SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

# Re-Imagining the 3-2-1 Backup Rule for Cloud Native Applications Running on Kubernetes

Jagadish Mukku
Technical Director, Robin.io

# Agenda

- 3-2-1 rule introduction
- Data protection methods
- Kubernetes data protection and 3-2-1 rule
- ROBIN storage architecture for data protection
- Demo

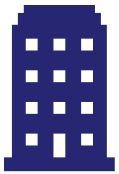
# 3-2-1 Rule of Backups

3 Copies

2 Copies Onsite
(different media)

Data Center

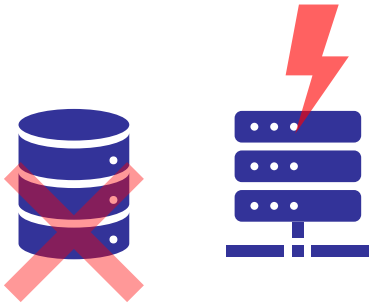
1 Copy Offsite



Offsite Data Center

Cloud

# Data Protection



Hardware Failures,
Natural Disasters

Manual Errors

Malicious Activity
(e.g ransomware)

# Data Protection With Copies

- Multiple copies of same data blocks
- Protects from hardware(disk, node) failures
- Copies can be at different layers
  - ✓ Disk (RAID-x)
  - ✓ Volume
  - ✓ File system
  - ✓ Application

# Data Protection With Snapshots

- Data state at a point in time
- Support at volume, filesystem, application
- Implemented as:
  - ✓ COW (Copy-On-Write),
  - ✓ Redirect-On-Write
- Rollback to snapshot on
  - ✓ Manual errors
  - ✓ Upgrade issues
  - ✓ Data corruption

# Data Protection With Backups

- Point in time copy of app
- Stored on separate media, hardware, location
- Independent life cycle than an app

- ✓ App restore
- ✓ Disaster recovery
- ✓ Test &dev
- ✓ Analytics
- ✓ Upgrade testing

# Methods to take backups

- Application tooling
  - ✓ App specific methods to take backups
    - e.g *mysqldump* for Mysql, *nodetool* for Cassandra, *mongodump* for MongoDB
  - Logical Backup
  - Physical Backup
- Backup software
  - ✓ Infrastructure coordinated (storage mgmt software)
  - ✓ Generate diffs independently (e.g rsync, restic) or call API's for changed blocks list
  - ✓ May co-ordinate with file system or storage drivers
  - ✓ Uniform backup operations across applications

# Data Accessibility Of Copies

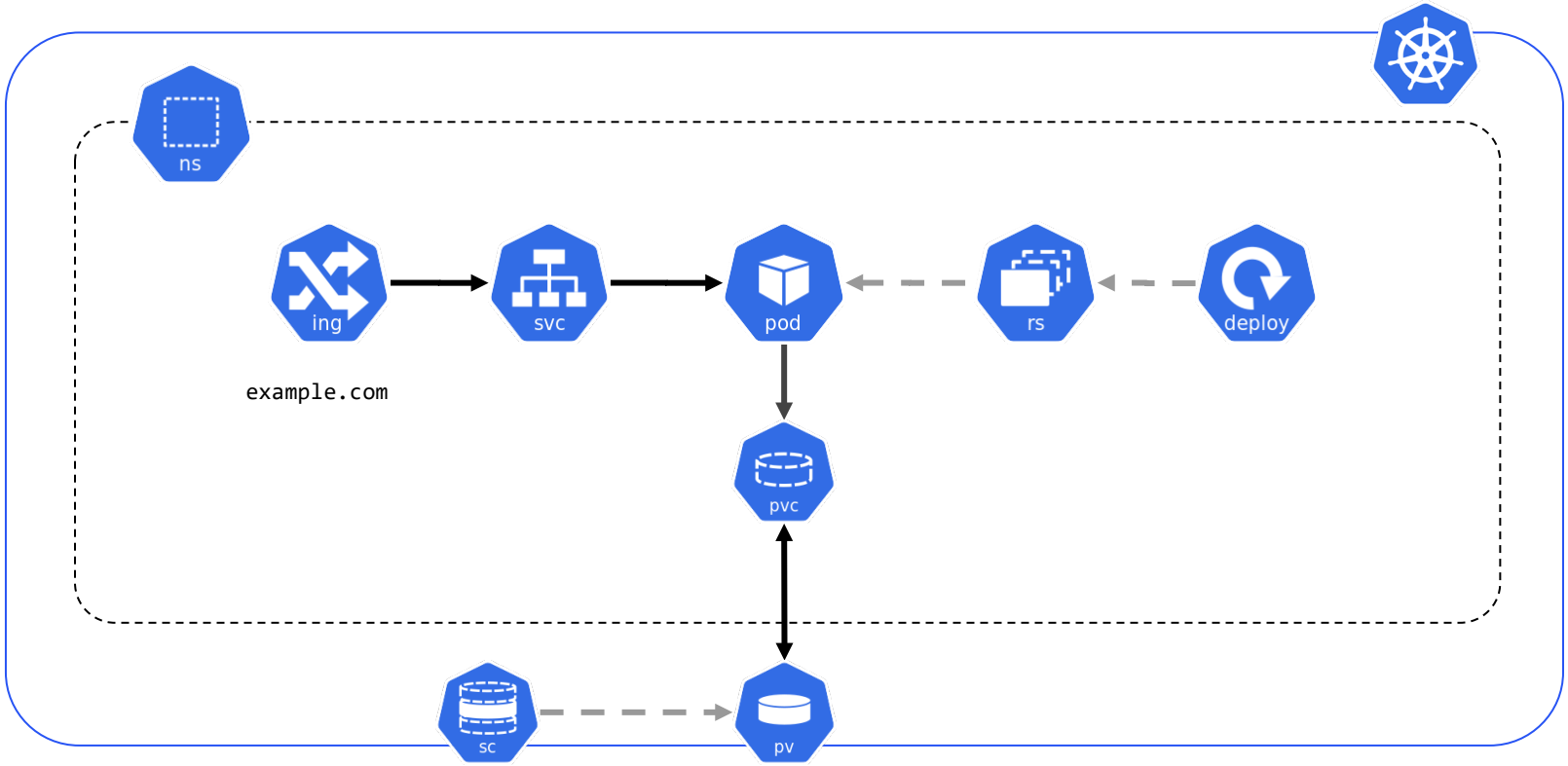
- Storage snapshots
  - In seconds to minutes
- Data mirrors
  - In milli seconds to minutes
- Restore from Backup
  - In minutes to hours

