

OpenBSD: add VMM to 'packer'

The red pill taken to develop a Go 'plugin' for packer.io to be able to create VM images on OpenBSD for VMM and many other virtualizers.

Philipp Bühler <pb@sysfive.com> @pb_double

sysfive.com portfolio

- Continuous system and application operation
- Collaborations with Providers, Developers, Services and QA
- Hybrid cloud provisioning
- cost efficient scaling on commodity HW
- scale out to AWS/GCE/Azure/..
- Incident, problem, disaster response
- Service availability independent of solution scenario
- migrate from or to private/public cloud or own HW
- robust, scalable technology portfolio
- continuous improvements in security and server architecture
- coherent provisioning across platforms (dev/stage/live)
- vendor/provider independence, OSS focus



Ze problems I tackle

- VM guest keepers need bootable stuff
- "infrastructure to go"
- reproducibility
- cross-host / virtualizer (OSX, OpenBSD, AWS / VirtualBox, VMM, AMI)

Solutions / Approaches

- my-imager.sh
- curl github | sh
- my-cold-hands ("VM guy is AWOL")
- deploy.pl ;-)
- packer (o'rly)

What's packer anyway?

Packer is an open source tool for creating **identical** machine images for multiple platforms from a **single source** configuration. Packer is lightweight, runs on every major operating system, and is highly performant, creating machine images for multiple platforms **in parallel**.

Packer does not replace configuration management like Chef or Puppet. In fact, when building images, Packer is able to use tools like Chef or Puppet to install software onto the image.

- written in golang
- small core providing communications (rpc)
- everything else is a plugin (but linked into one binary)
- configuration in one JSON file (+optional variables)

forms in the sandbox / Terminology

Artifacts

Outcome of a "Build", e.g. AMI, .vmdk, .box

Builds

The actual running task producing above artifacts

Builders

Code to steer the VM host, handle disk images, etc (see below)

Provisioners (optional)

Additional treatment, installation goes here and range from simple inline shell scripts to full-blown ansible, Chef, ..

Post-processors

Treat the Artifacts after creation, e.g. compress, upload AWS, ..

Templates

The JSON files defining all of the above (and some) - NOT a VM "template"

Builders + Post-Provisioning

By default the following “builder” engines are supported. Where needed the accompanying “post-processor” is typically available, too (e.g. EC2/AMI upload):

Alicloud ECS, Amazon EC2, Azure, CloudStack, DigitalOcean, Docker, File, Google Cloud, Hetzner Cloud, HyperOne, Hyper-V, Linode, LXC, LXD, NAVER Cloud, Null, 1&1, OpenStack, Oracle, Parallels, ProfitBricks, QEMU, Scaleway, Tencent Cloud, Triton, Vagrant, VirtualBox, VMware, Yandex.Cloud.

Further “builders” can be found in the wild and are just added as a single go binary in certain paths (e.g. ~/.packer.d/plugins/)

By default additional provisioning support for the following tools:

Ansible, Breakpoint, Chef, Converge, File, InSpec, PowerShell, Puppet, Salt Masterless, Shell.

OpenBSD VMM

vm(4)

virtual machine monitor (VMM) providing the required resources for the VMs (CPU, Disk, Net) and handles the necessary memory mapping (isolation).

vmd(8)

userland daemon to interact with vm(4) to create actual VMs and handle their lifecycle through:

vmctl(8)

administrative tool to create, start/stop, etc VMs. In this scope also the main 'interface' for the packer builder plugin.

vm.conf(5)

configuration file for vmd(8), persist VM/network configurations.

doas(1)

While most tasks the “builder” can (and should!) run as unprivileged user, some commands need to be run as root. The plugin does so automatically. **Caveat:** needs 'nopass' for now (no tty), 'persist' typically timeouts too early.

OpenBSD dependencies / configuration

- /etc/pf.conf:

```
1 pass in quick proto { udp tcp } from 100.64.0.0/10 to any port domain \
2                               rdr-to $dns_server port domain
3 pass out quick on $ext_if from 100.64.0.0/10 to any nat-to $ext_if
```

- /etc/sysctl.conf

```
1 net.inet.ip.forwarding=1
```

- vmd(8)

```
1 rcctl enable vmd
2 rcctl start vmd
```


Enough Introduction..

