var --sharedstatedir=/var/lib -D driver_
↪qemu=enabled build
ninja -C build
ninja -C build install
systemctl enable virtnetworkd libvirtd virtqemud virtstoraged
dnf install -y https://releases.hashicorp.com/vagrant/2.2.14/vagrant_2.2.14_x86_64.rpm
cd /tmp; wget https://vault.centos.org/8.3.2011/BaseOS/Source/SPackages/krb5-1.18.2-5.
↪el8.src.rpm
rpm2cpio krb5-1.18.2-5.el8.src.rpm | cpio -imdV
tar xf krb5-1.18.2.tar.gz
cd krb5-1.18.2/src
LDFLAGS='-L/opt/vagrant/embedded/' ./configure
make
cp lib/libk5crypto.so.3.1 /opt/vagrant/embedded/lib64/
ln -s libk5crypto.so.3.1 /opt/vagrant/embedded/lib64/libk5crypto.so.3
ln -s libk5crypto.so.3.1 /opt/vagrant/embedded/lib64/libk5crypto.so
cd /tmp; wget https://vault.centos.org/8.3.2011/BaseOS/Source/SPackages/libssh-0.9.4-2.
↪el8.src.rpm
rpm2cpio libssh-0.9.4-2.el8.src.rpm | cpio -imdV
tar xf libssh-0.9.4.tar.xz
cd libssh-0.9.4
mkdir build; cd build
cmake -DOPENSSL_ROOT_DIR=/opt/vagrant/embedded/ ..
make
cp lib/libssh.so.4.8.5 /opt/vagrant/embedded/lib64/
ln -s libssh.so.4.8.5 /opt/vagrant/embedded/lib64/libssh.so.4
ln -s libssh.so.4 /opt/vagrant/embedded/lib64/libssh.so
sh -c 'echo echo 5.2.30r130521 > /usr/bin/VBoxManage'
chmod +x /usr/bin/VBoxManage
```

注釈: CentOS Stream release 8 で vagrant 2.2.14 を安定して動作させるためには libvirt, libk5crypto, libssh をソースコードからビルドしてインストールする必要があります。バイナリ配布されている libvirt-6.0.0-29 では、

vagrant up 時に Waiting for domain to get an IP address... のメッセージの後、ストールする場合があり、利用できません。また、 vagrant に付属の libcrypto.so は CentOS 8 のものと互換性がなく「symbol EVP_KDF_ctrl version OPENSSL_1_1_1b not defined in file libcrypto.so.1.1 」というエラーが libk5crypto と libssh のロード時に発生し、利用できません。したがって、こちらもソースコードからビルドする必要があります。この手順は将来のバージョンで必要なくなる可能性があります。

注釈: 最後の 2 行は vagrant up で「Vagrant could not detect VirtualBox! Make sure VirtualBox is properly installed.」のエラーが出る場合に必要となる回避策です。Vagrant のバージョンによっては不要になる可能性があります。

上記の後、構築を実行するユーザで以下のコマンドを実行して vagrant のプラグインをロードしてください。

```
sudo usermod --append --groups libvirt `whoami`
vagrant plugin install vagrant-libvirt vagrant-disksize
vagrant box add centos/8 --provider=libvirt
# stream-8 を使う場合
# vagrant box add centos/8 https://cloud.centos.org/centos/8-stream/x86_64/images/CentOS-
↪Stream-Vagrant-8-20200113.0.x86_64.vagrant-libvirt.box
```

3.3 WSL（Windows Subsystem for Linux）の場合

WSL 上では vagrant の virtualbox プロバイダを使用します。VirtualBox を公式サイト https://www.virtualbox.org/ に従って Windows にインストールしてください。その後以下のコマンドで WSL に Vagrant をインストールしてください。

```
wget https://releases.hashicorp.com/vagrant/2.2.16/vagrant_2.2.16_x86_64.deb
dpkg -i vagrant_2.2.16_x86_64.deb
vagrant plugin install vagrant-disksize
```

3.4 Mac OS の場合

Mac OS では vagrant の virtualbox プロバイダを使用します。VirtualBox を公式サイト https://www.virtualbox.org/ に従ってにインストールしてください。その後以下のコマンドで Vagrant をインストールしてください。

```
brew cask install vagrant
vagrant plugin install vagrant-disksize
```

注釈: Mac OS では vagrant の libvirt プロバイダは動作しません。もし、vagrant-libvirt プラグインが入っている場合はアンインストールしてください。

第 4 章

プロキシ環境下での構築

hive-builder では、サーバの構築、コンテナのビルド時に yum, PyPi, dockerhub などのリポジトリにアクセスして、ソフトウェアをダウンロードしてセットアップします。したがって、プロキシ環境下でインターネットへのアクセスがプロキシ経由に限定されている場合、各インストーラがプロキシ経由でインタネットにアクセスするように設定する必要があります。ここでは、 hive-builder をプロキシ環境下で使用する際の使用方法について説明します。

> 警告: プロキシサーバは mother マシン側からアクセス可能である必要があります。ただし、ビルドするステージのプロバイダが vagrant プロバイダ、kickstart プロバイダ、 prepared プロバイダを使用する場合はサーバ側のプロキシを使用して構築していただけます。

4.1 mother マシン側プロキシサーバの利用

mother マシン側のプロキシサーバを利用する場合、プロキシサーバのアドレスを http_proxy 変数に設定して頂く必要があります。たとえば、 mother マシンで squid が起動していて、 3128 番ポートでサービスを提供している場合、以下のように設定してください。

```
hive set http_proxy localhost:3128
```

4.1.1 オフラインキャッシュサーバの利用

上記の設定とともに弊社が提供しているオフラインキャッシュサーバを使用することで、以下の効果を得ることができます。

- yum, dockerhub, pipy, npmjs などからダウンロード・インストールしたソフトウェアのバージョンを将来に渡って固定できる
- yum のリポジトリから互換性のあるバージョンのパッケージが削除されてしまってもキャッシュに残って

いるものをインストールできる

- サーバからインターネットへのアクセスを禁じられている場合でもインストールできる

> 警告: ここで紹介するオフラインキャッシュサーバは dockerhub 以外の外部コンテナリポジトリはサポートしていません。dockerhub 以外の外部コンテナリポジトリを利用しているプロジェクトではここに記載されている方法は利用できませんので、注意してください。

以下に適用手順を示します。

1. hive-builder コードの修正

プロジェクトのコードをオフラインキャッシュサーバに対応させるために以下の点を修正してください。

- プロジェクトのコードで build-images や initialize-services で PyPI 以外のパブリックリポジトリにアクセスするものについては、image.roles や initialize_roles にビルトインロール hive-trust-ca を追加してください。
- hive_ext_repositories で dockerhub の ID, パスワードを指定している場合にはそれをコメントアウトしてください。

> 警告: python_aptk ビルトインロールを使用している場合、 hive_trust_ca を利用することができません。python がインストールされたコンテナイメージを別途用意してください。

2. サーバと CA 局のビルド

以下のコマンドで全ビルドをかけてください。

```
cd プロジェクトのルートディレクトリ
hive set stage private # or staging or production
hive install-collection
hive set http_proxy localhost:3128
hive set registry_mirror http://registry-mirror.offline-cache
hive set pip_index_url http://devpi-server.offline-cache/root/pypi/+simple/
hive set pip_trusted_host devpi-server.offline-cache
hive build-infra
```

3. オフラインキャッシュサーバのソースコードの取得

以下のコマンドでオフラインキャッシュサーバのソースコードを取得してください。git の混乱を避けるためにプロジェクトのルートディレクトリとは別のディレクトリに取得してください。ただし、mother マシンに docker のサーバがインストールされている必要があります。また ansible-core、 docker、 docker-compose モジュールがインストールされた python の仮想環境を作成し、その仮想環境内で ansible-playbook を実行してください。hive-builder 用にインストールされた仮想環境を使用する場合 ` pip install docker-compose docker ` でモジュールを追加してください。credentials.yml.example を参考に、credentials.yml を作成してください。なお NetSoarer 製品を使用しない場合は、nssdc_client_cert と nssdc_client_key の部分は修正する必要はありません。

```
git clone https://github.com/procube-open/offline-cache.git
cd offline-cache
export HIVE_CONTEXT_DIR=<hive のコンテキストディレクトリのパス>
echo export HIVE_CONTEXT_DIR=<hive のコンテキストディレクトリのパス> >> ~/.bashrc
docker-compose up -d
ln -s $HIVE_CONTEXT_DIR/collections ~/.ansible/collections/
ansible-playbook -i squid,registry, -e @credentials.yml setup.yml
```

<hive のコンテキストディレクトリのパス>は、プロジェクトのルートディレクトリの .hive/ステージ名 のディレクトリを指定してください。例えば、 /home/mitsuru/Projects/pdns がプロジェクトのルートディレクトリであり、private ステージをビルドしている場合は、

```
/home/mitsuru/Projects/pdns/.hive/private
```

を指定してください。

4. 全ビルド

以下のコマンドで全ビルドをかけてください。

```
cd プロジェクトのルートディレクトリ
hive all
```

5. オフライン化

以下のコマンドでオフラインキャッシュサーバのオフライン化を実行してください。このときはまだネットワークに繋がっている必要があります。dockerhub のアカウントにログインするためのユーザ ID とパスワードを用意してください。

```
cd オフラインキャッシュサーバのディレクトリ
ansible-playbook -i squid,registry,devpi-server,nginx, offline.yml
```

6. 再ビルド

この mother 環境があれば、オフラインの状態で全サーバを OS インストールから再構築 (hive all) することができます。

7. squid コンテナの hosts ファイルの再設定

マザーマシンを再起動した場合など、squid コンテナの hosts ファイルの設定と、コンテナの実際の IP アドレスのズレが発生する場合があります。以下のコマンドで squid コンテナの hosts ファイルの再設定を行い、IP アドレスのズレを解消することができます。

```
cd オフラインキャッシュサーバのディレクトリ
ansible-playbook -i squid, reset-squid-hosts.yml
```

4.2 vagrant プロバイダの場合

vagrant プロバイダを使用する場合は vagrant-proxyconf プラグインを使用することで、プロキシ環境下で hive-builder を利用することが可能です。以下のコマンドで vagrant-proxyconf プラグインを mother マシンにインストールしてください。

```
vagrant add vagrant-proxyconf
```

また、プロキシサーバの情報を mother 環境の環境変数に設定する必要があります。設定すべき環境変数は、HTTP_PROXY、HTTPS_PROXY、NO_PROXY とそれぞれの小文字の変数です。NO_PROXY には、リポジトリサーバのサーバ名と localhost, 127.0.0.1 を含めてください。たとえば、プロキシサーバの URL が http://192.168.200.1:3128 の場合、 .bashrc などに以下のように記述してください。

```
### PROXY
export HTTP_PROXY=http://192.168.200.1:3128
export http_proxy=${HTTP_PROXY}
export HTTPS_PROXY=${HTTP_PROXY}
export https_proxy=${HTTPS_PROXY}
export NO_PROXY=p-hive0.pdns,localhost,127.0.0.1
export no_proxy=${NO_PROXY}
### PROXY END
```

上記の例ではリポジトリサーバのホスト名として p-hive0.pdns を指定しています。このホスト名は、hive 名が pdns で、private 環境で、サーバが 1 台（number_of_hosts=1）の場合のリポジトリサーバのホスト名です。リポジトリサーバのホスト名は以下のとおり決定できます。

ステージプリフィックス + "hive" + サーバ台数から 1 を引いた数字 + "." + hive 名

ステージプリフィックスは private 環境では "p-"、 staging 環境では "s-"、 production 環境では "" となります。

4.3 kickstart/prepared プロバイダの場合

kickstart プロバイダ、prepared プロバイダを使用する場合、setup-hosts フェーズの実行前に全てのサーバの /etc/environment でプロキシサーバの情報を環境変数に設定する必要があります。設定すべき環境変数は、HTTP_PROXY、HTTPS_PROXY、NO_PROXY とそれぞれの小文字の変数です。NO_PROXY には、リポジトリサーバのサーバ名と localhost, 127.0.0.1 を含めてください。例えば、プロキシサーバの IP アドレスが 192.168.56.100 で 3128 番ポートで待ち受けている場合、root ユーザで以下を実行します。

```
# cat <<'_EOF' > /etc/environment
HTTP_PROXY=http://192.168.56.100:3128
http_proxy=http://192.168.56.100:3128
HTTPS_PROXY=http://192.168.56.100:3128
https_proxy=http://192.168.56.100:3128
NO_PROXY=p-hive0.pdns,localhost,127.0.0.1
no_proxy=p-hive0.pdns,localhost,127.0.0.1
_EOF
```

上記の例ではリポジトリサーバのホスト名として p-hive0.pdns を指定しています。このホスト名は、hive 名が pdns で、private 環境で、サーバが 1 台（number_of_hosts=1）の場合のリポジトリサーバのホスト名です。リポジトリサーバのホスト名は以下のとおり決定できます。

ステージプリフィックス + "hive" + サーバ台数から 1 を引いた数字 + "." + hive 名

ステージプリフィックスは private 環境では "p-"、 staging 環境では "s-"、 production 環境では "" となります。

4.4 プロキシ環境の共通事項

以下にプロキシ環境での共通事項を説明します。

4.4.1 BUMP SSL のルート CA 局を信頼

プロキシサーバが BUMP SSL を使用する場合、ダウンロード・インストールを実行するサーバおよびコンテナで CA 局の証明書を信頼する必要があります。その方法については **証明書について** を参照してください。

4.4.2 サービス内のプロセスへの環境変数の引き継ぎ

サービス内から REST API 呼び出したり yum, npm, pip などのリポジトリへアクセスしたりする場合はサービス内のプロキシ関係の環境変数が適切に設定されている必要があります。各サービス内のプロキシ関係の環境変数は、それぞれ、 hive build-images の時はリポジトリサーバ、hive deploy-services 時は最初のコンテナ収容サーバの値が引き継がれます。各サーバの/etc/environment でサービス内に必要な値も設定してください。特にサービス間の REST API アクセスなどについてはサービス名を no_proxy に設定しておく必要がありますので、注意してください。例えば、examples/pdns のように pdnsadmin サービスから powerdns サービスの REST API を http://powerdns:8081/ のような URL で呼び出す場合、no_proxy には以下のように powerdns を追加する必要があります。ただし、hive set http_proxy を設定している場合は、/etc/environment の設定は自動的に行われ、no_proxy にはすべてのサービス名が登録されます。

```
NO_PROXY=powerdns,p-hive0.pdns,localhost,127.0.0.1
no_proxy=powerdns,p-hive0.pdns,localhost,127.0.0.1
```

警告: alpine linux のコンテナで最小構成の場合、 wget コマンドは no_proxy 環境変数が聞かない場合があります。この場合は apk add wget で GNU 版の wget をインストールすることで回避できます。参考：https://github.com/gliderlabs/docker-alpine/issues/259

第 5 章

サンプルを構築してみる

ここでは、 hive のソースコードの github に登録されているサンプルを使って、AWS 上に GSLB 機能を持つ権威 DNS サーバを構築してみます。

5.1 前提条件と準備

サンプルを構築するためには IaaS の API に対する鍵を取得する必要があります。また、ドメインを保有していれば実際にサブドメインを管理できます。

5.1.1 IaaS

このサンプルを実行するためには AWS EC2, VPC にアクセスできるユーザの API 鍵が必要です。

AWS **の準備** に従って IAM でビルド用のユーザを作成し、その API 鍵を取得してください。

5.1.2 サブドメイン委譲

ここで構築する権威 DNS サーバを保有するドメインに登録して、サブドメインの管理を委譲することで新しいサブドメインを管理できます。これにより、GSLB として動作させるとともに、Lets Encrypt でサーバ証明書を自動的に取得させることができます。このサンプルでは ddclient で自身のアドレスを DNS に登録し、自動的にサブドメインの委譲を受けれるようになっています。そのコードを利用するためには Google Domains など DDNS をサポートする DNS サーバでドメインを管理している必要があります。以下では Google Domains で管理されている前提で手順を説明します。

5.2 サンプルソースコードの取得

サンプルソースコードを github からダウンロードします。

```
svn export https://github.com/procube-open/hive-builder/branches/master/examples/pdns
cd pdns
```

svn コマンドを使用しない場合は、 hive-builder のソースコード全体を clone してください。

```
git clone https://github.com/procube-open/hive-builder
cd hive-builder/examples/pdns
```

zip ファイルをダウンロードした場合、解凍してください。その後、サンプルのディレクトリを部分的に切り取ってください。

```
tar xvzf hive-builder-master.zip
cp -r hive-builder/examples/pdns . || cd pdns
```

5.3 仮想環境の activate

hive 用の仮想環境を作成し、activate します。仮想環境ツールが pyenv で python 3.9.5 がインストールされている場合、以下のコマンドで activate できます。

```
pyenv virtualenv 3.9.5 hive
pyenv local hive
```

virtualenv, pipenv, conda を利用されている場合は、それぞれの方法で仮想環境を activate してください。

5.4 hive-builder のインストール

仮想環境が activate されている状態で以下のコマンドで hive-builder をインストールしてください。詳しくは **インストール** を参照してください。

```
pip install hive_builder
```

5.5 パラメータを設定

hive_email 変数にメールアドレス、domain 変数にドメイン名を設定して、 inventory/group_vars/all.yml に保存してください。以下に例を示します。

```
hive_email: hostmaster@example.com
domain: example.com
```

5.6 AWS の設定

GCP プロバイダを使用する場合、このセクションをスキップして、「GCP の設定」に進んんでください。

inventory/hive.yml に AWS の環境のパラメータを設定します。services.staging.region にリージョンを指定し、services.staging.subnets の available_zone にアカウントが利用できる 3 つの可用性ゾーンを指定してください。サンプルでは東京リージョンが設定されていますが、可用性ゾーンについては、4 つのうちどの 3 個が利用できるかがアカウントごとに異なるので、注意してください。

また、以下のコマンドで hive の環境に AWS EC2 API の鍵を設定してください。

```
hive set aws_access_key_id アクセスキー ID
hive set aws_secret_access_key アクセスキー
```

5.7 GCP の設定

AWS プロバイダを使用する場合、このセクションをスキップしてください。

inventory/hive.yml を編集してください。8 行目から 37 行目をコメントアウトして、38 行目から 45 行目のコメントアウトを外してください.

修正後

```
#production:
#  provider: azure
#  separate_repository: True
#  cidr: 192.168.0.0/24
#  instance_type: Standard_D2s_v3
#  region: japaneast
#  disk_size: 100
#  repository_disk_size: 150
#  mirrored_disk_size: 20
```

(次のページに続く)

(前のページからの続き)

```
#  repository_instance_type: Standard_D2s_v3
production:
  provider: gcp
  separate_repository: True
  cidr: 192.168.0.0/24
  instance_type: n1-standard-2
  region: asia-northeast2
  mirrored_disk_size: 20
  repository_instance_type: n1-standard-2
```

GCP プロバイダを使用する場合は、プロジェクトのルートディレクトリに gcp_credential.json という 名前でサービスアカウントキーを保持するファイルを置く必要があります。 サービスアカウントの作成については、https://cloud.google.com/iam/docs/creating-managing-service-accounts?hl=ja を参照してください。サービスアカウントを作成する際には「Compute 管理者」のロールを割り当ててください。 サービスアカウントキーについては、 https://cloud.google.com/iam/docs/creating-managing-service-account-keys?hl=ja を参照してください。 鍵の形式で JSON を選択して、プロジェクトのルートディレクトリに gcp_credential.json という名前で保存してください。

5.8 ステージの設定

今回は staging ステージを構築します。以下のコマンドで対象ステージ（デフォルトでは private）を 切り替えてください。AWS プロバイダを使用する場合は対象ステージが staging, GCP プロバイダを使用する場合は対象ステージが production になります。

AWS の場合

```
hive set stage staging
```

GCP の場合

```
hive set stage production
```

5.9 collection と role のインストール

仮想環境が activate されている状態で以下のコマンドで collection をインストールしてください。

```
hive install-collection
```

続いて以下のコマンドで　role をインストールしてください。ただし、AWS と GCP でステージが異なります。

AWS の場合

```
ansible-galaxy role install -p .hive/staging/roles powerdns.pdn
```

GCP の場合

```
ansible-galaxy role install -p .hive/production/roles powerdns.pdn
```

5.10 ドメインの委譲設定

この手順は必須ではありません。ドメインを保有していない場合は、この手順をスキップして「構築」セクションに進んでください。

5.10.1 certbot サービスの有効化

保有しているドメインからサブドメインの委譲ができる場合には、DNS の管理画面に対して Lets Encrypt 発行のサーバ証明書を自動的に付与することができます。この機能は certbot サービスで提供されるため、利用するためには、certbot サービスを staging 環境で有効化する必要があります。具体的には、inventory/powerdns.yml の serices.certbot.available_on 属性のステージのリストに 'staging' を追加します。

修正前

```
available_on:
- production
```

修正後

```
available_on:
- production
- staging
```

5.10.2 DNS レコードの登録

親ドメインに NS レコードと DDNS 対応の A レコードを登録してサブドメインの管理を構築したサーバに委譲してください。Google Domains で設定する場合は、以下の手順で設定します。

1. DNS 管理画面へのアクセス

トップメニューの「マイドメイン」で対象のドメインの「管理」をクリックしてください。表示された画面のメニューで「DNS」をクリックして DNS 管理画面を開いてください。

2. NS レコードの追加

「カスタムレコード」の「^」をクリックして詳細を開き「カスタムレコードの管理」をクリックしてカスタムレコードの管理画面を開いてください。カスタムレコードの管理画面で「新しいレコードを追加」をクリックしてください。ホスト名に "pdns" を入力し、タイプ "NS" を選択し、データに "s-hive0.pdns. ドメイン名" を入力します。次に「このレコードにさらに追加」をクリックし、データに "s-hive1.pdns. ドメイン名" を入力します。もう一度「このレコードにさらに追加」をクリックし、データに "s-hive2.pdns. ドメイン名" を入力します。その後、保存をクリックしてデータを登録してください。

3. ダイナミック DNS レコードの追加

DNS 管理画面で「詳細設定を表示」をクリックしてダイナミック DNS を表示し、「ダイナミック DNS の管理」をクリックしてダイナミック DNS の管理画面を開いてください。ダイナミック DNS の管理画面で「新しいレコードを追加」をクリックしてください。ホスト名に s-hive0.pdns を入力し、もう一度「新しいレコードを追加」をクリックしてください。次のホスト名に s-hive1.pdns を入力し、もう一度「新しいレコードを追加」をクリックしてください。次のホスト名に s-hive2.pdns を入力し、保存をクリックしてデータを登録してください。

4. 認証情報の入力

「カスタムレコード」の「^」をクリックして詳細を開き各レコードごとの「認証方法を表示」をクリックし、表示された画面で「表示」をクリックして、認証情報を取得してください。取得した認証情報を inventory/group_vars/all.yml 内に以下のように ddclient_cred 変数を書き加えてください。

```
ddclient_cred:
 s-hive0.pdns:
   name: ユーザー名
   password: パスワード
 s-hive1.pdns:
   name: ユーザー名
   password: パスワード
```

(次のページに続く)

(前のページからの続き)

```
s-hive2.pdns:
  name: ユーザー名
  password: パスワード
s-hive3.pdns:
  name:
  password:
```

レポジトリサーバーを設定している場合は name,password は空欄で書いてください。

5.11 構築

以下のコマンドで構築してください。

```
hive all
```

このコマンドで以下のことが行われます。

- 前のセクション「ドメインの委譲設定」をスキップしている場合には、このコマンドにより、VPC, サブネット、ゲートウェイ、ファイアウォール、EC2 インスタンス、 Elastic IP が 作成されます
- 各サーバにソフトウェアをインストールし、各種設定を行います
- 3 台のサーバを docker swarm と drbd9 のクラスタとして結合 (join) します
- リポジトリサーバにリポジトリサービス（registry)、監視サービス（zabbix)、日次バックアップサービスを起動します
- マイクロサービスを実装するコンテナイメージを構築し、サイト内のリポジトリに登録します
- ネットワークやボリュームを配備し、マイクロサービス群をデプロイします

5.12 テスト

dig コマンドで以下をテストしてください。10.1.1.4 は s-hive0 の Elastic IP アドレスで置き換えてください。

WSL, Linux の場合、

```
watch dig @10.1.1.4 pdnsadmin.pdns.example.com
```

Mac OS の場合

```
while :; do clear; dig @10.1.1.4 pdnsadmin.pdns.example.com; sleep 2; done
```

このコマンドで 2 秒おきに構築した権威 DNS サーバに GSLB として設定されているアドレスが返ります。すなわち、3 個の Elastic IP のうちの 1 個がランダムに選択されて表示され、ときどき値が変わります。また、http://10.1.1.4(s-hive0 の Elastid IP アドレスで置き換えてください) にアクセスすることで DNS の管理画面にアクセスできます。この画面にログインする際の ID は admin でパスワードは .hive/staging/registry_password の値となります。

また、AWS のコンソールから 3 台の EC2 インスタンスが起動していることを確認し、そのうち、1 台を AWS コンソールから落としても上記テストに異常がない（フェールオーバ時に一時的にエラーになりますが、数秒で復帰します）ことを確認してください。このとき、dig コマンドのテストでは GSLB が死活監視しているために、落とした 1 台のアドレスを返さなくなっていることを確認してください。さらに落としたサーバを AWS コンソールから起動し、dig コマンドの結果に復帰することを確認してください。

サブドメインの委譲の設定をしている場合には、正式な URL https://pdnsadmin.pdns.example.com （example.com の部分は設定した保有ドメインで置き換えてください）で管理画面にアクセスできるはずです。この管理画面には以下のアカウントでログインできます。サーバ証明書が Lets Encrypt から発行されていることを確認してください。

ID
: admin

Password
: .hive/staging/registry_password に書き込まれている値

5.13 サーバへのログインと zabbix の参照

hive コマンドでサーバにログインしてマイクロサービスの稼働状況を見てみましょう。また、zabbix の Web コンソールへのアクセスをポートフォワーディングしてブラウザで参照してみましょう。まず、以下のコマンドでサーバにログインしてください。

```
hive ssh -z
```

これでサーバにログインしますので、以下のコマンドでマイクロサービスの稼働状況を見ることができます。

```
docker service ls
```

表示されたサービスの REPLICAS 欄が 1/1 や 3/3 であれば正常です。 0/1 や 0/3 があれば、そのサービスは動作していないことになります。また、以下のコマンドで各サービスのログを見ることができます。

```
docker service logs サービス名
```

docker service logs コマンドの詳細については https://docs.docker.com/engine/reference/commandline/service_logs/ を参照してください。

ログイン時の hive ssh コマンドでは -z オプションを指定しているので、zabbix の Web コンソールへのアクセスが localhost の 10052 ポートにポートフォワーディングされています。ssh でログインしたままの状態でブラウザから http://localhost:10052 にアクセスして、以下の ID でログインしてください。

ID
: Admin

Password
: zabbix

一度、Web で接続した後、ssh をログアウトしようとすると、ポートの解放待ちで長い時間待たされます。その場合は、Ctrl-C を押して中断してください。

5.14 サーバの停止と環境の削除

hive の build-infra コマンドでサーバの停止と環境の削除が実行できます。

5.14.1 サーバの停止

以下のコマンドでサーバを停止できます。

