Your First  CoreOS Linux Cluster

Joe Thompson <joe.thompson@coreos.com>

Introductions

Who am I? Why am I here? Who made me king?

CoreOS Linux Basics

Why does CoreOS exist?

- To secure the Internet

CoreOS Linux Basics

How do we secure the Internet?

- Seamless, secure, automatic updates behind the scenes
- Hands up everybody who ever clicked “update later”, “not now”, “postpone”, “I hate you”, etc. on an update button...

CoreOS Linux Basics

So if it’s such a good idea, why doesn’t everybody do it?

- Because doing it right is hard

CoreOS Linux Basics

So how did you solve such a hard problem?

- We let Google do it with the Chrome web browser
- Omaha update process

CoreOS Linux Basics

How CoreOS updates work

- CoreOS has two operating system image partitions (you can see this by examining the partition info) - A and B
- CoreOS server boots off one (we'll say B)
- CoreOS server checks for updates regularly (by calling either home base or a customer's local CoreUpdate)
- If an update is found, it is downloaded in the background to image partition A
- When the server reboots, it sees a new image in partition A and attempts to boot off it

CoreOS Linux Basics

How CoreOS updates work (cont'd)

- If the boot is successful, so is the update, and A becomes the new boot partition going forward (B is now idle and can be updated)
- If the boot fails, partition A is marked with metadata indicating that the image on it is non-viable and boot reverts to B.
    - Status quo ante until the next update is available

CoreOS Linux Basics

But wait, my server's going to reboot everytime an update is available?

- By default, **yes**, and that's **good**
  - Reboots flush out ad-hoc state
    - Even the most perfect organizations should do them regularly as a test
  - The longer you delay a reboot, the longer you're vulnerable/affected by a bug/missing a cool new feature
- If your app is written robustly it will not notice a random node or two rebooting
  - We enable assistance for that in the OS which we'll get to later

CoreOS Linux Installation

Multiple release channels

- Alpha -- bleeding edge
  - New features land here first
- Beta -- testing in the fire
  - Not recommended for production, but please do download and use our betas!
- Stable -- production-quality code

CoreOS Linux Installation

Use whatever method works for you:

- ISO install
- Vagrant box
- PXE/iPXE network boot
- Cloud provider image (DigitalOcean, Google Compute, Amazon, Azure…)
    - The easiest for many users because cloud images typically support passing metadata

CoreOS Linux Installation

Metadata? Like what?

- Like this:

CoreOS

A very simple cloud-config

```
#cloud-config

coreos:
  etcd2:
    discovery: "https://discovery.etcd.io/<token>"
    advertise-client-urls: "http://$public_ipv4:2379"
    initial-advertise-peer-urls: "http://$private_ipv4:2380"
    listen-client-urls: "http://0.0.0.0:2379"
    listen-peer-urls: "http://$private_ipv4:2380"
  fleet:
    public-ip: "$public_ipv4"
    metadata: "region=us-west"
  update:
    reboot-strategy: "etcd-lock"
  units:
    - name: "etcd2.service"
      command: "start"
    - name: "fleet.service"
      command: "start"
ssh_authorized_keys:
  - "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC0g+ZTxC7weoIJLUafOgrm+h..."
```

DON’T PANIC

CoreOS Linux Installation

It’s actually pretty simple

- cloud-config is just YAML that describes configuration of various components

CoreOS

Here's what it all means:

```
coreos:
  etcd2:
    discovery: "https://discovery.etcd.io/<token>"
    advertise-client-urls: "http://$public_ipv4:2379"
    initial-advertise-peer-urls: "http://$private_ipv4:2380"
    listen-client-urls: "http://0.0.0.0:2379"
    listen-peer-urls: "http://$private_ipv4:2380"
```

Here we're setting up the configuration of the etcd process on this node (We'll talk about etcd next)

Here’s what it all means:

```
fleet:
  public-ip: "$public_ipv4"
  metadata: "region=us-west"
```

This is a very basic configuration for fleet (we’ll talk about that after etcd)

CoreOS

Here's what it all means:

```
update:
  reboot-strategy: "etcd-lock"
```

This says that nodes in this etcd cluster should only be rebooted if the cluster can stay up without them (which is one way we help keep your app up)

Here’s what it all means:

```
  units:
    - name: "etcd2.service"
      command: "start"
    - name: "fleet.service"
      command: "start"
ssh_authorized_keys:
  - "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC0g+ZTxC7weoIJLUafOgrm+h..."
```

The last few lines just set up systemd units to run etcd2 and fleet, and write an SSH key so you can log in after it boots up

Demo: Installing CoreOS Linux on DigitalOcean

What **is** etcd? It's actually pretty simple:

