KubeCon | CloudNativeCon

Europe 2018

Efficient IoT with Protocol Buffers and gRPC

Vladimir Vivien (VMware)

About me

Software Engineer @VMware (CNX)

Go / Author / Kubernetes

@VladimirVivien

Objective

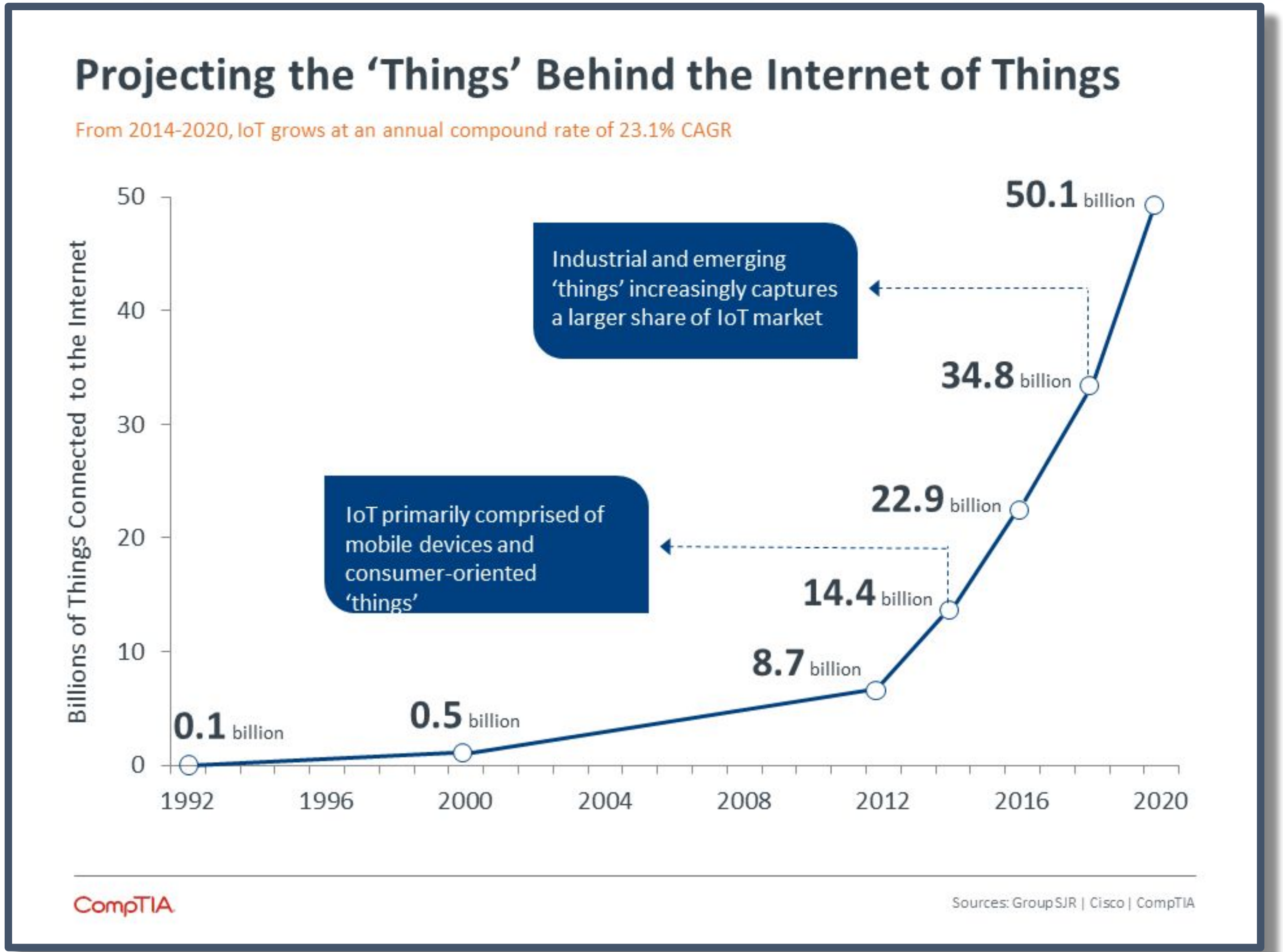
Explore the use of Protocol Buffers and gRPC for efficient IoT.

Internet of all the things

Not just PCs and servers

Beside the traditional computer, more things are getting connected to the Internet.

Internet of all the things



Source [CompTIA.Org](https://www.comptia.org) - https://www.comptia.org/resources/internet-of-things-insights-and-opportunities

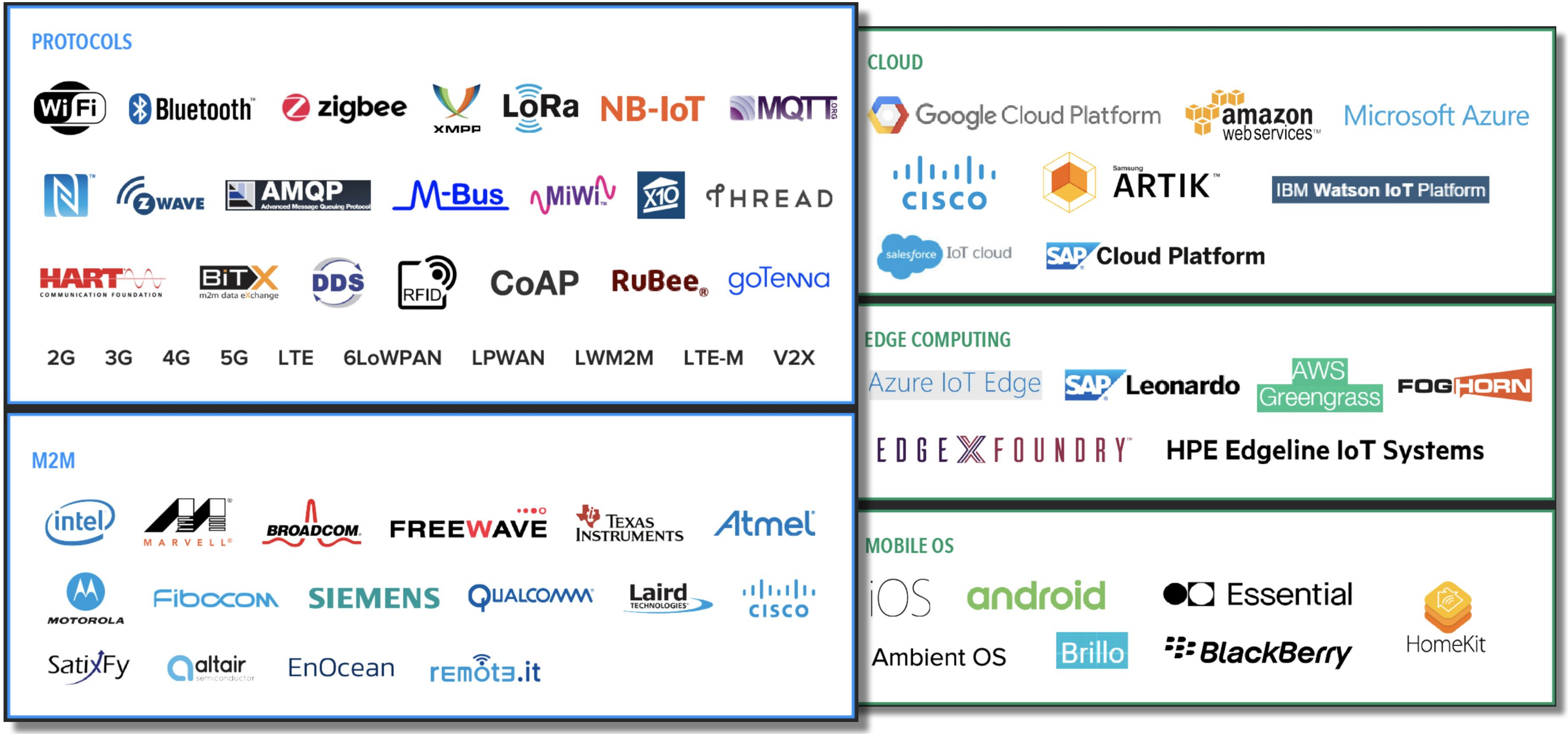
Explosion of chipsets, sensors, and dev platforms

IoT landscape by Matt Turck, Demi Obayomi, and FirstMark Capital - http://mattturck.com/iot2018/

