A Container-as-a-Service solution for CloudVeneto

Lisa Zangrando

Federica Fanzago, Matteo Migliorini, Sergio Traldi

Workshop sul Calcolo nell'INFN

May 23-27, 2022, Paestum

Cloud models for running containers on cloud

Kubernetes-as-Service

Kubernetes-as-a-Service (KaaS) is a cloud model which enables end users to deploy and manage Kubernetes clusters in on-demand and self-service mode.

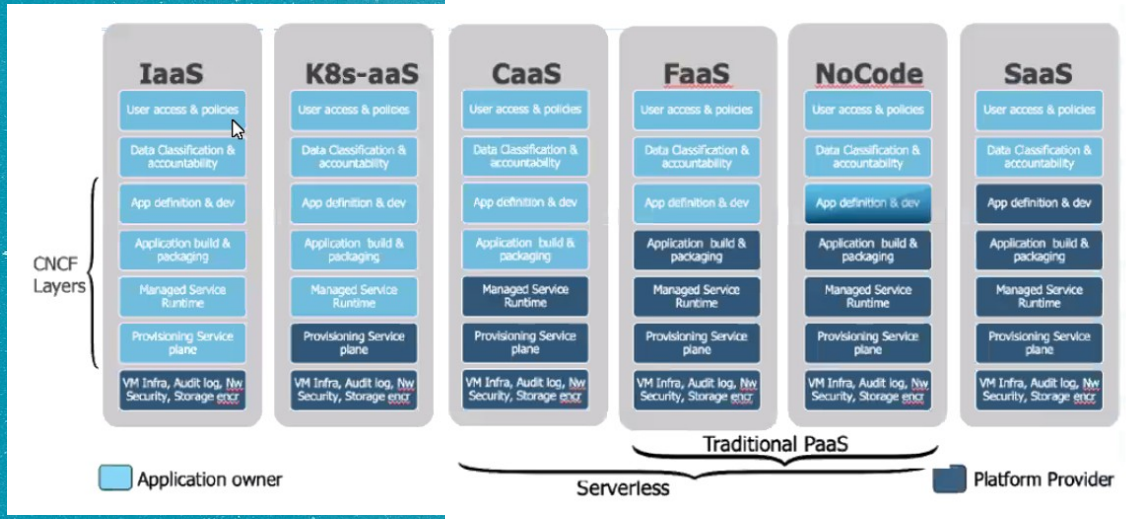
Examples: Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS)

Container-as-Service

Containers-as-a-Service (CaaS) is a cloud model that enables end users to execute their containers on cloud resources. Container engines, orchestration and the underlying compute resources are delivered to users as a service by the cloud provider.

Examples: Amazon Elastic Container Service (ECS), Amazon Fargate, and Azure Container Instances (ACI).

KaaS or CaaS?



KaaS

The user accesses its private K8s cluster.
The deployment + configuration takes time.
The provider makes just the cluster provisioning, the user has to fully administrate it.

CaaS

The user does not have a private cluster so no administrative skills are required.
The provider fully manages the K8s cluster including security and resource provisioning (Serverless).

KaaS vs CaaS efficiency

KaaS

The efficiency in terms of resource utilization degrades if the K8s clusters are not used for long time.
The cost (time) spent to recreate a cluster as needed is too much by the user.

CaaS

The provider fully manages the resource provisioning according to the Serverless model.

Implementing Caas is more complex than KaaS.

Our CaaS solution

A CaaS based on Kubernetes and OpenStack for CloudVeneto

It must include some KaaS features such as the ability to create private logical clusters

Several questions & issues

Resources

Which ones? How to manage them: pooling or partitioning? How to fairly assign them to the users?

Users

How to manage users (access, groups)
How to share the same cluster with multiple users?

Security

How to make strict isolation? How to avoid resource consumption abuse?

