RailwayCabal

Project Proposal

Team 12

Alex Bohlken
Abraham Dick
Noah Kenn
Qualen Pollard
Ryan Rodriguez

PROJECT DETAILS

Synopsis

Microservice based web-server framework designed to be scalable, secure, extensible and easily deployable.

Description

Advancements in container-based technologies, such as Docker, combined with the rise of cheap and accessible cloud-based services, such as Amazon Web Services and Microsoft Azure, has led the ability for developers to easily deploy web-based content distribution services across many server instances. This is in contrast to traditional web-based technologies that have relied on concurrency patterns for increasing performance. These older technologies rely on a monolithic based architecture, leading to large structures that make refactoring or modifying features difficult. Microservice architecture attempts to overcome this by segmenting each responsibility of a program into its own service and setting up a communication mechanism between them. The result is a composite structure that can vary in language and even platform between different responsibilities. Discrete services allow systems to easily grow and be pruned as products and business services change and grow without large refactoring efforts.

Unfortunately, long term scalability concerns are often beat out by costlier upfront development and the relative simplicity and availability of monolithic frameworks such as ExpressJS, Rails, and Django. RailwayCabal will provide the same ease of access as these frameworks while providing all benefits of the underlying microservice based architecture.

PROJECT DESIGN

Railway & Cabal

The code generation and server management tools are known as Railway. The server framework architecture is known as Cabal. These two pieces of software combine to create the complete server framework architecture.

Cabal is centered around a service called CORE. CORE is in charge of onlining processes and managing permissions for all internal communications. A service called CONTENT will be in charge of managing all user-facing data generating services such as collections, and users. The process of onlining is done over second channel communications and are referred to as internal coms. *See 'Figure 1 -Service Architecture' in Appendix.*

Several important technical constraints have been made to ensure that communication and granting permissions among services happens in a quick and secure manner. Google Go has been chosen as the primary backend development language due to easy concurrency handling and native support for the communication mechanism gRPC. gRPC will be the only internal communications method. This has been decided because its' use of HTTP/2 binary serialization along with a well-structured messaging system known as protobufs. Specifically, Cabal will use Proto3 as our protobuf interface.

As a result of these constraints, each service will follow standardized practices. Each service will run inside a Docker container with a range of exposed ports. Each service will have a custom process manager. This process manager will implement the CORE gRPC client stubs that serve as all internal communication methods. The process manager will update the permissions table and then start the process binary. *See `Figure 2` in appendix.* Upon success, failure, and error the process manager will report to CORE. The service will then implement a gRPC server in which it can host its own requirements through, also known as first channel communications, or those between services. Designers of service may also provide a Go Package to further abstract the gRPC communications from the programmer.

All service gRPC servers will report to client requests made by other services or end-user requests via the API Endpoints or Webserver services. These two services will be in charge of coalescing all user requests and providing the HTTP/1.1 response to the user caller. CORE dispatches permissions to all services attached that allow the service to implement the gRPC client stubs of other services. An example of this being the interaction between the Authentication service and the User service. Before credentials can be granted, the Authentication service must make contact with the User service and ensure that the user-given credentials match what is on record in the User service database.

Almost all public requests will be internally routed through the CONTENT service. This is where users of our framework will include their own services and build out their product. The goal of Cabal is to provide an architecture where attaching services is easy for the end user. The goal is to abstract away as much of the microservice architecture as possible. As a result, if the user decides to implement a monolithic service behind the CONTENT service, they can make that choice. Otherwise, if the user wants to build out their own microservices, Cabal has done the heavy lifting to get them started by providing the ability to extend the CORE permissions model and communication mechanisms. Additionally, four default and extensible services have been provided: a logger, a user, an authentication, and a collections service.

These four default services have each been designed to take advantage of the Cabal architecture. Authentication is done via a hybrid token and session design in which Websever or APIEndpoints makes a request, and if successful the response includes two JWT tokens: an access token and a session token. The access token will provide a mechanism to grant credentials to users without having to check the database for each request and instead use a Go Package to quickly check the token against the known keys and kids. If the token is expired, then the session token can be used to renew the user's session for some amount of time before forcing the user to re-enter their

username and password. *See Figure 3 for an example of New Token Authentication.*

The logger is made available via a Go Package that each service can quickly use to connect to the Logger services' gRPC server. Every service will be given access to the logger by default. It serves as the backbone of error tracing communication across the platform via dumping all data into a real time series database and accessed via something such as Grafana that runs alongside Cabal.

