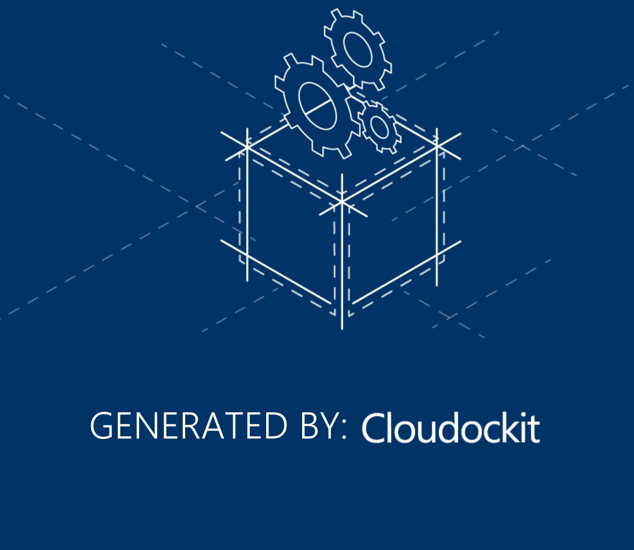
Cloudockit

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:

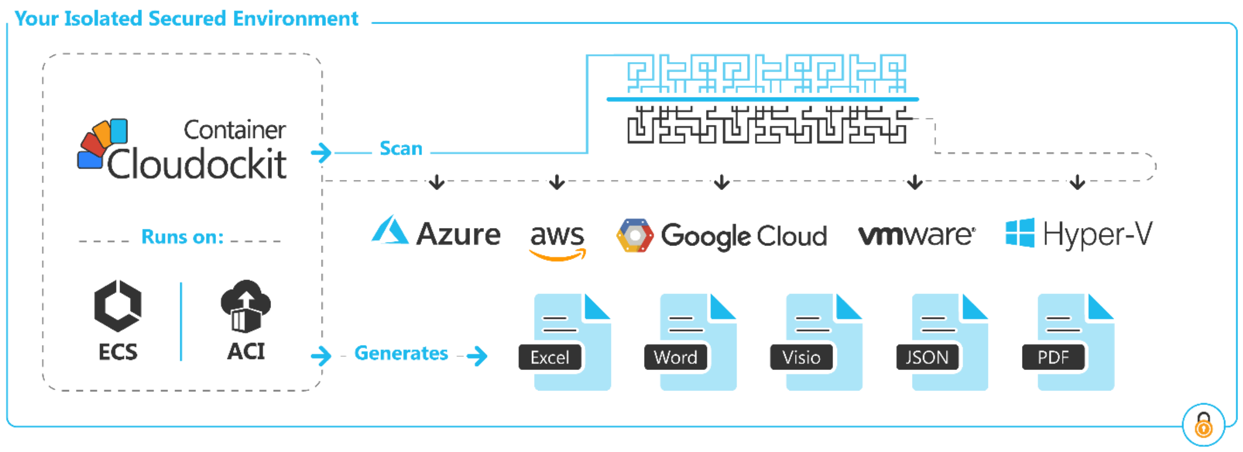
- **cdk-web-linux** that contain the Cloudockit API/Web interface. This is mandatory to run this container.
- **cdk-scheduler-linux** that contain the Cloudockit Scheduling features. This is an optional container you do not need to install if you do not want to use schedules.

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

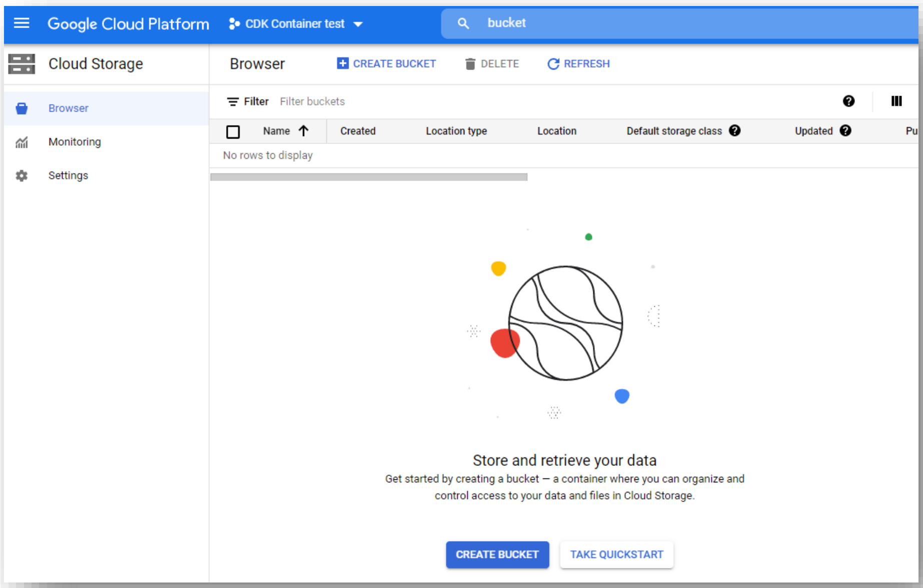
To install Cloudockit Container in your environment, you will need:

- 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.

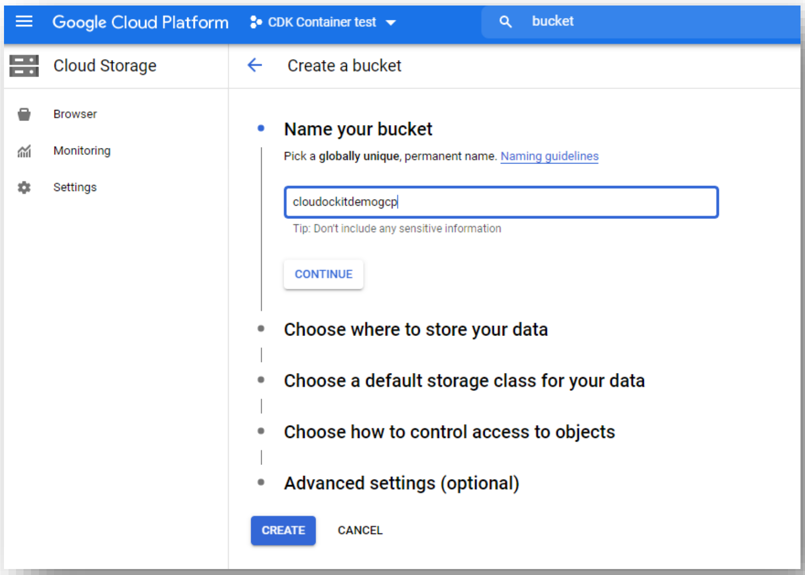


No rows to display

Store and retrieve your data

Get started by creating a bucket — a container where you can organize and control access to your data and files in Cloud Storage.

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:



Name your bucket

Pick a **globally unique**, permanent name. Naming guidelines

cloudockitdemogcp

Tip: Don't include any sensitive information

Choose where to store your data

Choose a default storage class for your data

Choose how to control access to objects

Advanced settings (optional)

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

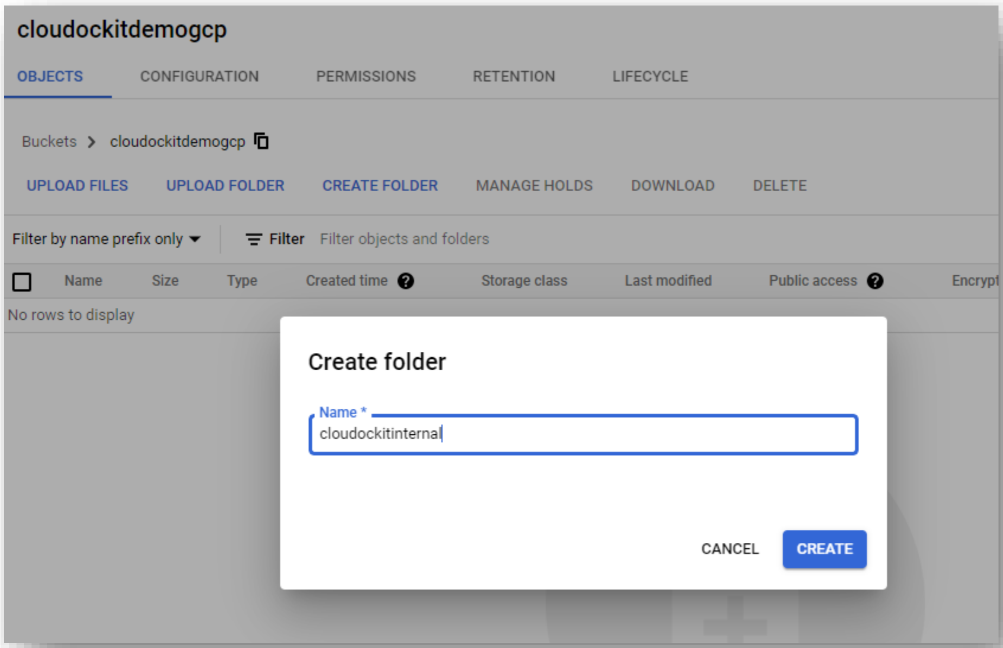
Licensing

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):

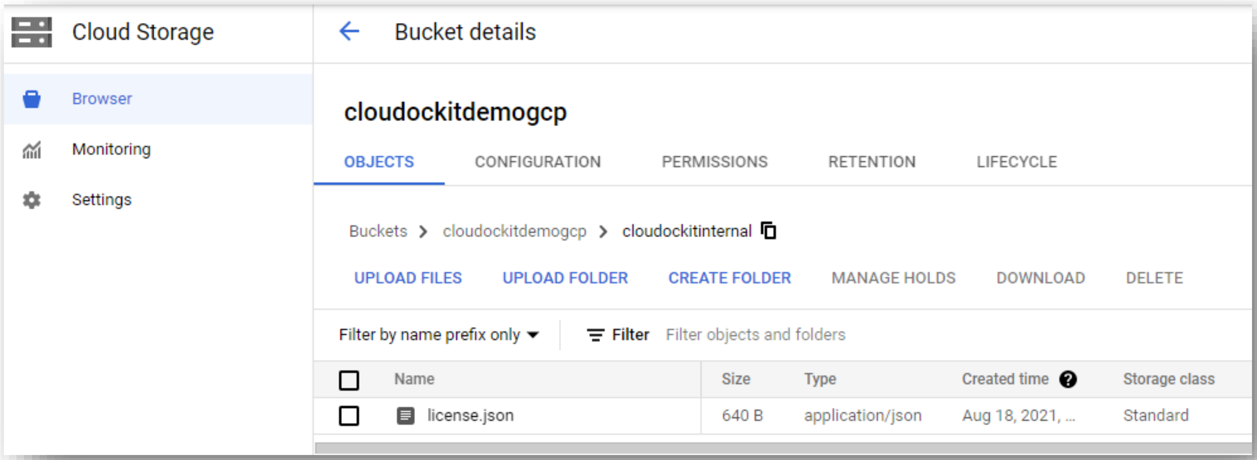


No rows to display

Create folder

Name *
cloudockitinternal

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

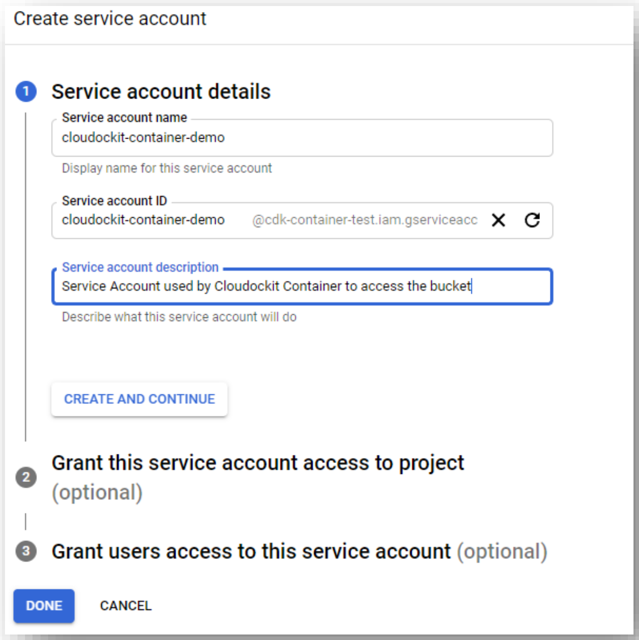


Name	Size	Type	Created time	Storage class
license.json	640 B	application/json	Aug 18, 2021, ...	Standard

