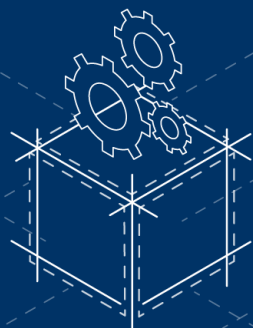




Container

GCP - INSTALLATION & CONFIGURATION GUIDE



Contents

Introduction.....	3
Requirements.....	5
Step A – Create a Bucket and Get a License	6
Create a Bucket	6
Licensing	7
Create a service account to access the bucket	9
Step B – Create your container environment and Start your container.....	11
Create a Google Container Registry and Upload image	11
Create a Kubernetes Cluster.....	12
Deploy Clouddokit Container.....	14
Step C (Optional) – Configure Clouddokit Web UI.....	22
Step D (Optional) – Configure Clouddokit Container to support Scheduling.	27
Start Clouddokit Scheduler Container.....	27
Set Settings in the settings file	27
Step E (Optional) – Configure Clouddokit Container to support the creation of Compliance Rules, Tailored Diagrams and Settings	28
Create (or re-use) an Azure Cosmos DB.....	28
Configure Clouddokit Container to use the Azure Comos DB.....	30
Step F – Understand Clouddokit API Container	31
Step G – Test your license	32
Activate and setup components for your license.....	32
Step H – Validate that you can authenticate to the environment that you want to scan.....	33
Step I – Test the document generation	35
Step J – Manage your document generation (Preview).....	36
/ListDocumentGeneration.....	36
/StopDocumentGeneration	36
Annex – Deploy multiple instances of Clouddokit Container	37
Annex – Troubleshooting	39

Introduction

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

There are two types of images that you should run:

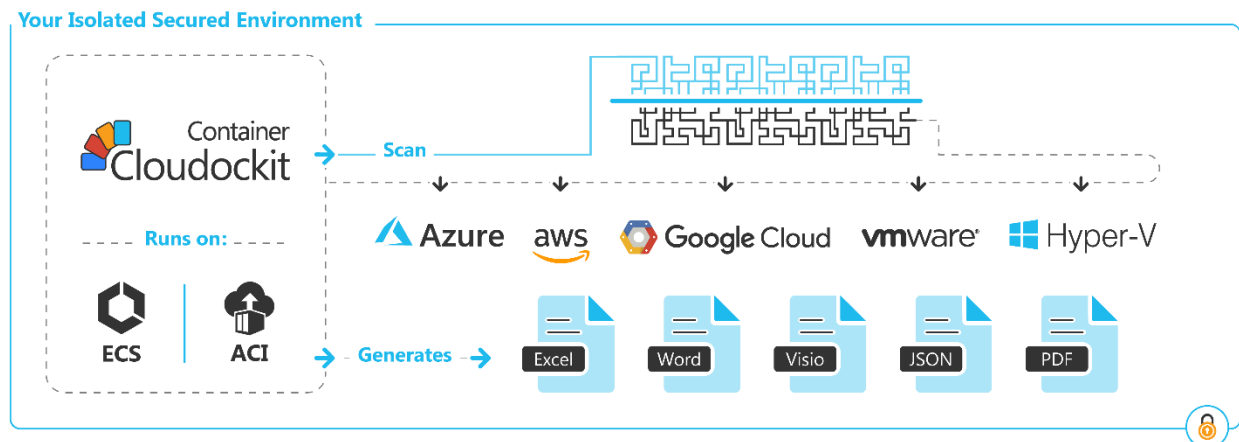
- **cdk-web-linux** that contain the Clouddokit API/Web interface. This is mandatory to run this container.
- **cdk-scheduler-linux** that contain the Clouddokit Scheduling features. This is an optional container you do not need to install if you do not want to use schedules.

The cdk-web image contains the Clouddokit 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 Clouddokit Web UI in the image so that you can get all the features that you are accustomed to.

Clouddokit Docker container images provide you a way to run Clouddokit into your own isolated Cloud environment and gives you the exact same features as Clouddokit Website and Clouddokit 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 Clouddokit 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 Clouddokit 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
- Clouddokit 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 Clouddokit Docker container image on the GCP Platform.

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

Step 1 - Create a Bucket and Get A License

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

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

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

Step 3 (Optional) - Activate Clouddokit 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

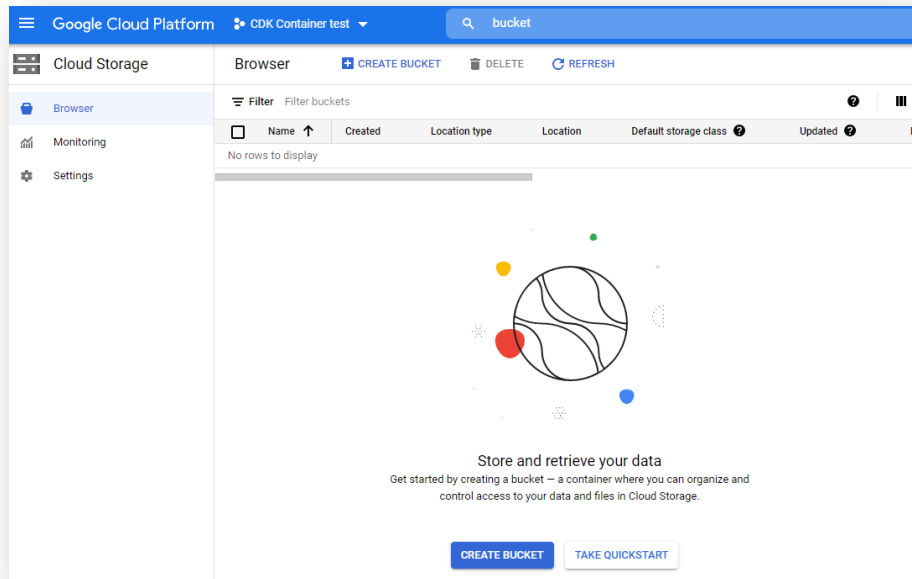
To install Clouddokit Container in your environment, you will need:

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

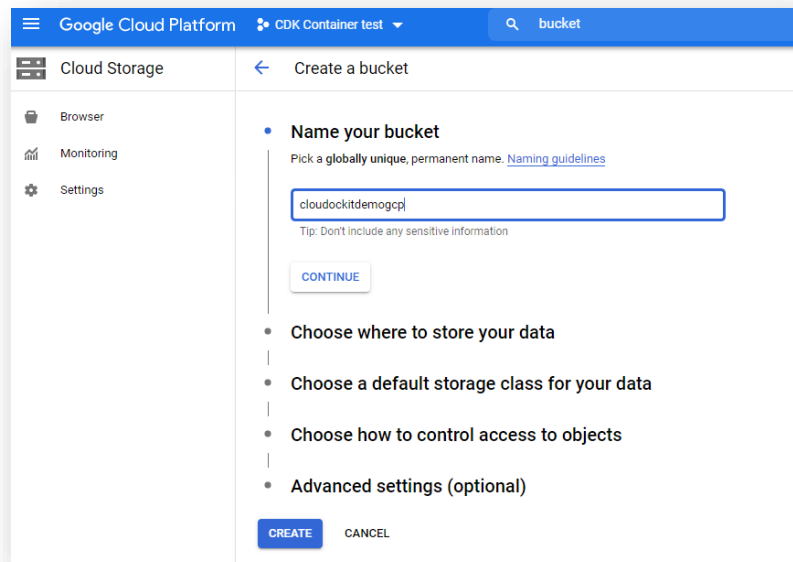
Step A – Create a Bucket and Get a License

Create a Bucket

The first step to deploy Clouddokit 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 Clouddokit 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

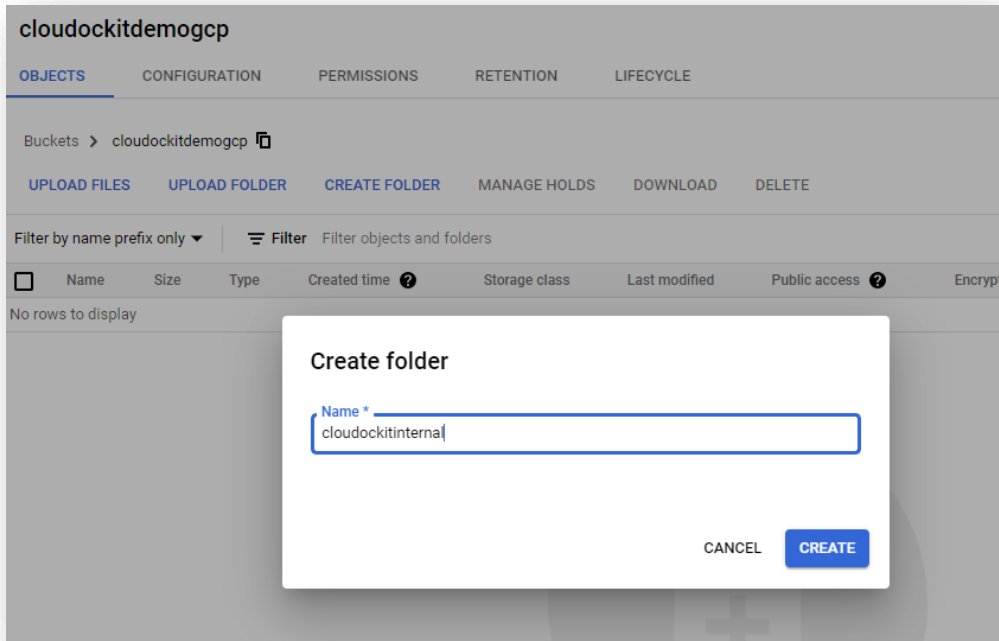
Licensing

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

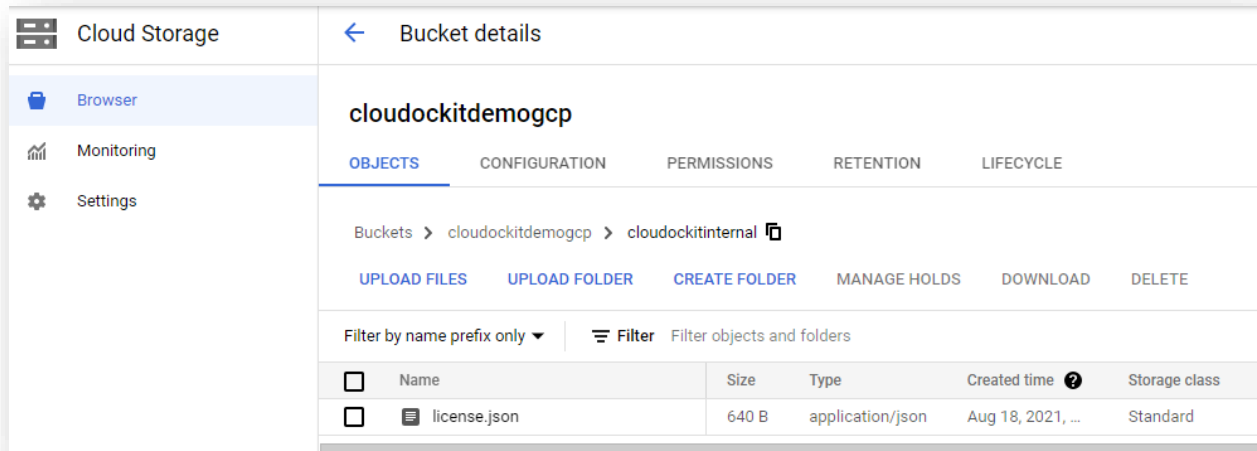
When you receive the license file, you will also get the following information:

- 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:

Create service account

1 Service account details

Service account name
clouddockit-container-demo

Display name for this service account

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

Service account description
Service Account used by Clouddockit 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:

← clouddockit-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 "clouddockit-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	Resource
cloudockit-container-demo@cdk-container-test.iam.gserviceaccount.com	cloudockitdemogcp

Role
Storage Admin ▼
Full control of GCS resources.

