Deploying an R Shiny Dashboard on GCP Cloud Run

Poorna Chathuranjana · Follow
6 min read · Feb 4, 2023

img_1.jpg

If you are into statistical data analysis and forecasting, you are probably already familiar with R programming language. Then, you must know that R shiny dashboards can be used to visualize data easily, without worrying much about the user interface (UI). You can use Google Cloud Platform (GCP) to host R shiny dashboard apps, since GCP Cloud Run started supporting WebSockets. We will be creating a container image with the app using docker.

(This was tested with Ubuntu 18.04)

0. Prerequisites

- Install R
- Install RStudio
- Setup Docker (or to install, you could try my older post with appropriate updates.)
- Google Cloud CLI (gcloud)

1. Understanding the directory & file tree

To dockerize, we will be using the following directory and file structure.

```
CloudRunShinyDashboard/
  |-- Dockerfile
  |-- shiny-server.conf
  |-- shiny-server.sh
  |-- app/
        |-- app.R
        |-- images/
              |-- img_1.jpg
```

- app.R = R shiny dashboard app.
- img_1.jpg (optional) = An image to be shown on the dashboard body. (change the code accordingly, if you decide not to have the image.)
- Dockerfile = To create the container image.
- shiny-server.conf = Shiny app server configurations.
- shiny-server.sh = To execute shiny app server.

2. Create app.R

We will create a relatively simple shiny app with a decent UI. The following packages can be used initially.

- shiny — To develop web apps with R.
- shinydashboard — Use shiny to create dashboards.
- shinydashboardPlus — Additional features for shinydashboard package.
- shinythemes — Some themes for the shiny app.

You can use RStudio to code and test this locally by simply changing `img_dir` path from `“/srv/shiny-server/images”` to `"images"` .

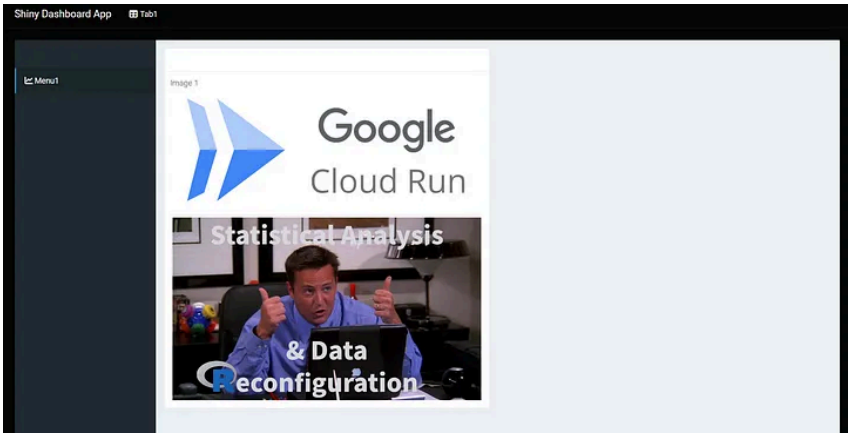
```
library(shiny)
library(shinydashboard)
library(shinydashboardPlus)
library(shinythemes)

img_dir = "/srv/shiny-server/images"
```

```
ui <- fluidPage(theme = shinytheme("cyborg"),
  navbarPage(
    title = "Shiny Dashboard App",
    windowTitle = "Shiny Dashboard App",
    id = "tabactive",
    tabPanel("Tab1",
             icon = icon("table"),
             tags$body(
               dashboardPage(
                 dashboardHeader(title = "Tab1", disable = TRUE),
                 dashboardSidebar(
                   width = "250",
                   sidebarMenu(id = "sidebarmenu",
                               menuItem("Menu1", tabName = "menu1", icon = icon(
                   ),
                   minified = FALSE
                 ),
                 dashboardBody(
                   tabItems(
                     tabItem(tabName = "menu1",
                             fluidRow(box("Image 1",
                                          imageOutput("img_1", height = "auto"))
                             )
                     )

                   )
                 )
               )
             )
            )
  )

)

server <- shinyServer(function(input, output, session){

  output$img_1 <- renderImage({
    list(src = paste(img_dir, "img_1.jpg", sep = "/"),
         contentType = "image/jpg",
         width = "100%",
         height = "auto",
         alt = "img_1_err")
  }, deleteFile = FALSE)

})

shinyApp(ui = ui, server = server)
```

In addition, you can use the following packages to implement more functionalities.