Multitude of protocols

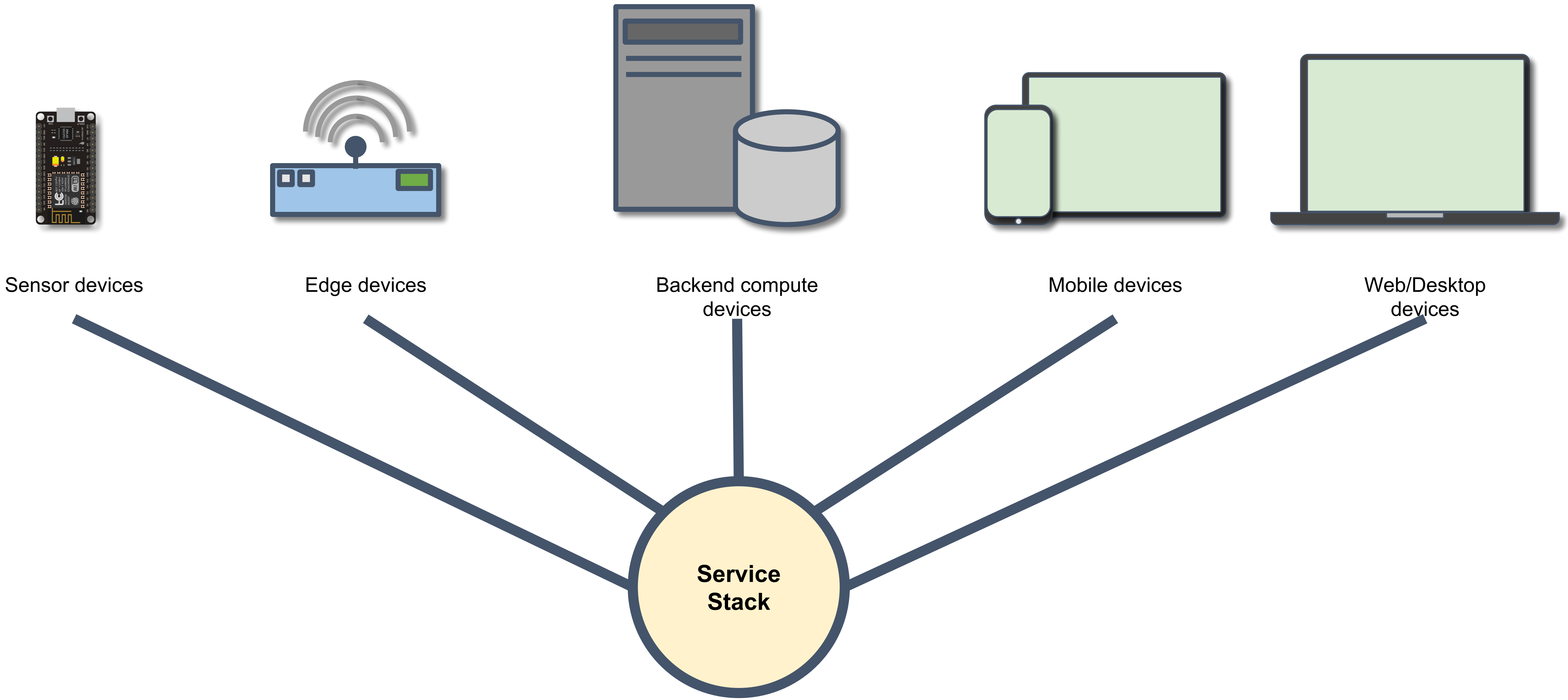
IoT landscape by Matt Turck, Demi Obayomi, and FirstMark Capital - http://mattturck.com/iot2018/

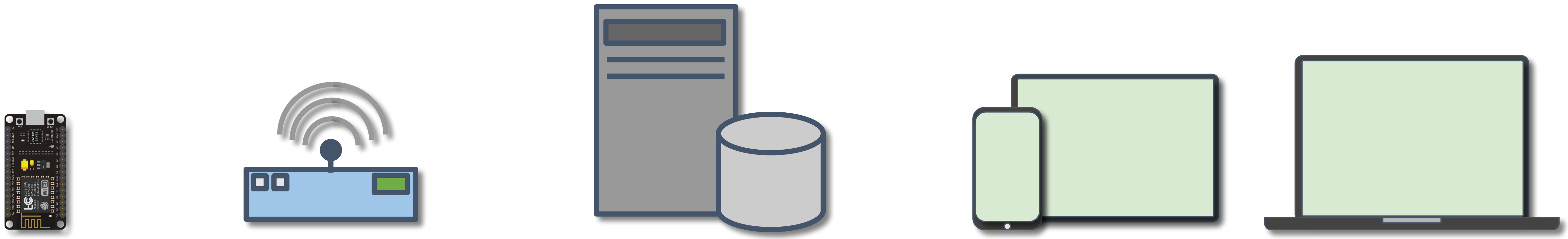
Multitude of protocols and platforms



IoT landscape by Matt Turck, Demi Obayomi, and FirstMark Capital - http://mattturck.com/iot2018/

A modern IoT Service


Sensor devices
Edge devices
Backend compute devices
Mobile devices
Web/Desktop devices
Service Stack



The problem

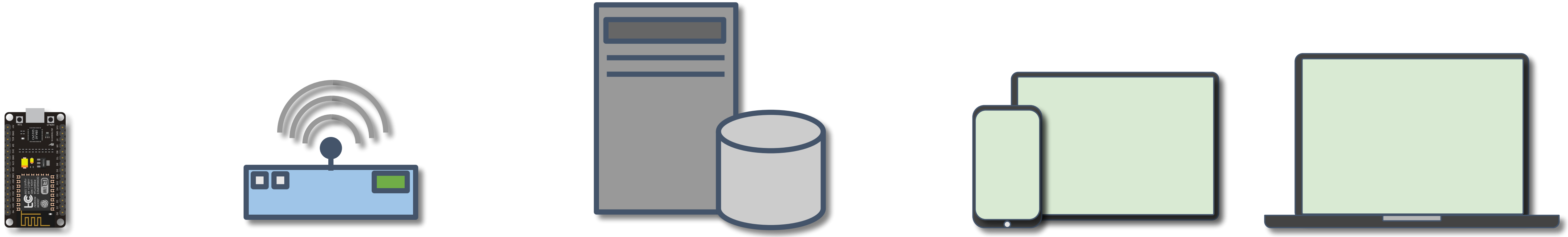
Heterogeneous devices

Varied constraint requirements

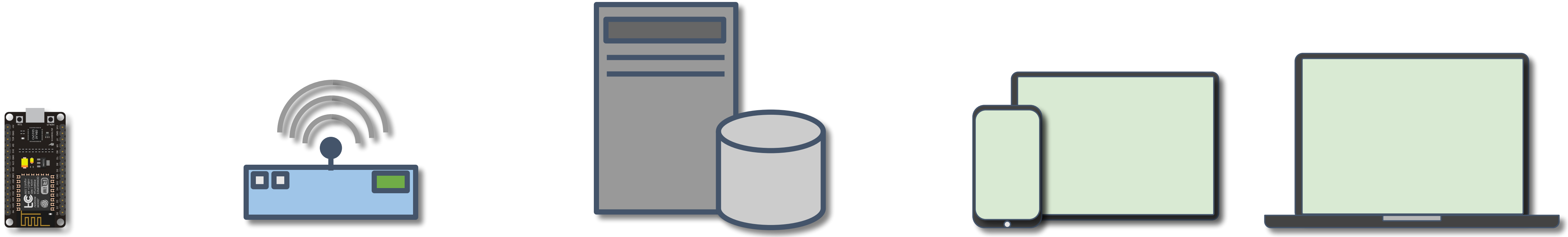
Diverse OS platforms (some no OS)

Many computing languages

Need for translation layer



The opportunity



The opportunity

Uniform communication and interoperability between devices and service components (using Protocol Buffers + gRPC).

Protocol buffers

Protocol buffers

A language and platform neutral mechanism for binary serialization of structured data.

Protocol buffers

Open source, created at Google

An efficient binary format

Allows serialization of typed data structures

Supports many languages and platforms

Ideal storage and wire format for IoT

Protocol buffers

KubeCon CloudNativeCon Europe 2018

Open source, created at Google

An efficient binary format

Wait, what about JSON?

Supports many languages and platforms

Ideal storage and wire format for IoT

JSON for IoT data serialization

It’s a good solution.

JSON is simple, flexible, and universally accepted approach with a healthy ecosystem built around it.

But ...

- JSON has weak data typing
- Can be inefficient (text-based encoding)
- Clients can be inconsistently implemented
- Data versioning, update, and backward compatibility are problematic

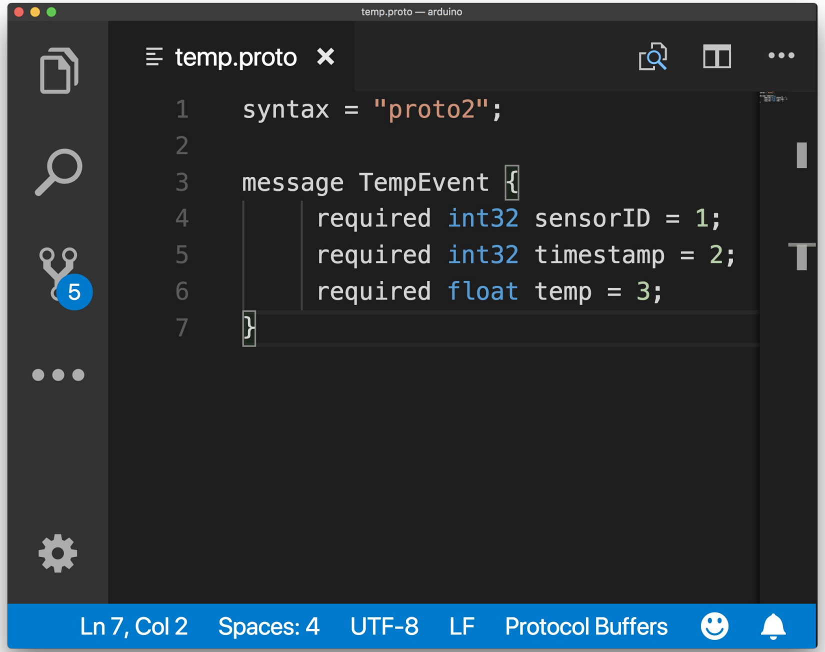
Protocol buffers and IoT

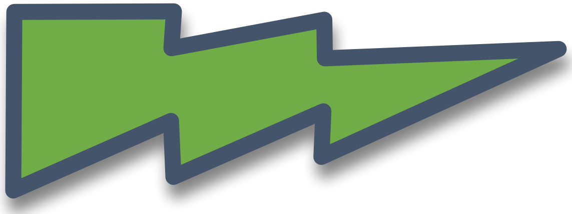
Protocol buffers + IoT

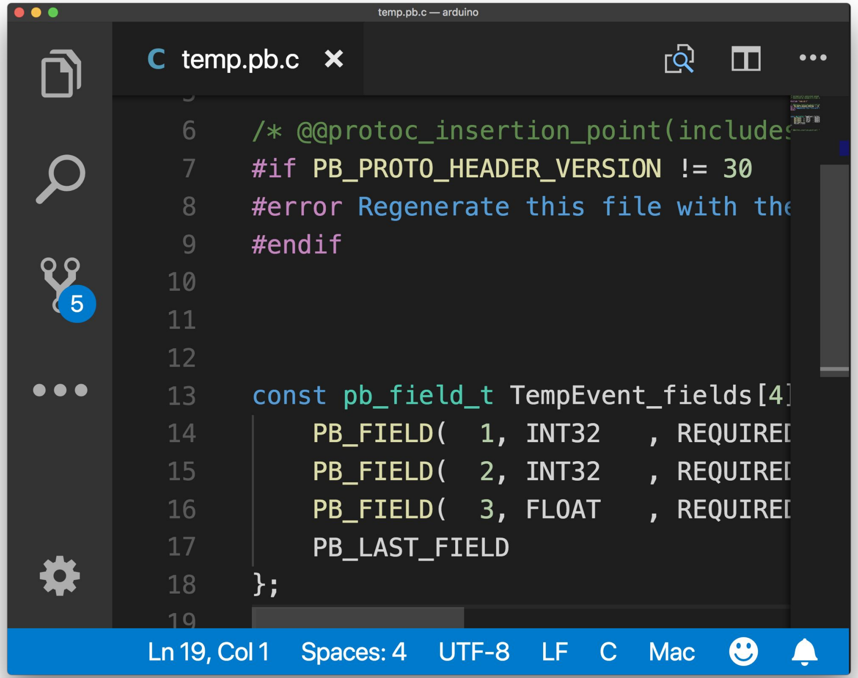
Protocol buffers are ideal for serializing data from IoT devices for logging, metrics, monitoring, etc.

