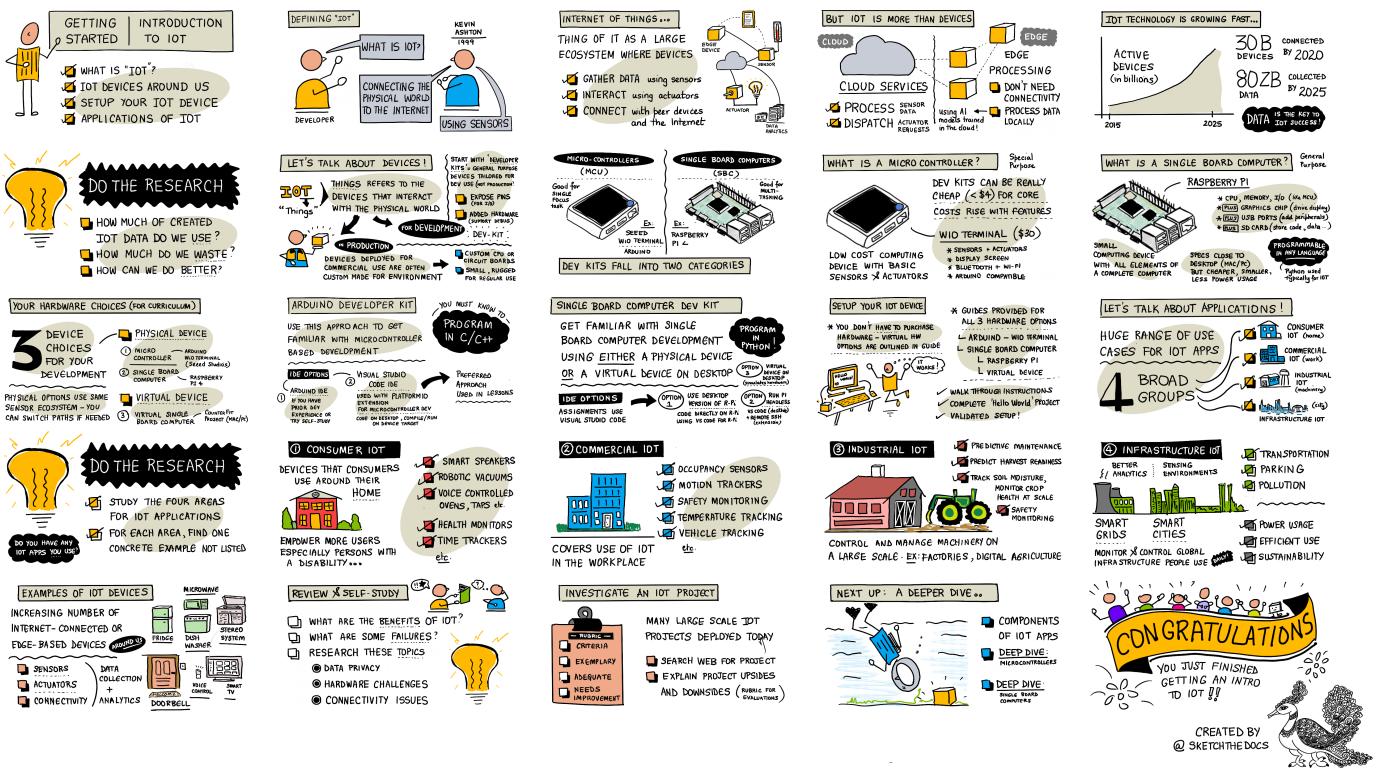


Introduction to IoT



Sketchnote by [Nitya Narasimhan](#). Click the image for a larger version.

Pre-lecture quiz

Pre-lecture quiz

Introduction

This lesson covers some of the introductory topics around the Internet of Things, and gets you going setting up your hardware.

In this lesson we'll cover:

- What is the 'Internet of Things'?
- IoT devices
- Set up your device

- [Applications of IoT](#)
- [Examples of IoT devices you may have around you](#)

What is the 'Internet of Things'?

The term 'Internet of Things' was coined by [Kevin Ashton](#) in 1999, to refer to connecting the Internet to the physical world via sensors. Since then, the term has been used to describe any device that interacts with the physical world around it, either by gathering data from sensors, or providing real-world interactions via actuators (devices that do something like turn on a switch or light an LED), generally connected to other devices or the Internet.

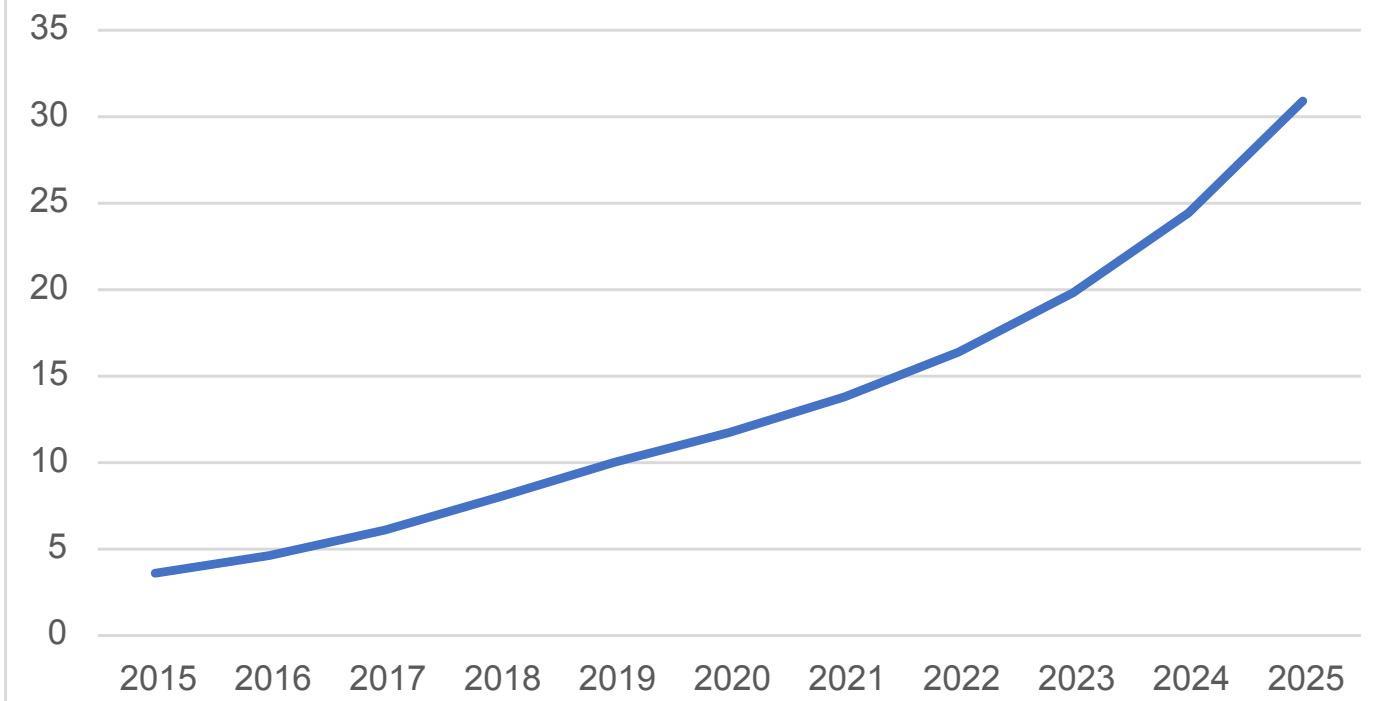
Sensors gather information from the world, such as measuring speed, temperature or location.

Actuators convert electrical signals into real-world interactions such as triggering a switch, turning on lights, making sounds, or sending control signals to other hardware, for example, to turn on a power socket.

IoT as a technology area is more than just devices - it includes cloud-based services that can process the sensor data, or send requests to actuators connected to IoT devices. It also includes devices that don't have or don't need Internet connectivity, often referred to as edge devices. These are devices that can process and respond to sensor data themselves, usually using AI models trained in the cloud.

IoT is a fast growing technology field. It is estimated that by the end of 2020, 30 billion IoT devices were deployed and connected to the Internet. Looking to the future, it is estimated that by 2025, IoT devices will be gathering almost 80 zettabytes of data or 80 trillion gigabytes. That's a lot of data!

Active devices (billions)



Do a little research: How much of the data generated by IoT devices is actually used, and how much is wasted? Why is so much data ignored?

This data is the key to IoT's success. To be a successful IoT developer, you need to understand the data you need to gather, how to gather it, how to make decisions based on it, and how to use those decisions to interact with the physical world if needed.

IoT devices

The **T** in IoT stands for **Things** - devices that interact with the physical world around them either by gathering data from sensors or providing real-world interactions via actuators.

Devices for production or commercial use, such as consumer fitness trackers, or industrial machine controllers, are usually custom-made. They use custom circuit boards, maybe even custom processors, designed to meet the needs of a particular task, whether that's being small enough to fit on a wrist, or rugged enough to work in a high temperature, high stress or high vibration factory environment.

As a developer either learning about IoT or creating a device prototype, you'll need to start with a developer kit. These are general-purpose IoT devices designed for developers to use, often with features that you wouldn't have on a production device, such as a set of external pins to connect sensors or actuators to, hardware to support debugging, or additional resources that would add unnecessary cost when doing a large manufacturing run.

These developer kits usually fall into two categories - microcontrollers and single-board computers. These will be introduced here, and we'll go into more detail in the next lesson.

 Your phone can also be considered to be a general-purpose IoT device, with sensors and actuators built-in, with different apps using the sensors and actuators in different ways with different cloud services. You can even find some IoT tutorials that use a phone app as an IoT device.

Microcontrollers

A microcontroller (also referred to as an MCU, short for microcontroller unit) is a small computer consisting of:

-  One or more central processing units (CPUs) - the 'brain' of the microcontroller that runs your program
-  Memory (RAM and program memory) - where your program, data and variables are stored
-  Programmable input/output (I/O) connections - to talk to external peripherals (connected devices) such as sensors and actuators

Microcontrollers are typically low cost computing devices, with average prices for the ones used in custom hardware dropping to around US\$0.50, and some devices as cheap as US\$0.03. Developer kits can start as low as US\$4, with costs rising as you add more features. The [Wio Terminal](#), a microcontroller developer kit from [Seeed studios](#) that has sensors, actuators, WiFi and a screen costs around US\$30.



💡 When searching the Internet for microcontrollers, be wary of searching for the term **MCU** as this will bring back a lot of results for the Marvel Cinematic Universe, not microcontrollers.

Microcontrollers are designed to be programmed to do a limited number of very specific tasks, rather than being general-purpose computers like PCs or Macs. Except for very specific scenarios, you can't connect a monitor, keyboard and mouse and use them for general purpose tasks.

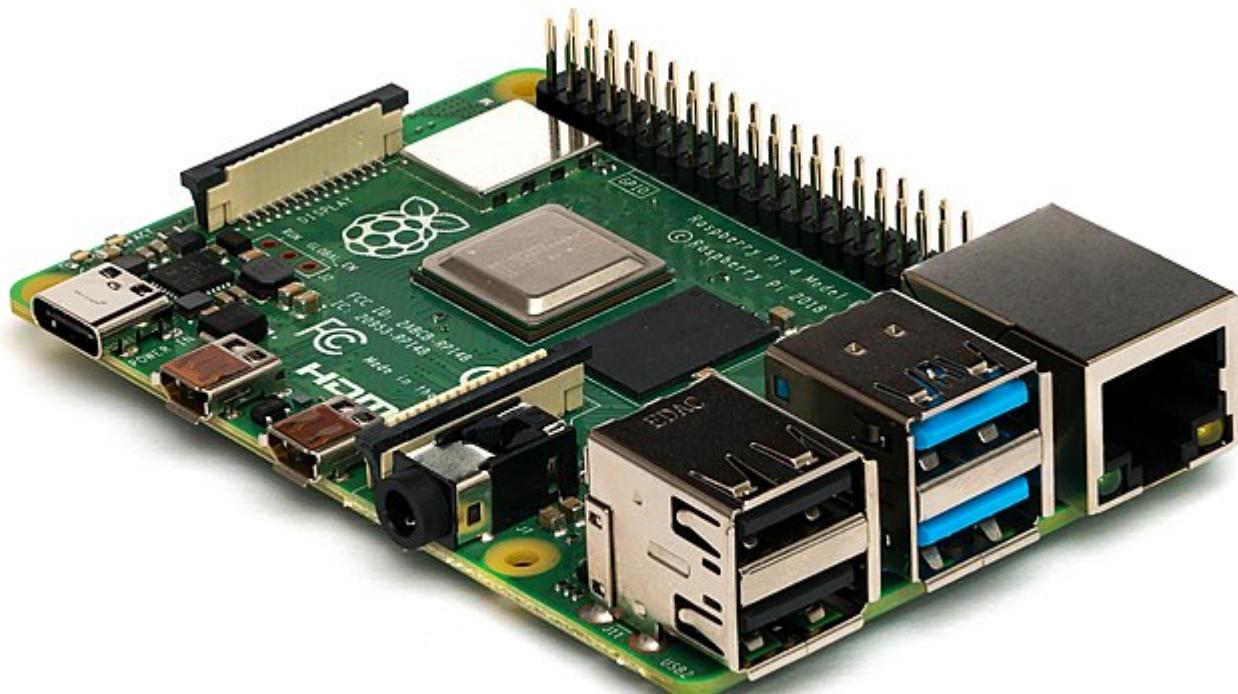
Microcontroller developer kits usually come with additional sensors and actuators on board. Most boards will have one or more LEDs you can program, along with other devices such as standard plugs for adding more sensors or actuators using various manufacturers' ecosystems or built-in sensors (usually the most popular ones such as temperature sensors). Some microcontrollers have built-in wireless connectivity such as Bluetooth or WiFi or have additional microcontrollers on the board to add this connectivity.

💡 Microcontrollers are usually programmed in C/C++.

Single-board computers

A single-board computer is a small computing device that has all the elements of a complete computer contained on a single small board. These are devices that have specifications close to a

desktop or laptop PC or Mac, run a full operating system, but are small, use less power, and are substantially cheaper.



Raspberry Pi 4. Michael Henzler / [Wikimedia Commons](#) / CC BY-SA 4.0

The Raspberry Pi is one of the most popular single-board computers.

Like a microcontroller, single-board computers have a CPU, memory and input/output pins, but they have additional features such as a graphics chip to allow you to connect monitors, audio outputs, and USB ports to connect keyboards mice and other standard USB devices like webcams or external storage. Programs are stored on SD cards or hard drives along with an operating system, instead of a memory chip built into the board.

 You can think of a single-board computer as a smaller, cheaper version of the PC or Mac you are reading this on, with the addition of GPIO (general-purpose input/output) pins to interact with sensors and actuators.

Single-board computers are fully-featured computers, so can be programmed in any language. IoT devices are typically programmed in Python.

Hardware choices for the rest of the lessons

All the subsequent lessons include assignments using an IoT device to interact with the physical world and communicate with the cloud. Each lesson supports 3 device choices - Arduino (using a

Seeed Studios Wio Terminal), or a single-board computer, either a physical device (a Raspberry Pi 4) or a virtual single-board computer running on your PC or Mac.

You can read about the hardware needed to complete all the assignments in the [hardware guide](#).

 You don't need to purchase any IoT hardware to complete the assignments, you can do everything using a virtual single-board computer.

Which hardware you choose is up to you - it depends on what you have available either at home or in your school, and what programming language you know or plan to learn. Both hardware variants will use the same sensor ecosystem, so if you start down one path, you can change to the other without having to replace most of the kit. The virtual single-board computer will be the equivalent of learning on a Raspberry Pi, with most of the code transferrable to the Pi if you eventually get a device and sensors.

Arduino developer kit

If you are interested in learning microcontroller development, you can complete the assignments using an Arduino device. You will need a basic understanding of C/C++ programming, as the lessons will only teach code that is relevant to the Arduino framework, the sensors and actuators being used, and the libraries that interact with the cloud.

The assignments will use [Visual Studio Code](#) with the [PlatformIO extension for microcontroller development](#). You can also use the Arduino IDE if you are experienced with this tool, as instructions will not be provided.

Single-board computer developer kit

If you are interested in learning IoT development using single-board computers, you can complete the assignments using a Raspberry Pi, or a virtual device running on your PC or Mac.

You will need a basic understanding of Python programming, as the lessons will only teach code that is relevant to the sensors and actuators being used, and the libraries that interact with the cloud.

 If you want to learn to code in Python, check out the following two video series:

- [Python for beginners](#)
- [More Python for beginners](#)

The assignments will use [Visual Studio Code](#).

If you are using a Raspberry Pi, you can either run your Pi using the full desktop version of Raspberry Pi OS, and do all the coding directly on the Pi using [the Raspberry Pi OS version of VS Code](#), or run your Pi as a headless device and code from your PC or Mac using VS Code with the [Remote SSH extension](#) that allows you to connect to your Pi and edit, debug and run code as if you were coding on it directly.

If you use the virtual device option, you will code directly on your computer. Instead of accessing sensors and actuators, you will use a tool to simulate this hardware providing sensor values that you can define, and showing the results of actuators on screen.

Set up your device

Before you can get started with programming your IoT device, you will need to do a small amount of setup. Follow the relevant instructions below depending on which device you will be using.

 If you don't have a device yet, refer to the [hardware guide](#) to help decide which device you are going to use, and what additional hardware you need to purchase. You don't need to purchase hardware, as all the projects can be run on virtual hardware.

These instructions do include links to third-party websites from the creators of the hardware or tools you will be using. This is to ensure you are always using the most up-to-date instructions for the various tools and hardware.

Work through the relevant guide to set your device up and complete a 'Hello World' project. This will be the first step in creating an IoT nightlight over the 4 lessons in this getting started part.

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Applications of IoT

IoT covers a huge range of use cases, across a few broad groups:

- Consumer IoT
- Commercial IoT
- Industrial IoT
- Infrastructure IoT

Do a little research: For each of the areas described below, find one concrete example that's not given in the text.

Consumer IoT

Consumer IoT refers to IoT devices that consumers will buy and use around the home. Some of these devices are incredibly useful, such as smart speakers, smart heating systems and robotic vacuum cleaners. Others are questionable in their usefulness, such as voice-controlled taps that then mean you cannot turn them off as the voice control cannot hear you over the sound of running water.

Consumer IoT devices are empowering people to achieve more in their surroundings, especially the 1 billion who have a disability. Robotic vacuum cleaners can provide clean floors to people with mobility issues who cannot vacuum themselves, voice-controlled ovens allow people with limited vision or motor control to heat their ovens with only their voice, health monitors can allow patients to monitor chronic conditions themselves with more regular and more detailed updates on their conditions. These devices are becoming so ubiquitous that even young children are using them as part of their daily lives, for example, students doing virtual schooling during the COVID pandemic setting timers on smart home devices to track their schoolwork or alarms to remind them of upcoming class meetings.

What consumer IoT devices do you have on your person or in your home?

Commercial IoT

Commercial IoT covers the use of IoT in the workplace. In an office setting, there may be occupancy sensors and motion detectors to manage lighting and heating to only keep the lights and heat off when not needed, reducing cost and carbon emissions. In a factory, IoT devices can monitor for safety hazards such as workers not wearing hard hats or noise that has reached dangerous levels. In retail, IoT devices can measure the temperature of cold storage, alerting the shop owner if a fridge or freezer is outside the required temperature range, or they can monitor items on shelves to direct employees to refill produce that has been sold. The transport industry is relying more and more on IoT to monitor vehicle locations, track on-road mileage for road user charging, track driver hours and break compliance, or notify staff when a vehicle is approaching a depot to prepare for loading or unloading.

What commercial IoT devices do you have in your school or workplace?

Industrial IoT (IIoT)

Industrial IoT, or IIoT, is the use of IoT devices to control and manage machinery on a large scale. This covers a wide range of use cases, from factories to digital agriculture.

Factories use IoT devices in many different ways. Machinery can be monitored with multiple sensors to track things like temperature, vibration and rotation speed. This data can then be monitored to allow the machine to be stopped if it goes outside of certain tolerances - it runs too hot and gets shut down for example. This data can also be gathered and analyzed over time to do predictive maintenance, where AI models will look at the data leading up to a failure, and use that to predict other failures before they happen.

Digital agriculture is important if the planet is to feed the growing population, especially for the 2 billion people in 500 million households that survive on subsistence farming. Digital agriculture can range from a few single digit dollar sensors to massive commercial setups. A farmer can start by monitoring temperatures and using growing degree days to predict when a crop will be ready for harvest. They can connect soil moisture monitoring to automated watering systems to give their plants as much water as is needed, but no more to ensure their crops don't dry out without wasting water. Farmers are even taking it further and using drones, satellite data and AI to monitor crop growth, disease and soil quality over huge areas of farmland.

- What other IoT devices could help farmers?

Infrastructure IoT

Infrastructure IoT is monitoring and controlling the local and global infrastructure that people use every day.

Smart Cities are urban areas that use IoT devices to gather data about the city and use that to improve how the city runs. These cities are usually run with collaborations between local governments, academia and local businesses, tracking and managing things varying from transport to parking and pollution. For example, in Copenhagen, Denmark, air pollution is important to the local residents, so it is measured and the data is used to provide information on the cleanest cycling and jogging routes.

Smart power grids allow better analytics of power demand by gathering usage data at the level of individual homes. This data can guide decisions at a country level including where to build new power stations, and at a personal level by giving users insights into how much power they are using, when they are using it, and even suggestions on how to reduce costs, such as charging electric cars at night.

- If you could add IoT devices to measure anything where you live, what would it be?

Examples of IoT devices you may have around you

You'd be amazed by just how many IoT devices you have around you. I'm writing this from home and I have the following devices connected to the Internet with smart features such as app control, voice control, or the ability to send data to me via my phone:

- Multiple smart speakers
- Fridge, dishwasher, oven and microwave
- Electricity monitor for solar panels
- Smart plugs
- Video doorbell and security cameras
- Smart thermostat with multiple smart room sensors
- Garage door opener
- Home entertainment systems and voice-controlled TVs
- Lights
- Fitness and health trackers

All these types of devices have sensors and/or actuators and talk to the Internet. I can tell from my phone if my garage door is open, and ask my smart speaker to close it for me. I can even set it to a timer so if it's still open at night, it will close automatically. When my doorbell rings, I can see from my phone who is there wherever I am in the world, and talk to them via a speaker and microphone built into the doorbell. I can monitor my blood glucose, heart rate and sleep patterns, looking for patterns in the data to improve my health. I can control my lights via the cloud, and sit in the dark when my Internet connection goes down.

Challenge

List as many IoT devices as you can that are in your home, school or workplace - there may be more than you think!

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

Read up on the benefits and failures of consumer IoT projects. Check news sites for articles on when it has gone wrong, such as privacy issues, hardware problems or problems caused by lack of connectivity.

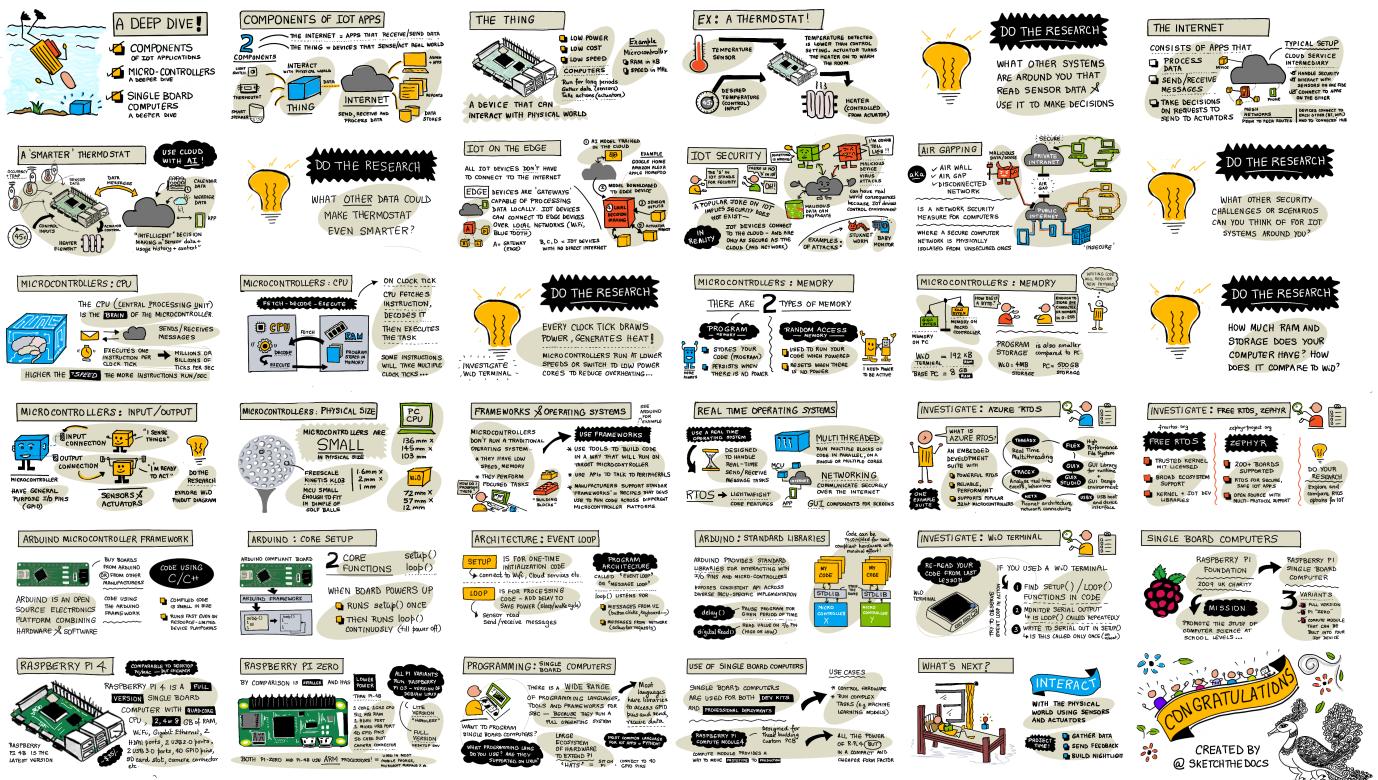
Some examples:

- Check out the Twitter account [Internet of Sh*t \(profanity warning\)](#) for some good examples of failures with consumer IoT.
- [c|net - My Apple Watch saved my life: 5 people share their stories](#)
- [c|net - ADT technician pleads guilty to spying on customer camera feeds for years \(trigger warning - non-consensual voyeurism\)](#)

Assignment

Investigate an IoT project

A deeper dive into IoT



Sketchnote by [Nitya Narasimhan](#). Click the image for a larger version.

Pre-lecture quiz

Pre-lecture quiz

Introduction

This lesson dives deeper into some of the concepts covered in the last lesson.

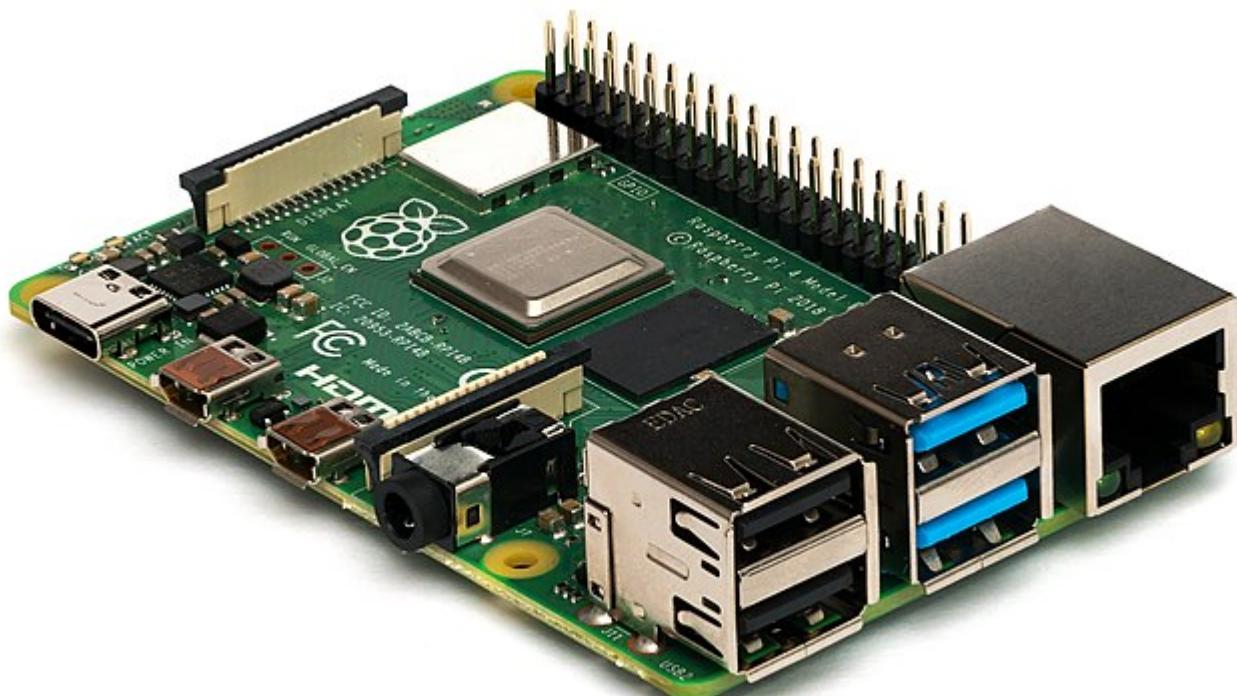
In this lesson we'll cover:

- Components of an IoT application
- Deeper dive into microcontrollers
- Deeper dive into single-board computers

Components of an IoT application

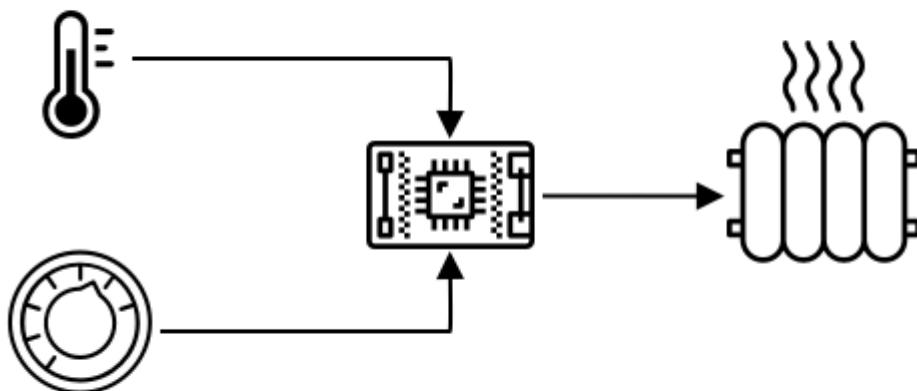
The two components of an IoT application are the *Internet* and the *thing*. Let's look at these two components in a bit more detail.

The Thing



The **Thing** part of IoT refers to a device that can interact with the physical world. These devices are usually small, low-priced computers, running at low speeds and using low power - for example, simple microcontrollers with kilobytes of RAM (as opposed to gigabytes in a PC) running at only a few hundred megahertz (as opposed to gigahertz in a PC), but consuming sometimes so little power they can run for weeks, months or even years on batteries.

These devices interact with the physical world, either by using sensors to gather data from their surroundings or by controlling outputs or actuators to make physical changes. The typical example of this is a smart thermostat - a device that has a temperature sensor, a means to set a desired temperature such as a dial or touchscreen, and a connection to a heating or cooling system that can be turned on when the temperature detected is outside the desired range. The temperature sensor detects that the room is too cold and an actuator turns the heating on.



A simple thermostat. Temperature by Vectors Market / Microcontroller by Template / dial by Jamie Dickinson / heater by Pascal Heß - all from the [Noun Project](#)

There are a huge range of different things that can act as IoT devices, from dedicated hardware that senses one thing, to general purpose devices, even your smartphone! A smartphone can use sensors to detect the world around it, and actuators to interact with the world - for example using a GPS sensor to detect your location and a speaker to give you navigation instructions to a destination.

- Think of other systems you have around you that read data from a sensor and use that to make decisions. One example would be the thermostat on an oven. Can you find more?

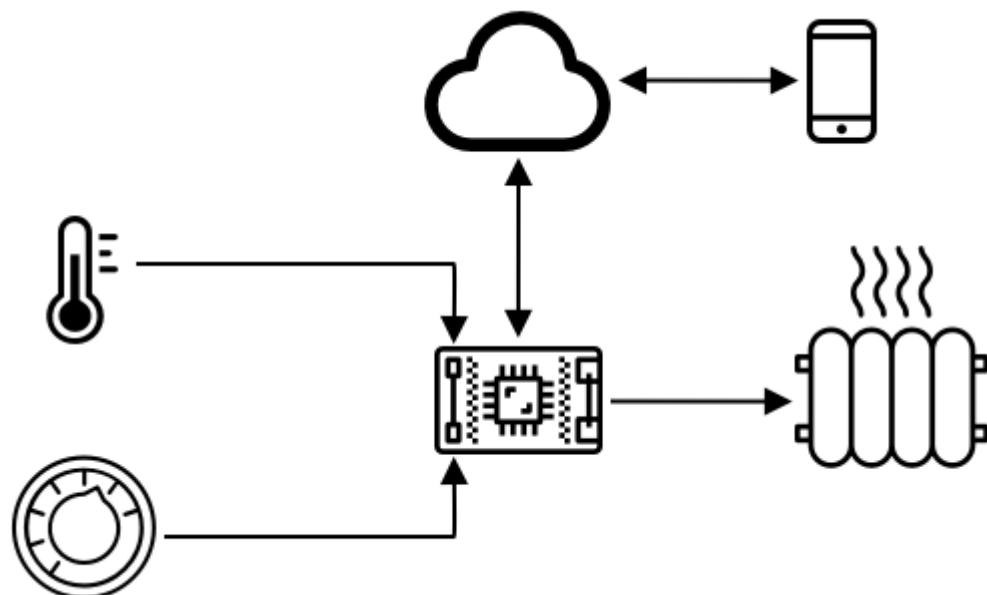
The Internet

The **Internet** side of an IoT application consists of applications that the IoT device can connect to send and receive data, as well as other applications that can process the data from the IoT device and help make decisions on what requests to send to the IoT devices actuators.

One typical setup would be having some kind of cloud service that the IoT device connects to, and this cloud service handles things like security, as well as receiving messages from the IoT device, and sending messages back to the device. This cloud service would then connect to other applications that can process or store sensor data, or use the sensor data with data from other systems to make decisions.

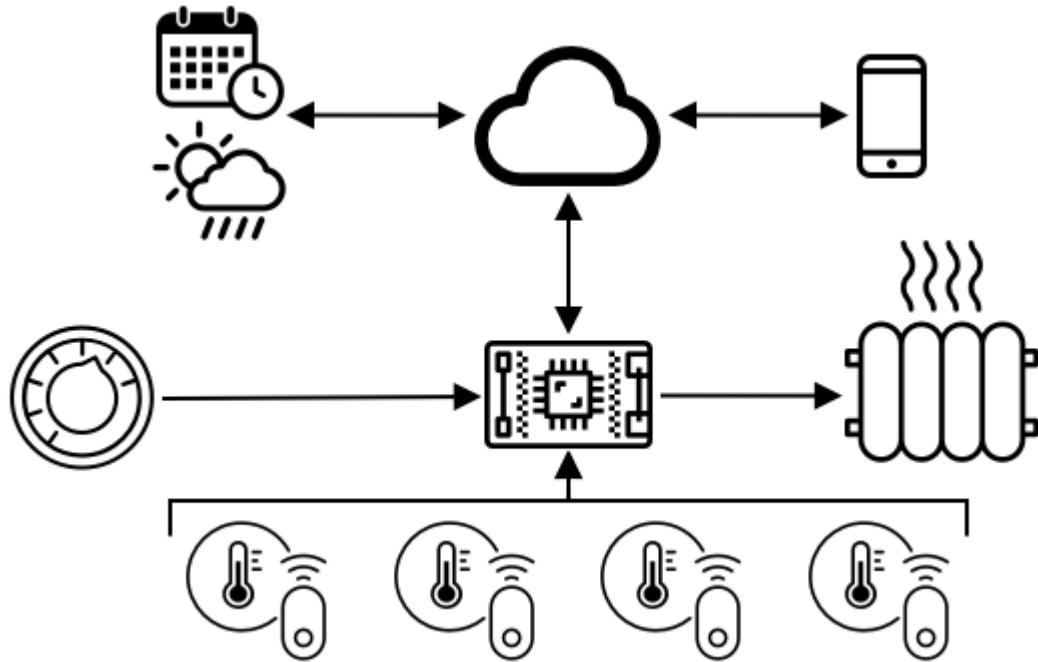
Devices also don't always connect directly to the Internet themselves via WiFi or wired connections. Some devices use mesh networking to talk to each other over technologies such as Bluetooth, connecting via a hub device that has an Internet connection.

With the example of a smart thermostat, the thermostat would connect using home WiFi to a cloud service running in the cloud. It would send the temperature data to this cloud service, and from there it will be written to a database of some kind allowing the homeowner to check the current and past temperatures using a phone app. Another service in the cloud would know what temperature the homeowner wants, and send messages back to the IoT device via the cloud service to tell the heating system to turn on or off.



An Internet connected thermostat with mobile app control. Temperature by Vectors Market / Microcontroller by Template / dial by Jamie Dickinson / heater by Pascal Heß / mobile phone by Alice-vector / Cloud by Debi Alpa Nugraha - all from the [Noun Project](#)

An even smarter version could use AI in the cloud with data from other sensors connected to other IoT devices such as occupancy sensors that detect what rooms are in use, as well as data such as weather and even your calendar, to make decisions on how to set the temperature in a smart fashion. For example, it could turn your heating off if it reads from your calendar you are on vacation, or turn off the heating on a room by room basis depending on what rooms you use, learning from the data to be more and more accurate over time.



An Internet connected thermostat using multiple room sensors, with mobile app control, as well as intelligence from weather and calendar data. Temperature by Vectors Market / Microcontroller by Template / dial by Jamie Dickinson / heater by Pascal Heß / mobile phone and Calendar by Alice-vector / Cloud by Debi Alpa Nugraha / smart sensor by Andrei Yushchenko / weather by Adrien Coquet - all from the [Noun Project](#)

- What other data could help make an Internet connected thermostat smarter?

IoT on the Edge

Although the I in IoT stands for Internet, these devices don't have to connect to the Internet. In some cases, devices can connect to 'edge' devices - gateway devices that run on your local network meaning you can process data without making a call over the Internet. This can be faster when you have a lot of data or a slow Internet connection, it allows you to run offline where Internet connectivity is not possible such as on a ship or in a disaster area when responding to a humanitarian crisis, and allows you to keep data private. Some devices will contain processing code created using cloud tools and run this locally to gather and respond to data without using an Internet connection to make a decision.

One example of this is a smart home device such as an Apple HomePod, Amazon Alexa, or Google Home, which will listen to your voice using AI models trained in the cloud, but running locally on the device. These devices will 'wake up' when a certain word or phrase is spoken, and only then send your speech over the Internet for processing. The device will stop sending speech at an appropriate point such as when it detects a pause in your speech. Everything you say before waking up the device with the wake word, and everything you say after the device has stopped listening will not be sent over the internet to the device provider, and therefore will be private.

- Think of other scenarios where privacy is important so processing of data would be better done on the edge rather than in the cloud. As a hint - think about IoT devices with cameras or other

imaging devices on them.

IoT Security

With any Internet connection, security is an important consideration. There is an old joke that 'the S in IoT stands for Security' - there is no 'S' in IoT, implying it is not secure.

IoT devices connect to a cloud service, and therefore are only as secure as that cloud service - if your cloud service allows any device to connect then malicious data can be sent, or virus attacks can take place. This can have very real world consequences as IoT devices interact and control other devices. For example, the [Stuxnet worm](#) manipulated valves in centrifuges to damage them. Hackers have also taken advantage of [poor security to access baby monitors](#) and other home surveillance devices.

 Sometimes IoT devices and edge devices run on a network completely isolated from the Internet to keep the data private and secure. This is known as [air-gapping](#).

Deeper dive into microcontrollers

In the last lesson, we introduced microcontrollers. Let's now look deeper into them.

CPU

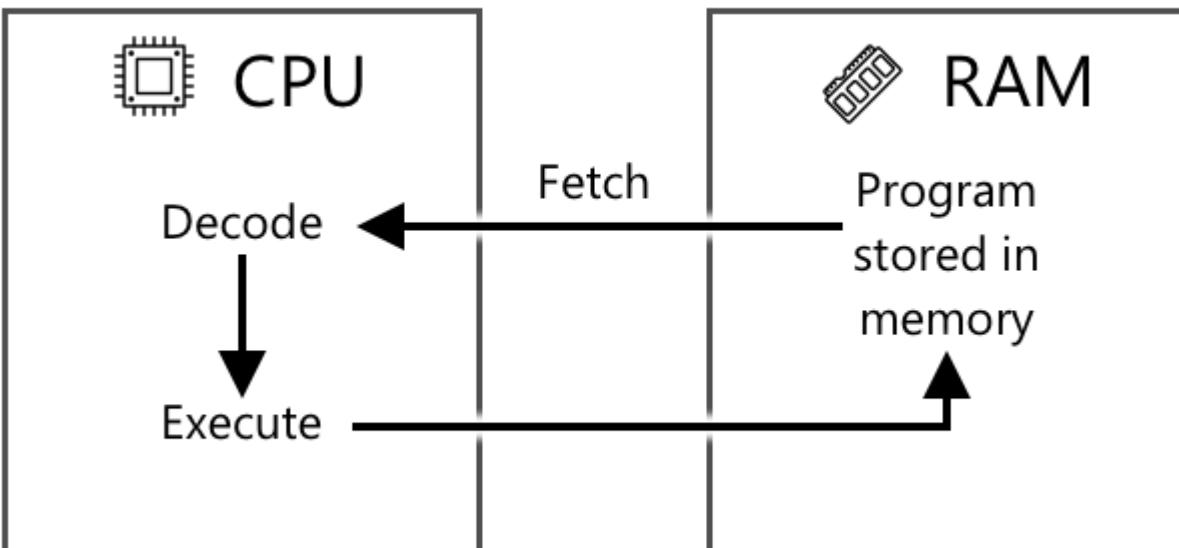
The CPU is the 'brain' of the microcontroller. It is the processor that runs your code and can send data to and receive data from any connected devices. CPUs can contain one or more cores - essentially one or more CPUs that can work together to run your code.

CPUs rely on a clock to tick many millions or billions of times a second. Each tick, or cycle, synchronizes the actions that the CPU can take. With each tick, the CPU can execute an instruction from a program, such as to retrieve data from an external device or perform a mathematical calculation. This regular cycle allows for all actions to be completed before the next instruction is processed.

The faster the clock cycle, the more instructions that can be processed each second, and therefore the faster the CPU. CPU speeds are measured in [Hertz \(Hz\)](#), a standard unit where 1 Hz means one cycle or clock tick per second.

 CPU speeds are often given in MHz or GHz. 1MHz is 1 million Hz, 1GHz is 1 billion Hz.

 CPUs execute programs using the fetch-decode-execute cycle. For every clock tick, the CPU will fetch the next instruction from memory, decode it, then execute it such as using an arithmetic logic unit (ALU) to add 2 numbers. Some executions will take multiple ticks to run, so the next cycle will run at the next tick after the instruction has completed.



CPU by Icon Lauk / ram by Atif Arshad - all from the [Noun Project](#)

Microcontrollers have much lower clock speeds than desktop or laptop computers, or even most smartphones. The Wio Terminal for example has a CPU that runs at 120MHz or 120,000,000 cycles per second.

- An average PC or Mac has a CPU with multiple cores running at multiple GigaHertz, meaning the clock ticks billions of times a second. Research the clock speed of your computer and compare how many times faster it is than the Wio terminal.

Each clock cycle draws power and generates heat. The faster the ticks, the more power consumed and more heat generated. PC's have heat sinks and fans to remove heat, without which they would overheat and shut down within seconds. Microcontrollers often have neither as they run much cooler and therefore much slower. PC's run off mains power or large batteries for a few hours, microcontrollers can run for days, months, or even years off small batteries. Microcontrollers can also have cores that run at different speeds, switching to slower low power cores when the demand on the CPU is low to reduce power consumption.

 Some PCs and Macs are adopting the same mix of fast high power cores and slower low power cores, switching to save battery. For example, the M1 chip in the latest Apple laptops can switch between 4 performance cores and 4 efficiency cores to optimize battery life or speed depending on the task being run.

- Do a little research: Read up on CPUs on the [Wikipedia CPU article](#)

Task

Investigate the Wio Terminal.

If you are using a Wio Terminal for these lessons, try to find the CPU. Find the *Hardware Overview* section of the [Wio Terminal product page](#) for a picture of the internals, and try to find the CPU through the clear plastic window on the back.

Memory

Microcontrollers usually have two types of memory - program memory and random-access memory (RAM).

Program memory is non-volatile, which means whatever is written to it stays when there is no power to the device. This is the memory that stores your program code.

RAM is the memory used by the program to run, containing variables allocated by your program and data gathered from peripherals. RAM is volatile, when the power goes out the contents are lost, effectively resetting your program.

 Program memory stores your code and stays when there is no power.

 RAM is used to run your program and is reset when there is no power

Like with the CPU, the memory on a microcontroller is orders of magnitude smaller than a PC or Mac. A typical PC might have 8 Gigabytes (GB) of RAM, or 8,000,000,000 bytes, with each byte enough space to store a single letter or a number from 0-255. A microcontroller would have only Kilobytes (KB) of RAM, with a kilobyte being 1,000 bytes. The Wio terminal mentioned above has 192KB of RAM, or 192,000 bytes - more than 40,000 times less than an average PC!

The diagram below shows the relative size difference between 192KB and 8GB - the small dot in the center represents 192KB.



Program storage is also smaller than a PC. A typical PC might have a 500GB hard drive for program storage, whereas a microcontroller might have only kilobytes or maybe a few megabytes (MB) of storage (1MB is 1,000KB, or 1,000,000 bytes). The Wio terminal has 4MB of program storage.

- Do a little research: How much RAM and storage does the computer you are using to read this have? How does this compare to a microcontroller?

Input/Output

Microcontrollers need input and output (I/O) connections to read data from sensors and send control signals to actuators. They usually contain a number of general-purpose input/output (GPIO) pins. These pins can be configured in software to be input (that is they receive a signal), or output (they send a signal).

  Input pins are used to read values from sensors

  Output pins send instructions to actuators

- You'll learn more about this in a subsequent lesson.

Task

Investigate the Wio Terminal.

If you are using a Wio Terminal for these lessons, find the GPIO pins. Find the *Pinout diagram* section of the [Wio Terminal product page](#) to learn which pins are which. The Wio Terminal comes with a sticker you can mount on the back with pin numbers, so add this now if you haven't already.

Physical size

Microcontrollers are typically small in size, with the smallest, a [Freescale Kinetis KL03 MCU](#) is small enough to fit in the dimple of a golf ball. Just the CPU in a PC can measure 40mm x 40mm, and that's not including the heat sinks and fans needed to ensure the CPU can run for more than a few seconds without overheating, substantially larger than a complete microcontroller. The Wio terminal developer kit with a microcontroller, case, screen and a range of connections and components isn't much bigger than a bare Intel i9 CPU, and substantially smaller than the CPU with a heat sink and fan!

Device	Size
Freescale Kinetis KL03	1.6mm x 2mm x 1mm
Wio terminal	72mm x 57mm x 12mm
Intel i9 CPU, Heat sink and fan	136mm x 145mm x 103mm

Frameworks and operating systems

Due to their low speed and memory size, microcontrollers don't run an operating system (OS) in the desktop sense of the word. The operating system that makes your computer run (Windows, Linux or macOS) needs a lot of memory and processing power to run tasks that are completely unnecessary for a microcontroller. Remember that microcontrollers are usually programmed to perform one or more very specific tasks, unlike a general purpose computer like a PC or Mac that needs to support a user interface, play music or movies, provide tools to write documents or code, play games, or browse the Internet.

To program a microcontroller without an OS you do need some tooling to allow you to build your code in a way that the microcontroller can run, using APIs that can talk to any peripherals. Each microcontroller is different, so manufacturers normally support standard frameworks which allow you to follow a standard 'recipe' to build your code and have it run on any microcontroller that supports that framework.

You can program microcontrollers using an OS - often referred to as a real-time operating system (RTOS), as these are designed to handle sending data to and from peripherals in real time. These operating systems are very lightweight and provide features such as:

- Multi-threading, allowing your code to run more than one block of code at the same time, either on multiple cores or by taking turns on one core
- Networking to allow communicating over the Internet securely
- Graphical user interface (GUI) components for building user interfaces (UI) on devices that have screens.

Read up on some different RTOSes: [Azure RTOS](#), [FreeRTOS](#), [Zephyr](#)

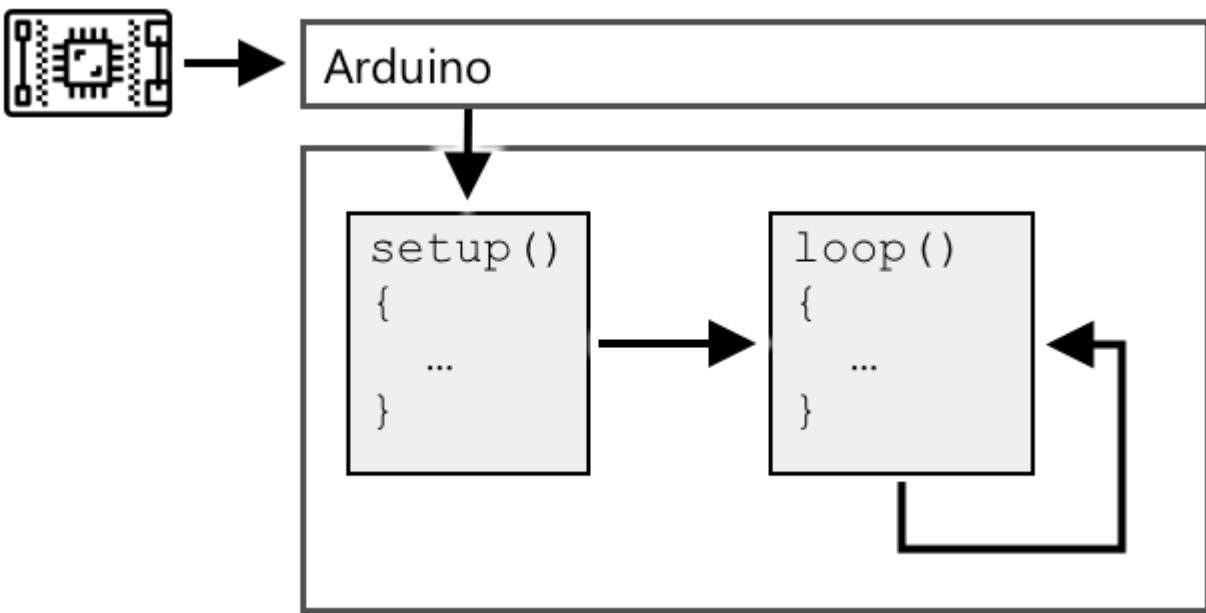
Arduino



Arduino is probably the most popular microcontroller framework, especially among students, hobbyists and makers. Arduino is an open source electronics platform combining software and hardware. You can buy Arduino compatible boards from Arduino themselves or from other manufacturers, then code using the Arduino framework.

Arduino boards are coded in C or C++. Using C/C++ allows your code to be compiled very small and run fast, something needed on a constrained device such as a microcontroller. The core of an Arduino application is referred to as a sketch and is C/C++ code with 2 functions - `setup` and `loop`. When the board starts up, the Arduino framework code will run the `setup` function once, then it will run the `loop` function again and again, running it continuously until the power is powered off.

You would write your setup code in the `setup` function, such as connecting to WiFi and cloud services or initializing pins for input and output. Your loop code would then contain processing code, such as reading from a sensor and sending the value to the cloud. You would normally include a delay in each loop, for example, if you only want sensor data to be sent every 10 seconds you would add a delay of 10 seconds at the end of the loop so the microcontroller can sleep, saving power, then run the loop again when needed 10 seconds later.



This program architecture is known as an *event loop* or *message loop*. Many applications use this under the hood and is the standard for most desktop applications that run on OSes like Windows, macOS or Linux. The `loop` listens for messages from user interface components such as buttons, or devices like the keyboard, and responds to them. You can read more in this [article on the event loop](#).

Arduino provides standard libraries for interacting with microcontrollers and the I/O pins, with different implementations under the hood to run on different microcontrollers. For example, the `delay` function will pause the program for a given period of time, the `digitalRead` function will read a value of `HIGH` or `LOW` from the given pin, regardless of which board the code is run on. These standard libraries mean that Arduino code written for one board can be recompiled for any other Arduino board and will run, assuming that the pins are the same and the boards support the same features.

There is a large ecosystem of third-party Arduino libraries that allow you to add extra features to your Arduino projects, such as using sensors and actuators or connecting to cloud IoT services.

Task

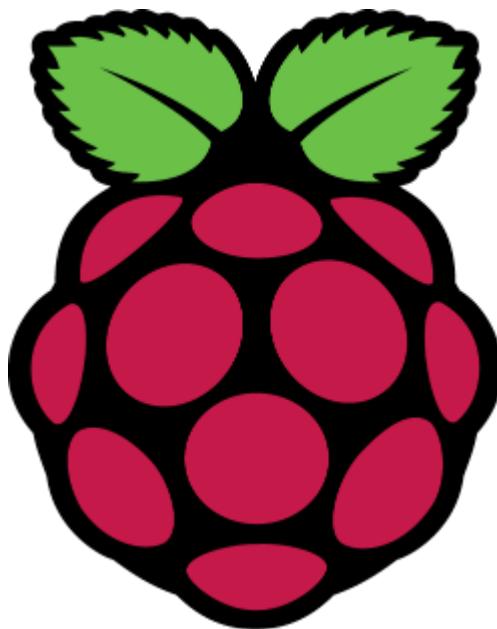
Investigate the Wio Terminal.

If you are using a Wio Terminal for these lessons, re-read the code you wrote in the last lesson. Find the `setup` and `loop` function. Monitor the serial output for the `loop` function being called repeatedly. Try adding code to the `setup` function to write to the serial port and observe that this code is only called once each time you reboot. Try rebooting your device with the power switch on the side to show this is called each time the device reboots.

Deeper dive into single-board computers

In the last lesson, we introduced single-board computers. Let's now look deeper into them.

Raspberry Pi



The [Raspberry Pi Foundation](#) is a charity from the UK founded in 2009 to promote the study of computer science, especially at school level. As part of this mission, they developed a single-board computer, called the Raspberry Pi. Raspberry Pis are currently available in 3 variants - a full size version, the smaller Pi Zero, and a compute module that can be built into your final IoT device.

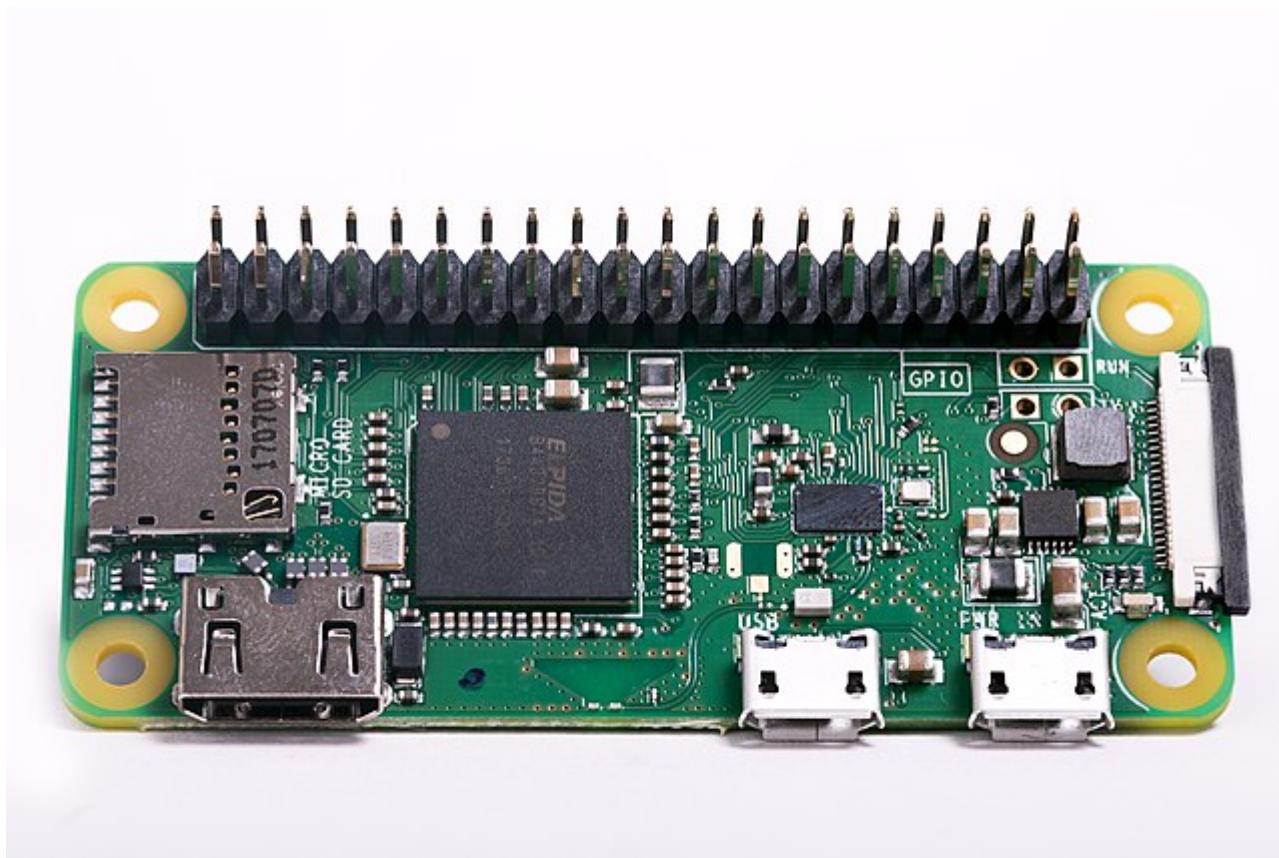


Raspberry Pi 4. Michael Henzler / [Wikimedia Commons](#) / CC BY-SA 4.0

The latest iteration of the full size Raspberry Pi is the Raspberry Pi 4B. This has a quad-core (4 core) CPU running at 1.5GHz, 2, 4, or 8GB of RAM, gigabit ethernet, WiFi, 2 HDMI ports supporting 4k

screens, an audio and composite video output port, USB ports (2 USB 2.0, 2 USB 3.0), 40 GPIO pins, a camera connector for a Raspberry Pi camera module, and an SD card slot. All this on a board that is 88mm x 58mm x 19.5mm and is powered by a 3A USB-C power supply. These start at US\$35, much cheaper than a PC or Mac.

💡 There is also a Pi400 all in one computer with a Pi4 built into a keyboard.



The Pi Zero is much smaller, with lower power. It has a single core 1GHz CPU, 512MB of RAM, WiFi (in the Zero W model), a single HDMI port, a micro-USB port, 40 GPIO pins, a camera connector for a Raspberry Pi camera module, and an SD card slot. It measures 65mm x 30mm x 5mm, and draws very little power. The Zero is US\$5, with the W version with WiFi US\$10.

🎓 The CPUs in both of these are ARM processors, as opposed to the Intel/AMD x86 or x64 processors you find in most PCs and Macs. These are similar to the CPUs you find in some microcontrollers, as well as nearly all mobile phones, the Microsoft Surface X, and the new Apple Silicon based Apple Macs.