Condition
[Add condition](#)

[+ ADD ANOTHER ROLE](#)

SAVE **CANCEL**

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 Clouddokit 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/clouddokitapi*

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

```
#This script will download Clouddokit 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 Clouddokit
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 Clouddokit Container registry. Please refer to the email you received
$sourceRepoPwd = '' #Password to Clouddokit 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/<PROJECT-ID>
$currentVersion = 'latest'
$type = 'linux'
$sourceRepoURL = 'clouddokitcontainerregistry.azurecr.io'

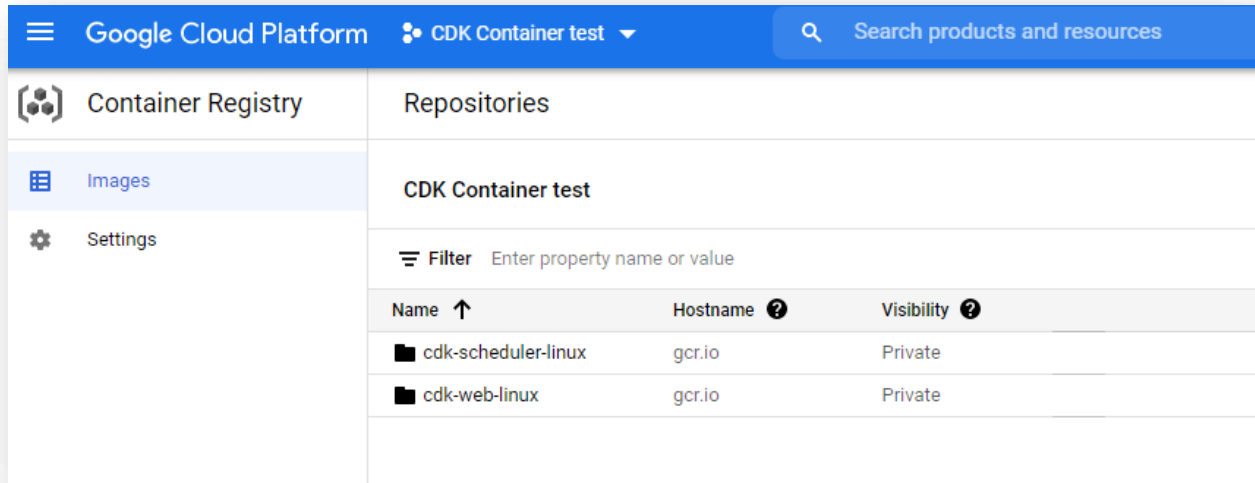
#connect to the Clouddokit Repository
docker login $sourceRepoURL -u $sourceRepoUser -p $sourceRepoPwd

#Pull the clouddokit 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
```

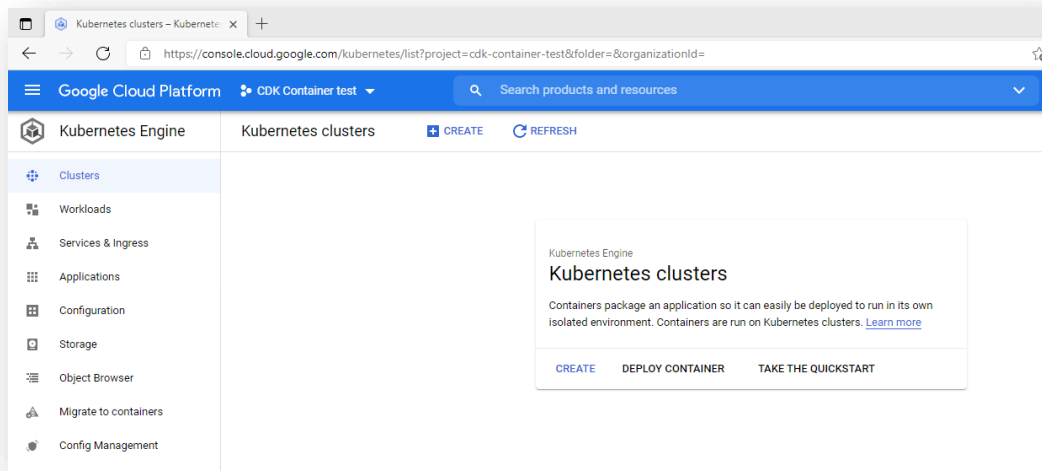
Then, validate that the images have been uploaded successfully:



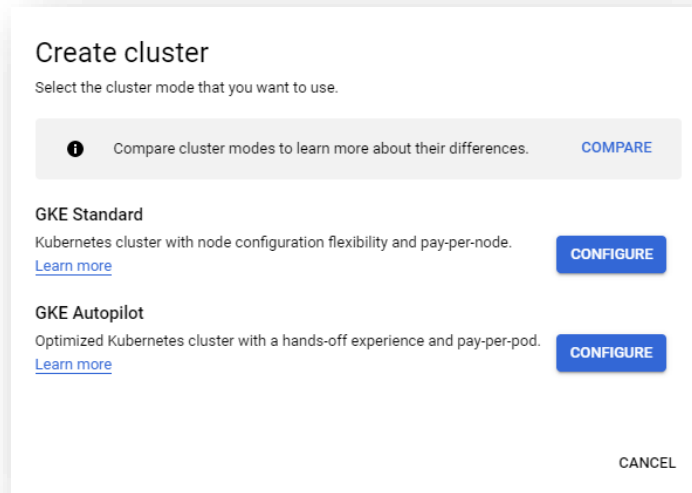
Create a Kubernetes Cluster

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](#).

