PACKER.
TO BE OR NOT TO BE?

About me

Iurii Medvedev

EPAM Systems,
Senior Systems Engineer

- Devops lead with more than 8 years of experience in system administration which includes
- Media servers
- Load balancing technologies and cloud technologies-
- Management and automation

Agenda.

- Pre-baked machine images
- What is it Packer? Advantages Of Using Packer
- Support Platforms and Packer Workflow
- Local build vs cloud's build
- Examples
- Q&A

Pre-baked machine images.

- What is pre-backer images?

- What problem do you can have with pre-backer?

Why use Packer?

- Packer is a tool for creating machine and container images for multiple platforms from a single source configuration.
- A machine image is a single static unit that contains a pre-configured operating system and installed software which is used to quickly create new running machines.
- Packer only builds images. It does not attempt to manage them in any way. After they're built, it is up to you to launch or destroy them

Advantages Of Using Packer

- **Super fast infrastructure deployment** - Packer images allow you to launch completely provisioned and configured machines in seconds. This benefits not only production, but development as well.

Advantages Of Using Packer

- **Multi-provider portability** - Packer creates identical images for multiple platforms, you can run production in AWS, staging/QA in a private cloud like OpenStack, and development in desktop virtualization solutions such as VMware or VirtualBox. Each environment is running an identical machine image, giving ultimate portability.

Advantages Of Using Packer

- **Improved stability** - Packer installs and configures all the software for a machine at the time the image is built. If there are bugs in these scripts, they'll be caught early, rather later when a machine is launched.

Advantages Of Using Packer

- **Greater testability** - After a machine image is built, that machine image can be quickly launched and smoke tested to verify that things appear to be working. If they are, you can be confident that any other machines launched from that image will function properly.

Use cases

- **Continuous Delivery** - Add Packer in the middle of your continuous delivery pipeline. It can be used to generate new machine images for multiple platforms on every change to Chef/Puppet. As part of this pipeline, the newly created images can then be launched and tested, verifying the infrastructure changes work. If the tests pass, you can be confident that the image will work when deployed. This brings a new level of stability and testability to infrastructure changes.

Use cases

- **Dev/Prod Parity** - Packer helps keep development, staging, and production as similar as possible. Packer can be used to generate images for multiple platforms at the same time. So if you use AWS for production and Docker for development, you can generate both an AMI and a VMware machine using Packer at the same time from the same template. Mix this in with the continuous delivery use case above, and you have a pretty slick system for consistent work environments from development all the way through to production.

Use cases

- **Appliance/Demo Creation** - Since Packer creates consistent images for multiple platforms in parallel, it is perfect for creating appliances and disposable product demos. As your software changes, you can automatically create appliances with the software pre-installed. Potential users can then get started with your software by deploying it to the environment of their choice. Packaging up software with complex requirements has never been so easy.

Supported Platforms

- Amazon EC2 (AMI)
- CloudStack
- OpenStack
- DigitalOcean
- Docker
- Google Compute Engine
- Parallels

Supported Provisioners

- Ansible Local
- Ansible Remote
- Chef Client
- ChefSolo
- Converge
- File
- PowerShell
- Puppet Masterless

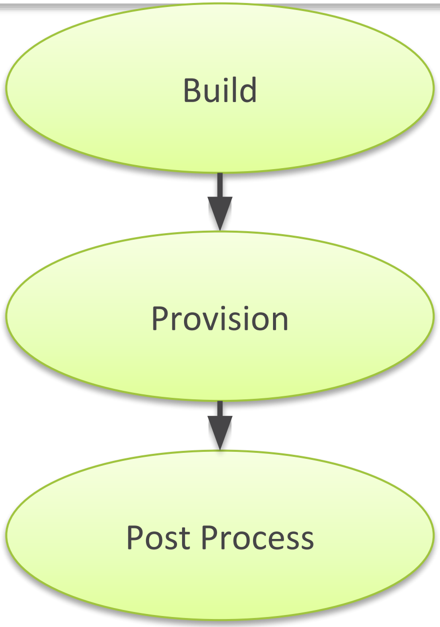
Supported Post-Processors

- Amazon Import
- Artifice
- Atlas
- Compress
- Checksum
- Docker Import
- Docker Push
- Docker Save
- Docker Tag

