KubeCon CloudNativeCon North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

From Pre-Population to Disasters: Manage and protect the state of KubeVirt VMs

Michael Henriksen
Red Hat

Session Overview

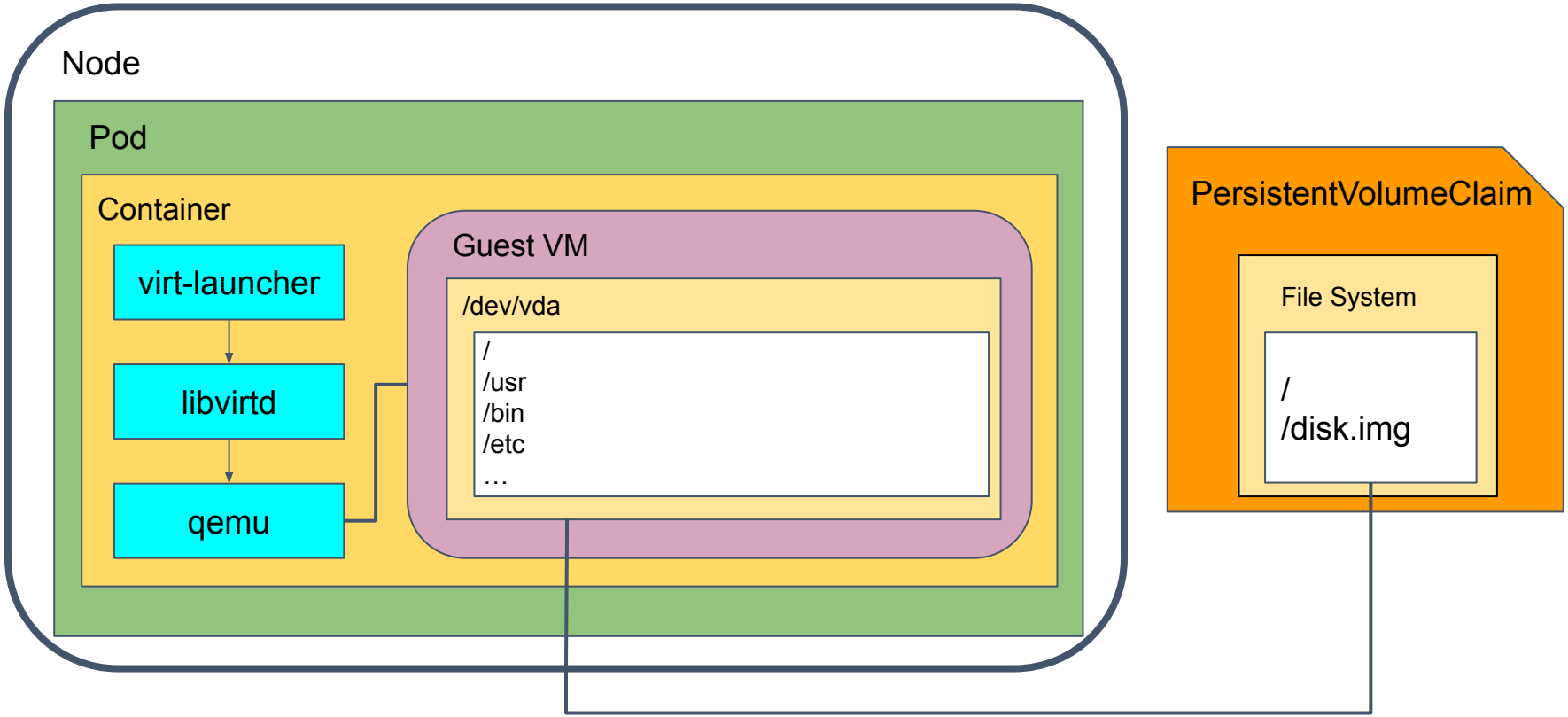
- KubeVirt Storage Architecture
- APIs, Flows, and Integrations
- Looking Forward