# Copy Time Interval & Resources

**How often does the copy occur?**

**How long does it take to recover from copy?**

# Cloud native application



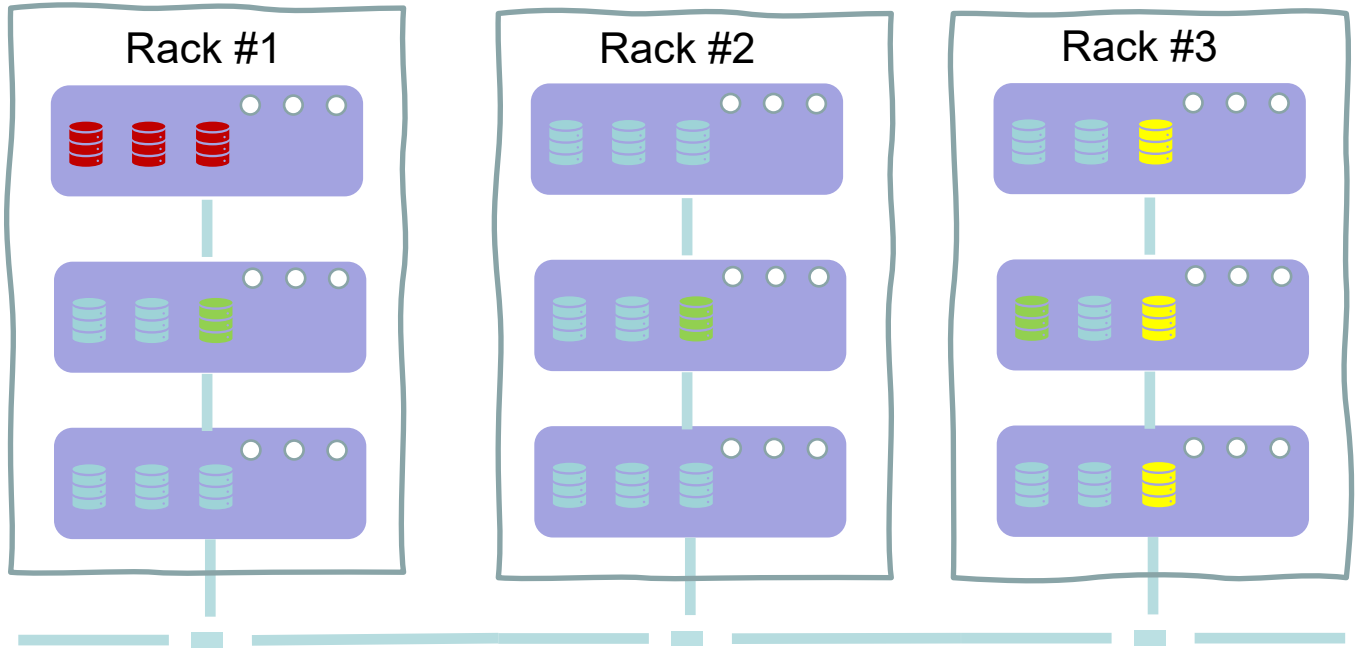
# Persistent Volume Claim

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: mypvc
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: robin
  resources:
    requests:
      storage: 10Gi
```

# Storage Placement Policy

Request: Allocate volume with three copies that are rack failure tolerant.




No Node failure tolerance ✗

No rack failure tolerance ✗

Node, Rack failure tolerant ✓

MySQL

PostgreSQL

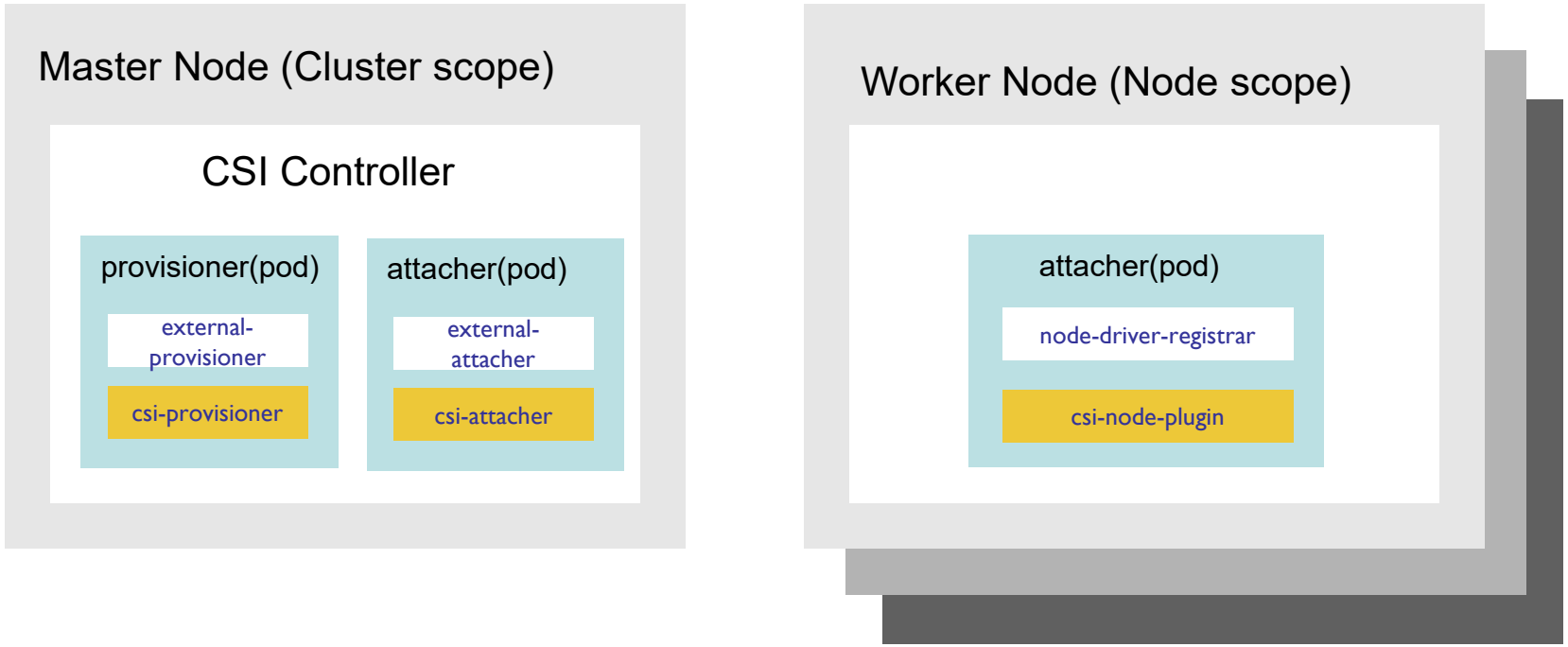
# Storage Placement

**Right storage placement is key for essential data protection**

# Kubernetes Storage Allocation

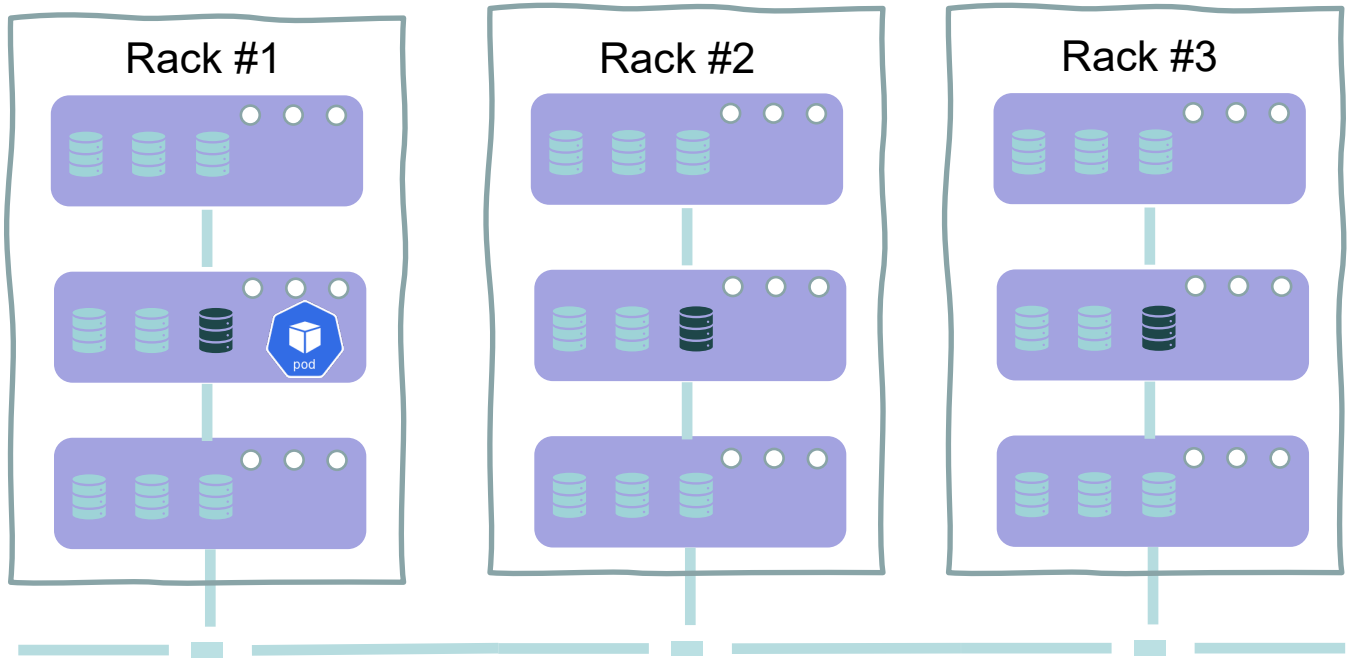
- Container Storage Interface (CSI)
  - Interface for k8s to expose storage systems to containers
  - Used for dynamic provisioning of volumes
  - Provides interface for volume snapshots, clones

# Kubernetes CSI Components




storage provider

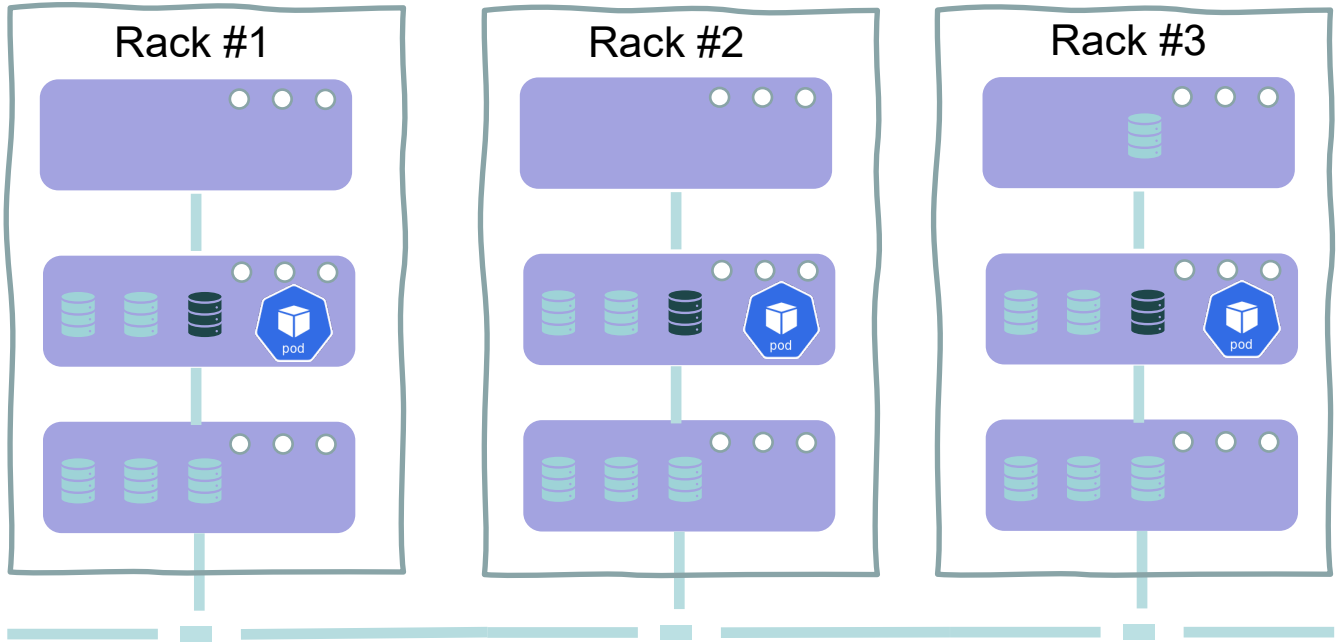
# Copy By Storage Replication



# PVC With Storage Replication

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: protected-pvc
  annotations:
    robin.io/replication: "3"
    robin.io/faultdomain: host
spec:
  storageClassName: robin
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

# Copy By Application Replicas




mongoDB

cassandra

# PVC with App Replicas

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
  labels:
    app: cassandra
spec:
  serviceName: cassandra
  replicas: 3
  template:
    spec:
      containers:
      - name: cassandra
        image: gcr.io/google-samples/cassandra:v13
        imagePullPolicy: Always
        resources:
          limits:
            cpu: "500m"
            memory: 1Gi
          requests:
            cpu: "500m"
            memory: 1Gi
        # These volume mounts are persistent. They are like inline claims,
        # but not exactly because the names need to match exactly one of
        # the stateful pod volumes.
        volumeMounts:
        - name: cassandra-data
          mountPath: /cassandra_data
  # These are converted to volume claims by the controller
  # and mounted at the paths mentioned above.
  volumeClaimTemplates:
  - metadata:
      name: cassandra-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: robin
      resources:
        requests:
          storage: 100Gi
```

```yaml
kind: StatefulSet
metadata:
spec:
  replicas: 3
...
```

```yaml
volumeClaimTemplates:
 - metadata:
     name: cassandra-data
   spec:
     .......
     resources:
       requests:
         storage: 100Gi
```