All variants of the Raspberry Pi run a version of Debian Linux called Raspberry Pi OS. This is available as a lite version with no desktop, which is perfect for 'headless' projects where you don't need a screen, or a full version with a full desktop environment, with web browser, office applications, coding

tools and games. As the OS is a version of Debian Linux, you can install any application or tool that runs on Debian and is built for the ARM processor inside the Pi.

Task

Investigate the Raspberry Pi.

If you are using a Raspberry Pi for these lessons, read up about the different hardware components on the board.

- You can find details on the processors used on the [Raspberry Pi hardware documentation page](#).
Read up on the processor used in the Pi you are using.
- Locate the GPIO pins. Read more about them on the [Raspberry Pi GPIO documentation](#). Use the [GPIO Pin Usage guide](#) to identify the different pins on your Pi.

Programming single-board computers

Single-board computers are full computers, running a full OS. This means there is a wide range of programming languages, frameworks and tools you can use to code them, unlike microcontrollers which rely on support for the board in frameworks like Arduino. Most programming languages have libraries that can access the GPIO pins to send and receive data from sensors and actuators.

What programming languages are you familiar with? Are they supported on Linux?

The most common programming language for building IoT applications on a Raspberry Pi is Python. There is a huge ecosystem of hardware designed for the Pi, and nearly all of these include the relevant code needed to use them as Python libraries. Some of these ecosystems are based off 'hats' - so called because they sit on top of the Pi like a hat and connect with a large socket to the 40 GPIO pins. These hats provide additional capabilities, such as screens, sensors, remote controlled cars, or adapters to allow you to plug in sensors with standardized cables

Use of single-board computers in professional IoT deployments

Single-board computers are used for professional IoT deployments, not just as developer kits. They can provide a powerful way to control hardware and run complex tasks such as running machine learning models. For example, there is a [Raspberry Pi 4 compute module](#) that provides all the power of a Raspberry Pi 4 but in a compact and cheaper form factor without most of the ports, designed to be installed into custom hardware.

The challenge in the last lesson was to list as many IoT devices as you can that are in your home, school or workplace. For every device in this list, do you think they are built around microcontrollers or single-board computers, or even a mixture of both?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read the [Arduino getting started guide](#) to understand more about the Arduino platform.
 - Read the [introduction to the Raspberry Pi 4](#) to learn more about Raspberry Pis.
- Use these guides, along with the costs shown by following the links in the [hardware guide](#) to decide on what hardware platform you want to use, or if you would rather use a virtual device.

Assignment

[Compare and contrast microcontrollers and single-board computers](#)

Interact with the physical world with sensors and actuators

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

Introduction

This lesson introduces two of the important concepts for your IoT device - sensors and actuators. You will also get hands on with them both, adding a light sensor to your IoT project, then adding an LED controlled by light levels, effectively building a nightlight.

In this lesson we'll cover:

- [What are sensors?](#)
- [Use a sensor](#)
- [Sensor types](#)
- [What are actuators?](#)
- [Use an actuator](#)
- [Actuator types](#)

What are sensors?

Sensors are hardware devices that sense the physical world - that is they measure one or more properties around them and send the information to an IoT device. Sensors cover a huge range of devices as there are so many things that can be measured, from natural properties such as air temperature to physical interactions such as movement.

Some common sensors include:

- Temperature sensors - these sense the air temperature or the temperature of what they are immersed in. For hobbyists and developer, these are often combined with air pressure and humidity in a single sensor.
- Buttons - they sense when they have been pressed.
- Light sensors - these detect light levels and can be for specific colors, UV light, IR light, or general visible light.
- Cameras - these sense a visual representation of the world by taking a photograph or streaming video.
- Accelerometers - these sense movement in multiple directions.
- Microphones - these sense sound, either general sound levels or directional sound.

 Do some research. What sensors does your phone have?

All sensors have one thing in common - they convert whatever they sense into an electrical signal that can be interpreted by an IoT device. How this electrical signal is interpreted depends on the sensor, as well as the communication protocol used to communicate with the IoT device.

Use a sensor

Follow the relevant guide below to add a sensor to your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Sensor types

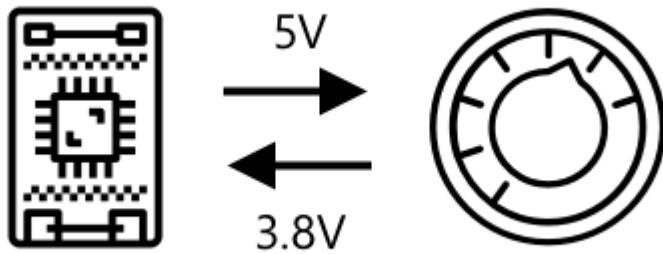
Sensors are either analog or digital.

Analog sensors

Some of the most basic sensors are analog sensors. These sensors receive a voltage from the IoT device, the sensor components adjust this voltage, and the voltage that is returned from the sensor is measured to give the sensor value.

 Voltage is a measure of how much push there is to move electricity from one place to another, such as from a positive terminal of a battery to the negative terminal. For example, a standard AA battery is 1.5V (V is the symbol for volts) and can push electricity with the force of 1.5V from its positive terminal to its negative terminal. Different electrical hardware requires different voltages to work, for example, an LED can light with between 2-3V, but a 100W filament lightbulb would need 240V. You can read more about voltage on the [Voltage page on Wikipedia](#).

One example of this is a potentiometer. This is a dial that you can rotate between two positions and the sensor measures the rotation.



A potentiometer. Microcontroller by Template / dial by Jamie Dickinson - all from the [Noun Project](#)

The IoT device will send an electrical signal to the potentiometer at a voltage, such as 5 volts (5V). As the potentiometer is adjusted it changes the voltage that comes out of the other side. Imagine you have a potentiometer labelled as a dial that goes from 0 to 11, such as a volume knob on an amplifier. When the potentiometer is in the full off position (0) then 0V (0 volts) will come out. When it is in the full on position (11), 5V (5 volts) will come out.

 This is an oversimplification, and you can read more on potentiometers and variable resistors on the [potentiometer Wikipedia page](#).

The voltage that comes out of the sensor is then read by the IoT device, and the device can respond to it. Depending on the sensor, this voltage can be an arbitrary value or can map to a standard unit. For example, an analog temperature sensor based on a [thermistor](#) changes its resistance depending on the temperature. The output voltage can then be converted to a temperature in Kelvin, and correspondingly into °C or °F, by calculations in code.

 What do you think happens if the sensor returns a higher voltage than was sent (for example coming from an external power supply)?  DO NOT test this out.

Analog to digital conversion

IoT devices are digital - they can't work with analog values, they only work with 0s and 1s. This means that analog sensor values need to be converted to a digital signal before they can be processed. Many IoT devices have analog-to-digital converters (ADCs) to convert analog inputs to digital representations of their value. Sensors can also work with ADCs via a connector board. For example, in the Seeed Grove ecosystem with a Raspberry Pi, analog sensors connect to specific ports on a 'hat' that sits on the Pi connected to the Pi's GPIO pins, and this hat has an ADC to convert the voltage into a digital signal that can be sent off the Pi's GPIO pins.

Imagine you have an analog light sensor connected to an IoT device that uses 3.3V and is returning a value of 1V. This 1V doesn't mean anything in the digital world, so needs to be converted. The voltage will be converted to an analog value using a scale depending on the device and sensor. One example is the Seeed Grove light sensor which outputs values from 0 to 1,023. For this sensor running at 3.3V, a 1V output would be a value of 300. An IoT device can't handle 300 as an analog value, so the value would be converted to `0000000100101100`, the binary representation of 300 by the Grove hat. This would then be processed by the IoT device.

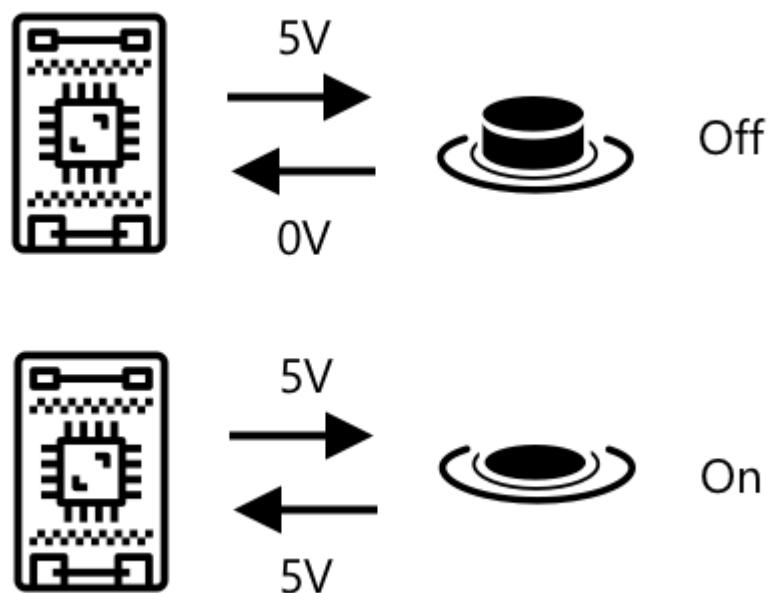
- ✓ If you don't know binary, then do a small amount of research to learn how numbers are represented by 0s and 1s. The [BBC Bitesize introduction to binary lesson](#) is a great place to start.

From a coding perspective, all this is usually handled by libraries that come with the sensors, so you don't need to worry about this conversion yourself. For the Grove light sensor you would use the Python library and call the `light` property, or use the Arduino library and call `analogRead` to get a value of 300.

Digital sensors

Digital sensors, like analog sensors, detect the world around them using changes in electrical voltage. The difference is they output a digital signal, either by only measuring two states or by using a built-in ADC. Digital sensors are becoming more and more common to avoid the need to use an ADC either in a connector board or on the IoT device itself.

The simplest digital sensor is a button or switch. This is a sensor with two states, on or off.



A button. Microcontroller by Template / Button by Dan Hetteix - all from the [Noun Project](#)

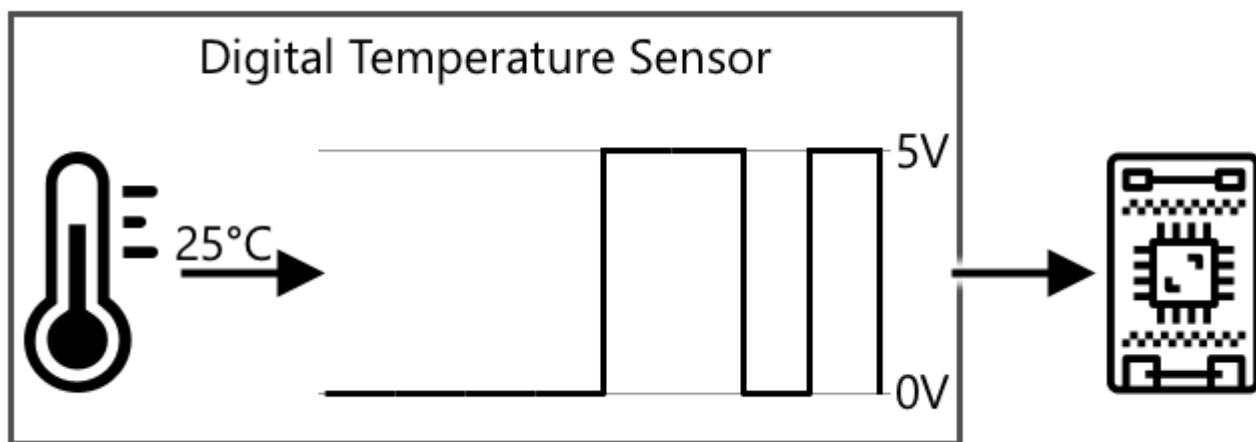
Pins on IoT devices such as GPIO pins can measure this signal directly as a 0 or 1. If the voltage sent is the same as the voltage returned, the value read is 1, otherwise the value read is 0. There is no

need to convert the signal, it can only be 1 or 0.

💡 Voltages are never exact especially as the components in a sensor will have some resistance, so there is usually a tolerance. For example, the GPIO pins on a Raspberry Pi work on 3.3V, and read a return signal above 1.8V as a 1, below 1.8V as 0.

- 3.3V goes into the button. The button is off so 0V comes out, giving a value of 0
- 3.3V goes into the button. The button is on so 3.3V comes out, giving a value of 1

More advanced digital sensors read analog values, then convert them using on-board ADCs to digital signals. For example, a digital temperature sensor will still use a thermocouple in the same way as an analog sensor, and will still measure the change in voltage caused by the resistance of the thermocouple at the current temperature. Instead of returning an analog value and relying on the device or connector board to convert to a digital signal, an ADC built into the sensor will convert the value and send it as a series of 0s and 1s to the IoT device. These 0s and 1s are sent in the same way as the digital signal for a button with 1 being full voltage and 0 being 0v.



A digital temperature sensor. Temperature by Vectors Market / Microcontroller by Template - all from the Noun Project

Sending digital data allows sensors to become more complex and send more detailed data, even encrypted data for secure sensors. One example is a camera. This is a sensor that captures an image and sends it as digital data containing that image, usually in a compressed format such as JPEG, to be read by the IoT device. It can even stream video by capturing images and sending either the complete image frame by frame or a compressed video stream.

What are actuators?

Actuators are the opposite of sensors - they convert an electrical signal from your IoT device into an interaction with the physical world such as emitting light or sound, or moving a motor.

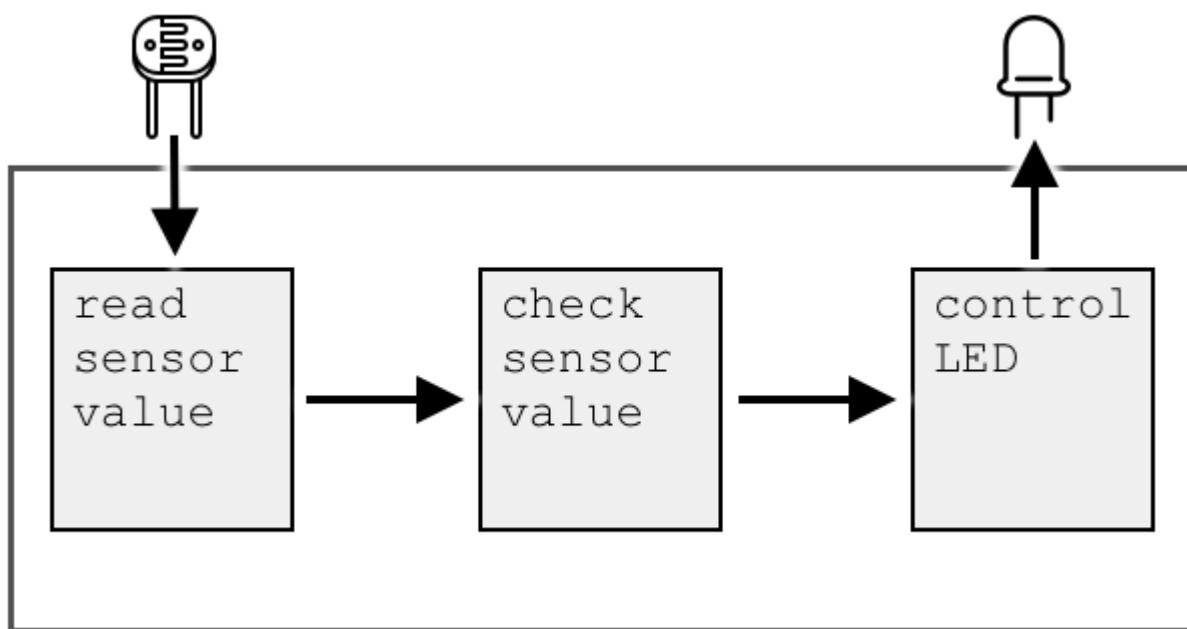
Some common actuators include:

- LED - these emit light when turned on
- Speaker - these emit sound based on the signal sent to them, from a basic buzzer to an audio speaker that can play music
- Stepper motor - these convert a signal into a defined amount of rotation, such as turning a dial 90°
- Relay - these are switches that can be turned on or off by an electrical signal. They allow a small voltage from an IoT device to turn on larger voltages.
- Screens - these are more complex actuators and show information on a multi-segment display. Screens vary from simple LED displays to high-resolution video monitors.

Do some research. What actuators does your phone have?

Use an actuator

Follow the relevant guide below to add an actuator to your IoT device, controlled by the sensor, to build an IoT nightlight. It will gather light levels from the light sensor, and use an actuator in the form of an LED to emit light when the detected light level is too low.



A flow chart of the assignment showing light levels being read and checked, and the LED begin controlled. Idr by Eucalyp / LED by abderraouf omara - all from the [Noun Project](#)

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)

- Single-board computer - Virtual device

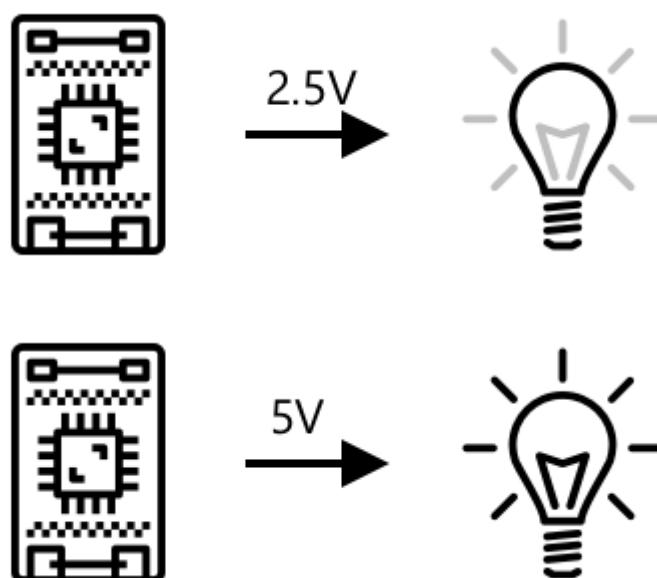
Actuator types

Like sensors, actuators are either analog or digital.

Analog actuators

Analog actuators take an analog signal and convert it into some kind of interaction, where the interaction changes based off the voltage supplied.

One example is a dimmable light, such as the ones you might have in your house. The amount of voltage supplied to the light determines how bright it is.



A light controlled by the voltage output of an IoT device. Idea by Pause08 / Microcontroller by Template - all from the Noun Project

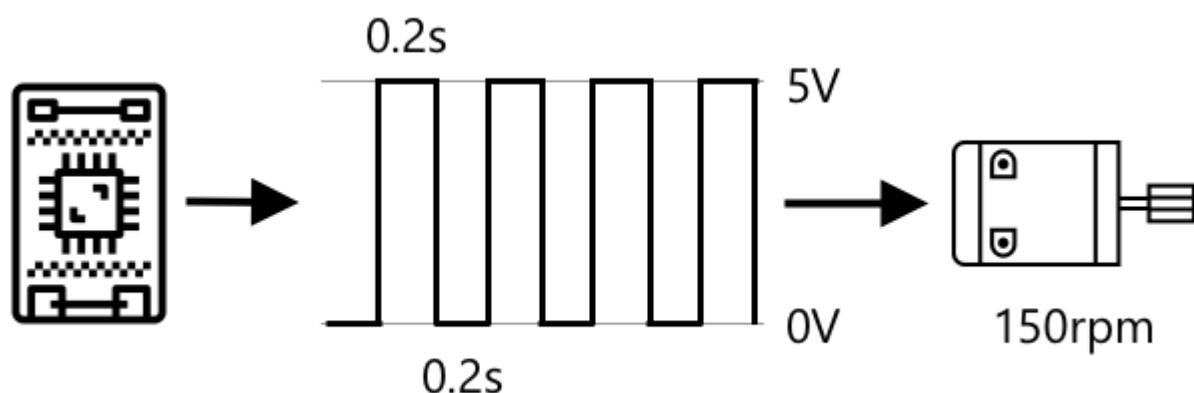
Like with sensors, the actual IoT device works on digital signals, not analog. This means to send an analog signal, the IoT device needs a digital to analog converter (DAC), either on the IoT device directly, or on a connector board. This will convert the 0s and 1s from the IoT device to an analog voltage that the actuator can use.

- What do you think happens if the IoT device sends a higher voltage than the actuator can handle?
- DO NOT test this out.

Another option for converting digital signals from an IoT device to an analog signal is pulse-width modulation. This involves sending lots of short digital pulses that act as if it was an analog signal.

For example, you can use PWM to control the speed of a motor.

Imagine you are controlling a motor with a 5V supply. You send a short pulse to your motor, switching the voltage to high (5V) for two hundredths of a second (0.02s). In that time your motor can rotate one tenth of a rotation, or 36°. The signal then pauses for two hundredths of a second (0.02s), sending a low signal (0V). Each cycle of on then off lasts 0.04s. The cycle then repeats.

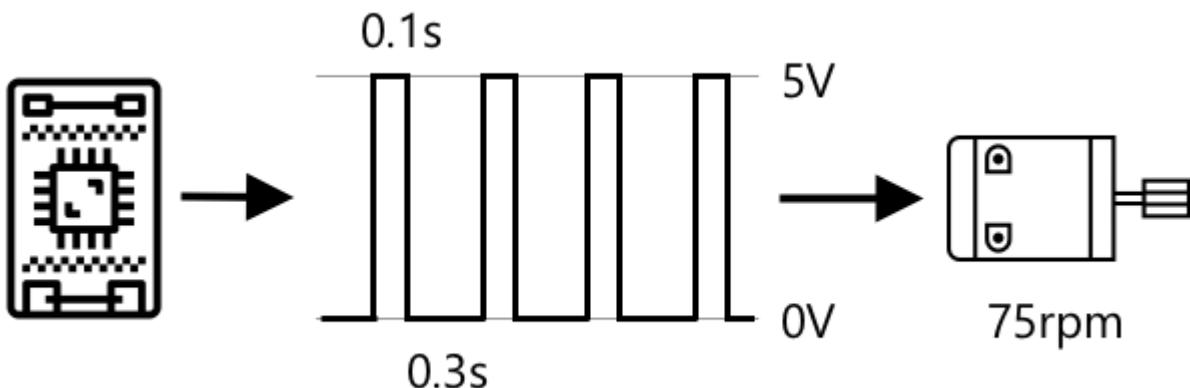


PWM rotation of a motor at 150RPM. motor by Bakunetsu Kaito / Microcontroller by Template - all from the [Noun Project](#)

This means in one second you have 25 5V pulses of 0.02s that rotate the motor, each followed by 0.02s pause of 0V not rotating the motor. Each pulse rotates the motor one tenth of a rotation, meaning the motor completes 2.5 rotations per second. You've used a digital signal to rotate the motor at 2.5 rotations per second, or 150 ([revolutions per minute](#), a non-standard measure of rotational velocity).

$$\begin{aligned} \text{output} \\ 25 \text{ pulses per second} \times 0.1 \text{ rotations per pulse} &= 2.5 \text{ rotations per second} \\ 2.5 \text{ rotations per second} \times 60 \text{ seconds in a minute} &= 150\text{rpm} \end{aligned}$$

🎓 When a PWM signal is on for half the time, and off for half it is referred to as a [50% duty cycle](#). Duty cycles are measured as the percentage time the signal is in the on state compared to the off state.



PWM rotation of a motor at 75RPM. motor by Bakunetsu Kaito / Microcontroller by Template - all from the [Noun Project](#)

You can change the motor speed by changing the size of the pulses. For example, with the same motor you can keep the same cycle time of 0.04s, with the on pulse halved to 0.01s, and the off pulse increasing to 0.03s. You have the same number of pulses per second (25), but each on pulse is half the length. A half length pulse only turns the motor one twentieth of a rotation, and at 25 pulses a second will complete 1.25 rotations per second or 75rpm. By changing the pulse speed of a digital signal you've halved the speed of an analog motor.

output

$$\begin{aligned} 25 \text{ pulses per second} \times 0.05 \text{ rotations per pulse} &= 1.25 \text{ rotations per second} \\ 1.25 \text{ rotations per second} \times 60 \text{ seconds in a minute} &= 75 \text{ rpm} \end{aligned}$$

- ✓ How would you keep the motor rotation smooth, especially at low speeds? Would you use a small number of long pulses with long pauses or lots of very short pulses with very short pauses?

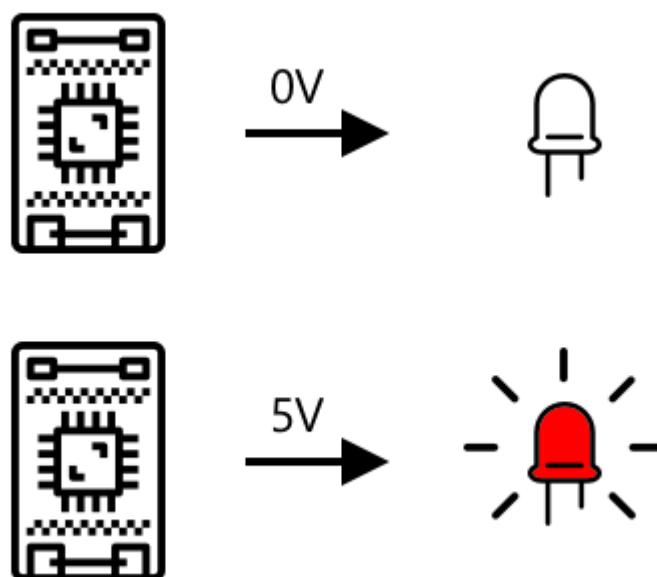
💡 Some sensors also use PWM to convert analog signals to digital signals.

🎓 You can read more on pulse-width modulation on the [pulse-width modulation page on Wikipedia](#).

Digital actuators

Digital actuators, like digital sensors, either have two states controlled by a high or low voltage or have a DAC built in so can convert a digital signal to an analog one.

One simple digital actuator is an LED. When a device sends a digital signal of 1, a high voltage is sent that lights the LED. When a digital signal of 0 is sent, the voltage drops to 0V and the LED turns off.



An LED turning on and off depending on voltage. LED by abderraouf omara / Microcontroller by Template - all from the [Noun Project](#)

- What other simple 2-state actuators can you think of? One example is a solenoid, which is an electromagnet that can be activated to do things like move a door bolt locking/unlocking a door.

More advanced digital actuators, such as screens require the digital data to be sent in certain formats. They usually come with libraries that make it easier to send the correct data to control them.

🚀 Challenge

The challenge in the last two lessons was to list as many IoT devices as you can that are in your home, school or workplace and decide if they are built around microcontrollers or single-board computers, or even a mixture of both.

For every device you listed, what sensors and actuators are they connected to? What is the purpose of each sensor and actuator connected to these devices?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read up on electricity and circuits on [ThingLearn](#).
- Read about the different types of temperature sensors on the [Seeed Studios Temperature Sensors guide](#)
- Read about LEDs on the [Wikipedia LED page](#)

Assignment

[Research sensors and actuators](#)

Connect your device to the Internet

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

The I in IoT stands for **Internet** - the cloud connectivity and services that enable a lot of the features of IoT devices, from gathering measurements from the sensors connected to the device, to sending messages to control the actuators. IoT devices typically connect to a single cloud IoT service using a standard communication protocol, and that service is connected to the rest of your IoT application, from AI services to make smart decisions around your data, to web apps for control or reporting.

 Data gathered from sensors and sent to the cloud is called telemetry.

IoT devices can receive messages from the cloud. Often the messages contain commands - that is instructions to perform an action either internally (such as reboot or update firmware), or using an actuator (such as turning on a light).

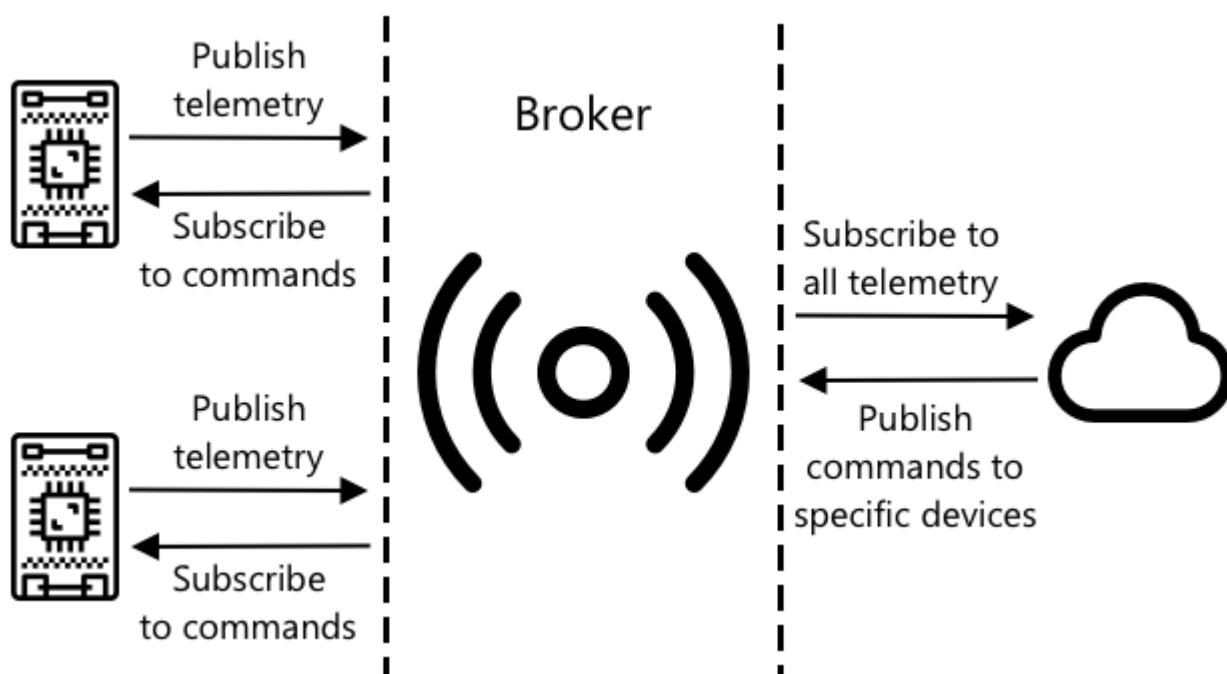
This lesson introduces some of the communication protocols IoT devices can use to connect to the cloud, and the types of data they might send or receive. You will also get hands-on with them both, adding internet control to your nightlight, moving the LED control logic to 'server' code running locally.

In this lesson we'll cover:

- [Communication protocols](#)
- [Message Queueing Telemetry Transport \(MQTT\)](#)
- [Telemetry](#)
- [Commands](#)

Communication protocols

There are a number of popular communication protocols used by IoT devices to communicate with the Internet. The most popular are based around publish/subscribe messaging via some kind of broker. The IoT devices connect to the broker and publish telemetry and subscribe to commands. The cloud services also connect to the broker and subscribe to all the telemetry messages and publish commands either to specific devices, or to groups of devices.



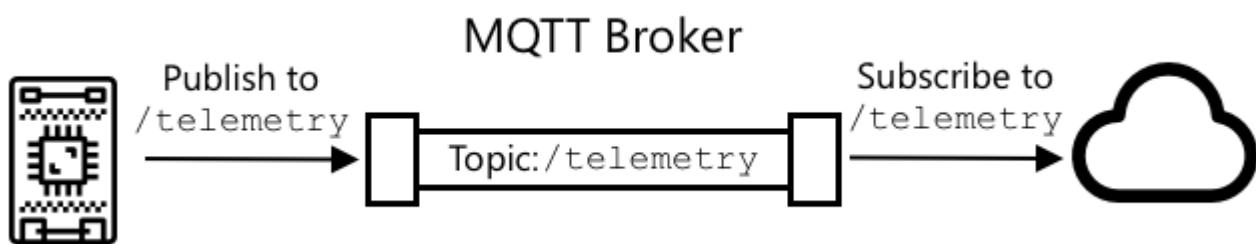
IoT devices connect to a broker and publish telemetry and subscribe to commands. Cloud services connect to the broker and subscribe to all telemetry and send commands to specific devices. Broadcast by RomStu / Microcontroller by Template / Cloud by Debi Alpa Nugraha - all from the [Noun Project](#)

MQTT is the most popular communication protocol for IoT devices and is covered in this lesson. Others protocols include AMQP and HTTP/HTTPS.

Message Queueing Telemetry Transport (MQTT)

MQTT is a lightweight, open standard messaging protocol that can send messages between devices. It was designed in 1999 to monitor oil pipelines, before being released as an open standard 15 years later by IBM.

MQTT has a single broker and multiple clients. All clients connect to the broker, and the broker routes messages to the relevant clients. Messages are routed using named topics, rather than being sent directly to an individual client. A client can publish to a topic, and any clients that subscribe to that topic will receive the message.



IoT device publishing telemetry on the /telemetry topic, and the cloud service subscribing to that topic. Microcontroller by Template / Cloud by Debi Alpa Nugraha - all from the Noun Project

- Do some research. If you have a lot of IoT devices, how can you ensure your MQTT broker can handle all the messages?

Connect your IoT device to MQTT

The first part of adding Internet control to your nightlight is connecting it to an MQTT broker.

Task

Connect your device to an MQTT broker.

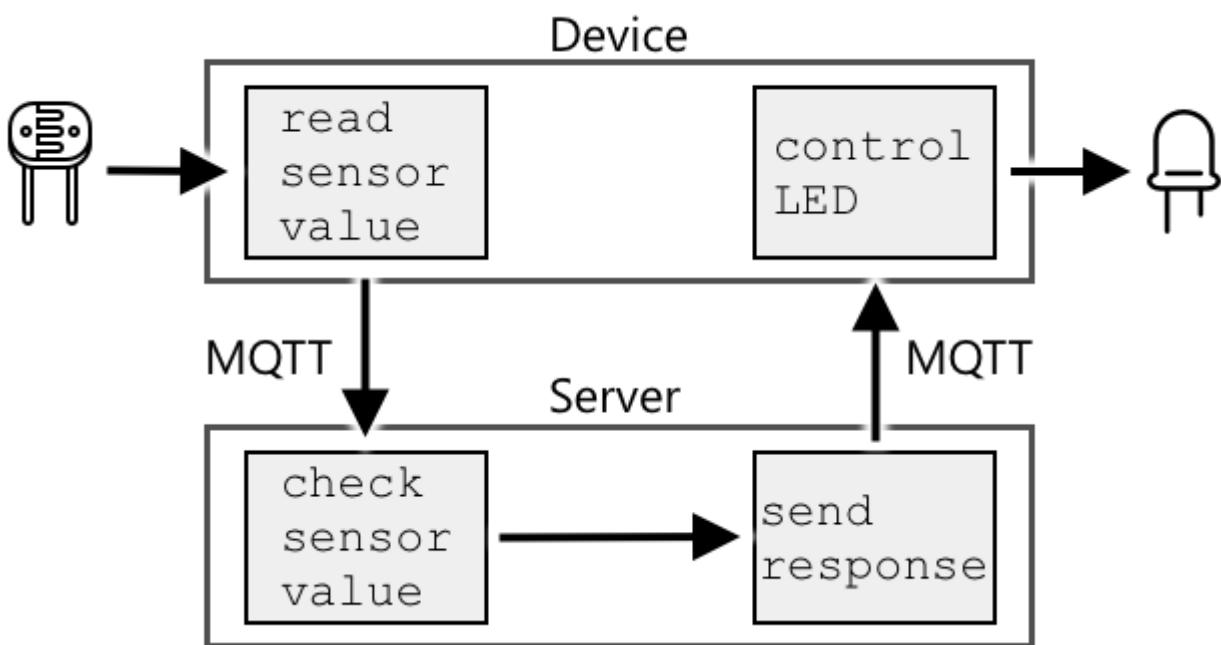
In this part of the lesson, you will connect your IoT nightlight to the internet to allow it to be remotely controlled. Later in this lesson, your IoT device will send a telemetry message over MQTT to a public MQTT broker with the light level, where it will be picked up by some server code that you will write. This code will check the light level and send a command message back to the device telling it to turn the LED on or off.

The real-world use case for such a setup could be to gather data from multiple light sensors before deciding to turn on lights, in a location that has a lot of lights, such as a stadium. This could stop the lights from being turned on if only one sensor was covered by clouds or a bird, but the other sensors detected enough light.

- ✓ What other situations would require data from multiple sensors to be evaluated before sending commands?

Rather than dealing with the complexities of setting up an MQTT broker as part of this assignment, you can use a public test server that runs [Eclipse Mosquitto](#), an open-source MQTT broker. This test broker is publicly available at test.mosquitto.org, and doesn't require an account to be set up, making it a great tool for testing MQTT clients and servers.

 This test broker is public and not secure. Anyone could be listening to what you publish, so it should not be used with any data that needs to be kept private



A flow chart of the assignment showing light levels being read and checked, and the LED being controlled. Idr by Eucalyp / LED by abderraouf omara - all from the [Noun Project](#)

Follow the relevant step below to connect your device to the MQTT broker:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi/Virtual IoT device](#)

A deeper dive into MQTT

Topics can have a hierarchy, and clients can subscribe to different levels of the hierarchy using wildcards. For example, you can send temperature telemetry messages to the `/telemetry/temperature` topic and humidity messages to the `/telemetry/humidity` topic, then in your cloud app subscribe to the `/telemetry/*` topic to receive both the temperature and humidity telemetry messages.

Messages can be sent with a quality of service (QoS), which determines the guarantee of the message being received.

- At most once - the message is sent only once and the client and broker take no additional steps to acknowledge delivery (fire and forget).
- At least once - the message is re-tried by the sender multiple times until acknowledgement is received (acknowledged delivery).
- Exactly once - the sender and receiver engage in a two-level handshake to ensure only one copy of the message is received (assured delivery).

 What situations might require an assured delivery message over a fire and forget message?

Although the name is Message Queueing (initials in MQTT), it doesn't actually support message queues. This means that if a client disconnects, then reconnects it won't receive messages sent during the disconnection, except for those messages that it had already started to process using the QoS process. Messages can have a retained flag set on them. If this is set, the MQTT broker will store the last message sent on a topic with this flag, and send this to any clients who later subscribe to the topic. This way, the clients will always get the latest message.

MQTT also supports a keep alive function that checks if the connection is still alive during long gaps between messages.

 [Mosquitto from the Eclipse Foundation](#) has a free MQTT broker you can run yourself to experiment with MQTT, along with a public MQTT broker you can use to test your code, hosted at test.mosquitto.org.

MQTT connections can be public and open, or encrypted and secured using usernames and passwords, or certificates.

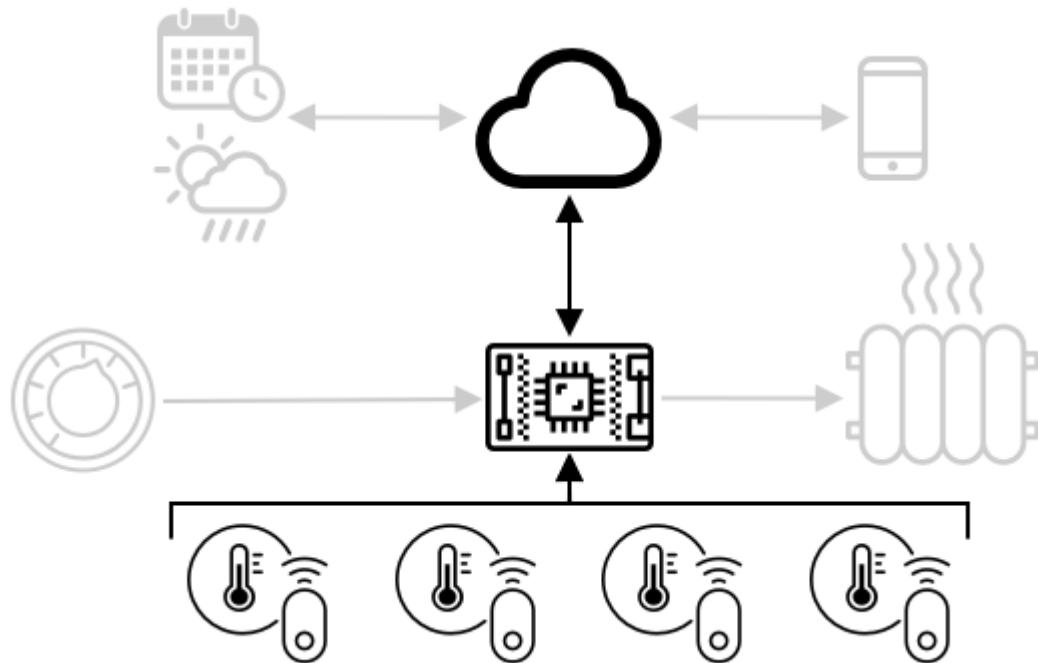
 MQTT communicates over TCP/IP, the same underlying network protocol as HTTP, but on a different port. You can also use MQTT over websockets to communicate with web apps running in a browser, or in situations where firewalls or other networking rules block standard MQTT connections.

Telemetry

The word telemetry is derived from Greek roots meaning to measure remotely. Telemetry is the act of gathering data from sensors and sending it to the cloud.

💡 One of the earliest telemetry devices was invented in France in 1874 and sent real-time weather and snow depths from Mont Blanc to Paris. It used physical wires as wireless technologies were not available at the time.

Let's look back at the example of the smart thermostat from Lesson 1.



An Internet connected thermostat using multiple room sensors. Temperature by Vectors Market / Microcontroller by Template / dial by Jamie Dickinson / heater by Pascal Heß / mobile phone and Calendar by Alice-vector / Cloud by Debi Alpa Nugraha / smart sensor by Andrei Yushchenko / weather by Adrien Coquet - all from the [Noun Project](#)

The thermostat has temperature sensors to gather telemetry. It would most likely have one temperature sensor built in, and it might connect to multiple external temperature sensors over a wireless protocol such as Bluetooth Low Energy (BLE).

An example of the telemetry data it would send could be:

Name	Value	Description
------	-------	-------------

Name	Value	Description
thermostat_temperature	18°C	The temperature measured by the thermostat's built-in temperature sensor
livingroom_temperature	19°C	The temperature measured by a remote temperature sensor that has been named <code>livingroom</code> to identify the room it is in
bedroom_temperature	21°C	The temperature measured by a remote temperature sensor that has been named <code>bedroom</code> to identify the room it is in

The cloud service can then use this telemetry data to make decisions around what commands to send to control the heating.

Send telemetry from your IoT device

The next part in adding Internet control to your nightlight is sending the light level telemetry to the MQTT broker on a telemetry topic.

Task

Send light level telemetry to the MQTT broker.

Data is sent encoded as JSON – short for JavaScript Object Notation, a standard for encoding data in text using key/value pairs.

 If you've not come across JSON before, you can learn more about it on the [JSON.org documentation](https://www.json.org).

Follow the relevant step below to send telemetry from your device to the MQTT broker:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi/Virtual IoT device](#)

Receive telemetry from the MQTT broker

There's no point in sending telemetry if there's nothing on the other end to listen for it. The light level telemetry needs something listening to it to process the data. This 'server' code is the kind of code you will deploy to a cloud service as part of a larger IoT application, but here you are going to run this code locally on your computer (or on your Pi if you are coding directly on there). The server code

consists of a Python app that listens to telemetry messages over MQTT with light levels. Later in this lesson you will make it reply with a command message with instructions to turn the LED on or off.

- Do some research: What happens to MQTT messages if there is no listener?

Install Python and VS Code

If you don't have Python and VS Code installed locally, you will need to install them both to code the server. If you are using a virtual device, or are working on your Raspberry Pi you can skip this step.

Task

Install Python and VS Code.

1. Install Python. Refer to the [Python downloads page](#) for instructions on install the latest version of Python.
2. Install Visual Studio Code (VS Code). This is the editor you will be using to write your virtual device code in Python. Refer to the [VS Code documentation](#) for instructions on installing VS Code.



You are free to use any Python IDE or editor for these lessons if you have a preferred tool, but the lessons will give instructions based off using VS Code.

3. Install the VS Code Pylance extension. This is an extension for VS Code that provides Python language support. Refer to the [Pylance extension documentation](#) for instructions on installing this extension in VS Code.

Configure a Python virtual environment

One of the powerful features of Python is the ability to install [pip packages](#) - these are packages of code written by other people and published to the Internet. You can install a pip package onto your computer with one command, then use that package in your code. You'll be using pip to install a package to communicate over MQTT.

By default when you install a package it is available everywhere on your computer, and this can lead to problems with package versions - such as one application depending on one version of a package that breaks when you install a new version for a different application. To work around this problem, you can use a [Python virtual environment](#), essentially a copy of Python in a dedicated folder, and when you install pip packages they get installed just to that folder.

Task

Configure a Python virtual environment and install the MQTT pip packages.

- From your terminal or command line, run the following at a location of your choice to create and navigate to a new directory:

```
sh
mkdir nightlight-server
cd nightlight-server
```

- Now run the following to create a virtual environment in the `.venv` folder

```
sh
python3 -m venv .venv
```

 You need to explicitly call `python3` to create the virtual environment just in case you have Python 2 installed in addition to Python 3 (the latest version). If you have Python2 installed then calling `python` will use Python 2 instead of Python 3

- Activate the virtual environment:

- On Windows run:

```
cmd
.venv\Scripts\activate.bat
```

- On macOS or Linux, run:

```
cmd
source ./venv/bin/activate
```

- Once the virtual environment has been activated, the default `python` command will run the version of Python that was used to create the virtual environment. Run the following to get the version:

```
sh
python --version
```

The output will be similar to the following:

output

```
(.venv) → nightlight-server python --version  
Python 3.9.1
```

 Your Python version may be different - as long as it's version 3.6 or higher you are good. If not, delete this folder, install a newer version of Python and try again.

5. Run the following commands to install the pip package for [Paho-MQTT](#), a popular MQTT library.

```
sh  
pip install paho-mqtt
```

This pip package will only be installed in the virtual environment, and will not be available outside of this.

Write the server code

The server code can now be written in Python.

Task

Write the server code.

1. From your terminal or command line, run the following inside the virtual environment to create a Python file called `app.py` :

- From Windows run:

```
cmd  
type nul > app.py
```

- On macOS or Linux, run:

```
cmd  
touch app.py
```

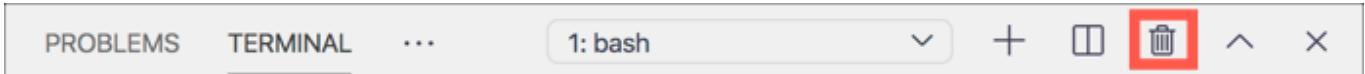
2. Open the current folder in VS Code:

```
sh  
code .
```

3. When VS Code launches, it will activate the Python virtual environment. This will be reported in the bottom status bar:



4. If the VS Code Terminal is already running when VS Code starts up, it won't have the virtual environment activated in it. The easiest thing to do is kill the terminal using the **Kill the active terminal instance** button:



5. Launch a new VS Code Terminal by selecting *Terminal -> New Terminal, or pressing **CTRL+`**. The new terminal will load the virtual environment, with the call to activate this appearing in the terminal. The name of the virtual environment (`.venv`) will also be in the prompt:

```
output
→ nightlight source .venv/bin/activate
(.venv) → nightlight
```

6. Open the `app.py` file from the VS Code explorer and add the following code:

```
python
import json
import time

import paho.mqtt.client as mqtt

id = '<ID>'

client_telemetry_topic = id + '/telemetry'
client_name = id + 'nightlight_server'

mqtt_client = mqtt.Client(client_name)
mqtt_client.connect('test.mosquitto.org')

mqtt_client.loop_start()

def handle_telemetry(client, userdata, message):
    payload = json.loads(message.payload.decode())
    print("Message received:", payload)

mqtt_client.subscribe(client_telemetry_topic)
mqtt_client.on_message = handle_telemetry
```

```
while True:  
    time.sleep(2)
```

Replace `<ID>` on line 6 with the unique ID you used when creating your device code.

⚠ This **must** be the same ID that you used on your device, or the server code won't subscribe or publish to the right topic.

This code creates an MQTT client with a unique name, and connects to the `test.mosquitto.org` broker. It then starts a processing loop that runs in on a background thread listening for messages on any subscribed topics.

The client then subscribes to messages on the `telemetry` topic, and defines a function that is called when a message is received. When a `telemetry` message is received, the `handle_telemetry` function is called, printing the message received to the console.

Finally an infinite loop keeps the application running. The MQTT client is listening to messages on a background thread and runs all the time the main application is running.

7. From the VS Code terminal, run the following to run your Python app:

```
sh  
python app.py
```

The app will start listening to messages from the IoT device.

8. Make sure your device is running and sending telemetry messages. Adjust the light levels detected by your physical or virtual device. Messages being received will be printed to the terminal.

```
output  
.venv) → nightlight-server python app.py  
Message received: {'light': 0}  
Message received: {'light': 400}
```

The `app.py` file in the `nightlight` virtual environment has to be running for the `app.py` file in the `nightlight-server` virtual environment to receive the messages being sent.

 You can find this code in the [code-server/server](#) folder.

How often should telemetry be sent?

One important consideration with telemetry is how often to measure and send the data? The answer is - it depends. If you measure often you can respond faster to changes in measurements, but you use more power, more bandwidth, generate more data and need more cloud resources to process. You need to measure often enough, but not too often.

For a thermostat, measuring every few minutes is probably more than enough as temperatures don't change that often. If you only measure once a day then you could end up heating your house for nighttime temperatures in the middle of a sunny day, whereas if you measure every second you will have thousands of unnecessarily duplicated temperature measurements that will eat into the users' Internet speed and bandwidth (a problem for people with limited bandwidth plans), use more power which can be a problem for battery powered devices like remote sensors, and increase the cost of the providers cloud computing resources processing and storing them.

If you are monitoring data around a piece of machinery in a factory that if it fails could cause catastrophic damage and millions of dollars in lost revenue, then measuring multiple times a second might be necessary. It's better to waste bandwidth than miss telemetry that indicates that a machine needs to be stopped and fixed before it breaks.

 In this situation, you might consider having an edge device to process the telemetry first to reduce reliance on the Internet.

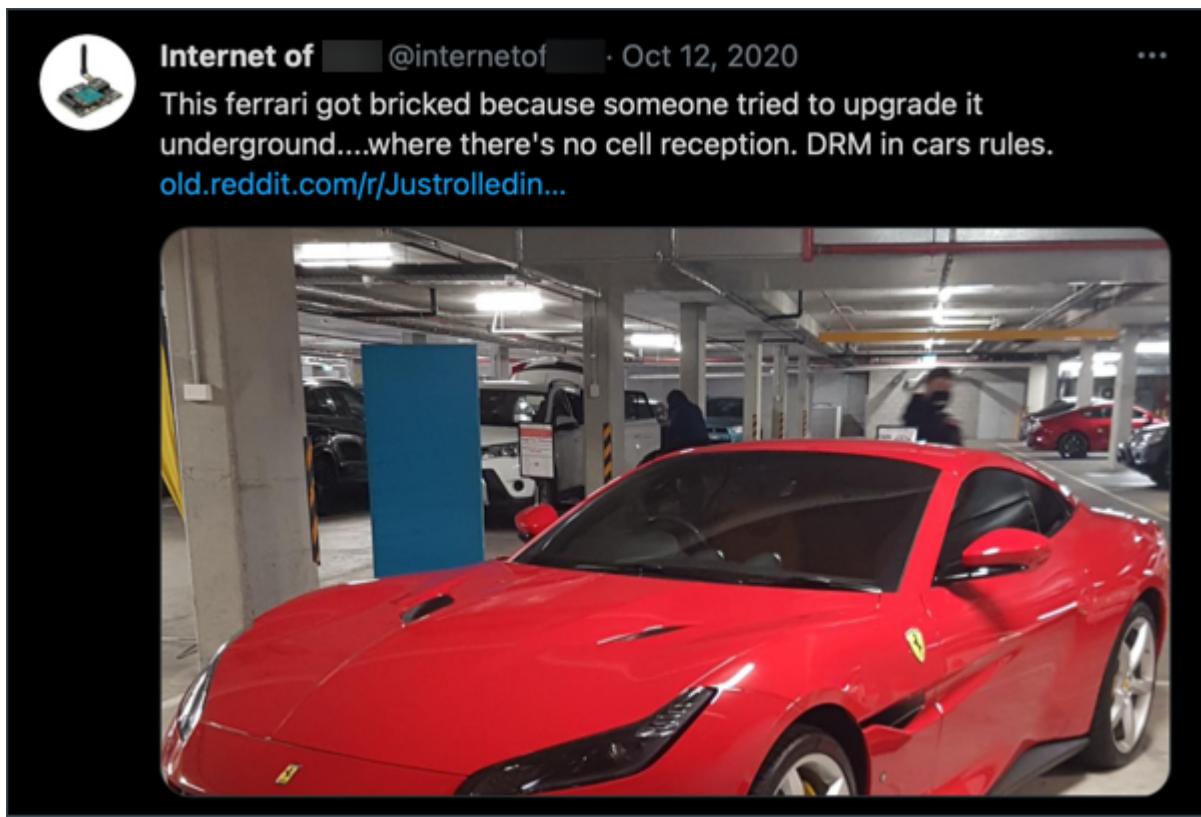
Loss of connectivity

Internet connections can be unreliable, with outages common. What should an IoT device do under these circumstances - should it lose the data, or should it store it until connectivity is restored? Again, the answer is it depends.

For a thermostat the data can probably be lost as soon as a new temperature measurement has been taken. The heating system doesn't care that 20 minutes ago it was 20.5°C if the temperature is now 19°C, it's the temperature now that determines if the heating should be on or off.

For machinery you might want to keep the data, especially if it is used to look for trends. There are machine learning models that can detect anomalies in streams of data by looking over data from defined period of time (such as the last hour) and spotting anomalous data. This is often used for predictive maintenance, looking for indications that something might break soon so you can repair or replace it before that happens. You might want every bit of telemetry for a machine sent so it can be processed for anomaly detection, so once the IoT device can reconnect it will send all the telemetry generated during the Internet outage.

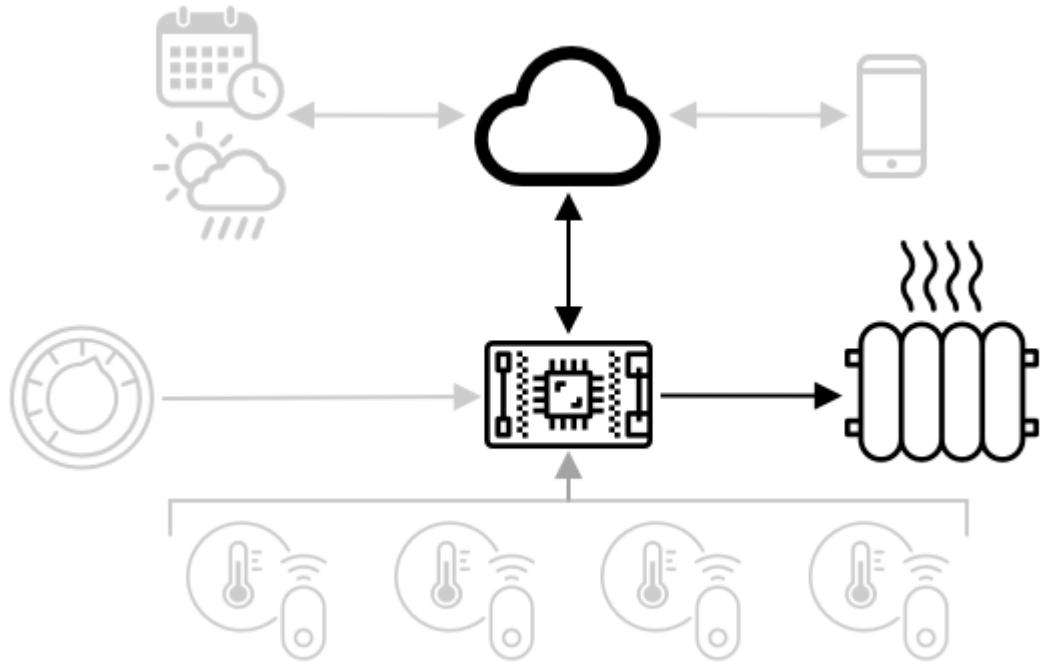
IoT device designers should also consider if the IoT device can be used during an Internet outage or loss of signal caused by location. A smart thermostat should be able to make some limited decisions to control heating if it can't send telemetry to the cloud due to an outage.



For MQTT to handle a loss of connectivity, the device and server code will need to be responsible for ensuring message delivery if it is needed, for example by requiring that all messages sent are replied to by additional messages on a reply topic, and if not they are queued manually to be replayed later.

Commands

Commands are messages sent by the cloud to a device, instructing it to do something. Most of the time this involves giving some kind of output via an actuator, but it can be an instruction for the device itself, such as to reboot, or gather extra telemetry and return it as a response to the command.



An Internet connected thermostat receiving a command to turn on the heating. Temperature by Vectors Market / Microcontroller by Template / dial by Jamie Dickinson / heater by Pascal Heß / mobile phone and Calendar by Alice-vector / Cloud by Debi Alpa Nugraha / smart sensor by Andrei Yushchenko / weather by Adrien Coquet - all from the [Noun Project](#)

A thermostat could receive a command from the cloud to turn the heating on. Based on the telemetry data from all the sensors, if the cloud service has decided that the heating should be on, so it sends the relevant command.

Send commands to the MQTT broker

The next step for our Internet controlled nightlight is for the server code to send a command back to the IoT device to control the light based on the light levels it senses.

1. Open the server code in VS Code
2. Add the following line after the declaration of the `client_telemetry_topic` to define which topic to send commands to:

```
python
```

```
server_command_topic = id + '/commands'
```

3. Add the following code to the end of the `handle_telemetry` function:

```
python
```

```
command = { 'led_on' : payload['light'] < 300 }
print("Sending message:", command)

client.publish(server_command_topic, json.dumps(command))
```

This sends a JSON message to the command topic with the value of `led_on` set to true or false depending on if the light is less than 300 or not. If the light is less than 300, true is sent to instruct the device to turn the LED on.

4. Run the code as before

5. Adjust the light levels detected by your physical or virtual device. Messages being received and commands being sent will be written to the terminal:

output

```
(.venv) → nightlight-server python app.py
Message received: {'light': 0}
Sending message: {'led_on': True}
Message received: {'light': 400}
Sending message: {'led_on': False}
```

 The telemetry and commands are being sent on a single topic each. This means telemetry from multiple devices will appear on the same telemetry topic, and commands to multiple devices will appear on the same commands topic. If you wanted to send a command to a specific device, you could use multiple topics, named with a unique device id, such as `/commands/device1` , `/commands/device2` . That way a device can listen on messages just meant for that one device.

 You can find this code in the [code-commands/server](#) folder.

Handle commands on the IoT device

Now that commands are being sent from the server, you can now add code to the IoT device to handle them and control the LED.

Follow the relevant step below to listen to commands from the MQTT broker:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi/Virtual IoT device](#)

Once this code is written and running, experiment with changing light levels. Watch the output from the server and device, and watch the LED as you change light levels.

Loss of connectivity

What should a cloud service do if it needs to send a command to an IoT device that is offline? Again, the answer is it depends.

If the latest command overrides an earlier one then the earlier ones can probably be ignored. If a cloud service sends a command to turn the heating on, then sends a command to turn it off, then the on command can be ignored and not resent.

If the commands need to be processed in sequence, such as move a robot arm up, then close a grabber then they need to be sent in order once connectivity is restored.

- How could the device or server code ensure commands are always sent and handled in order over MQTT if needed?
-

Challenge

The challenge in the last three lessons was to list as many IoT devices as you can that are in your home, school or workplace and decide if they are built around microcontrollers or single-board computers, or even a mixture of both, and think about what sensors and actuators they are using.

