CLOUD NATIVE
COMPUTING FOUNDATION

What's New in Kubernetes 1.10

Presenters

- Ihor Dvoretskyi, CNCF, ihor@cncf.io - Release Team Member, 1.10 Release Features Lead, SIG-PM Lead
- Saad Ali, Google, saadali@google.com - SIG-Storage Lead
- Michelle Au, Google, msau@google.com - SIG-Storage Member

Agenda

- Kubernetes 1.10 highlights
- Container Storage Interface (CSI) Beta
- Local Persistent Volumes Beta
- Q&A

Kubernetes 1.10 Highlights

Kubernetes 1.10

- The first release in 2018
- 25 new features!
- Community focus on enhancing the existing features
- The major areas of enhancement are:
  - Workloads
  - Security
  - Storage

Container Storage Interface (CSI) Beta

CSI in Kubernetes is beta!

Kubernetes implementation of Container Storage Interface (CSI) beta in Kubernetes v1.10

Introduced as alpha in Kubernetes v1.9

Kubernetes Volume Plugins

- Kubernetes has powerful volume plugin system
- Makes it easy to use block and file storage
- Challenging adding support for new “in-tree” volume plugins
  - Requires checking code in to Kubernetes and aligning with the Kubernetes release process to add support or fix a bug in plugin.

Container Storage Interface (CSI) Specification

- Spec published on GitHub
- CSI v0.2 released in February 2017
- Attempt to define standard for:
  - Container Orchestration Systems (COs) to expose arbitrary storage systems to containerized workloads.
  - Storage Providers (SPs) to write one CSI compliant Plugin that "just works" across all COs that implement CSI.

CONTAINER
STORAGE
INTERFACE

https://github.com/container-storage-interface

Why CSI?

- CSI makes Kubernetes volume layer truly extensible.

How to use a CSI Volume?

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast-storage
provisioner: com.example.csi-driver
parameters:
  type: pd-ssd
  csiProvisionerSecretName: mysecret
  csiProvisionerSecretNamespace: mynamespace
------
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-request-for-storage
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: fast-storage
```

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod
spec:
  containers:
    - name: my-frontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: my-csi-volume
  volumes:
    - name: my-csi-volume
      persistentVolumeClaim:
        claimName: my-request-for-storage
```

Pre-provisioned volumes?

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-manually-created-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: com.example.csi-driver
    volumeHandle: existingVolumeName
    readOnly: false
    fsType: ext4
    volumeAttributes:
      foo: bar
    controllerPublishSecretRef:
      name: mysecret1
      namespace: mynamespace
    nodeStageSecretRef:
      name: mysecret2
      namespace: mynamespace
    nodePublishSecretRef
      name: mysecret3
      namespace: mynamespace
```

What's new in Beta?

- Enabled by default
- Compatible with CSI spec v0.2
  - Warning: Breaking changes between v0.1 to v0.2; existing CSI drivers must be updated to be 0.2 before use with Kubernetes 1.10.0+.
- Mount propagation also moved to beta.
- VolumeAttachment object added to storage v1beta1 group.
- VolumeAttributes field added to CSIPersistentVolumeSource object (was passed via annotations)
- Node authorizer now limits access to VolumeAttachment objects from kubelet
- Secrets can now be referenced in CSIPersistentVolumeSource object
- FS type can now be specified in CSIPersistentVolumeSource object
- New (optional) NodeStageVolume call added to the CSI corresponds to MountDevice operation in Kubernetes (in alpha this step was a no-op).

What's next?

- Block volumes support
- Storage availability topology awareness
- Kubelet Device Plugin Integration
- Snapshot
- Targeting GA of CSI implementation in v1.12.
- Storage vendors: start developing CSI drivers!
- Kubernetes.io blog post out today!

Local Persistent Volumes Beta

Feature Overview

- Specify a local disk as a PersistentVolume (PV)
- Works best with StatefulSets
- Kubernetes scheduler enhancements
  - Pod always gets scheduled to the node where the local disk resides (data gravity)
  - Initial PersistentVolumeClaim (PVC) binding considers Pod scheduling constraints (resources, anti-affinity, taints, etc.) and multiple PVCs in a Pod

Caveats

- Pod is stuck if node/disk is unavailable
- Pod must fit on specific node (harder to schedule)
- Local disks in many cloud providers have lower durability
- **Local PV not suitable for most workloads!**

Use Cases

- Workload must tolerate node/data unavailability and data loss
- Caching to SSD
  - Data gravity to avoid cold restarts
- Distributed storage systems
  - Shards and replicates data across multiple nodes

How to Use

1. **Admin**: Create a StorageClass
2. **Admin**: Create local PVs
3. **User**: Create StatefulSet (with PVCs)

Complete documentation:

https://kubernetes.io/docs/concepts/storage/volumes/#local

1. Create a StorageClass

- Enable smarter PVC binding
- Dynamic provisioning not supported yet

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

2a. Manually Manage a Local PV

- Path is global mount point
- Disk must be preformatted and mounted beforehand!
- Only “Retain” reclaim policy supported for manual creation

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-local-pv
spec:
  capacity:
    storage: 500Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /mnt/disks/vol1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values: my-node
```

2b. Automatically Manage Local PVs

1. Preformat and mount local disks under a directory per StorageClass (node initialization)
2. Run the external local volume STATIC provisioner
    a. It detects all the mount points under the directory and creates PVs for them
    b. When PV is released, it cleans up the disk, and deletes PV
    c. Go to a)

3. Create a StatefulSet

- Process is exactly the same!
- Specify the corresponding StorageClass

```
kind: StatefulSet
...
 volumeClaimTemplates:
  - metadata:
      name: example-local-claim
    spec:
      accessModes:
      - ReadWriteOnce
      storageClassName: local-storage
      resources:
        requests:
          storage: 500Gi
```

Future Work

- Local raw block volumes (1.10 alpha)
- Dynamic provisioning using LVM
- Local provisioner plugins for node setup
- Join us in SIG-Storage!

How to get involved?

- Kubernetes Storage Special Interest Group (SIG)
  - Meetings 9 AM PT Thursdays every two weeks
    - Details: kubernetes/community/sig-storage
    - Google Group: kubernetes-sig-storage
    - Slack: https://kubernetes.slack.com/messages/sig-storage

- Presentations at Kubecon EU:
  - SIG Storage Intro
  - Kubernetes Storage Lingo 101
  - Using Kubernetes Local Storage for Scale-Out Storage Services in Production
  - Container Storage Interface: Present and Future
  - Policy-Based Volume Snapshots Management in Kubernetes

Q&A

Thank you!