# Storage And Pod Allocation

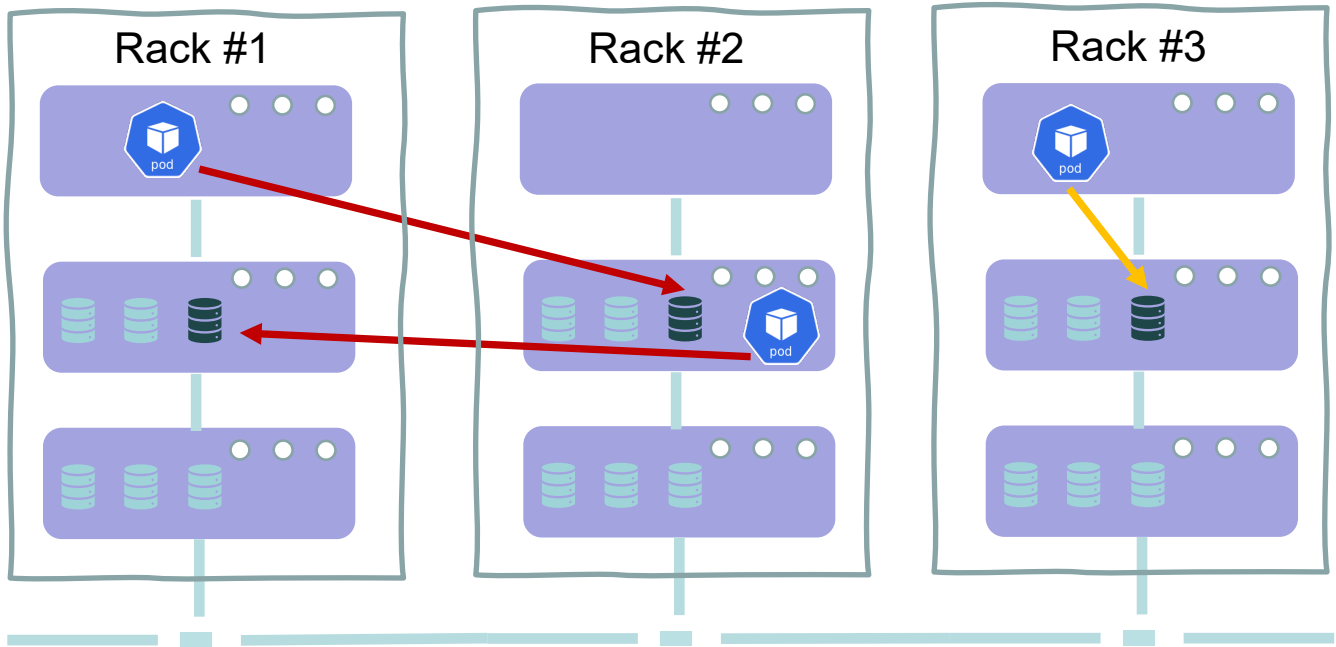


**Scheduler picks a node for pod to run**

**Storage class (CSI) allocates storage**

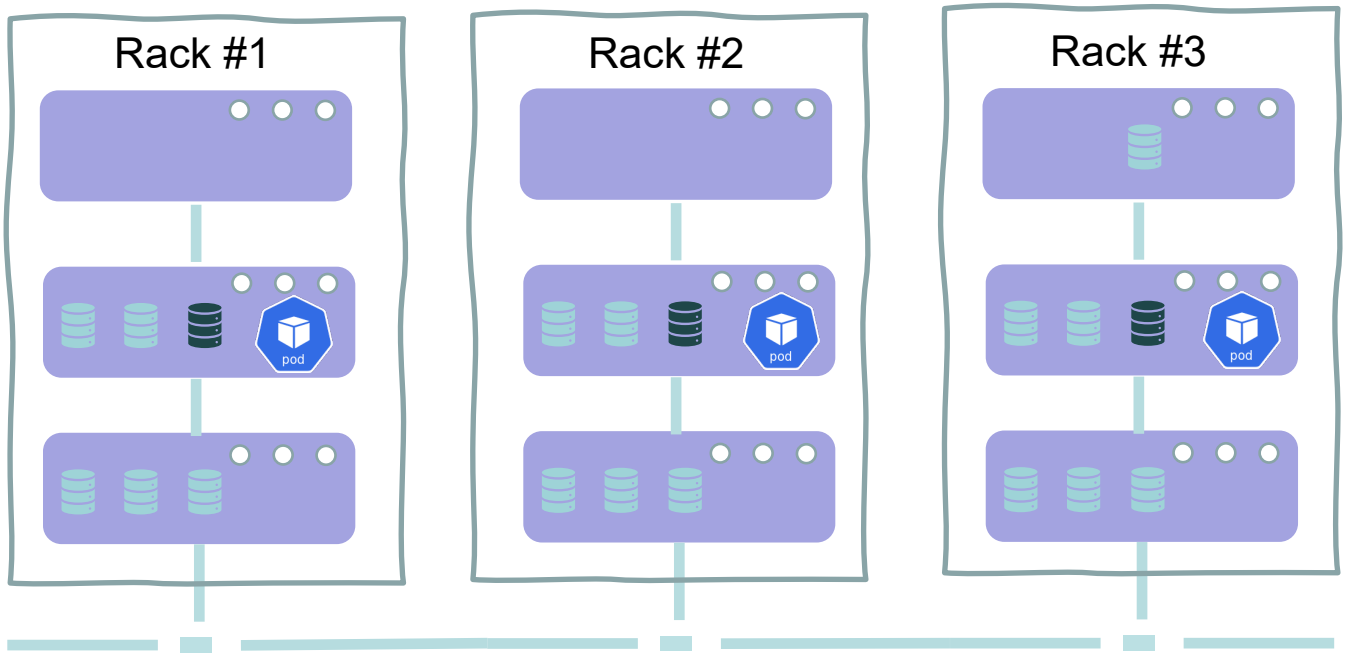
# Copy By Application Replicas

**What we want: Pod and Volume need to be rack failure tolerant**



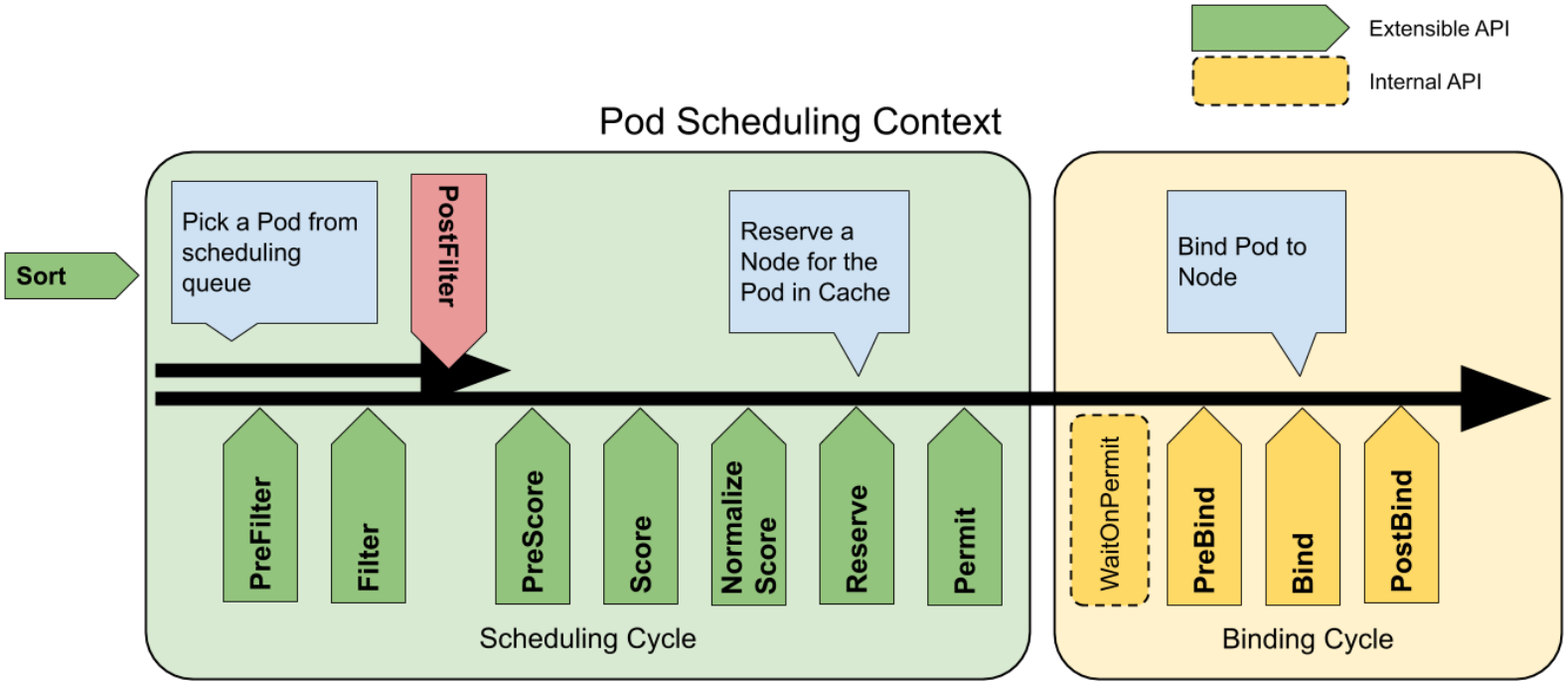

**Non-optimal allocations when pod scheduler and storage class allocate independently**

# Copy By Application Replicas




**Better when pod scheduler & storage class co-ordinate**

# Kubernetes Scheduling Framework Extension Points




picture from https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/

# Kubernetes Scheduler Extender

- Scheduler Extension
  - Custom Predicates in scheduling cycle
    - Filter out nodes that cannot run a Pod
      (e.g filter nodes that have no free storage, does not satisfy data protection requirement)
  - Priorities in the scheduling cycle
    - Rank the nodes
- Configurable webhooks
  - Add additional config during admission
- Pod affinity, anti-affinity, taints and tolerations

# Volume Binding Mode

- Immediate (default)
  - ✓ Provisioning and binding is done before pod is scheduled
- WaitForFirstConsumer
  - ✓ Provisioning and binding is done after pod using the volume is created

