Next-Gen Virtual Router

cloud STACK COLLABORATION CONFERENCE 2021

Rohit Yadav

Principal Engineer, ShapeBlue

rohit.yadav@shapeblue.com

whoami

- **Rohit Yadav,** Principal Engineer @ ShapeBlue
- From Gurugram, India
- Background:
  - 9+ years CloudStack Committer and PMC
  - RM and maintainer for several releases
  - Specialize in design and architecture, development work, framework/plugins, tooling; leadership, mentoring, training
  - Author of several flagship features and tools
  - CloudStack ARM64/RaspberryPi maintainer
  - Love cats 🐱 and programming

Agenda

- Problem and Pain Points
- Recap
- Ideas, Experiments, Demos
- Q&A

Are We Done Yet?

The Future of CloudStack Virtual Router

CCCNA Las Vegas, 10 September 2019

CloudStack Virtual Router: Past, Present, Future

CCCNA Montreal, 24 September 2018

Problem and Pain Points

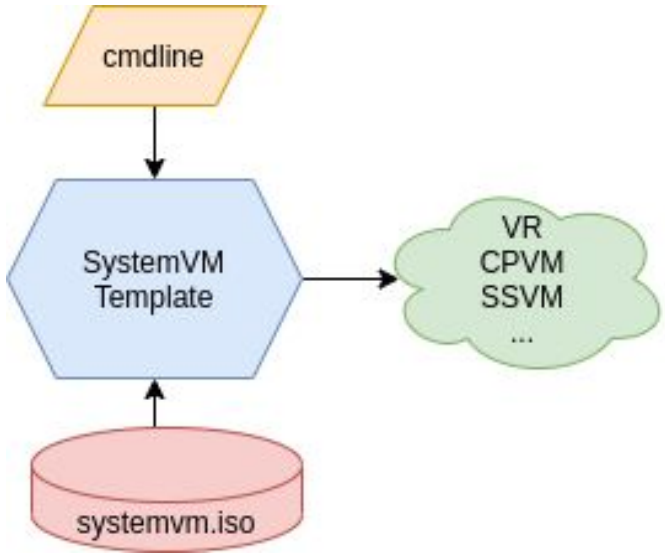
- VR:
  - Codebase: difficult to develop, debug, test, maintain
  - Upgrading (zero-downtime upgrades)
  - Lack of API: Hacky scripting, code, databags, scp/ssh
  - Performance: processing rules (firewall…)
- SystemVM template:
  - lifecycle and management (non-standard, ACS managed)
  - installation, register/upgrade with releases **(frequently)**

Solve for X

Solve for:

- development and maintainability
- testing
- upgrading
- performance, security & robustness
- clients: api, rpc, ...

Recap: SystemVM and VR Lifecycle



- Build (packer)
- Host (community servers, S3…)
- Install, upgrade (docs, scripts…)
- Lifecycle:
  - Create: copy disk
  - Networking: nics; link-local/private
  - Init/Config: cmdline
  - Patching: systemvm.iso (ssh, software)
  - RPC/Programming/Data-model:
    - VR: ssh + databags, json/xml
    - SSVM/CPVM: agent + Cmd/Answer
  - Upgrade: destroy older systemvm/VR

