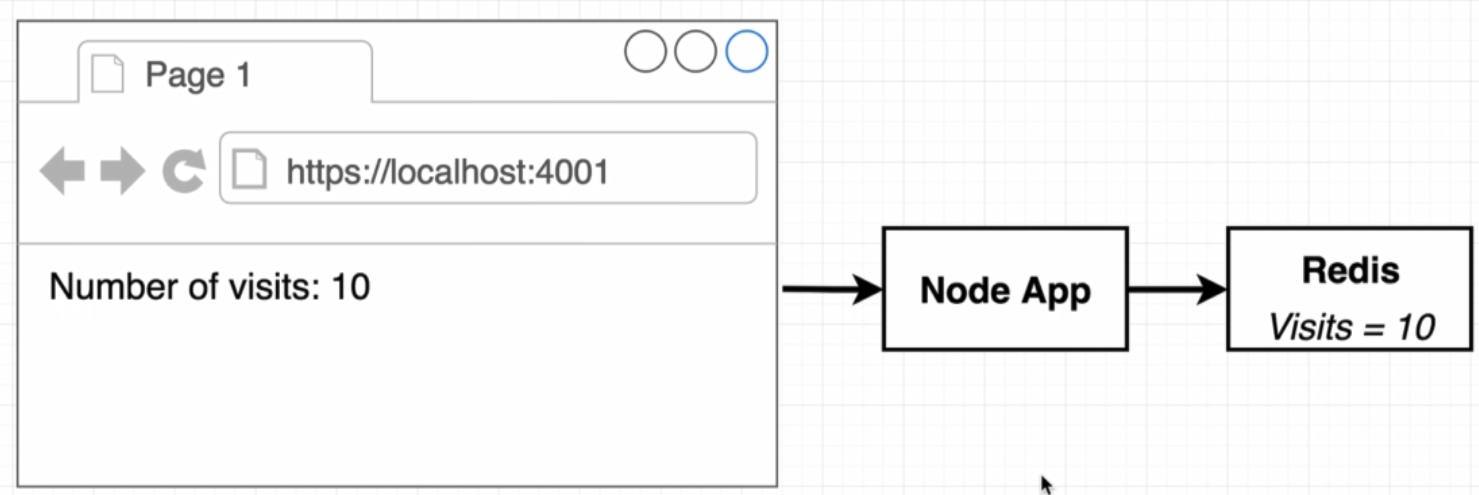
**05 Docker Compose with Multiple Local Containers**

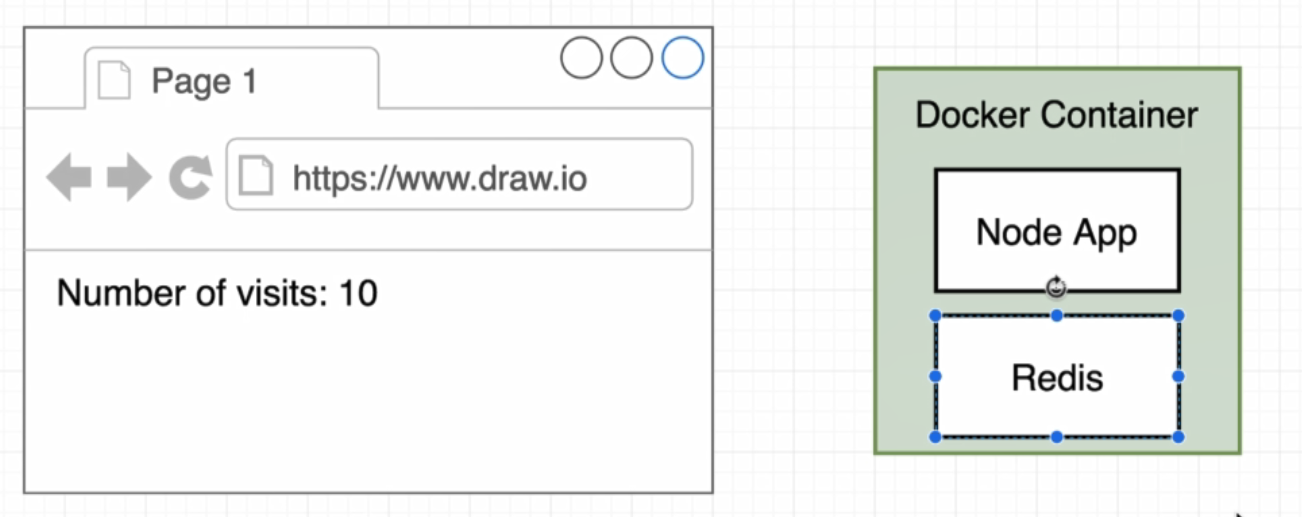
In this section we will create an container that contain web application which display the number of visited server, inside browser.