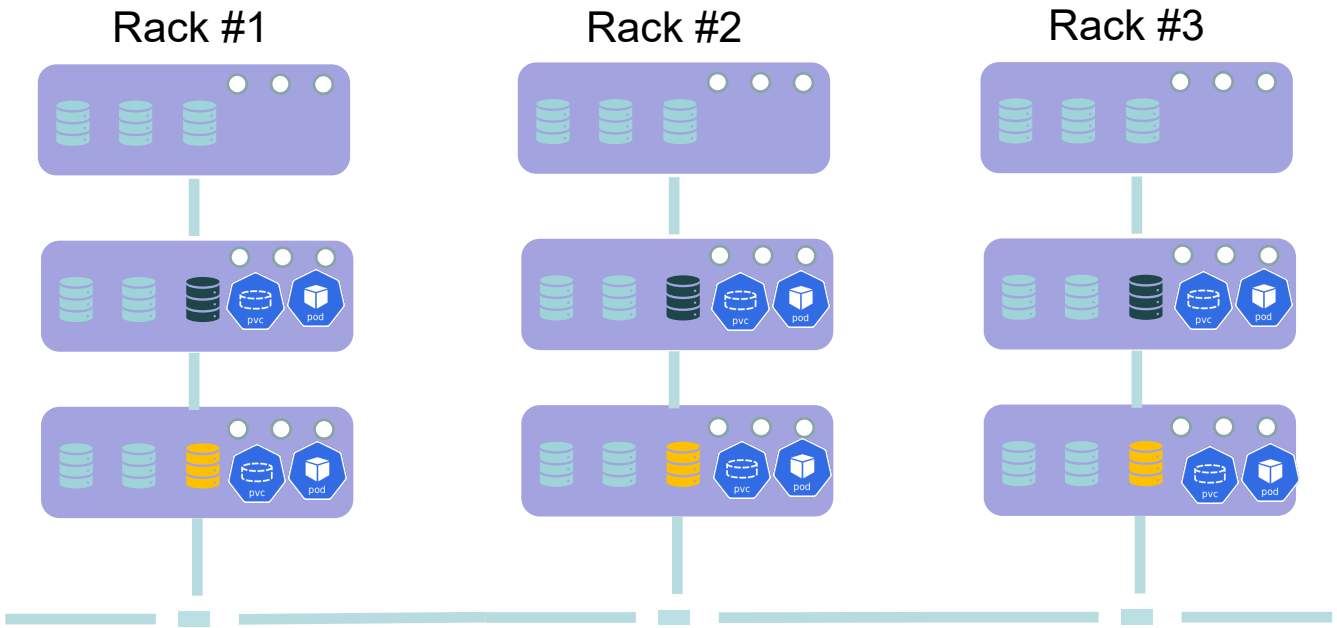
# Kubernetes Custom Scheduler

- Default scheduler works well with most of the cases
- Custom scheduler is an option if the current extensible points do not suffice
- Multiple schedulers can co-exist simultaneously

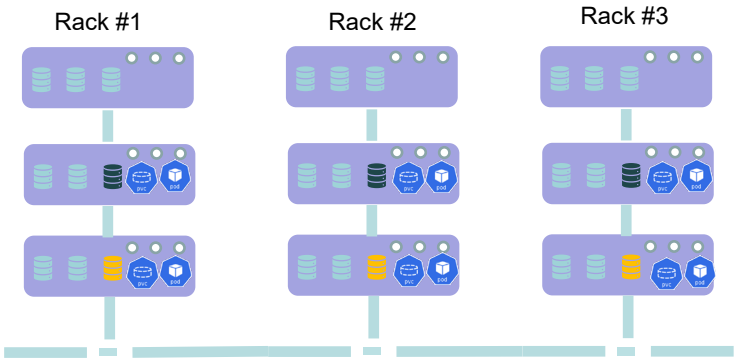
# Snapshots

- Snapshot is a **point-in-time copy**
- Maintains multiple versions of data on same media
- Most snapshots are implemented at certain block size (~4k, 32k, 1M)
- Used to rollback, act as a source for another app/volume (thin clone)

# Cloud-native app spanning nodes

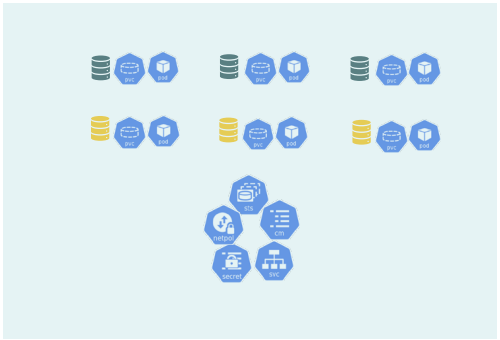


# Snapshots of Cloud-native apps

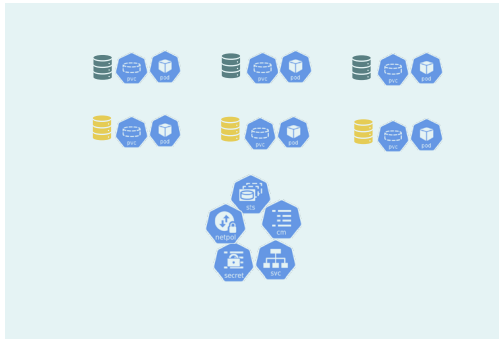



time (t)

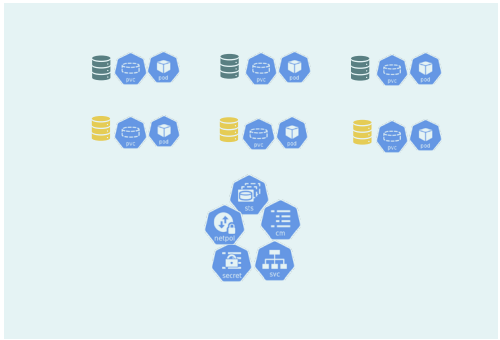
Snapshot #1



Snapshot #2



Snapshot #3



# What's needed for snapshot?

- Consistency volume groups
  - Needed for app consistency(e.g data, log volume should be at same points for a single snapshot.
  - Quick quiesce/unquiesce of volume group
- Capture application configuration
  - All k8s objects that belong to app.
- Call application specific flush commands
  - Snapshot pre/post snapshot commands
  - They make app consistent snapshots and backups

