虚拟开发环境

问题：在开发机上面开发完毕程序，放到正式环境之后会出现各种奇怪的问题：描述符少了、nginx配置不正确、MySQL编码不对、php缺少模块、glibc版本太低等。

解决：需要虚拟开发环境，在本机跑虚拟机，例如VMware、VirtualBox等。在本机可以运行自己喜欢的OS（Windows、Ubuntu、Mac等），开发的程序运行在虚拟机中，这样迁移到生产环境可以避免环境不一致导致的莫名错误。

Vagrant

方便建立虚拟环境，而且可以模拟多台虚拟机，这样我们平时还可以在开发机模拟分布式系统。

Vagrant还会创建一些共享文件夹，用来给你在主机和虚拟机之间共享代码用。这样就使得我们可以在主机上写程序，然后在虚拟机中运行。如此一来团队之间就可以共享相同的开发环境，就不会再出现类似“只有你的环境才会出现的bug”这样的事情。

提升工作效率：团队新员工加入，常常会遇到花一天甚至更多时间来从头搭建完整的开发环境，而有了Vagrant，只需要直接将已经打包好的package（里面包括开发工具，代码库，配置好的服务器等）拿过来就可以工作了。

通过 Vagrant 封装一个 Linux 的开发环境，分发给团队成员。成员可以在自己喜欢的桌面系统（Mac/Windows/Linux）上开发程序，代码却能统一在封装好的环境里运行，非常霸气。

对于自动部署，vagrant支持多种机制：可以使用puppet，chef或者用于在vagrant配置文件中定义的所有虚拟机上自动安装软件程序和配置的shell脚本等。

Install

1. VirtualBox安装
2. Vagrant安装
3. Vagrant配置
4. 下载VagrantBox
5. 添加box

$ vagrant box add base 远端的box地址或者本地的box文件名

base是box的名称，可以是任意的字符串，base是默认名称，主要用来标识一下你添加的box

box中的镜像文件被放到了：/Users/astaxie/.vagrant.d/boxes/，如果在window系统中应该是放到了： C:\Users\当前用户名\.vagrant.d\boxes\目录下。

box是一个配置好的基础虚拟机映像，Vagrant使用盒子来创建开发环境，支持VirtualBox、VMware、KVM等虚拟机。

$vagrant box list #显示当前已经添加的box列表

单机单虚拟机

1. 初始化

$ vagrant init

如果你添加的box名称不是base，那么需要在初始化的时候指定名称vagrant init haha。实质是在当前目录生成一个 Vagrantfile的文件，里面有很多配置信息

1. 启动虚拟机

$vagrant up

1. 连接到虚拟机

$vagrant ssh

这样我们就可以像连接到一台服务器一样进行操作了。

打包分发

$ vagrant package

打包完成后会在当前目录生成一个 package.box 的文件，将这个文件传给其他用户，其他用户只要添加这个 box 并用其初始化自己的开发目录就能得到一个一模一样的开发环境了。

部署自定义虚拟主机

1) 在待部署的主机上安装 Vagrant 软件环境

2) 将打包 xxx.box 映像复制到待部署主机上

3) 在待部署的主机上使用 Vagrant 添加 xxxx.box

4) 执行 vagrant up启动虚拟主机（部署完成）

$vagrant box list #显示当前已经添加的box列表

$vagrant box remove #删除相应的box

$vagrant destroy #停止当前正在运行的虚拟机并销毁所有创建的资源

$vagrant halt #关机

$vagrant package #打包命令，可以把当前的运行的虚拟机环境进行打包

$vagrant reload #重新启动虚拟机，主要用于重新载入配置文件

$vagrant resume #恢复前面被挂起的状态

$vagrant ssh-config #输出用于ssh连接的一些信息

$vagrant status #获取当前虚拟机的状态

$vagrant suspend #挂起当前的虚拟机

Vagrantfile配置文件

配置这个虚拟主机网络连接方式，端口转发，同步文件夹，以及怎么和puppet，chef结合的一个配置文件

* VM的名称

config.vm.box = "base"

