Ignition:
declarative first-boot
host configuration

Luca BRUNO
@lucabruno | lucab@redhat.com | github.com/lucab
$ whoami

“OS engineer, Rust & Go developer, enthusiast FLOSS supporter”

- ex-CoreOS Software Engineer
- Redhat, CoreOS-Berlin office
- Previously: security researcher/engineer
Overview

Ignition development in a nutshell:
- Historical problems
- Whiteboarding session
- Under the hood
Historical problems
Compact distribution for containerized workloads

- Immutable OS image (i.e. read-only /usr)
- No interpreters (i.e. no python)
- No package manager

ContainerLinux
Cloud-init

Handling early-initialization of cloud instances

- Implemented as python modules/scripts
- YAML userdata
- Set of services that runs in rootfs
  - On each boot*
Problems (1)

Relevant in the context of ContainerLinux

- Need python interpreter and modules
- Interact with package manager
- Many features that don’t fit well, due to:
  - Minimal, read-only /usr
  - Systemd-only
Problems (2)

Generally relevant

- YAML is a human-interface
- Config is a mix of declarative and imperative statements
- Race with rest of services startup
- Runs too late in the boot process
- Can run multiple times / on every boot
- Can leave nodes half-configured

Configuration Manager for a mutable OS
CoreOS cloud-init

Rewrite of cloud-init, tailored for ContainerLinux

- Self-contained native binary, implemented in Go
- Non-strict subset of cloud-config features
  - No package management
  - Logic for etcd/flannel/docker/etc.
- Successful in driving CL provisioning story

Still too many caveats, abused as a generic script-runner
Problems (wishlist)

On top of that, we collected a list of wishes:

- Declarative configuration
- Machine-friendly interface
  - Easy to generate/parse by other tools
- Atomic provisioning
  - Configuration is either fully applied, or node does not boot
- Avoid interferences and ordering issues with other services
Whiteboarding session
Go

Makes it easy to build simple and reliable programs

Pros:
- Static typing and concurrency
- Safer than C
- Simple to learn, read and write
- Compiles to native binary
Boot overview

Host firmware:
- BIOS
- UEFI
- NetBoot
- Hypervisor specific

Bootloader:
- GRUB
- PXE
- ISO boot

Early Userspace:
- (initramfs)

Userspace:
- (rootfs)
initramfs

“Early” userspace

- Minimal environment
- Can run systemd as environment supervisor
- Does not depend on storage/network
  - Useful to initialize them :)


Boot overview

- Bootloader
- Early userspace
- Systemd
- Ignition
- Userspace
JSON (schema)

Javascript data-interchange format

- Easy to consume/produce from any language
- Machine-friendly
- Human-readable
- Out-of-band schema via JSON-Schema
Under the hood
Ignition overview

First-boot host configuration tool

- Fully-declarative userdata
- Run only on first boot
- Run in initramfs
  - Can partition disks, format volumes, etc.
- Typed JSON as input
- Support fetching remote assets

https://github.com/coreos/ignition
Ignition execution

Multi-stage execution

- First-boot detection (external)
- Config fetching
- “disks” stage
  - Prepares new volumes or ensures existing ones
- “files” stage
  - Write user content to disk
- “quench” stage
  - Successful provisioning → machine boots
Ignition initramfs

```
systemd.generator

- ignition-setup.service
  - cmdline coreos.first_boot
    - systemd-networkd.service

  - ignition-disks.service
    - ignition-files.service
      - ignition-quench.service

<table>
<thead>
<tr>
<th>Service</th>
<th>ExecStart</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignition-disks</td>
<td>/usr/bin/ignition --root=/sysroot --oem=${OEM_ID} --stage=disk</td>
</tr>
<tr>
<td>ignition-files</td>
<td>/usr/bin/ignition --root=/sysroot --oem=${OEM_ID} --stage=files</td>
</tr>
<tr>
<td>ignition-quench</td>
<td>/bin/rm /sysroot/boot/coreos/first_boot</td>
</tr>
</tbody>
</table>
```
Platform providers

- Azure
- DigitalOcean
- URL / File
- GCE
- EC2
- Openstack
- Packet
- PXE / TFTP
- Qemu
- ...

```
  EC2 Fetcher

  File Fetcher
  config.ign

  QEMU Fetcher
  /sys/firmware/qemu_fw_cfg/...
```
Ignition JSON

JSON as userdata input format

- Schema-first
  - Go code auto-generated from JSON schema
- Support for chaining config snippets
  - Useful for dynamic or instance-specific bits
- Support for full-config replacement
  - Useful for stubbing
- Semver-versioned
  - 3.0.0 currently in progress

https://github.com/coreos/ignition/blob/master/doc/examples.md
JSON Producers

JSON userdata is a **machine-interface**

- **ct** (Container Linux Config Transpiler)
  - Higher level YAML $\rightarrow$ JSON
- **terraform-provider-ignition**
  - Terraform $\rightarrow$ JSON
- **matchbox**
  - Portable PXE provisioner with a cloud-like API
- **openshift-installer** (*v4*)
  - From-scratch autonomous cluster bootstrapping
Questions?

With credits to Dalton (@dghubble) for graphics and inspiration