Kubernetes

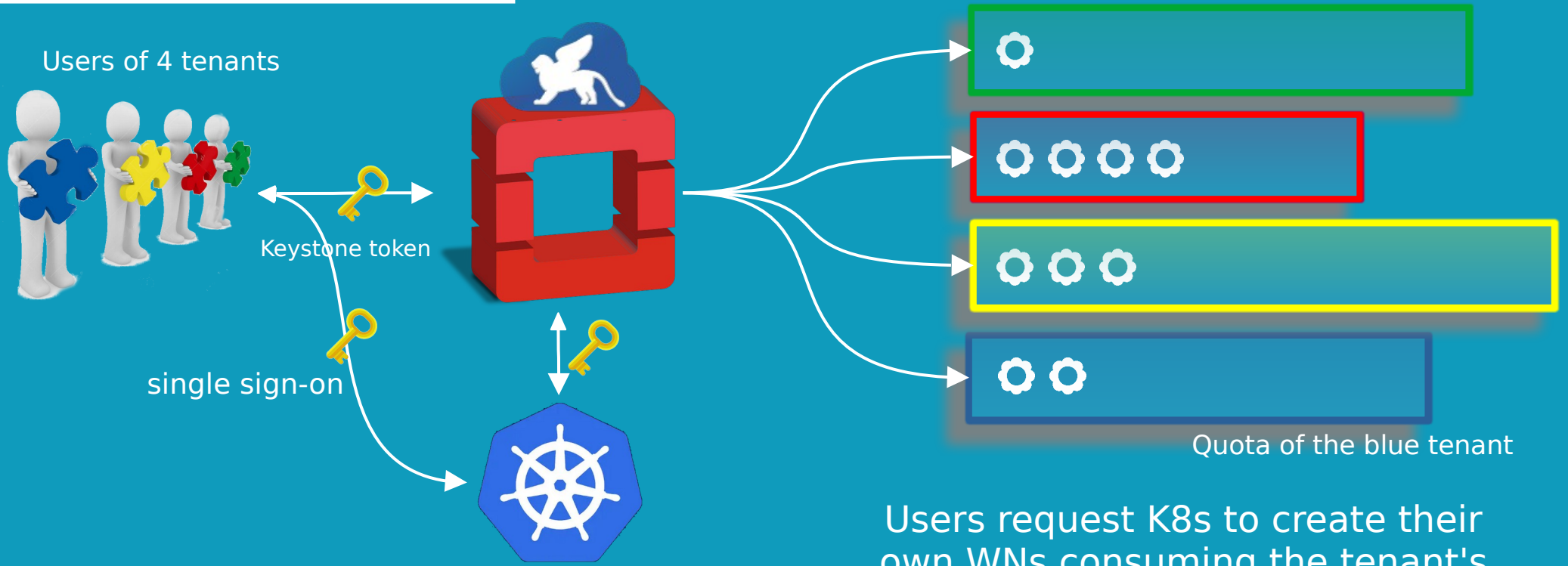
Does K8s interact with OpenStack?

Multi-tenancy

Software architecture where a single software instance (i.e. OpenStack) can serve multiple, distinct user groups (tenants/projects/namespaces). It supports customization for tenants (i.e. quotas) and allows strict isolation at tenant and user level.

Openstack: **hard** multi-tenancy (strict isolation at tenant and user level)
Kubernetes: **soft** multi-tenancy (strict isolation at tenant level)

