

IoT Workflow with Azure IoT Hub – Cloud Platform Demonstration

Objectives :

- Showcase a complete IoT data flow
- Demonstrate real-time data visualization
- Implement downlink control

Part 1 : Azure IoT Hub Setup

Step 1.1 :Creation of an IOT Hub

The screenshot shows two side-by-side browser windows from the Azure Marketplace.

Left Window (Marketplace): This window shows search results for "IoT Hub". It includes a sidebar with categories like Get Started, Service Providers, Management, and My Marketplace. The main area lists three items:

- ux4iot for IoT Hub** by Device Insight GmbH, described as an Azure Application. It connects web applications to IoT devices through IoT Hub.
- IoT Hub** by Microsoft, described as an Azure Service. It connects, monitors, and manages IoT devices.
- Azure IoT Hub Device Provisioning Service** by Microsoft, described as an Azure Service. It provides seamless, zero-touch registration of IoT devices.

Right Window (IoT hub details): This window shows the details for creating an IoT hub. It has tabs for Basics, Networking, Management, Add-ons, Tags, and Review + create (which is selected). The Basics section includes fields for Subscription (Azure for Students), Resource group (IOT), IoT hub name (EmnaAzureHub), Region (East US), Disaster recovery enabled (Yes), Tier (Free), and Daily message limit (8,000 (\$0/month)). Buttons at the bottom include Create, < Previous: Tags, Next >, and Automation options.

Step 1.2 :Creation of an IOT device

The screenshot shows the Azure IoT Hub Device Management interface. On the left, a sidebar menu includes 'Device management' (selected), 'Devices' (highlighted in blue), 'IoT Edge', 'Configurations + Deployments', 'Updates', 'Queries', 'Hub settings', and 'Security settings'. The main area is titled 'EmnaAzureHub | Devices' and shows a table with one row for 'virtualCapter'. The table columns are 'Device ID', 'Type', 'Status', 'Last st...', 'Authe...', 'C2D ...', and 'Tags'. The device details are: Device ID: virtualCapter, Type: IoT Dev..., Status: Enabled, Last st...: --, Authe...: Shared..., C2D ...: 0, Tags: 0.

Part 2: Sensor Simulation (Python)

Step2.1 : Create a script Python to send data to the device

```
!pip install azure-iot-device

# Import libraries
from azure.iot.device import IoTHubDeviceClient, Message
import random
import time

# === Azure IoT Hub Device Connection String ===
CONNECTION_STRING = "HostName=EmnaAzureHub.azure-devices.net;DeviceId=virtualCapter;SharedAccessKey=6IaXnVPqEi4VHIqVFczDb65UTv63isVw2ip1DD0Jn/o="

# Create IoT Hub client
client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)

# Connect to IoT Hub
print("Connecting to Azure IoT Hub...")
client.connect()
print("Connected successfully!")

# Send simulated telemetry
print("Sending telemetry data...")
for i in range(10): # Send 10 messages
    temperature = round(random.uniform(20, 30), 2)
    humidity = round(random.uniform(40, 60), 2)
    payload = f'{{"temperature": {temperature}, "humidity": {humidity}}}'
    message = Message(payload)
    client.send_message(message)
    print(f"Message {i+1} sent: {payload}")
    time.sleep(5) # Wait 5 seconds between messages
```

```
client.publish(topic, payload)
print("Données envoyées :", payload)
time.sleep(5)

...
SAS Token généré : SharedAccessSignature sr=EmmaAzureHub.azure-devices.net/devices/virtualCapter&sig=JJOvM8338LgLBcdhKevCqyNP
/tmipython-input-2716408879.py:28: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client(client_id=device_id, protocol=mqtt.MQTTv311)
Données envoyées : {"temperature": 23.42, "humidity": 56.69}
Données envoyées : {"temperature": 22.53, "humidity": 53.53}
Données envoyées : {"temperature": 24.06, "humidity": 54.01}
Données envoyées : {"temperature": 21.7, "humidity": 56.49}
Données envoyées : {"temperature": 21.89, "humidity": 59.75}
Données envoyées : {"temperature": 23.79, "humidity": 58.38}
Données envoyées : {"temperature": 22.18, "humidity": 46.11}
Données envoyées : {"temperature": 24.15, "humidity": 46.27}
Données envoyées : {"temperature": 21.68, "humidity": 53.56}
```

