VAGRANT 101

A VERY BRIEF INTRODUCTION

WHAT ARE WE TALKING ABOUT?

1. What is Vagrant?
2. How do we use Vagrant at Komodo?
3. A simple Vagrant setup
4. The `VagrantFile`
5. Automating VM provisioning
6. Some common commands

WHAT IS VAGRANT?

Vagrant is a cross-platform (WinNT, OSX, Linux) command line interface (CLI) for creating and managing virtual machine instances.

Vagrant can be used with several VM "providers" which provide the VM infrastructure. VirtualBox is the default provider, but more can be added via the plugin system (e.g. VMWare, AWS, Rackspace Cloud).

It gives us the vital ability to create and configure virtual machines in a consistent manner.

It is perfect for sharing server environments amongst the team, all packaged neatly in a few text files.

HOW DO WE USE VAGRANT AT KOMODO?

We primarily use Vagrant (with VirtualBox) to create and configure individual development environments for each project.

Having project-specific VMs allows us to install only those dependencies (and specific versions of those dependencies) required for a project.

These virtual environments mirror the live, production environments for each project as closely as possible.

We run our applications and tests locally within these virtual environments during development.

A SIMPLE VAGRANT SETUP

- Install VirtualBox (virtualbox.org)
- Install Vagrant (vagrantup.com)
- Setup Vagrant for a new project:

```
$ cd /path/to/my/app
$ vagrant init
```

- Edit the resulting `VagrantFile` (more on this later ...)
- Load up the virtual machine (headless):

```
$ vagrant up
```

- SSH into the machine as needed:

```
$ ssh vagrant@192.68.50.100
```

(A keypair is automatically created for the `vagrant` user and made available to all VMs created through Vagrant)

THE `VAGRANTFILE`

The `VagrantFile` is a simple Ruby script that tells Vagrant how to configure our virtual machine(s).

You can configure several virtual machines within the same `VagrantFile`. This is useful for applications that make use of a cluster of different servers.

We bundle this file with the project's Mercurial repostory so that others in the team can easily spin up an identical virtual environment on their local machine.

SOME THINGS YOU CAN CONFIGURE IN VAGRANTFILE

- The flavour of OS to install (find bare bones boxes at vagrantcloud.com or vagrantbox.es, or package your own)
- The IP address of the VM
- A shared folder between your host machine and the virtual machine
- A provisioning script or manifest files

SAMPLE VAGRANTFILE

```
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |c|

    # The base "bare bones" OS image
    c.vm.box = "ubuntu/trusty32"
    c.vm.box_url = "https://vagrantcloud.com/ubuntu/.../virtualbox.box"

    # A hostname for this machine
    c.vm.hostname = "my-vagrant-box"

    # A local IP (use private address space, 192.168.*.*)
    # The host machine will be on 192.168.50.1
    c.vm.network "private_network", ip: "192.168.50.100"

    # Setup shared folder (needed for OSX)
    c.vm.synced_folder "./",
        "/vagrant",
        id: "vagrant-root",
        owner: "vagrant",
        group: "www-data",
        mount_options: ["dmode=775", "fmode=664"]

    # A provisioning shell script (run once; more on this later)
    c.vm.provision :shell, :path => "provision.sh"
end
```

PROVISIONING

Provisioning is the term used to describe the initial, one-time setup of the virtual machine. This may include the installation of dependant software packages, creation of required project directories, configration of service daemons, etc.

Vagrant supports a few ways to provision machines, such as Ansible, Puppet and Chef (all cross platform) or a simple a shell script.

Importantly, provisioning is non-interactive - you cannot (and should not) need to interact with the process.

Here's an example of the `provision.sh` shell script that we earlier defined in our example `VagrantFile` ...

EXAMPLE PROVISIONING SHELL SCRIPT

```
# Install some packages (includes some mysql-specific hackery)
# Note that the script runs as root by default
apt-get update
debconf-set-selections <<< 'mysql-server-<version> mysql-server/root_password password root'
debconf-set-selections <<< 'mysql-server-<version> mysql-server/root_password_again password root'
apt-get install nginx mysql nodejs
cp -f /vagrant/provisioning/local/my.cnf /etc/mysql/my.cnf
service mysql restart

# Create some logging directories
mkdir /var/logs/mylogs/error.log
chown root:www-data /var/logs/mylogs/error.log
chmod 0644 /var/logs/mylogs/error.log

# Copy my upstart script from shared folder to OS directory
cp -f /vagrant/my-daemon-upstart.conf /etc/init.d/
start my-daemon-upstart

# Setup DB
echo "create database mydb" | mysql -uroot -proot
mysql -uroot -proot -e "grant all privileges on *.* to root@'%' identified by 'root'";
mysql -uroot -proot mydb < /vagrant/db-schema.sql

# Symlink website docroot to our shared folder
ln -s /var/www /vagrant/public_html
```

A great guide for setting up a secure machine from scratch:
http://plusbryan.com/my-first-5-minutes-on-a-server-or-essential-security-for-linux-servers

VAGRANT UP!

Now we'll start the virtual machine:

```
$ cd /path/to/my/app
$ vagrant up
```

(It'll take a few minutes the first time whilst the provisioning script is executed, but thereafter it'll be snappier)

And we can load our app into a browser:

```
http://192.168.50.100/
```

SOME COMMON COMMANDS

`vagrant init`
Creates a new `VagrantFile` with a few basic settings

`vagrant up <alias>`
Spins up the <alias> virtual machine or all machines and triggers provisioning if needed.

`vagrant reload --provision`
Reboots (and re-provisions) the VM.

`vagrant halt <alias>`
Shuts down the <alias> virtual machine or all machines.
(Use this rather than `sudo halt` from within the VM)

SOME COMMON COMMANDS (CONT ...)

`vagrant ssh <alias>`
Shells into the virtual machine
(or `putty -ssh vagrant@ip-address` on Windows)

`vagrant destroy <alias>`
Destroys the <alias> virtual machine instance or all machines

`vagrant help`
List all other commands

RECAP

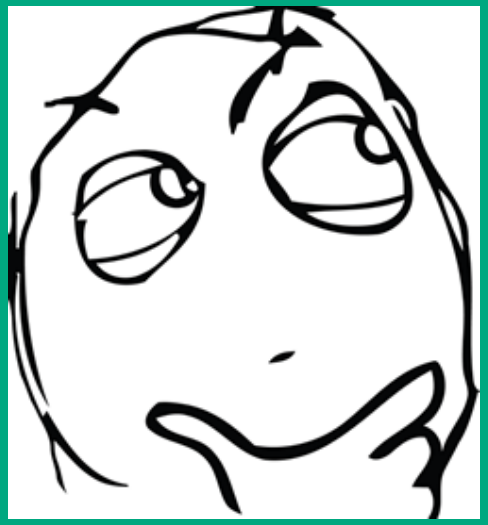
- Vagrant helps provide consistent development environments across teams.
- Using project-specific VMs helps us accurately mirror the live environment, minimizing WTFs when it comes to deployment.
- We bundle the `VagrantFile`, and any accompanying files required for provisioning, with the project repository, so others in the team simply need to run:

```
$ hg pull -u; vagrant up
```

- Running tests within the VMs ensures tests are carried out in an identical environment. Again minimizing WTF moments.

THE END

ANY QUESTIONS?



(Let me google that for you ...)

Slides available in `https://github.com/KomodoHQ/Talks-Vagrant101` repo.