C1: Introduction to IoT

* 3 IoT Elements

<table>
<tr>
<td>Identity</td>
<td>
Every human being living on earth has a unique identity

Unique identification of every device in the IoT system to ensure that each device can be recognized, authenticated and addressed individually

Template: Each [device] have its own MAC address and device ID for clear and unique identification in the network

<table>
<tr><td>Component</td><td>Example</td><td>Explanation</td></tr>
<tr><td>MAC Address</td><td>C4:85:08:AB:67:10</td><td>Unique hardware identifier for smart home light bulb or security camera</td></tr>
<tr><td>RFID / NFC Tag</td><td>RFID tag on a cow</td><td>Used in smart farming to identify and track individual animals</td></tr>
<tr><td>Device ID</td><td>Sensor-XYZ123</td><td>A unique ID assigned to a temperature sensor in a smart building</td></tr>
<tr><td>IMEI</td><td>Smartwatch IMEI number</td><td>Identifies each wearable device on mobile networks</td></tr>
</table>
</td>
</tr>
<tr>
<td>Intelligence</td>
<td>
Each human is born with a brain that can think and process information intelligently

Ability to process data and make decisions for adding value by interpreting data, triggering actions and learning patterns

Template: [Device] is equipped with built-in-logic to [perform smart functions, e.g. process sensor data, make decisions, or
</td>
</tr>
</table>

<table>
<tr><td></td><td>automate response].

<table><tr><td>Component</td><td>Example</td><td>Explanation</td></tr><tr><td>Microcontroller logic</td><td>Smart thermostat</td><td>Adjusts home temperature based on user habits</td></tr><tr><td>AI algorithms</td><td>Surveillance camera with motion detection</td><td>Uses AI to differentiate between humans, pets and objects</td></tr><tr><td>Edge processing</td><td>Soil moisture sensor with threshold logic</td><td>Turns on irrigation only when soil is too dry</td></tr><tr><td>Predictive maintenance</td><td>Factory machine sensor</td><td>Predicts failure based on vibration and usage patterns</td></tr></table></td></tr>
<tr><td>Communication</td><td>People in the world can communicate via social networks and existing Internet infrastructures

Ability to exchange data between devices and systems for enabling connected devices to send and receive data over networks

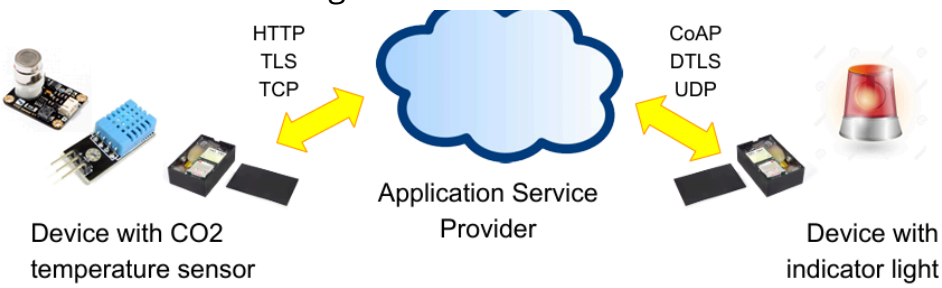
Template: Sends [what] to [cloud service / user's smartphone] via Wi-FI or 4G for [purpose (E.g. easy access / record-keeping)]

<table><tr><td>Technology</td><td>Example</td><td>Explanation</td></tr><tr><td>Wi-Fi</td><td>Smart fridge sending data to mobile app</td><td>Sends energy usage and temperature status</td></tr><tr><td>Bluetooth</td><td>Smartwatch syncing with phone</td><td>Transfers fitness and health data</td></tr></table></td></tr>
</table>

	LoRaWAN	Smart agriculture sensors in fields	Long-range low-power communication for sending weather / soil data
	MQTT Protocol	IoT home automation hub	Lightweight messaging protocol used for device-to-cloud communication
	5G / 4G	Connected car	Transmits GPS location, diagnostic and traffic data in real-time

* IoT Communication Model

Device-to-Device Communication Model	• Represents **two or more devices that directly connect** and communicate between one another, rather than through an intermediary application server
	• These devices **communicate over networks**, including IP networks or the Internet
	• These devices use protocols such as Bluetooth, Z-Wave or ZigBee to establish direct device-to-device communications
	• Examples:
	  ○ Home automation systems (use small data packets of information to communicate between devices with relatively low data rate requirements)
	  ○ Light bulbs (turn on light command)
	  ○ Light switches
	  ○ Thermostats
	  ○ Door locks (door lock status message)

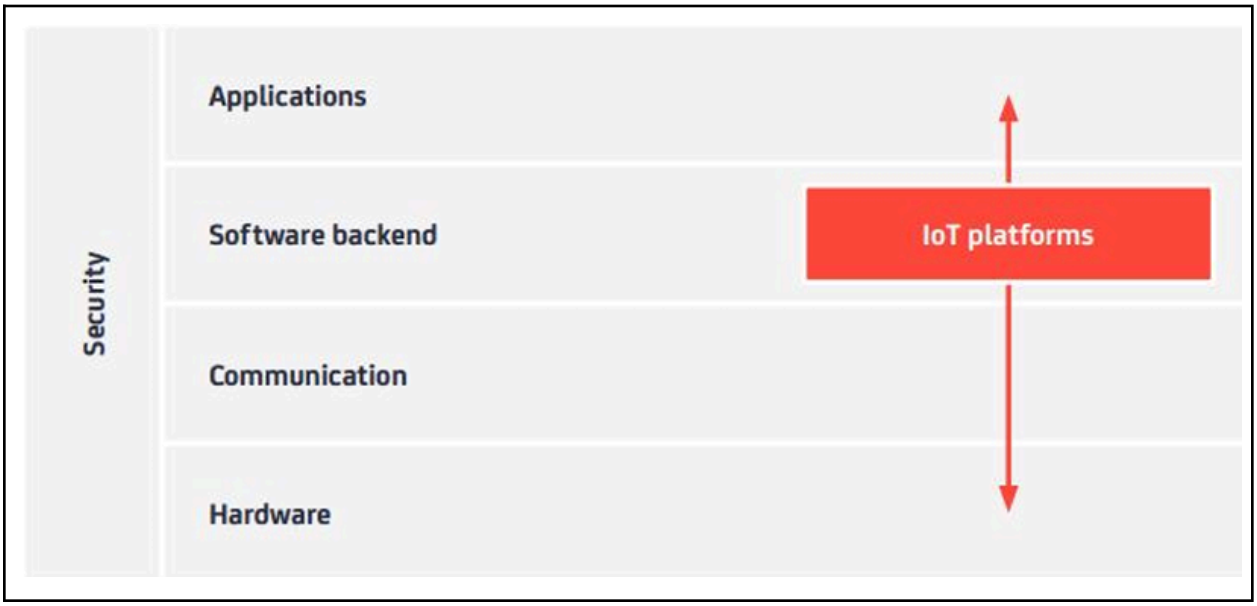
	
Device-to-Cloud Communication Model	• **IoT devices connect directly to an Internet cloud service** like an application service provider to exchange data and control message traffic • Takes advantage of existing communications mechanisms like traditional wired **Ethernet or Wi-Fi connections to establish a connection between device and IP network**, which ultimately connects to the cloud service • Examples: ○ Samsung SmartTV (television uses an Internet connection to transmit user viewing information to Samsung for analysis and to enable the interactive voice recognition features of the TV. 

Device-to-Gateway Communication Model	• a.k.a **device-to-application layer gateway (ALG) model**
	• IoT device connects through an ALG service as a conduit to reach a cloud service
	• Cloud service
	  ○ Application software operating on a local gateway device, which acts as an intermediary between device
	  ○ Provides security and other functionality such as **data or protocol transmission**
	• Examples:

	○ Smartphone can be a local gateway device to run an app to communicate with a device and relay data to a cloud service ○ Personal fitness trackers (this device do not have native ability to connect directly to a cloud service, so they frequently rely on smartphone app to serve as an intermediary gateway to connect the fitness device to the cloud)
Back-End Data Sharing Model	● A communication architecture that enables users to **export and analyze smart object data from a cloud service in combination with data from other sources** ● Suggests a federated cloud services approach or **cloud applications programming interfaces (APIs)** for achieving interoperability of smart device hosted in the cloud ● Allow company to **easily access and analyze the data in the cloud** produced by the whole spectrum of devices in the building ● Facilitates **data portability** needs, allows users to move their data when they switch between IoT servers, breaking down traditional data silo barriers ● Examples: ○ Office (consolidate and analyze energy consumption and utilities data produced by all IoT sensors and Internet-enabled utility systems)

C2: Common Structure of IoT System

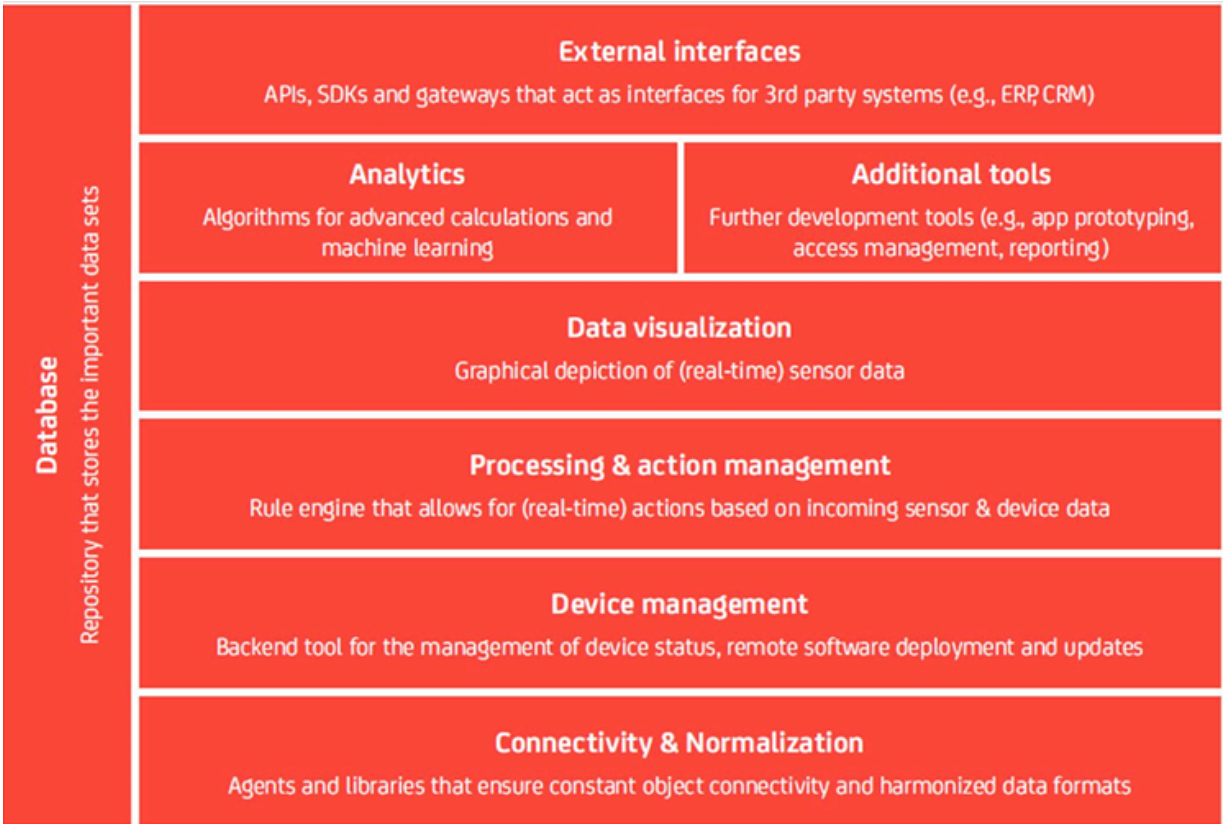
IoT Technology Infrastructure



IoT Platform

- Facilitates **communication, data flow, device management and functionality of applications**
- Links machines, devices, applications and people to data and control centers
- Employs better, quicker search engines and data storage systems with the capacity and sophistication to handle volume far beyond what has brought industry to the present moment
- Most of its elements are **cloud-based and running on wireless connectivity**, which may be established via third-party providers, applications programming interfaces (APIs), cellular capabilities or a combination of these technologies
- A platform enables all these elements to **connect, monitor and communicate** with each other with far greater speed and flexibility
- **Data from an ever-expanding ecosystem can be collected, sorted and harnessed entirely online**
- Platform also enables **data prioritization**, ample **security features**, **scalability** and abundant capacity for **pulling in, storing and analyzing data**

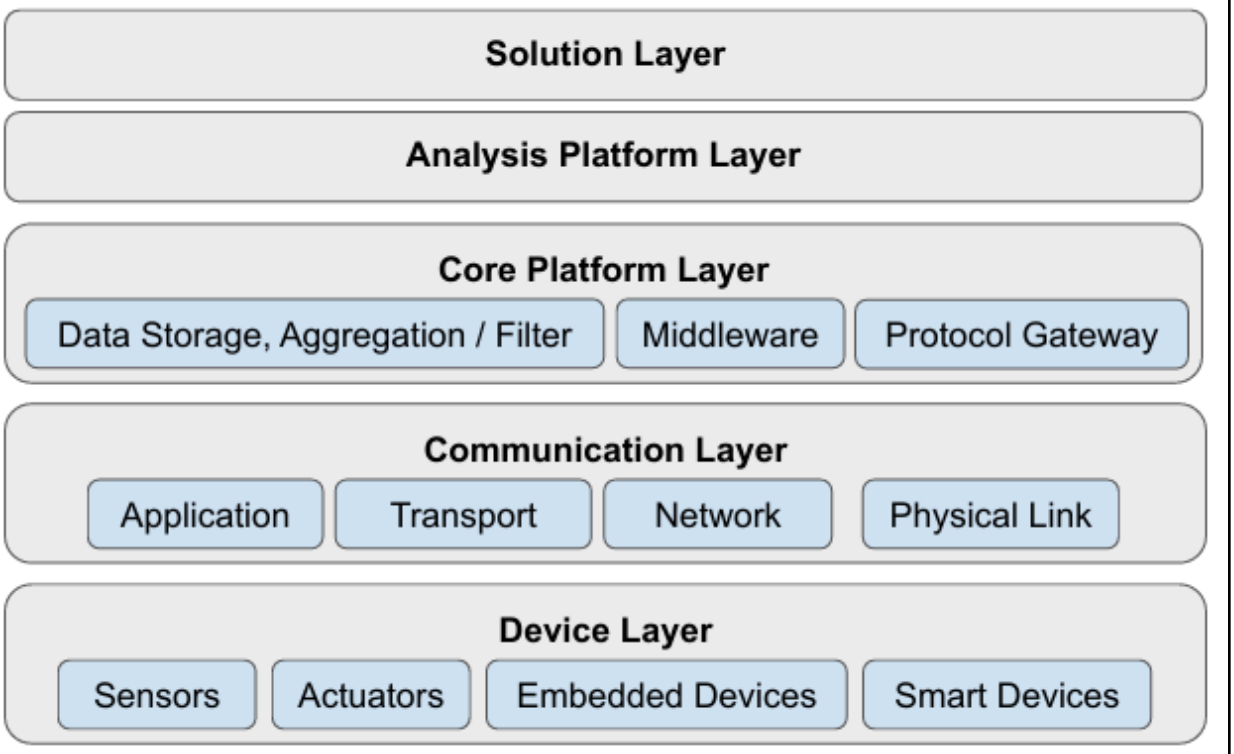
* IoT Platform Architecture



Database	**Scalable storage of device data** brings the requirements for hybrid cloud-based databases to a new level in terms of data volume, variety, velocity and veracity
External Interfaces	**Integrate with third party systems and the rest of the wider IT-ecosystem** via built-in application programming interfaces (API), software development kits (SDK) and gateways
Analytics	**Performs a range of complex analysis from basic data clustering and deep machine learning to predictive analytics** extracting the most value out of the IoT data-stream
Additional tools	**Allows IoT developers prototype, test and market the IoT use case creating platform ecosystem apps** for visualizing, managing and controlling connected devices