For these devices, think about what messages they might be sending or receiving. What telemetry do they send? What messages or commands might they receive? Do you think they are secure?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

Read more on MQTT on the [MQTT Wikipedia page](#).

Try running an MQTT broker yourself using [Mosquitto](#) and connect to it from your IoT device and server code.

 Tip - by default Mosquitto doesn't allow anonymous connections (that is connecting without a username and password), and doesn't allow connections from outside of the computer it's running on. You can fix this with a [mosquitto.conf config file](#) with the following:

```
sh
listener 1883 0.0.0.0
allow_anonymous true
```

Assignment

[Compare and contrast MQTT with other communication protocols](#)

Predict plant growth with IoT

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

Plants need certain things to grow - water, carbon dioxide, nutrients, light, and heat. In this lesson, you'll learn how to calculate the growth and maturity rates of plants by measuring the air temperature.

In this lesson we'll cover:

- [Digital agriculture](#)
- [Why is temperature important when farming?](#)

- Measure ambient temperature
- Growing degree days (GDD)
- Calculate GDD using temperature sensor data

Digital agriculture

Digital Agriculture is transforming how we farm, using tools to collect, store and analyze data from farming. We are currently in a period described as the 'Fourth Industrial Revolution' by the World Economic Forum, and the rise of digital agriculture has been labelled as the 'Fourth Agricultural Revolution', or 'Agriculture 4.0'.

 The term Digital Agriculture also includes the whole 'agriculture value chain', that is the entire journey from farm to table. It includes tracking produce quality as food is shipped and processed, warehouse and e-commerce systems, even tractor rental apps!

These changes allow farmers to increase yields, use less fertilizers and pesticides, and water more efficiently. Although primarily used in richer nations, sensors and other devices are slowly reducing in price, making them more accessible in developing nations.

Some techniques enabled by digital agriculture are:

- Temperature measurement - measuring temperature allows farmers to predict plant growth and maturity.
- Automated watering - measuring soil moisture and turning on irrigation systems when the soil is too dry, rather than timed watering. Timed watering can lead to crops being under-watered during a hot, dry spell, or over-watered during rain. By watering only when the soil needs it farmers can optimize their water use.
- Pest control - farmers can use cameras on automated robots or drones to check for pests, then apply pesticides only where needed, reducing the amount of pesticides used and reducing pesticide run-off into local water supplies.

 Do some research. What other techniques are used to improve farming yields?

 The term 'Precision Agriculture' is used to define observing, measuring and responding to crops on a per-field basis, or even on parts of a field. This includes measuring water, nutrient and pest levels and responding accurately, such as watering only a small part of a field.

Why is temperature important when farming?

When learning about plants, most students are taught about the necessity of water, light, carbon dioxide (CO_2) and nutrients. Plants also need warmth to grow - this is why plants bloom in spring as the temperature rises, why snowdrops or daffodils can sprout early due to a short warm spell, and why hothouses and greenhouses are so good at making plants grow.

 Hothouses and greenhouses do a similar job, but with an important difference. Hothouses are heated artificially and allow farmers to control temperatures more accurately, greenhouses rely on the sun for warmth and usually the only control is windows or other openings to let heat out.

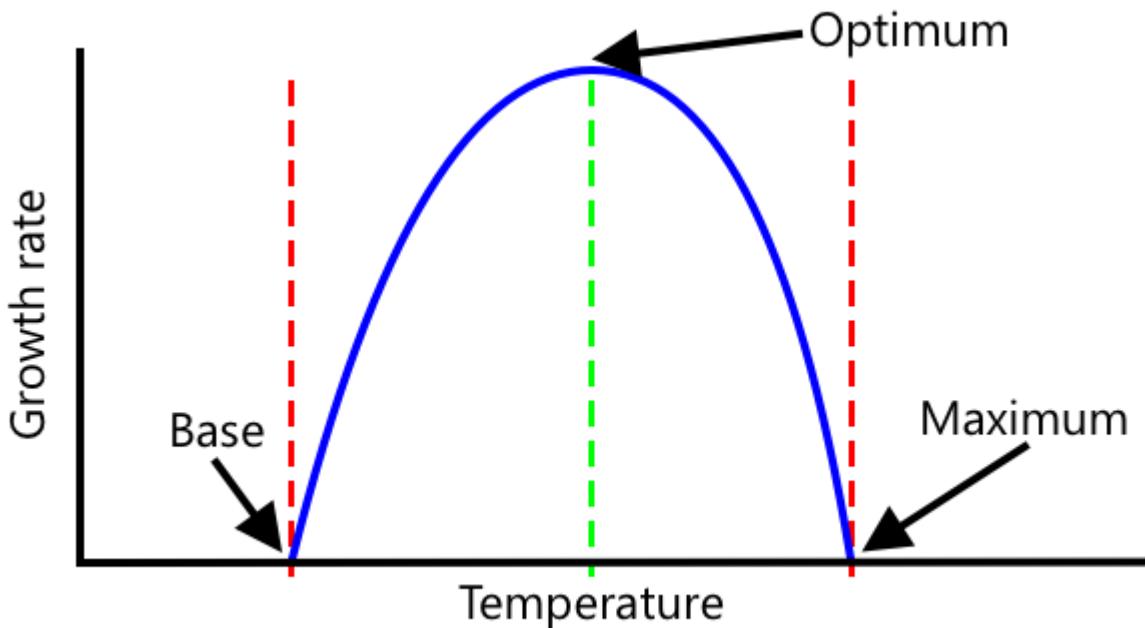
Plants have a base or minimum temperature, optimal temperature, and maximum temperature, all based on daily average temperatures.

- Base temperature - this is the minimum daily average temperature needed for a plant to grow.
- Optimum temperature - this is the best daily average temperature to get the most growth.
- Maximum temperature - This is the maximum temperature a plant can withstand. Above this the plant will shut down its growth in an attempt to conserve water and stay alive.

 These are average temperatures, averaged over the daily and nightly temperatures. Plants also need different temperatures day and night to help them photosynthesize more efficiently and save energy at night.

Each species of plant has different values for their base, optimal and maximum. This is why some plants thrive in hot countries, and others in colder countries.

-  Do some research. For any plants you have in your garden, school, or local park see if you can find the base temperature.



The graph above shows an example growth rate to temperature graph. Up to the base temperature there is no growth. The growth rate increases up to the optimum temperature, then falls after reaching this peak. At the maximum temperature growth stops.

The shape of this graph varies from plant species to plant species. Some have sharper drop offs above the optimum, some have slower increases from the base to the optimum.

 For a farmer to get the best growth, they will need to know the three temperature values and understand the shape of the graphs for the plants they are growing.

If a farmer has control of temperature, for example in a commercial hothouse, then they can optimise for their plants. A commercial hothouse growing tomatoes for example will have the temperature set to around 25°C during the day and 20°C at night to get the fastest growth.

 Combining these temperatures with artificial lights, fertilizers and controlled CO₂ levels means commercial growers can grow and harvest all year round.

Measure ambient temperature

Temperature sensors can be used with IoT devices to measure ambient temperature.

Task - measure temperature

Work through the relevant guide to monitor temperatures using your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Growing degree days

Growing degree days (also known as growing degree units) are a way of measuring the growth of plants based off the temperature. Assuming a plant has enough water, nutrients and CO₂, the temperature determines the growth rate.

Growing degree days, or GDD are calculated per day as the average temperature in Celsius for a day above the plants base temperature. Each plant needs a certain number of GDD to grow, flower or produce and mature a crop. The more GDD each day, the faster the plant will grow.

🇺🇸 For Americans, growing degree days can also be calculated using Fahrenheit. 5 GDD^C (growing degree days in Celsius) is the equivalent of 9 GDD^F (growing degree days in Fahrenheit).

The full formula for GDD is a little complicated, but there is a simplified equation that is often used as a good approximation:

$$GDD = \frac{T_{\max} + T_{\min}}{2} - T_{\text{base}}$$

- **GDD** - this is the number of growing degree days
- **T_{max}** - this is the daily maximum temperature in degrees Celsius
- **T_{min}** - this is the daily minimum temperature in degrees Celsius
- **T_{base}** - this is the plants base temperature in degrees Celsius

💡 There are variations that deal with T_{max} above 30°C or T_{min} below T_{base}, but we'll ignore these for now.

Example - Corn/Maize

Depending on the variety, corn (or maize) needs between 800 and 2,700 GDD to mature, with a base temperature of 10°C.

On the first day above the base temperature, the following temperatures were measured:

Measurement	Temp °C
Maximum	16
Minimum	12

Plugging these numbers in to our calculation:

- $T_{\max} = 16$
- $T_{\min} = 12$
- $T_{\text{base}} = 10$

This gives a calculation of:

$$\text{GDD} = \frac{16+12}{2} - 10 = 4$$

The corn received 4 GDD on that day. Assuming a corn variety that needs 800 GDD days to mature, it will need another 796 GDD to reach maturity.

- Do some research. For any plants you have in your garden, school, or local park see if you can find the number of GDD required to reach maturity or produce crops.

Calculate GDD using temperature sensor data

Plants don't grow on fixed dates - for example you can't plant a seed and know that the plant will bear fruit exactly 100 days later. Instead as a farmer you can have a rough idea how long a plant takes to grow, then you would check daily to see when the crops were ready.

This has a huge labour impact on a large farm, and risks the farmer missing crops that are ready unexpectedly early. By measuring temperatures, the farmer can calculate the GDD a plant has received, allowing them to only check close to the expected maturity.

By gathering temperature data using an IoT device, a farmer can automatically be notified when plants are close to maturity. A typical architecture for this is to have the IoT devices measure temperature, then publish this telemetry data over the Internet using something like MQTT. Server code then listens to this data and saves it somewhere, such as to a database. This means the data can then be analyzed later, such as a nightly job to calculate the GDD for the day, total up the GDD for each crop so far and alert if a plant is close to maturity.



Telemetry data is sent to a server and then saved to a database. database by Icons Bazaar - from the [Noun Project](#)

The server code can also augment the data by adding extra information. For example, the IoT device can publish an identifier to indicate which device it is, and the sever code can use this to look up the location of the device, and what crops it is monitoring. It can also add basic data like the current time as some IoT devices don't have the necessary hardware to keep track of an accurate time, or require additional code to read the current time over the Internet.

- Why do you think different fields might have different temperatures?

Task - publish temperature information

Work through the relevant guide to publish temperature data over MQTT using your IoT device so it can be analyzed later:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi/Virtual IoT device](#)

Task - capture and store the temperature information

Once the IoT device is publishing telemetry, the server code can be written to subscribe to this data and store it. Rather than save it to a database, the server code will save it to a Comma Separated Values (CSV) file. CSV files store data as rows of values as text, with each value separated by a comma, and each record on a new line. They are a convenient, human-readable and well supported way to save data as a file.

The CSV file will have two columns - *date* and *temperature*. The *date* column is set as the current date and time that the message was received by the server, the *temperature* comes from the telemetry message.

1. Repeat the steps in lesson 4 to create server code to subscribe to telemetry. You don't need to add code to publish commands.

The steps for this are:

- Configure and activate a Python Virtual Environment
- Install the paho-mqtt pip package
- Write the code to listen for MQTT messages published on the telemetry topic

 ⚠ You can refer to [the instructions in lesson 4 for creating a Python app to receive telemetry if needed.](#)

Name the folder for this project `temperature-sensor-server`.

2. Make sure the `client_name` reflects this project:

```
client_name = id + 'temperature_sensor_server'
```

3. Add the following imports to the top of the file, below the existing imports:

```
from os import path
import csv
from datetime import datetime
```

This imports a library for reading files, a library to interact with CSV files, and a library to help with dates and times.

4. Add the following code before the `handle_telemetry` function:

```
temperature_file_name = 'temperature.csv'
fieldnames = ['date', 'temperature']

if not path.exists(temperature_file_name):
    with open(temperature_file_name, mode='w') as csv_file:
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
```

This code declares some constants for the name of the file to write to, and the name of the column headers for the CSV file. The first row of a CSV file traditionally contains column headers separated by commas.

The code then checks to see if the CSV file already exists. If it doesn't exist, it is created with the column headers on the first row.

5. Add the following code to the end of the `handle_telemetry` function:

```
python  
with open(temperature_file_name, mode='a') as temperature_file:  
    temperature_writer = csv.DictWriter(temperature_file, fieldnames=fieldnames)  
    temperature_writer.writerow({'date' : datetime.now().astimezone().replace(tzinfo=None).isoformat(), 'temperature' : temperature})
```

This code opens the CSV file, then appends a new row on the end. The row has the current date and time formatted into a human-readable format, followed by the temperature received from the IoT device. The data is stored in ISO 8601 format with the timezone, but without microseconds.

6. Run this code in the same way as before, making sure your IoT device is sending data. A CSV file called `temperature.csv` will be created in the same folder. If you view it you will see date/times and temperature measurements:

```
output  
date,temperature  
2021-04-19T17:21:36-07:00,25  
2021-04-19T17:31:36-07:00,24  
2021-04-19T17:41:36-07:00,25
```

7. Run this code for a while to capture data. Ideally you should run this for an entire day to gather enough data for GDD calculations.

 If you are using Virtual IoT Device, select the random checkbox and set a range to avoid getting the same temperature everytime the temperature value is returned.

Temperature Pin 6

Units: Celsius

Value range: -273.15 to 999,999,999.00

Value: 0

Random:

Min

-10



Max

100



Set

Delete

💻 If you want to run this for an entire day, then you need to make sure the computer your server code is running on won't go to sleep, either by changing your power settings, or running something like [this keep system active Python script](#).

 You can find this code in the [code-server/temperature-sensor-server](#) folder.

Task - calculate GDD using the stored data

Once the server has captured temperature data, the GDD for a plant can be calculated.

The steps to do this manually are:

1. Find the base temperature for the plant. For example, for strawberries the base temperature is 10°C.
2. From the `temperature.csv`, find the highest and lowest temperatures for the day
3. Use the GDD calculation given earlier to calculate GDD

For example, if the highest temperature for the day is 25°C, and the lowest is 12°C:

$$\text{GDD} = \frac{25 + 12}{2} - 10 = 8.5$$

- $25 + 12 = 37$
- $37 / 2 = 18.5$
- $18.5 - 10 = 8.5$

Therefore the strawberries have received **8.5** GDD. Strawberries need about 250 GDD to bear fruit, so still a while to go.

Challenge

Plants need more than heat to grow. What other things are needed?

For these, find if there are sensors that can measure them. What about actuators to control these levels? How would you put together one or more IoT devices to optimize plant growth?

Post-lecture quiz

Review & Self Study

Read more on digital agriculture on the [Digital Agriculture Wikipedia page](#). Also read more about precision agriculture the [Precision Agriculture Wikipedia page](#).

The full growing degree days calculation is more complicated than the simplified one given here. Read more about the more complicated equation and how to deal with temperatures below the baseline on the [Growing Degree Day Wikipedia page](#).

Assignment

[Visualize GDD data using a Jupyter Notebook](#)

Detect soil moisture

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last lesson we looked at measuring an ambient property and using it to predict plant growth. Temperature can be controlled, but it is expensive to do so, requiring controlled environments. The easiest ambient property to control for plants is water - something that is controlled everyday from large-scale irrigation systems to young kids with watering cans watering their gardens.



In this lesson you will learn about measuring soil moisture, and in the next lesson you will learn how to control an automated watering system. This lesson introduces a third sensor, you've already used a light sensor, a temperature sensor, so in this lesson you will also learn more about how sensors and actuators communicate with IoT devices to understand more about how a soil moisture sensor can send data to an IoT device.

In this lesson we'll cover:

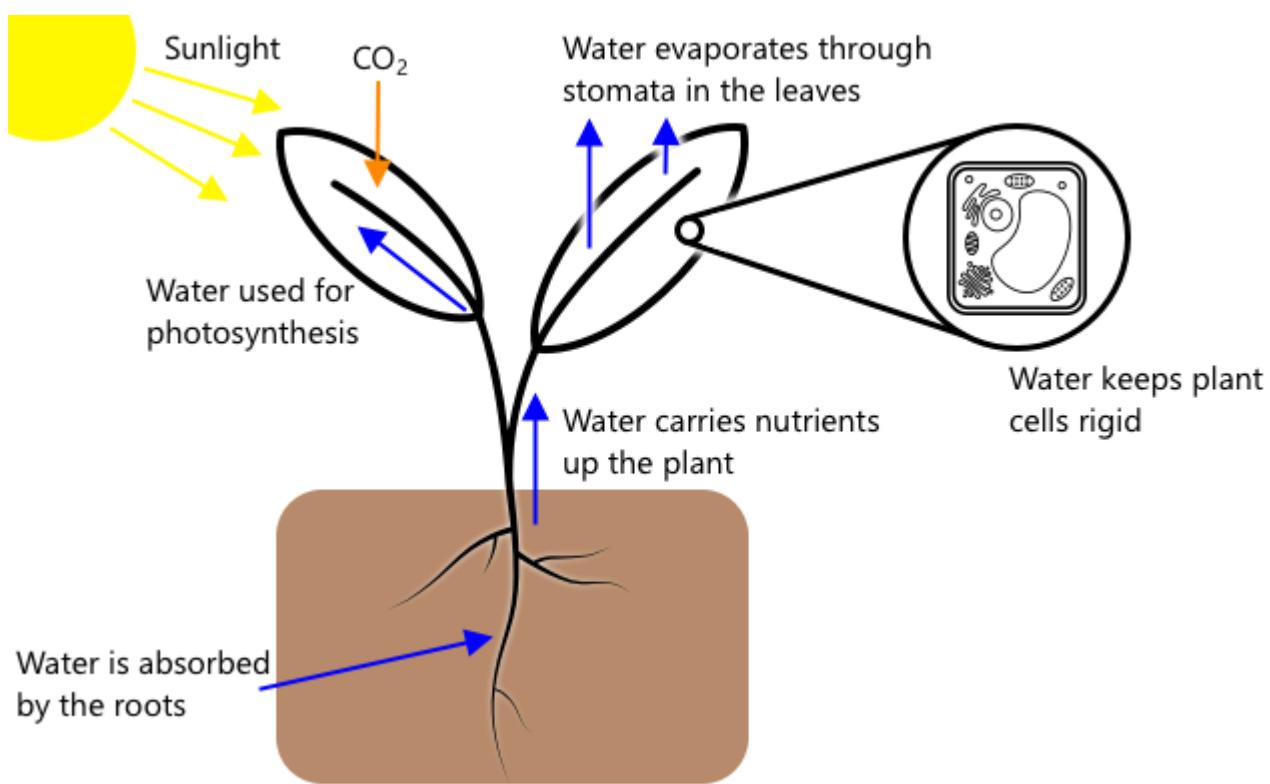
- Soil moisture
- How sensors communicate with IoT devices
- Measure the moisture levels in soil
- Sensor calibration

Soil moisture

Plants require water to grow. They absorb water throughout the entire plant, with the majority being absorbed by the root system. Water is used by the plant for three things:

- Photosynthesis - plants creates a chemical reaction with water, carbon dioxide and light to produce carbohydrates and oxygen.
- Transpiration - plants use water for diffusion of carbon dioxide from the air into the plant via pores in the leaves. This process also carries nutrients around the plant, and cools the plant, similar to how humans sweat.
- Structure - plants also need water to maintain their structure - they are 90% water (as opposed to humans at only 60%), and this water keeps the cells rigid. If a plant doesn't have enough water it

will wilt and eventually die.



Water is absorbed through plant roots then carried around the plant, being used for photosynthesis and plant structure. Plant by Alex Muravev / Plant Cell by Léa Lortal - all from the Noun Project

Do some research: how much water is lost through transpiration?

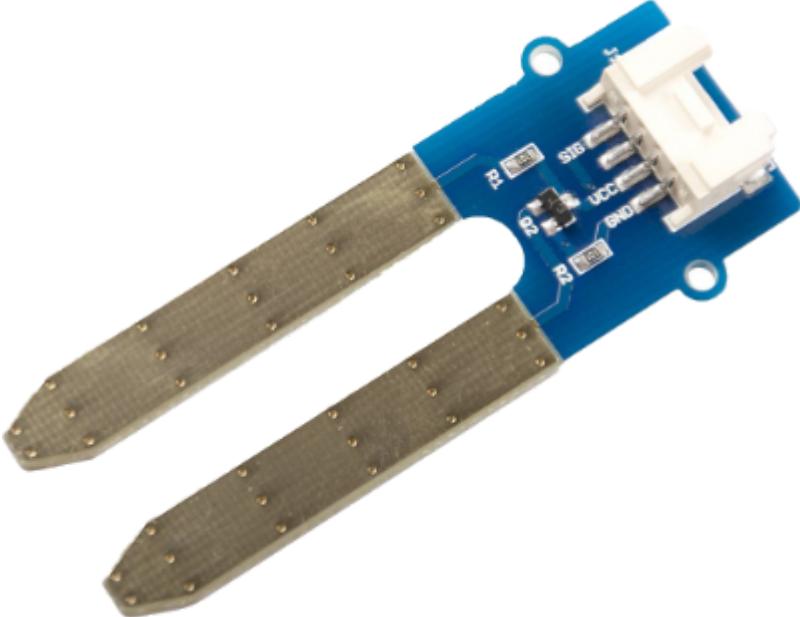
The root system provides water from moisture in the soil where the plant grows. Too little water in the soil and the plant cannot absorb enough to grow, too much water and roots cannot absorb enough oxygen needed to function. This leads to roots dying and the plant unable to get enough nutrients to survive.

For a farmer to get the best plant growth, the soil needs to be not too wet and not too dry. IoT devices can help with this by measuring soil moisture, allowing a farmer to only water when needed.

Ways to measure soil moisture

There are a range of different types of sensor you can use to measure soil moisture:

- Resistive - a resistive sensor has 2 probes that go into the soil. An electric current is sent to one probe, and received by the other. The sensor then measures the resistance of the soil - measuring how much the current drops at the second probe. Water is a good conductor of electricity, so the higher the water content of the soil, the lower the resistance.



💡 You can build a resistive soil moisture sensor using two pieces of metal, such as nails, separated by a couple of centimeters, and measuring the resistance between them using a multimeter.

- Capacitive - a capacitive moisture sensor measures the amount of electric charge that can be stored across a positive and a negative electrical plate, or capacitance. The capacitance of soil changes as the moisture level changes, and this can be converted to a voltage that can be measured by an IoT device. The wetter the soil, the lower the voltage that comes out.



These are both analog sensors, returning a voltage to indicate soil moisture. So how does this voltage get to your code? Before going any further with these sensors, let's look at how sensors and actuators communicate with IoT devices.

How sensors communicate with IoT devices

So far in these lessons you've learned about a number of sensors and actuators, and these have been communicating with your IoT dev kit if you've been doing the physical hardware labs. But how

does this communication work? How does a resistance measurement from a soil moisture sensor become a number you can use from code?

To communicate with most sensors and actuators you need some hardware, and a communication protocol - that is a well defined way for data to be sent and received. Take for example a capacitive soil moisture sensor:

- How is this sensor connected to the IoT device?
- If it measures a voltage that is an analog signal, it will need an ADC to create a digital representation of the value, and this value is sent as an alternating voltage to send 0s and 1s - but how long is each bit sent for?
- If the sensor returns a digital value, that will be a stream of 0s and 1s, again how long is each bit sent for?
- If the voltage is high for 0.1s is that a single 1 bit, or 2 consecutive 1 bits, or 10?
- At what point does the number start? Is `00001101` 25, or are the first 5 bits the end of the previous value?

The hardware provides the physical connectivity over which the data is sent, the different communication protocols ensure that the data is sent or received in the correct way so it can be interpreted.

General Purpose Input Output (GPIO) pins

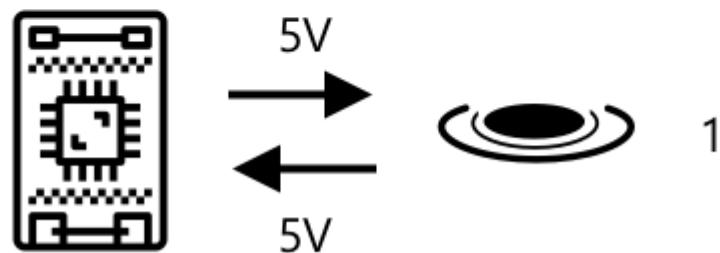
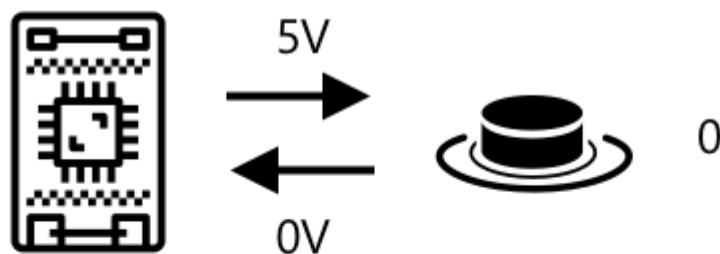
GPIO is a set of pins you can use to connect hardware to your IoT device, and are often available on IoT developer kits such as the Raspberry Pi or Wio Terminal. You can use the various communication protocols covered in this section over the GPIO pins. Some GPIO pins provide a voltage, usually 3.3V or 5V, some pins are ground, and others can be programmatically set to either send a voltage (output), or receive a voltage (input).

 An electrical circuit needs to connect a voltage to ground via whatever circuitry you are using. You can think of voltage as the positive (+ve) terminal of a battery and ground as the negative (-ve) terminal.

You can use GPIO pins directly with some digital sensors and actuators when you only care about on or off values - on referred to as high, off as low. Some examples are:

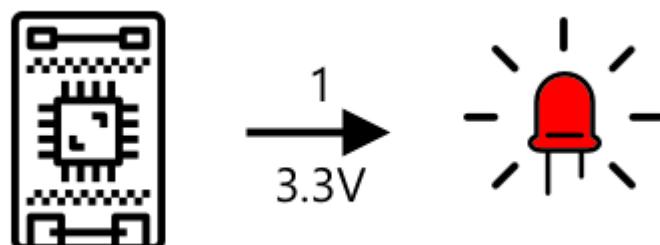
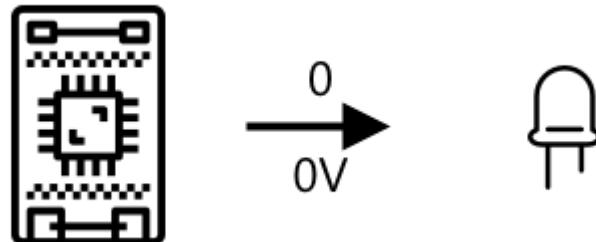
- Button. You can connect a button between a 5V pin and a pin set to input. When you press the button it completes a circuit between the 5V pin, through the button to the input pin. From code you can read the voltage at the input pin, and if it is high (5V) then the button is pressed, if it is low

(0v) then the button is not pressed. Remember the actual voltage itself is not read, instead you get a digital signal of 1 or 0 depending on if the voltage is above a threshold or not.



A button is sent 5 volts. When not pressed it returns 0 volts, or 0, when pressed it returns 5 volts, or 1. Microcontroller by Template / Button by Dan Hetteix - all from the [Noun Project](#)

- LED. You can connect an LED between an output pin and a ground pin (using a resistor otherwise you'll burn out the LED). From code you can set the output pin to high and it will send 3.3V, making a circuit from the 3.3V pin, through the LED, to the ground pin. This will light the LED.



An LED is sent a signal of 0 (3.3V), which lights the LED. If it is sent 0 (0v), the LED is not lit. LED by abderraouf omara / Microcontroller by Template - all from the [Noun Project](#)

For more advanced sensors, you can use GPIO pins to send and receive digital data directly with digital sensors and actuators, or via controller boards with ADCs and DACs to talk to analog sensors and actuators.

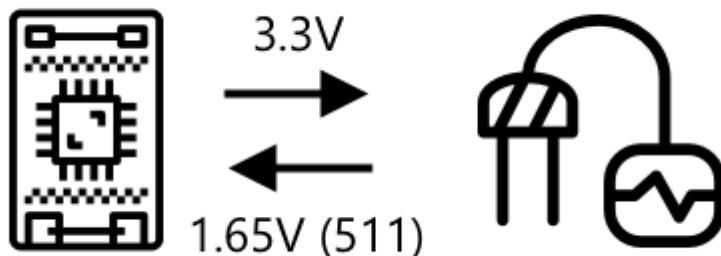
 if you are using a Raspberry Pi for these labs, the Grove Base Hat has hardware to convert analog sensor signals to digital to send over GPIO.

- If you have an IoT device with GPIO pins, locate these pins and find a diagram indicating which pins are voltage, ground or programmable.

Analog pins

Some devices, such as Arduino devices, provide analog pins. These are the same as GPIO pins, but instead of only supporting a digital signal, they have an ADC to convert voltage ranges to numerical values. Usually the ADC has a 10-bit resolution, meaning it converts voltages to a value of 0-1,023.

For example, on a 3.3V board, if the sensor returns 3.3V, the value returned would be 1,023. If the voltage returned was 1.65v, the value returned will be 511.



A soil moisture sensor sent 3.3V and returning 1.65v, or a reading of 511. probe by Adnen Kadri / Microcontroller by Template - all from the [Noun Project](#)

 Back in nightlight - lesson 3, the light sensor returned a value from 0-1,023. If you are using a Wio Terminal, the sensor was connected to an analog pin. If you are using a Raspberry Pi, then the sensor was connected to an analog pin on the base hat that has an integrated ADC to communicate over the GPIO pins. The virtual device was set to send a value from 0-1,023 to simulate an analog pin.

Soil moisture sensors rely on voltages, so will use analog pins and give values from 0-1,023.

Inter Integrated Circuit (I²C)

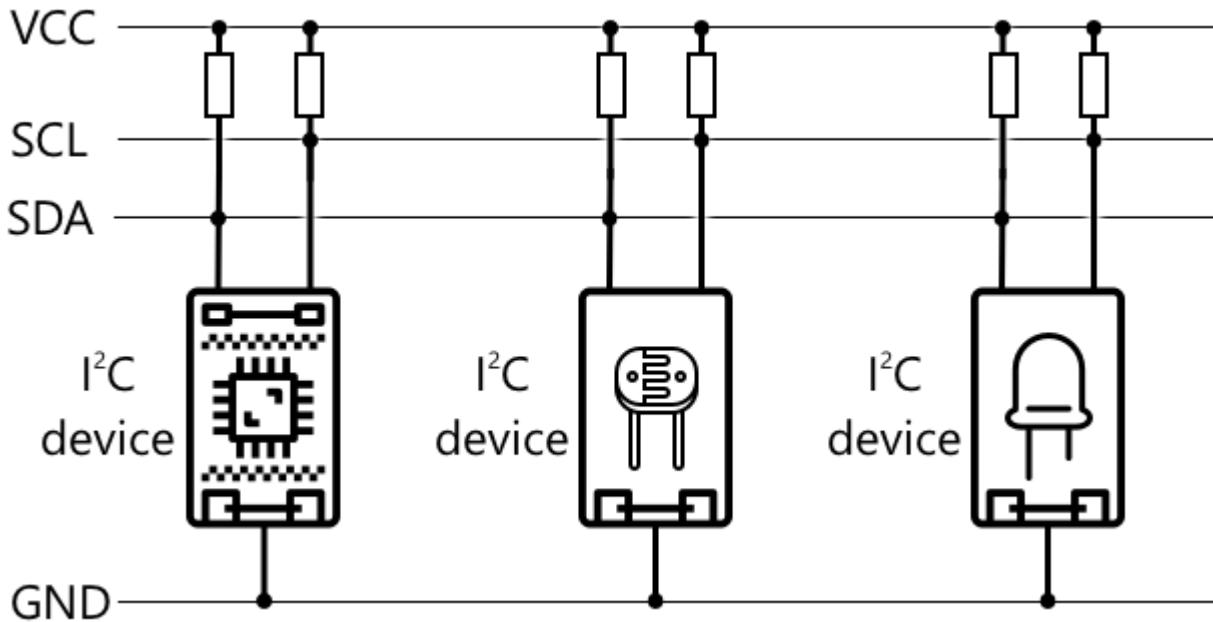
I²C, pronounced *I-squared-C*, is a multi-controller, multi-peripheral protocol, with any connected device able to act as a controller or peripheral communicating over the I²C bus (the name for a communication system that transfers data). Data is sent as addressed packets, with each packet containing the address of the connected device it is intended for.

 This model used to be referred to as master/slave, but this terminology is being dropped due to its association with slavery. The [Open Source Hardware Association has adopted controller/peripheral](#), but you may still see references to the old terminology.

Devices have an address that is used when they connect to the I²C bus, and is usually hard coded on the device. For example, each type of Grove sensor from Seeed has the same address, so all the light sensors have the same address, all the buttons have the same address that is different from the light sensor address. Some devices have ways to change the address, by changing jumper settings or soldering pins together.

I²C has a bus made of 2 main wires, along with 2 power wires:

Wire	Name	Description
SDA	Serial Data	This wire is for sending data between devices.
SCL	Serial Clock	This wire sends a clock signal at a rate set by the controller.
VCC	Voltage common collector	The power supply for the devices. This is connected to the SDA and SCL wires to provide their power via a pull-up resistor that switches the signal off when no device is the controller.
GND	Ground	This provides a common ground for the electrical circuit.



I²C bus with 3 devices connected to the SDA and SCL wires, sharing a common ground wire.
Microcontroller by Template / LED by abderraouf omara / Ldr by Eucalyp - all from the [Noun Project](#)

To send data, one device will issue a start condition to show it is ready to send data. It will then become the controller. The controller then sends the address of the device that it wants to communicate with, along with if it wants to read or write data. After the data has been transmitted, the controller sends a stop condition to indicate that it has finished. After this another device can become the controller and send or receive data.

I²C has speed limits, with 3 different modes running at fixed speeds. The fastest is High Speed mode with a maximum speed of 3.4Mbps (megabits per second), though very few devices support that speed. The Raspberry Pi for example, is limited to fast mode at 400Kbps (kilobits per second). Standard mode runs at 100Kbps.

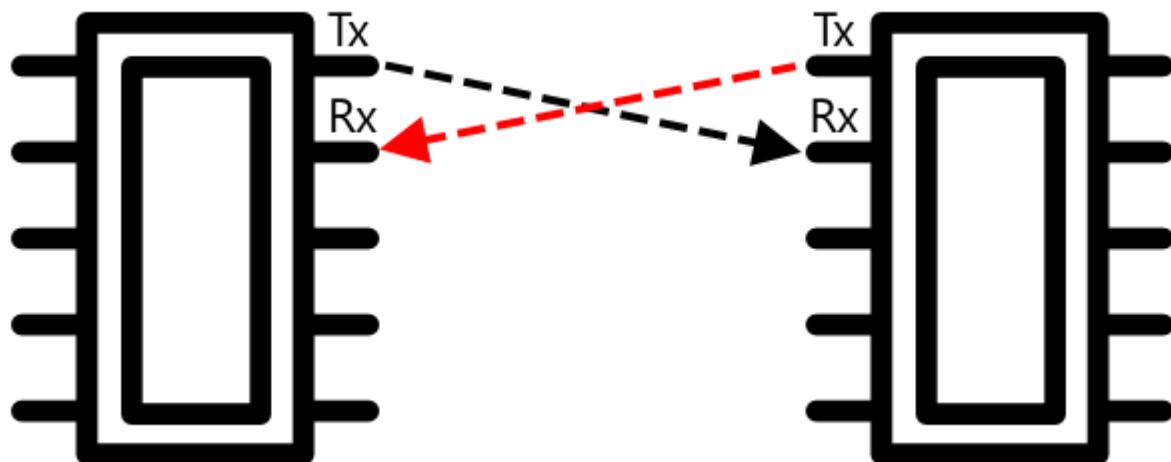
 If you are using a Raspberry Pi with a Grove Base hat as your IoT hardware, you will be able to see a number of I²C sockets on the board you can use to communicate with I²C sensors. Analog Grove sensors also use I²C with an ADC to send analog values as digital data, so the light sensor you used simulated an analog pin, with the value sent over I²C as the Raspberry Pi only supports digital pins.

Universal asynchronous receiver-transmitter (UART)

UART involves physical circuitry that allows two devices to communicate. Each device has 2 communication pins - transmit (Tx) and receive (Rx), with the Tx pin of the first device connected to

the Rx pin of the second, and with the Tx pin of the second device connected to the Rx pin of the first. This allows data to be sent in both directions.

- Device 1 transmits data from its Tx pin, which is received by device 2 on its Rx pin
- Device 1 receives data on its Rx pin that is transmitted by device 2 from its Tx pin



UART with the Tx pin on one chip connected to the Rx pin on another, and vice versa. chip by Astatine Lab - all from the [Noun Project](#)

🎓 The data is sent one bit at a time, and this is known as serial communication. Most operating systems and microcontrollers have serial ports, that is connections that can send and receive serial data that are available to your code.

UART devices have a baud rate (also known as Symbol rate), which is the speed that data will be sent and received in bits per second. A common baud rate is 9,600, meaning 9,600 bits (0s and 1s) of data are sent each second.

UART uses start and stop bits - that is it sends a start bit to indicate that it's about to send a byte (8 bits) of data, then a stop bit after it sends the 8 bits.

UART speed is dependent on hardware, but even the fastest implementations don't exceed 6.5 Mbps (megabits per second, or millions of bits, 0 or 1, sent per second).

You can use UART over GPIO pins - you can set one pin as Tx and another as Rx, then connect these to another device.

👤 If you are using a Raspberry Pi with a Grove Base hat as your IoT hardware, you will be able to see a UART socket on the board you can use to communicate with sensors that use the UART protocol.

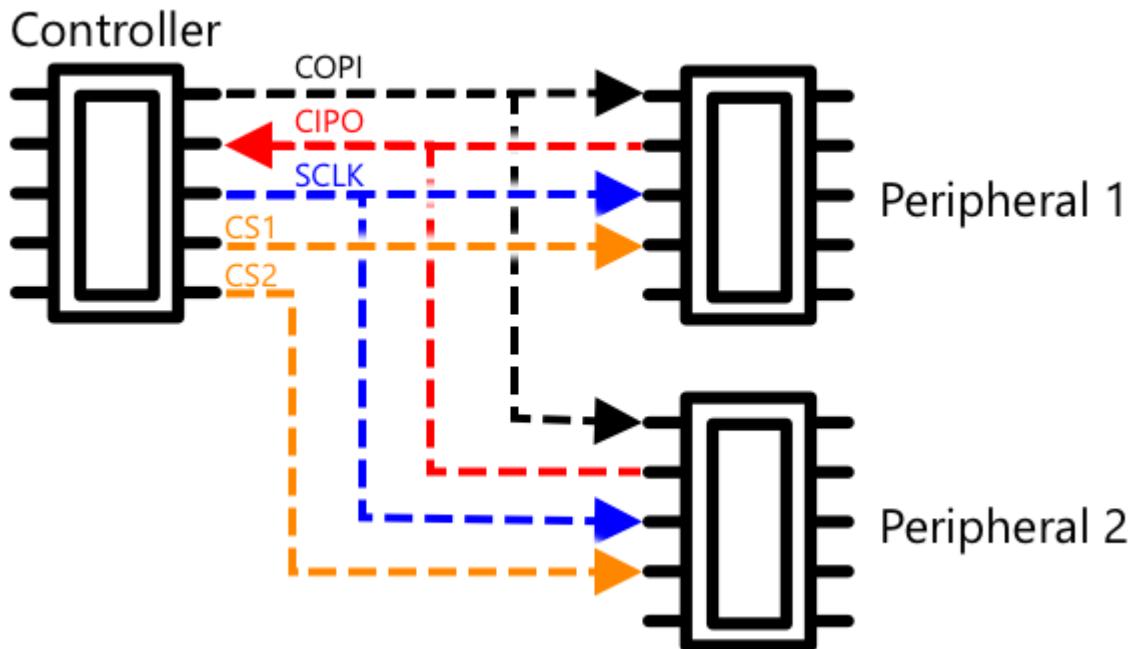
Serial Peripheral Interface (SPI)

SPI is designed for communicating over short distances, such as on a microcontroller to talk to a storage device such as flash memory. It is based on a controller/peripheral model with a single controller (usually the processor of the IoT device) interacting with multiple peripherals. The controller controls everything by selecting a peripheral and sending or requesting data.

💡 Like I²C, the terms controller and peripheral are recent changes, so you may see the older terms still used.

SPI controllers use 3 wires, along with 1 extra wire per peripheral. Peripherals use 4 wires. These wires are:

Wire	Name	Description
COPI	Controller Output, Peripheral Input	This wire is for sending data from the controller to the peripheral.
CIPO	Controller Input, peripheral Output	This wire is for sending data from the peripheral to the controller.
SCLK	Serial Clock	This wire sends a clock signal at a rate set by the controller.
CS	Chip Select	The controller has multiple wires, one per peripheral, and each wire connects to the CS wire on the corresponding peripheral.



The CS wire is used to activate one peripheral at a time, communicating over the COPI and CIPO wires. When the controller needs to change peripheral, it deactivates the CS wire connected to the currently active peripheral, then activates the wire connected to the peripheral it wants to communicate with next.

SPI is *full-duplex*, meaning the controller can send and receive data at the same time from the same peripheral using the COPI and CIPO wires. SPI uses a clock signal on the SCLK wire to keep the devices in sync, so unlike sending directly over UART it doesn't need start and stop bits.

There are no defined speed limits for SPI, with implementations often able to transmit multiple megabytes of data per second.

IoT developer kits often support SPI over some of the GPIO pins. For example, on a Raspberry Pi you can use GPIO pins 19, 21, 23, 24 and 26 for SPI.

Wireless

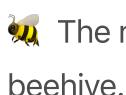
Some sensors can communicate over standard wireless protocols, such as Bluetooth (mainly Bluetooth Low Energy, or BLE), LoRaWAN (a **Long Range** low power networking protocol), or WiFi. These allow for remote sensors not physically connected to an IoT device.

One such example is in commercial soil moisture sensors. These will measure soil moisture in a field, then send the data over LoRaWan to a hub device, which will process the data or send it over the Internet. This allows the sensor to be away from the IoT device that manages the data, reducing power consumption and the need for large WiFi networks or long cables.

BLE is popular for advanced sensors such as fitness trackers work on the wrist. These combine multiple sensors and send the sensor data to an IoT device in the form of your phone via BLE.

 Do you have any bluetooth sensors on your person, in your house or in your school? These might include temperature sensors, occupancy sensors, device trackers and fitness devices.

One popular way for commercial devices to connect is Zigbee. Zigbee uses WiFi to form mesh networks between devices, where each device connects to as many nearby devices as possible, forming a large number of connections like a spiders web. When one device wants to send a message to the Internet it can send it to the nearest devices, which then forward it on to other nearby devices and so on, until it reaches a coordinator and can be sent to the Internet.



Measure the moisture levels in soil

You can measure the moisture level in soil using a soil moisture sensor, an IoT device, and a house plant or nearby patch of soil.

Task - measure soil moisture

Work through the relevant guide to measure soil moisture using your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Sensor calibration

Sensors rely on measuring electrical properties such as resistance or capacitance.

 Resistance, measured in ohms (Ω) is how much opposition there is to the electric current travelling through something. When a voltage is applied to a material, the amount of current that passes through it is dependant on the resistance of the material. You can read more on the [electrical resistance page on Wikipedia](#).

 Capacitance, measured in farads (F), is the ability of a component or circuit to collect and store electrical energy. You can read more on capacitance on the [capacitance page on Wikipedia](#).

These measurements are not always useful - imagine a temperature sensor that gave you a measurement of 22.5K Ω ! Instead the value measured needs to be converted into a useful unit by being calibrated - that is matching the values measured to the quantity measured to allow new measurements to be converted to the right unit.

Some sensors come pre-calibrated. For example the temperature sensor you used in the last lesson was already calibrated so that it can return a temperature measurement in °C. In the factory the first sensor created would be exposed to a range of known temperatures and the resistance measured.

This would then be used to build a calculation that can convert from the value measured in Ω (the unit of resistance) to $^{\circ}\text{C}$.

👉 The formula to calculate resistance from temperature is called the [Steinhart–Hart equation](#).

Soil moisture sensor calibration

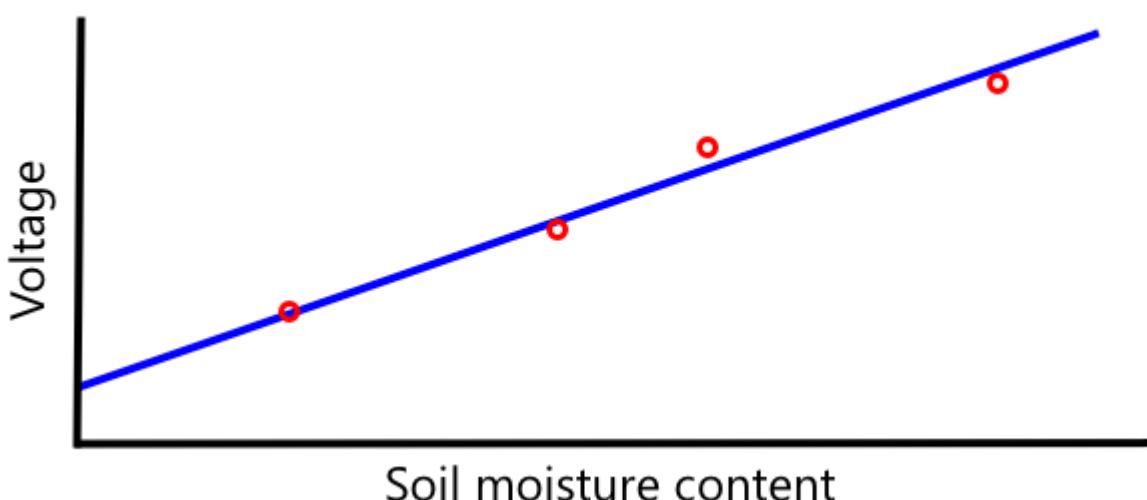
Soil moisture is measured using gravimetric or volumetric water content.

- Gravimetric is the weight of water in a unit weight of soil measured, as the number of kilograms of water per kilogram of dry soil
- Volumetric is the volume of water in a unit volume of soil measured, as the number of cubic metres of water per cubic metres of dry soil

🇺🇸 For Americans, because of the consistency of the units, these can be measured in pounds instead of kilograms or cubic feet instead of cubic metres.

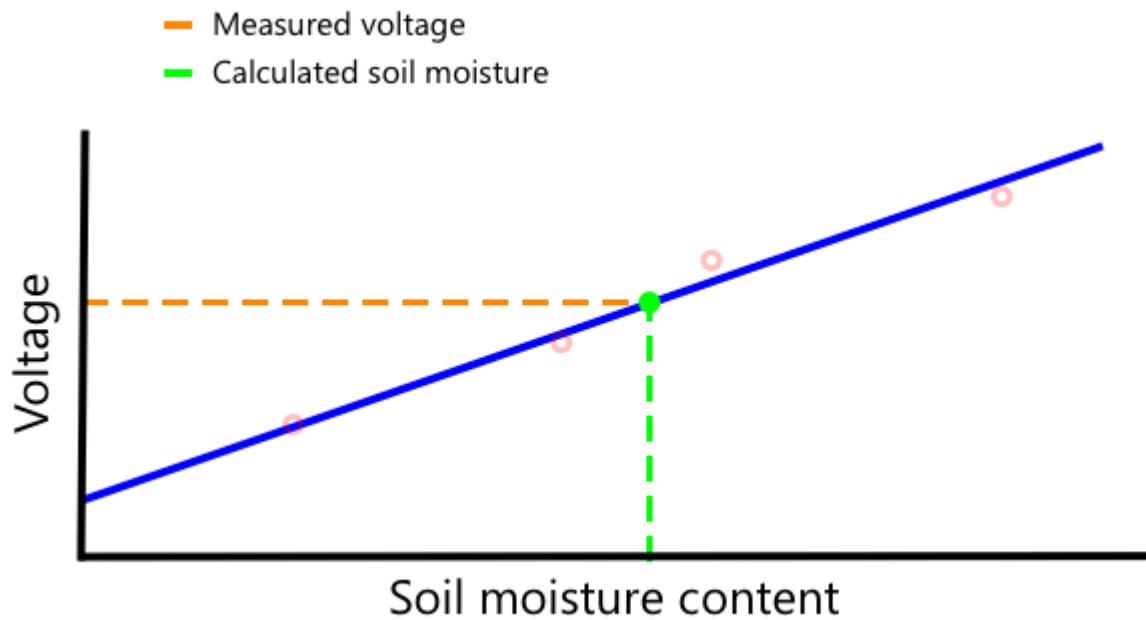
Soil moisture sensors measure electrical resistance or capacitance – this not only varies by soil moisture, but also soil type as the components in the soil can change its electrical characteristics. Ideally sensors should be calibrated – that is taking readings from the sensor and comparing them to measurements found using a more scientific approach. For example a lab can calculate the gravimetric soil moisture using samples of a specific field taken a few times a year, and these numbers used to calibrate the sensor, matching the sensor reading to the gravimetric soil moisture.

- Lab reading of soil moisture at a known voltage
- Fitted curve for predicting moisture values



The graph above shows how to calibrate a sensor . The voltage is captured for a soil sample that is then measured in a lab by comparing the moist weight to the dry weight (by measuring the weight wet, then drying in an oven and measuring dry). Once a few readings have been taken, this can be plotted on a graph and a line fitted to the points. This line can then be used to convert soil moisture sensor readings taken by an IoT device into actual soil moisture measurements.

💡 For resistive soil moisture sensors, the voltage increases as soil moisture increases. For capacitive soil moisture sensors, the voltage decreases as soil moisture increases, so the graphs for these would slope downwards, not upwards.



The graph above shows a voltage reading from a soil moisture sensor, and by following that to the line on the graph, the actual soil moisture can be calculated.

This approach means the farmer only needs to get a few lab measurements for a field, then they can use IoT devices to measure soil moisture - drastically speeding up the time to take measurements.

🚀 Challenge

Resistive and capacitive soil moisture sensors have a number of differences. What are these differences, and which type (if any) is the best for a farmer to use? Does this answer change between developing and developed countries?

Post-lecture quiz

Review & Self Study

Read up on the hardware and protocols used by sensors and actuators:

- [GPIO Wikipedia page](#)
- [UART Wikipedia page](#)
- [SPI Wikipedia page](#)
- [I²C Wikipedia page](#)
- [Zigbee Wikipedia page](#)

Assignment

[Calibrate your sensor](#)

Automated plant watering

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last lesson, you learned how to monitor soil moisture. In this lesson you will learn how to build the core components of an automated watering system that responds to soil moisture. You'll also learn about timing - how sensors can take a while to respond to changes, and how actuators can take time to change the properties being measured by sensors.

In this lesson we'll cover:

- [Control high power devices from a low power IoT device](#)
- [Control a relay](#)
- [Control your plant over MQTT](#)
- [Sensor and actuator timing](#)
- [Add timing to your plant control](#)

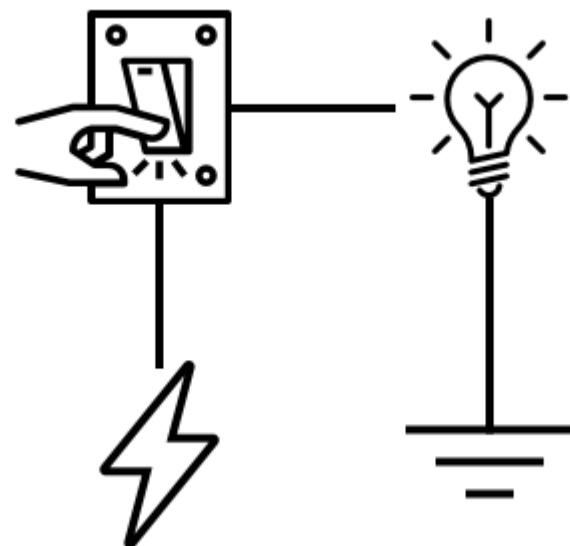
Control high power devices from a low power IoT device

IoT devices use a low voltage. While this is enough for sensors and low-power actuators like LEDs, this is too low to control larger hardware, such as a water pump used for irrigation. Even small pumps you could use for houseplants draw too much current for an IoT dev kit and would burn out the board.

💡 Current, measured in Amps (A), is the amount of electricity moving through a circuit.

Voltage provides the push, current is how much is pushed. You can read more about current on the [electric current page on Wikipedia](#).

The solution to this is to have a pump connected to an external power supply, and use an actuator to switch on the pump, similar to how you would switch on a light. It takes a tiny amount of power (in the form of energy in your body) for your finger to flip a switch, and this connects the light to mains electricity running at 110v/240v.



A light switch turns power on to a light. switch by Chattapat / lightbulb by Maxim Kulikov - all from the [Noun Project](#)

💡 Mains electricity refers to the electricity delivered to homes and businesses through national infrastructure in many parts of the world.

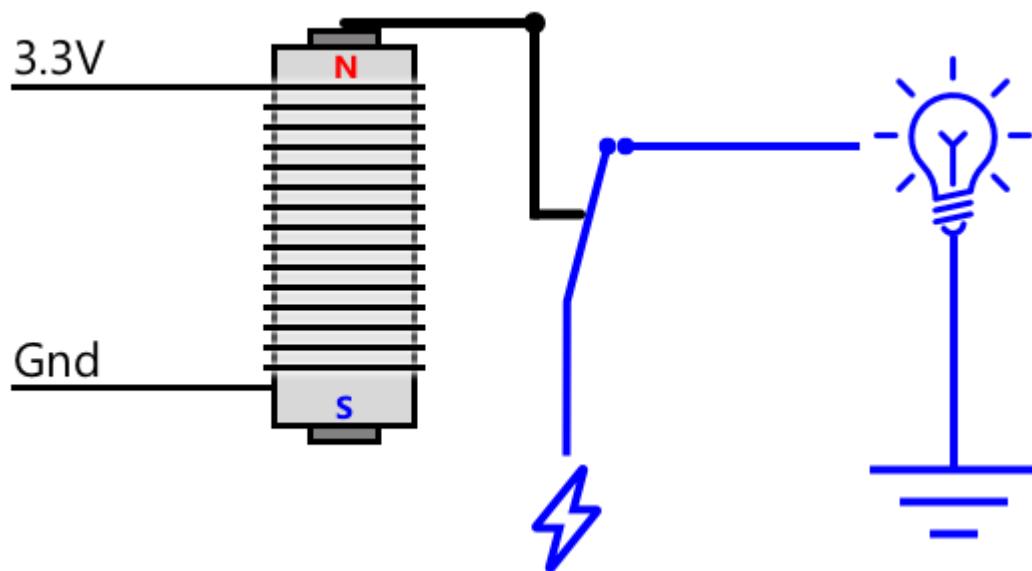
- ✓ IoT devices can usually provide 3.3V or 5V, at less than 1 amp (1A) of current. Compare this to mains electricity which is most often at 230V (120V in North America and 100V in Japan), and can provide power for devices that draw 30A.

There are a number of actuators that can do this, including mechanical devices you can attach to existing switches that mimic a finger turning them on. The most popular is a relay.

Relays

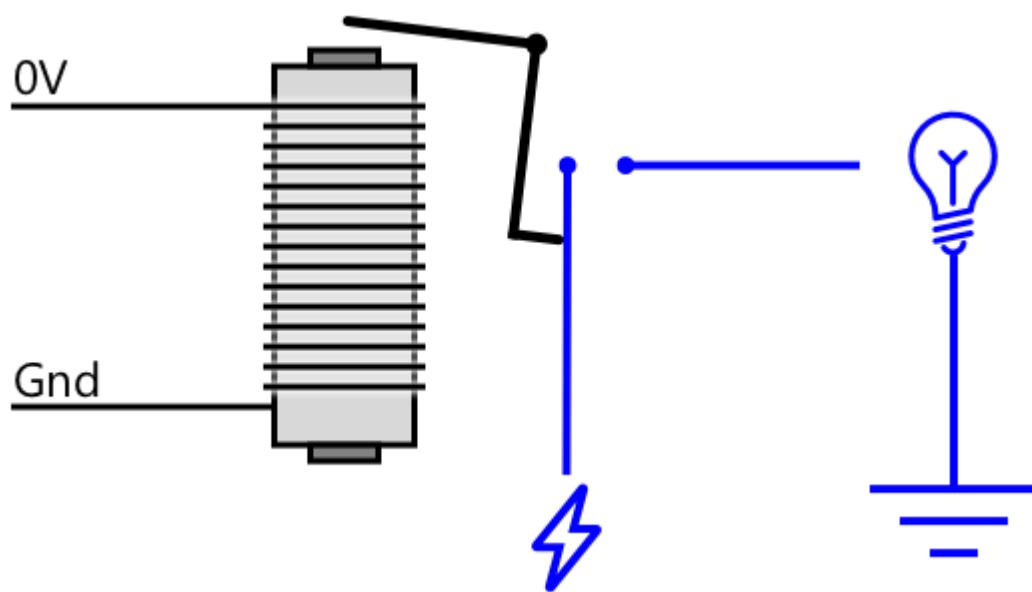
A relay is an electromechanical switch that converts an electrical signal into a mechanical movement that turns on a switch. The core of a relay is an electromagnet.

💡 Electromagnets are magnets that are created by passing electricity through a coil of wire. When the electricity is turned on, the coil becomes magnetized. When the electricity is turned off, the coil loses its magnetism.



When on, the electromagnet creates a magnetic field, turning on the switch for the output circuit. lightbulb by Maxim Kulikov - from the [Noun Project](#)

In a relay, a control circuit powers the electromagnet. When the electromagnet is on, it pulls a lever that moves a switch, closing a pair of contacts and completing an output circuit.



When off, the electromagnet doesn't create a magnetic field, turning off the switch for the output circuit. lightbulb by Maxim Kulikov - from the [Noun Project](#)

When the control circuit is off, the electromagnet turns off, releasing the lever and opening the contacts, turning off the output circuit. Relays are digital actuators - a high signal to the relay turns it on, a low signal turns it off.

The output circuit can be used to power additional hardware, like an irrigation system. The IoT device can turn the relay on, completing the output circuit that powers the irrigation system, and plants get watered. The IoT device can then turn the relay off, cutting the power to the irrigation system, turning the water off.



In the video above, a relay is turned on. An LED on the relay lights up to indicate it is on (some relay boards have LEDs to indicate if the relay is on or off), and power is sent to the pump, turning it on and pumping water into a plant.

💡 Relays can also be used to switch between two output circuits instead of turning one on and off. As the lever moves, it moves a switch from completing one output circuit to completing a different output circuit, usually sharing a common power connection, or common ground connection.

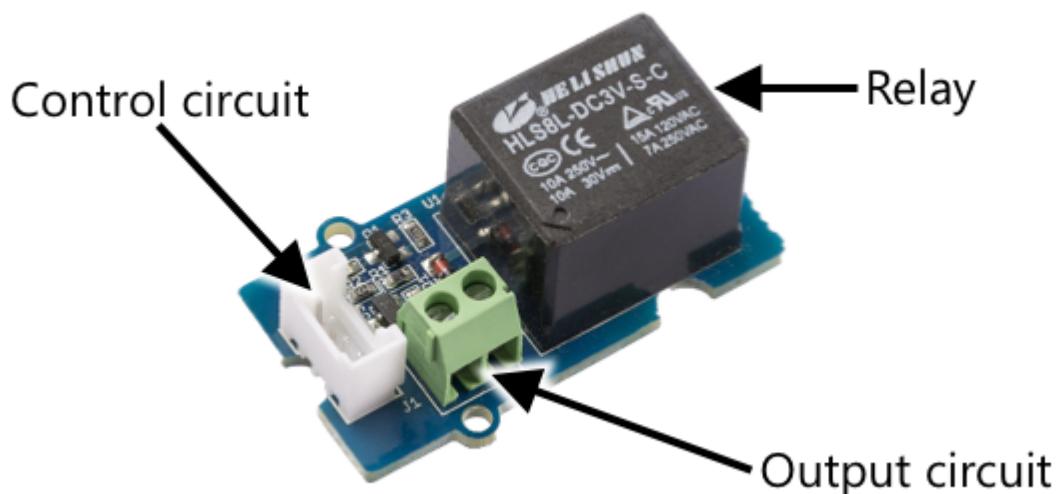
- ✓ Do some research: There are multiple types of relays, with differences such as if the control circuit turns the relay on or off when power is applied, or multiple output circuits. Find out about these different types.

