



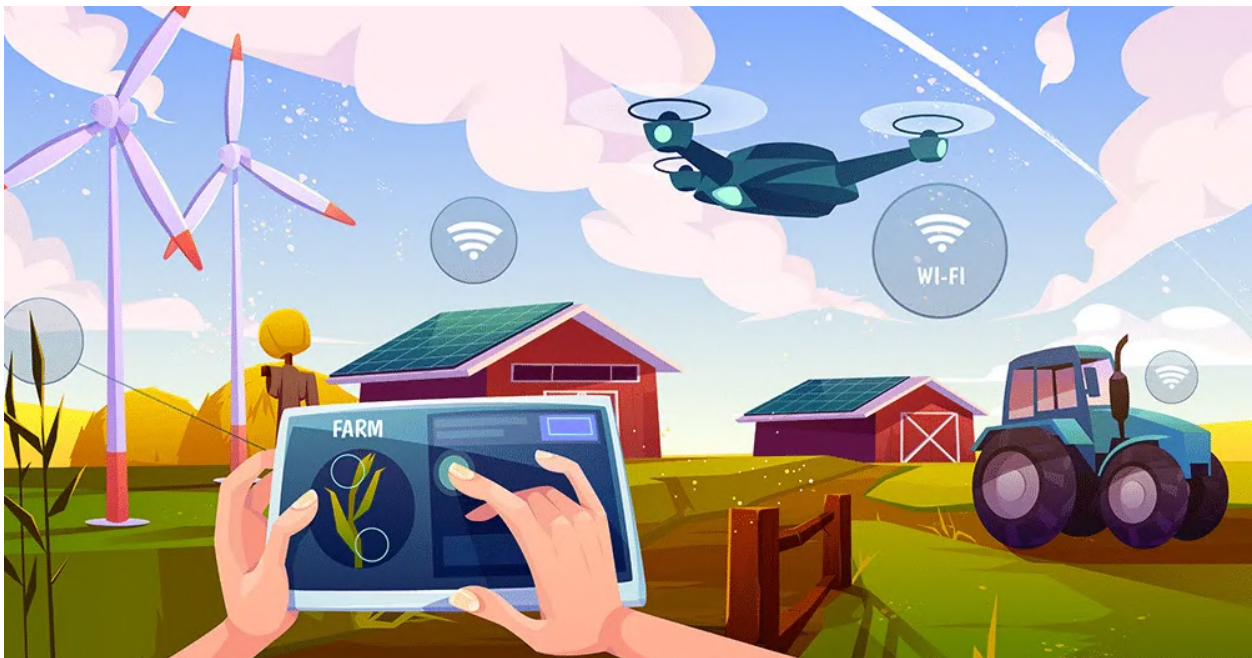
IoT-Enabled Agricultural Prospects Using Microsoft Azure and Python

I. Introduction

✓ Definition of IoT and its potential in the agricultural industry

The internet-**primarily** based interconnectedness of **bodily gadgets** like **home equipment** and **devices** is **referred to as** the Internet of Things (IoT). These **gadgets** can **gather** and **change statistics** with **each other way** to sensors and software. They are **consequently capable of** collaborating and **speaking** with **each other**.

The Internet of Things has **an enormous capacity within the** agricultural sector. Sensors, for example, **may be used to hold an eye fixed on matters just like the** temperature of the air, how **vegetation** grows, and **what kind of** moisture is **inside the soil**. After that, **those statistics may be used to make irrigation structures paintings higher** and **practice the proper quantity of fertiliser at the adequate time to grow** crop yields and use higher **sources**.



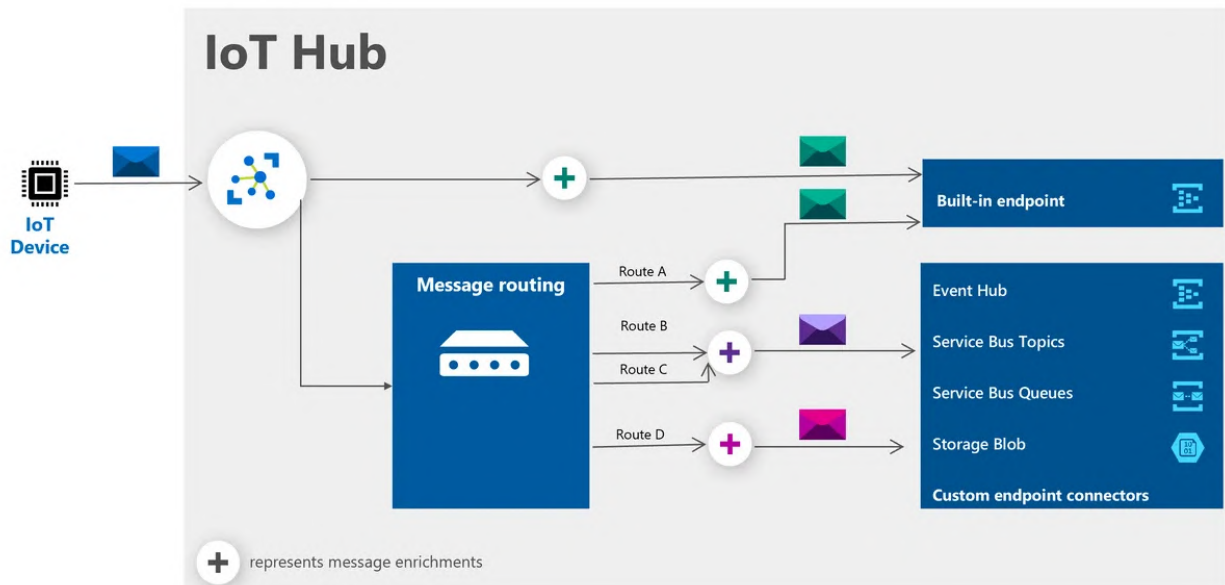
credit: www.hestabit.com

The **inherent character** of the Internet of Things to **tune farm animals** permits farmers to display the fitness and moves of their farm animals constantly. As a result, farmers **can** feed and breed their animals **extra** effectively, which **can be helpful to** their welfare.

Precision farming, **wherein the era** is used to manage pesticides, fertilisers, and seeds precisely, is another **capacity IoT utility** in agriculture. As a result, farming **methods may also** require fewer **sources and feature** fewer **adverse effects on the** environment.

The agricultural sector's adoption of the Internet of Things (IoT) has the **potential to boost productivity, sustainability, and efficiency noticeably**. IoT **applications** in agriculture will **become** even **more innovative** and **significant in destiny** as the **era** advances.

Overview of Microsoft Azure and its capabilities for IoT



credit: techcommunity.microsoft.com

Microsoft Azure is a carrier such as a couple of gears for cloud computing structures and infrastructures for building, deploying, and handling applications and services via an international network of data centres managed with the helpful resource of Microsoft. It offers computing, analytics, storage, networking, and numerous exceptional cloud services. These services can be determined on and mounted through the use of clients to fulfil their specific requirements.

Support for the Internet of Things (IoT) is an essential feature of Microsoft Azure. Azure IoT is a cloud providing for building Internet of Things applications. It makes it possible for businesses to securely connect, control, and look at the data produced by IoT sensors and devices.

For building IoT solutions, Azure IoT offers numerous services, including:

IoT Hub in Azure: a managed company for bidirectional device communicate that serves as a critical message hub. Device manipulation and provisioning are made possible similarly to strong device connection and communicate.

IoT Edge in Azure is a company that lets clients run custom code and Azure services without delay on IoT devices. Data processing and assessment can now be completed on edge even as now no longer have to deliver numerous data to the cloud.

IoT Central in Azure: a Software as a Service (SaaS) solution with entire manipulation that makes it clean to connect, monitor, and control IoT devices. It gives clients the proper access to a web-based dashboard and pre-built connectors for uncommon Internet of Things protocols, making it smooth for them to begin with little coding.

Azure moreover provides numerous gadgets and services for data storage, assessment, and visualisation, encompassing Power BI, Azure Stream Analytics, and Azure Time Series Insights. Data-driven alternatives and insights from IoT device data can be obtained with the help of that gadget.

In general, businesses can securely connect, control, and look at data from a large shape of sensors and devices with Microsoft Azure's sizable set of gadgets and services for building and deploying IoT solutions.

Introduction to Python and its role in IoT development



Credit: midjourney.com

The high-stage programming language Python is famous for its ease of use, clarity, and adaptability. Applications like internet improvement, medical computing, records analysis, and synthetic intelligence employ it.

Python plays an important position withinside the advent of IoT programs withinside the Internet of Things (IoT) world. Python is an excellent preference for IoT improvement for the following reasons:

The Python network is extensive and energetic: Python has a significant and vibrant developer network that contributes to its improvement and protection since it is far from an open-supply language. This shows that Python builders have to get the right of entry to several assets and assistance.

