Hammer-io IoT Extension

Design Document

Team: sddec19-24
Client/Advisor: Lotfi Ben-Othmane
Members:

Yussef Saleh	*Team Leader*
Matt Bechtel	*Chief Designer / Engineer*
Brett Wilhelm	*Assistant Programmer / Engineer*
Chakib Ahlouche	*Assistant Programmer / Engineer*

Email: sddec1924@gmail.com
Website: sddec19-24.sd.ece.iastate.edu

Table of Contents

Project Statement

Our project is to extend the existing DevOps system 'hammer-io' (a past senior design project) to support deployment of applications to IoT devices and also to provide basic services for micro-services such as fail detection. We will continue working on a framework for DevOps for Node.js that our client has previously developed.

Project Goals and Deliverables

The goal of the project, as mentioned above, is to provide a extension to the current DevOps system, hammer-io, that will allow clients to deploy their services to the IoT device of their choosing. As of now the system only allows deployment to Heroku, so we will have to deliver an extension to the project that allows for deployment to IoT devices. This extension will be in the form of the new services 'IoT Deployment Service' and the 'IoT Deployment Manager'. These new services will add on to the existing microservices architecture of the hammer-io system, easily integrating and allowing for clients to have the option to use the hammer-io system to push and deploy services to their desired IoT devices. The two new services will work in conjunction to provide this functionality, with the 'IoT Deployment Service' running alongside the rest of the hammer-io system in the cloud and the 'IoT Deployment Manager' running on the actual IoT device.

The deliverable will be as mentioned above, the extended hammer-io system with the new services 'IoT Deployment Service' and 'IoT Deployment Manager' added on to provide the needed IoT deployment functionality.

Proposed Design

Our extension design for the hammer-io DevOps system will involve the creation of a new 'IoT Deployment Service', and its companion (which will run on the IoT device itself) the 'IoT Deployment Manager', for the deployment of client code onto Linux-based IoT devices. The existing hammer-io codebase was built using a microservices architecture, so we will simply add another microservice and integrate it with the existing system.

To integrate the 'IoT Deployment Service' with the existing services, endor, koma, tyr, and the frontend yggdrasil, we will have to extend them so that they may communicate with our new deployment service in the same fashion they communicate with Heroku (the deployment service currently used by hammer-io for deployment of docker containers).

When implemented, the 'IoT Deployment Service' will field requests from the backend service, endor, to control a client's deployment. To start a deployment pipeline, the client will need to 1. Setup their IoT device so that it is running Linux and is connected to the internet, 2. install our instance side service, the 'IoT Deployment Manager', and run it on a port which they have opened to the public (or at least open to the 'IoT Deployment Service's' IP). From here, the client should go through either the UI, yggdrasil, or the CLI tyr to pass off their IoT device endpoint (IP plus the port), the git path and credentials of the code they want to run on the device, and the launch script for said code.

Once the 'IoT Deployment Service' has the information necessary to talk to the client's IoT device, the client can then kick off their deployment. Deployment will involve a request from either yggdrasil or tyr to the backend endor, which will then make a request to the 'IoT Deployment Service' to begin deployment. When the 'IoT Deployment Service' receives a request it will then in turn send a request to the 'IoT Deployment Manager' which is running on the client's specific IoT device to perform the deployment-related command.

The end goal of the 'IoT Deployment Service' and its companion the 'IoT Deployment Manager' will be to function as an enterprise deployment service, such as Heroku, but for IoT devices.