- shinyBS — More features for shiny, such as tooltips.
- tseries — To work with time series data.
- forecast — To forecast with data. (principles)
- data.table — An enhanced and high-performance version of data.frame
- DT — To visualize DataTables.
- ggplot2 — Plot data
- plotly — Visualize plots
- bigrquery — Load data from GCP BigQuery (you may need "*readr*" package too.)

The dashboard may look as follows.



Dashboard UI

3. Create the Dockerfile

The Dockerfile should look as follows.

```
# get shiny server plus tidyverse packages image
FROM rocker/shiny-verse:latest

# system libraries of general use
RUN apt-get update && apt-get install -y \
    curl \
    sudo \
    pandoc \
    pandoc-citeproc \
    libcurl4-gnutls-dev \
    libcairo2-dev \
    libxt-dev \
    libssl-dev \
    libssh2-1-dev\
    ## clean up
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/ \
    && rm -rf /tmp/downloaded_packages/ /tmp/*.rds

# install R packages required
# (change it depending on the packages you need)
RUN R -e "install.packages('shiny', repos='http://cran.rstudio.com/')"
RUN R -e "install.packages('shinydashboard', repos='http://cran.rstudio.com/')"
RUN R -e "install.packages('shinydashboardPlus', repos='http://cran.rstudio.com/
RUN R -e "install.packages('shinythemes', repos='http://cran.rstudio.com/')"

# clean up
RUN rm -rf /tmp/downloaded_packages/ /tmp/*.rds

# Copy configuration files into the Docker image
COPY shiny-server.conf  /etc/shiny-server/shiny-server.conf

# Copy shiny app into the Docker image
COPY app /srv/shiny-server/

RUN rm /srv/shiny-server/index.html

# Make the ShinyApp available at port 5000
EXPOSE 5000

# Copy shiny app execution file into the Docker image
COPY shiny-server.sh /usr/bin/shiny-server.sh

USER shiny

CMD ["/usr/bin/shiny-server"]
```

Remember to add lines to install the packages you are using in "**app.R**". In addition, use the same port number (e.g. — 5000) throughout the process.

4. Create shiny-server.conf

Remember to use the same port number (e.g. — 5000).

```
# Define the user we should use when spawning R Shiny processes
run_as shiny;

# Define a top-level server which will listen on a port
server {
  # Instruct this server to listen on port 5000.
  listen 5000;

# Define the location available at the base URL
  location / {

# Run this location in 'site_dir' mode, which hosts the entire directory
# tree at '/srv/shiny-server'
    site_dir /srv/shiny-server;

    # Define where we should put the log files for this location
    log_dir /var/log/shiny-server;

    # Should we list the contents of a (non-Shiny-App) directory when the user
    # visits the corresponding URL?
    directory_index on;
  }
}
```

5. Create shiny-server.sh

```
#!/bin/sh

# Make sure the directory for individual app logs exists
mkdir -p /var/log/shiny-server
chown shiny.shiny /var/log/shiny-server

exec shiny-server >> /var/log/shiny-server.log 2>&1
```

6. Create the docker image and push it to GCP

Open a terminal in "*CloudRunShinyDashboard*" directory. Let's define the following terms.

- {PROJECT_ID} = GCP Project ID (e.g. = shiny-project)
- {REGION} = GCP Project Region (e.g. = us-central1)
- {REPO_NAME} = GCP Artifact Registry Repository Name (e.g. = shiny-repo)
- {IMAGE_NAME} = Container Image Name (e.g. = shiny-image)
- {IMAGE_TAG} = Container Image Tag (e.g. = v0.1, v1.2)

6.1 Create the image

Don't forget the dot `.` at the end of the command.

```
docker build --tag={REGION}-docker.pkg.dev/{PROJECT_ID}/{REPO_NAME}/{IMAGE_NAME}
```

