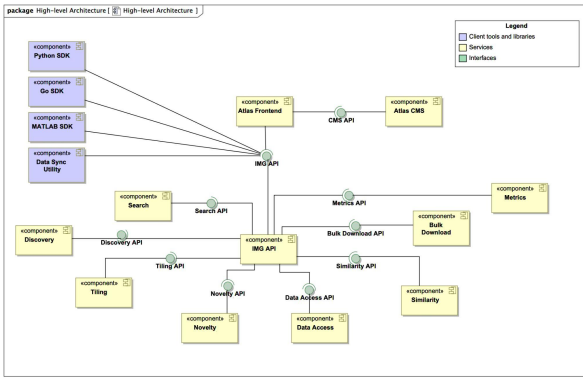
CLOUD PROCESSING OF PDS ARCHIVAL PRODUCTS WITH AMAZON WEB SERVICES, KUBERNETES, AND ELASTICSEARCH. Kevin M. Grimes II[1], Rishi Verma[1], James Michael McAuley[1], Tariq Soliman[1], Anil Natha[1], Zachary M. Taylor[1], [1]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA (kevin.m.grimes@jpl.nasa.gov).

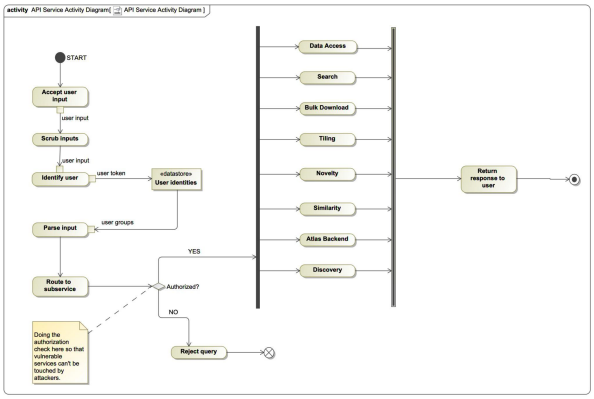
Introduction: As cameras and other spacecraft instruments improve, the data they produce becomes richer in quality and, inevitably, larger in volume. Processing these data in a timely fashion via traditional means becomes sluggish, inefficient, and expensive. In order to address challenges posed by next-generation missions taking place in our solar system, a reconsideration of processes used to process these data is required.

The Planetary Data System (PDS) Cartography and Imaging Sciences Node (IMG) retains hundreds of terabytes of data, collected from dozens of missions and spacecraft over as many years. Among the responsibilities of IMG is to make the data not only accessible by the public, but also searchable. By leveraging PDS Engineering Node's (ENG) [1] software, Amazon's [2] cloud offering AWS [3], and the open-source container orchestration platform Kubernetes [4], IMG has made strides to provide a rich search experience of its data for the community.

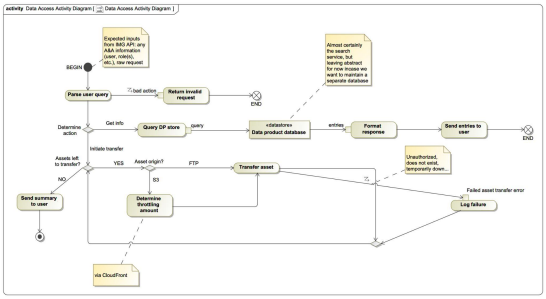
Architecture: IMG follows the microservices pattern for its backend architecture, which enables "rapid, frequent and reliable delivery of large, complex applications" [5] in the form of small, individual services. These services may be developed independently from one another. In IMG's case, they do not run on traditional servers, but instead on AWS's EKS [6] service, which provides Kubernetes clusters using Docker [7] as a service. Communication between these services is achieved via APIs, and common data is stored in various databases, including Elasticsearch [8] and DynamoDB [9].



IMG API. Central to the architecture is the IMG API, a lightweight application interface responsible for interpreting general user requests, authenticating the user who made them, determining the user's access via role authorization, and routing the request to the target backend service. By forcing all requests to the system (either from tools internal to IMG, or from outside entities) to go through this central location, we can easily revoke access to entities who abuse our services. For this component, IMG uses API Gateway [10], another AWS offering that interprets OpenAPI 3.0 [11] specifications and allows routing to other services IMG maintains in the cloud. Additionally, API Gateway offers token-based authentication, which IMG uses.



Data Access. Previously, IMG has allowed users to download its data holdings via HTTPS directly from our servers. However, IMG does not intend to continue holding all of its data on-premises; instead, it is exploring gradually moving its holdings to Amazon's Simple Storage Service (S3). Of course, with movement to the cloud comes a variety of concerns, including cost. A requirement of the new system is that IMG be able to regulate users who abuse our system and, consequently, rack up a large bill, have their download rates be curtailed. This is enabled by the Data Access API, a lightweight application inspired by TEA [12]. The Data Access Application Programming Interface (API) abstracts away the physical location of files, allowing them to be stored in various different S3 buckets and actual servers. Additionally, access to these files is controlled by tokens, which the IMG API manages. Finally, users who abuse the system are curtailed, and risk having their tokens revoked altogether.



