Astarte

From data collection to fleet management

Riccardo Binetti

FOSDEM 2022

About me

Who am I

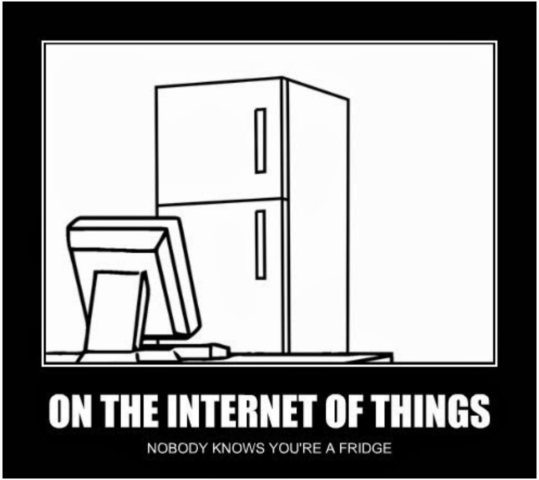
- AKA: rbino
  - Website: rbino.com
  - Github: rbino
  - Twitter: @errebino
- Cloud Specialist at SECO Mind (Padova, Italy)
- Started as embedded Qt/C++ developer
- Now almost full time Elixir developer

Why delivering data is not enough

IoT may seem like an already solved problem[1], but it's not.

[1] and an overused buzzword

Authentication



Data modeling

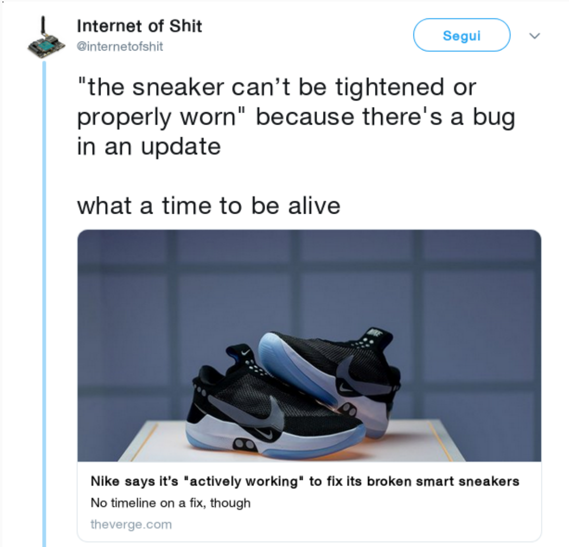
- Many IoT solutions don’t force you to declare the shape of data sent from and to the device
- This seems faster, but takes away guarantees from your system
- Think schema vs schemaless DBs

One size fits all approach

- Do you want to use our cool and shiny dashboard? You’re good to go
- Do you need some well structured APIs to build your own frontend? Not so fast
- IoT platforms are usually centered around a set of widgets used to expose data

Results (slightly dramatized)

- It's usually easier to roll your own IoT platform than to adapt an existing one, so there's lots of reinventing the wheel
- "Smart" devices are not so smart, and in a bunch of years you won't even have the choice of buying non-smart ones
- Our homes will become an ensemble of insecure devices many of which have blades or can start fires
- Singularity will happen
- You won't even be able to wear your own shoes to run away



Internet of Shit
@internetofshit

"the sneaker can’t be tightened or properly worn" because there's a bug in an update

