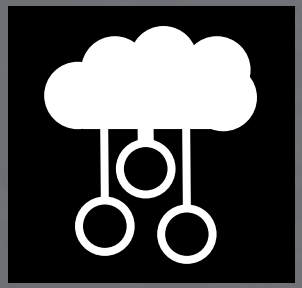
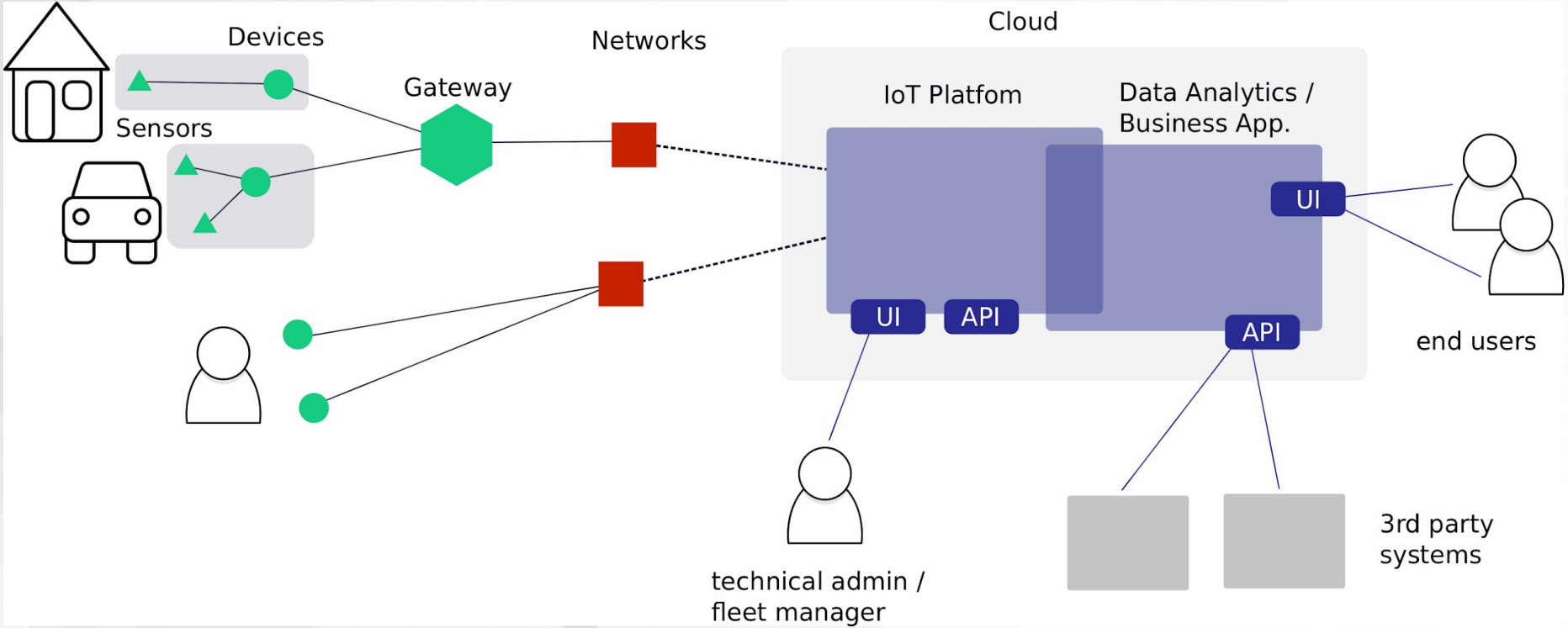


