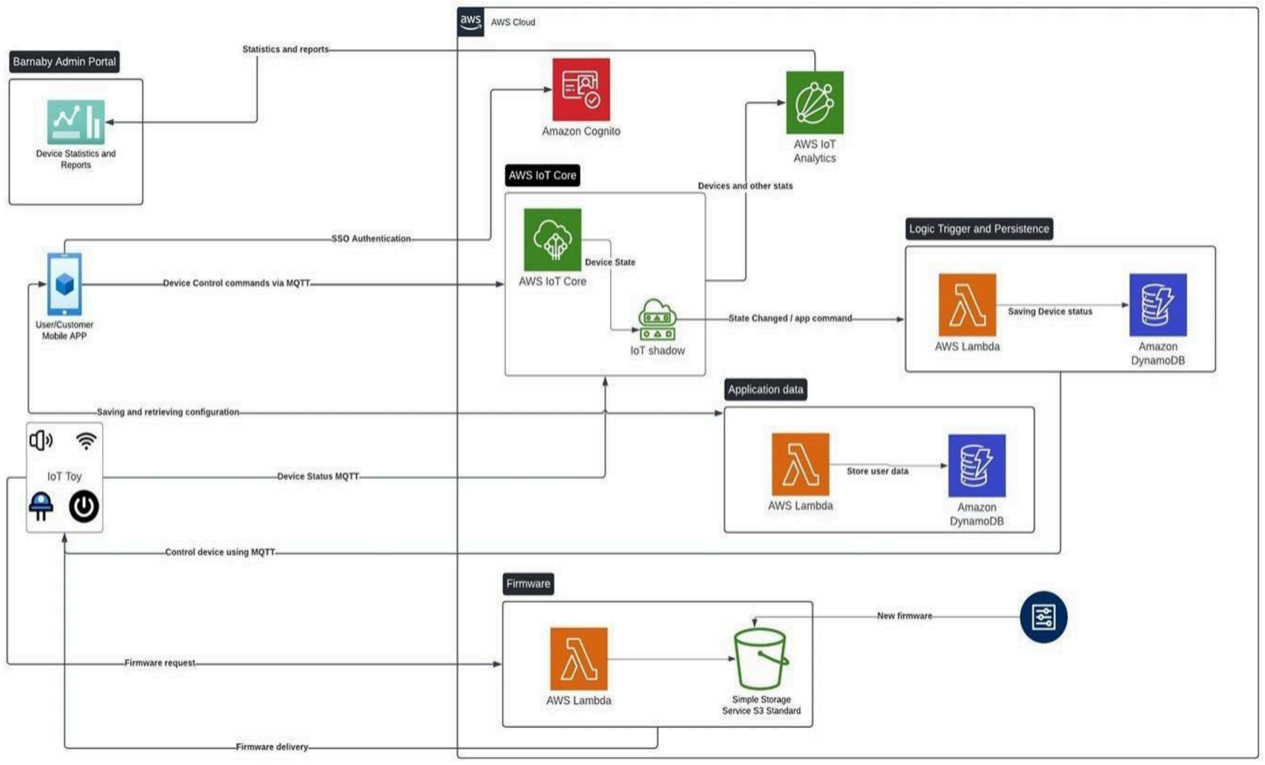
SUBJECT: **IoT och systemarkitektur (IoT23)update**

PROJECT BY: **Nnaka Joyce**

TOPIC: **Sömnis AB:s smarta ljud- och ljusmaskin för bebisar.**

THE ARCHITECTURE



OVERVIEW

SYSTEM COMPONENTS

- **Physical Product** (the toy)
- **Mobile App** (can be iOS/Android):the app enables secure user authentication via Amazon Cognito, facilitates device pairing and control through AWS IoT Core, manages user preferences in DynamoDB, and supports OTA updates, child lock, and favorite settings, while ensuring secure communication via HTTPS, WebSocket for real-time updates, offline functionality with local caching and push notifications, while the Toy's Admin Portal provides company administrators access to device analytics through SSO integration without separate account management.
- **Cloud Backend** (Amazon Web Services)
- **Toy company's Portal** (Web): it is a web-based platform, the system will use the company's existing SSO solution for login (e.g., Active Directory or LDAP, a dashboard with regional reports, comprehensive device management and statistics and firmware distribution management.

