Cloud native Multi tenancy proposal(Deprecated)

Motivation

In ICN, we required to share resources with multiple users and/or application. In the web enterprise segment, it is like multiple deployments team sharing the Kubernetes(K8s) cluster. In the case of Telco or Cable segment, we have multiple end users sharing the same edge compute resource. This proposal refers to the Kubernetes Multi-tenancy options and how to utilize it in ICN architecture and also to benefit Multi-tenancy use case in K8s

Goal(in Scope)

Focusing on the solution within the cluster for tenants, and working with Kubernetes SIG groups and adapt the solution in the ICN

Goal(Out of Scope)

Working in Kubernetes core or API is clearly out of the scope of these documents. There are the solutions available to provide a separate control plane to each tenant in a cluster, it is quite expensive and hard to have such a solution in a cloud-native space.

Outline

In this section, we define Multi-Tenancy in general for the Orchestration engine. A tenant can be defined as a group of resources bounded and isolated amount of compute, storage, networking and control plane in a kubernetes cluster. A tenant can also be defined as a group of users slicing a bounded resource allocated for them. These resources can be as follows:

- CPU, Memory, Extended Resources
- Network bandwidth, I/O bandwidth, Kubernetes cluster resource
- Resource reservation to provide the Guaranteed QoS in Kubernetes

Multi-tenancy can be distinguished as "Soft Multitenancy" and "Hard Multitenancy"

- Soft Multitenancy tenants are trusted(means tenant obey the resource boundary between them). One tenant should not access the resource of another tenant
- Hard Multitenancy tenants can't be trusted(means any tenant can be malicious, and there must be a strong security boundary between them), So one tenant should not have access to anything from other tenants.

Requirement

1. For a service provider, a tenant is basically a group of end-user sharing the same cluster, we have to make sure that the end user resources are tracked and accountable for their consumption in a cluster
2. In a few cases, admin or end-user application is shared among multiple tenants, in such case application resource should be tracked across the cluster
3. Centralization resource quota or the allocation limits record should be maintained by admin or for the end user. For example, just a kubectl "query" to Kubernetes API should display the resource quota and policy for each end-user or tenant
4. In Edge use case, the service orchestration like ICN should get the resource details across multiple clusters by resource orchestration, should set the resource allocation for the cluster and decide the scheduling mechanism
5. User credential centralization with application orchestration

Cloud Native Multi-tenancy Proposal - Tenant controller

Cloud Native Multi-tenancy proposal reuses the Kubernetes Multi-tenancy works to bind the tenant at the service orchestration and resource orchestration level.

Kubernetes Tenant project

> All the materials discussed in the following section are documented in the reference section link, and the contents are belongs to authors to respective doc

Kubernetes community working on Tenant controller that define tenant as Custom resource definition, CRD, and define the following elements.

Kubernetes Tenant controller creates a multi-tenant ready kubernetes cluster that allows the creation of the following new types of kubernetes objects/ resources:

a. A tenant resource (referred as “Tenant-CR” for simplicity)
b. A namespace template resource (referred as “NamespaceTemplate-CR” for simplicity)

Tenant resource: Is a simple CRD object for the tenant with namespace associated with the tenant name

Namespace template resource: Define like Role, RoleBinding, ResourceQuota, network policy for the namespace associated with tenant-CR

Tenant controller set up

Build the Tenant controller

The following steps explain how to run the tenant controller in kubernetes

Tenant controller Build

```
$ go get github.com/kubernetes-sigs/multi-tenancy
$ cd $GOPATH/src/github.com/kubernetes-sigs/multi-tenancy
$ cat <<EOF > $PWD/.envrc
export PATH="`pwd`/tools/bin:$PATH"
EOF
$ source .envrc
<<Have to install additional few golang package, the project is not having vendor folder>>
$ go get github.com/golang/glog
$ go get k8s.io/client-go/kubernetes
$ go get k8s.io/client-go/kubernetes/scheme
$ go get k8s.io/client-go/plugin/pkg/client/auth/gcp
$ go get k8s.io/client-go/tools/clientcmd
$ go get github.com/hashicorp/golang-lru
$ devtk setup
$ devtk build
<<Running tenant controller>>
$ $PWD/out/tenant-controller/tenant-ctl -v=99 -kubeconfig=$HOME/.kube/config
```

