
OS X templates for Packer and VeeWee

This is a set of Packer templates and support scripts that will prepare an OS X installer media that performs an unattended install for use with Packer and VeeWee. These were originally developed for VeeWee, but support for the VeeWee template has not been maintained since Packer's release and so it is only provided for historical purposes. I plan on removing VeeWee support from this repo soon, but VeeWee can still make use of the preparation script and the OS X template remains in the core VeeWee repo.

The machine built by this Packer template defaults to being configured for use with Vagrant, and supports three Vagrant providers by using Packer's respective builders:

- The Hashicorp VMware Fusion provider (recommended)
- Vagrant's included VirtualBox provider
- Parallels

It's possible to build a machine with different admin account settings, and without the vagrant ssh keys, for use with other systems, e.g. continuous integration.

Use with the Fusion provider requires Vagrant 1.3.0, and use with the VirtualBox provider Vagrant 1.6.3 if using the Rsync file sync mechanism. Note that the VeeWee template also does not have any VirtualBox or Parallels support.

Provisioning steps that are defined in the template via items in the scripts directory: - Vagrant-specific configuration - VM guest tools installation if on VMware - Xcode CLI tools installation - Chef installation via the Chef Omnitruck method - Puppet installation via Puppetlabs Mac installers, both legacy and 4 - Disk shrinking for VMware

Supported guest OS versions

Currently this prepare script and template supports all versions of OS X that are distributed through the App Store: OS X Lion (10.7) through El Capitan (10.11), and macOS Sierra (10.12).

This project currently only supplies a single Packer template (`template.json`), so the hypervisor's configured guest OS version (i.e. `darwin12-64`) does not accurately reflect the actual installed OS. I haven't found there to be any functional differences depending on these configured guest versions.

To build a VMware box of an OS version less than El Capitan (10.11), note that as of VMare Fusion 8.5.4, you will need to change the `tools_upload_flavor` from `darwin` to `darwinPre15`.

Issue with 10.12.4

Important note: The Sierra 10.12.4 installer seems to no longer support including custom packages as part of the installer, unless they are signed by Apple. This produces an error with the text, “macOS could not be installed on your computer.. The package veewee-config.pkg is not signed.”

The `prepare_iso.sh` script in this repo makes use of functionality Apple supports as part of a NetInstall workflow, but because of this (undocumented) additional requirement of additional packages needing to be signed by Apple as of 10.12.4, these tools can’t currently install the necessary configuration for Packer to log in to perform additional configuration, installing guest tools, etc. The rest of the OS install still completes successfully.

It may be possible to work around this by modifying the rc script directly with the contents of our postinstall script.

The `prepare_vdi.sh` script uses AutoDMG’s approach running the installer’s `OSInstall.pkg` creating a fresh install in a temporary DMG sparse disk image which is converted into a VDI disk image using VirtualBox’s command line tools.

Preparing the ISO

OS X’s installer cannot be bootstrapped as easily as can Linux or Windows, and so exists the `prepare_iso.sh` script to perform modifications to it that will allow for an automated install and ultimately allow Packer and later, Vagrant, to have SSH access.

Note: VirtualBox users currently have to disable Remote Management to avoid periodic freezing of the VM by adding `-D DISABLE_REMOTE_MANAGEMENT` to the `prepare_iso.sh` options. See Remote Management freezing issue for more information.

Run the `prepare_iso.sh` script with two arguments: the path to an `Install OS X.app` or the `InstallESD.dmg` contained within, and an output directory. Root privileges are required in order to write a new DMG with the correct file ownerships. For example, with a 10.8.4 Mountain Lion installer:

```
sudo prepare_iso/prepare_iso.sh "/Applications/Install OS X Mountain Lion.app"out
```

...should output progress information ending in something this:

```
1 -- MD5: dc93ded64396574897a5f41d6dd7066c
2 -- Done. Built image is located at out/OSX_InstallESD_10.8.4_12E55.dmg.
   Add this iso and its checksum to your template.
```

`prepare_iso.sh` accepts command line switches to modify the details of the admin user installed by the script.