LAYERS TO THIS ARCHITECTURE:

- **Perception Layer:** the physical devices on the toy like the sensors, microcontrollers, etc.retrieve data from various sources and produce actions based on the data collected.
- **Connectivity Layer:** via the ESP32, Wi-Fi connects the toy to AWS IoT Core using MQTT.
- **Data Processing Layer:** Amazon web services (AWS Lambda, DynamoDB, IoT Analytics) help process and store the data.
- **Application Layer:** interfaces (both admin and toy user) is provided by the mobile application and admin portal.
- **Process Layer:** this layer integrates business logic, it ensures that raw data which is retrieved from the toy and other sources is converted into useful insights and actionable responses (user preferences, analytics, firmware updates).
- **Security Layer:** data authentication (Amazon Cognito), communication encryption, and overall security on the whole process happens at this layer.

PHYSICAL COMPONENTS

The toy produces light and sound effects, this includes different colours with brightness adjustments.

To achieve its capabilities, the toy involves several key components:

- **Microcontroller:** in this case, an ESP32 chip is right for this, it manages sound, light, and Wi-Fi connectivity for app integration. Wi-Fi (+ESP32) allows wireless control via mobile apps.
- **Power Supply:** a portable, long-lasting Li-ion battery, there is also an additional power source, charging via USB.
- **Speaker:** The DFPlayer Mini reads and plays MP3 files from a microSD card, it supports small speakers without an amplifier.
- **RGB LEDs:** WS2812B LEDs allow for an array of colours with adjustable brightness. It has pulse width modulation (PWM) control for precision.
- **Light Sensor:** it senses and automatically adjusts LED brightness based on ambient light.
- **Voltage Regulation & Circuit Components:** AMS1117 voltage regulators, resistors and capacitors stabilize power and improve responsiveness.
- **Push Buttons:** for user input on the toy.
- **MicroSD Card Slot:** contains pre-recorded sounds and music files for playback.

How the components connect: a toy user presses a button, the microcontroller detects it (the microcontroller receives input from the buttons, app, and sensors). It processes the input and controls the LEDs (colour and brightness) and speaker (for playing sounds) accordingly. The microcontroller effects a change in the toy e.g.sound or light pattern changes.The microcontroller uses PWM to adjust brightness and changes the colours based on button presses or app commands. The microcontroller triggers the DFPlayer Mini to play specific sound files stored on an SD card.

CLOUD, SOFTWARE, FIRMWARE AND DATA

AWS & CLOUD COMPONENTS

The toy is integrated with Amazon Web Services for remote control, storage and analytics:

- **AWS IoT Core:** it facilitates the toy's communication via **MQTT protocol**, managing updates and control commands. It receives status updates and control commands from the toy and mobile app respectively. The toy sends its status (e.g., light on/off, music) to AWS IoT Core via MQTT.
- **AWS Lambda:** Triggers updates in **device shadow**, manages database entries and activates firmware updates stored in **AWS S3**. Lambda can be triggered by a user interacting with the toy or app, this will lead to an update in the device shadow, update in databases or notifications to users. AWS Lambda is well suited for these functions because it is serverless, so, no need to manage servers. The user controls the toy via the mobile app, sending commands to AWS IoT Core.
- **AWS IoT Device Shadow Service:** Stores the toy's last known state, ensuring a seamless experience when powered on.
- **Databases:**

  - DynamoDB: it stores settings and device status in a NoSQL, highly available database. Ideal for storing dynamic, fast-access data like user preferences and device states. It offers millisecond response times and is serverless, making it perfect for storing device state and app data.
  - Amazon RDS: it is a relational database that can be used to store user account information or more structured data like analytics or device history logs. It is great for relational data and structured reports.

