hive-builder

リリース *0.1.0*

Mitsuru Nakakawaji

2020 年 11 月 07 日

Contents:

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

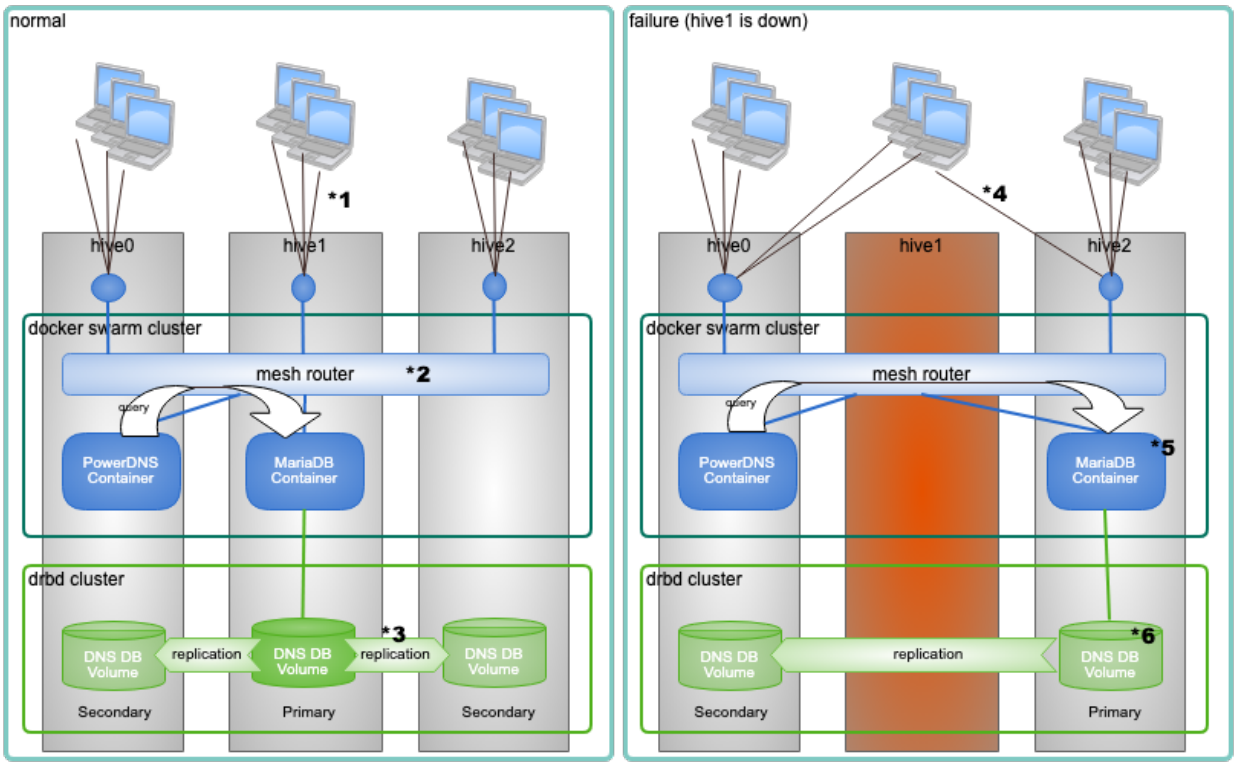
警告: まだ、ドキュメントに未執筆部分がたくさんあることと、今後、非互換となる変更を行う可能性があることから、ビジネス用途の本番環境でご利用になるのは難しいかもしれません。本番環境でご利用になる場合はご一報ください。

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 LDAP サーバを hive2 にマイグレーション
6. LDAP サーバ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

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 のインストール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 3.6 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

各 OS ごとにインストールの手順を示します。

2.2 Centos の場合

注釈: 現在サポートしているのは CentOS 7 のみです。CentOS 6, CentOS 8 は未サポートです。

2.2.1 docker コマンドのインストール

以下のコマンドを root で実行して docker-client をインストールしてください。

```
yum install -y docker-client
```

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。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。

2.2.2 仮想環境の作成例

hive-builder をインストールするための仮想環境 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。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しても良いでしょう。

