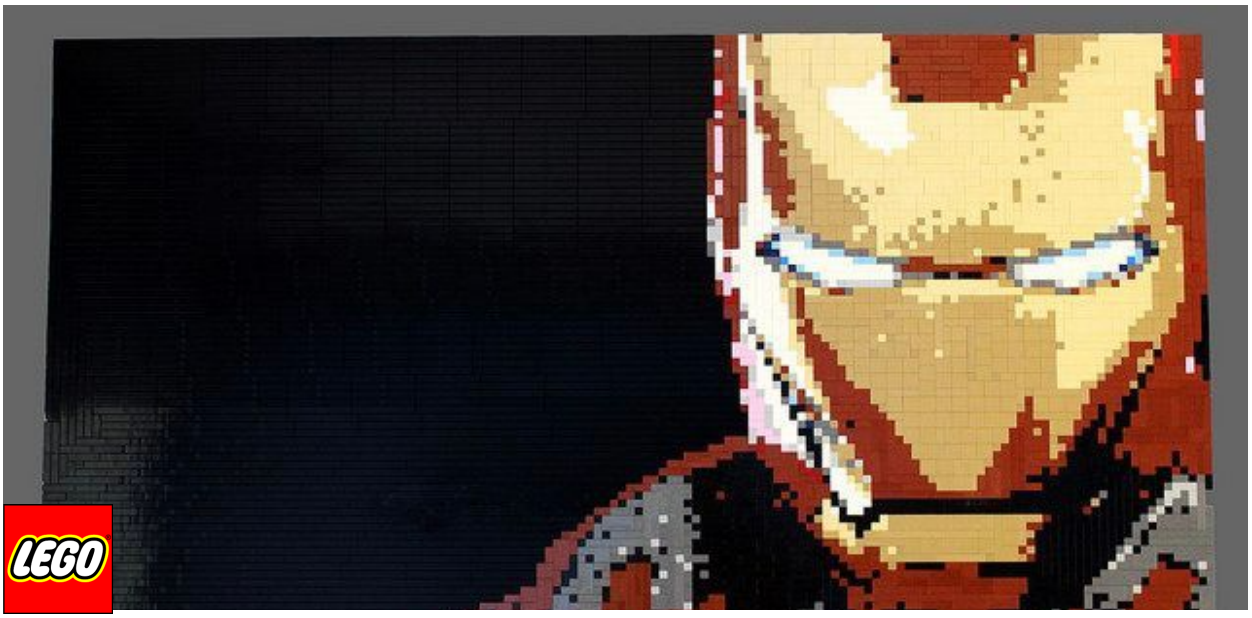


2D LEGO MOSAIC GENERATOR

CSCI 5253 – Datacenter Scale Computing
Project Final Report

FEELGYUN OH

University of Colorado, Boulder

Contents

Project Goals

- Recently, we have been suffering a hard time ever due to COVID-19. This pandemic has been changing our normal life into a shape that we never experienced. Amid this situation, I have no choice but to stay with my son who needs to be taken care of everything all day long. Nowadays he is obsessed with LEGOs and wants to make everything into LEGOs. The problem is that he requests me to make real things into unique LEGOs, but it is time-consuming for me to make a lego every time.
- The main goal of this project is to give a similar LEGO building instruction to users if they upload pictures on a website. An additional goal that I hope to obtain is to make me free from my son's ceaseless requests for making a new lego shape.
- Main goal of this project is to launch a web service that change an image uploaded by users into a 2D mosaic lego., I will focus on making 2D mosaic lego result.
- With this generator, we don't need to abandon lego or sell it out with a big loss. Because this service is able you to have a special lego picture such as your family picture with seemed worthless bricks.

[1] Lego Picture-1

[2] Lego Picture-2

Dataset

- Input data: Mostly this app will get the picture from users on the website.
- Open-source: I will use the open-source R language package named 'brickr'
- Github Link: https://github.com/ryantimpe/brickr [3]
- Format: R language
- Encoding: UTF-8
- Imports Libraries: ggplot2, magrittr, dplyr, tidyr, purrr, scales, farver, colorspace, rgl
- Suggested Libraries: knitr, rmarkdown, gridExtra, png, jpeg, tibble, raster, stringr
- Enhancing a real world building experience with mosaics, generated instructions, and piece counts.
    - Lego color dataset: LEGO Module Colour Palette 2016. [4]
    - 'brickr' also includes tools help users create the 2D Mosaics and 3D model output using real LEGO elements.