* 网络设置

1. host-only(主机模式)，意思是主机和虚拟机之间的网络互访

config.vm.network :forwarded\_port, guest: 80, host: 8080

虚拟机80端口映射到物理机8080端口

1. Bridge(桥接模式)，该模式下的VM就像是局域网中的一台独立的主机，也就是说需要VM到你的路由器要IP，这样的话局域网里面其他机器就可以访问它了

桥接网络（公共网络，局域网DHCP服务器自动分配IP）

config.vm.network :public\_network

1. 私有网络：允许多个虚拟机通过主机通过网络互相通信，vagrant允许用户分配一个静态IP，然后使用私有网络设置。

config.vm.network :private\_network, ip: "192.168.50.4"

* hostname设置

config.vm.hostname = "go-app"

* 同步目录

/vagrant目录默认就是当前的开发目录，这是在虚拟机开启的时候默认挂载同步的。我们还可以通过配置来设置额外的同步目录：

默认的，vagrant将共享你的工作目录（即Vagrantfile所在的目录）到虚拟机中的/vagrant

config.vm.synced\_folder "src/", "/srv/website"

"src/"：物理机目录;"/srv/website"虚拟机目录

* vagrant和puppet

config.vm.provision :puppet #这里没有配置pp文件等的路径，全部采用默认

end

默认配置的目录结构：

Vagrantfile

manifests

default.pp

单机多虚拟机 （分布式集群系统）

* 快速建立产品网络的多机器环境，例如web服务器、db服务器
* 建立一个分布式系统，学习他们是如何交互的
* 测试API和其他组件的通信
* 容灾模拟，网络断网、机器死机、连接超时等情况

Vagrantfile

Vagrant.configure("2") do |config|

config.vm.define :web do |web|

web.vm.provider "virtualbox" do |v|

v.customize ["modifyvm", :id, "--name", "web", "--memory", "512"]

end

web.vm.box = "vagrant-ubuntu14"

web.vm.hostname = "web"

web.vm.network :private\_network, ip: "11.11.1.1"

end

config.vm.define :db do |db|

db.vm.provider "virtualbox" do |v|

v.customize ["modifyvm", :id, "--name", "db", "--memory", "512"]

end

db.vm.box = "vagrant-ubuntu14"

db.vm.hostname = "db"

db.vm.network :private\_network, ip: "11.11.1.2"

end

end

$vagrant up

vagrant up启动后，stdout会打印ip and 端口号

登录虚机

双击Putty.exe

登录虚机web 127.0.0.1 2222

登录虚机db 127.0.0.1 2200

虚拟机之间通信

vagrant@web:~$ ssh 11.11.1.2

vagrant@db:~$

通常情况下Box只做最基本的设置，而不是设置好所有的环境，因此Vagrant通常使用Chef或者Puppet来做进一步的环境搭建

Puppet

将基础设施作为代码进行保存与版本化

无论是在笔记本上的开发环境，还是在生产环境上，开发人员和运维人员都能够使用相同的清单对系统进行管理

将配置作为代码处理，系统管理员就能够为开发人员提供独占的测试环境

甚至可以将Puppet代码交付给审记，如今有许多审记都接收Puppet清单，以进行一致性验证

能够将Puppet代码签入到某个共享的版本控制工具中，这将为你的基础设施提供一个可控的历史记录。

Puppet 是一种 Linux、Unix、Windows 平台的集中配置管理系统，使用自有的 Puppet 描述语言，可管理配置文件、用户、cron任务、软件包、系统服务等。Puppet把这些系统实体称之为资源，其设计目标是简化对这些资源的管理以及妥善处理资源之间的依赖关系。

若为vagrant设置代理

export http\_proxy="http://user:password@host:port"

vagrant plugin install vagrant-proxyconf

export VAGRANT\_HTTP\_PROXY="http://user:password@host:port"

vagrant up

VAGRANT\_HTTP\_PROXY设置可以由Vagrantfile里的proxy替代

Vagrantfile文件

Vagrant.configure("2") do |config|

config.proxy.http = "http://165.225.96.34:10015"

config.proxy.https = "http://165.225.96.34:10015"

config.proxy.no\_proxy = "localhost,127.0.0.1"

config.vm.provision :puppet

config.vm.network :public\_network

config.vm.network :forwarded\_port, guest:80, host:8080

config.vm.define :web do |web|

web.vm.provider "virtualbox" do |v|

v.customize ["modifyvm", :id, "--name", "web", "--memory", "512"]

end

web.vm.box = "vagrant-ubuntu14"

web.vm.hostname = "web"

web.vm.network :private\_network, ip: "11.11.1.1"

end

end

Vagrantfile

manifests

default.pp

Puppet清单

# Basic Puppet Apache manifest

class apache {

exec { 'apt-get update':

command => '/usr/bin/apt-get update'

