SSC20-VII-03

Applying Modern Software System Design to Small Satellite Development and Operations

Michael Wilson
University of Michigan
mewil@umich.edu
Faculty Advisor: Professor James Cutler

ABSTRACT

Small satellite development and operations can be dramatically improved using contemporary software practices and such technologies as system monitoring, containerization, and data analytics. By logging system metrics at all stages of satellite development and operation, as is common practice in current software development operations practices, satellite operators can implement new software more quickly, detect failures sooner, and analyze information more effectively. Michigan eXploration Laboratory (MXL) at the University of Michigan has created a system that aggregates millions of data points from a wide range of sources including our satellite development units, our servers, our ground stations, open source ground station networks, and our flight spacecraft. The system uses multiple data stores, distributed across a number of servers, to house hundreds of gigabytes of telemetry data. Our system then provides novel ways to access that data, including online APIs and GUIs. These tools have afforded MXL an unprecedented ability to rapidly assess and maintain the health of our spacecraft both in development and in orbit.

INTRODUCTION

We present the design and evolution of the MXL Integrated Data Analysis System (MIDAS), along with lessons learned from its operation and the many technical challenges it has simplified. We also demonstrate that small satellite operations are enhanced through the use of modern software practices and tools. We first outline the inception of MIDAS, before discussing its design and the increased operational capability that it provides for MXL.

In early 2019, MXL was developing the prototype for our M-Cubed 9 (MC9) mission,[1] while also finishing the final verification and delivery of the Extended Tandem Beacon Experiment (E-TBEx).[1] At the same time, we discovered that we could leverage the work of SatNOGS,[2] a federated global satellite ground station network,[3,4] to increase our operational capacity for one of our existing spacecraft, the GEO-CAPE ROIC In-Flight Performance Experiment (GRIFEX).[1] We realized that we needed a system to augment both our spacecraft development and our operational capabilities while we worked with six satellites (development and flight units) between the three missions. We set out to build such a system. First, we created a versatile ground station client that allowed our team to more easily command and receive telemetry from our spacecraft, as well as uplink and downlink files. Shortly afterwards, we wrote some rudimentary Python scripts that pulled and decoded data from both the SatNOGS API[2] and the ground station client logs. We then dumped the decoded telemetry data into Elasticsearch,[5] a NoSQL database used for search and analytics by such organizations as CERN.[6,7] The results could then be viewed as time series graphs in Kibana,[8] a visualization tool and another part of the Elastic stack.[5] This loosely coupled collection of Python scripts was the inception of the system that over the next year grew to become MIDAS.

After the launch of E-TBEx on SpaceX's STP-2 mission[12] in June 2019, we began turning this set of Python scripts into a robust Go library and set of applications. We chose Go for its simplicity, productivity, and versatility. Go has its own straightforward build system, testing framework, and formatting rules, which reduce complexity and make it easier for new lab members who are still learning programming to understand our systems. The core application we developed was a server that could receive data from the existing ground station client via HTTP requests, as well as pull data from the SatNOGS API[2] at regular intervals. The server then stored data packets into a MySQL[13] database, before decoding and dumping the packets into Elasticsearch[5] for indexing. We then connected Grafana,[14] an analytics and monitoring platform similar to Kibana,[8] to both data stores. This allowed for the creation of such panels as the one displayed in Figure 4, letting us easily display and monitor data from our space-

