Best practices for authentication and authorization in Azure Kubernetes Service (AKS)

Best practices for authentication and authorization in Azure Kubernetes Service (AKS)

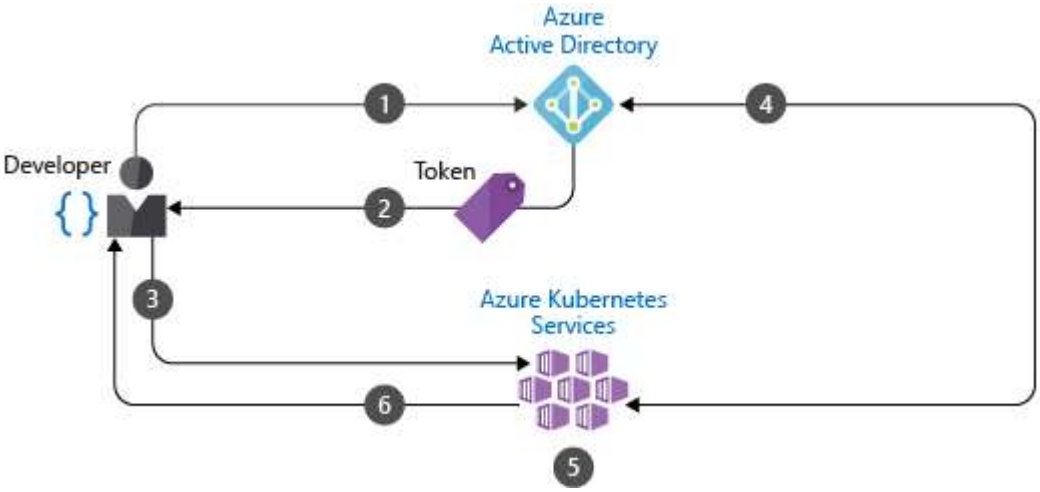
1. Authenticate AKS cluster users with Azure Active Directory.
2. Control access to resources with Kubernetes role-based access control (Kubernetes RBAC).
3. Use Azure RBAC to granularly control access to the AKS resource, the Kubernetes API at scale, and the kubeconfig.
4. Use a managed identity to authenticate pods themselves with other services.

For more information about these features, see https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-identity & https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-cluster-isolation

1. **Use Azure Active Directory (Azure AD) - Authentication**

Kubernetes lacks an identity management solution for you to control the resources with which users can interact. Instead, you typically integrate your cluster with an existing identity solution.

Deploy AKS clusters with Azure AD integration. Using Azure AD centralizes the identity management component. Any change in user account or group status is automatically updated in access to the AKS cluster. Scope users or groups to the minimum permissions amount using Roles, ClusterRoles, or Bindings - https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-identity#use-kubernetes-role-based-access-control-kubernetes-rbac .



1. Developer authenticates with Azure AD.
2. The Azure AD token issuance endpoint issues the access token.
3. The developer performs an action using the Azure AD token, such as kubectl create pod.
4. Kubernetes validates the token with Azure AD and fetches the developer's group memberships.
5. Kubernetes RBAC and cluster policies are applied.
6. Developer's request is successful based on previous validation of Azure AD group membership and Kubernetes RBAC and policies.

How Azure AD integration works behind the scenes? https://docs.microsoft.com/en-us/azure/aks/concepts-identity#azure-ad-integration

With Azure AD-integrated AKS clusters, admins can grant AAD users or groups, access to Kubernetes resources within a namespace or across the cluster. This can be accomplished by Azure role-based access control (RBAC) in conjunction with Kubernetes RBAC.
An AKS cluster has two types of credentials for calling the Kubernetes API server:

- cluster admin - full access to the AKS cluster
- cluster user - no permissions by default on the AKS cluster

The above two roles are assigned to an Azure AD user by using the Azure RBAC Roles mentioned below:

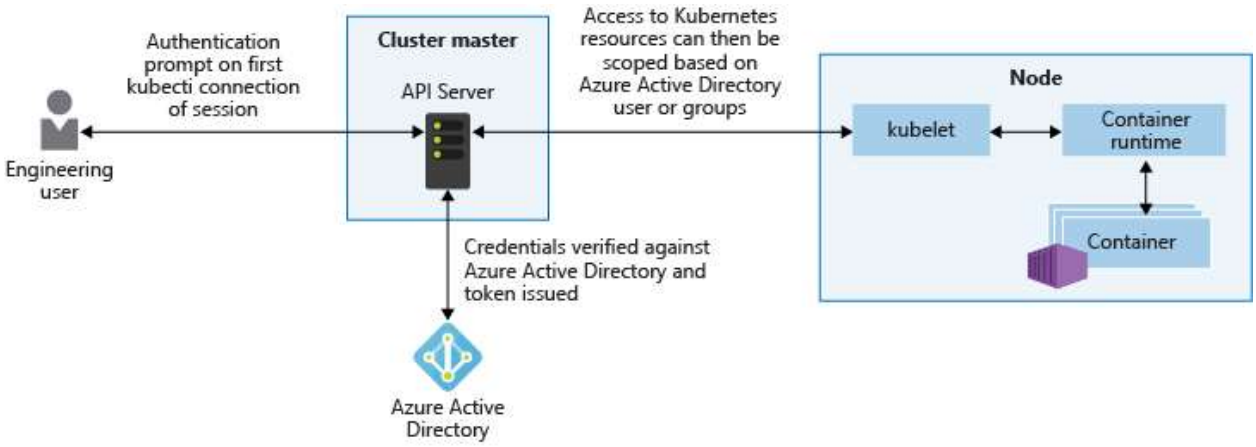
- **Azure Kubernetes Service Cluster Admin Role** - It has permission to download the cluster admin credentials. Only cluster admin should be assigned this role. Azure Contributor Role has this in-built role added to it, that's why all Azure AD users with **Contributor Role** are cluster admins.
- **Azure Kubernetes Service Cluster User Role** - It has permission to download the cluster user credentials. Non-admin users can be assigned to this role. This role does not give any particular permissions on Kubernetes resources inside the cluster — it just allows a user to connect to the API server. Kubernetes permissions using RBAC (Roles, RoleBindings & ClusterRoles for cases like installing Couchbase DB) will be assigned to these Azure AD users.

When the command “az aks get-credentials --admin” is executed by cluster admin, it downloads the cluster admin credentials and saves them into the kubeconfig file.

The cluster administrator can use this kubeconfig to create Roles and RoleBindings, and assign them to the user.

Lets say an Azure AD user "Engineering User" runs the command “az aks get-credentials” to obtain a kubectl configuration context

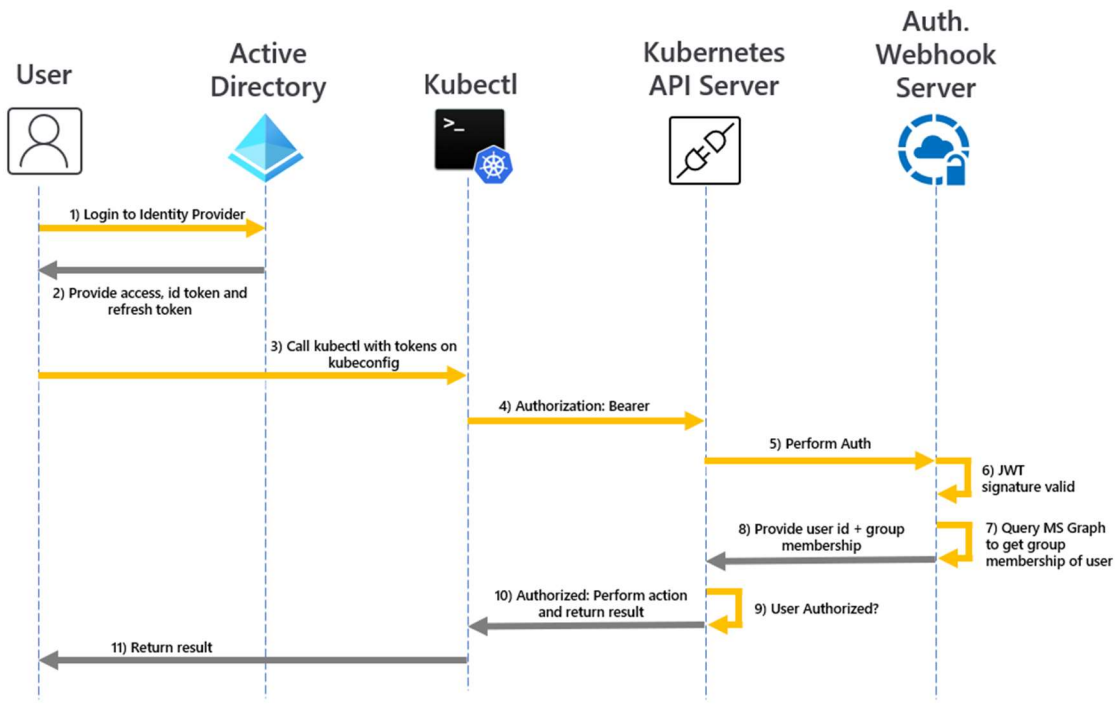
When this user then interacts with the AKS cluster with kubectl, he is prompted to sign in with his Azure AD credentials. This approach provides a single source for user account management and password credentials. The user can only access the resources as defined by the cluster administrator.