KubeVirt Storage Architecture

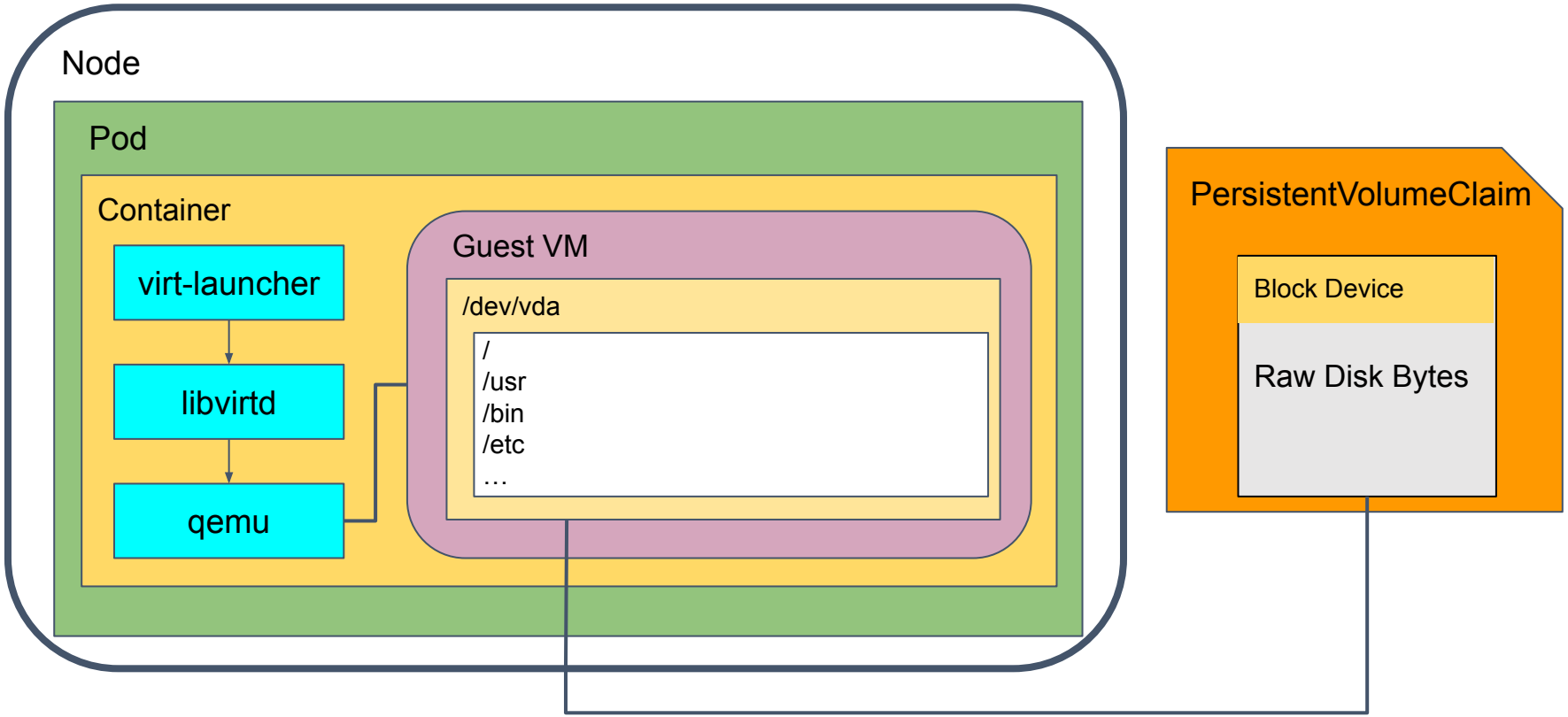
VMs in Containers on Kubernetes

- VMs
  - QEMU/KVM
  - Persistent disks
- Containers
  - NonRoot
  - No special capabilities
- Kubernetes
  - Pods
  - PersistentVolumeClaims

