

# hammer-io

PROJECT PLAN

**Team:** sdmay18-19

**Client/Adviser:** Lotfi Ben-Othmane

**Members:**

Erica Clark – Data Analytics Lead & Website/Content Management

Nathan De Graaf – Asana Expert & Weekly Status Reports

Nathan Karasch – Project Manager & Technical Writing

Jack Meyer – Communications & Software Architecture

Nischay Venkatram – UI Lead & Node.js SME

**Email:** [sdmay18-19@iastate.edu](mailto:sdmay18-19@iastate.edu)

**Website:** [sdmay18-19.sd.ece.iastate.edu](http://sdmay18-19.sd.ece.iastate.edu)

**Revised:** 5 Dec 2017 (version 3)

# Contents

<b>i - List of Tables and Figures</b>	<b>3</b>
<b>ii - Acronyms and Definitions</b>	<b>3</b>
<b>1 - Introduction</b>	<b>4</b>
1.1 - Project statement	4
1.2 - Purpose	4
1.3 - Goals	4
<b>2 - Deliverables</b>	<b>5</b>
<b>3 - Design</b>	<b>6</b>
3.1 - Previous work/literature	6
3.1.1 - Spring Boot	6
3.1.2 - Docker	7
3.1.3 - Travis CI	7
3.1.4 - IBM Bluemix (now IBM Cloud)	8
3.2 - Proposed System Block diagram	9
3.3 - Flowchart of Project Generation with the CLI	10
3.4 - Assessment of Proposed methods	11
<b>4 - Test Plan</b>	<b>11</b>
4.1 - Validation	11
4.2 - Verification	12
4.2.1 - Continuous Integration & Automated Testing	12
4.2.2 - Manual Testing	12
<b>5 - Project Requirements/Specifications</b>	<b>12</b>
5.1 - Functional	12
5.1.1 - Automated DevOps process for Node.js applications	12
5.1.2 - Framework to develop Node.js microservice applications	13

5.1.3 - Monitoring Interface	13
5.2 - Nonfunctional	13
5.3 - Standards	14
5.4 - Functional and Nonfunctional Requirements Validation	15
<b>6 - Challenges</b>	<b>16</b>
<b>7 - Timeline</b>	<b>17</b>
7.1 - First Semester	17
7.2 - Second Semester	17
<b>8 - Conclusion</b>	<b>18</b>
<b>9 - References</b>	<b>19</b>

## i - List of Tables and Figures

Figure 1	The Spring Boot project generator	6
Figure 2	The Bluemix user dashboard	8
Figure 3	Proposed System Block Diagram	9
Figure 4	Tyr CLI Flowchart	10
Figure 5	First Semester Timeline	17
Figure 6	Second Semester Timeline	17

## ii - Acronyms and Definitions

CI	Continuous Integration
CD	Continuous Deployment
CLI	Command Line Interface
DevOps	DevOps is a software engineering practice that aims at unifying software development (Dev) and software operation (Ops). <sup>1</sup>
Docker	A container platform used for deploying distributed software applications.
GUI	Graphical User Interface
MS	Microservice
Node.js	A framework for running standalone JavaScript applications.
UI	User Interface

---

<sup>1</sup> <https://en.wikipedia.org/wiki/DevOps>

# 1 - Introduction

## 1.1 - PROJECT STATEMENT

From the project abstract: “There is a tendency to develop software as a collection of managed micro-services. Often these software are to be deployed to the cloud. The goal of this project is to develop an environment to develop Javascript-based micro-services that exhibit specific quality attributes and automated devOps process for managing the development and deployment of these JavaScript micro-services.”

We seek to create a framework to automate the development and operation of microservices in JavaScript using Node.js.

## 1.2 - PURPOSE

One paradigm of modern software development is the concept of using a collection of microservices. However, in order for these services to be deployed to the cloud, a developer must go through a significant amount of work to build an automated deployment process. The extra overhead work represents a large barrier to surmount before even a simple “Hello World” application can be created. This barrier can seriously hinder early startups, slow down development, and stifle creativity and innovation. We seek to give developers a tool around this problem.

By creating a framework for Node.js microservices, teams with limited knowledge, resources, and time can quickly get started with a simple infrastructure. We also provide the tools and structure to monitor and maintain the health of their applications in development and in production. This lets more developers start making cool things faster.

## 1.3 - GOALS

The main goal of our project is to create an environment to develop JavaScript microservices as well as a DevOps process to manage and monitor its deployment.

Specifically, we hope to create an app that requires minimal configuration and setup. This means the user would download one or two different tools and then have scripts create most of the configuration files needed with account info (e.g. GitHub, Docker Hub, etc.) supplied by the developers. Easy-to-use CLI/GUI tools would be available to push, deploy, and create new microservices.