Using protocol buffers

Generally involves 3 steps







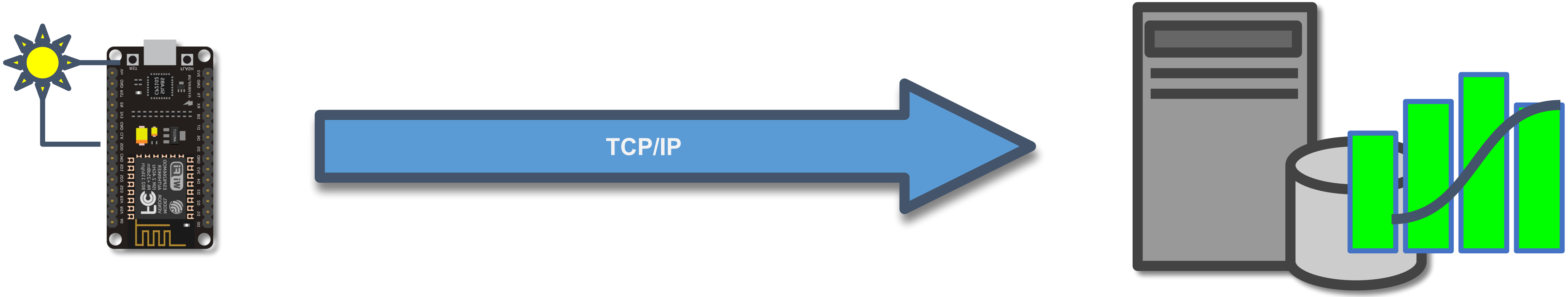
1 Define IDL

2 Compile

3 Integrate

Example: collect temperature data

Example: collect temperature data



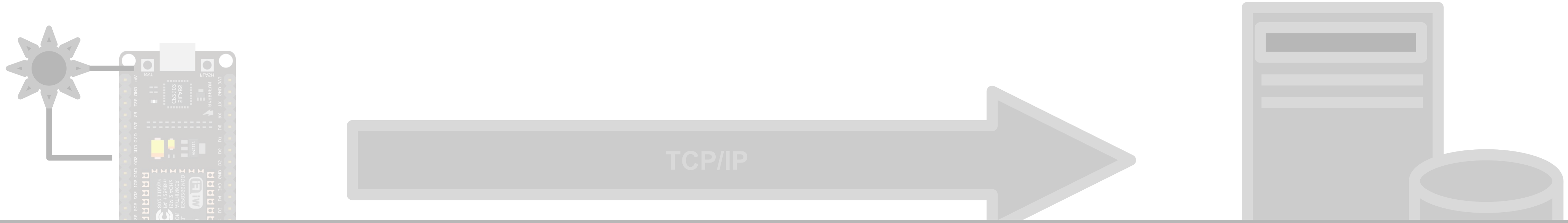
Collect temperature data from microcontroller

Use protocol buffers to serialize data on device

Send data over TCP/IP to remote server

Post data to time series database for visualization

Example: collect temperature data



A look at the server code

Use protocol buffers to serialize data on device

Send data over TCP/IP to remote server

Post data to time series database for visualization

Example: the Go server code

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

1 Define IDL

2 protoc

temp.pb.go

```
const _ = proto.ProtoPackageIsVersion2 //

type TempEvent struct {
    DeviceId          *int32   `protobuf:"
    EventId           *int32   `protobuf:"
    Humidity          *float32 `protobuf:"
    TempCel           *float32 `protobuf:"
    HeatIdxCel        *float32 `protobuf:"
    XXX_unrecognized  []byte   `json:"-"`
}
```

3 Integrate Go code

Example: the Go server code

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

Declare protocol buffers version

A message represents a structured container type.

Each message contains definition of typed values to be encoded.

Example: the Go server code

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel =
}
```

temp.pb.go

```
const _ = proto.ProtoPackageIsVersion2 //

type TempEvent struct {
    DeviceId         *int32   `protobuf:"
    EventId          *int32   `protobuf:"
    Humidity         *float32 `protobuf:"
    TempCel          *float32 `protobuf:"
    HeatIdxCel       *float32 `protobuf:"
    XXX_unrecognized []byte   `json:"-"`
}
```

1 Define

```
protoc --go_out=. temp.proto
```

Example: the Go server code

Generated Go type that represents data to be encoded.

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

1 Define DSL

2 pr

temp.pb.go

```go
const _ = proto.ProtoPackageIsVersion2 //

type TempEvent struct {
	DeviceId           *int32   `protobuf:"v
	EventId            *int32   `protobuf:"v
	Humidity           *float32 `protobuf:"
	TempCel            *float32 `protobuf:"
	HeatIdxCel         *float32 `protobuf:"
	XXX_unrecognized   []byte   `json:"-"`
}
```

Example: the Go server code

Deserialize data from remote device into generated type using protocol buffers library.

tempsvr.go

```go
    n, err := conn.Read(buf)
    if err != nil {
        log.Println(err)
        return
    }
    if n <= 0 {
        log.Println("no data received")
        return
    }

    var e temp.TempEvent
    if err := proto.Unmarshal(buf[:n], &e); err != nil {
        log.Println("failed to unmarshal:", err)
        return
    }
```

Example: the Go server code

KubeCon CloudNativeCon Europe 2018

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
```

Post temperature data to timeseries server (influxDB).

psvr.go

```go
func postEvent(e temp.TempEvent) error {
    if db != nil {
        log.Println("posting temp event to influxDB")
        // Create a new point batch
        bp, err := influx.NewBatchPoints(influx.Batch
            Database:  "dht11",
            Precision: "s",
        })
        if err != nil {
            return err
        }
```

Example: the Go server code

KubeCon CloudNativeCon Europe 2018

Let’s look at the device

Example: the device

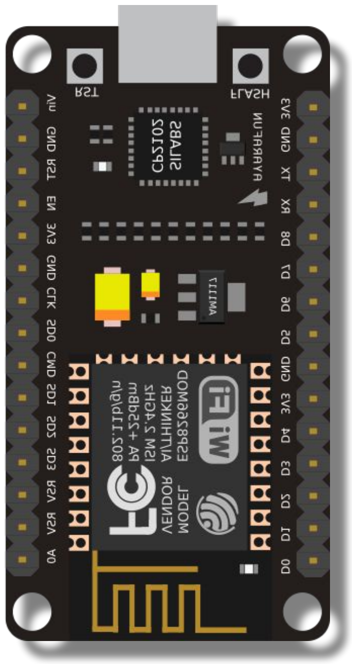
Low cost microcontroller devices (i.e. ESP8266)

80 MHz, 32 KBi

WIFI radio, with supports for TCP/IP

No operating system

Programmed in C/C++/Arduino



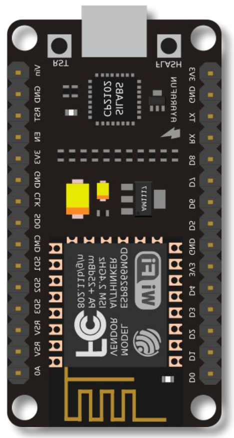
Example: programming the device

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

temp.pb.h

```
typedef struct _pb_TempEvent {
    int32_t deviceId;
    int32_t eventId;
    float humidity;
    float tempCel;
    float heatIdxCel;
/* @@protoc_insertion_point(struct:pb_T
} pb_TempEvent;
```



1 Define IDL 2 protoc 3 Integrate C code 4 Deploy

Example: programming the device

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

Use same IDL as before.

```
edef struct _pb_TempEvent {
 int32_t deviceId;
 int32_t eventId;
 float humidity;
 float tempCel;
 float heatIdxCel;
 @@protoc_insertion_point(struct:pb_T
b_TempEvent;
```

ntegrate C code

4 Deploy

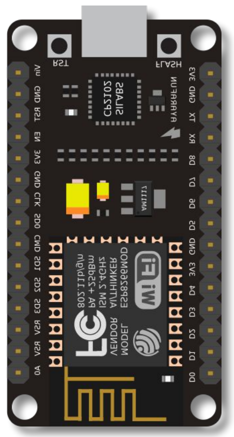
Example: programming the device

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

temp.pb.h

```
typedef struct _pb_TempEvent {
    int32_t deviceId;
    int32_t eventId;
    float humidity;
    float tempCel;
    float heatIdxCel;
/* @@protoc_insertion_point(struct:pb_T
} pb_TempEvent;
```



