/

**Table of Contents**

[***Problem Statement 2***](#_heading=h.gjdgxs)

[***ISS Speed of Radio Transmission 3***](#_heading=h.30j0zll)

[***Ground Control Telemetry Data Layer (GCTDL) 3***](#_heading=h.1fob9te)

[***Technology Stack 4***](#_heading=h.3znysh7)

[***Space platform data collection layer 4***](#_heading=h.2et92p0)

[***Technology Stack 4***](#_heading=h.tyjcwt)

[***Command and Control Layer 5***](#_heading=h.3dy6vkm)

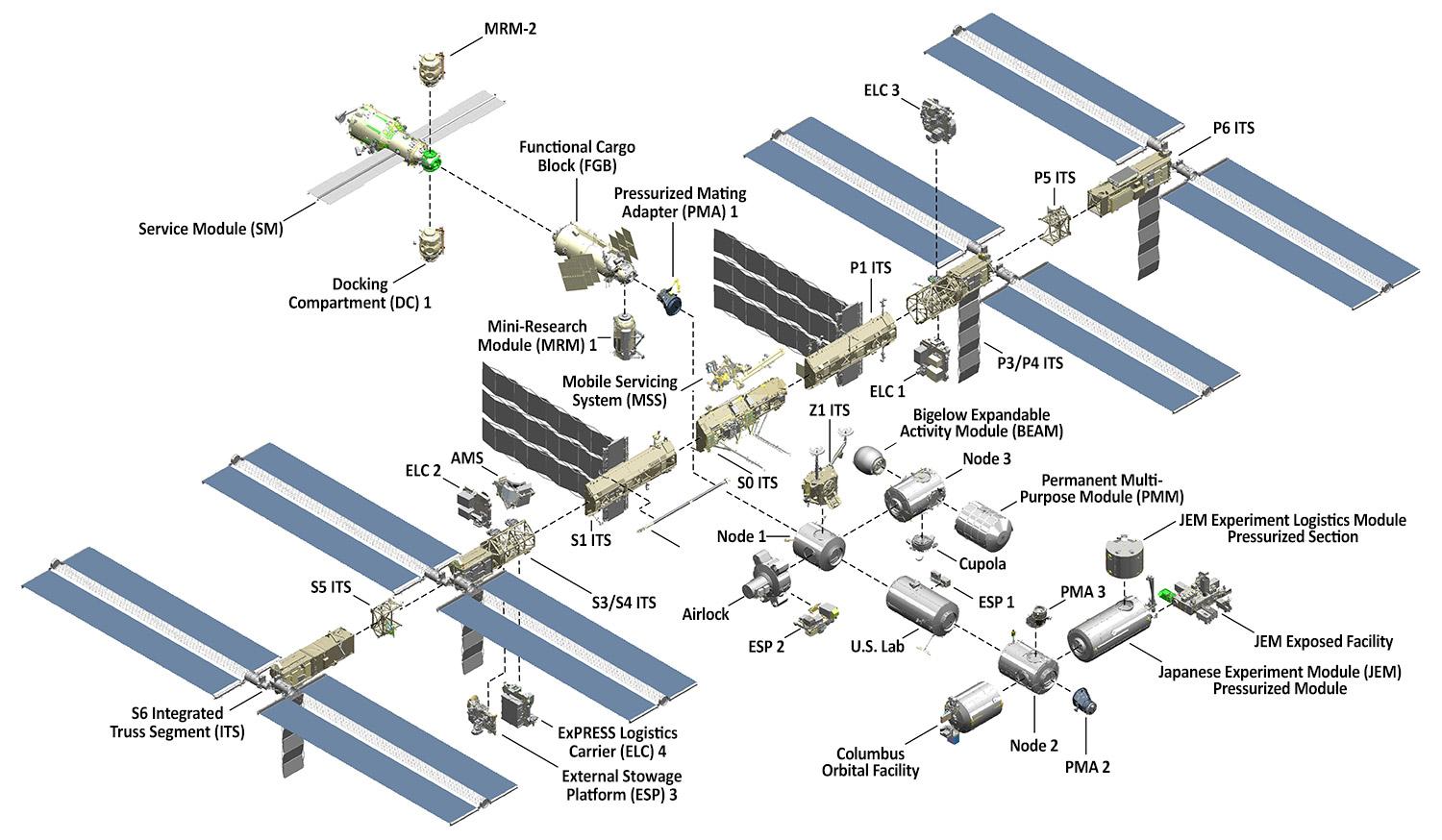
[**Functional Requirements 5**](#_heading=h.1t3h5sf)

[**Non-Functional Requirements 6**](#_heading=h.4d34og8)





The International Space Station was approved for construction in 1984 and currently earth's only permanent space station. {ADD Reference}

