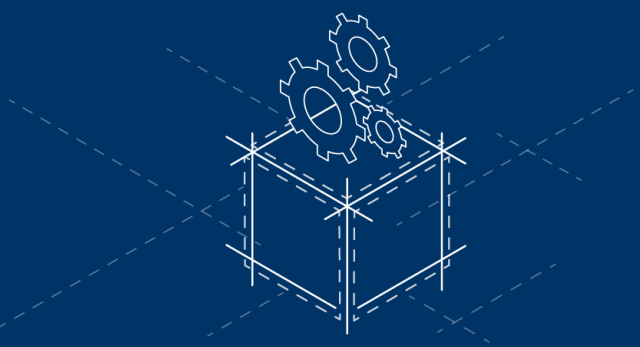
Cloudockit

Container

AZURE - INSTALLATION & CONFIGURATION GUIDE



GENERATED BY: Cloudockit

Contents

Introduction....4
Requirements....6
Step A – Deploy Cloudockit Container....7
Step 1 – Create a Storage Account and upload the license file. 7
Step 2 – Create your container environment and start your container. 8
Step B (Optional) – Configure Cloudockit Web UI....11
Optional – Store these credentials in Azure Key Vault 15
Step C (Optional) – Configure Cloudockit Container to support Scheduling....16
Start Cloudockit Scheduler Container....16
Set Settings in the settings file 16
Step D (Optional) – Configure Cloudockit Container to support the creation of Compliance Rules, Tailored Diagrams and Settings 17
Create (or re-use) an Azure Cosmos DB....17
Configure Cloudockit Container to use the Azure Comos DB....19
Step E (Optional) – Configure additional App Registrations for Cloudockit Container 20
Microsoft Graph....20
OneDrive for Business – Drop-off....20
Microsoft Graph....20
Step F – Understand Cloudockit API Container 22
Step G – Test your license 23
Activate and setup components for your license....23
Step H – Validate that you can authenticate to the environment that you want to scan....24
Step I – Test the document generation 26
Step J – Manage your document generation....27
/ListDocumentGeneration....27
/StopDocumentGeneration 27
Annex – Deploy multiple instances of Cloudockit Container 28
Step 1 – Create / Configure Azure Key Vault 29
Step 2 – Configure Azure Redis Cache 30

Step 3 – Define the Environment Variables required to run the Cloudockit Container 30
Annex – Troubleshooting 34

Introduction

The purpose of this document is to provide the detailed steps to run and configure Cloudockit Docker container images.

There are two types of images that you should run:

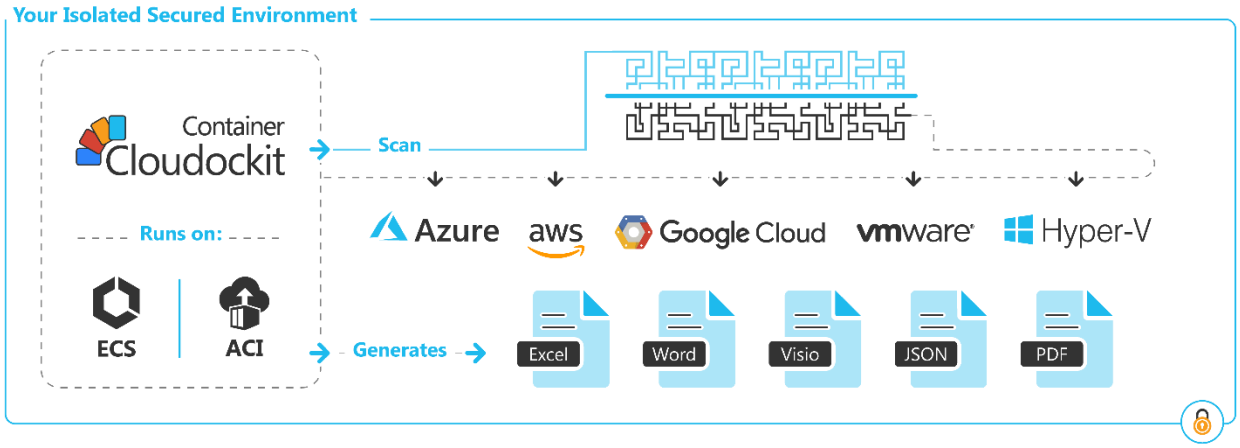
- **cdk-web-linux** that contains the Cloudockit API/Web interface. This is mandatory to run this container.
- **cdk-scheduler-linux** that contains the Cloudockit Scheduling features. This is an optional container you do not need to install if you do not want to use schedules.

The cdk-web image contains the Cloudockit API that you can call from your CI/CD processes or any other process / scenario which fits your business needs.

In addition to the API, we have integrated the complete Cloudockit Web UI in the image so that you can get all the features that you are accustomed to.

Cloudockit Docker container images provides you a way to run Cloudockit into your own isolated Cloud environment and gives you the exact same features as Cloudockit Website and Cloudockit Desktop.

Here is the high-level overview of the solution:



The following hosting environments are currently supported:

- Web App for Containers on Azure - Recommended
- ECS (Elastic Container Services) on AWS
- ACI (Azure Container Instance) on Azure
- GKE (Google Kubernetes Engine) on GCP

A few important things to note:

- These configurations are for the hosting of the container, not for the environment that you scan which means that you can scan a GCP project using the Cloudockit Container API even if the container runs on Azure.

- Depending on the hosting option that you choose, there could be some limitations. Those limitations are related to the hosting option and not the Cloudockit Container itself.
- The current document does not detail networking configuration like isolation/https setup as this is highly dependent on your internal setup.
- Container is currently designed to have one node running which should be more than enough to generate all the documents you need.
- For production environment, we recommend 4vCPU + 8 Gb RAM
- Cloudockit Web UI only supports Azure AD as SSO authentication. If you do not set it up, you will only be able to access the API portion.

The following sections contain the different steps to deploy the Cloudockit Docker container image on the Azure Platform.

Here is an overview of the different steps you must do to deploy Cloudockit Container:

Step A - Deploy Cloudockit Container

- This Step is subdivided in 4 different steps
- Those steps have been scripted to ease the deployment process so we strongly advise to use the scripted approach

Step B (Optional) - Activate Cloudockit Container UI

- Create an Azure AD Application

Step C (Optional) - Activate the Scheduling feature (cdk-scheduler)

- Set the appropriate settings to activate scheduling

Step D ... - Do some tests

- Test the license validity
- Start some documentation

Requirements

To install Cloudockit Container in your environment, you will need:

- A Storage Account
- An App Service or an Azure Container Instance to run the Container.
- (Optional) A Microsoft Entra ID App Registration if you want to activate Cloudockit Container Web UI

Important note

We highly recommend that you use the script (based on Azure CLI) provided by Cloudockit Team to provision the Cloudockit Container.

Step A – Deploy Cloudockit Container

Step 1 – Create a Storage Account and upload the license file.

To run Cloudockit Container, you need to have a Storage Account that will be used to store information (license file, settings…). As the license file is linked to this Storage Account, you need to choose a Storage Account Name (short name like *yourcompanycloudockit*) and send that name to Cloudockit Support Team (support@cloudockit.com) so that they generate a license file.

Once you receive your license file, you can upload the license file into a file named ***cloudockitinternal/license.json*** in the container.

Please note that the Json license file is tied to the storage account name so you need to create/use a storage account with a name that matches the name provided to Cloudockit Support Team.

Important note

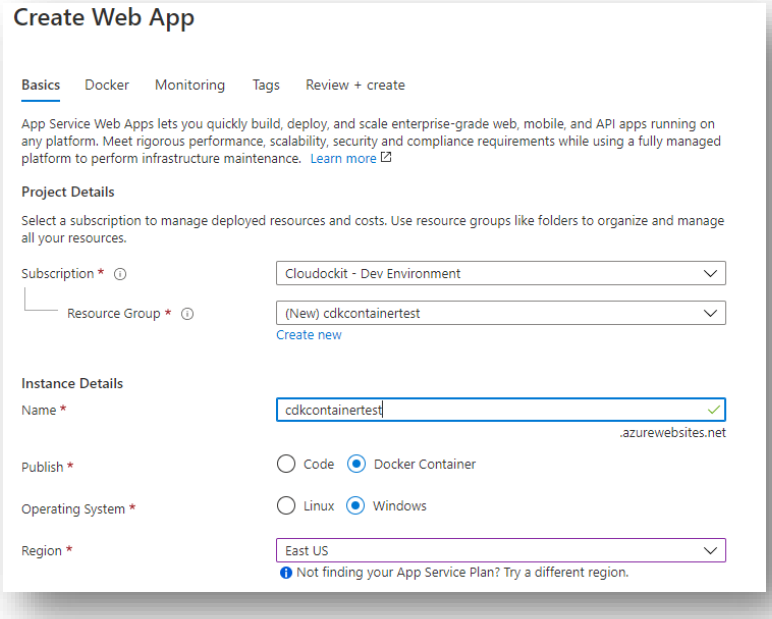
Ensure that the Storage Account exists in your environment before sending it to the Cloudockit Support Team.

The Json license file that you'll upload to your storage will be read by your Cloudockit Container when you start it. You will need to specify an Environment Variable / App Settings named DockerStorageAzureCnxString that will store a complete connection string to the storage account (cf. section below).

Step 2 – Create your container environment and start your container.

