SSC23-XIII-04

Extension of Cloud Computing to Small Satellites

Kathryn O'Donnell, Meghan Weber, Joy Fasnacht, Jeff Maynard, Margaret Cote, Shayn Hawthorne
Amazon Web Services
1320 SW Broadway #400, Portland, OR 97201; 503-703-3638
klodon@amazon.com

ABSTRACT

Time-to-insight is a critical measure in a number of satellite mission applications: detection and warning of fast-moving events like fires and floods, or identification and tracking of satellites or missiles, for example. Current data flows delay the time-to-insight on the order of minutes or hours, as all collected data must be downlinked in one or more contact windows, then transited over terrestrial networks to the location of the analytic software. Additionally, mission applications on spacecraft are often static: built prior to launch, they cannot rapidly adapt to changing needs based on these insights.

To reduce time-to-insight and provide a dynamic application update capability, Amazon Web Services (AWS), D-Orbit, and Unibap conducted a joint experiment in which we deployed AWS edge compute and network management software onto Unibap's SpaceCloud® iX5 platform for edge computing in space, integrated onto a D-Orbit ION Satellite Carrier launched into low-earth orbit (LEO) in early 2022.

In this paper, we present the results of this experiment. We will discuss the software specifics and network management capabilities we developed to write mission applications and update those mission applications on-orbit, and detail the process of mission deployment and modification, communications latency, and data volume reduction. We will also discuss how the space and satellite community can use this capability to deploy new applications, performing complex tasks and reducing time-to-insight, to cloud-enabled satellites immediately without needing to wait for a new launch.

INTRODUCTION

The AWS cloud is built to serve five primary user advantages: agility, to allow teams to experiment and innovate more frequently and quickly; cost savings, in a pay-for-use model that leverages the economies of cloud-scale infrastructure; elasticity, to easily scale to the needs of the mission; faster innovation, to focus on mission and not infrastructure; and global reach, providing the most extensive, reliable, and secure global cloud infrastructure.

Increasingly, satellite operators and space enterprises tell us that they want access to these cloud advantages on their space assets; in particular, they want the agility to modify mission applications on their satellites post-launch frequently, quickly, and securely, and they want the faster time to innovation, to more easily implement AI/ML capabilities to reduce downlink volumes and potentially enable autonomous operations. With this in mind, we developed a proof of concept that makes an in-orbit satellite look and operate like any other edge device in an AWS Virtual Private Cloud (VPC). This gives operators the ability to securely log into that edge device from the ground, start and stop processes, run ML inference on the satellite, securely copy data to the AWS cloud, and update mission software through standard software release methods. These are necessary components to quickly test and modify AI/ML capabilities, and additionally provide AWS's built-in security, user access controls, and user interfaces.

We successfully achieved the two primary goals of our experiment: perform inference and processing on data while in orbit using AWS edge capabilities, and change model configurations in flight using AWS edge service APIs. Below, we detail the specific software and hardware we used and the mission scenarios we tested to achieve those goals.

SOFTWARE

Communications

To achieve our goal of looking and operating like any other edge device in AWS, we wanted to use existing AWS capabilities. AWS services are designed with an API-first methodology that allows communication with those services via the internet, specifically over IP links, so we first needed to develop a way to manage IP traffic across an RF link. With a LEO satellite, we also needed to be able to handle intermittent connectivity on a set time schedule, which is not a common use case in our terrestrial applications.

To do this, we built custom network proxies with added resiliency, managing the AWS-specific IP traffic over the satellite's physical RF link and creating a connection between terrestrial AWS services and AWS edge services on the satellite, abstracting the connection so end users don't need to spend time managing communications. One end of the connection is in a terrestrial VPC, with the other installed onto the satellite.

We called the solution we created for the AWS cloud-based end of this network pipe the space gateway. It runs on an Elastic Compute Cloud (EC2) instance, a virtualized computing environment, in an AWS VPC. For the satellite, we created software we called the space edge engine, which runs on the edge computer onboard, in this case the Unibap SpaceCloud® iX5 platform for edge computing in space.