- Think of it simply as a distributed /etc directory
- Uses the Raft consensus algorithm to maintain consistency
- Conceptually inspired by a paper from Google Research
    - Google built an internal distributed locking service named Chubby that had similar challenges

etcd

How etcd works:

- etcd nodes are in a cluster with their peers
- etcd nodes may also talk to non-peer clients
- etcd nodes publish their peer and client URLs to the cluster
- when the cluster is formed, the set of peers must agree on some key things
  - the cluster token
  - the list of peers

etcd

Operating etcd in the real world:

- etcd is not for storing large amounts of data
- clusters should, with rare exceptions, be 3, 5, or 7 nodes
  - never use an even number
  - never use just one nodes
  - **more** nodes are better, but only up to a point, and that point is about 7
- etcd is very resilient to node failures
  - An elected leader
  - If the leader dies, the cluster briefly stops while a new leader is elected
  - If a non-leader dies, nobody cares unless quorum is actually lost

etcd

Monitoring etcd is easy:

- etcdctl cluster-health
- /health URL
- exposes metrics via Prometheus

fleet

What the heck is fleet?!

- fleet is a distributed systemd
- Can run ordinary units on a single host, or global units on all hosts
- Also supports some intelligence to control unit placement via extensions to the unit file spec
- Jobs submitted using the fleetctl client

CoreOS

A simple fleet job spec

```
[Unit]
Description=My Apache Frontend
After=docker.service
Requires=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull
coreos/apache
ExecStart=/usr/bin/docker run --rm --name
apache1 -p 80:80 coreos/apache
/usr/sbin/apache2ctl -D FOREGROUND
ExecStop=/usr/bin/docker stop apache1
```

Not much going on here; this unit will be scheduled on some available host, and then presumably Apache will come up and do something.

But wait, there's more!

A simple fleet job spec

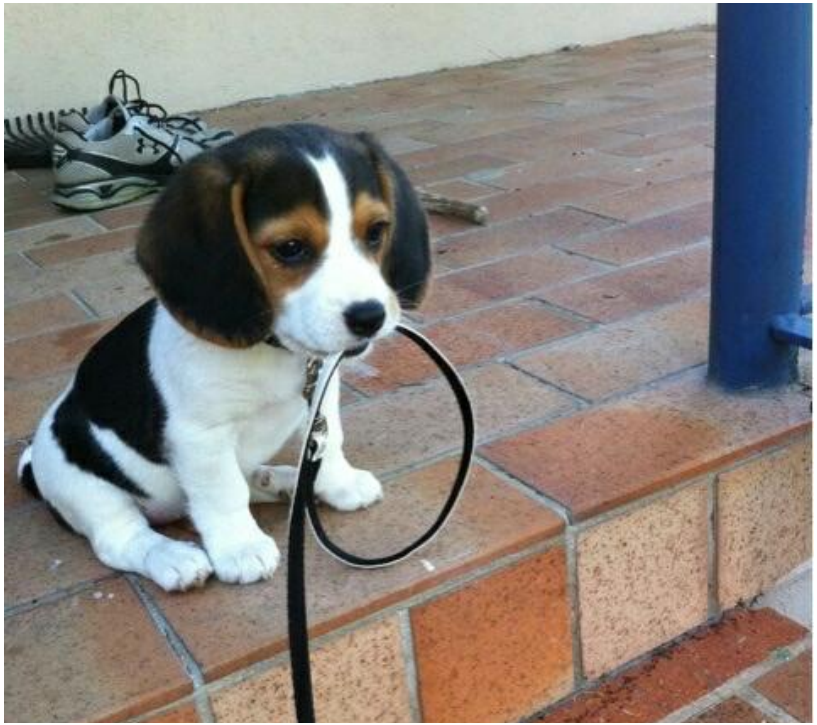
```
[Unit]
Description=My Apache Frontend
After=docker.service
Requires=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache
ExecStart=/usr/bin/docker run --rm --name
apache1 -p 80:80 coreos/apache
/usr/sbin/apache2ctl -D FOREGROUND
ExecStop=/usr/bin/docker stop apache1

[X-Fleet]
Conflicts=apache@*.service
```

This could be a template unit, in which case it probably does not make sense to run two copies of it on one host...

Demo: Running jobs with fleet



Enjoy this picture of a puppy and relax. You've earned it.

Resources

Various links/other items of interest

CEO and co-founder Alex Polvi on the origins of CoreOS:

http://www.activestate.com/blog/2013/08/alex-polvi-explains-coreos

Customizing CoreOS Linux with cloud-config:

https://coreos.com/os/docs/latest/cloud-config.html

DigitalOcean API reference:

https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-api-v2

Questions?