18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

SpaceOps-2025, ID # 263

Building a Ground Network Orchestration System for Newspace and the Future

Pradeep Peiris[a], Petrus Hyvönen[a]*, Javier Peña[b], Daniel Hessing[a]

[a] *Swedish Space Corporation (SSC), Sweden*
[b] *Swedish Space Corporation (SSC), Spain*
* Corresponding Author: petrus.hyvonen@sscspace.com

Abstract

Ground network services, or Ground Station as a Service (GSaaS), consist of several technical components, with a central role played by the software that provides the platform to efficiently enable a multi-mission service provisioning, sharing from a pool of infrastructure resources (such as antennas, basebands, networking) between different users and use-cases. This platform also integrates the ground station network into the user's system ground segment which can be either in the cloud or on-premises. Some parts of this platform are specific for the ground network domain, such as specialized baseband hardware or Software Defined Radios (SDRs), while others are parts familiar to software and cloud solutions in general. Challenges in GSaaS platforms are for example configuration of heterogenous sets of hardware resources (antennas, basebands, SDRs, etc) over a geographically distributed network, efficient APIs for users, resource optimization and observability. SSC is currently developing a next-generation GSaaS platform, that leverages the latest technology developments and utilises proven software engineering best practices, such as cloud technologies and modern microservice architecture, while also applying domain-specific expertise in the ground network operations. The paper discusses several interesting developments related to this in the IT industry, including platforms for the telecom industry. Additionally, the paper discusses how, in addition to the traditional CCSDS interfaces, SSC has developed APIs for RESTful scheduling, real-time monitoring, payload data distribution and real-time TT&C protocols based on WebSocket protocol. While the initial development targets newspace-type customers, there is growing interest in these APIs for larger missions as well.
Keywords: GSaaS, Software Architecture, API, Ground Network

1 Introduction

The ground station network, often solved today by Ground Station as a Service (GSaaS) is an essential component of an operational spacecraft system. This system has traditionally been associated with the space domain but is dependent upon several technical components with high connection to other technical domains. GSaaS platforms are inherently designed to support “multi-mission” operations, allowing multiple missions to utilize a standardized set of shared resources (e.g. antennas, modems, terrestrial communication), despite variations in the individual missions' protocols, operational needs, and originating organizations.

Software has an increasingly central role in this by providing a platform to efficiently enable multi-mission service provisioning, dynamically sharing from a pool of infrastructure resources such as antennas, basebands, networking, compute for Software Defined Radio SDR between different users and use-cases. This dynamic resource allocation is crucial not only to enhance operational flexibility, but also to maximize the return on investment for high-cost infrastructure assets, such as large antennas and specialized communication equipment.

Historically, the orchestration of these assets for commercial ground network operations has relied upon proprietary, full-stack monitoring and control systems often developed in-house or from a specialized provider. Such systems provided necessary functionalities to deliver reliable and efficient ground network services for the volume and needs at that time but have challenges in scaling up and cost of operations for reaching the level of service requested from today’s GSaaS market.

The rapid technological advances of recent decades in cloud computing, characterized by shared utilization of computational resources, storage systems, and network infrastructures, have demonstrated effective models for resource sharing. Cloud technologies allow multiple users to access shared resources in a secure and flexible manner, while maintaining data integrity and delivering user experiences comparable to dedicated resources often at a lower cost. Leveraging these modern cloud technologies presents a substantial opportunity for ground network operators. By

adopting cloud-based methodologies and platforms, GSaaS can achieve enhanced scalability, improved operational efficiency, and increased customer integration capabilities.

This paper discusses how SSC is utilizing these advancements to enhance current and future GSaaS platforms, initially through a streamlined platform implemented as a Proof-of-Concept (PoC) targeting newspace type of customers. The concepts and technologies discussed herein represent SSC's current exploratory efforts and should not be interpreted as definitive commitments, specifications, or service definitions for future offerings.

2 Evolving GSaaS Platform overview

SSC considers the next generation GSaaS product suite [1][2] as a platform to be integrated with the users' ground systems. To simplify integration with service APIs, the platform can be bundled with SDKs, CLIs and other tools [3]. The following figure depicts the concept of GSasS as a platform.

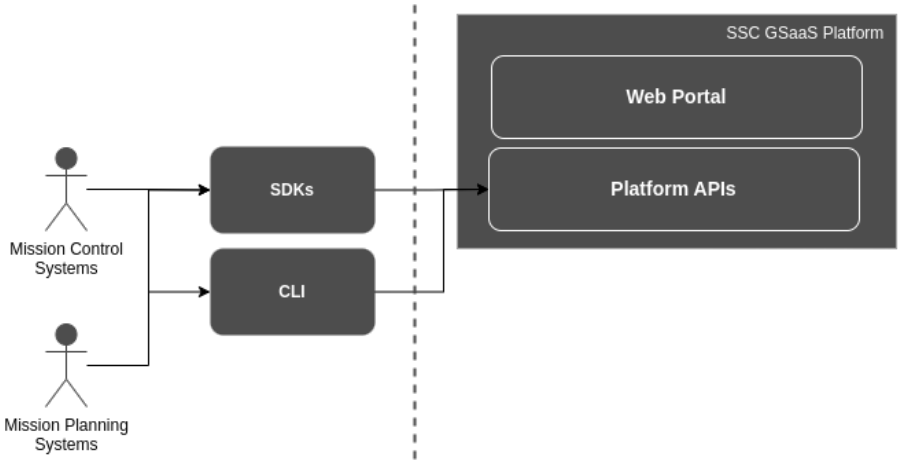

Figure 2.1 Example of GSaaS Platform and its integration with user systems.

The GSaaS platform concept is driven by few key principles; API-first approach, "Zero Trust" and "Platform as a Product".

2.1 API-First approach

Even if Web Portals, SDKs and CLIs enrich the user integration with the platform, the consistency among these integration tools is maintained by the API-first development approach. In this approach, the APIs serve as clearly defined contracts between the platform and its users, evolving through structured versioning and ensuring backward compatibility [4]. A detailed discussion on API-driven services is presented in Section 7.

2.2 Zero Trust

The GSaaS platform implement the Zero Trust security principles to further tighten security aspect of the APIs, i.e. by implementing the OAuth 2.0 security protocol for the PoC. This approach ensures that no API call is trusted until the caller is authenticated and proven for access to the requested resources based on defined access policies.

