/

SOLUTION PROPOSAL DOCUMENT SSDCS TEAM III (APRIL 2022)

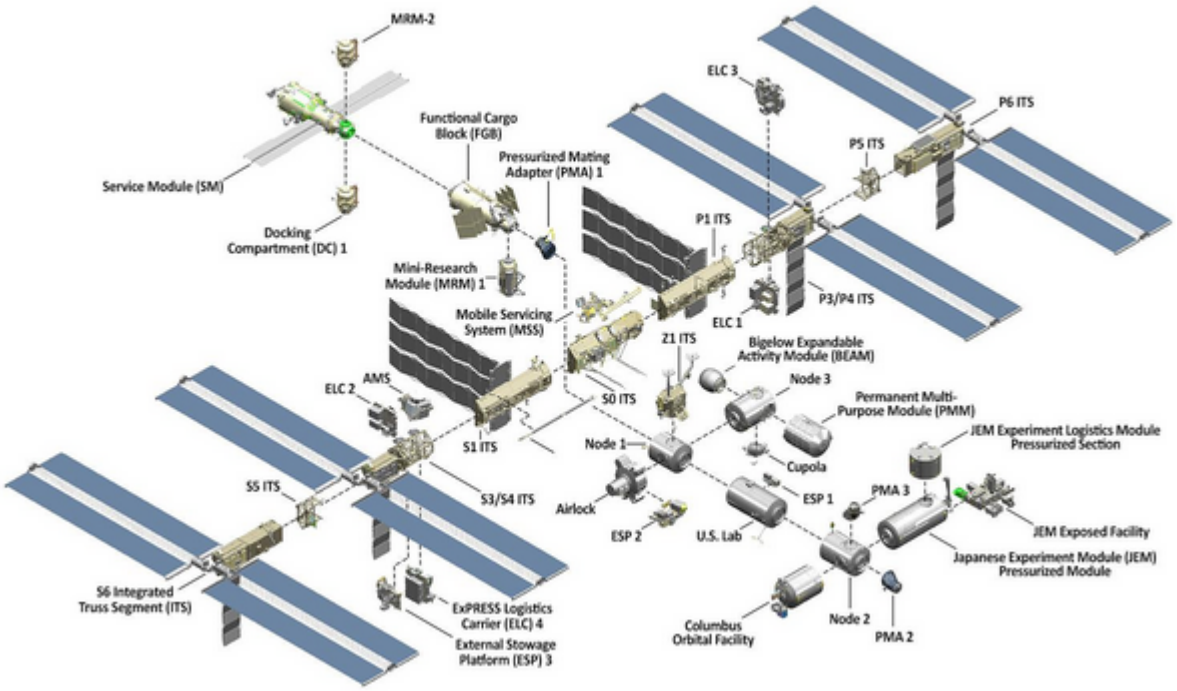
Aidan Curley Ian Wolloff & Kaoru Kitamura
University of Essex Secure Software Design Module

Table of Contents

SSDCS GROUP 3
ISS DOMAIN

Problem Statement

Figure 1: Diagram of the International Space Station



(NASA 2021)

Due to limited physical access to the structure and modules aboard the ISS, data from the station's sensors need to be transmitted to mission control permitting ground support services to monitor the condition of the station. Sensors include:

Table 1: Sensors on board the ISS

Sensor	Measurement
Oxygen	Percentage Atmospheric O2
Radiation	Picocuries (pCi)
Fuel	Lbs
Carbon Dioxide	Percentage Atmospheric CO2
Water	Liters
Gravity	Meters Per Second Squared
Photovoltaic Voltage	Volts
Airlock Status	Pascals (Pa)

*[Malesky Mallory] [*Stenzel C*]*

We propose to build a data-hub to collect data from the sensors which is stored locally and then transmitted to ground control in a single burst transmission as radio bandwidth is limited. [**Appendix 1**]

System Requirements

Table 2: System Requirements

Description	Priority
Data at rest is secured using encryption	High
Data in transit is encrypted using SSL certificates	High
Logon actions are recorded in external repository	Medium

Functional Requirements

Table 3: Functional Requirements

Description	Priority
Sensor data is stored in a DBMS	High
Data is stored in line with NASA retention policies and GDPR	High
Secure REST web services are used for consumption and integration	High
Solution scales to meet demand profile	Medium
System is fault tolerant and failure of any node does not cause outage	High

Non-Functional Requirements

Table 4: Non-Functional Requirements

Description	Priority
Linux OS used for improved reliability	Medium

(Anthony, 2013) (OpenSCAP, 2022)

Assumptions

We have made a number of assumptions about the system:

Table 5: Assumptions

Communication takes place over TCP/IP
Sensors are online and provide accurate readings
System operates 24/7 due to its nature as a life-support system

Security Risks & Mitigations

Due to the life-supporting nature of the data, it is important that our solution addresses the security triad of confidentiality, integrity, and availability. We have used the OWASP Proactive Control methodology to identify potential risks and their mitigations (OWASP, n.d.).

C1 Define Security Requirements

“Effective security of network and information systems should be driven by organisational management and corresponding policies and practices.” (NCCS, 2021)
We secure access to our system by:

1. Log Analysis — checking who is attempting to access the system and locking out unverified users.
2. Firewalls and network monitoring — blocking access to various parts of the network

C2 Leverage Security Frameworks and Libraries

We use Github’s Dependabot service to identify new versions of libraries we leverage, and automatically update our development branch. Any known vulnerabilities for libraries and tools we use are listed on page 17. Additionally static code analysis using Bandit and Sonarqube is conducted to identify potential security issues during development.

