Summary

I am pleased to offer this Disaster Recovery Proposal for your review and approval. The solution detailed in this proposal has been created with your needs and I am confident you'll find that it offers great value to your Disaster Recovery Plan. Please review the proposed solution carefully to ensure it meets your expectations. If you are willing to move forward with the solution as it is described in this proposal, simply inform me your approval.

Proposed Solution

Disaster Recovery Plan is designed to backup and restore akaChain networks from potentially catastrophic.

Backing up is the only way to protect network’s data. And it’s also useful for routine administrative purposes such as migrating cluster resources to other cluster, or replicating cluster to development and testing clusters.

For an akaChain network, we need to backup two things:

* Configuration, State and Metadata of the cluster: resources are defined with a Kubernetes CRD *(Custom Resource Definition)* stored in *etcd*.
* Persistent volumes: includes Crypto materials, Channel artifacts, Peer backup files, Orderer backup files to recreate the network.

I propose use **Velero** tool to back up and restore our K8s cluster resources and persistent volumes.

# Backup

## The Velero provides 2 types for backing up:

* On-deman backups
* Scheduled backups

## Both types allow us to:

* Upload a tarball of copied K8s objects into *cloud object storage* – for example, AWS S3
* Make disk snapshots of persistent volumes then saving to *VolumeSnapshotLocation*.

## Backup workflow:

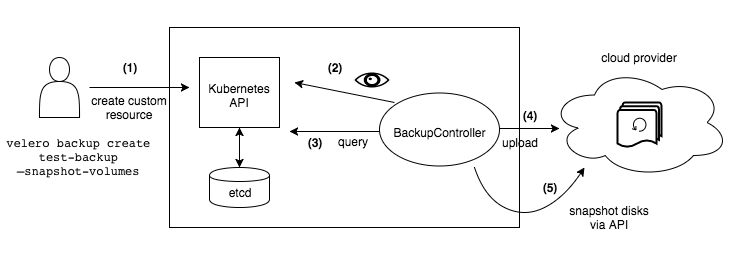


Figure 1. Back up Flow

1. The Velero client makes a call to the Kubernetes API server to create a Backup object.
2. The BackupController notices the new Backup object and performs validation.
3. The BackupController begins the backup process. It collects the data to back up by querying the API server for resources.
4. The BackupController makes a call to the object storage to upload the backup file.

## Backup files

### Data Objects in etcd

**etcd** is a consistent and highly-available key value store used as Kubernetes’ backing store for all cluster data.

#### Key types:

* Nodes (historically they were called Minion as we are reminded here)
* Namespaces
* ServiceAccounts
* Roles and RoleBindings, ClusterRoles / ClusterRoleBindings
* ConfigMaps
* Secrets
* Workloads: Deployments, DaemonSets, Pods, …
* Cluster’s certificates
* The resources within each apiVersion
* The events that bring the cluster in the current state

#### Get keys

ADVERTISE\_URL="https://134.209.178.162:2379"

kubectl exec etcd-node-01 -n kube-system -- sh -c \

"ETCDCTL\_API=3 etcdctl \

--endpoints $ADVERTISE\_URL \

--cacert /etc/kubernetes/pki/etcd/ca.crt \

--key /etc/kubernetes/pki/etcd/server.key \

--cert /etc/kubernetes/pki/etcd/server.crt \

get \"\" --prefix=true -w json" > etcd-kv.json

for k in $(cat etcd-kv.json | jq '.kvs[].key' | cut -d '"' -f2); do echo $k | base64 --decode; echo; done

/registry/apiregistration.k8s.io/apiservices/v1.

/registry/apiregistration.k8s.io/apiservices/v1.apps

/registry/apiregistration.k8s.io/apiservices/v1.authentication.k8s.io

/registry/apiregistration.k8s.io/apiservices/v1.authorization.k8s.io

/registry/apiregistration.k8s.io/apiservices/v1.autoscaling

/registry/apiregistration.k8s.io/apiservices/v1.batch

/registry/apiregistration.k8s.io/apiservices/v1.coordination.k8s.io

/registry/apiregistration.k8s.io/apiservices/v1.networking.k8s.io

/registry/apiregistration.k8s.io/apiservices/v1.rbac.authorization.k8s.io

/registry/apiregistration.k8s.io/apiservices/v1.scheduling.k8s.io

…

#### Values



### Persistent data

Backup persistent volumes to AWS EBS

A sample YAML BackupStorageLocation looks like the following:

apiVersion: velero.io/v1

kind: BackupStorageLocation

metadata:

name: default

namespace: velero

spec:

backupSyncPeriod: 2m0s

provider: aws

objectStorage:

bucket: myBucket

config:

region: us-west-2

profile: "default"

## Velero supported providers

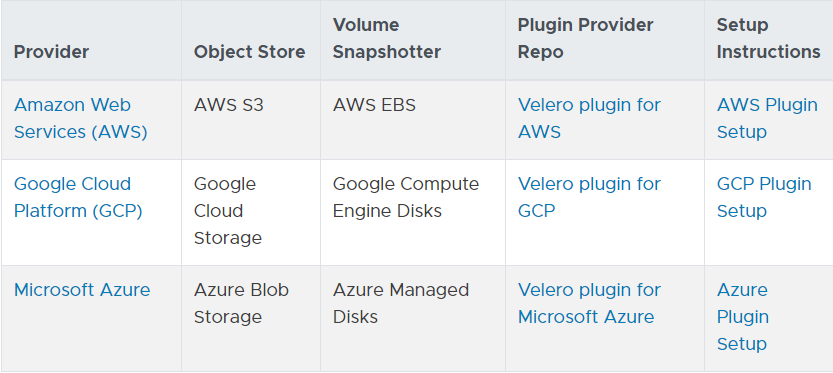


Figure 2. Storage Providers

I refer to use AWS S3 to upload the data objects that is stored in *etcd*, and AWS EBS to upload persistent data.

In case we want to use an on-premises cluster:

* We have to select an object storage backend that Velero can use to store backup data. If not, [MinIO](https://min.io/) is an open-source S3 compatible object storage system that can be installed on-premises.
* We must select a volume backup solution.   
  If we use [Portworx](https://portworx.com/) (or [another supported options](https://velero.io/docs/v1.3.0/supported-providers/)) for persistent storage, we can install Velero plugin to get native Portworx snapshots as part of backups.  
  Otherwise, we are able to use Velero’s [restic integration](https://velero.io/docs/v1.3.0/restic/), which provides a platform-agnostic file-level backup solution for volume data.

# Restores

The restore operation allows us to restore all of the objects and persistent volumes from a previously created backup. We can also restore only a filtered subset of objects and persistent volumes.

The default name of a restore is <BACKUP NAME>-<TIMESTAMP>, where <TIMESTAMP> is formatted as YYYYMMDDhhmmss. We can also specify a custom name. A restored object also includes a label with key velero.io/restore-name and value <RESTORE NAME>.

By default, backup storage locations are created in read-write mode. However, during a restore, we can configure a backup storage location to be in read-only mode, which disables backup creation and deletion for the storage location. This is useful to ensure that no backups are inadvertently created or deleted during a restore scenario.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

March 4, 2020

Name : Daniel Pham (Phạm Hồng Sơn)

Account : SonPH9