# Internet of Things

# architectures et technologies

janvier 2021 - master *Big Data* - Telecom Paristech

# Chapitre #3
# Plateformes

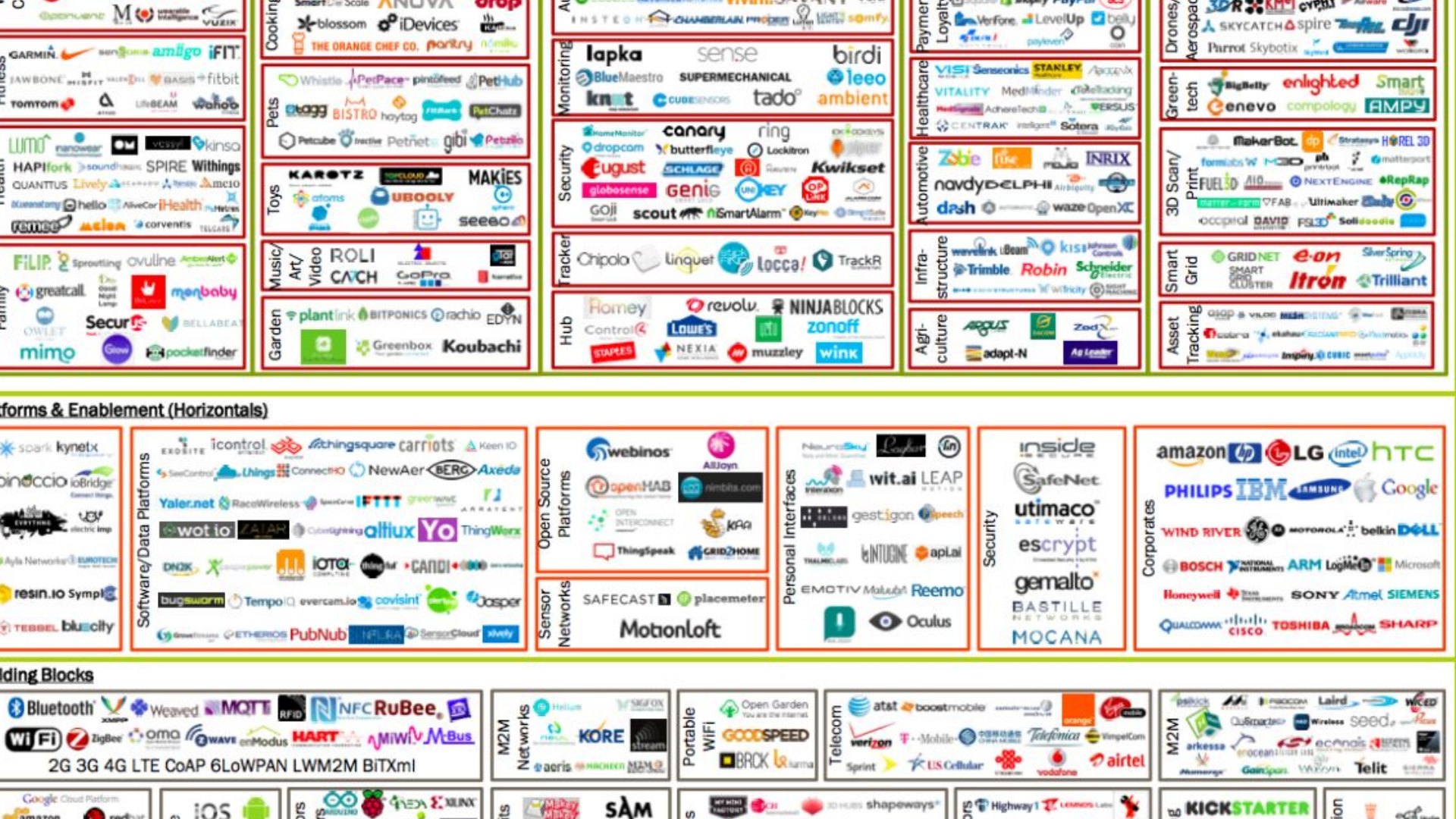

Devices
Networks
Cloud
Gateway
IoT Platfom
Data Analytics /
Business App.
Sensors
UI
UI
API
API
end users
technical admin /
fleet manager
3rd party
systems

# « Plateforme IoT» ?

On entend par « plateforme » ici l'interlocuteur commun aux membres d'une même flotte d'objet collectés.

Rôles:

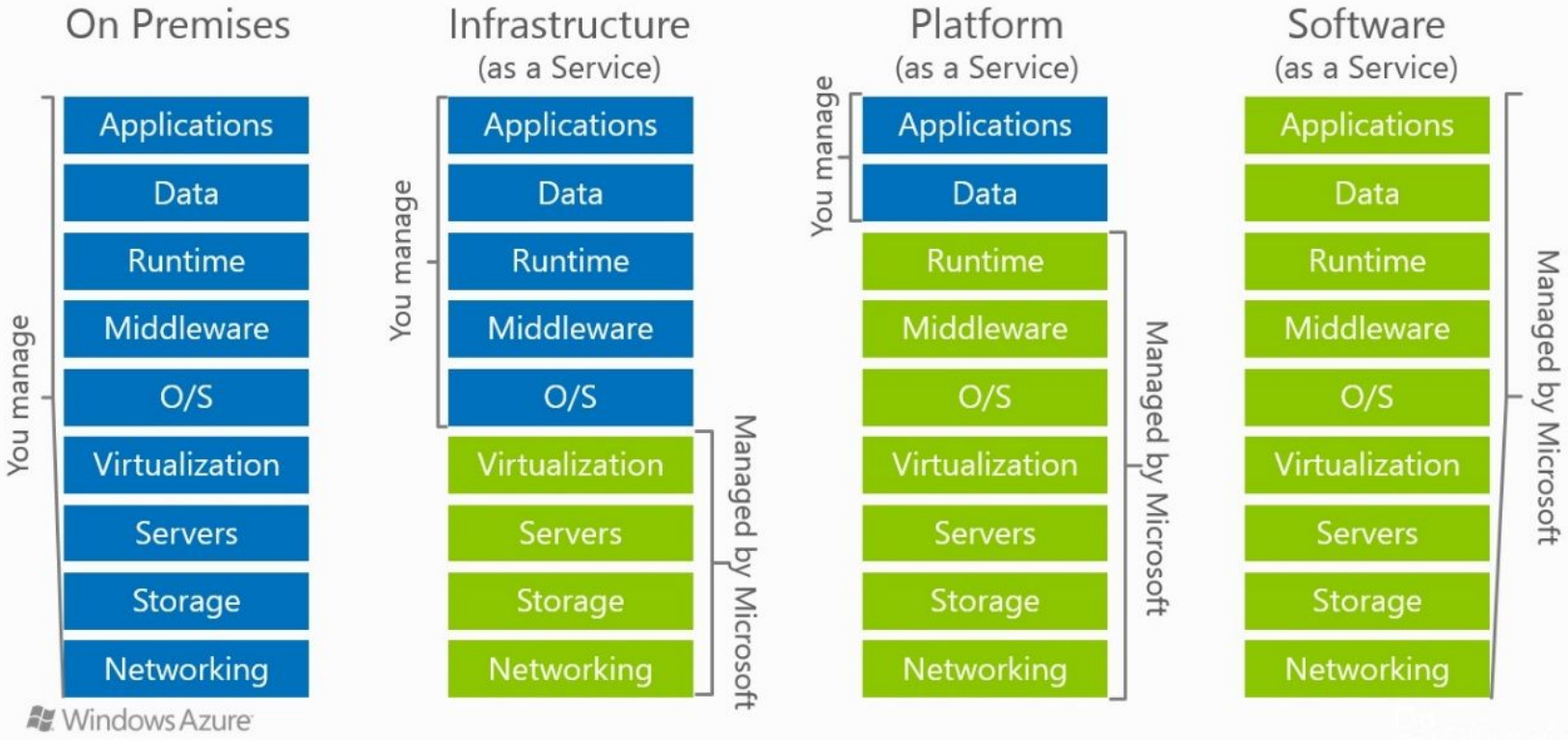
- « Hub » de communication,
- Gestion technique des équipements (« Device Management »)
- Lien vers la partie « Application / Business »
  (ex: la chaîne de traitement Big Data)