During a pass, once the signal is acquired, the space edge engine on the satellite connects to the space gateway with a full Transport Layer Security (TLS) handshake. Users on the ground can then send instructions to the AWS assets on the satellite through the engine-to-gateway IP tunnel, and responses come back through the tunnel. If a request or response is too large to be sent before the end of signal, the gateway will cache the data, close the connection cleanly, and attempt to resume the data transfer during the next connection window. The gateway keeps track of all data transferred, and even if the signal is lost before closing the connection, it will smoothly pick up that data transfer the next time connection is re-established.

Security

At AWS, we build our applications, and recommend all customers build their applications, according to our published AWS Well-Architected Framework. This framework consists of six pillars, one of which is security. Our focus on security takes advantage of AWS technologies to protect data, systems, and assets, and we built our software for this experiment to adhere to the best practices of the security pillar.

We utilized the AWS Identity and Access Management service to ensure that AWS services terrestrially, as well as space edge processing, can only be accessed by the proper users using the proper roles. We implemented Multi-Factor Authentication as a second factor for defense in depth. We ensured that the IP traffic that flows through our gateway-to-edge tunnel between space and ground is cryptographically secured using AES-256 encryption in the TLS protocol, and encrypted the data that is stored within the AWS cloud by making using of private keys created and rotated within the AWS Key Management System (KMS). We also made use of logging and monitoring systems that