```
hive build-infra -H
```

停止したサーバは以下のコマンドで起動できます。

```
hive build-infra
```

5.14.2 環境の削除

以下のコマンドで環境を削除できます。

```
hive build-infra -D
```

このコマンドにより、VPC, サブネット、ゲートウェイ、ファイアウォール、EC2 インスタンス、 Elastic IP が削除されます。Elastic IP が開放されるため、再構築した際にはグローバル IP アドレスが変わることに注意してください。

5.15 サンプルのサービス

サンプルの inventory/powerdns.yml に定義されているマイクロサービスについて、以下に説明します。

サービス名	説明
powerdns	GSLB として動作する権威 DNS サーバです
pdnsdb	powerdns のデータを保持するデータベースです
pdnsadmin	powerdns の Web コンソールです
proxy	サイト内の Web サービス（今は pdnsadmin のみ）に Web のリクエストを振り分けるためのリバースプロキシです
configure	Web サービスやサーバ証明書を自動的に検知して、proxy を設定します
certbot	サーバ証明書の取得・更新を自動的に実行します

第 6 章

hive 構築ガイド

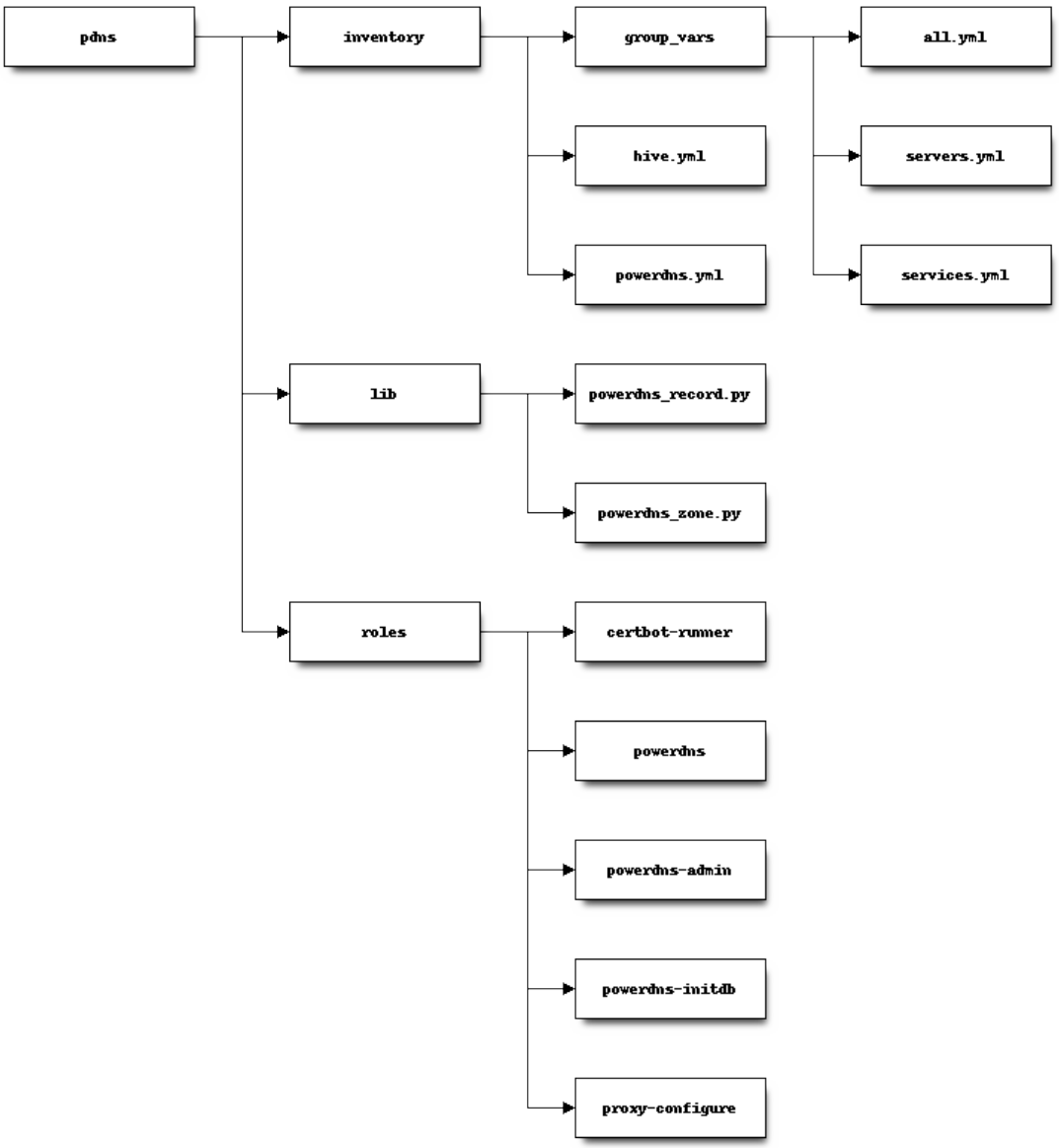
ここでは、hive のサイトを構築する方法を説明します。

6.1 マザーマシンの構築

hive-builder でサイトを構築するために最初にマザーマシンを構築する必要があります。マザーマシンの OS は Linux か Mac OS である必要があります。Windows 10 であれば、Windows Subsystem for Linux を利用していただけます。マザーマシンには CPU1 個、メモリ 1GB、ディスク 3GB 程度のリソースがあれば十分です。そのようなマシンを用意し、 **インストール** を参照して、 hive-builder をインストールしてください。

6.2 プロジェクトディレクトリの作成

マザーマシンにプロジェクトディレクトリを作成してください。サンプルの pdns プロジェクトを例に `hive` のディレクトリ構造を説明します。


pdns
inventory
group_vars
all.yml
hive.yml
servers.yml
powerdns.yml
services.yml
lib
powerdns_record.py
powerdns_zone.py
roles
certbot-runner
powerdns
powerdns-admin
powerdns-initdb
proxy-configure

ディレクトリ/ファイル名	必須	説明
pdns	必須	プロジェクトのルートディレクトリ
inventory	必須	インベントリを保持するディレクトリ
group_vars	任意	リソースグループごとの変数を保持するディレクトリ
all.yml	任意	すべてのリソースに共通の変数を保持するディレクトリ
servers.yml	任意	すべてのサーバに共通の変数を保持するディレクトリ
services.yml	任意	すべてのサービスに共通の変数を保持するディレクトリ
hive.yml	必須	hive 定義（ファイル名は任意）
powerdns.yml	必須	サービス定義（ファイル名は任意）
lib	任意	ansible モジュールを保持するディレクトリ
powerdns_record.py	必須	powerdns のレコードをプロビジョニングするモジュール
powerdns_zone.py	必須	powerdns のゾーンをプロビジョニングするモジュール
roles	任意	コンテナイメージの構築時に呼び出す role を保持するディレクトリ
certbot-runner	任意	サーバ証明書を自動的に取得する certbot-runner サービスのコンテナイメージを構築する role です
powerdns	任意	権威 DNS サーバである powerdns サービスのコンテナイメージを構築する role です
powerdns-admin	任意	権威 DNS サーバの Web コンソールである powerdns-admin サービスのコンテナイメージを構築する role です
powerdns-initdb	任意	権威 DNS サーバのデータベースを初期化する role です
proxy-configure	任意	リバースプロキシの構成を自動的に行う proxy-configure サービスのコンテナイメージを構築する role です

プロジェクトを新規に開発する際は、まず、上記の必須となっているディレクトリとファイルを作成する必要があります。

6.2.1 role のディレクトリ構造

roles 配下には role ごとのディレクトリを作成する必要があります。role の記述方法については ansible の公式ドキュメント を参照してください。たとえば、pdns プロジェクトの powerdns ロールでは以下のディレクトリ構造を持っています。

powerdns/tasks/main.yml の内容は以下の通りです。

```
---
- name: install powerdns
  apk:
    name:
      - pdns
      - pdns-backend-mysql
      - pdns-backend-lua
    state: present
    repository:
    - http://dl-cdn.alpinelinux.org/alpine/edge/community/
    - http://dl-cdn.alpinelinux.org/alpine/edge/main/
    update_cache: yes
- name: install endpoint shell
  copy: src=docker-entrypoint.sh dest=/ mode=0775
- name: "patch default config file - set default"
  lineinfile:
    path: /etc/pdns/pdns.conf
    regexp: "^(# *)?{{item.key}}=.*"
    line: "{{ item.key }}={{ item.value }}"
  with_items:
    - key: daemon
      value: "no"
    - key: guardian
      value: "no"
    - key: launch
      value: gmysql
    - key: chroot
      value: ""
```

(次のページに続く)

(前のページからの続き)

```
- name: "patch default config file - comment out"
  lineinfile:
    path: /etc/pdns/pdns.conf
    regexp: "^(# *)?{{ item }}=.*"
    line: "# {{ item }}="
  with_items:
    - use-logfile
    - wildcards
```

上記の playbook で以下のことを実行しています。

- PowerDNS のソフトウェアのインストール
- エントリポイントのシェルスクリプトを追加
- 設定ファイル /etc/pdns/pdns.conf を編集

6.3 基盤の構築

基盤を構築するためには、inventory/hive.yml を作成し、hive 定義を記述する必要があります。hive 定義の記述方法については **インベントリ** を参照してください。最初は private ステージを作成することが推奨されます。作業をするパソコンのメモリに余裕があれば、 vagrant プロバイダで 4G 以上のメモリをもったサーバを構築するのが良いでしょう。

6.3.1 基盤のテスト

hive コマンドで build-infra フェーズと setup-hosts フェーズがエラーなく成功するようになれば、hive 定義は完成と言えるでしょう。以下のコマンドがエラーなく成功するまで、hive 定義の内容を調整してください。

```
hive build-infra
hive setup-hosts
```

6.4 サービスの開発

hive の中でサービスを起動するためにはサービスをインベントリに定義する必要があります。

ほとんどのサービス（= docker コンテナ）は以下の 5 段階の構築が必要です。

- コンテナイメージのビルド（コンテナへのソフトウェアのインストール）
- ボリュームのマウント
- ネットワークの配備
- サービスのデプロイ（サイト固有パラメータの設定を含む）
- サイトの初期データのロード

以下に順に説明します。

6.4.1 コンテナイメージのビルド

docker では、ソフトウェアのインストールが終わったコンテナのイメージをリポジトリに登録しておき、これをダウンロードして利用するのがベストプラクティスとなります。dockerhub などの外部のリポジトリに登録されているコンテナイメージをそのまま利用する場合は、 hive の中でコンテナイメージを作成する必要はありませんが、プロジェクト固有のカスタマイズが入ったサービスを開発する場合は、 hive の中でコンテナイメージをビルドし、プロジェクト内部のリポジトリに登録する必要があります。

hive ではコンテナイメージの登録とプロジェクト内部のリポジトリへの登録を build-images フェーズで実行できます。このビルドを行うには、サービス定義の image 属性に from 属性と roles 属性を持ったビルド定義オブジェクトを設定する必要があります。image 属性にイメージタグの文字列が設定されている場合は、build-images フェーズの対象となりません。

以下のコマンドで、build-images を実行し、コンテナイメージを登録してください。

```
hive build-images
```

サービス定義の from 属性、roles 属性の記述方法については **インベントリ** を参照してください。

リポジトリの掃除

build-images フェーズを実行すると新しいコンテナイメージが登録されますが、古いコンテナイメージはリポジトリに残ったままになります。ディスク残量が少なくなってきた場合には、 hive ssh コマンドでリポジトリサーバにログインし、以下のコマンドを実行してリポジトリを掃除してください。

```
docker exec -it registry registry garbage-collect -m /etc/docker/registry/config.yml
```

6.4.2 ボリュームのマウント

docker の通常のボリュームに加えて drbd でサーバ間で複製同期するボリュームを利用できます。その仕組みについては、**概要** の高可用性の節を参照してください。サーバ間で複製同期しているため、サービスがどのサーバで起動しても同じ内容のボリュームが見えます。この複製同期するボリュームを使用する場合は、サービス定義の volumes 属性に指定するボリューム定義で drbd 属性を指定してください。drbd 属性の設定方法については **インベントリ** を参照してください。

ボリュームの作成は build-volume フェーズで行われます。以下のコマンドでボリュームを作成できます。

```
hive build-volumes
```

6.4.3 ネットワークの配備

hive は docker swarm の overlay ドライバを使用し、クラスタに参加するすべてのサービスが接続するデフォルトのネットワークを 1 個作成します。このネットワークの名前は hive_default_network です。各サービスはこのネットワークを経由して他のサービスにアクセスすることができます。たとえば、サンプルの powerdns のサービスはデータベースサービス pdnsdb にその名前でアクセスします。docker の内部 DNS が pdnsdb の hive_default_network 上のアドレスを解決し、PowerDNS のサーバはデータベースにアクセスできます。この仕組みは powerdns サービスと pdnsdb サービスがどのホストで動作しているかと関係なく動作します。通常は、このデフォルトのネットワークで十分ですので、特にインベントリにネットワークを定義する必要はありません。

ネットワークの配備は build-networks フェーズで行われます。以下のコマンドで実行できます。

```
hive build-networks
```

6.4.4 サービスのデプロイ

ここでは、 docker swarm サービスをデプロイします。デプロイ時ににサイト固有のパラメータを渡すために、起動時にコマンドラインを指定したり、環境変数にパラメータを指定したりするのが一般的です。また、デプロイしたサービスを外部に対して公開する場合のポート番号を指定する必要があります。

例えば、サンプルの powerdns サービスでは、以下の指定で、サイト固有パラメータを指定しています。

```
environment:
  MYSQL_PASSWORD: "{{db_password}}"
  MYSQL_HOST: pdnsdb
  MYSQL_DNSSEC: "yes"
  PDNSCONF_DEFAULT_SOA_NAME: "{{ (groups['first_hive'] | intersect(groups[hive_stage]) |␣
↪first) + '.' + domain }}"
command:
- "--api=yes"
- "--api-key={{db_password}}"
- "--webserver=yes"
- "--webserver-address=0.0.0.0"
- "--webserver-allow-from=0.0.0.0/0"
ports:
- "53:53/tcp"
- "8081"
- "53:53/udp"
```

環境変数 (environments の配下) で DB サーバへの接続パラメータを渡しています。ここでは、DB にアクセスするためのパスワード（MYSQL_PASSWOWRD）は動的に生成したものを ansible のテンプレート機能で展開しています。また、コマンド引数 (command の配下) で POWERDNS の API を有効化しています。さらに ports でサービスの公開仕様を定義しています。この例では udp/tcp DNS サービスを 53 番ポートで公開し、API サービスのポート 8081 を自動的に割当られるポート番号で公開しています。

ただし、 hive は 10000 以上の番号は外部に公開しないようになっています。IaaS のファイアウォールおよび iptables （未実装）で外部からのアクセスを遮断しています。上記であれば、API サービスは内部からのみアクセスでき、外部には公開されません。

このようにして、定義されたサービスを以下のコマンドで起動することができます。

```
hive deploy-services
```

6.4.5 サイトの初期データのロード

複数のマイクロサービスが連携して機能を実装する場合、build-images や deploy-services では初期データをロードできない場合があります。たとえば、サンプルの powerdns では、ゾーンや A レコードを API から登録しようとすると、 Power DNS とバックエンドのデータベースの両方を稼働させる必要があります。

hive では、 initialize-services フェーズですべてのサービスを稼働させた状態で初期データを登録できます。initialize-services フェーズで初期データを登録するためには、サービス定義の initialize_roles プロパティにデータを初期化するための role を指定し、その role を定義する必要があります。例えば、サンプルでは Power DNS のモジュールを使って初期データを登録しています。サービス定義で initialize_roles に powerdns-init を指定しており、roles/powerdns-init/tasks/main.yml の内容は以下のとおりです。

```
---
- name: get my public IP
  ipify_facts:
  delegate_to: "{{item}}"
  delegate_facts: True
  when: hive_provider not in ['vagrant']
  loop: "{{ groups['hives'] | intersect(groups[hive_stage]) }}"
- name: set published
  set_fact:
    published_ip: "{% if hive_provider not in ['vagrant'] %}{{ hostvars['p-hive0.pdns'].
↪hive_private_ip }}{% else %}{{ hostvars['p-hive0.pdns'].ansible_facts.ipify_public_ip }
↪}{% endif %}"
  delegate_to: "{{item}}"
  delegate_facts: True
  loop: "{{ groups['hives'] | intersect(groups[hive_stage]) }}"
- name: install pip
  apk:
    name: py-pip
- name: install requests module
  pip:
    name: requests
- name: wait for powerdns api available
  wait_for:
    host: localhost
    port: 8081
- name: add zone
  powerdns_zone:
    name: "{{ hive_name }}.{{ domain }}."
    nameservers: "{{ groups['hives'] | intersect(groups[hive_stage]) | map('regex_replace
```

(次のページに続く)

(前のページからの続き)

```
↪', '^(.*)$', '\\1.' + domain +'.' ) | list }}"
    kind: native
    state: present
    pdns_api_key: "{{ hostvars['powerdns'].db_password }}"
- name: add records for hives
  powerdns_record:
    name: "{{ item + '.' + domain + '.' }}"
    zone: "{{ hive_name }}.{{ domain }}"
    type: A
    content: "{{ hostvars[item].published_ip }}"
    ttl: 3600
    pdns_api_key: "{{ hostvars['powerdns'].db_password }}"
  loop: "{{ groups['hives'] | intersect(groups[hive_stage]) }}"
- name: add records for web services
  powerdns_record:
    name: "{{ item + '.' }}"
    zone: "{{ hive_name }}.{{ domain }}"
    type: LUA
    content: A "ifportup(80, {'{{ groups['hives'] | intersect(groups[hive_stage]) | map(
↪'extract', hostvars, ['published_ip']) | join(delimiter)}}'})"
    ttl: 20
    pdns_api_key: "{{ hostvars['powerdns'].db_password }}"
  loop: "{{ groups['services'] | intersect(groups[hive_stage]) | map('extract', hostvars,
 'hive_labels') | select('defined') | map(attribute='published_fqdn') | select('defined
↪') | list }}"
```

最初の 2 つのタスクで各コンテナ収容サーバ (グループ名= hives) のグローバル IP を調べて、host 変数の published_ip に設定しています。この role は powerdns サービスの initialize_roles を定義されているので、対象が powerdns サービスのコンテナとなることに注意してください。最初の 2 つのタスクではコンテナ収容サーバに対象を切り替えるために delegate_to, delegate_facts を使用しています。

続くタスクでゾーンとレコードを登録しています。ここで使用している powerdns_zone モジュールと powerdns_record モジュールは ansible のオフィシャルモジュールではありません。hive では lib ディレクトリの下に置くことでカスタムモジュールを使用できます。サンプルでは、 github の https://github.com/Nosmoht/ansible-module-powerdns で公開されているモジュールをダウンロードして、lib の下に配置しています。また、このモジュールは pdns_port プロパティに API のポート番号を指定する必要がありますが、サンプルでは、 hive-builder が自動的に割り当てたポート番号を powerdns サービスのホスト変数 hive_ports からポート番号 8081 を公開しているものを検索し、公開されるポート番号を取得しています。

6.4.6 サービスのログの閲覧

サービスのログはデフォルト (サービス定義で logging 属性を指定しなければ) ではリポジトリサーバに収集されます。サービスのログを採取するためには

```
hive ssh
```

コマンドでリポジトリサーバにログインし /var/log/services/サービス名 のファイルを参照してください。

スタンドアロン型サービスのログ収集

スタンドアロン型サービスではサービス自身のログは init プロセス（=systemd）の出力となり、何も出力されません。スタンドアロン型サービスのログをリポジトリサーバに収集する場合は、image のビルド時にビルトイン role の hive-syslog を指定してください。

6.4.7 外部リポジトリへのログイン

build-images、および deploy-services フェーズでイメージをダウンロードする際に外部リポジトリを利用することができます。外部リポジトリにアクセスする際にログインが必要な場合、 hive_ext_repositories にログインに必要な情報を設定してください。例えば、dockerhub にアクセスする場合、インベントリ（例えば、inventory/group_vars/all.yml）に以下のように設定してください。

```
credentials: "{{ lookup('file', lookup('env', 'HOME') + '/.hive/credentials.yml') | from_
↪yaml }}"
hive_ext_repositories:
  - login_user: "{{ credentials.dockerhub_login_user}}"
    password: "{{ credentials.dockerhub_login_password}}"
```

上記では、ログインユーザとパスワードを秘密情報をまとめたファイル ~/.hive/credentials.yml から読み込んでます。

6.4.8 サーバにソフトウェアを追加インストール

hive-builder では、 addon という名前の role を追加することで、サーバにソフトウェアを追加インストールすることができます。addon ロールは setup-hosts フェーズで呼ばれ、デフォルトでは become: True, become-user: root で実行されます。アドミニストレータユーザで実行したい場合は、 become: False を指定してください。例えば、サーバに zsh をインストールし、アドミニストレータのシェルを変更するのであれば、 roles/addon/tasks/main.yml を以下のように作成します。

```
---
- name: install zsh
  yum:
    name: zsh
    state: present
- name: set zsh as login shell for administrator
  user:
    name: "{{hive_safe_admin}}"
    shell: /bin/zsh
```

構築済みのサイトに addon ロールを追加した際には、以下のコマンドで反映します。

```
hive setup-hosts -T addon
```

separate_repository: True の状態で、リポジトリサーバを除いてコンテナ収容サーバにのみインストールしたいときは、以下のような when 条件を付与してください。

```
when: inventory_hostname in groups['hives']
```

6.4.9 グループを使った切り分け

hive のインベントリには以下のグループが定義されています。

グループ名	説明
drbd_volumes	DRBD のボリューム
first_hive	1 台目のコンテナ収容サーバ
hives	コンテナ収容サーバ
hosts	全サーバ（mother + servers）
images	ビルドイメージ
mother	mother マシン
networks	仮想ネットワーク
private	private ステージでビルド対象となるもの
production	production ステージでビルド対象となるもの
repository	リポジトリサーバ
servers	サーバ（repsoitory ∪ hives）
services	サービス
staging	staging ステージでビルド対象となるもの

注釈：separate_repository: False の場合は最後の 1 台はコンテナ収容サーバ兼リポジトリサーバとなるため、hives, repository の両方に属します。

これらのグループを使って、when でビルド対象がグループに含まれる場合に限定したり、 inventory/group_vars/グループ名.yml に変数定義することで変数のスコープを限定したりすることができます。

第 7 章

サービスパターン

この章では hive-builder のサービス定義でのボリュームとネットワークの構成方法について、パターンとして解説します。パターンの選択の軸として、負荷分散軸、ボリューム軸、公開軸にわけて、以下に順次説明します。

> 警告: ここでは、サービスにボリューム、公開ポートが複数ある場合、その構成方法は統一することを前提としています。組み合わせによって構成方法を混合させることもできる場合もありますが、複雑になるため推奨されません。

7.1 負荷分散軸

サービスを実装するコンテナをクラスタ内で複数起動することで負荷分散することができます。ここでは、コンテナの稼働数に着目して、パターンに分類します。

7.1.1 シングルトン

サービスとコンテナが 1 対 1 で、負荷分散を行わないサービス。サービスの mode に replicated（デフォルト）を指定し、replicas に 1（デフォルト）を指定して利用します。

7.1.2 スケーラブル

サービスに対して複製のコンテナを配備し、負荷分散を行うサービス。サービスの mode に replicated（デフォルト）を指定し、replicas に複製数を指定して利用します。

7.1.3 グローバル

コンテナ収容サーバごとにコンテナを配備し、外部のロードバランサで負荷分散を行うサービス。サービスの mode に global を指定します。replicas は指定できません。

7.2 ボリューム軸

サービスに対してボリュームをどのように結合するかに着目して、パターン分けします。

7.2.1 ステートレス

サービスが一切ボリュームを持たないパターンです。リバースプロキシやステートレスななロジック層のサービスで利用できます。

7.2.2 サービス固有

サービスに対して固有のボリューム１個割り当てます。データベースのサービスやデータベースを内包するスタンドアロン型のサービスのように、排他的にボリュームを利用する必要があるサービスでは、このパターンを利用する必要があります。サービスのコンテナがクラスタ間を移動した場合でも同じボリュームが見えるようにできます。ボリュームはコンテナイメージの内容で初期化されます。つまり、サービスの最初のコンテナが起動したとき１回だけコンテナイメージの内容がボリュームにコピーされます。ボリュームは DRBD か NFS で構成されていなければなりません。ただし、コンテナ収容サーバが１台構成である場合は、バインドボリュームや通常ボリュームを使用して（次節参照)、構成を単純化しビルドにかかる時間を短縮することができます。

7.2.3 ホスト固有

ホストごとに固有のボリュームを割り当てます。前述の負荷分散軸でグローバルパターンを選択する場合で、ホストごとに内容に差を持たせたい場合に利用します。バインドボリュームと通常ボリュームが利用できます。バインドボリュームとは、コンテナが配備されたホストのディレクトリやファイルをコンテナにマウントするもので、ボリューム定義で type: bind を指定することで利用できます。バインドボリュームで内容の初期化を addon ロールで行うことができます。通常ボリュームとは、ホストごとに docker がオンデマンドにボリュームを作成するもので、ボリューム定義で type: volume（デフォルト） を指定し、NFS/DRBD を構成しないことで利用できます。通常ボリュームでは、ボリュームはコンテナイメージの内容で初期化されます。つまり、最初のサービスが起動したとき１回だけ内容がボリュームにコピーされます。

7.2.4 共有

サービスを実装する複数のコンテナ間でボリュームの内容を共有します。ボリュームは NFS で構成されていなければなりません。ボリュームへの書き込みの排他制御をサービス側で実装する必要があることに注意してください。共通の設定情報を共有する場合など、ボリュームへの書き込みがない場合は、このパターンではなくビルド時にイメージに埋め込むことが推奨されます。

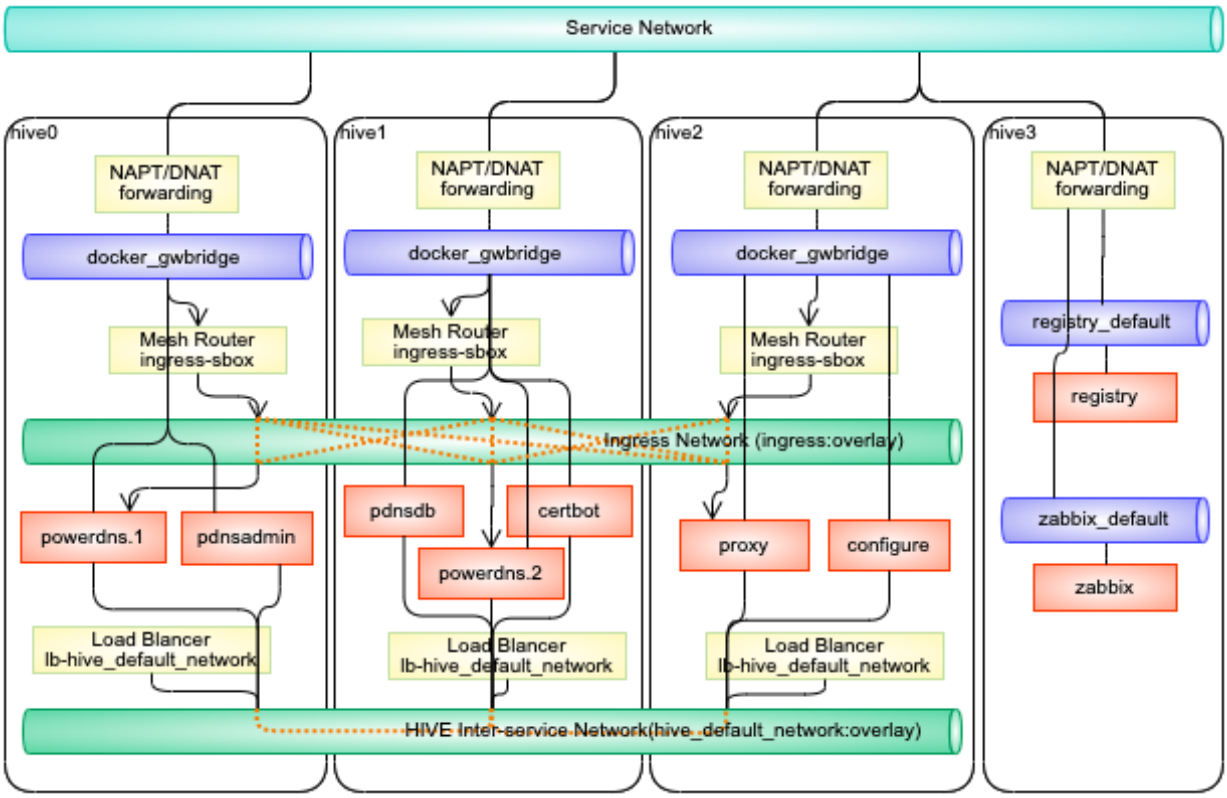
7.3 負荷分散軸とボリューム軸のパターンの組み合わせの可否

以下に、負荷分散軸とボリューム軸のパターンについて組み合わせの可否を示します。

	シングルトン	スケーラブル	グローバル
ステートレス	○	○	○
サービス固有	○	âŸ	âŸ
ホスト固有	âŸ	âŸ	○
共有	âŸ	○	○

7.4 公開軸

サービス定義の ports 属性で公開することが宣言され、外部からのアクセスを受け付けるポートを公開ポートと呼びます。公開ポートのパターンを説明するために、まず、docker swarm の内部ネットワークの仕組みについて説明します。



公開ポート宛のパケットは、サーバで着信し、DNAT により宛先 IP アドレスが書き換えられ、 docker_gwbridge 経由でコンテナかメッシュルータに転送されます。

ここで、公開ポートの mode 属性に ingress を指定（デフォルト）している場合は、パケットはメッシュルータに送られます。メッシュルータに着信したパケットは送信元 IP と宛先 IP の両方を書き換えられて、コンテナに転送されます。公開ポートを持つサービスが複数のコンテナからなっている場合は負荷分散します。

一方で、公開ポートの mode 属性に host を指定している場合は、宛先 IP が書き換えられ、直接コンテナに転送されます。

続いて、公開ポートにの構成方法に着目してパターン別に解説します。

7.4.1 内部サービス

サービス間通信のみをおこない、公開ポートを持たないサービスです。

> 警告: endpoint_mode に vip を指定すると、 ingress ネットワーク上にメッシュルータが配備されるだけではなく、hive_default_network ネットワークにもロードバランサが配備されます。他のコンテナからサービス名で内部サービスを参照する場合、コンテナ内 DNS は実 IP に加えてロードバランサにつけられた仮想 IP も返します。不要なロードバランサと仮想 IP を避けるために、内部サービスの endpoint_mode には、 dnsrr を指

定することが推奨されます。

7.4.2 メッシュルーティング

外部からのアクセスをすべてのコンテナ収容サーバで受け付け、メッシュルータで転送するサービスです。前述の負荷分散軸でグローバル以外のパターンを選択した場合に利用できます。負荷分散を構成できますが、コンテナで受信するパケットの送信元 IP はメッシュルータの IP で書き換えられるため、本来の送信元 IP が不明となります。port の公開モードに ingress (デフォルト) を指定して利用します。サービスの endpoint_mode は vip (デフォルト) でなければなりません。

7.4.3 一対一ルーティング

外部からのアクセスをすべてのコンテナ収容サーバで受け付け、サーバ内のコンテナに転送するサービスです。前述の負荷分散軸でグローバルパターンを選択した場合にのみ利用できます。port の公開モードに host を指定して利用します。サービスの endpoint_mode には dnsrr を指定しなければなりません。

7.4.4 仮想 IP

コンテナが配置されたホストに swarm 拡張機能で仮想 IP を付与し、その仮想 IP への通信をコンテナに転送するパターンです。負荷分散軸でシングルトンパターンを選択した場合にのみ利用できます。

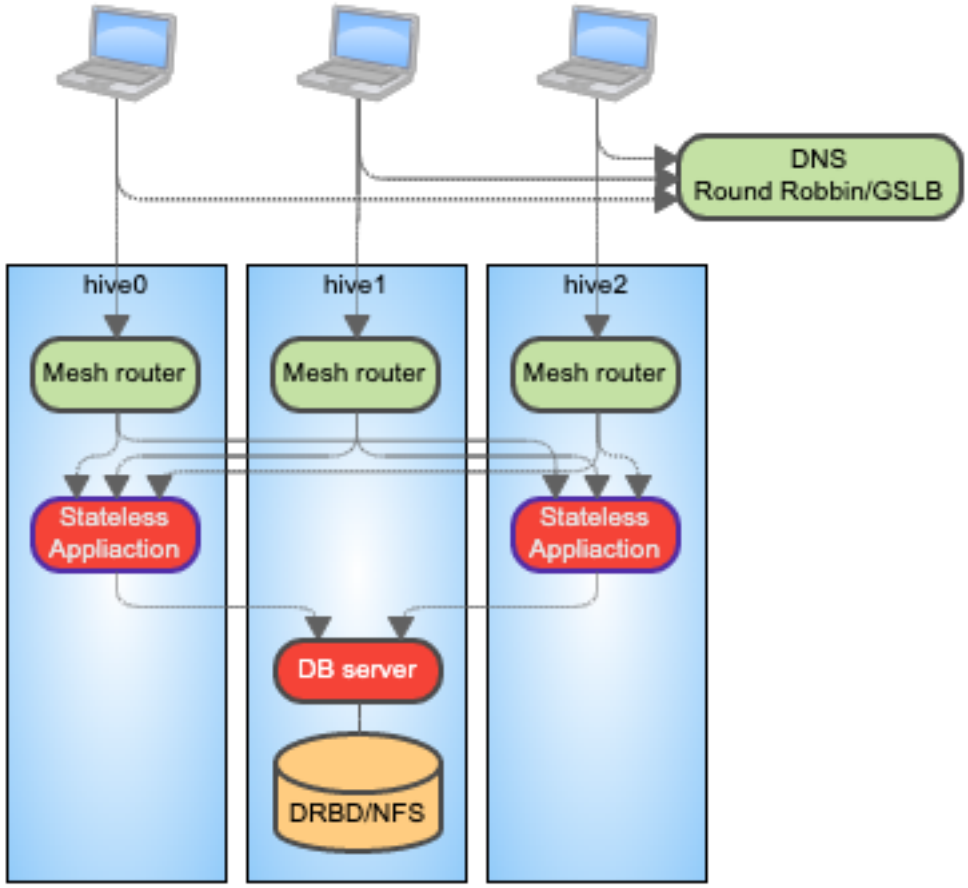
警告: 内部サービスの場合と同じ理由で、仮想 IP でポートを公開するサービスの endpoint_mode には、 dnsrr を指定することが推奨されます。

7.5 構成ガイド

サービスの種類ごとに推奨される構成を示します。

7.5.1 ステートレスメッシュルーティング

インターネットからのアクセスを GSLB で冗長化し、リバースプロキシサービスで受信する場合、コンテナをグローバルパターンで配備することが推奨されます。DNS や RADIUS などクライアント側に複数 IP フォールバック（複数の IP に対して 1 個めにアクセスしてみてだめなら 2 個めにアクセスする）がある場合を含みます。DNS サーバのようなステートレスなサービスをシンプルに冗長化することができます。以下にステートレスなサービスと DB サービスで構成する場合の図を示します。



それぞれの軸のパターンを以下のように選択します。

Statteless Application

パターン軸	パターン
負荷分散軸	スケーラブル
ボリューム軸	ステートレス
公開軸	メッシュルーティング

DB Server

パターン軸	パターン
負荷分散軸	シングルトン
ボリューム軸	サービス固有
公開軸	内部サービス

- replica 属性でスケールを調整できます
- 受診時の送信元 IP をアプリケーションで識別することはできません

以下に設定例を示します。