We need two separate component.

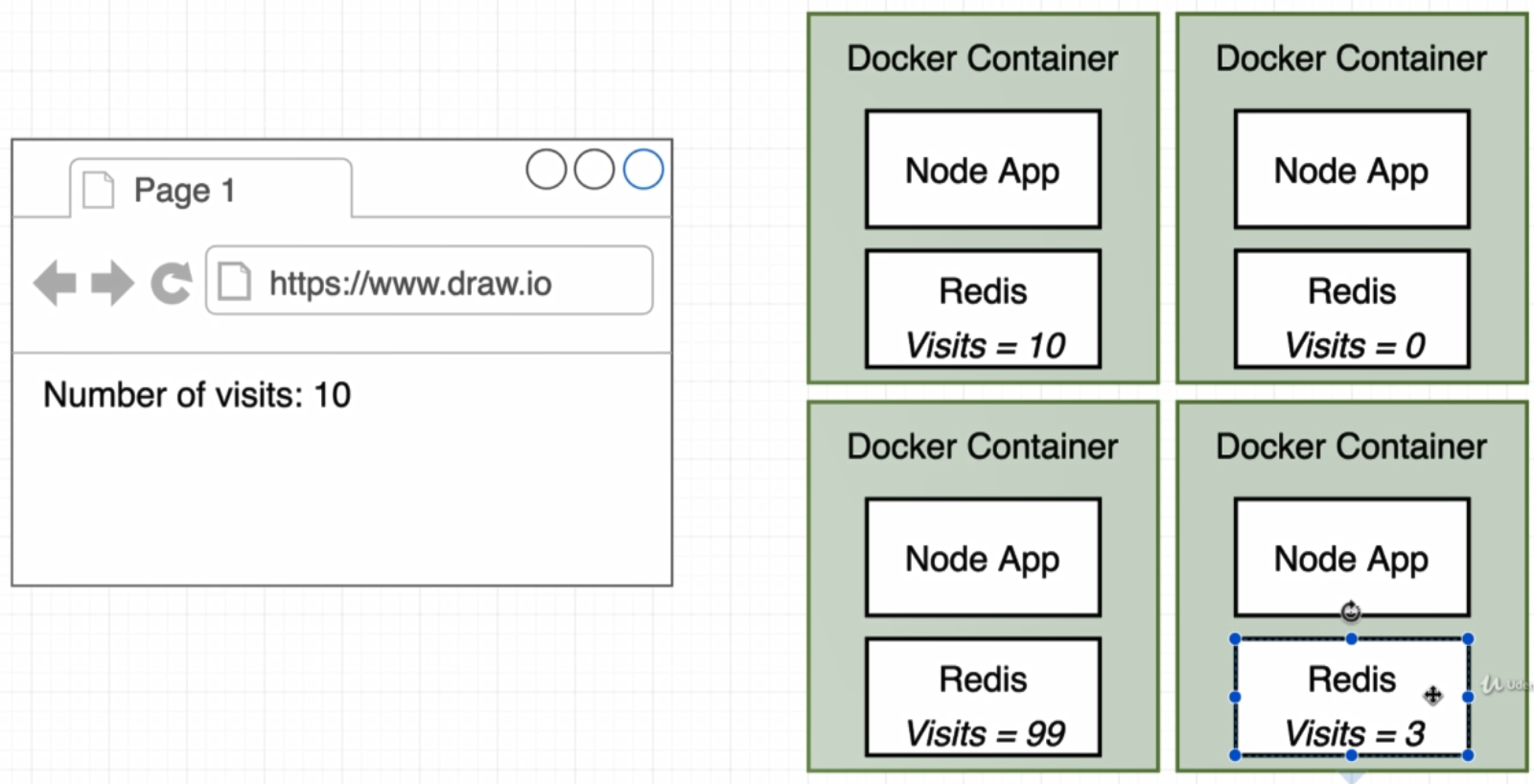
* A web server that respond to http request and generate html to show inside browser.
* A Redis(In memory data store) server to save number of time that the web server has been visited.



