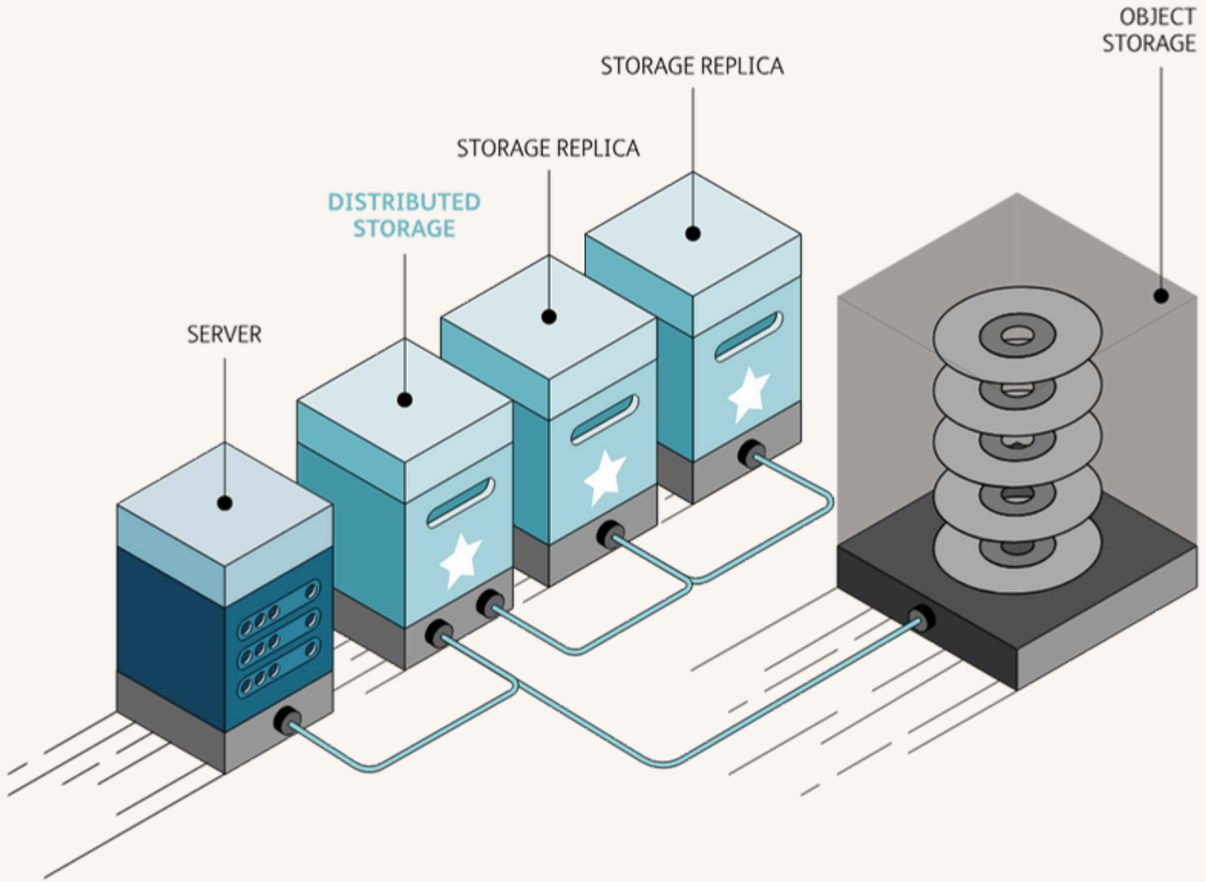
Managing Object Storage

The Kubernetes way using COSI

Karanjot Singh
Supervisors: Jack Henschel & Ankur Kothiwal



What we'll discuss today?

