Click ‘engage’ to rate sessions and ask questions
Kubernetes
Changing the way we think and talk about computing

GOTO Berlin - December 2015
What is this talk?

Container
Brian Dorsey
Developer Advocate - Google Cloud platform

+BrianDorsey
@briandorsey
Containers? Yes/No

if yes GOTO slide 7;
if no GOTO slide 15;
Containers
The Old Way: Shared Machines

No isolation

No namespaces

Common libs

Highly coupled apps and OS
The Old Way: Virtual Machines

- Some isolation
- Expensive and inefficient
- Still highly coupled to the guest OS
- Hard to manage
The New Way: Containers

App specific isolation
Lightweight & efficient
Independent of the host
Linux distribution

... Lots of containers to manage
Container Images

- An image is a stack of Read-Only file system layers.

- Usual process:
  - build
  - push to repository
  - pull to execution host
  - start container from image
Image Layers

- A container is a process
  - started with kernel restrictions
  - a stack of shared Read-Only file system layers
  - plus a process specific Read-Write layer

- Every new container gets a new Read-Write later. All containers from the same image start from exactly the same state!
Mounting Host Directories

- It's possible to mount host directories into a container's filesystem.
- These are mutable and do outlive the container.
- They're only available on that host.
Why containers?

• Performance
• Repeatability
• Quality of service
• Accounting
• Portability

A fundamentally different way of managing applications
Containers are awesome!
Let’s run lots of them!
Kubernetes
Physical Computers

Virtual Machines

Container Clusters

PaaS

@briandorsey  #kubernetes  #GOTOber
Kubernetes

Greek for “Helmsman”; also the root of the words “governor” and “cybernetic”

- Runs and manages containers
- Inspired and informed by Google’s experiences and internal systems
- Supports multiple cloud and bare-metal environments
- Supports multiple container runtimes
- **100% Open source**, written in Go

Manage **applications**, not machines
Everything at Google runs in containers:

- Gmail, Web Search, Maps, ...
- MapReduce, batch, ...
- GFS, Colossus, ...
- Even Google’s Cloud Platform: VMs run in containers!
Everything at Google runs in containers:

- Gmail, Web Search, Maps, ...
- MapReduce, batch, ...
- GFS, Colossus, ...
- Even Google’s Cloud Platform: VMs run in containers!

We launch over 2 billion containers per week
A toolkit for running distributed systems in production

- co-locating helper processes
- mounting storage systems
- distributing secrets
- application health checking
- replicating application instances
- horizontal auto-scaling
- naming and discovery
- load balancing
- rolling updates
- resource monitoring
- log access and ingestion
- support for introspection and debugging
Start with a Cluster

**Laptop** to high-availability **multi-node cluster**

**Hosted** or **self managed**

**On-Premise** or **Cloud**

**Bare Metal** or **Virtual Machines**

Most OSes (inc. **RedHat Atomic, Fedora, CentOS**)

Or just a bunch of **Raspberry PIs**

Many options, See Matrix for details

Start with a Cluster
Setting up a cluster

- Choose a platform: GCE, AWS, Azure, Rackspace, Ubuntu, Juju ...
  - Then run:

    ```bash
    export KUBERNETES_PROVIDER=<your_provider>; curl -sS https://get.k8s.io | bash
    ```

- Or choose a distro such as RedHat Atomic, CoreOS Tectonic, Mirantis Murano (OpenStack), Mesos
- Or use a recipes for bare metal configurations for Centos, Fedora, etc
- Use a hosted option such as Google Container Engine
Deploy containers

$ kubectl run my-nginx --image=nginx --replicas=2 --port=80
A pod of whales containers

The atom of scheduling for containers

An application specific logical host

Hosts containers and volumes

Each has its own routable IP address (no NAT)

Ephemeral
  • Pods are functionally identical and therefore ephemeral and replaceable
Pods

Can be used to group multiple containers & shared volumes

Containers within a pod are **tightly** coupled

Shared namespaces
  • Containers in a pod share IP, port and IPC namespaces
  • Containers in a pod talk to each other through localhost
Pod Networking (across nodes)

Pods have IPs which are routable

Pods can reach each other without NAT
   Even across nodes

No Brokering of **Port Numbers**

These are fundamental requirements

Many solutions
   Flannel, Weave, OpenVSwitch, Cloud Provider
Create a service

$ kubectl expose rc my-nginx --port=80 --type=LoadBalancer
Services

A logical grouping of pods that perform the same function
  • grouped by label selector

Load balances incoming requests across constituent pods

Choice of pod is random but supports session affinity (ClientIP)

Gets a **stable** virtual IP and port
  • also a DNS name
Labels ← These are important

- Metadata with semantic meaning
- Membership identifier
- The only Grouping Mechanism

