Deployment to Google Cloud

1. **Create a Dockerfile containing the following:**

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["todo-list-api.csproj", "."]
RUN dotnet restore "./todo-list-api.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "todo-list-api.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "todo-list-api.csproj" -c Release -o /app/publish
/p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "todo-list-api.dll"]
```

*** The below will explain the dockerfile in further detail by each line:**

1.1. Setting up a .NETimage with version 6

1.2. Set the working directory

1.3. Expose the 80 port with 443 for http and https respectively

1.4. Copy the project, add to src directory and run build in Release mode in the /build directory

1.5. Create a publish version from build

1.6. Copy the contents to the Docker image directory

2. **Create a cloubuild.yaml file**

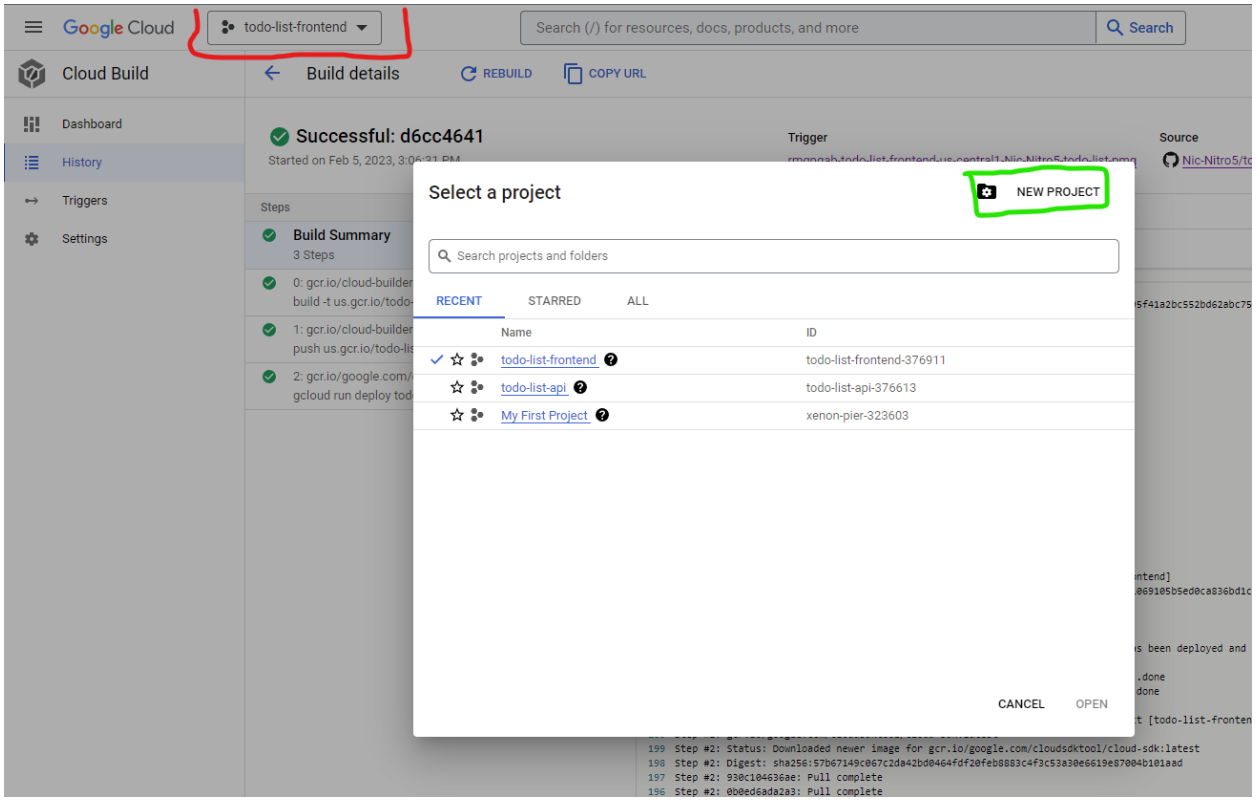
```yaml
steps:
 - name: gcr.io/cloud-builders/docker
   args:
    - build
    - '--no-cache'
    - '-t'
    - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
    - .