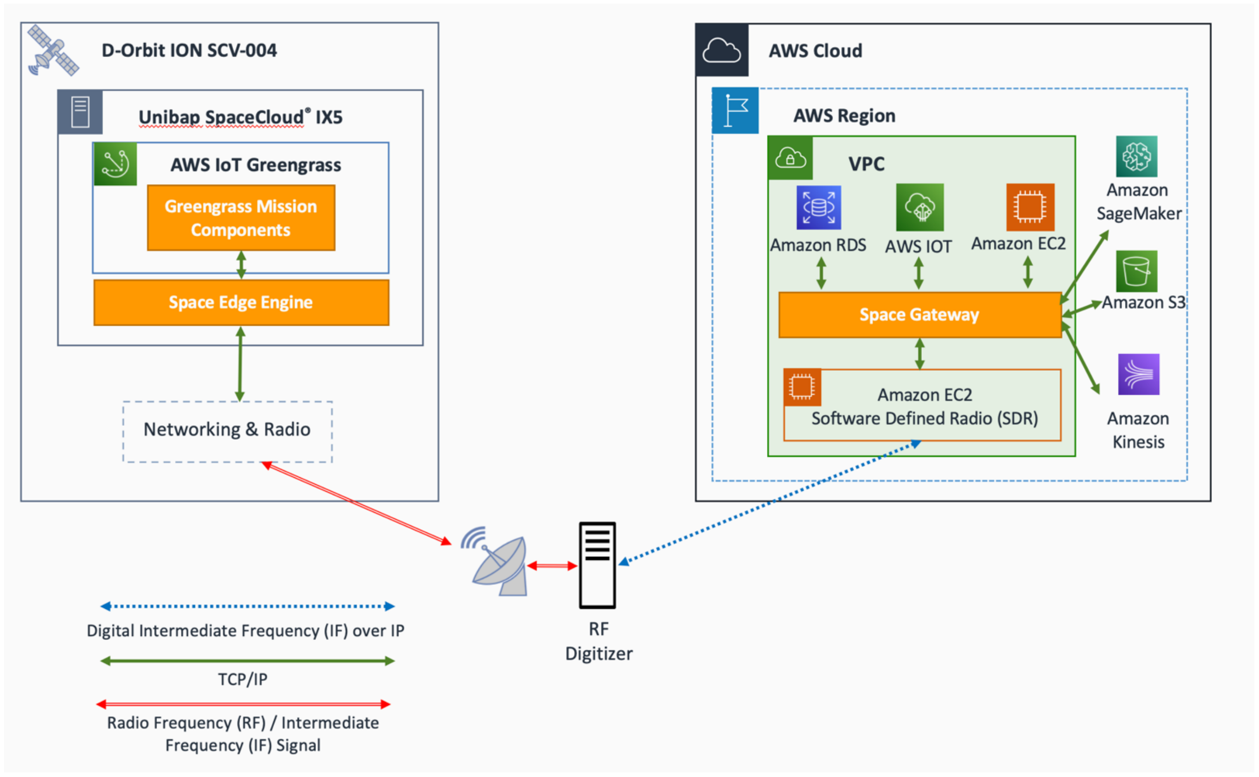

Figure 1: A representative architecture of the extension of the AWS cloud to a small satellite.

come by default to all services within the AWS cloud such as logging of all API and network traffic, and automatic alerting via the Amazon CloudWatch system.

Processing

With the networking ability to communicate between AWS edge processing onboard and the AWS cloud in place, building on top of the secure foundation taking advantage of existing cloud services, we then examined options for how to run AWS services on the space edge computer. We knew we would be in an intermittently-connected state, we wanted to run containerized applications, and we wanted to modify those applications in-flight.

With those requirements in mind, we opted to implement AWS IoT Greengrass on the space edge engine. Greengrass is an open-source Internet of Things (IoT) edge runtime and AWS cloud service that helps users build, deploy and manage IoT applications on devices. AWS IoT Greengrass inherently already had a lot of the capabilities that we wanted: it is lightweight and meant for edge devices that can go through long periods of disconnection, which a satellite could certainly be defined as; it is highly portable to any X86 or ARM processor; and it gives the capability to upload and deploy custom components that can run without needing an always-on connection back to the AWS cloud. The custom components give us flexibility in the type of applications we could run – these can be containers, micro-services, or even scripts. AWS IoT Greengrass also manages the memory and compute consumed by the components, so we didn't need to create additional capabilities to avoid overloading the space edge computer. Finally, Greengrass was a good choice for our requirements as it provides a lightweight level of container orchestration, including dependency adherence, monitoring, re-starting, and life-cycle management through Over-the-Air (OTA) updates that can be managed from our AWS VPC.

Additionally, AWS IoT Greengrass can support machine learning inferencing, including the AWS AI/ML service, Amazon SageMaker. SageMaker is a managed service to prepare data and build, train, and deploy machine learning models, is well-known within the data science and machine learning engineering communities, and is something we often use in-house to build models. SageMaker supports a variety of ML frameworks including TensorFlow, PyTorch, MXNet, and XSBoost, and AWS IoT Greengrass inference support allows models built with SageMaker (or other AI/ML on-site tools) to be deployed to Greengrass and make inferences on the edge device without additional tuning.

AWS IoT Greengrass requires ARM or x86_64 architectures, 256MB of disk space, 96MB of RAM, Java Runtime Environment (JRE) 8+ and GNU C Library version 2.25+. Disk and RAM needed by the custom components is in addition to these requirements.

HARDWARE

Satellite

For this experiment, our main requirement was an IP-based radio to support our space gateway-to-engine tunnel. We were not testing communication data rate or sensor capability; we wanted to focus on our capability to communicate between applications and AWS edge service APIs on the satellite and AWS assets on the ground, along with our capability to run ML inference on the satellite.

We were happy to collaborate with D-Orbit, who built and operated the host satellite, the D-Orbit ION Satellite Carrier SCV-004, *Elysian Elenora,* and Unibap, who built the host space computer, the SpaceCloud® iX5 platform for edge computing in space. D-Orbit's ION Satellite Carriers have a customizable 64U satellite dispenser, and can both deploy CubeSats as well as directly integrate payloads onto the platform. SCV-004 launched into low-earth orbit (LEO) in early 2022 on the Falcon 9 Block 5, and the integrated payload is shared by multiple customers, one of which is AWS.