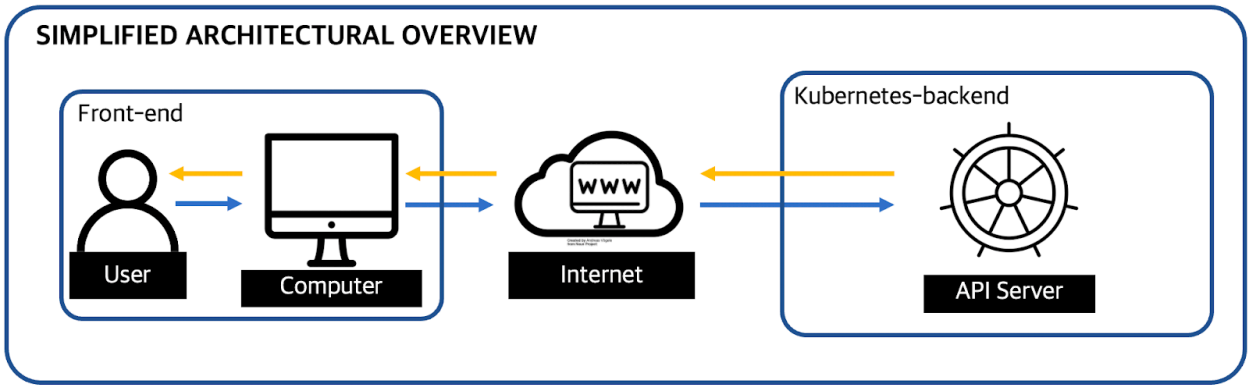
Software and Hardware Components

- **R**
    - Main packages: Shiny, Brickr
    - Rstudio

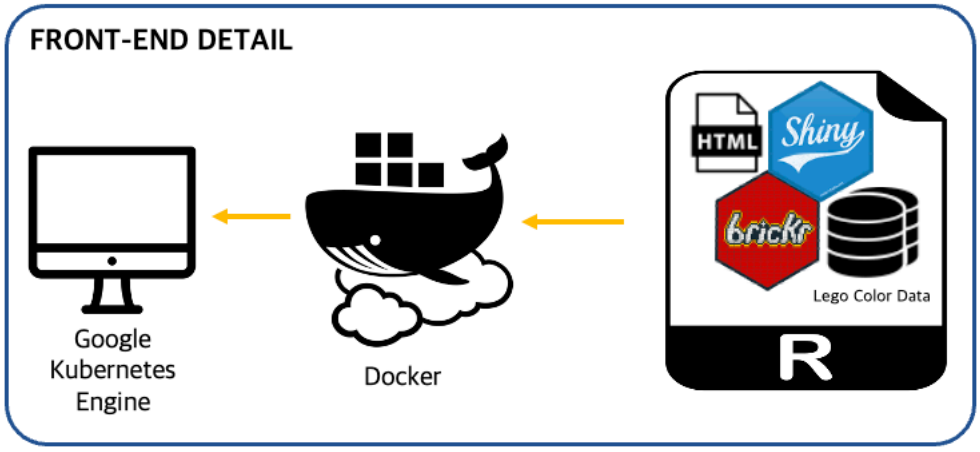
- **Google Cloud Platform**
    - Cloud Shell
    - Docker
    - Storage Bucket
    - Google Kubernetes Engine: Deployment, Service, Ingress
    - APIs: Compute Engine, Cloud Logging, Cloud Deployment Manager V2, Cloud Pub/Sub

- **Sort by Rubric**
    - RPC / API interfaces: Compute Engine, Cloud Logging, Cloud Deployment Manager V2, Cloud Pub/Sub
    - Databases: Lego color dataset
    - Key-Value Stores: bs4Dash, shinyjs packages for frontend configuration
    - Virtual Machines, containers or "functions as a service": Kubernetes
    - Storage services (s3, object storage, etc): Google Storage Bucket for logging