When the lever moves, you can usually hear it make contact with the electromagnet with a well defined click noise.

💡 A relay can be wired so that making the connection actually breaks power to the relay, turning the relay off, which then sends power to the relay turning it back on again, and so on. This means the relay will click incredibly fast making a buzzing noise. This is how some of the first buzzers used in electric doorbells worked.

Relay power

The electromagnet doesn't need a lot of power to activate and pull the lever, it can be controlled using the 3.3V or 5V output from an IoT dev kit. The output circuit can carry a lot more power, depending on the relay, including mains voltage or even higher power levels for industrial use. This way an IoT Dev kit can control an irrigation system, from a small pump for a single plant, up to a massive industrial system for an entire commercial farm.



The image above shows a Grove relay. The control circuit connects to an IoT device and turns the relay on or off using 3.3V or 5V. The output circuit has two terminals, either one can be power or ground. The output circuit can handle up to 250V at 10A, enough for a range of mains-powered devices. You can get relays that can handle even high power levels.



In the image above, power is supplied to a pump via a relay. There is a red wire connecting the +5V terminal of a USB power supply to one terminal of the output circuit of the relay, and another red wire connecting the other terminal of the output circuit to the pump. A black wire connects the pump to the ground on the USB power supply. When the relay turns on, it completes the circuit, sending 5V to the pump, turning the pump on.

Control a relay

You can control a relay from your IoT Dev kit.

Task - control a relay

Work through the relevant guide to control a relay using your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Control your plant over MQTT

So far your relay is controlled by the IoT device directly based off a single soil moisture reading. In a commercial irrigation system, the control logic will be centralized, allowing it to make decisions on watering using data from multiple sensors, and allowing any configuration to be changed in one single place. To simulate this, you can control the relay over MQTT.

Task - control the relay over MQTT

1. Add the relevant MQTT libraries/pip packages and code to your `soil-moisture-sensor` project to connect to MQTT. Name the client ID as `soilmoisturesensor_client` prefixed by your ID.

⚠ You can refer to [the instructions for connecting to MQTT in project 1, lesson 4 if needed.](#)

2. Add the relevant device code to send telemetry with the soil moisture settings. For the telemetry message, name the property `soil_moisture`.

⚠ You can refer to [the instructions for sending telemetry to MQTT in project 1, lesson 4 if needed.](#)

3. Create some local server code to subscribe to telemetry and send a command to control the relay in a folder called `soil-moisture-sensor-server`. Name the property in the command message `relay_on`, and set the client ID as `soilmoisturesensor_server` prefixed by your ID. Keep the same structure as the server code you wrote for project 1, lesson 4 as you will be adding to this code later in this lesson.

⚠ You can refer to [the instructions for sending telemetry to MQTT and sending commands over MQTT in project 1, lesson 4 if needed.](#)

4. Add the relevant device code to control the relay from received commands, using the `relay_on` property from the message. Send true for `relay_on` if the `soil_moisture` is

greater than 450, otherwise send false, the same as the logic you added for the IoT device earlier.

⚠ You can refer to [the instructions for responding to commands from MQTT in project 1, lesson 4](#) if needed.

💡 You can find this code in the [code-mqtt](#) folder.

Make sure the code is running on your device and local server, and test it out by changing soil moisture levels, either by changing the values sent by the virtual sensor, or by changing the moisture levels of the soil by adding water or removing the sensor from the soil.

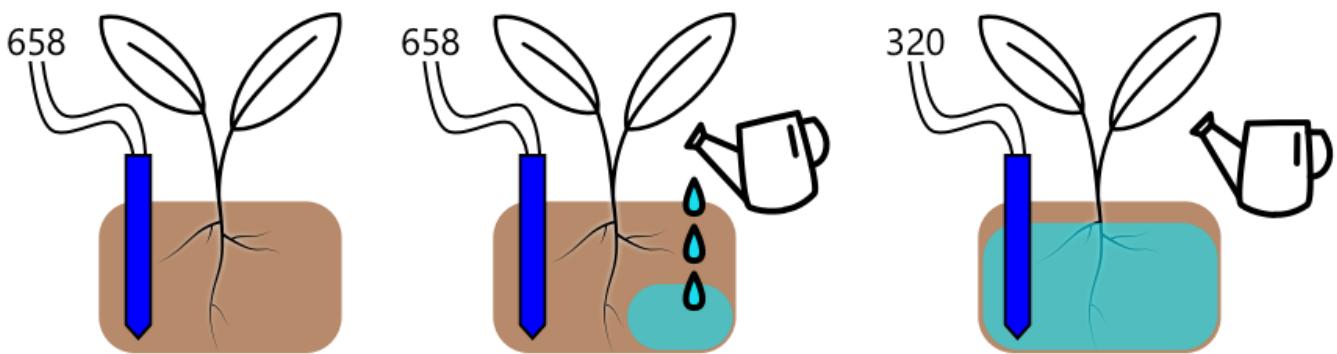
Sensor and actuator timing

Back in lesson 3 you built a nightlight - an LED that turns on as soon as a low level of light was detected by a light sensor. The light sensor detected a change in light levels instantly, and the device was able to respond quickly, only limited by the length of the delay in the `loop` function or `while True: loop`. As an IoT developer, you can't always rely on such a fast feedback loop.

Timing for soil moisture

If you did the last lesson on soil moisture using a physical sensor, you would have noticed that it took a few seconds for the soil moisture reading to drop after you watered your plant. This is not because the sensor is slow, but because it takes time for water to soak through the soil.

💡 If you watered too close to the sensor you may have seen the reading drop quickly, then come back up - this is caused by water near the sensor spreading throughout the rest of the soil, reducing the soil moisture by the sensor.



A soil moisture measurement of 658 doesn't change during watering, it only drops to 320 after watering when water has soaked through the soil. Plant by Alex Muravev / Watering Can by Daria Moskvina - all from the [Noun Project](#)

In the diagram above, a soil moisture reading shows 658. The plant is watered, but this reading doesn't change immediately, as the water has yet to reach the sensor. Watering can even finish before the water reaches the sensor and the value drops to reflect the new moisture level.

If you were writing code to control an irrigation system via a relay based off soil moisture levels, you would need to take this delay into consideration and build smarter timing into your IoT device.

Take a moment to think about how you might do this.

Control sensor and actuator timing

Imagine you have been tasked with building an irrigation system for a farm. Based on the soil type, the ideal soil moisture level for the plants grown has been found to match an analog voltage reading of 400-450.

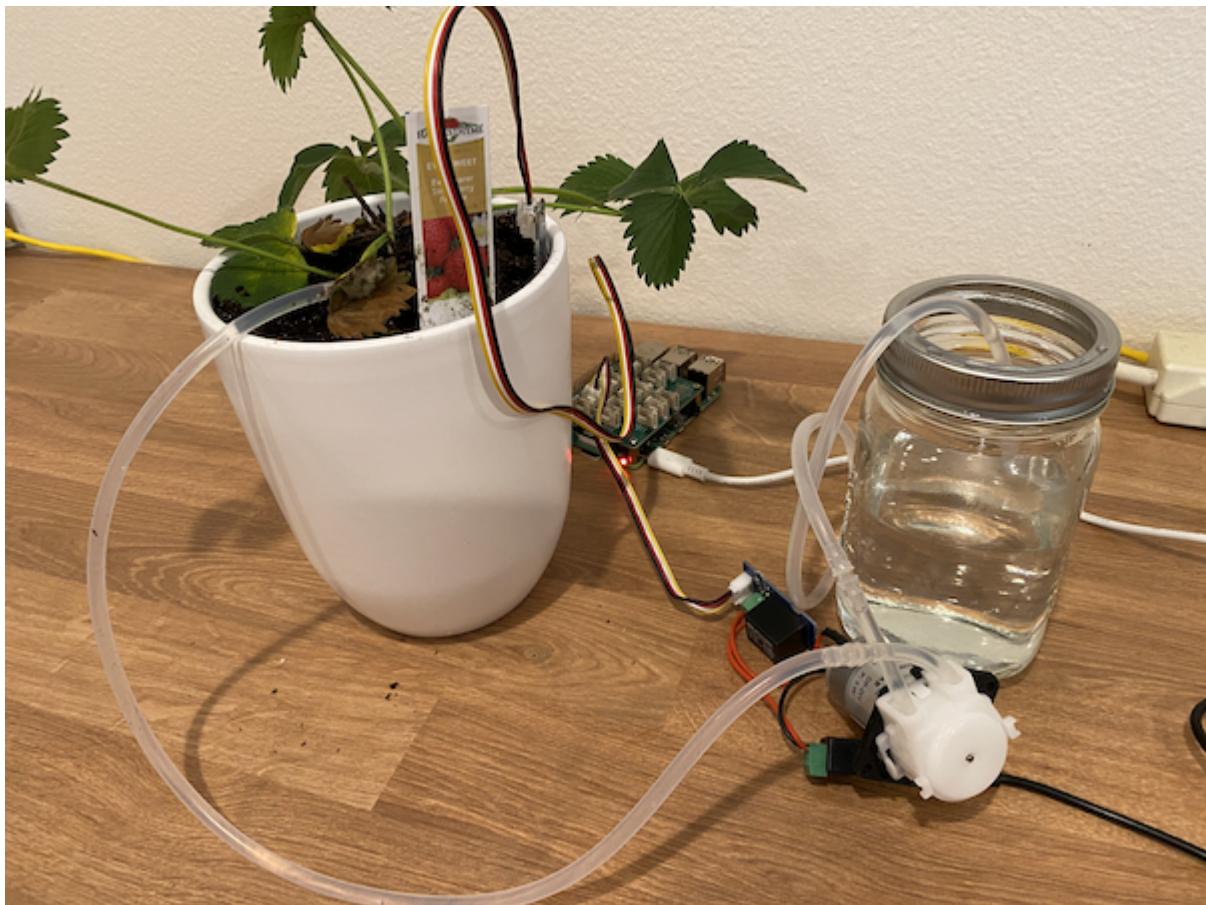
You could program the device in the same way as the nightlight - all the time the sensor reads above 450, turn on a relay to turn on a pump. The problem is that water takes a while to get from the pump, through the soil to the sensor. The sensor will stop the water when it detects a level of 450, but the water level will continue dropping as the pumped water keeps soaking through the soil. The end result is wasted water, and the risk of root damage.

Remember - too much water can be as bad for plants as too little, and wastes a precious resource.

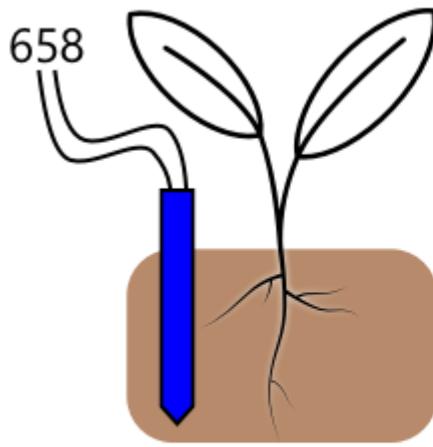
The better solution is to understand that there is a delay between the actuator turning on and the property that the sensor reads changing. This means not only should the sensor wait for a while before measuring the value again, but the actuator needs to turn off for a while before the next sensor measurement is taken.

How long should the relay be on each time? It's better to err on the side of caution and only turn the relay on for a short time, then wait for the water to soak through, then re-check the moisture levels. After all, you can always turn it on again to add more water, you can't remove water from the soil.

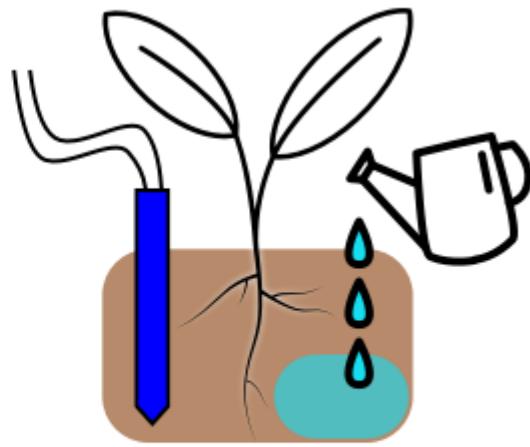
👉 This kind of timing control is very specific to the IoT device you are building, the property you are measuring and the sensors and actuators used.



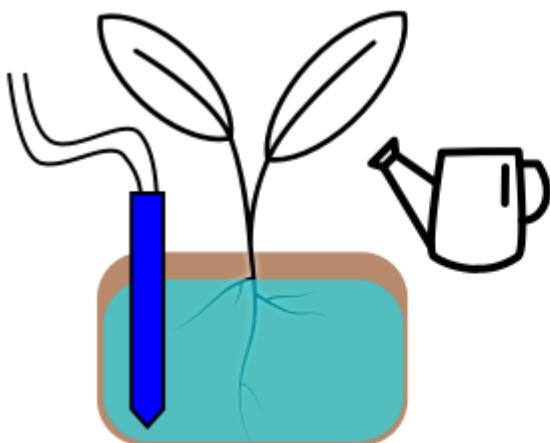
For example, I have a strawberry plant with a soil moisture sensor and a pump controlled by a relay. I've observed that when I add water it takes about 20 seconds for the soil moisture reading to stabilize. This means I need to turn the relay off and wait 20 seconds before checking the moisture levels. I'd rather have too little water than too much - I can always turn the pump on again, but I can't take water out of the plant.



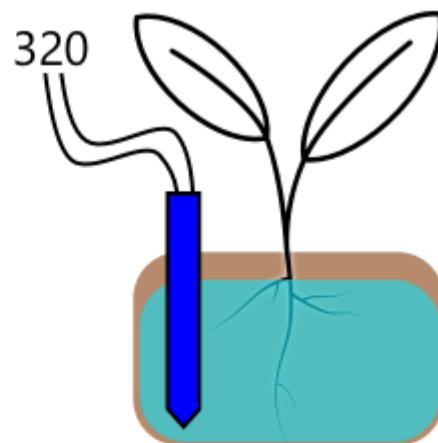
Step 1 - take measurement



Step 2 - add water



Step 3 - wait for water to soak through the soil



Step 4 - retake measurement

Measure, add water, wait, remeasure. Plant by Alex Muravev / Watering Can by Daria Moskvina - all from the Noun Project

This means the best process would be a watering cycle that is something like:

- Turn on the pump for 5 seconds
- Wait 20 seconds
- Check the soil moisture
- If the level is still above what I need, repeat the above steps

5 seconds could be too long for the pump, especially if the moisture levels are only slightly above the required level. The best way to know what timing to use is to try it, then adjust when you have sensor data, with a constant feedback loop. This can even lead to more granular timing, such as turning the pump on for 1s for every 100 above the required soil moisture, instead of a fixed 5 seconds.

Do some research: Are there other timing considerations? Can the plant be watered any time that the soil moisture is too low, or are there specific times of day that are good and bad times to water

 Weather predictions can also be taken into consideration when controlling automated watering systems for outdoor growing. If rain is expected, then the watering can be put on hold till after the rain finishes. At that point the soil may be moist enough that it doesn't need watering, much more efficient than wasting water by watering just before rain.

Add timing to your plant control server

The server code can be modified to add control around the timing of the watering cycle, and waiting for the soil moisture levels to change. The server logic for controlling the relay timing is:

1. Telemetry message received
2. Check the soil moisture level
3. If it's ok, do nothing. If the reading is too high (meaning the soil moisture is too low) then:
 1. Send a command to turn the relay on
 2. Wait for 5 seconds
 3. Send a command to turn the relay off
 4. Wait for 20 seconds for the soil moisture levels to stabilize

The watering cycle, the process from receiving the telemetry message to being ready to process soil moisture levels again, takes about 25 seconds. We're sending soil moisture levels every 10 seconds, so there is an overlap where a message is received whilst the server is waiting for soil moisture levels to stabilize, which could start another watering cycle.

There are two options to work around this:

- Change the IoT device code to only send telemetry every minute, this way the watering cycle will be completed before the next message is sent
- Unsubscribe from the telemetry during the watering cycle

The first option is not always a good solution for large farms. The farmer might want to capture the soil moisture levels as the soil is being watered for later analysis, for example to be aware of water flow in different areas on the farm to guide more targeted watering. The second option is better - the code is just ignoring telemetry when it can't use it, but the telemetry is still there for other services that might subscribe to it.

 IoT data is not sent from only one device to only one service, instead many devices can send data to a broker, and many services can listen to the data off the broker. For example, one service could listen to soil moisture data and store it in a database for analysis at a later date. Another service can also listen to the same telemetry to control an irrigation system.

Task - add timing to your plant control server

Update your server code to run the relay for 5 seconds, then wait 20 seconds.

1. Open the `soil-moisture-sensor-server` folder in VS Code if it isn't already open. Make sure the virtual environment is activated.
2. Open the `app.py` file
3. Add the following code to the `app.py` file below the existing imports:

```
python  
import threading
```

This statement imports `threading` from Python libraries, threading allows python to execute other code while waiting.

4. Add the following code before the `handle_telemetry` function that handles telemetry messages received by the server code:

```
python  
water_time = 5  
wait_time = 20
```

This defines how long to run the relay for (`water_time`), and how long to wait afterwards to check the soil moisture (`wait_time`).

5. Below this code, add the following:

```
python  
def send_relay_command(client, state):  
    command = { 'relay_on' : state }  
    print("Sending message:", command)  
    client.publish(server_command_topic, json.dumps(command))
```

This code defines a function called `send_relay_command` that sends a command over MQTT to control the relay. The telemetry is created as a dictionary, then converted to a JSON string. The value passed in to `state` determines if the relay should be on or off.

6. After the `send_relay_code` function, add the following code:

python

```
def control_relay(client):
    print("Unsubscribing from telemetry")
    mqtt_client.unsubscribe(client_telemetry_topic)

    send_relay_command(client, True)
    time.sleep(water_time)
    send_relay_command(client, False)

    time.sleep(wait_time)

    print("Subscribing to telemetry")
    mqtt_client.subscribe(client_telemetry_topic)
```

This defines a function to control the relay based off the required timing. It starts by unsubscribing from telemetry so that soil moisture messages are not processed whilst the watering is happening. Next it sends a command to turn the relay on. It then waits for the `water_time` before sending a command to turn the relay off. Finally it waits for the soil moisture levels to stabilize for `wait_time` seconds. It then re-subscribes to telemetry.

7. Change the `handle_telemetry` function to the following:

python

```
def handle_telemetry(client, userdata, message):
    payload = json.loads(message.payload.decode())
    print("Message received:", payload)

    if payload['soil_moisture'] > 450:
        threading.Thread(target=control_relay, args=(client,)).start()
```

This code checks the soil moisture level. If it is greater than 450, the soil needs watering, so it calls the `control_relay` function. This function is run on a separate thread, running in the background.

8. Make sure your IoT device is running, then run this code. Change the soil moisture levels and observe what happens to the relay - it should turn on for 5 seconds then remain off for at least 20 seconds, only turning on if the soil moisture levels are not sufficient.

```
(.venv) → soil-moisture-sensor-server ✘ python app.py
Message received: {'soil_moisture': 457}
Unsubscribing from telemetry
Sending message: {'relay_on': True}
Sending message: {'relay_on': False}
Subscribing to telemetry
Message received: {'soil_moisture': 302}
```

A good way to test this in a simulated irrigation system is to use dry soil, then pour water in manually whilst the relay is on, stopping pouring when the relay turns off.



You can find this code in the [code-timing](#) folder.



If you want to use a pump to build a real irrigation system, then you can use a [6V water pump](#) with a [USB terminal power supply](#). Make sure the power to or from the pump is connected via the relay.



Challenge

Can you think of any other IoT or other electrical devices that have a similar problem where it takes a while for the results of the actuator to reach the sensor. You probably have a couple in your house or school.

- What properties do they measure?
- How long does it take for the property to change after an actuator is used?
- Is it ok for the property to change past the required value?
- How can it be returned back to the required value if needed?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

Read more on relays including their historical use in telephone exchanges on the [relay Wikipedia page](#).

Assignment

[Build a more efficient watering cycle](#)

Migrate your plant to the cloud

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last lesson, you learned how to connect your plant to an MQTT broker and controlled a relay from some server code running locally. This forms the core of the kind of internet-connected automated watering system that is used from individual plants at home up to commercial farms.

The IoT device communicated with a public MQTT broker as a way to demonstrate the principles, but this is not the most reliable or secure way. In this lesson you will learn about the cloud, and the IoT capabilities provided by public cloud services. You will also learn how to migrate your plant to one of these cloud services from the public MQTT broker.

In this lesson we'll cover:

- [What is the cloud?](#)
- [Create a cloud subscription](#)
- [Cloud IoT services](#)

- [Create an IoT service in the cloud](#)
- [Communicate with IoT Hub](#)
- [Connect your device to the IoT service](#)

What is the cloud?

Before the cloud, when a company wanted to provide services to their employees (such as databases or file storage), or to the public (such as websites), they would build and run a data center. This ranged from a room with a small number of computers, to a building with many computers. The company would manage everything, including:

- Buying computers
- Hardware maintenance
- Power and cooling
- Networking
- Security, including securing the building and securing the software on the computers
- Software installation and updates

This could be very expensive, require a wide range of skilled employees, and be very slow to change when needed. For example, if an online store needed to plan for a busy holiday season, they would need to plan months in advance to buy more hardware, configure it, install it and install the software to run their sales process. After the holiday season was over and sales dropped back down, they would be left with computers they've paid for sitting idle till the next busy season.

Do you think this would allow companies to move quickly? If an online clothing retailer suddenly got popular due to a celebrity being seen in their clothes, would they be able to increase their computing power quickly enough to support the sudden influx of orders?

Someone else's computer

The cloud is often jokingly referred to as 'someone else's computer'. The initial idea was simple – instead of buying computers, you rent someone else's computer. Someone else, a cloud computing provider, would manage huge data centers. They would be responsible for buying and installing the hardware, managing power and cooling, networking, building security, hardware and software updates, everything. As a customer, you would rent the computers you need, renting more as demand spikes, then reducing the number you rent if demand drops. These cloud data centers are all around the world.



These data centers can be multiple square kilometers in size. The images above were taken a few years ago at a Microsoft cloud data center, and show the initial size, along with a planned expansion. The area cleared for the expansion is over 5 square kilometers.

💡 These data centers require such large amounts of power that some have their own power stations. Because of their size and the level of investment from the cloud providers, they are usually very environmentally friendly. They are more efficient than huge numbers of small data centers, they run mostly on renewable energy, and cloud providers work hard to reduce waste, cut water usage, and replant forests to make up for those cut down to provide space to build data centers. You can read more about how one cloud provider is working on sustainability on the [Azure sustainability site](#).

✓ Do some research: Read up on the major clouds such as [Azure from Microsoft](#) or [GCP from Google](#). How many data centers do they have, and where are they in the world?

Using the cloud keeps costs down for companies, and allows them to focus on what they do best, leaving the cloud computing expertise in the hands of the provider. Companies no longer need to rent or buy data center space, pay different providers for connectivity and power, or employ experts. Instead, they can pay one monthly bill to the cloud provider to have everything taken care off.

The cloud provider can then use economies of scale to drive costs down, buying computers in bulk at lower costs, investing in tooling to reduce their workload for maintenance, even designing and building their own hardware to improve their cloud offering.

Microsoft Azure

Azure is the developer cloud from Microsoft, and this is the cloud you will be using for these lessons. The video below gives a short overview of Azure:



Create a cloud subscription

To use services in the cloud, you will need to sign up for a subscription with a cloud provider. For this lesson, you will be signing up for a Microsoft Azure subscription. If you already have an Azure subscription you can skip this task. The subscription details described here are correct at the time of writing, but may change.

 If you are accessing these lessons through your school, you may already have an Azure subscription available to you. Check with your teacher.

There are two different types of free Azure subscription you can sign up for:

- **Azure for Students** - This is a subscription designed for students 18+. You don't need a credit card to sign up, and you use your school email address to validate that you are a student. When you sign up you get US\$100 to spend on cloud resources, along with free services including a

free version of an IoT service. This lasts 12 months, and you can renew every year that you remain a student.

- **Azure free subscription** - This is a subscription for anyone who is not a student. You will need a credit card to sign up to for the subscription, but your card will not be billed, this is just used to verify you are a real human, not a bot. You get \$200 of credit to use in the first 30 days on any service, along with free tiers of Azure services. Once your credit has been used up, your card will not be charged unless you convert to a pay as you go subscription.

 Microsoft does offer an Azure for Students Starter subscription for students under 18, but at the time of writing this doesn't support any IoT services.

Task - sign up for a free cloud subscription

If you are a student aged 18+, then you can sign up for an Azure for Students subscription. You will need to validate with a school email address. You can do this in one of two ways:

- Sign up for a GitHub student developer pack at education.github.com/pack. This gives you access to a range of tools and offers, including GitHub and Microsoft Azure. Once you sign up for the developer pack, you can then activate the Azure for Students offer.
- Sign up directly for an Azure for Students account at azure.microsoft.com/free/students.

 If your school email address is not recognized, raise an [issue in this repo](#) and we'll see if it can be added to the Azure for Students allow list.

If you are not a student, or you don't have a valid school email address, then you can sign up for an Azure Free subscription.

- Sign up for an Azure Free Subscription at azure.microsoft.com/free

Cloud IoT services

The public test MQTT broker you have been using is a great tool when learning, but has a number of drawbacks as a tool to use in a commercial setting:

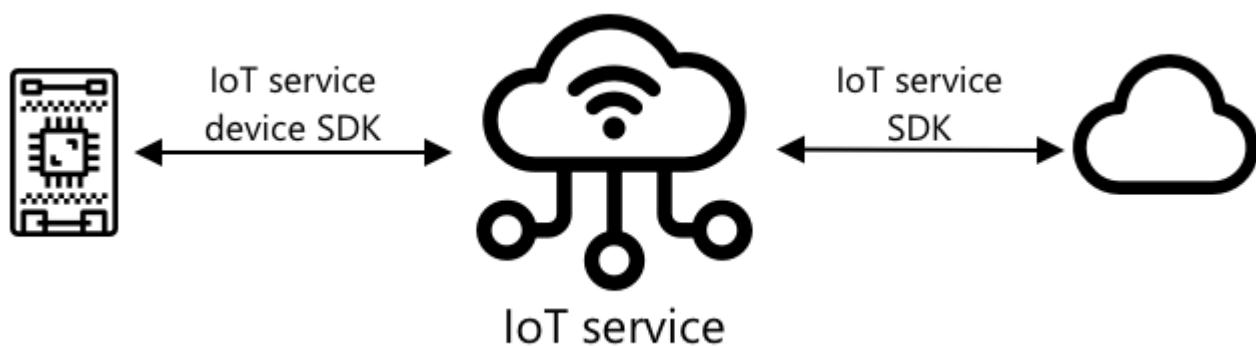
- Reliability - it's a free service with no guarantees, and can be turned off at any time

- Security - it is public, so anyone could listen to your telemetry or send commands to control your hardware
- Performance - it is designed for only a few test messages, so wouldn't cope with a large amount of messages being sent
- Discovery - there is no way to know what devices are connected

IoT services in the cloud solve these problems. They are maintained by large cloud providers who invest heavily in reliability and are on hand to fix any issues that might arise. They have security baked in to stop hackers reading your data or sending rogue commands. They are also high performance, being able to handle many millions of messages every day, taking advantage of the cloud to scale as needed.

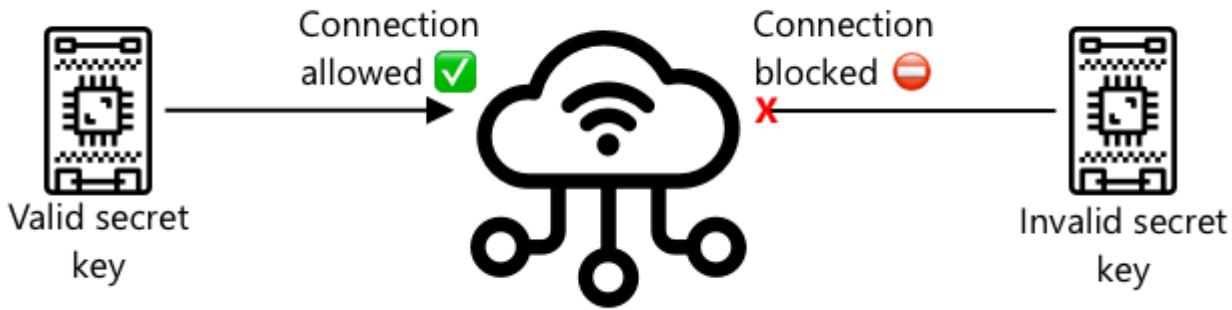
 Although you pay for these upsides with a monthly fee, most cloud providers offer a free version of their IoT service with a limited amount of messages per day or devices that can connect. This free version is usually more than enough for a developer to learn about the service. In this lesson you will be using a free version.

IoT devices connect to a cloud service either using a device SDK (a library that provides code to work with the features of the service), or directly via a communication protocol such as MQTT or HTTP. The device SDK is usually the easiest route to take as it handles everything for you, such as knowing what topics to publish or subscribe to, and how to handle security.



Devices connect to a service using a device SDK. Server code also connects to the service via an SDK. Microcontroller by Template / Cloud by Debi Alpa Nugraha / IoT by Adrien Coquet - all from the [Noun Project](#)

Your device then communicates with other parts of your application over this service - similar to how you sent telemetry and received commands over MQTT. This is usually using a service SDK or a similar library. Messages come from your device to the service where other components of your application can then read them, and messages can then be sent back to your device.



Devices without a valid secret key cannot connect to the IoT service. Microcontroller by Template / Cloud by Debi Alpa Nugraha / IoT by Adrien Coquet - all from the [Noun Project](#)

These services implement security by knowing about all the devices that can connect and send data, either by having the devices pre-registered with the service, or by giving the devices secret keys or certificates they can use to register themselves with the service the first time they connect. Unknown devices are unable to connect, if they try the service rejects the connection and ignores messages sent by them.

- ✓ Do some research: What is the downside of having an open IoT service where any device or code can connect? Can you find specific examples of hackers taking advantage of this?

Other components of your application can connect to the IoT service and learn about all the devices that are connected or registered, and communicate with them directly in bulk or individually.

💡 IoT services also implement additional capabilities, and the cloud providers have additional services and applications that can be connected to the service. For example, if you want to store all the telemetry messages sent by all the devices in a database, it's usually only a few clicks in the cloud provider's configuration tool to connect the service to a database and stream the data in.

Create an IoT service in the cloud

Now that you have an Azure subscription, you can sign up for an IoT service. The IoT service from Microsoft is called Azure IoT Hub.



Azure IoT Hub

The video below gives a short overview of Azure IoT Hub:



🎥 Click the image above to watch a video

- ✓ Take a moment to do some research and read the overview of IoT hub in the [Microsoft IoT Hub documentation](#).

The cloud services available in Azure can be configured through a web-based portal, or via a command-line interface (CLI). For this task, you will use the CLI.

Task - install the Azure CLI

To use the Azure CLI, first it must be installed on your PC or Mac.

1. Follow the instructions in the [Azure CLI documentation](#) to install the CLI.
2. The Azure CLI supports a number of extensions that add capabilities to manage a wide range of Azure services. Install the IoT extension by running the following command from your command

line or terminal:

sh

```
az extension add --name azure-iot
```

3. From your command line or terminal, run the following command to log in to your Azure subscription from the Azure CLI.

sh

```
az login
```

A web page will be launched in your default browser. Log in using the account you used to sign up for your Azure subscription. Once you are logged in, you can close the browser tab.

4. If you have multiple Azure subscriptions, such as a school provided one, and your own Azure for Students subscription, you will need to select the one you want to use. Run the following command to list all the subscriptions you have access to:

sh

```
az account list --output table
```

In the output, you will see the name of each subscription along with its `SubscriptionId`.

output

```
→ ~ az account list --output table
```

Name	CloudName	SubscriptionId
School-subscription	AzureCloud	cb30cde9-814a-42f0-a111-754cb788e4e
Azure for Students	AzureCloud	fa51c31b-162c-4599-add6-781def2e1fb

To select the subscription you want to use, use the following command:

sh

```
az account set --subscription <SubscriptionId>
```

Replace `<SubscriptionId>` with the Id of the subscription you want to use. After running this command, re-run the command to list your accounts. You will see the `IsDefault` column will be marked as `True` for the subscription you have just set.

Task - create a resource group

Azure services, such as IoT Hub instances, virtual machines, databases, or AI services, are referred to as **resources**. Every resource has to live inside a **Resource Group**, a logical grouping of one or more resources.

💡 Using resource groups means you can manage multiple services at once. For example, once you have finished all the lessons for this project you can delete the resource group, and all the resources in it will be deleted automatically.

1. There are multiple Azure data centers around the world, divided up into regions. When you create an Azure resource or resource group, you have to specify where you want it created. Run the following command to get the list of locations:

```
sh  
az account list-locations --output table
```

You will see a list of locations. This list will be long.

💡 At the time of writing, there are 65 locations you can deploy to.

```
output  
→ ~ az account list-locations --output table  


| DisplayName      | Name           | RegionalDisplayName   |
|------------------|----------------|-----------------------|
| East US          | eastus         | (US) East US          |
| East US 2        | eastus2        | (US) East US 2        |
| South Central US | southcentralus | (US) South Central US |
| ...              |                |                       |


```

Note down the value from the `Name` column of the region closest to you. You can find the regions on a map on the [Azure geographies page](#).

2. Run the following command to create a resource group called `soil-moisture-sensor`. Resource group names have to be unique in your subscription.

```
sh  
az group create --name soil-moisture-sensor \  
--location <location>
```

Replace <location> with the location you selected in the previous step.

Task - create an IoT Hub

You can now create an IoT Hub resource in your resource group.

1. Use the following command to create your IoT hub resource:

```
sh  
az iot hub create --resource-group soil-moisture-sensor \  
    --sku F1 \  
    --partition-count 2 \  
    --name <hub_name>
```

Replace <hub_name> with a name for your hub. This name needs to be globally unique - that is no other IoT Hub created by anyone can have the same name. This name is used in a URL that points to the hub, so needs to be unique. Use something like `soil-moisture-sensor-` and add a unique identifier on the end, like some random words or your name.

The `--sku F1` option tells it to use a free tier. The free tier supports 8,000 messages a day along with most of the features of the full-price tiers.

 Different pricing levels of Azure services are referred to as tiers. Each tier has a different cost and provides different features or data volumes.

 If you want to learn more about pricing, you can check out the [Azure IoT Hub pricing guide](#).

The `--partition-count 2` option defines how many streams of data the IoT Hub supports, more partitions reduce data blocking when multiple things read and write from the IoT Hub. Partitions are outside the scope of these lessons, but this value needs to be set to create a free tier IoT Hub.

 You can only have one free tier IoT Hub per subscription.

The IoT Hub will be created. It make take a minute or so for this to complete.

Communicate with IoT Hub

In the previous lesson, you used MQTT and sent messages back and forward on different topics, with the different topics having different purposes. Rather than send messages over different topics, IoT Hub has a number of defined ways for the device to communicate with the Hub, or for the Hub to communicate with the device.

 Under the hood this communication between IoT Hub and your device can use MQTT, HTTPS or AMQP.

- Device to cloud (D2C) messages - these are messages sent from a device to IoT Hub, such as telemetry. They can then be read off the IoT Hub by your application code.

 Under the hood, IoT Hub uses an Azure service called [Event Hubs](#). When you write code to read messages sent to the hub, these are often called events.

- Cloud to device (C2D) messages - these are messages sent from application code, via an IoT Hub to an IoT device
- Direct method requests - these are messages sent from application code via an IoT Hub to an IoT device to request that the device does something, such as control an actuator. These messages require a response so your application code can tell if it was successfully processed.
- Device twins - these are JSON documents kept synchronized between the device and IoT Hub, and are used to store settings or other properties either reported by the device, or should be set on the device (known as desired) by the IoT Hub.

IoT Hub can store messages and direct method requests for a configurable period of time (defaulting to one day), so if a device or application code loses connection, it can still retrieve messages sent whilst it was offline after it reconnects. Device twins are kept permanently in the IoT Hub, so at any time a device can reconnect and get the latest device twin.

 Do some research: Read more on these message types on the [Device-to-cloud communications guidance](#), and the [Cloud-to-device communications guidance](#) in the IoT Hub documentation.

Connect your device to the IoT service

Once the hub is created, your IoT device can connect to it. Only registered devices can connect to a service, so you will need to register your device first. When you register you can get back a connection string that the device can use to connect. This connection string is device specific, and contains information about the IoT Hub, the device, and a secret key that will allow this device to connect.

🎓 A connection string is a generic term for a piece of text that contains connection details. These are used when connecting to IoT Hubs, databases and many other services. They usually consist of an identifier for the service, such as a URL, and security information such as a secret key. These are passed to SDKs to connect to the service.

⚠ Connection strings should be kept secure! Security will be covered in more detail in a future lesson.

Task - register your IoT device

The IoT device can be registered with your IoT Hub using the Azure CLI.

1. Run the following command to register a device:

```
sh
az iot hub device-identity create --device-id soil-moisture-sensor \
--hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub.

This will create a device with an ID of `soil-moisture-sensor`.

2. When your IoT device connects to your IoT Hub using the SDK, it needs to use a connection string that gives the URL of the hub, along with a secret key. Run the following command to get the connection string:

```
sh
az iot hub device-identity connection-string show --device-id soil-moist \
--output table \
```

Replace <hub_name> with the name you used for your IoT Hub.

3. Store the connection string that is shown in the output as you will need it later.

Task - connect your IoT device to the cloud

Work through the relevant guide to connect your IoT device to the cloud:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi/Virtual IoT device](#)

Task - monitor events

For now, you won't be updating your server code. Instead you can use the Azure CLI to monitor events from your IoT device.

1. Make sure your IoT device is running and sending soil moisture telemetry values
2. Run the following command in your command prompt or terminal to monitor messages sent to your IoT Hub:

```
sh
az iot hub monitor-events --hub-name <hub_name>
```

Replace <hub_name> with the name you used for your IoT Hub.

You will see messages appear in the console output as they are sent by your IoT device.

```
output
Starting event monitor, use ctrl-c to stop...
{
  "event": {
    "origin": "soil-moisture-sensor",
    "module": "",
    "interface": "",
    "component": "",
    "payload": "{\"soil_moisture\": 376}"
  }
},
{
  "event": {
```

```
        "origin": "soil-moisture-sensor",
        "module": "",
        "interface": "",
        "component": "",
        "payload": "{\"soil_moisture\": 381}"
    }
}
```

The contents of the `payload` will match the message sent by your IoT device.

- These messages have a number of properties attached to them automatically, such as the timestamp they were sent. These are known as *annotations*. To view all the message annotations, use the following command:

```
sh
az iot hub monitor-events --properties anno --hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub.

You will see messages appear in the console output as they are sent by your IoT device.

```
output
Starting event monitor, use ctrl-c to stop...
{
    "event": {
        "origin": "soil-moisture-sensor",
        "module": "",
        "interface": "",
        "component": "",
        "properties": {},
        "annotations": {
            "iothub-connection-device-id": "soil-moisture-sensor",
            "iothub-connection-auth-method": "{\"scope\":\"device\", \"ty",
            "iothub-connection-auth-generation-id": "637553997165220462",
            "iothub-enqueuedtime": 1619976150288,
            "iothub-message-source": "Telemetry",
            "x-opt-sequence-number": 1379,
            "x-opt-offset": "550576",
            "x-opt-enqueued-time": 1619976150277
        },
        "payload": "{\"soil_moisture\": 381}"
    }
}
```

The time values in the annotations are in UNIX time, representing the number of seconds since midnight on 1st January 1970.

Task - control your IoT device

You can also use the Azure CLI to call direct methods on your IoT device.

1. Run the following command in your command prompt or terminal to invoke the `relay_on` method on the IoT device:

```
sh  
az iot hub invoke-device-method --device-id soil-moisture-sensor \  
        --method-name relay_on \  
        --method-payload '{}' \  
        --hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub.

This sends a direct method request for the method specified by `method-name`. Direct methods can take a payload containing data for the method, and this can be specified in the `method-payload` parameter as JSON.

You will see the relay turn on, and the corresponding output from your IoT device:

```
output  
Direct method received - relay_on
```

2. Repeat the above step, but set the `--method-name` to `relay_off`. You will see the relay turn off and the corresponding output from the IoT device.

Challenge

The free tier of IoT Hub allows 8,000 messages a day. The code you wrote sends telemetry messages every 10 seconds. How many messages a day is one message every 10 seconds?

Think about how often soil moisture measurements should be sent? How can you change your code to stay within the free tier and check as often as needed but not too often? What if you wanted to add a second device?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

The IoT Hub SDK is open source for both Arduino and Python. In the code repos on GitHub there are a number of samples showing how work with different IoT Hub features.

- If you are using a Wio Terminal, check out the [Arduino samples on GitHub](#)
- If you are using a Raspberry Pi or Virtual device, check out the [Python samples on GitHub](#)

Assignment

[Learn about cloud services](#)

Migrate your application logic to the cloud

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

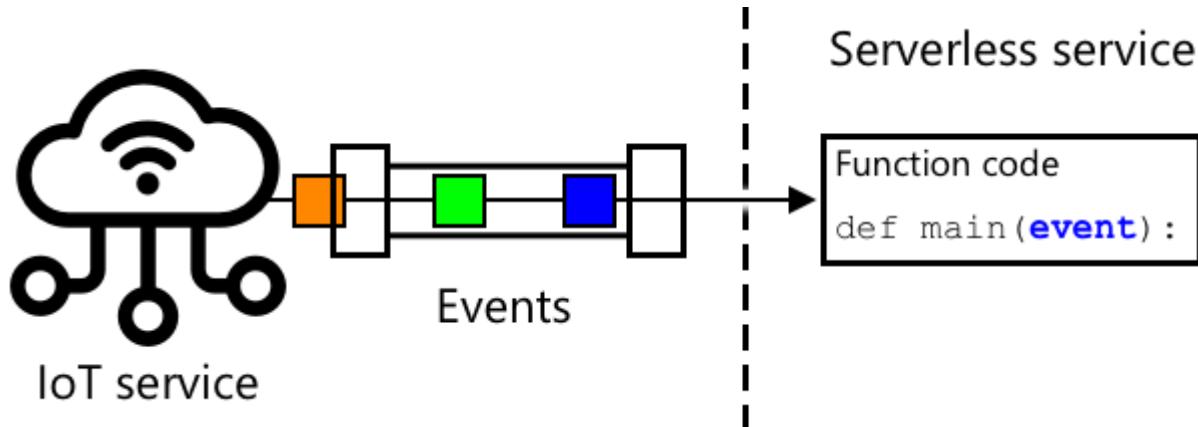
In the last lesson, you learned how to connect your plant soil moisture monitoring and relay control to a cloud-based IoT service. The next step is to move the server code that controls the timing of the relay to the cloud. In this lesson you will learn how to do this using serverless functions.

In this lesson we'll cover:

- [What is serverless?](#)
- [Create a serverless application](#)
- [Create an IoT Hub event trigger](#)
- [Send direct method requests from serverless code](#)
- [Deploy your serverless code to the cloud](#)

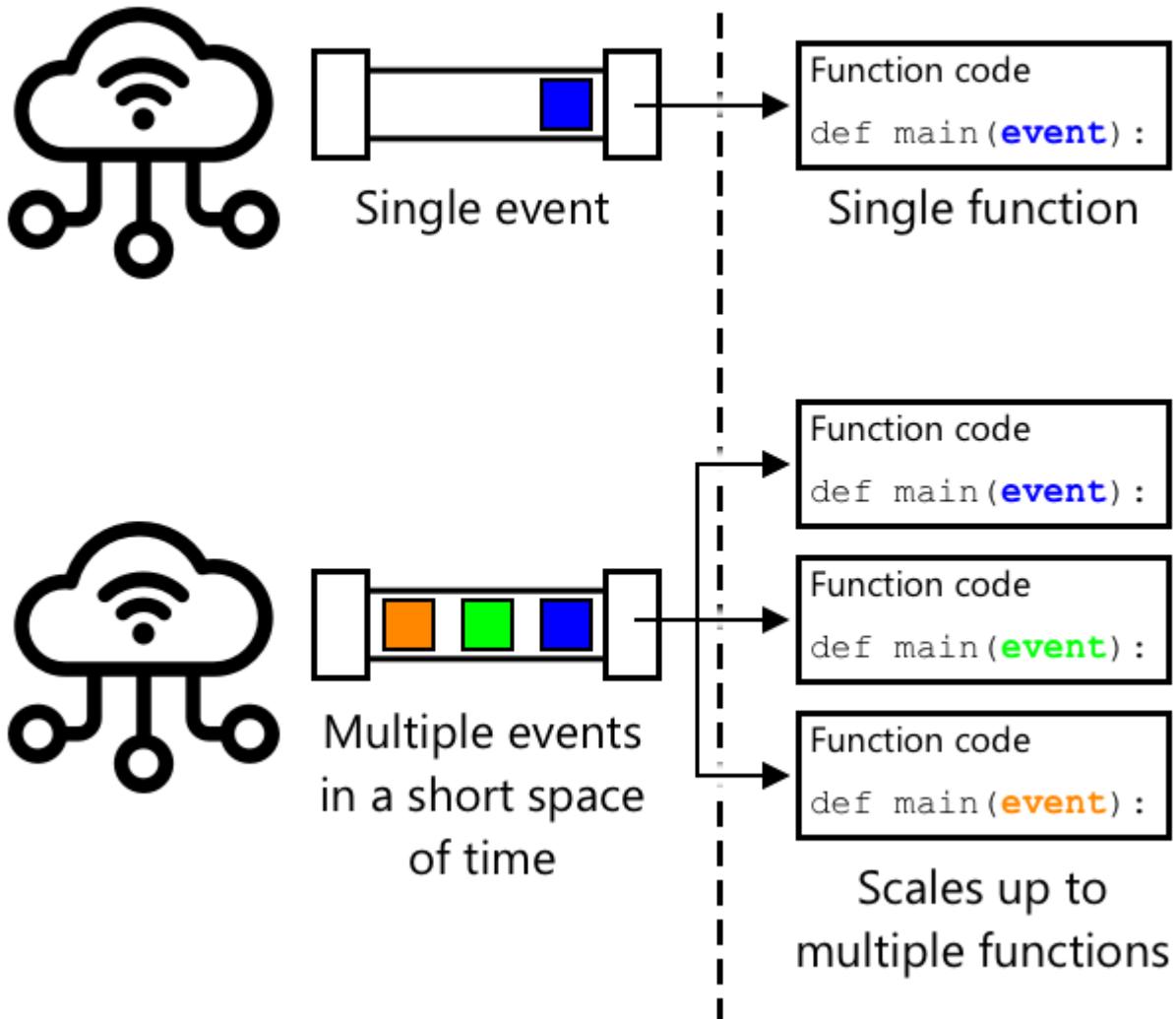
What is serverless?

Serverless, or serverless computing, involves creating small blocks of code that are run in the cloud in response to different kinds of events. When the event happens your code is run, and it is passed data about the event. These events can be from many different things, including web requests, messages put on a queue, changes to data in a database, or messages sent to an IoT service by IoT devices.



Events being sent from an IoT service to a serverless service, all being processed at the same time by multiple functions being run. IoT by Adrien Coquet from the [Noun Project](#)

 If you've used database triggers before, you can think of this as the same thing, code being triggered by an event such as inserting a row.



When many events are sent at the same time, the serverless service scales up to run them all at the same time. IoT by Adrien Coquet from the [Noun Project](#)

Your code is only run when the event happens, there is nothing keeping your code alive at other times. The event happens, your code is loaded and run. This makes serverless very scalable - if many events happen at the same time, the cloud provider can run your function as many times as you need at the same time across whatever servers they have available. The downside to this is if you need to share information between events, you need to save it somewhere like a database rather than storing it in memory.

Your code is written as a function that takes details about the event as a parameter. You can use a wide range of programming languages to write these serverless functions.

🎓 Serverless is also referred to as Functions as a service (FaaS) as each event trigger is implemented as a function in code.

Despite the name, serverless does actually use servers. The naming is because you as a developer don't care about the servers needed to run your code, all you care about is that your code is run in

response to an event. The cloud provider has a serverless *runtime* that manages allocating servers, networking, storage, CPU, memory and everything else needed to run your code. This model means you can't pay per server for the service, as there is no server. Instead you pay for the time your code is running, and the amount of memory used.

💰 Serverless is one of the cheapest ways to run code in the cloud. For example, at the time of writing, one cloud provider allows all of your serverless functions to execute a combined 1,000,000 times a month before they start charging you, and after that they charge US\$0.20 for each 1,000,000 executions. When your code is not running, you don't pay.

As an IoT developer, the serverless model is ideal. You can write a function that is called in response to messages sent from any IoT device that is connected to your cloud-hosted IoT service. Your code will handle all messages sent, but only be running when needed.

✓ Look back at the code you wrote as server code listening to messages over MQTT. How might this run in the cloud using serverless? How do you think the code might be changed to support serverless computing?

💡 The serverless model is moving to other cloud services in addition to running code. For example, serverless databases are available in the cloud using a serverless pricing model where you pay per request made against the database, such as a query or insert, usually using pricing based on how much work is done to service the request. For example a single select of one row against a primary key will cost less than a complicated operation joining many tables and returning thousands of rows.

Create a serverless application

The serverless computing service from Microsoft is called Azure Functions.



The short video below has an overview of Azure Functions



🎥 Click the image above to watch a video

✓ Take a moment to do some research and read the overview of Azure Functions in the [Microsoft Azure Functions documentation](#).

To write Azure Functions, you start with an Azure Functions app in the language of your choice. Out of the box Azure Functions supports Python, JavaScript, TypeScript, C#, F#, Java, and Powershell. In this lesson you will learn how to write an Azure Functions app in Python.

💡 Azure Functions also supports custom handlers so you can write your functions in any language that supports HTTP requests, including older languages such as COBOL.

Functions apps consist of one or more *triggers* - functions that respond to events. You can have multiple triggers inside one function app, all sharing common configuration. For example, in the configuration file for your Functions app you can have the connection details of your IoT Hub, and all the functions in the app can use this to connect and listen for events.

Task - install the Azure Functions tooling

One great feature of Azure Functions is that you can run them locally. The same runtime that is used in the cloud can be run on your computer, allowing you to write code that responds to IoT messages

and run it locally. You can even debug your code as events are handled. Once you are happy with your code, it can be deployed to the cloud.

The Azure Functions tooling is available as a CLI, known as the Azure Functions Core Tools.

1. Install the Azure Functions core tools by following the instructions on the [Azure Functions Core Tools documentation](#)
2. Install the Azure Functions extension for VS Code. This extension provides support for creating, debugging and deploying Azure functions. Refer to the [Azure Functions extension documentation](#) for instructions on installing this extension in VS Code.

When you deploy your Azure Functions app to the cloud, it needs to use a small amount of cloud storage to store things like the application files and log files. When you run your Functions app locally, you still need to connect to cloud storage, but instead of using actual cloud storage, you can use a storage emulator called [Azurite](#). This runs locally but acts like cloud storage.

 In Azure, the storage that Azure Functions uses is an Azure Storage Account. These accounts can store files, blobs, data in tables or data in queues. You can share one storage account between many apps, such as a Functions app and a web app.

1. Azurite is a Node.js app, so you will need to install Node.js. You can find the download and installation instructions on the [Node.js website](#). If you are using a Mac, you can also install it from [Homebrew](#).
2. Install Azurite using the following command (`npm` is a tool that is installed when you install Node.js):

```
sh  
npm install -g azurite
```

3. Create a folder called `azurite` for Azurite to use to store data:

```
sh  
mkdir azurite
```

4. Run Azurite, passing it this new folder:

```
sh  
azurite --location azurite
```

The Azurite storage emulator will launch and be ready for the local Functions runtime to connect.

```
output
→ ~ azurite --location azurite
Azurite Blob service is starting at http://127.0.0.1:10000
Azurite Blob service is successfully listening at http://127.0.0.1:10000
Azurite Queue service is starting at http://127.0.0.1:10001
Azurite Queue service is successfully listening at http://127.0.0.1:10001
Azurite Table service is starting at http://127.0.0.1:10002
Azurite Table service is successfully listening at http://127.0.0.1:10002
```

Task - create an Azure Functions project

The Azure Functions CLI can be used to create a new Functions app.

1. Create a folder for your Functions app and navigate to it. Call it `soil-moisture-trigger`

```
sh
mkdir soil-moisture-trigger
cd soil-moisture-trigger
```

2. Create a Python virtual environment inside this folder:

```
sh
python3 -m venv .venv
```

3. Activate the virtual environment:

- On Windows run:

```
cmd
.venv\Scripts\activate.bat
```

- On macOS or Linux, run:

```
cmd
source ./venv/bin/activate
```

4. Run the following command to create a Functions app in this folder:

```
sh
func init --worker-runtime python soil-moisture-trigger
```

This will create three files inside the current folder:

- `host.json` - this JSON document contains settings for your Functions app. You won't need to modify these settings.
- `local.settings.json` - this JSON document contains settings your app would use when running locally, such as connection strings for your IoT Hub. These settings are local only, and should not be added to source code control. When you deploy the app to the cloud, these settings are not deployed, instead your settings are loaded from application settings. This will be covered later in this lesson.
- `requirements.txt` - this is a [Pip requirements file](#) that contains the Pip packages needed to run your Functions app.

5. The `local.settings.json` file has a setting for the storage account that the Functions app will use. This defaults to an empty setting, so needs to be set. To connect to the Azurite local storage emulator, set this value to the following:

```
json  
"AzureWebJobsStorage": "UseDevelopmentStorage=true",
```

6. Install the necessary Pip packages using the requirements file:

```
sh  
pip install -r requirements.txt
```

 The required Pip packages need to be in this file, so that when the Functions app is deployed to the cloud, the runtime can ensure it installs the correct packages.

7. To test everything is working correctly, you can start the Functions runtime. Run the following command to do this:

```
sh  
func start
```

You will see the runtime start up and report that it hasn't found any job functions (triggers).

```
output  
(.venv) → soil-moisture-trigger func start  
Found Python version 3.9.1 (python3).
```

Core Tools Version: 3.0.3442 Commit hash: 6bfab24b2743f8421475d996
Function Runtime Version: 3.0.15417.0

[2021-05-05T01:24:46.795Z] No job functions found.

⚠ If you get a firewall notification, grant access as the `func` application needs to be able to read and write to your network.

⚠ If you are using macOS, there may be warnings in the output:

```
output
(.venv) → soil-moisture-trigger func start
Found Python version 3.9.1 (python3).

Azure Functions Core Tools
Core Tools Version: 3.0.3442 Commit hash: 6bfab24b2743f84214
Function Runtime Version: 3.0.15417.0
```

[2021-06-16T08:18:28.315Z] Cannot create directory for shared memo
[2021-06-16T08:18:28.316Z] System.IO.FileSystem: Access to the pat
[2021-06-16T08:18:30.361Z] No job functions found.

You can ignore these as long as the Functions app starts correctly and lists the running functions. As mentioned in [this question on the Microsoft Docs Q&A](#) it can be ignored.

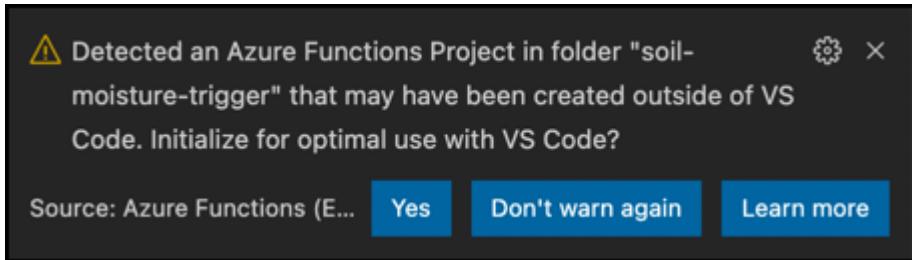
8. Stop the Functions app by pressing `ctrl+c`.

9. Open the current folder in VS Code, either by opening VS Code, then opening this folder, or by running the following:

```
sh
code .
```

VS Code will detect your Functions project and show a notification saying:

```
output
Detected an Azure Functions Project in folder "soil-moisture-trigger" th
VS Code. Initialize for optimal use with VS Code?
```



Select **Yes** from this notification.

10. Make sure the Python virtual environment is running in the VS Code terminal. Terminate it and restart it if necessary.

Create an IoT Hub event trigger

The Functions app is the shell of your serverless code. To respond to IoT hub events, you can add an IoT Hub trigger to this app. This trigger needs to connect to the stream of messages that are sent to the IoT Hub and respond to them. To get this stream of messages, your trigger needs to connect to the IoT Hubs *event hub compatible endpoint*.

IoT Hub is based upon another Azure service called Azure Event Hubs. Event Hubs is a service that allows you to send and receive messages, IoT Hub extends this to add features for IoT devices. The way you connect to read messages off the IoT Hub is the same as you would if you were using Event Hubs.

- ✓ Do some research: Read the overview of Event Hubs in the [Azure Event Hubs documentation](#). How do the basic features compare to IoT Hub?

For an IoT device to connect to the IoT Hub, it has to use a secret key that ensures only allowed devices can connect. The same applies when connecting to read off messages, your code will need a connection string that contains a secret key, along with details of the IoT Hub.

 The default connection string you get has **iothubowner** permissions, which gives any code that uses it full permissions on the IoT Hub. Ideally you should connect with the lowest level of permissions needed. This will be covered in the next lesson.

Once your trigger has connected, the code inside the function will be called for every message sent to the IoT Hub, regardless of which device sent it. The trigger will be passed the message as a parameter.

Task - get the Event Hub compatible endpoint connection string

1. From the VS Code terminal run the following command to get the connection string for the IoT Hubs Event Hub compatible endpoint:

```
sh  
az iot hub connection-string show --default-eventhub \  
--output table \  
--hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub.

2. In VS Code, open the `local.settings.json` file. Add the following additional value inside the `Values` section:

```
json  
"IOT_HUB_CONNECTION_STRING": "<connection string>"
```

Replace `<connection string>` with the value from the previous step. You will need to add a comma after the line above to make this valid JSON.

Task - create an event trigger

You are now ready to create the event trigger.

1. From the VS Code terminal run the following command from inside the `soil-moisture-trigger` folder:

```
sh  
func new --name iot-hub-trigger --template "Azure Event Hub trigger"
```

This creates a new Function called `iot-hub-trigger`. The trigger will connect to the Event Hub compatible endpoint on the IoT Hub, so you can use an event hub trigger. There is no specific IoT Hub trigger.

This will create a folder inside the `soil-moisture-trigger` folder called `iot-hub-trigger` that contains this function. This folder will have the following files inside it:

- `__init__.py` - this is the Python code file that contains the trigger, using the standard Python file name convention to turn this folder into a Python module.

This file will contain the following code:

```
from typing import List
import logging
import azure.functions as func

def main(events: List[func.EventHubEvent]):
    for event in events:
        logging.info('Python EventHub trigger processed an event: %s',
                     event.get_body().decode('utf-8'))
```

The core of the trigger is the `main` function. It is this function that is called with the events from the IoT Hub. This function has a parameter called `events` that contains a list of `EventHubEvent`. Each event in this list is a message sent to IoT Hub, along with properties that are the same as the annotations you saw in the last lesson.

