1. Introduction

1.1 Purpose

The purpose of Augur is to aggregate open source data and metrics in order to make sense of data with human centered data science strategies.

1.2 Scope

Augur is a Flask web app, python library, and REST server that uses the CHAOSS project and open source projects to collect software metrics. The goal is to allow users the ability to navigate complex unknowns in order to enable comparisons. All data will be available to the user and downloadable, as well as having unique visualizations. From the start, time has been a fundamental dimension in all metrics.

1.3 Assumptions and Dependencies

Augur will be running on a Unix system or Ubuntu 18.04 Virtual Machine.

The Augur database will be run with PostgreSQL 10 or higher, with read/write access.

A GitHub access token will need to be given in order to collect data.

Python 3.6 or higher must be installed to run the API.

In order to run the front end locally, a user will need Vue.js, vue-cli, node, and npm installed.

2. Software Product Overview

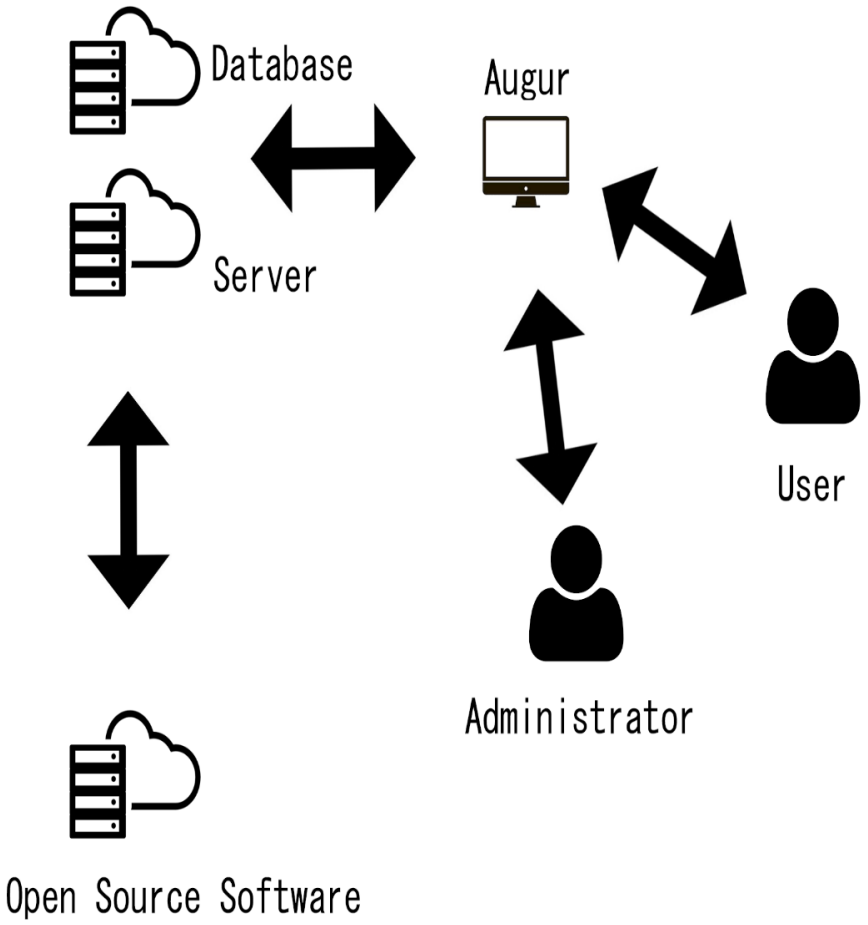
2.1 System Scope

Augur is an open source project and anyone is encouraged to contribute. Augur can be run on a virtual machine or server running Ubuntu 18.04 or higher. There is no hardware specific to Augur. Augur uses many technologies to create a seamless web app and REST API, including:

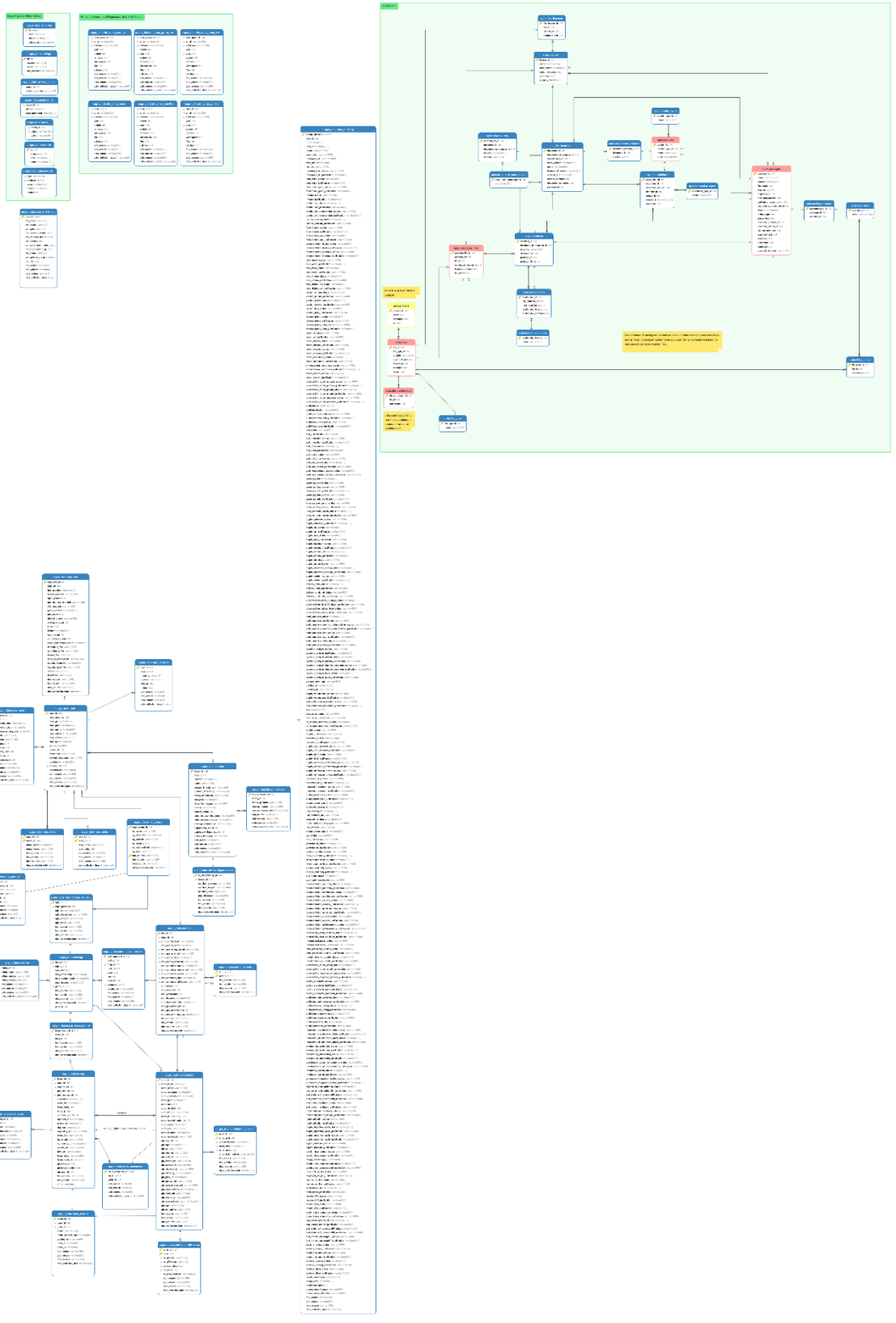
- Python
- Pandas
- Flask
- Requests
- MySQL
- Gunicorn
- Docker
- Pytest
- Tox
- Sphinx
- Apidocsjs
- Vue

2.2 System Architecture

2.2.1 External View:



2.2.2 Internal View:



2.3 Feature Overview

Batch Endpoints –

Returns batch requests results in POST JSON of API requests.

Evolution Endpoints –

Returns aggregation of Repo data and Repo group data including issues, contributors, reviews, and code.

Experimental Endpoints –

New and less used endpoints that return experimental features in data aggregation including issues, contributors, reviews, and code.

Risk Endpoints –

Returns data relating to a tools risk association, such as licensing, time resolution, forks, and individuals involved.

Utility Endpoints –

Returns data valuable to using Augur as well as developing.

Value Endpoints –

Returns data important in determining the popularity of a tool, such as stars and watchers.

3. System Use

Augur provides many ways to interact with its systems.

To control the backend, add repos and repo groups to your database for collection, or perform a variety of utilites, users have access to a variety of built in Augur commands.

To install the project, start the frontend & backend servers simultaneously, build the docs or rebuild the binaries, users have access to a variety of built in make commands.

3.1 Actor Survey

Project Lead

A project lead is defined as the manager/owner of a open source tool. A project lead can find many statistics using Augur that can be helpful, such as checking the current usages and fame associated with their project, as well as managing open issues and assigning developers accordingly.

Project Developer

A project developer is defined as any developer that is making contributions to an open source tool. A developer of an open source tool may use Augur for a variety of its tools, including seeing issues with their project that need immediate attention, checking their contributions to a project, or seeing the number of pull requests a day in order to measure work output.