Fitness
Health
Family
Cooking
Pets
Toys
Music/ Art/ Video
Garden
Monitoring
Security
Tracker
Hub
Payment/ Loyalty
Healthcare
Automotive
Infra-structure
Agri-culture
Drones/ Aerospace
Green-tech
3D Scan/ Print
Smart Grid
Asset Tracking
Software/Data Platforms
Open Source Platforms
Sensor Networks
Personal Interfaces
Security
Corporates
Bluetooth
Weaved
MQTT
RFID
NFC
RuBee
WiFi
ZigBee
HART
MiWi
MBus
2G 3G 4G LTE CoAP 6LoWPAN LWM2M BiTXml
M2M Networks
Portable WiFi
Telecom
M2M
Google Cloud Platform
iOS
shapeways
KICKSTARTER

# (Quelques) “plateformes IoT” d’intérêt

# Modèle d’hébergement

# Cloud Models




Source: https://stack247.files.wordpress.com/2015/05/azure-on-premises-vs-iaas-vs-paas-vs-saas.png

# Modèles d’hébergement

1. **« On Premises » / « cloud privé »:**

ici l’entreprise gère elle-même ses serveurs et les applicatifs qui y sont déployés,

l’intégrateur a le choix entre le développement de modules applicatifs sur mesure ou la réutilisation de briques commerciales ou open-source.

L’enjeu devient d’identifier les bons middleware (coût, fonctionnel, performances, support) et de correctement les mettre en jeu.

# Modèles d'hébergement

**2. Utilisation d'un IaaS**
**(« Infrastructure as a Service ») / « cloud publique »**

= utilisation de serveurs virtualisés (ou non => on parle alors de serveurs « dédiés ») exploitant une infrastructure mutualisés et déjà opérée,
accès donné à la couche « OS » et gestion du réseau.

Les critères de choix d'un IaaS sont le coût, les SLA (Service Level Agreements: engagements de performance et disponibilité…), la localisation, les fonctions secondaires (ex: backup de stockage, load balancer, outils de protection, outils d'administration et supervision, etc.)

Ex: Amazon WebServices, Azure, OVH

# Modèles d’hébergement

**2.bis Utilisation d’un CaaS (Containers as a Service »)**

ici on déploie / gère non pas des machines mais des “containers”, donc des applicatifs packagés.

On parle de plateforme d’“orchestration”.

Implémentation de référence: Kubernetes, disponible chez de multiples cloud providers (Google Cloud Engine, Microsoft Azure, AWS, etc.)

# Modèles d'hébergement

**3. Utilisation d'un PaaS (« Platform as a Service ») /**

Ici l'approche est semblable à un IaaS sauf que l'accès n'est pas donné à l'OS des serveurs loués: une sélection d'applications / middleware est rendu disponibles à l'instanciation à la demande, et seuls des outils de configuration applicative haut niveau sont fournis.

Les problématique de gestion de l'OS, de configuration réseau etc. sont abstraites.

