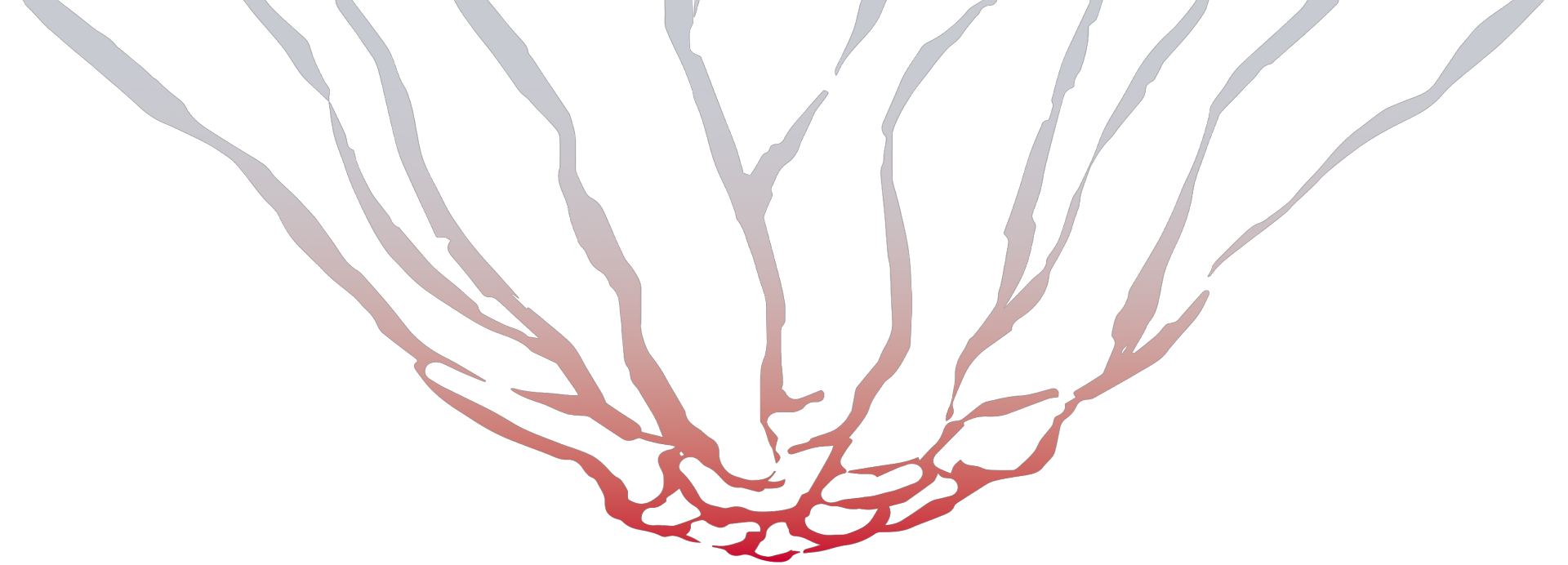


# ESPRESSIF

## TRAINING

ESP Bootstrap

# ESP Bootstrap: Objective

- Build a complete “product”: Smart Power Outlet
  - Control GPIO output
  - Network Configuration
  - Remote control through cloud
  - OTA Upgrades
  - Reset to Factory