Introduction to object storage

CSI and K8s

Bucket Creation

User Access to Buckets

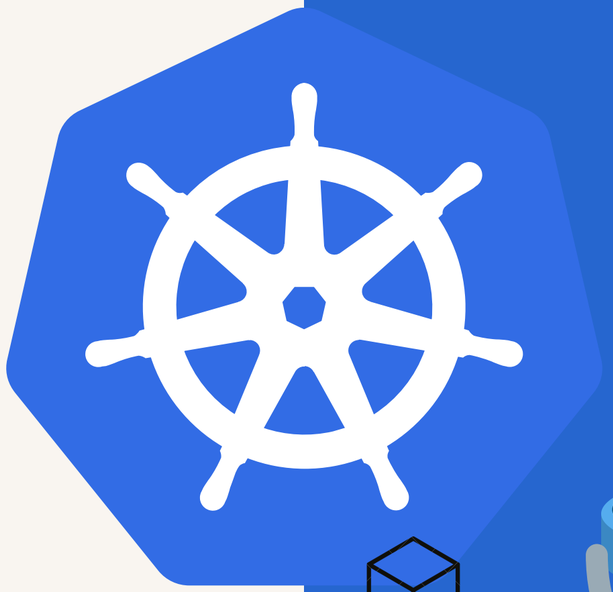
Problems with COSI

s3 Bucket Quotas

s3 Backups

Moving beyond file and block storage in Kubernetes

COSI

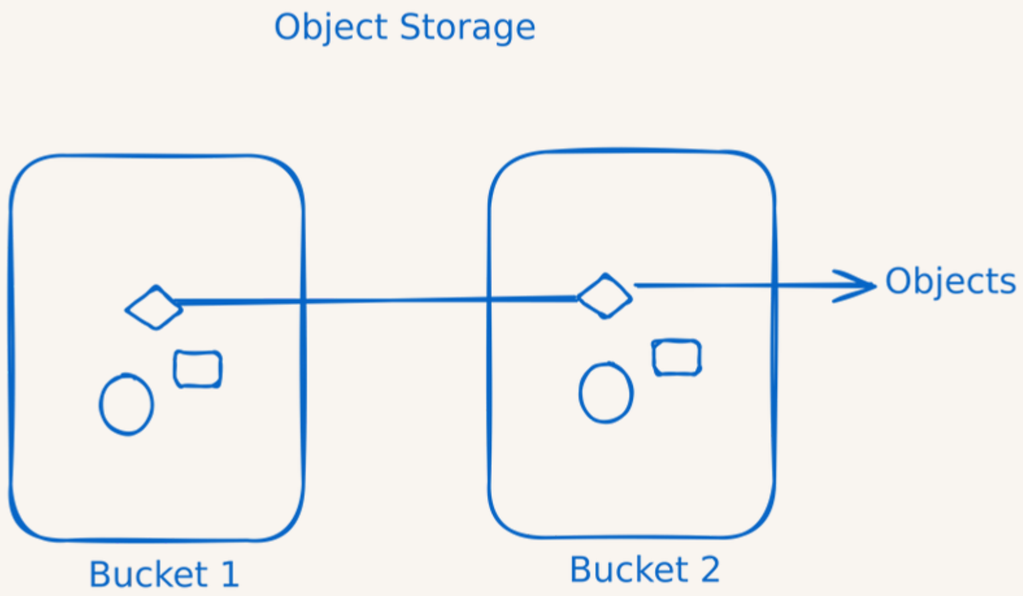


Introduction to Object storage

1.0

What is Object Storage?

- Data is broken into small discrete units known as objects and stored in a flat architecture
- It can be accessed by simple network APIs
- Organized into logical containers which store the objects, commonly known as buckets
- It is cost efficient and can scale into extremely large quantities while maintaining quick access



CSI and K8s

- Container Storage Interface provides a platform to expose block and file storage systems.

- Prior to CSI, connecting to new volumes plugins needed to be directly a part of core Kubernetes. CSI allowed vendors to move this logic into separate drivers. Some popular CSI drivers expose Amazon EBS, Ceph, or Google Cloud Store.

- This meant more options for storage, and it made core Kubernetes more secure and reliable.

- API-driven provisioning: It has StorageClass (SC), PersistentVol (PV), PersistentVolClaim (PVC)

Need for COSI

- Provide a generic, dynamic provisioning API to consume object store
- App Pods can access the bucket in the underlying object-store like a PVC
- Be vendor agnostic (S3, RGW, Swift, GCS , etc..)
- Object storage can’t use the block and file storage primitives used in CSI. The unit of provisioning in object storage is a bucket (not a volume) and buckets are not mounted. Rather, they are accessed over the network. Moreover, object storage allows for more granular access control. - Requires a new standard for managing object storage

Container Object Storage Interface (COSI)

COSI Architecture

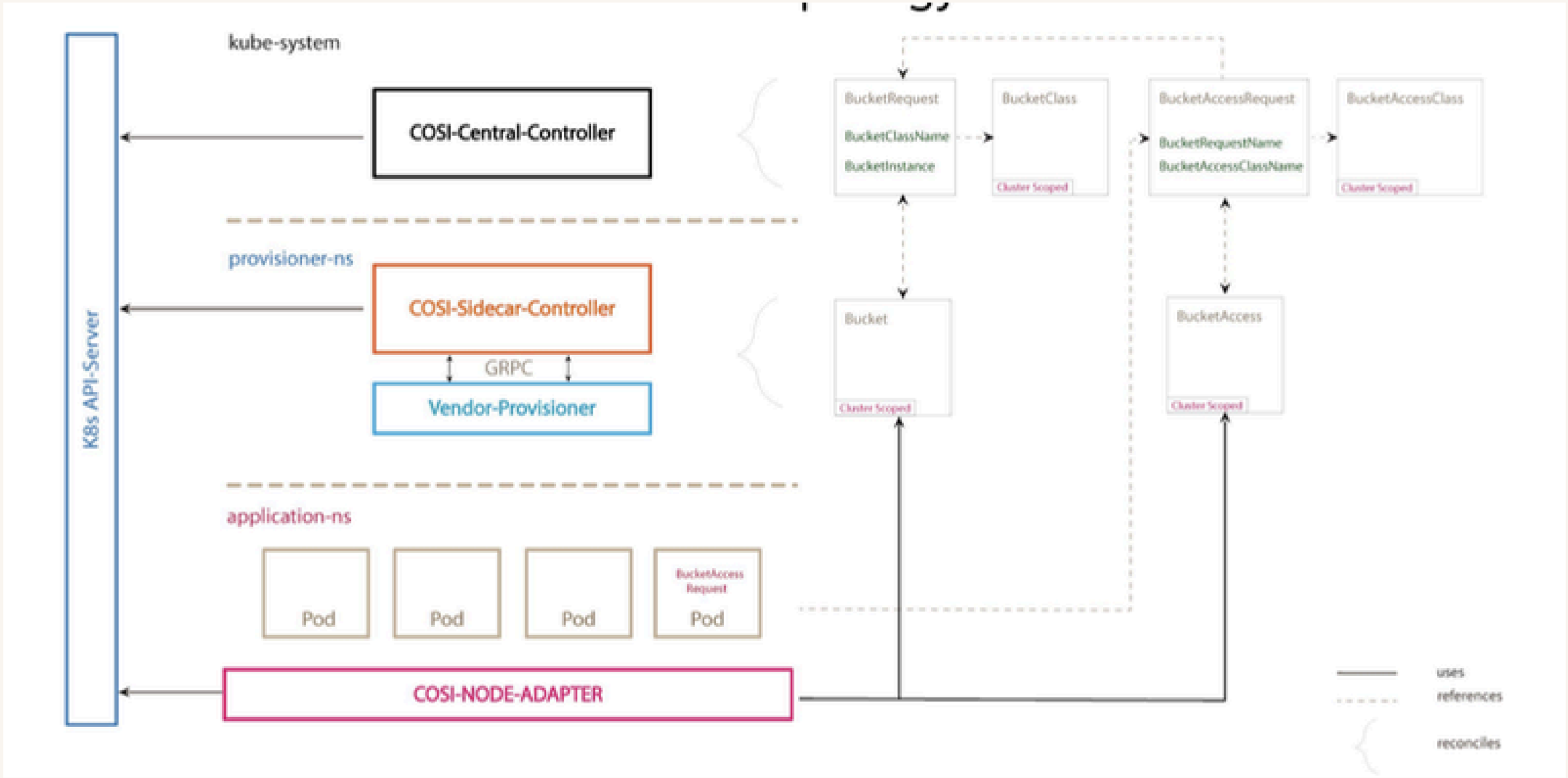
COSI Controller Manager: acts as the main controller that processes changes to COSI API objects. A central controller that validates, authorizes and binds COSI created buckets to BucketClaims.