The collections service has been provided to support any sort of simple data store needed and can be extended into n number of individual services. This service has been optimized to store things such as blog posts and user comments with minimal effort. With the help of Railway, the programmer can easily extend the types of data that can be stored via a similar experience to Ruby on Rails' command line interface data store generator.

Throughout developing the architecture, Railway provides convenient support tools. As laid out above, the goal of Railway Cabal is to abstract away as much of the underlying architecture of the server as possible without hindering the benefits of having a microservice based framework. Railway provides a convenient way to translate a source program into a service that can be connected to via gRPC by the Webserver. This includes tools that Dockerize services using our process manager and tools to modify services and configurations. For development purposes, the entire framework will be able to be run locally without rebuilding Docker containers for each restart via the process manager in Railway.

Following the lead of both ExpressJS and Ruby on Rails, Railway and Cabal will be released via the MIT license. This has been made as a business constraint because as a team we all believe in the open source model of software distribution. As a framework, the only real value that Cabal can provide is to be made free to modify and distribute as any user chooses so long as they agree to the same.

PROJECT LOGISTICS

Milestones

MILESTONE	ESTIMATION
Initial API, CORE Feature definitions	Nov 2018
Use Case Diagrams, CORE Coms Definition, Process Manager Complete	Late Nov 2018
Static Content Delivery, process mgmt model, Internal Communications Model	Dec 2018
Content Delivery Coms Implementation	Late Jan 2019
CORE services completeness, Integration testing and docs, Impl' plugin extension support	March 2019
Feature Complete, begin debugging, robustness testing, proofing	Early April 2019
Packaging, Documentation, Delivery	May 2019

Work Plan

CONCERN	ASSIGNMENT
Public facing routing and sub-routing, static content delivery, front-end	Alex B
IPC, Containerization, Process Management	Abe D
Internal Communications, Statistics tracking	Noah K
CORE, process permissioning	Qualen P
Code Generation, System Logistics	Ryan R

Budget

While deployment of RailwayCabal software might cost money via Amazon AWS or Microsoft Azure compute time, there will be no development cost and the completed software will be 100% operable without cost.

Ethical and Intellectual Property

The main ethical concern is security. We are providing a software library that heavily depends on communication between services, some of which on the same physical server and others on different servers. It is important that when messages are passed between services that they cannot be intercepted. It is also important that if one service is compromised that it cannot compromise other services. Failure to meet these standards corresponds to violation of several principles of The ACM Code of Ethics and Professional Conducting, including but not limited to:

> **2.5** Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.
>
> **2.9** Design and implement systems that are robustly and usably secure.

We will use two methods to satisfy these principles. First, we will be using gRPC, which will bear most of the burden of security concerns in that it will handle HTTP/2.0 calls. Because gRPC handles most of the lower-level security issues for us, we only need to ensure that there are no high-level security issues. High-level security issues should not be an issue as long as two conditions are met. First, that our server framework is well-designed. Second, that our server framework is well-tested. As long as messages are being passed along expected chains and those chains do not allow other services to be compromised, the security of the library itself should be robust.

