Managing Kubernetes Clusters in a multicloud environment

Sep 2021

AUTHOR(S):
Daniel Vera Nieto

Universidad Politécnica de Madrid

SUPERVISOR:
Antonio Nappi

CERN openlab

Acknowledgments

I would like to express my deep gratitude to my supervisor Antonio Nappi for his guidance and encouragement throughout this project, without him this would have not been the great learning experience that it has been. Special thanks to Artur Wiecek and the entire IT-DB-DAR team, who have been really supportive during the project; as well as the entire Openlab Summer programme organization. I'd also like to mention the MALT-auth team, especially Mary, who patiently helped me a lot to understand the authentication protocols used at CERN.

ABSTRACT

Nowadays production applications of the Database Applications and Reporting services section (IT-DB-DAR) are running in Kubernetes, taking advantage of the benefits of this technology. The portability of the new infrastructure allows running applications in a Hybrid Cloud environment for Disaster Recovery purposes, but also adds extra complexity to managing clusters in different cloud providers. To this aim, this project explored the use of Rancher to monitor the resource usage of Kubernetes clusters deployed on different clouds and also to provide read-only access to any user using them. The integration with CERN's authentication service has been the main challenge of the project, having to extend Rancher functionality to be able to use it. Additionally, a basic access control schema using local users was developed using Helm charts. Further work is required in order to effectively implement the extended functionalities in the official Rancher version.

TABLE OF CONTENTS

1. INTRODUCTION

During the last few years, CERN Database Applications and Reporting services section (IT-DB-DAR) has been migrating the applications servers to Kubernetes. Kubernetes [1] is an open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. Nowadays production applications are running in Kubernetes, benefiting from the high availability and scaling capabilities of this technology and making it possible to use a declarative continuous delivery approach. The portability of the new infrastructure also allowed the team to run applications in a Hybrid Cloud environment, using Oracle Cloud Infrastructure for Disaster Recovery purposes. But of course, new challenges appeared: how can we manage and monitor Kubernetes clusters running in different places using just a tool?

To solve this problem, the aim of the project is to investigate Rancher, a complete container management platform for Kubernetes, as a useful tool to manage Kubernetes Cluster deployed on-premise and on Oracle Cloud Infrastructure (OCI). The main use cases we investigated are

1. having a way to easily and quickly get an overall idea of the resources used by the clusters
2. providing read-only access to the teams using the applications running in the clusters.

The main objectives of the project are:

- deploy Rancher in Kubernetes
- configure Rancher to manage K8s cluster in OpenStack and Oracle Cloud Infrastructure
- deploy a simple application on-premise and on the Cloud
- setup basic access rules to control login to the Rancher dashboard
- Document challenges and experience with the tool and the two Platform (OpenStack and OCI)

The access control integration with the current CERN Single Sign-On authentication service is the main challenge encountered during the project. In short, we wanted to provide read-only access to users only to specific clusters based on the user's role at CERN, or at least, based on the user's group membership. However, this has not been possible with the current Rancher version (v2.6). Reasons for this are further explained in Section 4.3.4 where we describe the steps followed to find out the root of the problem. As this feature was critical for the intended use cases, Rancher did not fit our necessities for the moment.

This report is structured as follows. First, we describe Rancher and its main architectural components (Section 2). Then, we describe the deployment process in Section 3. In Section 4, we focus on the access control configuration, describing the different approaches followed and the difficulties we encountered. The use of Rancher for our use case is described in Section 5. We conclude this work by highlighting the main outcomes and future work of the project (Section 6).

2. Rancher

Rancher [2] is a complete container management platform for Kubernetes. It allows managing on-prem clusters and those hosted on cloud services like EKS, OCI & GKE from a single user interface. One of its main advantages is **centralizing authentication and role-based access control** (RBAC) for all of the clusters, giving global admins the ability to control cluster access from one location. Control access can be implemented by connecting Rancher to your internal identity provider like Active Directory, Keycloak, or Okta.

Rancher has a **server-agents architecture**, where the majority of Rancher operations are run on the server and the agents connect to the Kubernetes API of the downstream cluster to communicate with the server. All information is persisted in the Rancher Server data store based on etcd, the consistent and highly-available key-value store used as Kubernetes' backing store for all cluster data.

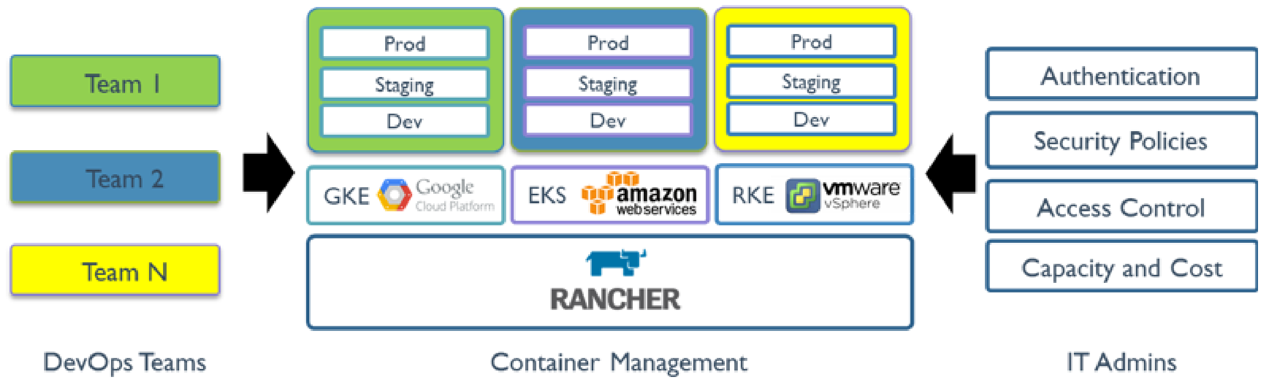


Figure 1: Rancher architecture overview. Source.