Questions so far? Keep 'em until the end, way too big a stage here

Plugin development

tools + space

- pkg_add: go, golang, packer, git
- \$VISUAL / \$EDITOR
- diskspace: 1.5G go-dependencies + generated images/diskfiles

directory layout

- / - Makefile, main.go, go.mod
- /builder/packer-builder-openbsd-vmm - work cellar

Plugin development

what is doing what

Makefile

Targets: build, install, vmb, fmt, vet, test, clean, uninstall

main.go

- "import" builder
- initialize builder as a "server" plugin
- register builder (rpc configuration)
- "daemonize", spinning mode, ..

config.go

holds the configuration struct read from template (JSON)

builder.go

- new - instantiate driver with global vars/logs
- prepare - populate configuration
- run - tokenize build into "steps", hand over "artifact" information
- cancel - clean up the mess if SIGABRT

driver.go

Interact via vmctl(8) to create disks, start/stop the VM. Gather additional information like tap(4) IP address. **Fixme:** Get information of “ifconfig tap” to find the IP address. Currently there’s some assumption about how VM-Id and tapN correlate (or not). awk->golang or ioctl needed.

Also interface with the VM (serial console) to "type" so called boot_command. Additionally catching all the output into a logfile (linefeed).

step_*.go

The various “steps” needed to create an artifact (next page)

“Stepping the build”

`step_outdir.go`

create a directory that will hold the artifacts

`step_create_disks.go`

create the empty disk that will be installed on (via `vmctl`, no `doas`).

`step_get_vm_params.go`

Put information about the VM into the 'state bag', to be consumed by following steps.

`step_launch_vm.go`

Start a VM (via "driver") with above created disk and the configured options (name, memory, kernel). **Fixme:** shutdown the VM if there are errors (right now I don't, because `debug` and `auto_install(8)` already stops the VM)

`step_bootcmd.go`

- Get configuration (IP/port) from the built-in `httpd`
- gather the configured `boot_command`, enrich with above data
- "type" the command via the connected serial console. Basically the `httpd` cannot run on `80/tcp`, so `auto_install` will fail-ask for the URL and the plugin provides this (e.g. `http://100.64.1.2:8230/packer-auto_install.conf`)

* DEMO TIME*

```
1 {
2   "builders": [
3     {
4       "type": "openbsd-vmm",
5       "name": "packer-obsd64-vmm-amd64",
6       "vm_name": "myvm",
7       "disk_size": "1500M",
8       "disk_format": "raw",
9       "output_directory": "images",
10      "http_directory": "./docroot",
11      "iso_image": "~/Downloads/install65.iso",
12      "bios": "/bsd.rd",
13      "boot_wait": "5s",
14      "boot_command": [
15        "A<enter>",
16        "http://{{ .HTTPIP }}:{{ .HTTPPort }}/packer-auto_install.conf<enter>",
17      ],
18      "ssh_username": "root"
19    }
20  ]
21 }
```

<https://asciinema.org/a/246654>

asciinema play ~/devel/presenter/presentations/h4g/2019/tl-demo.cast

Status

- VMM: everything on deck (full/real PXE would be add-on)
- plugin: all architectural integration (config/builder/driver)
- plugin_steps: create disk/VM/boot/provide auto_install infra

Outlook

Fixme / Finish

- GetTapIpAddress
- more observation than fixed timeouts - if possible

Needables

- plugin: provide/check dependencies (vmd running, pf.conf)
- plugin: make it a ports-pkg
- plugin: support fiddlings (like variables for boot_command, basically there)
- plugin: config item 'disklabel'
- plugin: multiple disks
- plugin: "complex" networking?
- plugin: adapt to upcoming VMM features

Integration (Puffalanche)

- make 'vagrant' a port and use it as a vbox post-processor
- Cloudify the clones (cloud-init, context,..)
- "Ecosystem" - cloud at puffy fingertips (hi mischa, sorry)
- Disk resizing (maybe not needed with images-a-la-carte..)

Ohai + Links + Thanks

- Code/Slides - <https://v.gd/packobsd>
- Kickoff - Glarus, Switzerland / <https://hack4glarus.ch>
- Thanks to Claudio for the idea of vmctl -B
- Thanks to grubernaut for go.mod, review, ISO+QCOW2 support and other spurious fixes
- Any help/pull request very welcome (e.g. multi-disk)

Questions?



BEER after the closing session

DO NOT MISS - see you there

Code/Slides -

<https://github.com/double-p/packer-builder-openbsd-vmm>

<https://github.com/double-p/presentations>



Golang

- GOPATH (disk sizes/isolation)
- go.mod/go.sum
- go vet
- go fmt
- // interface{ }