C3 Secure Database Access

Data integrity is enforced through three methods:

1. **Access restricted secure API.** HTTPS requests include a custom header value X-**ACCESS-TOKEN** containing a JWT encoded token (Jones, 2015). Requests without this token are refused and flagged as suspicious.

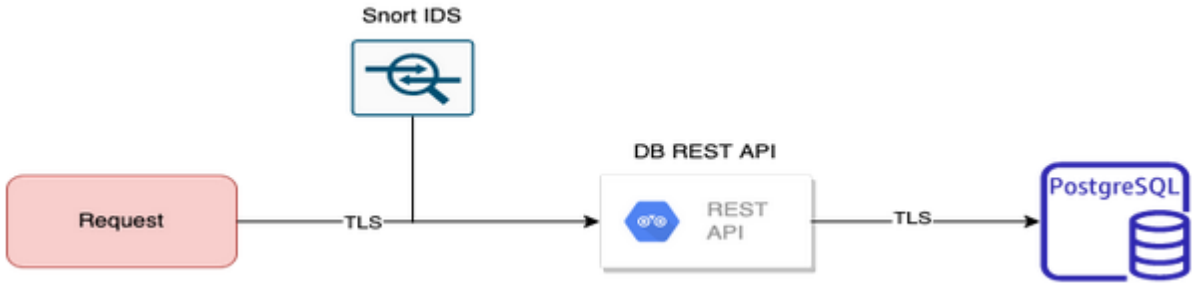
Figure 2: Sample JWT Token

Response Body Display Options


```
{
    "Token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NDgwNjY2NTksImlhdCI6MTY0ODA2MzA1OSwidXNlciI6Ik5BU0EifQ.KtDiVTkvMKlrAhPmJtFSUobz6whR4V31E8vh5oVsnNk"
}
```


2. **IP address restrictions.** We limit access to a pool of permitted addresses defined in the Web Application Firewall that sits between our CDN and API endpoints.
3. **Network monitoring**. Requests are completed only if the requesting IP is on our IP address white list, stored in the application. We use Snort intrusion detection system to monitor network traffic and alert us to any suspicious behaviour.

Figure 3: Network monitoring using an Intrusion Detection System



C4 Encode and Escape Data

Client and server data is checked at each step in the system through the use of features like escaping single quotes to prevent SQL injection attacks.

C5 Validate All Inputs

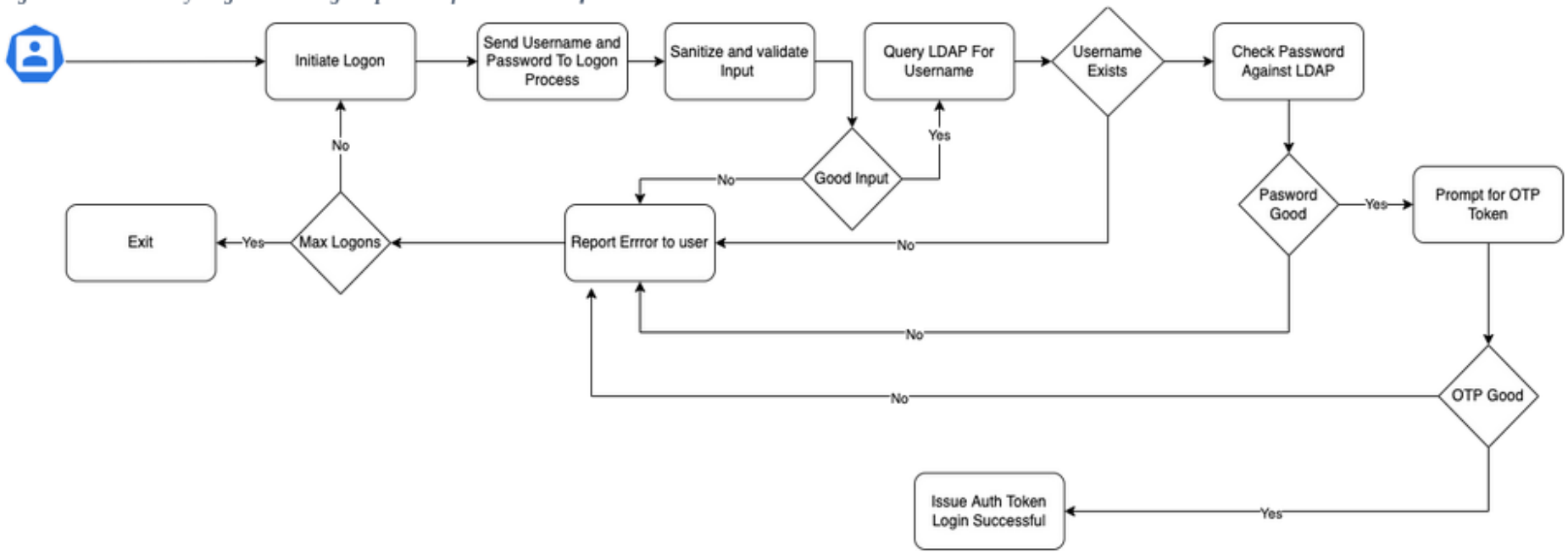
All user inputs are validated by the application using a series of Regex checks to ensure that the input is in the expected format.

C6 Implement Digital Identity

Accounts are stored in an LDAP Database. Identity is verified using passwords enforced by system group policies, ensuring passwords meet best practice in complexity and prevents users from utilising known common passwords.

Passwords are only one step in securing an account. Microsoft recently found that implementing MFA increases an account's security by 99.9% (Weinert, 2019). We use the python **PY-OTP** library to provide a OTP code, as described in the sequence diagram in Figure 4 overleaf.