Tenant CRD definitions

The Tenant CRD object is defined by following CRD objects:

Tenant-CRD

```
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: tenants.tenants.k8s.io
spec:
group: tenants.k8s.io
versions:
- name: v1alpha1
served: true
storage: true
scope: Cluster
names:
plural: tenants
singular: tenant
kind: Tenant
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: namespacetemplates.tenants.k8s.io
spec:
group: tenants.k8s.io
versions:
- name: v1alpha1
served: true
storage: true
scope: Cluster
names:
plural: namespacetemplates
singular: namespacetemplate
kind: NamespaceTemplate
shortNames:
- nstpl
```

Let's run the tenant controller and create a tenant object as follows:

Tenant object

```
$ kubectl create -f $GOPATH/src/github.com/kubernetes-sigs/multi-tenancy/poc/tenant-controller/data/manifests
/crd.yaml
$ kubectl create -f $GOPATH/src/github.com/kubernetes-sigs/multi-tenancy/poc/tenant-controller/data/manifests
/rbac.yaml
<<Running tenant controller>>
$ $GOPATH/src/github.com/kubernetes-sigs/multi-tenancy/out/tenant-controller/tenant-ctl -v=99 -
kubeconfig=$HOME/.kube/config

$ # kubectl get crd
NAME                                          CREATED AT
namespacetemplates.tenants.k8s.io             2019-05-01T16:34:49Z
tenants.tenants.k8s.io                        2019-05-01T16:34:49Z

$ kubectl create -f $GOPATH/src/github.com/kubernetes-sigs/multi-tenancy/poc/tenant-controller/data/manifests
/sample-nstemplate.yaml
$ kubectl create -f $GOPATH/src/github.com/kubernetes-sigs/multi-tenancy/poc/tenant-controller/data/manifests
/sample-tenant.yaml
$ kubectl get tenants
NAME       AGE
tenant-a   7d

$ kubectl get ns
NAME            STATUS   AGE
default         Active   42d
kube-public     Active   42d
kube-system     Active   42d
tenant-a-ns-1   Active   7d18h
tenant-a-ns-2   Active   7d18h
```

A closer look into tenant Object

The tenant object looks like below:

tenant object

```
---
apiVersion: tenants.k8s.io/v1alpha1
kind: Tenant
metadata:
  name: tenant-a
spec:
  namespaces:
      - name: ns-1
      - name: ns-2
```

The tenant controller takes this tenant spec as a template to creates the namespace for each namespace as tenant-a-ns1, tenant-a-ns2. In addition, the tenant object can also create an admin object for a tenant with a user for admin.

In addition, it creates a namespace template, it defines templates that define Rolebinding, ClusterRole, NetworkPolicy for the namespace tenant-a-ns1 and tenant-a-ns2.

namespaceTemplate

```
$ kubectl get namespacetemplate
NAME         AGE
restricted   7d

$ kubectl get namespacetemplate restricted -o yaml
apiVersion: tenants.k8s.io/v1alpha1
kind: NamespaceTemplate
metadata:
  creationTimestamp: "2019-05-01T17:37:11Z"
  generation: 1
  name: restricted
  resourceVersion: "3628408"
  selfLink: /apis/tenants.k8s.io/v1alpha1/namespacetemplates/restricted
  uid: bffbe9c8-6c37-11e9-91c3-a4bf014c3518
spec:
  templates:
  - apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    metadata:
      name: multitenancy:podsecuritypolicy
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: ClusterRole
      name: multitenancy:use-psp:restricted
    subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: Group
      name: system:serviceaccounts
  - apiVersion: networking.k8s.io/v1
    kind: NetworkPolicy
    metadata:
      name: multitenancy-default
    spec:
      podSelector: {}
      policyTypes:
      - Ingress
      - Egress
```

Resource quota proposal for the tenant CRD

A tenant-based resource quota is required to implement resource tracking in ICN. The proposal here is to reuse the tenant controller work in Kubernetes and introduce the tenant resource quota CRD on the top of tenant controller

