SE-IOT: Internet of Things

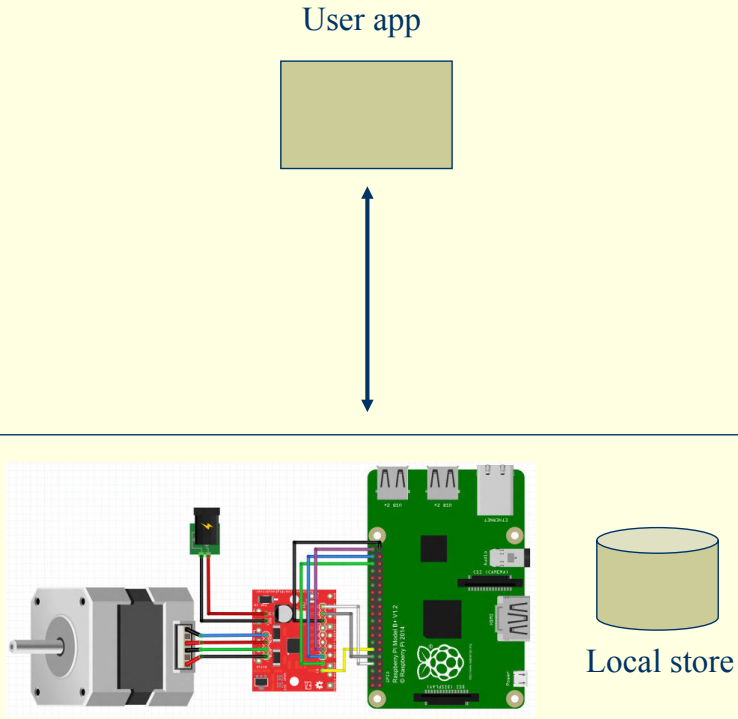
Working with Sensor Data

Derek Kiong
dkiong@nus.edu.sg

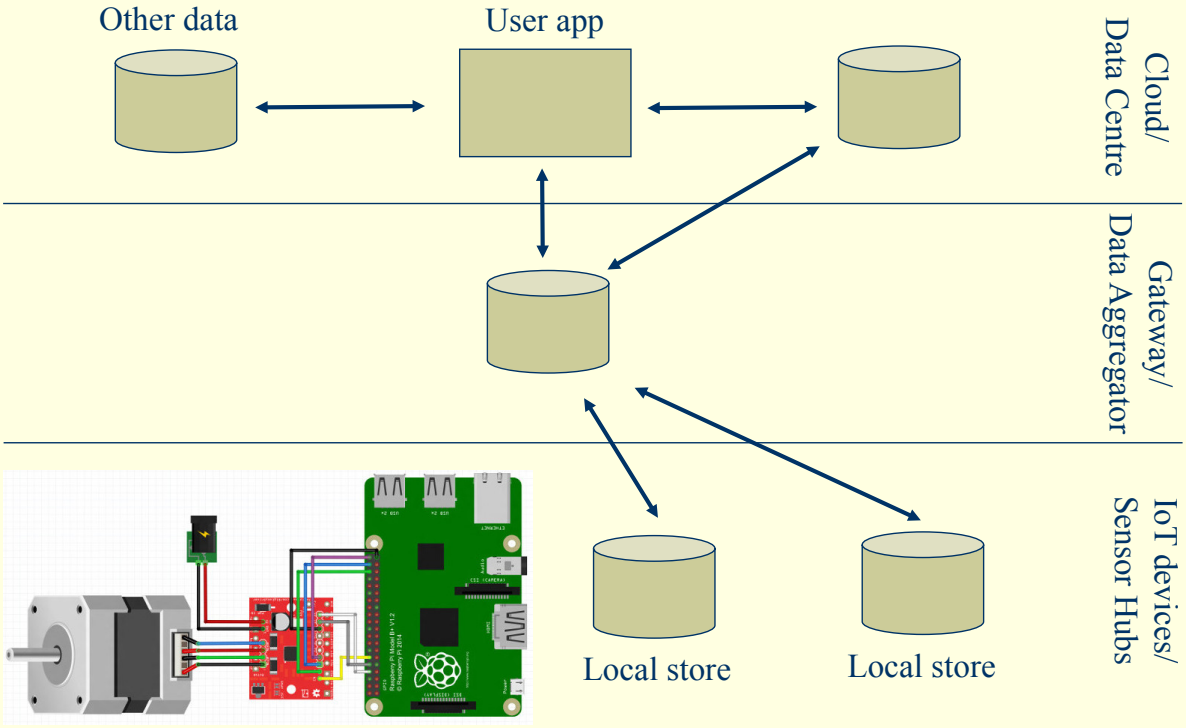
Outline

- Data flow paradigms
- Database operations on Raspberry PI
- Data output format
- Design considerations

Stand-alone system



Coordinated-sensors system



Handling Data

- Local vs remote
- Instantaneous data transfer vs batched
  - Immediate, hourly, daily, weekly
- Push vs pull
  - http, ftp, sftp, scp etc
- Formats
  - csv, Json

Database operations on Raspberry PI

- Install embedded database SQLite and its Python driver:

```
$ sudo apt-get install sqlite3
$ sudo apt-get install python-sqlite
```

- Connect to SQLite database.

```
import sqlite3
connection = sqlite3.connect('mydatabase.db')
```

- SQLite stores database as a file. It will create the file if the file does not exist yet.

