Gerhard van Andel
CSCI 5273 Network systems

gMonitor

Spring 2020

Overview

Just another service level indicator (SLI), an SLI is a carefully defined quantitative measure of some aspect of the level of service that provides an idea on the health of the service. With this a SLI can be as simple as a web page load time. If a website does not load within three seconds the chance that the user gives up increases by 32% and goes to a different page. So monitoring traffic and response time from servers can be an important data point for web experiences across all industry sectors.

gMonitor will take a request for an endpoint that will send a list of urls to search. The service will then create a task for each url and using a pub sub model the web_bots will pull the web sites for this data and store it into google cloud storage.

Redis - used as a datastore for search data, and a lock manager

RabbitMQ - give tasks (web-site urls) to then read and parse

controller - web service to request website crawling to and search requests

spider - takes a task off of RabbitMQ (web-site urls) to get

scanner - takes a task off of RabbitMQ (web-site urls) to get

cleaner - takes the entries out of the cache and uploads them to google cloud storage

Goals

1. Monitor http get load times between web domains
2. Use google cloud platform to host in different regions around the united states to be able to test connections speeds on the same domain and endpoints from different regions to simulate east coast and west coast traffic.
3. Test a variety of industry web sites from sports to news to popular sites.
4. Pool the data to analyze the results of accessing websites from different regions.

Milestones

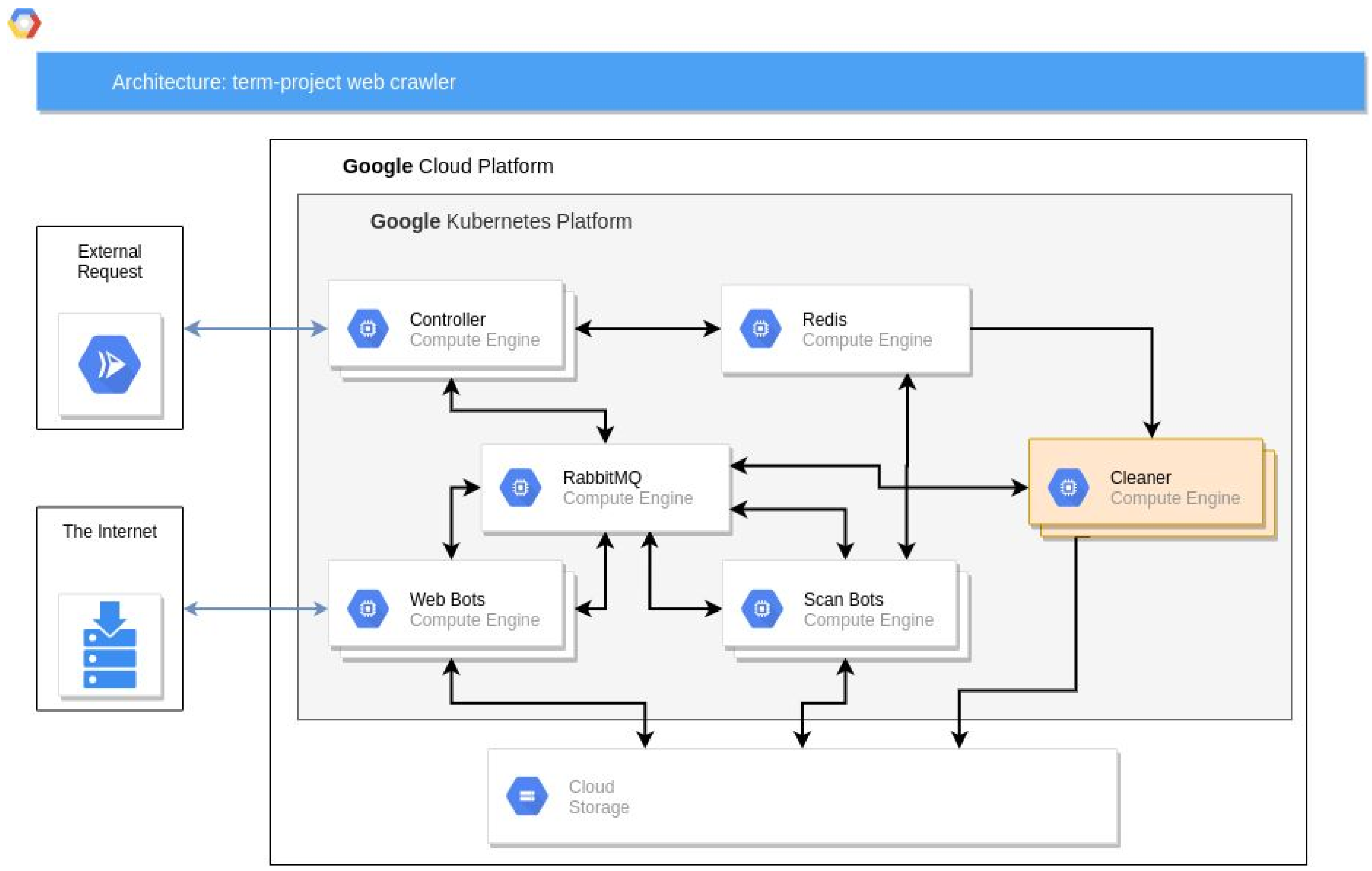
Checkpoint 1:

Have a working prototype using both icmp ping based metrics and tcp http/https based metrics. I will use python requests to pull data from a website then using beautiful soup to parse the web site, pull all the links from the site and then follow the links to determine site speed on leaf sites. Using redis to lock on the domain so workers can follow robots.txt and then store this data into google cloud storage. To test this ecosystem I used docker-compose this allowed me to make changes locally. This increased my productivity a lot as I no longer had to push the changes to the docker registry. I am still working on how to do the correlation across requests. To compare load times of the web sites.

Checkpoint 2:

I simplified the correlation and the task output to reduce space on the database. The redis database works well but I can fill it very fast as the crawler grows exponentially. To reduce the number of tasks in the database by pushing completed jobs back into cloud storage and on request load it out of cloud storage. I didn't think that link information would take up as much space as I thought but I crawled 70 web pages and that linked to more than 1200 other pages so I may have to think how to store more information elsewhere. Have two region instances working to poll web traffic from more than one location. Then figure a way to collectively pool the data sources. I would like to keep the data locality, something that I don't need to share between regions. Still testing using docker compose locally. Thinking of a simple way I can visualize the web that the system crawls and time it takes to crawl.

Architectural Diagram


Architecture: term-project web crawler
Google Cloud Platform
Google Kubernetes Platform
External Request
The Internet
Controller Compute Engine
Redis Compute Engine
RabbitMQ Compute Engine
Cleaner Compute Engine
Web Bots Compute Engine
Scan Bots Compute Engine
Cloud Storage