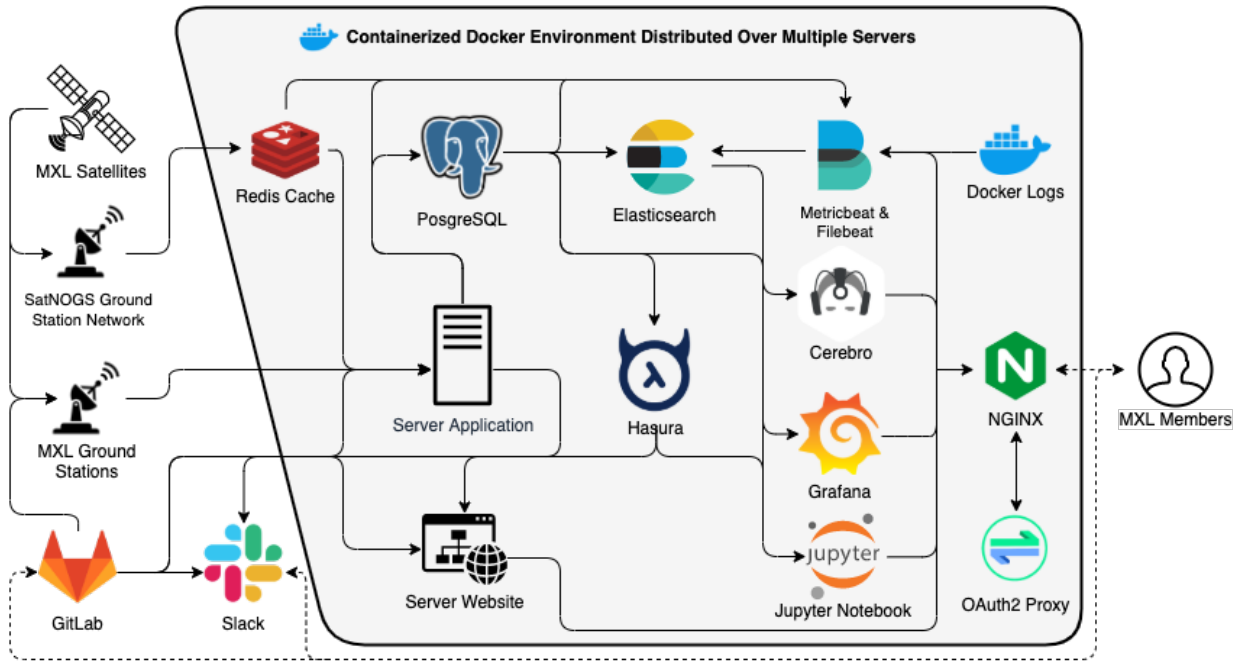

Figure 1: The MIDAS Architecture: a set of containerized applications, distributed across physical machines using Docker,[9] that interface with external services such as GitLab[10] and Slack[11] and connect MXL members to our data.

craft. Using NGINX,[15] a popular web server, and OAuth2 Proxy,[16] we made the sites secure, allowing lab members to log in with their University of Michigan Google accounts. Since the fall of 2019, we have continued to apply modern software system design to this server and add new applications to the suite of software that has become MIDAS. We outline the design of MIDAS as of April 2020 in the following section.

MIDAS ARCHITECTURE

MIDAS, illustrated in Figure 1, consists of numerous containerized services distributed across multiple servers using Docker,[9] a containerization platform. Containers, a form of operating system virtualization commonly described as lightweight virtual machines, isolate our individual services, as if they each existed on their own physical hardware. Containerization technologies are commonplace in cloud-based software environments and facilitate both development and deployment by improving security, service independence, and scalability.[17] MIDAS uses Docker to seamlessly deploy and connect its services across multiple physical machines in a single distributed environment.

MIDAS ingests data from our spacecraft via ground stations, operated by both MXL and the SatNOGS network.[2] Pulling data from SatNOGS requires both rate limiting and caching our requests to their servers; we use Redis[18] to cache received data from the SatNOGS API. Our server application then processes and pushes the data into PostgreSQL[19] and subsequently into Elasticsearch,[5] the system's primary two data stores. Other services, including Grafana, Cerebro, Jupyter Notebook,[14,20,21] and the server website, shown in Figures 2 and 5, access data from these stores. These services also pull data from Hasura,[22] a service that implements a GraphQL API on top of the PostgreSQL database.[19] MIDAS users access all services through NGINX,[15] which, in combination with OAuth2 Proxy,[16] handles authentication and forwards requests to the appropriate service. Metrics and logs are collected across the whole system using Metricbeat[23] and Filebeat,[24] "lightweight data shippers" for Elasticsearch.[5] Finally, various services can send notifications to our team via Slack[11] and when our developers make changes to our software, the changes are automatically tested and deployed using GitLab[10] pipelines.

Incorporating such recent software practices as continuous integration testing, automated deployments, and containerized environments have allowed

us to rapidly prototype new features and new ways to analyze data, while maintaining an operationally stable environment. In addition, the system's low-latency provides such new operational features as the ability to dynamically generate and execute spacecraft commands in response to incoming telemetry. Furthermore, the system integrates with communications tools such as Slack to notify operators in real time if we receive telemetry indicating a failure at any level of the system. In the following sections, we describe how MIDAS has provided vital functionality in four specific areas: ground systems and satellite communications, telemetry aggregation and analysis, systems monitoring and development, and team communications and collaboration.