2.1. Benefits & Drawbacks

Rancher fulfills the requirements for our intended use case, where we want to have an easy way to have a glance at the **current state of all clusters** of the team and provide **read-only access to all users**. Rancher makes possible multicluster management from multiple providers (public or on-prem), so we could manage clusters from both Openstack and Oracle Cloud Infrastructure within Rancher. Additionally, we can control access to the different clusters with the centralized user authentication and authorization capabilities of Rancher, which we can connect to CERN's identity provider based on Keycloak. There are additional **benefits** of using Rancher such as:

- Cluster provisioning. We are not interested in this specific Rancher's feature for our intended use cases, since cluster provisioning is already done using Openstack Magnum.
- An easy-to-use User Interface, which reduces the technical knowledge required to manage Kubernetes clusters.
- Create any resource in Rancher in a declarative way using YAML templates, allowing us to follow GitOps practices.

However, we have faced some **drawbacks** during the development of the project. The debug logs and the documentation are not really informative sometimes, which has made it difficult to debug the problems encountered along with the project. Although the most important issue is the Kubernetes resources left behind after uninstalling Rancher from a cluster or in the imported clusters once the Rancher server is removed. This problem is further described in Section 3.2. Moreover, during this project, we have been working with the development version of Rancher v2.6, hitting some bugs along the way. Rancher v2.6 has been recently released in September 2021, but it still has some issues that hopefully will be addressed in future versions.

2.2. Initial questions

At the beginning of the project, various initial questions were investigated in order to primarily assess the suitability of Rancher for our use cases.

- **Can we run in High Available mode?**

  In a high-availability installation, a load balancer serves as the single point of contact for clients, distributing network traffic across multiple servers in the cluster and helping to prevent anyone server from becoming a point of failure. It is the **recommended way of deploying Rancher** and they provide a guide describing how to build a highly available Kubernetes cluster.

- **Do we need to persist data somehow?**

  Rancher uses the Kubernetes etcd store system to persist the information, so all Rancher server resources are saved as Kubernetes objects in the cluster where Rancher is deployed. Most objects in the Rancher API map to Kubernetes objects. While Kubernetes comes with some of these objects out of the box, others were added by creating CRDs [3]. **CRDs** (custom resource definitions) are a means of extending the Kubernetes API by creating custom objects. This allows the information to be always accessible from the pods even when they restart. This is also an important advantage of Rancher, since it allows to follow a **GitOps** approach to deploy and manage Rancher.

- **How upgrades are managed?**
  - Rancher server upgrades:

    For installations using Helm charts (preferred), it is enough to update Rancher's Helm repository to fetch the latest (or specific) version and upgrade the release passing the same values used in the initial installation. It is also possible to delete the Rancher release and install it from scratch. Both options are described in the documentation.

- **Can we import existing clusters to be managed by Rancher?**

  Yes, Rancher supports a number of cloud providers. We have tested the importing process with Openstack clusters and Oracle cloud clusters

- **Can we set up permissions based on CERN roles?**

  Quick response: No.

  This has been one of the pain points of the project. CERN roles [4] are identifiers for a set of permissions in your application (in this case, Rancher), that can be assigned to a group of users.

> The authentication token provided to users when they log in to Rancher will contain all the roles that the users have in your application only, in a special claim named *cern_roles* for OIDC or *CernRoles* for SAML. The concept of roles was introduced to let applications enforce their own authorization scheme without querying for groups membership. The reasons for this and an explanation of how to set up a roles-based permissions scheme for your application are described in CERN's internal authentication documentation.

However, Rancher only supports authorization based on groups. Although our Rancher application was granted the permissions to get the groups from CERN's identity provider, due to how Rancher is implemented it has not been possible to get access to them. This problem is further described in Section 4.3.4.

3. Rancher Deployment

3.1. Installing Rancher

Rancher should work with any modern Linux distribution and with any modern Docker version. There are additional hardware requirements depending on the size of the clusters Rancher is going to manage. All of them are described in the documentation.

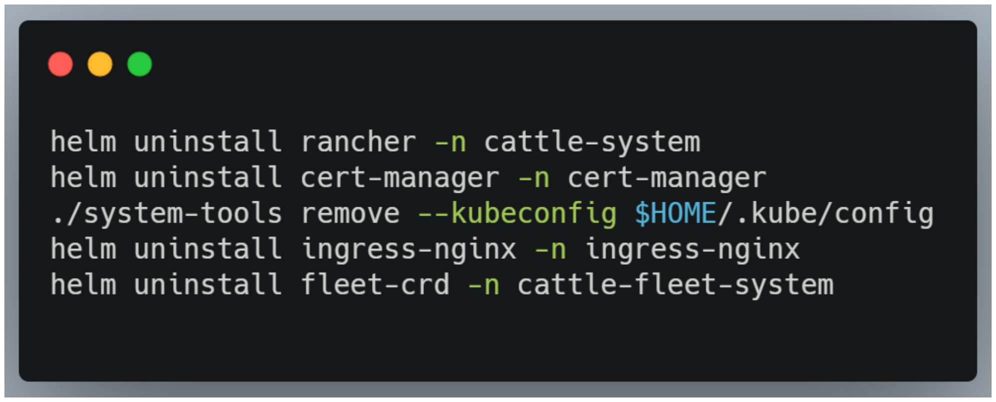
There are different ways of installing Rancher, during this project we have used:

- **Installation on a single node with Docker**, recommended only for development and testing purposes. It is useful to get familiar with Rancher and approach later a production deployment.
- **Installation on a Kubernetes cluster using Helm charts**. We also installed an ingress controller using the ingress-nginx Helm chart, which triggered the creation of a load balancer in the cloud provider where we could set the hostname for the Rancher server. It is also possible to use a specific image setting the *rancherImageTag* option of the Helm chart (docs). This has been useful to work with the Rancher version in development, since it is possible to install any Rancher commit selecting the corresponding tag at DockerHub. A **script** has been developed to automate the installation process and it is available at the project's Gitlab repo.