Learning Python is easy: Python's syntax is simple and easy, emphasising clarity and decreasing the price of application maintenance. It is a fantastic programming language for beginners and skilled programmers who need to fast prototype and take a look at concepts.

Python can do a lot: Web servers, computer programs, and medical simulations are only a few of the various programs that may be constructed with Python. Because of its adaptability, it is an excellent preference for IoT improvement as it permits builders to assemble and combine various sensors and devices.

IoT protocols are adequately supported through Python: Some libraries and frameworks blanketed in Python make it easy to connect with and speak with IoT devices. The "paho-MQTT" library may connect with MQTT brokers, and the Python library "pyserial" can speak with serial devices.

In a nutshell, Python is a properly-applicable programming language for the Internet of Things. It is a fantastic preference for growing IoT programs because of its simplicity, adaptability, and giant aid for IoT protocols.

```
import serial
import paho.mqtt.client as mqtt
import numpy as np
import matplotlib.pyplot as plt

#Connecting to the serial port
ser = serial.Serial('/dev/ttyACM0', 9600)

#Connecting to the MQTT broker
client = mqtt.Client()
client.connect("", 1883)
```

```

#Subscribing to the MQTT topic
client.subscribe("/sensor/data")

#Creating a NumPy array to store the sensor data
data = np.array([])

#Defining a function to plot the sensor data
def plot_data():
    plt.plot(data)
    plt.show()

#Defining a function to handle the MQTT messages
def on_message(client, userdata, message):
    #Reading the sensor data from the serial port
    sensor_data = ser.readline()
    #Converting the sensor data to a float
    sensor_data = float(sensor_data)
    #Appending the sensor data to the NumPy array
    data = np.append(data, sensor_data)
    #Printing the sensor data
    print(sensor_data)
    #Plotting the sensor data
    plot_data()

#Setting the on_message function as the message callback
client.on_message = on_message

#Looping to handle the MQTT messages
client.loop_forever()

```

II. IoT in Agricultural Applications



Examples of IoT-based solutions for the agricultural industry

By enabling farmers and agribusinesses to collect and analyse data from various sensors and devices, the Internet of Things (IoT) can transform agribusiness. This data can improve multiple aspects of agriculture, including crop yields, water use, and livestock management.

The following are some examples of IoT -based solutions currently in use or maybe in service soon:

1. Precision Farming:

Data on soil conditions, crop growth and weather patterns can be collected through precision farming using drones, sensors and other IoT Devices. Machine learning algorithms are used to analyse this data to improve irrigation, fertilisation and pest control performance. Farmers can increase crop yields and save money on pesticides and water by adopting precision farming methods.



Credit: www.bearingtips.com

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Load the training data from a CSV file
data = np.loadtxt("training_data.csv", delimiter=",")

# Split the data into input features and output labels
X = data[:,0:2]
y = data[:,2]

# Train a linear regression model
model = LinearRegression()
model.fit(X, y)

# Use the trained model to make predictions on new data
predictions = model.predict([[5, 6], [6, 7]])
print(predictions)

# Save the trained model to a file so we can use it to make predictions later
import pickle
pickle.dump(model, open("model.pkl", "wb"))

# Load the model from the file
model = pickle.load(open("model.pkl", "rb"))

# Use the model to make predictions
predictions = model.predict([[5, 6], [6, 7]])
print(predictions)
```

The code trains a linear regression model on a dataset with two input features (e.g. soil moisture and temperature) and one output label (e.g. crop yield).

2. **Monitoring of farm animals:** The health and behaviour of farm animals can be monitored using Internet of Things (IoT) devices such as cameras and sensors that place them in the body. This data can be used to improve feeding and breeding schedules and spot early signs of illness or stress. Farmers can use IoT-based livestock monitoring systems to increase productivity and enhance the welfare of their animals.



Credit: www.nationalhogfarmer.com

```
import paho.mqtt.client as mqtt
import datetime

# Define the MQTT callback functions
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe("my/topic")

def on_message(client, userdata, msg):
    # Parse the incoming message as a JSON object
    data = json.loads(msg.payload)

    # Extract the sensor data from the message
    animal_id = data['animal_id']
    temperature = data['temperature']
    heart_rate = data['heart_rate']
    timestamp = data['timestamp']

    # Print the sensor data to the console
    print("Animal ID: " + animal_id)
    print("Temperature: " + str(temperature) + "C")
    print("Heart rate: " + str(heart_rate) + "bpm")
    print("Timestamp: " + str(datetime.datetime.fromtimestamp(timestamp)))

# Create an MQTT client and connect to the Azure IoT Hub
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.username_pw_set("<IoT Hub connection string>")
client.connect("<IoT Hub hostname>.azure-devices.net", 8883, 60)

# Start the MQTT loop to listen for messages
client.loop_forever()
```

Run the Python script from the command line:

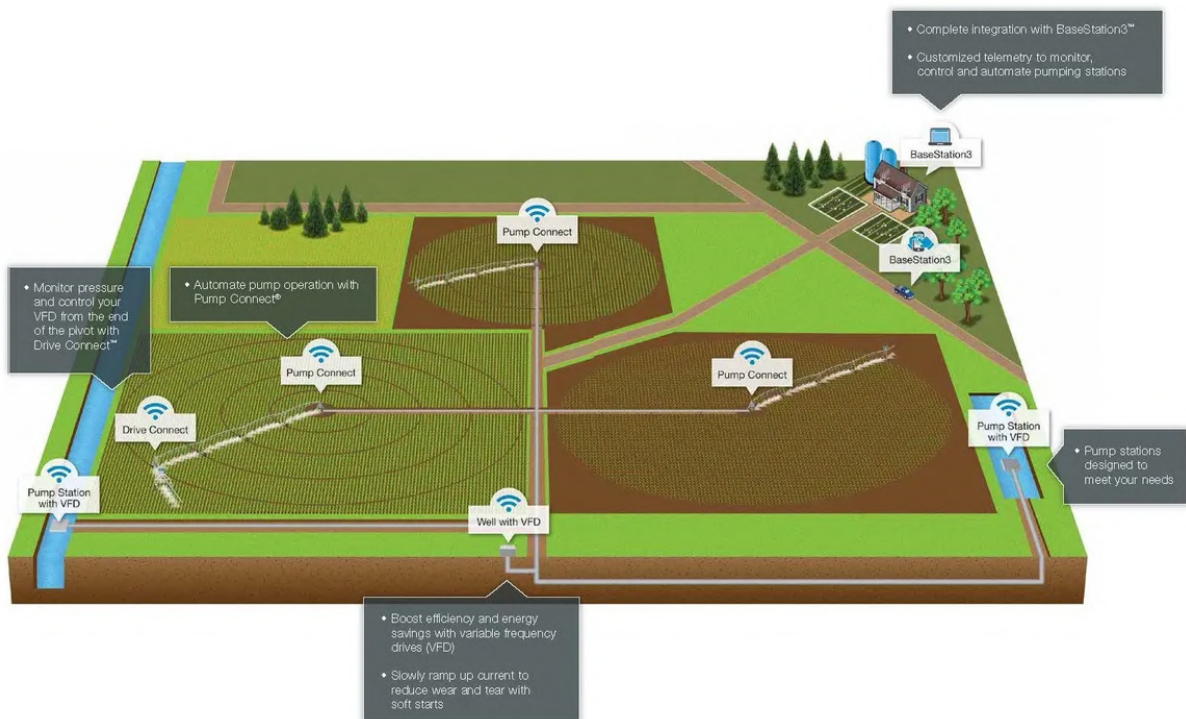
```
python3 mqtt_listener.py
```

You should see the following output:

```
Connected with result code 0
Animal ID: cow-1
Temperature: 38.5C
Heart rate: 72bpm
Timestamp: 2023-01-10 14:00:00
```

i This code sets up an MQTT client that subscribes to a topic on the Azure IoT Hub and receives messages containing

- Irrigation Management:** It is possible to optimise irrigation schedules and save water with IoT devices such as weather stations and soil moisture sensors. Farmers can increase crop yields and reduce water consumption by using IoT-based irrigation systems.



```
import requests
import json
import paho.mqtt.client as mqtt

# Define the MQTT callback functions
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe("my/topic")

def on_message(client, userdata, msg):
    # Parse the incoming message as a JSON object
    data = json.loads(msg.payload)

    # Extract the soil moisture data from the message
    soil_moisture = data['soil_moisture']

    # If the soil moisture is below a threshold, send a command to the irrigation system
```

```

if soil_moisture < 20:
requests.post("http://irrigation.local/on")
else:
requests.post("http://irrigation.local/off")

# Create an MQTT client and connect to the Azure IoT Hub
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.username_pw_set("<IoT Hub connection string>")
client.connect("<IoT Hub hostname>.azure-devices.net", 8883, 60)

# Start the MQTT loop to listen for messages
client.loop_forever()

```

Now to Test the application

1. Run the application on Raspberry Pi.
2. Use the Azure IoT Hub Device Explorer to send a message to the device.

