Deployment to Google Cloud

1. Create a Dockerfile containing the following:

```
FROM node:16-alpine3.16 as build
WORKDIR /app
COPY ./package*.json ./

RUN npm ci

COPY . .
RUN npm run build

FROM nginx:1.23.0-alpine
EXPOSE 8080
COPY nginx.conf /etc/nginx/nginx.conf
COPY --from=build /app/dist/to-do-list /usr/share/nginx/html
```

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

1.1. Setting up a Node image with version 16, This will use the Alpine image

1.2. Set the working directory

1.3. Copy the package.json and the package-lock.json, using the wildcard with the * character

1.4. npm ci will only install the items from the package-lock.json

1.5. Copy rest of the files

1.6. Build the Angular project

1.7. Setup web server with nginx

1.8. Google clouds default port is 8080 so we will expose port 8080

1.9. Copy the nginx config file

1.10. Copy files from build to the nginx web server directory

2. Create an nginx.conf file

```
events {}
http {
    include /etc/nginx/mime.types;
    server {
        listen 8080;
        server_name localhost;
        root /usr/share/nginx/html;
        index index.html;
        location / {
            try_files $uri $uri/ /index.html;
        }
    }
}
```

*** The below will explain the nginx.conf in further detail by each line:**
2.1. We will listen on port 8080 as its the default port on Google Cloud Run

2.2. Server name will be localhost

2.3. Root will specify where our project files are

2.4. Index page will be index.html

2.5. Location will tell nginx how to manage incoming requests
 * We will try match the exact request or a match for that directory else if all fails we will default to index.html which is handled by Angular

3. Create a .dockerignore file and be sure to add all unwanted files and directories in the .dockerignore file

4. Create a cloudbuild.yaml file

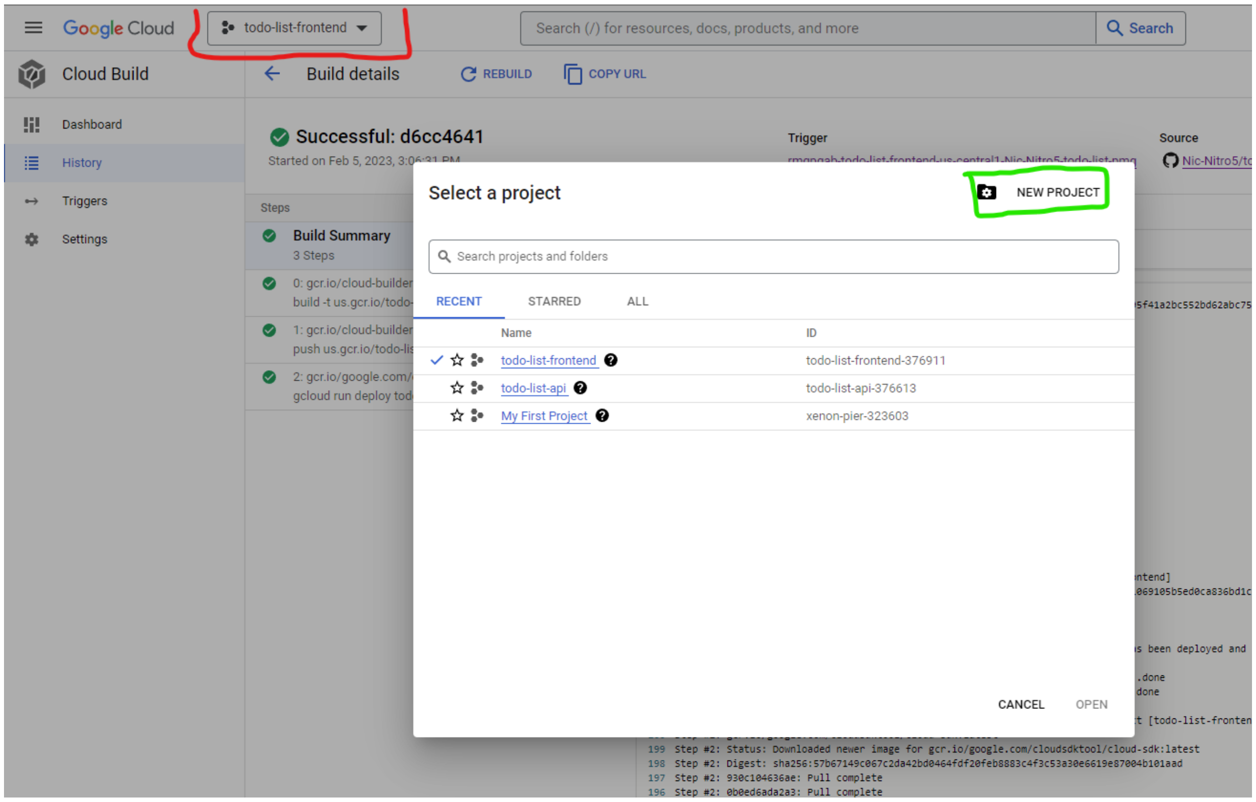
```
steps:
 - name: gcr.io/cloud-builders/docker
   args:
    - build
    - '-t'
    - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
    - .
 - name: gcr.io/cloud-builders/docker
   args:
    - push
    - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
 - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
   args:
    - run
    - deploy
    - $_SERVICE_NAME
    - '--image'
    - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
    - '--region'
    - $_DEPLOY_REGION
    - '--platform'
    - $_PLATFORM
   entrypoint: gcloud
timeout: 1200s
images:
 - '$_GCR_HOSTNAME/$PROJECT_ID/$_SERVICE_NAME:$COMMIT_SHA'
```