1 Define IDL

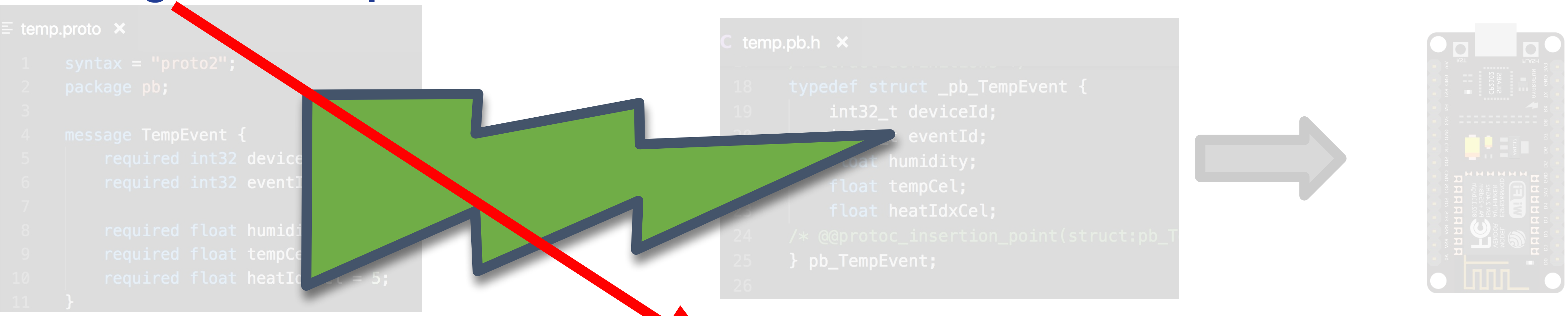
2 protoc

3 Integrate C code

4 Deploy

Example: programming the device

Use the *nanopb* protoc plugin to generate C protobuf serializers.




```
protoc --plugin=protoc-gen-nanopb=\
    ~/nanopb/generator/protoc-gen-nanopb \
    --nanopb_out=. temp.proto
```

Example: programming the device

KubeCon CloudNativeCon Europe 2018

Generated C type for data encoding.

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

1 Define DSL

2 protoc

temp.pb.h

```
typedef struct _pb_TempEvent {
    int32_t deviceId;
    int32_t eventId;
    float humidity;
    float tempCel;
    float heatIdxCel;
/* @@protoc_insertion_point(struct:pb_T
} pb_TempEvent;
```

Example: programming the device

Encode temperature data as protocol buffers on device.

Send the temperature data to remote server.

dht11_proto.ino

```
void sendTemp(pb_TempEvent e) {
  uint8_t buffer[128];
  pb_ostream_t stream = pb_ostream_from_buffer(buffer,

  if (!pb_encode(&stream, pb_TempEvent_fields, &e)){
    Serial.println("failed to encode temp proto");
    Serial.println(PB_GET_ERROR(&stream));
    return;
  }

  Serial.print("sending temp...");
  Serial.println(e.tempCel);
  client.write(buffer, stream.bytes_written);
}
```

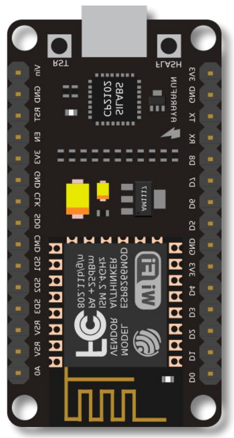
Example: programming the device

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

temp.pb.h

```
typedef struct _pb_TempEvent {
    int32_t deviceId;
    int32_t eventId;
    float humidity;
    float tempCel;
    float heatIdxCel;
/* @@protoc_insertion_point(struct:pb_T
} pb_TempEvent;

```



1 Define IDL

2 protoc

3 Integrate C code

4 Deploy

Example: programming the device

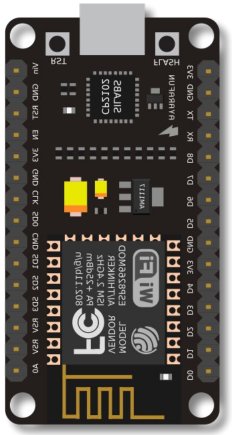
KubeCon CloudNativeCon Europe 2018

temp.proto

```
syntax = "proto2";
package pb;

message TempEvent {
    required int32 deviceId = 1;
    required int32 eventId = 2;

    required float humidity = 3;
    required float tempCel = 4;
    required float heatIdxCel = 5;
}
```

temp.pb.h

```
typedef struct _pb_TempEvent {
    int32_t deviceId;
    int32_t eventId;
    float humidity;
    float tempCel;
    float heatIdxCel;
/* @@protoc_insertion_point(struct:pb_T
} pb_TempEvent;
```



1 Define IDL

2 protoc

3 Integrate C code

4 Deploy

Example: programming the device

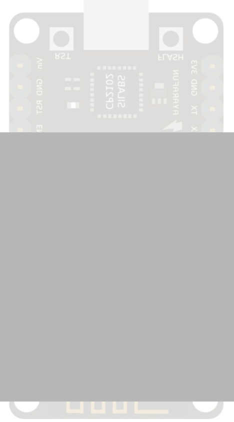
temp.proto

```
syntax = "proto2";
package pb;
```

```
}
```

temp.pb.h

```
typedef struct _pb_TempEvent {
```



How does it all work?

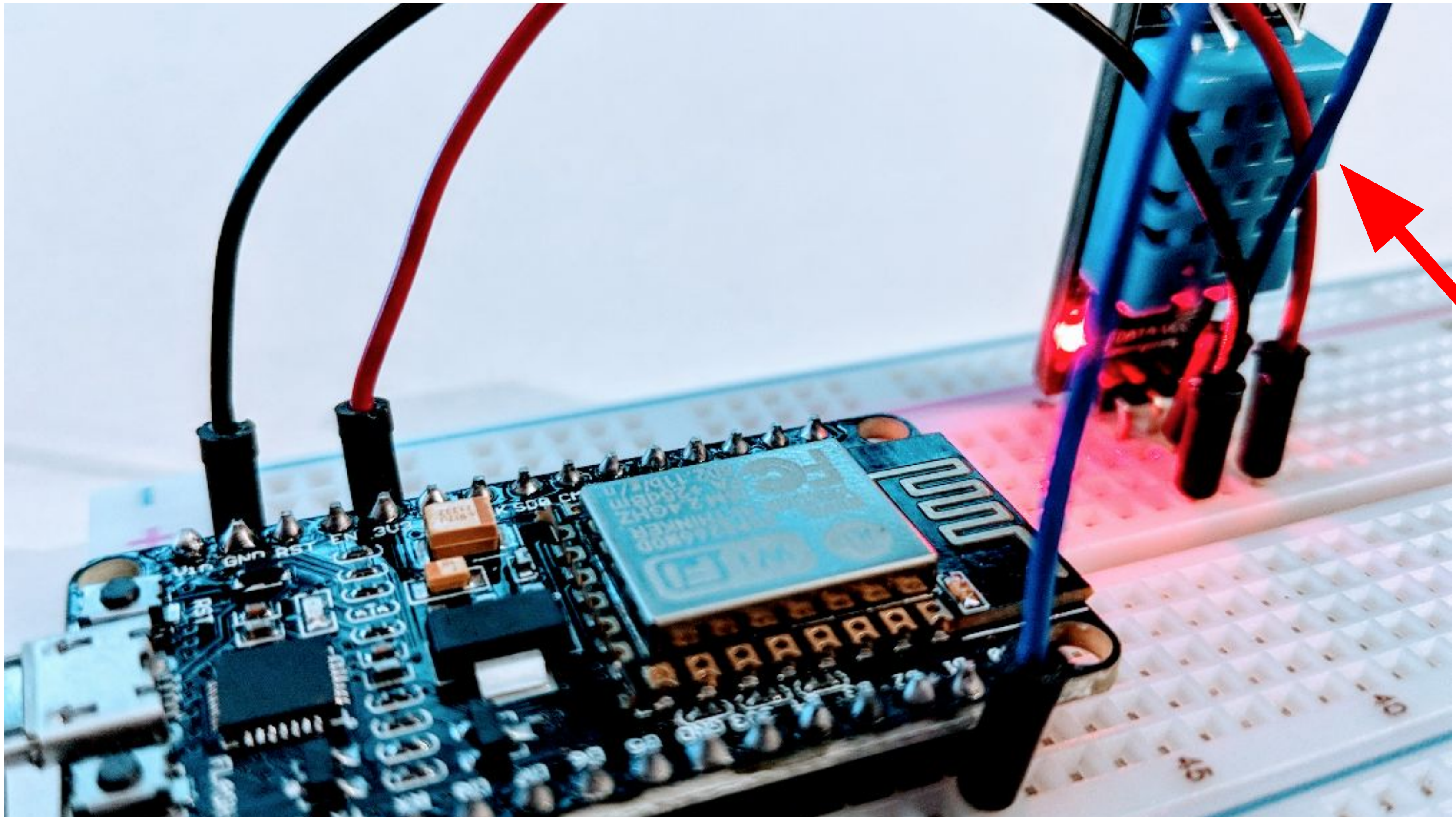
1 Define IDL

2 protoc

3 Integrate C code

4 Deploy

Start the device



Temp sensor (DHT11)

Start grafana

```
> grafana-server --config=/usr/local/etc/grafana/grafana.ini --homepath /usr/local/shar
INFO[04-22|22:30:00] Starting Grafana                         logger=server version=5.0
INFO[04-22|22:30:00] Config loaded from                       logger=settings file=/usr
INFO[04-22|22:30:00] Config loaded from                       logger=settings file=/usr
INFO[04-22|22:30:00] Path Home                                logger=settings path=/usr
INFO[04-22|22:30:00] Path Data                                logger=settings path=/usr
INFO[04-22|22:30:00] Path Logs                                logger=settings path=/usr
INFO[04-22|22:30:00] Path Plugins                             logger=settings path=/usr
INFO[04-22|22:30:00] Path Provisioning                        logger=settings path=/usr
INFO[04-22|22:30:00] App mode production                      logger=settings
INFO[04-22|22:30:00] Initializing DB                          logger=sqlstore dbtype=sq
INFO[04-22|22:30:00] Starting DB migration                    logger=migrator
INFO[04-22|22:30:00] Executing migration                      logger=migrator id="creat
INFO[04-22|22:30:00] Executing migration                      logger=migrator id="creat
INFO[04-22|22:30:00] Executing migration                      logger=migrator id="add u
INFO[04-22|22:30:00] Executing migration                      logger=migrator id="add u
INFO[04-22|22:30:00] Executing migration                      logger=migrator id="drop
```

