**The ESP8266 Kegerator Project –**

**Making native calls to AWS with a $3 Arduino-compatible wireless SOC module from Espressif**

This project intent was to demonstrate a simple, cheap wireless device supporting an IoT application (logging when beer is dispensed from a kegerator) by making native calls to the AWS platform. The actual function (waiting for pulses from flow meters and counting them) was trivial, wrapping “zero wire” configuring and the requirements to call AWS directly is responsible for most of the code.

Stripping out the “kegerator beer flow component”, this code allows arbitrary JSON objects to be posted to an SNS topic on AWS using HTTP with a $3 wireless module that uses little power unless transmitting (200mA needed for TX). Anything you can measure, you can log to an SNS topic (which can then send you an SNS notification, feed to a SQS queue, an HTTP endpoint, mobile app push notification, etc.). I hope that others take this code base and do some awesome things with it.

**The Code –** The code is written for readability and to make the process of making a signed request on AWS as easy to follow as possible. This means that there are WAY more Strings used than needed, and significant RAM/heap/stack savings are available with some more efficient coding. The code also allows you to “bake in” wireless passwords and AWS credentials which is a BAD idea. I found this useful in debugging the EEPROM config save/restore and web server module, but hardcoding secrets and credentials in sourcecode is a great way to get yourself hacked – be warned. The credentials that you see published are bogus.

|  |  |
| --- | --- |
| Kegerator-vXXXXX.ino | Main module, contains global variables, setup, ISR routines for the flow meters and a main loop that looks for tap-pulled events and calls “send to SNS” |
| Configuration.h | Defines constants and some globals, most of what you will mess with for configuration is located here. The #verbose flag is used to control serial port verbosity |
| EEPROM\_Utils.ino | This is where the config is written to/read from EEPROM to provide persistence across power cycles. |
| Get\_wireless\_credentials | If the module can’t access the configured SSID, it will turn into an Access Point, broadcast it’s own SSID, and act as a web server to clients that log into (configurable) 192.168.4.1. This is where you would configure the SNS region, endpoint&ARN, the AWS credentials (AKI and secret) for the tightly scoped role (write to SNS only), etc. |
| Make\_AWS\_Signing\_keys | This module contains the V2 and V4 signing code, it is passed info by tellSNS |
| NTPGetDateTime | Instead of using a realtime clock/battery, this module makes an NTP call to a configured server in order to be able to sign the requests to AWS with the correct time (and timestamp messages). |
| SHA256 -- .h and .cpp | This is an abbreviated version of Cathedrow’s great crytosuite - <https://github.com/Cathedrow/Cryptosuite> - does HMAC and SHA256 |
| Time - .h and .cpp | Time module from Adafruit |
| tellSNS.ino | Using the above, posts passed JSON string to the configured SNS topic. This is not a generic “call AWS” function, but it is easily extensible to other AWS services. NOTE – the ESP8266 is today only capable of HTTP calls (not TLS/HTTPS) so only services that support HTTP endpoints are callable in this fashion… CloudWatch, DynamoDB, EC2, ELB, EMR, S3, SNS, SQS – see <http://docs.aws.amazon.com/general/latest/gr/rande.html>. This project uses SNS as it is very extensible-friendly (subscribe an email to the topic, and your ESP can send you emails, subscribe Lambda and you can invoke Node.js functions, subscribe an SQS queue and you have highly durable queued events for processing….) |

Practically, to just “read some other sensor and send it as a JSON event to an SNS topic” you will only need to change the Kegerator and configuration.h files.

**The Module** – The ESP8266 by Espressif is a low power, low cost wifi SOC that is available for under $3 in single quantities (note, multiple HW versions are made) if you mail order - <https://github.com/esp8266/esp8266-wiki/wiki/Hardware_ESP8266-Versions>, also available via Amazon Prime - <http://www.amazon.com/s/ref=sr_nr_p_85_0?fst=as%3Aoff&rh=i%3Aaps%2Ck%3Aesp8266%2Cp_85%3A2470955011&keywords=esp8266&ie=UTF8&qid=1435783827&rnid=2470954011>

This project specifically used the ESP8266-03 version as it was the smallest physical packaging available. The Arduino IDE can be used to program and configure this module following the install instructions at <https://github.com/esp8266/Arduino>

NOTE – the ESP8266 is a 3.3V module, so you will need a 3.3V FTDI or USB:Serial converter such as <http://www.amazon.com/GearMo%C2%AE-3-3v-Header-like-TTL-232R-3V3/dp/B004LBXO2A> or <http://www.amazon.com/FT232RL-Serial-Adapter-Module-Arduino/dp/B00YHH9VG6/ref=sr_1_14?s=electronics&ie=UTF8&qid=1435775924&sr=1-14&keywords=3.3+ftdi>

to program and communicated with the device safely. If you use a 5V interface, you may smoke the module irreparably (See schematic below).

Note – some of the GPIO leads are reserved for determining boot mode (boot from serial, boot from flash, etc.), so if you are making your own circuit RTM and in the case of the ESP8266-03 you can’t use pins XXXXXXXX.

**How do I use this** – You need to have some breadboarding & soldering skills to make the physical circuit in the schematic – choose a different ESP module than the -03 if you don’t like soldering REALLY small pads. Let’s assume that you’ve wired the circuit properly, and followed the instruction linked above to get the Arduino IDE ready to compile for and download to the ESP8266. Download this code into a directory, open the KegeratorXXXXXXX .ino file in the Arduino IDE, select “Generic 8266 Module” in “Tools/Board”, then (cross your fingers) and select “->” (the Upload button) – if all goes well you will see a successful compile, then an attempt to upload.

