**DOMAIN NAME: CLOUD APPLICATION DEVELOPMENT**

**PROJECT NAME: TRANSFORM YOUR HOME INTO A SMART**

**LIVING SPACE USING IBM CLOUD FUNCTION FOR IOT DATA PROCESSING.**

**Phase 4 Submission Document**

Over the past few years, we’ve seen a significant rise in popularity for intelligent personal assistants, such as Apple’s Siri, Amazon Alexa, and Google Assistant. Though they initially appeared to be little more than a novelty, they’ve evolved to become rather useful as a convenient interface to interact with service APIs and IoT connected devices.

This series will guide users through setting up their own starter home automation hub using a Raspberry PI. This first blog post provides a step by step tutorial to create a RF circuit that’ll enable the Raspberry PI to turn power outlets off and on. Once the circuit and software dependencies are installed and configured properly, users will also be able to leverage Watson’s language services to control the power outlets via voice and/or text commands. Furthermore, we’ll show how Openwhisk serverless functions can be leveraged to trigger these sockets based on a timed schedule, changes to the weather, motion sensors being activated, etc. We’ll assume that the reader has a basic understanding of Linux and electronic circuits.

**Steps:**

* Connect And Configure Hardware
* Assemble RF Circuit
* Install Software Dependencies + Libraries
* Capture RF codes corresponding to wireless sockets
* Provision IBM Cloud Services
* Create Serverless Functions
* Deploy to IBM Cloud

This tutorial requires the following components:

* [*Raspberry PI3*](https://www.raspberrypi.org/products/raspberry-pi-3-model-b/)
* [*Etekcity 433 MHz Outlets*](https://www.amazon.com/Etekcity%C2%AE-Wireless-Control-Switches-included/dp/B00DQELHAE/)
* *USB Microphone*
* [*GPIO Ribbon cable + Breakout Board*](https://www.adafruit.com/product/914)
* [*433MHz RF transmitter and receiver*](https://www.amazon.com/SMAKN%C2%AE-433Mhz-Transmitter-Receiver-Arduino/dp/B00M2CUALS)
* [*Electronic Breadboard*](https://www.adafruit.com/categories/124)

**Install software dependencies:**

Login to Raspberry PI and install prerequisites for the [wiringPi](http://wiringpi.com/" \t "_blank) library. This library enables applications to read/control the Raspberry Pi’s GPIO pins.

sudo apt-get -y update

sudo apt-get -y install git-core

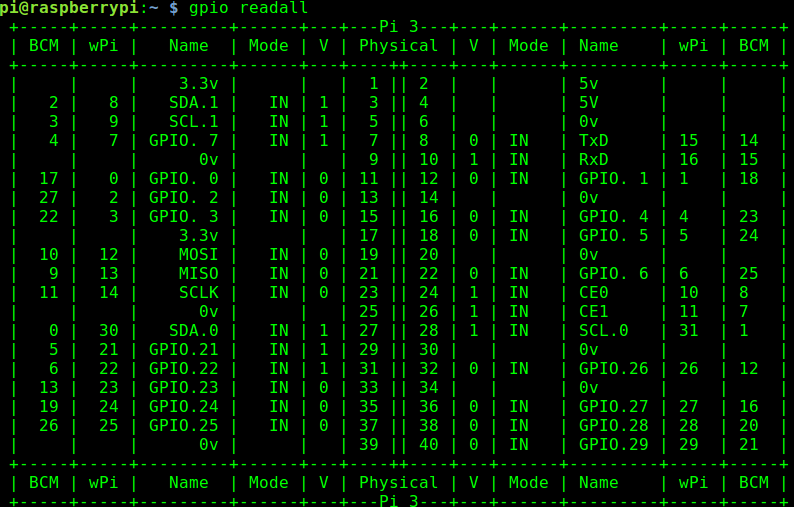
git clone git://git.drogon.net/wiringPi

git pull origin master

./wiringPi/build

Ensure the wiringPi library is installed properly by running the following command.