Design Specifications: IoT Deployment Service

Functional Requirements

- Provides API for deployment requests
    - Initial Setup
        - Sets up initial communication with IoT instance, using the credentials provided by the client (endpoint)
        - Passes clients version control credentials and the client code's start script to our service, 'IoT Deployment Manager' via HTTPS, which is running on the client's IoT instance
        - Once the connection is successfully set up, the 'IoT Deployment Service' will set the state of the client's instance to 'Ready for Deployment'
    - Deployment
        - If the instance is 'Ready for Deployment' the 'IoT Deployment Service' will send a request to the running 'IoT Deployment Manager' to tell it to pull the code and check for updates, if there are updates, the manager will run the client's code with the given start script that was provided by the client
        - The 'IoT Deployment Service' will field a request from the 'IoT Deployment Manager' to notify the client about the state of their deployment (this notification will be passed back to the UI or tyr for client viewing)
- Provides API for endor to acquire feedback about a client's instance
    - Instance State
        - Information about whether the instance is running/in a healthy state
    - Deployment History
        - Information about the deployment history of the instance, this should include, deployment times, commit information (version) about said deployments, user that executed said deployment

Non-Functional Requirements

- Usability
    - The ‘IoT Deployment Service’ API should be easy to use and should be easily extensible if new functionality is needed
- Supportability
    - The ‘IoT Deployment Service’ should be written in NodeJS and should be usable with (supported by) Linux systems
- Reliability
    - The service should be continuously running so that clients can control their instances with complete reliability
    - Uptime should be 99.9% meaning that updates to it must be done in a rolling fashion
- Security
    - The service must communicate with the client’s instance over a secure connection (HTTPS) as sensitive information must be exchanged

Design Specifications: IoT Deployment Manager

Functional Requirements

- Provides API for deployment requests
    - Initial Setup
        - Controller exists to receive a request with the client’s information, including git credentials and start up script for the client’s code
    - Deployment
        - Controller exists to handle requests to kick off deployment
    - Security
        - In order for the API requests to be secure, the Deployment Manager running on the instance needs to be provided the ‘IoT Deployment Service’s IP address so that the manager knows that the request to setup the instance is coming from the correct IP
            - This identity check could also be done using an RSA key pair (this is undecided as of now)

Non-Functional Requirements

- Usability
    - The ‘IoT Deployment Manager’ API should be easy to use and should be easily extensible if new functionality is needed
- Supportability

        - The ‘IoT Deployment Manager’ should be written in NodeJS and should be usable (supported) Linux IoT devices
    - Reliability
        - The service should be continuously running on the IoT device, so that clients can control their instances with complete reliability
        - Uptime should be 99.9% meaning that updates to it must be done in a rolling fashion
    - Security
        - The manager must verify the identity of the service sending the request for deployment to it
            - This could be done through whitelisting or through signing using an RSA key pair

Design Analysis

Our team has created a viable extension to the existing hammer-io system so that clients using the system may deploy their code to their IoT devices in an automated fashion. The extension relies on the hammer-io project existing microservices architecture to be able to leverage easy integration and extensibility.

The fairly straightforward extension that we have proposed should provide the solution to our client’s problem that he is looking for, a way to leverage the hammer-io system to create a deployment pipeline to user’s IoT devices.

Following the previous team’s lead, our extension will provide an easy to use piece of functionality so that user’s of the hammer-io for IoT system will be able to both automate and simplify their deployment at an enterprise level.

Looking forward, we will continue to refine our design, thinking about the nuances of such an extension, hopefully ending next semester with a completely functional, easy to use, service for continuous deployment to IoT devices.

Hardware/Software

Hardware

Raspberry Pi: This will be our Linux-based IoT device for testing

Software

Mocha: Framework used for unit testing
Node: JavaScript runtime for easy building of fast and scalable network applications
Docker: Service that will be used to containerize our services for easy testing and deployment
Bash: Programming language used for IoT start scripts
Firebase: Real time database and backend, synchronize and store client data

Functional Testing

Manual Testing

Much manual testing will have to be done to verify the functionality of our extension, because of the nature of our extension. Such a thing is a bit hard to test using automated testing methods especially for the integration portion of it. We will perform automated regression testing on the existing system to make sure that our newly added pieces to endor, koma, and yggdrasil do not break any functionality, but beyond that we will have to go through the use cases of our extension manually to verify that the functionality of the system is what is expected. This manual testing will involve standing up the hammer-io system, with our extensions active, and performing a deployment onto an IoT device. Of course for convenience sake, this IoT device will simply be a Docker container running Raspbian so that the dev team may have an easily reproducible IoT environment to test with.

