Azure Kubernetes Service - Putting it All Together.

Microsoft

Warning

MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.
Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.
Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, our provision of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.
The descriptions of other companies' products in this document, if any, are provided only as a convenience to you. Any such references should not be considered an endorsement or support by Microsoft. Microsoft cannot guarantee their accuracy, and the products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. For authoritative descriptions of these products, please consult their respective manufacturers.
© 2021 Microsoft Corporation. All rights reserved. Any use or distribution of these materials without express authorization of Microsoft Corp. is strictly prohibited.
Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Version 1.0
Publication: Janurary 20th, 2021
Author: Mohamed Soumare
Senior Consultant | Microsoft Consulting Services

Introduction

Kubernetes in action…

Kubernetes with Docker have taken containerized application development and deployment by storm. To ensure deployment consistency across multiple environments whether it's from your local machine, on-Premises server, or any of the Cloud providers (Azure, AWS, GCP and others alike), Docker is the leading container solution compatible with both Linux and Windows operating systems.

With the multitude of containerized applications giving birth in the development world, comes the challenge and need of managing containers lifecycle along with their networking and storage components. Among the widely used container management solutions stands out Kubernetes, a Google open-source project.

Kubernetes by itself is a beast to handle, because it requires lot of experience to setup the raw Kubernetes engine along with the Master nodes & Agent nodes, security, monitoring etc. DevOps engineers will have to go through a steep learning curve if they want to feel comfortable operating a Kubernetes environment, let alone Developers who usually don't deal with setting up servers.

But wait… help is here! Microsoft, just like the major Cloud providers, has taken the burden of setting the Kubernetes engine properly and optimally, so you can focus on the other aspects of building your applications, and came to light Managed Kubernetes such as Azure Kubernetes Service.

In this whitepaper, using Infrastructure as Code & Azure DevOps for CI/CD, I will take you through deploying Azure Kubernetes Service with the other related components such as Azure Virtual Network, Azure Bastion, Azure Container Registry, Azure Active Directory, Application Gateway, Ingress Controller, Pod Identity, Log Analytics, Certificate Management etc.

Contents

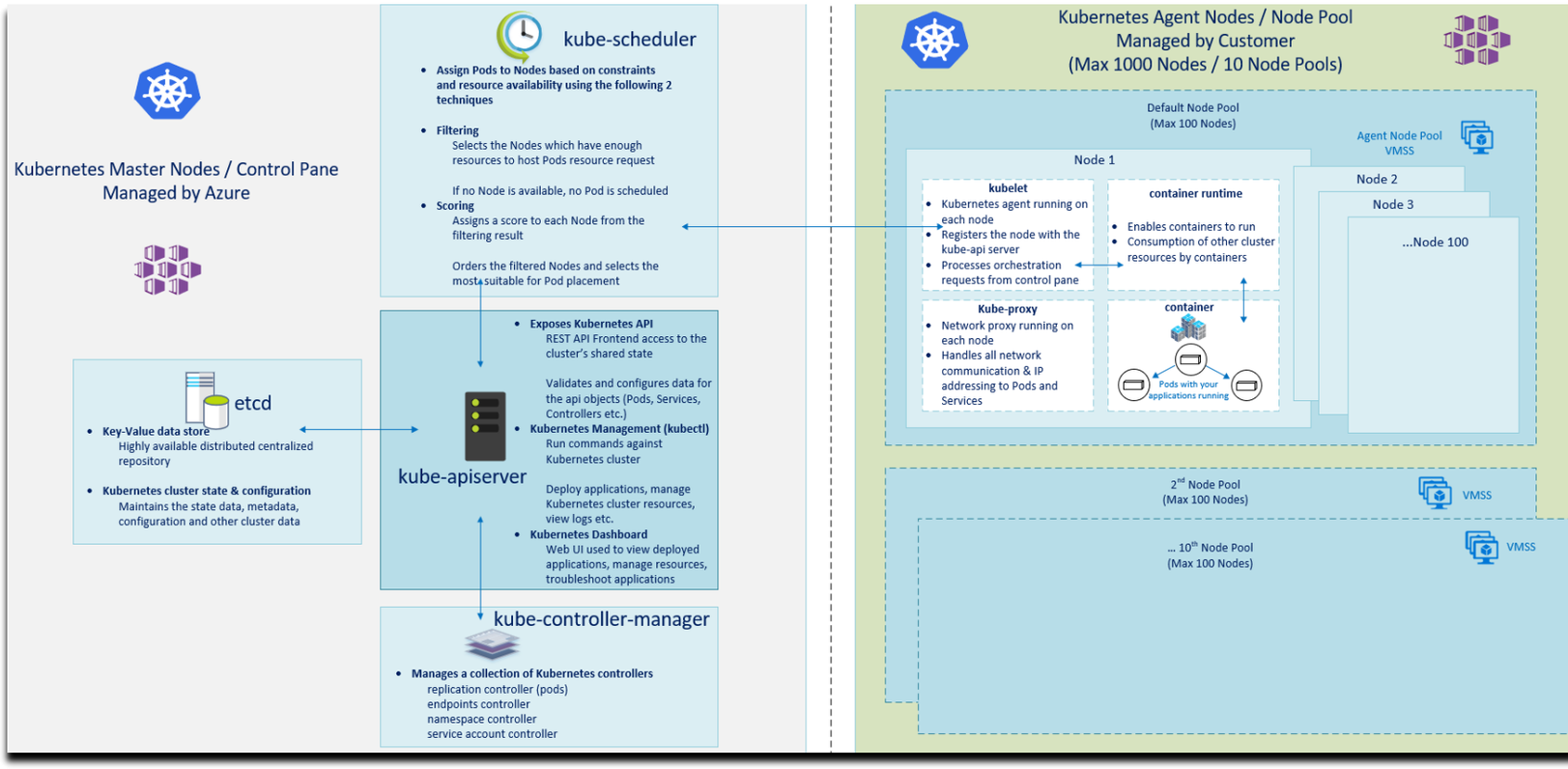
1. Understanding the components of Azure Kubernetes Service 6
2. Our Target Architecture 7
3. Virtual Network 8
3.1 Deployment with Azure DevOps & CI/CD 8
4. Key Vault 14
4.1. Deployment with Azure DevOps & CI/CD 14
5. Build Agent 18
5.1. Deployment with Azure DevOps & CI/CD 18
5.2. Generate a Personal Access Token 23
5.3. Download and Install the Agent 25
5.7. Install Chocolatey 30
5.8. Install sed 30
5.9. Install Kube client 30
5.10. Install Kubelogin 30
6. Management Box 30
6.1. Deployment with Azure DevOps & CI/CD 30
7. Azure Container Registry 36
7.1. Deployment with Azure DevOps & CI/CD 36
7.2. Generate Docker files 38
7.3. Create Container Images 40
7.4. Push Container Images to ACR with Azure DevOps & CI/CD 41
8. Azure Kubernetes Service 48
8.1. Cluster Sizing & Forcasting for Growth 48
8.2. Public Cluster vs Private Cluster 48
8.3. Integration with Azure AD 48
8.4. Deployment with Azure DevOps & CI/CD 50
9. Deploying our microservices in AKS 52
9.1. Preparing Kubernetes Deployment & Service Manifests (using default namespace) 53
9.2. Preparing Kubernetes Deployment & Service Manifests (using custom namespaces) 54
9.3. Deployment with Azure DevOps & CI/CD 57
10. Application Gateway 66
10.1. Deployment with Azure DevOps & CI/CD 66
11. Azure AD Pod Identity & Application Gateway Ingress Controller (AGIC) 72
12. DNS Zone 73
12.1. Deployment with Azure DevOps & CI/CD 74
13. Bonus 77

1. Understanding the components of Azure Kubernetes Service

Before we discuss the target architecture to deploy in this whitepaper, it's important to understand the different components that make up Azure Kubernetes Service (AKS). Like we already talked about in the introduction, Kubernetes is a beast to tame. Microsoft Azure, just like the other major Cloud providers, has taken the burden of configuring the Kubernetes engine so we can focus on deploying and running our applications in it. This process has led to what's called "Managed Kubernetes", which is essentially composed of 2 major components.

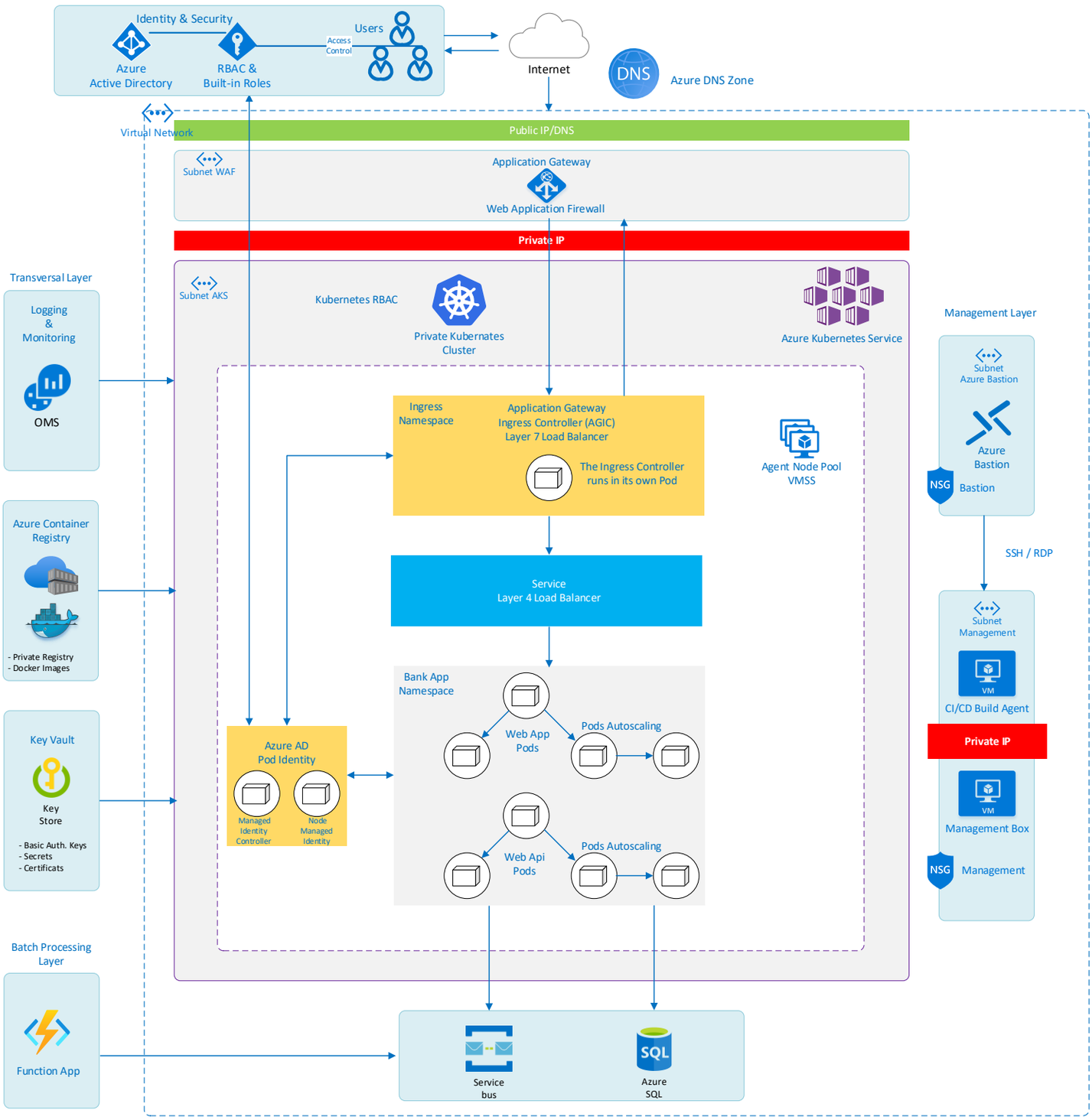
- Kubernetes Master Nodes (Control Pane) => Managed by Azure
- Kubernetes Agent Nodes (Nodes & Node Pools) => Managed by Customer

The diagram below illustrates the Master and Agents Nodes of Azure Kubernetes Service along with their internals **(Zoom for better viewing).**



2. Our Target Architecture

The following diagram depicts the architecture that we will put in place along with the other related workloads we will deploy on Azure. To make things simpler we will deploy all workloads/resources in 1 Virtual Network. Another way to do this is to use a Hub & Spoke Architecture whereby we will have 2 different virtual networks and connect them together using VNET peering. In that case, the Hub will contain all management workloads and the Spoke will house the actual workloads responsible for running our applications.



3. Virtual Network

Virtual Networking in AKS is made possible either by using Kubernetes built-in networking feature **kubenet** which is the default option or by using **Azure Container Networking Interface,** which allows you to connect to a new or existing Azure Virtual Network. Even though kubenet works, it comes with its limitations such as 400 max number of nodes in the cluster and 110 max number of pods per node.

In this example, we will be using Azure CNI, however a good network planning needs to happen beforehand to account for cluster upgrades, nodes, pods, load balancers and all other resources that will be deployed throughout the lifetime of the cluster.

As a rule of thumb, the following formula is used to calculate the minimum subnet size required including 1 additional node for rolling upgrade operations.

*(number of nodes + 1) + ((number of nodes + 1) * number of pods per node)*

In our scenario, we plan on deploying an AKS cluster with:

- 1 Node Pool (max is 10).
- **45 Nodes** (max is 100 per node pool) and we want to account for cluster scaling to **10 Nodes** additional.
- **60 pods** per nodes (max is 250, default is 30).

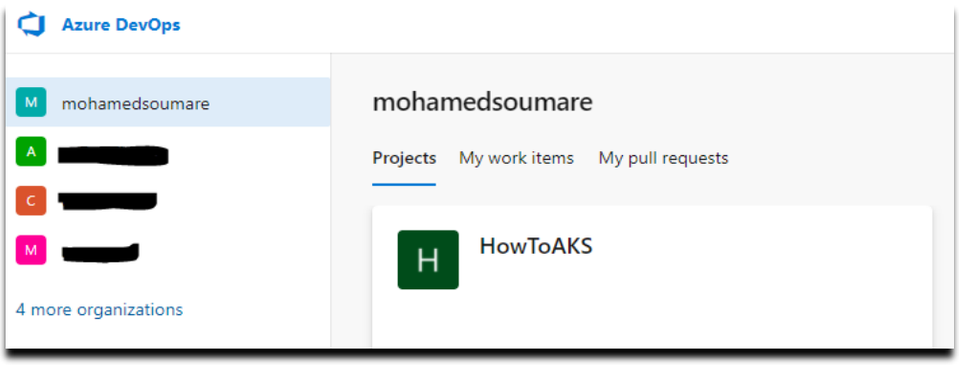
Hence, based on the formula above, our calculation will result to: (45 + 10 + 1) + ((45 +10 + 1) * 60) = 3360, which means a network size of **/20** (4096 IP addresses or larger would be good enough to accommodate all the resources in our AKS cluster.

Using Infrastructure as Code with Azure DevOps for CI/CD, we will deploy our Virtual Network with 4 subnets (Application Gateway Subnet, AKS Subnet, Bastion Subnet and Management Subnet), 2 Network Security Groups (Bastion NSG and Management NSG). Please find the reusable ARM templates in the following GitHub repository.

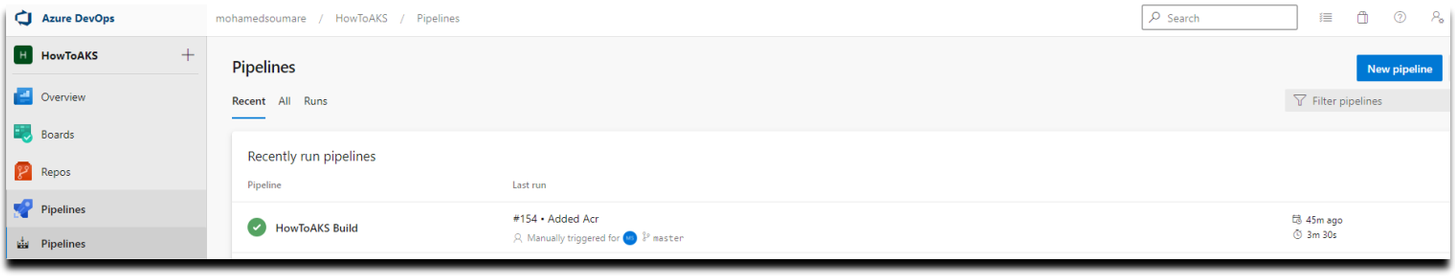
Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/1-network*

3.1 Deployment with Azure DevOps & CI/CD

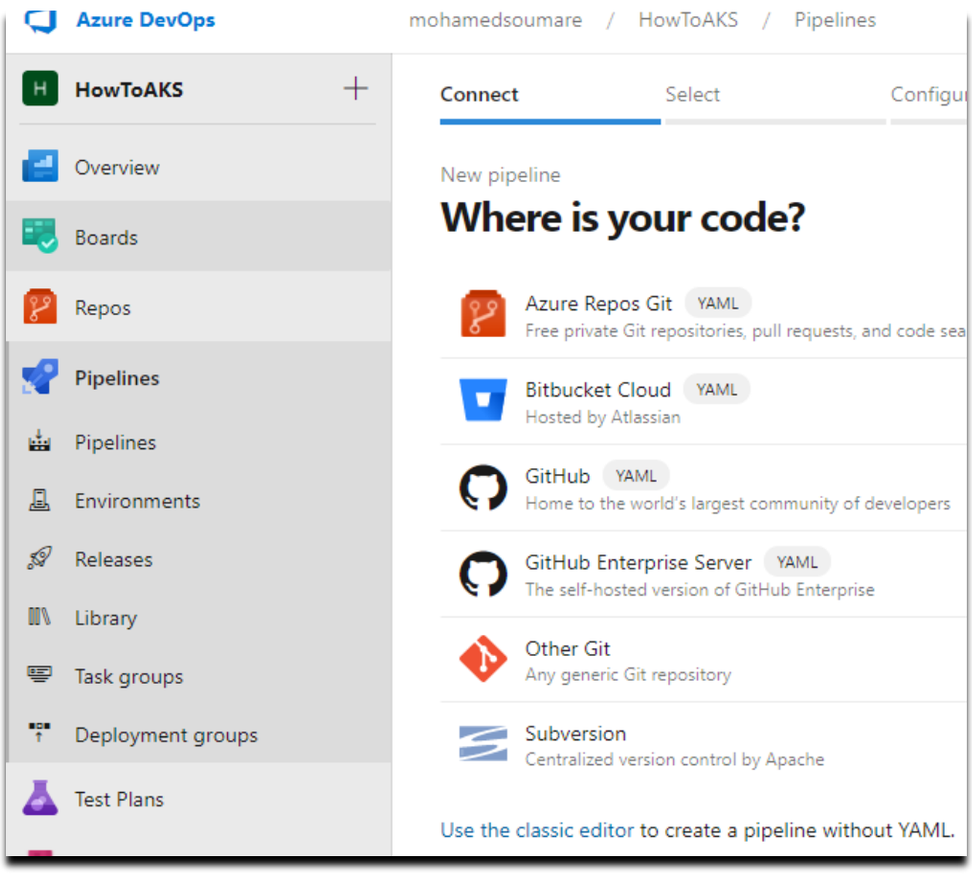
Step 1 > Navigate to Azure DevOps > Select your project. In our case we've named it HowToAKS as you can see in the following screenshot.



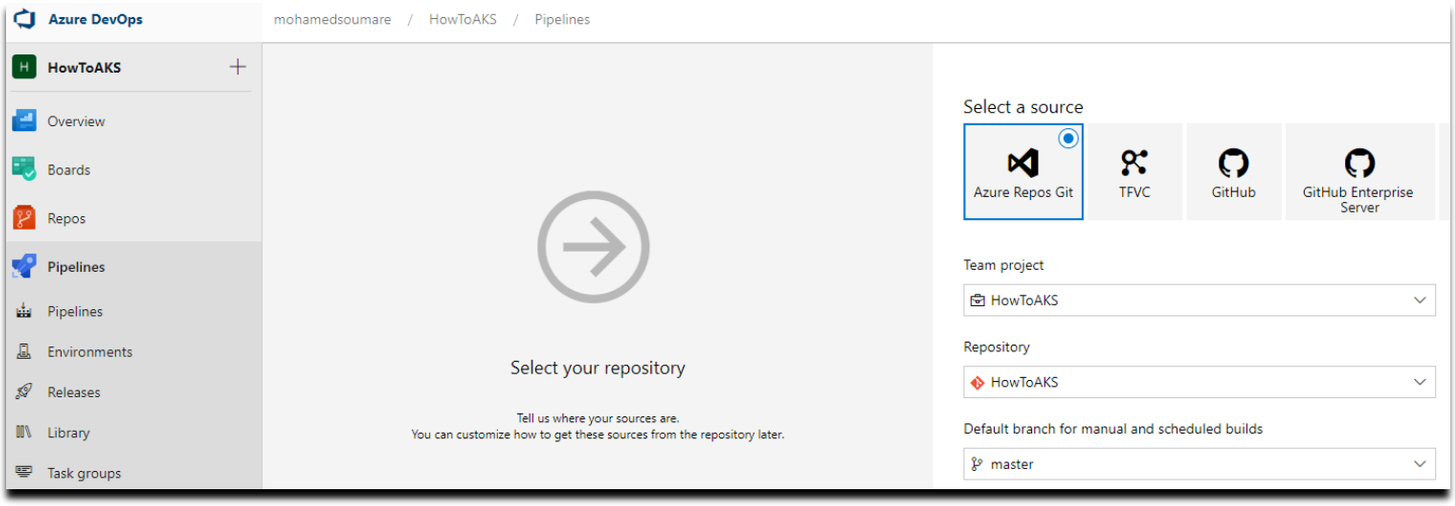
Step 2 > Select Pipelines > New Pipeline.



Step 3 > Select Use the classic editor (or use the YAML Editor)

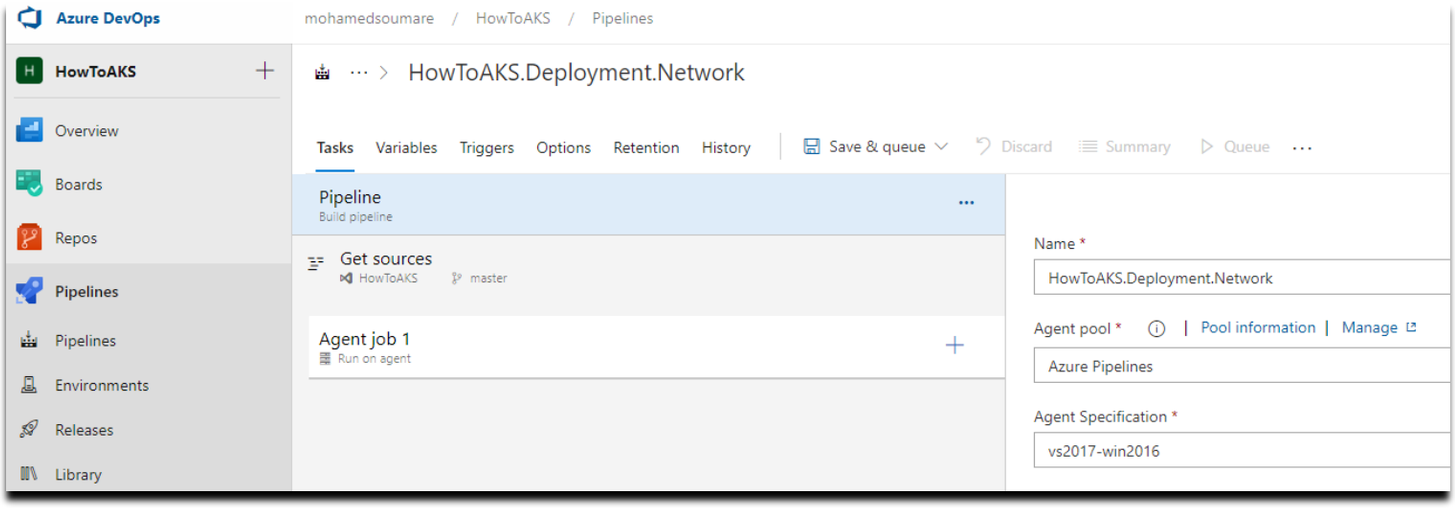


Step 4 > Select your source code repository and click on Continue.

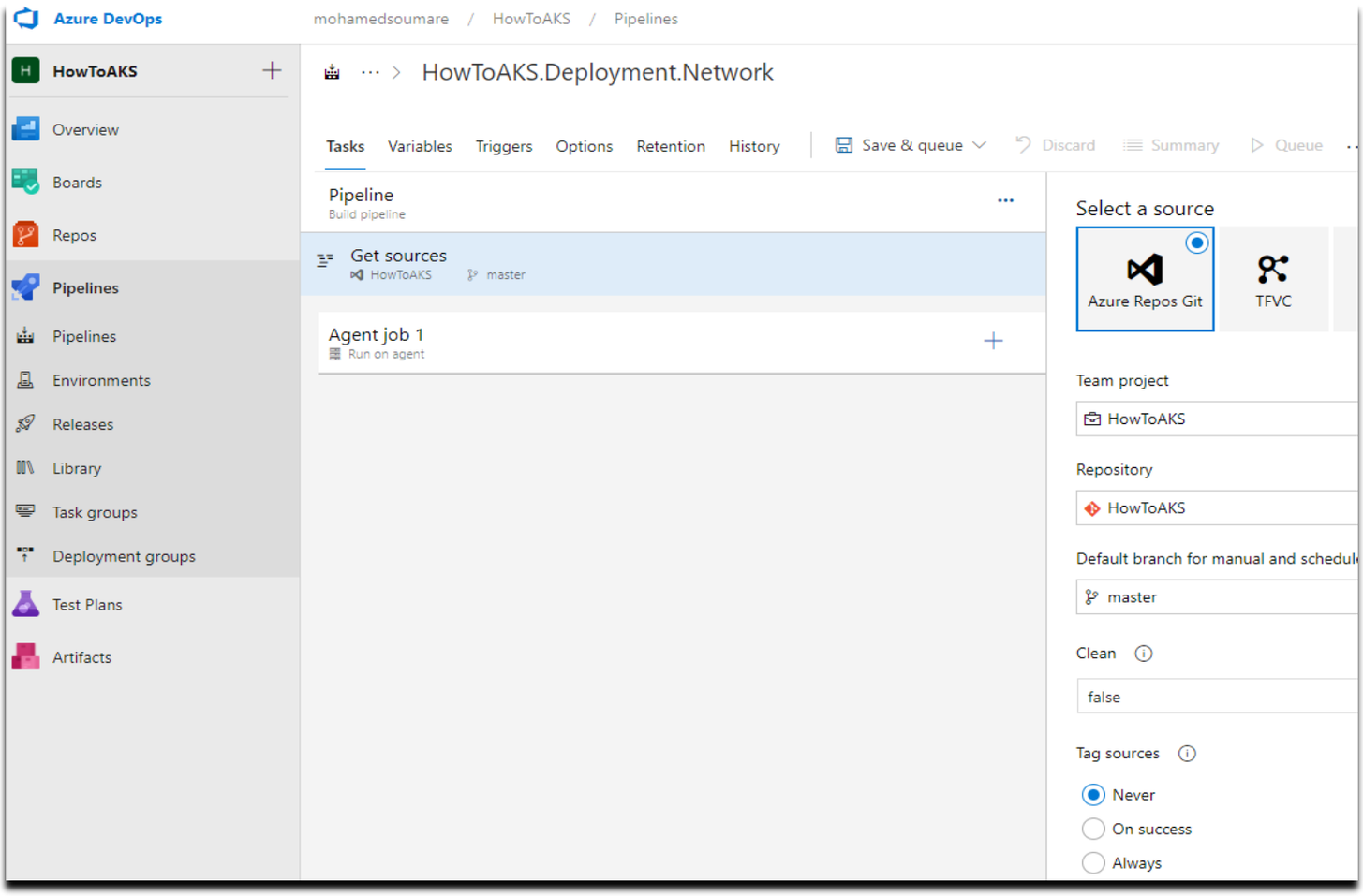


Step 5 > Start with an Empty Job.

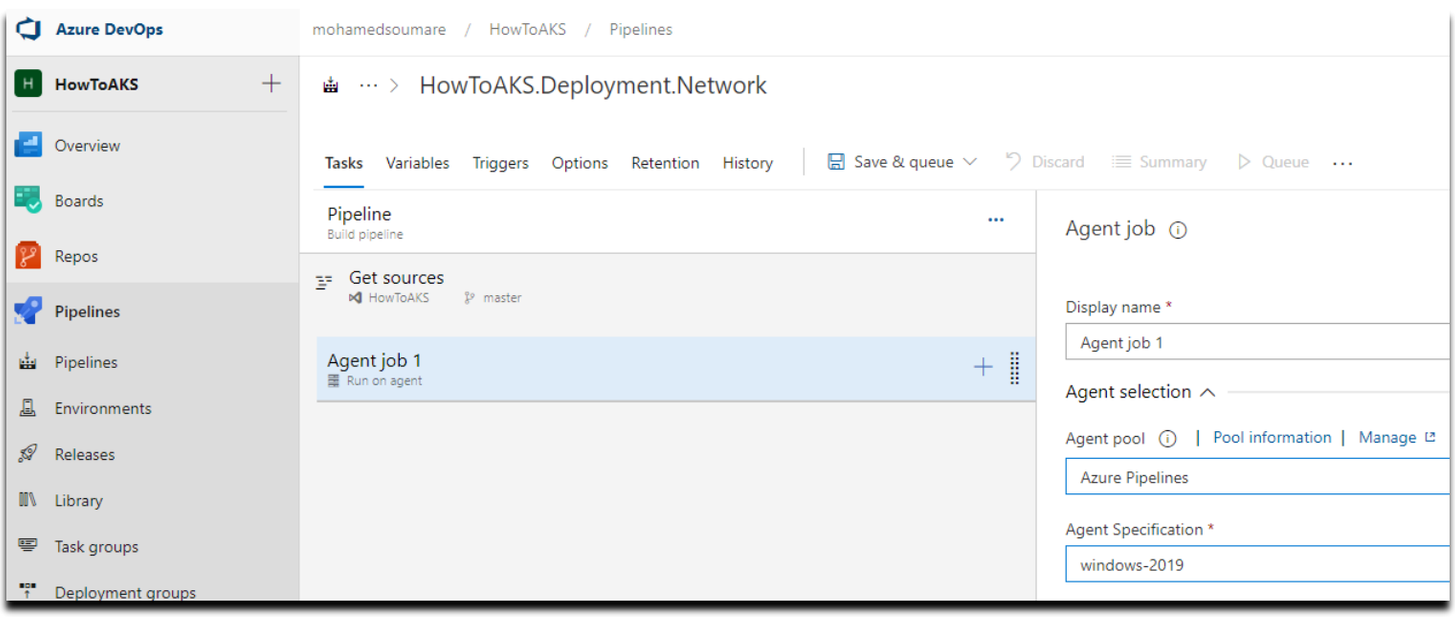
Step 6 > Give a name to the Pipeline (e.g., HowToAKS.Deployment.Network).



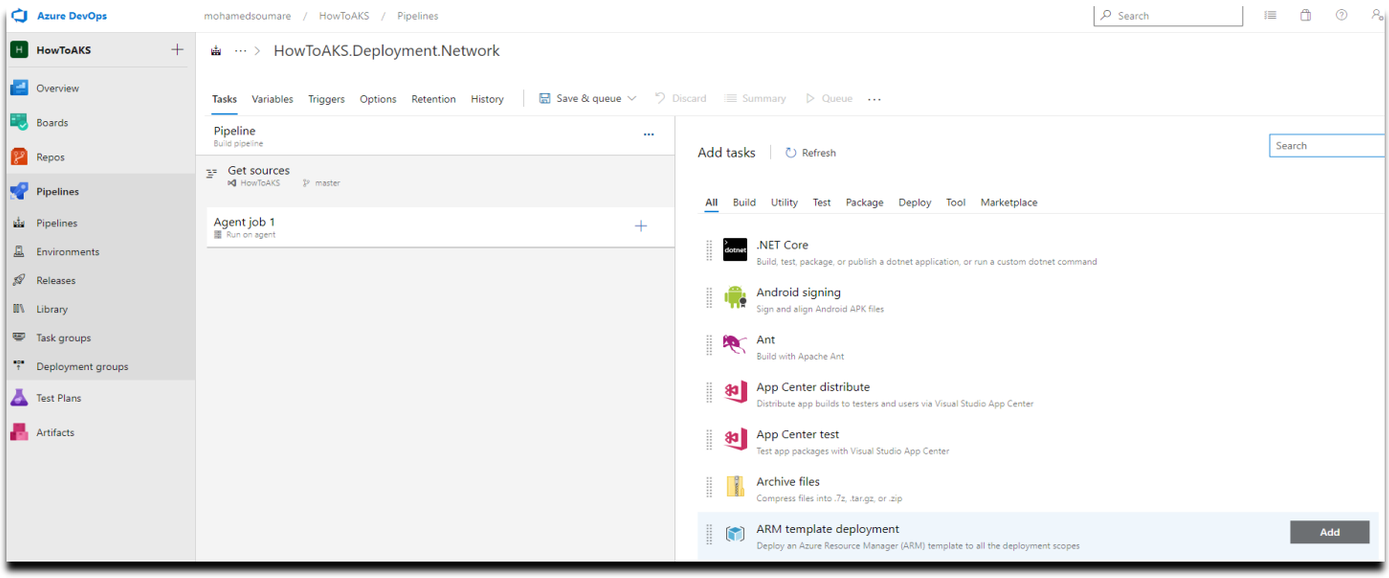
Step 7 > Select Get sources and make any necessary adjustments (such as tagging, cleaning etc.)



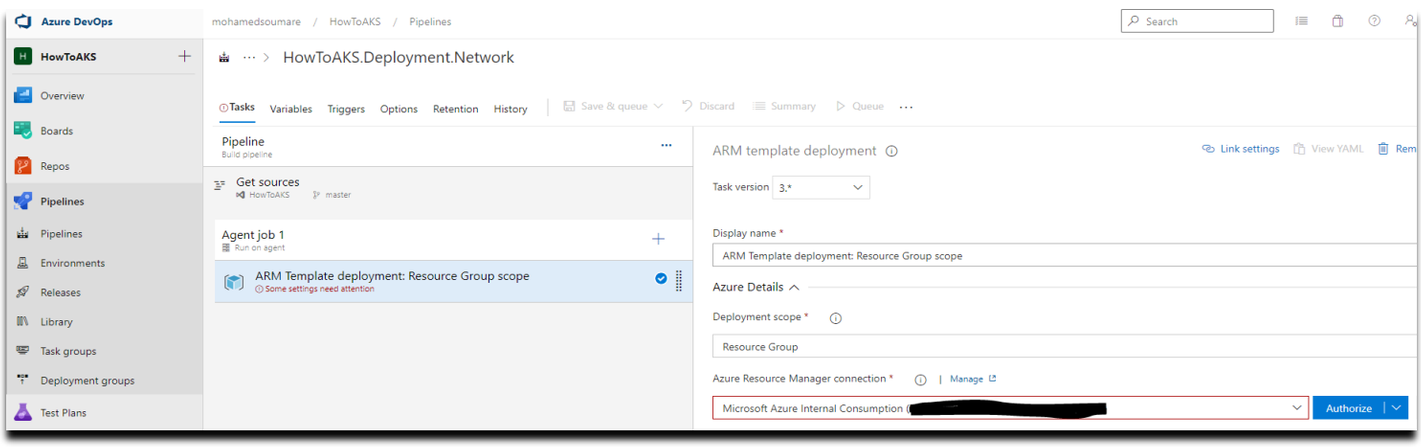
Step 8 > Select Agent Job > Choose the appropriate Agent Pool and Agent Specifications



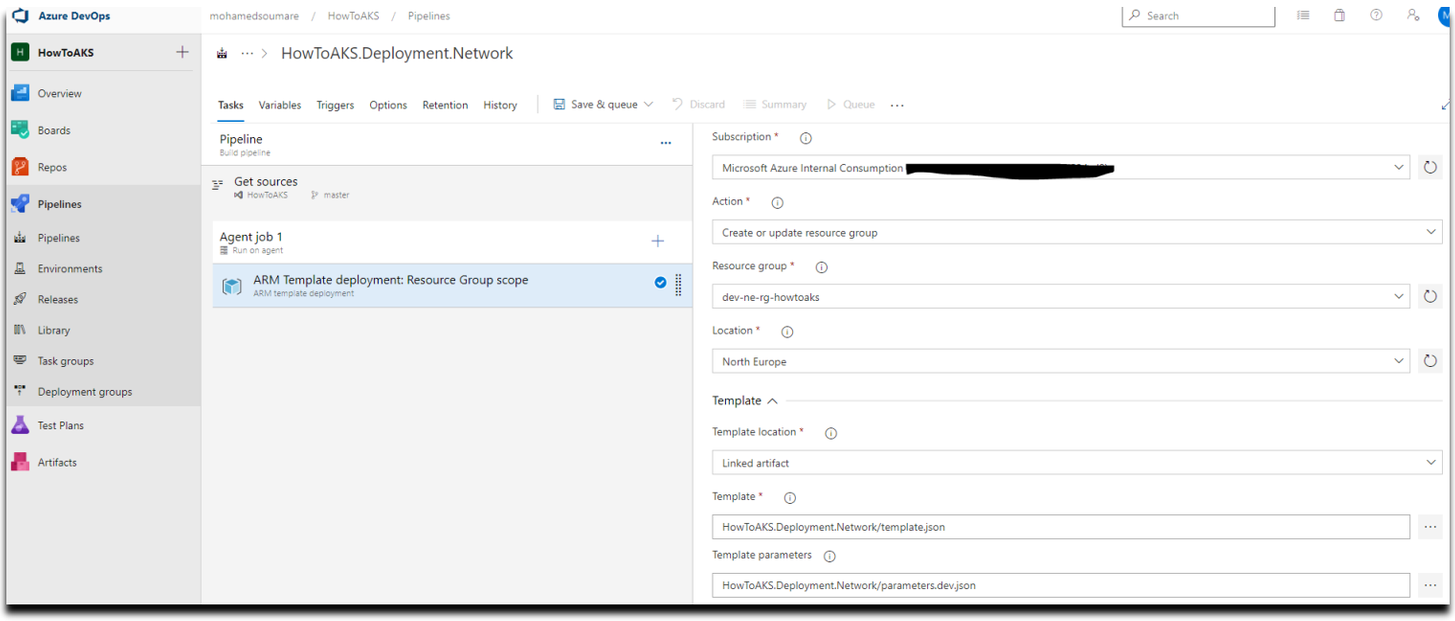
Step 9 > Click on the + (plus sign) next to Agent Job to add a task > Click Add ARM template deployment from the list.