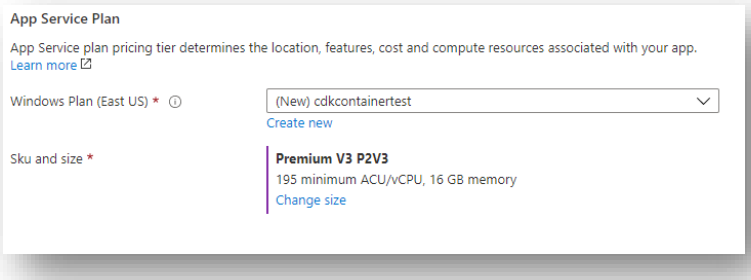
Once you have created your Storage Account, you need to create your container environment and start the container.

First, create a new Web App for Containers instance:



Ensure that you have selected **Docker Container** and Linux for the Operating System.

Then create a new App Service Plan:



Then, in the Docker Tab, select Private Registry and enter the following information:

Name	Value
Server URL	https://cloudockitcontainerregistry.azurecr.io
User Name	User provided by Cloudockit Team
Password	Password provided by Cloudockit Team
Image and Tag	cloudockitcontainerregistry.azurecr.io/cdk-web-linux:latest (linux)
Startup Command	Leave empty

Then, click on **Review+Create** and proceed.

Once done, you need to navigate to the Application Settings of the App Service that you have created and enter the following values:

Name	Value
WEBSITE_MEMORY_LIMIT_MB	Amount of RAM (in MB) that you have in your App Service Plan (1 GB is 1024) Minimum value is **2048**, recommended value is **4096**
AppInsightsConnectionString (optional)	An Azure Application Insights Connection String for advanced logging
DockerStorageCloudProvider	Specify if your Storage Account is stored in Azure, AWS or GCP. Possible values are: • Azure • GCP • AWS
DockerStorageAzureCnxString	Enter the complete connection string (Full access) of the Storage Account.
AuthenticationEndPoint (optional)	Specify if you want to use a different Azure endpoint. Possible values are: • gov • de • cn Leave empty for default endpoint (Global).
ExternalBaseUrl (optional)	Full public URL of the container when running behind a reverse proxy or load balancer (e.g. https://cloudockit.company.com). Required for OAuth redirect URIs to resolve correctly. When not set, the URL is derived from request headers only.
ExternalBaseUrlProxyHeader (optional)	Name of an HTTP header whose presence signals a proxied request (e.g. X-Forwarded-Proto). When set, the ExternalBaseUrl override applies only to requests carrying this header; other requests use the Host header.

TrustAllProxies (optional)	Set to True to trust all X-Forwarded-* headers without a URL override. Use only when the container is behind a trusted proxy and is not directly internet-exposed. Prefer ExternalBaseUrl.
ForwardProxyHopLimit (optional)	Number of proxy hops to trust for X-Forwarded-For. Default: 1. Increase when multiple load balancers sit in front of the container.

Step B (Optional) – Configure Cloudockit Web UI

Cloudockit Container supports a Web UI that allows users to authenticate by using an App Registration or Azure User Authentication.

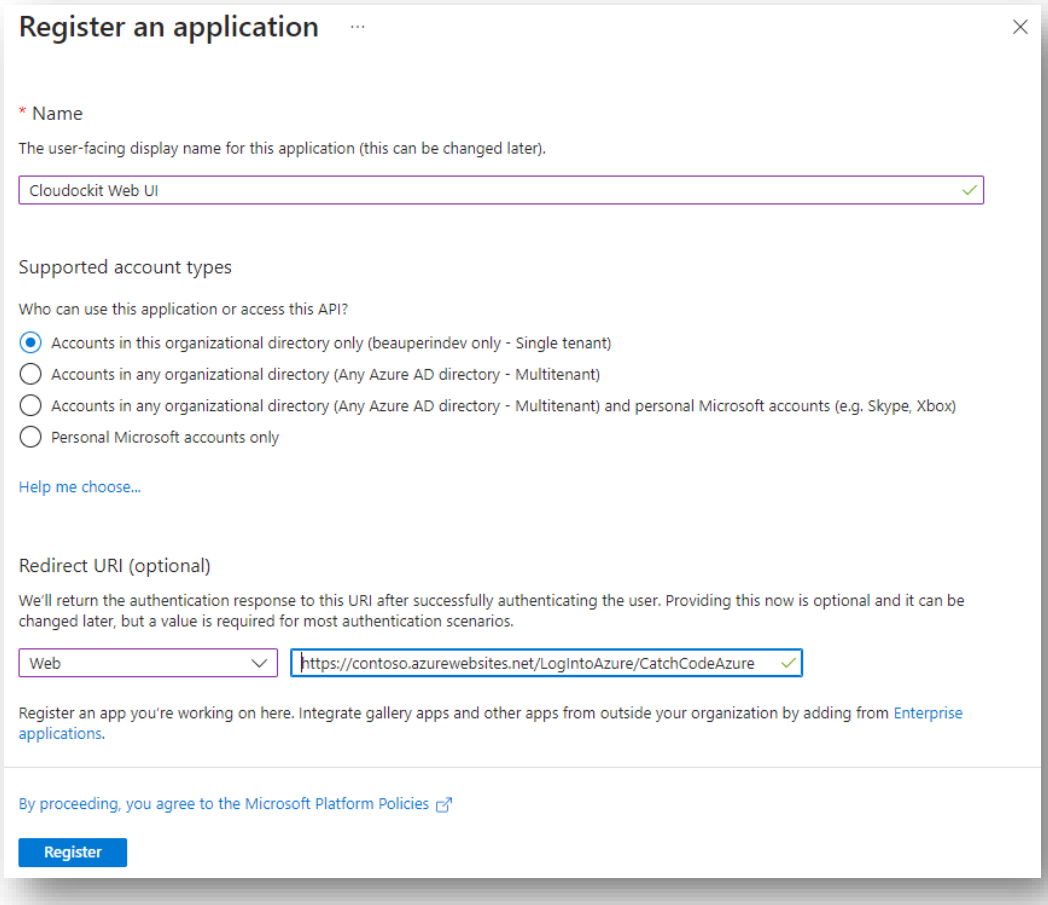
This Web UI supports Microsoft Entra ID as a first step to authenticate users.

Once connected, you will be able to connect to Azure, AWS and GCP using Service Accounts (Azure AD App, GCP Service Credentials, AWS Access Keys).

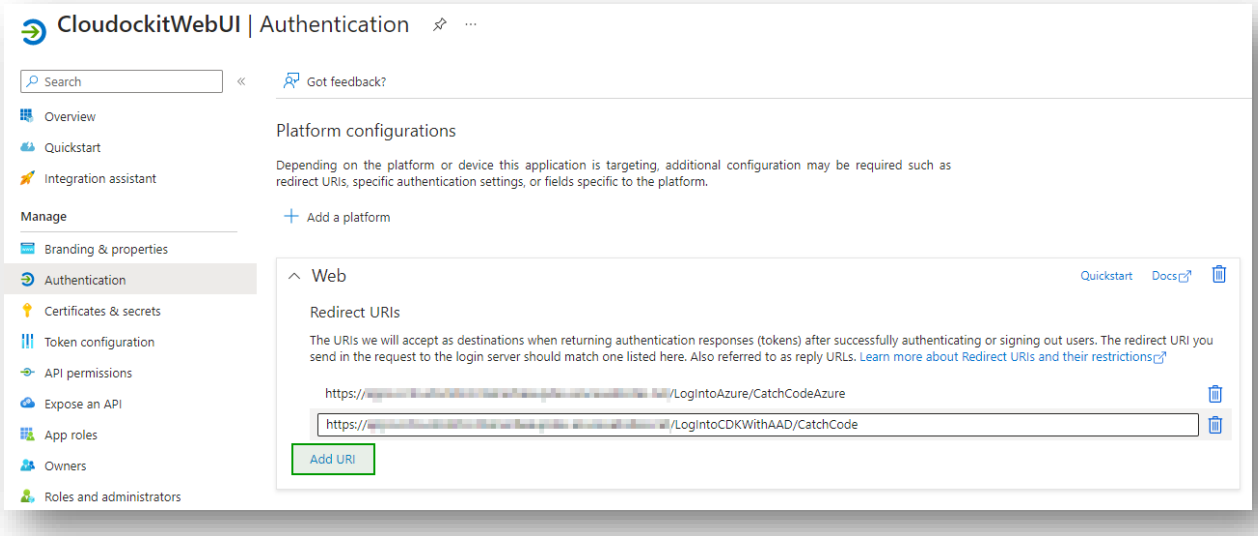
To activate Azure AD Authentication, you need to follow these steps:

- Go to your **Microsoft Entra ID**
- Click on **App Registration** and then click **New Registration**
- Enter a **Name** (any name you want) and select **Single Tenant**
- Enter the following redirect URIs (reply url):
    - `https://<AppSvcName>.azurewebsites.net/LogIntoAzure/CatchCodeAzure`
    - `https://<AppSvcName>.azurewebsites.net/LogIntoCDKWithAAD/CatchCode`

      where *AppSvcName* is the name of your App Service

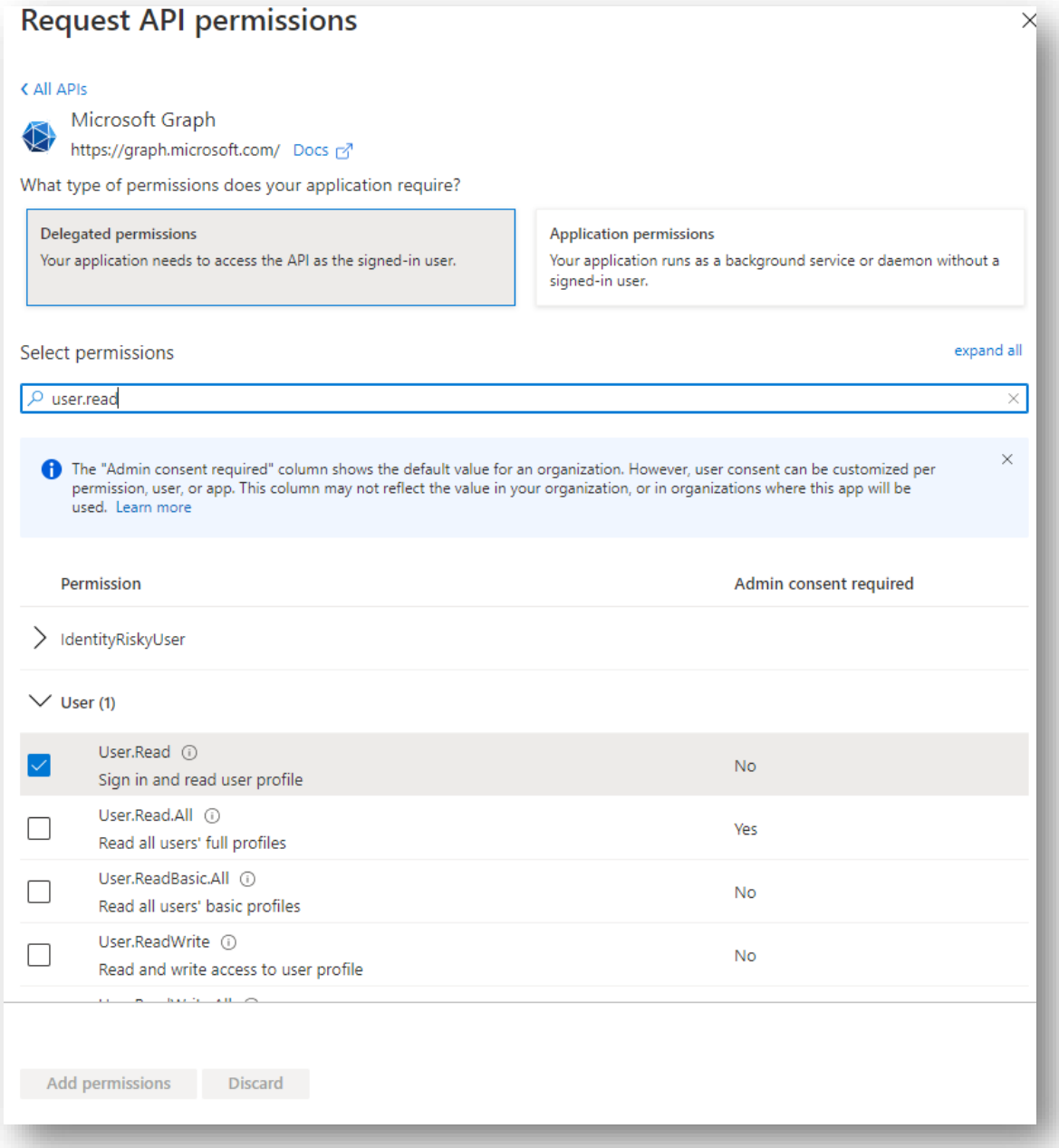


Note: the interface will not let you enter the 2nd URL before clicking on **Register** so you'll have to enter it after registration, in the Authentication page:

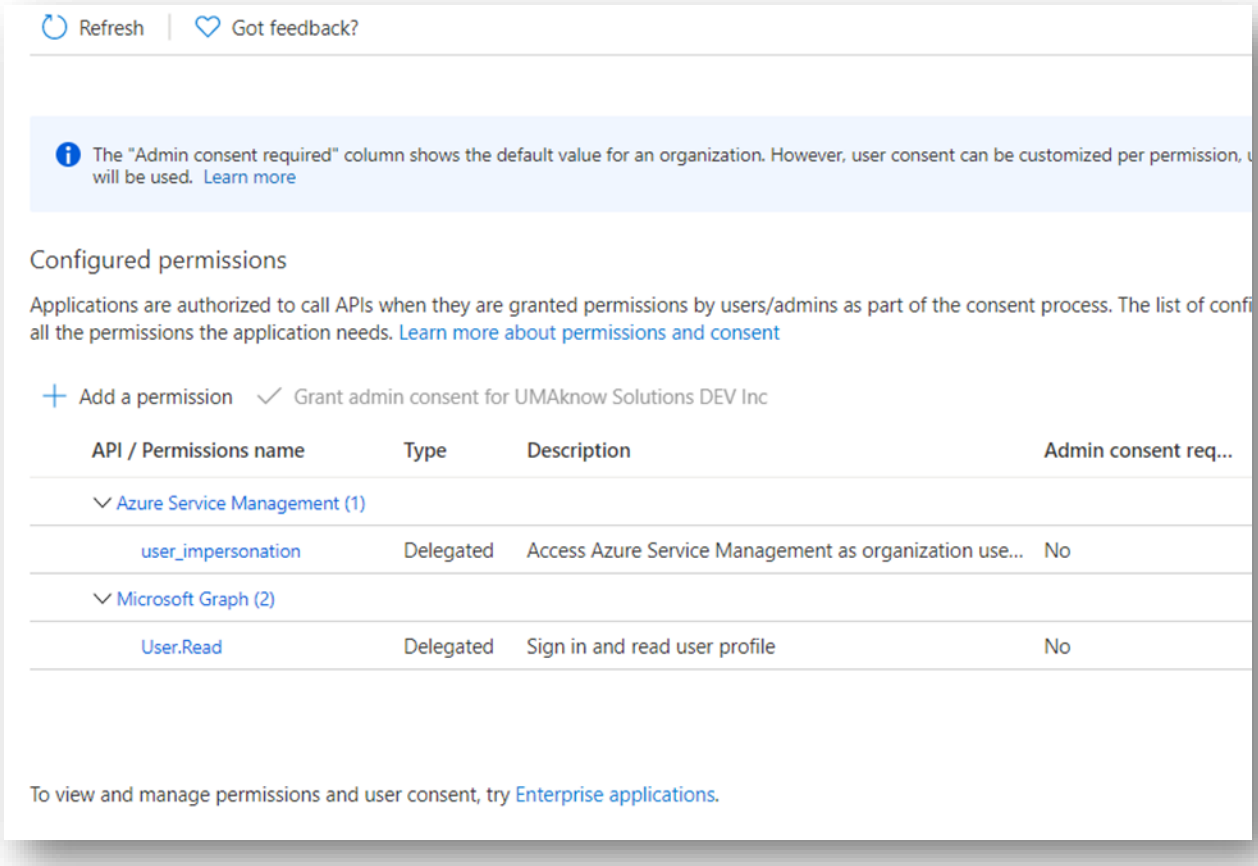


Then, go to **API Permissions**, click on **+Add a permission** and select:

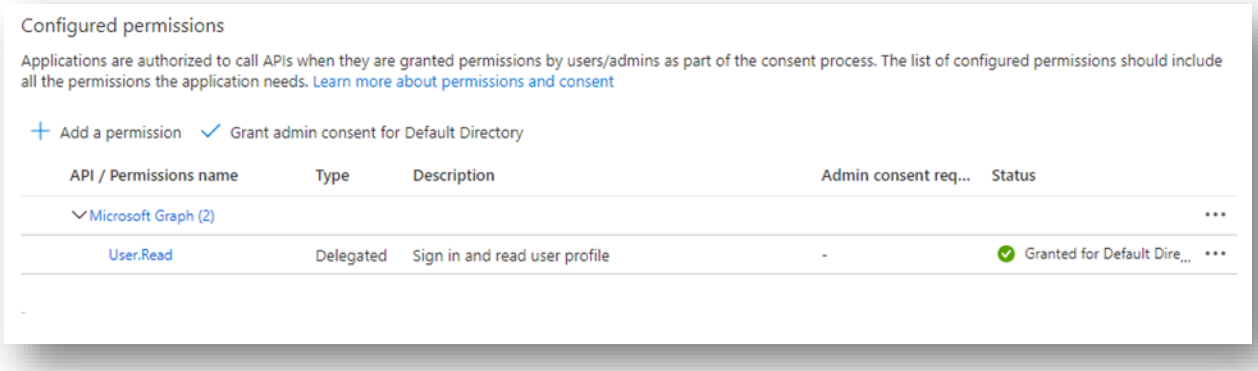
- **Microsoft Graph**, then **Delegated permissions** and then select ***User.Read***:
- **Azure Service Management**, then **Delegated permissions** and then select ***user_impersonation***:



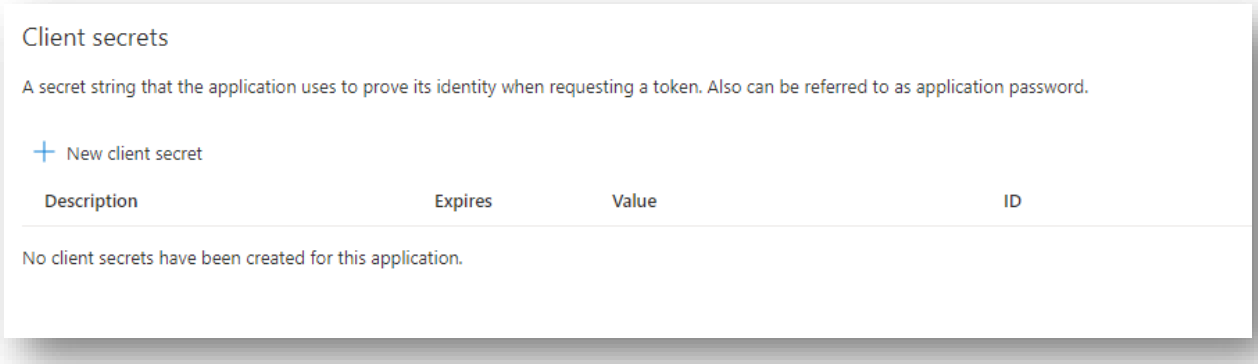
Click **Add permissions**. You should now see the following:



Then, click on **Grant Admin consent** for Default Directory (if you don't have the permissions to click on **Grant admin consent**, please contact your IT admin to do it for you):



Then, take note of the client ID from the Overview tab and then go to **Certificates & Secrets** and generate a new Client Secret, take note of it.



Update the settings file from your storage account (in the cloudockitinternal folder) with the value of the previously created Azure AD Application:

```
{
  "AzureADTenant": "mytenant.onmicrosoft.com",
  "AzureADAppID": "zzzzz",
  "AzureADAppKey": "zzzzz"
}
```

Optional – Store these credentials in Azure Key Vault

Instead of keeping the tenant, application (client) ID and client secret in plain text in the settings file, the container can read these three values from Azure Key Vault. When a Key Vault is configured, the container reads the credentials from it and only falls back to the settings file when the Key Vault is not configured or a value cannot be read. Existing setups keep working without any change.

Create one secret per credential in your Key Vault, using these names:

Secret name	Settings file field	Value
cloudockit-entra-tenant-id	AzureADTenant	Microsoft Entra tenant, e.g. mytenant.onmicrosoft.com
cloudockit-entra-client-id	AzureADAppID	Application (client) ID
cloudockit-entra-client-secret	AzureADAppKey	Application client secret

Then configure the container:

- Set the environment variable DockerSecretsAzureKeyVaultUri to your Key Vault URL, for example https://<your-vault>.vault.azure.net/.
- Assign the Key Vault Secrets User role on the Key Vault to the container's managed identity. The system-assigned managed identity is used by default; to use a user-assigned identity instead, also set DockerSecretsAzureManagedIdentityClientId to its client ID.

The credential values are never written to the container logs.

Step C (Optional) – Configure Cloudockit Container to support Scheduling.

Cloudockit Container supports a Scheduling Web UI that allows users to choose when they want to schedule the document generation.

To activate scheduling, you need to spin-up a new container based on the **cloudockitscheduler** image and set the appropriate settings in your settings file.

Start Cloudockit Scheduler Container

You need to follow the same procedure as you did in the previous step to spin up a new Scheduling container. You need to use the **cloudockitscheduler** image. This scheduler is basically reading the schedules files created from the UI and calling the API according to the schedule.

Here are the settings for the container:

- CPU: 1+
- RAM: 1.5GB+
- No inbound networking is required
- Outbound networking needs access to the storage account where the settings are stored and the API URL where Cloudockit is deployed.
- The following 3 environment variables are required:

Name	Value
DockerStorageCloudProvider	Specify if your Storage Account is stored in Azure, AWS or GCP. Possible values are: • **Azure** (select this value) • GCP • AWS
DockerStorageAzureCnxString	Enter the complete connection string (Full access) of the Storage Account.
DockerUrlForSchedulingStarts	Enter the URL of your API that hosts Cloudockit like: https://testcdkapi.azurewebsites.net/

Set Settings in the settings file

To activate the scheduling, you need to update the settings file from your storage account (in the *cloudockitinternal* folder) to specify the URL of your Cloudockit Container:

```
{
  "DockerUrlForSchedulingStarts": "https://mycloudockitcontainer"
}
```

This information will be used by Cloudockit Scheduling feature to specify which Web API to call.

Note: the **Scheduling** menu will only appear in the interface if you login with an App Registration.

Step D (Optional) – Configure Cloudockit Container to support the creation of Compliance Rules, Tailored Diagrams and Settings

Cloudockit Container supports the creation of new Compliance Rules, new Tailored Diagram and new Settings.

This feature requires that you deploy an Azure Cosmos DB to save the Compliance Rules and Tailored Diagrams.

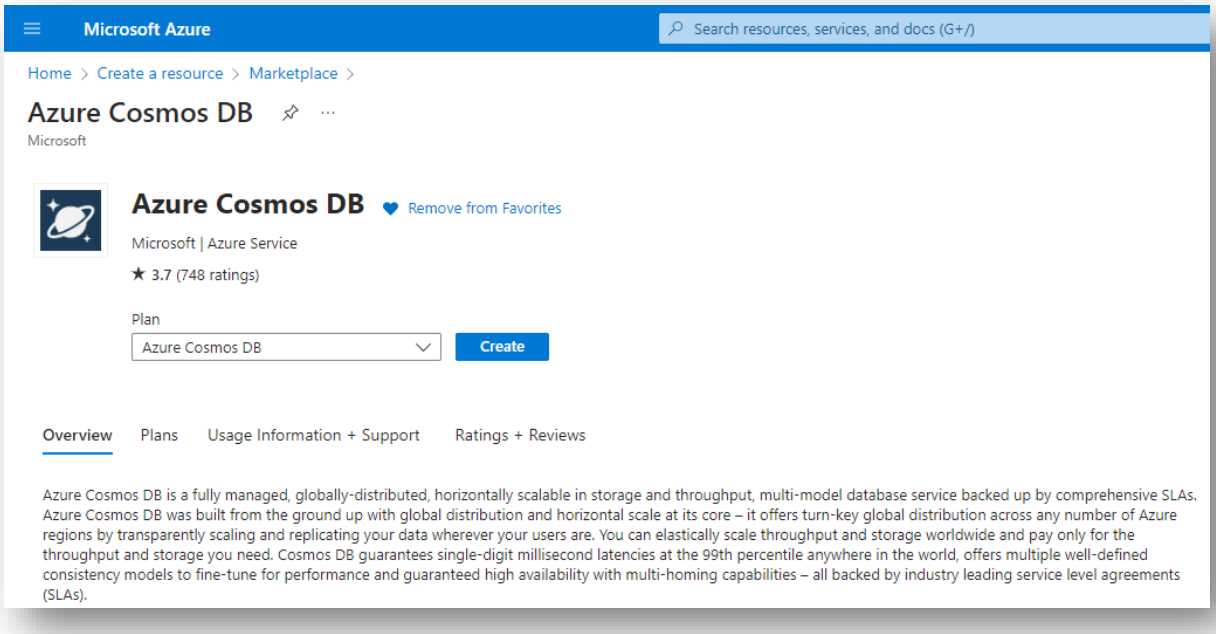
There are two steps required:

- Create (or re-use) an Azure Cosmos DB
- Add environment variables to the Cloudockit Container to specify which Azure Cosmos Database to use

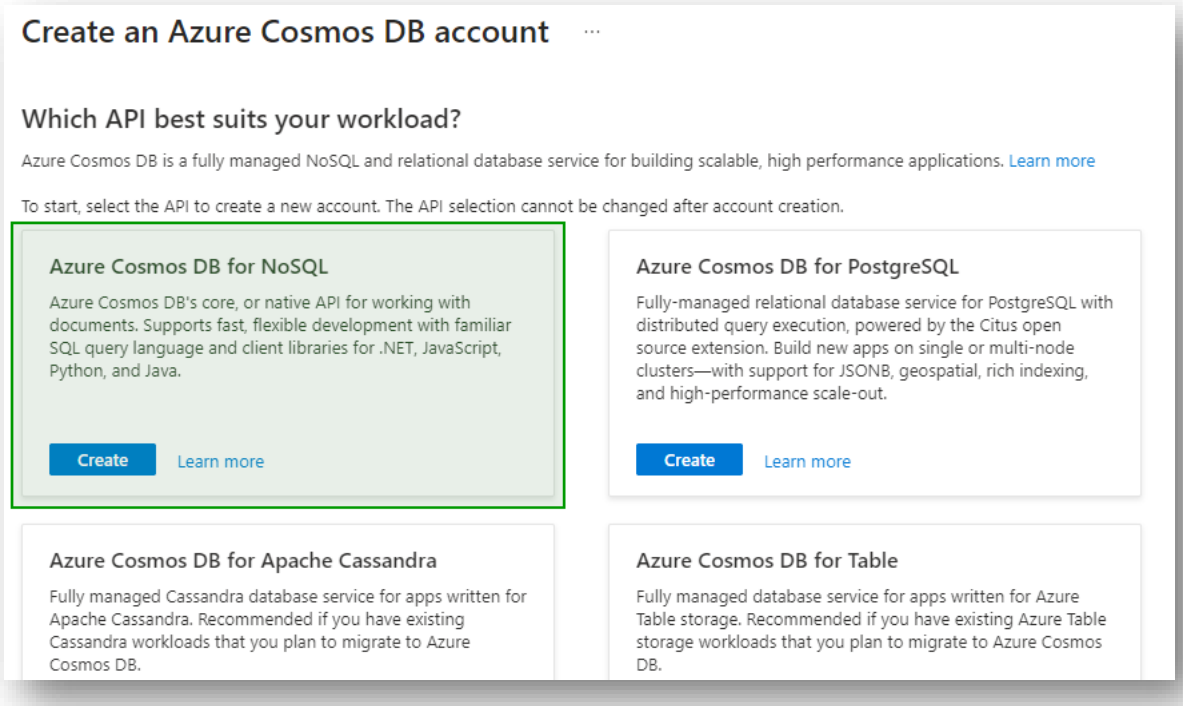
Create (or re-use) an Azure Cosmos DB

From the Azure Portal, create a new Cosmos DB: (you can skip those steps if you already have a Cosmos DB that you want to reuse)