- `-u` modifies the name of the admin account, defaulting to `vagrant`
- `-p` modifies the password of the same account, defaulting to `vagrant`
- `-i` sets the path of the account's avatar image, defaulting to `prepare_iso/support/vagrant.jpg`

For example:

```
sudo prepare_iso/prepare_iso.sh -u admin -p password -i /path/to/
image.jpg "/Applications/Install OS X Mountain Lion.app"out
```

Additionally, flags can be set to disable certain default configuration options.

- `-D DISABLE_REMOTE_MANAGEMENT` disables the Remote Management service.
- `-D DISABLE_SCREEN_SHARING` disables the Screen Sharing service.

Clone this repository The `prepare_iso.sh` script needs the `support` directory and its content. In other words, the easiest way to run the script is after cloning this repository.

Use with Packer

The path can now be added to your Packer template or provided as user variables. The `packer` directory contains a template that can be used with the `vmware-iso` and `virtualbox-iso` builders. The checksum does not need to be added because the `iso_checksum_type` has been set to “none”. The `veewee` directory contains a definition, though as mentioned above it is not currently being maintained.

The Packer template adds some additional VM options required for OS X guests. Note that the paths given in the Packer template's `iso_url` builder key accepts file paths, both absolute and relative (to the current working directory).

Given the above output, we could run then run packer:

```
1 cd packer
2 packer build \
3   -var iso_url=../out/OSX_InstallESD_10.8.4_12E55.dmg \
4   template.json
```

You might also use the `-only` option to restrict to either the `vmware-iso` or `virtualbox-iso` builders.

If you modified the name or password of the admin account in the `prepare_iso` stage, you'll need to pass in the modified details as packer variables. You can also prevent the vagrant SSH keys from being installed for that user.

For example:

```
1 packer build \  
2   -var iso_url=../out/OSX_InstallESD_10.8.4_12E55.dmg \  
3   -var username=youruser \  
4   -var password=yourpassword \  
5   -var install_vagrant_keys=false \  
6   template.json
```

Building to a device with more space

Local VM builds take up a lot of space. It's possible to make packer work in different directories.

- `PACKER_CACHE_DIR` is an out-of-the-box environment variable that configures where it will cache ISOs etc.
- `PACKER_OUTPUT_DIR`: configure where packer will build artifacts (like OVF files) to
- `PACKER_VAGRANT_BOX_DIR`: configure where packer will build vagrant boxes via the post-processor to.

Note: don't make `PACKER_OUTPUT_DIR` and `PACKER_VAGRANT_BOX_DIR` the same place. `keep_input_artifacts` in the post-processor defaults to `false`, and it removes them by removing the directory, not the individual files. So if you use the same place, you'll end up with no output at all (packer `v1.0.0`).

Automated installs on OS X

OS X's installer supports a kind of bootstrap install functionality similar to Linux and Windows, however it must be invoked using pre-existing files placed on the booted installation media. This approach is roughly equivalent to that used by Apple's System Image Utility for deploying automated OS X installations and image restoration.

The `prepare_iso.sh` script in this repo takes care of mounting and modifying a vanilla OS X installer downloaded from the Mac App Store. The resulting `.dmg` file can then be added to the Packer template. Because the preparation is done up front, no boot command sequences, attached devices or web server access is required.

More details as to the modifications to the installer media are provided in the comments of the script.

Automated GUI logins

For some kinds of automated tasks, it may be necessary to have an active GUI login session (for example, test suites requiring a GUI, or Jenkins SSH slaves requiring a window server for their tasks). The Packer templates support enabling this automatically by using the `autologin` user variable, which can be set to 1 or **true**, for example:

```
packer build -var autologin=true template.json
```

This was easily made possible thanks to Per Olofsson's CreateUserPkg utility, which was used to help create the box's vagrant user in the `prepare_iso` script, and which also supports generating the magic `kcpassword` file with a particular hash format to set up the auto-login.

Configuration management

By default, the packer template does not install the Chef or Puppet configuration management tools. You can enable the installation of configuration management by setting the `chef_version`, `puppet_agent_version`, `puppet_version`, `facter_version`, and `hieraproject_version` variables to `latest`, or to a specific version.

