Managing Compute and Storage at Scale with Kubernetes

Dan Paik / Google
WHO ARE WE?
CUSTOMERS!
WHAT DO WE WANT?
LARGE, SCALABLE & RELIABLE CLUSTERS!
Have You Recently ...

... played a hit mobile game?

... shopped at an online marketplace?

... followed breaking news?

... attended a concert?

... filed expenses?
Have You Recently ...

... played a hit mobile game?

... shopped an online marketplace?

... followed breaking news?

... attended a concert?

... filed expenses?
Fast, Scalable, Open

- **Fast: Developer productivity**
  - Minutes from commit to prod
  - Release 20-50x/day

- **Scalable: efficient scale out**
  - Fastest app to $1B
  - Black Friday demand

- **Open: use anywhere**
  - 200 warehouses on VMware
  - Hybrid and multi-cloud
Google has been developing and using containers to manage our applications for over 12 years.
Everything at Google runs in containers:

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

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

We launch over **2 billion** containers per week
But it’s all so different!

- Deployment
- Management, monitoring
- Isolation (very complicated!)
- Updates
- Discovery
- Scaling, replication, sets

A **fundamentally different** way of managing applications requires different tooling and abstractions.
Why Containers?

Containers make operations easier

Enabled Google to grow our fleet over 10x faster than we grew our ops team
Needs to run anywhere

- On-Premises
- Hybrid
- Cloud
Kubernetes

Open Source, Run Anywhere, Container-centric Infrastructure, for managing containerized applications across multiple hosts.
- Auto-scaling, rolling upgrades, A/B testing ...

Commercial Enterprise Support. Service Partners

Inspired by Google’s experiences and internal systems (blog post, research paper)

1,050+ Contributors
43,000+ Commits
4,000+ External Projects Based on Kubernetes
200+ Meetups Around the World
Velocity

Commits Since July 2014

- Total Commits: 1.6
- Commits Since July 2014: 1.4

Graph showing the increase in total commits from 7/1/2014 to 1/1/2017.
Can it scale?
How Can We Scale Out Container Workloads?

- How to handle replication?
- What about node failure?
- What about container failure?
- How do we manage application upgrades?
Kubernetes Scalability

- Scalability - exponentially more nodes and pods
- Multi-cluster federation
- HA masters
- Monitoring
- Global
- Enhanced resource management and isolation
- Hybrid
- Multi-cloud
This year, our customers flourished during Black Friday and Cyber Monday with zero outages, downtime or interruptions in service thanks, in part, to Google Container Engine and Kubernetes

Will Warren, Chief Technology Officer at GroupBy
Cloud Datastore Transactions Per Second

- **50X**: Actual traffic
- **5X**: Worst case estimate
- **1X**: Target traffic

- Orange: Original launch target
- Green: Estimated worst case
- Blue: Actual traffic
Scaling best practices
Pod

**Small group** of containers & volumes

Tightly coupled

**The atom** of Kubernetes

Shared ip, networking, lifecycle, disk
Horizontal Pod Autoscaling

Automatically add (or remove) pods as needed
- Based on CPU utilization (for now)
- Custom metrics in Alpha

- Efficiency now, capacity when you need it
- Operates within user-defined min/max bounds
- Set it and forget it
Cluster autoscaling

Add Nodes/VMs when needed

- Based on unschedulable pods
- New VMs self-register with API server

Remove Nodes/VMs when not needed

  e.g. CPU usage too low
Performance & scalability

ONE DOES NOT SIMPLY START LARGE CLUSTERS

100 nodes
v1.0 07/2015
↓
250 nodes
v1.1 11/2015
↓
1000 nodes
v1.2 03/2016
↓
2000 nodes
v1.3 07/2016
↓
5000 nodes
v1.6 03/2017
↓
...

Google
Hybrid / Multi-Cloud
Build for a Hybrid/Multi-Cloud World

Public Cloud

Private Cloud

Open Source & APIs
Hybrid Cloud

- An elastic computing environment
- On-demand self service for users
- Network delivered services

- Secure network connectivity
- Data security
- Operational management and workflow
Goal: Avoid vendor lock-in

Runs in many environments, including “bare metal” and “your laptop”

The API and the implementation are 100% open

The whole system is modular and replaceable
Goal: Write once, run anywhere*

Don’t force apps to know about concepts that are cloud-provider specific

Examples of this:

- Network model
- Ingress
- Service load-balancers
- PersistentVolumes

*mostly approximately

Workload portability
Goal: Avoid coupling

Don’t force apps to know about concepts that are Kubernetes-specific

Examples of this:

- Namespaces
- Services / DNS
- Downward API
- Secrets / ConfigMaps
Result: Portability

Build your apps on-prem, lift-and-shift into cloud when you are ready

Don’t get stuck with a platform that doesn’t work for you

Put your app on wheels and move it whenever and wherever you need
Storage
Problem

Can’t share files between containers

Files in containers are ephemeral
  Can’t run stateful applications
  Container crashes result in loss of data
Volumes

Kubernetes Volumes

Directory, possibly with some data

Accessible by all containers in pod
Lifetime same as the pod or longer

Volume Plugins Define
• How directory is setup
• Medium that backs it
• Contents of the directory
### Volumes

