

Semester Project

Course: Connected Devices

Instructor: Andrew King

an.king@northeastern.edu

About This Document

This document provides a detailed overview of the Semester Project for Connected Devices.

As with all previous Lab Modules, the Semester Project assumes the student is already familiar with Python, Java, and the various messaging formats, IoT protocols, and cloud services discussed during class. This project provides a framework by which the student can aggregate their assignments and knowledge into a large, integrated project.

All labs are expected to be collaborative, where students can work together as needed to accomplish the task at hand. However, each student **MUST** do their own unique work.

Your work for the Semester Project must be turned in according to the specified rubric referenced in Blackboard.

Semester Project

Build an end-to-end IoT solution using the concepts learned through class lectures and Lab Module assignments

Summary: This module will focus on developing an IoT solution that connects an IoT constrained device sensor to an IoT smart device (gateway), which in turn communicates with a cloud service to store data, perform some analytics function, and return a result that the gateway can use to signal the device's actuation.

Lecture References:

- Presentation Slides: Connected Devices - Weeks #1 - #10
- Presentation Video: N/A

Online References:

- See previous Lab Modules

Purpose: This lab module has three objectives:

- 1) Learn how to design an integrated IoT solution from a problem statement / use case you define.
- 2) Learn how to use existing functionality and solutions to implement your design.
- 3) [Possibly] Learn how to integrate your solution with another student's using common protocols and data formats.

Create an end-to-end IoT solution

Pre-requisites

1. Access to your own cloud-based Git source code repository
 - a. There are many options available for creating and managing an online Git-based source code repository. As described during Lecture #4, BitBucket setup / configuration has been described as one such option.
2. Java experience
 - a. As indicated in the course pre-requisites, you should already have a solid handle on Java development in general, and associated object-oriented design and implementation concepts. Experience with callbacks, data structures, and IP-based communications paradigms is also very helpful.
3. Python experience
 - a. As indicated in the course pre-requisites, you should already have a solid handle on Python development in general. It would also be helpful if you understand object-oriented design and implementation concepts, multi-threaded programming, callbacks, data structures, and IP-based communication paradigms.
4. Cloud API experience
 - a. Using knowledge gleaned from in-class lectures and Lab Modules, you should be able to integrate a simple software application with a cloud service provider's MQTT (or other) broker for communicating bi-directionally with your IoT gateway.
5. WireShark (or similar protocol analyzer) experience
 - a. You should be familiar with WireShark or another similar protocol analyzer for your platform: <https://www.wireshark.org/download.html>
 - b. You may also need to read the following if you run a server locally: <https://wiki.wireshark.org/CaptureSetup/Loopback>
6. Successful completion of Lab Modules #1 - #8.