Architectural Diagram



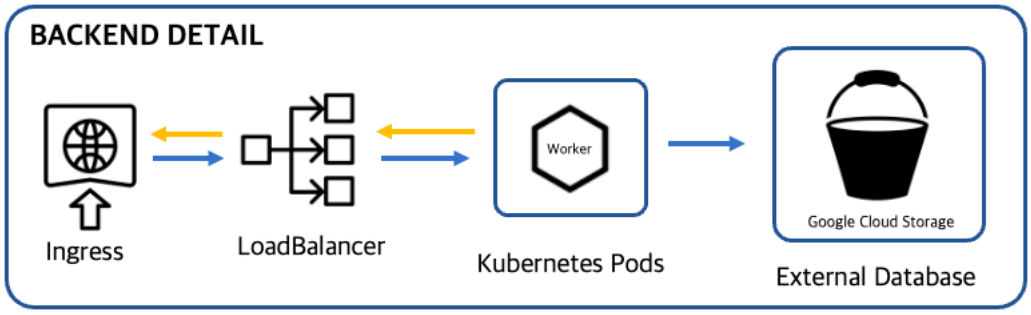
- Users upload png or jpeg type of pictures through the front-end webpage made by R with shiny package.
- The internet service passes the request from the users to the backend server which is consisted of Kubernetes pods
- At the backend side, worker pod that contains R and open-source software package 'brickr' that changes uploaded image to 2D mosaic. The orchestration, Kubernetes controls the deployment, service and ingress pod. Also, Kubernetes save the logs into google storage bucket.
- The work pod passes the 2D mosaic lego result to users via frontend.
- Moreover, users are able to customize the result with navigation bars in app.



- Shiny package is an R package that make it easy to build interactive web app directly from theR language. With the shiny based app, you can host standalone app on a webpage. Shiny based apps are consisted of two big part. One is ui part and another is

server part. UI part nested R functions that assemble an HTML user interface for the app. Server part is a function with instructions on how to build and rebuild the R objects displayed in the UI part. Lastly, shinyapp combines ui.R and server.R files into a functioning app. Wrap with runApp() method if calling from a sourced script or inside a function.

- I used bs4Dash package to configure the overall shape of the web site user interface. Bs4dash package is a shiny dashboard using AdminLTE3 that is a fully responsive administration template and based on Bootstrap 4.5 framework and JS/jQuery plugin.
- I used rocker/shiny-verse docker image as a base image and install necessary several packages to run my app. After the docker file built, I uploaded it in the container registry at the google cloud platform.
- I make the frontend webpage as simple as possible to be looked good.



- The back-end side is the core of the app. This is an invisible but very much important part of whole part.
- The open-source brickr package is used for this part. Brickr helps users create the 2D Mosaics output using real LEGO elements. Let's look into by step by step, first the worker needs to import the photo that user uploaded. The result of this function will be a data frame with one row for each pixel, with the X, Y, and red, green, & blue channel values. It then rescales the X and Y values to be within the uploaded image size, averaging the R, G, and B values. If the image is not square, the function resizes the image to a square in the center of the image. Next step is converting the uploaded image to LEGO standard colors, by pixel by pixel, in accordance with lego color dataset. Then lastly, worker count the collected lego pieces from previous step and display the result to the user to frontend.
- After all frontend and backend parts are prepared, I made a docker image as specified in front-end detail section.
- By using an orchestration tool, Kubernetes, I made a worker deployment pod and a service and an ingress for the deployment to service on the web.

- Lastly, I made the Kubernetes to save system log into an independent google storage bucket to check the log whenever I want to check.