Another goal that fits into this process is creating a seamless deployment workflow. This looks like a user pushing code to a master branch and watching his or her changes reflected in the live product a short while later. Once the application is deployed, the user then tracks the status of the application through a Data Monitoring app, which can notify interested parties when a service goes offline, display metrics related to server performance, etc. In addition, the Data Monitoring tool should be able to perform load balancing between microservices.

## 2 - Deliverables

By the end of development for the project, we will have created three products:

1. An automated DevOps process for Node.js applications,
2. A framework to develop Node.js microservice applications, and
3. An interface to monitor the health and status of the deployed Node.js applications.

The first product we are going to develop is the automated DevOps process for Node.js applications. In this product, the user should have the ability to do the following:

- Manage their development workflow
- Have their code automatically pushed to a CI environment to have it tested and built
- Have their code automatically packaged
- Manually deploy and revert deployments of their application
- Perform security analysis
- View common DevOps statistics such as code coverage stats, code style stats, percentage of build failures, status of builds, percentage of test failures, and time for task completion

The DevOps application will be able to be consumed in two different ways. The first is the CLI. Here the user will be able to have the code base for their project automatically generated. This should include the configuration files needed for the many services our application will be connecting to. The web application will allow users to view statistics about the DevOps process, manage the build and deployments of their services, and perform the same functions as the CLI. The DevOps application should be able to be deployed on the user's own server or be consumed through our cloud service.

The second product we will create is the microservice development framework. This will be distributed with the DevOps application in that it will use the CLI to generate new microservices in the project and link them to existing services. As discussed in the timeline (section 7), this will be one of the last pieces of the project that we implement. Because it will leverage the existing foundations built in the DevOps application, it should be fairly straightforward to extend the CLI to be able to generate the required microservice components.

The third product we are going to build is an interface to monitor the health and status of the deployed Node.js applications. The user should have the ability to do the following:

- View uptime statistics about their deployed applications
- View data flow statistics (how much data is going in and out of the applications)
- View history about both of the above

The monitoring interface will be able to be consumed via a web application. Here the user will have the ability to view the information listed above in a dashboard like interface. This functionality should come out of the box and require no configurations if the user chooses this service for their product. Again, this application should be able to be consumed through our cloud service or deployed on the user's own server.

## 3 - Design

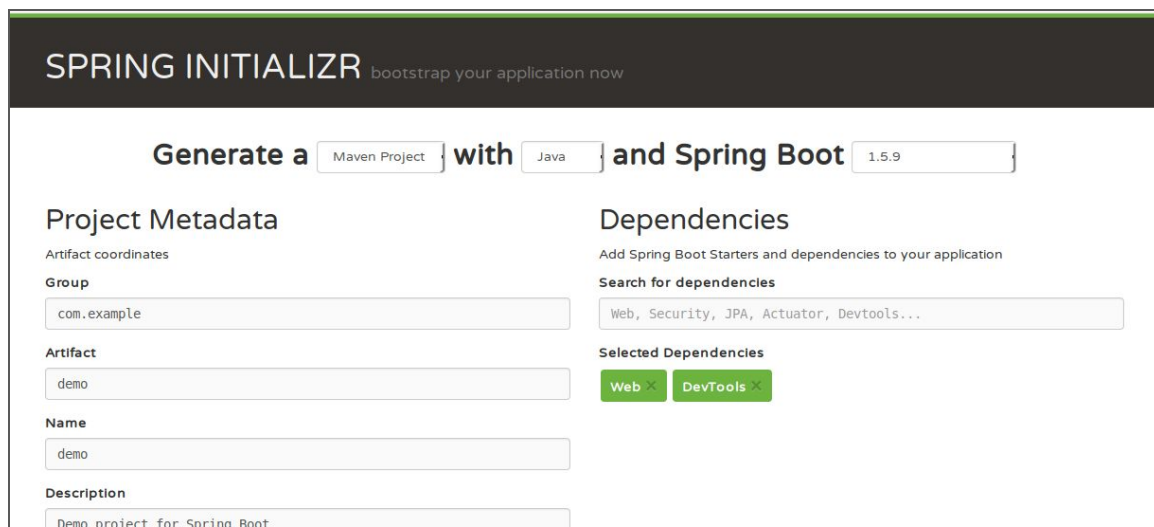
### 3.1 - PREVIOUS WORK/LITERATURE

#### 3.1.1 - Spring Boot

Spring Boot is an opinionated approach to the creating Spring Based Applications. The Spring Framework is a Java framework which makes it very easy to make web based applications. Spring Boot makes it easy to create applications in the Spring Framework by including third party libraries and doing most of the configuration work. Spring Boot has the ability to make standalone applications, which already include a server embedded. It automatically does configuration and pom file generation, meaning it easy to get up-and-running quickly. Finally, it provides health checks and metrics for the application.