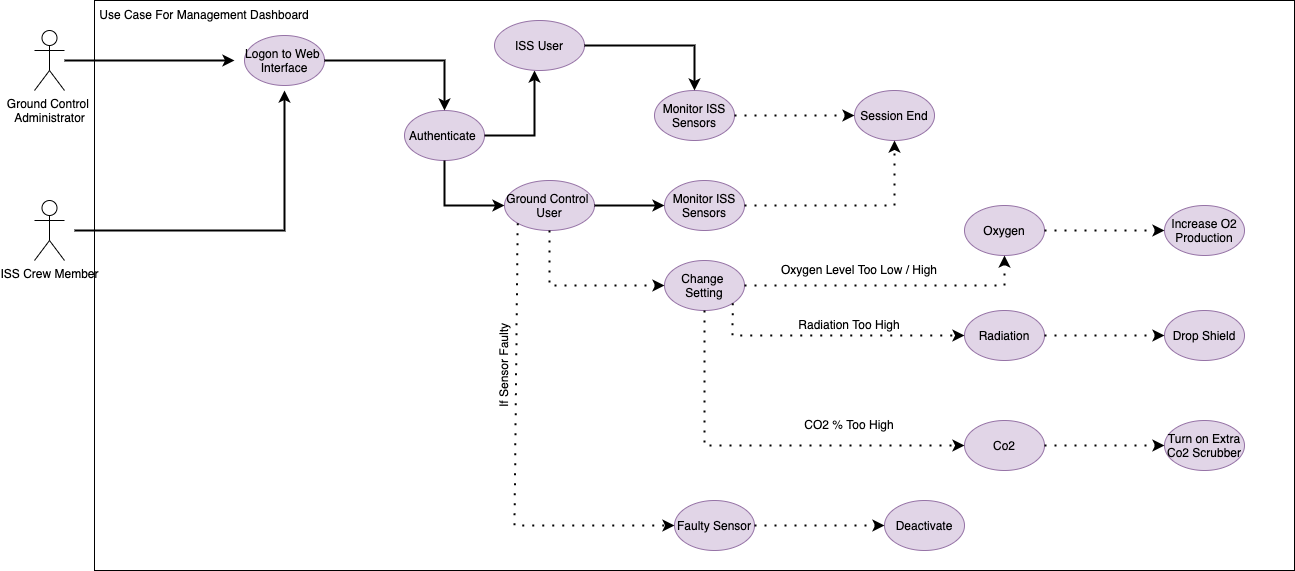


***Figure 1 (Nasa 2019)***

As a highly multifaceted machine needing hundreds of people to manage and maintain and due to limited physical accessibility the structure and modules that make up the ISS include numerous automated sensors to monitor the condition of the ISS systems sensors include

* Oxygen
* Radiation
* Fuel
* Carbon Dioxide
* Water
* Gravity
* Photovoltaic Voltage
* Airlock status

The ISS telemetry systems enable the data flowing from the station's numerous sensors to be transmitted back down to mission control allowing support services to monitor the condition of the station and to be alerted in the case of data outside of prescribed mission limits. Our proposal is to design and build a data hub to allow the collection of data from each of the sensors to be stored locally and then transmitted in a single burst transmission of data as radio bandwidth is limited. See Appendix 1



# 

****

Our Solution is built on two principles: the solution should be a High Availability (HA) solution. Once delivered into deployment it will not be taken offline by the failure of any one component or service, and the solution delivers a robust mechanism for the capturing and analysis of data.

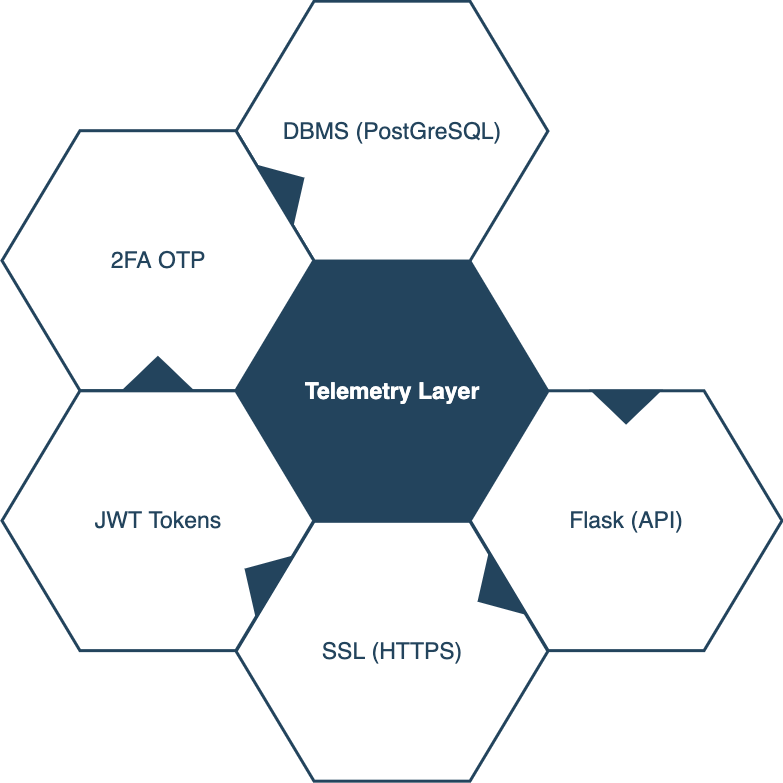
To achieve this our solution is based on redundant clustered microservices with distributed fault tolerant storage. Our solution makes use of failover technology to ensure 24/7 availability. The delivery of these services is made up of three functional nodes permitting for the failure of any two nodes without affecting delivery of the service.

