CSCI-B 649: Applied Distributed Systems

FaaS on IoT

Anisha Bajaj, Ankush Pathak, Mohit Dalvi, Shreyas Vaidya, Shubham Bipin Kumar

1. Background

AWS IoT Greengrass is a cloud service and open source software that helps to develop and manage IoT applications on user's devices. It allows them to process data generated locally, apply machine learning models, and filter and combine device data.

With AWS IoT Greengrass,

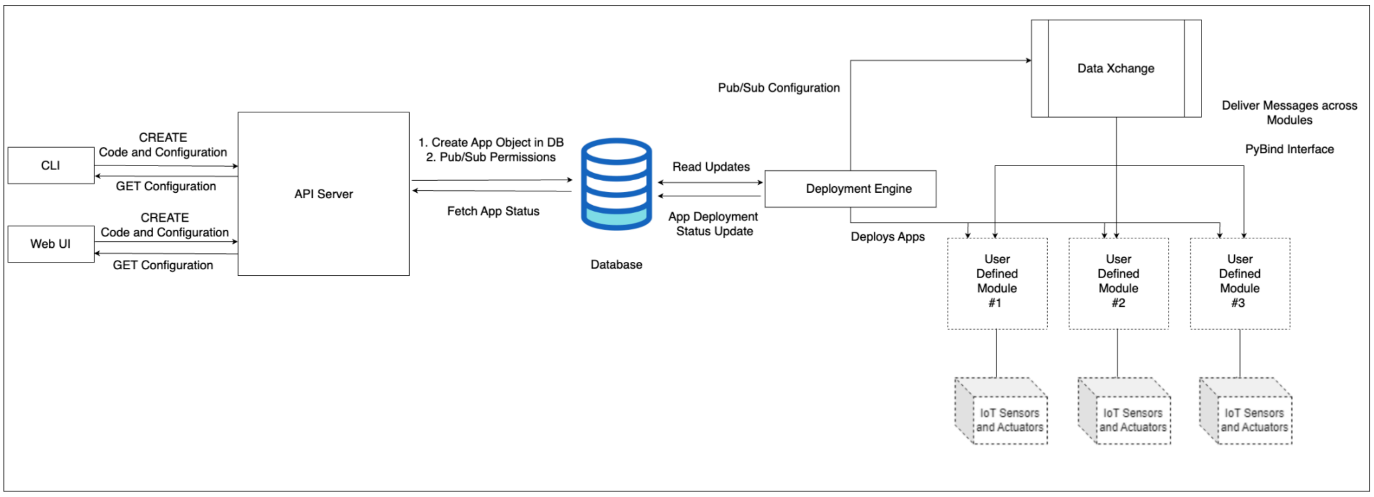
- Devices can gather and analyse data closer to where it's generated, react automatically to local events, and communicate securely with other devices on the local network.
- Devices can communicate securely with AWS IoT Core and send IoT data to the AWS Cloud.
- Edge applications can be created by using pre-built software modules, called components, that can connect edge devices to AWS services or third-party services.
- Lambda functions, Docker containers, native operating system processes, or custom runtimes can be packaged and operated.

Even though Greengrass provides multiple services as above, It has certain limitations such as:

- It's primarily designed and optimised to work with AWS platforms and services.
- Greengrass is compatible with a variety of platforms such as Windows, Linux, X86 and ARM based processors. However, currently it does not support Microcontroller platforms such as Arduino. Furthermore, it does not support microkernels such as L4, Xinu.
- Lastly, though Greengrass is not proprietary, it's tightly coupled with the AWS ecosystem, which is proprietary.

Thus, we aimed to design a solution that can address some of these limitations of AWS Greengrass. The primary objective of our work is to create an alternative platform for IoT devices edge computing that supports different platforms and provides flexibility for using microkernel architectures.

2. Overall Design



3. Components

3.1. User Interface

Users will be provided GUI and CLI to select/ input configuration details(functions to be deployed, device information, dependency information).

