



Student-as-a-Service

Exploiting the Kubernetes functionalities for the Student-as-a-Service use case

Context and scenario



Intro

Imagine an IT office where all the employees has to deal with proper accounts in some PCs, personal certificates, authentication keys, VPN accounts and many more...

The IT administrator has to add to his tasks also the management of all this objects and he must ensure that all the personal stuff is always synchronized (e.g.: if an employee has been fired, all his accesses must be deleted).

Automazing this task means lighten work load and increase security.



Where this can be applied?

Lab9 and networking courses students!

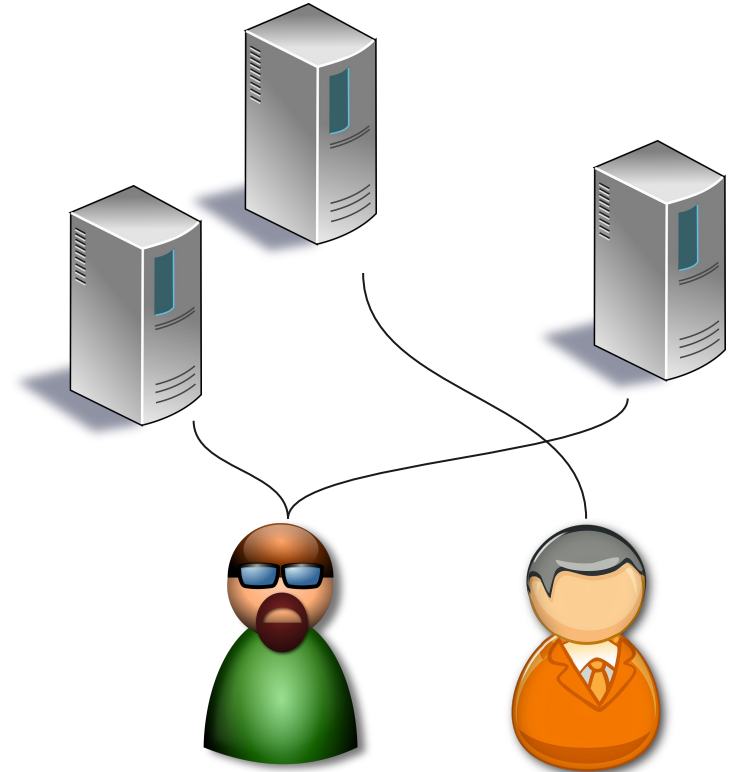
Lab9 users such as PhD students, professors, temporary users represents the perfect environment in which this process can be automatized.



Lab9

Lab9 is an heterogeneous environment. Lots of technologies cohesist, as well as many machines.

But not all the users have the privileges to access all the machines.



Problem

How can we automate the task of setting accounts and certificates handling?



Used approach: Kubernetes

Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management.

Many of the traditional sysadmin tasks like upgrading servers, installing security patches, configuring networks, and running backups are less of a concern in the cloud native world. Kubernetes can automate these things for you so that your team can concentrate on doing its core work.

‘Kubernetes does the things that the very best system administrator would do: automation, failover, centralized logging, monitoring.’

-Kelsey Hightower, developer at Google



What can Kubernetes do

- ***Service discovery and load balancing:*** If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- ***Storage orchestration:*** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- ***Automated rollouts and rollbacks:*** you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- ***Self-healing:*** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- ***Secret and configuration management:*** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration



What are Kubernetes objects

Kubernetes Objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster. Specifically, they can describe:

- What **containerized applications** are running (and on which nodes)
- The **resources** available to those applications
- The **policies** around how those applications behave, such as restart policies, upgrades, and fault-tolerance



What are Kubernetes objects

- **Pod:** A Pod is a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers. Containers within a Pod share an IP address and port space, and can find each other via localhost.
- **ReplicaSet:** *A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.*
- **Deployment:** A Deployment provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate.
- **ConfigMap:** The ConfigMap is the primary object for storing configuration data in Kubernetes. You can think of it as being a named set of key-value pairs that stores configuration data. Once you have a ConfigMap, you can supply that data to an application either by creating a file in the Pod, or by injecting it into the Pod's environment.
- *many more...*



Kubernetes API

The **REST API** is the fundamental fabric of Kubernetes. All operations and communications between components, and external user commands are REST API calls that the API Server handles. Consequently, everything in the Kubernetes platform is treated as an API object and has a corresponding entry in the API.