    - '-f'
    - Dockerfile
   id: Build
 - name: gcr.io/cloud-builders/docker
   args:
    - push
    - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
   id: Push
 - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk:slim'
   args:
    - run
    - deploy
    - $_SERVICE_NAME
    - '--image'
    - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
    - '--region'
    - $_DEPLOY_REGION
   entrypoint: gcloud
   id: Deploy
timeout: 1200s
images:
 - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
```

2.1. This file tells cloudbuild to use a docker image followed by running cloud build. It's going to give it a tag name using solution variables.

2.2. We use docker to push to that location.

2.3. We use the cloudrun sdk to deploy this.0

3. Create a Google Cloud user and create a new project



3.1. The red outline is where to create a new project.

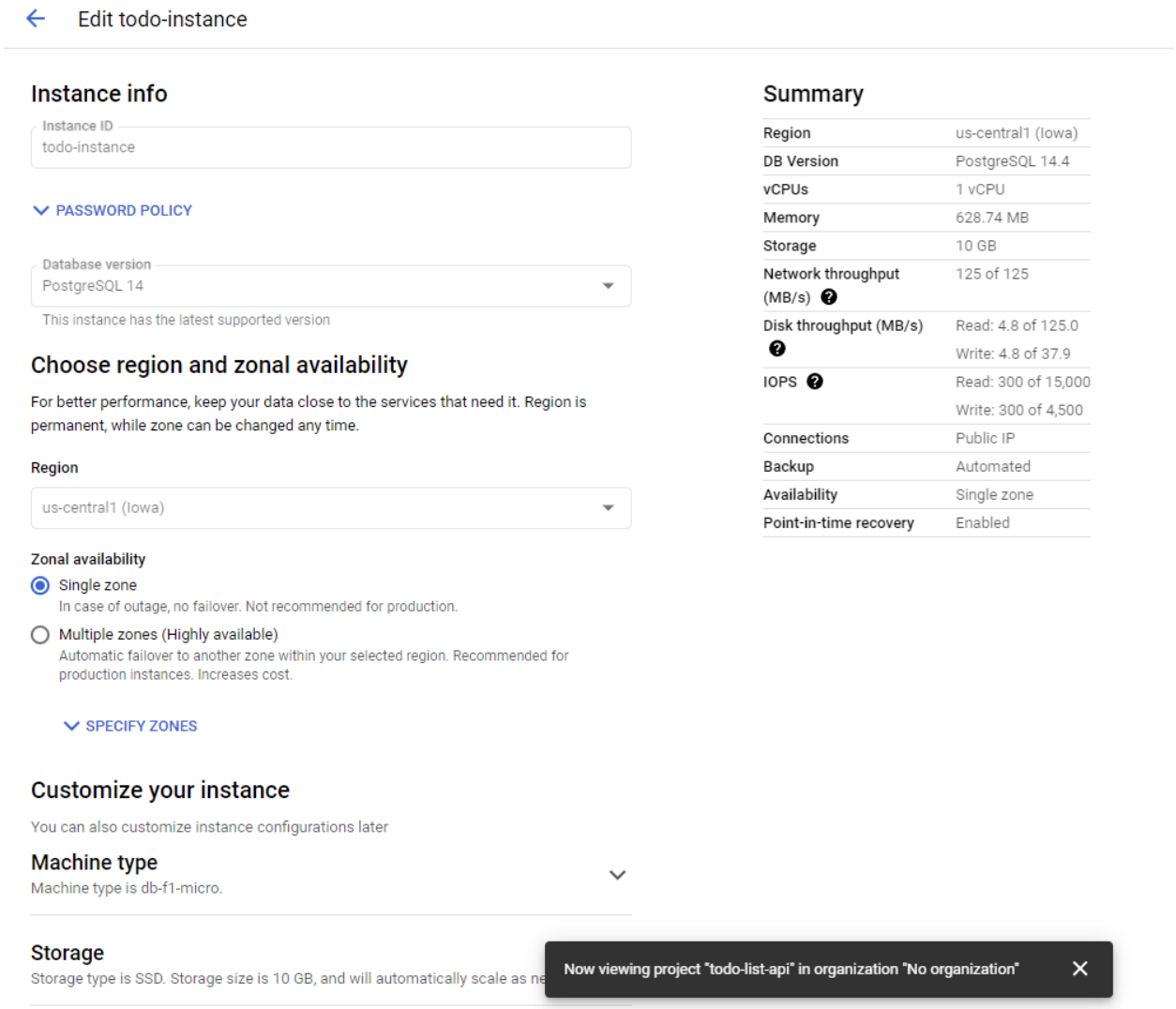
3.2. Click on the green outlined area at the top right of the popup to create the new project