The OS Choice will be Linux for the Application Tier and Windows for the Active Directory LDAP Tier. Services will be deployed via docker containers residing in a Kubernetes cluster. This is for reasons of scaling and fault tolerance. The system design is made up of a number of sub-components & integrated services these are



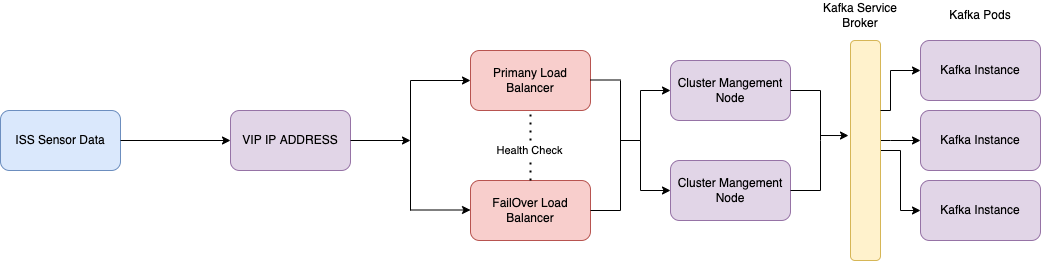
Responsible for syncing & replication of the data captured from the ISS sensors and stored on the ISS local storage; it will enable the download of this data to mission control for detailed analysis. The Telemetry layer will communicate with the ISS via a Fully Authenticated and Encrypted REST API which will act as ground control's C&C (Command and Control) channel used to initiate the download of sensor Data to the Ground control DBMS.



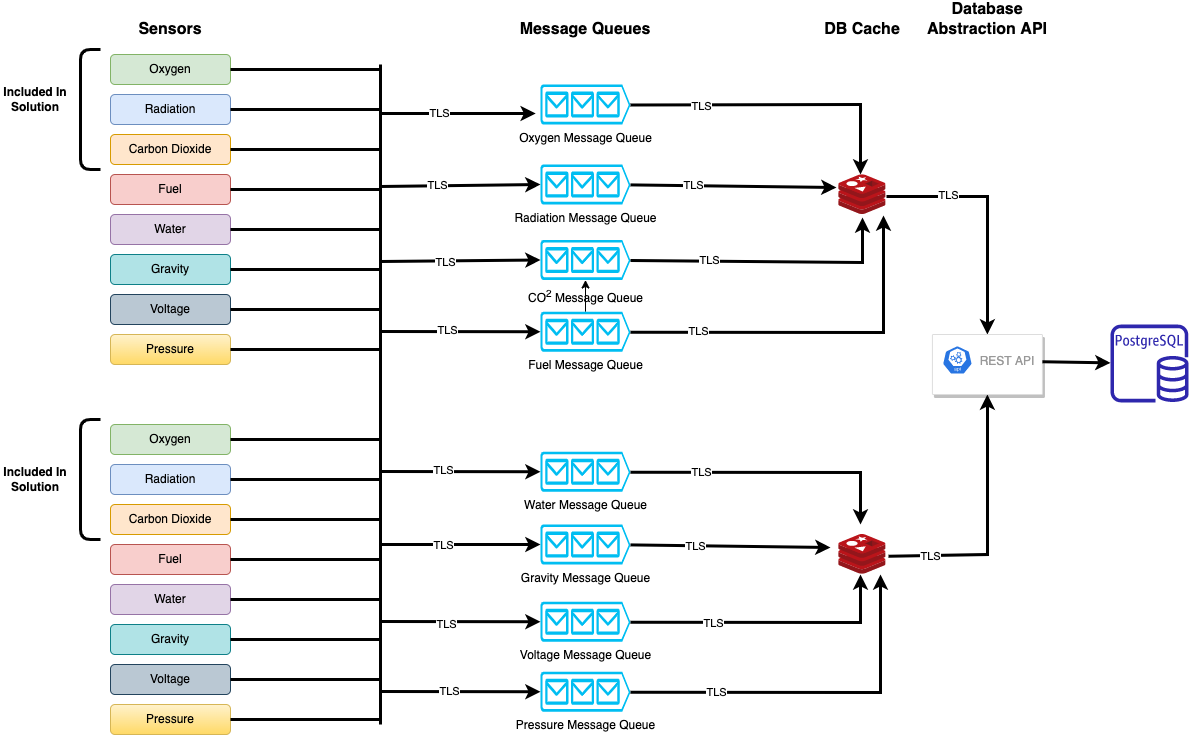
+

***Figure 2***

The second part of the system is the data collection module which will collect the readings data from the various sensor modules located around the ISS. This data will then be passed to a Kafka message queue service. The data returned from the sensors will be analysed based on a rule set. If the data being returned is outside of the expected range for the sensor type an alert would be triggered to ISS Crew and to mission control. The data is archived to a local DBMS. 