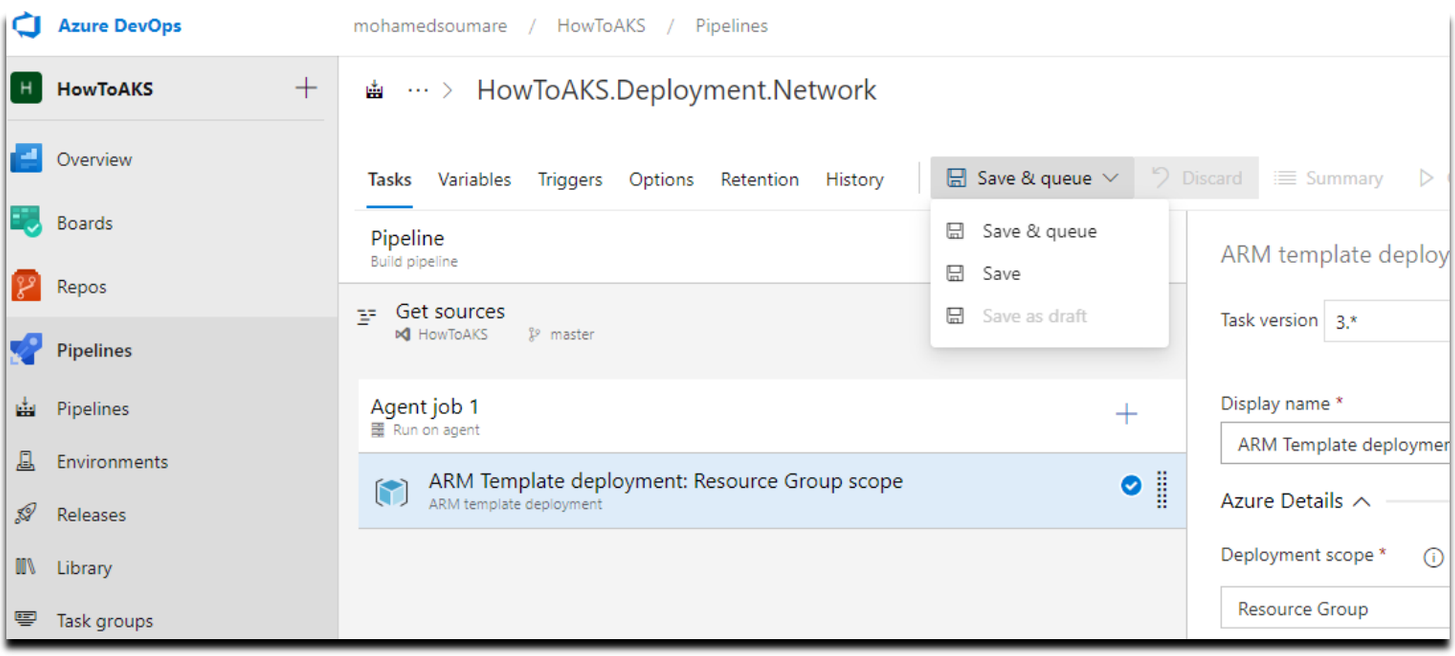
Step 10 > Set the Azure Resource Manager connection (choose an existing service connection or select your subscription and click authorize to create a new service connection). In our case we are creating a new service connection, so let's click on Authorize and login to our Azure Subscription.



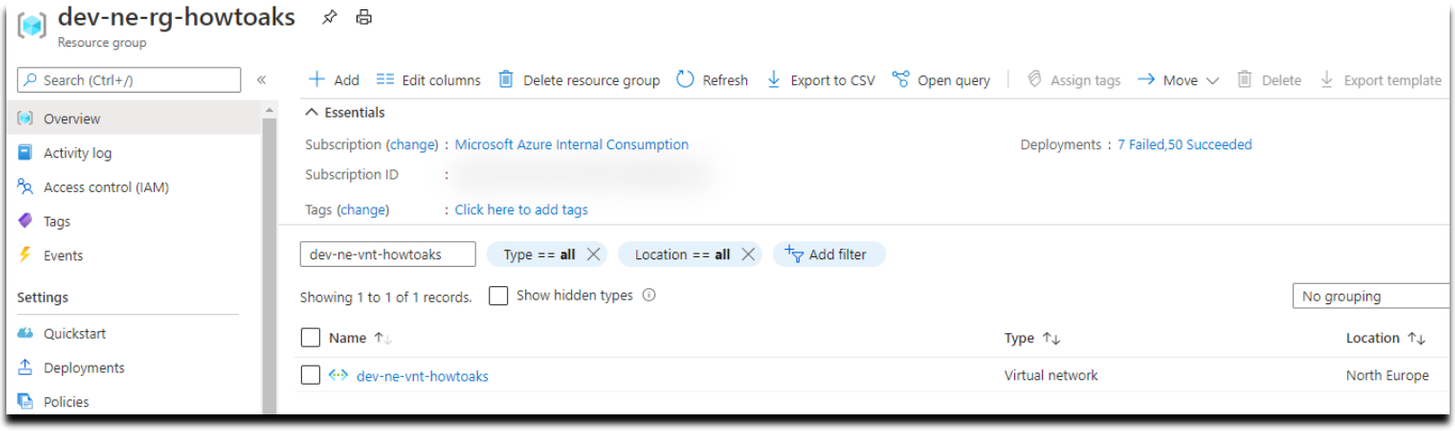
Step 11 > Select your Subscription, Resource Group, Location, Template, and Template Parameters.



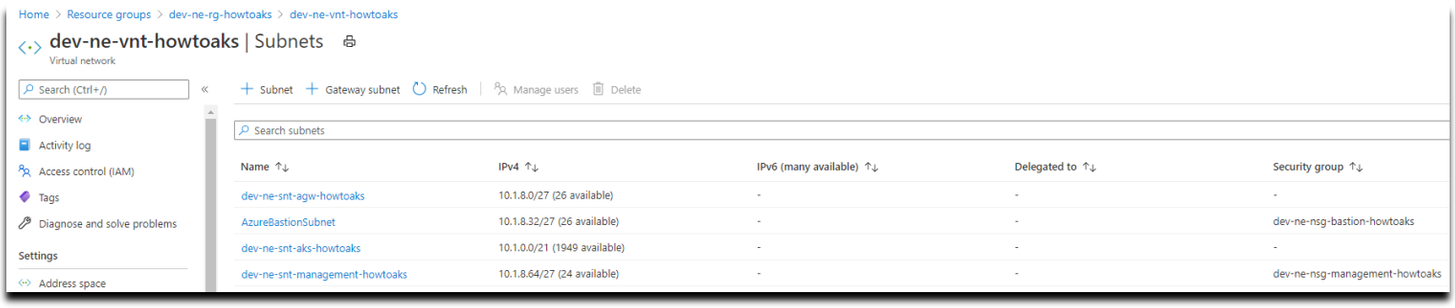
Step 12 > Select Save & Queue to launch the deployment of the Virtual Network.



Step 13 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the newly created Virtual Network.



Step 14 > Select Subnets > You will see the Subnets and Network Security Groups.



4. Key Vault

To securely store and access our secrets, keys, and certificates we will use Azure Key Vault. Since we are using Infrastructure as Code with Azure DevOps for CI/CD, we will create the following passwords in the Key Vault and allow Azure DevOps to retrieve the passwords during the creation of the resources.

- Virtual Machine Build Agent Password
- Virtual Machine Management Password
- SQL Server DB Password

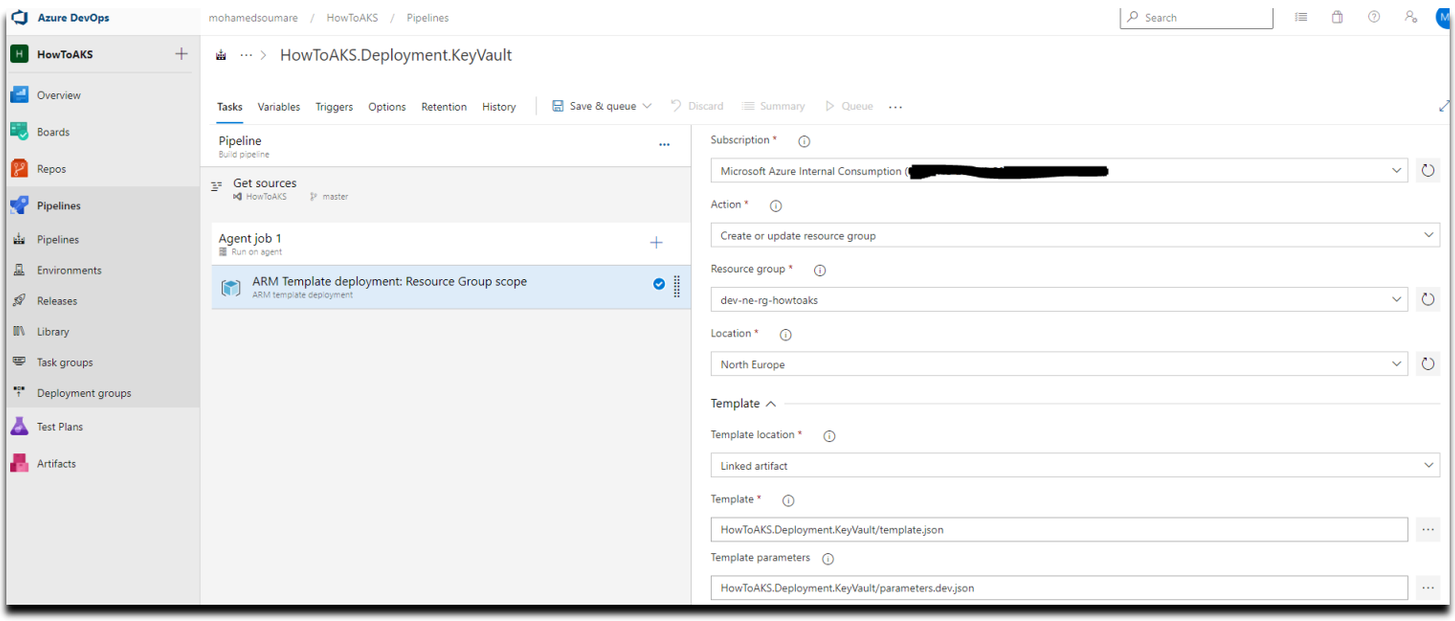
Please find the reusable ARM templates in the following GitHub repository.

Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/0-keyvault*

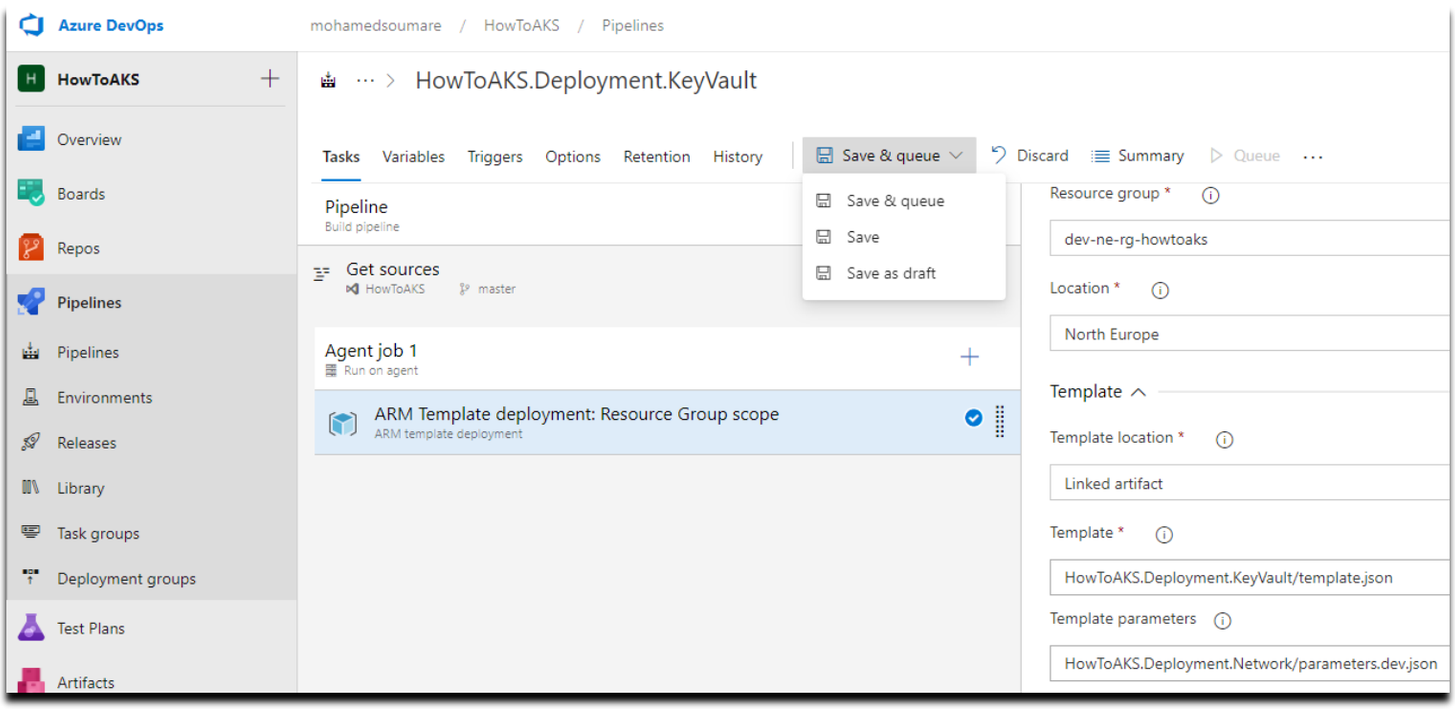
4.1. Deployment with Azure DevOps & CI/CD

Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.KeyVault).

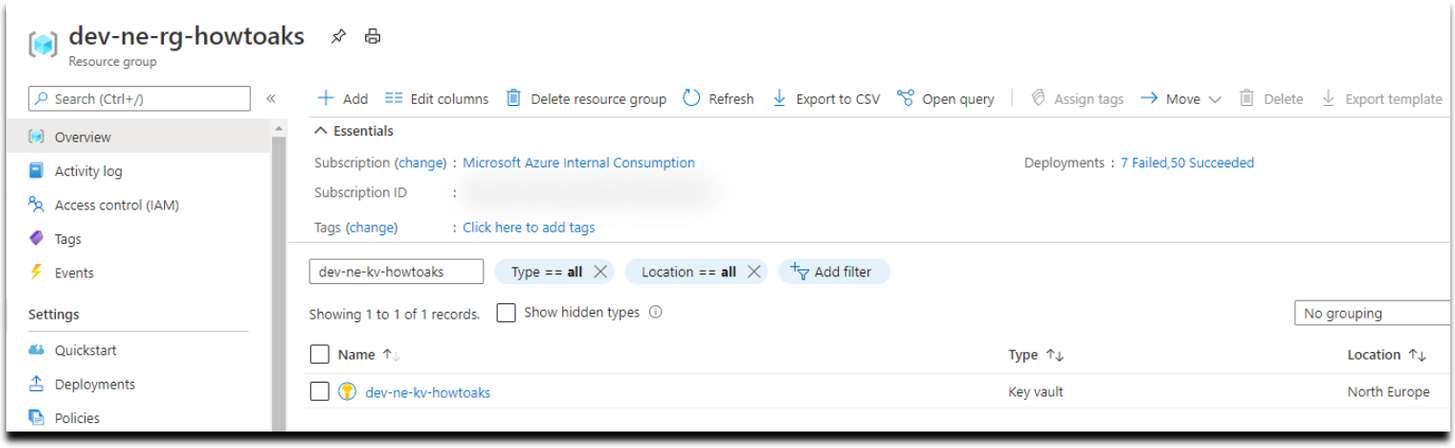
Step 11 > Select your Subscription, Resource Group, Location, Template, and Template Parameters.



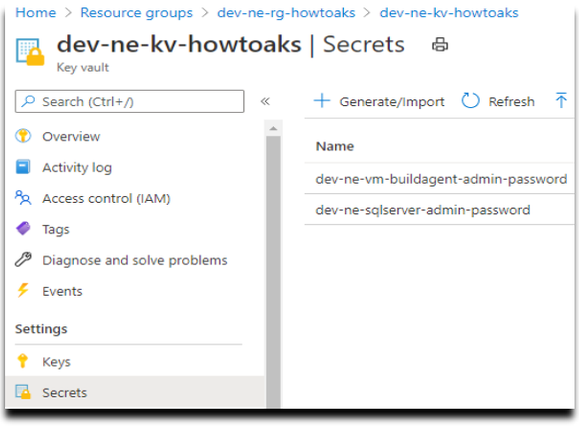
Step 12 > Select Save & Queue to launch the deployment of the Virtual Network.

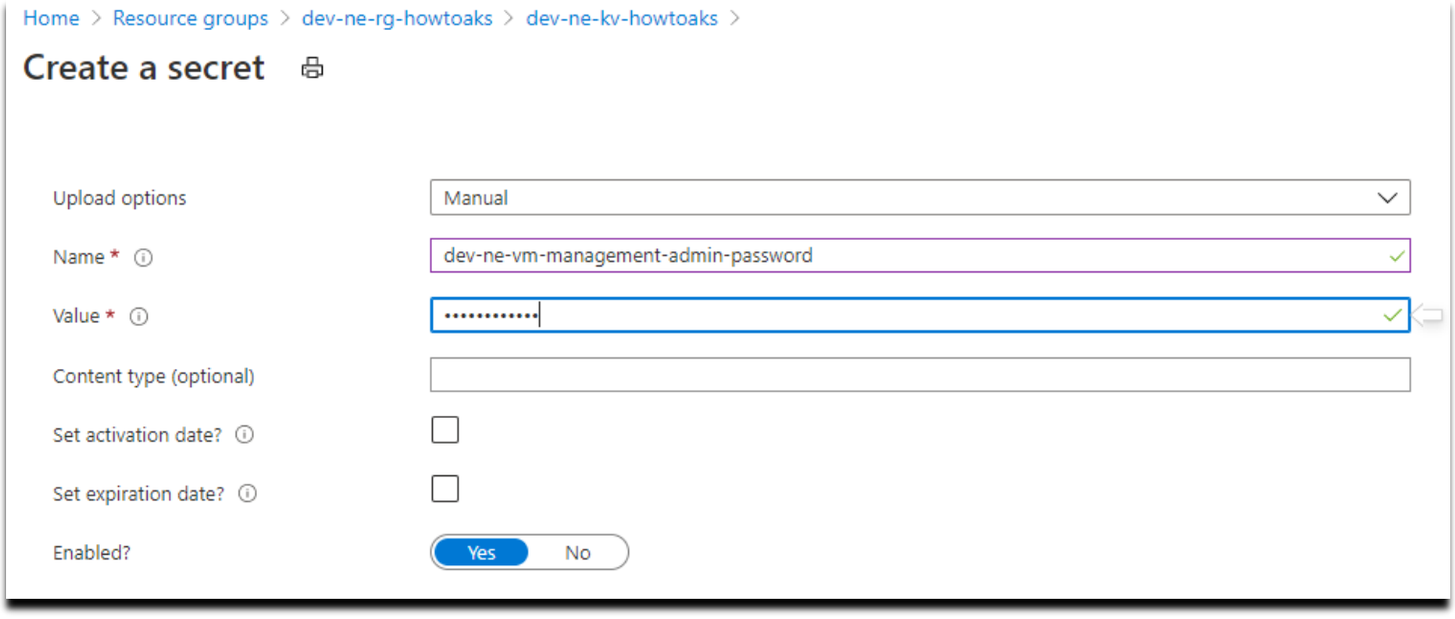


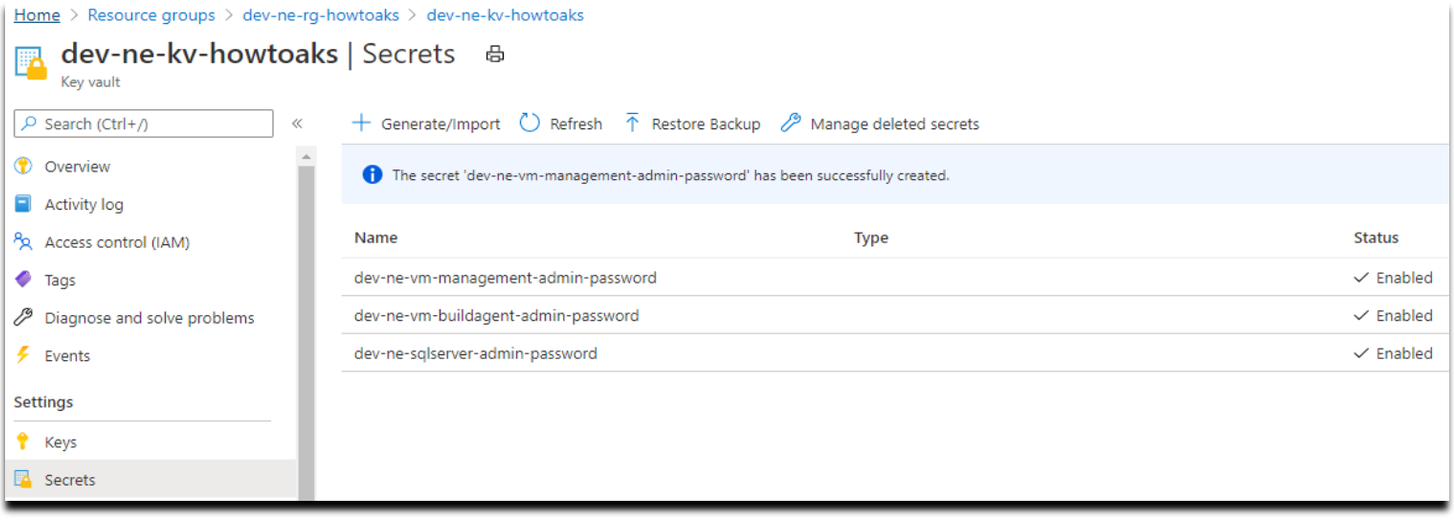
Step 13 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the KeyVault.



Step 14 > Select Secrets > Generate/Import > Create the different passwords listed above as shown in the following screenshots






Home > Resource groups > dev-ne-rg-howtoaks > dev-ne-kv-howtoaks
dev-ne-kv-howtoaks | Secrets
Key vault
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Events
Settings
Keys
Secrets
Generate/Import
Refresh
Restore Backup
Manage deleted secrets
The secret 'dev-ne-vm-management-admin-password' has been successfully created.
Name
Type
Status
dev-ne-vm-management-admin-password
Enabled
dev-ne-vm-buildagent-admin-password
Enabled
dev-ne-sqlserver-admin-password
Enabled

5. Build Agent

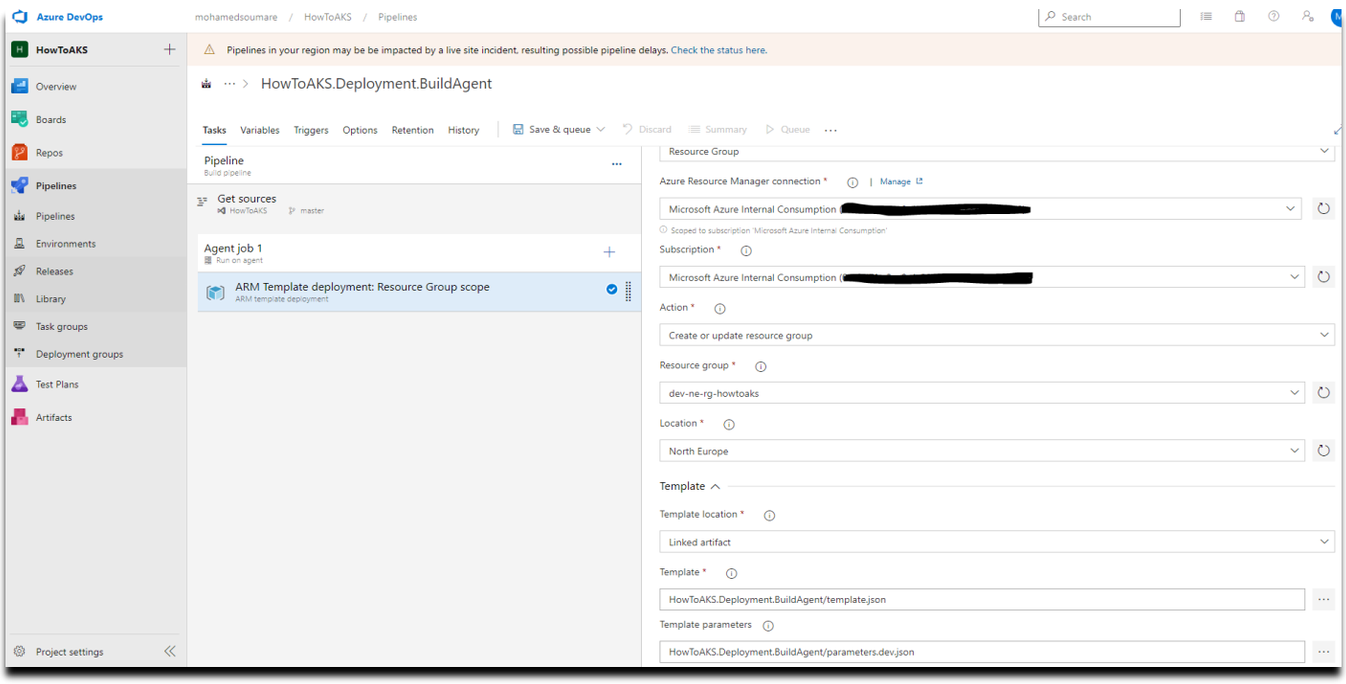
With Azure DevOps, we will be using a private Azure VM build agent to deploy our AKS resources. To avoid managing credentials for our Azure DevOps service connection, we will also configure our build agent with a Managed Identity in Azure AD, so that it can get Azure AD tokens and grant access to any Azure resource that supports Azure AD. Please find the reusable ARM templates in the following GitHub repository.

Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/2-buildagent*

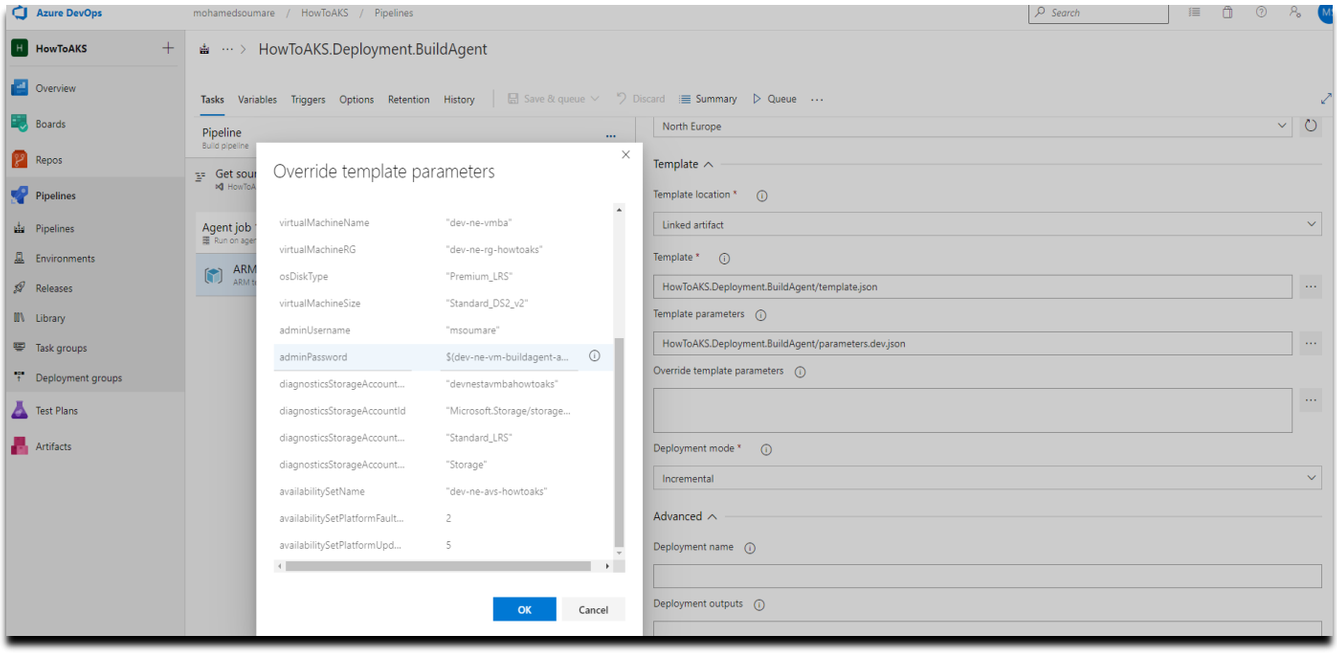
5.1. Deployment with Azure DevOps & CI/CD

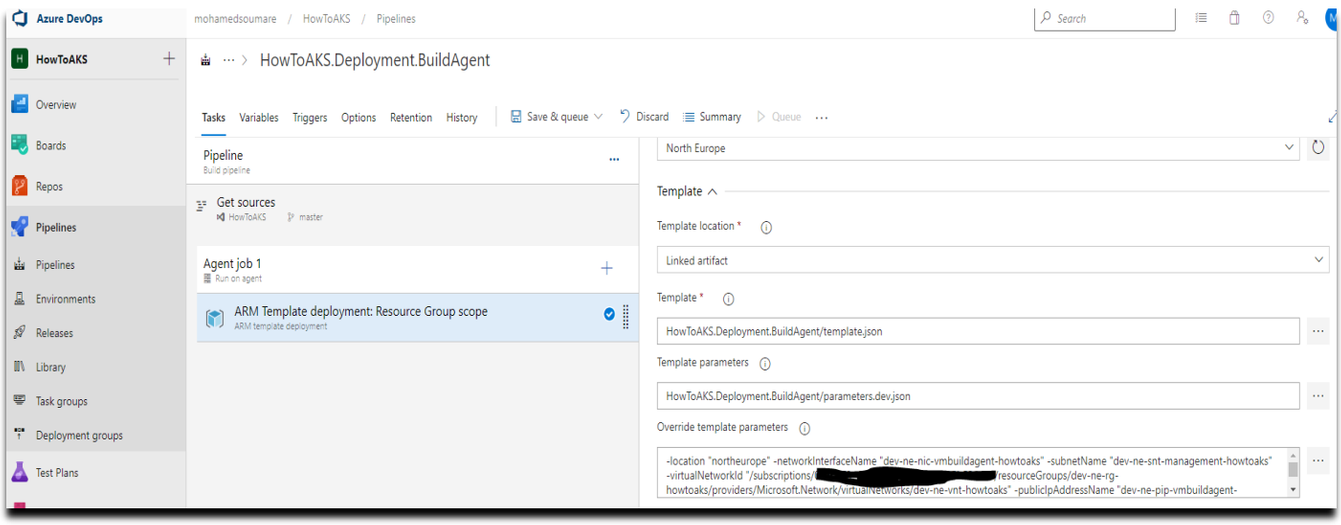
Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.BuildAgent).

Step 11 > Select your Subscription, Resource Group, Location, Template, and Template Parameters.

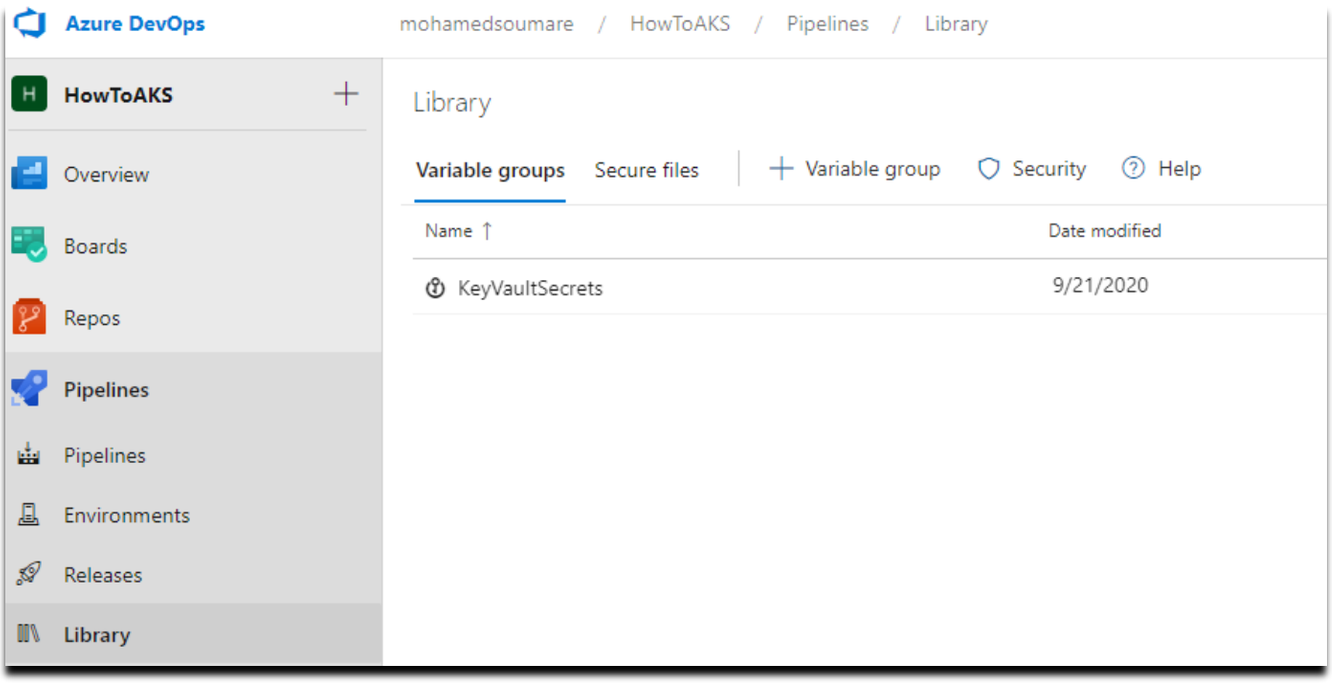


Step 12 > Update the Build Virtual Machine Admin password by overriding the template parameter password with the variable $(dev-ne-vm-buildagent-admin-password). This variable corresponds to a secret that we created earlier in the KeyVault.

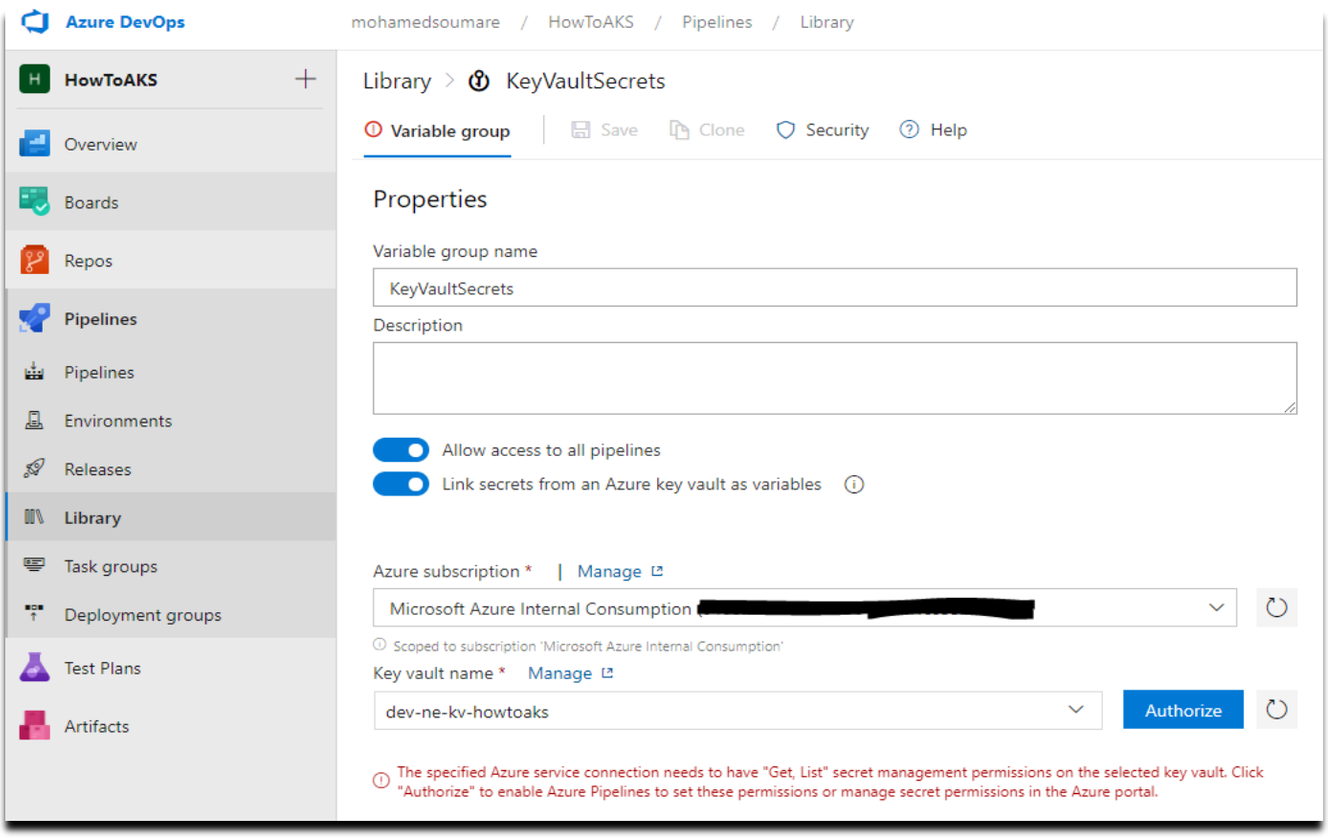




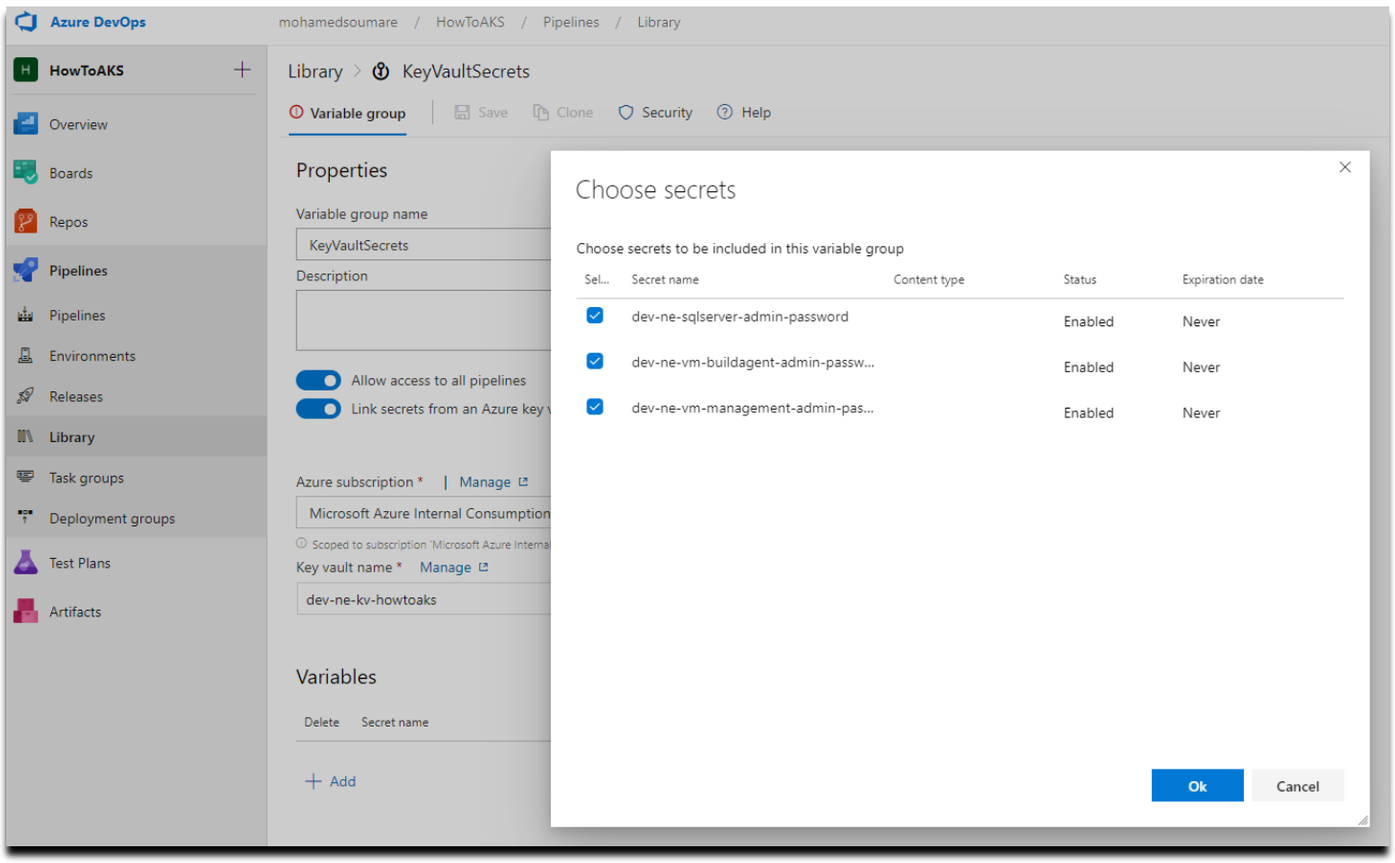
Step 13 > Select Library > Create a Variable Group that will contain our variables/secrets we created in the KeyVault section > Link the secrets in the KeyVault as variables.

