Golang and IoT

Golang and IoT?

HELLO!

I am Stas Arsoba

Software Developer @ Itransition

Everything is better with Bluetooth.

Sheldon Cooper

Why Golang and IoT?

Only 10% Golang developers work with IoT. (Go 2018 Survey)

What is IoT?

Internet connectivity in physical devices.


nest
nest
65
HEATING
68
nest
nest


Pick up Joseph
from soccer
15:00
12:45
68°
Dinner with Judy
65°


UBER
ADVANCED
TECHNOLOGIES
CENTER
UBERATC.COM/CAR

1.567T $
by 2025

73B devices
by 2025

IoT spheres:

- healthcare
- retail
- smart homes
- advertising
- transport
- urban infrastructure
- agriculture
- … and many others

IoT project

How does it look?

Devices

Protocols

Software

Devices

Basically, any type of device with connection to the Internet or to other device.

Protocols

Device 2 Device

- Bluetooth
- ZigBee
- Z-Wave
- Thread

Device 2 Internet

- WiFi
- Ethernet

Software

- Firmware
- Applications
- Backend (cloud)

Demo Project

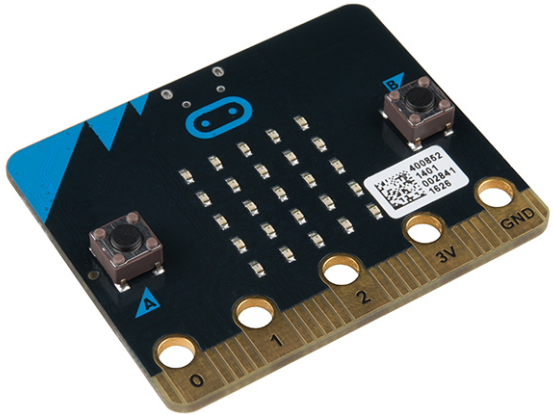
Raspberry Pi gateway and BLE sensor.

Xiaomi BLE Temperature Humidity Sensor

- BLE
- Temperature
- Humidity
- Display

BBC micro:bit

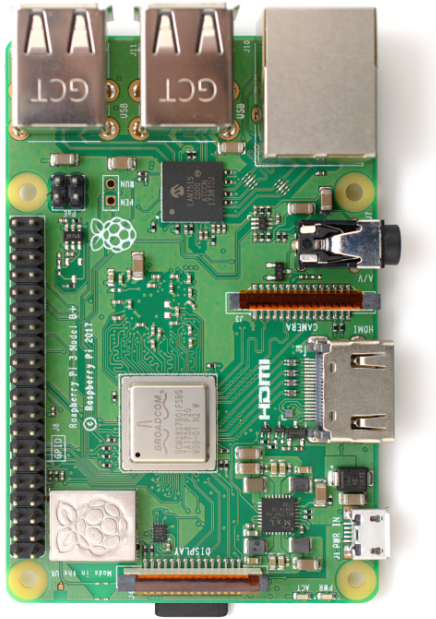
- Developed on nRF51822 SoC.
- Bluetooth
- 5x5 LED
- 2 buttons
- … and others



Raspberry Pi

Small single-board computer.

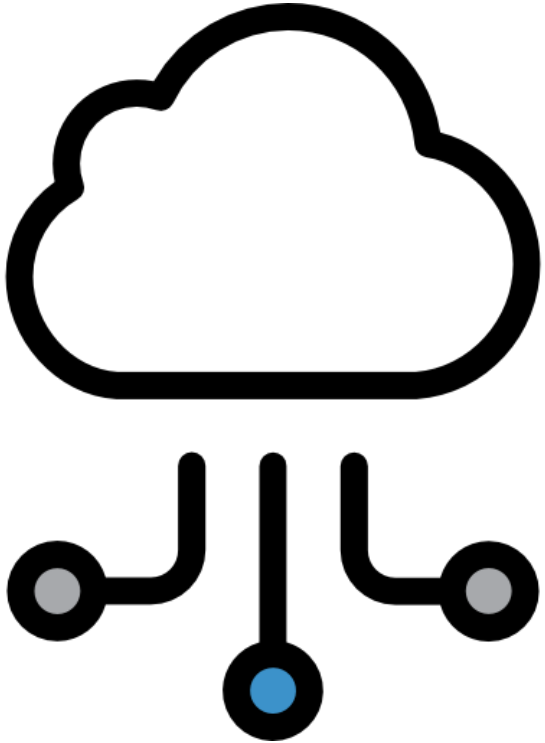
- ARM processor
- Ethernet/WiFi
- GPIO
- … and others



Backend

Any backend. Cloud service, for example (AWS, Azure …)

Just for demo laptop will be the backend.