Kubernetes has many volume plugins

<table>
<thead>
<tr>
<th>Persistent</th>
<th>Ephemeral</th>
<th>Other</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCE Persistent Disk</td>
<td>Empty dir (and tmpfs)</td>
<td>Flex (exec a binary)</td>
<td>Local Storage?</td>
</tr>
<tr>
<td>AWS Elastic Block Store</td>
<td>Expose Kubernetes API</td>
<td>Host path</td>
<td></td>
</tr>
<tr>
<td>Azure File Storage</td>
<td>• Secret</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azure Data Disk</td>
<td>• ConfigMap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iSCSI</td>
<td>• DownwardAPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flocker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vSphere</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GlusterFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ceph File and RBD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cinder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quobyte Volume</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FibreChannel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VMWare Photon PD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
apiVersion: v1
kind: Pod
metadata:
  name: sleepypod
spec:
  volumes:
    - name: data
gcePersistentDisk:
    pdName: panda-disk
    fsType: ext4
  containers:
    - name: sleepycontainer
      image: gcr.io/google_containers/busybox
      command:
        - sleep
        - "6000"
      volumeMounts:
        - name: data
          mountPath: /data
          readOnly: false
apiVersion: v1
type: Pod
metadata:
  name: sleepypod
spec:
  volumes:
  - name: data
gcePersistentDisk:
    pdName: panda-disk
    fsType: ext4
  containers:
  - name: sleepycontainer
    image: gcr.io/google_containers/busybox
    command:
      - sleep
      - "6000"
    volumeMounts:
      - name: data
        mountPath: /data
        readOnly: false
apiVersion: v1
kind: Pod
metadata:
  name: sleepypod
spec:
  volumes:
  - name: data
gcePersistentDisk:
    pdName: panda-disk
    fsType: ext4
  containers:
  - name: sleepycontainer
    image: gcr.io/google_containers/busybox
    command:
      - sleep
      - "6000"
    volumeMounts:
    - name: data
      mountPath: /data
      readOnly: false
apiVersion: v1

kind: Pod

metadata:
  name: sleepypod

spec:
  volumes:
    - name: data
      gcePersistentDisk
        pdName: panda-disk
        fsType: ext4

  containers:
    - name: sleepycontainer
      image: gcr.io/google_containers/busybox
      command:
      - sleep
      - "6000"

      volumeMounts:
      - name: data
        mountPath: /data
        readOnly: false

Volume referenced directly

GCE PD Example

If you directly reference a pod
You're gonna have a bad time
PersistentVolumes

A higher-level storage abstraction insulation from any one cloud environment

Admin provisions them, users claim them

Static and dynamic provisioning (via StorageClass)

Independent lifetime from consumers lives until user is done with it can be handed-off between pods
Walkthrough

Cluster Admin
apiVersion: v1
kind: PersistentVolume
metadata:
  name: myPV1
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    fsType: ext4
    pdName: panda-disk

---

apiVersion: v1
kind: PersistentVolume
metadata:
  name: myPV2
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 100Gi
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    fsType: ext4
    pdName: panda-disk2
$ kubectl create -f pv.yaml
persistentvolume "pv1" created
persistentvolume "pv2" created

$ kubectl get pv

<table>
<thead>
<tr>
<th>NAME</th>
<th>CAPACITY</th>
<th>ACCESSMODES</th>
<th>STATUS</th>
<th>CLAIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv1</td>
<td>10Gi</td>
<td>RWO</td>
<td>Available</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1m</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pv2</td>
<td>100Gi</td>
<td>RWO</td>
<td>Available</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1m</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cluster Admin creates PersistentVolumes.
Cluster Admin

User

PersistentVolumes
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
  namespace: testns
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
$ kubectl create -f pv.yaml
persistentvolume "pv1" created
persistentvolume "pv2" created

$ kubectl get pv
<table>
<thead>
<tr>
<th>NAME</th>
<th>CAPACITY</th>
<th>ACCESSMODES</th>
<th>STATUS</th>
<th>CLAIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv1</td>
<td>10Gi</td>
<td>RWO</td>
<td>Available</td>
<td></td>
</tr>
<tr>
<td>pv2</td>
<td>100Gi</td>
<td>RWO</td>
<td>Available</td>
<td></td>
</tr>
</tbody>
</table>

$ kubectl create -f pvc.yaml
persistentvolumeclaim "mypvc" created

$ kubectl get pv
<table>
<thead>
<tr>
<th>NAME</th>
<th>CAPACITY</th>
<th>ACCESSMODES</th>
<th>STATUS</th>
<th>CLAIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv1</td>
<td>10Gi</td>
<td>RWO</td>
<td>Available</td>
<td></td>
</tr>
<tr>
<td>pv2</td>
<td>100Gi</td>
<td>RWO</td>
<td>Bound</td>
<td>testns/mypvc</td>
</tr>
</tbody>
</table>
Volume no longer referenced directly

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sleepypod
spec:
  volumes:
  - name: data
gcePersistentDisk:
    pdName: panda-disk
    fsType: ext4
  containers:
  - name: sleepycontainer
    image: gcr.io/google_containers/busybox
    command:
      - sleep
      - "6000"
    volumeMounts:
      - name: data
        mountPath: /data
        readOnly: false
```

```
 volumes:
 - name: data
   persistentVolumeClaim:
     claimName: mypvc
```
Dynamic provisioning for scalability

- Allows storage to be created on-demand (when requested by user)
- Eliminates need for cluster administrators to pre-provision storage
- Great for public cloud - API can provision new storage
- Also works on-prem - trigger new NFS share on demand
Dynamic provisioning

- Cluster/Storage admins "enable" dynamic provisioning by creating StorageClass objects.
- StorageClass objects define the parameters used during creation.
- StorageClass parameters are opaque to Kubernetes so storage providers can expose any number of custom parameters for the cluster admin to use.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
```

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```
Dynamic provisioning

- Users consume storage the same way: PVC
- “Selecting” a storage class in PVC triggers dynamic provisioning

pvc.yaml (on k8s 1.5)

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
  namespace: testns
annotations:
  volume.beta.kubernetes.io/storage-class: fast
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
```

pvc.yaml (on k8s 1.6+)

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
  namespace: testns
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: fast
```
Thank You!

danpaik@google.com