NewgenONE OmniDocs

Configuration and Deployment Guide for Azure

Version: 11.3

Newgen Software Technologies Ltd.

Table of Contents

1 Preface

This guide describes the deployment of OmniDocs deliverables like OmniDocs Docker containers and its required configuration files on the Azure Kubernetes Service (AKS).

1.1 Revision history

Revision Date	Description
July 2024	Initial publication

1.2 Intended audience

This guide is intended for System Administrators, developers, and any other users seeking information about the deployment of OmniDocs containers on Azure Kubernetes Services. The reader must be comfortable to understand the computer terminology.

1.3 Documentation feedback

To provide feedback or any improvement suggestions on technical documentation, you can write an email to docs.feedback@newgensoft.com.
To help capture your feedback effectively, request you to share the following information in your email.

- Document Name
- Version
- Chapter, Topic, or Section
- Feedback or Suggestions

1.4 Third-party product information

This guide contains third-party product information about configuring Microsoft Azure CICD Pipeline for Container Deployment on AKS Azure Kubernetes Cluster. Newgen Software Technologies Ltd does not claim any ownership on such third-party content. This information is shared in this guide only for convenience of our users and could be an excerpt from the Azure documentation. For latest information on configuring the Azure Kubernetes Cluster and Azure DevOps refer to the Azure documentation.

2 Configuration of Azure Kubernetes cluster

This section contains the steps to configure the Kubernetes Cluster on Azure.

2.1 Create an Azure Kubernetes cluster

This section explains how to create an Azure Kubernetes Cluster:

Pre-requisites:

Following are the prerequisites for Azure Kubernetes Cluster creation:

- Signed in user must have below roles:
  - At Subscription: Contributor Role
  - At Subscription: User Access Administrator
- Virtual network and subnet must be created for the Kubernetes cluster.

Before creating the Azure Kubernetes Cluster also known as AKS, you must sign in to the Azure portal at *https://portal.azure.com*.

Perform the below steps to create an Azure Kubernetes Cluster:

1. On the Azure portal menu or from the Home page, select **Create a resource**.
2. Select **Containers** and **Kubernetes Service**.

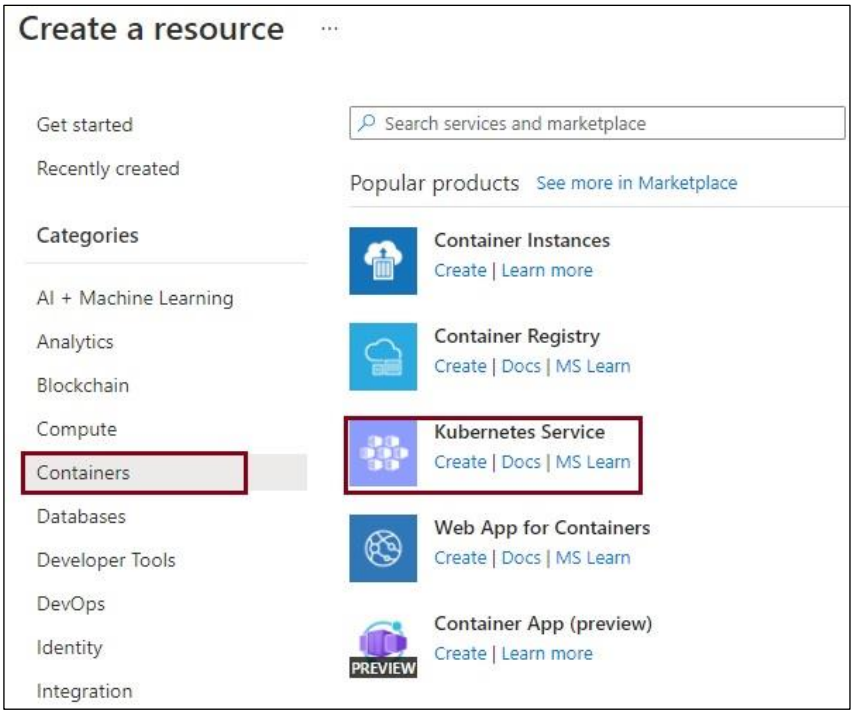

Figure 2.1

3. In the **Basics** tab, specify the following details on the Create Kubernetes cluster:
   - **Subscription**: Select a valid Azure subscription.
   - **Resource group**: Select or create an Azure Resource group, such as **AzureKubernetes**.
   - **Kubernetes cluster name**: Enter a Kubernetes cluster name such as **BPMSuite-AKSCluster**.
   - **Region**: Select a region into which you want to create an AKS cluster.
   - **Availability zones**: Usually there are three availability zones per region that allows you to spread the nodes across different physical locations for high availability. Select the availability zones as per your business requirement. [By default, select all the availability zones].
   - **Kubernetes version**: Select the default one that is, **1.20.9 (default)**.
   - **Primary node pool**: Select a VM Node size for the AKS nodes and select the number of nodes to be deployed into the AKS cluster.

NOTE:
The VM size can't be changed after the AKS cluster deployment. However, node count is adjustable.

   - **Scale method**: Select the scale method as **Autoscale**. Autoscaling helps to ensure that your cluster is running efficiently with the right number of nodes for the workloads present.
   - Click **Next: Node Pools**.

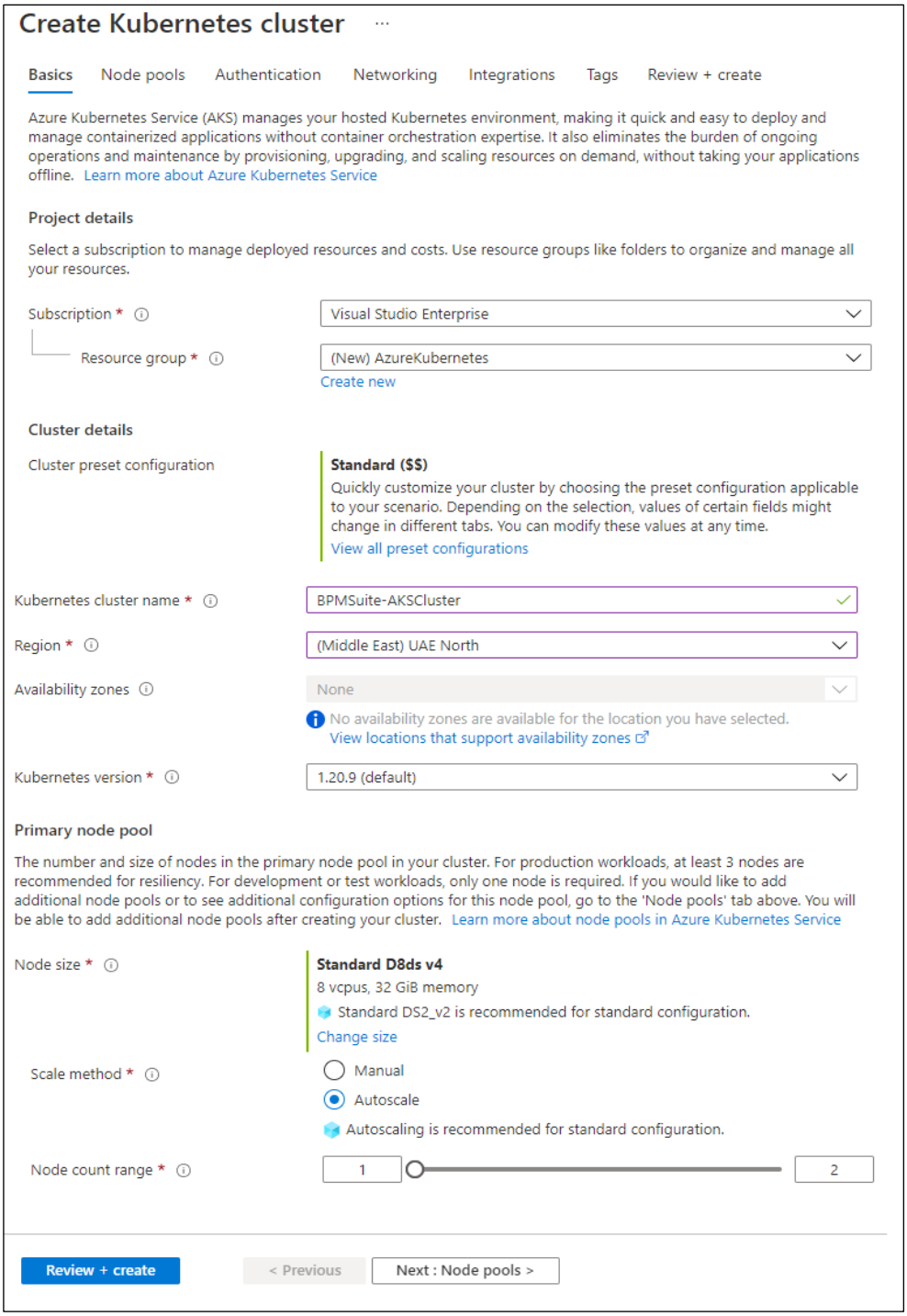

Figure 2.2

4. On the **Node pools** page, keep the default options and click on the **Next: Authentication>**.

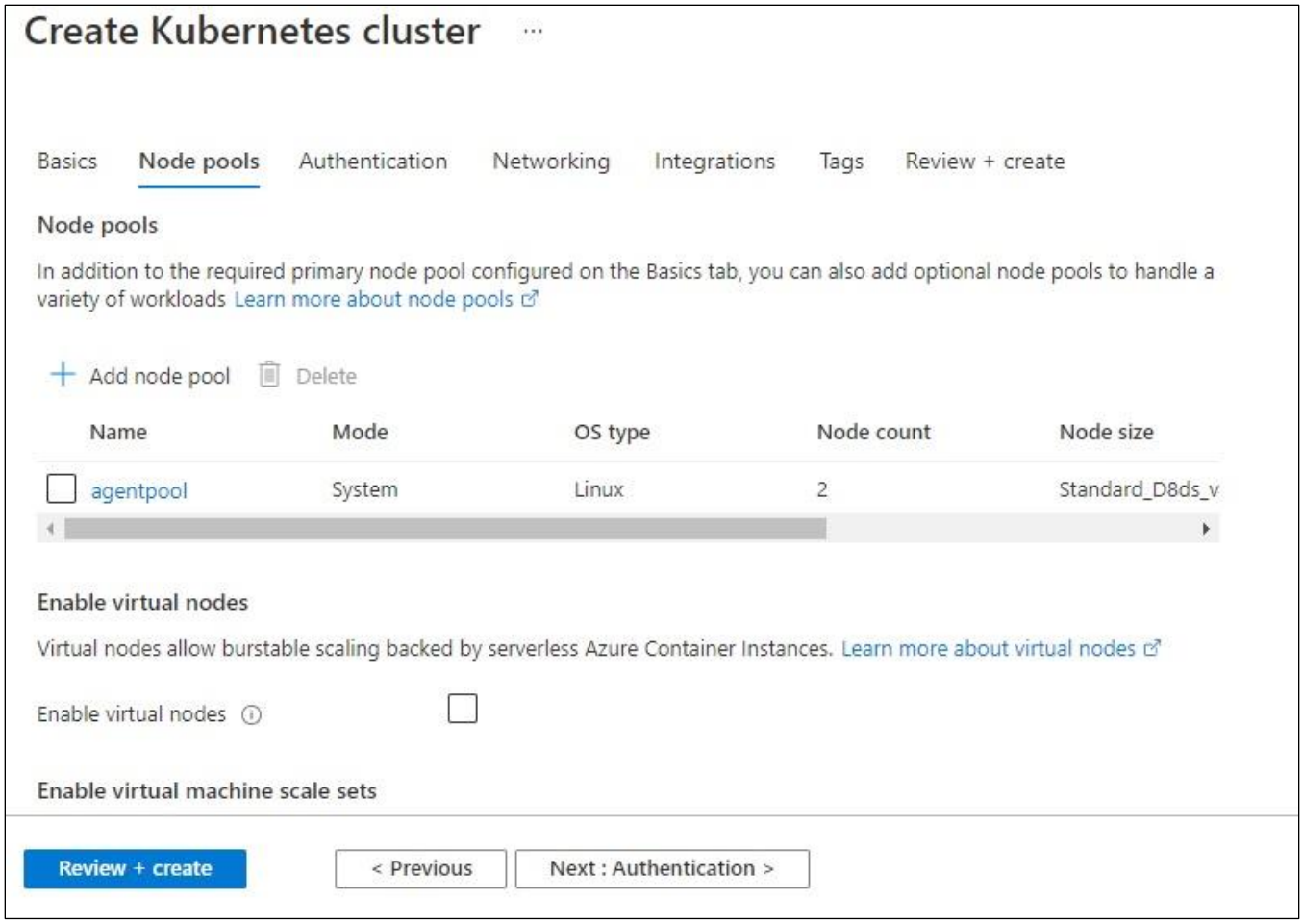

Figure 2.3

5. On the **Authentication** page, keep the default options and click on the **Next: Networking>**.

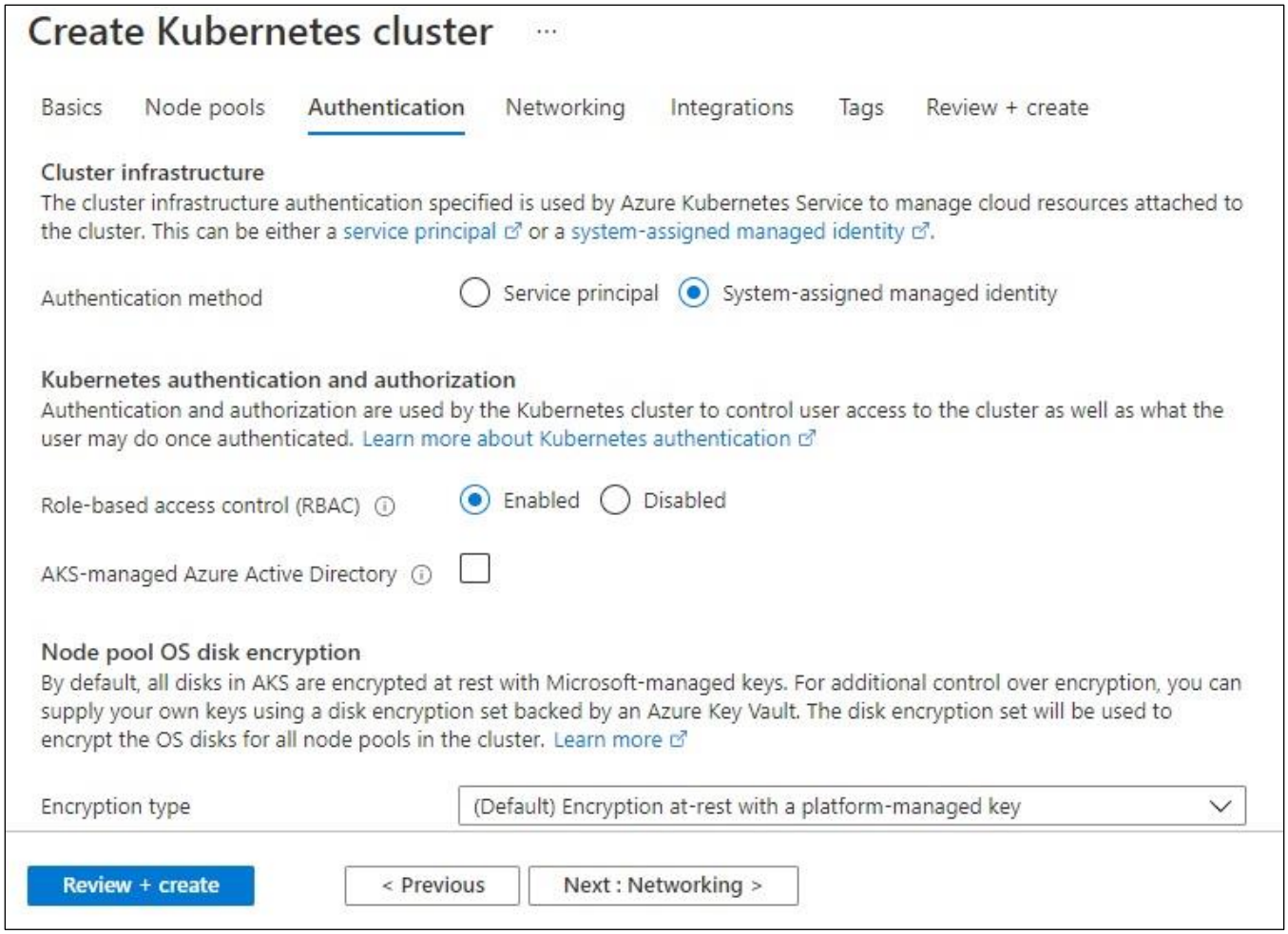

Figure 2.4

6. Select the **Azure CNI** as Network configuration and specify the following details:
   - **Virtual network:** Select the created VNet for this AKS cluster deployment that is, **Vnet_for_AzureKubernetes**.
   - **Cluster subnet:** Select the subnet into which both the nodes and containers in the cluster gets placed that is, subnet_dev (**10.0.2.0/23**).

NOTE:
This IP range 10.0.2.0/23 must be large enough to accommodate the nodes, pods, and all the Kubernetes resources that might be provisioned in your cluster.

- **Kubernetes service address range**: Specify the IP range from which you can assign Ips to the internal Kubernetes services. This range must not be connected to this virtual network, or it must not overlap with any Subnet IP ranges. For example: **10.0.0.0/25**.
- You can reuse this range across different AKS clusters.

- **Kubernetes DNS service IP address:** An IP address assigned to the Kubernetes DNS service. It must be within the Kubernetes service address range. For example: **10.0.0.10**.

NOTE:
Don't use the first IP address in your address range. The first address is used for the *10ubernetes.default.svc.cluster.local* address.

- **Docker Bridge address:** Docker bridge is not used by AKS clusters or the pods themselves, you must set this address to continue to support scenarios such as docker build within the AKS cluster. It is required to select a CIDR for the Docker bridge network address. Else, Docker picks a subnet automatically, which can conflict with other CIDRs. You must pick an address space that does not collide with the rest of the CIDRs on your networks, including the cluster's service CIDR and pod CIDR that is, **172.17.0.1/25**.
- You can reuse this range across different AKS clusters.
- Select **Azure** as **Network policy** and keep the other settings as default.
- Click **Next: Integrations>**.

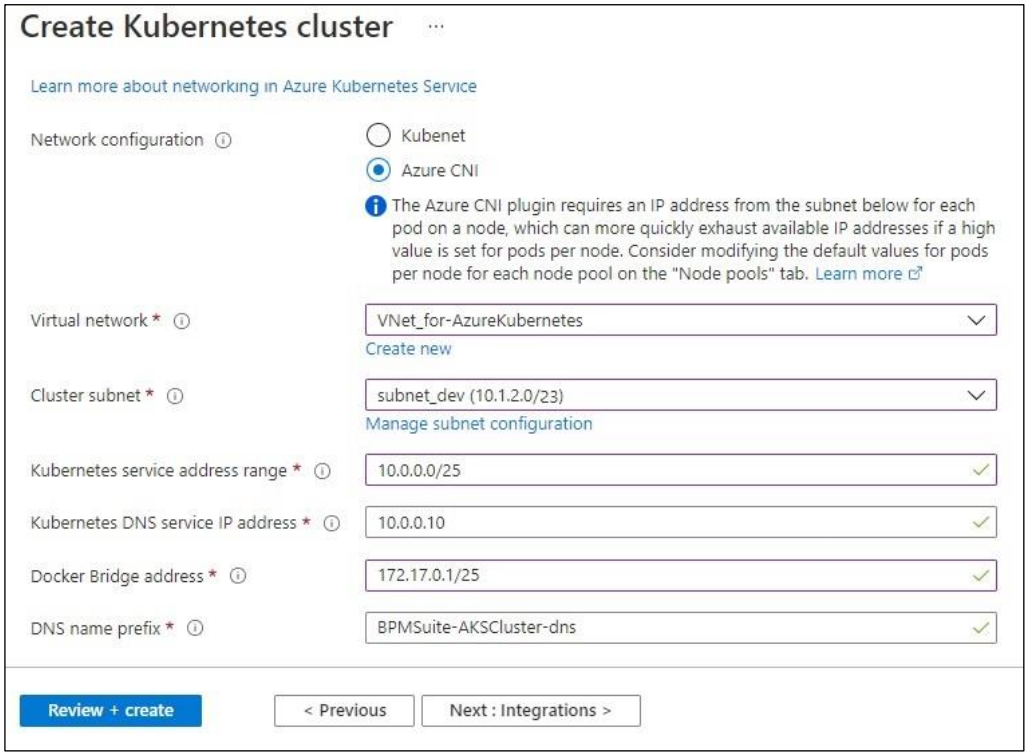

Figure 2.5

7. On the **Integration** page, keep the default options and click the **Next: Tags>**.

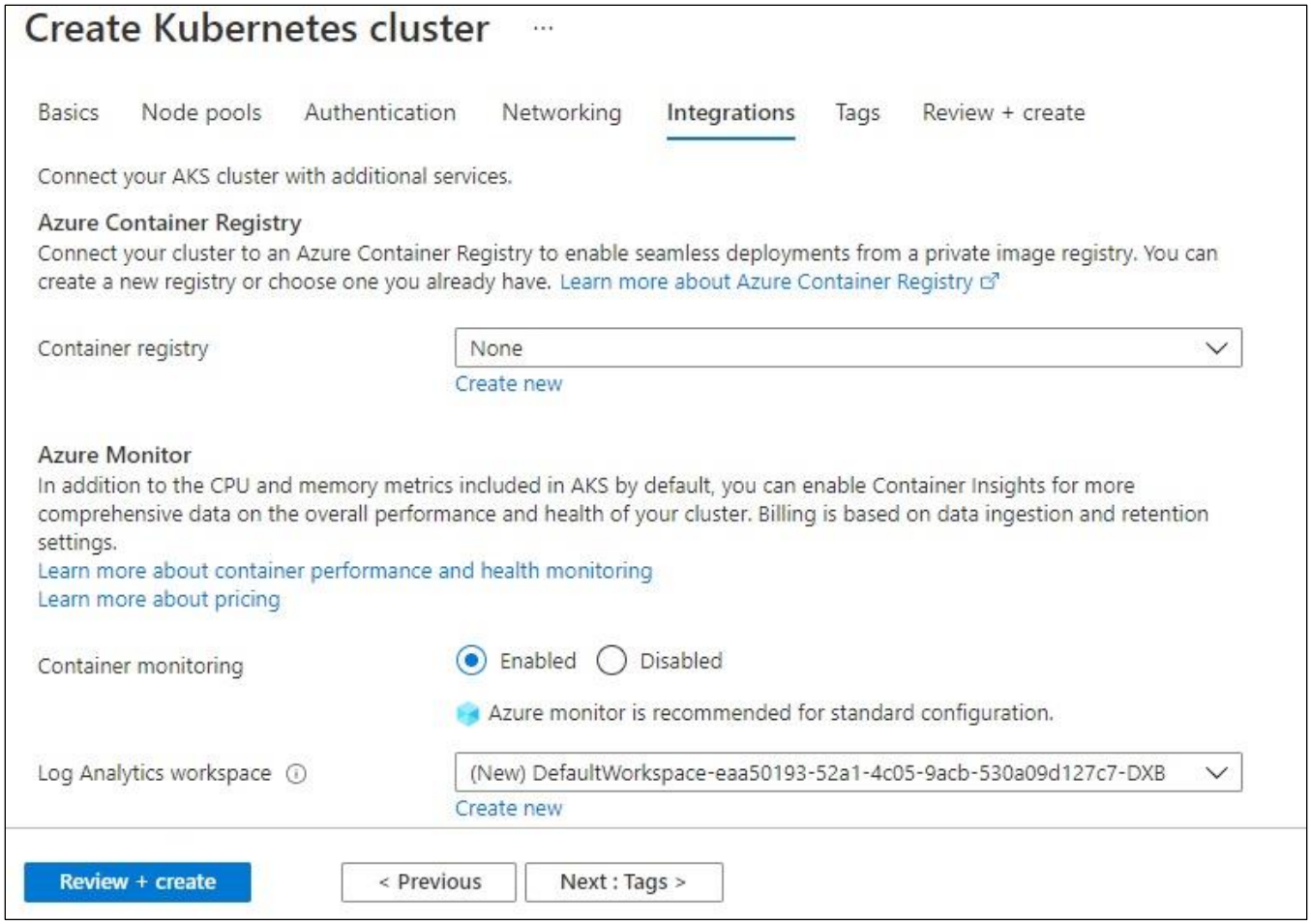

Figure 2.6

8. On the **Tags** page, keep the default options and click **Next: Review + create>**.

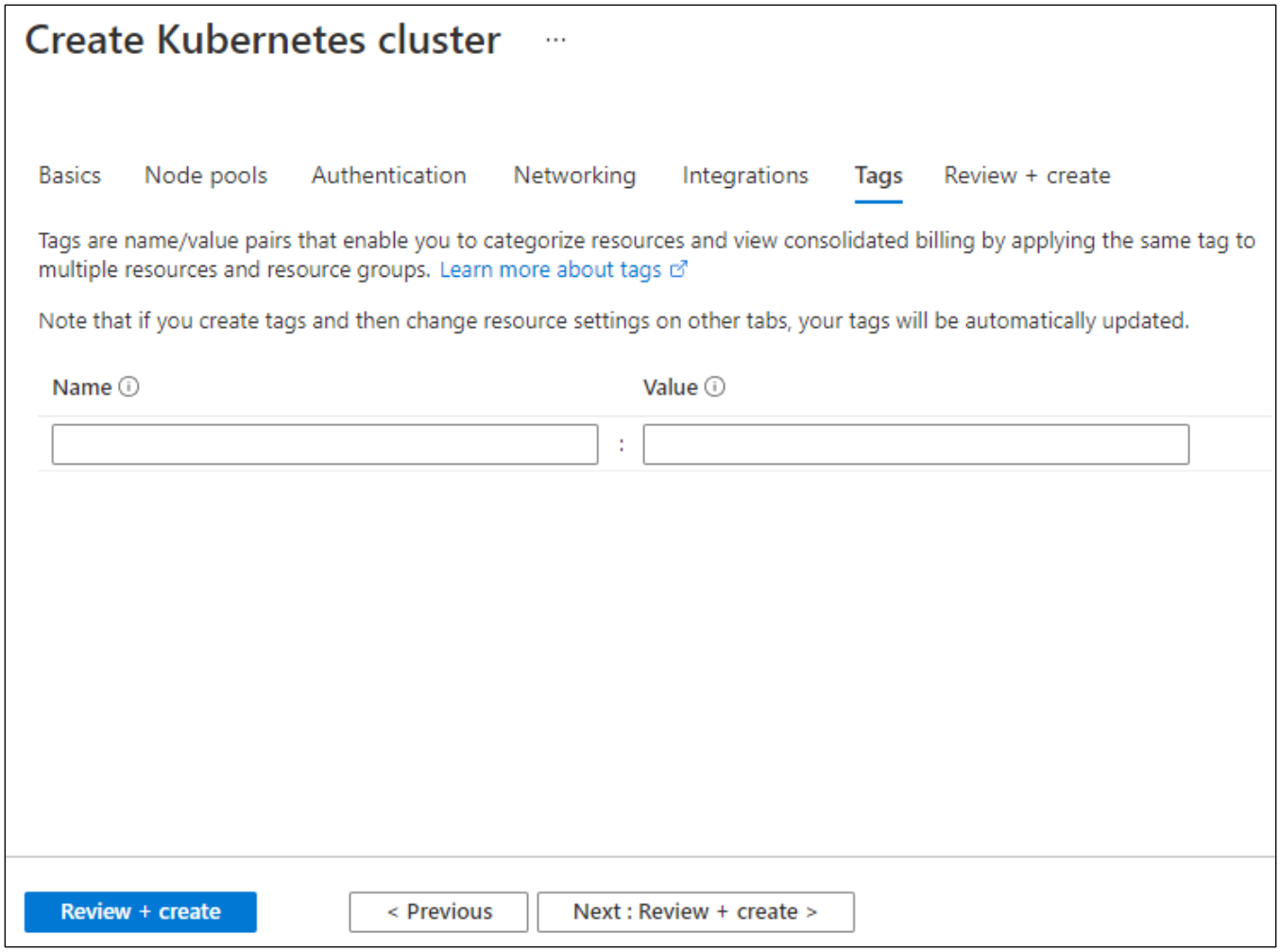

Figure 2.7

9. On the **Review + create** page, click **Create** once validation is passed.

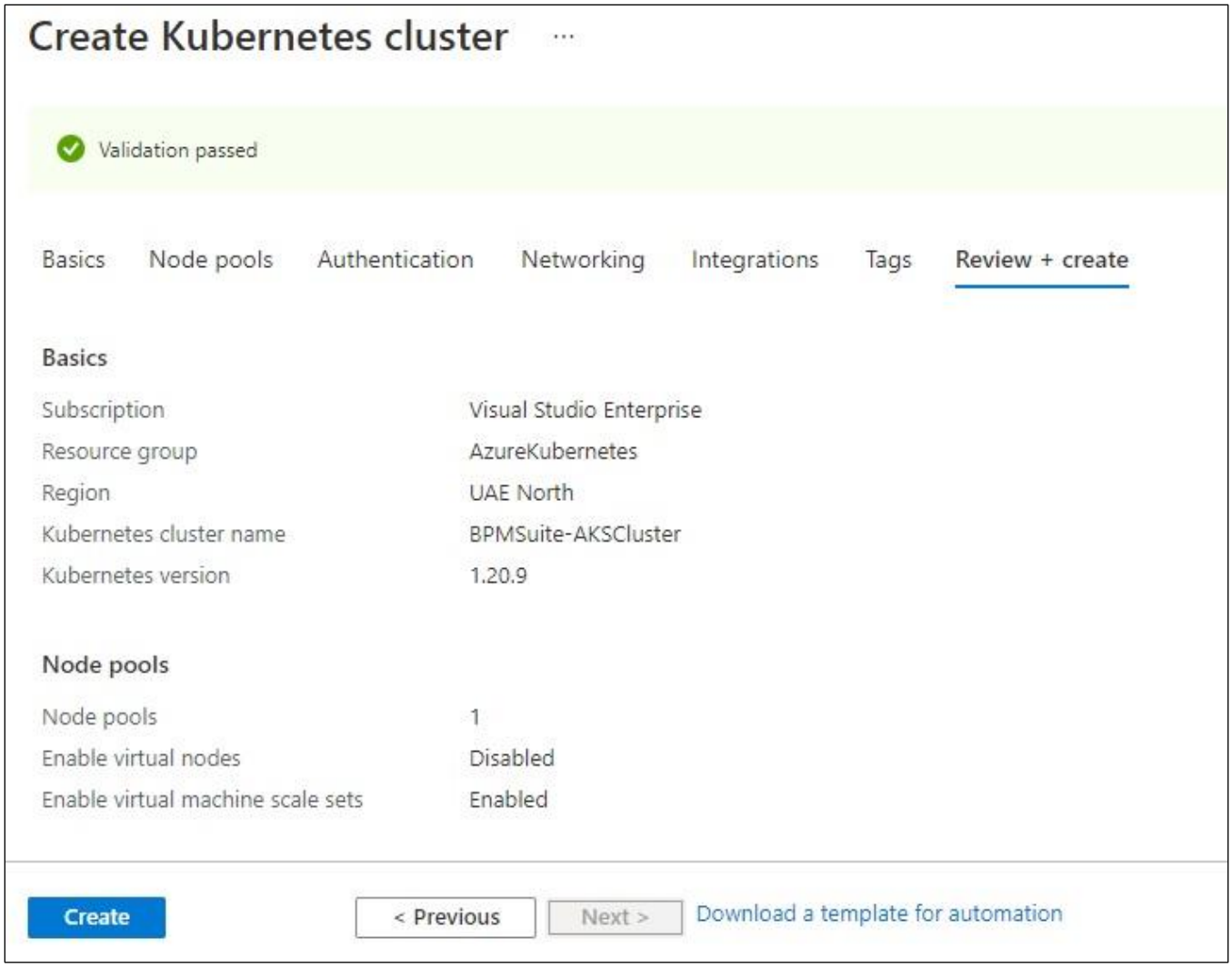

Figure 2.8

10. Once deployment is complete, click **Go to resource**.

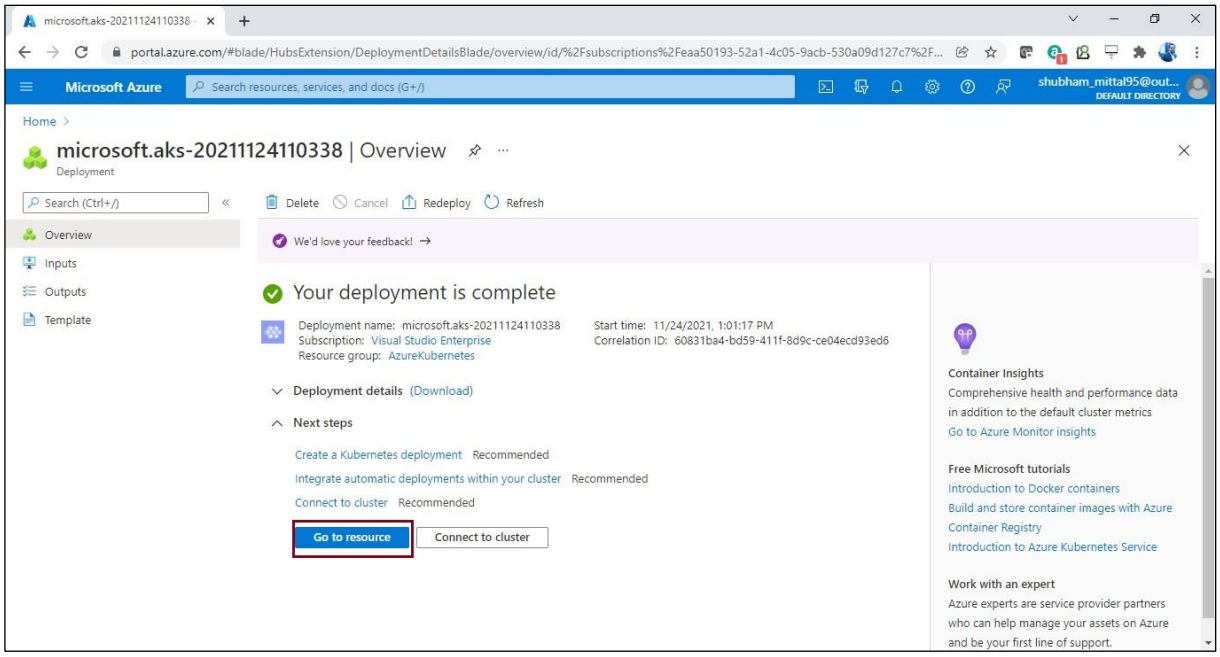

Figure 2.9

The Azure Kubernetes Cluster dashboard appears:

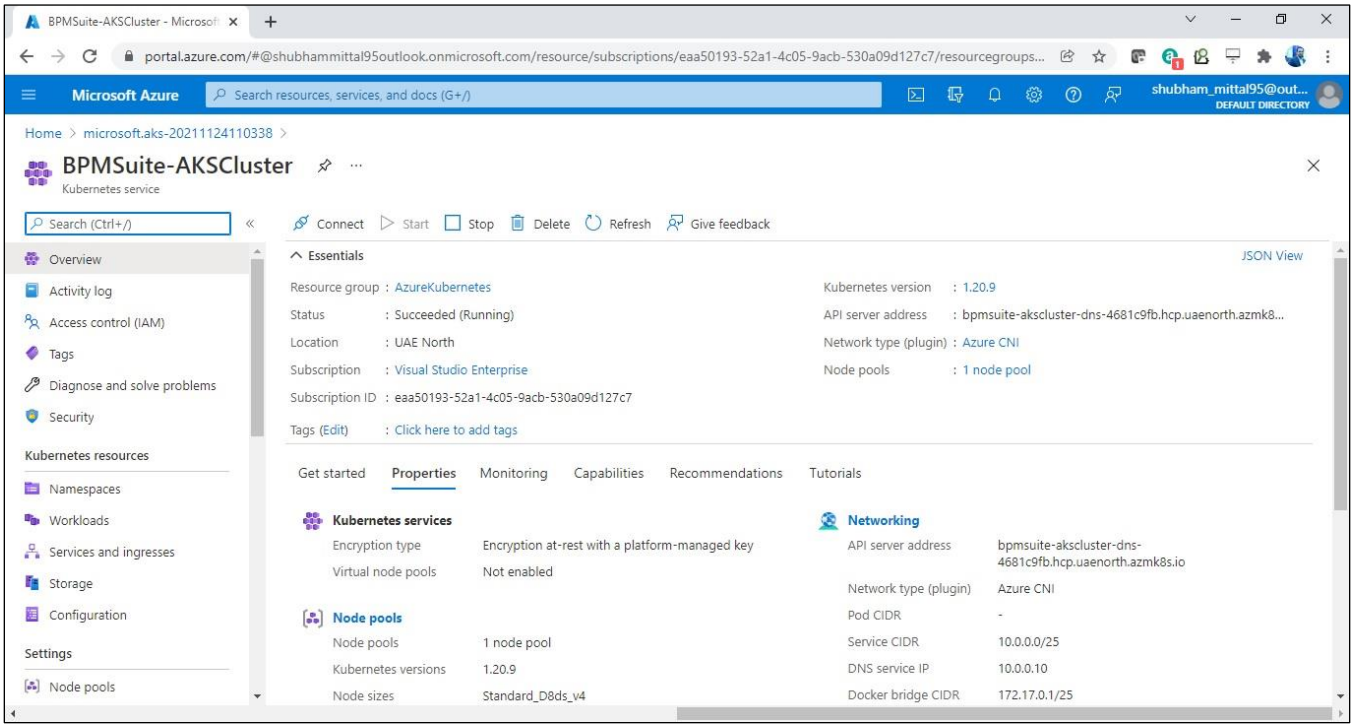

Figure 2.10

2.2 Configuration of Azure container registry

Perform the below steps to configure Azure Container Registry:

1. Sign in to the Azure Portal using the below URL:
   https://portal.azure.com/

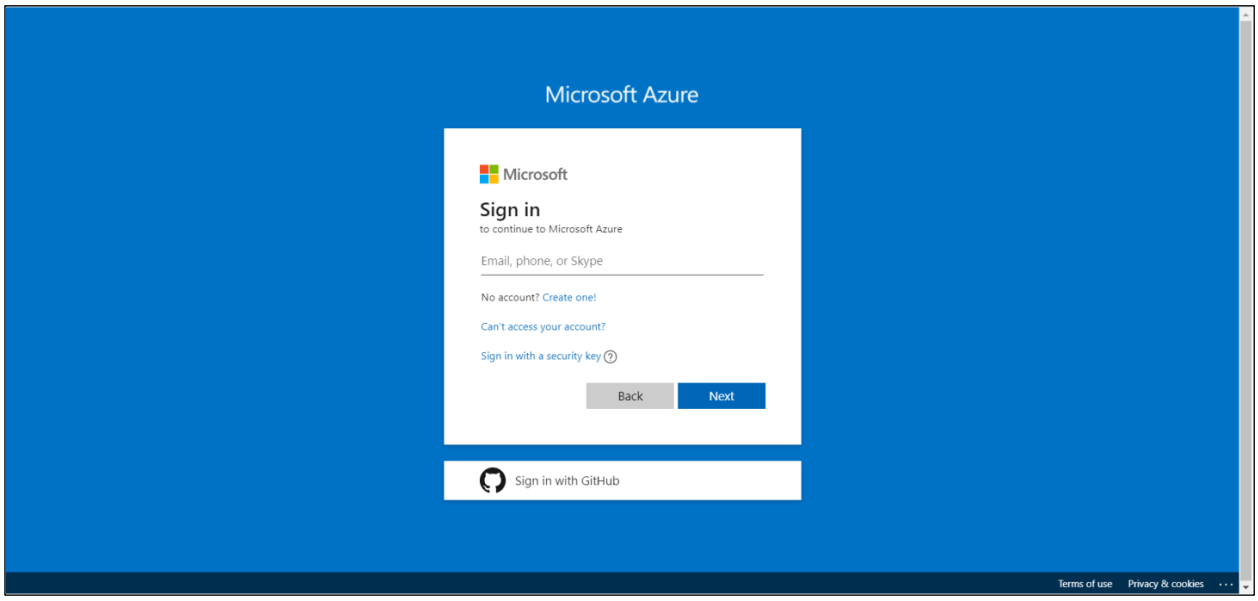

Figure 2.11

2. After a successful sign in, select **Create a resource.**
3. Select **Containers** and then select **Container Registry**.

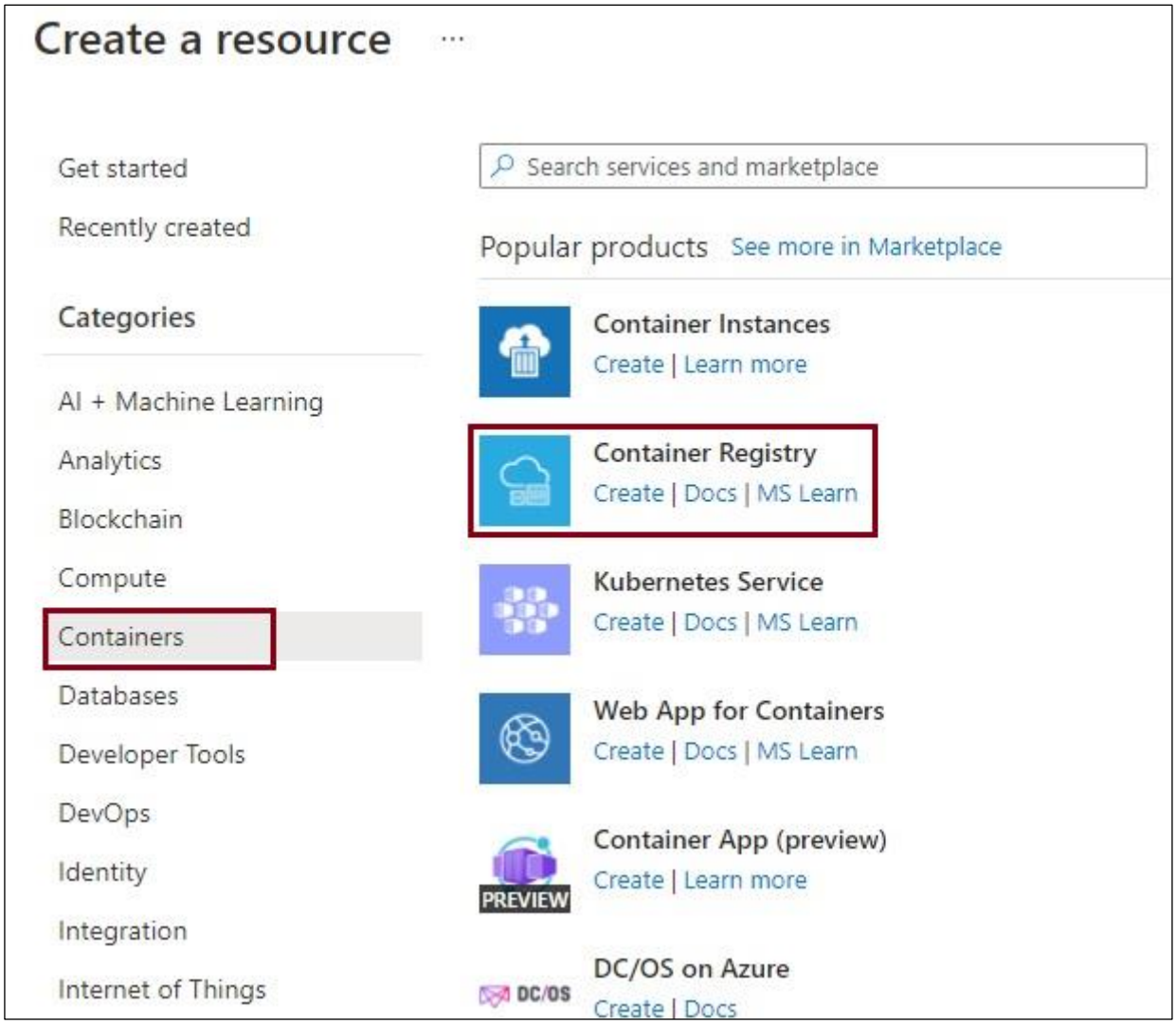

Figure 2.12

4. In the **Basic** tab, specify the following details:
   - **Resource group:** Select the existing resource group or create a new resource group that is, **AzureKubernetes**.
   - **Registry name:** Specify the user-defined name that is, **newgencontainerregistry**.
   - **Location:** Select the location that is, UAE North, and so on.

- **SKU:** Select the SKU based on your usage as Basic, Standard, or Premium. Each SKU carry a different storage size.
  For example,

SKU	Storage Limit
Basic	10 GiB
Standard	100 GiB
Premium	500 GiB or more

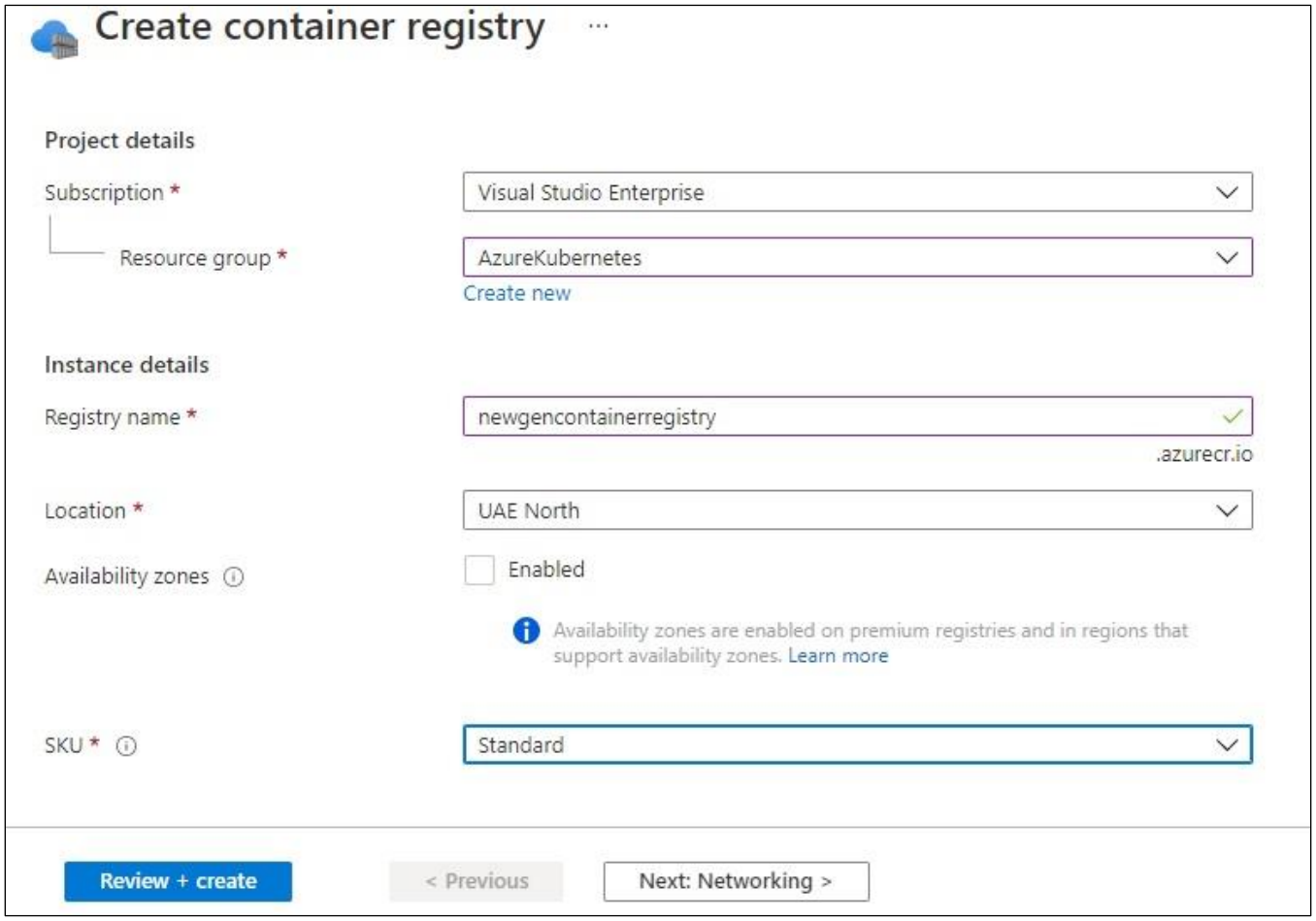

Figure 2.13

5. Accept default values for the remaining settings. Then select **Review + create**. After reviewing the settings, select **Create**.
6. When the **Deployment succeeded** message appears, select the container registry in the portal.

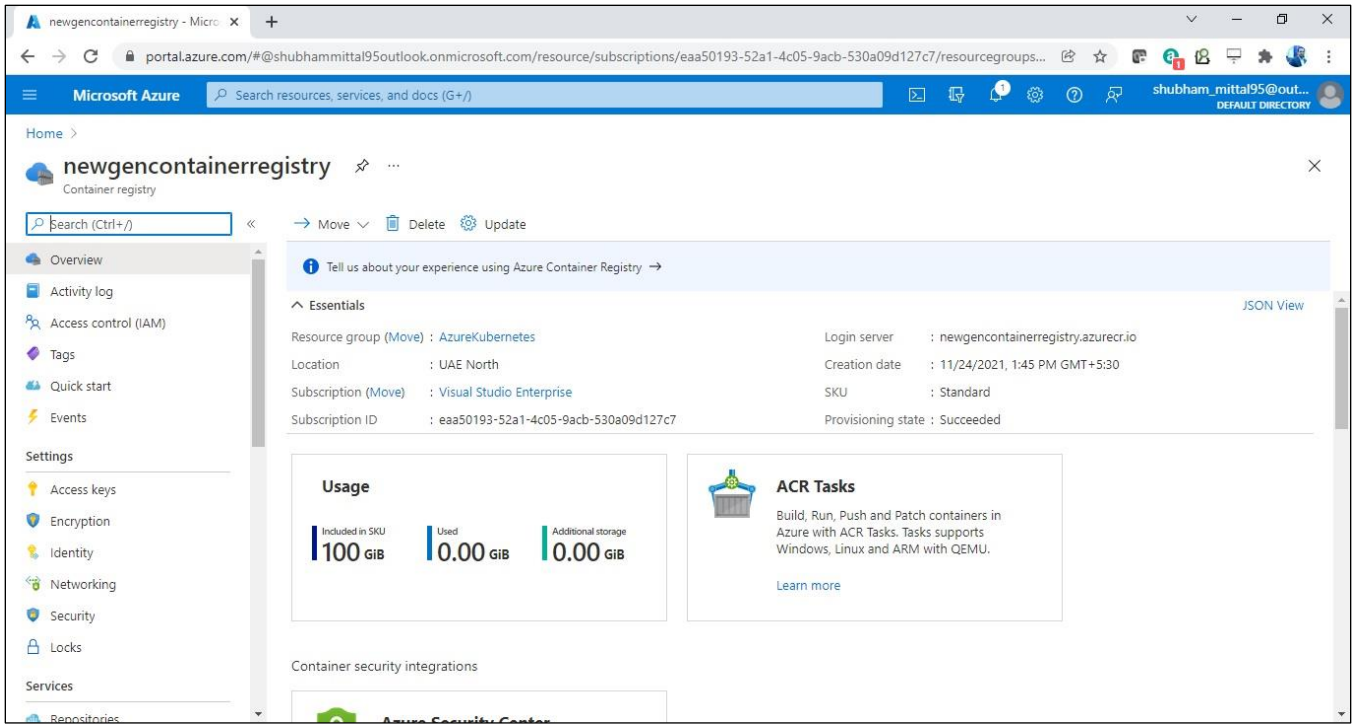

Figure 2.14

7. Click **Access keys** from **Settings** and enable **Admin user**.

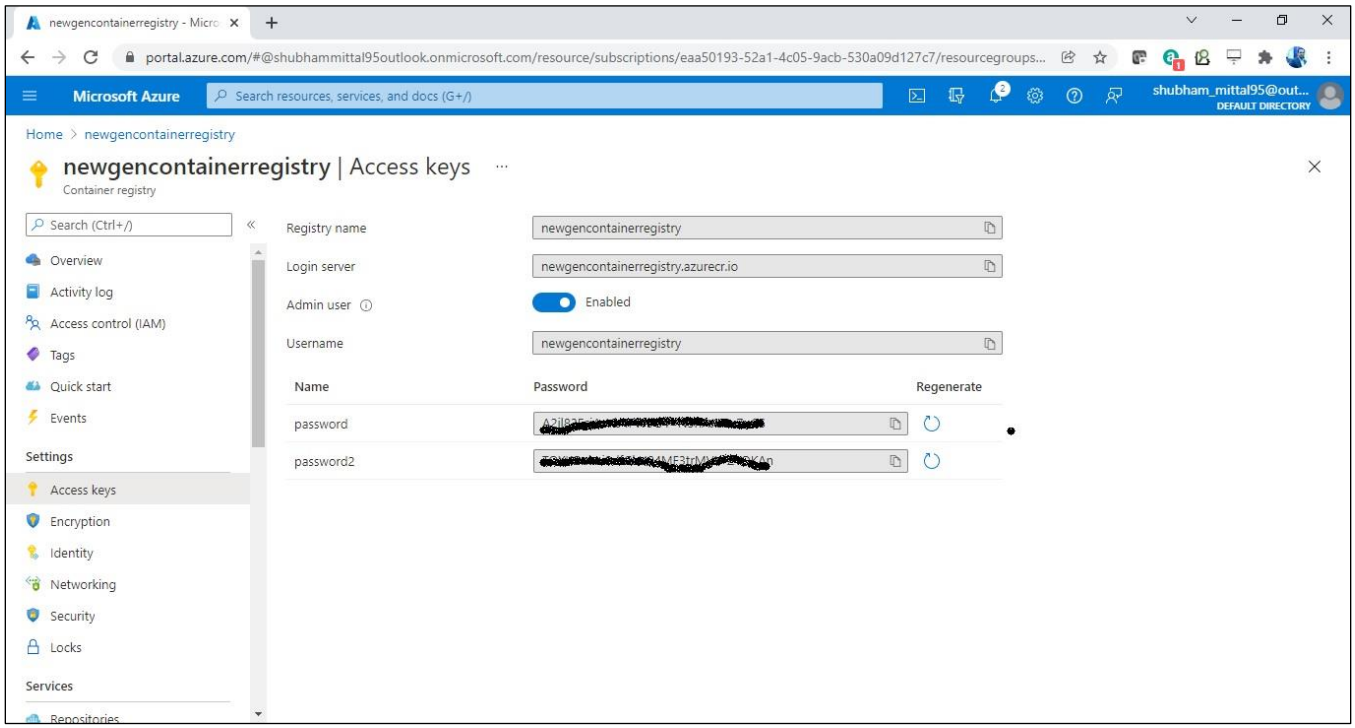

Figure 2.15

NOTE:
Kindly keep the **Login server**, **Username**, and **password** (or password2) as these values are required in the following steps to push or pull Docker images.

8. Use the below command to connect to the created container registry from your local machine (Where Docker Engine is already installed):

   docker login <Container Registry Login server> -u <Container Registry username> -p <Container Registry password>

   For example**,**

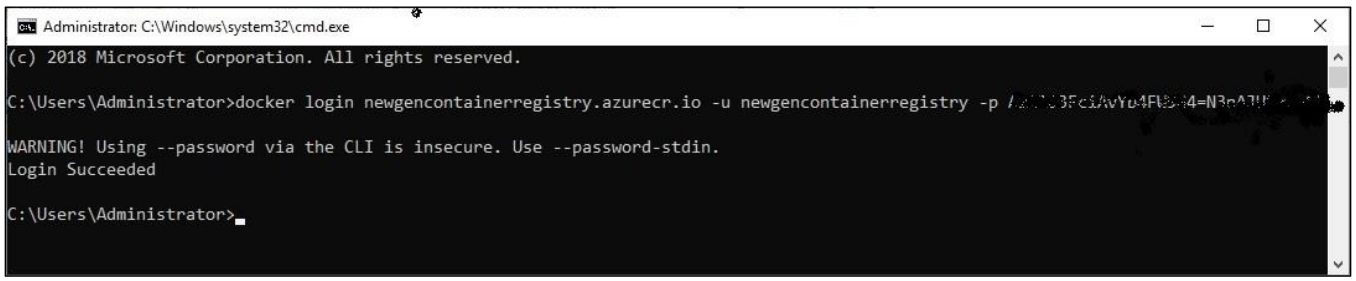

Figure 2.16

9. After a successful sign in to the Container Registry, use the below command to tag and push the Docker images from your local machine to ACR (Azure Container Registry):

```
docker tag <image name>:<image tag> <container registry server>/<image name>:<image tag>
docker push <container registry server>/<image name>:<image tag>
```

For Example,

```
docker tag ibps5serviceinstanceweb:sp2 newgencontainerregistry.azurecr.io/ibps5serviceinstanceweb:sp2
docker push newgencontainerregistry.azurecr.io/ibps5serviceinstanceweb:sp2
```

Where **newgencontainerregistry.azurecr.io** is the Container Registry Login server name.

NOTE:
Pushing any local Docker images to a repository is mandatory to tag that image 1st. You can also configure these commands in Jenkins to execute them automatically.

10. Use the below command to pull the Docker images from ACR to your local machine:

```
docker pull <container registry server>/<image name>:<image tag>
```

For Example,

```
docker pull newgencontainerregistry.azurecr.io/ibps5serviceinstanceweb:latest
```

2.3 Configuration of ACR image scanning

Perform the below steps to configure ACR Image Scanning:

1. **Microsoft Defender for Cloud** perform the ACR image scanning. Once the image scanning is configured and whenever a Docker image is pushed to the Azure Container Repository, Microsoft Defender for Cloud automatically scans that Docker image. Hence, it is mandatory to push that image in ACR to trigger the scan of an image.

NOTE:
Ensure that the Defender plan is enabled for the **Container registries**.

---

2. Go to the **Microsoft Defender for Cloud** page.
3. Click **Environment settings** under **Management.**
4. Click listed subscription.