3.2. Uninstalling Rancher

Complete cleanup of a cluster after uninstalling Rancher has proven to be a difficult task. The first issue we experienced is an error that happens when we tried to reinstall Rancher in a cluster after uninstalling it previously. The installation would fail with an internal **error related to a webhook** call. After a while, we found out this was an active Rancher issue (issue#33988) and that the workaround to solve the problem until the Rancher team fixed the issue was simply to manually delete the problematic webhook.

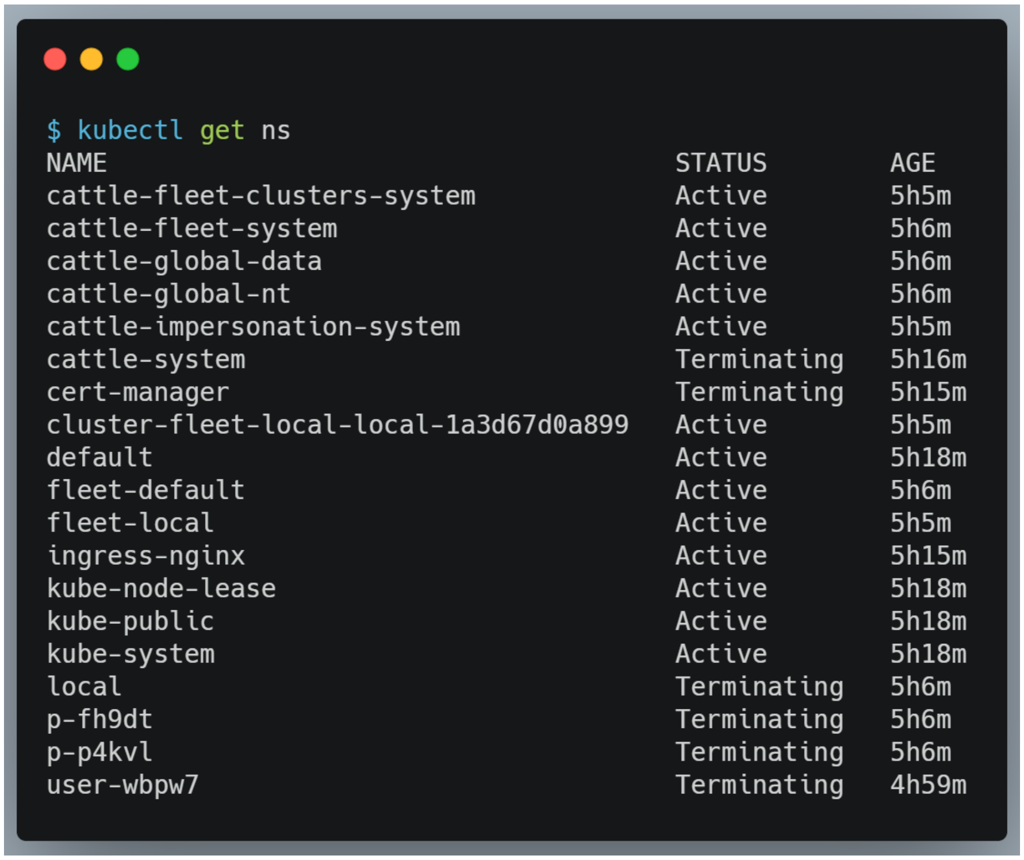
This was not the only problem related to resources left behind after uninstalling Rancher. We uninstalled the helm releases installed by Rancher and also used Rancher system tools to remove all the resources as explained in the documentation (see used commands in Figure 2).



```
helm uninstall rancher -n cattle-system
helm uninstall cert-manager -n cert-manager
./system-tools remove --kubeconfig $HOME/.kube/config
helm uninstall ingress-nginx -n ingress-nginx
helm uninstall fleet-crd -n cattle-fleet-system
```

Figure 2. Commands to uninstall Rancher and all related resources.

However, **not all resources were removed** and some of them got stuck in the *Terminating* status. According to issues submitted by other users (issue#34570) this is a general problem. The only solution we have found is to manually delete the finalizers of the Kubernetes objects in the terminating status. This is a tedious process due to the large number of resources to manually edit (see Figure 3).



```
$ kubectl get ns
NAME                                      STATUS        AGE
cattle-fleet-clusters-system              Active        5h5m
cattle-fleet-system                       Active        5h6m
cattle-global-data                        Active        5h6m
cattle-global-nt                          Active        5h6m
cattle-impersonation-system               Active        5h5m
cattle-system                             Terminating   5h16m
cert-manager                              Terminating   5h15m
cluster-fleet-local-local-1a3d67d0a899    Active        5h5m
default                                   Active        5h18m
fleet-default                             Active        5h6m
fleet-local                               Active        5h5m
ingress-nginx                             Active        5h15m
kube-node-lease                           Active        5h18m
kube-public                               Active        5h18m
kube-system                               Active        5h18m
local                                     Terminating   5h6m
p-fh9dt                                   Terminating   5h6m
p-p4kvl                                   Terminating   5h6m
user-wbpw7                                Terminating   4h59m
```

Figure 3. Namespaces status after running uninstalling commands.

4. Access Control in Rancher

One of the main use cases for us was to provide users with a way to access their clusters in read-only mode. This section describes how roles and permissions work in Rancher, and how to set up the authentication and authorization system following different methods.