2.3 Product Capabilities over the Product Suite.

Figure 2.2 illustrates the product capabilities of the GSaaS Product suite. The main capabilities include Data Distribution, TT&C, Reservation and Monitor & Control. These capabilities are driven by separated service modules from the platform product stack.

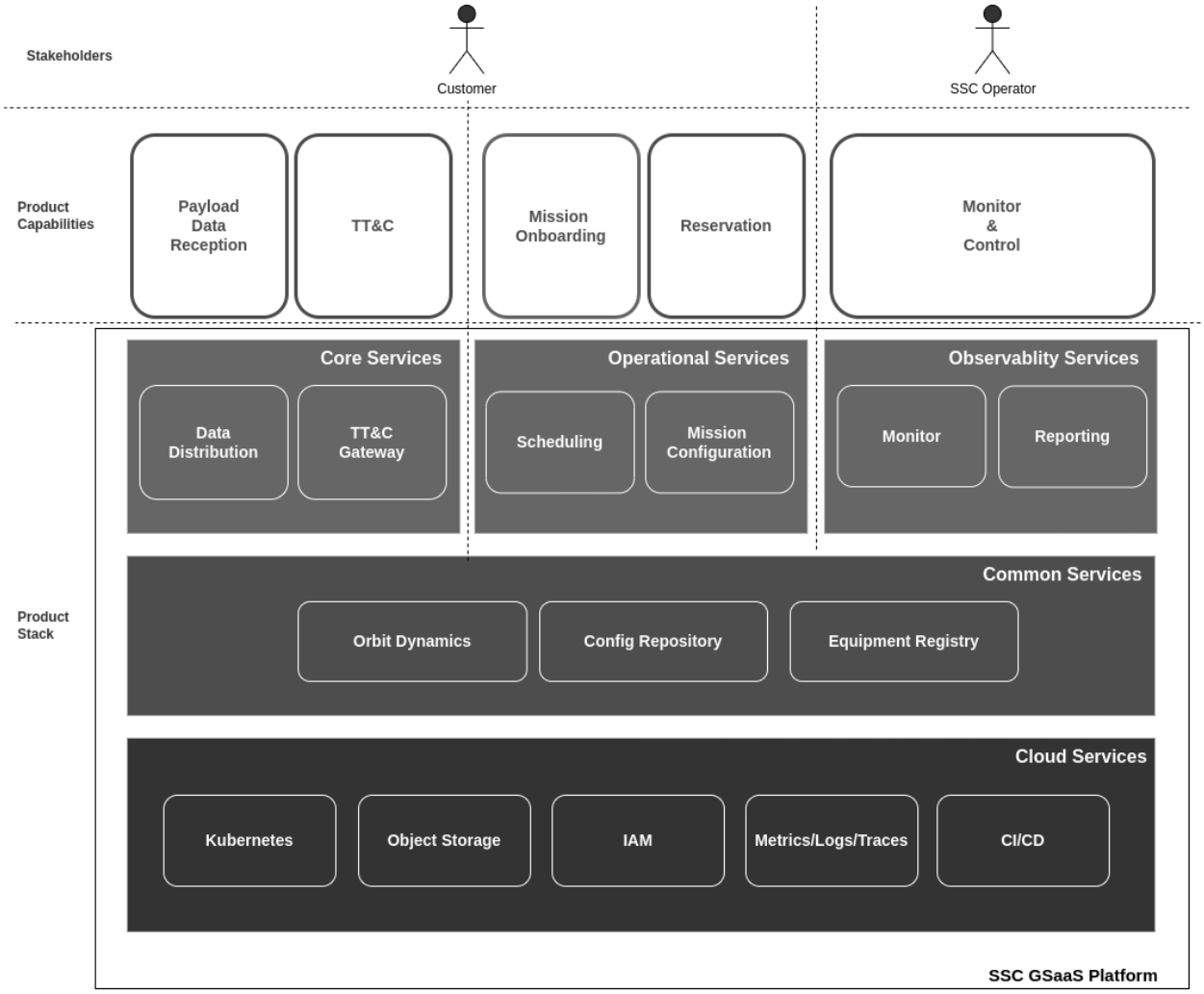

Figure 2.2 SSC PoC GSaaS Platform Product suite, showing the underlying platform layers which is part of the GSaaS technology stack/platform, and the high-level services offered on this platform.

2.3.1 Payload Data Reception

The Payload Data Distribution service is responsible for receiving data from the satellite payload and deliver it to the user in non-real-time as files (in contrast to TT&C which is real-time and streaming). This data is typically high-bitrates and received in X-band, although other bands such as S-band can also be delivered in this model. Upon reception, the payload data is first stored at the ground station, after which it is transferred to a preconfigured customer location for further processing or archival.

2.3.2 TT&C

The TT&C service provides customers real-time uplink and downlink to their spacecraft. This can be performed in several bands, where SSC is designing the newspace service for S-band TT&C. The system manages the link configuration, including modulation, bitrates and polarisation – RHCP or LHCP within the compatible frequency bands. The PoC system allows reception in either a single polarisation or simultaneous dual polarisation within the downlink frequency bands.

2.3.3 Scheduling/Reservation

The reservation and scheduling component of the platform provides time-based planning of services over time, including configuration of service parameters in advance - i.e. Service Management under CCSDS nomenclature.

2.3.4 Mission Onboarding

The mission onboarding components include functions to define parameters and needs for supporting one or more satellites from a customer in the future. This involves specifying the communication protocols to be used, the regulatory constraints, and establishing and linking the customer account in the invoicing system.

3 Modern Software Architecture

Microservice is a well-established modern software architecture approach that divides the application domain into small, independently deployable services, each managing a specific functionality. Microservices promote resilience by isolating failures within individual services, preventing system-wide outages, and facilitating continuous delivery practices that allow teams to release updates more frequently and reliably [5]. The isolation of failures corresponds well in the GSaaS context as partial failures may have no major impact on the continuation of the service functionality. An event-driven architecture model for asynchronous communication through events also enables microservices to respond in real time to changes in data and user interactions. The event-driven architecture further strengthens the microservices deployment model allowing services to be decoupled in the communication paradigm. Instead of a direct call on services, the microservice posts an event related to its internal state change, notification or command for any interested services to be reacted. Together, these software architectures enhance the development, deployment, and management of software systems.