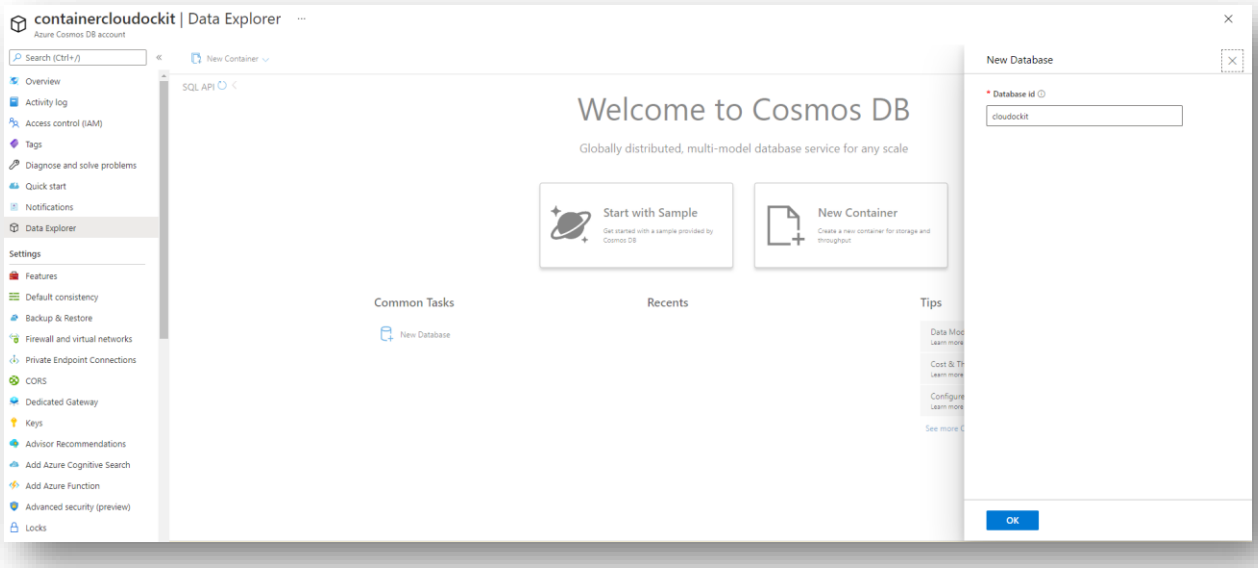
- Create a Cosmos DB



- Choose Azure Cosmos DB for NoSQL for the type



Once the Cosmos DB is created, you need to create a new Database named **cloudockit**:



Configure Cloudockit Container to use the Azure Comos DB

To ensure that the container can connect to the Database, you need to start the container and specify the following 2 required environment variables:

Name	Value
CosmosDb__DatabaseName	Enter the name of the Database that you have created in the previous step (**cloudockit** in the example)
ConnectionStrings__CosmosDb	Azure CosmosDB Connection string

Step E (Optional) – Configure additional App Registrations for Cloudockit Container

By default, Cloudockit Container uses the App Registration defined in the settings.json file for most authentication needs.

However, if you prefer not to use that App Registration, you can configure a different one. This may be useful if you want to manage permissions separately or avoid granting all permissions to a single registration. Alternatively, you can use multiple App Registrations if you prefer to manage permissions separately. Only the permissions relevant to your use case need to be added.

If you want to scan your Microsoft Entra ID you will need the following API permissions your App Registration:

Microsoft Graph

- Application.Read.All
- Directory.Read.All
- Group.Read.All
- User.Read.All
- Device.Read.All

If you want to authenticate using an App Registration without user interaction (i.e., not on behalf of a user), you must assign the required permissions as Application permissions rather than Delegated permissions, which require admin consent.

You will also need to add the following Web Redirect URI in the Authentication blade of the App Registration:
`https://<AppSvcName>.azurewebsites.net/CDKExternalAuthentication/AuthenticateMicrosoftEntraCatchCode`

Finally, you will need to add the two following Environment Variables in the App settings of your Web App:

Name	Value
MicrosoftEntraClientId	Client ID of the App Registration
MicrosoftEntraClientSecret	Client Secret of the App Registration

OneDrive for Business – Drop-off

If you want your generated documents to be dropped in your OneDrive for Business, you will need the following API permissions on your App Registration:

Microsoft Graph

- Files.ReadWrite

You will also need to add the following Web Redirect URI in the Authentication blade of the App Registration:

https://<AppSvcName>.azurewebsites.net/CDKExternalAuthentication/AuthenticateToOffice365CatchCode

Finally, you will need to add the two following Environment Variables in the App settings of your Web App:

Name	Value
OneDriveClientId	Client Id of the App Registration
OneDriveClientSecret	Client Secret of the App Registration

After adding the Environment Variables, make sure to restart your Web App to ensure the changes are applied correctly.

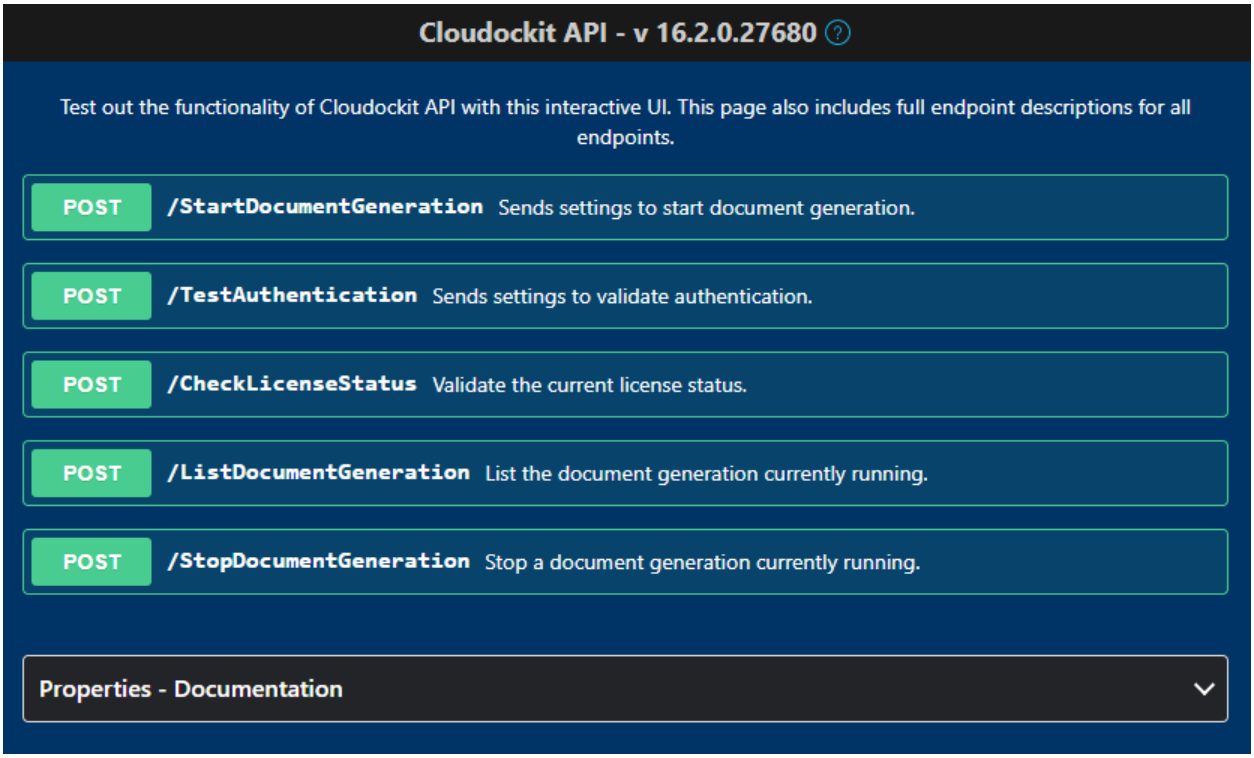
It is important to note that OneDrive only supports delegated flow; therefore, user interaction is always required for authentication as opposed to Microsoft Entra ID.

Step F – Understand Cloudockit API Container

Once you have installed the Cloudockit Container, you can navigate to the Container Home Page and you will see the following screen.

It gives you the option to test the different endpoints offered by Cloudockit API.

Please note that you can do everything from command lines/scripts and not use the interface if you prefer.