You can test the app by running the image locally, as long as it doesn't require any additional permission from GCP. (e.g.— if you're using "**bigrquery**" package). Make sure to use the same port number (e.g. — 5000) as before.

```
docker run -d -p 5000:5000 {REGION}-docker.pkg.dev/{PROJECT_ID}/{REPO_NAME}/{IMA
```

6.2 Create GCP Artifact Registry repository

```
gcloud beta artifacts repositories create {REPO_NAME} --repository-format=docker
```

6.3 Add permission in order to push the image

```
gcloud auth configure-docker {REGION}-docker.pkg.dev
```

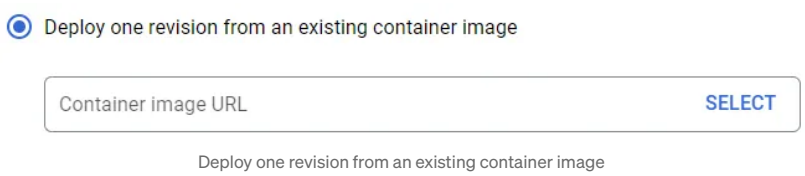
6.4 Push the image to the created repo

```
docker push {REGION}-docker.pkg.dev/{PROJECT_ID}/{REPO_NAME}/{IMAGE_NAME}:{IMAGE
```

7. Deploy the image with Cloud Run

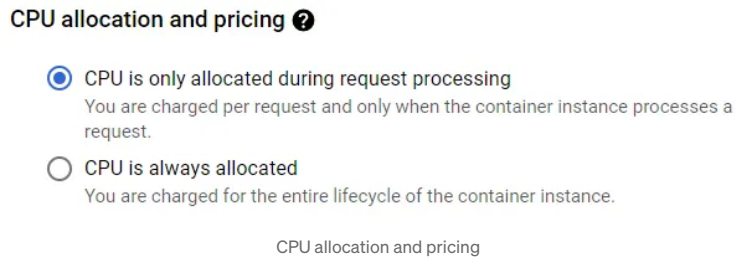
Navigate to Cloud Run on GCP Console. Go to create a new service. Make the following changes.

- Choose "**Deploy one revision from an existing container image**".



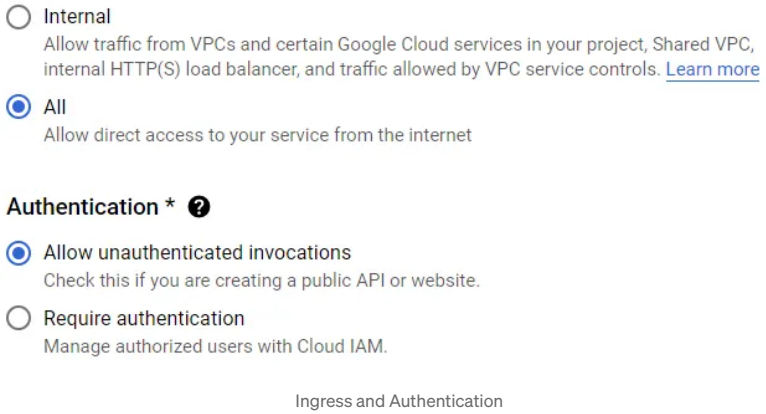
Deploy one revision from an existing container image

- Click "**Select**" in "**Container image URL**" field. Choose the pushed image from "Artifact Registry".
- Service name = something like "**shiny-dashboard-test**". (Make it as unique as possible.)
- Region = {REGION} from **Section 6**.



