Heimdall Spring Presentation

Kyle Cullion
Michael Keenan
Vivek Kunapareddy
Zack Steck

Advisor: Dr. Badri Vellambi

Goals

- Create a free, open-source server monitoring application
- Fill a niche in the server monitoring industry for this open-source solution
  - Current solutions are expensive to maintain
- Idea came from a co-op experience in 2018
  - Company needed a server monitoring application for their project
  - Underwhelmed by current solutions

Intellectual Merits

- Open-source is an industry term that has existed since the late 1990s
  - Created some of the world's most popular software
- Based on the novel idea of anyone in the world contributing to the project
  - Allows for some incredible collaboration towards a common goal
  - Worked quite well in the past 20 years
- Making Heimdall open-source allows for an original contribution to the application monitoring industry, and is therefore a worthwhile venture

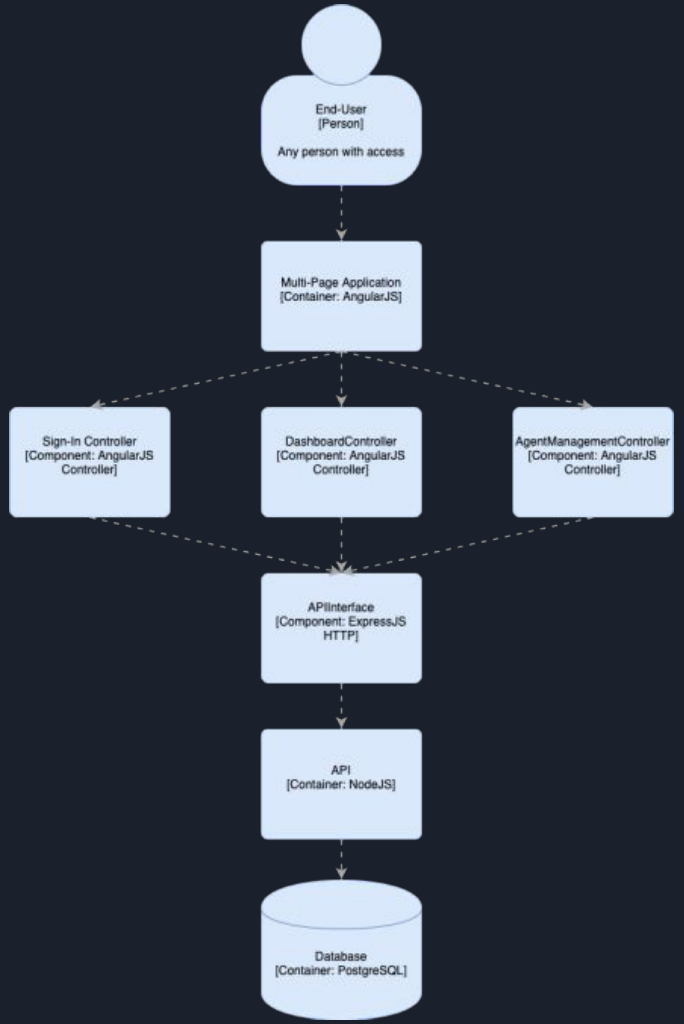
Broader Impacts

- Technology isn’t perfect
- As it reaches further and further across different industries, the potential impact of a system failure has risen exponentially
    - Server crashes, power outages, unforeseen bugs in production code
- Using Heimdall allows companies to know immediately when a system goes down
- Real-time metrics allow insight into expected and unexpected server behavior on different fronts for clients to further analyze for business purposes.

Design Specifications

- Heimdall's design consists of an agent-server relationship.
- The agent, written in the Go, is installed on the end user's desired server and proceeds to collect low-level server metrics to move via API to Heimdall's central server.
- This external facing API will securely deliver data requests and interact with the database. This backend database will be using PostgreSQL to store information sent via the agents to be fetched by the central server.
- The central server will be tasked with serving various files and assets to the web-based application to display information to end users seeking their usage levels for each metric.
- The API and central server architecture will be built using Node.JS, whereas the web-based application will be built using AngularJS.
- The end user interacts with the web application interface to obtain their sought after metrics

Design Diagram



Technologies

- The entire HTTP request architecture is serviced by servers running Node.JS
  - The front-end is served by a Express.js framework
  - The back-end HTTP routes are also written in Nodejs
- The front-end of the application was written in AngularJS
  - The front-end functions as a multi-page application, with the application view being rendered from the server on each switch.
- The agents which monitors the server were written in Golang
  - The agents run OS specific commands to get the metrics out and then send HTTP requests to the back-end

Results

- Completed functional server agent that is sending the specified metrics via HTTP requests
- Web-app is functional with core metrics visualizations and agent management
- Multiple platform compatibility for the agent software
- Ability to switch between multiple registered agents along with selecting specific time frames and network adaptors to view respective metrics.

Demo