Tenant resource quota

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: tenantresourcequota.tenants.k8s.io
spec:
  group: tenants.k8s.io
  versions:
    - name: v1alpha1
      served: true
      storage: true
  scope: Cluster
  names:
    plural: tenantresourcequotas
    singular: tenantresourcequota
    kind: TenantResourcequota
    shortNames:
    - trq
```

Before jumping into the definition of the tenant resource quota, refer Kubernetes resource quota - https://kubernetes.io/docs/concepts/policy/resource-quotas/ for more understanding on the resource quota

And tenant resource quota schema should be like this below

tenant resourcequota

```
// NamespaceTemplate defines a template of resources to be created inside a namespace.
type TenantResourceQuota struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec TenantResourceQuotaSpec `json:"spec"`
}

// TenantResourceQuotaSpec defines the desired hard limits to enforce for Quota
type TenantResourceQuotaSpec struct {
        // Hard is the set of desired hard limits for each named resource
        // +optional
        Hard v1.ResourceList `json:"hard"`
        // A collection of filters that must match each object tracked by a quota.
        // If not specified, the quota matches all objects.
        // +optional
        Scopes []v1.ResourceQuotaScope `json:"scopes"`
        // ScopeSelector is also a collection of filters like Scopes that must match each object tracked by a
quota
        // but expressed using ScopeSelectorOperator in combination with possible values.
        // +optional
        ScopeSelector *v1.ScopeSelector `json:"scopeSelector"`
    UserResourcequota  []string `json:"userResourcequota"`
}
```

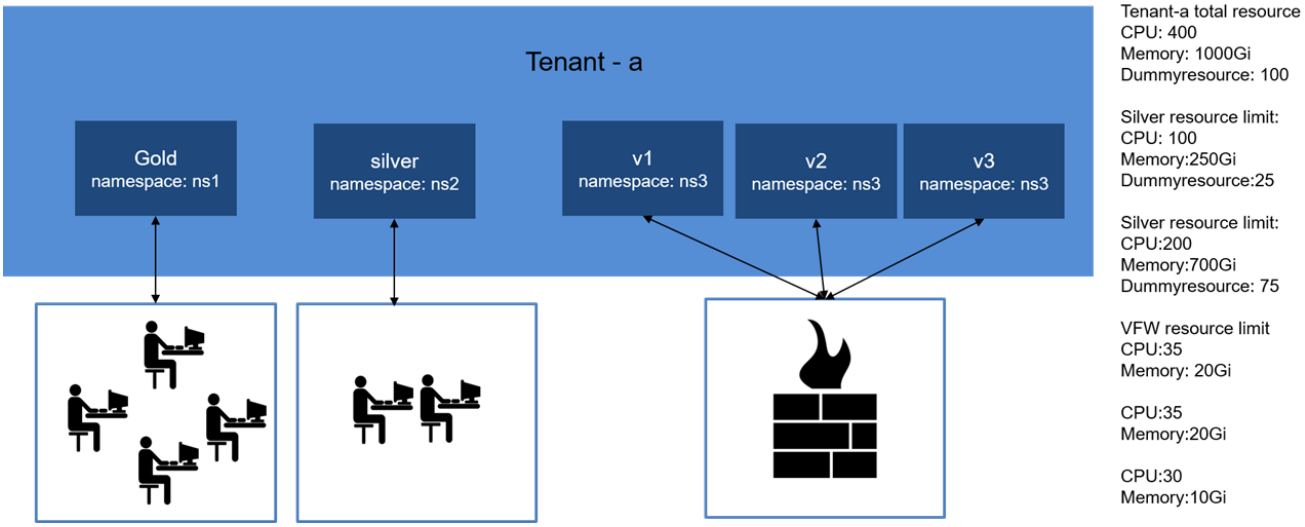
And the example Tenant resource quota should be like this.

tenant resourcequota

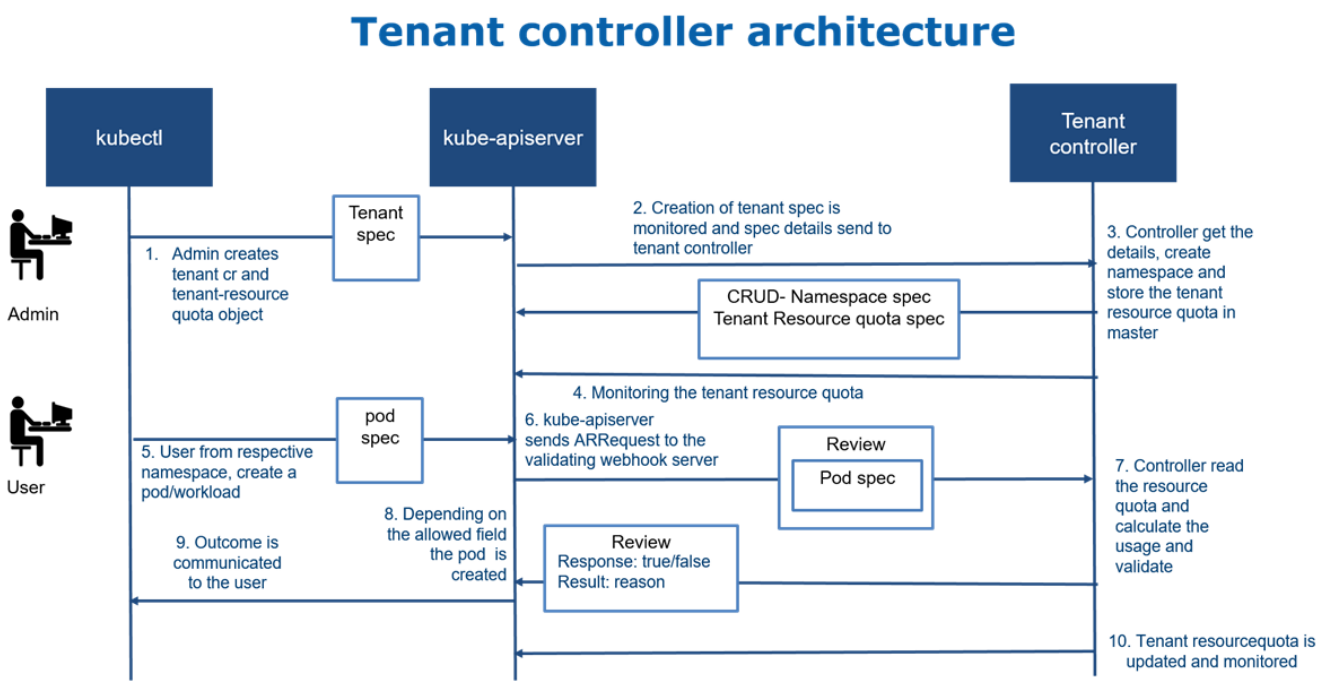
```
---
apiVersion: tenants.k8s.io/v1alpha1
  kind: TenantResourcequota
  metadata:
    name: Tenant-a-resource-quota
  spec:
    hard:
      cpu: "400"
      memory: 1000Gi
      pods: "500"
          requests.dummy/dummyResource: 100
    scopeSelector:
      matchExpressions:
      - operator : In
        scopeName: Class
        values: ["common"]
    userResourcequota:[
                      "silver-pool-resourcequota",
                                    "gold-pool-resourcequota",
                                    "vfirewall-pool-resourcequota"
                  ]
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: silver-pool-resourcequota
  namespace: tenant-a-ns-2
spec:
  hard:
    limits.cpu: "100"
```