Ubuntu Core

The secure, lightweight and robust operating system for the internet of things.

Core

Snappy

Software deployment and package management system.

The system is designed to work for internet of things.

snappy

TinyGo

Golang compiler for small devices.
Can compile and run programs on Arduino, BBC micro:bit and Nordic Developer Kits.

MQTT

Message Queuing Telemetry Transport is an ISO standard publish-subscribe-based messaging protocol.

MQTT

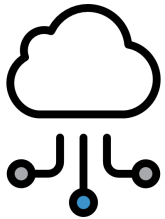
Works on top of the TCP/IP.

Roles:

- Client
- Broker

Client can publish messages and subscribe to topic.
Information is organized in a hierarchy of topics.

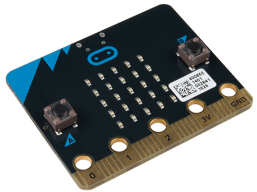
Ex. `/home/kitchen/temperature`

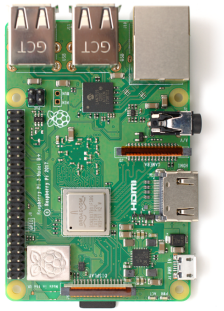


MQTT

BLE

UART





BLE

BLE

Bluetooth Low Energy

GAP

Generic Access Profile controls connections and advertising in Bluetooth.

Peripheral

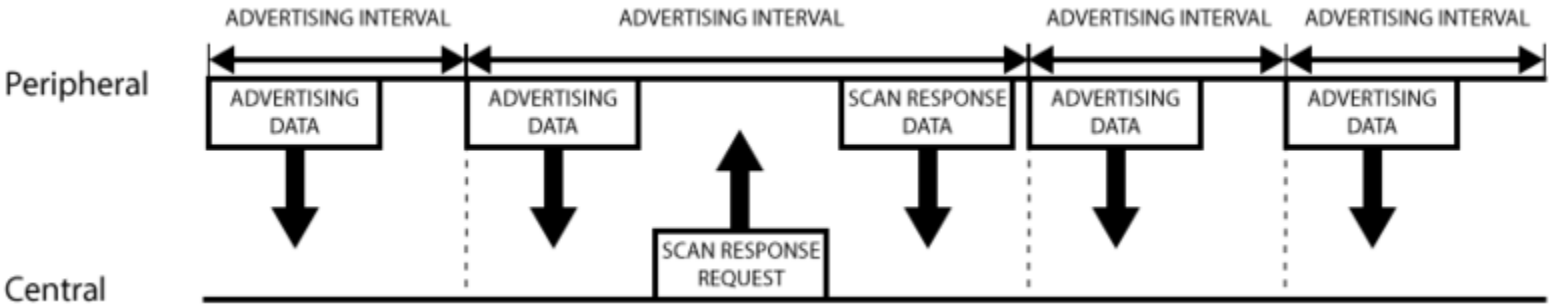
Small, low power devices that can connect to central device.

Central

Mobile phone, tablet or gateway that you connect to with far more processing power and memory.

Advertising

Process that uses a Peripheral device to advertise information about itself. These are small packets of 31 bytes.



GATT

Generic Attribute Profile defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called **Services** and **Characteristics**.

Service is a set of related characteristics.

Characteristic which encapsulates a single data point.

Battery Service

Battery Service is adopted by BLE.

UUID length: 16 bytes for adopted and 128 bytes for custom.

Battery Service UUID is 0x180F.

Has only one characteristic Level with UUID 0x2A19.

HCI

Host Controller Interface is a thin layer which transports commands and events between the host and controller elements of the Bluetooth protocol stack.

go-ble/ble

In fact, the only stable library for working with BLE.

