Optimizing a Distributed Home Automation System

CS535 Advanced Operating Systems

Samantha Comeau, Simon Redding, MaryAnn VanValkenburg

Table of Contents

Abstract

In this project, we set out to determine if we could improve the reliability, maintainability, and ease-of-use of a home automation system with minimal increase in network traffic with use of a distributed system manager. A home automation system represents, in a literal and tangible way, a distributed system of machines. Our goal was to evaluate and understand a distributed software tool while improving upon a common household system.

We studied the open-source manager, Nomad. We found that Nomad improved the reliability of the system by detecting node failures and automatically restarting tasks. Nomad also improved the maintainability of the system with job specification files that encapsulated the logic for our automation tasks and allowed for live updates. However, we found both Nomad and its partner program, Consul, difficult to learn and not well supported with documentation and debugging features. While we eventually overcame this difficulty, we do not believe that Nomad improved the ease of use for the system. We conclude that Nomad can improve the reliability and maintainability of the system, but that it is too heavy-duty and difficult to use for a typical home automation system.

Introduction

IOT devices are abundant and diverse in capability. It is quite possible for a user to own many such devices, and all of which can be co-located on the same local area network. This setup can be thought of as a microcosm for Internet-wide distributed systems. Each individual device is simple and relatively cheap, but the system as a whole is capable of performing complex tasks.

Home automation systems are popular projects for system enthusiasts. We found two such enthusiasts who used Docker containers to improve their own home automation systems. Docker for Home Automation [1] mentions using containers to isolate IOT devices and make it easier to scale up the underlying hardware. Docker Home Assistant [2] also found use for containers in his home automation system as a means to apply security patches and be resilient to device failure. We wanted to extend these ideas to optimize network communication and build a more efficient and easy to use system.

Justification of Work

Optimization of microservices and proper use of containers is an area of interest, according to this 2018 review of microservice technology [3]. The following are some of the challenges mentioned in the article:

- Front-end integration: how easy it is to set up and modify services (ease of use)
- Resource monitoring and management: detecting errors that consume resources
- Failure, recovery, repair (security): how fragile the system is, what happens when lines of communication are broken, what happens when devices are compromised (e.g., turned into bots)
- Coordination: how scalable/reusable are the services, how well do different services play with each other (i.e. developed in isolation or developed as a whole)

In our project, we studied these challenges with respect to our use case, the home automation system. The two home automation articles mentioned the difficulties with managing a cluster of devices, especially with respect to propagating security updates. We only discuss security in passing; the broader issues of ease of use and system efficiency are the focus of our study.

System Design

Our system can be thought of in three separate layers. The topmost layer is Nomad [4], an open source container manager and job scheduler that distributes the workload on available nodes in the cluster. Our next layer is the nodes in the cluster. Each node is capable of performing any home automation action by connecting to sensors or actuators and giving them commands. At the lowest layer, we have the sensors and actuators of the system. We used Arduino Unos with Bluetooth modules, each of which is responsible for a single actuator or sensor in our system. We next provide futher detail for each layer.

Nomad

Nomad is similar in function to Kubernetes. We chose Nomad rather than Kubernetes because of its versatility running different applications. In the Nomad job specification, defined below, Nomad can start Docker containers as well as run isolated forks, raw bash scripts, Java, and Qemu. This means that transitioning an existing home automation system is simple as existing executive scripts can be called directly in the job specification. Nomad also has the option to be secured using encryption, TLS, add access control modules, and also supports using namespaces. We felt these would be useful abilities if we wanted to expand our home automation system and properly security it for use.

Nomad uses *job specifications* as building blocks for its services. After registering all nodes in the system as either a client or a server, the user can "run" jobs on the cluster. Nomad and its supporting program, Consul [5], determine which client in the cluster has the least work load and assigns the job to it.

We used two types of job specifications in our cluster: batch and batch/parameterized. Batch jobs are only processed once when run. We used these to run continuously looping scripts that listen to sensors. The benefit of using a batch job instead of a service job, which is designed to run periodically, is that Nomad/Consul will detect if the batch job fails ungracefully and can restart the job on another node. In contrast, batch/parameterized jobs are run once at cluster initiation and then "dispatched" as many times as necessary. We used these to control the actuators in the system once a sensor was triggered.

As mentioned above, Nomad uses a program called Consul. Consul keeps track of the health of nodes and the status of running jobs. While Consul played a lesser role in the design of our project, we will show how Consul drastically increases the network traffic of our cluster.

Raspberry Pi Cluster

We configured the Raspberry Pis using HypriotOS [6], which is a minimal Debian-based operating system optimized to run Docker. We configured the Raspberry Pis in a cluster in an isolated network by using a network switch and ethernet cables so we can easily monitor the network traffic between them. These Pis used Bluetooth to communicate with the Arduinos which controlled the sensors and actuators of our system. For more detailed setup diagrams see Appendix A and for setup instructions see Appendix B.

Sensors and Actuators

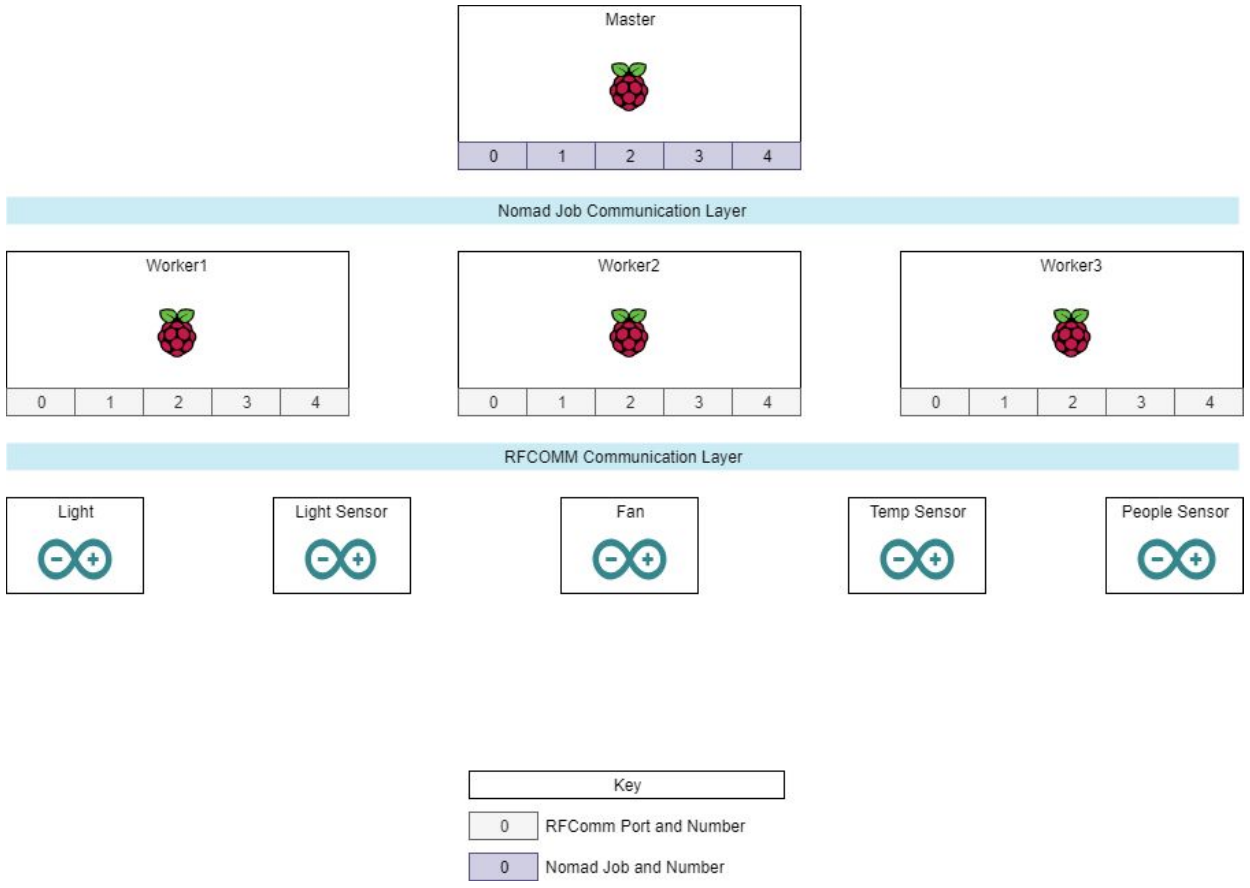
We used five different Arduinos to run three sensors and two actuators. This table shows our Arduinos, their function, and the MAC address of the Bluetooth module through which we communicated with them.