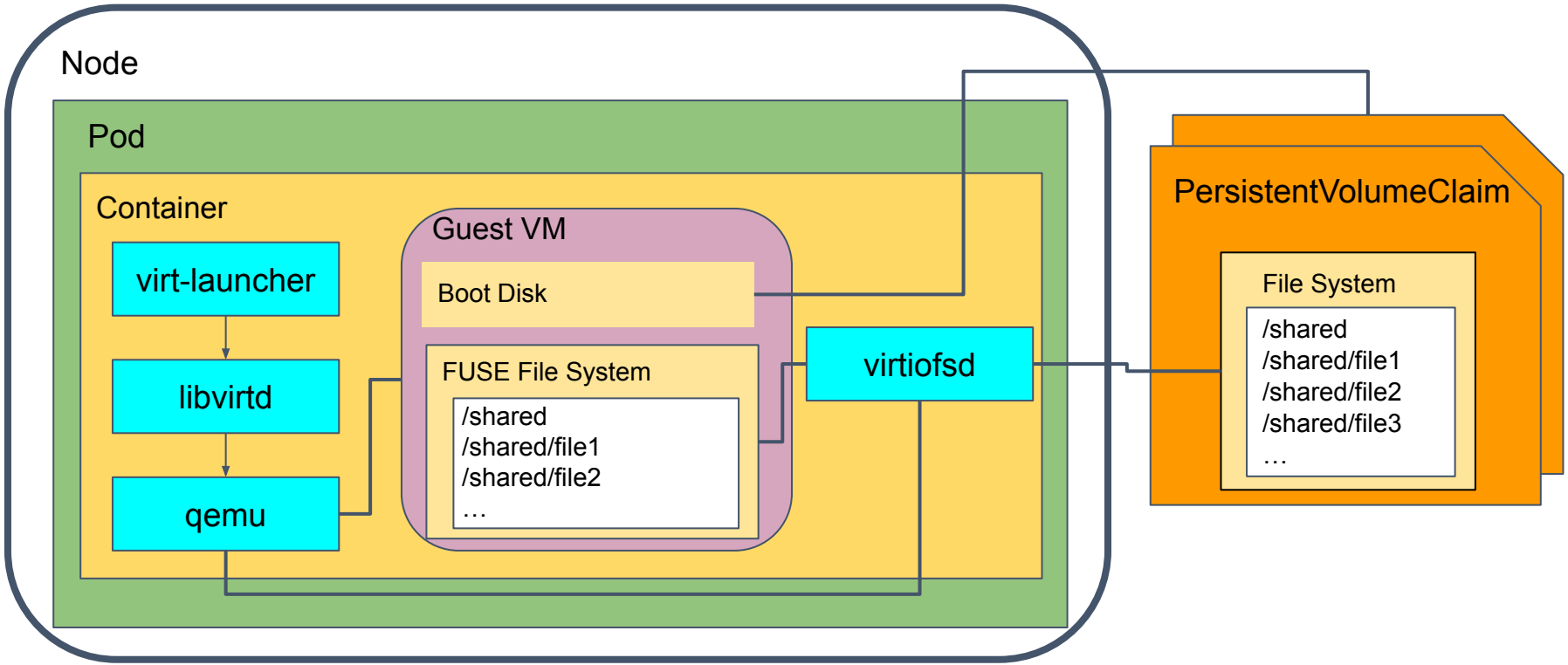
Persistent FileSystem Storage



Persistent Block Storage



Shared Persistent Storage with virtiofs



APIs, Flows and Integrations

Day 1
(Provisioning)

DataVolume API


```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: dv1
spec:
  source:
    http:
      url: "http://disk-server/cirros-qcow2.img"
  pvc:
    storageClassName: rook-ceph-block
    volumeMode: Block
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 10Gi
```