TELEMETRY AGGREGATION AND ANALYSIS

MIDAS enables our lab to take full advantage of our operational capability, as it accepts data in many different forms. It accepts satellite telemetry through the original Python ground station client, a new web-based Go ground station client, and from scraping the SatNOGS API.[2] The aggregation of these three sources allows us to consolidate telemetry data in different forms into a single, queryable, data store. The server application puts binary packet data from these three sources into the PostgreSQL database.[19] Telemetry packets are also indexed in Elasticsearch,[5] while downlinked file parts are stored in another PostgreSQL table. Before pushing the telemetry data into Elasticsearch, the server transforms each packet from its binary form into specific decimal fields according to a configuration stored in PostgreSQL. There exists a separate Elasticsearch index for each satellite. This process of shipping data from one data store to another, commonly referred to as extract-transform-load, is commonplace in modern data warehousing solutions.[25] Storing data points in Elasticsearch enables querying the time series telemetry data in milliseconds and improves performance when using such visualization software as Grafana.[14] Grafana allows MIDAS users to create such informative panels as the one in Figure 4. In addition to visualization, Grafana allows our team to monitor the state of our satellites and software and to be alerted of any changes in telemetry; this is discussed further in the section on monitoring. The quantity of data that the system has ingested as of April 2020 for each flight spacecraft, stored in PostgreSQL and Elasticsearch, is shown in Table 1.

While some satellite telemetry does not require computational analysis, for example, free disk space or memory usage, it is also important to user such telemetry as magnetometer data to determine spacecraft attitude. Using Jupyter Notebook[21] and MIDAS data APIs, our system provides lab members with the tools they need to assess such characteristics. In collaborative Python Notebooks, Jupyter[21] allows users to make requests to the MIDAS API, PostgreSQL, Elasticsearch, and Hasura,[5,19,22] extract and transform the results, and make informed operational decisions. Figure 3 shows a heatmap, generated in Jupyter, of power generated per solar panel from the E-TBEx-A flight unit in orbit, from Jan 1, 2020 to Apr 27, 2020. This diagram allows our team to identify an issue with our +Y and -Y deployable wing panels, as the average power generated does not exceed 0.5W.

GROUND SYSTEMS AND SATELLITE COMMUNICATIONS

In addition to providing analytical functionality to MXL, MIDAS provides increased operational commanding capability. By querying our data stores, we can make informed decisions about what data to request from the spacecraft, eliminating duplicate data requests and therefore increasing our downlink throughput. On orbit, our satellites send out telemetry data in a beacon every ten seconds, but they only pass over our ground station at the University of Michigan for a small percentage of the day. For this reason, our flight spacecraft log and compress data into archive files to be downloaded during our short windows of contact. Our GRIFEX satellite currently stores more than five years of historical telemetry beacons from orbit. However, until MIDAS was created, we did not have a way to quickly downlink and analyze this archived data. While operating a pass, ground station users can now request a file downlink and subsequently query the MIDAS API to determine which file parts are still missing from the download. In addition, using aggregated upcoming pass data from both SatNOGS and FetchTLE,[2,26] one of our legacy services for processing two-line element sets, we generate lists of future times during which our spacecraft will be over either SatNOGS or our own ground stations. MXL operators therefore schedule downlinks of missing file parts over stations around the world, further increasing our downlink throughput. In the case of our two E-TBEx spacecraft, this integration of data from our own ground stations with data from SatNOGS ground stations led to a 2.2x increase in received telemetry data. The left side of Figure 4 depicts the low resolution of GRIFEX beacons collected only during passes over