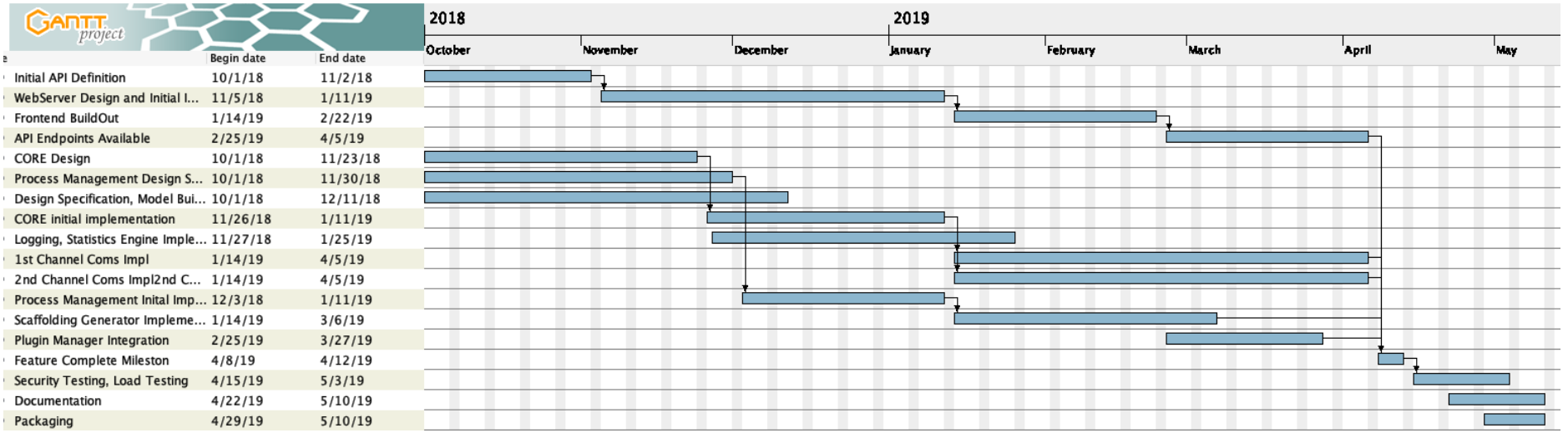
There is also a concern with respect to intellectual property. Our dependencies, such as gRPC, are all open source. gRPC is licensed under the Apache license, so our software must preserve the copyright and license notices. Other dependencies will be handled accordingly. Failure to preserve notices indicated the by the particular license would be a legal violation in addition to an ethical one. For that reason, we will need to pay careful attention to the licenses of our dependencies.

For the licensing of our own software, we are using the MIT License. The MIT License maintains limitations on liability and warranty and also requires that the copyright and license notices be maintained. In exchange, the derived work may be distributed commercially or not as well as with or without source code.

Change Log

Since the initial project description, none of the high-level details have been modified. The overall architecture has remained the same.

GANTT CHART



	Begin date	End date
Initial API Definition	10/1/18	11/2/18
WebServer Design and Initial I...	11/5/18	1/11/19
Frontend BuildOut	1/14/19	2/22/19
API Endpoints Available	2/25/19	4/5/19
CORE Design	10/1/18	11/23/18
Process Management Design S...	10/1/18	11/30/18
Design Specification, Model Bui...	10/1/18	12/11/18
CORE initial implementation	11/26/18	1/11/19
Logging, Statistics Engine Imple...	11/27/18	1/25/19
1st Channel Coms Impl	1/14/19	4/5/19
2nd Channel Coms Impl2nd C...	1/14/19	4/5/19
Process Management Inital Imp...	12/3/18	1/11/19
Scaffolding Generator Impleme...	1/14/19	3/6/19
Plugin Manager Integration	2/25/19	3/27/19
Feature Complete Mileston	4/8/19	4/12/19
Security Testing, Load Testing	4/15/19	5/3/19
Documentation	4/22/19	5/10/19
Packaging	4/29/19	5/10/19