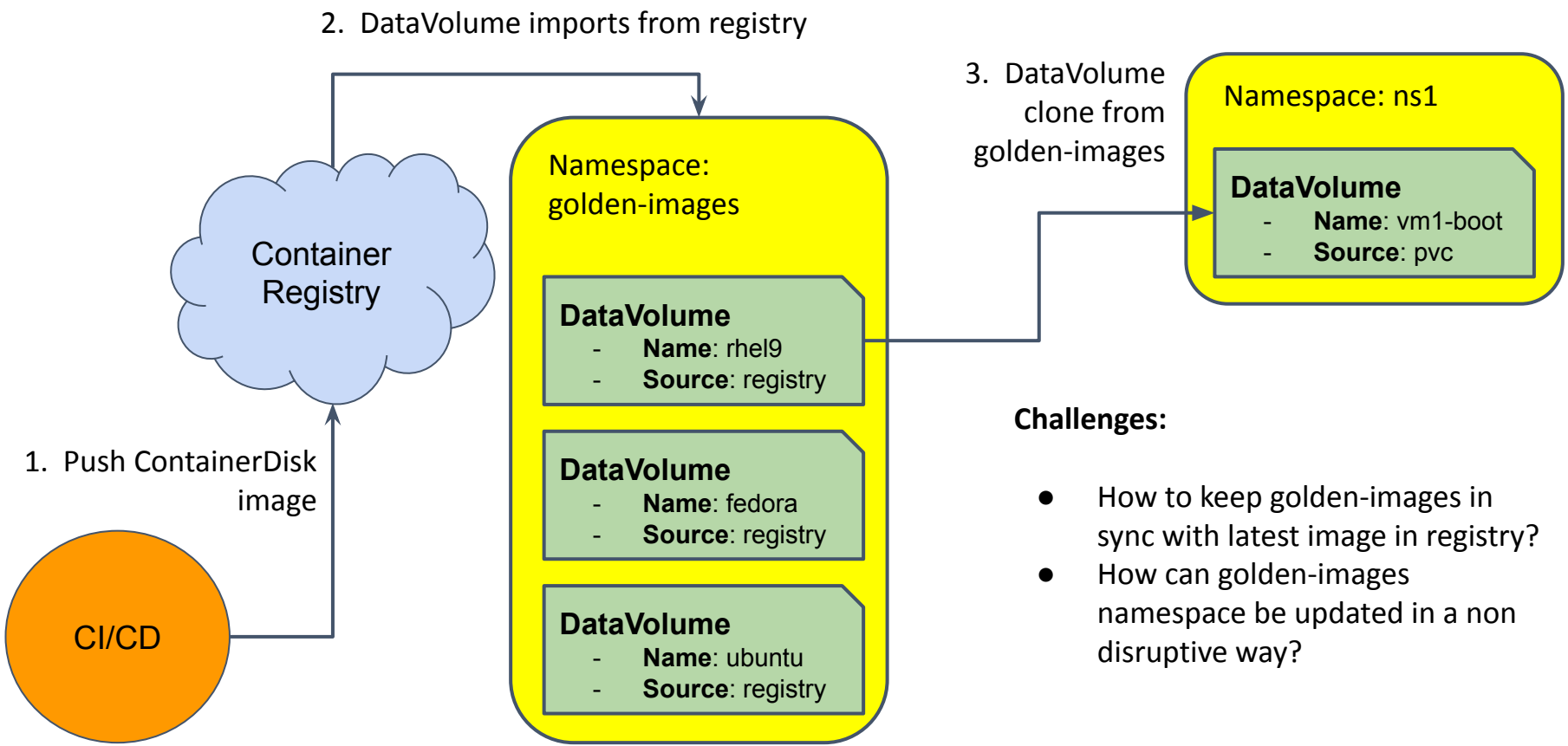
```yaml
source:
  registry:
    url: "docker://mhenriks/fedora-cd:latest"
    pullMethod: node # or pod
```

```yaml
source:
   pvc:
     namespace: golden-images
     name: rhel9
```

```yaml
source:
  upload: {}
```

```yaml
source:
  blank: {}
```