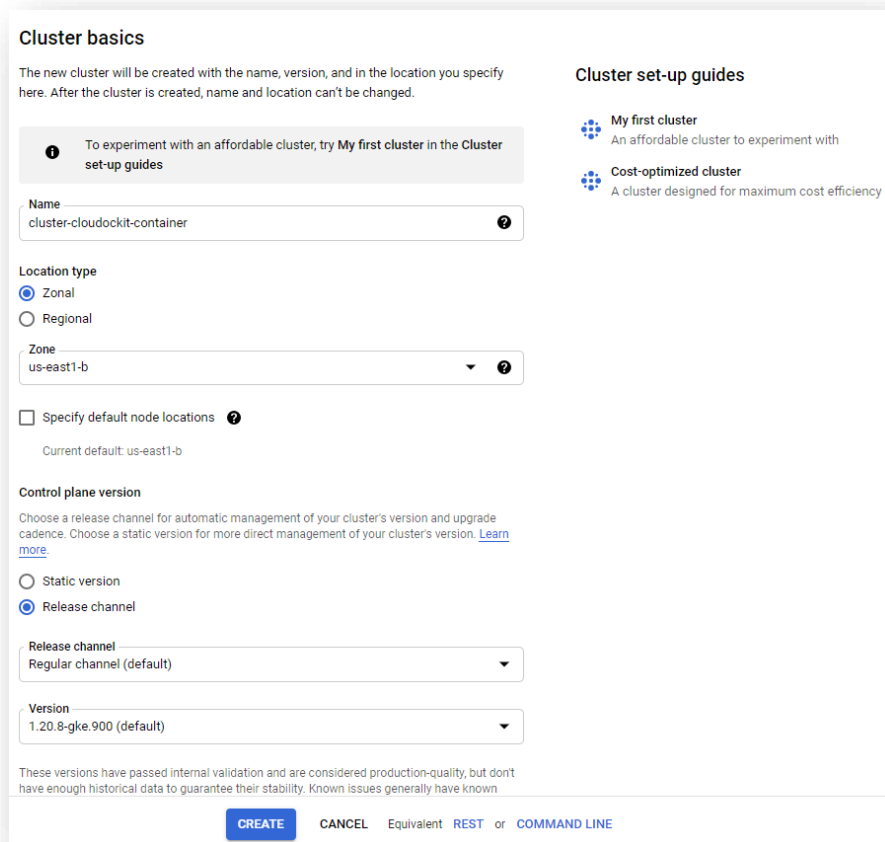
Click on Create



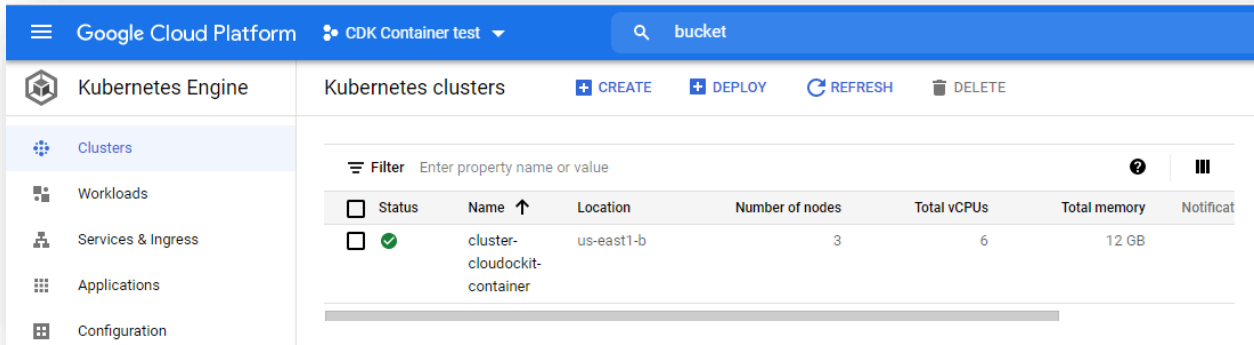
Choose a GKE Standard Cluster:



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

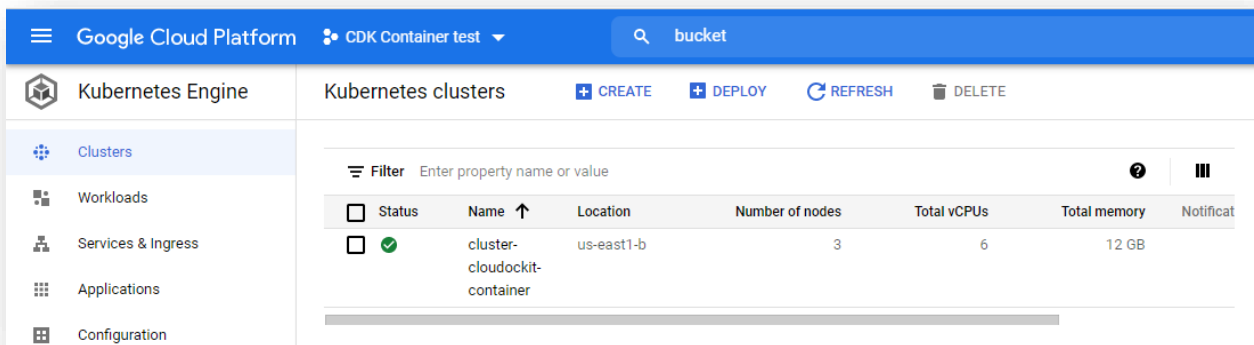


Then click on Create and wait for the cluster to be created:

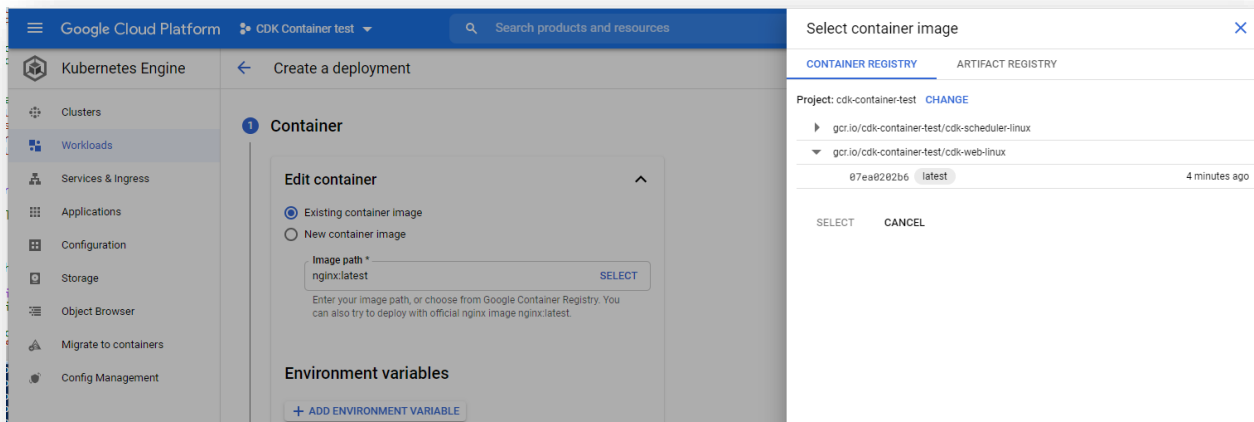


Deploy Clouddokit Container

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: <ul style="list-style-type: none"> • 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://cloudokit.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.