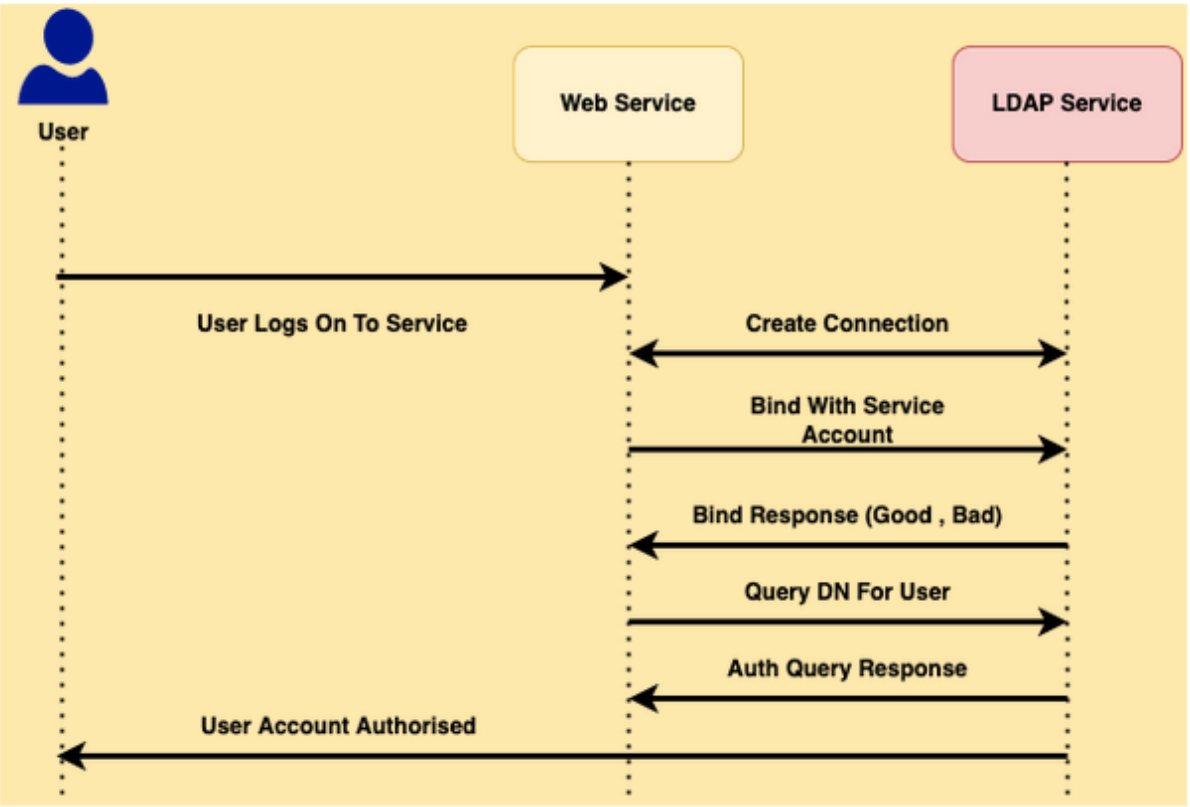
Figure 4: UML activity diagram showing the process of authentication



C7 Enforce Access Controls

User security is provided by separation between the application tier and the security tier. User accounts are stored in an LDAP Database with security GPOs applied defining access rights and account lockout policies. The process of authentication is shown in Figure 5. The Linux infrastructure is configured with fail2ban software to lock access if non authorised access is attempted.

Figure 5: Authentication Sequence Diagram



C8 Protect Data Everywhere

Data protection Is implemented by two main tenants,

1. End-to-end encryption via SSL certificates, securing all data while in transit.
2. Encrypted-at-rest model. Data is stored in an encrypted state when not being transferred across the network.

C9 Implement Security Logging and Monitoring

Security logging is a three part process:

1. VM Hosts forward security events to a central server for alerting and analysis.
2. Network Traffic is monitored by an IDS to identify security threats.
3. FIM (File Integrity Monitoring) is enabled on key system files.

C10 Handle all error and Exceptions

Complete exception handling is provided by the software, and tested by unit tests and user acceptance tests. The microservice design ensures that service is maintained by another pod taking over from any failed instance.

Proposed System / Design & Architecture

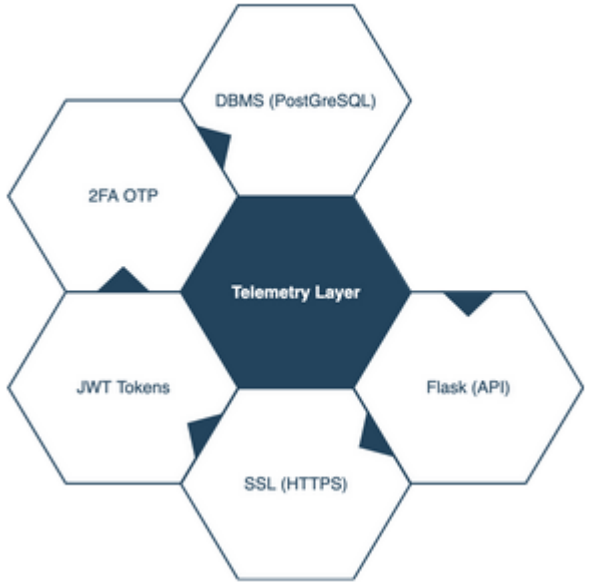
Our “High Availability” solution, once deployed, cannot be taken offline by the failure of a single component or service. We deliver a robust mechanism for the capturing and analysis of data, using redundant clustered microservices and fault tolerant storage.