This trigger processes a list of events, rather than individual events. When you first run the trigger it will process any unprocessed events on the IoT Hub (remember that messages are stored for a while so they are not lost if your application code is offline). After this it will generally process a list containing only one event, unless a lot of events are sent to the Hub in a short space of time.

The core of this function loops through the list and logs the events.

- `function.json` - this contains configuration for the trigger. The main configuration is in a section called `bindings`. A binding is the term for a connection between Azure Functions and other Azure services. This function has an input binding to an event hub - it connects to an event hub and receives data.



You can also have output bindings so that the output of a function is sent to another service. For example you could add an output binding to a database and return the IoT Hub event from the function, and it will automatically be inserted into the database.

- Do some research: Read up on bindings in the [Azure Functions triggers and bindings concepts documentation](#).

The `bindings` section includes configuration for the binding. The values of interest are:

- `"type": "eventHubTrigger"` - this tells the function it needs to listen to events from an Event Hub
- `"name": "events"` - this is the parameter name to use for the Event Hub events. This matches the parameter name in the `main` function in the Python code.

- "direction": "in", - this is an input binding, the data from the event hub comes into the function
- "connection": "" - this defines the name of the setting to read the connection string from. When running locally, this will read this setting from the `local.settings.json` file.

 The connection string cannot be stored in the `function.json` file, it has to be read from the settings. This is to stop you accidentally exposing your connection string.

1. Update the value of "connection" in the `function.json` file to point to the new value you added to the `local.settings.json` file:

```
json  
"connection": "IOT_HUB_CONNECTION_STRING",
```

 Remember - this needs to point to the setting, not contain the actual connection string.

Task - run the event trigger

1. To run the Functions app, run the following command from the VS Code terminal

```
sh  
func start
```

The Functions app will start up, and will discover the `iot-hub-trigger` function. It will then process any events that have already been sent to the IoT Hub in the past day.

```
output  
.venv) → soil-moisture-trigger func start  
Found Python version 3.9.1 (python3).
```

Azure Functions Core Tools

Core Tools Version: 3.0.3442 Commit hash: 6bfab24b2743f8421475d996
Function Runtime Version: 3.0.15417.0

Functions:

```
iot-hub-trigger: eventHubTrigger
```

For detailed output, run func with --verbose flag.

```
[2021-05-05T02:44:07.517Z] Worker process started and initialized.
```

```
[2021-05-05T02:44:09.202Z] Executing 'Functions.iot-hub-trigger' (Reason
```

```
[2021-05-05T02:44:09.205Z] Trigger Details: PartitionId: 0, Offset: 101124
```

```
[2021-05-05T02:44:09.352Z] Python EventHub trigger processed an event: {
```

```
[2021-05-05T02:44:09.354Z] Python EventHub trigger processed an event: {
```

```
[2021-05-05T02:44:09.395Z] Executed 'Functions.iot-hub-trigger' (Succeed
```

Each call to the function will be surrounded by a

Executing 'Functions.iot-hub-trigger' / Executed 'Functions.iot-hub-trigger' block in the output, so you can know how many messages were processed in each function call.

If you get the following error:

```
          output  
The listener for function 'Functions.iot-hub-trigger' was unable to star
```

Then check Azurite is running and you have set the `AzureWebJobsStorage` in the `local.settings.json` file to `UseDevelopmentStorage=true`.

2. Make sure your IoT device is running, You will see new soil moisture messages appearing in the Functions app.
3. Stop and restart the Functions app. You will see that it won't process messages previous messages again, it will only process new messages.

 VS Code also supports debugging your Functions. You can set break points by clicking on the border by the start of each line of code, or putting the cursor on a line of code and selecting Run -> Toggle breakpoint, or pressing `F9`. You can launch the debugger by selecting Run -> Start debugging, pressing `F5`, or selecting the Run and debug pane and selecting the **Start debugging** button. By doing this you can see the details of the events being processed.

Send direct method requests from serverless code

So far your Functions app is listening to messages from the IoT Hub using the Event Hub compatible end point. You now need to send commands to the IoT device. This is done by using a different connection to the IoT Hub via the *Registry Manager*. The Registry Manager is a tool that allows you to see what devices are registered with the IoT Hub, and communicate with those devices by sending cloud to device messages, direct method requests or updating the device twin. You can also use it to register, update or delete IoT devices from the IoT Hub.

To connect to the Registry Manager, you need a connection string.

Task - get the Registry Manager connection string

1. To get the connection string, run the following command:

```
sh  
az iot hub connection-string show --policy-name service \  
--output table \  
--hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub.

The connection string is requested for the *ServiceConnect* policy using the `--policy-name service` parameter. When you request a connection string, you can specify what permissions that connection string will allow. The *ServiceConnect* policy allows your code to connect and send messages to IoT devices.

- Do some research: Read up on the different policies in the [IoT Hub permissions documentation](#)

2. In VS Code, open the `local.settings.json` file. Add the following additional value inside the `Values` section:

```
json  
"REGISTRY_MANAGER_CONNECTION_STRING": "<connection string>"
```

Replace `<connection string>` with the value from the previous step. You will need to add a comma after the line above to make this valid JSON.

Task - send a direct method request to a device

1. The SDK for the Registry Manager is available via a Pip package. Add the following line to the `requirements.txt` file to add the dependency on this package:

```
sh  
azure-iot-hub
```

2. Make sure the VS Code terminal has the virtual environment activated, and run the following command to install the Pip packages:

```
sh  
pip install -r requirements.txt
```

3. Add the following imports to the `__init__.py` file:

```
python  
import json  
import os  
from azure.iot.hub import IoTHubRegistryManager  
from azure.iot.hub.models import CloudToDeviceMethod
```

This imports some system libraries, as well as the libraries to interact with the Registry Manager and send direct method requests.

4. Remove the code from inside the `main` method, but keep the method itself.

5. When multiple messages are received, it only makes sense to process the last one as this is the current soil moisture. It makes no sense to process messages from before. Add the following code to get the last message from the `events` parameter:

```
python  
event = events[-1]
```

6. Below this, add the following code:

```
python  
body = json.loads(event.get_body().decode('utf-8'))  
device_id = event.iothub_metadata['connection-device-id']  
  
logging.info(f'Received message: {body} from {device_id}')
```

This code extracts the body of the event which contains the JSON message sent by the IoT device.

It then gets the device ID from the annotations passed with the message. The body of the event contains the message sent as telemetry, the `iothub_metadata` dictionary contains properties set by the IoT Hub such as the device ID of the sender, and the time that the message was sent.

This information is then logged. You will see this logging in the terminal when you run the Function app locally.

7. Below this, add the following code:

```
python  
soil_moisture = body['soil_moisture']  
  
if soil_moisture > 450:  
    direct_method = CloudToDeviceMethod(method_name='relay_on', payload=  
else:  
    direct_method = CloudToDeviceMethod(method_name='relay_off', payload=
```

This code gets the soil moisture from the message. It then checks the soil moisture, and depending on the value, creates a helper class for the direct method request for the `relay_on` or `relay_off` direct method. The method request doesn't need a payload, so an empty JSON document is sent.

8. Below this add the following code:

```
python  
logging.info(f'Sending direct method request for {direct_method.method_n  
registry_manager_connection_string = os.environ['REGISTRY_MANAGER_CONNEC  
registry_manager = IoTHubRegistryManager(registry_manager_connection_str
```

This code loads the `REGISTRY_MANAGER_CONNECTION_STRING` from the `local.settings.json` file. The values in this file are made available as environment variables, and these can be read using the `os.environ` function, a function that returns a dictionary of all the environment variables.



When this code is deployed to the cloud, the values in the `local.settings.json` file will be set as Application Settings, and these can be read from environment variables.

The code then creates an instance of the Registry Manager helper class using the connection string.

9. Below this add the following code:

```
python  
registry_manager.invoke_device_method(device_id, direct_method)  
  
logging.info('Direct method request sent!')
```

This code tells the registry manager to send the direct method request to the device that sent the telemetry.

 In the versions of the app you created in earlier lessons using MQTT, the relay control commands were sent to all devices. The code assumed you would only have one device. This version of the code sends the method request to a single device, so would work if you had multiple setups of moisture sensors and relays, sending the right direct method request to the right device.

10. Run the Functions app, and make sure your IoT device is sending data. You will see the messages being processed and the direct method requests being sent. Move the soil moisture sensor in and out of the soil to see the values change and the relay turn on and off

 You can find this code in the [code/functions](#) folder.

Deploy your serverless code to the cloud

Your code is now working locally, so the next step is to deploy the Functions App to the cloud.

Task - create the cloud resources

Your Functions app needs to be deployed to a Functions App resource in Azure, living inside the Resource Group you created for your IoT Hub. You will also need a Storage Account created in Azure to replace the emulated one you have running locally.

1. Run the following command to create a storage account:

```
az storage account create --resource-group soil-moisture-sensor \
    --sku Standard_LRS \
    --name <storage_name>
```

Replace `<storage_name>` with a name for your storage account. This will need to be globally unique as it forms part of the URL used to access the storage account. You can only use lower case letters and numbers for this name, no other characters, and it's limited to 24 characters. Use something like `sms` and add a unique identifier on the end, like some random words or your name.

The `--sku Standard_LRS` selects the pricing tier, selecting the lowest cost general-purpose account. There isn't a free tier of storage, and you pay for what you use. The costs are relatively low, with the most expensive storage at less than US\$0.05 per month per gigabyte stored.

 Read up on pricing on the [Azure Storage Account pricing page](#)

2. Run the following command to create a Function App:

```
az functionapp create --resource-group soil-moisture-sensor \
    --runtime python \
    --functions-version 3 \
    --os-type Linux \
    --consumption-plan-location <location> \
    --storage-account <storage_name> \
    --name <functions_app_name>
```

Replace `<location>` with the location you used when creating the Resource Group in the previous lesson.

Replace `<storage_name>` with the name of the storage account you created in the previous step.

Replace `<functions_app_name>` with a unique name for your Functions App. This will need to be globally unique as it forms part of a URL that can be used to access the Functions App. Use something like `soil-moisture-sensor-` and add a unique identifier on the end, like some random words or your name.

The `--functions-version 3` option sets the version of Azure Functions to use. Version 3 is the latest version.

The `--os-type Linux` tells the Functions runtime to use Linux as the OS to host these functions. Functions can be hosted on Linux or Windows, depending on the programming

Task - upload your application settings

When you developed your Functions App, you stored some settings in the `local.settings.json` file for the connection strings for your IoT Hub. These need to be written to Application Settings in your Function App in Azure so that they can be used by your code.

 The `local.settings.json` file is for local development settings only, and these should not be checked into source code control, such as GitHub. When deployed to the cloud, Application Settings are used. Application Settings are key/value pairs hosted in the cloud and are read from environment variables either in your code or by the runtime when connecting your code to IoT Hub.

1. Run the following command to set the `IOT_HUB_CONNECTION_STRING` setting in the Functions App Application Settings:

```
sh  
az functionapp config appsettings set --resource-group soil-moisture-ser  
--name <functions_app_name> \  
--settings "IOT_HUB_CONNECTION_STR
```

Replace `<functions_app_name>` with the name you used for your Functions App.

Replace `<connection string>` with the value of `IOT_HUB_CONNECTION_STRING` from your `local.settings.json` file.

2. Repeat the step above, but set the value of `REGISTRY_MANAGER_CONNECTION_STRING` to the corresponding value from your `local.settings.json` file.

When you run these commands, they will also output a list of all the Application Settings for the function app. You can use this to check that your values are set correctly.

 You will see a value already set for `AzureWebJobsStorage`. In your `local.settings.json` file, this was set to a value to use the local storage emulator. When you created the Functions App, you pass the storage account as a parameter, and this gets set automatically in this setting.

Task - deploy your Functions App to the cloud

Now that the Functions App is ready, your code can be deployed.

1. Run the following command from the VS Code terminal to publish your Functions App:

```
sh  
func azure functionapp publish <functions_app_name>
```

Replace `<functions_app_name>` with the name you used for your Functions App.

The code will be packaged up and sent to the Functions App, where it will be deployed and started. There will be a lot of console output, ending in confirmation of the deployment and a list of the functions deployed. In this case the list will only contain the trigger.

output

```
Deployment successful.  
Remote build succeeded!  
Syncing triggers...  
Functions in soil-moisture-sensor:  
    iot-hub-trigger - [eventHubTrigger]
```

Make sure your IoT device is running. Change the moisture levels by adjusting the soil moisture, or moving the sensor in and out of the soil. You will see the relay turn on and off as the soil moisture changes.

Challenge

In the previous lesson, you managed timing for the relay by unsubscribing from MQTT messages whilst the relay was on, and for a short while after it was turned off. You can't use this method here - you cannot unsubscribe your IoT Hub trigger.

Think about different ways you could handle this in your Functions App.

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read up on serverless computing on the [Serverless Computing page on Wikipedia](#)
- Read about using serverless in Azure including some more examples on the [Go serverless for your IoT needs Azure blog post](#)
- Learn more about Azure Functions on the [Azure Functions YouTube channel](#)

Assignment

[Add manual relay control](#)

Keep your plant secure

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last few lessons you've created a soil monitoring IoT device and connected it to the cloud. But what if hackers working for a rival farmer managed to seize control of your IoT devices? What if they sent high soil moisture readings so your plants never got watered, or turned on your watering system to run all the time killing your plants from over-watering and costing you a small fortune in water?

In this lesson you will learn about securing IoT devices. As this is the last lesson for this project, you will also learn how to clean up your cloud resources, reducing any potential costs.

In this lesson we'll cover:

- [Why do you need to secure IoT devices?](#)
- [Cryptography](#)

- [Secure your IoT devices](#)
- [Generate and use an X.509 certificate](#)

 This is the last lesson in this project, so after completing this lesson and the assignment, don't forget to clean up your cloud services. You will need the services to complete the assignment, so make sure to complete that first.

Refer to [the clean up your project guide](#) if necessary for instructions on how to do this.

Why do you need to secure IoT devices?

IoT security involves ensuring that only expected devices can connect to your cloud IoT service and send them telemetry, and only your cloud service can send commands to your devices. IoT data can also be personal, including medical or intimate data, so your entire application needs to consider security to stop this data being leaked.

If your IoT application is not secure, there are a number of risks:

- A fake device could send incorrect data, causing your application to respond incorrectly. For example, they could send constant high soil moisture readings, meaning your irrigation system never turns on and your plants die from lack of water
- Unauthorized users could read data from IoT devices including personal or business critical data
- Hackers could send commands to control a device in a way that could cause damage to the device or connected hardware
- By connecting to an IoT device, hackers can use this to access additional networks to get access to private systems
- Malicious users could access personal data and use this for blackmail

These are real world scenarios, and happen all the time. Some examples were given in earlier lessons, but here are some more:

- In 2018, hackers used an open WiFi access point on a fish tank thermostat to gain access to a casino's network to steal data. [The Hacker News - Casino Gets Hacked Through Its Internet-Connected Fish Tank Thermometer](#)
- In 2016, the Mirai Botnet launched a denial of service attack against Dyn, an Internet service provider, taking down large portions of the Internet. This botnet used malware to connect to IoT devices such as DVRs and cameras that used default usernames and passwords, and from there launched the attack. [The Guardian - DDoS attack that disrupted internet was largest of its kind in history, experts say](#)

- Spiral Toys had a database of users of their CloudPets connected toys publicly available over the Internet. [Troy Hunt - Data from connected CloudPets teddy bears leaked and ransomed, exposing kids' voice messages.](#)
- Strava tagged runners that you ran past and showed their routes, allowing strangers to effectively see where you live. [Kim Komndo - Fitness app could lead a stranger right to your home — change this setting.](#)

Do some research: Search for more examples IoT Hacks and breaches of IoT data, especially with personal items such as Internet connected toothbrushes or scales. Think about the impact these hacks could have on the victims or customers.

 Security is a massive topic, and this lesson will only touch on some of the basics around connecting your device to the cloud. Other topics that won't be covered include monitoring for data changes in transit, hacking devices directly, or changes to device configurations. IoT hacking is such a threat, tools like [Azure Defender for IoT](#) have been developed. These tools are similar to the anti-virus and security tools you might have on your computer, just designed for small, low powered IoT devices.

Cryptography

When a device connects to an IoT service, it uses an ID to identify itself. The problem is this ID can be cloned - a hacker could set up a malicious device that uses the same ID as a real device but sends bogus data.



Both valid and malicious devices could use the same ID to send telemetry. Microcontroller by Template / IoT by Adrien Coquet - all from the [Noun Project](#)

The way round this is to convert the data being sent into a scrambled format, using some kind of value to scramble the data known to the device and the cloud only. This process is called *encryption*, and the value used to encrypt the data is called an *encryption key*.



If encryption is used, then only encrypted messages will be accepted, others will be rejected.

Microcontroller by Template / IoT by Adrien Coquet - all from the [Noun Project](#)

The cloud service can then convert the data back to a readable format, using a process called *decryption*, using either the same encryption key, or a *decryption key*. If the encrypted message cannot be decrypted by the key, the device has been hacked and the message is rejected.

The technique for doing encryption and decryption is called *cryptography*.

Early cryptography

The earliest types of cryptography were substitution ciphers, dating back 3,500 years. Substitution ciphers involve substituting one letter for another. For example, the [Caesar cipher](#) involves shifting the alphabet by a defined amount, with only the sender of the encrypted message, and the intended recipient knowing how many letters to shift.

The [Vigenère cipher](#) took this further by using words to encrypt text, so that each letter in the original text was shifted by a different amount, rather than always shifting by the same number of letters.

Cryptography was used for a wide range of purposes, such as protecting a potters glaze recipe in ancient Mesopotamia, writing secret love notes in India, or keeping ancient Egyptian magical spells secret.

Modern cryptography

Modern cryptography is much more advanced, making it harder to crack than early methods. Modern cryptography uses complicated mathematics to encrypt data with far too many possible keys to make brute force attacks possible.

Cryptography is used in a lot of different ways for secure communications. If you are reading this page on GitHub, you may notice the web site address starts with *HTTPS*, meaning that the communication between your browser and the web servers of GitHub is encrypted. If someone was able to read the internet traffic flowing between your browser and GitHub, they wouldn't be able to read the data as it is encrypted. Your computer might even encrypt all the data on your hard drive so if someone steals it, they won't be able to read any of your data without your password.

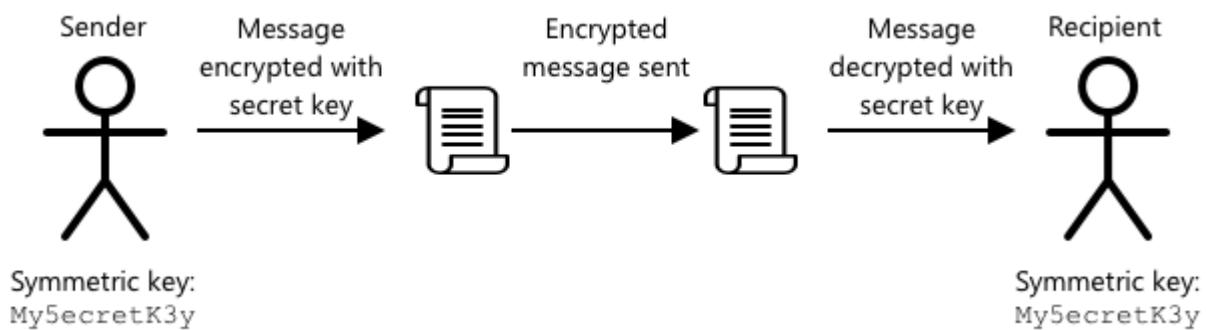
Unfortunately, not everything is secure. Some devices have no security, others are secured using easy to crack keys, or sometimes even all the devices of the same type using the same key. There have been accounts of very personal IoT devices that all have the same password to connect to them over WiFi or Bluetooth. If you can connect to your own device, you can connect to someone else's. Once connected you could access some very private data, or have control over their device.

💡 Despite the complexities of modern cryptography and the claims that breaking encryption can take billions of years, the rise of quantum computing has led to the possibility of breaking all known encryption in a very short space of time!

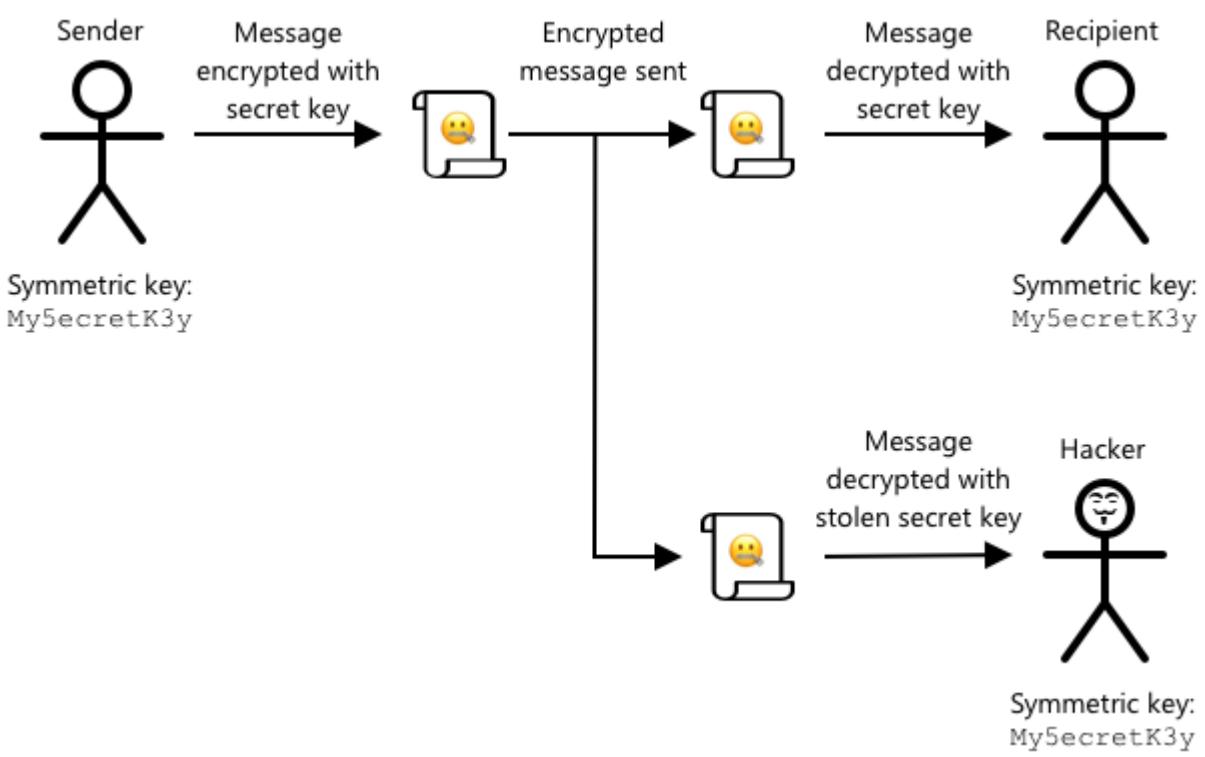
Symmetric and asymmetric keys

Encryption comes in two types - symmetric and asymmetric.

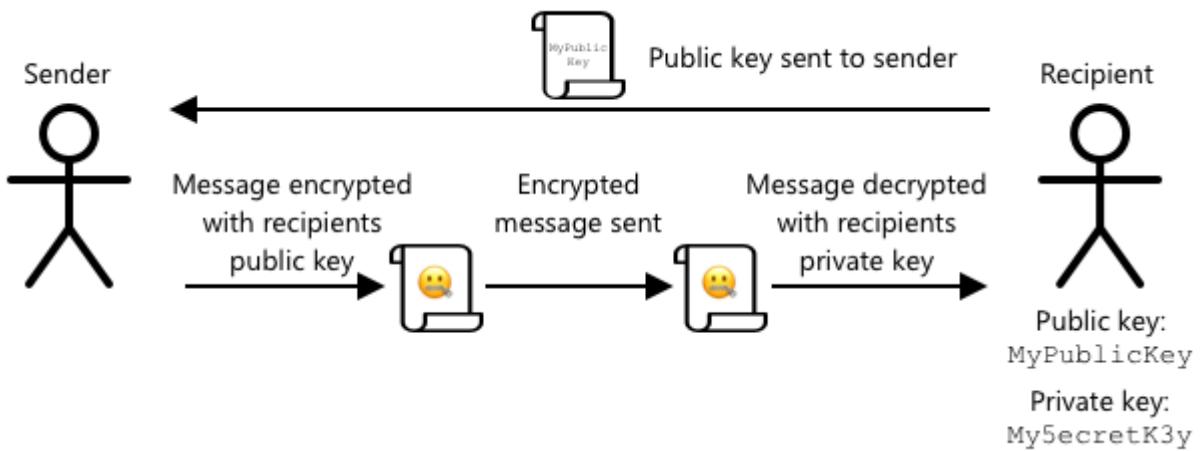
Symmetric encryption uses the same key to encrypt and decrypt the data. Both the sender and receiver need to know the same key. This is the least secure type, as the key needs to be shared somehow. For a sender to send an encrypted message to a recipient, the sender first might have to send the recipient the key.



If the key gets stolen in transit, or the sender or recipient get hacked and the key is found, the encryption can be cracked.



Asymmetric encryption uses 2 keys - an encryption key and a decryption key, referred to as a public/private key pair. The public key is used to encrypt the message, but cannot be used to decrypt it, the private key is used to decrypt the message but cannot be used to encrypt it.



The recipient shares their public key, and the sender uses this to encrypt the message. Once the message is sent, the recipient decrypts it with their private key. Asymmetric encryption is more secure as the private key is kept private by the recipient and never shared. Anyone can have the public key as it can only be used to encrypt messages.

Symmetric encryption is faster than asymmetric encryption, asymmetric is more secure. Some systems will use both - using asymmetric encryption to encrypt and share the symmetric key, then using the symmetric key to encrypt all data. This makes it more secure to share the symmetric key between sender and recipient, and faster when encrypting and decrypting data.

Secure your IoT devices

IoT devices can be secured using symmetric or asymmetric encryption. Symmetric is easier, but less secure.

Symmetric keys

When you set up your IoT device to interact with IoT Hub, you used a connection string. An example connection string is:

```
output  
HostName=soil-moisture-sensor.azure-devices.net;DeviceId=soil-moisture-sen:
```

This connection string is made of three parts separated by semi-colons, with each part a key and a value:

Key	Value	Description
HostName	soil-moisture-sensor.azure-devices.net	The URL of the IoT Hub
DeviceId	soil-moisture-sensor	The unique ID of the device
SharedAccessKey	Bhry+ind7kKEIDxubK61RiEHHRTrPl7HUow8cEm/mU0=	A symmetric key known by the device and the IoT Hub

The last part of this connection string, the `SharedAccessKey`, is the symmetric key known by both the device and the IoT Hub. This key is never sent from the device to the cloud, or the cloud to the device. Instead it is used to encrypt data that is sent or received.

 Do an experiment. What do you think will happen if you change the `SharedAccessKey` part of the connection string when connecting your IoT device? Try it out.

When the device first tries to connect it sends a shared access signature (SAS) token consisting of the URL of the IoT Hub, a timestamp that the access signature will expire (usually 1 day from the current time), and a signature. This signature consists of the URL and the expiry time encrypted with the shared access key from the connection string.

The IoT Hub decrypts this signature with the shared access key, and if the decrypted value matches the URL and expiry, the device is allowed to connect. It also verifies that the current time is before the expiry, to stop a malicious device capturing the SAS token of a real device and using it.

This is an elegant way to verify that the sender is the correct device. By sending some known data in both a decrypted and encrypted form, the server can verify the device by ensuring when it decrypts the encrypted data, the result matches the decrypted version that was sent. If it matches, then both the sender and recipient have the same symmetric encryption key.

 Because of the expiry time, your IoT device needs to know the accurate time, usually read from an [NTP](#) server. If the time is not accurate, the connection will fail.

After the connection, all data sent to the IoT Hub from the device, or to the device from the IoT Hub will be encrypted with the shared access key.

 What do you think will happen if multiple devices share the same connection string?

 It is bad security practice to store this key in code. If a hacker gets your source code, they can get your key. It is also harder when releasing code as you would need to recompile with an updated key for every device. It is better to load this key from a hardware security module - a chip on the IoT device that stores encrypted values that can be read by your code.

When learning IoT it is often easier to put the key in code, as you did in an earlier lesson, but you must ensure this key is not checked into public source code control.

X.509 certificates

When you are using asymmetric encryption with a public/private key pair, you need to provide your public key to anyone who wants to send you data. The problem is, how can the recipient of your key be sure it's actually your public key, not someone else pretending to be you? Instead of providing a key, you can instead provide your public key inside a certificate that has been verified by a trusted third party, called an X.509 certificate.

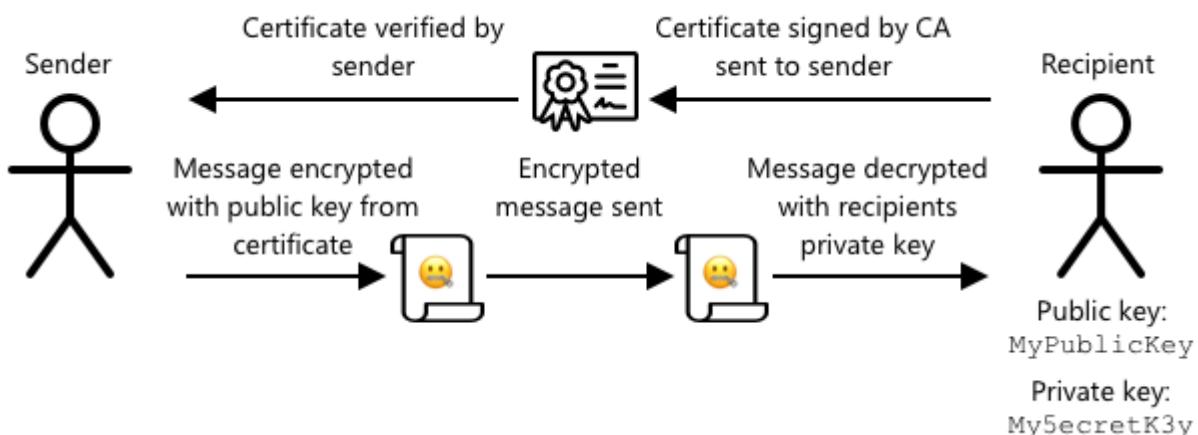
X.509 certificates are digital documents that contain the public key part of the public/private key pair. They are usually issued by one of a number of trusted organizations called [Certification authorities](#) (CAs), and digitally signed by the CA to indicate the key is valid and comes from you. You trust the certificate and that the public key is from who the certificate says it is from, because you trust the CA, similar to how you would trust a passport or driving license because you trust the country issuing it. Certificates cost money, so you can also 'self-sign', that is create a certificate yourself that is signed by you, for testing purposes.

 You should never use a self-signed certificate for a production release.

These certificates have a number of fields in them, including who the public key is from, the details of the CA who issued it, how long it is valid for, and the public key itself. Before using a certificate, it is good practice to verify it by checking that it was signed by the original CA.

- You can read a full list of the fields in the certificate in the [Microsoft Understanding X.509 Public Key Certificates tutorial](#)

When using X.509 certificates, both the sender and the recipient will have their own public and private keys, as well as both having X.509 certificates that contain the public key. They then exchange X.509 certificates somehow, using each other's public keys to encrypt the data they send, and their own private key to decrypt the data they receive.



Instead of sharing a public key, you can share a certificate. The user of the certificate can verify that it comes from you by checking with the certificate authority who signed it. Certificate by alimasykurm from the [Noun Project](#)

One big advantage of using X.509 certificates is that they can be shared between devices. You can create one certificate, upload it to IoT Hub, and use this for all your devices. Each device then just needs to know the private key to decrypt the messages it receives from IoT Hub.

The certificate used by your device to encrypt messages it sends to the IoT Hub is published by Microsoft. It is the same certificate that a lot of Azure services use, and is sometimes built into the SDKs

 Remember, a public key is just that - public. The Azure public key can only be used to encrypt data sent to Azure, not to decrypt it, so it can be shared everywhere, including in source code. For example, you can see it in the [Azure IoT C SDK source code](#).

 There is a lot of jargon with X.509 certificates. You can read the definitions of some of the terms you might come across in [The layman's guide to X.509 certificate jargon](#)

Generate and use an X.509 certificate

The steps to generate an X.509 certificate are:

1. Create a public/private key pair. One of the most widely used algorithm to generate a public/private key pair is called [Rivest–Shamir–Adleman\(RSA\)](#).
2. Submit the public key with associated data for signing, either by a CA, or by self-signing

The Azure CLI has commands to create a new device identity in IoT Hub, and automatically generate the public/private key pair and create a self-signed certificate.

 If you want to see the steps in detail, rather than using the Azure CLI, you can find it in the [Using OpenSSL to create self-signed certificates tutorial in the Microsoft IoT Hub documentation](#)

Task - create a device identity using an X.509 certificate

1. Run the following command to register the new device identity, automatically generating the keys and certificates:

```
sh
az iot hub device-identity create --device-id soil-moisture-sensor-x509
--am x509_thumbprint \
--output-dir . \
--hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub.

This will create a device with an ID of `soil-moisture-sensor-x509` to distinguish from the device identity you created in the last lesson. This command will also create 2 files in the current directory:

- `soil-moisture-sensor-x509-key.pem` - this file contains the private key for the device.
- `soil-moisture-sensor-x509-cert.pem` - this is the X.509 certificate file for the device.

Keep these files safe! The private key file should not be checked into public source code control.

Task - use the X.509 certificate in your device code

Work through the relevant guide to connect your IoT device to the cloud using the X.509 certificate:

- [Arduino - Wio Terminal](#)
 - [Single-board computer - Raspberry Pi/Virtual IoT device](#)
-

Challenge

There are multiple ways to create, manage and delete Azure services such as Resource Groups and IoT Hubs. One way is the [Azure Portal](#) - a web-based interface that gives you a GUI to manage your Azure services.

Head to portal.azure.com and investigate the portal. See if you can create an IoT Hub using the portal, then delete it.

Hint - when creating services through the portal, you don't need to create a Resource Group up front, one can be created when you are creating the service. Make sure you delete it when you are finished!

You can find plenty of documentation, tutorials and guides on the Azure Portal in the [Azure portal documentation](#).

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read up on the history of cryptography on the [History of cryptography page on Wikipedia](#).
- Read up on X.509 certificates on the [X.509 page on Wikipedia](#).

Assignment

[Build a new IoT device](#)

Location tracking

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

The main process for getting food from a farmer to a consumer involves loading boxes of produce onto trucks, ships, airplanes, or other commercial transport vehicles, and delivering the food somewhere - either directly to a customer, or to a central hub or warehouse for processing. The whole end-to-end process from farm to consumer is part of a process called the *supply chain*. The video below from Arizona State University's W. P. Carey School of Business talks about the idea of the supply chain and how it is managed in more detail.



🎥 Click the image above to watch a video

Adding IoT devices can drastically improve your supply chain, allowing you to manage where items are, plan transport and goods handling better, and respond quicker to problems.

When managing a fleet of vehicles such as trucks, it is helpful to know where each vehicle is at a given time. Vehicles can be fitted with GPS sensors that send their location to IoT systems, allowing the owners to pinpoint their location, see the route they have taken, and know when they will arrive at their destination. Most vehicles operate outside of WiFi coverage, so they use cellular networks to send this kind of data. Sometimes the GPS sensor is built into more complex IoT devices such as electronic log books. These devices track how long a truck has been in transit to ensure drivers are in compliance with local laws on working hours.

In this lesson you will learn how to track a vehicles location using a Global Positioning System (GPS) sensor.

In this lesson we'll cover:

- [Connected vehicles](#)
- [Geospatial coordinates](#)
- [Global Positioning Systems \(GPS\)](#)
- [Read GPS sensor data](#)
- [NMEA GPS data](#)
- [Decode GPS sensor data](#)

Connected vehicles

IoT is transforming the way goods are transported by creating fleets of *connected vehicles*. These vehicles are connected to central IT systems reporting information on their location, and other sensor data. Having a fleet of connected vehicles has a wide range of benefits:

- Location tracking - you can pinpoint where a vehicle is at any time, allowing you to:
 - Get alerts when a vehicle is about to arrive at a destination to prepare a crew for unloading
 - Locate stolen vehicles
 - Combine location and route data with traffic problems to allow you to re-route vehicles mid-journey
 - Be compliant with tax. Some countries charge vehicles for the amount of mileage driven on public roads (such as New Zealand's RUC), so knowing when a vehicle is on public roads vs private roads makes it easier to calculate tax owed.
 - Know where to send maintenance crews in the event of a breakdown
- Driver telemetry - being able to ensure drivers are adhering to speed limits, cornering at appropriate speeds, braking early and efficiently, and driving safely. Connected vehicles can also have cameras to record incidents. This can be linked to insurance, giving reduced rates for good drivers.
- Driver hours compliance - ensuring drivers only drive for their legally allowed hours based on the times they turn the engine on and off.

These benefits can be combined - for example, combining driver hours compliance with location tracking to re-route drivers if they cannot reach their destination within their allowed driving hours. These can also be combined with other vehicle-specific telemetry, such as temperature data from temperature-controlled trucks, allow vehicles to be re-routed if their current route would mean goods cannot be kept at temperature.

 Logistics is the process of transporting goods from one place to another, such as from a farm to a supermarket via one or more warehouses. A farmer packs boxes of tomatoes that are loaded onto a truck, delivered to a central warehouse, and put onto a second truck that may contain a mixture of different types of produce which are then delivered to a supermarket.

The core component of vehicle tracking is GPS - sensors that can pinpoint their location anywhere on Earth. In this lesson you will learn how to use a GPS sensor, starting with learning about how to define a location on Earth.

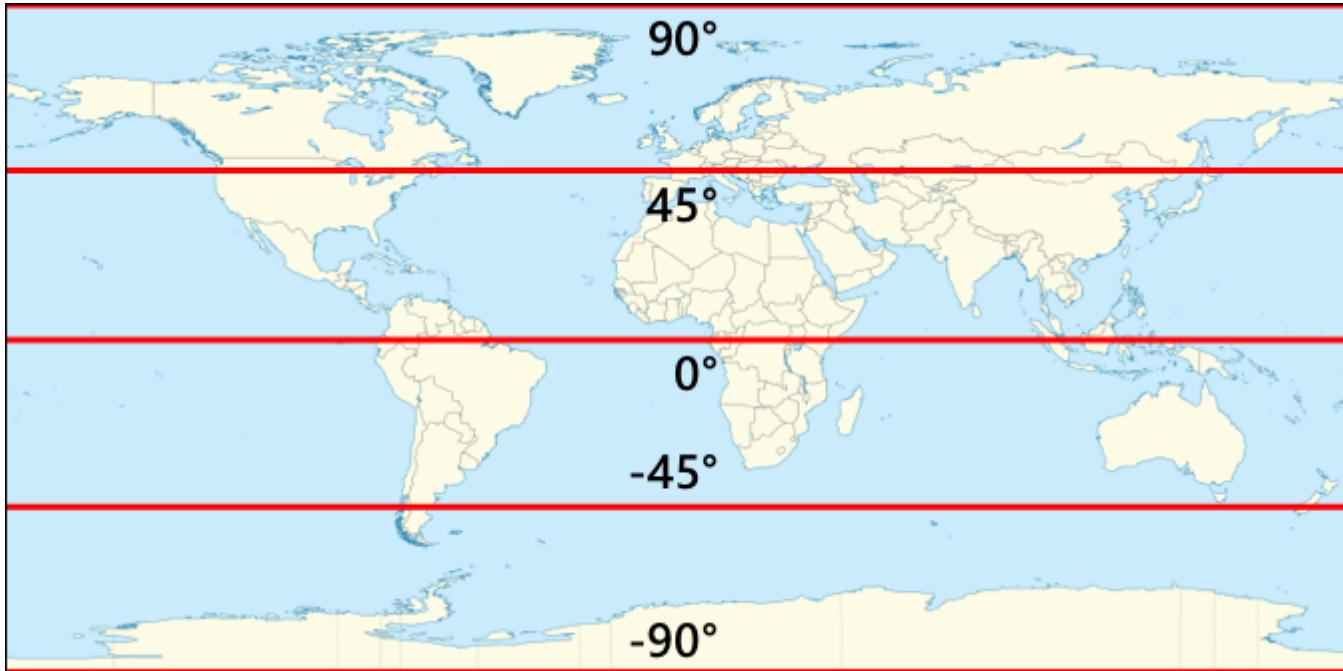
Geospatial coordinates

Geospatial coordinates are used to define points on the Earth's surface, similar to how coordinates can be used to draw to a pixel on a computer screen or position stitches in cross stitch. For a single point, you have a pair of coordinates. For example, the Microsoft Campus in Redmond, Washington, USA is located at 47.6423109, -122.1390293.

Latitude and longitude

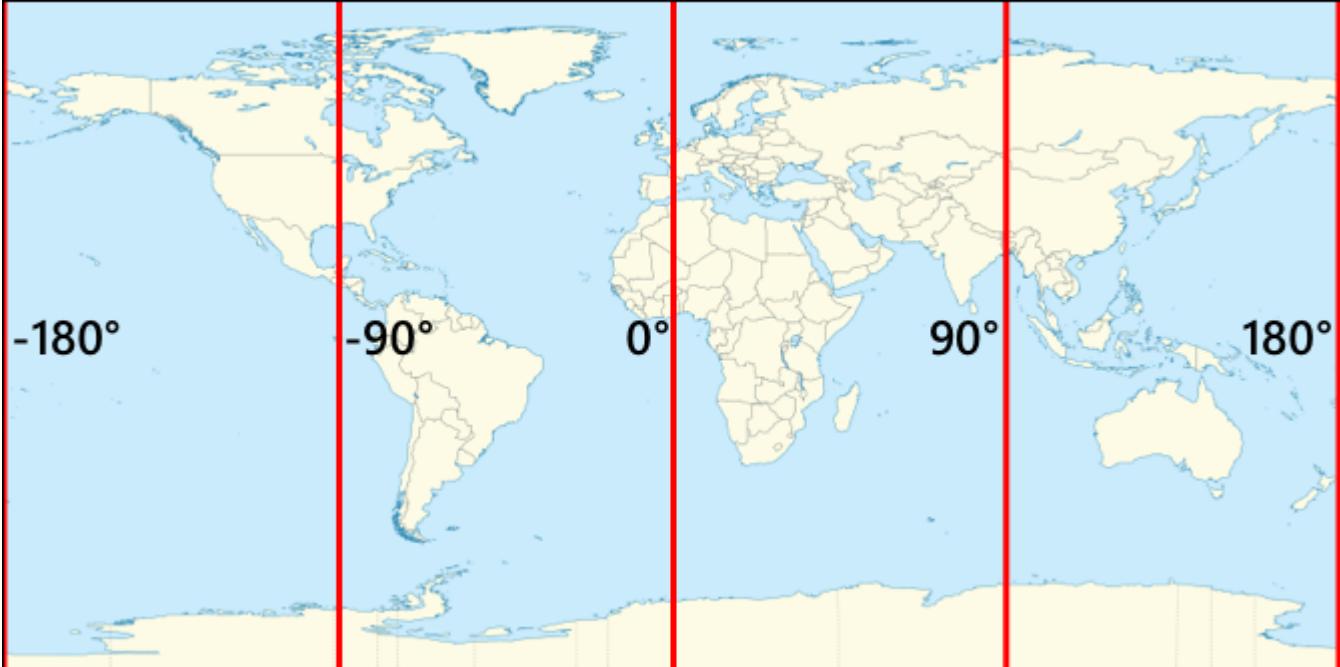
The Earth is a sphere - a three-dimensional circle. Because of this, points are defined by dividing it into 360 degrees, the same as the geometry of circles. Latitude measures the number of degrees north to south, longitude measures the number of degrees east to west.

 No-one really knows the original reason why circles are divided into 360 degrees. The [degree \(angle\) page on Wikipedia](#) covers some of the possible reasons.



Latitude is measured using lines that circle the Earth and run parallel to the equator, dividing the Northern and Southern Hemispheres into 90° each. The equator is at 0°, the North Pole is 90°, also known as 90° North, and the South Pole is at -90°, or 90° South.

Longitude is measured as the number of degrees measured east and west. The 0° origin of longitude is called the *Prime Meridian*, and was defined in 1884 to be a line from the North to the South Pole that goes through the [British Royal Observatory in Greenwich, England](#).



🎓 A meridian is an imaginary straight line that goes from the North Pole to the South Pole, forming a semicircle.

To measure the longitude of a point, you measure the number of degrees round the equator from the Prime Meridian to a meridian that passes through that point. Longitude goes from -180°, or 180° West, through 0° at the Prime Meridian, to 180°, or 180° East. 180° and -180° refer to the same point, the antimeridian or 180th meridian. This is a meridian on the opposite side of the Earth from the Prime Meridian.

👉 The antimeridian is not to be confused with the International Date Line, which is in approximately the same position, but is not a straight line and varies to fit around geo-political boundaries.

✓ Do some research: Try to find the latitude and longitude of your current location.

Degrees, minutes and seconds vs decimal degrees

Traditionally, measurements of degrees of latitude and longitude were done using sexagesimal numbering, or base-60, a numbering system used by the Ancient Babylonians who did the first measurements and recordings of time and distance. You use sexagesimal every day probably without even realising it - dividing hours into 60 minutes and minutes into 60 seconds.

Longitude and latitude are measured in degrees, minutes and seconds, with one minute being 1/60 of a degree, and 1 second being 1/60 minute.

For example, at the equator:

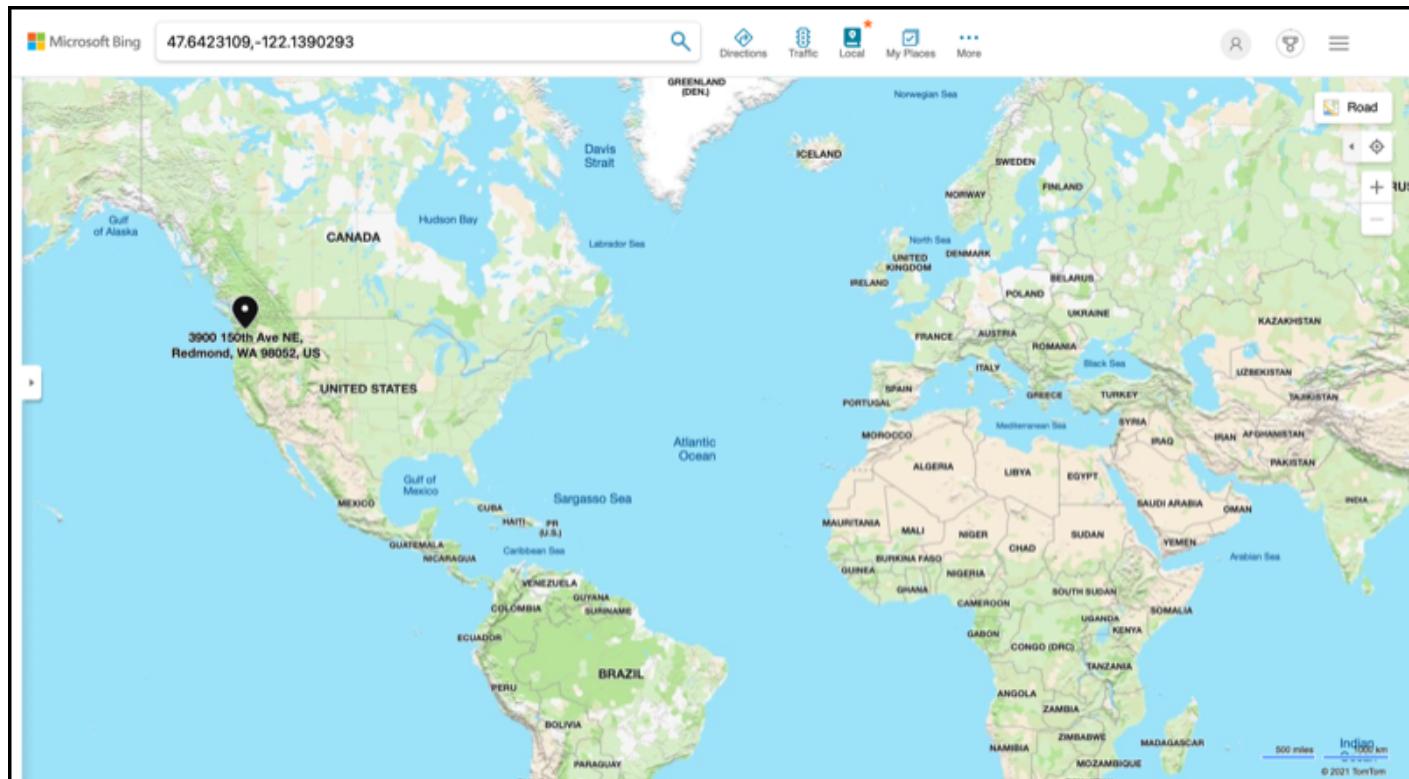
- 1° of latitude is **111.3 kilometers**
- 1 minute of latitude is $111.3/60 = \mathbf{1.855 \text{ kilometers}}$
- 1 second of latitude is $1.855/60 = \mathbf{0.031 \text{ kilometers}}$

The symbol for a minute is a single quote, for a second it is a double quote. 2 degrees, 17 minutes, and 43 seconds for example, would be written as $2^{\circ}17'43''$. Parts of seconds are given as decimals, for example half a second is $0^{\circ}0'0.5''$.

Computers don't work in base-60, so these coordinates are given as decimal degrees when using GPS data in most computer systems. For example, $2^{\circ}17'43''$ is 2.295277. The degree symbol is usually omitted.

Coordinates for a point are always given as `latitude, longitude`, so the example earlier of the Microsoft Campus at 47.6423109,-122.117198 has:

- A latitude of 47.6423109 (47.6423109 degrees north of the equator)
- A longitude of -122.1390293 (122.1390293 degrees west of the Prime Meridian).



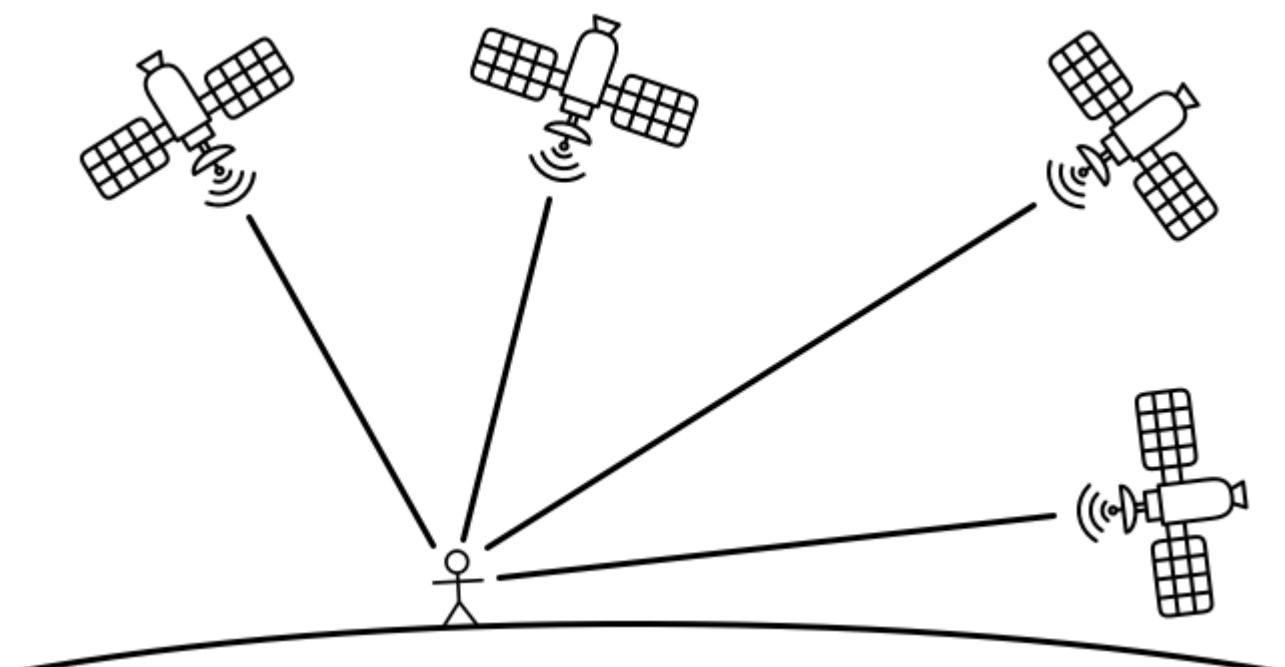
Global Positioning Systems (GPS)

GPS systems use multiple satellites orbiting the Earth to locate your position. You've probably used GPS systems without even knowing it - to find your location on a mapping app on your phone such as Apple Maps or Google Maps, or to see where your ride is in a ride hailing app such as Uber or Lyft, or when using satellite navigation (sat-nav) in your car.

👉 The satellites in 'satellite navigation' are GPS satellites!

GPS systems work by having a number of satellites that send a signal with each satellite's current position, and an accurate timestamp. These signals are sent over radio waves and are detected by an antenna in the GPS sensor. A GPS sensor will detect these signals, and using the current time measure how long it took for the signal to reach the sensor from the satellite. Because the speed of radio waves is constant, the GPS sensor can use the time stamp that was sent to work out how far away the sensor is from the satellite. By combining the data from at least 3 satellites with their positions sent, the GPS sensor is able to pinpoint its location on Earth.

👉 GPS sensors need antennas to detect radio waves. The antennas built into trucks and cars with on-board GPS are positioned to get a good signal, usually on the windshield or roof. If you are using a separate GPS system, such as a smartphone or an IoT device, then you need to ensure that the antenna built into the GPS system or phone has a clear view of the sky, such as being mounted on your windshield.



By knowing the distance from the sensor to multiple satellites, the location can be calculated.

Satellite by Noura Mbarki from the [Noun Project](#)

GPS satellites are circling the Earth, not at a fixed point above the sensor, so location data includes altitude above sea level as well as latitude and longitude.

GPS used to have limitations on accuracy enforced by the US military, limiting accuracy to around 5 meters. This limitation was removed in 2000, allowing an accuracy of 30 centimeters. Getting this accuracy is not always possible due to interference with the signals.

- If you have a smart phone, launch the mapping app and see how accurate your location is. It may take a short period of time for your phone to detect multiple satellites to get a more accurate location.

 The satellites contain atomic clocks that are incredibly accurate, but they drift by 38 microseconds (0.0000038 seconds) a day compared to atomic clocks, due to time slowing down as speed increases as predicted by Einstein's theories of special and general relativity - the satellites travel faster than the Earth's rotation. This drift has been used to prove the predictions of special and general relativity, and has to be adjusted for in the design of GPS systems. Literally time runs slower on a GPS satellite.

GPS systems have been developed and deployed by a number of countries and political unions including the US, Russia, Japan, India, the EU, and China. Modern GPS sensor can connect to most of these systems to get faster and more accurate fixes.

 The groups of satellites in each deployment are referred to as constellations.

Read GPS sensor data

Most GPS sensors send GPS data over UART.

 UART was covered in [project 2, lesson 2](#). Refer back to that lesson if needed.

You can use a GPS sensor on your IoT device to get GPS data.

Task - connect a GPS sensor and read GPS data

Work through the relevant guide to measure soil moisture using your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

NMEA GPS data

When you ran your code, you would have seen what might appear to be gibberish in the output. This is actually standard GPS data, and it all has meaning.

GPS sensors output data using NMEA messages, using the NMEA 0183 standard. NMEA is an acronym for the [National Marine Electronics Association](#), a US-based trade organization that sets standard for communication between marine electronics.

 This standard is proprietary and sells for at least US\$2,000, but enough information about it is in the public domain that most of the standard has been reverse engineered and can be used in open source and other non-commercial code.

These messages are text-based. Each message consists of a sentence that starts with a `$` character, followed by 2 characters to indicate the source of the message (e.g GP for the US GPS system, GN for GLONASS, the Russian GPS system), and 3 characters to indicate the type of message. The rest of the message is fields separated by commas, ending in a new line character.

Some of the types of messages that can be received are:

Type	Description
GGA	GPS Fix Data, including the latitude, longitude, and altitude of the GPS sensor, along with the number of satellites in view to calculate this fix.
ZDA	The current date and time, including the local time zone
GSV	Details of the satellites in view - defined as the satellites that GPS sensor can detect signals from

 GPS data includes time stamps, so your IoT device can get the time if needed from a GPS sensor, rather than relying on an NTP server or internal real-time clock.

The GGA message includes the current location using the `(dd)dmm.mmmm` format, along with a single character to indicate direction. The `d` in the format is degrees, the `m` is minutes, with seconds as decimals of minutes. For example, $2^{\circ}17'43''$ would be 217.716666667 - 2 degrees, 17.716666667 minutes.

The direction character can be `N` or `S` for latitude to indicate north or south, and `E` or `W` for longitude to indicate east or west. For example, a latitude of $2^{\circ}17'43''$ would have a direction character of `N`, $-2^{\circ}17'43''$ would have a direction character of `S`.

For example - the NMEA sentence

`$GNGGA,020604.001,4738.538654,N,12208.341758,W,1,3,,164.7,M,-17.1,M,,*67`

- The latitude part is `4738.538654,N`, which converts to 47.6423109 in decimal degrees. `4738.538654` is 47.6423109, and the direction is `N` (north), so it is a positive latitude.
- The longitude part is `12208.341758,W`, which converts to -122.1390293 in decimal degrees. `12208.341758` is 122.1390293°, and the direction is `W` (west), so it is a negative longitude.

Decode GPS sensor data

Rather than use the raw NMEA data, it is better to decode it into a more useful format. There are multiple open-source libraries you can use to help extract useful data from the raw NMEA messages.

Task - decode GPS sensor data

Work through the relevant guide to decode GPS sensor data using your IoT device:

- [Arduino - Wio Terminal](#)
 - [Single-board computer - Raspberry Pi/Virtual IoT device](#)
-

Challenge

Write your own NMEA decoder! Rather than relying on third party libraries to decode NMEA sentences, can you write your own decoder to extract latitude and longitude from NMEA sentences?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read more on Geospatial Coordinates on the [Geographic coordinate system page on Wikipedia](#).
- Read up on the Prime Meridians on other celestial bodies besides the Earth on the [Prime Meridian page on Wikipedia](#)
- Research the various different GPS systems from various world governments and political unions such as the EU, Japan, Russia, India and the US.

Assignment

[Investigate other GPS data](#)

Store location data

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last lesson, you learned how to use a GPS sensor to capture location data. To use this data to visualize the both the location of a truck laden with food, but also its journey, it needs to be sent to an IoT service in the cloud, and then stored somewhere.

In this lesson you will learn about the different ways to store IoT data, and learn how to store data from your IoT service using serverless code.

In this lesson we'll cover:

- [Structured and unstructured data](#)
- [Send GPS data to an IoT Hub](#)
- [Handle GPS events using serverless code](#)
- [Azure Storage Accounts](#)
- [Connect your serverless code to storage](#)

Structured and unstructured data

Computer systems deal with data, and this data comes in all manner of different shapes and sizes. It can vary from single numbers, to large amounts of text, to videos and images, and to IoT data. Data can usually be divided into one of two categories - *structured* data and *unstructured* data.

- **Structured data** is data with a well-defined, rigid structure that doesn't change and usually maps to tables of data with relationships. One example is a person's details including their name, date of birth and address.
- **Unstructured data** is data without a well-defined, rigid structure, including data that can change structure frequently. One example is documents such as written documents or spreadsheets.