Part 3: Stream Analytics Job

Step3.1 : Create a steam job

The screenshot shows the Azure Stream Analytics Job Overview page. At the top, there's a navigation bar with Microsoft, a search bar, and Copilot. Below it, the title is 'StreamAnalyticsJob | Overview'. A deployment icon and the text 'Deployment' are visible. A green checkmark icon indicates 'Your deployment is complete'. Deployment details are listed: Deployment name: StreamAnalyticsJob, Subscription: Azure for Students, Resource group: IOT, Start time: 11/22/2025, 10:18:54 PM, and Correlation ID: 97927df5-b4be-41df-9db0-1730ad4e0bc8. There are two expandable sections: 'Deployment details' and 'Next steps'. A blue button labeled 'Go to resource' is at the bottom. Feedback links are at the very bottom.

All services >

StreamAnalyticsJob | Overview

Deployment

>Your deployment is complete

Deployment name : StreamAnalyticsJob
Subscription : Azure for Students
Resource group : IOT
Start time : 11/22/2025, 10:18:54 PM
Correlation ID : 97927df5-b4be-41df-9db0-1730ad4e0bc8

Deployment details

Next steps

Go to resource

Give feedback

Tell us about your experience with deployment

Step 3.2 :Creation of an input and output for the job

IoT Hub

New input

Input alias *
EmnaAzureHub ✓

Provide IoT Hub settings manually
 Select IoT Hub from your subscriptions

Subscription
Azure for Students

IoT Hub * (1)
EmnaAzureHub

Consumer group * (1)
\$Default

Shared access policy name * (1)
iothubowner

Shared access policy key (1)
.....

Endpoint (1)
Messaging

Save

Power BI

New output

Stream Analytics Power BI output will be retiring on 31st Oct 2027. To learn more about the Power BI output retirement click here.

Output alias *
PowerBIOutput ✓

Provide Power BI settings manually
 Select Power BI from your subscriptions

Group workspace *
My workspace

Authentication mode
User token

Dataset name * (1)
IoTDashboard ✓

Table name *
TelemetryData ✓

Authorize connection
You'll need to authorize with Power BI to configure your output settings.

Save

Step 3.3 :Create a query and start the job

The screenshot shows the Azure Dashboard | Query interface for a Stream Analytics job. The left sidebar lists components: Inputs (1) with IoTHubInput selected, Outputs (1) with PowerBIOOutput, and Functions (0). The main area displays a test query script:

```
1 SELECT
2     temperature,
3     humidity,
4     System.Timestamp AS time
5 INTO
6     PowerBIOOutput
7 FROM
8     IoTHubInput
```

Below the query, there are buttons for Test query, Save query, and Discard changes. A message box indicates: "Query can't be edited while a job is running. You can stop the job to edit the query."

Step 3.4 :Getting data from the script

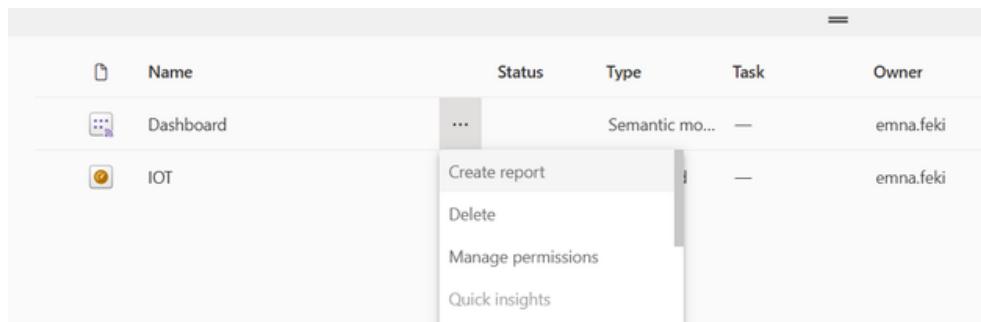
The screenshot shows the Input preview section of the Stream Analytics job configuration. It displays sample events from the IoTHubInput source. The columns are temperature, humidity, EventProcessedUtcTime, PartitionId, EventEnqueuedUtcTime, and IoTHub.