```

{
"soil_moisture": 10
}

```


3. The irrigation system should turn on.
4. Use the Azure IoT Hub Device Explorer to send a message to the device.

```

{
"soil_moisture": 50
}

```

5. The irrigation system should turn off.

 This code sets up an MQTT client that subscribes to a topic on the Azure IoT Hub and receives messages containing soil moisture data. When the soil moisture falls below a certain threshold, the code sends a command to an irrigation system to turn on. When the soil moisture rises above the threshold, the code sends a command to turn off the irrigation system. This can help optimise irrigation schedules and save water.

3. **Supply chain traceability:** The movement of agricultural products through the supply chain can be tracked using Internet of Things (IoT) devices such as RFID tags and QR codes. This can reduce inefficiencies in food safety, traceability and supply chain.



Credit: sustainablebrands.com

```
import os
import time
import json
import logging
import azure.functions as func
from azure.iot.device import IoTHubDeviceClient, Message

# Set up logging
logging.basicConfig(level=logging.INFO)
# Set up Azure IoT hub connection
CONNECTION_STRING = os.environ["IOT_HUB_CONNECTION_STRING"]
DEVICE_ID = os.environ["IOT_HUB_DEVICE_ID"]

# Set up the client
client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)

# Define a function to send data to Azure IoT hub
def send_data(data):
    message = Message(json.dumps(data))
    client.send_message(message) logging.info("Data sent to Azure IoT hub: {}".format(data))
# Set up a loop to continuously collect and send data from the IoT device
while True:
    # Collect data from the IoT device
    data = {"temperature": 25, "humidity": 60} # Replace with actual data from the device

    # Send the data to Azure IoT hub
    send_data(data)

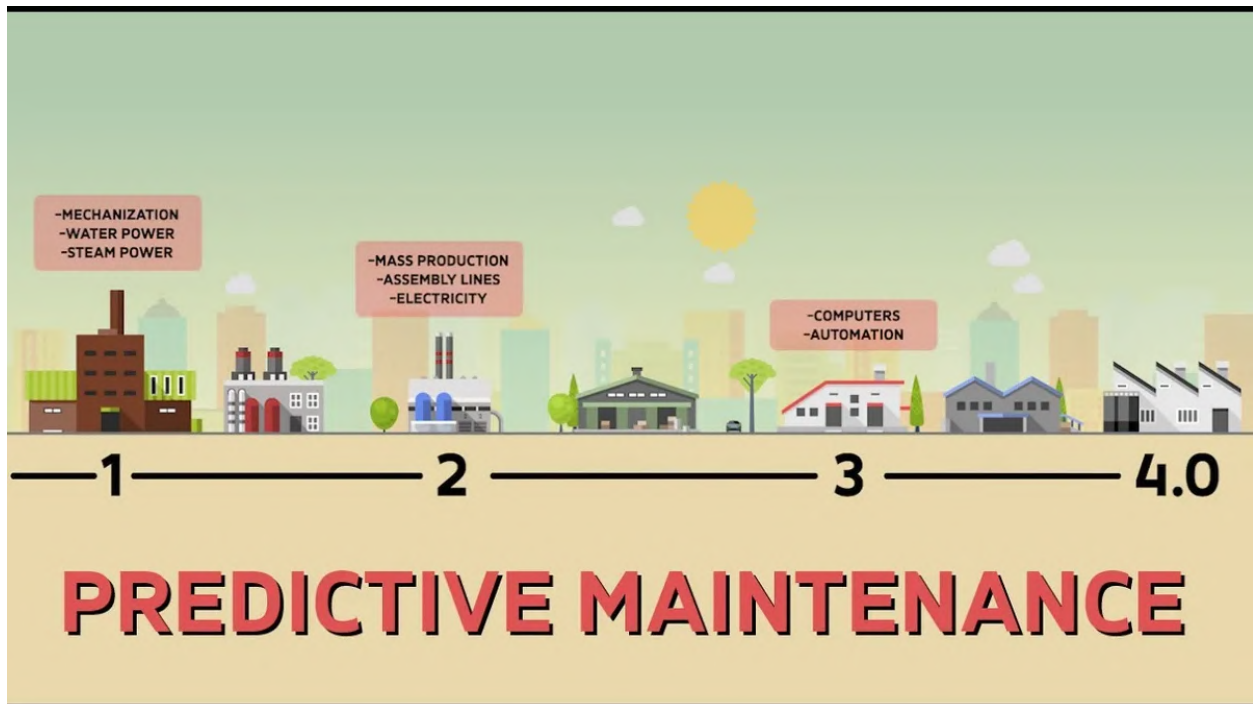
    # Wait for a minute before collecting and sending more data
    time.sleep(60)

# Disconnect from Azure IoT hub
client.disconnect()
```

i This code sets up a connection to an Azure IoT hub using the `IoTHubDeviceClient` class and defines a function `send_data` to send data to the hub. It then enters a loop where it continuously collects data from the IoT device (in this case, we are just using some dummy data) and sends it to the hub using the `send_data` function.

i You can then use Azure services such as Azure Stream Analytics, Azure Functions, and Azure Table Storage to store, process, and visualise the data being sent from the IoT device. This can be used to create a supply chain traceability system for agriculture by tracking the data related to various stages of the supply chain, such as the location and environmental conditions of the crops as they are grown, transported, and sold.

4. **Predictive maintenance:** It is possible to use IoT devices as sensors and telemetry to foresee when agricultural machinery will fail. This can help farmers reduce downtime and streamline maintenance schedules.



credit: thedatascientist.com

Farmers and agribusinesses can increase productivity, reduce costs and improve efficiency by harnessing the power of the Internet of Things.

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
import paho.mqtt.client as mqtt
import json

# Load the training data from a CSV file
data = np.loadtxt("training_data.csv", delimiter=",")

# Split the data into input features and output labels
X = data[:,0:5]
y = data[:,5]

# Train a random forest classifier
model = RandomForestClassifier()
model.fit(X, y)

# Define the MQTT callback functions
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe("my/topic")

def on_message(client, userdata, msg):
    # Parse the incoming message as a JSON object
    data = json.loads(msg.payload)

    # Extract the telemetry data from the message
    telemetry_data = data['telemetry_data']

    # Use the trained model to make a prediction on the telemetry data
    prediction = model.predict([telemetry_data])

    # If the prediction is positive, send an alert to the
    # "alerts" topic
```

```
if prediction > 0:
    client.publish("alerts", "Alert!")

# Create an MQTT client
client = mqtt.Client()

# Set the callback functions
client.on_connect = on_connect
client.on_message = on_message

# Connect to the MQTT broker
client.connect("localhost", 1883, 60)

# Start the MQTT client loop
client.loop_forever()
```

i The above code trains a random forest classifier on a dataset with five input features and one output label and uses the trained model to make predictions on incoming telemetry data received via MQTT messages from the Azure IoT Hub, sending an alert if the prediction is positive.

These are just a few use cases of IoT-based solutions in the agriculture sector that currently exist or have the potential to develop.

Benefits of using IoT in agriculture, including increased efficiency, cost savings, and improved crop yields



The farming industry could be transformed thanks to the Internet of Things (IoT), which can boost productivity, reduce prices, and increase crop yields.

Among the ways, IoT is altering using sensors and other monitoring systems to improve farm efficiency. For instance, the sensors can track soil moisture and give farmers up-to-the-minute information on irrigation requirements. Farmers may be able to reduce water waste and optimise their

irrigation schedules.

IoT can lower the cost of farming by decreasing the need for labour.

For instance, drones and autonomous tractors in precision farming can displace human workers for tasks like planting and watering crops.

IoT, in addition to enhancing efficiency and lowering costs, can also help to improve crop yields. For example, the sensors can monitor crop growth and health in real-time, allowing farmers to identify and address any issues that may arise quickly.

```
# This code sets up an IoT sensor network to monitor soil moisture levels and stores the data in Microsoft Azure

import sensor
import time
import azure.storage.blob as azureblob

def main():
    # Set up a connection to the Microsoft Azure storage account
    storage_account_name = "mystorageaccount"
    storage_account_key = "mystorageaccountkey"
    container_name = "soil-moisture-data"
    blob_service = azureblob.BlockBlobService(account_name=storage_account_name, account_key=storage_account_key)

    # Initialise the soil moisture sensor
    soil_moisture_sensor = sensor.SoilMoistureSensor()

    while True:
        # Read the soil moisture level from the sensor
        soil_moisture = soil_moisture_sensor.read()

        # Save the soil moisture data to a file
        with open('soil_moisture_data.txt', 'a') as f:
            f.write(str(soil_moisture) + "\n")

        # Upload the soil moisture data file to Microsoft Azure
        azureblob.upload_file_to_container(blob_service, container_name, 'soil_moisture_data.txt')

        # Wait 1 hour before taking another reading
        time.sleep(3600)

if name == 'main':
    main()
```

Run the code

- ✓ Run the code on the Raspberry Pi. `python3 soil_moisture_monitor.py`
- ✓ After the code has been running for a while, you can check the data in the Microsoft Azure storage account.