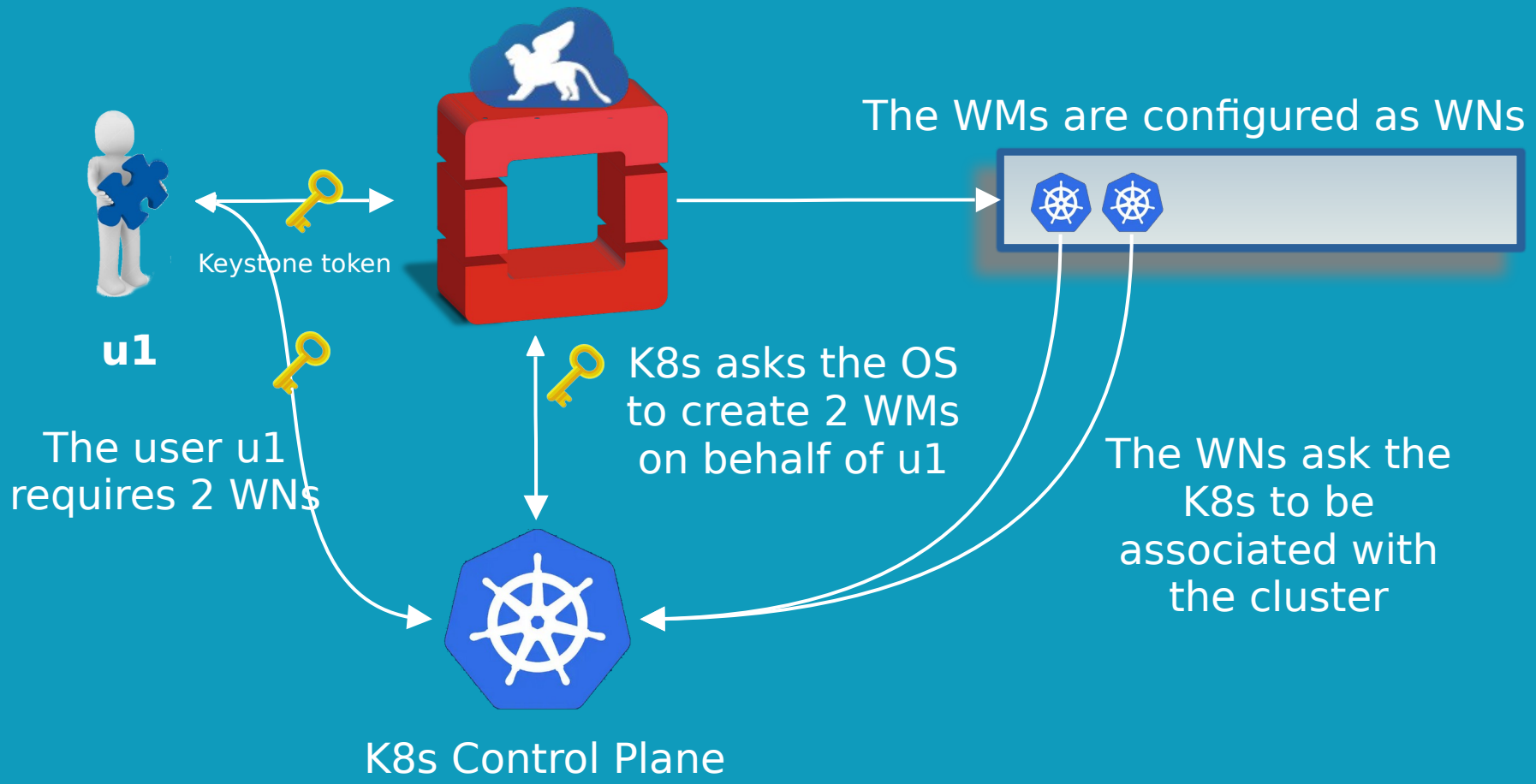
The high level architecture of our CaaS



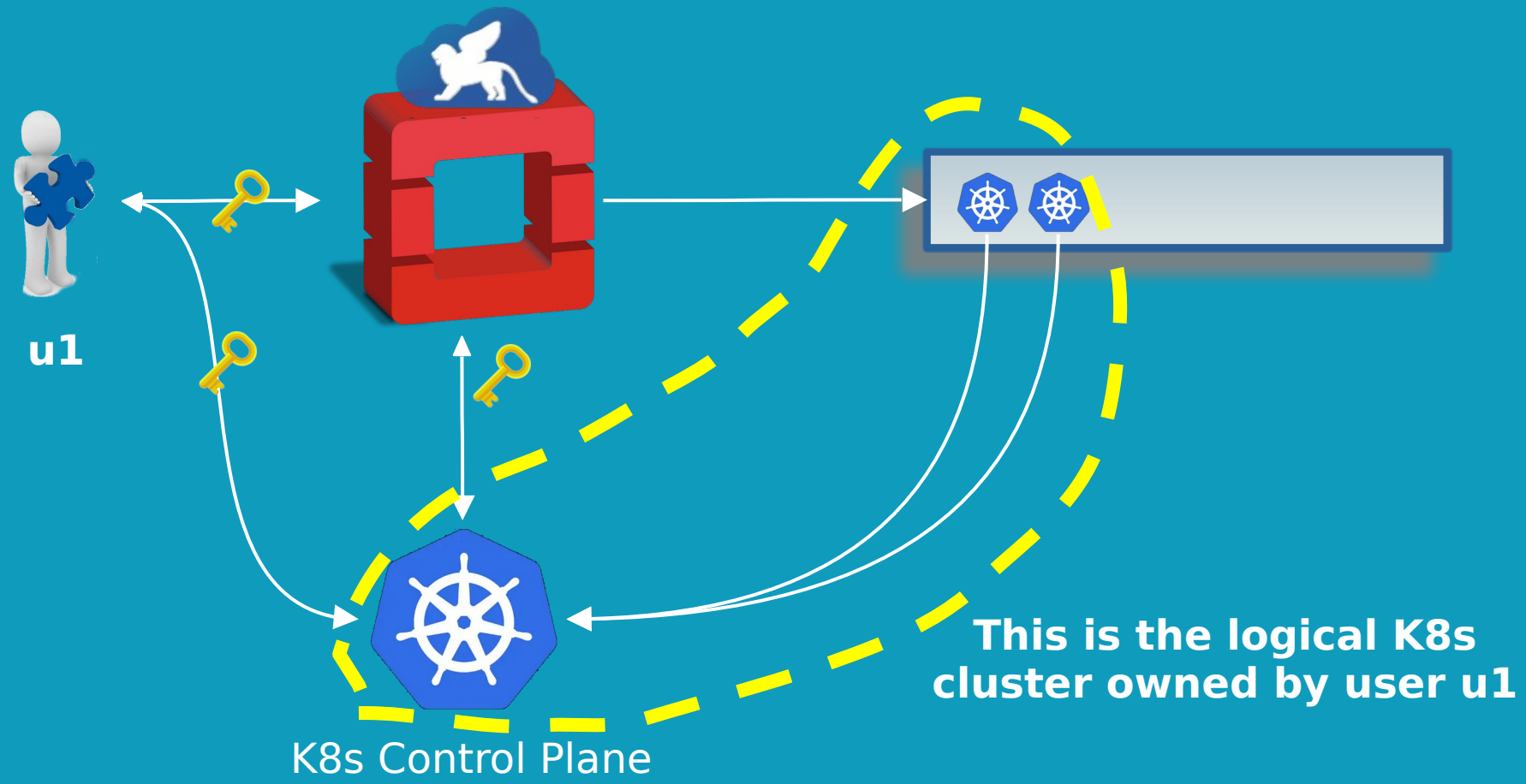
Kubernetes Control plane
(3 masters in HA)
without worker nodes

Users request K8s to create their own WNs consuming the tenant's quota in CloudVeneto.