```
    limits.memory: 250Gi
        pod: 100
    requests.dummy/dummyResource: 25
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: gold-pool-resourcequota
  namespace: tenant-a-ns-1
spec:
  hard:
    limits.cpu: "200"
    limits.memory: 700Gi
        pod:300
    requests.dummy/dummyResource: 75
---
apiVersion: v1
kind: List
items:
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: vfirewall-v1
    namespace: tenant-a-ns-3
  spec:
    hard:
      cpu: "35"
      memory: 20Gi
      pods: "50"
    scopeSelector:
      matchExpressions:
      - operator : In
        scopeName: Firewall
        values: ["v1"]
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: vfirewall-v3
    namespace: tenant-a-ns-3
  spec:
    hard:
      cpu: "30"
      memory: 10Gi
      pods: "15"
    scopeSelector:
      matchExpressions:
      - operator : In
        scopeName: Firewall
        values: ["v3"]
```

Block diagram representation of tenant resource slicing



Tenant controller architecture



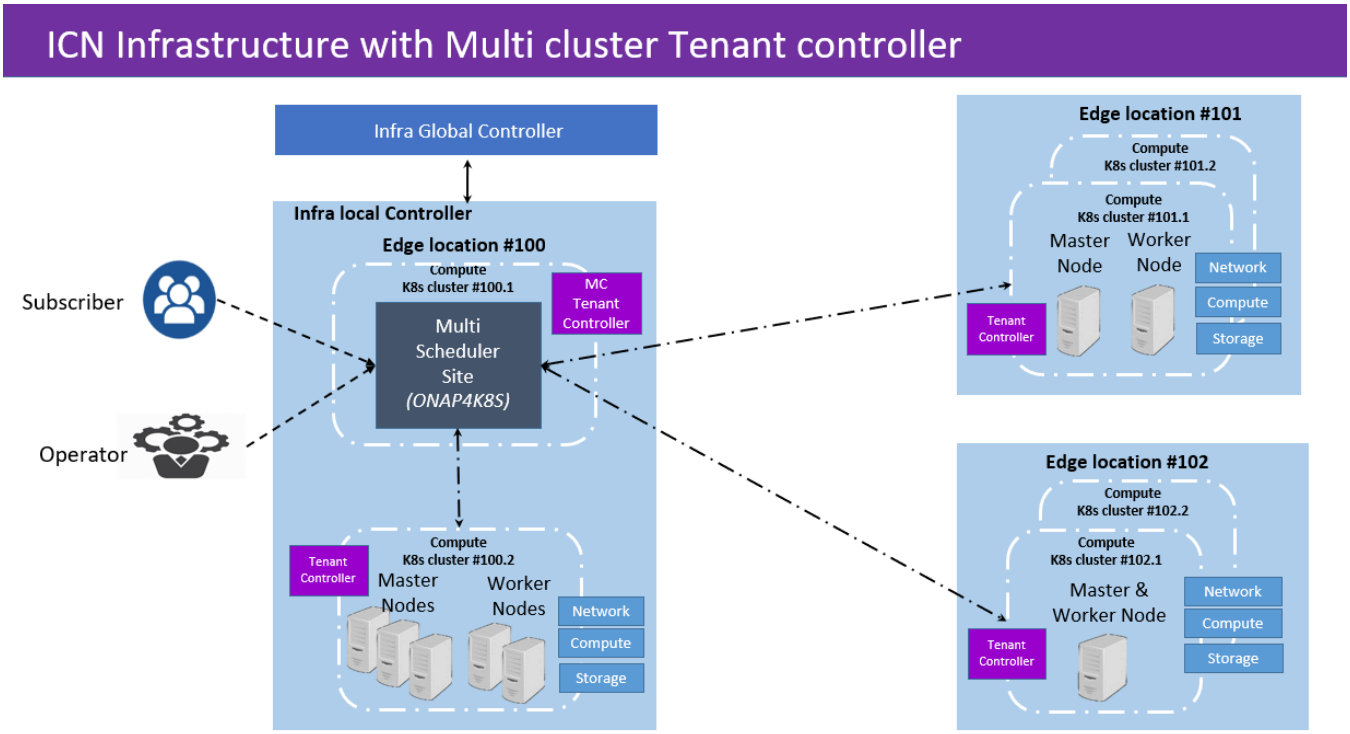
ICN Requirement and Tenant controller gaps

ICN Requirement	Tenant Controller
Multi-cluster tenant controller 1. Tenant created at Multi scheduler site (ONAP4K8S)	Cluster level tenant controller
Identifying K8S clusters for this tenant based on cluster labels 1. Send the Tenant details to the K8s cluster	Tenant is created with CR at cluster level **[Implemented]**

At K8s cluster level 1. Creating namespace 2. Creating K8S users (Tokens, Certificates and User/Pwds) 3. Creating K8S roles 4. Creating permissions to various roles.	1. Tenant controller at K8s cluster level **[Implemented]** a. A tenant can have multiple namespaces i. Tenant-a 1. ns1 2. ns2 ii. It creates Tenant-a-ns1 and Tenant-a-ns 2. **Cluster-admin**: This entity has full read/write privileges for all resources in the cluster including resources owned by various Tenants of the cluster **[Not implemented].