Ex: AWS / Azure / IBM Bluemix / ThingWorx

# Modèles d’hébergement

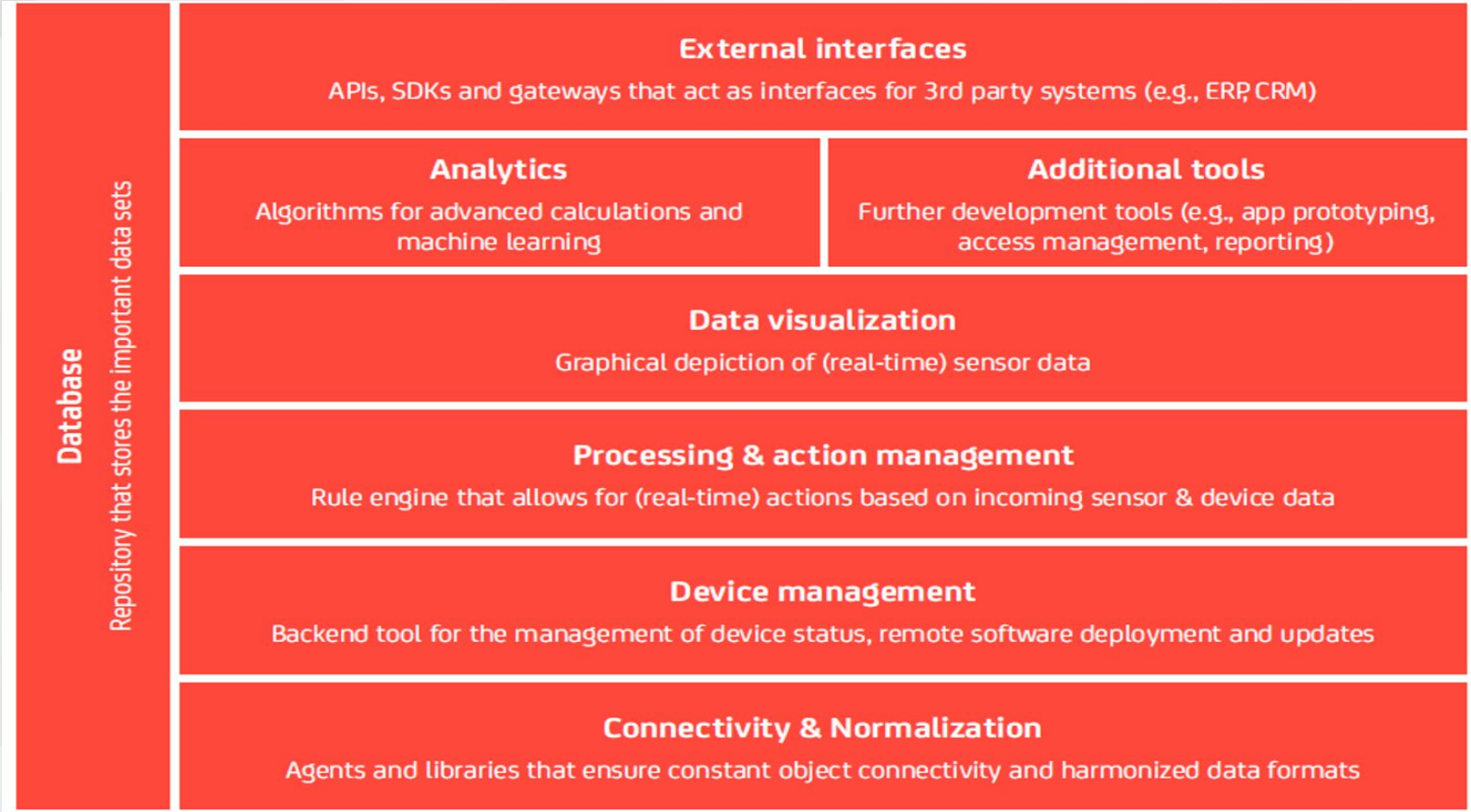
**4. Utilisation d’un SaaS (« Solution as a Service ») /**

Ici l’utilisateur n’a plus à se préoccuper des problématiques d’infrastructure / d’instanciation, rien ne garantit qu’un espace (même virtualisé) lui soit dédié,
Il ne voit que le service final.

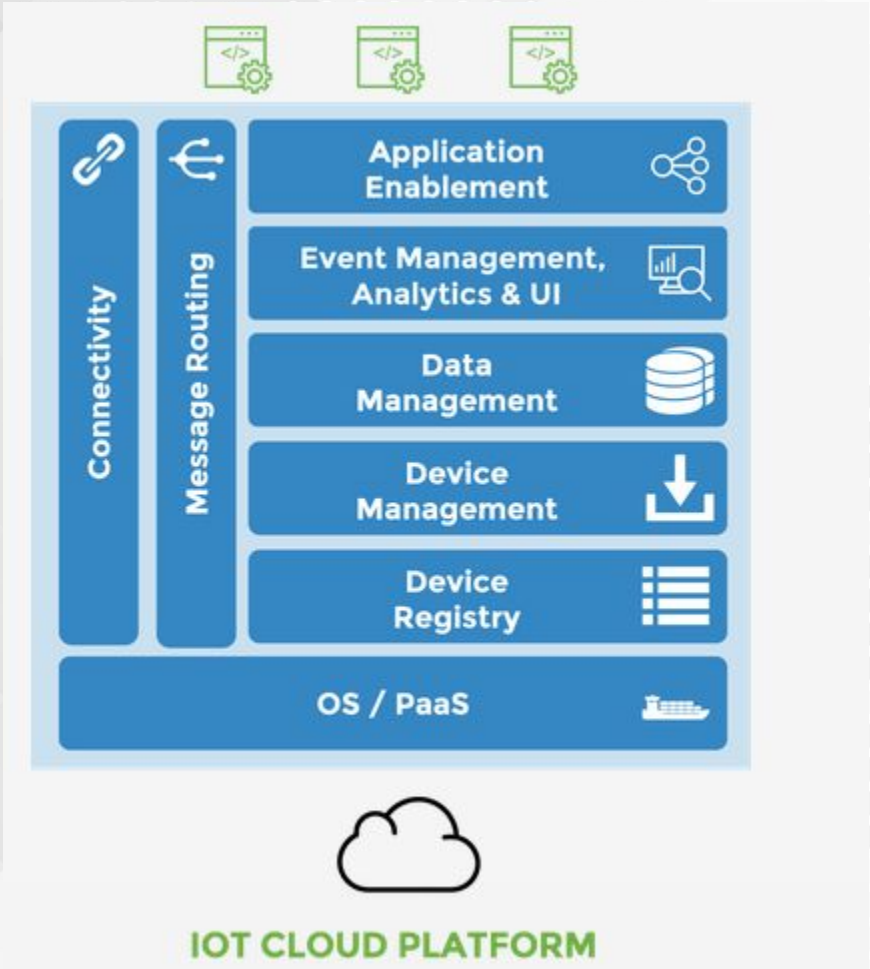
C’est l’approche de la majorité des « services en ligne ».