One of the possible architecture is like bellow but it has problem:

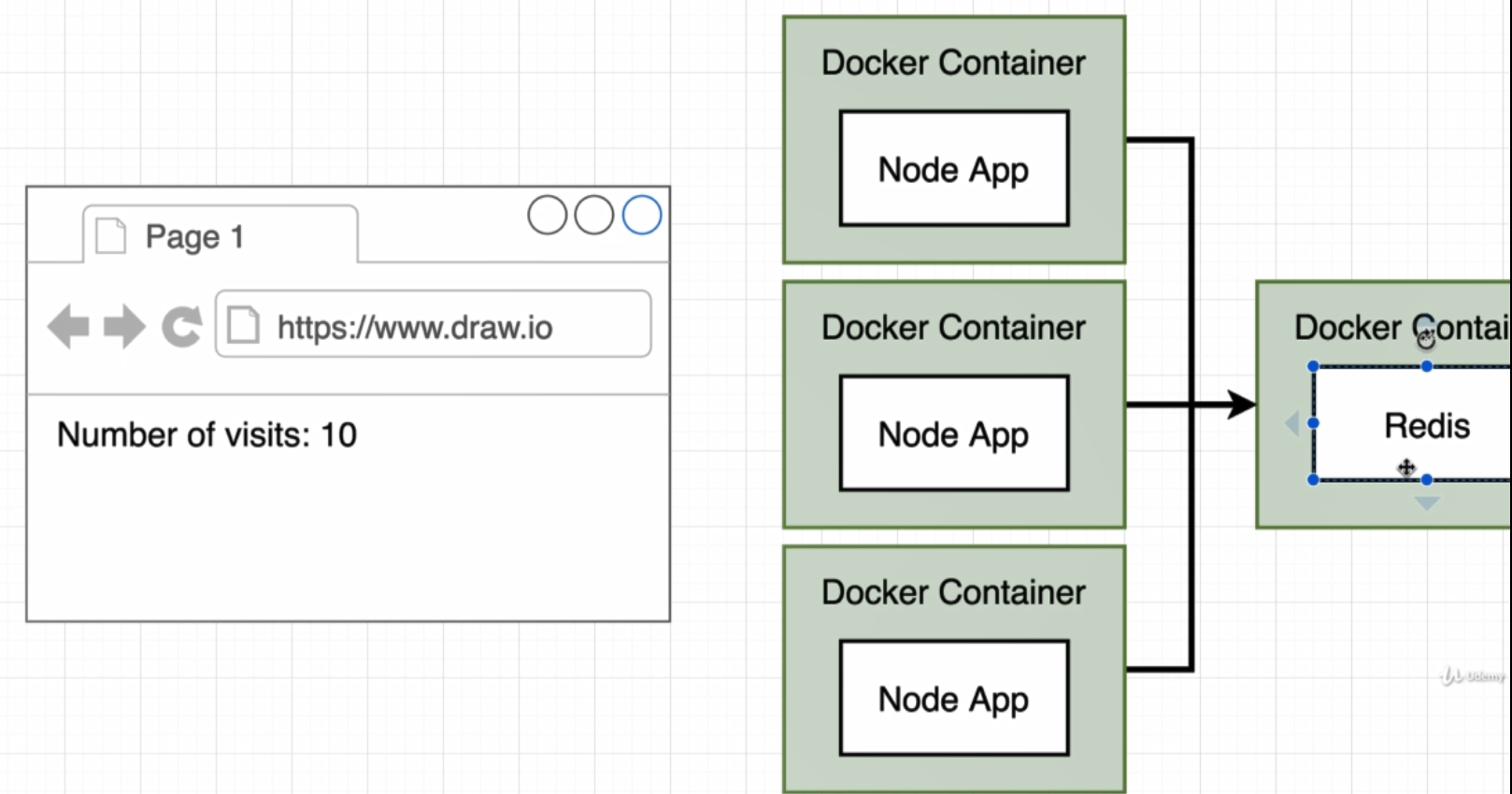


If our web site has more traffic to response and we want to create more than one server to response to request then we will have problem as shown in figure bellow:

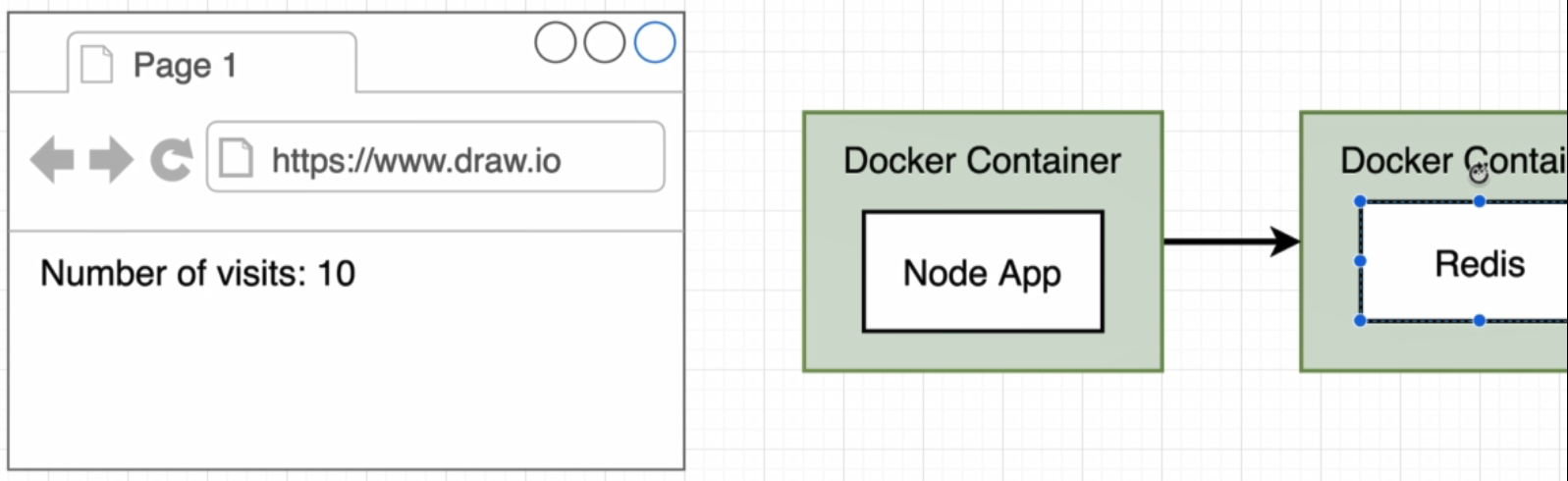


For every created server we will have copy of redis inside it and containers will not know about each other, each one will save ourselves visited count in redis.(99,10,0,3)

To increase web server without problem that described above, we can create containers as bellow:



To start for this section first we will create project as bellow:



First of all we create a node.js project with package.json and index.js file. After that create a Dockerfile with content bellow:

FROM node:alpine

WORKDIR '/app'

COPY package.json .

RUN npm install

COPY . .

CMD ["npm", "start"]

Now we can build over image:

* docker build . (build without tag)
* docker build –t tohid1987/visits:latest . (build with tag)

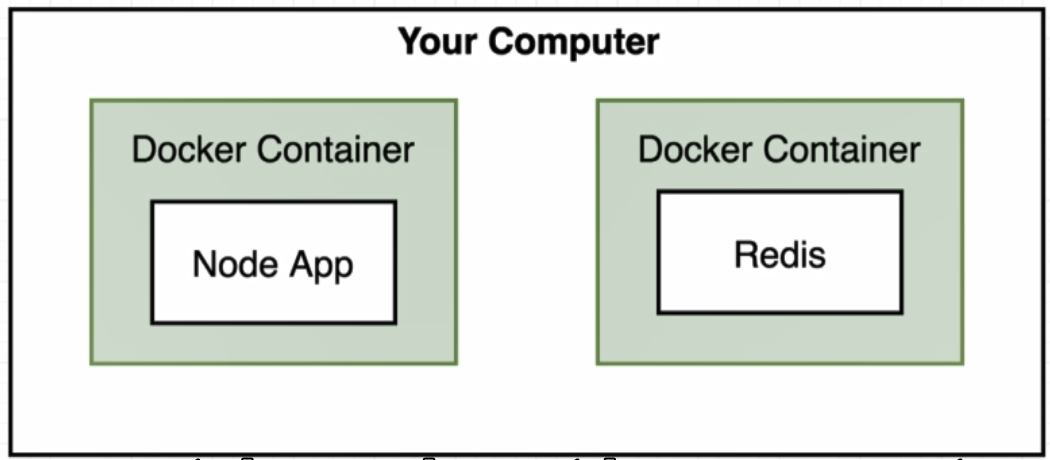
Now we have image with name **tohid1987/visits:latest**

By running container we will get an error because there is no redis server running to connect to.