➔ Allow for intent of many users (e.g. dashboards)
➔ Build higher level systems ...
➔ Queryable by Selectors
Replication Controllers

**Behavior**
- Keeps Pods running
- Gives direct control of Pod #s
- Grouped by Label Selector

**Benefits**
- Recreates Pods, maintains desired state
- Fine-grained control for scaling
- Standard grouping semantics

#pods = 2
version = v1

#pods = 1
version = v2
Replication Controllers

Canonical example of control loops

Have one job: ensure N copies of a pod
- if too few, start new ones
- if too many, kill some
- group == selector

Replicated pods are fungible
- No implied order or identity

Replication Controller
- Name = “backend”
- Selector = {“name”: “backend”}
- Template = { ... }
- NumReplicas = 4

API Server
Managing Deployments
$ kubectl scale rc my-nginx --replicas=5
Scaling Example

Replication Controller
version = v1
type = FE
#pods = 4

Pod
version = v1
type = FE

Pod
version = v1
type = FE

Pod
version = v1
type = FE

Pod
version = v1
type = FE

Service
name = frontend
Label selector:
type = FE

@briandorsey  #kubernetes  #GOTOber
Canary

Replication Controller
version = v1
type = BE
#pods = 2

Service
name = backend
Label selector:
type = BE

Replication Controller
version = v2
type = BE
#pods = 1

Pod
version = v1
type = BE

Pod
version = v1
type = BE

Pod
version = v2
type = BE

@briandorsey  #kubernetes  #GOTObert
Rolling Update

$ kubectl rolling-update frontend --image=frontend:v2
Rolling Update

Replication Controller
version = v1
type = BE
#pods = 2

Service
name = backend
Label selector:
type = BE

Replication Controller
version = v2
type = BE
#pods = 2

Pod
version = v1
type = BE

Pod
version = v2
type = BE

Pod
version = v2
type = BE
Autoscale

$ kubectl autoscale rc frontend --min=1 --max=20
Pod Horizontal Autoscaling Beta (1.1)

Replication Controller
name=locust
role=worker
#pods = 4

Scale
CPU Target% = 50

Pod
name=locust
role=worker

Pod
name=locust
role=worker

Pod
name=locust
role=worker

Pod
name=locust
role=worker

70% CPU
greater than 50% CPU

40% CPU
less than 50% CPU

Heapster

≈ 50% CPU

@briandorsey  #kubernetes  #GOTOber
Managing State
I still have questions about state!

In a cluster of ephemeral containers
Application state must exist outside of the container
Volumes

Bound to the Pod that encloses it
Look like Directories to Containers
What and where they are determined by Volume Type
Many Volume options
- EmptyDir
- HostPath
- nfs, iSCSI (and similar services)
- Cloud Provider Block Storage
Outside the Cluster

e.g.: MySQL managed by DBAs or managed cloud services

@briandorsey  #kubernetes  #GOTOber
Adapt to run in the Cluster

e.g.: MySQL runs in a pod and mounts a filesystem provided by the cluster
Cluster Native

ex: run Cassandra or Riak inside the cluster
Cluster native - MySQL on Vitess

Open source MySQL scaling solution

Vitess has been serving all YouTube database traffic since 2011

Replication, dynamic sharding, caching and more

Designed for a distributed, containerized world

Kubernetes configs included

http://vitess.io/

@briandorsey  #kubernetes  #GOTOber
Secrets

Problem: how to grant a pod access to a secured something?
- don’t put secrets in the container image!

12-factor says: config comes from the environment
- Kubernetes is the environment

Manage secrets via the Kubernetes API

Inject them as virtual volumes into Pods
- late-binding
- tmpfs - never touches disk
Wrap-up
Kubernetes status & plans

Open sourced in June, 2014

v1.0 in July, 2015, v1.1 in November 2015

Google Container Engine (GKE)
- hosted Kubernetes - don’t think about cluster setup
- GA in August, 2015

PaaSes:
- RedHat OpenShift, Deis, Stratos

Distros:
- CoreOS Tectonic, Mirantis Murano (OpenStack), RedHat Atomic, Mesos

Working towards a 1.2 release

@briandorsey  #kubernetes  #GOTOber
Google Container Engine (GA) -- Demo

Managed Kubernetes (Kubernetes v1.1)
Manages Kubernetes master uptime
Manages Updates
Cluster Resize via Managed Instance Groups
Cluster Node Autoscaling
Centralized Logging
Google Cloud VPN support
Kubernetes is Open Source
We want your help!

http://kubernetes.io
https://github.com/GoogleCloudPlatform/kubernetes
Slack: #kubernetes-users
@kubernetesio
Your app?

Physical Computers
Virtual Machines
Container Clusters
PaaS
Tweet questions afterwards to: @briandorsey

Slides: goo.gl/NI1GaM

Questions
Please remember to rate the session.

Thank you!