# Choosing the best hardware for your next IoT project

## Use this hardware guide to determine which hardware to use when prototyping and developing IoT projects

Anna Gerber

January 03, 2018
(First published May 05, 2017)

This article provides a balanced overview of different types of hardware that are commonly adopted for IoT, including micro controllers (for example, Arduino), single board computers (for example, Raspberry Pi), and embedded devices. The article describes the high-level building blocks and key characteristics of hardware, including an overview of needed security features and firmware capabilities, in the context of IoT. The article also provides guidance for when, where and why a developer might choose one type of hardware over another for an IoT project, and examples of use across a range of industries.

### IoT 101: Getting started with IoT development

This article is part of the IoT 101 learning path, a quick-start guide for IoT developers.

- IoT concepts and skills
- IoT hardware guide (this article)
- IoT networking guide
- IoT platforms
- Tutorial: Build a simple home automation system

Connected hardware devices are at the heart of IoT. IoT devices monitor and instrument "things," or real-world objects, including industrial equipment, home appliances, buildings, cars, warehouse inventory items, and people (in the case of wearable devices).

When you develop new IoT solutions, hardware and software components are designed, prototyped, and refined through an iterative process of feedback and evaluation. Hobbyist hardware platforms like Arduino and Raspberry Pi can help jump start this process of rapid prototyping and refinement, because they are readily available and require less investment than designing and fabricating custom *printed circuit boards* (PCBs) at each iteration of the design. As part of this process, you'll need to consider the hardware requirements for your own IoT application, and evaluate and refine the prototype IoT devices that you build against these

requirements and adapt them, adopting off-the-shelf components or custom components as appropriate.

In the context of IoT, *device* is an overloaded term that describes hardware that has been designed or adapted for a particular purpose. It is used to refer to individual hardware components including sensors and actuators, as well as to off-the-shelf boards like Raspberry Pi, and also to custom prototype and production units that are built from a number of constituent devices. In this article, I'll review some of the widely available off-the-shelf hardware options and discuss why you might choose one device over another when prototyping and developing your next IoT project.

## IoT device characteristics

New devices and device platforms are continually being released as the IoT landscape matures. You need to understand the key characteristics that are common across most IoT devices to compare and evaluate new devices as they become available.

We can characterize IoT devices in terms of these high-level capabilities:

- Data acquisition and control
- Data processing and storage
- Connectivity
- Power management

### Data acquisition and control

*Data acquisition* (DAQ) is the process of measuring real-world conditions and converting these measurements into digital readings at fixed-time intervals (the data sample rate). DAQ also involves signal conditioning, which is used to manipulate and scale raw sensor readings, and analog-to-digital converters, which are used to convert the analog sensor readings into digital values so that they can be processed and analyzed.

*Sensors* are the input components that measure physical variables and convert them to electrical signals (voltages). You can choose from thousands of types of off-the-shelf sensors to measure a range of variables, including temperature, humidity, pressure, smoke, gas, light, sound, vibration, air-flow, water-flow, speed, acceleration, proximity, GPS position, altitude, or force, and the list goes on. But sensors don't just measure ambient conditions; proprioceptive sensors monitor the internal state of the device, and sensors like buttons, sliders, or a touchscreen can be used for interacting directly with the device, providing a Human-Machine Interface. For each individual type of sensor, like a temperature sensor, you'll have dozens of alternative component choices from a range of manufacturers, each with slightly different specifications in terms of accuracy and precision, and each designed for specific applications and operating conditions, such as for use underwater or to withstand extremes of heat and cold.

An important characteristic of sensor components is their resolution. The resolution of a sensor represents the smallest amount of change that the sensor can reliably read and is related to the size of the numeric value that is used to represent raw sensor readings. For example, an analog temperature sensor with 10 bits of resolution represents a temperature reading using a

numeric value between 0 and 1023. Bits are binary, so 10 bits provides two to the power of 10, or 1024 possible values in total. However, in practice, sensors are affected by electrical noise which reduces the actual resolution.