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:



Create service account

1 Service account details

Service account name: cloudockit-container-demo

Display name for this service account

Service account ID: cloudockit-container-demo @cdk-container-test.iam.gserviceacc

Service account description: Service Account used by Cloudockit Container to access the bucket

Describe what this service account will do

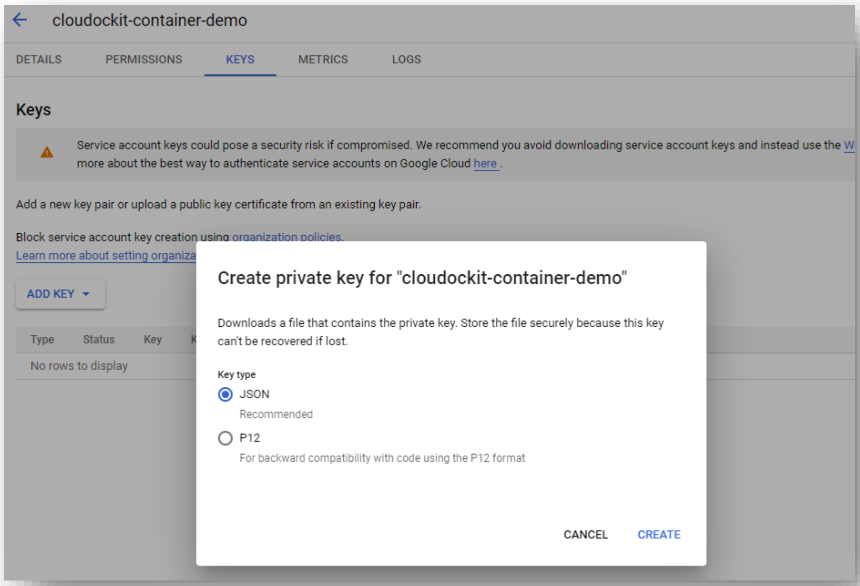
CREATE AND CONTINUE

2 Grant this service account access to project (optional)

3 Grant users access to this service account (optional)

DONE CANCEL

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



cloudockit-container-demo

DETAILS PERMISSIONS KEYS METRICS LOGS

Keys

Service account keys could pose a security risk if compromised. We recommend you avoid downloading service account keys and instead use the W
more about the best way to authenticate service accounts on Google Cloud here .

Add a new key pair or upload a public key certificate from an existing key pair.

Block service account key creation using organization policies.
Learn more about setting organiza

ADD KEY

Type Status Key
No rows to display

Create private key for "cloudockit-container-demo"

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

Key type

[✓ JSON] Recommended

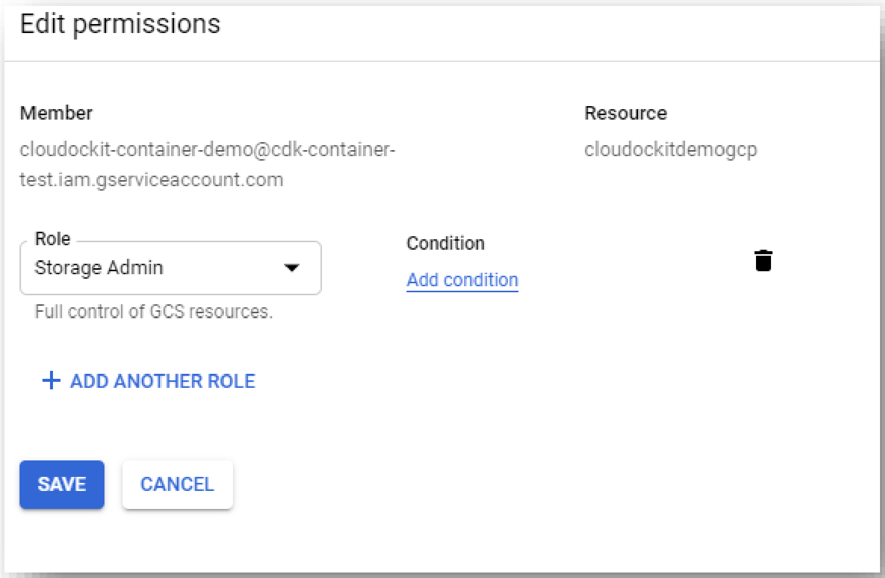
P12 For backward compatibility with code using the P12 format

CANCEL CREATE

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:



Edit permissions

Member
cloudockit-container-demo@cdk-container-test.iam.gserviceaccount.com

Resource
cloudockitdemogcp

Role: Storage Admin
Full control of GCS resources.

Condition

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

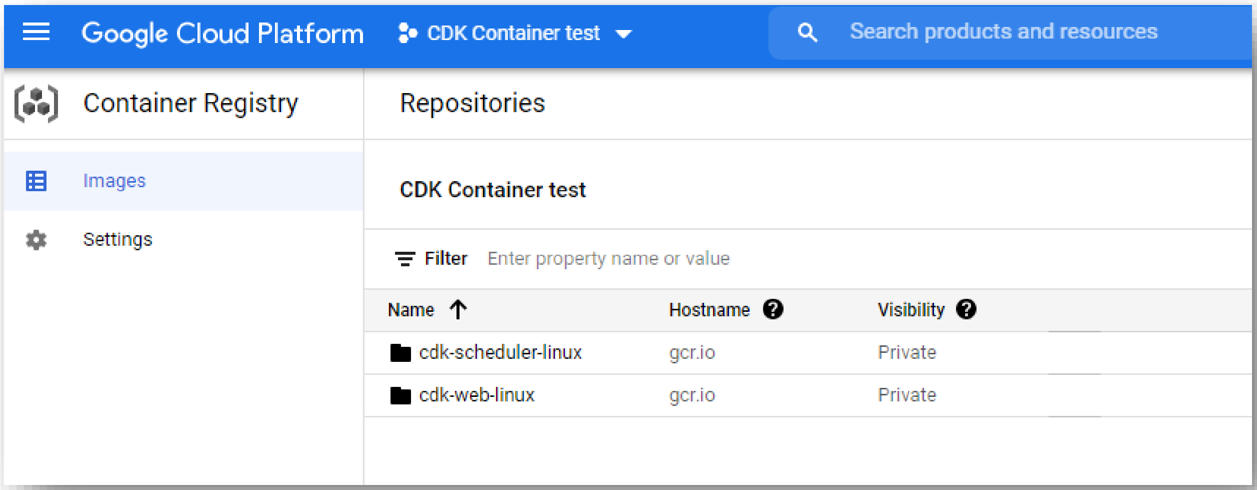
First, you need to upload the image provided by Cloudockit support team to your own Google Container Registry.

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:



Repositories

CDK Container test

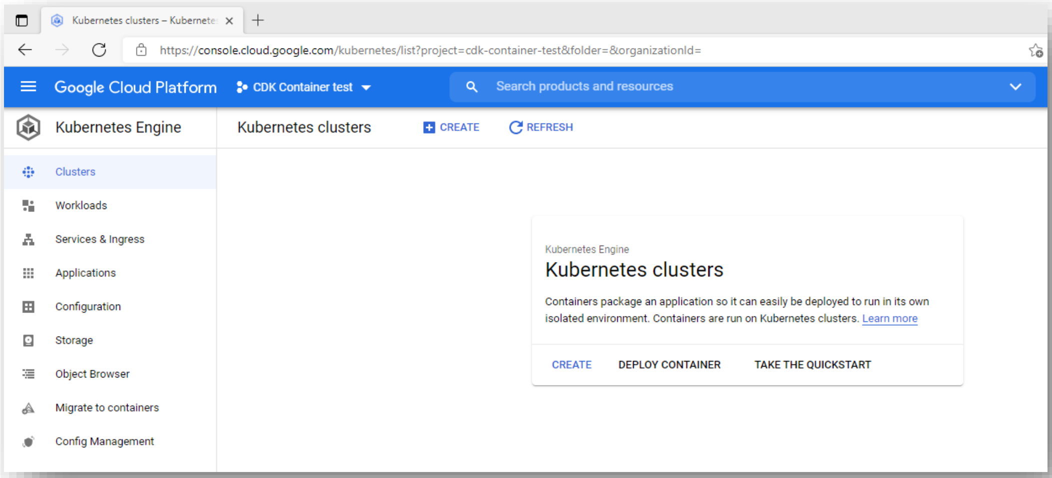
Filter Enter property name or value

Name	Hostname	Visibility
cdk-scheduler-linux	gcr.io	Private
cdk-web-linux	gcr.io	Private

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

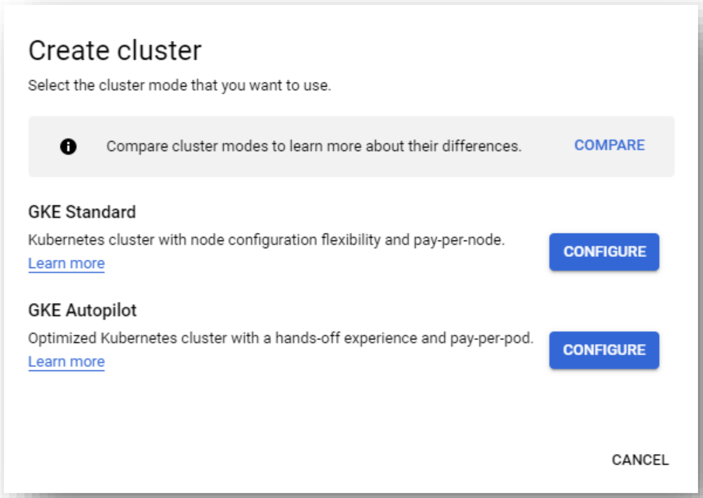


Kubernetes Engine

Kubernetes clusters

Containers package an application so it can easily be deployed to run in its own isolated environment. Containers are run on Kubernetes clusters. Learn more

Choose a GKE Standard Cluster:



Create cluster

Select the cluster mode that you want to use.