# Hello World

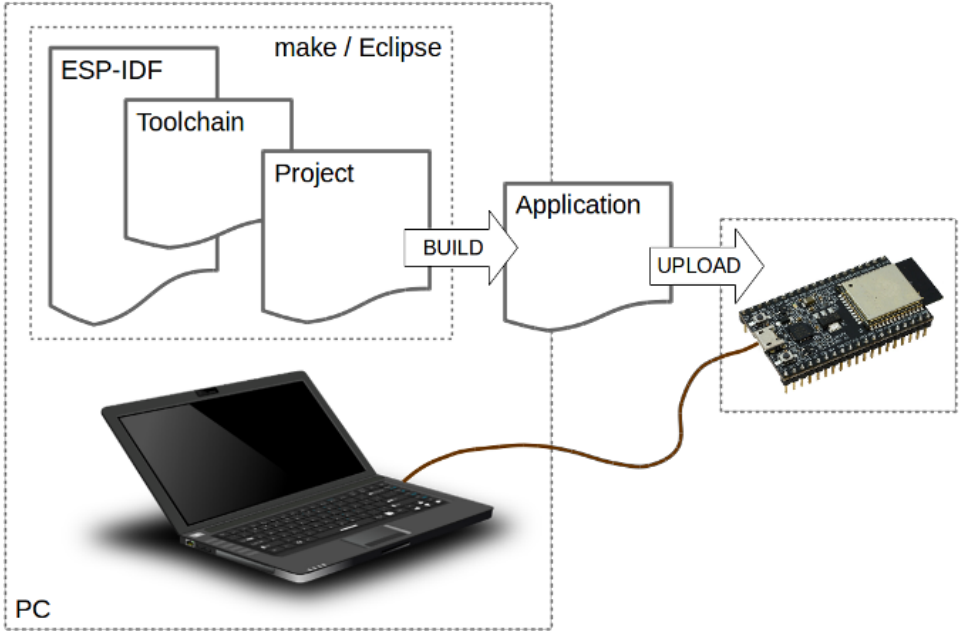
# Objective

- Getting Ready
- Development Host Setup
- Configure and Build
- Flash
- Console

# Getting Started

- Development Host PC
  - Windows, Linux, Mac
- ESP32 DevKit - C
- USB to mini-USB cable
- Wi-Fi Access Point