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

2.2.3 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
```

インストールがエラーになる場合は、 pip install -U pip wheel で pip と wheel を最新バージョンにアップデートしてみてください。

2.3 Windows Subsystem for Linux の場合

2.3.1 docker コマンドのインストール

以下のコマンドを root で実行して docker.io をインストールしてください。

```
apt-get update
apt-get install docker
apt docker.io
```

2.3.2 仮想環境と hive-builder のインストール

仮想環境と hive-builder のインストールについては、Cent OS の場合と同じです。 **そちら** を参照してください。

2.3.3 ssh 鍵の mode の問題

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 /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。

2.4 Mac OS の場合

2.4.1 docker コマンドのインストール

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。

2.4.2 仮想環境と hive-builder のインストール

仮想環境と hive-builder のインストールについては、Cent OS の場合と同じです。 **そちら** を参照してください。

2.5 raspbian へのインストール

raspberry pi にインストールする場合は、OS に raspbian を利用し、以下の手順で必要なソフトウェアをインストールしてください。

```
apt-get update
apt-get upgrade
curl -sSL https://get.docker.com | sh
usermod -aG docker pi
apt-get install build-essential libssl-dev libffi-dev python3-dev subversion python3-
↪venv subversion xorriso
```

2.5.1 仮想環境と hive-builder のインストール

仮想環境と hive-builder のインストールについては、Cent OS の場合と同じです。 **そちら** を参照してください。

第 3 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: 現在のバージョンでは vagrant プロバイダを使用する場合のみプロキシ環境下で構築していただけます。

3.1 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、プロキシサーバの内容を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
```

(次のページに続く)

(前のページからの続き)

```
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。

第 4 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。

4.1 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。

4.1.1 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。以下の権限を持つ IAM ユーザを作成し、その API 鍵を取得してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole",
        "aws-marketplace:*",
        "resource-groups:*",
        "ec2:*",
        "tag:*",
        "route53:AssociateVPCWithHostedZone"
      ],
      "Resource": "*"
    }
```

(次のページに続く)

(前のページからの続き)

```
  ]
}
```

4.1.2 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。このためには、親ドメインに NS レコードと A レコードを登録する必要がありますが、これは必須ではありません。

4.2 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
```

4.3 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 3.7.3 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 3.7.3 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。

4.4 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
```

4.5 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
```

4.6 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 staging に切り替えてください。

```
hive set stage staging
```

4.7 AWS の設定

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
```

4.8 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。

4.8.1 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
```

4.8.2 build-infra の実行

以下のコマンドで build-infra フェーズを実行して、 Elastic IP を割り当ててください。

```
hive build-infra
```

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。また、コマンドの実行に成功すると、 start_phase 変数に 'setup-hosts' が設定され、次に hive all を実行した際には setup-hosts フェーズから始まります。

4.8.3 DNS レコードの登録

親ドメインに NS レコードと 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。設定例は以下の通りです。

```
pdns.example.com. IN NS s-hive0.pdns.example.com.
pdns.example.com. IN NS s-hive1.pdns.example.com.
pdns.example.com. IN NS s-hive2.pdns.example.com.
```

(次のページに続く)

(前のページからの続き)

```
s-hive0.pdns.example.com. IN A 10.1.1.4
s-hive1.pdns.example.com. IN A 10.1.2.4
s-hive2.pdns.example.com. IN A 10.1.3.4
```

ここで 10.1.1.4, 10.1.2.4, 10.1.3.4 の部分は EC2 インスタンスに関連付けられた Elastic IP で置き換えます。Elastic IP は AWS コンソールか .hive/staging/ssh_config のファイル内の Host ディレクティブの値を見ることで調べることができます。

4.9 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)、日次バックサ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

4.10 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。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。

4.11 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。

4.12 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。

4.12.1 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
```