Could be used for Peripheral and Central devices.

https://github.com/go-ble/ble

Connection to BLE device

```
filter := func(a ble.Advertisement) bool {
    return strings.ToUpper(a.Addr().String()) == mac
}

client, err = ble.Connect(context.Background(), filter)
```

Write characteristic

```go
func (t *Thigy) writeCharacteristic(p *ble.Profile, m BleMsg) (err error) {
    characteristicUUID := parseUUID(m.CharacteristicUUID)
    if u := p.Find(ble.NewCharacteristic(characteristicUUID)); u != nil {
        err := t.client.WriteCharacteristic(u.(*ble.Characteristic), m.B, true)
        return errors.Wrap(err, "can't write characteristic")
    }
    return err
}
```

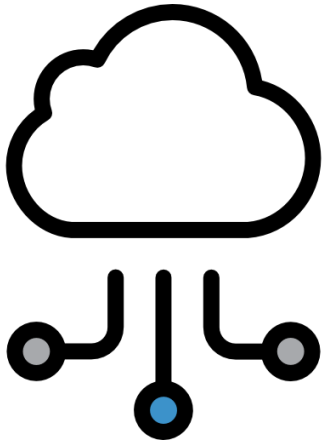
Read characteristic

```go
func (t *Thigy) readTemp(p *ble.Profile) error {
    characteristicUUID := parseUUID(TempUUID)
    h := func(req []byte) {
        t.ThigyState.Temp = [2]int{int(req[0]), int(req[1])}
    }
    if u := p.Find(ble.NewCharacteristic(characteristicUUID)); u != nil {
        err := t.client.Subscribe(u.(*ble.Characteristic), false, h)
        return errors.Wrap(err, "can't subscribe to characteristic")
    }
    return nil
}
```

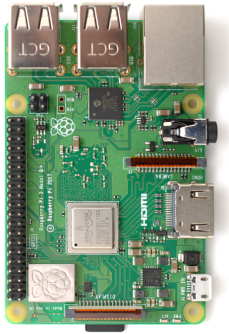
Snap manifest

```yaml
parts:
  ble:
    source: .
    plugin: go
    go-importpath: ble

apps:
  ble:
    command: ble
    daemon: simple
    restart-condition: always
    plugs: &plugs [network, bluetooth-control]
```



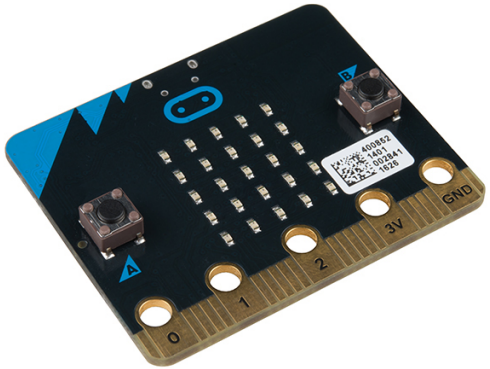
MQTT



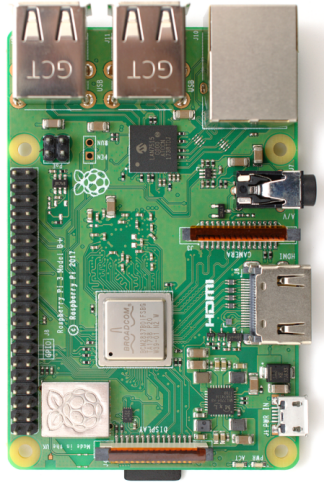
Publish

```go
func (m *MqttClient) Publish(topic string, payload interface{}) {
    message, err := json.Marshal(payload)
    if err != nil {
        fmt.Println(err)
        return
    }
    token := m.PahoMqttClient.Publish(topic, byte(0), false, message)
    token.Wait()
}
```

https://github.com/eclipse/paho.mqtt.golang



UART



GPIO
UART

```go
left := machine.Pin(machine.BUTTONA)
left.Configure(machine.PinConfig{Mode: machine.PinInput})

right := machine.Pin(machine.BUTTONB)
right.Configure(machine.PinConfig{Mode: machine.PinInput})

uart.Configure(machine.UARTConfig{TX: tx, RX: rx})

for {
    if !left.Get() {
        uart.Write([]byte{1})
    }
    if !right.Get() {
        uart.Write([]byte{2})
    }
}
```

Read Serial

```go
c := &serial.Config{Name: "/dev/ttyACM0", Baud: 115200}
s, err := serial.OpenPort(c)
if err != nil {
    log.Fatal(err)
}
for {
    buf := make([]byte, 5)
    n, err := s.Read(buf)
    if err != nil {
        log.Fatal(err)
    }
    log.Printf("%q", buf[:n])
}
```

https://github.com/tarm/serial

Golang and IoT?
Yes!

THANKS

Any questions?

You can find details:

- https://github.com/arsoba/golang-ble-demo
- https://tinygo.org/