Compare cluster modes to learn more about their differences. COMPARE

GKE Standard

Kubernetes cluster with node configuration flexibility and pay-per-node.
Learn more

CONFIGURE

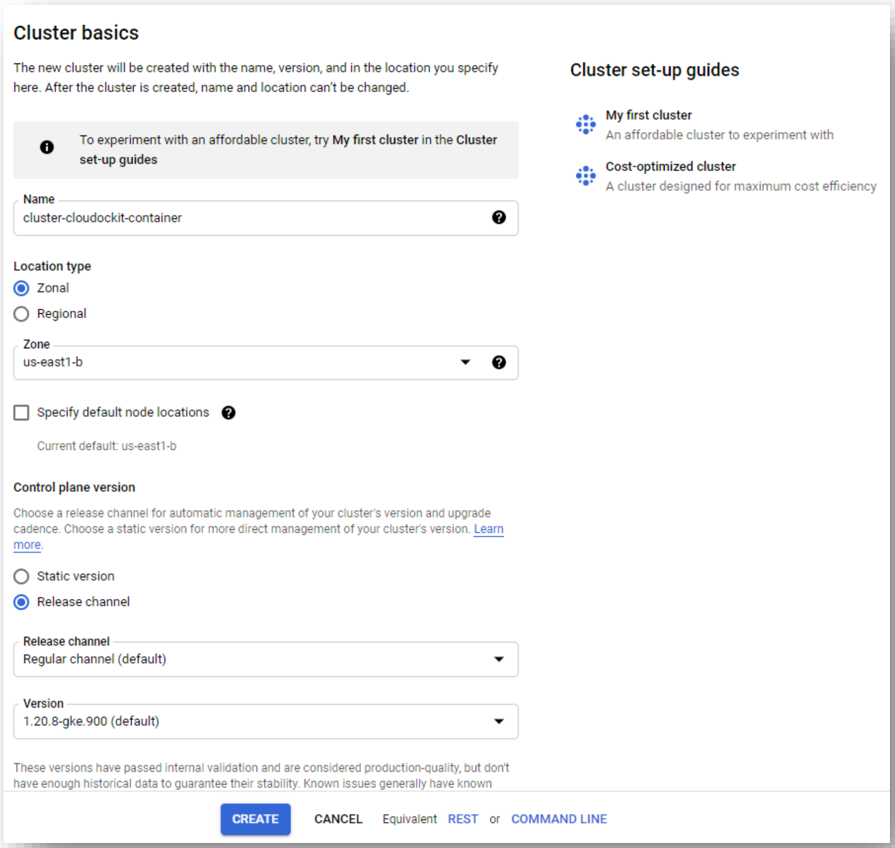
GKE Autopilot

Optimized Kubernetes cluster with a hands-off experience and pay-per-pod.
Learn more

CONFIGURE

CANCEL

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



Cluster basics

The new cluster will be created with the name, version, and in the location you specify here. After the cluster is created, name and location can't be changed.

To experiment with an affordable cluster, try **My first cluster** in the **Cluster set-up guides**

Name
cluster-cloudockit-container

Location type
(●) Zonal
( ) Regional

Zone
us-east1-b

[ ] Specify default node locations

Current default: us-east1-b

Control plane version

Choose a release channel for automatic management of your cluster's version and upgrade cadence. Choose a static version for more direct management of your cluster's version. Learn more.

( ) Static version
(●) Release channel

Release channel
Regular channel (default)

Version
1.20.8-gke.900 (default)

These versions have passed internal validation and are considered production-quality, but don't have enough historical data to guarantee their stability. Known issues generally have known

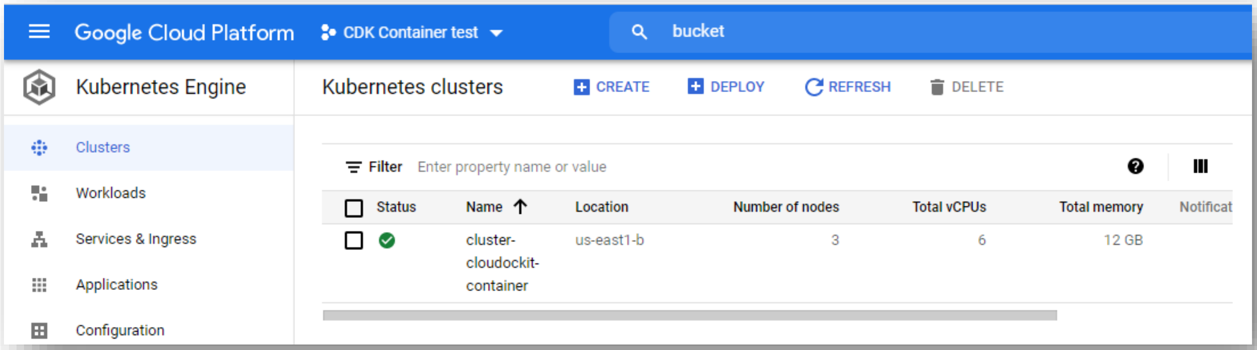
CREATE CANCEL Equivalent REST or COMMAND LINE

Cluster set-up guides

My first cluster
An affordable cluster to experiment with

Cost-optimized cluster
A cluster designed for maximum cost efficiency

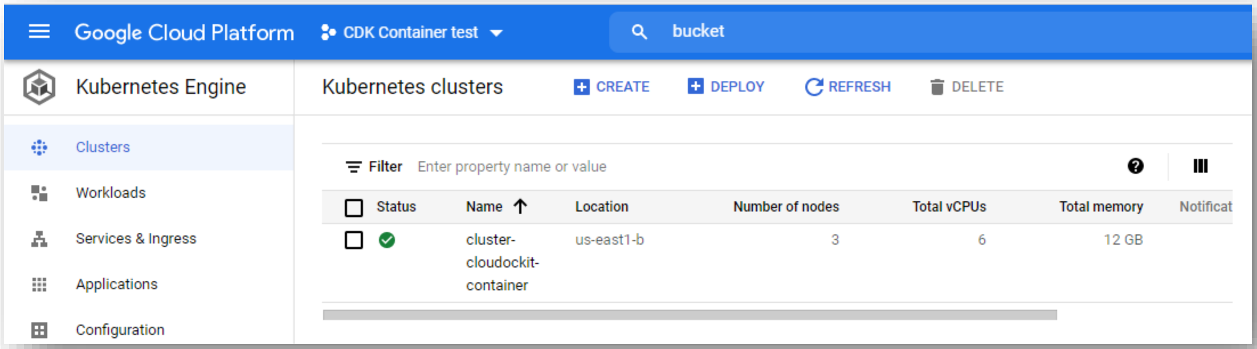
Then click on Create and wait for the cluster to be created:



Status	Name	Location	Number of nodes	Total vCPUs	Total memory	Notificat
✓	cluster-cloudockit-container	us-east1-b	3	6	12 GB	

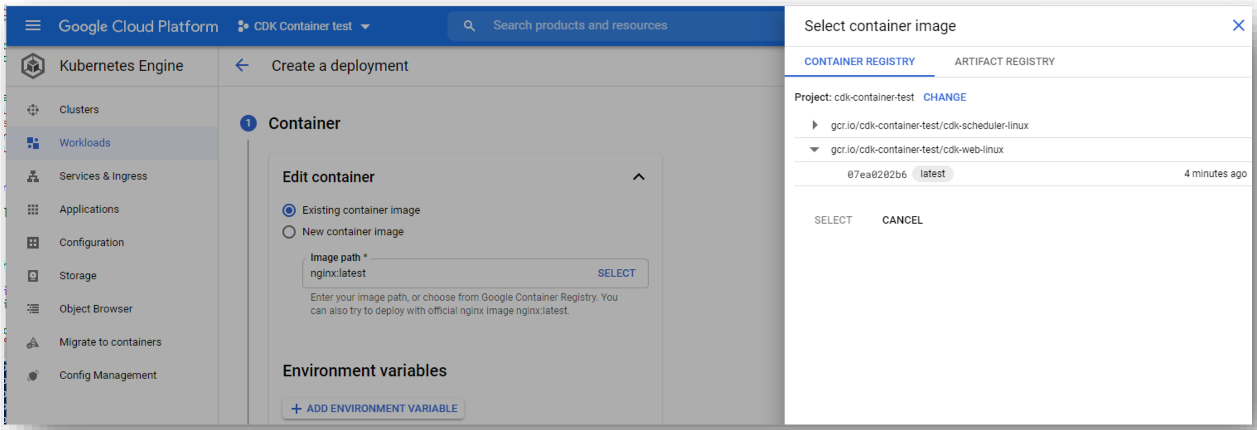
Deploy Cloudockit Container

Click on Deploy :



Status	Name	Location	Number of nodes	Total vCPUs	Total memory	Notificat
✓	cluster-cloudockit-container	us-east1-b	3	6	12 GB	

Select the cdk-web-linux container:



[✓ Existing container image] New container image

Image path *: nginx:latest

Enter your image path, or choose from Google Container Registry. You can also try to deploy with official nginx image nginx:latest.

Project: cdk-container-test

gcr.io/cdk-container-test/cdk-scheduler-linux

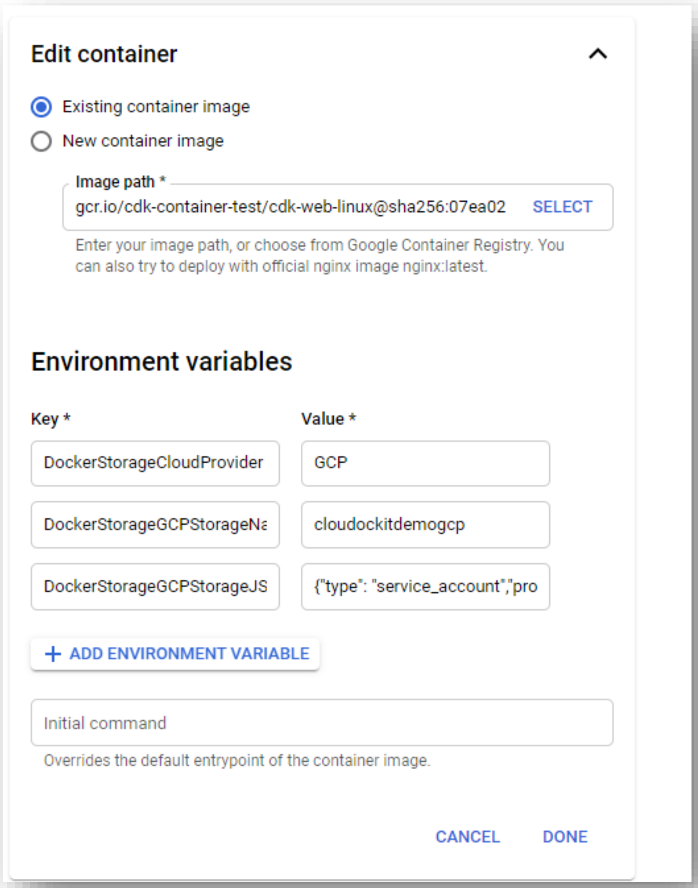
gcr.io/cdk-container-test/cdk-web-linux

07ea0202b6 latest 4 minutes ago

Enter the following environment variables and click Done

Name	Value
AppInsightKey (optional)	An Azure App Insight Instrumentation Key for advanced login
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

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



Edit container

[✓ Existing container image]
New container image

Image path *
gcr.io/cdk-container-test/cdk-web-linux@sha256:07ea02
SELECT

Enter your image path, or choose from Google Container Registry. You can also try to deploy with official nginx image nginx:latest.