A **resource** is an endpoint in the Kubernetes API that stores a collection of API objects of a certain kind. For example, the built-in pods resource contains a collection of Pod objects.



Not only standard objects: Custom Resources

A **custom resource** is an extension of the Kubernetes API that is not necessarily available in a default Kubernetes installation. It represents a customization of a particular Kubernetes installation. However, many core Kubernetes functions are now built using custom resources, making Kubernetes more modular.

Kubernetes provides two ways to add custom resources to your cluster:

- **CRD (Custom Resource Definition)**
- **API Aggregation**

I used the CRD to create the custom resource ***StudentAPI***. The CustomResourceDefinition API resource allows you to define custom resources. Defining a CRD object creates a new custom resource with a name and schema that you specify.



StudentAPI CRD

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: studentapis.netgroup.com
spec:
  group: netgroup.com
  names:
    kind: StudentAPI
    listKind: StudentAPIList
    plural: studentapis
    singular: studentapi
  scope: Namespaced
  subresources:
    status: {}
  validation:
    openAPIV3Schema:
      description: StudentAPI is the Schema for the studentapis API
      properties:
        apiVersion:
          description: 'APIVersion defines the versioned schema of this re
            of an object. Servers should convert recognized schemas to the
            internal value, and may reject unrecognized values. More info:
            type: string
        info:
          description: StudentAPIInfo defines the general info of the user
          properties:
            email:
              description: student email
              minLength: 1
              type: string
            group:
              description: student group
              minLength: 1
              type: string
            id:
              description: student id
              minLength: 1
              type: string
            name:
              description: student name
              minLength: 1
              type: string
            publicKey:
              description: student PublicKey
```



StudentAPI CR

