Bottlerocket OS

An Operating System for Hosting Containers:
github.com/bottlerocket-os

Justin Haynes
8/17/2020

Where are we headed today?

- What is a container host OS?
- Why a container OS vs a full Linux distro?
- What did we build?
- How do you use it?
- Where are we headed next?

Ok, I’ll bite: what is a container host OS?

A Linux distribution purpose built from the ground up to run containers.

- There are others: CoreOS, RancherOS, Talos, etc.
- They’re all “minimal”
- They’re all updatable – usually transactionally

What makes Bottlerocket?

- Secure: Opinionated, specialized, highly secure
- Flexible: Multi-cloud, multi-orchestrator
- Transactional: Image-based upgrade and rollback
- Isolated: Separate container runtimes

What makes Bottlerocket? (First priority: security)

- SELinux is enabled in enforcing mode by default
- dm-verity protects read only root filesystem
- /etc is a stateless tmpfs
- No shell or interpreters installed
- Binaries built with hardening flags

github.com/bottlerocket-os/bottlerocket/blob/develop/SECURITY_FEATURES.md

What makes Bottlerocket? (flexibilty)

- Different builds of Bottlerocket are called a variant
- Variants are a combination of software, modeled settings and disk layout
- Bottlerocket can be thought of as a container host OS builder

What makes Bottlerocket? (transactional updates)

- We use The Update Framework (TUF) to publish and secure our repositories
    - github.com/awslabs/tough – TUF implementation in Rust
- Full disk images download to an alternate set of partitions
- *updog* toggles the partition priority – and falls back on failure

What makes Bottlerocket? (isolated runtimes)

- Bottlerocket has two container runtimes running…
- Host containers
    - Control container on by default (remote API access)
    - Admin container off by default (deep debugging and exploration)

What makes Bottlerocket? (modeled settings)

- Key-value pairs under “settings” prefix
- Validation rules prevent invalid configuration
- Read once from user data on boot
- Modified via the API socket
- Written to a small dedicated filesystem

What makes Bottlerocket? (settings: the hard part)

- Settings may only be known at runtime
- Settings can depend on other settings
- Settings need to be written to a configuration file
- Settings can evolve over time

Solved with metadata for settings!

What makes Bottlerocket? (settings: the metadata)

- Generators run if a setting is missing
- Configuration files that specify template and output paths
- Services that have associated files and start/restart commands
- All exposed through the API

What makes Bottlerocket? (settings: an example)

- Can be specified directly:

  settings.host-containers.admin.source = 863599026182.dkr.ecr.us-west-2-amazonaws.com/fedora-admin:v0.4.0-698d2978-dev

  (short circuits all the fancy)

What makes Bottlerocket? (settings: an example)

- No value set by default
- Metadata specifies a generator:

  schnauzer settings.host-containers.admin.source
- Metadata specifies a template:

  328549459982.dkr.ecr.{{ settings.aws.region }}.amazonaws.com/bottlerocket-admin:v0.4.0

What makes Bottlerocket? (settings: an example)

- *early-boot-config* runs and populates settings based on EC2 instance metadata, including *settings.aws.region*
- *sundog* runs and calls generators including *schnauzer*
- *schnauzer* retrieves the region setting and writes *settings.host-containers.admin.source*

What makes Bottlerocket? (migrations)

- Need a way to change defaults over time
- Settings are persisted to the filesystem
- Settings can be modified by the user
- Some distros let packages run scripts on upgrade
- We have helpers called migrations:
    - Must run in both directions: forward and back
    - Must be a static binary to avoid dependencies on host OS details
    - Tied to individual settings rather than variants

How do I use Bottlerocket?

On AWS, launch Bottlerocket AMIs just like you would any other instance.

- Bottlerocket's user data is structured TOML
- Remote API access via AWS Systems Manager (SSM Agent)
- You can launch a troubleshooting container via user data:

```
[settings.host-containers.admin] enabled = true
```

Or the API:

```
apiclient -u /settings -m PATCH -d '{"host-containers": {"admin": {"enabled": true}}}’
apiclient -u /tx/commit_and_apply -m POST
```

How do I use Bottlerocket?

SSH into the admin container with the keys provided when the instance was launched and explore:

- Data persists at *.bottlerocket/host-containers/admin***/** (where the ssh public keys are stored)
- *superpowered = true*
- *sheltie* to “breakout” of the admin container onto the host for deeper debugging
- *logdog* to capture system information for debugging

How do I use Bottlerocket?

Bottlerocket can be updated automatically via a Kubernetes operator:

```
$ kubectl apply –f Bottlerocket_k8s.csv.yaml
$ kubectl get ClusterServiceVersion Bottlerocket_k8s | jq '.status'
```

Or manually via the API:

```
apiclient -u /actions/refresh-updates -m POST
apiclient -u /actions/prepare-update -m POST
apiclient -u /actions/activate-update -m POST
```

Our Growing List of Certified Bottlerocket Partners

alcide

armory

CROWDSTRIKE

DATADOG

epsagon

Logic Monitor

puppet

New Relic

sysdig

TREND MICRO™

TIGERA

weaveworks

aws

Where are we headed next?

- GA in 2020 and 1.0 release
    - Availability in all standard regions
    - ECS Optimized variant
    - Support for all EC2 instance types and families (GPU coming soon)
- VM images for on-prem usage
- Bare metal support
- Firecracker integration
- ???

We’re on GitHub:

github.com/bottlerocket-os and
github.com/awslabs/tough