Environment variables

Key *	Value *
DockerStorageCloudProvider	GCP
DockerStorageGCPStorageNa	cloudockitdemogcp
DockerStorageGCPStorageJS	{"type": "service_account","pro

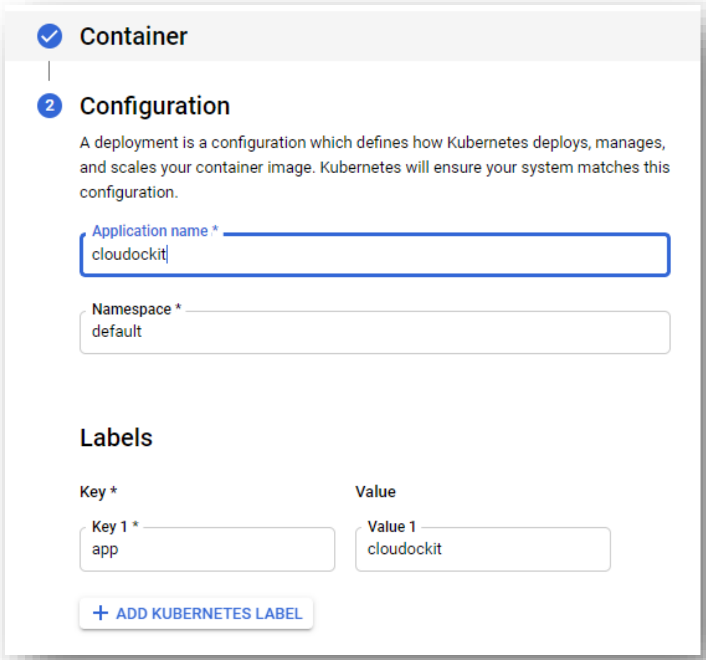
+ ADD ENVIRONMENT VARIABLE

Initial command

Overrides the default entrypoint of the container image.

CANCEL DONE

Then, in configuration, enter the following information


Container

Configuration

A deployment is a configuration which defines how Kubernetes deploys, manages, and scales your container image. Kubernetes will ensure your system matches this configuration.

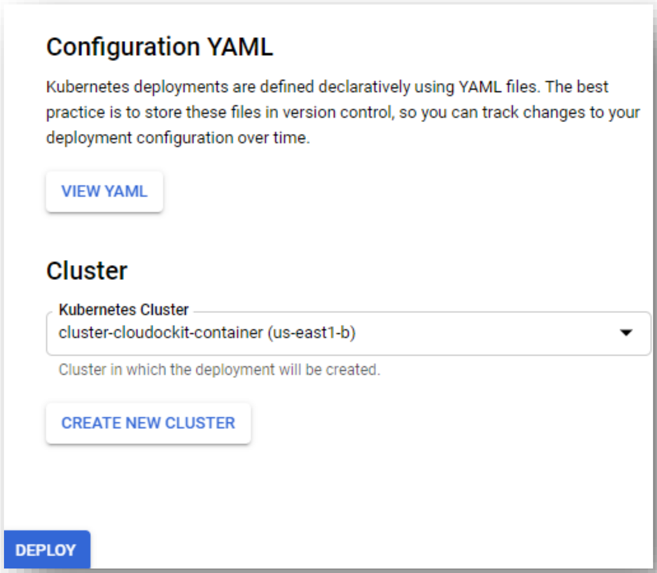
Application name *: cloudockit

Namespace *: default

Labels

Key *	Value
Key 1 *: app	Value 1: cloudockit

Then, click Deploy:


Configuration YAML

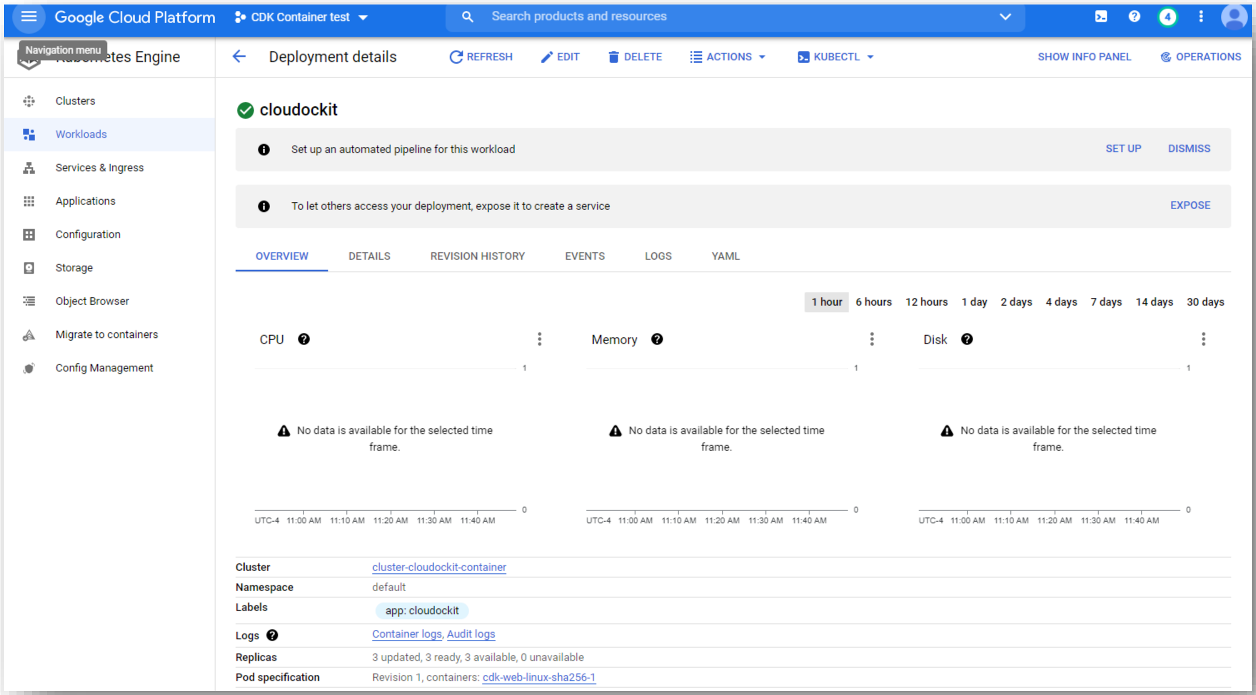
Kubernetes deployments are defined declaratively using YAML files. The best practice is to store these files in version control, so you can track changes to your deployment configuration over time.

Cluster

Kubernetes Cluster: cluster-cloudockit-container (us-east1-b)

Cluster in which the deployment will be created.

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



cloudockit

Set up an automated pipeline for this workload

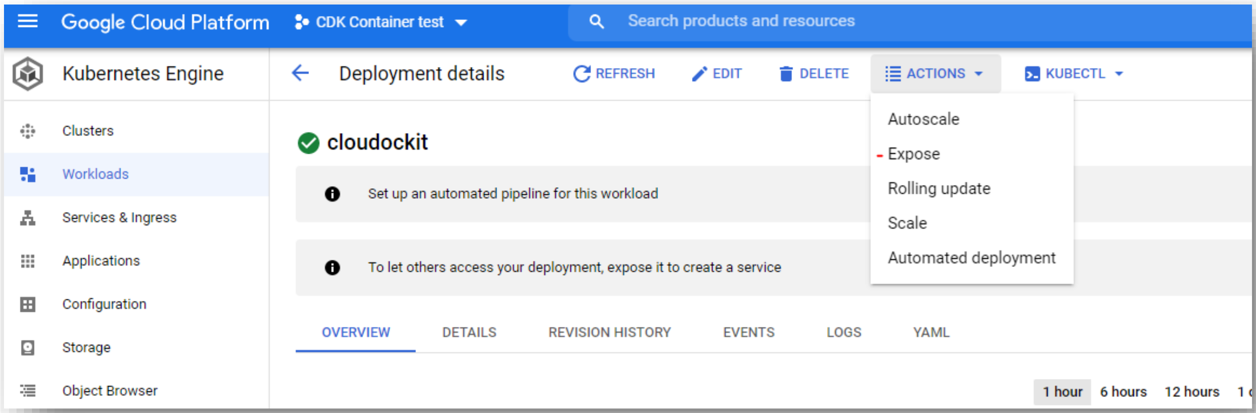
To let others access your deployment, expose it to create a service

No data is available for the selected time frame.

Cluster	cluster-cloudockit-container
Namespace	default
Labels	app: cloudockit
Logs	Container logs, Audit logs
Replicas	3 updated, 3 ready, 3 available, 0 unavailable
Pod specification	Revision 1, containers: cdk-web-linux-sha256-1

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:

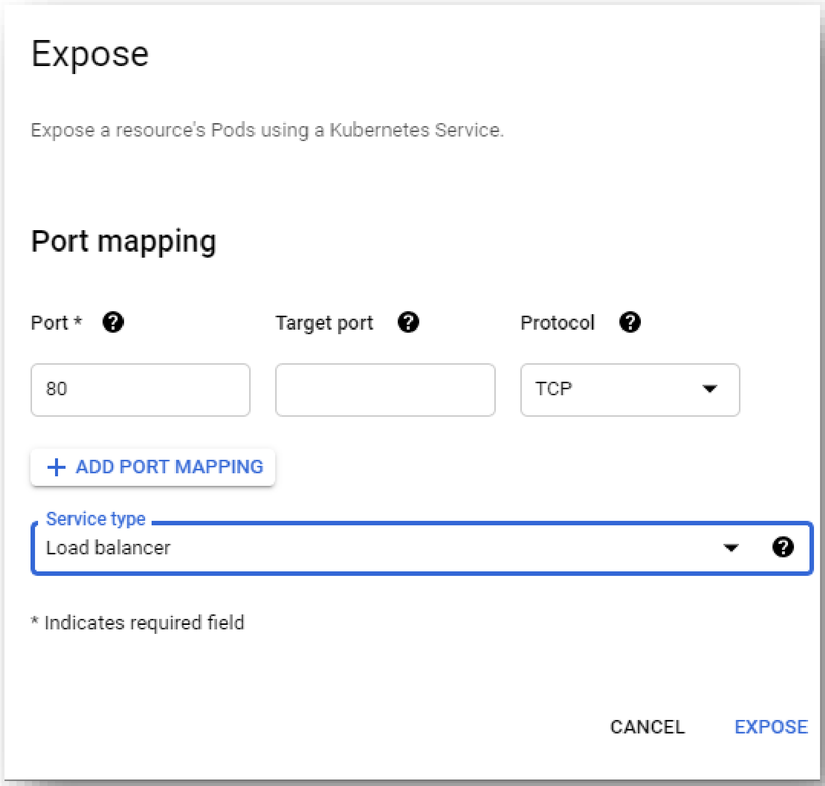


cloudockit

Set up an automated pipeline for this workload

To let others access your deployment, expose it to create a service

Then, click Expose:



Expose

Expose a resource's Pods using a Kubernetes Service.