Data visualization	**Enables humans to see patterns and observe trends from visualization dashboards** where data is vividly portrayed through line-stacked-, or pie charts, 2D- or even 3D-models
Processing and action management	Brings data to life with **rule-based event-action-triggers** enabling **execution of "smart" actions based on specific sensor data**
Device management	Ensures the **connected "things" are working properly**, **seamlessly running patches** and **updates for software and applications** running on the device or edge gateways
Connectivity and normalization	Brings different **protocols** and different **data formats** into one **"software" interface** ensuring **accurate data streaming** and **interaction with all devices**

* IoT Stack

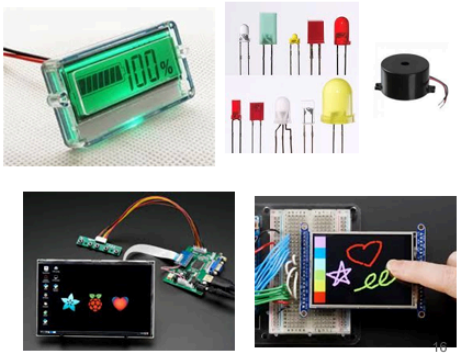


Device Layer	• The bottommost layer lays the end devices, edge devices and gateways

	• Includes apps that can be installed in the end devices to enhance their functionality
	
	<u>Sensors</u>
	• Detect events and read changes in its environment
	• Can be programmed to sense the required environmental parameters that need to be monitored and convert them into meaningful data
	• Connected sensor
	  ○ Capture information from the physical resource within a particular area and send it across over the network
	  ○ e.g. smart water measure quality of water and pollution levels by sensing polluting ingredients in the water
	• Single form sensor
	  ○ Standalone, emits signals based on the target object's movement
	  ○ Makes as an alert beep at regular intervals
	  ○ e.g. medicine bottle with sensors to alerts with a beep to patient for taking medicine on time
	
	<u>Actuators</u>
	• Act with the physical world based on the input received and carry out the required action
	• Examples:
	  ○ DC / AC servo motor, step motor
	  ○ Connected door locks controlled using a remote control device. Actuator being responsible for controlling the movement of the lock motor.

Actuator = to execute

Indicator = to show



Embedded System (Controller)

- Prototype for testing IoT use cases
- Well-known controller boards such as Arduino, Raspberry Pi and BeagleBone
- Connects with different sensors and actuators and provides network ports for internet connectivity

Smart Devices

- Controller can be envisioned to embed in any object and more real-time taking object
- Examples of smart devices:
    - Smart mobile phone
    - Smart tablets
    - Smart watches
- Examples of industry applied in:
    - Health industry (glucometer, heart rate, blood pressure monitors, imaging systems)
    - Automotive industry (tire pressure monitor, connected transportation services)
    - Supply chain industry (RFID tagging for efficient tracking)

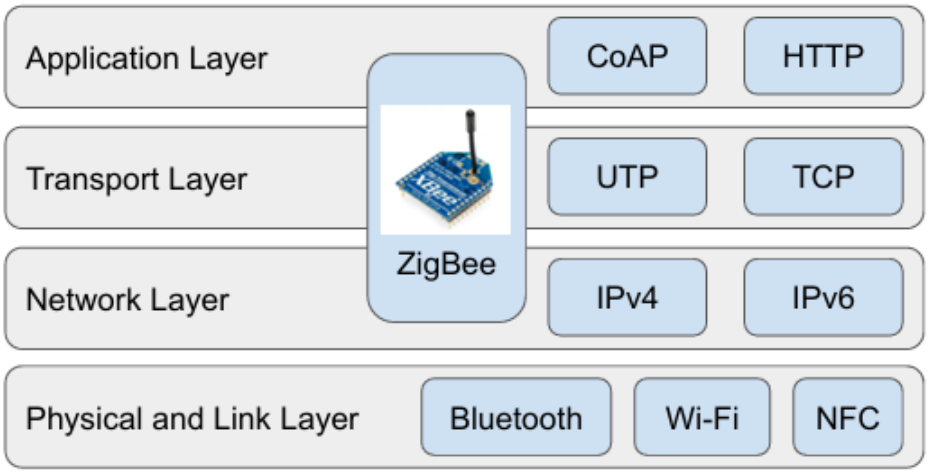
<table>
<tr>
<td></td>
<td>

Energy and utilities industry (smart grids, smart meter)



</td>
</tr>
<tr>
<td>Communication Layer</td>
<td>

Utilizes devices in device layer for communicating among themselves, and between them and the cloud



Communication strategy

Gateway

Device gateway
Smart gateway (Edge gateway)
Smartphone as a gateway

Connectivity

Direct connectivity
Device-to-device connectivity
Application Program Interface (API) connectivity

Communication protocol

Wi-Fi
Ethernet
Cellular

</td>
</tr>
</table>

	○ Bluetooth Low Energy ○ RFID ○ NFC ○ ZigBee / Z-wave ● Application protocol ○ Message Query Telemetry Transport (MQTT) ○ Constrained Application Protocol (CoAP) ○ Advanced Message Queuing Protocol (AMQP) ○ WebSocket ○ AllJoyn ○ Data Connectivity Standard (DDS, Data Distribution Services) ● Industrial protocol ○ BACnet ○ Supervisory Control and Data Acquisition (SCADA) ○ Modbus
Core Platform Layer	<u>Messaging Middleware</u> ● Acts as the **communication backbone between distributed enterprise systems**. ● Ensures **secure, scalable, high-performance messaging** for many connected devices. ● Handles **device registration, data storage, and controlled access**. ● Temporarily **stores service data** before passing it to dedicated storage for **analysis**. <u>Data Storage</u> ● **Stores large volumes (terabytes) of continuous data** from devices. ● Uses scalable NoSQL databases for **fast data retrieval and processing**. ● **Supports structured, semi-structured, and unstructured data** to enable deep analytics. <u>Data Aggregation and Filter</u> ● **Filters and processes raw data** to reduce unnecessary load. ● Since edge devices have limited power, filtering is

	handled centrally based on data dependency. ● **Adds context to data to make it meaningful and useful** for downstream applications.
Analytic Platform Layer	● **Flow and Task placement** ○ **Chooses the best cloud/fog resources** to run incoming tasks based on current availability. ○ Works with Resource Provisioning to **balance workload and trigger more resource allocation** if needed. ● **Knowledge Base** ○ **Stores past data about app and resource usage.** ○ Helps other services **make smarter decisions** using historical trends. ● **Raw Data Management** ○ **Connects directly to data sources.** ○ **Creates simple or complex data views** for other services, hiding the complexity of how views are generated. ● **Performance Prediction (Machine Learning)** ○ **Uses historical data to predict how cloud resources will perform**. ○ Helps Resource Provisioning **decide how many resources are needed** during high load or poor performance. ● **Monitoring & Profiling (Actionable Insights)** ○ **Continuously monitors system and app performance**. ○ **Creates detailed profiles of resources and applications** for better insight and optimization. ● **Resource Provisioning** ○ **Dynamically allocates cloud, fog, and network resources based on app needs and system conditions**. ○ Uses input from other services and user latency/security requirements. ● **Security** ○ **Handles authentication, authorization, and**

<table>
<tr><td></td><td>
encryption.
○ Ensures secure access to services and resources.

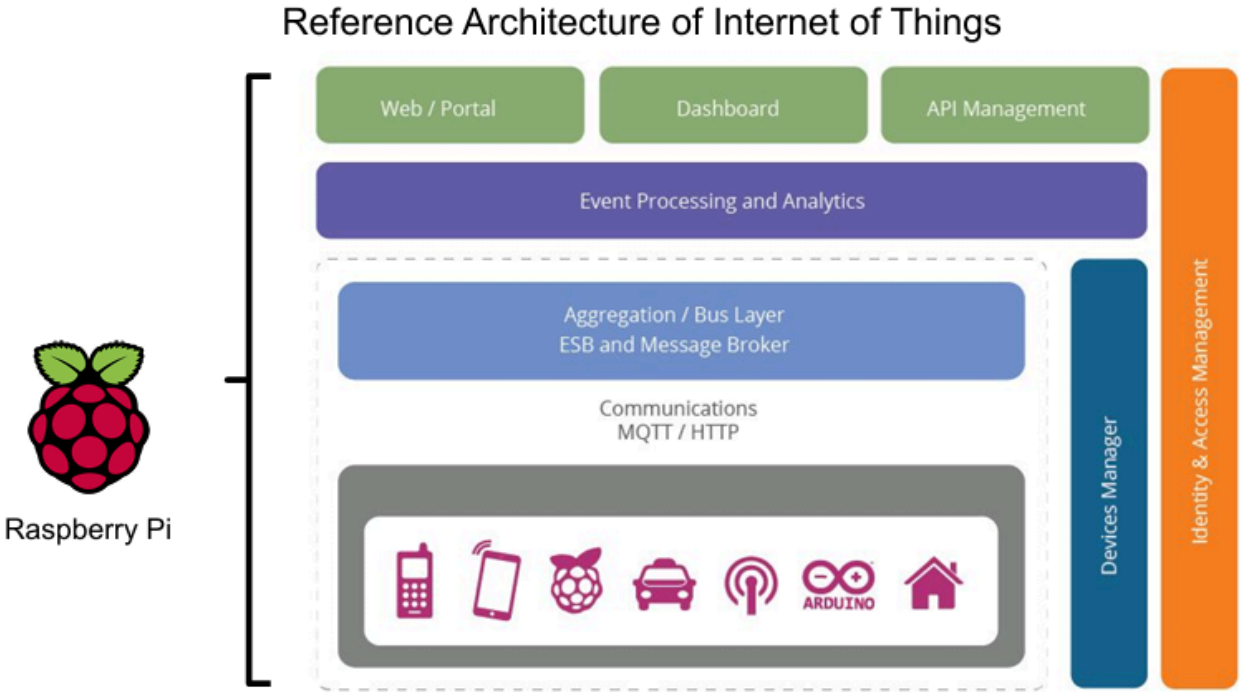
<table>
<tr><th>Component</th><th>Function</th><th>Key Role in IoT System</th></tr>
<tr><td>Flow and Task Placement</td><td>Tracks system state to assign tasks to cloud/fog/edge resources</td><td>Optimizes performance by choosing the best execution location</td></tr>
<tr><td>Knowledge Base</td><td>Stores historical data on app and resource usage</td><td>Supports informed decision-making across services</td></tr>
<tr><td>Raw Data Management</td><td>Interfaces with data sources and prepares data views</td><td>Supplies data (raw or processed) to analytics and other services</td></tr>
<tr><td>Performance Prediction (ML)</td><td>Uses historical data to predict how resources will perform</td><td>Helps in proactive scaling and efficient resource planning</td></tr>
<tr><td>Monitoring & Profiling</td><td>Monitors current system performance and application behavior</td><td>Builds profiles and provides real-time insights for other services</td></tr>
<tr><td>Resource Provisioning</td><td>Dynamically allocates computing and network resources</td><td>Ensures sufficient resources are available to meet app requirements</td></tr>
<tr><td>Security</td><td>Manages authentication, authorization, and cryptography</td><td>Ensures secure communication and access control across the platform</td></tr>
</table>
</td></tr>
<tr><td>Solution Layer</td><td>
● Top most layer contains the applications that leverage fog computing to deliver innovative and intelligent application to end users
● Divided into 2 parts:
○ Solution Template
■ Includes Device Model, SDKs, Application Model, Composite IoT Services, Machine Learning Models and Event Services
○ IoT Application
■ Customized application that requires common set of tasks and services, configurations of components via dashboard or APIs and the entire process can be simplified and abstracted through the use of solution templates
</td></tr>
<tr><td colspan="2"><u>Summary</u></td></tr>
</table>

Layer	Description	Main Responsibilities
1. Device Layer	The **physical layer** where sensors, actuators, and IoT devices exist.	- Collect data from the environment - Perform actions - Initial data generation
2. Communication Layer	Handles **data transmission** between devices, gateways, and platforms.	- Enable network connectivity (Wi-Fi, Bluetooth, 5G, etc.) - Ensure secure, reliable data transfer
3. Core Platform Layer	The **central control system** that manages devices, data flow, and basic processing.	- Device management - Data routing - Integration with services
4. Analytic Platform Layer	Responsible for **data analysis, intelligence, and decision-making** using AI/ML and other tools.	- Data processing and insights - Task distribution - Performance prediction
5. Solution Layer	Provides **end-user applications** and interfaces that use the processed data to deliver value.	- Visualization dashboards - Mobile/web apps - Automation and smart actions