COSI SideCar: acts as a translator between COSI API requests and vendor-specific COSI Drivers. All operations that require communication with the OSP is triggered by the Sidecar using gRPC calls to the driver

COSI Driver: vendor specific component that receives requests from the sidecar and calls the appropriate vendor APIs to create buckets, manage their lifecycle and manage access to them.

COSI Terminologies

- **Bucket:** Resource to represent a Bucket in OSP. Buckets are cluster-scoped.
- **BucketClaim:** A claim to create a Bucket. BucketClaim is namespace-scoped
- **BucketClass:** Resouce for configuring common properties for multiple Buckets. BucketClass is cluster-scoped.
- **BucketAccessClass:** Resource for configuring common properties for multiple BucketClaims. BucketAccessClass is a clustered resource

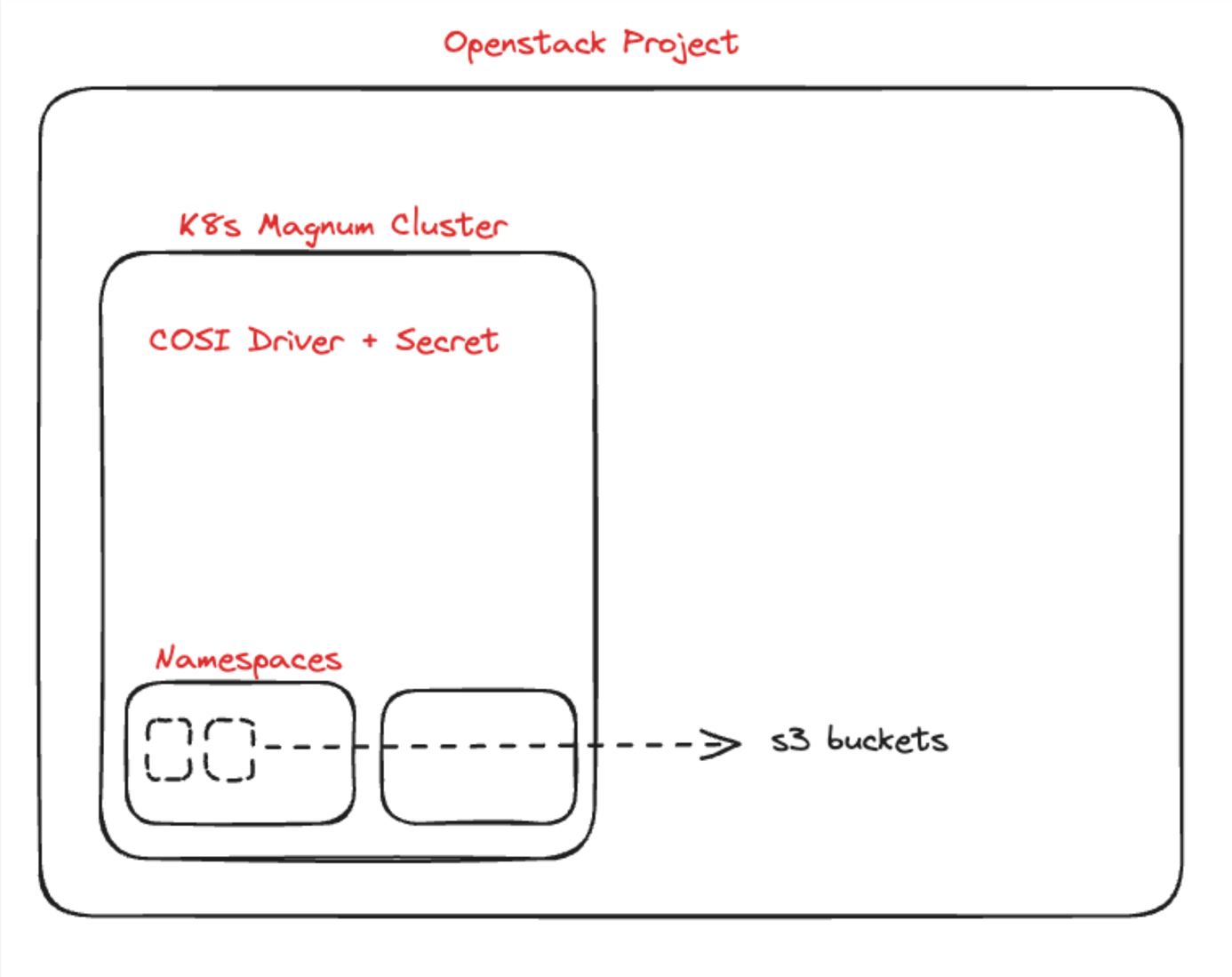

kube-system
COSI-Central-Controller
provisioner-ns
COSI-Sidecar-Controller
GRPC
Vendor-Provisioner
application-ns
Pod
Pod
Pod
BucketAccess Request
Pod
COSI-NODE-ADAPTER
K8s API-Server
BucketRequest
BucketClassName
BucketInstance
BucketClass
Cluster Scoped
BucketAccessRequest
BucketRequestName
BucketAccessClassName
BucketAccessClass
Cluster Scoped
Bucket
Cluster Scoped
BucketAccess
Cluster Scoped
uses
references
reconciles

tldr;