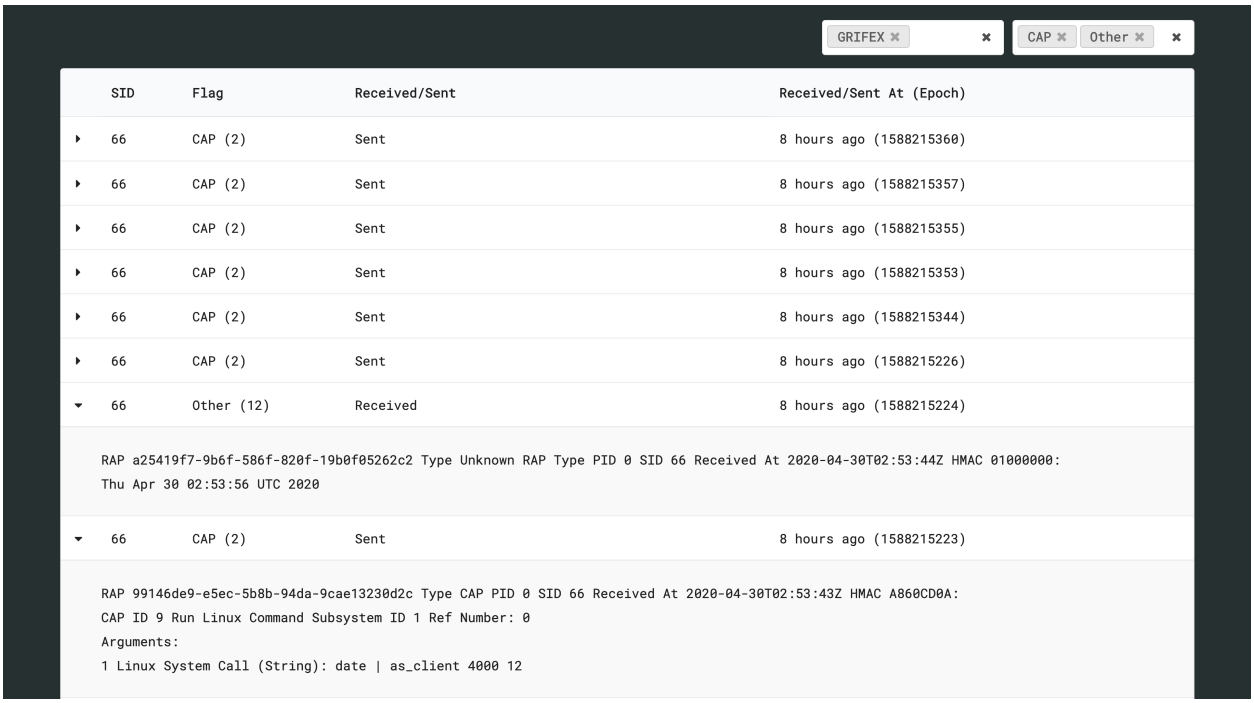

Figure 2: The MIDAS Packet Dashboard: a filterable list of incoming decoded packets. The columns SID and Flag tell us which satellite the packet is from or has been sent to and the packet type, respectively. 66 references the GRIFEX satellite, CAP (2), or Command Application Packet, is a command packet sent by our team to the spacecraft, and the Other (12) packet type denotes that the packet contains arbitrary ASCII data.