temperature	humidity	EventProcessedUtcTime	PartitionId	EventEnqueuedUtcTime	IoTHub
28.57	47.12	"2025-11-23T14:52:25....	1	"2025-11-23T14:52:00....	{"MessageId":
26.28	51.86	"2025-11-23T14:52:25....	1	"2025-11-23T14:52:06....	{"MessageId":
25.52	48.9	"2025-11-23T14:52:25....	1	"2025-11-23T14:52:11....	{"MessageId":
25.64	51.4	"2025-11-23T14:52:25....	1	"2025-11-23T14:52:16....	{"MessageId":

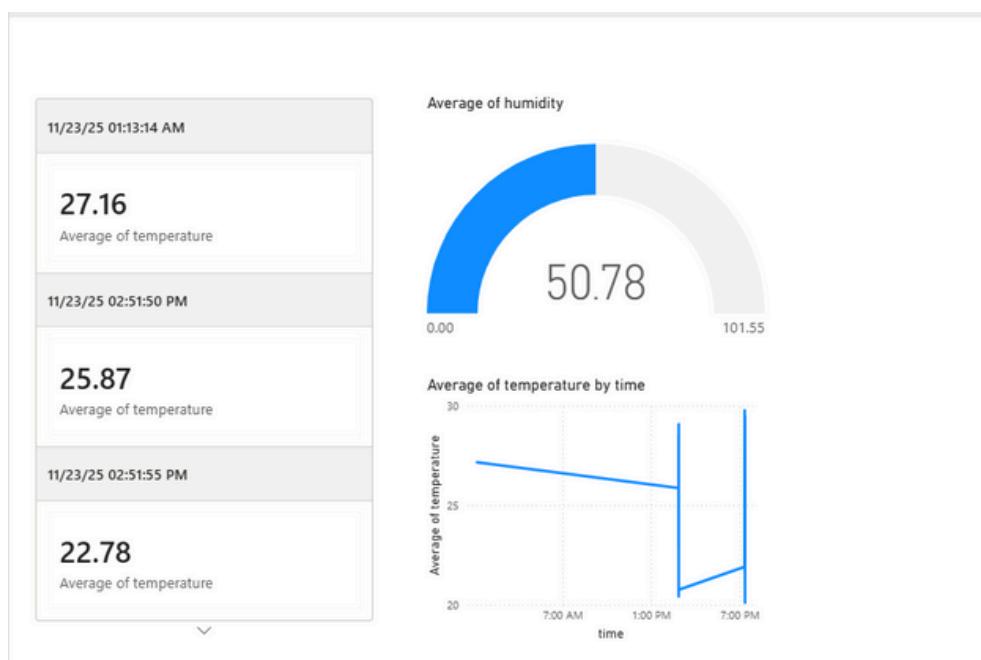
At the bottom, a note says: "While sampling data, no data was received from '1' partitions." and "Ln 10, Col 1".