- BucketClaims/Bucket are similar to PVC/PV.
- **BucketClaim** is used to request generation of new buckets.
- **Buckets** represent the actual Bucket.
- **BucketClass** is similar to StorageClass. It is meant for admins to define and control policies for Bucket Creation
- **BucketAccess** is required before a bucket can be “attached” to a pod.
- BucketAccess both represents the attachment status and holds a pointer to the access credentials secret.
- **BucketAccessClass** is meant for admins to control authz/authn for users requesting access to buckets.
- The two APIs, namely, BucketAccess and BucketAccessClass are used to denote access credentials and policies for authentication since object storage is always authenticated, and over the network, access credentials are required to access buckets

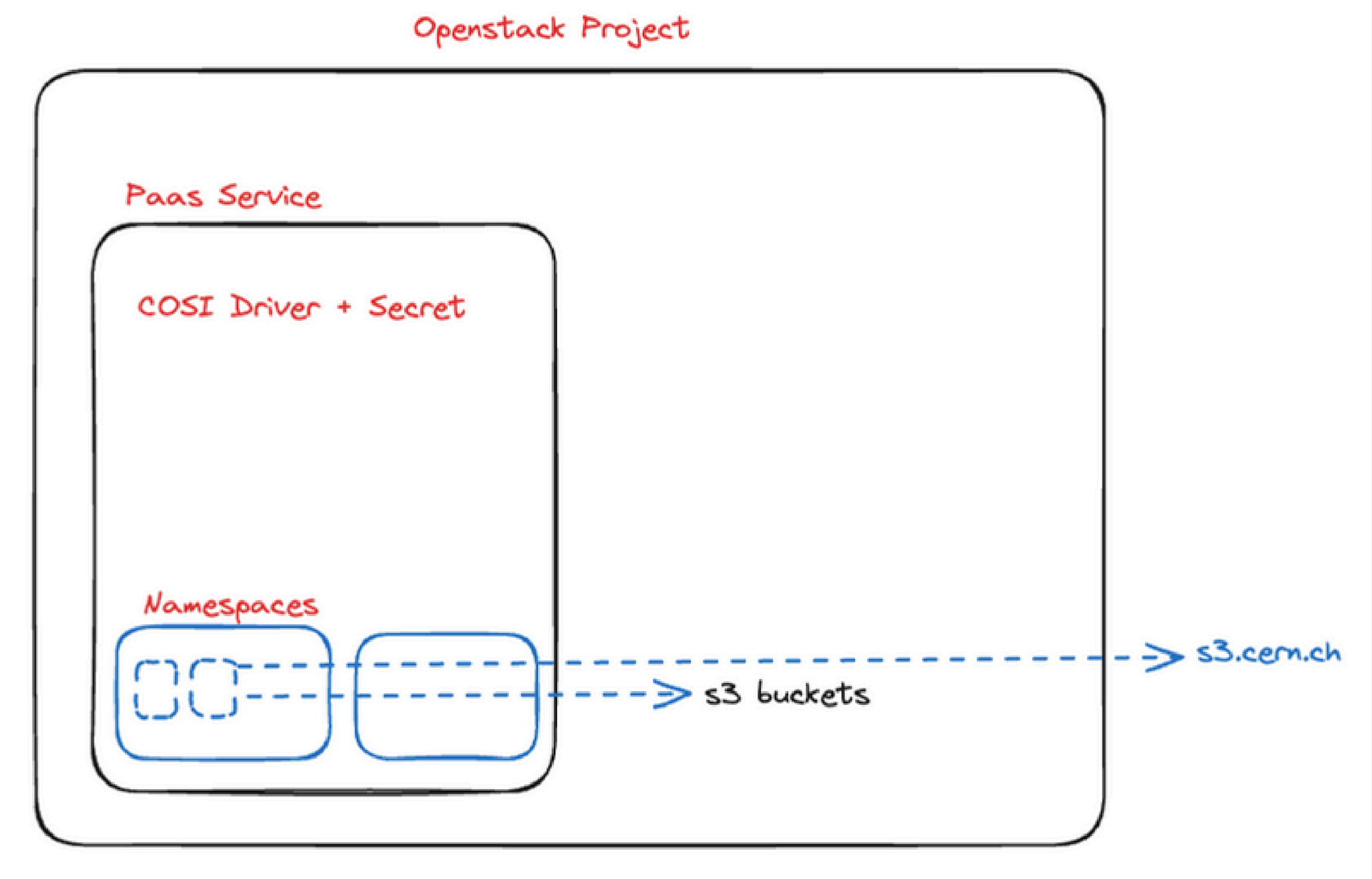
Two Deployment Models:

Deployment Model 1 : K8s Magnum



Two Deployment Models:

Deployment Model 2 : PaaS



Bucket Creation using COSI

Bucket Creation

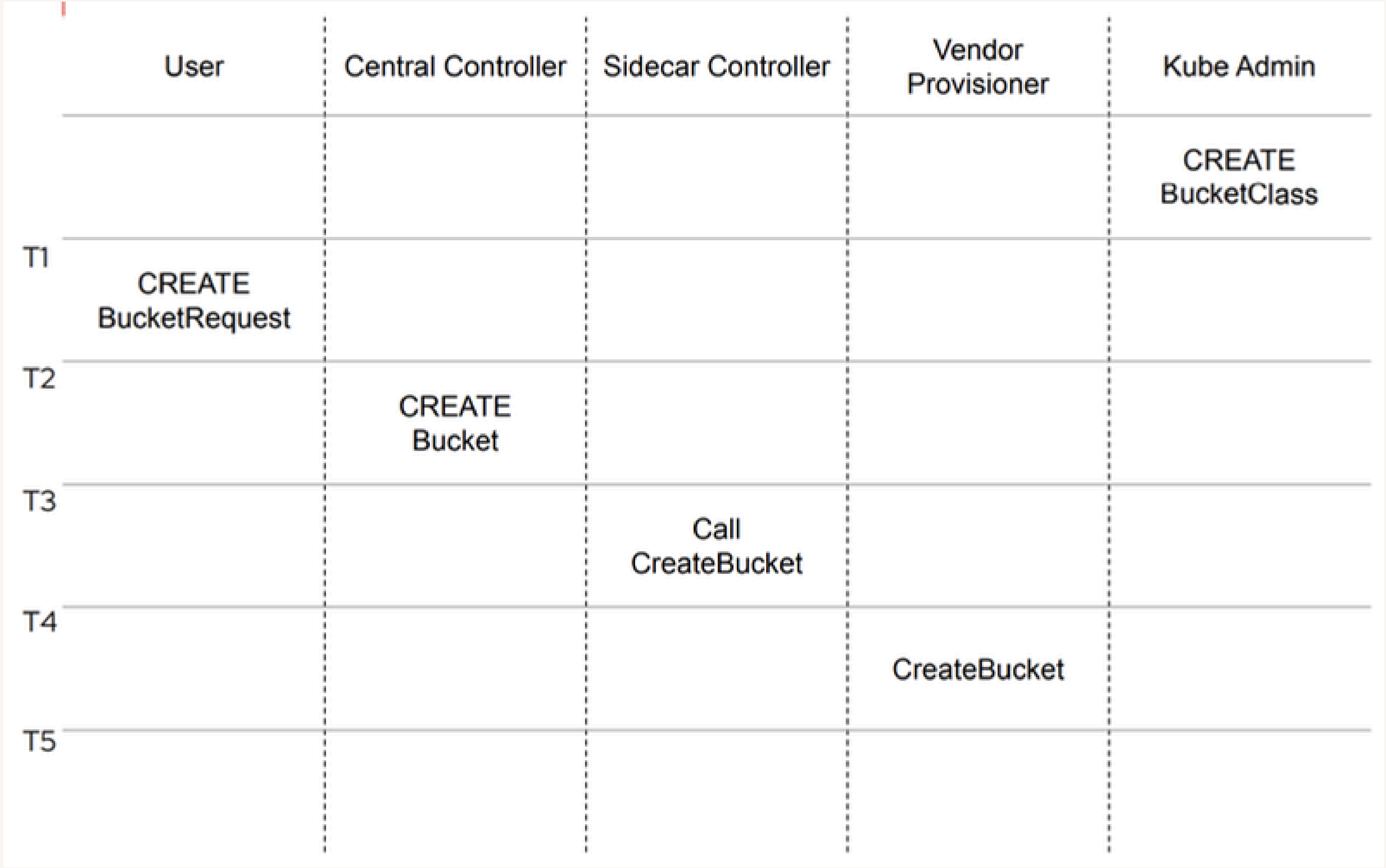
User story: Wants to Create an S3 Bucket and has access to the bucket.

Admins - creates `BucketClass` represents a set of common properties shared by multiple buckets. It is used to specify the driver for creating Buckets, and also for configuring driver-specific parameters

```
kind: BucketClass
apiVersion: objectstorage.k8s.io/v1alpha1
metadata:
  name: sample-bcc
driverName: cosi.ceph.objectstorage.k8s.io
deletionPolicy: Delete
parameters:
  objectStoreUserSecretName: ceph-object-user-my-store-cosi
  objectStoreUserSecretNamespace: ceph-cosi-driver
```

Users - request buckets to be created for their workload by creating an `ObjectBucketClaim`

```
kind: BucketClaim
apiVersion: objectstorage.k8s.io/v1alpha1
metadata:
  name: sample-bucket3
  namespace: notea
spec:
  bucketClassName: sample-bcc
  protocols:
  - s3
```


User
Central Controller
Sidecar Controller
Vendor Provisioner
Kube Admin
CREATE BucketClass
T1
CREATE BucketRequest
T2
CREATE Bucket
T3
Call CreateBucket
T4
CreateBucket
T5

```
[karanjot@lxplus968 examples]$ kubectl get pods -n notea
NAME                              READY   STATUS    RESTARTS   AGE
notea-deployment-5bc9f99669-wzx9m   1/1     Running   0          5d20h
[karanjot@lxplus968 examples]$ kubectl get pods -n ceph-cosi-driver
NAME                                    READY   STATUS    RESTARTS   AGE
objectstorage-provisioner-867fd4b79-m5cwp   2/2     Running   0          6d1h
[karanjot@lxplus968 examples]$ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
objectstorage-controller-bc6586f8d-67r4g   1/1     Running   0          15d
[karanjot@lxplus968 examples]$ vim bucketclass.yaml
[karanjot@lxplus968 examples]$ vim bucketclaim.yaml
```

Generating Access Credentials for Buckets using COSI

Generating User Bucket Creds

User story: Wants to Create Creds for Access to each Bucket

Admins - creates BucketAccessClass which represents a set of common properties shared by multiple BucketAccesses. It is used to specify policies for creating access credentials, and also for configuring driver-specific access parameters.

```
kind: BucketAccessClass
apiVersion: objectstorage.k8s.io/v1alpha1
metadata:
  name: sample-bac
driverName: cosi.ceph.objectstorage.k8s.io
authenticationType: KEY
parameters:
  objectStoreUserSecretName: ceph-object-user-my-store-cosi
  objectStoreUserSecretNamespace: ceph-cosi-driver
```

Users - request access to buckets by creating an `BucketAccess`

```
kind: BucketAccess
apiVersion: objectstorage.k8s.io/v1alpha1
metadata:
  name: sample-access3
  namespace: notea
spec:
  bucketClaimName: sample-bucket3
  bucketAccessClassName: sample-bac
  credentialsSecretName: sample-access-secret3
  protocol: s3
```