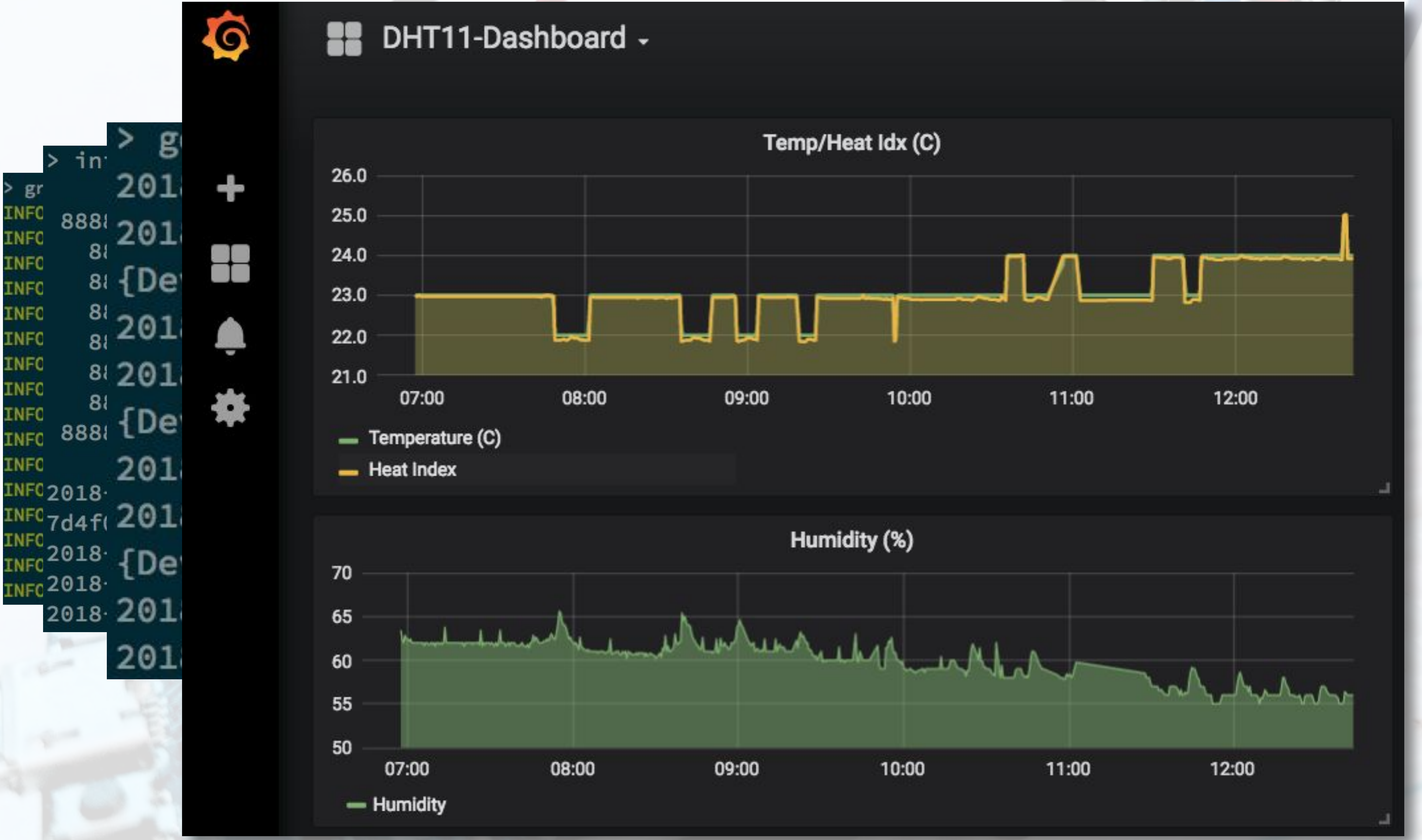
Start InfluxDB

```
> grafana
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
INFO[04-2
```

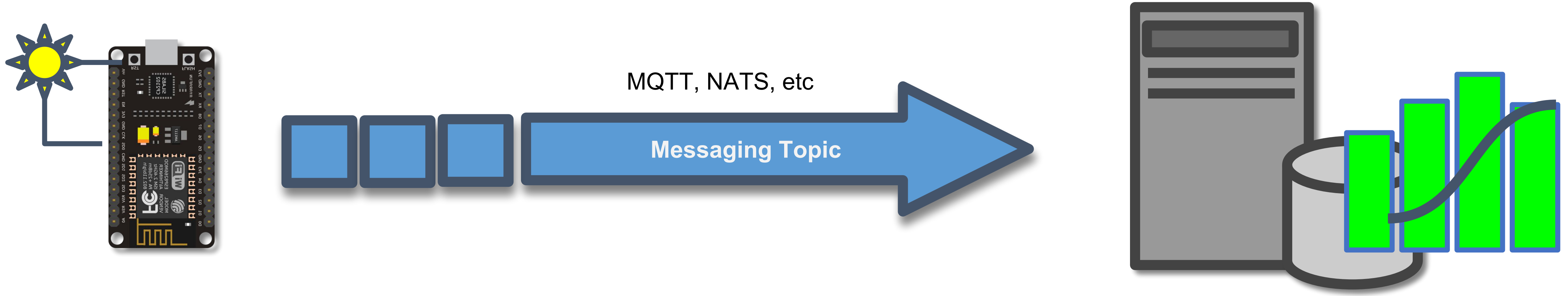
```
> influxd

 8888888           .d888 888                   8888888b.  888888b.
   888            d88P"  888                   888  "Y88b 888  "88b
   888            888    888                   888    888 888  .88P
   888   88888b.  888888 888 888  888 888  888 888    888 8888888K.
   888   888 "88b 888    888 888  888  Y8bd8P' 888    888 888  "Y88b
   888   888  888 888    888 888  888   X88K   888    888 888    888
   888   888  888 888    888 Y88b 888 .d8""8b. 888  .d88P 888   d88P
 8888888 888  888 888    888  "Y88888 888  888 8888888P"  8888888P"

2018-04-23T02:16:16.407987Z     info    InfluxDB starting       {"log
7d4f043b34ecb4e9b2dbec298c6f9450c2a32"}
2018-04-23T02:16:16.408020Z     info    Go runtime      {"log_id": "0
2018-04-23T02:16:16.509936Z     info    Using data dir  {"log_id": "0
2018-04-23T02:16:16.509984Z     info    Open store (start)      {"log
```

Run the server

```
> go run tempsvr.go
2018/04/22 21:57:42 Temperator Service started: (tcp)
2018/04/22 21:57:46 Connected to  192.168.1.115:49199
{DeviceID:12, EventID:100, Temp: 24.00, Humidity:57.00
2018/04/22 21:57:46 INFO: closing connection
2018/04/22 21:57:52 Connected to  192.168.1.115:49200
{DeviceID:12, EventID:100, Temp: 24.00, Humidity:56.00
2018/04/22 21:57:52 INFO: closing connection
2018/04/22 21:57:57 Connected to  192.168.1.115:49201
{DeviceID:12, EventID:100, Temp: 24.00, Humidity:57.00
2018/04/22 21:57:57 INFO: closing connection
2018/04/22 21:58:03 Connected to  192.168.1.115:49202
```


DHT11-Dashboard
Temp/Heat Idx (C)
26.0
25.0
24.0
23.0
22.0
21.0
07:00
08:00
09:00
10:00
11:00
12:00
Temperature (C)
Heat Index
Humidity (%)
70
65
60
55
50
07:00
08:00
09:00
10:00
11:00
12:00
Humidity

Optional setup with messaging



gRPC

A universal open-source RPC framework designed to create efficient and fast real time services.

gRPC

Uses protocol buffers for efficient binary encoding

Defines service and methods using IDL

Relies on HTTP/2 for fast multiplexed transport

Ability for bi-directional and data streaming

Extensible middleware for authN, authZ, tracing, logging, etc

Support for 11 languages (and counting)

gRPC and IoT

gRPC + IoT

gRPC goes beyond the gather/analyze model and makes it possible to build real-time interactive IoT services.