While sensors convert a physical variable like temperature to an electrical signal, output devices are the inverse: they convert an electrical signal to a physical outcome. Output devices include LEDs, speakers and screens, and *actuators* like motors or solenoids that move or control things in the physical world. Actuators are commonly deployed within industrial IoT applications; for example, pneumatic linear actuators are widely adopted in manufacturing to move and grip products during the assembly process.

## Data processing and storage

IoT devices require data processing and storage capabilities to perform basic handling, transformation, and analysis of the data that they capture. IoT devices can process data directly, or they can transmit this data to other devices, gateway devices, or cloud services or apps for aggregation and analysis.

Edge analytics involves performing data analysis at the edges of a network rather than in a centralized location. Data can be analyzed in near real-time on the devices themselves, or on a nearby gateway device (like a router) that the IoT devices are immediately connected to, rather than devices transmitting large volumes of data upstream to a cloud server or data center for further analysis. Processing data at the edge provides an opportunity to aggregate and filter the data as it is collected, with only the most salient data selected to be sent upstream. Ultimately, edge analytics reduces the upstream processing and storage requirements as well as relieves the load on the network.

The processing power and storage that is used by an IoT application will depend on how much processing occurs on the device itself as opposed to how much processing is performed by the services or apps that consume the data. The amount of memory that is available and the specifications of the processor, including the clock speed and number of cores, determine the rate at which data can be processed by the device. The capacity of the non-volatile flash memory, which is used to persist data until it can be transmitted upstream, determines how much data can be stored on the device. Devices performing edge analytics will require substantially more processing capabilities than devices that perform only basic data processing like validating, normalizing, scaling, or converting readings, such as converting raw temperature readings into Celsius.

## Connectivity

Network connectivity is one of the defining characteristics of any IoT device. Devices communicate with other devices locally, and publish data to services and apps in the cloud. Some devices communicate wirelessly, by using 802.11 (wifi), Bluetooth, RFID, cellular networks, or Low Power wide area network (LPWAN) technologies like LoRa, SigFox or NB-IoT. Wired communication is suited to stationary devices, which are installed in smart buildings, home automation, and industrial control applications, where they can be connected with Ethernet or retrofitted with Ethernet over power. Serial communication is also a form of wired connectivity between devices, using standard

protocols like *Universal Asynchronous Receiver Transmitter* (UART), or the *Controller Area Network* (CAN) protocol, which has its origins in the automotive industry.

### Power management

Power management is of particular concern for portable and wearable IoT devices that rely on batteries or other non-wired power sources like solar. Depending on the usage patterns and the power requirements of the attached sensors, actuators, or *Integrated Circuits* (ICs) that provide data acquisition and control, storage, processing and networking capabilities, a device might need to be put into sleep mode or into low-power mode periodically to conserve power or extend battery life. For example, a single-board computer like the Raspberry Pi 3 requires around 700 - 1000mA of current to operate under typical usage. If you were transmitting data constantly over the wifi network, or if you were placing the device under heavy load by performing a lot of data processing on the device, the power usage would be at the upper end of that scale and would drop whenever the device was idle. If you connect a camera module, the current required increases by around 250mA whenever the camera is in use. Sensors usually require power to operate, and the GPIO pins on the Raspberry Pi supply 3.3V or 5V, up to a total of 50mA current across all of the pins, so the power consumption of the device as a whole also increases as you increase the number of components that are attached to the pins.

## Types of off-the-shelf hardware for prototyping your IoT project

Developing IoT applications is more accessible than ever, thanks to the growing range of low-cost, commercially available off-the-shelf hardware development boards, platforms, and prototyping kits. Modular hardware designs provide a great deal of flexibility. You can substitute alternative components and try out sensors with slightly different specifications, or you can independently upgrade the networking, data processing, or storage modules of a device to cater for evolving requirements.

Many commercial off-the-shelf hardware devices, including micro controllers and single board computers, are designed around *System-on-a-Chip* (SoC) ICs. SoCs bundle a number of capabilities including data processing, storage, and networking, onto a single chip. This configuration means that you sacrifice some flexibility for the sake of convenience, but, fortunately, there are a huge number of commodity devices available with a range of configurations to choose from. For example, Table 1 lists the technical specifications for a selection of microcontrollers that can be used for prototyping IoT projects, while Table 2 provides a comparison of three popular *Single-Board-Computers* (SBCs).