Arduino Name	What it controls	MAC	RFCOMM Port
HC05	LIGHT	98:D3:11:FC:1C:45	0
STEVE	LIGHT SENSOR	98:D3:A1:FD:44:FE	1
LEONARD	FAN	98:D3:91:FD:4B:59	2
TIAN	PEOPLE SENSOR	98:D3:A1:FD:49:8D	3
HC05	TEMP SENSOR	98:D3:C1:FD:41:1E	4

For more information on the Arduinos see Appendix A and C.

Implementation

Our cluster is structured as such:



At startup, three batch jobs are run that continuously listen to sensor data (i.e. light.nomad, people.nomad, and temperature.nomad). These can be thought of as Nomad jobs 0, 1, and 2 (colored purple in the above figure). Each of these batch jobs is assigned to a worker Pi and will run indefinitely on the same worker. At the same time, two batch/parameterized jobs are started (event_handler.nomad and action.nomad), but they are not immediately assigned to a worker. These can be thought of as Nomad jobs 3 and 4.

As an example, imagine that light.nomad is assigned to Worker1. Worker1 pairs with the Light Sensor Arduino (LS) and reads incoming traffic. When Worker1 receives a "1" from LS (meaning that the sensor was triggered by a bright light), Worker1 *dispatches* the event_handler job using the Nomad HTTP API and includes as parameters the identity of the sensor (LS) and the signal ("1"). This creates a "dispatched" instantiation of event_handler.nomad, which the Master Pi will assign to a free worker. Worker2 is assigned the dispatched job, and it subsequently dispatches an instantiation of action.nomad that includes as parameters the actuator to be controlled and the message. In this case, action.nomad will take as parameters "Light" and "0", meaning that the room light should be turned off. Worker3 is assigned the dispatched action.nomad job. It pairs with the Light Arduino (L), sends it the signal, "0", which turns off the room light.

In our setup, we use two batch/parameterized jobs; event_handler.nomad and action.nomad. While these jobs could be combined, we chose to make them separate such that the user could specify different actions to occur as a result of a sensor trigger. In this example, the user could easily add another action, like rolling down the window blinds, simply by updating the control logic in event_handler.nomad. Nomad allows for live updates, so the user could update the event_handler.nomad file (job specification) and re-run it, and Nomad would handle propagating the changes to the cluster.

Methodology

Overview

We started with one actuator connected to the Raspberry Pi cluster. We describe our measurement of the ease of use and efficiency of the home automation system in the proceeding Measure the Problem section. After measuring with one actuator, we added a sensor that could trigger the actuator and re-measured the ease of use and efficiency of the system. We set out to continue adding devices until we reached our maximum of five, but we ran out of time.

Measure the Problem

Throughout our project, we focused on two opposing performance metrics in order to measure the problem and to determine the effectiveness of our solutions. Our metrics of choice were **ease of use** and **efficiency**.

Ease of use is the simplicity and straightforwardness of the system. This is an important metric for a home automation system as there will likely be only one system administrator: the homeowner. In addition, maintenance of the system will, hopefully, only occur so sporadically that the administrator is likely to forget implementation details between maintenance. Thus, the system should not require the knowledge of many details. To measure ease of use, we consider the complexity and number of steps necessary to update the system. A step is defined as either one line typed and run in a terminal, the click-through of a menu, or anything else that can be considered as a single action. Complexity of a step is measured on a scale of 1 to 5, where 1 is advanced system configuration that involves typing carefully crafted lines or knowledge of many system details and 5 is a line typed verbatim or explicitly detailed instructions. We can calculate the ease-of-use of the system by taking the sum of all steps according to complexity (a high score means the system is easy to use).

We define **efficiency** as the extent to which resources are properly used. In particular, we will measure local network traffic between devices and process running time.

We purposefully chose these two performance metrics because they oppose each other. As the system becomes easier to use, we expect that it will generate more network traffic or will require more system overhead. In the other extreme, we expect a highly efficient system to be hard to understand and maintain. Our objective is to find the proper balance between these two metrics that is appropriate for a home automation system.

We gathered our measurements while varying the complexity of our system (according to the number of connected devices) and while using Nomad. Our baseline setup was the Raspberry Pi cluster without Nomad. We then "updated" the system by adding devices to the network and performing synchronized device activities.

Results

Ease of Use

The table below discusses the steps necessary to setup or update the system. The number and complexity of the steps is the reasoning behind the assigned score.

Component	Score	Explanation
Initial Cluster Configuration	5	Configuring the Raspberry Pis into a networked cluster was straightforward. We assigned each Pi a static IP and set our host computer's ethernet port to an IP in the same subnet.
Actuator and Sensor Development	5	Developing programs on the Arduino's to control Bluetooth communication and the different sensors and actuators was straightforward. We used online tutorials and our existing embedded system development knowledge.
Container Development	2	Developing the program to communicate with the actuators and sensors was very difficult. We used a Python script which opened a virtual Bluetooth serial port to the Arduinos. While this script was simple to implement, containerizing the program proved near-impossible due to the complications of Dockers virtual running environment and accessing the device driver for the serial port. We further discuss problems with this component in the Lessons Learned section.
Nomad Configuration	3	Nomad configuration took a significant amount of trial-and-error at first as a result of poor documentation and lack of online

		tutorials. However, after an initial learning curve, it became manageable. The lack of helpful debugging messages continued to present challenges throughout the development process.

