Dots and Brackets: Code Blog

Blog about DevOps, distributed applications and microservices

Building VM image with Packer

Quite often building a VM from scratch is not very wise. Unless server configuration is trivial, its provisioning might take significant amount of time. For example, creating an instance of a build server for my current project takes about 40 minutes. This includes installing updates, various SDKs and other dependencies. How is it possible then that I can add new build server to a cluster in about two or three minutes?

The secret is that the most of the software is baked into a VM image, so I never start from scratch. New VM still needs some steps like final configuration and registering within the cluster, but that's fast. The slowest part is allocating resources for VM, not provisioning.

Historically, we've been using own tools for that, but there're also free and open source ones. Like Packer.

What's Packer

Packer is virtual machine image builder. It can create images for most of big cloud providers, like AWS, GCE, Azure and Digital Ocean, or even local hypervisors: VMWare, VirtualBox and few others. It also doesn't discriminate and works well with both Linux and Windows images.

Packer needs something to build an image from and that something is template file. Template is regular JSON file with three main sections:

- builders,
- provisioners and
- post-processors

```
{
  "builders": [
  ],
  "provisioners": [
  ],
  "post-processors": [
  ]
}
```

While “builders” are responsible for creating virtual machine for future image and therefore vary from one cloud provider to another, “provisioners” are atomic configuration steps which are usually reusable. “post-processors” on the other hand are the final touches, like compressing output image or turning it into Vagrant box. Latter two sections are optional, so having just a builder is enough to create an image.

The plan

As I tend to reuse examples from one article to the other, let’s use something from Consul cluster we created before and turn it into a Packer image. I think creating Ubuntu image with Consul binaries baked into it would make some sense as an example.

We’ll use VirtualBox as a destination for image. It’s not that exciting like building an image in AWS or GCE, but it’s radically cheaper.

Installation

Packer is downloadable as an archive with binaries and unzipping concludes the installation. Installing VirtualBox involves more clicking, but it’s still trivial. Both tools are cross-platform and I’m using their Mac-compatible incarnations.

Setting up the “builder”

The worst part of using VirtualBox as Packer builder is that it's more complex than all other cloud providers combined. The reason is simple: whereas we usually start creating cloud virtual machines from more or less ready VM image, VirtualBox VM starts right from operating system ISO file. Fortunately, mighty Google has enough of links to Packer/VirtualBox examples and this particular guy has templates for virtually everything.

VirtualBox Ubuntu builder needs two things:

- **builder** section itself, which will describe what provider it uses (VirtualBox), where to pick OS ISO from, where to click during GUI installation and few virtual machine settings;
- `preseed.cfg` file, which contains Ubuntu installation options. I just picked the first one found on internet.

This is the initial version of Packer template file with builder only:

```
{
    "builders": [{
        "type": "virtualbox-iso",
        "guest_os_type": "Ubuntu_64",
        "iso_url": "http://releases.ubuntu.com/16.04/ubuntu-16.04.2-server-amd64.isc
        "iso_checksum": "737ae7041212c628de5751d15c3016058b0e833fdc32e7420209b76ca3c
        "iso_checksum_type": "sha256",
        "output_directory": "output-ubuntu-consul",
        "vm_name": "ubuntu-consul-16.04-amd64",
        "disk_size": 15000,
        "headless": "true",
        "http_directory": "http",
        "boot_wait": "5s",
        "boot_command": [
            "<enter><wait>",
            "<f6><esc>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
            "<bs><bs><bs>",
            "/install/vmlinuz ",
            "initrd=/install/initrd.gz ",
            "net.ifnames=0 ",
            "auto-install/enable=true ",
            "debconf/priority=critical ",
            "preseed/url=http://{{.HTTPIP}}:{{.HTTPPort}}/preseed.cfg",
            "<enter>"
        ],
        "ssh_timeout": "60m",
        "ssh_username": "ubuntu",
        "ssh_password": "ubuntu",
        "shutdown_command": "sudo systemctl poweroff",
        "vboxmanage": [
            ["modifyvm", "{{.Name}}", "--memory", 512],
            ["modifyvm", "{{.Name}}", "--cpus", 1]
        ]
    }],
    "provisioners": [
    ]
}
```

Even though it looks cumbersome, it actually makes a lot of sense. `type: virtualbox-iso` means we're using VirtualBox provider. `iso_*` properties define where OS ISO comes from. `vboxmanage` define VirtualBox VM hardware settings and `boot_command` actually specifies what keys to press when OS installer starts. I'd never memorized this command, but with the help of mighty Google I don't have to.