Port mapping

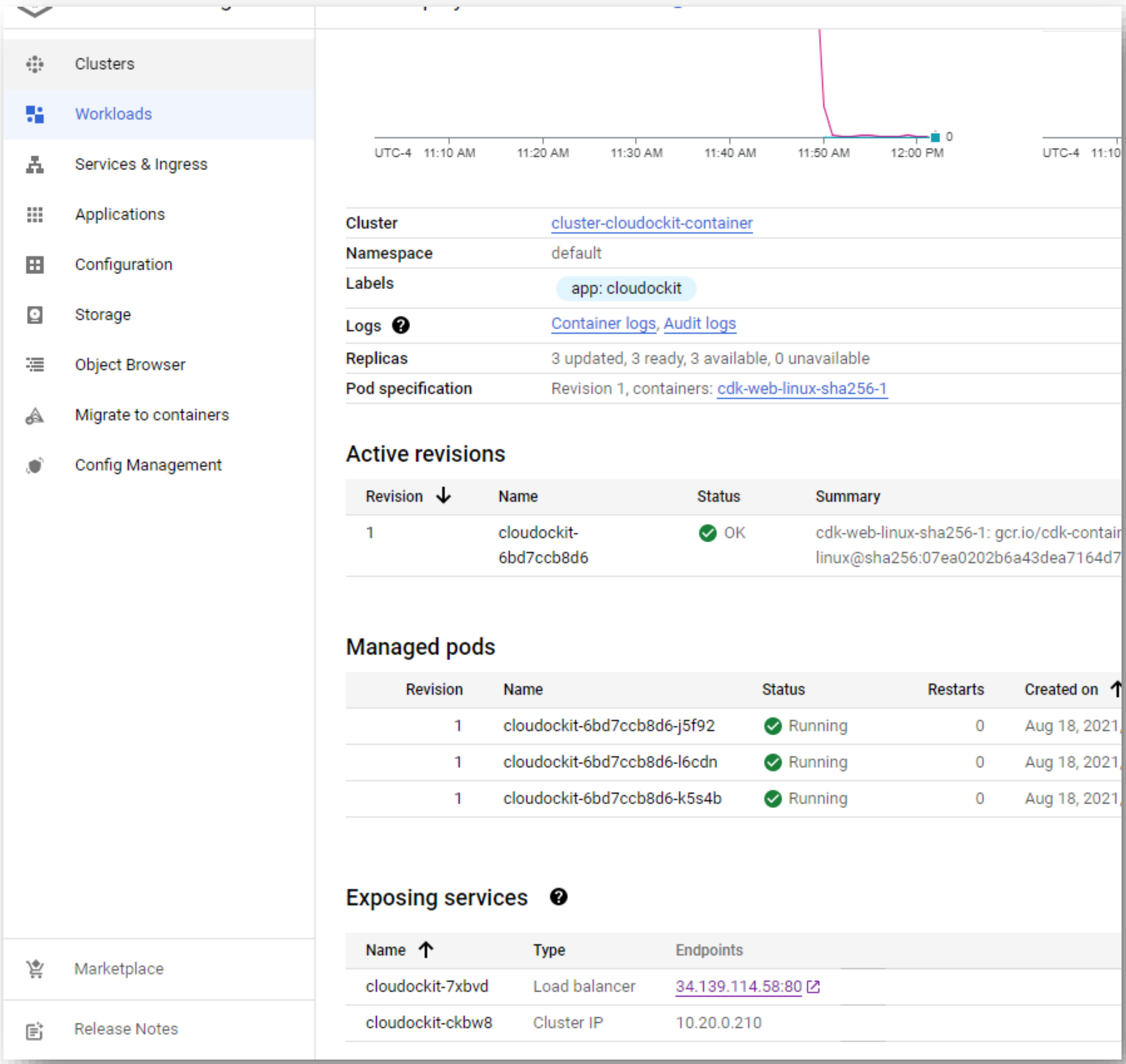
Port *	Target port	Protocol
80		TCP

Service type: Load balancer

* Indicates required field

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:



Cluster	cluster-cloudockit-container
Namespace	default
Labels	app: cloudockit
Logs	Container logs, Audit logs
Replicas	3 updated, 3 ready, 3 available, 0 unavailable
Pod specification	Revision 1, containers: cdk-web-linux-sha256-1

Active revisions

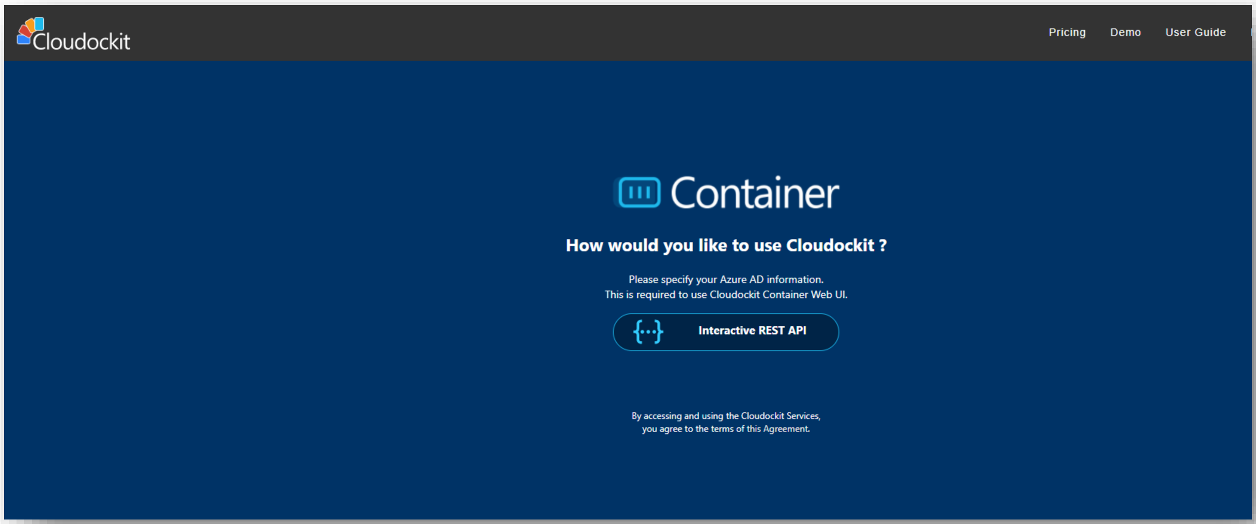
Revision	Name	Status	Summary
1	cloudockit-6bd7ccb8d6	OK	cdk-web-linux-sha256-1: gcr.io/cdk-contai linux@sha256:07ea0202b6a43dea7164d7

Managed pods

Revision	Name	Status	Restarts	Created on
1	cloudockit-6bd7ccb8d6-j5f92	Running	0	Aug 18, 2021
1	cloudockit-6bd7ccb8d6-l6cdn	Running	0	Aug 18, 2021
1	cloudockit-6bd7ccb8d6-k5s4b	Running	0	Aug 18, 2021

Exposing services

Name	Type	Endpoints
cloudockit-7xbvd	Load balancer	34.139.114.58:80
cloudockit-ckbw8	Cluster IP	10.20.0.210



How would you like to use Cloudockit ?

Please specify your Azure AD information.
This is required to use Cloudockit Container Web UI.

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.

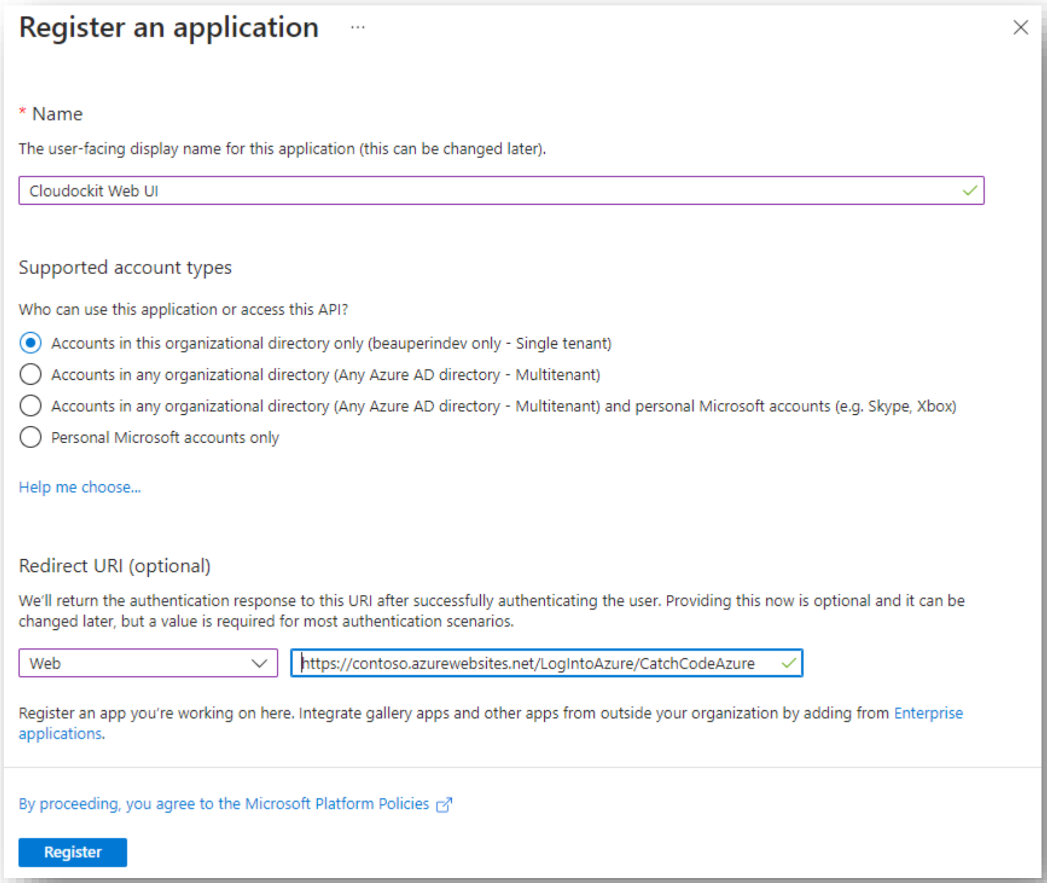
This Web UI supports Azure Active Directory as a first step to authenticate users.

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



* Name

The user-facing display name for this application (this can be changed later).

Cloudockit Web UI

Supported account types

Who can use this application or access this API?

- ● Accounts in this organizational directory only (beauperindev only - Single tenant)
- ○ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ○ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ○ Personal Microsoft accounts only

Help me choose...

Redirect URI (optional)

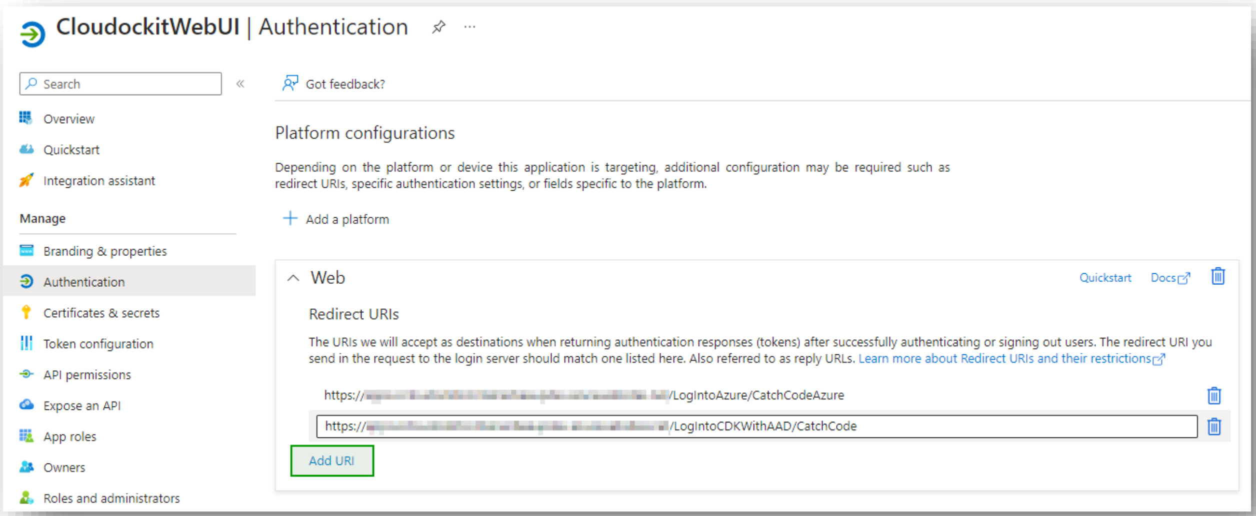
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web | https://contoso.azurewebsites.net/LogIntoAzure/CatchCodeAzure

Register an app you're working on here. Integrate gallery apps and other apps from outside your organization by adding from Enterprise applications.