Chart 1 - Project Schedule

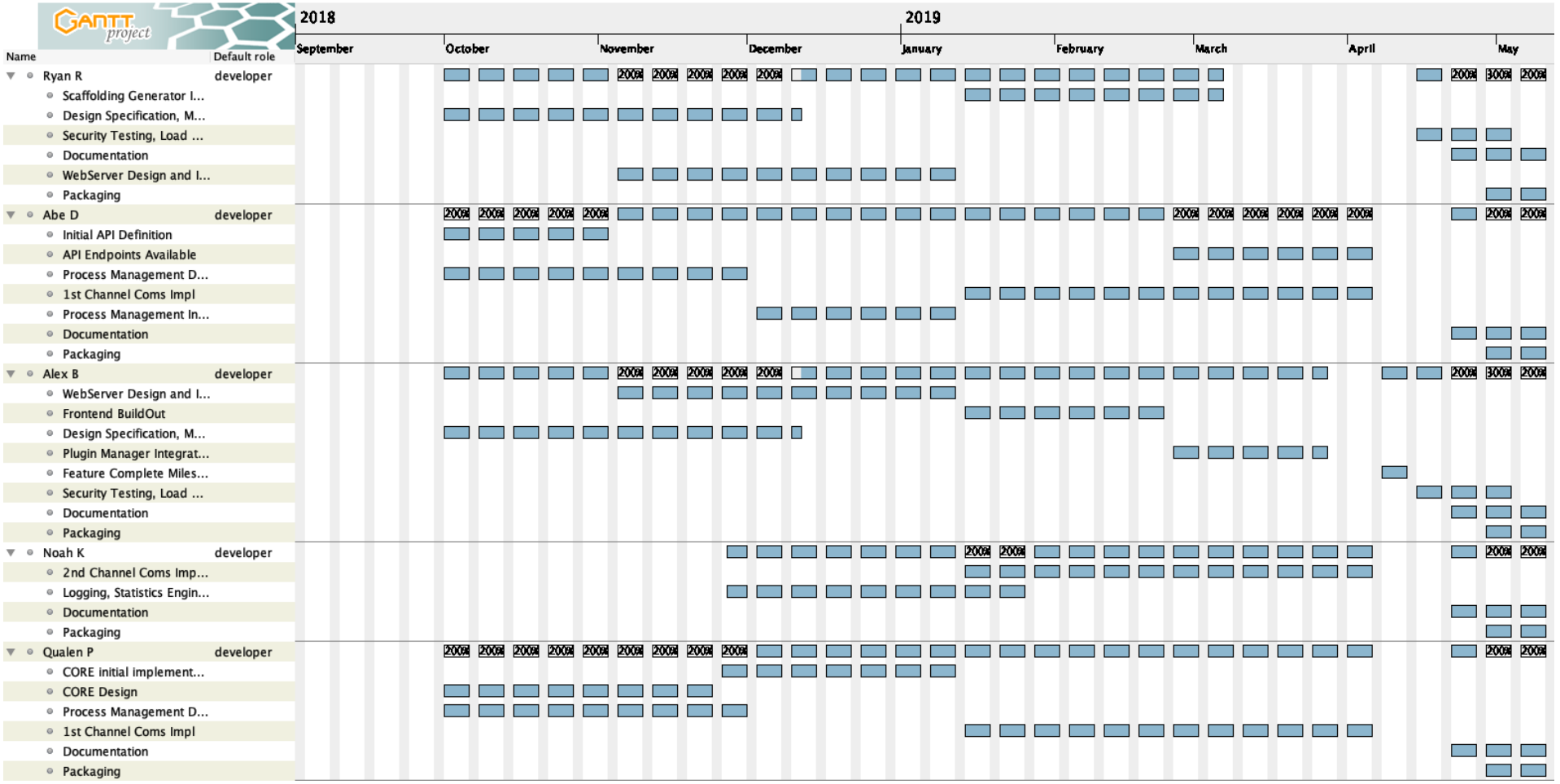


Chart 2 - Resource Distribution Chart

