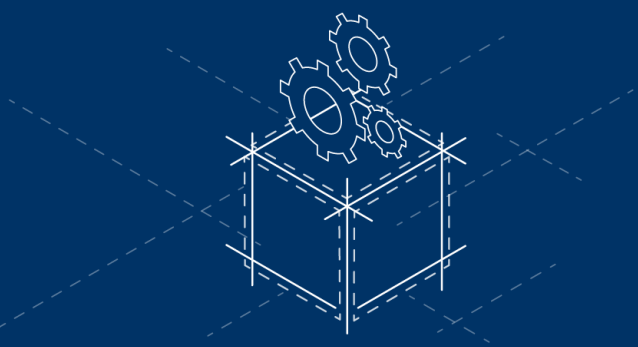
Cloudockit

Container

GCP - INSTALLATION & CONFIGURATION GUIDE



GENERATED BY: Cloudockit

Contents

Introduction

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

There are two types of images that you should run:

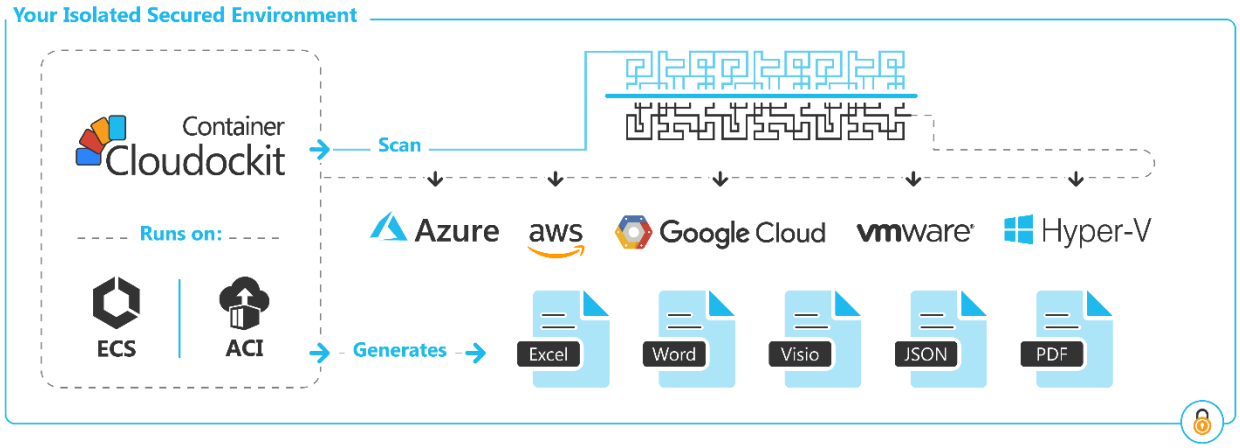
- **cdk-web-linux** that contain the Cloudockit API/Web interface. This is mandatory to run this container.
- **cdk-scheduler-linux** that contain the Cloudockit Scheduling features. This is an optional container you do not need to install if you do not want to use schedules.

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 provide 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 by itself. As an example, ACI currently does not yet support private networking (virtual networks) for Windows Based Container.
- The current document does not detail networking configuration like isolation/https setup as this is highly depending on your internal setup.
- Container is currently designed to have one node running which should be more than enough to generate all your documents you need.
- For production environment, we recommend 4vCPU + 8 Gb RAM
- Cloudockit Web UI only supports Azure AD as SSO authentication (Preview). If you do not set it up, you will only be able to access the API portion.

The following sections contains the different steps to deploy the Cloudockit Docker container image on the GCP Platform.

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

Step 1 - Create a Bucket and Get A License

- Create a Bucket
- Ask Cloudockit Support team for a license file

Step 2 - Create and Start Cloudockit Container (cdk-web)

- Retrieve the Cloudockit Container image
- Create the Cloudockit Container hosting environment with the Cloudockit Container image

Step 3 (Optional) - Activate Cloudockit Container UI

- Create an Azure AD Application

Step 4 (Optional) - Activate the Scheduling feature (cdk-scheduler)

- Set the appropriate settings to activate scheduling

Step 5 ... - Do some tests

- Test the license validity
- Start some documentation

Requirements

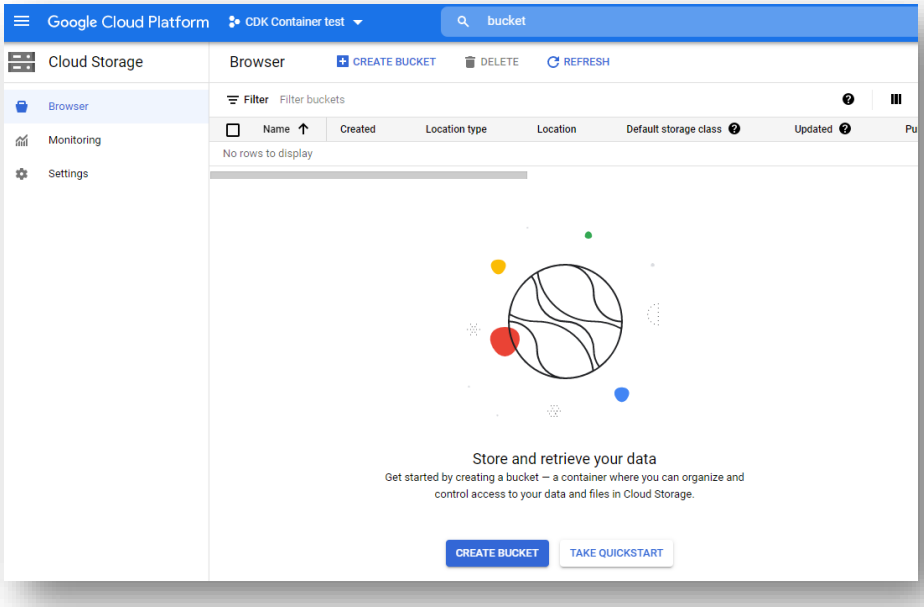
To install Cloudockit Container in your environment, you will need:

- A Bucket
- A Container Registry
- A GKE Cluster
- An Azure Active Directory Application if you want to activate Cloudockit Container Web UI (Optional)

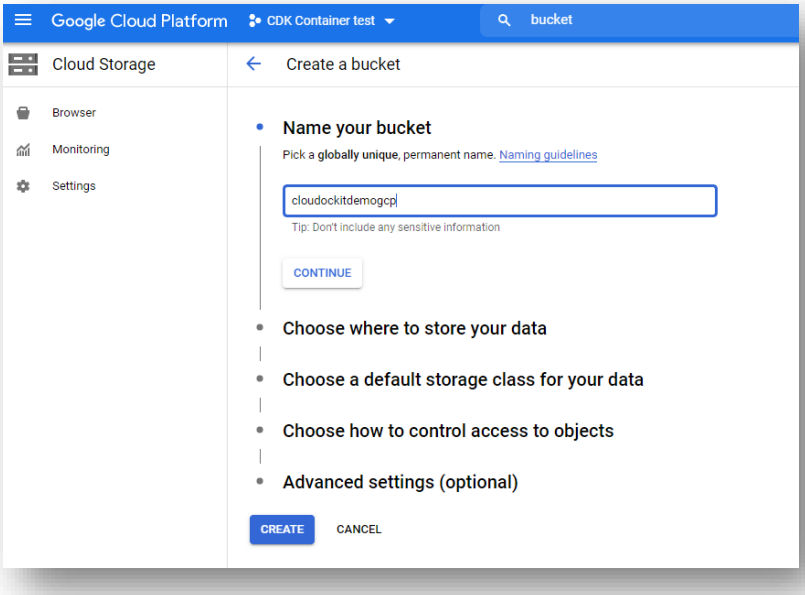
Step A – Create a Bucket and Get a License

Create a Bucket

The first step to deploy Cloudockit Container is to create a Bucket in your GCP Project.



You need to ensure that the name that you provide matches the name provided to Cloudockit team as license validation is done with the name of the bucket:



Click on **Create** (you can choose the location that you want and leave the default settings for the other options)

The only requirement for the Bucket is that it is visible by the Container as the Container will communicate with the Bucket to retrieve the configuration files.

This bucket will be used to store the following information:

- License file
- Users registered in the product
- Settings of the Container
- Schedules
- Other configurations like Compliance Rules, Tailored Diagrams

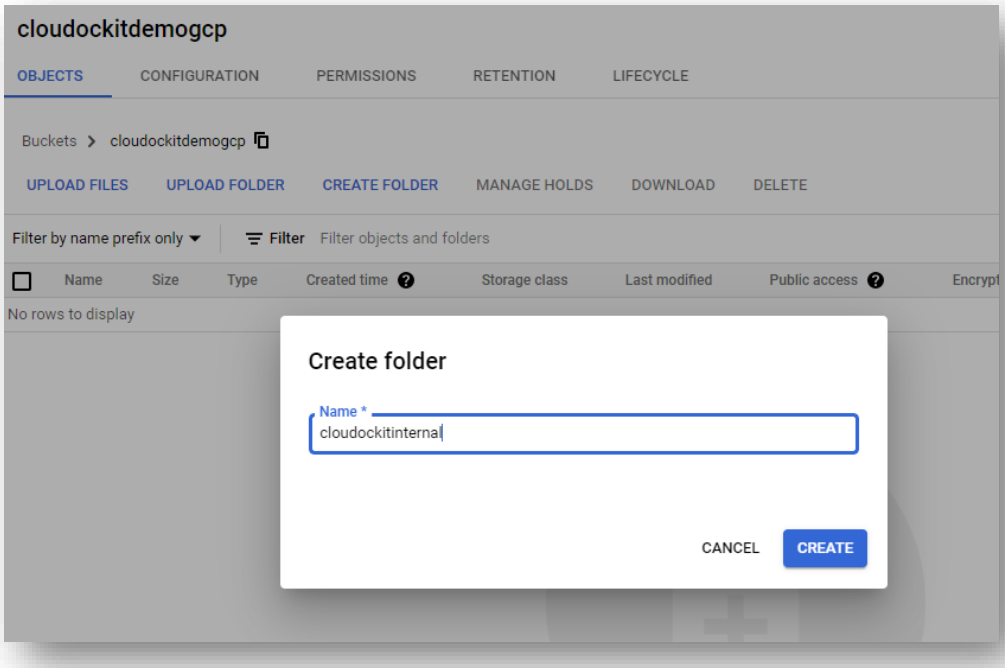
Licensing

Once you have created the bucket, you need to send the bucket name to Cloudockit Support Team (support@cloudockit.com) so that they generate a license file.