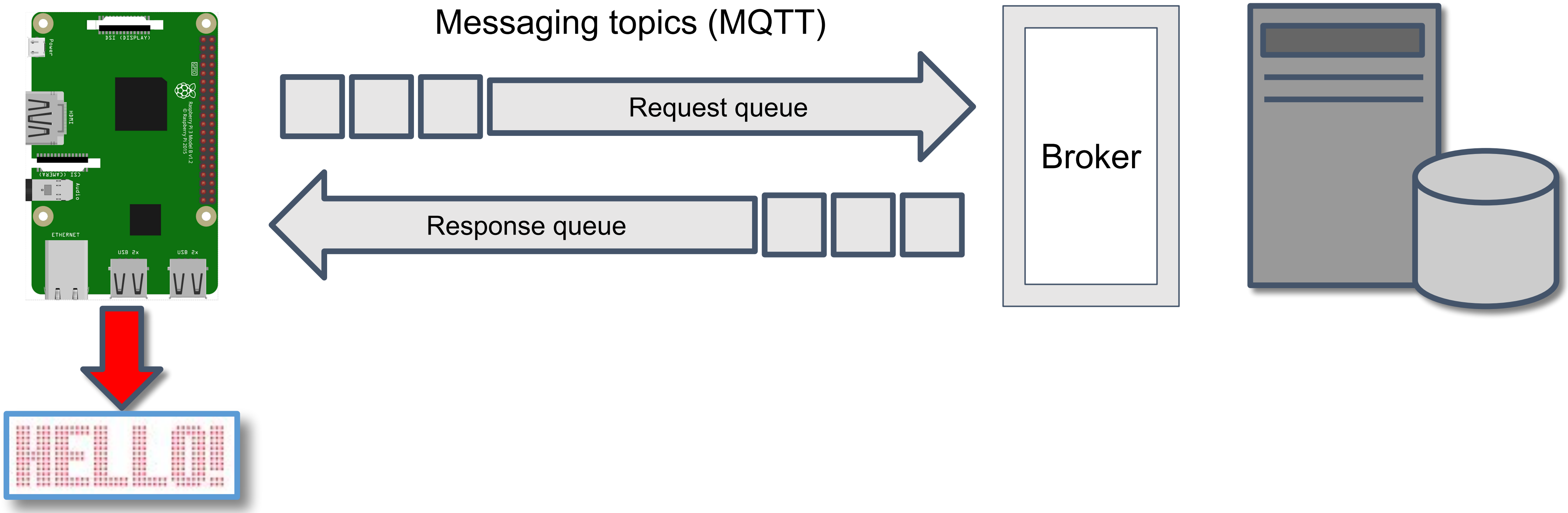
gRPC + IoT

gRPC goes beyond the gather/analyze

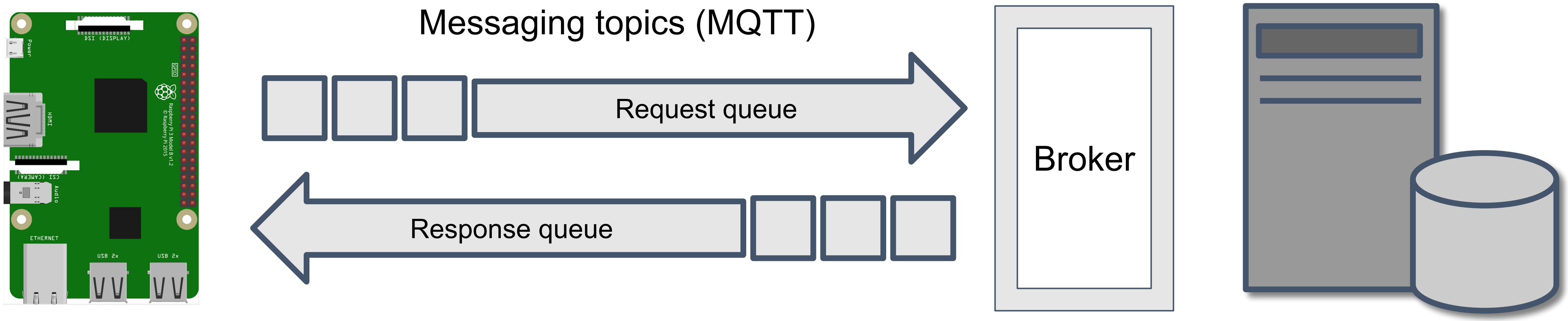
What does it all mean?

real-time IoT applications.

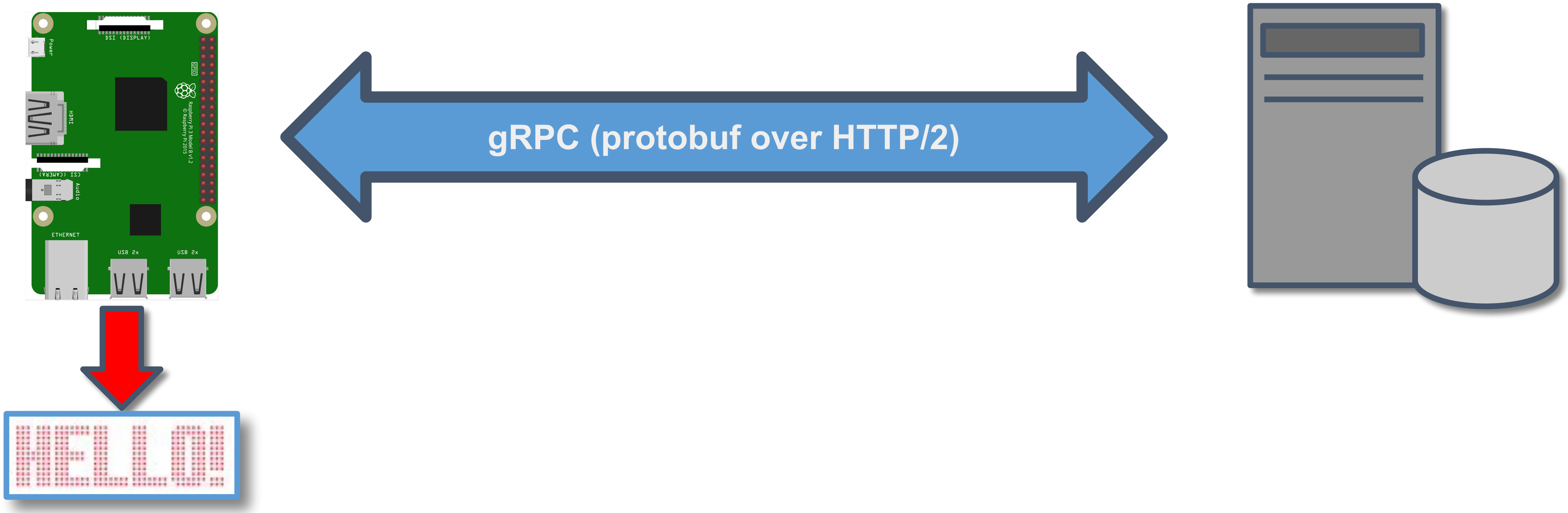
Common interactive IoT setup



Common IoT setup



gRPC IoT service setup



Using gRPC

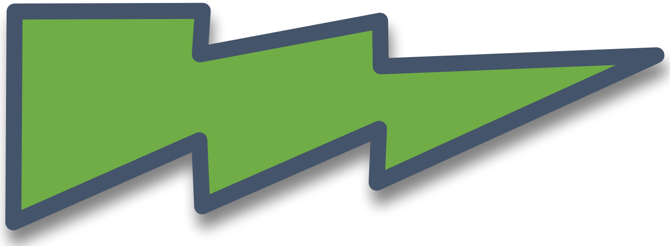
Generally involves 3 steps

Using gRPC

Generally involves 3 steps

cloud_speech.proto

```
syntax = "proto3";
package google.cloud.speech.v1;

// Service that implements Google Cloud Speech API.
service Speech {
  rpc Recognize(RecognizeRequest) returns (RecognizeResponse) {
    option (google.api.http) = { post: "/v1/speech:recognize" body:
  }
  rpc LongRunningRecognize(LongRunningRecognizeRequest) returns (g
    option (google.api.http) = { post: "/v1/speech:longrunningrecog
  }
  rpc StreamingRecognize(stream StreamingRecognizeRequest) returns
}

// The top-level message sent by the client for the `Recognize` met
message RecognizeRequest {
  RecognitionConfig config = 1;
  RecognitionAudio audio = 2;
}
```



cloud_speech_pb2_grpc.py

```
# Generated by the gRPC Python protocol compiler plugin.
import grpc

import google.cloud.speech_v1.proto.cloud_speech_pb2 as
import google.longrunning.operations_pb2 as google_dot_l


class SpeechStub(object):
  """Service that implements Google Cloud Speech API.
  """

  def __init__(self, channel):
    """Constructor.
```