3.1 Microservice Architecture

3.1.1 Advantages of microservices

Microservices architecture offers several notable advantages in the context of GSaaS context.

- **Flexibility and best technology adoption:** Each service has the flexibility to pick the technologies, tools and even a programming language best suited to its requirements. For example, in the GSaaS PoC, networking-related microservices are built using Golang, while data processing-related microservices are based on Python. This allows us to adopt the best technology stack for each service without needing to overhaul the entire GSaaS system.
- **Resilience:** The independent nature of microservices contributes to overall application resilience. If one service fails, it does not necessarily bring down the entire system. This isolation helps maintain operational continuity and simplifies recovery processes.
- **Continuous Integration and Delivery (CI/CD):** Microservices architecture aligns well with modern software development practices, such as continuous delivery and DevOps. This allows teams to develop, test, and deploy individual services in parallel, facilitating faster release cycles and a more agile response to change management.

-

3.1.2 Challenges of microservices

Despite its advantages, the adoption of microservices architecture also presents some challenges.

- **Complexity Management:** The increased number of services introduces complexity in managing inter-service communication, versioning, and dependencies. Ensuring seamless integration and maintaining backward compatibility among services can become complicated as the system evolves.
- **Monitoring and Debugging:** With multiple services operating across various environments, tracking performance issues and monitoring application health can be challenging. The decentralized nature of logging and metrics often makes it difficult to pinpoint the root causes of failures, thus requiring a robust monitoring solution.
- **Security Aspects:** Microservices can expose numerous endpoints, potentially increasing the attack surface. This increases the need for stringent security measures, including robust authentication, authorization protocols, and API gateways to secure data in transit and maintain service integrity.

3.2 Event driven Architecture

Event-driven architecture emphasizes the coordination of various microservices within a system through asynchronous events i.e. notifications that something has happened. In this architecture, distinct roles are played by event producers, event consumers, and event channels. Event producers generate a stream of events, while event consumers listen for and react to these events. Event channels facilitate the mediation for transferring events from producers to consumers, allowing for real-time delivery and immediate response to events [6][7].

There are unique characteristics in event-driven architecture that improve the resilience of the GSaaS platform:

- **Delivery Mechanism:** Immediate data processing is crucial in some GSaaS applications, so events are typically delivered in near real-time. The architecture may adopt either a publish-subscribe model or an event stream model, that manages event distribution.
- **Reliability and Consistency:** To enhance reliability and fault tolerance, the architecture incorporates several mechanisms. Event persistence ensures that events are durable and recoverable in failures or disruptions. Acknowledgement systems provide feedback to producers, confirming a successful delivery of events. Additionally, delivery retry mechanisms are implemented to resend events in case of failures, often with an increasing time interval between attempts to avoid overwhelming the system. Idempotency is another crucial aspect, allowing the same event to be processed multiple times without effects on the system state.
- **Guaranteed Delivery Pattern:** Adopting the Guaranteed Delivery pattern significantly improves the reliability, fault tolerance, and consistency of event processing. This pattern ensures that critical events are not lost, thus helping the systems maintain integrity and consistency even when failures or short downtime of individual microservices occur.

3.3 GSaaS Event-driven Microservices Architecture Overview

Figure 3.1 depicts the Event-driven microservices architecture of the GSaaS platform. As it illustrates, the platform spans over multiple ground station segments. The global segment includes microservices, *Scheduler Service*, *Orbit Dynamic Service (ODS)*, *Equipment Registry*, and *Service Configuration Repo (SCR)*, each of which performs direct communication over gRPC. Following an event-driven approach, the Scheduler Service performs indirect communication over services in site segments via a distributed message channel mechanism. A site segment consists of microservices, *Site Controller*, *Equipment Controllers (Antenna Controller, Baseband Controller, SDR Controller, etc), TT&C Gateway* and *Data Distributor*. All these microservices in the site segment follow an event-driven approach with the use of local messaging channels.

A typical application flow begins when the Scheduler Service posts a command event of the spacecraft's contact reservation on the Event Channel. The distributed message channel replicates the event on the designated site's channel, on which the Site Controller reacts to the reservation event. At the site segment, the Site Controller follows an event-driven approach to coordinate between relevant Equipment Controllers, TT&C Gateway and Data Distributor to fulfil the spacecraft's contact.