### Microcontroller development boards

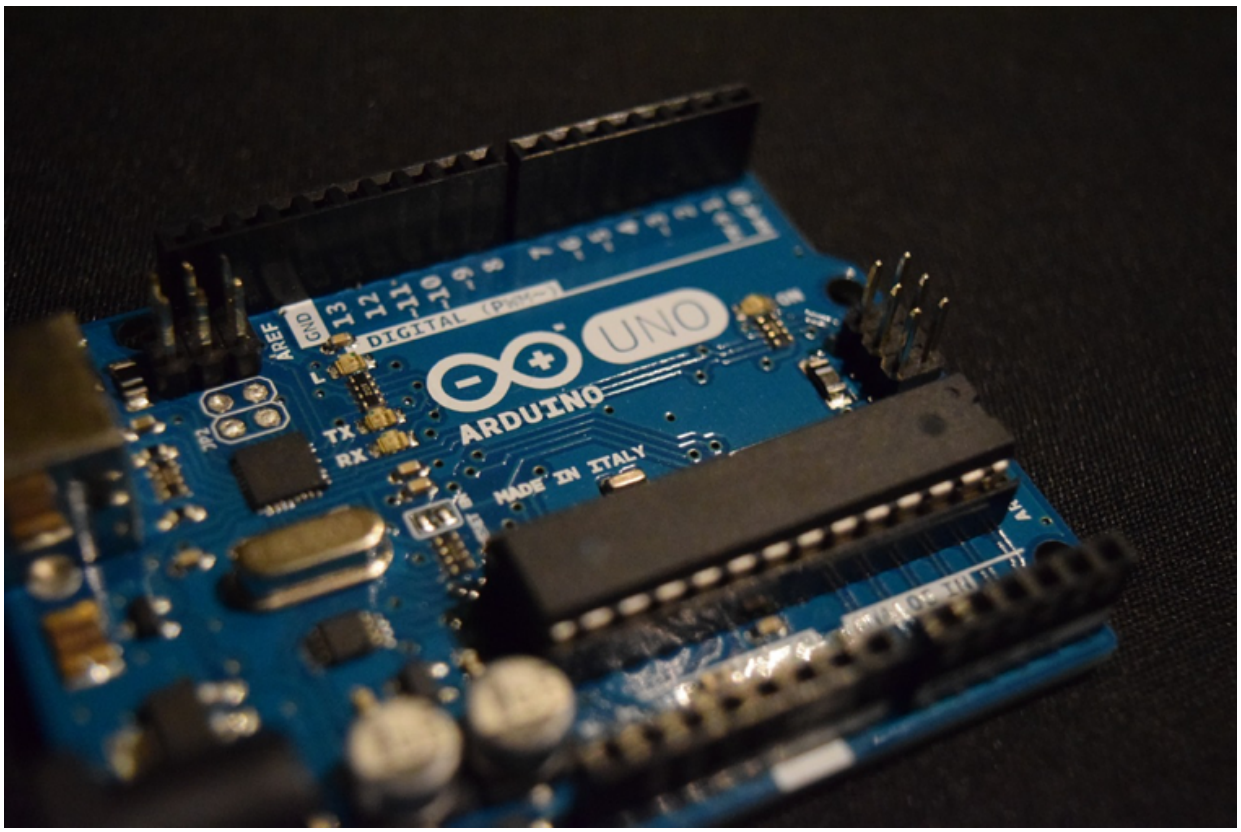
A *microcontroller* is a SoC that provides data processing and storage capabilities. Microcontrollers contain a processor core (or cores), memory (RAM), and *erasable programmable read-only memory* (EPROM) for storing the custom programs that run on the microcontroller. *Microcontroller development boards* are PCBs with additional circuitry to support the microcontroller to make it more convenient to prototype with and program the chip.

Sensors and actuators connect to the microcontroller through digital or analog *General Purpose Input/Output* (GPIO) pins or through a hardware bus. Standard communication protocols like I2C

and SPI are used for intra-device communication with the components that are connected with the bus. Adopting standards makes it easier to add or swap out components that are connected with the bus.