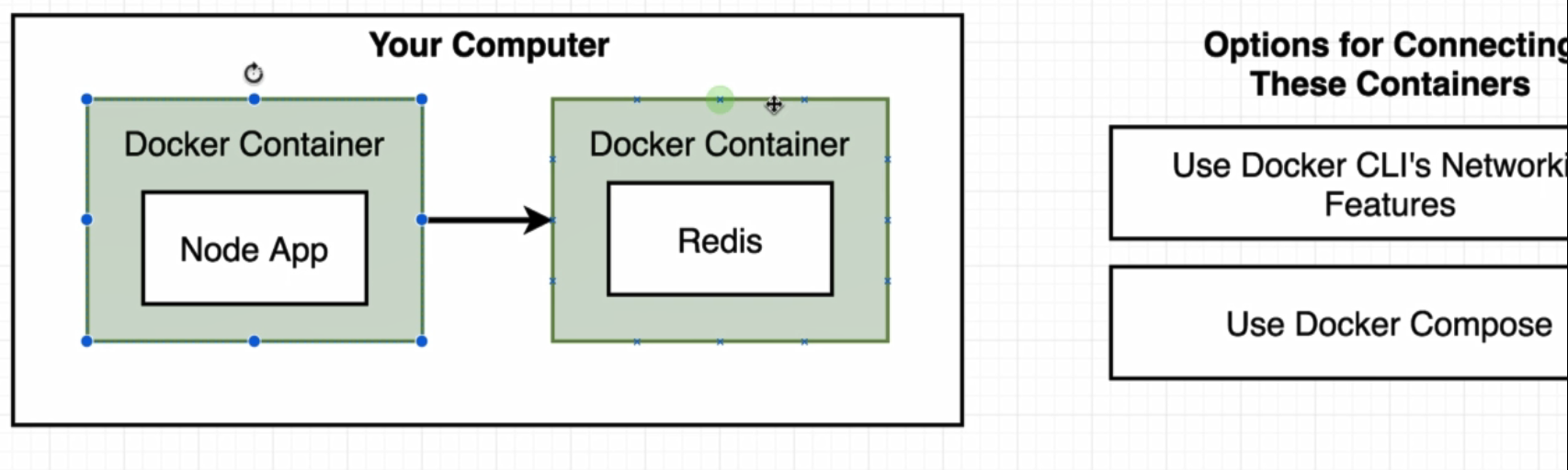
Before run visits we have to run redis before that :

* docker run redis
* docker run tohid1987/visits

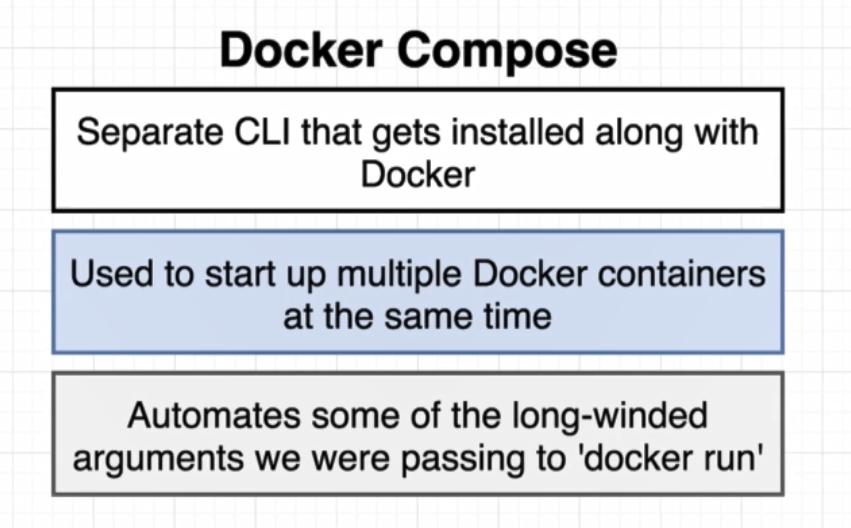
But this is not solve our problem because two container as shown below cannot communicate with each other.