4.12.2 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。

4.13 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 5 章

hive 構築ガイド

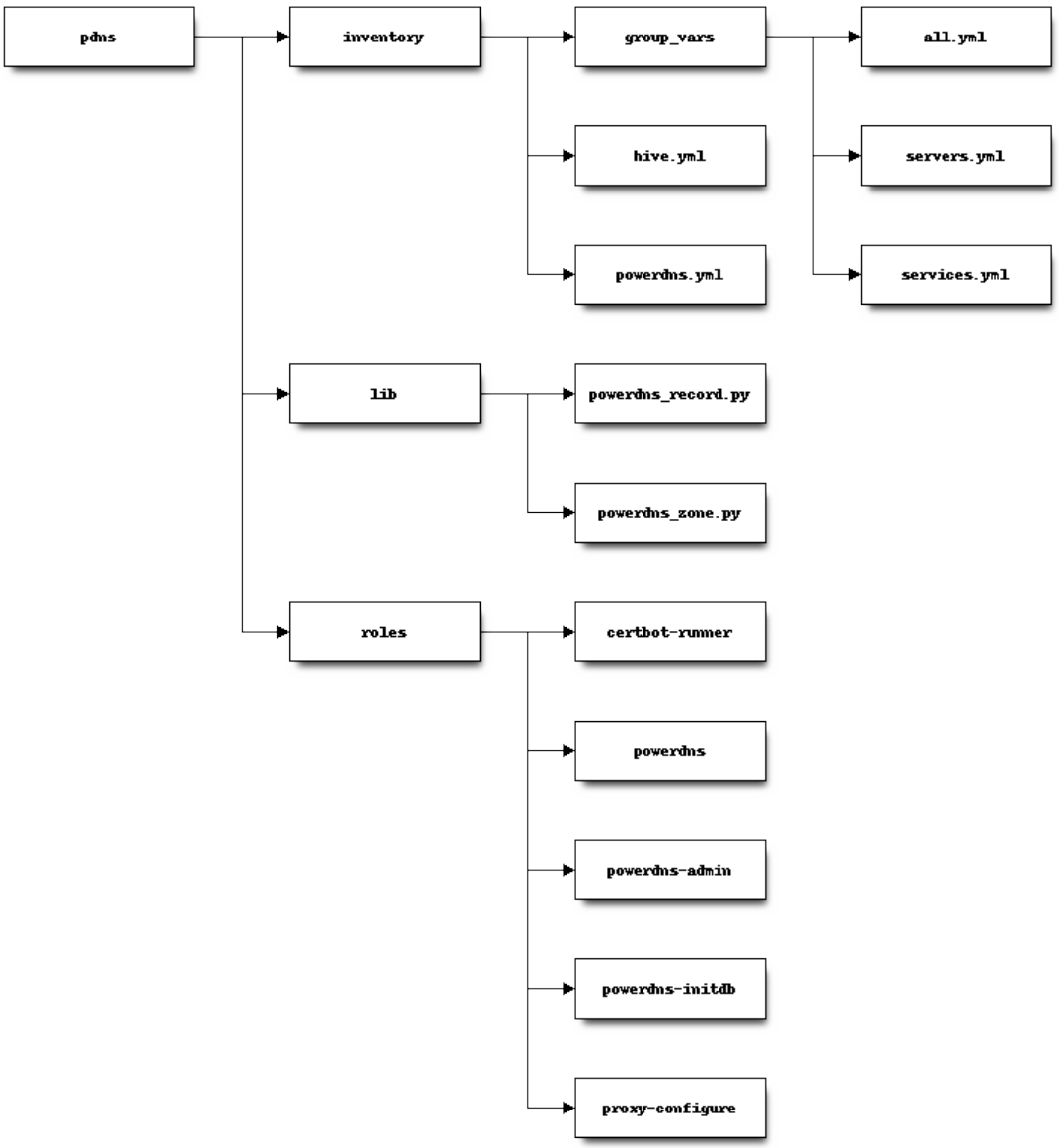
ここでは、hive のサイトを構築する方法を説明します。

5.1 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。

5.2 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。

5.2.1 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: ""
- name: "patch default config file - comment out"
  lineinfile:
    path: /etc/pdns/pdns.conf
    regexp: "^(# *)?{{ item }}=.*"
    line: "# {{ item }}="
  with_items:
    - use-logfile
```

(次のページに続く)

(前のページからの続き)