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-us/azure/active-directory/develop/v2-protocols-oidc. From inside of the Kubernetes cluster, Webhook Token Authentication -

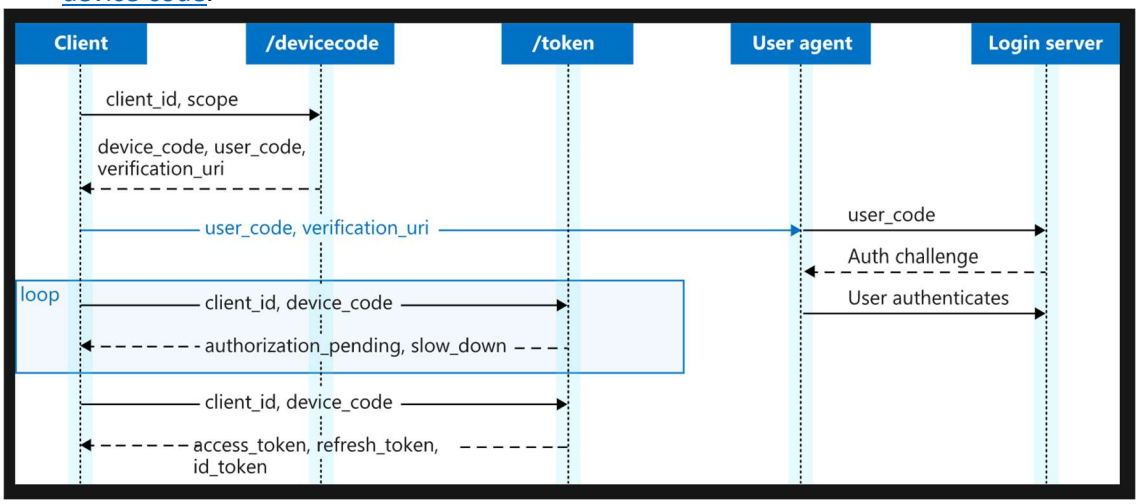
https://kubernetes.io/docs/reference/access-authn-authz/authentication/#webhook-token-authentication is used to verify authentication tokens. Webhook token authentication is configured and managed as part of the AKS cluster.

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-us/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.

Integrate AKS with Azure AD with AKS-managed Azure AD integration how-to guide - https://docs.microsoft.com/en-us/azure/aks/managed-aad.

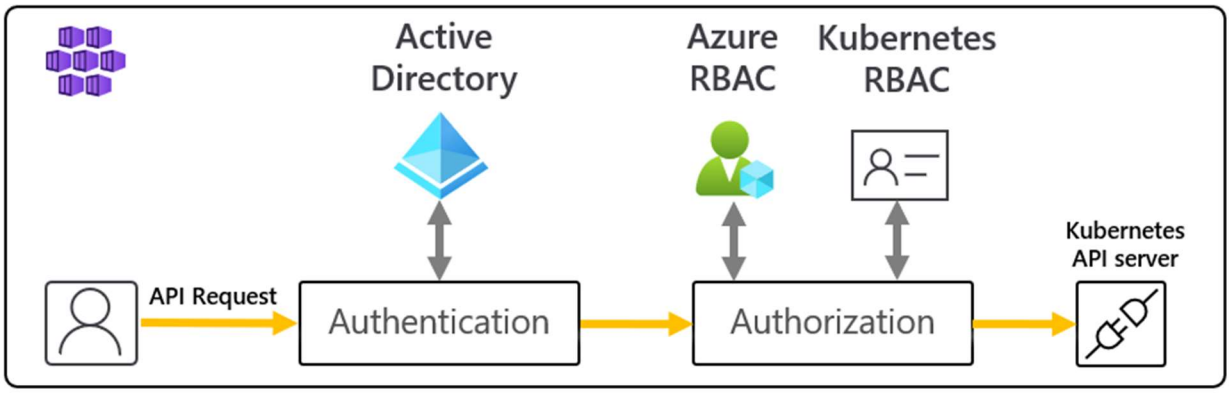
Azure built-in roles for Containers (AKS & ACR):
https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#containers
https://docs.microsoft.com/en-us/answers/questions/162264/built-in-roles-for-azure-kubernetes-service-aks.html