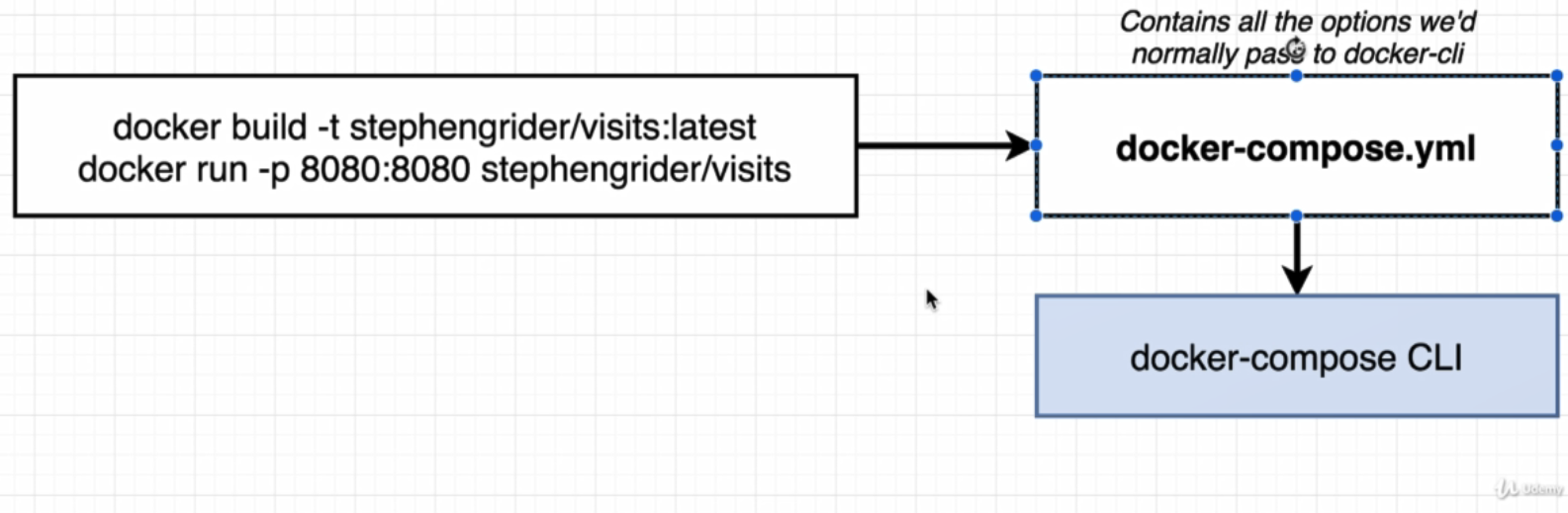
To make sure that our node app has ability to reach out to the redis server we need to set up network infrastructure between them. To do this we have two options:



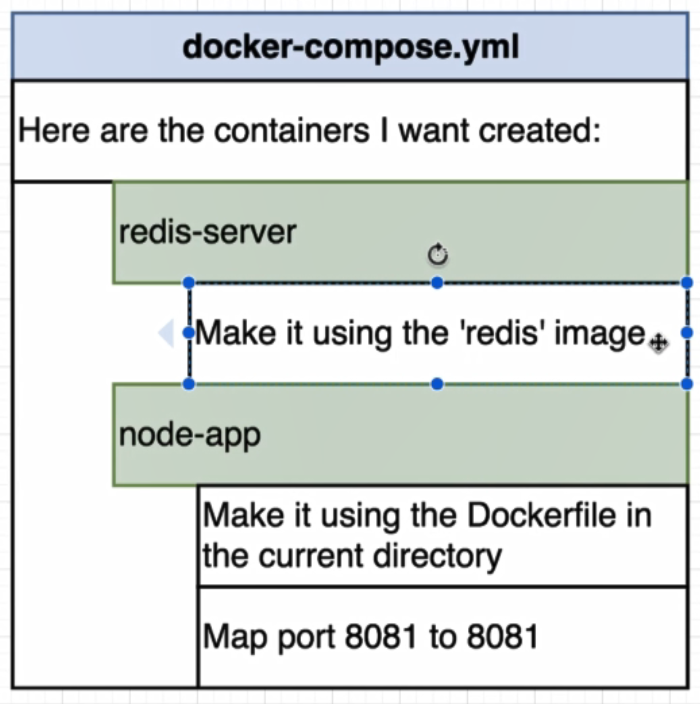
* Use Docker Cli’s networking features
  + It is not a good option because in every docker run we have to type a lot of options with that.
* Use Docker Compose
  + It is a separate tool that gets installed along with docker.