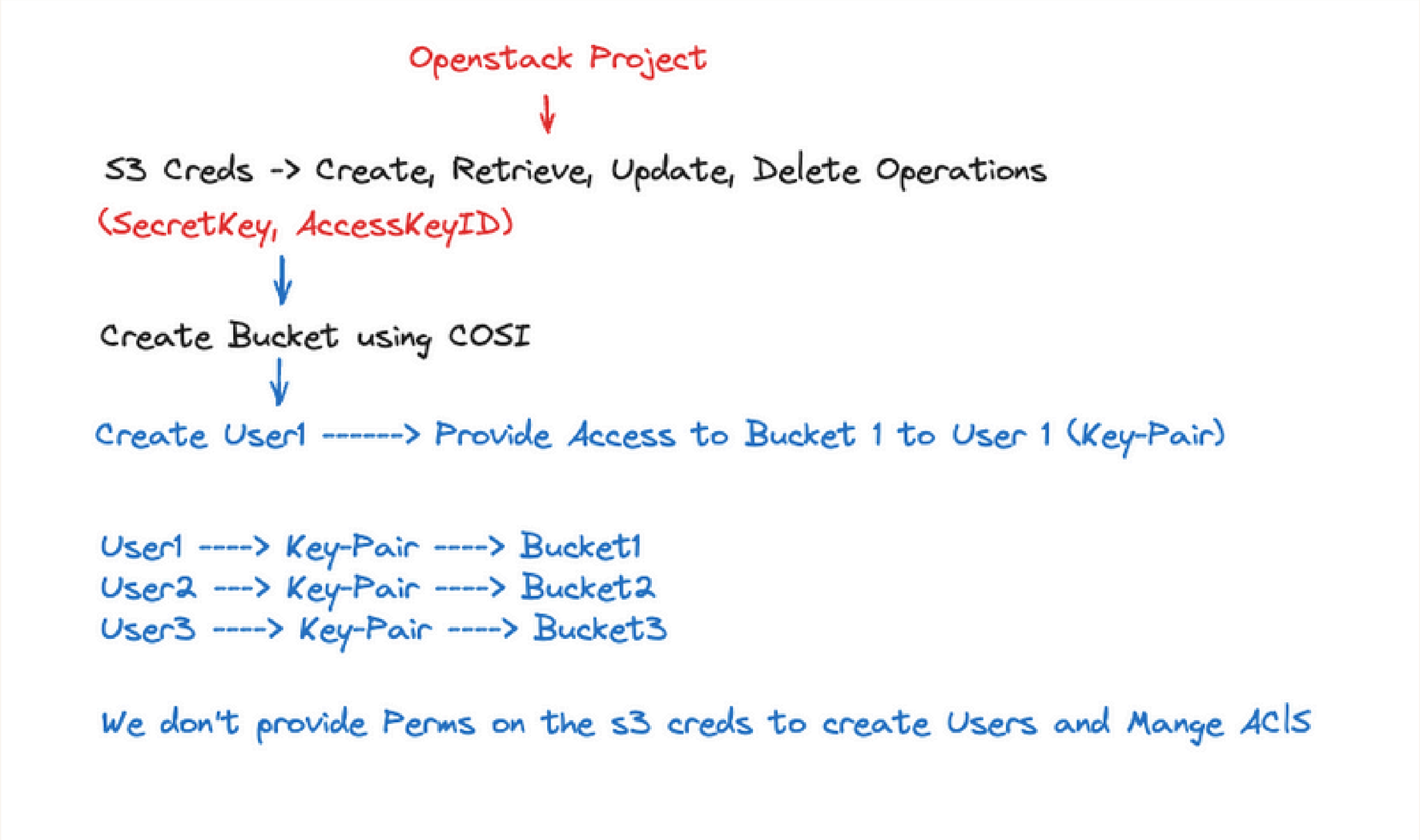
- Secret Created

```
karanjot@lxplus932.cern.ch ~/ceph-cosi/examples git:(master)±8 (0.419s)
kubectl get secret -n notea
NAME                                TYPE                 DATA   AGE
ceph-object-user-my-store-cosi      Opaque               2      13d
sample-access-secret                Opaque               2      13d
sample-access-secret1               Opaque               1      10d
sample-access-secret2               Opaque               1      9d
sample-access-secret3               Opaque               1      22h
```

```
[karanjot@lxplus951 examples]$ cat bucketclaim.yaml
kind: BucketClaim
apiVersion: objectstorage.k8s.io/v1alpha1
metadata:
  name: sample-bucket-test-demo
  namespace: notea
spec:
  bucketClassName: sample-bcc
  protocols:
  - s3
[karanjot@lxplus951 examples]$ vim bucketaccess.yaml
[karanjot@lxplus951 examples]$ kubectl create -f bucketaccess.yaml
bucketaccess.objectstorage.k8s.io/sample-access-secret-test-demo created
[karanjot@lxplus951 examples]$ kubectl get secret -n notea
```

Generating User Bucket Creds

Problem: Project-Wide s3 creds doesn't have access to creating User-Bucket Key-Pairs