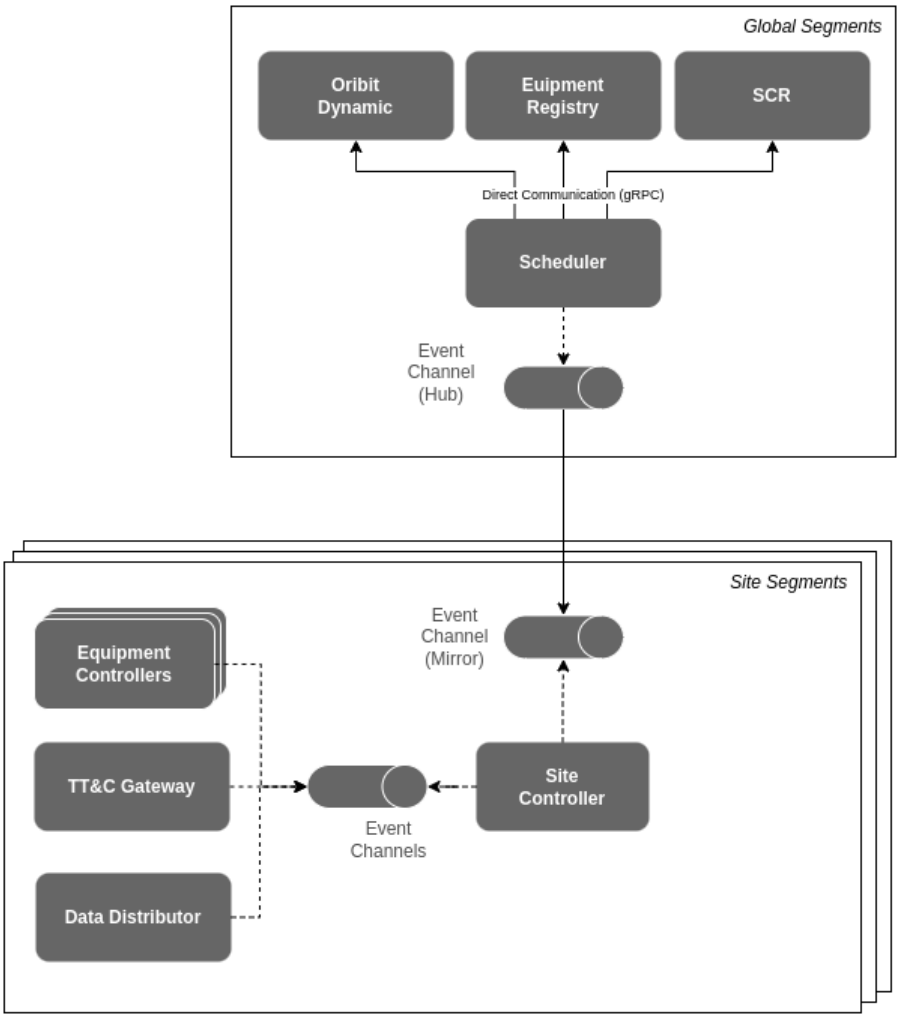

Figure 3.1 Event-driven microservices architecture of GSaaS PoC.

3.3.1 Example of microservice: Orbit Dynamics

The Orbit Dynamics Service (ODS) provides an illustrative example of the microservice architecture applied within the GSaaS context. ODS provides all astrodynamical calculations for ground network operations, such as orbit predictions, and antenna pointing. Additionally, the ODS interprets a wide variety of orbital data formats widely utilized across the space industry.

The service is, to a large degree, built on the open-source project Orekit. SSC is a Project Committee Member and the maintainer of the python part of Orekit, which is used by a range of agencies, companies and universities. Open-source projects benefit from testing, validation and inclusion of features that would be too large an effort for a single organisation. As is common with microservices, the ODS service is interfaced through RESTful APIs with clear separation between states and functions to enable usage in a Kubernetes environment.

3.4 Distributed Message Channels

The distributed message channel model is the heart of the event-driven architecture, interconnecting geo-distributed site segments with the global segments. Figure 3.2 shows an example of how contact reservations are distributed from global sites to relevant site segments. Following the Hub and Spoke deployment model, the distributed message channel system is configured to route messages to the correct site segment from the global site segment. Microservices therefore offload message routing logic to the distributed message channel system.

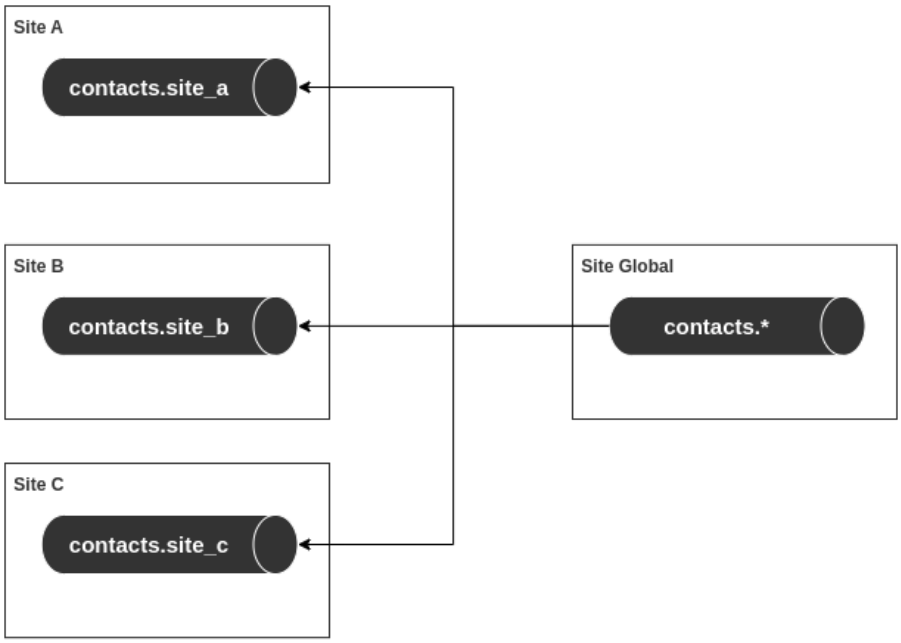

Figure 3.2 Hub and Spoke distributed messages channels used for the GSaaS PoC.

4 Geo-distributed Deployment Model and Infrastructure

Despite the advantages of microservices architecture, their deployment poses a challenge. To mitigate these challenges, it is essential to have a robust deployment environment adhering to best practices in continuous integration and delivery (CI/CD). The robustness of the deployment environment for the GSaaS platform is achieved with proper CI/CD tools and well-known industry standards technologies. These include:

- Containerization of microservices using docker.
- Orchestration of containers using Kubernetes.
- Managing Kubernetes clusters and containers from Cloud hosted services.
- Programmatic provisioning and management of Cloud infrastructure with Infrastructure as Code (IaC).

4.1 Containerization with Docker

Containerization is a lightweight form of virtualization that allows applications to run in isolated environments called containers. Each container encapsulates a microservice along with its dependencies, creating a standardized and self-sufficient unit that is ready for seamless deployment. Containerization with Docker is a well-established technology that emerged as an open-source project that established itself as the industry standard for containerization, enabling developers to package applications along with their dependencies into isolated containers that can run consistently across different computing environments [13].

All microservices in the GSaaS platform are containerized using docker. To optimize the use of docker images we have followed some best practices such as the use of minimal base images and layering that lead to faster deployments.

However, security remains a significant concern, as traditional security practices may not adequately address the complexities of containerized environments, leading to potential vulnerabilities during development and deployment.

We navigate these challenges as part of the CI/CD pipelines and hosted docker registry to perform frequent vulnerability checks to ensure reliable and secure application environments.