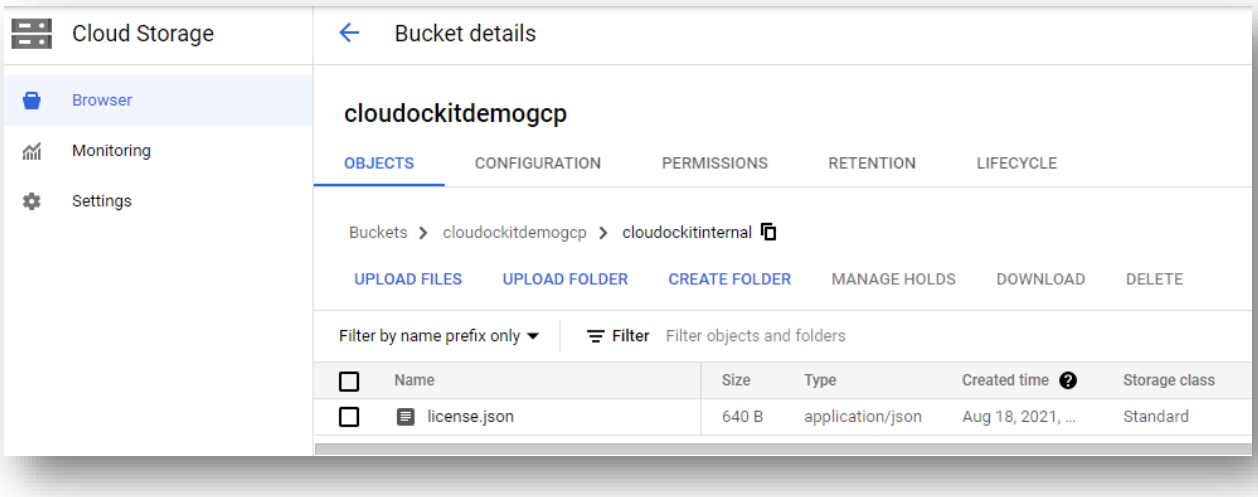
When you receive the license file, you will also get the following information:

- Product Key: used by your users who will connect to the Web UI.
- Admin Product Key: used by your admin users who will connect to the Web UI. This allows to access additional features.
- API Key: used when you want to trigger API calls
- Admin API Key: used when you want to trigger API calls. Compared to the API Key, it gives you extra features like the ability to list the Document Generation currently running and to stop running documents

Once you receive the License file (*license.json*), you need to upload that file into a folder named **cloudockitinternal** in the bucket (this folder will be automatically created for you if you have already started the container otherwise you need to create it manually):



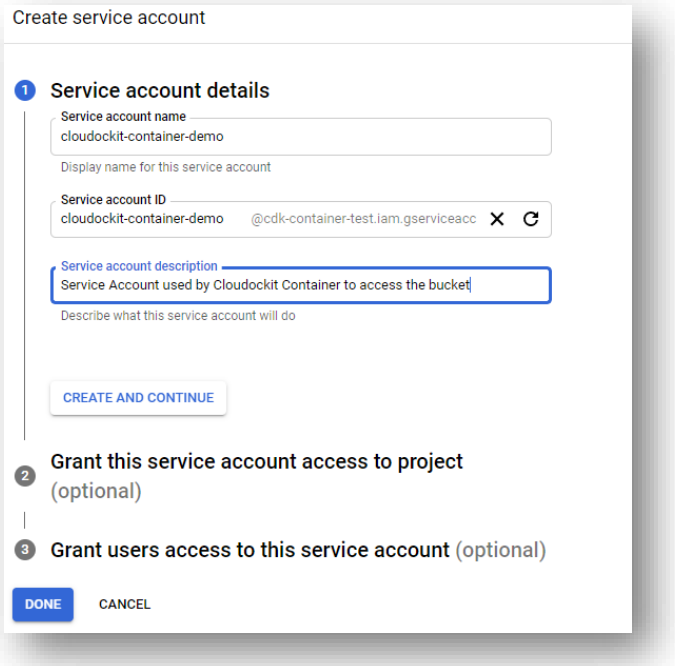
Then, upload the license.json file into that folder:



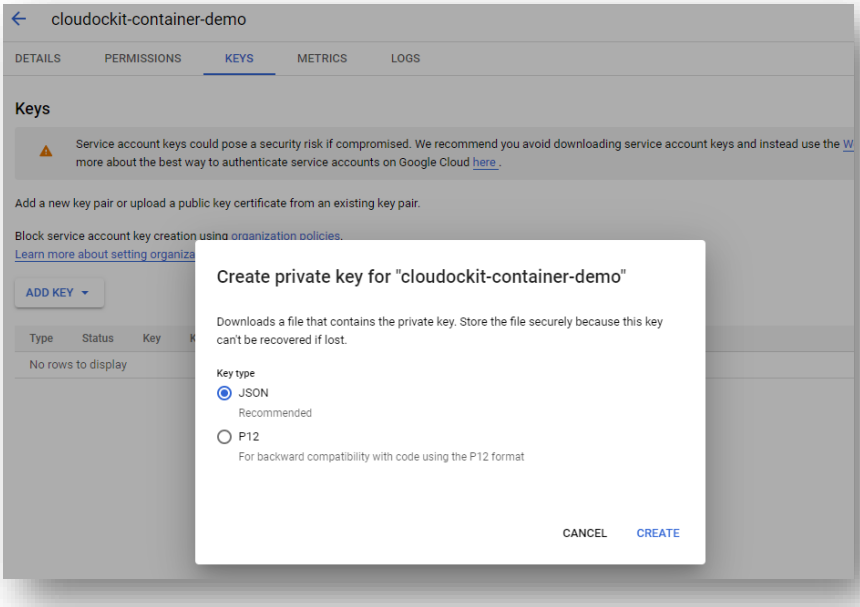
Create a service account to access the bucket

As the container needs to access the bucket to read the license file, you need to create a new service account:

From IAM menu / Service accounts, create a new service account:



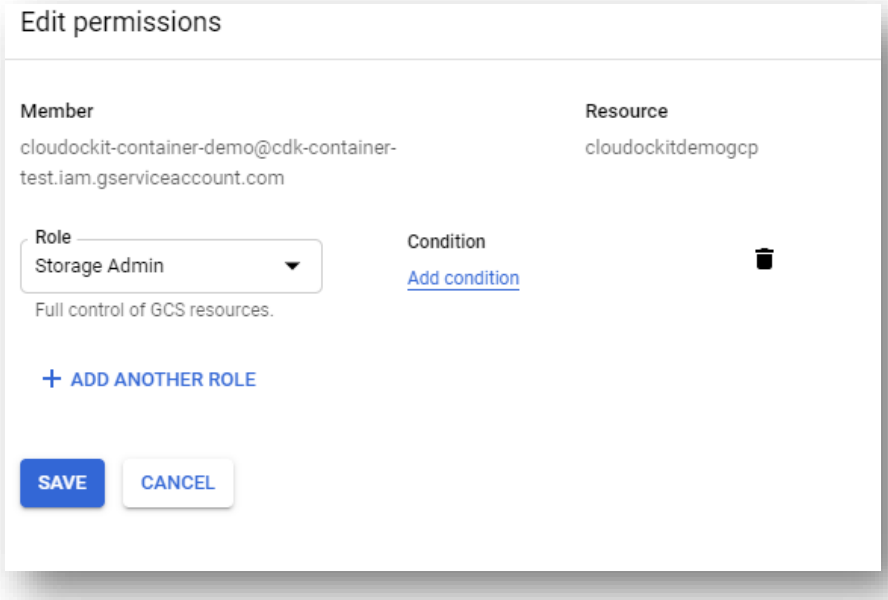
Generate a JSON Credential file for this service account as this will be required in the following step:



Open the JSON generated file and remove all Carriage Return characters to make it a single line.

Save the JSON cred file for further use.

From the bucket, give the service account the Storage Admin role on the bucket:



Step B – Create your container environment and Start your container

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

Create a Google Container Registry and Upload image

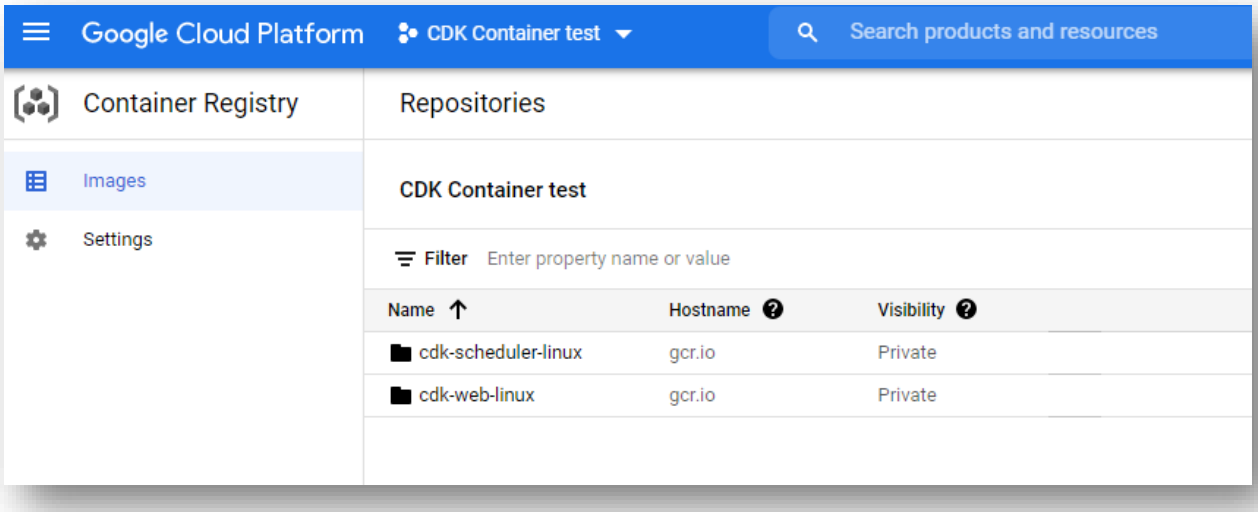
First, you need to upload the image provided by Cloudockit support team to your own Google Container Registry.

Please note the URL of the uploaded image as you will need that in the next step. Our example is using *gcr.io/projectCDKContainer/cloudockitapi*

As an example, here is a script that you can use to pull the image from Cloudockit Repo and push that to your GCP Container Registry:

```
#This script will download Cloudockit Container image and upload that to your Google Cloud Container Registry
#Please ensure that you have Docker installed to execute this script (required to pull/push Cloudockit
Container image to your GCP Registry)
#Please ensure that you have GCloud installed to execute this script (required to authenticate to your GCP
Registry)
#update the following values
$gcpProjectID = 'cdk-container-test'
$sourceRepoUser = #Login to Cloudockit Container registry. Please refer to the email you received
$sourceRepoPwd = '' #Password to Cloudockit Container registry. Please refer to the email you received

#$targetRepoUser = 'yourlogintoyourregistry' #Please go to your container registry and activate Admin user
#$targetRepoPwd = 'yourpasswordtoyourregistry'


#Internal values
$targetRepoURL = 'gcr.io/'+$gcpProjectID #This is the container registry where you want to upload. Basically
gcr.io/<PORJECT-ID>
$currentVersion = 'latest'
$type = 'linux'
$sourceRepoURL = 'cloudockitcontainerregistry.azurecr.io'

#connect to the Cloudockit Repository
docker login $sourceRepoURL -u $sourceRepoUser -p $sourceRepoPwd

#Pull the cloudockit image
docker pull $sourceRepoURL/cdk-scheduler-$type':'$currentVersion
docker pull $sourceRepoURL/cdk-web-$type':'$currentVersion

#Login to the target Repository
gcloud auth login
gcloud config set project $gcpProjectID
#docker login $targetRepoURL -u $targetRepoUser -p $targetRepoPwd

#Tag the docker image and push it to the registry
docker tag $sourceRepoURL/cdk-web-$type':'$currentVersion $targetRepoURL/cdk-web-$type':'$currentVersion |
docker push $targetRepoURL/cdk-web-$type':'$currentVersion
docker tag $sourceRepoURL/cdk-web-$type':'$currentVersion $targetRepoURL/cdk-web-$type':latest' | docker push
$targetRepoURL/cdk-web-$type':latest'
docker tag $sourceRepoURL/cdk-scheduler-$type':'$currentVersion $targetRepoURL/cdk-scheduler-
$type':'$currentVersion | docker push $targetRepoURL/cdk-scheduler-$type':'$currentVersion
docker tag $sourceRepoURL/cdk-scheduler-$type':'$currentVersion $targetRepoURL/cdk-scheduler-$type':latest' |
docker push $targetRepoURL/cdk-scheduler-$type':latest'
```