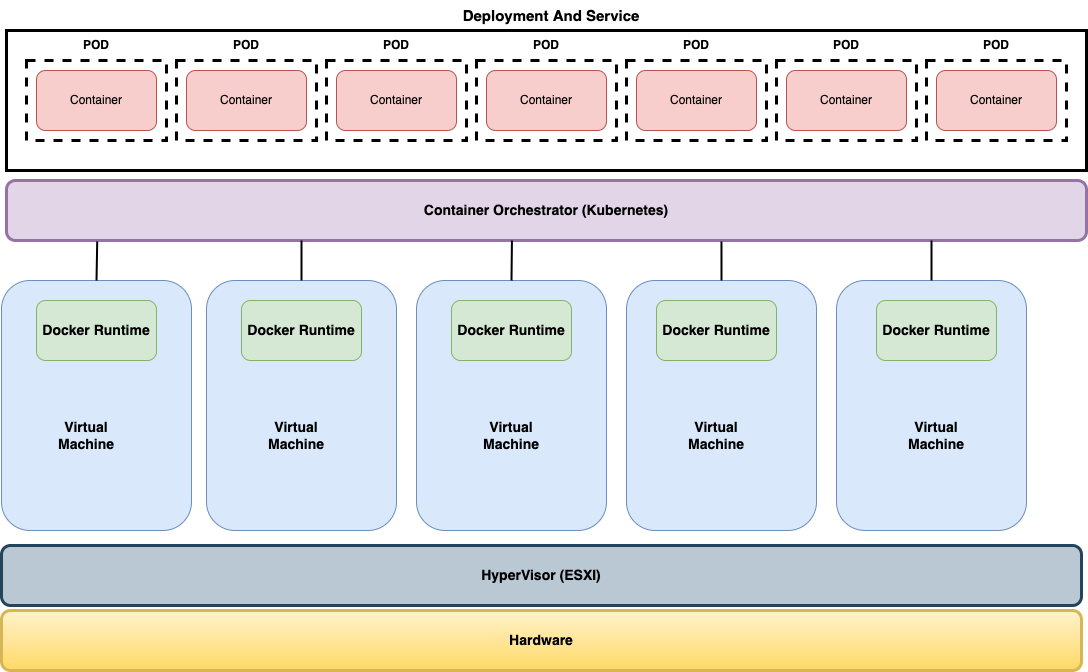




****

The Data Collection layer will be designed using containers and swarm architecture to allow for maximum horizontal scaling of the solution.





***Figure 3***

# Our Solution uses microservices (Containers) to deliver the core services of the solution. To provide high availability and scaling to meet periods of increased demand a clustered setup will be utilised to deliver the service. Two options were looked at for this part of the solutions

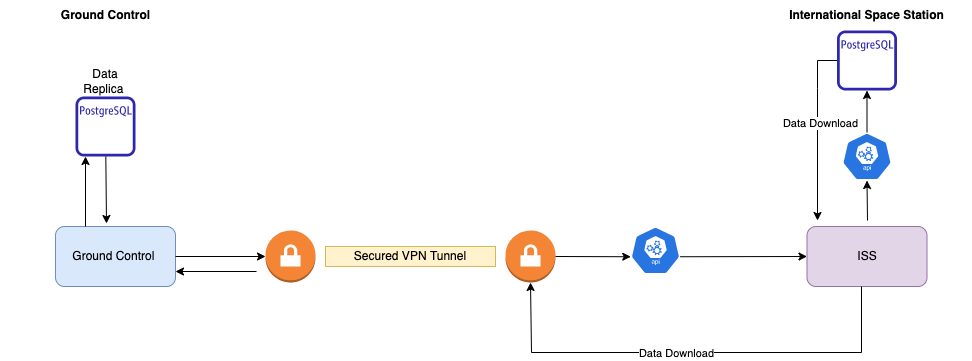
* Docker Swarm
* Kubnetetes

While docker swarm is the easier solution to implement it also lacks some of the features of the Kubnetetes solution and questions about long term support remain so for that reason for our solution we will implement the core HA components using a multi node Kubnetetes cluster.



Our solution will also have a C&C (Command and Control) element allowing ground control to change parameters on the ISS to allow the smooth operation of the station. such as turning sensors on and off and changing sensor parameters. This is achieved through the use of an API and standard HTTP Post and Get requests in keeping with our design methodology. This mechanism is fault tolerant and secure utilising industry best practice in security to ensure only those authorised are able to use the interface. The C&C Allows

1. Update of system parameters located on the ISS
2. Download of Data from ISS sensor Array
3. Removal of historic data for GDPR compliance purposes
4. Addition of New Astronauts into the system

****



* Communication will be via TCP/IP rather than X25 packet switching.
* Sensors are online and give accurate data.
* System is 24/7 Due to life support requirements

****

1. Data at rest will be secured using AES Encryption
2. Data in Transit will be Encrypted By TLS 1.2+ SSL Certificates on the Communication Channels where supported and by the use of a terminating SSL proxy where native support is not available.
3. Access to Data will be protected by strong Authentication
4. System will log all logon actions to an external log repository.



* Data captured from ISS Sensors will be stored in a PostgreSQL DBMS
* Solution will be based on Microservices & Containers to allow for horizontal scaling to meet periods of high demand
* Historic Data will be stored for trend analysis in line with NASA data retention schedules and GDPR policies.
* Data and command and control channels will be exposed via secure REST Web Services for consumption and integration with NASA / ESA Monitoring platforms.
* Solution needs to scale to meet changes in the demand profile and to ensure that service is proficient in its design to meet these changes in demand.
* Solution should be fault tolerant failure of any one node in the solution should not cause a system outage.



* The system will run on the Linux platform using OS level virtualization which ensures greater reliability and faster computation.

****

Our Solution is closely aligned to the OWASP Proactive Control methodology as this is a solution covering sensitive data that is important to ensure confidence in the solution.