C3: Physical Component of IoT System

Raspberry Pi

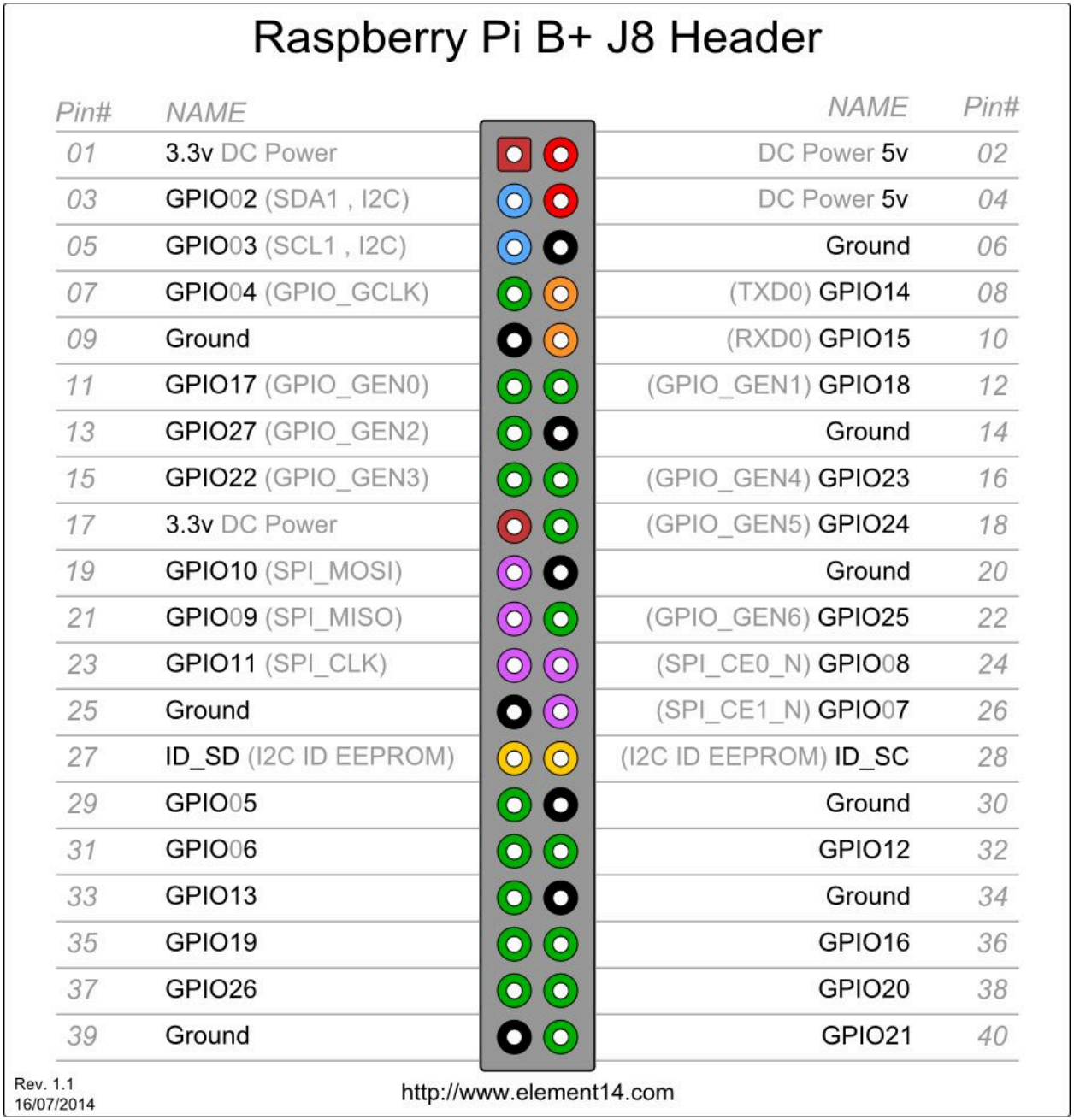
- A series of small single-board computers developed in United Kingdom by Raspberry Pi Foundation in year 2011
- Promote teaching of basic computer science in schools and in developing countries
- Provides low-cost, high-performance computers that people use to learn, solve problems and have fun
- Develop free resources to help people learn about computing and how to make things with computers, and train educators who can guide other people to learn



Raspberry Pi series

Source: http://socialcompare.com/en/comparison/raspberrypi-models-compari

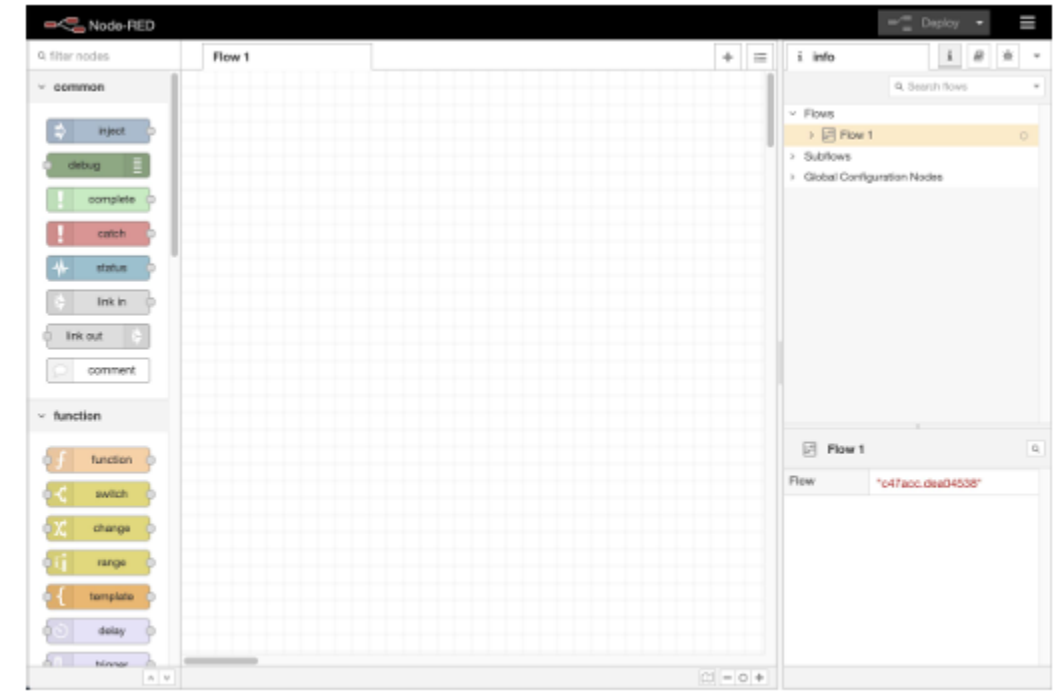
	Raspberry Pi 4 Model B	Raspberry Pi 3 Model B+	Raspberry Pi 2 Model B v1.2	Raspberry Pi Model B+	Raspberry Pi Model A+
Processor Chipset	Broadcom 2711, Cortex-A73, 64-bits SoC	Broadcom BCM2837B0, Cortex-A53, 64-bits SoC	Broadcom BCM2837 64Bit SoC	Broadcom BCM2835 32Bit SoC full HD	Broadcom BCM2835 32Bit SoC full HD
Processor Speed	Quad-core @ 1.5GHz	Quad-core @1.2 GHz	Quad-core @900 MHz	Single Core @700 MHz	Single Core @700 MHz
RAM	1GB, 2GB, 4GB LPDDR4 SDRAM	1GB SDRAM @ 400 MHz	1GB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz	256 MB SDRAM @ 400 MHz
Storage	MicroSD	MicroSD	MicroSD	MicroSD	MicroSD
USB 2.0	2 USB 2.0, 2 USB 3.0	4x USB Ports	4x USB Ports	4x USB Ports	1x USB Port
Max Power Draw/voltage	3A @ 5V	2.5A @ 5V	1.8A @ 5V	1.8A @ 5V	1.8A @ 5V
GPIO	40 pin	40 pin	40 pin	40 pin	40 pin
Ethernet Port	Yes	Yes	Yes	Yes	No
WiFi	Built in	Built in	No	No	No
Bluetooth LE	Built in	Built in	No	No	No



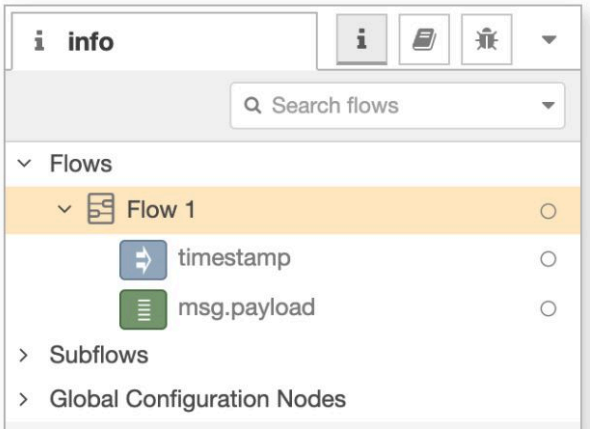
Node-RED

- A flow-based programming tool
- Consists of a Node.js based runtime that we point a web browser at to access the flow editor
- Within the browser we can create our application by dragging nodes from our palette into a workspace and start to wire them together

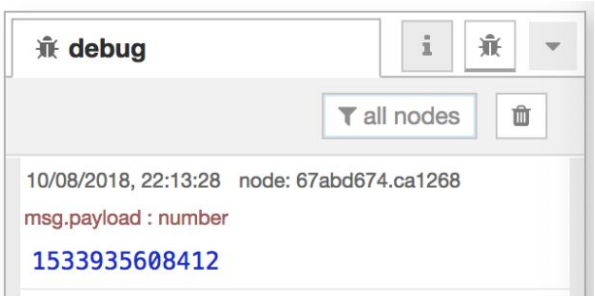
- The palette of nodes can be easily extended by installing new nodes created by the community and the flows that we created can be easily shared as JSON files



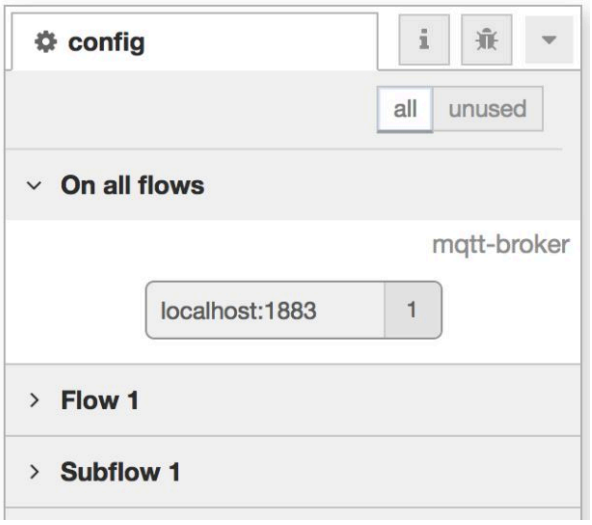
Palette	• Contains all the **nodes** that are installed and available to use • Organized into a number of categories, with **inputs, outputs and functions** at the top • If there are any subflows, they appear in a category at the top of the palette
Workspace	• The place where flows are developed by **dragging nodes from the palette and wiring them together** • Has row of tabs along the top • One for each **flow** and any **subflows** that have been opened
Sidebar	• Contains panels that provide a number of useful tools within the editor • **Information** ○ View information about nodes and their help



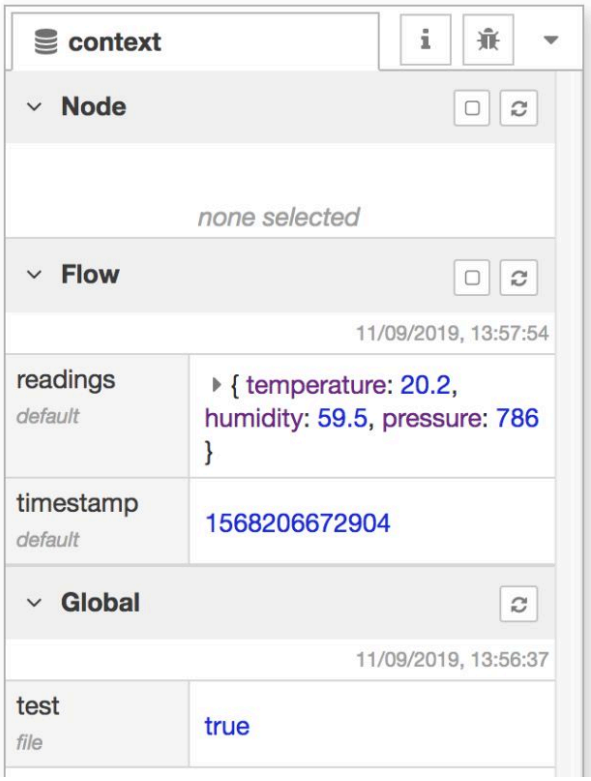
- **Debug**
  - View messages passed to Debug nodes



- **Configuration Nodes**
  - Manage configuration nodes



- **Context Data**
  - View the contents of context


context
Node
none selected
Flow
11/09/2019, 13:57:54
readings
default
{ temperature: 20.2, humidity: 59.5, pressure: 786 }
timestamp
default
1568206672904
Global
11/09/2019, 13:56:37
test
file
true

C9: IoT Adoption - Business Values and Impact

Benefits / Potential Value of IoT

Customer value of IoT	• Enables new sources of value • Most of the value lies in powerful analytics and cross-ecosystem collaboration • e.g. New product capabilities, list of customers' benefits
Competitive advantage	• Unique data sets, superior algorithms and exclusive partnerships serve as the major sources of competitive advantage • If they are protected, it can lead to long-standing advantage over the competition • e.g. New resources and activities

Customer Value of IoT

- The real value of Internet of Things lies in the data that is generated, the analytics that can be performed with the data, and the connection with the whole ecosystem (e.g. other components of the home) or even adjacent ecosystems (e.g. Smart Grids)

New Product Capabilities

Connected car Digital / Driver assisted car	• Digital experiences such as navigation systems or digital information on dashboards were added to enable a new driver-assisted experience • Connected car → Managed car → Automated car → Self-driving car → Car ecosystem → Connected ecosystem
Change of product life cycles	• It will be possible to update product functionalities of physical devices through over-the-air product updates in future • e.g. People download new apps on their smartphone to receive new functionalities

* List of Customers' Benefits

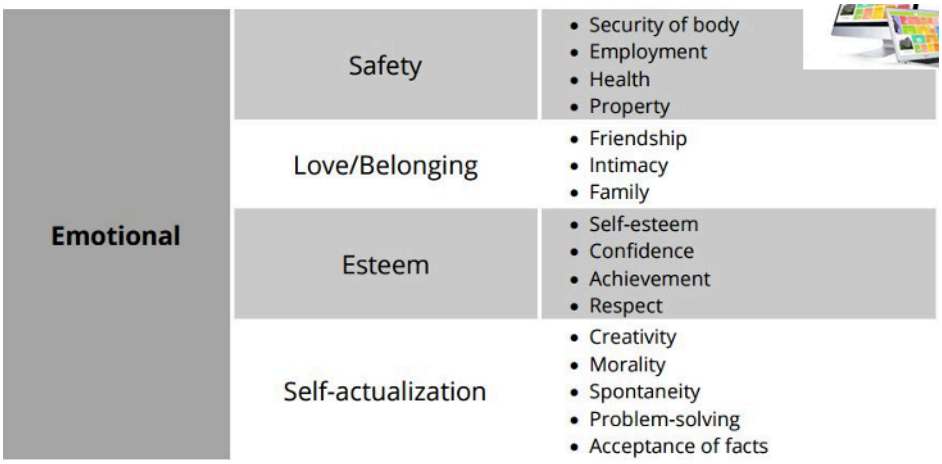
<table>
<tr>
<td>Functional Product Benefits</td>
<td>
<table>
<tr><th>Benefit type</th><th>Category</th><th>Customer value</th></tr>
<tr><td rowspan="3">Product/ Machine</td><td>Daily operations</td><td>• Monitoring
• Localization
• Control
• Safety</td></tr>
<tr><td>Optimization</td><td>• Functionality
• Convenience
• Cost</td></tr>
<tr><td>Maintenance/Service</td><td>• Availability
• Cost</td></tr>
</table>
</td>
</tr>
<tr>
<td>* Functional Process Benefits</td>
<td>
<table>
<tr><td rowspan="3">Process</td><td>Strategic information</td><td>• Choice of machines
• Sales optimization</td></tr>
<tr><td>Daily operations</td><td>• Monitoring
• Control
• Billing
• Documentation
• Contract management
• Usability</td></tr>
<tr><td>Optimization</td><td>• Availability
• Quality
• Automation
• Costs</td></tr>
</table>

<u>Strategic information</u>
● Choice of machines
○ e.g. Analyze historical performance data to choose vending machines with better uptime and lower maintenance costs.
● Sales optimization
○ e.g. Use customer purchase trends and location analytics to decide which products to stock or remove.