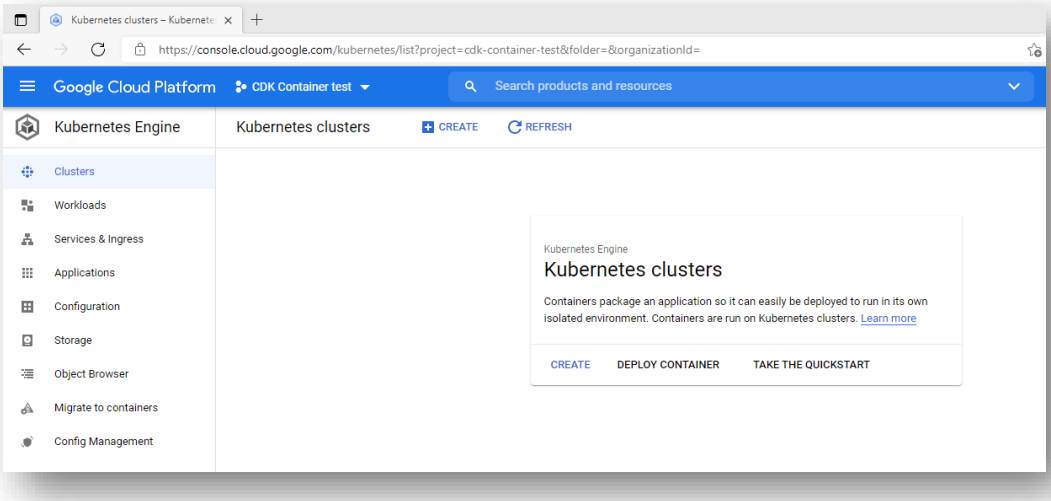
Then, validate that the images have been uploaded successfully:



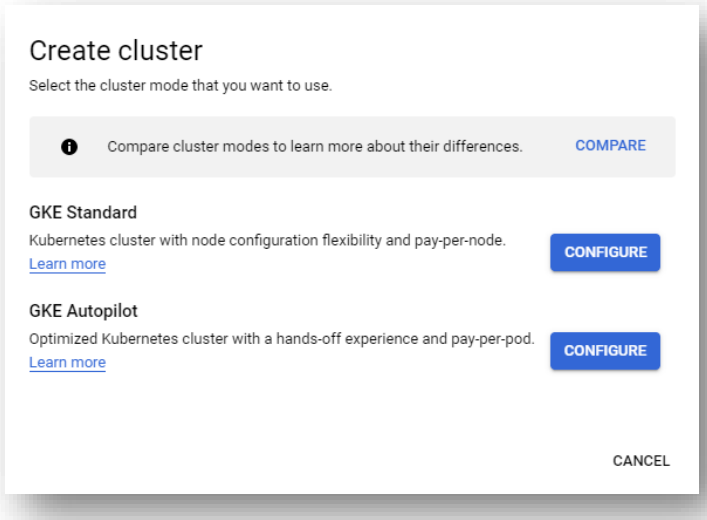
Create a Kubernetes Cluster

First, you need to create a Kubernetes Cluster using Google Kubernetes Engine. You can use your own procedure to do that or refer to [Step 1: Create a GKE cluster | Apigee | Google Cloud](#).

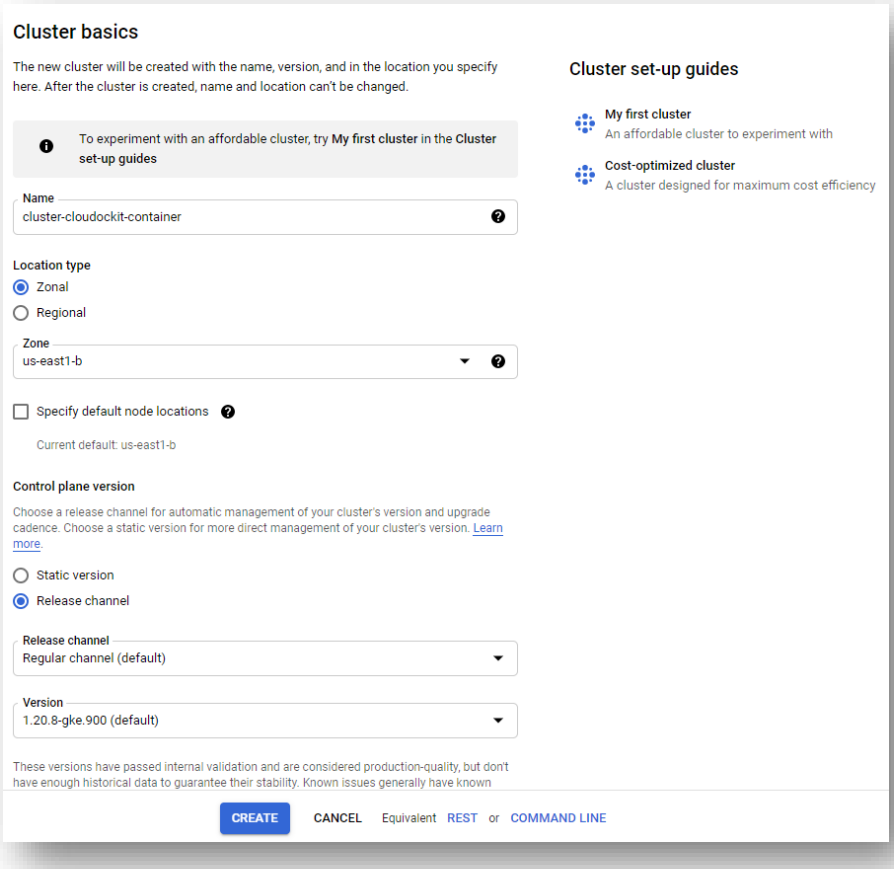
Click on Create



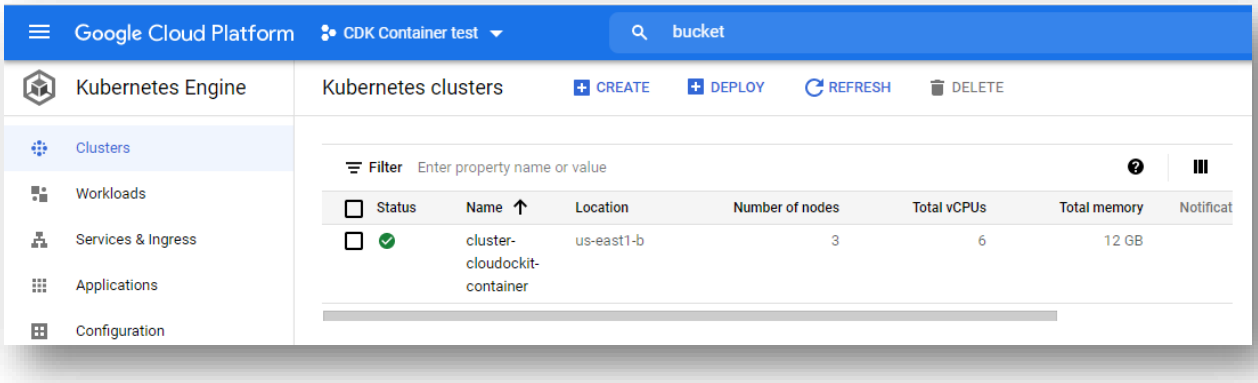
Choose a GKE Standard Cluster:



Enter the cluster name and appropriate region and then click on Create :

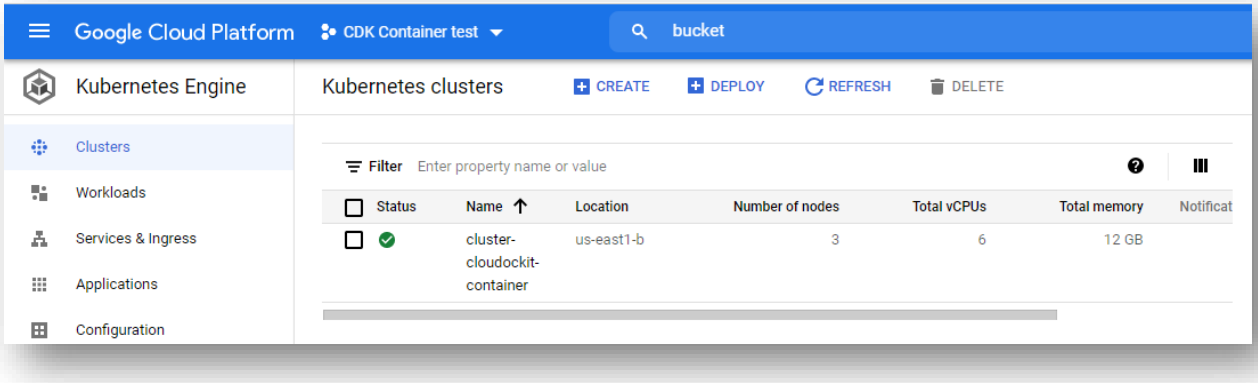


Then click on Create and wait for the cluster to be created:

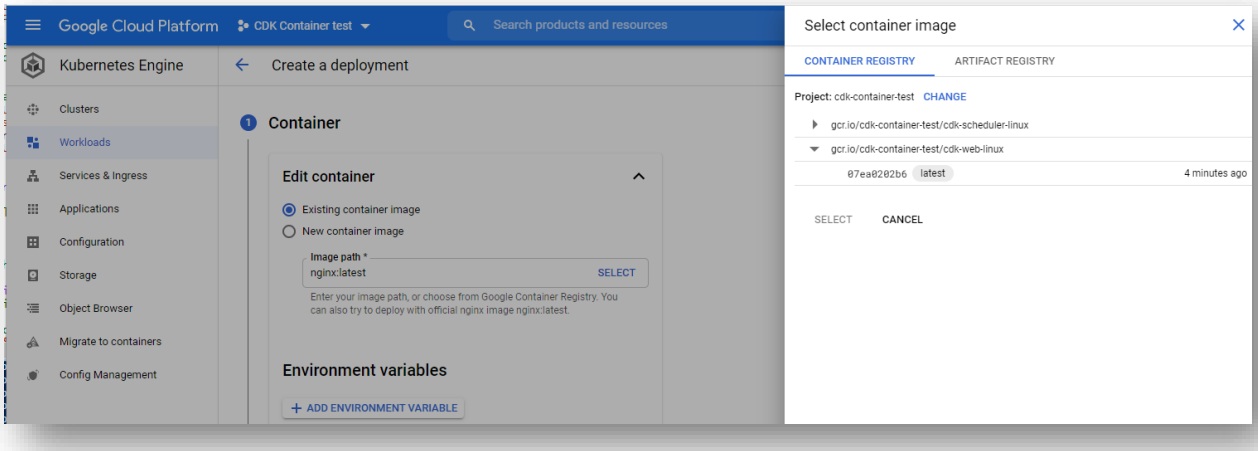


Deploy Cloudockit Container

Click on Deploy:



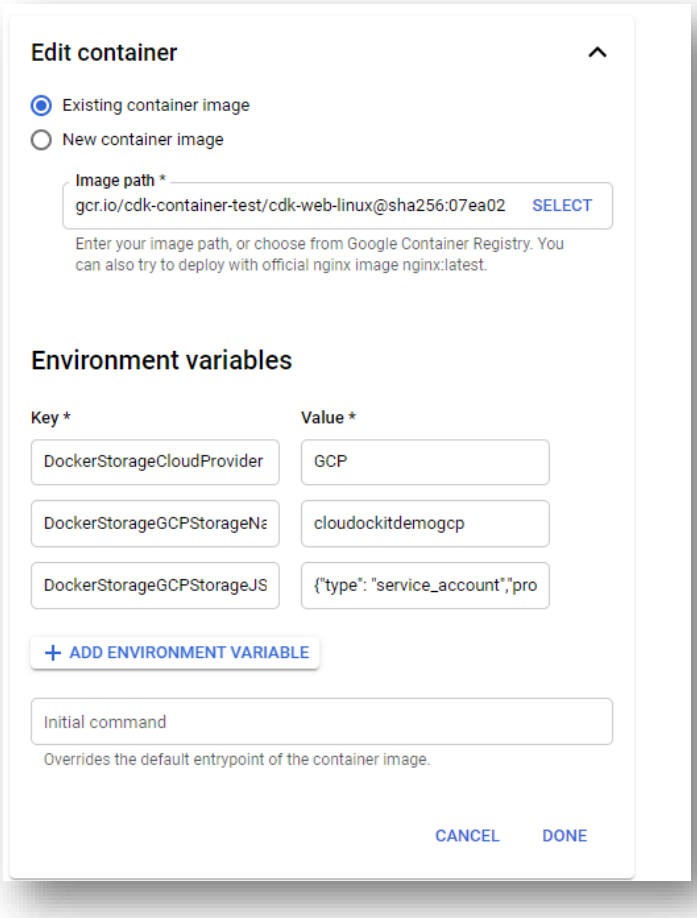
Select the cdk-web-linux container:



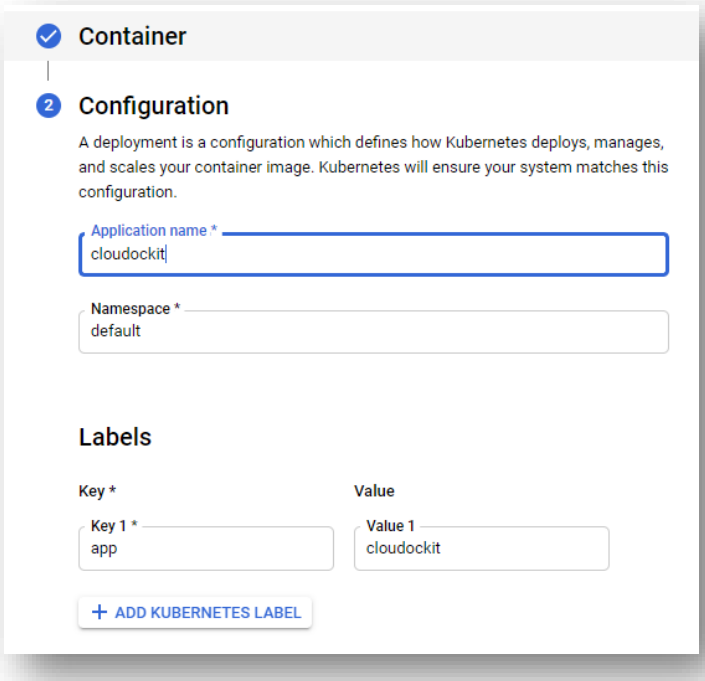
Enter the following environment variables and click Done

Name	Value
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
DockerStorageGCPStorageName	Enter the GCP Bucket Name
DockerStorageGCPStorageJSONCredentials	Enter the JSON Credentials with Full Control on the GCP Bucket
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.

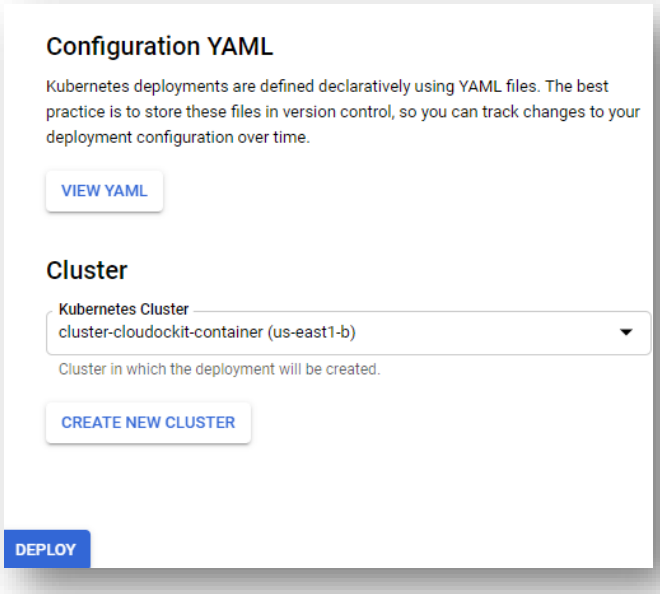
Please ensure that you have the complete JSON credentials on a one line for the Env variable DockerStorageGCPStorageJSONCredentials.



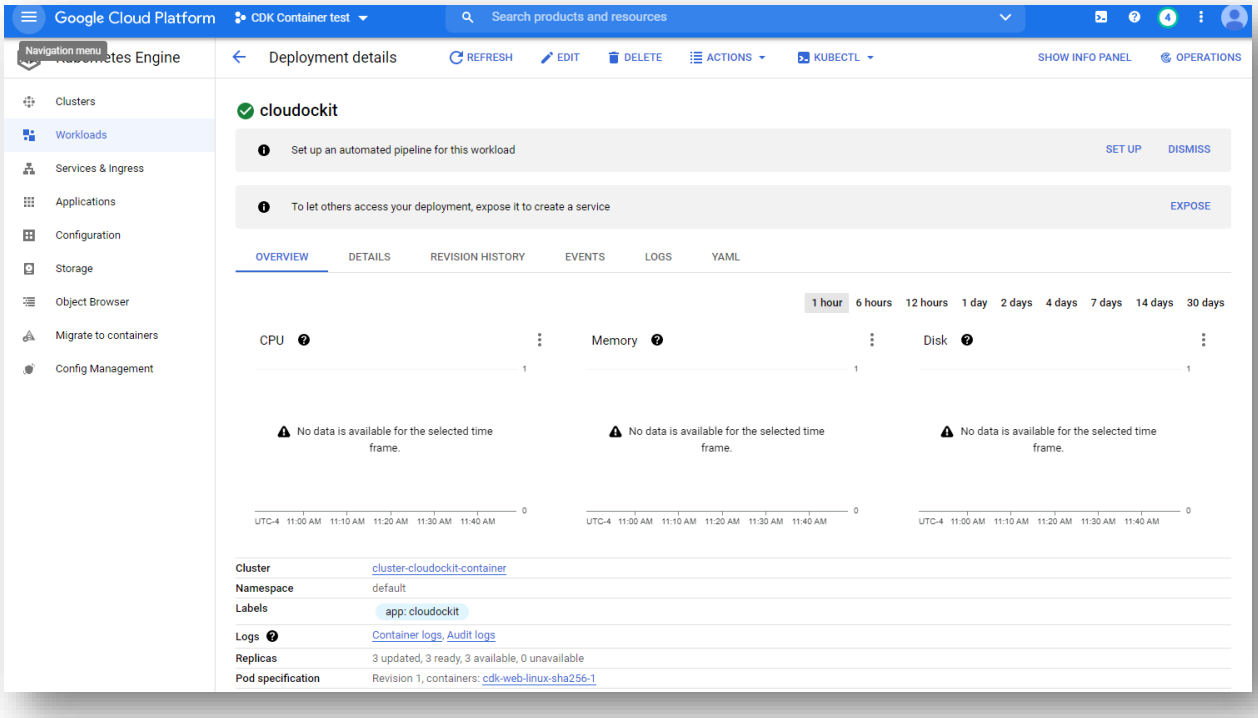
Then, in configuration, enter the following information



Then, click Deploy:

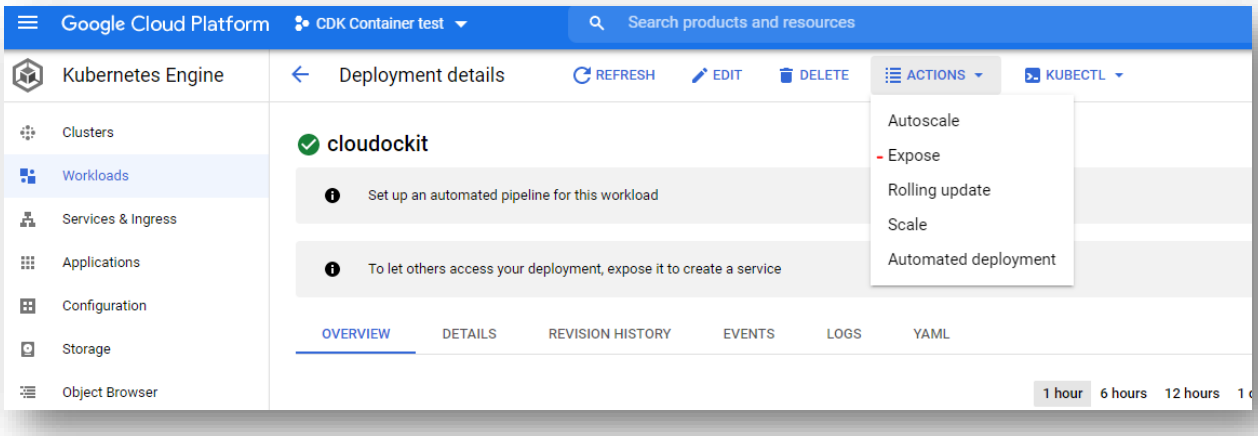


Wait a few minutes and validate that the deployment is done:

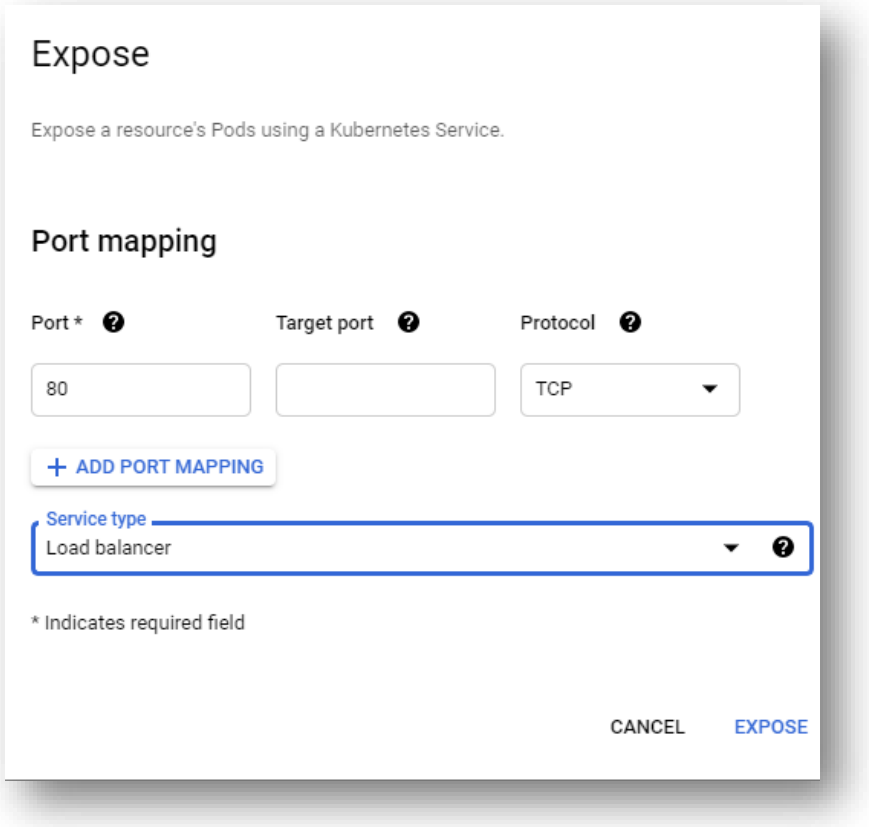


Then, expose the container with port 80 (for test only, you should use https/443 with appropriate certificates for production deployment):

Click on Actions / Expose:

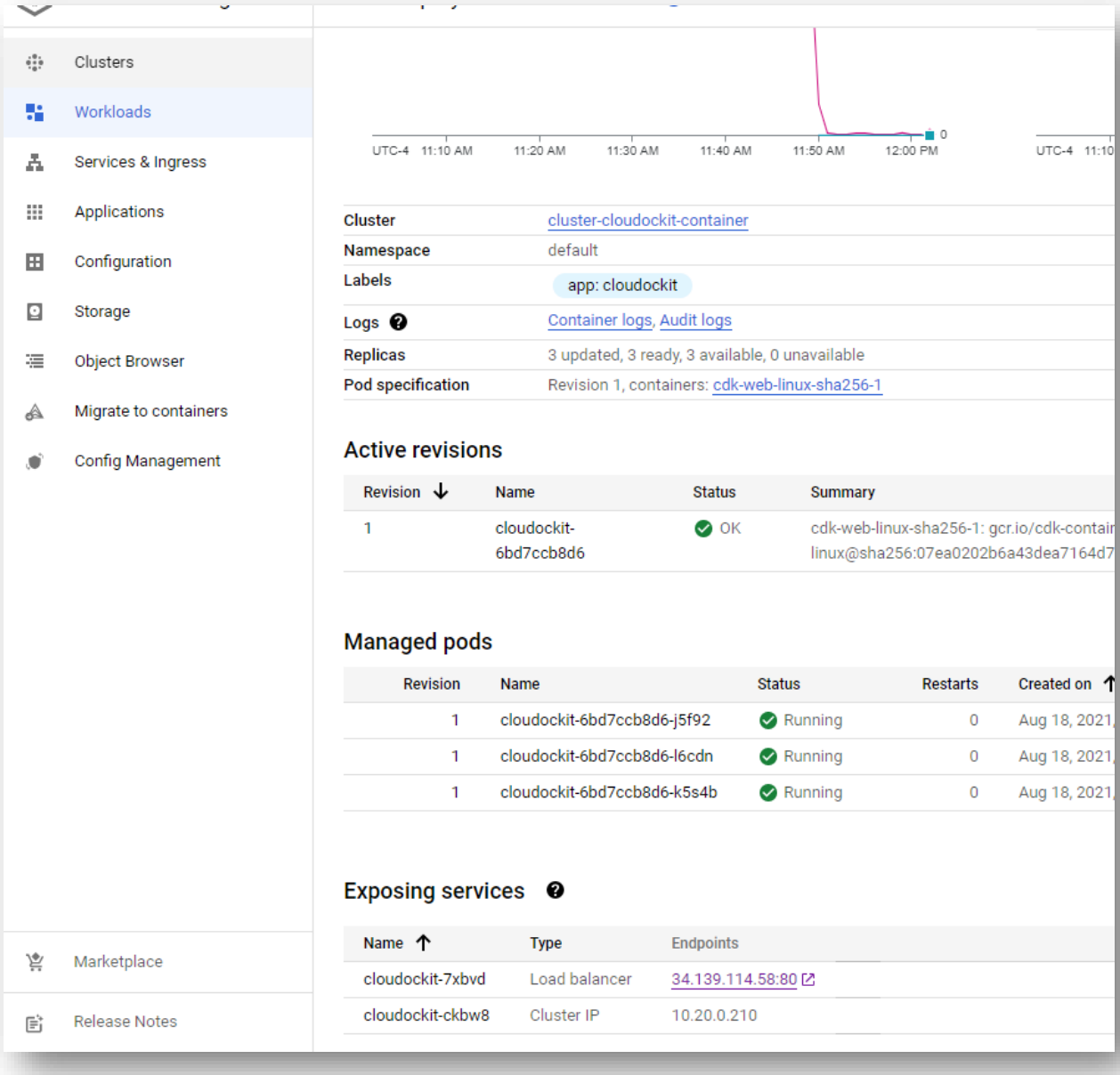


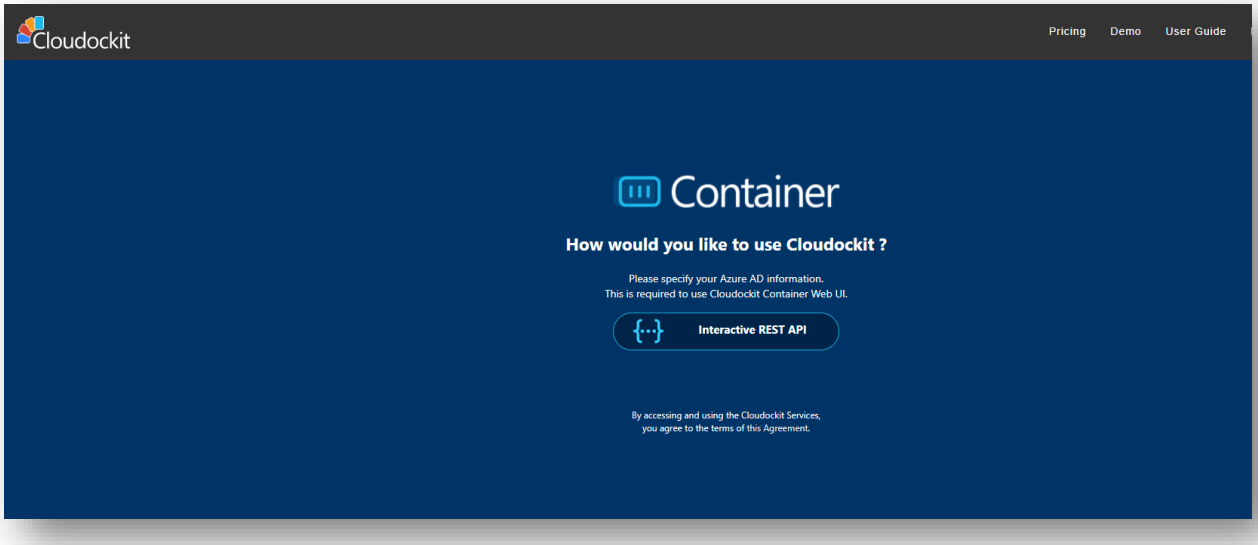
Then, click Expose:



Ensure that you select service type = Load balancer if you want to have external communications allowed to the cluster. If you select Cluster IP instead, ensure that you have internal network connectivity property setup.

Then, from the exposing Services section, click on the EndPoint near the Load Balancer type and validate that the container is up and running:




Cloudockit
Pricing
Demo
User Guide
Container
How would you like to use Cloudockit ?
Please specify your Azure AD information.
This is required to use Cloudockit Container Web UI.
Interactive REST API
By accessing and using the Cloudockit Services,
you agree to the terms of this Agreement.

Step C (Optional) – Configure Cloudockit Web UI

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

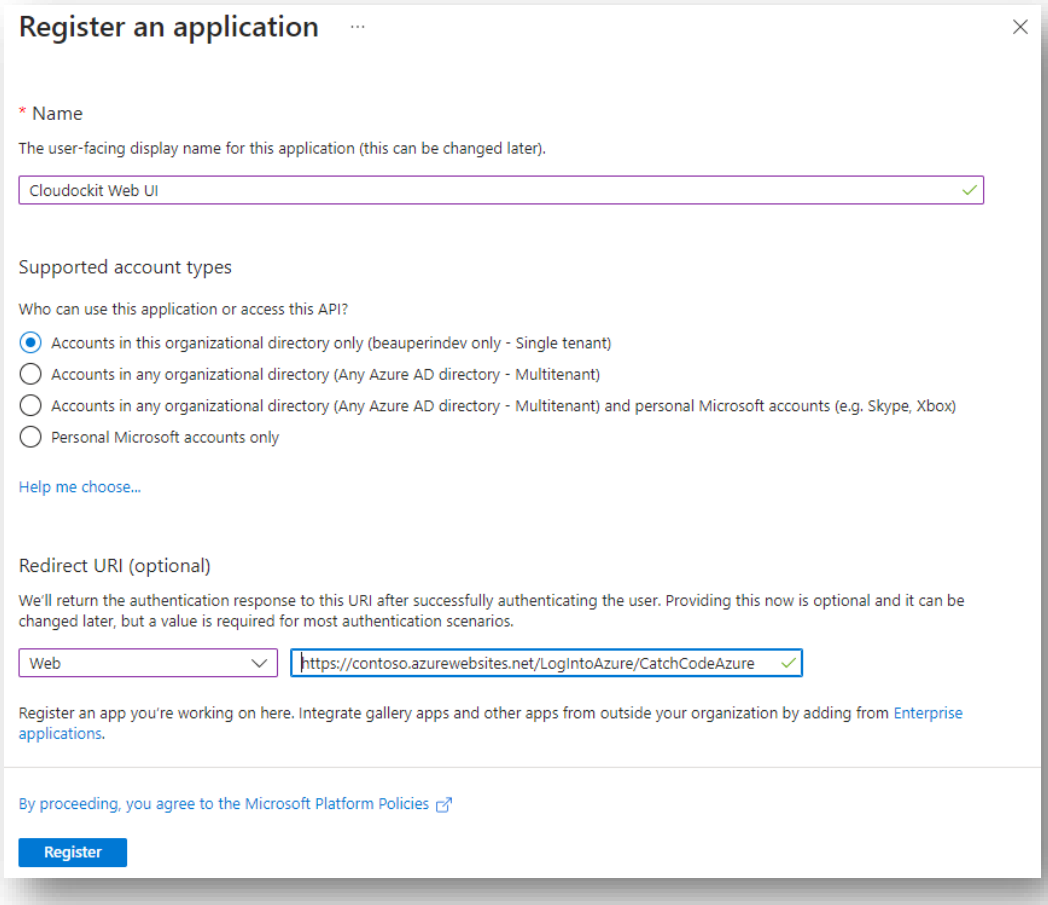
This Web UI supports Azure Active Directory as a first step to authenticate users.