For simplicity of usage, all the endpoint are POST endpoints. Not all settings are mandatory for each endpoint and you can refer to that section to see which endpoints require which parameters.

Step G – Test your license

Activate and setup components for your license

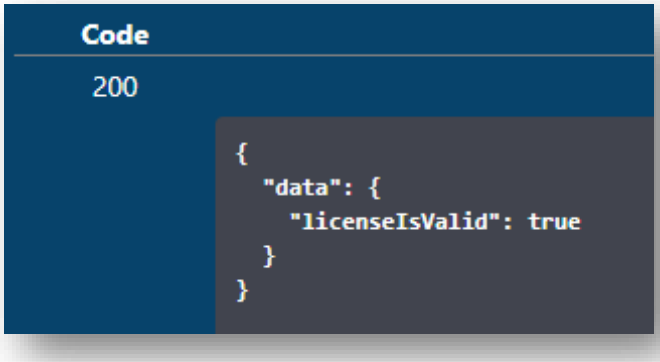
Once you get the API Key from Cloudockit team and you have the appropriate credentials for the license validation, you can check that your API Key is working by using the **/CheckLicenseStatus** endpoint.

First, navigate to the home page of the container and click on **CheckLicenseStatus** and Try it now. Then, replace the following values in the JSON that you are sending to Cloudockit API:

```
{
  "ApiKey": "API Key provided by Cloudockit Team"
}
```

Click on Execute.

You should receive the following response body:



Step H – Validate that you can authenticate to the environment that you want to scan

Once the license validation is successful, you need to test that the authentication to the environment you want to scan is working.

To do that, you need to use the **`/TestAuthentication`** endpoint.

First, you need to ensure that you specify the values from the above Step 2 for license validation.

Then, you need to specify the following additional values:

Name		**Value**
ADKCloudType		Azure/AWS/GCP depending on the platform that you want to scan.
SubscriptionID		Id/Alias of the subscription (Azure) or account (AWS) or project (GCP) that you want to scan.
(for AWS)	AWSAccessKeyId	AWS Access Key
	AWSSecretAccessKey	AWS Secret Access Key
	AWSScanRoleName	(Container only) Short role name (no ARN) assumed in each spoke account during scanning. Cloudockit builds arn:aws:iam::<account>:role/<name> for each spoke account. Leave empty to scan using the credentials produced at login.
(for Azure)	TenantID	Tenant name of the Azure Subscription to scan
	AppClientIdForAutomation	App Registration Client ID for the scan
	AppClientKeyForAutomation	App Registration Client Secret for the scan
(for GCP)	GCPServiceAccountCredentials	Content of the JSON Service Credential file
AzureStorageNameForDropOff		**Do not change the name of the parameter for AWS, this is also called AzureStorageNameForDropOff** You should specify one of these values: • the Azure Storage Account Name (it can be the unique Storage Account name that is in the same tenant as the subscription that you scan *or* the complete Azure Storage Account Connection String) • AWS S3 bucket • GCP Bucket where Cloudockit should store the documents generated.

Example of Payload for an AWS environment scan:

```
{
  "ApiKey": "xxxx",
  "AWSAccessKeyId": "XXXX",
  "AWSSecretAccessKey": "8PoBo+4XXXX+/k/MzQ",
  "SubscriptionID": "34XXXX2",
  "AzureStorageNameForDropOff": "XXXdockit",
  "ADKCloudType": "AWS"
}
```

Example of Payload for an Azure environment scan:

```
{
  "ApiKey": "xxxx",
  "TenantID": "X2.onmicrosoft.com",
  "AppClientIdForAutomation": "XXXXX",
  "AppClientKeyForAutomation": "mln/XXXXX=",
  "SubscriptionID": "XXX",
  "AzureStorageNameForDropOff": "XXX",
  "ADKCloudType": "Azure"
}
```

Example of Payload for an GCP environment scan:

```
{
  "ApiKey": "xxxx",
  "GCPServiceAccountCredentials": {"type":
"service_account","project_id": ""cdkXXXX"",""private_key_id"":
""XXXXX"",""private_key"": ""-----BEGIN PRIVATE KEY-----
"nMIIEvQIXXXXXZGy5PArVQS"n2buDJi0URXCKoeWnukG9Cl0fHlP8rFK6+XXXXXX+kJm0Y
xuFOwxdbgpS1n38mQyez7EK"nObnp9wP05ynOxKXJqJx0r1k="n-----END PRIVATE
KEY-----"n"",""client_email"":
""XXXX@cdkproject1.iam.gserviceaccount.com"",""client_id"":
""XXXXX"",""auth_uri"":
""https://accounts.google.com/o/oauth2/auth"",""token_uri"":
""https://oauth2.googleapis.com/token"",""auth_provider_x509_cert_url""
:
""https://www.googleapis.com/oauth2/v1/certs"",""client_x509_cert_url":
""https://www.googleapis.com/robot/v1/metadata/x509/test-
XXXX.iam.gserviceaccount.com"},   "SubscriptionID": "XXXX",
  "AzureStorageNameForDropOff": "XXXX",
  "ADKCloudType": "GCP"
}
```

Step I – Test the document generation

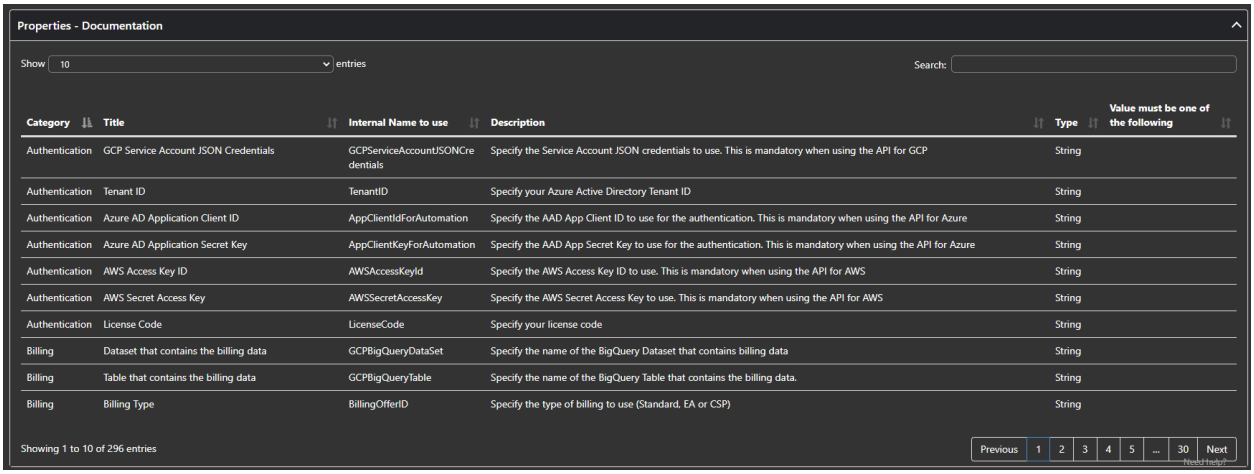
Once all the tests above have been done, you can start the document generation.

To do that, you need to use the **`/StartDocumentGeneration`** endpoint.

First, you need to ensure that you specify the same values as the above steps for CheckLicenseStatus and TestAuthentication endpoints.

Then, you need to specify additional values based on the type of document you want to generate and which option you would like to use.

You get a list of all options from the properties list at the bottom of the screen:



Properties - Documentation

Show 10 entries Search:

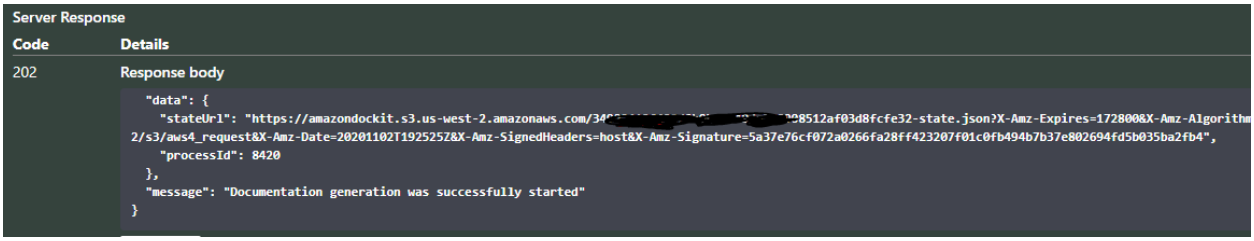
Category	Title	Internal Name to use	Description	Type	Value must be one of the following
Authentication	GCP Service Account JSON Credentials	GCPServiceAccountJSONCredentials	Specify the Service Account JSON credentials to use. This is mandatory when using the API for GCP	String	
Authentication	Tenant ID	TenantID	Specify your Azure Active Directory Tenant ID	String	
Authentication	Azure AD Application Client ID	AppClientIdForAutomation	Specify the AAD App Client ID to use for the authentication. This is mandatory when using the API for Azure	String	
Authentication	Azure AD Application Secret Key	AppClientKeyForAutomation	Specify the AAD App Secret Key to use for the authentication. This is mandatory when using the API for Azure	String	
Authentication	AWS Access Key ID	AWSAccessKeyId	Specify the AWS Access Key ID to use. This is mandatory when using the API for AWS	String	
Authentication	AWS Secret Access Key	AWSSecretAccessKey	Specify the AWS Secret Access Key to use. This is mandatory when using the API for AWS	String	
Authentication	License Code	LicenseCode	Specify your license code	String	
Billing	Dataset that contains the billing data	GCPBigQueryDataSet	Specify the name of the BigQuery Dataset that contains billing data	String	
Billing	Table that contains the billing data	GCPBigQueryTable	Specify the name of the BigQuery Table that contains the billing data.	String	
Billing	Billing Type	BillingOfferID	Specify the type of billing to use (Standard, EA or CSP)	String	

Showing 1 to 10 of 296 entries Previous 1 2 3 4 5 ... 30 Next

As there are many options that you can provide, we strongly advise that you use Cloudockit Website to generate the JSON file with the options.