gpio readall



Pi GPIO output

Next, install 433Utils, which will call the wiringPi library to transmit and receive messages via the 433MHz frequency. In our case, each outlet has a unique RF code to turn power on and off. We’ll use one of the wiringPi utilities, titled “RFSniffer” to essentially register each of these unique codes. The 433MHz frequency is standard among many common devices such as garage door openers, thermostats, window/door sensors, car keys, etc. So this initial setup is not limited to only controlling power outlets. This library can be installed by running the following commands on the Raspberry Pi.

sudo apt-get install build-essential

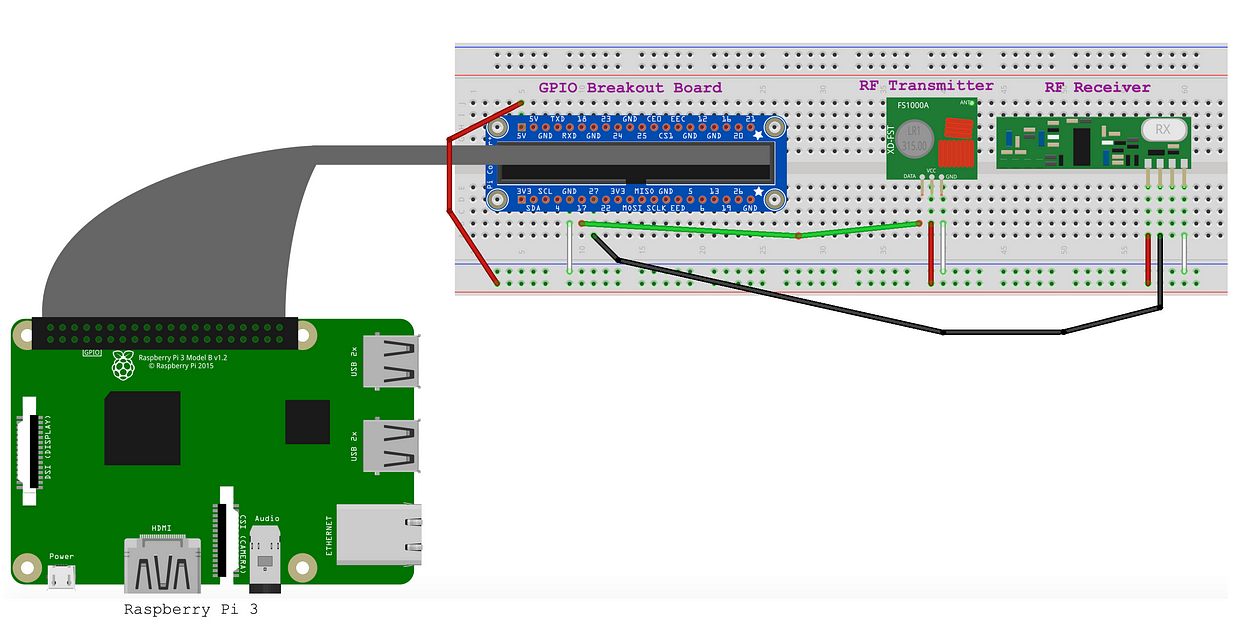
git clone git://github.com/ninjablocks/433Utils.git

cd 433Utils/RPi\_utils

make

# Setup RF Circuit:

Arrange the hardware components to complete the following circuit.



Now we will determine which RF codes correspond with the Etekcity outlets. Start by running.

sudo /var/www/rfoutlet/RFSniffer

This will listen on the RF receiver for incoming signals, and write them to stdout. As the on/off buttons are pressed on the Etekcity remote, the Raspberry Pi should show the following output if the circuit is wired correctly.

pi@raspberrypi:~ $ sudo /var/www/rfoutlet/RFSniffer

Received 5528835

Received pulse 190

Received 5528844

Received pulse 191

After determining the on/off signal for the RF sockets, place the captured signals into the /etc/environment file like so.