```
apiVersion: netgroup.com/v1
kind: StudentAPI
metadata:
  name: davidecota
info:
  # Add fields here
  id: s263084
  name: davide
  surname: cota
  email: s263084@studenti.polito.it
  group: group0
  roles:
    - student
  publicKey: ssh-rsa
  AAAAB3NzaC1yc2EAAAADAQABAAQCF0qyR7dUgkgRDwy5Mh0MTCqiCrcEJUWHNJmd90zWfRCKR3f3BmM+zC+rb7dFvon/
  AbTZcuPz10JxyX0/mwJiJ7PSEM+FlW9T8knFkCJs14zbRkVTpkUwrIMHhN/Ev6z/
  4aJk5YrZwxuJ0KaKQejWH00bZUAkF6mZA+1Wa53s8H640Y3k8B5SnwXRsr3LZV8KnoFq+mRdtSdMC9M2ozjQAbx5UOCiBBQ7
  ikhvWsb10+dn6xm+Dm9U+0NZMRsRpNsrM+FmN+lwGZR6d80f+PnoEseURa0fhTVHmJ7kTFAJxu9a1rC8EE0IOevK+IQxV0Rd
  +lP davide@davpc
```



How to handle different privileges?

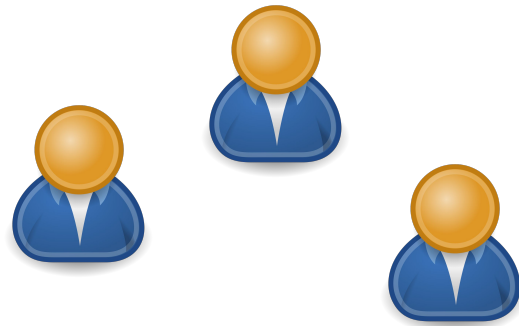
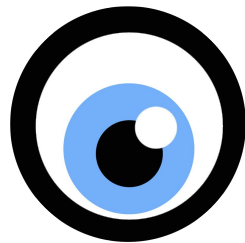
Each Custom Resource has a field 'Role' in which it is specified what kind of permissions that student can have. The same field is present in the Server ConfigMap that will be explained later.



Custom Resources comes with Custom Controllers

On their own, custom resources simply let you store and retrieve structured data. When you combine a custom resource with a custom controller, custom resources provide a true declarative API.

A declarative API allows you to declare or specify the desired state of your resource and tries to keep the current state of Kubernetes objects in sync with the desired state. The controller interprets the structured data as a record of the user's desired state, and continually maintains this state.





Beyond Controllers: Operator

Custom resources, together with a Controller acting on these resources, form the **Operator** pattern. This quote by Jimmy Zelinskie probably describes the characteristics of Operators best:

'An operator is a Kubernetes controller that understands two domains: Kubernetes and something else. By combining knowledge of both areas, it can automate tasks that usually require a human operator that understands both domains'

What I have done



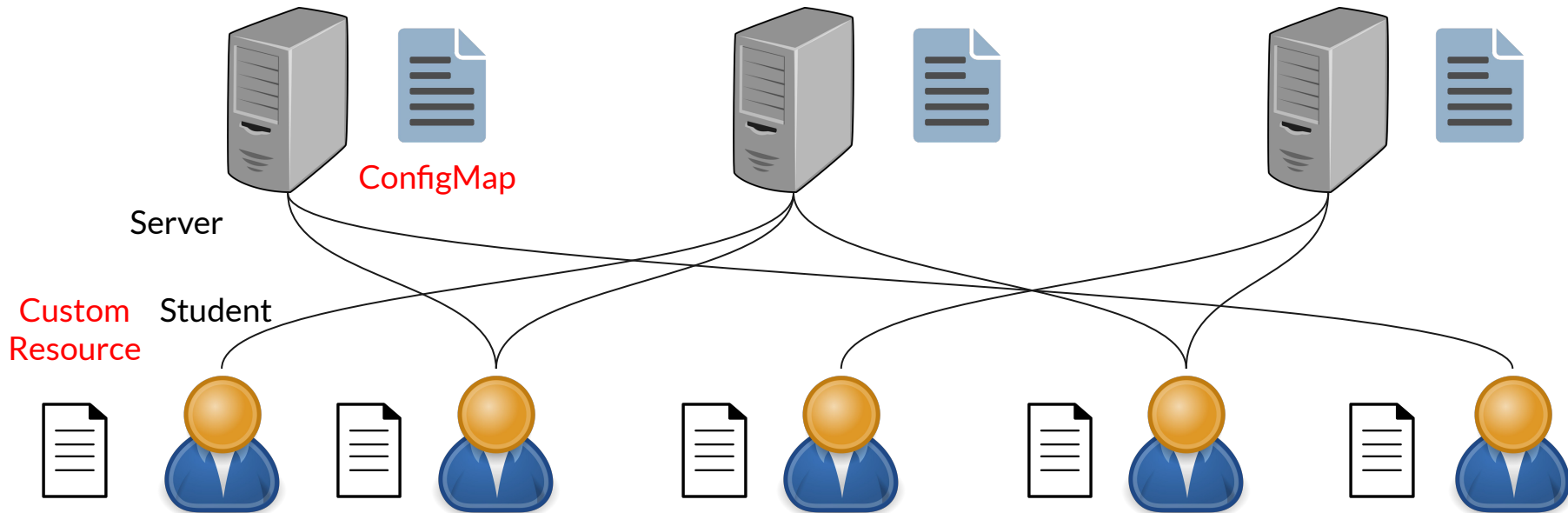
Operator-SDK

The Operator Framework provides extensive support for developing Golang-based operators. One of these is the ***Operator SDK*** that provides a high-level API for accessing a Kubernetes cluster and a scaffolding to start up an operator project.

The SDK provides workflows to develop operators in Go:

1. Create a new operator project using the SDK Command Line Interface(CLI)
2. Define new resource APIs by adding Custom Resource Definitions(CRD)
3. Define Controllers to watch and reconcile resources
4. Write the reconciling logic for your Controller using the SDK and controller-runtime APIs
5. Use the SDK CLI to build and generate the operator deployment manifests

Resources being watched by StudentOperator





Server ConfigMap