Terminology

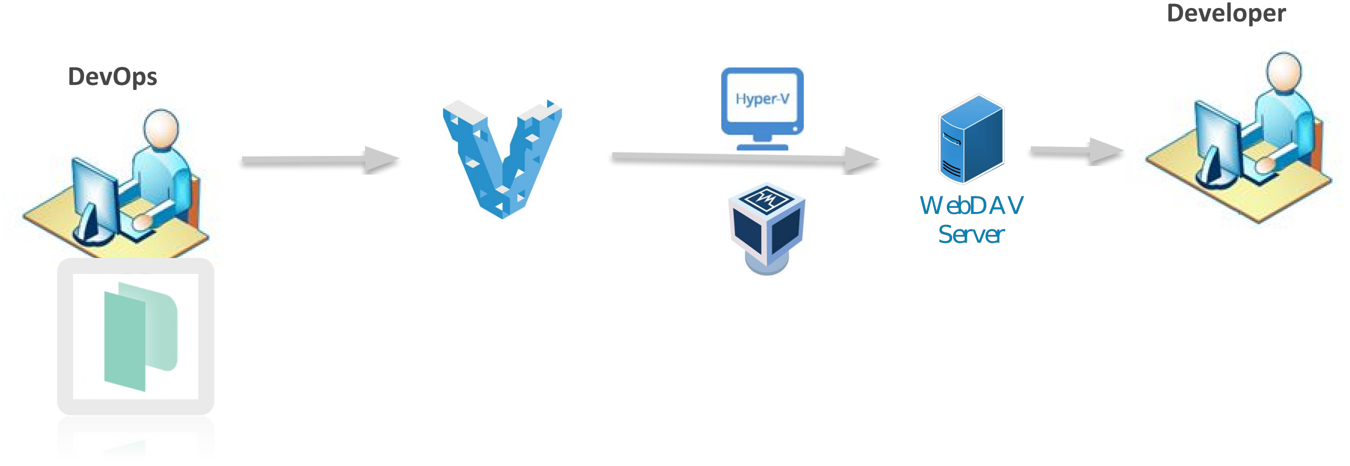
- **Templates**: JSON files containing the build information
- **Builders**: Platform specific building configuration
- **Provisioners**: Tools that install software after the initial OS install
- **Post-processors**: Actions to happen after the image has been built

Workflow


Build
Provision
Post Process

AWS
VmWare
VirtualBox
Docker

Building



Ubuntu 16.04 builder

```
"type": "virtualbox-iso",
"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<wait>",
  " auto<wait>",
  " console-setup/ask_detect=false<wait>",
  " console-setup/layoutcode=us<wait>",
  " console-setup/modelcode=pc105<wait>",
  " debconf/frontend=noninteractive<wait>",
  " debian-installer=en_US<wait>",
  " fb=false<wait>",
  " initrd=/install/initrd.gz<wait>",
  " kbd-chooser/method=us<wait>",
  " keyboard-configuration/layout=USA<wait>",
  " keyboard-configuration/variant=USA<wait>",
  " locale=en_US<wait>",
  " netcfg/get_domain=vm<wait>",
  " netcfg/get_hostname=vagrant<wait>",
  " grub-installer/bootdev=/dev/sda<wait>",
  " noapic<wait>",
  " preseed/url=http://{{ .HTTPIP }}:{{ .HTTPPort }}/preseed.cfg",
  " -- <wait>",
  "<enter><wait>"
],
```