Failover technology ensures 24/7 availability. Services consist of three functional nodes permitting for the failure of any two nodes without affecting delivery.

Ground Control Telemetry Data Layer (GCTDL)

Responsible for replication of data captured from ISS sensors and stored on ISS local storage, this layer enables the download of data to mission control, communicating with the ISS via fully authenticated and encrypted REST API’s.

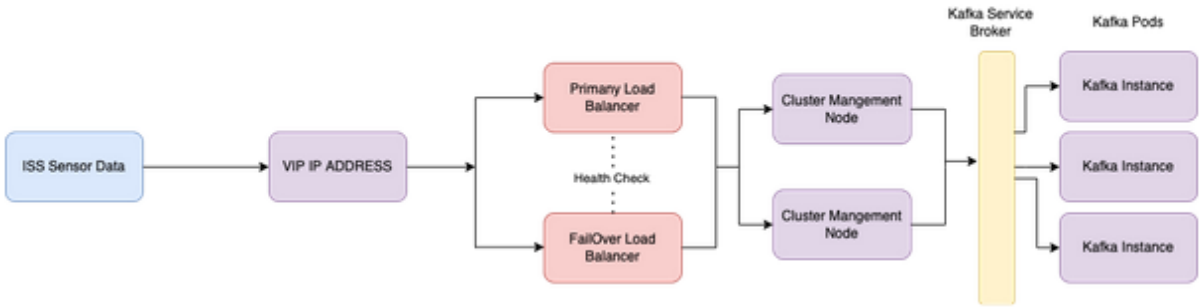
Figure 6: Overview of the technology used in Ground Control Layer



Space platform data collection layer (SPCL)

Data from the ISS sensors is ingested by a Kafka message queue service. The data is analysed based on a defined rule set. Should the data fall outside the expected range for the sensor type, mission control is alerted. The data is archived to a local DBMS.

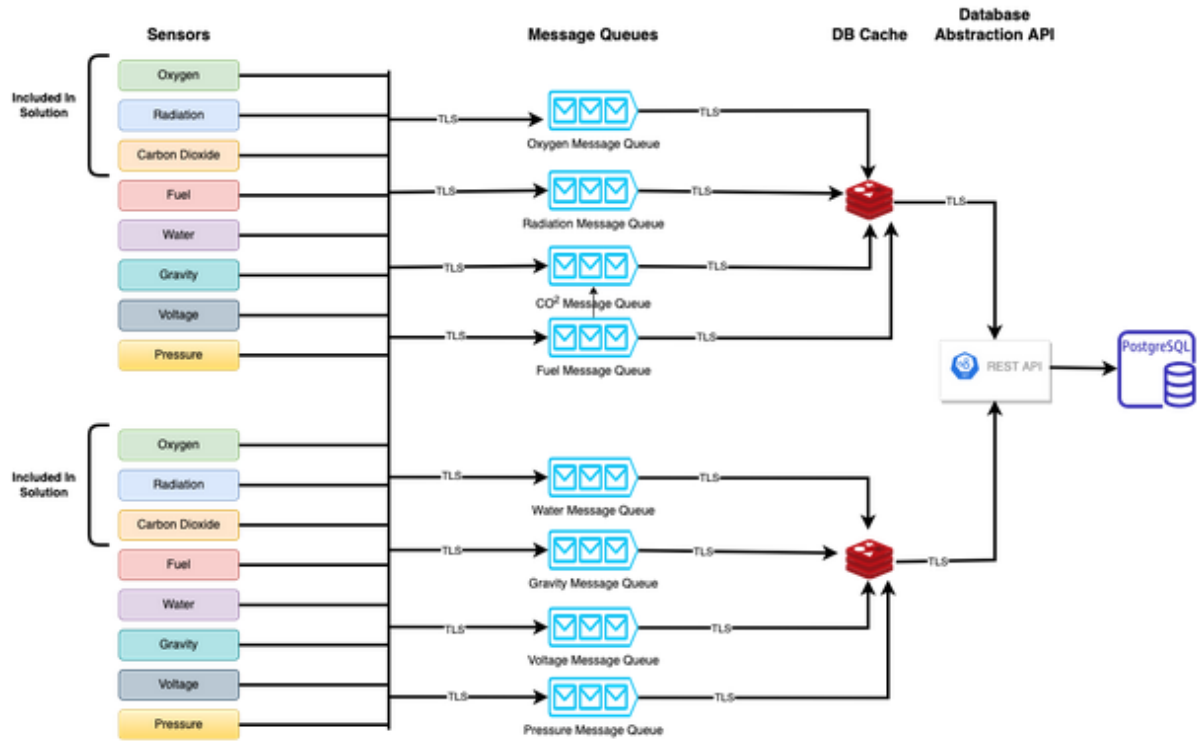
Figure 7: High Availability Data Logging System Overview



High Level Design Overview Space Side

The Data Collection layer is designed using container architecture to allow for maximum horizontal scaling of the solution (Nguyen et al., 2020).

Figure 8: Overview of the connection between the sensors and the database



Command and Control Layer

Permits ground control to change parameters and toggle sensors through the use of APIs. The functionality is detailed in table 6 and an overview of the system is described in Figure 8a.