Ex: Cumulocity, Axeda,

# Fonctions




Source: https://iot-analytics.com/



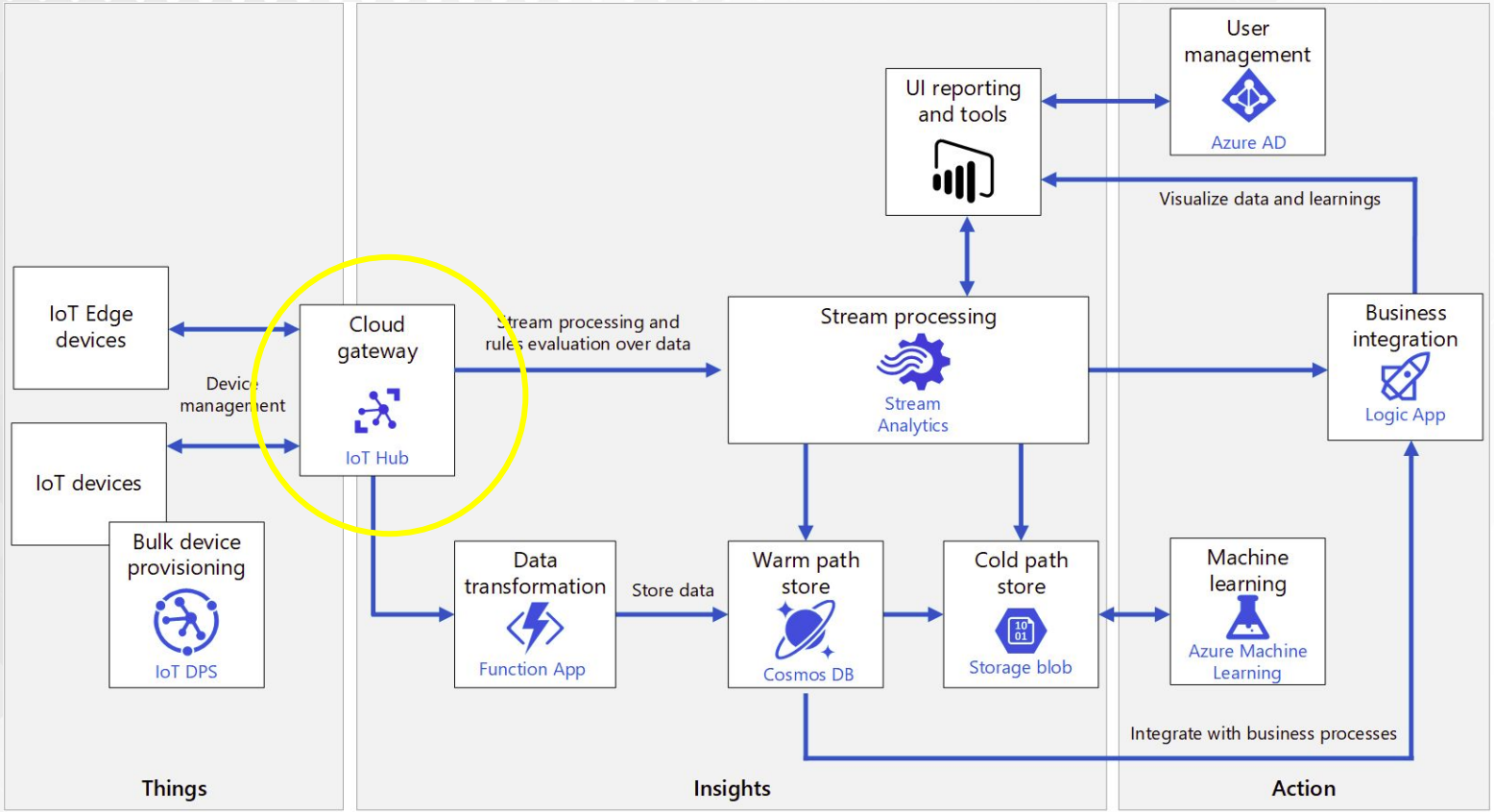

Source: https://iot.eclipse.org/cloud/

# Spécifique IoT: “Device Management”

- Inventaire d’équipements:
  « provisioning », identifiants, authentification (certificats, tokens)
- Supervision de connectivité:
  équipement joignable ou non? Historique de communication…
- Configuration à distance (paramétrage)
- Mise à jour à distance (« firmware » et applications)
- Pilotage à distance (reset, reboot, actions…)
- Diagnostique (journal de logs)