Arduino (http://arduino.cc/en/Main/) is an open source device platform, with an active community who are creating compatible development boards and tooling. Device capabilities vary across the official Arduino models (https://www.arduino.cc/en/Products/Compare), and also between the dozens of third-party compatible boards. All of the devices in Table 1 are Arduino-compatible microcontrollers, including the ubiquitous Arduino Uno, Particle's Electron, which includes an integrated cellular modem, and Espressif Systems' (https://espressif.com/en/products/hardware/esp8266ex/overview) ESP8266-01, a low cost, low-power microcontroller with integrated wifi.

## Figure 1. Arduino micro-controller development board




Like Arduino, the ESP8266, has an active community of adopters. Notable development boards that are based around the ESP8266, include NodeMCU (http://nodemcu.com/index_en.html), WeMos D1 (https://www.wemos.cc/) and AdaFruit's Feather Huzzah (https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/overview). A number of alternative firmware options for ESP8266-based boards have been developed by the open source and maker community, enabling IoT developers to program for these boards using Lua, Python, and JavaScript, and to support over-the-air (OTA) updates.

## Table 1. Technical specifications for Uno, Electron and ESP82566-01 microcontrollers

| Characteristic | Feature | Arduino Uno | Particle Electron | Espressif Systems ESP8266-01 |
|---|---|---|---|---|
| *Data acquisition and control* | | | | |
| | **GPIO pins** | 6 Analog in<br>14 Digital - 6 PWM | 12 Analog in<br>2 Analog out<br>30 Digital - 15 PWM | 2 Digital<br>1 Analog |
| | **Logic level voltage** | 5V | 3.3V | 3.3V |
| *Data processing and storage* | | | | |
| | **Processor** | ATMega328PU | 32-bit STM32F205 | 32-bit Tensilica L106 |
| | **Processor speed** | 16 KHz | 120 MHz | 80 MHz |
| | **Memory** | 32 kB flash,<br>1 kB EEPROM | 1 Mb flash,<br>128 kB RAM | 1 Mb |
| *Connectivity* | | | | |
| | **Network Interfaces** | None by default. Can be added with shields. | Integrated cellular modem (2G / 3G) | Integrated wifi |
| *Power* | | | | |
| | **Recommended Power Supply** | 9-12V DC 0.5 - 2A barrel, or 5V 500mA USB or 9 - 12 V on VIN pin | 5V micro USB or 3.9V-12VDC on VIN pin | Regulated 3.3V 300mA supply on VCC pin |
| *Other* | | | | |
| | **Dimensions** | 2.7 in X 2.1 in | 2.0 in x 0.8 in | 1.4 in x 1 in |
| | **Typical cost** | $25 | $39 - $59 | $10 |

The standard approach for developing the software to run on Arduino-compatible microcontrollers is to use C or C++ and the Arduino IDE, however community-developed language bindings and visual programming tools also exist. Arduino-compatible boards that share common pin layouts are able to be expanded by using optional third-party shields, for example, to add an Ethernet port or Bluetooth to an Arduino Uno. Arduino is the most widely adopted hobbyist microcontroller development environment, but others like Tessel (https://tessel.io/) and Particle.io (http://particle.io) natively support JavaScript, while Python is supported for boards like MicroPython's PyBoard (http://micropython.org) and by WeIO (http://we-io.net/hardware/).