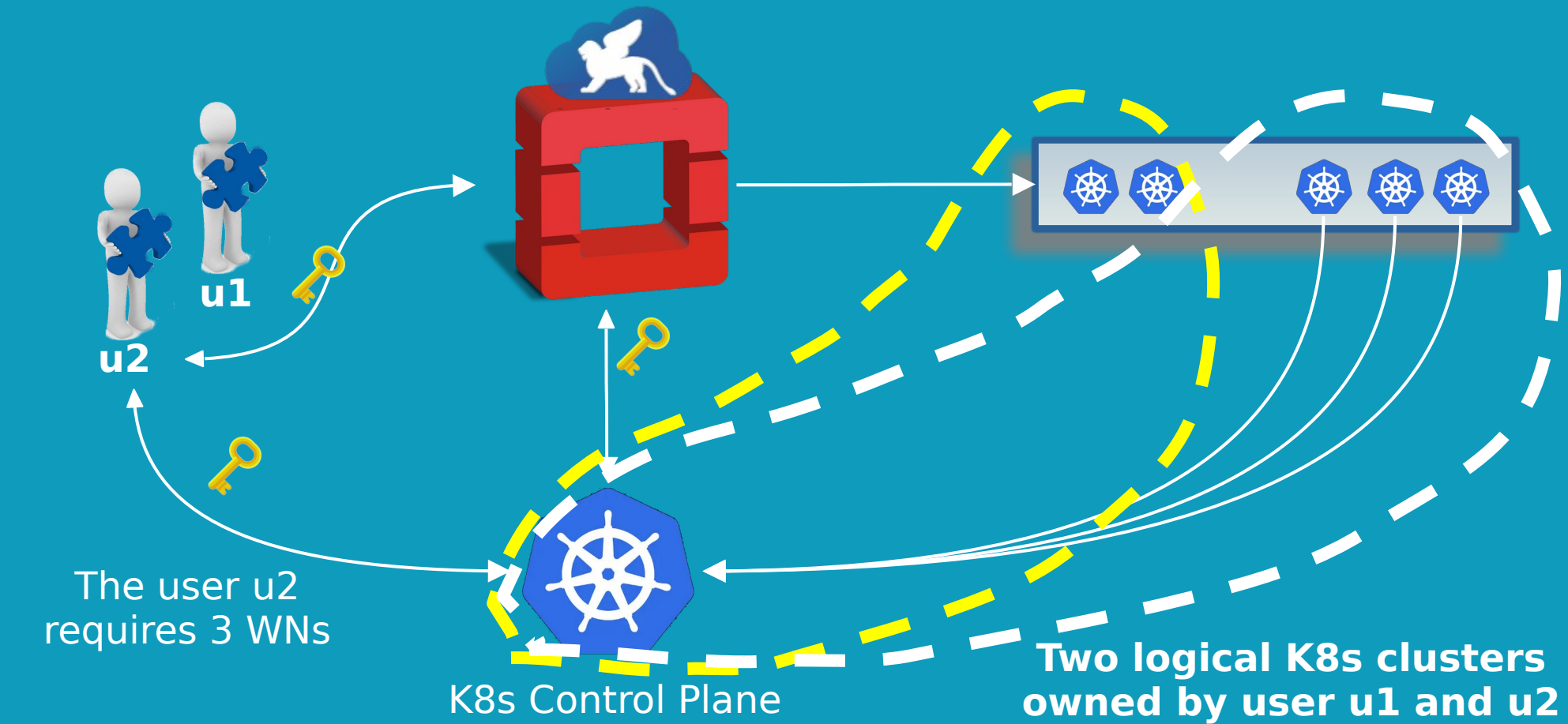
The high level architecture of our CaaS



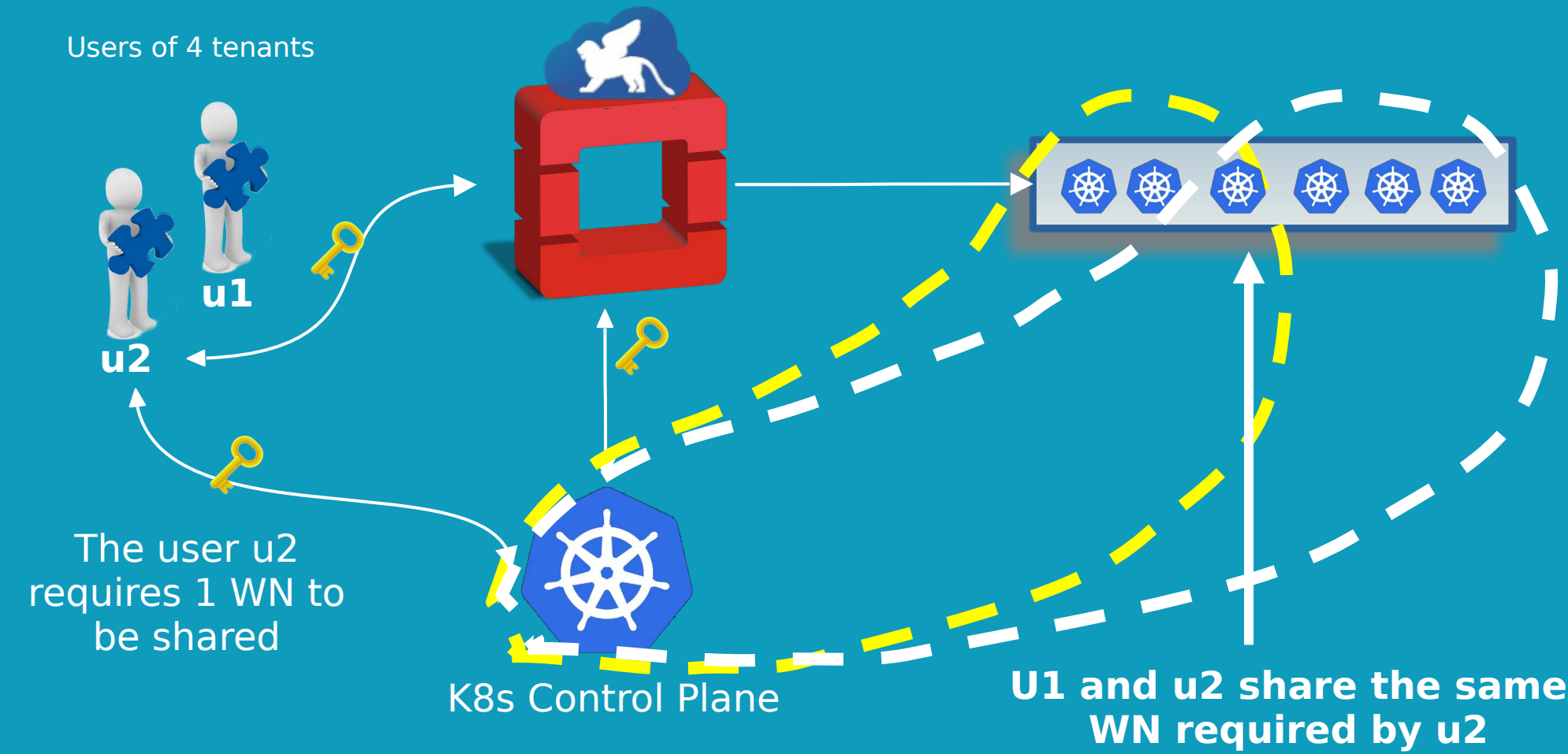
The high level architecture of our CaaS



The high level architecture of our CaaS



The high level architecture of our CaaS



How to extend the K8s functionalities

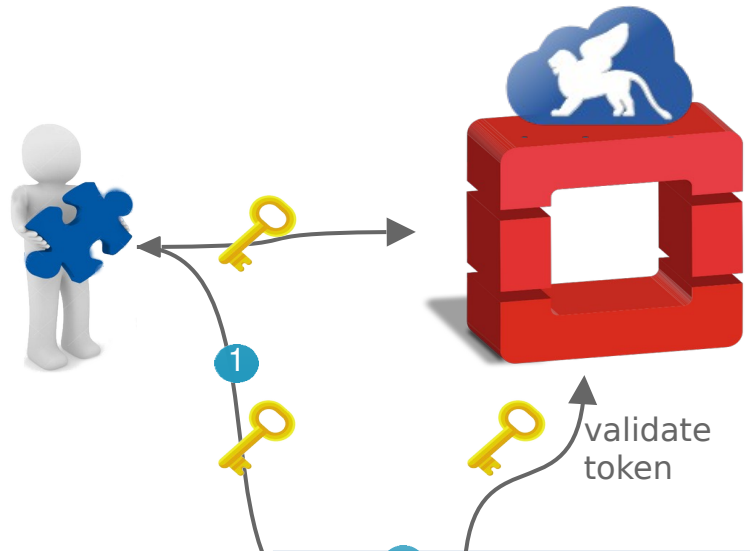
Kubernetes operator

An operator is a K8s API that allows developers to extend the Kubernetes capabilities by defining