Analyse the data

- ✓ Download the data from the Microsoft Azure storage account.
- ✓ Open the data in a spreadsheet program.
- ✓ Create a line graph of the data.
- ✓ Analyse the data to determine the optimal time to water the plants.
- ✓ Automate the watering

Create a new Python script to automate the watering.

```
import time
import RPi.GPIO as GPIO

def main():
    # Set up the GPIO pins
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(17, GPIO.OUT)

    # Turn on the water pump
    GPIO.output(17, GPIO.HIGH)

    # Wait for 5 seconds
```

```

time.sleep(5)

# Turn off the water pump
GPIO.output(17, GPIO.LOW)

# Clean up the GPIO pins
GPIO.cleanup()

if name == 'main':
    main()

```

✓ Run the code on the Raspberry Pi. `python3 water_plants.py`

✓ Test the water pump to make sure it works.

Add the code to the soil moisture monitor to automate the watering.

```

import sensor
import time
import azure.storage.blob as azureblob
import RPi.GPIO as GPIO

def main():
    # Set up a connection to the Microsoft Azure storage account
    storage_account_name = "mystorageaccount"
    storage_account_key = "mystorageaccountkey"
    container_name = "soil-moisture-data"
    blob_service = azureblob.BlockBlobService(account_name=storage_account_name, account_key=storage_account_key)

    # Initialise the soil moisture sensor
    soil_moisture_sensor = sensor.SoilMoistureSensor()

    # Set up the GPIO pins
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(17, GPIO.OUT)

    while True:
        # Read the soil moisture level from the sensor
        soil_moisture = soil_moisture_sensor.read()

        # Save the soil moisture data to a file
        with open('soil_moisture_data.txt', 'a') as f:
            f.write(str(soil_moisture) + "\n")

        # Upload the soil moisture data file to Microsoft Azure
        azureblob.upload_file_to_container(blob_service, container_name, 'soil_moisture_data.txt')

        # Check if the soil moisture level is below a certain threshold
        if soil_moisture < 500:
            # Turn on the water pump
            GPIO.output(17, GPIO.HIGH)

            # Wait for 5 seconds
            time.sleep(5)

            # Turn off the water pump
            GPIO.output(17, GPIO.LOW)

            # Wait 1 hour before taking another reading
            time.sleep(3600)

    if name == 'main':
        main()

```

✓ Run the code on the Raspberry Pi. `python3 soil_moisture_monitor.py`

✓ Test the automated watering system.

Create a dashboard

Create a new Python script to create a dashboard.

```

import matplotlib.pyplot as plt
import pandas as pd

def main():

```

```

# Read the data from the file
data = pd.read_csv('soil_moisture_data.txt', header=None)

# Create a line graph of the data
plt.plot(data)
plt.show()

if name== 'main':
main()

```

2. Run the code on the Raspberry Pi. `python3 dashboard.py`

- ✓ Test the dashboard.
- ✓ Create a web app

Create a new Python script to create a web app.

```

import matplotlib.pyplot as plt
import pandas as pd
from flask import Flask, render_template

app = Flask(name)

@app.route('/')
def index():
# Read the data from the file
data = pd.read_csv('soil_moisture_data.txt', header=None)

# Create a line graph of the data
plt.plot(data)
plt.savefig('static/soil_moisture_data.png')

# Render the web page
return render_template('index.html')

if name== 'main':
app.run(host='0.0.0.0', port=80, debug=True)

```

2. Create a new HTML file to create a web page.

```

<!DOCTYPE html>
<html>
<head>
<title>Soil Moisture Monitor</title>
</head>
<body>
<h1>Soil Moisture Monitor</h1>

</body>
</html>

```

Run the code on the Raspberry Pi. `python3 web_app.py`

- ✓ Test the web app.
- ✓ Deploy the web app

Create a new Python script to deploy the web app.

```

import matplotlib.pyplot as plt
import pandas as pd
from flask import Flask, render_template
from waitress import serve

app = Flask(name)
@app.route('/')
def index():
# Read the data from the file
data = pd.read_csv('soil_moisture_data.txt', header=None)
# Create a line graph of the data
plt.plot(data)
plt.savefig('static/soil_moisture_data.png')

```

```
# Render the web page
return render_template('index.html')
if name == ' main':
serve(app, host='0.0.0.0', port=80)
...
```

(EXECUTE THE CODE)

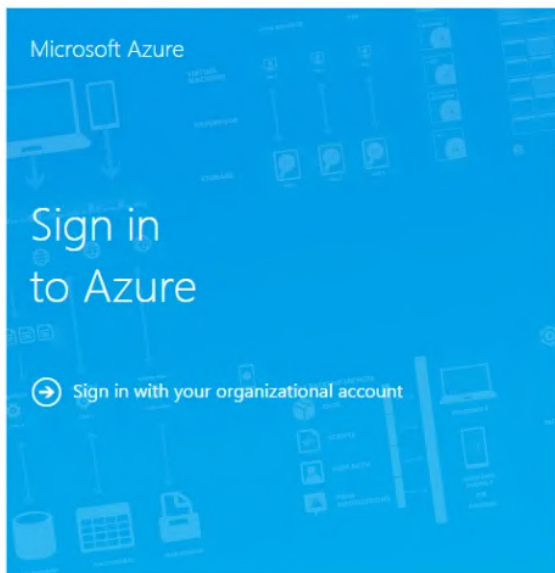
This code imports the necessary libraries and sets up a connection to a Microsoft Azure storage account using the storage account name and key. It then initialises a soil moisture sensor and enters a loop to continuously read the soil moisture level and store the data. The soil moisture data is saved to a local file and then uploaded to the specified container in the Microsoft Azure storage account. The code then waits for 1 hour before taking another reading.

III. Implementing an IoT Solution with Microsoft Azure and Python

⚙️ Setting up an Azure IoT Hub and connecting devices

To set up an Azure IoT Hub and connect devices, follow these steps:

1. Sign in to Azure portal (<https://portal.azure.com/>) with your Microsoft account.



Sign in

Microsoft account [What's this?](#)

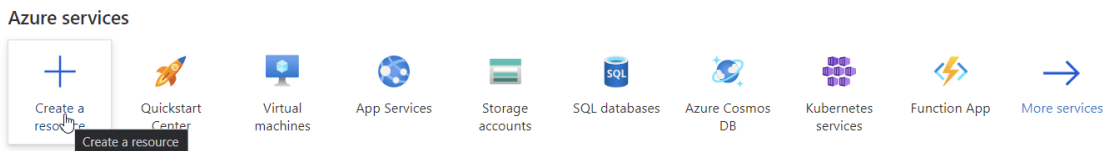
Keep me signed in

Sign in

Can't access your account?
[Sign in with a single-use code](#)

Don't have a Microsoft account? [Sign up now](#)

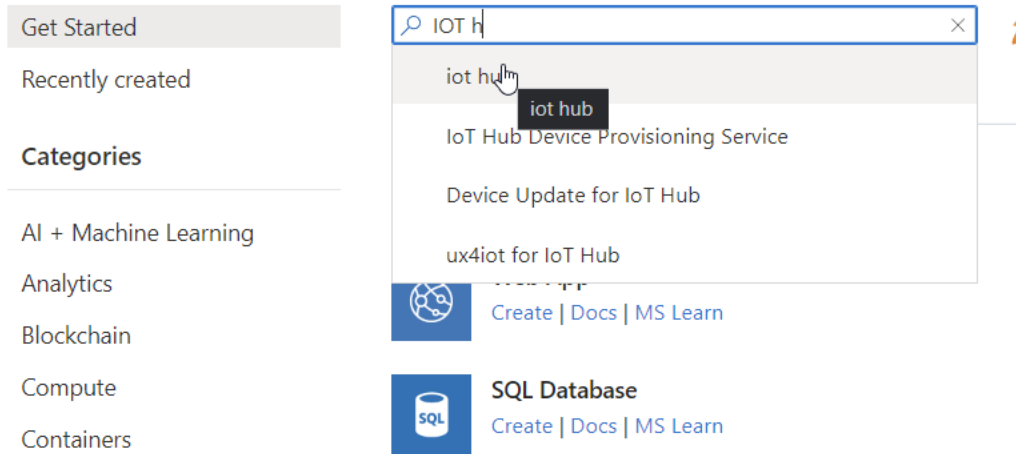
2. Click on "Create a resource" button in the top left corner of the portal.