```
---
plugin: hive_services
services:
  powerdns:
    image: procube/powerdns:latest
    environment:
      MYSQL_PASSWORD: "{{db_password}}"
      MYSQL_HOST: pdnsdb
      MYSQL_DNSSEC: "yes"
      MYSQL_CHECK_INTERVAL: "10"
      MYSQL_CHECK_RETRY: "50"
    command:
    - "--api=yes"
    - "--api-key={{db_password}}"
    - "--webserver=yes"
    - "--webserver-address=0.0.0.0"
    - "--webserver-allow-from=0.0.0.0/0"
    ports:
    - target_port: 53
      published_port: 53
      protocol: udp
      mode: ingress
```

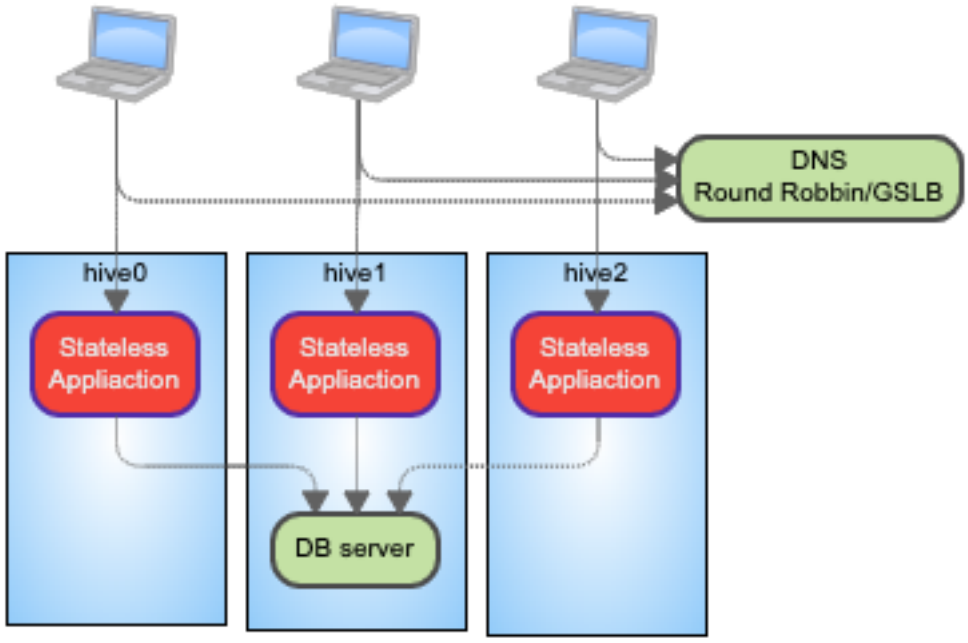
(次のページに続く)

(前のページからの続き)

```
    - target_port: 53
      published_port: 53
      protocol: tcp
      mode: ingress
    initialize_roles:
    - python-aptk
    - powerdns-init
    mode: replicated
    replicas: 2
    endpoint_mode: vip
  pdnsdb:
    image:
    from: "mariadb:10.4"
    roles:
    - python-aptk
    - powerdns-initdb
    endpoint_mode: dnsrr
    environment:
      MYSQL_ROOT_PASSWORD: "{{db_password}}"
      MYSQL_USER: powerdns
      MYSQL_PASSWORD: "{{db_password}}"
      MYSQL_DATABASE: powerdns
    volumes:
    - source: pdnsdb_data
      target: /var/lib/mysql
      type: volume
      drbd:
        size: 500M
        fstype: xfs
```

7.5.2 ステートレスグローバル

DNS のアクセス制御や VIEW 制御のような機能を実装する場合など、前項の構成で送信元 IP を識別できないことが問題となる場合は、グローバルパターンを使用する必要があります。以下に図を示します。



DB Server については、ステートレスメッシュルーティングの項で記載した内容から変更がありませんので、説明を省略します。Statteless Application の軸のパターンを以下のように選択します。

パターン軸	パターン
負荷分散軸	グローバル
ボリューム軸	ステートレス
公開軸	一対一ルーティング

- クライアントからのアクセスは GSLB で冗長負荷分散され、コンテナ収容サーバごとに 1 個ずつ配備されたコンテナで受信します
- 受診時の送信元 IP をアプリケーションで識別して、アクセス制御や VIEW 制御に使用できます
- 複製数はサーバ数に固定されます

以下に設定例を示します。

```
---
plugin: hive_services
services:
  powerdns:
    image: procube/powerdns:latest
    environment:
      MYSQL_PASSWORD: "{{db_password}}"
      MYSQL_HOST: pdnsdb
```

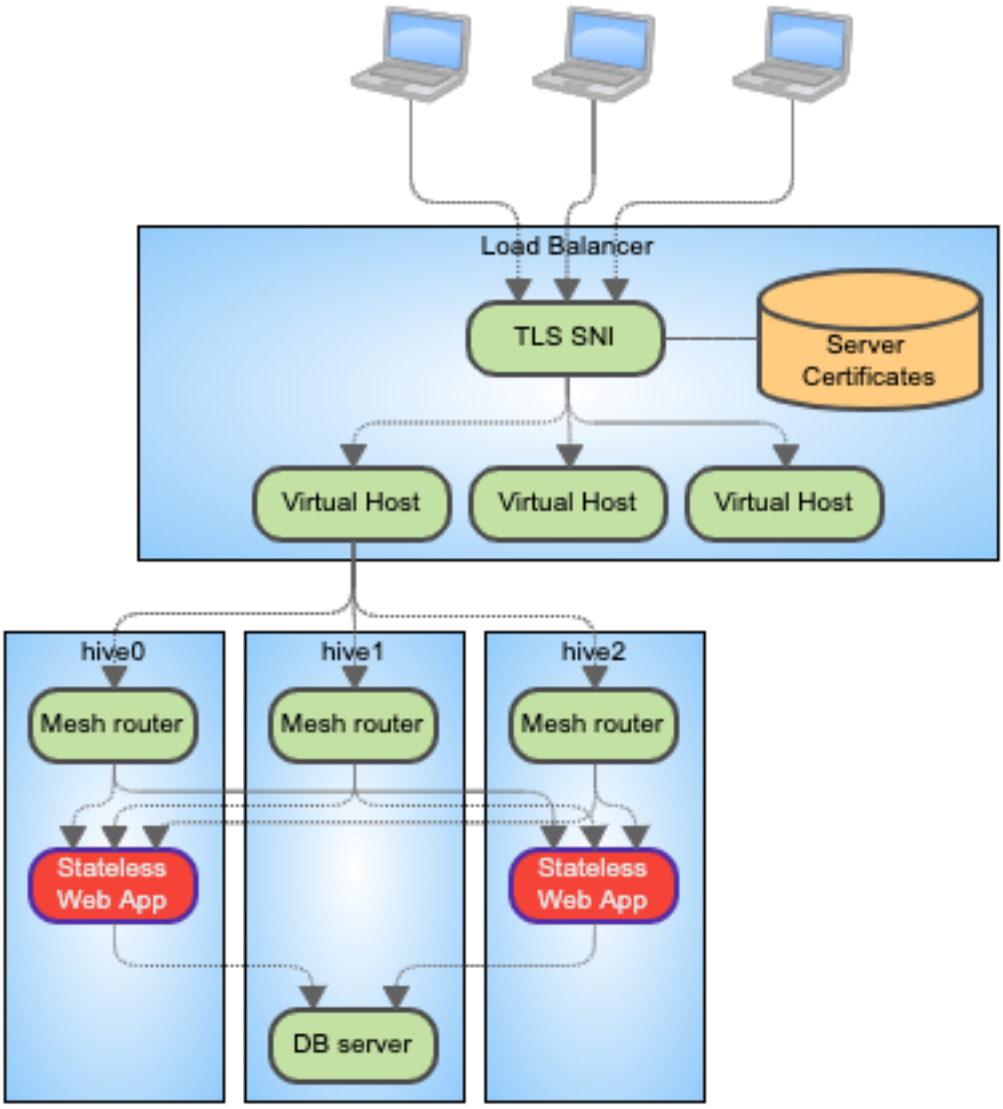
(次のページに続く)

(前のページからの続き)

```
    MYSQL_DNSSEC: "yes"
    MYSQL_CHECK_INTERVAL: "10"
    MYSQL_CHECK_RETRY: "50"
  command:
  - "--api=yes"
  - "--api-key={{db_password}}"
  - "--webserver=yes"
  - "--webserver-address=0.0.0.0"
  - "--webserver-allow-from=0.0.0.0/0"
  mode: global
  endpoint_mode: dnsrr
  ports:
  - target_port: 53
    published_port: 53
    protocol: udp
    mode: host
  - target_port: 53
    published_port: 53
    protocol: tcp
    mode: host
  initialize_roles:
  - python-aptk
  - powerdns-init
```

7.5.3 ロードバランサ背後ステートレスメッシュルーティング

インターネットからのアクセスを外部のロードバランサで受け付けて、コンテナ収容サーバに負荷分散して転送する場合、ステーテレスな Web サービスをメッシュルーティングパターンで公開することが推奨されます。



DB Server については、ステートレスメッシュルーティングの項で記載した内容から変更がありませんので、説明を省略します。Statteless Application の軸のパターンを以下のように選択します。

パターン軸	パターン
負荷分散軸	スケーラブル
ボリューム軸	ステートレス
公開軸	メッシュルーティング

- replica 属性でスケールを調整できます。
- データベースをコンテナ内に持たないようにしてステートレスパターンを採用します
- 受診時の送信元 IP はロードバランサの IP アドレスになるので、メッシュルーティングを採用することのデ

メリットはありません。

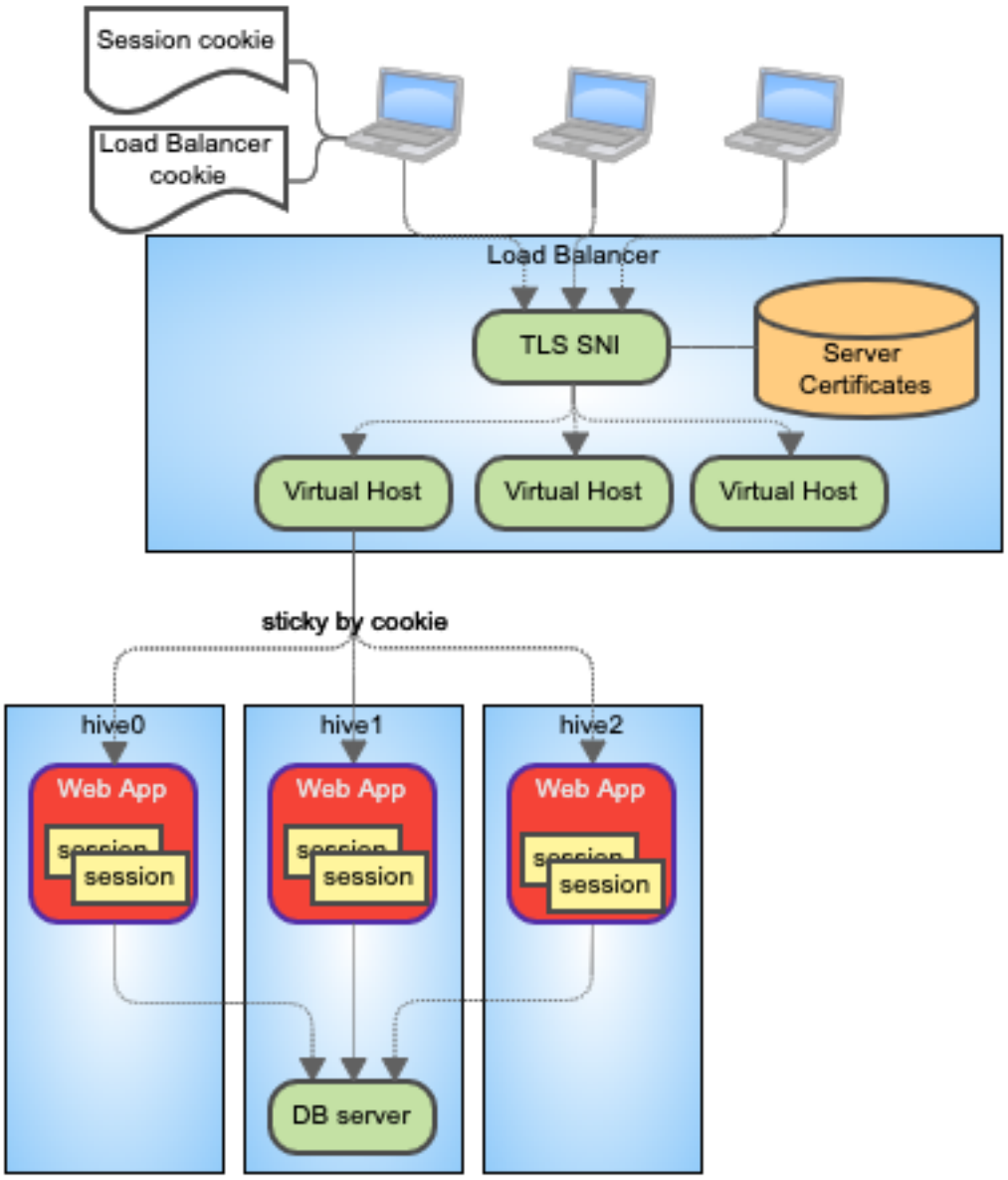
- 送信元 IP は送信元 IP は外部のロードバランサが追加する X-Forwarded-For ヘッダで識別できますので、アクセスログにその値を記録できます
- サーバ証明書はロードバランサで保持してさせロードバランサとの間は非暗号化通信とすることで、コンテナ内でのサーバ証明書の交換は不要となります
- メッシュルータから負荷分散で転送する場合はスティッキーセッションが使用できないので、セッション情報などをコンテナごとのメモリ上に保持するようなアプリケーションには適用できません（セッション情報を DB で共有することでステートレス化する必要があります）

以下に設定例を示します。

```
---
plugin: hive_services
services:
  speedtest:
    image: linuxserver/librespeed
    environment:
      PUID: "1000"
      PGID: "1000"
      TZ: Asia/Tokyo
      DB_TYPE: mysql
      DB_NAME: speedtest
      DB_HOSTNAME: stdb
      DB_USERNAME: speedtest
      DB_PASSWORD: speedtest
    mode: replicated
    replicas: 2
    endpoint_mode: vip
    ports:
    - target_port: 80
      published_port: 80
      protocol: tcp
      mode: ingress
```

7.5.4 ロードバランサ背後スティッキーグローバル

セッション情報などをコンテナごとのメモリ上に保持するようなアプリケーションであるため、前項のパターンのスティッキーセッションを利用できないことが問題となる場合は、グローバルパターンを使用する必要があります。以下に図を示します。



DB Server については、ステートレスメッシュルーティングの項で記載した内容から変更がありませんので、説明を省略します。Web App の軸のパターンを以下のように選択します。

パターン軸	パターン
負荷分散軸	グローバル
ボリューム軸	ステートレス
公開軸	一対一ルーティング

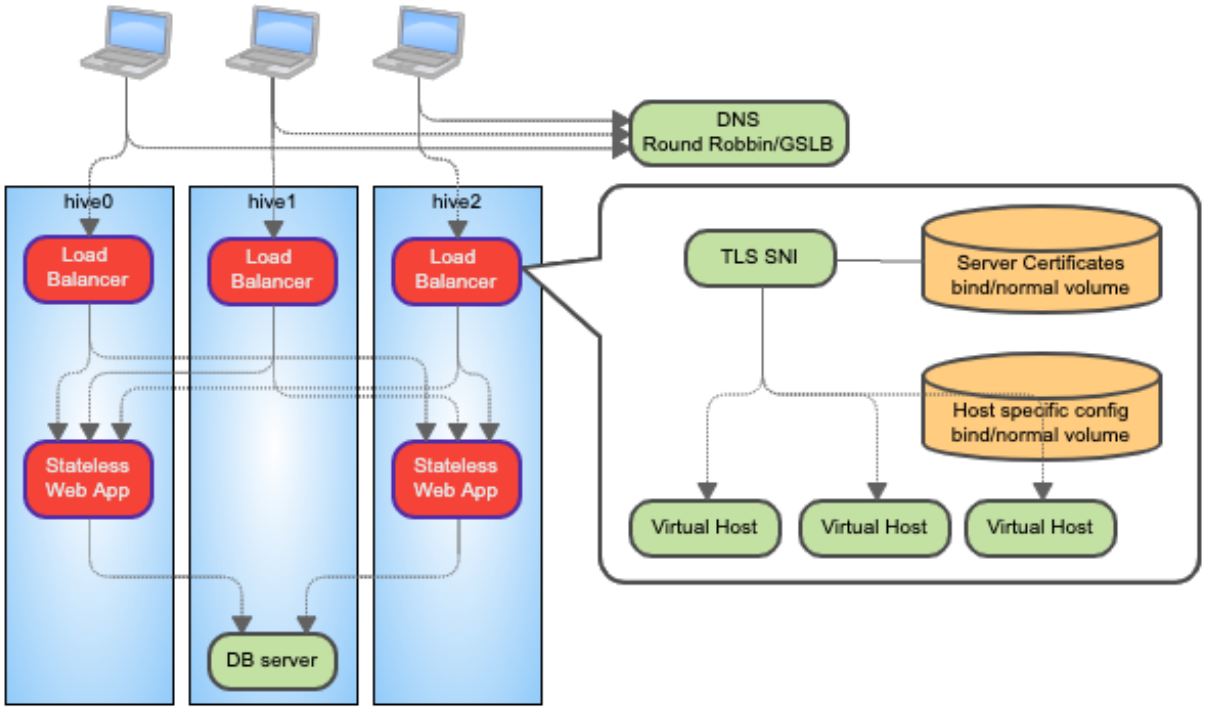
- データベースをコンテナ内に持たないようにしてステートレスパターンを採用します
- 受診時の送信元 IP はクライアントの IP アドレスとなり、アクセスログに記録することが可能です
- サーバ証明書はロードバランサに保持させロードバランサとの間は非暗号化通信とすることで、コンテナ内でのサーバ証明書の交換は不要となります
- ロードバランサからの負荷分散でスティッキーセッションを使用することで、セッション情報などをコンテナごとのメモリ上に保持するようなアプリケーションにも適用できます

以下に設定例を示します。

```
---
plugin: hive_services
services:
  speedtest:
    image: linuxserver/librespeed
    environment:
      PUID: "1000"
      PGID: "1000"
      TZ: Asia/Tokyo
      DB_TYPE: mysql
      DB_NAME: speedtest
      DB_HOSTNAME: stdb
      DB_USERNAME: speedtest
      DB_PASSWORD: speedtest
    mode: global
    endpoint_mode: dnsrr
    ports:
    - target_port: 80
      published_port: 80
      protocol: tcp
      mode: host
```

7.5.5 内蔵ロードバランサ

インターネットからのアクセスを GSLB で冗長不可分させ、ロードバランササービスで受信する場合、コンテナをグローバルパターンで配備することが推奨されます。Apache httpd や nginx のような Web のリバースプロキシをロードバランサとして Web サービスの前段に立てる場合に使用します。また、 dnsdist など非 Web の L7 プロキシの場合にも利用できるパターンです。以下に図を示します。



DB Server については、ステートレスメッシュルーティングの項で記載した内容から変更がありませんので、説明を省略します。それぞれの軸のパターンを以下のように選択します。

Load Balancer

パターン軸	パターン
負荷分散軸	グローバル
ボリューム軸	ホスト固有
公開軸	一対一ルーティング

Statteless Application

パターン軸	パターン
負荷分散軸	スケーラブル
ボリューム軸	ステートレス
公開軸	メッシュルーティング

- クライアントからのアクセスは GSLB で冗長負荷分散され、コンテナ収容サーバごとに 1 個ずつ配備されたコンテナで受信します
- 構成情報やサーバ証明書はホスト固有パターンのボリュームに保存することで、サービスを再起動せずに構成変更やサーバ証明書を交換できるようにします
- 受診時の送信元 IP はクライアントの IP アドレスとなり、アクセスログに記録することが可能です
- Statteless Application については、 replica 属性でスケールを調整できます。
- メッシュルータから負荷分散で転送する場合はスティッキーセッションが使用できないので、セッション情報などをコンテナごとのメモリ上に保持するようなアプリケーションには適用できません（セッション情報を DB で共有することでステートレス化する必要があります）

以下に設定例を示します。

```
---
plugin: hive_services
services:
  proxy:
    image: "procube/nginx:latest"
    mode: global
    endpoint_mode: dnsrr
    ports:
    - target_port: 80
      published_port: 80
      protocol: tcp
      mode: host
    - target_port: 443
      published_port: 443
      protocol: tcp
      mode: host
    volumes:
    - source: /var/proxy/nginx/conf.d
      target: /etc/nginx/conf.d
      type: bind
    - source: /var/proxy/pki/tls/
      target: /etc/pki/tls/
      type: bind
  speedtest:
    image: linuxserver/librespeed
    environment:
      PUID: "1000"
```

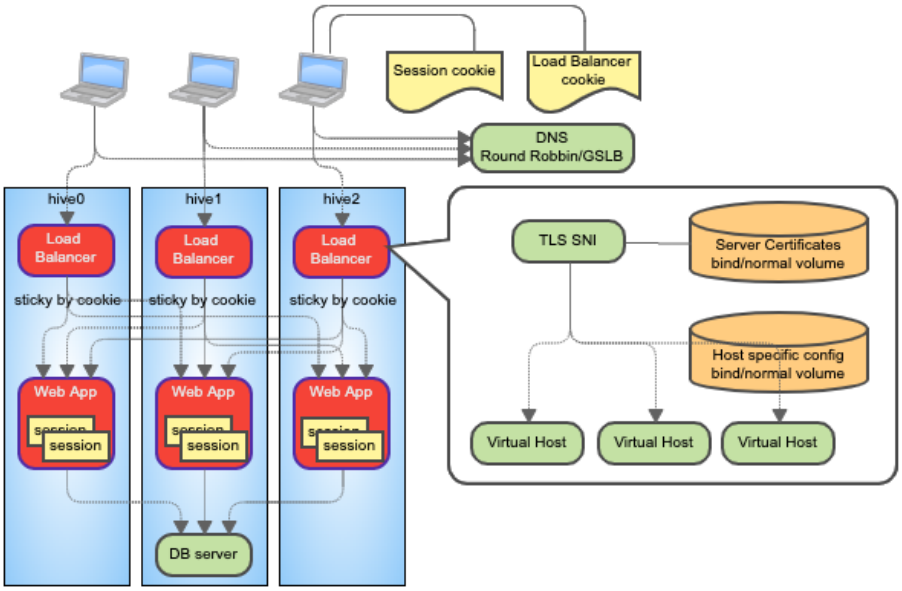
(次のページに続く)

(前のページからの続き)

```
      PGID: "1000"
      TZ: Asia/Tokyo
      DB_TYPE: mysql
      DB_NAME: speedtest
      DB_HOSTNAME: stdb
      DB_USERNAME: speedtest
      DB_PASSWORD: speedtest
    mode: replicated
    replicas: 2
    endpoint_mode: dnsrr
```

7.5.6 スティッキー対応内蔵ロードバランサ

セッション情報などをコンテナごとのメモリ上に保持するようなアプリケーションであるため、前項のパターンのスティッキーセッションを利用できないことが問題となる場合は、Web アプリケーションをグローバルで配置し、一対一ルーティングでパケットを転送することで、内蔵ロードバランサからホストの IP アドレスに対してスティッキーセッションで負荷分散することができます。以下に図を示します。



DB Server については、ステートレスメッシュルーティングの項で記載した内容から変更がありませんので、説明を省略します。それぞれの軸のパターンを以下のように選択します。

Load Balancer

パターン軸	パターン
負荷分散軸	シングルトン
ボリューム軸	
公開軸	一対一ルーティング

Web App

パターン軸	パターン
負荷分散軸	グローバル
ボリューム軸	ステートレス
公開軸	一対一ルーティング

- クライアントからのアクセスは GSLB で冗長負荷分散され、コンテナ収容サーバごとに 1 個ずつ配備されたコンテナで受信します
- 構成情報やサーバ証明書はホスト固有パターンのボリュームに保存することで、サービスを再起動せずに構成変更やサーバ証明書を交換できるようにします
- 受診時の送信元 IP はクライアントの IP アドレスとなり、アクセスログに記録することが可能です
- ロードバランサからの負荷分散でスティッキーセッションを使用することで、セッション情報などをコンテナごとのメモリ上に保持するようなアプリケーションにも適用できます
- Web App のポート公開については、外部への公開をしないことをマークするために 10000 以上のポート番号を付与してください
- 内蔵ロードバランサからアップストリームへの転送は Web App のサービス名ではなく、コンテナ収容サーバのホスト名で行ってください

以下に設定例を示します。

```
---
plugin: hive_services
services:
  proxy:
    image: "procube/nginx:latest"
    mode: global
    endpoint_mode: dnsrr
    ports:
    - target_port: 80
      published_port: 80
```

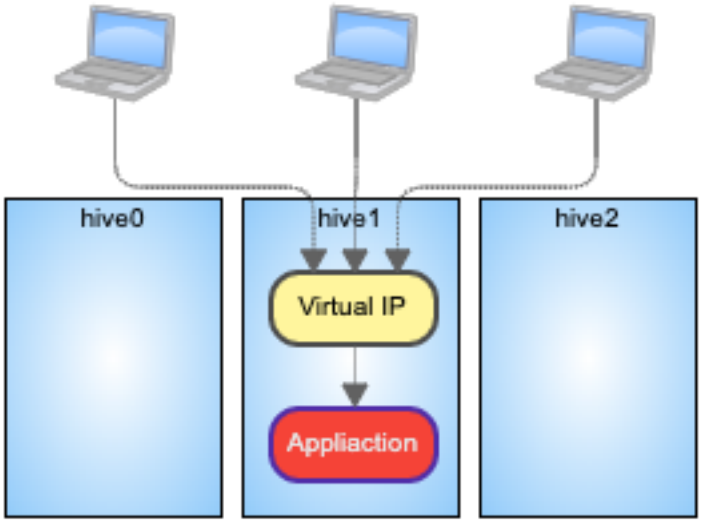
(次のページに続く)

(前のページからの続き)

```
    protocol: tcp
    mode: host
  - target_port: 443
    published_port: 443
    protocol: tcp
   mode: host
  volumes:
  - source: /var/proxy/nginx/conf.d
    target: /etc/nginx/conf.d
    type: bind
  - source: /var/proxy/pki/tls/
    target: /etc/pki/tls/
    type: bind
speedtest:
  image: linuxserver/librespeed
  environment:
    PUID: "1000"
    PGID: "1000"
    TZ: Asia/Tokyo
    DB_TYPE: mysql
    DB_NAME: speedtest
    DB_HOSTNAME: stdb
    DB_USERNAME: speedtest
    DB_PASSWORD: speedtest
  mode: global
  endpoint_mode: dnsrr
  ports:
  - target_port: 80
    published_port: 10080
    protocol: tcp
    mode: host
```

7.5.7 仮想 IP

オンプレミスのサーバでは、サービスをシングルトンで配備し、Swarm 拡張機能で、仮想 IP を利用することができます。以下に図を示します。



Application の軸のパターンを以下のように選択します。

Application

パターン軸	パターン
負荷分散軸	シングルトン
ボリューム軸	サービス固有
公開軸	仮想 IP

- コンテナ 1 個で実装され、負荷分散はありません
- 受診時の送信元 IP はクライアントの IP アドレスとなり、アクセスログに記録することが可能です
- サービス固有ボリュームを持つことでステートフルなアプリケーションでも実装可能です

以下に設定例を示します。

```
---
plugin: hive_services
services:
  speedtest:
    image: linuxserver/librespeed
    environment:
```

(次のページに続く)

(前のページからの続き)