```
    - 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

5.3 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。

5.3.1 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
```

5.4 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。

5.4.1 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
```

5.4.2 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
```

5.4.3 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
```

5.4.4 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
```

5.4.5 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
```

(次のページに続く)

(前のページからの続き)

```
- name: add zone
  powerdns_zone:
    name: "{{ hive_name }}.{{ domain }}."
    nameservers: "{{ groups['hives'] | intersect(groups[hive_stage]) | map('regex_
↪replace', '^(.*)$', '\\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。

5.4.6 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。

5.4.7 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。

第 6 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}
            ...
```

6.1 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/foros7/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

6.2 Sub-commands:

6.2.1 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

6.2.2 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

6.2.3 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

6.2.4 build-volumes

build volumes on hives

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

Named Arguments

-D, --destroy destroy volume

Default: False

-l, --limit_target limit target

6.2.5 build-networks

build networks for swarm

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

6.2.6 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

6.2.7 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

6.2.8 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.

6.2.9 inventory

list ansible inventory

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

6.2.10 init

initialize hive environment

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

6.2.11 set

set hive variable persistently

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

Positional Arguments

variable_name variable name

value variable value

6.2.12 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

6.3 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

6.4 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。

6.5 .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。

6.6 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。

6.7 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。

6.8 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

6.9 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。

6.10 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"
```

6.11 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 7 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。

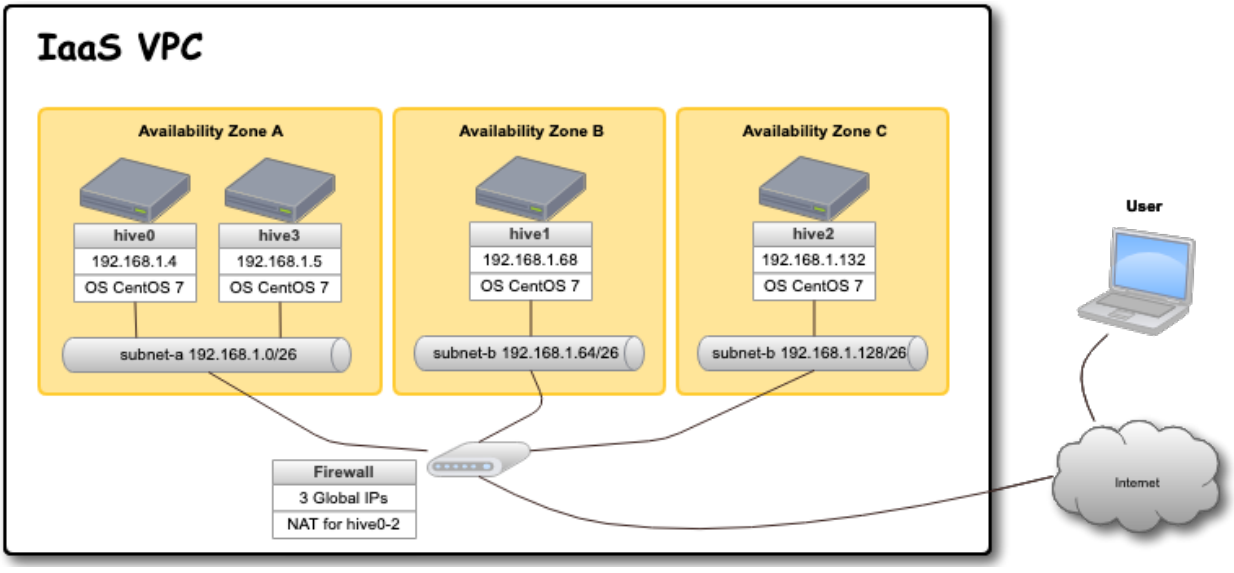
hive 定義　サーバやネットワークで構成される基盤（infrastructure）を定義する

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

7.1 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。



7.1.1 hive 定義のフォーマット

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

(次のページに続く)

(前のページからの続き)