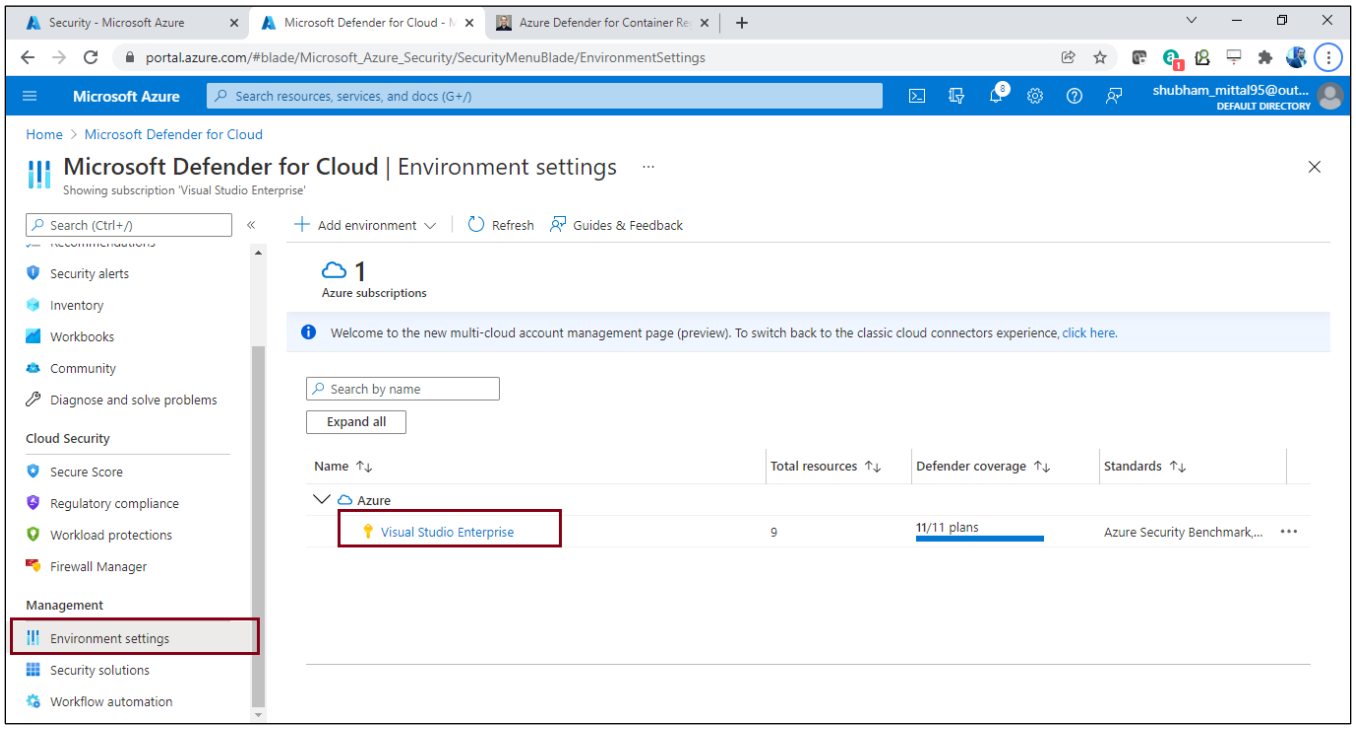

Figure 2.17

5. Enable the **Container registries** defender plan if it is not already enabled.

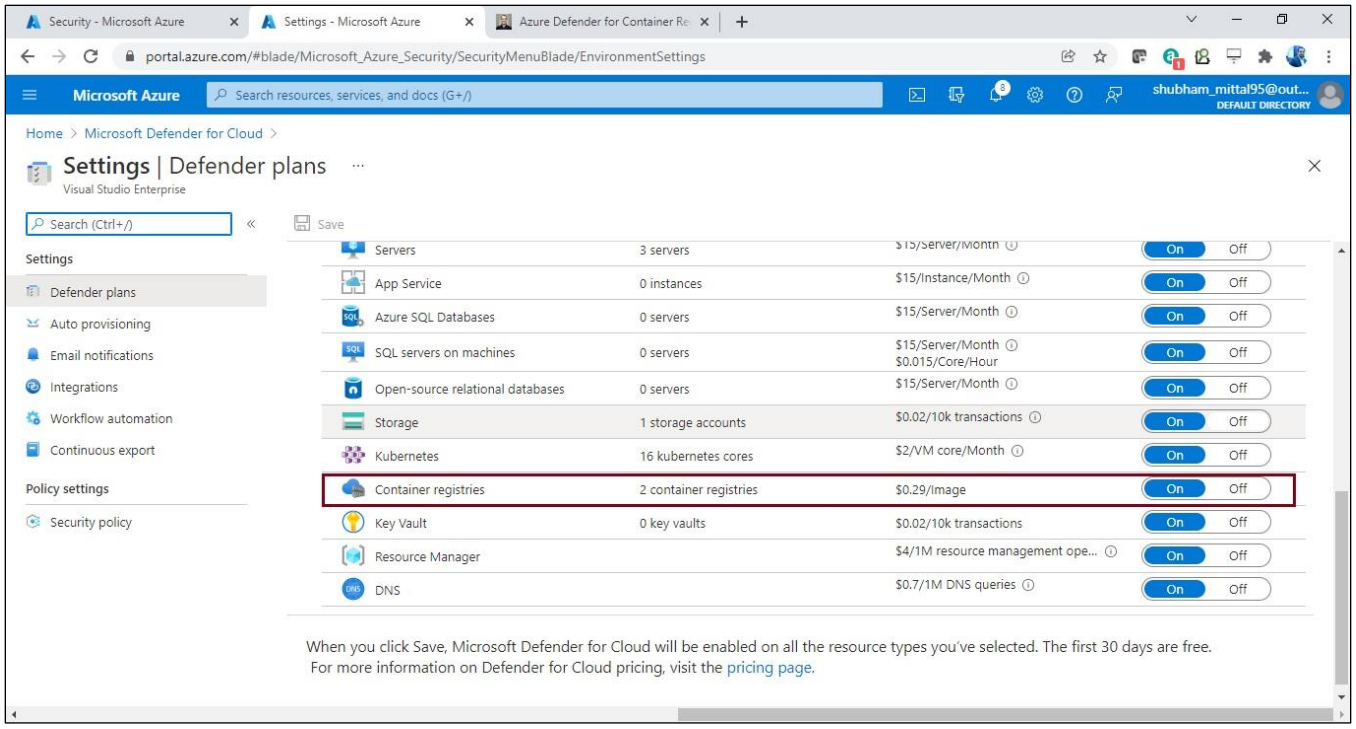

Figure 2.18

Microsoft Defender for container registries includes a vulnerability scanner to scan the images in your Azure Resource Manager-based Azure Container Registry registries.

6. Provide deeper visibility into your images vulnerabilities. The integrated scanner is powered by Qualys, the industry-leading vulnerability scanning vendor.
   When issues are found – by Qualys or Defender for Cloud – you get notified in the workload protection dashboard.
   For example,

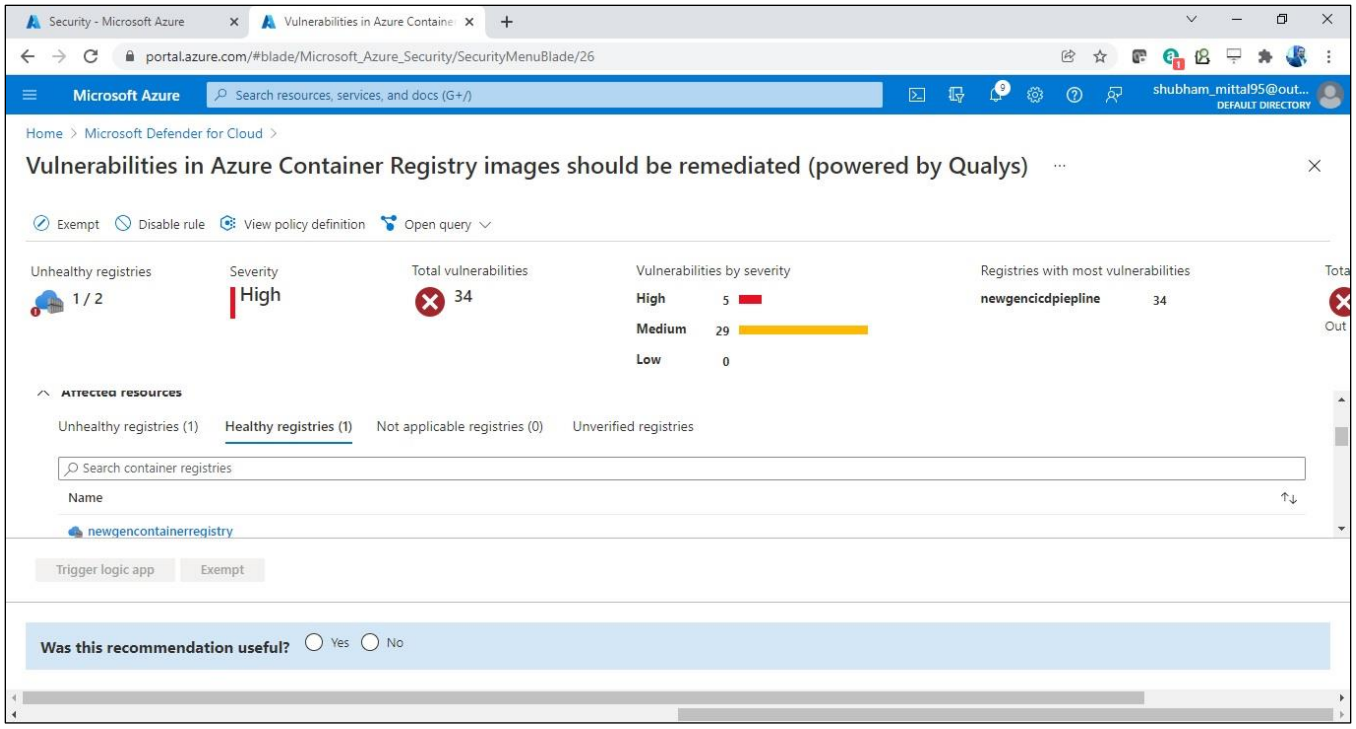

Figure 2.19

2.4 Create storage account

Perform the below to configure a storage account:

2.4.1 Create a BLOB storage

Perform the below steps to create IAM Policy and Role:

1. Sign in to the Azure Portal using the below URL: https://portal.azure.com/
2. Select **All services** on the Azure portal menu.
3. Select **Storage Accounts**.
4. Click **Create** on the Storage Accounts.
5. On the **Basics** tab, select an active Azure subscription.
6. Under the Resource group field, select your desired resource group, or create a new resource group like **AzureKubernetes.**
7. Enter a name for your storage account like **bpmsuitestoage.**
8. Select a **location or region** in which you want to create your storage account that is, UAE North.
9. Select a performance tier. The default tier is Standard.
10. Specify how you want the storage account to replicate. The default replication option is Geo-redundant storage (GRS).

11. Keep the other settings as default and click **Next: Advanced>**.

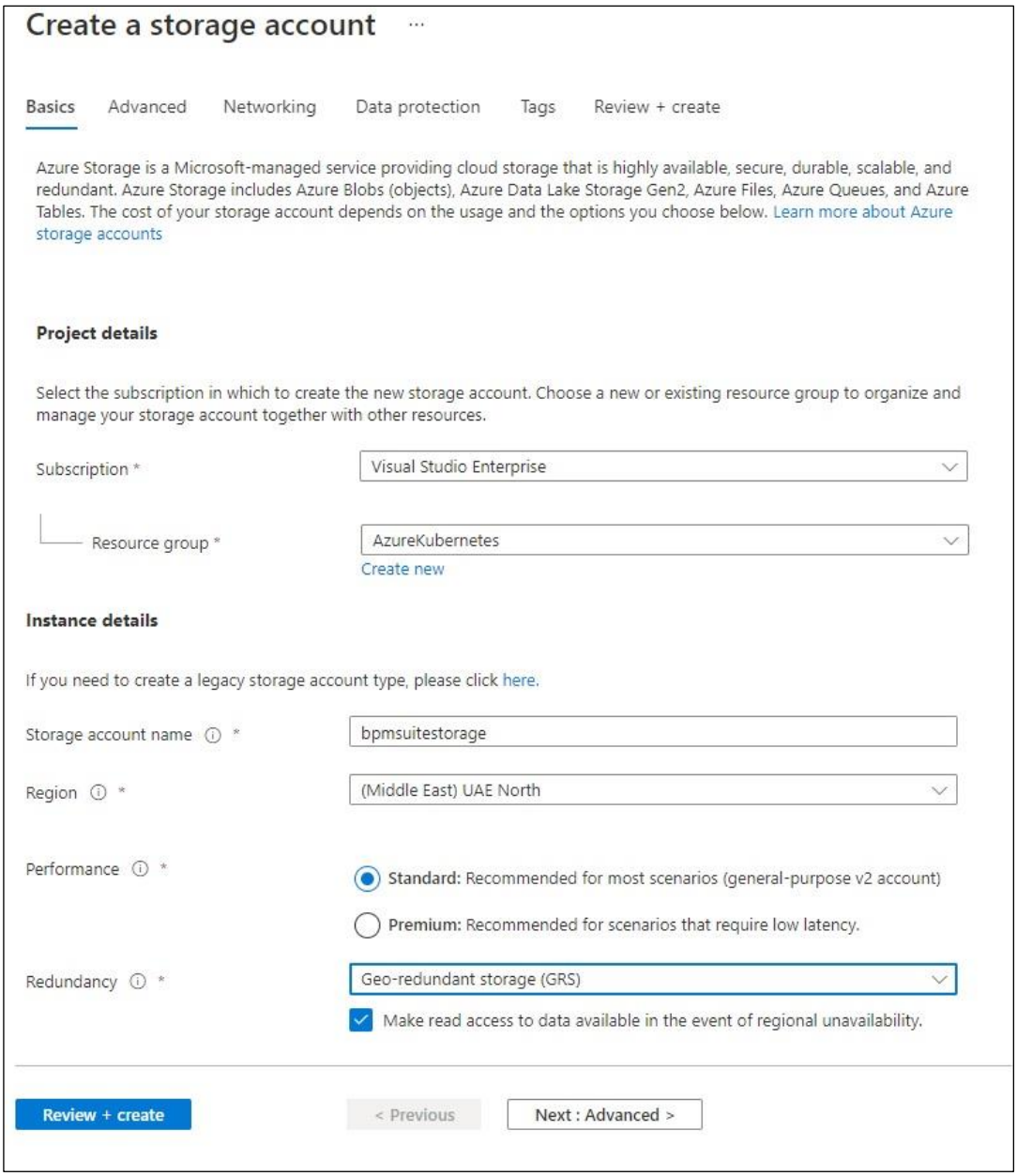

Figure 2.20

12. On the **Advanced** tab, keep the default options and click **Next: Networking>**.

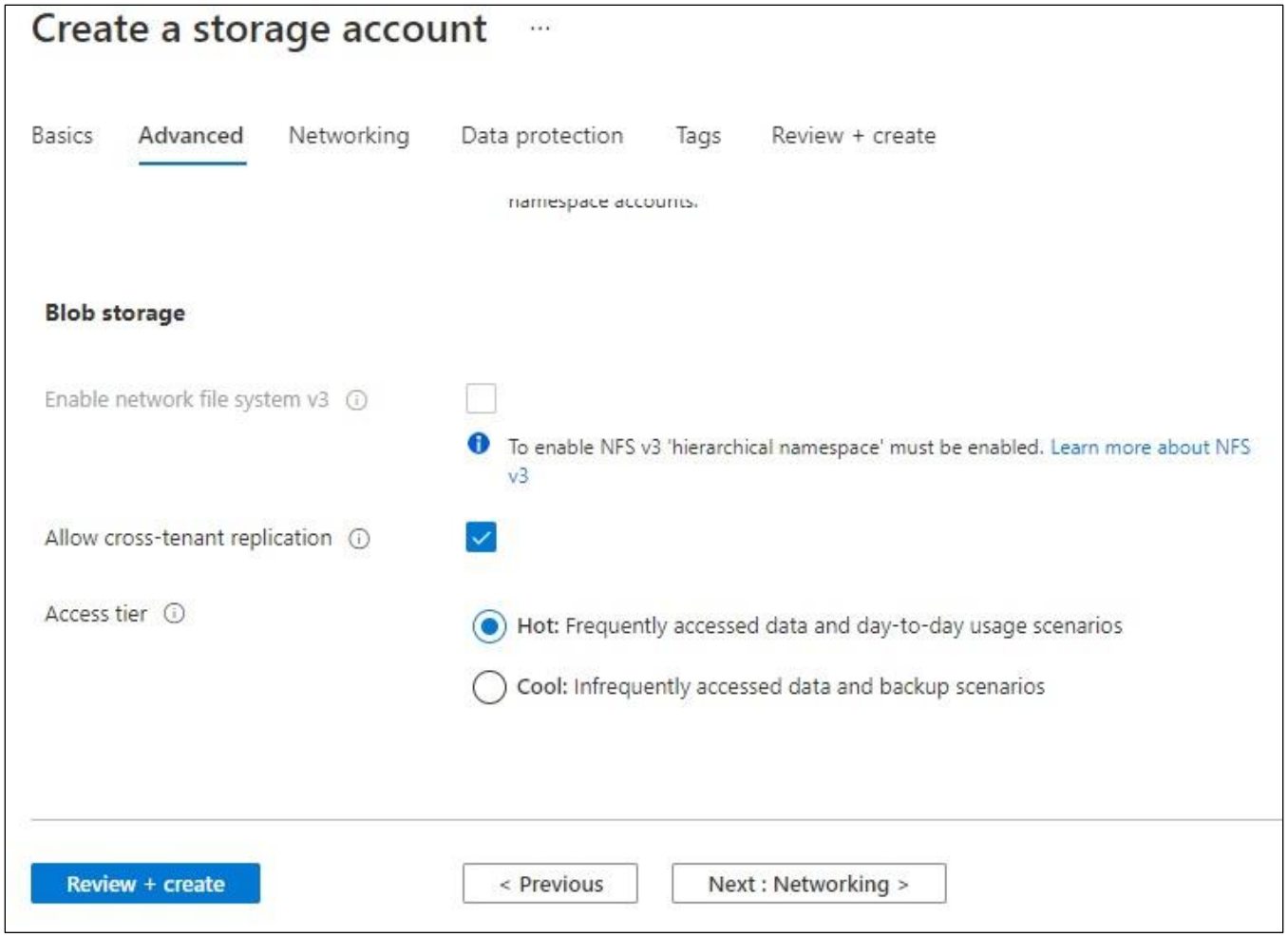

Figure 2.21

13. On the **Networking** tab, keep the default options and click **Next: Data protection>**.

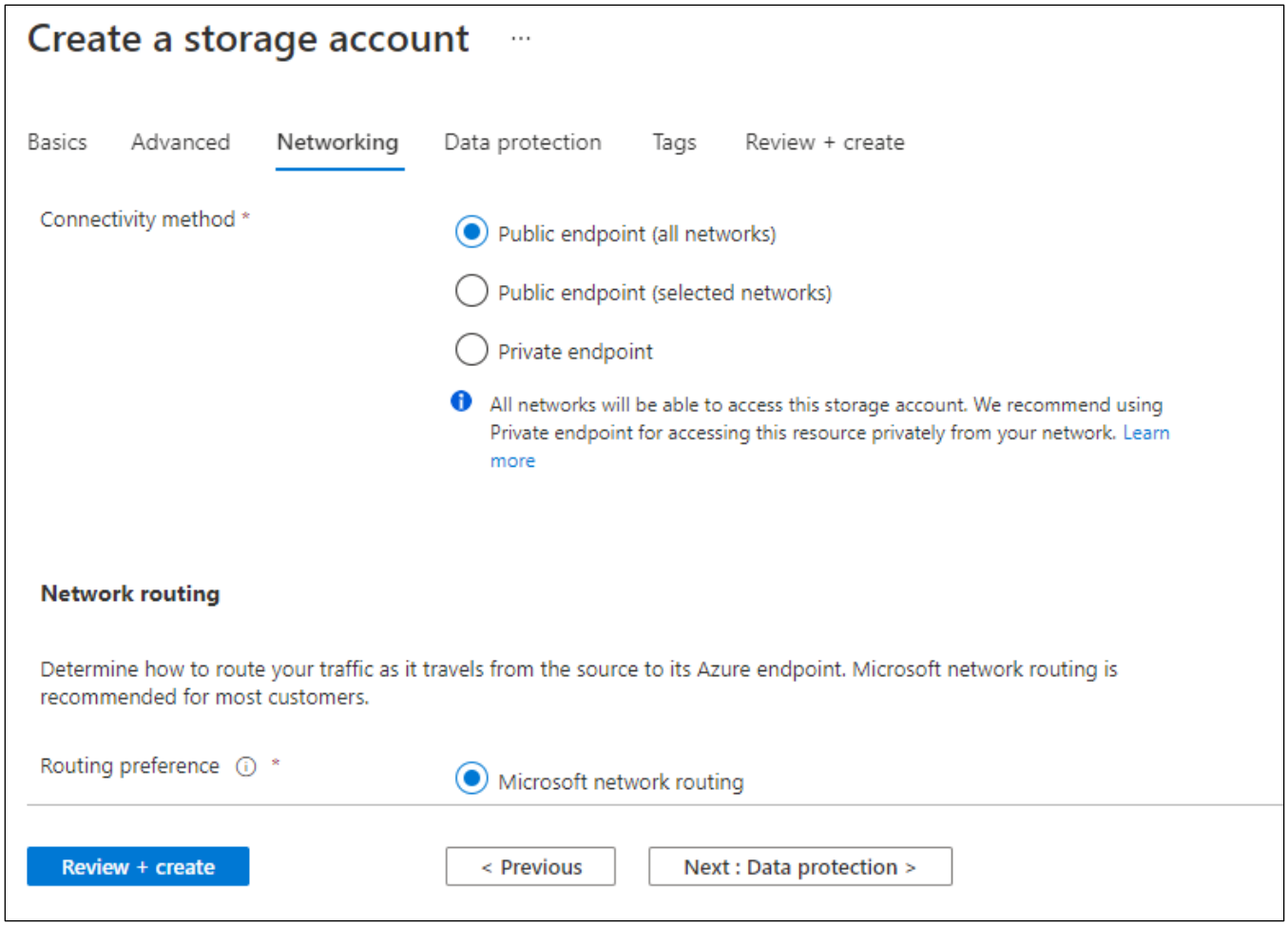

Figure 2.22

14. On the **Data protection** tab, keep the default options and click **Next: Tags>**.

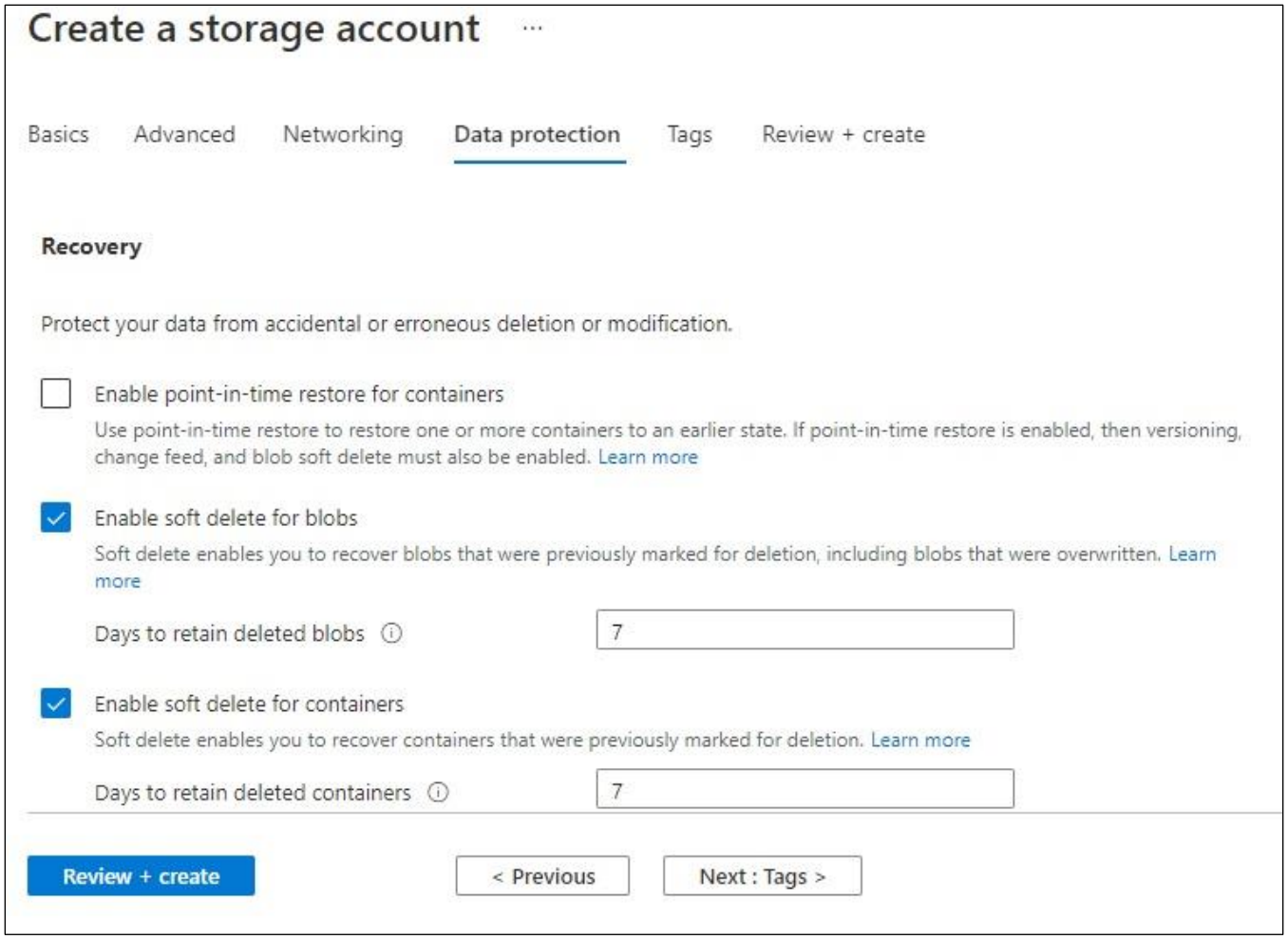

Figure 2.23

15. On the **Tags** tab, keep the default options and click **Next: Review + create**.

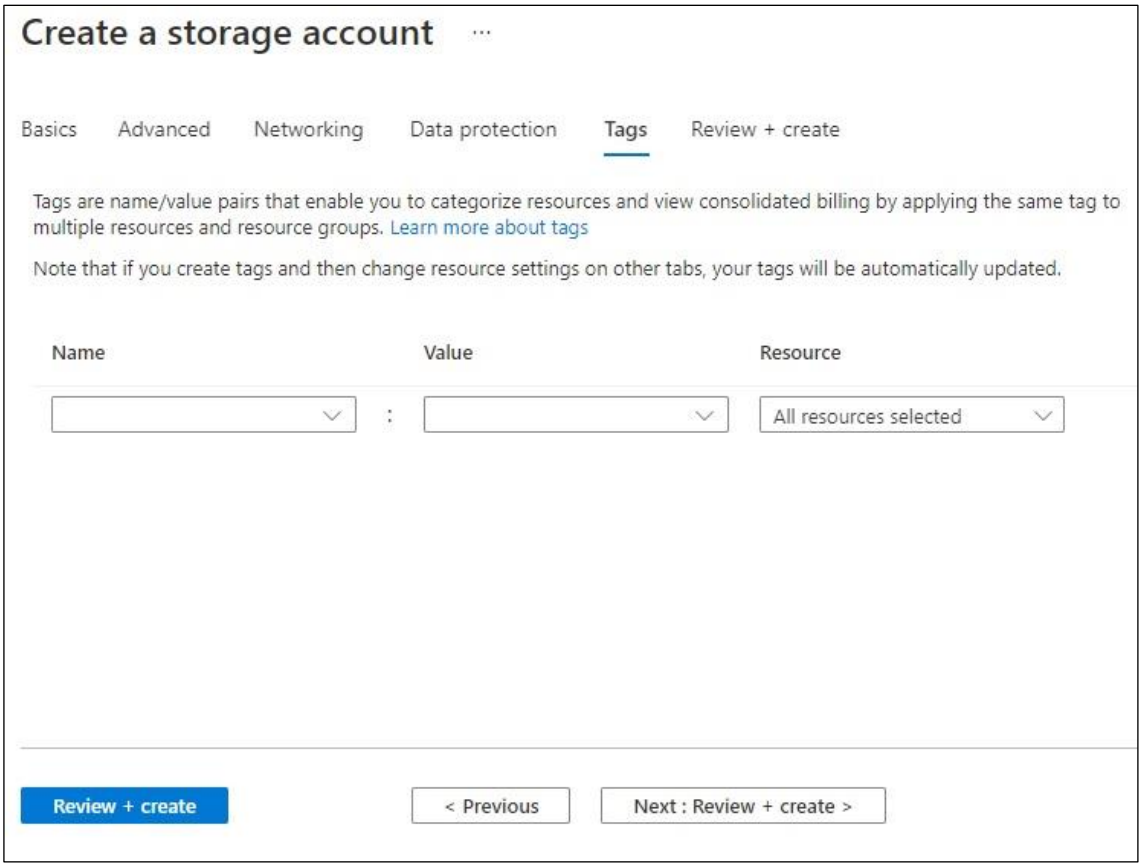

Figure 2.24

16. On the **Review + create** tab, click **Create** once validation is passed.

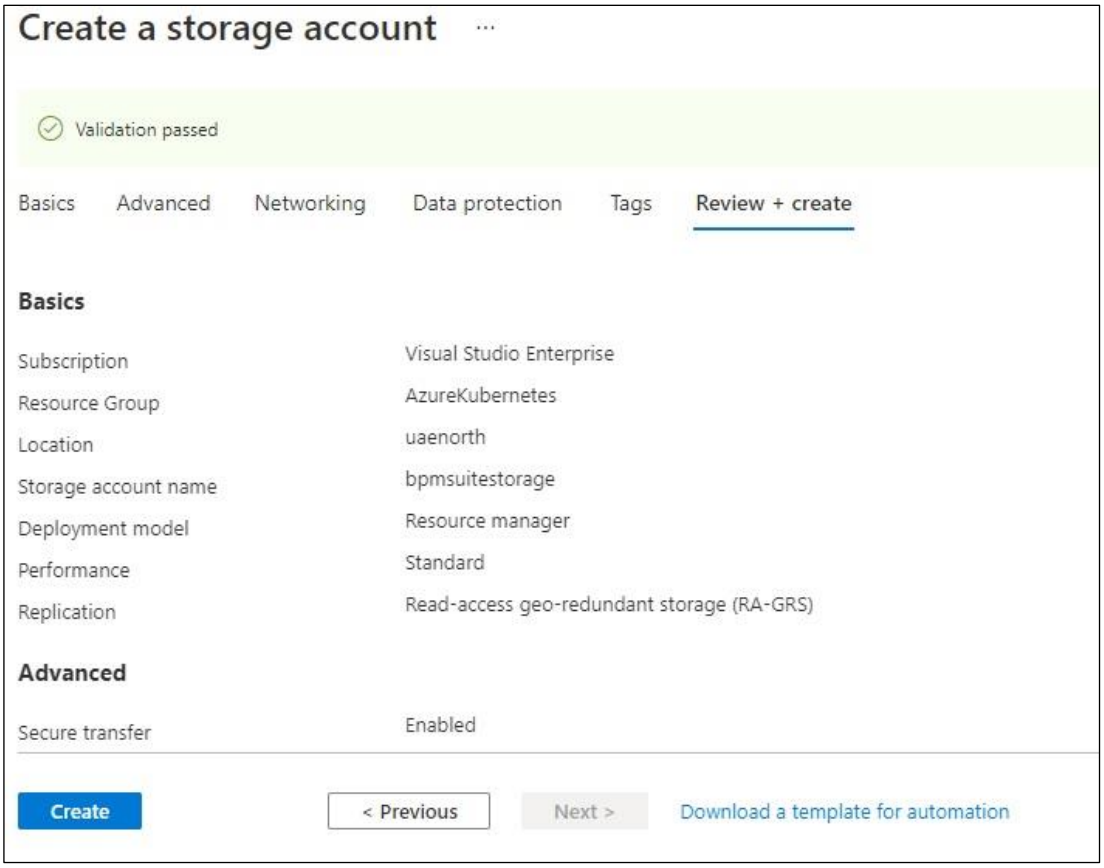

Figure 2.25

17. Once deployment is complete, click **Go to resource**.

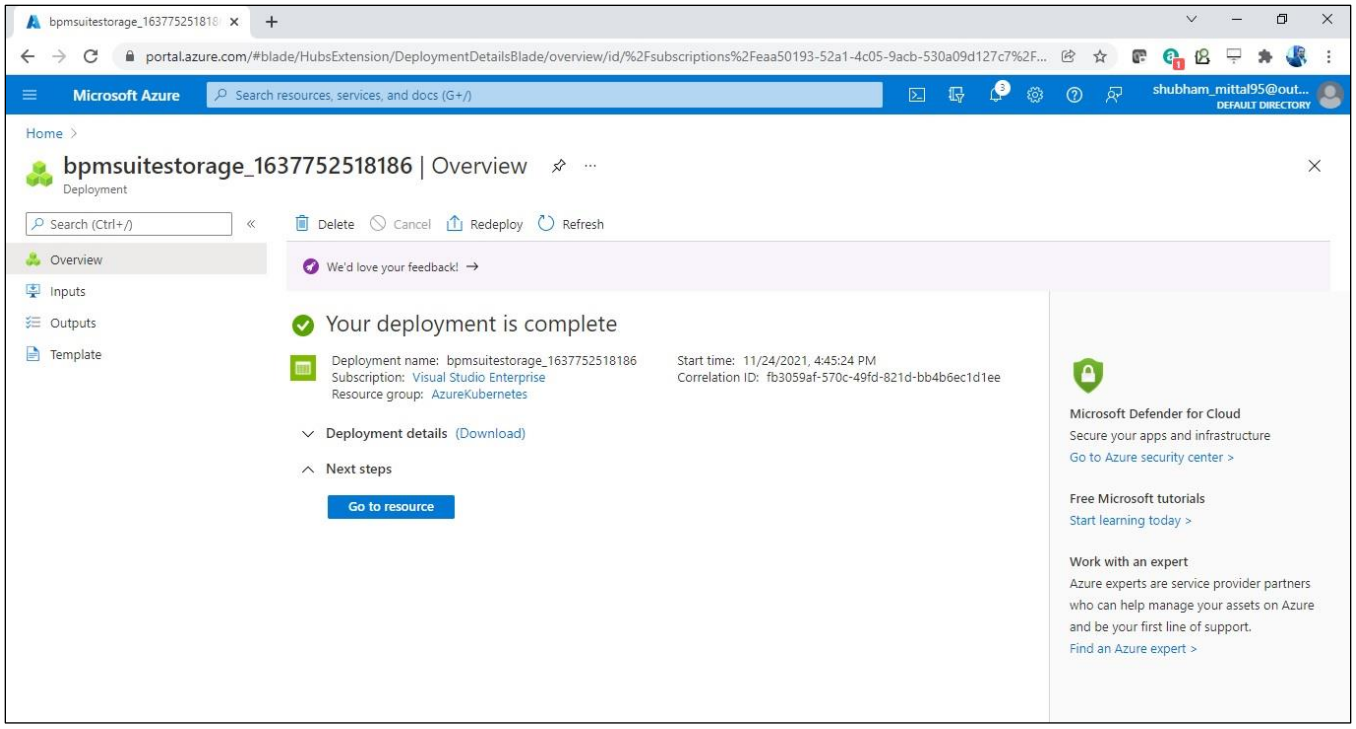

Figure 2.26

18. Click **Access keys** from Settings under Security + Networking.

NOTE:
Keep the **Storage account name** and **key1** (or key2) as these values are required in the following steps for the Kubernetes volume mounting.

---

19. Click **Containers** under the Data storage. The **Containers** screen appears.
20. Click **+Container.** The **New Container** dialog appears.
21. Specify the following details:
    - **Name**: Specify the unique blob storage name.
    - **Public access level**: Select default “Private (no anonymous access)”.
22. Click **Create.**

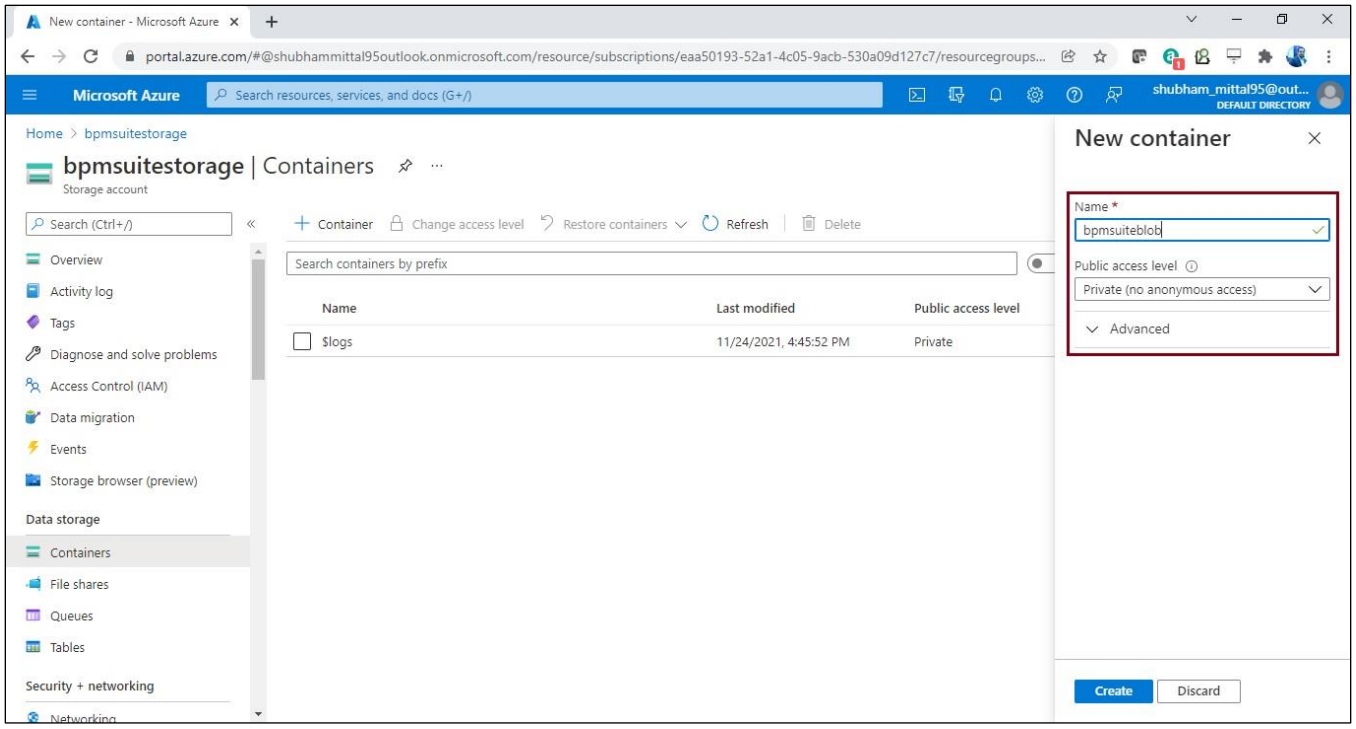

Figure 2.27

2.4.2 Create an Azure file share

Perform the below steps to create an Azure File Share:

1. Click **Overview** of the created storage account.
2. Click **File shares** under the Data storage. The **File shares** dialog appears.

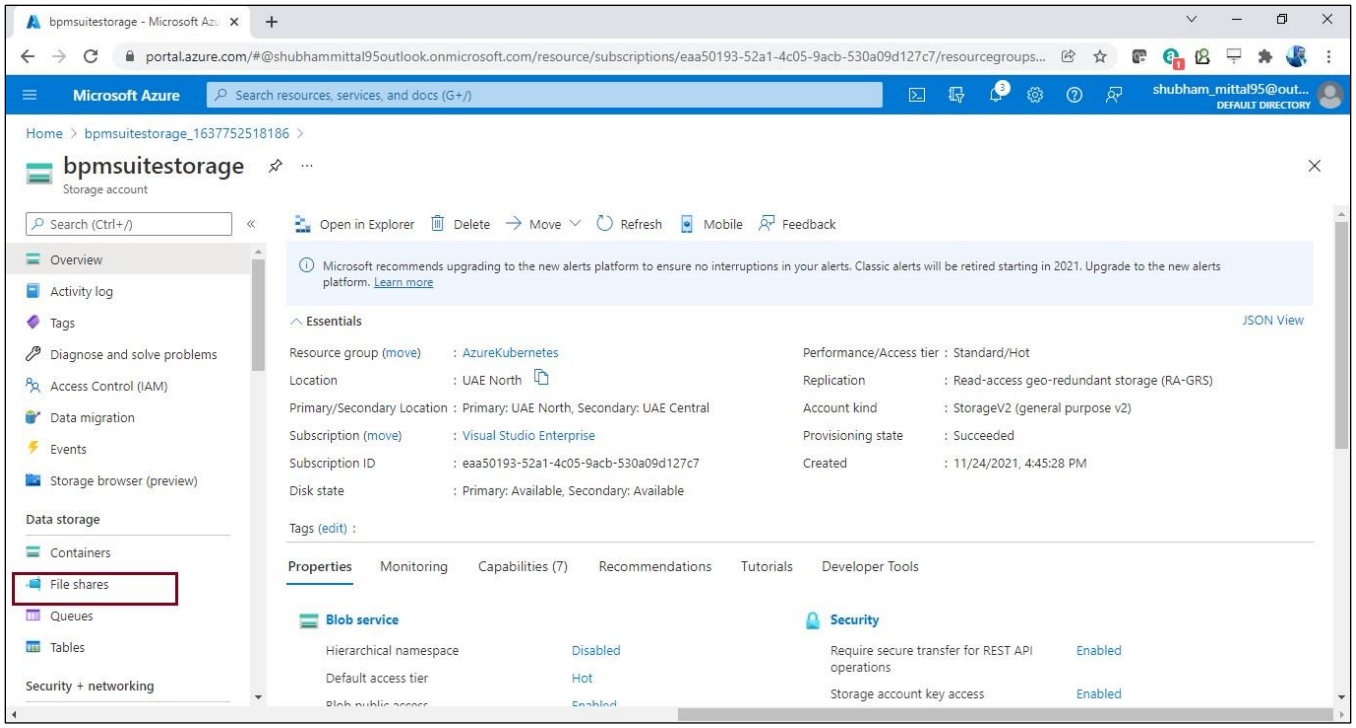

Figure 2.28

3. Click **+File share**. The **New File share** dialog appears.
4. Specify the following details:
   - **Name**: Specify the unique file share name.
   - **Tiers:** Select the 'Transaction optimized' as tier.
   - Click **Create.**

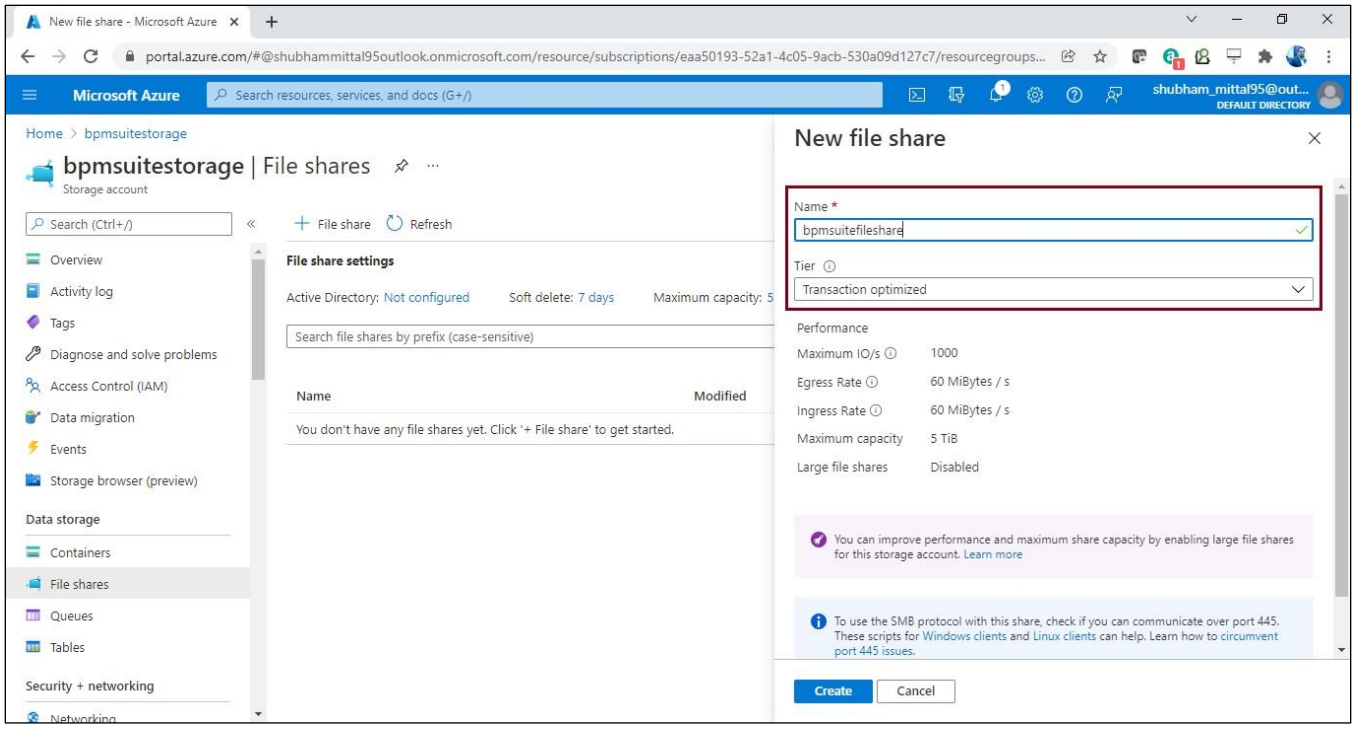

Figure 2.29

2.5 Configuration of Azure cache for Redis

Azure Cache for Redis provides fully managed open-source Redis within Azure that can be used as a distributed data or content cache. In addition, it can be used as a session store and so on along with that it provides an in-memory data store.

Perform the below steps to configure the Azure Cache for Redis:

1. Sign in to the Azure Portal using the below URL:
   https://portal.azure.com/
2. On the Azure portal menu or from the home page, select **Create a resource**.
3. Select **Databases**.
4. Select **Azure Cache for Redis**.

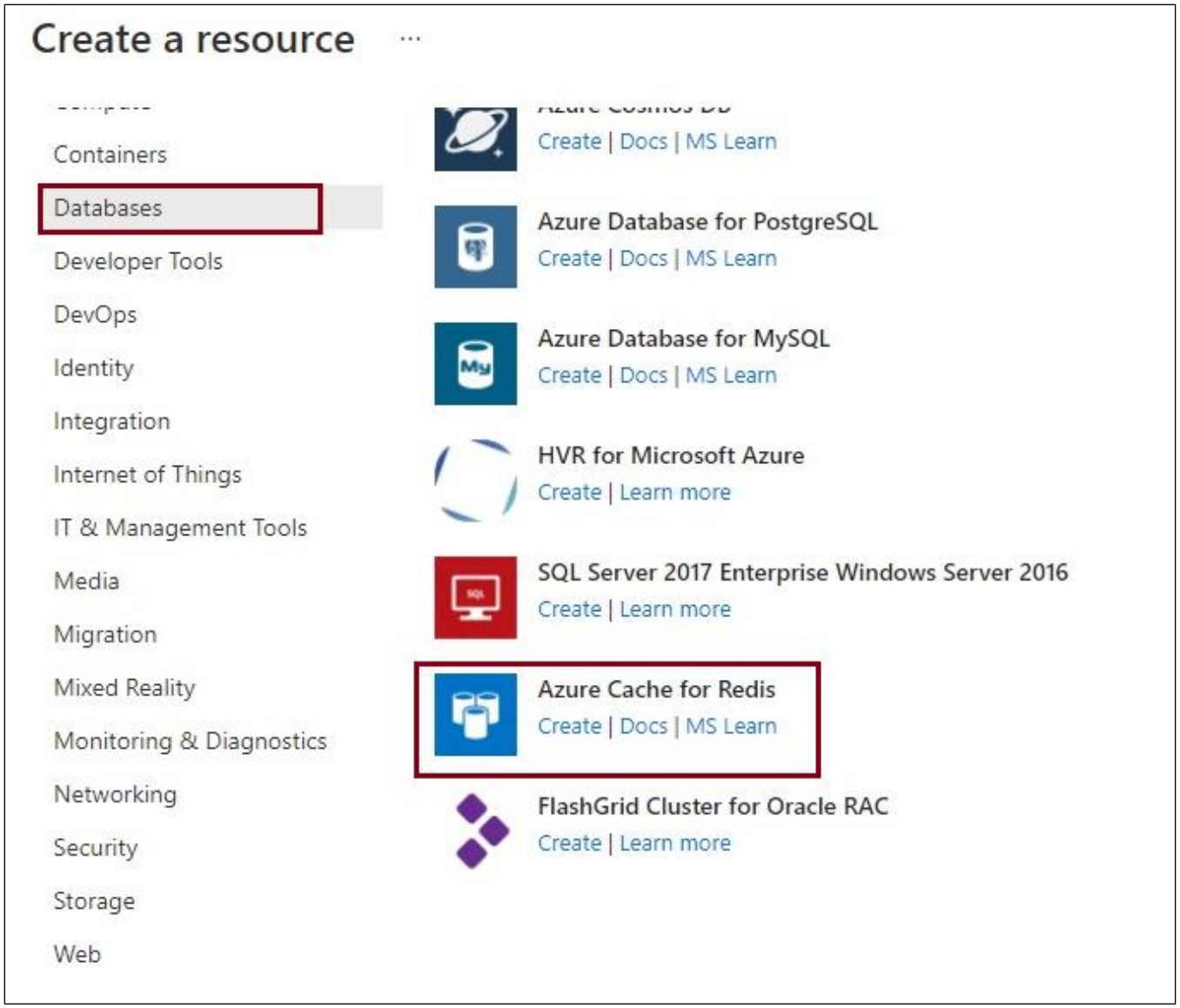

Figure 2.30

5. Specify the following details under the **Basics** tab:
   - **Subscription**: Select a valid Azure subscription.
   - **Resource group**: select or create an Azure Resource group, such as **AzureKubernetes**.
   - **DNS name**: Enter a Redis cache DNS name such as **azrediscache**.
   - **Location**: Select a region into which you want to create an Azure Cache for Redis.
   - **Cache type:** Select the Redis cache service tier as per your requirement. You can select from 250 MB to 1455 GB in-memory cache.
   - Click **Next: Networking**>.

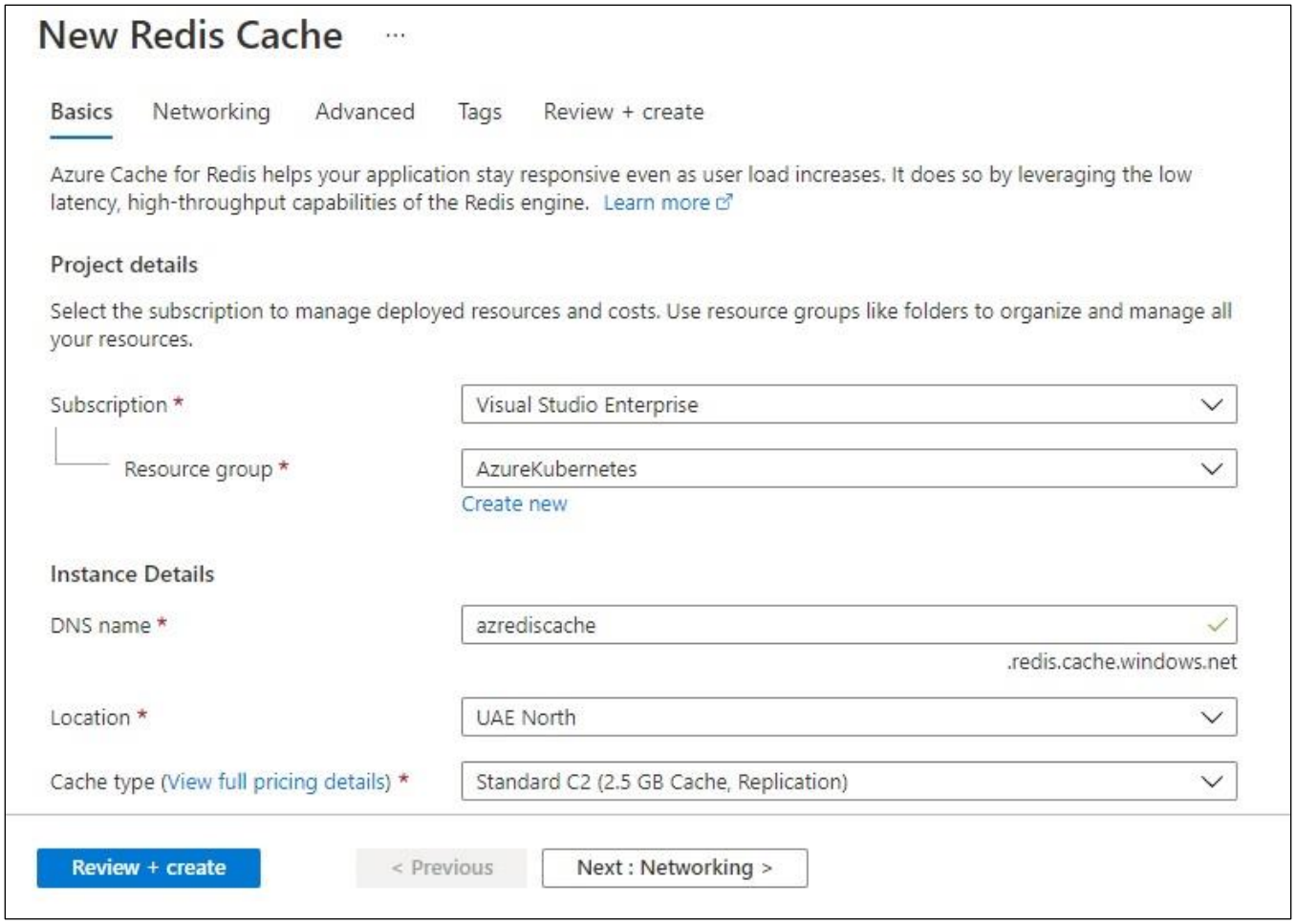

Figure 2.31

6. On the **Networking** tab, select the connectivity method as 'Public Endpoint' and click **Next: Advanced>**.

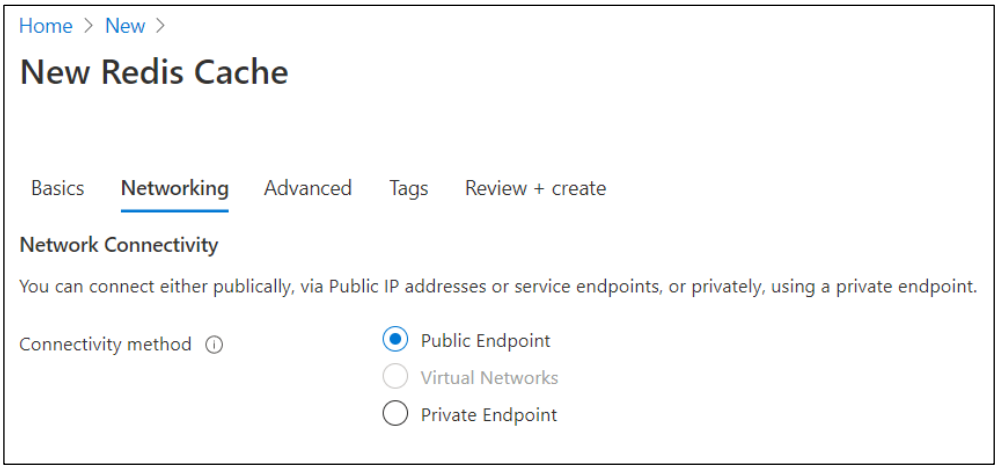

Figure 2.32

7. On the **Advanced** tab, enable the Non-TLS port, select the Redis version as 6 and click **Next: Tags>**.

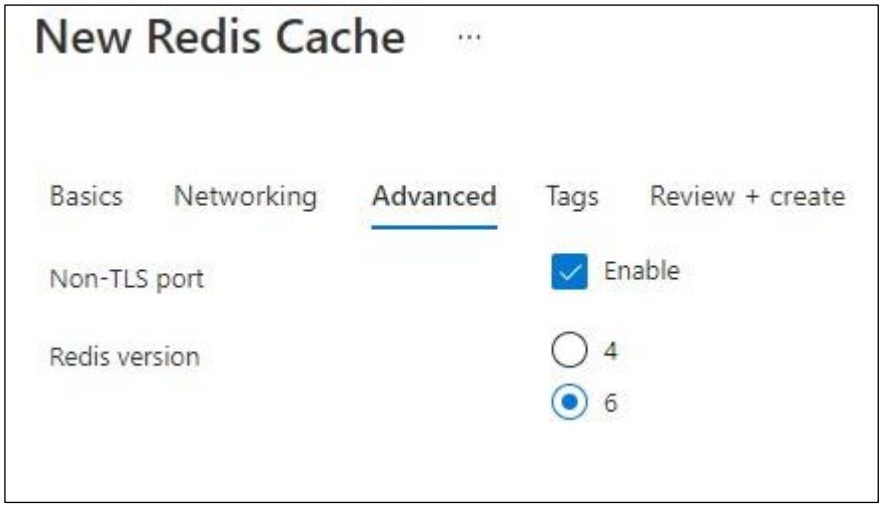

Figure 2.33

8. On the **Tags** tab, keep the default options and click **Next: Review + create>**.
9. On the **Review + create** tab, click **Create** once validation is passed.\

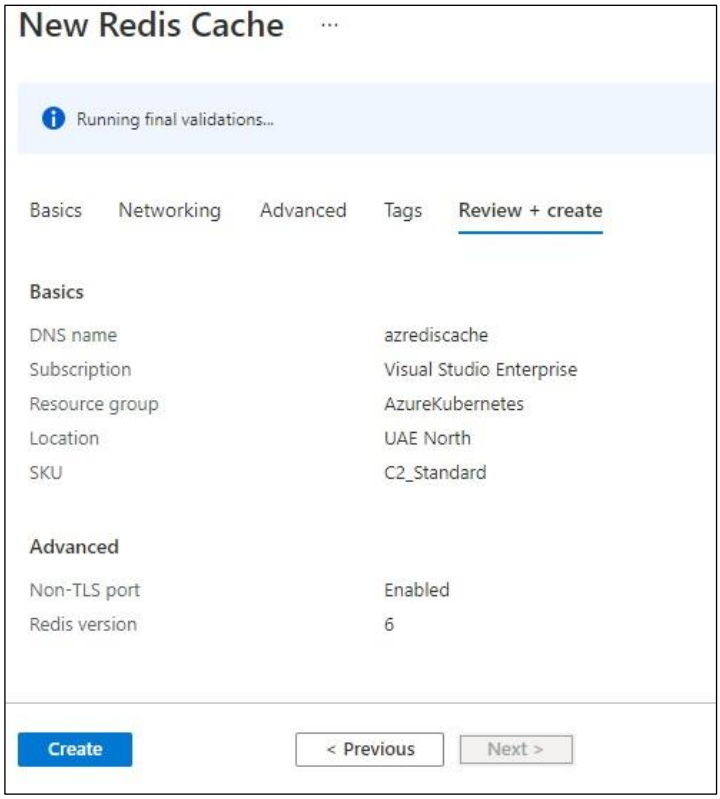

Figure 2.34

10. Once deployment is complete, click **Go to resource.**

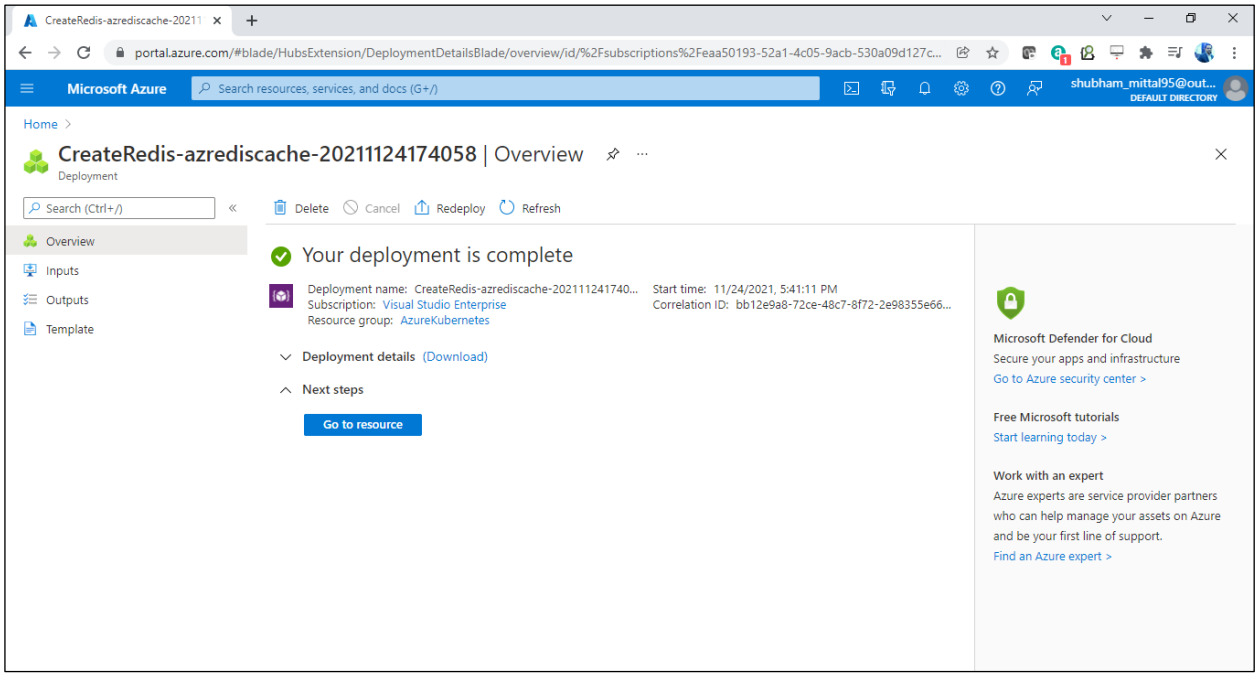

Figure 2.35

2.6 Configuration of application gateway ingress controller

This section explains how to configure Application Gateway Ingress Controller.