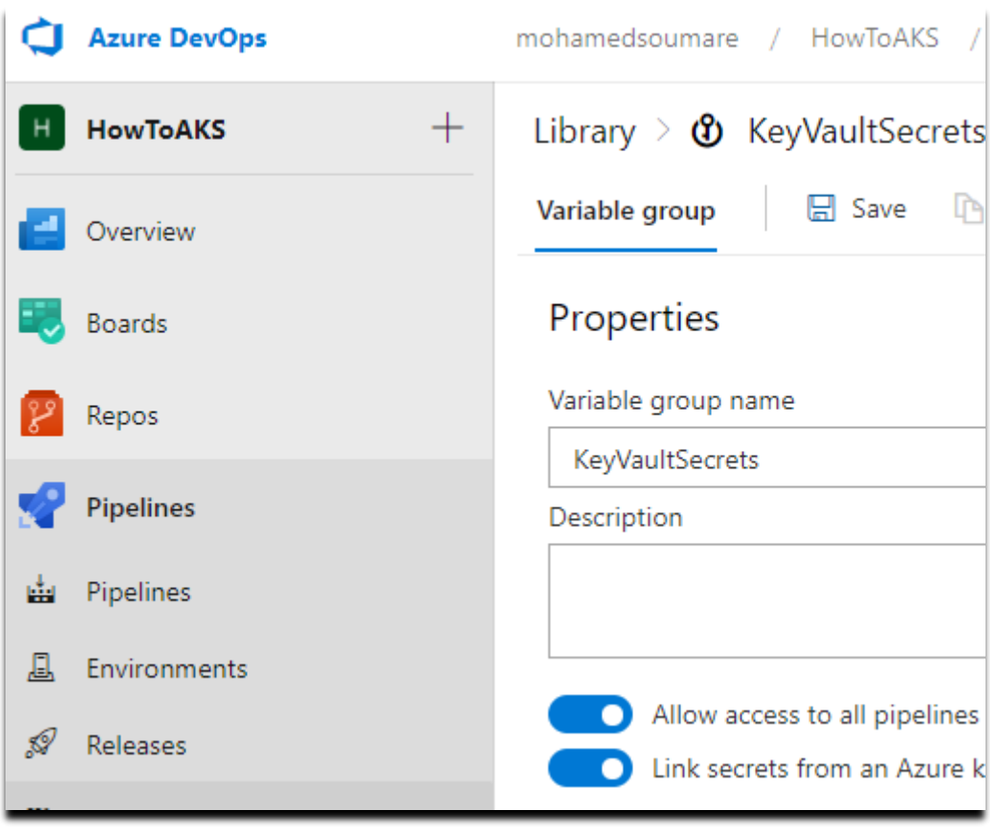


Step 14 > Select your Azure Subscription or an existing Service Connection > Select the KeyVault we created earlier and click Authorize to give Azure DevOps "Get & List" permissions on the KeyVault.

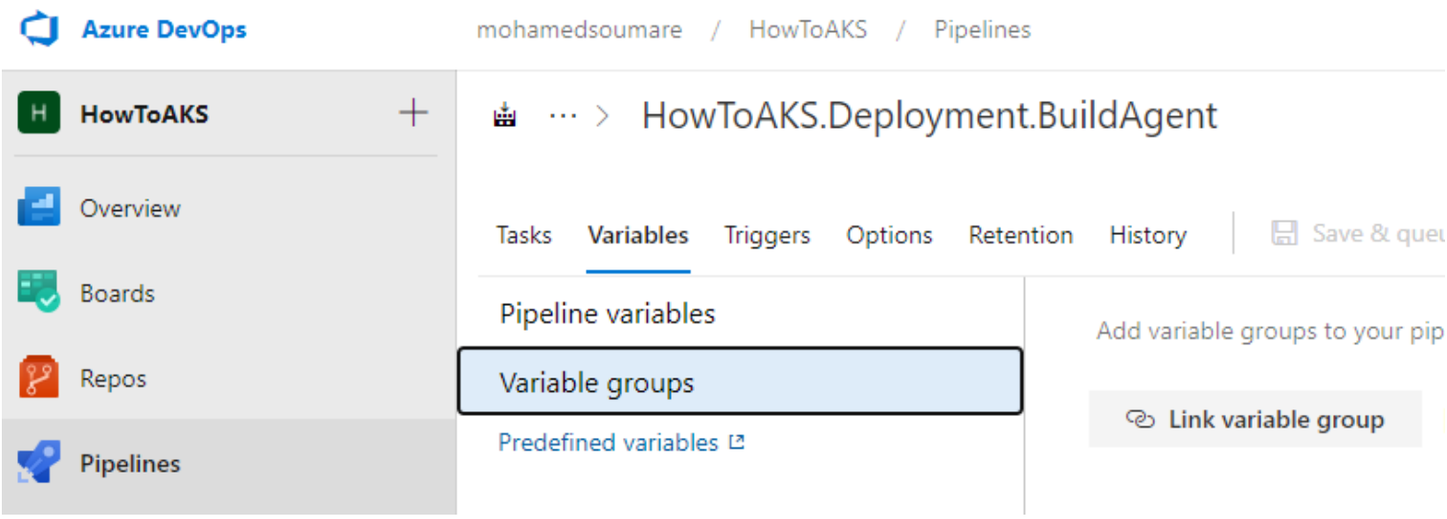


Step 15 > Add the variables by click on the + (Plus sign) next Add in the Variables section > Check all 3 variables and click OK > Select Save.

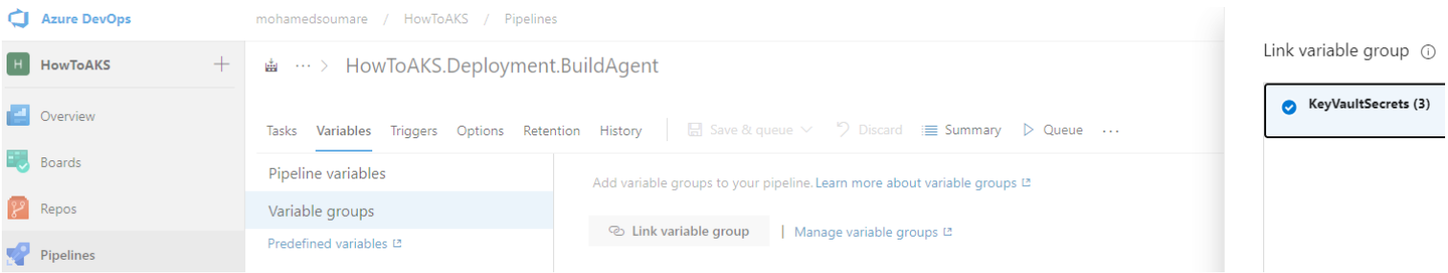




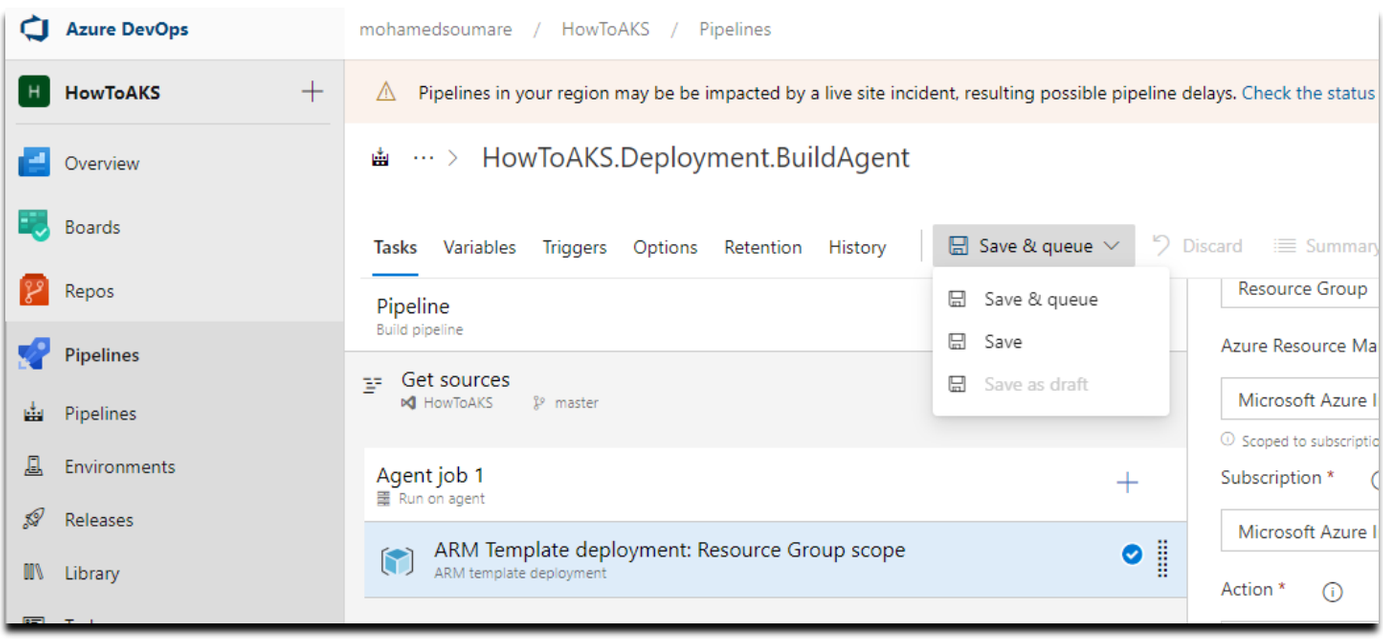
Step 16 > Select Pipelines > Open the Build Agent Pipeline > Select Variables > Variable groups.



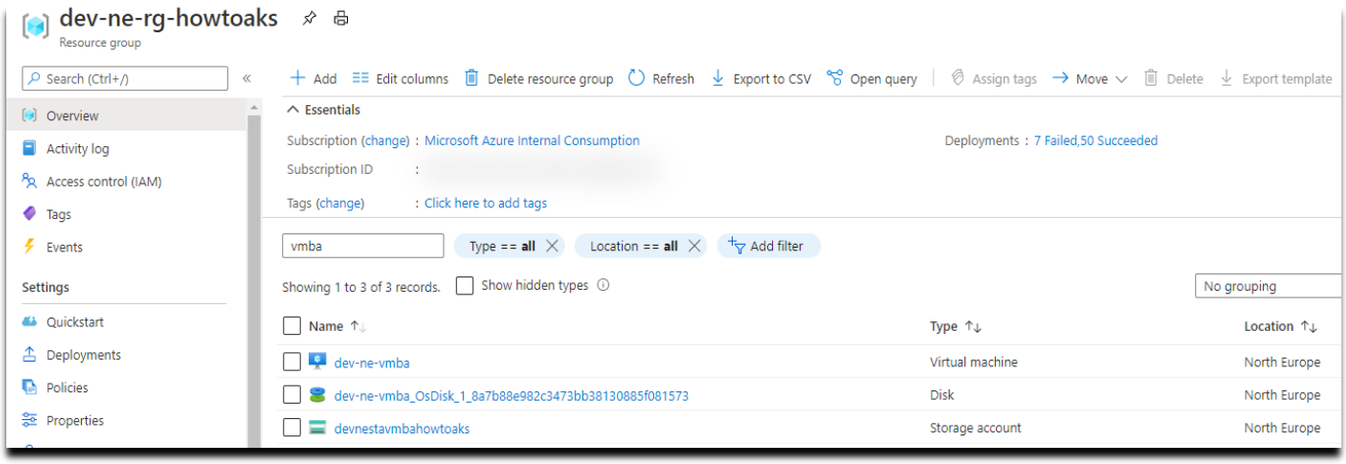
Step 17 > Select Link variable > Check the Variable Group we had created earlier > Select Link.



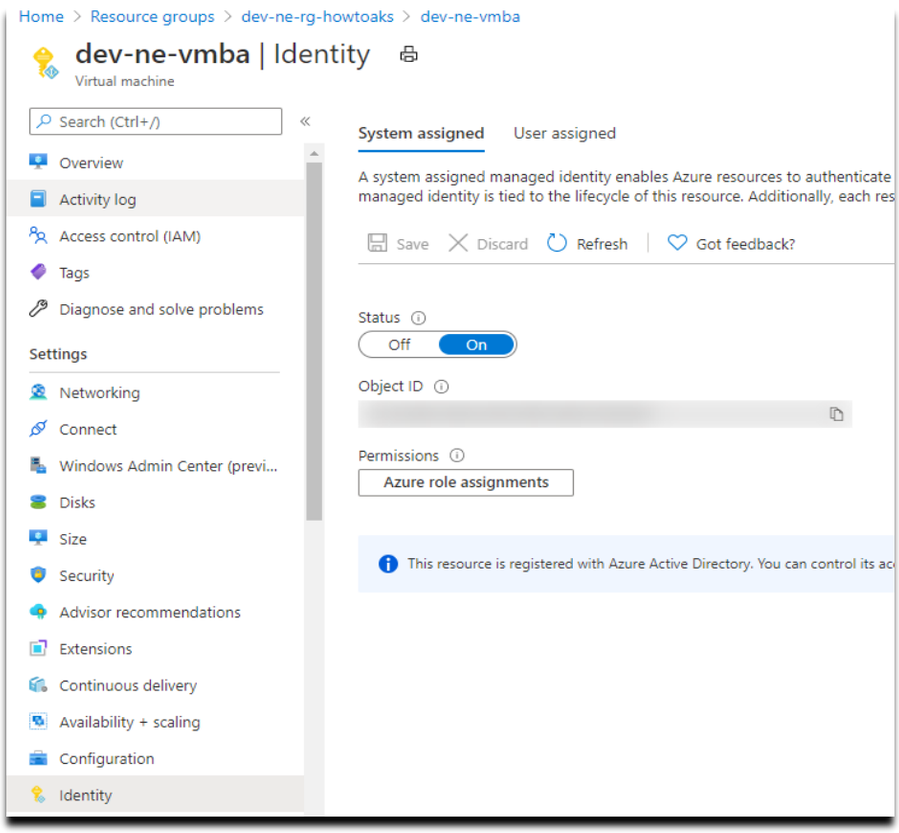
Step 18 > Save & Queue a new build to launch the deployment of the Build Machine.



Step 19 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the Build Machine.



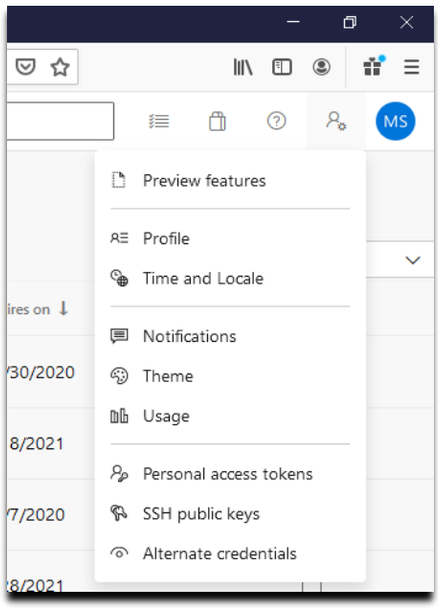
Step 20 > Select Identity > You will see the Build Machine is setup with a System Assigned Managed Identity.



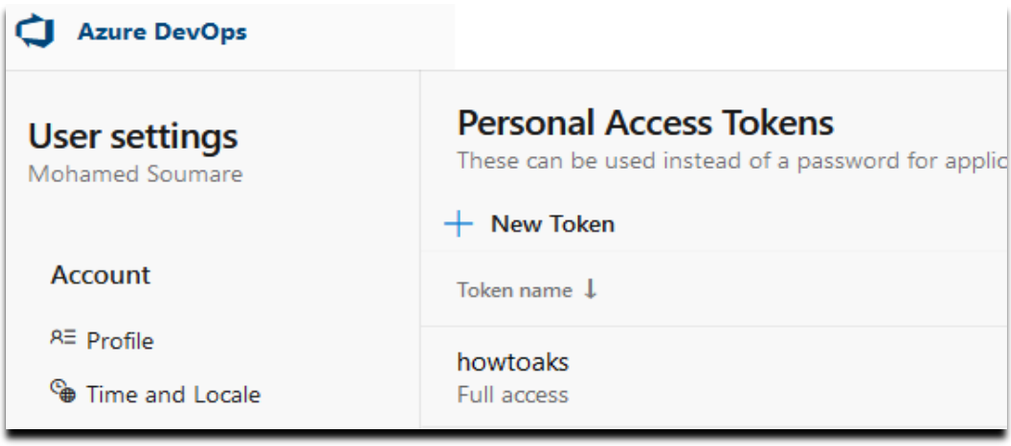
5.2. Generate a Personal Access Token

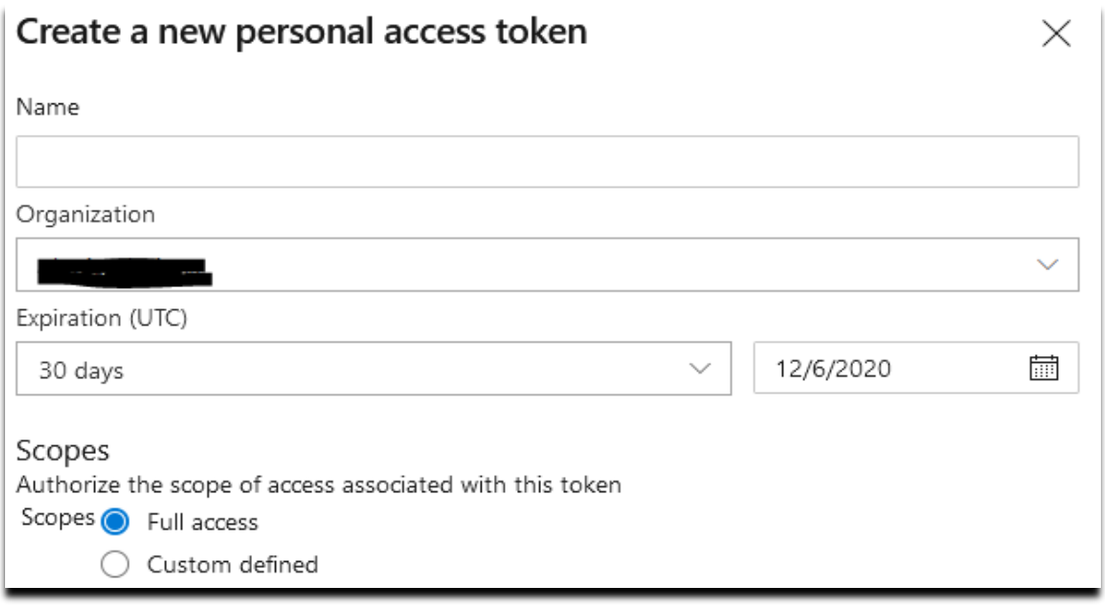
After deploying the Build virtual machine, we will need to generate a Personal Access Token which we will use to setup the actual Build Agent.

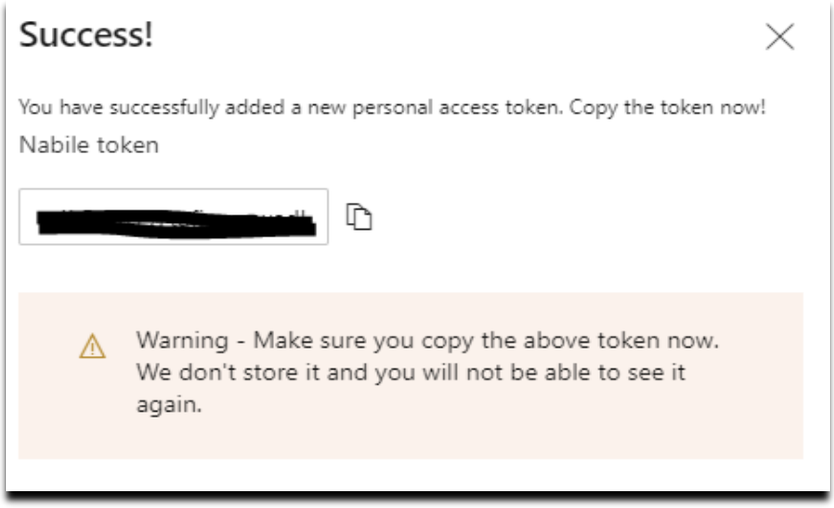
Step 1 > Navigate to Azure DevOps > Profile > Personal Access Token > New Token.



Step 2 > Select New Token and fill out the form and click Create (Remember to copy the access token somewhere, because you will not have access to it again after creation)



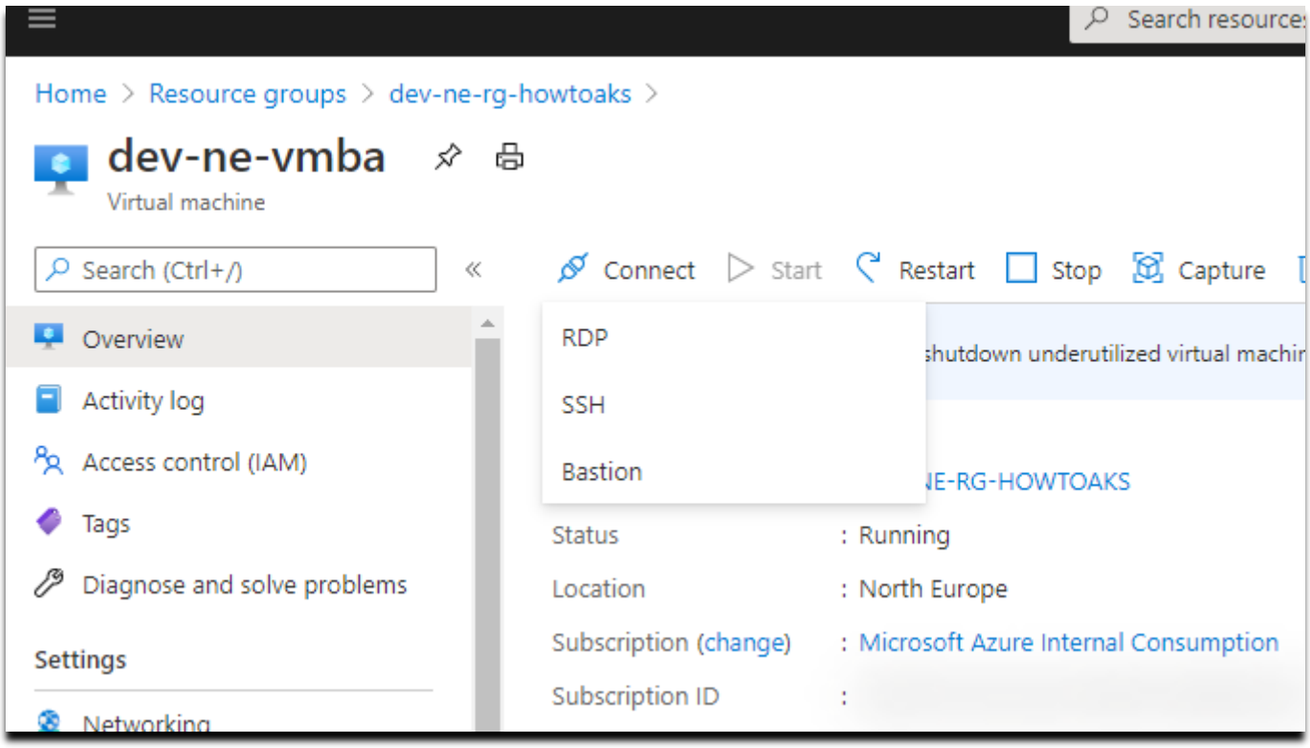




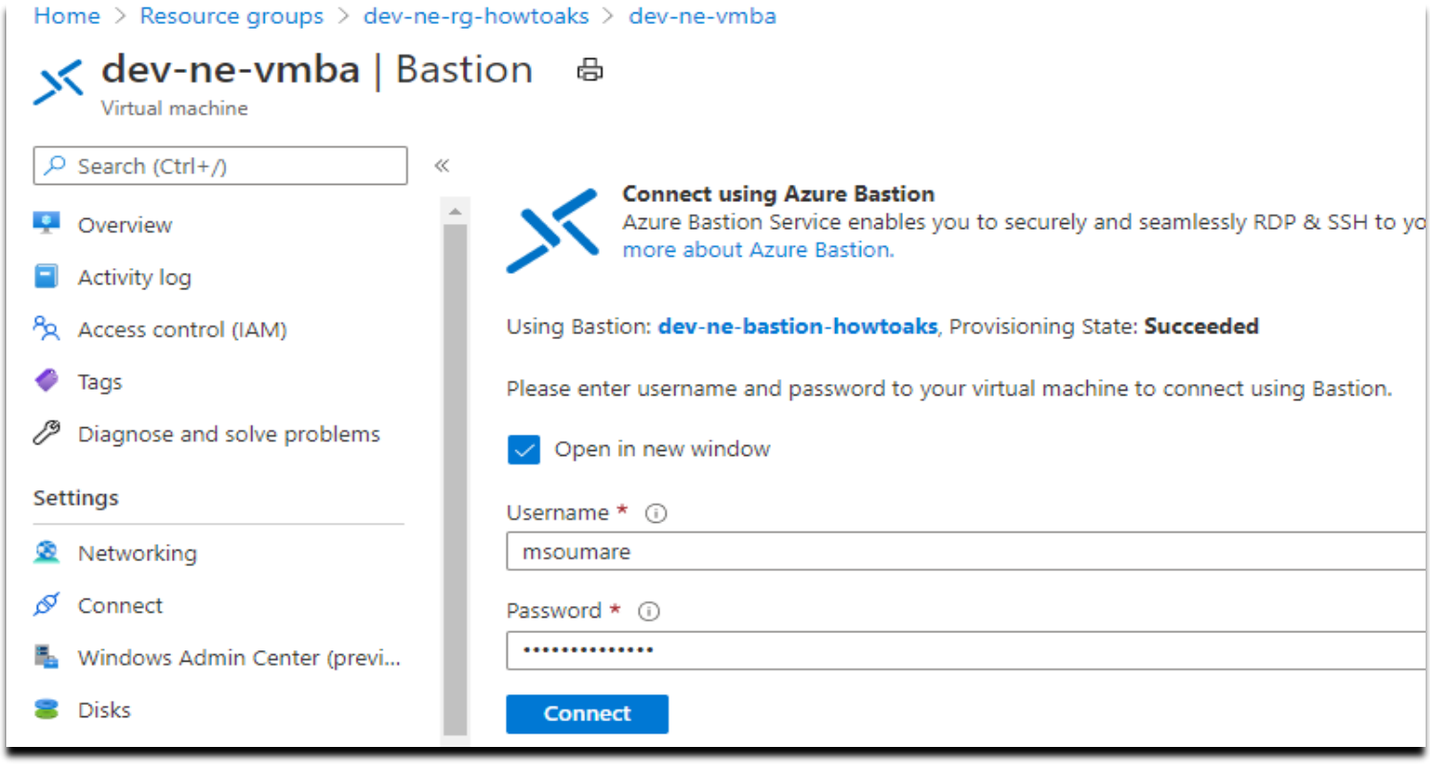
5.3. Download and Install the Agent

Since we want to install the agent in the Azure Build virtual machine, let's navigate to the Azure Portal and connect to the virtual machine using Azure Bastion.

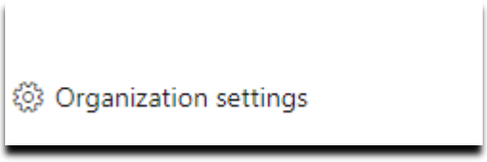
Step 1 > Select the Build Virtual Machine > Connect > Bastion.



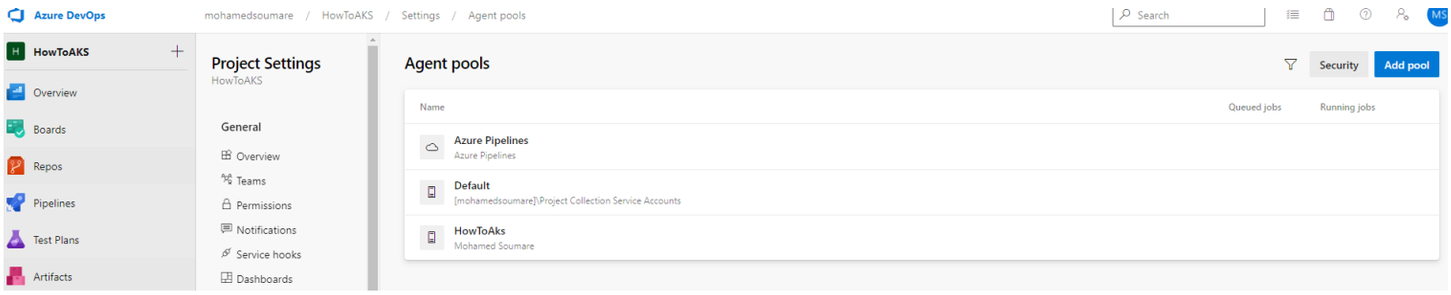
Step 2 > Select the Build Virtual Machine > Connect > Bastion.



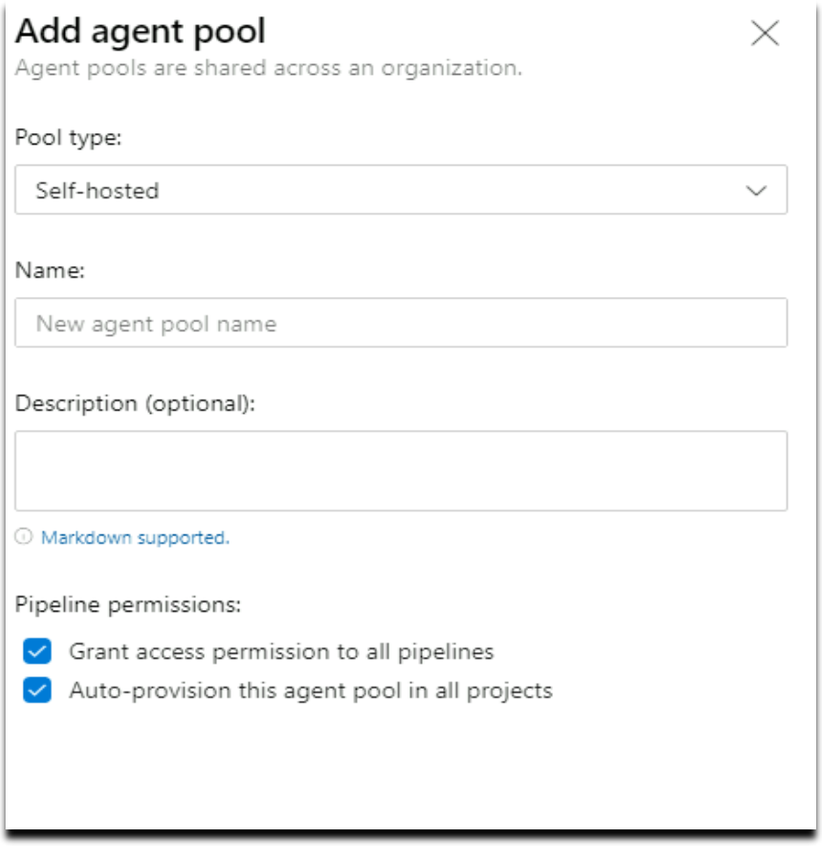
Step 3 > Navigate to Azure DevOps from the Build Machine > Click on Organization Settings at the bottom left.



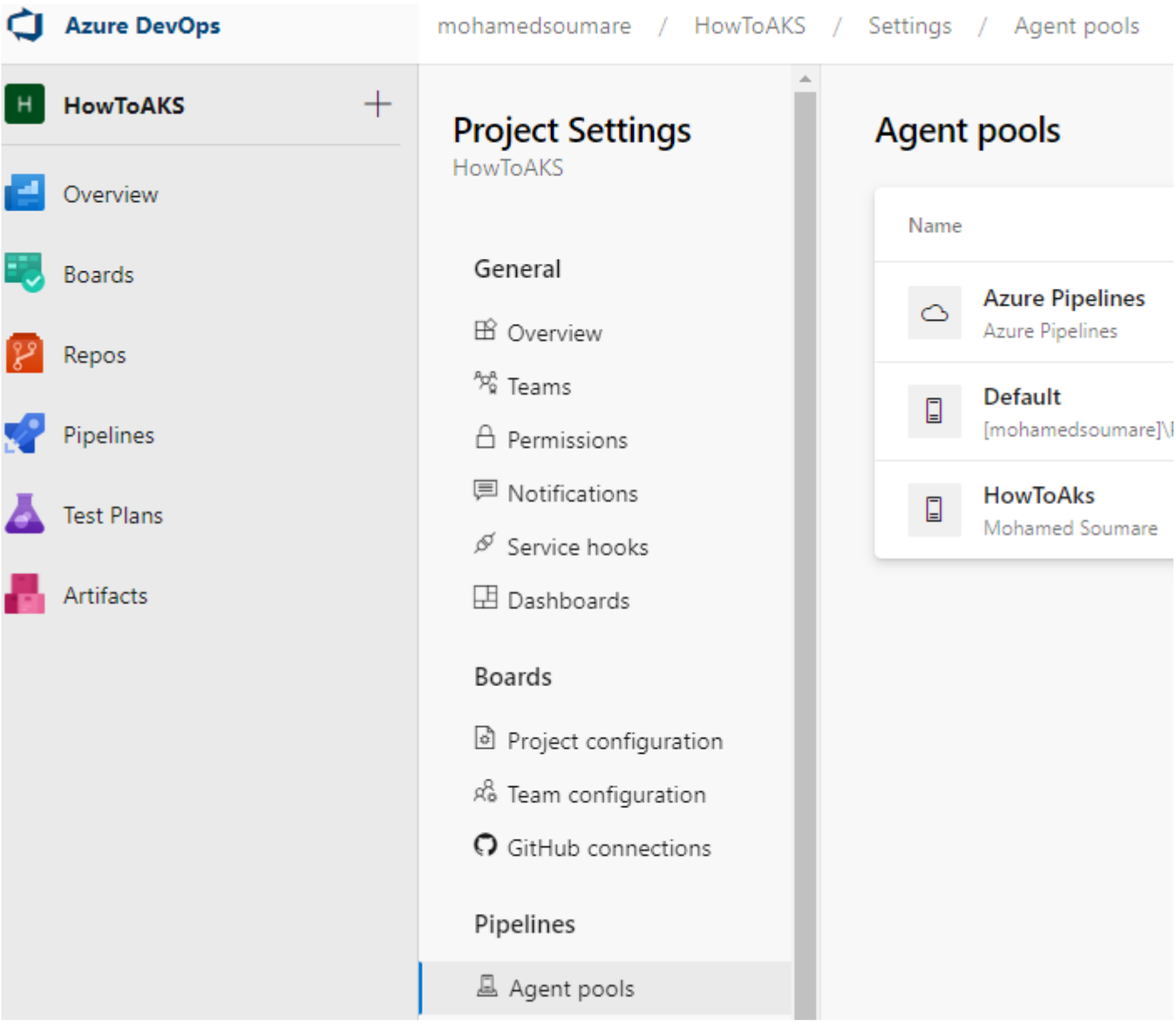
Step 4 > Select Agent Pools > Click on Add Pool.



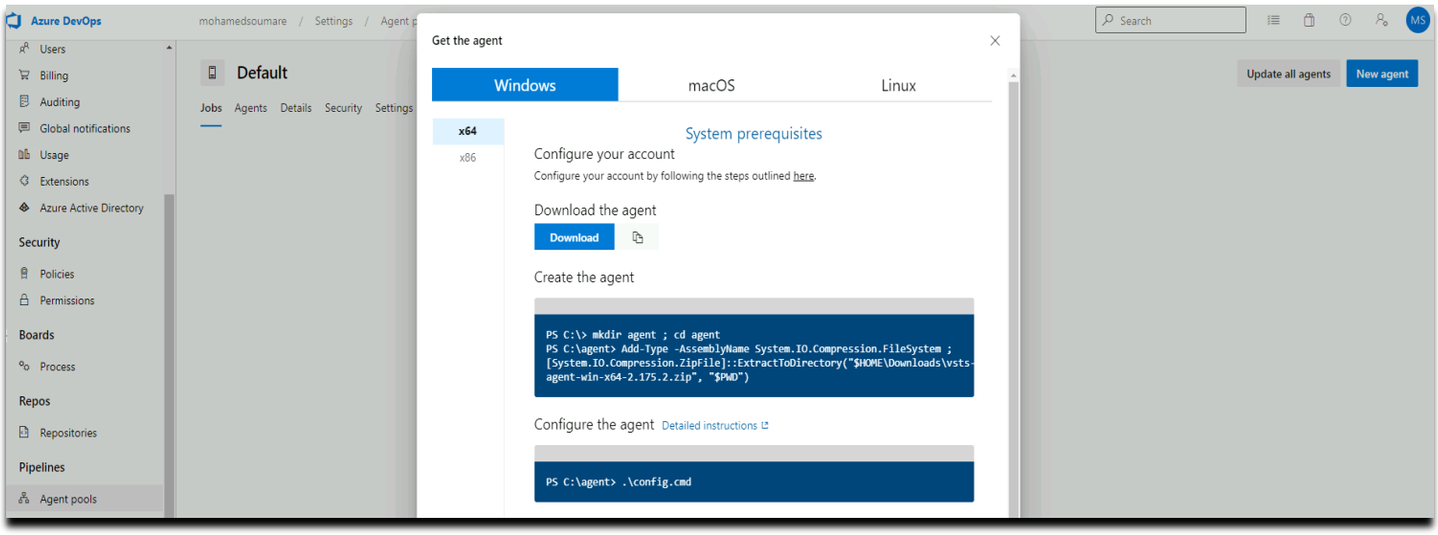
Step 5 > Select Self Hosted > Give a name to the Agent Pool > Click on Create.