Ubuntu 16.04 builder

```
"boot_wait": "10s",
"disk_size": 61920,
"guest_os_type": "Ubuntu_64",
"headless": false,
"http_directory": "http",
"iso_urls": [
  "iso/ubuntu-16.04.2-server-amd64.iso",
  "http://url/ubuntu-16.04.2-server-amd64.iso"
],
"iso_checksum_type": "sha256",
"iso_checksum": "check_summ",
"ssh_username": "vagrant",
"ssh_password": "vagrant",
"ssh_port": 22,
"ssh_wait_timeout": "10000s",
"shutdown_command": "echo 'vagrant'|sudo -S shutdown -P now",
"guest_additions_path": "VBoxGuestAdditions_{{.Version}}.iso",
"virtualbox_version_file": ".vbox_version",
"vm_name": "packer-ubuntu-16.04-postgresql",
```

Ubuntu 16.04 provisioners

```
"provisioners": [
  {
    "override": {
      "virtualbox-iso": {
        "execute_command": "echo 'vagrant' | sudo -S sh '{{ .Path }}'"
      }
    },
    "scripts": [
      "scripts/root_setup.sh"
    ],
    "type": "shell"
  },
  {
    "scripts": [
      "scripts/setup.sh"
    ],
    "type": "shell"
  }
],
```

root_setup.sh

```
#!/bin/bash
set -e
PG_VERSION=9.5
PG_HBA="/etc/postgresql/$PG_VERSION/main/pg_hba.conf"
sudo apt-get update -y -qq > /dev/null
sudo apt-get upgrade -y -qq > /dev/null
sudo apt-get -y -q install linux-headers-$(uname -r) build-essential dkms nfs-common curl wget git vim
groupadd -r admin
usermod -a -G admin vagrant
cp /etc/sudoers /etc/sudoers.orig
sed -i -e '/Defaults\s\+env_reset/a Defaults\texempt_group=admin' /etc/sudoers
sed -i -e 's/%admin ALL=(ALL) ALL/%admin ALL=NOPASSWD:ALL/g' /etc/sudoers

# Install Postgresql
sudo apt-get -y -q install postgresql libpq-dev postgresql-contrib  postgresql-client

# Set Password to test for user postgres and simple configurations
sudo update-rc.d postgresql enable
sudo echo "local   all             postgres                                md5"  > "$PG_HBA"
sudo echo "host    all             all             all                     trust" >> "$PG_HBA"
sudo -u postgres psql -c "ALTER USER postgres WITH PASSWORD 'test';"
```

setup.sh

```
#!/bin/bash

set -e
# Installing vagrant keys
mkdir ~/.ssh
chmod 700 ~/.ssh
cd ~/.ssh
wget --no-check-certificate \
'https://raw.github.com/mitchellh/vagrant/master/keys/vagrant.pub' -O authorized_keys

chmod 600 ~/.ssh/authorized_keys
chown -R vagrant ~/.ssh
```

Ubuntu 16.04 post-processors

```
"post-processors": [[
   {
     "type": "vagrant"
   },
   {
     "type":"webdav",
     "url": "http://example.com/upload/"

   }]
  ]
```

WebDav для boxes

```
server {
   listen 80;
   server_name example.com;
location / {
    dav_methods PUT DELETE MKCOL COPY MOVE;
    create_full_put_path on;
    dav_access group:rw all:r;
    auth_basic "Please login for access";
    auth_basic_user_file /srv/.passwd.dav;
    autoindex on;
    client_max_body_size 0;
    root /srv/vagrant;
    }
}
```

Использование **webdav** для **Vagrant box**

#Загружаем box в репозиторий

```
curl -T ./vagrant.box http://username:password@example.com/vagrant.box
```

#Пример использования

```
mkdir my_cool_box
cd my_cool_box

vagrant init my_cool_box http://username:password@example.com/vagrant.box

vagrant up
```

#Удаление box

```
curl -X DELETE http://username:password@example.com/vagrant.box
```

THANK YOU FOR ATTENTION