4. Enable these APIs and services

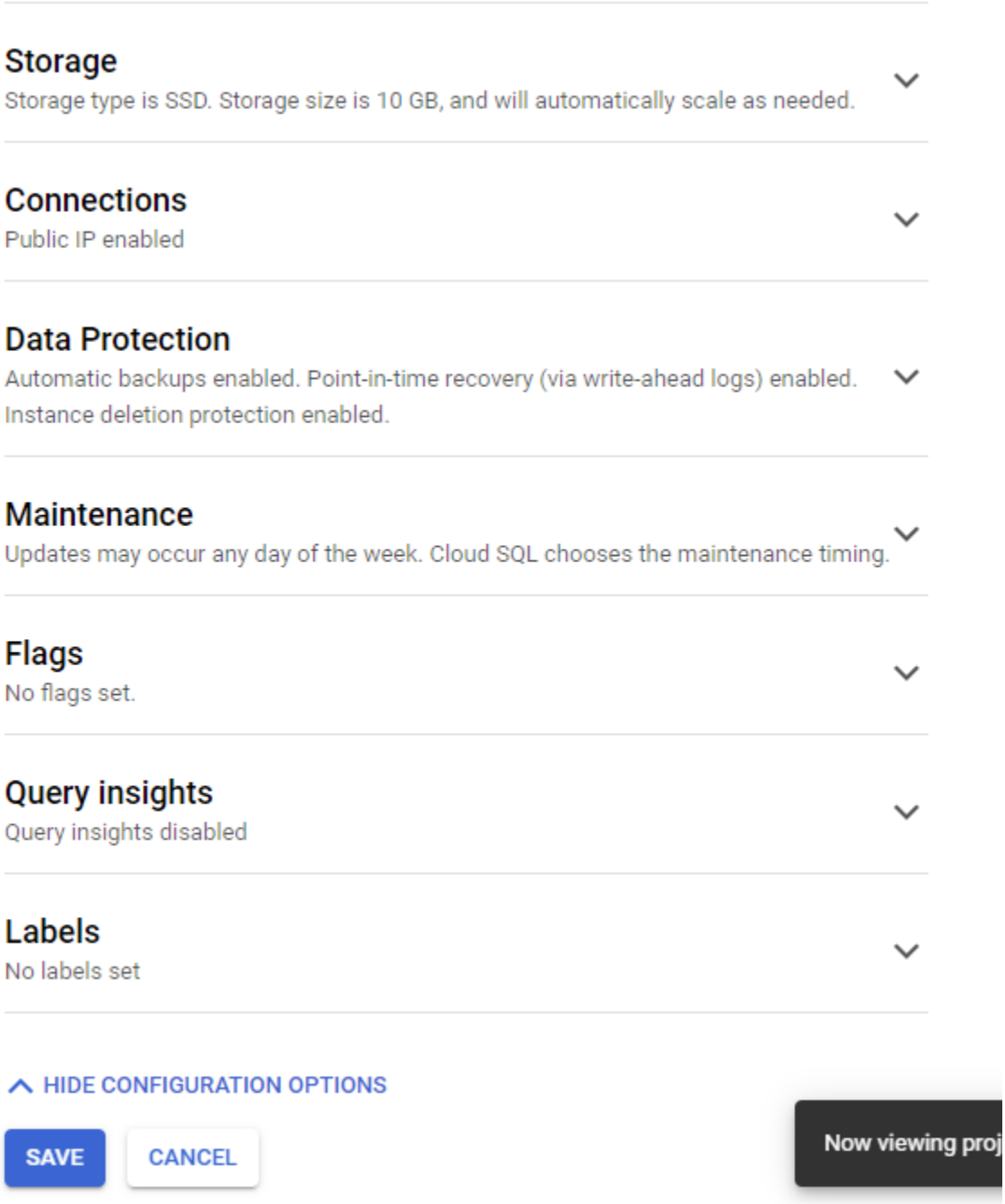
- Cloud SQL and API
- Cloud Build API
- Cloud Run API
- Secret Management API
- Compute API

4.1 Navigate to SQL in the sidebar

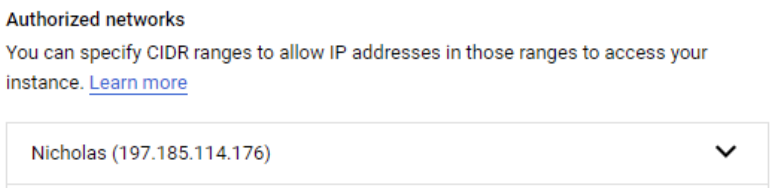
- Create a new instance and select PostgresSQL
- Enable the API
- Add a name and password
- Select your version
- Here you can configure according to your requirements