Step 6 > Select Agent Pools > Click on the newly created Agent Pool.



Step 7 > Click on New Agent > Download the agent and follow the instructions to set it up using the new Agent Pool name we defined earlier.



5.4. Install Docker Enterprise (You can use the trial version for this demo)

Since our Build Machine is a Windows Server 2019, we need to install Docker Enterprise for creating the container images to be pushed to our Azure Container Registry. Please navigate to https://hub.docker.com/editions/enterprise/docker-ee-server-windows and follow along the instructions.

5.5. Install Docker Compose

To allow our Build Machine to create the Docker-Compose file, please navigate to https://docs.docker.com/compose/install and follow along the instructions.

1. Start PowerShell as an Administrator
2. In PowerShell, since GitHub now requires TLS1.2, run the following:
3. [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12

   Then run the following command to download the current stable release of Compose (v1.27.4):

   Invoke-WebRequest "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-Windows-x86_64.exe" -UseBasicParsing -OutFile $Env:ProgramFiles\Docker\dock

5.6. Install Azure CLI

To invoke Azure CLI commands from our Build pipeline, we need to setup this feature in our Build Machine. Please navigate to https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli and follow along the instructions.

5.7. Install Chocolatey

For better Package Management we need to install Chocolatey. Please navigate to https://chocolatey.org/install and follow along the instructions.

5.8. Install sed

In our Kubernetes Deployment & Service manifest files we are replacing the static version of the container build image with our DevOps pipeline build version using SED. Therefore, we need to install it using Chocolatey. Please navigate to https://chocolatey.org/packages/sed and follow along the instructions.

5.9.Install Kube client

To deploy our microservices using the Kubernetes deployment and service manifests by our DevOps pipeline, we invoke kubectl. Hence, we need to install it either using PowerShell, Chocolatey or Scoop etc. Please navigate to https://kubernetes.io/docs/tasks/tools/install-kubectl/ and follow along the instructions.

5.10. Install Kubelogin

Because our cluster is integrated with Azure AD, when we try to connect to AKS using kubectl, we are prompted to authenticate with a device login. This presents a challenge for our DevOps Build pipeline, so we need to be able to authenticate non-interactively.

To solve this problem, we need to install Kubelogin which allows us to use the **Managed identity of our Build Agent** as the current user Service Principal to authenticate on behalf of the Build Machine. Please navigate to https://github.com/int128/kubelogin/releases/tag/v1.22.0 and download the latest version of Kubelogin, then to install it, navigate to https://github.com/Azure/kubelogin and follow along the instructions.

After installing Kubelogin, create a folder under the C drive (C:\KubeLogin)
Copy the executable kubelogin.exe in that folder. We will make use of this executable during the release of our container application in AKS.

6. Management Box

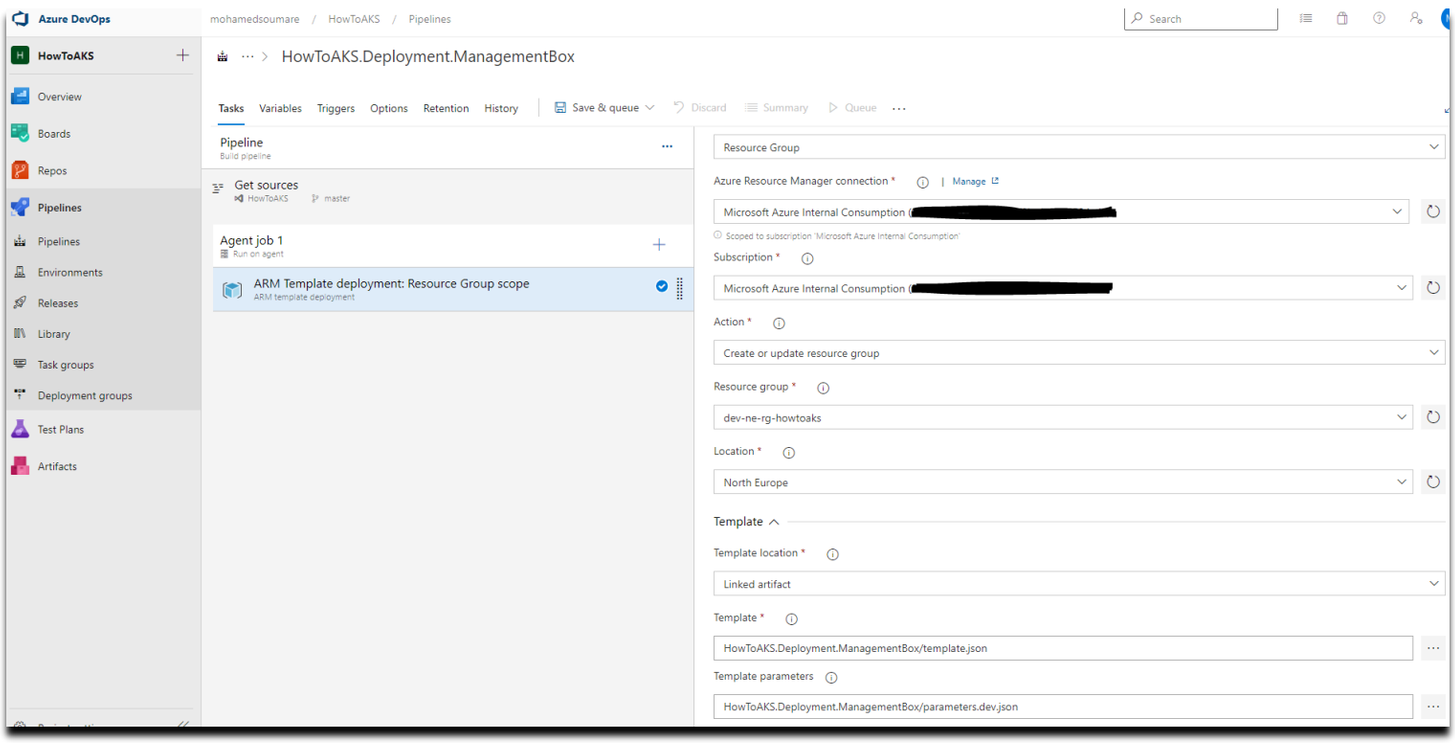
In case we want to connect to our private network and work with the resources with private IP addresses, we will deploy a Management Box. To keep our network secure and prevent any public access to the Management Box, we will build the virtual machine with a private IP Address. Then we will set up Network security rules that will only allow inbound calls from the Azure Bastion Subnet to the Management Subnet. Please find the reusable ARM templates in the following GitHub repository.

Infrastructure as Code - https://github.com/msoumare/HowToAKS/tree/main/3-managementbox

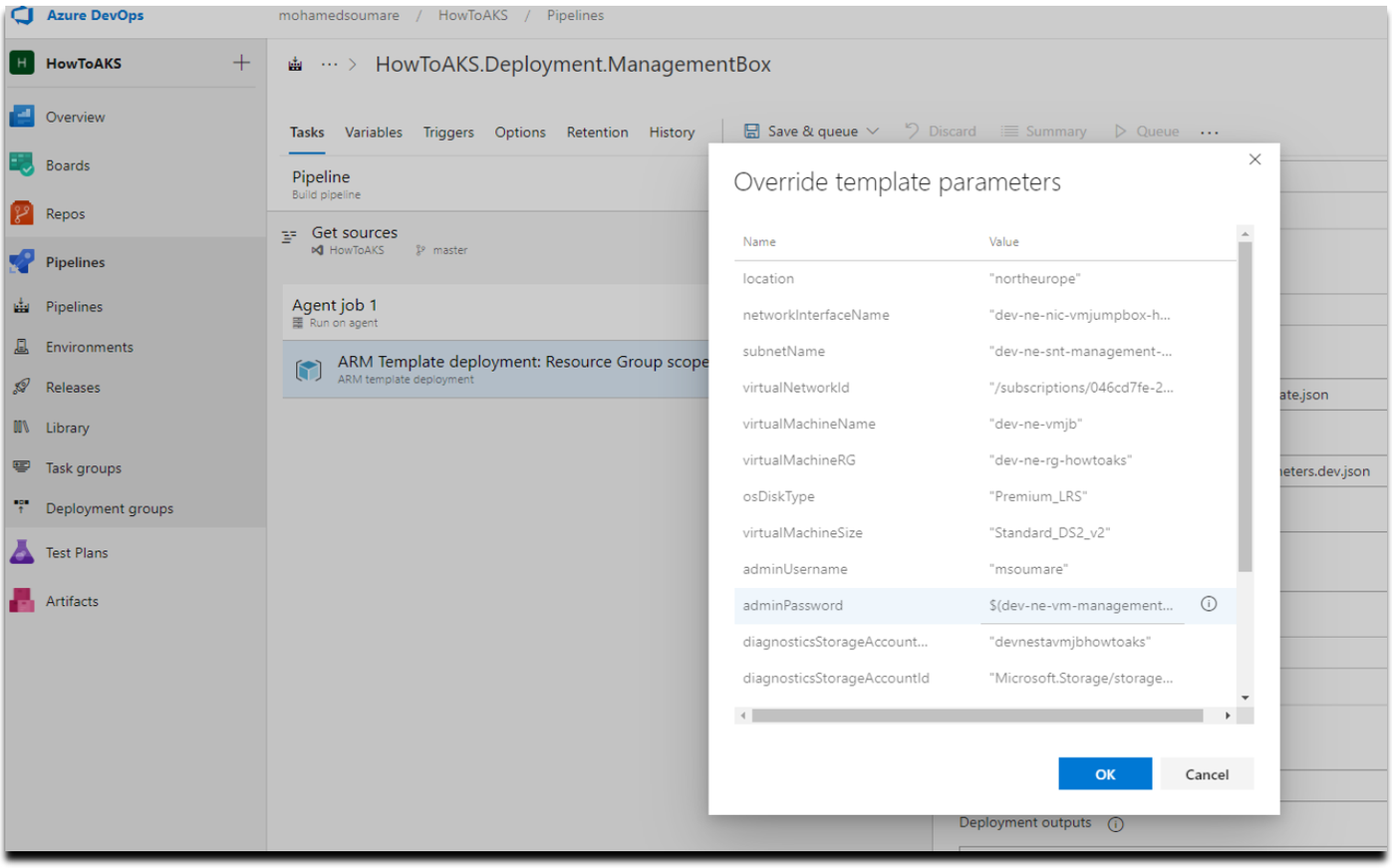
6.1. Deployment with Azure DevOps & CI/CD

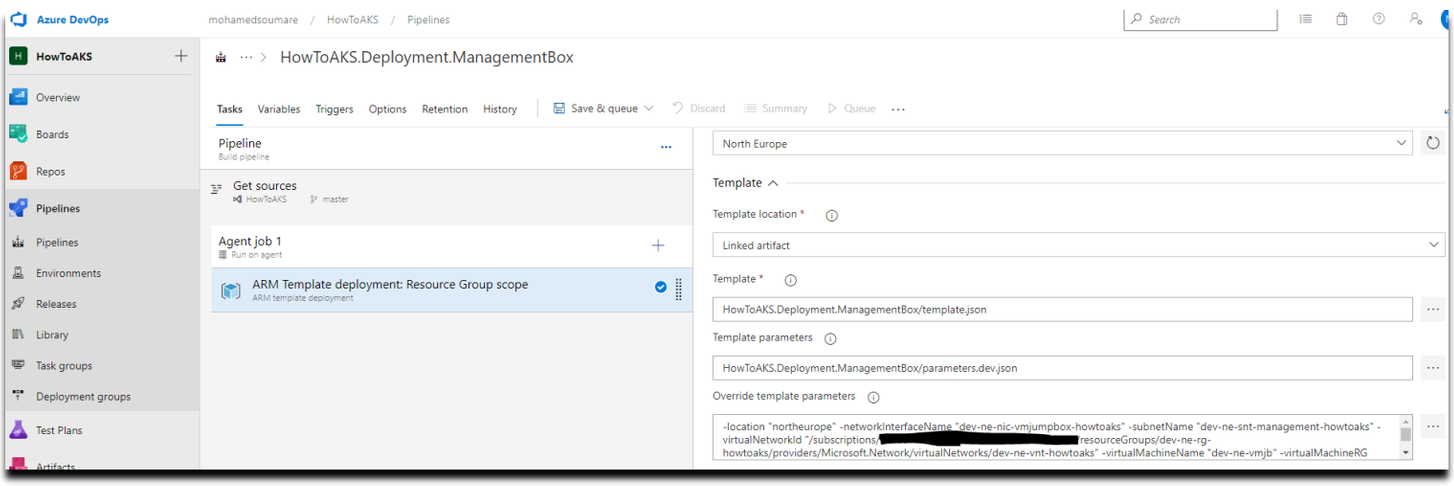
Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.ManagementBox).

Step 11 > Select your Subscription, Resource Group, Location, Template, and Template Parameters.

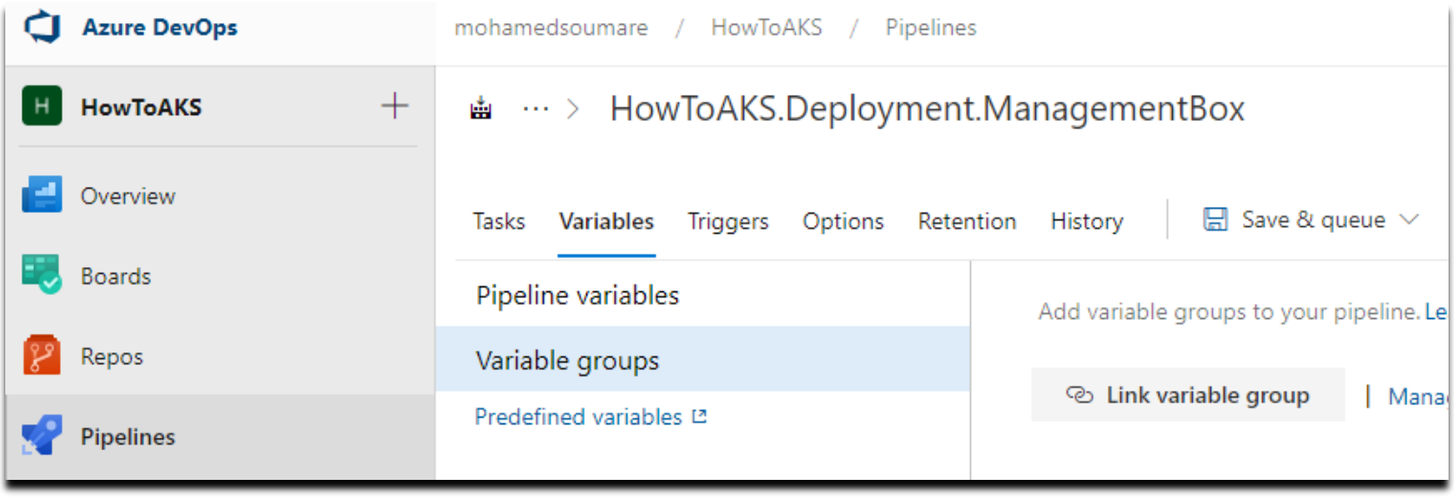


Step 12 > Update the Management Box Virtual Machine Admin password by overriding the template parameter password with the variable $(dev-ne-vm-management-admin-password). This variable corresponds to a secret that we created earlier in the KeyVault.

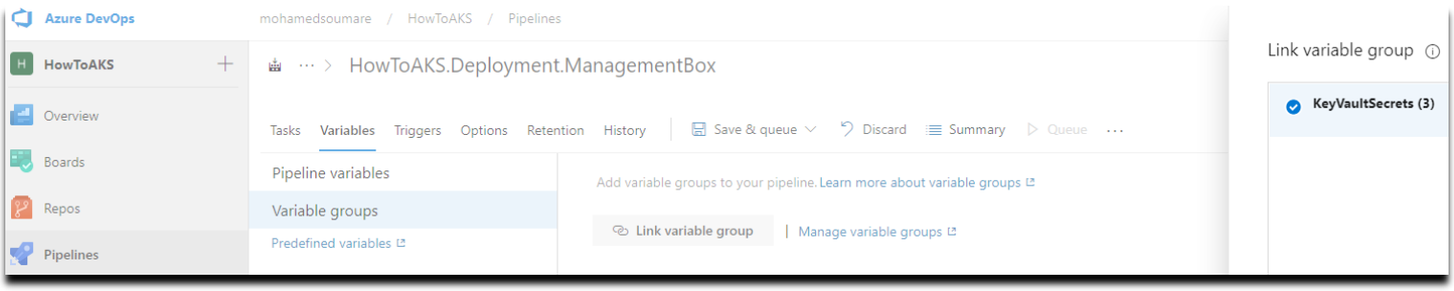




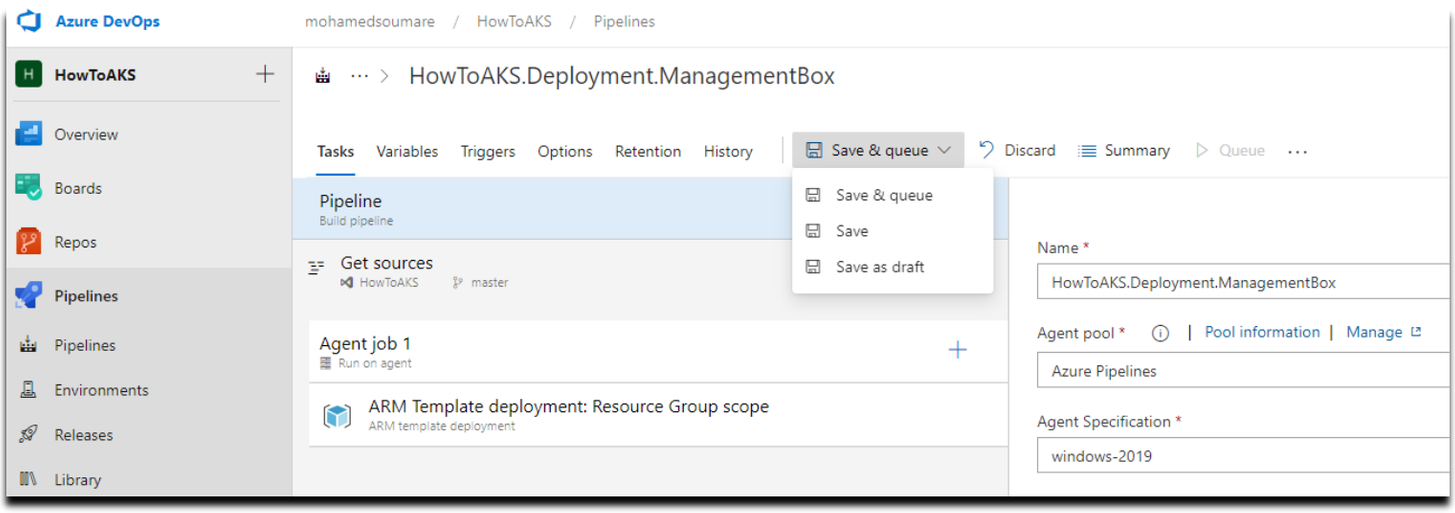
Step 13 > Select Variables > Variable groups.



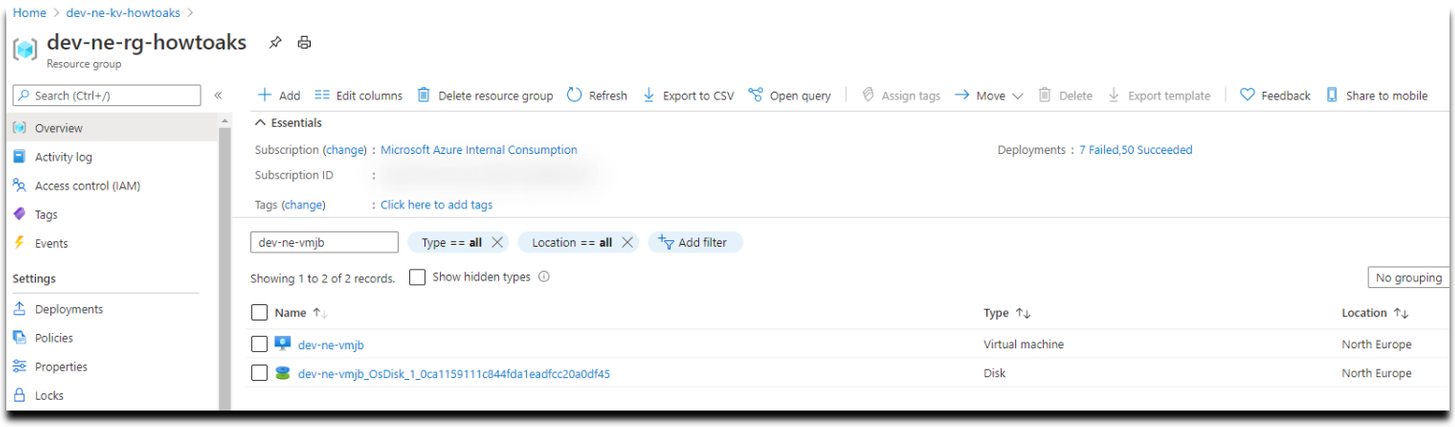
Step 14 > Select Link variable > Check the Variable Group we had created earlier > Select Link.



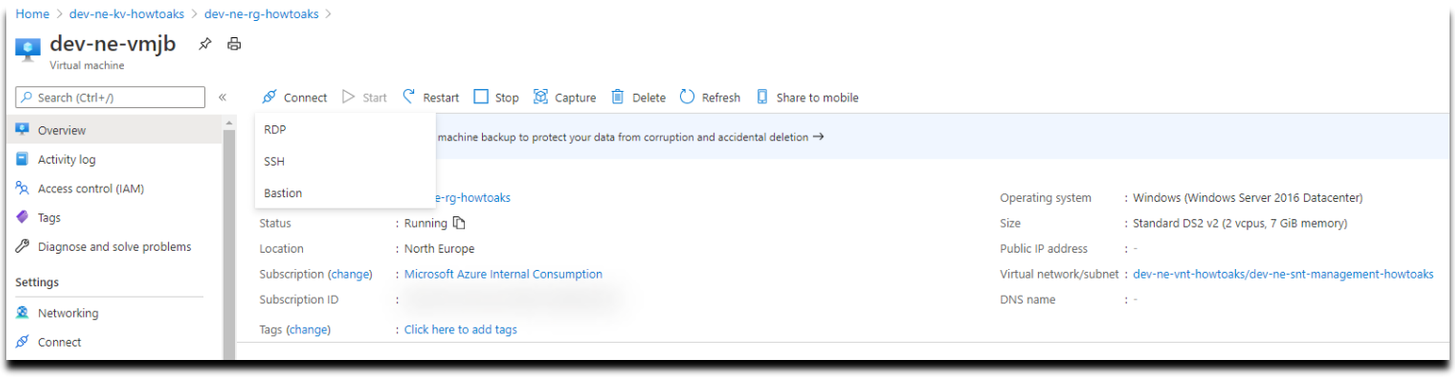
Step 15 > Save & Queue a new build to launch the deployment of the Build Machine.



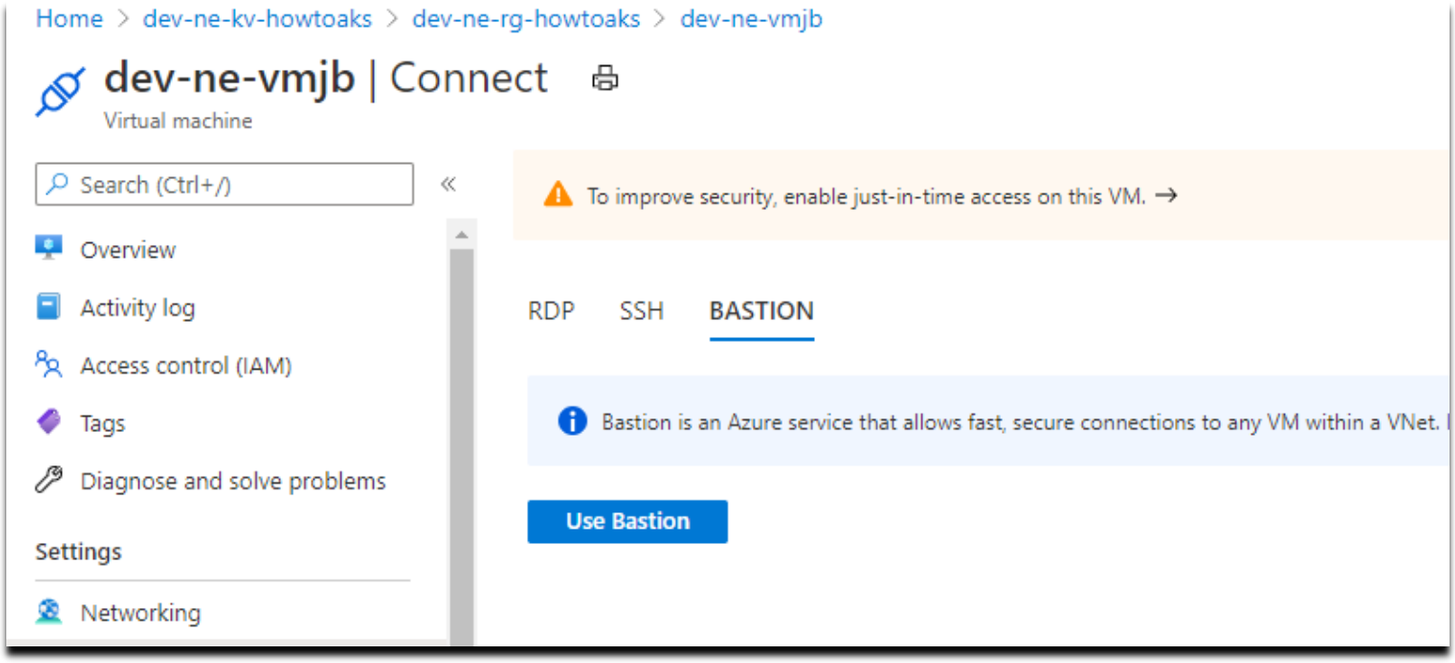
Step 16 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the Magement Box Machine.

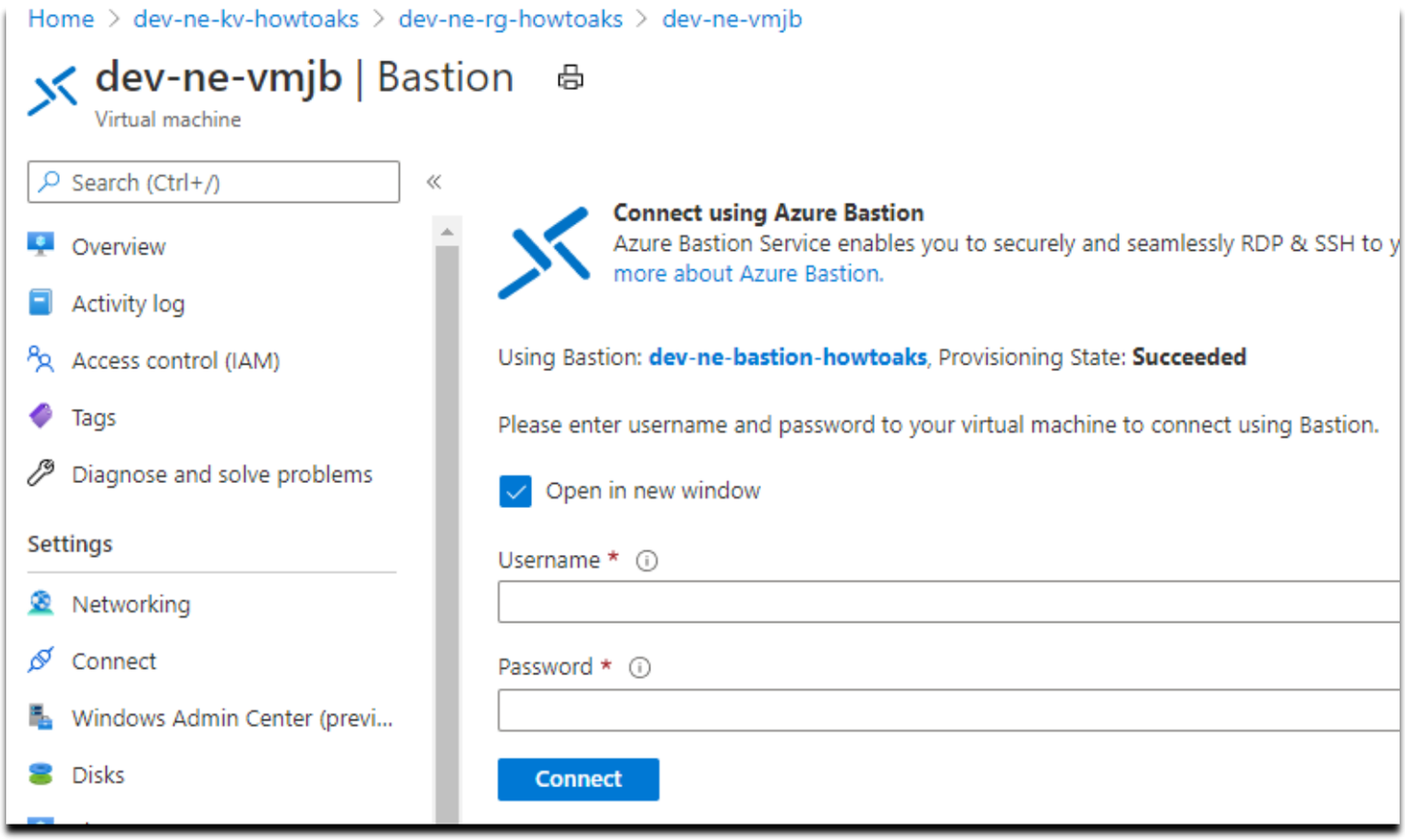


Step 17 > You will notice the Virtual Machine does not have a public IP. This is done purposely to only allow connection from Azure Bastion subnet > Select Connect Bastion.



Step 18 > Select Use Bastion > Connect with the username we defined in the template ARM and the password we created in the KeyVault.





Once you're connected you will be able to do any management of your other private IP resources from this Virtual Machine.

7. Azure Container Registry

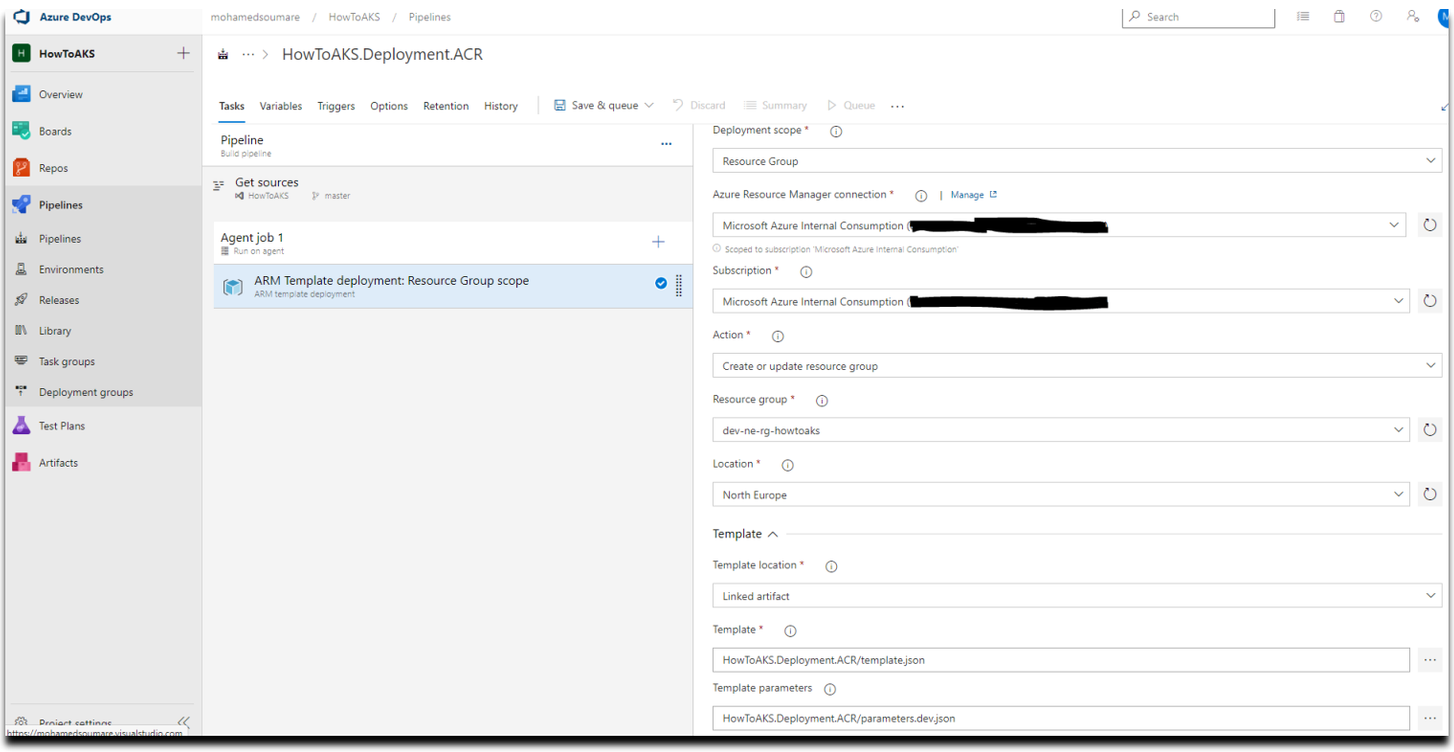
To securely store and access our Docker container images for the applications, we will use Azure Container Registry, which is a managed, private Docker registry service based on the open-source Docker Registry 2.0. Then we will give our AKS cluster the permissions to retrieve our Docker images and deploy them to the cluster. Please find the reusable ARM templates in the following GitHub repository.

Infrastructure as Code - https://github.com/msoumare/HowToAKS/tree/main/4-acr

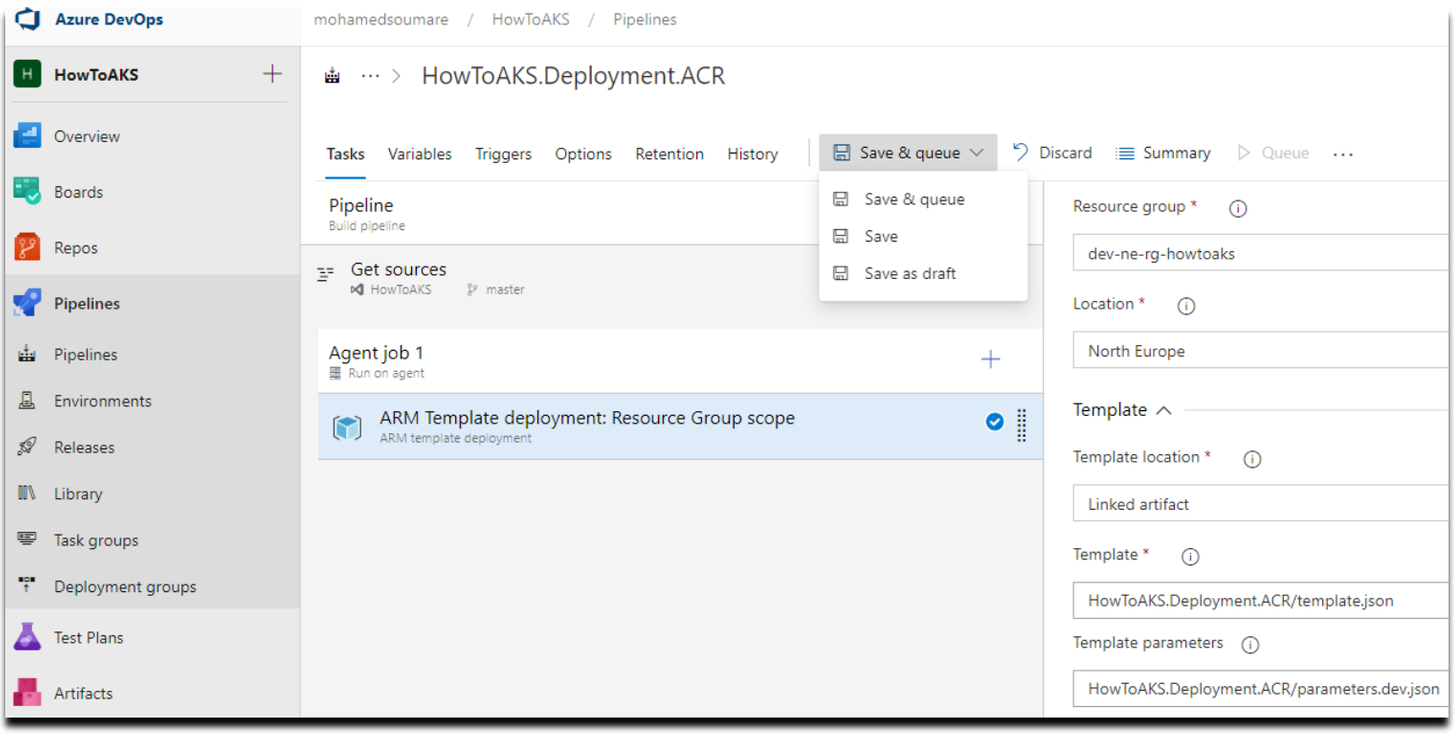
7.1. Deployment with Azure DevOps & CI/CD

Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.ACR).

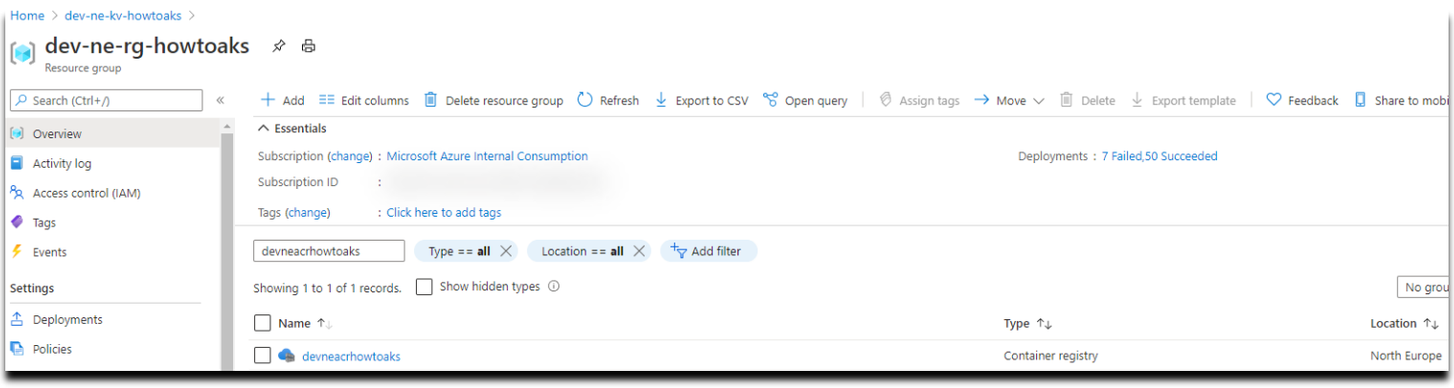
Step 11 > Select your Subscription, Resource Group, Location, Template, and Template Parameters.