Part 4: Power BI Dashboard

Step 4.1 : Create a report for the DB already created in the output

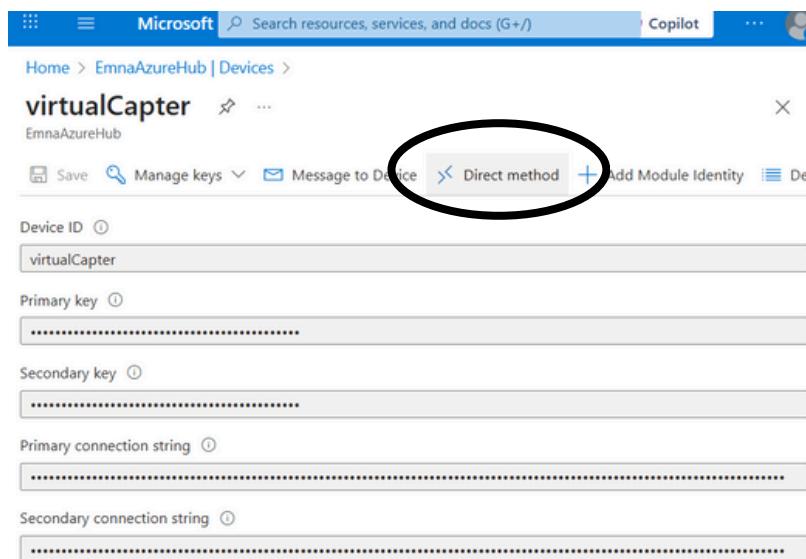


Step 4.2 :Display Data



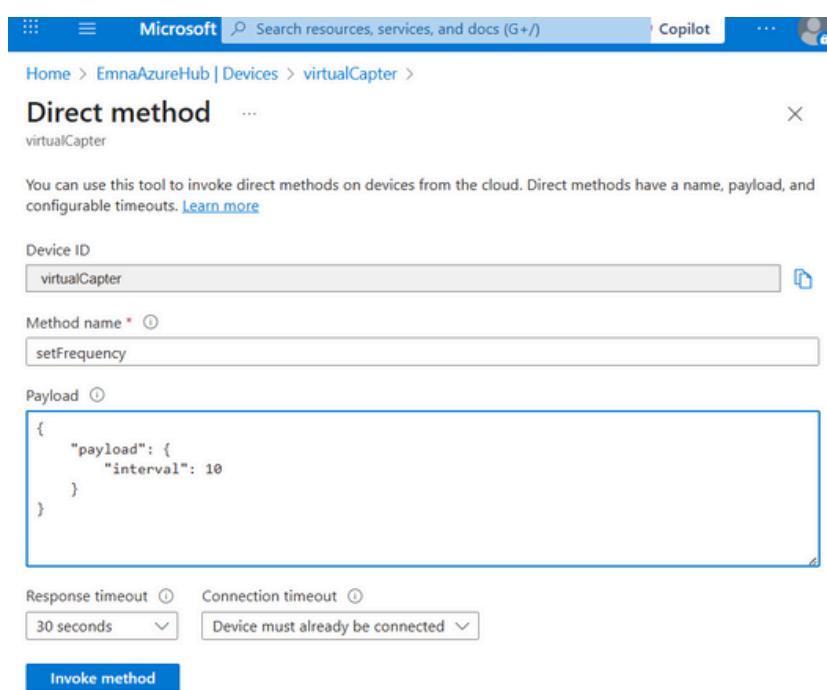
Part 5: Downlink control

Step 5.1 : Create a direct method to the virtual device



The screenshot shows the Azure portal interface for managing a device named 'virtualCapter'. The top navigation bar includes 'Microsoft', 'Search resources, services, and docs (G+)', 'Copilot', and a user profile icon. Below the navigation, the path 'Home > EmnaAzureHub | Devices > virtualCapter' is displayed. The main content area shows the device configuration with fields for 'Device ID' (set to 'virtualCapter'), 'Primary key', 'Secondary key', 'Primary connection string', and 'Secondary connection string', each represented by a redacted text box. At the top of the configuration pane, there are tabs: 'Save', 'Manage keys', 'Message to Device', **'Direct method'** (which is circled in black), and 'Add Module Identity'. A large 'X' button is located in the top right corner of the configuration pane.