2.6.1 Creation of an application gateway

Pre-requisites:

- A subnet must be created in the same virtual network in which the Kubernetes cluster exists.

Perform the below steps to create an Application Gateway:

1. On the Azure portal menu or from the Home page, select **Create a resource**.
2. Select **Networking**.

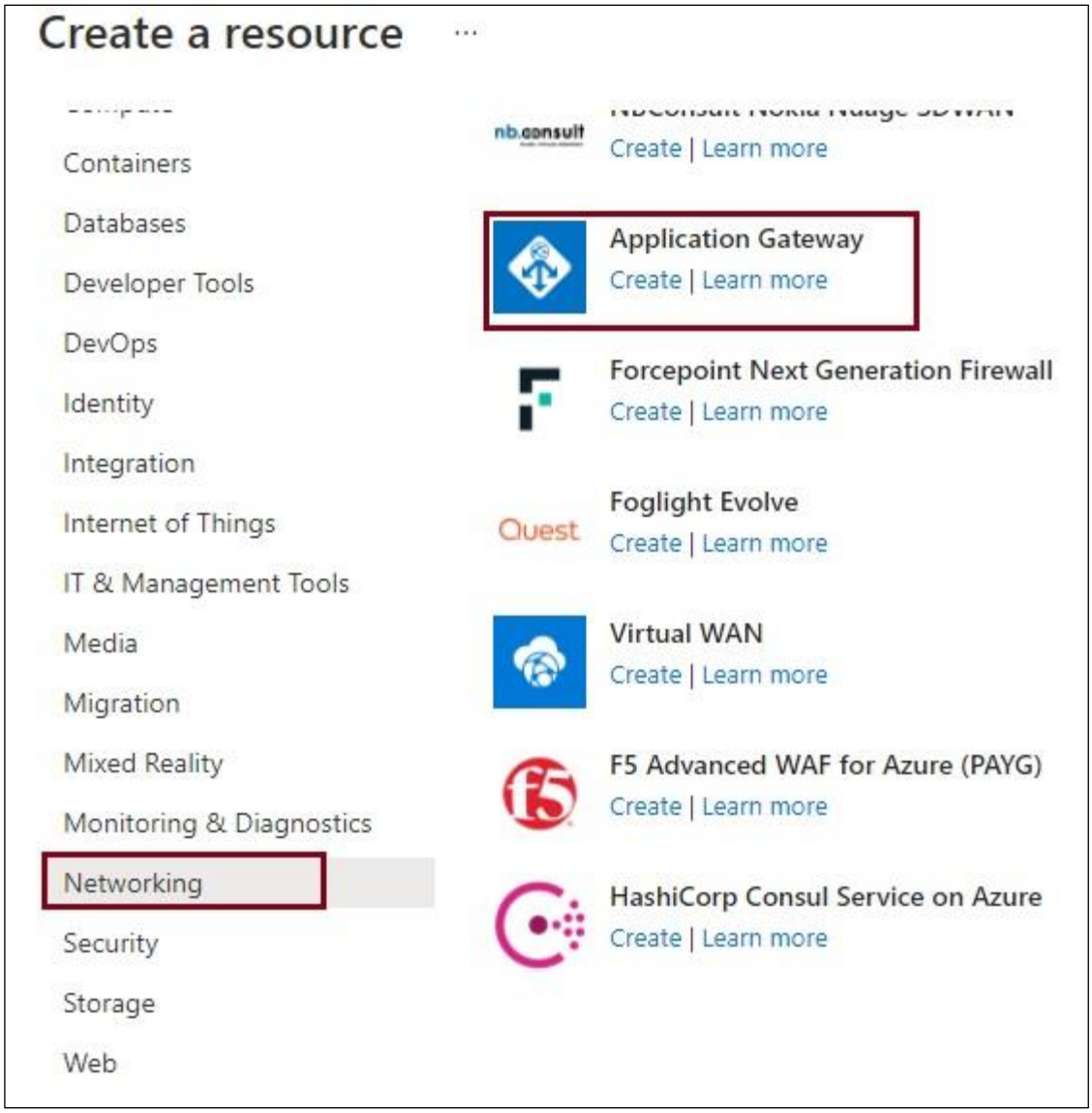

Figure 2.36

3. Select **Application Gateway**. The **Create application gateway** screen appears.
4. Specify the following details under the **Basics** tab:
   - **Subscription**: Select a valid Azure subscription.
   - **Resource group**: Select or create an Azure Resource group, such as **AzureKubernetes**.
   - **Application gateway name**: Enter a Kubernetes cluster name such as **AppGateway-AKSCluster**.
   - **Region**: Select a region into which you want to create an AKS cluster that is, UAE North
   - **Tier**: Select Standard V2.
   - **Virtual network**: Select the same virtual network in which the Kubernetes cluster exists.
   - **Subnet**: Select the created subnet for the application gateway.

- Keep the other settings as default and then select the **Next: Frontends**.

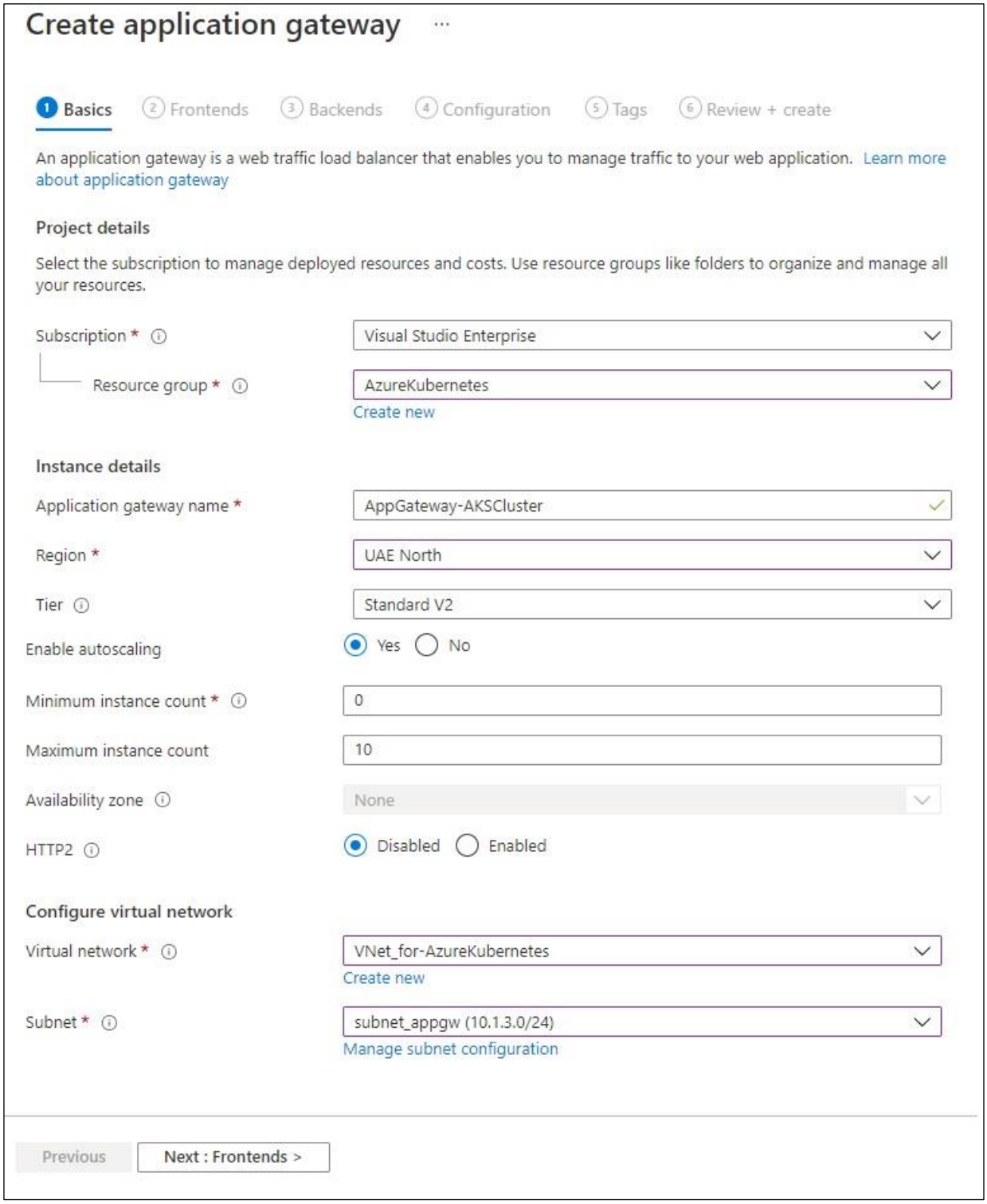

Figure 2.37

5. Set the Frontend IP address type as **Public**.

6. Select **Add new** for the **Public IP address** and enter a user-defined name that is, *appgwpublicip* and then click **OK**.

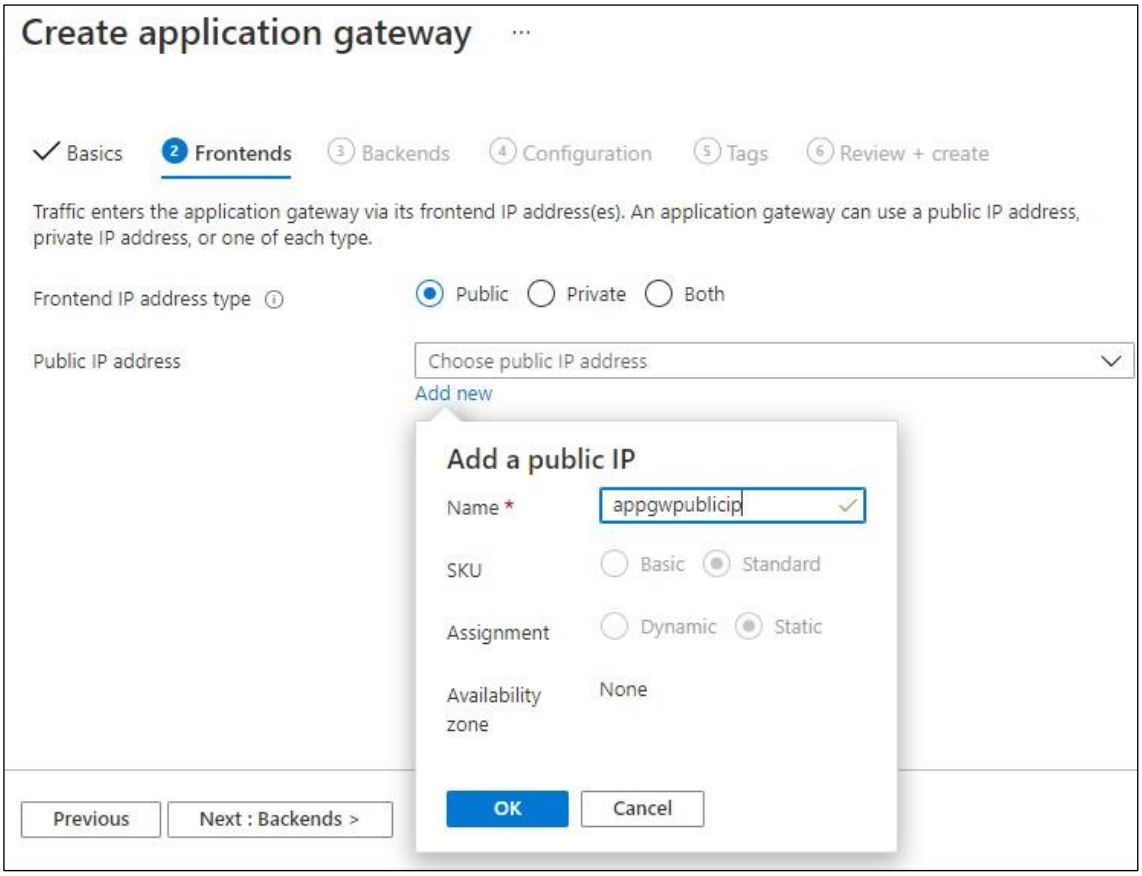

Figure 2.38

7. Select **Next: Backends**. The **Backends** tab appears.
8. Select **add a backend pool**. The Add a backend pool dialog appears.
9. Enter the following details to create an empty backend pool:
   - **Name**: Enter a user-defined name that is, **appgwbackendpool**.
   - **Add backend pool without targets**: Select **Yes** to create a backend pool with no targets.
   - Select **Add** to save the backend pool configuration and return to the **Backends** tab.

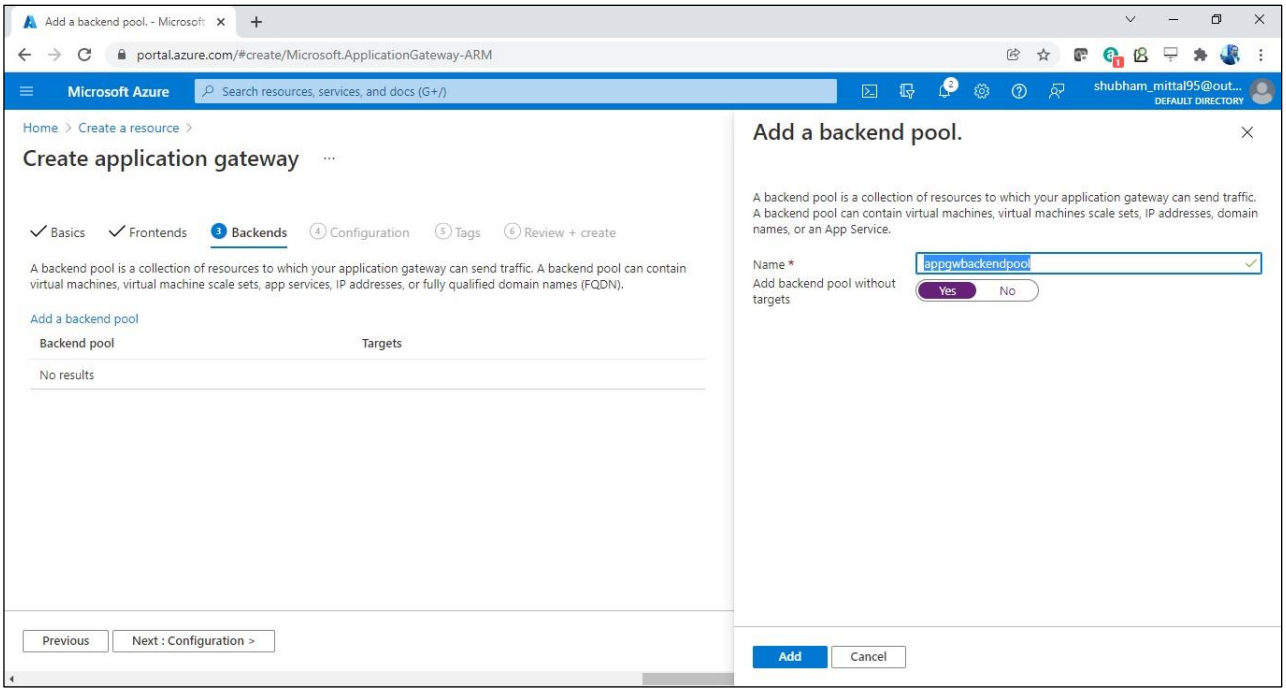

Figure 2.39

10. On the **Backends** tab, select **Next: Configuration.** The Configuration tab appears.
11. Select **Add a routing rule** in the **Routing rules** column. The Add a routing rule dialog appears.
12. Enter the user-defined rule name that is, **routingrule1.**
13. A routing rule requires a listener. On the **Listener** tab, enter the following details:
    - **Listener name**: Enter a user-defined listener name that is, **appgwlistener**.
    - **Frontend IP**: Select Public to select the public IP that you have created in the Frontends tab.
    - Keep the other settings as default and switch to the **Backend targets** tab.

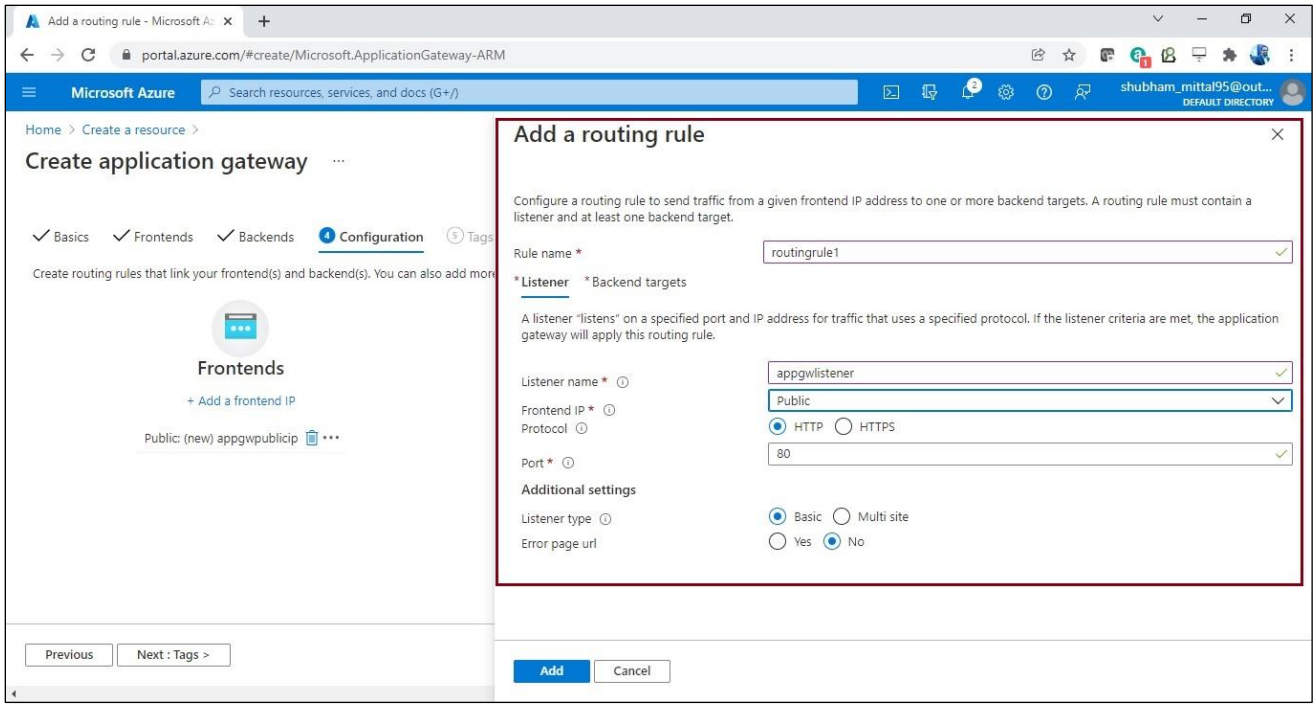

Figure 2.40

14. In the **Backend targets** tab, select the backend pool created in the **Backends** tab for the **Backend target**.
15. For the **HTTP settings**, select **Add new** to add a new HTTP setting.

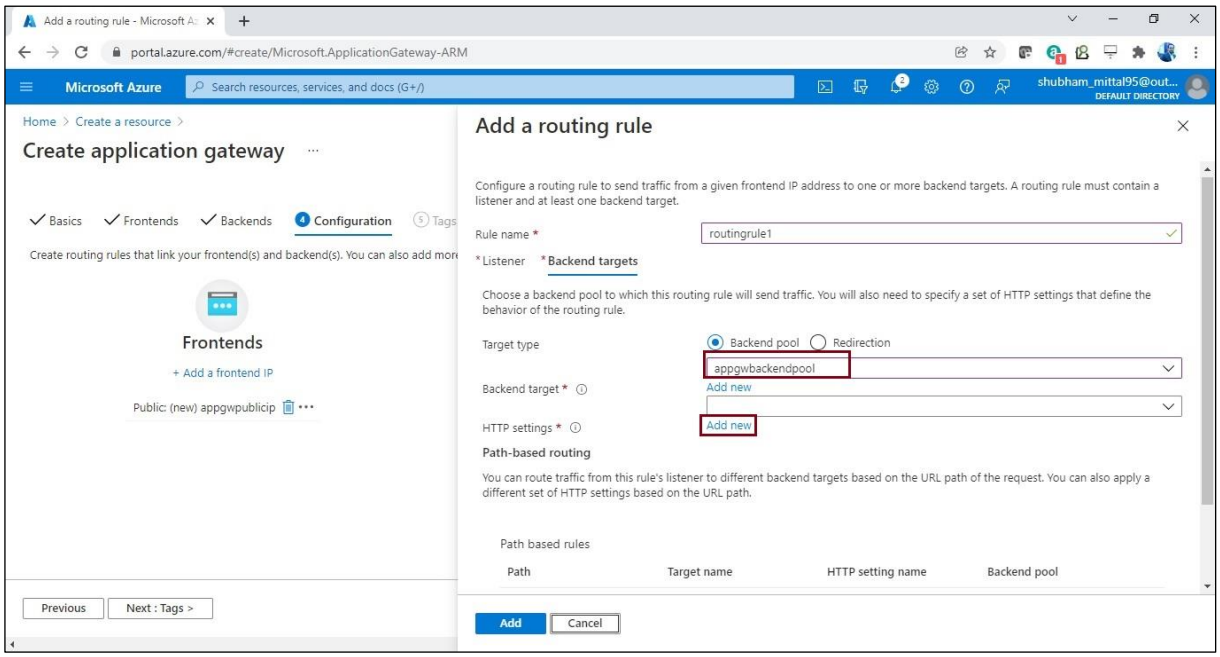

Figure 2.41

16. In the Add an HTTP setting, enter the user-defined HTTP setting name that is, **appgwhttpsetting**.
17. Keep the other settings as default and then click **Add** to return to the Add a routing rule.

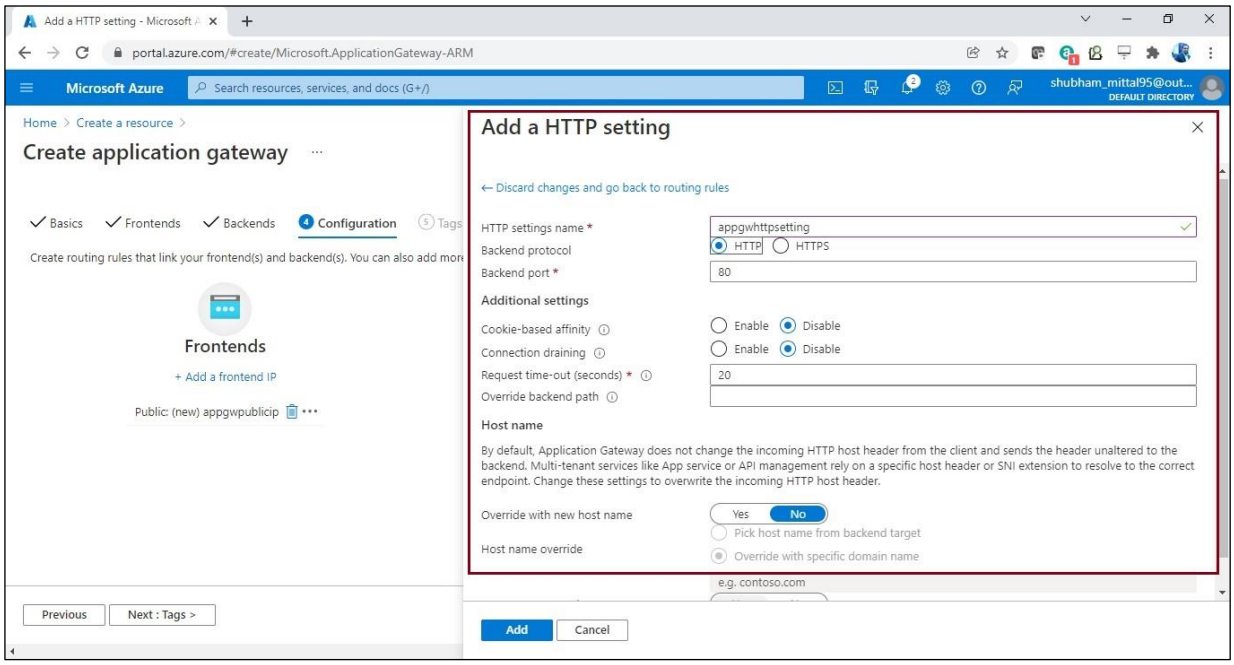

Figure 2.42

18. Select **Add** to save the routing rule in the Add a Routing and return to the **Configuration** tab.
19. Select **Next: Tags** and then click **Next: Review + create**.
20. Once validation is passed, select **Create**.

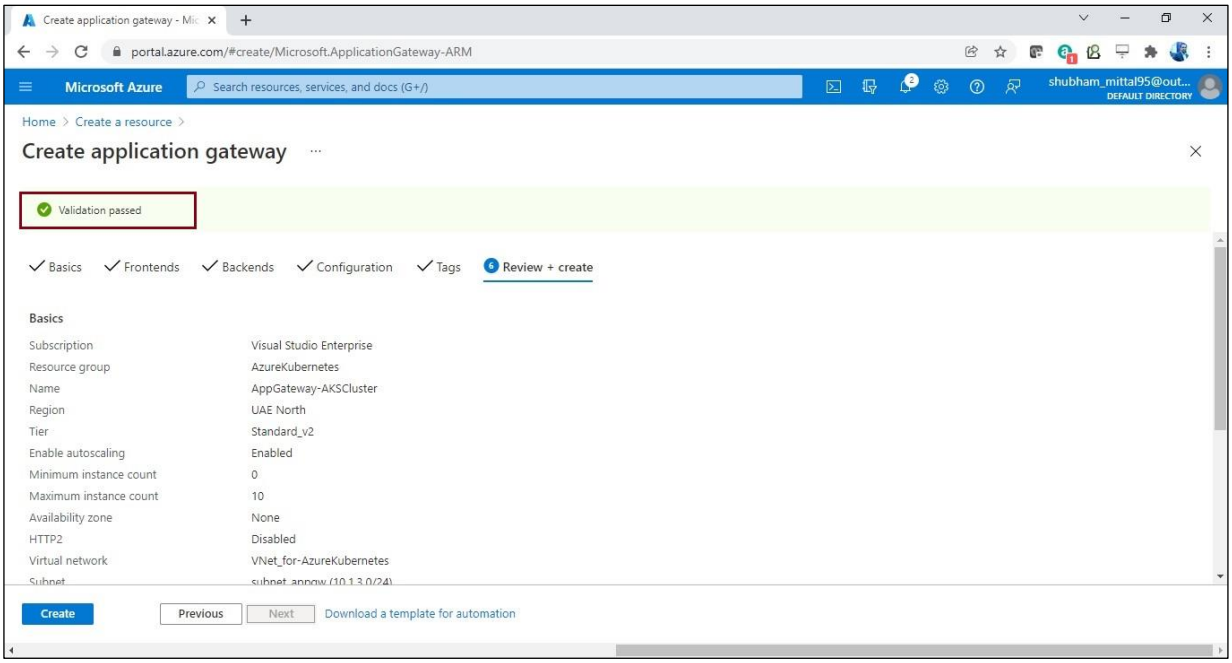

Figure 2.43

21. Once the deployment is complete, click **Go to resource**.

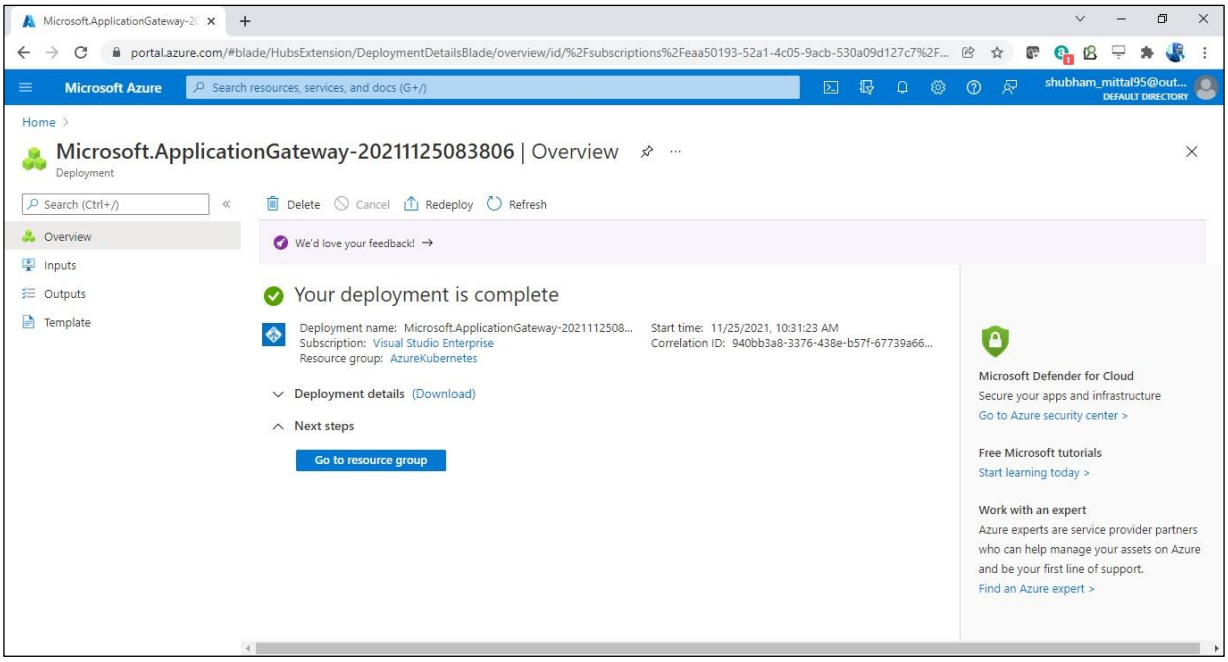

Figure 2.44

2.6.2 Installation of an application gateway ingress controller

An ingress controller is a piece of software that provides reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. Kubernetes ingress resources are used to configure the ingress rules and routes for individual Kubernetes services. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster.

Pre-requisites:

- Azure Kubernetes Service must be created.
- Application Gateway must be created.

To install an Application Gateway Ingress Controller (AGIC), follow the below steps:

- Install Helm
- Azure Resource Manager Authentication using a Service Principle
- Install Ingress Controller using Helm

2.6.2.1 Install Helm

1. If you use the Azure Cloud Shell https://portal.azure.com/#cloudshell/ then the Helm CLI is already installed. To install Helm on other platforms please refer to https://helm.sh/docs/intro/install/.

2. Open the Azure Cloud Shell and run the following command to add the **application-gateway-kubernetes-ingress** helm package.

```
helm repo add application-gateway-kubernetes-ingress
https://appgwingress.blob.core.windows.net/ingress-azure-helm-package/
helm repo update
```

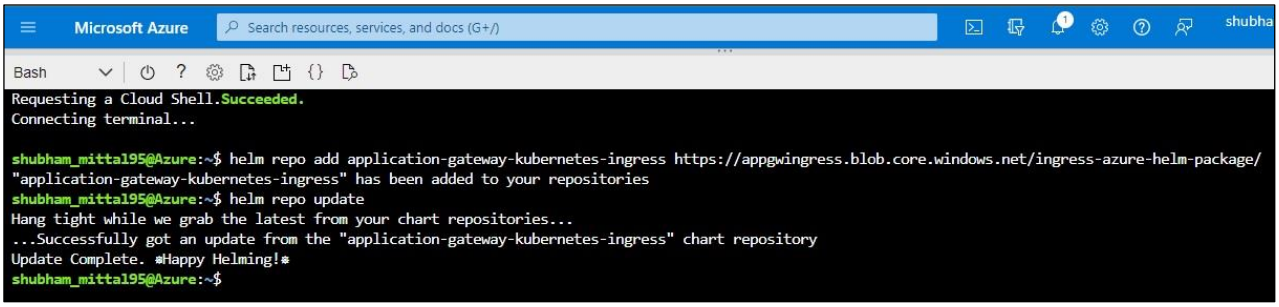

Figure 2.45

2.6.2.2 ARM authentication using a service principle

Perform the below steps for ARM Authentication using a service principle:

1. Application Gateway Ingress Controller (AGIC) communicates with the Kubernetes API Server and Azure Resource Manager. It requires authentication to access these APIs
2. Open the Azure Cloud Shell https://portal.azure.com/#cloudshell/ and run the following command to create a service principle and encode with base64. The base64 encoding is required for the JSON blob to be saved to Kubernetes.

```
az ad sp create-for-rbac --role Contributor --sdk-auth --scopes
/subscriptions/<Subscription-id>/resourceGroups/<Resource group> | base64 -
w0
```

Where,

Subscription-id – Enter your account subscription id

Resource group – Ente the name of resource group associated with kubernetes cluster

For Example –

```
az ad sp create-for-rbac --role Contributor --sdk-auth --scopes
/subscriptions/323527f6b-535a1-406d-239b-0972646c8500c3/resourceGroups/
AzureKubernetes | base64 -w0
```

NOTE:

Keep the **base64 encoded JSON blob** as these values are required in the following steps for installing AGIC.

2.6.2.3 Add or update Kubeconfig file

Perform the below steps to add or update kubeconfig file:

1. Open the Azure Cloud Shell https://portal.azure.com/#cloudshell/ .
2. Delete the **.kube/config** file (if already exists) using below command:

```
rm .kube/config
```

3. Now execute the below command to re-create .kube/config file:

```
az aks get-credentials --resource-group <ResourceGroupName> --name
<AzureEKSClusterName>
```

For example,

```
az aks get-credentials --resource-group AzureKubernetes --name BPMSuite-
AKSCluster
```

2.6.2.4 Install ingress controller using Helm

Perform the below steps to install ingress controller using Helm:

1. Open the Azure Cloud Shell https://portal.azure.com/#cloudshell/ and run the following command to download the *helm-config.yaml* file which configures the Application Gateway Ingress Controller.

```
wget https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-ingress/master/docs/examples/sample-helm-config.yaml -O helm-config.yaml
```

2. Edit the helm-config.yaml file and fill in the values for **appgw** (Application Gateway) and **armAuth** (ARM Authentication using Service Principle).

```
nano helm-config.yaml
```

3. Update the <**subscriptionId**>, <**resourceGroupName**>, and <**applicationGatewayName**> for **appgw**.
   For example,

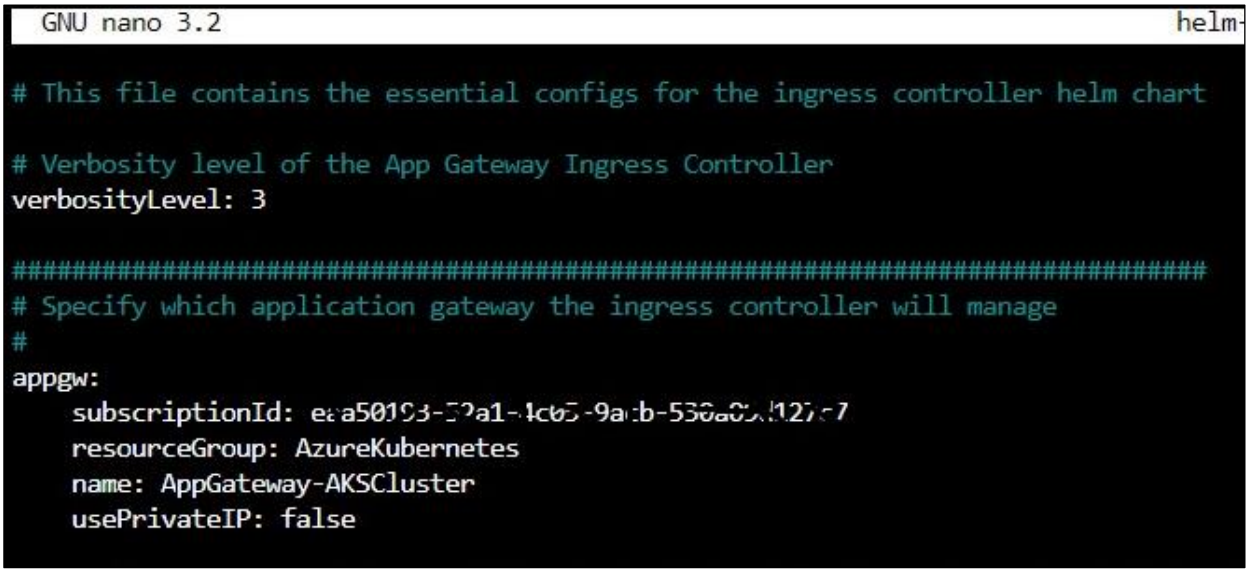

Figure 2.46

4. Comment the **armAuth** using **AAD-Pod-Identity** and uncomment the **armAuth** using **Service Principle**.
5. Update the **base64 encoded JSON blob** created in the previous step 'ARM Authentication using a Service Principle' for **secretJSON**.
   For example,

```
Bash
GNU nano 3.2                                                    helm-config.yaml

#   watchNamespace: <namespace>

################################################################################
# Specify the authentication with Azure Resource Manager
#
# Two authentication methods are available:
# - Option 1: AAD-Pod-Identity (https://github.com/Azure/aad-pod-identity)
#armAuth:
#    type: aadPodIdentity
#    identityResourceID: <identityResourceId>
#    identityClientID:  <identityClientId>

## Alternatively you can use Service Principal credentials
 armAuth:
    type: servicePrincipal
    secretJSON: ewogICJjbGllbnRJZCI6ICJhZGJhYjQ5ZS1iNGI3LTQ0YjAtYjc2ZC0zMjM4N2ZhYWQzODQiLAogICJjbGllbnRT

################################################################################
# Specify if the cluster is RBAC enabled or not
rbac:
    enabled: false # true/false
```

Figure 2.47

6. Specify the rbac enabled as **true** if the cluster is RBAC enabled.
   For example,

```
################################################################
# Specify if the cluster is RBAC enabled or not
rbac:
    enabled: true # true/false
```

Figure 2.48

7. Install Helm chart **application-gateway-kubernetes-ingress** with the *helm-config.yaml* configuration from the previous step.

```
helm install ingress-azure \
  -f helm-config.yaml \
  application-gateway-kubernetes-ingress/ingress-azure \
  --version 1.4.0
```

For example,

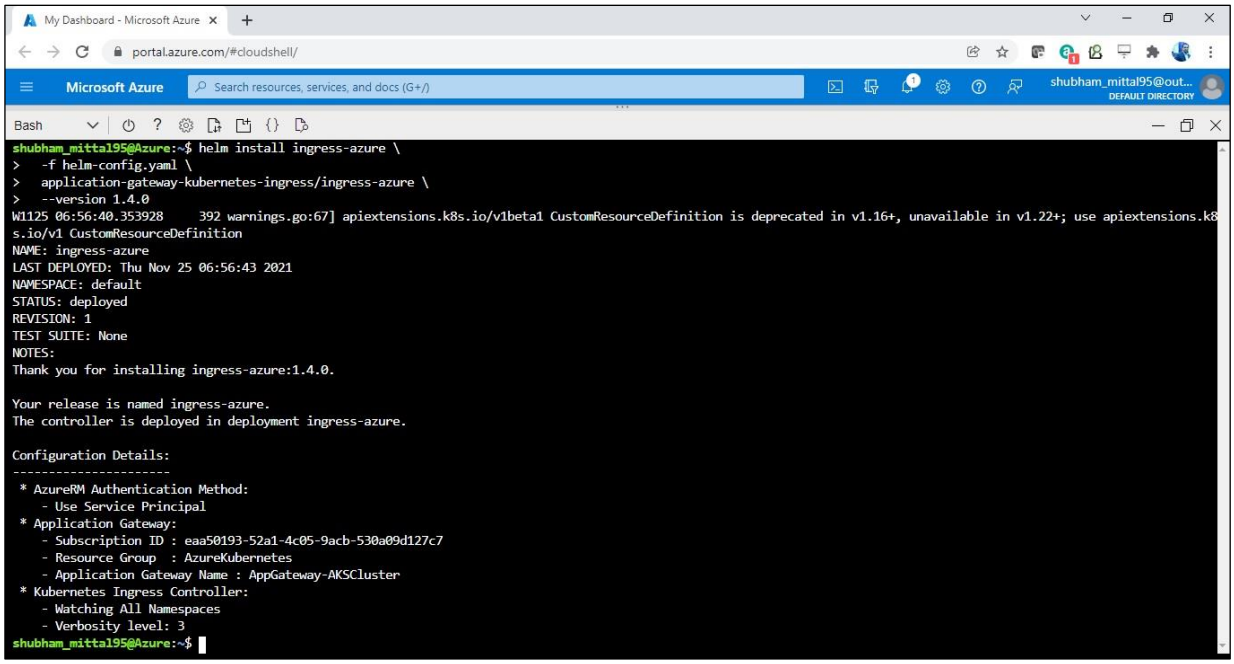

Figure 2.49

NOTE:

Use the latest version of ingress-azure. You can get the release information from the below link:
https://github.com/Azure/application-gateway-kubernetes-ingress/releases

---

8. Application Gateway Ingress Controller runs as a pod in the Kubernetes cluster. You can check the running status of the AGIC pod using the below command:

```
Kubectl get po | grep ingress
```

For example,

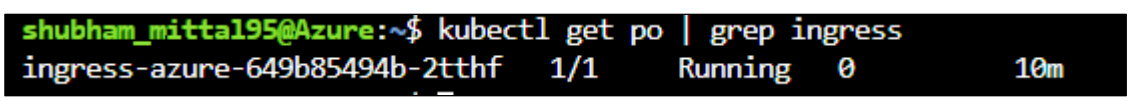

Figure 2.50

2.7 Configuration of DNS zone

Ingress Controller creates a Load Balancer and routes the incoming requests to the target Kubernetes services according to the host-based routing rules. Host-based routing is a capability of Ingress Controller that redirects the user requests to the right service based on the request-host header.

For example, you can set the rules as below:

- IF URL is 'ibps5serviceinstance.azure.co.in' then redirect to iBPS ServiceInstance Web container.
- IF URL is 'ibps5userinstance.azure.co.in' then redirect to the iBPS UserInstance Web container.

To support the host-based routing, you must register a custom domain and create a new RecordSet in DNS Zone for each host-path.

Perform the below steps to create a DNS Zone:

1. Sign in to the Azure Portal using https://portal.azure.com.
2. After a successful sign in, click **Create a resource** and search for the **DNS Zone**.

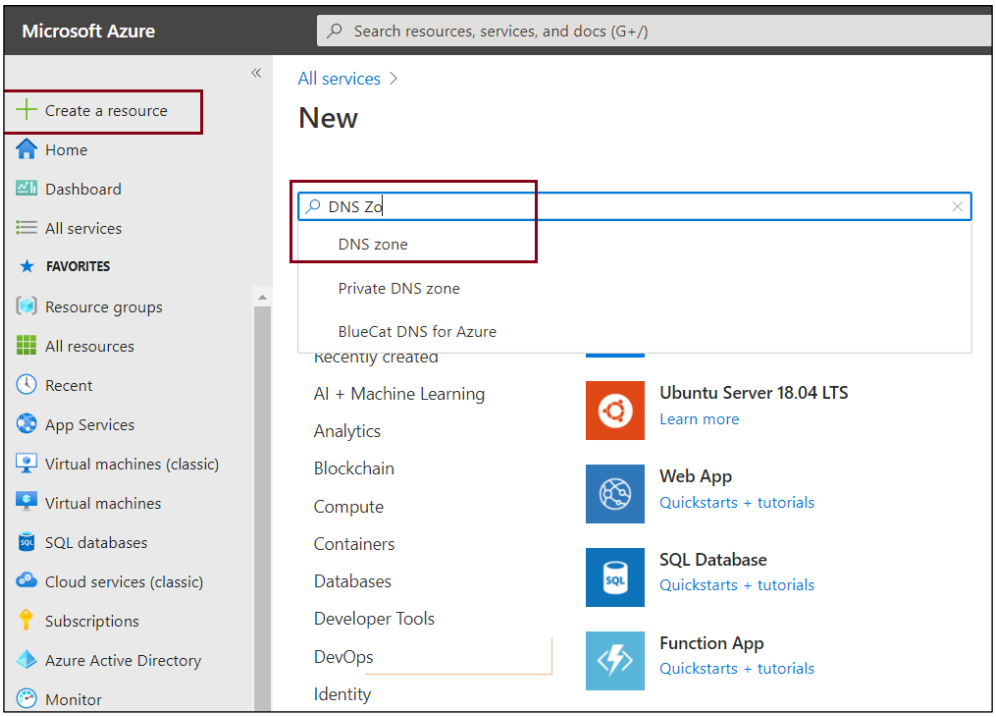

Figure 2.51

3. Click **Create**.

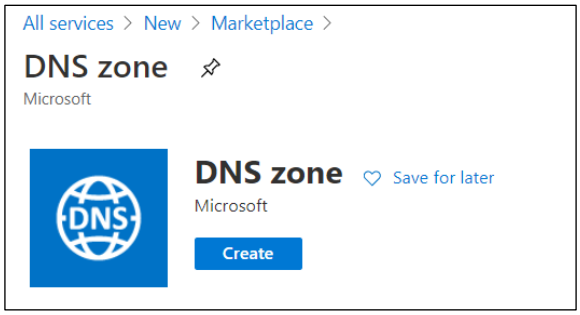

Figure 2.52

4. On the **Create DNS zone**, specify the following details under the **Basics** tab:
   a. **Subscription**: Select a valid Azure subscription.
   b. **Resource group**: Select or create an Azure Resource group, such as **AzureKubernetes**.
   c. **Name**: Specify a valid DNS Zone name such as **azure.co.in.**
   d. Click **Next: Tags**>.

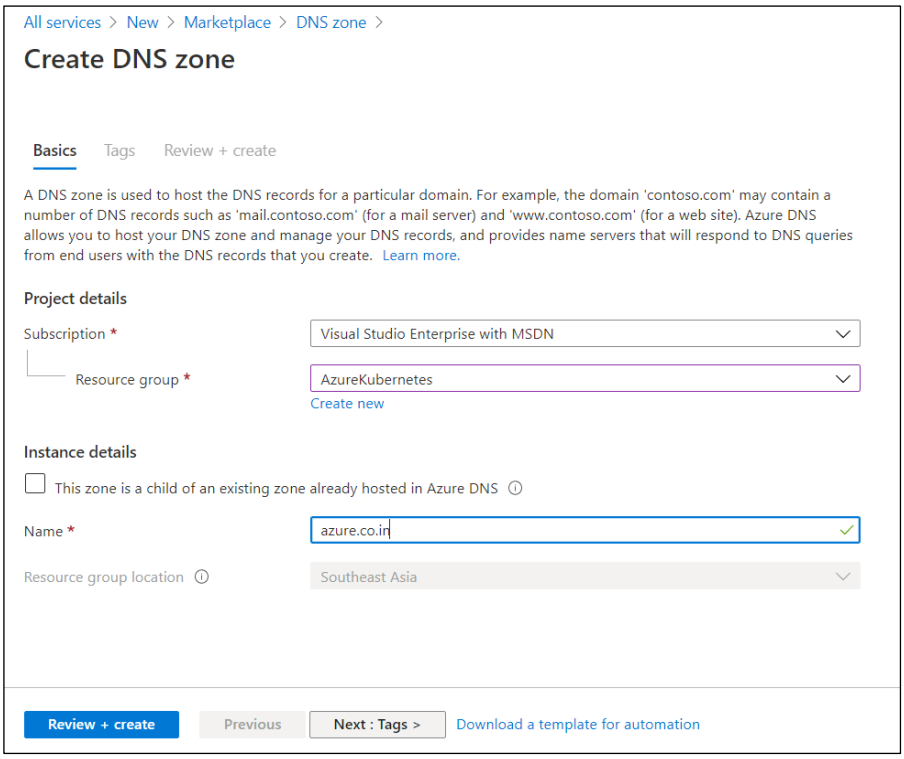

Figure 2.53

5. On the **Tags** tab, keep the default options and click **Next: Review + create>**.
6. On the **Review + create** tab, click **Create** once validation is passed.

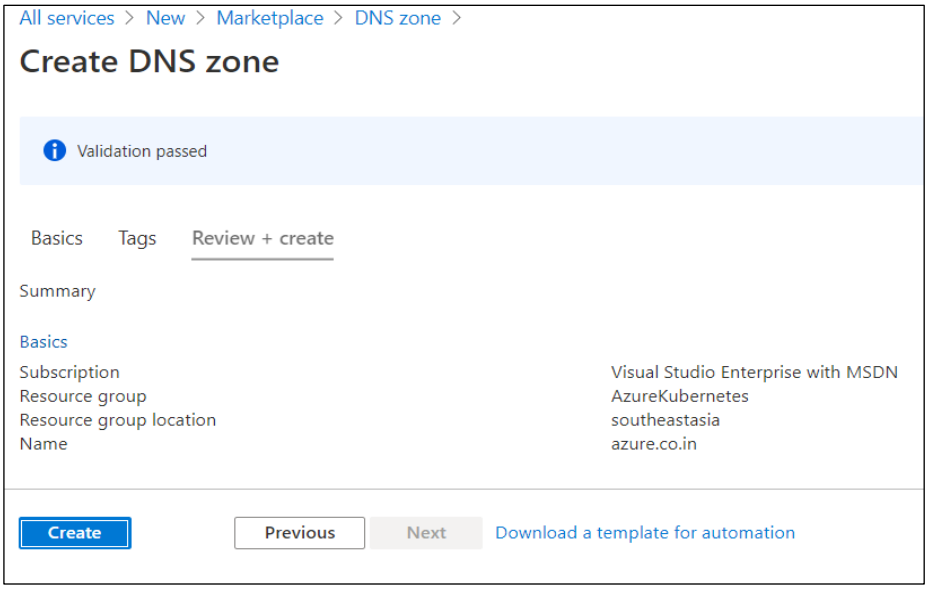

Figure 2.54

7. Once deployment is complete, click **Go to resource**. The Created DNS Zone's Overview screen appears.

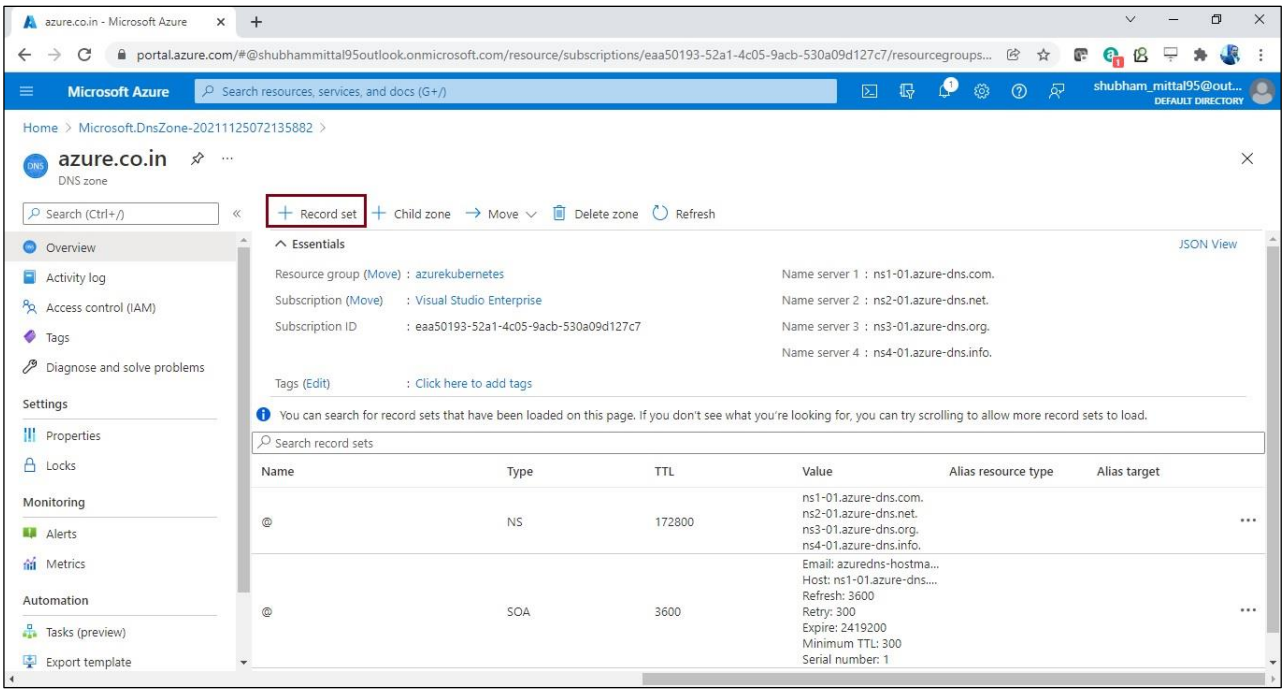

Figure 2.55

8. On the top of the **DNS Zone** tab, select **+ Record set**.
9. On the **Add record set** tab, type or select the following values:
    a. **Name**: Enter the user-defined name.

b. **Type**: Select type as “A – IPv4-address”
c. **Alias record set**: Select alias as **Yes**.
d. **Alias type**: Select the alias type as **Azure resource**.
e. **Choose a subscription**: Select a valid Azure subscription.
f. **Azure resource:** Select the Public IP Address created for the Application Gateway that is, **appgwublicip**.
g. **TTL (Time To Live):** Time-to-live of the DNS request specifies how long DNS servers and clients can cache a response.

NOTE:
There is no change in the default value.

h. Click **OK** to save the record set**.**

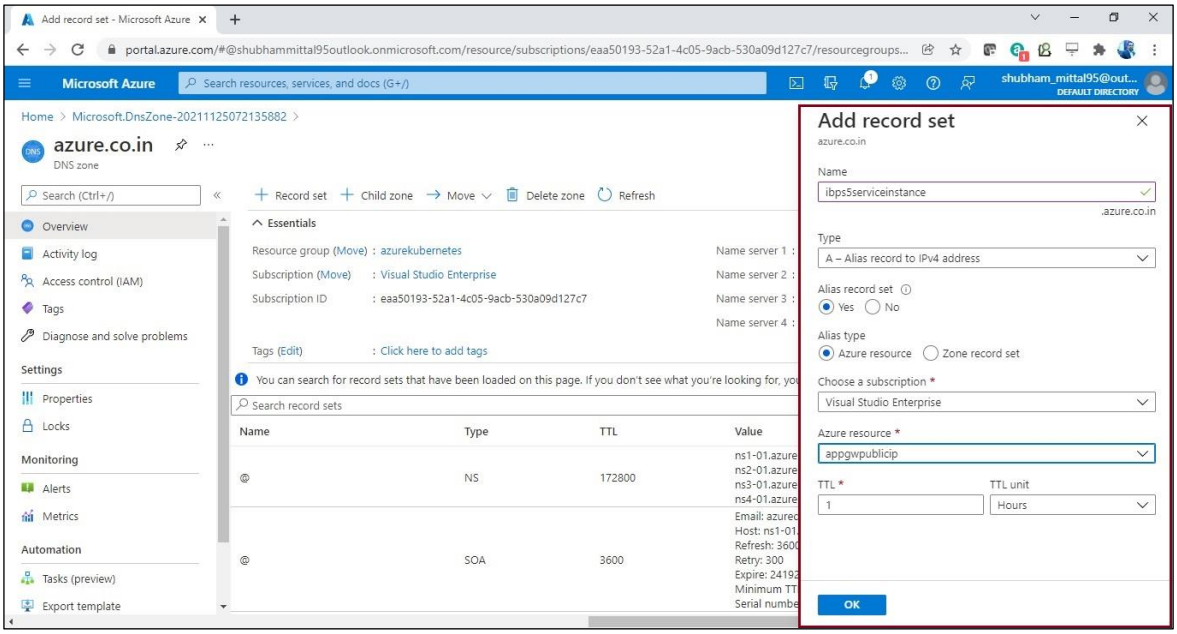

Figure 2.56

10. Similarly, you can add other record sets for each host-path defined in *AppGateway-IngressController.yaml* file.
    For example,

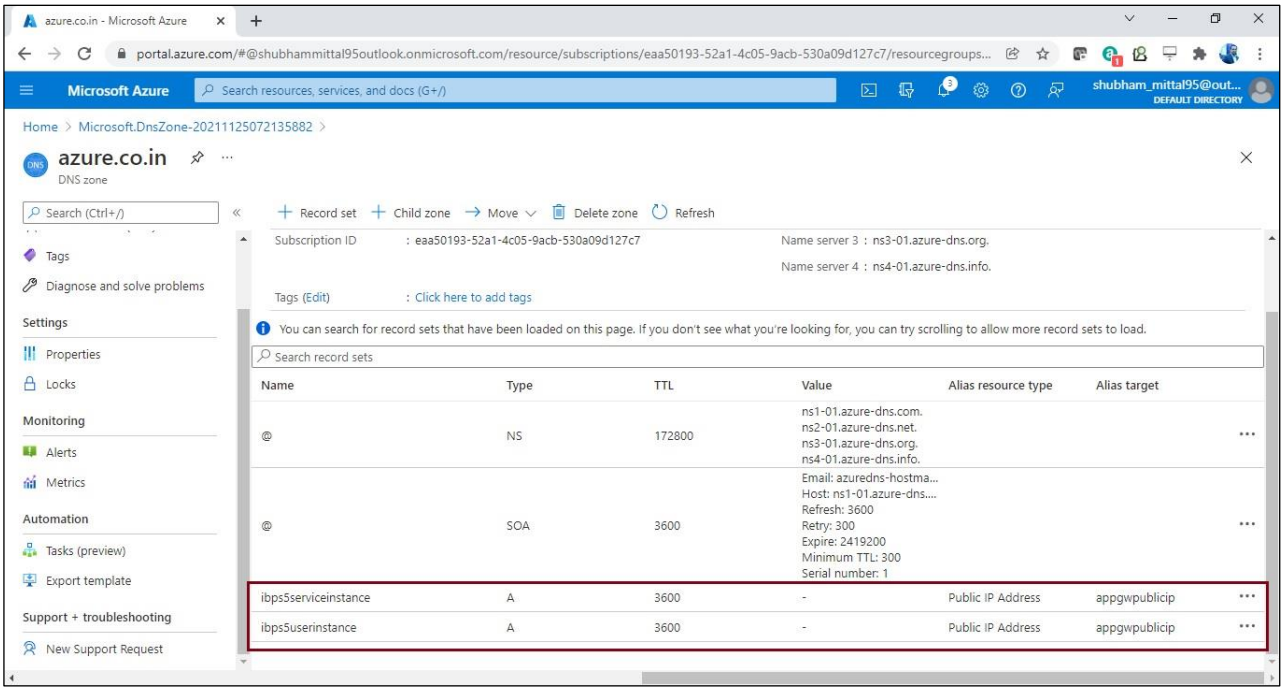

Figure 2.57

2.8 Run Kubectl from local machine

Before running the kubectl commands from your local machine, you must have the following pre-requisites:

- kubectl: https://kubernetes.io/docs/tasks/tools/install-kubectl/
- azure-cli: https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli
- Delete the **.kube** folder from C:\Users\<Logged-in UserName> folder if exists.
- Now execute the below command to worker node:

```
az aks get-credentials --resource-group <ResourceGroupName> --name
<AzureEKSClusterName>
```

For example,

```
az aks get-credentials --resource-group AzureKubernetes --name
BPMSuite-AKSCluster
```

- Once you have run the above command to connect to the AKS cluster, you can run any kubectl commands. Here are a few examples of useful commands you can try.
For example,

```
# List all the pods
        kubectl get pods
```

```
# List all deployments in all namespaces
kubectl get deployments --all-namespaces=true

# List all deployments in a specific namespace
# Format :kubectl get deployments --namespace <namespace-name>
kubectl get deployments --namespace kube-system
```

2.9 Monitor Kubernetes dashboard

- The Azure portal includes a Kubernetes resource view for easy access to the Kubernetes resources in your Azure Kubernetes Service (AKS) cluster.
- To see the Kubernetes resources, navigate to your AKS cluster in the Azure portal. The navigation pane on the left is used to access your resources. The resources include:
  - **Namespaces:** Displays the namespaces of your cluster. The filter at the top of the namespace list provides a quick way to filter and display your namespace resources.
  - **Workloads:** Displays information about deployments, pods, replica sets, stateful sets, daemon sets, jobs, and cron jobs deployed to your cluster.
  - **Services and ingresses**: Display all of your cluster's service and ingress resources.
  - **Storage:** Displays your Azure storage classes and persistent volume information.
  - **Configuration:** Displays your cluster's config maps and secrets.