# Exemple: Azure IoT

# Azure IoT - “architecture de référence”



# Azure IoT - “IoT Hub”

**Canal de communication sécurisé** pour envoyer et recevoir des données à partir d'appareils IoT

**Gestion intégrée des appareils** et provisionnement pour connecter et gérer les appareils IoT à grande échelle

**Intégration complète à Event Grid** et calcul sans serveur pour simplifier le développement d'applications IoT

**Compatibilité avec Azure IoT Edge** pour créer des applications IoT hybrides

# Azure IoT - “IoT Hub” (1/4)

## Secure your communications

IoT Hub gives you a secure communication channel for your devices to send data.

- Per-device authentication enables each device to connect securely to IoT Hub and for each device to be managed securely.
- You have complete control over device access and can control connections at the per-device level.
- The IoT Hub Device Provisioning Service automatically provisions devices to the right IoT hub when the device first boots up.
- Multiple authentication types support a variety of device capabilities:
  - SAS token-based authentication to quickly get started with your IoT solution.
  - Individual X.509 certificate authentication for secure, standards-based authentication.
  - X.509 CA authentication for simple, standards-based enrollment.

# Azure IoT - “IoT Hub” (2/4)

## Route device data

Built-in message routing functionality gives you flexibility to set up automatic rules-based message fan-out:

- Use message routing to control where your hub sends device telemetry.
- There is no additional cost to route messages to multiple endpoints.
- No-code routing rules take the place of custom message dispatcher code.

# Azure IoT - “IoT Hub” (3/4)

IoT Hub and the device SDKs support the following protocols for connecting devices:

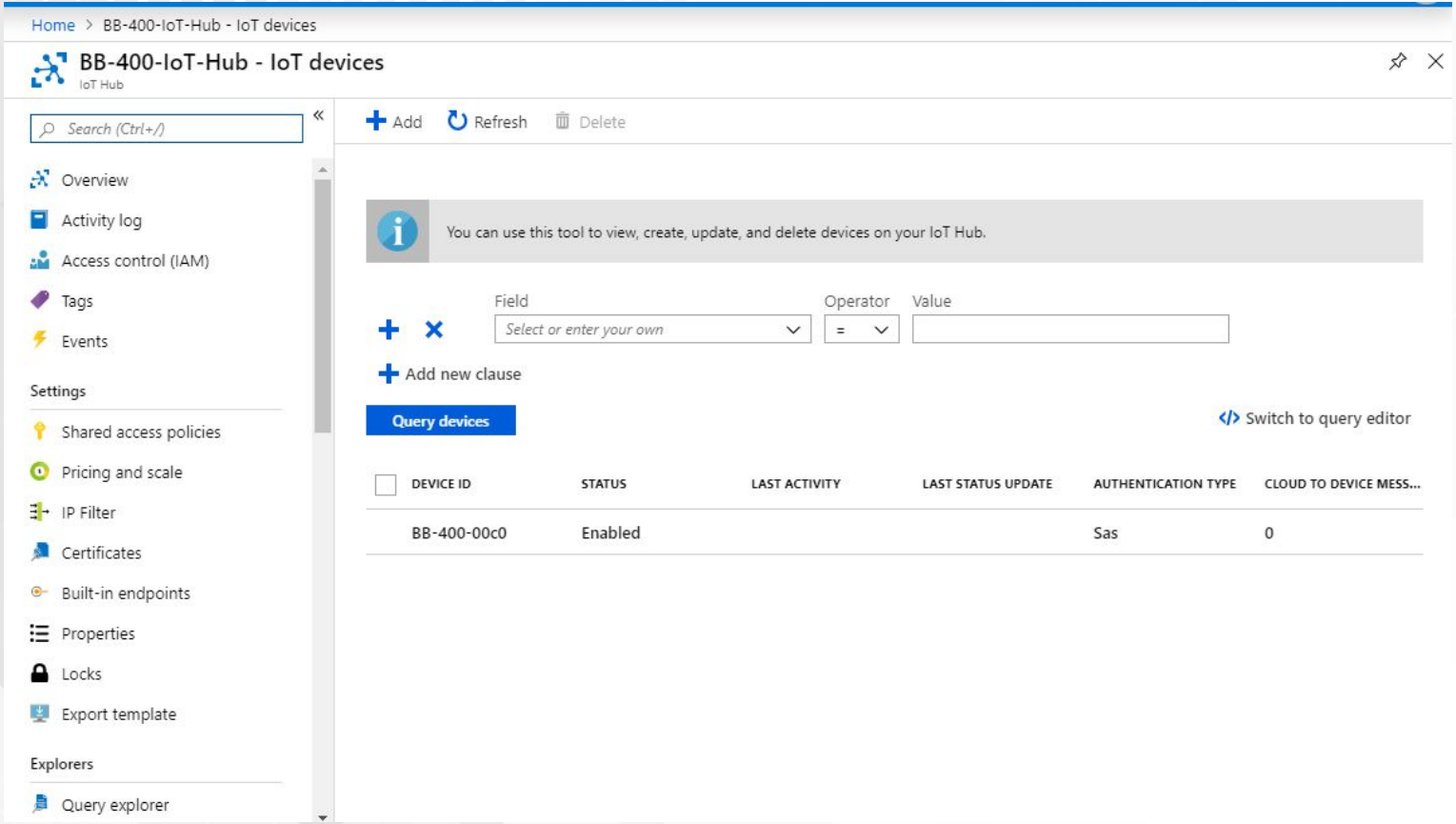
- HTTPS
- AMQP
- AMQP over WebSockets
- MQTT
- MQTT over WebSockets