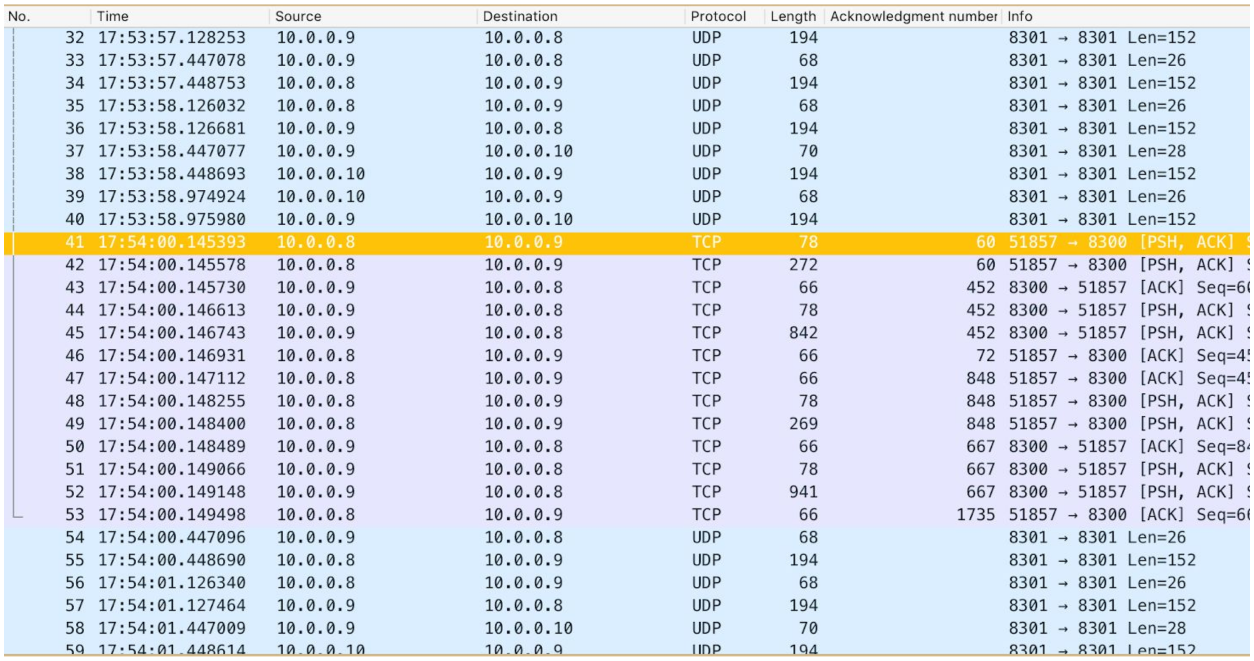
We found the sensors and actuators were easy to develop. However, this was ancillary to the main goal of our project, which was to update an existing home automation system. The cluster was also simple to configure, and the private network in which our Pis communicated was devoid of unnecessary traffic.

We experienced significant difficulty in automating the process of pairing Pis and Arduinos, mostly because Raspberry Pi’s bluetooth control menu is designed to be interactive and not scripted. We found that we needed to “pair” to the Arduinos, but we didn’t need to “connect” to the Arduinos to send and receive messages. The difference in these two verbs is that “pairing” can occur between an Arduino and many devices (we were able to pair to the Arduino from multiple Pis as well as our cell phones at the same time), while “connecting” establishes an exclusive communication between an Arduino and a Pi. Usually, devices need to be connected to communicate over Bluetooth. We do not fully understand why we were able to communicate with the Arduinos without connecting, but it greatly eased our development efforts.

While using the interactive control menu or manually calling our Python script to pair Bluetooth, we had great success in pairing to devices and sending test messages. However, we had intermittent success when we encapsulated the pairing script in a Nomad job or Docker container. Usually, we would get an error signifying that the socket did not exist or that it already existed and could not be re-bound. We do not know why these errors occurred only after encapsulating the script, but it significantly decreased our ease of use.

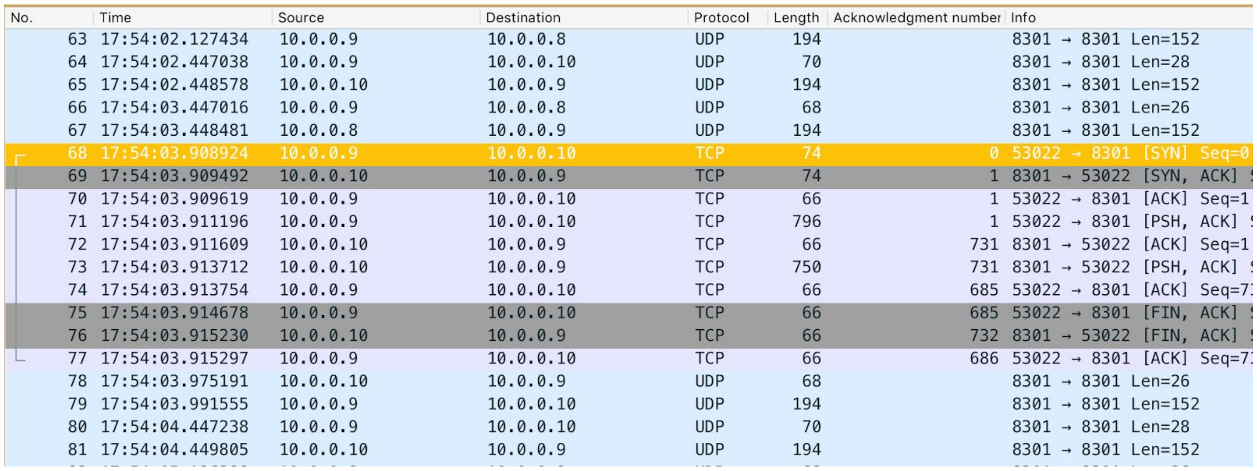
Efficiency

To measure efficiency, we looked at the network trace between nodes in the cluster. The following is a screenshot of such a trace.