 Do some research: Can you think of some other examples of structured and unstructured data?

 There is also semi-structured data that is structured but doesn't fit into fixed tables of data

IoT data is usually considered to be unstructured data.

Imagine you were adding IoT devices to a fleet of vehicles for a large commercial farm. You might want to use different devices for different types of vehicle. For example:

- For farm vehicles like tractors you want GPS data to ensure they are working on the correct fields

- For delivery trucks transporting food to warehouses you want GPS data as well as speed and acceleration data to ensure the driver is driving safely, and drive identity and start/stop data to ensure drive compliance with local laws on working hours
- For refrigerated trucks you also want temperature data to ensure the food doesn't get too hot or cold and spoil in transit

This data can change constantly. For example, if the IoT device is in a truck cab, then the data it sends may change as the trailer changes, for example only sending temperature data when a refrigerated trailer is used.

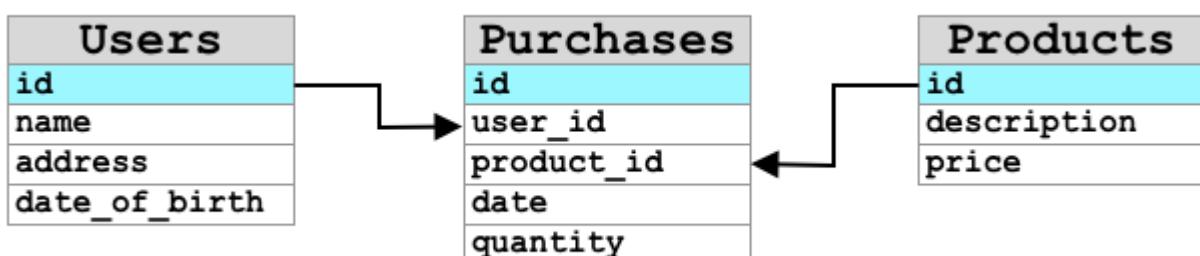
This data varies from vehicle to vehicle, but it all gets sent to the same IoT service for processing. The IoT service needs to be able to process this unstructured data, storing it in a way that allows it to be searched or analyzed, but works with different structures to this data.

SQL vs NoSQL storage

Databases are services that allow you to store and query data. Database come in two types - SQL and NoSQL

SQL databases

The first databases were Relational Database Management Systems (RDBMS), or relational database. These are also known as SQL databases after the Structured Query Language (SQL) used to interact with them to add, remove, update or query data. These database consist of a schema - a well-defined set of tables of data, similar to a spreadsheet. Each table has multiple named columns. When you insert data, you add a row to the table, putting values into each of the columns. This keeps the data in a very rigid structure - although you can leave columns empty, if you want to add a new column you have to do this on the database, populating values for the existing rows. These databases are relational - in that one table can have a relationship to another.



For example, if you stored a users personal details in a table, you would have some kind of internal unique ID per user that is used in a row in a table that contains the users name and address. If you then wanted to store other details about that user, such as their purchases, in another table, you would have one column in the new table for that users ID. When you look up a user, you can use their ID to get their personal details from one table, and their purchases from another.

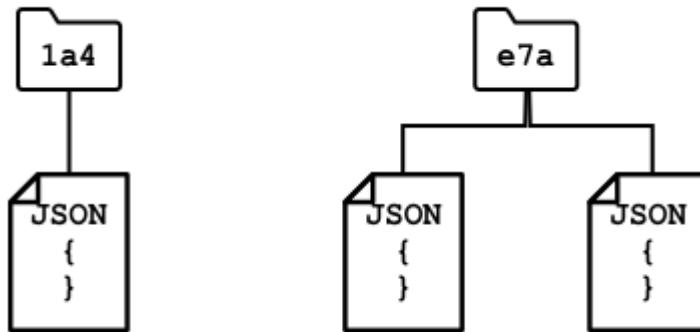
SQL databases are ideal for storing structured data, and for when you want to ensure the data matches your schema. Some well known SQL databases are Microsoft SQL Server, MySQL, and PostgreSQL.

- ✓ If you haven't used SQL before, take a moment to read up on it on the [SQL page on Wikipedia](#).

NoSQL database

NoSQL databases are so called because they don't have the same rigid structure of SQL databases. There are also known as document databases as they can store unstructured data such as documents.

 Despite their name, some NoSQL databases allow you to use SQL to query the data.



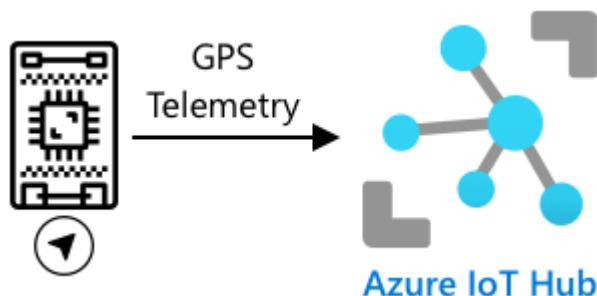
NoSQL database do not have a pre-defined schema that limits how data is stored, instead you can insert any unstructured data, usually using JSON documents. These documents can be organized into folders, similar to files on your computer. Each document can have different fields from other documents - for example if you were storing IoT data from your farm vehicles, some may have fields for accelerometer and speed data, others may have fields for the temperature in the trailer. If you were to add a new truck type, such as one with built in scales to track the weight of produce carried, then your IoT device could add this new field and it could be stored without any changes to the database.

Some well known NoSQL databases include Azure CosmosDB, MongoDB, and CouchDB.

In this lesson, you will be using NoSQL storage to store IoT data.

Send GPS data to an IoT Hub

In the last lesson you captured GPS data from a GPS sensor connected to your IoT device. To store this IoT data in the cloud, you need to send it to an IoT service. Once again, you will be using Azure IoT Hub, the same IoT cloud service you used in the previous project.



Sending GPS telemetry from an IoT device to IoT Hub. GPS by mim studio / Microcontroller by Template - all from the [Noun Project](#)

Task - send GPS data to an IoT Hub

1. Create a new IoT Hub using the free tier.

⚠ You can refer to [the instructions for creating an IoT Hub from project 2, lesson 4 if needed.](#)

Remember to create a new Resource Group. Name the new Resource Group `gps-sensor` , and the new IoT Hub a unique name based on `gps-sensor` , such as `gps-sensor-<your name>` .

💡 If you still have your IoT Hub from the previous project, you can re-use it. Remember to use the name of this IoT Hub and the Resource Group it is in when creating other services.

2. Add a new device to the IoT Hub. Call this device `gps-sensor` . Grab the connection string for the device.
3. Update your device code to send the GPS data to the new IoT Hub using the device connection string from the previous step.

⚠ You can refer to [the instructions for connecting your device to an IoT from project 2, lesson 4 if needed.](#)

4. When you send the GPS data, do it as JSON in the following format:

```
json
{
  "gps" :
  {
    "lat" : <latitude>,
    "lon" : <longitude>
  }
}
```

5. Send GPS data every minute so you don't use up your daily message allocation.

If you are using the Wio Terminal, remember to add all the necessary libraries, and set the time using an NTP server. Your code will also need to ensure that it has read all the data from the serial port before sending the GPS location, using the existing code from the last lesson. Use the following code to construct the JSON document:

cpp

```
DynamicJsonDocument doc(1024);
doc["gps"]["lat"] = gps.location.lat();
doc["gps"]["lon"] = gps.location.lng();
```

If you are using a Virtual IoT device, remember to install all the needed libraries using a virtual environment.

For both the Raspberry Pi and Virtual IoT device, use the existing code from the last lesson to get the latitude and longitude values, then send them in the correct JSON format with the following code:

python

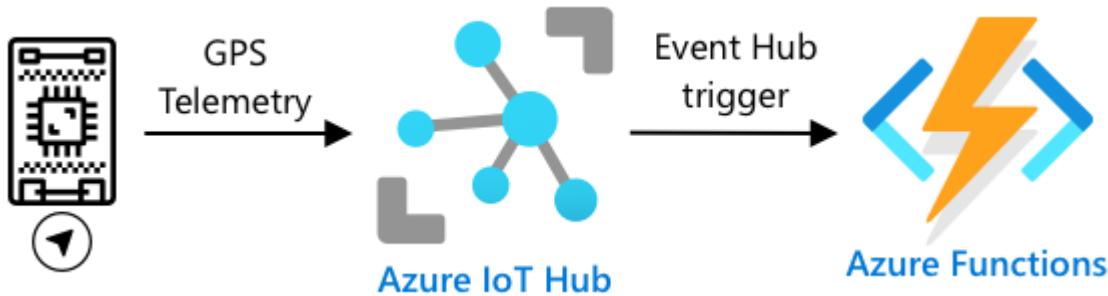
```
message_json = { "gps" : { "lat":lat, "lon":lon } }
print("Sending telemetry", message_json)
message = Message(json.dumps(message_json))
```

 You can find this code in the [code/wio-terminal](#), [code/pi](#) or [code/virtual-device](#) folder.

Run your device code and ensure messages are flowing into IoT Hub using the `az iot hub monitor-events` CLI command.

Handle GPS events using serverless code

Once data is flowing into your IoT Hub, you can write some serverless code to listen for events published to the Event-Hub compatible endpoint.



Sending GPS telemetry from an IoT device to IoT Hub, then to Azure Functions via an event hub trigger. GPS by mim studio / Microcontroller by Template - all from the [Noun Project](#)

Task - handle GPS events using serverless code

1. Create an Azure Functions app using the Azure Functions CLI. Use the Python runtime, and create it in a folder called `gps-trigger`, and use the same name for the Functions App project name. Make sure you create a virtual environment to use for this.

⚠ You can refer to [the instructions for creating an Azure Functions Project from project 2, lesson 5 if needed.](#)

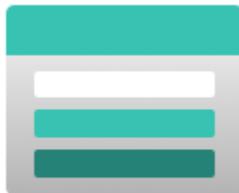
2. Add an IoT Hub event trigger that uses the IoT Hub's Event Hub compatible endpoint.

⚠ You can refer to [the instructions for creating an IoT Hub event trigger from project 2, lesson 5 if needed.](#)

3. Set the Event Hub compatible endpoint connection string in the `local.settings.json` file, and use the key for that entry in the `function.json` file.
4. Use the Azurite app as a local storage emulator
5. Run your functions app to ensure it is receiving events from your GPS device. Make sure your IoT device is also running and sending GPS data.

Python EventHub trigger processed an event: {"gps": {"lat": 47.73481, "l

Azure Storage Accounts



Azure Storage

Azure Storage Accounts is a general purpose storage service that can store data in a variety of different ways. You can store data as blobs, in queues, in tables, or as files, and all at the same time.

Blob storage

The word *Blob* means binary large objects, but has become the term for any unstructured data. You can store any data in blob storage, from JSON documents containing IoT data, to image and movie files. Blob storage has the concept of *containers*, named buckets that you can store data in, similar to tables in a relational database. These containers can have one or more folders to store blobs, and each folder can contain other folders, similar to how files are stored on your computer hard disk.

You will use blob storage in this lesson to store IoT data.

Do some research: Read up on [Azure Blob Storage](#)

Table storage

Table storage allows you to store semi-structured data. Table storage is actually a NoSQL database, so doesn't require a defined set of tables up front, but is designed to store data in one or more tables, with unique keys to define each row.

Do some research: Read up on [Azure Table Storage](#)

Queue storage

Queue storage allows you to store messages of up to 64KB in size in a queue. You can add messages to the back of the queue, and read them off the front. Queues store messages indefinitely as long as

there is still storage space, so allows messages to be stored long term. then read off when needed. For example, if you wanted to run a monthly job to process GPS data you could add it to a queue every day for a month, then at the end of the month process all the messages off the queue.

- ✓ Do some research: Read up on [Azure Queue Storage](#)

File storage

File storage is storage of files in the cloud, and any apps or devices can connect using industry standard protocols. You can write files to file storage, then mount it as a drive on your PC or Mac.

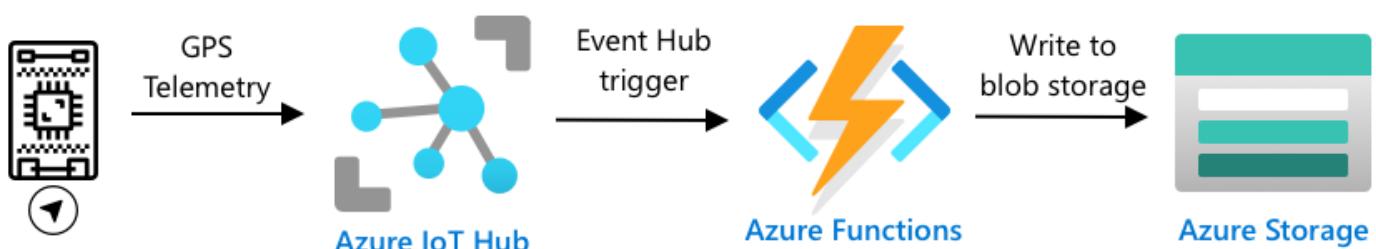
- ✓ Do some research: Read up on [Azure File Storage](#)

Connect your serverless code to storage

Your function app now needs to connect to blob storage to store the messages from the IoT Hub. There's 2 ways to do this:

- Inside the function code, connect to blob storage using the blob storage Python SDK and write the data as blobs
- Use an output function binding to bind the return value of the function to blob storage and have the blob saved automatically

In this lesson, you will use the Python SDK to see how to interact with blob storage.



Sending GPS telemetry from an IoT device to IoT Hub, then to Azure Functions via an event hub trigger, then saving it to blob storage. GPS by mim studio / Microcontroller by Template - all from the [Noun Project](#)

The data will be saved as a JSON blob with the following format:

```
{  
  "device_id": <device_id>,  
  "timestamp" : <time>,  
  "data": <data>}
```

```
"gps" :  
{  
    "lat" : <latitude>,  
    "lon" : <longitude>  
}  
}
```

Task - connect your serverless code to storage

1. Create an Azure Storage account. Name it something like `gps<your name>`.

 You can refer to [the instructions for creating a storage account from project 2, lesson 5 if needed.](#)

If you still have a storage account from the previous project, you can re-use this.

 You will be able to use the same storage account to deploy your Azure Functions app later in this lesson.

2. Run the following command to get the connection string for the storage account:

```
sh  
az storage account show-connection-string --output table \  
--name <storage_name>
```

Replace `<storage_name>` with the name of the storage account you created in the previous step.

3. Add a new entry to the `local.settings.json` file for your storage account connection string, using the value from the previous step. Name it `STORAGE_CONNECTION_STRING`
4. Add the following to the `requirements.txt` file to install the Azure storage Pip packages:

```
sh  
azure-storage-blob
```

Install the packages from this file in your virtual environment.

If you get an error, then upgrade your Pip version in your virtual environment to the latest version with the following command, then try again:

sh

```
pip install --upgrade pip
```

5. In the `__init__.py` file for the `iot-hub-trigger`, add the following import statements:

python

```
import json
import os
import uuid
from azure.storage.blob import BlobServiceClient, PublicAccess
```

The `json` system module will be used to read and write JSON, the `os` system module will be used to read the connection string, the `uuid` system module will be used to generate a unique ID for the GPS reading.

The `azure.storage.blob` package contains the Python SDK to work with blob storage.

6. Before the `main` method, add the following helper function:

python

```
def get_or_create_container(name):
    connection_str = os.environ['STORAGE_CONNECTION_STRING']
    blob_service_client = BlobServiceClient.from_connection_string(conne

    for container in blob_service_client.list_containers():
        if container.name == name:
            return blob_service_client.get_container_client(container.na

    return blob_service_client.create_container(name, public_access=Publ
```

The Python blob SDK doesn't have a helper method to create a container if it doesn't exist. This code will load the connection string from the `local.settings.json` file (or the Application Settings once deployed to the cloud), then create a `BlobServiceClient` class from this to interact with the blob storage account. It then loops through all the containers for the blob storage account, looking for one with the provided name - if it finds one it will return a `ContainerClient` class that can interact with the container to create blobs. If it doesn't find one, then the container is created and the client for the new container is returned.

When the new container is created, public access is granted to query the blobs in the container. This will be used in the next lesson to visualize the GPS data on a map.

- Unlike with soil moisture, with this code we want to store every event, so add the following code inside the `for event in events:` loop in the `main` function, below the `logging` statement:

```
python
```

```
device_id = event.iothub_metadata['connection-device-id']
blob_name = f'{device_id}/{str(uuid.uuid1())}.json'
```

This code gets the device ID from the event metadata, then uses it to create a blob name. Blobs can be stored in folders, and device ID will be used for the folder name, so each device will have all its GPS events in one folder. The blob name is this folder, followed by a document name, separated with forward slashes, similar to Linux and macOS paths (similar to Windows as well, but Windows uses back slashes). The document name is a unique ID generated using the Python `uuid` module, with the file type of `.json`.

For example, for the `gps-sensor` device ID, the blob name might be `gps-sensor/a9487ac2-b9cf-11eb-b5cd-1e00621e3648.json`.

- Add the following code below this:

```
python
```

```
container_client = get_or_create_container('gps-data')
blob = container_client.get_blob_client(blob_name)
```

This code gets the container client using the `get_or_create_container` helper class, and then gets a blob client object using the blob name. These blob clients can refer to existing blobs, or as in this case, to new blob.

- Add the following code after this:

```
python
```

```
event_body = json.loads(event.get_body().decode('utf-8'))
blob_body = {
    'device_id' : device_id,
    'timestamp' : event.iothub_metadata['enqueuedtime'],
    'gps': event_body['gps']
}
```

This builds the body of the blob that will be written to blob storage. It is a JSON document containing the device ID, the time the telemetry was sent to IoT Hub, and the GPS coordinates

from the telemetry.

 It is important to use the enqueued time of the message as opposed to the current time to get the time that the message was sent. It could be sitting on the hub for a while before being picked up if the Functions App is not running.

10. Add the following below this code:

```
python
logging.info(f'Writing blob to {blob_name} - {blob_body}')
blob.upload_blob(json.dumps(blob_body).encode('utf-8'))
```

This code logs that a blob is about to be written with it's details, then uploads the blob body as the content of the new blob.

11. Run the Functions app. You will see blobs being written for all the GPS events in the output:

```
output
[2021-05-21T01:31:14.325Z] Python EventHub trigger processed an event: {
...
[2021-05-21T01:31:14.351Z] Writing blob to gps-sensor/4b6089fe-ba8d-11eb
```

 You can find this code in the [code/functions](#) folder.

Task - verify the uploaded blobs

1. To view the blobs created, you can either use the [Azure Storage Explorer](#), a free tool that allows you to view and manage your storage accounts, or from the CLI.

1. To use the CLI, first you will need an account key. Run the following command to get this key:

```
sh
az storage account keys list --output table \
--account-name <storage_name>
```

Replace `<storage_name>` with the name of the storage account.

Copy the value of `key1`.

2. Run the following command to list the blobs in the container:

```
sh  
az storage blob list --container-name gps-data \  
    --output table \  
    --account-name <storage_name> \  
    --account-key <key1>
```

Replace `<storage_name>` with the name of the storage account, and `<key1>` with the value of `key1` you copied in the last step.

This will list out all the blobs in the container:

Name	Blob Type	Bl	output
gps-sensor/1810d55e-b9cf-11eb-9f5b-1e00621e3648.json	BlockBlob	Ho	
gps-sensor/18293e46-b9cf-11eb-9f5b-1e00621e3648.json	BlockBlob	Ho	
gps-sensor/1844549c-b9cf-11eb-9f5b-1e00621e3648.json	BlockBlob	Ho	
gps-sensor/1894d714-b9cf-11eb-9f5b-1e00621e3648.json	BlockBlob	Ho	

3. Download one of the blobs using the following command:

```
sh  
az storage blob download --container-name gps-data \  
    --account-name <storage_name> \  
    --account-key <key1> \  
    --name <blob_name> \  
    --file <file_name>
```

Replace `<storage_name>` with the name of the storage account, and `<key1>` with the value of `key1` you copied in the earlier step.

Replace `<blob_name>` with the full name from the `Name` column of the output of the last step, including the folder name. Replace `<file_name>` with the name of a local file to save the blob to.

Once downloaded, you can open the JSON file in VS Code, and you will see the blob containing the GPS location details:

```
json  
{"device_id": "gps-sensor", "timestamp": "2021-05-21T00:57:53.878Z",
```

Task - deploy your Functions App to the cloud

Now that your Function app is working, you can deploy it to the cloud.

1. Create a new Azure Functions app, using the storage account you created earlier. Name this something like `gps-sensor-` and add a unique identifier on the end, like some random words or your name.

⚠ You can refer to [the instructions for creating a Functions app from project 2, lesson 5 if needed.](#)

2. Upload the `IOT_HUB_CONNECTION_STRING` and `STORAGE_CONNECTION_STRING` values to the Application Settings

⚠ You can refer to [the instructions for uploading Application Settings from project 2, lesson 5 if needed.](#)

3. Deploy your local Functions app to the cloud.

⚠ You can refer to [the instructions for deploying your Functions app from project 2, lesson 5 if needed.](#)

🚀 Challenge

GPS data is not perfectly accurate, and the locations being detected can be out by a few meters, if not more especially in tunnels and areas of tall buildings.

Think about how satellite navigation could overcome this? What data does your sat-nav have that would allow it to make better predictions on your location?

Post-lecture quiz

Review & Self Study

- Read about structured data on the [Data model page on Wikipedia](#)
- Read about semi-structured data on the [Semi-structured data page on Wikipedia](#)
- Read about unstructured data on the [Unstructured data page on Wikipedia](#)
- Read more on Azure Storage and the different storage types in the [Azure Storage documentation](#)

Assignment

[Investigate function bindings](#)

Visualize location data

Add a sketchnote if possible/appropriate

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last lesson you learned how to get GPS data from your sensors to save to the cloud in a storage container using serverless code. Now you will discover how to visualize those points on an Azure map.

In this lesson we'll cover:

- [What is Azure maps?](#)
- [Create an Azure Maps resource](#)
- [Show a map on a web page](#)
- [The GeoJSON format](#)

- [Plot GPS data on a Map using GeoJSON](#)

What is Azure maps?

Working with maps is an interesting exercise, and there are many to choose from such as Bing Maps, Leaflet, Open Street Maps, and Google Maps. In this lesson, you will learn about [Azure Maps](#) and how they can display your GPS data.

- ✓ Check out [this video](#) on using Azure Maps with IoT.

Azure Maps is "a collection of geospatial services and SDKs that use fresh mapping data to provide geographic context to web and mobile applications." Developers are provided with tools to create beautiful, interactive maps that can do things like provide recommended traffic routes, give information about traffic incidents, indoor navigation, search capabilities, elevation information, weather services and more.

- ✓ Experiment with some [mapping code samples](#)

You can display the maps as a blank canvas, tiles, satellite images, satellite images with roads superimposed, various types of grayscale maps, maps with shaded relief to show elevation, night view maps, and a high contrast map. You can get real-time updates on your maps by integrating them with [Azure Event Grid](#). You can control the behavior and look of your maps by enabling various controls to allow the map to react to events like pinch, drag, and click. To control the look of your map, you can add layers that include bubbles, lines, polygons, heat maps, and more. Which style of map you implement depends on your choice of SDK.

You can access Azure Maps APIs by leveraging its [REST API](#), its [Web SDK](#), or, if you are building a mobile app, its [Android SDK](#).

In this lesson, you will use the web SDK to draw a map and display your sensor's GPS location's path.

Create an Azure Maps resource

Your first step is to create an Azure Maps account. The easiest way to do this is in the [Azure portal](#).

1. After logging in to the portal, click the "Create a resource" button and in the search box that appears, type "Azure Maps".

2. Select "Azure Maps" and click 'Create'.

3. On the Create Azure Maps Account page, enter:

- Your subscription in the dropdown box.
- A Resource group to use (use 'gps-sensor' as you have done throughout these lessons)
- Add a name for your account
- Choose a Pricing tier. The Pricing tier for this account. S0 will work for this small project.

4. Read and accept the terms of service, and click the 'Create' button.

5. The service will deploy and you can visit it by clicking 'Go to resource' in the next screen.

6. Navigate to your new Azure Maps Account Authentication screen. Here, you discover that you have two ways to authenticate your maps in a web app: using Active Directory (AD) or 'Shared Key Authentication', also known as Subscription Key. We'll use the latter, for simplicity. Copy the Primary Key value and make a note of it!

- You will be able to rotate and swap keys at will using the Shared Keys; switch your app to use the Secondary Key while rotating the Primary Key if needed.

Show a map on a web page

Now you can take the next step which is to display your map on a web page. We will use just one .html file for your small web app; keep in mind that in a production or team environment, your web app will most likely have more moving parts!

1. Create a file called index.html in a folder somewhere on your local computer. Add HTML markup to hold a map:

html

```
<html>
<head>
    <style>
        #myMap {
            width:100%;
            height:100%;
        }
    </style>
</head>

<body onload="init()">
    <div id="myMap"></div>
```

```
</body>
</html>
```

The map will load in the 'myMap' div . A few styles allow it so span the width and height of the page.

2. Under the opening `<head>` tag, add an external style sheet to control the map display, and an external script from the Web SDK to manage its behavior:

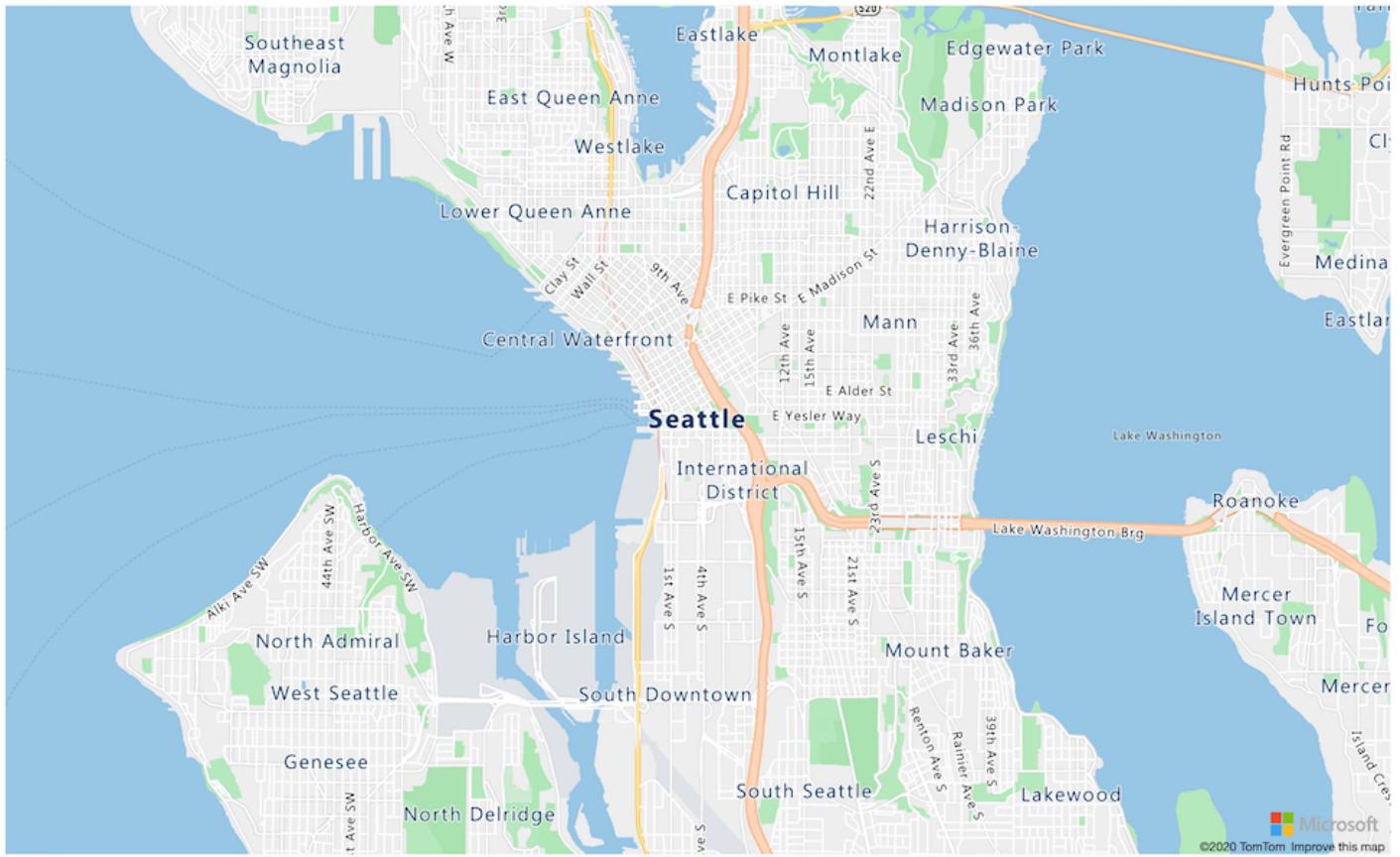
```
<link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.css"/>
<script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.js">
```

3. Under that script, add a script block to launch the map. Add your own subscriptionKey in the init() function:

```
javascript
<script type='text/javascript'>

    function init() {
        var map = new atlas.Map('myMap', {
            center: [-122.33, 47.6],
            zoom: 12,
            authOptions: {
                authType: "subscriptionKey",
                subscriptionKey: "<your-key-here>",
            }
        });
    }
</script>
```

If you open your index.html page in a web browser, you should see a map loaded, and focused on Seattle:



Experiment with the zoom and center parameters to change your map display.

A better way to work with web apps locally is to install [http-server](#). You will need [node.js](#) and [npm](#) installed before using this tool. Once those tools are installed, you can navigate to the location of your `index.html` file and type `http-server`. The web app will open on a local webserver <http://127.0.0.1:8080/>.

The GeoJSON format

Now that you have your web app in place with the map displaying, you need to extract GPS data from your storage and display it in a layer of markers on top of the map. Before we do that, let's look at the [GeoJSON](#) format that is required by Azure Maps.

[GeoJSON](#) is an open standard JSON specification with special formatting designed to handle geographic-specific data. You can learn about it by testing sample data using [geojson.io](#), which is also a useful tool to debug GeoJSON files.

Sample GeoJSON data looks like this:

```
{  
  "type": "FeatureCollection",  
  "features": [  
    {  
      "type": "Feature",  
      "geometry": {  
        "type": "Point",  
        "coordinates": [-122.332, 47.606]  
      },  
      "properties": {  
        "name": "Seattle Center",  
        "address": "1201 Third Avenue, Seattle, WA 98101",  
        "lat": 47.606, "lon": -122.332  
      }  
    }  
  ]  
}
```

```
"features": [
  {
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -2.10237979888916,
        57.164918677004714
      ]
    }
  }
]
```

Of particular interest is the way the data is nested as a 'FeatureCollection'. Within that object can be found 'geometry' with the 'coordinates' indicating latitude and longitude. Geometry can have different 'types' designated to that a polygon could be drawn to a map; in this case, a point is drawn with two coordinates designated.

- ✓ Azure Maps supports standard GeoJSON plus some [enhanced features](#) including the ability to draw circles and other geometries.

Plot GPS data on a Map using GeoJSON

Now you are ready to consume data from the storage that you built in the previous lesson. As a reminder, it is stored as a number of files in blob storage so you will need to retrieve the files and parse them so that Azure Maps can use the data.

If you make a call to your storage to fetch the data you might be surprised to see errors occurring in your browser's console. That's because you need to set permissions for [CORS](#) on this storage to allow external web apps to read its data. CORS stands for "Cross-Origin Resource Sharing" and usually needs to be set explicitly in Azure for security reasons. Do this using the Azure CLI, adding the name of your storage container and its key. We only need to 'GET' data from this container:

dotnetcli

```
az storage cors add --methods GET \
                     --origins "*" \
                     --services b \
                     --account-name <storage_name> \
                     --account-key <key1>
```

TODO - fetch call explanation

Challenge

It's nice to be able to display static data on a map as markers. Can you enhance this web app to add animation and show the path of the markers over time, using the timestamped json files? Here are some samples of using animation

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

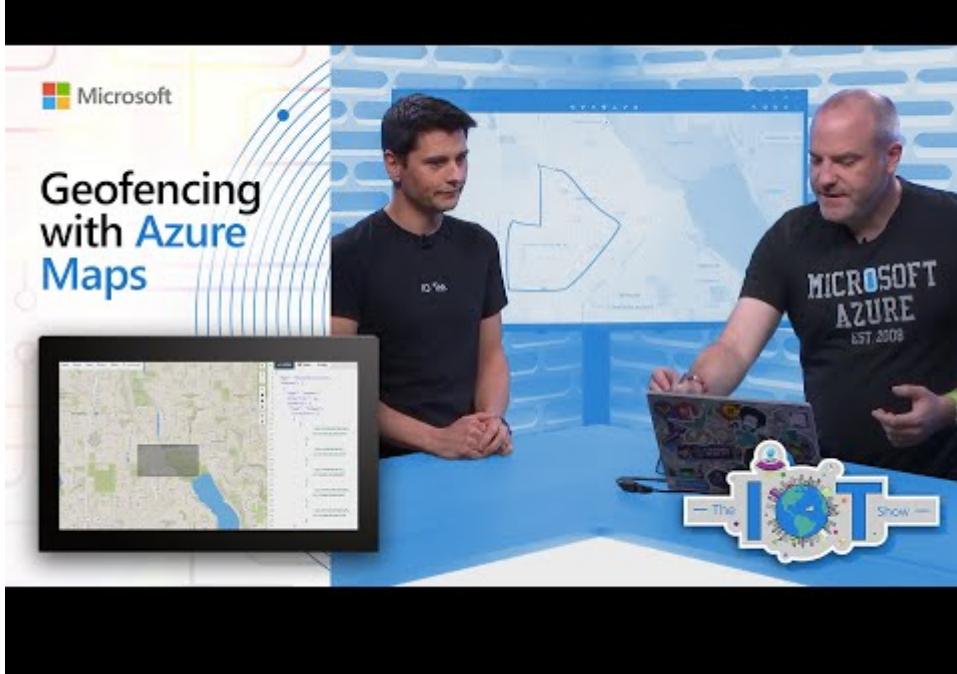
Assignment

[Deploy your app](#)

Geofences

Add a sketchnote if possible/appropriate

This video gives an overview of geofences and how to use them in Azure Maps, topics that will be covered in this lesson:



🎥 Click the image above to watch a video

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last 3 lessons, you have used IoT to locate the trucks carrying your produce from your farm to a processing hub. You've captured GPS data, sent it to the cloud to store, and visualized it on a map. The next step in increasing the efficiency of your supply chain is to get an alert when a truck is about to arrive at the processing hub, so that the crew needed to unload can be ready with forklifts and other equipment as soon as the vehicle arrives. This way they can unload quickly, and you are not paying for a truck and driver to wait.

In this lesson you will learn about geofences - defined geospatial regions such as an area within a 2km minute drive of a processing hub, and how to test if GPS coordinates are inside or outside a geofence, so you can see if your GPS sensor has arrived or left an area.

In this lesson we'll cover:

- [What are geofences](#)

- [Define a geofence](#)
- [Test points against a geofence](#)
- [Use geofences from serverless code](#)

💡 This is the last lesson in this project, so after completing this lesson and the assignment, don't forget to clean up your cloud services. You will need the services to complete the assignment, so make sure to complete that first.

Refer to [the clean up your project guide](#) if necessary for instructions on how to do this.

What are Geofences

A geofence is a virtual perimeter for a real-world geographic region. Geofences can be circles defined as a point and a radius (for example a circle 100m wide around a building), or a polygon covering an area such as a school zone, city limits, or university or office campus.



💡 You may have already used geofences without knowing. If you've set a reminder using the iOS reminders app or Google Keep based off a location, you have used a geofence. These

apps will set up a geofence based off the location given and alert you when your phone enters the geofence.

There are many reasons why you would want to know that a vehicle is inside or outside a geofence:

- Preparation for unloading - getting a notification that a vehicle has arrived on-site allows a crew to be prepared to unload the vehicle, reducing vehicle waiting time. This can allow a driver to make more deliveries in a day with less waiting time.
 - Tax compliance - some countries, such as New Zealand, charge road taxes for diesel vehicles based on the vehicle weight when driving on public roads only. Using geofences allows you to track the mileage driven on public roads as opposed to private roads on sites such as farms or logging areas.
 - Monitoring theft - if a vehicle should only remain in a certain area such as on a farm, and it leaves the geofence, it might be being stolen.
 - Location compliance - some parts of a work site, farm or factory may be off-limits to certain vehicles, such as keeping vehicles that carry artificial fertilizers and pesticides away from fields growing organic produce. If a geofence is entered, then a vehicle is outside of compliance and the driver can be notified.

Can you think of other uses for geofences?

Azure Maps, the service you used in the last lesson to visualize GPS data, allows you to define geofences, then test to see if a point is inside or outside of the geofence.

Define a geofence

Geofences are defined using GeoJSON, the same as the points that were added to the map in the previous lesson. In this case, instead of being a `FeatureCollection` of `Point` values, it is a `FeatureCollection` containing a `Polygon`.

```
{  
  "type": "FeatureCollection",  
  "features": [  
    {  
      "type": "Feature",  
      "geometry": {  
        "type": "Polygon",  
        "coordinates": [  
          [  
            [
```

```

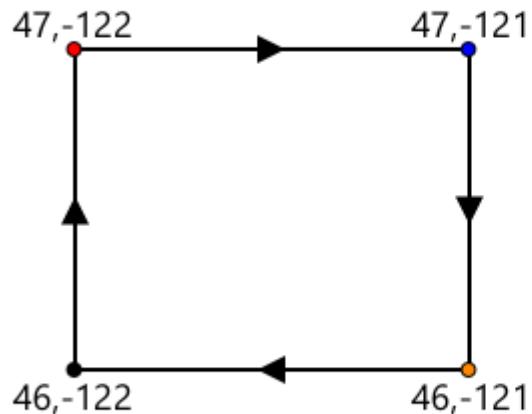
        -122.13393688201903,
        47.63829579223815
    ],
    [
        -122.13389128446579,
        47.63782047131512
    ],
    [
        -122.13240802288054,
        47.63783312249837
    ],
    [
        -122.13238388299942,
        47.63829037035086
    ],
    [
        -122.13393688201903,
        47.63829579223815
    ]
]
],
{
    "properties": {
        "geometryId": "1"
    }
}
]
}

```

Each point on the polygon is defined as a longitude, latitude pair in an array, and these points are in an array that is set as the `coordinates`. In a `Point` in the last lesson, the `coordinates` was an array containing 2 values, latitude and longitude, for a `Polygon` it is an array of arrays containing 2 values, longitude, latitude.

 Remember, GeoJSON uses `longitude, latitude` for points, not `latitude, longitude`

The polygon coordinates array always has 1 more entry than the number of points on the polygon, with the last entry being the same as the first, closing the polygon. For example, for a rectangle there would be 5 points.



```
"coordinates" : [  
  [ -122, 47  
  ],  
  [ -121, 47  
  ],  
  [ -121, 46  
  ],  
  [ -122, 46  
  ],  
  [ -122, 47  
  ],  
]
```

In the image above, there is a rectangle. The polygon coordinates starts at the top-left at 47,-122, then moves right to 47,-121, then down to 46,-121, then right to 46, -122, then back up to the start point at 47, -122. This gives the polygon 5 points - top-left, top-right, bottom-right, bottom-left, then top-left to close it out.

✓ Try creating a GeoJSON polygon around your home or school. Use a tool like [GeoJSON.io](#).

Task - define a geofence

To use a geofence in Azure maps, first it has to be uploaded to your Azure Maps account. Once uploaded, you will get a unique ID that you can use to test a point against the geofence. To upload geofences to Azure Maps, you need to use the maps web API. You can call the Azure Maps web API using a tool called curl.



Curl is a command line tool to make requests against web endpoints

1. If you are using Linux, macOS, or a recent version of Windows 10 you probably have curl installed already. Run the following from your terminal or command line to check:

sh

If you don't see version information for curl, you will need to install it from the [curl downloads page](#).

 If you are experienced with Postman, then you can use that instead if you prefer.

2. Create a GeoJSON file containing a polygon. You will be testing this using your GPS sensor, so create a polygon around your current location. You can either create one manually by editing the GeoJSON example given above, or use a tool like [GeoJSON.io](#).

The GeoJSON will need to contain a `FeatureCollection`, containing a `Feature` with a `geometry` of type `Polygon`.

You **MUST** also add a `properties` element at the same level as the `geometry` element, and this has to contain a `geometryId`:

```
json  
"properties": {  
    "geometryId": "1"  
}
```

If you use [GeoJSON.io](#), then you will manually have to add this item to the empty `properties` element, either after downloading the JSON file, or in the JSON editor in the app.

This `geometryId` must be unique in this file. You can upload multiple geofences as multiple `Features` in the `FeatureCollection` in the same GeoJSON file, as long as each one has a different `geometryId`. Polygons can have the same `geometryId` if they are uploaded from a different file at a different time.

3. Save this file as `geofence.json`, and navigate to where it is saved in your terminal or console.

4. Run the following curl command to create the GeoFence:

```
sh  
curl --request POST 'https://atlas.microsoft.com/mapData/upload?api-version=1.0' --header 'Content-Type: application/json' --include --data @geofence.json
```

Replace `<subscription_key>` in the URL with the API key for your Azure Maps account.

The URL is used to upload map data via the

`https://atlas.microsoft.com/mapData/upload` API. The call includes an `api-version` parameter to specify which Azure Maps API to use, this is to allow the API to change over time but maintain backwards compatibility. The data format that is uploaded is set to `geojson`.

This will run the POST request to the upload API and return a list of response headers which includes a header called `location`

```
content-type: application/json  
location: https://us.atlas.microsoft.com/mapData/operations/1560ced6-3a8  
x-ms-azuremaps-region: West US 2  
x-content-type-options: nosniff  
strict-transport-security: max-age=31536000; includeSubDomains  
x-cache: CONFIG_NOCACHE  
date: Sat, 22 May 2021 21:34:57 GMT  
content-length: 0
```

output

 When calling a web endpoint, you can pass parameters to the call by adding a `?` followed by key value pairs as `key=value` , separating the key value pairs by a `&` .

5. Azure Maps doesn't process this immediately, so you will need to check to see if the upload request has finished by using the URL given in the `location` header. Make a GET request to this location to see the status. You will need to add your subscription key to the end of the `location` URL by adding `&subscription-key=<subscription_key>` to the end, replacing `<subscription_key>` with the API key for your Azure Maps account. Run the following command:

```
sh  
curl --request GET '<location>&subscription-key=<subscription_key>'
```

sh

Replace `<location>` with the value of the `location` header, and `<subscription_key>` with the API key for your Azure Maps account.

6. Check the value of `status` in the response. If it is not `Succeeded` , then wait a minute and try again.

7. Once the status comes back as `Succeeded` , look at the `resourceLocation` from the response. This contains details on the unique ID (known as a UDID) for the GeoJSON object. The UDID is the value after `metadata/` , and not including the `api-version` . For example, if the `resourceLocation` was:

```
{  
  "resourceLocation": "https://us.atlas.microsoft.com/mapData/metadata/7  
}
```

json

Then the UDID would be `7c3776eb-da87-4c52-ae83-caadf980323a` .

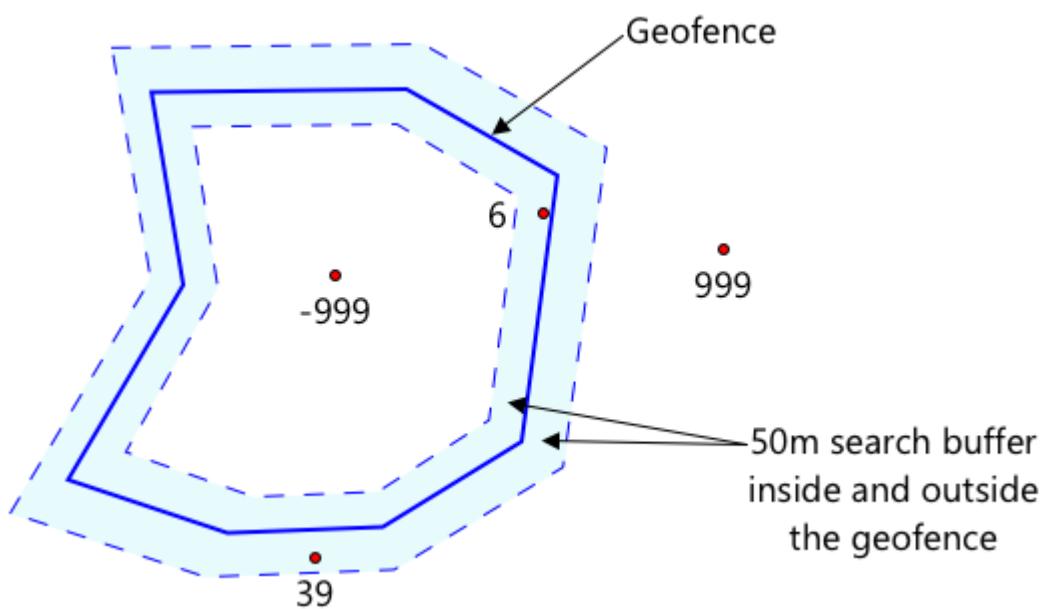
Keep a copy of this UDID as you will need it to test the geofence.

Test points against a geofence

Once the polygon has been uploaded to Azure Maps, you can test a point to see if it is inside or outside the geofence. You do this by making a web API request, passing in the UDID of the geofence, and the latitude and longitude of the point to test.

When you make this request, you can also pass a value called the `searchBuffer` . This tells the Maps API how accurate to be when returning results. The reason for this is GPS is not perfectly accurate, and sometimes locations can be out by meters if not more. The default for the search buffer is 50m, but you can set values from 0m to 500m.

When results are returned from the API call, one of the parts of the result is a `distance` measured to the closest point on the edge of the geofence, with a positive value if the point is outside the geofence, negative if it is inside the geofence. If this distance is less than the search buffer, the actual distance is returned in meters, otherwise the value is 999 or -999. 999 means that the point is outside the geofence by more than the search buffer, -999 means it is inside the geofence by more than the search buffer.



In the image above, the geofence has a 50m search buffer.

- A point in the center of the geofence, well inside the search buffer has a distance of **-999**
- A point well outside the search buffer has a distance of **999**
- A point inside the geofence and inside the search buffer, 6m from the geofence, has a distance of **6m**

- A point outside the geofence and inside the search buffer, 39m from the geofence, has a distance of **39m**

It is important to know the distance to the edge of the geofence, and combine this with other information such as other GPS readings, speed and road data when making decisions based off a vehicle location.

For example, imagine GPS readings showing a vehicle was driving along a road that ends up running next to a geofence. If a single GPS value is inaccurate and places the vehicle inside the geofence, despite there being no vehicular access, then it can be ignored.



In the above image, there is a geofence over part of the Microsoft campus. The red line shows a truck driving along the 520, with circles to show the GPS readings. Most of these are accurate and along the 520, with one inaccurate reading inside the geofence. There is no way that reading can be correct - there are no roads for the truck to suddenly divert from the 520 onto campus, then back onto the 520. The code that checks this geofence will need to take the previous readings into consideration before acting on the results of the geofence test.

- What additional data would you need to check to see if a GPS reading could be considered correct?

Task - test points against a geofence

1. Start by building the URL for the web API query. The format is:

```
output
https://atlas.microsoft.com/spatial/geofence/json?api-version=1.0&device
```

Replace <subscription_key> with the API key for your Azure Maps account.

Replace <UDID> with the UDID of the geofence from the previous task.

Replace <lat> and <lon> with the latitude and longitude that you want to test.

This URL uses the `https://atlas.microsoft.com/spatial/geofence/json` API to query a geofence defined using GeoJSON. It targets the `1.0` api version. The `deviceId` parameter is required and should be the name of the device the latitude and longitude comes from.

The default search buffer is 50m, and you can change this by passing an additional parameter of `searchBuffer=<distance>`, setting `<distance>` to the search buffer distance in meters, from 0 to 500.

2. Use curl to make a GET request to this URL:

```
sh
curl --request GET <URL>
```

 If you get a response code of `BadRequest`, with an error of:

```
output
Invalid GeoJSON: All feature properties should contain a geometryI
```

then your GeoJSON is missing the `properties` section with the `geometryId`. You will need to fix up your GeoJSON, then repeat the steps above to re-upload and get a new UDID.

3. The response will contain a list of `geometries`, one for each polygon defined in the GeoJSON used to create the geofence. Each geometry has 3 fields of interest, `distance`, `nearestLat` and `nearestLon`.

```
output
{
  "geometries": [
    {
      "deviceId": "gps-sensor",
      "udId": "1ffb2047-6757-8c29-2c3d-da44cec55ff9",
      "geometryId": "1",
      "distance": 999.0,
      "nearestLat": 47.645875,
      "nearestLon": -122.142713
    }
  ]
}
```

```
        },
        "nearestLat": null,
        "nearestLon": null,
        "distance": null,
        "expiredGeofenceGeometryId": [],
        "invalidPeriodGeofenceGeometryId": []
    }
}
```

- `nearestLat` and `nearestLon` are the latitude and longitude of a point on the edge of the geofence that is closest to the location being tested.
- `distance` is the distance from the location being tested to the closest point on the edge of the geofence. Negative numbers mean inside the geofence, positive outside. This value will be less than 50 (the default search buffer), or 999.

4. Repeat this multiple times with locations inside and outside the geofence.

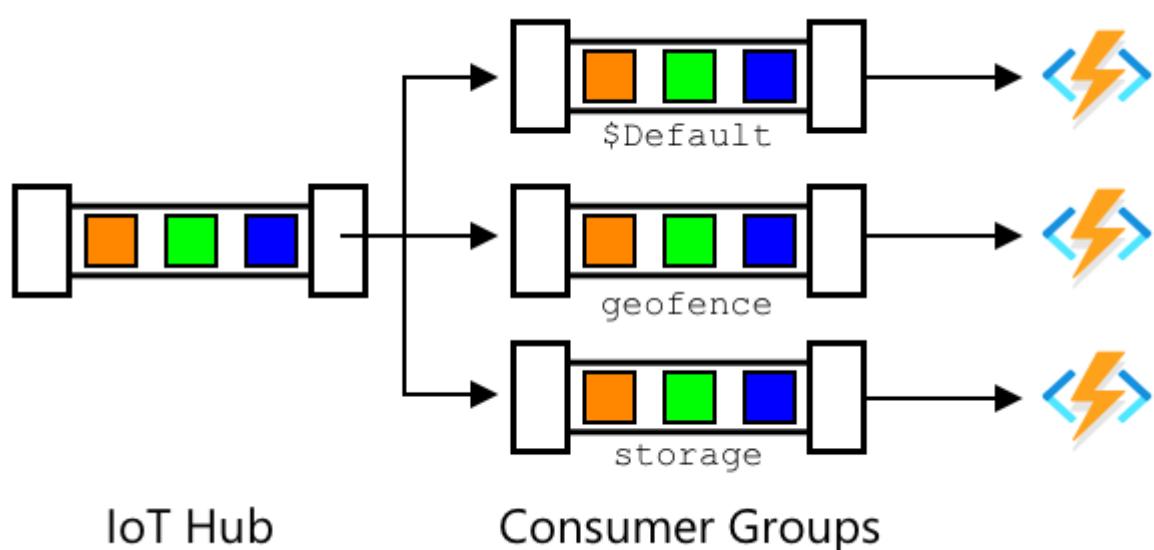
Use geofences from serverless code

You can now add a new trigger to your Functions app to test the IoT Hub GPS event data against the geofence.

Consumer groups

As you will remember from previous lessons, the IoT Hub will allow you to replay events that have been received by the hub but not processed. But what would happen if multiple triggers connected? How will it know which one has processed which events.

The answer is it can't! Instead you can define multiple separate connections to read off events, and each one can manage the replay of unread messages. These are called *consumer groups*. When you connect to the endpoint, you can specify which consumer group you want to connect to. Each component of your application will connect to a different consumer group



In theory up to 5 applications can connect to each consumer group, and they will all receive messages when they arrive. It's best practice to have only one application access each consumer group to avoid duplicate message processing, and ensure when restarting all queued messages are processed correctly. For example, if you launched your Functions app locally as well as running it in the cloud, they would both process messages, leading to duplicate blobs stored in the storage account.

If you review the `function.json` file for the IoT Hub trigger you created in an earlier lesson, you will see the consumer group in the event hub trigger binding section:

```
json
"consumerGroup": "$Default"
```

When you create an IoT Hub, you get the `$Default` consumer group created by default. If you want to add an additional trigger, you can add this using a new consumer group.

 In this lesson, you will use a different function to test the geofence to the one used to store the GPS data. This is to show how to use consumer groups and separate the code to make it easier to read and understand. In a production application there are many ways you might architect this - putting both on one function, using a trigger on the storage account to run a function to check the geofence, or using multiple functions. There is no 'right way', it depends on the rest of your application and your needs.