Business Team

A business team is defined as a group of individuals in charge of marketing, risk management, or implementation of an open source tool. Augur is incredibly helpful for this type of team as the aggregation of data includes risk licensing, statistics, and usages for the tools being used. A

business team could find easily what type of people are watching a tool and who their target audience may be.

Project Users

Project users are defined as any individual that uses an open source tool. Project users can find value in Augur by having the ability to see the work that is being done on an open source tool at any given time, see the popularity of the project, and upcoming features/requests.

4. Specific Requirements

4.1 System Use-Cases

Use-case name	UC1 Manager Overview of Project
System	AUGUR
Actors	Project Manager
Brief Description	This use case explains how the manager of an open source tool manages developers through the evolution endpoints.
Basic flow of events	Flow begins when user adds their repo to the Augur system. 1. Manager opens a web browser and navigates to their instance of Augur. 2. Manager navigates to groups 3. Manager selects a group 4. Manager is able to see different insights such as issues and aggregation of repository information 5. Manager makes decisions and delegates issues to team

Use-case name	UC2 Business Utilization
System	AUGUR
Actors	Business Team
Brief Description	This use case explains how an open source tools business can use augur through risk and value endpoints.
Basic flow of events	Flow begins with repo added to the Augur system. 1. Team member opens a web browser and navigates to their instance of Augur. 2. Team member navigates to groups 3. Team member selects a group 4. Team member is able to see aggregation of Risk information or Value information for their tool 5. Team member creates a data model or document displaying/interpreting the aggregated data.

4.2 System Functional Specifications

- User can create an instance of Augur and run collectors to aggregate data.

- User can navigate to hosted version of Augur with tool repos added.

- User is presented with Insights from across the tool repository.

- User can navigate to Repo section and see a list of all added repositories as well as their URLs, names, issues, and commits.

- User can navigate to Group section to see a list of stored Repo Groups as well as their names, descriptions, website, last modified date, and type.

4.3 Non-functional Requirements

- **Enable Comparisons**

  Allow users to see how their project compares with other projects they are familiar with.

- **Time as a Fundamental Dimension**

  Projects data should focus on time statistics as point in time scores are useful and can be made more useful if they are compared historically to allow prediction of future times.

- Downloadable Data

All data should be downloadable as a .csv file or some other data exchange format. Users can trust metrics with underlying data, which requires transparency.

- Downloadable Visualizations

All data should be downloadable as .svg files. Users should be able to use the data in reports and in ways that are easily explainable.

5. Design Constraints (5)

Hardware constraints

Augur is available for Linux systems and can be run on a Virtual Machine running Ubuntu 18.04 or higher.

Software constraints

Augur is to run as a web application through Vue.js.

Augur is to run with a PostegreSQL 10 database or higher.

Augur requires a GitHub key to collect data.

Augur runs on all major web browsers.

Development constraints

Augur follows a traditional Git workflow of:

Forking

Making a change

Create a pull request

Merging

Updating the fork

Etc.

Augur requires all commits to be signed off with a Developer Certificate of Origin in accordance with the CHAOSS Project Charter section 8.2.1.

Open Source Constraints

Augur is available for anyone to build upon and is open source to be used.

Augur will only aggregate data from open source tools using GitHub keys and collections.

Documentation Constraints

Augur API documentation is written inline using apiDoc. Each public API route is decorated with a documentation block.

Augur maintains 2 distinct sets of documentation for each release. The first is the API documentation located on each hosted instance, and the second is the library and usage documentation.

Augur will have its own API docs available under <host>/api_docs/ for each running instance.

6. Purchased Components

Augur needs to be hosted on a Ubuntu 18.04 or higher capable machine in order to be used effectively. Purchase or acquisition of a server that meets this requirement is needed for Augur to fit industry level purposes. An example of this being an Amazon EC2 Linux server.

7. Interfaces

7.1 User Interfaces

Augur uses Vue.js to display the data aggregation visualizations it creates and controls for this in a Vue web application. This interface does not require authentication to use and is available to anyone with access to the hosted instance. The visualizations are created with Vega-Lite for the metrics exposed by the backend.

7.2 Software Interfaces

Augur has a REST backend application written in Flask and Python. This backend has Workers which are responsible for collecting data and inserting it into our unified data model, Broker which is responsible for distributing data collection tasks to the correct workers, Housekeeper which is responsible for keeping the data up to date, a main class which is in charge of caching registering plugins, and reading the configuration file. It also has a WSGI server built in Flask which exposes the REST backend. This backend can be accessed through any instance of Augur through <host>/api_docs/.