4.2 Orchestration of containers using Kubernetes

Managing containerized microservices over a set of virtual machines introduces problems of resource allocation and scheduling. To address those challenges, we have employed Kubernetes which provides a robust framework for managing containers across multiple VMs. Kubernetes facilitates the automation of tasks related to the deployment, scaling, and updating of applications, ensuring that systems remain resilient even in the face of failures [14]. For instance, if a container goes down, Kubernetes will automatically start another container to maintain service continuity. The microservices of both Global and Site segments in the GSaaS platform are running on Kubernetes. The following are some of the benefits that Kubernetes brings to the platform:

- **Automation:** It automates the deployment of microservice with Kubernetes built-in rolling updates strategies
- **Resource Management:** Kubernetes efficiently allocates resources among containers, ensuring optimal performance and lead to cost-effectiveness
- **Self-healing:** It can automatically restart or replace containers that fail, promoting reliability and availability.

4.3 Managed Services from Cloud

Managing Kubernetes clusters across multiple sites with its related networking and security is challenging. It is an important decision we made in the development of the next-generation GSaaS platform to use managed services from the cloud.

Kubernetes as a managed service from the Cloud allows us to utilize a Kubernetes cluster without the burden of managing the underlying infrastructure. There are three major cloud providers of managed Kubernetes services: Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE). All three services provide managed Kubernetes clusters, with some differences in terms of integration, usability, and cost structures. EKS and AKS are built on their respective cloud platforms (AWS and Azure), while GKE is recognized as Google's native Kubernetes offering [15].

Considering various factors AWS was selected as the cloud provider for managed Kubernetes and other services. AWS-managed Kubernetes, EKS has its fully managed control plane, where AWS takes care of operational tasks such as patching, upgrades, and scaling, which reduces the overhead of managing the Kubernetes infrastructure. Also, EKS offers seamless integration with a wide array of AWS services, such as Amazon S3, Amazon ECR and Elastic Load Balancing (ELB). These integrations enhance the functionality of Kubernetes applications by allowing us to take full advantage of AWS's ecosystem. Security is also a priority for EKS, which integrates with AWS IAM for fine-grained access control and authentication. EKS also includes features like VPC isolation, encryption, and private cluster networking, to meet regulatory compliance requirements.

Figure 4.1 represents an overview of AWS-managed services and integration with SSC ground stations that together build a hybrid environment for GSaaS platform.

4.4 Provisioning and management of Cloud infrastructure with IaC

Infrastructure as Code (IaC) is a modern approach for managing and provisioning cloud infrastructure. We have adopted IaC as a practice in the development of the GSaaS platform, which allows us to automate the deployment and configuration of cloud resources in a declarative way rather than manual processes.

The practice of IaC brings several advantages to the development of the GSaaS platform:

- **Version Control:** It is possible to use version control systems like Git for tracking changes, conducting reviews, and rolling back to previous configurations as needed.
- **Idempotence**: IaC tools support idempotence, thus applying the same configuration multiple times has no impact. This ensures consistent deployment and reduces the risk of unintended changes.
- **Efficiency and Speed:** As an automating tool, it reduces the time spent on manual configurations and human errors.

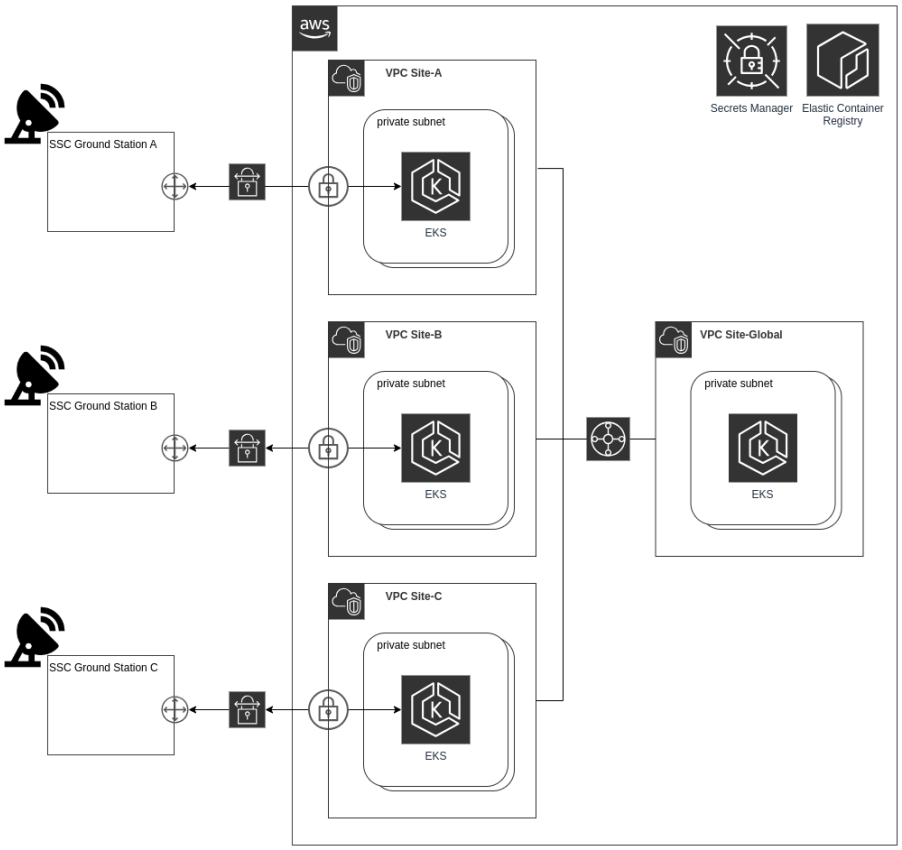

Figure 4.1 AWS managed services used for geo-distributed GSaaS PoC.

5 Observability and Monitoring

Observability is an important property of a high reliability system, to ensure continuous operations, predict issues, and conduct root-cause analysis. From the service provider's perspective, observability is based on monitoring components, including metrics, logs, and traces.

Specific challenges related to observability in GSaaS are mainly related to the geographically distributed nature of the systems. Factors such as IP latency variations, heightened security considerations, and potential disruptions in connectivity introduce complexity into monitoring strategies and data aggregation, necessitating robust, adaptable, and distributed monitoring solutions.

5.1 Observability of Microservices

Microservices' observability refers to the ability to monitor and understand the performance, health, and behavior of distributed systems through data collection and analysis. As GSaaS adopts microservices architecture, the significance of observability has grown. It is a vital part of the platform to detect issues proactively, optimize performance, and ensure system reliability. The concept of observability is anchored in three key pillars: logs, metrics, and traces [9].