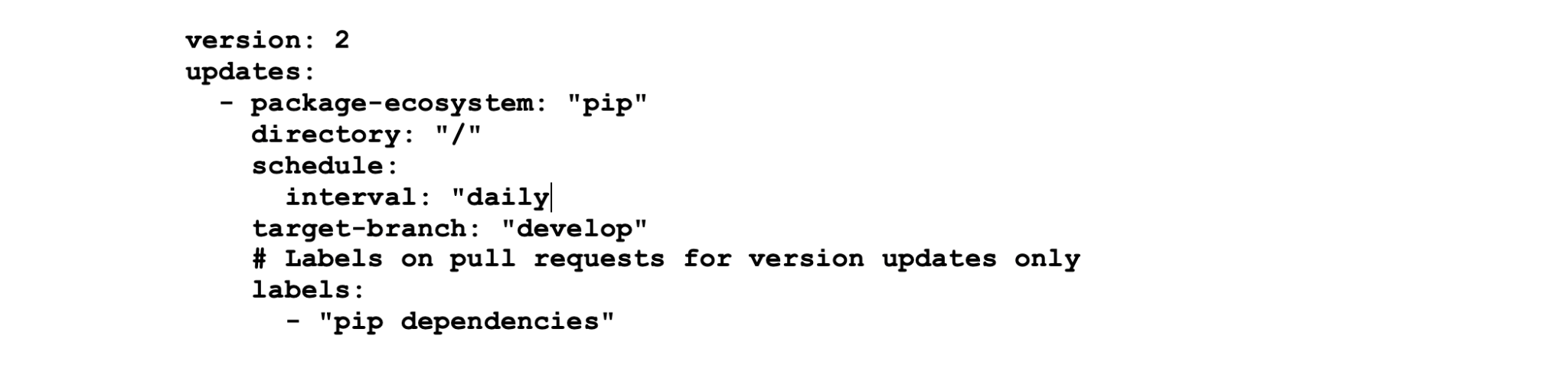


“Effective security of network and information systems should be driven by organisational management and corresponding policies and practices.” NCCS 2021 Being a cutting edge research and a valued target for hackers and rogue states the security used in any solution needs to be strong in this regard we are recommending the following after deployment.

1. Regular PenTests.
2. Server Log Analysis
3. Firewalls and Passive network monitoring to lower the target surface
4. Dedicated security team to ensure full compliance

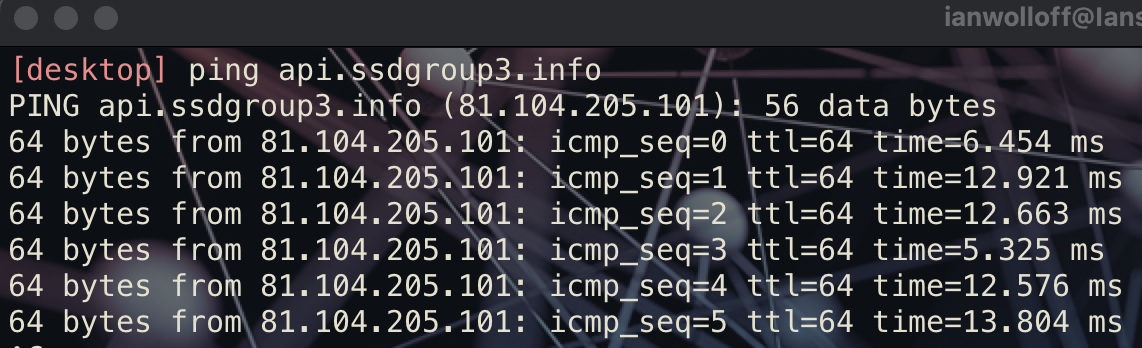
****

Our Solution will use the latest version of any libraries outside of the python base libraries. While in development our production codebase is set up to use the dependbot service to identify new versions of the libraries in use and to automatically update our solutions development branch to use these updated versions this is achieved by marking the branch to be updated in the .yml configuration file used with the dependbot Github action.

