

初めてのインフラ

PHH16 杉山 湧生 (pikatyuu)

今日はインフラについてのLTをしたいと思います

インフラてなんだっけ？

いや待て、そもそもインフラってなんだっけ？

インフラとはインフラストラクチャの略で

意味は基板、つまり、システムが動作するための基板となる回線やOSの設定の作業のことを指します

最近ではアマゾンが提供しているAWSやMicrosoftが提供しているazureといったクラウドシステムが見られるようになり、OSの設定から始めることができるようになってきているようです

なぜインフラ？

インフラというものを知ったところで次に聞きたいこと、なぜインフラなのか

- ・ **ISUCON**で**武器**になりそうだから
- ・ セキュリティ関係のサイトを調べると『**ログ見ればわかりますよ**』とかが多い気がしたから
- ・ 授業でやった中で**一番よくわからない分野**だったから

インフラを勉強したいと思った理由はまずイスコンをやってみてインフラの勉強してたらOSとかに詳しくなって武器になりそうだったからです

勝負は買ったほうが楽しいですからね

あとは、セキュリティ関係のサイトを見るとログ見ればわかりますよ、みたいな文が多く見られた気がしてログを学んでみたかったからってことや、授業で一番よくわからなかった分野だったから、ということが挙げられます。

目標

では早速、今回の課題で課せられた目標

Ubuntu を Vagrant で 自動構築

UbuntuをVagrantで自動構築

Vagrant ?

ん？Vagrantとはなんだ？

授業でやったこと

VirtualBoxを使ってUbuntu環境を構築



すごい時間かかるし

めんどくさい

授業ではバーチャルボックスを使いUbuntuを構築しましたよね

しかし、やたら時間がかかったし何よりもめんどくさかった印象を受けたと思います

そこで**Vagrant!**

そこで Vagrantの出番です

利点

はやい

かんたん

仮想環境構築を共有出来る

使う利点としては

早い、かんたん、そして仮想環境構築を共有できることにあります

つまり、言ってしまうとそこらへんに上がっているVagrantfileというものをgit cloneしてくるなりしてローカルに落としてしまえば何も設定することなく仮想環境を作れてしますわけですから
もちろん今回は勉強しにやっているわけなので一からvagrantfileを作成します

使ってみる

実際に使ってみます

適当にディレクトリ作って…

```
$ mkdir src/github.com/vantan-phh/stand_up_server  
$ cd !$
```

適当にディレクトリを作って

Vagrantfileを作成

```
$ vagrant init -m ubuntu/trusty64
```

ここまではかんたん！

vagrant initというコマンドで新規のvagrantファイルを作成します

ここまでは簡単ですね

内容確認

doの後にあるconfigのversion

```
Vagrantfile
1  Vagrant.configure("2") do |config|
2    config.vm.box = "ubuntu/trusty64"
3    config.vm.provider :virtualbox do |v|
4      v.customize ["modifyvm", :id, "--ostype", "Ubuntu_64"]
5    end
6  end
```

今回利用するOS

使用するVMの設定

OSのbitの明示

内容の確認をします

まず行の一番上、`vagrant.configure(2)do config`の所ではdoの後にあるconfigのversionの設定を2で設定しています

これは1と2がありデフォルト値は2なのですが

後に説明するproviderという機能が1にはないため2が良いと思います

3行目ではproviderと言ってvagrantで使用するVMの設定を行っています。今回はvirtual boxを使用するのでvirtual boxを設定しておきます