## Configure and control your devices

You can manage your devices connected to IoT Hub with an array of built-in functionality.

- Store, synchronize, and query device metadata and state information for all your devices.
- Set device state either per-device or based on common characteristics of devices.
- Automatically respond to a device-reported state change with message routing integration.

# Azure IoT - “IoT Hub” (4/4)



# Patterns

# Identification

- Chaque équipement a besoin de disposer dans le système d'un identifiant unique.
- Classiquement, un identifiant "physique" est utilisé:
- Doit pouvoir être lu par le firmware (=> permet d'avoir un firmware unique)
- Doit pouvoir être visible sur l'équipement lorsqu'il est manipulé…
- Exemples:
  IMEI (identifiant modem GSM), Adresse MAC (identifiant interface réseau), IMSI (identifiant carte SIM), numéro de série fabricant (mais alors pas toujours disponible en hw)

# Authentification

- Un secret (login/mdp, PSK, paire clé privée/publique) doit être déployé sur l'équipement.
- Si le secret est partagé entre plusieurs équipement alors la compromission d'un équipement entraîne la compromission de toute la flotte, par ailleurs la porte est surement ouverte à de l'usurpation d'identité.
- Approche industrielle: phase de "bootstrap" distribue un secret individuel.
  Implémenté par exemple en usine, en sortie de chaîne de production sur un "banc de test".
- Autre approche: utilisation d'un secret element (et alors besoin de disposer d'un inventaire croisant identité de SE et identité device)

# Data Stream IoT

- **Ordonnancement**
  du fait des pertes de connexion, l'équipement peut "bufferiser" des données
  => garanties d'ordonnancement des données uniquement par équipement (au mieux!)
  **=> device = clé de "sharding" privilégiée**
  (Ex: clé de partitionnement Kafka)
  **=> à prendre en compte dans algorithmes de traitement**
  (ex: ignorer les données trop désynchronisées, ou corriger à posteriori les stats)

- **Versioning**
  en cas de mises à jour logicielles, l'ensemble de la "flotte" ne peut être à jour au même instant
  **il faut prendre en compte des flux de données hétérogènes (ex: ajout d'un champ, modification d'unité, correctif d'algo embarqué…)**
  **=> important de versionner les données, et conserver une "rétro-compatibilité" des traitements**

# Wake-up SMS

Maintenir une connection au réseau GSM sans "contexte Data" / sans connexion (TCP…) ouverte est beaucoup moins coûteux.

Principe:

- lorsque la plateforme a besoin de joindre un équipement, elle le "réveille" au moyen d'un message SMS (contenant éventuellement un message cryptographique)
- Sur réception, l'équipement authentifie le message et établit une connexion à la plateforme (canal bidirectionnel, ex: MQTT)

# Firmware (1/2)

- Le terme "firmware" désigne le logiciel embarqué remplissant les fonctions de base de l'équipement (gestion alimentation, communications, etc.).
- Un firmware "factory" est installé par défaut en usine.
- Ce firmware peut être bugué, ou suivre des évolutions fonctionnelles, il devient utile de pouvoir le mettre à jour à distance.
- Une mise à jour peut corrompre l'équipement: besoin de conserver plusieurs versions du firmware (ex: "usine" + récente) pour éventuel fallback
  (ex: bouton "factory reset" qui fait booter à nouveau sur le firmware usine)

# Firmware (2/2)

- Enjeux sécurité:

- le firmware peut contenir des secrets => chiffrement;

- le device doit s'assurer de ne pas exécuter de programme malveillant => hash / signature

- Enjeux scalabilité: toute une flotte d'équipements est à mettre à jour dans un temps court

- Besoin de "lisser" la charge => planification / lotissement..

- S'appuyer sur des protocoles / briques applicatives performantes
  (=> "bon vieux serveur FTP/HTTP", bittorrent, ...)

# Bootstrap

- Dans son comportement par défaut l'équipement peut permettre une phase de pré-configuration (secret device, serveur de rattachement, certificat serveur, profile de communication…) avant de basculer en fonctionnement nominal.