- **Amazon Cognito:** it facilitates account creation, login password creation and reset, user authentication for both the mobile app and the web portal. It is secure, scalable and integrates well with other AWS platforms and services like Lambda, API gateway etc. It securely manages authentication for both the mobile app and the web portal.

- **AWS IoT Events:** it allows users to detect and respond to changes or patterns in toy data and automates actions when necessary. It helps monitor, analyse, and act on changes in data streams from toy and app, enabling automation and proactive responses to problems. It enables monitoring of the toy for failures or changes and to trigger actions when such events occur.
- **AWS IoT Analytics:** this enables the gathering of data about the toy's usage, this can be used to generate reports for data analysis and insight into the toy's performance and user interaction. It processes toy usage data to generate reports on performance and user interactions.
- **AWSS3(Simple Storage Service):** It functions as a firmware storage system. When an update on this firmware is required, the toy will download the latest firmware from S3. It is secure, affordable, scalable and great for large firmware files. It integrates well with other AWS services like Lambda and IoT Core.
- **Amazon API Gateway:** it provides a REST API for mobile apps, offering authentication and authorization mechanisms while enforcing rate limiting and security measures to protect user data and ensure optimal performance.
  Route: App → API Gateway → Lambda → IoT Core → Toy
- **Amazon CloudWatch:** provides monitoring and logging possibilities, eases system health alerting and tracks performance metrics to ensure optimal operation and reliability of cloud-based applications and services.

SOFTWARE & FIRMWARE

AWS S3 and AWS Lambda ensures that software updates go on without a hitch or security glitch, by managing the process. ***This is how it happens:***

AWS S3 and AWS Lambda ensure secure and seamless firmware updates by managing the process from verification to installation. When the toy detects an available update, AWS Lambda checks the latest firmware version in AWS S3, authorizing the download if necessary. The toy securely retrieves the firmware via HTTPS, verifies its integrity with digital signatures and installs it while AWS Lambda oversees authentication and security measures. The firmware, built on FreeRTOS, supports WiFi, MQTT communication and local state management, ensuring responsiveness and efficient task scheduling. OTA mechanisms enable smooth updates, with AWS IoT Core and Lambda managing distribution. Once installed, the ESP32 reboots to apply the new firmware while CloudWatch logs update status for monitoring.

DATA

The toy's performance relies on collecting accurate data at the right time by tracking user preferences such as light settings, sound choices and usage patterns while ensuring smart sensor readings through well-timed intervals for precision and adapting data collection frequency based on user activity to maintain efficiency and responsiveness without unnecessary updates. This way, the toy stays responsive, efficient and perfectly tuned to the user's experience.

DATA FLOW

The user presses a button on the **toy** or via the **toy’s mobile app** to control features like sound or light. This input is detected by the **ESP32** microcontroller, it then updates the device state and prepares the status data for communication. The toy then sends updates like light brightness for example to **AWS IoT Core** via **MQTT**, this ensures real-time synchronization.

The toy's latest settings are then recorded in the **device shadow**, this ensures that any preferences or settings on the toy are maintained even if the toy is switched off and restarted. **AWS Lambda** processes commands from users or scheduled events, like firmware updates or status changes. The toy preferences, statuses, and user settings are securely stored securely in **Amazon DynamoDB** and structured analytics are stored in **Amazon RDS**. The mobile app allows users to send commands and inputs, collect toy data and manage their settings via **AWS IoT Core** hitch-free. Administrators’ portal access reports and analytics via a secure portal, providing insight into device usage and performance trends.