4行目ではUbuntuを起動するときにOSTypeの明示を行っています

これを行わないと32bitのパソコンでこのVagrantfileを利用しようとしたときに

なかなかUbuntuが起動しないなどのエラーが発生してしまうことがあるようです

動かしてみる

ではこのように設定したのち動かしてみます

```
$ vagrant status
```

```
not created (virtualbox)
```

```
$ vagrant up
```

```
$ vagrant status
```

```
running (virtualbox)
```

vagrant statusコマンドで現在動いているVMの確認をします
not createdとなってますのでvagrant upコマンドで起動します
もう一度ステータスをみると
runningとなって起動したことがわかります


```
$ vagrant ssh
```

```
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-100-generic x86_64)
```

```
* Documentation: https://help.ubuntu.com/
```

```
System information as of Thu Oct 27 04:32:51 UTC 2016
```

```
System load: 0.0          Processes:      86  
Usage of /:  4.6% of 39.34GB  Users logged in:  0  
Memory usage: 28%          IP address for eth0: 10.0.2.15  
Swap usage:  0%            IP address for docker0: 172.17.42.1
```

```
Graph this data and manage this system at:  
https://landscape.canonical.com/
```

```
Get cloud support with Ubuntu Advantage Cloud Guest:  
http://www.ubuntu.com/business/services/cloud
```

```
New release '16.04.1 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.
```

```
Last login: Thu Oct 27 04:32:51 2016 from 10.0.2.2  
vagrant@vagrant-ubuntu-trusty-64:~$
```

起動したVMにsshで入るときはvagrant sshでできます

動いただけだろ！
動いた！

こんな感じでWelcome to Ubuntuの表示が出たので、無事動いたっぽいです

まあ動いただけです

とりあえず箱(Ubuntu)ができたので
装飾していく

とりあえず箱が出来上がったので装飾していきたいと思います

目標

目標

作ったUbuntuに
docker と 各logのローテート機能 を実装

作ったUbuntuにdockerと各ログのローテート機能を実装せよ

ログのローテート？

ログのローテート？

- ・ とあるウェブサービスがある
- ・ 日本での1日のアクセス数は約1000万
- ・ そのアクセスログを取ることにした
- ・ 1 か月（31日）ではどれくらいのログがある？

A. 3 億1000万回分

例えば、とあるウェブサービスがあって

日本での1日のアクセス数が約1 0 0 0万らしいです

そのアクセスログを取った時に1 か月ではどのくらいになるかと考えた時に

答えは3億1 0 0 0万分のログを取ることになるんですが

3 億1000万回分のログを一つのファイルにまとめる



仮に1つのログが1行だとしても3億行以上



読みづらい 読みたくない

そのログを一つのファイルにまとめるとして
仮に1つのログが1行だとしても3億行以上となって
読みづらい、読みたくない

こうならないための ログローテート

こうならないためのログローテートでして、

- ・ 指定の日数が経過したら、
自動で別のファイルにログの出力を変えてくれる
- ・ 古いログファイルは削除されるようにしてくれる
- ・ 古いログファイルを圧縮してくれる

指定の日数が経過したら自動で別のファイルにログの出力を変えてくれたり、
古いログファイルは削除してくれたり、
古いログファイルを圧縮してくれたりしてくれます

ぜひ実装しよう！

素晴らしい、ぜひ実装しよう！

どうやって？

どうやって？

手作業で！

手作業で！

勘違いするなよ？
お前は僕のパシリに
過ぎないんでちゅ。

はい…

分かったなら
さっさと手積み

やってこいよ。

はい、ピカチュウさん…



...

これは嫌なので

これは嫌なので

itamae使って自動化

itamae使って自動化しようと

vagrant-itamae
というプラグインがあると教えてもらい導入

```
$ vagrant plugin install vagran-itamae
```

これでかんたんにVagrantでitamaeが利用可能に！

vagrant-itamaeというプラグインがあると教えてもらったので早々に導入します

こんな感じでコマンド打つと簡単に導入されます

```
Vagrantfile
1  Vagrant.configure("2") do |config|
2    config.vm.box = "ubuntu/trusty64"
3    config.vm.provider :virtualbox do |v|
4      v.customize ["modifyvm", :id, "--ostype", "Ubuntu_64"]
5    end
6
7    config.vm.provision :itamae do |config|
8      config.sudo = true
9      config.shell = "/bin/bash"
10     config.recipes = [".recipe.rb"]
11   end
12 end
```