- Select the smaller size of the machine should you wish to reduce costs



- Enable public IP access in the connections tab and create instance

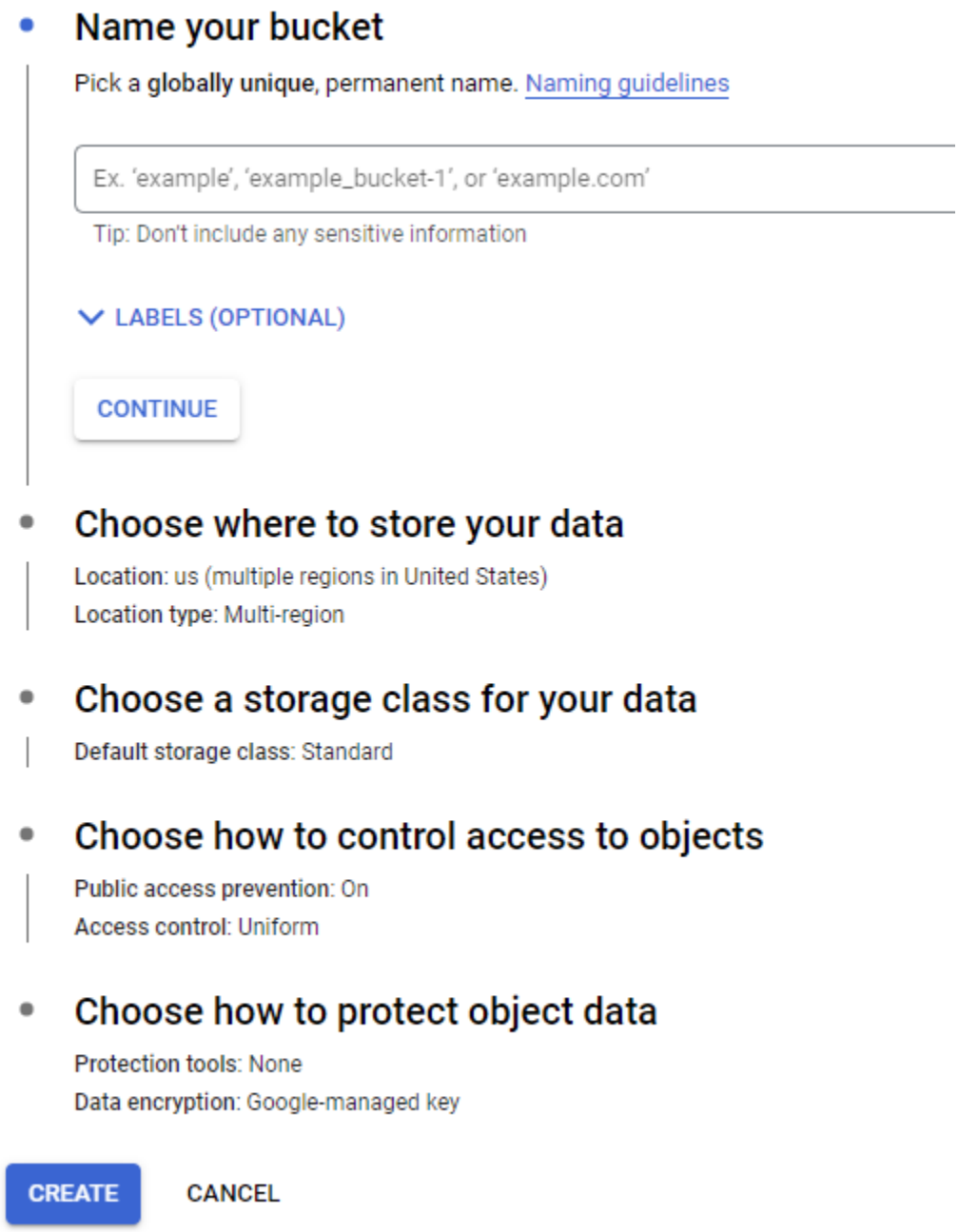


- Add the IP address to Authorized networks should you wish to connect to the DB from a local machine



4.2. Create a bucket (Cloud Storage)

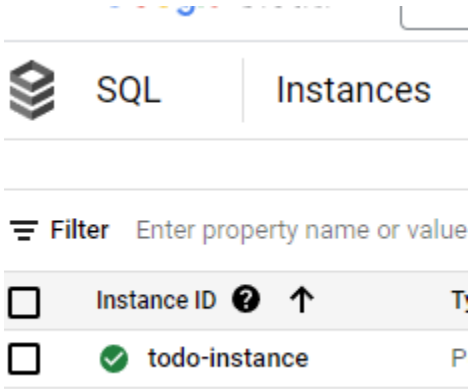
- Navigate to Cloud Storage in the sidebar on the left and click on create a bucket



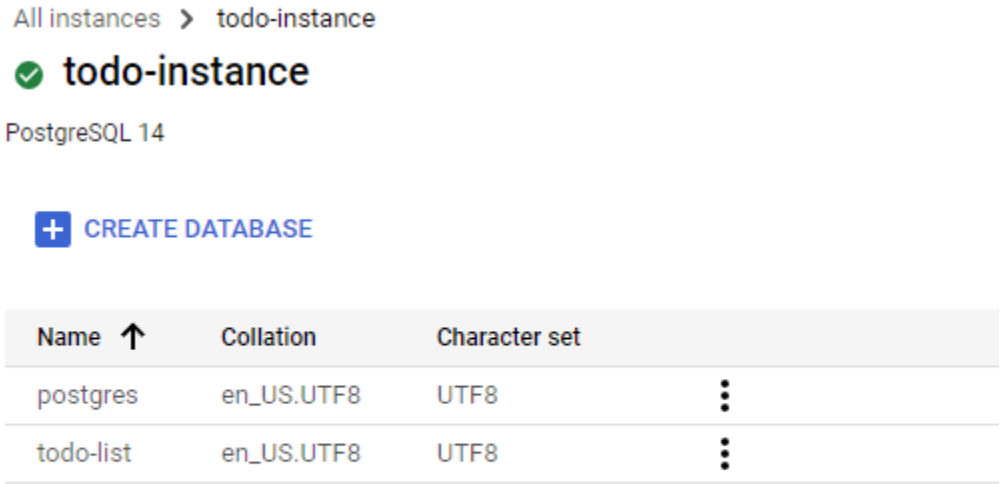
- Give the bucket a name