```
    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。

7.1.2 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

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。

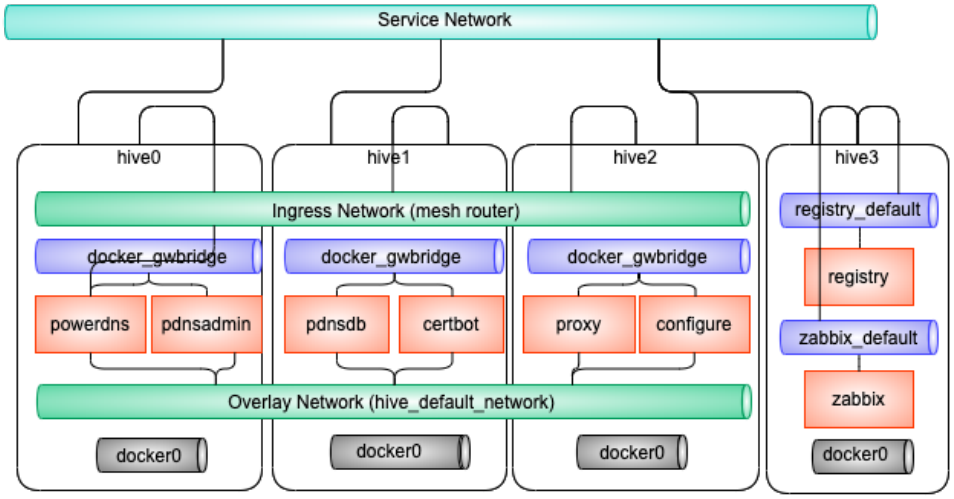
上記以外の場合：サーバの IP アドレスは cidr 属性から自動的に割り振ります。アベラブルゾーンは自動的にできるだけ分散するように割り当てます。

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。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

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
```

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

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。

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、現在のバージョンではサーバは 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）
media_usb	• True • False	False	インストール媒体が USB であるかいなか
networks	未執筆	必須	ネットワーク定義オブジェクトのリスト

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
  media_usb: True
  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
  - interface: bond0
    vlanid: 2
  - interface: bond0
    vlanid: 4
  - interface: bond0
    vlanid: 1000
  - interface: bond0
    vlanid: 1001
  - interface: bond0
    vlanid: 1002
    ips:
    - 192.168.203.20
    - 192.168.203.21
    - 192.168.203.22
    netmask: 255.255.255.0
    gateway: 192.168.203.1
    nameservers:
```

(次のページに続く)

(前のページからの続き)

```
    - 192.168.203.1
  - interface: bond0
    vlanid: 1003
  - interface: bond0
    vlanid: 1004
```

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/sdc
- /dev/sdb
- /dev/vdc
- /dev/vdb

- /dev/xvdb
- /dev/nvme1n1
- /dev/sda

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。

7.2 サービス定義

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）
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）
mode	• replicated • global	replicated	サービス・モード（docker service create の --mode オプションと等価）

placement	{"constraints": "node.hostname!=hive2"}	{}	サービスの配置に関するルールを constraints 属性、preferences 属性で指定する（docker service create の

7.2.1 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 属性と同時に指定することはできません）

drbd 属性、driver 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。

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。

7.2.2 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 role

python-aptk はビルトイン role であり、イメージのビルド時に role 定義を行わずに使用できます。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
```

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。

7.2.3 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
```

7.2.4 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 }}"
```

(サービス定義の backup_scripts, volumes 属性の詳細については未執筆)

第 8 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。

8.1 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）

8.1.1 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。

8.2 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。

8.2.1 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
NetworkManager	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。

8.2.2 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 の Base リポジトリの yum のダウンロード元として指定します。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 の無効化

selinux を無効にします。

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。

auxiliary-networks role

hive_auxiliary_networks 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追加します。（詳細省略）

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。

8.2.3 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。

8.2.4 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。

8.3 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。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、 hive ssh でリポジトリサーバにログインし、 docker rm で build_サービス名の名前のコンテナを削除してから build-images をやり直してください。

8.3.1 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、 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）

8.4 build-networks

内部ネットワークを構築します。

（未執筆）

8.5 build-volumes

ボリュームを構築します。

（未執筆）

8.6 deploy-services

サービスを配備します。

（未執筆）

8.6.1 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。

8.7 initialize-services

サービスを初期化します。

（未執筆）

8.7.1 docker コネクション

initialize-serivices フェーズでは、ssh tunneling でサーバの /var/run/docker.sock をマザーマシンの /var/tmp/hive/docker.sock@サーバ名 に転送します。docker コネクションを使用してサービスのコンテナ内に対して ansible を実行する場合には、最初に docker service ps でコンテナが動作しているノードを特定してから、ssh tunneling で転送されているソケットに接続する必要があります。以下に playbook の例を示します。