☠️ DON'T EVEN TRY TO READ THIS

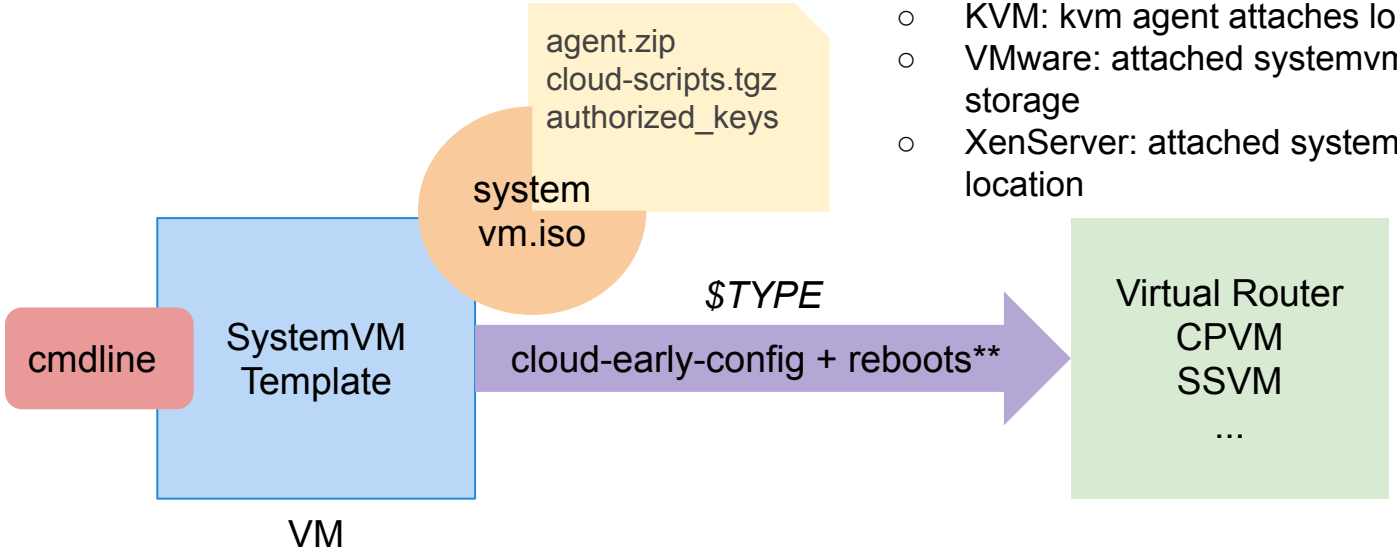
Present VR Programming

- Orchestration: *VirtualRoutingResource* -> *RouterDeployer*
- Executable scripts at **/opt/cloud/bin/** in VR
- Executable scripts run via **router_proxy.sh** or directly in the /opt/cloud/bin path
- Commands sent as json saved at **/var/cache/cloud/** and updated in VR by **update_config.py**. Once processed they are moved and gzip-ed at /var/cache/cloud/processed.
- VR Config (VR-<uuid>.cfg) file has aggregated file+contents and commands in a custom xml format, processed by vr_cfg.sh.
- VR config jsons are stored at **/etc/cloudstack/** which is used to compare existing vs new config and only diffs (changes) are applied that are calculated by per-command type *databag* handlers (in cs_*.py, merge.py).

Yak Shaving: SystemVM Templates

How SystemVMs are patched?

- Non-standard files, patching process
- Attached to a systemvm while booting:
  - KVM: kvm agent attaches local systemvm.iso
  - VMware: attached systemvm.iso on sec storage
  - XenServer: attached systemvm.iso from local location



Ideas & Experiments

4.16 SystemVM Template Improvements

- Templates bundled in the cloudstack-management (noredist) pkg
- Auto-seed/install/upgrade logic
  apt upgrade | *yum update!*
- Turnkey (separate hosting is optional)
- Now also used for CKS host template

SystemVM Template Idea#1

Why need systemvm.iso?

- Let's **deprecate/remove** systemvm.iso

How?

- pass ssh public key in cmdline string
- faster startup - no more injectkeys! (remove injectkeys.sh)
- scp/ssh necessary payload/tarball (based on systemvm type)
- one less dependency that requires secondary storage

SystemVM Template Idea#2

Deprecate non-standard user-data (cmdline) based patching?

- Can we explore and use cloud-init?
- Can we explore use of config-drive iso?
- Should we have a dhcp-enabled system-network (user-data, config-drive, metadata service)?
- Use IPv6 link-local address?

SystemVM Template Idea#3

Can we have smaller templates and/or faster builds?

- Build and bundle with maven build (60s):
  Alpine SystemVM using qemu-nbd
  https://github.com/shapeblue/alpine-systemvm
- Root ISO + datadisk for persistence
- ~~Use lighter linux-image-cloud kernel~~ (doesn't work on VMware)

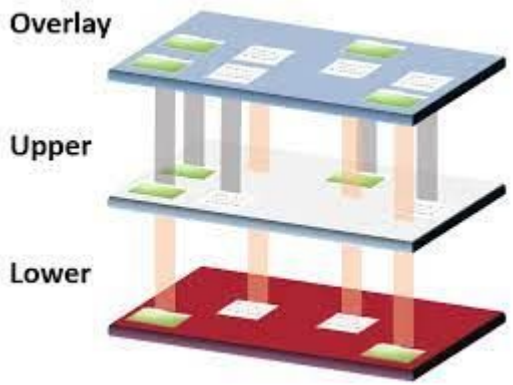
SystemVM Template Idea#4

Can we revisit patching and upgrading?