4.1. Permissions

There are three different types of permissions depending on their scope: **global permissions, cluster roles and project roles**. You can assign any of those permissions/roles to single users or groups.

- Global Permissions define user authorization **outside the scope of any particular cluster**. Out-of-the-box, there are four default global permissions: Administrator, Restricted Admin, Standard User, and User-base.
- Cluster and Project Roles define user authorization inside a cluster or project. **Cluster roles** are roles that you can assign to users, granting them access to a cluster. There are two primary cluster roles: Owner and Member. **Project roles** are roles that can be used to grant users access to a project. There are three primary project roles: Owner, Member, and Read Only.

It is also possible to create custom roles at any level, setting fine-grained permissions for each of the resources within Rancher. The default role assigned to each user can also be customized. **Roles are defined as Custom Resource Definitions (CRD)**. Thus, they can be created using YAML templates. Assigning a role to a specific user can be also done using YAML templates, creating a ***ClusterRoleBinding*** resource. Those are the resources that define the roles assigned to specific users or groups. You can find the YAML files that created the current roles and roles bindings at the left *navbar* > *local cluster* > *More resources>RBAC>ClusterRoles/ClusterRoleBindings*. The Global Roles can be found at *local cluster> More resources* > *rancher* > *GlobalRoleBindings*.

4.2. Authentication with CERN SSO

The preferred authentication method is using the **Single Sign-On service**, so any CERN member can log in to Rancher. Rancher has a number of connectors to different identity providers. In our case, as the SSO is based on Keycloak, we tested both SAML and OIDC Keycloak connectors. To this aim, we followed CERN Authentication documentation to configure the application. The **main challenges we faced in this part of the project are related to the integration with CERN SSO**.

In the beginning, we used the SAML authentication method but due to its complexity and the problem we had to enforce authorization, we decided to move to the OIDC authentication protocol. Although it simplifies the authentication configuration, we still were not able to enforce authorization. After testing different hypotheses of what could be the root of the problem, we found out that it is not possible to integrate CERN SSO with Rancher. The first solution was to adapt Rancher to be able to integrate it but resulted problematic, so we decided to set up a local authentication schema using Helm charts.

4.2.1. SAML

Security Assertion Markup Language (**SAML**)[5] is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. SAML is an XML-based markup language. We based our configuration on Rancher documentation, although it assumes you have direct access to the Keycloak server (unfortunately, we do not have it).

The steps followed to set up authentication using this method were:

1. Register a new application at CERN's Application Portal.
2. Enable SSO authentication in the created application exchanging Rancher's SAML metadata file.
3. Configure Keycloak authentication in Rancher. First, you have to map the SAML claim names to the required fields so Rancher is able to get the user's attributes such as *Display Name* or *Groups*. In this case, we mapped the *cern_roles* claim to the *Groups* field. Then you have to provide the SSO certificate and a private key.

Once we have connected to CERN SSO, users were able to log in to Rancher using their CERN credentials. However, we were able to get neither the roles of a user nor additional information like the Display Name. Thus, our first hypothesis was that the attribute names mapping it's not working, as Rancher only sees the *EmailAddress* (used as user identifier). Moreover, we found in the documentation this statement: "SAML Protocol does not support search or lookup for users or groups. Therefore, there is no validation on users or groups when adding them to Rancher." After trying to debug the problem with no success, **we decided to use the OIDC method** that was being developed at that time.

4.2.2. OIDC

OpenID Connect [6] is a simple identity layer on top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner. We also decided to use this method due to its simplicity compared to SAML. The steps we followed to set up this authentication method in Rancher are:

1. **Install a master-head version**. As OIDC authentication was being implemented at that time, we needed to use one of the latest versions in Rancher repository. We selected the version we wanted with the *rancherImageTag* option of the Rancher installation Helm chart.
2. **Register a new application** at CERN's Application Portal to generate a ClienID and Client Secret.
3. Select the **Keycloak OIDC authentication** provider and fill the configuration form with the client ID, client secret, CERN SSO certificate, authentication provider endpoint and realm. Here he hit a bug in Rancher (issue#3406) that was fixed later.

Some of the errors we encountered trying to configure the authentication provider are:

- *Invalid redirect_uri* error when I set the redirect url at the OIDC configuration at the Application Portal to *https://[HOST NAME].cern.ch/* instead of *https://[HOST NAME].cern.ch/** (you need to specify the valid redirect urls, the * wildcard allows any from that host).
- *401 invalid client secret* error if the client secret is invalid.

- *certificate signed by unknown authority*: when trying to set up OIDC authentication with a private Keycloak server, see issue description and solution in Section 4.3.3.

We still had the same problem we found with the SAML authentication method: we could not get access to the CERN roles or groups from Rancher.

4.3. Authorization with CERN SSO

As we continued not getting access to CERN groups from Rancher and were not able to find the root of the problem, we decided to deploy our own Keycloak [7] server, which hopefully would help us clarify where could be the problem.

4.3.1. Private Keycloak Server Authentication

SAML

As we suspected there may be a problem with the SAML attributes mapping in the Application Portal, we used deployed our own Keycloak server following the docs:

- To setup a new realm, users, groups & stuff: Keycloak documentation.
- Rancher authentication with Keycloak: Rancher documentation.

OIDC

We also tried to connect our private Keycloak server to Rancher using the OIDC method. However, the new Rancher version enforces hard requirements for this configuration related to the trusted certificates, getting a *Certificate signed by unknown authority* error when using the self-signed certificate from our Keycloak server. To debug this problem with the certificates, the solution was to generate a CERN certificate for the Keycloak server and add the CERN Certificates Authority root certificate to Rancher. We followed the documentation for adding additional certificates to Rancher.