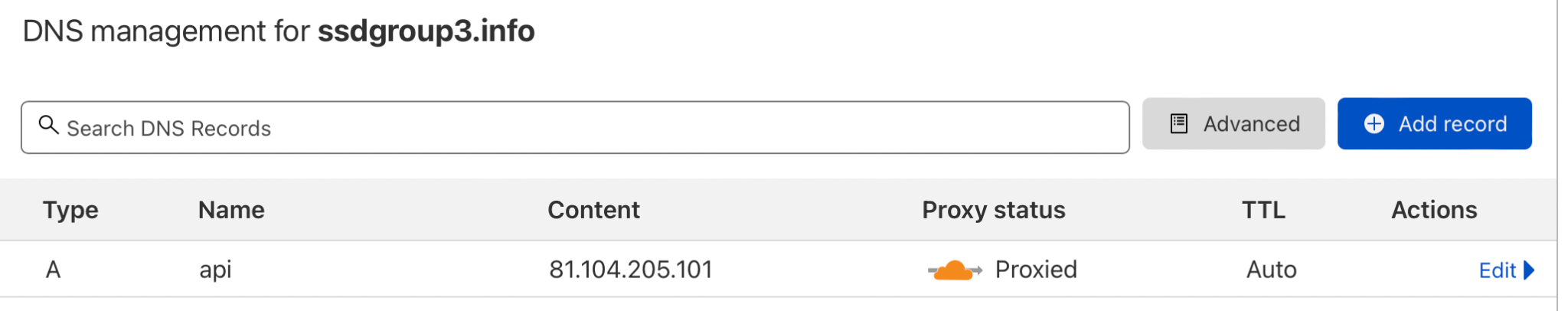


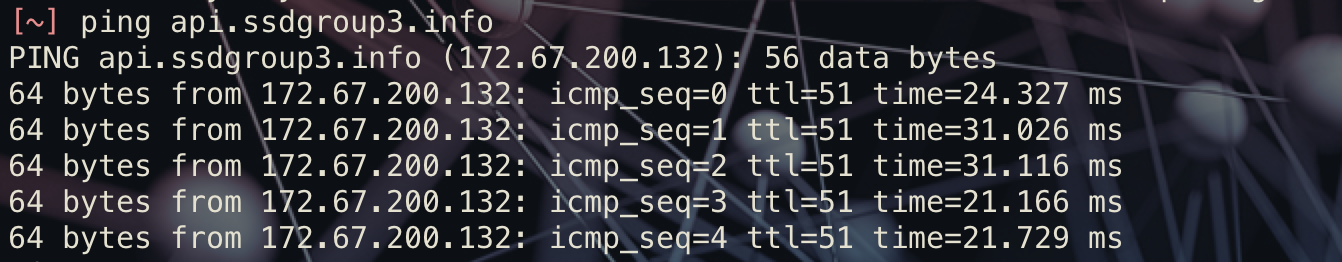
Our Solution thinks about security from the ground up. An example of this is how we secure information about our API endpoints. In a traditional API endpoint a DNS name e.g api.example.com would map to an IP address via a DNS lookup. This could allow for data leakage as once the IP of the end point is known then an attacker could use this information to gather more information on the network infrastructure

for example if we ping our api endpoint <https://api.ssdgroup3.info> we get the following



An Address of **81.194.205.101** we would not want an attacker to easily know this address. To overcome this our api is registered with a CDN that allows for proxying of DNS requests in our case this is operated by Cloudflare





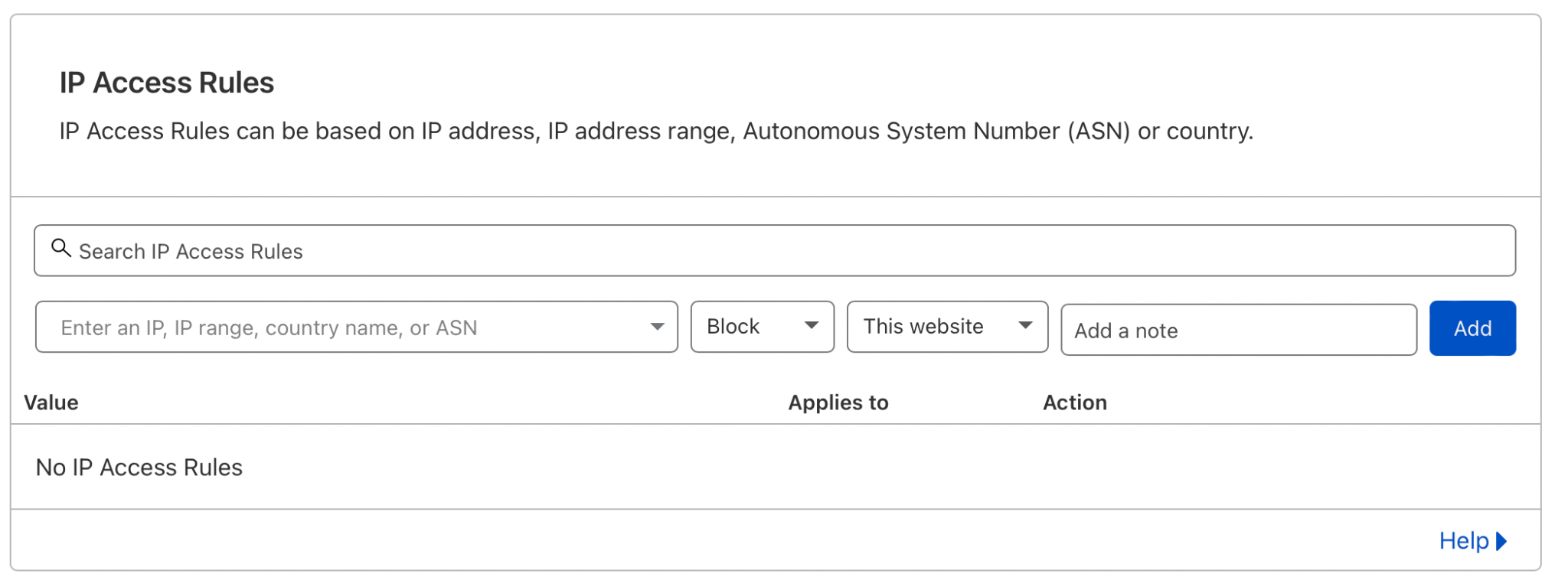
When proxied through our CDN a request to our api endpoint would now show an ip address of **172.67.200.132** while this in itself is not a substitute for proper implemented security, it is one of the tools we are going to utilise to ensure the solution can be as secure as possible.

****

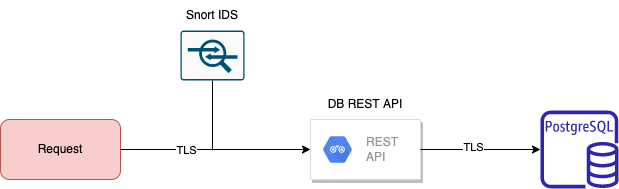
All Access to the database is secured by three methods the first is a secure API where access to the API is restricted by the use of a security token. The token we have chosen to use based on its high level of security along with ease of use is JWT token (JSON Web Token) (<https://datatracker.ietf.org/doc/html/rfc7519>) our HTTPS request would include a custom header value **X-ACCESS-TOKEN** this will contain a JWT encoded token

****This token is validated by our web service and only if the token passes validation will the API permit access to the database. The second method is restrictions on the IP address that is permitted to connect to the database. We limit access to a pool of permitted addresses. The 3rd Security technology we have in place is an inline IDS (Snort3) to monitor network traffic and to alert in the event of an attempted breach of our secure API.