a new resource type (Custom Resource Definition) and implementing its manager.

Kubernetes webhooks

The webhook is a powerful mechanism to extend the Kubernetes API-servers capabilities with custom code for authentication, authorization and admission control.

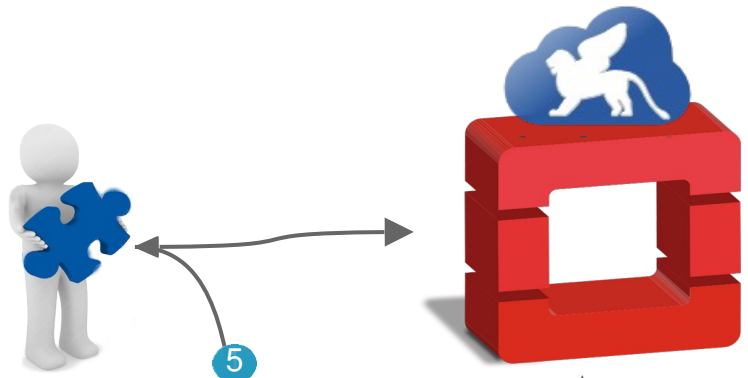
Authentication webhook



Kubernetes by default doesn't support the Keystone authN.
Added Keystone support by implementing a specific webhook.