The screenshot shows a dialog box titled "Edit container" with a close button (upward arrow) in the top right corner. It has two radio button options: "Existing container image" (selected) and "New container image". Below these is a text field for "Image path *" containing "gcr.io/cdk-container-test/cdk-web-linux@sha256:07ea02" and a "SELECT" button. A note below the field says: "Enter your image path, or choose from Google Container Registry. You can also try to deploy with official nginx image nginx:latest." The "Environment variables" section has two columns: "Key *" and "Value *". It contains three rows of input fields: 1) Key: "DockerStorageCloudProvider", Value: "GCP"; 2) Key: "DockerStorageGCPStorageName", Value: "cloudockitdemogcp"; 3) Key: "DockerStorageGCPStorageJSONCredentials", Value: '{"type": "service_account", "pro". Below the table is a "+ ADD ENVIRONMENT VARIABLE" button. At the bottom is an "Initial command" text field with the note "Overrides the default entrypoint of the container image." and "CANCEL" and "DONE" buttons.

Key *	Value *
DockerStorageCloudProvider	GCP
DockerStorageGCPStorageName	cloudockitdemogcp
DockerStorageGCPStorageJSONCredentials	{"type": "service_account", "pro

Then, in configuration, enter the following information

✓ **Container**

2 **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 *

Namespace *

Labels

Key *	Value
Key 1 * <input type="text" value="app"/>	Value 1 <input type="text" value="cloudockit"/>

[+ ADD KUBERNETES LABEL](#)

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.

[VIEW YAML](#)

Cluster

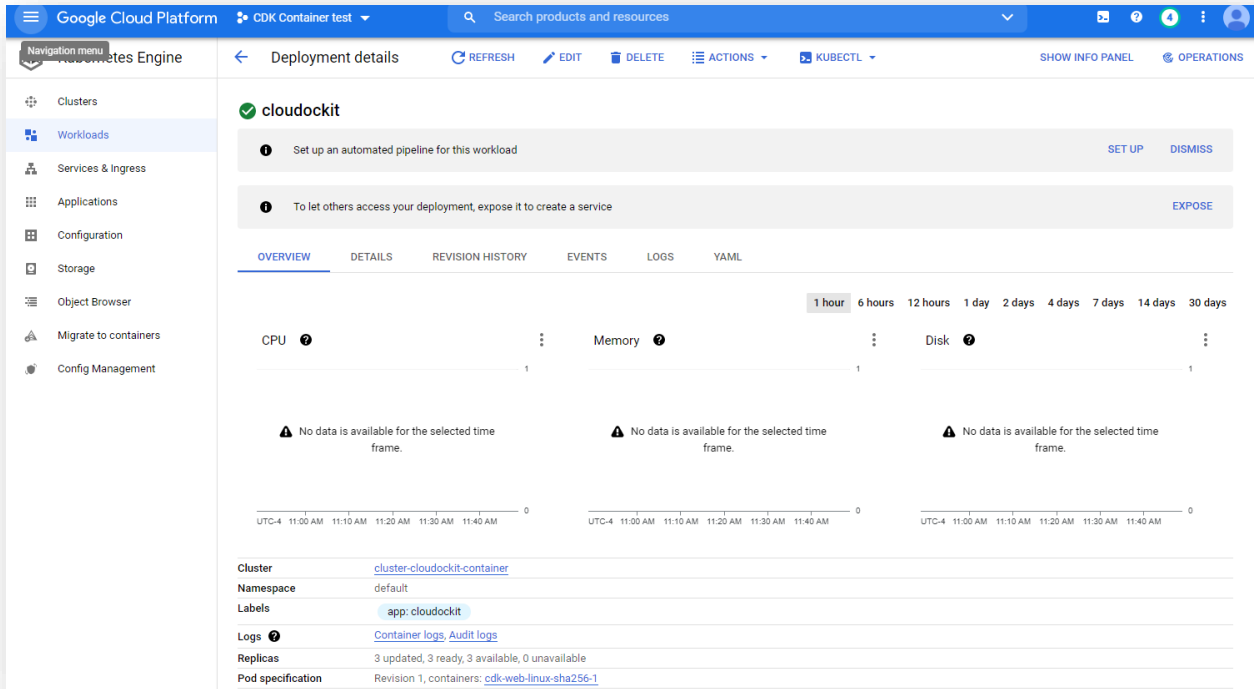
Kubernetes Cluster

Cluster in which the deployment will be created.