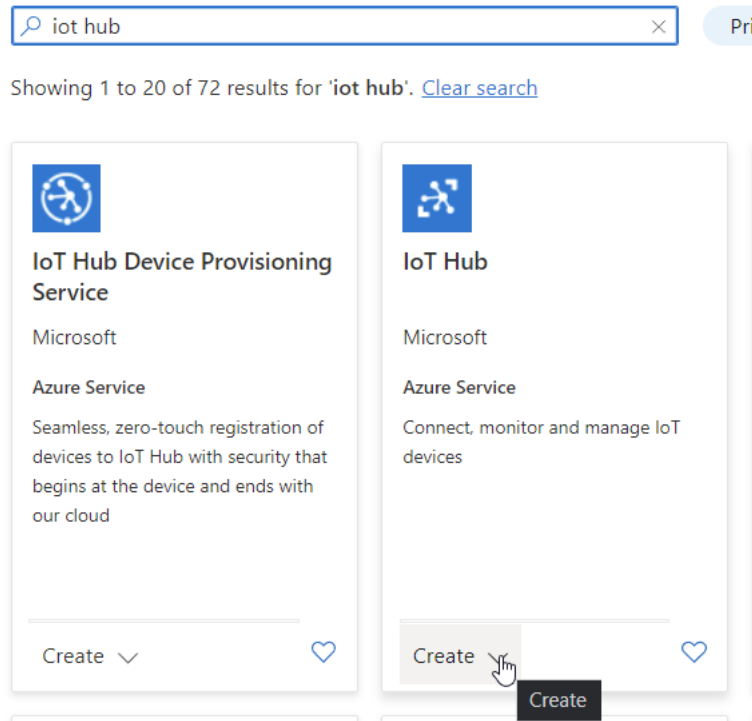
3. In the "Search the Marketplace" field, search for "IoT Hub".

Home >

Create a resource ...

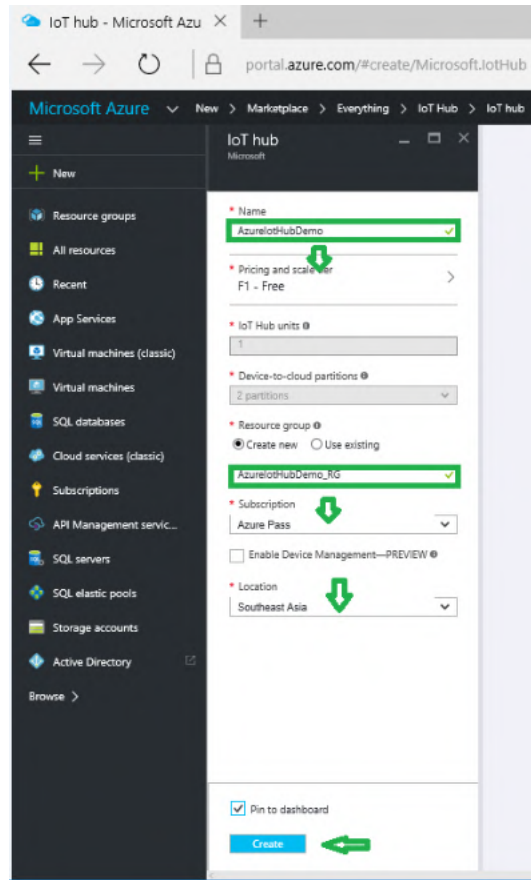


4. Click on the "IoT Hub" result and then click the "Create" button.

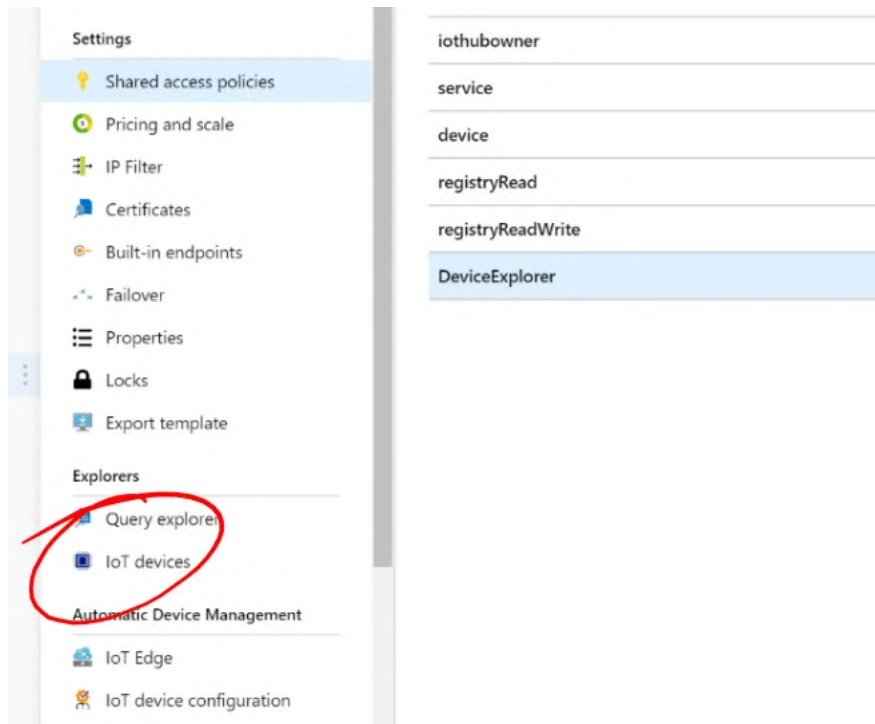


5. In the "Create IoT Hub" blade, enter a name for your IoT hub and select a resource group. You can also choose the pricing and scale tier, as well as the number of IoT Edge devices you want to support.

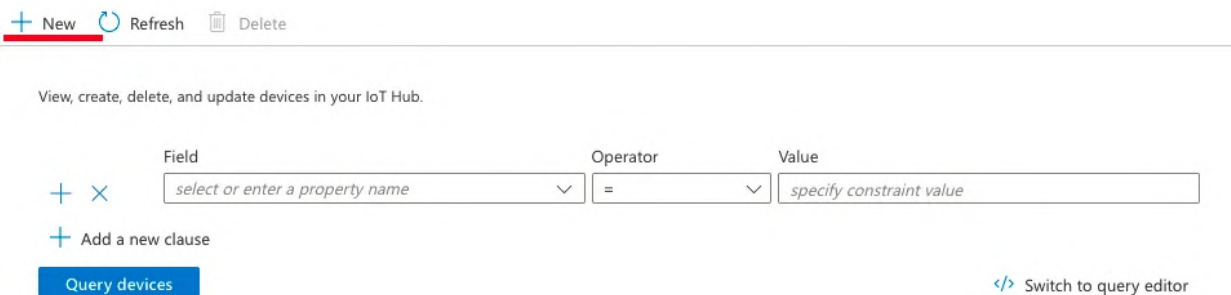
6. Click the "Create" button to create the IoT hub.



7. Once the IoT hub has been created, click on the "IoT Hub" item in the resource list to open the IoT hub blade.
8. In the IoT hub blade, click on the "IoT Devices" item in the left menu.



9. Click the "New" button to create a new IoT device.
10. Enter a name for the device and click the "Save" button.
11. Click on the newly created device in the device list to open the device details blade.
12. In the device details blade, click on the "Copy" button next to the "Connection string-primary key" field to copy the connection string to your clipboard.



13. On your device, use the connection string to connect to the IoT hub using the appropriate library or API for your device's platform.

For example, here is some Python code that demonstrates how to connect a device to an Azure IoT hub using the [azure-iot-device](#) library:

```

from azure.iot.device import IoTHubDeviceClient
# Replace the connection string with the one for your device
connection_string = "HostName=myiothub.azure-devices.net;DeviceId=mydevice;SharedAccessKey=1234567890abcdefghijklmnopqrstuvwxyz"
# Create a client for the device
client = IoTHubDeviceClient.create_from_connection_string(connection_string)
# Connect the device to the IoT hub
client.connect()
# Send a message to the IoT hub
client.send_message("Hello world!")

```

```
# Disconnect the device from the IoT hub
client.disconnect()
```

This code imports the `IoTHubDeviceClient` class from the `azure-iot-device` library and uses it to create a client for the device. The client is then connected to the IoT hub using the connection string. Once connected, the device can send and receive messages to and from the IoT hub.