さっきのvagrant fileに黄色の枠を追加します

provisionというのはUbuntu起動後に行う作業のことで、

板前を指定した後sudoやshell, レシピの場所などを指定するだけで使用できるようになります

気になるrecipe.rbの中身

では、気になるrecipe.rbの中身はというと

```
recipe.rb
1  execute "apt-get update and upgrade" do
2    command "apt-get update && apt-get upgrade -y"
3  end
4
5  package "docker.io" do
6    action :install
7  end
8
9  execute "install logrotate" do
10    command "aptitude install logrotate"
11  end
```

こんな感じになってます

packageのインストールを行うので

一度全パッケージのupdateとupgradeを行います

その後、dockerとlog rotateをインストールします

動かしてみる

では動かしています

provision動作は

`$ vagrant provision`

でリブートしないでできる

毎回vagrant upとかしているのは馬鹿らしいと思っていたらprovisionの動作はvagrant provisionというコマンドでリブートせずにできるということだったので使用

```
$ docker
```

```
Usage: docker [OPTIONS] COMMAND [arg...]
```

```
A self-sufficient runtime for linux containers.
```

```
Options:
```

```
—api-cors-header= Set CORS headers in the remote API
```

```
...
```

Ubuntu側でdockerとlogrotateの動作を確認します


```
$ logrotate
```

```
logrotate 3.8.7 - Copyright (C) 1995-2001 Red Hat, Inc.  
This may be freely redistributed under the terms of the GNU Public License
```

```
Usage: logrotate [-dfv?] [-d|--debug] [-f|--force] [-m|--mail=command]  
        [-s|--state=statefile] [-v|--verbose] [--version] [-?|--help]  
        [--usage] [OPTION...] <configfile>
```

無事に両方インストールされてる

無事に両方インストールされていることが確認できました

しかし、
logrotateの設定しないといけない

しかし、log rotateを使用するには設定をしないといけないということでした

どうやら設定ファイルはログを吐くものそれぞれにあるらしい

調べてみると設定ファイルはログを吐くものそれぞれにあるらしく

/etc/logrotate.d内にlogrotateが自動で書いてくれてるけど
なるべく多くのOSに対応できるようにしているようで
たくさんファイルあってよくわからない…

/etc/logrotate.d内にlog rotateが自動で書いてくれているらしいのですが、なるべく多くのOSに対応できるようにしているらしくものすごい量のファイルがあって無駄が多いと思ったので

全部消しちゃえ！

全部消しちゃえ！

recipe.rbにrm追加

```
recipe.rb
1  execute "apt-get update and upgrade" do
2    command "apt-get update && apt-get upgrade -y"
3  end
4
5  package "docker.io" do
6    action :install
7  end
8
9  execute "install logrotate" do
10    command "aptitude install logrotate"
11  end
12
13  execute "remove all file on /etc/logrotate.d" do
14    command "rm /etc/logrotate.d/*"
15  end
```

recipe.rbにrm /etc/logrotate.d/*とかいて全消去

では新しく作っていこう

綺麗になったので新しく作ります

まずsyslog周りから

まずはsyslog周りからです

syslog?

syslogとは？

- ・ syslogはUbuntuでは/var/log/内にある
- ・ システムの動作状況や、メッセージを記録するログ
- ・ Unix/Linuxに標準に備わっている

syslogはUbuntuでは/var/log内であって

システムの動作状況や、メッセージを記録するログです

Unix/Linuxに標準に備わっているものだそうです

設定内容

古いファイルを消すまでのローテート数

ローテートするインターバル

ファイルが存在しなくてもエラーを出さない

古いファイルはgzipで圧縮する

ファイルが空ならローテートしない

```
/var/log/syslog
{
  rotate 10
  daily
  missingok
  compress
  notifempty
}
```