Step 12 > Select Save & Queue to launch the deployment of the Container Registry.



Step 13 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the ACR.



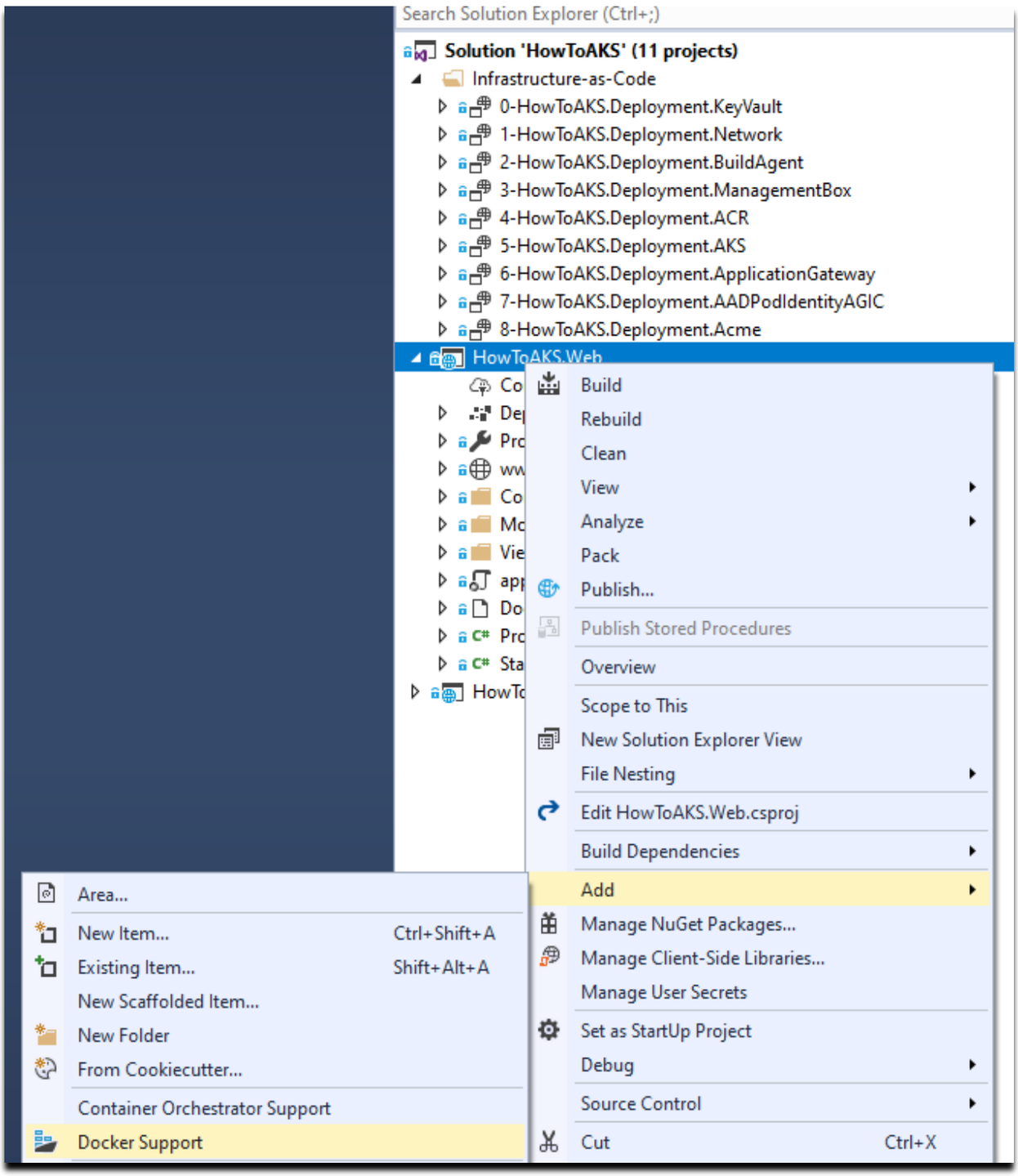
Step 14 > You will notice the ACR Login server address. We will make use of this server address later.

7.2.Generate Docker files

In this tutorial we will be building a Web application with an API Layer connecting to the Data Stores. I will not go into the implementation details of connecting to the SQL Database or Service Bus, because that's not the scope of this paper. I will just demonstrate the Web and API parts of the solution.

We can use a Visual Studio functionality to generate the Docker files as shown in the next pictures (you can also do it with another IDE or notepad). However, you must install Docker as a prerequisite on your machine.

Step 1 > From the example below, we have a solution with 2 .NET Core applications. One Web (Front End) and one Web API (Back End). To create a Docker File, you can simply right click on each project > Click Add > Docker Support.



This will generate a Docker File which we will use to create a container image of our application. If you are working with another code editor that does not have support for Docker, you can also write it yourself as shown below.

DockerFile for the Web App project:

```
FROM microsoft/dotnet:2.1-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY ["./HowToAKS.Web/HowToAKS.Web.csproj", "HowToAKS.Web/"]
RUN dotnet restore "HowToAKS.Web/HowToAKS.Web.csproj"
COPY . .
WORKDIR "/src/HowToAKS.Web"
RUN dotnet build "HowToAKS.Web.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "HowToAKS.Web.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "HowToAKS.Web.dll"]
```

DockerFile for the Web API project:

```
FROM microsoft/dotnet:2.1-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY ["./HowToAKS.WebApi/HowToAKS.WebApi.csproj", "HowToAKS.WebApi/"]
RUN dotnet restore "HowToAKS.WebApi/HowToAKS.WebApi.csproj"
COPY . .
WORKDIR "/src/HowToAKS.WebApi"
RUN dotnet build "HowToAKS.WebApi.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "HowToAKS.WebApi.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "HowToAKS.WebApi.dll"]
```

7.3. Create Container Images

To create the container images from our application we will use Docker Compose with Azure DevOps and CI/CD. Let's build the YAML file that defines all the application containers along with the ports.

As you can see in the following Docker Compose YAML file, the container image `howtoaks-web` runs on the container port 80 internally and forwards the exposed port 80 (Docker file above) on the container to port 80 externally on the Docker host Linux VM.

The same goes for the container image `howtoaks-webapi` it runs on container port 80 internally and forwards the exposed port 80 (Docker file above) on the container to port 8080 externally on the Docker host Linux VM. This means we can access both the Web and Web API from the outside.

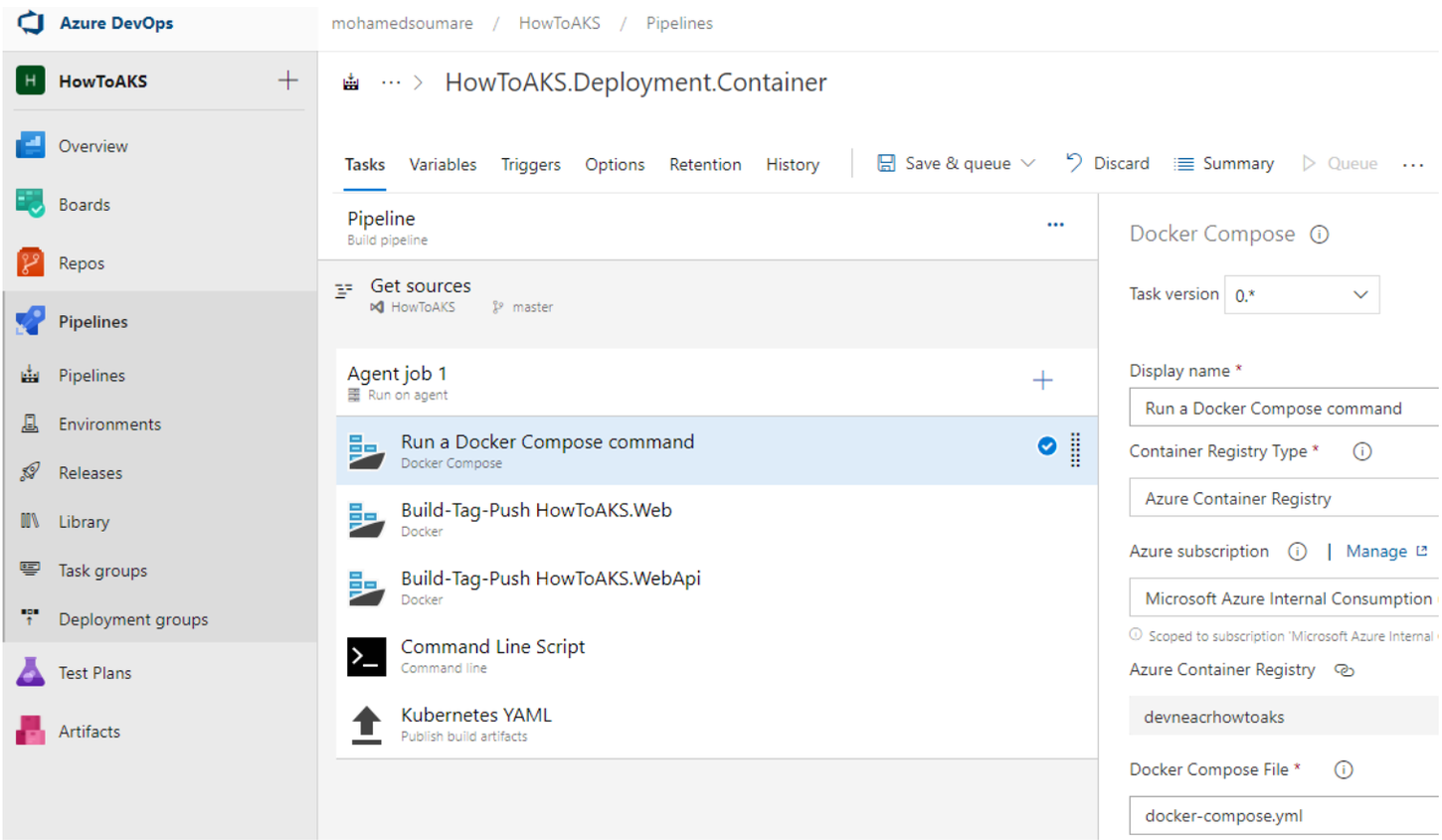
Another way we could do this is by restricting the Web API access only to the Web container (e.g., `- 80`) so it is not mapped to any of the port on the Docker host Linux VM.

```yaml
version: '3.4'

services:
  howtoaks-web:
    image: howtoaks-web
    container_name: howtoaks-web
    build:
      context: ./HowToAKS.Web
    ports:
      - 8080:80

  howtoaks-webapi:
    image: howtoaks-webapi
    container_name: howtoaks-webapi
    build:
      context: ./HowToAKS.WebApi
    ports:
      - 8081:80
```

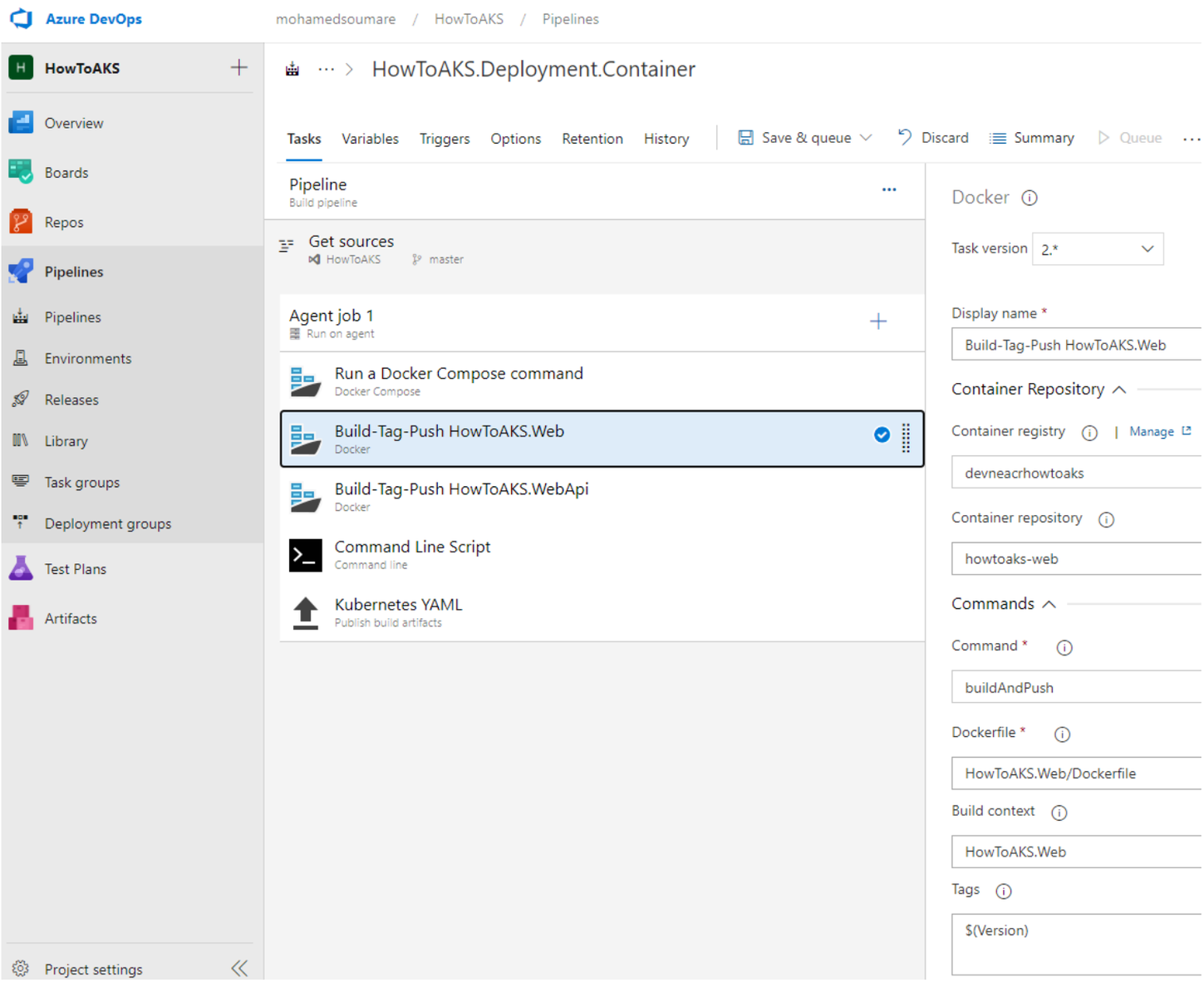
7.4. Push Container Images to ACR with Azure DevOps & CI/CD

Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.ACR) and choose Ubuntu as Agent.

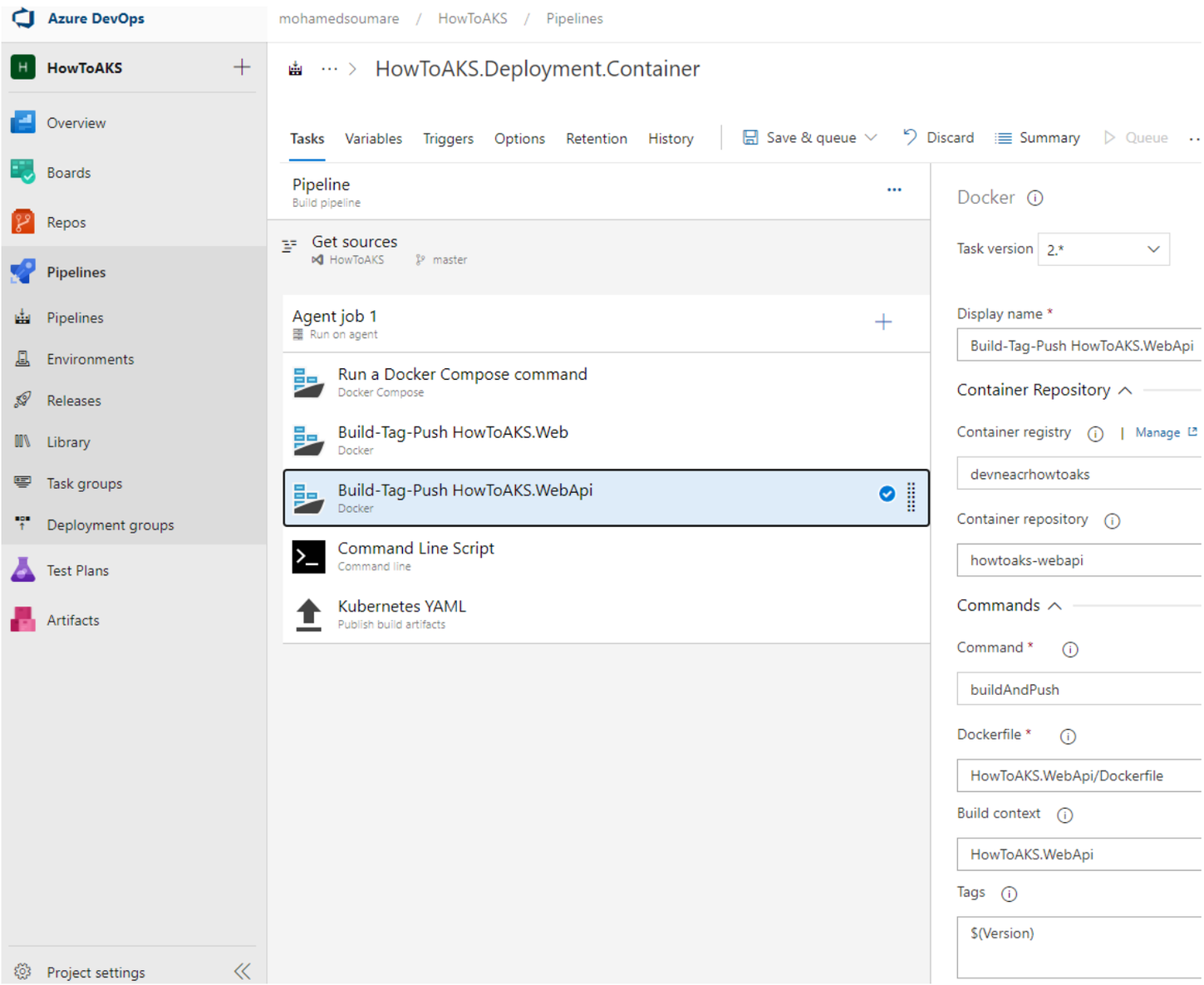
Step 11 > Select your Subscription, Azure Container Registry, and Docker Compose file.



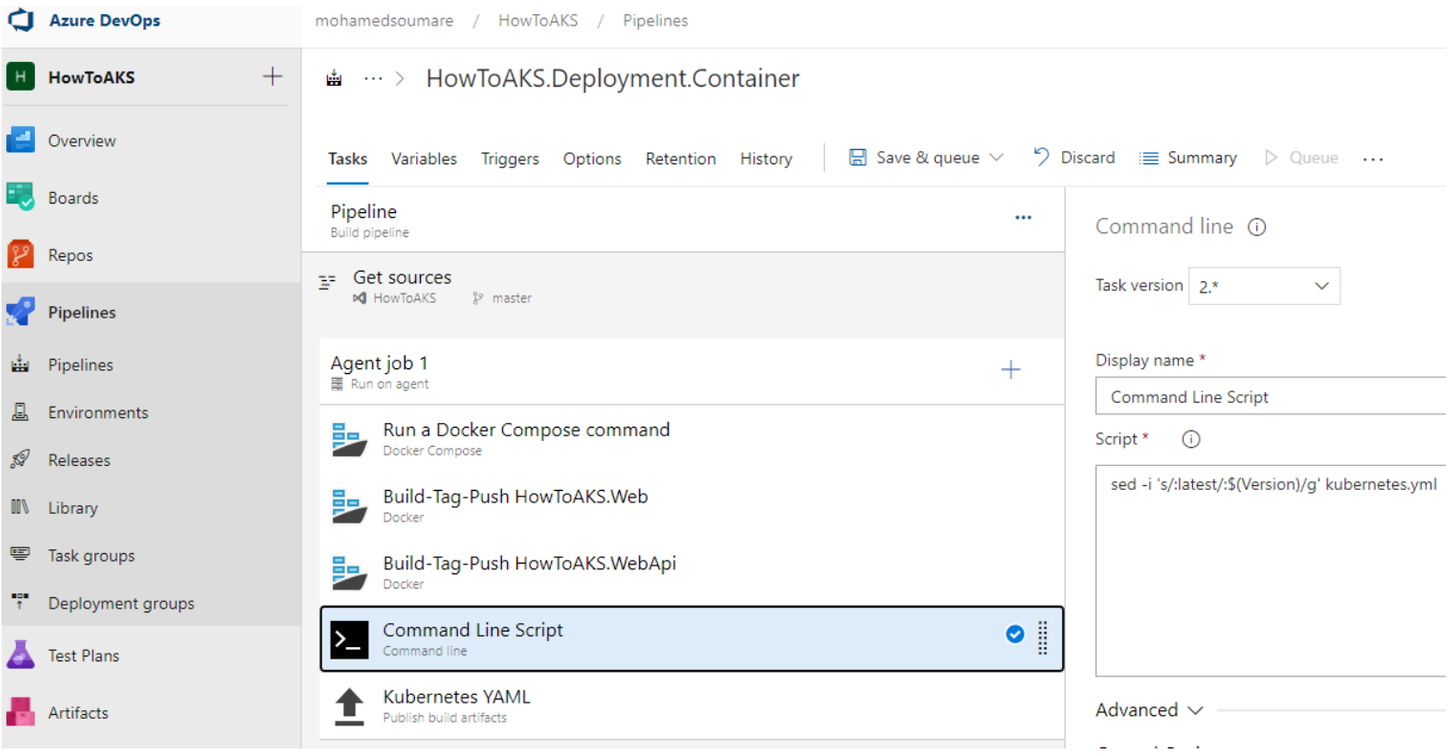
Build-Tag-Push the Web project



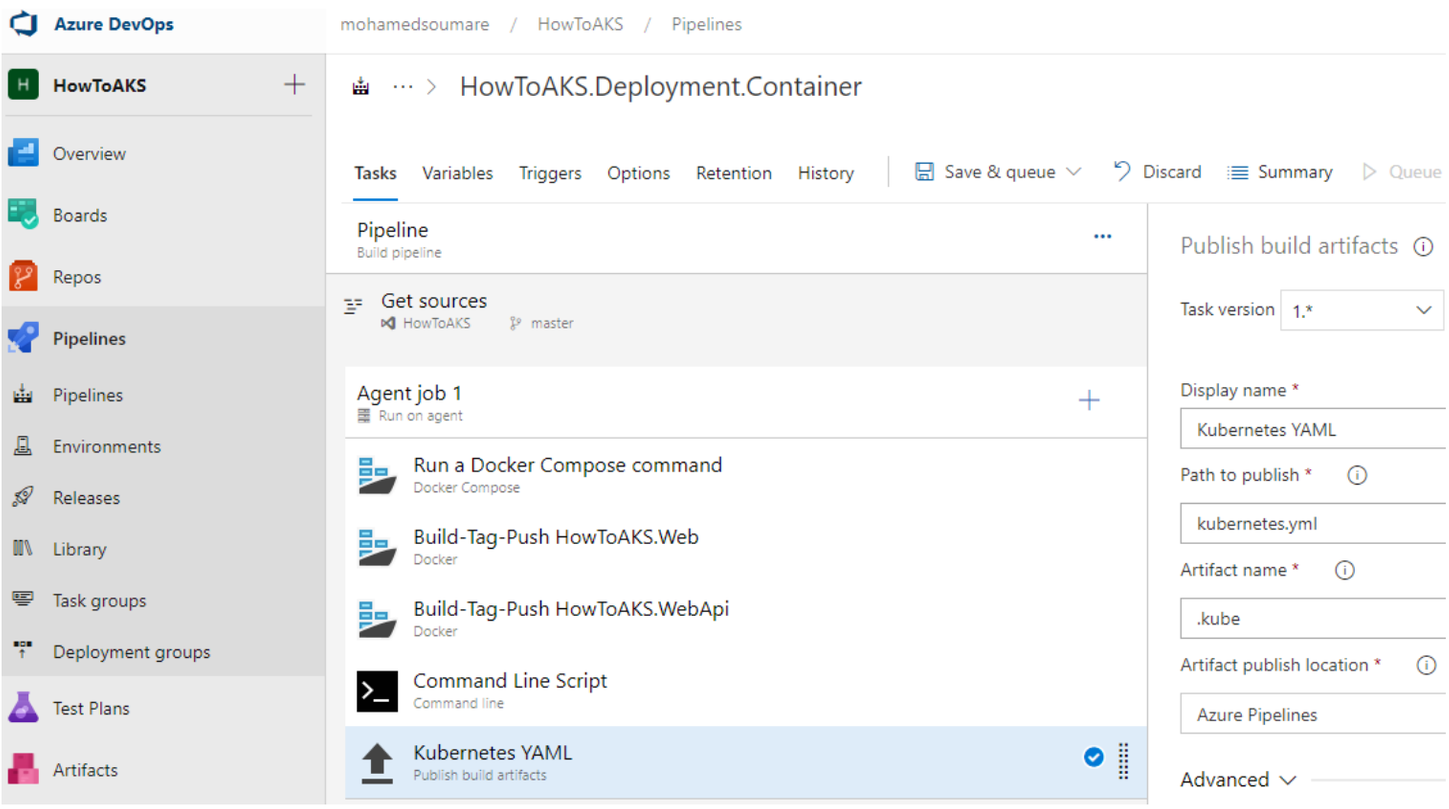
Build-Tag-Push the Web API project



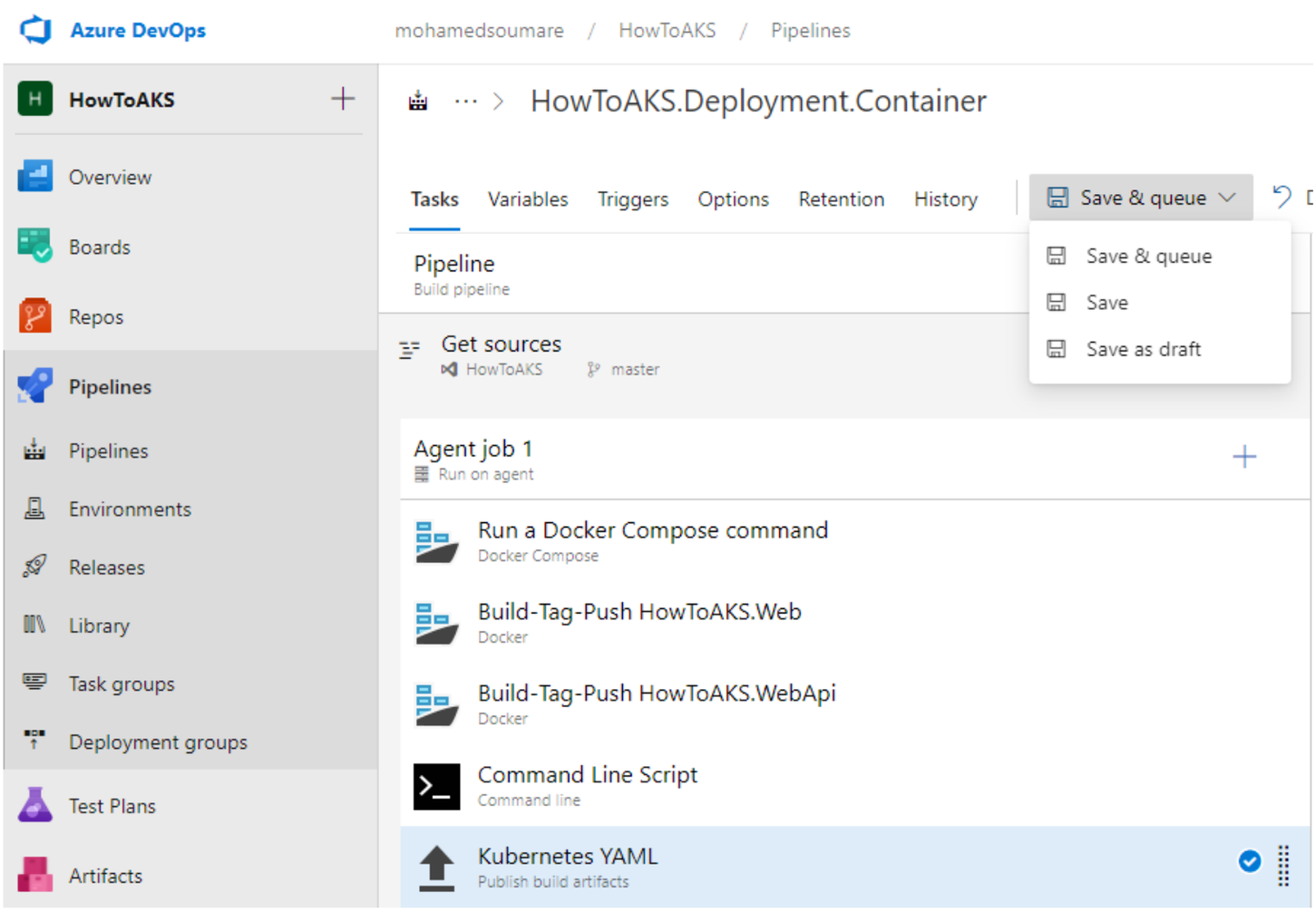
Add the "sed" command line to replace the value "latest" in the Kubernetes.yaml file with the current Build Version.



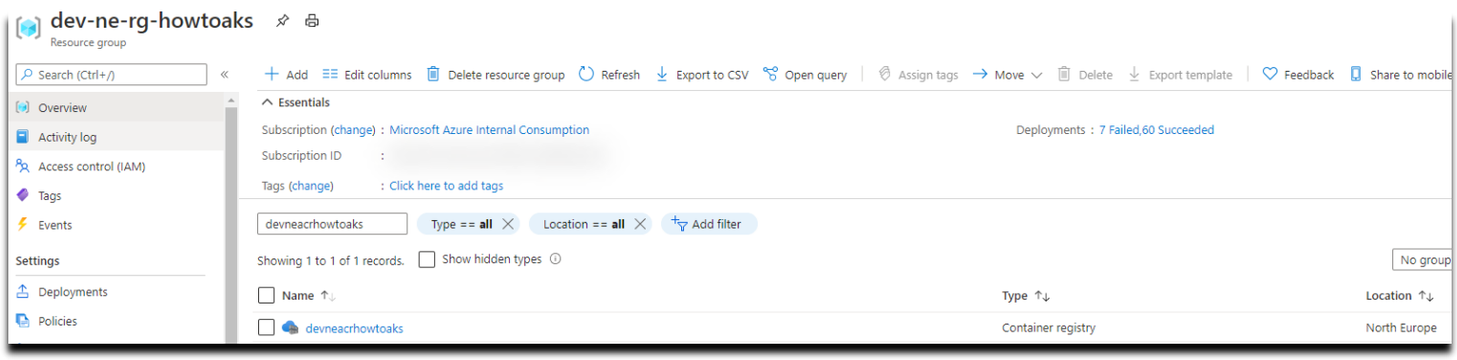
Publish the Kubernetes Artifacts



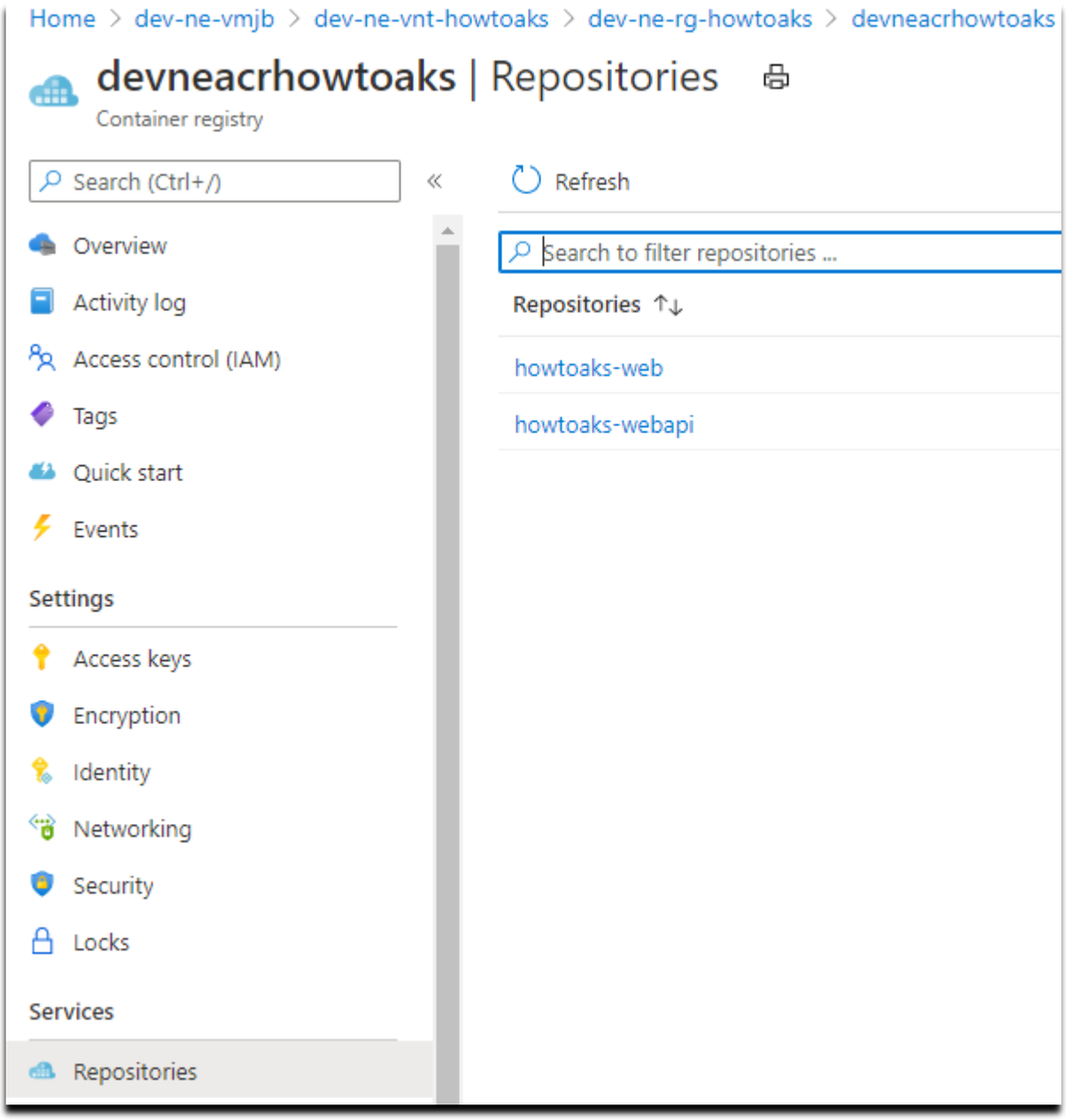
Step 12 > Select Save & Queue to launch the deployment of the Containers to ACR.



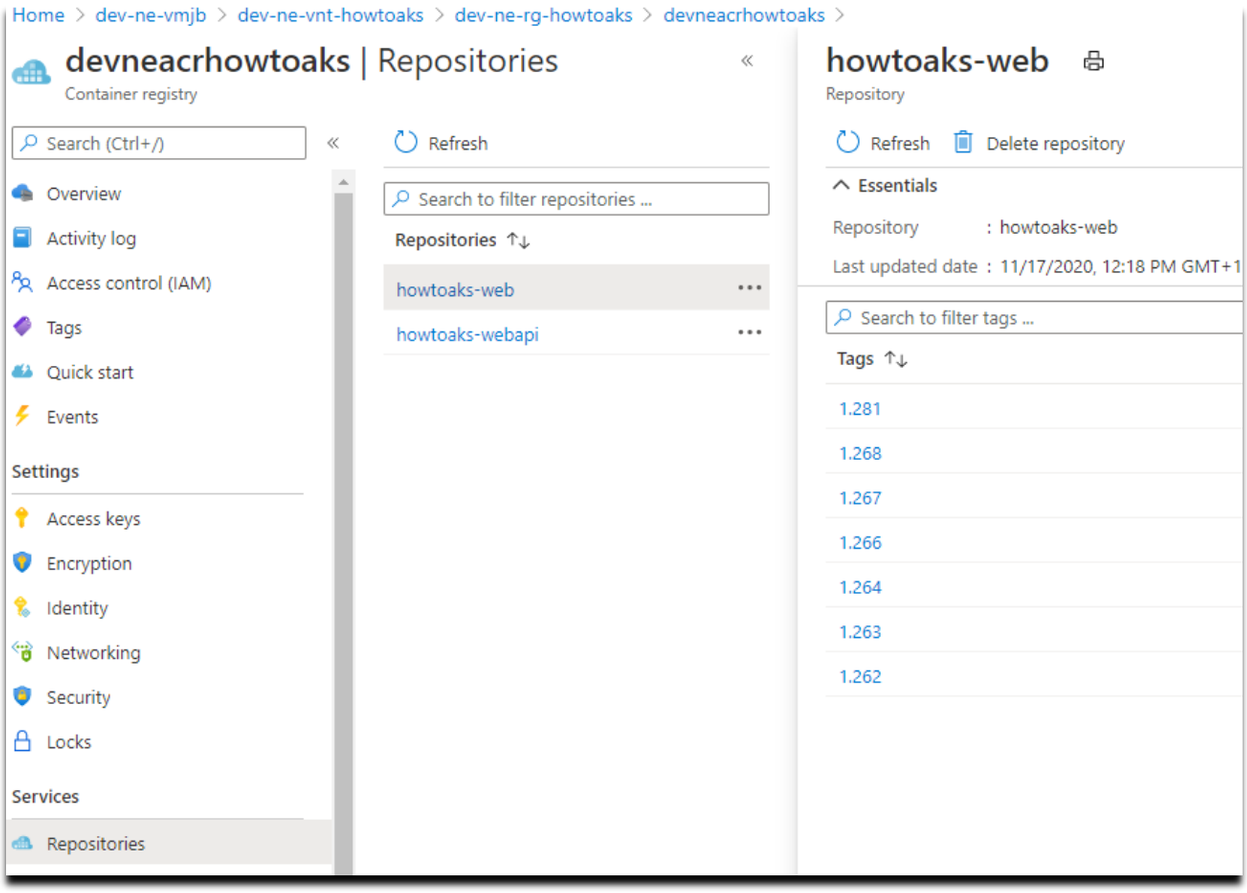
Step 13 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the Container Registry.