Database operations on Raspberry PI

- Typical SQL operation:

```
import sqlite3
import sys

connect = None
try:
    connection = sqlite3.connect('mydatabase.db')
    cursor = connection.cursor()
    cursor.execute(SQL query string)
    ...
except sqlite.Error, e:
    print "Error %s:" % e.args[0]
    sys.exit(1)
finally:
    if connection:
        connection.close()
```


Data output format

- CSV
  - Simplified spreadsheets stored as plaintext files.
  - Each line in a CSV file represents a row in the spreadsheet, and commas separate the cells in the row.
  - `12,iPhone 6,899.0`
- JSON
  - De facto standard for data exchange
  - (Almost) human-readable string
  - Many websites offer JSON content as a way for programs to interact with the website
  - `{ "Id": 12, "Name": "iPhone 6", "Price": 899.0 }`

CSV output

- Use Python **`csv`** library (**`python-csvkit`**)
- Write to file:

```
import csv

w = open('f.csv', 'w')
writer = csv.writer(w)
writer.writerow([12, 'iPhone 6', 899.0])
writer.writerow([14, 'iPhone 5', 699.0])
w.close()
```


CSV input

- Read from file:

```
import csv

with open('f.csv', 'rb') as csvfile:
    r = csv.reader(csvfile, delimiter=',')
    for row in r:
        print ', '.join(row)
```

CSV output in HTTP

- Downloadable file over HTTP:

```
import csv
from django.http import HttpResponse

def some_view(request):
    # Create HttpResponse object with the appropriate CSV header.
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] =
       'attachment; filename="somefilename.csv"'

    writer = csv.writer(response)
    writer.writerow([12, 'iPhone 6', 899.0])
    writer.writerow([14, 'iPhone 5', 699.0])

    return response
```


JSON output

- Use Python **`json`** library
- **`json.dumps(data)`** converts Python **`dict`** to JSON string.

```
import json

data = {'inStock': True,
        'quantity': 10,
        'name': 'iPhone 6', 'accessories': None}

print (json.dumps(data))
```

Output:

```
{ "inStock": true, "quantity": 10, "name": "iPhone 6",
"accessories": null }
```


JSON output over HTTP

- Downloadable file over HTTP:

```
import json
from django.http import HttpResponse

def some_view(request):
    data = {"Id": 12, "Name": "iPhone 6",
            "Price": 899.0}

    return HttpResponse(json.dumps(data),
      content_type='application/json')
```


Reading JSON

- `json.loads(data)` converts JSON string to Python `dict` data structure.

```
import json

string = '{"inStock": true, "quantity": 10,
      "name": "iPhone 6", "accessories": null}'

print (json.loads(string))
```

Output:

```
{'inStock': True, 'quantity': 10,
 'name': 'iPhone 6', 'accessories': None}
```


Example:
Fetch JSON weather data from OpenWeatherMap.org

Before you start:

- Sign up **OpenWeatherMap.org** to get an API key on your account page.
- Try fetching weather data with the URL below, by replacing {APIKEY} with your API key:

```
http://api.openweathermap.org/data/2.5/weather
  ?q=Changi,SG&APPID={APIKEY}&units=metric
```


Example:
Fetch JSON weather data from OpenWeatherMap.org

```
import json, requests

url =
"http://api.openweathermap.org/data/2.5/weather?q=C
hangi,SG&APPID={APIKEY}&units=metric"

response = requests.get(url)
try:
    response.raise_for_status()
    w = json.loads(response.text)
    print ("Current weather in %s: %s" %
       (w["name"],w["weather"][0]["description"]))
    print ("Temperature: %s" % w["main"]["temp"])
except requests.exceptions.HTTPError as e:
    print "HTTP Error:", e.message
```


Design considerations

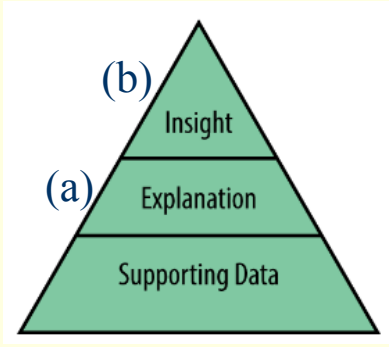
Network latency and reliability

- To handle data processing locally (in edge devices) or centrally (in a cloud server)?
- If the immediate response or near-failsafe reliability is required, the device needs to process data locally.
  - E.g. Cameras, image processing and collision detection in self-driving cars can't wait for the Internet to tell it what to do next.
- If the network is down, the device should be able to cache data, perhaps in SQLite database or a local file, and send them in batch later.
- If the readings go over a certain threshold, the device may send data more frequently.

Design considerations

What to do with the data?

- Make data meaningful to users (a):
  - Energy consumption in $ is easier to understand than in kWh; storage capacity in number of MP3 songs is easier to understand than in Gigabytes (GB).
- Make data actionable (b):
  - From data to insights that are actionable to achieve users' goals.
  - Make use of third party data to augment your data and to provide context.
  - E.g. A activity tracker might suggest the user who is falling behind his target step count to walk to canteen instead of driving to achieve user's fitness goal, because third party weather data predicts pleasant weather at noon.



Summary

- Data flow in IoT: Local processing in edge device vs. remote processing in cloud server
- Database operations on Raspberry PI: Use SQLite Python driver to store data locally
- Data output format in CSV and JSON format. When accessing third party data is necessary, JSON format is popular among many data sources.
- When designing IoT system architecture, data processing should take network latency and reliability into account.
- When presenting data to the users, it should be meaningful and actionable.