Task - create a new consumer group

1. Run the following command to create a new consumer group called `geofence` for your IoT Hub:

sh

```
az iot hub consumer-group create --name geofence \  
--hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub.

2. If you want to see all the consumer groups for an IoT Hub, run the following command:

sh

```
az iot hub consumer-group list --output table \  
--hub-name <hub_name>
```

Replace `<hub_name>` with the name you used for your IoT Hub. This will list all the consumer groups.

output

Name	ResourceGroup
-----	-----
\$Default	gps-sensor
geofence	gps-sensor

Task - create a new IoT Hub trigger

1. Add a new IoT Hub event trigger to your `gps-trigger` function app that you created in an earlier lesson. Call this function `geofence-trigger`.

 You can refer to [the instructions for creating an IoT Hub event trigger from project 2, lesson 5 if needed.](#)

2. Configure the IoT Hub connection string in the `function.json` file. The `local.settings.json` is shared between all triggers in the Function App.
3. Update the value of the `consumerGroup` in the `function.json` file to reference the new `geofence` consumer group:

json

```
"consumerGroup": "geofence"
```

4. You will need to use the subscription key for your Azure Maps account in this trigger, so add a new entry to the `local.settings.json` file called `MAPS_KEY`.
5. Run the Functions App to ensure it is connecting and processing messages. The `iot-hub-trigger` from the earlier lesson will also run and upload blobs to storage.

To avoid duplicate GPS readings in blob storage, you can stop the Functions App you have running in the cloud. To do this, use the following command:

```
sh  
az functionapp stop --resource-group gps-sensor \  
--name <functions_app_name>
```

Replace `<functions_app_name>` with the name you used for your Functions App.

You can restart it later with the following command:

```
sh  
az functionapp start --resource-group gps-sensor \  
--name <functions_app_name>
```

Replace `<functions_app_name>` with the name you used for your Functions App.

Task - test the geofence from the trigger

Earlier in this lesson you used curl to query a geofence to see if a point was located inside or outside. You can make a similar web request from inside your trigger.

1. To query the geofence, you need its UDID. Add a new entry to the `local.settings.json` file called `GEOFENCE_UDID` with this value.
2. Open the `__init__.py` file from the new `geofence-trigger` trigger.
3. Add the following import to the top of the file:

```
python  
import json  
import os  
import requests
```

The `requests` package allows you to make web API calls. Azure Maps doesn't have a Python SDK, you need to make web API calls to use it from Python code.

4. Add the following 2 lines to the start of the `main` method to get the Maps subscription key:

python

```
maps_key = os.environ['MAPS_KEY']
geofence_udid = os.environ['GEOFENCE_UDID']
```

5. Inside the `for event in events` loop, add the following to get the latitude and longitude from each event:

python

```
event_body = json.loads(event.get_body().decode('utf-8'))
lat = event_body['gps']['lat']
lon = event_body['gps']['lon']
```

This code converts the JSON from the event body to a dictionary, then extracts the `lat` and `lon` from the `gps` field.

6. When using `requests`, rather than building up a long URL as you did with curl, you can use just the URL part and pass the parameters as a dictionary. Add the following code to define the URL to call and configure the parameters:

python

```
url = 'https://atlas.microsoft.com/spatial/geofence/json'

params = {
    'api-version': '1.0',
    'deviceId': 'gps-sensor',
    'subscription-key': maps_key,
    'udid' : geofence_udid,
    'lat' : lat,
    'lon' : lon
}
```

The items in the `params` dictionary will match the key value pairs you used when calling the web API via curl.

7. Add the following line of code to call the web API:

python

```
response = requests.get(url, params=params)
```

This calls the URL with the parameters, and gets back a response object.

8. Add the following code below this:

python

```
distance = response_body['geometries'][0]['distance']

if distance == 999:
    logging.info('Point is outside geofence')
elif distance > 0:
    logging.info(f'Point is just outside geofence by a distance of {distance}')
elif distance == -999:
    logging.info(f'Point is inside geofence')
else:
    logging.info(f'Point is just inside geofence by a distance of {distance}'
```

This code assumes 1 geometry, and extracts the distance from that single geometry. It then logs different messages based off the distance.

9. Run this code. You will see in the logging output if the GPS coordinates are inside or outside the geofence, with a distance if the point is within 50m. Try this code with different geofences based off the location of your GPS sensor, try moving the sensor (for example tethered to WiFi from a mobile phone, or with different coordinates on the virtual IoT device) to see this change.

10. When you are ready, deploy this code to your Functions app in the cloud. Don't forget to deploy the new Application Settings.

⚠ You can refer to [the instructions for uploading Application Settings from project 2, lesson 5 if needed.](#)

⚠ You can refer to [the instructions for deploying your Functions app from project 2, lesson 5 if needed.](#)

💡 You can find this code in the [code/functions](#) folder.



In this lesson you added one geofence using a GeoJSON file with a single polygon. You can upload multiple polygons at the same time, as long as they have different `geometryId` values in the `properties` section.

Try uploading a GeoJSON file with multiple polygons and adjust your code to find which polygon the GPS coordinates are closest to or in.

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read more on geofences and some of their use cases on the [Geofencing page on Wikipedia](#).
- Read more on Azure Maps geofencing API on the [Microsoft Azure Maps Spatial - Get Geofence documentation](#).
- Read more on consumer groups in the [Features and terminology in Azure Event Hubs - Event consumers documentation on Microsoft docs](#)

Assignment

[Send notifications using Twilio](#)

Train a fruit quality detector

Add a sketchnote if possible/appropriate

This video gives an overview of the Azure Custom Vision service, a service that will be covered in this lesson.



🎥 Click the image above to watch a video

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

The recent rise in Artificial Intelligence (AI) and Machine Learning (ML) is providing a wide range of capabilities to todays developers. ML models can be trained to recognize different things in images, including unripe fruit, and this can be used in IoT devices to help sort produce either as it is being harvested, or during processing in factories or warehouses.

In this lesson you will learn about image classification - using ML models to distinguish between images of different things. You will learn how to train an image classifier to distinguish between fruit that is good, and fruit that is bad, either under or over ripe, bruised, or rotten.

In this lesson we'll cover:

- [Using AI and ML to sort food](#)
- [Image classification via Machine Learning](#)
- [Train an image classifier](#)

- [Test your image classifier](#)
- [Retrain your image classifier](#)

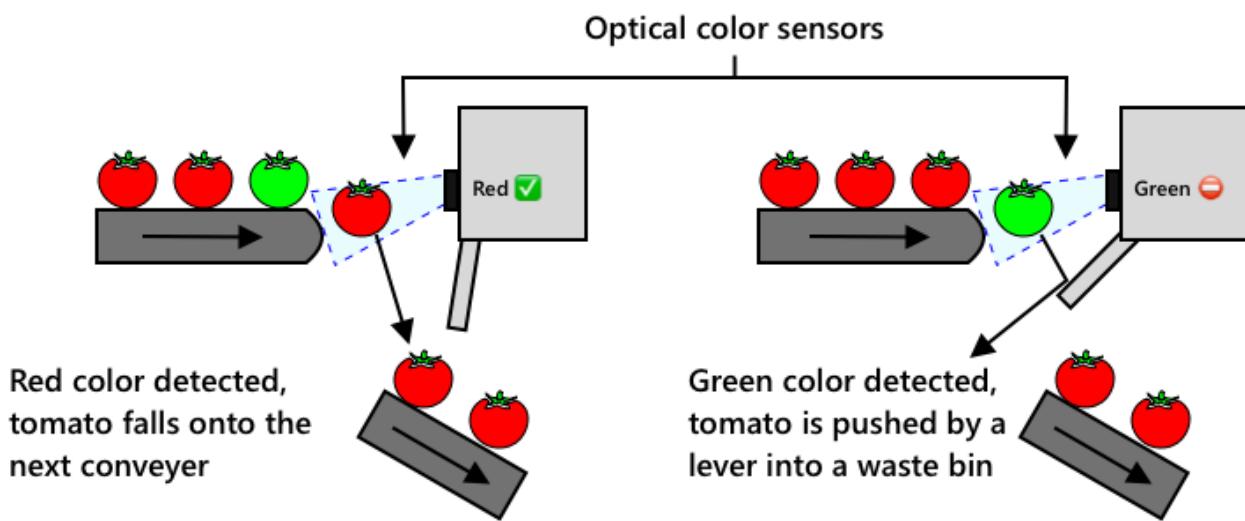
Using AI and ML to sort food

Feeding the global population is hard, especially at a price that makes food affordable for all. One of the largest costs is labor, so farmers are increasingly turning to automation and tools like IoT to reduce their labor costs. Harvesting by hand is labor intensive (and often backbreaking work), and is being replaced by machinery, especially in richer nations. Despite the savings in cost of using machinery to harvest, there is a downside - the ability to sort food as it is being harvested.

Not all crops ripen evenly. Tomatoes, for example, can still have some green fruits on the vine when the majority is ready for harvest. Although it is a waste to harvest these early, it is cheaper and easier for the farmer to harvest everything using machinery and dispose of the unripe produce later.

- Have a look at different fruits or vegetables, either growing near you in farms or in your garden, or in shops, Are they all the same ripeness, or do you see variation?

The rise of automated harvesting moved the sorting of produce from the harvest to the factory. Food would travel on long conveyer belts with teams of people picking over the produce removing anything that wasn't up to the required quality standard. Harvesting was cheaper thanks to machinery, but there was still a cost to manually sort food.



If a red tomato is detected it continues its journey uninterrupted. If a green tomato is detected it is flicked into a waste bin by a lever. tomato by parkjisun from the Noun Project - from the [Noun Project](#)

The next evolution was to use machines to sort, either built into the harvester, or in the processing plants. The first generation of these machines used optical sensors to detect colors, controlling

actuators to push green tomatoes into a waste bin using levers or puffs of air, leaving red tomatoes to continue on a network of conveyor belts.

The video below shows one of these machines in action.



🎥 Click the image above to watch a video

In this video, as tomatoes fall from one conveyer belt to another, green tomatoes are detected and flicked into a bin using levers.

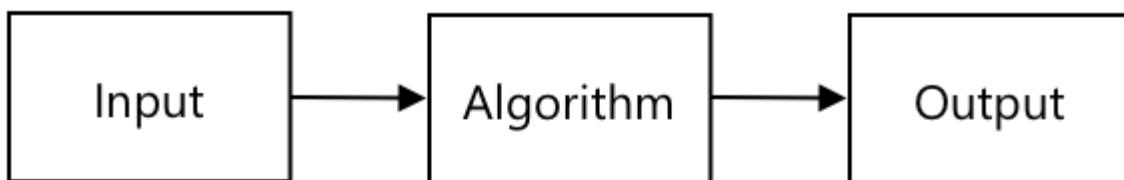
What conditions would you need in a factory or in a field for these optical sensors to work correctly?

The latest evolutions of these sorting machines take advantage of AI and ML, using models trained to distinguish good produce from bad, not just by obvious color differences such as green tomatoes vs red, but by more subtle differences in appearance that can indicate disease or bruising.

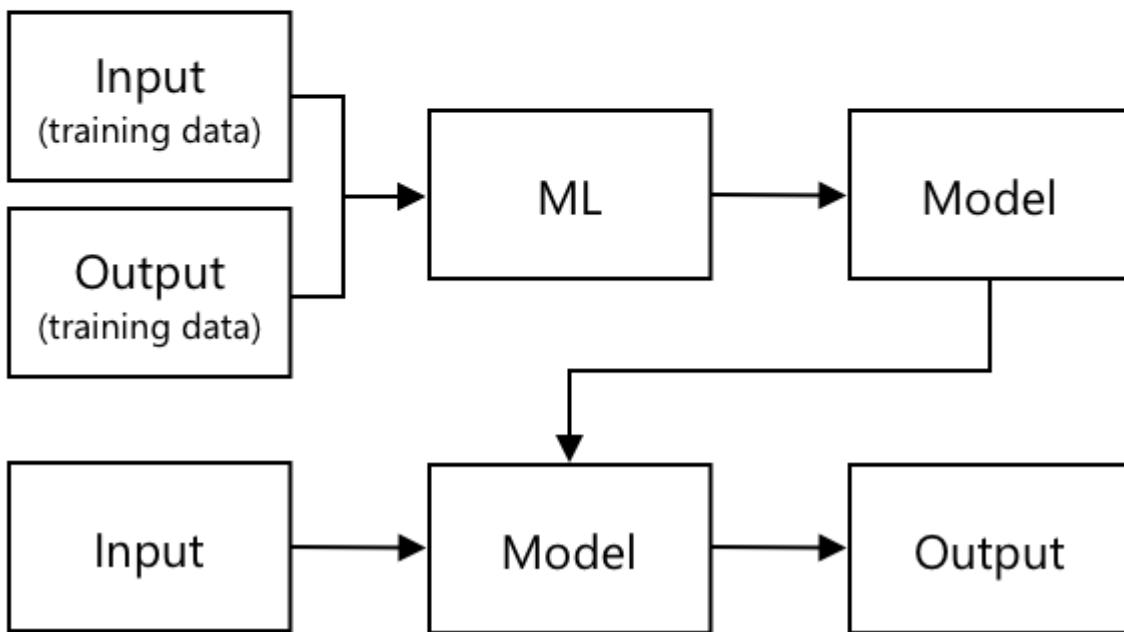
Image classification via Machine Learning

Traditional programming is where you take data, apply an algorithm to the data, and get output. For example, in the last project you took GPS coordinates and a geofence, applied an algorithm that was provided by Azure Maps, and got back a result of if the point was inside or outside the geofence. You input more data, you get more output.

Traditional application development



Machine Learning development

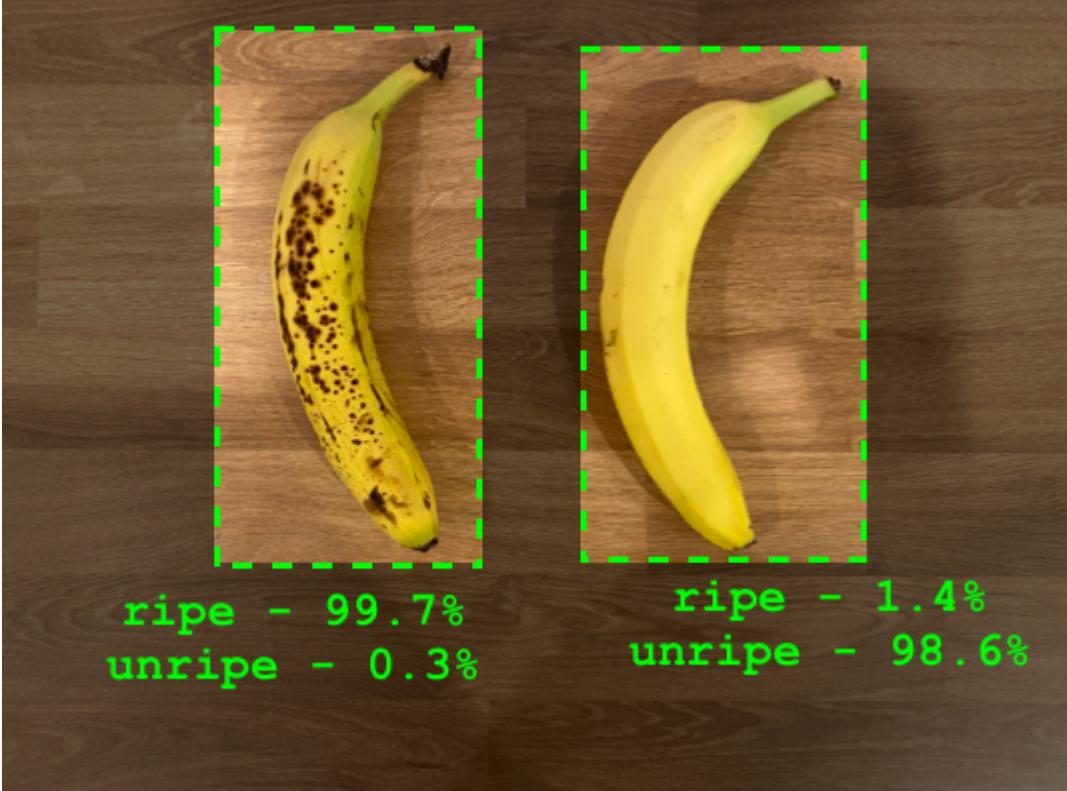


Machine learning turns this around - you start with data and known outputs, and the machine learning tools work out the algorithm. You can then take that algorithm, called a *machine learning model*, and input new data and get new output.

🎓 The process of a machine learning tool generating a model is called training. The inputs and known outputs are called training data.

For example, you could give a model millions of pictures of unripe bananas as input training data, with the training output set as `unripe` , and millions of ripe banana pictures as training data with the output set as `ripe` . The ML tools will then generate a model. You then give this model a new picture of a banana and it will predict if the new picture is a ripe or an unripe banana.

🎓 The results of ML models are called predictions



ML models don't give a binary answer, instead they give probabilities. For example, a model may be given a picture of a banana and predict `ripe` at 99.7% and `unripe` at 0.3%. Your code would then pick the best prediction and decide the banana is ripe.

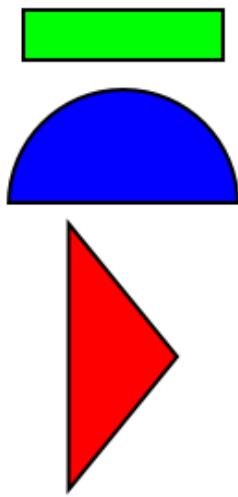
The ML model used to detect images like this is called an *image classifier* - it is given labelled images, and then classifies new images based off these labels.

Train an image classifier

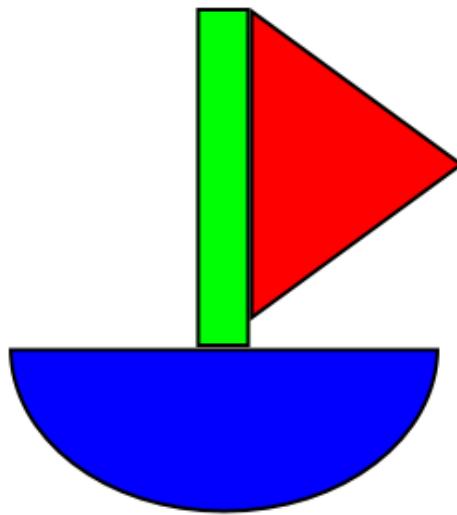
To successfully train an image classifier you need millions of images. As it turns out, once you have an image classifier trained on millions or billions of assorted images, you can re-use it and re-train it using a small set of images and get great results, using a process called *transfer learning*.

🎓 Transfer learning is where you transfer the learning from an existing ML model to a new model based off new data.

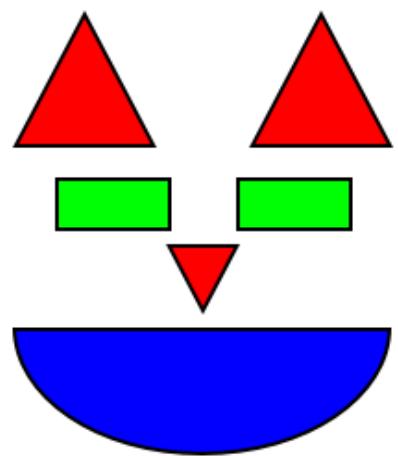
Once an image classifier has been trained for a wide variety of images, its internals are great at recognizing shapes, colors and patterns. Transfer learning allows the model to take what it has already learned in recognizing image parts, and use that to recognize new images.



Shapes



Boat



Cat

You can think of this as a bit like children's shape books, where once you can recognize a semi-circle, a rectangle and a triangle, you can recognize a sailboat or a cat depending on the configuration of these shapes. The image classifier can recognize the shapes, and the transfer learning teaches it what combination makes a boat or a cat - or a ripe banana.

There are a wide range of tools that can help you do this, including cloud-based services that can help you train your model, then use it via web APIs.

 Training these models takes a lot of computer power, usually via Graphics Processing Units, or GPUs. The same specialized hardware that makes games on your Xbox look amazing can also be used to train machine learning models. By using the cloud you can rent time on powerful computers with GPUs to train these models, getting access to the computing power you need, just for the time you need it.

Custom Vision

Custom Vision is a cloud based tool for training image classifiers. It allows you to train a classifier using only a small number of images. You can upload images through a web portal, web API or an SDK, giving each image a *tag* that has the classification of that image. You then train the model, and test it out to see how well it performs. Once you are happy with the model, you can publish versions of it that can be accessed through a web API or an SDK.



Custom Vision

💡 You can train a custom vision model with as little as 5 images per classification, but more is better. You can get better results with at least 30 images.

Custom Vision is part of a range of AI tools from Microsoft called Cognitive Services. These are AI tools that can be used either without any training, or with a small amount of training. They include speech recognition and translation, language understanding and image analysis. These are available with a free tier as services in Azure.

💡 The free tier is more than enough to create a model, train it, then use it for development work. You can read about the limits of the free tier on the [Custom Vision Limits and quotas page on Microsoft docs](#).

Task - create a cognitive services resource

To use Custom Vision, you first need to create two cognitive services resources in Azure using the Azure CLI, one for Custom Vision training and one for Custom Vision prediction.

1. Create a Resource Group for this project called `fruit-quality-detector`
2. Use the following command to create a free Custom Vision training resource:

```
sh
az cognitiveservices account create --name fruit-quality-detector-traini
--resource-group fruit-quality-detec
--kind CustomVision.Training \
--sku F0 \
--yes \
--location <location>
```

Replace `<location>` with the location you used when creating the Resource Group.

This will create a Custom Vision training resource in your Resource Group. It will be called `fruit-quality-detector-training` and use the `F0` sku, which is the free tier. The `--yes` option means you agree to the terms and conditions of the cognitive services.

3. Use the following command to create a free Custom Vision prediction resource:

```
sh
az cognitiveservices account create --name fruit-quality-detector-predic
                                     --resource-group fruit-quality-detec
                                     --kind CustomVision.Prediction \
                                     --sku F0 \
                                     --yes \
                                     --location <location>
```

Replace `<location>` with the location you used when creating the Resource Group.

This will create a Custom Vision prediction resource in your Resource Group. It will be called `fruit-quality-detector-prediction` and use the `F0` sku, which is the free tier. The `--yes` option means you agree to the terms and conditions of the cognitive services.

Task - create an image classifier project

1. Launch the Custom Vision portal at CustomVision.ai, and sign in with the Microsoft account you used for your Azure account.
2. Follow the [Create a new Project section of the Build a classifier quickstart on the Microsoft docs](#) to create a new Custom Vision project. The UI may change and these docs are always the most up to date reference.

Call your project `fruit-quality-detector`.

When you create your project, make sure to use the `fruit-quality-detector-training` resource you created earlier. Use a *Classification* project type, a *Multiclass* classification type, and the *Food* domain.

Create new project



Name*

Description

Resource

[create new](#)[Manage Resource Permissions](#)

Project Types

- Classification
- Object Detection

Classification Types

- Multilabel (Multiple tags per image)
- Multiclass (Single tag per image)

Domains:

- General [A2]
- General [A1]
- General
- Food
- Landmarks
- Retail
- General (compact) [S1]
- General (compact)
- Food (compact)
- Landmarks (compact)
- Retail (compact)

Task - train your image classifier project

To train an image classifier, you will need multiple pictures of fruit, both good and bad quality to tag as good and bad, such as an ripe and an overripe banana.

👉 These classifiers can classify images of anything, so if you don't have fruit to hand or differing quality, you can use two different types of fruit, or cats and dogs!

Ideally each picture should be just the fruit, with either a consistent background, or a wide variety of backgrounds. Ensure there's nothing in the background that is specific to ripe vs unripe fruit.

👉 It's important not to have specific backgrounds, or specific items that are not related to the thing being classified for each tag, otherwise the classifier may just classify based on the background. There was a classifier for skin cancer that was trained on moles both normal and cancerous, and the cancerous ones all had rulers against them to measure the size. It turned out the classifier was almost 100% accurate at identifying rulers in pictures, not cancerous moles.

Image classifiers run at very low resolution. For example Custom Vision can take training and prediction images up to 10240x10240, but trains and runs the model on images at 227x227. Larger images are shrunk to this size, so ensure the thing you are classifying takes up a large part of the image otherwise it may be too small in the smaller image used by the classifier.

1. Gather pictures for your classifier. You will need at least 5 pictures for each label to train the classifier, but the more the better. You will also need a few additional images to test the classifier. These images should all be different images of the same thing. For example:

- Using 2 ripe bananas, take some pictures of each one from a few different angles, taking at least 7 pictures (5 to train, 2 to test), but ideally more.



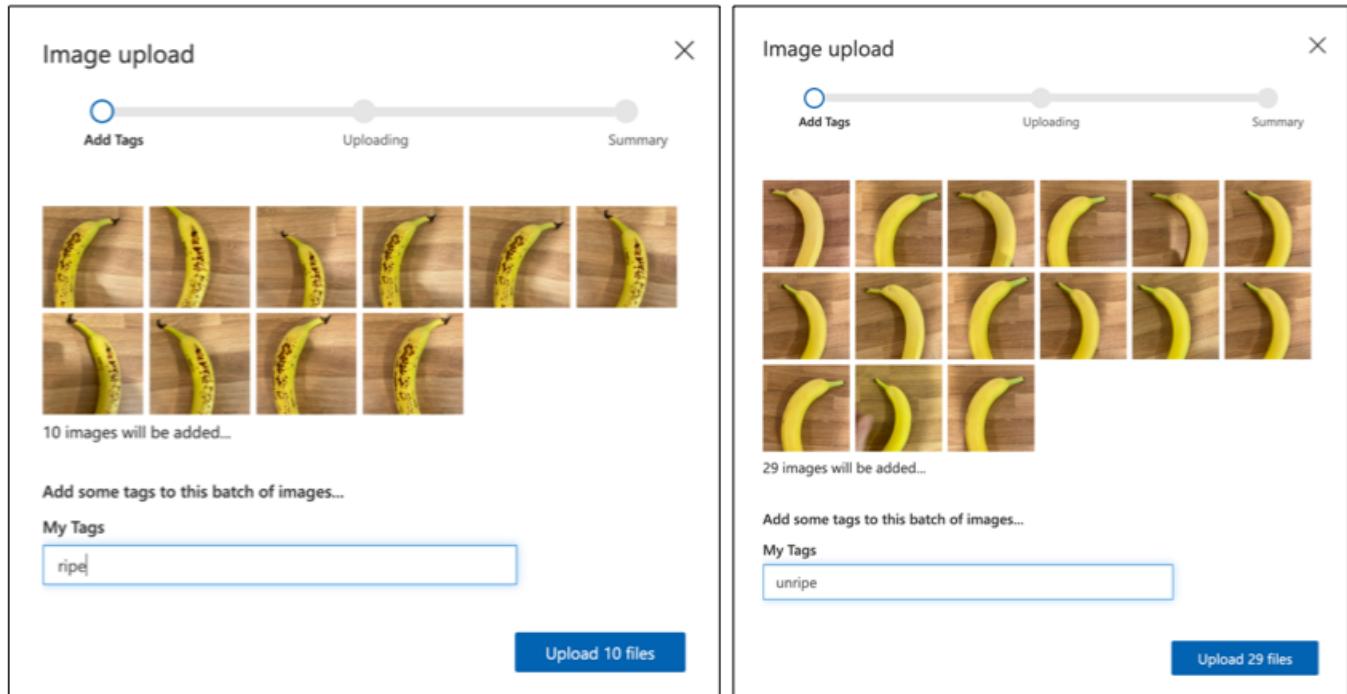
- Repeat the same process using 2 unripe bananas

You should have at least 10 training images, with at least 5 ripe and 5 unripe, and 4 testing images, 2 ripe, 2 unripe. Your images should be png or jpeg, smaller than 6MB. If you create them with an iPhone for example they may be high-resolution HEIC images, so will need to be

converted and possibly shrunk. The more images the better, and you should have a similar number of ripe and unripe.

If you don't have both ripe and unripe fruit, you can use different fruits, or any two objects you have available. You can also find some example images in the [images](#) folder of ripe and unripe bananas that you can use.

2. Follow the [Upload and tag images](#) section of the [Build a classifier quickstart](#) on the Microsoft docs to upload your training images. Tag the ripe fruit as `ripe`, and the unripe fruit as `unripe`.



3. Follow the [Train the classifier](#) section of the [Build a classifier quickstart](#) on the Microsoft docs to train the image classifier on your uploaded images.

You will be given a choice of training type. Select **Quick Training**.

The classifier will then train. It will take a few minutes for the training to complete.

👉 If you decide to eat your fruit whilst the classifier is training, make sure you have enough images to test with first!

Test your image classifier

Once your classifier is trained, you can test it by giving it a new image to classify.

Task - test your image classifier

1. Follow the [Test your model documentation on the Microsoft docs](#) to test your image classifier. Use the testing images you created earlier, not any of the images you used for training.

Image URL

Enter Image URL →

or

Browse local files

File formats accepted: [jpg](#), [png](#), [bmp](#)
File size should not exceed: [4mb](#)

Using model trained in

Iteration

Iteration 3 ▼

Predictions

Tag	Probability
unripe	98.8%
ripe	1.1%



2. Try all the testing images you have access to and observe the probabilities.

Retrain your image classifier

When you test your classifier, it may not give the results you expect. Image classifiers use machine learning to make predictions about what is in an image, based on probabilities that particular features of an image mean that it matches a particular label. It doesn't understand what is in the image - it doesn't know what a banana is or understand what makes a banana a banana instead of a boat. You can improve your classifier by retraining it with images it gets wrong.

Every time you make a prediction using the quick test option, the image and results are stored. You can use these images to retrain your model.

Task - retrain your image classifier

1. Follow the [Use the predicted image for training documentation on the Microsoft docs](#) to retrain your model, using the correct tag for each image.
2. Once your model has been retrained, test on new images.

Challenge

What do you think would happen if you used a picture of a strawberry with a model trained on bananas, or a picture of an inflatable banana, or a person in a banana suit, or even a yellow cartoon character like someone from the Simpsons?

Try it out and see what the predictions are. You can find images to try with using [Bing Image search](#).

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- When you trained your classifier, you would have seen values for *Precision*, *Recall*, and *AP* that rate the model that was created. Read up on what these values are using [the Evaluate the classifier section of the Build a classifier quickstart on the Microsoft docs](#)
- Read up on how to improve your classifier from the [How to improve your Custom Vision model on the Microsoft docs](#)

Assignment

[Train your classifier for multiple fruits and vegetables](#)

Check fruit quality from an IoT device

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

Introduction

In the last lesson you learned about image classifiers, and how to train them to detect good and bad fruit. To use this image classifier in an IoT application, you need to be able to capture an image using some kind of camera, and send this image to the cloud to be classified.

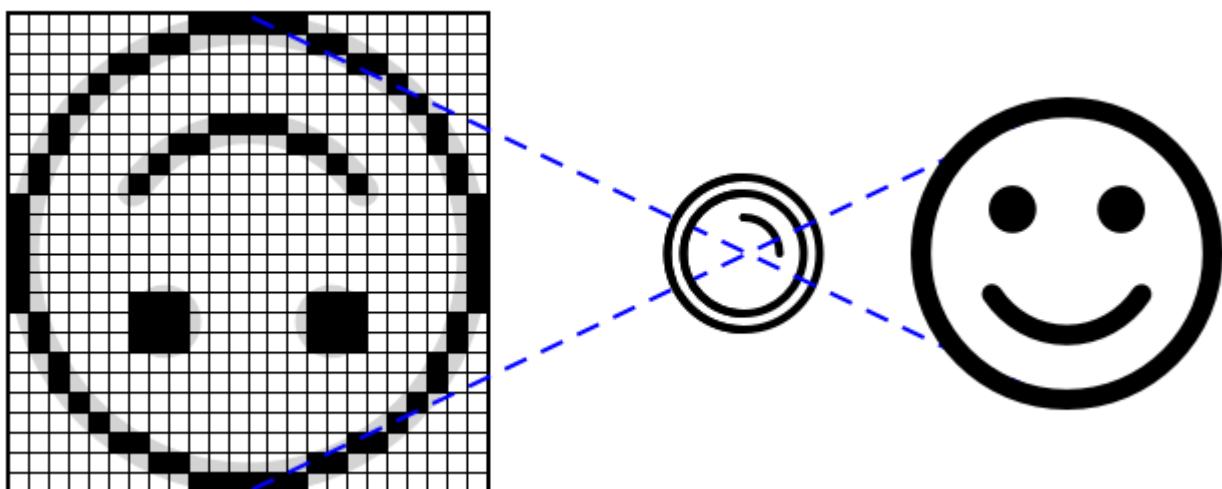
In this lesson you will learn about camera sensors, and how to use them with an IoT device to capture an image. You will also learn how to call the image classifier from your IoT device.

In this lesson we'll cover:

- [Camera sensors](#)
- [Capture an image using an IoT device](#)
- [Publish your image classifier](#)
- [Classify images from your IoT device](#)
- [Improve the model](#)

Camera sensors

Camera sensors, as the name suggests, are cameras that you can connect to your IoT device. They can take still images, or capture streaming video. Some will return raw image data, others will compress the image data into an image file such as a JPEG or PNG. Usually the cameras that work with IoT devices are much smaller and lower resolution than what you might be used to, but you can get high resolution cameras that will rival top end phones. You can get all manner of interchangeable lenses, multiple camera setups, infra-red thermal cameras, or UV cameras.



Most camera sensors use image sensors where each pixel is a photodiode. A lens focuses the image onto the image sensor, and thousands or millions of photodiodes detect the light falling on each one, and record that as pixel data.

 Lenses invert images, the camera sensor then flips the image back the right way round.

This is the same in your eyes - what you see is detected upside down on the back of your eye and your brain corrects it.

 The image sensor is known as an Active-Pixel Sensor (APS), and the most popular type of APS is a complementary metal-oxide semiconductor sensor, or CMOS. You may have heard the term CMOS sensor used for camera sensors.

Camera sensors are digital sensors, sending image data as digital data, usually with the help of a library that provides the communication. Cameras connect using protocols like SPI to allow them to send large quantities of data - images are substantially larger than single numbers from a sensor such as a temperature sensor.

 What are the limitations around image size with IoT devices? Think about the constraints especially on microcontroller hardware.

Capture an image using an IoT device

You can use your IoT device to capture and image to be classified.

Task - capture an image using an IoT device

Work through the relevant guide to capture an image using your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Publish your image classifier

You trained your image classifier in the last lesson. Before you can use it from your IoT device, you need to publish the model.

Model iterations

When your model was training in the last lesson, you may notice that the **Performance** tab shows iterations on the side. When you first trained the model you would have seen *Iteration 1* in training. When you improved the model using the prediction images, you would have seen *Iteration 2* in training.

Every time you train the model, you get a new iteration. This is a way to keep track of the different versions of your model trained on different data sets. When you do a **Quick Test**, there is a drop-down you can use to select the iteration, so you can compare the results across multiple iterations.

When you are happy with an iteration, you can publish it to make it available to be used from external applications. This way you can have a published version that is used by your devices, then work on a new version over multiple iterations, then publish that once you are happy with it.

Task - publish an iteration

Iterations are published from the Custom Vision portal.

1. Launch the Custom Vision portal at CustomVision.ai and sign in if you don't have it open already.
2. Select the **Performance** tab from the options at the top
3. Select the latest iteration from the *Iterations* list on the side
4. Select the **Publish** button for the iteration

The screenshot shows the Microsoft Custom Vision Studio interface. At the top, there are tabs for 'Training Images', 'Performance' (which is selected), 'Predictions', 'Train', and 'Quick Test'. On the left, there's a sidebar with 'Iterations' and a probability threshold slider set to 50%. The main area shows 'Iteration 2' was trained 2 hours ago with a Food domain. A red box highlights the 'Publish' button. Below it, it says 'Iteration 2' finished training on 6/3/2021 at 5:11:45 PM using the Food domain. It also shows 'Iteration 1' trained 1 day ago with the Food domain. The 'Performance' section contains three donut charts: Precision (100.0%, purple), Recall (100.0%, blue), and AP (100.0%, green).

5. In the *Publish Model* dialog, set the *Prediction* resource to the `fruit-quality-detector-prediction` resource you created in the last lesson. Leave the name as `Iteration2`, and select the **Publish** button.
6. Once published, select the **Prediction URL** button. This will show details of the prediction API, and you will need these to call the model from your IoT device. The lower section is labelled *If you have an image file*, and this is the details you want. Take a copy of the URL that is shown which will be something like:

output

```
https://<location>.api.cognitive.microsoft.com/customvision/v3.0/Predict
```

Where `<location>` will be the location you used when creating your custom vision resource, and `<id>` will be a long ID made up of letters and numbers.

Also take a copy of the *Prediction-Key* value. This is a secure key that you have to pass when you call the model. Only applications that pass this key are allowed to use the model, any other applications are rejected.

How to use the Prediction API

X

If you have an image URL:

```
https://westus2.api.cognitive.microsoft.com/customvision/v3.0/Prediction/
```

```
Set Prediction-Key Header to : [REDACTED]
```

```
Set Content-Type Header to : application/json
```

```
Set Body to : {"Url": "https://example.com/image.png"}
```

If you have an image file:

```
https://westus2.api.cognitive.microsoft.com/customvision/v3.0/Prediction/
```

```
Set Prediction-Key Header to : [REDACTED]
```

```
Set Content-Type Header to : application/octet-stream
```

```
Set Body to : <image file>
```

Got it!

- When a new iteration is published, it will have a different name. How do you think you would change the iteration an IoT device is using?

Classify images from your IoT device

You can now use these connection details to call the image classifier from your IoT device.

Task - classify images from your IoT device

Work through the relevant guide to classify images using your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi/Virtual IoT device](#)

Improve the model

You may find that the results you get when using the camera connected to your IoT device don't match what you would expect. The predictions are not always as accurate as using images uploaded from your computer. This is because the model was trained on different data to what is being used for predictions.

To get the best results for an image classifier, you want to train the model with images that are as similar to the images used for predictions as possible. If you used your phone camera to capture images for training, for example, the image quality, sharpness, and color will be different to a camera connected to an IoT device.



In the image above, the banana picture on the left was taken using a Raspberry Pi Camera, the one on the right was taken of the same banana in the same location using an iPhone. There is a noticeable difference in quality - the iPhone picture is sharper, with brighter colors and more contrast.

- What else might cause the images captured by your IoT device to have incorrect predictions? Think about the environment an IoT device might be used in, what factors can affect the image being captured?

To improve the model, you can retrain it using the images captured from the IoT device.

Task - improve the model

1. Classify multiple images of both ripe and unripe fruit using your IoT device.
2. In the Custom Vision portal, retrain the model using the images on the *Predictions* tab.

⚠ You can refer to [the instructions for retraining your classifier in lesson 1 if needed.](#)

3. If your images look very different to the original ones used to train, you can delete all the original images by selecting them in the *Training Images* tab and selecting the **Delete** button. To select an image, move your cursor over it and a tick will appear, select that tick to select or deselect the image.
 4. Train a new iteration of the model and publish it using the steps above.
 5. Update the endpoint URL in your code, and re-run the app.
 6. Repeat these steps until you are happy with the results of the predictions.
-

Challenge

How much does image resolution or lighting affect the prediction?

Try changing the resolution of the images in your device code and see if it makes a difference to the quality of the images. Also try changing lighting.

If you were to create a production device to sell to farms or factories, how would you ensure it gives consistent results all the time?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

You trained your custom vision model using the portal. This relies on having images available - and in the real world you may not be able to get training data that matches what the camera on your device captures. You can work round this by training directly from your device using the training API, to train a model using images captured from your IoT device.

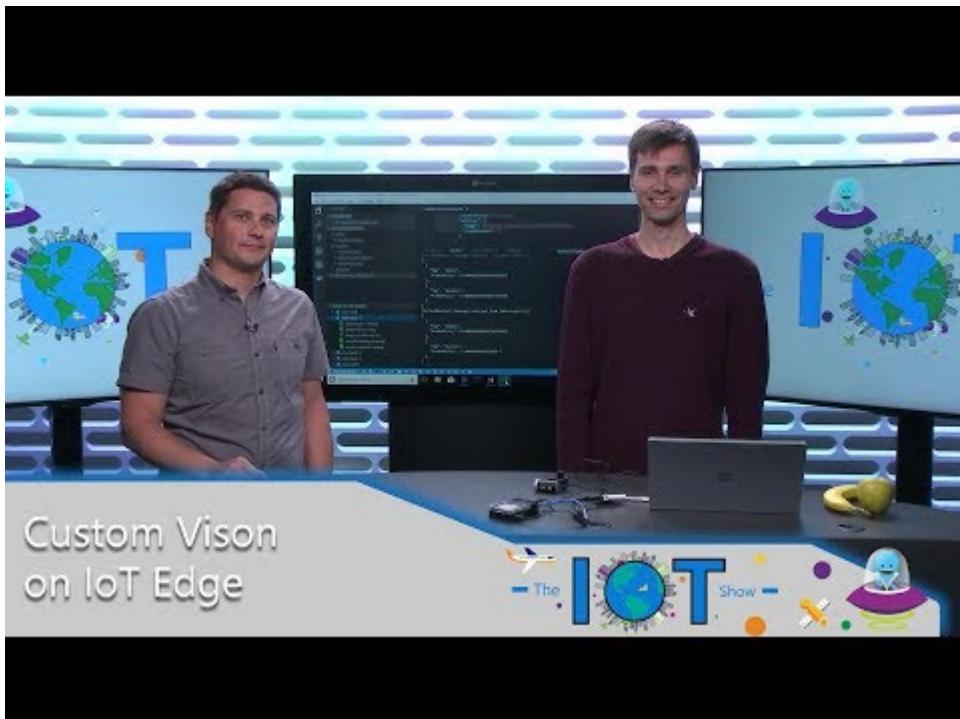
- Read up on the training API in the [Using the Custom Vision SDK quick start](#)

Assignment

Run your fruit detector on the edge

Add a sketchnote if possible/appropriate

This video gives an overview of running image classifiers on IoT devices, the topic that is covered in this lesson.



🎥 Click the image above to watch a video

Pre-lecture quiz

Pre-lecture quiz

Introduction

In this lesson you will learn about

In this lesson we'll cover:

- [Thing 1](#)

Thing 1

Challenge

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

Assignment

Trigger fruit quality detection from a sensor

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

An IoT application is not just a single device capturing data and sending it to the cloud, it is more often than not multiple devices all working together to capture data from the physical world using sensors, make decisions based off that data, and interacting back with the physical world via actuators or visualizations.

In this lesson you will learn more about architecting complex IoT applications, incorporating multiple sensors, multiple cloud services to analyze and store data, and showing a response via an actuator. You will learn how to architect a fruit quality control system prototype, including about using proximity sensors to trigger the IoT application, and what the architecture of this prototype would be.

In this lesson we'll cover:

- [Architect complex IoT applications](#)
- [Design a fruit quality control system](#)
- [Trigger fruit quality checking from a sensor](#)
- [Data used for a fruit quality detector](#)
- [Using developer devices to simulate multiple IoT devices](#)
- [Moving to production](#)

 This is the last lesson in this project, so after completing this lesson and the assignment, don't forget to clean up your cloud services. You will need the services to complete the assignment, so make sure to complete that first.

Refer to [the clean up your project guide](#) if necessary for instructions on how to do this.

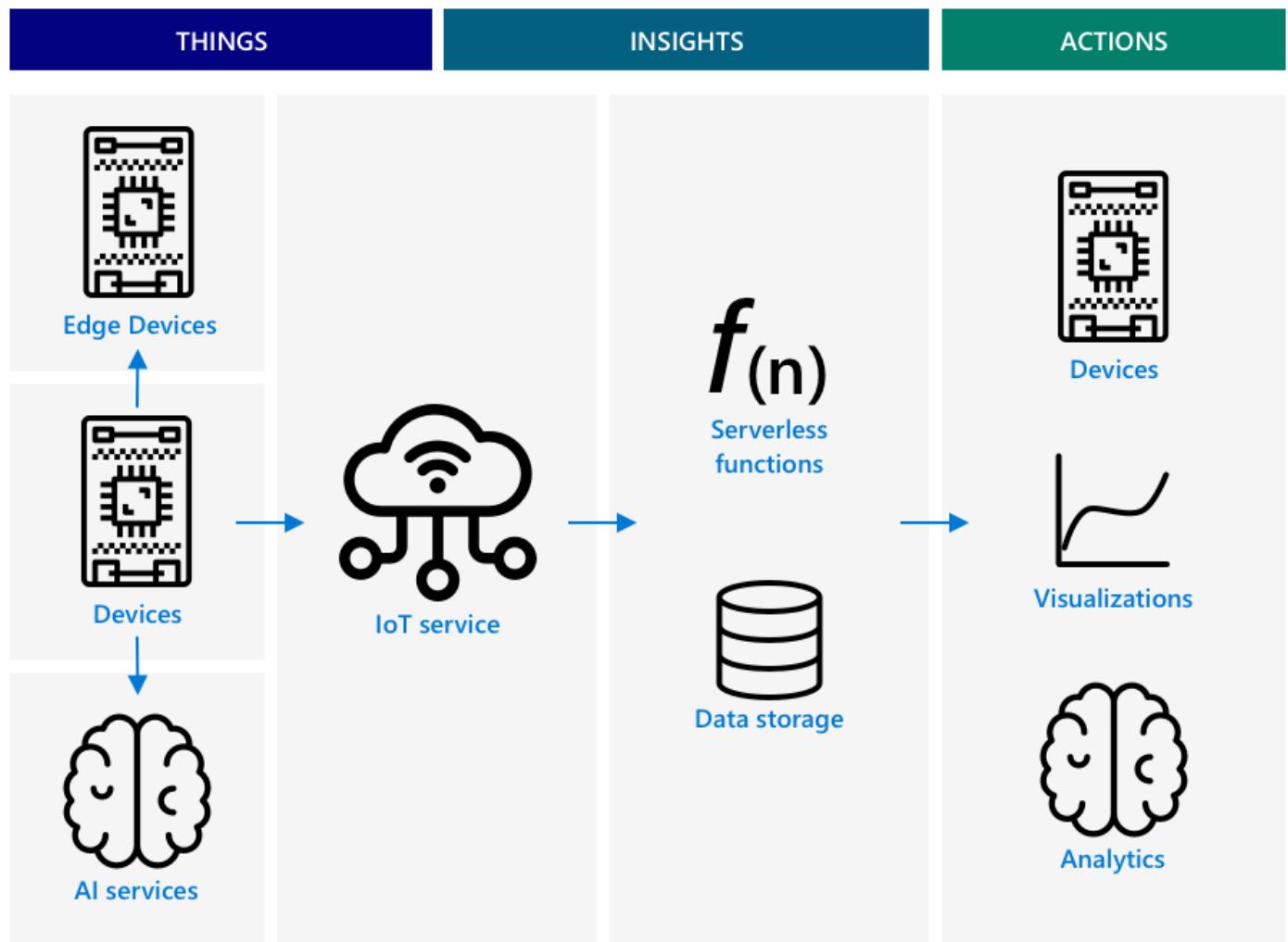
Architect complex IoT applications

IoT applications are made up of many components. This includes a variety of things, and a variety of internet services.

IoT applications can be described as *things* (devices) sending data that generates *insights*. These *insights* generate *actions* to improve a business or process. An example is an engine (the thing) sending temperature data. This data is used to evaluate whether the engine is performing as expected (the insight). The insight is used to proactively prioritize the maintenance schedule for the engine (the action).

- Different things gather different pieces of data.
- IoT services give insights over that data, sometimes augmenting it with data from additional sources.
- These insights drive actions, including controlling actuators in devices, or visualizing data.

Reference IoT architecture



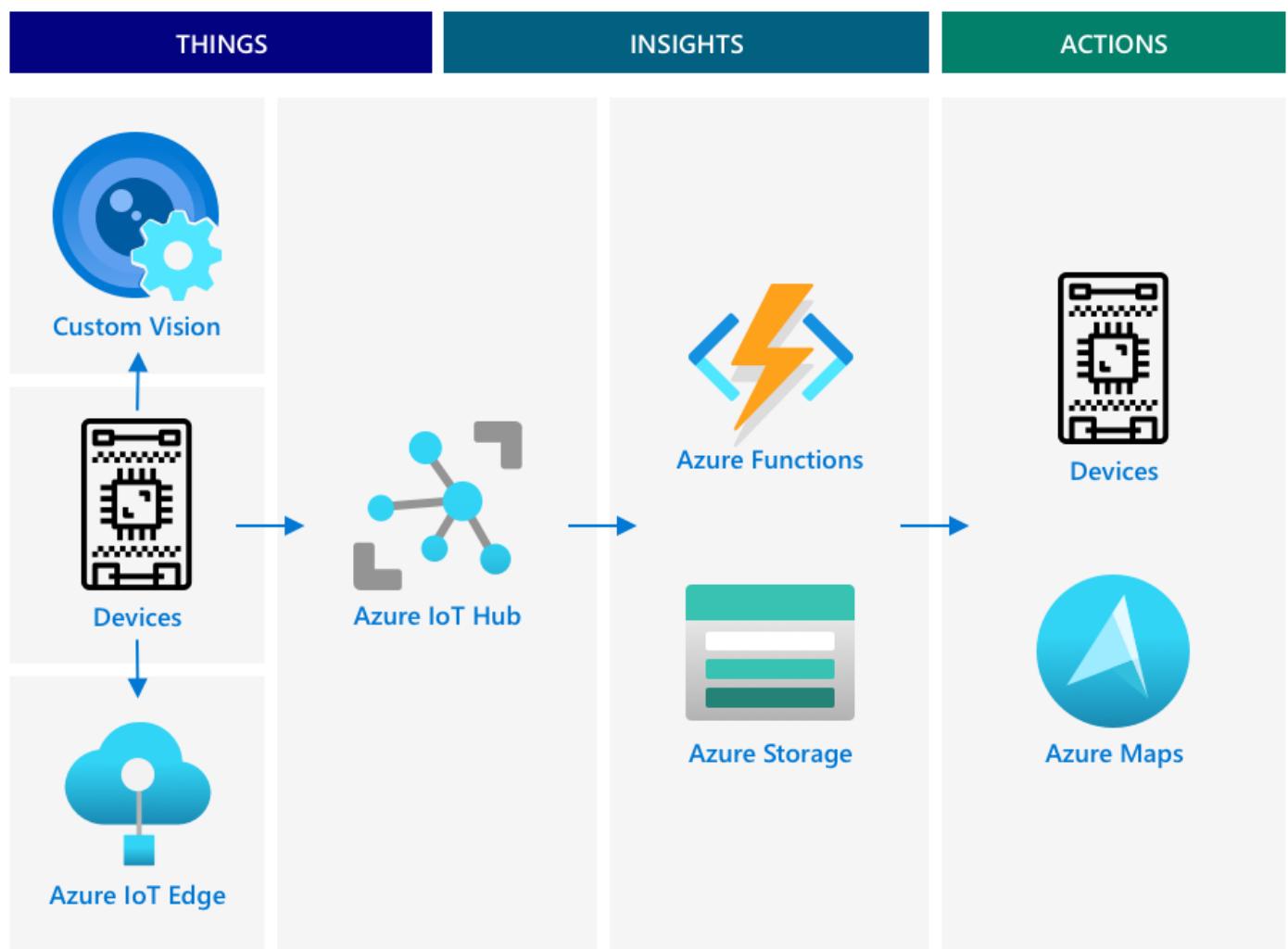
A reference iot architecture. Microcontroller by Template / IoT by Adrien Coquet / Brain by Icon Market - all from the [Noun Project](#)

The diagram above shows a reference IoT architecture.

👉 A reference architecture is an example architecture you can use as a reference when designing new systems. In this case, if you were building a new IoT system you can follow the reference architecture, substituting your own devices and services where appropriate.

- **Things** are devices that gather data from sensors, maybe interacting with edge services to interpret that data, such as image classifiers to interpret image data. The data from the devices is sent to an IoT service.

- **Insights** come from serverless applications, or from analytics run on stored data.
- **Actions** can be commands sent to devices, or visualization of data allowing humans to make decisions.



A reference iot architecture. Microcontroller by Template - all from the [Noun Project](#)

The diagram above shows some of the components and services covered so far in these lessons and how they link together in a reference IoT architecture.

- **Things** - you've written device code to capture data from sensors, and analyse images using Custom Vision running both in the cloud and on an edge device. This data was sent to IoT Hub.
- **Insights** - you've used Azure Functions to respond to messages sent to an IoT Hub, and stored data for later analysis in Azure Storage.
- **Actions** - you've controlled actuators based on decisions made in the cloud and commands sent to the devices, and you've visualized data using Azure Maps.

Think about other IoT devices you have used, such as smart home appliances. What are the things, insights and actions involved in that device and its software?

This pattern can be scaled out as large or small as you need, adding more devices and more services.

Data and security

As you define the architecture of your system, you need to constantly consider data and security.

- What data does your device send and receive?
- How should that data be secured and protected?
- How should access to the device and cloud service be controlled?

Think about the data security of any IoT devices you own. How much of that data is personal and should be kept private, both in transit or when stored? What data should not be stored?

Design a fruit quality control system

Lets now take this idea of things, insights, and actions and apply it to our fruit quality detector to design a larger end-to-end application.

Imagine you have been given the task of building a fruit quality detector to be used in a processing plant. Fruit travels on a conveyer belt system where currently employees spend time checking the fruit by hand and removing any unripe fruit as it arrives. To reduce costs, the plant owner wants an automated system.

One of the trends with the rise of IoT (and technology in general) is that manual jobs are being replaced by machines. Do some research: How many jobs are estimated to be lost to IoT? How many new jobs will be created building IoT devices?

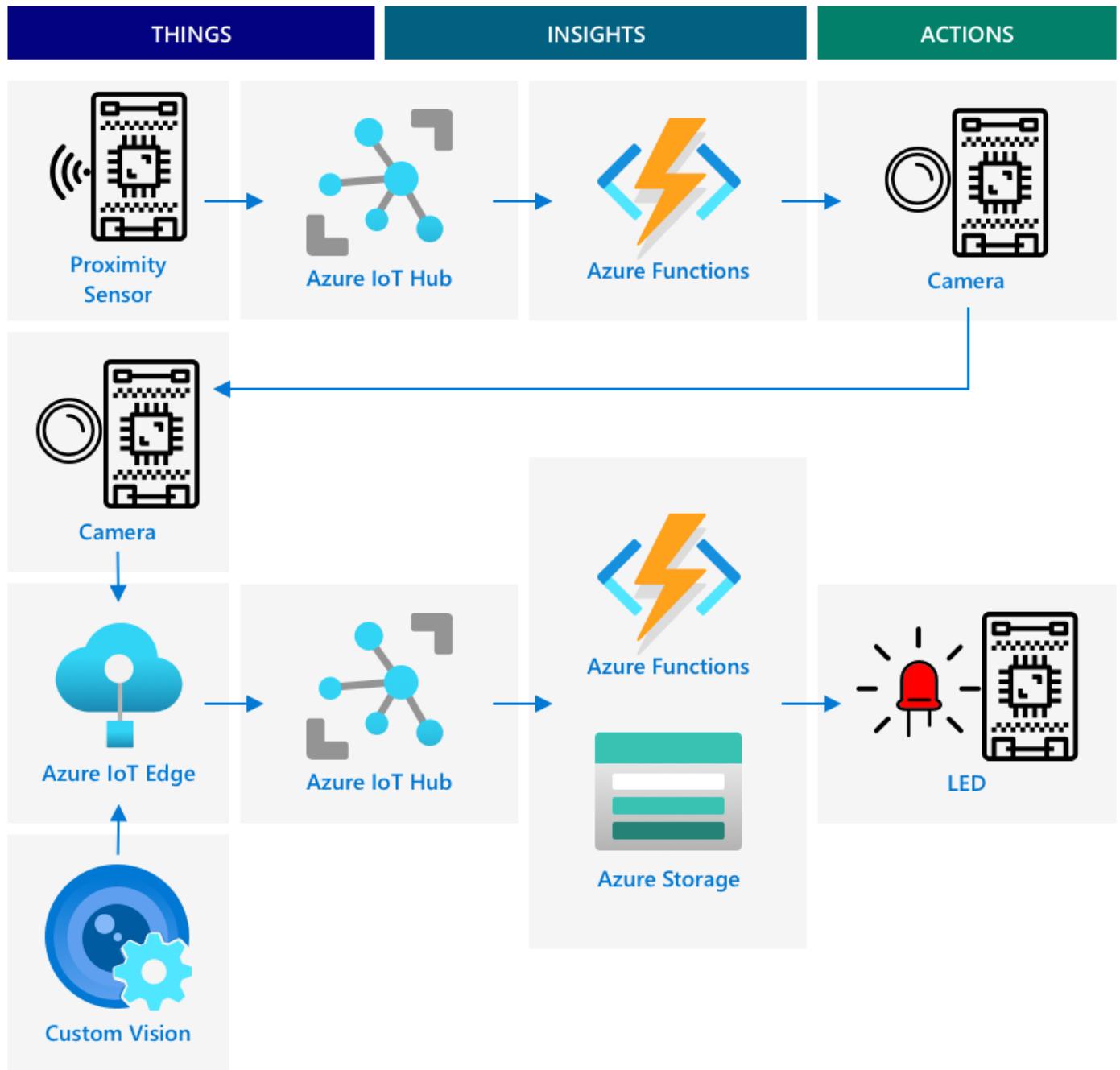
You need to build a system where fruit is detected as it arrives on the conveyer belt, it is then photographed and checked using an AI model running on the edge. The results are then sent to the cloud to be stored, and if the fruit is unripe a notification is given so the unripe fruit can be removed.

Things	Detector for fruit arriving on the conveyor belt Camera to photograph and classify the fruit Edge device running the classifier Device to notify of unripe fruit
---------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Insights	Decide to check the ripeness of the fruit Store the results of the ripeness classification Determine if there is a need to alert about unripe fruit
-----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------

Actions	Send a command to a device to photograph the fruit and check it with an image classifier Send a command to a device to alert that the fruit is unripe
----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------

Prototyping your application



A reference iot architecture for fruit quality checking. LED by abderraouf omara / Microcontroller by Template - all from the [Noun Project](#)

The diagram above shows a reference architecture for this prototype application.

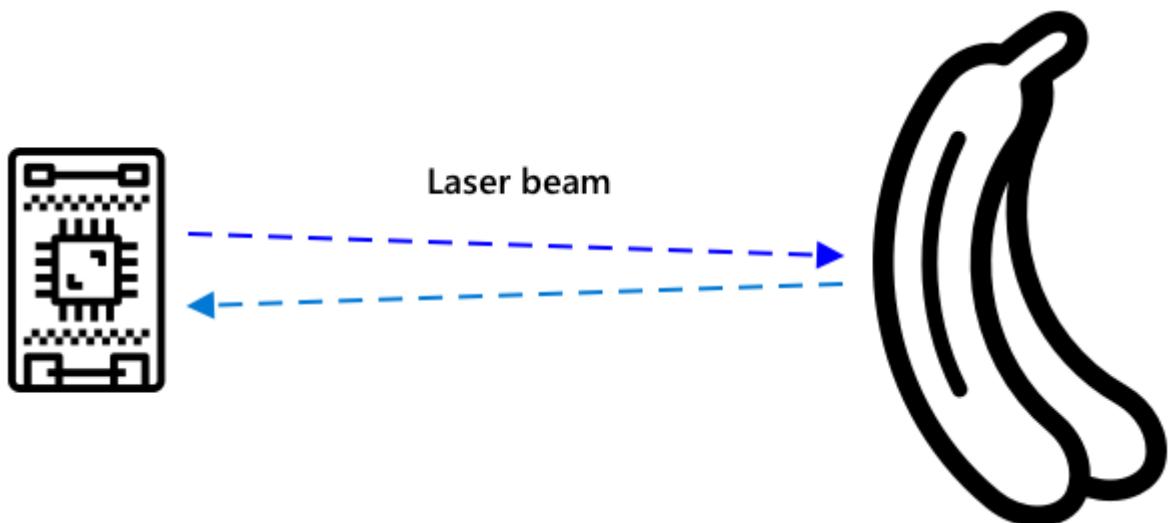
- An IoT device with a proximity sensor detects the arrival of fruit. This sends a message to the cloud to say fruit has been detected.
- A serverless application in the cloud sends a command to another device to take a photograph and classify the image.
- An IoT device with a camera takes a picture and sends it to an image classifier running on the edge. The results are then sent to the cloud.
- A serverless application in the cloud stores this information to be analyzed later to see what percentage of fruit is unripe. If the fruit is unripe it sends a command to another iot device to alert factory workers there is unripe fruit via an LED.

 This entire IoT application could be implemented as a single device, with all the logic to start the image classification and control the LED built in. It could use an IoT Hub just to track the number of unripe fruits detected and configure the device. In this lesson it is expanded to demonstrate the concepts for large scale IoT applications.

For the prototype, you will implement all of this on a single device. If you are using a microcontroller then you will use a separate edge device to run the image classifier. You have already learned most of the things you will need to be able to build this.

Trigger fruit quality checking from a sensor

The IoT device needs some kind of trigger to indicate when fruit is ready to be classified. One trigger for this would be to measure when the fruit is at the right location on the conveyor belt by measuring the distance to a sensor.



Proximity sensors send laser beams to objects like bananas and time how long till the beam is bounced back. Bananas by abderraouf omara / Microcontroller by Template - all from the [Noun Project](#)

Proximity sensors can be used to measure the distance from the sensor to an object. They usually transmit a beam of electromagnetic radiation such as a laser beam or infra-red light, then detect the radiation bouncing off an object. The time between the laser beam being sent and the signal bouncing back can be used to calculate the distance to the sensor.

 You have probably used proximity sensors without even knowing about it. Most smartphone will turn the screen off when you hold them to your ear to stop you accidentally

ending a call with your earlobe, and this works using a proximity sensor, detecting an object close to the screen during a call and disabling the touch capabilities until the phone is a certain distance away.

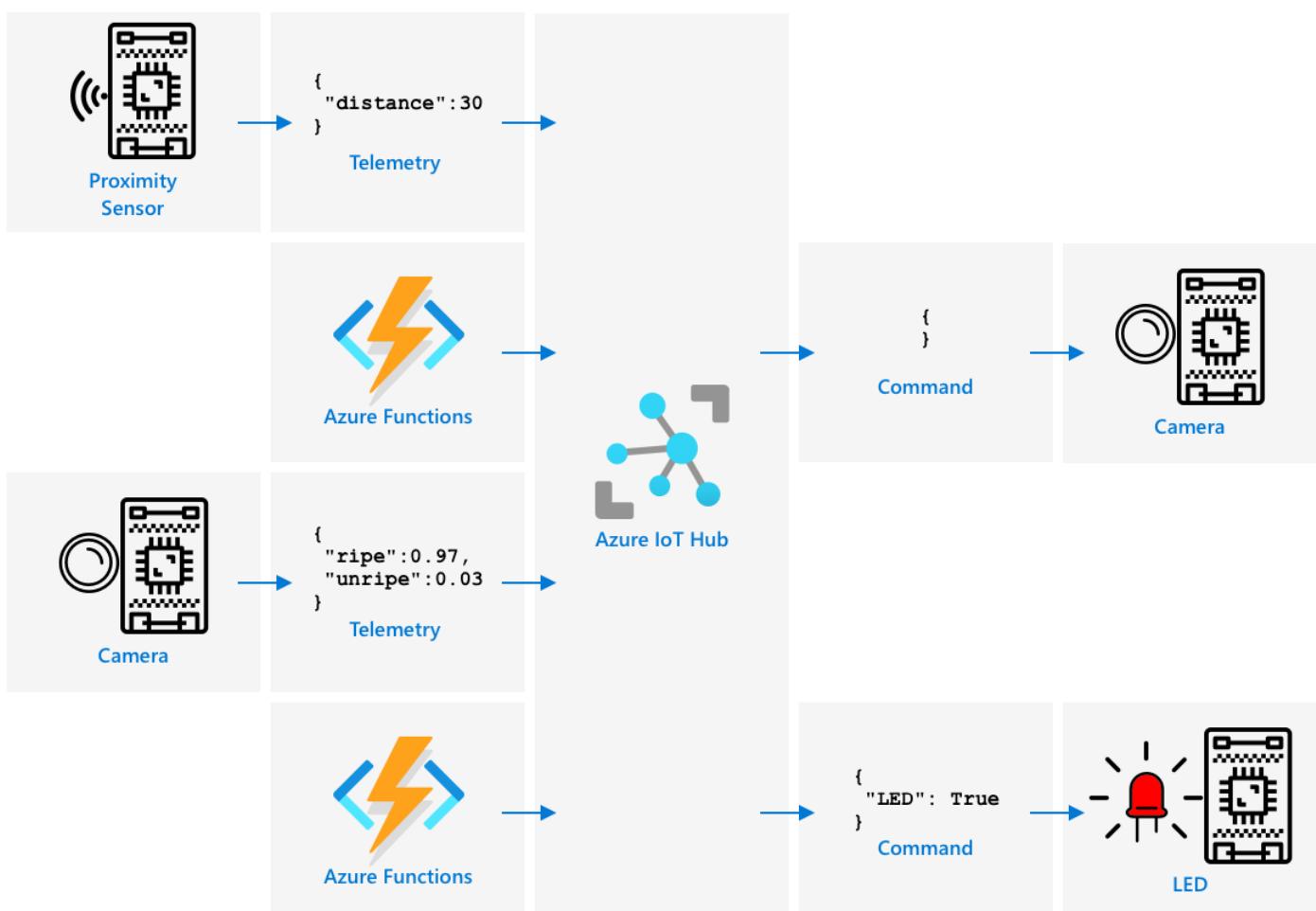
Task - trigger fruit quality detection from a distance sensor

Work through the relevant guide to use a proximity sensor to detect an object using your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Data used for a fruit quality detector

The prototype fruit detector has multiple components communicating with each other.



- A proximity sensor measuring the distance to a piece of fruit and sending this to IoT Hub
- The command to control the camera coming from IoT Hub to the camera device
- The results of the image classification being sent to IoT Hub

- The command to control an LED to alert when the fruit is unripe being sent from IoT Hub to the device with the LED

It is good to define the structure of these messages up front, before you build out the application.

 Pretty much every experienced developer has at some point in their career spent hours, days or even weeks chasing down bugs caused by differences in the data being sent compared to what is expected.

For example - if you are sending temperature information, how would you define the JSON? You could have a field called `temperature` , or you could use the common abbreviation `temp` .