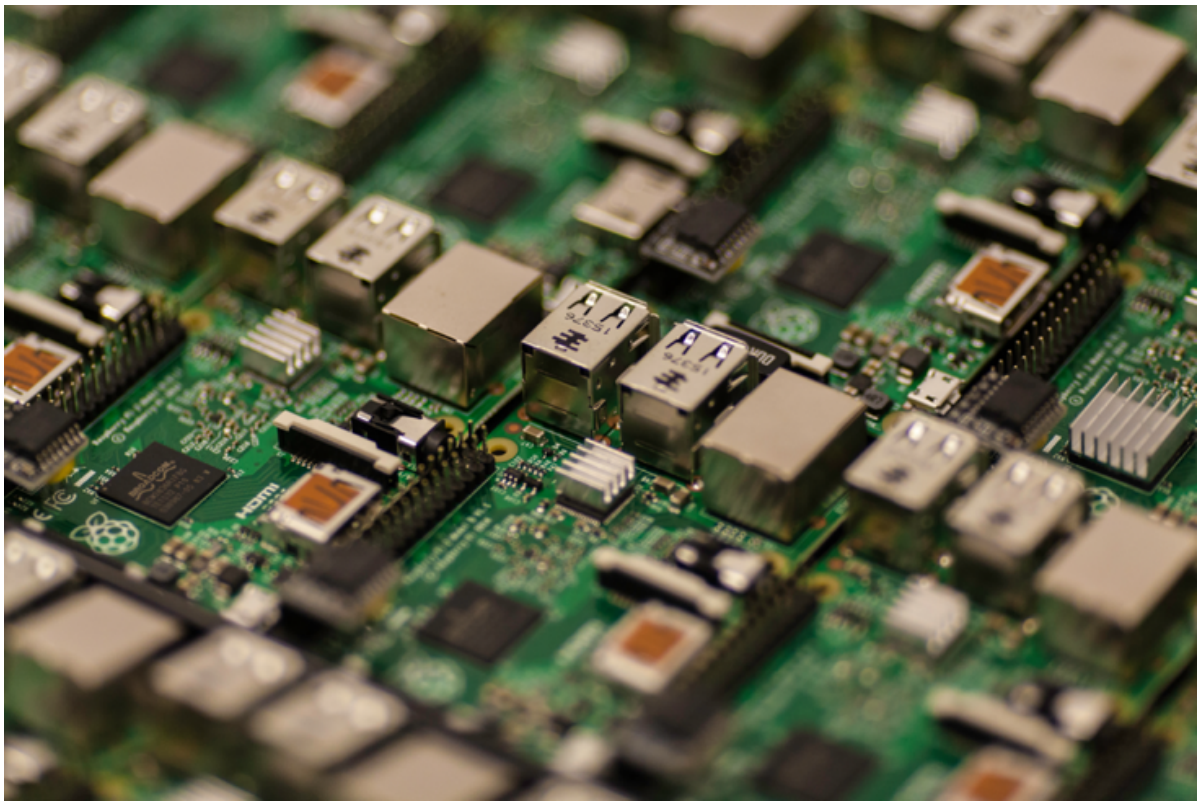
Selecting an Arduino-compatible microcontroller makes it easier to port programs that are developed using the cross-platform Arduino libraries and Arduino IDE to run on other Arduino-compatible devices. You will still have differences to work around, for example, the Arduino Uno uses 5V logic on digital I/O pins (where 0 volts equals LOW or OFF and 5 volts equals HIGH or ON), but the ESP8266 and Particle boards use 3.3V logic (HIGH is 3.3V). This might affect your choice of sensor or actuator components, as some components only work with one or the other. Swapping sensors that are designed for 5V to 3.3V logic might result in unpredictable results and possibly damage the pins that are intolerant to higher voltages, and so you'd need to add a logic-level converter to make this work. When you get down to implementing low-level hardware features like enabling deep sleep mode or reading from connected sensors by using specific

protocols, you'll likely need to rely on device or component-specific libraries that will make your code less portable.

## Single board computers

*Single board computers* (SBCs) are a step up from microcontrollers, because they allow you to attach peripheral devices like keyboards, mice, and screens, as well as offering more memory and processing power (for example, a 1.2 GHz 32-bit ARM microprocessor from Table 2 compared to an 8-bit 16KHz microcontroller from Table 1). Table 2 lists technical specifications for three SBCs, the Raspberry Pi 3 Model B (https://www.raspberrypi.org/), BeagleBone Black (http://beagleboard.org/black) and DragonBoard 410c (https://developer.qualcomm.com/hardware/dragonboard-410c).

### Figure 2. Raspberry Pi single-board-computer



The distinction between microcontrollers and single-board-computers is somewhat arbitrary. Some devices, like the Onion Omega 2 (https://docs.onion.io/omega2-docs/omega2.html#omega2), fall somewhere in between, with almost as much on-board memory and processing capability as a low-end SBC. There are also a number of hybrid devices, like the UDOO Quad (http://www.udoo.org/docs/Introduction/Introduction.html) that integrate an ARM-based Linux system with an Arduino-compatible micro-controller.