Search. Searching IMG's holdings is a capability that has been in place for years; however, several improvements are being made in various regards to the system. First, every file in our archive is being indexed, rather than just image files. This allows users to query programmatically for all the files in a virtual directory, for example. The entire label is being indexed into Elasticsearch for every product in the archive, using tools like Harvest [13] for PDS4, and Logstash [14] + PVL [15] for PDS3.

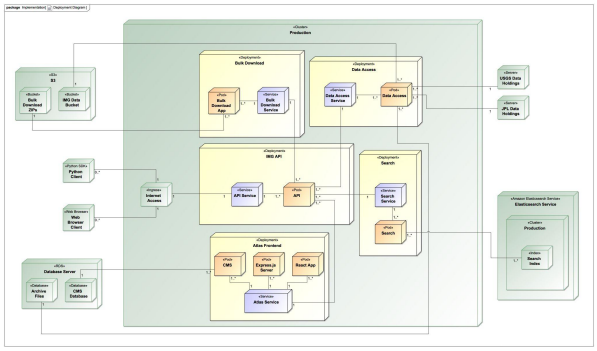
Of course, mission-specific nuances exist in both the PDS3 and PDS4 cases. For example, the fields "longitude" and "latitude" may imply the location of the spacecraft for landed missions, but for orbital missions, may instead describe the location of the target being captured. Multimission tools like the Atlas [16] require these discrepancies to be resolved, otherwise search across all of them would result in inconsistencies. Toward this end, IMG separately converts the label contents into a "common" structure that maps between PDS3 and PDS4, while allowing users to query for the individual label fields as well.

Deployment: The majority of the components described above are packaged as Docker images, which offer isolated runtime environments and virtually infinite scalability. Connecting containers to each other via their APIs is a challenging task, and requires additional tooling to orchestrate properly. Additionally, scaling (increasing the number of instances of) individual Docker containers to match increased consumer requests is a non-trivial task.

To this end, we leverage Kubernetes, an open-source container orchestration framework. By defining individual services within the context of Kubernetes-native resources, we can easily connect different components to each other via their APIs, regardless of how many replicas for each exist. The applications themselves and their corresponding definitions in Kubernetes are packaged into Helm [17] charts, which enable rapid deployment to the cluster.

An advantage of using Kubernetes and Docker for application deployment is that the system can be run effectively anywhere: as long as the target system can run Kubernetes (and, therefore, Docker), it can run the services. With few exceptions, there is little to no dependency on the host operating system; Docker containers can operate in their own contexts without worrying about the larger context. In IMG's case, this involves running our development environment on hardware at JPL, but our production environment out of AWS.

Scaling and upgrading our services with no downtime is another requirement of the system. Kubernetes enables this with the help of a few additional technologies: Flagger [18], FluxCD [19], and Istio [20]. With the help of these tools, "canary" rollouts [21] are enabled which slowly redirect traffic to new, upgraded versions of services. If the system detects that a significant number of requests are failing, it cancels the rollout and automatically redirects traffic to the previous instance. If there do not appear to be any problems with the upgraded instance, however, all traffic is redirected to the new instance and the old instance is removed.



Future work: Due to the rapidly changing nature of the technologies described above, IMG is constantly learning new and improved design patterns and technologies. Additionally, IMG hopes to interface with PDS Engineering Node's PDS API [22]–a centralized API that routes traffic to different nodes.

References: [1] https://pds-engineering.jpl.nasa.gov. [2] https://amazon.com. [3] https://aws.amazon.com. [4] https://kubernetes.io. [5] https://microservices.io/. [6] https://aws.amazon.com/eks/. [7] https://docker.com. [8] https://www.elastic.co/enterprise-search. [9] https://aws.amazon.com/dynamodb. [10] https://aws.amazon.com/api-gateway/. [11] https://swagger.io/specification/. [12] https://github.com/asfadmin/thin-egress-app. [13] https://pds-engineering.jpl.nasa.gov/development/pds4/5.0.0/ingest/harvest/. [14] https://www.elastic.co/logstash. [15] https://pypi.org/project/pvl/. [16] https://pds-imaging.jpl.nasa.gov/search. [17] https://helm.sh. [18] https://flagger.app. [19] https://fluxcd.io. [20] https://istio.io. [21] https://semaphoreci.com/blog/what-is-canary-deployment. [22] https://github.com/NASA-PDS/pds-api.