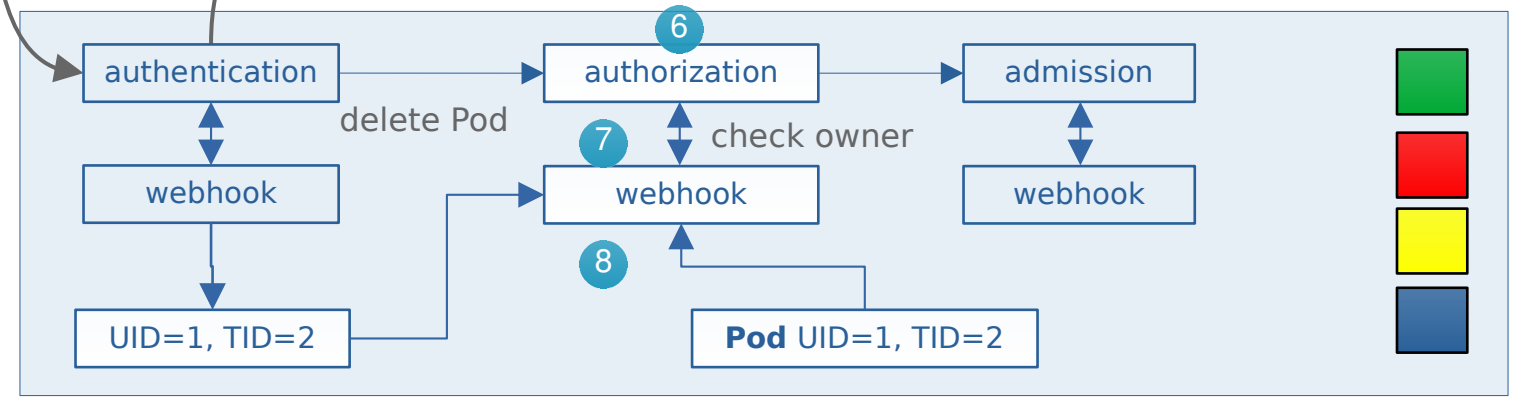
1) A CloudVeneto user (blue tenant) accesses to K8s cluster by using its Keystone credentials via kubectl
2) the K8s authN layer receives the request and asks to its webhook to validate the Keystone token (since it is not able to do it by itself)
3) the authN webook validates the token and, if valid, extracts the info (UserID and TenantID) about the user and its tenant
4) the webhook creates the namespace if not existing, and associate the user to it.

Authorization webhook (multi-tenancy)



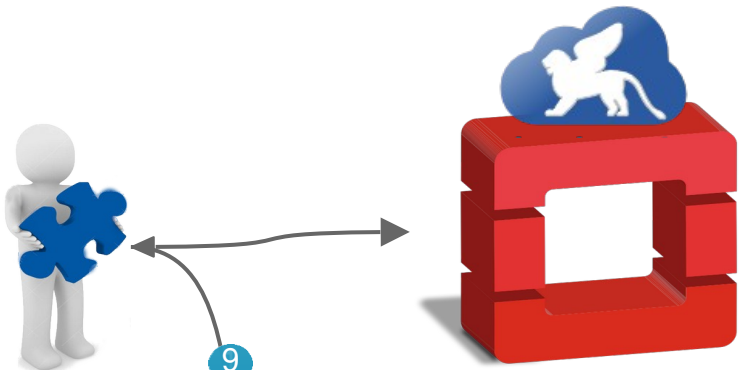
Webhook implemented to extend the authZ level in order to achieve strict isolation at the user level.

5) The authenticated user requires a pod deletion (or any verb: edit, get, delete, create, update)
6) the authZ layer enforces the request against the RBAC (Role Based Access Control) policies
7) if successfully, it asks its webhook to check the ownership of the pod (or any other kind of resource)
8) the authZ webhook compares the match of the user info provided by the authN webhook with the pod authN labels inserted by the admission webhook (see next slide)



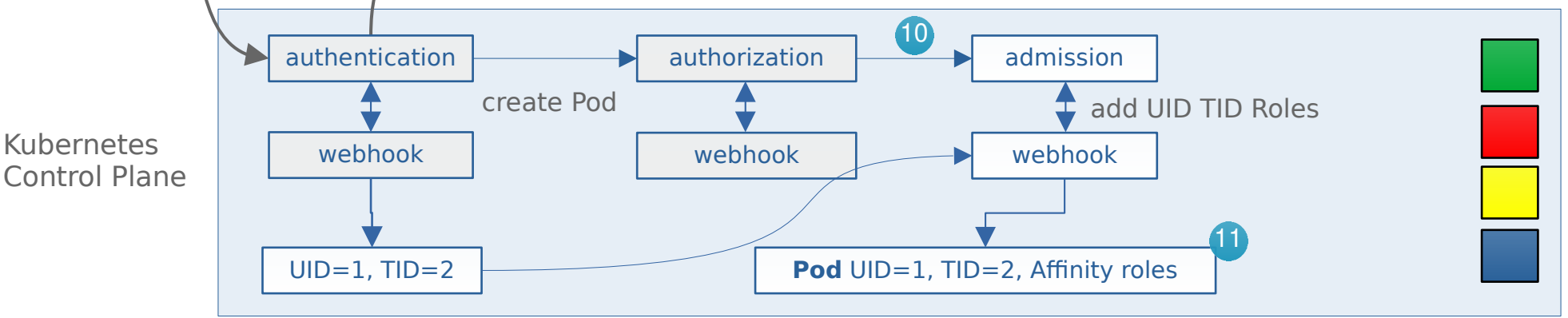
Kubernetes Control Plane

Admission webhook (multi-tenancy)