4.1. 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.

4.2. We use docker to push to that location.

4.3. We use the cloudrun sdk to deploy this.

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



Name	ID
todo-list-frontend	todo-list-frontend-376911
todo-list-api	todo-list-api-376613
My First Project	xenon-pier-323603

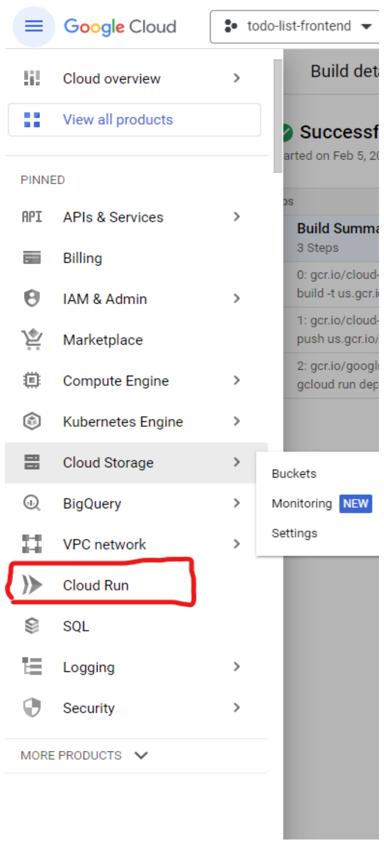
```
Step #2: Status: Downloaded newer image for gcr.io/google.com/cloudsdktool/cloud-sdk:latest
Step #2: Digest: sha256:57b67149c067c2da42bd0464fdf20feb8883c4f3c53a30e6619e87004b101aad
Step #2: 930c104636ae: Pull complete
Step #2: 0b0ed6ada2a3: Pull complete
```

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

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

6. Navigate to Cloud Run to create a service

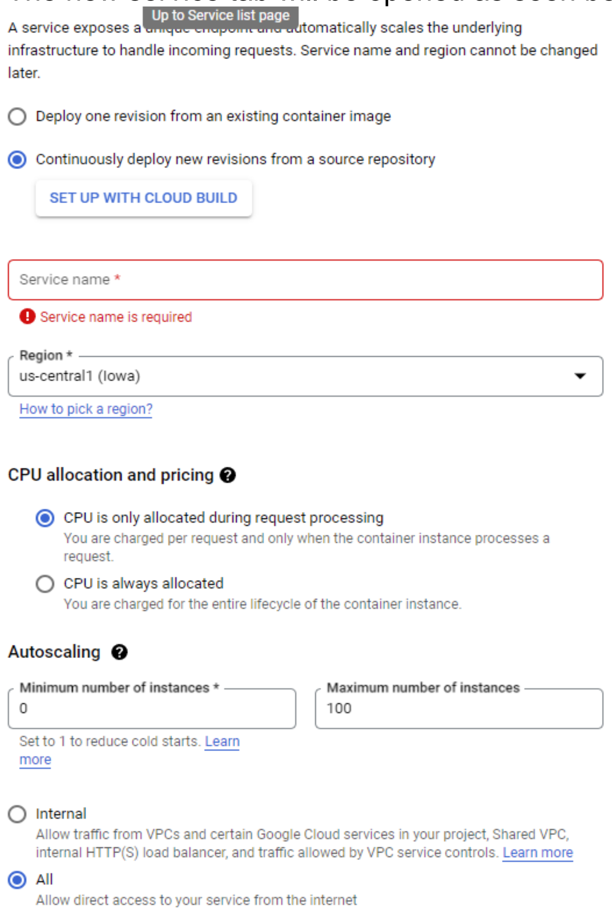
6.1. Click on the sidebar and select Cloud Run



6.2. Click on Create Service

CREATE SERVICE

The new service tab will be opened as seen below:



A service exposes a unique endpoint and automatically scales the underlying infrastructure to handle incoming requests. Service name and region cannot be changed later.