```
    PUID: "1000"
    PGID: "1000"
    TZ: Asia/Tokyo
    DB_TYPE: mysql
    DB_NAME: speedtest
    DB_HOSTNAME: stdb
    DB_USERNAME: speedtest
    DB_PASSWORD: speedtest
  mode: replicated
  replicas: 1
  endpoint_mode: dnsrr
  ports:
  - target_port: 80
    published_port: 80
    protocol: tcp
    mode: host
  - target_port: 443
    published_port: 443
    protocol: tcp
    mode: host
```

第 8 章

hive コマンド

hive コマンドはいくつかのサブコマンドを指定して利用します。以下にその形式とオプションを説明します。

support tool to build docker site

```
usage: hive [-h] [-r ROOT_DIR] [-s {production,staging,private}]
            [-i INVENTORY_PATH] [-c CONTEXT_DIR] [-W TEMP_DIR]
            [-P PLAYBOOKS_DIR] [-v]
            {build-infra,setup-hosts,build-images,build-volumes,build-networks,deploy-
↪services,initialize-services,all,inventory,init,set,ssh,install-collection,list-hosts,
↪list-services,list-volumes,get-install-dir,setup-bash-completion}
            ...
```

8.1 Named Arguments

-r, --root_dir directory where hold project inventory/settings(default: current directory)

Default: /home/docs/checkouts/readthedocs.org/user_builds/hive-builder/checkouts/latest/docs

-s, --stage Possible choices: production, staging, private

stage name which is passed to ansible-playbook with --limit option as inventory group name

-i, --inventory_path path of inventory

-c, --context_dir directory where save context(ex. VagrantFile, CA key, password).

-W, --temp_dir directory where save temporary file(ex. ansible.cfg, vars.yml).

-P, --playbooks_dir path of hive playbooks(default: where hive.py is installed)

-v, --verbose output verbose log

Default: False

8.2 Sub-commands

8.2.1 build-infra

build infrastructure, setup networks, global ip, firewall

```
hive build-infra [-h] [-H] [-D] [-C] [-l LIMIT_TARGET]
```

Named Arguments

-H, --halt stop vpc/subnet/host

Default: False

-D, --destroy destroy vpc/subnet/host

Default: False

-C, --check_mode check mode of ansible

Default: False

-l, --limit_target limit target

8.2.2 setup-hosts

setup hosts, install software, configure services, configure cluster

```
hive setup-hosts [-h] [-T TAGS] [-C] [-l LIMIT_TARGET]
```

Named Arguments

-T, --tags select task

-C, --check_mode check mode of ansible

Default: False

-l, --limit_target limit target

8.2.3 build-images

build container images

```
hive build-images [-h] [-C] [-l LIMIT_TARGET]
```

Named Arguments

-C, --check_mode check mode of ansible

Default: False

-l, --limit_target limit target

8.2.4 build-volumes

build volumes on hives

```
hive build-volumes [-h] [-D] [-l LIMIT_TARGET] [-x LIMIT_SERVER]
```

Named Arguments

-D, --destroy destroy volume

Default: False

-l, --limit_target limit target

-x, --limit_server limit server on build-volumes

8.2.5 build-networks

build networks for swarm

```
hive build-networks [-h]
```

8.2.6 deploy-services

deploy services

```
hive deploy-services [-h] [-D] [-C] [-l LIMIT_TARGET]
```

Named Arguments

-D, --destroy destroy service

Default: False

-C, --check_mode check mode of ansible

Default: False

-l, --limit_target limit target

8.2.7 initialize-services

initialize services

```
hive initialize-services [-h] [-C] [-l LIMIT_TARGET]
```

Named Arguments

-C, --check_mode check mode of ansible

Default: False

-l, --limit_target limit target

8.2.8 all

do all phase

```
hive all [-h]
         [-S {build-infra,setup-hosts,build-images,build-volumes,build-networks,deploy-
↪services,initialize-services}]
```

Named Arguments

-S, --start_phase Possible choices: build-infra, setup-hosts, build-images, build-volumes, build-networks, deploy-services, initialize-services

default start phase. before hive command execute a phase specified by command line, all preceeding phase are executed implicitly from this value. if success to execute the stage, then set persistently (saved into .hive/persistent_values.yml) the next stage to start_phase.

8.2.9 inventory

list ansible inventory

```
hive inventory [-h]
```

8.2.10 init

initialize hive environment

```
hive init [-h]
```

8.2.11 set

set hive variable persistently

```
hive set [-h] variable_name value
```

Positional Arguments

variable_name variable name

value variable value

8.2.12 ssh

ssh to hive server

```
hive ssh [-h] [-t SSH_HOST] [-z] [-Z FOWARD_ZABBIX_PORT] [-L PORT_FORWARDING]
```

Named Arguments

-t, --ssh_host target host

-z, --foward_zabbix if true, forward zabbix web console to localhost on ssh

Default: False

-Z, --foward_zabbix_port port number for forwarding zabbix port

-L, --port_forwarding port forwading on ssh

8.2.13 install-collection

install ansible collections

```
hive install-collection [-h]
```

8.2.14 list-hosts

return hive hosts list

```
hive list-hosts [-h]
```

8.2.15 list-services

return services list

```
hive list-services [-h]
```

8.2.16 list-volumes

return volumes list

```
hive list-volumes [-h]
```

8.2.17 get-install-dir

return install dir

```
hive get-install-dir [-h]
```

8.2.18 setup-bash-completion

setup hive command bash completion

```
hive setup-bash-completion [-h]
```

8.3 変数

hive コマンドは様々な変数を参照しながら動作します。変数の値は以下の順序で決定します。

- root_dir にカレントディレクトリをセット
- install_dir に hive がインストールされているディレクトリをセット
- local_python_path に python コマンドの絶対パスをセット
- コマンドラインで --root-dir が指定されている場合は、 root_dir にセット
- context_dir に {root_dir}/.hive' をセット
- 永続変数を {context_dir}/persistent_values.yml からロード
- 変数 stage の値が設定されていなければデフォルト値をセット

- 永続変数で global 値を持つものをセット
- 永続変数で stage 固有値を持つものをセット
- コマンドラインの root_dir 以外の変数値をセット
- デフォルト値のうちまだ設定されていないものをセット
- フェーズを実行するサブコマンドの場合は phase 変数にサブコマンド名をセット

8.4 ログレベル

--verbose を指定するか set サブコマンドで verbose 変数に True を設定することでデバッグログを出力することができます。

8.5 .hive ディレクトリ

hive コマンドを実行するとカレントディレクトリの下に .hive ディレクトリが生成され、様々なコンテキストを保存します。- .hive/persistent_values.yml には hive の永続変数が保存されます。

8.6 作業ディレクトリ

hive コマンドを実行すると/var/tmp/hive ディレクトリが生成され、hive の作業ディレクトリとして利用されます。

8.7 マザーマシンからサーバへのアクセス

マザーマシンからは各サーバの ssh にアクセスします。build-infra の playbook では、処理の最後に ssh_config をコンテキストディレクトリに出力します。以降の ansible からの ssh アクセスはこのファイルに基づいて行われるため、サーバ名が DNS や hosts ファイルで解決できる必要はありません。build-images のフェーズでは、イメージ構築用の playbook をリポジトリサーバに転送してコンテナイメージを構築します。

8.8 ステージング

ステージングはインベントリ内でステージごとのグループを定義し、そのグループ名で対象となるホストを切り替えます。ステージ名は hive コマンドの -s オプションでにより指定できます。有効なステージの値は production, staging, private であり、ステージのデフォルト値は private です。

```
hive set stage ステージ名
```

を実行することで、以降の hive コマンド実行時のステージを指定できます。このコマンドにより、ステージ名が .hive/persistent_values.yml に保存されます

8.9 ステージングの切り替え

グループ名でインベントリ内のどのホストを対象とするかを切り替えるので、同一の役割のホストでもステージが異なる場合はその名前が異なる必要があり、staging ステージのホスト名には s-、 privaite ステージのホスト名には p- のプリフィックスを付与されます。サービス、ボリューム、イメージ、ネットワークなどのリソースはデフォルトですべてのステージで有効です。available_on を指定して、有効なステージを指定することができます。プロジェクトのロール内でステージ固有の挙動を行う場合は、hive_stage 変数の値で挙動を切り替える必要があります。

8.10 hive コマンドを使わずに playbook を実行

hive コマンドを使わずに playbook を実行する場合は、ANSIBLE_CONFIG 環境変数に/var/tmp/hive/ansible.cfg を指定し、ansible の変数を/var/tmp/hive/vars.yml から読み込んでください。また、 -l オプションにステージ名を指定し対象を絞り込んでください。例えば、private ステージで test.yml を実行する場合は、以下のように指定してください。

```
ANSIBLE_CONFIG=/var/tmp/hive/ansible.cfg ansible-playbook -e @/var/tmp/hive/vars.yml -l␣
↪private test.yml
```

ただし、この場合、hive の組み込み変数のいくつかが使えません。hive の組み込み変数を参照したい場合は、playbook で以下のように変数の定義を読み込んでください。

```
vars_files:
- "{{ hive_playbooks_dir }}/group_vars/hosts.yml"
```

8.11 hive コマンドを使わずに ssh/scp を実行

hive コマンドを使わずに ssh/scp を実行する場合は、コンテキストディレクトリの ssh_config ファイルを使用してください。例えば、hive0.pdns の /etc/hosts ファイルをカレントディレクトリにコピーする場合は、以下のコマンドを実行してください。

```
scp -F .hive/production/ssh_config hive0.pdns:/etc/hosts .
```

第 9 章

インベントリ

hive-builder のインベントリは以下の 2 つからなります。

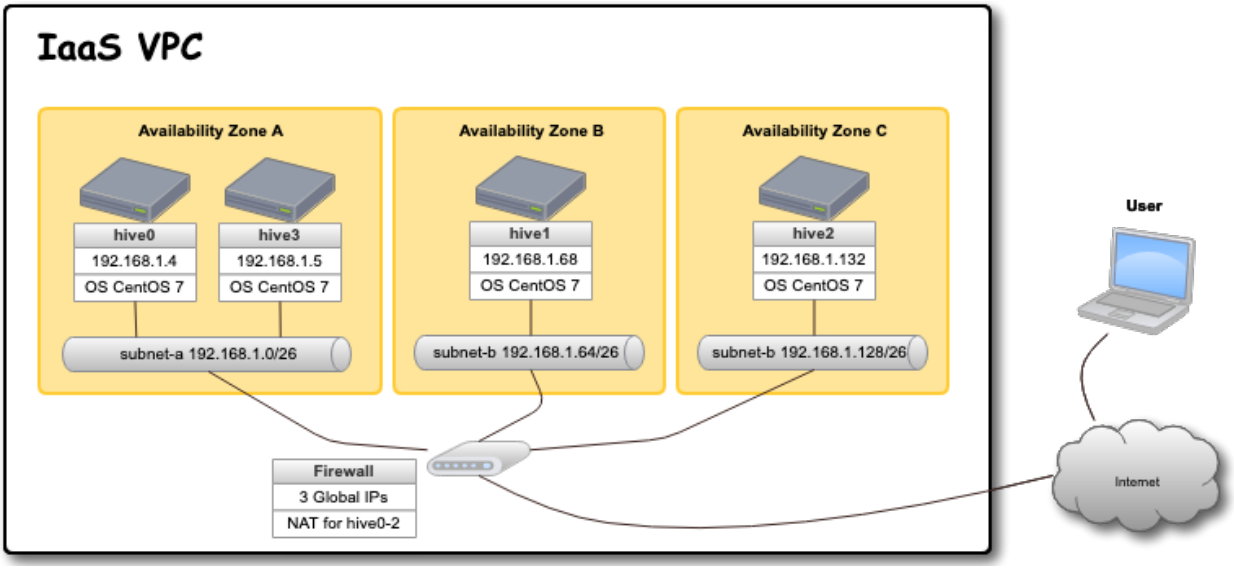
hive 定義
: サーバやネットワークで構成される基盤（infrastructure）を定義する

サービス定義
: コンテナイメージ、ボリューム、内部ネットワークで構成されるサービスを定義する

9.1 hive 定義

hive 定義では基盤を構成するサーバやネットワークを記述します。

IaaS 上にサイトを構築する場合、コンテナ収容サーバは 3 つの Availability Zone に分かれて配置されます。以下にその例を示します。



9.1.1 hive 定義のフォーマット

以下の形式を持つ Yaml ファイルを inventory ディレクトリの下に置くことで hive を定義できます。

パラメータ	選択肢/例	デフォルト	意味
plugin	hive_inventory	必須	このファイルが hive 定義で有ることを示す
name	pdns	必須	hive の名前
stages			ステージオブジェクトへの辞書 private, staging, production の 3 つが定義できる

stages の下にステージ名をキーとしてステージオブエクトを指定します。以下に例を示します。

```
plugin: hive_inventory
stages:
  private:
    provider: vagrant
    separate_repository: False
    cidr: 192.168.0.96/27
    memory_size: 4096
    mirrored_disk_size: 10
    number_of_hosts: 1
  production:
    provider: aws
    separate_repository: False
    cidr: 192.168.0.0/24
    instance_type: t3.medium
    region: ap-northeast-1
    mirrored_disk_size: 20
    repository_instance_type: t3.large
    subnets:
    - cidr: 192.168.0.0/26
      name: subnet-a
      available_zone: ap-northeast-1d
```

(次のページに続く)

(前のページからの続き)

```
  - cidr: 192.168.0.64/26
    name: subnet-b
    available_zone: ap-northeast-1b
  - cidr: 192.168.0.128/26
    name: subnet-c
    available_zone: ap-northeast-1c
```

この例では、private ステージと production ステージが定義されています。private ステージは vagrant で VirtualBox 上に 4G のメモリのサーバ 1 台を構築します。

production ステージでは aws 上に t3.medium のサーバ 3 台を東京リージョンの 3 つの可用性ゾーンに分けて構築します。リポジトリサーバは 3 台目のコンテナ収容サーバと兼用し、t3.large で構築します。

9.1.2 ステージオブジェクト

ステージオブジェクトのフォーマットは以下の通り。

パラメータ	選択肢/例	デフォルト	意味
provider	• aws • azure • gcp • prepared • kickstart • vagrant	必須	基盤を提供するシステム
cidr	192.168.1.0/24	必須	ネットワークのアドレス
number_of_hosts	1	4 if separate_repository else 3	サーバの台数
separate_repository	• True • False	True	リポジトリサーバをコンテナ収容サーバとは別に建てるか否か
subnets	後述		サブネットの定義のリスト
ip_address_list	["192.168.20.5","192.168.20.6"]		IP アドレスのリスト
disk_size	16	プロバイダによる	コンテナ収容サーバの起動ディスクのサイズ（GBytes)
repository_disk_size	16	プロバイダによる	リポジトリサーバの起動ディスクのサイズ (GBytes)
mirrored_disk_size	16	必須	drbd で複製同期するディスクのサイズ（GBytes)
root_password	X12bv5riykfid	""	最初に ssh でログインするためのユーザを作成する必要がある場合に root のパスワードを指定する
internal_cidr	10.254.0.0/16	172.31.252.0/22	docker コンテナがコンテナ間やホストとの通信に使用するネットワークのアドレス
nameservers	["192.168.0.1", "192.168.0.2"]	""	kickstart プロバイダの場合に DNS のサーバの IP アドレスを指定する
custom_hostname	test-hive	hive	独自のホスト名を設定したい場合に指定する（大文字アルファベットの使用不可）

上記はプロバイダ共通の属性ですが、プロバイダ固有の属性もあります。以下にプロバイダ固有の属性をプロバイダごとに説明します。

IP アドレスと可用性ゾーンの割当

IP アドレスと可用性ゾーンの割当は以下のルールで行われます。ただし、subnet 属性はプロバイダごとに指定できなかったり、必須であったりしますので、プロバイダごとに割当方法が異なります。

subnets が指定されている場合：サーバは subnets に指定された subnet オブジェクトに順に割り振られます。サーバの台数のほうが subnet オブジェクトの数よりも大きい場合は、先頭に戻ります。サーバの IP アドレスは subnet オブジェクトに指定された CIDR から自動的に割り振ります。サーバは subnet オブジェクトの available_zone 属性で指定された可用性ゾーンに配備されます。

ip_address_list が指定されている場合：サーバの IP アドレスは ip_address_list から順に割り当てられます。ip_address_list の要素数はサーバの台数と一致しなければなりません。アベラブルゾーンは自動的にできるだけ分散するように割り振ります。

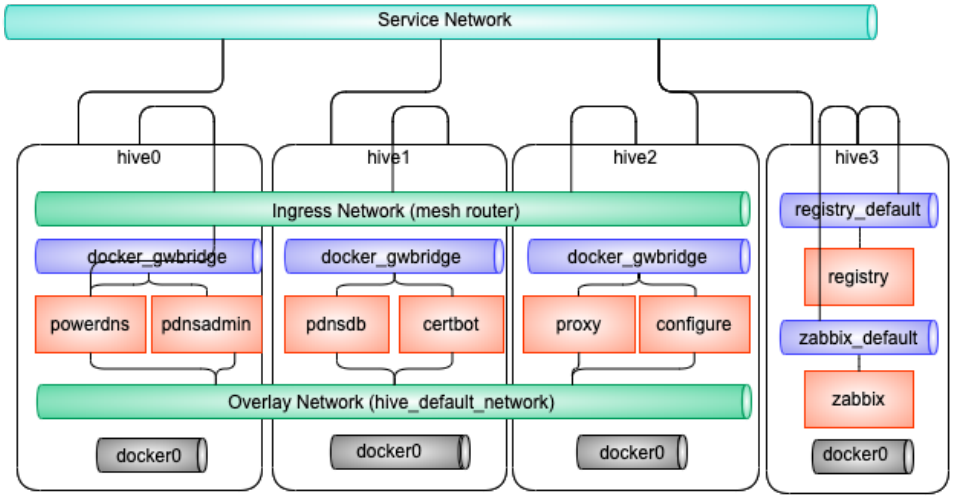
上記以外の場合：サーバの IP アドレスは cidr 属性から自動的に割り振ります。アベラブルゾーンは自動的にできるだけ分散するように割り当てます。

docker が内部的に使用するネットワークのアドレス

コンテナ間通信やコンテナとホスト間通信には docker が必要に応じて作成する仮想ブリッジが使用されます。このような仮想ブリッジ内でコンテナやホストに割り当てられる IP アドレスは外から見えることはありませんが、コンテナから外部に通信する際に IP アドレスが重複していると接続することができません。このため、仮想ブリッジのネットワークのアドレスも外部とかぶらないように割り当てる必要があります。これらのネットワークのアドレスは、デフォルトでは docker デーモンが適宜 LOCAL な IP アドレス領域から割り当てますが、hive 定義の internal_cidr プロパティで変更できます。以下に swarm クラスタで利用される仮想ブリッジの種別について説明します。

種別	説明	接続の条件	比率
Overlay	docker swarm クラスタに分散して配置されるコンテナ間で通信するためのネットワークで、異なるサーバのコンテナと仮想的なネットワークで通信する。管理セグメント内をトンネリングして実装される VPN である。	サービスのコンテナはデフォルトで接続される	1/4
Ingress	docker swarm クラスタで内のコンテナが負荷分散モードでサービスを公開するときに利用するネットワークで、サーバ間の仮想的なネットワークで通信する。管理セグメント内をトンネリングして実装される VPN である。	サービスのコンテナがポートを公開すると接続される	1/4
docker_gwbridge	サーバ内で動作している swarm モードのコンテナが属するネットワークでサーバ上に設置されたブリッジを経由し外部と接続できる。このとき、コンテナ側のアドレスは NAT により、サーバのアドレスに付け替えられる。	サービスのコンテナはデフォルトで接続される	1/8
docker0	swarm モードでないコンテナが属するネットワークでサーバ上に設置されたブリッジを経由し外部と接続できる。このとき、コンテナ側のアドレスは NAT により、サーバのアドレスに付け替えられる。	非 swarm モードのコンテナのみが接続される	1/8
registry_default	リポジトリサーバ上のリポジトリサービスが使用するネットワークである。コンテナ側のアドレスは NAT により、サーバのアドレスに付け替えられるが、リポジトリサーバ内の別のコンテナへの通信の場合は付け替えられない。	リポジトリサービスのコンテナのみが接続される	1/8
zabbix_default	リポジトリサーバ上の zabbix が使用するネットワークである。コンテナ側のアドレスは NAT により、サーバのアドレスに付け替えられるが、リポジトリサーバ内の別のコンテナへの通信の場合は付け替えられない。	zabbix のコンテナのみが接続される	1/8

ここで、比率とは internal_cidr で指定されたネットワークを分割して割り当てる際の比率を示しています。以下に上記種別の接続関係の例を簡略化して図に示します。



vagrant プロバイダ

vagrant プロバイダを利用するには、 Vagrant がインストールされていて、 virtualbox か libvirt の Vagrant プロバイダがセットアップされている必要があります。また、 vagrant-disksize プラグインがインストールされている必要があります。詳細についてはインストールの章を参照してください。

vagrant プロバイダ固有の属性には以下のものがあります。

パラメータ	選択肢/例	デフォルト	意味
memory_size	4096	Vagrant のデフォルト	コンテナ収容サーバに割り当てるメモリのサイズで (MBytes)
repository_memory_size	4096	Vagrant のデフォルト	リポジトリサーバに割り当てるメモリのサイズで (MBytes)
cpus	2	Vagrant のデフォルト	サーバに割り当てる仮想 CPU の個数
bridge	brHive	''	外部のネットワークへブリッジ経由で接続するための仮想ブリッジをこの名前で生成する
dev	brHive	''	この名前の既設の仮想ブリッジに接続する（Vagrant プロバイダが libvirt である場合のみ利用できる）

- disk_size, repository_disk_size を省略した場合、Vagrant のデフォルトのサイズになります。

- subnets 属性は指定できません
- bridge, dev のどちらも指定しない場合、ホストオンリーネットワークに接続されます。

aws プロバイダ

aws プロバイダを使用する場合はアクセスキーと暗号化鍵を設定する必要があります。その設定手順については *AWS* **の準備** を参照してください。aws プロバイダ固有の属性には以下のものがあります。

パラメータ	選択肢/例	デフォルト	意味
instance_type	t3.medium	必須	コンテナ収容サーバのインスタンスタイプ
repository_instance_type	t3.medium	必須	リポジトリサーバのインスタンスタイプ
region	ap-northeast-1	必須	構築先のリージョン
disk_encrypted	• True • False	False	コンテナ収容サーバの起動ディスクのを暗号化するか否か（AWS プロバイダのときのみ有効）
repository_disk_encrypted	• True • False	False	リポジトリサーバの起動ディスクのを暗号化するか否か（AWS プロバイダのときのみ有効）
mirrored_disk_encrypted	• True • False	False	drbd で複製同期するディスクのを暗号化するか否か（AWS プロバイダのときのみ有効）
kms_key_id	dedfc30c-0f77-4c97-8afc-7772f39eb6d5	False	ディスクを暗号化するために用いるデータキーを暗号化するためのマスタキーの ID
image_name	ami-09ebacdc178ae23b7	False	デフォルト以外の AMI を使用する場合の AMI ID（AMI を変更する場合はルートデバイス名を設定する必要あり）

aws プロバイダを使用する場合は、以下のコマンドで hive の環境に AWS EC2 API の鍵を設定する必要があります。

```
hive set aws_access_key_id アクセスキー ID
hive set aws_secret_access_key アクセスキー
```

aws プロバイダでは hive_ec2_group_rules 変数にルールオブジェクトのリストを指定することで、セキュリティグ

ループのインバウンドルールに独自のものを設定できます。（デフォルトでは、サービスの ports 属性に従ってインバウンドルールが設定されます。ただし、 10000 以上の番号は外部に公開されません。）

ルールオブジェクトの属性は以下の通り。

パラメータ	選択肢/例	デフォルト	意味
group_name	default		アクセス元のセキュリティグループ名。default は自グループを意味する。
proto	tcp,udp, icmp, icmpv6, all	必須	アクセスプロトコル。all はすべてのプロトコルを許可することを意味する。
cidr_ip	0.0.0.0/0		アクセスを許可する IP アドレスの CIDR 表記
from_port	22	-1	アクセスを許可するポート番号の最小値。-1 はすべてのポートを許可することを意味する。
to_port	22	-1	アクセスを許可するポート番号の最大値。-1 はすべてのポートを許可することを意味する。

以下に設定例を示します。

```
hive_ec2_group_rules:
- group_name: default
  proto: all
- cidr_ip: 8.8.8.8/32
  proto: tcp
  to_port: '22'
  from_port: '22'
- cidr_ip: 0.0.0.0/0
  proto: tcp
  from_port: '53'
  to_port: '53'
- cidr_ip: 0.0.0.0/0
  proto: tcp
  from_port: '80'
  to_port: '80'
- cidr_ip: 0.0.0.0/0
  proto: tcp
  from_port: '443'
  to_port: '443'
- cidr_ip: 0.0.0.0/0
  proto: tcp
  from_port: '53'
```

(次のページに続く)

(前のページからの続き)

```
  to_port: '53'
```

この例では、以下の設定になっています。

- セキュリティグループ内のアクセスは自由
- 22/TCP には IP アドレス 8.8.8.8 からのみアクセス許可
- 80/TCP, 443/TCP, 53/TCP, 53/UDP にはどこからでもアクセス許可

gcp プロバイダ

gcp プロバイダ固有の属性には以下のものがあります。

パラメータ	選択肢/例	デフォルト	意味
instance_type	n1-standard-2	必須	コンテナ収容サーバのインスタンスタイプ
repository_instance_type	n1-standard-2	必須	リポジトリサーバのインスタンスタイプ
region	asia-northeast2	必須	構築先のリージョン
filestore_cidr	10.255.0.0/26		Cloud Filestore に割り当てられる IP アドレスの CIDR 表記。サービス定義の volumes 属性で nfs を選択した場合に有効。/24 もしくは/26 の範囲で指定してください。デフォルトでは GCP がランダムに割り当てますが、Hive の構築で docker が内部で使用するネットワークのサブネットと重複する場合があります。その場合、Cloud Filestore と通信できないため本パラメータで重複が発生しないように Cloud Filestore 側のサブネットを調整してください。

gcp プロバイダを使用する場合は、プロジェクトのルートディレクトリに gcp_credential.json という名前でサービスアカウントキーを保持するファイルを置く必要があります。サービスアカウントの作成については、https://cloud.google.com/iam/docs/creating-managing-service-accounts?hl=ja を参照してください。サービスアカウントを作成する際には「Compute 管理者」のロールを割り当ててください。サービスアカウントキーについては、https://cloud.google.com/iam/docs/creating-managing-service-account-keys?hl=ja を参照してください。鍵の形式で JSON を選択して、プロジェクトのルートディレクトリに gcp_credential.json という名前で保存してください。

gcp プロバイダで disk_size 属性を省略した場合、コンテナ収容サーバの起動ディスクのサイズは 25G になります。

azure プロバイダ

azure プロバイダ固有の属性には以下のものがあります。

パラメータ	選択肢/例	デフォルト	意味
instance_type	n1-standard-2	必須	コンテナ収容サーバのインスタンスタイプ
repository_instance_type	n1-standard-2	必須	リポジトリサーバのインスタンスタイプ
region	asia-northeast2	必須	リソースグループのロケーション

azure プロバイダを使用する場合は、 Azure AD アプリケーションを作成し、仮想マシーンやネットワークの課金先のサブスクリプションにサービスプリンシパルとしてアプリケーションを設定（ロールを割り当てる）していただく必要があります。Azure ポータルで作成する場合は、Azure のオフィシャルサイト方法:リソースにアクセスできる Azure AD アプリケーションとサービスプリンシパルをポータルで作成するを参照してください。

作成後、以下のコマンドで、そのクレデンシャルを認証情報として hive 変数に設定してください。

```
hive set azure_client_id ...
hive set azure_secret ...
hive set azure_subscription_id ...
hive set azure_tenant ...
```

azure_subscription_id にはポータルの「サブスクリプション」サービスで、表示されるサブスクリプション ID を設定してください。azure_client_id には、ポータルの「Azure Active Directory」サービスの「アプリの登録」からアプリケーションを選択したときに表示される「アプリケーション（クライアント ID)」の値を設定してください。azure_tenant には、ポータルの「Azure Active Directory」サービスの「アプリの登録」からアプリケーションを選択したときに表示される「ディレクトリ（テナント)」の値を設定してください。azure_secret には、アプリケーション上に作成したシークレットの値を設定してください。シークレットの値は作成時にしか表示されないため、値が不明の場合はシークレットを作り直してください。

region 属性には Azure Location をコードで指定してください。有効な値のリストは Azure Cloud Shell 上で以下のコマンドを実行して取得することができます。

```
Get-AzureRmLocation |Format-Table
```

instance_type 属性、repository_instance_type には VM のサイズをコードで指定してください。有効な値のリストは Azure Cloud Shell 上で以下のコマンドを実行して取得することができます。

```
Get-AzureRmVMSize -Location region 属性の値
```

kickstart プロバイダ

kickstart プロバイダは OS のインストール媒体を生成します。インストール媒体は、USB メモリ、DVD、iso イメージファイルに出力できます。VirtualBox や iDRAC で OS を最初からインストールする際にこのインストール媒体をセカンダリの光学ドライブとしてマウントして利用できます。

kickstart プロバイダを利用するには mother マシンは linux でなければなりません。また、サーバは UEFI ブート可能なものである必要があります。

kickstart プロバイダを利用する場合は、ステージオブジェクトに kickstart_config 属性にサーバのインストールパラメータを指定してください。 kickstart_config 属性には以下の属性が指定できます。

パラメータ	選択肢/例	デフォルト	意味
iso_src	/var/lib/isos/CentOS-7-x86_64-Minimal-2003.iso	必須	元となる OS のインストール媒体（媒体が挿されているデバイス名か ISO イメージのファイル名）
iso_dest	/dev/sda	必須	インストール媒体の出力先（媒体が挿されているデバイス名か ISO イメージのファイル名）
target	/dev/disks/by-path/pcie-pci-0000:65:00.0-scsi-0:2:0:0	sda	インストール先のディスク
networks	後述	必須	ネットワーク定義オブジェクトのリスト

ネットワーク定義オブジェクトには以下の属性を指定できます。

パラメータ	選択肢/例	デフォルト	意味
interface	eth0	必須	設定対象のネットワークインタフェース名
gateway	192.168.1.1	なし	デフォルトゲートウェイ
nameservers	["192.168.1.1", "192.168.1.2"]	なし	DNS リゾルバ
ips	["192.168.1.21", "192.168.1.22"]	なし	IP アドレスのリスト（hive0, hive1 .. の順で割り当てられる）
netmask	"255.255.255.0"	なし	netmask （ips を指定したときのみ使用される）
vlanid	1001	なし	VLAN ID
bonding_interfaces	["eth0", "eth1"]	なし	ボンディングを作成する場合のスレーブインタフェースのリスト

bonding_interfaces を指定すると、ボンディングしたインタフェースが追加されます。ボンディングのオプションは miimon=100,mode=802.3ad,updelay=600 となります。

以下に kickstart_config の例を示します。