** 3. **Cluster-view:** This entity has read privileges for all resources in the cluster including reasources owned by various Tenants **[Not implemented].** 4. **Tenant-admin:** This entity has privileges to create a new tenant, read/write resources scoped to that Tenant and update or delete that Tenant. This persona does not have any privileges for accessing resources that are either cluster-scoped or scoped to namespaces that are not associated with the Tenant object for which this persona has Tenant-admin privileges.**[Implemented]** 5. **Tenant-user:** This entity has read/write privileges for all resources scoped within a specific Tenant (that is resources that are scoped within namespaces that are owned by a specific Tenant) **[Not implemented].**
Certificate Provisioning with Tenant • Suggestion to use Isito using citadel	Suggestion to bind the tenant with kubernetes context to see namespaces associated with it**[Not implemented].**
• Quota at the application level. • Tenant group support: Quota at the tenant group level (Multiple namespaces), ISTIO at the tenant group level.	• Resource quota based on the tenant with multiple namespaces **[Not implemented].**

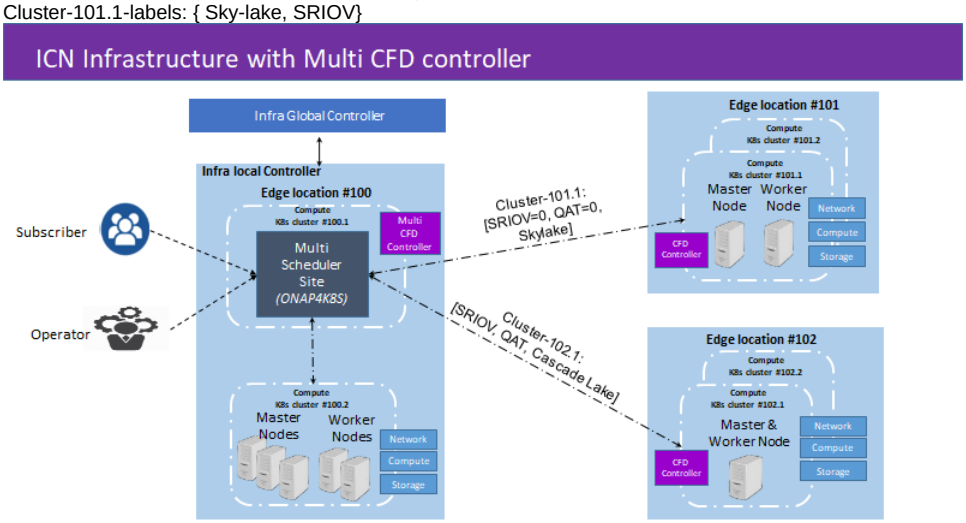
Multi-Cluster Tenant controller

<This section is incomplete and a work in progress ... needs rework and further updates ... >

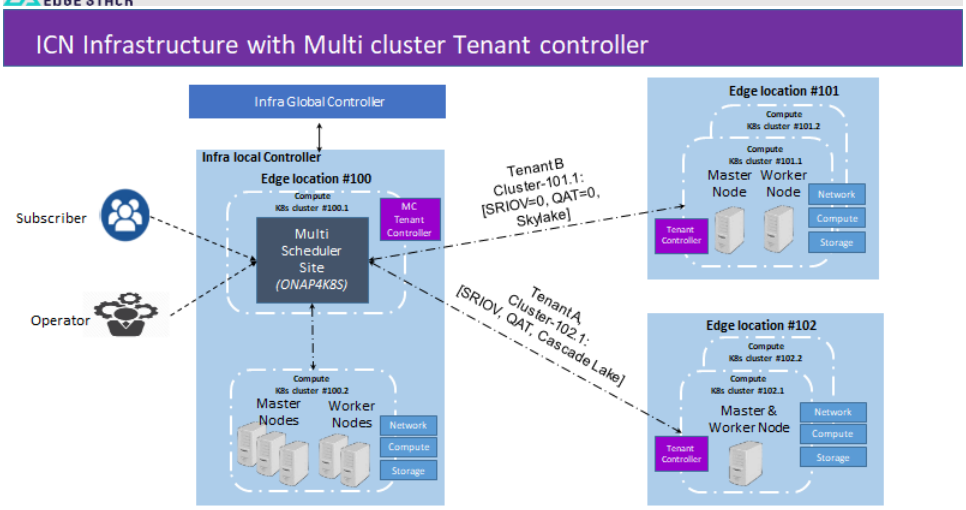


1. Define CRUD API - add/delete/modify/read MC Tenant.
    a. Cluster from the the Edge location are registered to the ONAP as follows :

i. Cluster-100.2-labels: { Cascade, SRIOV, QAT,}
ii. Cluster-101.1-labels: { Sky-lake, SRIOV}



iii.



iv.

b. Tenant Object is template in ONAP with following filed