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).

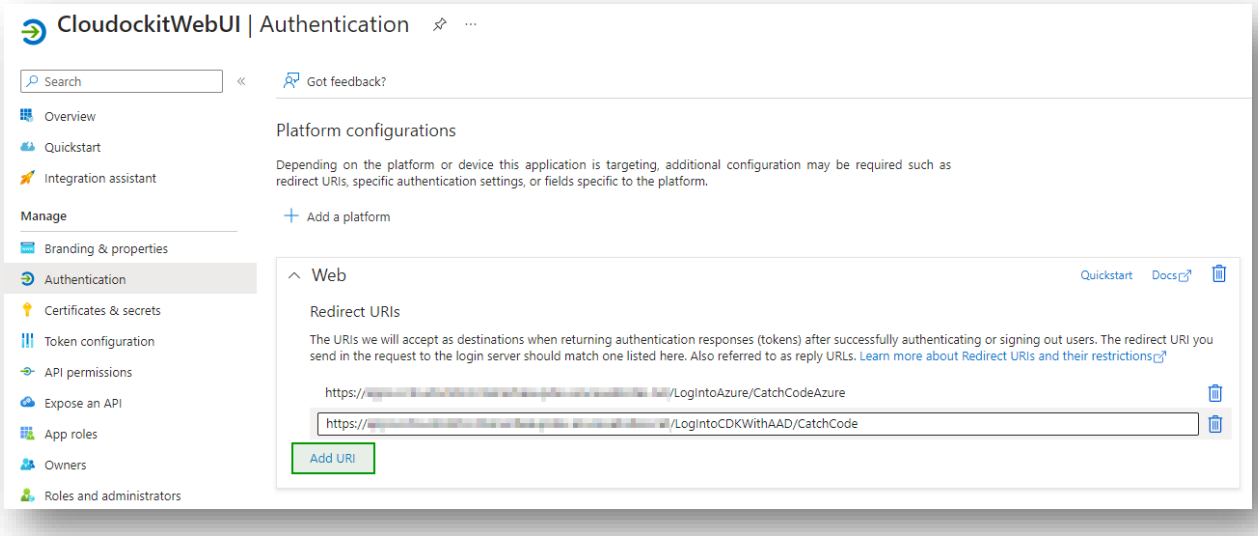
To activate Azure AD Authentication, you need to follow these steps:

- Go to your **Azure Active Directory**
- 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

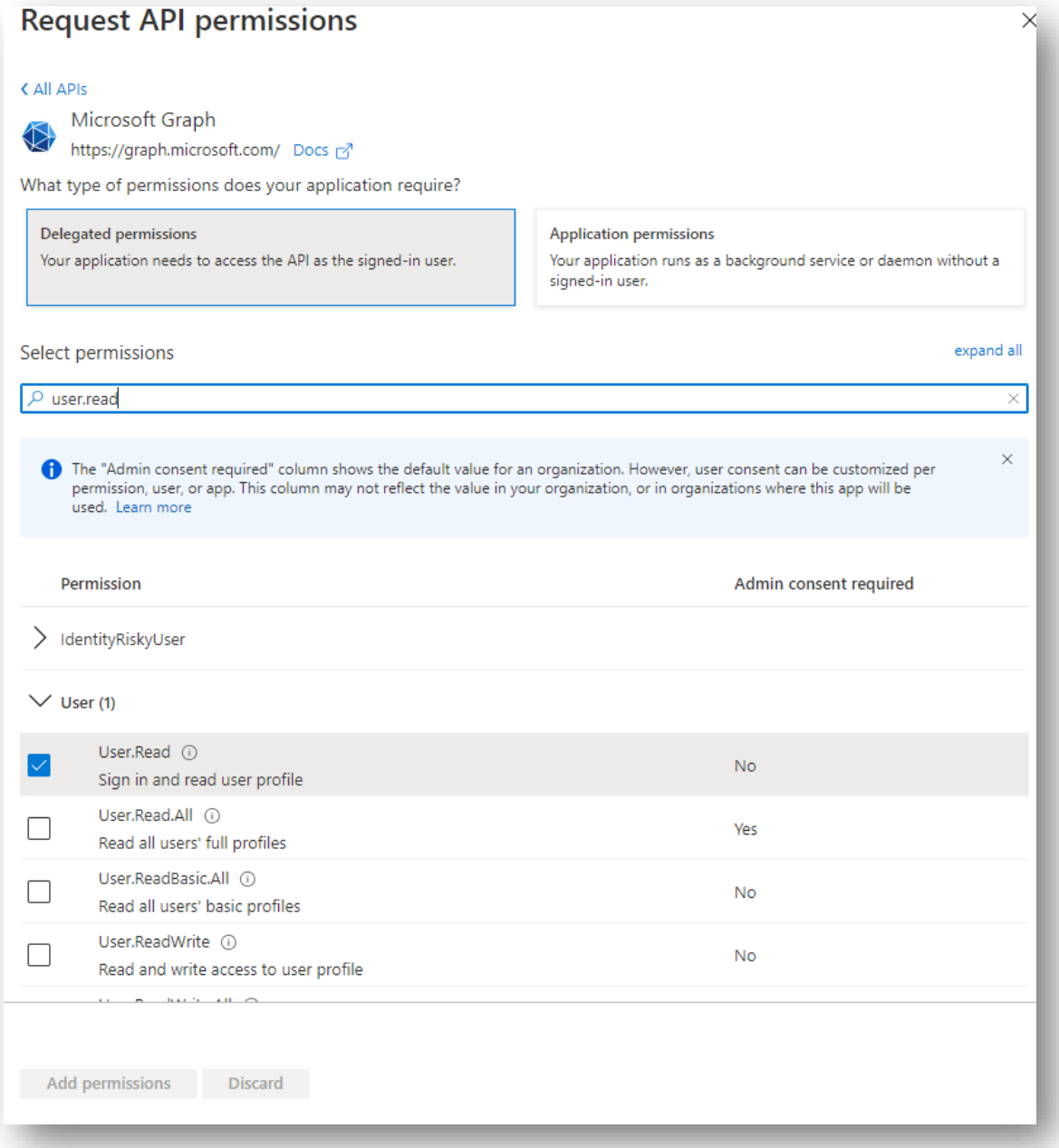


Note: the interface will not let you enter the 2nd URL before clicking on **Register** so you'll have to enter it after registration, in the Authentication page:

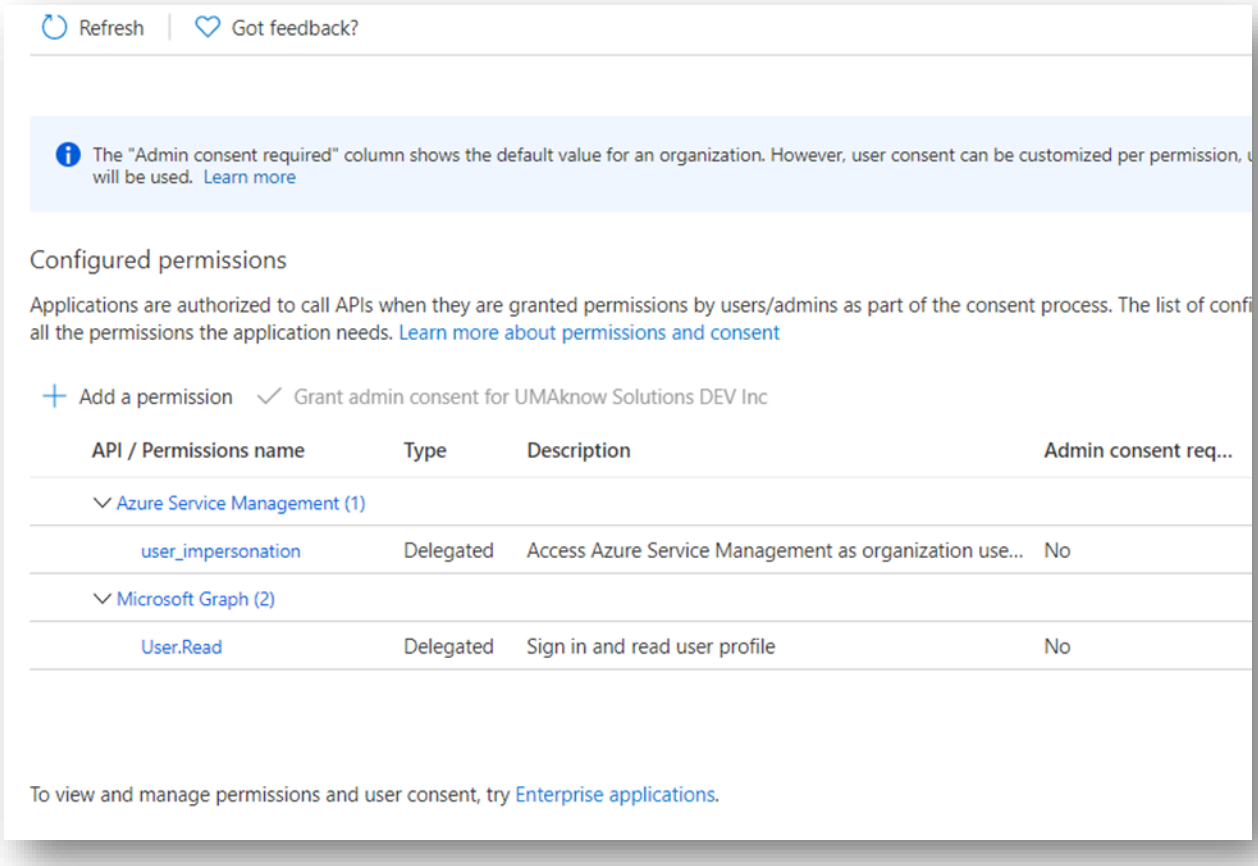


Then, go to **API Permissions**, click on **+Add a permission** and select:

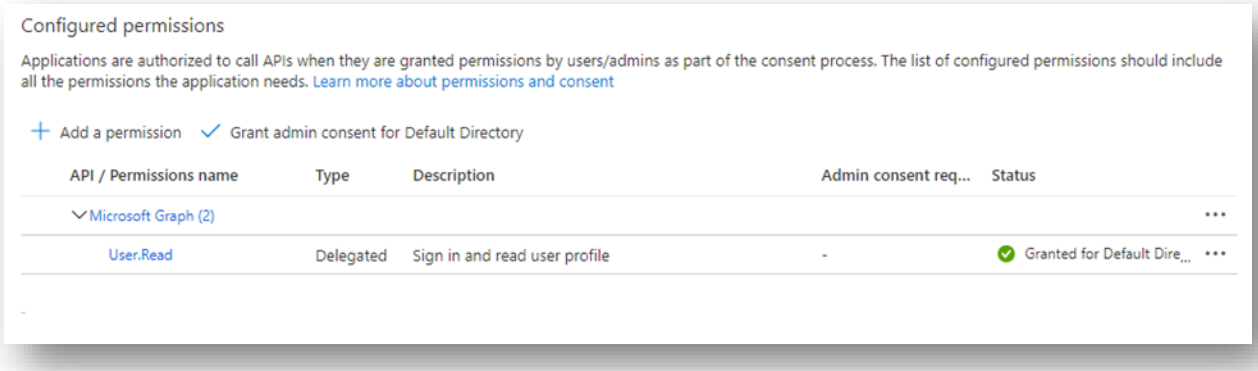
- **Microsoft Graph**, then **Delegated permissions** and then select ***User.Read***:
- **Azure Service Management**, then **Delegated permissions** and then select ***user_impersonation***:



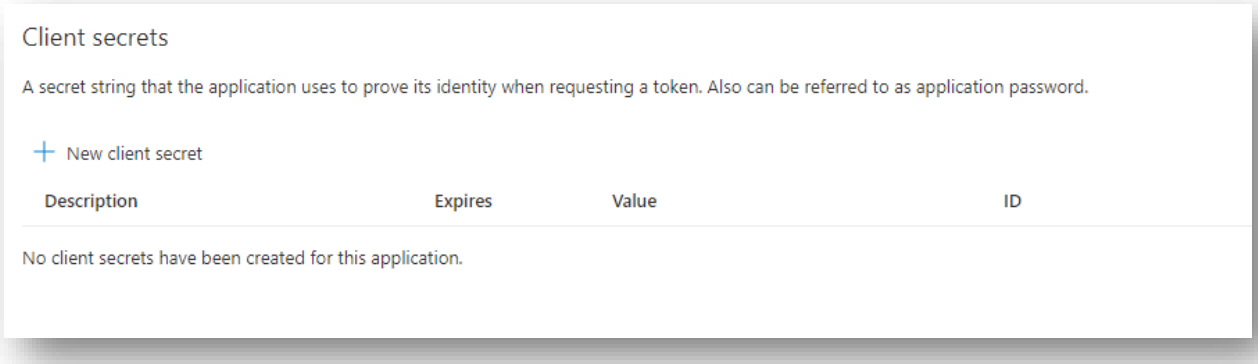
Click **Add permissions**. You should now see the following:



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"
}
```

Step D (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 to access to the storage account where are 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 • **GCP** • AWS
DockerStorageGCPStorageName	Enter the GCP Bucket Name
DockerStorageGCPStorageJSONCredentials	Enter the JSON Credentials with Full Control on the GCP Bucket
DockerUrlForSchedulingStarts	Enter the URL of your API that host 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.

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

Cloudockit Container now 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 Diagram.

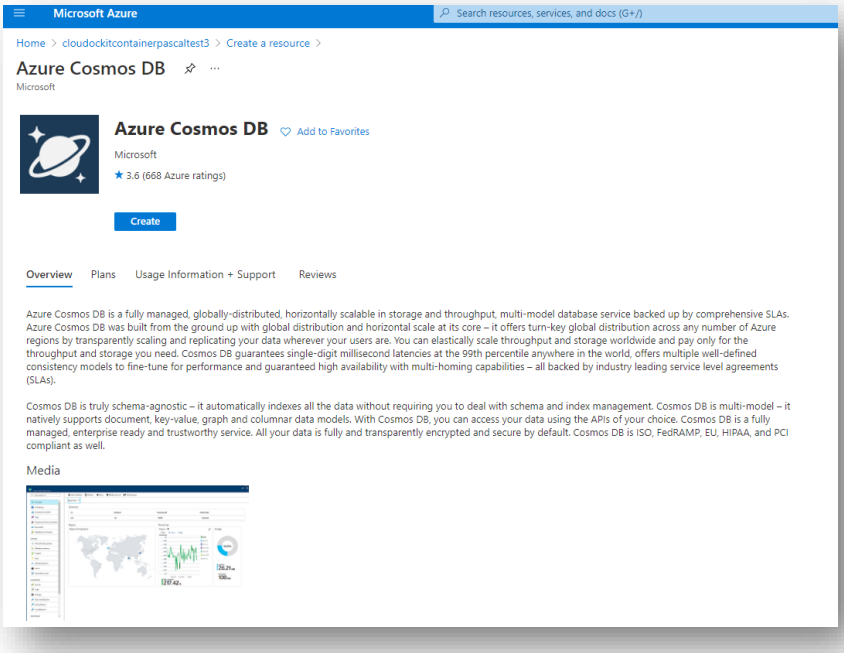
There are two steps required:

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

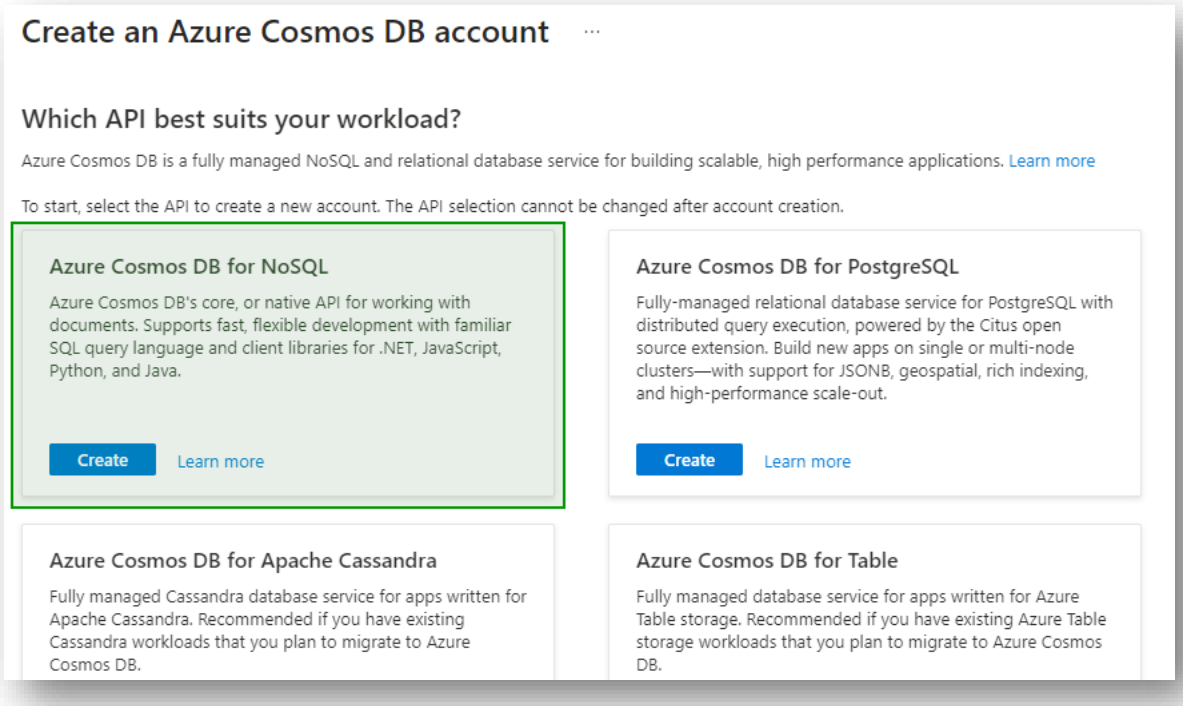
Create (or re-use) an Azure Cosmos DB

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)

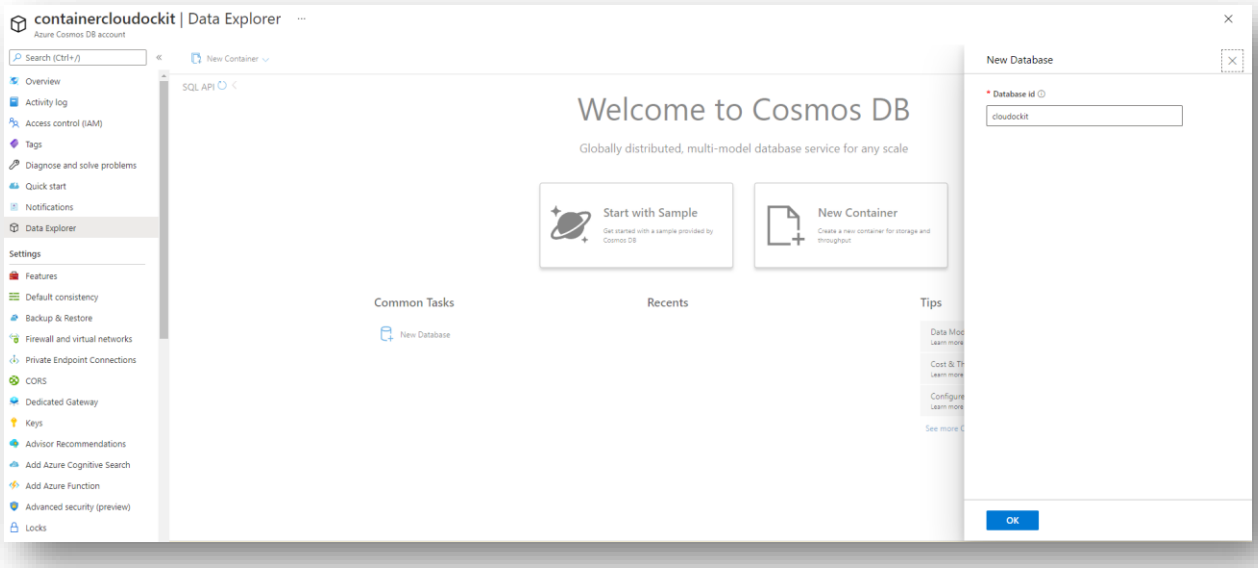
- Create a Cosmos DB



- 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:

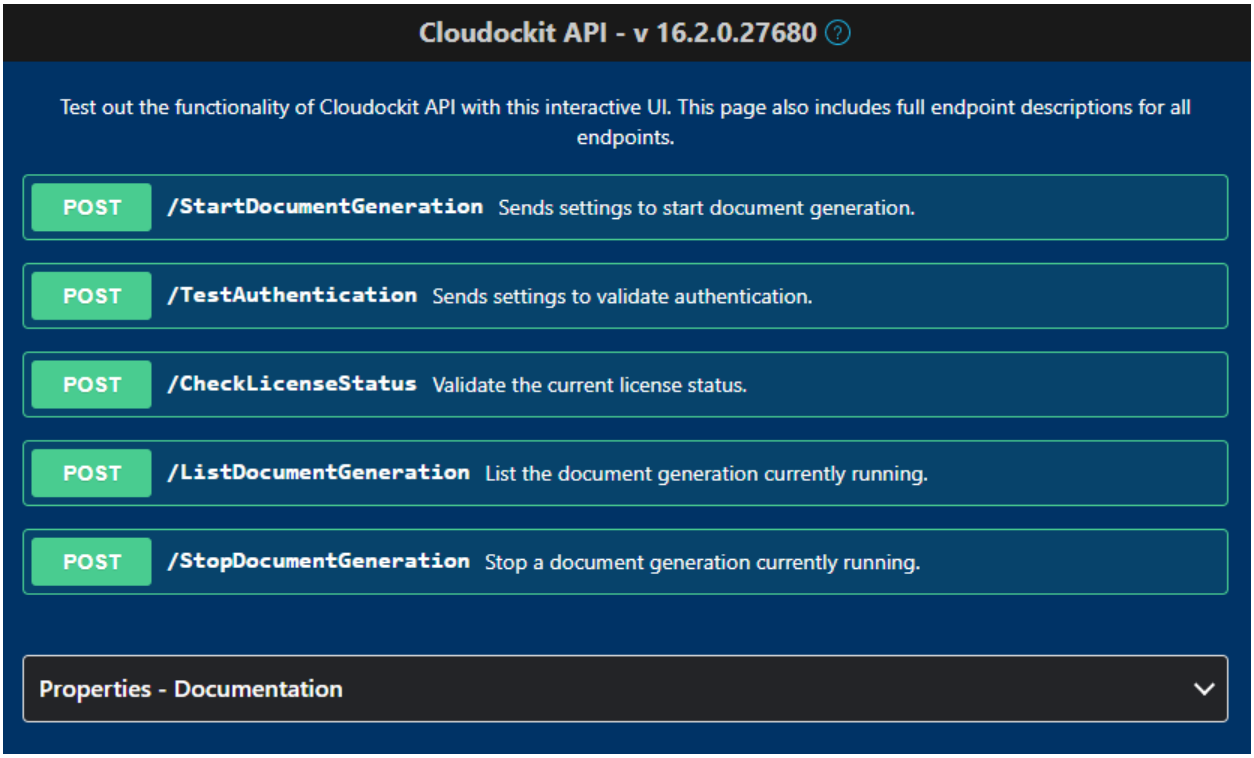
Name	Value
CosmosDb__DatabaseName	Enter the name of the Database that you have created in the previous step (cloudockit in the example)
ConnectionStrings__CosmosDb	Azure CosmosDB Connection string

Step 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

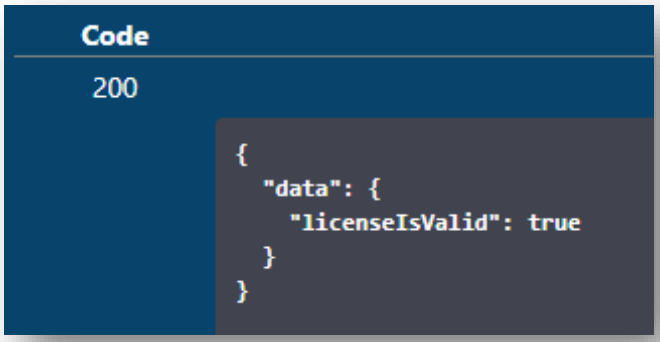
Once you get the API Key from Cloudockit team and you have the appropriate credentials for the license validation, you can check that your API Key is working by using the **/CheckLicenseStatus** endpoint.

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	AAD App ID for the scan
	AppClientKeyForAutomation	AAD App Key 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 still call AzureStorageNameForDropOff** You should specify one of this value: • the Azure Storage Account Name (it can be storage short 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

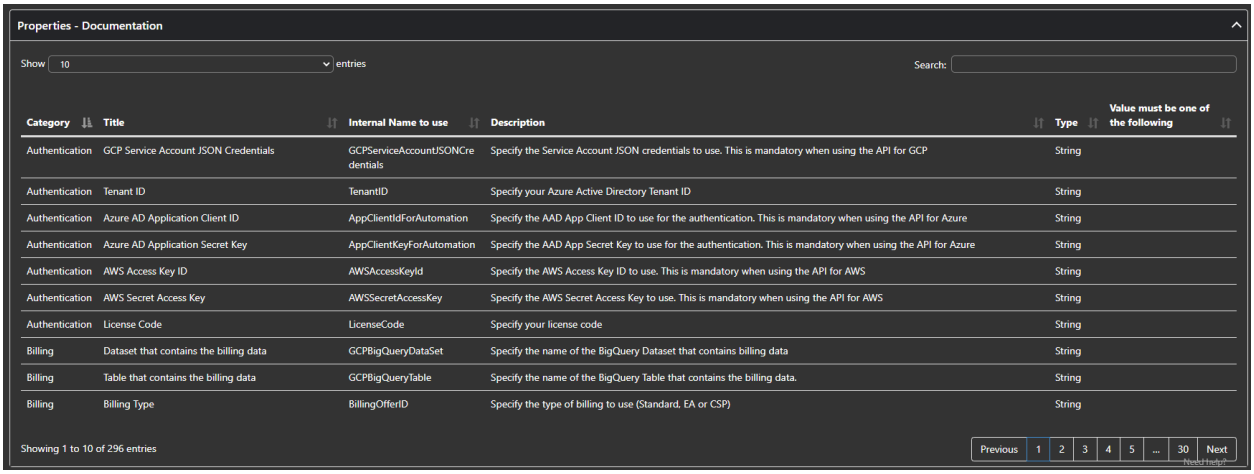
Once all the tests above have been done, you can start the document generation.

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:

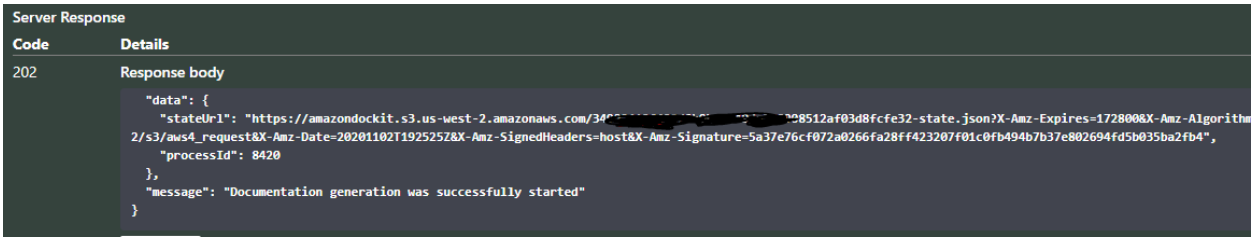
Category	Title	Internal Name to use	Description	Type	Value must be one of the following
Authentication	GCP Service Account JSON Credentials	GCPServiceAccountJSONCredentials	Specify the Service Account JSON credentials to use. This is mandatory when using the API for GCP	String	
Authentication	Tenant 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.

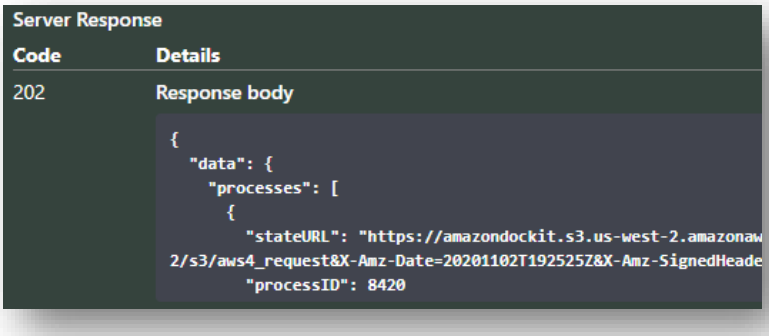
Step J – Manage your document generation (Preview)

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

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

/ListDocumentGeneration

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

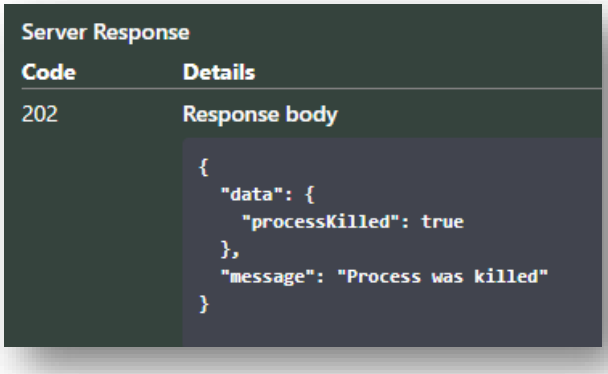


/StopDocumentGeneration

This endpoint is used to kill a document generation running.

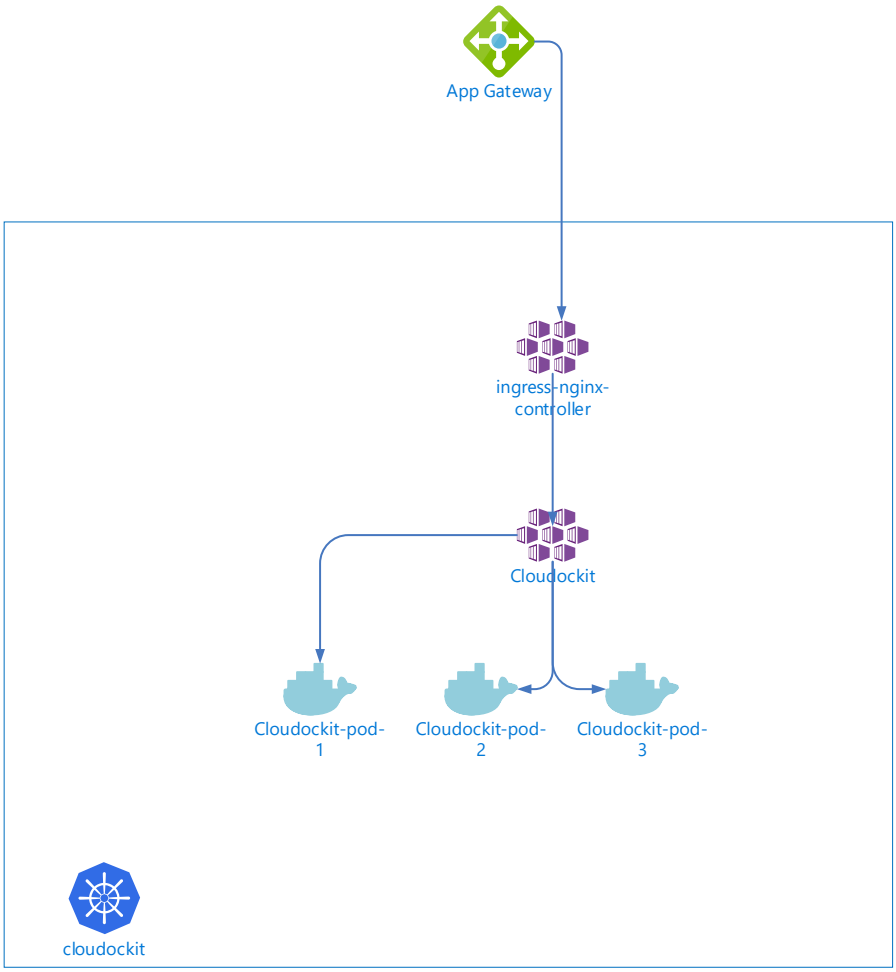
Name	Value
DockerProcessToKill	Value of the process ID to kill

It will give you back 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 have a mounted volume that will be accessible from all pods where the data protection key and certificates will be stored.

You need to define the following environment variables:

Name	Description	Example
DataProtection__KeysStorePath	Path of the volume mounted on all pods	/etc/cloudockit
DataProtection__Certificates__Current__FileName	Filename of the certificate used to encrypt the key	mycert.pfx
DataProtection__Certificates__Current__Password	Password of the certificate used to encrypt the key	xxxxx

DataProtection__Certificates__Previous__FileName	Filename of the certificate used to encrypt the key (for rotation)	Mycert2.pfx
DataProtection__Certificates__ Previous __Password	Password of the certificate used to encrypt the key (for rotation)	xxxx
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.	

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 AAD 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` and 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.