So far so good, it works! You can authenticate with the users we created at Keycloak and we also have access to the groups these users are members of with both SAML and OIDC methods. So **we concluded that the problems likely come from the integration with CERN SSO** and not Keycloak itself.

4.3.2. The Groups problem

Ideally, **we would like to assign permissions in Rancher and the clusters managed by Rancher based on CERN roles**. The main (huge) issue during the whole project has been the impossibility to access CERN roles or user groups from Rancher.

What we tried to debug the problem:

- **Setting up a private Keycloak server.** A headache to configure the certificates for the server and to make Rancher accept them, but finally accomplished. We created users and groups and voilà, they are accessible from Rancher! So it seems it's not a problem of the application itself.
- **Inspect the OIDC token** sent by SSO or the private Keycloak server. The best way possible to inspect the OIDC tokens was to launch one of the example SSO applications. The only one that showed the *cern_roles* (and *groups* if activated) was the go example app. Result: the groups information was there.
- So if the info was in the token, let's try to **translate *cern_roles* to *groups*** so Rancher can read them using Dex. Dex [8] acts as a portal to other identity providers through "connectors",

working as a middleware between the identity provider and the application. Unfortunately, Rancher is still not able to get the groups.

So, why groups do not work in Rancher? The SSO was sending the groups claim in the token but Rancher could not read it. Why? The root of the problem is the actual way Rancher works. It does not get the groups from the OIDC token but queries the identity provider (SSO in our case) to get users and groups. From the documentation:

"When adding a user or group to a resource, you can search for users or groups by beginning to type their name. The **Rancher server will query the authentication provider to find users and groups** that match what you've entered. Searching is limited to the authentication provider that you are currently logged in with."

But then, why Rancher cannot query CERN SSO? The answer to this is a bit tricky. It worked for the private Keycloak server, but not for CERN SSO because **Rancher queries the default Keycloak server API**. Basically, Rancher code flow to request users/groups to the identity provider (see source code) is:

1. It gets the URL from the Issuer parameter (this is configured at Rancher UI), that looks like this: *https://auth.cern.ch/auth/realms/cern* and, using the *getSearchURL* method, convert it to the admin endpoint *https://auth.cern.ch/auth/admin/realms/cern*
2. Then build the query URL using the method *searchURL*: https://auth.cern.ch/auth/admin/realms/cern/users?search=SEARCHTERM
3. Then it sends a get request to that url with the method *k.getFromKeyCloak(searchURL)*
4. It fails trying to get a user:

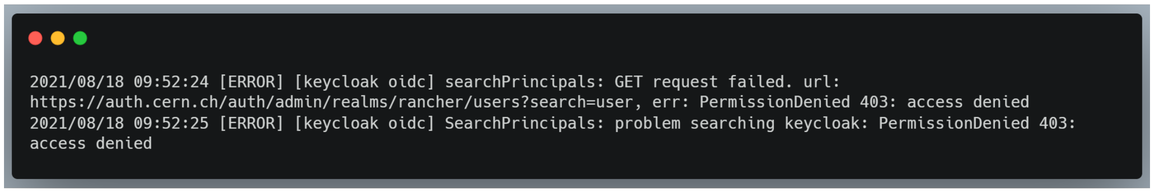

```
2021/08/18 09:52:24 [ERROR] [keycloak oidc] searchPrincipals: GET request failed. url:
https://auth.cern.ch/auth/admin/realms/rancher/users?search=user, err: PermissionDenied 403: access denied
2021/08/18 09:52:25 [ERROR] [keycloak oidc] SearchPrincipals: problem searching keycloak: PermissionDenied 403:
access denied
```

Figure 5. Permission denied error trying to query CERN's Keycloak admin API.

It fails because we cannot query that endpoint of CERN SSO, it is forbidden. To query CERN SSO you must:

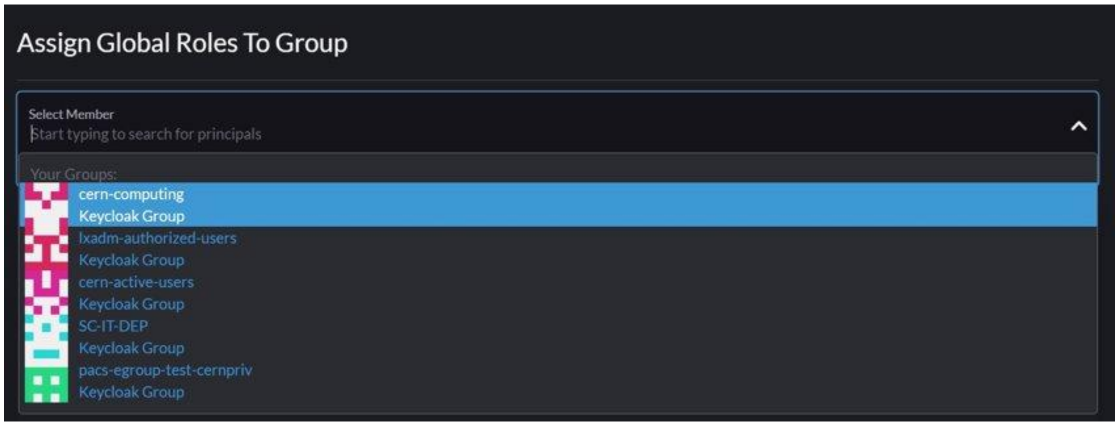
1. Request an API access token. See documentation.
2. Use it to query the right API endpoint at CERN, which is not the default one for a Keycloak server. See documentation.
3. To query for the right resources, because at CERN there are no *Users* and *Groups* resources but *Identity* and *Group* resources. See the swagger app at CERN to find all possible resources and operations using the SSO API.