2. **Authorization - Use Kubernetes RBAC – Grant access to K8s objects like pods, services, etc)**
Define user or group permissions to cluster resources with Kubernetes RBAC. Create roles and bindings that assign the least amount of permissions required. Integrate with Azure AD to automatically update any user status or group membership change and keep access to cluster resources current.
https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-identity#use-kubernetes-role-based-access-control-kubernetes-rbac

3. **Authorization - Use Azure RBAC – Grant access to Azure resources like AKS, ACR, etc.**
Use Azure RBAC to define the minimum required user and group permissions to AKS resources in one or more subscriptions.
   - Kubernetes RBAC - Designed to work on Kubernetes resources within your AKS cluster.
   - Azure RBAC - Designed to work on resources within your Azure subscription.

   There are two levels of access needed to fully operate an AKS cluster:
   1. Access the AKS resource in your Azure subscription.
      - Control scaling or upgrading your cluster using the AKS APIs. For example, you could use the Azure Kubernetes Service Contributor role to scale and upgrade your cluster.
      - Pull your kubeconfig. For example, user with the Azure Kubernetes Service Cluster Admin role only has permission to pull the Admin kubeconfig. *This role by-passes the AAD authentication & should only be used as a backdoor.*
   2. Access to the Kubernetes API. This access is controlled by either:
      - Kubernetes RBAC (traditionally) i.e., creating roles & role bindings. https://docs.microsoft.com/en-us/azure/aks/concepts-identity#kubernetes-rbac
      - Integrating Azure RBAC with AKS for Kubernetes authorization - With this feature, you not only give users permissions to the AKS resource across subscriptions, but you also configure the role and permissions for inside each of those clusters controlling Kubernetes API access. For example, you can grant the Azure Kubernetes Service RBAC Reader role on the subscription scope. The role recipient will be able to list and get all Kubernetes objects from all clusters without modifying them.

https://docs.microsoft.com/en-us/azure/aks/concepts-identity#azure-rbac-for-kubernetes-authorization



4. **Use pod identities with a central Azure AD identity solution**

Use Azure Active Directory pod-managed identities in Azure Kubernetes Service (Preview) - https://docs.microsoft.com/en-us/azure/aks/use-azure-ad-pod-identity – This service is in preview & will be will be replaced with Azure AD Workload Identity - https://azure.github.io/azure-workload-identity/docs/. The timeline for Workload Identity to be GA is Q3, 2022. If you use Pod Identity, there will be a migration path to Workload Identity.
Azure AD Workload Identity best explained here - Azure AD Workload Identity (24/03/2022) – YouTube - https://www.youtube.com/watch?v=FKQG3RjFbfU

Azure Container Registry (ACR)

The Azure Container Registry service supports a set of built-in Azure roles that provide different levels of permissions to an Azure container registry. You can also define custom roles with fine-grained permissions to a registry for different operations. https://docs.microsoft.com/en-us/azure/container-registry/container-registry-roles?tabs=azure-cli#custom-roles
You can use repository namespaces sharing a single registry across multiple groups within your organization, but you can't currently assign repository-scoped permissions to an Azure Active Directory object such as a service principal or managed identity. Token with repository scoped permissions - in preview - https://docs.microsoft.com/en-us/azure/container-registry/container-registry-repository-scoped-permissions
Azure Container Registry provides the automated methods to remove tags and manifests using acr purge & setting a retention policy. Automatically purge tags & manifests - in preview - https://docs.microsoft.com/en-us/azure/container-registry/container-registry-delete#automatically-purge-tags-and-manifests