The admission webhook validates the user request and adds, if needed, some extra metadata for internal things (e.g. scheduler)

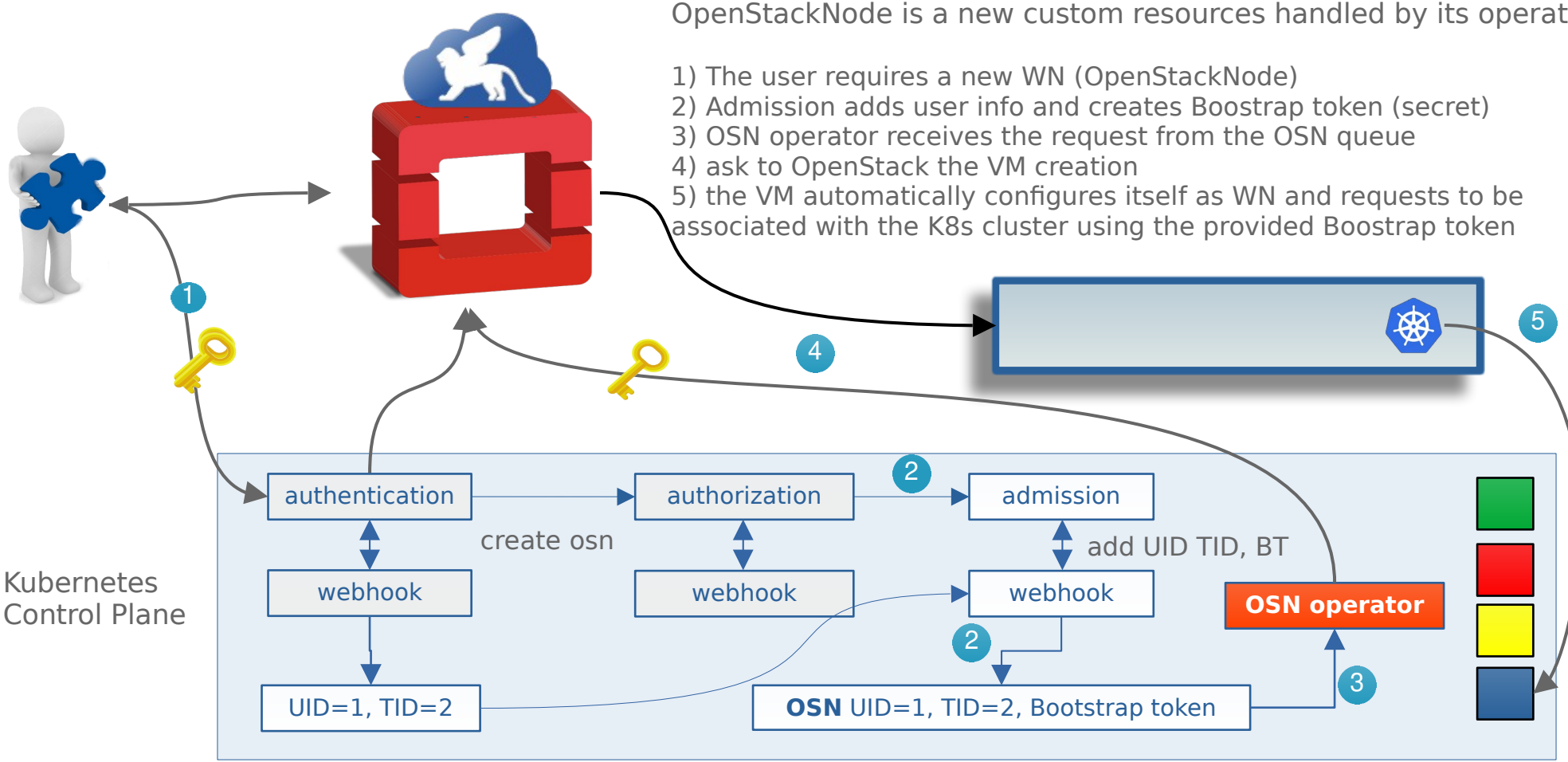
9) The authenticated user requires a **Pod creation** (or any resource)
10) the authZ layer authorizes the request and asks the admission layer to add the info (UserID and TenantID) about the user and its tenant within the pod metadata (labels)
11) in case of creation of the Pod, the admission webhook also adds some affinity & toleration roles to force the execution of the Pod only in the WNs owned by the user (or shared)



OpenStackNode operator

OpenStackNode is a new custom resources handled by its operator

1) The user requires a new WN (OpenStackNode)
2) Admission adds user info and creates Boostrap token (secret)
3) OSN operator receives the request from the OSN queue
4) ask to OpenStack the VM creation
5) the VM automatically configures itself as WN and requests to be associated with the K8s cluster using the provided Boostrap token



