CoreOS

Your First Kubernetes Cluster

or: How to build the one you get in trouble with

Joe Thompson <joe.thompson@coreos.com>

Introductions

(Who am I? Why am I awesome?)

- Joe Thompson

CoreOS

The Agenda, part 1

A whirlwind tour of Kubernetes operational components:

- Docker and Flannel
- Controller (fka “master”)
  - kubelet and kube-proxy
  - API server, Controller manager, scheduler, podmaster
- Node (fka “minion”)
  - kubelet and kube-proxy
- kubectl CLI
- Add-ons

What is Kubernetes?

- Container orchestration
- Grew out of the old Borg system at Google as discussions on Borg's successor Omega began

CoreOS

Docker and Flannel (just in case you're new to this)

- Docker is a container runtime
  - gets your containers running on a node
- Flannel provides the overlay network Kubernetes uses
  - Kubernetes basically wants to assign an IP to everything from a Pod* up
  - flannel takes an allocation block and parcels it out to hosts in subnets
  - This is part of how containers can communicate with each other across the entire cluster
    - Hold that thought

Controller

- In charge of everything to do with the cluster's state
  - Provides an API for clients to query
  - Makes scheduling decisions
  - Tries to reconcile current cluster state with desired state

Node

- Does all the work
  - Checks in through the API server to see what it should be doing
- May be combined with a controller, either as a single-node “cluster” or as part of a cluster where controller nodes are workers too
  - Try not to do that though...

kubectl

- The CLI to interface with Kubernetes
  - creating/deleting/editing resources
  - interacting with the cluster itself

Add-ons

- Optional components that add features to your cluster
  - DNS discovery
  - UI
  - etc.
- Often deployed as pods themselves

CoreOS

The Agenda, part 2

A whirlwind tour of Kubernetes resources:

- Pods
- ReplicationControllers
- Services
- Volumes and Secrets
- Cloud integration magic and a quick glance ahead

Pods

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

- The basic unit of Kubernetes
- Containers running in a pod share a namespace and an IP address

ReplicationControllers

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    [...]
```

- ReplicationControllers make sure the right number of Pods with labels matching the defined criteria in the selector statement are running
- If too many, one/some get killed
- If too few, more get created using the inline Pod template in the ReplicationController
  - This **may not be the same** as currently-running Pods managed by that ReplicationController

Services

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
     app: guestbook
     tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
     app: guestbook
     tier: frontend
```

- Provides a way to discover network services running in the cluster
- type: LoadBalancer load-balances traffic to the selected pods
- Creating a service, then a Pod causes the containers in the Pod to see environment variables pointing at the IP(s)/port(s) the service exposes
- If you have the DNS add-on, records will be populated as well
- This is the other half of that thought I told you to hold

Volumes

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
[...]
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    gcePersistentDisk:
      pdName: my-data-disk
      fsType: ext4
```

- Main types: emptyDir, hostPath, various network storage, something integrated with your cloud provider
- Volumes mounted in the Pod are available to all containers in the Pod

Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: dmFsdWUtMg0K
  username: dmFsdWUtMQ0K
```

- Data is base64-encoded
- Used by mounting as a Volume
  - Pods mounting Secrets do not pick up updates to them after Pod instantiation

CoreOS

Other goodies - cloud integration; the future

- Various cloud providers have k8s LoadBalancer integration
  - Actually creates a cloud LB (whatever that means on that provider) with an external IP
- New types of resources are in development
  - DaemonSet: runs something on every node that matches defined criteria
    - Easy autoscaling, auto-deploy of things like monitoring
  - Deployment: a resource containing all the resources that make up a complete app and defining relationships between them

CoreOS

The Agenda, part 3

A whirlwind tour of deploying Kubernetes the CoreOS way:

- Create an etcd cluster and a separate set of Kubernetes compute resources
- Generate and install your TLS assets
  - CA cert, host certs/keys, client cert(s)/key(s)
- Install the Controller(s)
  - flannel, Docker, kubelet config
  - API server, kube-proxy, controller manager, scheduler, podmaster pods
  - Create namespace
- Install the Node(s)
  - flannel, Docker, kubelet config
  - kube-proxy pod, kubeconfig
- Configure kubectl

Deployment prerequisites: etcd

- Kubernetes uses etcd as its knowledge store and source of truth
- etcd cluster should, if possible, be separate (but not isolated) from Kubernetes to avoid resource contention issues
- Use TLS to secure etcd communication between peers and with clients
- Set flannel network key/value in etcd

