**AZURE ACTIVE DIRECTORY**

1. Azure AD is nothing but an identity store in Azure. Here we can define users, groups, applications, and service principles. These users can authenticate onto Azure and they can access resources that are part of Azure subscription.
2. We can assign Azure AD roles to a user and these permissions are normally given to manage the various aspects of Azure AD.
3. While Kubernetes doesn't provide an identity management solution to store regular user accounts and passwords, you can integrate external identity solutions into Kubernetes. For AKS clusters, this integrated identity solution is Azure AD.

**ROLE-BASED ACCESS CONTROL**

Role-based access control (RBAC) is meant to authorize a user to use resources in Azure.

So for example, you could give a role for a user to go ahead and give them the ability to create a storage account or to manage resource groups.

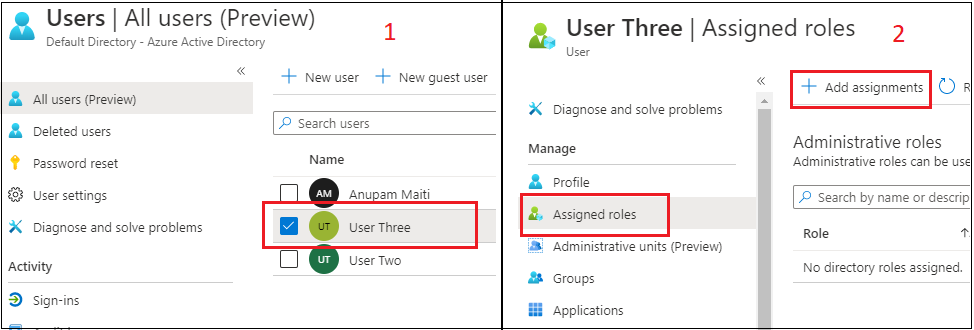
Role-based access control can be given at the management group level, subscription level, resource group level, or at the resource level.

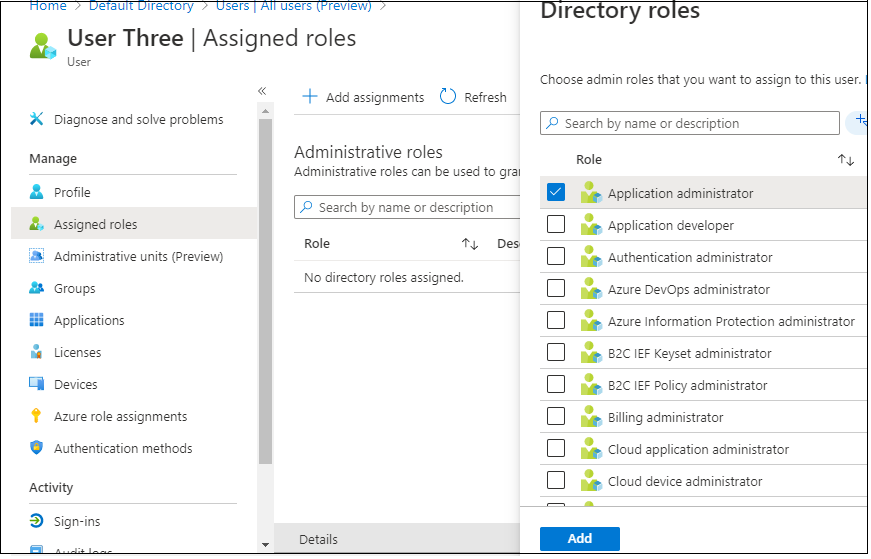
RBAC has three basic roles that apply to all resource types:

* *Owner*  
  This role has full access to all the resources and can delegate access to others.
* *Contributor*  
  This role can create and manage all types of resources, but can’t grant access to other users and groups.
* *Reader*  
  This role can view existing Azure resources.

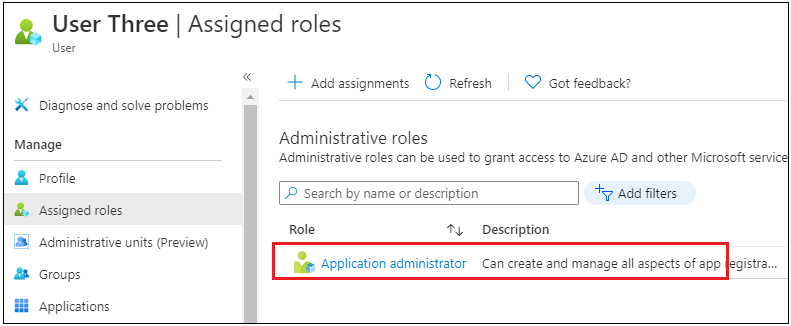
Assigning Azure AD Roles

Go to Azure Active Directory, and go to the Users section => click on a user for whom you want to add an AD Role. Now we can Assign roles for the user => Click on Assigned Role => + Add assignments.





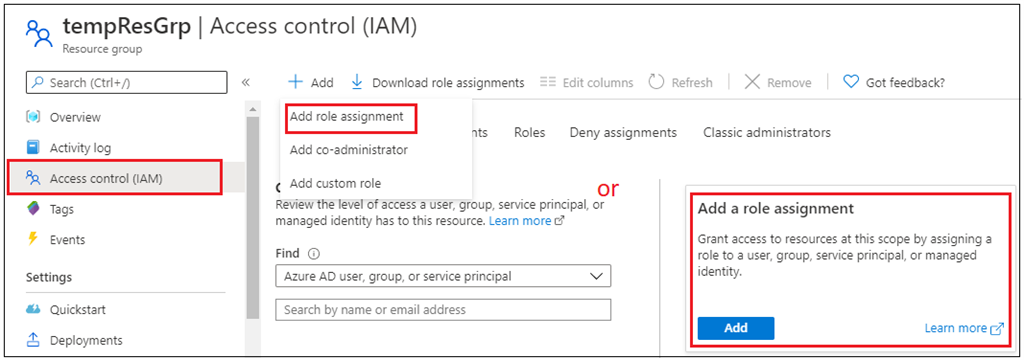
You will then see the role assigned to the user.



