RISK: K8s with Azure AD integration, Access control

Contents

[1. General 1](#_Toc98951618)

[1.1. Purpose 1](#_Toc98951619)

[1.2. Scope 1](#_Toc98951620)

[2. Azure AD Integration with Kubernetes 1](#_Toc98951621)

[2.1. Azure AD App registration 1](#_Toc98951622)

[2.2. API server configuration 3](#_Toc98951623)

[3. Access Control on K8s 4](#_Toc98951624)

[3.1. Admin 5](#_Toc98951625)

[3.2. Developers 5](#_Toc98951626)

[3.3. Operations 6](#_Toc98951627)

[3.4. Users 6](#_Toc98951628)

[3.5. Authorization table 6](#_Toc98951629)

# General

This document describes how to perform the access control on Kubernetes. And also, how to integrate the Azure AD With the Kubernetes cluster.

## Purpose

How to control the access of a user in Kubernetes cluster and how we can be able to integrate the Kubernetes with the Azure Active Directory

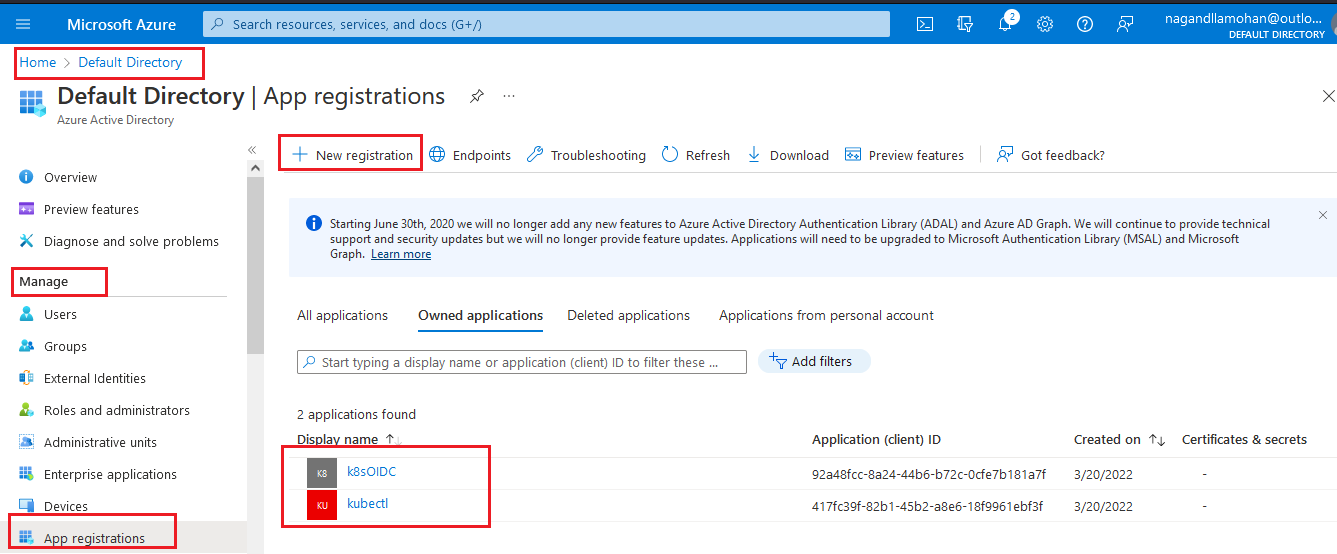
## Scope

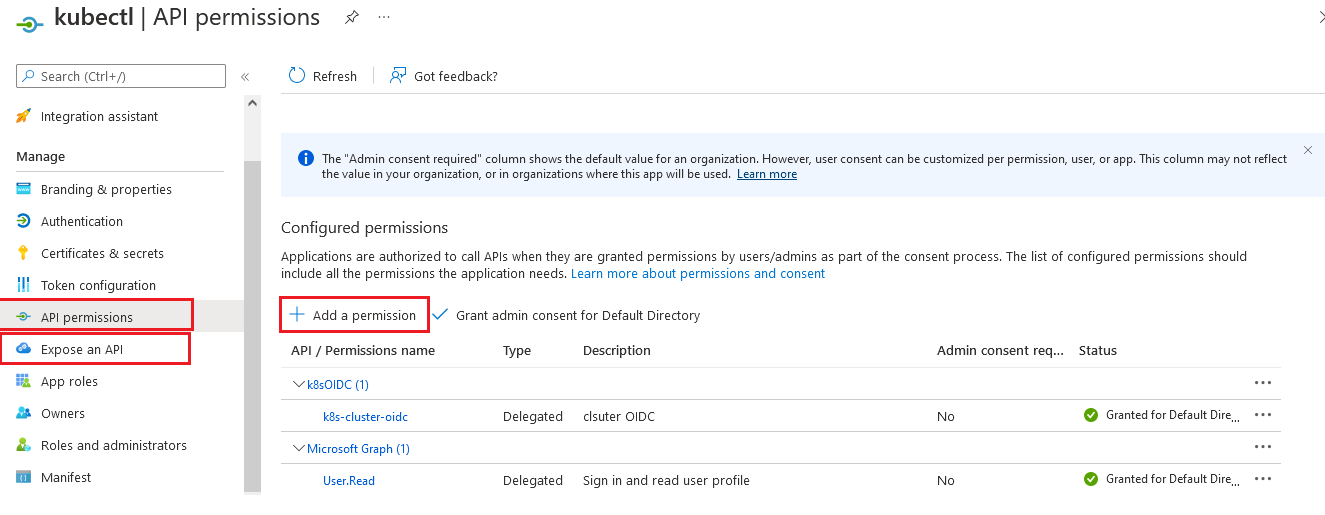
Successful Integration between Kubernetes and Azure Active Directory and access control between the user and the cluster.