No.	Time	Source	Destination	Protocol	Length	Acknowledgment number	Info
32	17:53:57.128253	10.0.0.9	10.0.0.8	UDP	194		8301 → 8301 Len=152
33	17:53:57.447078	10.0.0.9	10.0.0.8	UDP	68		8301 → 8301 Len=26
34	17:53:57.448753	10.0.0.8	10.0.0.9	UDP	194		8301 → 8301 Len=152
35	17:53:58.126032	10.0.0.8	10.0.0.9	UDP	68		8301 → 8301 Len=26
36	17:53:58.126681	10.0.0.9	10.0.0.8	UDP	194		8301 → 8301 Len=152
37	17:53:58.447077	10.0.0.9	10.0.0.10	UDP	70		8301 → 8301 Len=28
38	17:53:58.448693	10.0.0.10	10.0.0.9	UDP	194		8301 → 8301 Len=152
39	17:53:58.974924	10.0.0.10	10.0.0.9	UDP	68		8301 → 8301 Len=26
40	17:53:58.975980	10.0.0.9	10.0.0.10	UDP	194		8301 → 8301 Len=152
41	17:54:00.145393	10.0.0.8	10.0.0.9	TCP	78	60	51857 → 8300 [PSH, ACK]
42	17:54:00.145578	10.0.0.8	10.0.0.9	TCP	272	60	51857 → 8300 [PSH, ACK]
43	17:54:00.145730	10.0.0.9	10.0.0.8	TCP	66	452	8300 → 51857 [ACK] Seq=6
44	17:54:00.146613	10.0.0.9	10.0.0.8	TCP	78	452	8300 → 51857 [PSH, ACK]
45	17:54:00.146743	10.0.0.9	10.0.0.8	TCP	842	452	8300 → 51857 [PSH, ACK]
46	17:54:00.146931	10.0.0.8	10.0.0.9	TCP	66	72	51857 → 8300 [ACK] Seq=4
47	17:54:00.147112	10.0.0.8	10.0.0.9	TCP	66	848	51857 → 8300 [ACK] Seq=4
48	17:54:00.148255	10.0.0.8	10.0.0.9	TCP	78	848	51857 → 8300 [PSH, ACK]
49	17:54:00.148400	10.0.0.8	10.0.0.9	TCP	269	848	51857 → 8300 [PSH, ACK]
50	17:54:00.148489	10.0.0.9	10.0.0.8	TCP	66	667	8300 → 51857 [ACK] Seq=8
51	17:54:00.149066	10.0.0.9	10.0.0.8	TCP	78	667	8300 → 51857 [PSH, ACK]
52	17:54:00.149148	10.0.0.9	10.0.0.8	TCP	941	667	8300 → 51857 [PSH, ACK]
53	17:54:00.149498	10.0.0.8	10.0.0.9	TCP	66	1735	51857 → 8300 [ACK] Seq=6
54	17:54:00.447096	10.0.0.9	10.0.0.8	UDP	68		8301 → 8301 Len=26
55	17:54:00.448690	10.0.0.8	10.0.0.9	UDP	194		8301 → 8301 Len=152
56	17:54:01.126340	10.0.0.8	10.0.0.9	UDP	68		8301 → 8301 Len=26
57	17:54:01.127464	10.0.0.9	10.0.0.8	UDP	194		8301 → 8301 Len=152
58	17:54:01.447009	10.0.0.9	10.0.0.10	UDP	70		8301 → 8301 Len=28
59	17:54:01.448614	10.0.0.10	10.0.0.9	UDP	194		8301 → 8301 Len=152

The UDP packets, shown in blue, we believe are health checks. These are sent between the cluster server (10.0.0.9) and two clients (10.0.0.8 and 10.0.0.10). (Note: 10.0.0.11 died unexpectedly in the middle of our project, so we ended the project with three nodes in our cluster). These health checks by default occur every second, though this is likely configurable. The TCP packets shown in the middle of the screenshot are from dispatching a Nomad job. Client node 10.0.0.8, who is listening for signals from the light sensor, initiates contact with Server node 10.0.0.9 via event_handler.nomad when the light sensor is triggered.



No.	Time	Source	Destination	Protocol	Length	Acknowledgment number	Info
63	17:54:02.127434	10.0.0.9	10.0.0.8	UDP	194		8301 → 8301 Len=152
64	17:54:02.447038	10.0.0.9	10.0.0.10	UDP	70		8301 → 8301 Len=28
65	17:54:02.448578	10.0.0.10	10.0.0.9	UDP	194		8301 → 8301 Len=152
66	17:54:03.447016	10.0.0.9	10.0.0.8	UDP	68		8301 → 8301 Len=26
67	17:54:03.448481	10.0.0.8	10.0.0.9	UDP	194		8301 → 8301 Len=152
68	17:54:03.908924	10.0.0.9	10.0.0.10	TCP	74	0	53022 → 8301 [SYN] Seq=0
69	17:54:03.909492	10.0.0.10	10.0.0.9	TCP	74	1	8301 → 53022 [SYN, ACK]
70	17:54:03.909619	10.0.0.9	10.0.0.10	TCP	66	1	53022 → 8301 [ACK] Seq=1
71	17:54:03.911196	10.0.0.9	10.0.0.10	TCP	796	1	53022 → 8301 [PSH, ACK]
72	17:54:03.911609	10.0.0.10	10.0.0.9	TCP	66	731	8301 → 53022 [ACK] Seq=1
73	17:54:03.913712	10.0.0.10	10.0.0.9	TCP	750	731	8301 → 53022 [PSH, ACK]
74	17:54:03.913754	10.0.0.9	10.0.0.10	TCP	66	685	53022 → 8301 [ACK] Seq=7
75	17:54:03.914678	10.0.0.9	10.0.0.10	TCP	66	685	53022 → 8301 [FIN, ACK]
76	17:54:03.915230	10.0.0.10	10.0.0.9	TCP	66	732	8301 → 53022 [FIN, ACK]
77	17:54:03.915297	10.0.0.9	10.0.0.10	TCP	66	686	53022 → 8301 [ACK] Seq=7
78	17:54:03.975191	10.0.0.10	10.0.0.9	UDP	68		8301 → 8301 Len=26
79	17:54:03.991555	10.0.0.9	10.0.0.10	UDP	194		8301 → 8301 Len=152
80	17:54:04.447238	10.0.0.9	10.0.0.10	UDP	70		8301 → 8301 Len=28
81	17:54:04.449805	10.0.0.10	10.0.0.9	UDP	194		8301 → 8301 Len=152

In this second screenshot, we see that, three seconds after the end of the last TCP traffic, Server node 10.0.0.9 contacts Client node 10.0.0.10 via action.nomad. This communication tells Client node 10.0.0.10 to pair with the light Arduino and to turn on the light.

This trace contains two complete actions; the first turns off the light when the light sensor gets bright light, the second turns the light back on when the light sensor goes dark. The first of these actions took two seconds from the start of the first TCP packet to the last. In reality, it took about four seconds from exposing the light sensor to the LED turning off. The second action took about four seconds from the start of the first TCP packet to the last. It took about eight seconds for the action to take physical effect. The extra delay in the system likely has more to do with pairing with the Bluetooth and for the light sensor to regain resistance after losing light exposure than with Nomad.

The total number of packets between the first and last TCP transmissions in this trace were 20 and 36, respectively. This factors to approximately 10 packets per second, half of which were health checks and half of which were action-related. While this is not as efficient as a single TCP packet for an action, it is well within the networking capabilities of Ethernet. According to this Raspberry Pi benchmark, our Pis have LAN bottleneck speeds of about 11 MBps [7]. Assuming an average packet size of 2000 bits (in our capture, packets were ~500 bits, ~1500 bits, or ~6000 bits), our cluster is capable of handling up to 5,500 packets per second.