}

package { "apache2":

ensure => present,

}

service { "apache2":

ensure => running,

require => Package["apache2"],

}

file { '/var/www':

ensure => link,

target => "/vagrant",

notify => Service['apache2'],

force => true

}

}

include apache

vagrant@web:~$ ls /vagrant

$ls /var/www

以上都是显示的是Vagrantfile文件父目录的子目录和文件

已经安装好ubunbu and apache2, apache2的静态站点根目录是/var/www/html

所以本地开发目录是

vagrant

Vagranfile

manifests

defaults.pp

html

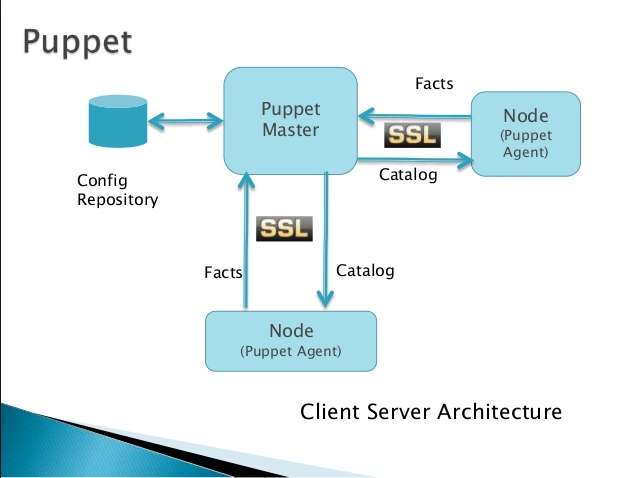
index.html

登录虚拟机$curl 127.0.0.1

或者host机浏览器: localhost:8080 （因为puppet已将虚拟机端口转发）

主机-代理部署

1. 主机puppet master集中存储所有清单，通过/etc/puppet/manifests/site.pp找到条目确定每个代理执行的Puppet代码
2. 主机将清单编译为编目并将其传输给代理执行
3. 代理向主机报告Puppet运行状态



实战

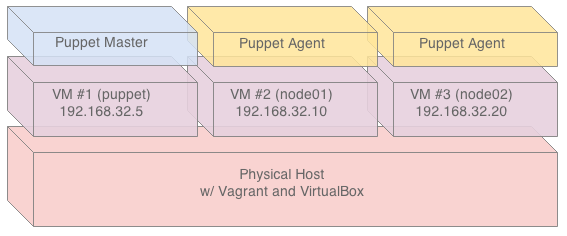
Refer:

<http://gold.xitu.io/entry/569d9cff816dfa005958feb5>

<https://programmaticponderings.wordpress.com/2014/12/14/installing-puppet-master-and-agents-on-multiple-vm-using-vagrant-and-virtualbox/>

<https://github.com/garystafford/multi-vagrant-puppet-vms>

Installing Puppet Master and Agents on Multiple VM Using Vagrant and VirtualBox



one server VM running Puppet Master, and two agent node VMs running Puppet Agent. Both agent nodes should have successfully been registered with Puppet Master, and configured themselves based on the Puppet Master’s main manifest. Agent node configuration includes installing ntp, git, Fig, and Docker.

multi-vagrant-puppet-vms

Vagrantfile

nodes.json

bootstrap-master.sh

bootstrap-node.sh

site.pp

创建三个虚拟机，以及虚拟机的安装脚本

nodes.json

{

"nodes": {

"puppet.example.com": {

":ip": "192.168.32.5",

"ports": [],

":memory": 1024,

":bootstrap": "bootstrap-master.sh"

},

"node01.example.com": {

":ip": "192.168.32.10",

"ports": [],

":memory": 1024,

":bootstrap": "bootstrap-node.sh"