<u>Daily operations</u>
● Monitoring
○ e.g. Real-time dashboard showing device status, temperature, and stock levels for all vending machines.
● Control
○ e.g. Remote configuration of pricing, content updates, or shutdown of malfunctioning devices.
● Billing
○ e.g. Automated billing based on usage, payment
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
tracking, and generation of financial reports.
● Documentation
○ e.g. Auto-generation of service logs, maintenance records, and usage summaries.
● Contract Management
○ e.g. Digital management of service-level agreements (SLAs), maintenance contracts, and vendor terms.
● Usability
○ e.g. User-friendly interface for operators to manage multiple devices with minimal training.

<u>Optimization</u>
● Availability
○ e.g. Predictive maintenance to reduce downtime and increase machine availability.
● Quality
○ e.g. Monitor environmental conditions (temperature, humidity) to ensure product quality in vending machines.
● Automation
○ e.g. Auto-replenishment orders triggered by stock level thresholds; dynamic pricing based on demand.
● Costs
○ e.g. Optimize delivery routes to save fuel; reduce energy consumption through smart scheduling of machine operations.
</td>
</tr>
<tr>
<td>Functional Business Benefits</td>
<td>
<table>
<tr><td rowspan="4">Business</td><td>Customers</td><td>• Customer retention
• Customer satisfaction
• Lead generation
• Offer optimization
• Sales proposition</td></tr>
<tr><td>Products</td><td>• Requirements management</td></tr>
<tr><td>New markets</td><td>• Data as a product
• IoT as catalyst</td></tr>
<tr><td>New business models</td><td>• Added-value offering / new services
• Pay-per-use / Product-as-a-service
• Contracting</td></tr>
</table>
</td>
</tr>
</table>

Emotional Benefits	

Competitive Advantage

- Selling products that deliver customer value does not enable competitive advantage by itself
- Competitive advantage comes once a company serves this value better than its competition
- Key to sustaining competitive advantage: **unique resources** and **activities**

* Resources and Activities

Unique Data Sets	• Get a hold of large IoT data sets that can build up a position not reachable by competitors • e.g. Google and Facebook
Analytics	• Superior algorithms to automate and optimize become a key source of competitive advantage • e.g. Google had developed superior algorithms to index the world wide web
Exclusive co-operations and contracts	• Possible to strategically lock-out competitors by securing exclusive partnerships with other companies

Talent Access	• Mastering IoT requires different skill sets:
	  ○ Application development
	  ○ Enterprise architecture
	  ○ Hardware engineering
	  ○ Data science

	○ Cybersecurity expertise ● e.g. Creating strong demand of these job profiles
IoT-device / Product Development	● Toyota's engineers mastered the new vehicle development process such that Toyota was able to perform this process twice as fast as some competitors (and also at higher quality)
Algorithm Development	● Innovate and develop these new algorithms that are ahead of the competition

Impacts of IoT on Business and Society

Bargaining power of buyers	● **Pressure that customers / consumers can put on business** to get them to **provide higher quality products, better customer service and / or lower prices** ● Allow companies to **tailor their products more to customers** and generate new offerings
Bargaining power of suppliers	● **Pressure that suppliers can put on companies by raising their prices, lowering their quality or reducing the availability of their products** ● As more and more products become network-enabled and data becomes a key differentiator, the sources of what make a product valuable will shift ● Companies that supply critical components such as interactive screens today, may find that tomorrow their offering becomes commoditized ● Meanwhile, new powerful suppliers emerge to change the supplier power thus have to be evaluated on a case-by-case basis
Threat of substitutes	● Occurs when **companies within one industry are forced to compete with industries producing substitute products or services** ● A rise in product attributes therefore leaves more room for potential substitutes

	• Threat of substitutes may increase with the Internet of Things
	• Examples:
	  ○ **Software-as-a-Service**
	  ■ MS Windows is nowadays sold on a subscription rather than a one-off - with multiple advantages to both customers and suppliers
	  ○ **Products-as-a-Service**
	  ■ Car sharing company are already employing this model as cars in cities are paid on a per-use basis
	  ○ Cloud Computing

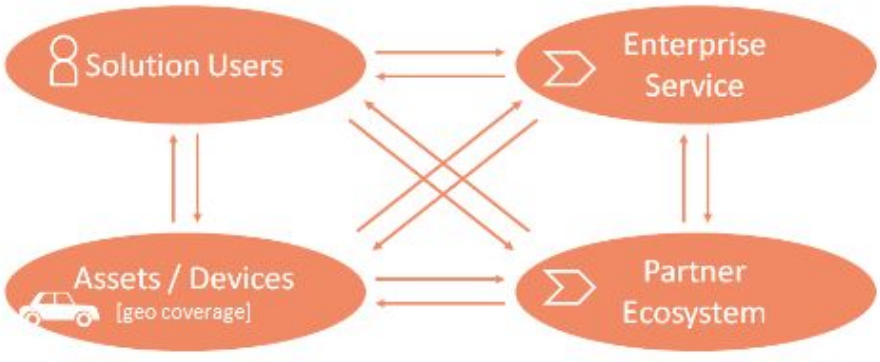
Threat of new entrants	• **Threat that new competitors pose to current players within an industry**
	• Key value drivers: Data from IoT turns into Data Analytic Application
	• Successful IoT companies will have access to data and will have to develop (or source) algorithms that make sense of the data and utilize it with a benefit to the customer
	• Companies that are trying to enter an industry but they will **not have this data or these algorithms when they start**
	• New entry barriers to new entrants:
	  ○ Upfront costs
	  ○ Brand loyalty
	  ○ Supplier and distribution channels
	  ○ Government regulations
	• **Developing IoT-enabled products costs both money to develop and requires the acquisition of new talent** (hardware designers, data scientists, etc)
	• The **threat of new entrants for successful IoT companies** in an industry may thus **go down**

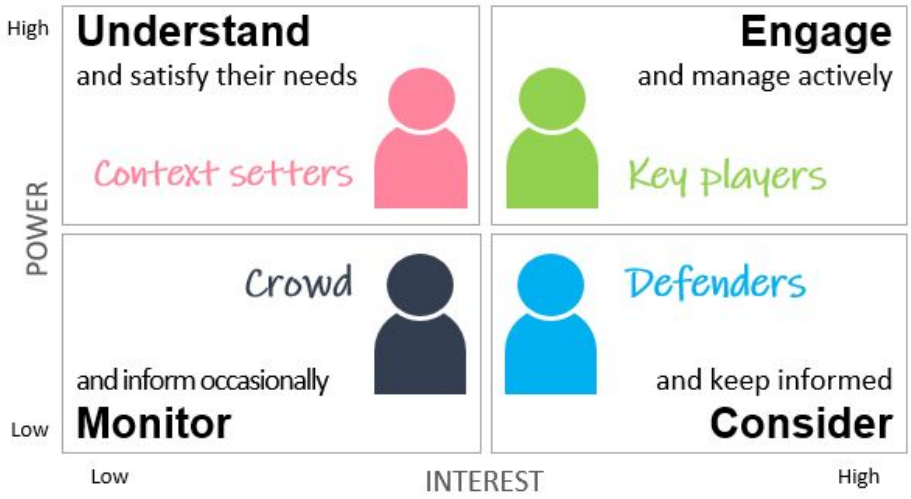
Rivalry among existing competitors	• **Rivalry tends to increase in intensity when companies either feel competitive pressure or see an opportunity to improve their position**

	• As the Internet of Things brings together traditional companies from the industry, IT-companies, hardware-companies, and others the rivalry for market share in most industries is set to increase • Examples: ○ In the auto-industry, for years, companies like Toyota, Volkswagen or GM fought the rivalry battle with Tesla being the only new entrant. But now it is Google and Apple announcing that they are entering the car market and potentially others following soon

C11: IoT Adoption - Industrial Internet Reference Architecture (IIRA) View Points

Business View Point

Problem Statement (Stakeholder Analysis)	• Most IoT solutions are made up of 4 central elements, each with associated stakeholders:
	  ○ Assets and corresponding devices
	  ○ Related enterprise services
	  ○ Solution users
	  ○ Partners
	 
	
	Source: www.enterprise-iot.org
	Four key elements of an IoT solution
	• We can consider to adapt the power-interest matrix by Mendelow (1991) to extend our analysis and

	understand the needs of each stakeholder 

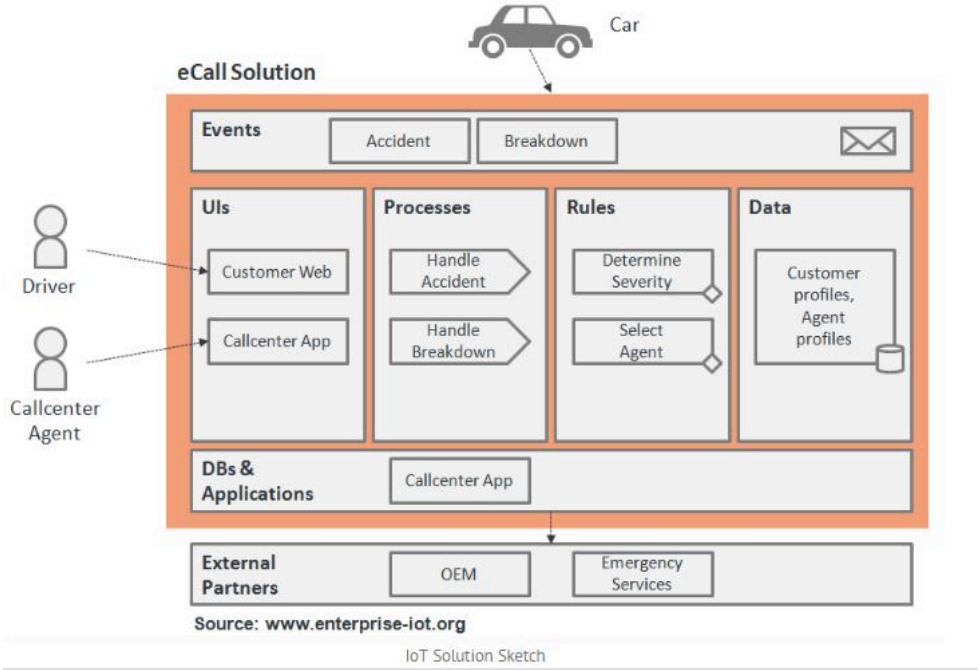
Site Survey	• Physical visit to sites
	• Examples:
	  ○ Factory where the assets are manufactured
	  ○ A site where IoT devices and equipments are used
	• Site survey document should cover all asset- and site-specific aspects:
	  ○ **Type of asset**
	  ■ e.g. stationary or moving
	  ○ **Behavior of asset**
	  ■ dumb asset (e.g. a beer keg or pallet)
	  ■ intelligent asset (e.g. vending machine, PLC, vehicle)
	  ○ **Business operations** and asset **lifecycle**
	  ○ **Solution integration**
	  ■ native or retrofit
	  ○ **Asset interfaces**
	  ■ e.g. ModBus, CAN bus, MDB, serial bus, GPIO
	  ○ Internet **connectivity** around asset
	  ■ e.g. Cellular, Ethernet, Wi-Fi network or GPS
	  ○ **Environmental conditions** around asset
	  ■ e.g. operating temperature or asset location (in a moving container/vehicle,

	factory, retail store, outdoors) ○ **Power supply** requirements, **sensor** installation requirements
Project Dimensions	● Purpose: is to provide a checklist that helps the project manager to carry out their own assessment for the project (self-assessment) ● Results used to identify critical areas and project risks as well as input for the solution architecture and design decisions <u>5 Main Dimensions</u> ● Assets and Devices ○ Covers aspects like number and value of assets, asset complexity, required on-asset processing power, hardware requirements and lifecycle management ● Communications and Connectivity ○ Focuses on technology, bandwidth and latency both for local and remote communication ● Backend Services ○ Looks at the complexity of the backend business solution, as well as aspects specific to data management ● Standards and Regulatory Compliance ○ Relates to the external project environment ● Project Environment ○ Examines the internal project environment
Projection (Quantity Structure)	● Quantity structure must be closely aligned with the requirements of the business stakeholders, and should show projected growth in key areas ● Areas that should be included in IoT: ○ Number of assets over time ○ Number of events sent from assets to the backend ○ Number of users (total and concurrent) ○ Number of process instances per year (for key processes such as asset activation, incidents,

	etc) ● This information will be of vital importance to the solution architect Quantity Structure (in every year) 
Milestone Plans	● Plan will include some normal / IoT-specific milestones: ○ Make-or-buy decision ○ Vendor selection ○ Prototype ○ Pilot / field test ○ SOP / Start of production (hardware) ○ Initial release of backend solution ● When structuring the milestone plan, it can make sense to have two separate timelines, one for the on-asset components and one for the backend, and to include the key dependencies between them

* Usage View Point

Solution Sketch	● Attempts to identify or define the key events (or any other key communications) submitted from the asset to the backend ● Captures key UIs, business processes, rules and data entities (all in list form) ● Databases and applications are shown, as well as external partners to be integrated ● The solution sketch is more detailed than the executive