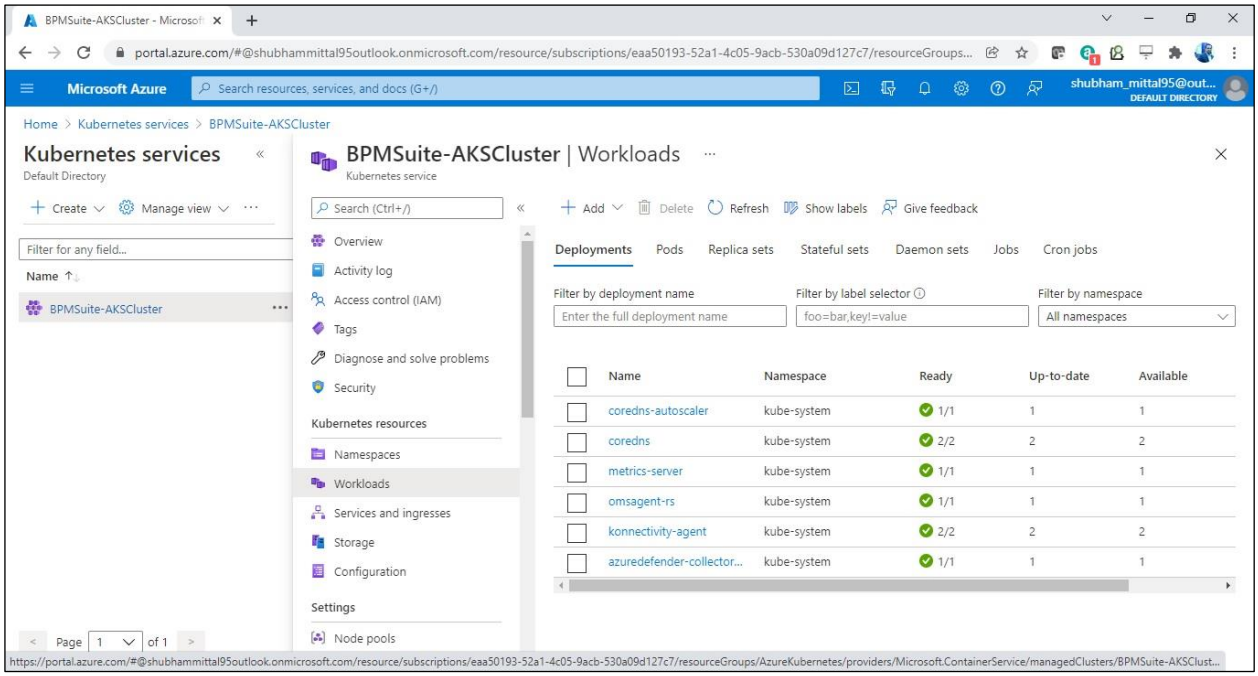

Figure 2.58

2.10 Azure monitor for container insights

Azure Monitor for containers is a feature designed to monitor the health and performance of container workloads deployed to Azure Kubernetes service. It delivers a comprehensive monitoring experience and gives us performance visibility by collecting memory and processor metrics from controllers, nodes, and containers that are available in Kubernetes through the Metrics API.

By default, Azure Monitor is enabled for container monitoring during Azure Kubernetes service creation (Under Integrations tab).

For example,

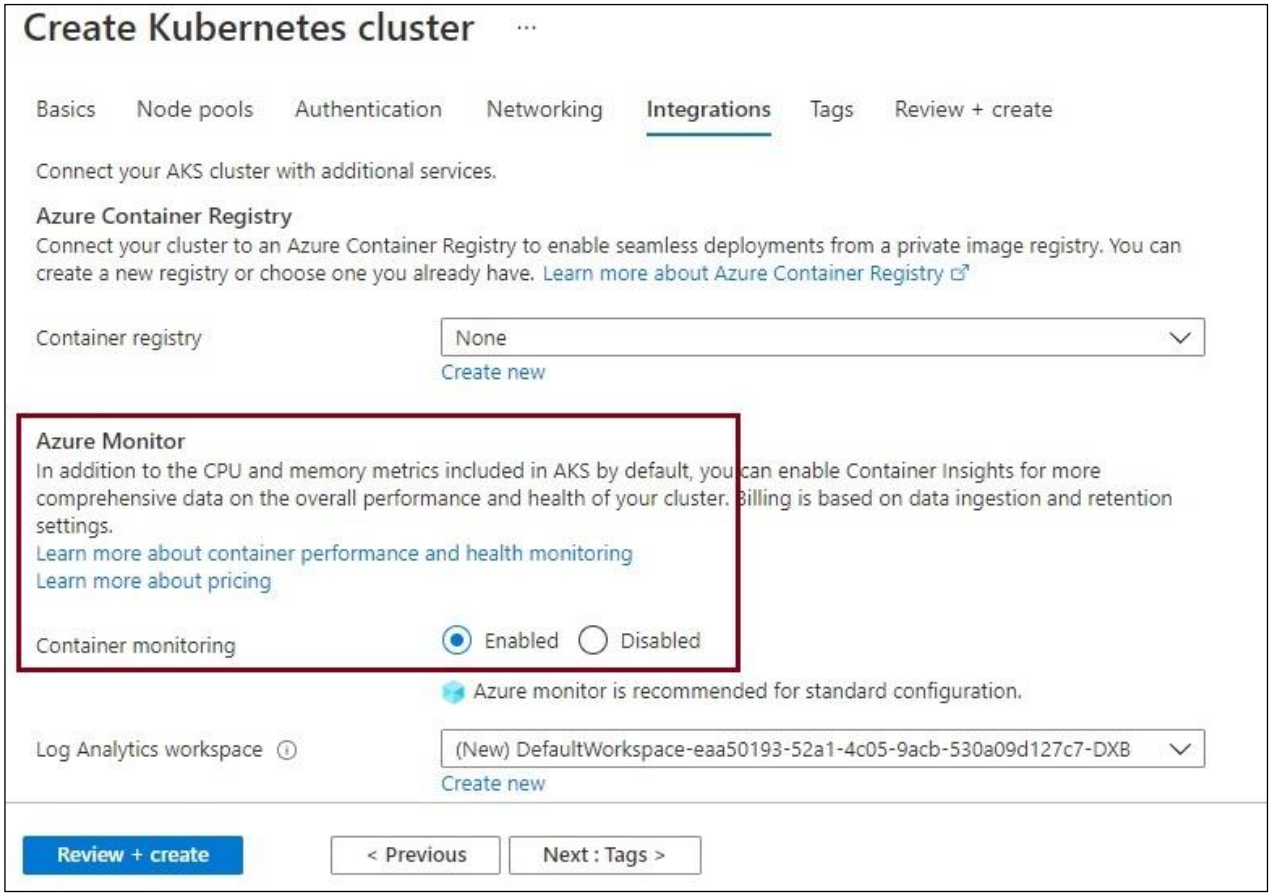

Figure 2.59

Perform the below steps to view the container insights:

1. Sign in to the Azure portal at https://portal.azure.com.
2. On the Azure portal menu or from the Home page, select **All resources.**
3. Click on the created **Kubernetes service.**
4. Click **Monitoring >> Insights**. Here are the series of tabs to monitor your AKS Cluster, Nodes, Containers, Controllers, and so on.

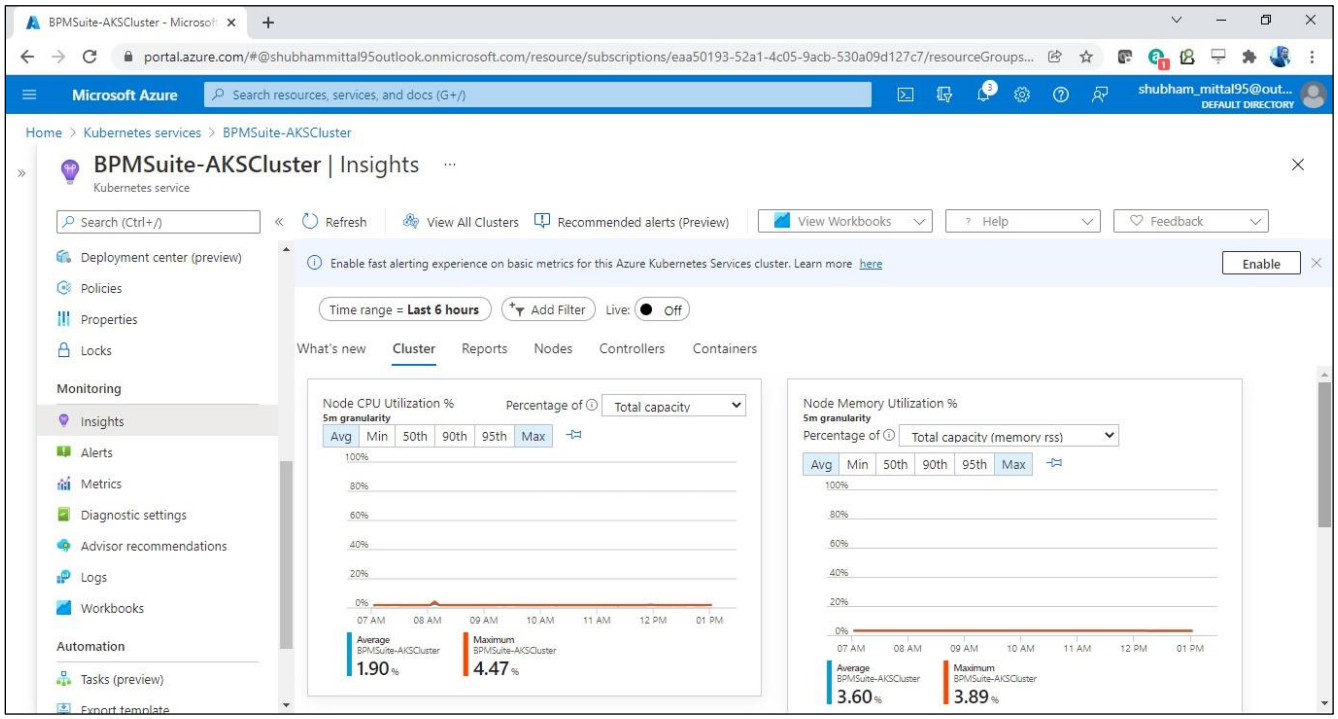

Figure 2.60

3 Deployment of OmniDocs containers on Azure Kubernetes service

This section describes the deployment of OmniDocs containers. Refer the below sections for procedural details.

3.1 Prerequisites

Azure Kubernetes Service must be configured, and its Worker nodes must be in Ready state.

NOTE:
Refer to the Configuration of Azure Kubernetes Cluster for the configuration of Azure Elastic Kubernetes Service.

3.2 Deliverables

Newgen has isolated the product suite into multiple Docker containers to enable the independent scalability of each Docker container. This separation is done based on the product's usability. At a broad level, Web components and EJB components are isolated for deployment in separate container instances. Web components is deployed on the underlying web server JBoss WebServer 6.0.x. EJB components is deployed on the underlying application server JBoss EAP 7.4.x. Newgen has released multiple Docker images for the different product suites along with some configuration files for data persistence, YAML files for deployment, and some documentation for end-to-end configurations and deployments.
The followings are the list of deliverables:

The Newgen delivers the following:

- Docker Images
- Configuration Files
- YAML Files

3.2.1 Docker images

The following Docker images are delivered for the initial product deployment:

- OmniDocs Web Components
- OmniDocs Web Service Components
- OmniDocs EJB Components
- OmniDocs Add-on Services (Wrapper, AlarmMailer, Scheduler, ThumbnailManager and LDAP)
- EasySearch (Apache Manifold only)
- Text Extraction Manager or Full-Text Search (TEM/FTS)
- OmniScan Web Components
- OmniDocs WOPI

NOTE:
These Docker images can be delivered to a private Docker repository like ACR (Azure Container Registry) or in the form of compressed files that can be shared over the FTP or similar kind of media.

3.2.2 Configuration files

Configuration files are dynamic in nature and data is written at runtime. Database details in configuration files such as *Server.xml* and *standalone.xml* are written at runtime. These types of files must be kept outside the container to persist the data. Here, Azure FileShare is used to persist configuration files.

The following configuration files are shared for OmniDocs Docker images:

- OmniDocsWeb
- OmniDocsEJB
- ODServices
- EasySearch
- TEM
- OmniScanWeb7.0
- OmniDocsWOPI

3.2.3 YAML files

YAML files stands for "YAML Ain't Markup Language". It is a human-readable object configuration file that is used to deploy and manage the objects on the Kubernetes cluster. In other words, it is a manifest file that contains the deployment descriptor of Kubernetes containers. You can execute YAML files using "kubectl apply –f <YAMLFile>" or use these files in Azure DevOps Release Pipeline to deploy the containers.

The following configuration files has shared for OmniDocs Docker images:

- OmniDocsWeb.yml
- OmniDocsWeb_Services.yml
- OmniDocsEJB.yml
- OmniDocsServices.yml
- EasySearch_ApacheOnly.yml
- TEM.yml
- OmniScanWeb7.0.yml
- OmniDocswopi.yml
- AzureFile_PV_PVC.yml
- AppGateway-IngressController.yml

Here's an example of a YAML file:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dev
---

apiVersion: apps/v1
kind: Deployment
metadata:
#Name should not be more than 13 letters
  name: odll0web
  namespace: dev
spec:
  replicas: 1
  selector:
      matchLabels:
        name: odll0web
  strategy:
   type: RollingUpdate
   rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
  template:
    metadata:
      labels:
        name: odll0web
    spec:
      containers:
      - name: odll0web
        image: k8snonprdcidevacr.azurecr.io/omnidocs11.0web:#{RELEASE.ARTIFACTS._OMNIDOCS11.0WEB.BUILDID}#
        imagePullPolicy: Always
        securityContext:
          runAsNonRoot: true
        resources:
          requests:
            memory: 2048Mi
            cpu: 800m
          limits:
```

Figure 3.1

AzureFile_PV_PVC.yml file is used for Persistent Volume and Persistent Volume Claim. Persistent Volume (PV) is a storage piece in the cluster that is provisioned using Storage Classes. It contains the Azure FileShare **secretName** and **shareName** that is already created during Azure FileShare creation. It is also used to set the access permission on Azure FileShare using **mountOptions** attribute.

A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (for example, they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany).

AppGateway-IngressController.yml is used for the ingress controller. An ingress controller is an object running inside the Kubernetes cluster that is used to manage the host-based routing rules. For example, you can set the host-based routing rules like if the URL is *omnidocs.newgendocker.com* then the ingress controller redirects the user request to OmniDocs

WEB containers. and if the URL is *omniscan.newgendocker.com* then it redirects the user request to the OmniScan WEB containers.

NOTE:
You can store the above YAML files in Azure Repo that is used by Azure DevOps Release Pipeline.

3.3 Changes in product's YAML files

The changes in the Product's YAML Files are as follows:

- **Namespace**: In the YAML files, default namespace is given as **dev.** You can change this name as per your requirement.

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev
```

Figure 3.2

- **Name:** In the *OmniDocsWeb.yml* file, **od110web** is given as the default name of Kubernetes objects - deployment, replicas, container, and service. You can change this name as per your requirement. While changing the name, ensure that this name is not more than 13 letters in length and must contain small letters only.
  For example,

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
#Name should not be more than 13 letters
name: od110web
spec:
replicas: 1
selector:
    matchLabels:
        name: od110web
strategy:
type: RollingUpdate
rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
template:
    metadata:
    labels:
        name: od110web
    spec:
    containers:
    - name: od110web
```

Figure 3.3

- **Replica:** In the *OmniDocsWeb.yml* file, the default replica is given as **1**. That means only one container is created after the deployment. You can increase this number as per our choice.
- **Image:** In the *OmniDocsWeb.*yml file, update the **image** location. By default, the below value is given:

```yaml
image: newgencontainerregistry.azurecr.io/omnidocs11.0web:#{RELEASE.ARTIFACTS._OMNIDOCS11.0WEB.BUILDID}#
```

Here:

- **newgencontainerregistry.azurecr.io -** It's the name of the Azure Container Registry.
- **omnidocsweb –** It's the OmniDocsWeb Docker image name.

- **#{RELEASE.ARTIFACTS._OMNIDOCSWEB.BUILDID}#:** It's a Docker image's tag name in the form of a dynamic variable whose value gets updated at runtime using AzureDevOps Release Pipeline. Specify the static tag name like latest, build-number1, and so on.

- **SecurityContext:** In the *OmniDocsWeb.yml* file, **SecurityContext [runAsNonRoot: true]** is defined. It means the OmniDocsWeb container can never be run with root privileges. If any container tries to run with the root user, then Kubernetes stops its deployments.

```
securityContext:
  runAsNonRoot: true
```

Figure 3.4

- **Resource Request and limit:** In the *OmniDocsWeb.yml* file, resource request and resource limit parameters are defined. The **request** parameter specifies the minimum required resources to run the particular container and the **limit** parameter specifies the maximum resource limit that a container can use. In other words, a running container is not allowed to use more than the resource limit you set.
  For Example,

```
requests:
    memory: 1024Mi
    cpu: 800m
limits:
    memory: 2048Mi
    cpu: 1000m
```

Figure 3.5

Here, 1000m CPU = 1 Core CPU

The above-specified limit is the minimum required resource to run a container. If users are increasing, then you must increase the limit range accordingly.

- **VolumeMounts and Volume:** Volume mounts and volumes are used to persist the data outside the container so that whenever the container terminates due to any reason our data is always persisted. In the *OmniDocsWeb.yml* file, we have persisted configuration files or folders and log files.

```
volumeMounts:
- name: azurefile-volume-mount
  mountPath: /Newgen/jws-6.0/tomcat/bin/Newgen/NGConfig
  subPath: OmniDocs11.0SP1/OmniDocs11.0Web/Newgen/NGConfig
- name: azurefile-volume-mount
  mountPath: /Newgen/jws-6.0/tomcat/conf/web.xml
  subPath: OmniDocs11.0SP1/OmniDocs11.0Web/web.xml
- name: azurefile-volume-mount
  mountPath: /Newgen/jws-6.0/tomcat/conf/redisson.yaml
  subPath: OmniDocs11.0SP1/OmniDocs11.0Web/redisson.yaml
- name: azurefile-volume-mount
  mountPath: /Newgen/jws-6.0/tomcat/lib/jboss-ejb-client.properties
  subPath: OmniDocs11.0SP1/OmniDocs11.0Web/jboss-ejb-client.properties
- name: azurefile-volume-mount
  mountPath: /Newgen/jws-6.0/tomcat/logs
  subPathExpr: OmniDocs11.0SP1/ProductLogs/odl10web-#{RELEASE.RELEASENAME}#/tomcat_logs/$(POD_NAME)
- name: azurefile-volume-mount
  mountPath: /Newgen/jws-6.0/tomcat/bin/Newgen/NGLogs
  subPathExpr: OmniDocs11.0SP1/ProductLogs/odl10web-#{RELEASE.RELEASENAME}#/NGLogs/$(POD_NAME)
- name: azurefile-volume-mount
  mountPath: /Newgen/jws-6.0/tomcat/bin/Newgen/NGTemp
  subPath: OmniDocs11.0SP1/ProductLogs/odl10web-#{RELEASE.RELEASENAME}#/NGTemp
- name: azurefile-volume-mount
  mountPath: /Newgen/jboss-eap-7.4/bin/SystemReports
  subPath: OmniDocs11.0SP1/SystemReports
```

Figure 3.6

In volumeMounts, **mountPath** is a path inside the container that is mounted. Here, mountPath cannot be changed as this structure is predefined in a Docker container. **subPath** works as a relative path that is appended to the attached persistent volume's shareName. **subPathExpr** is used to segregate the product logs container wise. In addition, the **name** is a user-defined name that must be matched with the name specified in volumes.

```
volumes:
- name: azurefile-volume-mount
  persistentVolumeClaim:
    claimName: azurefile-pvc
```

Figure 3.7

In volumes, azurefile-pvc is the persistent volume claim name.

- **Ports:** In the *OmniDocsWeb.yml* file, containerPort is specified as **8080**. That means only port 8080 is exposed outside the container and no other port is accessible from outside.

```
ports:
 - name: http
   containerPort: 8080
```

Figure 3.8

- **ReadinessProbe:** The kubelet uses the readiness probe to know when a container is ready to start accepting traffic. Until unless the readiness probe is not succeeded, the container does not serve the user requests.

```
readinessProbe:
  httpGet:
    path: /omnidocs/web
    port: 8080
  # Intial delay is set to a high value to have a better
  # visibility of the ramped deployment
  initialDelaySeconds: 30
  periodSeconds: 5
```

Figure 3.9

Here, until unless *ip:port/omnidocs/web* is not accessible through a browser, the container does not accept the user request.

- **LivenessProbe**: Docker containers have healing power, if an application running inside the container gets down due to any reason or becomes unresponsive then Kubernetes restarts the application automatically inside the container. This feature is known as **LivenessProbe** in Kubernetes.
  For Example,

```
livenessProbe:
  httpGet:
    path: /omnidocs/web
    port: 8080
  initialDelaySeconds: 300
  failureThreshold: 5
  periodSeconds: 10
```

Figure 3.10

- **Environment variable:** In the *OmniDocsWeb.yml* file, the **JAVA_OPTS** parameter is defined that is used to set the heap size in the WEB container dynamically.

```
- name: JAVA_OPTS
  value: "-XX:+UseContainerSupport -XX:+DisableExplicitGC -XX:InitialRAMPercentage=50.0
  -XX:MaxRAMPercentage=50.0"
```

Figure 3.11

Ensure '**-XX:MaxRAMPercentage**' is a parameter through which you can provide the available memory to use as a max heap size to JVM. In the above example, 75% of total memory is allocated as heap size.

- **ImagePullSecret:** ImagePullSecret is a secret value that is used to pull an image from a private container repository like Azure **Container** Registry.
  For example,

```
imagePullSecrets:
  - name: azurepullsecret
```

Figure 3.12

Execute the below command to create an ImagePullSecret:

```
kubectl create secret docker-registry azurepullsecret --docker-server
newgencontainerregistry.azurecr.io --docker-username= newgencontainerregistry
--docker-password kmPF/ytffu5q6NazqvVYtJ???????
```

You can also create ImagePullSecret from Azure DevOps Release Pipeline.

NOTE:

You can use the above guidelines to update other YAML Files that are as follows:

- OmniDocsWeb_Services.yml
- OmniDocsEJB.yml
- OmniDocsServices.yml
- EasySearch_ApacheOnly.yml
- TEM.yml
- OmniScanWeb7.0.yml
- OmniDocswopi.yml

3.4 Changes in application gateway Ingress YAML files

Along with the product's YAML file, AppGateway Ingress Controller's YAML file **AppGateway-IngressController.yaml** is also shared. Using an ingress controller and ingress rules, a single IP address can be used to route traffic to multiple services in a Kubernetes cluster. The AppGateway Ingress Controller creates a Load Balancer with its external IP and routes the incoming requests to the target Kubernetes services according to the host-based routing rules. Host-based routing is a capability of Ingress Controller that redirects the user requests to the right service based on the request-host header.

For example, you can set the rules as below:

- If URL is *omnidocs.newgendocker.com,* then redirect to the OmniDocsWeb container.
- If URL is *omniscan.newgendocker.com,* then redirect to the OmniScanWeb container.

NOTE:
To support the host-based routing, register a domain and create a new RecordSet in DNS Zone for each host-path. Refer to the document Configuration of Azure Kubernetes Cluster to see the configuration of Application Gateway Ingress Controller and DNS Zone.

- Once Application Gateway Ingress is configured and RecordSets are created in DNS Zone, then must deploy the Ingress controller along with its ruleset using the YAML file.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: appgw-ingress
  namespace: dev
  annotations:
    kubernetes.io/ingress.class: azure/application-gateway
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
    appgw.ingress.kubernetes.io/request-timeout: "300"
    appgw.ingress.kubernetes.io/ssl-policy: "AppGwSSlPolicy20170401S"
spec:
  tls:
    - hosts:
      - newgendocker.com
    - secretName: appgw-cert
  rules:
  - host: omnidocs.newgendocker.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: odll0web
            port:
              number: 8080
  - host: omnidocswebservices.newgendocker.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: odll0websvc
            port:
              number: 8080
  - host: apachemanifold.newgendocker.com
```

Figure 3.13

- In *AppGateway-IngressController.yml* file, there are multiple host-based rules defined.
  - **omnidocs.newgendocker.com [Specified as a record set in Route-53]**
    If the host URL is *omnidocs.newgendocker.com*, then it redirects the user request to the **od110web** container's service which is running on port 8080. Here, od110web is the name of the OmniDocsWeb container.
  - **omnidocswebservices.newgendocker.com [Specified as a record set in Route-53]**
    If the host URL is *omnidocswebservices.newgendocker.com,* then it redirects the user request to the **od110websvc** container's service which is running on port 8080. Here, od110websvc is the name of the OmniDocs Web Service container.
  - **omnidocsconsole.newgendocker.com [Specified as a record set in Route-53]**
    If the host URL is *omnidocsconsole.newgendocker.com,* then it redirects the user request to the **od110ejb** container's service which is running on port 9990. Here, od110ejb is the name of the OmniDocsEJB container.
  - **apachemanifold.newgendocker.com [Specified as a record set in Route-53]**
    If the host URL is *apachemanifold.newgendocker.com* then it redirects the user request to the easysearch11 container's service which is running on port 8345. Here, easysearch11 is the name of the EasySearch container.
  - **omniscan.newgendocker.com [Specified as a record set in Route-53]**
    If the host URL is *omniscan.newgendocker.com,* then it redirects the user request to the **omniscan web** container's service which is running on port 8080. Here, omniscanweb is the name of the OmniScan Web container.
- In this YAML file, change the host URL, ServiceName, ServicePort, and the name **name: appgw-ingress** as required.
- In this YAML file, there is defined SSL or TLS configuration through specifying the **tls** spec along with hosts and secretName.

```
spec:
  tls:
    - hosts:
      - bpmsoncloud.co.in
    - secretName: appgw-cert
```

Figure 3.14

- You can specify the valid DNS against hosts that is*, newgendocker.com.*
- Before deploying the ingress controller, create a Kubernetes secret to host the certificate and private key. Execute the below command to create a Kubernetes secret:

```
kubectl create secret tls <secret-name> --key <path-to-key> --cert <path-
to-crt> -n <Namespace>
```

For example,

```
kubectl create secret tls appgw-cert --key azure.key --cert azure.crt -n
dev
```

- After making the required changes as required, deploy the Ingress controller by executing this YAML file using below command or can configure it to AzureDevOps Release Pipeline.

```
kubectl apply -f AppGateway-IngressController.yml
```

NOTE:
To execute the above command, kubectl must be configured on your local server. Refer to the Configuration of Azure Kubernetes Cluster to run kubectl from your local machine.

3.5 Changes in configuration files

This section describes the changes in configuration files.

3.5.1 Prerequisites

The Prerequisites are as follows:

- All the configuration files and folders must be uploaded to the Azure FileShare defined in the YAML file *AzureFile_PV_PVC.yml*. You can upload the configuration files and folder using Azure Storage Explorer.
- The Redis Cache server is already configured.
- A valid wildcard certificate and the domain are already configured.
- SSL or TLS must be configured at the Ingress Controller or Load balancer level.

NOTE:
By default, all Docker containers are running with HTTPS protocol only. If you want to run with HTTP protocol, then some additional settings must be required. For more details, refer to the *Docker Troubleshooting Guide*.

3.5.2 OmniDocsWeb changes

The changes in OmniDocsWeb are as follows:

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *NGOClientData.xml* file in between the *<endPointURL></endPointURL>* tags located inside the *OmniDocsWeb\Newgen\NGConfig\ngdbini* folder at the mapped location on the Worker node.

```
<ClientInfo>
    <ProviderUrl></ProviderUrl>
    <jndiServerName></jndiServerName>
    <jndiServerPort></jndiServerPort>
    <ContextSuffix></ContextSuffix>
    <WildFlyUserName></WildFlyUserName>
    <WildFlyPassword></WildFlyPassword>
    <JndiContextFactory></JndiContextFactory>
    <ClientLookUpName>ejb:omnidocs_ejb/omnidocs_ejb/NGOClientServiceHandlerBean!com.newgen.omni.jts.txn.NGOClientServiceHandlerHome</ClientLookUpName>
    <AdminLookUpName>ejb:omnidocs_ejb/omnidocs_ejb/NGOAdminServiceHandlerBean!com.newgen.omni.jts.txn.NGOAdminServiceHandlerHome</AdminLookUpName>
    <UrlPackagePrefix></UrlPackagePrefix>
    <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
    <xmlParamName>InputXML</xmlParamName>
    <HeaderKey></HeaderKey>
    <HeaderValue></HeaderValue>
</ClientInfo>
```

Figure 3.15

Here, **od110ejb** is the name of the OmniDocsEJB container.

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *IS.ini* file in between the *<endPointURL ></endPointURL >* tags located inside the *OmniDocsWeb\Newgen\NGConfig* folder at the mapped location on the Worker node.

  For example,

```
<LogPath>Replication.log</LogPath>
<SMSTimeOut>60000</SMSTimeOut>
<SMSReadInterval>30000</SMSReadInterval>
<SMSRetryCount>5</SMSRetryCount>
<SMSGenerateLog>true</SMSGenerateLog>
<IsJNDI>true</IsJNDI>
<endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
<xmlParamName>InputXML</xmlParamName>
<HeaderKey></HeaderKey>
<HeaderValue></HeaderValue>
<ProviderUrl></ProviderUrl>
<jndiServerName></jndiServerName>
<jndiServerPort></jndiServerPort>
```

Figure 3.16

- Update the OmniDocsEJB container name [Defined in OmniDocsEJB.yml file] in *jboss-ejb-client.properties* file located inside the OmniDocsWeb folder kept inside the Azure Fileshare. For example,

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=od110ejb
remote.connection.default.port = 8080
remote.connection.default.username=[redacted]
remote.connection.default.password=[redacted]
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

Figure 3.17

Here, **od110ejb** is the name of the OmniDocsEJB container.

- Update the Azure Redis cache's configuration endpoint in *redisson.yaml* file against the *singleServerConfig* or *clusterServersConfig*. If redis cache is SSL enabled then use *rediss://<endpoint url>:port* and if SSL is not enabled then use *redis://<endpoint url>:port*. This file *redisson.yaml* is located inside the OmniDocsWeb folder kept inside the Azure Fileshare.

```
---
singleServerConfig:
  idleConnectionTimeout: 10000
  connectTimeout: 10000
  timeout: 3000
  retryAttempts: 3
  retryInterval: 1500
  password: "A+[redacted]4M="
  subscriptionsPerConnection: 5
  clientName: null
  address: "rediss://content[redacted]net:6380"
  subscriptionConnectionMinimumIdleSize: 1
  subscriptionConnectionPoolSize: 50
  connectionMinimumIdleSize: 24
  connectionPoolSize: 64
  database: 0
  dnsMonitoringInterval: 5000
threads: 16
nettyThreads: 32
codec: !<org.redisson.codec.MarshallingCodec> {}
transportMode: "NIO"

#Reference: https://github.com/redisson/redisson/wiki/2.-Configuration#26-single-instance-mode

#CLUSTER ---
#CLUSTER clusterServersConfig:
#CLUSTER   idleConnectionTimeout: 10000
#CLUSTER   connectTimeout: 10000
#CLUSTER   timeout: 3000
#CLUSTER   retryAttempts: 3
#CLUSTER   retryInterval: 1500
#CLUSTER   failedSlaveReconnectionInterval: 3000
#CLUSTER   failedSlaveCheckInterval: 60000
#CLUSTER   password: null
#CLUSTER   subscriptionsPerConnection: 5
#CLUSTER   clientName: null
#CLUSTER   loadBalancer: !<org.redisson.connection.balancer.RoundRobinLoadBalancer> {}
#CLUSTER   subscriptionConnectionMinimumIdleSize: 1
```

Figure 3.18

- Open the *web.xml* file in edit mode located inside the OmniDocsWeb folder kept inside the Azure Fileshare.

- Search for filter **httpHeaderSecurity** and update the *<param-value></param-value>* tag's value with OmniDocs URL without context name against <param-name> *antiClickJackingUri</param-name>.*

```
<filter>
    <filter-name>httpHeaderSecurity</filter-name>
    <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
    <async-supported>true</async-supported>
    <init-param>
        <param-name>antiClickJackingOption</param-name>
        <param-value>ALLOW-FROM</param-value>
    </init-param>
    <init-param>
            <param-name>antiClickJackingUri</param-name>
            <param-value>omnidocs.newgendocker.com</param-value>
    </init-param>
</filter>
```

Figure 3.19

- Search for filter-class *<filter-class>org.apache.catalina.filters.CorsFilter</filter-class>* and update the *<param-value></param-value>* tag's value with OmniDocs URL with protocol against *<param-name> antiClickJackingUri</param-name>*.

```
<filter>
    <filter-name>CorsFilter</filter-name>
    <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
    <init-param>
        <param-name>cors.allowed.origins</param-name>
        <param-value>https://omnidocs.newgendocker.com,https://88xggq.sharepoint.com</param-value>
    </init-param>
    <init-param>
        <param-name>cors.allowed.methods</param-name>
        <param-value>GET,POST,HEAD,OPTIONS,PUT</param-value>
    </init-param>
    <init-param>
        <param-name>cors.allowed.headers</param-name>
        <param-value>
        Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-Method,Access-Control-Request-Headers,Access-Control
        -Allow-Origin</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CorsFilter</filter-name>
    <url-pattern>/osaweb/*</url-pattern>
</filter-mapping>
```

Figure 3.20

- Open the web_svc.xml file in edit mode located inside the OmniDocsWeb folder at the mapped location on the Worker node.
- Search for filter-class "<filter-class>org.apache.catalina.filters.CorsFilter</filter-class>" and update the <param-value></param-value> tag's value with OmniDocs URL with protocol against <param-name> antiClickJackingUri</param-name>.

```
<filter>
    <filter-name>CorsFilter</filter-name>
    <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
    <init-param>
        <param-name>cors.allowed.origins</param-name>
        <param-value>https://omnidocs.bpmsoncloud.co.in</param-value>
    </init-param>
            <init-param>
                    <param-name>cors.allowed.methods</param-name>
                    <param-value>GET,POST,HEAD,OPTIONS,PUT</param-value>
            </init-param>
            <init-param>
                    <param-name>cors.allowed.headers</param-name>
                    <param-value>
                    Content-Type,X-Requested-With,accept,Origin,Access-Control-Request
                    </param-value>
            </init-param>
            <init-param>
                    <param-name>cors.exposed.headers</param-name>
                    <param-value>Access-Control-Allow-Origin</param-value>
            </init-param>
</filter>
```

Figure 3.21

3.5.3 Wrapper changes

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in NGOClientData.xml in between the *<endPointURL></endPointURL>* tags file located inside the *ODServices/Wrapper/ngdbini* folder kept inside the Azure Fileshare.

```
<ClientInfo>
    <ProviderUrl></ProviderUrl>
    <jndiServerName></jndiServerName>
    <jndiServerPort></jndiServerPort>
    <ContextSuffix></ContextSuffix>
    <WildFlyUserName></WildFlyUserName>
    <WildFlyPassword></WildFlyPassword>
    <JndiContextFactory></JndiContextFactory>
    <ClientLookUpName>ejb:omnidocs_ejb/omnidocs_ejb/NGOClientServiceHandlerBean!com.newgen.omni.jts.txn.NGOClientServiceHandlerHome</ClientLookUpName>
    <AdminLookUpName>ejb:omnidocs_ejb/omnidocs_ejb/NGOAdminServiceHandlerBean!com.newgen.omni.jts.txn.NGOAdminServiceHandlerHome</AdminLookUpName>
    <UrlPackagePrefix></UrlPackagePrefix>
    <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
    <xmlParamName>InputXML</xmlParamName>
    <HeaderKey></HeaderKey>
    <HeaderValue></HeaderValue>
</ClientInfo>
```

Figure 3.22

Here, **od110ejb** is the name of the OmniDocsEJB container.

3.5.4 AlarmMailer changes

Prerequisite:

The cabinet is created and associated with the running containers. If the cabinet is not created, then refer to Cabinet and Data Source Creation section.

Make the changes in AlarmMailer that are as follows:

1. Update the OmniDocsEJB container name [Defined in OmniDocsEJB.yml file] in *IS.ini* in between the *<endPointURL></endPointURL>* tags file located inside the *ODServices* or *AlarmMailer* folder kept inside the Azure Fileshare.
   For example,
   <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
   Here, **od110ejb** is the name of the OmniDocsEJB container.
2. Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *NGOClientData.xml* in between the *<endPointURL></endPointURL>* tags file located inside the *ODServices/AlarmMailer/ngdbini* folder kept inside the Azure Fileshare.
   For example,
   <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
   Here, **od110ejb** is the name of the OmniDocsEJB container.
3. Update the below settings in the *Alarm.ini* file located inside the *ODServices/AlarmMailer* folder kept inside the Azure Fileshare.
   i. Update the OmniDocs URL without context name in between the *<webservername></webservername>* tag.
      For example, <webservername>omnidocs.newgendocker.com</webservername>
      Here, *omnidocs.newgendocker.com* is the host path defined in the *AppGateway-IngressController.yml* file.
   ii. Leave the WebServerPort as blank if OmniDocsWEB URL does not contain a port.
      For example, <webserverport></webserverport>
   iii. Update the OmniDocs cabinet name in between **<cabinetname></cabinetname>** tag.
      For example, <cabinetname>ecmsuite</cabinetname>
      Here, **ecmsuite** is the OmniDocs cabinet name gets created.
   iv. Update the OmniDocs supervisor group's user in between the **<user></user>** tag.
      For example, <user>supervisor</user>
   v. Update the OmniDocs supervisor group's user password in between the **<password></password>** tag. Ensure that this password must be in an encrypted format.
      For example, <password>:X-D;U:T-C;P-C;p5-C;b:d:</password>

3.5.5 LDAP changes

Prerequisite:

The cabinet is created and associated with the running containers. If the cabinet is not created, then refer to the Cabinet and Data Source Creation section.

The changes in LDAP are as follows: (For On_Prem Active Directory)

- Ensure that the LDAP Domain server is configured, and a private tunnel is created between the Kubernetes worker nodes and the LDAP Domain server.
- Update the OmniDocsEJB container name [Defined in OmniDocsEJB.yml file] in *NGOClientData.xml* in between the *<endPointURL></endPointURL>* tags file located inside the *ODServices/ODAuthMgr/ngdbini* folder kept inside the Azure Fileshare.
  For example,
  <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
  Here, **od110ejb** is the name of the OmniDocsEJB container.
- Update the cabinet name and domain name in the ldap.ini and Hook.ini file located inside the *ODServices/ODAuthMgr* folder at the mapped location.

```
#
#Tue Nov 26 11:34:40 IST 2013
DISPort=1999
DISIPAddress=127.0.0.1
Log4j_properties_file=jtshook_log4j.properties
Encoding=UTF-8
PROTOCOL=ldap
LOGOUTTIME=15000
DIRECTORYSERVICE=ActiveDS
REACTUI=true

# Default domain name to add user For multidomain LDAP
DEFAULTDOMAIN=eco.com
ecmsuite=eco.com
```

Hook.ini

Figure 3.23

```
#
#Wed Dec 23 10:53:14 GMT+05:30 2009
DISPort=1999
DISIPAddress=127.0.0.1
Encoding=UTF-8
setEncoding=true
LogOutTime=15000
IsMakerChecker=N

# Default domain name to add user For multidomain LDAP
ecmsuite=eco.com
```

LDAP.ini

Figure 3.24

Here, **ecmsuite** is the cabinet name and *eco.com* is the domain name.

- Update the same cabinet name and domain name in the *ldap.ini* and *Hook.ini* file located inside the *OmniDocsWeb\Newgen\NGConfig* folder at the mapped location.

- Update the ODServices container's service name [Defined in respective YAML file] in *ldap.ini* and *Hook.ini* file located inside the *OmniDocsWeb\Newgen\NGConfig* folder at the mapped location.

```
#
#Tue Nov 26 11:34:40 IST 2013
DISPort=1999
DISIPAddress=od110services
Log4j_properties_file=jtshook_log4j.properties
Encoding=UTF-8
PROTOCOL=ldap
LOGOUTTIME=15000
DIRECTORYSERVICE=ActiveDS
REACTUI=true

# Default domain name to add user For multidomain LDAP
DEFAULTDOMAIN=eco.com
ecmsuite=eco.com
```

Hook.ini

Figure 3.25

```
#
#Wed Dec 23 10:53:14 GMT+05:30 2009
DISPort=1999
DISIPAddress=od110services
Encoding=UTF-8
setEncoding=true
LogOutTime=15000
IsMakerChecker=N

# Default domain name to add user For multidomain LDAP
ecmsuite=eco.com
```

ldap.ini

Figure 3.26

Here, **od110services** is the service name of the ODServices container.

- Set **<Display>** as true for LDAP in *AdminMenuOptions.xml* located inside *OmniDocsWeb/Newgen/NGConfig/ngdbini/Custom/CABINETNAME* folder at mapped location.

```
<SSALink>
    <LinkName>Ldap</LinkName>
    <LinkDescription>LdapDescription</LinkDescription>
    <JspName>/ldap/config.jsp</JspName>
    <Display>true</Display>
    <IconURL></IconURL>
</SSALink>
```

Figure 3.27

The changes in LDAP are as follows: (For Azure Active Directory)

- Update the OmniDocsEJB container name [Defined in OmniDocsEJB.yml file] in *NGOClientData.xml* in between the *<endPointURL></endPointURL>* tags file located inside the *ODServices/ODAuthMgr/ngdbini* folder kept inside the Azure Fileshare.
  For example,
  <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
  Here, **od110ejb** is the name of the OmniDocsEJB container.
- Update the cabinet name, domain name, and directory service as **AzureAD** in the Hook.ini file located inside the *ODServices/ODAuthMgr* folder at the mapped location.

```
DISPort=1999
DISIPAddress=127.0.0.1
Log4j_properties_file=jtshook_log4j.properties
Encoding=UTF-8
PROTOCOL=ldap
LOGOUTTIME=15000
DIRECTORYSERVICE=AzureAD
REACTUI=true

# Default domain name to add user For multidomain LDAP
DEFAULTDOMAIN=eco.com
ecmsuite=eco.com
```

Hook.ini

Figure 3.28

- Update the cabinet name and domain name in the *ldap.ini* file located inside the *ODServices* or *ODAuthMgr* folder at the mapped location.

```
#
#Wed Dec 23 10:53:14 GMT+05:30 2009
DISPort=1999
DISIPAddress=127.0.0.1
Encoding=UTF-8
setEncoding=true
LogOutTime=15000
IsMakerChecker=N

# Default domain name to add user For multidomain LDAP
ecmsuite=eco.com
```

LDAP.ini

Figure 3.29

Here, **ecmsuite** is the cabinet name and *eco.com* is the domain name.

- Update the directory service as **AzureAD** in the DIS.xml file located inside the **ODServices** or **ODAuthMgr** folder at the mapped location.

```
<XML>
<Title>Cabinet Mapping Information</Title>
<Body>
<Debug>
<Log4jPropPath>dissynch_log4j.properties</Log4jPropPath>
<SynchLog>true</SynchLog>
<DISLog>false</DISLog>
<changePass>false</changePass>
<Language>english</Language>
<Encoding>UTF-8</Encoding>
<JTSIP>127.0.0.1</JTSIP>
<JTSPort>3333</JTSPort>
</Debug>
<RootDirectoryInformation>
<DirectoryService>AzureAD</DirectoryService>
</RootDirectoryInformation>
<NoOfCabinets>0</NoOfCabinets>
<doSynch>true</doSynch>
<GroupSynch>true</GroupSynch>
<ExistingUserAsNonDomain>false</ExistingUserAsNonDomain>
<NonDomainSupport>true</NonDomainSupport>
<InactivateDeleteUser>true</InactivateDeleteUser>
<Protocol>ldap</Protocol>
<DISPort>1999</DISPort>
<CabinetList>
</CabinetList>
</Body>
</XML>
```

Figure 3.30

- Update the same cabinet name and domain name in the *ldap.ini* and *Hook.ini* file located inside the **OmniDocsWeb\Newgen\NGConfig** folder at the mapped location.
- Update the ODServices container's service name [Defined in respective YAML file] in ldap.ini and Hook.ini file located inside the **OmniDocsWeb\Newgen\NGConfig** folder at the mapped location.
- Update the directory service as **AzureAD** in Hook.ini and config.ini located inside the **OmniDocsWeb\Newgen\NGConfig** folder at the mapped location.

```
DISPort=1999
DISIPAddress=od110services
Log4j_properties_file=jtshook_log4j.properties
Encoding=UTF-8
PROTOCOL=ldap
LOGOUTTIME=15000
DIRECTORYSERVICE=AzureAD
REACTUI=true

# Default domain name to add user For multidomain LDAP
DEFAULTDOMAIN=eco.com
ecmsuite=eco.com
```

Hook.ini

Figure 3.31

```
#
#Wed Dec 23 10:53:14 GMT+05:30 2009
DISPort=1999
DISIPAddress=od110services
Encoding=UTF-8
setEncoding=true
LogOutTime=15000
IsMakerChecker=N

# Default domain name to add user For multidomain LDAP
ecmsuite=eco.com
```

ldap.ini

Figure 3.32

```
DIRECTORYSERVICE=AzureAD
REACTUI=true
```

config.ini

Figure 3.33

Here, **od110services** is the service name of the ODServices container.

- Set **<Display>** as true for ldap in *AdminMenuOptions.xml* located inside *OmniDocsWeb/Newgen/NGConfig/ngdbini/Custom/CABINETNAME* folder at mapped location. For example,

```
<SSALink>
    <LinkName>Ldap</LinkName>
    <LinkDescription>LdapDescription</LinkDescription>
    <JspName>/ldap/config.jsp</JspName>
    <Display>true</Display>
    <IconURL></IconURL>
</SSALink>
```

Figure 3.34

3.5.6 SSO changes

Prerequisite:
The cabinet is created and associated with the running containers. If the cabinet is not created, then refer to Cabinet and Data Source Creation section.

The changes in SSO are as follows:

- Update the *<Host-Path URL of OmniDocsWeb container>* at the place of *ibps5aurora.newgendocker.com* in *mapping.xml* file located inside the *OmniDocsWeb/Newgen/NGConfig/ngdbini/SSOConFig* folder.
- Update the **CabinetName** in *mapping.xml* file located inside the *OmniDocsWeb/Newgen/NGConfig/ngdbini/SSOConFig* folder.
- Configure the CabinetName=DomainName in *sso.ini* file located inside the *OmniDocsWeb/Newgen/NGConfig/ngdbini/SSOConFig* folder.
- *ecmsuite=eco.com*

3.5.7 Scheduler changes

Prerequisite:
The cabinet is created and associated with the running containers. If the cabinet is not created, then refer to Cabinet and Data Source Creation section.

The changes in Scheduler are as follows:

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *IS.ini* in between the *<endPointURL></endPointURL>* tags file located inside the **ODServices** or **Scheduler** folder kept inside the Azure Fileshare.
  For example,
  <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
  Here, **od110ejb** is the name of the OmniDocsEJB container.
- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *NGOClientData.xml* in between the *<endPointURL></endPointURL>* tags file located inside the *ODServices/Scheduler/ngdbini* folder kept inside the Azure Fileshare.
  For example,
  <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
  Here, **od110ejb** is the name of the OmniDocsEJB container.
- Update the ODServices container's service name [Defined in respective YAML file] in *SchedulerConf.ini* file located at **ODServices** or **Scheduler** folder at the mapped location.
  For example: **schedulerIpAddress=od110services**
- Update the ODServices container's service name [Defined in respective YAML file] in eworkstyle.ini file located at

OmniDocsWeb/Newgen/NGConfig/ngdbini/Custom/<CABINETNAME> folder at mapped location.
For example: **schedularLocation=od110services**

3.5.8 ThumbnailManager changes

Prerequisite:
The cabinet is created and associated with the running containers. If the cabinet is not created, then refer to Cabinet and Data Source Creation section.

The changes in ThumbnailManager are as follows:

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *IS.ini* in between the *<endPointURL></endPointURL>* tags file located inside the **ODServices** or **ThumbnailManager** folder kept inside the Azure Fileshare.
  For example,
  <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
  Here, **od110ejb** is the name of the OmniDocsEJB container.
- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *NGOClientData.xml* in between the *<endPointURL></endPointURL>* tags file located inside the *ODServices/ThumbnailManager/ngdbini* folder kept inside the Azure Fileshare.
  For example,
  <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
  Here, **od110ejb** is the name of the OmniDocsEJB container.
- Update the cabinet name, supervisor group's user name, and password in *ThumnailConfig.xml* located inside the **ODServices** or **ThumbnailManager** folder at the mapped location on the Worker node.

```
<cabinets><cabinet><cabinetname>ecmsuite</cabinetname><jtsip>127.0.0.1
</jtsip><jtsport>3333</jtsport><user>supervisor</user><password>:X-D;U:T-C;P-C;p5-C;b:
</password><BatchSize>10</BatchSize><priority>1</priority><encoding>UTF-8
</encoding></cabinet></cabinets>
```

Figure 3.35

3.5.9 TEM changes

Prerequisite:

The cabinet is created and associated with the running containers. If the cabinet is not created, then refer to Cabinet and Data Source Creation section.

The changes in TEM are as follows:

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *IS.ini and NGOClientData.xml* in between the *<endPointURL></endPointURL>* tags file located inside the **TEM** folder kept inside the Azure Fileshare.
  For example,
  <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
- Here, **od110ejb** is the name of the OmniDocsEJB container.
- Update the cabinet name in filename FTSServer-CABINETNAME-1.properties.
  For example: FTSServer-**ecmsuite**-1.properties [ecmsuite is the cabinet name].
- Update the OmniDocsEJB container name [Defined in OmniDocsEJB.yml file] in FTSServer-**ecmsuite**-1.properties renamed earlier.
- Update the OmniDocs supervisor group's user name.
- Update the OmniDocs supervisor group's user password. Ensure this password must be in an encrypted format.

```
ServerAddress=od110ejb
SiteId=1
UserName=supervisor
Password=:X-D;U:T-C;P-C;p5-C;b:d:
PollTime=10
OCRPath=tesseract
DocumentCount=1000
Language=eng
SleepTime=15
```

Figure 3.36

3.5.10 EasySearch changes

Prerequisite:

The cabinet is created and associated with the running containers. If the cabinet is not created, then refer to Cabinet and Data Source Creation section.

The changes in EasySearch (Apache Manifold only) are as follows:

- Update the Database details in the *ESconfig.ini* file located inside the *EasySearch\ESConfigurator\conf* folder kept inside the Azure Fileshare.
  - ESClusterName=CABINETNAME_cluster
  - OdDBIPAddress=DBIP
  - OdDBPort=DBPORT
  - OdCabinetName=CABINETNAME
  - OdDBUserName=DBUSER
  - OdDBPassword=DBPASSWORD in encrypted format
  - OdDBType=sqlserver | oracle | postgres

```
ESServerTCPPort=9300
ESServerHttpPort=9200
ESProtocol=http
ESClusterName=ecmsuite_cluster
OdDBIPAddress=omnidocs-aurorards-db-instance-1-restore
OdDBPort=5432
OdCabinetName=ecmsuite
OdDBUserName=postgres
OdDBPassword=:X-D;Y-D;L-C;N-C;VSJ-C;4T-C;r
OdDBType=postgres
MCFIPAddress=127.0.0.1
```

Figure 3.37

- Update **AppToBeConfigured=ApacheManifold** in the *ESconfig.ini* file located inside the *EasySearch\ESConfigurator\conf* folder kept inside the Azure Fileshare
- Update the cabinet name in the **CrawlerConfig.xml** file located inside the **EasySearch\apache-manifoldcf-2.25\example** folder kept inside the Azure Fileshare.
- Update the OmniDocs supervisor group's user name.
- Update the OmniDocs supervisor group's user password. Ensure this password must be in an encrypted format.

```
<cabinets>
<jtsip>127.0.0.1</jtsip>
<jtsport>3333</jtsport>
<StopPhraseFlag>N</StopPhraseFlag>
<StopPhrases>
<StopPhrase>Newgen Software Technologies</StopPhrase>
<StopPhrase>omnidocs</StopPhrase>
</StopPhrases>
<Pages>ALL</Pages>
<KeyVault>false</KeyVault>
<isAuthEnabled>N</isAuthEnabled>
<cabinet>
<cabinetname>odazure24nov</cabinetname>
<user>supervisor</user>
<password>:X-D;Y-D;L-C;N-C;VSJ-C;4T-C;r</password>
<ESEnabled>N</ESEnabled>
</cabinet>
</cabinets>
```

Figure 3.38

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in ***NGOClientData.xml*** *and* ***RMClientData.xml*** in between the *<endPointURL></endPointURL>* tags file located inside the EasySearch/apache-manifoldcf-2.25/example/Newgen/NGConfig/ngdbini folder kept inside the Azure Fileshare.
- Update the EnableEasySearch=Y in the *eworkstyle.ini* file located inside the *OmniDocsWeb\Newgen\NGConfig\ngdbini\Custom\CABINET_NAME* folder kept inside the Azure Fileshare.

```
#For EasySearch
EnableEasySearch=Y
EnableEasySearchIndexDropDown=N
EasySearchIPAddress=127.0.0.1
EasySearchHttpPort=9200
```

Figure 3.39

3.5.11 WOPI changes

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *NGOClientData.xml* file in between the *<endPointURL></endPointURL>* tags located inside the *OmniDocs_WOPI\Newgen\NGConfig\ngdbini* folder at the mapped location on the Worker node.

```
<?xml version="1.0" ?>
<ClientInfo>
    <ProviderUrl></ProviderUrl>
    <jndiServerName></jndiServerName>
    <jndiServerPort></jndiServerPort>
    <ContextSuffix></ContextSuffix>
    <WildFlyUserName></WildFlyUserName>
    <WildFlyPassword></WildFlyPassword>
    <JndiContextFactory></JndiContextFactory>
    <ClientLookUpName>ejb:omnidocs_ejb/omnidocs_ejb/NGOClientServiceHandlerBean!com.newgen.omni.jts.txn.NGOClientServiceHandlerHome</ClientLookUpName>
    <AdminLookUpName>ejb:omnidocs_ejb/omnidocs_ejb/NGOAdminServiceHandlerBean!com.newgen.omni.jts.txn.NGOAdminServiceHandlerHome</AdminLookUpName>
    <UrlPackagePrefix></UrlPackagePrefix>
    <endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
    <xmlParamName>InputXML</xmlParamName>
    <HeaderKey></HeaderKey>
    <HeaderValue></HeaderValue>
</ClientInfo>
```

Figure 3.40

Here, **od110ejb** is the name of the OmniDocsEJB container.

- Update the OmniDocsEJB container name [Defined in *OmniDocsEJB.yml* file] in *IS.ini* file in between the *<endPointURL ></endPointURL >* tags located inside the *OmniDocs_WOPI\Newgen\NGConfig* folder at the mapped location on the Worker node. For example,

```
<LogPath>Replication.log</LogPath>
<SMSTimeOut>60000</SMSTimeOut>
<SMSReadInterval>30000</SMSReadInterval>
<SMSRetryCount>5</SMSRetryCount>
<SMSGenerateLog>true</SMSGenerateLog>
<IsJNDI>true</IsJNDI>
<endPointURL>http://od110ejb:8080/callbroker/execute/GenericCallBroker</endPointURL>
<xmlParamName>InputXML</xmlParamName>
<HeaderKey></HeaderKey>
<HeaderValue></HeaderValue>
<ProviderUrl></ProviderUrl>
<jndiServerName></jndiServerName>
<jndiServerPort></jndiServerPort>
```

Figure 3.41

- Update the *WOPI_SOURCE*, *OMNIDOCS_REDIRECTURL* and *CABINETNAME* in *WOPIConfiguration.ini* file located inside the *OmniDocs_WOPI\Newgen\NGConfig\AddInsConfig* folder at the mapped location on the Worker node.

```
WOPIPrivateSecEnq=77+9PjBHaDZ3bgLvv70pJe+/vSoT77+977+9E13vv73vv71i77+977+9dO+/vce8

WOPI_SOURCE=https://wopi.newgendocker.com

#To enable custom app functionality and breadcrumb URL redirection,should incorporate the app at this location.
OMNIDOCS_REDIRECTURL=https://omnidocs11alpine.newgendocker.com/omnidocs/wopi/redirect.html
#MYAPP_REDIRECTURL=https://wopi.domain.com/Apps/redirect.html

#Cabinet Index If admin wants to configure multiple cabinet then need to add new cabinet with increment index .
#ngofficewopi_INDEX=1 This is example for ngofficewopi cabinetname and index 1
odpostgres15dec_INDEX=1
```

Figure 3.42

Where,
https://wopi.newgendocker.com is host URL of WOPI container.
https://omnidocs11alpine.newgendocker.com is Host URL of Omnidocs WEB container.
odpostgres15dec is cabinet name.

- Open the *web.xml* file in edit mode located inside the *OmniDocs_WOPI* folder at the mapped location on the Worker node.
- Search for filter-class *<filter-class>org.apache.catalina.filters.CorsFilter</filter-class>* and update the *<param-value></param-value>* tag's value with OmniDocs URL against *<param-name> antiClickJackingUri</param-name> and * against <param-name>cors.allowed.origins</param-name>*

```
    <filter>
        <filter-name>httpHeaderSecurity</filter-name>
        <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
        <async-supported>true</async-supported>
        <init-param>
            <param-name>antiClickJackingOption</param-name>
            <param-value>ALLOW-FROM</param-value>
        </init-param>
        <init-param>
                <param-name>antiClickJackingUri</param-name>
                <param-value>omnidocs11alpine2.newgendocker.com</param-value>
        </init-param>
    </filter>

<filter>
    <filter-name>CorsFilter</filter-name>
        <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
    <init-param>
        <param-name>cors.allowed.origins</param-name>
        <param-value>*</param-value>
    </init-param>
    <init-param>
        <param-name>cors.allowed.methods</param-name>
        <param-value>GET,POST,HEAD,OPTIONS,PUT</param-value>
    </init-param>
    <init-param>
        <param-name>cors.allowed.headers</param-name>
        <param-value>Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-Method,Access-Control-Request-Headers,Access-Control-Allow-Origin
        </param-value>
    </init-param>
</filter>
```

Figure 3.43

- Add the *CSPHeaderAllowedDomains* tag in the *eworkstyle.ini* file located inside the *OmniDocsWeb/Newgen/NGConfig/ngdbini/odwebini* folder at the mapped location on the Worker node.

```
CSPHeaderAllowedDomains=default-src * data: 'unsafe-inline' 'unsafe-eval';
```

- Add the WOPIOfficeExtensionSuppport and WOPIOfficeExtensionSuppportURL tag in the *eworkstyle.ini* file located inside the *OmniDocsWeb/Newgen/NGConfig/ngdbini/Custom/CABINET_NAME* folder at the mapped location on the Worker node.
  - WOPIOfficeExtensionSuppport = doc, docx, DOCX, DOC, xls, xlsx, XLSX, XLS, ppt, pptx, PPTX, PPT, wopitest, WOPITEST, wopitestx, and WOPITESTX
  - WOPIOfficeExtensionSuppportURL = https://wopi.newgendocker.com

3.5.12 OmniScanWeb changes

Perform the below steps to register the cabinet in OmniScanWeb:

1. Open the OmniScanWeb using the following URL:
   http://<Host-Path URL of OmniScanWeb container>/omniscanweb
   For example,
   https://omniscan.newgendocker.com/omniscanweb
2. Click **Register New Cabinet** link on the OmniScan Web login screen.