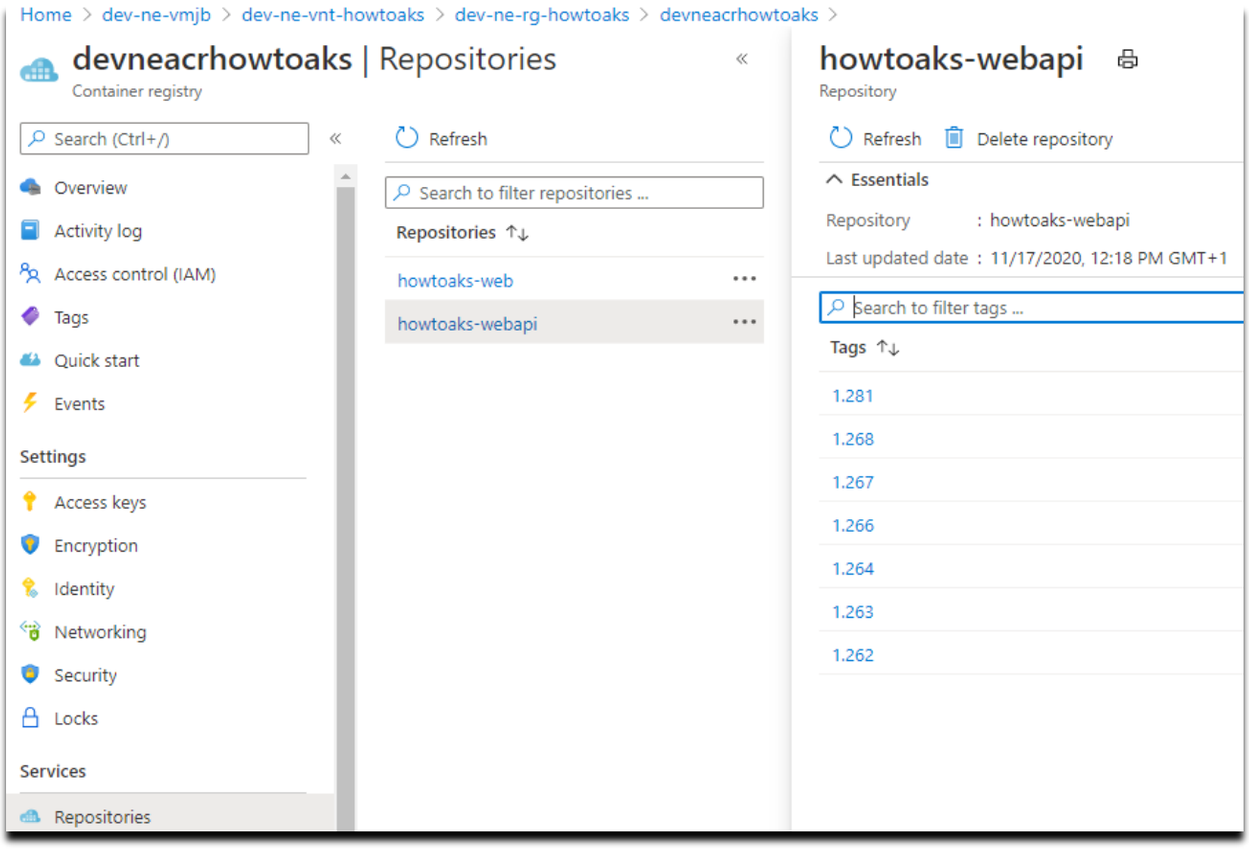


Step 14 > Select Repositories > You will see the 2 Container Repositories we pushed earlier from Azure DevOps to ACR (howtoaks-web & howtoaks-webapi)



Step 15 > Select each repository > You will see different tags created using the Azure DevOps Build Version.





8. Azure Kubernetes Service

Now that we have containerized our application and deployed the containers to Azure Container Registry, we need a way to orchestrate the containers. Hence, we will deploy Azure Kubernetes Service (AKS) which provides a managed Kubernetes service that reduces the complexity for deployment and management of cluster resources such as resource reservation, node pools, namespaces, pods, add-ons, upgrades etc.

8.1. Cluster Sizing & Forcasting for Growth

When planning for a cluster capacity, you must understand the nature of your services (Stateless and/or Stateful), the workloads, the infrastructure/hardware, and the security requirements. Here are some other points to remember when making such decision.

For services with little to no data

- ✓ Focus on **Performance** of the machine
- ✓ CPU Core, Speed and Network Bandwitdth

For services with a lot of data

- ✓ Focus on **Size** of the machine
- ✓ RAM and Disk storage

In AKS it's recommended to have at least 3 nodes for a production cluster, however you must keep in mind some of the resources are reserved on each node for Kubernetes itself to function properly (kube-reserved, memory available, eviction rule etc.).

You must also plan for growth of your microservices, because as the number of pods increase the allocated resources for AKS increase. So, you must include these aspects in your calculation and plan accordingly.

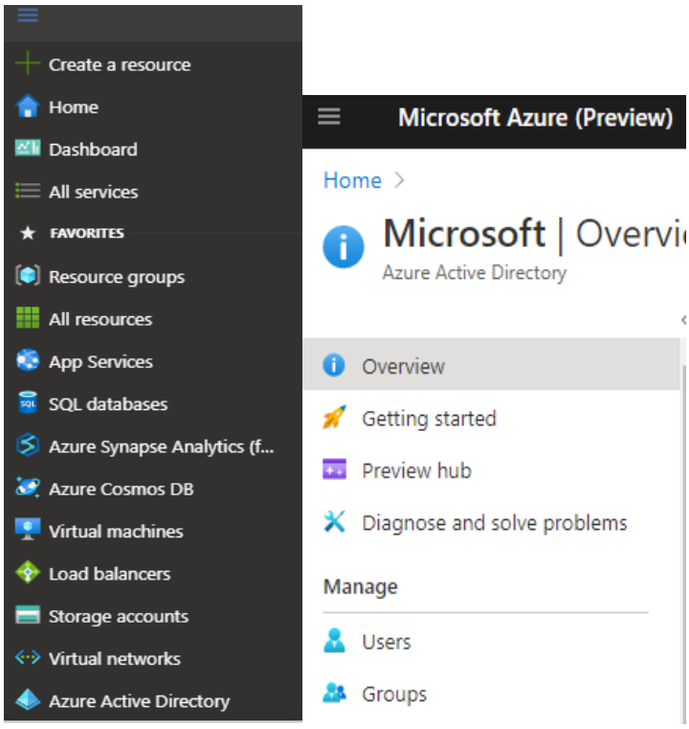
8.2.Public Cluster vs Private Cluster

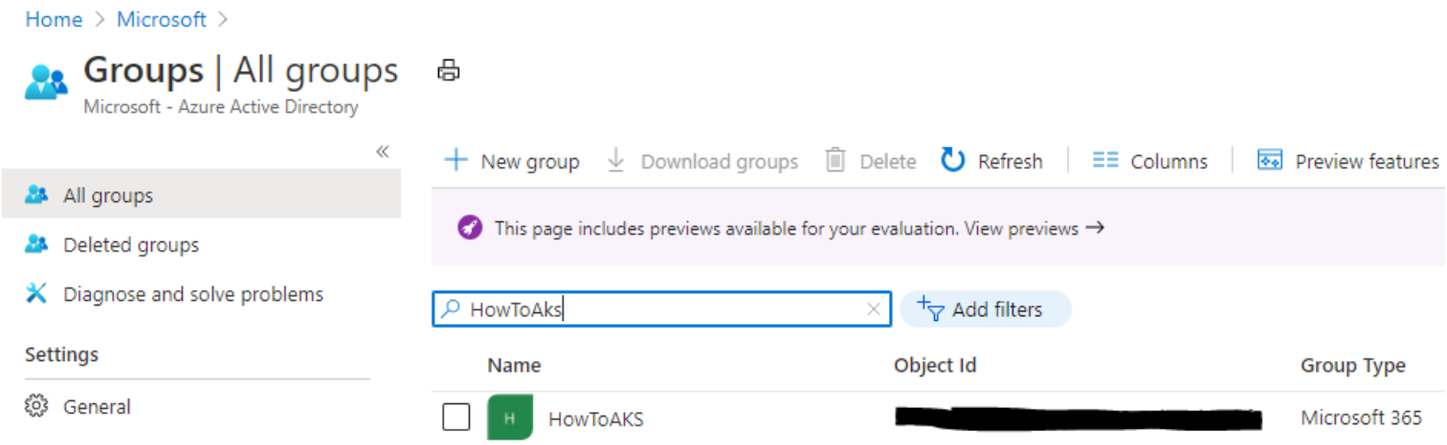
When creating an AKS cluster, by default the API server exposes a Public endpoint and FQDN. If you want to protect this public endpoint you can either define **Authorized IP ranges** or you can create a **Private cluster** (so the API server is only accessible within your virtual network). In our scenario, we will be deploying a Private cluster, and we will use our Management Box and Build agent deployed earlier to work with the API server by using kubectl client and Azure DevOps & CI/CD.

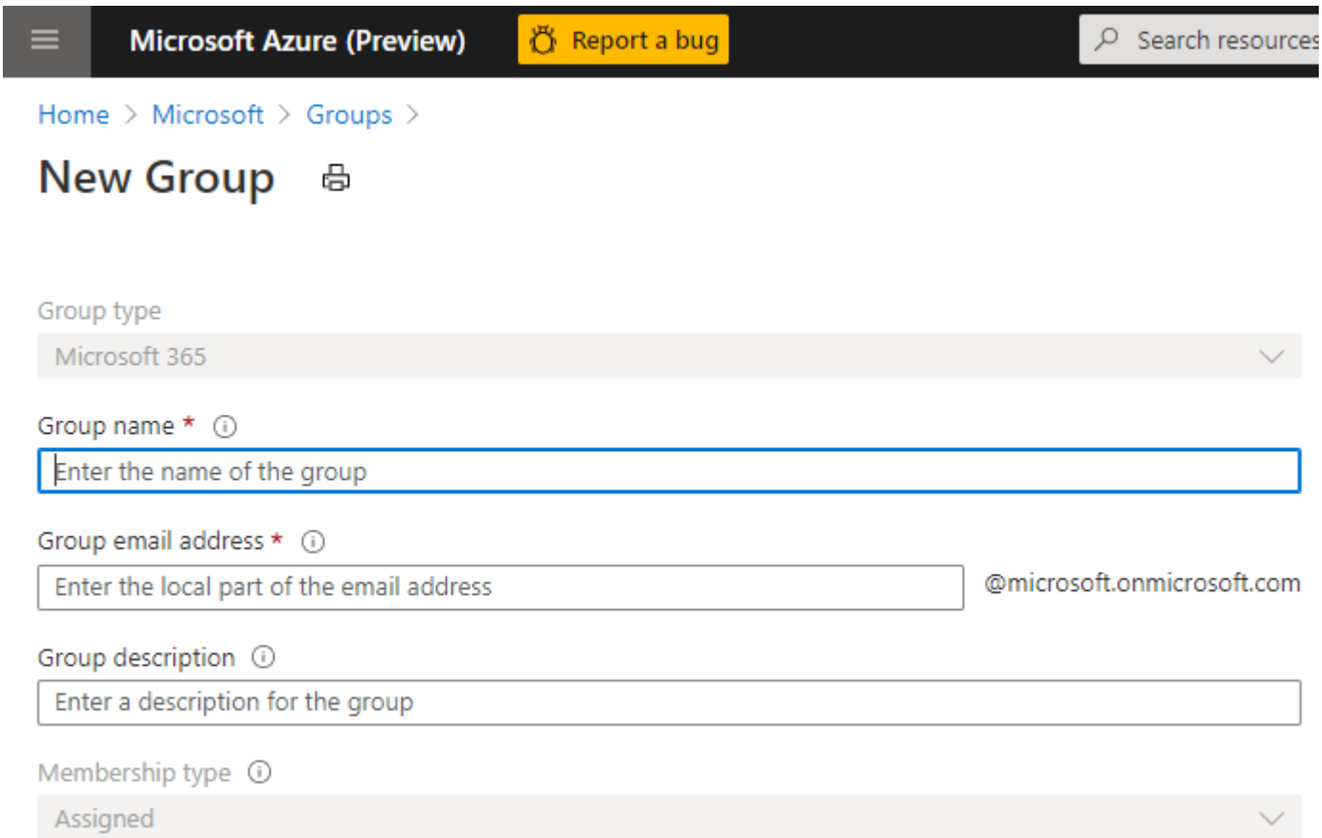
8.3. Integration with Azure AD

For a finer grained control over our cluster resources, we can integrate AKS RBAC with Azure AD. This process will lead us to define an **Administrator Group** in Azure AD to manage the cluster resources and **User Groups** to grant users or groups, access to our cluster resources within a namespace or across the cluster. In our scenario, using the following step we will define the Admin Group used to deploy our AKS cluster.

Step 1 > Navigate to Azure Portal > Select Active Directory > Select Groups > New Group. In our case we've named it HowToAKS as you can see in the following screenshots.


Create a resource
Home
Dashboard
All services
FAVORITES
Resource groups
All resources
App Services
SQL databases
Azure Synapse Analytics (f...
Azure Cosmos DB
Virtual machines
Load balancers
Storage accounts
Virtual networks
Azure Active Directory
Microsoft Azure (Preview)
Home >
Microsoft | Overvi
Azure Active Directory
Overview
Getting started
Preview hub
Diagnose and solve problems
Manage
Users
Groups


Home > Microsoft >
Groups | All groups
Microsoft - Azure Active Directory
All groups
Deleted groups
Diagnose and solve problems
Settings
General
New group
Download groups
Delete
Refresh
Columns
Preview features
This page includes previews available for your evaluation. View previews →
HowToAks
Add filters
Name
Object Id
Group Type
HowToAKS
[redacted]
Microsoft 365



Step 2 > Make a note of the Group Id generated after creation. We will use this group id in our ARM template to create the integration of Azure AD and AKS RBAC.

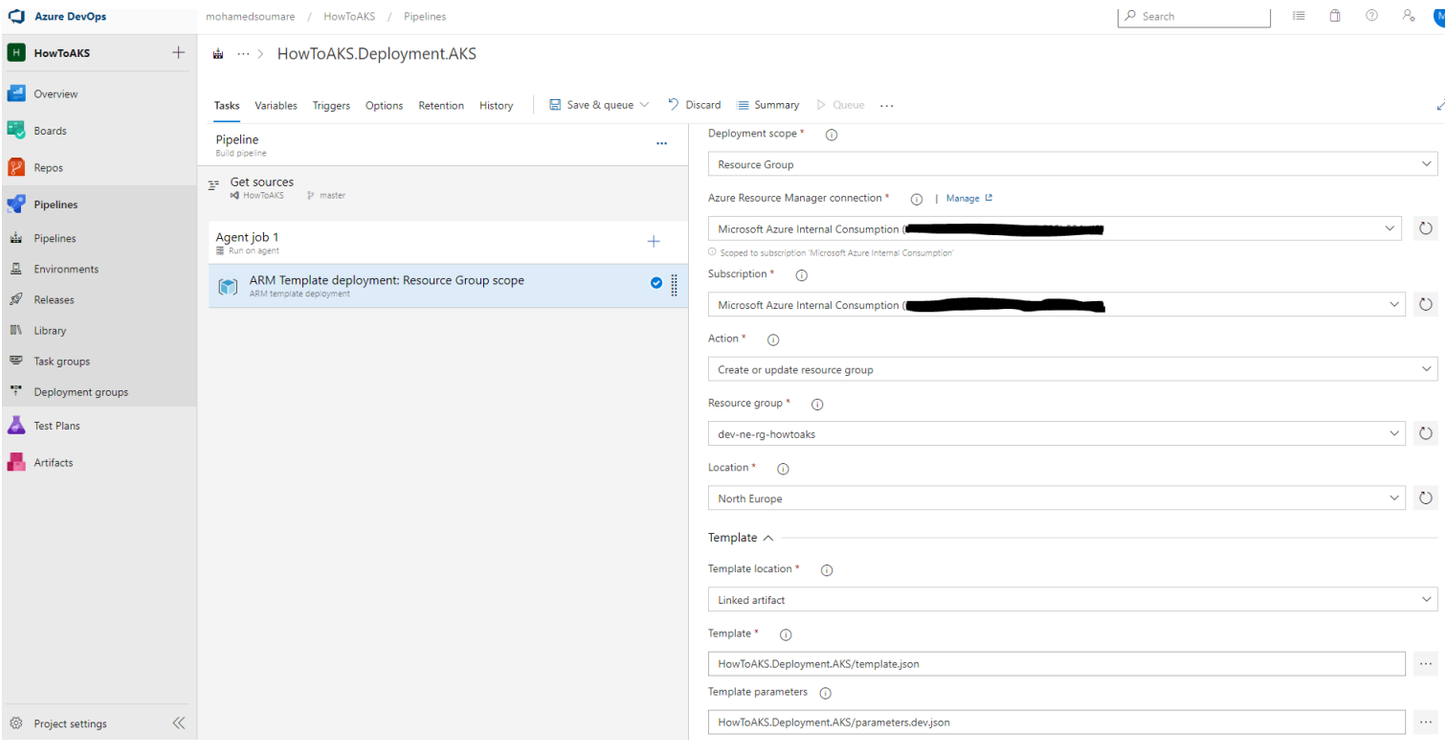
Please find the reusable ARM templates in the following GitHub repository.

Infrastructure as Code - https://github.com/msoumare/HowToAKS/tree/main/5-aks

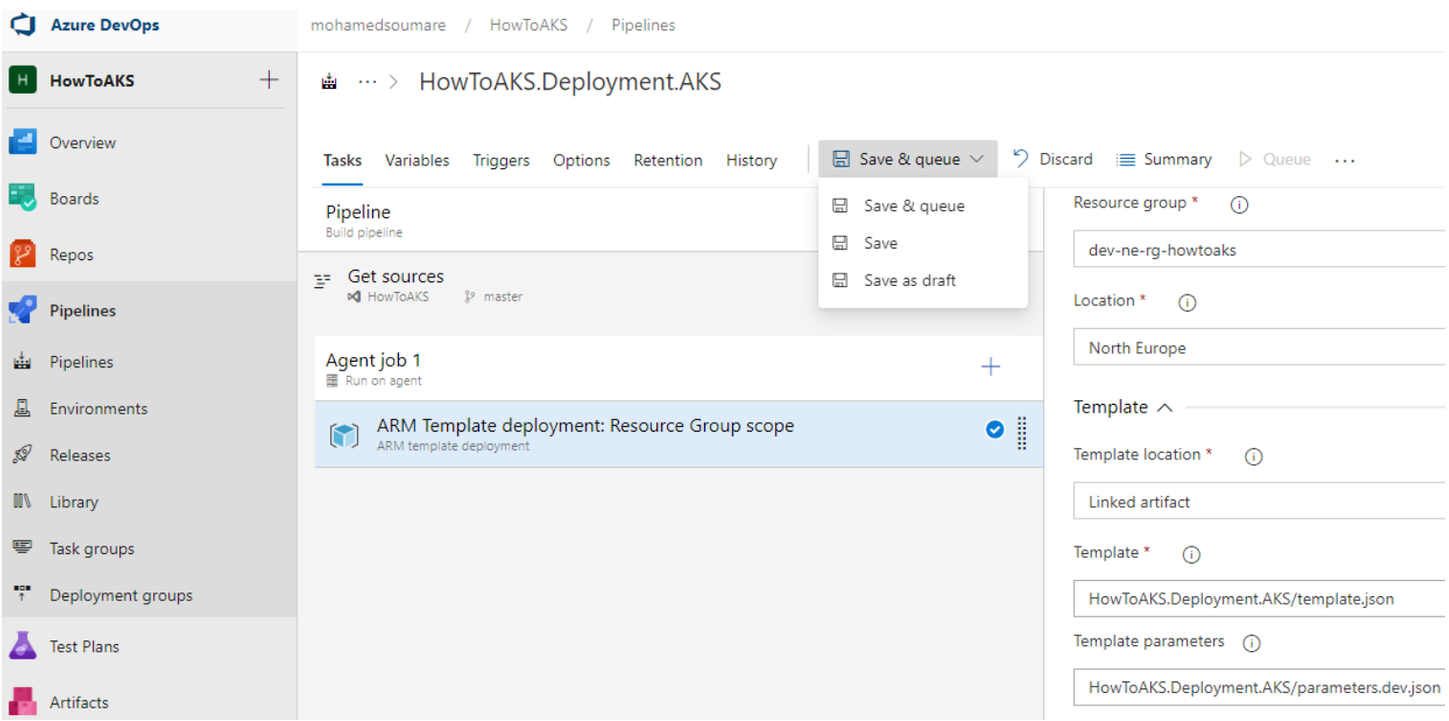
8.4. Deployment with Azure DevOps & CI/CD

Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.AKS).

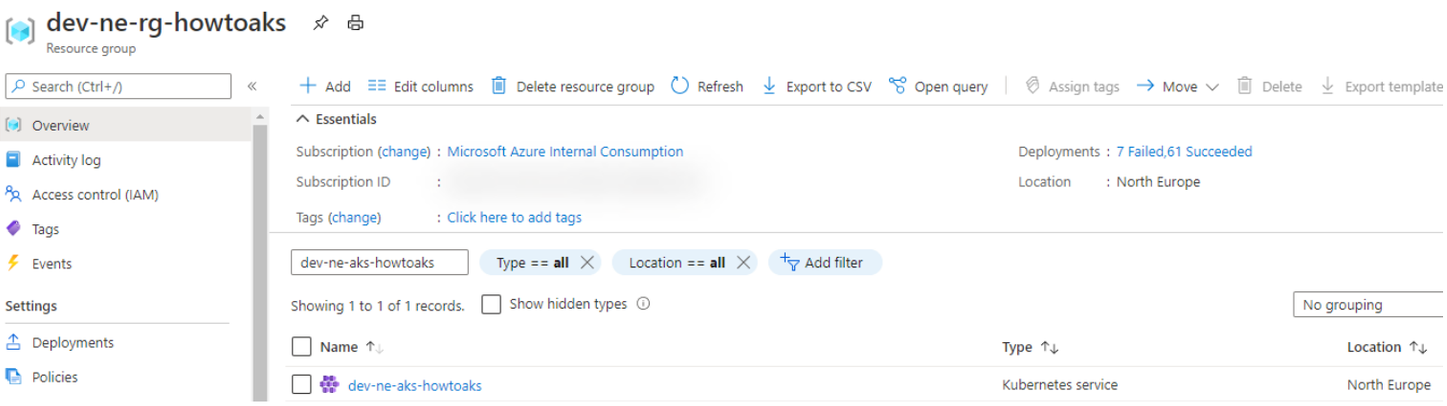
Step 11 > Select your Subscription, Resource Group, Location, Template, and Template Parameters.



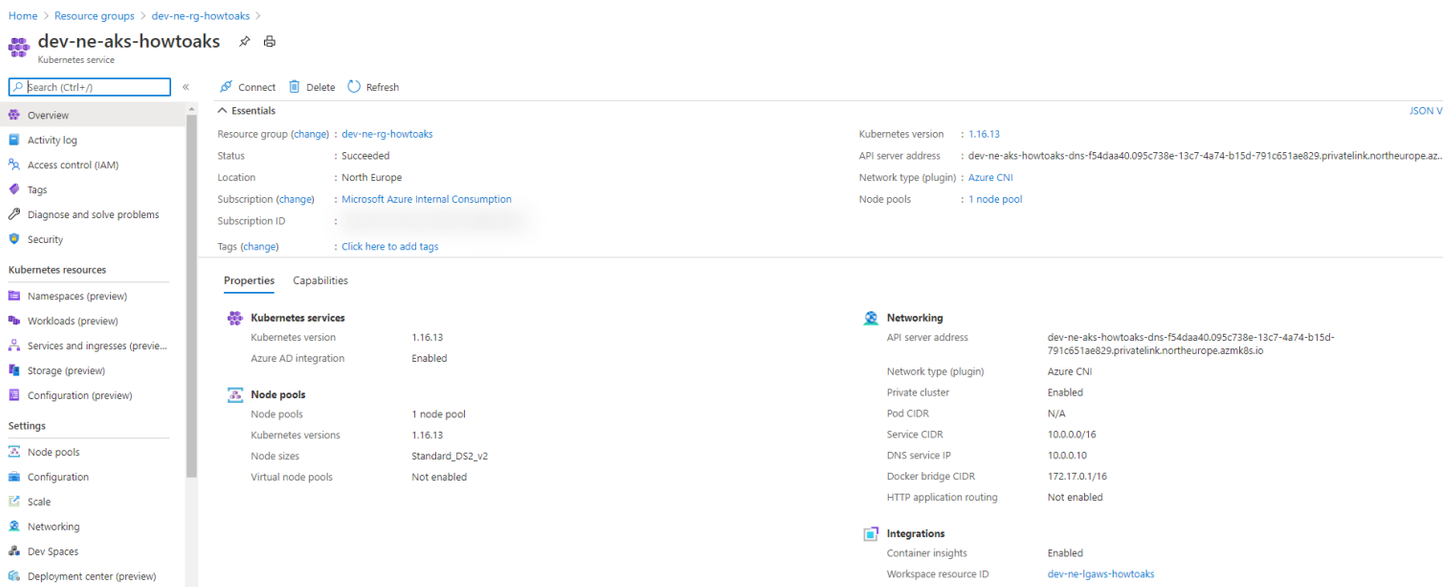
Step 12 > Select Save & Queue to launch the deployment of AKS.



Step 13 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the AKS.



Step 14 > You will notice the AKS API server address with a private link, along with the Azure AD integration.



9. Deploying our microservices in AKS

It's about that time to deploy our workloads to AKS. We will prepare our Kubernetes deployment files in the following section.

Please find the reusable ARM templates in the following GitHub repository.

Infrastructure as Code - https://github.com/msoumare/HowToAKS/tree/main/app

9.1. Preparing Kubernetes Deployment & Service Manifests (using default namespace)

To create the deployment YAML and the service YAML we will use VSCode (you can also use a text editor of your choice). One way I like to do is, combine both the Deployment & Service into 1 YAML file called kubernetes.yaml as shown below.

In this file you will notice the **Kind Deployment howtoaks-web** with its container image address (the Azure Container Registry URL we deployed in an earlier topic), **devneacrhowtoaks.azurecr.io/howtoaks-web:latest** followed by the name of the container and the version (latest, will be replaced with the Azure DevOps current build version). The number of replicas indicates how many pods will be running our web app.

You will also see the **Kind Service** with a type **Load Balancer**, external **port:8080** and the internel target port (which is the container port) **targetPort:80**.

The next part of the file illustrates the **Kind Deployment howtoaks-webapi** with its container image address (the Azure Container Registry URL we deployed in an earlier topic), **devneacrhowtoaks.azurecr.io/howtoaks-webapi:latest** followed by the name of the container and the version (latest, will be replaced with the Azure DevOps current build version). The number of replicas indicates how many pods will be running our web api.

You will also see the **Kind Service** with a type **Load Balancer** and external **port:8081** and the internel target port (which is the container port) **targetPort:80**.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: howtoaks-web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: howtoaks-web
  template:
    metadata:
      labels:
        app: howtoaks-web
    spec:
      containers:
      - name: howtoaks-web
        image: devneacrhowtoaks.azurecr.io/howtoaks-web:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: howtoaks-web
spec:
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 8080
```

```yaml
    targetPort: 80
  selector:
    app: howtoaks-web

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: howtoaks-webapi
spec:
  replicas: 3
  selector:
    matchLabels:
      app: howtoaks-webapi
  template:
    metadata:
      labels:
        app: howtoaks-webapi
    spec:
      containers:
      - name: howtoaks-webapi
        image: devneacrhowtoaks.azurecr.io/howtoaks-webapi:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: howtoaks-webapi
spec:
  type: LoadBalancer
  ports:
  - port: 8081
    targetPort: 80
  selector:
    app: howtoaks-webapi
```

9.2. Preparing Kubernetes Deployment & Service Manifests (using custom namespaces)

Another way to deploy our microservices is to define custom namespaces and assign our pods a namespace. In the following deployment YAML file you will see 2 namespaces (front-end & back-end) assigned respectively to the Web & Web API containers. Using namespaces is a good way to handle cluster multitenancy logically and achieve greater density in your nodes, hence maximing your cluster resources.

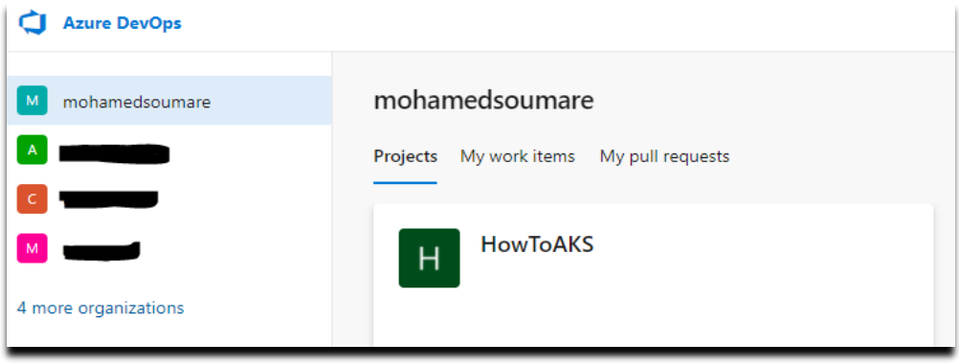
When working with different teams or business units it's a good practice to share a cluster resources using namespaces instead of creating multiple physical clusters per team. However, it makes sense though to create different clusters per environment (e.g., development, quality assurance, staging, production).

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: front-end

---

apiVersion: v1
kind: Namespace
metadata:
  name: back-end

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: howtoaks-web
  namespace: front-end
spec:
  replicas: 3
  selector:
    matchLabels:
      app: howtoaks-web
  template:
    metadata:
      labels:
        app: howtoaks-web
    spec:
      containers:
      - name: howtoaks-web
        image: devneacrhowtoaks.azurecr.io/howtoaks-web:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
---

apiVersion: v1
kind: Service
metadata:
  name: howtoaks-web
  namespace: front-end
spec:
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
  selector:
    app: howtoaks-web
```

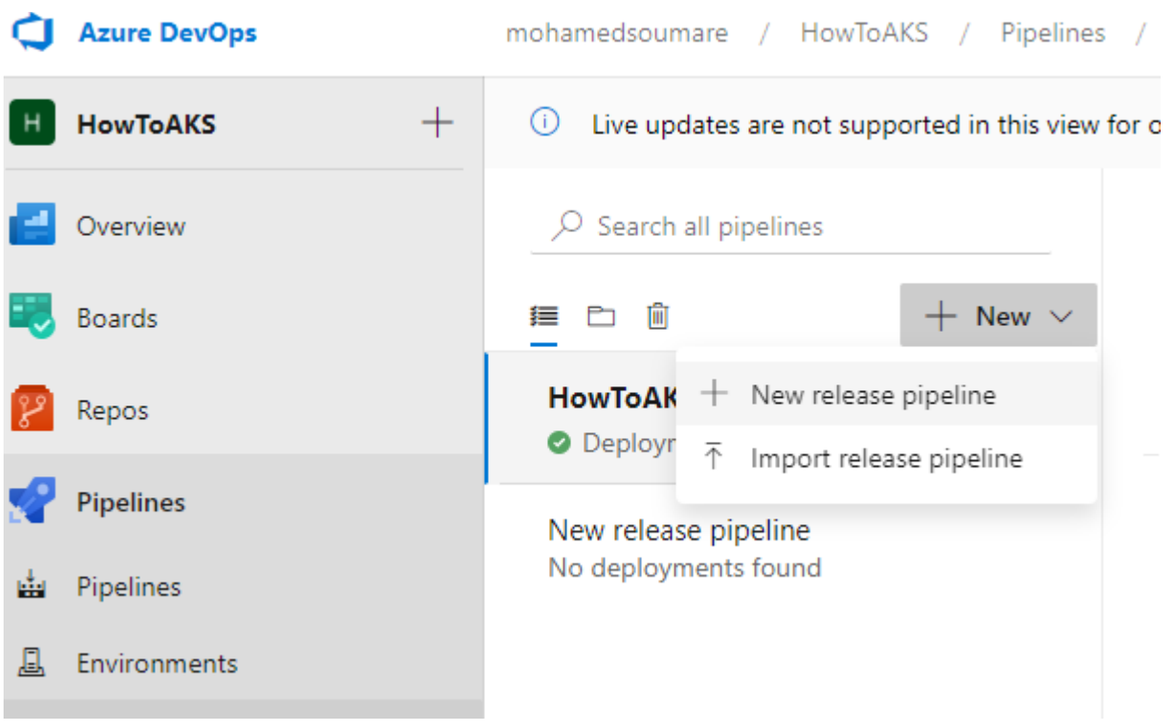
```yaml
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: howtoaks-webapi
  namespace: back-end
spec:
  replicas: 3
  selector:
    matchLabels:
      app: howtoaks-webapi
  template:
    metadata:
      labels:
        app: howtoaks-webapi
    spec:
      containers:
      - name: howtoaks-webapi
        image: devneacrhowtoaks.azurecr.io/howtoaks-webapi:latest
        ports:
        - containerPort: 80
---

apiVersion: v1
kind: Service
metadata:
  name: howtoaks-webapi
  namespace: back-end
spec:
  type: LoadBalancer
  ports:
  - port: 8081
    targetPort: 80
  selector:
    app: howtoaks-webapi
```

9.3.Deployment with Azure DevOps & CI/CD

Step 1 > Navigate to Azure DevOps > Select your project. In our case we've named it HowToAKS as you can see in the following screenshot.



Step 2 > Select Pipelines > Releases > New Release Pipeline



Step 3 > Start with an Empty Job (if you are using Private AKS, like in our case) or Select Deploy to a Kubernetes Cluster (if you are using a Public AKS)

Select a template

Or start with an Empty job

Featured

Azure App Service deployment

Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.

Deploy a Java app to Azure App Service

Deploy a Java application to an Azure Web App.

Deploy a Node.js app to Azure App Service

Deploy a Node.js application to an Azure Web App.

Deploy a PHP app to Azure App Service and Azure Database for MySQL

Deploy a PHP application to an Azure Web App and database to Azure Database for MySQL.

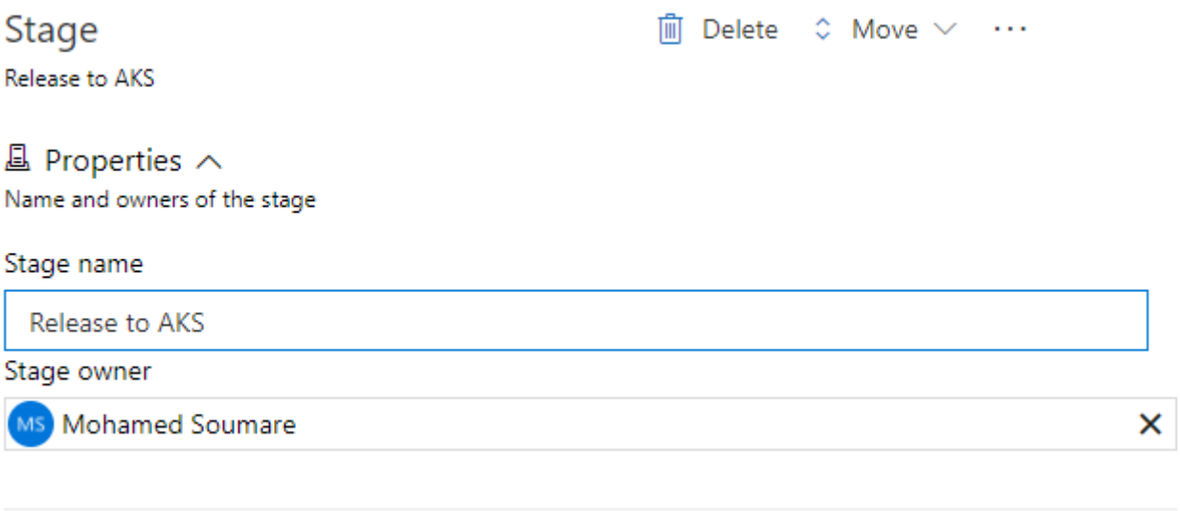
Deploy a Python app to Azure App Service and Azure database for MySQL

Deploy a Python Django, Bottle, or Flask application to an Azure Web App and database to Azure Database for MySQL.

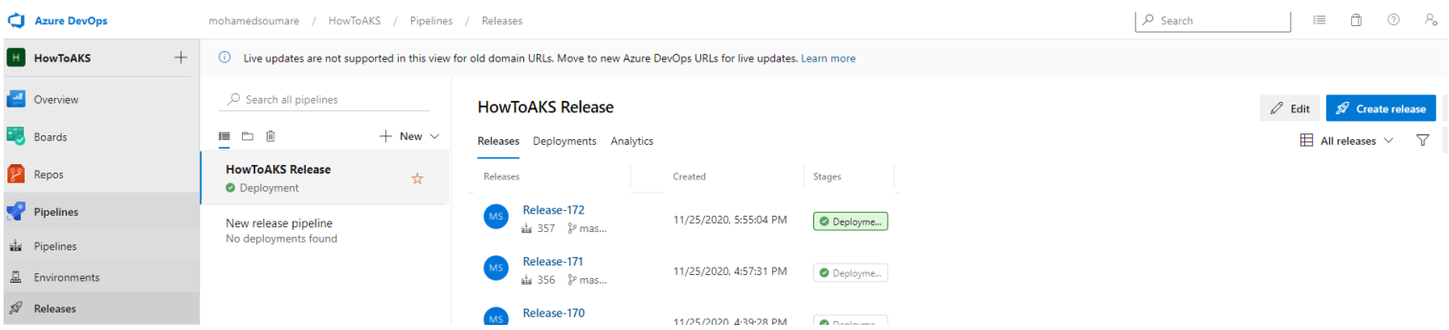
Deploy to a Kubernetes cluster

Deploy, configure, update your containerized applications to a Kubernetes cluster.

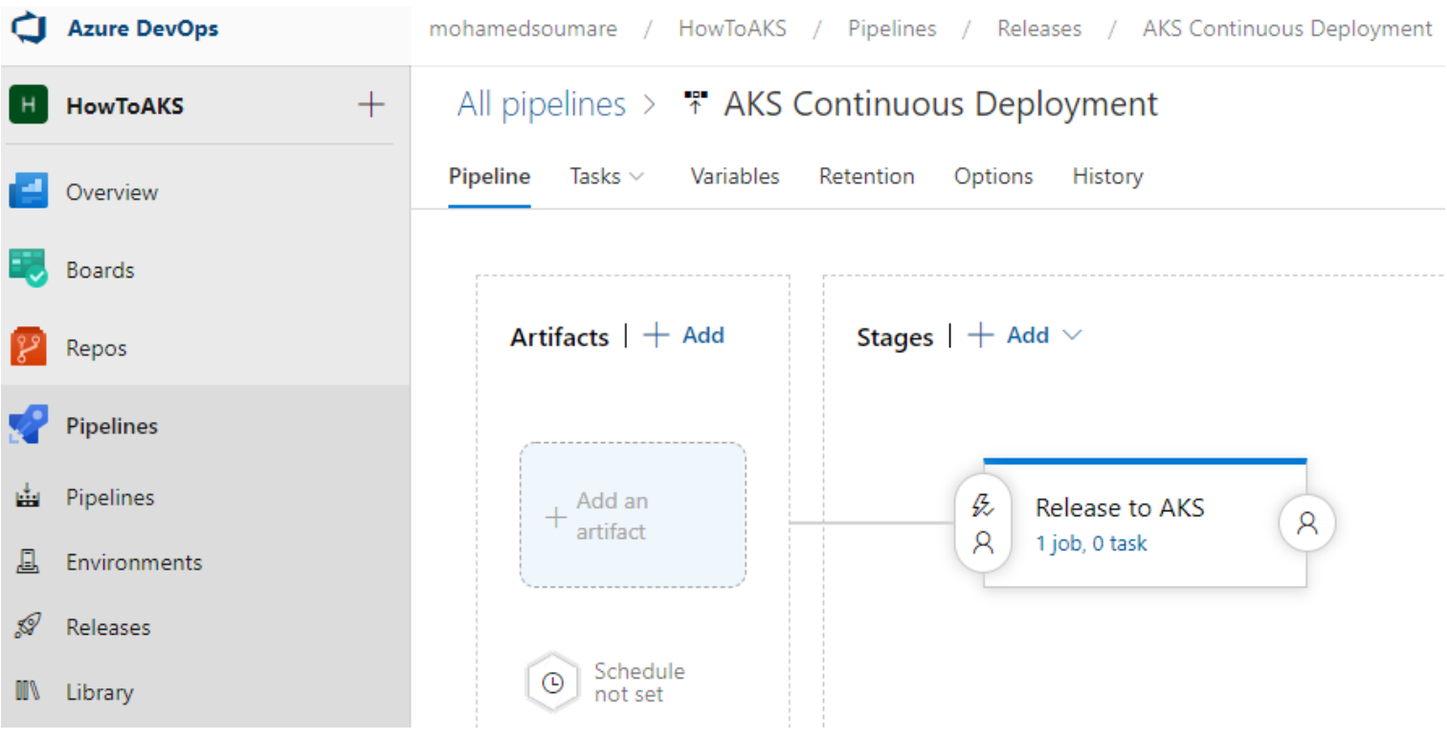
Step 4 > Give a name to your Release Stage.