ACTION	FROM	TO	DATA FORMAT	END-GOAL
Toy or app usage	Toy or App	Microcontroller		to capture the input
Device status update	Toy	AWS IoT Core	JSON	to report the state
App usage	App	Toy (via AWS)	JSON	to control the toy
Analytics	AWS IoT Core	Admin Portal	JSON/SQL	usage stats, & diagnostics
Firmware update check	Toy	AWS S3	Binary	to download new firmware

DATA COMMUNICATION

Toy-to-Cloud Communication: to make sure that the whole operation goes seamlessly, the toy exchanges data with the cloud using JSON, it's clear and flexible makes data transmission very efficient. Ambient light levels are captured using floating-point values (e.g., 40.2), enabling subtle variations in readings, while light brightness and sound volume follow a 0–255 scale for intuitive control.

DATA TYPE	TRANSMISSION FREQUENCY	DESCRIPTION
State Changes	Immediate (event-driven via MQTT)	Reports real-time interactions, such as button presses or app commands.

Heartbeat Signal	Every 60 seconds	Performs routine health checks to maintain connectivity and system integrity.
Usage Statistics	Every 15 minutes	Sends aggregated data for monitoring and performance analysis.
Firmware Check	At startup and every ~6 hours	Ensures software updates occur smoothly without disrupting functionality.

The toy maintains communication with the cloud using the MQTT protocol over TLS, to ensure secure and reliable transmission. Whenever a state change occurs, for example, a button press or command from the app, the update is sent immediately. In addition, a heartbeat signal is transmitted every 60 seconds, ensuring the system remains connected and operational, while usage statistics are logged every 15 minutes to facilitate monitoring and analytics. To keep the device running efficiently, firmware updates are checked at startup and at scheduled intervals (approximately 6 hours), ensuring the software stays current without causing interruptions.

App-to-Toy Interaction via Cloud: the app communicates with the toy through the cloud by sending requests to an API Gateway, which triggers an AWS Lambda function to process the request and interact with AWS IoT Core. This process ensures commands and updates are delivered seamlessly, optimizing responsiveness and efficiency. The system balances precision and timing to maintain an adaptive, fluid experience that responds dynamically to user input.

Toy-Cloud Data Communication (JSON):

```
{
 "device_id": "toy1234",
 "timestamp": "2025-06-10T12:50:00Z",
 "data": {
  "state_changes": {
   "button_pressed": true,
   "light_level": 40.2,
   "sound_volume": 128
  },
  "heartbeat_signal": {
   "status": "online",
   "battery_level": 85
  },
  "usage_statistics": {
   "play_time_minutes": 45,
   "interaction_count": 120
  },
  "firmware_check": {
   "version": "1.3.5",
   "last_update": "2025-06-10T06:50:00Z",
   "update_available": false
  }
 }
}
```

App-Toy interaction via the cloud(JSON):

```
{
 "app_request": {
  "device_id": "toy1234",
  "timestamp": "2025-06-10T12:55:00Z",
  "command": {
   "action": "change_light",
   "parameters": {
    "brightness": 180,
    "color": "blue"
   }
  }
 },
 "cloud_processing": {
  "api_gateway": "AWS API Gateway",
  "lambda_function":
"ProcessToyCommand",
  "iot_core": "AWS IoT Core"
 },
 "toy_response": {
  "status": "success",
  "updated_values": {
   "light_brightness": 180,
   "light_color": "blue"
  }
 }
}
```

REFERENCES

- Microcontrollers for IoT | Medium
- DFPlayer Mini Mp3 Player - DFRobot Wiki
- IC in Addressable LED Strip Lights: Exploring Its Functionality and Common Models
- Azure IoT – Internet of Things Platform | Microsoft Azure
- Arduino JSON for IoT
- IoT Architecture: Six Levels, Core Components and Use Cases
- Precision Counts in IoT | Analog Devices
- IoT Toys: A New Vector for Cyber Attacks | Apriorit
- 20190903_DC_109_IoT_Production-to-Product_Report_Digital_1___1_.pdf
- Understanding the JSON Data Format for Industrial Control Architecture - Technical Articles
- **Youtube**