Discussion

Lessons Learned

We found this project to be nearly impossible to do properly because of the Bluetooth modules. Docker containers have a difficult time routing to the serial ports for the RF communication from the Raspberry Pi's to the Arduino controllers. It was much easier to write to the devices than it was to read from them. In the future, we would recommend to only attempt to containerize modules that communicate to Bluetooth devices, but do not read from them.

Summary

- Our system uses tools designed for massive networks with high failure rates
- Nomad is not best suited for home automation (although it does offer some nice advantages)
    - Mainly, should a job fail (bluetooth connection fails, etc), Nomad will automatically detect this and schedule a rerun of the job
- To expand on this project, a similar setup could be implemented on a much wider scale. The larger the network, the more cause there is for a system like ours.

References

Our source code can be found here:
https://github.com/sjcomeau43543/ContainerizedHomeAuto.git

[1] R. Stroffolino, “Docker for Home Automation,” Gestalt IT, 13-Feb-2018. [Online]. Available: https://gestaltit.com/favorites/rich/docker-home-automation/.

[2] J. Vorst. “Home Assistant + Docker: The Perfect Home Automation Match,” JosiahVorst, May-2018. [Online]. Available: https://josiahvorst.com/home-assistant-docker-the-perfect-home-automation-match/.

[3] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. IEEE Software, 35(3), 24-35.

[4] “Nomad,” HashiCorp. [Online]. Available: https://www.nomadproject.io/.

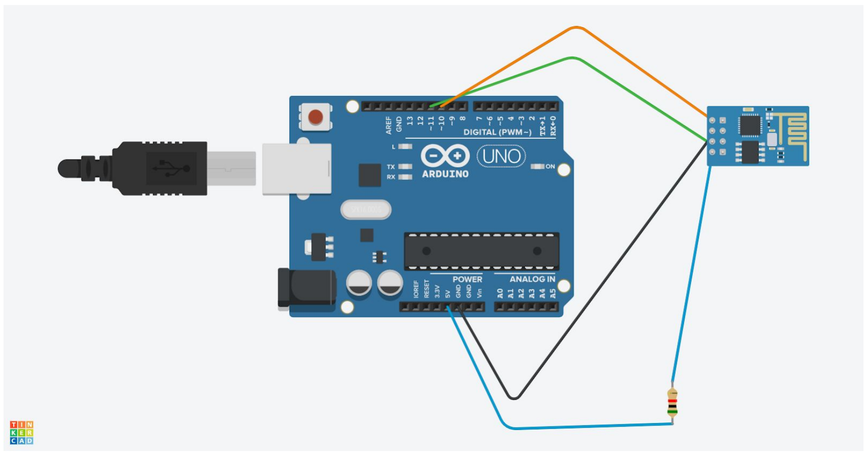
[5] “Consul,” HashiCorp. [Online]. Available: https://www.consul.io/

[6] “Hypriot,” [Online]. Available: https://blog.hypriot.com/

[7] J. Geerling, “Networking Benchmarks.” [Online]. Available: https://www.pidramble.com/wiki/benchmarks/networking

Appendix A: Design Diagrams

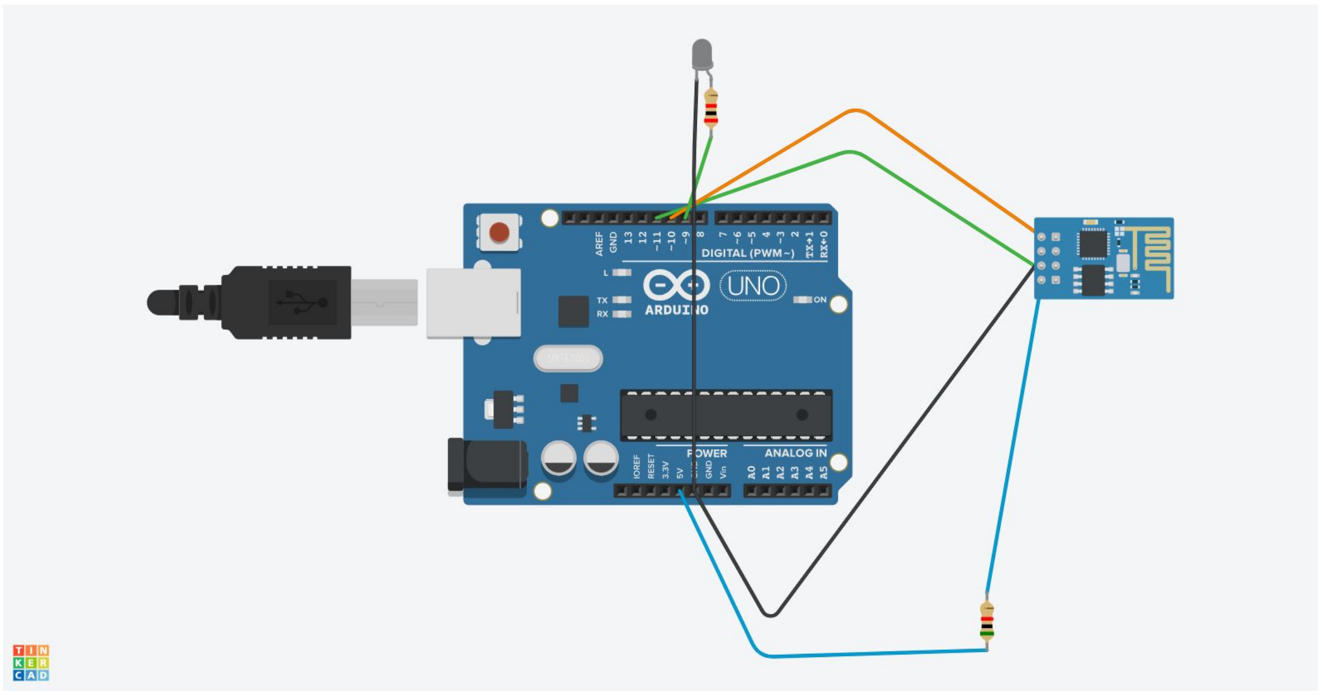
Bluetooth



```
#include <SoftwareSerial.h>

SoftwareSerial EEBlue(10, 11); // RX | TX
int EEBlue_data;
int Serial_data;

void setup() {
  Serial.begin(9600);
  EEBlue.begin(9600);  //Default Baud for comm, it may be different for your
Module.
  Serial.println("The bluetooth gates are open.\n Connect to HC-05 from any
other bluetooth device with 1234 as pairing key!.");
}

void loop(){
  // Feed any data from bluetooth to Terminal.
  if (EEBlue.available()){
    EEBlue_data = EEBlue.read();
    Serial.write(EEBlue_data);
  }

  // Feed all data from terminal to bluetooth
  if (Serial.available()){
    Serial_data = Serial.read();
    EEBlue.write(Serial_data);
  }
}
```