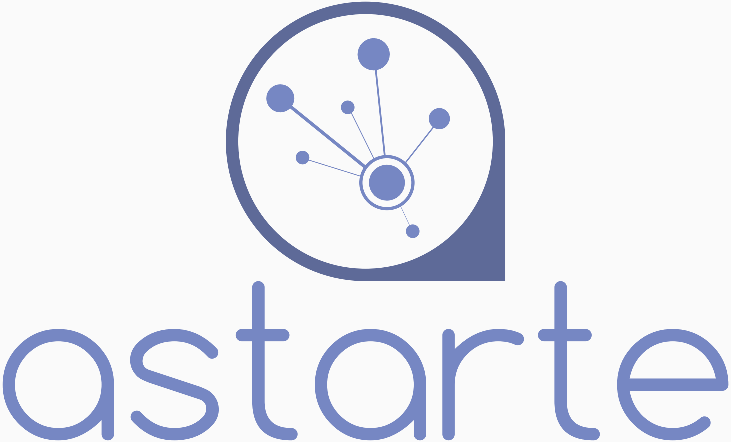
what a time to be alive

Nike says it’s "actively working" to fix its broken smart sneakers

No timeline on a fix, though

theverge.com

Introducing Astarte



Astarte is the open source Data Orchestration Platform focused on IoT

The Astarte approach

- Everyone has slightly different requirements for their IoT platform
  - Different device protocols
  - Different applications that use devices' data
- Astarte tries to solve this by focusing on the common building blocks and making it easy to integrate custom stuff you might need

Components

- Astarte
  - Set of microservices written in Elixir
  - Made to be modular and replaceable if needed
  - Designed to be Kubernetes native
- Device SDKs that cover a lot of different usecases
  - Qt5, C (ESP32), Go, Java, Elixir, Rust, Python, Javascript
- Astarte Flow
  - Data processing framework focused on reusable blocks and pipelines

Astarte concepts

Modeling data: Interfaces

- Data flowing in Astarte must adhere to predefined schemas, called interfaces, represented with a json file
- A device usually supports multiple interfaces, and declares its capabilities during connection (and each time they change)
- Interfaces can be shared across different devices
- Interface versioning adheres to semantic versioning
- The API that is generated for a device is derived from its interfaces

Datastream vs Properties

- Every interface belongs to one of two types: datastream or properties
- Datastream: stateless, ordered stream of data
  - All values are trasmitted and persisted
  - Used for: sensor samples, commands, events
- Properties: stateful, synchronized state with no history
  - Only the last value is required to be trasmitted and persisted
  - Used for: device state, policies depending only on the server side

```json
{
  "interface_name": "com.example.TempAndHumidity",
  "version_major": 1,
  "version_minor": 0,
  "type": "properties",
  "ownership": "device",
  "mappings": [
    {
      "endpoint": "/rooms/%{roomId}/humidity",
      "type": "double"
    },
    {
      "endpoint": "/rooms/%{roomId}/temperature",
      "type": "double"
    }
  ]
}
```

Credentials management: Pairing

- Pairing is the Astarte component dedicated to emitting and renewing credentials
- It signs SSL Client certificates used to perform SSL mutual authentication with the MQTT broker
- SDKs implement automatic credentials request and renewal

Pairing flow

- Agent: the entity that is in charge of registering the devices. It registers a device and obtains a credentials secret.
  - This operation can happen during manufacturing or directly on the device.
- The device exchanges its credentials secret with actual credentials
  - In the SSL mutual authentication case, it sends the credentials secret and a CSR, Pairing signs the CSR and returns a signed certificate to the device.
- Device credentials can (and should) be short lived since the device can request new credentials whenever it wants

React to device data: Triggers

- Triggers are a way to perform an action when a specific condition is met
- Conditions are defined by matching interfaces and paths, and optionally by defining simple operators (comparison, inclusion)
- Actions define what is executed when the condition is satisfied
- The currently supported actions allow to send an HTTP request to a URL or publishing a message to a RabbitMQ exchange

```json
{
  "name": "kitchen_high_humidity",
  "action": {
    "http_url": "https://example.com/webhooks",
    "http_method": "post"
  },
  "simple_triggers":
  [
    {
      "type": "data_trigger",
      "interface_name": "com.example.TempAndHumidity",
      "interface_major": 0,
      "on": "incoming_data",
      "match_path": "/rooms/kitchen/humidity"
    }
  ]
}
```