Configuration Specification will be in following format:

- Name
- Version
- Resources
    - Subscription topics
    - Publication topics
- Runtime environment: python(planned) , Java, C++(Future work)
- Required Platform: Linux
- Commands to run:
    - Mode: preserved flat to indicate command execution privilege level
    - Commands to execute
    - Wait for exit: Wait for command to exit before processing next command in the list
- Dependencies
    - Requirements.txt for Python runtime

Tech Stack: Javascript (Node.js)

3.2. Command Line Interface

CLI interface supports the Create and Get commands and based on input from the user CLI commands and their args:

a. Create:
   i. Artifact
   ii. Configuration
b. Get
   i. Configuration

Tech Stack: Golang

3.3. API Server

API server will provide functionality to verify and store the artifact and configuration via GUI and CLI In database. The Configuration will be stored as a deserialized object.

a. Accept artifacts and configuration.
b. Load configuration as an object (deserialize into an object)
c. The sanity of configuration and artifact
   i. Check if files referenced in the configuration exist.
   ii. Other sanity checks
d. Put into platform deployment engine’s queue

Tech Stack: Spring Boot (Java)

3.4. Deployment Engine

Check database periodically to see if there are any new entries based on isProcessed flag and get the artifact location of each entries to be processed and performs below steps:

a. Initiate deployment process for given artifact
b. Check for the records in the database and update the flags accordingly.
c. Create directory structure for the deployment.
d. Create an isolated directory environment
e. Move the user artifact to the appropriate location in the created directory structure.
f. Communicate with Data Xchange to load pub/sub permission of the current module in its memory.
g. Perform deployment based on the listed user commands.
h. Update the database status as per stated business logic.

Tech Stack: Python

3.5. Database

Database interacts with API server, Deployment Engine and Data Xchange.

There are primarily two tables called application and commands.
The applications table has *name, version, runtime Environment, required platform, isContainerized, dependencies, subsTopic, pubsTopic, commands, status, deploymentEngineReported status, isProcessed, created, updated, extracted path* columns.

The commands table has *mode, execCommands, waitForExit, application_id* columns.

- These tables are created during API server startup and tables are populated with the user requests.
- The deployment engine initially checks the application table on startup and gets all the entries based on the isProcessed flag. This flag tells if the request is processed or not.
- At all steps, the deployment engine updates the status in the application table.

The API server can fetch the deployment engine status from the Database.

Tech Stack: PostgreSQL

3.6. Data Xchange

It takes care of reading pub/sub permissions and topic permissions, if the permissions checks then it pushes messages to all subscribers.

- Creates linux message queues for publishers and subscribers
- Moves messages across queue to deliver messages published on a topic by publishers to respective subscribers
- The underlying IPC mechanism is linux message queue, but that part can be swapped out to use a different IPC mechanism
- Publishers and subscribers have to register with Data Xchange before using its services
- Deployment engine communicates a permission list to the Data Xchange. This list includes a list of topics that an application can subscribe or publish to.
- We have also implemented a Python API using Pybind to allow Python applications to interface with Data Xchange and use its services

To ensure Deployment Engine and Data Xchange are working as expected, we used the following Test function:

- Client-server architecture to compute roots of given polynomial function.
- Client publishes the request to compute roots to a specific topic along with the coefficients.
- Server subscribes to the same topic as published by client, to fetch the request
- Server computes the roots and publishes the result to the same topic.
- All the communication via message queue wrapper class and data exchange.

Tech Stack: C++

3.7. Potential Future Deployment Platforms

L4Re

- L4Re is an application environment on top of the L4 microkernel
- L4Re has some barebones services for memory and task management
- No driver availability for networking
- L4Re allows creating tasks within an application through pthread. Also provides some IPC mechanisms
- Overall the system has a relatively small memory and compute footprint, ideal for edge processing applications, but lack of networking services is a critical downside.
- We developed a simple client server RPC application as a proof-of-concept and were able to get it to work.

Xinu

- Another simple barebones OS platform
- We have verified that Xinu has some networking support. Has networking support on qemu-arm and BeagleBoneBlack, no other platform.
- Further work required to dynamically load application code into Xinu and run edge computing applications

Tech Stack: L4Re and Xinu

4. Timeline

Timeframe	Progress
1 Feb - 15 Feb 2023	Brainstorming of potential ideas
16 Feb - 28 Feb 2023	Background Research and Initial Design
1 Mar - 15 Mar 2023	Data Xchange Development; Exploring L4Re as Potential Deployment Platform
16 Mar - 31 Mar 2023	Deployment Engine Development; Simple client server RPC application on L4Re
1 Apr - 15 Apr 2023	API Server, CLI and GUI Development; Exploring Xinu for Deployment Platform
16 Apr - 30 Apr 2023	Integration and Testing of all components

5. Conclusion and Future Scope

We have implemented a FaaS on IoT platform that is easy to build and deploy on platforms like the Raspberry Pi. Our platform provides simple interfaces in the form of a web UI and CLI to deploy and manage code remotely.

L4Re and Xinu are potential platforms that can be explored to deploy and load code dynamically through the interface implemented correctly. This would require engineering runtime code loading through an external source on L4Re and Xinu. Furthermore, applications that require networking capabilities are feasible only on Xinu right now.