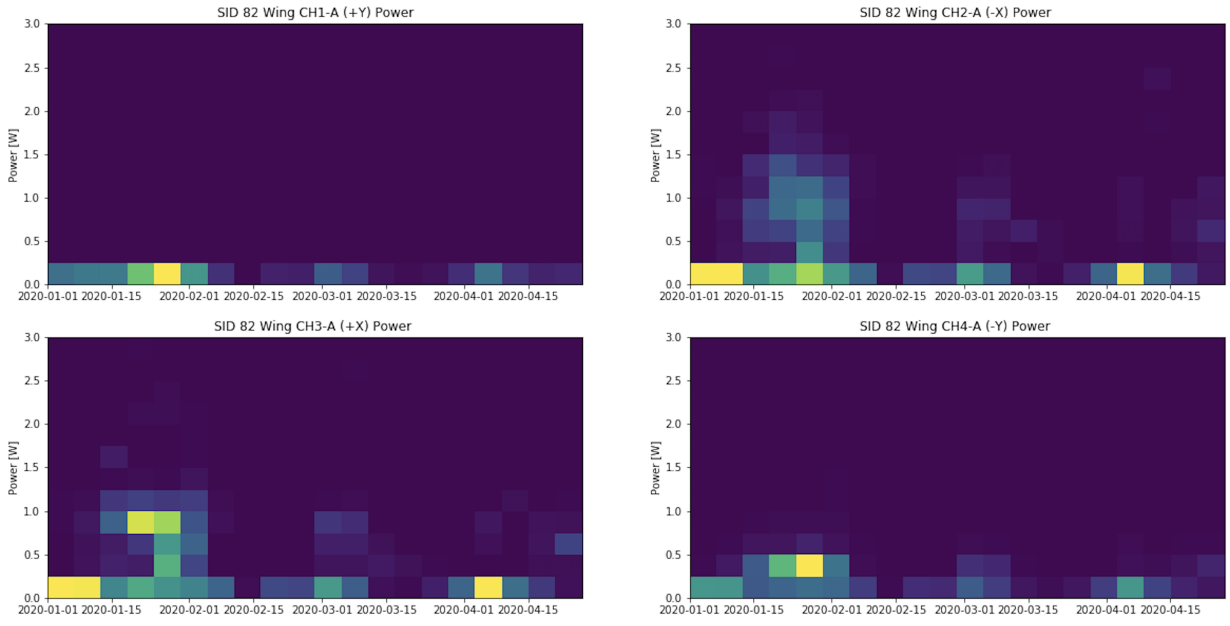

Figure 3: A heatmap generated using Jupyter that illustrates power generated per deployable wing panel for E-TBEx-A from Jan 1, 2020 to Apr 27, 2020. Color brightness indicates the frequency of occurrence.

Table 1: Data Collection Metrics for MXL Flight Spacecraft

Craft	Pass Count	Packet Count	File Count	Individual Data Point Count
GRIFEX	236,881	437,967	2,240	32,137,973
E-TBEx-A	157,668	37,059	23	2,167,941
E-TBEx-B	154,071	33,579	28	1,898,988
M CUBED-2	233,467	86,771	110	8,869,300

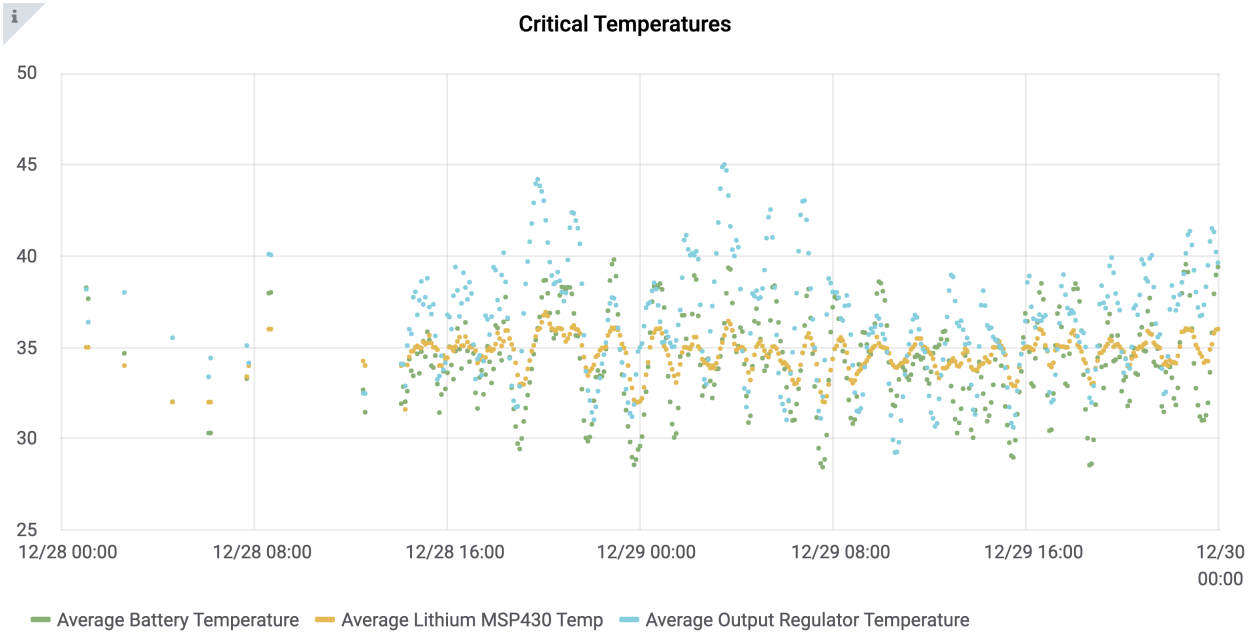

Figure 4: Critical Temperatures from the GRIFEX satellite from Dec 28, 2019 to Dec 30, 2019.

active ground stations. The middle and right side of Figure 4 show the distinct improvement that downlinked telemetry archives provide, a ten second beacon resolution. We monitor the progress of various downloads on our file dashboard, as well as verify the integrity of files using an MD5 sum and mark the files as downloaded telemetry archives as shown in Figure 5.