Space Edge Computer

Given the requirements of AWS IoT Greengrass and the types of machine learning inference payloads we wanted to run, Unibap's SpaceCloud® iX5 platform for edge computing in space was a fit for our needs. It has an embedded x86 AMD G-series System on Chip (SOC), up to 240GB of SSD storage and 2GB DDR3 ECC RAM (CPU/GPU) coupled with accelerators for neural network inference. The computer is designed with a heterogeneous architecture with multiple execution hosts and access to high-speed IO from sensors and peripherals.

MISSION SCENARIO TESTING

Raw satellite imagery produces large amounts of data which must be downloaded in limited connection windows for LEO satellites. To account for this, most satellite imagery is tasked for areas of the Earth that are suspected to be of interest, with all post-processing taking place terrestrially after the raw imagery data is downloaded. The downside of this is that anomalous events may not be captured, and the downloaded data

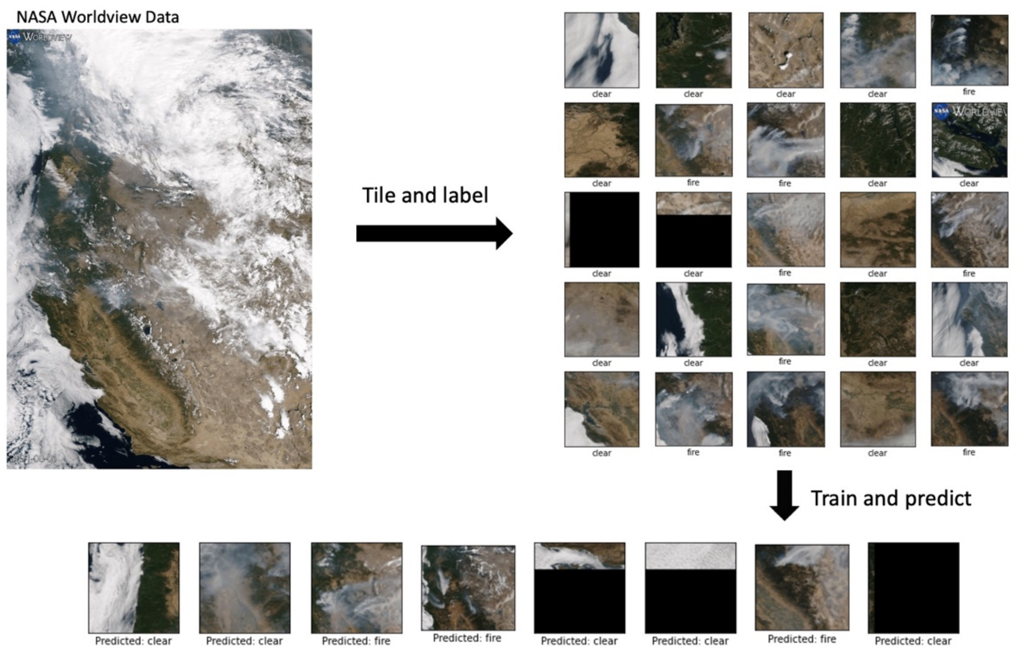

Figure 2: NASA Worldview 1300x1870 images are split into overlapping 256x256 tiles, classified as fire or clear, then used to train a TF-Lite image classification model using EfficientNet-Lite0.

may be of low information value due to natural phenomena like cloud cover.

The natural remedy to this downside is to perform image processing and object detection on the satellite. We wanted to raise the bar a little further, and not just process and detect, but also create a capability to update that processing and parameters to machine learning inferencing in-orbit to accommodate changing mission objectives.

With that in mind, we identified two primary goals with our experiments: perform inference and processing on data while in orbit, and change model configurations in flight. We tested four common mission scenarios that customers had indicated were of high importance: fire detection, cloud masking, image processing, and sensor outlier detection.