Table 6: Command and Control Layer Required Functionality

Update of system parameters located on the ISS
Download of Data from ISS sensor Array
Removal of historic data for GDPR compliance purposes

Figure 8a: Overview of the Command & Control Data Replication Layer

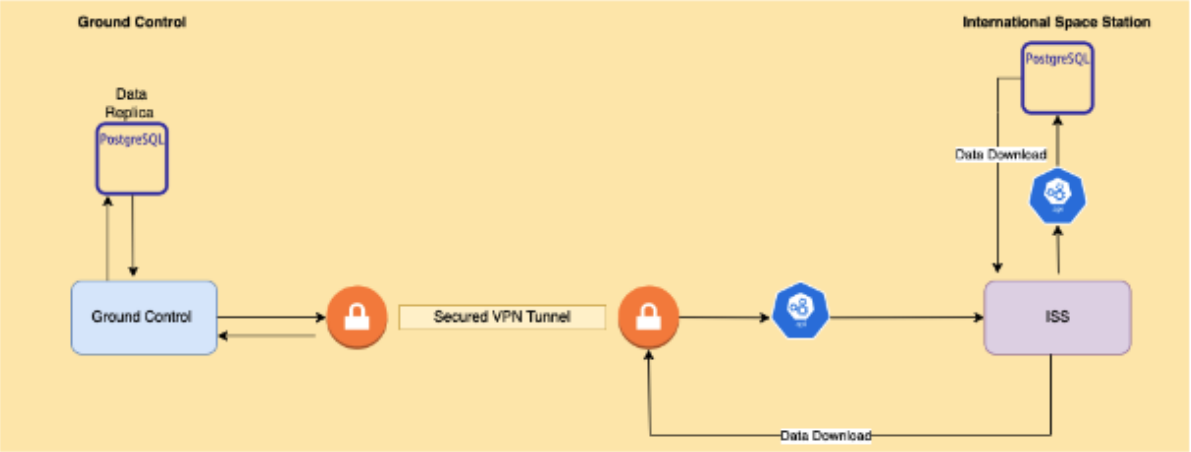
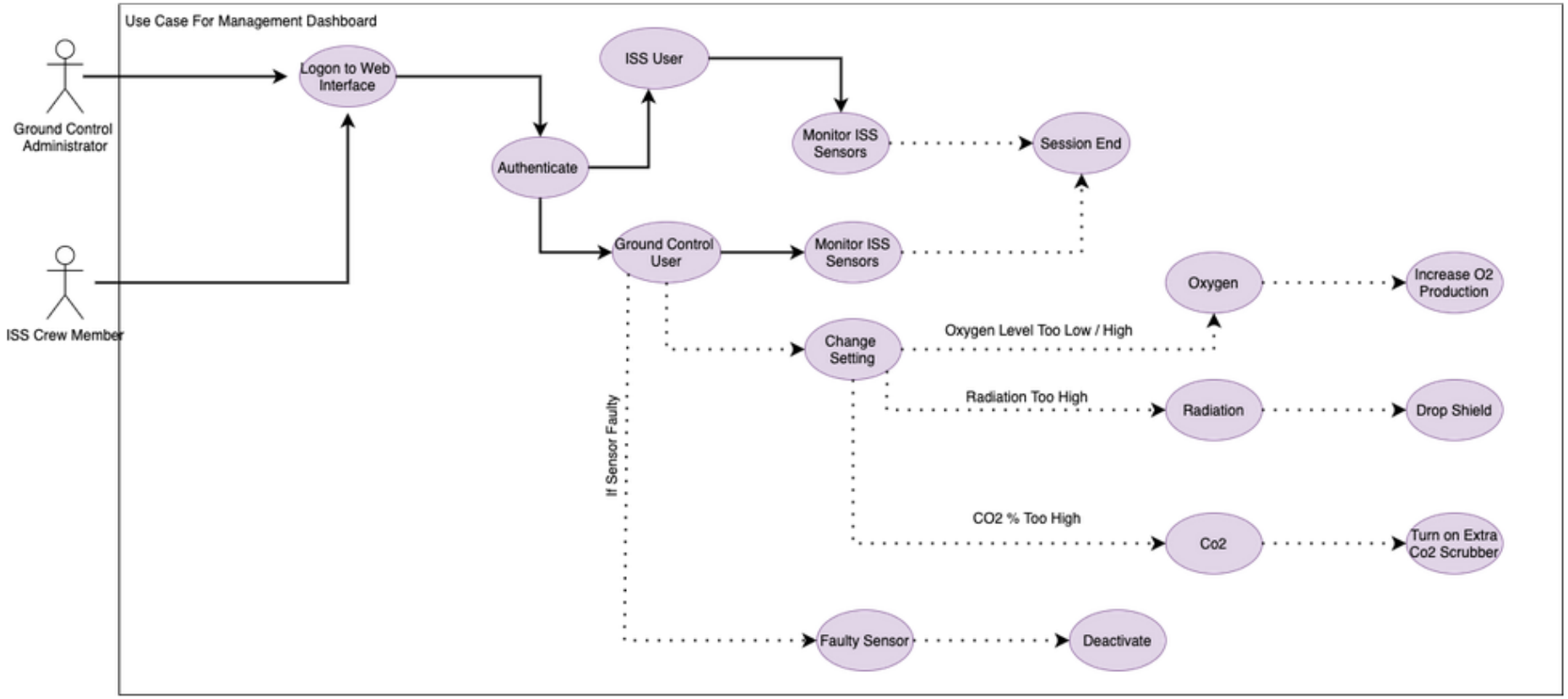


Figure 9 overleaf illustrates a use case diagram for the command and control layer dashboard.