OpenStackNode custom resource

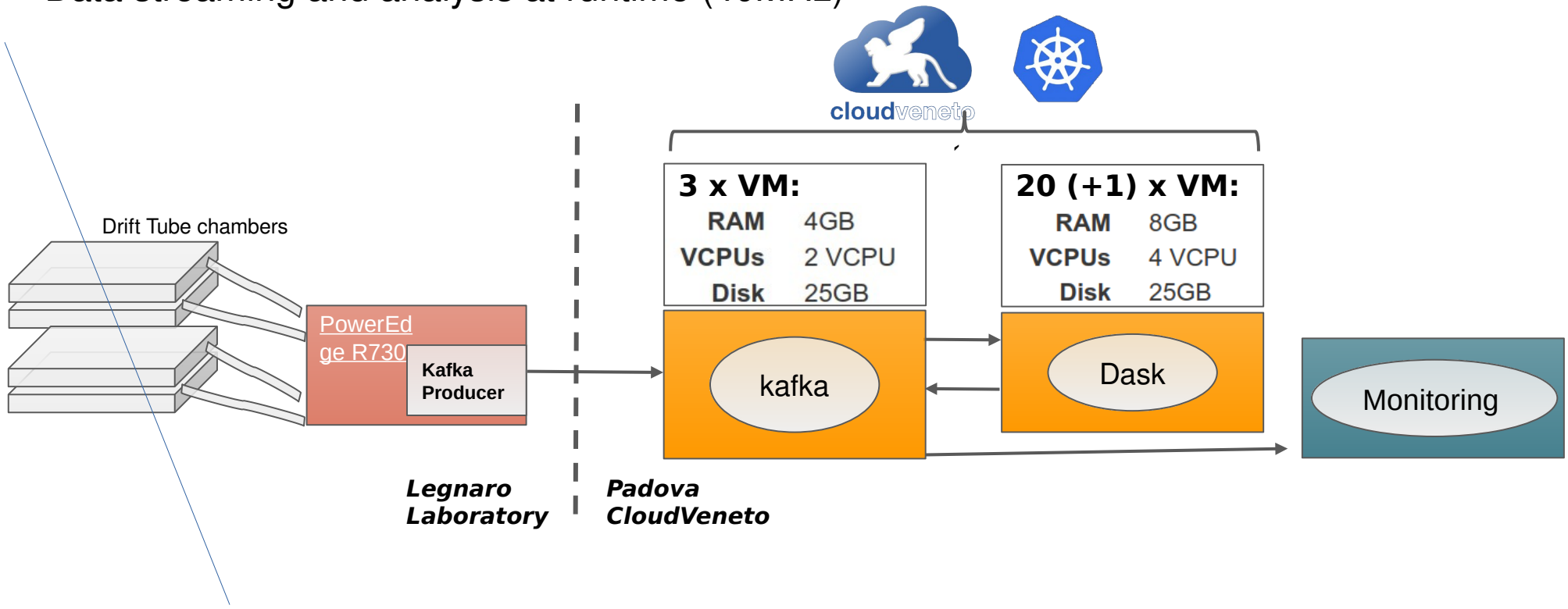
```
---
apiVersion: osnode.infn.it/v1
kind: OpenStackNode
metadata:
  name: my-node-01
spec:
  flavor: cloudveneto.large
  keyPair: Lisa
  policy: private
  provider: CloudVeneto
  securityGroups:
  - default
```

OpenStackNode example

```
$ kubectl apply -f my-node-01.yml


$ kubectl get osn my-node-01 -o wide
NAME          PHASE      OWNER                  POLICY    PROVIDER      VM FLAVOR           VM STATUS   VM IPV4        AGE
my-node-01    Building   zangrand-at-infn.it    private   CloudVeneto   cloudveneto.large   BUILD                      2m10s

$ kubectl get osn my-node-01 -o wide
NAME          PHASE      OWNER                  POLICY    PROVIDER      VM FLAVOR           VM STATUS   VM IPV4        AGE
my-node-01    Available  zangrand-at-infn.it    private   CloudVeneto   cloudveneto.large   ACTIVE      10.64.22.127   3m53s

$ kubectl get osn my-node-01 -o wide
NAME          PHASE      OWNER                  POLICY    PROVIDER      VM FLAVOR           VM STATUS   VM IPV4        AGE
my-node-01    Joining    zangrand-at-infn.it    private   CloudVeneto   cloudveneto.large   ACTIVE      10.64.22.127   4m38s

$ kubectl get osn my-node-01 -o wide
NAME          PHASE      OWNER                  POLICY    PROVIDER      VM FLAVOR           VM STATUS   VM IPV4        AGE
my-node-01    Running    zangrand-at-infn.it    private   CloudVeneto   cloudveneto.large   ACTIVE      10.64.22.127   7m18s


$ kubectl get nodes
NAME                          STATUS   ROLES                  AGE    VERSION
cld-k8-01.cloud.pd.infn.it    Ready    control-plane,master   232d   v1.23.5
cld-k8-02.cloud.pd.infn.it    Ready    control-plane,master   232d   v1.23.5
cld-k8-03.cloud.pd.infn.it    Ready    control-plane,master   232d   v1.23.5
my-node-01                    Ready    <none>                 4m     v1.23.5
```

Use case CMS

Data streaming and analysis at runtime (40MHz)



next steps

automatic WNs provisioning (serverless)

finalize tests

documentation

Thank you