APPENDIX

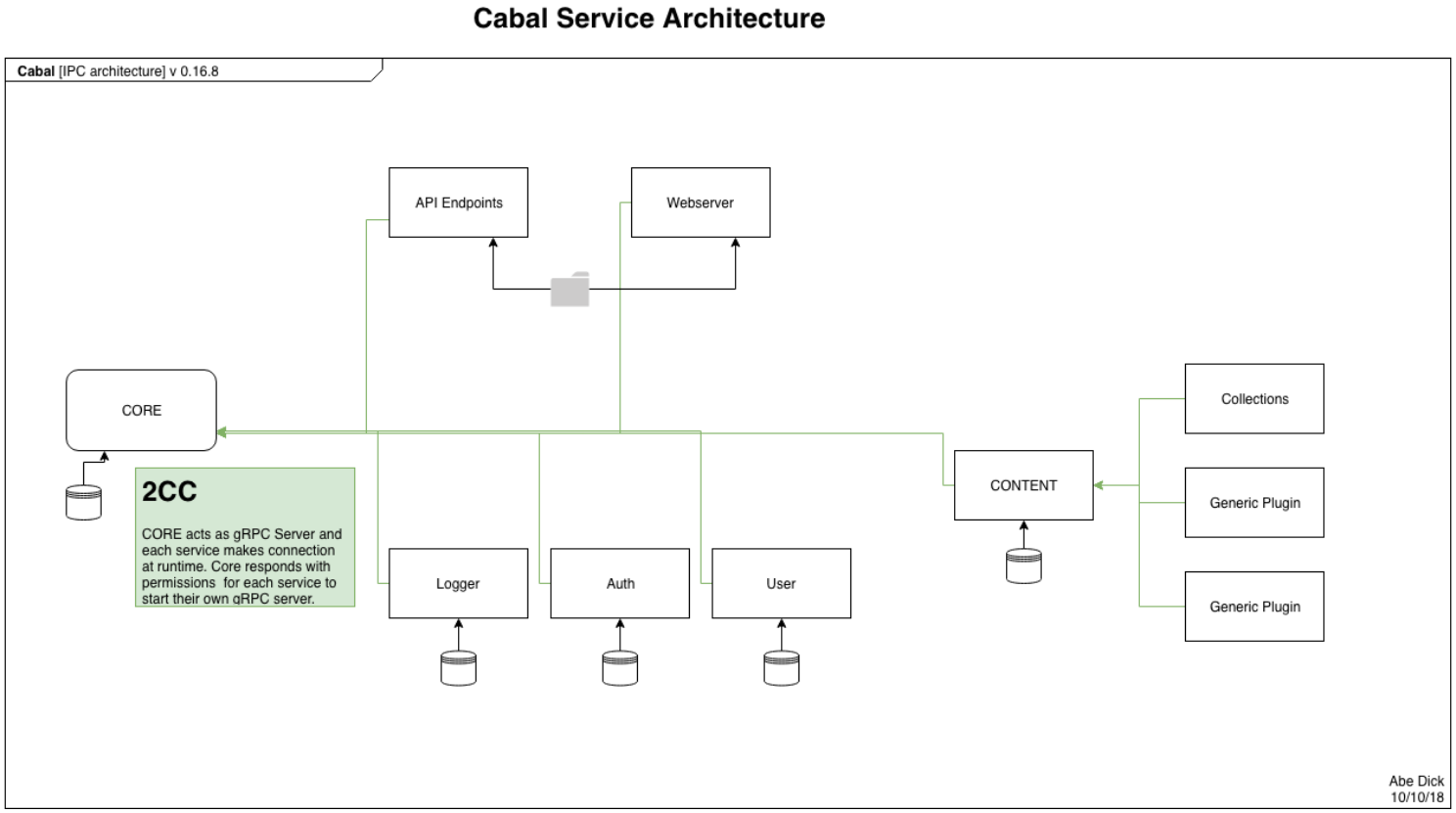


Figure 1 - Service Architecture

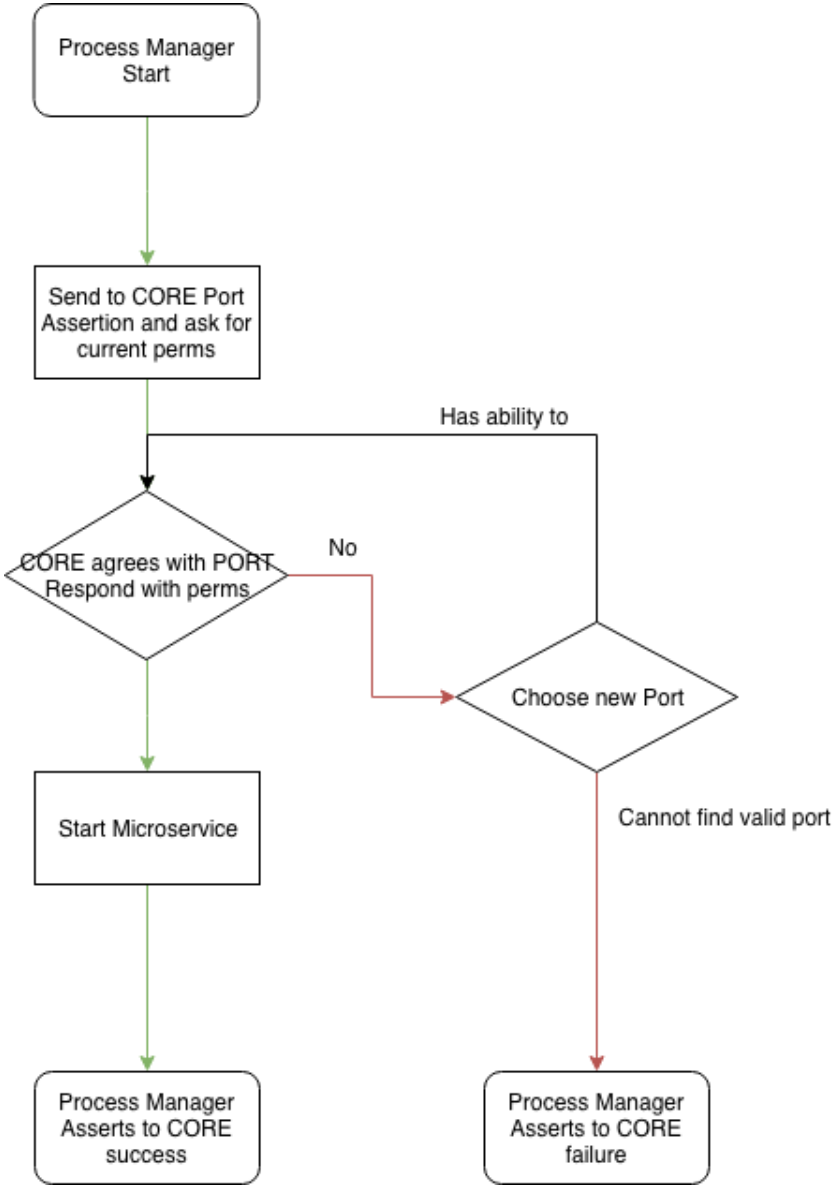