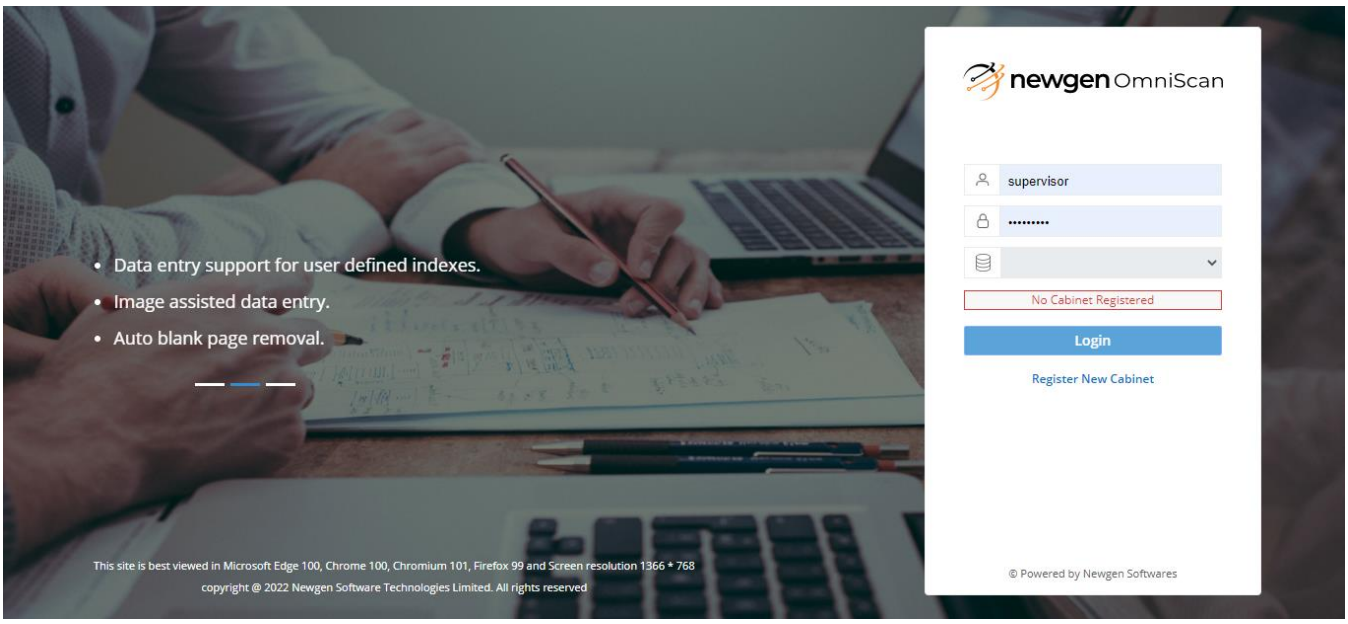

Figure 3.44

3. Specify the Server URL as given below:
   http://<Host-Path URL of OmniDocsWeb container>/NGServlet/servlet/ExternalServlet
   For example,
   https://omnidocs.newgendocker.com/NGServlet/servlet/ExternalServlet

4. Specify the **OmniDocs EJB** container name for AppServer IP or Server URL, 8080 for AppServer Port, and JBOSSEAP for AppServer Type.

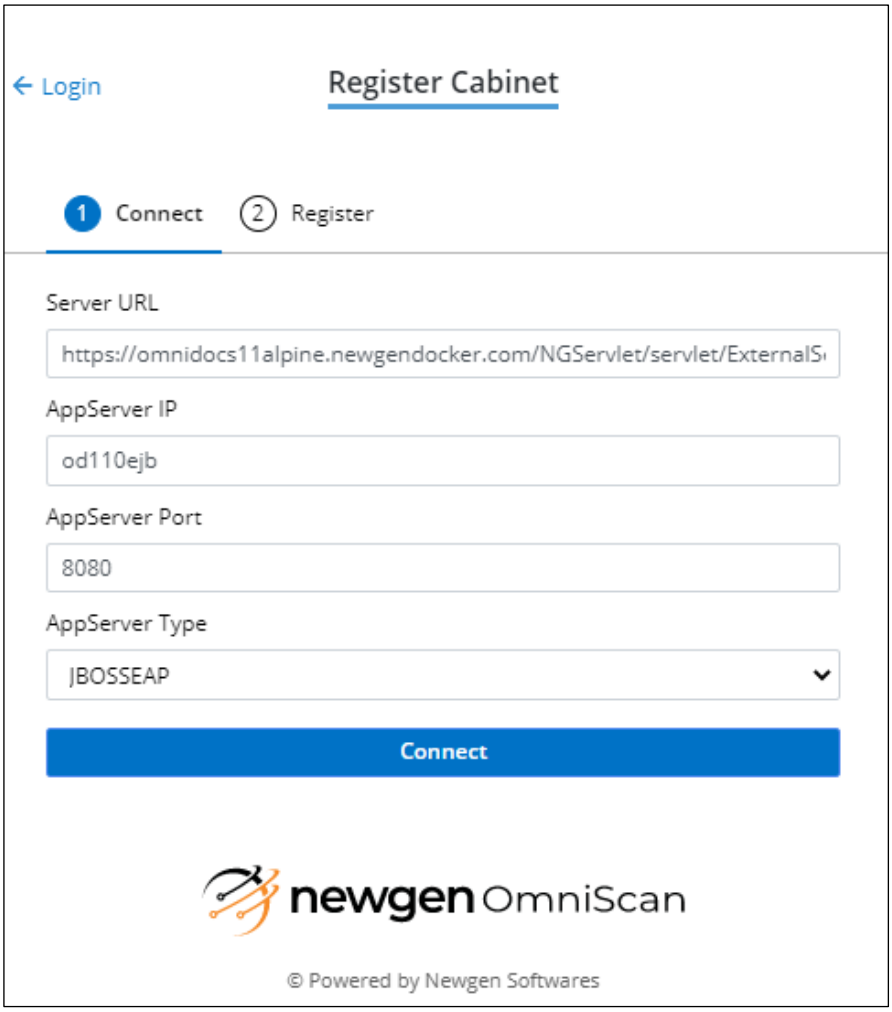

Figure 3.45

5. Click **Connect**.

6. Select the **Cabinet Name**, **Site ID**, and **Volume ID** from the list.

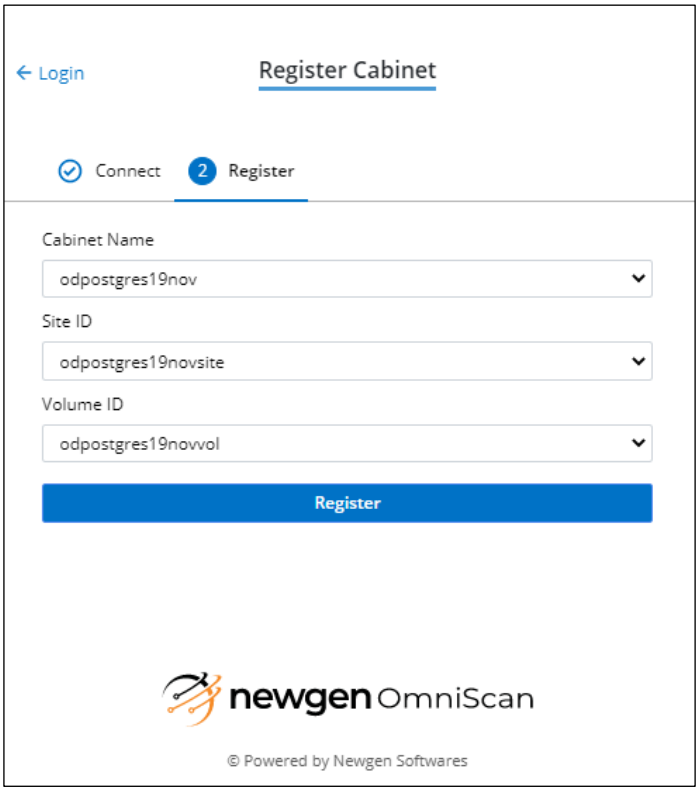

Figure 3.46

7. Click **Register**.

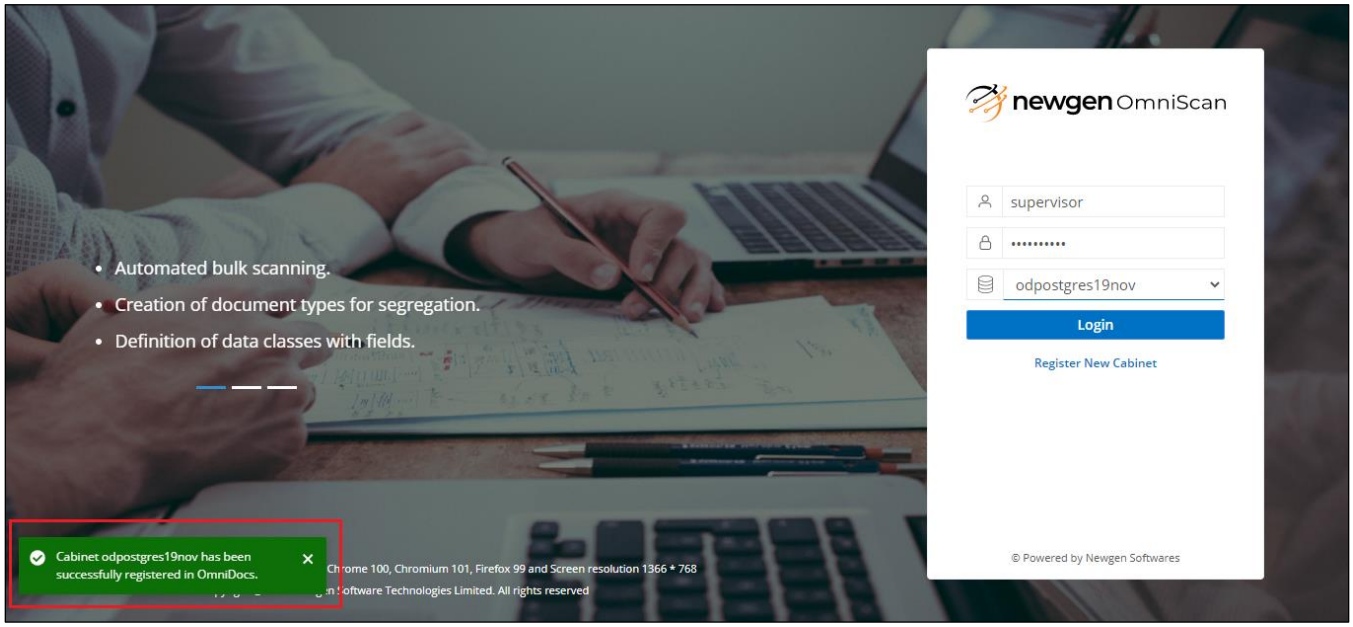

Figure 3.47

The registered cabinet appears in the **Cabinet Name** list on the login screen. Now you can log into OmniScan Web.

NOTE:
Ensure that the **OmniScan_Template_Repository** folder is already created in OmniDocs before logging into OmniScan Web.

3.6 Deployment of containers

Perform the below steps to deploy the containers:

- Deploy the containers on Azure Kubernetes Service from your local machine by executing the below command or you can deploy them using Azure DevOps Release Pipeline. However, it recommends deploying the containers using Azure DevOps for better traceability.

  ```
  kubectl apply -f <YAML_File>
  ```

  For example,

  ```
  kubectl apply -f OmniDocsWeb.yml
  ```

NOTE:

- To execute the above command, kubectl must be configured on your local server. Refer to the Configuration of Azure Kubernetes cluster section to run kubectl from your local machine.
- To deploy the containers using Azure DevOps Release Pipeline, Azure DevOps must be configured. Refer to the Configuration of Azure DevOps Release Pipeline section.

- In Azure DevOps Pipeline, a separate Release pipeline is created for each Docker image like OmniDocsWeb, OmniDocsWebService, OmniDocsEJB, OmniDocsServices, EasySearch, TEM, and OmniScanWeb7.0.
  For Example,

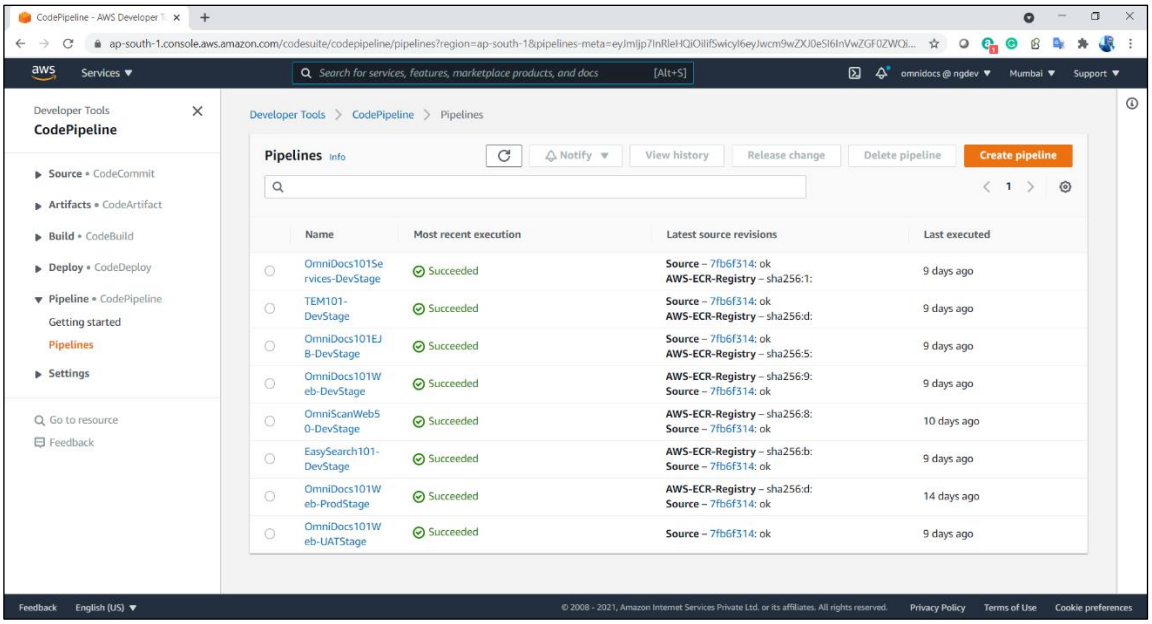

Figure 3.48

- Trigger the Release Pipeline to deploy the required Docker containers.
- Once the deployment is done, deployed containers can be visible from the Kubernetes Dashboard. Refer to the [Configuration of Azure Kubernetes cluster] to configure the Kubernetes Dashboard.

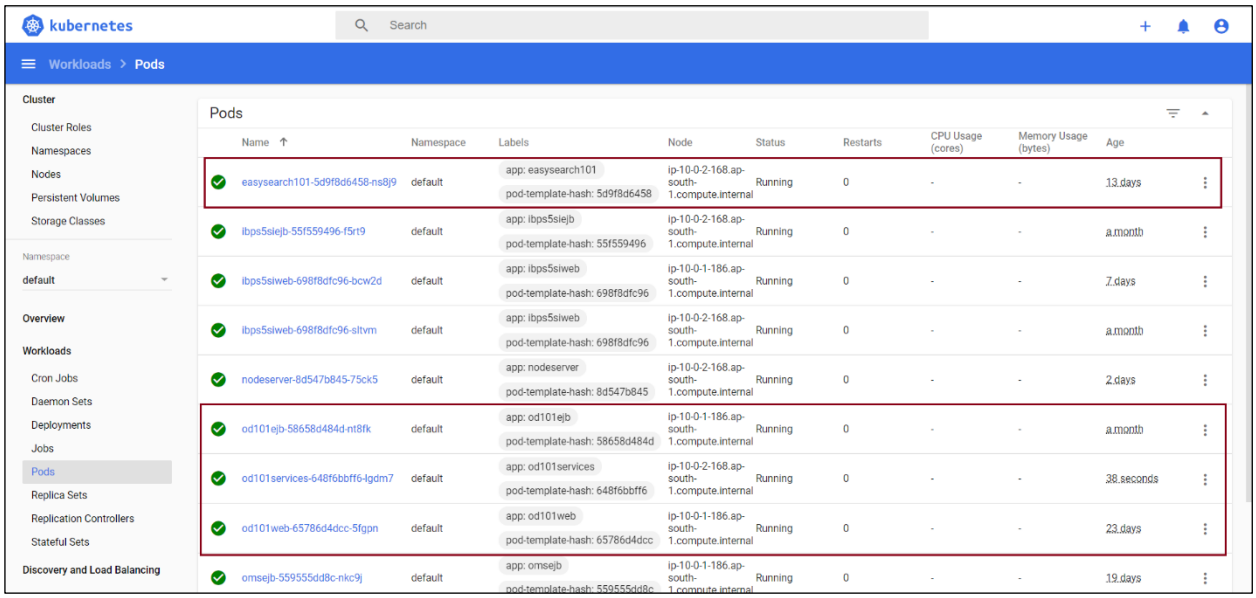

Figure 3.49

- Update the container's replica set from **1** (default value) to any other number in YAML files, then that number of containers is listed in Kubernetes Dashboard.

- In any case to restart the container then there are two options either redeploy the container from Azure DevOps Release Pipeline which launches the new container by following up the rolling update feature of Kubernetes or execute the restart command from Kubernetes' pod's shell.
- The restart command is different for each container.
  For example,

Container Name	Restart Command
OmniDocsWeb, OmniDocsWebService	restartjws.sh
OmniDocsEJB	restartjboss.sh
OmniDocsServices	restartalarm.sh, restartauthmgr.sh, restartscheduler.sh,restartthumbnail.sh, restartwrapper.sh
EasySearch	restartapache.sh
TEM	restarttem.sh
OmniScanWeb7.0	restartjws.sh
OmniDocsWOPI	restartjws.sh

- Once the EasySearch11 container is deployed, execute the below command in Kubernetes pod's shell for the 1st time to configure the Apache Manifold jobs. After that in subsequent deployments, this execution is not required.

```
runESConfigurator.sh
```

3.7 Cabinet and data source creation

Prerequisites:

- OmniDocsWeb, OmniDocsEJB, and OmniDocsServices are already deployed.
- ALB Ingress Controller is already configured and deployed using the *AppGateway-IngressController.yml* file.
- Azure BLOB Storage is already created to store the PN files. PN files are encrypted files that contain all the added, uploaded, and scanned documents by Newgen products.

Once the above prerequisites are fulfilled, refer the below sections to create the Cabinet and Data Source.

- Getting started with OSA
- Register JTS Server
- Connecting OSA to the JTS Server
- Creating a Cabinet
- Associating the Cabinet
- Creating a Data Source

- Registration of the Cabinet
- Creating Site and Volume

3.7.1 Getting started with OSA

Perform the below steps to start the OSA:

1. Since the container is a CLI-based deployment you can't launch any GUI-based application inside the container. But you must use the OSA to create a cabinet that is a GUI-based application. In such a case, deploy OSA to some GUI-based machine either on a local server or on an EC2 instance. Also, add an inbound rule in the Kubernetes worker node's security group to allow OSA to communicate with the OmniDocs Services container deployed on that worker node.
2. Once OSA is deployed on a machine, navigate to the OSA folder on that machine and double click on RunAdmin.bat (For Windows) or RunAdmin.sh (For Linux) to start OSA.
3. When the application is launched. The Login dialog appears.

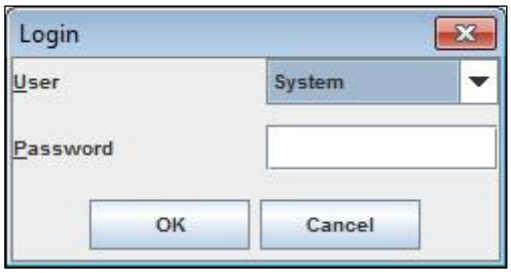

Figure 3.50

4. Select the user as **System** and specify the password as **system**.
5. Click **OK** to log in. After the successful login, the OSA screen appears displaying the list of registered services.

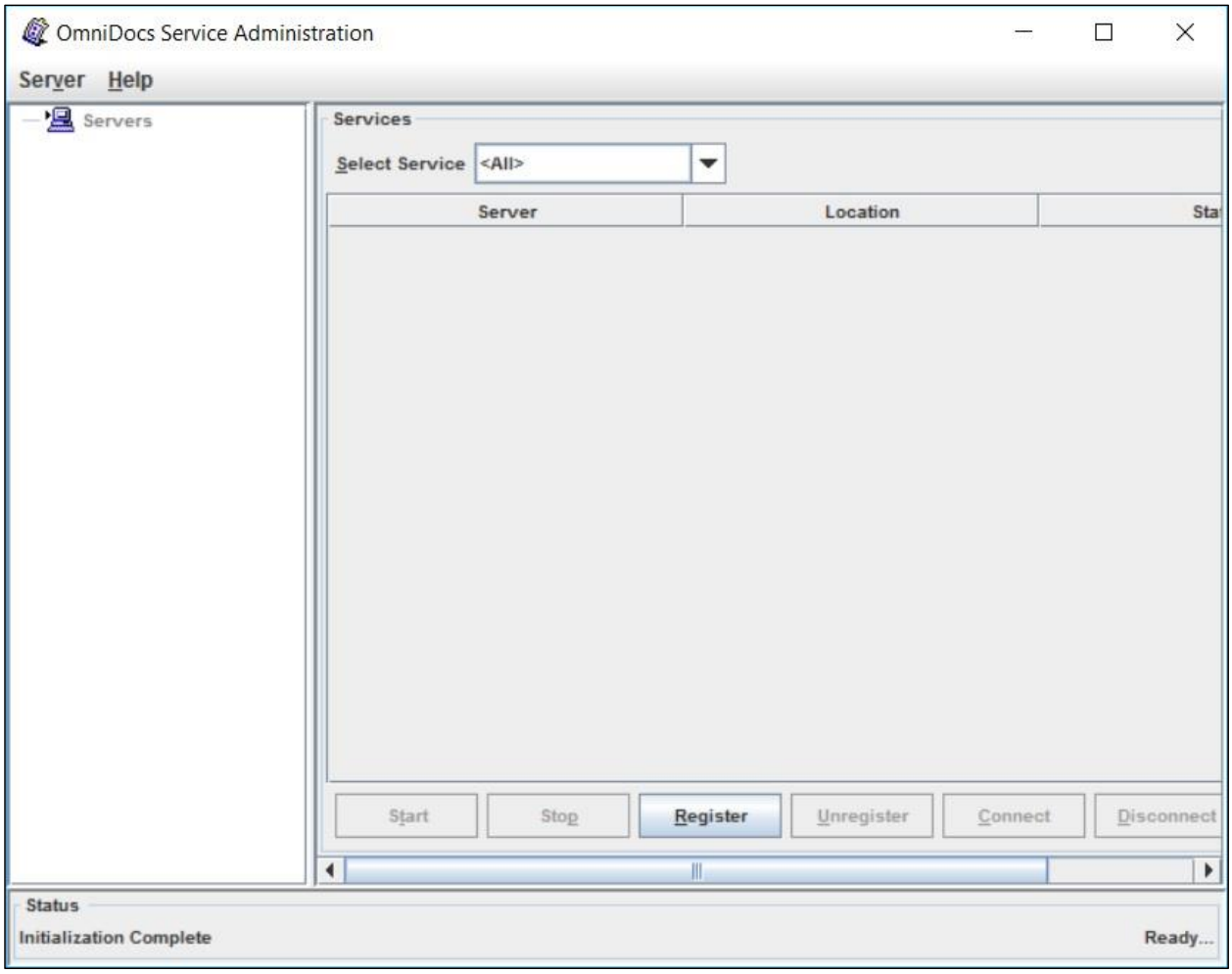
OmniDocs Service Administration
Server
Help
Servers
Services
Select Service
<All>
Server
Location
Start
Stop
Register
Unregister
Connect
Disconnect
Status
Initialization Complete
Ready...

Figure 3.51

3.7.2 Register JTS server

Perform the below steps to register the JTS Server:

1. To register the JTS server, click **Register** button. The **Register New Server** dialog appears.

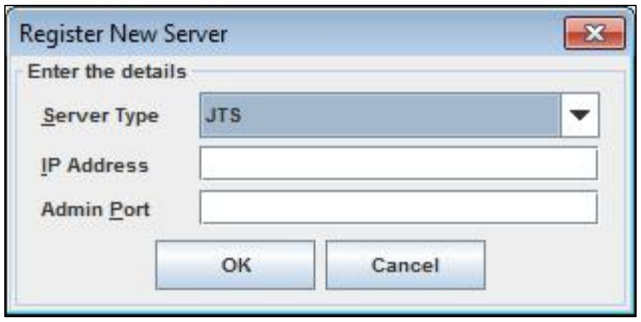

Figure 3.52

2. Select the JTS and specify the public IP address of the Kubernetes Worker node on which the OmniDocsServices (Wrapper, AlarmMailer, THN, and so on) container is deployed.
   For example, suppose there are two worker nodes in the Kubernetes cluster and after deploying the OmniDocsServices container, it gets deployed to the 1st worker node then specify the IP address of the 1st worker node. But in a case, 2 replicas are deployed on the OmniDocsServices container, one on each worker node, in that case, specify the IP address of any worker node.
3. Specify the Admin port of Wrapper service running inside the OmniDocsServices container.
   Since Wrapper is running inside the container with Admin port 9996 but that Admin port cannot be accessed directly. Kubernetes generates a random port (aka NodePort) for each port running inside the container that is exposed outside the container for public use. To get this NodePort either from Kubernetes Dashboard or by executing the below command from your local machine:
   ```
   kubectl get svc <OmniDocsServices container name>
   ```
   For example,

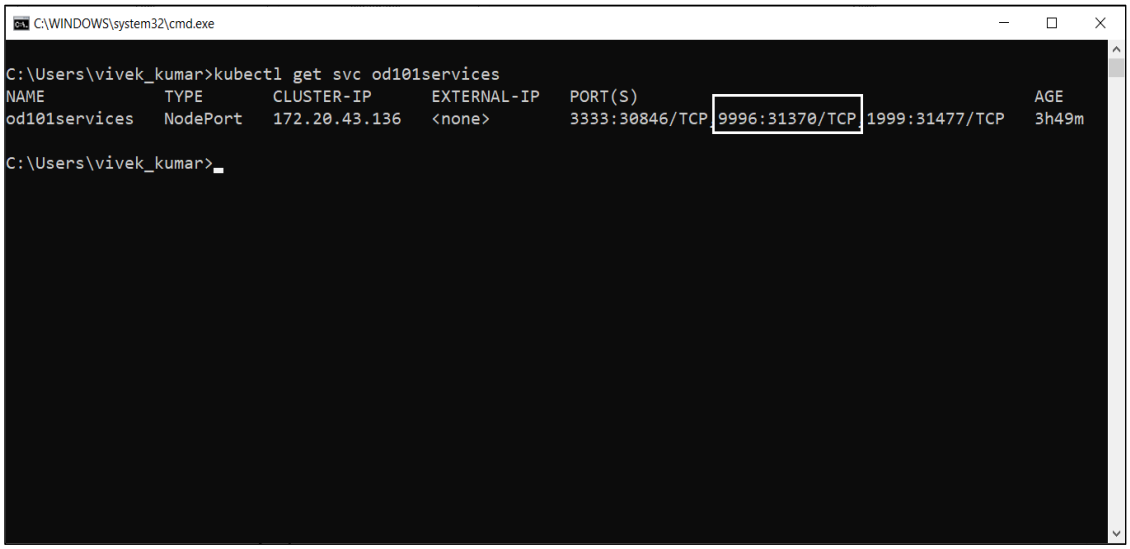

Figure 3.53

Here, **Wrapper Admin port 9990** is exposed outside the container and Kubernetes has generated a random port **31370** as a NodePort. This NodePort keeps changing whenever you redeploy the container.

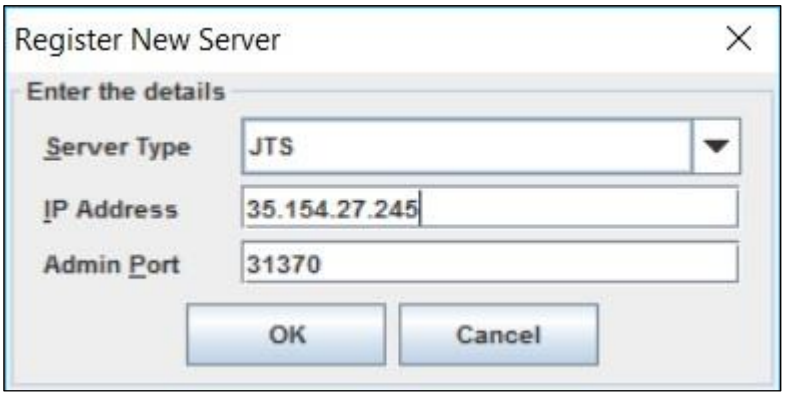

Figure 3.54

4. Click **OK** to register the JTS Server.

3.7.3 Connecting OSA to the JTS Server

Perform the below steps to connect the OSA to the JTS Server:

1. Once the JTS Server is registered, it is displayed in the list in a disconnected state.

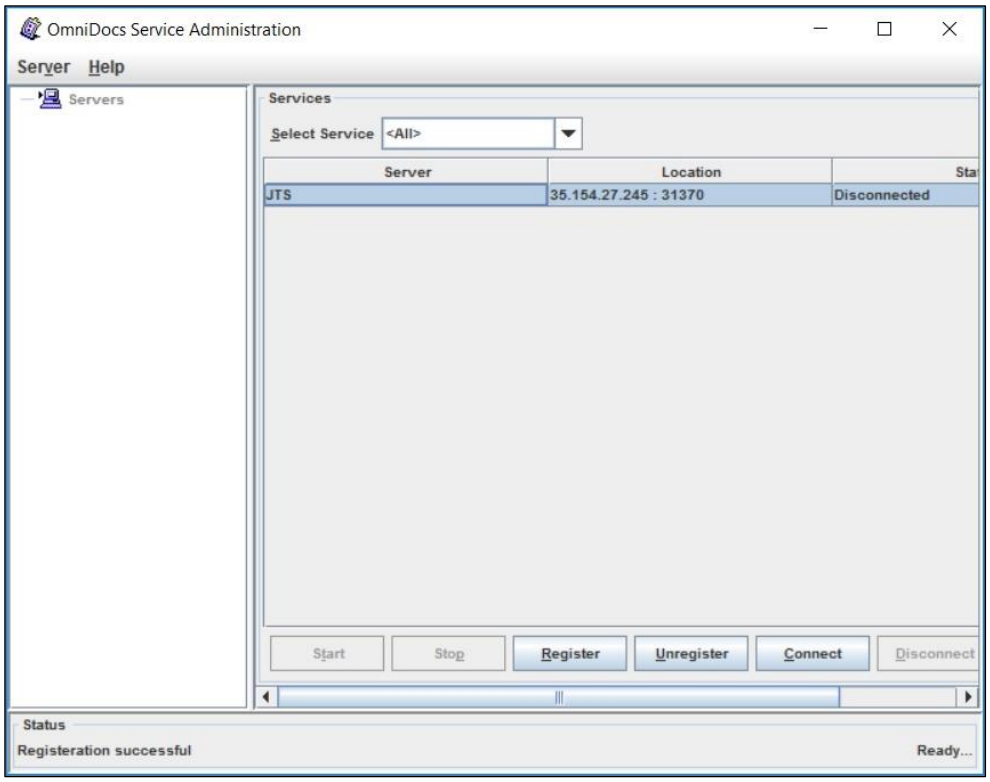

Figure 3.55

2. Select the registered JTS Server and click **Connect**. Once JTS is connected, the **Manage** button gets enabled.
3. Click **Manage** button, after clicking on the Manage button, an entry of the connected JTS server along with its IP Address is displayed on the upper-left panel in the repository view.
4. Select the JTS from the repository view. The list of already created and associated cabinets, appears.

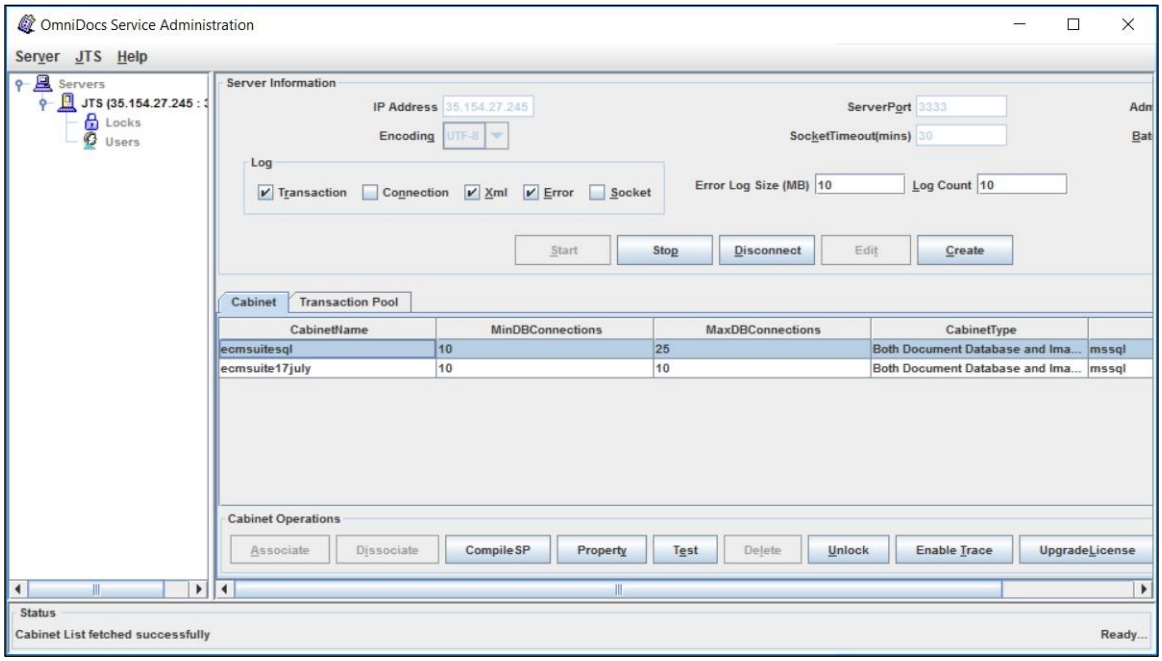

Figure 3.56

3.7.4 Creating a cabinet

Perform the below steps to create a cabinet:

For MSSQL:

1. Click **Create**. The Create Cabinet dialog appears.

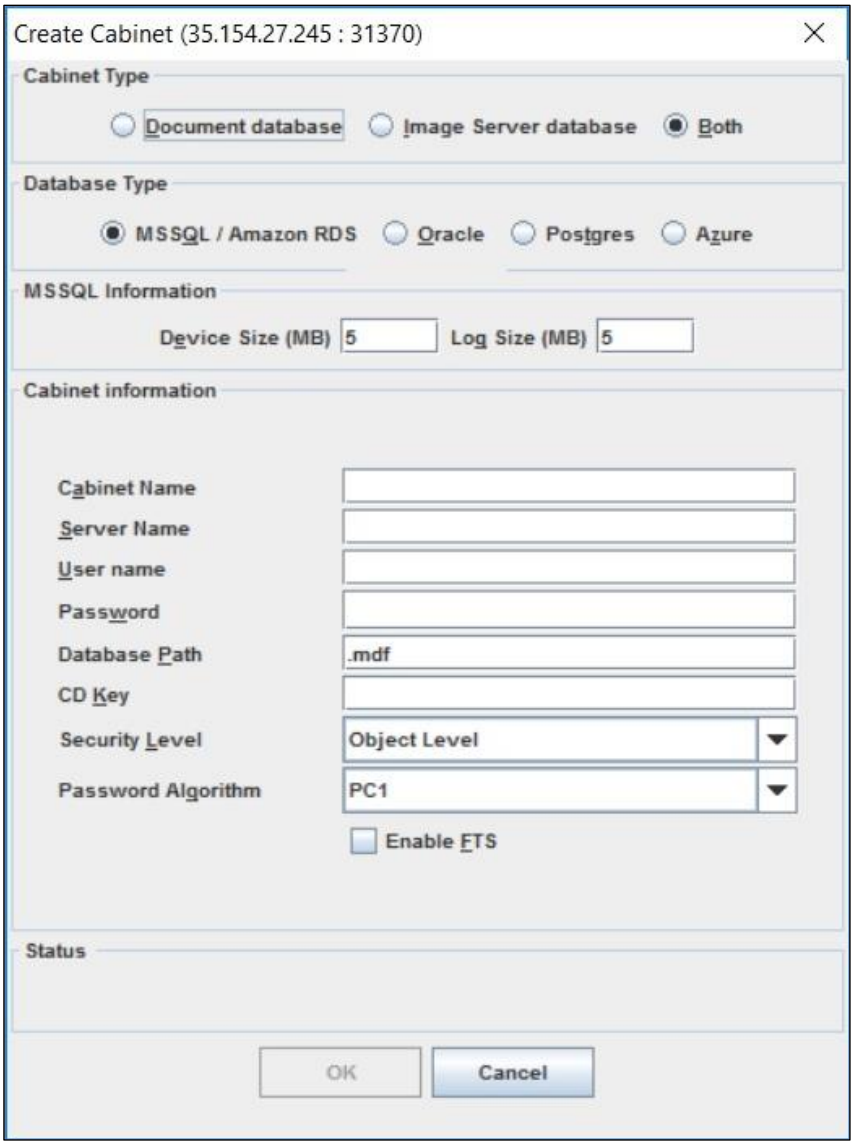

Figure 3.57

2. Select the cabinet type that needs to be created from the Cabinet Type area. The Cabinet can be a **Document database**, **an Image server database**, or both.
3. Select the database option from the Database Type section.
4. Specify the initial database size in the **Device Size** textbox and specify the initial log size in the **Log Size** textbox. Else, continue with the default values.
5. Specify the following cabinet information:
   - Specify the cabinet name in the **Cabinet Name** textbox.
   - Specify the server name (name of the machine where the MS SQL server is running) in the **Server I.P.** textbox.
   - Specify the username in the **User name** textbox.
   - Specify the password in the **Password** textbox.

- Specify the CD key in the **CD Key** textbox.
- Select the **Enable FTS** checkbox.

NOTE:
In the case of MSSQL if the Database port is not equal to 1433 (Default port) update the database port in the *DatabaseDriver.xml* file located inside the OmniDocsEjb/ngdbini folder kept inside the Azure FileShare before creating the cabinet.

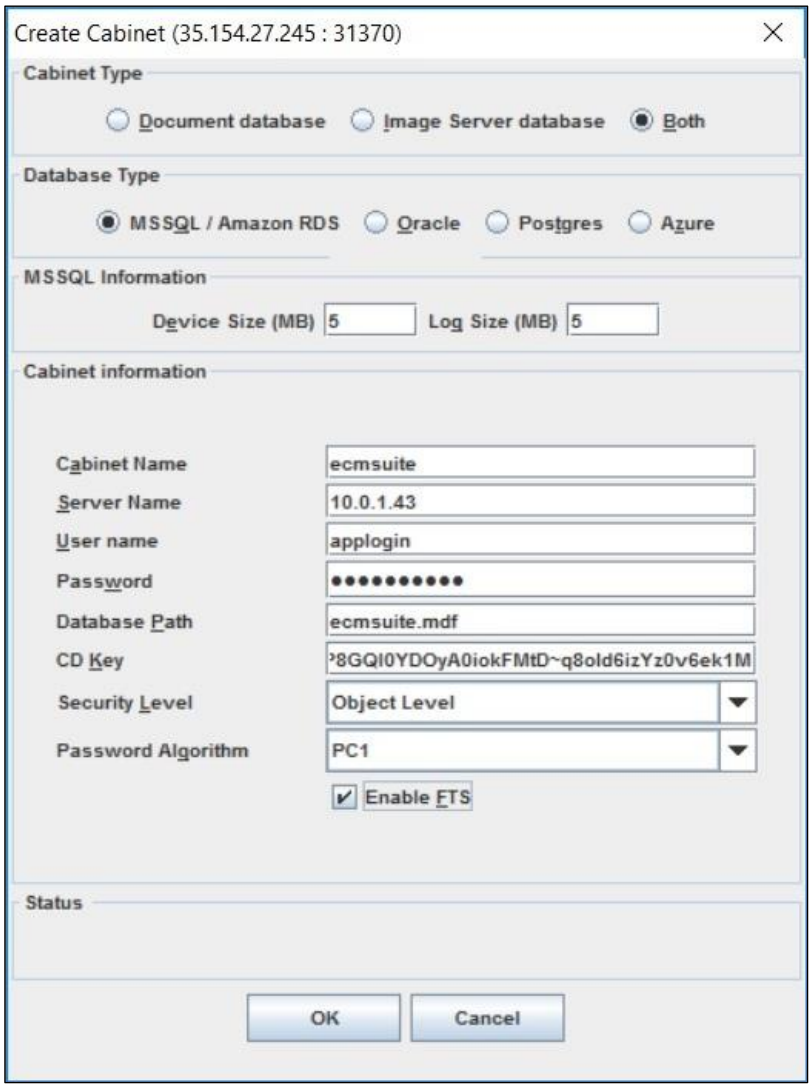

Figure 3.58

6. Click **OK** to create the cabinet. The Cabinet created successfully dialog appears.

Figure 3.59

3.7.5 Associating a cabinet

Perform the below steps to associate the cabinet:

For MSSQL:

1. Click **Stop** to enable the Associate button.
2. Click **Associate**. The Associate a Cabinet dialog appears with the following tabs:
   i. **Database tab:** Select the database type.
   ii. **Cabinet properties tab:** Specify the cabinet details that you have specified during cabinet creation.

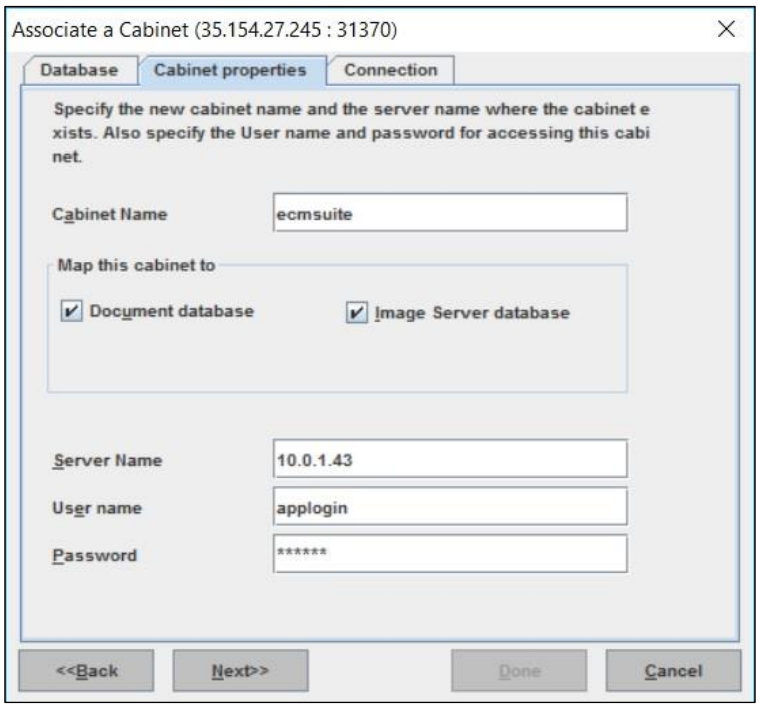

Figure 3.60

iii. **Connection tab:** Specify the **maximum** and the **minimum** number of connections that the JTS should maintain with the database, specify the **query time** out for the selected cabinet in the Query timeout text box and specify the **refresh interval** time for connection.

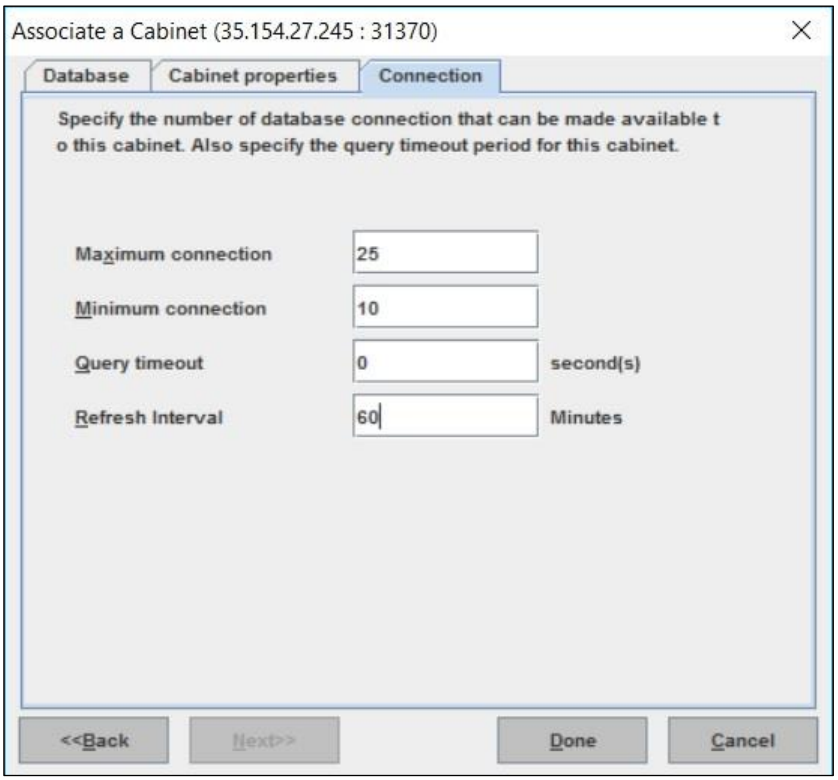

Figure 3.61

3. Click **Done** to associate the selected cabinet. Once the cabinet is associated successfully, it appears with the list.

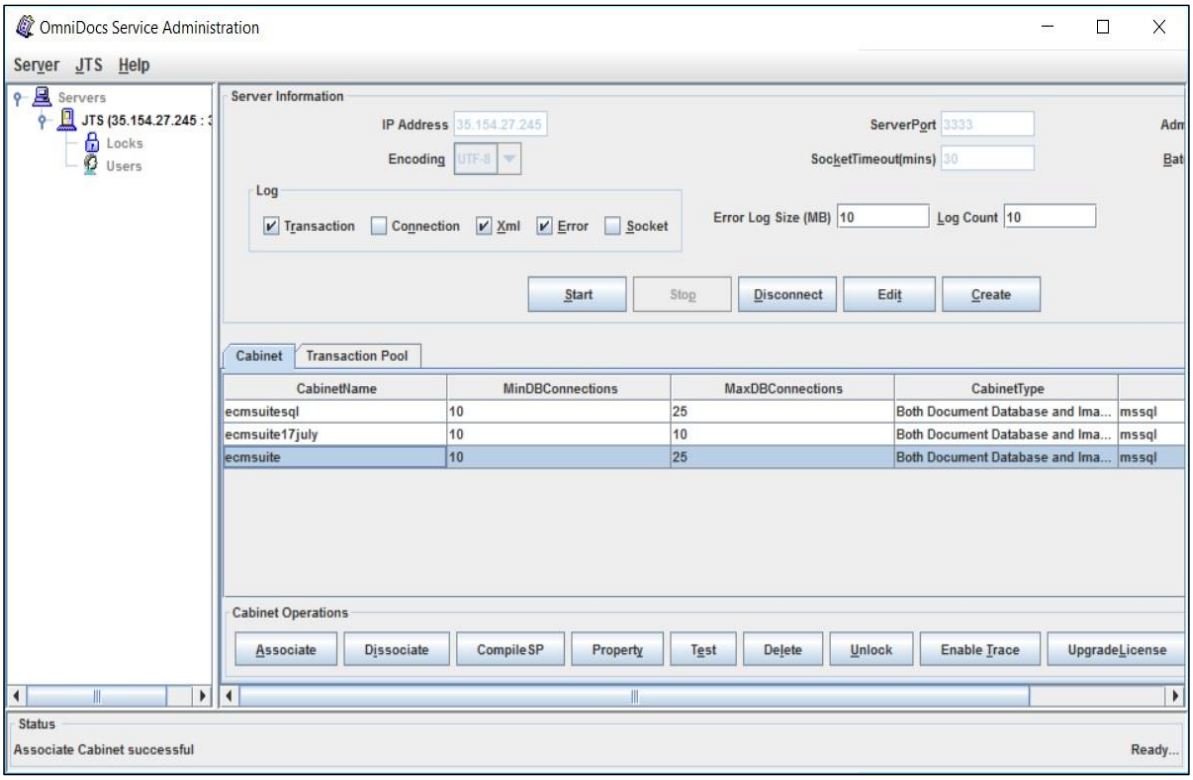

Figure 3.62

3.7.6 Creating a data source

Perform the below steps to create the data source:

For MSSQL:

1. Open the<Host-Path URL of OmniDocsEJB container> like *http://ecmsuiteconsole.newgendocker.com*.in as defined in the *AppGateway-IngressController.yml* file. It automatically redirects to the JBoss EAP 7.4 Admin console.
2. Enter the newgen as username and password system123# respectively to login to the Admin console. After a successful login, the Red Hat JBoss Enterprise Application Platform screen appears.

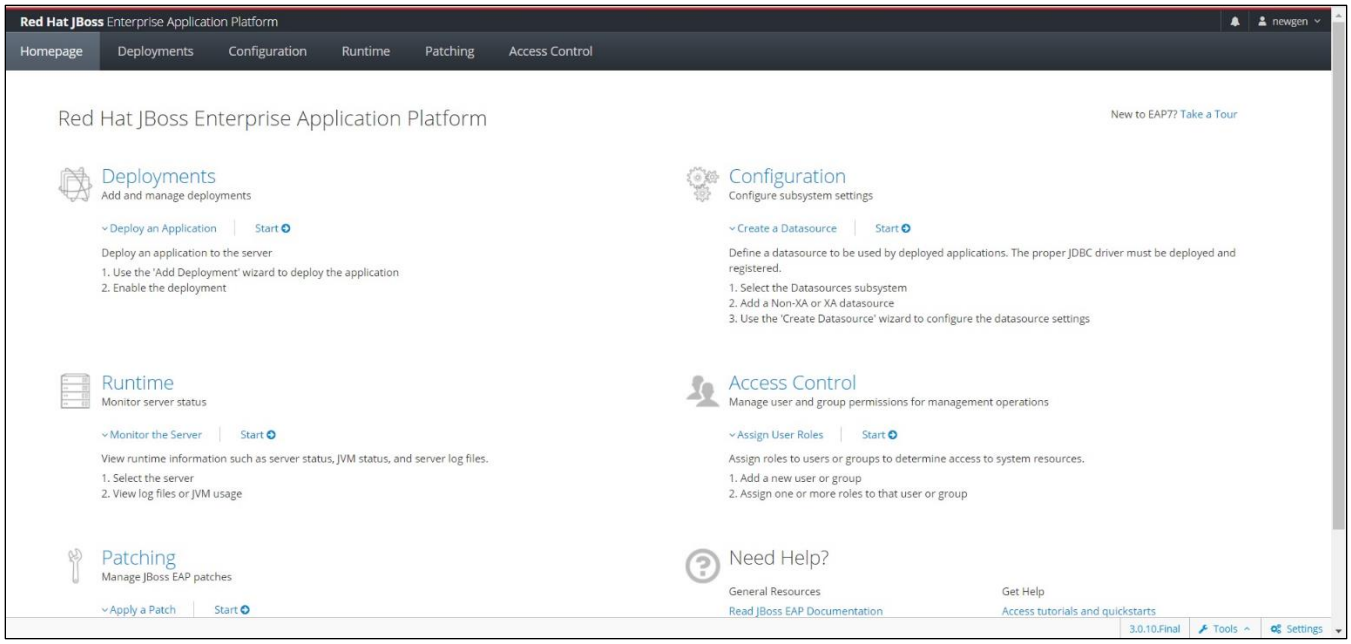

Figure 3.63

3. Go to the **Subsystems** in the Configuration tab.
4. Go to the **Datasources & Drivers**. Then, click Datasources.

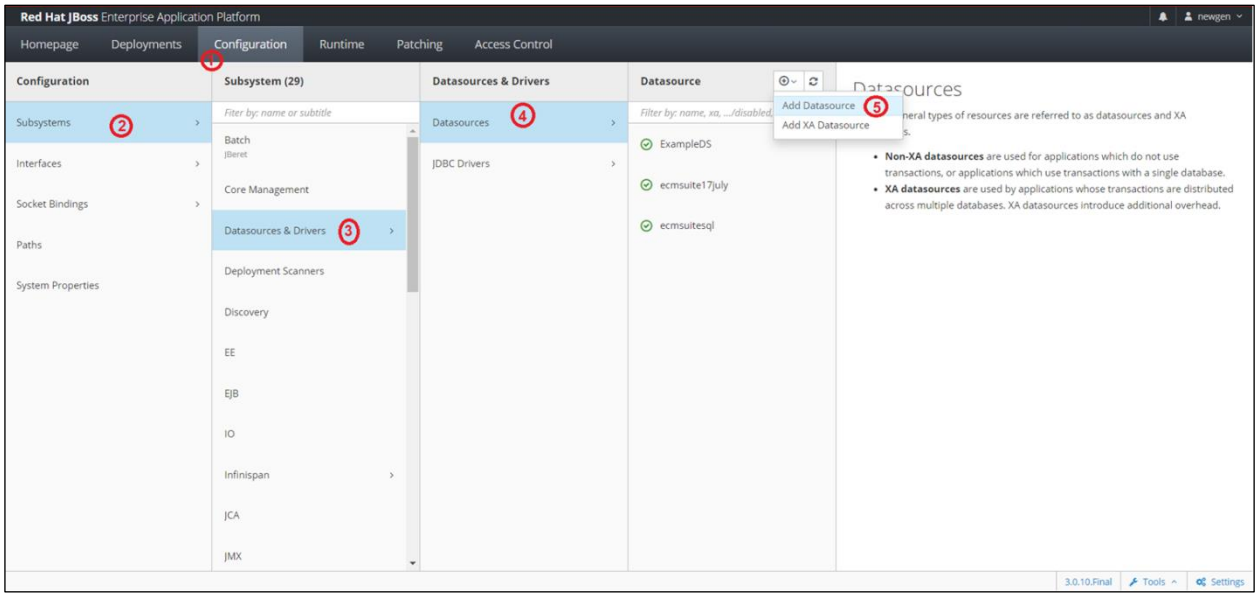

Figure 3.64

5. Click Plus **+** icon and select **Add Datasource**. The Add Datasource dialog appears.
6. For MSSQL Database Server, select **Microsoft SQLServer** and click **Next**.

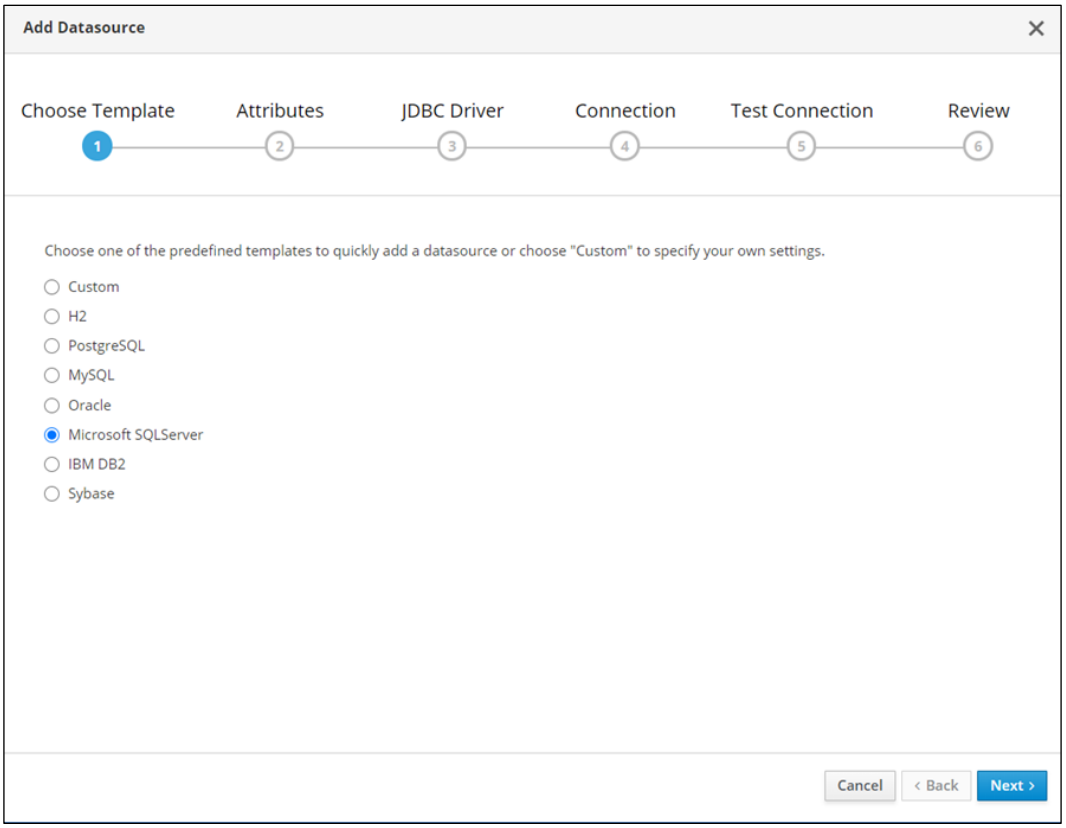

Figure 3.65

7. Provide a DataSource Name and JNDI Name.
   - **Name:** Enter the OmniDocs cabinet name that is cabinet name.
   - **JNDI Name:** java:/same as OmniDocs cabinet name
8. Click **Next**.

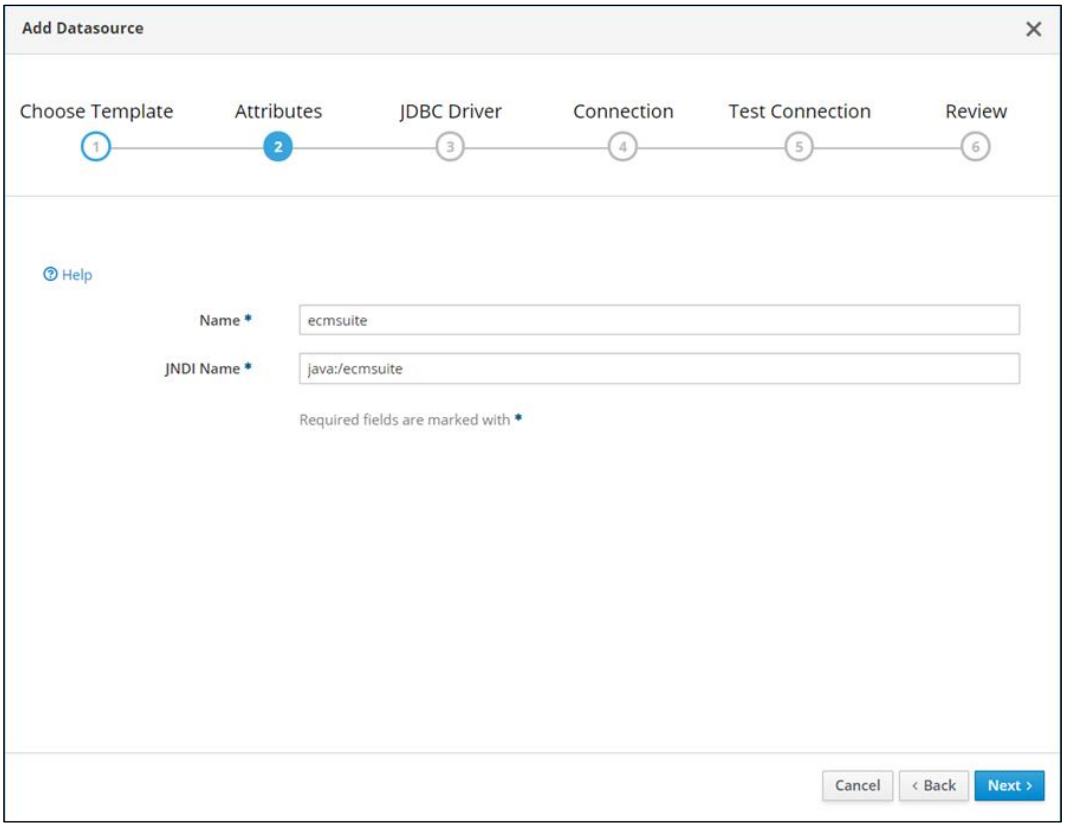

Figure 3.66

9. Select JDBC Driver Name.
10. For MSSQL, select **sqljdbc42.jar**.
11. Clear **Drive Module Name** and **Driver Class Name** textboxes.
12. Click **Next**.