```
- name: setup awx project
  gather_facts: False
  hosts: awx_web,awx_task

  tasks:
  - name: get server
    delegate_to: "{{ groups['first_hive'] | intersect(groups[hive_stage]) | first }}"
    shell: docker service ps --format "{% raw %}{{.Name}}.{{.ID}}@{{.Node}}{% endraw %}
↪.{{ hive_name }}" --filter desired-state=running --no-trunc {{ inventory_hostname }}
    changed_when: False
    check_mode: False
    register: hive_safe_ps

  - name: setup docker socket
    set_fact:
      ansible_docker_extra_args: "-H unix://{{ hive_temp_dir }}/docker.sock@{{ hive_
↪safe_ps.stdout.split('@') | last }}"
      ansible_connection: docker
      ansible_host: "{{ hive_safe_ps.stdout.split('@') | first }}"

  - name: copy project for fun
    copy:
      dest: /var/lib/awx/project
      src: fun
```

第 9 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

9.1 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。

9.2 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
```

9.3 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 10 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。

10.1 1.2.1 からの移行

バージョン 1.2.1 では、リポジトリサーバのログローテートの設定が間違っていました。リポジトリサーバにログインして /etc/logrotate.d/syslog を編集してください。

誤： /var/log/services/*

正： /var/log/services/*.log

修正せずに新しいバージョンで setup-hosts を実行すると設定が 2 行になってしまいますので、ご注意ください。

10.2 1.2.1 以前からの移行

リポジトリサーバにインストールされる zabbix のテンプレートが 1.2.2 で変更になりました。以下の手順で更新してください。1. hive ssh -z でログイン 2. ブラウザで http://localhost:10052 を開いて、zabbix のコンソールにログイン（id: admin, password: zabbix) 3.「設定」→「テンプレート」を開いて、一覧表示 4.「Hive Repository Server」のチェックボックスをチェックして、最下行の削除をクリック 5.「削除しますか」の確認メッセージが表示されるので、OK をクリック 6.「設定」→「ホストグループ」を開いて、一覧表示 7.「Docker」のチェックボックスをチェックして、最下行の削除をクリック 8.「削除しますか」の確認メッセージが表示されるので、OK をクリック 9.「設定」→「ホスト」を開いて、一覧表示 10. hive0 の「ディスカバリールール」クリックして一覧表示 11.「Volume usage discoverry」のチェックボックスをチェックして、最下行の削除をクリック 12.「削除しますか」の確認メッセージが表示されるので、OK をクリック 13. 10. から 12. の操作を hive1, hive2 についても実行 14. 1. のコマンドを exit 15. hive setup-hosts -T zabbix,zabbix-agent を実行 16. hive ssh -t hive0.hive 名 で hive0 にログイン 17. sudo systemctl restart zabbix-agent 18. exit 19. 16. から 18. を hive1, hive2 についても実行

10.3 1.1.7 以前からの移行

1.1.7 以前でマイクロサービス型のサービスのログを参照する場合には、コンテナ収容サーバにログインして、docker service logs サービス名で参照する必要がありましたが、1.1.8 からはリポジトリサーバの /var/log/services/サービス名.log で参照できるようになりました。これを利用するためには、 hive setup-hosts -T rsyslogd を実行してリポジトリサーバに rsyslogd をセットアップし、 hive deploy-services でサービスをデプロイし直す必要があります。