Note – GPIO pin XXX needs to be grounded and GPIO pin YYY needs to be at +3.3V in order to make the module boot from the serial interface (which means download your code from the serial port, save to flash, and boot from it). You need to power cycle your module with the pins set this way just before the IDE tries to upload the code – otherwise the module or IDE will time out. This can be a bit finicky.

Once uploaded, monitor the serial port. Unless “#verbose” is enabled in configuration.h, you will see “….” While the module is trying to access your AP, “++++” while it is a web server waiting for configuration, and “?????!!!!!!” while it is trying to get the time from the configured NTP server. In verbose mode it is quite verbose. If you are using a brand new module, it should immediately realize and become a web server that will broadcast the SSID “Confused Kegerator needs love”. Connect to this wireless network, then connect to 192.168.4.1 – if you are patient you should see a barley-colored web page that allows you to configure SSID, PW, SNS ARN, region, endpoint, credentials, etc. All fields must be populated, note that the Secret key will never be displayed. If after 10 minutes the form is not submitted, it will revert to attempting to join the configured AP as a client and the process will repeat (this avoids a wireless glitch taking the device permanently offline).

Once you submit your configuration, if all is correct, the module will revert to being a wireless client and wait for pulses from the kegerator fluid flow module. When it detects pulses (note the pushbutton wired in parallel on the schematic - this lets you test without pouring beer), it will post a timestamped “pour started” or “pour completed, amount poured” message to SNS in JSON format. You can subscribe many different clients to this SNS topic (read below), this allows you to decouple the publishing of events (from many devices potentially) from the cloud-based processing or distribution of the events.

If you are new to AWS, then you should familiarize yourself with:

<http://aws.amazon.com/products/>

<http://aws.amazon.com/sns/>

<http://aws.amazon.com/sqs/>

Hint – subscribe to this topic using your personal email, confirm the subscription (via email) then you can see when you post successfully by receiving an email of the post.

/\* schematic of the system in Fritzing \*/

/\* screenshot of configuration webpage \*/

**How to configure AWS to support this? (Italics means “do this in the console”)**

**1 – Create an AWS account** -note, you will need a credit card (SNS does provide free tier service however) – goto <https://aws.amazon.com/> and “create account”, read about the free service tier at <http://aws.amazon.com/free/>

**2 – Create an SNS Topic that you will post to** – FIRST – select a region (if unsure, use US East, N. Virginia – you would normally choose a region that is close to you or based on service availability for what you want to do if using newer services such as AWS Lambda) then click Services, then SNS. Create Topic, enter Topic name and Display Name (no spaces – try “CloudKegs”)

**3 – Subscribe your email to the Topic** – *“Create Subscription”, Protocol – email, Endpoint – your email*

**4 – Confirm your subscription** – SNS sent you an email asking you to confirm that you want to subscribe – click the link in the email to confirm

**5 – Test the topic** – *“Publish to topic” from the console, make sure that you see an email notification… if so, then your SNS topic is functional and ready for your project to post to it.* Note down (copy to a document) the topic ARN (unique identifier), and region. For SNS, the service endpoint will be sns.***“region”***.amazonaws.com , as in “sns.us-east-1.amazonaws.com” – you will need to enter these into the device config screen.

**6 – Create a security policy that can only publish to SNS** – AWS has very granular access control. We recommend employing a least-privilege model to tightly control access to your AWS resources, so now we will create policy that only allows publishing to SNS.

*Services – IAM, Policies (left hand side), Get Started*

*Create Policy, Policy Generator (select) - you are going to create a custom security policy now*

*Effect = Allow, AWS Service = Amazon SNS, Actions = Publish, Amazon Resource Name –provide the ARN of the topic (from above), “Add Statement”, “Next Step”*

*Policy Name = “publishtoyourkegtopic”, “Create Policy”….. policy is created*

**7 – Create a user that can only publish to SNS** – Users are members of groups that are associated with security policies (like what you created above)

*Services – IAM, Groups (you are going to create a group that can only publish to SNS)*

*Create New Group, Group Name = “SNSPublishers” , NextStep*

*Attach Policy – Filter:”Customer Managed Policies”, attach the “publishtoyourkegtopic” policy that you just created to this group, “Next Step”, “Create Group”*

*Users –“ Create New Users” - Enter User Name “Kegerator” – “Show User Security Credentials”*

NOTE – THESE ARE THE CREDENTIALS YOUR ESP8266 NEEDS TO PUBLISH TO SNS – COPY THEM TO A LOCAL DOCUMENT AND DOWNLOAD A COPY. They will only be available ONCE, NOW – if you lose them, you will need to create a new pair from this screen.

*“Continue”, select Kegerator user, “Add User to Groups”, select SNSPublishers group*

Now you should have a user called Kegerator that can ONLY publish to the SNS topic you created above. You should have copied down the credentials that you will input to the web page of the ESP8266, and this should give you an AWS back-end for your IoT kegerators.

Specifically, you will need to enter the region name, the service endpoint, the ARN of the SNS topic and the credentials into the ESP8266 in order to publish to SNS, such as:

|  |  |
| --- | --- |
| Region |  |
| Endpoint |  |
| SNS ARN | sns.us-east-1.amazonaws.com |
| AKI | AKIAJI6BOGUSBWHLTOUQ |
| Secret | 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 |

Access Key ID:

AKIAJI6X3WN2BWHLTOUQ

Secret Access Key:

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