`"http_directory": "http"` and `preseed/url=http://{{.HTTPIP}}{{.HTTPPort}}/preseed.cfg` are particularly interesting ones. In order to provide Ubuntu installer with default options (`preseed/url`) Packer starts its own embedded web server. However, it doesn't come with a content to serve, so `http_directory` has to specify the place to serve the files from. I have local `http` directory with `preseed.cfg` in it (thanks again, Google!), so Ubuntu installation will go in totally unattended mode. You can take a look at this file at github.

Configuring "provisioners"

Dealing with provisioners is much simpler. Packer works well with configuration managers like Ansible or Chef, but in our case simple "shell" provisioner will be enough:

```
{
    "builders": [/*...*/]
    "provisioners": [{
        "type": "shell",
        "scripts": [
            "install-consul.sh"
        ]
    }]
}
```

I picked the code for `install-consul.sh` from my previous post about Vagrant and Consul and changed it just a little bit:

```
#!/bin/bash
```

```

dpkg -s unzip &>/dev/null || {
    # update and unzip
    sudo apt-get -y update && sudo apt-get install -y unzip
}


# install consul
if [ ! -f ~/consul ]; then
    cd ~
    version='0.8.0'
    wget https://releases.hashicorp.com/consul/${version}/consul_${version}_linux_am
    unzip consul.zip
    rm consul.zip

    # make consul executable
    chmod +x consul
fi
```

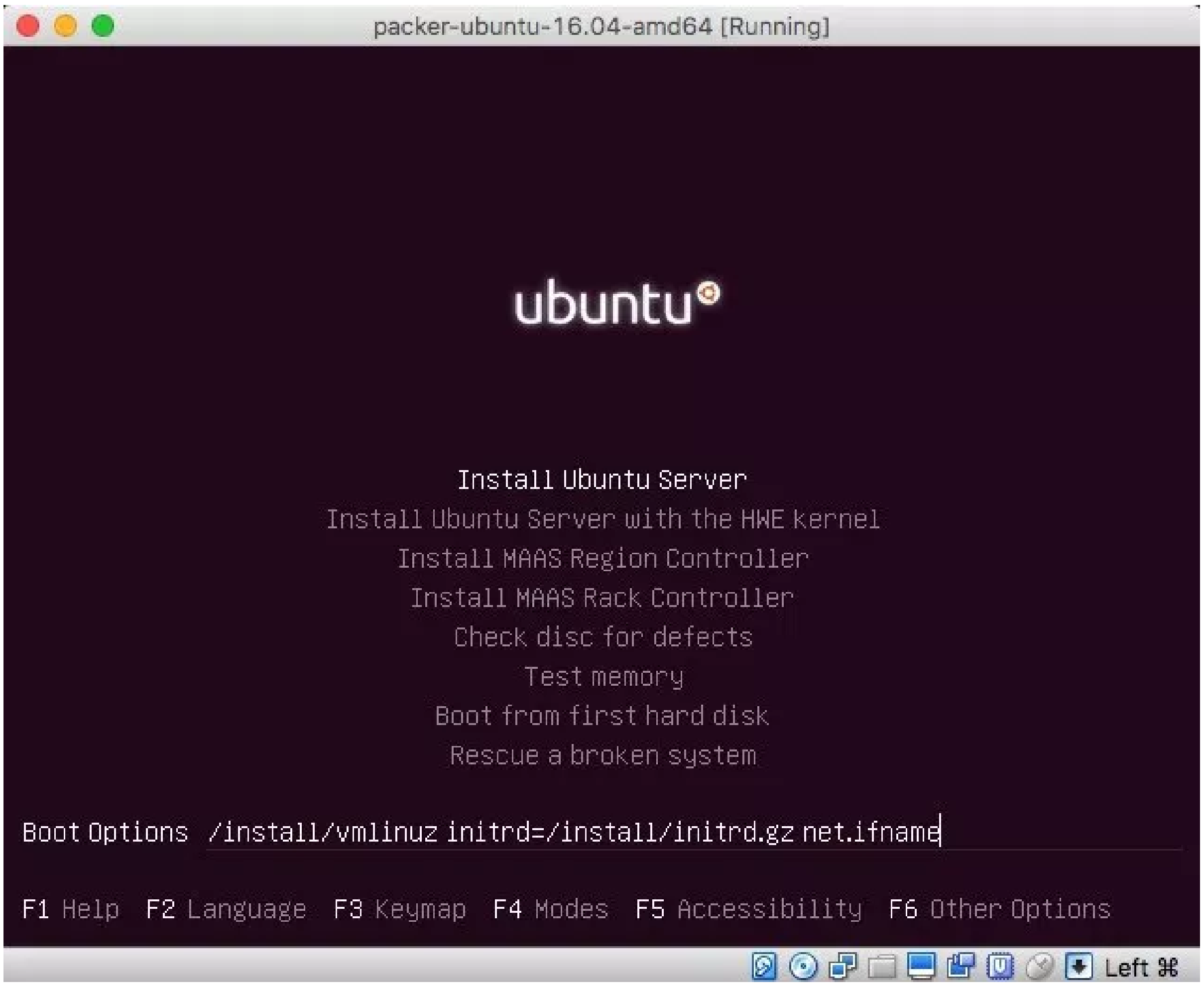
And that's it!