Through the addition of some of the open source libraries such as Zuul, Hystrix, RabbitMQ, and Eureka, Spring Boot has become the most common choice for creating a microservice architecture. Spring Boot does many other things such as providing ways to register applications with Facebook and Twitter and providing a uniform way to access data in Neo4J, MySQL, and MongoDB. Finally, there are easy ways to automate the configuration of Docker to make the Spring Boot application into a docker container.

This is only scratching the service of what this framework can do. We want to do some very similar things with hammer-io. However, our product will live in a completely different domain. Instead of Java, we will be targeting node.js developers. Below is a screenshot of the Spring Initializr<sup>2</sup>, a handy means of generating a Spring Boot project on the web. We hope to provide a similar means of project generation for our users.



The screenshot shows the Spring Initializr web application interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below this, there is a form to generate a project. The main heading is "Generate a Maven Project with Java and Spring Boot 1.5.9". The form is divided into two main sections: "Project Metadata" and "Dependencies".

**Project Metadata**

Artifact coordinates

**Group**  
com.example

**Artifact**  
demo

**Name**  
demo

**Description**  
Demo project for Spring Boot

**Dependencies**

Add Spring Boot Starters and dependencies to your application

**Search for dependencies**  
Web, Security, JPA, Actuator, Devtools...

**Selected Dependencies**  
Web X DevTools X

Figure 1. The Spring Boot project generator.

<sup>2</sup> <https://start.spring.io/>

### 3.1.2 - Docker

Docker is a widely-popular containerization platform that makes it easy to deploy software with specific environment requirements. It's sort of like a Virtual Machine, but it shares the kernel with the host machine it runs on, so it can share many common processes, keeping it lightweight and fast. There are many benefits of using docker, the first being that it takes the environment complexity out of a project. If your software needs specific libraries and other versioned software in the environment, it can be configured with a dockerfile, and those libraries and such will be sure to exist in the docker container when it gets run. This means that it works the same on any machine it's run on, regardless of the machine's environment.

Some other benefits include easier performance and traffic monitoring, since everything gets bundled and executed as a single file. It also comes with a lot of "security by default" measures, meaning they are turned on by default instead of requiring the user to manually turn them on. Take isolation for example. If you have one container serving your web user interface and other containers serving other services, if an attacker were to gain access to the web UI container, he would not be able to move laterally or escape the container to access files in another container or in the root system. This is only one of the many security features that come with a docker container.

### 3.1.3 - Travis CI

Travis CI is a continuous integration tool that is widely used across most open source projects and by large companies like Facebook, Twitter, Mozilla, etc. It is compatible with a variety of different languages like C, C++, C#, Clojure, D, Erlang, F#, Go, Groovy, Haskell, Java, JavaScript, Julia, Perl, PHP, Python, R, Ruby, Rust, Scala and Visual Basic. Continuous integration tools like Travis CI, allow users to build, test, and deploy applications.

The application is built and tested on the Travis CI servers/virtual machines. Users can choose to deploy to their own servers or to external cloud hosting services like AWS, Heroku, Azure, etc. All the required configurations are passed as a config file in the code repository. Travis CI also supports Docker and can build, pull, and push images to external Docker registries. Travis CI can also be integrated with code hosting platforms like GitHub, to run builds on code changes, new pull requests, or commits.



### 3.1.4 - IBM Bluemix (now IBM Cloud)

“IBM Bluemix is a cloud platform as a service (PaaS) developed by IBM. It supports several programming languages and services as well as integrated DevOps to build, run, deploy and manage applications on the cloud.”<sup>3</sup> Started in 2014, Bluemix offers the same sort of DevOps platform that we are aiming to create, with several notable exceptions. The first major difference between our product and Bluemix will be our focus on streamlined development of microservices in Node.js. While Bluemix supports the Node.js language, it doesn’t emphasize microservice architectures, nor does it bootstrap projects for you or enable the entire DevOps pipeline in a single command.

A second notable exception is that most of Bluemix’s functionality is locked behind a paywall. Our products, on the other hand, will be open-source and freely available for use.<sup>4</sup>

Finally, Bluemix aims to be a more all-around DevOps platform, supporting many languages and integrations, so it doesn’t focus heavily on microservices in particular. Though it was recently rebranded, the product is already three years old and fully funded by IBM, so it will obviously be more fully featured than we can hope to achieve in two semesters. However, it can still serve as an example for our product to model as we continue with development. The screenshot of the Bluemix dashboard<sup>5</sup> shown below offers insight into what our own DevOps platform may look like in the future.