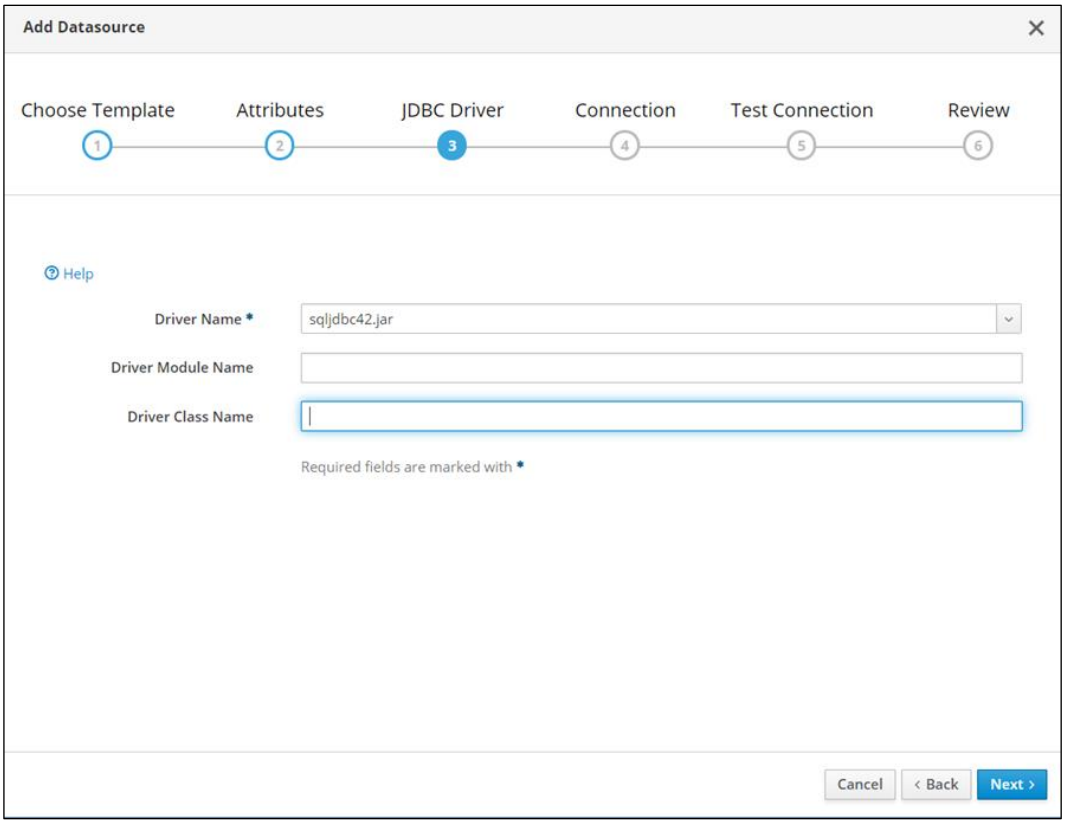

Figure 3.67

13. Provide the following Connection Setting details and click **Next**:
    - **Connection URL:** jdbc:sqlserver://MSSQL_Server_IP:MSSQL_Server_Port;databaseName=CABINET_NAME
    - **UserName:** Enter the SQL Server User Name
    - **Password:** Enter the SQL Server Password
    - **Security Domain:** Keep this blank.

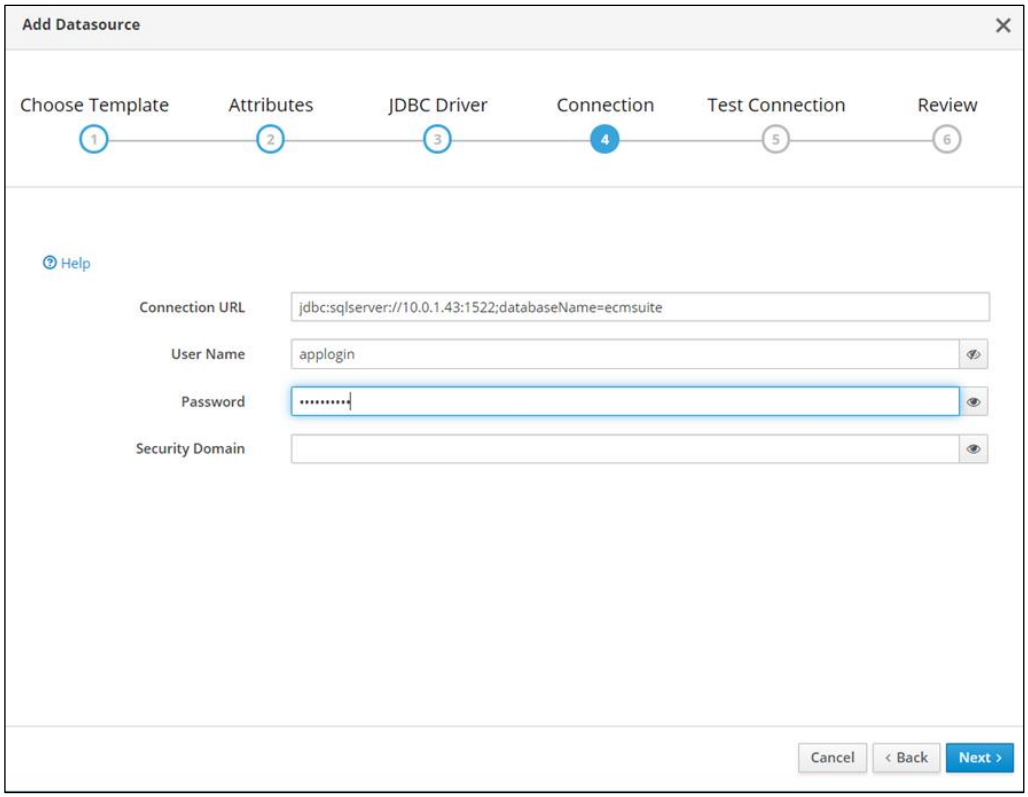

Figure 3.68

14. Click **Next** on the **Test Connection** page.

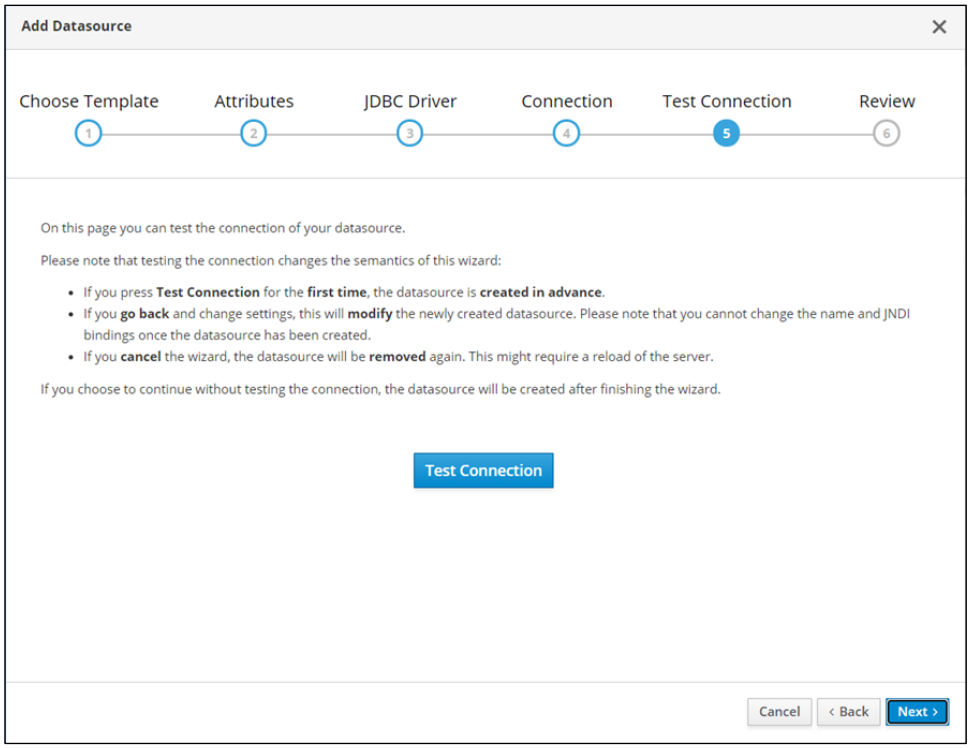

Figure 3.69

15. Click **Finish.** After the creation of the datasource, a success message appears.

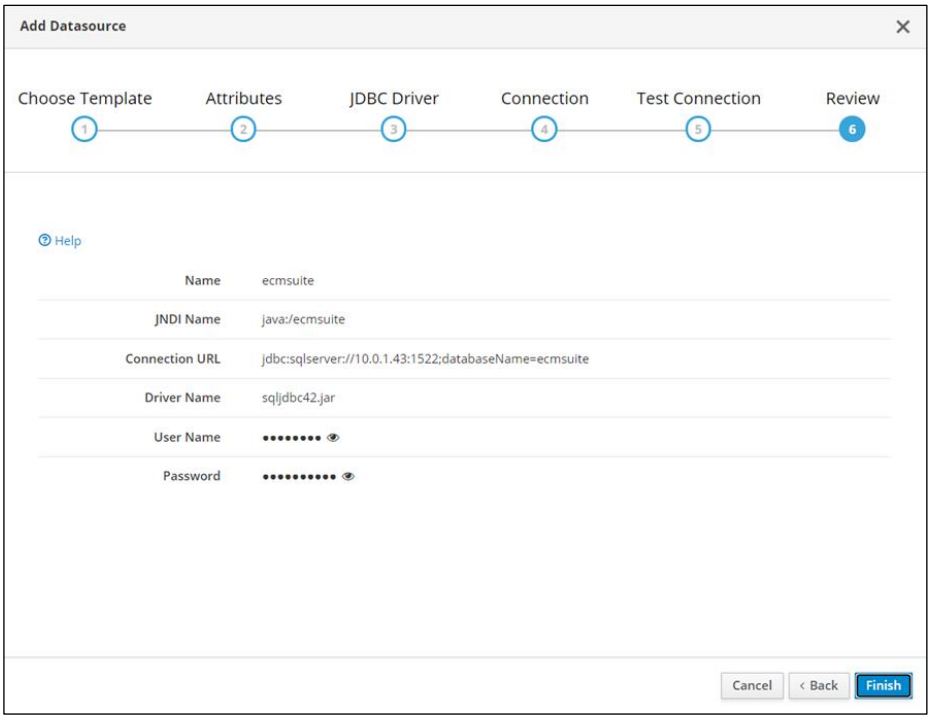

Figure 3.70

16. Click V**iew Datasource** to view the created datasource. The created datasource appears in the list of **Datasource**.

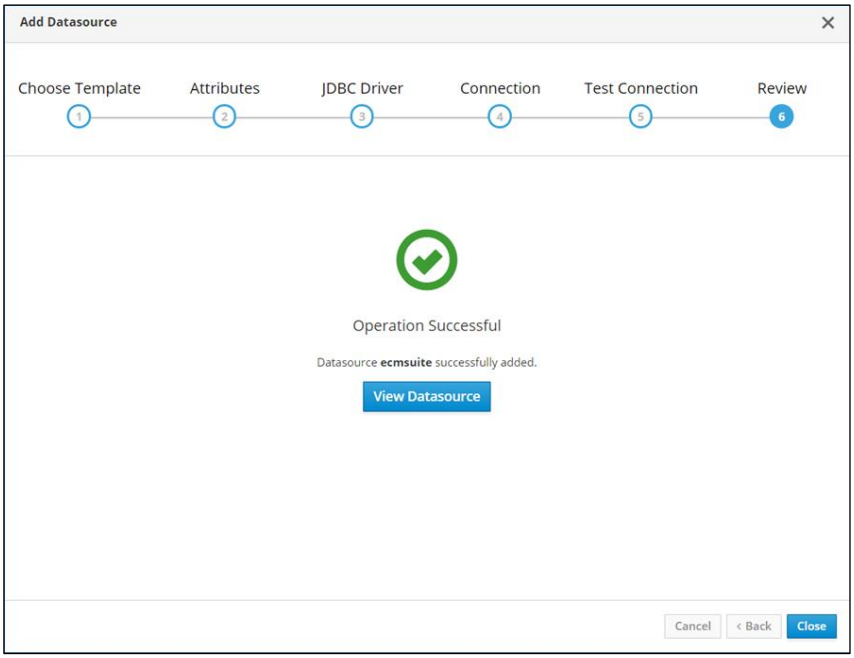

Figure 3.71

17. Click **View** against the datasource. A screen appears with the attributes of the datasource appears.
18. Click **Edit** link.

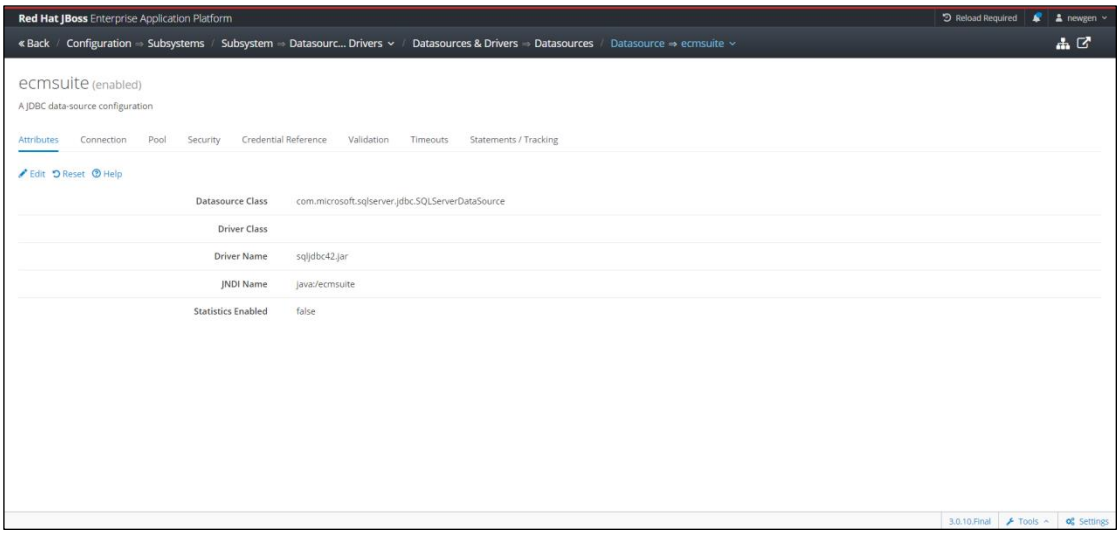

Figure 3.72

19. Clear the **Datasource Class** textbox and click **Save**.

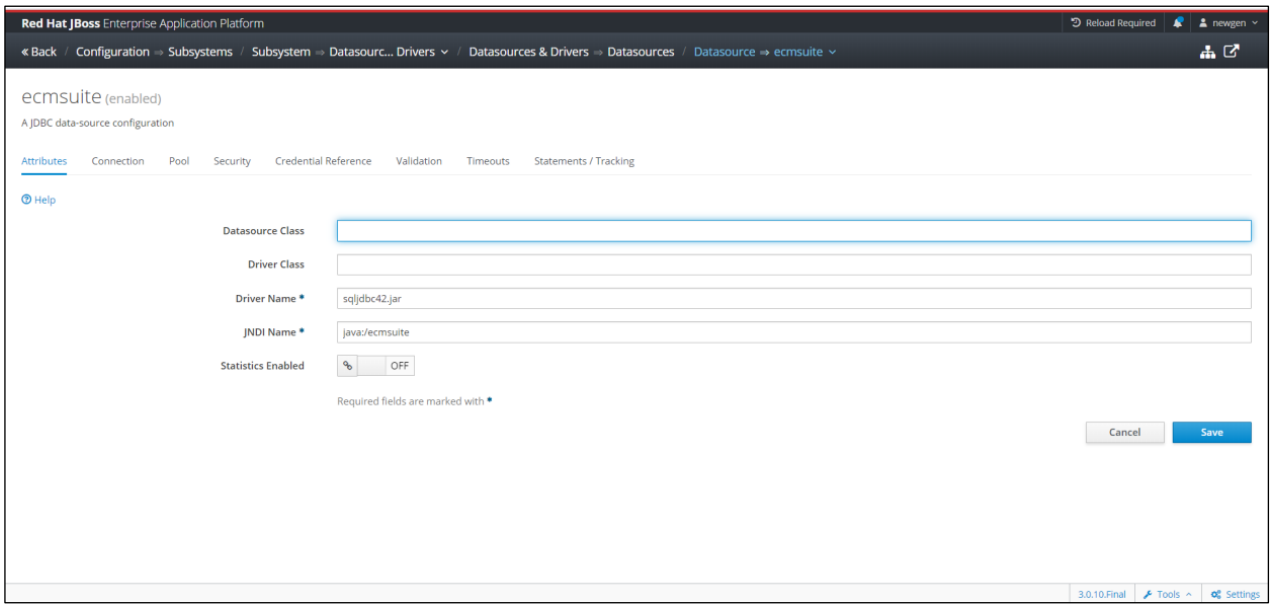

Figure 3.73

20. After that restart the OmniDocsEJB container.
21. Once the OmniDocsEJB container is restarted, open the JBossEAP Admin console once again.
22. Go to the **Subsystems** in the Configuration tab.
23. Go to the **Datasources & Drivers**. Then, click Datasources.

    Select the created data source and click **Test connection** from the dropdown list. On the successful data connection, a success message appears.

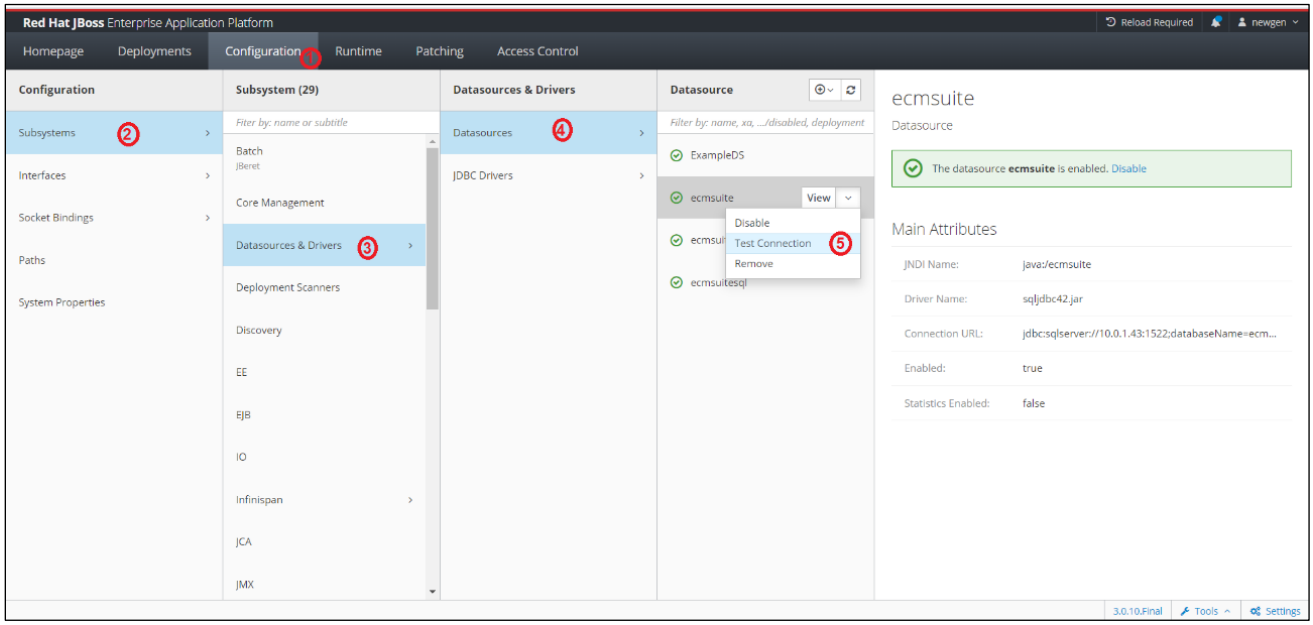

Figure 3.74

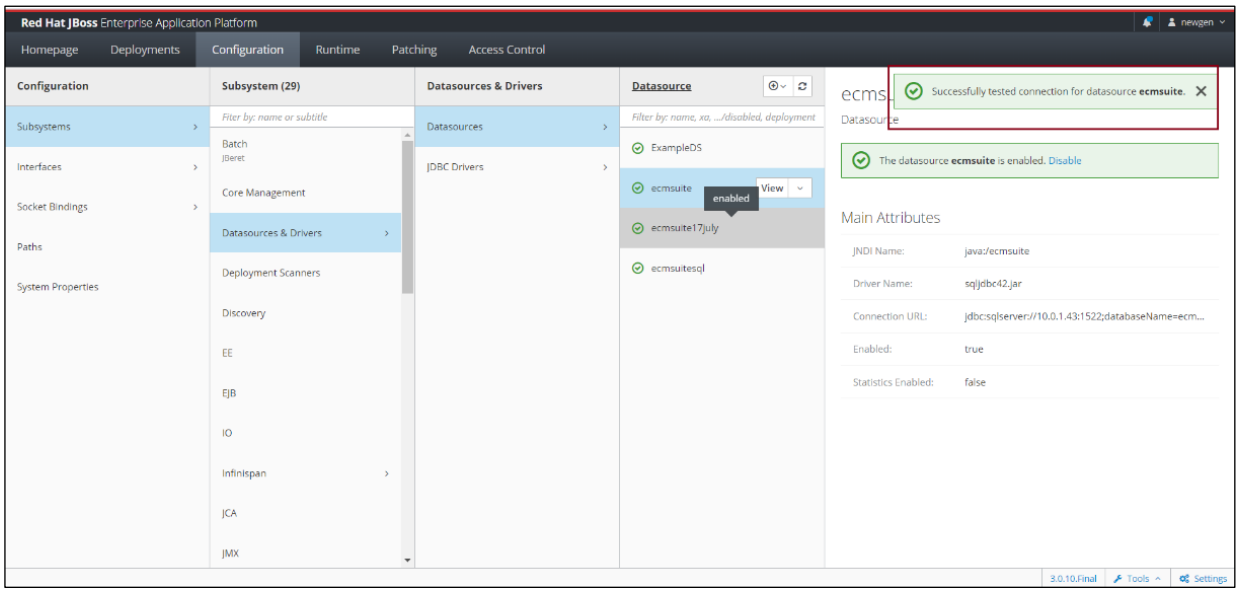

Figure 3.75

24. Add the below connection pool setting and idle-connection-timeout setting inside the created DataSource in *standalone.xml* file located inside the **OmniDocsEjb** or **configuration** folder kept inside the Azure FileShare.

```
<pool>
      <min-pool-size>100</min-pool-size>
      <initial-pool-size>100</initial-pool-size>
      <max-pool-size>600</max-pool-size>
      <flush-strategy>Gracefully</flush-strategy>
</pool>

<timeout>
      <idle-timeout-minutes>5</idle-timeout-minutes>
</timeout>
```

For example,

```
<datasource jndi-name="java:/auroraod23oct1" pool-name="auroraod23oct1">
    <connection-url>jdbc:
    postgresql://omnidocs[redacted]re-cluster.cluster-cv4updtekwxu.ap-south-1.rds.amazonaws.com:5432/auroraod23oct1
    </connection-url>
    <driver>postgresql-42.2.18.jar</driver>
    <pool>
        <min-pool-size>0</min-pool-size>
        <initial-pool-size>0</initial-pool-size>
        <max-pool-size>600</max-pool-size>
        <flush-strategy>Gracefully</flush-strategy>
    </pool>
    <security>
        <user-name>po[redacted]res</user-name>
        <password>a[redacted]#</password>
    </security>
    <validation>
        <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker"/>
        <background-validation>true</background-validation>
        <background-validation-millis>10000</background-validation-millis>
        <exception-sorter class-name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter"/>
    </validation>
     <timeout>
        <idle-timeout-minutes>5</idle-timeout-minutes>
    </timeout>
</datasource>
```

Figure 3.76

25. Restart the **OmniDocsEJB** container once again.

3.7.7 Registering a cabinet

Perform the below steps to register a cabinet:

1. Register the cabinet for OmniDocs Admin using the following URL:
   http*://<Host-Path URL of* OmniDocsWeb container>/omnidocs/admin/main/registration/registration.jsp
   For example,
   http://ecmsuite. newgendocker.com /omnidocs/admin/main/registration/registration.jsp

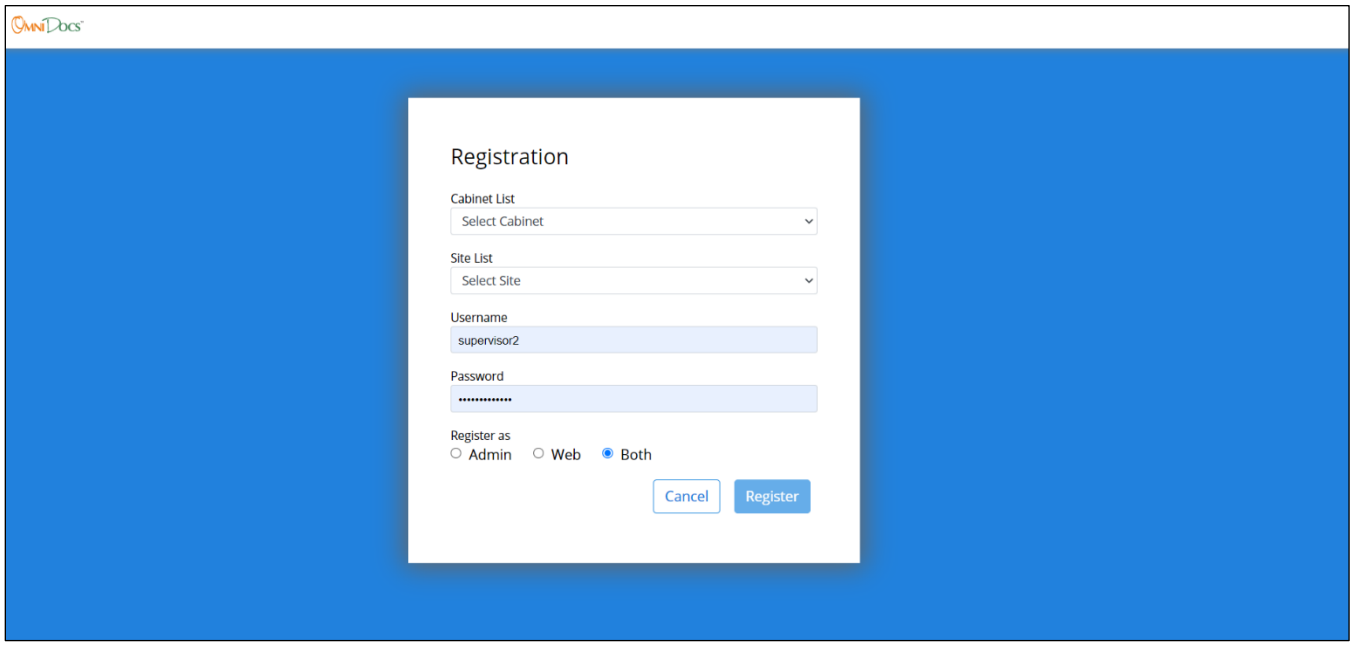

Figure 3.77

All the created cabinets get auto populated in the **Cabinet List** dropdown list.

2. Select the required cabinet, select the associated site, and specify the **Username** and **Password**.
3. Select the Register as **Both** and click **Register.** After successful registration, a confirmation message appears.

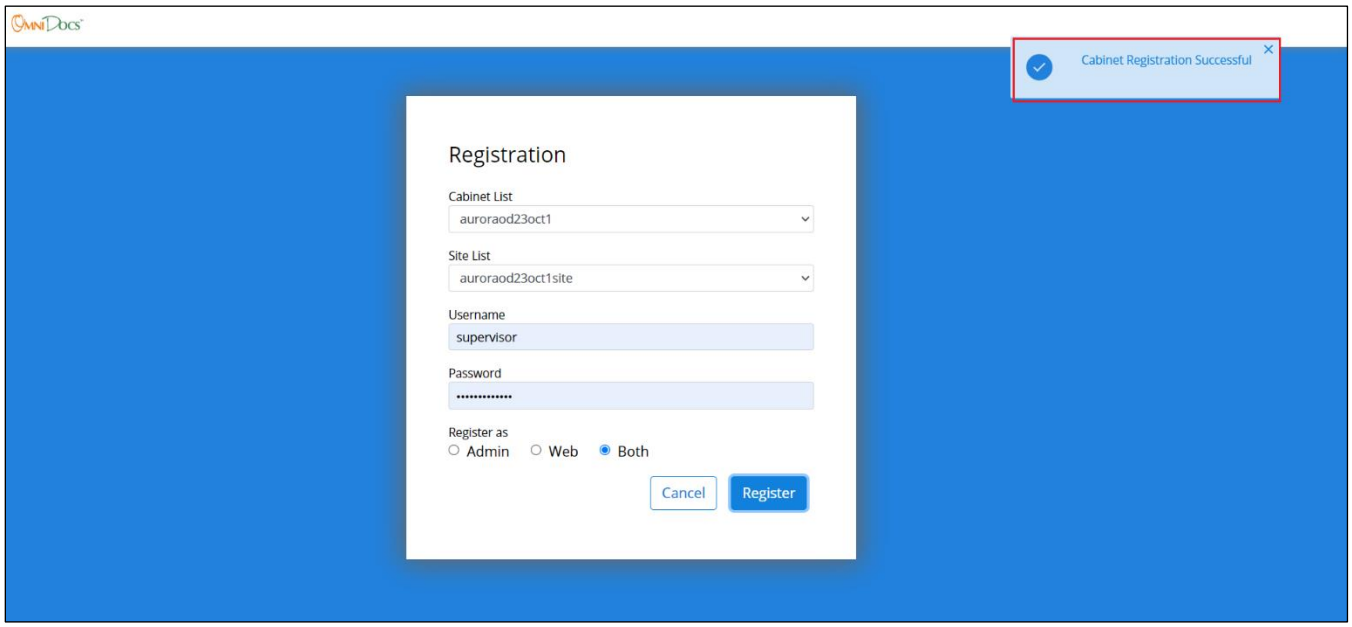

Figure 3.78

3.7.8 Creating site and volume

Perform the below steps to create site and volume:

1. Login to the OmniDocs Admin using the following URL:
   http*://<Host-Path URL of OmniDocsWeb container>/omnidocs/admin*
   For example,
   http://ecmsuite.newgendocker.com/omnidocs/admin

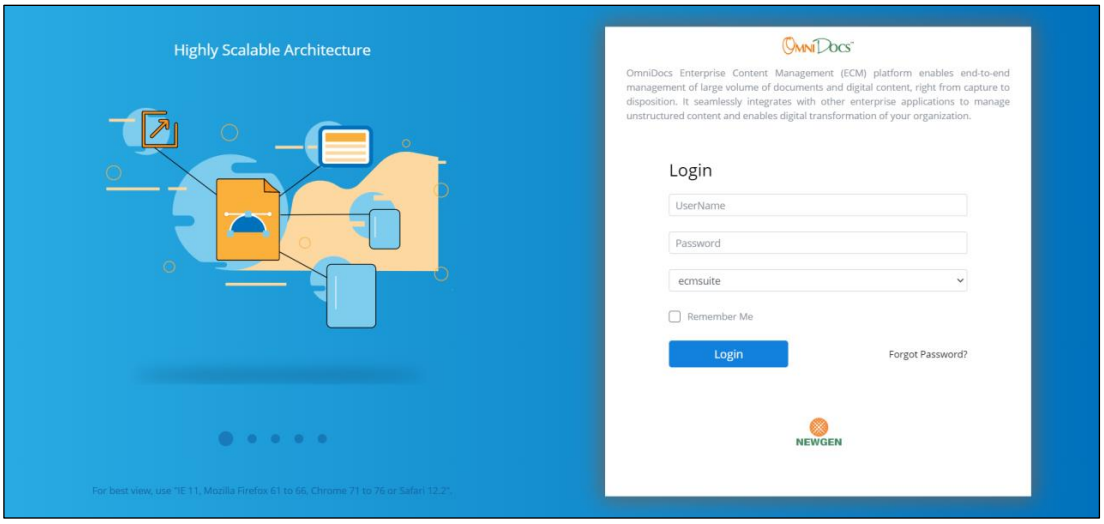

Figure 3.79

2. After a successful login, click **Sites** link under **Administration**.

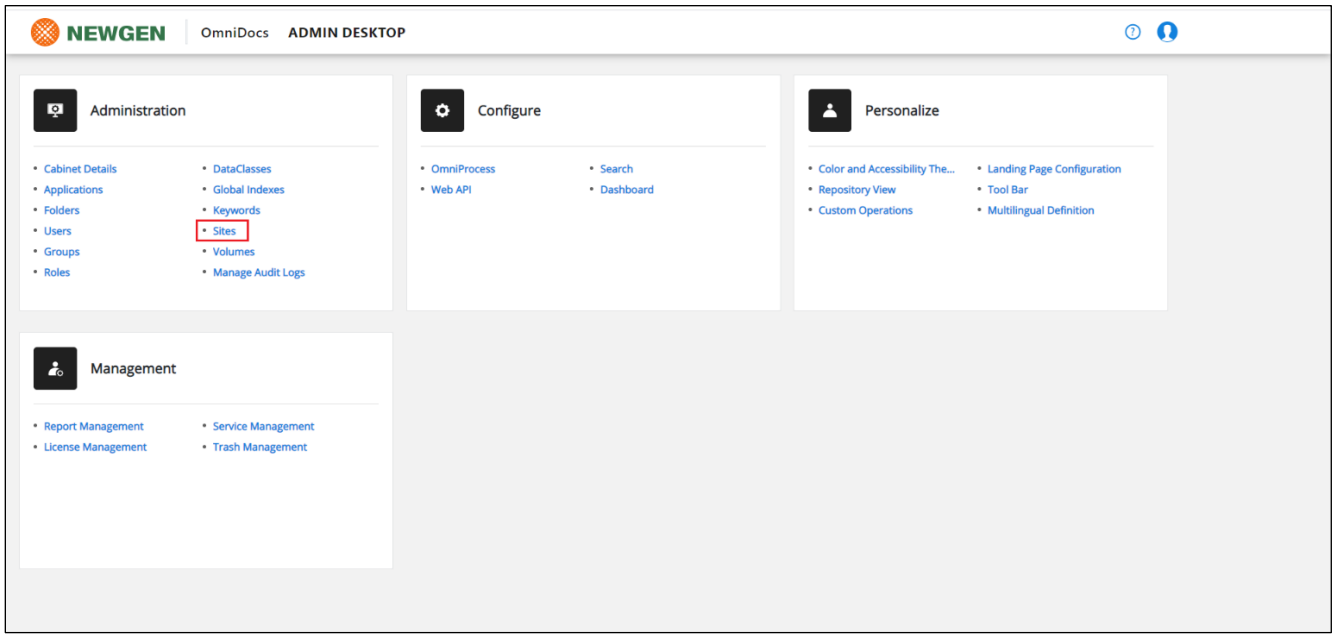

Figure 3.80

3. Click **+Add**. The Add Site dialog appears.

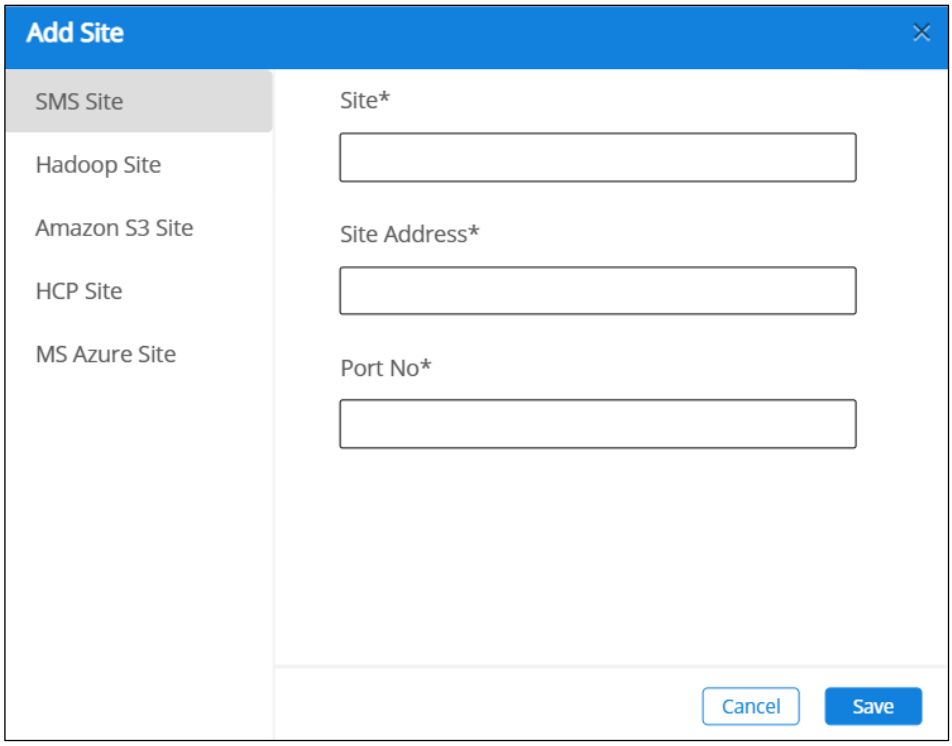

Figure 3.81

4. Click **Amazon S3 Site**.
5. Specify the user-defined site name, **Access Key**, and **Secret Key** that have rights to the S3 bucket.
6. Click **Save**.

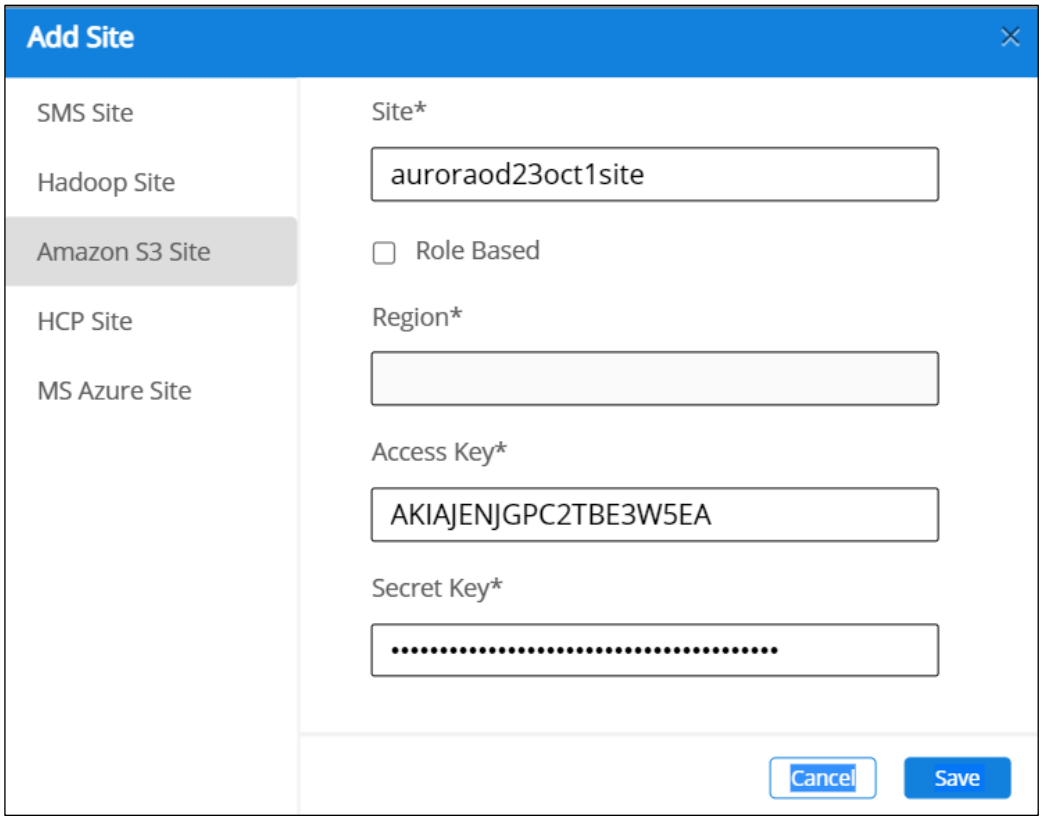

Figure 3.82

The added Site appears under Sites in the left pane.

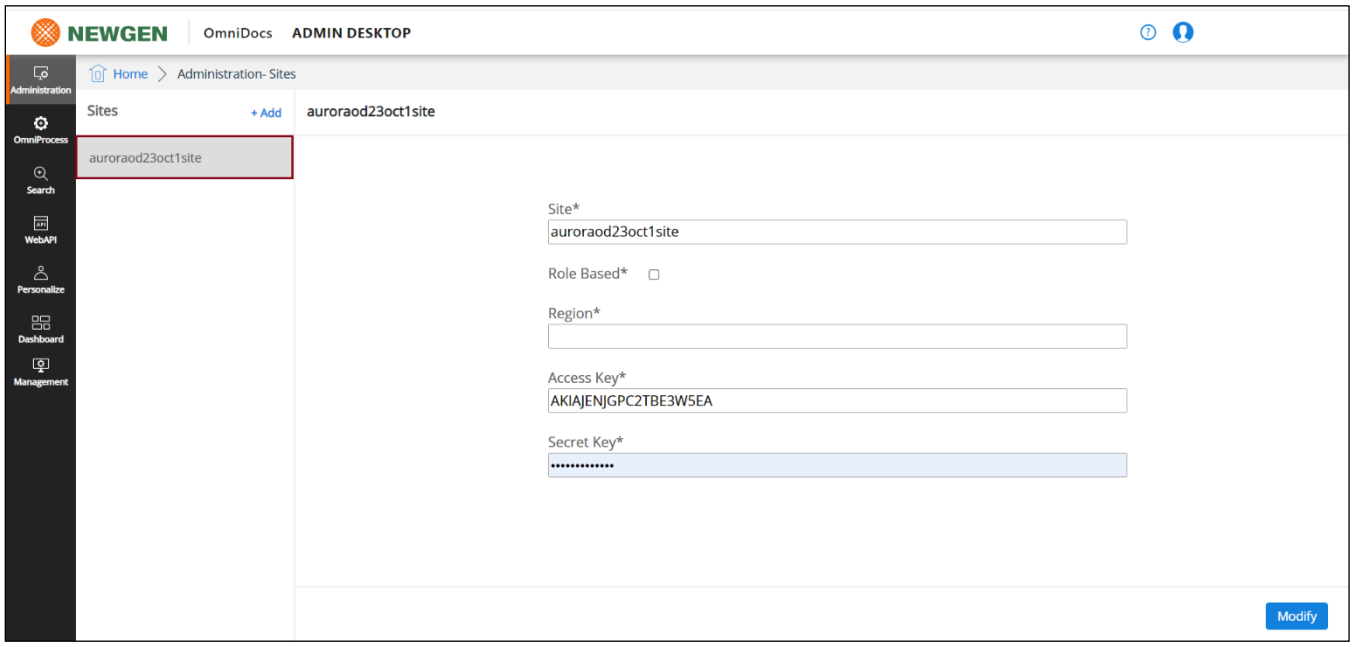

Figure 3.83

7. Go back to the **Home** page.

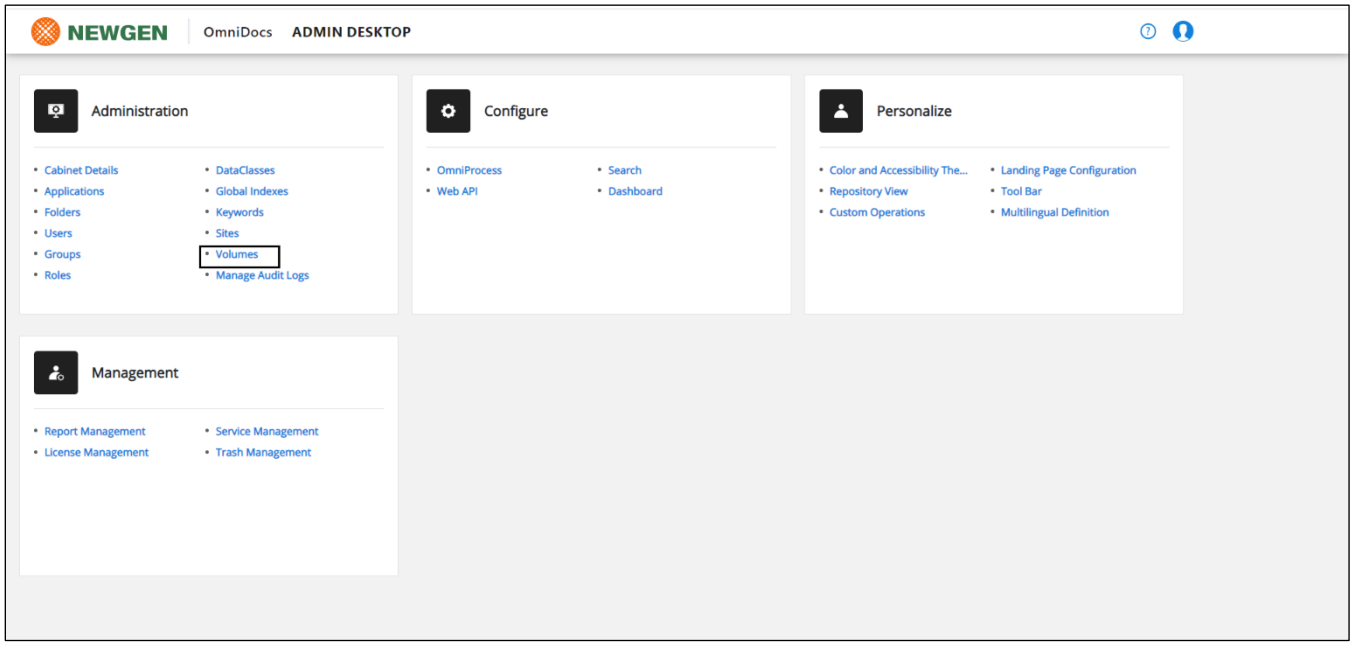

Figure 3.84

8. Select **Volumes**. The Volumes screen appears.

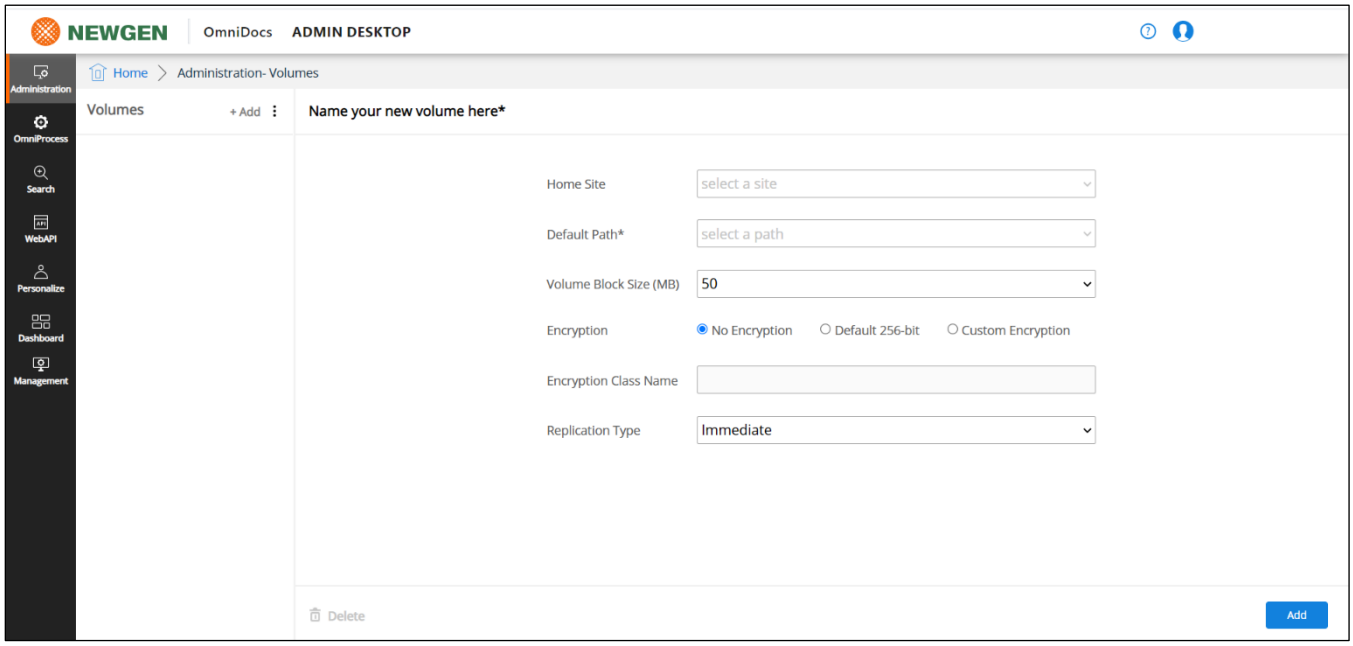

Figure 3.85

9. Specify the following details:

- **Home Site**: Select the newly created Site name.
- **Default Path:** Select the S3 bucket in which you want to store PN files.
- **Volume Name**: Specify the user-defined volume name.

10. Click **Add**.

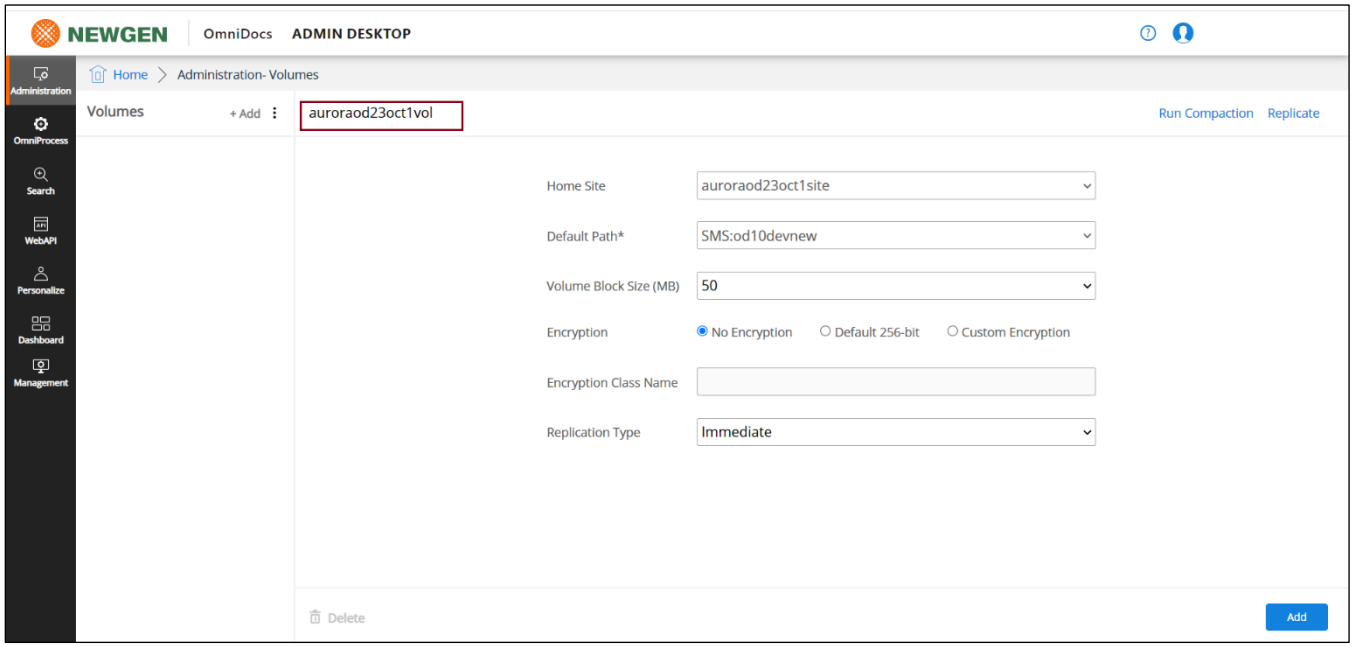

Figure 3.86

The added volume appears under **Image Volumes** in the left panel.

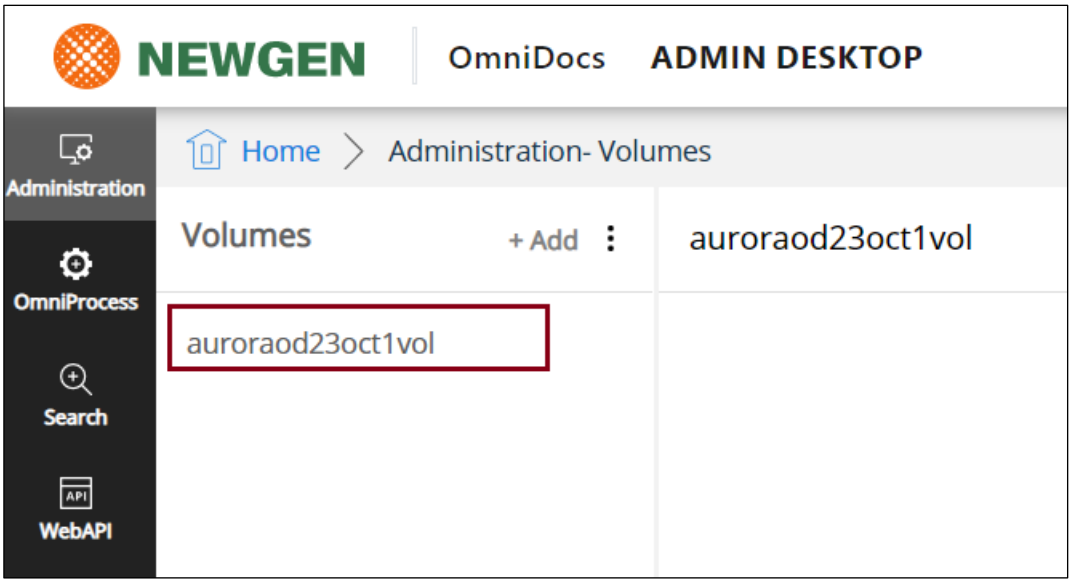

Figure 3.87

11. Go back to the **Home** screen.

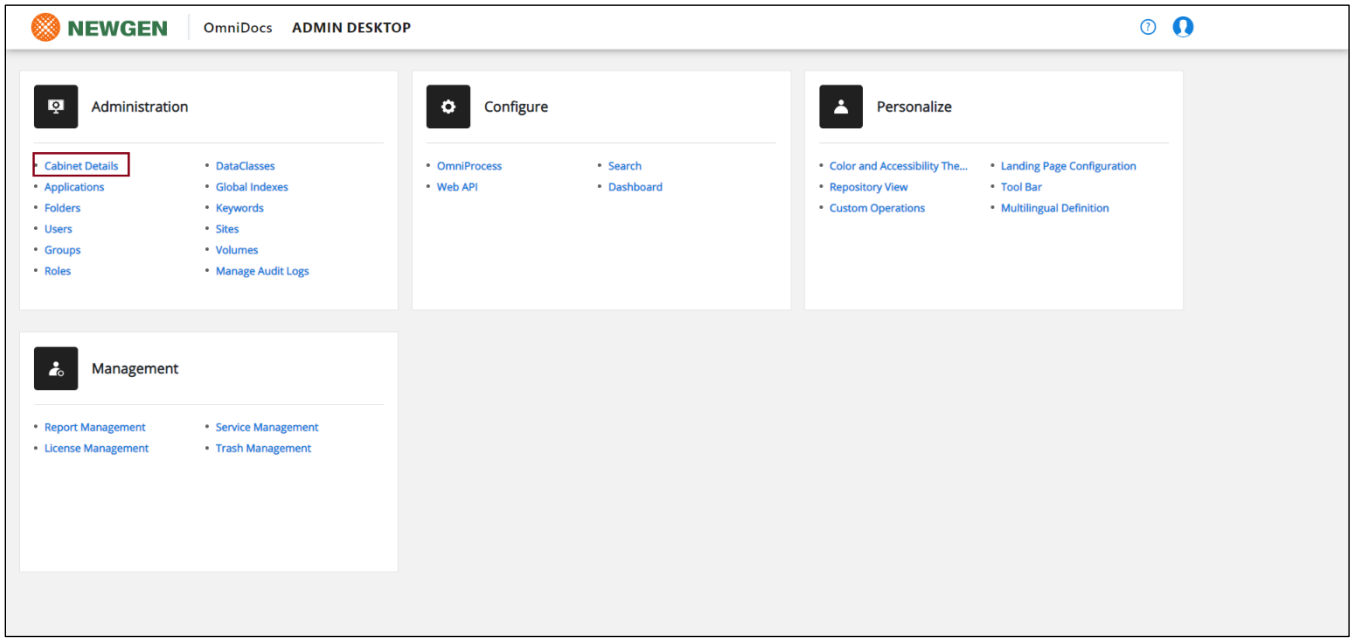

Figure 3.88

12. Click **Cabinet Details**.
13. Select the added volume from the **Default Image Volume** using the dropdown
14. Click **Save**. The Site and Volume are now created successfully.

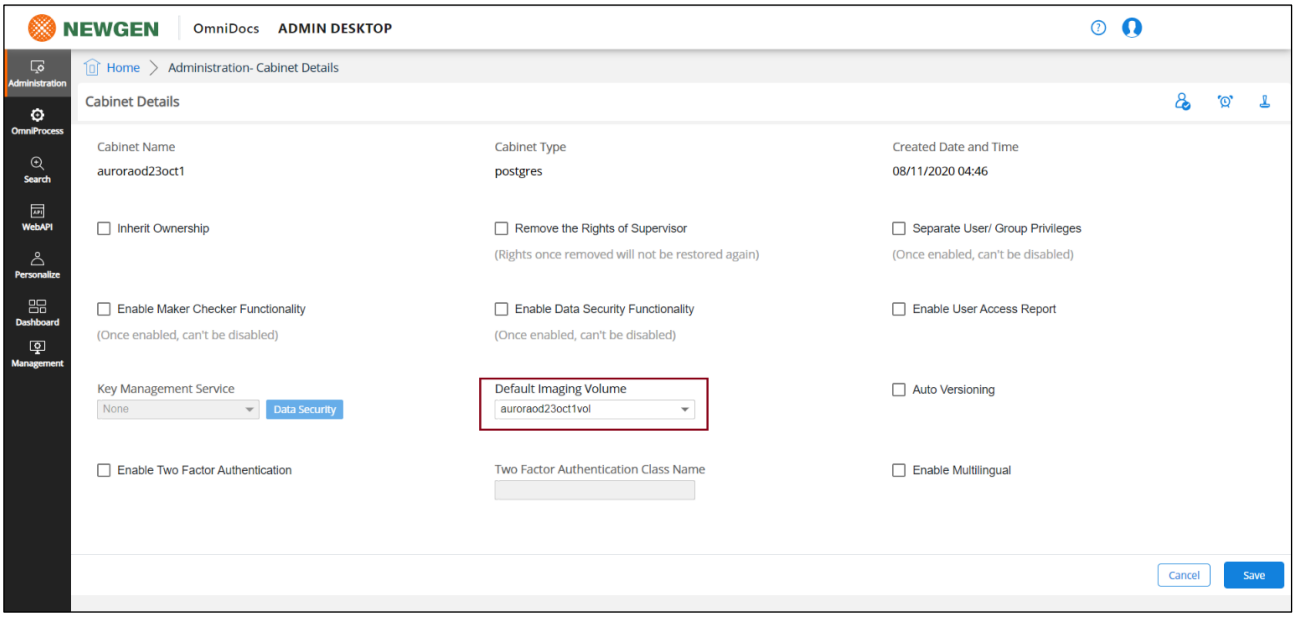

Figure 3.89

15. Log in to the OmniDocs Web using the below URL to start.
    http://<Host-Path URL of OmniDocsWeb container>/omnidocs/web
    For example: *http://ecmsuite.newgendocker.com/omnidocs/web*

3.8 EasySearch post-deployment changes

Perform the below steps to do EasySearch post-deployment changes:

1. Login to the ApacheManifold Admin using the following URL:
   <Host-Path URL of ApacheManifold>/mcf-crawler-ui/login.jsp
   For example,
   http://ecmsuiteapache.newgendocker.com/mcf-crawler-ui/login.jsp

Figure 3.90

2. Log in with the following credentials:
   - **User ID:** admin
   - **Password**: admin
3. After a successful login, click **Jobs** tree showing in the left panel.
4. Click **Status and Job Management**. The below job list appears:
   - <CABINET_NAME>_Document
   - <CABINET_NAME>_Folder
5. Start both the jobs.
6. Once both the jobs started, the Job's status appears as **Running.**

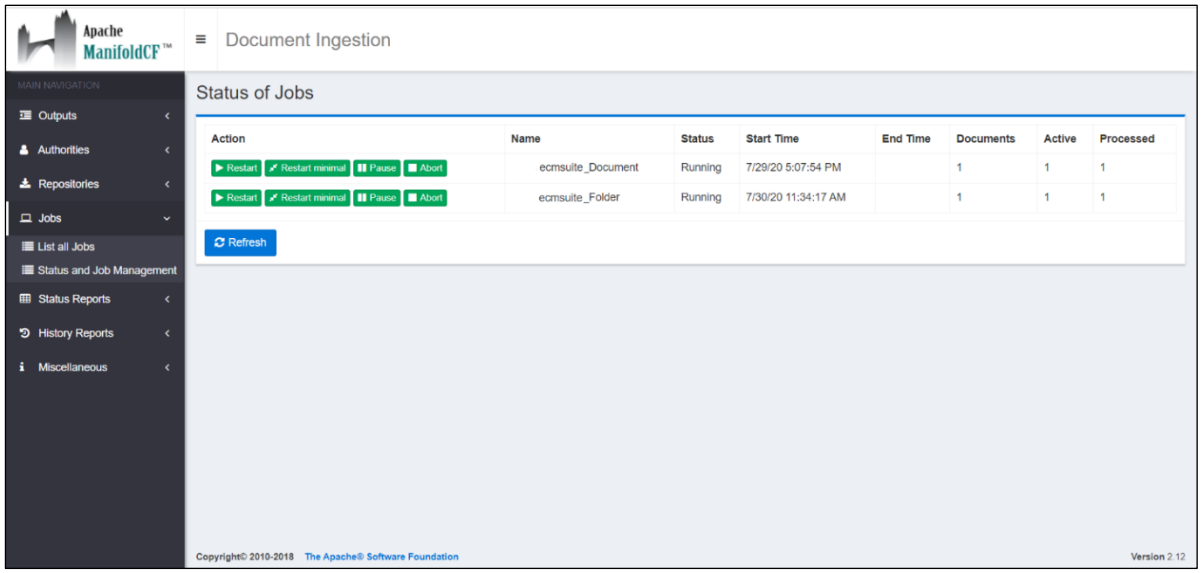

Figure 3.91

3.9 OmniScanWeb: registration of cabinet

Perform the below steps to register the cabinet in OmniScanWeb:

1. Open the OmniScanWeb using the following URL:
   http://<Host-Path URL of OmniScanWeb container>/omniscanweb
   For example,
   https://omniscan.newgendocker.com/omniscanweb
2. Click **Register New Cabinet** link on the OmniScan Web login screen.