One of the options that is particularly useful in this scenario are the CallbackURL and CallBackUrlRequired parameters that gives you the ability to be notified once document generation have been done.

When you hit Execute, you get the state URL of the current document generation:



Server Response

Code: 202

Details

Response body


```
{
  "data": {
    "stateUrl": "https://amazondockit.s3.us-west-2.amazonaws.com/34[redacted]8512af03d8fcfe32-state.json?X-Amz-Expires=172800&X-Amz-Algorithm
2/s3/aws4_request&X-Amz-Date=20201102T192525Z&X-Amz-SignedHeaders=host&X-Amz-Signature=5a37e76cf072a0266fa28ff423207f01c0fb494b7b37e802694fd5b035ba2fb4",
    "processId": 8420
  },
  "message": "Documentation generation was successfully started"
}
```


For Payload example, you can simply re-use the previous ones.

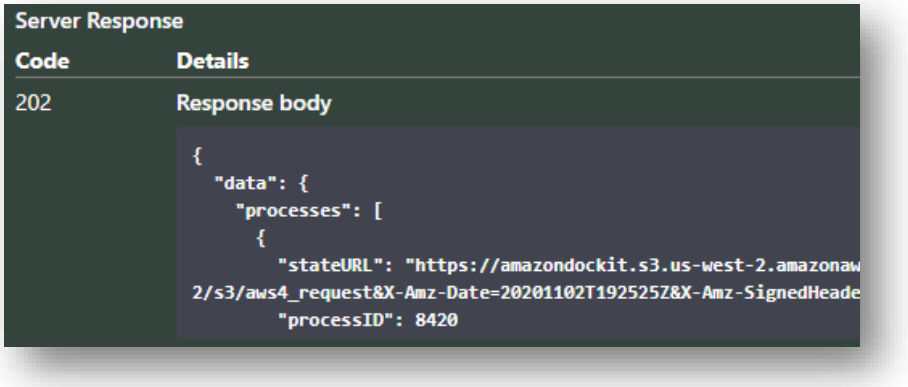
Step J – Manage your document generation

The Cloudockit API offers two endpoints to facilitate the management of document generation.

Please note that for these endpoints, you need to specify an Admin API Key for the ApiKey value.

/ListDocumentGeneration

This will allow you to see which scans are running. It gives you the list of running processes with their Process ID and State:

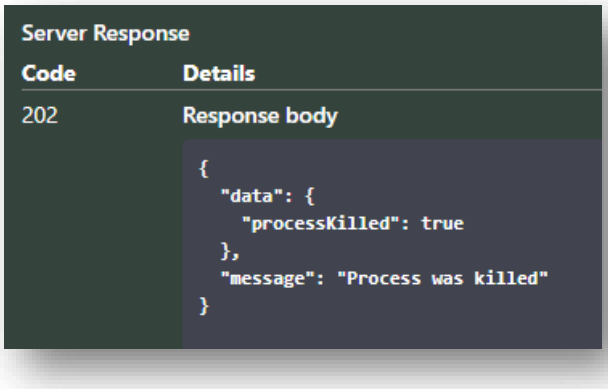


/StopDocumentGeneration

This endpoint is used to kill a running document generation.

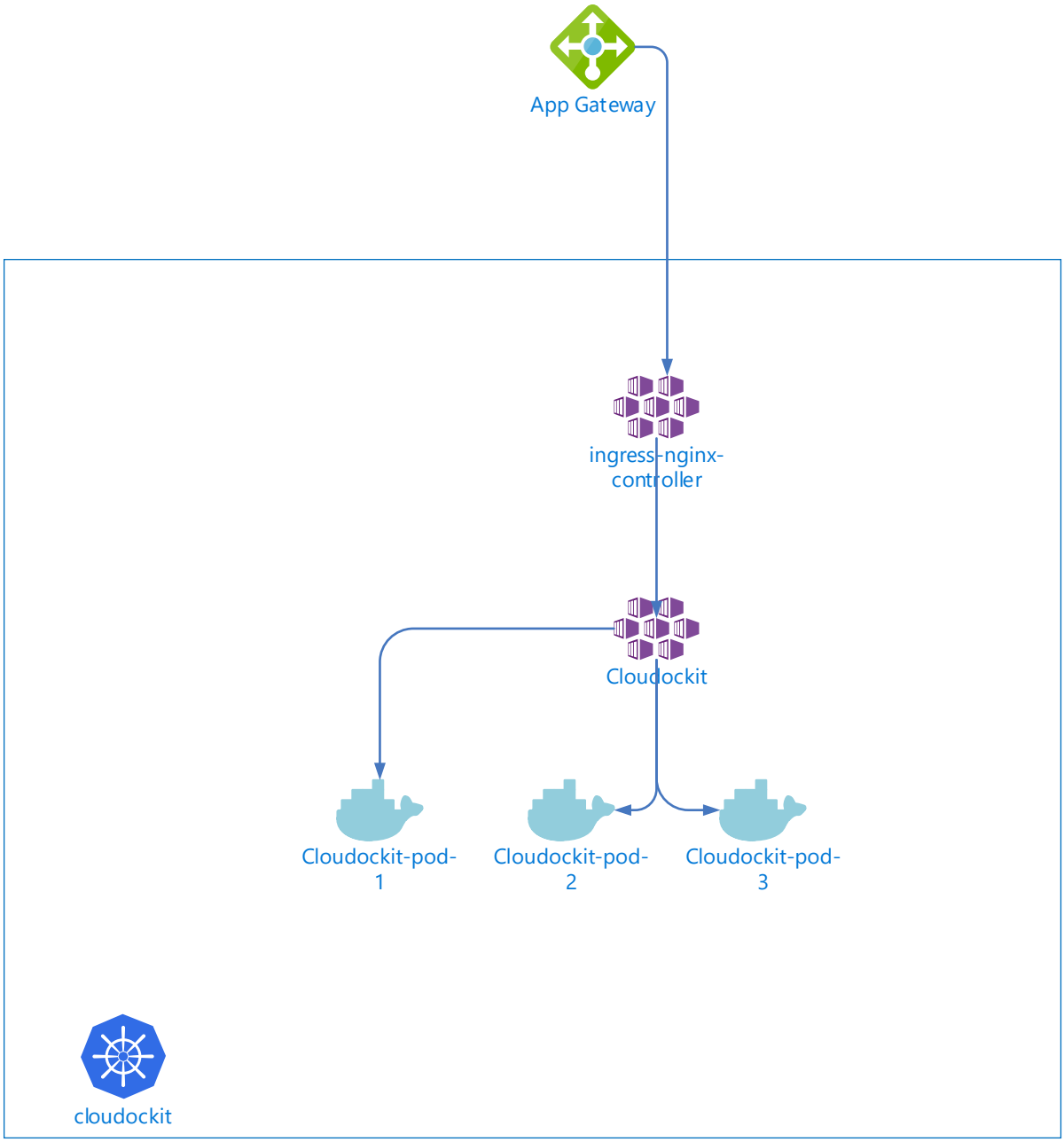
Name	Value
DockerProcessToKill	Value of the process ID to kill

It will reply with a confirmation message that the process has been killed.



Annex – Deploy multiple instances of Cloudockit Container

Cloudockit can be deployed in multiple instances in scenarios like this one:



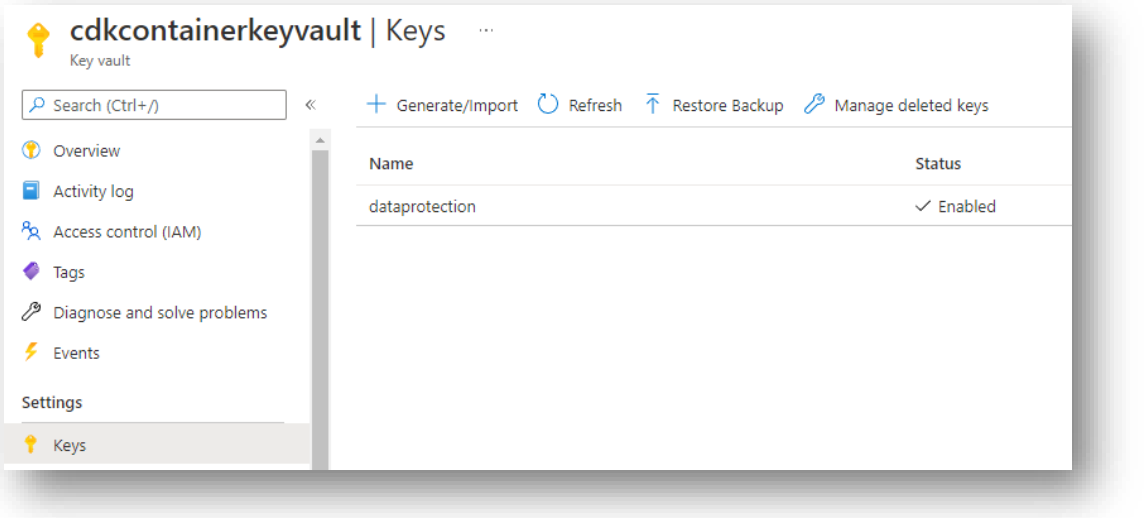
If you plan to use Cloudockit Container in a multi-pods environment, you need to configure some extra components. If you plan to use Cloudockit Containers in multiple instances with sticky session (for example App Services with a Traffic Manager), you do not need those extra components.