DIY Golden Image Provisioning



Challenges:

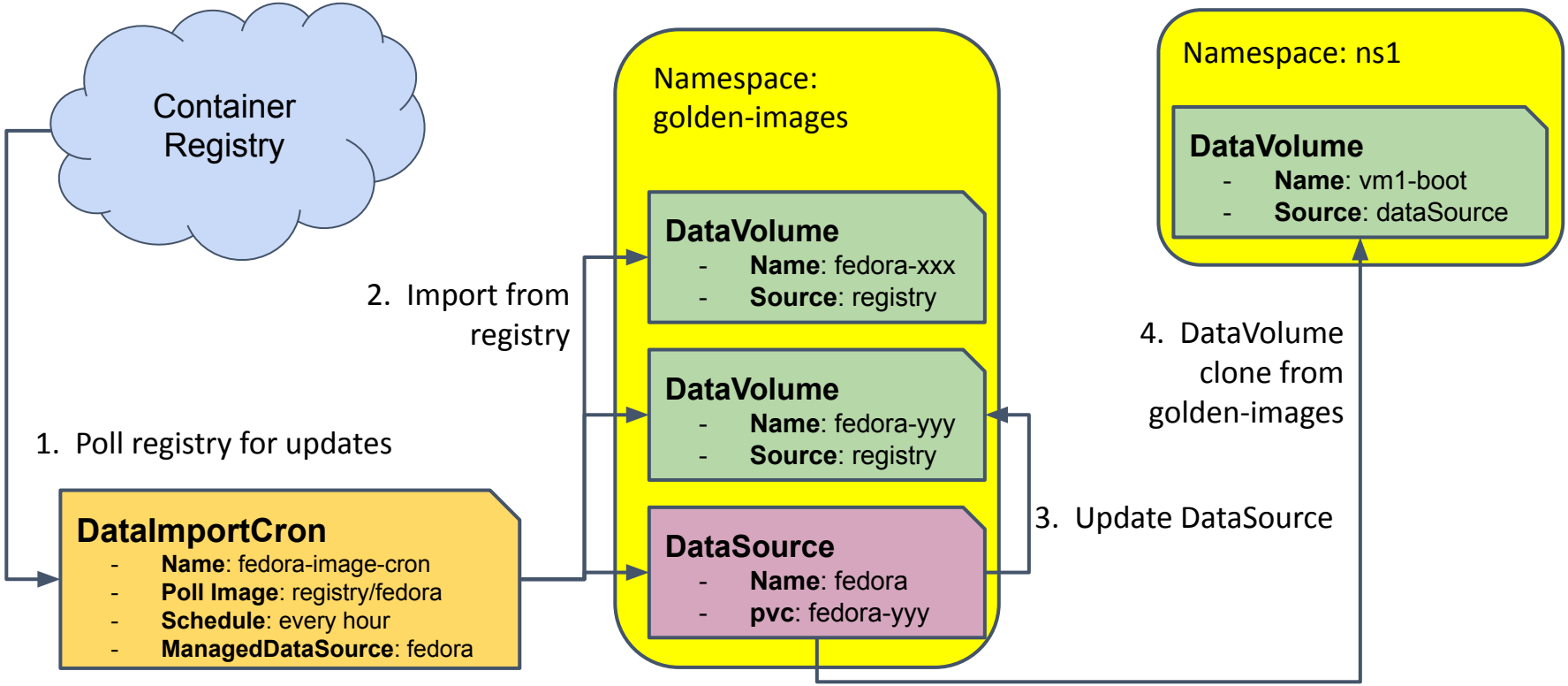
- How to keep golden-images in sync with latest image in registry?
- How can golden-images namespace be updated in a non disruptive way?