[CREATE NEW CLUSTER](#)

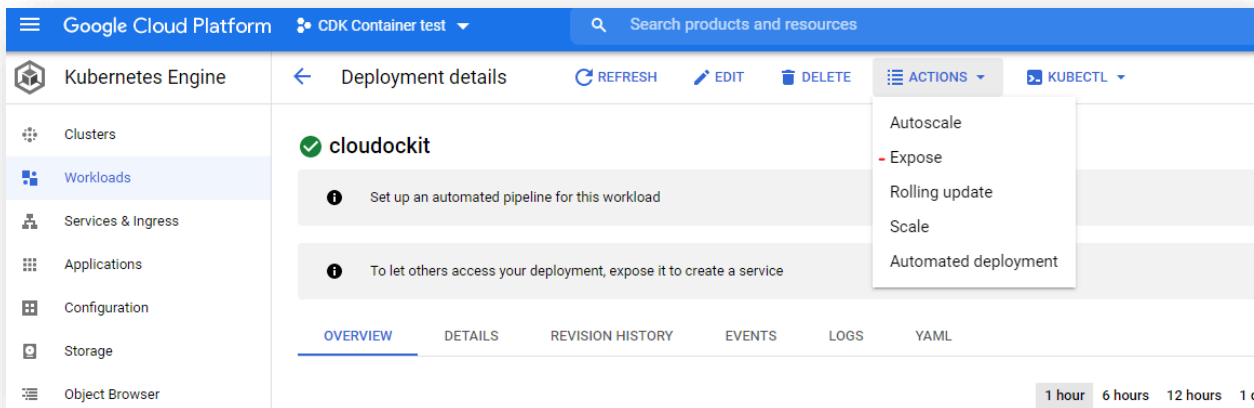
DEPLOY

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:

Expose

Expose a resource's Pods using a Kubernetes Service.

Port mapping

Port * ? Target port ? Protocol ?

80 TCP

+ ADD PORT MAPPING

Service type ?

Load balancer

* Indicates required field

CANCEL 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:

The screenshot displays the Kubernetes dashboard interface. On the left is a navigation sidebar with the following menu items: Clusters, Workloads (selected), Services & Ingress, Applications, Configuration, Storage, Object Browser, Migrate to containers, and Config Management. Below this are Marketplace and Release Notes. The main content area shows details for a workload named 'cluster-cloudockit-container' in the 'default' namespace. At the top, there is a graph showing a sharp spike in CPU usage at 11:50 AM, followed by a drop to zero. Below the graph, the following metadata is shown:

- 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](#)

The **Active revisions** section contains a table with one entry:

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

The **Managed pods** section contains a table with three entries:

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


The **Exposing services** section contains a table with two entries:

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

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 Clouddokit Web UI

Clouddokit 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

Register an application ...

* Name
The user-facing display name for this application (this can be changed later).
Clouddokit Web UI ✓

Supported account types
Who can use this application or access this API?
 Accounts in this organizational directory only (beauperinddev 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...](#)

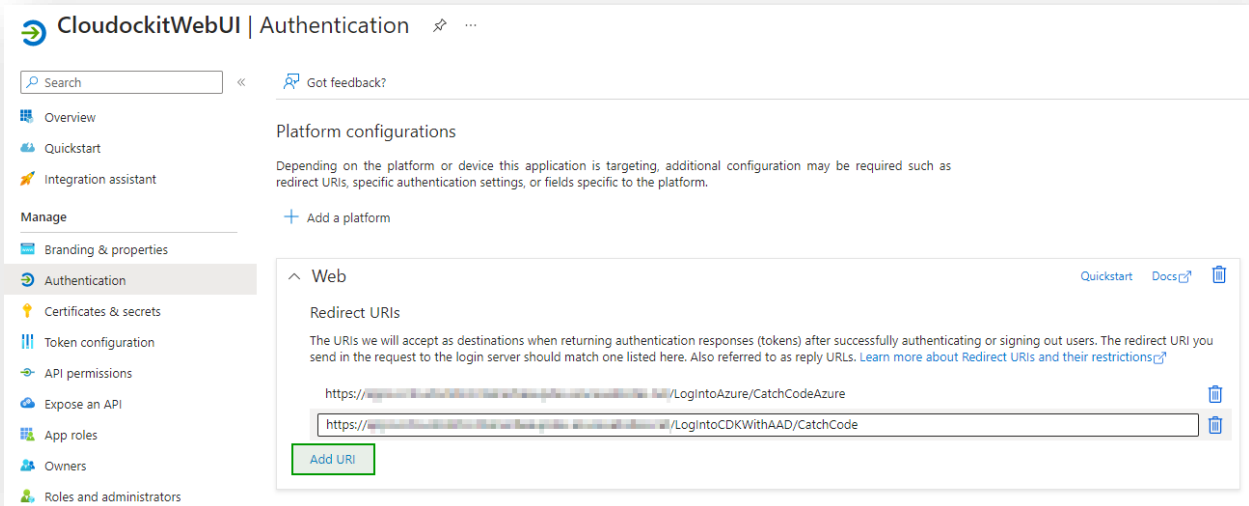
Redirect URI (optional)
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.
Web | `https://contoso.azurewebsites.net/LogIntoAzure/CatchCodeAzure` ✓

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](#)

Register

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**:

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)	
<input checked="" type="checkbox"/> User.Read ⓘ Sign in and read user profile	No
<input type="checkbox"/> User.Read.All ⓘ Read all users' full profiles	Yes
<input type="checkbox"/> User.ReadBasic.All ⓘ Read all users' basic profiles	No
<input type="checkbox"/> User.ReadWrite ⓘ Read and write access to user profile	No

Add permissions Discard

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

Refresh | Got feedback?

i The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission. 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 configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission ✓ Grant admin consent for UMAknow Solutions DEV Inc

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):

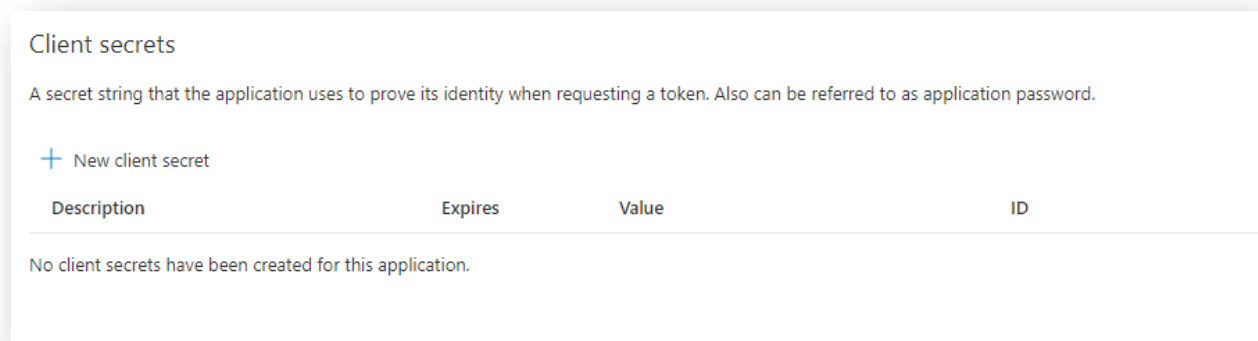
Configured permissions

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](#)

+ Add a permission ✓ Grant admin consent for Default Directory

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.