- Select your region
- Choose a storage class
- Access control as required
- Protect the data as required
- Create the bucket

5. Migrations for Database

- Navigate to the SQL tab and select your created database instance. Jere you will get your database details (hots, user etc)

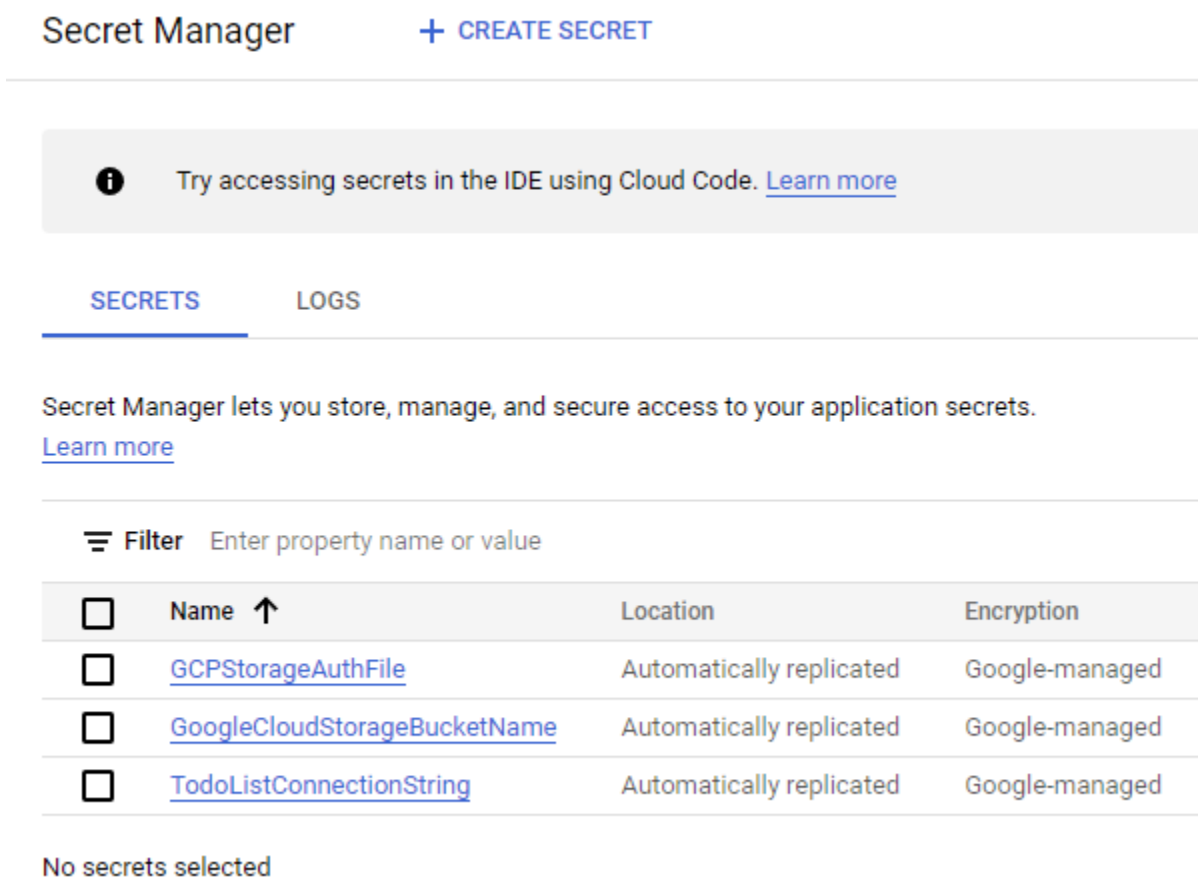


- **We will run migration scripts to update the cloud database**
    - Connect the cloud database in your IDE (appsettings.json) and an SQL client.