### Table 2. Technical specifications for Raspberry Pi 3, BeagleBone Black and DragonBoard SBCs

| Characteristic | Feature | Raspberry Pi 3 Model B | BeagleBone Black | Qualcomm DragonBoard 410c |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| ***Data acquisition and control*** | | | | |
| | **GPIO pins** | 40 I/O pins, including 29 Digital | 65 Digital - 8 PWM<br>7 Analog in | 12 Digital |
| | **Logic level voltage** | 3.3V | 5V | 1.8V |
| ***Data processing and storage*** | | | | |
| | **Processor** | ARM Cortex A53 | AM335X ARM Cortex A8 | ARM Cortex A53 |
| | **Processor speed** | 1.2GHz | 1 GHz | 1.2 GHz |
| | **Memory** | 1 Gb | 4 Gb | 1Gb, 8Gb Flash |
| ***Connectivity*** | | | | |
| | **Network Interfaces** | Wifi, Ethernet, Bluetooth | Ethernet,<br>USB ports allow external wifi / Bluetooth adaptors | Wifi, Bluetooth, GPS |
| ***Power*** | | | | |
| | **Recommended Power Supply** | 5V 2.5A micro USB port | 5V 1.2A - 2A barrel | 6.5 to 18V 2A barrel |
| ***Other*** | | | | |
| | **Dimensions** | 3.4 x 2.2 in | 3.4 x 2.1 in | 3.3 in x 2.1 in |
| | **Typical cost** | $35 | $55 | $75 |

As with microcontrollers, SBC device capabilities can be expanded through the addition of stackable expansion boards known as *hats* on Raspberry Pi and *capes* on BeagleBone Black, and through the addition of external modules, such as motor controllers or analog-to-digital converters, to mitigate limitations with the built-in device capabilities.

Many SBC devices are more like a mini-PC, and run an embedded operating system, typically a streamlined Linux distribution. As a result, there are many more development tools and language choices that are available for developing embedded applications that work with the attached sensors and actuators on these devices than on microcontroller boards. However, SBCs are more complex to set up, larger, more power hungry, and more prone to problems like corruption of the SD card or flash memory where applications are stored.

## Choosing between microcontroller development boards and single-board computers

Although off-the-shelf microcontroller development boards and single-board computers might only get you part of the way towards a fully realized IoT solution, they are great for bootstrapping its development.

One way to get started is to consider the key IoT device characteristics in light of your application's requirements, and then work through the following design decisions:

- Determine the type and number of peripheral sensors and output components that you need, and, if necessary, any design circuits for these components
- Select a microcontroller or single-board device to co-ordinate reading from and controlling the peripheral components

- Decide on the data communication protocols that you need to use for intra-device communication (for example, using I2C for communication between the microcontroller and any attached sensors)
- Select the networking hardware and protocols that you need to use to communicate with cloud services and apps

For example, for setting up a home automation system on a budget, I'd choose the Raspberry Pi Zero W, because it is a small and very low-cost SBC device (around $10), with ample processing power and memory (1GHz ARM6 processor and 512 MB RAM) for performing data processing and analytics on the device. It supports microSD card flash memory expansion up to 64GB for storing programs and data. And, it is equipped with a full 40-pin GPIO header, just like the Raspberry Pi 3, which allows connecting multiple sensors and supports both SPI and I2C protocols. It has on-board wifi for connecting with a home network, and it can be powered with micro-USB off a portable power pack or wall power supply.

As you progress further with prototyping your IoT devices and embedded software, as well as the upstream services and apps, you can periodically assess your prototypes against your functional and non-functional requirements including performance, reliability, and security, and revisit these choices as necessary.