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 Clouddokit Container to support Scheduling.

Clouddokit 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 **clouddokitscheduler** image and set the appropriate settings in your settings file.

Start Clouddokit 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 **clouddokitscheduler** 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 Clouddokit 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: <ul style="list-style-type: none">• 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 Clouddokit 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 clouddokitinternal folder) to specify the URL of your Clouddokit Container:

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

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

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

Clouddokit 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.

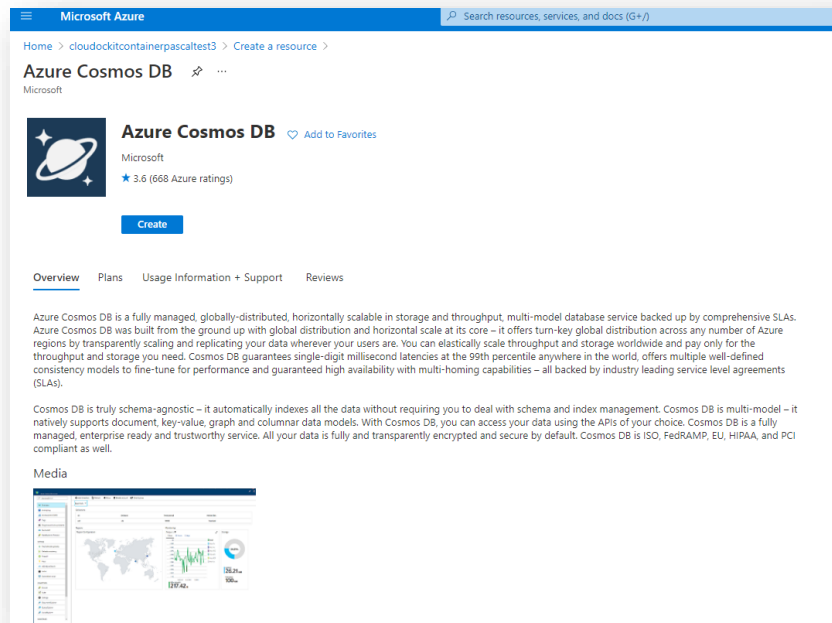
There are two steps required:

- Create (or re-use) an Azure Cosmos DB
- Add environment variables to the Clouddokit 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

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**:

The screenshot shows the Azure Cosmos DB Data Explorer interface. The main window displays a 'Welcome to Cosmos DB' message and two primary actions: 'Start with Sample' and 'New Container'. A 'New Database' dialog box is open on the right, with the 'Database id' field containing the text 'cloudockit'. The interface includes a left-hand navigation menu with options like Overview, Activity log, and Settings, and a top navigation bar with a search bar and 'New Container' button.

Configure Clouddockit Container to use the Azure Cosmos 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:

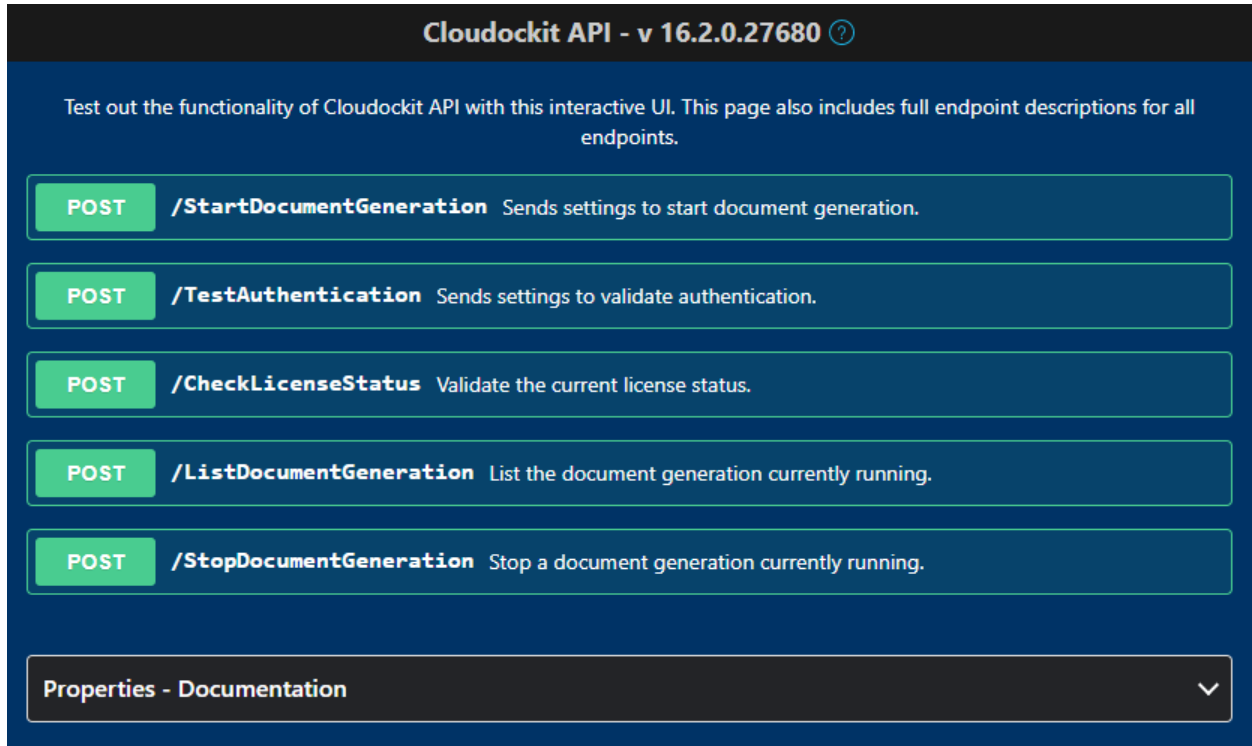
Name	Value
CosmosDb__DatabaseName	Enter the name of the Database that you have created in the previous step (clouddockit in the example)
ConnectionStrings__CosmosDb	Azure CosmosDB Connection string