},

"node02.example.com": {

":ip": "192.168.32.20",

"ports": [],

":memory": 1024,

":bootstrap": "bootstrap-node.sh"

}

}

}

Vagrantfile

# read vm and chef configurations from JSON files

nodes\_config = (JSON.parse(File.read("nodes.json")))['nodes']

VAGRANTFILE\_API\_VERSION = "2"

Vagrant.configure(VAGRANTFILE\_API\_VERSION) do |config|

config.proxy.http = "http://165.225.96.34:10015"

config.proxy.https = "http://165.225.96.34:10015"

config.proxy.no\_proxy = "localhost,127.0.0.1"

config.vm.box = "ubuntu/trusty64"

nodes\_config.each do |node|

node\_name = node[0] # name of node

node\_values = node[1] # content of node

config.vm.define node\_name do |config|

# configures all forwarding ports in JSON array

ports = node\_values['ports']

ports.each do |port|

config.vm.network :forwarded\_port,

host: port[':host'],

guest: port[':guest'],

id: port[':id']

end

config.vm.hostname = node\_name

config.vm.network :private\_network, ip: node\_values[':ip']

config.vm.provider :virtualbox do |vb|

vb.customize ["modifyvm", :id, "--memory", node\_values[':memory']]

vb.customize ["modifyvm", :id, "--name", node\_name]

end

config.vm.provision :shell, :path => node\_values[':bootstrap']

end

end

end

Bootstrapping Puppet Master Server (搭建Puppet服务器)

# Install Puppet Master （安装puppetmaster）

sudo apt-get install -yq puppetmaster

# Configure /etc/hosts file

echo "192.168.32.5 puppet.example.com puppet" | sudo tee --append /etc/hosts

echo "192.168.32.10 node01.example.com node01" | sudo tee --append /etc/hosts

echo "192.168.32.20 node02.example.com node02" | sudo tee --append /etc/hosts

# Add optional alternate DNS names to /etc/puppet/puppet.conf

sudo sed -i 's/.\*\[main\].\*/&\ndns\_alt\_names = puppet,puppet.example.com/' /etc/puppet/puppet.conf

# symlink manifest from Vagrant synced folder location

ln -s /vagrant/site.pp /etc/puppet/manifests/site.pp

Bootstrapping Puppet Agent Nodes （搭建Puppet客户端）

sudo apt-get install -yq puppet

#向Puppet master进程发起一个连接，这个进程会监听puppet server的8140端口

sudo puppet resource cron puppet-agent ensure=present user=root minute=30 \ command='/usr/bin/puppet agent --onetime --no-daemonize --splay'

sudo puppet resource service puppet ensure=running enable=true

# Configure /etc/hosts file

echo "192.168.32.5 puppet.example.com puppet" | sudo tee --append /etc/hosts

echo "192.168.32.10 node01.example.com node01" | sudo tee --append /etc/hosts

echo "192.168.32.20 node02.example.com node02" | sudo tee --append /etc/hosts

# Add agent section to /etc/puppet/puppet.conf

#编辑文件，告诉puppet代理puppet服务器的域名

echo "" && echo "[agent]\nserver=puppet" | sudo tee --append /etc/puppet/puppet.conf

sudo puppet agent --enable

登录puppet client 1: node01

登录puppet client 2: node02

//客户端创建的SSL认证请求，并将其发送给服务器, 同理node01

vagrant@node02:~$ sudo puppet agent –test –waitforcert=60

登录puppet master

//查看还未被签署的认证请求列表。

vagrant@puppet:~$sudo puppet cert –list

//签署请求

$ sudo puppet cert --sign puppetclient

Once the certificate signing process is complete, the Puppet Agent retrieves the client configuration from the Puppet Master and applies it to the local agent node. The Puppet Agent will execute all applicable steps in the site.pp manifest on the Puppet Master server, designated for that specific Puppet Agent node

site.pp manifest on the Puppet Master server, applied by Puppet Agent on the agent nodes.

site.pp

node default {

notify { "Debug output on ${hostname} node.": }

include ntp, git

}

node 'node01.example.com', 'node02.example.com' {

notify { "Debug output on ${hostname} node.": }

include ntp, git, docker, fig

}