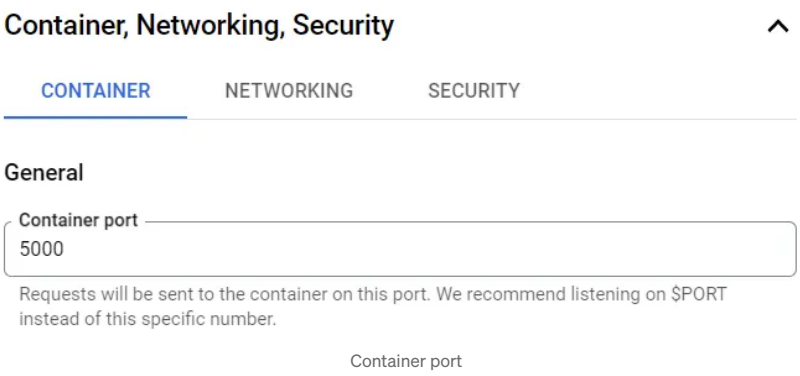
CPU allocation and pricing

- Autoscaling instances
  - Min=0, if you want to shut the container down, when there are no users for some time. Min≥1, if you want to run the container all the time. (Note that, Min ≤ Max.)
  - Max=1, if only a small group of people (5~10) are using the dashboard (e.g. — Management) once in a while. Otherwise, set "Max" accordingly. However, note that "Session affinity" of Cloud Run is still in the preview stage (at the moment of writing). Therefore, things could go haywire, if there are more than 1 instance and "Session affinity" does not work properly.
- Allow all traffic.



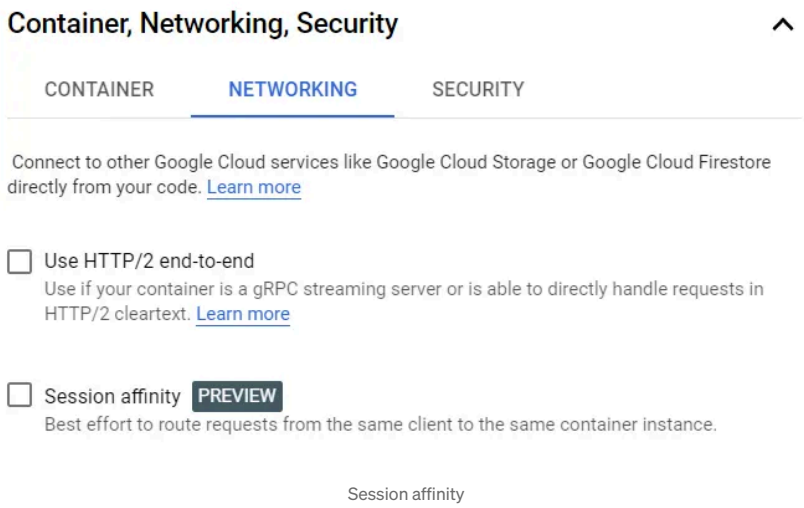
Ingress and Authentication

- Authentication = Allow unauthenticated invocations.
- Container port = 5000



Container port

- Memory = 4GB (Depends on your shiny app)
- CPU = 2 (Depends on your shiny app)
- Maximum requests per container = 80 to 1000 (Depends on the number of users and your shiny app)
- Execution environment = “First generation” is sufficient.
- Session affinity = tick, if max instances > 1.



Session affinity

Click “**Create**”. After a little while, the service “**shiny-dashboard-test**” will finish deploying. You’ll see the URL next to the service name. (something like “*https://{SERVICE_NAME}-{SERVICE_ID}-{REGION_SHORTENED}.a.run.app*”). Now, ***anyone on the internet*** can access your dashboard with this link.

. . .

References

- https://www.youtube.com/watch?v=uu97P0IWsO0
- https://towardsdatascience.com/dockerizing-and-deploying-a-shiny-dashboard-on-google-cloud-a990ceb3c33a

Shiny Google Cloud Run R Docker Websocket

Written by Poorna Chathuranjana

31 Followers · 3 Following

Follow

LinkedIn (https://www.linkedin.com/in/hdpoorna/) || WordPress (https://hdpoorna.wordpress.com/) || YouTube (https://www.youtube.com/@tekcerpts)

Responses (3)

Samwise Gamgee

What are your thoughts?

Huiyan Wan
May 14, 2024

Wow thank you so much you saved me!

1 Reply

Shamildilshan
Feb 5, 2023

Great article ✌

1 Reply

Flavio M
Aug 27, 2024

Thank you for the article.

What is the current state of Session Affinity? Is it properly working by now? If so, how to configure it? This is important otherwise it is not possible to scale the application...

Reply

More from Poorna Chathuranjana