Documentation

✓✓ The documentation for the SDK is available at: <https://azure-iot-sdk-python.readthedocs.io/en/latest/>

Samples:

✓✓ The samples for the SDK are available at:

<https://github.com/Azure/azure-iot-sdk-python/tree/master/azure-iot-device/samples>

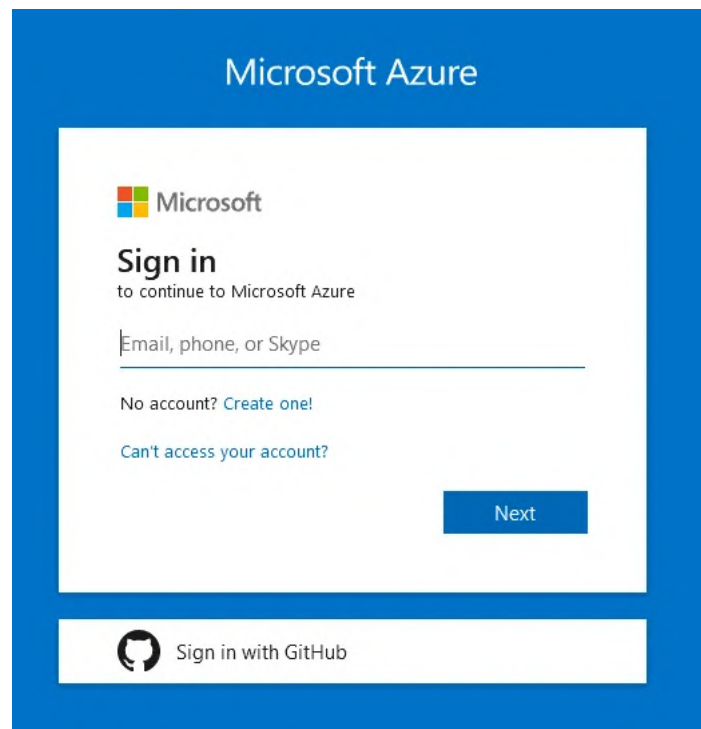
Visualising and interpreting data with Azure Time Series Insights and Power BI

A cloud-based analytics service called Azure Time Series Insights (TSI) is used to interpret, analyze, and display data from IoT devices and other sources. TSI offers a robust set of tools that can be used to store, process, and investigate time series data.

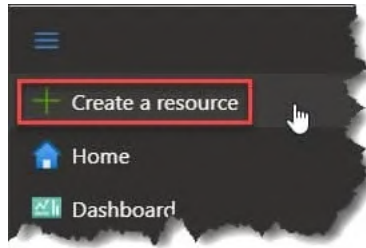
One way to interpret TSI data is to use Microsoft's business analytics service Power BI, which lets you create interactive visualizations and reports from a wide range of data sources.

To get started with TSI and Power BI, follow these steps:

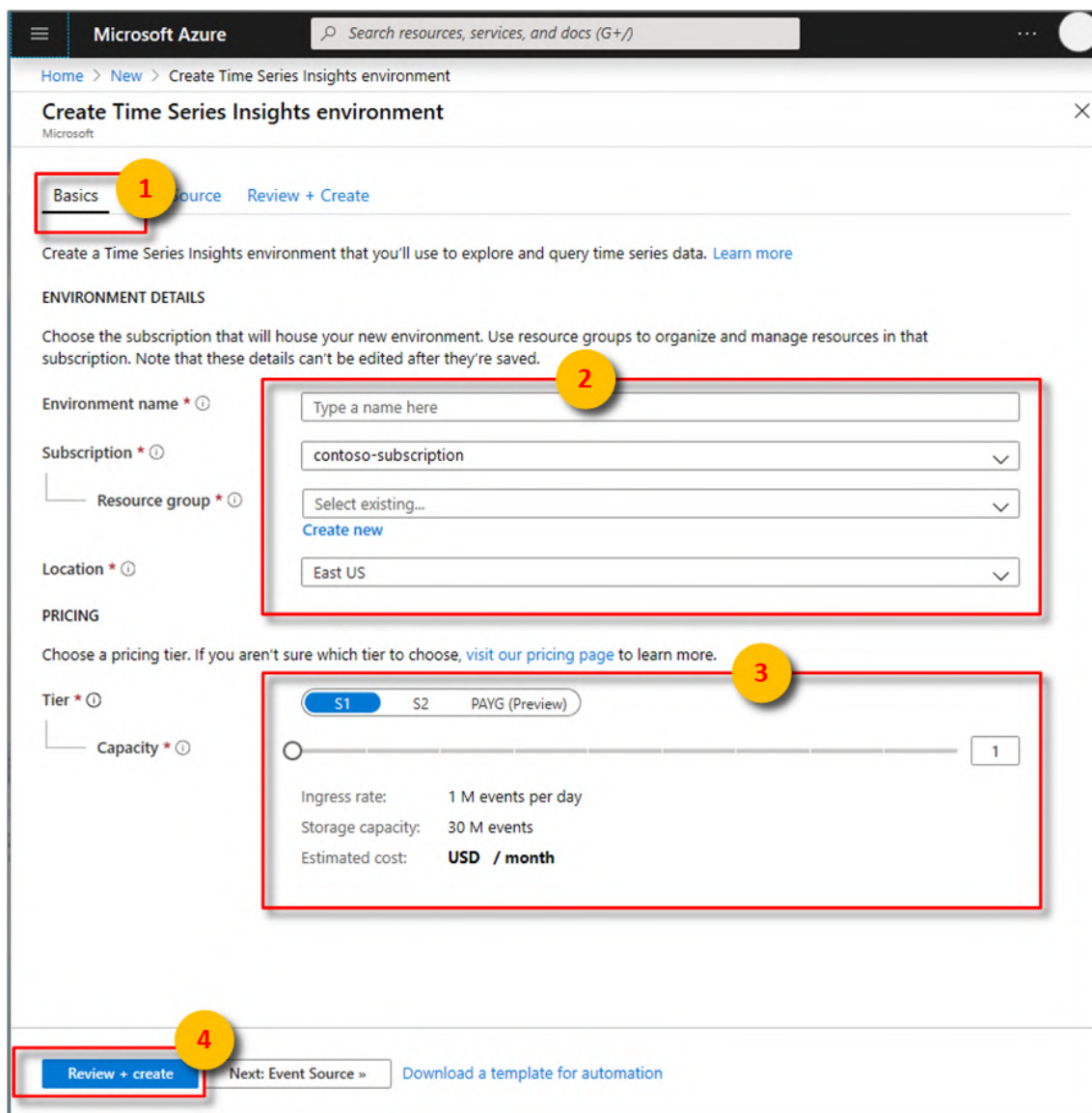
✓ Log in to the Azure portal using your Microsoft account at <https://portal.azure.com/>



✓ Click the "Create a resource" button in the portal's upper left corner.



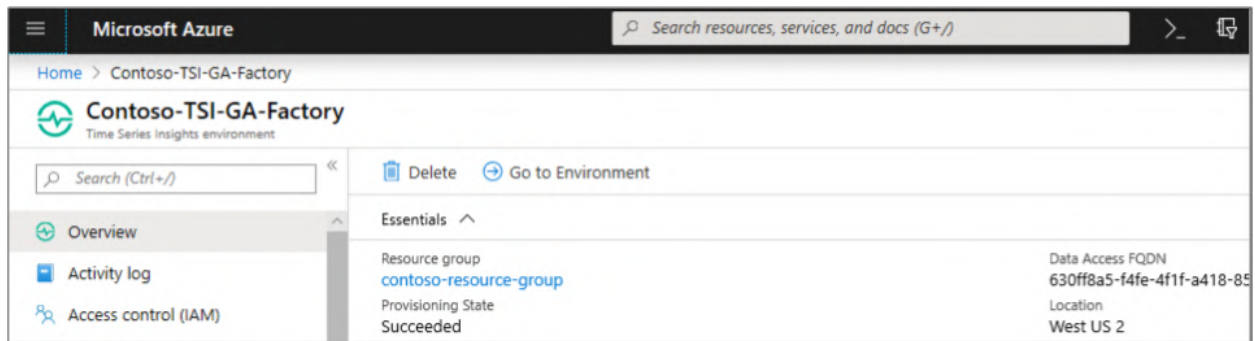
- ✓ In the "Search the Marketplace" section, search for "Time Series Insights."
- ✓ After selecting the "Time Series Insights" result, click the "Create" button.



✔ Select a resource group in the blade labelled "Create Time Series Insights environment" and give your TSI environment a name. Additionally, you can select the region where the environment should be created and the pricing level.

✔ Click the "Create" button to create the TSI environment.

✔ After creating the TSI environment, select the resource list item with the title "Time Series Insights" to open the TSI blade.



✔ Click the "Add data" button in the TSI blade to add data to the environment.

✔ Follow the instructions in the "Add data" wizard to connect to your data source and set the settings for the ingestion.

✔ Once the data has been ingested into TSI, you can use the TSI Explorer tool to see and analyze it.

✔ To view and analyze TSI data in Power BI, follow these steps:



Credit: techcommunity.microsoft.com

Log in to Power BI with your Microsoft account at <https://powerbi.com/> and select "Getdata" from the top menu.