DataImportCron API


```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataImportCron
metadata:
  name: fedora-image-cron
  namespace: golden-images
spec:
  template:
    spec:
      source:
        registry:
          url:
"docker://mhenriks/fedora-cloud-registry-disk-demo:latest"
          pullMethod: node
      storage:
        resources:
          requests:
            storage: 5Gi
        storageClassName: rook-ceph-block
  schedule: "0 * * * *"
  garbageCollect: Outdated
  importsToKeep: 2
  managedDataSource: fedora
```

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: dv1
spec:
  sourceRef:
    kind: DataSource
    namespace: golden-images
    name: fedora
  pvc:
    storageClassName: rook-ceph-block
    volumeMode: Block
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 10Gi
```

DataImportCron Provisioning



Day 2
(Data Protection)

VM Snapshot/Restore

VirtualMachineSnapshot API


```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  name: vm1-snapshot
spec:
  source:
    apiGroup: kubevirt.io/v1alpha3
    kind: VirtualMachine
    name: vm1
```

VM Snapshot Process

1. Lock VM specification (no updates)
2. Invoke QEMU guest agent freeze API
   a. Execute user defined hook
   b. fsfreeze all mounted filesystems
3. Create VolumeSnapshot for each supported VM volume
4. Invoke QEMU guest agent thaw API
   a. fsfreeze –unfreeze
5. Capture the VM specification as well as any other resource definitions required for restore
6. Create VirtualMachineSnapshotContent resource containing embedded resource definitions and references to VolumeSnapshots
7. Unlock VM specification

VirtualMachineRestore API


```yaml
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineRestore
metadata:
  name: vm1-restore
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: vm1
  virtualMachineSnapshotName: vm1-snapshot
```

VM Restore Process

1. Ensure VM not running
2. Create new PVCs from VolumeSnapshots
3. Update VM spec
    a. Overwrite entire spec with snapshotted value
    b. Set volume references to refer to newly created PVCs
4. Delete previous PVCs/DataVolumeTemplates

Challenges:

- How to handle catastrophic failure?
- How to integrate with existing backup/Disaster Recovery solutions?

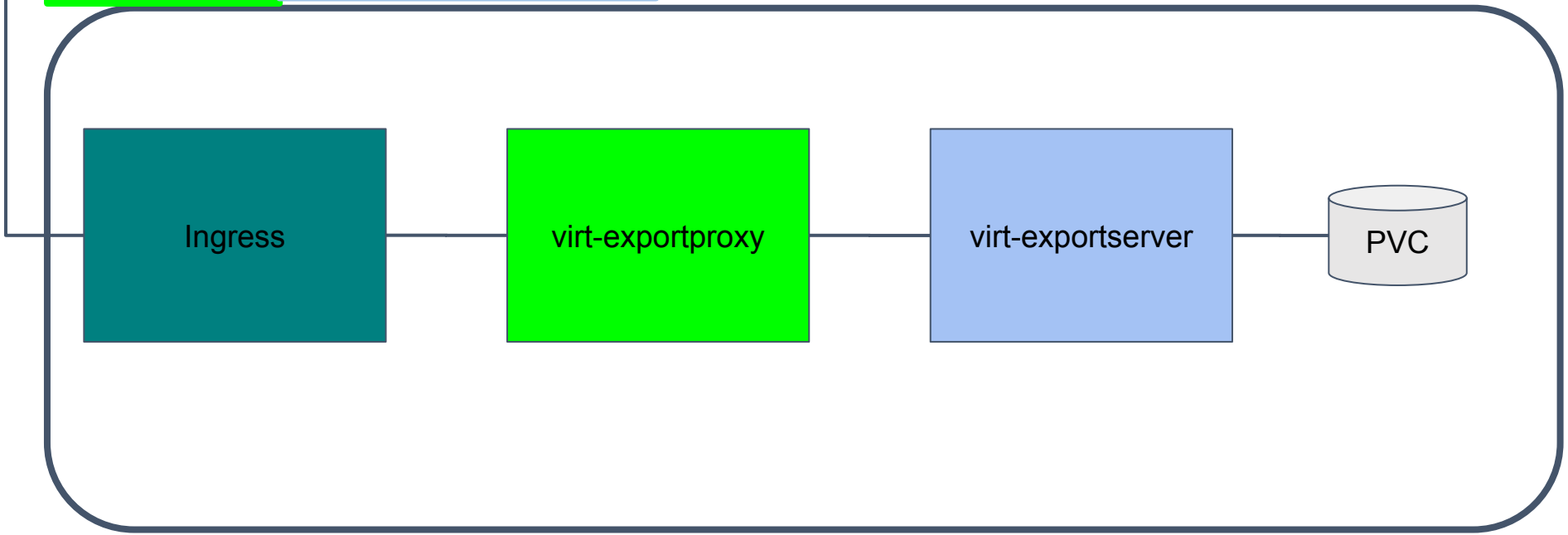
VirtualMachineExport API