Our Permitted IP Addresses will first of all be defined in the WAF (Web Application Firewall) that sits between our CDN and our API endpoint as this gives the option to block / permit by individual address or geographic location



But a secondary white list will be coded into the application so operations can only be carried out if the requesting IP is one in a known set of values



****

****

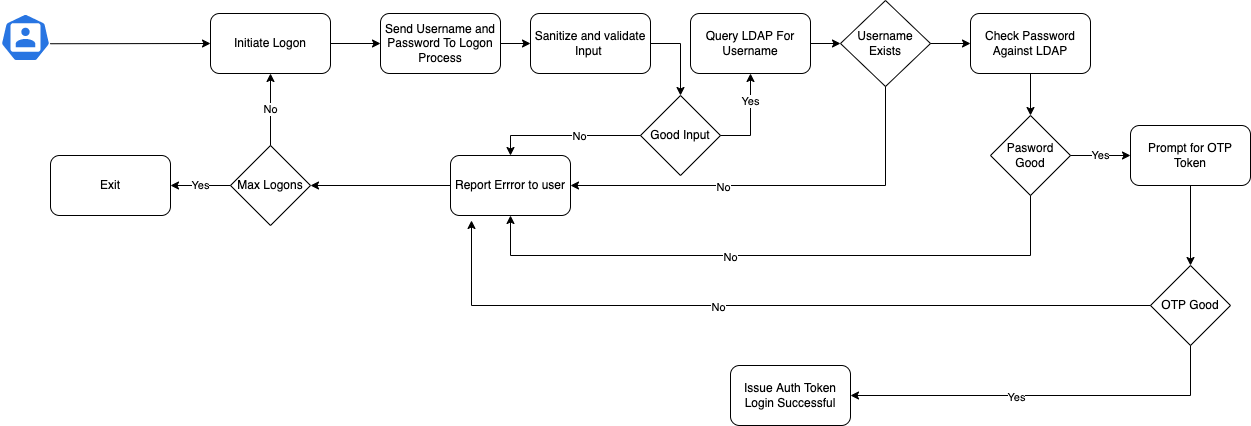
There will be two types of input into our solution client data and server data both types of data will be fully validated by the use of techniques such as escaping single quotes to prevent possible SQL injection attacks we will also carry out static code analysis on our code to ensure that any potential security flaws are identified and mitigated against. Once data makes it into the core system we dont treat it as trusted data is validated at every step throughout the platform to ensure that it is valid data and does not pose a threat to the security of the platform.

****

Identity will be used throughout the system; this will be in the Form of users accounts stored in a LDAP identity server; this will act as a single point of truth in regard to user identity. This Identity will be validated by the use of passwords enforced by system group policies ensuring passwords meet or exceed current best practice in complexity and forcing users into not using known common passwords.



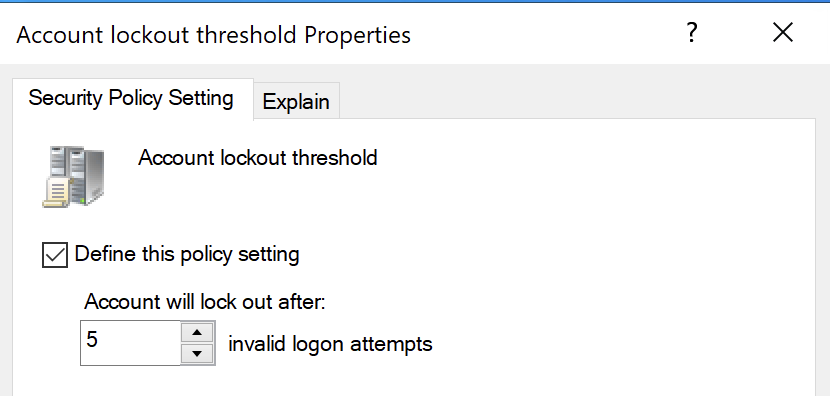
Along with implementing a **2FA-OTP** process as part of validating the identity of logons to the system



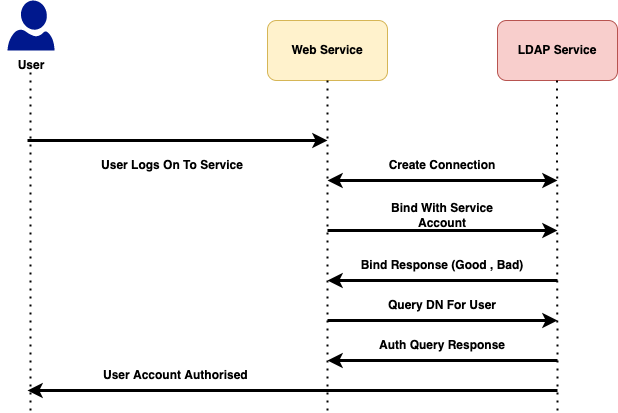
For this we plan to use the python PY-OTP library to handle the OTP code generation as this provides the functionality we need while also supporting the latest security standards such as SHA1-512 Hashes etc.

****

User level security will be provided by a couple of means the first of these is separation between the application tier and the security tier all user accounts are stored in an active directory domain with security GPOs applied defining access rights and account lockout policies.