In conclusion, it was not possible to get CERN groups from Rancher due to the way Rancher is implemented. This was a major issue since setting permissions based on a user's groups was one of the reasons to use Rancher.

4.3.3. Modifying Rancher source code

As the implementation of Rancher prevented us from using it for our use case, **we decided to modify the source code and adapt it to our needs**. To this aim, we set up the Rancher development environment based on the instructions at the project wiki using the Windows Subsystem for Linux. First, we developed an example application to query the SSO Authorization API in Go language based on previous examples. Once we were able to get the groups in this example application, we started integrating the different components in Rancher:

1. First, request the API access token.
2. Then use it to query the SSO authentication API and request the groups asking for the right resources.
3. To enable the groups in Rancher, it was needed to fetch them first. Originally, Rancher does this querying the OIDC UserInfo endpoint. However, in our case, the groups' information was not there but in the OIDC id token. So we needed to modify Rancher to look into the id token to fetch the groups and, thus, activate the groups feature in Rancher.
4. It finally worked! Now it would be possible to assign roles based on users' groups.


Assign Global Roles To Group

Select Member
Start typing to search for principals

Your Groups:
cern-computing
Keycloak Group
lxadm-authorized-users
Keycloak Group
cern-active-users
Keycloak Group
SC-IT-DEP
Keycloak Group
pacs-egroup-test-cernpriv
Keycloak Group

Figure 6. CERN groups are listed in Rancher.

Afterward, we worked on refactoring our implementation so it can support more general use cases that could benefit organizations with a similar problem. Our idea is to work on a Pull Request that would extend Rancher functionality.

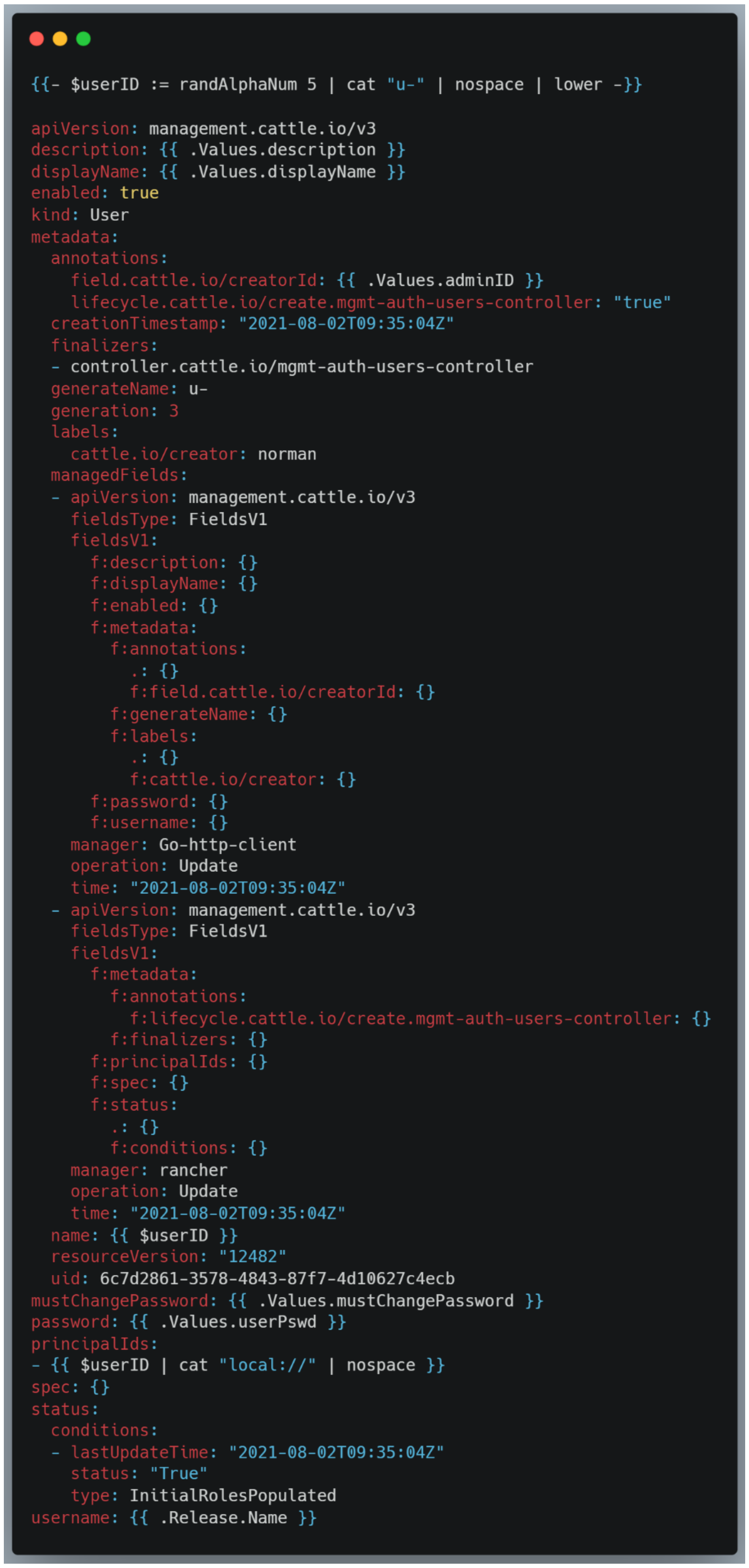
4.4. Local authentication & authorization

As the extended functionality still had problems working in certain scenarios, we implemented another option to enforce access control in Rancher using the **local authentication system**. We set up the basic authorization configuration using **Helm charts**. Helm Charts [9] help you define, install, and upgrade whole Kubernetes applications, although in this case, we used them just to create some Rancher resources. We created local users and set the corresponding permissions to have admin and read-only users for each of the teams.