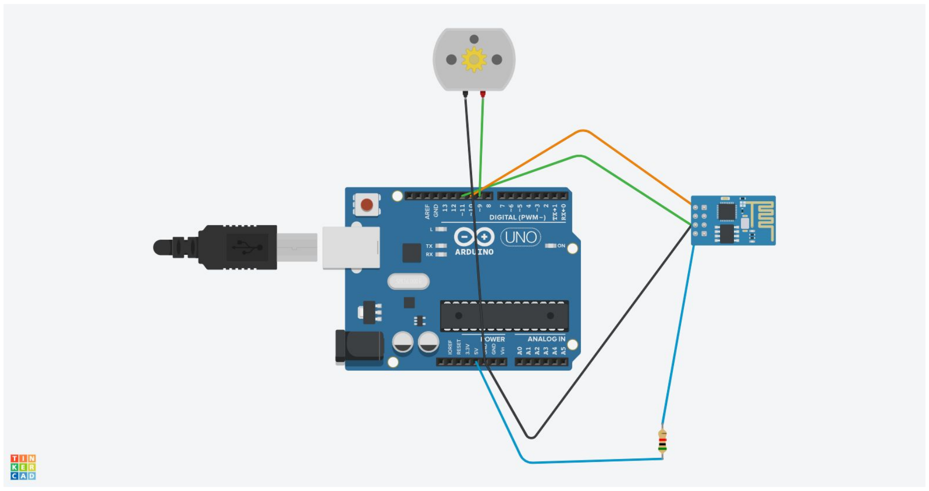
Actuators

Lights



```
#include <SoftwareSerial.h>

SoftwareSerial EEBlue(10, 11); // RX | TX
int ledPin = 9;
int EEBlue_data;
int Serial_data;

void setup() {
  Serial.begin(9600);
  EEBlue.begin(9600);  //Default Baud for comm, it may be different for your
Module.
  Serial.println("The bluetooth gates are open.\n Connect to HC-05 from any
other bluetooth device with 1234 as pairing key!.");
  pinMode(ledPin, OUTPUT);
}

void loop(){
  // Feed any data from bluetooth to Terminal.
  if (EEBlue.available()){
    EEBlue_data = EEBlue.read();
    Serial.println(EEBlue_data);
    if (EEBlue_data == '0'){
      EEBlue.println("SUCCESS : Turning the LED off");
      digitalWrite(ledPin, 0);
    } else if (EEBlue_data == '1'){
      EEBlue.println("SUCCESS : Turning the LED on");
      digitalWrite(ledPin, 1);
    }
  }

  // Feed all data from terminal to bluetooth
  if (Serial.available()){
    Serial.println("Got Serial Data");
    Serial_data = Serial.read();
    EEBlue.println(Serial_data);
  }
}
```

Fan

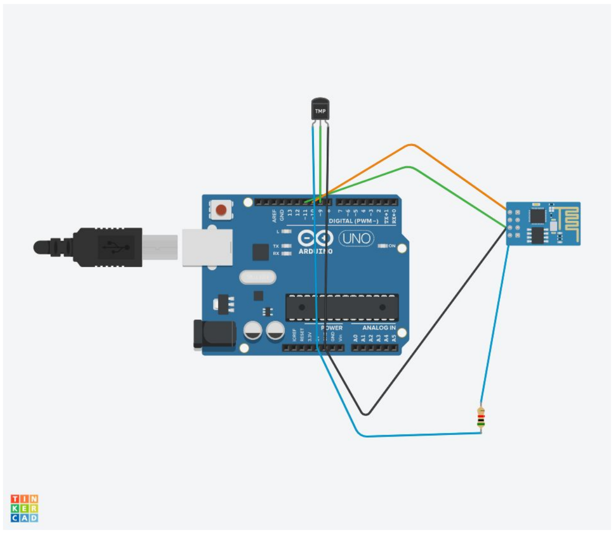


```
#include <SoftwareSerial.h>

SoftwareSerial EEBlue(10, 11); // RX | TX
int EEBlue_data;
int Serial_data;

int motorPin = 9;

void setup() {
  Serial.begin(9600);
  EEBlue.begin(9600);  //Default Baud for comm, it may be different for your
Module.
  Serial.println("The bluetooth gates are open.\n Connect to HC-05 from any
other bluetooth device with 1234 as pairing key!.");

```

```
  pinMode(motorPin, OUTPUT);
}

void loop(){
  // Feed any data from bluetooth to Terminal.
  if (EEBlue.available()){
    EEBlue_data = EEBlue.read();
    Serial.write(EEBlue_data);

    if (EEBlue_data == '0'){
      EEBlue.println("SUCCESS : Turning the fan off");
      digitalWrite(motorPin, LOW);
    } else if (EEBlue_data == '1'){
      EEBlue.println("SUCCESS : Turning the fan on");
      digitalWrite(motorPin, HIGH);
    }
  }

  // Feed all data from terminal to bluetooth
  if (Serial.available()){
    Serial_data = Serial.read();
    EEBlue.write(Serial_data);
  }
}
```

Sensors

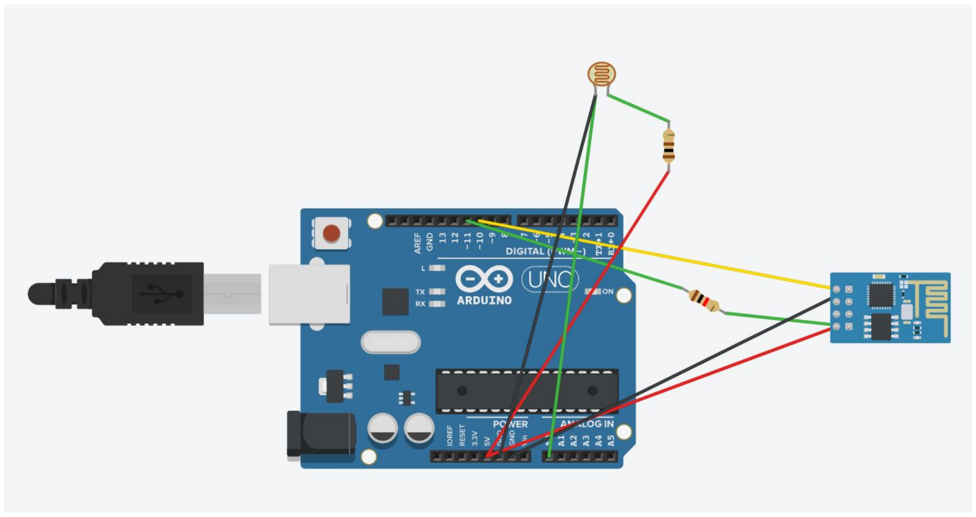
Temperature / Humidity Sensor



```
#include <SoftwareSerial.h>
#include <Adafruit_Sensor.h>
```