By proceeding, you agree to the Microsoft Platform Policies

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:


CloudockitWebUI | Authentication

Platform configurations

Depending on the platform or device this application is targeting, additional configuration may be required such as redirect URIs, specific authentication settings, or fields specific to the platform.

Redirect URIs

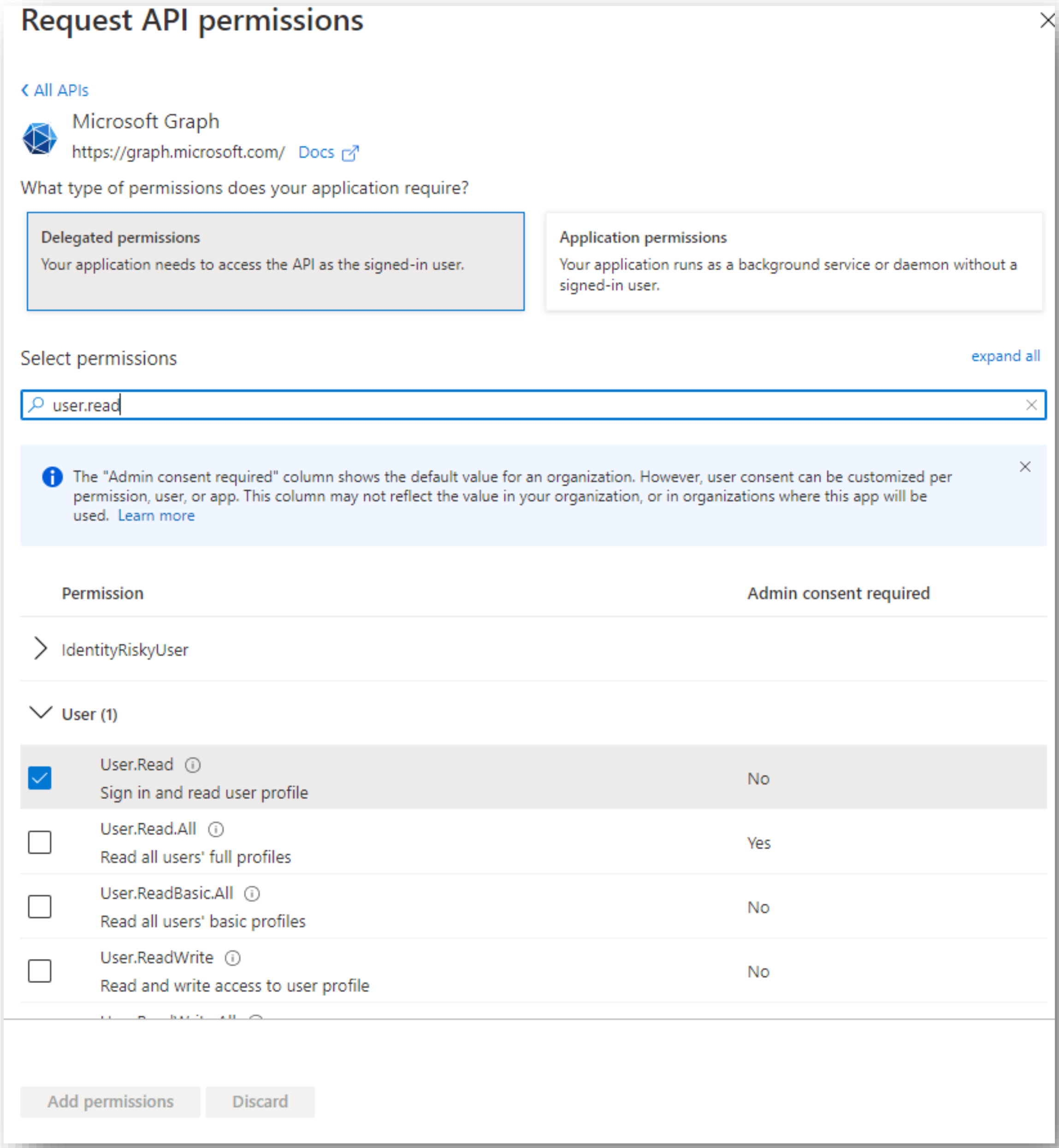
The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating or signing out users. The redirect URI you send in the request to the login server should match one listed here. Also referred to as reply URLs. Learn more about Redirect URIs and their restrictions

https://[illegible]/LogIntoAzure/CatchCodeAzure

https://[illegible]/LogIntoCDKWithAAD/CatchCode

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***:



Request API permissions

< All APIs

Microsoft Graph

https://graph.microsoft.com/ Docs

What type of permissions does your application require?

Delegated permissions
Your application needs to access the API as the signed-in user.

Application permissions
Your application runs as a background service or daemon without a signed-in user.

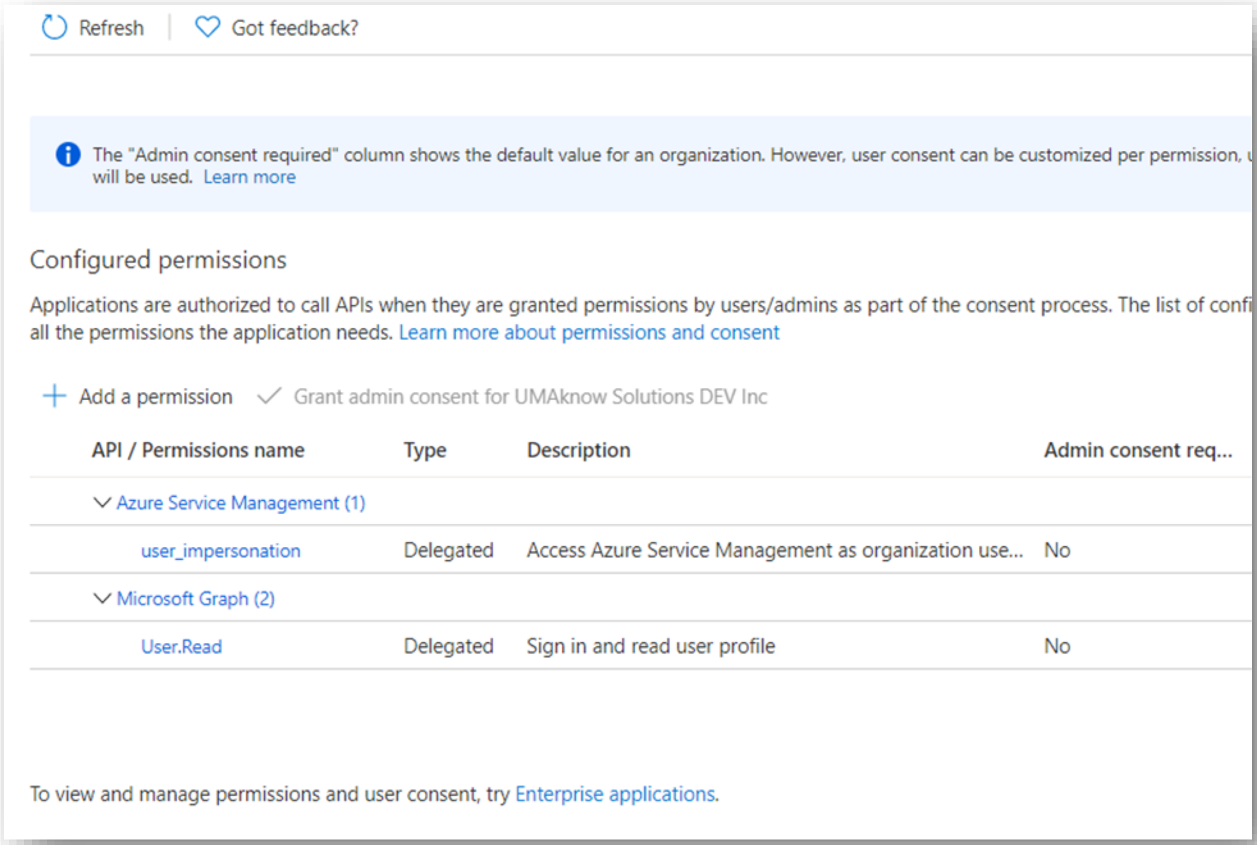
Select permissions — expand all

user.read

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. Learn more

	Permission	Admin consent required
	> IdentityRiskyUser	
	∨ User (1)	
✓	User.Read Sign in and read user profile	No
☐	User.Read.All Read all users' full profiles	Yes
☐	User.ReadBasic.All Read all users' basic profiles	No
☐	User.ReadWrite Read and write access to user profile	No

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



The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, u
will be used. Learn more

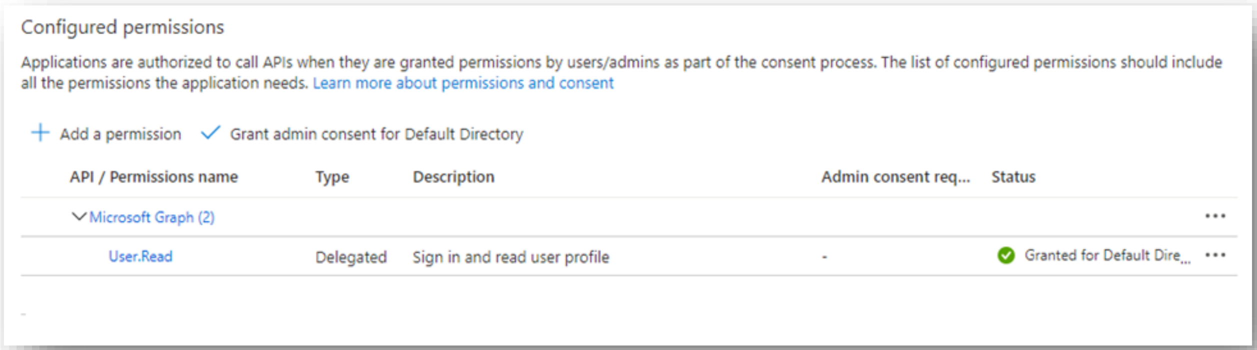
Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of confi
all the permissions the application needs. Learn more about permissions and consent

API / Permissions name	Type	Description	Admin consent req...
Azure Service Management (1)			
user_impersonation	Delegated	Access Azure Service Management as organization use...	No
Microsoft Graph (2)			
User.Read	Delegated	Sign in and read user profile	No

To view and manage permissions and user consent, try Enterprise applications.

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):

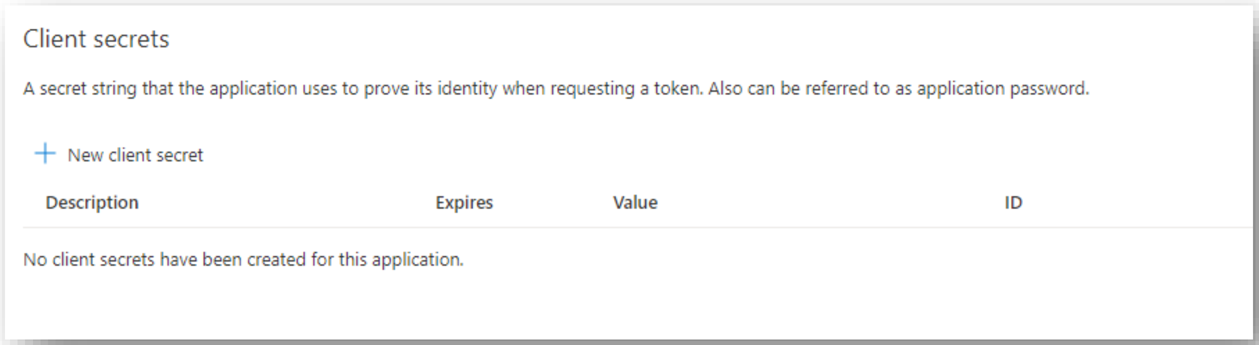


Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. Learn more about permissions and consent

API / Permissions name	Type	Description	Admin consent req...	Status
Microsoft Graph (2)				
User.Read	Delegated	Sign in and read user profile	-	✓ Granted for Default Dire...