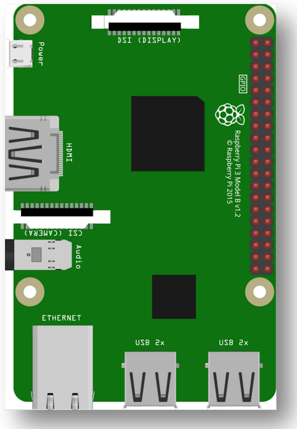
1 Define IDL

2 Compile

3 Integrate stubs

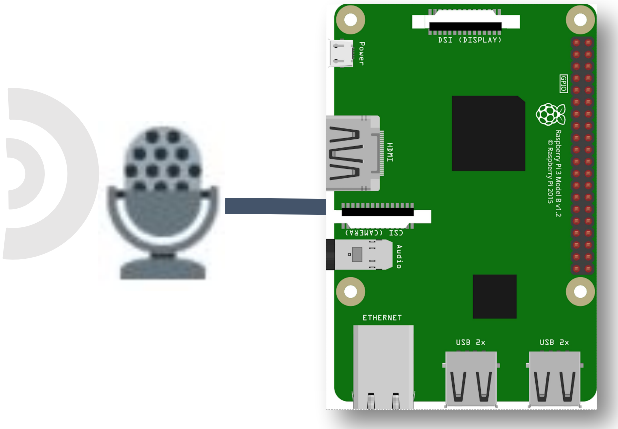
Example:
speech transcription with gRPC

Example: Raspberry Pi speech transcription



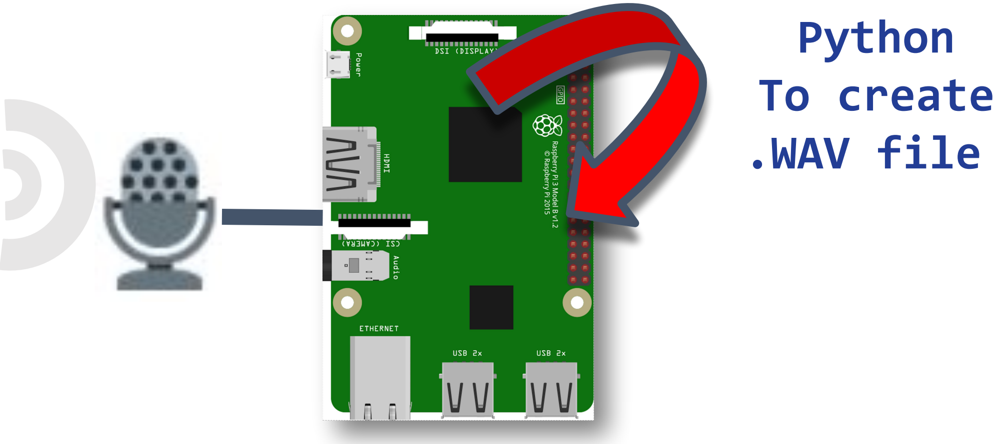
Raspberry Pi 3

Example: Raspberry Pi speech transcription



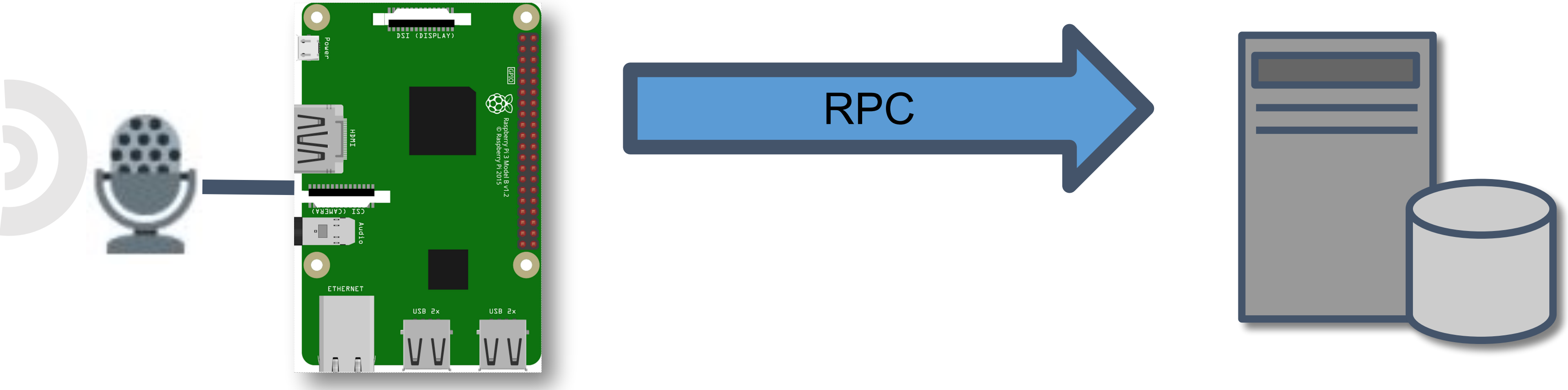
Raspberry Pi 3

Example: Raspberry Pi speech transcription



Raspberry Pi 3

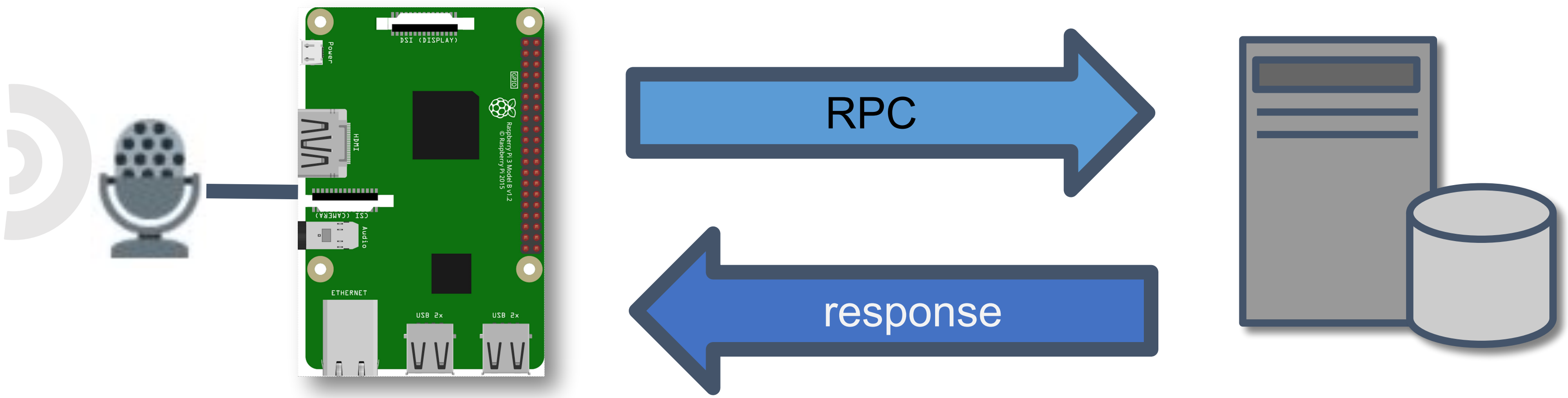
Example: Raspberry Pi speech transcription



Raspberry Pi 3

Google Cloud
Speech-to-Text Service

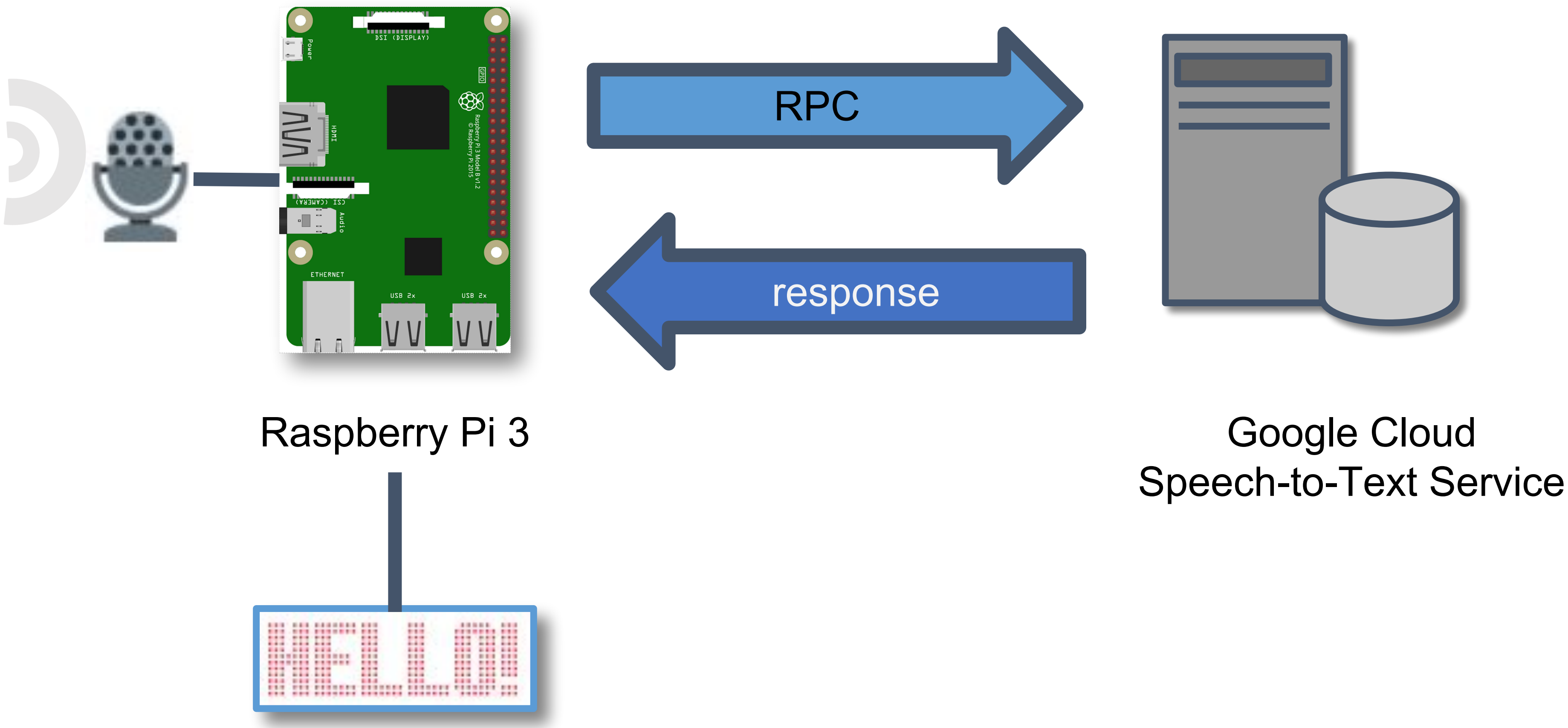
Example: Raspberry Pi speech transcription



