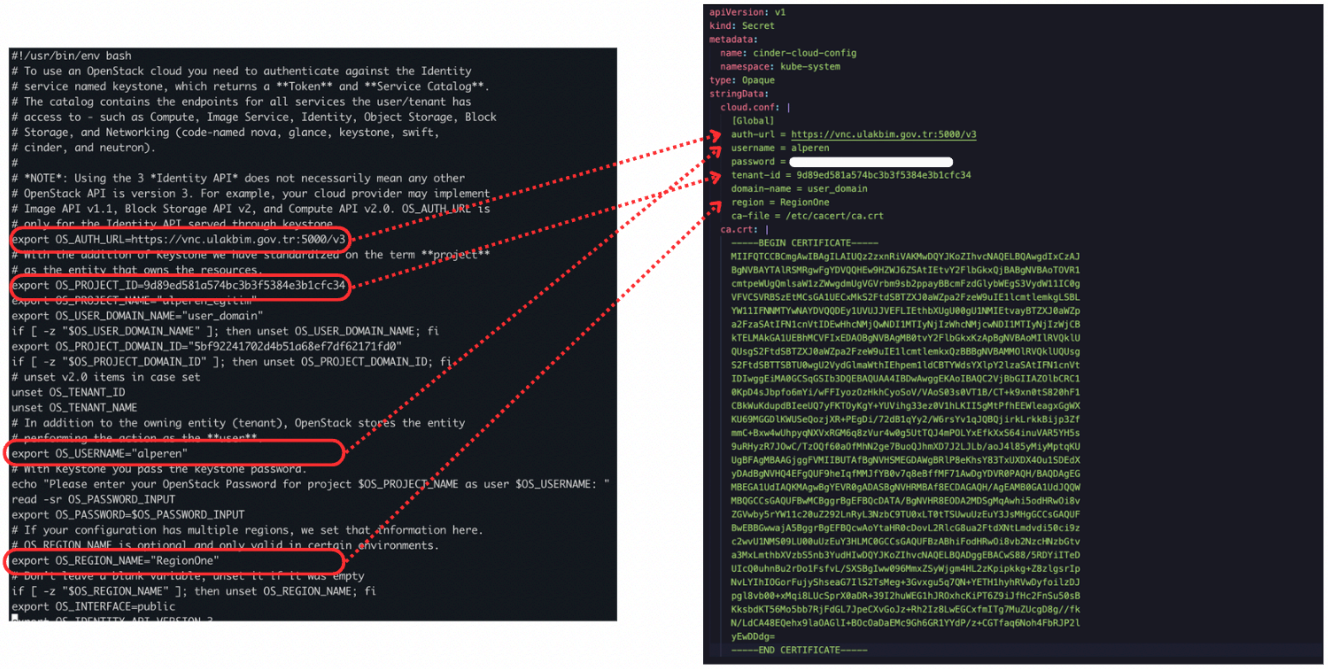
Cinder Csi Manifest Dosyalarının Düzenlenmesi

Güncel manifest dosyalarına erişmek için github adresine gidebiliriz https://github.com/kubernetes/cloud-provider-openstack. Burada manifests > cinder-csi-plugin dizinini kendi tarafımıza kopyalayarak düzenlemelere başlayabiliriz. Bu dizin altında 6 adet yaml dosyası mevcuttur bunlar:

- **cinder-csi-controllerplugin-rbac.yaml** (Pluginin controller kısmının Kubernetes API objelerinde sahip olduğu roller ve bunların bindingi mevcuttur.)
- **cinder-csi-controller-plugin.yaml** (Pluginin csi controller konteynırlarının bulunduğu ve sahip olduğumuz cloud konfigürasyonu ve sertifika mount edeceğimiz kısımdır.)
- **cinder-csi-nodeplugin-rbac.yaml** (Pluginin her bir node'da çalışacak ve volume claim edeceği daemonset kısmının rolleri ve bunların binding'ini içerir)
- **cinder-csi-nodeplugin.yaml** (Pluginin node registration ve csi socket bağlantılarının yaptığı işçi servislerinin daemonset pod dosyadır.)
- **csi-cinder-driver.yaml** (Pluginimizin sahip olduğu specler hakkında düzenleme yapılabilen kısımdır)
- **csi-secret-cinderplugin.yaml** (Openstack bağlantısınıın özelliklerini vereceğimiz ve kullanacağımız openstack'in sahip olduğu ve pluginin güveneceği kök sertifikayı vereceğimiz kısımdır.)