```
kickstart_config:
  iso_src: /var/lib/isos/CentOS-7-x86_64-Minimal-2003.iso
  iso_dest: /dev/sda
  target: "/dev/disks/by-path/pcie-pci-0000:65:00.0-scsi-0:2:0:0"
  networks:
  - interface: bond0
    bonding_interfaces:
    - eth0
    - eth1
    ips:
    - 192.168.200.20
    - 192.168.200.21
    - 192.168.200.22
    netmask: 255.255.255.0
```

prepared プロバイダ

prepared プロバイダは OS がインストール済みのホストが事前に用意されている場合に使用します。以下に prepared プロバイダの hive 定義の例を示します。

```
staging:
  provider: prepared
  separate_repository: False
  cidr: 192.168.0.96/27
  ip_address_list:
  - 192.168.0.98
  - 192.168.0.99
  - 192.168.0.100
  root_password: mzYY3qjdvBiD
```

root_password が指定された場合は、build-infra フェーズは鍵認証ではなく root ユーザでパスワードでログインして実行されます。mother 環境が CentOS の場合、 root_password を使用して build-infra を実行するためには、sshpass パッケージが必要ですので、以下を実行して事前にインストールしておいてください。

```
yum install sshpass
```

build-infra フェーズで ssh 鍵を生成し、 hive_admin で指定されたユーザを作成して、 authorized_keys を設定します。

複製同期用デバイスの割当

hive では、デフォルトで複製同期用のデバイスを使用し、そのデバイスに drbd の複製同期ディスクを割り当てます。各プロバイダは自動的に mirrored_disk_size で指定された大きさで複製同期用デバイスを作成しますが、prepared プロバイダの場合は、事前に複製同期用のデバイスを割り当てて、ホストのデバイスとして見えている必要があります。複製同期用のデバイスは以下のうちのいずれかの名前である必要があります。

- /dev/disk/azure/scsi1/lun0
- /dev/sdc
- /dev/sdb
- /dev/vdc
- /dev/vdb
- /dev/xvdb
- /dev/nvme1n1
- /dev/sda

複製同期デバイスですでにパーティションを割り当てている場合でも、その残領域に複製同期ディスクを割り当てることができます。その場合、事前にパーティションテーブルのタイプを parted -l コマンドなどで調べてください。パーティションテーブルのタイプが gpt である場合は以下の変数を inventory/group_vars/all.yml などで設定してください。パーティションテーブルのタイプが msdos である場合は省略可能です。

```
hive_partition_label: gpt
```

また、場合によって、複製同期用デバイスを持たないホストを作成したい場合もあります。その場合、ホストの hive_no_mirrored_device 変数に True を設定することで、当該ホストの複製同期用デバイスが無いものとして扱われます。たとえば、inventory/host_vars/hive2.pdns.yml に以下のように指定すると hive2.pdns には複製同期用のデバイスは無いものとして扱われます。

```
hive_no_mirrored_device: True
```

この変数に True を指定した場合、以下の仕様となります。

- 各プロバイダで複製同期用のディスクを作成しません
- setup-hosts フェーズの drbd のインストール時の複製同期用のディスクの初期化処理をスキップします
- build-volumes フェーズで drbd リソースを作成する際、当該ホスト上では diskless として設定します。

9.2 サービス定義

以下の形式を持つ Yaml ファイルを inventory ディレクトリの下に置くことでサービスを定義できます。

パラメータ	選択肢/例	デフォルト	意味
plugin	hive_services	必須	このファイルが hive 定義で有ることを示す
services	{ ... }	必須	サービス定義の辞書オブジェクト

services 属性にサービスごとにサービス名をキーとしてサービス定義を記述してください。サービス名はホスト名として認識させたい場合があるので、ハイフン以外の記号は使用しないでください。また、 hive- で始まるサービス名は予約されていますので、使用しないでください。

サービス定義には、サービスをどのように構築するかが書かれます。以下の属性を記述できます。

パラメータ	選択肢/例	デフォルト	意味
available_on	["production"]	["production", "staging", "private"]	サービスが有効になるステージ
backup_scripts	後述	[]	バックアップ、リストア、夜間バッチのスクリプト（詳細後述）
command	["--api=yes", "--api-key={{db_password}}"]	イメージの command の値	サービス実行時に entrypoint に与えられる引数（entrypoint が [] の場合、1 個めが実行コマンドとなる）
dns	"192.168.1.2"	""	サービス内で使用する DNS サーバのアドレス（docker service create の --dns オプションと等価）
endpoint_mode	• VIP • DNSRR	VIP	エンドポイント・モード（docker service create の --endpoint-mode オプションと等価）
entrypoint	["/docker-entrypoint.sh"]	イメージの entrypoint の値	サービスの起動時に実行されるコマンド
environment	{"MYSQL_PASSWORD": "{{db_password}}", "MYSQL_HOST": "pdnsdb"}	{}	サービス実行時にプロセスに付与される環境変数（docker service create の --env オプションと等価）
hosts	{"test.example.com" "192.168.1.2"}	{}	サービス内の /etc/hosts ファイルに追加するホスト名をキーとした IP アドレスの辞書（docker service create の --hosts オプションと等価）
monitor_error	["Error:", "Exception"]	[]	サービスが出力するログからエラーメッセージとして検知するものの正規表現パターンの配列
ignore_error	["(favicon.ico|echo.php)", "^ [^[]"]	[]	サービスが出力するエラーメッセージのうち、無視したいものに対する正規表現パターンの配列。monitor_error 属性が定義されている場合のみ有効である。
image	後述	[]	サービスのもととなるコンテナイメージの取得方法（詳細後述）
initialize_roles	["python-aptk", "powerdns"]	[]	サービスのイメージのビルド時に適用される role 名のリスト
labels	{"published_fqdn": "pdnsadmin.pdns.procube-demo.jp"}	{}	サービスに付与されるラベル（ラベル名と値の dict を指定する。値に文字列以外のものを指定すると JSON 文字列化されるが、 constraint などで使用する場合には文字列としてしか参照できないので注意を要する）
logging	{"driver": "journald", "options": {"tag": "powerdns"}}	{"driver": "fluentd", "options": {"fluentd-address": リポジトリサーバ:24224}}	サービスのログ出力方法（docker service create の --log-driver("driver"に指定), --log-opt（"options"に指定） オプションと等価）

9.2.1 volumes 属性

volumes 属性には、そのサービスが利用するボリュームの内容を記述できます。また、必要に応じて build-volumes フェーズでボリュームを作成することができます。以下の属性を持つボリュームオブジェクトのリストを指定してください。

パラメータ	選択肢/例	デフォルト	意味
target	/var/lib/mysql	必須	コンテナ上のマウントポイント
type	volume	bind	マウントタイプ（drbd の場合は volume を指定、ホストのディレクトリをマウントする場合は bind を指定）
readonly	• True • False	False	ボリュームを読み取り専用でマウントするか否か
driver	local	local	ボリュームを作成する際のドライバ（drbd 属性と同時に指定することはできません）
driver_opts	{device: "/dev/sda2"}	{}	ボリュームを作成する際のドライバのオプション（drbd 属性, nfs 属性と同時に指定することはできません）
drbd	{fstype: "xfs", size: "500M"}	omit	drbd のボリュームを作成する場合の作成内容（driver 属性, nfs 属性と同時に指定することはできません）
nfs	{}	omit	nfs のボリュームを作成する場合の作成内容（drbd 属性, driver 属性と同時に指定することはできません）

drbd 属性、driver 属性、 nfs 属性のいずれかを指定すると対応するボリュームが build-volume フェーズで作成されます。nfs 属性、driver 属性の使用方法については *NFS* **の利用** を参照してください。

drbd 属性

hive 環境では docker swarm の機能により、コンテナがサーバ間を移動するため、ボリュームは原則として drbd によりすべてのコンテナ収容サーバ間で複製同期しておく必要があります。このためには、volume 属性に drbd 属性を指定してください。drbd 属性はオブジェクトであり、以下の属性を指定できます。

パラメータ	選択肢/例	デフォルト	意味
fstype	• xfs • ext4	必須	ファイルシステムのタイプを指定
size	100M	必須	ボリュームのサイズを 100M, 20G などのように単位を付与した文字列で指定
diskless	['s-hive1.pdns', 'hive1.pdns']	[]	このボリュームを diskless とするサーバのリスト

fstype について、小さいボリュームに xfs を指定すると、領域のムダが大きく、また、フォーマットでエラーになる場合があります。100M 以下のボリュームについては ext4 が推奨されます。大きいボリュームでは xfs が推奨されます。xfs でフォーマットすると、ボリュームの初期データは docker のコンテナにマウントする際にディレクトリが空っぽであることを契機としてマウント前のデータがコピーされます。ext4 でフォーマットすると lost+found ディレクトリが存在するため、空っぽであると認識されずこの機能は動作しませんので、注意が必要です。

9.2.2 image 属性

image 属性には、そのサービスを構成する docker コンテナのイメージの取得方法を記載してください。image 属性には、以下のいずれかを設定できます。

- タグ指定（文字列が指定された場合）
- ビルド方法指定（オブジェクトが指定された場合）

タグ指定

image 属性に文字列を指定すると、それはイメージのタグとみなされます。サービスの起動時には、 docker pull により、タグに対応するイメージをダウンロードします。

ビルド方法指定

image 属性にオブジェクトを指定すると、イメージのビルド方法の指定とみなされます。この場合、その内容に従って、 build-images フェーズでコンテナイメージがビルドされ、リポジトリサーバのプライベートリポジトリに push されます。ビルド方法指定には以下の属性が指定できます。

パラメータ	選択肢/例	デフォルト	意味
from	mariadb:10.4	必須	ビルドのもととなるイメージのタグ
roles	['python-aptk', 'powerdns']	必須	ビルド時に適用する role のリスト（対応する role が roles ディレクトリ配下に定義されていなければならない）
standalone	• True • False	False	ビルド時にスタンドアローン型としてビルドするか否か
env	{"HTTPD_USER": "admin"}	{}	イメージの設定：ビルドで追加される環境変数
stop_signal	"2"	"15"	イメージの設定：コンテナを終了させる場合にルートプロセスに送られるシグナルの番号
user	admin	root	イメージの設定：コンテナでルートプロセスを起動する際のユーザ ID
working_dir	/home/admin	/	イメージの設定：コンテナでルートプロセスを起動する際の作業ディレクトリ
entrypoint	/docker_entrypoint.sh	"[]"	イメージの設定：コンテナの entrypoint
command	["--api-port","5000"]	"[]"	イメージの設定：コンテナのデフォルト command
privileged	• True • False	False	イメージの設定：コンテナのプロセスに特権を与えるか否か
pull_on	["production"]	省略可能	指定されたステージではイメージのビルドを行わず、 pull_from に指定されたイメージを使用する
pull_from	procube/certbot	省略可能	pull_on に指定されたステージでサービスを deploy する際のイメージのタグ。イメージはタグのリポジトリからダウンロードされる。

ビルトインロール

hive_builder ではビルトインロールと呼ばれる組み込み式のロールがあります。これはサービスごとの image.roles 属性の下に追加することで使用でき、イメージのビルド時に role 定義を行う必要がありません。

現在、hive_builder には 4 つのビルトインロールがあります。以下に詳細を示します。

- python-aptk

build-images フェーズでは、 ansible で中身を構築するため、ビルド用に起動したコンテナに python がインストールされていなければなりません。しかし、 ubuntu や alpine をベースとしたイメージには python がインストールさ

れていないものが多々あります。このような場合、ビルドの最初の role として python-aptk を指定してください。python-aptk には以下のようにタスクが定義されており、ubuntu や alpine をベースとしたコンテナに python をインストールできます。

```
- name: install python
  raw: if [ -x /usr/bin/apt-get ]; then (apt-get update && apt-get -y install python3);␣
↪else (apk update && apk add python3); fi
  changed_when: False
```

- hive-syslog

スタンドアロン型のサービスでは、そのままではリポジトリサーバにログが出力されません。スタンドアロン型の中の syslog をリポジトリサーバに送るためには hive-syslog ビルトインロールを組み込んでください。

- hive-certificate

サービスを構築する上で、複数枚の証明書が必要な場合があります。hive_certificate ビルトインロールでは変数 certificates を指定することで任意のドメイン名、サフィックス、有効期限で証明書の生成が可能です。詳細は **こちら** を確認ください。

- hive-trust-ca

CA 局証明書をコンテナのトラストストアにインストールします。詳細は **こちら** を確認ください。

プライベートリポジトリ上のタグとイメージの共有

build-images でビルドされたイメージはプライベートリポジトリに push されます。このときのタグは、以下のようになります。

リポジトリサーバ名:5000/image_サービス名

例：separate_repository=True の production ステージの powerdns サービスのイメージの場合

```
hive3.pdns:5000/image_powerdns
```

build-images でビルドするイメージを複数のサービスで共有するためには、最初のサービス定義で image 属性にオブジェクトを指定してビルドし、二個目以降のサービスでは image 属性にプライベートリポジトリ上のタグを指定してイメージを参照する必要があります。

外部リポジトリを経由したイメージのリリース

staging ステージの build-images でビルドされたイメージを外部リポジトリに push し、これを production ステージから参照するように設定できます。例えば、imgage オブジェクトで以下のように指定すると、production ステージでは、 build-images でのビルドを skip し、代わりに外部リポジトリからイメージをダウンロードします。

```
pull_on:
  - production
pull_from: procube/pdnsadmin_test
```

staging ステージでテストし、イメージをリリースできる状態になったら、外部リポジトリに push します。例えば、pdnsadmin サービスのイメージを procube/pdnsadmin_test にリリースする場合、以下のコマンドを実行します。

```
$ docker tag s-hive2.pdns:5000/image_pdnsadmin:latest procube/pdnsadmin_test:latest
$ docker push  procube/pdnsadmin_test:latest
```

この後、production ステージで pdnsadmin サービスをデプロイすると、外部リポジトリからイメージがダウンロードされます。

9.2.3 standalone 属性

docker のコンテナはスタンドアローン型とマイクロサービス型の 2 種類に分類することができます。

型	説明
スタンドアローン型	• centos:7 などスーパバイザ機能を持った OS のイメージをベースとして構築する • 実行時には /sbin/init を起動する • systemd により内部のプロセスが管理される
マイクロサービス型	• dockerhub のオフィシャルイメージをベースとして構築する • ベースの OS は Ubuntu や alpine などの軽量 OS を採用する • 実行時にはサービスを提供するプロセス 1 個を起動する

コンテナがスタンドアローン型である場合、standalone 属性に True を指定してください。スタンドアローン型かマイクロサービス型かで、イメージのビルド時の entrypoint の値とデフォルトボリュームの値が異なります。

ビルド時の entrypoint の値

build-images フェーズでスタンドアローン型のコンテナをビルドする場合（standalone 属性が True で image 属性にビルド方法が指定されている場合）は、from に指定されたイメージのデフォルトの entrypoint, command でコンテナを起動します。これにより、ルートプロセスとして /sbin/init が起動され、ビルドが終了してシャットダウンされるまで仮想マシンとして動作し、ansible でコンテナにプロビジョニングすることができます。

build-images フェーズでマイクロサービス型のコンテナをビルドする場合（standalone 属性が False で image 属性にビルド方法が指定されている場合）は、ルートプロセスとして、以下のような sleep をし続ける 1 行のシェルスクリプトが起動されます。

```
/bin/sh -c 'trap "kill %1" int;sleep 2400 &wait'
```

このコマンドでルートプロセスが 40 分間 sleep するため、その間に ansible でコンテナにプロビジョニングできます。ビルドが終了すると、ルートプロセスに INT シグナルが送られ、コンテナは停止します。

デフォルトボリュームの値

サービスがスタンドアローン型である場合、以下のボリュームが volumes に暗黙的に追加されます。

```
- source: '/sys/fs/cgroup'
  target: /sys/fs/cgroup
  readonly: True
- source: ''
  target: /run
  type: tmpfs
- source: ''
  target: /tmp
  type: tmpfs
```

9.2.4 ports 属性

ports 属性にはポート定義のリストを指定できます。ポート定義の属性は以下のとおりです。

Option	Short syntax	Long syntax	Description
published_port and target_port	"8080:80"	{published_port:8080, target_port: 80}	The target port within the container and the port to map it to on the nodes, using the routing mesh (ingress) or host-level networking. More options are available, later in this table. The key-value syntax is preferred, because it is somewhat self-documenting.
mode	Not possible to set using short syntax.	{published_port:8080, target_port: 80, mode: "host"}	The mode to use for binding the port, either ingress or host. Defaults to ingress to use the routing mesh.
protocol	"8080:80/tcp"	{published_port: 8080, target_port: 80, protocol: "tcp"}	The protocol to use, tcp , udp, or sctp. Defaults to tcp. To bind a port for both protocols, specify the -p or --publish flag twice.

また、サービス定義では published_port を省略できます。Short Syntax で "80" のように 1 個のポート番号を記載した場合や、Long Syntax で published_port 属性を省略した場合は、hive-builder が自動的に 61001 から順にポート番号を割り当てます。これらはサービスのホスト変数で調べることができます。たとえば、外からそのポートに接続するためにポート番号を調べる場合、initialize-services から起動される role で以下のように参照することができます。

```
pdns_port: "{{ hostvars['powerdns'].hive_ports | selectattr('target_port', 'eq', 8081) |␣
↪map(attribute='published_port') | first }}"
```

9.2.5 backup_scripts 属性

backup_scripts 属性には、バックアップの採取方法、リストア方法、日次バッチを記述できます。この属性を指定することで、volumes で永続化したデータのバックアップを採取したり、障害発生時にリストアすることができます。バックアップとリストアの運用方法については **バックアップとリストア** を参照してください。

バックアップ定義

backup_scripts 属性はバックアップ定義の配列です。複数指定することで一つのコンテナから複数種類のバックアップを採取できます。バックアップ定義には以下の属性が指定できます。

パラメータ	選択肢/例	デフォルト	意味
name	pdnsdb	必須	バックアップ定義の名称
directory	/etc/letsencrypt	""	コンテナ内の指定されたディレクトリのバックアップを採取する
backup_command	"mysqldump -u powerdns -p{{db_password}} powerdns -r /root/today.sql"	""	バックアップを採取するコマンド。コンテナ内にバックアップファイルを出力するコマンドを指定する。
backup_file	/root/today.sql	""	backup_command 属性で指定したコマンドが出力するバックアップファイルのパス
restore_command	"echo source /root/today.sql | mysql -B -u powerdns -p{{db_password}} -D powerdns"	""	リストアするコマンド。コンテナ内にバックアップファイルからリストアするコマンドを指定する。
restore_file	/root/today.sql	""	restore_command 属性で指定したコマンドが読み込むバックアップファイルのパス
ext	"sql"	""	バックアップファイルをリポジトリサーバに移動する際のファイル名につける拡張子。
cleanup_days_before	3	""	指定された日数を超えて古いバックアップファイルを日次で削除する。例えば、0 を指定すると、当日のバックアップファイルのみが残り、3 を指定すると 3 日前、2 日前、1 日前、当日分の 4 個のバックアップファイルが残る。
batch_scripts	["mysql -u powerdns -p{{db_password}} powerdns -e 'ALTER TABLE records;'"]	""	深夜日次バッチの実行時にバックアップの採取前に指定されたコマンドのリストを順に実行する。

ここで directory 属性とコマンドでバックアップを採取する属性（backup_command, backup_file, restore_command, restore_file, ext）は排他的でどちらか一方しか指定できません。バックアップファイルの名前は backup-{name 属性の値}-{採取日時}.{ext 属性の値} になります。directory でバックアップを採取する場合は拡張子は tar.gz になり

ます。

ディレクトリのバックアップ例

例えば、以下のように指定することで、コンテナ内の /etc/letsencrypt のバックアップを採取できます。

```
backup_scripts:
- name: certificates
  directory: /etc/letsencrypt
  cleanup_days_before: 10
```

MySQL のバックアップ例

例えば、以下のように指定することで、MySQL のデータベースのバックアップを採取できます。

```
- name: pdnsdb
  backup_command: "mysqldump -u powerdns -p{{db_password}} powerdns -r /root/today.sql"
  restore_command: "echo source /root/today.sql | mysql -B -u powerdns -p{{db_password}}␣
↪-D powerdns"
  backup_file: /root/today.sql
  restore_file: /root/today.sql
  ext: sql
  cleanup_days_before: 10
```

第 10 章

AWS の準備

aws プロバイダを使用する場合は AWS マネジメントコンソールから hive-builder 用のユーザを作成して、そのアクセスキーを設定してください。また、暗号化ボリュームを使用する場合は、暗号化鍵を設定してください。

10.1 1. ユーザグループの作成

ユーザにポリシーを適用するためにユーザグループを作成することをおすすめします。ユーザグループを作成し、アクセス許可に組み込みポリシー AmazonEC2FullAccess および AWSKeyManagementServicePowerUser をアタッチしてください。

10.2 2. ユーザの作成

以下の手順でユーザを作成してください。

10.2.1 2-1. ユーザを追加ウィザードの起動

AWS マネジメントコンソールの IAM サービスで、「ユーザ」を開き、「ユーザを追加」ボタンをクリックしてユーザを追加ウィザードの起動してください。

10.2.2 2-2. ユーザを追加

ユーザ名を指定してください。ユーザ名は hive-builder 用のものであることが分かる名前にすることをおすすめします。作成時の「AWS アクセスの種類を選択」で「プログラムによるアクセス」を選択してください。「AWS マネジメントコンソールへのアクセス」は不要ですので、選択しないでください。入力が完了したら、「次のステップ：アクセス権限」をクリックしてください。

10.2.3 2-2. アクセス権限

「ユーザをグループに追加」で手順 1. で作成したユーザグループを選択してください。入力が完了したら、「次のステップ：タグ」をクリックしてください。

10.2.4 2-3. タグ

hive-builder では特に必要なタグはありません。業務の都合に応じてタグを設定してください。入力が完了したら、「次のステップ：確認」をクリックしてください。

10.2.5 2-4. 確認

内容を確認し問題がなければ、「ユーザの作成」をクリックしてください。

10.2.6 2-5. アクセスキーの取得

ユーザを追加に成功すると「成功」が表示され、アクセスキー ID とシークレットアクセスキーが表示されます。シークレットアクセスキーが ** となっている場合は「表示」をクリックして内容を表示してください。アクセスキー ID とシークレットアクセスキーをコピーして hive-builder で以下のコマンドを実行して設定してください。

```
hive set aws_access_key_id アクセスキー ID
hive set aws_secret_access_key アクセスキー
```

10.3 3. EBS 暗号化用の鍵の設定

AWS マネジメントコンソールの KMS（Key Management Service）にアクセスし、AWS マネージド型キーを開き、aws/ebs をクリックしてください。これは、EBS を暗号化する際にデフォルトで利用されるマスタキーです。この鍵の ARN をコピーし hive 定義の kms_key_id に設定してください。変数名が _id となっていますが、設定するのは ID ではなく、 ARN であることに注意してください。ボリュームを暗号化する際は、以下のように disk_encrypted、mirrored_disk_encrypted、repository_disk_encrypted 属性に True を指定してください。

```
disk_encrypted: True
mirrored_disk_encrypted: True
repository_disk_encrypted: True
kms_key_id: "aws/ebs 鍵の ARN"
```

ボリュームを暗号化しない場合はこの設定は不要です。disk_encrypted, mirrored_disk_encrypted, repository_disk_encrypted, kms_key_id 属性を省略してください。

第 11 章

フェーズ

ここでは、 hive-builder のサイト構築機能をフェーズごとに説明します。

11.1 build-infra

ホストとネットワークを作成し、環境を構築します。

（未執筆）

11.1.1 プロバイダ

build-infra フェーズでは、サーバを配備する基盤のプロバイダをステージオブジェクトの provider 属性に指定することで、様々なプロバイダを利用できます。プロバイダとして有効な値は以下のとおりです。

プロバイダ ID	**説明**
vagrant	Vagrant for VirtualBox/libvirt on local machine
aws	Amazon Web Service
azure	Microsoft Azure（未実装）
gcp	Gooble Computing Platform
openstack	Some OpenStack Provider（未実装）
prepared	ssh でアクセス可能なサーバ群
kickstart	OS が未インストールの物理サーバ

vagrant

プロバイダ ID に vagrant を指定した場合、vagrant のプロバイダは libvirt, VirtualBox の順に試して、成功したものを使用します。

11.2 setup-hosts

ホストを設定します。setup-hosts は 3 個の PLAY に分割された 27 個の role からなります。setup-hosts コマンドでは -T オプションで適用する role を限定できます。以下にインストールされるソフトウェアとそれぞれの PLAY で実行される role のタスク内容について説明します。

11.2.1 インストールされるソフトウェア

以下にインストールされるソフトウェアの一覧を示します。

パッケージ名	role	リポジトリ	説明
bridge-utils	base	CentOS yum repository	仮想ブリッジ制御
docker	docker	docker CE repository (1)	docker（ただし、 AWS EC2 の場合は Amazon Linux Repository からインストール）
docker (Python)	docker	PyPI	docker python API
docker-compose	docker-compose	PyPI	docker-compose コマンド
drbd	drbd	procube-open/drbd-rpm(2)	drbd
glibc-common	base	CentOS yum repository	ロケール情報（hive_locale が設定されているときのみ）
iptables	iptalbes	CentOS yum repository	サーバファイアウォール（firewalld は削除します）
libselinux-python	base	CentOS yum repository	SE Linux python API
lsof	base	CentOS yum repository	ファイルディスクリプタ情報採取
mariadb	zabbix	dockerhub	zabbix 用 DBMS
Network-Manager	internal-network	CentOS yum repository	ネットワーク管理サービス
pip	pip-venv	PyPI	Python パッケージマネージャ（pytho3-pip でインストールされたものをバージョンアップ）
python3	pip-venv	CentOS yum repository	Python 処理系
python3-libs	pip-venv	CentOS yum repository	Python 処理系ライブラリ

次のページに続く

表 1 – 前のページからの続き

パッケージ名	role	リポジトリ	説明
python3-devel	pip-venv	CentOS yum repository	Python 処理系開発ツール
python3-pip	pip-venv	CentOS yum repository	Python パッケージマネージャ
python3-setuptools	pip-venv	CentOS yum repository	Python パッケージマネージャ開発ツール
python-dxf	pip-venv	PyPI	Docer registry API
python-virtualenv	pip-venv	CentOS yum repository	Python 仮想環境構築ツール
registry	registry	dockerhub	docker プライベートリポジトリ
strace	base	CentOS yum repository	システムコールトレース
sysstat	base	CentOS yum repository	性能統計情報採取
tcpdump	base	CentOS yum repository	パケットキャプチャ
telnet	base	CentOS yum repository	telnet コマンド
unzip	base	CentOS yum repository	圧縮ファイル解凍
vim	base	CentOS yum repository	テキストエディタ
wget	base	CentOS yum repository	ファイルダウンロード
zabbix	zabbix-agent	zabbix download site (3)	zabbix エージェント
zabbix/zabbix-server-mysql	zabbix	dockerhub	zabbix server
zabbix/zabbix-web-mysql	zabbix	dockerhub	zabbix web UI

(1) docker CE repository https://download.docker.com/linux/centos/docker-ce.repo を yum リポジトリとして登録後、yum でインストール。

(2) procube のオープンソース https://github.com/procube-open/drbd9-rpm からカーネルのバージョンに従ってダウンロード。

- Amazon Linux の場合、9.0.22/drbd9-rpm-amzn2
- カーネルのバージョンが 3.10.0-1127 より小さい場合、 9.0.20/drbd9-rpm
- 上記以外の場合、9.0.22/drbd9-rpm

(3) zabbix repository https://repo.zabbix.com/zabbix/3.0/rhel/7/x86_64/zabbix-release-3.0-1.el7.noarch.rpm をインストール。

11.2.2 hive サーバ設定 PLAY

最初に実行される "setup hive servers" という名称の PLAY では各サーバに共通の role を適用します。以下に各 role について説明します。

base role

base role で実施するタスクについて以下に説明します。

yum の設定

hive_yum_url を指定されている場合は、 CentOS および AlmaLinux の Base, AppStream, Extras リポジトリの yum のダウンロード元として指定します。https://mirrors.cat.net/centos のようにミラーサイトの '/centos' までを切り取ったパスで指定してください。 almalinux の場合も http://ftp.iij.ad.jp/pub/linux/almalinux のように '/almalinux' までを切り取ったパスで指定してください。また、この場合、yum の fastestmirror の機能を無効にします。CentOS のミラーサイトで近いものがわかっている場合は、指定してインストールにかかる時間を短縮できます。

AWS, Azure, GCP などの IaaS の場合は、デフォルトで近くのサイトが設定されている場合が多いので、指定しないほうが良いでしょう。

パッケージのインストール

yum で CentOS の標準パッケージをインストールします。インストールされるソフトウェアの節で示したパッケージのうち、 role 欄が base となっているものをインストールします。sysstat については、インストール後、有効にします。

selinux の Permissive 化

selinux のモードを Permissive に設定にします。

ホスト名の設定

ホスト名を設定します。プロバイダが AWS の場合は、再起動時にホスト名が巻き戻らないように /etc/cloud/cloud.cfg に preserve_hostname: true　の設定を追加します。

デフォルトタイムゾーンの設定

hive_timezone が設定されている場合、その値を OS のデフォルトのタイムゾーンとして設定します。

デフォルトロケールの設定

hive_locale が設定されている場合、その値を OS のデフォルトのロケールとして設定します。この場合、ロケール設定のために glibc-common を追加でインストールします。

sshd の設定

sshd を以下の仕様で設定します。

- パスワードによるログインはできません
- チャレンジレスポンスによるログインはできません
- 送信元 IP に対する DNS への逆引き問い合わせは行いません

NetworkManager へのパッチ

仮想マシンを起動する過程で、インタフェースのデバイスの生成前にサービスが起動してしまい起動に失敗する場合があり、これを回避するパッチをあてます。