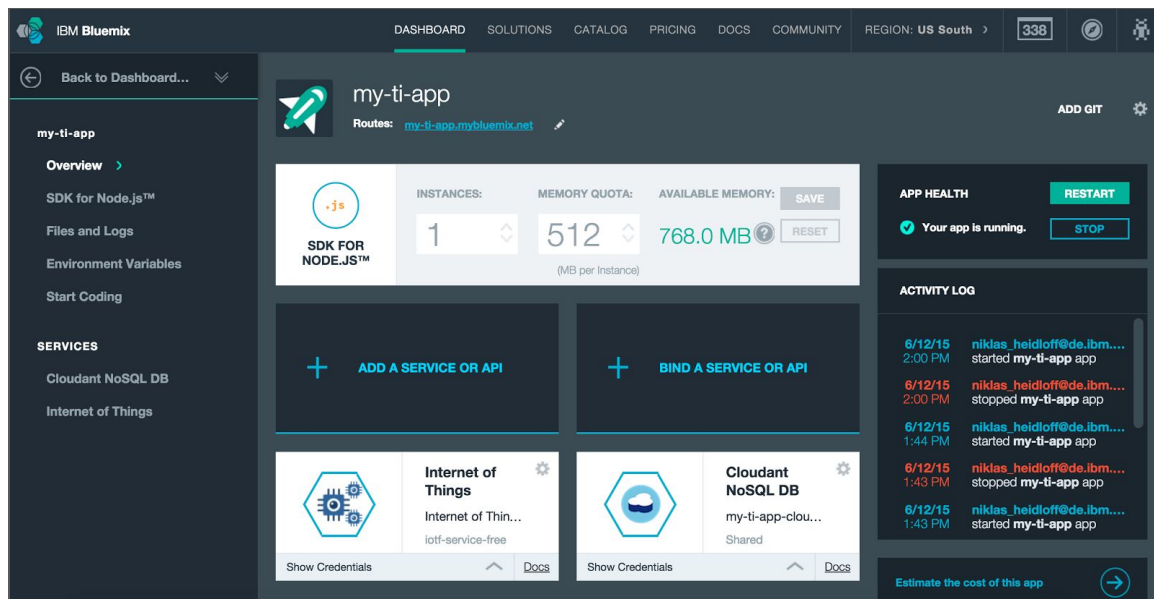


Figure 2. The Bluemix user dashboard.

<sup>3</sup> <https://en.wikipedia.org/wiki/Bluemix>

<sup>4</sup> Our product will be freely available for use; however, for those wishing to use it as teaching material in a class, they must gain express permission from the project owners/maintainers.

<sup>5</sup> <https://console.bluemix.net/>

### 3.2 - PROPOSED SYSTEM BLOCK DIAGRAM

The following is the proposed system block diagram for the deployed web application portion of the project. The user has the option of generating a project from the CLI or downloading a zip archive of the files generated by the web platform. Once the project is created, it gets pushed to Source Control (GitHub) by the CLI, enabled in the Continuous Integration suite (TravisCI), and deployed to the Cloud Hosting platform (Heroku). The TravisCI automation, which runs static and dynamic tests, is triggered by web hooks in the GitHub repository. Tests are run on code pushes, merges, and pull requests, and upon merging to master it is setup to deploy a docker container of the app to Heroku.

The DevOps Monitoring Platform provides the graphical user interface to monitor various aspects of the project: hosting uptime, test results, code coverage, repository statistics, etc.