# Kubernetes snapshot primitives

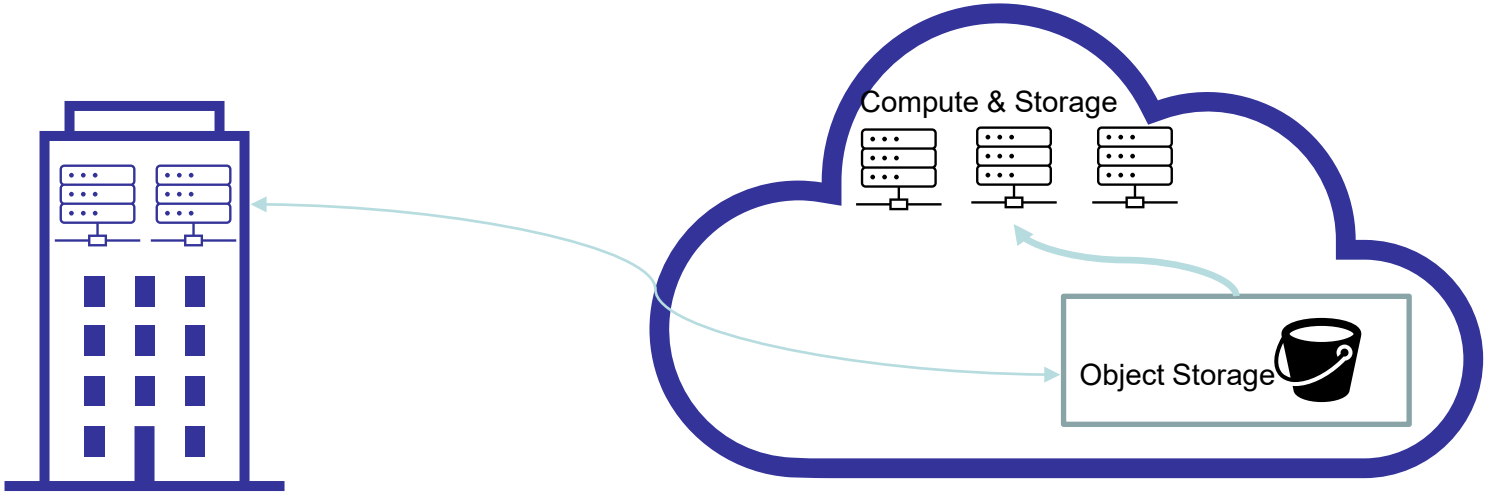
```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: new-snapshot-test
spec:
  volumeSnapshotClassName: robin-snapshotclass
  source:
    persistentVolumeClaimName: pvc-test
```

- Kubernetes has no interface to take volume group snapshot (as of version 1.17)
- Can be done with storage vendor interfaces

# Cloud Storage For Backups

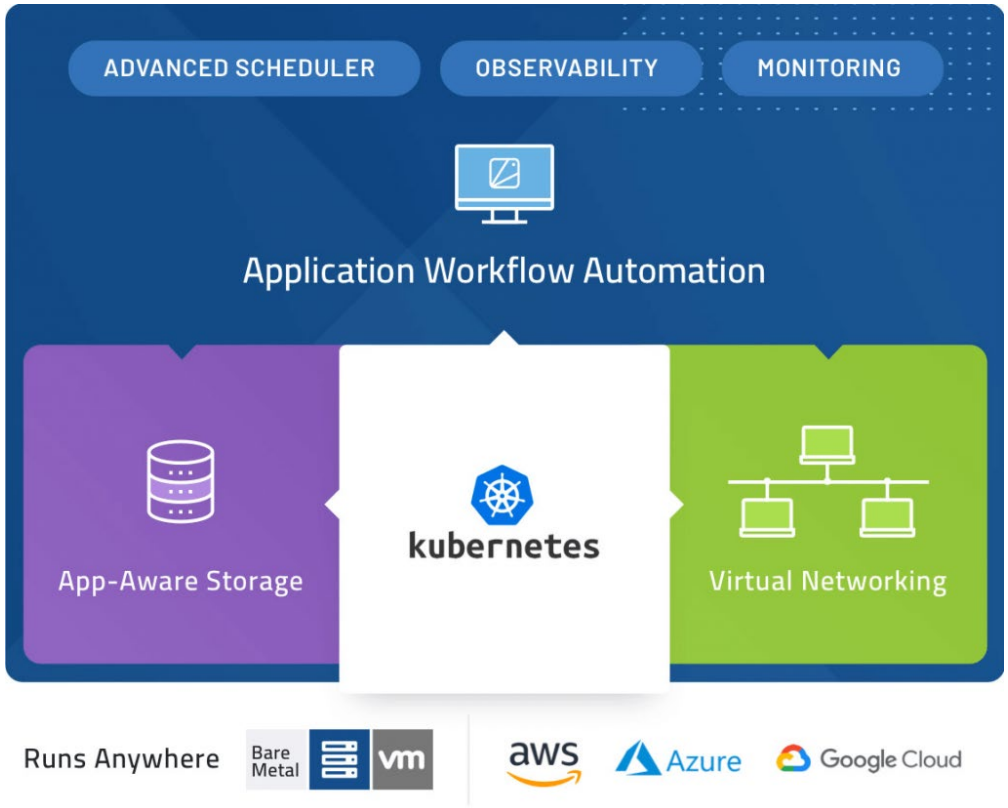
- Cloud object Storage
  - ✓ AWS S3, Glacier..
  - ✓ Google nearline, cold line, archive storage
  - ✓ Azure blobs
- On-premise (private cloud) object store
  - ✓ Minio
  - ✓ Ceph object storage

# Backups for …




- Disaster Recovery
- Analytics
- Test-dev

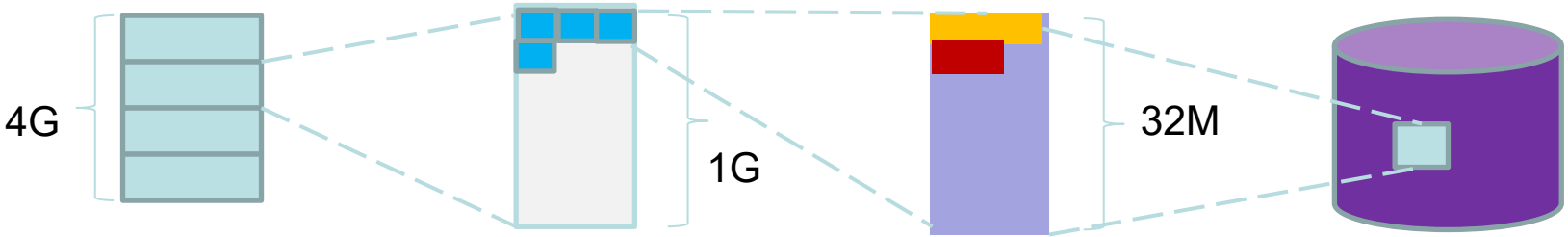
# ROBIN Cloud Native Platform



# ROBIN Scale Out Storage

- Redirect-On-Write design
- Consistent Volume groups for distributed apps
- Application aware
  - ✓ Snapshots, Clones, Backups, Encryption, Compression, QoS, Tiering
  - ✓ One API call for the operations
- ROBIN brings data management to
  - Helm applications
  - k8s stateful sets, deployment, replica sets