SYSTEMS MONITORING AND DEVELOPMENT

In contemporary software system design, monitoring and alerting have become the standard for ensuring operational success. MIDAS takes these concepts and applies them to satellite development and operations. Such tools as Grafana[14] allow us to closely monitor our systems, easily compare data between our flight and development units, and alert us of system errors or abnormalities. For example, using Metricbeat[23] and Filebeat,[24] we store millions of MIDAS log files and system metrics in Elasticsearch.[5] These log files and metrics give us valuable insights into such important operational data as how often the server makes requests to SatNOGS[2] and what the percentile latencies of our REST API are (see Figure 6). Most importantly, these queryable logs allow us to know what part, if any, of the system encounters errors. By combining these data aggregation tools with Grafana, we not only clearly report the status of our systems, but we also alert ourselves to errors using a Slack webhook.[11] These webhooks send a message to our Slack workspace whenever MIDAS encounters an error or sees abnormal telemetry data from our satellites, allowing us to address such issues as soon as they are detected. In addition to Grafana,[14] we also use Cerebro,[20] an open source health monitoring and query tool for Elasticsearch,[5] to ensure that our systems are nominal and to test novel queries on our data.

MIDAS additionally lets MXL take full advantage of our satellite development units. By running our development units for days at a time while connected to the server, we accumulate hundreds of

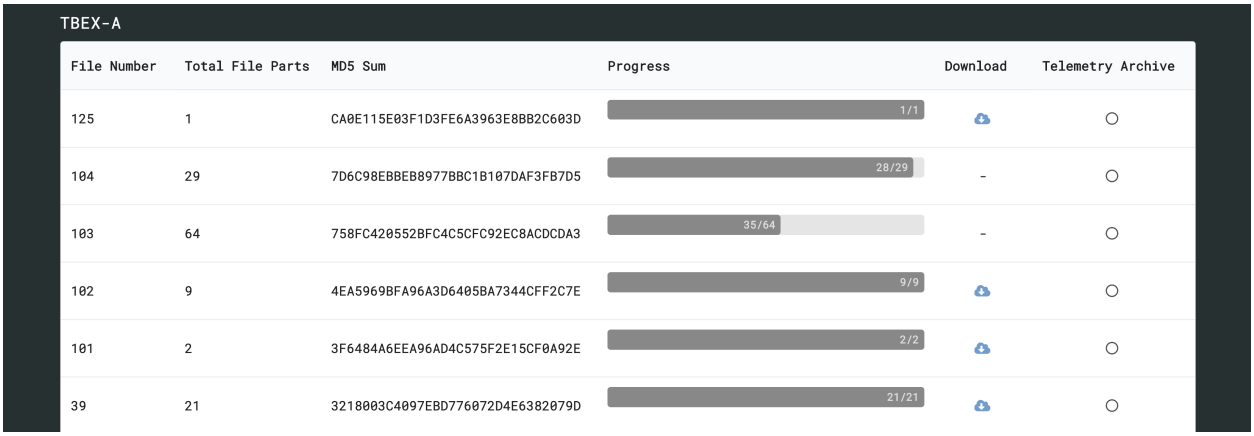

Figure 5: The MIDAS File Dashboard: a list of files with progress summary and metrics to assess data integrity. The dashboard also gives the user the ability to download completed files and mark the file as a downloaded telemetry archive, which causes the file to automatically be decoded and added to our aggregated telemetry data store.