****

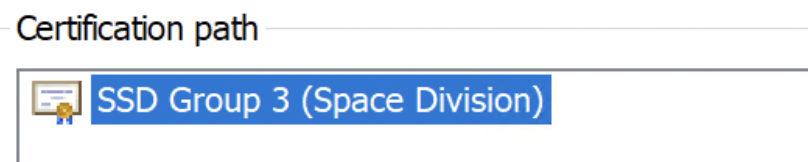
The logical process of Authentication against a LDAP instance is show below



All the Linux infrastructure that makes up our solution will be configured with the fail2ban software to lock access if non authorised access is attempted.

****

The protection of data is a key part of our solution. This is implemented by two main tenants the first of these is the use of end to end encryption via the use of SSL certificates throughout the system securing all data while in transit.



but also implementing an encrypted at rest model where data is stored in an encrypted state when not being transferred across the network.

****

Our Security strategy is made up of 2 parts

1. All VM Hosts utilise the Wazuh Client that forwards security events to a central Wazuh server for alerting and analysis. (https://wazuh.com)
2. Network Traffic is sniffed and monitored by a Passive IDS to identify security threats.

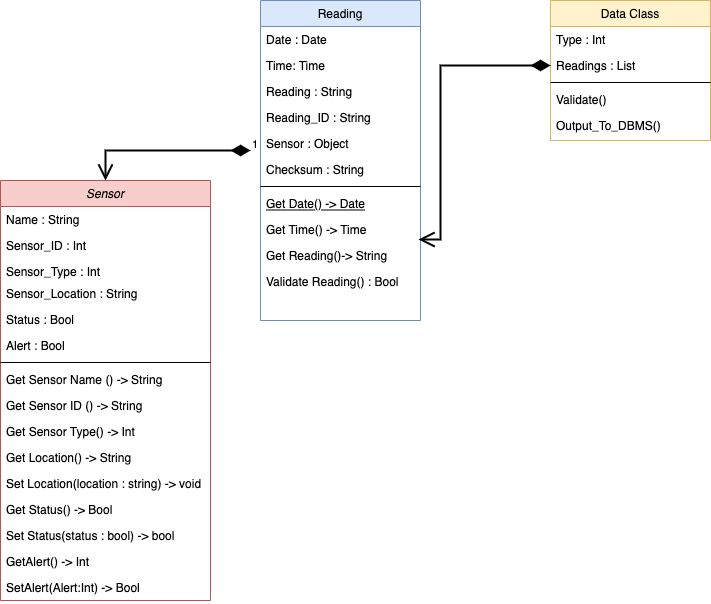
1. All Servers forward syslogs to a central syslog server to archive os level events for security and fault diagnosis.
2. Implement FIM (File Integrity Monitoring) is enabled on key system files to ensure that the system has not been compromised by hackers or rogue internal employees.

****

Our Solution is written to have full exception handling so where possible the application will recover from any issues during execution where this is not possible the microservice design of the solution will ensure that service is maintained by another pod taking over from the failed service to handle requests on the service by end users.

The Solution also includes logging of exceptions out to standard syslog (Linux) Event Viewer (Windows) which are shipped to a central location for analysis of any errors.

****

****

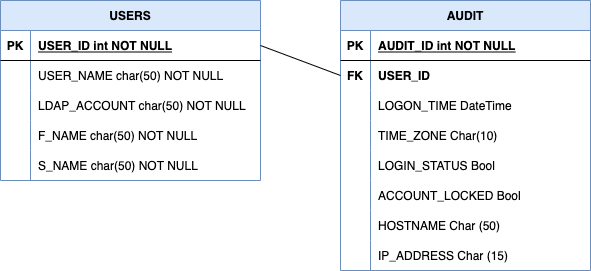
****

The Database is a key part of our solution and as such the design of the database structures is important in our overall design our database design is split into two areas

* Security and Authentication
* Data Logging

****

in our standard configuration there are two tables one for the main User account details and an supporting audit table to log logon and logoff events

****

| **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 | Utillity | 1.0 | Able to Monitor Both DNS Servers and Containers |
| WireGuard | 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 |

****

| **Software** | **Type** | **CVE** |
| --- | --- | --- |
|  |  |  |

****

| **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 goto 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 |

****

| **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 |

****

| **Service Name** | **Service** |
| --- | --- |
| CloudFlare | External DNS and DNS Proxy |
| Lets Encrypt | SSL Certificates |

****

| AES | <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf> |  |
| --- | --- | --- |
| QOS | <https://assetwolf.com/learn/mqtt-qos-understanding-quality-of-service> |  |
| Nasa Data Retention Schedules | <https://nodis3.gsfc.nasa.gov/NPR_attachments/NRRS_1441.1A.pdf> | (May 2020) |
| MQTT | <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> |  |
| Malesky Mallory Minimum Oxygen Concentration for Human Breathing | <https://sciencing.com/minimum-oxygen-concentration-human-breathing-15546.html> |  |
| Docker | <https://docs.docker.com/engine/swarm/> |  |
| NCSC A1 Governance | https://www.ncsc.gov.uk/collection/caf/caf-principles-and-guidance/a1-governance | 2021 |
| CO2 removal on the ISS | https://magnitude.io/co2-on-the-international-space-station/ | So Anthony (2018) |
| Harden Ubuntu Linux | https://www.ncsc.gov.uk/collection/device-security-guidance/platform-guides/ubuntu-lts | NCSC 2019 |

****

# ISS Speed of Radio Transmission

Rearranged to

Speed constant is calculated by or 299792458 m/s time taken to communicate with the ISS would be

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.

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