DIY Device Cloud Documentation

Release 1.0

Tony DiCola

May 11, 2014

Contents

The internet of things sucks. We were promised a future of amazing devices that connect seamlessly and communicate freely. In reality we've been given gadgets that solve questionable needs, services that wall off and own our data, and endless marketing nonsense. It's time to stop talking about the internet of things and start making it a reality. Like the underground music scene of the 80's that built its own future, the makers, hackers, and garage inventors of today need to lead the way to an internet of things that doesn't suck. The DIY device cloud project's goal is to provide tools and information for anyone to build their own cloud of connected devices.

You can find more details about this project at its home on projects.hackaday.com.

Contents:

CHAPTER 1

Overview

1.1 What is a device cloud?

I consider a device cloud to be a service that provides both a broker for communication between devices and a host for applications that interact with devices. The broker allows devices to communicate to other devices or applications without having to be tightly coupled or directly connected to each other. The device cloud is also an easily accessible host for applications that interact with devices, such as visualizations of data or control of device functionality.

1.2 Why do you want a device cloud?

- Discovering and connecting to your devices over the internet is hard! Dynamic IP addresses, NAT, firewalls, and more prevent you from easily accessing your home network and devices. Hosting your own device cloud on the internet allows devices to connect and communicate without the trouble of opening connectivity to your home network.
- Not all devices can be connected and online all the time. A device cloud allows for asynchronous communication between devices. For example a sensor or actuator can periodically connect to your device cloud to record data or receive new instructions; because the device isn't connected all the time it can greatly reduce its power consumption.
- You have total control over the data and services available to your devices. You aren't limited by what a 3rd party provides or concerned about what they might do with your data. You own the entire infrastructure and can mold it to your needs.