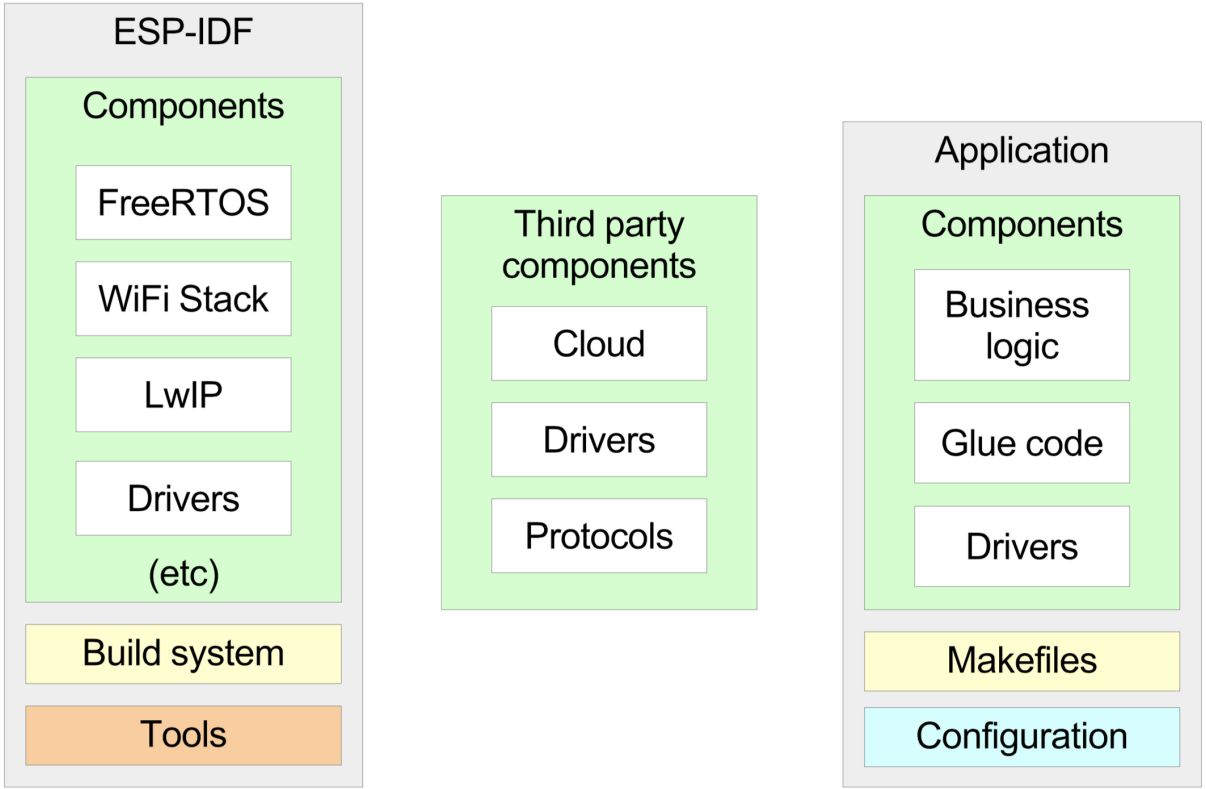
# Overview



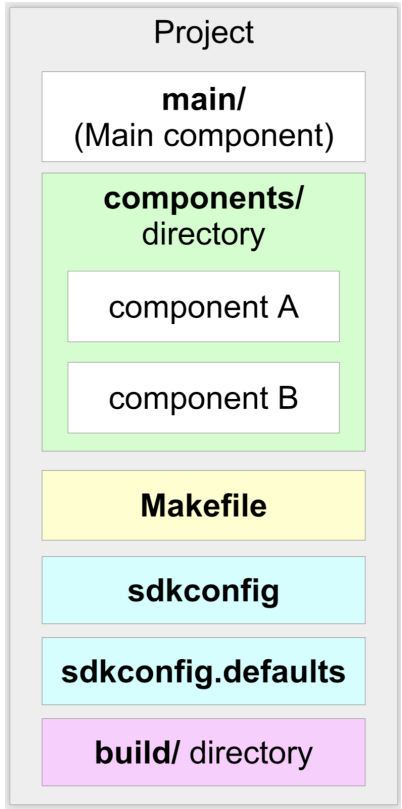
# ESP-IDF

- Espressif’s IoT Development Framework
- Collection of Libraries and Header files
- Tools for build, flash, debug

# Component Based Design



# Structure of an application/project



# Configuration

- Configuration system based on Linux kernel’s kconfig
- Configure settings for your project (impacts feature, performance, memory)
- Each component can define its own configuration options
- Use **make menuconfig** command to edit the configuration

# Build and Load

- make flash
- make monitor

# Code

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"


void app_main()
{
    int i = 0;
    while (1) {
        printf("[%d] Hello world!\n", i);
        i++;
        vTaskDelay(5000 / portTICK_PERIOD_MS);
    }
}
```

# Code

- app_main: Application Entry Point
- Runs in the application thread
- C Library
- FreeRTOS

# The C standard Library

ESP-IDF uses newlib 2.2.0 as C standard library

- Parts of newlib are placed into ESP32 ROM
- Newlib interacts with RTOS through a syscall table
- Newlib supports per-task reentrant structures
- Most of the C standard library is supported, plus extras

# The C standard Library

**Not supported**

- Wide character functions
- Signals
- Locales
- atexit, system, mallinfo

**Supported**

- stdlib, stdio, string, math, time
- File/Directory operations
- Reentrant functions
- Timezones, DST

# FreeRTOS

Real-Time Operating System https://www.freertos.org

- Threads
- Interrupts
- Message Queues, Semaphores and Mutexes
- Timers

API Reference: https://www.freertos.org/a00106.html

# Outlet

# Objective

- Use GPIOs
- Configure a push-button
- Control an output

# Push-Button

- Assert a GPIO
- Callback handler
- iot_button component: Allows long-hold actions

# Code

```
#include <iot_button.h>


button_handle_t
btn_handle=iot_button_create(BUTTON_GPIO,
                                    BUTTON_ACTIVE_LEVEL);


iot_button_set_evt_cb(btn_handle, BUTTON_CB_RELEASE,
                                btn_cb, "RELEASE");
```

# GPIO

- Configure mode
- Set output

# Code

```
gpio_config_t io_conf;
io_conf.mode = GPIO_MODE_OUTPUT;
io_conf.pull_up_en = 1;
io_conf.pin_bit_mask = ((uint64_t)1 << OUTPUT_GPIO);

/* Configure the GPIO */
gpio_config(&io_conf);

/* Assert GPIO */
gpio_set_level(OUTPUT_GPIO, target);
```

# Wi-Fi

# Objective

- Configure Wi-Fi station interface
- Connect to pre-configured AP
- Asynchronous event handler:
  - Connected
  - Disconnected
  - IP Address assignment

# Code

```
#include <esp_wifi.h>
#include <esp_event_loop.h>

tcpip_adapter_init();
esp_event_loop_init(event_handler, NULL);


wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
esp_wifi_init(&cfg);
esp_wifi_set_mode(WIFI_MODE_STA);
```

# Code