```
Bringing up interface eth0: Error: Connection activation failed: No suitable device␣
↪found for this connection.
```

具体的には、NetworkManager-wait-online.service で実行される nm-online コマンドの -s オプションを削除します。

hostsfile role

サーバ間で通信する際に互いを hive0.pdns のような内部名で指定できるように /etc/hosts ファイルに登録します。

ntp-client role

hive_ntp_servers が指定されている場合、その値の NTP サーバから時刻を取得するように chronyd を設定します。

iptables role

iptables をインストールし、 firewalld を削除します。

pip-venv role

python, pip, virtualenv をインストールします。インストールされるソフトウェアの節で示したパッケージのうち、role 欄が pip-venv となっているものをインストールします。

addon role

サイト固有のインストールを実行します。サイトの roles に addon role が定義されていればそれを適用し、そうでなければ何もしません。

internal-network role

hive_internal_net_if が定義されている場合、その値でネットワークインタフェースを設定します。このネットワークには hive_private_ip の値の IP アドレスが付与され、サーバ間のクラスタ通信に利用されます。VPS サービス上の仮想マシンなどで、グローバル IP を持つインタフェースとは別に内部通信用のネットワークを追加できるが、OS には設定されていない状態で提供される場合に利用します。

users role

hive_users が指定されている場合、その値に従ってユーザを追加します。その場合、 hive_user_groups も指定しなければなりません。また、ssh で root によるログインを拒否するよう設定します。

グループの定義

hive_user_groups にはグループ名をキーにしてグループオブジェクトを指定してください。グループに属するユーザは sudo をパスワードなしで実行できるように設定します。グループオブジェクトの属性は以下の通り。

属性名	説明
gid	グループの gid (1 から 2147483647 までの整数)

ユーザの定義

hive_users にはユーザ名をキーにしてユーザオブジェクトを指定してください。ユーザごとの SSH 設定で、公開鍵認証でログインできるように設定し、サーバの鍵を既知のホストとして登録します。ユーザオブジェクトの属性は以下の通り。

属性名	説明
uid	ユーザの uid (1 から 2147483647 までの整数)
group	ユーザの基本グループの gid
id_rsa_pub	ユーザの SSH ログインのための公開鍵

strict-source-ip role

hive_ssh_source_ips が定義されている場合、sshd への接続の送信元 IP アドレスを制限します。hive_ssh_source_ips にはアクセスを許容する IP アドレスをリストで指定してください。また、hive_safe_sshd_port が指定されている場合には、 sshd の受付ポート番号をその値に変更します。

tls-certificate role

docker API および registry API に使用するプライベート証明書を生成します。

docker role

docker をインストールします。

- リモートから docker API を呼び出せるように設定します。
- docker デーモン間の通信を許可します。
- GCP の場合は、docker が仮想ネットワークを利用できるように IP forwarding を可能なように設定します。
- hive 定義に internal_cidr 属性が定義されている場合は、その値の範囲からネットワークアドレスを割り当て docker ネットワークを設定します。

drbd role

drbd をインストールします。

- drbd 間の通信を許容します。
- セカンダリドライブに drbd resource pool を作成します。

docker-client role

docker python API をインストールし、 API クライアントの TLS 認証を設定し、 hive のユーティリティコマンドをインストールします。

follow-swarm-service role

swarm 拡張機能をインストールします。

docker-client-proxy role

プロキシに対応するように docker を設定します。HTTP_PROXY 環境変数が設定されている場合のみに適用されます。

zabbix-agent role

zabbix-agent をインストールします。

11.2.3 リポジトリサーバ設定 PLAY

二番目に実行される "setup repository and zabbix" という名称の PLAY ではリポジトリサーバに共通の role を適用します。以下に各 role について説明します。

docker-compose role

docker-compose をインストールします。

zabbix role

zabbix コンテナをインストールします。

registry role

registry コンテナをインストールします。

backup-tools role

バックアップツールをインストールします。サービス定義にしたがって、バックアップ/リストア用のシェルスクリプトを生成し、夜間バッチでバックアップを実行するように設定します。

rsyslogd role

マイクロサービス型のコンテナのログを受信して記録するように rsyslogd を設定します。

11.2.4 クラスタ構築 PLAY

三番目に実行される "build cluster" という名称の PLAY ではコンテナ収容サーバ間のクラスタ連携を設定する role を適用します。以下に各 role について説明します。

swarm role

docker swarm クラスタを設定します。

- hive 定義に internal_cidr 属性が定義されている場合は、その値の範囲からネットワークアドレスを割り当て docker_gwbridge ネットワークを設定します。
- hive 定義に internal_cidr 属性が定義されている場合は、その値の範囲からネットワークアドレスを割り当て ingress ネットワークを設定します。
- docker swarm ノードとして初期化し、クラスタとして結合します。
- サーバが属する ansible グループ名をノードのラベルとして設定します。

11.3 build-images

コンテナイメージをビルドします。サービス定義で image 属性の下に from 属性を指定した場合にビルドの対象となります。

11.3.1 再実行

build-images フェーズを複数回行う場合、前回のビルドに利用したコンテナを再利用することでビルドにかかる時間を短縮しています。このため、image 属性の配下の属性を変更して build-images をやり直しても反映されません。また、roles に指定したタスクについて内容が減少する方向の変更が行われた場合、反映されません。たとえば、ファイルのインストール先が変更された場合や、設定ファイルの行追加をやめた場合などがこれに該当します。このような場合は、以下の手順でビルド用のコンテナを削除してから build-images をやり直してください。

```
hive ssh
docker rm build_image_サービス名
exit
```

11.3.2 デバッグ

build-images でエラーが発生し、-v オプションで詳細ログをみても原因がわからない場合、以下の手順でビルド用コンテナにログインして build-images で実行する内容をコマンドで実行してみることでデバッグすることができます。

1. リポジトリサーバへのログイン

以下のコマンドでリポジトリサーバにログインしてください。

```
hive ssh
```

2. ビルド用のコンテナの起動

以下のコマンドでビルド用コンテナが起動しているかを確かめてください。

```
CN=build_image_サービス名
docker ps -f name=$CN -a
```

この結果 STATUS 列に Exited が表示される場合は次のコマンドでコンテナを起動してください。

```
docker start $CN
```

3. ビルド用のコンテナへの起動

以下のコマンドでビルド用コンテナにログインしてデバッグしてください。

```
docker exec -it $CN /bin/bash
```

ただし、alpine ベースのコンテナの場合、/bin/bash が入っていない場合があります。その場合、"stat /bin/bash: no such file or directory" というエラーが表示されますので、 /bin/bash のかわりに /bin/ash を利用してください。

作業が終わりましたら、exit を 2 回実行して mother 環境に戻ってください。

11.3.3 外部リポジトリへのログイン

イメージをダウンロードする際に外部リポジトリを利用することができます。外部リポジトリにアクセスする際にログインが必要な場合、 hive_ext_repositories にログインに必要な情報を設定してください。hive_ext_repositories は docker ログインオブジェクトの配列です。 docker ログインオブジェクトは以下の属性を持ちます。

属性名	説明
repository	リポジトリ。FQDN:ポート番号の形式で指定してください。省略すると dockerhub にログインします。
login_user	ユーザ ID
password	パスワード
email	メールアドレス（省略可能）

11.4 build-networks

内部ネットワークを構築します。

（未執筆）

11.5 build-volumes

ボリュームを構築します。

（未執筆）

11.6 deploy-services

サービスを配備します。

（未執筆）

11.6.1 外部リポジトリへのログイン

イメージをダウンロードする際に外部リポジトリを利用することができます。外部リポジトリにアクセスする際にログインが必要な場合、build-images の場合と同様に hive_ext_repositories にログインに必要な情報を設定してください。

11.7 initialize-services

サービスを初期化します。

（未執筆）

第 12 章

swarm 拡張機能

ここでは、hive のコンテナ収容サーバにインストールされる swarm 拡張機能について説明します。swarm 拡張機能では、ノードでサービスのコンテナがデプロイされるのを観測することによって以下の 2 つの機能を提供します。

- 仮想 IP 付与機能
- ラベル付与機能

12.1 仮想 IP 付与機能

仮想 IP 付与機能では、サービスのコンテナがノードにデプロイされると、そのラベルにしたがってノードに仮想 IP を付与する機能を提供します。この機能を利用する場合にはサービスの複製数は 1 に設定してください。これにより、複数のコンテナ収容サーバのうち当該サービスのコンテナがデプロイされた 1 台だけに仮想 IP を付与することができます。また、障害が発生してコンテナが別のサーバに移動した場合に仮想 IP も同じサーバに移動するため、仮想 IP のフェイルオーバが実現され、高可用性を確保することができます。

この機能を利用するたには、サービス定義の labels 属性に以下のラベルを設定する必要があります。

ラベル名	値の例	説明
HIVE_VIP	192.168.0.10	仮想 IP アドレス
HIVE_ROUTER	192.168.0.1	仮想 IP アドレスを付与するインタフェースのルータのアドレス（省略可能）

仮想 IP 付与機能では、 HIVE_VIP ラベルが付いたサービスのコンテナがノードにデプロイされると、そのサーバが持っているインタフェースを調べ、指定された仮想 IP アドレスが付与可能なのインタフェースを探します。インタフェースが見つかった場合、そこに仮想 IP アドレスを付与します。仮想 IP アドレスが付与可能かどうかはそのインタフェースが持っている IP アドレスのネットワークアドレスに仮想 IP が含まれるかどうかで判断します。付与しようとする仮想 IP アドレスを含むネットワークの IP アドレスが事前にサーバに付与されていなければ、仮想 IP アドレスを付与することはできませんので、注意してください。仮想 IP 付与後に Gratuitous ARP を送

信して、隣接するデバイスの ARP テーブルを更新します。HIVE_ROUTER が指定されている場合は、ARP テーブルの更新を確認するために HIVE_ROUTER で指定されたアドレスに ping を送ります。ping に失敗した場合は Gratuitous ARP を送信を再実行します。5 回以上失敗すると処理を中止します。

12.2 ラベル付与機能

ラベル付与機能では、サービスのコンテナがノードにデプロイされると、そのラベルにしたがってノードにラベルを付与する機能を提供します。この機能を利用することで特定のサービス（以降、リーダサービスと呼ぶ）に依存するサービス（以降、メンバサービスと呼ぶ）のグループを構成することができます。リーダサービスの複製数は 1 に設定してください。これにより、複数のコンテナ収容サーバのうち当該サービスのコンテナがデプロイされた 1 台のノードのみにラベルを付与することができます。また、障害が発生してリーダサービスのコンテナが別のサーバに移動した場合にメンバサービスも同じノードに移動するため、フェイルオーバが実現され、高可用性を確保することができます。

この機能を利用するたには、サービス定義の labels 属性に以下のラベルを設定する必要があります。

ラベル名	値の例	説明
HIVE_MARK	zabbix	ノードに付与するラベル名でラベルの値は "true" になります（例えば、zabbix を指定すると zabbix=true というラベルが付与されます）

リーダサービスのサービス定義で zabbix のラベル付与を行う場合、以下のように定義します。

```
labels:
  HIVE_MARK: zabbix
```

メンバサービス側にはラベルが付与されているノードにだけデプロイされるように placement.constraints 属性を指定します。例えば、 zabbix のラベルが付与されているノードにのみデプロイされるようにする場合は、以下のようにサービス定義に指定します。

```
placement:
  constraints:
  - node.labels.zabbix == true
```

12.3 follow-swarm-service デーモン

swarm 拡張機能は follow-swarm-service デーモンにより実装されています。コンテナ収容サーバで journalctl コマンドを利用することでそのログを見ることができます。たとえば、ページャを使用して直近のログを見る場合は、以下のコマンドを実行します。

```
journalctl -e -u follow-swarm-service
```

第 13 章

旧バージョンからの移行

ここでは、すでに構築された環境で hive-builder をバージョンアップした際の移行作業について説明します。

13.1 1 系（1.x.x） からの移行

2 系は 1 系（1.x.x）とサイトの互換性はありません。サイトを再構築するか、1 系の最新版（1.2.3）を利用してください。1 系の最新版を利用する場合は、以下のようにバージョンを指定してバージョンアップしてください。

```
pip install -U hive-builder=1.2.3
```

また、1 系のマニュアルについては、https://hive-builder.readthedocs.io/ja/foros7/ を参照してください。

13.2 2.0.x からの移行

2.0.x では、リポジトリサーバのログ収集機能のポート番号は 514 固定でしたが、 2.1.0 以降では、ポート番号のデフォルトが 10514 に変わりました。バージョンアップ後に既存の環境で deploy-services を実行するとそのサービスはポート番号 10514 にログを送ろうとします。これを避けるためには、 inventory/grup_vars/servers.yml で以下のように指定してください。

```
hive_syslog_port: 514
```

13.3 2.0.1 以前からの移行

2.0.2 からは zabbix の SELinux に対する監視方法が変わりました。既存サイトでは hive-builder をバージョンアップ後、zabbix の Web UI にアクセスし、 Configuration -> Templates で Hive Server SELinux テンプレートを Delete and Clear で削除してください。その後、以下のコマンドを実行してください。

```
hive setup-hosts -T zabbix,zabbix-agent
# 以下をすべてのコンテナ収容サーバで繰り返し
hive ssh -t サーバ名
sudo su -
make -f /usr/share/selinux/devel/Makefile reload
systemctl restart zabbix-agent
exit
exit
```

13.4 2.1.2 以前からの移行

2.2.0 からは zabbix がログを監視するようになりました。また、バックアップスクリプトの生成は deploy-services で行うように変更されました。既存サイトでは hive-builder をバージョンアップ後、zabbix の Web UI にアクセスし、 Configuration -> Templates で Hive Server SELinux テンプレートを Delete and Clear で削除してください。その後、以下のコマンドを実行してください。

```
hive setup-hosts -T zabbix,zabbix-agent,backup-tools,rsyslogd
hive deploy-services
```

第 14 章

hive 内 zabbix の利用

リポジトリサーバではサーバと Swarm サービスを監視する Zabbix が稼働しています。

14.1 監視内容

hive 内 zabbix では、以下の状態が発生するとイベントが発生します。このイベントをメールや Slack などに通知することによって、障害の発生や予兆を検知できます。

- プロセス数が上限の 80% を超えると警告
- CPU 使用率が 5 分間継続して 90% を超えると警告
- メモリ使用率が 5 分間継続して 90% を超えると警告
- スワップの使用率が 5 分間継続して 50% を超えると警告
- メモリ残量が 5 分間継続して 5Mbytes を切ると警告
- ロードアベレージが 5 分間継続して 1.5 を超えるとと警告
- サーバの時刻が 60 秒以上ずれると警告
- サーバの Zabbix エージェントが 3 分以上にわたって応答がない場合は警告
- サーバの再起動が行われると警告
- ディスク I/O の応答時間が 15 分間に渡って 20ms を超えている場合は警告
- マウントしているファイルシステムの使用率が 80% を超えて、かつ残量が 10G を切ると警告 - マウントしているファイルシステムの inode の残量が 5 分間継続して 20% を切ると警告
- ネットワークインタフェースのエラー率が 5 分間継続して 2% を超えると警告
- ネットワークインタフェースのリンクダウンがあると警告

- サービスに対応するコンテナが落ちていると警告
- サービスに対応するコンテナが再起動すると警告
- ボリュームの使用率が 30 分間継続して 90% を超えると警告
- DRBD の同期状態が崩れると警告
- Standalone 型コンテナの内部サービスが落ちていると警告
- Standalone 型コンテナの内部サービスが再起動すると警告
- リポジトリサーバに送られてくるログの中に monitor_error 属性で指定された文字列が含まれていると警告
- SE Linux で設定されたポリシーに違反したアクセスがあると警告
- /etc/passwd の内容が変更されると警告

14.2 Slack に通知する

Zabbix のイベントを Slack に通知するための、設定手順を示します。

14.2.1 slack 側の設定

１．Application を作成

slack に Application を作成してください。Slack の Your Apps の「Create New App」でアプリケーション「hive_builder_zabbix」を作成してください。この名前は任意のもので構いません。

２．Bots を追加

Basic Information の Add features and functionality で Bots を選択してください。

３．OAuth & Permissions

左メニュー「Features」-「OAuth & Permissions」を選択してください。「Scopes」で Add an OAuth Scope を実行して chat: write を追加してください。

４．Access Token コピー

「Bot User OAuth Access Token」の Token 文字列をコピーしておいてください（後ほど Zabbix に設定する）。

14.2.2 hive-builder 設定

１．メディアタイプを作成

「Administration」-「Media Types」で（標準添付の）「Slack」を開き、「Parameters」で以下３点を修正してください。

- 「bot_token」に、前項でコピーした Token を入力
- 「channel」に、通知先のチャネル名を入力
- 「zabbix_url」に、「http://127.0.0.1:10052/」を入力

注釈**:** zabbix_url は slack に通知されるメッセージに「Open in Zabbx」というリンクで埋め込まれます。プロキシサーバなどを使用して hive 内 zabbix に外部からアクセスできるようにしている場合は zabbix_url にその URL を設定するほうが良いでしょう。

参考：https://qiita.com/migaras/items/bc0dde421af9650d109c

第 15 章

バックアップとリストア

サービス定義の backup_scripts にバックアップの採取方法を指定することで、リポジトリサーバにバックアップを採取することができます。

15.1 バックアップファイルの保存先

バックアップは毎日深夜 1 時にリポジトリサーバの ~/backup に backup-{name 属性の値}-{採取日時}.{ext 属性の値} という名前で保存されます。また、最後に採取したバックアップに対して backup-{name 属性の値}-latest.{ext 属性の値} という名前でシンボリックリンクが貼られます。したがって、このディレクトリのからコピーすることで、外部のストレージにバックアップを保存することができます。最後のバックアップをアーカイブするのであれば、リポジトリサーバで以下のコマンドを実行することで bakcup.tar.gz というファイルにまとめることができます。

```
tar cvzhf backup.tar.gz backup/*-latest.*
```

15.2 リポジトリと監視データのバックアップ

リポジトリサーバで動作しているリポジトリサービスと監視サービスのデータもバックアップが採取されます。このときのサービス名は hive-registry と hive-zabbix です。

15.3 リストア方法

リポジトリサーバの ~/backup に各サービスの latest のバックアップが置かれていれば、以下のコマンドでデータをリストアできます。

```
hive-backup.sh  -r
```

サービスごとに個別にリストアする場合は、 -l オプションでサービス名を指定します。例えば、pdnsdb サービスのデータとリポジトリサーバの zabbix のデータをリストアする場合は以下のコマンドになります。

```
hive-backup.sh  -r -l pdnsdb,hive-zabbix
```

15.4 個別バックアップの採取

深夜 1 時の日次バッチ以外のタイミングでバックアップを採取する場合、リポジトリサーバで以下のコマンドを実行してバックアップを採取できます。

```
hive-backup.sh
```

このコマンドで~/backup にすべてのサービスのバックアップを採取します。サービスごとに個別にバックアップを採取する場合は、 -l オプションでサービス名を指定します。例えば、pdnsdb サービスのデータとリポジトリサーバの zabbix のデータをバックアップする場合は以下のコマンドになります。

```
hive-backup.sh  -l pdnsdb,hive-zabbix
```

第 16 章

タグマッピング

build-images フェーズ、 deploy-services フェーズでコンテナイメージをダウンロードする場合に、コンテナイメージが不本意にバージョンアップしていしまい、障害の原因となる場合があります。タグマッピングでは、これを避けるために docker のイメージのタグを固定したり一時的に変更したりする方法を提供します。

16.1 タグマッピング指定

タグマッピングはコンテキストディレクトリ (.hive/ステージ名) に tag-mapping.json というファイルを作成することで利用できます。このファイルにタグマッピングオブジェクトを記述すると、その内容に応じてダウンロードするタグが変換されます。タグマッピングオブジェクトは、マッピング前のタグをキー、マッピング後のタグを値に持つオブジェクトです。例えば、以下のように指定することで、 mariadb:10.5 を mariadb:10.5.12 に、hive3.pdns:5000/image_pdnsrecursor:latest を hive3.pdns:5000/image_pdnsrecursor@sha256:cdf0f7d5ac067a3df4b5e08047e7240d57cc49fff55742e66be5a816cce968c2 に変換することができます。

```
{
  "mariadb:10.5": "mariadb:10.5.12",
  "hive3.pdns:5000/image_pdnsrecursor:latest": "hive3.pdns:5000/image_
↪pdnsrecursor@sha256:cdf0f7d5ac067a3df4b5e08047e7240d57cc49fff55742e66be5a816cce968c2"
}
```

16.2 タグマッピングの対象

イメージをダウンロードする以下の 3 つのケースがタグマッピングの対象となります。

- build-images フェーズでサービス定義の image.from 属性に指定されいるものをダウンロードする場合
- deploy-services フェーズでサービス定義の image 属性に指定されいるものをダウンロードする場合
- deploy-services フェーズでビルドされてリポジトリサーバに登録されているものをダウンロードする場合

最後のケースでは、変換対象のイメージタグは以下のパターンになります。

```
${リポジトリサーバ名}:5000/image_${サービス名}:latest
```

例えば、サーバが 1 台で hive 名が "pdns"、 ステージが "private" 、サービス名が pdnsrecursor の場合は以下のようなイメージタグになります。

```
p-hive0.pdns:5000/image_pdnsrecursor:latest
```

また、リポジトリ上のダイジェスト値を @ に続けて指定することで、最新でない (latest タグが付いていない) ものを指定してダウンロードすることが可能です。例えば、イメージ ID がわかっていれば、

```
docker inspect --format='{{index .RepoDigests 0}}' イメージ ID
```

を実行して、リポジトリ上のダイジェスト値を調べ、

```
p-hive0.pdns:5000/image_
↪pdnsrecursor@sha256:cdf0f7d5ac067a3df4b5e08047e7240d57cc49fff55742e66be5a816cce968c2
```

のように指定することができます。

16.3 タグマッピングの運用

hive-builder のソースコードを git で版管理を行っている場合でも、タグマッピングを利用して、開発途上の最新版を利用するか、動作が確認されている安定版を利用するかを切り分けることができます。ここではその運用方法について提案します。まず、git に登録されるソースコード上でどのようにタグを指定するかについて、開発フェーズと運用フェーズに分けることとします。

16.3.1 開発フェーズ

開発フェーズでは、ソースコード上のサービス定義の image.from 属性や image 属性には、latest など常に最新版を指し示すタグを指定しておきます。このコードを git に登録しておくことにより、git pull で取得したソースコードからそのままビルドすると最新版がダウンロードされます。一方で、結合テスト環境やデモ環境のように安定稼働が重視される環境では、コンテナイメージのバージョンを固定したい場合があります。このような場合はタグマッピング機能でコンテナイメージのバージョンを固定できます。結合テスト環境やデモ環境では、最初に稼働確認が取れた時点ですべてのコンテナイメージのバージョンをタグマッピングで固定することを推奨します。latest などの最新版を指し示すタグに対して稼働確認が取れたバージョンのタグへのマッピングを指定して tag-mapping.json を作成してください。以降イメージを明示的にバージョンアップする場合は tag-mapping.json を編集してバージョンアップの対象となるタグのマッピング先を修正してください。

例：powerdns のコンテナのバージョンを検証環境のテスト中は 4.5.1-1 に固定しておきたい場合、

inventory/powerdns.yml:

```
plugin: hive_services
services:
 powerdns:
  image: procube/powerdns:latest
...
```

.hive/staging/tag-mapping.json

```
{
  "procube/powerdns:latest": "procube/powerdns:4.5.1-1"
}
```

16.3.2 運用フェーズ

運用フェーズでは、ソースコード上のサービス定義の image.from 属性や image 属性には、安定稼働が確認されたバージョンのイメージのタグを指定しておきます。このコードを git に登録しておくことにより、 git pull で取得したソースコードからそのままビルドすると安定版がダウンロードされます。一方で、ステージング環境のように新しいバージョンのテストが優先される環境では、タグマッピング機能でコンテナイメージの最新版がダウンロードされるように設定できます。このような環境では、安定版をを指し示すタグに対して latest などの最新版へのタグへのマッピングを指定して tag-mapping.json を作成してください。

例：powerdns のコンテナのバージョンを運用中の環境では 4.5.1-1 を使用しているが、テスト環境でコンテナを最新にバージョンアップしてテストする場合、

inventory/powerdns.yml:

```
 plugin: hive_services
 services:
  powerdns:
   image: procube/powerdns:4.5.1-1
...
```

.hive/staging/tag-mapping.json

```
{
  "procube/powerdns:4.5.1-1": "procube/powerdns:latest"
}
```

16.3.3 本番環境での切り戻し

本番環境で特定のサービスに対して build-images フェーズ、deploy-services フェーズを実行してコンテナをバージョンアップした後、その内容に問題があって切り戻さなければならない場合があります。このような場合にタグマッピングを利用して切り戻すことが可能です。

```
docker images
```

で、切り戻し対象のイメージを特定し、

```
docker inspect --format='{{index .RepoDigests 0}}' イメージ ID
```

でタグを取得し、これを tag-mapping.json に指定してからサービスをデプロイすることで切り戻しできます。たとえば、タグが hive3.pdns:5000/image_pdnsrecursor@sha256:cdf0f7d5ac067a3df4b5e08047e7240d57cc49fff55742e66be5a816cce968c2 である場合、tag-mapping.json を

```
{
  "hive3.pdns:5000/image_pdnsrecursor:latest": "hive3.pdns:5000/image_
↪pdnsrecursor@sha256:cdf0f7d5ac067a3df4b5e08047e7240d57cc49fff55742e66be5a816cce968c2"
}
```

のように指定し、

```
hive deploy-services -l pdnsrecursor
```

を実行することで切り戻しを実行できます。

16.4 ソフトウェアパッケージのバージョン管理について

タグマッピングではコンテナイメージのバージョンを管理できますが、build-images フェーズで yum, pipy, npmjs などのパブリックリポジトリからダウンロードされるソフトウェアは管理できません。プロジェクトごとに ansilbe コードの中で対応する必要があることにご注意ください。

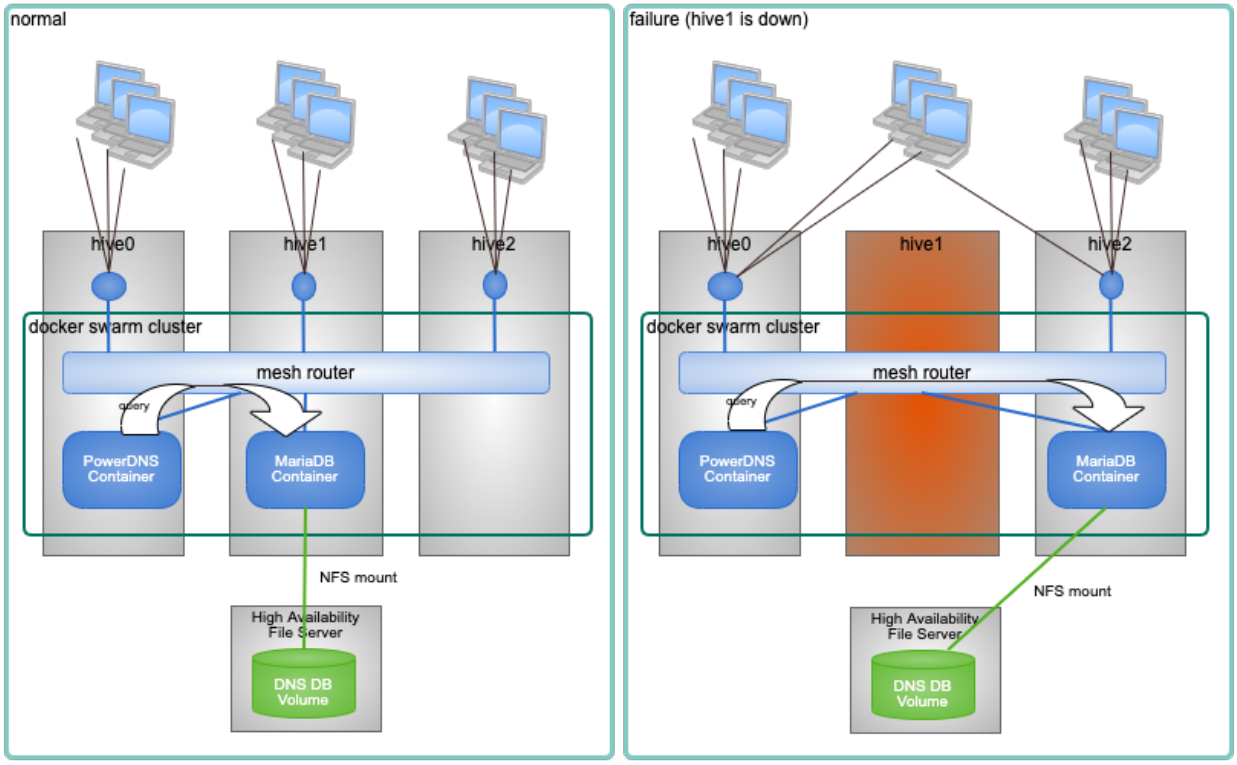
第 17 章

NFS の利用

hive-builder では NFS ボリュームを使用することができます。NFS で高可用性ストレージをマウントすることで、DRBD のディスクミラーリングと同様の高可用性クラスタを構成できます。

17.1 高可用性の実装

NFS を使用したクラスタ構成では、コンテナを収容しているサーバで障害が発生した場合、そのサービスのコンテナを別のサーバに移動し、そこで、コンテナが使用していたファイルシステムを NFS マウントすることで、サービスを継続します。以下にその様子を示します。



17.2 事前に用意した NFS サーバを利用する場合

別途事前に用意した NFS サーバを利用する場合、volume の定義の driver 属性に local を指定してください。また、driver_options 属性で NFS をマウントするために以下の属性を指定してください。

属性名	説明
type	"nfs" を指定してください。
device	":" のあとに続けて nfs の共有パスを指定してください
o	マウントオプションを指定してください。 addr オプションに NFS サーバの IP アドレスあるいは FQDN を指定してください。

以下にその設定例を示します。