設定内容は

上からrotateが古いファイルを消すまでのローテート数

dailyがローテートするインターバル

これはweeklyやmonthlyなどもあります

missingはファイルがなくてもエラーを出さないようにするもので

compressは古いファイルはgzipで圧縮するものです

notifemptyはがいるが空ならローテートしない設定です

各ログの設定に合わせてログについて説明します

kernlog: カーネルのログ
起動時の動作が入ってる

dmesg: カーネルリングバッファのログ
カーネルの出力した情報を記録している

```
/var/log/kern.log  
/var/log/dmesg  
{  
    rotate 0  
    weekly  
}
```

kern.logというのはカーネルのログで起動時の動作がは記録されています

dimesはカーネリングバッファのログでカーネルの出力した情報を記録しています

つまり標準出力です

dpkg.log: Debian package managerの略
OS向けのパッケージ操作を行った時間と
内容を記録する

```
/var/log/dpkg.log
{
    rotate 7
    daily
    missingok
    compress
    notifempty
}
```

dpkglogはデビアンで作られたOS向けのパッケージ操作を行った時間と内容を記録する

apt/term.log: aptが行った動作で出力したメッセージを記録する

apt/history.log: 過去にaptが行った動作を記録する

```
/var/log/apt/term.log
/var/log/apt/history.log
{
    rotate 12
    monthly
    compress
    missingok
    notifempty
}
```

term.logはaptが行った動作で出力したメッセージを記録する

history.logは過去にaptが行った動作を記録するログです

auth.log: ログイン時のパスワードエラーが起きた時に記録される

```
/var/log/auth.log
{
    rotate 4
    weekly
    missingok
    compress
    notifempty
}
```

auth.logはログイン時のパスワードエラーが起きた時に記録されるログです

設定は大体適当

dockerデーモンも取っておきます

dockerデーモンのlogも取っておく

docker.dのログはubuntuでは/var/log/upstartてののの中にあります

ちなみに、upstartはシステムの起動や終了時の情報を記録したものです

デーモン: バックグラウンドで動作するプログラム

```
docker
1  /var/log/upstart/docker.log
2  {
3      rotate 3
4      weekly
5      missingok
6      compress
7      delaycompress
8      notifempty
9  }
```

dockerデーモンのはくログはbuildやrunなどのdocker自体に向けて行った動作で
コンテナ内で行った動作は記録してないので
あまり量が増えることはないだろうと思ったのですが
確証は持てないので一応1週間間隔でローテートするように設定しました
1ヶ月分取ればいいかなという勝手な考えからローテートの回数は三回になっています

設定終了

以上で設定終了です

動くかどうかを確認するため
無理矢理logrotate動かしてみる

動くかどうかのテストをするため無理矢理logrotate
を動かしてみます

```
$ logrotate -f /etc/logrotate.conf
```

```
$ ls
```

```
auth.log  
auth.log.1  
dpkg.log  
dpkg.log.1  
...
```

log rotateコマンドに-f フォース optionをつけて

実行します

で、lsで/var/log/内を見ていると見事ログがローテートされていることが確認できました

無事動いた

これでようやく目標達成です

まとめ

まとめ

- ・ 結局インフラらしいことは実際あんまりできてない？
- ・ ともあれ、
ログについてはある程度は知ることだできたので
成果はあった
- ・ まだまだ勉強していきたい！

結局インフラらしいことはあまりできてない印象でした

ともあれ、ログについてはある程度知ることができたので成果はあったと思います

でも今回は構えていたというか圧倒的な知らないことの前になかなか前に進めなかった感があるので

こういう分野は習うより慣れよみたいなのところも大きいと思いました

そんなことを意識してこれからも勉強したいと思いました。

質疑応答

質疑応答

ご静聴ありがとうございました

ご静聴ありがとうございました。