```
wifi_config_t wifi_config = {
    .sta = {
        .ssid = EXAMPLE_ESP_WIFI_SSID,
        .password = EXAMPLE_ESP_WIFI_PASS,
    },
 };
esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config);

esp_wifi_start();
```

# Code

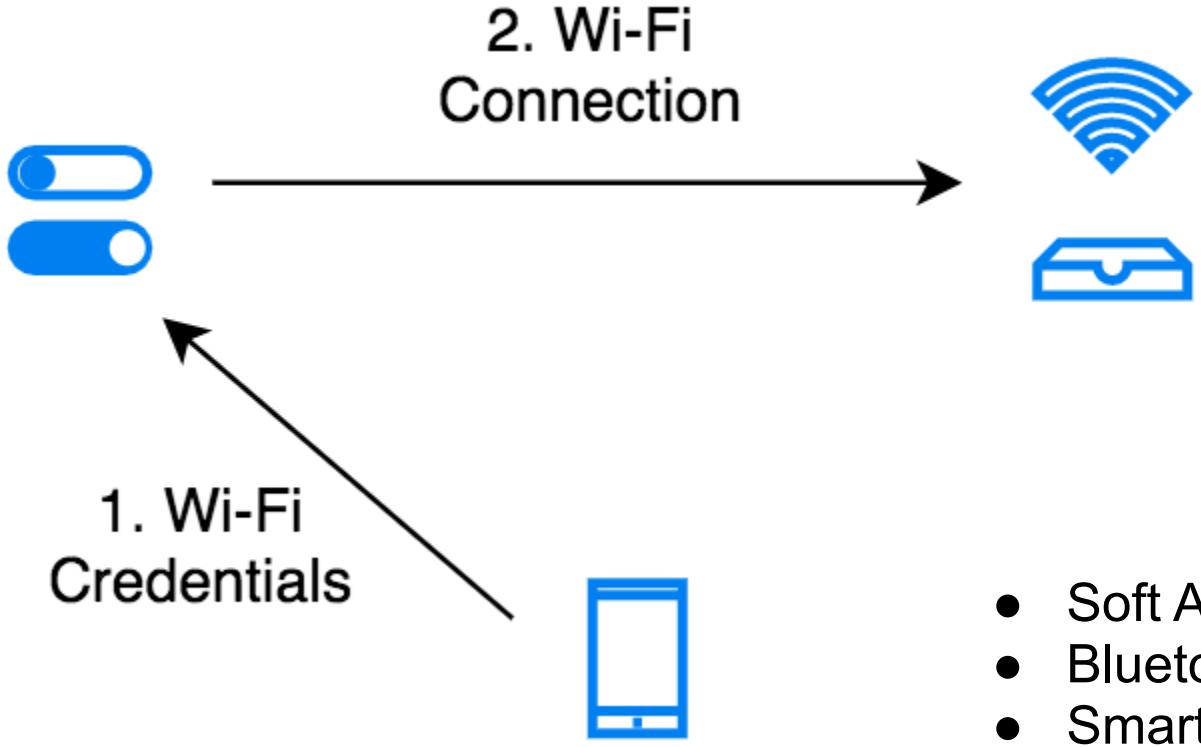
```
esp_err_t event_handler(void *ctx, system_event_t
*event)
{
    switch(event->event_id) {
    case SYSTEM_EVENT_STA_START:
        esp_wifi_connect();
        break;
    case SYSTEM_EVENT_STA_GOT_IP:
        ESP_LOGI(TAG, "Connected with IP Address:%s",
ip4addr_ntoa(&event->event_info.got_ip.ip_info.ip));
        break;
    case SYSTEM_EVENT_STA_DISCONNECTED:
        esp_wifi_connect();
        break;
    return ESP_OK;
}
```

# Network Configuration

# Objective

- Get your product on the user's home network
  - as opposed to a pre-configured network
- User Settings and Reset to Factory

# Method




- Soft AP
- Bluetooth
- Smart-Config

# Considerations

- Soft AP
  - Manual Wi-Fi change on iOS
- Bluetooth
  - Larger flash footprint
- Smart-Config

# Network Provisioning

- Specification to securely transfer credentials to the device, independent of the transport (SoftAP, Bluetooth)
- Device-side implementation of the specification
- Libraries for Android/iOS that implement the specification
- Reference applications on the device and Android/iOS

# Configurations

- SoftAP or BLE?
- Secure Exchange?
- Proof of possession?
- Service Name?

# Code