```yaml
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: ex1
spec:
  source:
    apiGroup: snapshot.kubevirt.io
    kind: VirtualMachineSnapshot
    name: snap1
  tokenSecretRef: virt-export-token
---
apiVersion: v1
kind: Secret
metadata:
  name: virt-export-token
stringData:
  token: 5kvJ2j4KPl
```

```yaml
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
...
status:
  links:
    external:
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - name: testvm-fedora
        formats:
        - format: raw
          url: https://<redacted>/testvm-fedora/disk.img
        - format: gzip
          url: https://<redacted>/testvm-fedora/disk.img.gz
     internal:
     ...
```

VirtualMachineExport In Action

https://virt-exportproxy.mycluster/api/export.kubevirt.io/v1alpha1/namespaces/ns1/virtualmachineexports/export1/volumes/fedora/disk.img

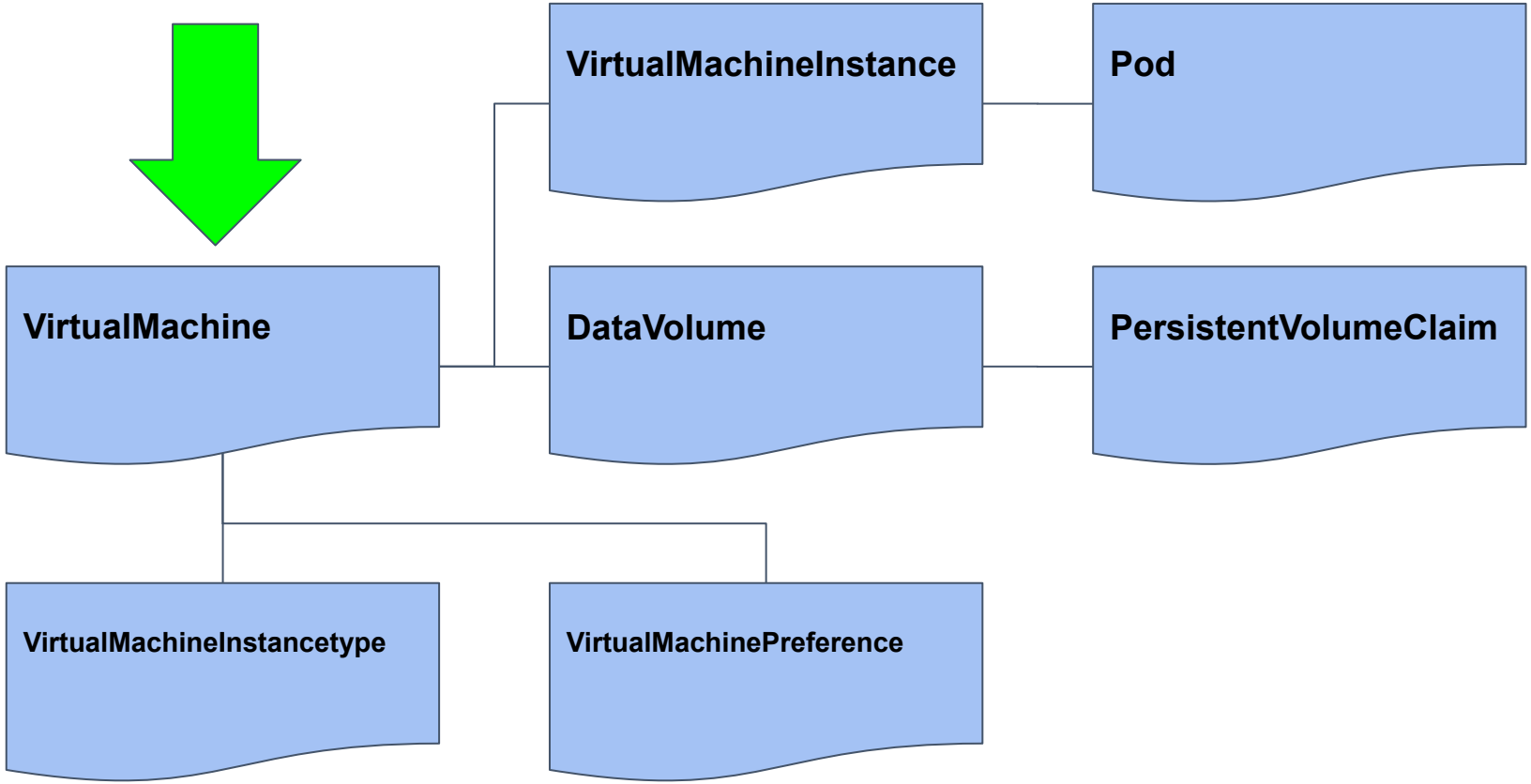


VirtualMachineExport Use Cases

- Disaster Recovery
  - Stream to Object Store
  - Stream to Registry (ContainerDisk)
- Migration
  - DataVolume Import on remote cluster
- Local Sharing
  - DataVolume Import across namespaces

Velero Plugin

Velero Plugin - Object Graph



Velero Plugin - Actions

- Backup Hooks
  - QEMU Guest Agent Freeze/Thaw
- Add Annotations
  - DataVolume
  - PersistentVolumeClaim
- Skip Backup/Restore
  - VirtualMachineInstance if owned by VirtualMachine
  - virt-launcher Pod

Looking Forward

Volume Populators API