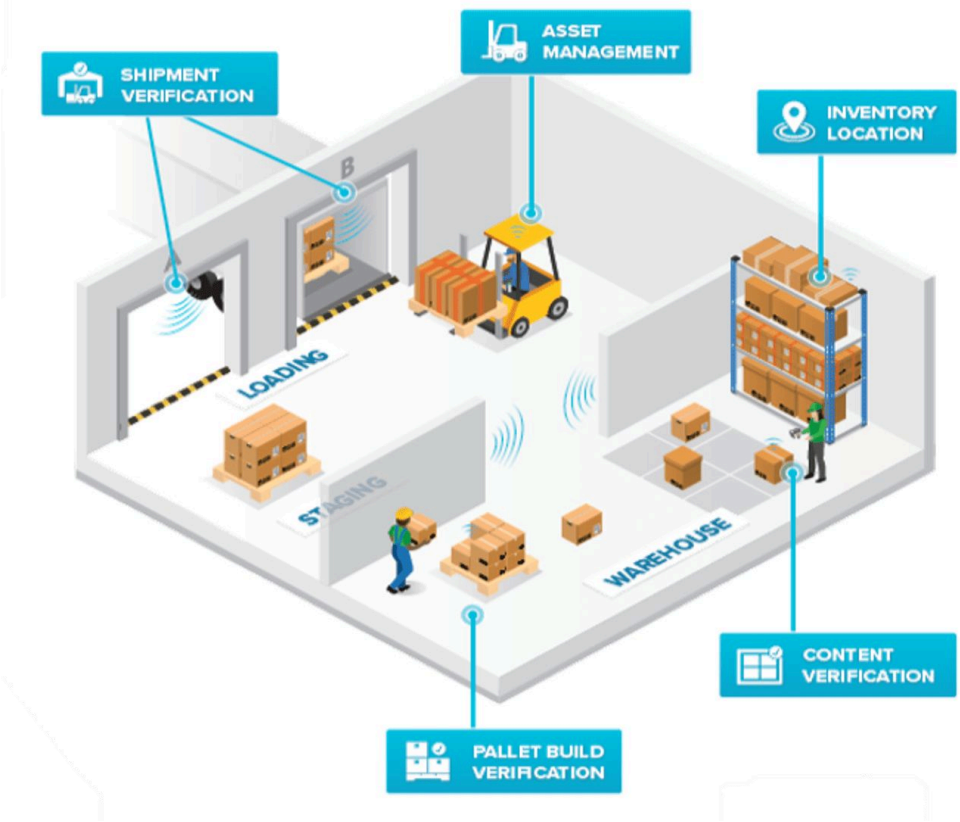
	summary • Focuses on key entities only, and does not provide a formal and complete list of these entities • Main purpose: start narrowing down the solution scope and to create a common basis for communication between business and technical project stakeholders  IoT Solution Sketch

Functional View Point

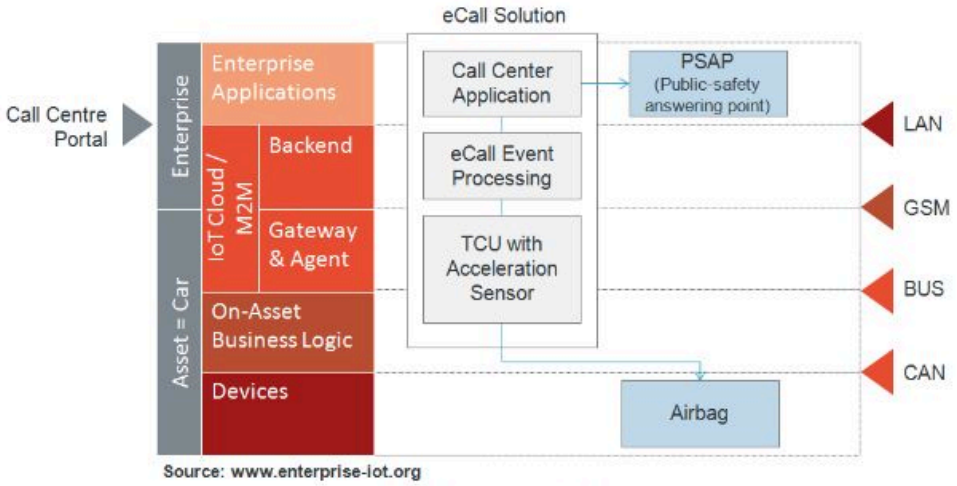
- High-level catalog of the most important use cases
- BPMN (Business Process Model and Notation)
    - Management diagram for a more process-oriented organization
    - **Carried by**: Business Analyst
    - **Supported by**: Solution Architect
    - **Key aspect of functional design**: close interaction with end users and business stakeholders
    - **Recommendation**: User Interface / Human Machine Interfaces (HMI) mock-ups
    - **Usefulness**: an ideal basis for discussing ideas and validating requirements with end users and business stakeholders. They can also help identify and capture the full picture in terms of important data entities, use cases, processes, events and so on

UI Mock-up	For IoT solution, the following mock-ups are typically relevant: ● **Human Machine Interaction (HMI)** ○ Asset user and asset administrator interfaces ○ e.g. machine HMI displays, remote web interfaces, car on-board displays ● **End user self-service interfaces** ○ User account management, personal usage statistics ● **Process support interfaces** ○ e.g. UI for center agents ● **Partner interfaces** ○ UIs for solution partners, incident management for external support providers
Domain Model	● Powerful tool in helping to create a business-oriented, consolidated view of the key data entities of an IoT solution ● Recommend using very **basic UML elements** for the domain model: ○ Classes ○ Attributes ○ Associations ○ Aggregations ○ Inheritance / Specialization ● Advantage: ○ Ability to create a visual representation of the data distribution across the different solution layers. Leveraging SOA (Service-Oriented Architecture), we can map the entities from the domain model to the different components

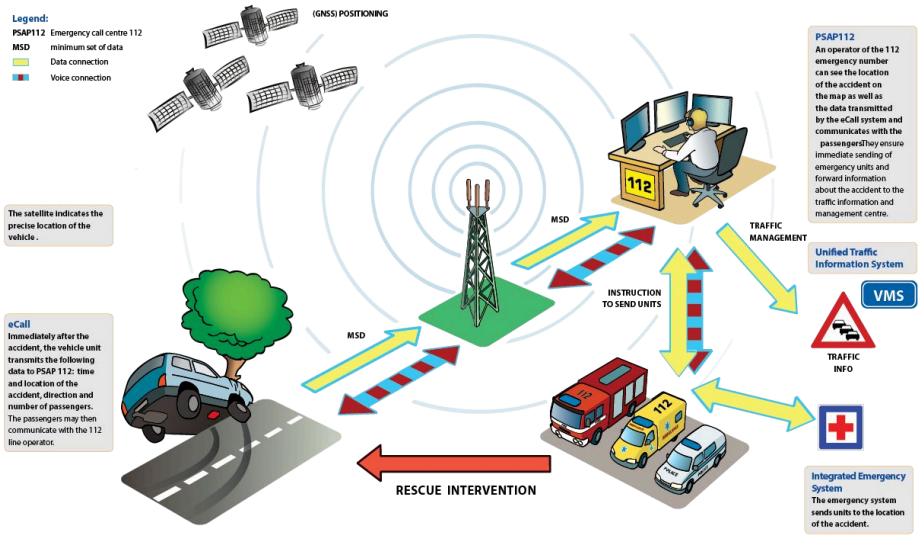
	
Asset Integration Architecture (AIA)	• An architectural approach with two main tiers: ○ **Asset** and **Enterprise** (backend), integrated through an IoT Cloud or an M2M Platform • In the **asset**, the local gateway and agent software allow integration with different devices that form part of the asset and ensures connectivity with the backend • In the **backend**, the IoT Cloud usually has an integrated database containing information about all of the assets that have been registered with the system • IoT Cloud ensures that this information synchronized with the actual data from the asset • IoT Cloud backend typically contains every asset- and device-specific data and functionality and is integrated with backend applications to provide additional business services



- AIA Example: eCall Service
  - Consists of call center application in the backend, which integrated with PSAP (Public Safety Answering Point) 112
  - The backend also contains eCall Event Processing subsystem, which receives events from vehicles in potential crash situation
  - Events are submitted from a TCU (Telematics Control Unit) in the car. Main purpose of TCU is to act as a gateway, but also has some limited business logic running locally, as well as an acceleration sensor which forms part of the solution. TCU integrated with the airbag via a CAN bus
  - Main purpose of AIA is to provide canvas that present different architectural elements of an IoT solution in a standardized way and to provide information about the distribution of the different architecture elements in the IoT solution



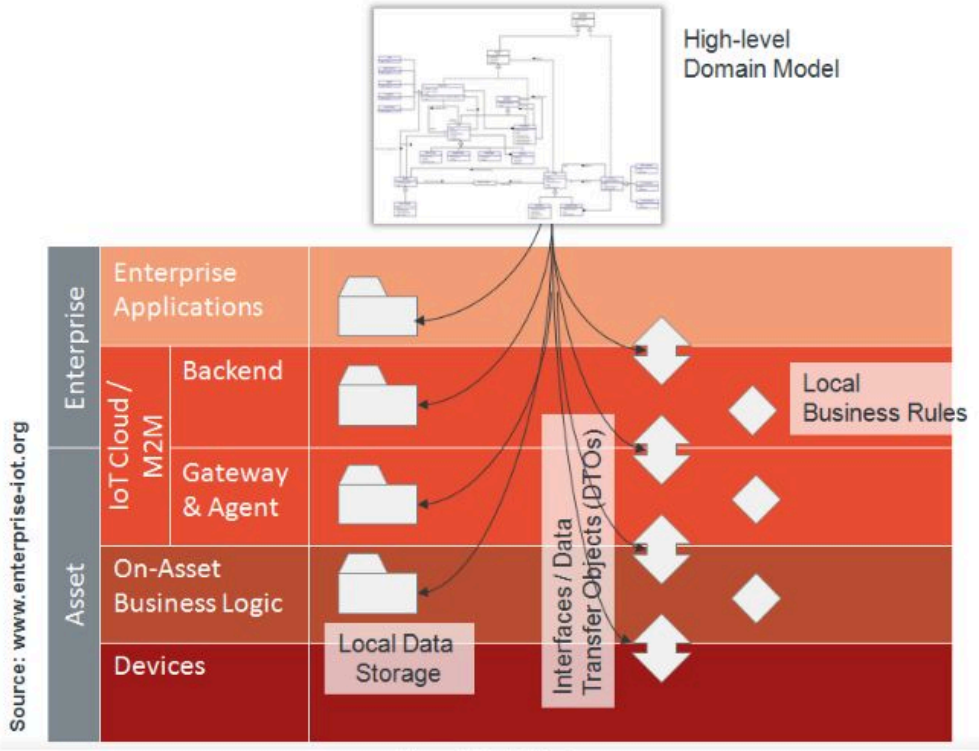
AIA example: eCall service



- AIA Example: Distributed Database / Data Distribution Design
  - **Combine Domain Model with Asset integration Architecture** to address data distribution
  - For each of the entities identified in the domain model, the task is to identify where in the AIA these entities might exist (either by design (green-field), or because they already do (brown-field). Closely related to the distribution of data is the distribution of key business rules
  - Experienced solution architect will typically have a good overview of the key restrictions that apply to

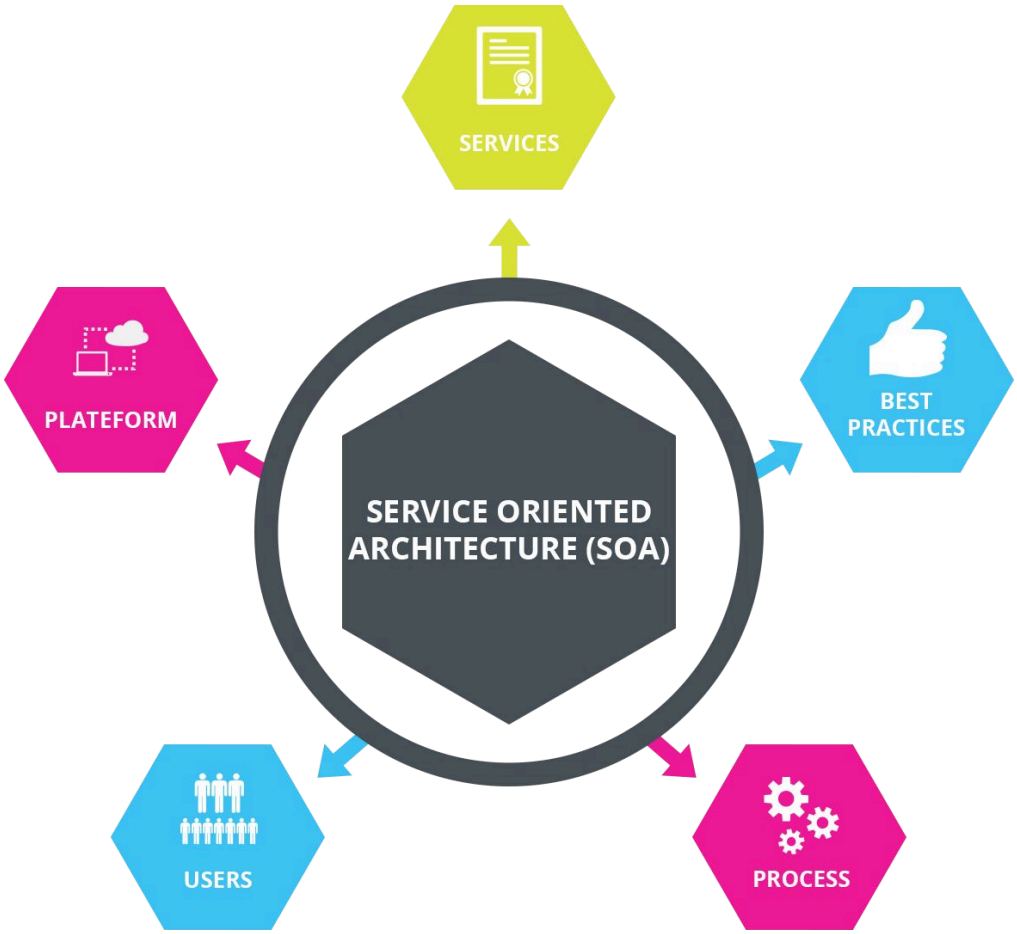
data and business rule distribution, such as **latencies** and **bandwidths** between the different tiers of AIA, **local storage restrictions** and **processing limitations**