```
if (conn_mgr_prov_is_provisioned(&provisioned) !=
ESP_OK) {
    return;
}

if (! provisioned) {
    /* Starting provisioning */
    conn_mgr_prov_start_provisioning(prov_type,
                security, pop, service_name,
service_key);
} else {
    /* Start the station */
    wifi_init_sta();
}
```

# Code

```
esp_err_t event_handler(void *ctx, system_event_t
*event)
{
      conn_mgr_prov_event_handler(ctx, event);

      switch(event->event_id) {
      case SYSTEM_EVENT_STA_START:
…
…
…
```

# Flash Partitions

- Store user’s configuration: NVS
- Custom Partition Table: partitions.csv
- Update in ‘menuconfig’

# NVS

- Key-Value store in flash
- Power loss resilient
- Wear-levelling: Log based structure
- Support namespaces
- Used for: Manufacturing settings, User's configuration, Maintaining state across resets

# Reset to Factory

- Long-press Push Button
- Erase NVS

# Code

```
static void button_press_3sec_cb(void *arg)
{
    nvs_flash_erase();
    esp_restart();
}

iot_button_add_on_press_cb(btn_handle, 3,
                           button_press_3sec_cb, NULL);
```

# Phone Application

- Android and iOS
- Structured as library and phone app
- Android: https://github.com/espressif/esp-idf-provisioning-android
- iOS: https://github.com/espressif/esp-idf-provisioning-ios

# Cloud: Remote Control & Analytics

# Objective

- Cloud communication:
  - Update local state information
  - Fetch commands
- Security:
  - Cloud CA Certificate
  - Device’s authentication Certificates
- Example Cloud: AWS

# AWS

- Example Cloud
- Uses Device Certificate and Key for authentication
  - Pre-generated for the training

# Embedding Files in the Firmware

- Update your component.mk

```
COMPONENT_EMBED_TXTFILES := certs/aws-root-ca.pem  \
                            certs/certificate.pem.crt \
                            certs/private.pem.key
```

- Access in your code

```
extern const uint8_t aws_root_ca_pem_start[]

asm("_binary_aws_root_ca_pem_start");
extern const uint8_t aws_root_ca_pem_end[]
                     asm("_binary_aws_root_ca_pem_end");
```

# Change the thing name

- Update the thing-name in your code to match your thing-name

# RESTful API read

- Read state using HTTP API

```
$ curl --tlsv1.2 --cert /work/certificate.pem.crt --
key /work/private.pem.key https://aln7lww42a72l-
ats.iot.us-east-2.amazonaws.com:8443/things/chicago1/
shadow | python -mjson.tool
```

# RESTful API write

- Update state using HTTP API