# Distributed Storage Layout




## Volume

Comprises of logical 1GB slices

## Slice

- ✓ Logical 1GB sub-part of volume
- ✓ Comprises of 32MB physical disk segments
- ✓ Segment allocation on write

(e.g above slice has only 4 segments)

## Segment

- ✓ Data written in log structured format
- ✓ Multiple data ranges within 1G logical slice can be stored in same segment

## Disk

- ✓ Divided into 32M segments

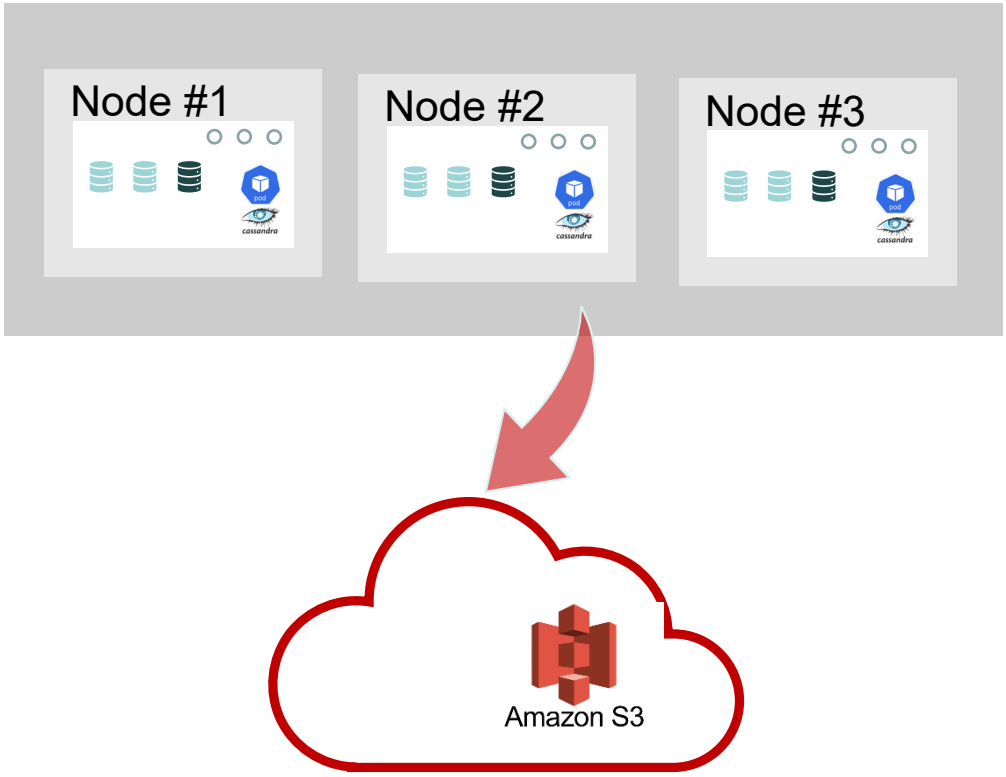
# Conclusion

- ✓ App/Storage replication for onsite copy
- ✓ Pod, Storage placement matters for data protection

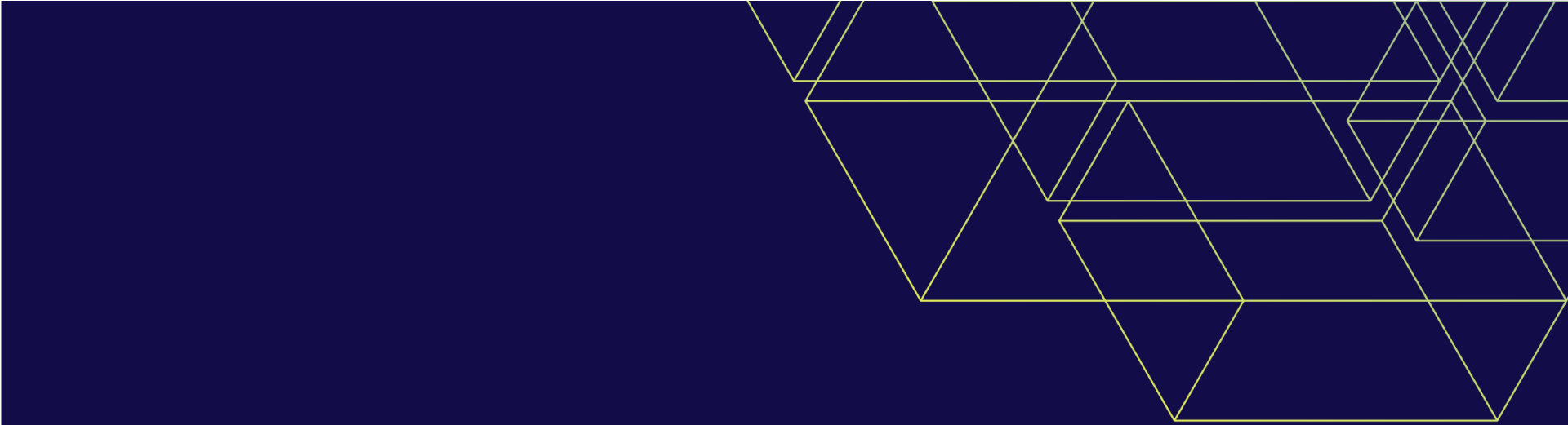
- ✓ Snapshots for quick recovery
- ✓ Multi-Volume apps need consistency volume groups

- ✓ Offsite/Cloud backups for third copy
- ✓ Backup at app level (has application metadata, config, volume data)

# Demo Goals




- Cassandra pod and storage copies are fault tolerant after allocation
- Snapshot entire Cassandra app and restore from it
- Backup the app to cloud aws s3 and restore app from the backup



**Please take a moment**
**to rate this session.**

**Your feedback matters to us.**