Figure 2 - Starting A Service

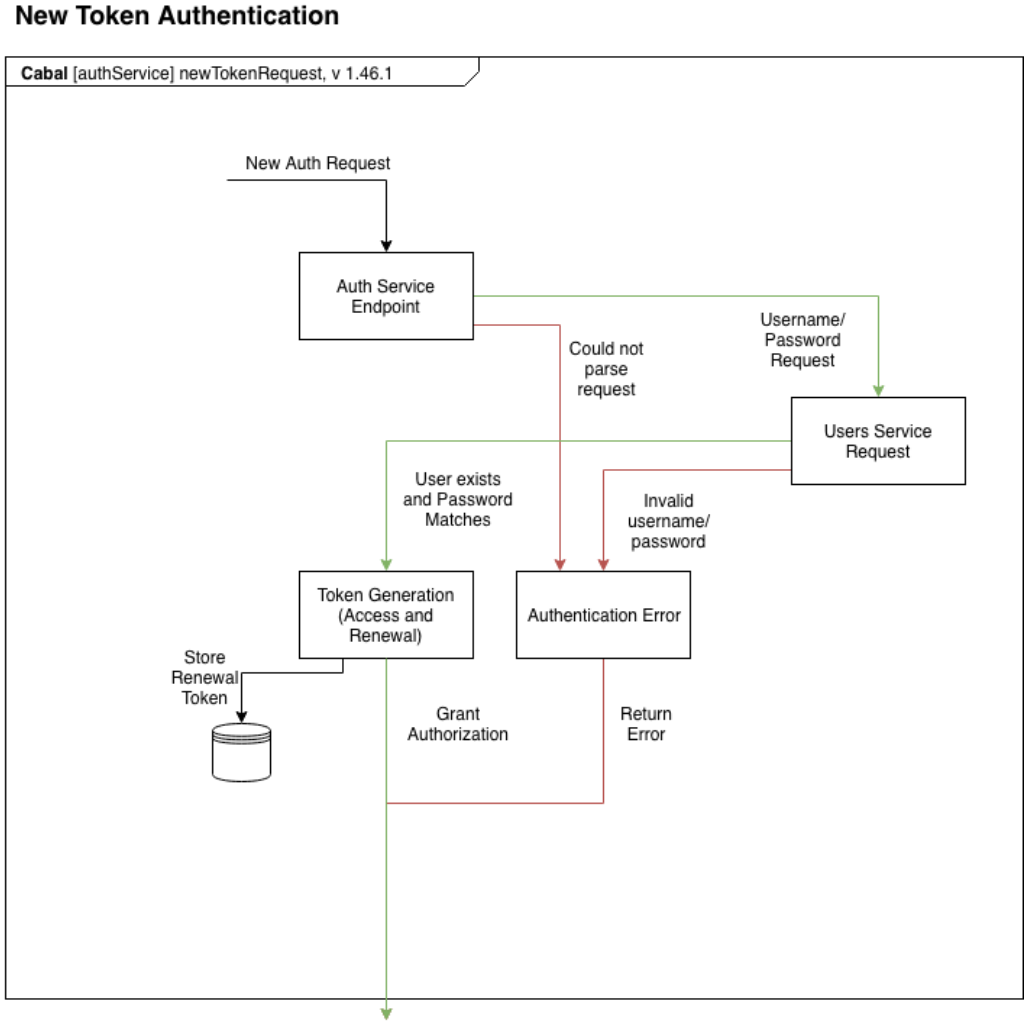


Figure 3 - Token Authentication