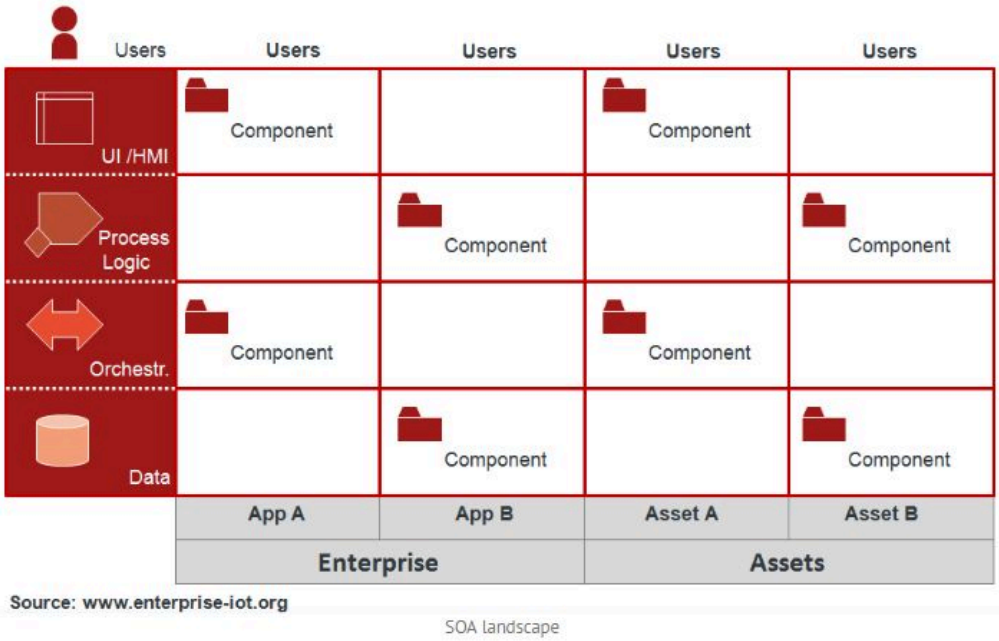
- By combining these restriction with quantity structure information from the original analysis, they can create an initial proposal for the data distribution design by mapping data entities and rules to AIA



AIA and data distribution

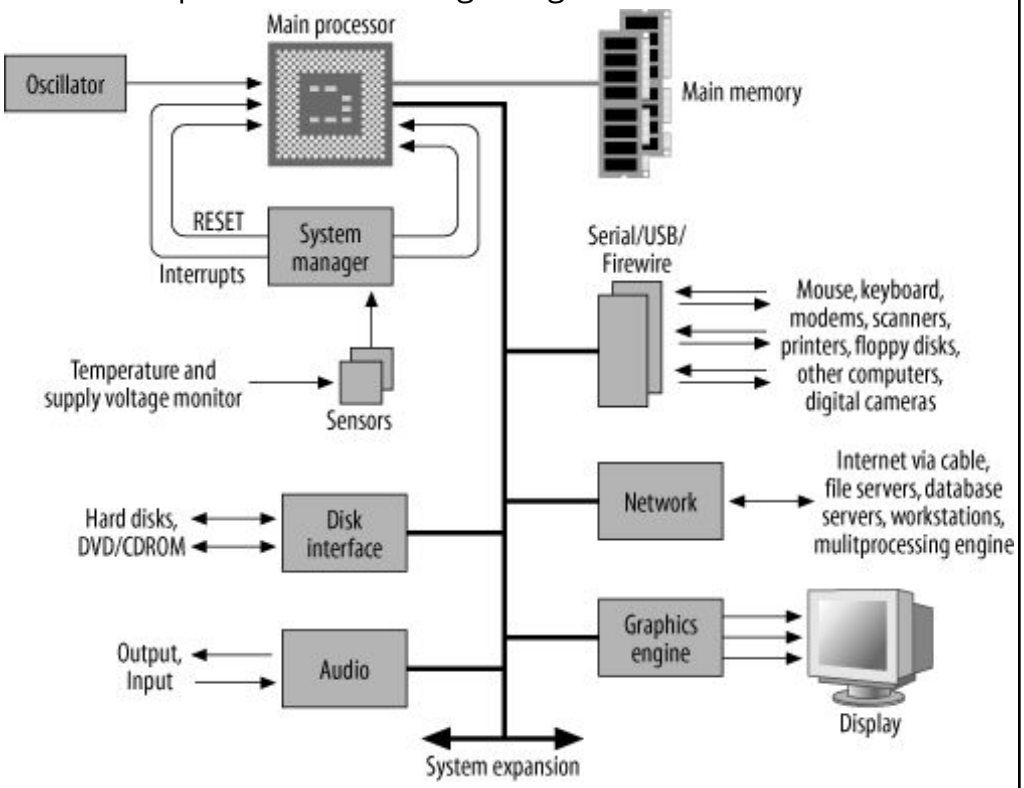
- Service-Oriented Architecture (SOA) Landscape
  - Approach to address problems associated with distributed, heterogeneous software systems
  - In a distributed system with network latencies and bandwidth limitations, the design of the component interfaces must be as efficient as possible
  - Distributed object computing has shown that complex object interaction patterns are not suitable for most distributed systems
  - Instead, simpler, RESTful services have been introduced. The same patterns and restrictions also

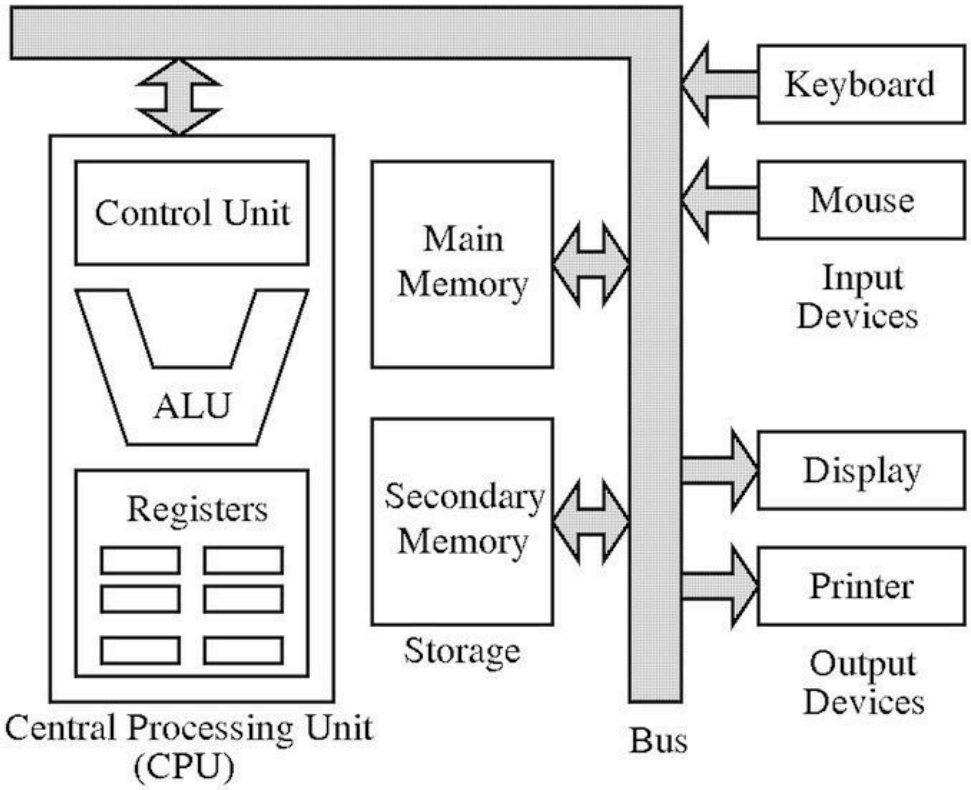
	apply to the design of software components in the IoT - if anything they apply more, given that aspects such as high latencies, network service disruptions and other problems are even more of a factor due to the open nature of the IoT ○ From SOA point of view, it takes a technology-agnostic perspective, adding embedded applications and real-time systems to the existing already heterogeneous SOA shouldn't make a difference 

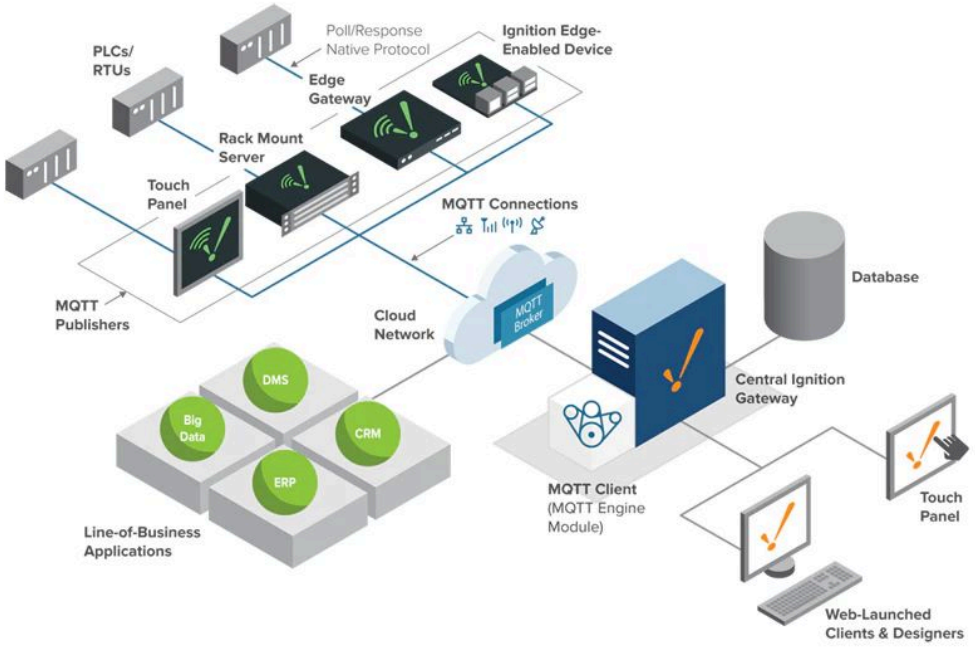
	 SOA landscape

Implementation View Point

Software Architecture	Define the key software components and their dependencies: ● On-asset OS / firmware, application container and middleware ● Backend OS, application container and middleware (e.g. messaging, BPM or BRM) ● Database technologies and libraries ● Identity management, security technologies, certificate management ● Important software libraries and open-source components ● Infrastructure for software updates on the assets ● Development tools
Hardware Design	● Focus on those elements that are specific to IoT solution - usually the on-asset hardware: ○ On-asset hardware (e.g. gateway consisting elements: CPU, memory, local interfaces, cellular modem, local wireless connectivity, antenna, casing, distributed sensor nodes) ○ Position and mounting on the asset ○ If distributed sensor nodes are deployed, the position

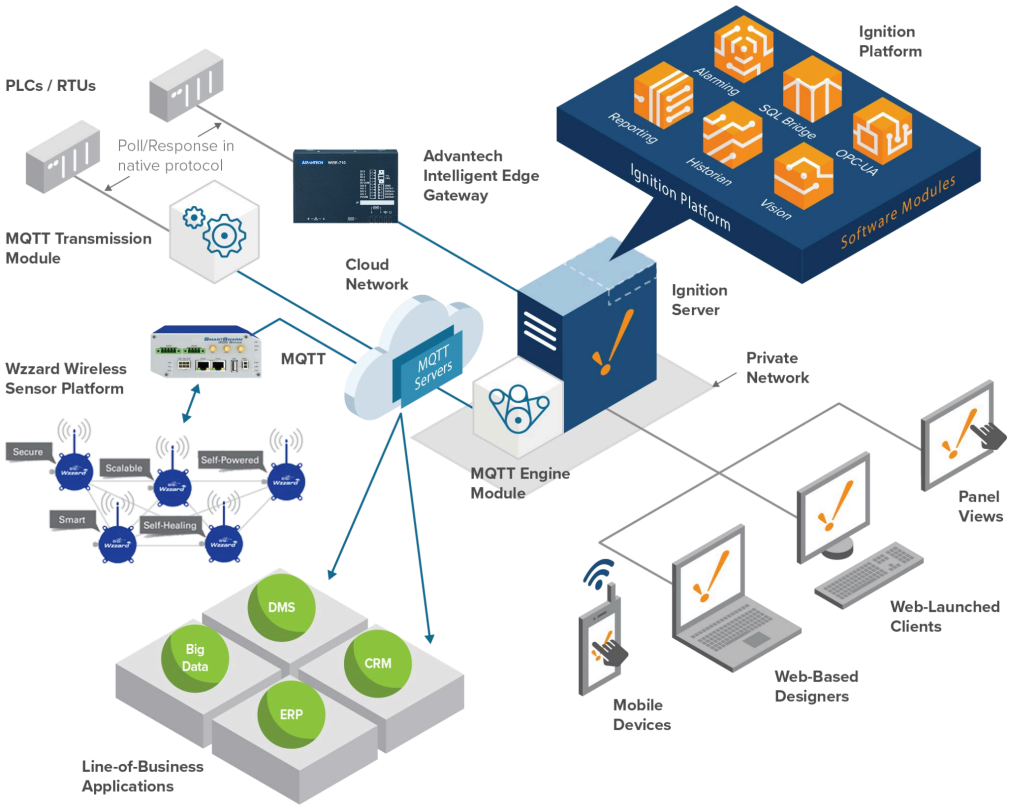
	and mounting of these nodes on the asset also • In case of custom hardware development, the next level of detail in the hardware design would address the main components (e.g. CPU, memory, power supply, digital I/O, communication modules) • Example: Hardware design diagram 

	
Integration Architecture (Technical Infrastructure)	• Design of technical infrastructure will be highly specific to the individual solution and, in particular, to the network infrastructure used • Example: Advantech.com



Ignition IIoT Architecture

Cloud-Based Redundant MQTT Server Solution



* Vending Machine

IoT Architecture

Device Layer	• Sensors for temperature, weight, stock level • Payment modules (NFC, QR) • Cameras
Communication Layer	• Sends data via Wi-Fi, 4G/LTE or NB-IoT to the cloud

Core Platform Layer	• Device management
	  ○ Registering new machines into the network.
	  ○ Tracking each machine's operational status (online/offline, errors, inventory).
	  ○ Managing access control (who can view/control a device).
	• Firmware updates
	  ○ Security patches to prevent vulnerabilities.
	  ○ Feature upgrades (e.g., new payment options).
	  ○ Fixing bugs without physical access.
	• Data integration
	  ○ Synchronizing stock data with supply chain systems.
	  ○ Pushing transaction records to accounting software.
	• Real-time event handling
	  ○ Triggering alerts to operators.
	  ○ Activating specific actions (e.g., reorder restock, lock system).