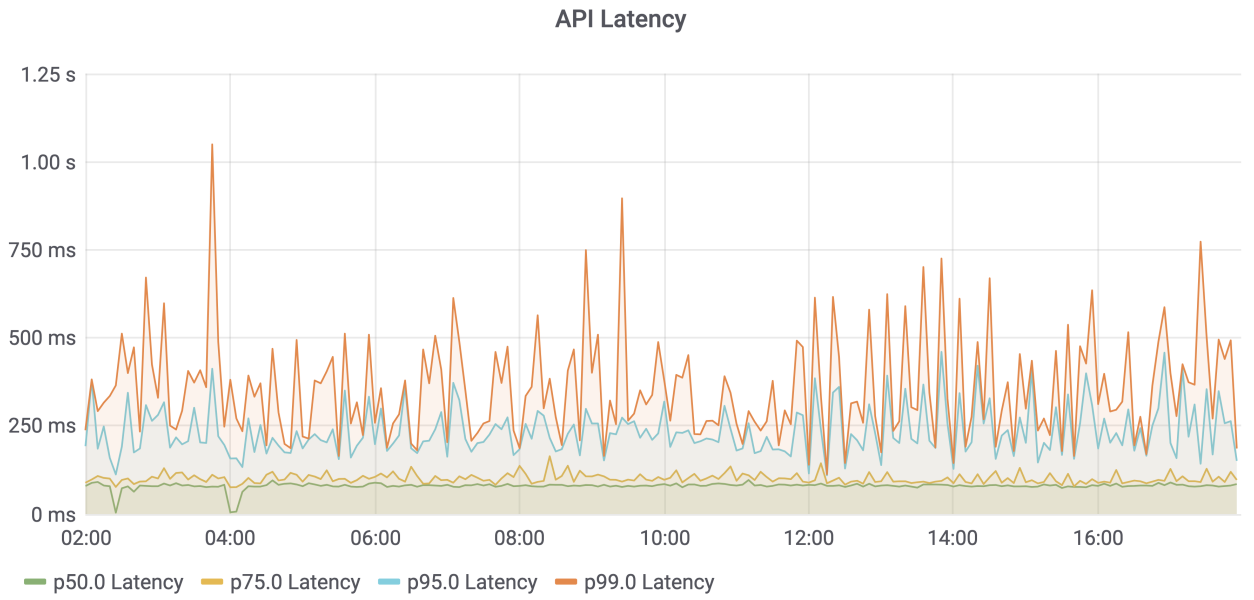

Figure 6: MIDAS API Latency Percentiles, used to assess performance. More specifically, this tells our team that 75 percent of all requests to the MIDAS API return within approximately 100 milliseconds or less. It also allows us to detect spikes indicating requests with high latency, enabling us to investigate and improve the performance of the system.

thousands of beacons that we use not only to spot errors before they happen on orbit, but also to improve our flight software before launch. Just as most large software companies have some sort of staging environment where they test their software, MIDAS enables MXL satellites to have an operational staging environment for development units that closely mirrors their flight environment.

NASA research states that of nearly 37% of CubeSat missions had never make contact with the ground and that 30% of missions do not complete their objectives.[27] To minimize failure through automated testing, MIDAS takes advantage of GitLab's[10] pipeline feature for continuous integration (CI), continuous deployments (CD), data backups, as well as general maintenance tasks. GitLab pipelines are traditionally used for running tests and deployments. Our software team uses these features to ensure that all MXL code passes rigorous tests before being deployed and that our running applications such as the MIDAS server stay up to date. Not only do we run unit tests of our server, ground station, and flight software, but we also test the integration of these three major components entirely in software, using Docker[9] containers to simulate the computing environment in which they operate. These end-to-end integration tests, similar to the practice of testing mobile app integration with web servers in the software world, allow us to validate our changes before we risk putting them into production on the ground or on orbit. In addition to using GitLab pipelines for CI/CD, we also use the scheduled pipeline feature as a distributed job scheduler. For example, every day, GitLab automatically runs a pipeline to back up the MIDAS server, including PostgreSQL[19] and Grafana.[14]

TEAM COMMUNICATIONS AND COLLABORATION

As previously mentioned, MXL uses Slack[11] not only to communicate, but also in combination with Grafana,[14] to send alerts when systems encounter errors or abnormal values. In addition, we use the Hasura[22] service's event trigger feature to listen for new packets and provide a live stream of all incoming telemetry in one of our Slack channels.

MIDAS also synchronizes upcoming passes over our ground station at the University of Michigan campus to Google Calendar, as seen in Figure 7. Furthermore, it sends a Slack notification to our workspace prior to each pass. These features increase operational visibility and allow users to easily plan their day around the passes they will operate.