- **Logs**: Provide a chronological record of events and errors, offering valuable context for troubleshooting.
- **Metrics**: Capture numerical performance data over time, facilitating time-series analysis.
- **Traces**: visualize the journey of requests through microservices, allowing for the identification of bottlenecks and failures.

Despite the advantages, the implementation of observability in the microservices landscape faces considerable challenges, including the complexity of the underlying infrastructure, the overwhelming volume of generated data, and the risk of data silos. To achieve effective observability over the challenges, the GSaaS platform has embraced a structured approach that integrates logs and metrics into well-established industry systems such as Prometheus and OpenSearch but as hosted services from AWS.

5.1.1 Central Metrics analysis with Amazon Managed Prometheus

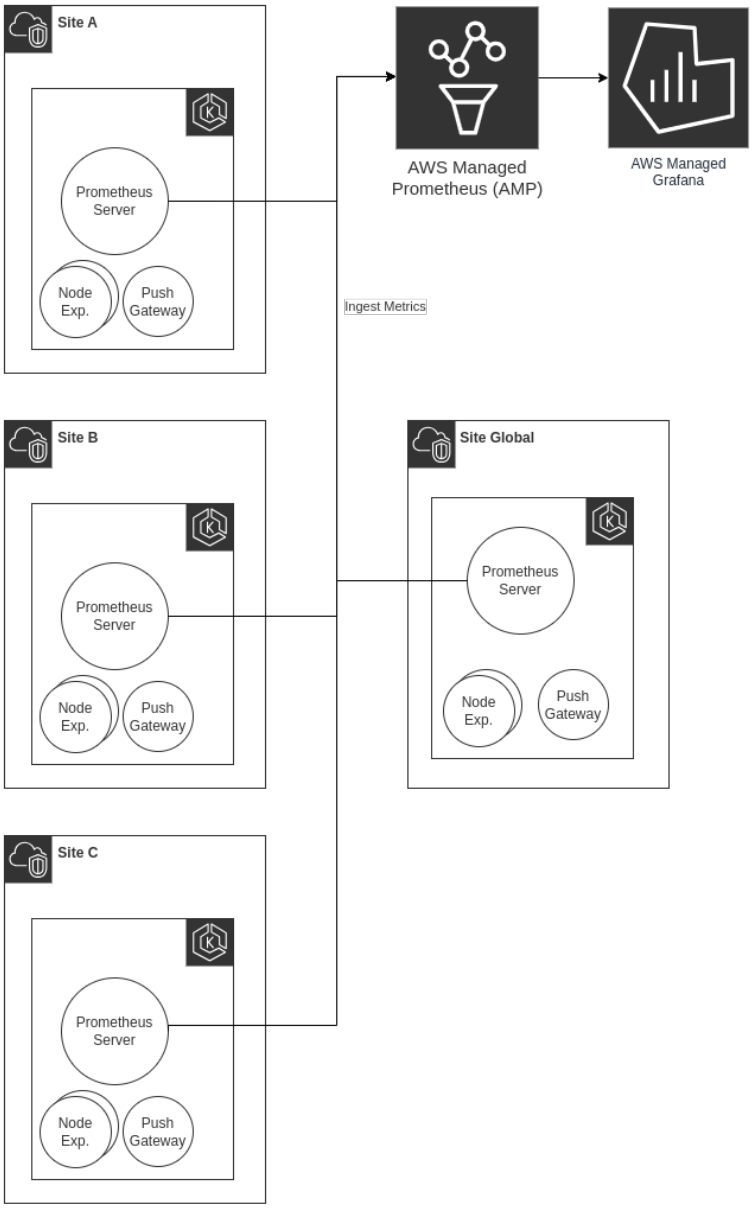

Figure 5.1 AWS managed Prometheus for centrally collected metrics.

Remote Write for Prometheus is a feature that enables the efficient transfer of time-series metrics from Prometheus instances to central storage AWS-managed Prometheus cluster [10]. The Remote Write protocol is significant for our geo-distributed GSaaS platform that manages large volumes of metrics across multiple sites, facilitating a holistic view of system performance and health.