10.4 1.1.5 以前からの移行

1.1.5 以前では zabbix/zabbix-web-nginx-mysql:ubuntu-trunk で提供されるイメージが 80 番ポートを公開している前提でしたが、zabbix/zabbix-web-nginx-mysql:ubuntu-trunk の公開ポートが 8080 に変わったためイメージをロードし直す必要があります。hive setup-hosts を実行する場合は、その前に以下のコマンドを実行して、zabbix のイメージを更新してください。

```
$ hive ssh
$ cd zabbix
$ docker-compose down
$ docker image rm zabbix/zabbix-server-mysql:ubuntu-trunk
$ docker image rm zabbix/zabbix-web-nginx-mysql:ubuntu-trunk
$ exit
```

この後、hive setup-hosts を実行してください。

第 11 章

よくある質問

11.1 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。

11.2 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。

11.3 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。

11.4 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

対応方法 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。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。

11.5 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。

11.6 UDP のサービスが特定のクライアントからのパケットを全く受信できません

準備 hive-builder で構築したサーバでは conntrack コマンドがインストールされていません。 コンテナ収容サーバに yum install conntrack-tools で conntrack コマンドをインストールしてください

発生条件 1 系の hive-builder で構築したサーバで特定の IP アドレス、ポート番号から 30 秒より短い間隔で常時 UDP パケットを受信している状態でサービスを起動あるいは再起動した場合に発生します。忙しい DHCP リレーエージェントや syslog クライアントのリクエストを hive-builder のサービスで処理する場合はこれに該当します。

原因 ホストに着信したパケットは、仮想ブリッジを経由してサービスを実装するコンテナに転送されるべきですが、この転送機能が機能していません。docker では、Linux の iptables の DNAT 機能を使用してパケットを転送しますが、このパケット転送が設定されていない状態で UDP パケットを受信すると DNAT がされていない情報が conntrack に登録されます。その状態でサービスを起動すると、docker が iptables に DNAT を登録しますが、 iptables の DNAT 機能は対象となるパケットにマッチする情報が conntrack テーブルにすでに登録されている場合は、その情報が優先して利用されるため、作動しません。conntrack テーブルには、送信元 IP アドレス、宛先ポート番号、宛先 IP アドレス、宛先ポート番号をキーとして登録されますが、プロトコルが UDP の場合はパケットを着信しない状態で 30 秒経過す

ると削除されます。発生しているかどうかは conntrack コマンドで確認できます。conntrack に DNAT された情報が登録されていれば、2 個めの src= の後ろにコンテナの IP アドレスが表示されますが、問題が発生している場合は、サーバの IP アドレスが表示されます。以下にリクエストを受け付けるポート番号が 20514 である場合の例を示します。

```
# 正常な場合
$ sudo conntrack -L -p udp --dport 20514
udp      17 26 src=192.168.56.1 dst=192.168.56.4 sport=55646 dport=20514 [UNREPLIED]␣
↪src=172.21.34.130 dst=192.168.56.1 sport=514 dport=55646 mark=0 secctx=system_
↪u:object_r:unlabeled_t:s0 use=1
conntrack v1.4.4 (conntrack-tools): 1 flow entries have been shown.
# 問題が発生している場合
$ sudo conntrack -L -p udp --dport 20514
udp      17 26 src=192.168.56.1 dst=192.168.56.4 sport=64953 dport=20514 [UNREPLIED]␣
↪src=192.168.56.4 dst=192.168.56.1 sport=20514 dport=64953 mark=0 secctx=system_
↪u:object_r:unlabeled_t:s0 use=1
conntrack v1.4.4 (conntrack-tools): 1 flow entries have been shown.
```

対応方法 問題となる conntrack 情報を削除してください。conntrack コマンドに -D オプションを指定することで削除できます。conntrack コマンド実行直後のパケットから受信を再開するはずです。以下にリクエストを受け付けるポート番号が 20514 である場合の例を示します。

```
$ conntrack -D -p udp --dport 20514
udp      17 25 src=192.168.56.1 dst=192.168.56.4 sport=51109 dport=20514 [UNREPLIED]␣
↪src=192.168.56.4 dst=192.168.56.1 sport=20514 dport=51109 mark=0 secctx=system_
↪u:object_r:unlabeled_t:s0 use=1
conntrack v1.4.4 (conntrack-tools): 1 flow entries have been deleted.
```