Now we are create a docker-compose file to run two docker cli command as shown bellow :



Create docker-compose for



# Version of docker compose that we will use in this page

version: '3'

services:

redis-server:

image: 'redis'

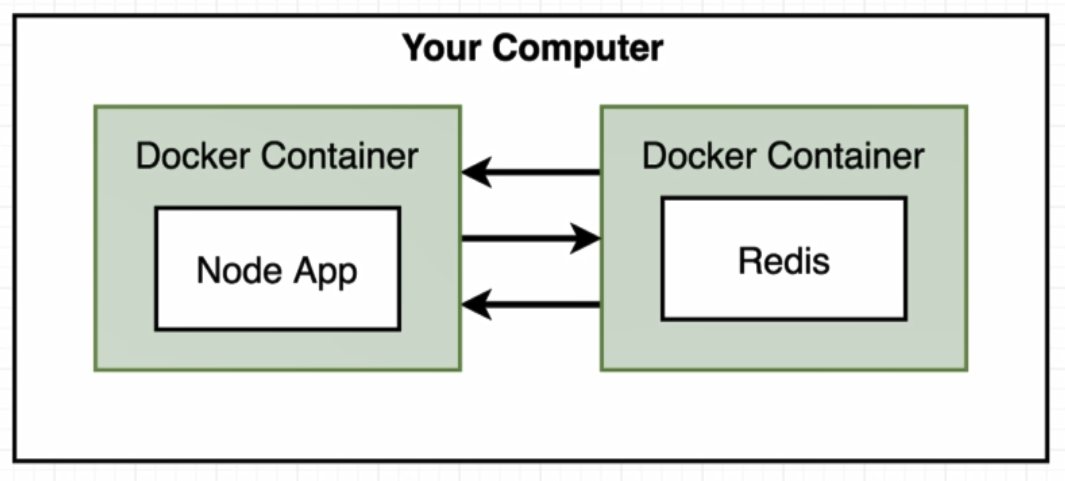
node-app:

build: . #build Dockerfile inside current directory to use it as image

ports:

- "4001:8081" # - in yml file specify an array

By creating the docker-compose , docker automatically create containers defined inside yml file in same network, so **there is no need to set network infrastructure between container.**



Inside index.js we should change createClient functiona s bellow:

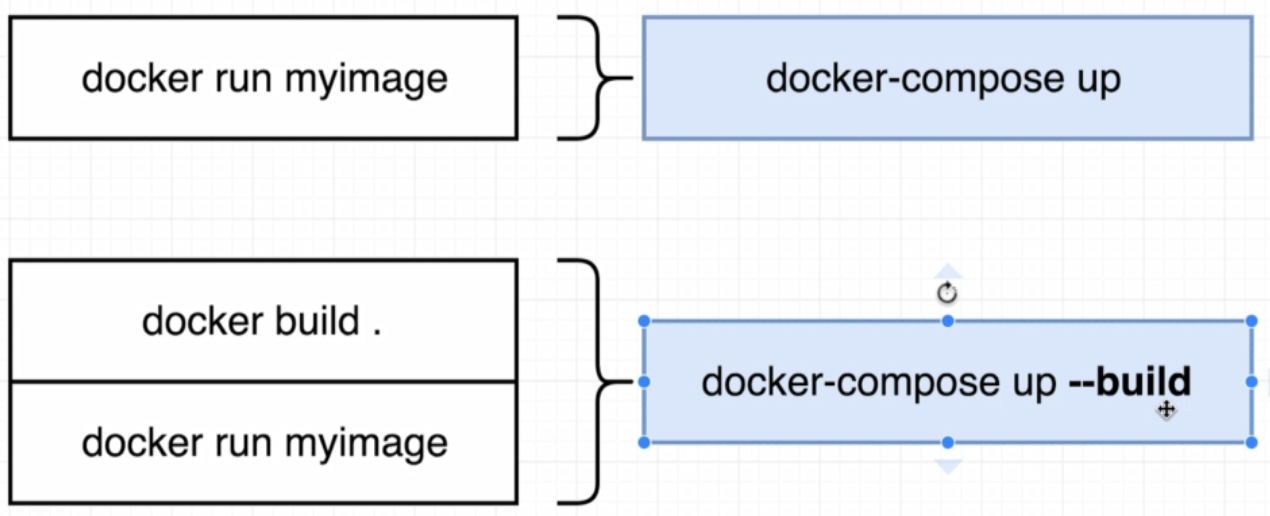
const client = redis.createClient({

host: 'redis-server', #Name of the service inside docker-compose.yml

port: 6379 # default port for redis

});

To run docker-compose file:

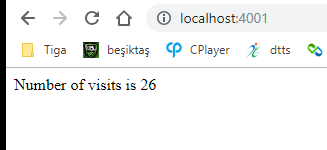


* docker-compose up

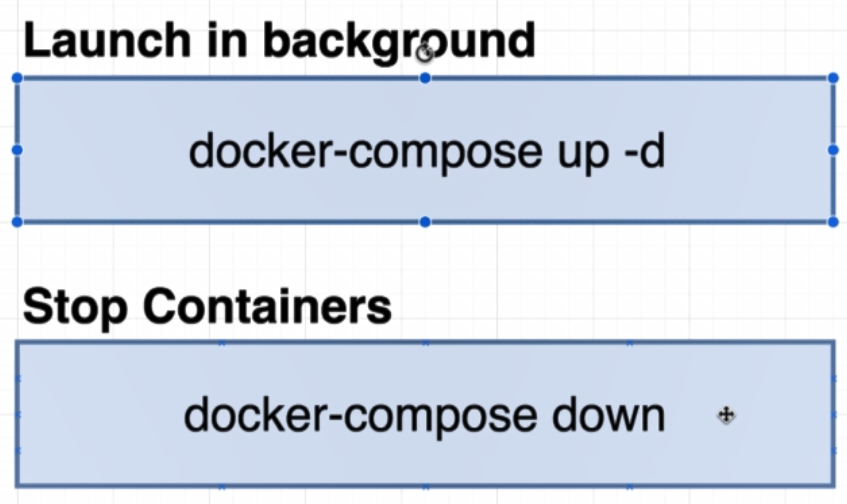


Create a network to join container inside dpcker-compose.yml file.

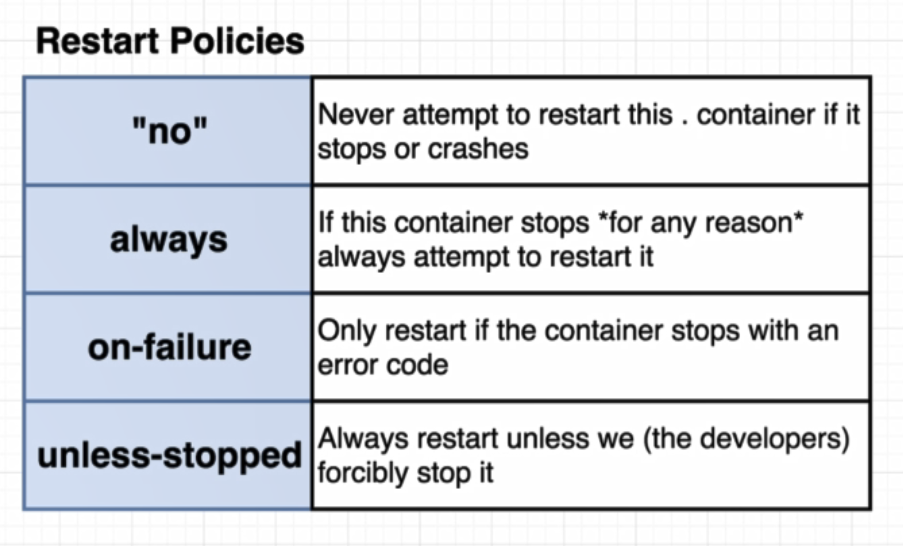
Result:



To stop docker compose we can use command bellow:



If we get error from docker we can restart a container like this (by default we have no restart policies on docker compose ):



node-app:

restart: always ################ restart if an error accure on node app

build: . #build Dockerfile inside current directory to use it as image

ports:

- "4001:8081" # - in yml file specify an array