Generating User Bucket Creds

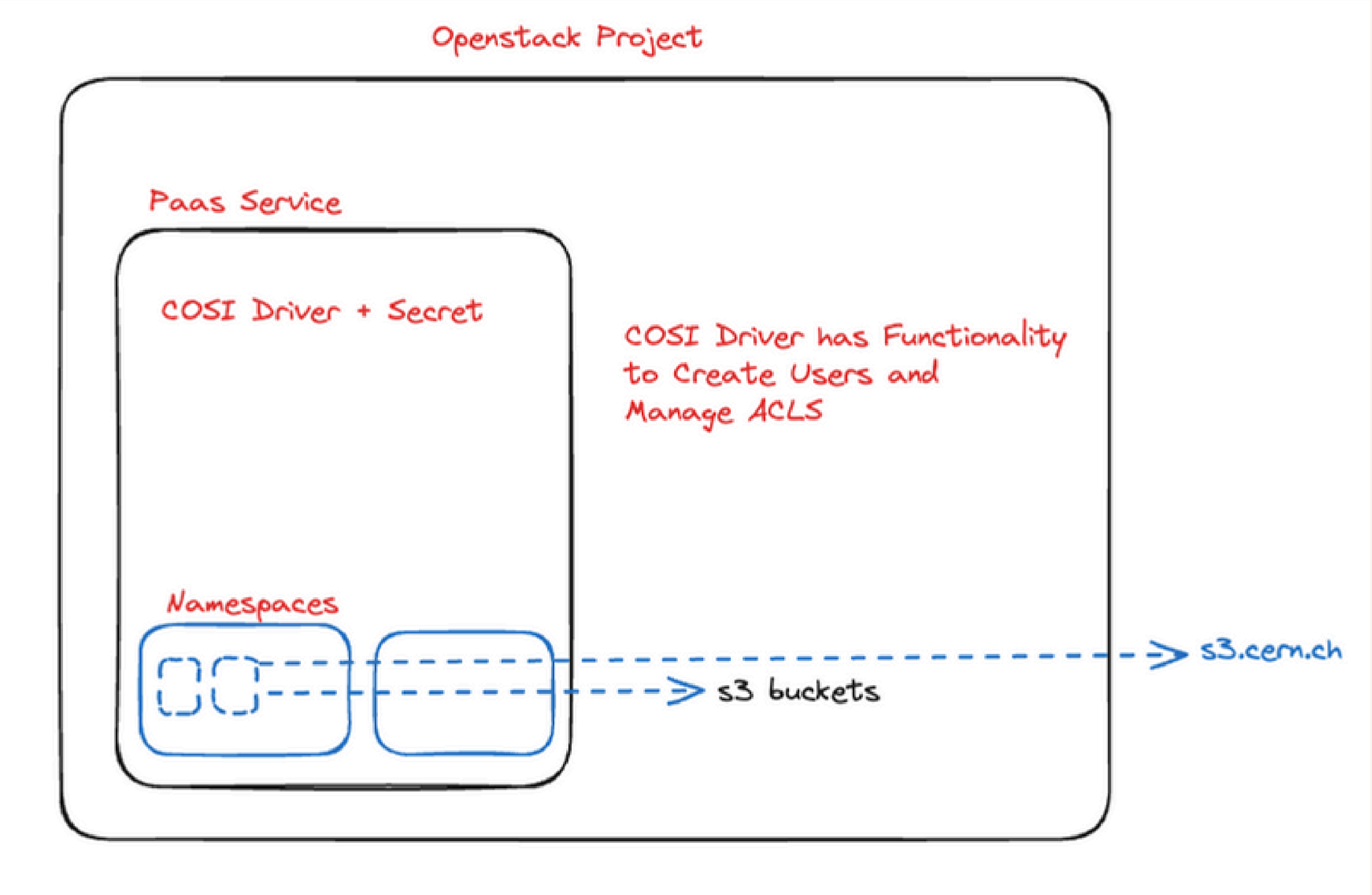
Workaround for K8s Magnum - Directly Pass Secret Key for every Bucket

```
148  147  func (s *provisionerServer) DriverGrantBucketAccess(ctx context.Context,
149  148          req *cosispec.DriverGrantBucketAccessRequest) (*cosispec.DriverGrantBucketAccessResponse, error) {
150       -      // TODO : validate below details, Authenticationtype, Parameters
151  149          userName := req.GetName()
152  150          bucketName := req.GetBucketId()
153  151          klog.V(5).Infof("req %v", req)
154       -      klog.Info("Granting user accessPolicy to bucket ", "userName", userName, "bucketName", bucketName)
     152  +      klog.Info("Granting user access to bucket ", "userName", userName, "bucketName", bucketName)
155  153          parameters := req.GetParameters()
156  154
157       -      s3Client, rgwAdminClient, err := initializeClients(ctx, s.Clientset, parameters)
     155  +      secretKey, err := fetchSecretKey(ctx, s.Clientset, parameters)
158  156          if err != nil {
159       -              klog.ErrorS(err, "failed to initialize clients")
160       -              return nil, status.Error(codes.Internal, "failed to initialize clients")
161       -      }
162       -
163       -      user, err := rgwAdminClient.CreateUser(ctx, rgwadmin.User{
164       -              ID:          userName,
165       -              DisplayName: userName,
166       -      })
167       -
168       -      // TODO : Do we need fail for UserErrorExists, or same account can have multiple BAR
169       -      if err != nil && !errors.Is(err, rgwadmin.ErrUserExists) {
170       -              klog.ErrorS(err, "failed to create user")
171       -              return nil, status.Error(codes.Internal, "User creation failed")
172       -      }
173       -
174       -      policy, err := s3Client.GetBucketPolicy(bucketName)
175       -      if err != nil {
176       -              if aerr, ok := err.(awserr.Error); ok && aerr.Code() != "NoSuchBucketPolicy" {
177       -                      return nil, status.Error(codes.Internal, "fetching policy failed")
178       -              }
     157  +              klog.ErrorS(err, "failed to fetch secret key")
     158  +              return nil, status.Error(codes.Internal, "failed to fetch secret key")
179  159          }
```

Full Snippet: https://gitlab.cern.ch/karanjot/ceph-cosi/-/commit/d31be044faf09a290562ee4872b1e1accdb57e02

