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
Website: sdmay18-19.sd.ece.iastate.edu

Revised: 24 Sept 2017 (version 1)

Contents

i. List of Tables and Figures

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).[1]
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

[1] 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.

A detailed description of the framework and monitoring service will come at a later date.

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.

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.

3.2 Proposed System Block diagram

The following is an early draft of the proposed system block diagram. The details will be refined as the design continues to be fleshed out.

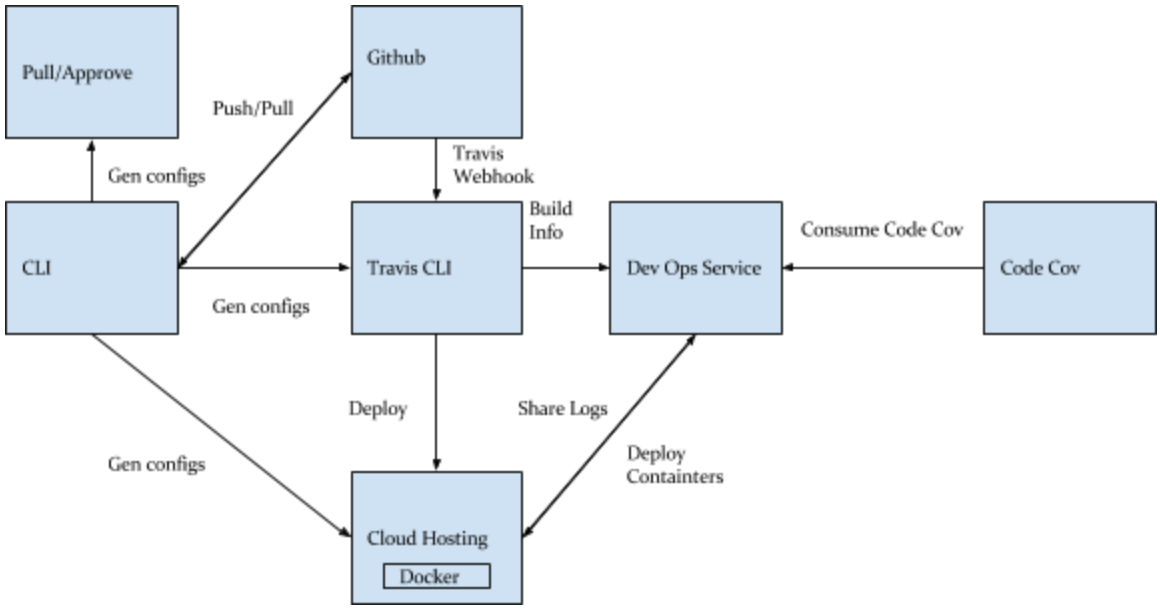

Figure 1. Proposed System Block Diagram

3.3 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 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 mircroservice 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.

3.4 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. 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.

In this manner, we can continue to make development progress on the project with continual validation as we complete each iteration.

4 Project Requirements/Specifications

4.1 FUNCTIONAL

4.1.1 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

4.1.2 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)

4.1.3 Monitoring Interface

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

4.2 NON-FUNCTIONAL

- 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
    - We expect to have an update of greater than 99%
- Security
    - The system will not store any plain-text passwords in configuration files or elsewhere
- More non-functional requirements TBD

4.3 Standards

The team will used 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 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
    - To Be Determined

5 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.

6 Timeline

6.1 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, and the orange bars indicate deliverables.

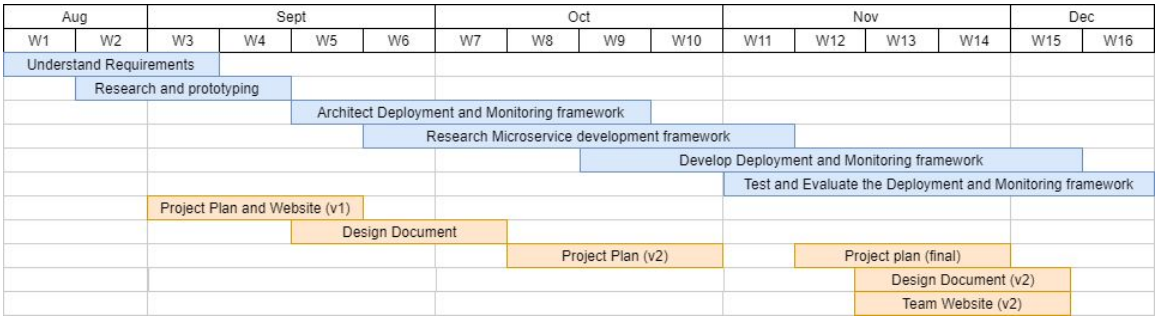

Figure 2. First Semester Timeline

6.2 Second Semester

The timeline for the second semester will be determined at a later date during the first semester.

7 Conclusions

Hammer consists of three main parts: the 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. Hammer will only require simple installation and minimal overhead to set up, making it an ideal choice for small teams that want to focus their limited time and resources on creating their application.

8 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/