Analytic Platform Layer	• Predicts stock refill times, user demand patterns • Machine health monitoring
Solution Layer	• Dashboards • Mobile apps • Alert systems • Automated maintenance requests

IoT Use Cases

Real-time inventory monitoring	• Detects when an item is low or out-of-stock • Sends automatic refill alerts to operators • Reduces missed sales and manual checking
Predictive maintenance	• Monitors machine health (temperature, vibration, power usage) • Predicts and prevents breakdowns before they happen • Sends maintenance alerts when abnormal behavior is detected
Remote monitoring and management	• Allows operators to view the status of all vending machines remotely • Enables remote software updates or locking the machine in emergencies
Smart payment integration	• Accepts cashless payment (NFC, credit card, QR code, mobile wallets) • Secure transaction handling through encrypted channels
User behavior analytics	• Tracks popular products, buying times and demographics (via camera or user profiles) • Supports personalized promotions or restocking decisions
Dynamic pricing and promotions	• Changes prices based on demand, time of day or stock levels • Offers discounts or combo deals dynamically
Energy management	• Controls internal lighting, cooling systems and display screens for energy efficiency • Detects when the machine is idle to reduce power usage

Benefits of IoT in Vending Machines

Automation:	• Guarantee fewer out-of-stock situations

Increased supply chain efficiency	• When supplies run low, the software automatically transmits a order to the warehouse • Automatic ordering can save the store managers' effort to call the contact center when preventing lost sales due to lack of merchandise • Drivers can restock more locations each day because they already know which locations actually need supplies
Sales optimization: Increase products sales	• Customize product assortment and placement based on review of purchase histories • If a certain beverage and snack combination proves popular in a particular region, the two items can be placed near each other to encourage additional purchases
Availability: Proactive problem management	• Vending Machine API can send alerts when it senses conditions that typically precede problems • Technicians can diagnose issues over the network, eliminating unnecessary service calls • Reduces service and maintenance costs • The action of repairing problems before equipment stops working also prevents lost sales
Usability: Enhanced customer experience	• Enable cashless and contactless payments • Support multi-language interfaces and touchless control via mobile apps • Offer personalized promotions using customer profiles or loyalty programs
Monitoring: Improved security and theft prevention	• Use camera sensors, motion detectors or tamper sensors to detect theft or vandalism • Send real-time alerts and video snapshots to security personnel • Lockdown capabilities during unauthorized access attempts
Cost: Energy efficiency and sustainability	• Smart power usage by dimming lights or controlling cooling based on ambient conditions or activity • Predictive maintenance reduces waste by

	preventing full machine replacements ● Eco-friendly restocking routes can be optimized using real-time need data, reducing fuel use

Resources & Activities Examples

Unique data sets	Collect data on customer preferences, peak hours, product combinations - build unbeatable demand models for restocking and promotions
Analytics	Use ML to predict top-selling items per location and auto-adjust pricing (dynamic pricing)
Exclusive co-operations & contracts	Partner exclusively with a popular beverage or snack brand, giving your machines exclusive products
Talent access	Hire embedded engineers + data scientists to develop custom vending OS and real-time analytics
Iot device / product development	Develop a modular vending machine that can self-diagnose issues and reorder stock automatically
Algorithm development	Build predictive algorithms that combine weather + foot traffic + purchase trends to forecast demand

* Smart Farming

General Market Opportunity

Factor	Analysis
Growing global population	Rising food demand is pushing the need for higher agricultural productivity - a key opportunity for smart farming to optimize output
Labor shortage in agriculture	Many countries face aging farmer populations and youth disinterest in farming - automation and smart farming tools fill this gap
Climate change and sustainability pressure	There is increasing pressure to use water, fertilizer and land more efficiently - smart farming supports sustainable, data-driven practices
Government initiatives and funding	Many governments are investing in smart agriculture through subsidies and grants (e.g. EU, US, Japan, Malaysia, India)
Tech readiness and IoT growth	IoT devices (drones, sensors, automation tools) are more affordable and reliable, making implementation in farms more accessible
Demand for transparency and quality	Consumers and businesses are demanding traceable, high-quality product - smart farming helps monitor and certify crop conditions

Time and Channel to Market

Aspect	Analysis
Time to market	Moderate - building a smart farming solution may take 6-18 months depending on the complexity (e.g. drone automation, soil sensors, AI analysis). Faster time-to-market possible if using ready-made IoT platforms.
Hardware lead	IoT sensors, drones, smart irrigation controllers need

times	sourcing - can delay deployment by weeks / months. Partnering with existing suppliers is ideal.
Software Platform	If cloud-based dashboard or analytics is required, development time must be allocated. Using platforms like AWS IoT, Blynk or ThingsBoard can reduce development time
Channels to Market	• B2B model: Direct to farms / agriculture companies via agricultural expos • Partnerships: Collaborate with agriculture departments, NGOs, or agri-tech distributors • Digital platforms: Online campaigns, agricultural SaaS listings or IoT marketplaces
Pilot opportunities	• Pilot programs in collaboration with universities, smart farming zones or local farmer associations can validate early product versions
Adoption barriers	• Farmers' resistance to new tech • Limited internet in rural areas • Initial cost vs ROI concerns • These can be mitigated via training, mobile apps, and government grants / subsidy awareness

IoT Application / Use Cases

Key Features of Smart Farming

Real-time monitoring	Uses sensors to monitor soil moisture, temperature, humidity, livestock health and crop growth in real time
Precision irrigation	Automatically adjusts water supply based on soil moisture and weather data to avoid over- or under-watering
Automated machinery	Self-driving tractors, robotic seeders and drone sprayers that reduce human labor and increase efficiency
Data analytics and forecasting	AI and machine learning models analyze historical and real-time data to predict yields, detect diseases and plan

	harvesting
Remote management	Farmers can manage field operations using smartphones or web dashboards from anywhere
Livestock monitoring	Wearable IoT devices monitor animal vitals, behavior and location
Climate monitoring	Weather stations and sensors help forecast conditions and prepare accordingly
Inventory and supply chain tracking	Tracks the storage and distribution of produce using GPS and RFID for freshness and traceability

Use Cases of Smart Farming

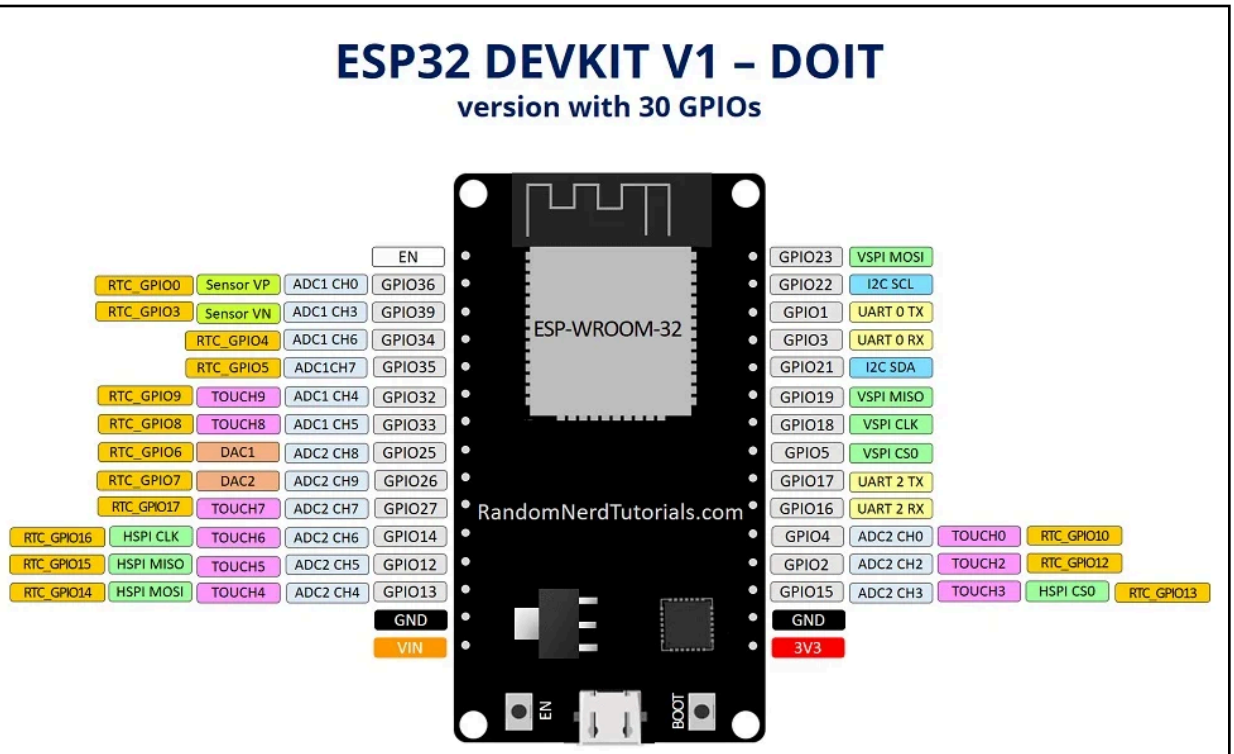
Smart Irrigation	Sensors detect dry soil and activate irrigation only when needed → save water and improves crop health
Crop health monitoring	Drones or cameras scan crops to identify disease or pests early → reduces pesticide use and prevents spread
Autonomous Tractors and Robots	Robotic seeders plant crops with high accuracy → saves time and labor
Weather-based farming decisions	AI predicts rainfall and frost → helps farmers plan planting or harvesting schedules
Livestock health and tracking	GPS ear tags monitor cattle location and activity → reduces loss and ensures animal welfare
Greenhouse automation	Temperature, light and humidity are automatically adjusted → ensures optimal growth conditions
Yield prediction	AI analyzes past data to estimate crop output → helps with pricing and logistics planning
Supply chain optimization	RFID tracks produce from farm to market → improves traceability and reduces spoilage

Resource & Activities Examples

Unique data sets	Gather crop/soil/weather/yield data per region - optimize planting schedules and develop highly localized farming models
Analytics	Apply AI to predict irrigation needs or pest risk, optimizing input use and improving yield
Exclusive co-operations & contracts	Partner with a major agri-equipment maker or sensor provider to offer bundled services competitors can't match
Talent access	Combine agronomists with IoT and AI talent to offer farmers decision support systems tailored to local conditions
Iot device / product development	Design cost-effective solar-powered sensor nodes with multi-sensing capability (soil, moisture, pH, light) and long-range communication
Algorithm development	Use multi-layered ML models that forecast crop yield or detect early signs of plant disease via imagery and environmental data

ESP32 Programming using Arduino IDE

ESP32



Processor	Features either a dual-core or single-core Xtensa LX6 microprocessor, with clock speeds ranging up to 240MKz
Memory	Includes on-chip SRAM and varying amounts of flash memory for storing code
Wi-Fi	Supports 802.11 b/g/n Wi-Fi standards
Bluetooth	Dual-mode bluetooth (Classic and Bluetooth Low Energy - BLE)
GPIO Pins	A range of general purpose input / output pins for connecting sensors actuators, LEDs, displays and other peripherals
Peripherals	Includes analog-to-digital converters (ADCs), digital-to-analog converters (DACs) and more

Low Power Consumption	Designed for battery-powered applications and IoT devices
USB-to-Serial Chip	For easy programming and communication with a computer
Power regulator	To handle power input
Reset and Flash buttons	For programming the ESP32
Breadboard-friendly pin layout	For easy connections
Arduino compatible	NodeMCU modules often come pre-flashed with NodeMCU firmware, allowing programming with the Lua scripting language. They are also compatible with Arduino IDE

Key Features

- NodeMCU based on ESP-WROOM-32 module
- Based on ESP32 DEVKIT DOIT
- 30 GPIO Version
- ESP32 is a dual-core 32-bit processor with built-in 2.4 GHz Wi-Fi and Bluetooth
- 4MByte flash memory
- 520KByte RAM
- 2.2 to 3.6V Operating voltage range
- In breadboard-friendly breakout
- USB micro B for power and Serial communication, use to load program and serial debugging too

Functions

Setup function	• Execution: Runs only once when the ESP32 board powers up or resets • Purpose: Primarily used for initialization tasks • Pin Modes ○ Declaring whether pins are inputs, outputs, or used for specific protocols

	○ e.g. *pinMode(LED_BUILTIN, OUTPUT);* ● Initial States ○ Setting initial pin states ○ e.g. *digitalWrite(LED_BUILTIN, HIGH);* ● Serial Communication ○ Starting the serial monitor for debugging or communication ○ e.g. *Serial.begin(115200);* ● Library Initialization ○ Configuring and initializing any libraries you are using ○ e.g. Wi-Fi, sensors, displays ● Network Connection ○ Connecting to a Wi-Fi network or establishing other communication channels ● One-Time Tasks ○ Any other actions that need to happen only once at the beginning
Loop Function	● Execution: Runs repeatedly and indefinitely after the setup() function finishes ● Purpose: Contains the main logic of your program ● Reading Sensors ○ Get data from sensors attached to the ESP32 ● Processing Data ○ Perform calculations or make decisions based on sensor readings ● Controlling Outputs ○ Turn LEDs on / off, control motors, or send signals to other components ● Sending / Receiving Data ○ Communicate with other devices or services over networks like Wi-Fi or Bluetooth ● Periodic Tasks ○ Execute actions at regular intervals ○ e.g. Checking for updates, sending status messages ● Responding to Events ○ React to button presses, sensor triggers or incoming network messages

Important Considerations

Non-Blocking Code	Avoid long delays or blocking operations in the loop() function. Instead, use timers, interrupts or task scheduling to ensure your code remains responsive
Global Variables	Variables declared outside both setup() and loop() are accessible from both functions

Wi-Fi Connection + LED Toggling

```
#include <WiFi.h>

const char* wifi_ssid = "B100M";
const char* wifi_password = "12345678";
WiFiClient espClient;

const int indicator_WiFi = 25;
bool ledwifi_state = 0;

void setup() {
     Serial.begin(115200);
     initialize();
     setup_wifi();
}

void initialize() {
     pinMode(indicator_WiFi, OUTPUT);
}

void setup_wifi() {
     delay(10);
     WiFi.begin(wifi_ssid, wifi_password);
     while (WiFi.status() != WL_CONNECTED) {
          Serial.println("WiFi reconnecting…");
          delay(500);
     }
     ledwifi_state = !ledwifi_state;
     digitalWrite(indicator_WiFi, ledwifi_state);
     Serial.println("WiFi connected.");
}
```

Button Toggling LED

```
#define BUTTON_PIN 15
#define LED1_PIN 23
int buttonLastState = HIGH;
int buttonCurrentState;

void setup() {
     Serial.begin(115200);
     pinMode(BUTTON_PIN, INPUT_PULLUP);
     pinMode(LED1_PIN, OUTPUT);
}

void loop() {
     readButton();
}

void readButton() {
     buttonCurrentState = digitalRead(BUTTON_PIN);
     if (buttonLastState == LOW && buttonCurrentState = HIGH) {
          Serial.println("The state changed from LOW to HIGH");
          digitalWrite(LED1_PIN, 0);
     }
     else if (buttonLastState == HIGH && buttonCurrentState ==
     LOW) {
          Serial.println("The state changed from HIGH to LOW");
          digitalWrite(LED1_PIN, 1);
     }
     buttonLastState = buttonCurrentState;
}
```

* To upload compiled code to NodeMCU ESP32 Module,
    - Click Sketch → Verify / Compile and Sketch → Upload to compile the code and upload to the NodeMCU ESP32 module

LCD 16x2 Module

Protocol used	I^2C (Inter-Integrated Circuit) communication protocol

Connection lines for communication	• SDA (Serial Data Line)
	  ◦ Function: **Carries the data being transmitted between the master and slave devices**
	  ◦ Direction: Bidirectional - used for both sending and receiving data
	  ◦ Shared bus: All devices on the I^2C bus use the same SDA line
	  ◦ Pull-up: Requires a pull-up resistor to keep the line HIGH when idle
	• SCL (Serial Clock Line)
	  ◦ Function: **Carries the clock signal generated by the master device to synchronize data transmission**
	  ◦ Direction: Unidirectional - from master to all slaves
	  ◦ Timing: Controls when bits on the SDA line are valid
	  ◦ Pull-up: Also requires a pull-up resistor