- ✓ From the "Get Data" window, select the "Azure" category, and then select the "Azure Time Series Insights" option.
- ✓ Select "Connect" to connect to your TSI environment.
- ✓ Click the "Sign In" button after entering the URL of your TSI environment into the "Azure Time Series Insights" window.
- ✓ Follow the steps in the "Azure Time Series Insights" window to authenticate with your Azure account and select the data you want to import into Power BI.
- ✓ Select the "Load" option to load the data into Power BI.
- ✓ After the data has been loaded into the platform, you can create reports and dashboards that help you understand and interpret your data by utilizing the various visualization and analysis tools in Power BI.

Here is an example of how to use Python to retrieve data from TSI and display it in Power BI:

```
import requests
import json
import pandas as pd
from powerbi.api import WorkspacesApi
from powerbi.models import Import

# Set up the TSI API URL and query parameters
tsi_api_url = "https://mytsienvironment.env.timeseries.azure.com/timeseries/query"
params = {
    "query": "SELECT * FROM mydataset",
    "timespan": "P1D",
    "interval": "PT1H"
}

# Set up the TSI API access token
tsi_api_access_token = "mytsiaccesstoken"

# Set up the Power BI API client
powerbi_api_client = WorkspacesApi()

# Set up the Power BI API access token
powerbi_api_access_token = "mypowerbiaccesstoken"

# Send a request to the TSI API to retrieve the data
response = requests.get(tsi_api_url, params=params, headers={"Authorization": "Bearer " + tsi_api_access_token })

# Check the status code of the response
if response.status_code == 200:
    # Parse the response data as JSON
    data = json.loads(response.text)

    # Convert the data to a Pandas DataFrame
    df = pd.DataFrame(data["value"])

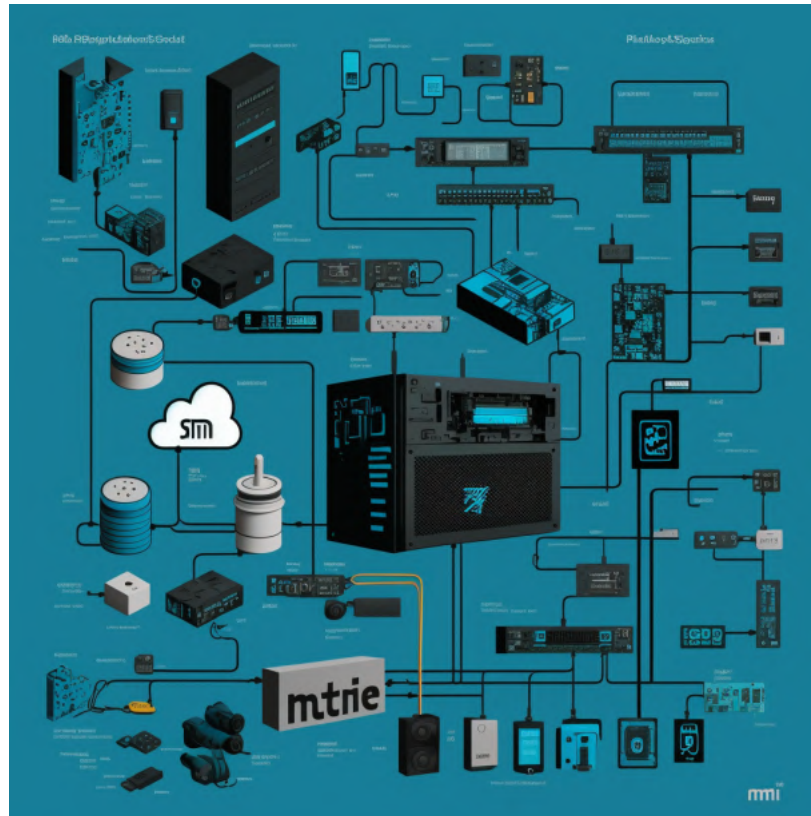
    # Use the Power BI API to import the data into a Power BI dataset
    result = powerbi_api_client.workspaces.import_dataset(workspace_id="myworkspaceid", import_request=Import(df=df), authorization=powerbi_api_access_token)

    # Print the result
    print(result)
else:
    # Print an error message
    print("Error retrieving data from TSI API: " + response.text)
```

The above code retrieves data from Azure Time Series Insights (TSI) using the TSI API, converts the data to a Pandas DataFrame, and imports the DataFrame into a Power BI dataset using the Power BI API.

Developing custom IoT applications with Python and Azure Functions

-

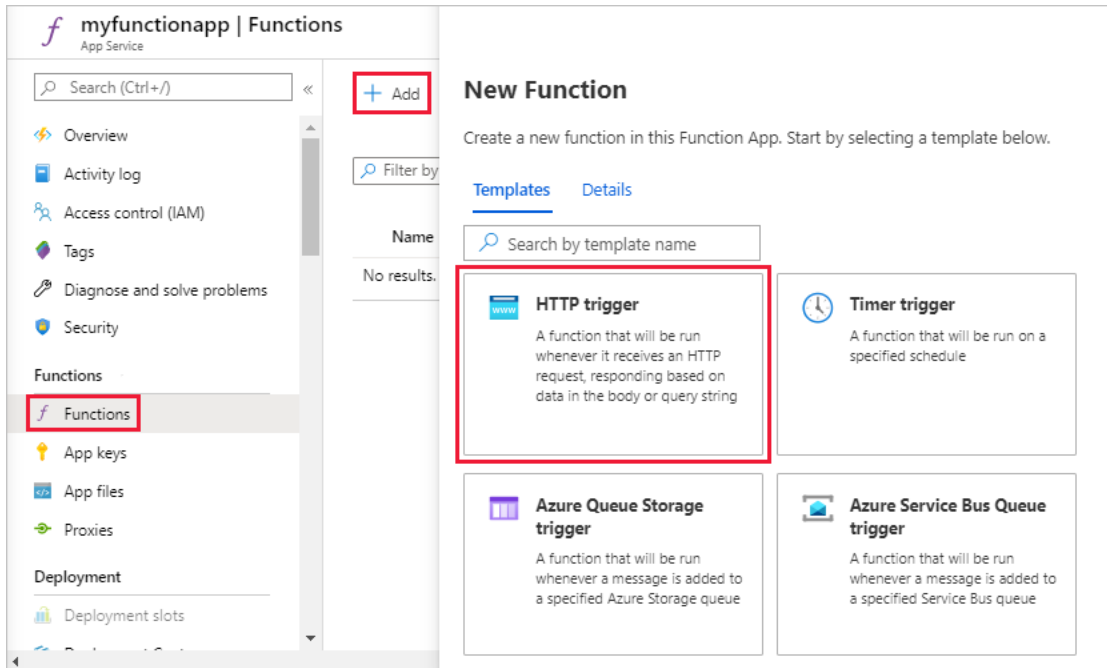


Internet of Things (IoT) technology is increasingly used in various industries, including agriculture, to improve efficiency, reduce costs, and increase productivity. One way to develop custom IoT applications is by using Python and Azure Functions, a serverless platform from Microsoft that allows you to run code on demand in response to events or triggers.

You will need an Azure account and a basic understanding of Python programming to develop custom IoT applications with Python and Azure Functions.

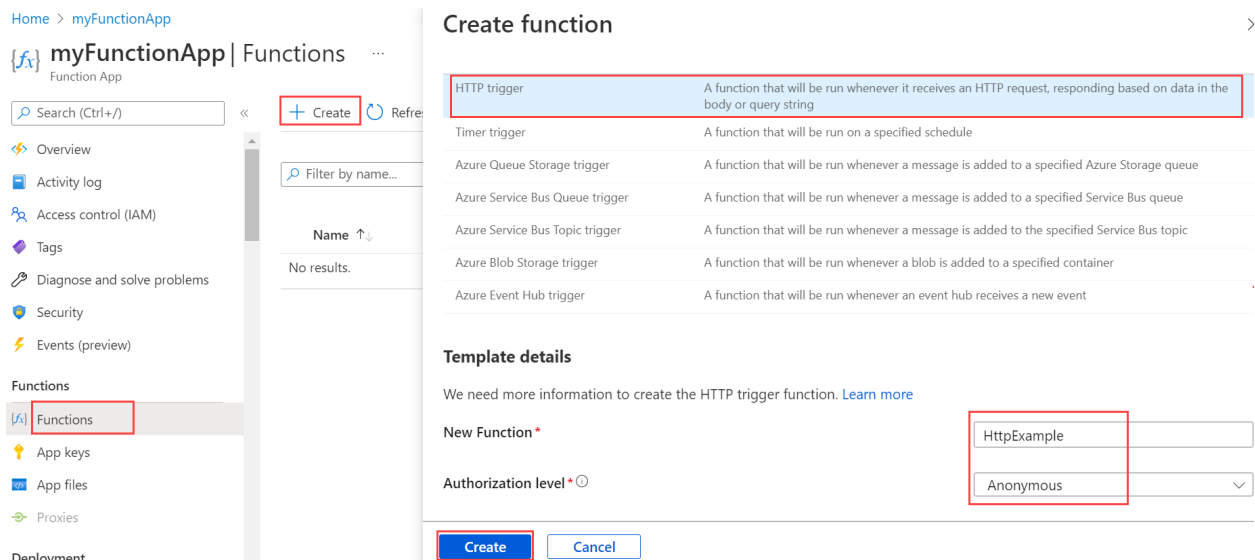
First, sign in to the Azure portal (<https://portal.azure.com/>) with your Microsoft account and create a new Function App. This container for your functions provides the infrastructure to host and run your code. In the "Create a function app" blade, give your role app a name and select a resource group. You can also choose the hosting plan and the runtime stack. Once the function app has been created, you can create a new function by clicking the "+" button next to the "Functions" header in the function app blade.