Figure 4 shows the template of a **user resource definition**. The values inside double brackets {{ *.Values.specific-value* }} are processed by Helm, inserting the values defined in the *Values.yaml* file or overriding the values in that file with the *-f* flag with a specific file containing the values. Helm template function *randAlphaNum* is used to generate user IDs. The username is taken from the release name. Username and name (user ID) must be different from any other user. UIDs, resourceVersion, timestamps and similar fields are regenerated by Rancher when the template is applied, so even if you apply the same YAML file multiple times these fields will be generated. Templates are available at the Gitlab repo.

The **role binding template** is similar to the user template, and it declares which role will be assigned to a user specifying the user ID. We automated the creation of the users and the role bindings with a simple script that installed a new user release in the cluster with a role and metadata specified in values files.

Adding a user as a member to a cluster can be done during the cluster import process using Rancher's UI but it may become handy to do it also using YAML files. However, this is left for future work as first attempts to apply the cluster-member-role created an inconsistent state in Rancher where the user would have access to the cluster but won't be listed as a member of the cluster.



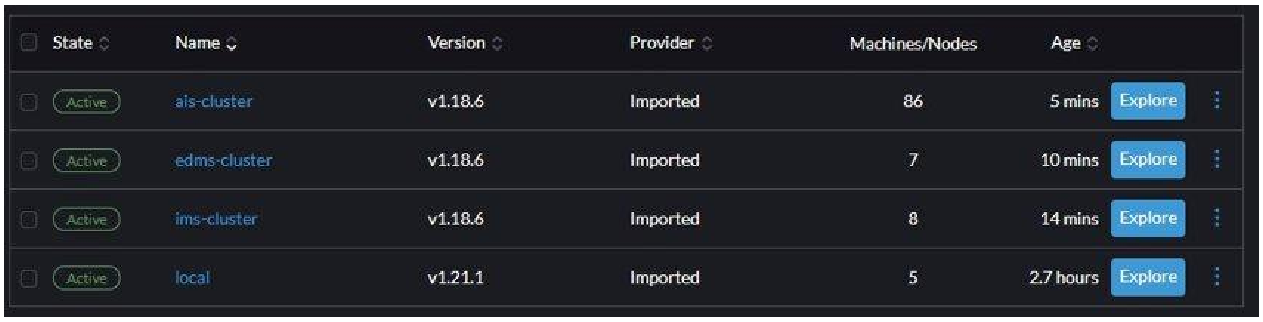
```
{{- $userID := randAlphaNum 5 | cat "u-" | nospace | lower -}}

apiVersion: management.cattle.io/v3
description: {{ .Values.description }}
displayName: {{ .Values.displayName }}
enabled: true
kind: User
metadata:
  annotations:
    field.cattle.io/creatorId: {{ .Values.adminID }}
    lifecycle.cattle.io/create.mgmt-auth-users-controller: "true"
  creationTimestamp: "2021-08-02T09:35:04Z"
  finalizers:
  - controller.cattle.io/mgmt-auth-users-controller
  generateName: u-
  generation: 3
  labels:
    cattle.io/creator: norman
  managedFields:
  - apiVersion: management.cattle.io/v3
    fieldsType: FieldsV1
    fieldsV1:
      f:description: {}
      f:displayName: {}
      f:enabled: {}
      f:metadata:
        f:annotations:
          .: {}
          f:field.cattle.io/creatorId: {}
        f:generateName: {}
        f:labels:
          .: {}
          f:cattle.io/creator: {}
      f:password: {}
      f:username: {}
    manager: Go-http-client
    operation: Update
    time: "2021-08-02T09:35:04Z"
  - apiVersion: management.cattle.io/v3
    fieldsType: FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          f:lifecycle.cattle.io/create.mgmt-auth-users-controller: {}
        f:finalizers: {}
      f:principalIds: {}
      f:spec: {}
      f:status:
        .: {}
        f:conditions: {}
    manager: rancher
    operation: Update
    time: "2021-08-02T09:35:04Z"
  name: {{ $userID }}
  resourceVersion: "12482"
  uid: 6c7d2861-3578-4843-87f7-4d10627c4ecb
mustChangePassword: {{ .Values.mustChangePassword }}
password: {{ .Values.userPswd }}
principalIds:
- {{ $userID | cat "local://" | nospace }}
spec: {}
status:
  conditions:
  - lastUpdateTime: "2021-08-02T09:35:04Z"
    status: "True"
    type: InitialRolesPopulated
username: {{ .Release.Name }}
```

Figure 4. User template used in the local authentication schema.