Fire Detection

In this scenario, we wanted to examine images of the Earth's surface and alert when indications of a fire were present. Using labeled tiled images from NASA Worldview data[1], we used Amazon SageMaker to train an EfficientNet-Lite image classification model on TensorFlow-Lite, a version of TensorFlow optimized for mobile, embedded, and edge devices [Figure 2]. In particular, we used EfficientNet-Lite0 due to low latency and small model size. The model we built achieved 95% accuracy on the testing dataset.

As we weren't testing the sensor with this model (and we would not wish for an actual fire), we loaded images to the satellite pre-launch to run our inference on. We packaged our model and our inference code as AWS IoT Greengrass custom components. We defined the inference interval as a configurable value, and set it to once per minute. The deployed model size was ~2.5MB, with inference latency in the range of 50-100ms.

With this scenario, we were able to achieve our goal of running inference on the satellite in orbit with AWS edge capabilities. In future tests, we plan to change configuration values as well as use processed imagery from the on-board sensor.

Cloud Masking

The presence of clouds in Earth images can result in unusable images occupying download bandwidth, increasing costs and potentially reducing the amount of information we can obtain from a satellite pass. In this scenario, we created a custom model using Amazon SageMaker that identified cloud-obscured portions of an image based on a luminance threshold on pixels, with an additional test for contours of the identified portion to except rectangular shapes as those were likely to be human-made objects. We then mask the cloud-covered segments of the image to reduce file size [Figure 3], with configurable settings for determining the type and amount of cloud cover to mask and download.

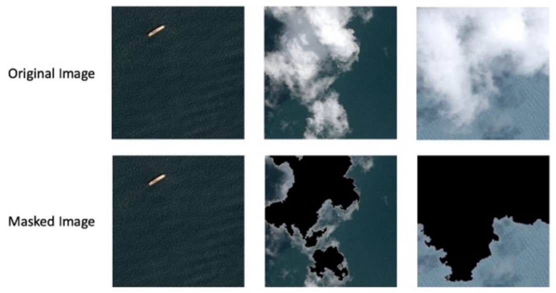

Figure 3: Clouds are masked with full black #00000000, allowing greater compression of the image.

We also successfully modified the thresholds for cloud cover using the AWS IoT Greengrass configuration update capability on the satellite to deploy new configuration files with the modified thresholds during a contact window.

With this scenario, we were able to achieve both our goals: we performed inference and processing on the satellite in orbit with AWS edge capabilities and changed model configurations in flight using built-in AWS edge service capabilities and APIs.

Image Processing

New types of imagers and cameras are being employed and launched into orbit at an increasing rate. Often, the new imagers have increased spectral range and resolution, resulting in the onboard generation of very large image datasets. If the images collected are of novel type and resolution, then computer vision models trained on older image types will not perform optimally. To address this, we wanted to effectively compress these images for download but maximize the amount of variance preserved. We could then subsequently label and train computer vision models from the compressed images terrestrially, which would then be uploaded to the satellite for optimal performance on the satellite's imagery.

In this scenario, inspired by published technical work[2], we processed images captured by a micro-opto-electro-mechanical system (MOEMS) spectral imager with up to 49 bands. We wanted to project the images from 49 to 3 bands, map those 3 bands to RGB [Figure 4], and encode as JPEG. This allows us to capitalize on efficient near-lossless JPEG compression.

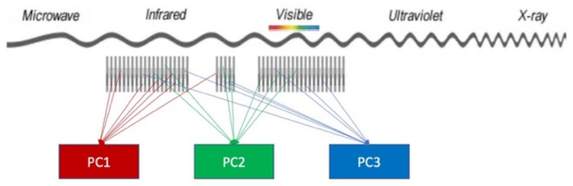

Figure 4: Multiple bands are efficiently compressed into three Principal Components, which are linear combinations of the original bands, selected to capture as much variance as possible.

Using Amazon SageMaker, we performed Principal Component Analysis (PCA) on a set of 49-band images similar to what would be available from a satellite sensor, to project the information from 49 bands down to 3 composite bands. We packaged the PCA model as an AWS IoT Greengrass component and deployed it to the satellite.