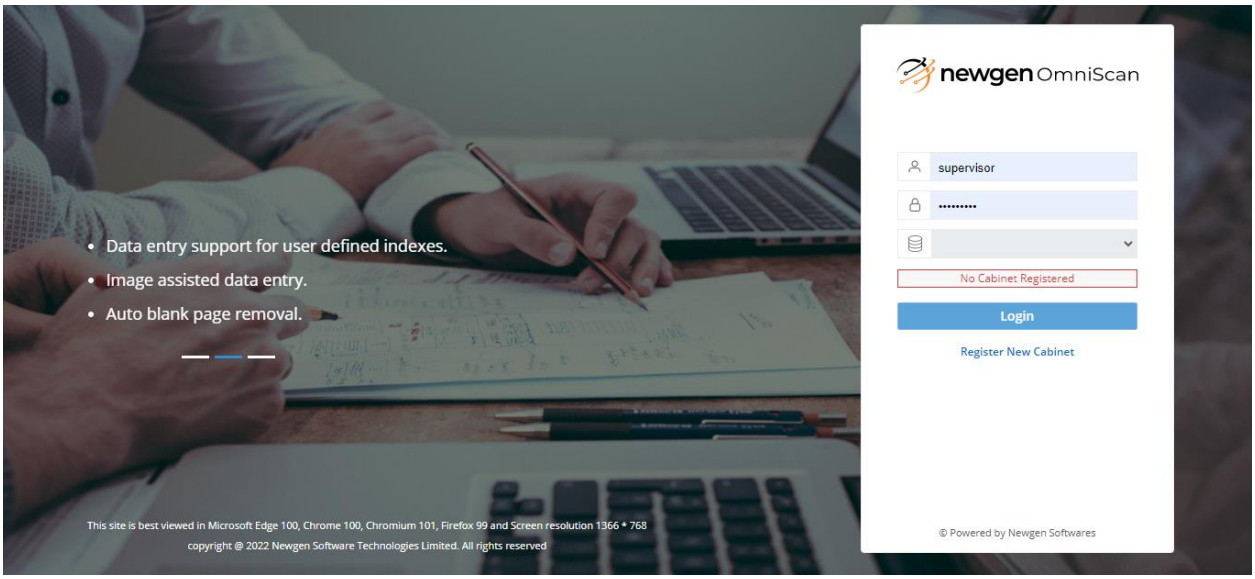

Figure 3.92

3. Specify the Server URL as given below:
   http://<Host-Path URL of OmniDocsWeb container>/NGServlet/servlet/ExternalServlet
   For example,
   https://omnidocs.newgendocker.com/NGServlet/servlet/ExternalServlet
4. Specify the **OmniDocs EJB** container name for AppServer IP or Server URL, 8080 for AppServer Port, and JBOSSEAP for AppServer Type.

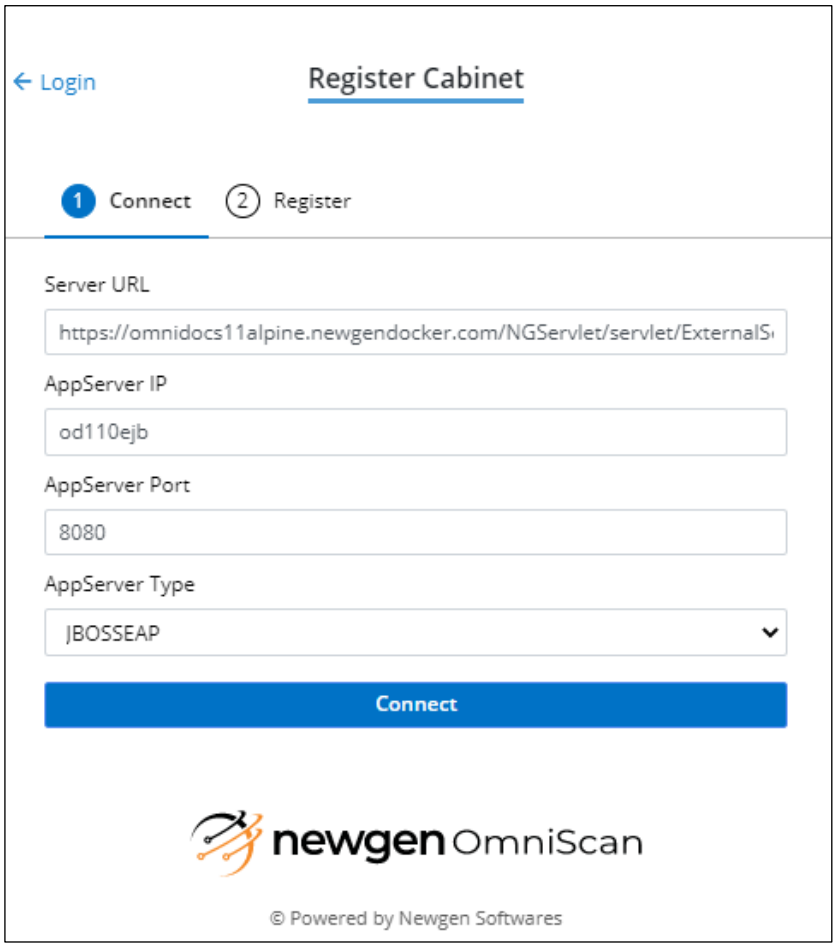

Figure 3.93

5. Click **Connect**.
6. Select the **Cabinet Name**, **Site ID**, and **Volume ID** from the list.

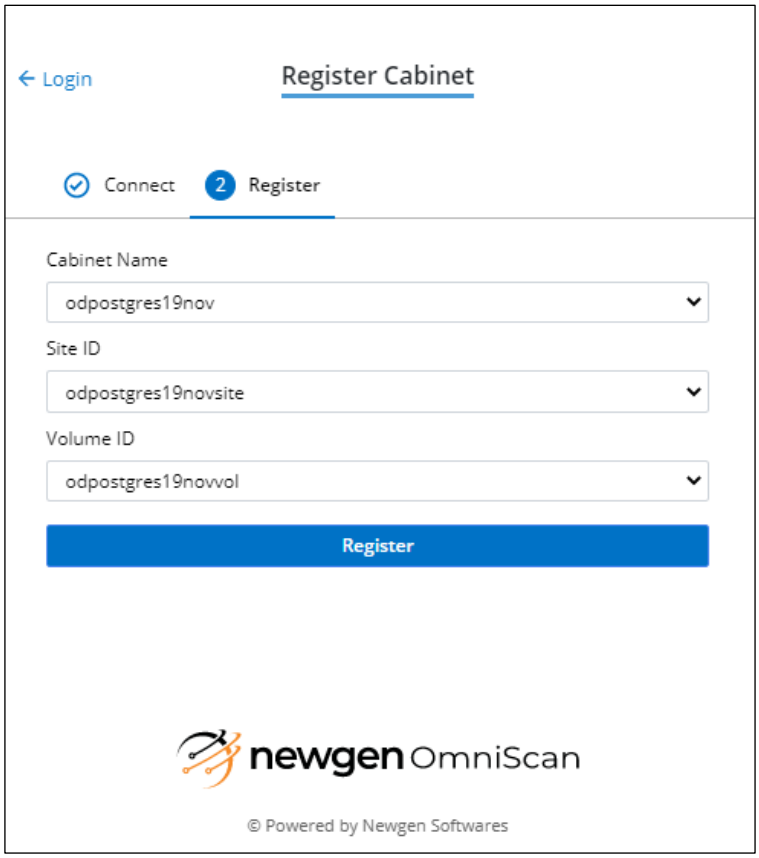

Figure 3.94

7. Click **Register**.

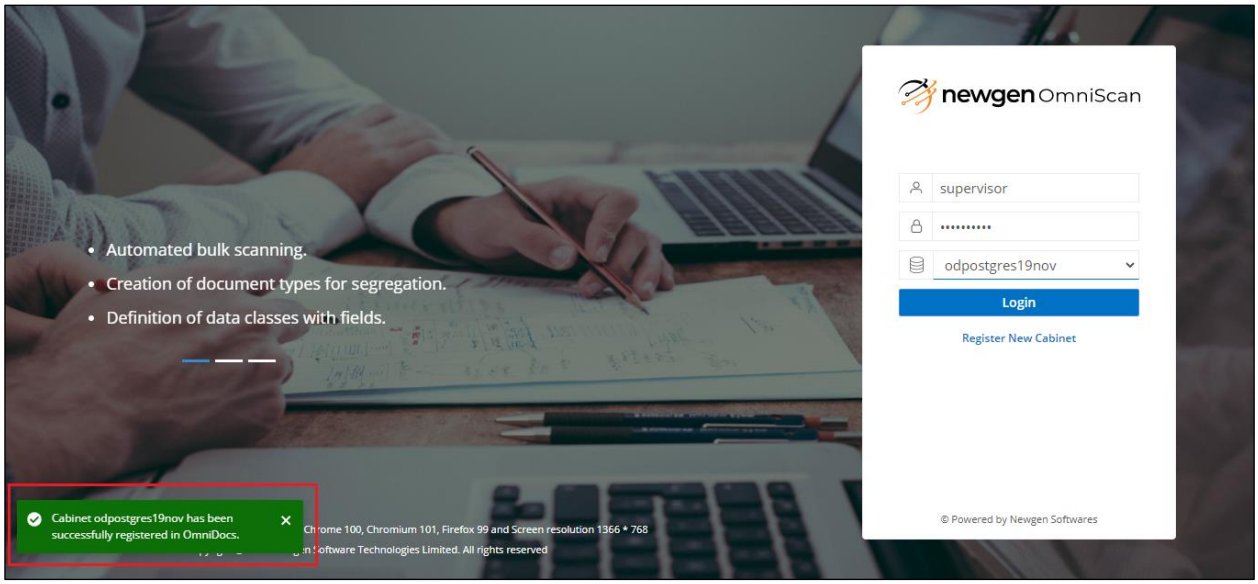

Figure 3.95

The registered cabinet appears in the **Cabinet Name** list on the login screen. Now you can log into OmniScan Web.

NOTE:
Ensure that the **OmniScan_Template_Repository** folder is already created in OmniDocs before logging into OmniScan Web.

---

4 Configuration of Azure DevOps release pipeline

This chapter describes the configuration of Azure DevOps Release Pipeline. Refer the below sections for procedural details.

4.1 Overview

The Build Pipeline and Release Pipeline are separated into two parts. Build Pipeline is done through the Jenkins server which can be installed on an on-premises machine or a cloud machine. Using the Azure DevOps Release Pipeline cloud service, you can manage the Release pipeline. In this architecture, three stages are created that is, Dev, UAT, and Production and in each stage, deployment is quite different. You can have some more stages depending on the requirements. This guide describes the configuration of the Azure DevOps Release Pipeline for container deployment on Azure Kubernetes Service (AKS).

4.2 CICD pipeline architecture

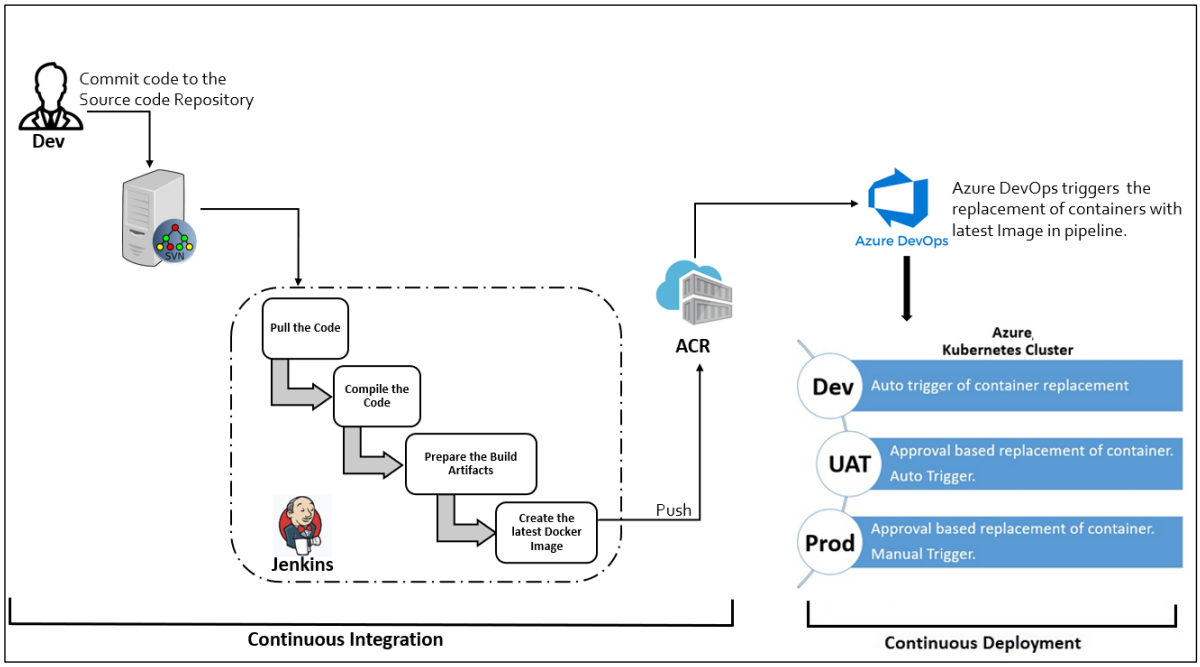

Figure 4.1

1. The Newgen representative builds the product's base Docker images on the company's on-premises servers using Jenkins.
2. As soon as the Dev team commits the code to the source code repository, the Jenkins pipeline gets triggered. It pulls the code then compiles them and prepares the build artifacts as well as creates Docker images and pushes the newly created Docker images to the Azure Container Registry.
3. As soon as any Docker image is pushed to the Azure Container Registry, Azure DevOps Release Pipeline triggers the deployment to the Dev environment. Here, you can configure the performance testing as well as security testing of the application. In Addition, you can perform manual testing as required.
4. UAT and Production deployments are based on approval and are available on-demand. To deploy to the UAT environment, you need to trigger the UAT deployment. Upon deployment trigger, an approval mail is sent to the project manager or the concerned team. As soon as the project manager approves the go-ahead, UAT deployment gets started automatically.
5. Production deployment is also based on approval, but it is multi-level approval. To deploy a production environment, you require the approval of all stakeholders, and the production environment doesn't get triggered automatically on receiving all the approvals. A manual intervention mail is sent to the engineer who is supposed to deploy to production with a checklist. During deployment, all the checklist points get verified before performing the

production deployment. In case any point of the checklist is not covered, then deployment to the production gets rejected.

4.3 Configuration of Azure DevOps

Perform the below steps to configure Azure DevOps:

1. Sign in to the Azure DevOps portal at https://azure.microsoft.com/en-in/services/devops/
2. After a successful sign in, click **New Project** to create a new project.

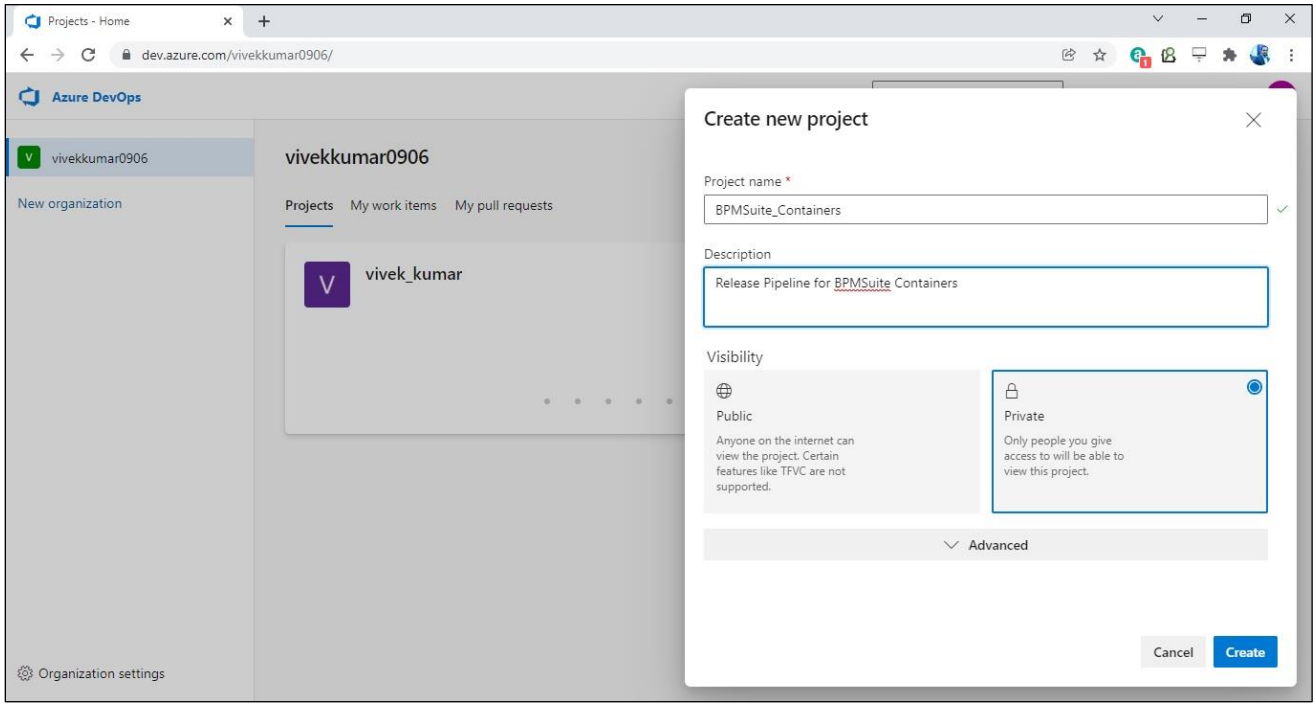

Figure 4.2

3. Specify the **Project name**, and **Description.**
4. Select the **Visibility** as **Private** and create the Azure DevOps Release Pipeline for different Docker Images.

4.3.1 Configuration of release pipeline

This section explains how to create Release Pipeline.

NOTE:

Refer the following steps to configure the Release Pipeline for the Docker Images.

- OmniDocsWeb
- OmniDocsWeb_Services
- OmniDocsEJB
- OmniDocsServices
- EasySearch
- TEM
- OmniScanWeb7.0
- OmniDocsWOPI

Perform the below steps to create Release Pipeline:

1. After project creation, the project summary screen appears. Hover over the **Repos** and select **Files**.

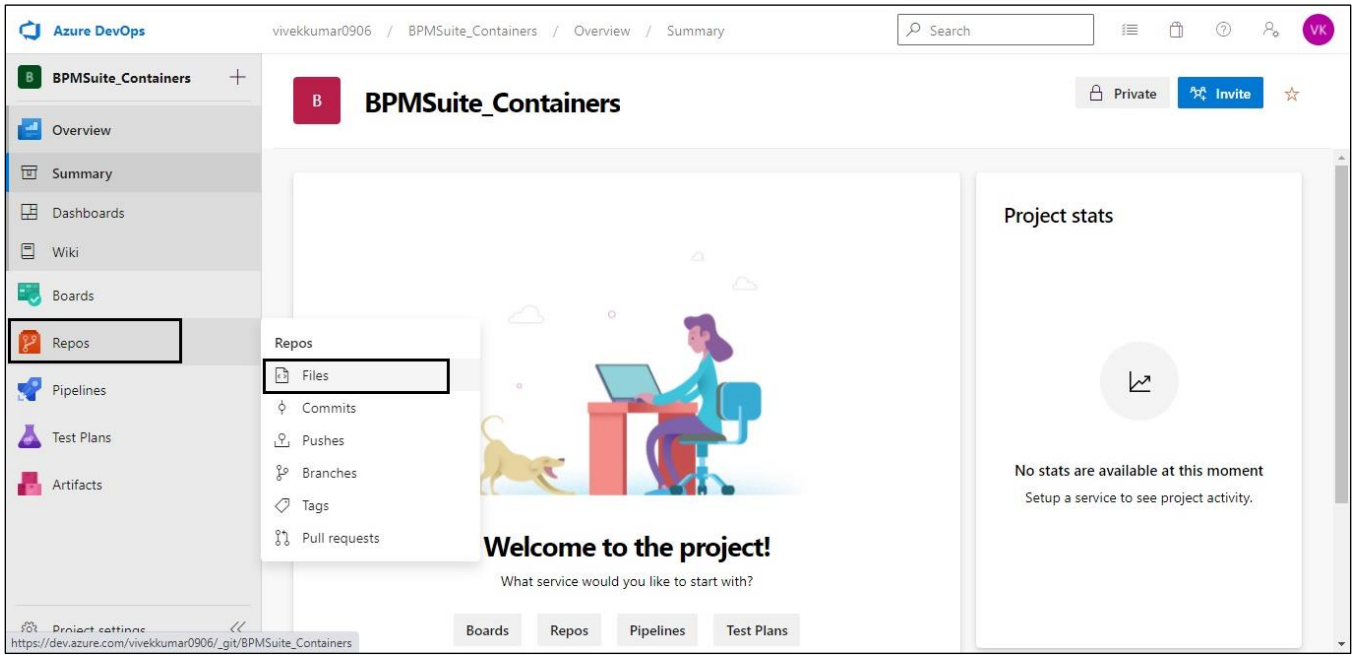

Figure 4.3

2. Click **Initialize**.

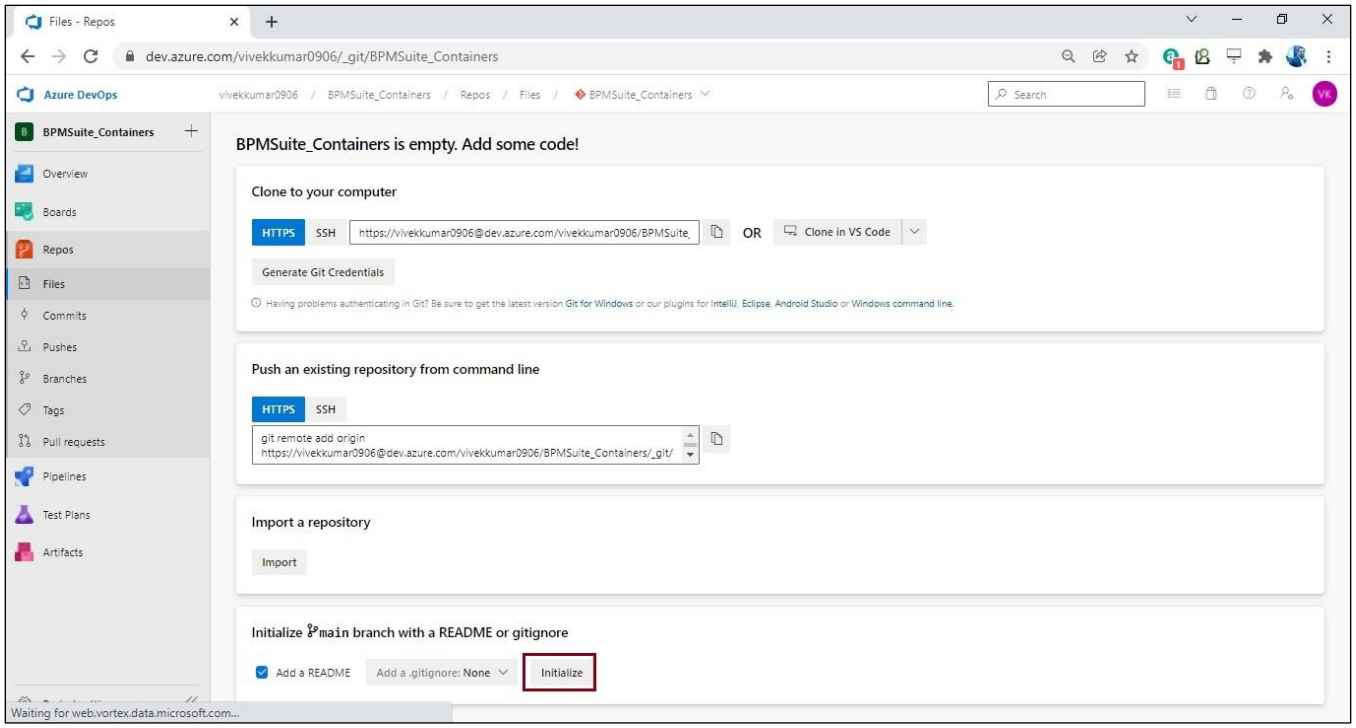

Figure 4.4

3. Click **More actions** and then select the **Upload file(s)**.

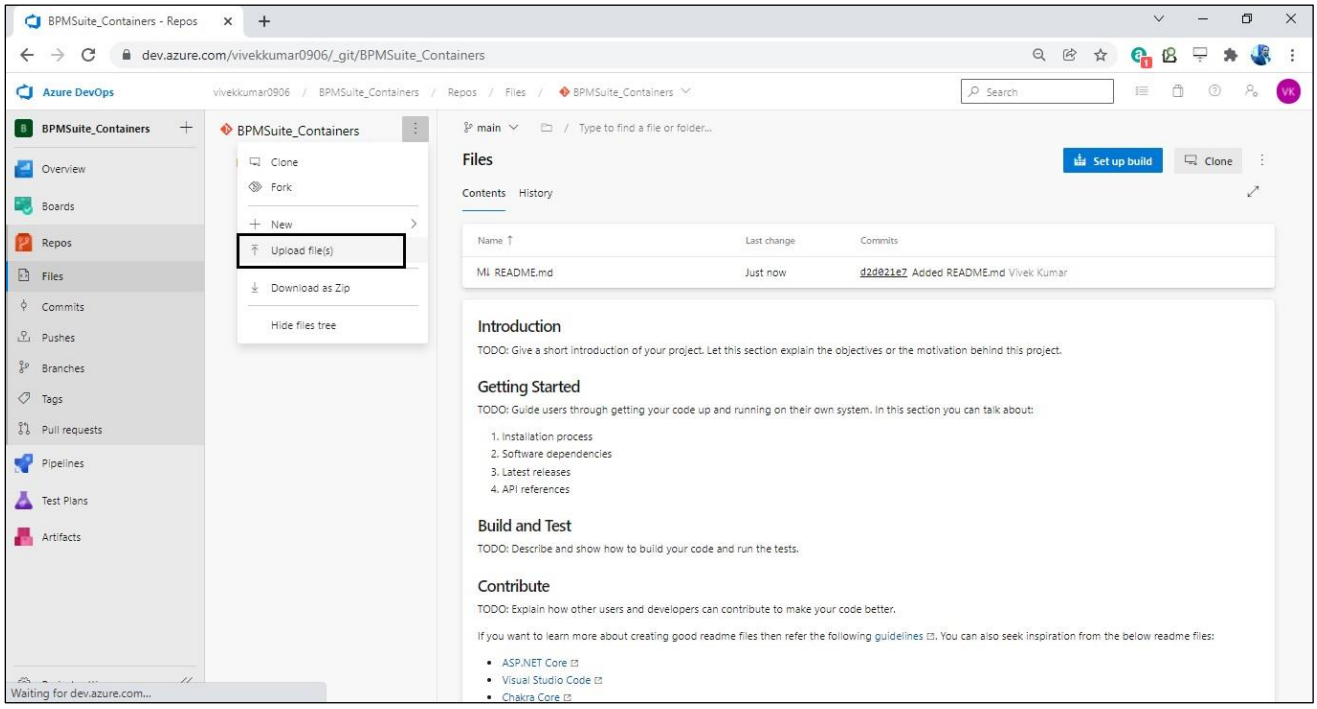

Figure 4.5

4. Browse or drag and drop all the YAML files that have shared and then select **Commit**.

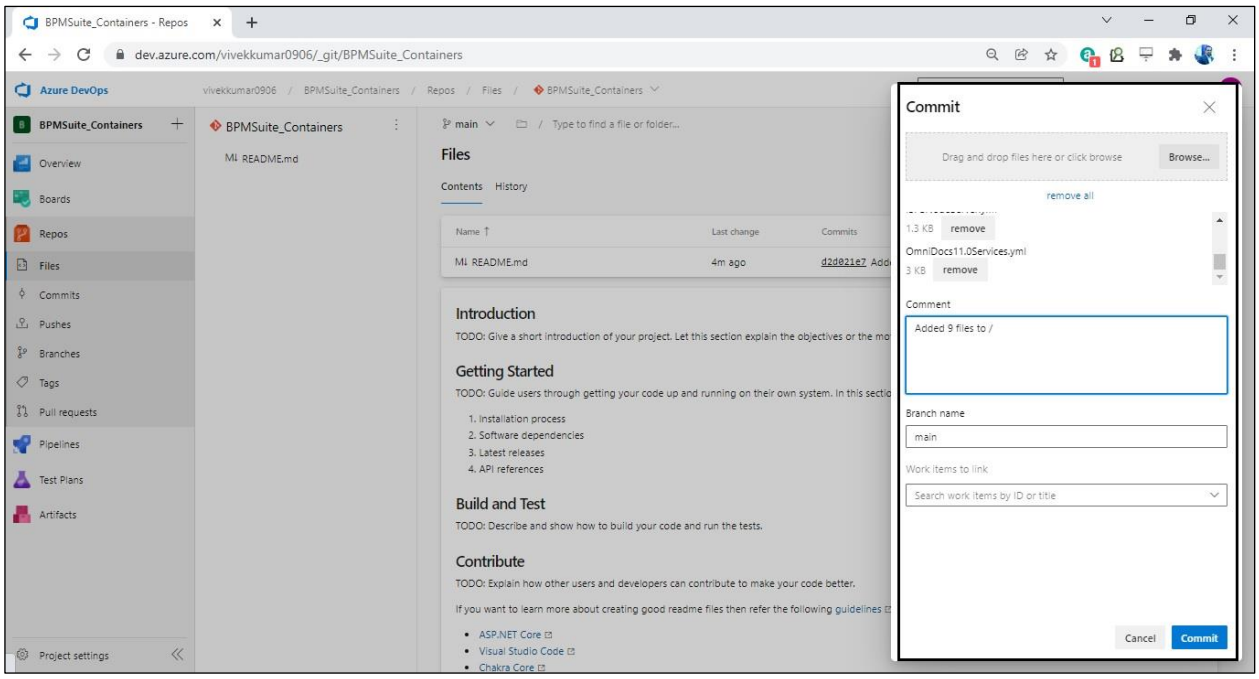

Figure 4.6

5. Hover over to the **Pipelines** in the left panel and select **Releases**.

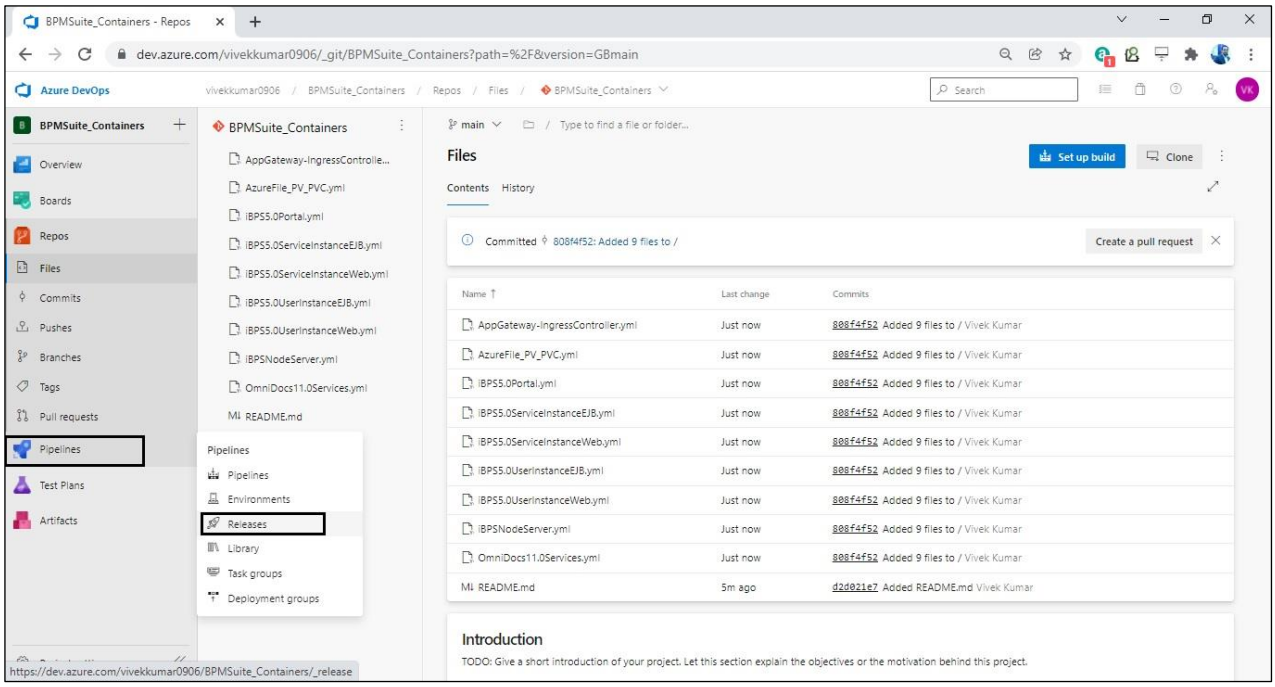

Figure 4.7

6. Click **New Pipeline** button. **Select a template** dialog appears.
7. Select the **Deploy to a Kubernetes cluster** template.

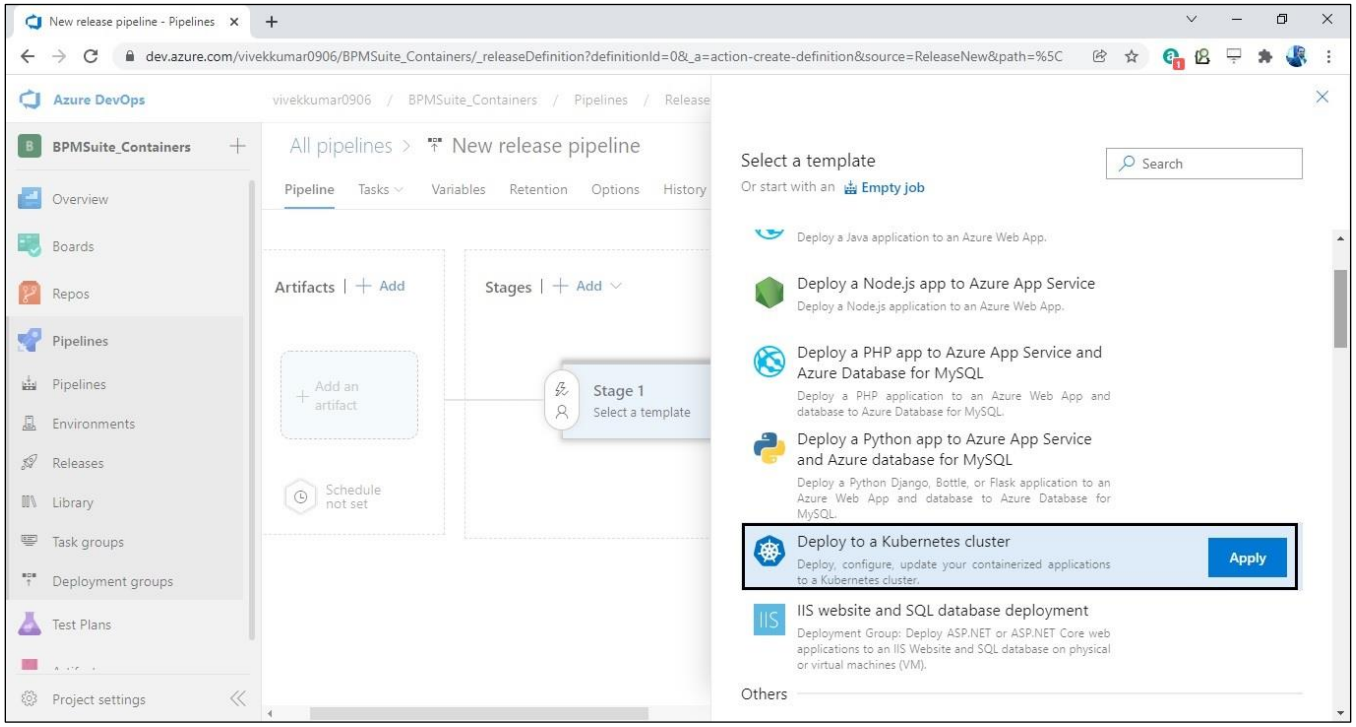

Figure 4.8

8. Click **Apply**. The **Stage** panel appears.
9. Specify the **Stage name** and click **close** icon to close the dialog.

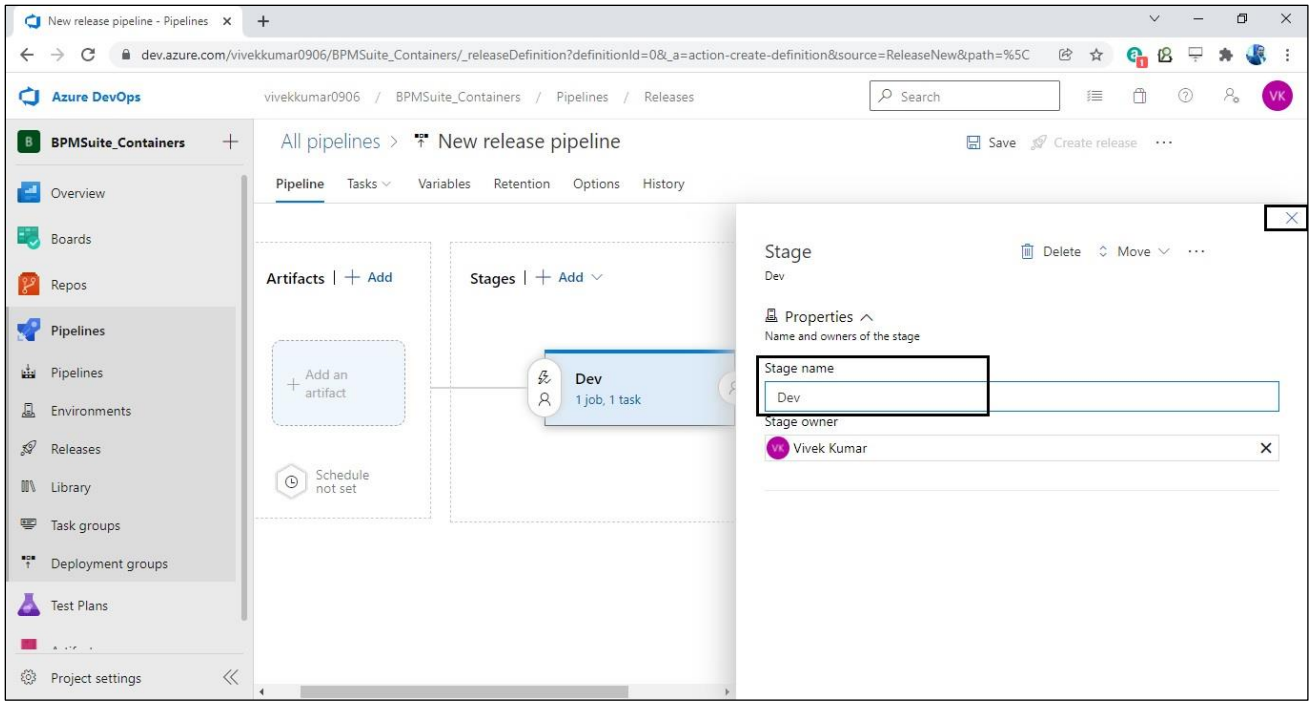

Figure 4.9

10. Enter the unique name for your pipeline and click **Save**.

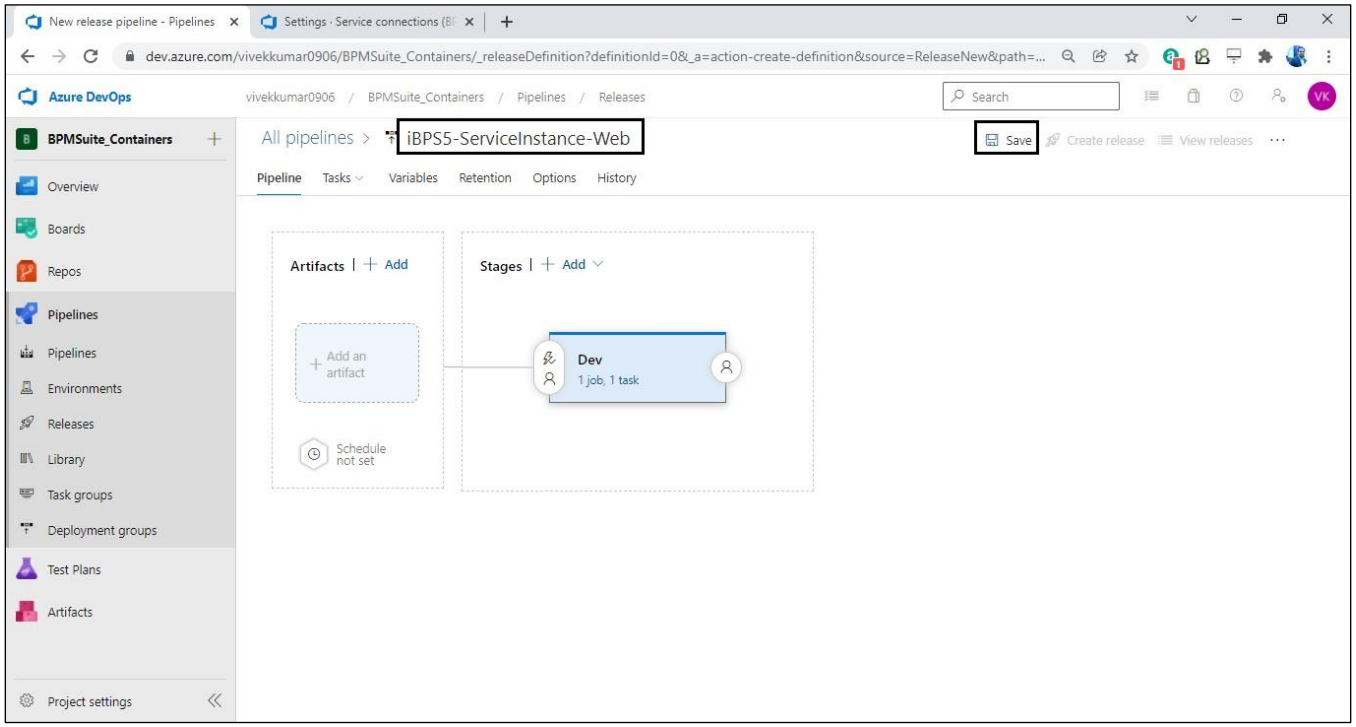

Figure 4.10

11. Specify **Comment** and click **OK** on the **Save** dialog.

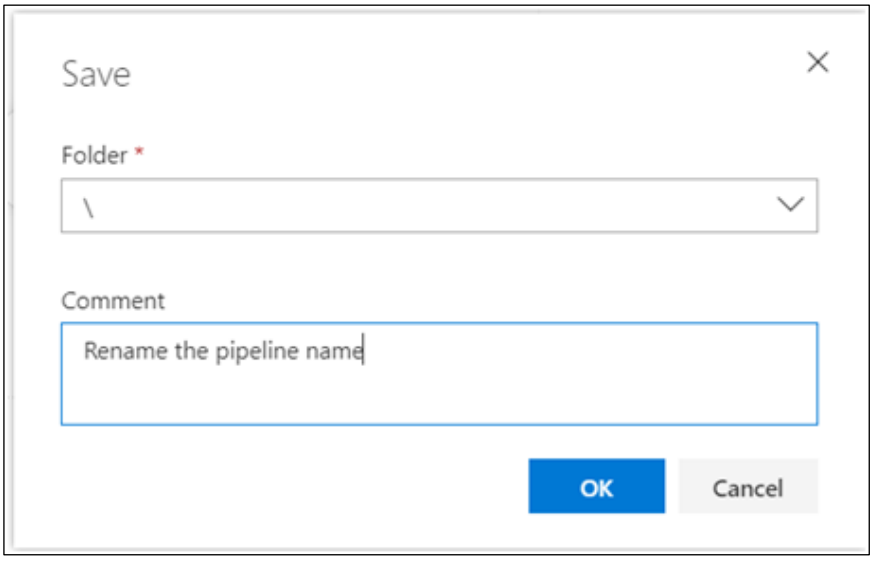

Figure 4.11

12. Click **Add an artifact**. The **Add an artifact** dialog appears.
13. Click **Azure Container Registry** under the **Source** type.
14. Select the **Service connection** which authenticates the Azure Container Registry.
15. In case **Service connection** is not created, follow the below steps to create Service connection:

    Configuration of Service connection for Azure Container Registry:

    - Click **Manage** link. The **Create service connection** page appears in a new tab.
    - Click **Create service connection**. The New service connection dialog appears.
    - Select **Azure Resource Manager** as the connection type and click **Next**.
    - Select **Service principle (automatic)** as the **Authentication method.**
    - Specify the following parameters:
        - **Subscription** as Scope level.
        - Select an existing **Azure subscription**.
        - Select the **Resource Group** in which Azure Container Registry is created.
        - (Optional) Specify the **Service connection** name and **Description**.
        - Select the checkbox **Grant access permission to all pipelines.**
    - Click **Save**. Once the service connection is created, it appears in the list.

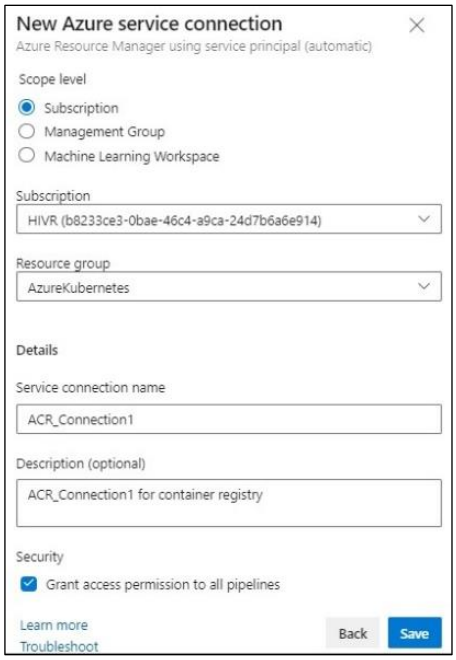

Figure 4.12

16. If the **Service connection** is already created, then select the created service connection.
17. Select **Resource Group** from the list in which Azure Container Registry is created.
18. Select the created **Azure Container Registry**.
19. Select a Docker image for example, **ibps5serviceinstanceweb** as a **Repository**.
20. Select **Latest** as the **Default version.** Leave the **Source alias** with its default value and click **Add**.

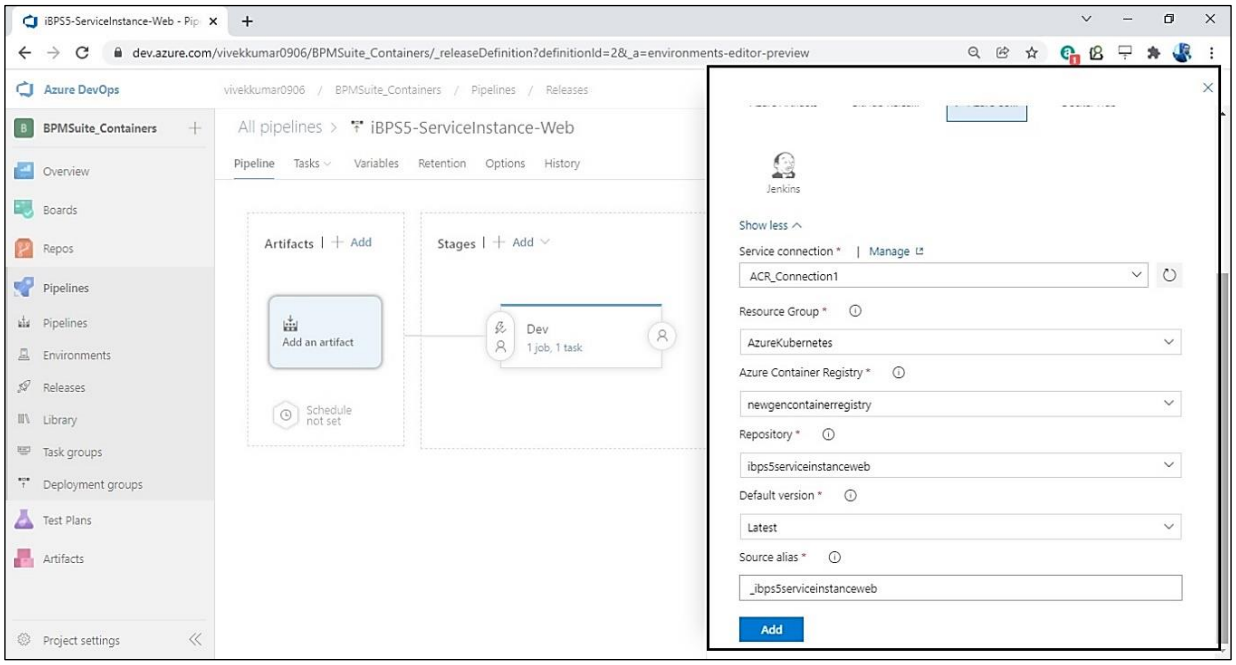

Figure 4.13

21. Once the artifact is added, it appears in the **Artifacts**. Click **Continuous deployment trigger** icon. The **Continuous deployment trigger** dialog appears.

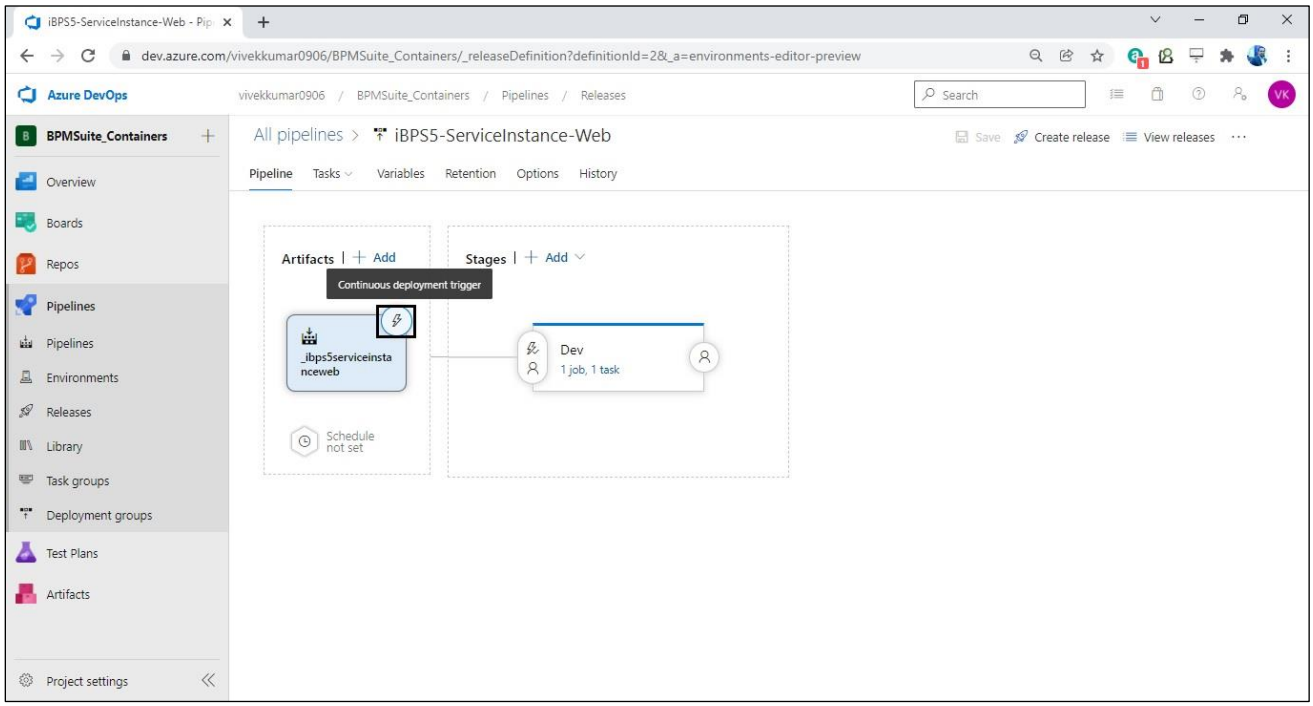

Figure 4.14

22. Enable the **Trigger** and specify the **Tag filter**.
    For example,
    ^latest$ - trigger the release only if the tag is "latest"
    v1\.[0-9] - trigger the release for tags like "v1.23", "beta-v1.3-test"

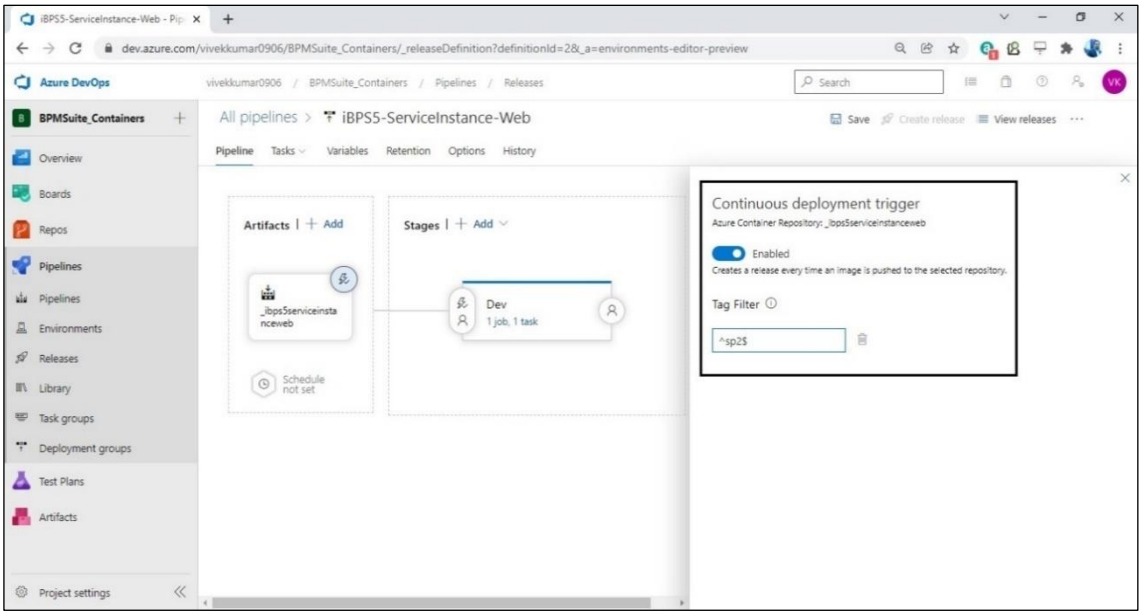

Figure 4.15

23. Click Close icon to close the Continuous deployment trigger dialog**.**
24. Click **Save**.
25. Click **Add an artifact**. The **Add an artifact** dialog appears.
26. Click **Azure Repos** under the Source type.
27. Select the **project**, **Source (repository)** and default branch **main.** Also, keep the other settings as default.

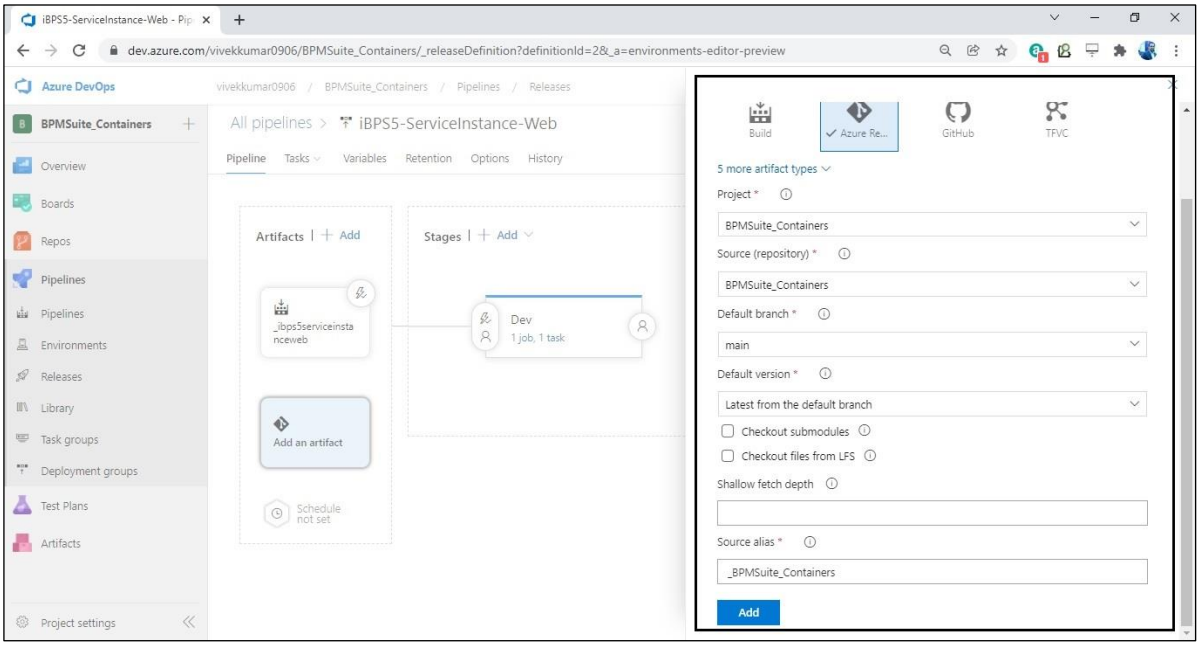

Figure 4.16

28. Click **Add** then **Save**.
29. Configure three stages: Dev, UAT, and Production, and on each stage deployment process is different. You can have some more stages depending on the requirements.