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.


Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

Description	Expires	Value	ID

No client secrets have been created for this application.

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.

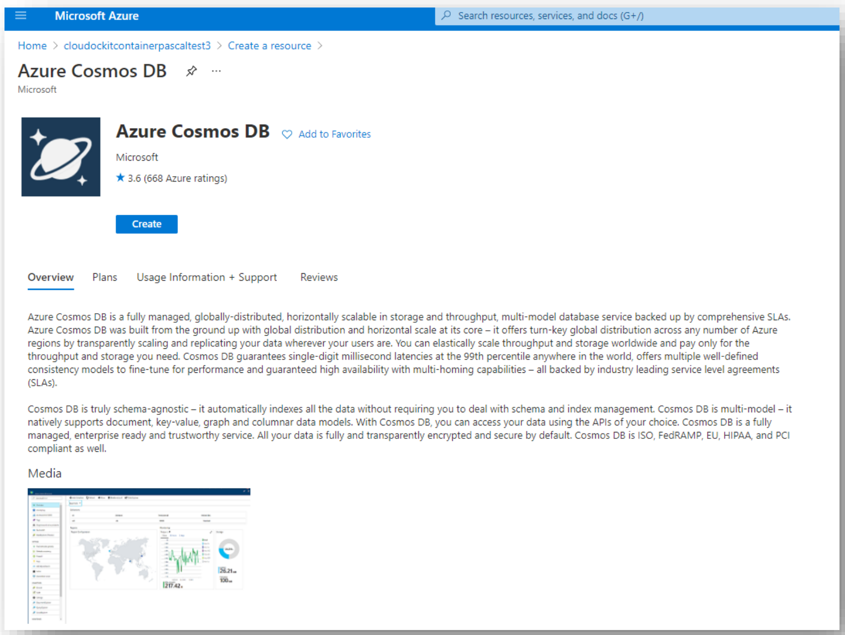
There are two steps required:

- 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)

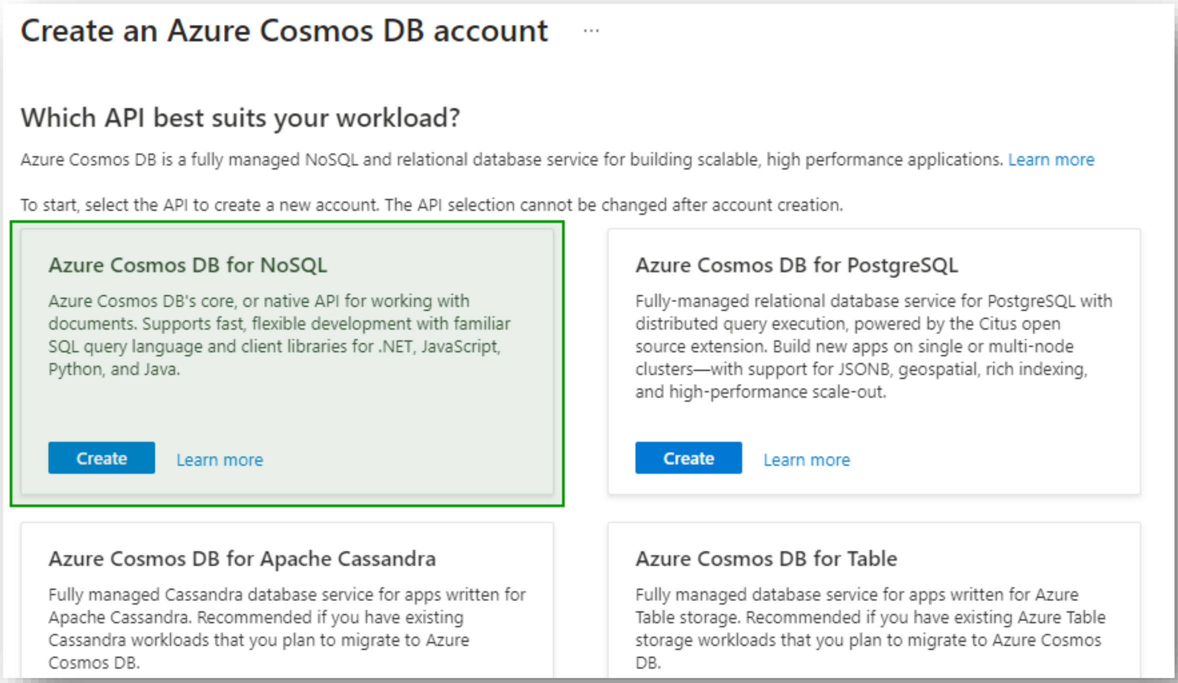
- Create a Cosmos DB


Microsoft Azure
Search resources, services, and docs (G+/)
Home > cloudockitcontainerpascaltest3 > Create a resource >
Azure Cosmos DB
Microsoft
Azure Cosmos DB
Add to Favorites
Microsoft
★ 3.6 (668 Azure ratings)
Create
Overview
Plans
Usage Information + Support
Reviews
Media

Azure Cosmos DB is a fully managed, globally-distributed, horizontally scalable in storage and throughput, multi-model database service backed up by comprehensive SLAs. Azure Cosmos DB was built from the ground up with global distribution and horizontal scale at its core – it offers turn-key global distribution across any number of Azure regions by transparently scaling and replicating your data wherever your users are. You can elastically scale throughput and storage worldwide and pay only for the throughput and storage you need. Cosmos DB guarantees single-digit millisecond latencies at the 99th percentile anywhere in the world, offers multiple well-defined consistency models to fine-tune for performance and guaranteed high availability with multi-homing capabilities – all backed by industry leading service level agreements (SLAs).

Cosmos DB is truly schema-agnostic – it automatically indexes all the data without requiring you to deal with schema and index management. Cosmos DB is multi-model – it natively supports document, key-value, graph and columnar data models. With Cosmos DB, you can access your data using the APIs of your choice. Cosmos DB is a fully managed, enterprise ready and trustworthy service. All your data is fully and transparently encrypted and secure by default. Cosmos DB is ISO, FedRAMP, EU, HIPAA, and PCI compliant as well.

- Choose Azure Cosmos DB for NoSQL for the type



Create an Azure Cosmos DB account ...

Which API best suits your workload?

Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high performance applications. Learn more

To start, select the API to create a new account. The API selection cannot be changed after account creation.

Azure Cosmos DB for NoSQL

Azure Cosmos DB's core, or native API for working with documents. Supports fast, flexible development with familiar SQL query language and client libraries for .NET, JavaScript, Python, and Java.

Create | Learn more

Azure Cosmos DB for PostgreSQL

Fully-managed relational database service for PostgreSQL with distributed query execution, powered by the Citus open source extension. Build new apps on single or multi-node clusters—with support for JSONB, geospatial, rich indexing, and high-performance scale-out.

Create | Learn more

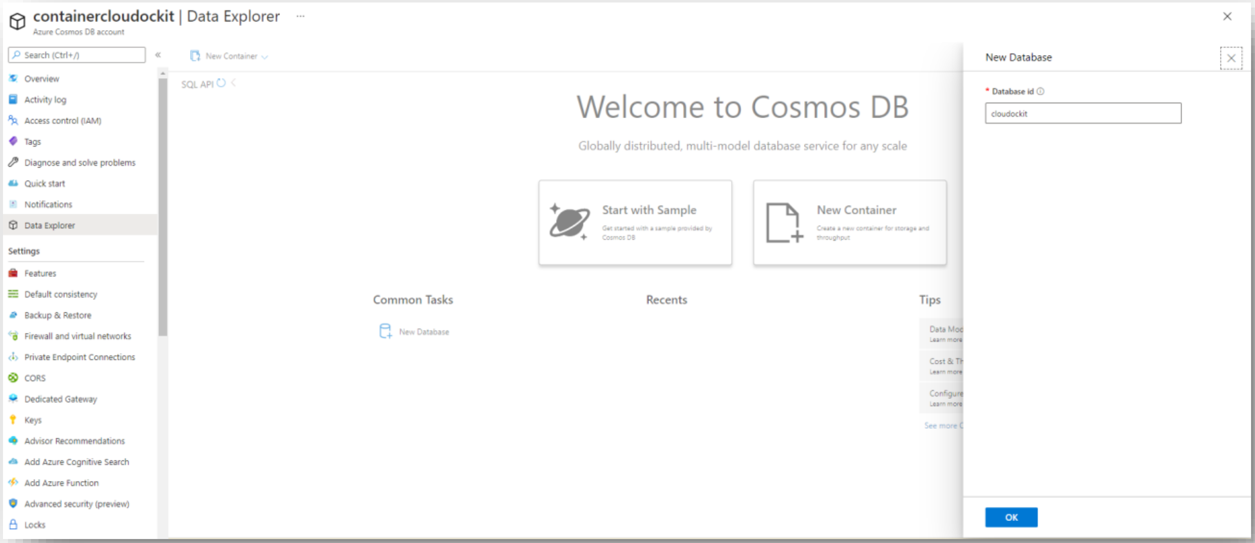
Azure Cosmos DB for Apache Cassandra

Fully managed Cassandra database service for apps written for Apache Cassandra. Recommended if you have existing Cassandra workloads that you plan to migrate to Azure Cosmos DB.

Azure Cosmos DB for Table

Fully managed database service for apps written for Azure Table storage. Recommended if you have existing Azure Table storage workloads that you plan to migrate to Azure Cosmos DB.

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



containercloudockit | Data Explorer

Azure Cosmos DB account

New Database

* Database id: cloudockit

OK

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:

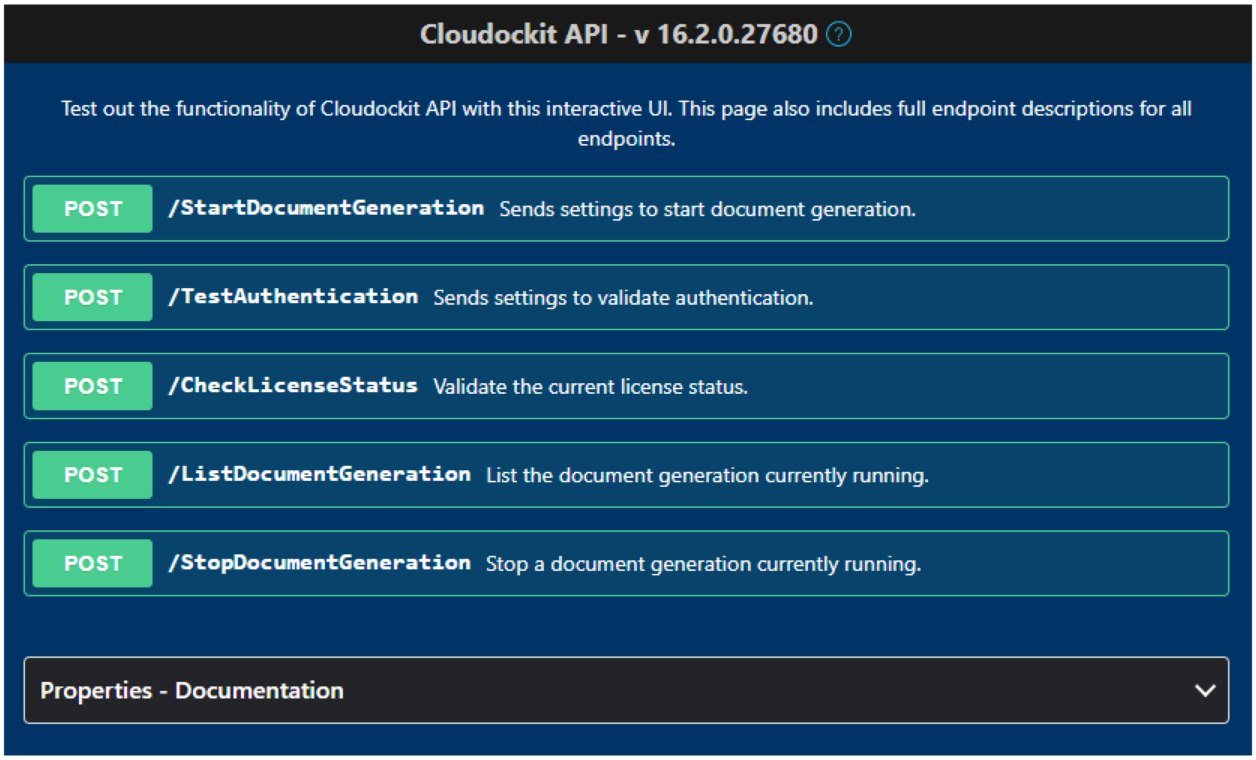
Name	Value
CosmosDb__DatabaseName	Enter the name of the Database that you have created in the previous step (cloudockit in the example)
ConnectionStrings__CosmosDb	Azure CosmosDB Connection string

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.



