PHASE 3

**Natural Language Interface based Home Automation**

Watson Conversation is now Watson Assistant. Although some images in this code pattern may show the service as Watson Conversation, the steps and processes will still work.

**Overview and goal**

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 developer pattern will guide users through setting up their own starter home automation hub by using a 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 IBM Cloud Functions serverless functions can be leveraged to trigger these sockets based on a timed schedule, changes to the weather, motion sensors being activated, etc.

**Audience level:**

Intermediate. User will need basic hardware skills to assemble electronic circuits on a breadboard, and be somewhat familiar with a Linux terminal to install Raspberry Pi dependencies.

**IBM Cloud Plans:**

This project will work with the Free/Lite version of all required services: Speech To Text, Text To Speech, Watson Assistant, and IBM Cloud Functions.

**Architecture**

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

**Steps**

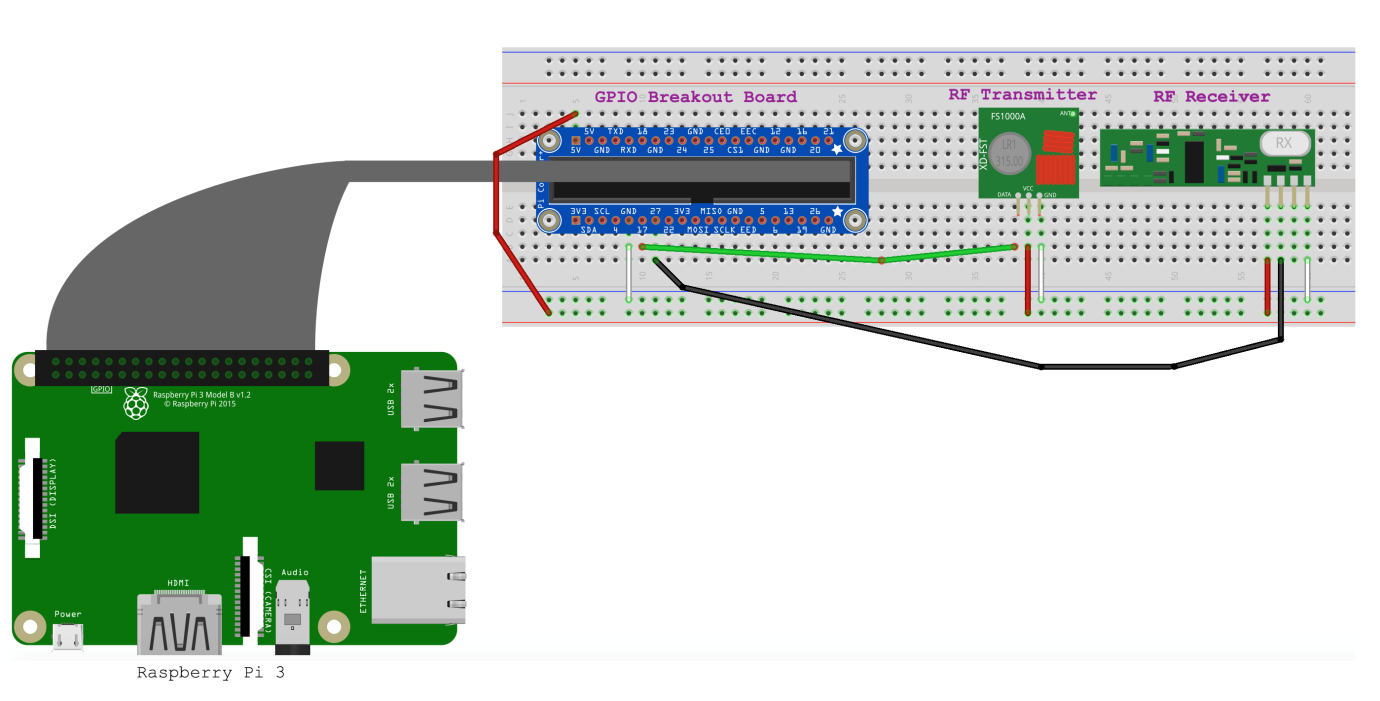
* Connect And Configure Hardware
  + Assemble RF Circuit
  + Install Software Dependencies + Libraries
  + Capture RF codes corresponding to wireless sockets
* Provision IBM Cloud Services
* Create Serverless Functions
* Deploy to IBM Cloud

**Configure Hardware Components**

We can get started by assembling and configuring the RF circuit. This circuit requires the following components

* Raspberry PI 3
* GPIO Ribbon cable + Breakout Board
* 433MHz RF transmitter and receiver
* Etekcity 433 MHz Outlets
* Electronic Breadboard
* USB Microphone

Once all components have been obtained, assemble them to form the circuit below. In this circuit, we have the Raspberry Pi connected to the electronic breadboard via the GPIO ribbon/breakout board.

****

The red wire just left of the breakout board is responsible for bridging 5 volts from the Raspberry Pi to one of the breadboard's power rails. The additional red wires to the bottom right of the diagram supply those 5 volts from the power rail to the RF receiver and transmitter. Similar concept for the white wires, except those provide a negative charge, commonly referenced to as "ground". Next, we have the green wire that connects the Raspberry Pi's GPIO pin 17 to the transmitter's data pin, and the black wire connects the GPIO pin 27 to the receiver's data pin. The reason for this can be seen in the gpio readall output in image below, as the transmitter defaults to wiringPi pin 0 which maps to BCM 17, and the receiver defaults to wiringPi pin 2, which maps to BCM 27. These default pins can be changed by modifying either of the linked files in the 433Utils library, and recompiling the library.

Once the Raspberry Pi is connected to the circuit, we'll need to install dependencies to allow us to interact with the RF transmitter and receiver. This can be accomplished by running the install\_deps.sh script.

The open source libraries that are being installed here are wiringPi and 433Utils. wiringPi enables applications to read/control the Raspberry Pi’s GPIO pins. 433Utils calls 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.

**Watson Assistant**

The Watson Assistant service is used to analyze natural language and determine which action(s) to take based on the user input. There are two main concepts to understand here. The first are referred to as "Intents", which determine what the user would like the application to do. Next, we have "Entities", which provide context of where the intent should be applied. To keep things simple, we have two intents, one is titled "turnoff", the other "turnon". Next, we have 3 entities, which are household devices that we'd like to turn off and on in this case. This pre-trained data model can be uploaded to the provisioned Watson Assistant service through the UI. To initiate the upload, login to the IBM Cloud console. Next select the Watson Assistant service, and then the button titled Launch Tool.

Watson IoT Platform

The Watson IoT Platform will be utilized as a MQTT messaging broker. This is a lightweight publish/subscribe messaging protocol that'll allow for various devices such as a Phone, Laptop, and Microphone to communicate with the Raspberry Pi. Once this service has been provisioned, we'll need to generate a set of credentials to securely access the MQTT broker. These steps are listed here

**IBM Cloud Functions**

Rather than writing and executing pipelines and complex automation logic on the Raspberry Pi, we’ll utilize a serverless, event driven platform called IBM Cloud Functions. In this implementation, IBM Cloud Functions actions forward their results to the Raspberry Pi as MQTT messages. IBM Cloud Functions is a serverless framework which has the ability to run code automatically in response to various events. It’s a fitting environment for our use case, because it can listen for various events such as IoT messages, schedule changes, HTTP POST requests, and many others.