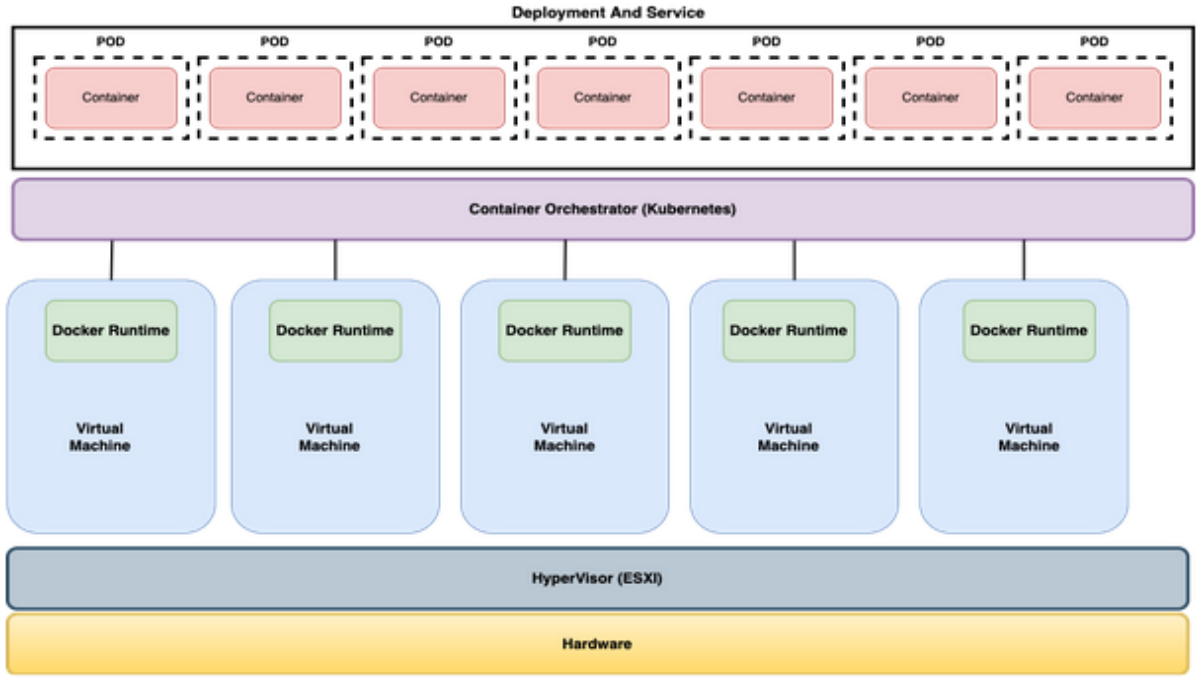
Figure 9: Use case diagram for C&C layer



Clustered Microservices Architecture

Our system is built on a microservices architecture to ensure high availability and scaling. (Nguyen et al., 2020). After investigating functionality and ease of deployment we decided to implement the core HA components using a multi-node Kubnetetes cluster, as depicted in Figure 5 (Vayghan et al., 2018).

Figure 10: Diagram illustrating the cluster of nodes orchestrated using a Kubernetes cluster



Database ERDs

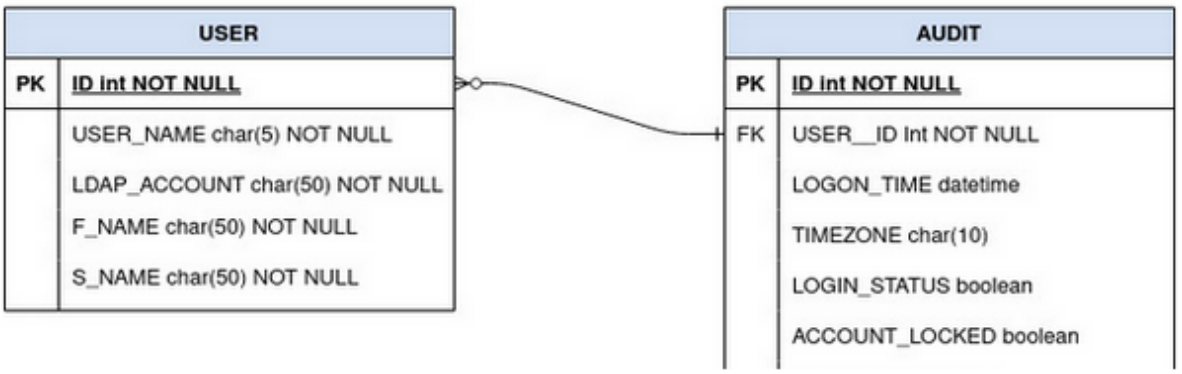
Our PostGres SQL database is normalised and split into two main areas:

- Security and Authentication
- Data Logging

Security

There are two tables: USER, storing account details and AUDIT, which stores logon and logoff events.

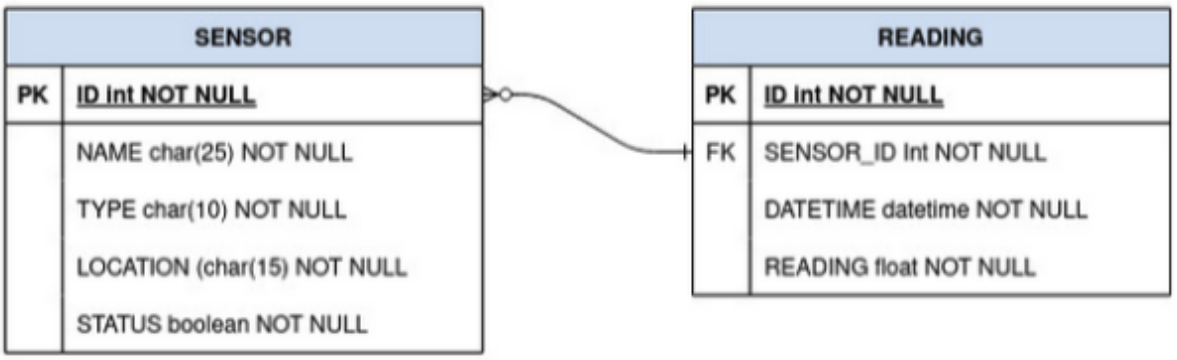
Figure 11: User and Audit tables in the Security database



Data Logging

There are two tables: SENSOR, for the details of each sensor, and READING to store the data read from each sensor.