In addition to improving operational communications, MIDAS helps MXL developers build new tools more quickly by automatically generating documentation for our services. We have also connected GitLab to Slack, ensuring that any errors in software development, deployment, or data backups are reported. These features not only make it easier for new members to understand existing services and illustrate our development process, but also help automate maintenance of our systems for experienced members.

CONCLUSION AND FUTURE WORK

The applications and tools that MIDAS has produced have undoubtedly transformed MXL satellite operations for the better. Furthermore, in the past several months, these collaboration tools along with our automation infrastructure and secure network access have allowed MXL to continue its research and satellite operations remotely as we face the COVID-19 pandemic. While we no longer have physical access to our hardware, our software has allowed us to continue to monitor and operate both our development and flight spacecraft.

We have demonstrated that current software practices and tools increase data visibility, team communication, and thus both operational and developmental capability for small satellites. These practices, including containerization, monitoring, decoupling of services, documentation, alerting, data analytics, and automation, have helped ensure that MXL remains at the forefront of small satellite development.

MIDAS will continue to be developed to further its goals of operational success through contemporary software system design with an increased focus on collaboration among lab members. Planned features include a remote collaborative ground station client, improved satellite state summarization and analysis capabilities, and eventually, the automation of our operations to ease the load of operations on our members. Our work clearly illustrates that modern software solutions that emphasize system autonomy and measurability have the power to dramatically improve small satellite operations and development.

References

[1] Missions and systems. *The Michigan eXploration Lab*, 2020.

[2] Satnogs. *SatNOGS*, 2020.

Figure 7: Google Calendar used to display upcoming passes over our ground station at the University of Michigan

[3] J.W. Cutler and P. Linder. A federated ground station network. *Proceedings of the 2002 SpaceOps Conference, Houston, TX, October*, October 2002.

[4] S. C. Spangelo, D. Boone, and J. Cutler. Assessing the capacity of a federated ground station. *2010 IEEE Aerospace Conference*, 2010.

[5] Elasticsearch: The official distributed search & analytics engine. *Elastic*, 2020.

[6] Felix Barnsteiner. Elasticsearch as a time series data store. *Elastic Blog*, April 2019.

[7] Z Mathe, A Casajus Ramo, F. Stagni, and L. Tomassetti. Evaluation of NoSQL databases for DIRAC monitoring and beyond. *Journal of Physics: Conference Series*, 664(4):042036, December 2015.

[8] Kibana: Explore, visualize, discover data. *Elastic*, 2020.

[9] Empowering app development for developers. *Docker*, 2020.

[10] The first single application for the entire devops lifecycle. *GitLab*, 2020.

[11] Where work happens. *Slack*, 2020.

[12] SpaceX. Stp-2 mission. *SpaceX*, April 2019.

[13] The world's most popular open source database. *MySQL*, 2020.

[14] Grafana: The open observability platform. *Grafana Labs*, 2020.

[15] High performance load balancer, web server, & reverse proxy. *NGINX*, 2020.

[16] oauth2 proxy. oauth2-proxy: A reverse proxy that provides authentication with google, github or other providers. *GitHub*, May 2020.

[17] What are containers and their benefits — google cloud. *Google*, 2020.

[18] Redis. *Redis*, 2020.

[19] The world's most advanced open source relational database. *PostgreSQL*, 2020.

[20] Lmenezes. lmenezes/cerebro: open source elasticsearch web admin. *GitHub*, April 2020.

[21] Jupyter notebook. *Project Jupyter*, 2020.

[22] Instant realtime graphql apis on postgresql. *Hasura GraphQL*, 2020.

[23] Metricbeat: Lightweight shipper for metrics. *Elastic*, 2020.

[24] Filebeat: Lightweight log analysis & elasticsearch. *Elastic*, 2020.

[25] Michael J Denney, Dustin M Long, Matthew G Armistead, Jamie L Anderson, and Baqiyyah N Conway. Validating the extract, transform, load process used to populate a large clinical research database. *International journal of medical informatics*, October 2016.

[26] Tools for fetching two-line elements sets. *FetchTLE*, 2020.

[27] M. Swartwout. Cubesats and mission success: 2016 update. *2016 Electronics Technology Workshop, NASA Goddard Space Flight Center*, June 2016.