Cloudockit API - v 16.2.0.27680

Test out the functionality of Cloudockit API with this interactive UI. This page also includes full endpoint descriptions for all endpoints.

POST **/StartDocumentGeneration** Sends settings to start document generation.

POST **/TestAuthentication** Sends settings to validate authentication.

POST **/CheckLicenseStatus** Validate the current license status.

POST **/ListDocumentGeneration** List the document generation currently running.

POST **/StopDocumentGeneration** Stop a document generation currently running.

Properties - Documentation

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

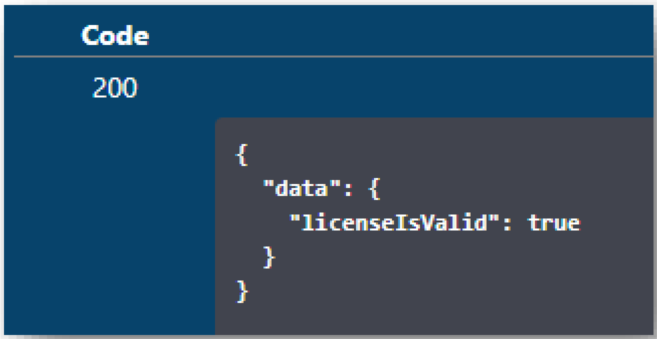
Once you get the API Key from Cloudockit team and you have the appropriate credentials for the license validation, you can check that your API Key is working by using the **/CheckLicenseStatus** endpoint.

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:



```
200
{
  "data": {
    "licenseIsValid": true
  }
}
```

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
(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

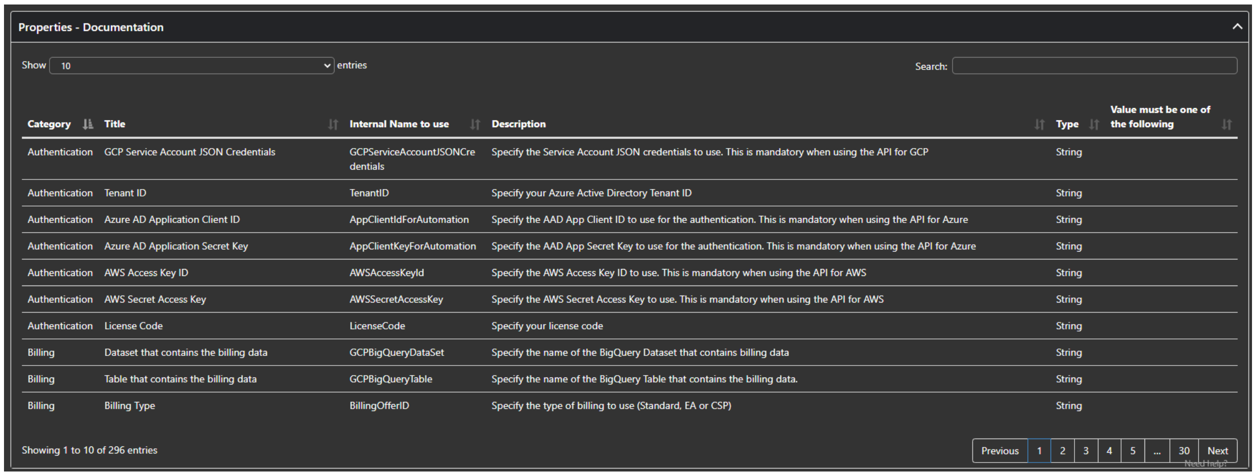
Once all the tests above have been done, you can start the document generation.

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:

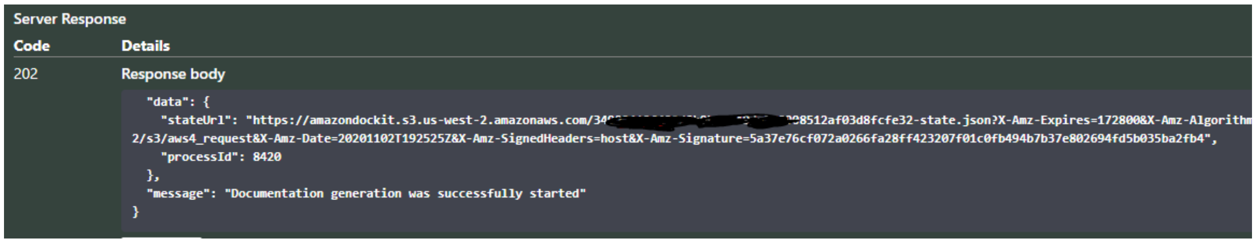
Category	Title	Internal Name to use	Description	Type	Value must be one of the following
Authentication	GCP Service Account JSON Credentials	GCPServiceAccountJSONCredentials	Specify the Service Account JSON credentials to use. This is mandatory when using the API for GCP	String	
Authentication	Tenant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

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	Details
202	Response body

```
  "data": {
    "stateUrl": "https://amazondockit.s3.us-west-2.amazonaws.com/34[redacted]28512af03d8fcfe32-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.

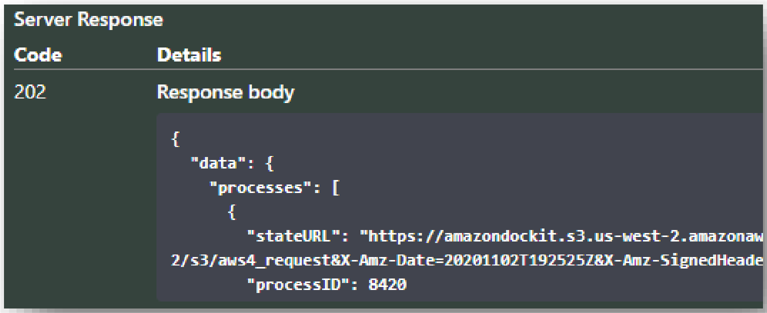
Step J – Manage your document generation (Preview)

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:



Server Response

Code	Details
202	Response body

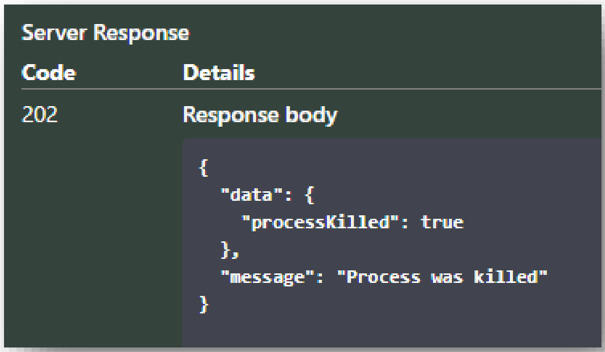
```
{
  "data": {
    "processes": [
      {
        "stateURL": "https://amazondockit.s3.us-west-2.amazonaw
2/s3/aws4_request&X-Amz-Date=20201102T192525Z&X-Amz-SignedHeade
        "processID": 8420
```

/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:



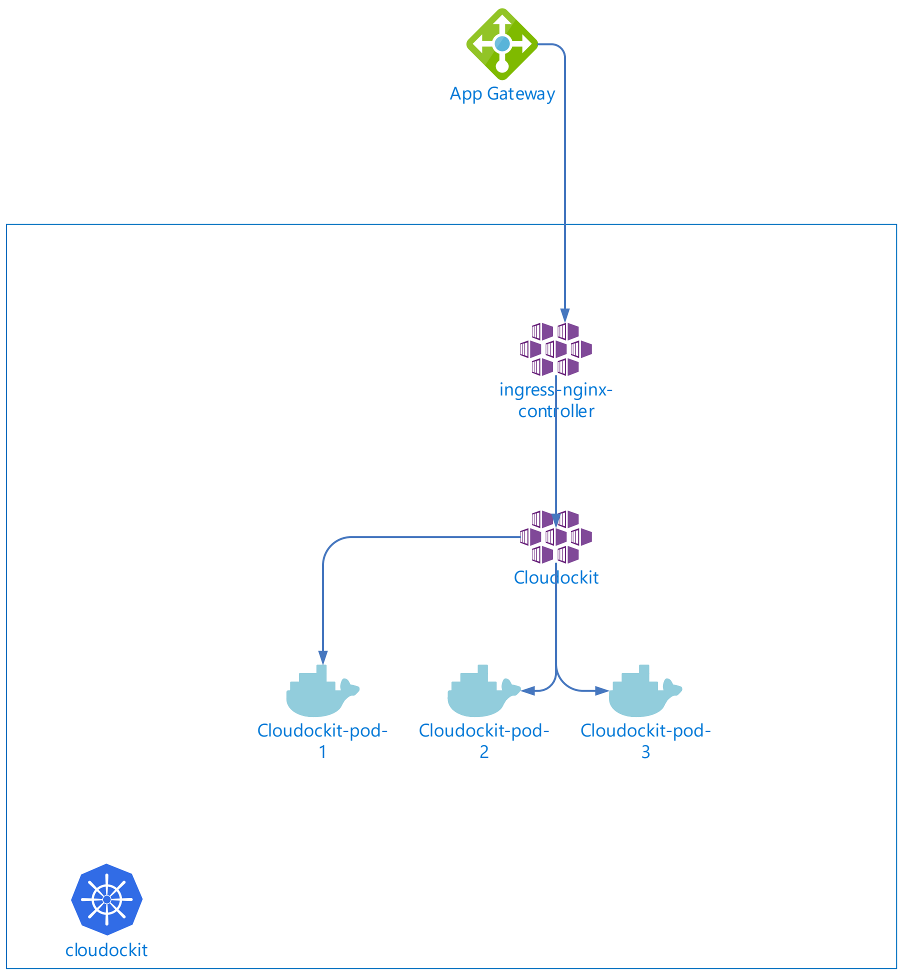
Server Response

Code	Details
202	Response body

```
{
  "data": {
    "processKilled": true
  },
  "message": "Process was 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	mycert.pfx

	used to encrypt the key	
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

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 `AppInsightKey` and that contains an Azure App Insight Instrumentation key 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.