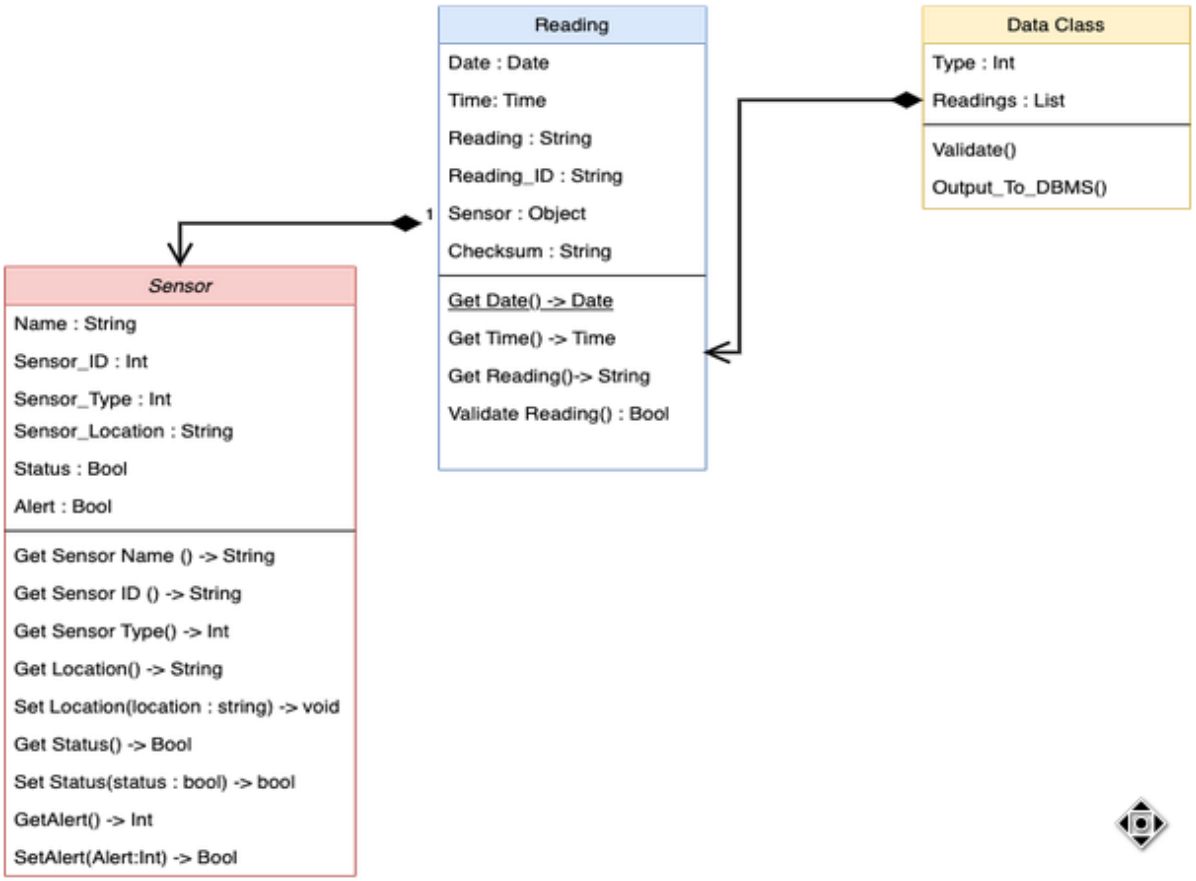
Figure 12: Sensor and Reading tables in the Data Logging database



Class Diagram

There are three main classes in the software, detailed in Figure 13.

Figure 13: Class Diagram



Tooling

Below is a table listing the tools and versions we will use for the application, as well as the reason for each one's choice.

Table 7: Software Versions utilised in our solution

Software	Type	Version	Reason
PostgreSQL	DBMS	14.2	Stable secure OSS DBMS Platform
Docker	Container Engine	20.10.14	High level of support and well documented
Redis	DMBS	6.2	High performance industry standard memory key store
Python	Development Language	3.10	Modern language ideally suited to microservice applications.
Uptime Kuma	Utility	1.0	Able to Monitor Both DNS Servers and Containers
Wire Guard	Service	1.0.15	Fast secure and built into Linux Kernel
Kubernetes	Container Orchestration	1.23.5	Greater Industry support then Docker Swarm
Rancher	KS8 Management	2.6	Best tool currently available for GUI management of KS8 Clusters.
KeepAliveD	Load Balancing	1.2.15	Only software Solution identified for load balancing
Grafana	Log Visualisation	8.4.15	Best tool in the OSS sphere for Visualisation of data
Grafana Loki	Grafana Log Agent	2.4.2	Part of the Grafana Solution to capture and visualise log and security info
Grafana Promtail	Grafana Agent	2.4.2	Agent to push logs to Loki Service
Apache Kafka	Queue Service Broker	3.1.0	Compared to MQQT Kafka is better supported with python Libraries.
GlusterFS	Distributed Storage	10.1	Distributed Storage so KS8 nodes can access files
NGINX	Reverse Proxy	1.18.0	Recommended Load Balancing Reverse Proxy with SSL Termination

We will use the following Python libraries:

Table 8: Python libraries leveraged in the application, and the reasons for their selection

Library	Version	Reason
PyTest	7.1.1	IDE Support and feature set easier to use the unittest library
Flask	2.1.1	Lightweight framework for producing web apps
JWT	1.3.1	Defacto well respected go to library for dealing with web security tokens
OpenSSL	2.8.3	Only real non-commercial option for dealing with SSL and TLS certificates.
Pyscopg2	2.9.3	faster and easier to implement then SQL academy
LDAP3	2.9.1	Support of NTLM and Secure LDAP

We will use the following static code analysis tools to ensure our code follows Python best practices, and reduce the possibility of security issues.