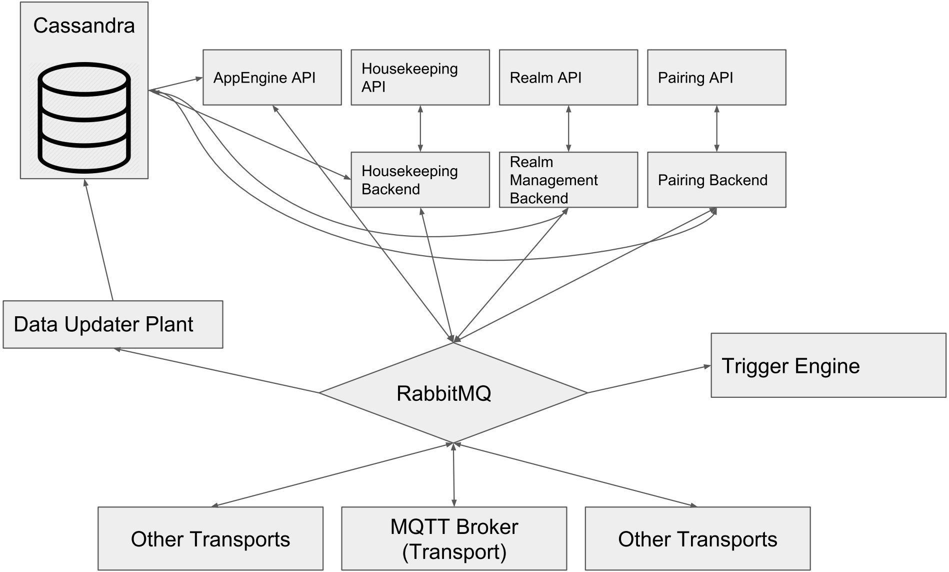
Transports

- The responsibility of a transport is converting incoming data to Astarte internal representation (BSON + AMQP metadata) and delivering it to RabbitMQ
- Currently, we provide an MQTT transport working with the available SDKs out of the box
  - Implemented as a VerneMQ[2] plugin
  - Mandatory SSL mutual authentication
  - The protocol is well documented[3] so you can roll your own compatible SDK

[2] https://vernemq.com/

[3] https://docs.astarte-platform.org/latest/080-mqtt-v1-protocol.html

Astarte components


Cassandra
AppEngine API
Housekeeping API
Realm API
Pairing API
Housekeeping Backend
Realm Management Backend
Pairing Backend
Data Updater Plant
RabbitMQ
Trigger Engine
Other Transports
MQTT Broker (Transport)
Other Transports

Cassandra/ScyllaDB

- All data is stored to Cassandra[4]
- We also support ScyllaDB[5] as drop-in replacement for Cassandra
- Tables are created based on interfaces, and data can be made expirable using Cassandra's TTL

[4] https://cassandra.apache.org/

[5] https://www.scylladb.com/

Housekeeping

- It’s the “superadmin” interface
- It is mainly used to manage realms and usually it’s not exposed to the end user
- “Realms” are used to isolate different users (they use differents keyspaces on Cassandra)

Realm Management

- It is the main administration interface exposed to the user
- It is used to manage realm configuration, interfaces and triggers

Pairing

- As described before, everything concerning device credentials resides here
- It is possible to also use it to inhibit devices, preventing them to obtain new credentials
- Devices use Pairing also to obtain information on their transports
    - For example, SDKs query Pairing at boot to ask the URL of the MQTT broker

Data Updater Plant

- It is the component that ingests all data coming from the devices
- It is called like that since a Data Updater BEAM process is spawned for every device
  - This ensures that a failure in a device does not disrupt other devices
- It is also able to communicate with the transports, so it is possible to force device disconnection when malformed data is received
- It verifies triggers' conditions and notifies Trigger Engine
- All data is guaranteed to be delivered (and saved to the DB) in the same order it was delivered to the transport

Trigger Engine

- It’s the component responsible of executing the trigger actions
- It performs HTTP requests when a trigger condition is met
- The event can be enriched with additional headers defined in the trigger

AppEngine

- It's the API used to retrieve data sent from devices and stream data towards them
- A REST tree is built based on the interfaces supported from the device
- The API always reflects the current state of the device
    - If a Device declares to support a new interface in its introspection, new data becomes accessible from AppEngine
- The other role of AppEngine is allowing data flow towards the device
    - Data sent to the device is also saved to the DB and can be accessed in the same way