Automated Testing

Automated testing for such an extension will be done leveraging Mocha and its unit testing functionality. Such tests will be ran automatically before the new code is pushed to production to verify that the code will function as expected once deployed.

Regression Testing

Regression Testing will be done every time a new build is created to make sure that changes to the code do not break the existing functionality. This testing will be done by executing all unit tests and verifying that they pass.

Non-functional Testing

Much of the non-functional testing will be done manually, as it is difficult to measure many of these qualities with any sort of tool. For example, ensuring ease of use is not really something that we can test with a piece of software.

Looking specifically at 'IoT Deployment Service', it should be fairly simple to manually ensure that our API is easily extensible and easy to utilize. Supportability requirements are similar in this aspect, as it will be fairly telling to us that our systems do not work on the necessary systems should our testing environment stop working properly. In establishing that the service meets our standards for reliability, we will aim to be able to utilize load balancing techniques. Finally, as was previously stated, we will be utilizing HTTPS to guarantee secure data transmission, something that will be a simple process to test for.

Looking at 'IoT Deployment Manager', we see only a few differences. With regard to reliability, we will likely utilize a sort of heartbeat testing to confirm the uptime of the service. Finally we will be able to manually test the effectiveness of our authentication method, whether it be a whitelist or the use of an RSA key pair.

Implementation Issues and Challenges

Extending an already existing system, one that we did not create in the first place, certainly comes with its challenges. First, we had to gain much knowledge about the state of the existing system to even begin to think about how we would implement an extension to it. This learning period took up a sizeable amount of time, as at first we had to meet with the previous contributors on the project to get an idea about its functionality, and we then had to see and discover the functionality that they had mentioned for ourselves. This is because simply gathering information about the existing system did not give us a great enough insight into the state of the project to actually begin designing. That level of insight came when we began to deploy the existing system and play around with its functionality.

Beyond the fact that we are performing an extension to a system that we did not create in the first place, we also had issues coming up with a design for the 'IoT Deployment Service' and 'IoT Deployment Manager' themselves, as they were fairly foreign concepts for most of the team. The biggest issue with designing such an extension is that we have to further prioritize security, more so than when using an existing deployment service such as Heroku or AWS which already guarantees such security. At first our ideas for the extension fell along the lines of a sort of SSH client that the user would be able to access and use to deploy their instances. The issue with this is pretty clear, having SSH access open to the public can lead to a potential breach allowing the malicious user full access to the device. By having our own service run on the instance, the 'IoT Deployment Manager', we can simply create an endpoint that only fields requests to control the instance via a secure connection with the 'IoT Deployment Service', taking away the need to access the full range of commands on the device (which could be done via SSH). This way is also safer for the client as they do not have to pass off SSH credentials to our service to be stored and used whenever deployment activities are executed.

With some minor hiccups along the way, our design did not come easily but we believe that it will be a viable solution to our client's problem.

Conclusion

The extension design that we have laid out above should provide a much needed way for DevOps engineers to deploy their services to the IoT devices of their choosing, while also keeping usability simplistic and reliability and security ensured. Such an extension to the existing DevOps framework, hammer-io, should boost its enterprise compatibility, forcing it into the ever growing IoT space and allowing for our clients leverage it for a solution of their choosing.

Overall, the extension should affect the hammer-io project as whole in a very positive manner, granting new clients and old the ability to automate their IoT service deployment.

References

Github: https://developer.github.com/v4/
Hammer-IO: http://sdmay18-19.sd.ece.iastate.edu/docs/
Heroku: https://devcenter.heroku.com/categories/platform-api/
Mocha: https://mochajs.org/api/mocha/
NodeJS: https://nodejs.org/en/docs/
NPM: https://docs.npmjs.com/
Docker: https://docs.docker.com/