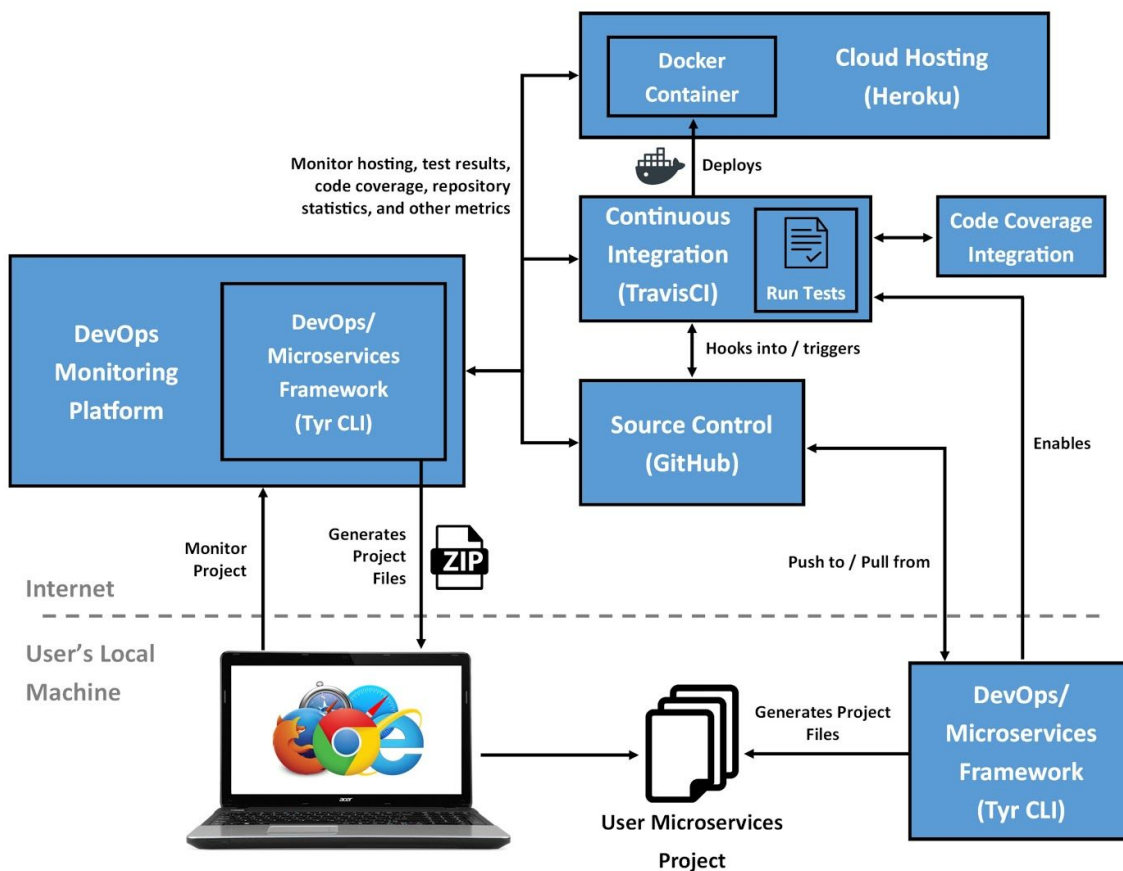


Figure 3. Proposed System Block Diagram

### 3.3 - FLOWCHART OF PROJECT GENERATION WITH THE CLI

As mentioned above, the CLI will setup the user's microservices project with a DevOps workflow. It takes various information about the project as input, as well as which 3rd party integrations the user wants to use, and generates the project files. If the user has selected integrations for source control, continuous integration, deployment, etc., the CLI will also push the code to the repository and initialize all appropriate environments upon generation of the project files. Below is a flowchart outlining some of the functionality provided by the CLI.