AppEngine Websocket API

- Based on Phoenix Channels and built on top of trigger mechanism
- The user can create rooms and then subscribe to specific volatile triggers
- When a trigger condition is met, an event is streamed to the room

API authentication and authorization

- All APIs are authenticated using JWT tokens, but Astarte is not responsible of emitting them
- The public key to verify them is installed when a realm is created
- Basically, Astarte checks the JWT for a well-defined set of claims and allows or denies the actions based on them
- Claims are expressed with a combination of regular expressions matched on the path and allowed verbs on that path
  - Example: “a_aea”: “/devices/.*/com.example.TempAndHumidity/.*::GET”
- This makes it possible to decouple user management and using a dedicated tool for that (e.g. Keycloak[6])

[6] https://www.keycloak.org/

RabbitMQ RPC

- All components inside Astarte are connected through RabbitMQ
- The RPC protocol is defined using Protobuf schemas
- This allows to replace arbitrary Astarte components with custom ones, provided they support the same interface

Astarte Flow

- Data processing framework based on reusable blocks and pipelines
- Separate project from core Astarte microservices
- Astarte connected devices can be used as source or as destination
- Many other sources and destinations available (HTTP, plain MQTT, AMQP...)
- A block can also be a container image that is managed by the Kubernetes operator
    - Just take care of implementing the algorithm and let Astarte Flow take care of the plumbing

Deployment

- Astarte is designed to be a Kubernetes native application
- Every service is containerized with Docker
- All our production deployments are running on Kubernetes using our Kubernetes Operator
- There's also a docker-compose based deployment but it's mainly oriented towards testing Astarte quickly rather than using it in production

Monitoring

- All Astarte services expose Prometheus metrics
- Future versions will include integrations with the Prometheus Operator

Clients

- Astarte Dashboard
  - Web based client used to interact with Realm APIs
  - Deployed by default in clusters
- astartectl
  - CLI written in Go
  - Can be used to interact with Realm APIs or to simplify cluster management
- Both just use Astarte REST API (plus some kubectl magic in astartectl), so you can roll your own

Usecases: Edgehog

Edgehog device manager

- Edgehog is a device and fleet manager built on top of Astarte
- Heavily under development as I speak
- All device operations are implemented on top of Astarte
- On the device side, an Edgehog SDK built on top of the Astarte one is provided to simplify development
  - Currently the SDK is based on ESP32, future releases will also cover Linux
- The backend exposes a GraphQL API, which can also be used by third party clients
- The frontend interacts with the API allowing to perform all the operations easily

Astarte integration

- Devices connected to Edgehog use a well defined set of interfaces[7]
- Data is queried using Astarte AppEngine API
- Edgehog backend also receives trigger events to update the device connection status
- Commands and OTA are pushed with Astarte AppEngine API ability to send data to devices

[7] https://github.com/edgehog-device-manager/edgehog-astarte-interfaces

Features

- Multitenancy
- Device hierarchy based on hardware models and appliance models
- Automatic device classification derived from the declared part number
- Geolocation and reverse geocoding with different possible strategies/providers (WiFi scan, hardware GPS sensor...)
- Telemetry (battery status, OS info, SIM status...)
- Commands (blink an LED to identify the device, control the network...)
- OTA update management

Usecases: Oniro integration

Oniro

- An Eclipse Foundation project
- Open source distributed operating system
    - Targeting IoT devices
    - Defragmenting development for embedded systems
- Same build system (bitbake) for different kernels
- Also see previous talk

Oniro integration

- Astarte Rust SDK can be integrated as a bitbake recipe[8]
- It has been packaged in Oniro using bitbake cargo
- Oniro's next release, goofy, will support it (on yocto Kirkstone branch)
- Feedback welcome [9]

[8] https://github.com/astarte-platform/astarte-device-sdk-rust/issues/20
[9] https://booting.oniroproject.org/distro/oniro/-/issues/191

References

https://astarte.cloud/

https://github.com/astarte-platform/astarte

https://docs.astarte-platform.org/

https://github.com/edgehog-device-manager

https://edgehog-device-manager.github.io/docs/snapshot