Öncelikle dosyaları çektikten sonra düzenleyeceğimiz kısımlar sırasıyla cinder-cloud-config secret objesidir **csi-secret-cinderplugin.yaml** dosyasında bulunmaktadır. İndirdiğimiz default kısmı aşağıdaki şekilde düzenlememiz gerekmektedir burada kullanacağımız verileri alabilmek için OpenStack arayüzü üzerinden OpenStack RC File indirmemiz gerekmektedir (arayüz üzerinden sağ yukarıdan kullanıcı kısmına tıklayarak bulunan menüden indirilebilir). Dosyayı indirip bir text editör ile açtıktan sonra içindeki bilgilere erişebiliriz.

Ardından bu bilgilere göre cinder-secret-cinderplugin secret dosyamızı düzenleyebiliriz. Aşağıda hangi bilgilerin nereye geleceğine dair bir yardımcı görsel bulunmaktadır.



Ekleyeceğimiz OpenStack kök sertifikası için ise aşağıdaki verilen komutlarla dosya oluşturulup sonrasında oluşan dosyanın içeriğini ca.crt şeklindeki kısma aynı şekilde eklenmesi gerekmektedir.

```
wget http://depo.kamusm.gov.tr/ssl/SSLSM.S2.cer -O kamusm-ssl-ca.cer
openssl x509 -in kamusm-ssl-ca.cer -inform der -out ca.pem -outform pem
# Copy the ca.pem file inside secret ca.crt section
cat ca.pem
```

Ardından düzenleyeceğimiz controller plugin ve node plugin dosyalarında ise volumes kısmında ayrıca kök sertifikanında konteynırlarımızın içerisinde /etc/cacert dizinine mount edildiğine emin olalım. İki volume ve volume mount aşağıdaki gibidir (**cinder-csi-controller-plugin.yaml** & **cinder-csi-nodeplugin.yaml**) :

```
#volume mount kısmı:
- name: secret-cinderplugin
  mountPath: /etc/config
  readOnly: true
- name: cacert
  mountPath: /etc/cacert
  readOnly: true
#volumes kısmı:
- name: secret-cinderplugin
  secret:
    secretName: cinder-cloud-config
- name: cacert
  secret:
```

```
    secretName: cinder-cloud-config
    items:
    - key: ca.crt
      path: ca.crt
```

Ardından cinder-csi-plugin'i deploy edebiliriz yaml dosyalarının olduğu dizinden

```
kubectl apply -f .
```

CSI plugin podlarımızın doğru çalıştığına emin olmak için deploy ettiğimiz namespace'den hepsinin sağlıklı bir şekilde Running durumunda olduğuna emin olalım.

```
kubectl get pod -n kube-system | grep -i cinder
```

Şimdi dinamik volume oluşturma için gerekecek StorageClass objemizi oluşturabiliriz. (availability kısmı için OpenStack üzerinden nereden volume oluşturulacak ona bakarak bu değeri alabiliriz). Kubernetes nodelerimizin cinder topolojisinde hangi alandan alacağını label olarak set'leyelim:

```
# Her bir node için tekrarlayalım rke2-worker-1 rke2-worker-2 ...
kubectl label node rke2-worker-X topology.cinder.csi.openstack.org/zone=nova
--overwrite
```

StorageClass objesi şu şekildedir.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-cinder-sc
provisioner: cinder.csi.openstack.org
reclaimPolicy: Delete
parameters:
  availability: nova
volumeBindingMode: WaitForFirstConsumer
```

Ardından volume claimi ve örnek bir pod ile kullanabiliyor muyuz test edebiliriz.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-cinder-pvc
spec:
```

```
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-cinder-sc
---
apiVersion: v1
kind: Pod
metadata:
  name: cinder-test-pod
spec:
  containers:
  - name: web-server
    image: nginx
    volumeMounts:
    - mountPath: /var/www/html
      name: data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: csi-cinder-pvc
```