    - Run in the package manager console:
        - dotnet update-database
    - The database should reflect as below



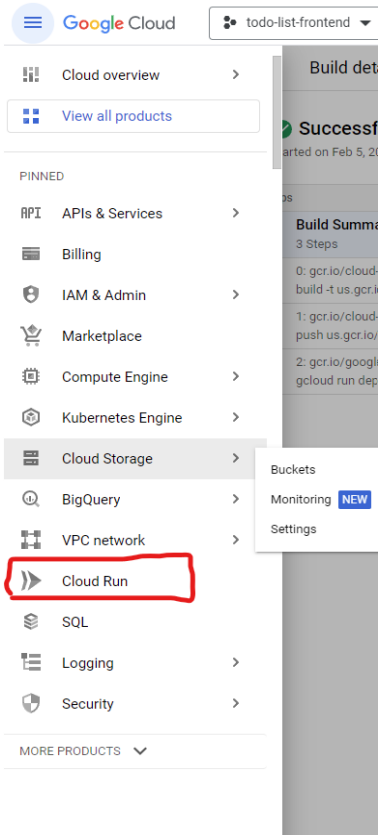
5.1. Security Manager

- Navigate to Security -> Secret Manager in the left sidebar
- Here we will need a secret for the bucket, connection string to db and a gcp auth file
- Values here need to be relative



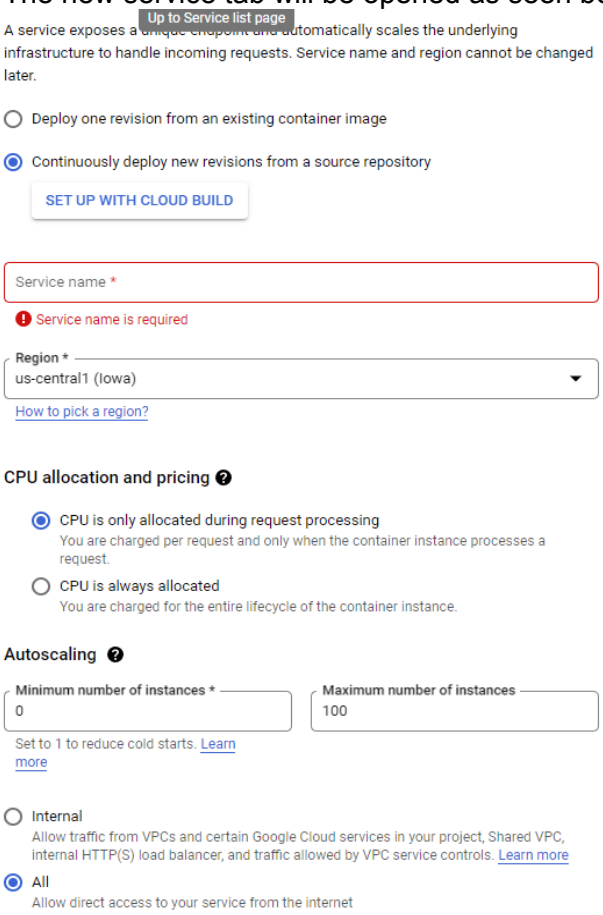
6. Navigate to Cloud Run to create a service

6.1. Click on the sidebar and select Cloud Run

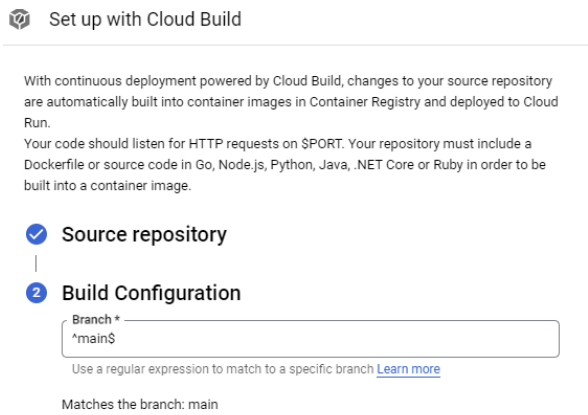


6.2. Click on Create Service

The new service tab will be opened as seen below:



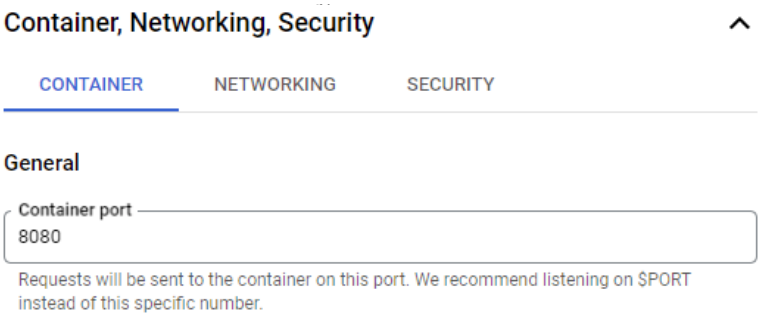
6.3. Select the continuously deploy new revisions option and link up the required repository by clicking on the setup build with cloud build, here you will configure which repo, branch and type of deployment you want to set up.



Choose the Build type as Dockerfile and save.