The three composite bands capture from 80 to 90% of the variability present in all 49 bands, and an image can be processed in the single millisecond range. We included JPEG encoding quality thresholds as a configurable value in the Greengrass component configuration, which allows operators to set the balance between file size and file integrity based on their mission parameters.

With this scenario, we were able to achieve our goal of performing processing on the satellite in orbit with AWS edge capabilities. In future work, we anticipate that we can retrain the PCA model in orbit using images provided by the satellite sensor on-board.

Sensor Outlier Detection

For this scenario, we wanted to automatically determine when internal sensor readings from the satellite go out of nominal. We used Amazon SageMaker and sklearn to implement the Isolation Forest Algorithm[3], which is an unsupervised method that creates binary trees to identify anomalous data points, without needing prior knowledge of nominal limits. We packaged this as an AWS IoT Greengrass custom component and deployed to the satellite.

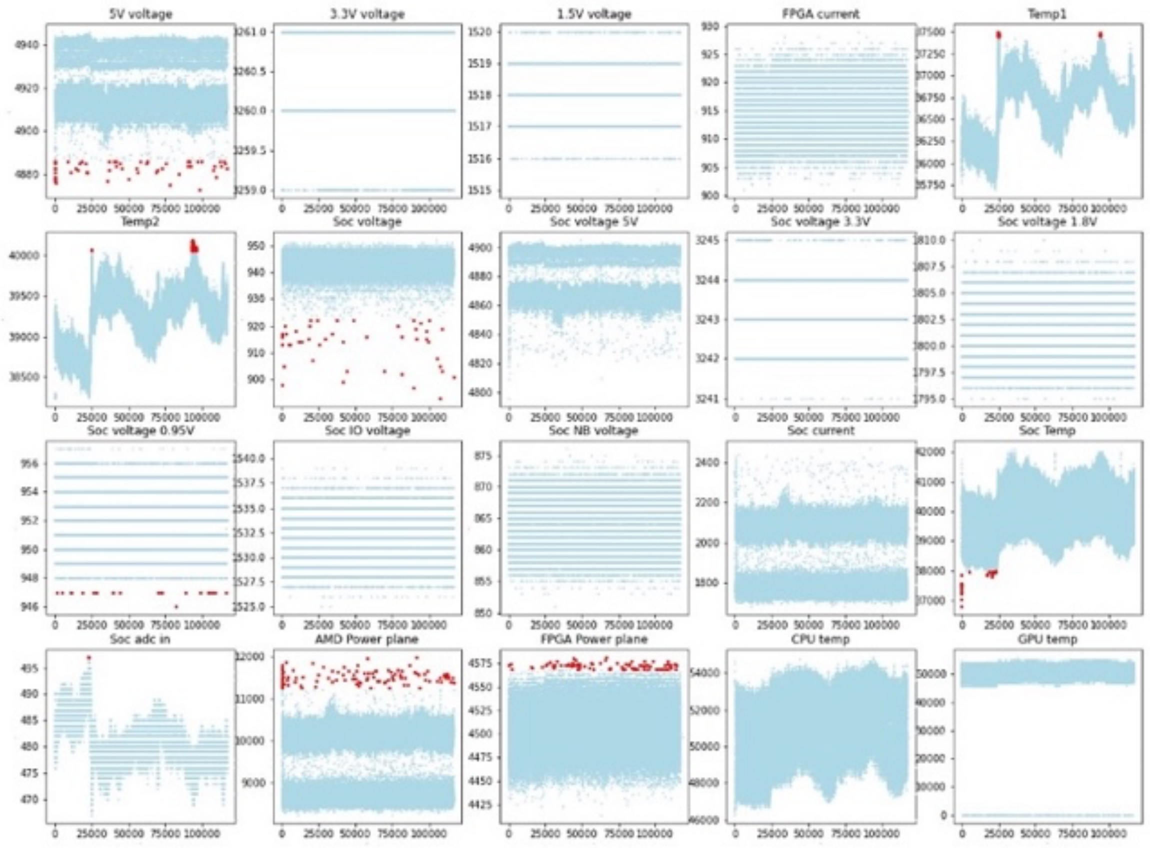

Figure 5: Illustration of outlier detection in time series data of sensor readings, with outliers marked in red.