RF\_PLUG\_ON\_1=5528835

RF\_PLUG\_ON\_PULSE\_1=190

RF\_PLUG\_OFF\_1=5528844

RF\_PLUG\_OFF\_PULSE\_1=191

Now, plug in the associated socket, and run the following command to ensure the Raspberry Pi can turn the socket on and off. This command simply sends the RF code at the requested pulse length, which is to be provided as the -l parameter.

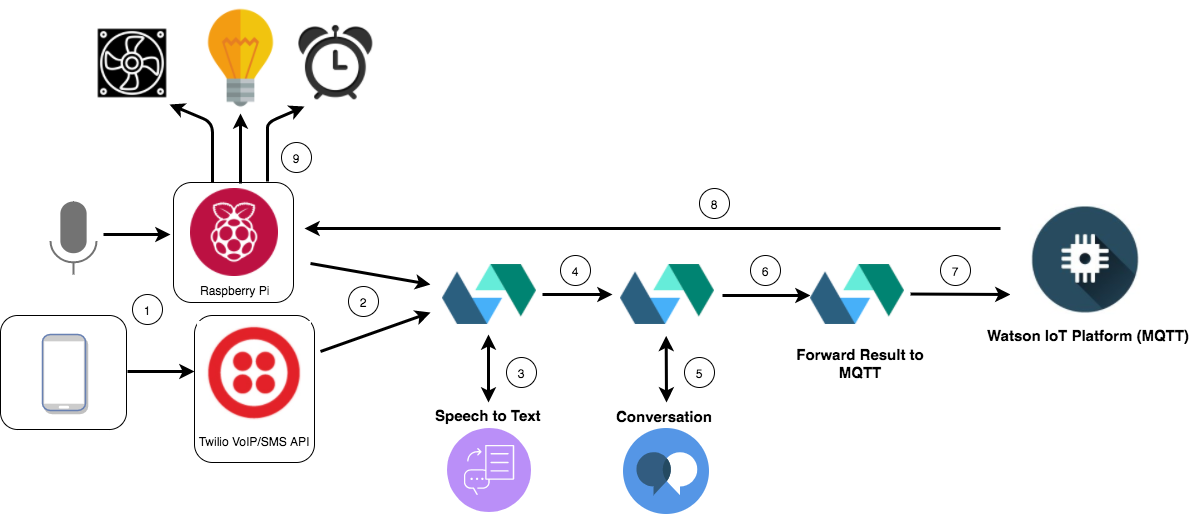
/var/www/rfoutlet/codesend ${RF\_PLUG\_ON\_1} -l ${RF\_PLUG\_ON\_PULSE\_1}

/var/www/rfoutlet/codesend ${RF\_PLUG\_OFF\_1} -l ${RF\_PLUG\_OFF\_PULSE\_1}

Now that we can control the sockets manually via cli, we’ll move forward and experiment with different ways to control them in an automated fashion. Rather than writing and executing pipelines and complex automation logic on the Raspberry Pi, we’ll utilize a serverless, event driven platform called Openwhisk. In this implementation, Openwhisk actions communicate with the Raspberry Pi via MQTT messages.

A [Bluemix](https://console.ng.bluemix.net/) account is required to set up Openwhisk and the accompanying Watson services.

**Architecture Diagram:**



Architecture flow

1. User says a command into the microphone, or sends a text to the Twilio SMS number
2. User input is captured and embedded in an HTTP POST request triggering an IBM Cloud Functions sequence
3. The first IBM Cloud Functions action in the sequence forwards the audio to Speech to Text service, and waits for the response
4. Transcription is forwarded to the second IBM Cloud Functions action
5. IBM Cloud Functions action 2 calls the Conversation service to analyze the user’s text input, again waits for the response
6. Conversation service result is forwarded to final IBM Cloud Functions action
7. Final IBM Cloud Functions action publishes a entity/intent pair (fan/turnon for example) to the IoT MQTT broker
8. MQTT client subscribed on Raspberry Pi receives and interprets result
9. Raspberry Pi transmits corresponding RF signal to adjust outlet state