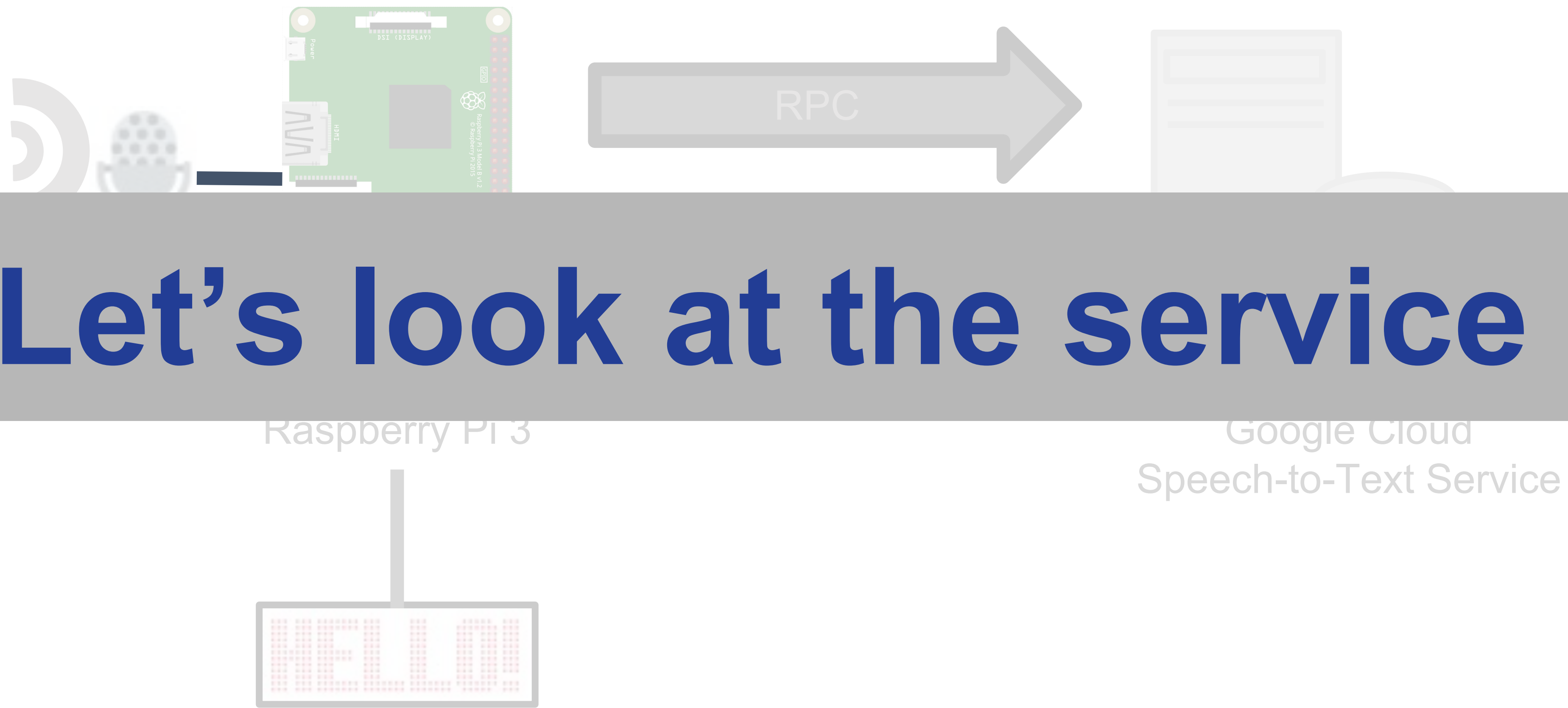
Raspberry Pi 3

Google Cloud
Speech-to-Text Service

Example: Raspberry Pi speech transcription



Example: Raspberry Pi speech transcription



Let’s look at the service

Google Cloud Speech-to-Text

Cloud-based speech-to-text service

Real-time speech transcription

Available RPC API via gRPC

Libraries for several languages (Go, Python, C#, etc)

Google Cloud Speech-to-Text

Cloud-based speech-to-text service

Real-time speech transcription

Now, the device

The device: Raspberry Pi

Raspberry Pi 3

Support for WIFI, Ethernet

Full blown Linux OS

The device: Raspberry Pi

Raspberry Pi 3

Support for WIFI, Ethernet

Full blown Linux OS

Python gRPC speech client (from Google Speech)

Programming the device

KubeCon CloudNativeCon
Europe 2018

Provided by Google Cloud

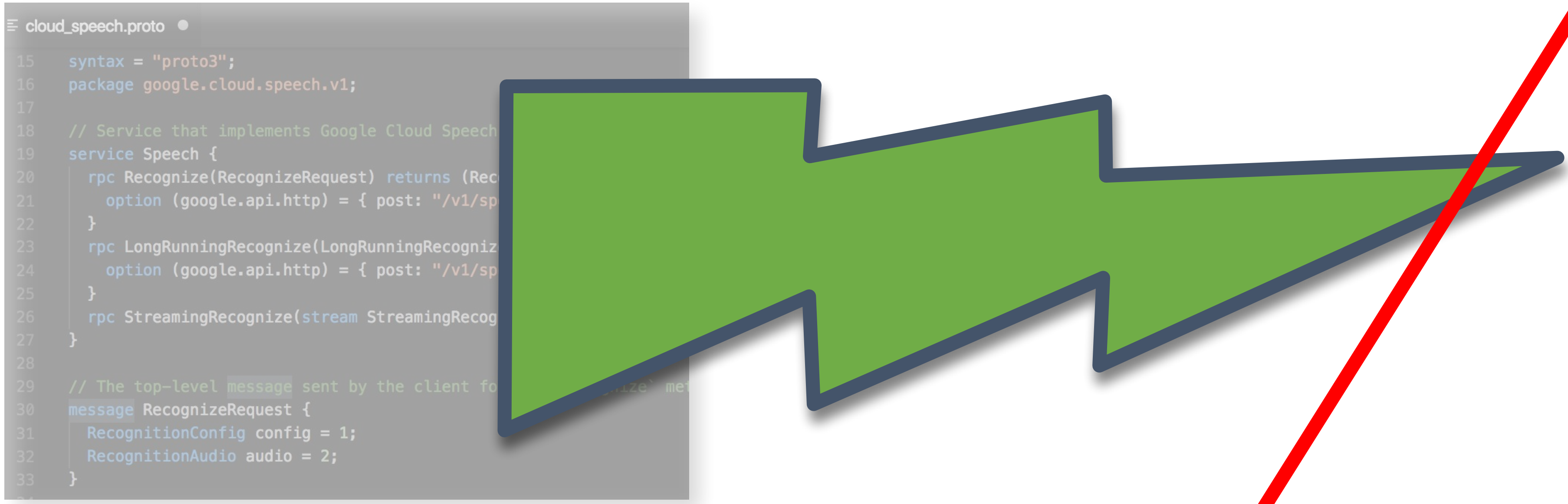
cloud_speech.proto

```
syntax = "proto3";
package google.cloud.speech.v1;

// Service that implements Google Cloud Speech API.
service Speech {
  rpc Recognize(RecognizeRequest) returns (RecognizeResponse) {
    option (google.api.http) = { post: "/v1/speech:recognize" body:
  }
  rpc LongRunningRecognize(LongRunningRecognizeRequest) returns (go
    option (google.api.http) = { post: "/v1/speech:longrunningrecog
  }
  rpc StreamingRecognize(stream StreamingRecognizeRequest) returns
}

// The top-level message sent by the client for the `Recognize` met
message RecognizeRequest {
  RecognitionConfig config = 1;
  RecognitionAudio audio = 2;
}
```

1 Define IDL

Programming the device

Use Python gRPC plugin to generate client stubs



1 Define

```
$> python -m grpc.tools.protoc  \
    --python_out=. --grpc_python_out=. \
    --proto_path=protobuf greeter.proto
```

Programming the device

cloud_speech.proto

```
syntax = "proto3";
package google.cloud.speech.v1;

// Service that implements Google Cloud Speech API.
service Speech {
  rpc Recognize(RecognizeRequest) returns (RecognizeResponse) {
    option (google.api.http) = { post: "/v1/speech:recognize" body:
  }
  rpc LongRunningRecognize(LongRunningRecognizeRequest) returns (go
    option (google.api.http) = { post: "/v1/speech:longrunningrecog
  }
  rpc StreamingRecognize(stream StreamingRecognizeRequest) returns
}

// The top-level message sent by the client for the `Recognize` met
message RecognizeRequest {
  RecognitionConfig config = 1;
  RecognitionAudio audio = 2;
}
```

cloud_speech_pb2_grpc.py

```
# Generated by the gRPC Python protocol compiler plugin.
import grpc

import google.cloud.speech_v1.proto.cloud_speech_pb2 as
import google.longrunning.operations_pb2 as google_dot_l


class SpeechStub(object):
  """Service that implements Google Cloud Speech API.
  """

  def __init__(self, channel):
    """Constructor.
```

1 Define IDL

2 Compile

3 Integrate client stub

Programming the device

cloud_speech.proto

```
syntax = "proto3";
package google.cloud.speech.v1;

  RecognitionAudio audio = 2;
}
```

eech_pb2_grpc.py

```
erated by the gRPC Python protocol compiler plugin.
 grpc
```

Fortunately, stubs already compiled with client libraries!

1 Define IDL

Integrate client stub

Speech-to-Text with Python + gRPC

Speech-to-Text with Python + gRPC

transcribe.py

```python
import pyaudio
import wave
import signal
import sys
import io
import os

# Imports the Google Cloud client library
from google.cloud import speech
from google.cloud.speech import enums
from google.cloud.speech import types

```

Python audio libraries

Import Google Cloud Speech-to-Text Python libraries.

Speech-to-Text with Python + gRPC

```python
frames = []

p = pyaudio.PyAudio()

RATE = (int)(p.get_device_info_by_index(

stream = p.open(format=FORMAT,
                channels=CHANNELS,
                rate=RATE,
                input=True,
                frames_per_buffer=CHUNK)
```

Use PyAudio to capture audio data from microphone.

Speech-to-Text with Python + gRPC

KubeCon CloudNativeCon Europe 2018

transcribe.py ×

```python
def transcribe():
    client = speech.SpeechClient()

    file_name = os.path.join(
    os.path.dirname(__file__),
    './',
    WAVE_OUTPUT_FILENAME)

    with io.open(file_name, 'rb') as audi
        content = audio_file.read()
        audio = types.RecognitionAudio(co
```

Initialize speech client.

Speech-to-Text with Python + gRPC

KubeCon CloudNativeCon Europe 2018

Synchronous RPC to speech service which returns the response.

transcribe.py

```python
config = types.RecognitionConfig(
    encoding=enums.RecognitionConfig.AudioEncoding.LINEAR
    sample_rate_hertz=RATE,
    language_code='en-US')

response = client.recognize(config, audio)

print ("transcribe...")
for result in response.results:
    print('{}'.format(result.alternatives[0].transcript))
```

Speech-to-Text with Python + gRPC

KubeCon | CloudNativeCon Europe 2018

transcribe.py

Use RPC result to print transcription responses.

```
fig = types.RecognitionConfig(
    encoding=enums.RecognitionConfig.AudioEncoding.LINEAR
    sample_rate_hertz=RATE,
    language_code='en-US')

response = client.recognize(config, audio)

print ("transcribe...")
for result in response.results:
    print('{}'.format(result.alternatives[0].transcript))
```

Running the example

```
ALSA lib conf.c:5007:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM iec958
ALSA lib confmisc.c:1281:(snd_func_refer) Unable to find definition 'cards.bcm2835.pcm.
ALSA lib conf.c:4528:(_snd_config_evaluate) function snd_func_refer returned error: No
ALSA lib conf.c:5007:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM spdif
ALSA lib confmisc.c:1281:(snd_func_refer) Unable to find definition 'cards.bcm2835.pcm.
ALSA lib conf.c:4528:(_snd_config_evaluate) function snd_func_refer returned error: No
ALSA lib conf.c:5007:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM spdif
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.hdmi
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.hdmi
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.modem
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.modem
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.phoneline
ALSA lib pcm.c:2495:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.phoneline
* recording
44100
```

Thank you

Vladimir Vivien (@VladimirVivien)

https://github.com/vladimirvivien/iot-dev