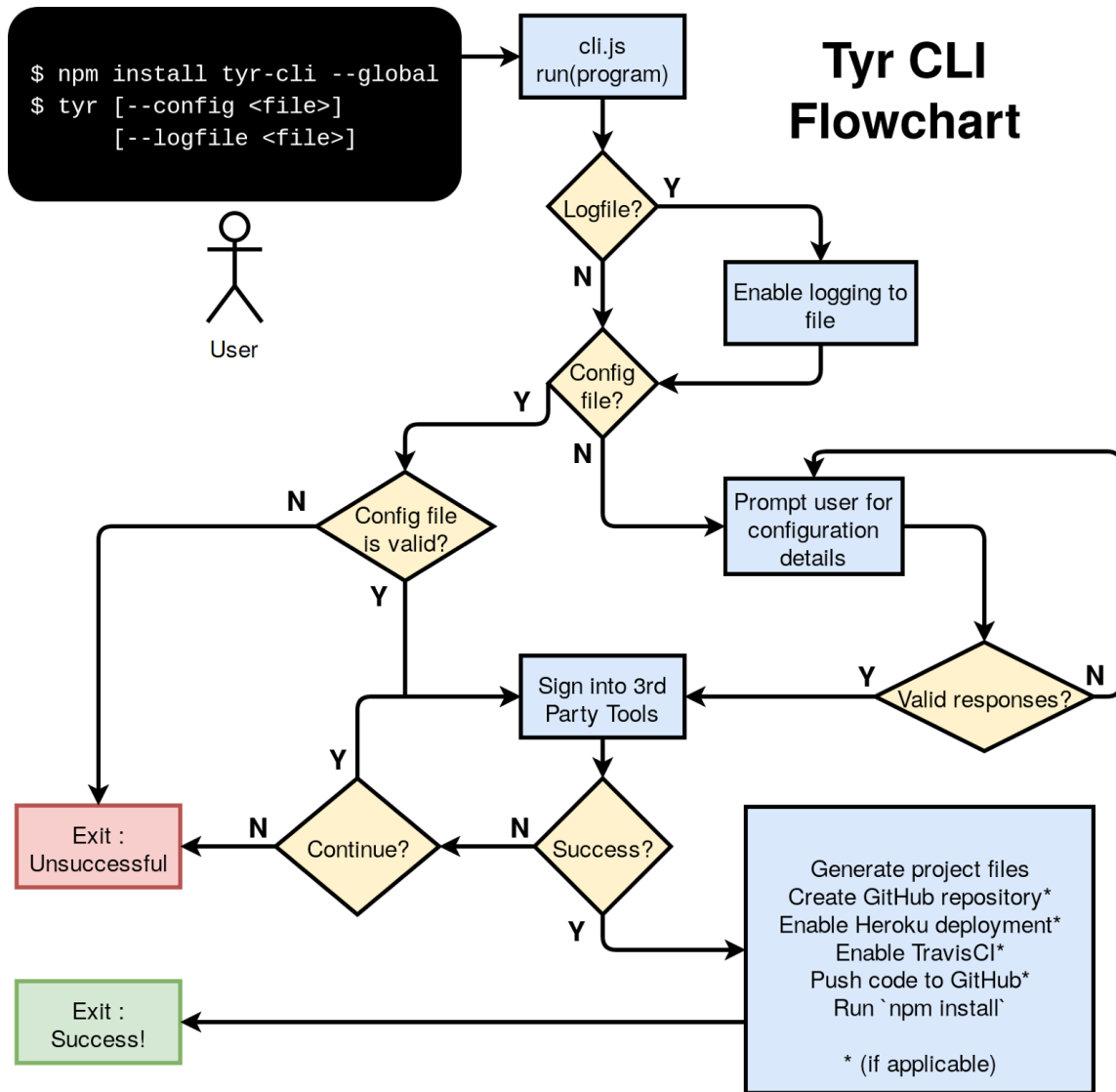


Figure 4. Tyr CLI Flowchart

### 3.4 - ASSESSMENT OF PROPOSED METHODS

Since a majority of the project involves standardizing around different tools and bringing them into one unified workflow, we had a plethora of options to choose from. For a Continuous integration tool in our framework, we decided to go ahead with Travis CI. Jenkins was the other option available, but our focus audience for this project were small teams, students, and startups. Travis CI takes care of building and testing the application on their own servers, while Jenkins would need to be installed on a personal server or a cloud hosting service. Given that smaller teams and startups do not have the resources to manage and maintain additional servers, Travis CI was the logical choice.

In terms of deployment and architecture, the traditional approach would be to develop a huge monolithic application, build it as a whole, and deploy it. The other route was using Docker and microservice architecture to containerize any application into smaller isolated components for additional benefits like scalability, load balancing, and more uptime. For example, suppose there are two services – payments service and authentication service. Should the payments service crash, the authentication service will be unaffected and will continue to function normally. In addition, Docker lets us spin up new containers of the payments service, should it crash, or if the number of requests that need to be processed is high. Enforcing this architecture and helping build such applications using our framework seemed like the correct approach.

## 4 - Test Plan

### 4.1 - VALIDATION

The project will be built in an escalating series of prototypes, each of which adds another layer of integration with additional software. This iterative development will allow us to continually analyze if our solution is valid. That is, we will continually assess the product to ensure it meets the business goals of the client. This validation with the client takes place at weekly meetings where we demo our progress and brief the client on the direction and planned development for the following week.

Through our initial research, we have discovered that there are many possible software options for each piece of functionality we need. If a software choice fails to integrate nicely with the rest of the project, we have the option of trying other options for that function.

For example, suppose we are integrating a code coverage tool to run during the continuous integration operations. There are several solid options, including CodeCov, Blanket.js, Istanbul, and Coveralls. Suppose we try to integrate CodeCov with the prototype, but there is a conflict with a previously-selected tooling. We have the option of choosing a different piece of software for code coverage, or we can revisit the other piece of software that we selected before. Upon reaching that decision point, we will confer with the client and assess which direction would better suit his end goal and meet the needs of the end user. In this manner, we can continue to make development progress on the project with continual validation as we complete each iteration.

## 4.2 - VERIFICATION

In order to make sure that each portion of our project meets both functional and nonfunctional requirements, we will verify our products in several ways:

### 4.2.1 - Continuous Integration & Automated Testing

First, we will have an automated test suite (facilitated by TravisCI) connected to our code repository. This test suite will be run in Travis for every commit to the repository. This test suite will, to the best of our abilities, cover all areas of our code base. It includes all types of testing (Unit Testing, Integration Testing, etc.) other than System Testing and Acceptance Testing. Merging from a feature branch to master cannot happen unless all tests are passed, and reviewers will ensure code authors have written unit tests for new code. This will ensure that when a new feature is added, previous features have not regressed.

### 4.2.2 - Manual Testing

Manual testing will be the means of conducting System Testing and Acceptance Testing. We will manually test the code upon integration of new features and functionality. This will ensure that both the functional and nonfunctional requirements are being met. We will also demo our application to our client at the weekly meetings when a new feature has been implemented. By demoing to our client, we will ensure that the vision and the goals of the project are being met.