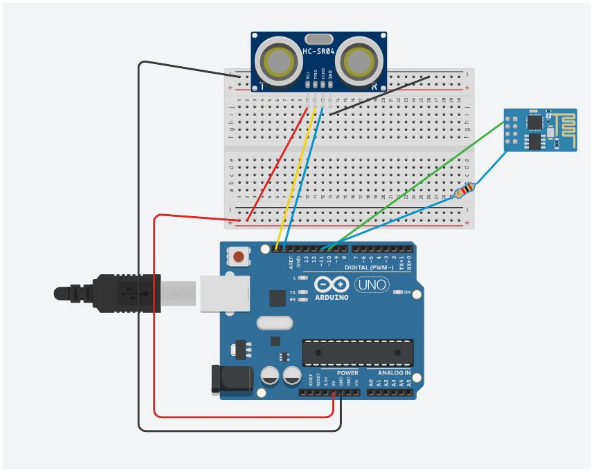
```
#include <DHT.h>

SoftwareSerial EEBlue(10, 11); // RX | TX
int EEBlue_data;
int Serial_data;

int sensorPin = 9;
DHT dht(sensorPin, DHT22);
float humidity;
float temperature;

void setup() {
  Serial.begin(9600);
  EEBlue.begin(9600);  //Default Baud for comm, it may be different for your
Module.
  Serial.println("The bluetooth gates are open.\n Connect to HC-05 from any
other bluetooth device with 1234 as pairing key!.");

  dht.begin();
  //pinMode(sensorPin, INPUT);
}

void loop(){
  // Read the temperature from the sensor
  delay(2000);
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

  Serial.print("Humidity:");
  Serial.print(humidity);
  Serial.print("%");
  Serial.print("Temperature:");
  Serial.print(temperature);
  Serial.print("degrees Celsius, ");
  Serial.print(temperature * 9 / 5 + 32);
  Serial.println("degrees Farenheit");
  delay(10000);

  // Feed any data from bluetooth to Terminal.
  if (EEBlue.available()){
    EEBlue_data = EEBlue.read();
    Serial.write(EEBlue_data);
  }

  // Feed all data from terminal to bluetooth
  if (Serial.available()){
    Serial_data = Serial.read();
    EEBlue.write(Serial_data);
  }
}
```

Environment Brightness



```
#include <SoftwareSerial.h>
SoftwareSerial EEBlue(10, 11); // RX | TX

int sensorPin = A0;    // select the input pin for the potentiometer
int sensorValue = 0;  // variable to store the value coming from the sensor

const int numReadings = 10;
const int lightCutoffVal = 30;

int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average

int lastAverage = 0;

void setup() {
  Serial.begin(9600);
  EEBlue.begin(9600);

  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }
}

void loop() {
  // subtract the last reading:
  total = total - readings[readIndex];
  // read from the sensor:
  readings[readIndex] = analogRead(sensorPin);
  // add the reading to the total:
  total = total + readings[readIndex];
  // advance to the next position in the array:
  readIndex = readIndex + 1;
  // if we're at the end of the array...
  if (readIndex >= numReadings) {
    // ...wrap around to the beginning:
    readIndex = 0;
  }

  lastAverage = average;
  // calculate the average:
  average = total / numReadings;

  // send it to the computer as ASCII digits
  Serial.println(average);
  if (average > lightCutoffVal && lastAverage <= lightCutoffVal) {
    EEBlue.write("1");
  }
  else if (average < lightCutoffVal && lastAverage >= lightCutoffVal) {
    EEBlue.write("0");
  }

  delay(500);
}
```

Thermal Camera



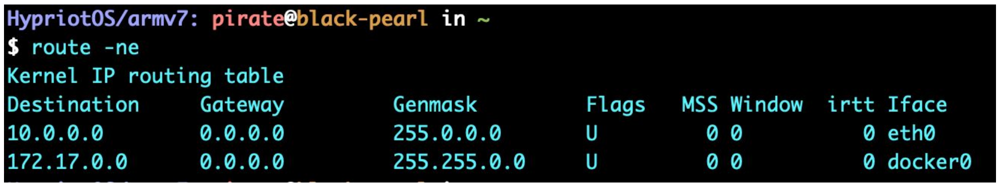
```
#include <Wire.h>
#include <Adafruit_AMG88xx.h>
#include "SoftwareSerial.h"

SoftwareSerial EEBlue(10, 11); // RX | TX

Adafruit_AMG88xx amg;

float pixels[AMG88xx_PIXEL_ARRAY_SIZE];

int count = 0;
```

```
bool lastTickPositiveSide = false;
bool lastTickNegativeSide = false;
int humanTempCutoff = 25;

void setup() {
    Serial.begin(9600);
    EEBlue.begin(9600);
    Serial.println(F("AMG88xx pixels"));

    bool status;

    // default settings
    status = amg.begin();
    if (!status) {
        Serial.println("Could not find a valid AMG88xx sensor, check
wiring!");
        while (1);
    }

    Serial.println("-- Pixels Test --");

    delay(100); // let sensor boot up
}


void loop() {
    //read all the pixels
    amg.readPixels(pixels);

    bool positiveSide = false;
    bool negativeSide = false;
    Serial.print("[");
    for(int i=1; i<=AMG88xx_PIXEL_ARRAY_SIZE; i++){
      if (pixels[i-1] > humanTempCutoff) {
        Serial.print("XXX");
        if( i%8 == 0 || (i+1)%8 == 0 || (i+2)%8 == 0 || (i+3)%8 == 0 ) //
"positive" side
          positiveSide = true;
        if( (i+4)%8 == 0 || (i+5)%8 == 0 || (i+6)%8 == 0 || (i+7)%8 == 0 ) //
"positive" side
          negativeSide = true;
      }
      else Serial.print("   ");
      if( i%8 == 0 ) Serial.print("|\n|");
    }
    Serial.println("]");
    Serial.println();

    // if they're all the same, don't bother
    if (!((lastTickNegativeSide && lastTickPositiveSide && positiveSide &&
negativeSide) ||
          (!lastTickNegativeSide && !lastTickPositiveSide && !positiveSide &&
!negativeSide))) {
    if (lastTickNegativeSide && positiveSide)
      moveNegativeToPositive();
    if (lastTickPositiveSide && negativeSide)
      movePositiveToNegative();
    lastTickPositiveSide = positiveSide;
    lastTickNegativeSide = negativeSide;
 }

  char buff[30];
  sprintf(buff, "Positive Side: %d", lastTickPositiveSide);
  Serial.println(buff);
  sprintf(buff, "negative Side: %d", lastTickNegativeSide);
  Serial.println(buff);

  sprintf(buff, "There are %i people in the room.", count);
  Serial.println(buff);

  //delay a second
  delay(100);
}

void moveNegativeToPositive() {
  count ++;
  char buff[30];
  sprintf(buff, "There are %i people in the room.\n", count);
  EEBlue.write(buff);
  Serial.print(buff);
}
void movePositiveToNegative() {
  count --;
  char buff[30];
  sprintf(buff, "There are %i people in the room.\n", count);
  EEBlue.write(buff);
  Serial.print(buff);
}
```