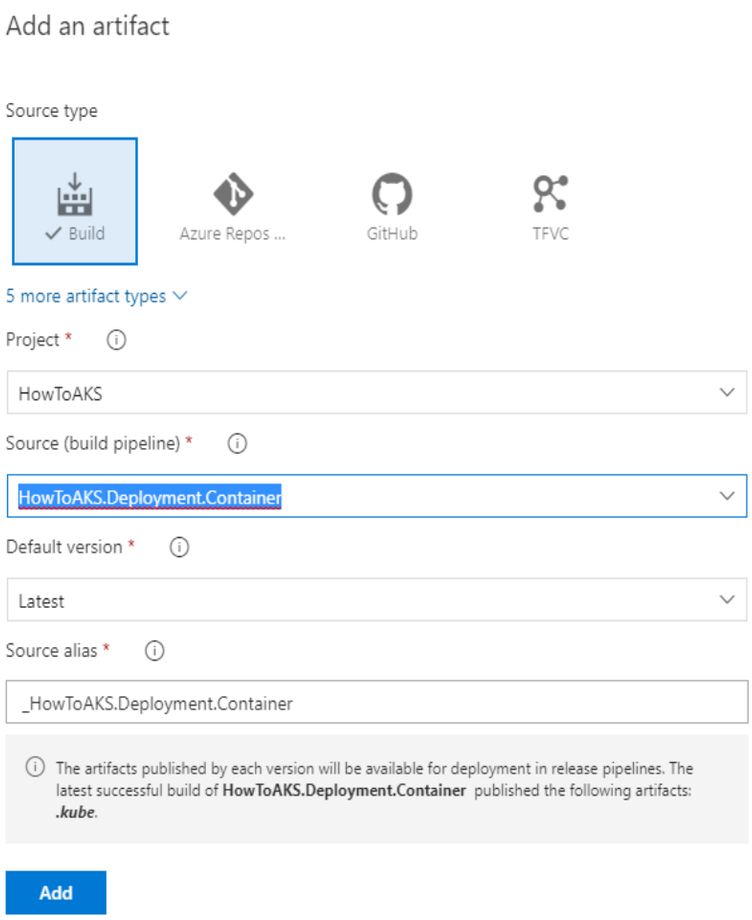
Step 5 > Give a name to your Release Stage.



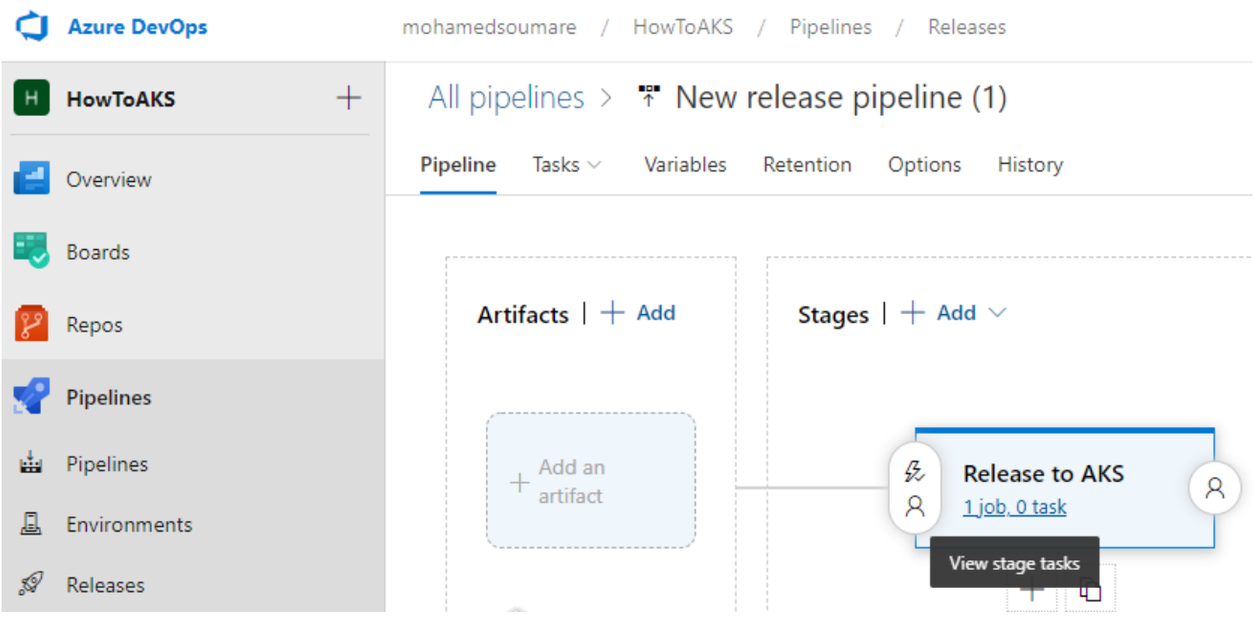
Step 6 > Give a name to your Pipeline > Click on Add Artifacts



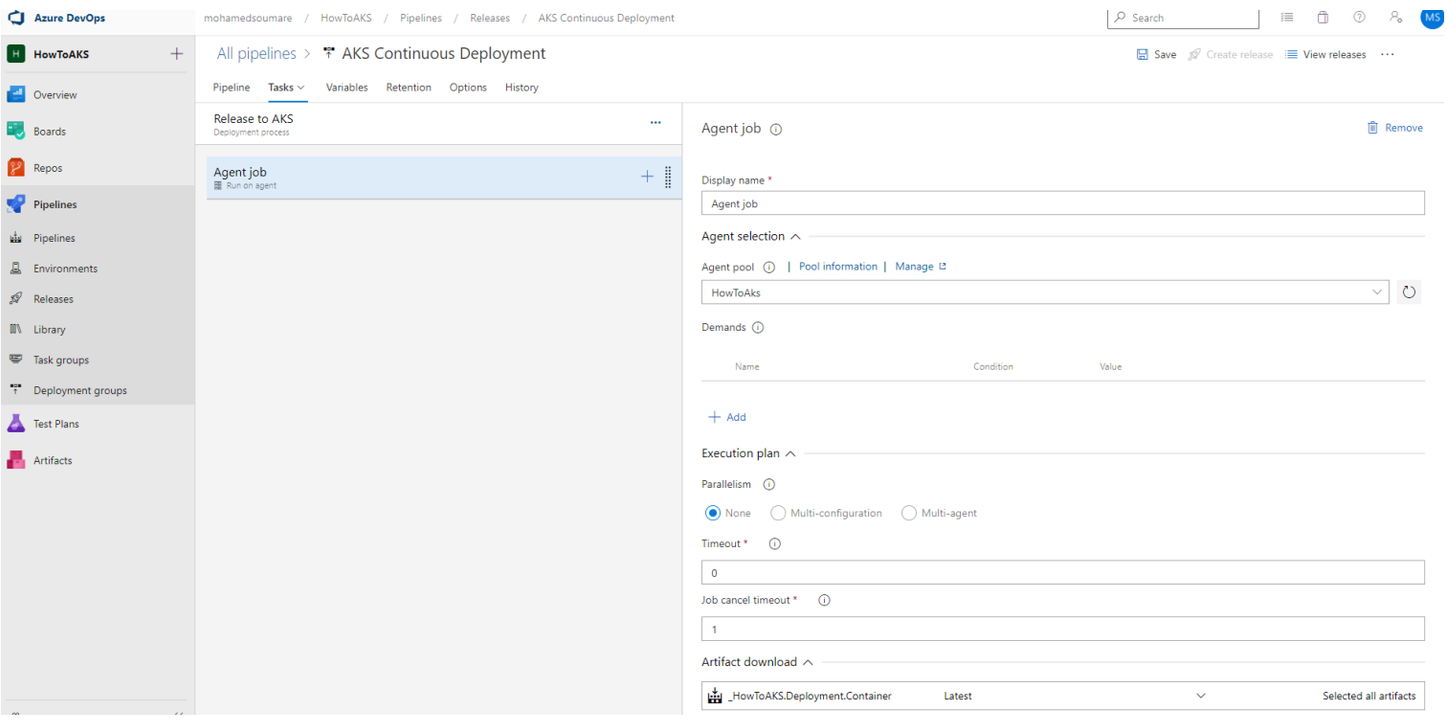
Step 7 > Select the artifact from the Container Build Pipeline we defined earlier > Click Add.



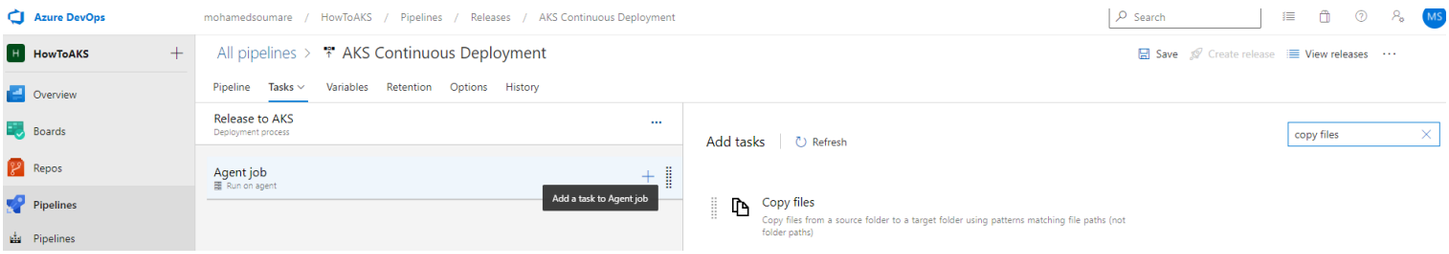
Step 8 > Click on View stage tasks.



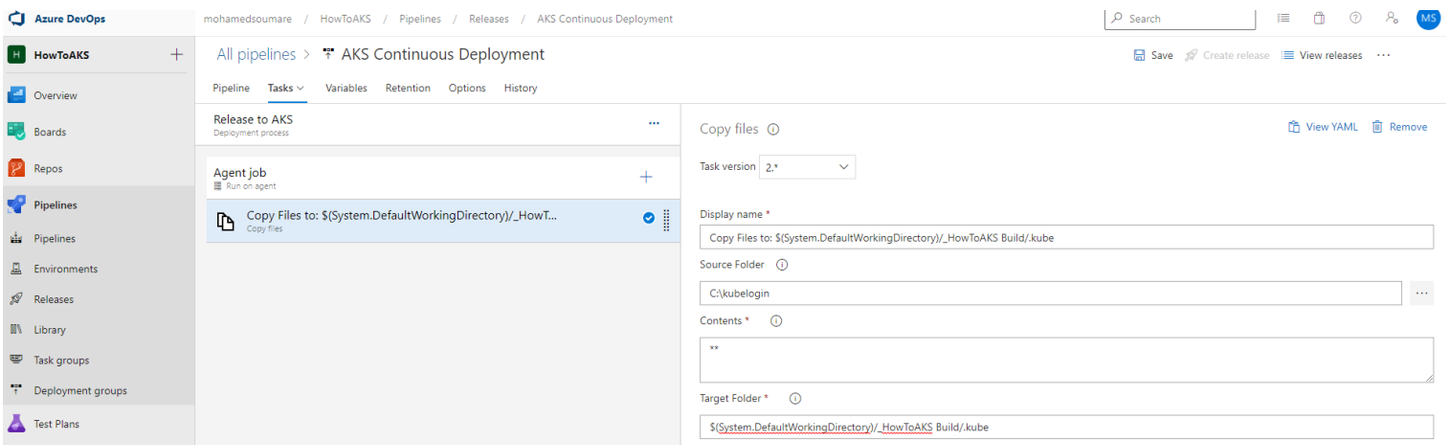
Step 9 > Select Agent job > Choose the Private Agent > Choose the Build Artifact to download.



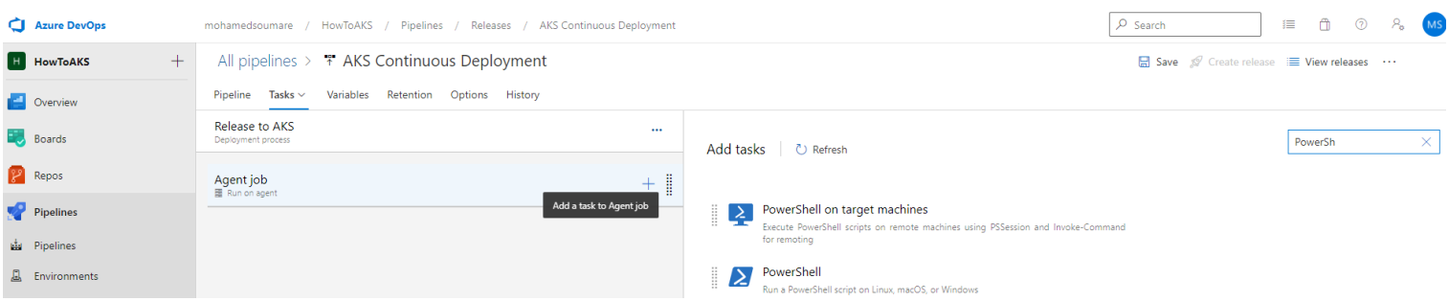
Step 10 > Add a task to the Agent > Search for copy files > Click Add.



Step 11 > Set the Source Folder to C:\KubeLogin (this path is to the directory we created during the installation of Kubelogin when we were setting up our Build Machine).

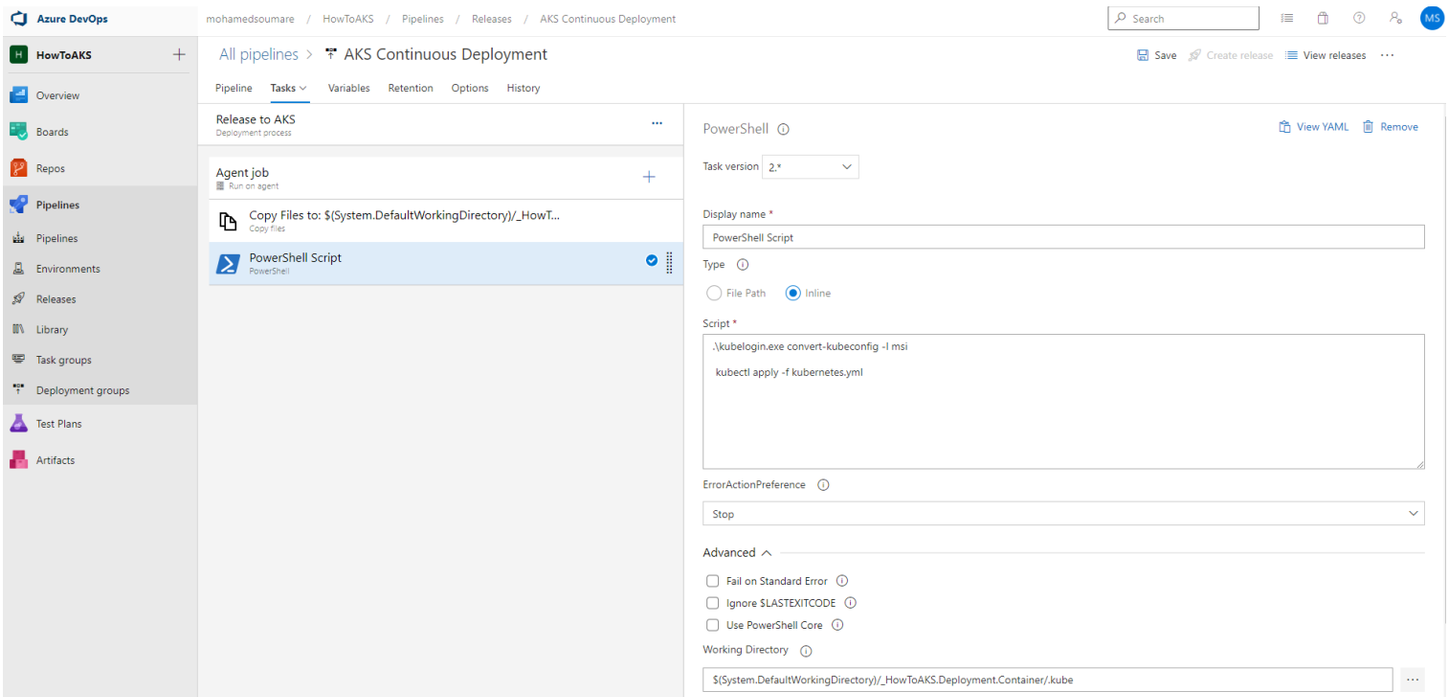


Step 12 > Add another task to the Agent > Search for PowerShell script > Click Add.

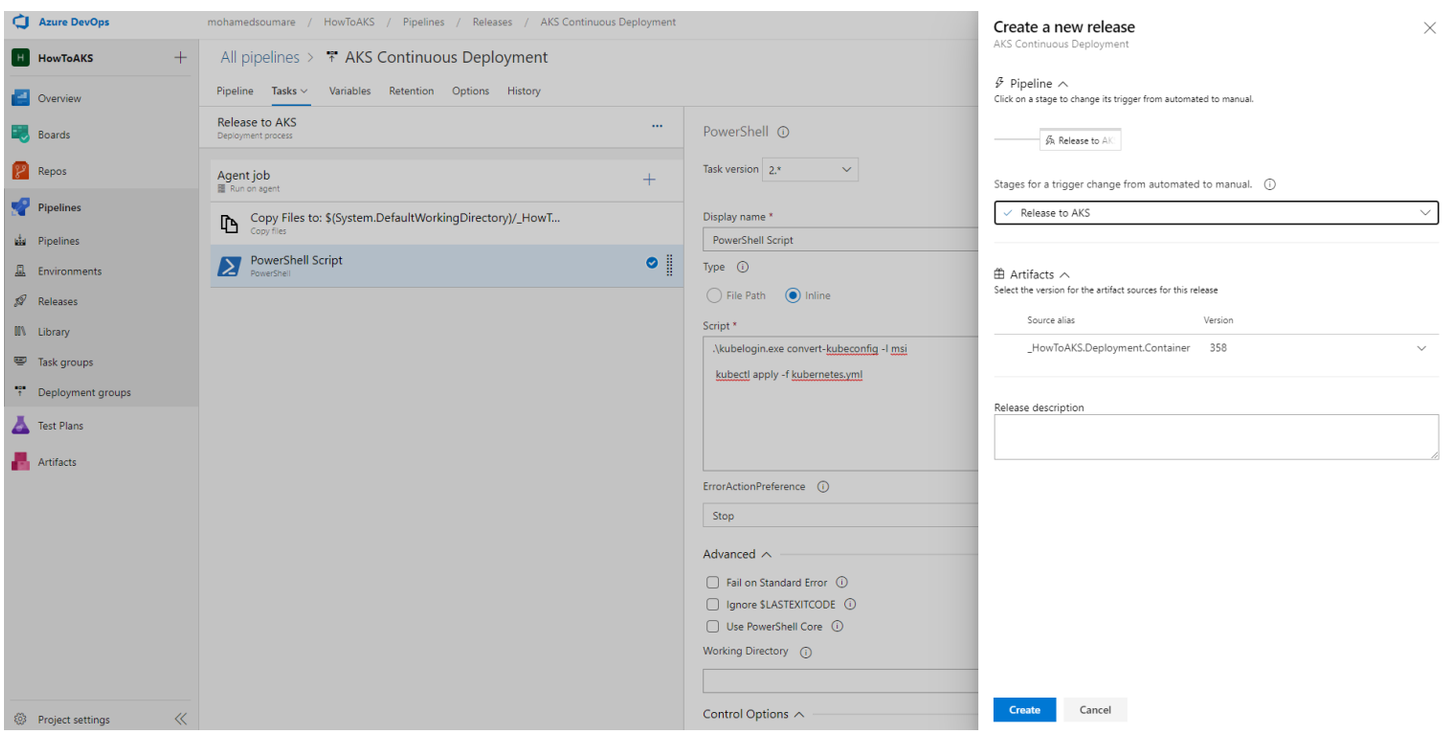


Step 13 > Set the PowerShell script type to inline > Paste the below script in the Script box > Set the Working Directory to the Build Kube directory we specified earlier in our Container Build pipeline.

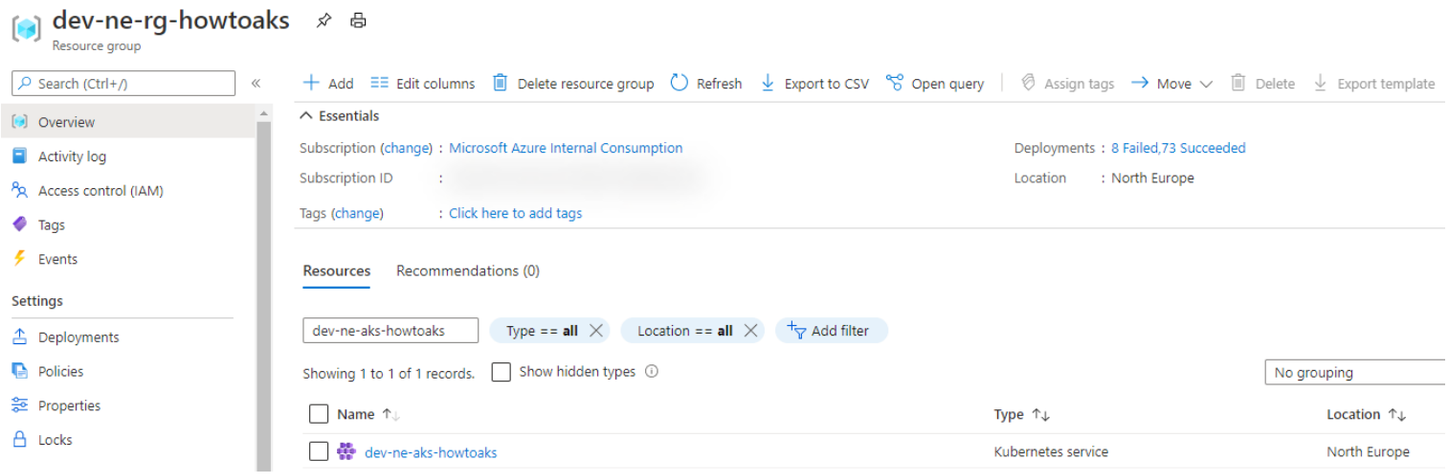
```
.\kubelogin.exe convert-kubeconfig -l msi
kubectl apply -f kubernetes.yml
```



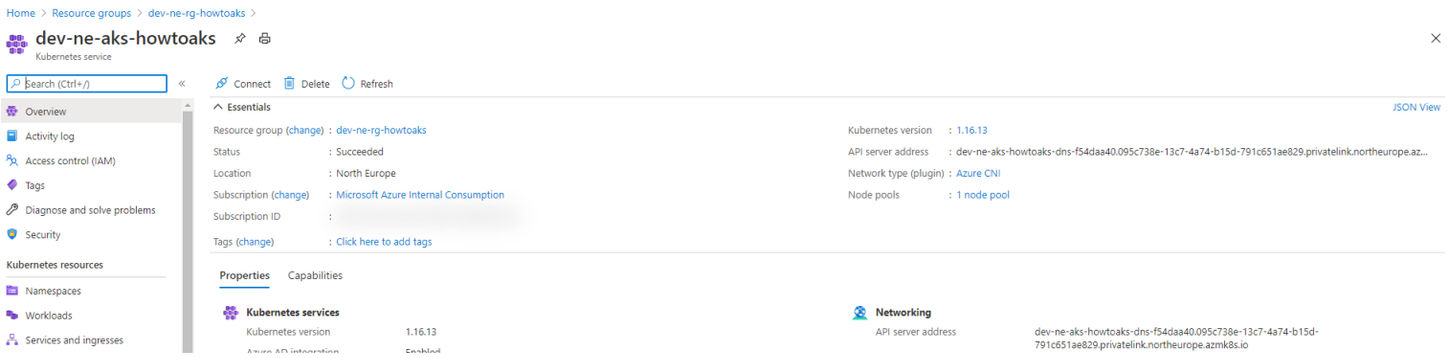
Step 14 > Save the pipeline > Create a new Release.

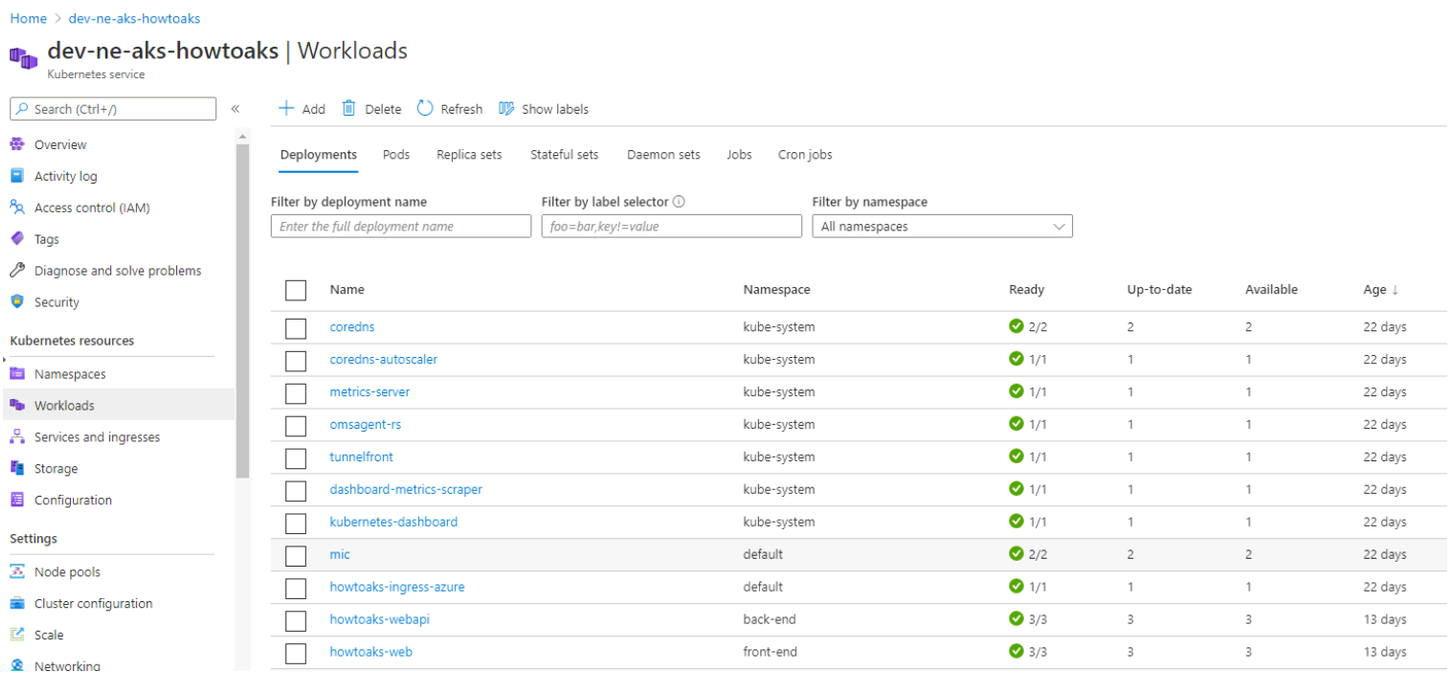


Step 15 > Navigate to the Azure Portal using your Management Box > Select your Resource Group > Select the AKS cluster.

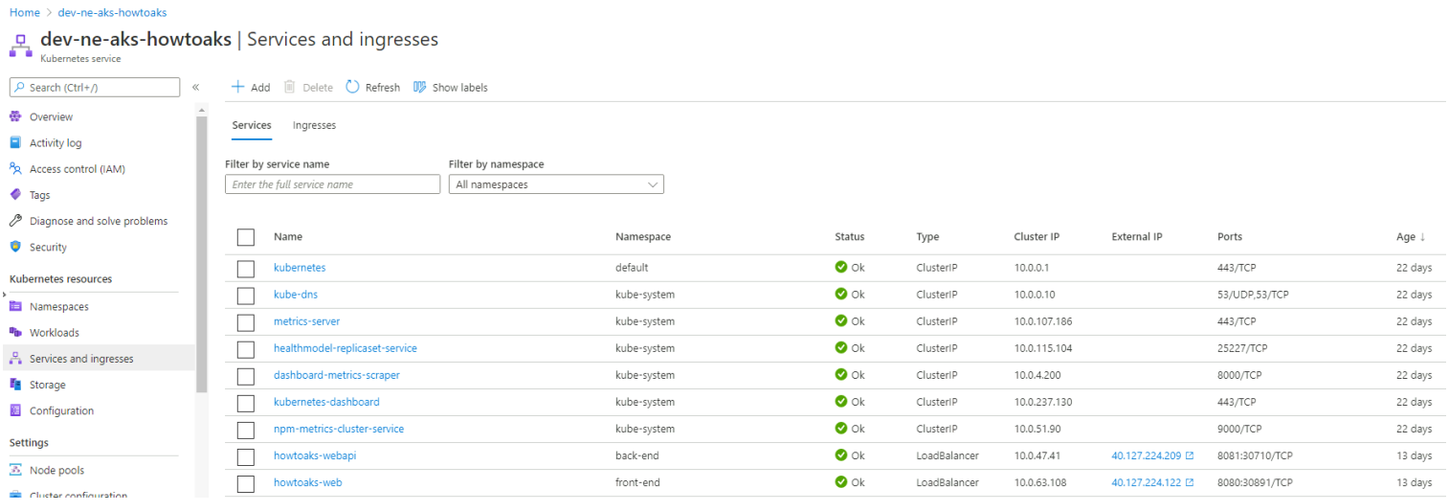


Step 16 > Select Workloads > You will see the newly deployed microservices (howtoaks-web & howtoaks-webapi).

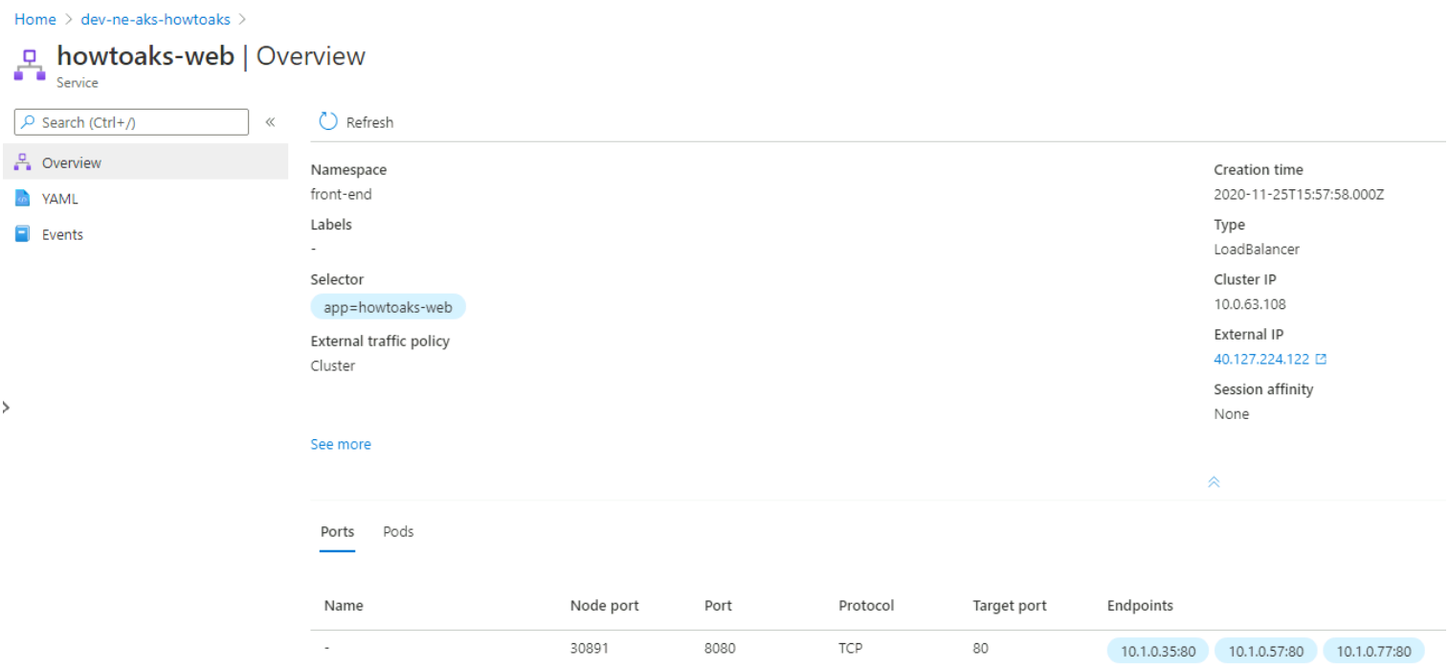




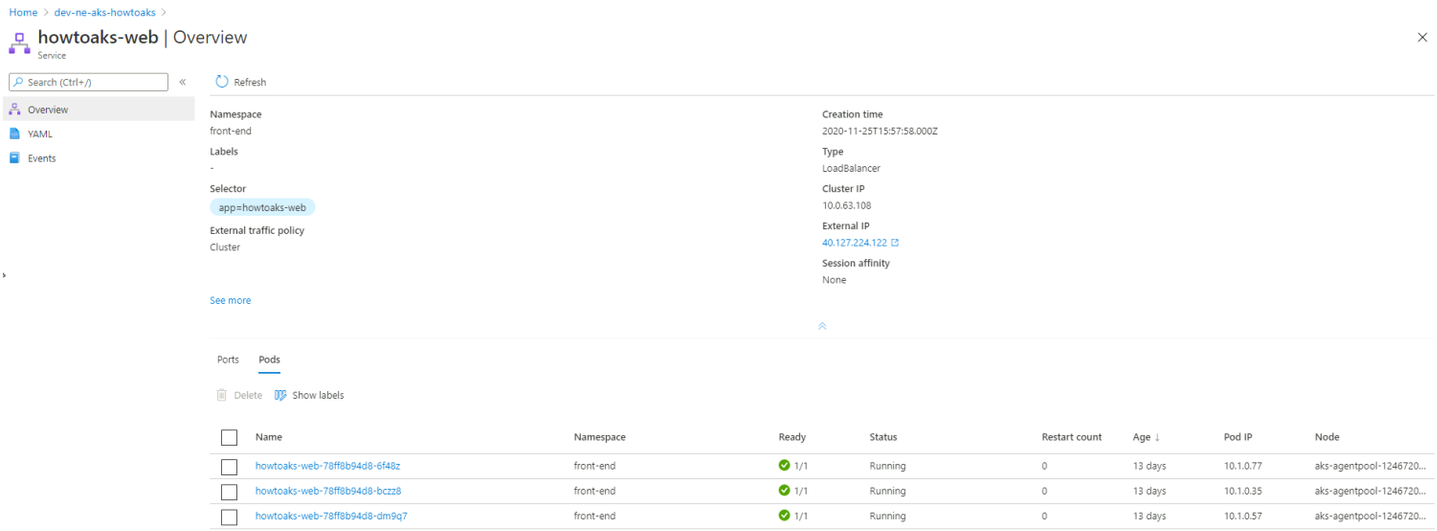
Step 17 > Select Services & Ingresses > You will see services with their external IP addresses (We will use the External IP for the front end microservice howtoaks-web in the Application Gateway).



Step 18 > Select the howtoaks-web service > You will see the ports and internal IP addresses.



Step 19 > Select Pods > You will see all the different Pods for the microservice.



Step 20 > Follow steps 18 & 19 to see the same information for the microservice howtoaks-webapi.

10. Application Gateway

So far, the microservices in our AKS cluster are exposed through the Internet using a Transport Load Balancer (OSI layer 4 - TCP and UDP). To add more protection to our microservices from external threats we will deploy Azure Application Gateway which is an Application Load Balancer (OSI layer 7) with SSL/TLS with aWeb Application Firewall (WAF) in Prevention mode.

Note: The recommend approach is to install the WAF in Detection mode first for a short period of time in a production environment and inspect the firewall logs so that you can mitigate any exceptions, update rules, then you can transition to Prevention mode.