What would happen if connecting 2 LCDs to the same Grove board?	Bus collision might occurs since most I^2C LCD 16x2 modules use a fixed address like 0x27 or 0x3F • Both devices respond at the same time • Garbled text, unreliable behavior or even no display would happen

Code Implementation

```
import time

from seeed_dht import DHT
from grove.display.grove_lcd import *

DHT_pin = 5
sensor = DHT("11", DHT_pin)
print("Detecting temperature...")
```

```
while True:
     try:
          time.sleep(0.5)
          humi, temp = sensor.read()
          t = str("{0:.1f}".format(temp))
          h = str("{0:.1f}".format(humi))
          print("temperature "+ t + "\u00b0C, " + "humidity " + h +
          "%")
          setText("Temp = " + t + "\337C" + "   Hum = " + h + " %")
     except KeyboardInterrupt:
          print("Program Exited")
          break
     except TypeError:
          print("Type Error occurred")
     except IOError:
          print("IO Error occurred")
```

`seeed_dht`	library for interacting with DHT11 sensor
`DHT("11", DHT_pin)`	Initializes DHT11 sensor object

`humi, temp = sensor.read()`
`t = str("{0:.1f}".format(temp))`
`h = str("{0:.1f}".format(humi))`
	• Reads temperature and humidity from the sensor
	• Formats readings to one decimal place and converts them to strings

`\u00b0C`	Unicode character for °C
`\337C`	Trick to display degree symbol (°) on Grove LCDs
`KeyboardInterrupt`	If pressing Ctrl + C, the loop breaks and the program exits cleanly
`TypeError, IOError`	Catches and reports errors that can occur due to faulty sensor reads or connection issues

MQTT (Message Query Telemetry Transport)

Establish MQTT Broker and Client (Subscriber and Publisher) using Raspberry Pi

Broker Address	• **Address of MQTT broker/server that handles message routing between publishers and subscribers** • All clients must connect to the same broker to exchange messages • e.g. test.mosquitto.org, broker.hivemq.com, 192.168.1.100
MQTT Topic	• **Defines the "channel" for publishing and subscribing messages**

Port Number	• **TCP port on which the broker listens for MQTT connections**
	• Common values:
	  ○ **1883** for unencrypted communication (default)
	  ○ 8883 for encrypted TLS/SSL communication
	• Ensures proper routing of the MQTT connection through the broker

Code Implementation in Thonny Python

```
from grovepi import *
from paho.mqtt.client import *

buzzer = 3
pinMode(buzzer, "OUTPUT")
MQTT_BROKER = [BROKER IP ADDRESS]
MQTT_TOPIC = "test"

def on_connect(client, userdata, flags, rc):
      print("Connected with result code " + str(rc))
      client.subscribe(MQTT_TOPIC)

def on_message(client, userdata, msg):
      print(msg.topic + " " + str(msg.payload))
      try:
```

```
            i = int(msg.payload)
            print(i)
            if i > 0 and i < 256:
                  analogWrite(buzzer, i)
      except:
            analogWrite(buzzer, 0)

client = Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(MQTT_BROKER, 1883, 50)
client.loop_forever()
```

`paho.mqtt.client`	MQTT client library for connecting and communicating with an MQTT broker
`on_connect(client, userdata, flags, rc)`	• Triggered when client connects to the broker • Subscribes to the "test" topic
`on_message(client, userdata, msg)`	• Called when a message is received • Converts the message payload to an integer (i) • If the value is between 1-255, it sets the buzzer strength using analogWrite() • If there is an error or invalid value, it turns the buzzer off
`client = Client()`	Creates a new MQTT client
`client.on_connect = on_connect` `client.on_message = on_message`	Assigns on_connect and on_message callbacks
`client.connect(MQTT_BROKER, 1883, 50)`	Connects to the broker at port 1883 (default for MQTT)
`client.loop_forever(0`	Keeps the client running and listening

Code Implementation in Arduino IDE

```
#include <WiFi.h>
```

```
#include <PubSubClient.h>

#define LED_WIFI 25
#define LED_MQTT 33
#define LED_COMEXE 32

bool ledwifi_state = 0;
bool ledmqtt_state = 0;
bool ledcommand_state = 0;
String mqtt_command = "";
bool serialUpdate = 0;
bool command_status1 = 0;

const char* wifi_ssid = "B100M";
const char* wifi_password = "12345678";
const char* mqtt_server = "192.168.200.100";
const char* mqtt_topic = "BAIT2123/DEMO";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
      Serial.begin(115200);
      initialize();
      setup_wifi();
      setup_mqtt();
}

void loop() {
      runLED();
      client.loop();
}

void initialize() {
      pinMode(LED_WIFI, OUTPUT);
      pinMode(LED_MQTT, OUTPUT);
      pinMode(LED_COMEXE, OUTPUT);
}

void setup_wifi() {
      ledwifi_state = 0;
      delay(10);
      WiFi.begin(wifi_ssid, wifi_password);
      while (WiFi.status() != WL_CONNECTED) {
            Serial.println("WiFi reconnecting...");
```

```
            ledwifi_state = !ledwifi_state;
            digitalWrite(LED_WIFI, ledwifi_state);
            delay(500);
      }
      digitalWrite(LED_WIFI, 1);
      Serial.println("WiFi connected.");
}

void setup_mqtt() {
      client.setServer(mqtt_server, 1883);
      client.setCallback(callback);
      reconnect();
      client.subscribe(mqtt_topic);
      Serial.println("MQTT setup done");
}

void reconnect() {
      ledmqtt_state = 0;
      while (!client.connected()) {
            if (client.connect("", NULL, NULL)) {
                  Serial.println("MQTT reconnecting...");
                  ledmqtt_state = !ledmqtt_state;
                  digitalWrite(LED_MQTT, ledmqtt_state);
                  delay(500);
            }
      }
      digitalWrite(LED_MQTT, 1);
      Serial.println("MQTT connected.");
}

void callback(char* topic, byte* message, unsigned int length) {
      ledcommand_state = !ledcommand_state;
      digitalWrite(LED_COMEXE, ledcommand_state);
      mqtt_command = "";
      serialUpdate = 0;
      for (int i = 0; i < length; i++) {
            mqtt_command += (char)message[i];
      }
      if (mqtt_command == "led1=0") {
            serialUpdate = 1;
            command_status1 = 0;
      }
      if (mqtt_command == "led1=1") {
            serialUpdate = 1;
            command_status1 = 1;
```

```
      }
      if (serialUpdate == 1) {
            Serial.print("MQTT command received: ");
            Serial.println(mqtt_command);
      }
}

void runLED() {
      if (command_status1 == 0) {
            // do something when this mqtt command is received
      }
      else if (command_status1 == 1) {
            // do something when this mqtt command is received
      }
}
```

`espClient`	Used by PubSubClient for network transport
`client.setServer(mqtt_server, 1883);`	Connect to broker
`client.setCallback(callback);`	Assigns a callback function (callback) to handle incoming messages
`reconnect();` `client.subscribe(mqtt_topic);`	Reconnects to the broker and subscribes to a topic
`callback(char* topic, byte* message, unsigned int length)`	• Toggles command LED to indicate reception • Stores message • Checks if it's "led1=0" or "led1=1" and sets command state • Outputs result to Serial

NodeRed Software

Input	• **Inject data into a flow - typically from sensors, user input, or external systems**
	• Features:
	  ○ Triggers the flow
	  ■ Starts the data flow by receiving or injecting messages

	○ Provides msg.payload ■ Sends the initial msg object (usually with msg.payload) ○ Can be time-based or event-based ■ e.g. periodic injection, MQTT message., HTTP request ● Example input node: **inject, mqtt in, http in, gpio in, serial in, websocket in**
Function	● **Process or transform data using customer JavaScript logic** ● Features: ○ Custom JavaScript code ■ Write JS to manipulate the message ○ Access to msg object ■ Full access to msg.payload, msg.topic, etc ○ Can create new properties ■ Can modify or add data to pass on to the next node ● Example function node: **function, change, switch, template, range, delay**
Output	● **Send or display the final processed data - to dashboards, devices, APIs** ● Features: ○ Ends the flow ■ Usually the last step - outputs result somewhere ○ Acts on msg.payload ■ Uses data in the msg object to send to destination ○ Various output targets ■ e.g. dashboard, MQTT broker, database, hardware ● Example output node: **debug, mqtt out, http response, gpio out, serial out, websocket out**

Webcam

Step 1: Connect an USB webcam to Raspberry Pi
Step 2: Install open-cv and numpy library by using the following command (typed in terminal)

```
sudo apt-get install python-opencv
sudo apt-get install python-numpy
```

Code Implementation

```
import cv2

cam = cv2.VideoCapture(0)

while True:
      ret, image = cam.read()
      cv2.imshow('Imagetest', image)
      k = cv2.waitKey(1)
      if k != -1:
            break

cv2.imwrite('/home/pi/testimage.jpg', image)
cam.release()
cv2.destroyAllWindows()
```

`import cv2`	Loads OpenCV library which allows access to the camera and image processing
`cam = cv2.VideoCapture(0)`	• Opens the default camera (usually the built-in webcam or first USB camera) • 0 = first camera device. Use 1 or 2 if having multiple cameras connected

`while True:`
`  ret, image = cam.read()`
`  cv2.imshow('Imagetest', image)`
	• cam.read() captures a frame:
	  ○ ret: Boolean (True if frame was successfully read)
	  ○ image: The actual frame (image) from the webcam
	• cv2.imshow() shows the frame in a window titled "Imagetest"

`k = cv2.waitKey(1)`
`if k != -1:`
`  break`
	• cv2.waitKey(1) waits 1 millisecond for a key press
	• If any key is pressed (k != -1), it breaks out of the loop

`cv2.imwrite('/home/pi/testimage.jpg', image)`	• Saves the most recent frame as testimage.jpg in /home/pi/ • If the folder doesn't exist or lacks write permission, this will fail silently
`cam.release()` `cv2.destroyAllWindows()`	• .release() stops the camera and frees the resource • .destroyAllWindows() closes all OpenCV windows

View the Webcam in Webpage using Flasks

index.html (Front-end, webpage that shows the video)

```
<body>
<div class="container">
	<div class="row">
		<div class="col-lg-8 offset-lg-2">
			<h3 class="mt-5">BAIT2123 Live Streaming</h3>
			<img src="{{ url_for('video_feed') }}" width="50%">
		</div>
	</div>
</div>
</body>
```

camera.py (Backend camera handler using OpenCV)

```
# import necessary packages
import cv2

class VideoCamera(object):
	# starts the webcam (device 0)
	def __init__(self):
		# capturing video
		self.video = cv2.VideoCapture(0)

	# releases it when the object is deleted
```

```
    def __del__(self):
        # releasing camera
        self.video.release()

    # reads a frame, encodes it as .jpg, and returns raw bytes
    (ready to stream)
    def get_frame(self):
        # extracting frames
        ret, frame = self.video.read()
        if ret == True:
            #encode OpenCV raw frame to jpg and displaying it
            ret, jpeg = cv2.imencode('.jpg', frame)
            return jpeg.tobytes()
```

test19.py (Main Flask app server)

```
from flask import Flask, render_template, Response
import cv2

# initialize the Flask app
app = Flask(__name__)

# function that continuously yields video frames
def gen(camera):
    while True:
        # read the camera frame
        frame = camera.get_frame()
        # concat frame one by one and show result
        yield(b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame +
            b'\r\n')

# / → renders the index.html
@app.route('/')
def index():
    return render_template('index.html')

# /video_feed → Streams video with correct MIME type
@app.route('/video_feed')
def video_feed():
    # multipart/x-mixed-replaced → used to send a series of images
    (frames) one after another
    # gen() → A generator function that yields each frame as a JPEG
    image to the browser
    return Response(gen(VideoCamera()),
```

```
      mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == "__main__":
      app.run(host='172.16.2.85', port='5000', debug=True,
      threaded=True)
```

MFRC522 RFID Module

Protocols Supported	Universal Asynchronous Receiver Transmitter (UART)
2 basic required hardware connection for data communication	● RX (Receive) ○ This pin is used to receive serial data from the Raspberry Pi or any other microcontroller ● TX (Transmit) ○ This pin is used to send serial data to the Raspberry Pi or any other microcontroller

Code Implementation

```
import serial
import grovepi
import time

buzzer = 2
grovepi.pinMode(buzzer, "OUTPUT")
rpiserl = serial.Serial('/dev/ttyS0', baudrate=9600, timeout=1,
bytesize=serial.EIGHTBITS, parity=serial.PARITY_NONE,
stopbits=serial.STOPBITS_ONE, xonxoff=False, rtscts=False,
dsrdtr=False)

rpiserl.flushInput()
rpiserl.flushOutput()

try:
      while True:
            s = rpiserl.read(14)
            if len(s) != 0:
                  print(s.hex())
                  grovepi.digitalWrite(buzzer, 1)
                  time.sleep(0.08)
                  grovepi.digitalWrite(buzzer, 0)
                  time.sleep(0.08)
                  grovepi.digitalWrite(buzzer, 1)
                  time.sleep(0.08)
                  grovepi.digitalWrite(buzzer, 0)
except KeyboardInterrupt:
```

`    print("Program ended")`
`finally:`
`    grovepi.digitalWrite(buzzer, 0)`	

`import serial`
`import grovepi`
`import time`
	• serial: used for serial communication with RFID module
	• grovepi: used to interact with Grove hardware like the buzzer
	• time: for delays in buzz sound

`rpiserl =`
`serial.Serial('/dev/ttyS0',`
`baudrate=9600, timeout=1,`
`bytesize=serial.EIGHTBITS,`
`parity=serial.PARITY_NONE,`
`stopbits=serial.STOPBITS_ONE,`
`xonxoff=False, rtscts=False,`
`dsrdtr=False)`
	• Opens serial port /dev/ttyS0 (used by many UART-based RFID readers)
	• Sets baudrate to 9600, no parity, 8 data bits, 1 stop bit (typical config for RFID)
	• Disable hardware / software flow control

`rpiserl.flushInput()` `rpiserl.flushOutput()`	Clears any leftover data in the input/output serial buffers
`s = rpiserl.read(14)`	Reads 14 bytes of data from the RFID reader. Most RFID cards send 14-byte ASCII data
`if len(s) != 0:` `    print(s.hex())`	If data is received, it is printed in hex format. This is often the UID of the RFID card

`except KeyboardInterrupt:`
`    print("Program ended")`
`finally:`
`    grovepi.digitalWrite(buzzer,`
`    0)`
	Stops the buzzer and ends the program when pressing Ctrl + C