GROOT, DESIGNED TO BE REPLACED

Naren Sivagnanadasan[2]
University of Illinois at Urbana-Champaign
sivagna2@illinois.edu

Sameet Sapra[1,3]
University of Illinois at Urbana-Champaign
ssapra2@illinois.edu

Benjamin Congdon[1,3]
University of Illinois at Urbana-Champaign
bcongdo2@illinois.edu

Aashish Kapur[2]
University of Illinois at Urbana-Champaign
askapur2@illinois.edu

Tyer Kim[1,3]
University of Illinois at Urbana-Champaign
tkim139@illinois.edu

ABSTRACT

We introduce Groot, a microservice architecture designed to be sustained through many generations of developers. Groot is structured so that any project or service can be easily replaced or rebuilt, improving and prolonging the utility and lifespan of the entire system without the affecting the functionality.

1 INTRODUCTION

As with most other student chapters of Association for Computing Machinery (ACM) and many companies, the student chapter of ACM at the University of Illinois at Urbana-Champaign (ACM@UIUC) faces many challenges as a result of fast-changing technology. Just in the past four years, Rails has been supplanted by Node.js and Angular/React, and PHP has gone from a “goto” language to an outdated language with only a few purposes. This poses a couple issues to student chapters when considering choices for technology stacks in the context of maintainability and future-proofing. Pragmatically, original authors of a project for a student chapter will not be in school for more than a few years to maintain the project, and the newcomers will have very little ideas of how to sustain the technology stack of the project created just a couple years ago.

2 SUSTAINABLE PROJECTS

2.1 Problem Description

We now discuss the issue our developers faced when surveying the collection of projects that have been maintained for the past few years.

The last of the original developers have graduated and the current team has no idea how to support our aging systems. However, these systems span a lot of the core services ACM@UIUC has to offer, including cluster access, member database management, recruiter services, event advertising, group management as well as more esoteric projects like our handmade internet-connected light system, office music manager, and our web-connected soda machine. As was the paradigm when these systems were made, all these separate projects were placed into one master web application with tight interdependencies between models in the system. This application architecture is referred to as the monolithic application architecture and is featured in frameworks like Rails and Django. However, this requires that any time a contributor wanted to make something new, they had two options: a) wade into the complex network of connections and code of this master application, or b) re-implement most of the system in an unconnected application to access to the functionality they need.

This complexity coupled with the high churn rate in maintainers made it challenging to maintain such a project and it meant development efforts were highly duplicative.

2.2 Micro-Service Architecture

January of 2017 marks the release of ACM@UIUC’s brand new infrastructure Groot. Groot directly addresses these issues by adopting a micro-service architecture. The system breaks down into three groups: clients, services and the gateway. All components of the infrastructure are completely separate applications running on separate

[1] Department of Computer Science
[2] Department of Electrical and Computer Engineering
[3] National Center for Supercomputing Applications

processes in deployment. Clients consist of mobile applications and web applications (ACM@UIUC currently has two clients: one for the web and another to manage a display in the office). Clients communicate with services, each of which is responsible for a different aspect of the ACM@UIUC infrastructure (events, members, sessions, etc.), via what is called an API Gateway. This gateway exposes all the services as a unified REST API, while managing application level security, inter-service communication, and routing of requests. This is a paradigm that has been becoming more common in industry; companies like Netflix have been championing the benefits of decomposing functionality into applications instead of components of a single application. For ACM@UIUC, this approach provides several key benefits:

1. Services are inherently simple. Application scale ranges from a standard 1000-line Sinatra app with multiple database models to a 100-line Node app which simply reads and writes to a YAML file.
2. Any service can be discarded and replaced without affecting other services (provided functionality parity).
3. Third party APIs are easily and securely accessible by any client that can access the API Gateway. This moves the burden off the client developer to juggle tokens.
4. Clients are far simpler because the application logic has been abstracted behind the gateway.
5. Maintainers can focus and own their own codebase.
6. Due to the nature of the API Gateway and declaration of routes, APIs are intrinsically documented.
7. The entire system is language agnostic. Services and clients (and even the gateway) can be written in any language or framework.

2.3 API Gateway and Arbor

The key to the entire micro-service architecture is the API gateway. There are many extant platforms which provide the functionality of an API gateway; Amazon, Nginx, and others have offerings that manage access to services and coordinate micro-service systems. However, these are high performance, complex systems that are meant to serve millions of users at once, which was considered a bit heavy handed for our purposes. Additionally, the use of these systems comes at the detriment of ease-of-use and clarity (and perhaps cost).

Thus, in order to support Groot's architecture, we introduce a novel framework for managing lightweight micro-service systems called Arbor. Arbor is a simple, statically configured API gateway written in Go. It provides service registration, a proxy, a security layer for client authorization, verification and simple request sanitization. Additionally, it has a verbose documentation-like schema format for describing service routes. This means that registering a service is just adding a file, recompiling and deploying, and authorizing a client is a just simple command. The question of what data is available is answered by the code itself. Arbor is simple, makes very few assumptions, and is easily extensible.

3 RESULTS and EVALUATION

The transition to this new infrastructure has opened up new opportunities to our members. Now, it is easy to access all the information and services we maintain. This has catalyzed a new wave of projects coming down the pipeline. Since Groot's deployment, the ACM@UIUC has deployed a quotes service, a meme service (like Google's memegen), a display client, a new web-connected soda machine, new swarm lights, a new music control system and Amazon Alexa integration for everything in the office.

3.1 Current Issues

Throughout Groot's development, we have uncovered a few issues which should be identified and addressed:

1. Some API endpoints need to be public as other services need this information without the valid credentials needed). Therefore, endpoints need to be designed with different access so that data is both protected yet accessible by the right services. While service oriented architecture itself allows both data and logic to be compartmentalized and separate from other services, some data requires the use of other services.
2. Due to the formulaic nature of adding API endpoints, there is some duplicity when adding a new page on a client that requires the same route to be added both in the API Gateway and the services. This results in many interdependent changes to multiple different code repositories, which leads to increased complexity when developing and requires our application deployment to be robust.

4 CONCLUSION

In summary, we have built a service-oriented system to serve and maintain the overall infrastructure of ACM@UIUC which can persist across generations of developers. Any project or service can be easily maintained, replaced, or discarded, even following the previously-traumatic eventuality of the graduation of the project's original author. This allows us to update our services to run on appropriate technology-stack of the time, and to easily onboard new contributors into the Groot ecosystem.

If you want to learn more about the new infrastructure, the source code is released under the NCSA/University of Illinois Open Source License and is available at https://github.com/acm-uiuc. You can also check out our new site at https://acm.illinois.edu

ACKNOWLEDGMENTS

Project Groot is supported by Amazon.com Inc. and Palantir Technologies. ACM@UIUC is completely funded by industry sponsorships, and we are grateful to our corporate sponsors for enabling us to solve challenging problems.