```
$ curl -d '{"state":{"desired":{"output":false}}}'  --
tlsv1.2 --cert /work/certificate.pem.crt --key /work/
private.pem.key https://aln7lww42a72l-ats.iot.us-
east-2.amazonaws.com:8443/things/chicago1/shadow |
python -mjson.tool
```

# OTA Upgrade

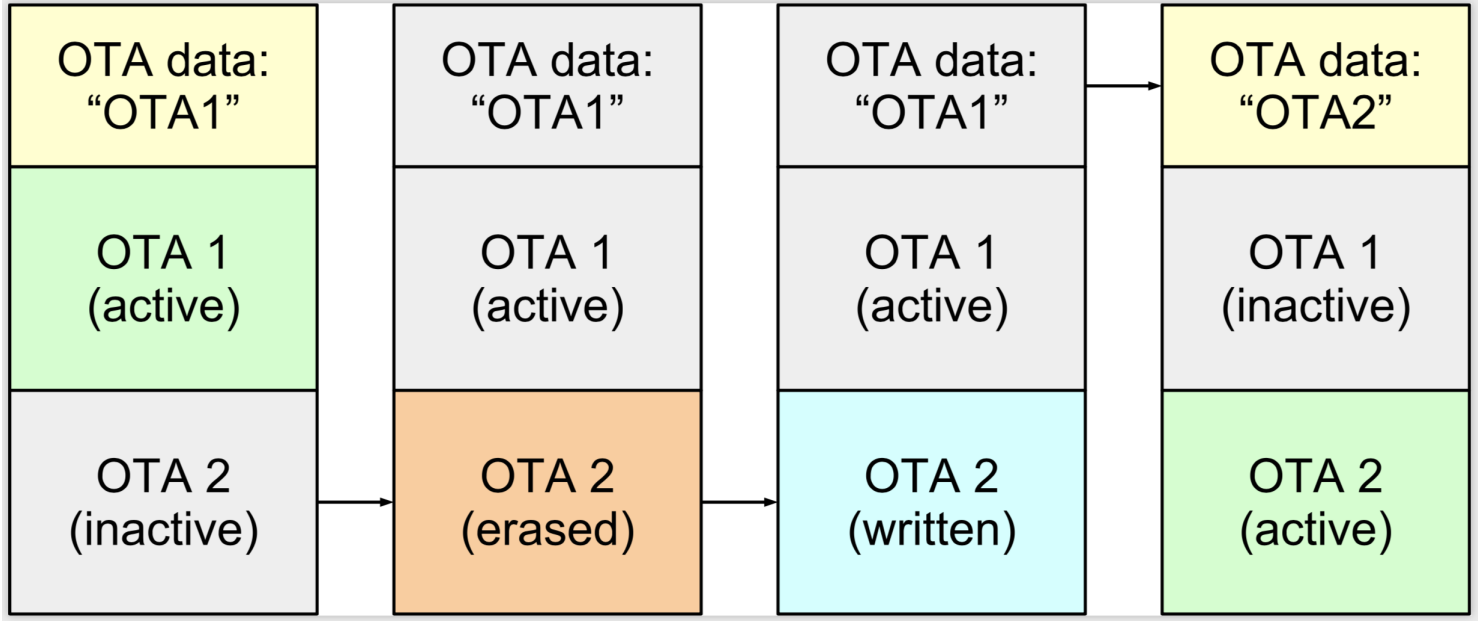
# Objective

- Over the Air Firmware Update

# Flash Partitions once more

- Active-Passive Firmware Partitions
- OTA Data partition stores currently active partition state

# Flash Partitions once more



# Typical Considerations

- Server’s CA certificate validation
- Different from device state change commands
- Upgrade deployment stages

# Code

```
    esp_http_client_config_t config = {
        .url = url,
        .cert_pem =
(char*)upgrade_server_cert_pem_start,
    };
    esp_err_t ret = esp_https_ota(&config);
    if (ret == ESP_OK) {
        esp_restart();
    } else {
        ESP_LOGE(TAG, "Firmware Upgrades Failed");
    }
```

# Test Upgrade URL

- https://raw.githubusercontent.com/kedars/drawer/master/hello-world.bin

# Factory Partition

# Objective

- Allow configuring per-device unique configurations
- To be applied at the factory during manufacturing

# NVS Once More

- Separate NVS partition for storing factory settings
- So 2 NVS partitititions:
  - For storing user configuration. Is erased on reset-to-factory action
  - For storing device-unique configuration at the time of manufacturing. Is never erased
- Update the partitions.csv with this new partition

# Pre-generate NVS Partition

- Decide partition content:

```
key,type,encoding,value
mfg_ns,namespace,,
serial_no,data,string,btest1
cert,file,string,/training/esp-bootstrap/7mfg/main/
certs/certificate.pem.crt
priv_key,file,string,/training/esp-bootstrap/7mfg/main/
certs/private.pem.key
```

- Three keys: serial_no, cert and priv_key, are part of the NVS within the ‘mfg_ns’ namespace of this NVS partition

# Pre-generate NVS Partition

- Convert the file to an NVS on-flash image:

```
$ python /training/esp-bootstrap/esp-idf/components/
nvs_flash/nvs_partition_generator/nvs_partition_gen.py
--version v1 mfg_config.csv mfg.bin
```

- Flash this partition at its right location:

```
$ /training/esp-bootstrap/esp-idf/components/esptool_py/
esptool/esptool.py --port /dev/tty.SLAB_USBtoUART
write_flash 0x340000 mfg.bin
```

# Read the manufacturing data

- Read the manufacturing data from the firmware using NVS API

```
nvs_handle fctry_handle;
nvs_flash_init_partition(MFG_PARTITION_NAME);

nvs_open_from_partition(MFG_PARTITION_NAME, "mfg_ns",
                                NVS_READONLY, &fctry_handle);
nvs_get_str(fctry_handle, “serial_no”, serial_no);
nvs_close(fctry_handle);
```

# Product Ready!

- Now we have a complete end-product:
  - Control GPIO output
  - Network Configuration
  - Remote control through cloud
  - OTA Upgrades
  - Reset to Factory

# Thank You!