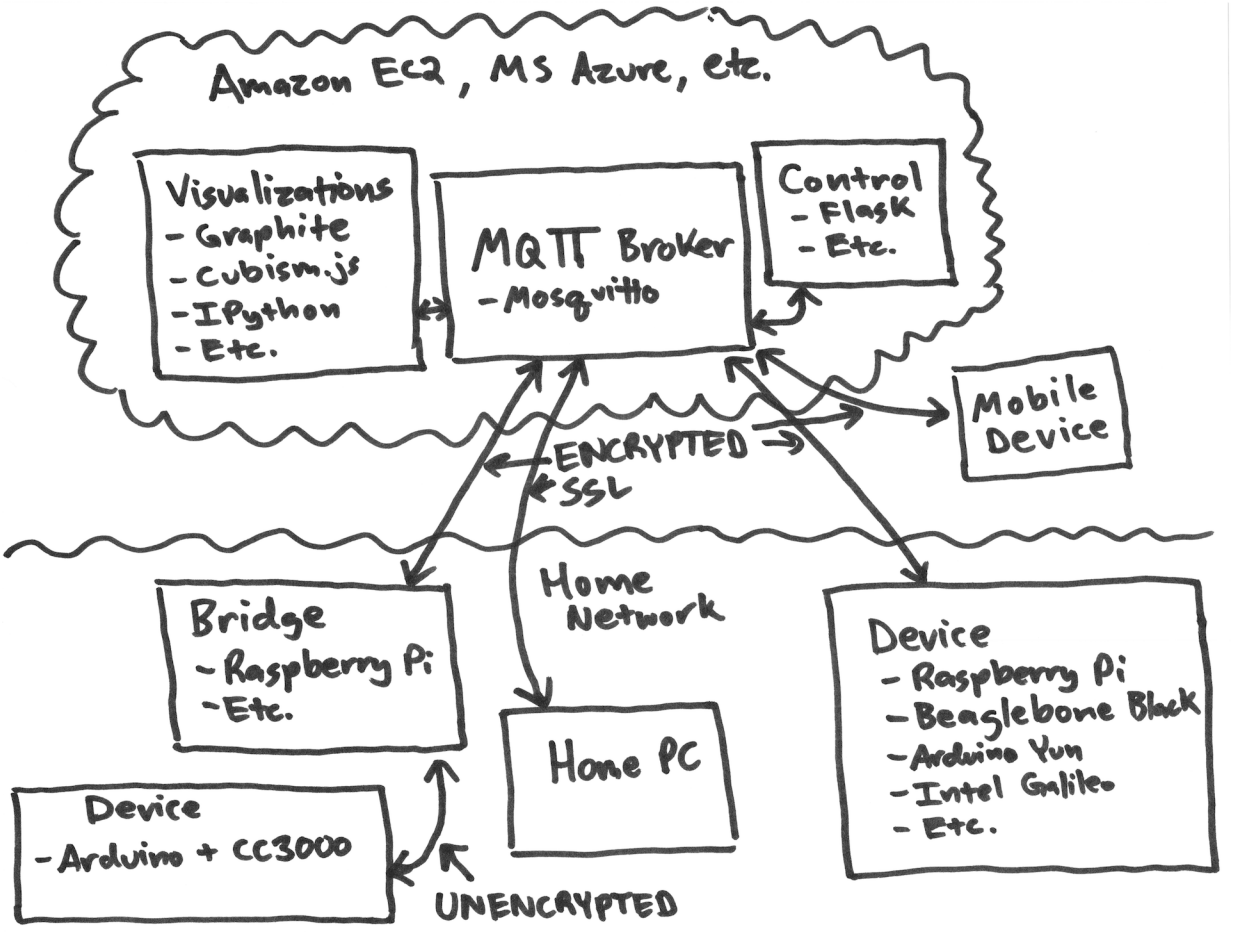
1.3 Project Goals

- Automate the setup and maintenance of services that form a personal device cloud. Setup should be an easily repeatable process from a single command or script.
- Target running the device cloud on cheap, low end cloud infrastructure like Amazon EC2's micro instance free tier.
- Build on existing tools, services, and protocols–don't reinvent the wheel! Many of the services to build a device cloud exist today, they just need to be put together into an easy to use package.
- Document how to connect popular development hardware like the Raspberry Pi, Beaglebone Black, and Arduino to a device cloud.

CHAPTER 2

Architecture

Early thoughts on high level device cloud architecture:



The MQTT broker, mosquitto, is at the center of all communication and lives on a server in Amazon EC2, Microsoft Azure, etc. Along with the broker other applications like visualizations or control apps are hosted on the same server and can communicate with the broker directly.

Devices in your home network, or really anywhere, connect to the MQTT broker using an encrypted SSL MQTT channel. Normally SSL is a tremendous pain to configure and manage (especially with self-signed certificates), however I've found it's easy to automate the certificate creation tasks using Ansible so it should be painless. I want the device cloud to be secure by default and not require someone to be an expert in security to setup or run the system.

Devices can be any hardware that's powerful enough to speak MQTT's protocol. Embedded Linux boards like the Raspberry Pi, Beaglebone Black, Arduino Yun, Intel Galileo, etc. will be very easy to configure and use with the device cloud. I hope to even provide Ansible tasks that can deploy and configure MQTT client tools and libraries on any embedded Linux device automatically.

What about devices which don't support SSL, like an Arduino & CC3000? In this case mosquitto has a concept of a bridge server which can act both as an MQTT broker and client. Any MQTT messages sent to the bridge will be relayed back up to the parent broker and vice-versa. With this setup it will be possible to build a small, low power Arduino + CC3000 device that periodically connects to the bridge over an unencrypted channel to send/receive commands. Because the bridge lives inside your home network (which is secured with wireless encryption, right?) it's still relatively secure. Certainly more secure than opening the broker in the cloud up to unencrypted MQTT traffic! Again Ansible tasks should be able to turn any Linux gadget into a bridge for the device cloud easily.

CHAPTER 3

Dependencies

If you're running Windows, you will need to install vagrant and virtualbox to setup an Ubuntu linux virtual machine for executing the playbooks. Don't worry! Vagrant makes the setup and usage of a virtual machine extremely easy.

TODO: Put in details on setting up vagrant.

If you're running Mac OS X or Linux you should have everything you need to get started.

You will need to install the following software:

3.1 Python

You probably have this installed already, to check open up a terminal and run:

```
python --version
```

Any Python version >2.6 should work. If you see an error that Python is not installed, please install the latest version of Python 2.7.

3.2 pip

pip is a Python package manager which can install all the required Python dependencies. To install pip execute:

```
wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
```

3.3 Ansible

Ansible is the automation tool that will set up and configure the machines in the device cloud. Install Ansible by executing:

```
sudo pip install ansible
```

3.4 boto

boto is a Python package for interacting with Amazon's AWS cloud services. If you are using Amazon AWS to host your device cloud you will need to install boto by executing:

```
sudo pip install boto
```