Step F – Understand Clouddokit API Container

Once you have installed the Clouddokit 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 Clouddokit 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

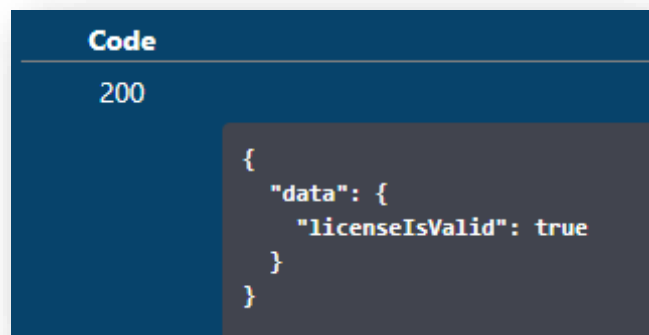
Once you get the API Key from Cludockit team and you have the appropriate credentials for the license validation, you can check that your API Key is working by using the **/CheckLicenseStatus** endpoint.

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 Cludockit API:

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

Click on Execute.

You should receive the following response body:



```
Code
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
	AWSScanRoleName	(Container only) Short role name (no ARN) assumed in each spoke account during scanning. Clouddokit builds <code>arn:aws:iam::<account>:role/<name></code> 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		<p>Do not change the name of the parameter for AWS, this is still call AzureStorageNameForDropOff</p> <p>You should specify one of this value:</p> <ul style="list-style-type: none"> the Azure Storage Account Name (it can be storage short name that is in the same tenant as the subscription that you scan <i>or</i> the complete Azure Storage Account Connection String) AWS S3 bucket GCP Bucket where Clouddokit 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": "XXXXXX",
  "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-----
nMIEvQIXXXXXZGy5PArVQS"n2buDji0URXCKoeWnukG9Cl0fHlP8rFK6+XXXXXX+kJm0Y
xuFOwxdbgpS1n38mQyez7EK"nObnp9wP05ynOxKXJqJx0rlk="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 J – Manage your document generation (Preview)

The Clouddokit 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.amazonaws.com/s3/aws4_request&X-Amz-Date=20201102T192525Z&X-Amz-SignedHeaders=host&X-Amz-Expires=3600&X-Amz-Credential=AKIAI4478V4M55243Q74:s3.amazonaws.com:us-west-2:aws4_request&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-SHA256=3091234567890123456789012345678901234567890123456789012345678901&X-Amz-Signature=3091234567890123456789012345678901234567890123456789012345678901"
                  "processID": 8420
                }
              ]
            }
          }
```

[/StopDocumentGeneration](#)

This endpoint is used to kill a document generation running.

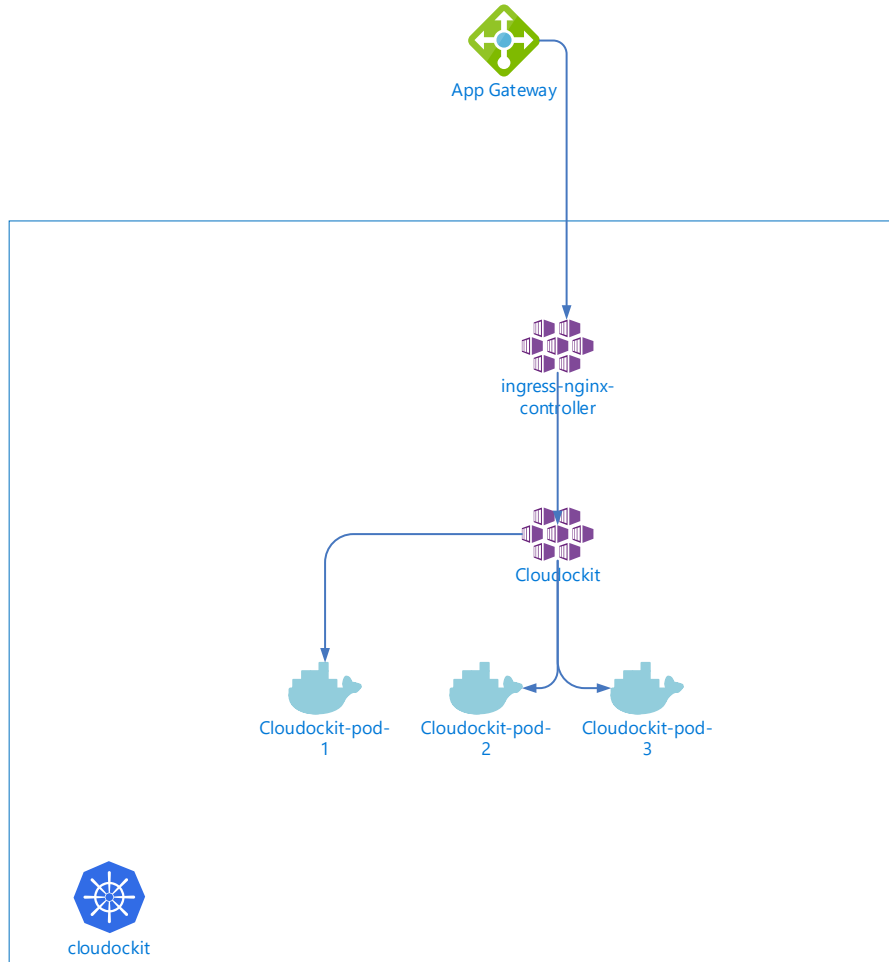
Name	Value
DockerProcessToKill	Value of the process ID to kill

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 Clouddockit Container

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



If you plan to use Clouddockit 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/clouddockit
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 Clouddokit Container.

- If you activate Clouddokit 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 Clouddokit 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.