Test run

Now let's validate template file we've just created and then try to build it.

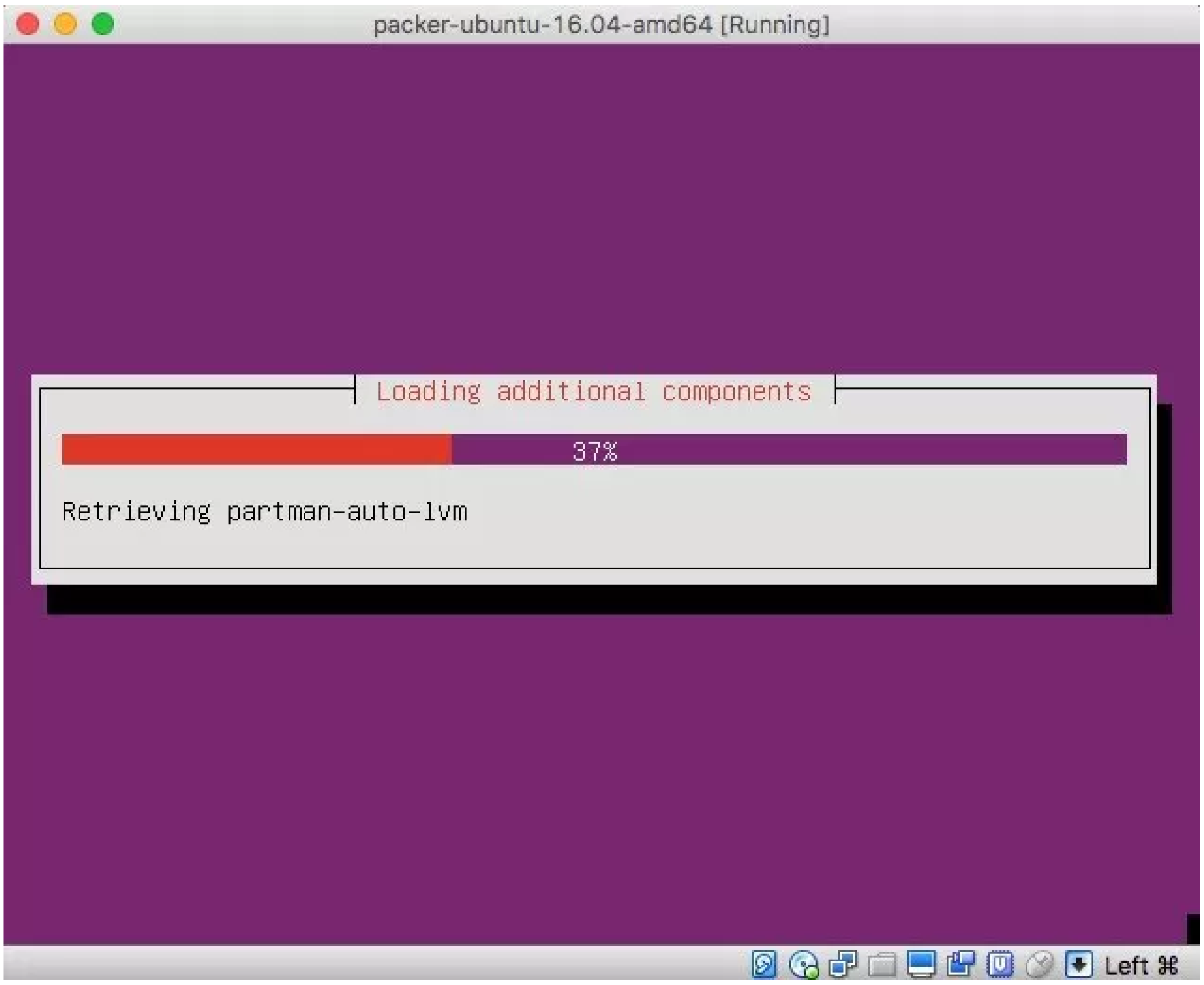
```
$ ./packer validate ubuntu-consul-template.json
# Template validated successfully.

$ ./packer build ubuntu-consul-template.json
#...
#     virtualbox-iso: Downloading or copying: http://releases.ubuntu.com/16.04/ubunt
#...
# ==> virtualbox-iso: Starting the virtual machine...
#...
#     virtualbox-iso: Saving to: 'consul.zip'
#...
#     virtualbox-iso: Executing: export packer-virtualbox-iso-1494908412 --output ou
#...
# --> virtualbox-iso: VM files in directory: output-ubuntu-consul
```