Table 9: Code Analysis Tools and the reasons for their inclusion

Tool / Library	Version	Reason
Sonarqube	9.3	GitHub Integration code analysis service.
Bandit	1.7.4	Local Static code analysis
Flake8	4.0.1	Python Style guide best practice PDP8
Pylint	2.13.5	Python Style best practice

We have checked the above tools for known vulnerabilities. Any CVEs that remain unaddressed in the versions we are using are listed below:

Table 10: Known vulnerabilities with the tools we are using

Software	Type	CVE	Fixed Version	Reference
Redis	Integer Overflow	CVE-2021-32687	6.2.6	https://nvd.nist.gov/vuln/detail/CVE-2021-32687#range-7197362
Python	Injection	CVE-2022-0391	3.10.Ob1	https://www.cvedetails.com/cve/CVE-2022-0391/

References

Non Academic References

Anthony, S. (2013) International Space Station switches from Windows to Linux, for improved reliability. Available from: https://www.extremetech.com/extreme/155392-international-space-station-switches-from-windows-to-linux-for-improved-reliability [Accessed 1/4/2022].

Anon (2019) MQTT Version 5.0. [Online]. Available from: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html [Accessed 12/4/22].

Anon (n.d.) MQTT QoS: Understanding Quality of Service. Available from: https://assetwolf.com/learn/mqtt-qos-understanding-quality-of-service [Accessed 25/3/2022].

Anon (2020) Kubernetes Concepts Available from: https://kubernetes.io/docs/concepts/ [Accessed 11/3/2022].

Anon (2022) Guide to the Secure Configuration of Ubuntu 20.04 Canonical Ubuntu 20.04 LTS Security Technical Implementation Guide (STIG) V1R1 Available from: https://static.open-scap.org/ssg-guides/ssg-ubuntu2004-guide-stig.html

Nasa (2020). NASA Records Retention Schedules. Available from: https://nodis3.gsfc.nasa.gov/NPR_attachments/NRRS_1441.1A.pdf [Accessed 11/3/2022].

Docker (n.d.) Docker Available from: https://docs.docker.com/engine/swarm/

Malesky M. (2018) Minimum Oxygen Concentration for Human Breathing. Available from: https://sciencing.com/minimum-oxygen-concentration-human-breathing-15546.html [Accessed 6/4/2022].

Nasa (2021) International Space Station facts and figures. Available from: https://www.nasa.gov/feature/facts-and-figures [Accessed 1/4/2022].

NCSC (2021) NCSC Governance. *Available from:* https://www.ncsc.gov.uk/collection/caf/caf-principles-and-guidance/a1-governance [Accessed 12/4/22].

NCSC (2019) Harden Ubuntu Linux. Available from: https://www.ncsc.gov.uk/collection/device-security-guidance/platform-guides/ubuntu-lts [Accessed 12/4/22].

OWASP (n.d.) OWASP Proactive Controls. Available from: https://owasp.org/www-project-proactive-controls/ [Accessed 14/2/22].

So, A.(2018) CO2 on the International Space Station - Magnitude.io. Available from: https://magnitude.io/co2-on-the-international-space-station/ [Accessed 12/3/22].

Weinert, A. (2019). Your Pa$$word Doesn't Matter. Available from: https://techcommunity.microsoft.com/t5/azure-active-directory-identity/your-pa-word-doesn-t-matter/ba-p/731984 [Accessed 12/3/22].

Academic References

Anon (2001). Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES). [Online]. Available at: http://csrc.nist.gov/csor.http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

Jones, M. (2015). JSON Web Algorithms (JWT). [Online]. Available at: doi:10.17487/RFC7518.

Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., Kim, S. (2020) 'Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration', Sensors, 20(16), 4621, available: http://dx.doi.org/10.3390/s20164621.

Stenzel C. Deployment of precise and robust sensors on board ISS-for scientific experiments and for operation of the station. Anal Bioanal Chem. 2016;408(24):6517-6536. doi:10.1007/s00216-016-9789-0

Vayghan, L. A., Saied, M. A., Toeroe, M. and Khendek, F., "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018, pp. 970-973, doi: 10.1109/CLOUD.2018.00148.

Appendix 1

ISS Speed of Radio Transmission

$Speed = \frac{Distance}{Time}$ Rearranged to $Time = \frac{Distance}{Speed}$

Speed constant is calculated by $C = f\lambda$ or 299792458 m/s time taken to communicate with the ISS would be

$$T = \frac{400000}{299792458}$$

Which results in a transmission time of **0.001334** seconds this assumes data is beamed directly from the ISS to a ground station which is not the case

The ISS is at an elevation of 400km the data covers a much greater distance to reach Earth. The ISS transmits the signal to a satellite positioned as high as 35,786km. Only from there can it reach ground space communication stations. This means the total distance covered by data from the ISS and the reply signal sent back is about 150,000 kilometres.

$$T = \frac{150000000}{299792458}$$

Which results in a true transmission time of **0.5003** Seconds **500.3** milliseconds