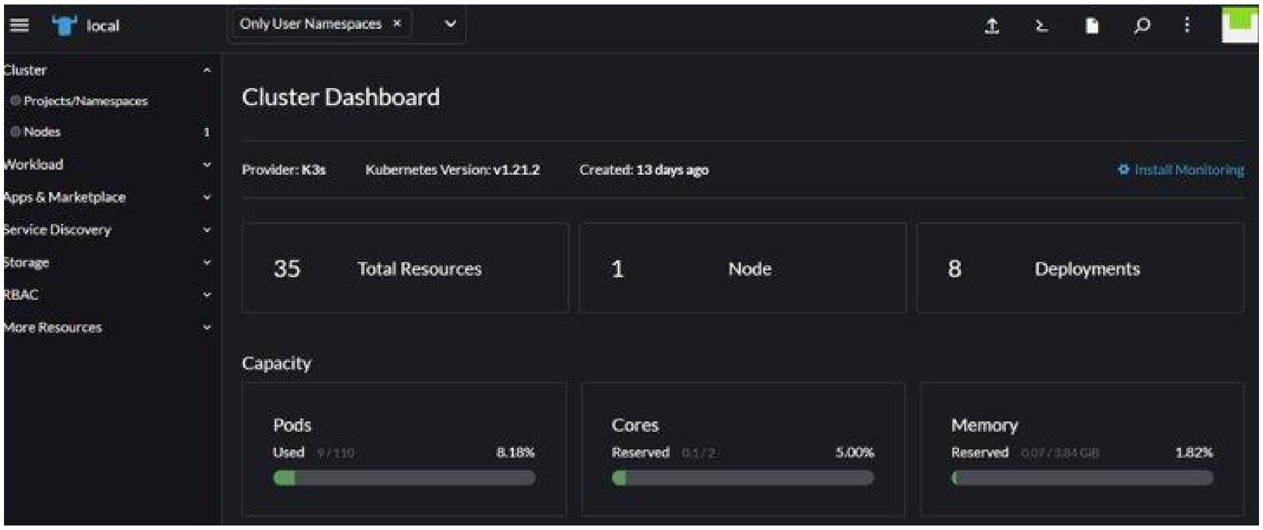
5. Using Rancher

The other main use case we wanted Rancher for was to manage Kubernetes clusters so that we can quickly get an overview of the resources used by each cluster. To this aim, we needed to import existing clusters into Rancher. This can be easily done using Rancher UI and following the documentation. Once we have registered the clusters, we will be able to list them as shown in Figure 7 and inspect the detailed resources used in each of them as shown in Figure 8.



State	Name	Version	Provider	Machines/Nodes	Age		
Active	ais-cluster	v1.18.6	Imported	86	5 mins	Explore	⋮
Active	edms-cluster	v1.18.6	Imported	7	10 mins	Explore	⋮
Active	ims-cluster	v1.18.6	Imported	8	14 mins	Explore	⋮
Active	local	v1.21.1	Imported	5	2.7 hours	Explore	⋮

Figure 7. List of imported clusters.



Cluster Dashboard

Provider: K3s   Kubernetes Version: v1.21.2   Created: 13 days ago

35 Total Resources   1 Node   8 Deployments

Capacity

Pods
Used 9 / 110 8.18%

Cores
Reserved 0.1 / 2 5.00%

Memory
Reserved 0.07 / 3.84 GiB 1.82%

Figure 8. Imported cluster dashboard.

6. Conclusions

This project explored the use of Rancher to monitor the resource usage of Kubernetes clusters and provide read-only access to any user using them. Rancher is a tool that accomplishes our use cases, although we have encountered different problems using it.

The **main challenge** we have encountered is the **integration with CERN authentication service**. It was not possible to configure OIDC authentication with CERN SSO because it does not use the default Keycloak configuration, which is what Rancher OIDC implementation expects. As this may be a problem general to many organizations, we worked on extending Rancher functionality. Additionally, as we were using the Rancher development version, we faced some bugs in the application (issue#33988, issue#34570) and we reported some of them (issue#3612, issue#33748). Another problem was related to the difficulties to reinstall Rancher in a cluster once we uninstalled it before. The resources left behind in the uninstalling process prevented us from installing it again or they saved previous information such as the past admin credentials. For this reason, we found that the easiest way to avoid this problem was to use a fresh cluster, but we should not need to launch a new cluster each time we deploy Rancher. It should be possible to use the same Kubernetes cluster as many times as we need.

The main **outcomes of the project** are:

- Investigated the suitability of Rancher for our intended use case.
- Extended Rancher functionality to be able to integrate it at CERN ecosystem.
- Basic authentication and authorization schema to create local users with specific permissions using Helm charts.
- Simple scripts to facilitate the installation of Rancher and the deployment of the basic authentication schema.
- Everything is available at the Gitlab repo.

The proposed **future work** is:

- Create a custom role template for cluster read-only access. This should be similar to the global roles templates.
- Prepare pull request to the Rancher repo, so the extended functionality could benefit the community.
- Improve the hardcoded information in the basic authentication helm charts.
- Investigate how to generate the YAML template to import a cluster, so it will not be needed to use Rancher UI and we could just apply the template in the cluster we want to import.
- Find ways to clean up the cluster completely after uninstalling Rancher.

7. References

1. "What Is Kubernetes?" *Kubernetes*, 23 July 2021, kubernetes.io/docs/concepts/overview/what-is-kubernetes.
2. "Why Rancher?" *Rancher Labs*, rancher.com/why-rancher. Accessed 24 Sept. 2021.
3. Rancher. "Rancher 2.x Development Overview · Rancher/Rancher Wiki." *GitHub*, github.com/rancher/rancher/wiki/Rancher-2.x-Development-Overview#understanding-crds. Accessed 24 Sept. 2021.
4. "Role Based Permissions (Recommended) - Authorization Service." *CERN Auth Docs*, auth.docs.cern.ch/applications/role-based-permissions. Accessed 24 Sept. 2021.
5. Wikipedia contributors. "Security Assertion Markup Language." *Wikipedia*, 11 Sept. 2021, en.wikipedia.org/wiki/Security_Assertion_Markup_Language.
6. OpenID. "OpenID Connect | OpenID." *OpenID - The Internet Identity Layer*, 6 Apr. 2021, openid.net/connect.
7. "Keycloak." Keycloak, www.keycloak.org. Accessed 24 Sept. 2021.
8. Dexidp. "GitHub - Dexidp/Dex: OpenID Connect (OIDC) Identity and OAuth 2.0 Provider with Pluggable Connectors." GitHub, github.com/dexidp/dex. Accessed 24 Sept. 2021.
9. "Helm | Helm." *Helm*, helm.sh. Accessed 24 Sept. 2021.