Generating User Bucket Creds

No Workaround Needed for PaaS Model

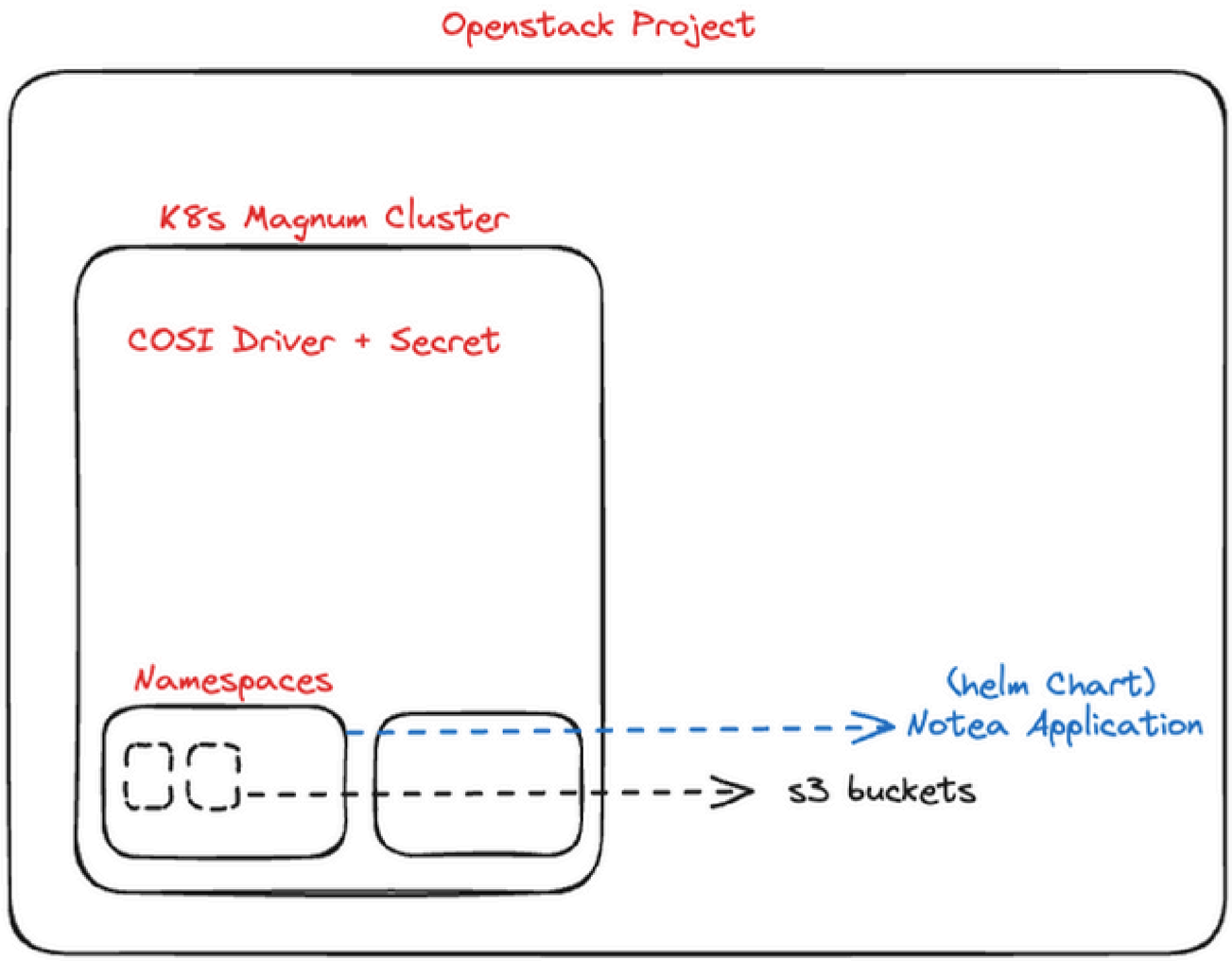


```
s3cmd --config S3CONFIG.cfg ls
2024-07-02 14:32  s3://sample-bcc6a41cbd4-aa19-401d-bcd3-b044b87f1cc0
2024-07-16 12:35  s3://sample-bcc962cb321-3e17-4e02-9b5d-bd8e2315d882
2024-07-01 09:53  s3://sample-bcca8fc655b-e4d6-4bb2-a5d6-2527470153c7
2024-07-10 16:14  s3://sample-bccd3233a6f-1639-48c2-91fb-caa31054c2ab
2024-06-20 13:45  s3://test-jack
2024-06-21 09:02  s3://test-karan
```

Problem with the Bucket Names

COSI generates random long bucket names to prevent hijacking of buckets

https://github.com/kubernetes-sigs/container-object-storage-interface-spec/issues/45



- Using `Notea` - a note-taking application, for sample deployment
- Namespace - Notea
- Deployed using Helm Chart

```yaml
application:
  # Required parameters:
  # Password to login to the app
  password: "<your-password>"
  # AccessKey
  store_access_key: "<your-s3-access-key>"
  # SecretKey
  store_secret_key: "<your-s3-secret-key>"
  # Bucket
  store_bucket: "<your-s3-bucket>"
  # Optional parameters:
  # Host name or an IP address.
  store_end_point: "<your-s3-end-point>"
```

- Longer Process - Getting bucket name and then including it in our values.yaml for helm chart
- Not a Good Practice to store secrets like this

Problems with BucketNames

Workaround

- Expose the secret as volume and use an entry point script to export the secrets/bucket-name as environment variables.

```
  volumeMounts:
  - name: cosi-secrets
    mountPath: "/var/run/secrets"
    readOnly: true
```

```
#!/bin/sh

S3_CONFIG_FILE=${S3_CONFIG_FILE:-"${SECRET_MOUNT_PATH}/BucketInfo"}

STORE_BUCKET=$(cat "$S3_CONFIG_FILE" | jq -r .spec.bucketName)
STORE_ACCESS_KEY=$(cat "$S3_CONFIG_FILE" | jq -r .spec.secretS3.accessKeyID)
STORE_SECRET_KEY=$(cat "$S3_CONFIG_FILE" | jq -r .spec.secretS3.accessSecretKey)

export STORE_BUCKET
export STORE_ACCESS_KEY
export STORE_SECRET_KEY

exec "$@"
```

Future Work: Bucket Quotas, Backups etc.

Bucket Quotas & Backups

Quota Management through COSI: Size quotas are left to drivers to implement b/c there is not uniform support between vendors – esp. Cloud providers don't have size quotas

Mutability in Quota: This will require mutability in COSI APIs tracked here currently: https://github.com/orgs/kubernetes-sigs/projects/63/views/1?pane=issue&itemId=58104711

Replication and Backups: Not currently in spec but planned for future. Replication is a higher level feature, probably good to wait to discuss until v1 or later API, after more baseline needs are in place

- A discussion regarding Bucket Quotas and Backups is scheduled for tomorrow with someone from the sig-storage-cosi core team who is also a maintainer at Ceph.
- Any Questions? That would be helpful to ask him

Thank you

Special thanks to my supervisors for answering my questions and their support.

Notes:
https://codimd.web.cern.ch/aQ6h8AjvTtyNH3M_pvY8tg