```
apiVersion: v1
items:
- apiVersion: v1
  data:
    config: |
      remote-user: davide
      remote-addr: 10.244.2.36
      remote-port: 22
      roles:
        - phD
        - student
  kind: ConfigMap
  metadata:
    labels:
      use: StudentAPI
    name: my-server
    namespace: dcota-ns1
  kind: List
```



StudentOperator Controllers

StudentOperator comes with 4 controllers that reacts, respectively, to:

1. Student creation
2. Student deletion
3. Server creation
4. Server deletion

N.B.: controller is not polling!



Student creation

The corresponding controller, when a new student CR has been created, performs the following actions:

1. sets a *finalizer*
2. retrieve all *Server ConfigMaps*
3. looks for matching 'Role' field
4. if at least one role corresponds, the user can have access to the machine; an account is created in the remote Server and the provided Public key is copied in *authorizedkeys*. He can already login via ssh.
5. '*Spec*' and '*Stat*' field are updated with the given Server address
6. An *ovpn* certificate is created for that student, then it is created the corresponding **Secret** object
7. An email notifies the student that his account is ready.



Finalizers, Spec and Stat

```
Finalizers:  
  finalizers/student
```

```
Spec:  
  Servers:  
    10.244.2.35  
Stat:  
  Servers:  
    10.244.2.35
```




Student deletion

The corresponding controller, when a student CR has been deleted, performs the following actions:

1. retrieve all *Server ConfigMaps*
2. looks for matching 'Role' field
3. if at least one role corresponds, delete his account from the server
4. update '*Spec*' and '*Stat*'
5. deletes the corresponding *ovpn Secret*
6. only when all external resources has been cleared, remove *finalizer* and delete the *Custom Resource*



Server ConfigMap creation

The corresponding controller, when a new student ConfigMap (with a given label) has been created, performs the following actions:

1. adds the ***finalizer*** to the ConfigMap
2. lists all the student *Custom Resources*
3. if role matches, create an account for the student
4. '*Spec*' and '*Stat*' are updated




Server ConfigMap deletion

The corresponding controller, when a Server ConfigMap has been deleted, performs the following actions:

1. retrieve all *students Custom Resources*
2. looks for matching 'Role' field
3. if at least one role corresponds, delete his account from the server
4. update 'Spec' and 'Stat'
5. only when all external resources has been cleared, remove *finalizer* and delete the *ConfigMap*


What has been challenging



Usually controllers reconcile built-in Kubernetes objects for standard Kubernetes operations, e.g. a controller that watches the number of Replicas of a Pod and fixes it when a node goes down. StudentOperator works in an external domain, performing actions very far away from Kubernetes standard ones such as SSH connections, VPN certificates and sending emails.

StudentOperator, furthermore, reacts only to specific events, dealing with each case with a proper reconcile logic.

Future works

- 
- not *Server ConfigMap* but *Server CRD* (adds validation and standardize the fields)
 - hardware in lab and network equipment could be modeled with a CRD
 - create a Dashboard to overcome the CLI
 - *Ansible* Operator
 - student can be added to Slack groups



Github repo

- <https://github.com/netgroup-polito/StudentOperator>