5.1.2 Central Logs analysis with Amazon Managed OpenSearch

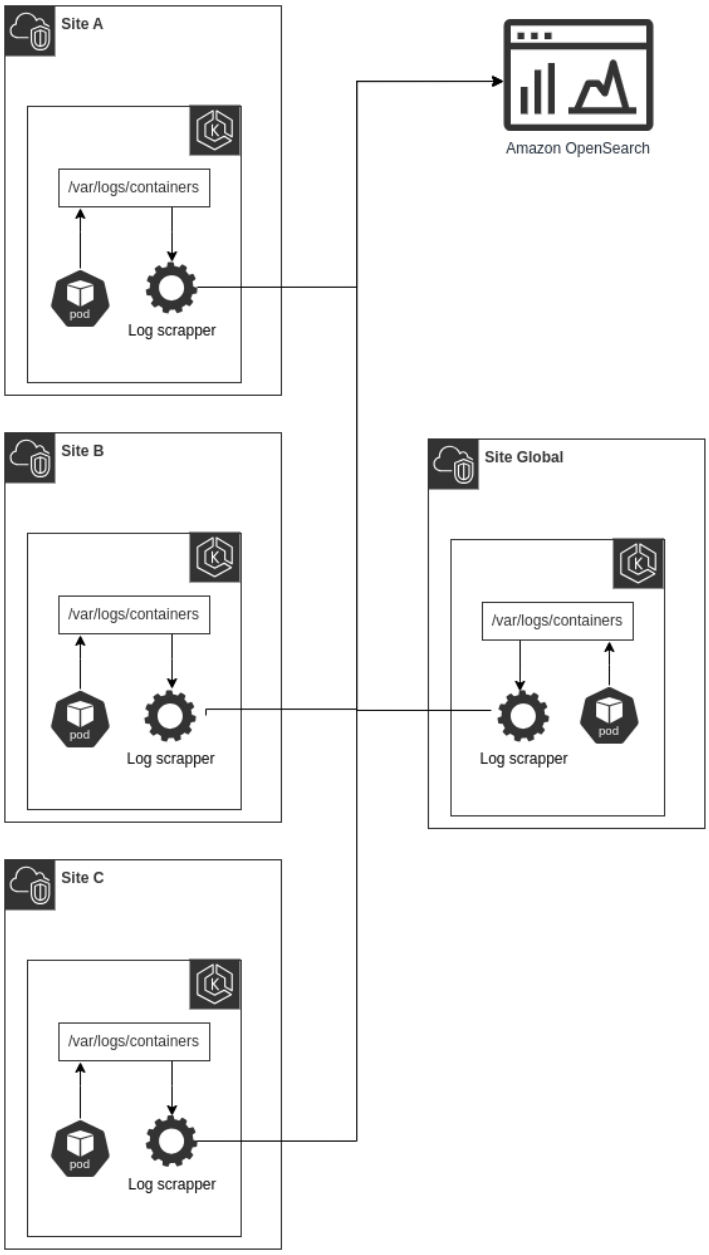

Figure 5.2 AWS managed OpenSearch for centrally collecting logs.

AWS OpenSearch is an open-source search and analytics suite that has evolved from Elasticsearch and Kibana, maintaining core functionalities while introducing additional features from AWS [11]. A central OpenSearch cluster facilitates a global view of log analytics.

6 Security

A microservice architecture introduces unique security challenges that require a comprehensive understanding of potential threats to ensure the integrity and availability of the system. Notable vulnerabilities within microservices security include authentication and authorization, data breaches, denial of service (DoS) attacks, and API injection attacks. Broken authentication may occur if attackers exploit flaws in authentication mechanisms to access user accounts and sensitive data. Similarly, broken authorization can lead to privilege escalation, enabling malicious actors to gain control over system resources. Data breaches remain a critical threat, with attackers employing techniques such as man-in-the-middle (MITM) attacks to exfiltrate sensitive information, while Distributed Denial of Service (DDoS) attacks aim to overwhelm services and disrupt application availability [5]. To mitigate these risks, best practices and architectural patterns such as Identity and access management, API Gateway and Services mesh are applied in the GSaaS platform.

6.1 Identity and Access Management

Identify and Access Management (IAM) refers to a collection of processes/tools for managing digital identities and resource accesses. IAM for GSaaS manages the entire life cycle of digital identities of the stakeholders on the platform using standard processes and technologies.

These include:

- A way to specify entitlements and roles related to the requested resources. Entitlements are usually a set of attributes that specify the access rights of an authenticated identity, e.g. Spacecraft ID.
- An authorization workflow process for approving the entitlements/roles grants for an identity.
- Managing entitlements/roles assigned to an identity.
- Auditing entitlements/roles of identities

We adopted OAuth 2.0, a widely used protocol that enhances identity and access management by allowing client applications such as mission control systems (MCS) and mission planning systems (MPS) to obtain limited access to their resources without sharing the credentials. The protocol performs this through access tokens, which are issued by an authorization server. Client applications use the access token to access protected resources governed by APIs in the GSaaS system.

6.2 API Gateway

The API gateway is a crucial component of the GSaaS platform that serves as a single entry point for clients to access the backend microservices. It applies identity and access management on all ingress traffic (North-South traffic) from an external origin to an internal backend application. Following the single responsibility principle, the API gateway offloads security concerns from microservices. By centralizing security functions, together with other functionalities such as retry logic, rate limiting, and circuit breaking, the API gateway reduces the complexity in microservices, allowing it to focus more on the business functions [5].

6.3 Service Mesh

A service mesh is a dedicated infrastructure layer that facilitates communication between microservices within a distributed application. Similar to API Gateway, which applies security protocols in North-South traffic, a service mesh implements security protocols in communication between microservices (East-west traffic). By enforcing security policies consistently across all communications, service meshes ensure that sensitive data remains protected within the microservices ecosystem. In addition, service meshes improve the resiliency of microservices through features like circuit breaking and retries, which help mitigate the impact of service failures [5].

7 GSaaS APIs

The SSC PoC GSaaS follows the API-first approach to ensure efficient and standardized communication between internal and external customer systems. It promotes the use of industry-widely adopted methodologies and protocols as API standards. The platform classifies its external-facing APIs into the concepts of control plane and data plane.

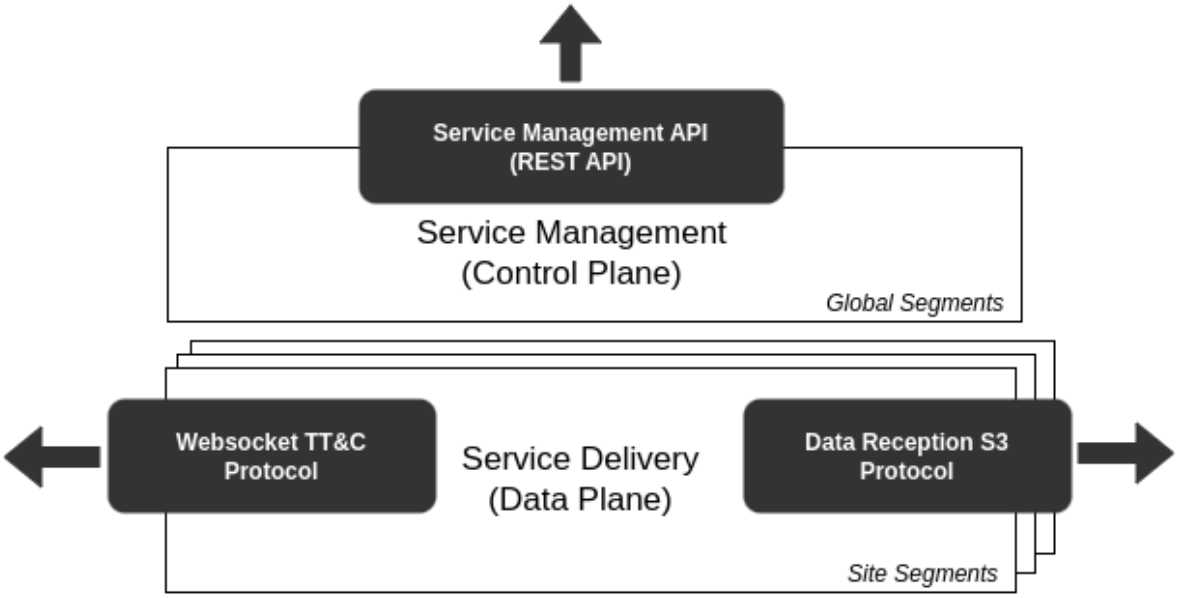

Figure 7.1 illustrates the classification of external APIs in GSaaS platform.