Step 5.2 :Code of the method

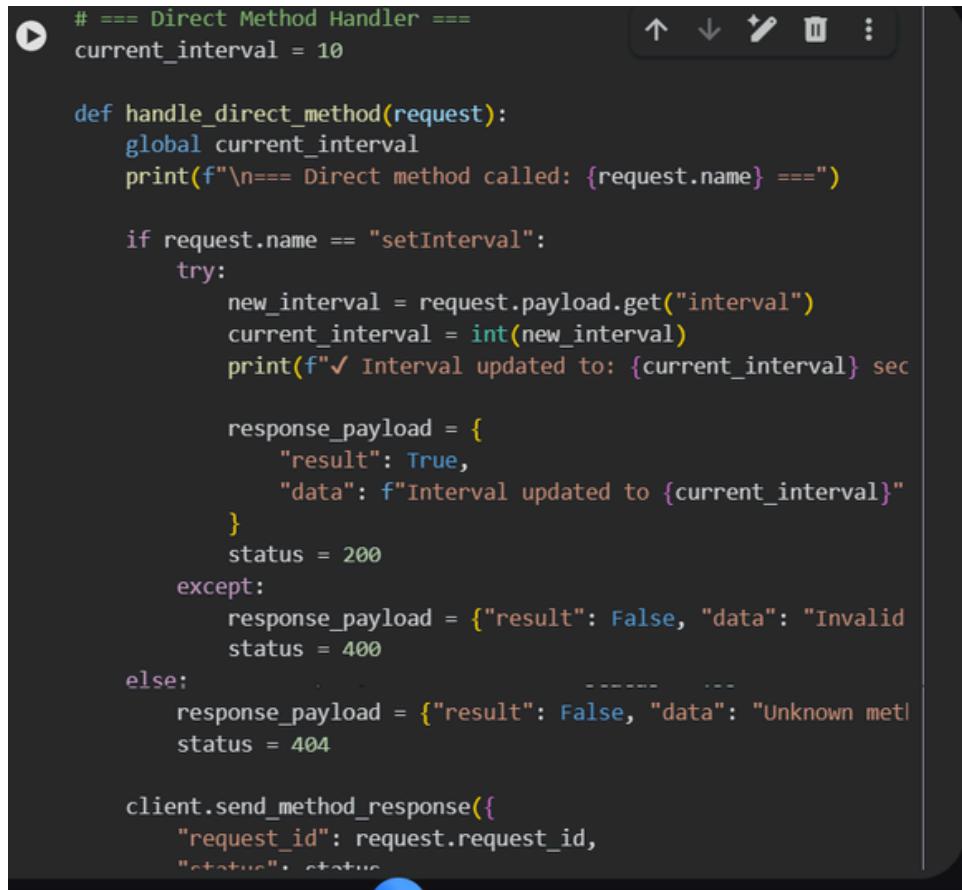


The screenshot shows the 'Direct method' configuration page for the 'virtualCapter' device. The top navigation bar and device path are identical to the previous screenshot. The main content area is titled 'Direct method' and shows the configuration for a method named 'setFrequency'. It includes fields for 'Device ID' (set to 'virtualCapter'), 'Method name' (set to 'setFrequency'), and a 'Payload' section. The payload is defined as a JSON object:

```
{  
    "payload": {  
        "interval": 10  
    }  
}
```

Below the payload, there are dropdowns for 'Response timeout' (set to '30 seconds') and 'Connection timeout' (set to 'Device must already be connected'). A large blue rectangular box highlights the payload JSON code. At the bottom of the page is a blue 'Invoke method' button.