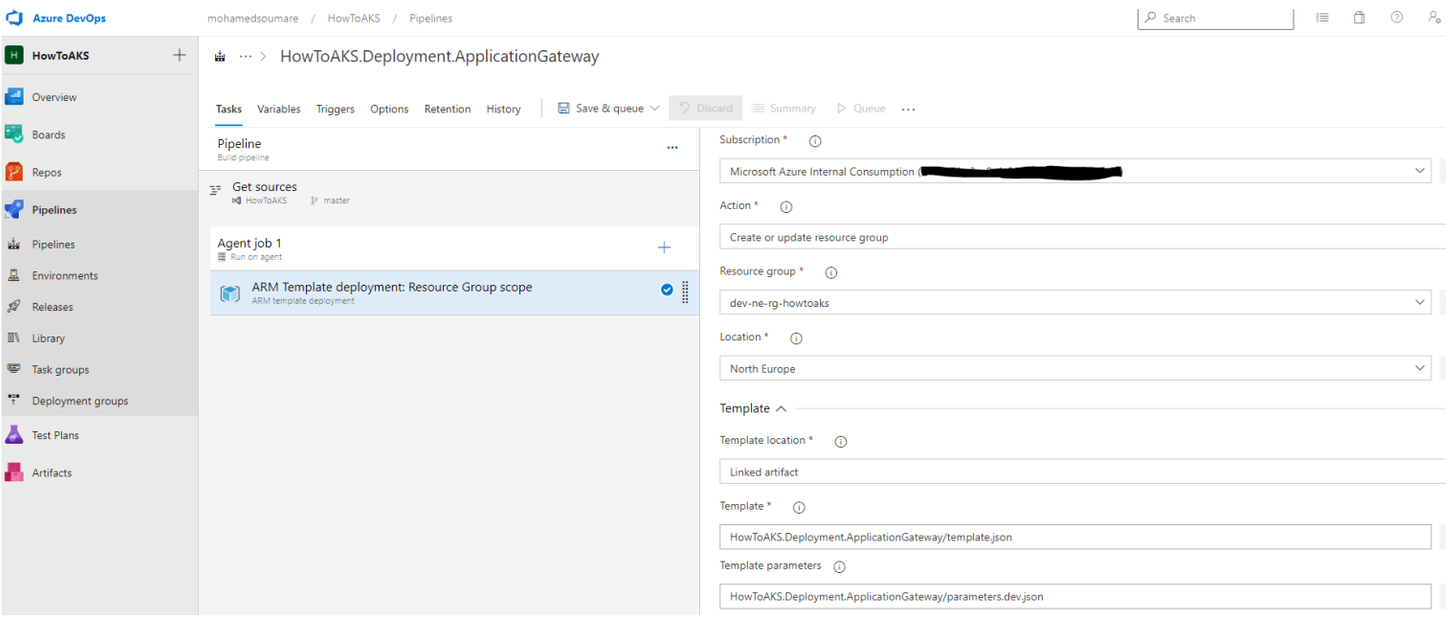
Using Infrastructure as Code with Azure DevOps for CI/CD, we will deploy our Application Gateway with Web Application Firewall v2. Please find the reusable ARM templates in the following GitHub repository

Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/6-applicationgateway*

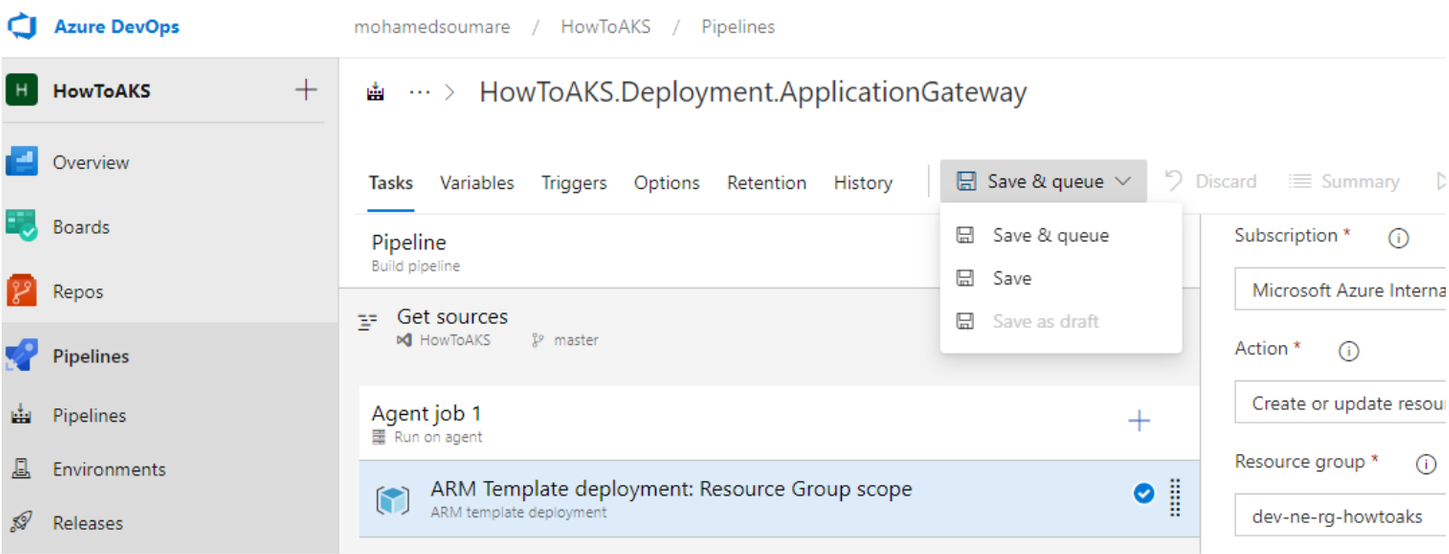
10.1. Deployment with Azure DevOps & CI/CD

Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.ApplicationGateway).

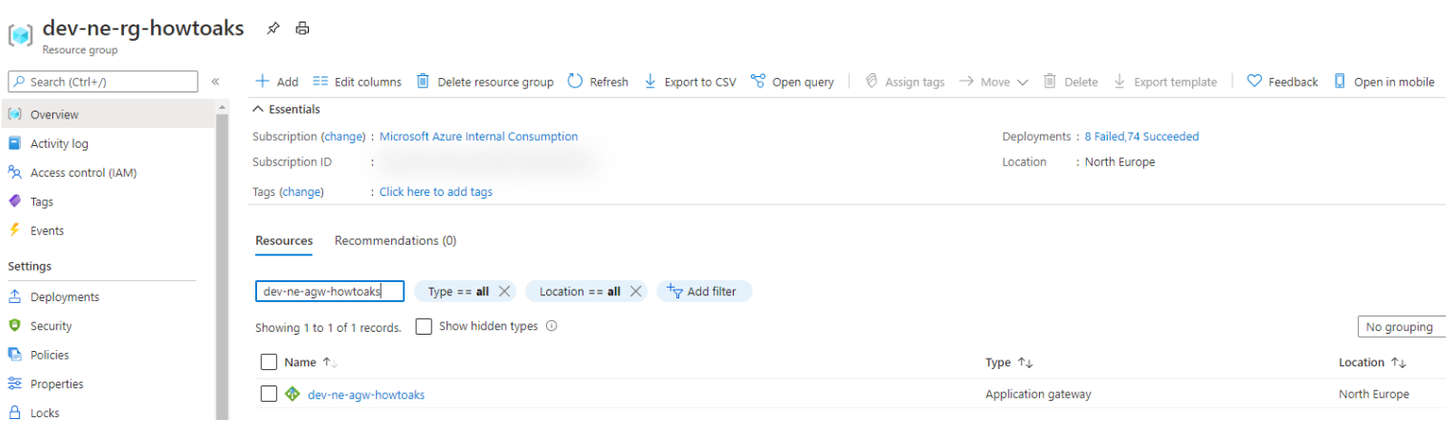
Step 11 > Select your Subscription, Resource Group, Location, Template and Template Parameters.



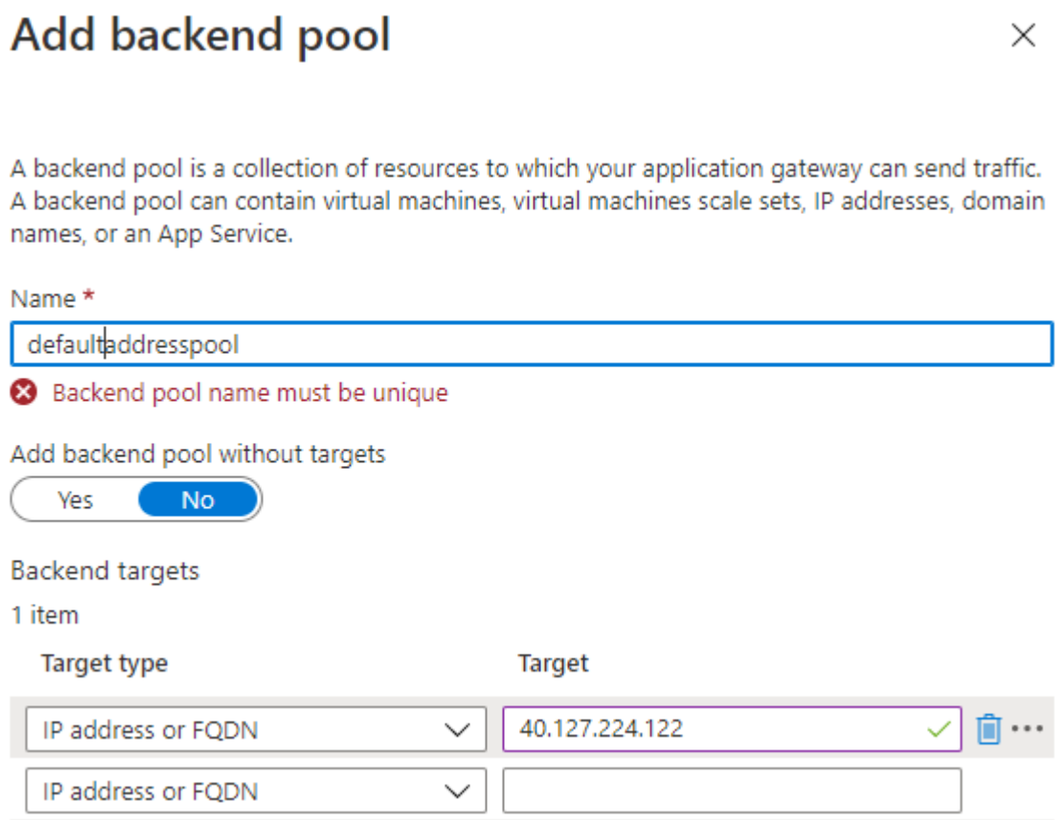
Step 12 > Select Save & Queue to launch the deployment of the Application Gateway.



Step 13 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the Application Gatway.



Step 14 > Select BackEnd Pools and create a new address pool. The target of the backend pool should the IP address of the microservice deployed in our AKS cluster. In our scenario it's the Web App Microservice IP address.



Step 15 > Select Health Probes and create a new Health Probe for both HTTP & HTTPS.

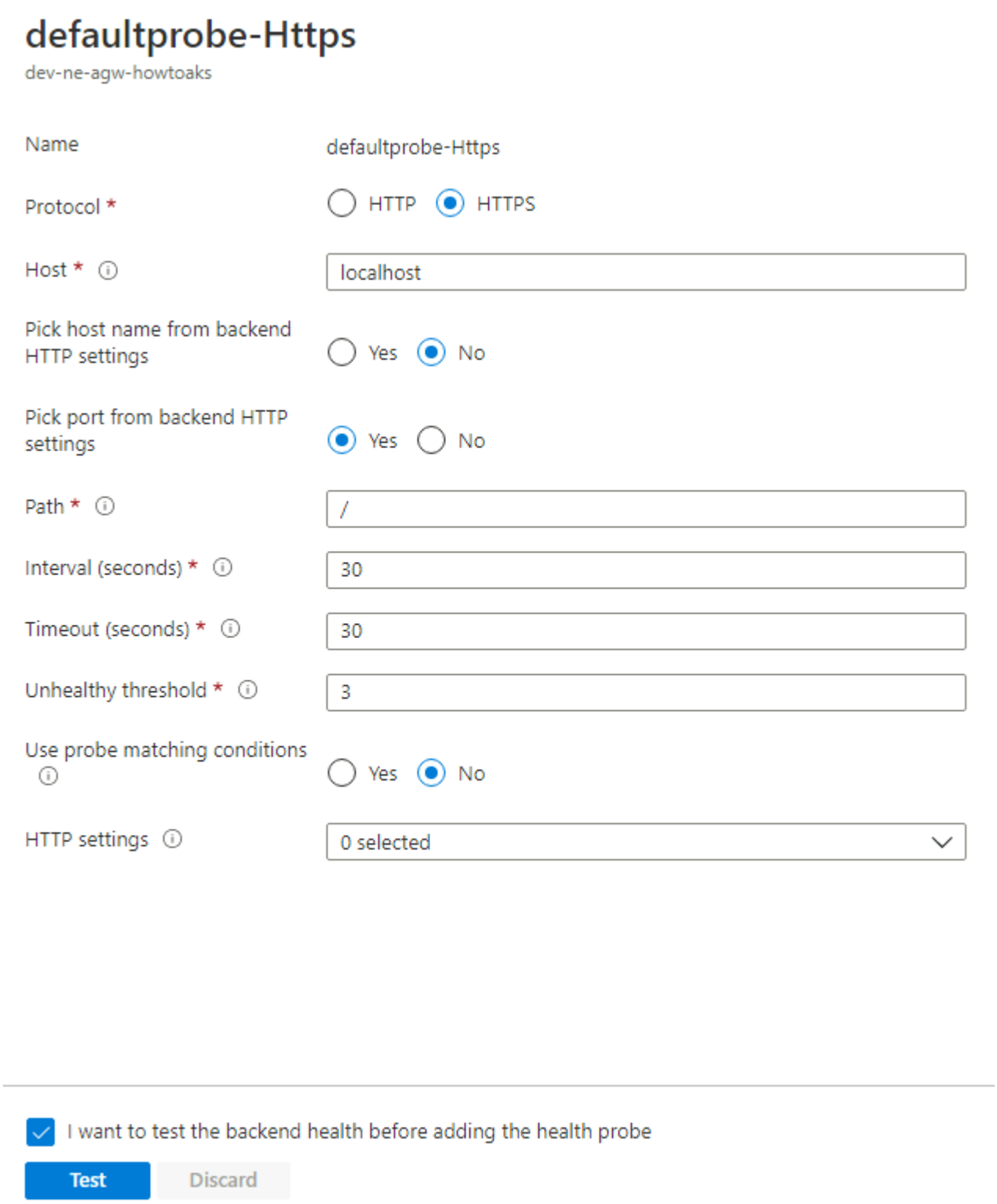
defaultprobe-Http

dev-ne-agw-howtoaks

Name	defaultprobe-Http
Protocol *	(●) HTTP ( ) HTTPS
Host *	localhost
Pick host name from backend HTTP settings	( ) Yes (●) No
Pick port from backend HTTP settings	(●) Yes ( ) No
Path *	/
Interval (seconds) *	30
Timeout (seconds) *	30
Unhealthy threshold *	3
Use probe matching conditions	( ) Yes (●) No
HTTP settings	defaulthttpsetting

[x] I want to test the backend health before adding the health probe

Test Discard



Step 16 > Select HTTP Settings > You will see the newly created HTTP Setting from the deployment of our Application Gateway earlier through CI/CD. We need to open it and update the Port (our Web App microservice Port) and the Custom Probe (The probe we created in the previous section).

Add HTTP setting

Connection draining

Enable Disable

Request time-out (seconds) *

30

Override backend path

Host name

By default, Application Gateway does not change the incoming HTTP host header f
unaltered to the backend. Multi-tenant services like App service or API managemei
extension to resolve to the correct endpoint. Change these settings to overwrite th

Override with new host name

Yes No

Host name override

Pick host name from backend target

Override with specific domain name

e.g. contoso.com

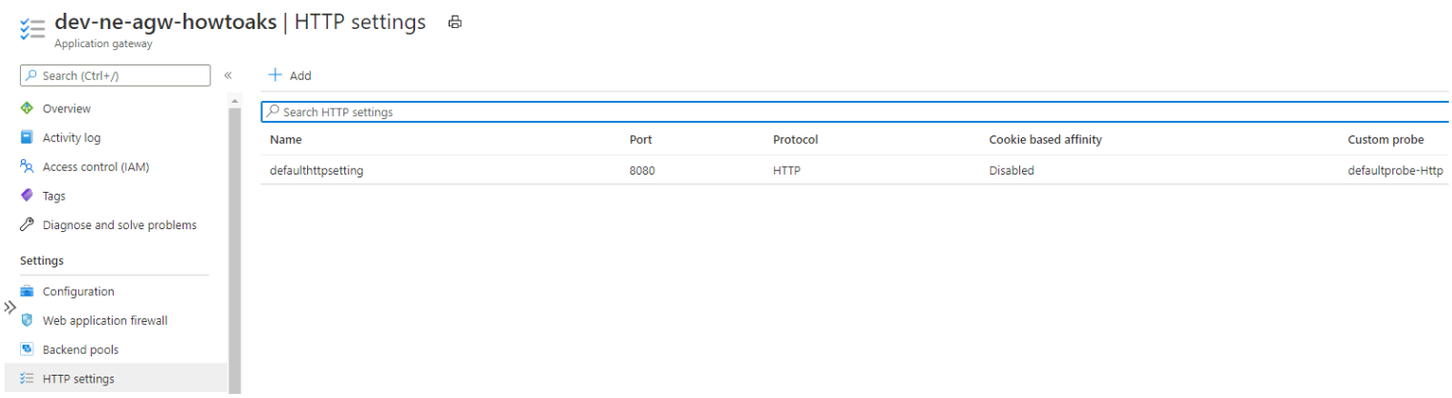
Use custom probe

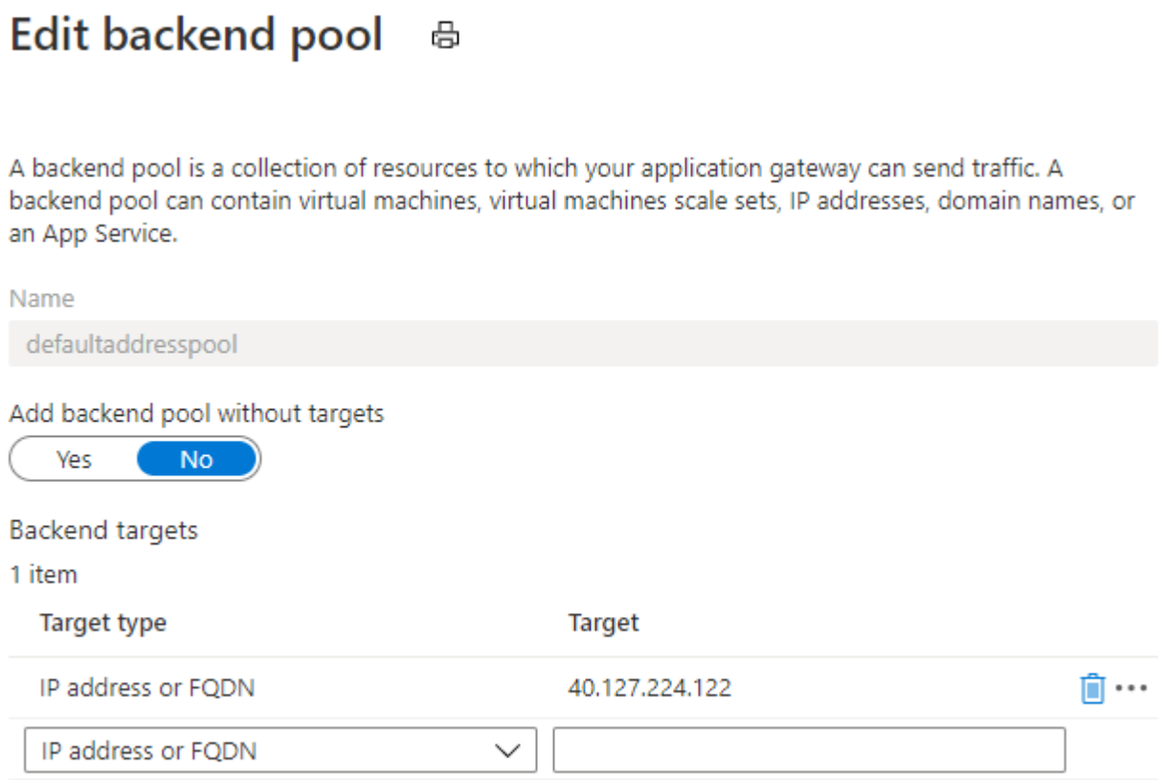
Yes No

Custom probe *

defaultprobe-Http

Save Cancel





11. Azure AD Pod Identity & Application Gateway Ingress Controller (AGIC)

To couple our AKS cluster with Application Gateway, we will deploy an Application Gateway Ingress Controller (AGIC), which is a Kubernetes application that runs in its own pod. This ingress controller monitors our Application Gateway and performs automatically all the necessary configurations needed for it.

You can learn more about AGIC with following link https://docs.microsoft.com/en-us/azure/application-gateway/ingress-controller-overview.

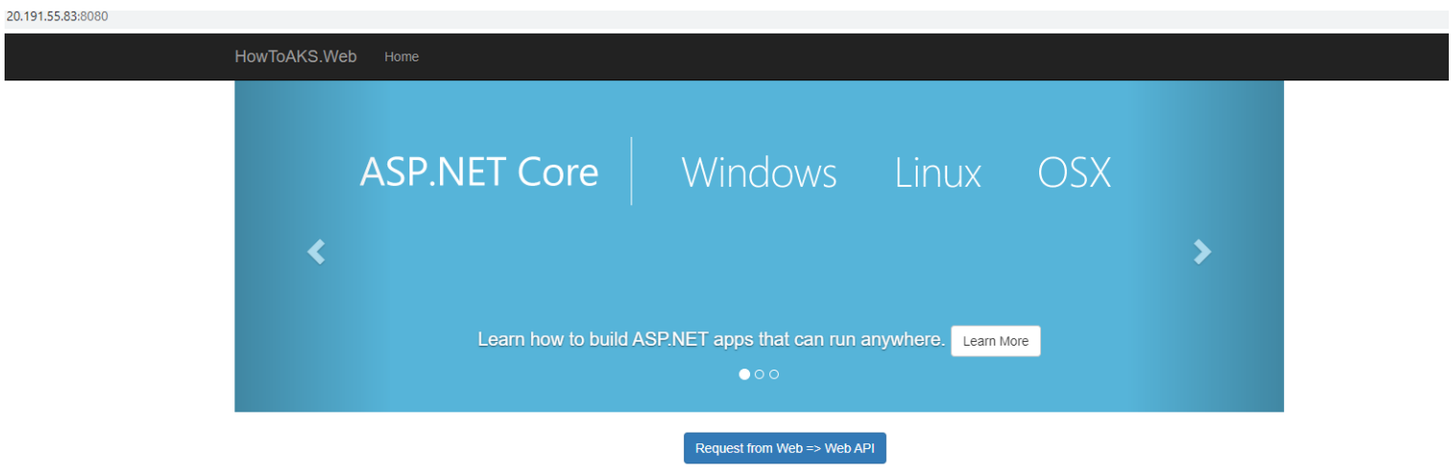
Using Infrastructure as Code declaratively, we have included a PowerShell script combined with a sample of helm config file for AGIC, that you can execute to easily configure your AGIC application. Please find the reusable ARM templates in the following GitHub repository.

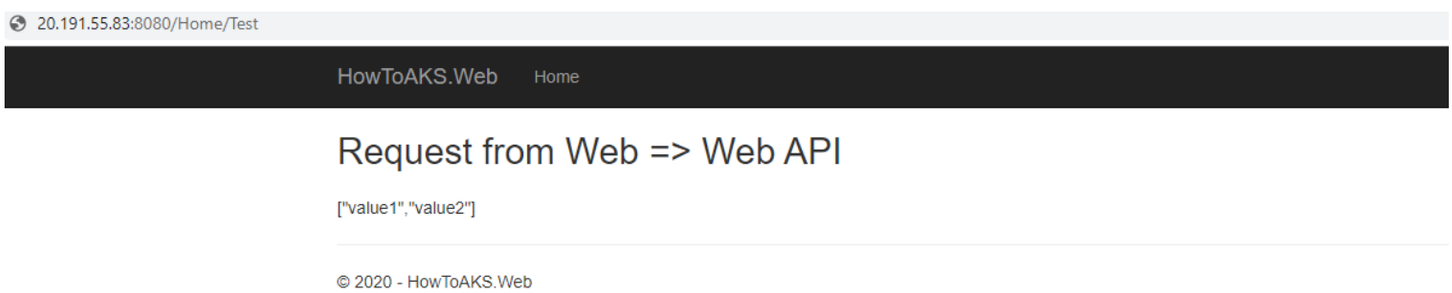
Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/7-azuread-podidentity-agic*

12. DNS Zone

After deploying all components, we can test the connectivity to our application by using the Public IP Address of our Application Gateway with the port number of our Web App microservice deployed in AKS.

In our scenario our public IP address is 20.191.55.83 and the port number of our Web APP microservice is 8080. Of course, you will have different values when you are implementing this guide 😊. When we click on the button "Request from Web => Web API" we are making a call between the Web APP microservice to the Web API APP microservice running inside AKS cluster.





Instead of accessing our application by IP Address we can purchase a domain name and configure it with Azure DNS or with another external DNS provider.

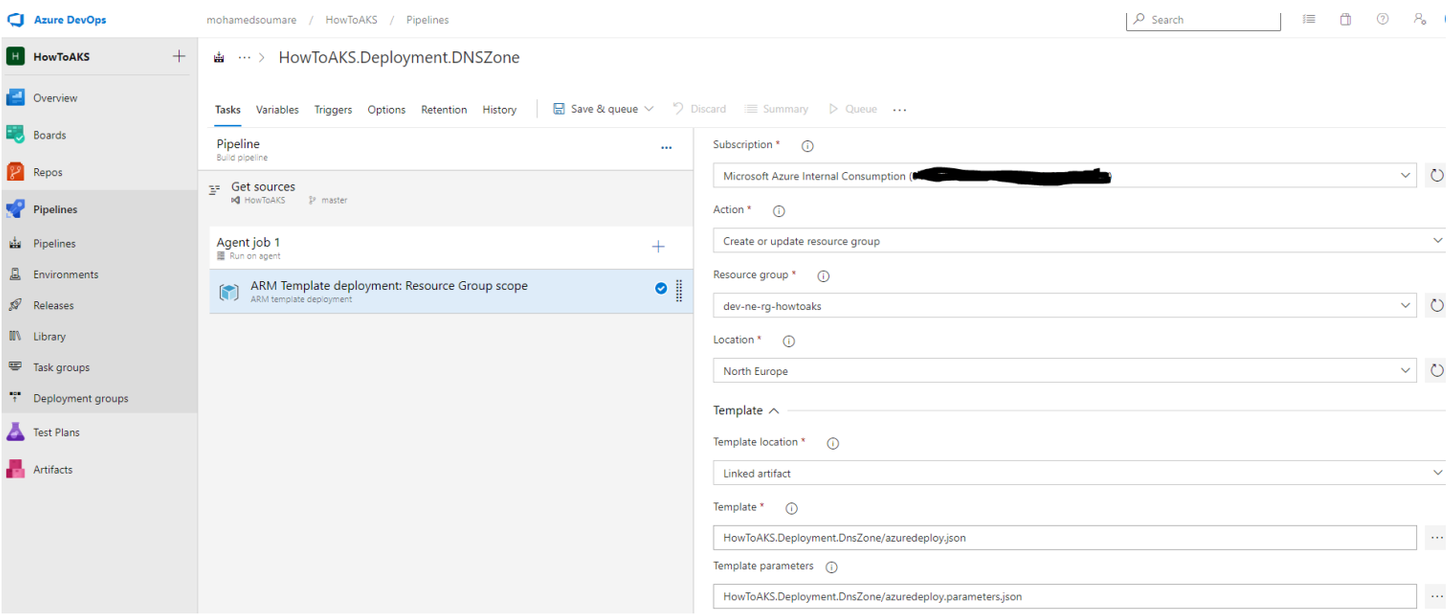
Using Infrastructure as Code with Azure DevOps for CI/CD, we will deploy our Azure DNS. Please find the reusable ARM templates in the following GitHub repository

Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/8-dnszone*

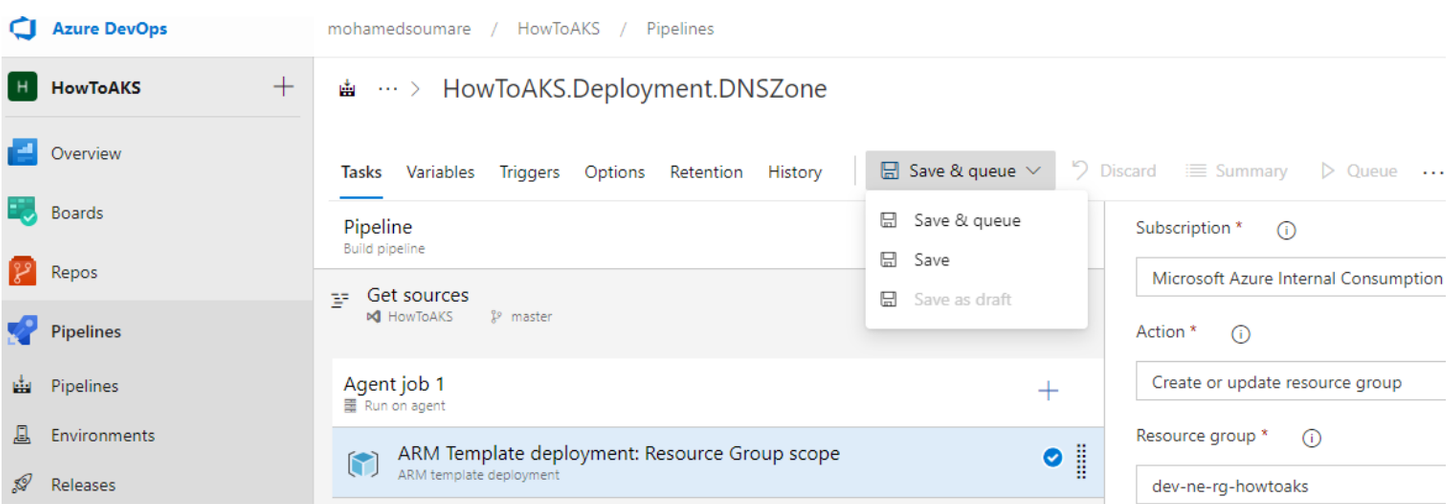
12.1. Deployment with Azure DevOps & CI/CD

Repeat Steps 1 through 10 from the previous topic and update the pipeline name on Step 6 (e.g., HowToAKS.Deployment.DNSZone).

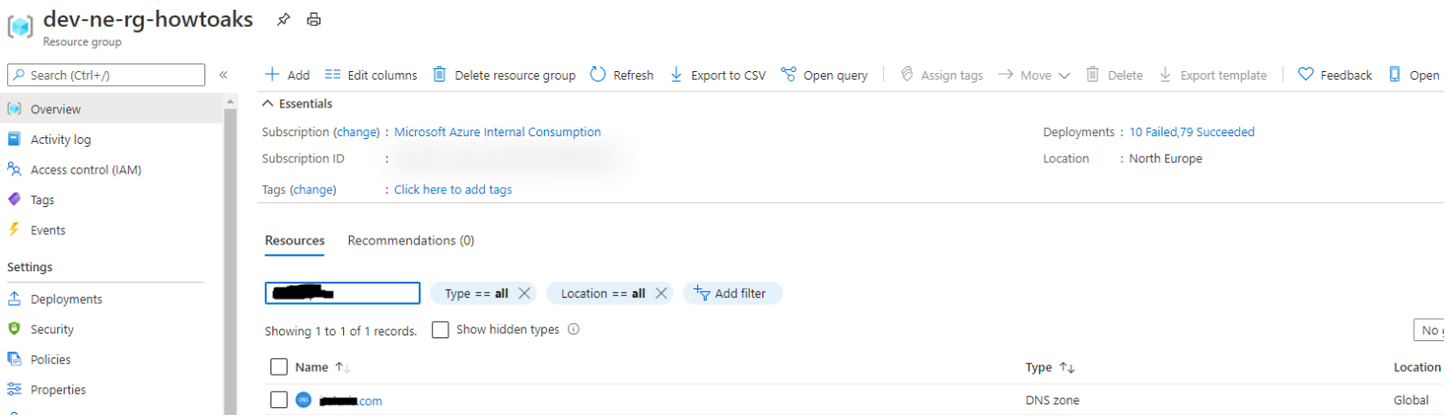
Step 11 > Select your Subscription, Resource Group, Location, Template and Template Parameters.

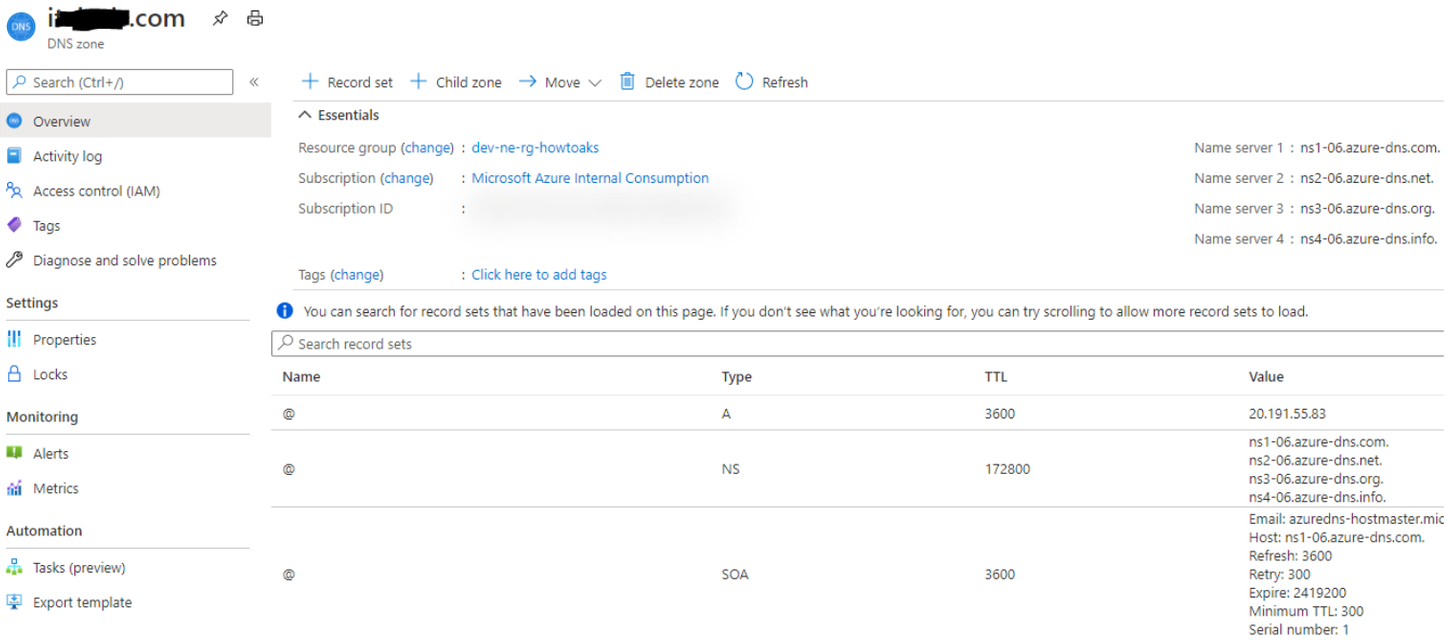


Step 12 > Select Save & Queue to launch the deployment of the DNS Zone.

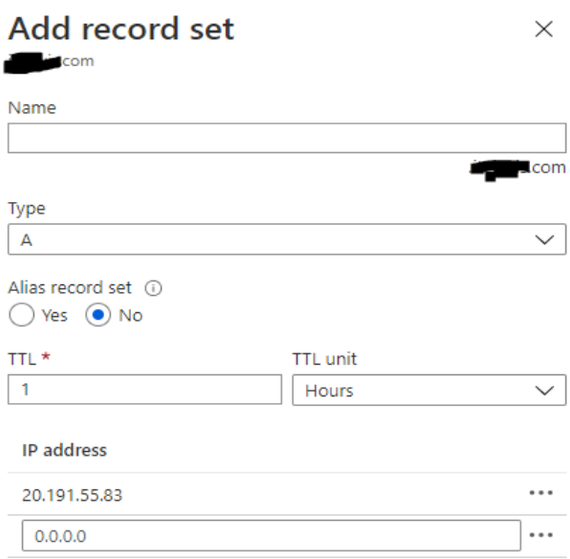


Step 13 > After the deployment is finished > Navigate to the Azure Portal > Select your Resource Group > Select the Azure DNS Zone.

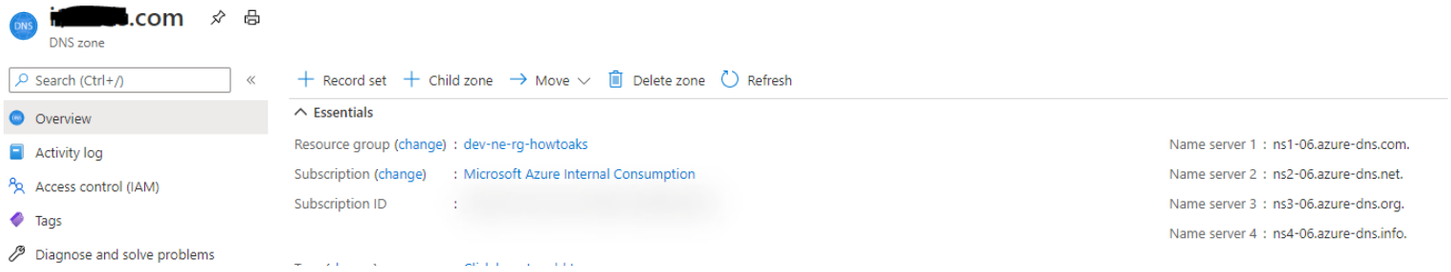




Step 14 > Select Recordset to create DNS records (A record and CNames).

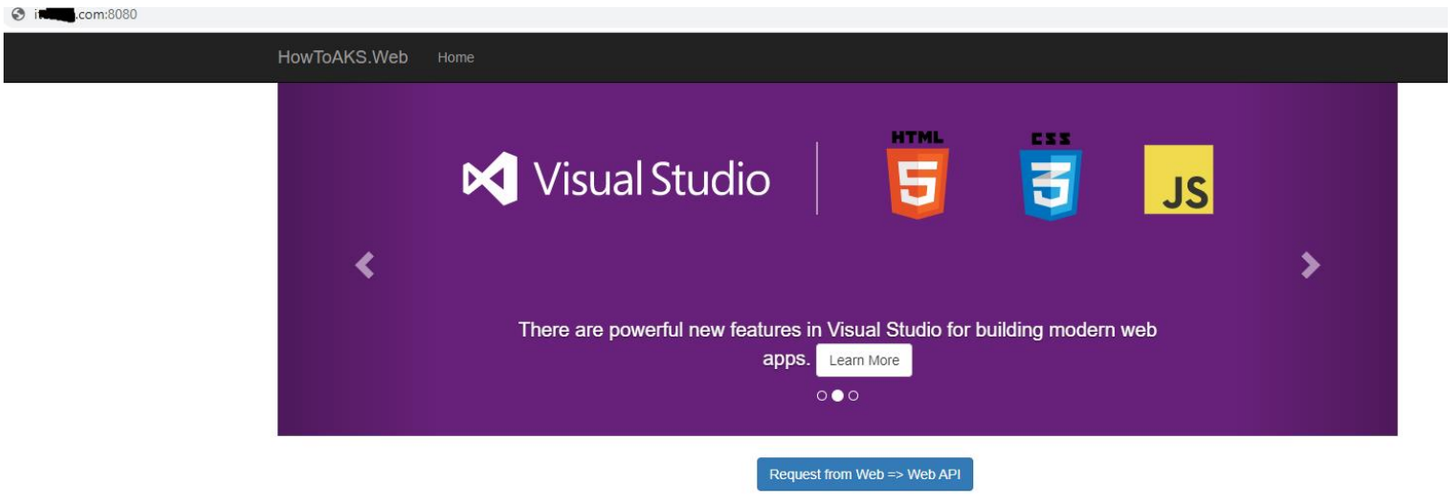


Step 15 > After creating the DNS records, you must update the names servers from the Domain Registrar where you purchased your domain name.



Step 16 > When finished you can now access your microservices apps using the custom domain name URL instead of using the Public IP address of our Application Gateway.

When we click on the button "Request from Web => Web API" we are making a call between the Web APP microservice to the Web API APP microservice running inside AKS cluster.



13. Bonus

As a bonus, we can deploy an automatic certificate management environment, which will allow us to automatically renew the certificate in the Application Gateway by using Let's Encrypt. Please find the reusable ARM templates in the following GitHub repository. You can learn more about ACME using the following link https://samcogan.com/lets-encrypt-certificates-in-azure-with-acmebotot/

Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/9-acme*

You can also deploy all Azure components discussed in this whitepaper using Terraform.
Please find the reusable ARM templates in the following GitHub repository

Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/terraform*

Sample Web App and Web API App with Docker Containers illustrating the microservices.

Infrastructure as Code - *https://github.com/msoumare/HowToAKS/tree/main/app*

Microsoft Azure