```
volumes:
- source: pdnsdb_data
  target: /var/lib/mysql
  type: volume
  driver: local
  driver_options:
    type: nfs
    device: ":/shared"
    o: "addr=192.168.0.10,hard,timeo=600,retrans=3,rsize=1048576,wsize=1048576,resvport,
↪async"
```

17.3 ファイルサーバの自動構築

aws, azure, gcp のいずれかの IaaS プロバイダを使用している場合は、NFS のファイルサーバを IaaS 内に自動的に構築することができます。自動構築を利用する場合、volume の定義に nfs 属性を指定してください。以下にプロバイダごとに利用方法を説明します。

17.3.1 AWS EFS の自動構築

aws プロバイダで volume の定義に nfs 属性が指定されている場合、以下のパラメータで Amazon EFS ファイルシステムが作成されます。

パラメータ	説明
名前	efs-hive 名
暗号化	hive 定義の mirrored_disk_encrypted 属性、kms_key_id 属性に従って暗号化
リージョン	hive 定義の region 属性の値
可用性	リージョンレベル可用性（特定の可用性ゾーンに障害が発生した場合他の可用性ゾーンでサービス提供可能）
Project タグ	hive 名

Amazon EFS ファイルシステムの作成は build-infra フェーズで実行されます。buidl-infra サブコマンドの -D オプションを使用することで Amazon EFS ファイルシステムを削除できます。ファイルシステム内にデータが残っていても強制的に削除されますので、注意してください。

このファイルシステムは 1 台目のコンテナ収容サーバの /mnt/efs_root にマウントされますが、コンテナから利用する場合は、このマウントは利用せず、docker volume として別途マウントされます。

具体的には、volume 定義に nfs 属性が指定されているとファイルシステム内に volume 名でディレクトリが作成され、サービスがデプロイされると volume 定義に従ってコンテナ内にマウントされます。ディレクトリと docker volume の作成は build-volumes フェーズで行われます。build-volumes の -D オプションでディレクトリを削除できます。ディレクトリ内にデータが残っていても強制的に削除されますので、注意してください。

AWS EFS の場合は nfs 属性のオブジェクトに有効な属性はありません。{} で空のオブジェクトを指定してください。以下に例を示します。

```
volumes:
- source: pdnsdb_data
  target: /var/lib/mysql
  type: volume
  nfs: {}
```

AWS EFS では容量に制限はありません。想定外にディスクを使用してしまうのを防ぐために Cloud Watch などで監視することが推奨されます。

AWS EFS の自動構築を使用する場合は、IAM で hive-builder で使用するユーザにアクセス許可に組み込みポリシー AmazonElasticFileSystemFullAccess を付与してください。

17.3.2 Azure Files NFS ファイル共有の自動構築

> 警告: Azure Files NFS ファイル共有の自動構築を利用する場合は hive 名の長さが 3 文字以上 24 文字以下でなければなりません。

azure プロバイダで volume の定義に nfs 属性が指定されている場合、以下のパラメータでストレージアカウントが作成されます。

パラメータ	説明
リソースグループ	hive 名
名前	hive 名
アカウントタイプ	プレミアム
リージョン	hive 定義の region 属性の値
可用性	リージョンレベル可用性（特定の可用性ゾーンに障害が発生した場合他の可用性ゾーンでサービス提供可能）

ストレージアカウントの作成は build-infra フェーズで実行されます。buidl-infra サブコマンドの -D オプションを使用することでストレージアカウントを削除できます。ストレージアカウント内にデータが残っていても強制的に削除されますので、注意してください。

volume 定義に nfs 属性が指定されているとこのサービスアカウントに対して NFS ファイル共有が作成されます。このとき、NFS ファイル共有の名前はボリューム名の'_' を'-' に置換したものになります。NFS ファイル共有と docker volume の作成は build-volumes フェーズで行われます。build-volumes の -D オプションで NFS ファイル共有を削除できます。NFS ファイル共有内にデータが残っていても強制的に削除されますので、注意してください。

サービスがデプロイされると volume 定義に従ってコンテナ内にマウントされます。Azure Files の場合は nfs 属性のオブジェクトで size 属性を指定する必要があります。size 属性の指定方法は drbd と同様ですが、 100G より小さい値を指定した場合、 100G に切り上げられます。以下に例を示します。

```
volumes:
- source: pdnsdb_data
  target: /var/lib/mysql
  type: volume
  nfs:
    size: 120G
```

17.3.3 GCP Cloud Filestore の自動構築

gcp プロバイダで volume の定義に nfs 属性が指定されている場合、以下のパラメータで Cloud Filestore インスタンスが作成されます。

パラメータ	説明
名前	filestore-hive 名
リージョン	hive 定義の region 属性の値
可用性	リージョンレベル可用性（特定の可用性ゾーンに障害が発生した場合他の可用性ゾーンでサービス提供可能）
ディスク	SSD
プロジェクト	gcp_credential.json で指定されたプロジェクト

Cloud Filestore インスタンスの作成は build-infra フェーズで実行されます。buidl-infra サブコマンドの -D オプションを使用することで Cloud Filestore インスタンスを削除できます。Cloud Filestore インスタンス内にデータが残っていても強制的に削除されますので、注意してください。

この Cloud Filestore インスタンスは 1 台目のコンテナ収容サーバの /mnt/filestore_root にマウントされますが、コンテナから利用する場合は、このマウントは利用せず、docker volume として別途マウントされます。

具体的には、volume 定義に nfs 属性が指定されているとファイルシステム内に volume 名でディレクトリが作成され、サービスがデプロイされると volume 定義に従ってコンテナ内にマウントされます。ディレクトリと docker volume の作成は build-volumes フェーズで行われます。build-volumes の -D オプションでディレクトリを削除できます。ディレクトリ内にデータが残っていても強制的に削除されますので、注意してください。

GCP Cloud Filestore の場合は nfs 属性のオブジェクトに有効な属性はありません。{} で空のオブジェクトを指定してください。以下に例を示します。

```
volumes:
- source: pdnsdb_data
  target: /var/lib/mysql
  type: volume
  nfs: {}
```

inventory/group_vars/all.yml に以下のパラメータを設定することで Cloud Filestore の容量とマウントオプションの設定が可能です。

パラメータ	選択肢/例	デフォルト	意味
hive_gcp_capacity_gb	2048	1024(1T バイト)	loud Filestore の合計の最大容量（GB 単位で記載）
hive_gcp_nfs_options	"hard,timeo=600,retrans=3,nsize=1048576,nfs=3,rsize=1048576,resvport,async,nolock"	"hard,timeo=600,retrans=3,nsize=1048576,nfs=3,rsize=1048576,resvport,async,nolock"	"hard,timeo=600,retrans=3,nsize=1048576,nfs=3,rsize=1048576,resvport,async,nolock"

GCP Cloud Filestore の自動構築を使用する場合は、サービスアカウントのロールに「Cloud Filestore 編集者」の権限を与えてください。

また、EnterpriseStorageGibPerRegion を以下の手順で割り当ててください。

1. IAM の割当で EnterpriseStorageGibPerRegion で検索し、自分のリージョンにチェックして上の割当を編集をクリック
2. 利用する容量（単位 GB）を割り当てて、「次へ」をクリック
3. 開いた連絡先を確認後「送信」をクリック

割当の変更には営業日で 2 日程度かかる場合があります。

> 警告：GCP Cloud Filestore の自動構築では、build-infra フェーズで 20 分から 30 分の時間がかかる場合がありますので注意してください。

17.4 DRBD との比較

DRBD を使用せずに NFS を使用するメリットは以下のとおりです。

- データの複製を行わないので、ディスク使用量を少なくすることができる
- NFS のサーバが DRBD よりも高い可用性・保全性を提供している場合にそれを利用できる

逆にデメリットについては以下のとおりです。

- NFS サーバの使用量が追加コストとなる
- aws, azure,gcp 以外のプロバイダでは自動構築が利用できない
- aws, gcp ではボリュームごとのディスク使用量に上限を指定できない

第 18 章

証明書について

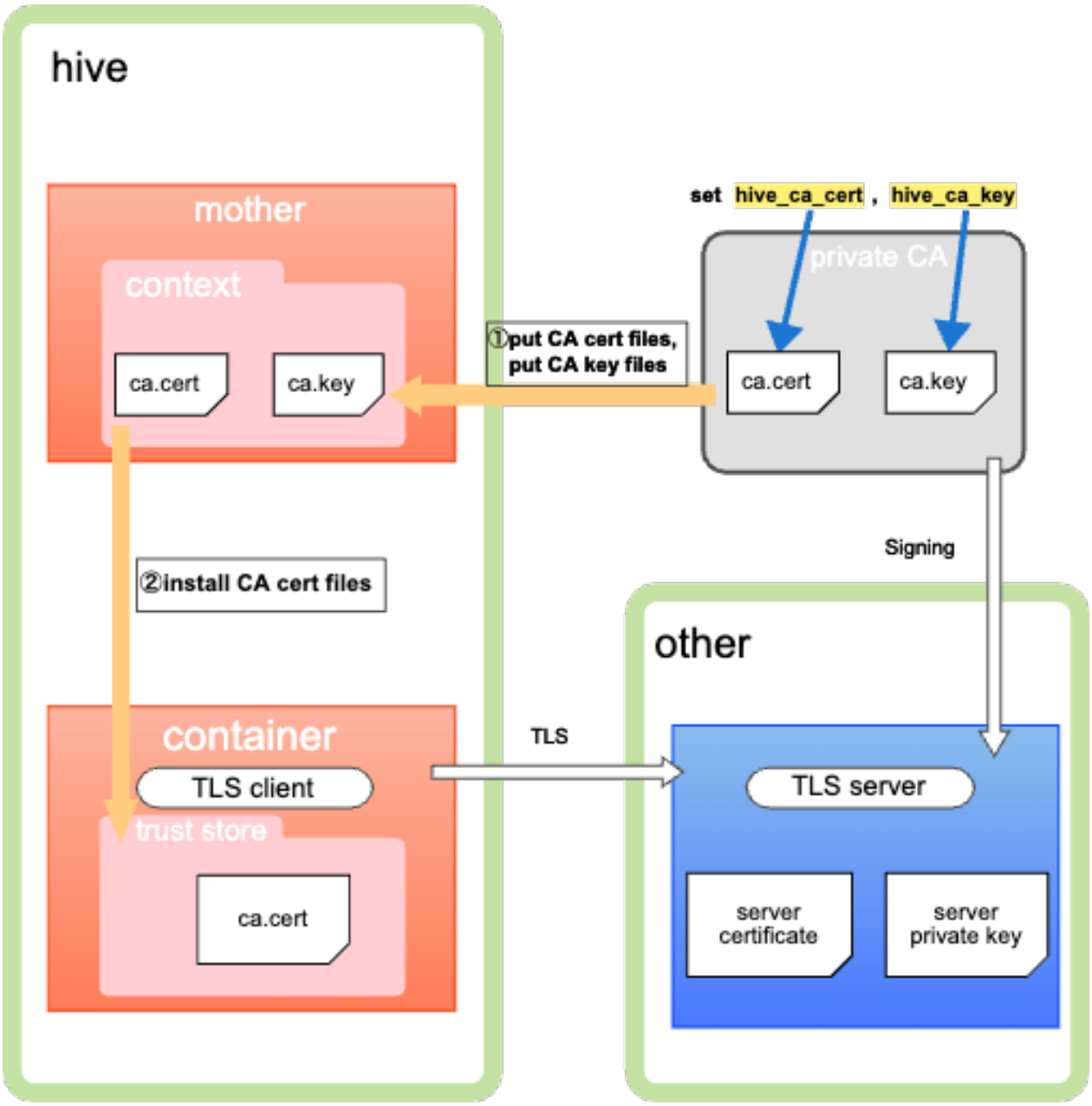
hive-builder では必要に応じて CA 局証明書の共有および証明書の生成が可能となっています。
サーバー間で CA 局証明書を共有した上での運用例を解説していきます。

18.1 A. コンテナをクライアントとして運用したい場合

コンテナをクライアントとして運用する場合、
「1.CA 局証明書の信頼」および「2. コンテナへのクライアント証明書の付与」が可能です。

18.1.1 A-1.CA 局証明書の信頼

hive の機能で共有したい CA 局証明書をコンテナのトラストストアに配置して信頼するため、通信相手のサーバー証明書を検証することが可能です。

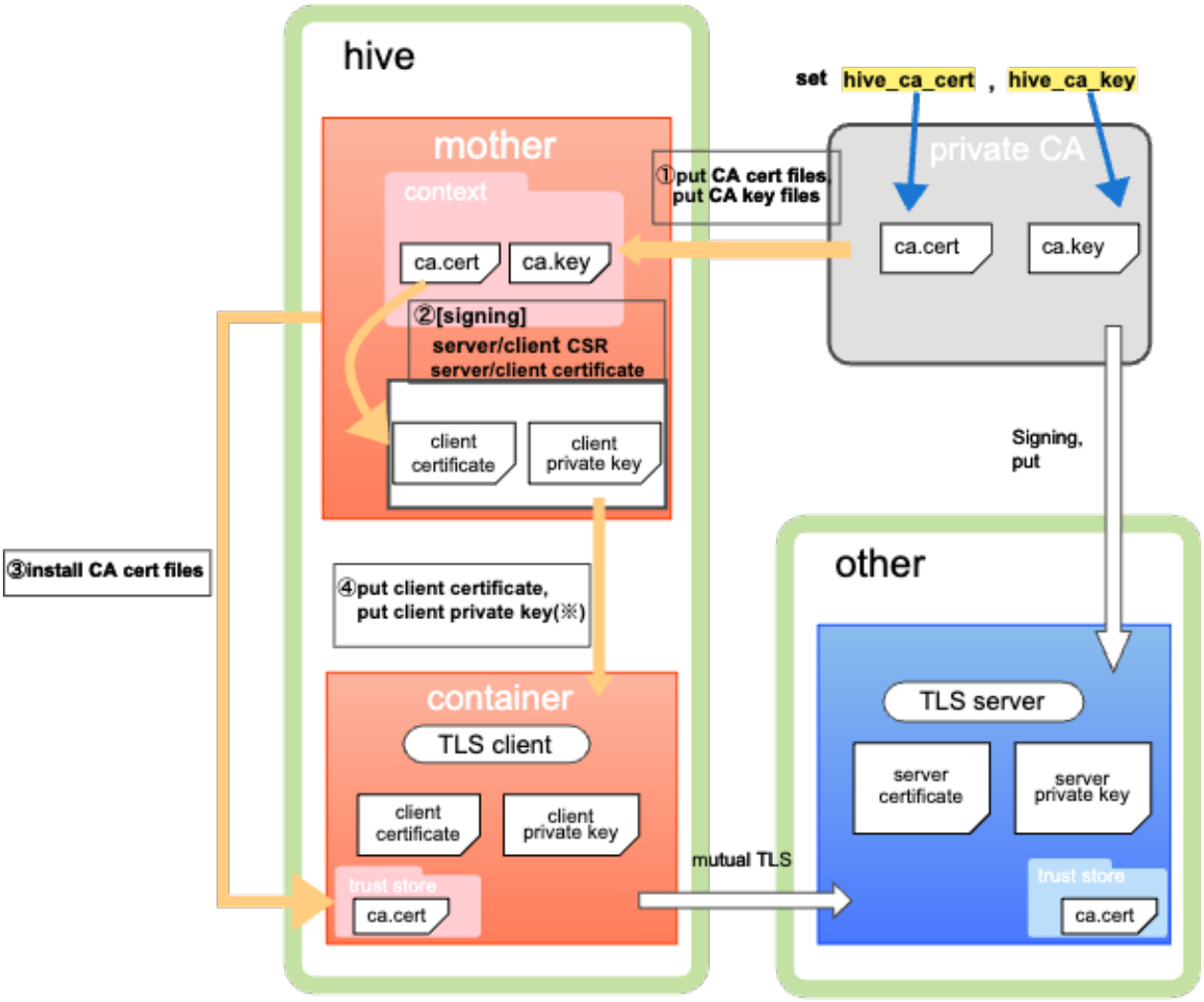


âŠăCA 局の証明書と秘密鍵を mother マシンのコンテキストディレクトリに配置します。証明書と鍵の配置方法は **「*CA* 局証明書の共有」** の節を参照してください。

âŠą 共有した CA 局の証明書をコンテナのトラストストアにインストールします。詳しくは **「ルート証明書信頼設定ビルトインロール」** の節を参照してください。

18.1.2 A-2. コンテナへのクライアント証明書の付与

「1.CA 局証明書の信頼」で共有した CA 局証明書を用いて生成したクライアントの証明書と秘密鍵をコンテナに付与することで、通信相手との TLS 相互認証が可能となります。



âŠăCA 局の証明書と秘密鍵を mother マシンのコンテキストディレクトリに配置します。証明書と鍵の配置方法は「*CA* 局証明書の共有」 の節を参照してください。

âŠą 共有した CA 局証明書を用いて、mother マシンでクライアント証明書およびクライアントの秘密鍵を生成します。

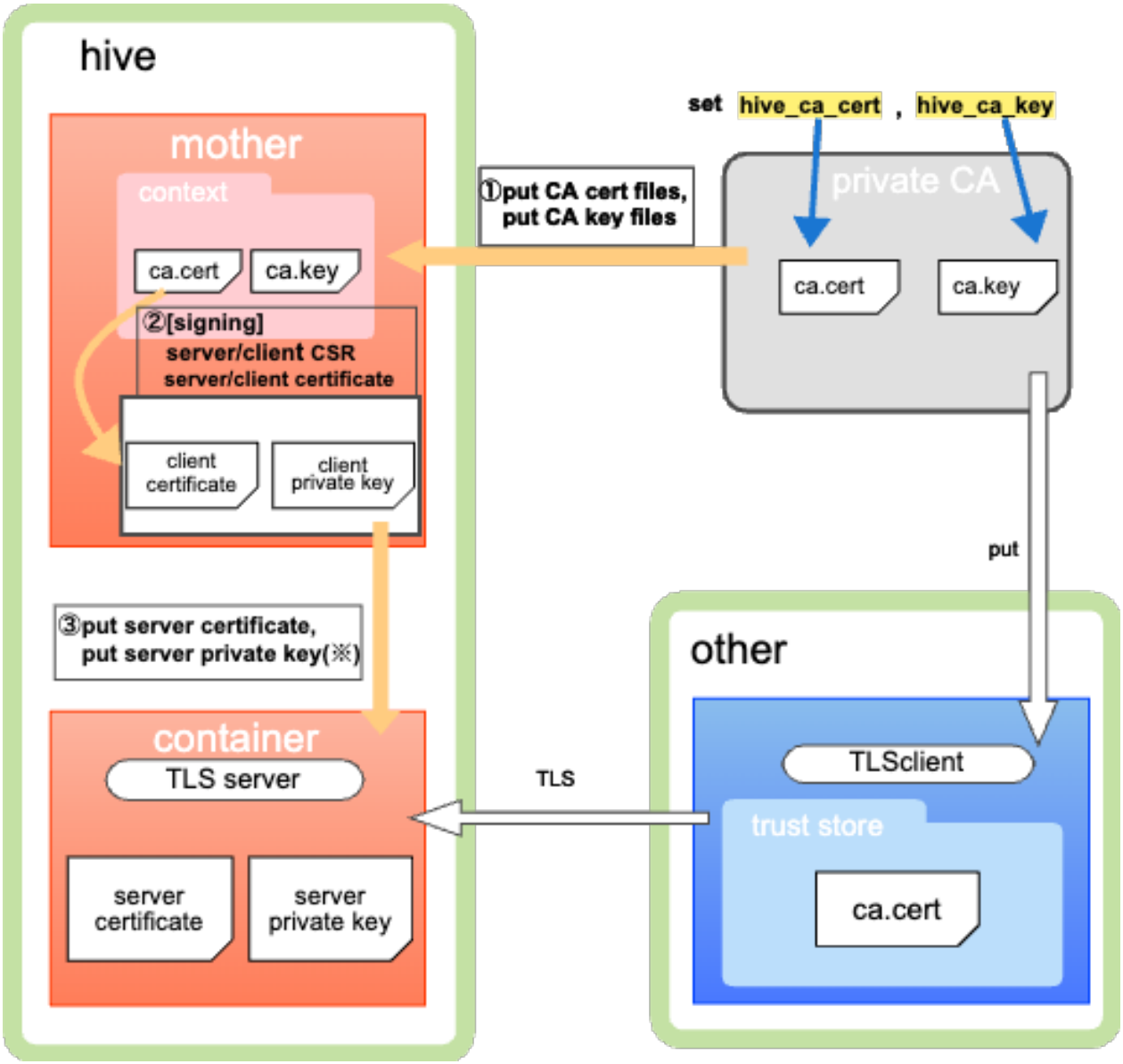
âŚć 共有した CA 局の証明書をコンテナのトラストストアにインストールします。詳しくは **「ルート証明書信頼設定ビルトインロール」** の節を参照してください。

âŚć 生成したクライアントの証明書と秘密鍵をコンテナに配置します。

クライアント証明書の生成の際に指定するドメイン名、サフィックス、有効期限の設定およびコンテナへの証明書の配置については **「証明書生成ビルトインロール」** の節を参照してください。

18.2 B. コンテナをサーバーとして運用したい場合

コンテナをサーバーとして運用する場合、コンテナへのサーバー証明書の付与が可能です。



âŠăCA 局の証明書と秘密鍵を mother マシンのコンテキストディレクトリに配置します。証明書と鍵の配置方法は **「*CA* 局証明書の共有」** の節を参照してください。

âŠą 共有した CA 局証明書を用いて、mother マシンでサーバー証明書およびサーバーの秘密鍵を生成します。

âŠć 生成したサーバー証明書と秘密鍵をコンテナに配置します。

サーバー証明書の生成の際に指定するドメイン名、サフィックス、有効期限の設定およびコンテナへの証明書の配置については **「証明書生成ビルトインロール」** の節を参照してください。

18.3 CA 局証明書の共有

hive_builder の機能で CA 局の証明書を共有することが可能です。(mother マシンへの配置) 下記の形式でインベントリ (例えば、inventory/group_vars/all.yml) に変数 hive_ca_cert にサーバー証明書を、変数 hive_ca_key にサーバ鍵を定義してください。以下に例を示します。

```
hive_ca_cert: |
  -----BEGIN CERTIFICATE-----
  MIIE6TCCAtECFFq7Q+zMjH+HbQILdIJV+dWM7vIeMA0GCSqGSIb3DQEBCwUAMDAx
  GDAWBgoJkiaJk/IsZAEZFghob2dlaGl2ZTEUMBIGA1UEAwwLY2EuaG9nZWhpdmUw
                                  .
                                  .
  /8RdE53g5XuXaHna5w==
  -----END CERTIFICATE-----

hive_ca_key: |
  -----BEGIN PRIVATE KEY-----
  MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQC5LaqGi+VrKEt/
  avMBKhnKhJ8Fuo37Zr/bNETEtPTfSnJ4xxVkNaCzksgLTNjPu3iF+rCw3QPUA4Bg
                                  .
                                  .
  K5hfEuwyPeeCaBuJua19DO/fl87L5pU=
  -----END PRIVATE KEY-----
```

CA 局の証明書と秘密鍵の共有は build-infra フェーズで実行されます。

(hive_ca_cert, hive_ca_key が定義されていない場合は、同様のフェーズで新たに CA 局証明書と秘密鍵が生成されます。)

CA 局証明書の共有機能を利用する場合は、必ず正しいペアの証明書と鍵の両方定義するようにしてください。

また、証明書、鍵の内容が全行インデントされていないと正常に動作しないため、ご注意ください。

18.4 証明書生成ビルトインロール

hive_builder のビルトインロール hive-certificate でアプリケーションのサーバに利用できるクライアント (サーバー) 証明書を生成することが可能です。

下記の形式でインベントリ (例えば、inventory/group_vars/all.yml) に変数 certificate_fqdn, sub_prefix, ca_valid_in を定義することで指定のドメイン、サフィックス、有効期限で証明書が生成されます。

また、証明書と秘密鍵をコピーしたいコンテナのパスは変数 cert_dest, key_dest で指定してください。

以下に例を示します。

```
certificates:
  - certificate_fqdn: "dnsdist-example-slave.test.procube-demo.jp"
    ca_valid_in: "{{ 365 * 100 }}"
    sub_prefix: /DC={{ hive_name.split('.') | reverse | join('/DC=') }}
    cert_dest: /etc/pki/tls/certs/dnsdist-example-slave.test.procube-demo.jp.crt
    key_dest: /etc/pki/tls/private/dnsdist-example-slave.test.procube-demo.jp.key
  - certificate_fqdn: "ddex.test.procube-demo.jp"
    ca_valid_in: "{{ 365 * 100 }}"
    sub_prefix: /DC={{ hive_name.split('.') | reverse | join('/DC=') }}
    cert_dest: /etc/pki/tls/certs/ddex.test.procube-demo.jp.crt
    key_dest: /etc/pki/tls/private/ddex.test.procube-demo.jp.key
```

証明書の生成は `build-images` **フェーズで実行されます。**

上記の例で作成される証明書は、

1 枚目:

CN=dnsdist-example-slave.test.procube-demo.jp, DC=${ hive_name }, 有効期限=100 年

2 枚目:

CN=ddex.test.procube-demo.jp, DC=${ hive_name }, 有効期限=10 年

となります。

sub_prefix については値のみを定義していただくことで指定の値を DC に設定することができます。(DC を'boo' にしたい時は、sub_prefix: /DC=boo とすることで設定可能)

ビルトインロール hive-certificate を適用するには、サービスの image.roles 属性の下に追加する必要があります。
以下に例を示します。

```
hoge:
  image:
    from: alpine:edge
    roles:
      - python-aptk
      - hive-certificate
```

18.5 ルート証明書信頼設定ビルトインロール

hive_builder のビルトインロール hive-trust-ca では、CA 局証明書のコンテナのトラストストアにインストールします。ビルトインロール hive-trust-ca を適用するには、サービスの image.roles 属性の下に追加する必要があります。
以下に例を示します。

```
hoge:
  image:
    from: alpine:edge
    roles:
      - python-aptk
      - hive-certificate
      - hive-trust-ca
```

18.6 OS ごとのデフォルトトラストストア確認コマンド

alpine 系、ubuntu 系、centos 系それぞれの OS で hive_builder を用いて環境を構築した際のデフォルトトラストストアを確認する方法を示します。
共有した CA 局の証明書、証明書生成ビルトインロールで生成されたクライアント (サーバー) の証明書と秘密鍵は、以下のコマンドを実行することで確認することが可能です。

alpine 系

```
ls /etc/pki/ca-trust/source/anchors/
```

ubuntu 系

```
ls /etc/ssl/certs/
```

centos 系

```
ls /etc/ssl/certs/
```

第 19 章

障害からの復旧方法

19.1 サーバのディスク残量

zabbix のトリガーで、ディスク残量が少なくなり、 "Problem: /: Disk space is low (used > 80%)" などの警告メッセージが上がる場合があります。このような場合、以下の方法で空き領域を確保してください。

19.1.1 docker の不要なイメージ

サーバで docker の不要なイメージやコンテナが残存してディスクを圧迫している場合があります。この場合、対象サーバに hive ssh コマンドでログインし、以下のコマンドを実行して使用していないイメージやコンテナを回収してください。

```
docker system prune
```

19.1.2 ログファイルの肥大化（リポジトリサーバ）

リポジトリサーバでディスク容量が不足した場合、 /var/log/services の下のログファイルのサイズを確認してください。特定のサービスのログファイルが極端に大きくなっている場合、その内容を確認し、デバッグログの抑制などの対策を実施してください。

19.1.3 リポジトリの肥大化（リポジトリサーバ）

デバッグ環境などで build-images の回数が多くなりますと、過去のイメージがリポジトリサーバ内に蓄積され、リポジトリサーバのディスクを消費します。hive ssh でリポジトリサーバにログインし、以下のコマンドでリポジトリのディスク使用量をチェックしてください。

```
sudo du -s -B G /var/lib/docker/volumes/registry_regdata
```

このディスク使用量がサーバのディスク残量不足の原因である場合は、以下のコマンドでリポジトリを一旦削除にした後、イメージをビルドし直してください。（本番運用環境では利用中のイメージが失われるため、おすすめできません）

```
hive ssh
cd registry
docker-compose down -v
docker-compose up -d
logout
hive build-images
```

19.2 コンテナ内のディスク消費量の増加

コンテナがディスクに書き込むとディスクが消費され、これによりコンテナ収容サーバのディスク残量や DRBD ボリュームのディスク残量が逼迫する場合があります。以下に対象となるボリュームごとに対応を示します。

19.2.1 DRBD ボリュームのディスク残量

コンテナがマウントするボリュームのうち drbd 属性が付与されたものは DRBD によるボリュームとなり、サーバのディスクとは独立したものとなります。このボリュームの残量が少なくなった場合、zabbix のトリガーで "Problem: Exceed 90% Docker volume ボリューム名 in サービス名 usage" のような警告メッセージが上がります。このボリュームのディスクの拡大が必要で DRBD 用の物理ボリュームに余裕がある場合、以下の手順でボリュームを拡張できます。

1. バックアップ採取

対象ボリューム内のデータのバックアップを採取してください。hive のサービス定義の backup_scripts に対象ボリュームの内容のバックアップスクリプトが指定されている場合は、以下のコマンドでバックアップを採取できます。

```
hive ssh
hive-backup.sh -l サービス名
logout
```

注釈: 上記はバックアップスクリプトが指定されている場合の方法であり、設定されていない場合は dsh, dcp などを使用してバックアップを採取する必要があります。

注釈: 内部のデータを新しいボリュームに引き継ぐ必要がない場合はこの手順はスキップできます。

---

2. ボリュームの削除

一旦、ボリュームを削除してください。ボリュームを削除するためにサービスも削除する必要があります。

```
hive deploy-services -l サービス名 -D
hive build-images -l ボリューム名 -D
```

3 ボリュームのサイズを修正

サービス定義内に定義されているボリューム定義の drbd 属性の size 属性を修正し、必要なボリュームのサイズを指定してください。

4. ボリュームのビルド

ボリュームをビルドしてください。ビルド後サービスをデプロイしてください。

```
hive deploy-services -l サービス名
hive build-images -l ボリューム名
```

5. データのリストア

手順 1. で採取したバックアップデータをリストアしてください。hive のサービス定義の backup_scripts に対象ボリュームの内容のバックアップスクリプトが指定されている場合は、以下のコマンドでリストアできます。

```
hive ssh
hive-backup.sh -l サービス名 -r
logout
```

19.2.2 コンテナ収容サーバのディスク残量

コンテナのディスクの一時的な領域に大量にデータが書き込まれることにより、コンテナ収容サーバのディスクが逼迫する場合があります。このような場合、以下の手順でデータを消費しているサービスを特定してください。

1. Overlay ディレクトリの特定

ディスク容量が逼迫しているホストで以下のコマンドを実行し、どの Overlay ディレクトリがディスクを消費しているか特定してください。