```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: dv1
spec:
  source:
    http:
      url: "http://disk-server/fedora.img"
  pvc:
    storageClassName: rook-ceph-block
    volumeMode: Block
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 10Gi
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: populated-pvc
spec:
  dataSourceRef:
    apiGroup: populator.cdi.kubevirt.io
    kind: Import
    name: fedora-import
  storageClassName: rook-ceph-block
  volumeMode: Block
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

Volume Populators API


```yaml
apiVersion: populator.cdi.kubevirt.io/v1alpha1
kind: Import
metadata:
  name: fedora-import
spec:
  contentType: kubevirt
  http:
    url: "http://cdi-file-host.cdi/fedora.img"
```

Populate Process

1. External Provisioner sees PVC with `spec.dataSourceRef` set. Ignores it.
2. Populator controller sees the same PVC and creates PVC' in "hidden" namespace
3. PVC' populated with appropriate data
4. PVC' retention policy set to "Retain"
5. PVC' deleted
6. PV that was associated with PVC' bound PVC

cdi.kubevirt.io to v1

- DavaVolume Garbage Collection
  - Recently merged
  - Configurable in CDI `spec.config.dataVolumeTTLSeconds`
- v1alpha1 removal

Other Stuff

- KubeVirt and CDI aligning on releases
  - Three times a year, like Kubernetes
- Snapshot API to v1beta1
- DataVolume Clone from VolumeSnapshot source
- DataVolume Import auto size detection
- Add VM manifests to VMExport API

Call For Action!

Questions?