In flight, we fed health data to the model, including readings such as CPU temperature, GPU temperature, and FPGA current [Figure 5]. The model identifies readings that are beyond a configurable threshold of number of branches created, and the threshold can be adjusted in Greengrass configuration. The configuration can also set the training windows for analysis.

This scenario achieved our goal of processing using AWS edge capabilities, and also demonstrated the capability to use data produced by the satellite in orbit. We designed the component to be able to retrain on-board, and our future work includes testing that capability.

FUTURE DIRECTIONS

In addition to the specific future work described above, we plan to work across AWS and our customers, including those with mission areas in agriculture, maritime, energy, automotive, and natural disaster response, to identify use cases that can be served by on-orbit processing, develop the components to execute that processing, and deploy and run those components.

Two additional use cases we have already identified are in the area of autonomous operations. The first is a tip-and-cue system, in which we use computer vision or other modalities of ML inferencing models to identify, with low latency, objects of interest in sensor data; once identified, tips are sent to other satellites to track the object and maintain continuous sensor coverage. The other use case is in onboard planning and scheduling based on tips or other external factors.

We are also working to improve the software we've developed for this experiment. We are driven by the democratization of space, and the extension of familiar AWS tools and capabilities to space: we want to enable any software developer to work on space applications, no longer constrained to those few with niche knowledge of protocols and systems. With this concept, developers can dynamically uplink and update applications based on changing mission needs, run advanced algorithms like sensor fusion, execute containerized applications, and trigger serverless compute jobs. With the flexibility that AWS IoT Greengrass provides, developers writing code in a variety of commonly-used languages such as Java, Go, PowerShell, Node.js, C#, Python, and Ruby can write satellite mission applications, not just traditional embedded software languages.

By extending cloud computing to small satellites, we also enable flexibility for mission software in a number of other ways. These extensions of the AWS cloud to space provide for the ability to use normal software CI/CD (Continuous Integration/Continuous Deployment) methodologies for on-orbit applications, transition existing software applications directly from ground to space, test software mission applications in a completely virtualized and inexpensive environment in the cloud, reuse application software between missions, and update mission software and configurations without impacting critical command and data handling systems and flight software. The AWS cloud also provides a new measure of operational and tool consistency, giving the ability to manage and automate terrestrial and space deployments with the same tools, use tested AWS security and infrastructure management, set and manage access permissions across space and ground, and sync data between space and ground using normal APIs.

ACKNOWLEDGEMENTS

We would like to thank all the people who provided support during the design, development, testing, and operations of our experiment. This includes our colleagues in the AWS Space Services Team, the AWS Aerospace & Satellite team, and the AWS Professional Services team, with particular appreciation to Nic Ansell and his tremendous innovation and contribution, and Kara Yang and Dan Iancu, our formidable Data Scientists. We also would not have been able to perform this experiment without our partners at D-Orbit and Unibap.

REFERENCES

1. Imagery from the Worldview Snapshots application (https://wvs.earthdata.nasa.gov), part of the Earth Observing System Data and Information System (EOSDIS).
2. Chen, G. and S.-E. Qian, “Evaluation and comparison of dimensionality reduction techniques and band selection”, *Can. J. Rem. Sens*. 34, no. 1, pp. 26-36, 2008.
3. Liu, F.T., K.M. Ting and Z. -H. Zhou, "Isolation Forest," *2008 Eighth IEEE International Conference on Data Mining*, Pisa, Italy, 2008, pp. 413-422, doi: 10.1109/ICDM.2008.17.