Debugging and Testing

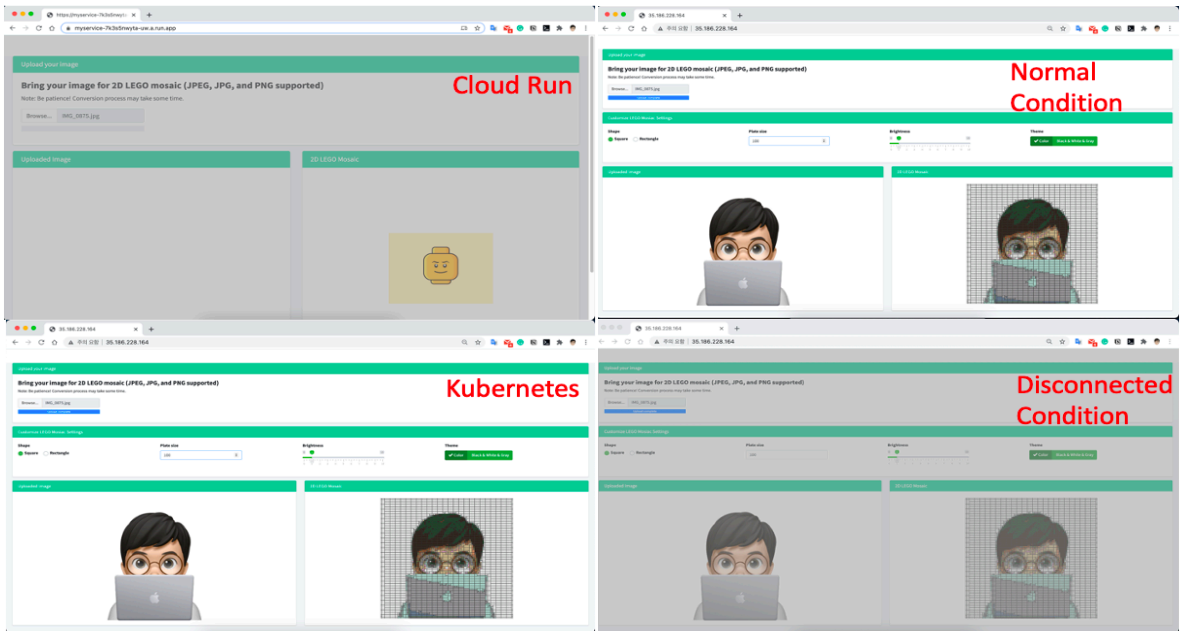
Solved

- Not launching the web app due to the wrong docker file command line.

```
# Failed Code.
CMD ["R", "-e", "shiny::runApp('dcs-pjt-lego')"]

# Succeed Code.
CMD ["R", "-e", "shiny::runApp('dcs-pjt-lego', host='0.0.0.0', port=3838, launch.browser = FALSE)"]
```

- Service at Google Cloud Run
    - The first time, I want to service my app as Continuous Integration and Continuous Deployment (CI/CD) and make it possible to be controlled at github. Then If It is succeeded, if I make a git push command from github then the google cloud run get the changing git and deploy a new service seamlessly. So, I tried with the exact same docker file I used for Kubernetes, but it failed. Because web-socket currently not supported on Cloud Run fully managed.



- So, I change the platform to Google Kubernetes Engine.

Continue

- App keeps disconnecting after a service. I think this problem has been related to the web-socket problem as specified above.

- Followings are suggested debugging methods from shiny and users.
  - Forgetting to close each database connection after loading in data
  - Making multiple long-running calls to a public API
  - Sub-optimal application instance or worker provisioning
  - Moving the code that uses Rjava outside of the shiny apps and into an executable set of R scripts and running those scripts with a system function call.

Evaluation and Conclusion

- I have learned how to utilize Kubernetes on GCP using pods, services and ingress. Before I start out this project, I've never done a web app service. But through the process, I am able to get a taste of how to configure the frontend side with shiny using various packages to decorate UI more looking good. This is a precious and huge gain in my career.
- Frankly speaking, when I plan this project, I really want to more focus on the web app service and related services. For example, I want to open this app only to the member who makes an ID and logged in. However, after I start out, I faced a lot of unexpected problems at each step. So, the final result, the 2d lego mosaic generator, is quite impressive and a great masterpiece itself, but that is not fully fulfill my first plan.
- However, I think this is the most important thing, I really enjoy this project even though I couldn't sleep before 3 AM from November, and I am happy that I succeed to make this app by using every skill what I learn from this course and what I study while I solve every problem I faced. And when I debug and test it, my son comes to me and takes a glance at the 2d lego mosaic and he says, 'Wow this is cool!!'. His small piece of reaction melts down every stresses right away. Thank you.

References

- [1] Lego Picture-1, https://seankenney.com/portfolio.php/family-portrait-2/
- [2] Lego Picture-2, https://seankenney.com/portfolio.php/family-portrait/
- [3] Open-source, brickr, https://github.com/ryantimpe/brickr
- [4] Lego color dataset, http://www.bartneck.de/wp-content/uploads/2016/09/2016-LEGO-color-palette.pdf