```
{
        "metadata": {
                "name": "tenant-a",
                "clusterlabels": "label-A"
        },
        "spec": {
                "users": [
                        {
                                "name": "users-1",
                                "crt": "/path/to/crt"
                        },
                        {
                                "name": "users-2",
                                "crt": "/path/to/crt"
                        }
                ]
        }
}
```

c. ONAP create the Tenant based on the cluster labels
d. Find the cluster Artifacts kubeconfig based on the cluster labels
e. Get the Multicloud k8s-plugin API to create the Tenant json
    i. `curl -d @create_tenant-a.json http://NODE_IP:30280/api/multicloud-k8s/v1/v1/tenant`
f. MC Cluster Tenant API - <This section is incomplete and a work in progress ... needs rework and further updates ... >
    i. Create

ii. Update
iii. Delete
iv. Get
v. List
vi. Watch
vii. Patch

g. Each corresponding MC Cluster Tenant API will have a K8s Tenant CR API

1. Design note :
   - On how this would be done as Micro-service in the ONAP.
   - How does interact with K8S clusters.
   - How does it ensure that all the configuration is applied (rollbacks, unsuccessful edges).
   - Visibility of the configuration applied on per MCTenant basis.
   - When new K8S cluster is added with the label of interest, taking care of creating tenant-specific information in that edge etc..
   - Extensibility (future K8S clusters having some other features that require configuration for multi-tenancy).

Open Questions:

Srinivasa Addepalli

1. Slice the tenant with the cluster "--context"
   a. [Kural]
      i. Tenant creation from the ONAP4K8s should be shared down to the cluster in the edge location
      ii. Tenant should have kubeconfig context a slice of their namespace alone
2. How to connect the istio Citadel certificates with Tenant? how to authenticate from the centralised location from onap4k8s to multi-cluster location?
   a. [Kural]
      i. Discuss so far with Istio folks and expertise, suggested that citadel certificate are bonded to namespace and specific for the application level. They are not targeted for the K8s Users
      ii. For the k8s user, the certificates should be generated by the external entity and bind to the service account and the tenant as shown in the example - https://docs.bitnami.com/kubernetes/how-to/configure-rbac-in-your-kubernetes-cluster/
3. Tenant user bind to the certificates created from Citadel?
   a. [kural]
      i. Initial Pathfinding show that Citadel may not be the right candidate for the K8s User certificate creation
4. How the cluster labels are configured in ONAP? how the MC tenant controller can identify them?
   a. [kural]
      i. Adding KUD and ONAP folks here Srinivasa Addepalli Akhila Kishore @Ritu @Kiran Itohan Ukponmwan Enyinna Ochulor
      ii. Kubeconfig context should be passed from each KUD cluster to the ONAP
      iii. KUD should invoke NFD immediately and enable the overall labels. And add those labels to cluster details and send back to the ONAP
      iv. Cluster feature Discovery controller should be there in each Edge location cluster along with KUD, Run for each interval along with the NFD

JIRA Story details

Reference

Kubernetes Multi-Tenancy Draft Proposal
Tenant Concept in Kubernetes

Kubernetes Tenant CRD
K8s Multi-tenancy WG Plan