This classification in the figure bridges the nomenclature of the space domain and the broader IT domain. In this context, the space domain CCSDS Service Management concept aligns with the IT control plane, encompassing operations such as pass booking and reporting. Similarly, the CCSDS Service Delivery concept maps to the IT data plane, covering real-time TT&C operations and payload data reception and distribution.

The Service Management APIs are REST (Representational State Transfer) APIs that provide management and control for the client's mission resources such as contacts and spacecrafts. The data plane or Service Delivery APIs are what provide the primary function of the GSaaS platform, mainly TT&C and Payload Data distribution. GSaaS APIs have standardized error responses with descriptive error messages that improve integration with the GSaaS platform. Based on RFC-7807 [12], Problem JSON or "Problem Details for HTTP APIs" is the standard in GSaaS APIs for describing any response errors.

7.1 GSaaS REST API

The control planes REST APIs provide a simple administrative APIs used to create, read/describe, update, delete, and list resources for example:

- Reserve a new contact: `POST /v1/contacts`
- List all contacts: `GET/v1/contacts`
- Get a specific contact: `GET/v1/contacts/{contact_id}`
- Get a list of spacecrafts according to the query parameters provided: `GET /v1/spacecraft`
- Get a specific spacecraft details: `GET /v1/spacecraft/{spacecraft_id}`
- Get current TLE for the spacecraft: `GET /v1/spacecraft/{spacecraft_id}/tle`
- Upload TLE for the spacecraft: `POST/v1/spacecraft/{spacecraft_id}/tle`
-

7.2 TT&C over Websocket protocol

Telemetry, Tracking, and Command (TT&C) data plane services are exposed over a standard Websocket API. Similar to REST API, the versioning is part of the protocol. And data model for both Telemetry and Command defined in JSON or Protobuf.

- Telemetry websocket endpoint: `/v1/{spacecraft_id}/tm?format=json|protobuf`
- Command websocket endpoint: `/v1/{spacecraft_id}/cm?format=json|protobuf`

7.3 Payload Data Distribution over S3 protocol

The GSaaS platform utilizes S3 as the main standard protocol for payload data distribution. S3 is well known as the object storage service from AWS. However, S3 has also been established as a standard protocol for many object storage systems because of its flexibility over HTTP that allows seamless integration with existing systems. The GSaaS platform offers two options with the S3 protocol for payload data, a Push or Pull model. In the push model, the GSaaS platform uploads the payload data into the target user-defined Object storage bucket over the S3 protocol. In the pull model, the platform enables users to download the payload data from our S3 buckets.

8 Conclusion and Future Directions

This paper has presented a range of concepts, models, and architectural approaches currently being explored by SSC as part of a Proof-of-Concept initiative to develop a highly streamlined GSaaS platform. The effort is primarily aimed at the customer segment commonly referred to as “newspace,” where the absence of extensive legacy systems allows for the immediate adoption of modern technologies and innovative operational practices.

While the focus is on newspace applications, the underlying principles and platform architecture are equally relevant for more traditional, heterogeneous mission environments. The key challenge lies in achieving a balance—supporting established operational conops and legacy mission requirements while enabling the benefits of scalable, modular, and cloud-native systems.

9 REFERENCES

[1] Watcher Joaquim, Here's how you can incorporate feedback into your product roadmap. 24th April 2024, https://medium.com/@watcherjoaquim/heres-how-you-can-incorporate-feedback-into-your-product-roadmap-ea9a642f96a9. (accessed 21.03.2025).

[2] Ali Rawashdeh , Challenges you can expect to face when managing a platform. 4th June 2021, https://medium.com/@al1ra/challenges-you-can-expect-to-face-when-managing-a-platform-f8907a94651e. (accessed 21.03.2025).

[3] Mauricio Salatino, Platform Engineering on Kubernetes, Manning Publications, 2024, pp. 1-7.

[4] Camille Fournier, Ian Nowland, Platform Engineering, O'Reilly Media, Inc., 2024, pp. 24-25

[5] Sam Newman, Building Microservices, second ed., O'Reilly Media, Inc., 2021.

[6] Adam Bellemare, Building Event-Driven Microservices, second ed., O'Reilly Media, Inc., 2025.[7] Bahadir Tasdemir, Event-Driven Microservice Architecture. 20th November 2019. https://medium.com/trendyol-tech/event-driven-microservice-architecture-91f80ceaa21e, (accessed 23.03.2025).

[8] CCSDS 901.0.G-1, Space Communications Cross Support – Architecture Description Document. November 2013, https://ccsds.org/publications/allpubs/ (accessed 25.03.2025).

[9] Arfan Sharif, The Three Pillars of Observability: Logs, Metrics, and Traces. 2nd October 2023. https://www.crowdstrike.com/en-us/cybersecurity-101/observability/three-pillars-of-observability/. (accessed 18.03.2025)

[10] AWS User Guide, Ingest metrics to your Amazon Managed Service for Prometheus workspace. https://docs.aws.amazon.com/prometheus/latest/userguide/AMP-ingest-methods.html (accessed 29.03.2025).

[11] AWS Developer Guide, Key concepts in Amazon OpenSearch Ingestion. https://docs.aws.amazon.com/opensearch-service/latest/developerguide/ingestion-process.html (accessed 29.03.2025)

[12] IETF RFC-7807, Problem Details for HTTP APIs. March 2016, https://datatracker.ietf.org/doc/html/rfc7807. (accessed 02.04.2025)

[13] WISP, Containerization. https://www.wisp.blog/glossary/containerization (accessed 29.03.2025)

[14] Jeremy H , What Are Containerized Microservices, 30th October 2024. https://blog.dreamfactory.com/what-are-containerized-microservices (accessed 29.03.2025).

[15] BLUEXP, EKS vs GKE, 28th April 2021. https://bluexp.netapp.com/blog/aws-gcp-cvo-blg-eks-vs-gke-managed-kubernetes-giants-compared (accessed 29.03.2025).