## 5 - Project Requirements/Specifications

### 5.1 - FUNCTIONAL

#### 5.1.1 - Automated DevOps process for Node.js applications

- Provides a CLI, which contains commands to:
  - Setup a new project (generates default configuration files) based on the services that a user wants to include
  - Serve the web application version of the CLI, which must:
    - Allow the user to select which services he/she wants to include in a new project
    - Download the default files for a new project based on the services that a user wants to include
    - View reports, results, and statistics about the DevOps process
    - Manage the build and deployment of services
  - Automate delivery of code to the CI environment, which must:
    - Build the application
    - Package the application in a Dockerfile
    - Run test suites
    - Report status and results
- Ability to configure various services with a configuration file
- Provides documentation for configuration

- Provides statistics, reports, and analytics about the development process
  - Issue completion
  - Development time per issue
  - Code Coverage
  - Build information

### 5.1.2 - Framework to develop Node.js microservice applications

- Provides a CLI, which contains commands to:
  - Generate Node.js templates for new microservices
  - Integrate new microservices into the existing microservice architecture
  - Configure the microservices (host, port, etc)

### 5.1.3 - Monitoring Interface

- A web application, which must:
  - Provide useful statistics, reports, and analytics about the deployed microservices
    - Uptime statistics
    - Data usage
  - Allow reports and logs to be downloaded

## 5.2 - NONFUNCTIONAL

- Usability
  - The system will only support the English language
  - The CLI must have a clean, consistent look and feel
  - The web application must have a clean, consistent look and feel
  - The application will be usable by those who have a limited understanding of DevOps
- Supportability
  - The system will support Unix-based systems (e.g. Mac or Linux)
  - The system will support Node.js version 8.x
- Reliability
  - The deployed web applications will run 24 hours a day, 7 days a week
  - Uptime for deployed web applications shall be >99%.
- Security
  - The system will not store any plain-text passwords in configuration files or elsewhere

### 5.3 - STANDARDS

The team will use the following standards during development of the project:

- Version Control System
  - Git will be used as the primary means of version control for all project code.
  - The Google software suite (Docs, Sheets, etc) will be used for all formatted documentation, such as planning and design documents. It has version control features that can be accessed from the menu (File → Version history).
- Code Review
  - Development will be done in feature branches.
  - When a feature branch is ready to be merged to the master branch, the author will assign one or more of the other project members as a reviewer.
  - Reviewers must check to ensure the code:
    - correctly implements the desired functionality
    - contains sufficient tests to ensure correctness
    - passes tests
    - is free of errors
    - integrates successfully with the existing software
  - Reviewers will communicate code issues to the author, who is then responsible for addressing all issues.
  - Once the code has passed inspection, it can be merged to master.
- Testing Standards
  - Unit Tests:
    - Unit tests shall be written for all code committed to the repository.
    - The author of the code is responsible for its unit tests, and the code reviewers are responsible for holding the author accountable.
  - Integration Tests:
    - Once a significant number of components have been integrated, tasks will be created and assigned to developers to write integration tests.
    - The integration tests should ensure the subset of components behave together as intended.
  - System and Acceptance Testing
    - Manual testing by developers will be the standard for System Testing. This will take place upon merging new code and upon completion of major portions of functionality.
    - Product demos to the client will be the standard for Acceptance Testing. These take place on a weekly basis at the team meeting.
  - Code Coverage:
    - We will install a code coverage tool into the continuous integration suite to report what level of test coverage we have achieved. Given the short time frame of the project and the rapid pace at which a new project is developed, we likely won't shoot for a hard code coverage metric until the second semester. At that point we will evaluate whether code coverage makes sense in the context of our project and what percentage we should be targeting.

## 5.4 - FUNCTIONAL AND NONFUNCTIONAL REQUIREMENTS VALIDATION

As mentioned previously, validation is conducted through manual testing. It is roughly outlined as follows:

- Verify documentation at the project website meets documentation requirements.
- Manually verify the deployed web applications meet the functional and nonfunctional requirements outlined above.
  
- Verify the usage of the CLI from the command line meets requirements:
  - Without configuration file:
    - Run: ``npm install --global tyr-cli``
    - Run: ``tyr``
    - Follow the prompts, and input information for a new project.
    - Enter any user credentials for services you have selected.
    - Verify project files were created successfully.
    - Verify code was pushed to the code repository (if applicable)
    - Verify unit tests were run in the continuous integration suite (if applicable)
    - Verify the new project was deployed to the selected hosting environment (if applicable)
  - With configuration file:
    - Create a configuration file `config.json` for a new project (follow format given in documentation)
    - Run: ``tyr --config config.json``
    - Enter any user credentials for services you have selected.
    - Verify project files were created successfully.
    - Verify code was pushed to the code repository (if applicable)
    - Verify unit tests were run in the continuous integration suite (if applicable)
    - Verify the new project was deployed to the selected hosting environment (if applicable)



## 6 - Challenges

The most significant challenge our group faces is defining the scope and direction of our project. We do not have a single clear client or use case; rather we are developing a tool that can be adopted and adapted by the open source community for various JavaScript microservice DevOps needs. Our group must initially spend a lot of time thinking about which use cases to prioritize and what types of users to cater to. For example, consider a large established company, a small startup, and an academic class of students. Each will have different resources, skills, and needs.