Step 5.3: Edit the script by adding a method handler



```
# === Direct Method Handler ===
current_interval = 10

def handle_direct_method(request):
    global current_interval
    print(f"\n--- Direct method called: {request.name} ---")

    if request.name == "setInterval":
        try:
            new_interval = request.payload.get("interval")
            current_interval = int(new_interval)
            print(f"✓ Interval updated to: {current_interval} sec")

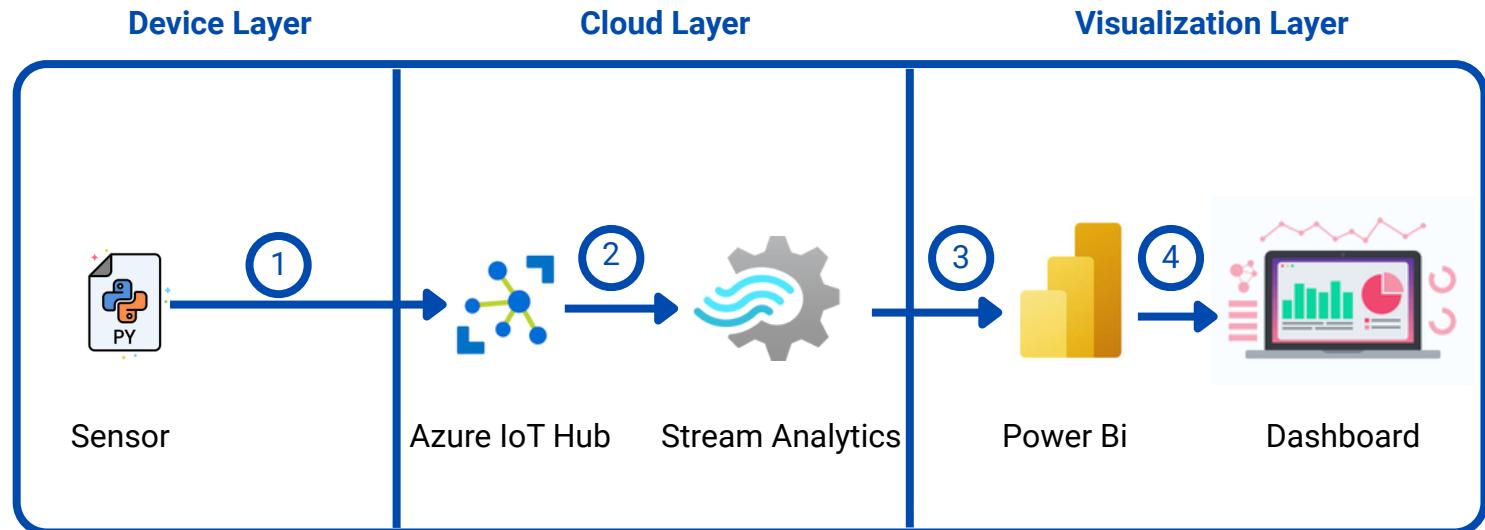
            response_payload = {
                "result": True,
                "data": f"Interval updated to {current_interval}"
            }
            status = 200
        except:
            response_payload = {"result": False, "data": "Invalid input"}
            status = 400
    else:
        response_payload = {"result": False, "data": "Unknown method"}
        status = 404

    client.send_method_response({
        "request_id": request.request_id,
        "status": status
    })
```

Step 5.4: The script get the method

```
Connected successfully!
Telemetry sent: {"temperature": 24.64, "humidity": 53.39}
Telemetry sent: {"temperature": 20.18, "humidity": 52.58}
WARNING:azure.iot.device.common.handle_exceptions:Exception caught in background thread. Unable to handle.
WARNING:azure.iot.device.common.handle_exceptions:[azure.iot.device.iothub.sync_handler_manager.HandlerManagerException: HANDLE
Received method: setInterval with payload: {'payload': {'interval': 10}}
Telemetry sent: {"temperature": 27.83, "humidity": 50.16}
Telemetry sent: {"temperature": 25.36, "humidity": 42.6}
Telemetry sent: {"temperature": 21.77, "humidity": 42.21}
Telemetry sent: {"temperature": 21.23, "humidity": 59.99}
```

Part 6: Architecture of the workflow



- ① The Python script simulates a virtual sensor that sends temperature and humidity telemetry to Azure IoT Hub using the MQTT protocol, authenticated via the device connection string.
- ② Azure IoT Hub receives the MQTT messages and forwards them as an input stream to the Azure Stream Analytics job for real-time processing.
- ③ The Stream Analytics job applies a query to transform and aggregate the IoT data, then outputs the processed dataset to Power BI for visualization.
- ④ Power BI consumes the dataset and renders a real-time dashboard with interactive charts, including a temperature line chart, a humidity gauge, and a summary card.