- Cela permet la diffusion de configurations individuelles, et de réaliser une "indirection": par exemple diriger une partie de la flotte vers une plateforme X (ex: espace de test, ou dédié à un projet donné) et une autre sur une autre plateforme…

Cf. LWM2M bootstrap

# Local queuing / logging

- Dans les cas d'utilisation où la fraicheur des données n'est pas critique, il peut être intéressant de stocker localement à l'équipement (ou un équipement passerelle) des données en attente de remontée, pour une transmission par lot moins fréquente (et donc moins coûteuse).

- Dans ce cas il convient de mettre en place une notion d' "index" des données (par date pour par id incrémental) pour permettre côté plateforme de rebatir un historique continue des données, ou encore permettre un acquittement

# Garantie de livraison

- Plusieurs profiles de transmission device > plateforme sont possibles, offrant différent niveaux d'effort et de garantie:

| | |
|---|---|
| **"At Most Once" (au maximum une fois)** | Émission "fire and forget"<br>(ex: envoi UDP)<br>=> peut provoquer des pertes |
| **"At Least Once" (au moins une fois)** | Emission puis attente d'un acquittement.<br>Ré-essai périodique jusqu'à succès<br>(avec possible "back-off": alongement des durées entre essais)<br>=> peut provoquer des doublons, parfois possible à traiter en aval. |
| **("Exactly Once" (exactement une fois)** | "Transactionalité"<br>=> aucun doublon. |

# Scalabilité (load balancing)

- Supporter les connexions d'une flotte importante => besoin de répartir la charge sur plusieurs serveurs = "scalabilité horizontale" + load balancing

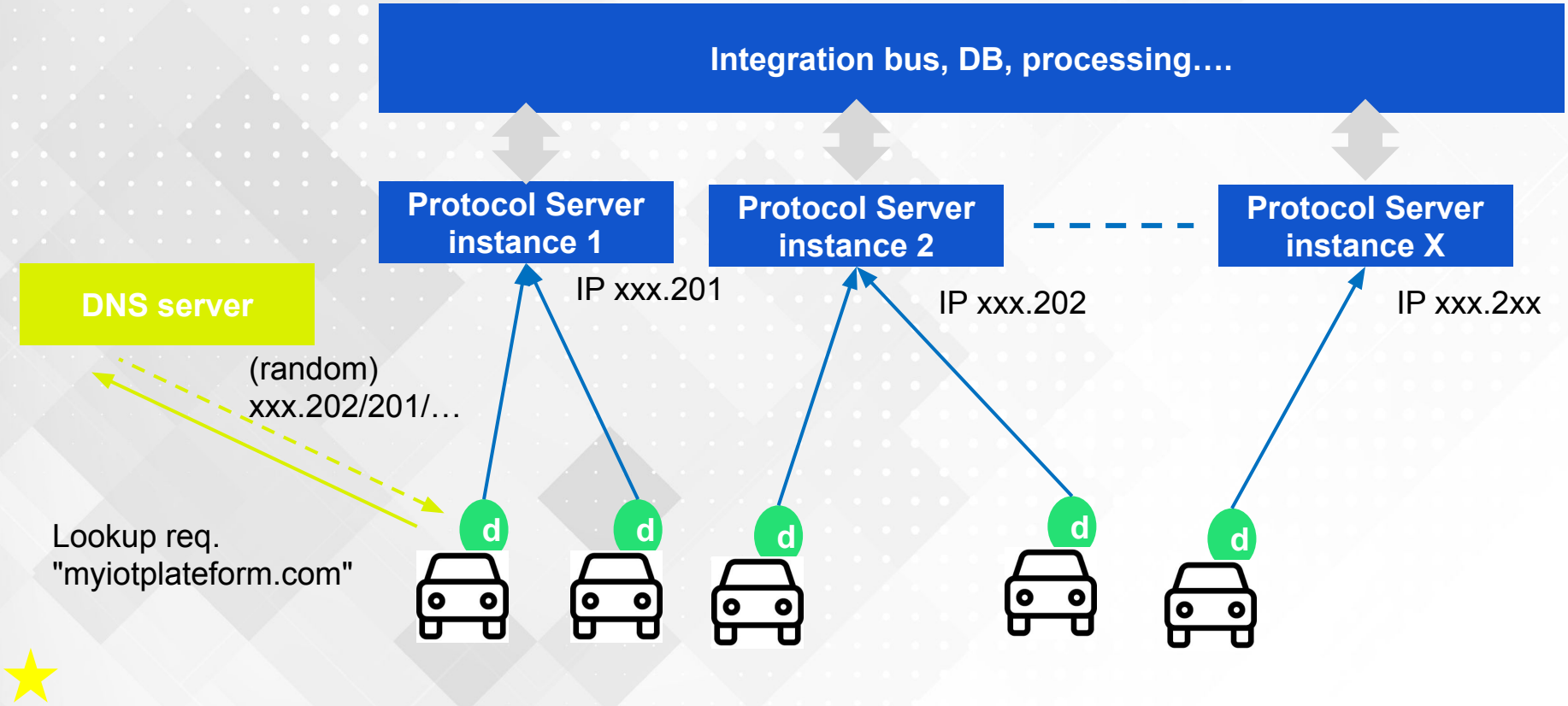
- Techniques de "load balancing":

- via DNS randomisé,

- par profile de configuration device,

- par "load balancer" frontal.

# Scalabilité - DNS load balancing



# Scalabilité – network load balancing