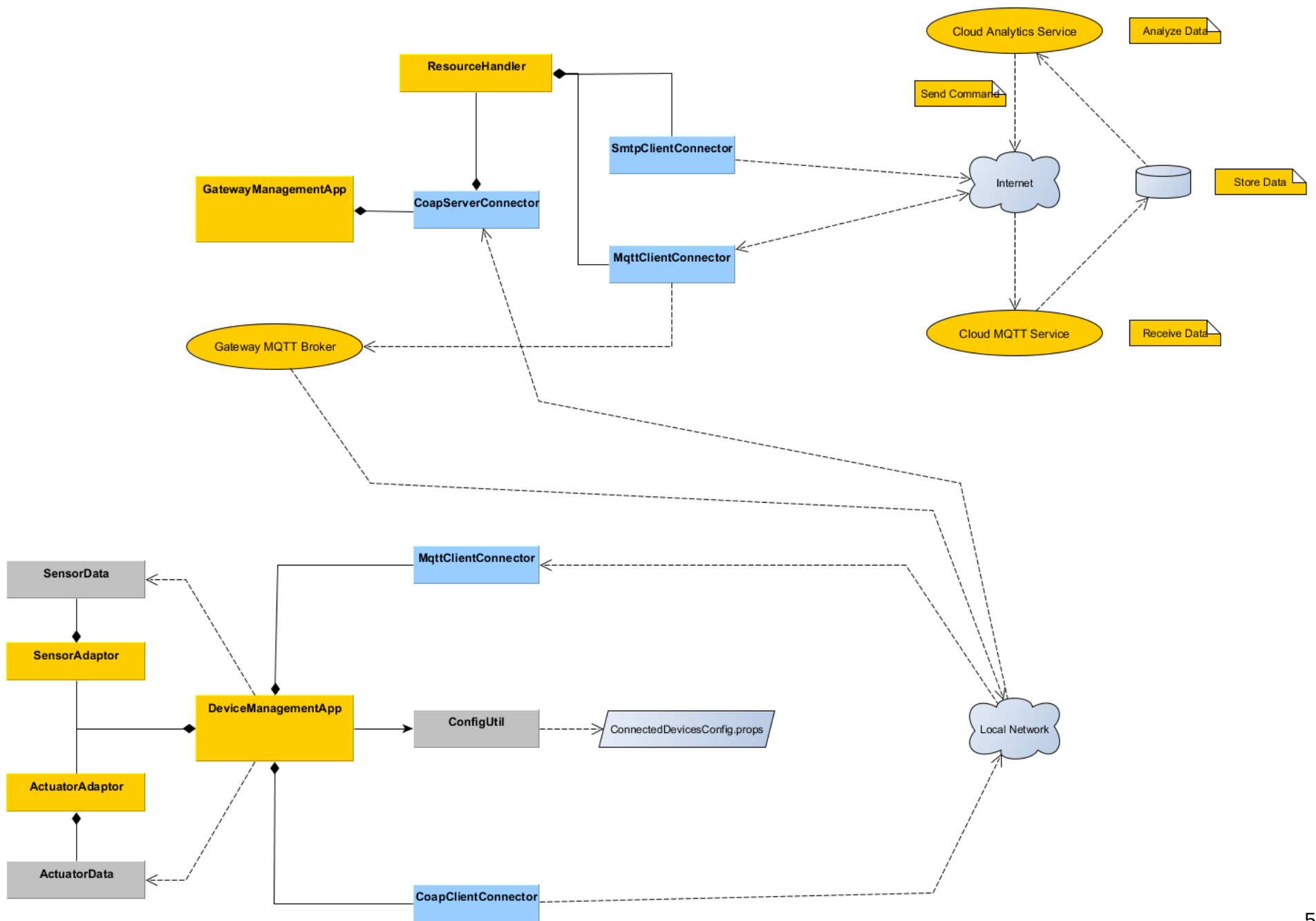
Summary


1. Objectives

- a. Create an app that runs on a “constrained” IoT device which can process sensor data and trigger an actuator.
 - i. The app must also be able to communicate with an IoT gateway using one of the protocols discussed in class.
- b. Create an app that runs on a “smart” IoT device (gateway) which can interact with the “constrained” device over the local network using one of the protocols discussed in class.
 - i. The gateway app must also be able to interact with a remote cloud service using their IoT API (or one of the protocols discussed in class, such as MQTT), and use one or more services to perform some analytics on the data from the gateway.
 - ii. The gateway must be able to process the cloud-derived analytics data to trigger an actuation event on the “constrained” device.

2. Requirements

- a. The constrained device app can be written in Python or Java
- b. The constrained device must interact with at least one sensor, and at least one actuator (the actuator can be an LED display, etc.)
- c. The gateway device app can be written in Python or Java
- d. You may use multiple protocols to communicate between cloud, gateway device and constrained device; however, at least one of these protocols must be either CoAP and / or MQTT
- e. You must communicate with a cloud-based IoT service (e.g. Ubidots, Amazon IoT, etc.), and capture data on at least three different variables (e.g. one for a sensor, one that may be derived, and one for an actuator)
 - i. NOTE: You may use any appropriate means available to communicate with the cloud-based IoT service (e.g. MQTT, WebSockets, HTTP, vendor-specific API, etc.)
- f. You must send an e-mail or SMS to yourself after the gateway processes an actuation event - this can be before or after the cloud service is invoked
- g. You must be able to show a demonstration (video or live is acceptable) of the end-to-end functionality of your system
- h. You may choose any appropriate IoT use case provided it adheres to University guidelines and meets the specified requirements
- i. You may work with up to two team members (three people total), but each must do their own work, provide a unique design and implementation contribution, and showcase their own unique use case
- j. A notional design is provided on the next page. You MUST create your own design, although you may base it on the one provided.












The applications and their respective components will probably look similar to the diagram depicted on the previous page; however, you may create your own design provided it meets the requirements specified. You must also create your own diagram depicting your architecture.

Evaluation

1. How to submit the assignment
 - a. Follow the instructions specified in the Semester Project Assignment
2. Grading
 - a. Points will be provided based on the rubric documented as part of the assignment and given on the following page.
 - b. **IMPORTANT:** If you plagiarize or otherwise do NOT do your own work (e.g. someone else implements your code), you will get 0 points for this assignment.

Criteria	Levels of Achievement		
	Novice	Competent	Proficient
Maintainability  Weight 20.00%	50.00 % Code and design artifacts can be read, but is inconsistently formatted / spaced / declared, comments are lacking, and flow is difficult to understand and / or doesn't align with standardized coding practices. Any documentation provided is obviously copied / pasted from someone else's code.	75.00 % Code and design artifacts are reasonably well written with a few formatting inconsistencies. Each method / function is documented. Majority of comments are in place and make sense, and code mostly aligns with standardized coding practices.	100.00 % Code and design artifacts are well written and formatted consistently throughout all modules. Each method / function is documented, and complex functionality is documented within each method / function. Comments / documentation are clear and accurate. Code is stylistically correct and consistent. Code aligns with standardized coding practices.
Timeliness  Weight 10.00%	50.00 % Within 24 hrs of due date / time.	75.00 % Within 1 hr of due date / time.	100.00 % On time (or before) due date / time.
Data Processing  Weight 20.00%	50.00 % System was implemented and functions, but less than 5 minutes of sample data was collected and processed.	75.00 % System was implemented and functions, but less than 1 hour of sample data was collected and processed.	100.00 % System was implemented and functions, and at least 2 hours of sample data was collected and processed.
Cloud Processing  Weight 15.00%	50.00 % At least 1 cloud variable was implemented correctly and is used by the system.	75.00 % At least 2 cloud variables were implemented correctly and are used by the system.	100.00 % At least 3 cloud variables were implemented correctly and are used by the system.
Protocol Implementation  Weight 15.00%	50.00 % 1 protocol is used / implemented correctly (e.g. MQTT, CoAP, HTTP, SMTP).	75.00 % 2 protocols are used / implemented correctly (e.g. MQTT, CoAP, HTTP, SMTP).	100.00 % 3 protocols are used / implemented correctly (e.g. MQTT, CoAP, HTTP, SMTP).
Design  Weight 10.00%	0.00 % Design diagram is not your own creation or does not accurately represent your implementation.	50.00 % Design diagram is your own creation, but is not 100% in sync with your implementation.	100.00 % Design diagram is your own creation, and 100% accurately represents your implementation.
Documentation  Weight 10.00%	0.00 % There is no write-up describing the purpose and approach of your project.	50.00 % You've provided a short write-up of at least two sentences describing the purpose and technical approach of your project.	100.00 % You've provided a write-up of at least four sentences describing the purpose, technical approach, challenges, and accomplishments of your project.