Our product is not unique in its market, so various competitors will be challengers for our group and product. Some of these tools are professionally developed and maintained, so our product must be especially well-suited to a specific need that these other DevOps and automated development tools are not meeting.

We hope cost will not become a challenge for our group; however, since our project is likely to be using services from various other products, we may run into paywalls at some point when scaling or accessing APIs. We hope our product will be free to use; however, as a user scales, they may be required to pay for subservices.

Our group has a remarkably strong collective knowledge of the area, including tools, processes, and best industry practices. However, one challenge could be a lack of user empathy as a result of few of the team members having ever worked in developing javascript microservices. Hopefully our advisor/client will be able to provide this knowledge and guidance for our group.

## 7 - Timeline

### 7.1 - FIRST SEMESTER

The following is a Gantt chart outlining the proposed timeline for the project's development through the first semester. The blue bars indicate project phases.

Aug				Sept				Oct				Nov				Dec					
W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4		
		Requirements gathering																			
				Research																	
				Build CLI tool for app generation																	
												Web app setup									
																Demos					

Figure 5. First Semester Timeline

The first few weeks of the project, we will spend time understanding the requirements, researching, and prototyping. Most of our design thinking will happen here and we'll make sure to meet with the client often in order to get the requirements. After we finish the first few weeks, we will start work on the command line tool to build an application and deploy it from scratch. We plan to spend a decent part of the semester building the CLI. Early November, we plan to transition to architecting and building the web app where users can set up, deploy, and monitor the deployed applications. Most of the work for the web app will continue into the next semester.

### 7.2 - SECOND SEMESTER

The following is a Gantt chart outlining the proposed timeline for the project's development through the second semester. The blue bars indicate project phases.

Jan				Feb				Mar				Apr					
W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4		
Monitoring Web application																	
				Deployment Web application													
										Development framework							
								Testing, Validation, Polishing									

Figure 6. Second Semester Timeline

The plan for the first 2 months is to continue working on the web application to monitor and deploy microservice applications. There will be some overlap while working on the monitoring solution and deployment solution. We will continuously test and validate the solution. In early to mid March the plan is to slowly transition to developing the development framework that will allow users to code scalable microservice applications. We plan to wrap everything up by the end of April.

## 8 - Conclusion

The popularity, simplicity, and usefulness of JavaScript microservices means there is a need from users for an easy to use and an all encompassing framework and devops systems to build and deploy these JavaScript services. The goal of this project is to meet these needs. Hammer seeks to provide three main parts: an automated deployment process, a Node.js framework to accommodate and enable microservices, and a data monitoring platform to monitor the health and metrics of the entire system.

In order to achieve this functionality, our team plans to implement the following three parts of the overall product. First we build a DevOps application to manage the code base of the service. This can be managed through either a CLI or a web application. Building these applications will be the first step of the project. Once finished with this portion, the user should be able to deploy their own service to be consumed by their users through our application. Second, we will create a monitoring application that will build upon the DevOps web application. This shows the users all relevant information about their information including logs, load, uptime, etc. This monitoring application is part of the features that makes our application better for our clients than managing each of these services individually. Finally, we create a framework to allow users to write these microservices in NodeJS. More details and plan for implementation to come.

Hammer, our solution to all JavaScript microservice development needs, seeks to meet the specific needs of our users in a way that other similar services cannot match. Our application's opinionated approach of setting up a suggested approach without the user needing any specialized knowledge makes Hammer the perfect choice for students or small teams that cannot afford to waste time or resources setting up and managing the logistics of their application. By focusing on simplicity, ease of use, and specializing in JavaScript microservices, our application can provide a better service than any other general DevOps or development framework service.

## 9 - References

- <https://www.docker.com/>
- <https://docs.docker.com/engine/security/security/>
- <https://hub.docker.com/>
- <https://docs.docker.com/engine/swarm/>
- <https://projects.spring.io/spring-boot/>
- <https://github.com/>
- <https://about.gitlab.com/>
- <https://bitbucket.org/>
- <https://sourceforge.net/>
- <https://www.sonarsource.com/products/codeanalyzers/sonarjs.html>
- <http://www.veracode.com/products/static-analysis-sast/static-analysis-tool>
- <https://scan.coverity.com/>
- <https://geekflare.com/nodejs-security-scanner/>
- <https://codecov.io/>
- <http://blanketjs.org/>
- <https://github.com/gotwarlost/istanbul>
- <https://coveralls.io/>
- <https://karma-runner.github.io/0.8/index.html>
- <https://circleci.com/docs/1.0/language-nodejs/>
- <https://travis-ci.org/>
- <http://www.shippable.com/>
- <https://jenkins.io/>
- <https://codefresh.io/>
- <https://kubernetes.io/>
- <http://mesos.apache.org/>
- <https://mesosphere.github.io/marathon/>
- <http://www.shippable.com/>
- <https://en.wikipedia.org/wiki/Bluemix>
- <https://console.bluemix.net/>