- Deploy one revision from an existing container image
- [✓ Continuously deploy new revisions from a source repository]

SET UP WITH CLOUD BUILD

Service name *

Service name is required

Region *: us-central1 (Iowa)

How to pick a region?

CPU allocation and pricing

- [✓ CPU is only allocated during request processing] You are charged per request and only when the container instance processes a request.
- CPU is always allocated. You are charged for the entire lifecycle of the container instance.

Autoscaling

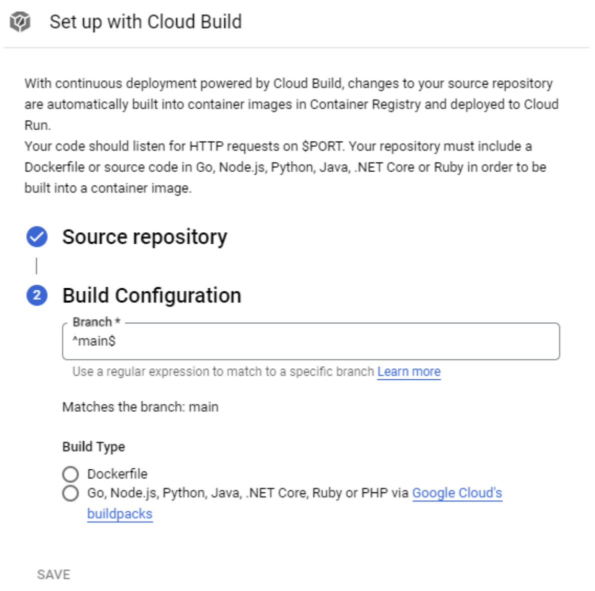
Minimum number of instances *: 0 — Set to 1 to reduce cold starts. Learn more

Maximum number of instances: 100

- Internal. Allow traffic from VPCs and certain Google Cloud services in your project, Shared VPC, internal HTTP(S) load balancer, and traffic allowed by VPC service controls. Learn more
- [✓ All] Allow direct access to your service from the internet

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.



Set up with Cloud Build

With continuous deployment powered by Cloud Build, changes to your source repository are automatically built into container images in Container Registry and deployed to Cloud Run.
Your code should listen for HTTP requests on $PORT. Your repository must include a Dockerfile or source code in Go, Node.js, Python, Java, .NET Core or Ruby in order to be built into a container image.

✓ Source repository

2 Build Configuration

Branch *: ^main$

Use a regular expression to match to a specific branch Learn more

Matches the branch: main

Build Type

- Dockerfile
- Go, Node.js, Python, Java, .NET Core, Ruby or PHP via Google Cloud's buildpacks

SAVE

6.4. Allocate CPU as required.

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

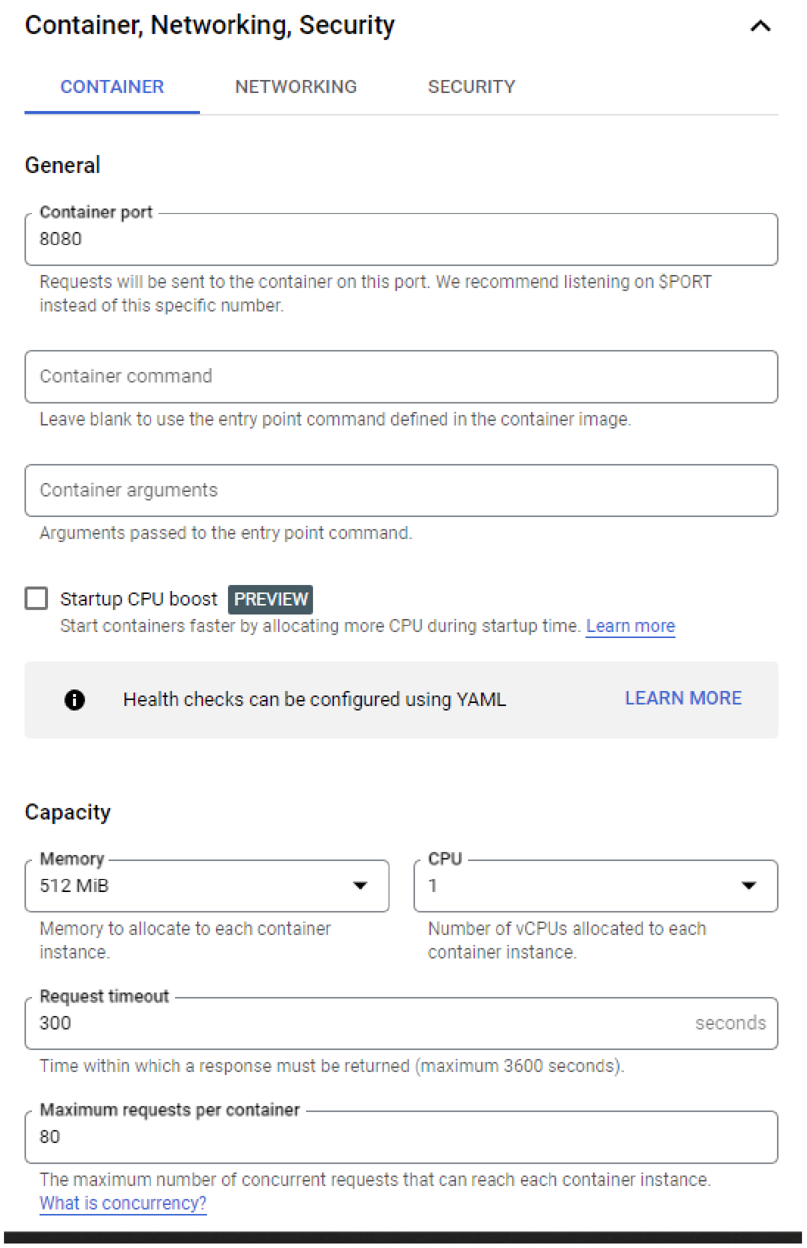
6.6. Allow unauthenticated invocations. Check this if you are creating a public API or website.