4.3.2 Configuration of Dev stage

Perform the below steps to configure the Dev Stage:

1. Click **View stage tasks**.

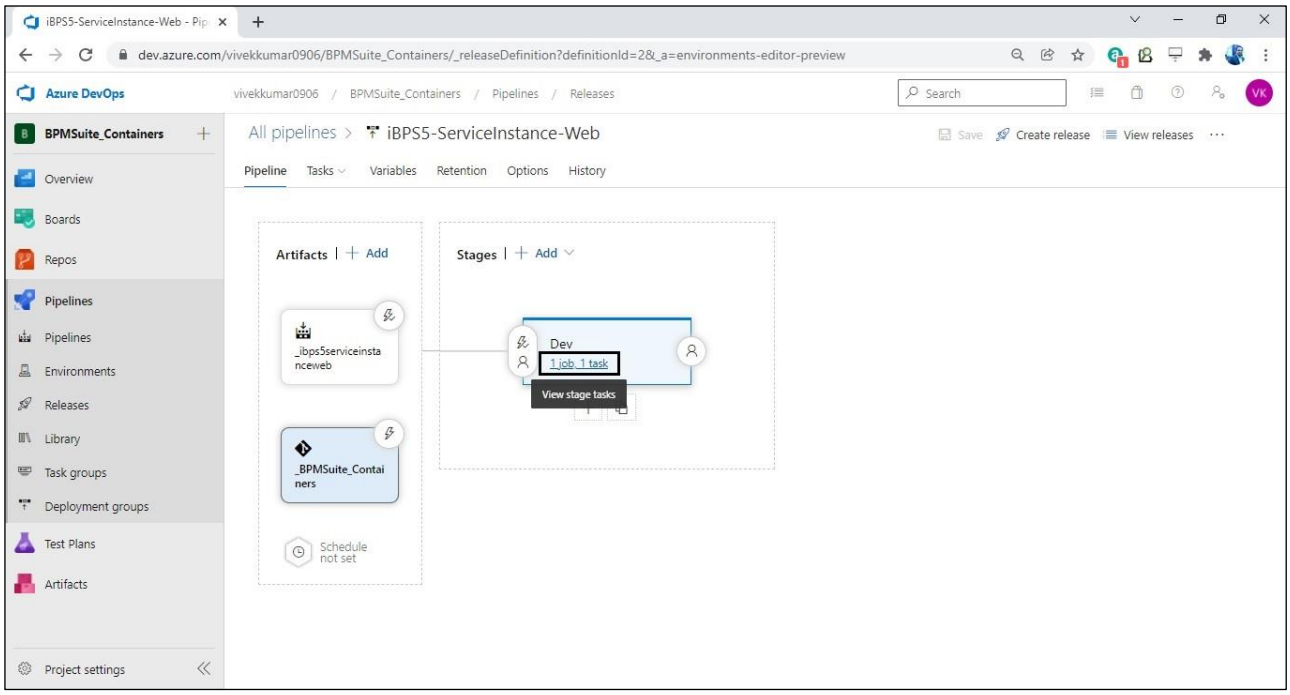

Figure 4.17

2. Click **Agent Job** and then select **ubuntu** as the **Agent Specification**.

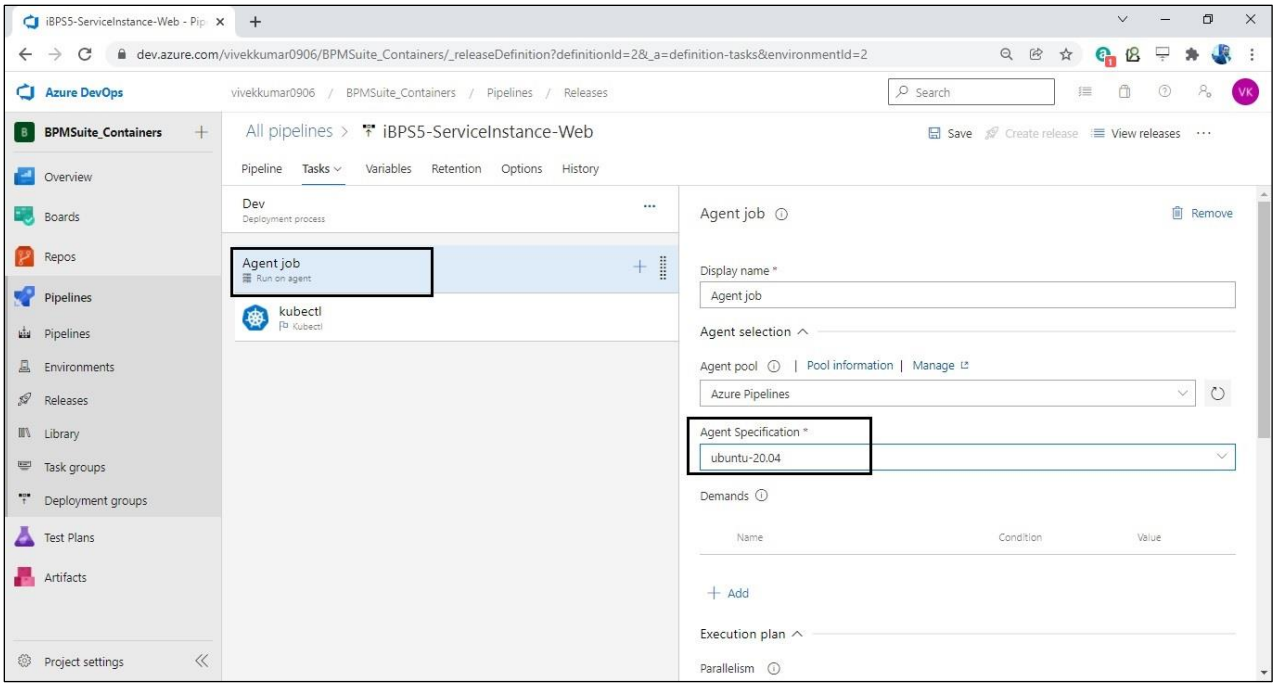

Figure 4.18

3. Click **Add a task to Agent Job +** icon and search for the **Replace Tokens** and add them.

NOTE:

Ensure **Replace Tokens** task must be the 1st task under the Agent Job.

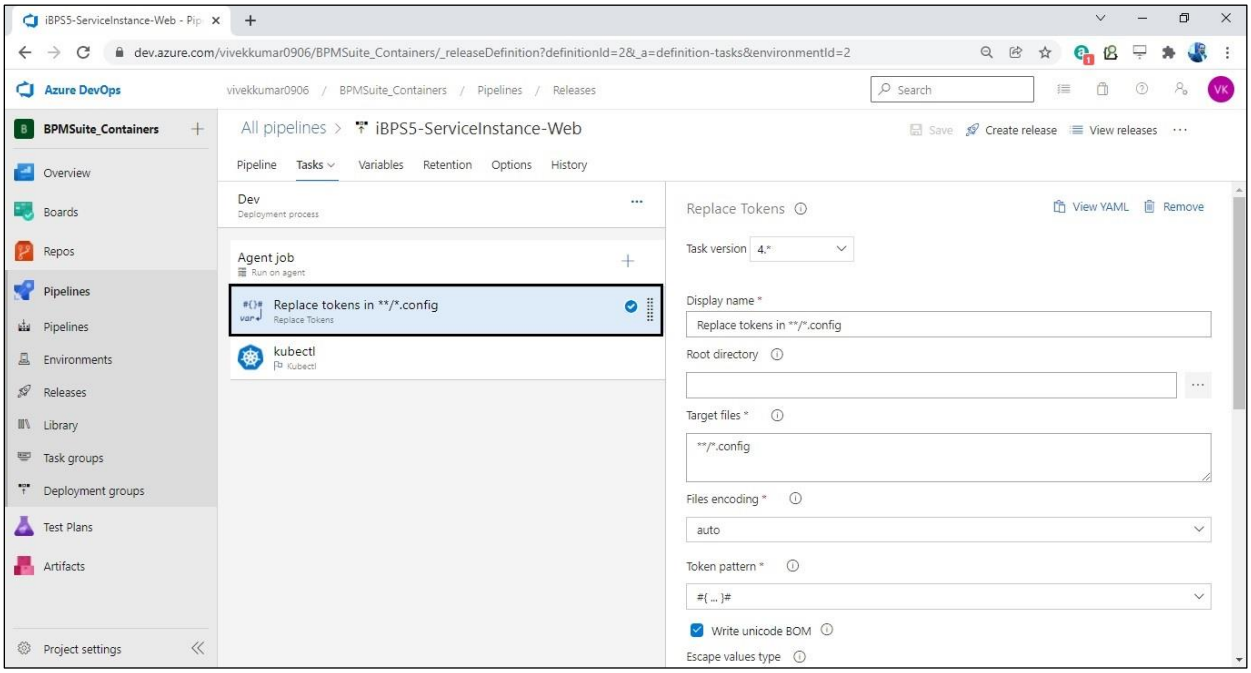

Figure 4.19

4. Click **Browse Root Directory** icon under the Replace Token settings.
5. Select appropriate YAML file (for example, *iBPS5.0ServiceInstanceWeb.yml*).
6. Copy the content of the **Root directory** and paste it to the **Target files** textbox.
7. Leave the other settings as default and click **Save.**

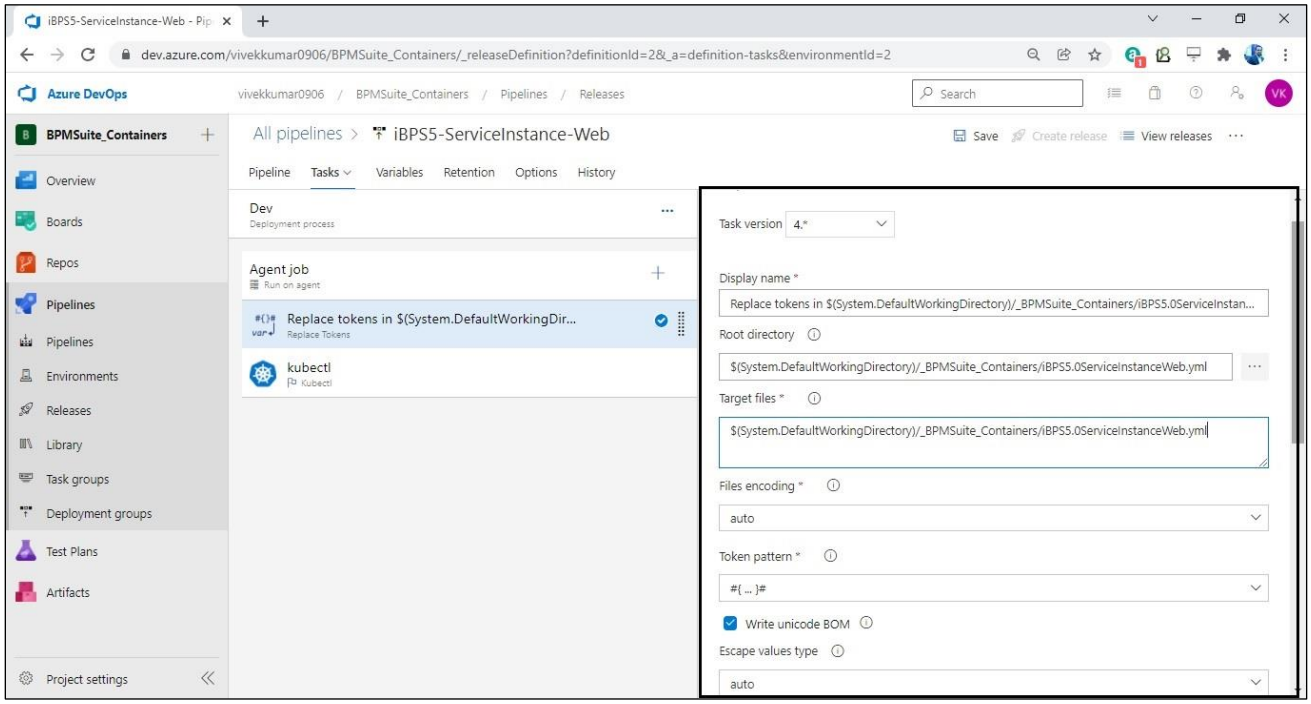

Figure 4.20

8. Click **Kubectl** task under the **Agent Job**.
9. Select Task version **1** and specify the **Display name**.
10. Select **Kubernetes Service Connection** as the Service connection type.
11. Select **Kubernetes service connection** which authenticates kubectl to interact with the Kubernetes cluster.
12. If **Kubernetes service connection** is not created, then follow the below step to create Kubernetes service connection.
13. **Configuration of Kubernetes service connection.**
    - Click **Manage** link. The Service connections page appears in a new tab.
    - Click **New service connection** or **Create service connection**. The New service connection dialog appears.
    - Select **Kubernetes** and click **Next**. The New Kubernetes service connection dialog appears.
    - Select **KubeConfig** as an **Authentication method**.
    - Copy the content of *KubeConfig* file.

NOTE:

You can get the KubeConfig file by executing below command:

```
az aks get-credentials --resource-group <ResourceGroupName> --name
<AzureEKSClusterName>
```

For example,

```
az aks get-credentials --resource-group AzureKubernetes --name BPMSuite-
AKSCluster
```

- Select an existing Azure Kubernetes cluster for example, BPMSuite_AKSCluster
- Specify the **Service connection name** and **Description**.
- Select the checkbox **Grant access permission to all pipelines** and click **Verify** and **Save**. Once the Service connection is created, it appears in the list.

Figure 4.21

14. If **Kubernetes service connection** is already created, then select the created connection.
15. Select the Namespace, that is, **dev.**
16. Select **Apply** command using the **Command** dropdown.

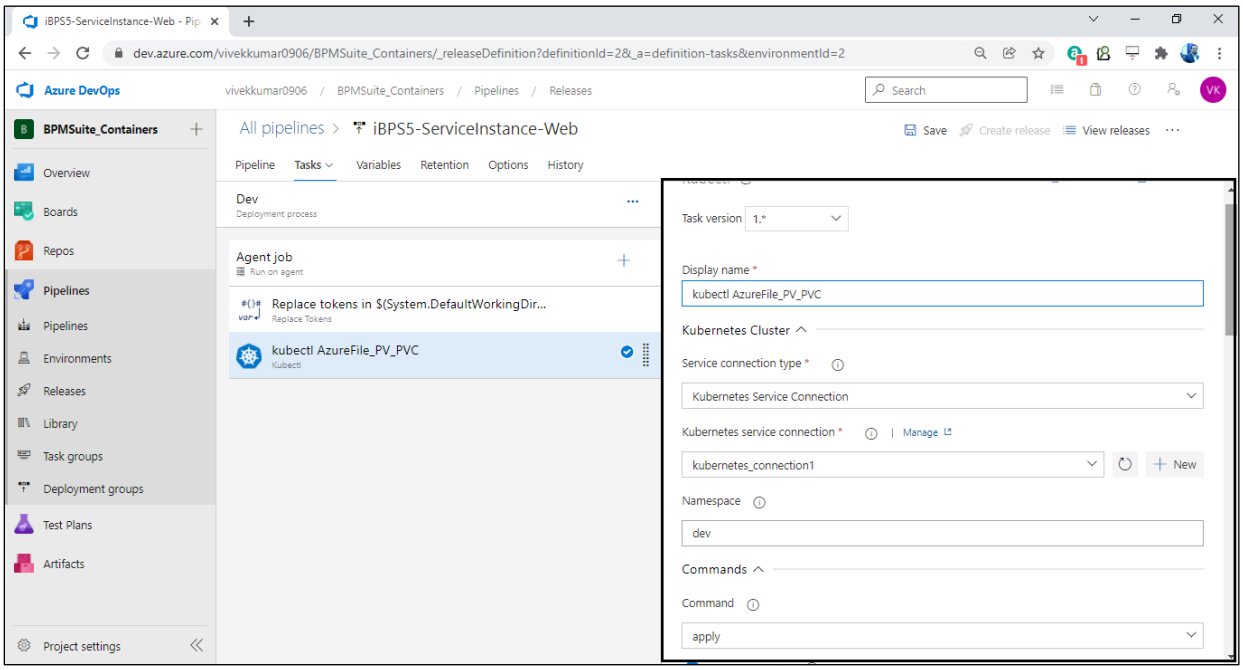

Figure 4.22

17. Select the checkbox **Use configuration**.
18. Select the radio button **File path**.
19. Browse the *AzureFile_PV_PVC.yml* file path from the **Azure Repos**.

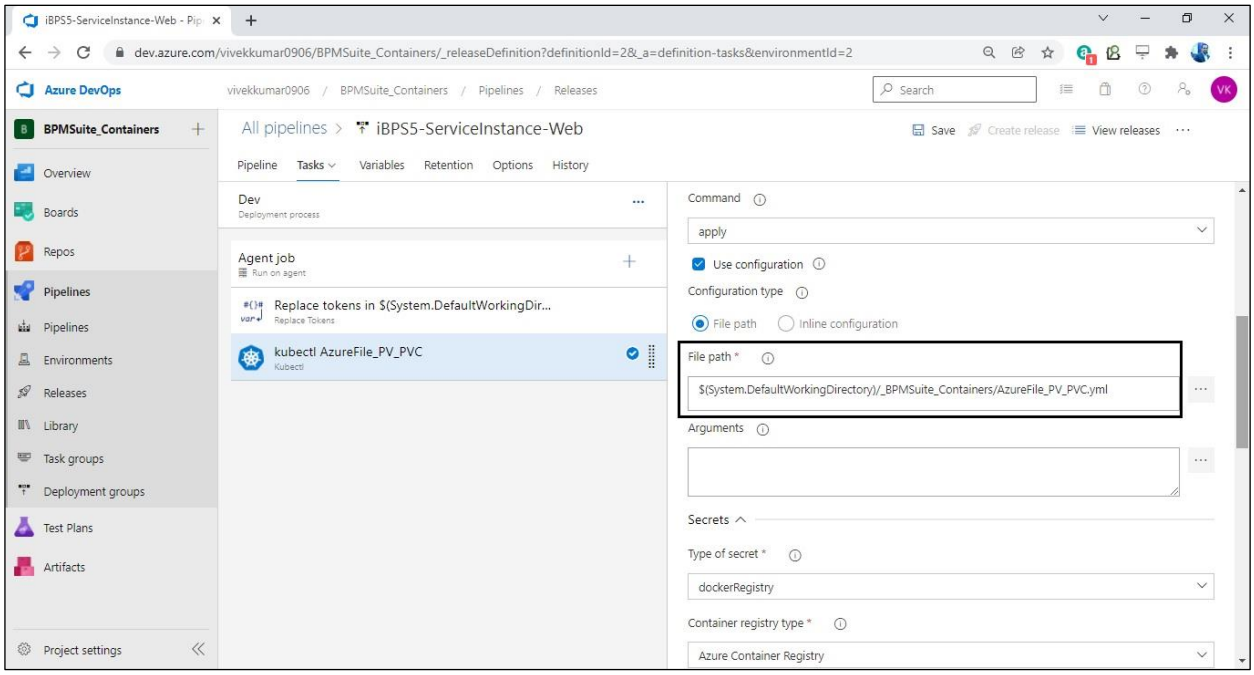

Figure 4.23

20. Expand the **Advanced** tree structure.
21. Select the **Check for latest version** checkbox.
22. Right click added kubectl task and select **Clone task(s)**.

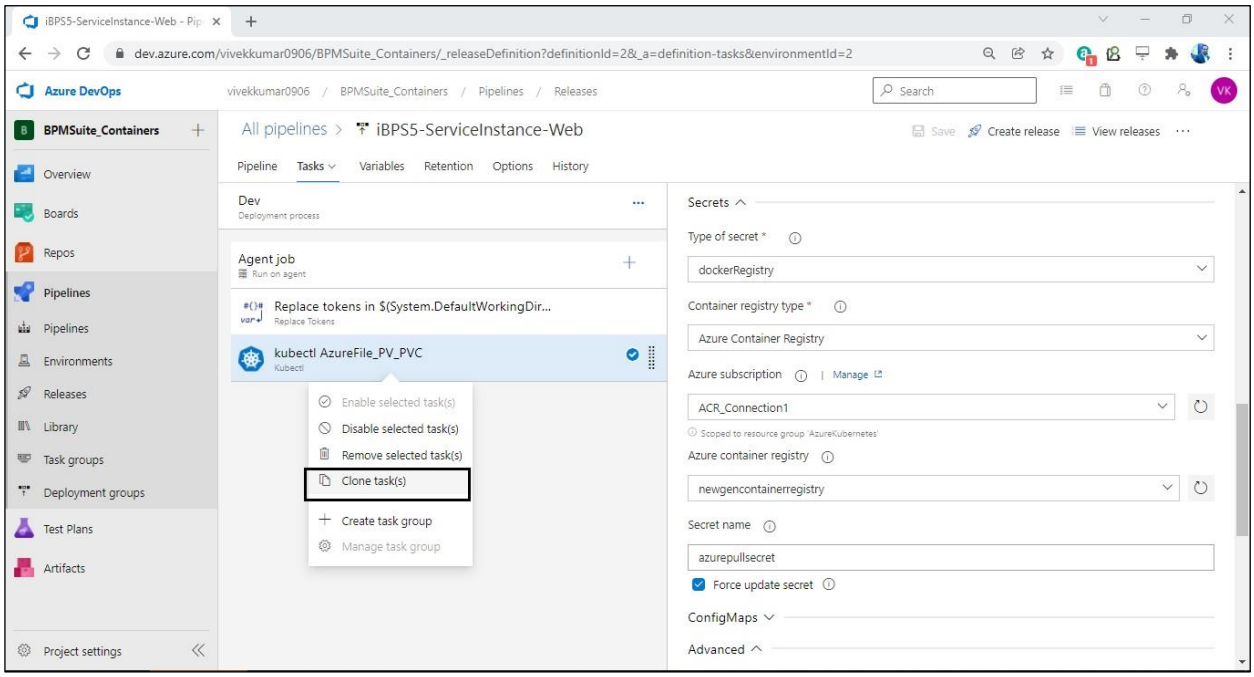

Figure 4.24

23. Change the **Display name** of newly cloned task.
24. Browse the *yml* file (for example, **iBPS5.0ServiceInstanceWeb.yml**) path from the **Azure Repos**.

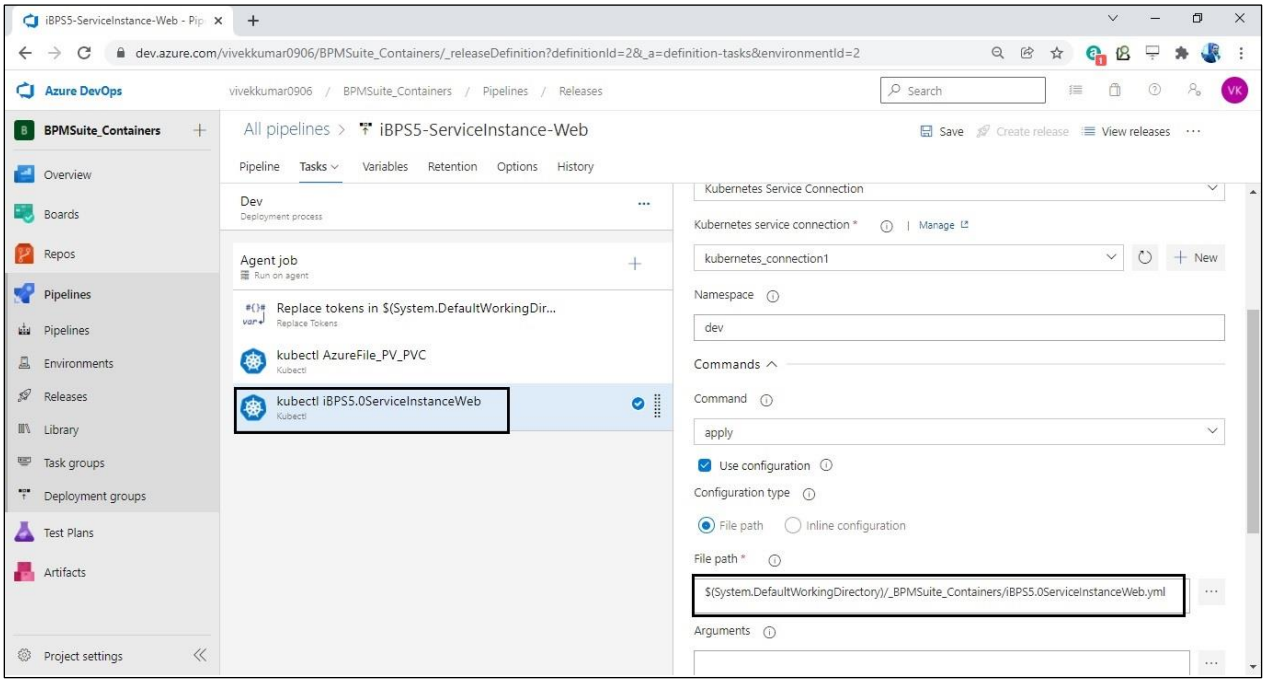

Figure 4.25

25. Expand the Secrets tree structure.
26. Select **dockerRegistry** as a **Type of secret.**
27. Select **Azure Container Registry** (ACR) as a **Container registry type**.
28. Select the created **Azure service connection** for ACR.
29. Select the created **Azure container registry**.
30. Specify the **secret name** such as **azurepullsecret.**
31. Select the **Force update secret** checkbox.

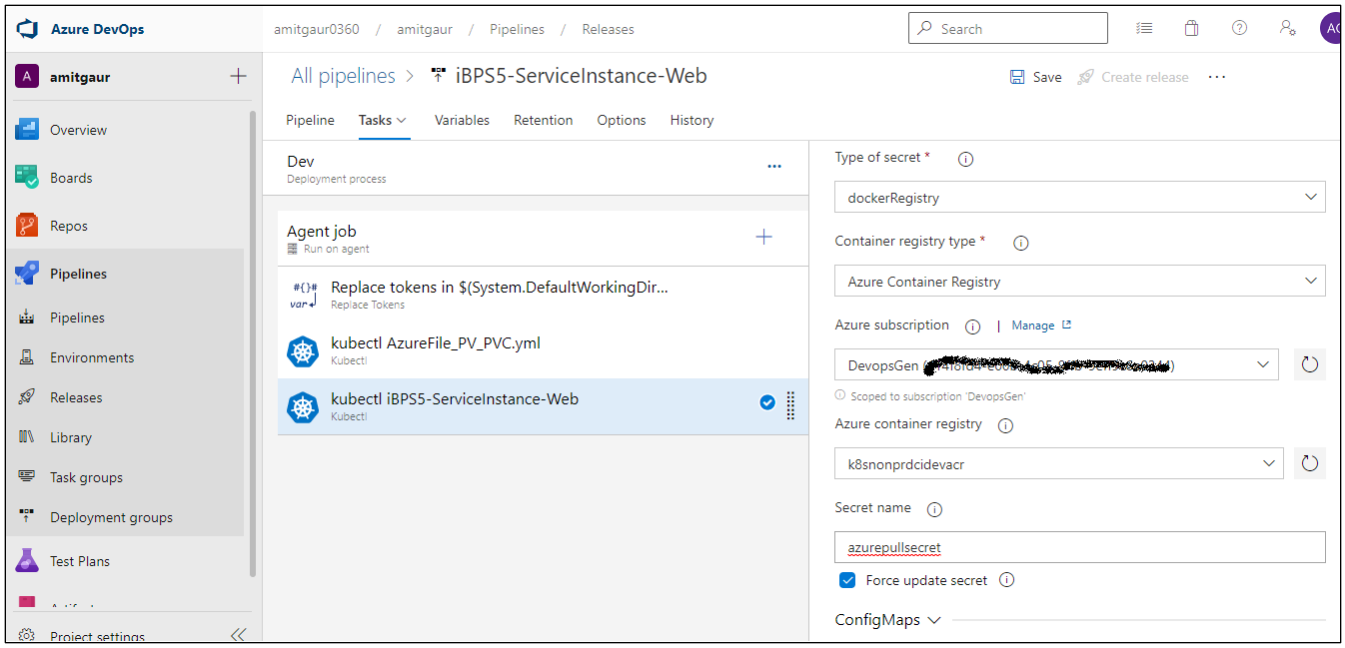

Figure 4.26

32. Right click cloned kubectl task and Select **Clone task(s)**.
33. You can change the **Display name** of newly cloned task.
34. Browse the *AppGateway-IngressController.yml* file path from the **Azure Repos**.

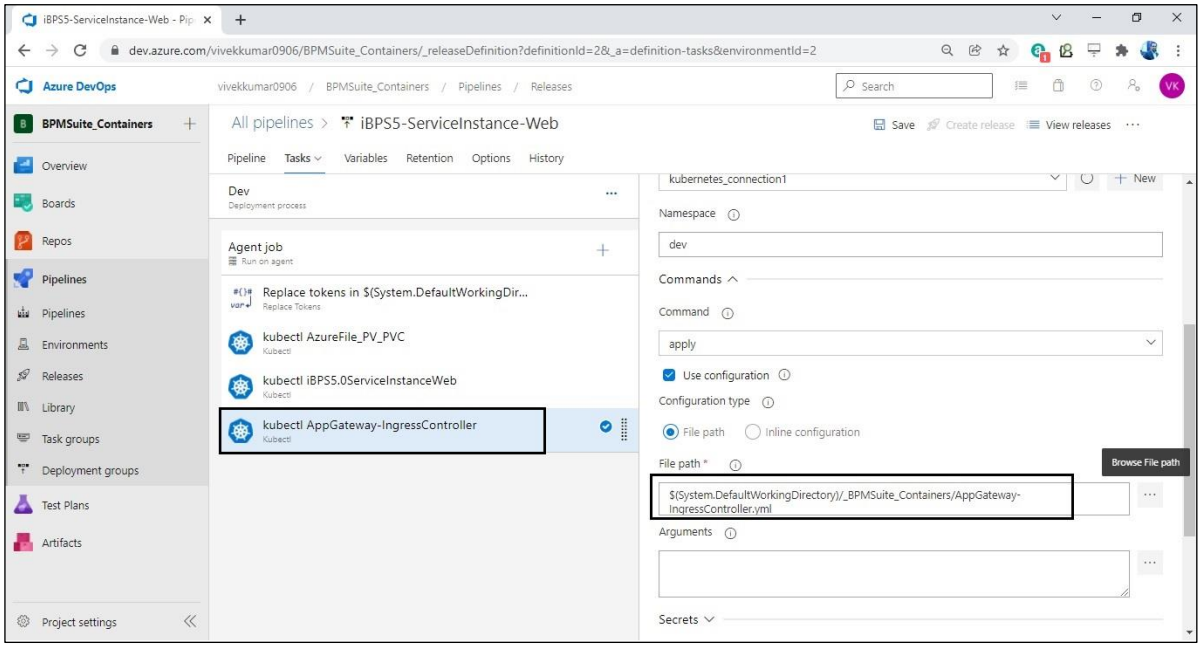

Figure 4.27

35. Click **Save**. Now, as soon as any Docker Image is pushed to the Azure container registry with the tag name **sp2**, Azure DevOps trigger the deployment to the **Dev Stage**.

4.3.3 Configuration of UAT stage

Perform the below steps to configure the UAT Stage:

1. UAT deployments are approval based and they are available on-demand. Once you are ready to deploy to the UAT environment, you just need to trigger the UAT deployment. When you trigger that deployment, an approval mail is sent to the project manager or the concerned team. As soon as the approval is provided for the go-ahead, the UAT deployment starts automatically.
2. Go to the **Pipeline** tab of the Release Pipeline for which **Dev stage** is configured (for example, **iBPS5-ServiceInstance-Web**).
3. Select **Dev stage** and click **Clone icon.** A cloned Stage gets created.

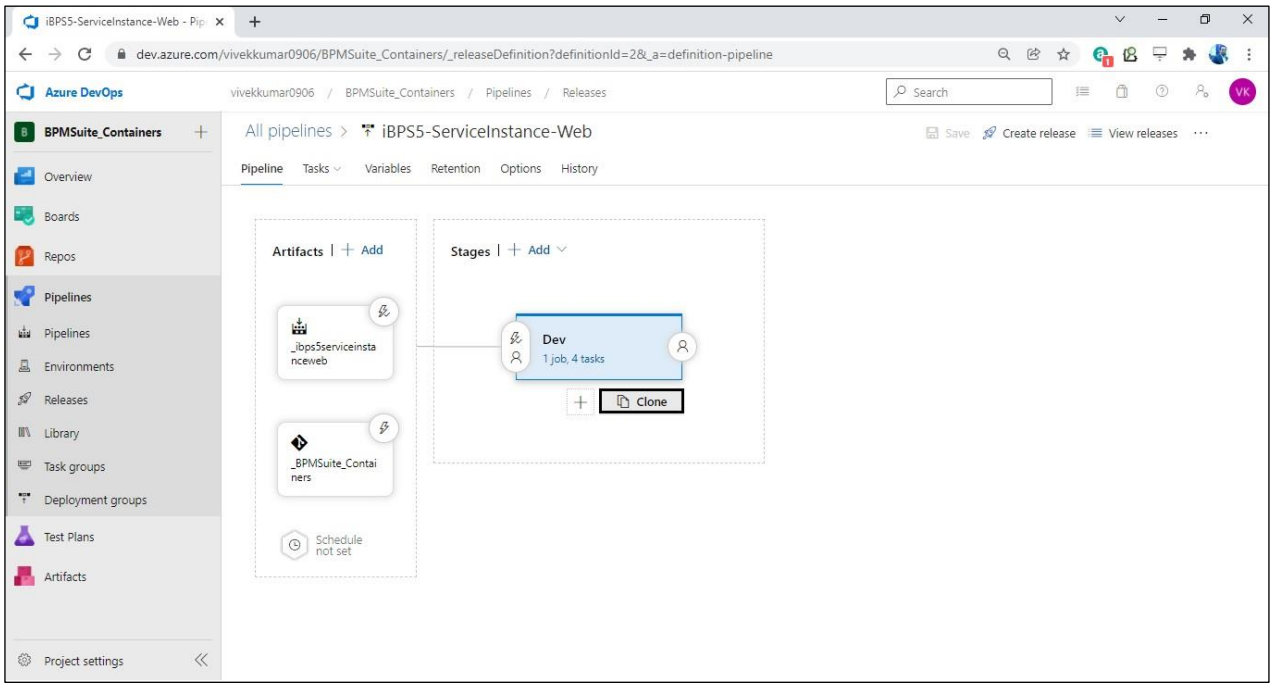

Figure 4.28

4. Specify the name of the cloned stage as **UAT** in the **Stage** panel.
5. Click **Pre-deployment conditions** icon of the UAT stage. The Pre-deployment conditions panel appears.

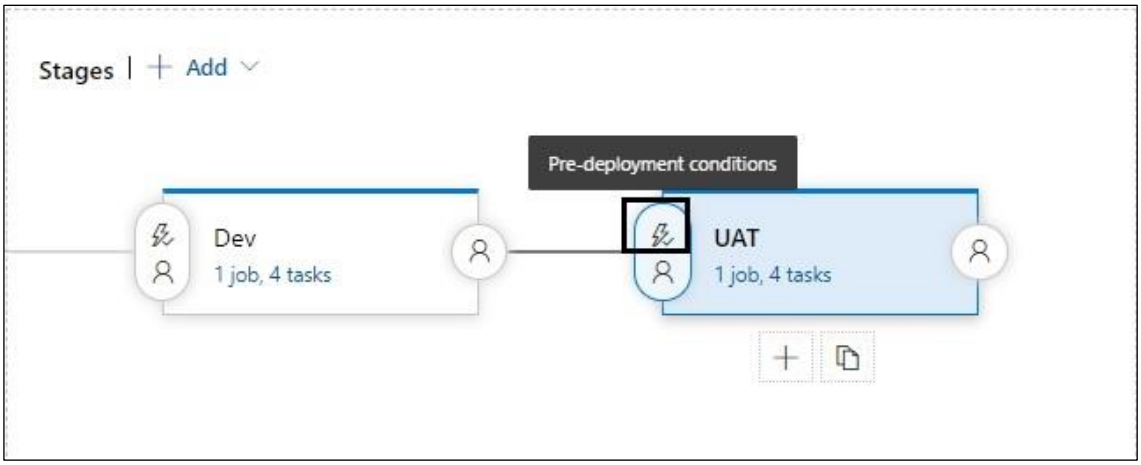

Figure 4.29

6. Select the **Manual Only** under the Triggers section.
7. As soon as the trigger type is changed from After **stage** to **Manual Only**, the UAT stage appears in parallel to Dev Stage instead of a series.

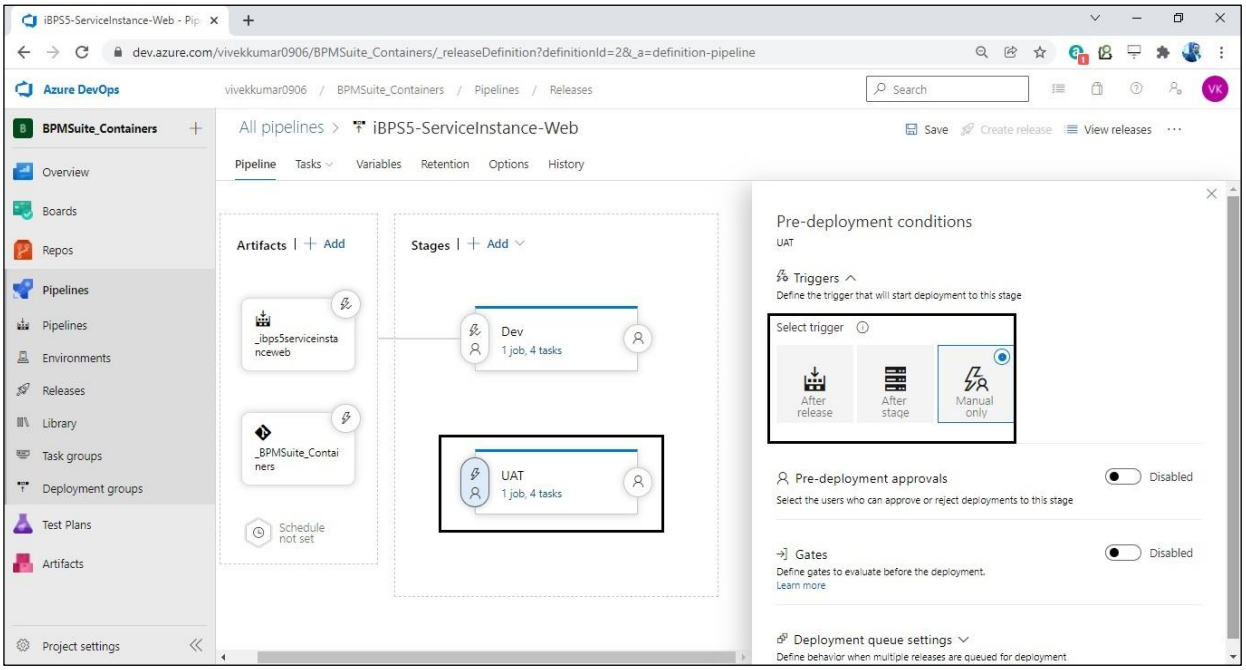

Figure 4.30

8. In the Pre-deployment conditions panel, enable the **Pre-deployment approvals**.
9. Select the list of users or groups who can approve or reject the deployment to this stage.
10. You can select users or groups by typing their names.
11. Select the **The user requesting a release or deployment should not approve it** checkbox in Approval policies.

12. Click **Close icon** to close the Pre-deployment conditions panel.
13. Click **Save** to save the changes.

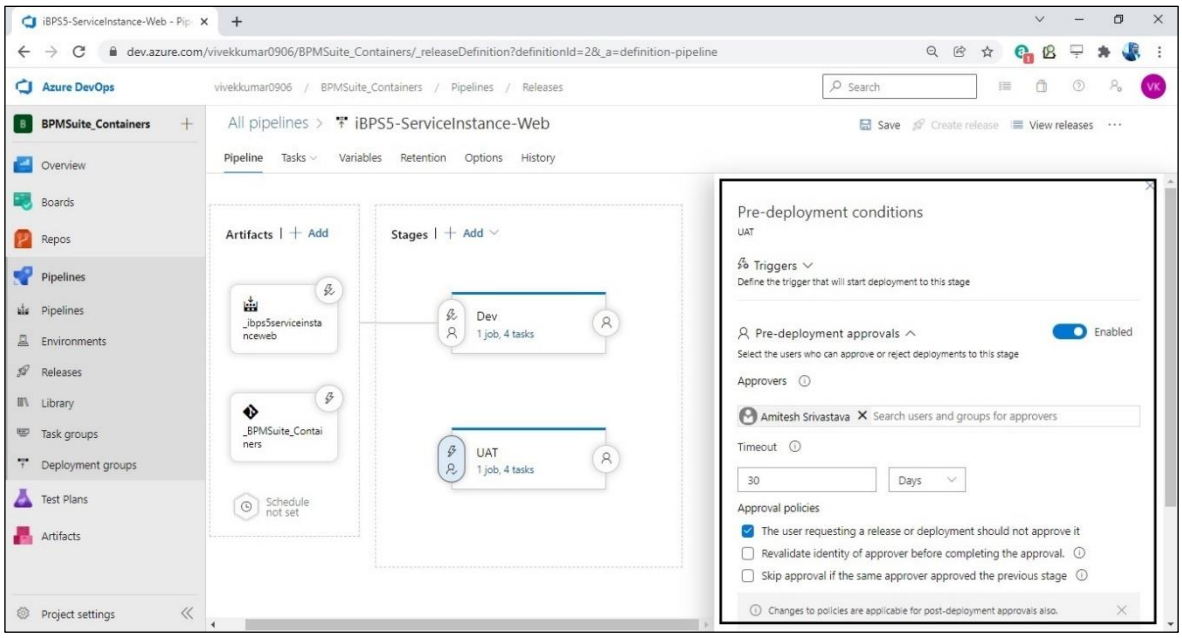

Figure 4.31

14. Click **View stage tasks** link of the UAT stage. Also, make the required changes in the UAT stage's tasks as per your requirements.
    For example, you can make the following changes in the below tasks:
    - **Kubectl Task:** Kubernetes service connection, Kubectl command, changes in YAML files, and so on.

4.3.4 Configuration of production stage

Production deployment is also based on approval, but it is multi-level approval. To deploy a production environment, you require the approval of all stakeholders, and the production environment doesn't get triggered automatically on receiving all the approvals. A manual intervention mail is sent to the engineer who is supposed to deploy to production with a checklist. During deployment, all the checklist points get verified before performing the production deployment. In case any point of the checklist is not covered, then deployment to the production gets rejected.

Perform the below steps to configure the Production Stage:

1. Go to the **Pipeline** tab of the Release Pipeline (for example, **iBPS5-ServiceInstance-Web**) for which **Dev and UAT stages** are just configured.
2. Select the **UAT stage** and click **Clone** icon**.** A cloned Stage gets created.

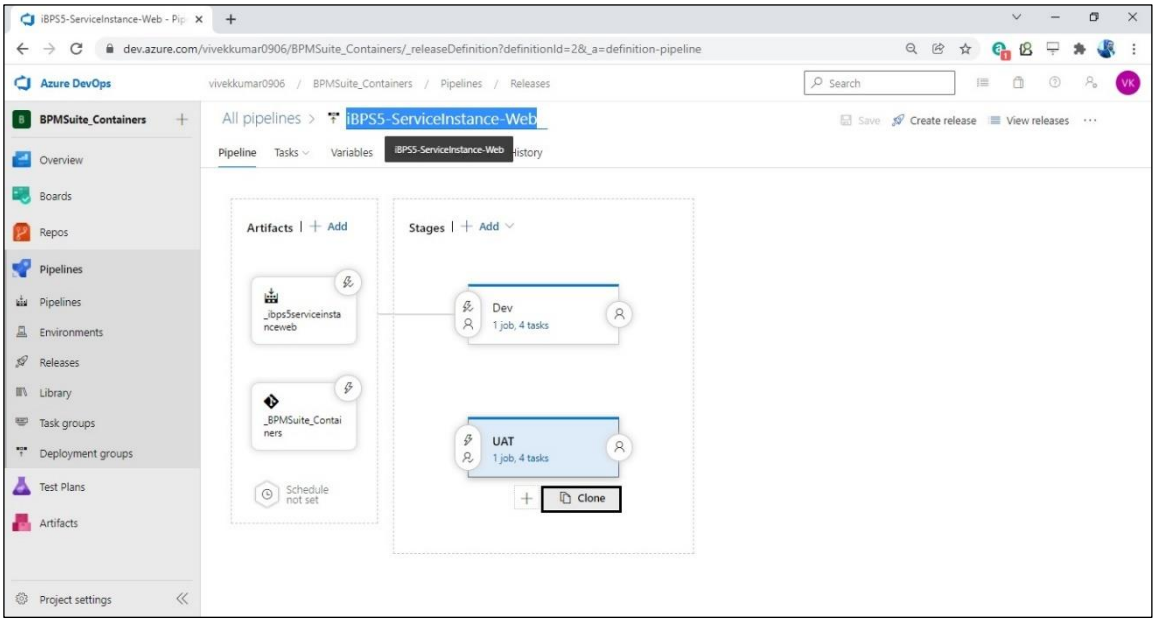

Figure 4.32

3. Specify the name of the cloned stage as **Production** in the **Stage** panel.
4. Click **Pre-deployment conditions** icon of the Production stage. The Pre-deployment conditions dialog appears.
5. Select **Manual Only** option under the Triggers section.
6. As soon the trigger type is changed from **After stage** to **Manual Only**, the Production stage appears in parallel to Dev and UAT stages instead of a series.

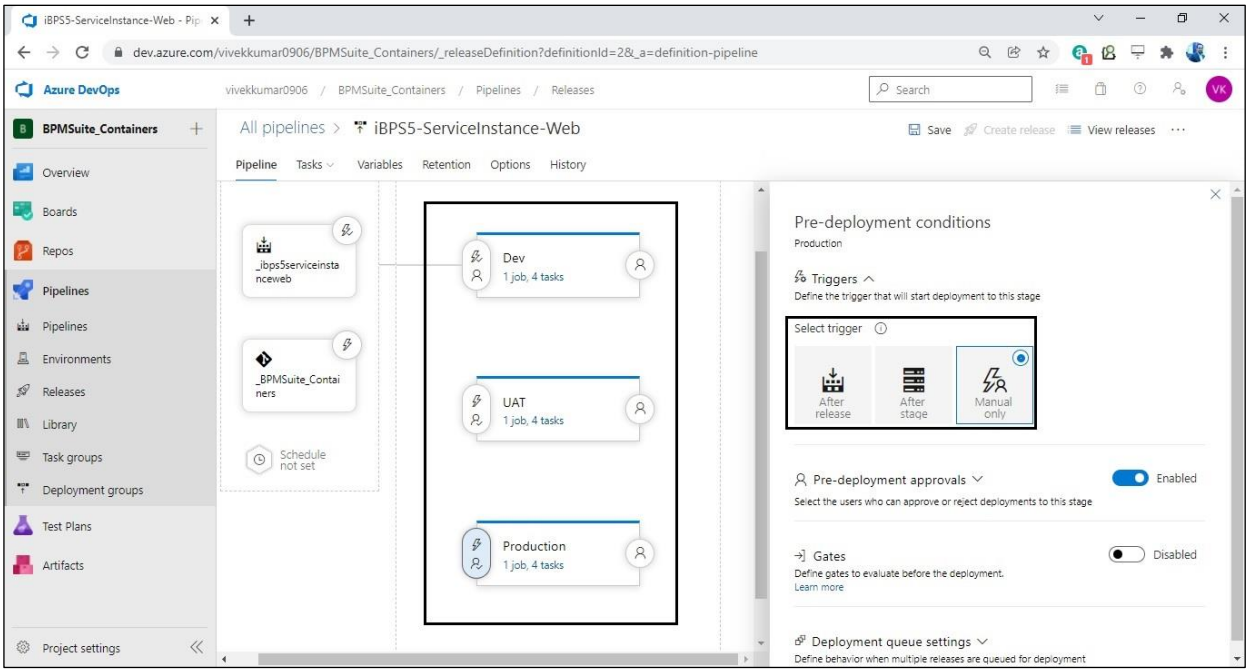

Figure 4.33

7. In the **Pre-deployment conditions** panel, select the list of users or stakeholders whose approval is required for the deployment to the Production stage.
8. Select **Any Order** as an **Approval order**. It indicates that approval of all Stakeholders is required (in any order).
9. Select **The user requesting a release or deployment should not approve it checkbox** in the select policies.
10. Click **Close icon** to close the **Pre-deployment conditions** panel.
11. Click **Save** to save the changes.

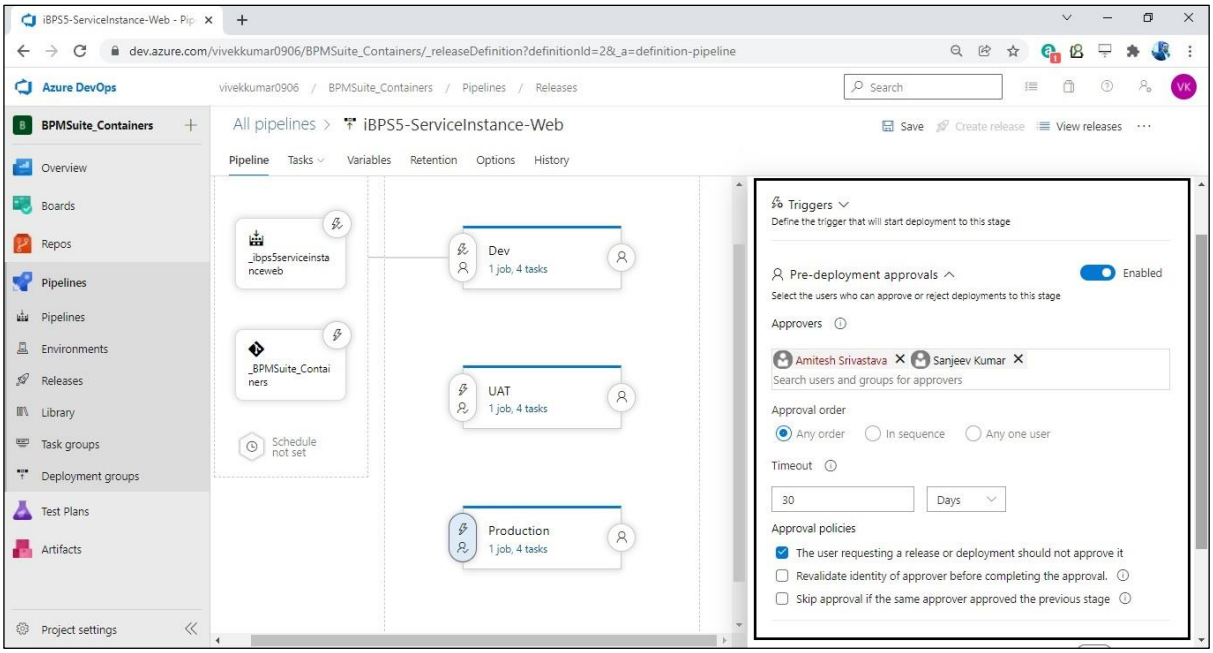

Figure 4.34

12. Click **View stage tasks** link to the **Production** stage.
13. Click **Add phase options** icon in the **Tasks** tab.

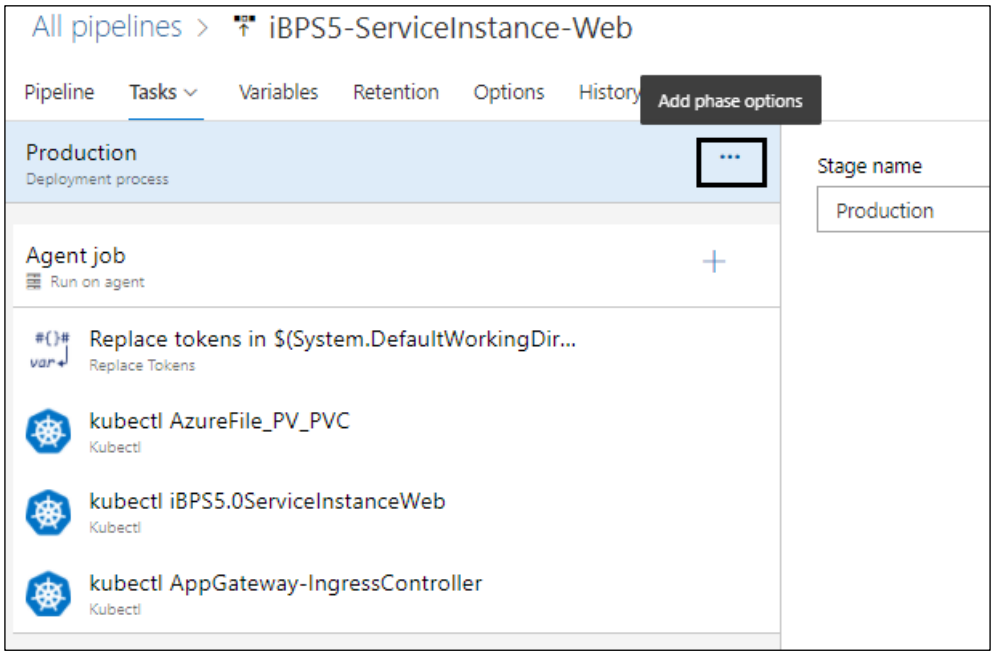

Figure 4.35

14. Select the **Add an agentless job**.

15. Move **Agentless job** above the **Agent Job** in the **Tasks** tab.

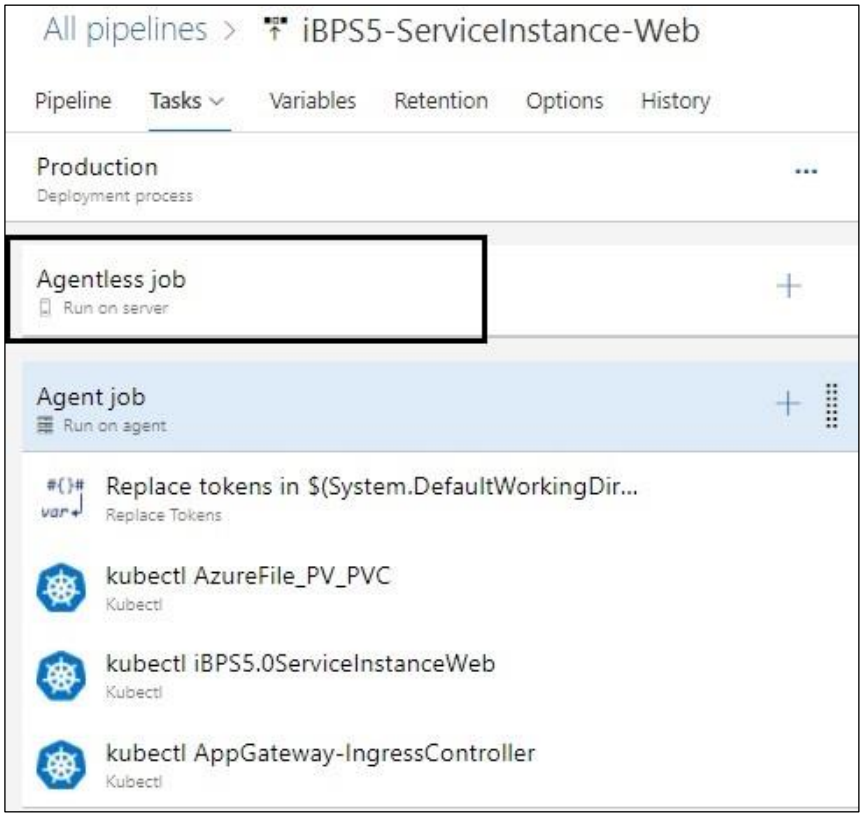

Figure 4.36

16. Click **Add a task to Agentless job** icon.
17. Add a **Manual intervention** task.

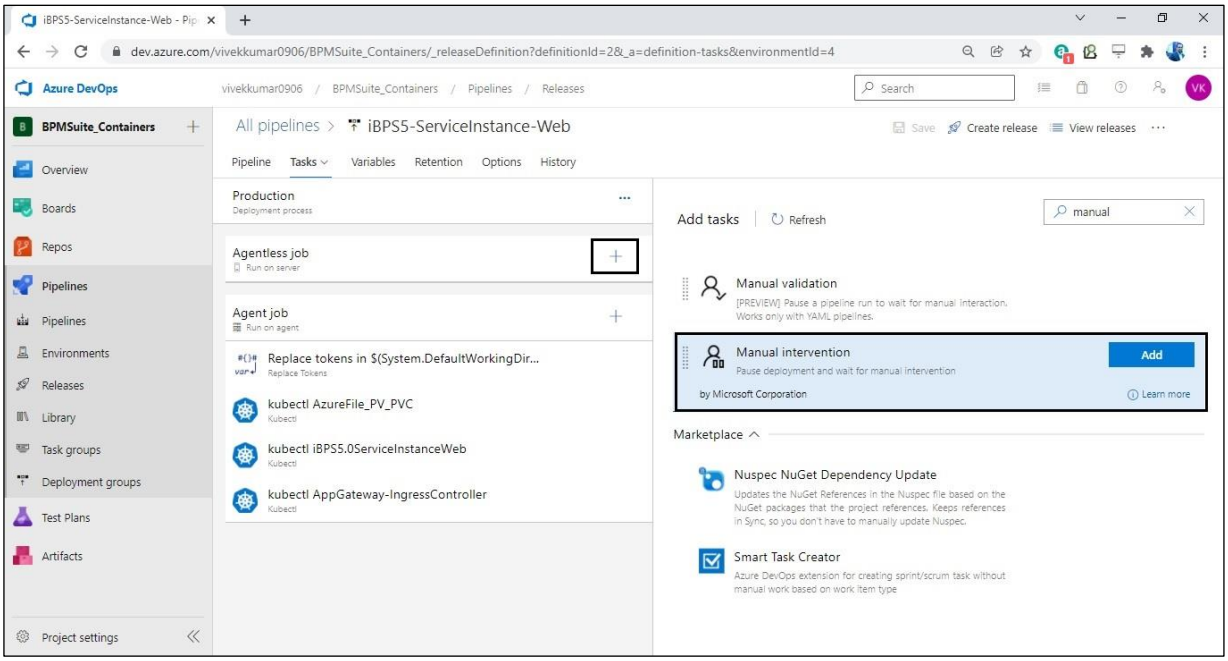

Figure 4.37

18. Click added task **Manual intervention**.
19. Specify the checklist points that need to execute before deploying to the production stage.

    For example:

    Before deploying to the Production, ensure that the below checklists are completed:

    - All Major and Catastrophic bugs must be fixed.
    - The latest images must be thoroughly tested on the Dev and UAT stages.
    - Approval has taken from all stakeholders.
    - Deployment downtime has taken from the client.
20. Select the user or group that are supposed to deploy to the production. A manual intervention mail with the above-mentioned checklist is sent to the engineer who is supposed to deploy to production with a checklist. During deployment, all the checklist points get verified before performing the production deployment. In case any point of the checklist is not covered, then deployment to the production gets rejected.

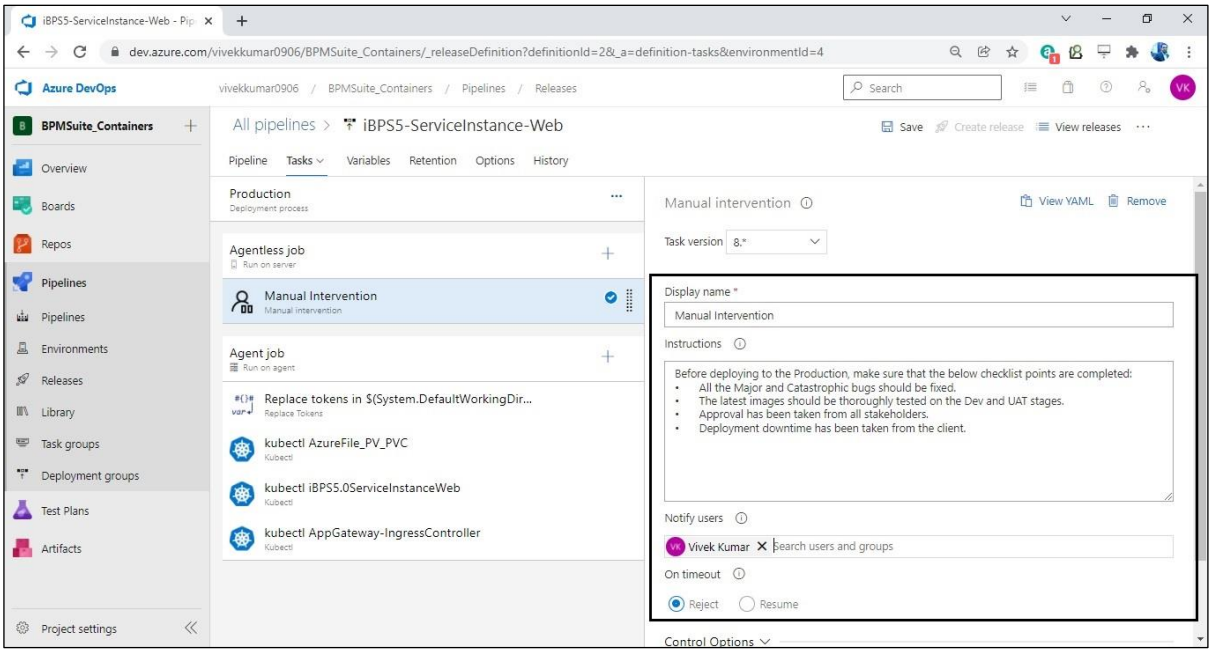

Figure 4.38

21. Make the other required changes in the Production stage's tasks as per your requirements. For example, you can make the following changes in the below tasks:
    - **Kubectl Task:** Kubernetes service connection, Kubectl command, changes in YAML files, and so on.

NOTE:
Refer the above steps to configure the Release Pipeline of other Docker Images.