Deployment prerequisites: TLS setup

- The easiest thing to do may be to run your own CA
  - openssl
  - cfssl
  - whatever other method you use for doing that
- Make sure that you get the details right
  - All identities listed as SANs if any are
  - No wildcard IP SANs

CoreOS

Deployment: Controller

- Deploy flannel and Docker config
- Deploy kubelet
  - If you have a fairly recent alpha or beta of CoreOS Linux, it's already in the OS, otherwise download it
- Insert manifests for api-server, controller-manager, scheduler, podmaster, kube-proxy
  - These will be run locally by the kubelet itself even though no API server exists to run pods until this is done
- Start services up
- Create namespace for pods
  - Pods start after their assigned namespace is created

CoreOS

Deployment: Nodes

- Lots simpler
- Deploy flannel and Docker config
- Deploy kubelet
- Insert manifest for kube-proxy
- Insert manifest for kube-config to enable authentication to controllers
- Start services

Deployment: kubectl

- Download binary
  - Make sure the version matches the version of Kubernetes in the cluster
- Configure cluster parameters
- Configure TLS certificates to authenticate as admin

CoreOS

Not really a demo: Kubernetes deployment for real

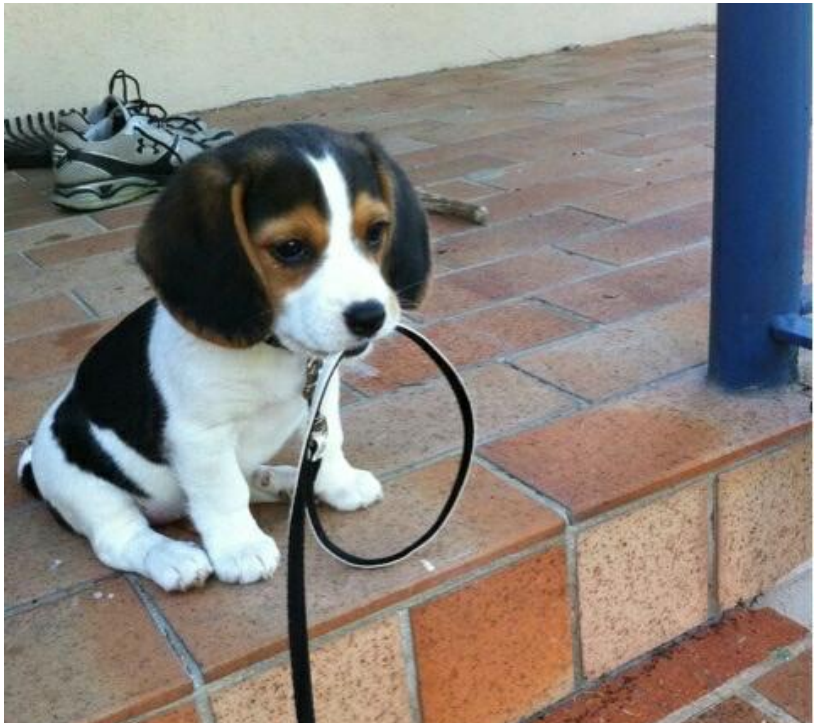
- Due to time slot constraints I'm not actually doing a Kubernetes deployment live
- Google has a nice little set of files in the Kubernetes project on GitHub which will easily bootstrap a basic cluster on GCE
  - What you're about to see is running on that
- If you go to the CoreOS website, we have a step-by-step generic guide to deploying it our way, which is what the preceding is directly based on
- So why didn't I do it the way my employer does it?

CoreOS

Because the point is...

- It doesn't matter. Kubernetes is Kubernetes.
  - Sort of (see previous slide about cloud integration)
- **This is a fundamental part of our philosophy re: Tectonic:** if your Kubernetes environment passes the conformance tests, we support running Tectonic in that environment, whatever it is

Demo: deploying a distributed app on Kubernetes



You can totally do this. The puppy believes in you!

Resources

Various links/other items of interest

CoreOS + Kubernetes Step By Step:

https://coreos.com/kubernetes/docs/latest/getting-started.html

Kubernetes Documentation:

http://kubernetes.io/v1.0/index.html

Kubernetes Guestbook example:

http://kubernetes.io/v1.0/examples/guestbook-go/README.html

Kubernetes deployment materials used in this demo:

https://github.com/kubernetes/kubernetes/blob/master/docs/getting-started-guides/coreos.md

Questions?