6.4. Allocate CPU as required.

6.5. Allow direct access to your service from the internet.



6.6. Allow unauthenticated invocations. Check this if you are creating a public API or website.

6.7. Set container **port to 80**

6.8. Should you wish you can check the Strartup CPU boost option

6.9. Configure the request timeout and max requests per container as required.

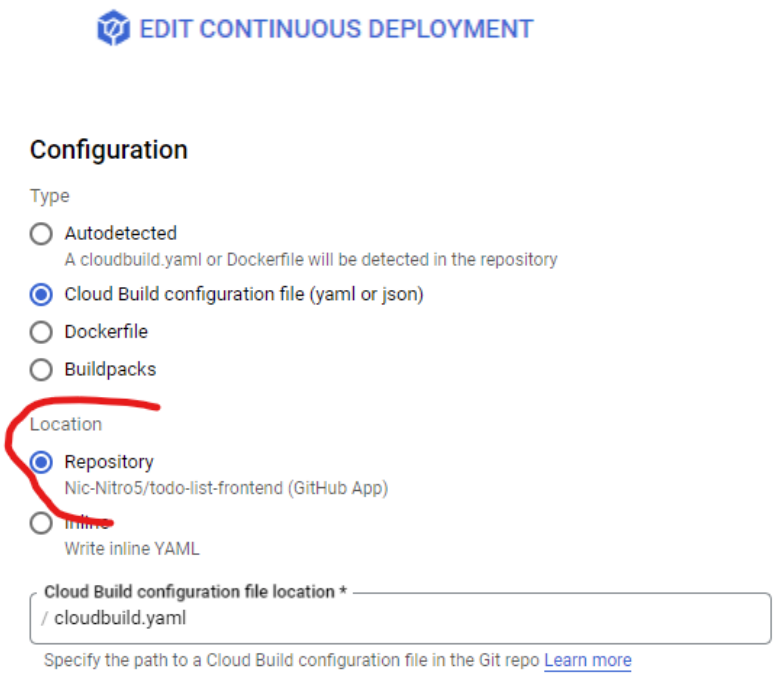
6.10. The networking and security can be configured as required.

6.11. Add your Database connection (Cloud SQL connections)

6.12. Reference and authorize all created secrets

- Expose these as environment variables
- Choose the version to be the latest

7. **Edit the continuous deployment**



Here we need to choose the location as Repository and set the path to our cloudbuild.yaml file. We will now have a deployment run every time we push to the branch we configure (main).

Upon successful deployment, you will now have access to the live URL.