# Azure AD Integration with Kubernetes

The way of integration between Azure Active Directory and Kubernetes involves 2 mechanisms one is related to Kubernetes API server configuration and the other one is related to Azure Active Directory application registrations.

## Azure AD App registration

* < how has the admin level access in Azure Portal>
* Log in to [Azure portal](https://portal.azure.com/)
* Here we need two applications, one is for the API server another one is for the kubectl component
* The navigation should be like Azure AD > App registration here we need to register the two applications one name should be like k8s-API for Kubernetes API other one is like kubectl for kubectl component authentication  
  
* During both application registrations you can ignore the redirection URL this is an optional parameter only
* once we successfully created the two applications then choose the k8s-API then choose the expose API under the manage section then please choose the set API then save the option.
* Then open the kubectl application registration under managed section authentication option is available please choose that authentication and inside the authentication at the bottom of the configuration you can find the Allow public client flows please flip that to yes if it is no.
* Again, the manage section you can find API permissions click on that one and choose to add A Permission under that section you can find APIs my organization user click on that option and you can use the exposed API off K8s-API application by searching then click on grant permission



## API server configuration

When we are trying to integrate the Azure AD for authentication to the Kubernetes cluster, we need to add a few command-line arguments to the API server static pod then the cluster will integrate to the Azure AD by using the URL provided in the arguments.

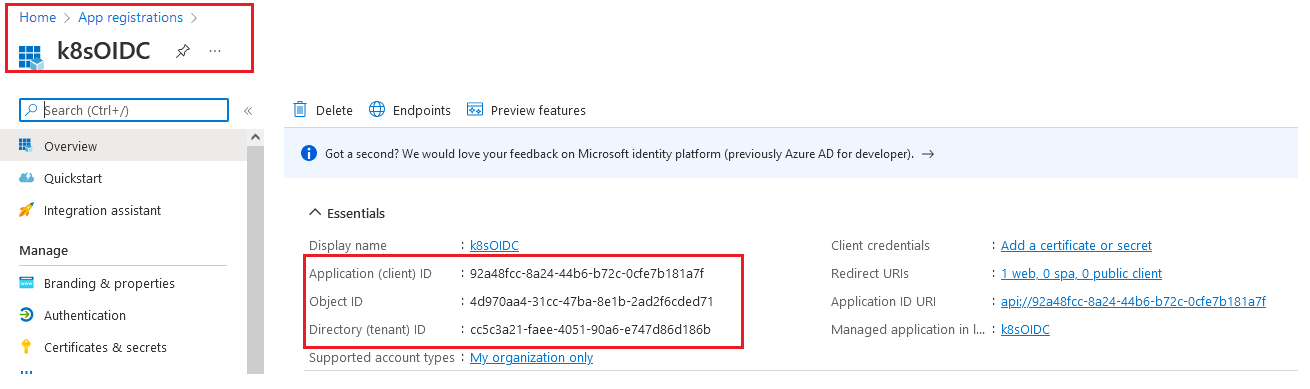
When we are going to integrate AD, first we need to log into the master node and then locate to the path /etc/kubernetes/manifests here we can find the kube-apiserver.yaml file

List of arguments which we are passed to the API static pod are

--oidc-client-id=spn:APISERVER\_APPLICATION\_ID

--oidc-issuer-url=https://sts.windows.net/TENANT\_ID/

--oidc-username-claim=sub

The APISERVER\_APPLICATION\_ID is the application ID of the K8s-API which will appear under an application under the App registration section in the portal same way you can get the data of the kubectl application also   


Once the admin did this all the things, he needs to create a Kube config file, use the below commands to generate the config file

kubectl config set-credentials "USER\_NAME" --auth-provider=azure \

--auth-provider-arg=environment=AzurePublicCloud \

--auth-provider-arg=client-id=APPLICATION\_ID \

--auth-provider-arg=tenant-id=TENANT\_ID \

--auth-provider-arg=apiserver-id=APISERVER\_APPLICATION\_ID

The USER\_NAME can be any. We will use this more in the access control part please refer to that section.