- Explore idea of live-patching/upgrading in-place
- Is live-patching possible between different ACS systemvm versions?

SystemVM Template Idea#4.1

OverlayFS2



- Have read-only root-fs with just the kernel and some utilities
- ACS version specific folders which can be switched between layers
- Example, VR agent/software lives/runs from /router/latest/...

```
# ls /router
4.15
4.16
latest -> 4.16
```

More: https://wiki.debian.org/ReadonlyRoot

SystemVM Template Idea#4.2

- Thin Template (just kernel + some utilities)
- Install and upgrade dependencies/packages on patching
  - Use a fatter systemvm.iso (with pkgs, ideas already in use for CKS)
  - Apt-pkgs repo on mgmt server (http://<ms:8080/client/static/repo…) or on secondary storage

SystemVM Template Idea#4.3

- Standalone VR agent (single binary or pkg) that may replace most of the non-core services
- Core services all provided by kernel: nic/address, routing, firewall (nat, pf, allow/deny acls etc)
- Non-core services: dhcp/dns, password, metadata, lb, vrrp, misc (health checks, monitoring…)
- Misc-services: vpn framework (strongswan->wireguard)

VR: Core vs Non-Core Services?

What causes network downtime?

- Firewall, ACLs
- NAT, SNAT/DNAT
- Forwarding Rules
- Guest Networking
- Static Routes
- VPN (strongswan, wireguard, openvpn)**

What causes service downtime?

- DNS, DHCP (dnsmasq)
- Password Server
- Metadata (apache2)
- LB (haproxy)
- Redundancy (Keepalived, conntrackd)
- Misc (health, monitoring, network stats)

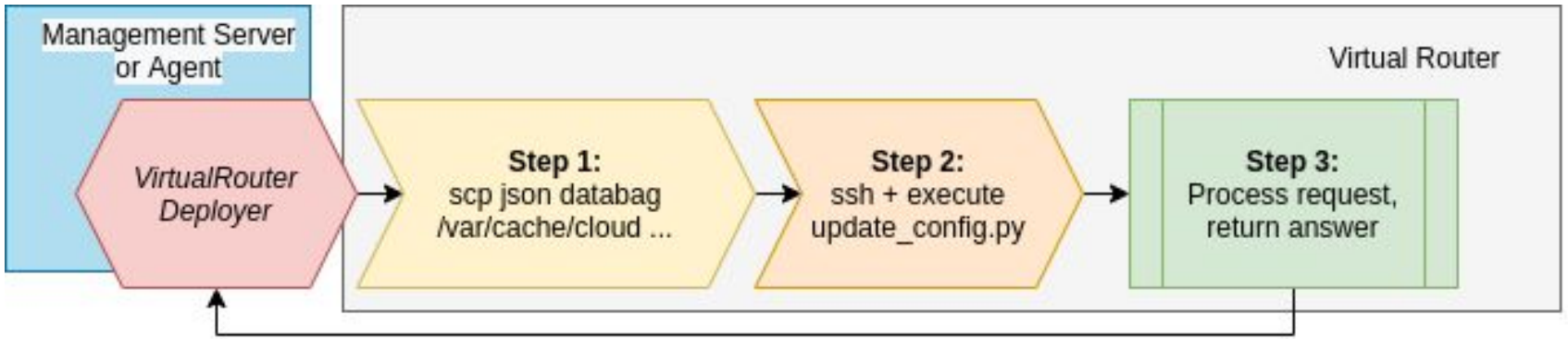
Replace with off-the-shelf router distro?

- VYoS
- pfSense
- OPNsense
- OpenWrt*
- DIY docker/containerd in VR?
- Common issues: licensing, longevity risk, hypervisor support, security and updates...