```
json
{
    "temperature": 20.7
}
```

compared to:

```
json
{
    "temp": 20.7
}
```

You also have to consider units - is the temperature in °C or °F? If you are measuring temperature using a consumer device and they change the display units, you need to make sure the units sent to the cloud remain consistent.

 Do some research: How did unit problems cause the \$125 million Mars Climate Orbiter to crash?

Think about the data being sent for the fruit quality detector. How would you define each message? Where would you analyze the data and make decisions about what data to send?

For example - triggering the image classification using the proximity sensor. The IoT device measures the distance, but where is the decision made? Does the device decide that the fruit is close enough and sends a message to tell the IoT Hub to trigger the classification? Or does it send proximity measurements and let the IoT Hub decide?

The answer to questions like this is - it depends. Each use case is different, which is why as an IoT developer you need to understand the system you are building, how it is used, and the data being detected.

- If the decision is made by the IoT Hub, you need to send multiple distance measurements.
- If you send too many messages, it increases the cost of the IoT Hub, and the amount of bandwidth needed by your IoT devices (especially in a factory with millions of devices). It can also slow down your device.
- If you make the decision on the device, you will need to provide a way to configure the device to fine tune the machine.

Using developer devices to simulate multiple IoT devices

To build your prototype, you will need your IoT dev kit to act like multiple devices, sending telemetry and responding to commands.

Simulating multiple IoT devices on a Raspberry Pi or virtual IoT hardware

When using a single board computer like a Raspberry Pi, you are able to run multiple applications at once. This means you can simulate multiple IoT devices by creating multiple applications, one per 'IoT device'. For example, you can implement each device as a separate Python file and run them in different terminal sessions.

 Be aware that some hardware won't work when being accessed by multiple applications running simultaneously.

Simulating multiple devices on a microcontroller

Microcontrollers are more complicated to simulate multiple devices. Unlike single board computers you cannot run multiple applications at once, you have to include all the logic for all the separate IoT devices in a single application.

Some suggestions to make this process easier are:

- Create one or more classes per IoT device - for example classes called `DistanceSensor` , `ClassifierCamera` , `LEDController` . Each one can have its own `setup` and `loop` methods called by the main `setup` and `loop` functions.
- Handle commands in a single place, and direct them to the relevant device class as required.
- In the main `loop` function, you will need to consider the timing for each different device. For example, if you have one device class that needs to process every 10 seconds, and another that

needs to process every 1 second, then in your main `loop` function use a 1 second delay. Every `loop` call triggers the relevant code for the device that needs to process every second, and use a counter to count each loop, processing the other device when the counter reaches 10 (resetting the counter afterwards).

Moving to production

The prototype will form the basis of a final production system. Some of the differences when you move to production would be:

- Ruggedized components - using hardware designed to withstand the noise, heat, vibration and stress of a factory.
- Using internal communications - some of the components would communicate directly avoiding the hop to the cloud, only sending data to the cloud to be stored. How this is done depends on the factory setup, with either direct communications, or by running part of the IoT service on the edge using a gateway device.
- Configuration options - each factory and use case is different, so the hardware would need to be configurable. For example, the proximity sensor may need to detect different fruit at different distances. Rather than hard code the distance to trigger the classification, you would want this to be configurable via the cloud, for example using a device twin
- Automated fruit removal - instead of an LED to alert that fruit is unripe, automated devices would remove it.

 Do some research: In what other ways would production devices differ from developer kits?

Challenge

In this lesson you have learned some of the concepts you need to know to architect an IoT system. Think back to the previous projects. How would do they fit into the reference architecture shown above?

Pick one of the projects so far and think of the design of a more complicated solution bringing together multiple capabilities beyond what was covered in the projects. Draw the architecture and think of all the devices and services you would need.

For example - a vehicle tracking device that combines GPS with sensors to monitor things like temperatures in a refrigerated truck, the engine on and off times, and the identity of the driver. What

are the devices involved, the services involved, the data being transmitted and the security and privacy considerations?

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read more about IoT architecture on the [Azure IoT reference architecture documentation on Microsoft docs](#)
- Read more about device twins in the [Understand and use device twins in IoT Hub documentation on Microsoft docs](#)
- Read about OPC-UA, a machine to machine communication protocol used in industrial automation on the [OPC-UA page on Wikipedia](#)

Assignment

[Build a fruit quality detector](#)

Train a stock detector

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In this lesson you will learn about

In this lesson we'll cover:

- Thing 1

Thing 1

Challenge

Post-lecture quiz

Post-lecture quiz

Review & Self Study

Assignment

Check stock from an IoT device

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

Pre-lecture quiz

Introduction

In this lesson you will learn about

In this lesson we'll cover:

- Thing 1

Thing 1

Challenge

Post-lecture quiz

Post-lecture quiz

Review & Self Study

Assignment

Recognize speech with an IoT device

Add a sketchnote if possible/appropriate

This video gives an overview of the Azure speech service, a topic that will be covered in this lesson:



🎥 Click the image above to watch a video

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

'Alexa, set a 12 minute timer'

'Alexa, timer status'

'Alexa set a 8 minute timer called steam broccoli'

Smart devices are becoming more and more pervasive. Not just as smart speakers like HomePods, Echos and Google Homes, but embedded in our phones, watches, and even light fittings and thermostats.

👤 I have at least 19 devices in my home that have voice assistants, and that's just the ones I know about!

Voice control increases accessibility by allowing folks with limited movement to interact with devices. Whether it is a permanent disability such as being born without arms, to temporary disabilities such as broken arms, or having your hands full of shopping or young children, being able to control our houses from our voice instead of our hands opens up a world of access. Shouting 'Hey Siri, close my garage door' whilst dealing with a baby change and an unruly toddler can be a small but effective improvement on life.

One of the more popular uses for voice assistants is setting timers, especially kitchen timers. Being able to set multiple timers with just your voice is a great help in the kitchen - no need to stop kneading dough, stirring soup, or clean dumpling filling off your hands to use a physical timer.

In this lesson you will learn about building voice recognition into IoT devices. You'll learn about microphones as sensors, how to capture audio from a microphone attached to an IoT device, and how to use AI to convert what is heard into text. Throughout the rest of this project you will build a smart kitchen timer, able to set timers using your voice with multiple languages.

In this lesson we'll cover:

- [Microphones](#)
- [Capture audio from your IoT device](#)
- [Speech to text](#)
- [Convert speech to text](#)

Microphones

Microphones are analog sensors that convert sound waves into electrical signals. Vibrations in air cause components in the microphone to move tiny amounts, and these cause tiny changes in electrical signals. These changes are then amplified to generate an electrical output.

Microphone types

Microphones come in a variety of types:

- Dynamic - Dynamic microphones have magnet attached to a moving diaphragm that moves in a coil of wire creating an electrical current. This is the opposite of most loudspeakers, that use an electrical current to move a magnet in a coil of wire, moving a diaphragm to create sound. This means speakers can be used as dynamic microphones, and dynamic microphones can be used as speakers. In devices such as intercoms where a user is either listening or speaking, but not both, one device can act as both a speaker and a microphone.

Dynamic microphones don't need power to work, the electrical signal is created entirely from the microphone.



[Beni Köhler / Creative Commons Attribution-Share Alike 3.0 Unported](#)

- Ribbon - Ribbon microphones are similar to dynamic microphones, except they have a metal ribbon instead of a diaphragm. This ribbon moves in a magnetic field generating an electrical current. Like dynamic microphones, ribbon microphones don't need power to work.

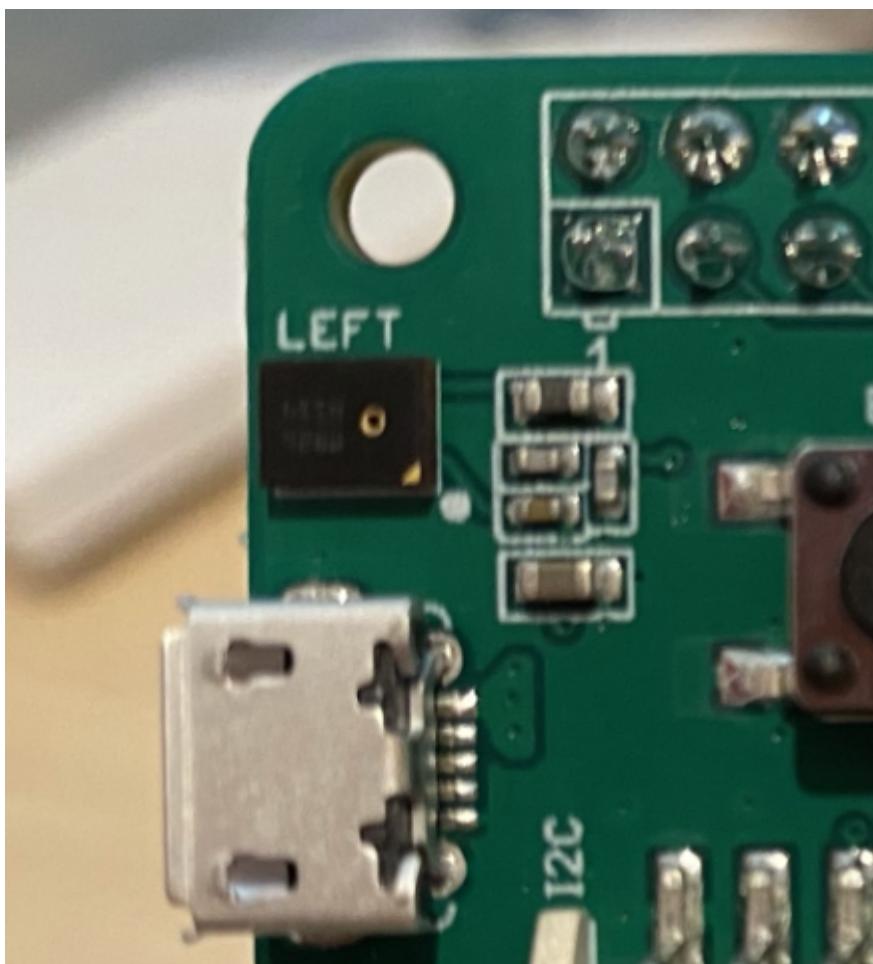


- Condenser - Condenser microphones have a thin metal diaphragm and a fixed metal backplate. Electricity is applied to both of these and as the diaphragm vibrates the static charge between the plates changes generating a signal. Condenser microphones need power to work - called *Phantom power*.



[Harumphy at en.wikipedia / Creative Commons Attribution-Share Alike 3.0 Unported](#)

- MEMS - Microelectromechanical systems microphones, or MEMS, are microphones on a chip. They have a pressure sensitive diaphragm etched onto a silicon chip, and work similar to a condenser microphone. These microphones can be tiny, and integrated into circuitry.



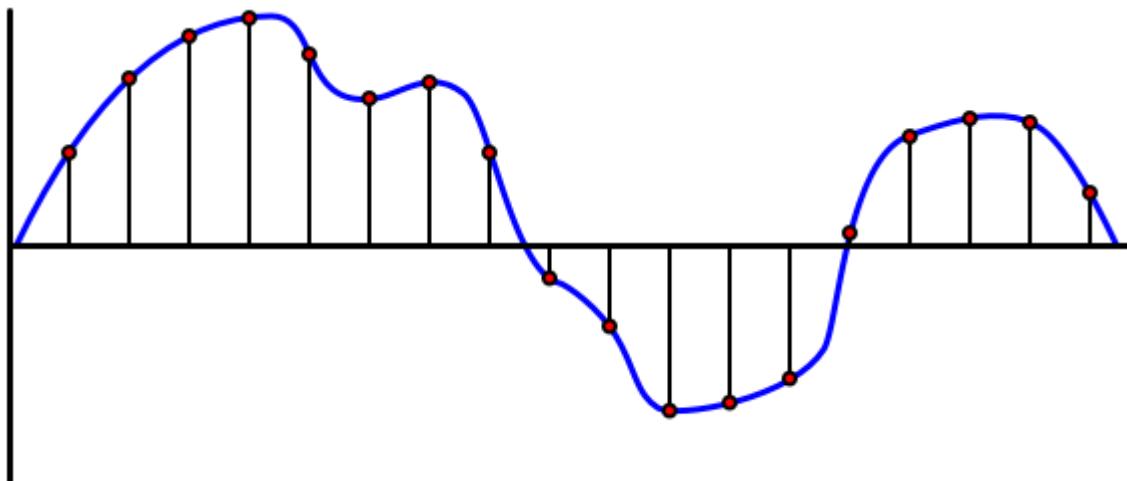
In the image above, the chip labelled **LEFT** is a MEMS microphone, with a tiny diaphragm less than a millimeter wide.

- Do some research: What microphones do you have around you - either in your computer, your phone, your headset or in other devices. What type of microphones are they?

Digital audio

Audio is an analog signal carrying very fine-grained information. To convert this signal to digital, the audio needs to be sampled many thousands of times a second.

👉 Sampling is converting the audio signal into a digital value that represents the signal at that point in time.



Digital audio is sampled using Pulse Code Modulation, or PCM. PCM involves reading the voltage of the signal, and selecting the closest discrete value to that voltage using a defined size.

👉 You can think of PCM as the sensor version of pulse width modulation, or PWM (PWM was covered back in [lesson 3 of the getting started project](#)). PCM involves converting an analog signal to digital, PWM involves converting a digital signal to analog.

For example most streaming music services offer 16-bit or 24-bit audio. This means they convert the voltage into a value that fits into a 16-bit integer, or 24-bit integer. 16-bit audio fits the value into a number ranging from -32,768 to 32,767, 24-bit is in the range -8,388,608 to 8,388,607. The more bits, the closer the sample is to what our ears actually hear.

 You may have heard of 8-bit audio, often referred to as LoFi. This is audio sampled using only 8-bits, so -128 to 127. The first computer audio was limited to 8 bits due to hardware limitations, so this is often seen in retro gaming.

These samples are taken many thousands of times per second, using well-defined sample rates measured in KHz (thousands of readings per second). Streaming music services use 48KHz for most audio, but some 'lossless' audio uses up to 96KHz or even 192KHz. The higher the sample rate, the closer to the original the audio will be, up to a point. There is debate whether humans can tell the difference above 48KHz.

Do some research: If you use a streaming music service, what sample rate and size does it use? If you use CDs, what is the sample rate and size of CD audio?

Capture audio from your IoT device

Your IoT device can be connected to a microphone to capture audio, ready for conversion to text. It can also be connected to speakers to output audio. In later lessons this will be used to give audio feedback, but it is useful to set up speakers now to test the microphone.

Task - configure your microphone and speakers

Work through the relevant guide to configure the microphone and speakers for your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Task - capture audio

Work through the relevant guide to capture audio on your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)
- [Single-board computer - Virtual device](#)

Speech to text

Speech to text, or speech recognition, involves using AI to convert words in an audio signal to text.

Speech recognition models

To convert speech to text, samples from the audio signal are grouped together and fed into a machine learning model based around a Recurrent Neural network (RNN). This is a type of machine learning model that can use previous data to make a decision about incoming data. For example, the RNN could detect one block of audio samples as the sound 'He!', and when it receives another that it thinks is the sound 'lō', it can combine this with the previous sound, find that 'Hello' is a valid word and select that as the outcome.

ML models always accept data of the same size every time. The image classifier you built in an earlier lesson resizes images to a fixed size and processes them. The same with speech models, they have to process fixed sized audio chunks. The speech models need to be able to combine the outputs of multiple predictions to get the answer, to allow it to distinguish between 'Hi' and 'Highway', or 'flock' and 'floccinaucinihilipilification'.

Speech models are also advanced enough to understand context, and can correct the words they detect as more sounds are processed. For example, if you say "I went to the shops to get two bananas and an apple too", you would use three words that sound the same, but are spelled differently - to, two and too. Speech models are able to understand the context and use the appropriate spelling of the word.



Some speech services allow customization to make them work better in noisy environments such as factories, or with industry-specific words such as chemical names. These customizations are trained by providing sample audio and a transcription, and work using transfer learning, the same as how you trained an image classifier using only a few images in an earlier lesson.

Privacy

When using speech to text in a consumer IoT device, privacy is incredibly important. These devices listen to audio continuously, so as a consumer you don't want everything you say being sent to the cloud and converted to text. Not only will this use a lot of Internet bandwidth, it also has massive privacy implications, especially when some smart device makers randomly select audio for humans to validate against the text generated to help improve their model.

You only want your smart device to send audio to the cloud for processing when you are using it, not when it hears audio in your home, audio that could include private meetings or intimate interactions. The way most smart devices work is with a *wake word*, a key phrase such as "Alexa", "Hey Siri", or

"OK Google" that causes the device to 'wake up' and listen to what you are saying up until it detects a break in your speech, indicating you have finished talking to the device.

 Wake word detection is also referred to as Keyword spotting or Keyword recognition.

These wake words are detected on the device, not in the cloud. These smart devices have small AI models that run on the device that listen for the wake work, and when it is detected, start streaming the audio to the cloud for recognition. These models are very specialized, and just listen for the wake word.

 Some tech companies are adding more privacy to their devices and doing some of the speech to text conversion on the device. Apple have announced that as part of their 2021 iOS and macOS updates they will support the speech to text conversion on device, and be able to handle many requests without needing to use the cloud. This is thanks to having powerful processors in their devices that can run ML models.

 What do you think are the privacy and ethical implications of storing the audio sent to the cloud? Should this audio be stored, and if so, how? Do you think the use of recordings for law enforcement is a good trade off for the loss of privacy?

Wake word detection usually uses a technique known as TinyML, that is converting ML models to be able to run on microcontrollers. These models are small in size, and consume very little power to run.

To avoid the complexity of training and using a wake word model, the smart timer you are building in this lesson will use a button to turn on the speech recognition.

 If you want to try creating a wake word detection model to run on the Wio Terminal or Raspberry Pi, check out this [Responding to your voice tutorial by Edge Impulse](#). If you want to use your computer to do this, you can try the [Get started with Custom Keyword quickstart on the Microsoft docs](#).

Convert speech to text



Just like with image classification in an earlier project, there are pre-built AI services that can take speech as an audio file and convert it to text. Once such service is the Speech Service, part of the Cognitive Services, pre-built AI services you can use in your apps.

Task - configure a speech AI resource

1. Create a Resource Group for this project called `smart-timer`
2. Use the following command to create a free speech resource:

```
sh
az cognitiveservices account create --name smart-timer \
--resource-group smart-timer \
--kind SpeechServices \
--sku F0 \
--yes \
--location <location>
```

Replace `<location>` with the location you used when creating the Resource Group.

3. You will need an API key to access the speech resource from your code. Run the following command to get the key:

```
sh
az cognitiveservices account keys list --name smart-timer \
--resource-group smart-timer \
--output table
```

Take a copy of one of the keys.

Task - convert speech to text

Work through the relevant guide to convert speech to text on your IoT device:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi](#)

- [Single-board computer - Virtual device](#)

Task - send converted speech to an IoT services

To use the results of the speech to text conversion, you need to send it to the cloud. There it will be interpreted and responses sent back to the IoT device as commands.

1. Create a new IoT Hub in the `smart-timer` resource group, and register a new device called `smart-timer`.
2. Connect your IoT device to this IoT Hub using what you have learned in previous lessons, and send the speech as telemetry. Use a JSON document in this format:

```
json  
{  
    "speech" : "<converted speech>"  
}
```

Where `<converted speech>` is the output from the speech to text call. You only need to send speech that has content, if the call returns an empty string it can be ignored.

3. Verify that messages are being sent by monitoring the Event Hub compatible endpoint using the `az iot hub monitor-events` command.

 You can find this code in the [code-iot-hub/virtual-iot-device](#), [code-iot-hub/pi](#), or [code-iot-hub/wio-terminal](#) folder.

🚀 Challenge

Speech recognition has been around for a long time, and is continuously improving. Research the current capabilities and compare how these have evolved over time, including how accurate machine transcriptions are compared to human.

What do you think the future holds for speech recognition?

Post-lecture quiz

Review & Self Study

- Read about the different microphone types and how they work on the [What's the difference between dynamic and condenser microphones article on Musician's HQ.](#)
- Read more on the Cognitive Services speech service on the [Speech service documentation on Microsoft Docs](#)
- Read about keyword spotting on the [Keyword recognition documentation on Microsoft Docs](#)

Assignment

Understand language

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

Pre-lecture quiz

Introduction

In the last lesson you converted speech to text. For this to be used to program a smart timer, your code will need to have an understanding of what was said. You could assume the user will speak a fixed phrase, such as "Set a 3 minute timer", and parse that expression to get how long the timer should be, but this isn't very user-friendly. If a user were to say "Set a timer for 3 minutes", you or I would understand what they mean, but your code would not, it would be expecting a fixed phrase.

This is where language understanding comes in, using AI models to interpret text and return the details that are needed, for example being able to take both "Set a 3 minute timer" and "Set a timer for 3 minutes", and understand that a timer is required for 3 minutes.

In this lesson you will learn about language understanding models, how to create them, train them, and use them from your code.

In this lesson we'll cover:

- [Language understanding](#)
- [Create a language understanding model](#)
- [Intents and entities](#)
- [Use the language understanding model](#)

Language understanding

Humans have used language to communicate for hundreds of thousands of years. We communicate with words, sounds, or actions and understand what is said, both the meaning of the words, sounds or actions, but also their context. We understand sincerity and sarcasm, allowing the same words to mean different things depending on the tone of our voice.

 Think about some of the conversations you have had recently. How much of the conversation would be hard for a computer to understand because it needs context?

Language understanding, also called natural-language understanding is part of a field of artificial intelligence called natural-language processing (or NLP), and deals with reading comprehension, trying to understand the details of words or sentences. If you use a voice assistant such as Alexa or Siri, you have used language understanding services. These are the behind-the-scenes AI services that convert "Alexa, play the latest album by Taylor Swift" into my daughter dancing around the living room to her favorite tunes.

 Computers, despite all their advances, still have a long way to go to truly understand text. When we refer to language understanding with computers, we don't mean anything anywhere near as advanced as human communication, instead we mean taking some words and extracting key details.

As humans, we understand language without really thinking about it. If I asked another human to "play the latest album by Taylor Swift" then they would instinctively know what I meant. For a computer, this is harder. It would have to take the words, converted from speech to text, and work out the following pieces of information:

- Music needs to be played
- The music is by the artist Taylor Swift

- The specific music is a whole album of multiple tracks in order
 - Taylor Swift has many albums, so they need to be sorted by chronological order and the most recently published is the one required
- ✓ Think of some other sentences you have spoken when making requests, such as ordering coffee or asking a family member to pass you something. Try to break them down into the pieces of information a computer would need to extract to understand the sentence.

Language understanding models are AI models that are trained to extract certain details from language, and then are trained for specific tasks using transfer learning, in the same way you trained a Custom Vision model using a small set of images. You can take a model, then train it using the text you want it to understand.

Create a language understanding model



You can create language understanding models using LUIS, a language understanding service from Microsoft that is part of Cognitive Services.

Task - create an authoring resource

To use LUIS, you need to create an authoring resource.

1. Use the following command to create an authoring resource in your `smart-timer` resource group:

```
python
az cognitiveservices account create --name smart-timer-luis-authoring \
--resource-group smart-timer \
--kind LUIS.Authoring \
--sku F0 \
--yes \
--location <location>
```

Replace <location> with the location you used when creating the Resource Group.

⚠️ LUIS isn't available in all regions, so if you get the following error:

```
output  
InvalidApisetId: The account type 'LUIS.Authoring' is either inval
```

pick a different region.

This will create a free-tier LUIS authoring resource.

Task - create a language understanding app

1. Open the LUIS portal at luis.ai in your browser, and sign in with the same account you have been using for Azure.
2. Follow the instructions on the dialog to select your Azure subscription, then select the `smart-timer-luis-authoring` resource you have just created.
3. From the *Conversation apps* list, select the **New app** button to create a new application. Name the new app `smart-timer`, and set the *Culture* to your language.

💡 There is a field for a prediction resource. You can create a second resource just for prediction, but the free authoring resource allows 1,000 predictions a month which should be enough for development, so you can leave this blank.

4. Read through the guide that appears once you create the app to get an understanding of the steps you need to take to train the language understanding model. Close this guide when you are done.

Intents and entities

Language understanding is based around *intents* and *entities*. Intents are what the intent of the words are, for example playing music, setting a timer, or ordering food. Entities are what the intent is

referring to, such as the album, the length of the timer, or the type of food. Each sentence that the model interprets should have at least one intent, and optionally one or more entities.

Some examples:

Sentence	Intent	Entities
"Play the latest album by Taylor Swift"	<i>play music</i>	<i>the latest album by Taylor Swift</i>
"Set a 3 minute timer"	<i>set a timer</i>	<i>3 minutes</i>
"Cancel my timer"	<i>cancel a timer</i>	None
"Order 3 large pineapple pizzas and a caesar salad"	<i>order food</i>	<i>3 large pineapple pizzas, caesar salad</i>

- With the sentences you thought about earlier, what would be the intent and any entities in that sentence?

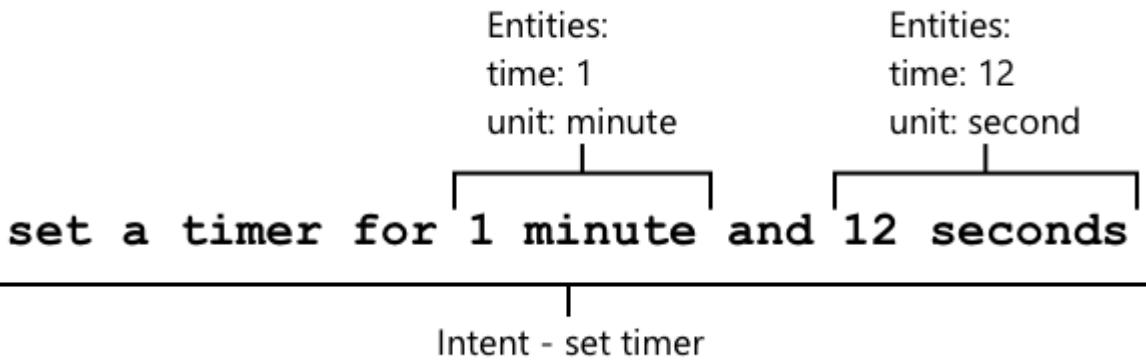
To train LUIS, first you set the entities. These can be a fixed list of terms, or learned from the text. For example, you could provide a fixed list of food available from your menu, with variations (or synonyms) of each word, such as *egg plant* and *aubergine* as variations of *aubergine*. LUIS also has pre-built entities that can be used, such as numbers and locations.

For setting a timer, you could have one entity using the pre-built number entities for the time, and another for the units, such as minutes and seconds. Each unit would have multiple variations to cover the singular and plural forms - such as minute and minutes.

Once the entities are defined, you create intents. These are learned by the model based on example sentences that you provide (known as utterances). For example, for a *set timer* intent, you might provide the following sentences:

- set a 1 second timer
- set a timer for 1 minute and 12 seconds
- set a timer for 3 minutes
- set a 9 minute 30 second timer

You then tell LUIS what parts of these sentences map to the entities:



The sentence `set a timer for 1 minute and 12 seconds` has the intent of `set timer`. It also has 2 entities with 2 values each:

time	unit
1 minute	1 minute
12 seconds	12 second

To train a good model, you need a range of different example sentences to cover the many different ways someone might ask for the same thing.

💡 As with any AI model, the more data and the more accurate the data you use to train, the better the model.

✓ Think about the different ways you might ask the same thing and expect a human to understand.

Task - add entities to the language understanding models

For the timer, you need to add 2 entities - one for the unit of time (minutes or seconds), and one for the number of minutes or seconds.

You can find instructions for using the LUIS portal in the [Quickstart: Build your app in LUIS portal documentation on Microsoft docs](#).

1. From the LUIS portal, select the *Entities* tab and add the *number* prebuilt entity by selecting the **Add prebuilt entity** button, then selecting *number* from the list.
2. Create a new entity for the time unit using the **Create** button. Name the entity `time unit` and set the type to *List*. Add values for `minute` and `second` to the *Normalized values* list, adding

the singular and plural forms to the *synonyms* list. Press `return` after adding each synonym to add it to the list.

Normalized value Synonyms

minute minute, minutes

second second, seconds

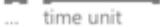
Task - add intents to the language understanding models

1. From the *Intents* tab, select the **Create** button to create a new intent. Name this intent `set timer`.
2. In the examples, enter different ways to set a timer using both minutes, seconds and minutes and seconds combined. Examples could be:
 - `set a 1 second timer`
 - `set a 4 minute timer`
 - `set a four minute six second timer`
 - `set a 9 minute 30 second timer`
 - `set a timer for 1 minute and 12 seconds`
 - `set a timer for 3 minutes`
 - `set a timer for 3 minutes and 1 second`
 - `set a timer for three minutes and one second`
 - `set a timer for 1 minute and 1 second`
 - `set a timer for 30 seconds`
 - `set a timer for 1 second`

Mix up numbers as words and numerics so the model learns to handle both.

3. As you enter each example, LUIS will start detecting entities, and will underline and label any it finds.

set a timer for 1 second



set a timer for 30 seconds



set a timer for 1 minute1 and 1 second



set a timer for 3 minutes and 1 second



set a timer for 3 minutes



Task - train and test the model

1. Once the entities and intents are configured, you can train the model using the **Train** button on the top menu. Select this button, and the model should train in a few seconds. The button will be greyed out whilst training, and be re-enabled once done.
2. Select the **Test** button from the top menu to test the language understanding model. Enter text such as `set a timer for 5 minutes and 4 seconds` and press return. The sentence will appear in a box under the text box that you typed it in to, and below that will be the *top intent*, or the intent that was detected with the highest probability. This should be `set timer`. The intent name will be followed by the probability that the intent detected was the right one.
3. Select the **Inspect** option to see a breakdown of the results. You will see the top-scoring intent with its percentage probability, along with lists of the entities detected.
4. Close the *Test* pane when you are done testing.

Task - publish the model

To use this model from code, you need to publish it. When publishing from LUIS, you can publish to either a staging environment for testing, or a product environment for a full release. In this lesson, a staging environment is fine.

1. From the LUIS portal, select the **Publish** button from the top menu.
2. Make sure *Staging slot* is selected, then select **Done**. You will see a notification when the app is published.

3. You can test this using curl. To build the curl command, you need three values - the endpoint, the application ID (App ID) and an API key. These can be accessed from the **MANAGE** tab that can be selected from the top menu.

1. From the *Settings* section, copy the App ID
2. From the *Azure Resources* section, select *Authoring Resource*, and copy the *Primary Key* and *Endpoint URL*
4. Run the following curl command in your command prompt or terminal:

```
sh
curl "<endpoint url>/luis/prediction/v3.0/apps/<app id>/slots/staging/pr
      --request GET \
      --get \
      --data "subscription-key=<primary key>" \
      --data "verbose=false" \
      --data "show-all-intents=true" \
      --data-urlencode "query=<sentence>"
```

Replace `<endpoint url>` with the Endpoint URL from the *Azure Resources* section.

Replace `<app id>` with the App ID from the *Settings* section.

Replace `<primary key>` with the Primary Key from the *Azure Resources* section.

Replace `<sentence>` with the sentence you want to test with.

5. The output of this call will be a JSON document that details the query, the top intent, and a list of entities broken down by type.

```
JSON
{
  "query": "set a timer for 45 minutes and 12 seconds",
  "prediction": {
    "topIntent": "set timer",
    "intents": {
      "set timer": {
        "score": 0.97031575
      },
      "None": {
        "score": 0.02205793
      }
    },
    "entities": {
      "number": [
        "45"
      ]
    }
  }
}
```

```

        45,
        12
    ],
    "time-unit": [
        [
            "minute"
        ],
        [
            "second"
        ]
    ]
}
}
}

```

The JSON above came from querying with

set a timer for 45 minutes and 12 seconds :

- The `set timer` was the top intent with a probability of 97%.
- Two `number` entities were detected, `45` and `12`.
- Two `time-unit` entities were detected, `minute` and `second`.

Use the language understanding model

Once published, the LUIS model can be called from code. In the last lesson you sent the recognized speech to an IoT Hub, and you can use serverless code to respond to this and understand what was sent.

Task - create a serverless functions app

1. Create an Azure Functions app called `smart-timer-trigger`.
2. Add an IoT Hub event trigger to this app called `speech-trigger`.
3. Set the Event Hub compatible endpoint connection string for your IoT Hub in the `local.settings.json` file, and use the key for that entry in the `function.json` file.
4. Use the Azurite app as a local storage emulator.
5. Run your functions app and your IoT device to ensure speech is arriving at the IoT Hub.

output

Python EventHub trigger processed an event: {"speech": "Set a 3 minute t

Task - use the language understanding model

1. The SDK for LUIS is available via a Pip package. Add the following line to the requirements.txt file to add the dependency on this package:

```
sh  
azure-cognitiveservices-language-luis
```

2. Make sure the VS Code terminal has the virtual environment activated, and run the following command to install the Pip packages:

```
sh  
pip install -r requirements.txt
```

3. Add new entries to the local.settings.json file for your LUIS API Key, Endpoint URL, and App ID from the **MANAGE** tab of the LUIS portal:

```
JSON  
"LUIS_KEY": "<primary key>",  
"LUIS_ENDPOINT_URL": "<endpoint url>",  
"LUIS_APP_ID": "<app id>"
```

Replace `<endpoint url>` with the Endpoint URL from the *Azure Resources* section of the **MANAGE** tab. This will be `https://<location>.api.cognitive.microsoft.com/`.

Replace `<app id>` with the App ID from the *Settings* section of the **MANAGE** tab.

Replace `<primary key>` with the Primary Key from the *Azure Resources* section of the **MANAGE** tab.

4. Add the following imports to the `__init__.py` file:

```
python  
import json  
import os  
from azure.cognitiveservices.language.luis.runtime import LuisRuntimeClient  
from msrest.authentication import CognitiveServicesCredentials
```

This imports some system libraries, as well as the libraries to interact with LUIS.

5. In the `main` method, before it loops through all the events, add the following code:

python

```
luis_key = os.environ['LUIS_KEY']
endpoint_url = os.environ['LUIS_ENDPOINT_URL']
app_id = os.environ['LUIS_APP_ID']

credentials = CognitiveServicesCredentials(luis_key)
client = LUISRuntimeClient(endpoint=endpoint_url, credentials=credentials)
```

This loads the values you added to the `local.settings.json` file for your LUIS app, creates a credentials object with your API key, then creates a LUIS client object to interact with your LUIS app.

6. Predictions are requested from LUIS by sending a prediction request - a JSON document containing the text to predict. Create this with the following code inside the `for event in events` loop:

python

```
event_body = json.loads(event.get_body().decode('utf-8'))
prediction_request = { 'query' : event_body['speech'] }
```

This code extracts the speech that was sent to the IoT Hub and uses it to build the prediction request.

7. This request can then be sent to LUIS, using the staging slot that your app was published to:

python

```
prediction_response = client.prediction.get_slot_prediction(app_id, 'Sta
```

8. The prediction response contains the top intent - the intent with the highest prediction score, along with the entities. If the top intent is `set timer`, then the entities can be read to get the time needed for the timer:

python

```
if prediction_response.prediction.top_intent == 'set timer':
    numbers = prediction_response.prediction.entities['number']
    time_units = prediction_response.prediction.entities['time unit']
    total_seconds = 0
```

The `number` entities will be an array of numbers. For example, if you said "Set a four minute 17 second timer.", then the `number` array will contain 2 integers - 4 and 17.

The `time unit` entities will be an array of arrays of strings, with each time unit as an array of strings inside the array. For example, if you said "Set a four minute 17 second timer.", then the `time unit` array will contain 2 arrays with single values each - `['minute']` and `['second']`.

The JSON version of these entities for "Set a four minute 17 second timer." is:

```
json
{
    "number": [4, 17],
    "time unit": [
        ["minute"],
        ["second"]
    ]
}
```

This code also defines a count for the total time for the timer in seconds. This will be populated by the values from the entities.

9. The entities aren't linked, but we can make some assumptions about them. They will be in the order spoken, so the position in the array can be used to determine which number matches to which time unit. For example:

- "Set a 30 second timer" - this will have one number, `30`, and one time unit, `second` so the single number will match the single time unit.
- "Set a 2 minute and 30 second timer" - this will have two numbers, `2` and `30`, and two time units, `minute` and `second` so the first number will be for the first time unit (2 minutes), and the second number for the second time unit (30 seconds).

The following code gets the count of items in the number entities, and uses that to extract the first item from each array, then the second and so on:

```
python
for i in range(0, len(numbers)):
    number = numbers[i]
    time_unit = time_units[i][0]
```

For "Set a four minute 17 second timer.", this will loop twice, giving the following values:

loop count	number	time_unit
0	4	minute

loop count	number	time_unit
------------	--------	-----------

1	17	second
---	----	--------

10. Inside this loop, use the number and time unit to calculate the total time for the timer, adding 60 seconds for each minute, and the number of seconds for any seconds.

python

```
if time_unit == 'minute':  
    total_seconds += number * 60  
else:  
    total_seconds += number
```

11. Finally, outside this loop through the entities, log the total time for the timer:

python

```
logging.info(f'Timer required for {total_seconds} seconds')
```

12. Run the function app and speak into your IoT device. You will see the total time for the timer in the function app output:

output

```
[2021-06-16T01:38:33.316Z] Executing 'Functions.speech-trigger' (Reason=  
[2021-06-16T01:38:33.329Z] Trigger Details: PartitionId: 0, Offset: 3144-3  
[2021-06-16T01:38:33.605Z] Python EventHub trigger processed an event: {  
[2021-06-16T01:38:35.076Z] Timer required for 257 seconds  
[2021-06-16T01:38:35.128Z] Executed 'Functions.speech-trigger' (Succeeded)
```

 You can find this code in the [code/functions](#) folder.

Challenge

There are many ways to request the same thing, such as setting a timer. Think of different ways to do this, and use them as examples in your LUIS app. Test these out, to see how well your model can cope with multiple ways to request a timer.

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read more about LUIS and it's capabilities on the [Language Understanding_\(LUIS\) documentation page on Microsoft docs](#)
- Read more about language understanding on the [Natural-language understanding page on Wikipedia](#)

Assignment

[Cancel the timer](#)

Set a timer and provide spoken feedback

Add a sketchnote if possible/appropriate

 Embed a video here if available

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

Smart assistants are not one-way communication devices. You speak to them, and they respond:

"Alexa, set a 3 minute timer"

"Ok, your timer is set for 3 minutes"

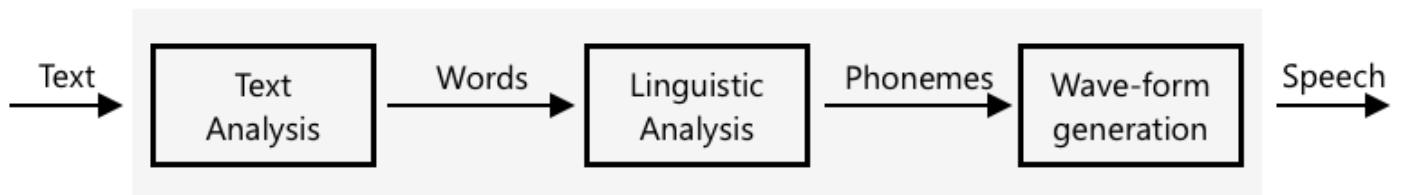
In the last 2 lessons you learned how to take speech and create text, then extract a set timer request from that text. In this lesson you will learn how to set the timer on the IoT device, responding to the user with spoken words confirming their timer, and alerting them when their timer is finished.

In this lesson we'll cover:

- [Text to speech](#)
- [Set the timer](#)
- [Convert text to speech](#)

Text to speech

Text to speech, as the name suggests, is the process of converting text into audio that contains the text as spoken words. The basic principle is to break down the words in the text into their constituent sounds (known as phonemes), and stitch together audio for those sounds, either using pre-recorded audio or using audio generated by AI models.



Text to speech systems typically have 3 stages:

- Text analysis
- Linguistic analysis
- Wave-form generation

Text analysis

Text analysis involves taking the text provided, and converting into words that can be used to generate speech. For example, if you convert "Hello world", there is no text analysis needed, the two words can be converted to speech. If you have "1234" however, then this might need to be converted either into the words "One thousand, two hundred thirty four" or "One, two, three, four" depending on the context. For "I have 1234 apples", then it would be "One thousand, two hundred thirty four", but for "The child counted 1234" then it would be "One, two, three, four".

The words created vary not only for the language, but the locale of that language. For example, in American English, 120 would be "One hundred twenty", in British English it would be "One hundred and twenty", with the use of "and" after the hundreds.

Some other examples that require text analysis include "in" as a short form of inch, and "st" as a short form of saint and street. Can you think of other examples in your language or words that are ambiguous without context.

Once the words have been defined, they are sent for linguistic analysis.

Linguistic analysis

Linguistic analysis breaks the words down into phonemes. Phonemes are based not just on the letters used, but the other letters in the word. For example, in English the 'a' sound in 'car' and 'care' is different. The English language has 44 different phonemes for the 26 letters in the alphabet, some shared by different letters, such as the same phoneme used at the start of 'circle' and 'serpent'!

Do some research: What are the phonemes for your language?

Once the words have been converted to phonemes, these phonemes need additional data to support intonation, adjusting the tone or duration depending on the context. One example is in English pitch increases can be used to convert a sentence into a question, having a raised pitch for the last word implies a question.

For example - the sentence "You have an apple" is a statement saying that you have an apple. If the pitch goes up at the end, increasing for the word apple, it becomes the question "You have an apple?", asking if you have an apple. The linguistic analysis needs to use the question mark at the end to decide to increase pitch.

Once the phonemes have been generated, they can be sent for wave-form generation to produce the audio output.

Wave-form generation

The first electronic text to speech systems used single audio recordings for each phoneme, leading to very monotonous, robotic sounding voices. The linguistic analysis would produce phonemes, these would be loaded from a database of sounds and stitched together to make the audio.

Do some research: Find some audio recordings from early speech synthesis systems. Compare it to modern speech synthesis, such as that used in smart assistants.

More modern wave-form generation uses ML models built using deep learning (very large neural networks that act in a similar way to neurons in the brain) to produce more natural sounding voices that can be indistinguishable from humans.

 Some of these ML models can be re-trained using transfer learning to sound like real people. This means using voice as a security system, something banks are increasingly trying to do, is no longer a good idea as anyone with a recording of a few minutes of your voice can impersonate you.

These large ML models are being trained to combine all three steps into end-to-end speech synthesizers.

Set the timer

The timer can be set by sending a command from the serverless code, instructing the IoT device to set the timer. This command will contain the time in seconds till the timer needs to go off.

Task - set the timer using a command

1. In your serverless code, add code to send a direct method request to your IoT device

 You can refer to the [instructions for sending direct method requests in lesson 5 of the farm project if needed.](#)

You will need to set up the connection string for the IoT Hub with the service policy (*NOT* the device) in your `local.settings.json` file and add the `azure-iot-hub` pip package to your `requirements.txt` file. The device ID can be extracted from the event.

2. The direct method you send needs to be called `set-timer`, and will need to send the length of the timer as a JSON property called `seconds`. Use the following code to build the `CloudToDeviceMethod` using the `total_seconds` calculated from the data extracted by LUIS:

```
python
payload = {
    'seconds': total_seconds
}
direct_method = CloudToDeviceMethod(method_name='set-timer', payload=jsc
```

 You can find this code in the [code-command/functions](#) folder.

Task - respond to the command on the IoT device

1. On your IoT device, respond to the command.

 You can refer to [the instructions for handling direct method requests from IoT devices in lesson 4 of the farm project if needed](#).

2. Work through the relevant guide to set a timer for the required time:

- [Arduino - Wio Terminal](#)
- [Single-board computer - Raspberry Pi/Virtual IoT device](#)

Convert text to speech

The same speech service you used to convert speech to text can be used to convert text back into speech, and this can be played through a speaker on your IoT device. The text to convert is sent to the speech service, along with the type of audio required (such as the sample rate), and binary data containing the audio is returned.

When you send this request, you send it using *Speech Synthesis Markup Language* (SSML), an XML-based markup language for speech synthesis applications. This defines not only the text to be converted, but the language of the text, the voice to use, and can even be used to define speed, volume, and pitch for some or all of the words in the text.

For example, this SSML defines a request to convert the text "Your 3 minute 5 second time has been set" to speech using a British English voice called `en-GB-MiaNeural`

xml

```
<speak version='1.0' xml:lang='en-GB'>
  <voice xml:lang='en-GB' name='en-GB-MiaNeural'>
    Your 3 minute 5 second time has been set
  </voice>
</speak>
```

 Most text to speech systems have multiple voices for different languages, with relevant accents such as a British English voice with an English accent and a New Zealand English voice with a New Zealand accent.

Task - convert text to speech

Work through the relevant guide to convert text to speech using your IoT device:

- [Arduino - Wio Terminal](#)
 - [Single-board computer - Raspberry Pi](#)
 - [Single-board computer - Virtual device](#)
-

Challenge

SSML has ways to change how words are spoken, such as adding emphasis to certain words, adding pauses, or changing pitch. Try some of these out, sending different SSML from your IoT device and comparing the output. You can read more about SSML, including how to change the way words are spoken in the [Speech Synthesis Markup Language \(SSML\) Version 1.1 specification from the World Wide Web consortium](#).

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read more on speech synthesis on the [Speech synthesis page on Wikipedia](#)
- Read more on ways criminals are using speech synthesis to steal on the [Fake voices 'help cyber crooks steal cash' story on BBC news](#)

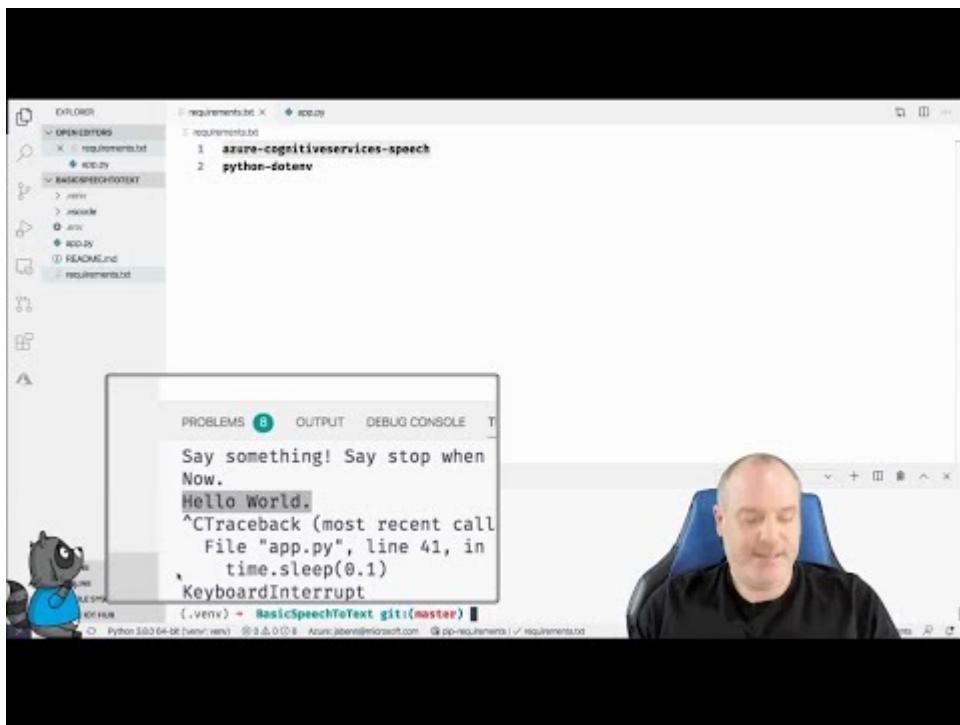
Assignment

[Cancel the timer](#)

Support multiple languages

Add a sketchnote if possible/appropriate

This video gives an overview of the Azure speech services, covering speech to text and text to speech from earlier lessons, as well as translating speech, a topic covered in this lesson:



 Click the image above to watch the video

Pre-lecture quiz

[Pre-lecture quiz](#)

Introduction

In the last 3 lessons you learned about converting speech to text, language understanding, and converting text to speech, all powered by AI. One other area of human communication that AI can

help with is language translation – converting from one language to another, such as from English to French.

In this lesson you will learn about using AI to translate text, allowing your smart timer to interact with users in multiple languages.

In this lesson we'll cover:

- [Translate text](#)
- [Translation services](#)
- [Create a translator resource](#)
- [Support multiple languages in applications with translations](#)
- [Translate text using an AI service](#)

Translate text

Text translation has been a computer science problem that has been researched for over 70 years, and only now thanks to advances in AI and computer power is close to being solved to a point where it is almost as good as human translators.

 The origins can be traced back even further, to [Al-Kindi](#), a 9th century Arabic cryptographer who developed techniques for language translation

Machine translations

Text translation started out as a technology known as Machine Translation (MT), that can translate between different language pairs. MT works by substituting words in one language with another, adding techniques to select the correct ways of translating phrases or parts of sentences when a simple word-for-word translation doesn't make sense.

 When translators support translating between one language and another, these are known as language pairs. Different tools support different language pairs, and these may not be complete. For example, a translator may support English to Spanish as a language pair, and Spanish to Italian as a language pair, but not English to Italian.

For example, translating "Hello world" from English into French can be performed with a substitution - "Bonjour" for "Hello", and "le monde" for "world", leading to the correct translation of "Bonjour le monde".

Substitutions don't work when different languages use different ways of saying the same thing. For example, the English sentence "My name is Jim", translates into "Je m'appelle Jim" in French - literally "I call myself Jim". "Je" is French for "I", "moi" is me, but is concatenated with the verb as it starts with a vowel, so becomes "m'", "appelle" is to call, and "Jim" isn't translated as it's a name, and not a word that can be translated. Word ordering also becomes an issue - a simple substitution of "Je m'appelle Jim" becomes "I myself call Jim", with a different word order to English.

 Some words are never translated - my name is Jim regardless of which language is used to introduce me.

Idioms are also a problem for translation. These are phrases that have an understood meaning that is different from a direct interpretation of the words. For example, in English the idiom "I've got ants in my pants" does not literally refer to having ants in your clothing, but to being restless. If you translated this to German, you would end up confusing the listener, as the German version is "I have bumble bees in the bottom".

 Different locales add different complexities. With the idiom "ants in your pants", in American English "pants" refers to outerwear, in British English, "pants" is underwear.

 If you speak multiple languages, think of some phrases that don't directly translate

Machine translation systems rely on large databases of rules that describe how to translate certain phrases and idioms, along with statistical methods to pick the appropriate translations from possible options. These statistical methods use huge databases of works translated by humans into multiple languages to pick the most likely translation, a technique called *statistical machine translation*. A number of these use intermediate representations of the language, allowing one language to be translated to the intermediate, then from the intermediate to another language. This way adding more languages involves translations to and from the intermediate, not to and from all other languages.

Neural translations

Neural translations involve using the power of AI to translate, typically translating entire sentences using one model. These models are trained on huge data sets that have been human translated, such

as web pages, books and United Nations documentation.

Neural translation models are usually smaller than machine translation models due to not needing huge databases of phrases and idioms. Modern AI services that provide translations often mix multiple techniques, mixing statistical machine translation and neural translation

There is no 1:1 translation for any language pair. Different translation models will produce slightly different results depending on the data used to train the model. Translations are not always symmetrical - in that if you translate a sentence from one language to another, then back to the first language you may see a slightly different sentence as the result.

- ✓ Try out different online translators such as [Bing Translate](#), [Google Translate](#), or the Apple translate app. Compare the translated versions of a few sentences. Also try translating in one, then translating back in another.

Translation services

There are a number of AI services that can be used from your applications to translate speech and text.

Cognitive services Speech service



The speech service you've been using over the past few lessons has translation capabilities for speech recognition. When you recognize speech, you can request not only the text of the speech in the same language, but also in other languages.

 This is only available from the speech SDK, the REST API doesn't have translations built in.

Cognitive services Translator service



The Translator service is a dedicated translation service that can translate text from one language, to one or more target languages. As well as translating, it supports a wide range of extra features including masking profanity. It also allows you to provide a specific translation for a particular word or sentence, to work with terms you don't want translated, or have a specific well-known translation.

For example, when translating the sentence "I have a Raspberry Pi", referring to the single-board computer, into another language such as French, you would want to keep the name "Raspberry Pi" as is, and not translate it, giving "J'ai un Raspberry Pi" instead of "J'ai une pi aux framboises".

Create a translator resource

For this lesson you will need a Translator resource. You will use the REST API to translate text.

Task - create a translator resource

1. From your terminal or command prompt, run the following command to create a translator resource in your `smart-timer` resource group.

```
sh
az cognitiveservices account create --name smart-timer-translator \
--resource-group smart-timer \
--kind TextTranslation \
--sku F0 \
--yes \
--location <location>
```

Replace `<location>` with the location you used when creating the Resource Group.

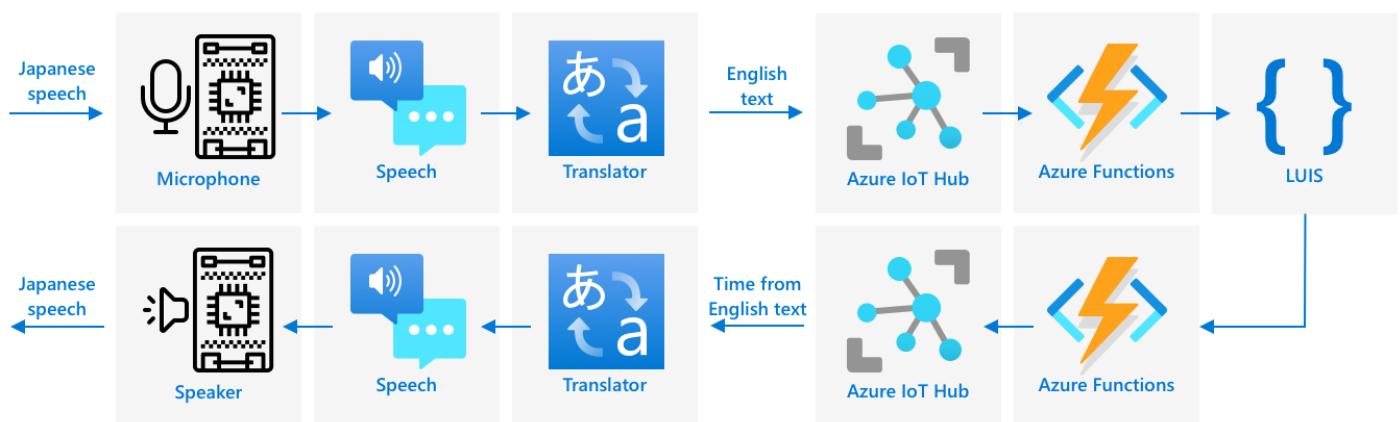
2. Get the key for the translator service:

```
sh
az cognitiveservices account keys list --name smart-timer-translator \
--resource-group smart-timer \
--output table
```

Take a copy of one of the keys.

Support multiple languages in applications with translations

In an ideal world, your whole application should understand as many different languages as possible, from listening for speech, to language understanding, to responding with speech. This is a lot of work, so translation services can speed up the time to delivery of your application.



A smart timer architecture translating Japanese to English, processing in English then translating back to Japanese. Microcontroller by Template / recording by Aybige Speaker / Speaker by Gregor Cresnar - all from the [Noun Project](#)

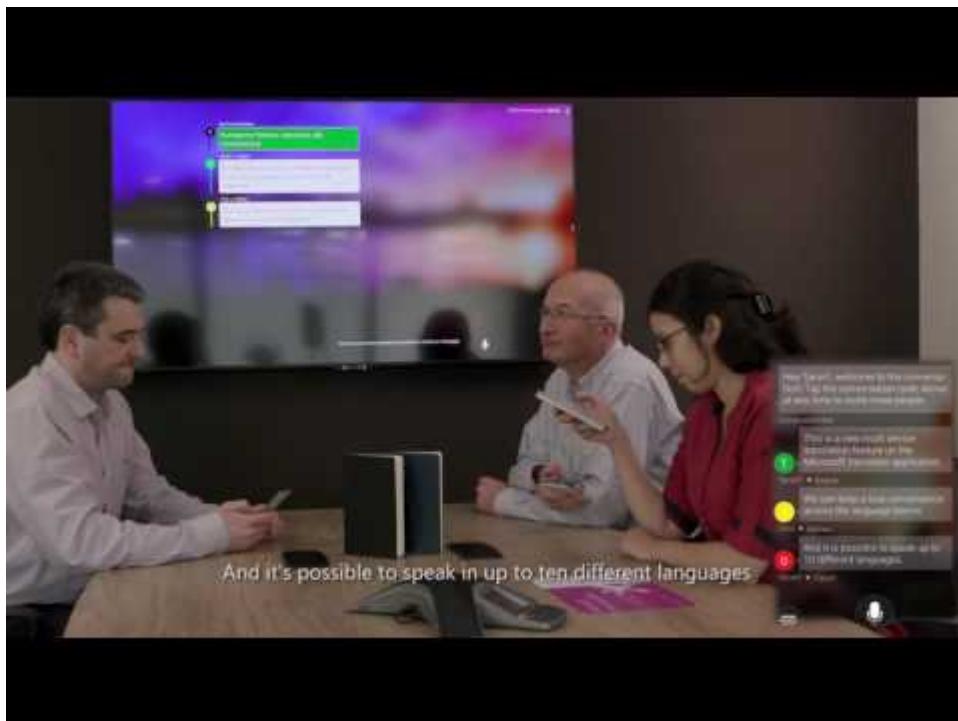
Imagine you are building a smart timer that uses English end-to-end, understanding spoken English and converting that to text, running the language understanding in English, building up responses in English and replying with English speech. If you wanted to add support for Japanese, you could start with translating spoken Japanese to English text, then keep the core of the application the same, then translate the response text to Japanese before speaking the response. This would allow you to quickly add Japanese support, and you can expand to providing full end-to-end Japanese support later.

 The downside to relying on machine translation is that different languages and cultures have different ways of saying the same things, so the translation may not match the expression you are expecting.

Machine translations also open up possibilities for apps and devices that can translate user-created content as it is created. Science fiction regularly features 'universal translators', devices that can

translate from alien languages into (typically) American English. These devices are less science fiction, more science fact, if you ignore the alien part. There are already apps and devices that provide real-time translation of speech and written text, using combinations of speech and translation services.

One example is the [Microsoft Translator](#) mobile phone app, demonstrated in this video:



🎥 Click the image above to watch the video

Imagine having such a device available to you, especially when travelling or interacting with folks whose language you don't know. Having automatic translation devices in airports or hospitals would provide much needed accessibility improvements.

✓ Do some research: Are there any translation IoT devices commercially available? What about translation capabilities built into smart devices?

👀 Although there are no true universal translators that allow us to talk to aliens, the [Microsoft translator does support Klingon](#). Qapla'!

Translate text using an AI service

You can use an AI service to add this translation capability to your smart timer.

Task - translate text using an AI service

Work through the relevant guide to convert translate text on your IoT device:

- [Arduino - Wio Terminal](#)
 - [Single-board computer - Raspberry Pi](#)
 - [Single-board computer - Virtual device](#)
-

Challenge

How can machine translations benefit other IoT applications beyond smart devices? Think of different ways translations can help, not just with spoken words but with text.

Post-lecture quiz

[Post-lecture quiz](#)

Review & Self Study

- Read more on machine translation on the [Machine translation page on Wikipedia](#)
- Read more on neural machine translation on the [Neural machine translation page on Wikipedia](#)
- Check out the list of supported languages for the Microsoft speech services in the [Language and voice support for the Speech service documentation on Microsoft Docs](#)

Assignment

[Build a universal translator](#)