Along the way you actually can see how the installation goes. Remember that ridiculously long `<bs><bs>...` sequence in the beginning? It's used right here, for `backspace` 'ing boot options:


packer-ubuntu-16.04-amd64 [Running]
ubuntu
Install Ubuntu Server
Install Ubuntu Server with the HWE kernel
Install MAAS Region Controller
Install MAAS Rack Controller
Check disc for defects
Test memory
Boot from first hard disk
Rescue a broken system
F1 Help F2 Language F3 Keymap F4 Modes F5 Accessibility F6 Other Options
Left ⌘

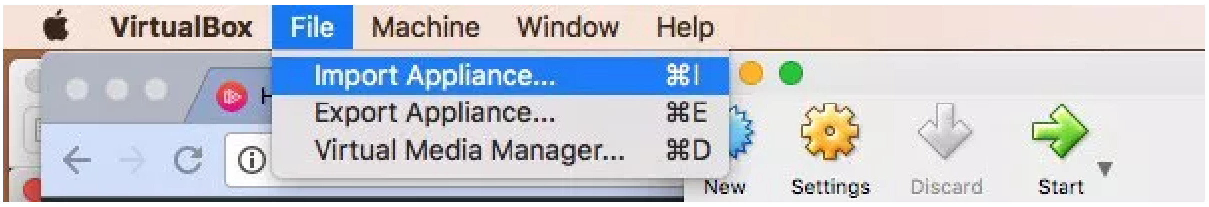
```
Boot Options /install/vmlinuz initrd=/install/initrd.gz net.ifname
```

And the rest of the process:

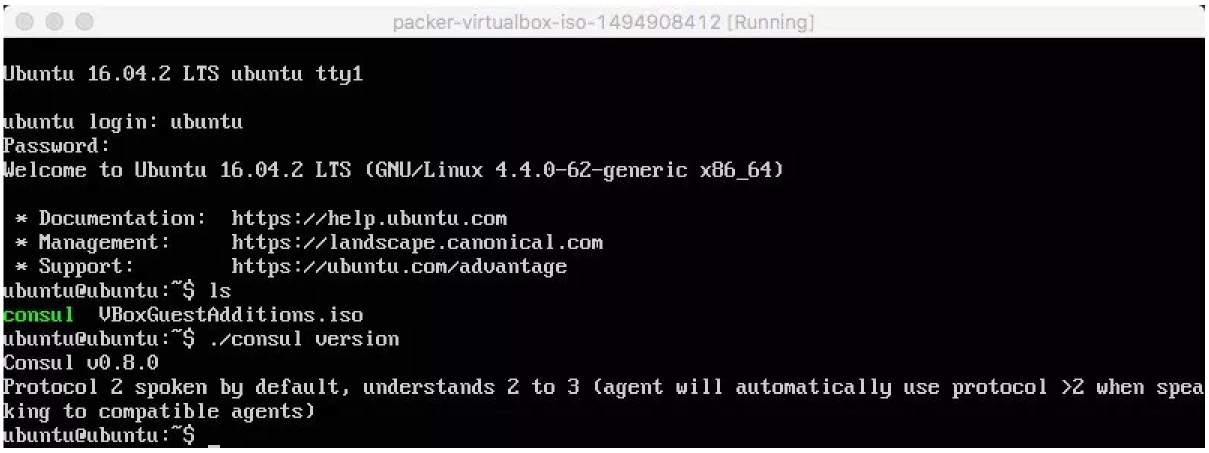


```
Loading additional components
37%
Retrieving partman-auto-lvm
```

And the whole process is fast. Like, few minutes fast. In the end it produces VMDK and OVF files, which are VM image equivalent in VirtualBox. We can import OVM as a new VM:



Then start the VM (login and password – `ubuntu` and `ubuntu`) and check if `consul` is there:



```
Ubuntu 16.04.2 LTS ubuntu tty1

ubuntu login: ubuntu
Password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-62-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
ubuntu@ubuntu:~$ ls
consul  VBoxGuestAdditions.iso
ubuntu@ubuntu:~$ ./consul version
Consul v0.8.0
Protocol 2 spoken by default, understands 2 to 3 (agent will automatically use protocol >2 when spea
king to compatible agents)
ubuntu@ubuntu:~$ _
```

Yup, it's definitely there.

Conclusion

If you're creating one-of-a-kind VM, it's probably OK to create it from scratch in one go (automatically, obviously). But if you have a cluster of VMs that come and go, it makes sense to prepare an image that has most of configuration and software installed, and then use it as a base for new VMs. Packer is a tool exactly for that – for creating preconfigured images.

The best selling feature for me is that I can create images in different cloud providers from single configuration. I like Google Compute Engine a lot, but it's nice to know that at any moment I can make few changes in "builders" and move the whole cluster to AWS.

As usual, the source code for this article can be found on github.

Share this:

pav / May 17, 2017 / Development / consul, packer, quick guide, ubuntu, virtualization

4 thoughts on "Building VM image with Packer"

EV

May 17, 2017 at 5:19 am

Well, it stuck at "Waiting for SSH to become available" for me. VirtualBox shows that the VM is stopped. And I've got some strange errors when tried to start it from UI.

pav

May 18, 2017 at 12:03 am

I need more details than that 🙂 What OS are you using? I just tried the sample one more time on another machine, Ubuntu 14.04. All worked.

LCE

October 8, 2018 at 6:17 pm

I'm having the same issue as EV using Mac High Sierra as the host OS. Has this only worked with Linux-based host operating systems?

pav

October 8, 2018 at 6:55 pm

I successfully tried this on both Mac and Ubuntu machines. Never tried it on Windows, though.