Current Implementation

- Client: VR deployer class (hypervisor specific)
- VR codebase: many script (python, shell), no API end-point (uses update_config.py)
- Data-model: json/xml databags
- Database: json files in /etc/cloudstack
- Communication (RPC): scp & ssh
- Logs: /var/log/cloud.log

Workflow: Current (*simplified)



VR Limitations

- 256MB RAM, 1 vCPU
- Debian-based, ext4 FS
- Patches via cloud-early-config/cmdline
- Hypervisor-specific guest-tools
- Current: python/shell scripts
- SSH based RPC

VR Agent Ideas: Codebase

- Python3 (module/files)
- Go (standalone)
- Lua/Luajit (scripts)
- Shell (scripts)

Tradeoff: community skills, maintenance/development, resource requirement (cpu/mem), availability of libraries and speed of execution

VR Agent Ideas: Data-model

- pojo/json (objects serialised to json)
- IDL: protobuf/grpc, thrift, avro, message pack...
- DIY?

VR Agent Ideas: Datastore/DB

- json
- sqlite3
- IDL objects (protobuf)

VR Agent Ideas: API/RPC Communication

- ssh/scp (port 3922)
- HTTP Server (tcp/tls port)
- Unix-domain sockets (/var/cache/vr-agent.sock)
- Named pipe (mkfifo, /var/cache/cloud/routerbus)
- socat/nc based relay or Java equivalent

VR Agent Ideas: Security

- CA-framework issues certificates (for client & server/VR agent)
- ssh or Tunnels/forwarding over ssh (mgmt server public key)

VR Agent Ideas: Live Patching

- In-place upgrades
- Mgmt server live-patches without rebooting SystemVM
- Pre and Post upgrade hooks to check live-patching
- ***Note: ssh works every between incompatible ACS-systemvmtemplate versions***

Connection: VR Agent-MS/Agent

- Mgmt server connects to VR agent directly
  - VMware: already has private/control nic on VR
  - KVM/XS/XCP: require adding a private/mgmt nic on VR (i.e. more pod IP range?)
- Mgmt server connects to VR via proxy via host or agent:
  - VMware: not needed/supported
  - KVM: mgmt server -> agent -> VR (via proxy or link-local IP)
  - XS/XCP: via host plugin

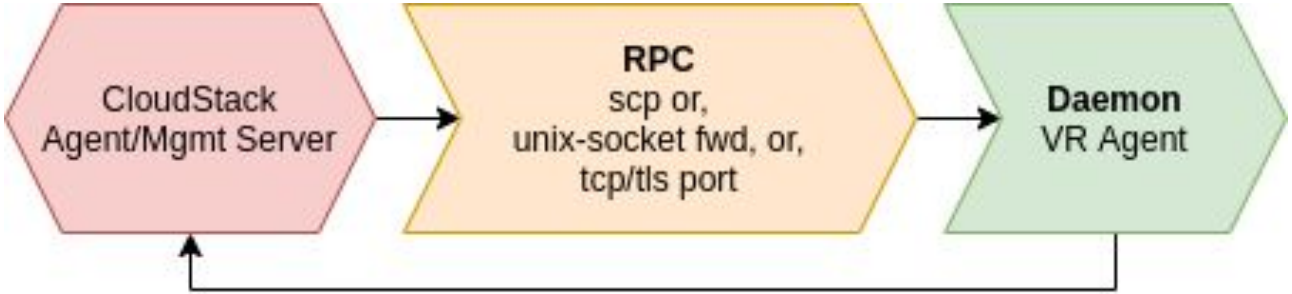
Experiment #1: IPC named pipes

- named pipes or message queue
- scp serialised object (say protobuf/json) to named pipe
- single operation instead of scp + ssh to exec
- cons: trade-off that both are fundamentally *unidirectional*

Experiment #2: Unix sockets

- Listen on unix domain socket
- Unix-domain socket forward over ssh
- Future-proof gRPC/RPC* - which can listen on both tcp/tls, and unix domain-socket

Workflow: Future (*simplified)



VR Agent: Possible Future

- Codebase: Python, Go, Lua, ...
- Data-model: Protobuf objection/json
- Database: sqlite3, or protobuf/json
- RPC: tls/tcp, ssh+unix-domain sock
- VR CLI!
- Workflow:

  Mgmt server|Agent <-> Cmd -> RPC <-> VR agent (Ans)

Q&A