```
hive ssh -t ホスト名
sudo su -
du -s -BM /var/lib/docker/overlay2/* | sort -nr | head -5
```

出力例：

```
13457M      /var/lib/docker/overlay2/
↪50109e612bd497c812ecffcedcfe890eadf69033c133a1e33b56962781c5080b
1639M       /var/lib/docker/overlay2/
↪4b280aa02d57f2cd2adf6bd1bd88b7917f253032b7bdffcebe4cf451e3d958e0
1363M       /var/lib/docker/overlay2/
↪947092c7f5914fd2b9341003d571045649a2d201005b8f024ece71a294760c5a
1363M       /var/lib/docker/overlay2/
↪17ec482a80844f10cea6e6f1257a055ae596634eb0bcb2993378395f368f291c
1109M       /var/lib/docker/overlay2/
↪0e8e71e842aed54fbce7fa711508d67eca1b627ebac5f9aacbad0184728dd18c
```

2. サービスの特定

上記で表示されたディレクトリの配下の merged ディレクトリに移ると、サービスの '/' ディレクトリが見えます。その配下のファイルを調べることでどのサービスのディレクトリであるかを特定してください。例えば、前項の例のトップの Overlay ディレクトリを調べる場合は以下のようなコマンドを実行してください。

```
cd /var/lib/docker/overlay2/
↪50109e612bd497c812ecffcedcfe890eadf69033c133a1e33b56962781c5080b/merged
ls
ls etc
ls var/log
```

サービスが特定できたら logout を 1 回実行して root ユーザのセッションを抜けてください。

3. ディスク領域の回収

前項で特定したサービスに dsh でログインし、ディスク領域の回収操作を実施してください。回収操作の終了後、ホストをログアウトしてください。

19.3 ホスト内サービスの失敗

ホストの中でサービスの実行に失敗し、zabbix から "Problem: At least one of the services is in a failed state" の警告メッセージが上がる場合があります。

この場合、以下のコマンドで失敗しているサービスを特定しその原因を追求して対策を講じてください。

```
hive ssh -t ホスト名
systemctl list-units --type=service --no-pager --no-legend --state=failed --all
logout
```

この失敗しているサービスがシステムとして不要な場合、このサービスを停止してもいいかもしれません。例えば、dnf-makecache.timer や getty@tty1.service を停止する場合、以下のコマンドで停止してください。

```
hive ssh -t ホスト名
sudo systemctl disable --now dnf-makecache.timer getty@tty1.service
logout
```

19.4 DRBD ディスクの同期の失敗

DRBD ディスクの同期に失敗し、zabbix から "Problem: DRBD resource ボリューム名 status is not UpTo-Date/Diskless client" の警告メッセージが上がる場合があります。この場合は、まず以下のコマンドでサービスを停止してください。

```
hive deploy-services -l サービス名 -D
```

その後、リポジトリサーバを除く各ホストで以下のコマンドを実行し、DRBD の状態を確認します。

```
hive ssh -t ホスト名
sudo drbdadm status ボリューム名
logout
```

以下に対象方法についてパターンごとに示します。

19.4.1 全部のホストで Outdated

すべてのホストで以下のように表示され、 Outdated になっている場合があります。

```
$ drbdadm status ボリューム名
ボリューム名 role:Secondary
  disk:Outdated
  hive1.hive 名 role:Secondary
    peer-disk:Outdated
  hive2.hive 名 role:Secondary
    peer-disk:Outdated
```

この場合は、以下のコマンドで 1 号機のディスクを強制的に Primary に昇格してください。

```
hive ssh -t 1号機
sudo drbdadm primary --force ボリューム名
logout
```

その後、他のホストで再接続を実行してください。

```
hive ssh -t ホスト名
sudo drbdadm disconnect ボリューム名
sudo drbdadm connect ボリューム名
logout
```

その後、１号機のディスクを secondary に降格してください。

```
hive ssh -t 1号機
sudo drbdadm secondary ボリューム名
logout
```

19.4.2 接続が不完全

DRBD のステータスで片側から見るとエラーにはなっていないのに、反対側から見るとエラーに見える場合があります。例えば、

１号機の結果（正常）

```
$ drbdadm status ボリューム名
ボリューム名 role:Secondary
  disk:UpToDate
  hive1.hive 名 role:Primary
    peer-disk:UpToDate
  hive2.hive 名 role:Secondary
    peer-disk:UpToDate
```

２号機の結果（同期途中）

```
$ drbdadm status ボリューム名
ボリューム名 role:Primary
  disk:UpToDate
  hive0.hive 名 role:Secondary
    replication:WFBitMapS peer-disk:Consistent
  hive2.hive 名 role:Secondary
    peer-disk:UpToDate
```

この場合、正常となっている方のホスト（上記の場合では１号機）で以下のコマンドを実行して再接続を行ってください。

```
hive ssh -t ホスト名
sudo drbdadm disconnect ボリューム名
sudo drbdadm connect ボリューム名
logout
```

19.5 サービスが再起動を繰り返す

障害からの復旧後、サービスが起動できず、再起動を繰り返す場合があります。このような場合、docker service ps コマンドに --no-trunc オプションを付与してそのエラーの原因を見てください。例えば、以下のように表示されます。

```
docker service ps --no-trunc ldap
ID                          NAME                IMAGE                                     NODE
              DESIRED STATE       CURRENT STATE             ERROR
                                                                                          PORTS
swcu3n4b70urjtwhdzf92jgh9   ldap.1              s-hive2.op:5000/image_ldap:latest   s-
↪hive1              Ready               Rejected 3 seconds ago    "failed to mount local␣
↪volume: mount /dev/drbd8:/var/lib/docker/volumes/ldap_data/_data: invalid argument"
vc210ej598da9yuc9l1iw3e0i    \_ ldap.1          s-hive2.op:5000/image_ldap:latest   s-
↪hive2              Shutdown            Rejected 9 seconds ago    "failed to mount local␣
↪volume: mount /dev/drbd8:/var/lib/docker/volumes/ldap_data/_data: invalid argument"
dk8sxitj3v7uyjhmkh57cneoz    \_ ldap.1          s-hive2.op:5000/image_ldap:latest   s-
↪hive0              Shutdown            Rejected 14 seconds ago   "failed to mount local␣
↪volume: mount /dev/drbd8:/var/lib/docker/volumes/ldap_data/_data: invalid argument"
xo6s6q13z2ipiok5459fnvhuy    \_ ldap.1          s-hive2.op:5000/image_ldap:latest   s-
↪hive1              Shutdown            Rejected 19 seconds ago   "failed to mount local␣
↪volume: mount /dev/drbd8:/var/lib/docker/volumes/ldap_data/_data: invalid argument"
bues7ycsvddiohserzage69lx    \_ ldap.1          s-hive2.op:5000/image_ldap:latest   s-
↪hive2              Shutdown            Rejected 23 seconds ago   "failed to mount local␣
↪volume: mount /dev/drbd8:/var/lib/docker/volumes/ldap_data/_data: invalid argument"
```

ここでは、エラーメッセージは "failed to mount local volume: mount /dev/drbd8:/var/lib/docker/volumes/ldap_data/_data: invalid argument" となっています。

19.5.1 DRBD のファイルシステムの破損

前項エラーメッセージ例の "invalid argument" はファイルシステムが破損している場合に出るメッセージです。以下のコマンドで修復してください。

```
hive ssh -t １号機
docker service scale サービス名=0
sudo xfs_repair $(docker volume inspect ボリューム名 --format '{{ .Options.device }}')
docker service scale サービス名=1
logout
```

19.6 サーバーが一台破損した場合

サーバーが一台完全に破損し、インスタンス一覧にも表示されない場合があります。この場合は、まず hive ssh で残っているサーバーに接続し hive node ls で swarm クラスタで STATUS の部分が Down となっているコンテナがあることを確認してください。その後 Down しているコンテナの ID をコピーして remove してください。例えば、s-hive0 が破損しており、かつ s-hive1 が残っている場合以下のコマンドを実行してください。

```
hive ssh -t s-hive1.pdns
docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ␣
↪ENGINE VERSION
sn3njpuc7dnevr8c6b8zqv1c7     s-hive0    Down      Active         Unreachable      20.10.
↪10
kcvmtp7dhq7513v0zihgw5cgl *   s-hive1    Ready     Active         Reachable        20.10.
↪10
8yzfk4x1bi5dr6kg03s67fodn     s-hive2    Ready     Active         Leader           20.10.
↪10
docker node demote sn3njpuc7dnevr8c6b8zqv1c7
docker node rm sn3njpuc7dnevr8c6b8zqv1c7
logout
```

変数の no_format に true、first_hive に生きている hive の数字を設定した後、フェーズとして build-infra setup-hosts build-volumes の順に実行してください。

```
hive set no_format true
hive set first_hive 1
hive build-infra
hive setup-hosts
hive build-volumes
```

第 20 章

よくある質問

20.1 build-images で Bad local forwarding specification のエラーになります

hive-builder は OpenSSH の unix domain socket のフォワーディング機能を使用していますが、OpenSSH-6.7 より古いバージョンの OpenSSH （例えば、 CentOS 7.2 では、OpenSSH-6.6）では、この機能をサポートしていません。OpenSSH-6.7 以上にバージョンアップしてご利用ください。

20.2 リポジトリサーバのログ収集で fluentd を使用しないのはなぜですか

docker の fluentd ロギングドライバーは fluentd と TCP 接続できないとサービスが起動しません。このため、リポジトリサーバに配置した fulentd が死んでいる場合はサービスが起動できません。hive-builder の要件としてリポジトリサーバが死んでいてもサービス提供を続けることというのがあり、採用できませんでした。

20.3 ビルドが止まってしまいます

ビルドで、ansible のタスク実行が次に進まなくなる場合があります。原因がわかっておりません。対象のサーバにログインして、プロセスの実行状況をみて、タスクを実行している様子がなければ、 ^C キーで hive コマンドを中断し、再実行してください。

20.4 build-images, initialize-services で fail to create socket のエラーになります

メッセージ

fail to create socket /var/tmp/hive/docker.sock@サーバ名, another hive process may doing build-image or the file has been left because previus hive process aborted suddenly

コマンド

build-images, initialize-services

対応方法

他の hive コマンドが同じマザーマシンで動作している場合はその終了を待ってください。そうでない場合は rm コマンドで /var/tmp/hive/docker.sock@サーバ名を削除してください。

20.5 initialize-services で Authentication or permission failure のエラーになります

メッセージ

Authentication or permission failure. In some cases, you may have been able to authenticate and did not have permissions on the target directory. Consider changing the remote tmp path in ansible.cfg to a path rooted in "/tmp".

コマンド

initialize-services

原因

initialize-services の実行中にサービスの再起動が行われた可能性があります。

対応方法

ログなどを確認して、initialize-services 実行中にサービスが再起動しないように修正してください。

20.6 build-infra で Vagrant command failed のエラーになります

メッセージ

Vagrant command failed: Command "["/usr/bin/vagrant", "up", "--provision"]" returned non-zero exit status 1

コマンド

build-infra

対応方法

cd .hive/ステージ名; /usr/bin/vagrant up --provision を実行してエラーメッセージを確認し、修正してください。

20.6.1 エラーメッセージに Could not create the directory が含まれる場合

エラーメッセージが以下のようなもので、 Could not create the directory が含まれている場合、 vagrant-disksize プラグインのバグにより、ディスクのサイズ拡張に失敗しています。参考： https://github.com/sprotheroe/vagrant-disksize/pull/27

```
There was an error while executing `VBoxManage`, a CLI used by Vagrant
for controlling VirtualBox. The command and stderr is shown below.
Command: ["clonemedium", "C:\\Users\\mitsuru\\VirtualBox VMs\\p-hive0.mic-env\\CentOS-8-
↪Vagrant-8.0.1905-1.x86_64.vmdk", "./C:\\Users\\mitsuru\\VirtualBox VMs\\p-hive0.mic-
↪env\\CentOS-8-Vagrant-8.0.1905-1.x86_64.vdi", "--format", "VDI"]
Stderr: 0%...
Progress state: VBOX_E_IPRT_ERROR
VBoxManage.exe: error: Failed to clone medium
VBoxManage.exe: error: Could not create the directory '\\wsl$\Ubuntu\home\mitsuru\hive\
↪private\C:\Users\mitsuru\VirtualBox VMs\p-hive0.mic-env' (VERR_INVALID_NAME)
VBoxManage.exe: error: Details: code VBOX_E_IPRT_ERROR (0x80bb0005), component␣
↪VirtualBoxWrap, interface IVirtualBox
VBoxManage.exe: error: Context: "enum RTEXITCODE __cdecl handleCloneMedium(struct␣
↪HandlerArg *)" at line 1071 of file VBoxManageDisk.cpp
```

この場合、vagrant-disksize プラグインを修正することで回避できます。

vagrant-disksize プラグインを以下のように修正してください。

ファイル

~/.vagrant.d/gems/2.6.6/gems/vagrant-disksize-0.1.3/lib/vagrant/disksize/actions.rb

修正箇所

151 行目

修正前

```
dst = File.join(src_path, src_base) + '.vdi'
```

修正後

```
dst = src_base + '.vdi'
```

20.6.2 エラーメッセージに Error: Unknown repo: 'C*-base' が含まれる場合

エラーが VirtualBox Guest addtions のインストール中に発生し、メッセージが Error: Unknown repo: 'C*-base' である場合、vagrant-vbguest プラグインのバグである可能性が高いです。参考： https://github.com/dotless-de/vagrant-vbguest/issues/367

この場合は、vagrant-vbguest プラグインを vagrant plugin uninstall vagrant-vbguest コマンドでアンインストールしてください。

20.7 build-images で Release file is not valid yet のエラーが出ます

メッセージ

Release file for http://security.ubuntu.com/ubuntu/dists/focal-security/InRelease is not valid yet (invalid for another XXh XXmin XXs). Updates for this repository will not be applied.

コマンド

build-images

発生条件

Vagrant プロバイダでパソコン内の VirtualBox 上に hive を構築している場合で、仮想マシンが起動している状態でパソコンをスリープ状態から復帰させた場合

原因

サーバの時刻がずれているため、apt のリポジトリの正当性の検証に失敗しています。

対応方法 各

サーバで systemctl restart chroyd を実行してください。

20.8 zabbix で Detect SELinux alert の障害（problem）が残ったままになります。

現象

Zabbix でアイテム SELinux alert の値が 0 になっているにも関わらず、トリガー Detect SELinux alert による障害（problem）が残ったままになります。

発生条件

SELinux alert の値が一度でも 1 になった場合（構築時には起こりやすいです）

原因 こ

のトリガーにはクローズの条件が指定されておらず、手動でクローズして頂く必要があります。

対応方法 構

築時の SELinux alert は無視していただいてかまいません。構築時以外でアラートが上がった場合は、SELinux のログで問題がないか確認してください。障害をクローズするために障害を開き、「確認済」のリンクを開き、メッセージを入力し、「障害確認」と「障害のクローズ」をチェックしてください。

20.9 zabbix の SELinux alert でエラーが出ます

メッセージ

Corrupted checkpoint file. Inode match, but newer complete event (XXX:YYY) found before loaded checkpoint XXXX:YYY

zabbix item

SELinux alert

発生条件

SELinux の audit log が短時間に大量に出力された場合

原因

SELinux の audit log が短時間に大量に出力されたために、 /var/log/audit/audit.log がローテートしてしまい、チェックポイント機能が利用できなかった

対応方法

対象サーバにログインして sudo ausearch -m AVC,USER_AVC,SELINUX_ERR,USER_SELINUX_ERR -i を実行し、 SELinux の audit ログが出力された原因を取り除いてください。その後、 sudo rm /var/run/zabbix/ausearch でチェックポイントファイルを削除してください。

20.10 deploy-services で renaming services is not supported のエラーが出ます

メッセージ

```
An exception occurred during task execution. To see the full traceback, use -vvv. The␣
↪error was: docker.errors.APIError: 501 Server Error: Not Implemented ("rpc error: code␣
↪= Unimplemented desc = renaming services is not supported")
failed: [s-hive0.hive 名] (item=サービス名) => changed=false
  ansible_loop_var: item
  item: サービス名
msg: 'An unexpected docker error occurred: 501 Server Error: Not Implemented ("rpc␣
↪error: code = Unimplemented desc = renaming services is not supported")'
```

発生条件

不明

原因

不明

対応方法

hive ssh -t ステージプリフィクス hive0.hive 名 でログインして、 docker service rm サービス名 を実行後に hive deploy-services を再実行してください。

20.11 build-volumes で modprobe: ERROR: could not insert 'drbd': Required key not available のエラーが出ます

メッセージ

```
modprobe: ERROR: could not insert 'drbd': Required key not available
Failed to modprobe drbd (No such file or directory)
Command 'drbdsetup new-resource kea_config 2 --quorum=majority --on-no-quorum=io-error'␣
↪terminated with exit code 20
```

原因

カーネルの機能で UEFI Secure boot が有効になっているため、署名されていない DRBD のカーネルモジュールは読み込むことができません

対応方法

物理サーバの場合は起動時の UEFI の設定画面で、VMWare などの仮想サーバの場合は Vsphere client などの設定ツールで、サーバの UEFI Secure Boot を無効にしてください。参考：https://docs.vmware.com/jp/VMware-vSphere/6.5/com.vmware.vsphere.vm_admin.doc/GUID-898217D4-689D-4EB5-866C-888353FE241C.html

20.12 mother 環境構築直後の build-infra フェーズで Unexpected failure during module execution. のエラーが出ます

メッセージ

```
TASK [Gathering Facts]␣
↪*********************************************************************************************************
An exception occurred during task execution. To see the full traceback, use -vvv. The␣
↪error was: TypeError: can only concatenate str (not "NoneType") to str
fatal: [p-mother.op]: FAILED! =>
  msg: Unexpected failure during module execution.
  stdout: ''
```

原因

python コマンドがインストールされていない。例えば、Ubuntu などで python2 系もインストールされていない状態で python3 コマンドをインストールし、 pip install hive-builder で mother 環境を構築した場合、 python3 コマンドしかなく python コマンドがない状態となる。

対応方法

仮想環境を作成し、そこに hive-builder をインストールして、仮想環境をアクティベートしてから hive コマンドを実行してください。仮想環境をアクティベートすると、OS には python3 しかインストールされていな状態でも python コマンドが利用できます。

20.13 異常がないのに zabbix で At least one of the services is in a failed state のトリガーがあがります

現象

異常がないのに zabbix で At least one of the services is in a failed state のトリガーがあがる。以下のコマンドを実行すると失敗しているサービス名はわかったが、そのサービスはすでに削除されている。たとえば、DRBD のボリュームがエラーになった後、 build-volumes -l ボリューム名 -D などで削除した場合、以下のように表示される

```
$ systemctl list-units --type=service --no-pager --no-legend --state=failed --all
drbd-resource@some_data.service loaded failed failed DRBD resource : some_data
```

原因

サービスを削除した後、 systemd が失敗したユニットを記憶しているため、アイテムの数が 0 になりません。

対応方法

以下のコマンドでリセットしてください。

```
$ sudo systemctl reset-failed
```

20.14 異なるホストに配置されたサービス間の通信ができません

現象

異なるホストに配置されたサービス間で通信できない。例えば、hive-builder のサンプルにおいて、powerdns サービスから pdnsdb へのアクセスが異なるホストに配置されたときのみアクセスができないという現象が発生する場合がある。

原因 1

VMWare の NSX 機能やネットワーク機器の VXLAN 機能が動作していることが原因で swarm のオー

バレイネットワークの通信に必要な 4789/udp のパケットが到達できない。https://stackoverflow.com/questions/43933143/docker-swarm-overlay-network-is-not-working-for-containers-in-different-hosts

原因 2 ホストに複数のネットワークインタフェースがある場合に swarm のオーバレイネットワークの通信に利用する IP アドレスが間違っている

原因 3 ネットワークカードに offload したチェックサム照合機能がパケットをチェックサム不整合で破棄している。VMWare の仮想 NIC はこれに該当する。https://stackoverflow.com/questions/66251422/docker-swarm-overlay-network-icmp-works-but-not-anything-else

原因 4 ホスト間のネットワークの mtu が 1500 より小さく、 VXLAN のヘッダが付いたパケットをドロップしてしまう。ただし、この場合は全く通信できないわけではなく、サイズが大きいパケットのみがドロップされる。

対応方法 原因 1, 原因 2 の場合は、以下の手順で docker swarm のオーバレイネットワークが使用するポート番号や IP アドレスを変更してください。原因 3 の場合は各ホストで ethtool -K <interface> tx off コマンドを実行してネットワークカードへの offload を無効化してください。原因 4 の場合は各環境に応じてホスト間のネットワークの mtu が 1500 以上となるように設定してください。ただし、GCP の VPN と併用する場合は、VPN 側の制限と相反する場合がありますので、注意が必要です。MTU に関する考慮事項 を参照してください。

20.14.1 1. 全サービスを削除

オーバレイネットワークを再構築するために一旦全サービスを削除してください。

```
hive deploy-services -D
```

20.14.2 2. iptables を修正

リポジトリサーバを除く各ホストの以下の手順で /etc/sysconfig/iptables を修正し、4789 を 8472 に置換して iptables を再起動してください。

```
hive ssh -t ホスト名
vim /etc/sysconfig/iptables
sudo systemctl restart iptables
sudo systemctl restart docker
logout
```

20.14.3 3. swarm クラスタの解除

リポジトリサーバを除く各ホストの以下の手順でクラスタを解除してください。

```
hive ssh -t ホスト名
docker swarm leave --force
logout
```

20.14.4 4. swarm クラスタの初期化

1 号機で以下の手順を実行してクラスタを構築してください。

```
hive ssh -t 1号機のホスト名
docker swarm init --advertise-addr 1号機の IP アドレス --data-path-port 8472
docker swarm join-token manager
logout
```

docker swarm join-token manager で表示されたトークンの値を記録してください。

20.14.5 5. swarm クラスタの構築

1 号機以外のホスト（リポジトリサーバを除く）で以下の手順を実行してクラスタを構築してください。

```
hive ssh -t ホスト名
docker swarm join --advertise-addr ホストの IP アドレス --token トークン 1号機の IP アドレス
:2377
logout
```

20.14.6 6. hive_default_network の復旧

以下のコマンドで hive_default_network を復旧してください。

```
hive build-networks
```

20.14.7 7. サービスを起動

以下のコマンドで全サービスを起動してください。

```
hive deploy-services
```

20.14.8 8. follow-swarm-service 再起動

以下のコマンドをリポジトリサーバを除く各ホストで実行し、follow-swarm-service を再起動してください。

```
hive ssh -t ホスト名
sudo systemctl restart follow-swarm-service.service
logout
```

20.15 dockerhub からイメージをダウンロードするときにダウンロードでエラーになります

現象

dockerhub からイメージをダウンロードする際に toomanyrequests のエラーになる。例えば、setup-hosts フェーズの zabbix サーバの起動タスクで以下のようなエラーになる。

```
TASK [zabbix : compose up zabbix container]␣
↪*********************************************************************************************************
fatal: [p-hive0.nec-hss]: FAILED! => changed=false
errors: []
  module_stderr: ''
  module_stdout: ''
  msg: 'Error starting project 500 Server Error for http+docker://localhost/v1.41/images/
↪create?tag=alpine-5.2-latest&fromImage=zabbix%2Fzabbix-server-mysql: Internal Server␣
↪Error ("toomanyrequests: You have reached your pull rate limit. You may increase the␣
↪limit by authenticating and upgrading: https://www.docker.com/increase-rate-limit")'
```

原因

dockerhub の アクセス頻度制限 の上限を超えてアクセスした。アクセス頻度制限は dockerhub から匿名でダウンロードしようとすると 100 回/6 時間の制限がかかる。通常、1 ユーザで 100 回/6 時間の制限に抵触することはが、匿名のアクセスについてはユーザを送信元 IP アドレスで特定されるため、企業内のネットワークから複数人でアクセスすると同じユーザと認識されて制限にかかる場合がある。

対応方法

hive_ext_repositories に dockerhub のアカウントを設定する。設定方法については *hive* **構築ガイド** の外部リポジトリへのログインの節を参照のこと。これにより送信元 IP ではなく、アカウントに結びついたダウンロードとなるため、200 回/6 時間の制限となるとともに企業内のネットワークから複数人でアクセスする場合でもここのユーザごとの制限数となる。

20.16 gcp プロバイダを使用している場合に hive build-infra で Permission denied のエラー

現象

gcp プロバイダを使用している場合に ssh 接続が Permission denied のエラーとなる。例えば、build-infra フェーズの wait_for_connection タスクで以下のようなエラーになる。

```
TASK [wait_for_connection] ***********************************************************
fatal: [hive3.pdns]: FAILED! => changed=false
elapsed: 600
msg: 'timed out waiting for ping module test: Failed to connect to the host via ssh:␣
↪admin@34.97.59.48: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).'
```

原因

プロジェクトの設定で OS Login が有効になっているため、 hive の管理者ユーザが生成されない。

対応方法

OS Login の設定方法の「ステップ 1: OS Login を有効または無効にする」を参照して OS Login を無効に設定する。具体的には、以下のように設定する。

「[メタデータ] に移動」→GCP コンソール画面「メタデータ」→ 編集

キー 1: enable-oslogin 値 1: TRUE

↓

キー 1: enable-oslogin 値 1: FALSE

参考

https://cloud.google.com/compute/docs/troubleshooting/troubleshooting-ssh

20.17 サービスが特定のサーバに偏ってしまったようですが、どうしたらいいですか

サーバの再起動などで、サービスの配置が偏ってしまった場合、以下の手順でサービスを再配置位することができます。

20.17.1 1. サービスの配置状況を確認

コンテナ収容サーバのいずれかに ssh でログインし、以下のコマンドを実行して replicated モードのサービスの配置状況を確認してください。global モードのサービスはすべてのコンテナ収容サーバに配置されていますので、偏ることはないので、作業の対象から除外します。

```
docker service ps $(docker service ls -q --filter mode=replicated) --format "{{.Name}}\t{
↪{.Node}}\t{{.CurrentState}}" -f desired-state=running
```

20.17.2 2. サービスを再配置

コンテナ収容サーバのいずれかに ssh でログインし、以下のコマンドを実行してサービスを再配置してください。

```
docker service ls -q --filter mode=replicated | xargs -L 1 docker service update --force
```

20.17.3 1. サービスの配置状況を再確認

1. と同じコマンドでサービスの配置状況を確認してください。

20.18 構築後に internal_cidr の値を変更するにはどうしたらいいですか

以下の手順で変更してください。

20.18.1 1. internal_cidr を変更

inventory/hive.yml の internal_cidr の値を変更してください。

20.18.2 2. swarm クラスタの解除

以下のコマンドをすべてのコンテナ収容サーバで実行して、 swarm クラスタを解除してください。

```
docker swarm leave -f
```

20.18.3 3. docker_gwbridge の削除

以下のコマンドをすべてのコンテナ収容サーバで実行して、 docker_gwbridge を削除してください。

```
docker network rm docker_gwbridge
```

20.18.4 4. zabbix とリポジトリサービスの停止

以下のコマンドをリポジトリサーバで実行して、 zabbix とリポジトリサービスを停止してください。

```
(cd zabbix; docker-compose down)
(cd registry; docker-compose down)
```

20.18.5 5. setup-hosts フェーズの実行

以下のコマンドをマザーマシンで実行して、setup-hosts フェーズを実行してください。

```
hive setup-hosts
```

20.18.6 6. docker デーモンの再起動

以下のコマンドをすべてのサーバで実行して、docker デーモンを再起動してください。

```
systemctl restart docker
```

20.18.7 7. ネットワークのビルド

以下のコマンドを実行して、ネットワークをビルドしてください。

```
hive build-networks
```

20.18.8 8. サービスのデプロイ

以下のコマンドを実行して、サービスをデプロイしてください。

```
hive deploy-services
```

20.18.9 アドレスの確認

割り当てられているアドレスを確認するためには手順の前後で以下のコマンドを各サーバで実行してください。

```
docker network inspect $(docker network ls --format "{{.Name}}") | grep Subnet
```

手順実行後に internal_cidr の範囲内のアドレスのみが出るようになれば正解です。サービスやボリュームは作り直す必要はありません。

20.19 hive ssh および ssh コマンドでの ssh 接続ができません

現象

hive ssh または ssh コマンドでの ssh 接続をしようとすると以下のメッセージが出て失敗する

```
Host key verification failed.
```

原因 1

サーバーの公開鍵の紛失または誤って内容を変更してしまった

原因 2

サーバーの秘密鍵が変更された

対応方法

公開鍵と秘密鍵の不一致による現象であるため、現在入っている公開鍵を削除し、build-infra フェーズに

て新たに公開鍵を生成する

20.19.1 1. 公開鍵の削除

以下のコマンドをマザーマシンで実行して、ssh 接続に使用されるサーバーの公開鍵を削除してください。

```
rm .hive/(ステージ名)/known_hosts
rm ~/.ssh/known_hosts
```

20.19.2 2.build-infra フェーズを実行

以下のコマンドをマザーマシンで実行して、build-infra フェーズを実行してください。

```
hive build-infra
```

20.19.3 ssh 接続の確認

上記手順実行後に、以下のコマンドを実行してください。

```
hive ssh
```

または

```
ssh -F .hive/(ステージ名)/ssh_config ホスト名
```

第 21 章

bash completion の利用

hive で bash completion を利用するためにはシェルに hive-completion.sh を bash に読み込ませる必要があります。例えば、 hive-builder がインストールされている仮想環境を activate した後、以下を実行することで hive の bash completion を利用できます。

```
source "$(hive get-install-dir)/hive-completion.sh"
```

あるいは、仮想環境が activate している状態で、以下のコマンドを実行することで、仮想環境の activate 時に自動的に hive-completion.sh を読み込むように設定することができます。

```
hive setup-bash-completion
```

このコマンドは、仮想環境の activate スクリプトを編集してスクリプトの末尾に以下のコードを追加します。

```
source "$(hive get-install-dir)/hive-completion.sh"
HIVE_VIRTUAL_ENV_INIT=1
```

bash completion を利用するためには、コマンド実行後、 activate しなおしてください。ただし、 pyenv で仮想環境を切り替えている場合は activate スクリプトは評価されませんので、この方法は利用できません。