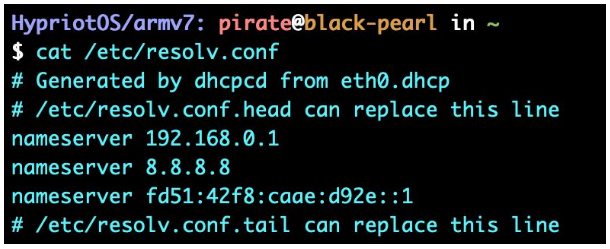
Appendix B: Setting up the System

Raspberry Pi

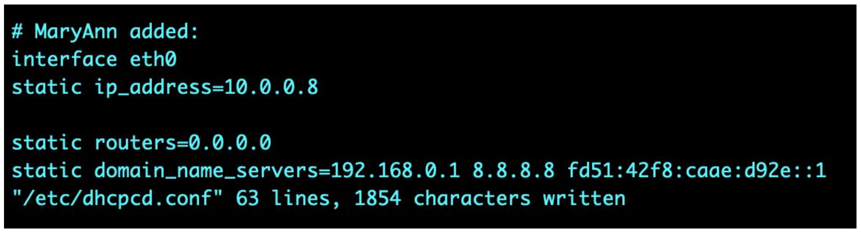
Setup the Operating System

- Downloaded image from https://blog.hypriot.com/downloads/
- Verified checksum with:
    - `shasum -a 256 hypriotos-rpi-v1.10.0.img.zip`
- Flash SD card
    - `unzip hypriotos-rpi-v1.10.0.img.zip`
    - `diskutil list` (lists all disk images. Looking for the one that's the SD card)
    - `diskutil unmountDisk /dev/disk4` (Have to unmount it before I can flash the image to it)
    - `sudo -s dd if=~/Desktop/hypriotos-rpi-v1.10.0.img | pv | sudo dd of=/dev/disk4 bs=1m` (The image is approx 1.05 GB. pv shows the progress. bs=1m is the block size. Time to copy: 7 mins 9 sec. Can try speeding this up by making the block size larger)
        - Setting bs=10m (MB?) took 7 mins 6 sec
        - Setting bs=4 (bytes?) took 24 mins 38 sec
        - Setting bs=1k took 9 mins 41 sec
    - Note: Once we configure one Pi, we can just copy the image from one SD card to the other. However, this means we will be copying an unpacked image (fills the 32 GB flash drive) instead of the packed image (approx 1.05 GB), so it will take much longer. Instead, I'll manually configure each Pi for now. This is something we will hope to fix with containers/container manager.
- Change security settings (default UN/PW)
- Connecting to Pi via Ethernet (no router) (https://youtu.be/5DCPDQnRXm8)
    - Set Ethernet address on computer to be between 10.0.0.100 and 10.0.0.200
    - On Pi, check gateway and get nameservers
        - 

```
HypriotOS/armv7: pirate@black-pearl in ~
$ route -ne
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        0.0.0.0         255.0.0.0       U         0 0          0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U         0 0          0 docker0
```

- 

```
HypriotOS/armv7: pirate@black-pearl in ~
$ cat /etc/resolv.conf
# Generated by dhcpcd from eth0.dhcp
# /etc/resolv.conf.head can replace this line
nameserver 192.168.0.1
nameserver 8.8.8.8
nameserver fd51:42f8:caae:d92e::1
# /etc/resolv.conf.tail can replace this line
```

- Add to the bottom of /etc/dhcpcd.conf
  - `sudo vi /etc/dhcpcd.conf`
  - 

```
# MaryAnn added:
interface eth0
static ip_address=10.0.0.8

static routers=0.0.0.0
static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e::1
"/etc/dhcpcd.conf" 63 lines, 1854 characters written
```

  - “Static routers” is the default gateway for eth0, found from running `route -ne`. “Static domain_name_servers” is the nameservers found in `/etc/resolv.conf`, each separated by a space.
- Reboot

Setup the Docker Image

- Ensure the Pi is connected to the internet
  - Settings are in /etc/wpa_applicant/wpa_applicant.conf
  - On change - reboot the Pi
- Clone the repo
- To build
  - Navigate to Docker/IMAGE
  - `Sudo docker build -t serial-bluetooth .`
- To run
  - `Sudo docker run --privileged -e MAC = “MAC” -e RW = “read” -e PORT = “0” serial-bluetooth`
  - `Sudo docker run --privileged -e MAC = “MAC” -e RW = “write” -e MESSAGE = “0” serial-bluetooth`

Arduino

Dependencies

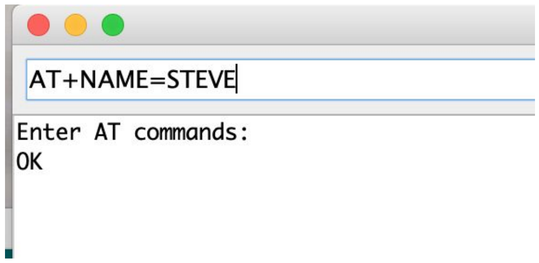
- https://github.com/sjcomeau/ContainerizedHomeAuto
- Arduino IDE

Functionality

- Setup the Arduino according to the corresponding schematic diagrams that are posted in the github
- Flash the Arduino using the IDE

Settings

- These instructions were adapted from this instructable: https://www.instructables.com/id/Modify-The-HC-05-Bluetooth-Module-Defaults-Using-A/
- Wire the Bluetooth module to the Arduino as specified in the "Arduino Bluetooth Setup" document.
- Add an additional wire connecting EN pin on the Bluetooth module to ~9 on the Arduino.
- Flash the Arduino with the "boot_at_mode.ino" file, located in the repo at IoTDevices/Bluetooth/boot_at_mode/
- From the Arduino Serial Monitor, type "AT+NAME=X" where X is the new name



```
AT+NAME=STEVE
```

```
Enter AT commands:
OK
```

(OK appears when you run AT+NAME=X)

- The Bluetooth module has been renamed. To return to normal Bluetooth:
    - Remove the EN wire (green in the picture)
    - Flash the Arduino with "bluetooth_default.ino" or whichever file.

Appendix C: Arduino Controllers

Actuators	Options	General Idea
Lights	On, Off	Triggered by the user. A bright LED will mimic a light bulb in this situation. Can automatically turn on if the motion sensor detects someone in the room.
Fan	On, Off	Triggered if the temperature from another sensor module reaches a certain threshold. Only turns on if the motion sensor detects someone in the room.

Sensors	General Idea
Motion	IR sensor array by the doorway, counts the number of people in the room (perhaps two sets so we can determine directionality into or out of the room)
Light	Looking out the window to gauge sunlight (variable resistor for intensity, one with a software interface already)
Temperature / Humidity	A temperature sensor somewhere in the room that measures the room's internal temperature.