Here are the components that you need to configure.

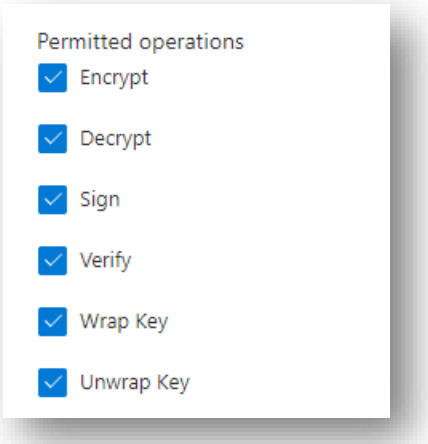
Step 1 – Create / Configure Azure Key Vault

To encrypt the anti-forgery keys used by ASPNETCore, an Azure Key Vault is required. You can create a new Azure Key vault or reuse an existing one.

Once you have the Azure Key Vault, you need to create a Key named **dataprotection**



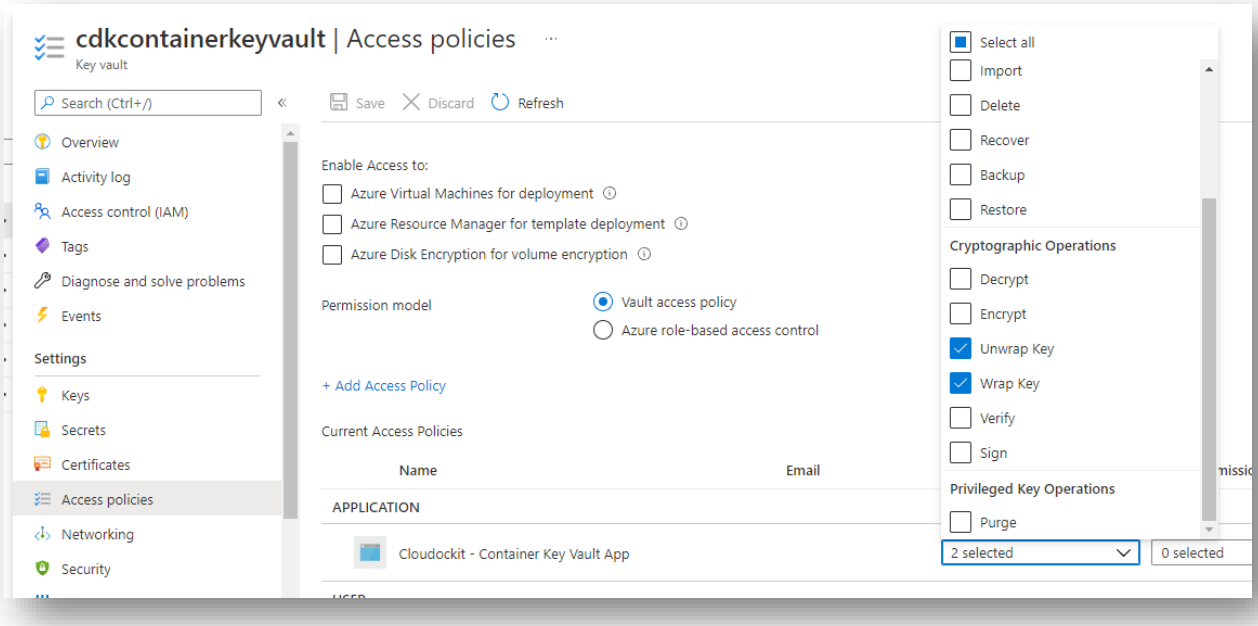
Please ensure that the Key have the following Permitted Operations (by default permissions)



Once you have done that, you need to create an Azure App Registration that will have access to this key. (you can also reuse the Azure AD App that you have created in the steps to configure Cloudockit Web UI if you prefer)

To do that, create a new App Registration (leave default settings) and note the Client ID and Client Secret as you will need that in the next steps.

Go back to the Azure Key Vault and give the Permissions to Unwrap Key / Wrap Key to the App that you just created



Step 2 – Configure Azure Redis Cache

As sessions can sprawl to multi pods, Azure Redis Cache is required to have consistent cache across all nodes.

Create a new Azure Cache for Redis (you can also reuse an existing one if you prefer) and select the Basic C0 (250MB Cache) as only small elements will be cached. Ensure that you select a region that is close to the one where Cloudockit will run for performance optimization.

Once created, take note of the Redis Connection String.

Step 3 – Define the Environment Variables required to run the Cloudockit Container

In addition to the environment variables defined in the step above, you now need to add the following environment variables.

Name	Description	Example
DataProtection__EncryptionKeyUrl	URL of the key vault Key that you have created. You need to specify the **Full Path to the key**, not only the key vault.	https://cdkcontainerkeyvault.vault.azure.net/keys/dataprotection
DataProtection__VaultClientId	Id of the Azure AD App that has privileges to Wrap / Unwrap key	760fb963-57a4-2303-1450-1b2dab513854

DataProtection__VaultSecret	Secret of the Azure AD App	SF7Q~NvuAYKF6.IBFjdewdewd
CacheSettings__UseRedis	Set to true to use redis instead of memory cache	true
ConnectionStrings__Redis	Connection String to the Redis	cdkmultipods.redis.cache.windows.net:6380,password=xx=,ssl=True,abortConnect=False
ExternalBaseUrl	Full public URL of the container when running behind a reverse proxy or load balancer. Required for OAuth redirect URIs to resolve correctly behind a proxy.	https://cloudockit.company.com
ExternalBaseUrlProxyHeader	HTTP header whose presence signals a proxied request. When set, the ExternalBaseUrl override only applies to requests that carry this header.	X-Forwarded-Proto
TrustAllProxies	Set to True to trust all X-Forwarded-* headers. Use only when the container is behind a trusted proxy and is not directly internet-exposed. Prefer ExternalBaseUrl.	true
ForwardProxyHopLimit	Number of proxy hops to trust for X-Forwarded-For. Increase when multiple load balancers sit in front of the container.	1

For reference, here is a sample yaml file to deploy that configuration

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cloudockit
spec:
  replicas: 4
  selector:
    matchLabels:
      app: cloudockit
  template:
    metadata:
      labels:
        app: cloudockit
    spec:
      containers:
        - name: cloudockit
          image: cloudockitcontainerregistry.azurecr.io/cdk-web-linux:latest
          ports:
```

```
            - containerPort: 8080
          env:
            - name: DockerStorageCloudProvider
              value: "Azure"
            - name: "DockerStorageAzureCnxString"
              value:
"DefaultEndpointsProtocol=https;AccountName=cloudockitcontainer;AccountKey=xxxx==
;EndpointSuffix=core.windows.net"
            - name: AzureADTenant
              value: "xxx.onmicrosoft.com"
            - name: AzureADAppID
              value: "9548a025-xxxx"
            - name: AzureADAppKey
              value: "W6UXfqC~xxxx"
            - name: Data__ProtectionEncryptionKeyUrl
              value:
"https://cdkcontainerkeyvault.vault.azure.net/keys/dataprotection"
            - name: DataProtection__VaultClientId
              value: "760fb9xxxx"
            - name: DataProtection__VaultSecret
              value: "VSF7xxxx"
            - name: CacheSettings__UseRedis
              value: "true"
            - name: ConnectionStrings__Redis
              value:
"cdkmultipods.redis.cache.windows.net:6380,password=xxxxk9g=,ssl=True,abortConnec
t=False"
            - name: APPINSIGHTS_CONNECTIONSTRING
              value:
"InstrumentationKey=xxx;IngestionEndpoint=xxx;LiveEndpoint=xxx
;ApplicationId=xxx"
            - name: TriggerDeployCount
              value: "5"
---
apiVersion: v1
kind: Service
metadata:
  name: cloudockit
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: cloudockit
```

Annex – Troubleshooting

Here are resolutions to common cases and how you can help find errors in Cloudockit Container.

- If you activate Cloudockit Container Web UI and noticed that in the upper right corner you have a Welcome message without your name, please check the credentials in the settings file
- If you are using Private endpoint for your App Service and Storage, please ensure that you activate vNET integration so that the App Service can communicate with the Storage Account
- You can specify an environment variable in your container named AppInsightsConnectionString that contains an Azure Application Insights Connection String so that you can see the logs.
- You can use the -logs.txt file in the storage that you have specified to see what is happening during document generation.
- If you get an error when the document generation starts, please ensure that you have Write privileges to your storage account
- If you see the message that the document generation is starting but do not see any progress, please verify that you have a CORS rule for GET Verb and origin that is your Cloudockit container website (should be done automatically).
- If you get an exception when starting the container that says “APPCMD failed with error code 87”, check that the variables that you are providing do not contain quotes.