Assigning Role-Based Access Control

As we know, RBAC is used to give access to resources in Azure. Let's say you want to give RBAC access to a user for Resource Group.

Go to go Resource Group and click on the Access Control (IAM) option



Now click on the Add role assignment. In the next screen, you have to choose the Role to assign, and the principal to assign it to.

First, you can choose a Role, then choose access assign to, the last search for the user, and select. Once the user is selected, click on Save.

Azure AD roles are used to manage access to Azure AD resources, whereas Azure roles are used to manage access to Azure resources.

* The scope of Azure AD roles is at the tenant level, whereas the scope of Azure roles can be specified at multiple levels including management group, subscription, resource group, resource.
* I hope you found this information useful!

# Access and identity options for Azure Kubernetes Service (AKS)

You can authenticate, authorize, secure, and control access to Kubernetes clusters in a variety of ways.

* Using Kubernetes role-based access control (Kubernetes RBAC), you can grant users, groups, and service accounts access to only the resources they need.
* With Azure Kubernetes Service (AKS), you can further enhance the security and permissions structure via Azure Active Directory and Azure RBAC.

**Kubernetes RBAC**

Kubernetes RBAC provides granular filtering of user actions. With this control mechanism:

* You assign users or user groups permission to create and modify resources or view logs from running application workloads.
* You can scope permissions to a single namespace or across the entire AKS cluster.
* You create *roles* to define permissions, and then assign those roles to users with *role bindings*.
* To grant permissions across the entire cluster or to cluster resources outside a given namespace, you can instead use Cluster Roles.
* A Cluster Role grants and applies permissions to resources across the entire cluster, not a specific namespace.

### **Role Bindings and ClusterRoleBindings**

Once you've defined roles to grant permissions to resources, you assign those Kubernetes RBAC permissions with a Role Binding. If your AKS cluster [integrates with Azure Active Directory (Azure AD)](https://docs.microsoft.com/en-in/azure/aks/concepts-identity#azure-ad-integration), Role Bindings grant permissions to Azure AD users to perform actions within the cluster.

#### Role Bindings

Assign roles to users for a given namespace using Role Bindings. With Role Bindings, you can logically segregate a single AKS cluster, only enabling users to access the application resources in their assigned namespace.

To bind roles across the entire cluster, or to cluster resources outside a given namespace, you instead use ClusterRoleBindings.

#### ClusterRoleBinding

With a ClusterRoleBinding, you bind roles to users and apply to resources across the entire cluster, not a specific namespace. This approach lets you grant administrators or support engineer’s access to all resources in the AKS cluster.

With Azure AD-integrated AKS clusters, you can grant users or groups access to Kubernetes resources within a namespace or across the cluster.

1. To obtain a kubectl configuration context, a user runs the [az aks get-credentials](https://docs.microsoft.com/en-us/cli/azure/aks" \l "az_aks_get_credentials) command.
2. When a user interacts with the AKS cluster with kubectl, they're prompted to sign in with their Azure AD credentials.

This approach provides a single source for user account management and password credentials. The user can only access the resources as defined by the cluster administrator.

Azure AD authentication is provided to AKS clusters with OpenID Connect. OpenID Connect is an identity layer built on top of the OAuth 2.0 protocol. For more information on OpenID Connect, see the [Open ID connect documentation](https://docs.microsoft.com/en-in/azure/active-directory/develop/v2-protocols-oidc). From inside of the Kubernetes cluster, [Webhook Token Authentication](https://kubernetes.io/docs/reference/access-authn-authz/authentication/" \l "webhook-token-authentication) is used to verify authentication tokens. Webhook token authentication is configured and managed as part of the AKS cluster.

### Webhook and API server

As shown in the graphic above, the API server calls the AKS webhook server and performs the following steps:

1. kubectl uses the Azure AD client application to sign in users with [OAuth 2.0 device authorization grant flow](https://docs.microsoft.com/en-in/azure/active-directory/develop/v2-oauth2-device-code).
2. Azure AD provides an access\_token, id\_token, and a refresh\_token.
3. The user makes a request to kubectl with an access\_token from kubeconfig.
4. kubectl sends the access\_token to API Server.
5. The API Server is configured with the Auth WebHook Server to perform validation.
6. The authentication webhook server confirms the JSON Web Token signature is valid by checking the Azure AD public signing key.
7. The server application uses user-provided credentials to query group memberships of the logged-in user from the MS Graph API.
8. A response is sent to the API Server with user information such as the user principal name (UPN) claim of the access token, and the group membership of the user based on the object ID.
9. The API performs an authorization decision based on the Kubernetes Role/RoleBinding.
10. Once authorized, the API server returns a response to kubectl.
11. kubectl provides feedback to the user.

Learn how to integrate AKS with Azure AD with our [AKS-managed Azure AD integration how-to guide](https://docs.microsoft.com/en-in/azure/aks/managed-aad).