6.7. Set container port to 8080

6.8. Should you wish you can check the Startup 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.



Container, Networking, Security

General

Container port: 8080

Requests will be sent to the container on this port. We recommend listening on $PORT instead of this specific number.

Container command

Leave blank to use the entry point command defined in the container image.

Container arguments

Arguments passed to the entry point command.

Startup CPU boost PREVIEW

Start containers faster by allocating more CPU during startup time. Learn more

Health checks can be configured using YAML LEARN MORE

Capacity

Memory: 512 MiB

Memory to allocate to each container instance.

CPU: 1

Number of vCPUs allocated to each container instance.

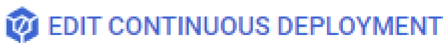
Request timeout: 300 seconds

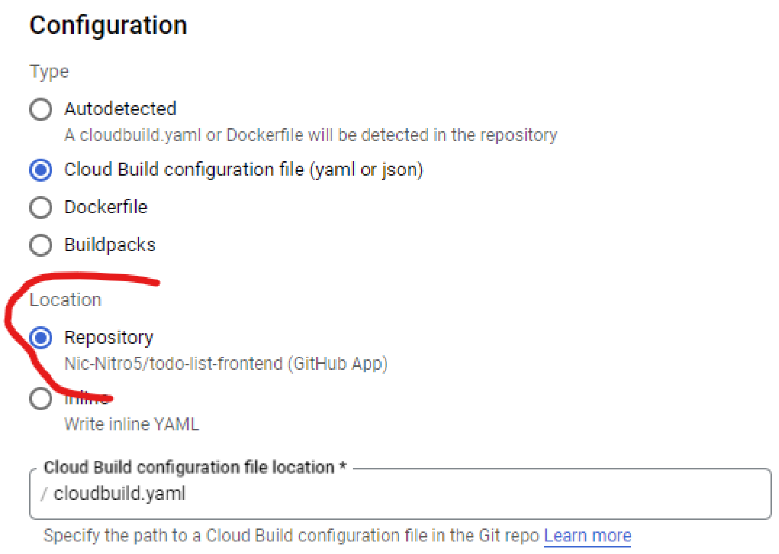
Time within which a response must be returned (maximum 3600 seconds).

Maximum requests per container: 80

The maximum number of concurrent requests that can reach each container instance. What is concurrency?

7. **Edit the continuous deployment**




Configuration

Type

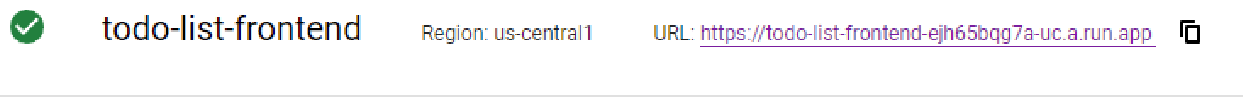
- [ ] Autodetected
  A cloudbuild.yaml or Dockerfile will be detected in the repository
- [✓ Cloud Build configuration file (yaml or json)]
- [ ] Dockerfile
- [ ] Buildpacks

Location

- [✓ Repository]
  Nic-Nitro5/todo-list-frontend (GitHub App)
- [ ] Inline
  Write inline YAML

Cloud Build configuration file location *
/ cloudbuild.yaml
Specify the path to a Cloud Build configuration file in the Git repo Learn more

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).


todo-list-frontend Region: us-central1 URL: https://todo-list-frontend-ejh65bqg7a-uc.a.run.app

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