Groot: Designing to be Replaced

Featuring Arbor: A Novel Highly Configurable API Gateway Framework to Facilitate Lightweight Microservices

Naren Sivagnanadasan, Sameet Sapra, Benjamin Congdon, Aashish Kapur, Taeyoung Kim

ACM@UIUC

Abstract

We introduce a novel framework for creating unified application/programmer interface (API) layers for the microservice architecture. We show that with this new framework, which called Arbor, we can allow a whole new class of developers, in particular, high churn development groups like college organizations, to create infrastructure in the microservice paradigm allowing them to continually update their software infrastructure, without significant additional documentation. We show the potential of this framework by introducing ACM@UIUC's new infrastructure, referred to as Groot.

Introduction

The impetus to create such a framework stems from the reality that the majority of developers who work in our development group, will not be there in 4 years. We also see new developers walk in oblivious of the frameworks we have used in the past. 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past four years, Rails has been supplanted by Node.js and Angular/React, and PHP has gone from a "goto" language to an outdated language with only a few purposes.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 these high churn development groups when considering choices for technology stacks in the context of maintainability and future-proofing. Original authors of a project for a student club, for example, will not be in school for more than a few years to maintain the project, and the newcomers will not know how to sustain the technology stack of the project created just a couple years ago.

Motivation

Monolithic architectures (single app infrastructures) have their benefits (clear model ownership, a write once use everywhere mentality) but the incurred complexity coupled with the high churn rate in maintainers in some development groups makes it challenging to maintain such a project and it means development efforts are highly duplicative.

Microservices

Our response to this problem was to adopt the aforementioned microservice architecture for our next generation infrastructure, which looks to promote modularity and minimization of external dependencies. The main proponents of this architecture include large scale companies like Netflix and many startups. The system breaks down into two groups: clients and services. All components of the infrastructure are completely separate applications running on separate processes in deployment. To prevent client (and service) developers to manage an ever-increasing list of services and their APIs, many implementations of the microservice architectures include what is referred to as an API Gateway.

This gateway exposes all the services as a unified RESTful API while managing application level security, inter-service communication, and routing of requests. 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.

API Gateway

The key to the entire micro-service architecture is the API gateway. There are many platforms which provide the functionality of an API gateway; Amazon Web Services, Nginx, and others have offerings that manage access to services and coordinate micro-service systems. There also exists a few frameworks to self-host an API Gateway. Two popular cross-platform options are Tyk and Kong, however, there are others that are language or web framework specific.

However, all these systems (Kong, Tyk, AWS, etc.) are high performance, complex pieces of software that are meant to serve millions of users at once, which is a bit heavy handed for nonindustrial use. Additionally, the use of these systems comes at the detriment of ease-of-use and clarity (and perhaps cost).

Arbor

We were not happy with the current offerings for API Gateways as many were just too complex to maintain in our organization (again churn being a big factor here). We, therefore, created our own API Gateway framework which we call Arbor.

Arbor is a statically configured API Gateway framework written in Go. It provides the key gateway services, it currently has a full featured REST proxy all behind its hostname, with websockets and more modern protocols in the pipeline, it manages application level security and simple request sanitization, it also features a verbose documentation like service API schema format.

This means that registering a service is just adding a list of routes, recompiling and deploying, and authorizing a client is a just requires a token. The question of what data is available is answered by the schema itself, so the infrastructure is self-documenting. Arbor is simple, and makes very few assumptions, and is easily extensible.

Results and Evaluation

This new framework and architecture choice has accelerated the creation of new pieces infrastructure in our organization many times over. It has given us the freedom to replace many aging systems with new modern ones without fear of breaking other things. It also gives the developers the leeway to focus more on functionality over sustainability, since it is now expected their code will last perhaps two or three years max. Each service can now range from 100 lines of code to 1000. Our infrastructure is now language agnostic as well. We have ruby, python and javascript developers all developing code for the same system. It is also easier now to replace a system since a spec of what needs to be supported is kept in the schema file for that service. We now have 15 services and 2 clients all managed through Arbor serving 1000s of people.

Future Work

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

1. Arbor currently only supports RESTful APIs, we are looking to add support for websockets so that the final parts of our infrastructure can come under the gateway umbrella.
2. Documentation generation from parsing schema files would be a welcomed feature so unaffiliated consumers of our API will have more documentation past the arbor required documentation.
3. Improved centralized functionality for things like security is being considered. Currently there is a simple sanitizer in the proxy pipeline that can be enabled. We have this so that service developers do not need to consider security other than that which is specific to their service. This sanitizer is not very powerful at this point and we must also balance the urge to centralize functionality with not making assumptions about development processes.

Conclusion

We have presented our new framework for API Gateway definition. Though the target for Arbor is not to be concurrently serving millions, we think that this framework will let a whole new set of developers experiment with microservices. It looks to provide the barebones API Gateway functionality and let the developer focus on the application functionality. Ultimately for us it means we no longer have to race against time to train developers before the lead maintainer leaves and that the infrastructure of our organization can now better reflect the needs of its members. We are now more open to one-off contributors, who wish to add one service and leave. When the next web framework of choice emerges, we will have a system ready to accept it with minimal wrestling. The source code is released under the University of Illinois/NCSA Open Source License and is available at https://github.com/acm-uiuc/arbor. The results of the architecture change can be seen at our new site at https://acm.illinois.edu

Sponsored by:

Palantir amazon