To install the latest version of Chef:

```
1 packer build -var chef_version=latest template.json
```

To install the last version of Puppet Agent:

```
1 packer build -var puppet_agent_version=latest template.json
```

To install the last versions of the deprecated standalone Puppet, Facter and Hieraproject packages:

```
1 packer build -var puppet_version=latest facter_version=latest
  hieraproject_version=latest template.json
```

Xcode Command Line Tools

The Xcode CLI tools are installed by the packer template by default. To disable the installation, set the `install_xcode_cli_tools` variable to **false**:

```
1 packer build -var install_xcode_cli_tools=false template.json
```

System updates

Packer will instruct the system to download and install all available OS X updates, if you want to disable this default behaviour, use `update_system` variable:

```
1 packer build -var update_system=0 template.json
```

Provisioning delay

In some cases, it may be helpful to insert a delay into the beginning of the provisioning process. Adding a delay of about 30 seconds may help subsequent provisioning steps that install software from the internet complete successfully. By default, the delay is set to 0, but you can change the delay by setting the `provisioning_delay` variable:

```
1 packer build -var provisioning_delay=30 template.json`
```

VirtualBox support

VirtualBox support is thanks entirely to contributions by Matt Behrens (@zigg) to this repo, Vagrant and Packer.

Caveats

Remote Management freezing issue The default `prepare_iso.sh` configuration enables Remote Management during installation, which causes the resulting virtual machine to periodically freeze. You can avoid enabling Remote Management when using `prepare_iso.sh` by passing `-D DISABLE_REMOTE_MANAGEMENT` this:

```
1 sudo ./prepare_iso/prepare_iso.sh -D DISABLE_REMOTE_MANAGEMENT "/Applications/Install OS X El Capitan.app" out
```

Shared folders Oracle's support for OS X in VirtualBox is very limited, including the lack of guest tools to provide a shared folder mechanism. If using the VirtualBox provider in Vagrant, you will need to configure the shared folder that's set up by default (current folder mapped to `/vagrant`) to use either the `rsync` or `nfs` synced folder mechanisms. You can do this like any other synced folder config in your Vagrantfile:

```
1 Vagrant.configure("2") do |config|
2   config.vm.provider "virtualbox" do |vb|
```

```
3     config.vm.synced_folder ".", "/vagrant", type: "rsync"
4   end
5 end
```

Alternative method using `prepare_vdi.sh`, `prepare_ovf.sh` and `packer virtualbox-ovf`

This approach requires VirtualBox and unfortunately almost 30GB of free space.

Installing and exporting to a VirtualBox virtual disk image. The `prepare_vdi.sh` command will run the installer's `OSInstall.pkg` creating a fresh install in a temporary disk image which is converted into a VDI disk image.

```
1 cd packer
2 sudo ../prepare_iso/prepare_vdi.sh \
3   -D DISABLE_REMOTE_MANAGEMENT \
4   -o macOS_10.12.vdi \
5   /Applications/Install\ macOS\ Sierra.app/ \
6   .
```

Generating a VirtualBox machine and exporting it into packed virtual machine OVF The `prepare_ovf.sh` command takes a virtual image disk and adds it into a temporary VirtualBox machine which is exported into a packed virtual machine using the OVF (Open Virtualization Format).

```
1 ../prepare_iso/prepare_ovf.sh \
2   macOS_10.12.vdi
```

Generating a packer box using the `virtualbox-ovf` builder Finally the `virtualbox-ovf` allows to use the previously generated exported virtual machine to generate the provisioned packer box.

```
1 packer build \
2   -var provisioning_delay=30 \
3   -var source_path=macOS_10.12.ovf \
4   template.json
```

Box sizes

A built box with CLI tools, Puppet and Chef is over 5GB in size. It might be advisable to remove (with care) some unwanted applications in an additional postinstall script. It should also be possible to

modify the OS X installer package to install fewer components, but this is non-trivial. One can also supply a custom “choice changes XML” file to modify the installer choices in a supported way, but from my testing, this only allows removing several auxiliary packages that make up no more than 6-8% of the installed footprint (for example, multilingual voices and dictionary files).

Alternate approaches to VM provisioning

Joe Chilcote has written a tool, `vfuse`, which converts a never-booted OS X image (such as created with a tool like AutoDMG) into a VMDK and configures a VMware Fusion VM. `vfuse` can also configure a Packer template alongside the VM, configured with the `vmware-vmx` builder.