In the "Create a function" blade, select the "Python" runtime and choose a trigger for the function. Triggers are events or conditions that cause your part to be executed. Various triggers are available, such as HTTP requests, messages from IoT devices, or timer events. Once you have chosen a trigger, click the "Create this function" button to create the function.



credit: <https://learn.microsoft.com/>

You can then write Python code in the function editor that runs in response to the chosen trigger. For example, if you have selected an HTTP trigger, you can write code that processes incoming HTTP requests and returns a response. If you have selected a trigger that listens for messages from IoT devices, you can write code that processes the messages and performs any desired actions.



credit: <https://learn.microsoft.com/>

Here is an example of Python code that runs in an Azure Function and listens for messages from an IoT device:

```
import logging
import os
import json
```



```

from azure.iot.hub import IoTHubModuleClient

# Set up the IoT Hub connection string
connection_string = os.getenv("IOTHUB_CONNECTION_STRING")

# Create a client for the IoT Hub
client = IoTHubModuleClient.create_from_connection_string(connection_string)

def main(event: dict):
    # Extract the message payload from the event
    payload = json.loads(event["payload"])

    # Process the message payload
    logging.info("Received message: " + str(payload))

    # Send a response to the device
    client.send_message_to_output(payload, "output1")

# Return the response
return payload

```

This code imports the required libraries and initiates a connection to the IoT hub using the Microsoft Azure Hub Python SDK's `IoTHubModuleClient` module. A top-notch feature is created, the occasion's message payload is extracted, the load is processed, and a lesser reply is sent out to the device.

After that, you may deploy your feature on Azure, and it'll be activated each time the necessary event takes place. When you select an HTTP cause, for instance, your quality can be employed each time it receives an HTTP request. Your component might well be called each time it receives a letter from a gadget if you've chosen a reason that checks letters from IoT devices.

IV. Case Studies

Examples of successful IoT projects in the agricultural industry using Azure and Python

Here are a few examples of successful IoT projects in the agricultural industry that have used Azure and Python:

Here are some examples of successful Internet of Things (IoT) projects in the agricultural industry that use Azure and Python:

The following are some examples of successful Internet of Things (IoT) projects in the agricultural sector using Azure and Python:

Smart System for Watering:

An intelligent irrigation system that automatically waters crops based on soil moisture content is integrated into this project with Python and Azure IoT Hub. In addition, the system can monitor water usage and alert farmers to problems with the irrigation system.



✓ **Livestock Monitoring System:**

The health and welfare of farm animals are monitored by this project, which uses Python and Azure Stream Analytics. The system can alert farmers via SMS or email when a cow is in danger. Farmers can use this to protect their herds' overall health and welfare and avoid costly losses.



✓ **Precision Farming:** In precision farming, this project uses Python and Azure Machine Learning to increase crop yields and reduce waste. The system can analyze data from drones and sensors to improve fertilization and irrigation methods. Farmers can use data-driven insights to make better decisions about how to care for their crops and increase yields. Hope you find these examples helpful! Please let me know if you have any other questions.



✓ **Greenhouse Monitoring System** - This project uses Azure IoT Hub and Python to build a system that monitors temperature, humidity, and soil moisture in a greenhouse. The system can alert farmers **when** there are **environmental** issues **in** the greenhouse, such **as: B.** high temperatures or low soil **moisture**. By using this system, **growers** can ensure their **crops receive** the optimal growing conditions they need.



V. CONCLUSION

💡 **Summary of the benefits and potential of using Microsoft Azure and Python for IoT in agriculture**

Using Microsoft Azure and Python for Internet of Things (IoT) projects in agribusiness can provide some benefits, including:



● **Automation:** Azure and Python can be used to create automated systems that can monitor and control various aspects of farming operations, such as irrigation, fertilization, and livestock management.

● **Data-Driven Insights:** Azure Machine Learning capabilities and Python data analysis libraries can be used to collect data from sensors and other IoT devices to analyse to gain insight into crop performance, soil conditions and more. These insights can help growers make more informed decisions about caring for their crops and improving yields.

● **Scalability:** Azure's cloud-based infrastructure makes it easy to expand or scale down, allowing farmers to quickly deploy new IoT solutions or expand existing ones as their operations grow.

● **Security:** Azure offers a range of security features, such as encryption and access controls, to protect customer data and privacy.

Using Azure and Python for IoT projects in agriculture can improve efficiency, reduce waste, increase profits for farmers, and increase agricultural businesses.

🌐 **Future outlook and potential developments in the field.**



Increasing Adoption of Precision Farming: As more farmers adopt IoT technologies, we will likely see an increase in precision farming techniques that use data and analytics to optimise farming practices. This could include using sensors and drones to collect data on soil conditions, crop growth, etc.

Further integration with other technologies: We may see further integration between IoT and systems in agriculture and other technologies such as artificial intelligence (AI) and blockchain. For example, AI algorithms could be used to analyse data from IoT systems and provide farmers with recommendations to optimise their operations. Instead, blockchain could be used to track and verify the provenance and quality of agricultural products.

Growth in usage of edge computing: Growing demand for real-time data and analytics in agriculture could lead to increased use of edge computing where Data processed at the computer is the starting point (i.e. the "edge" of the network) and not in the cloud. This could enable faster data analysis and more responsive control of IoT systems.

Increased focus on sustainability: As the effects of climate change become more apparent, there will likely be a stronger Focus on sustainability in agribusiness.

IoT technologies could play a role in helping farmers optimize their operations to reduce waste and energy consumption while improving animal health and welfare.

BIBLIOGRAPHY

1. *Azure IoT Samples (Python):* <https://github.com/Azure-Samples/azure-iot-samples-python>
2. *Azure Functions Documentation:* <https://docs.microsoft.com/en-us/azure/azure-functions/>
3. *Azure Functions Python QuickStart:* <https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-function-python>

4. *Cloud IoT-based novel livestock monitoring and identification system using UID:* <https://www.researchgate.net/publication/321539682> *Cloud IOT based novel livestock monitoring and identification system using*
5. *Connecting Raspberry Pi to Azure IoT Hub with Python:* <https://www.codeproject.com/Articles/1249621/Lets-IoT-Hub-Tutorial>
6. *Livestock Monitoring (cattle) and Farming using IoT:* <https://iotdunia.com/iot-based-livestock-monitoring-and-farming/>
7. *How To Build a Machine Learning Classifier in Python with Scikit-learn:* <https://www.digitalocean.com/community/tutorials/how-to-build-a-machine-learning-classifier-in-python-with-scikit-learn>
8. *Smart Agriculture using IoT and Benefits of Smart agriculture:* <https://iotdunia.com/smart-agriculture/>
9. *Get Started with MQTT:* <https://www.mathworks.com/help/icommm/ug/get-started-with-mqtt.html>
10. *How Internet of Things (IoT) is transforming the agriculture sector?:* <https://www.businessofapps.com/insights/internet-of-things-iot-agriculture-sector/>
11. *Arduino Irrigation and Plant Watering using Soil Moisture Sensor:* <https://www.circuitstoday.com/arduino-irrigation-plant-watering-using-soil-moisture-sensor>
12. *Build your first static web app:* <https://learn.microsoft.com/en-us/azure/static-web-apps/get-started-portal?tabs=vanilla-javascript&pivots=github>
13. *IoT In Action - The Next Agricultural Revolution:* <https://learn.microsoft.com/en-us/shows/Internet-of-Things-Show/IoT-In-Action-The-Next-Agricultural-Revolution>
14. *Azure and Power BI:* <https://learn.microsoft.com/en-us/power-bi/connect-data/service-azure-and-power-bi>
15. *Five Ways the IoT is Transforming Agriculture:* <https://innovationatwork.ieee.org/five-ways-iot-transforming-agriculture/>

Article By: SAPTARSHI HALDER