## IoT hardware requirements for deploying your IoT project

IoT devices are highly specialized and are designed to operate within very specific contexts and environments, so the hardware requirements for IoT projects vary widely. While you might start by prototyping using generic off-the-shelf hardware, as you progress through this iterative design and requirements validation process, eventually you can move toward designing and developing custom PCBs and components that are tailored to your requirements, which you deploy in your production IoT solutions. As part of this process, you will need to consider these kinds of hardware requirements:

- Security requirements
- Ease of development
- Data acquisition, processing and storage requirements
- Connectivity requirements
- Power requirements
- Physical device design
- Cost requirements

### Security requirements

Security is a critical element within IoT and must be considered at all stages of design and development. The integrity and security of the data that is captured by the device must remain intact, even during prototyping. Security requirements relate to the security of the IoT devices themselves, hardening of the network, and the security of related cloud services and mobile and web applications.

Related security requirements include:

- Ensuring that each device has enough processing power and memory to be able to encrypt and decrypt data and messages at the rate that they are sent and received
- Ensuring that the embedded software development libraries support whatever authorization and access control mechanisms are used to authenticate with upstream services and apps
- Choosing to adopt off-the-shelf devices that implement device management protocols for securely registering new devices as they are added to a network to avoid spoofing, and those that include firmware capabilities to support secure over the air updates for security patches

## Ease of development

While prototyping, ease of development is another high priority requirement so that you can quickly and easily get your IoT device up and running, capturing data, and communicating with other devices and the cloud.

Consider the accessibility, availability, and quality of API documentation, development tools, and support offered by the hardware manufacturer or by the development community. Select devices that are quick and easy to program and re-flash, as well as being low touch to deploy, with zero or minimal per-device configuration required, to cut down on frustration and save time while you are developing your IoT solution.

## Data acquisition, processing, and storage requirements

The number of sensors that are connected, the resolution of the data that is captured, and the rate at which the data is sampled all determine the volume of data to be processed, which impacts on data processing and storage requirements.

The amount of data that needs to be retained on a device is dependent on how frequently the device connects to transmit data upstream. A wired, always-connected device that is installed in a smart building, one that streams low volumes of raw data directly to a highly available server, will require less data processing power and storage compared to a device that needs to process large volumes of data in bursts. A device that only connects every few hours to conserve power will require more storage to log data locally in the interim.

## Connectivity requirements

Connectivity requirements for wireless networking include operating range, or how far the signal will need to be transmitted, as well as the anticipated volume and rate of data to be transmitted. Consider fault-tolerance and the ability for a device to reconnect and retry sending data after it was disconnected.

Your hardware might have integrated network connectivity like Bluetooth or wifi, or this capability might need to be added with an expansion board or module. An external module that can be upgraded can provide more flexibility, as you have the option to try different modules to evaluate their range and power consumption.

## Power requirements

Many of the other requirements, including the number of sensors that you need and the rate of network transmission, will have an impact on the device's power requirements. Consider

whether your device will be wired, or whether it will require a portable power source like a battery or supercapacitor. If it requires a battery, you need to know the size, weight, and capacity requirements for the battery, as well as whether the battry should be rechargeable, replaceable, or whether the device should be discarded after the battery dies. If the device is rechargeable, how often should be charged, and by what means?

### Physical device design requirements

The physical device design requirements include the appearance and size of the device.

The environmental conditions in which the device will be installed also need to be considered, for example, will it need a waterproof or ruggedized enclosure? For example, a device that is installed on the underside of a truck as part of a fleet monitoring application would need to be shielded to ensure it continued to operate under harsh conditions; it would need to be waterproof and resistant to dirt, shock, and vibration.

### Cost requirements

The cost of the hardware includes the initial outlay for the hardware and associated components (such as any sensors) as well as their on-going operating costs, such as power and maintenance costs in the form of replacing worn parts or defective components. You might also need to pay ongoing licensing fees for some components or device drivers. Purchasing a handful of commercially available off-the-shelf development boards or SBCs might be more affordable than producing tiny runs of custom boards in the early stages of development; however, as you begin to scale up into dozens or hundreds of devices, dedicated hardware devices might become a better value proposition.

## Conclusion

There is no one-size-fits all approach to selecting hardware for IoT projects. Adopting standards-based, commodity hardware like microcontrollers or single-board-computers can save time and expense in the early stages of development, without sacrificing flexibility. What you learn in the prototyping phase can help you use critical hardware design decisions when you move into deploying your IoT solution.