After the config generation, we will give the config file to the user he will use that config to access the cluster by using the Azure AD credentials.

If the user tries to get pod-like below

kubectl get pods

To sign in, use a web browser to open the page https://aka.ms/devicelogin and enter the code DEC7D48GA to authenticate.

The user will use that web address to confirm his identity to get the token to his config file this will automatically write the token to this config file.

# Access Control on K8s

All type of users can log into the cluster by using their Azure AD credentials. Now it's time to authorize. The authorization can be controlled by the RBAC in Kubernetes, please refer to this for details RBAC mechanism [Using RBAC Authorization | Kubernetes](https://kubernetes.io/docs/reference/access-authn-authz/rbac/#:~:text=ClusterRole%2C%20by%20contrast%2C%20is%20a%20non-namespaced%20resource.%20The,it%20can%27t%20be%20both.%20ClusterRoles%20have%20several%20uses.?msclkid=076c3715a91911ecb0cd43cd9ebecdf6)

**RBAC:**

The RBAC can be used for the authorization level for a particular user in the Kubernetes cluster. This contains cluster role and cluster role binding files. Cluster role can decide which type of privileges can the group have and what the group members can able to do with the provided access. Cluster role binding can bind the access provided in the cluster role to the group mentioned in the manifest file. You can find the example of these manifest files in the below dialog box

Cluster role

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: ClusterRole

**metadata**:

*# "namespace" omitted since ClusterRoles are not name spaced*

**name**: developer-access

**rules**:

- **apiGroups**: [""]

**resources**: [""] # can mention the list of resources

**verbs**: [""] # can mention the actions they can perform

ClusterRole binding

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: ClusterRoleBinding

**metadata**:

**name**: developer-global

**subjects**:

- **kind**: Group

**name**: developers *# Name is case sensitive*

**apiGroup**: rbac.authorization.k8s.io

**roleRef**:

**kind**: ClusterRole

**name**: developer-access

**apiGroup**: rbac.authorization.k8s.io

We can able to segregate the users based on the type of application usage and the behavior in the cluster. Admin in the Azure portal will create the below list of groups for authorization and a set of members will be assigned to that groups based on the role in Azure portal. The list of groups is

* Admin group
* Developers group
* Operation group
* Users group

Beyond the above-mentioned groups, there is one static cluster-admin who doesn’t have any limitations in the cluster. There is a significant file that can decide the cluster-admin role in the cluster that is Kube config. The cluster-admin has a unique Kube config file which is in the master node at /etc/kubernetes/ named as admin. conf and not all users are allowed to get the file, there are again some privileges at VM level. For that, please refer to section 2.3 VM in access management of Fenice assets document. The same cluster-admin config file will be stored in the Gerrit repo also. This static cluster-admin will generate all types of RBAC for several groups. The initial admin group allocations can be done from the master node because this is no admin members from AD till at.

## Admin

The initial admin RBAC can be deployed from the master node by using static cluster-admin. Once if static cluster-admin is deployed the RBAC for the admin group automatically the members who are in the admin group from AAD can get admin level access which can create apply delete operations in the cluster. The admin team is also responsible for generating the Kube config file for the remaining type of users like developers, operation persons, users, etc.

## Developers

Once the allocation for admin members is done inside the cluster the admin members can simply deploy an RBAC manifest file that contains the developer’s group name. Members in the developer group, automatically get the access that was designed in the RBAC.

## Operations

Admin can generate a Kube config file for the operations team and the operation teams will use this config file to connect the cluster and perform the operational level activities. The admin team will decide the RBAC file for the operations team and inside the manifest file, access can be mentioned.

## Users

Admin will generate a Kube config file for the user’s group and the user’s group will use this configuration file to connect to the cluster with a set of privileges. the privileges can be decided in the RBAC file.

## Authorization table

|  |  |  |  |
| --- | --- | --- | --- |
| Group | Environment | Scope | Resources Groups |
| Admin | All clusters | Cluster | All |
| Developer | only develop env | Cluster | All, except node |
| Operations | only Production env | Cluster | All, except node |
| Users | only Production env | Namespace | All, except node |

**Environment:**

this can tell us the group about how the access for which type of Cluster.

**Scope:**

the level of access it can describe, cluster means the cluster level access namespace means the namespace level access only

**resource groups:**

This can describe which type of resources they have the access to do the action. The resources group examples are pods, services, stateful sets, daemon sets, deployment sets, etc.