astrophysics data system

https://ui.adsabs.harvard.edu/

Are you a front-end developer? We are hiring!

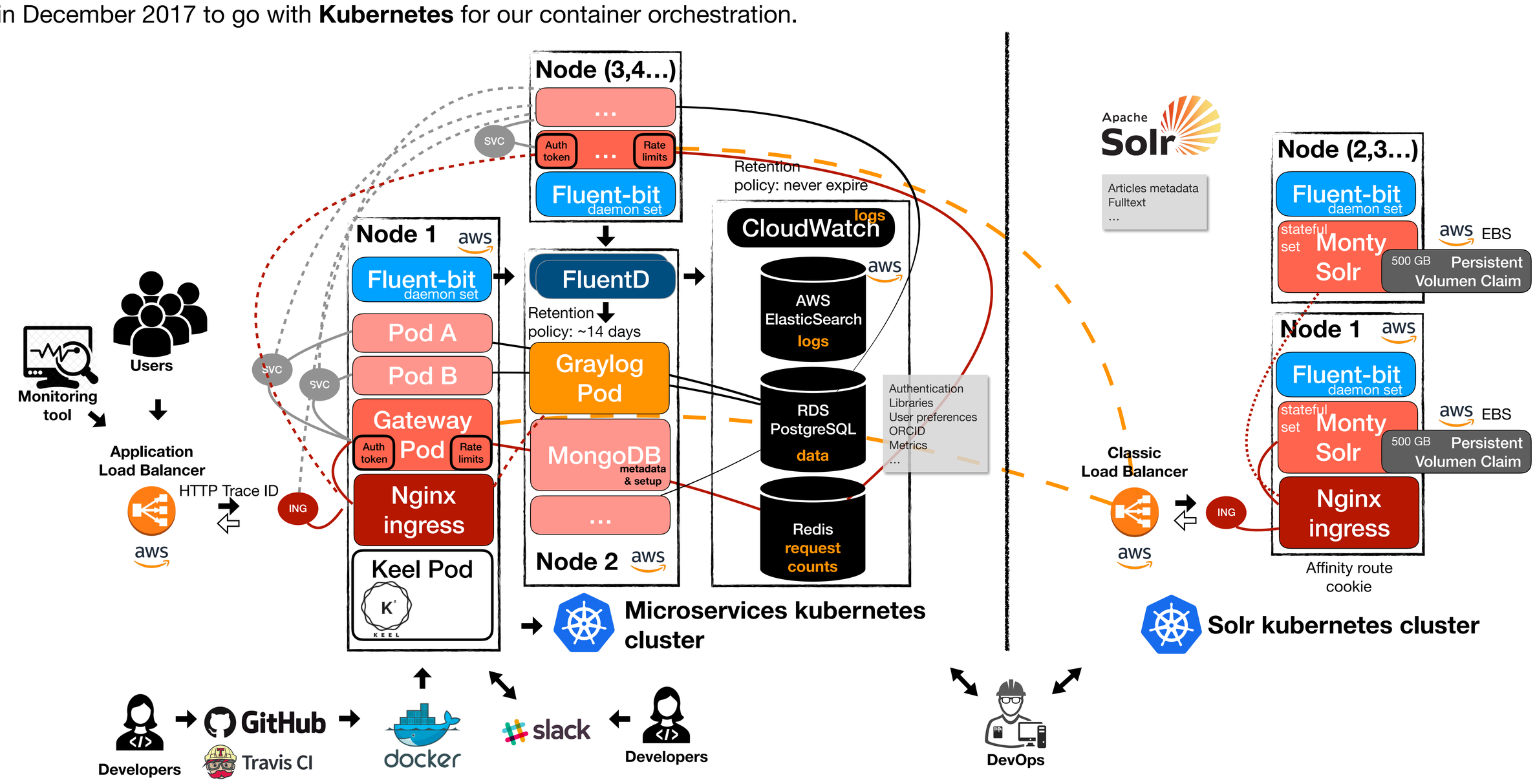
Fundamentals of effective cloud management for the new NASA Astrophysics Data System

Sergi Blanco-Cuaresma[1] and the ADS team

[1] Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA 02138, USA.

The new NASA Astrophysics Data System (ADS) is designed as a service-oriented architecture (SOA) that consists of multiple customized Apache Solr search engine instances plus a collection of microservices, containerized using Docker, and deployed in Amazon Web Services (**AWS**). For complex systems, like the ADS, the loosely coupled architecture can lead to a more scalable, reliable and resilient system if some fundamental questions are addressed. After having experimented with different AWS environments and deployment methods, we decided in December 2017 to go with **Kubernetes** for our container orchestration.

Defining the best strategy to properly set-up Kubernetes has shown to be challenging: automatic scaling services and load balancing traffic can lead to errors whose origin is difficult to identify, monitoring and logging the activity that happens across multiple layers for a single request needs to be carefully addressed, and the best workflow for a Continuous Integration and Delivery (CI/CD) system is not self-evident.



Monitoring

Making sure the whole system is healthy and responding to users' requests is a priority. We developed a custom **monitoring tool** that emulates users' behavior (e.g., executing searches, accessing libraries, exporting records, filtering results) and **alerts** us to unexpected results or errors via slack. This emulation happens every five minutes. Historical data is also accumulated and **daily reports** are generated to measure trends and improvements that could be correlated with microservices updates or infrastructure changes.

Logging

Responding to a single user request may involve multiple microservices (e.g., libraries, solr search service) and different data requests (e.g., bibcodes in a library, records in solr). At the very first step, when the user request reaches the AWS application load balancer, a **trace identifier** is attached to the HTTP request and we propagate it for each required internal request inside our infrastructure. All the microservices output logs to stdout, including key information such as the trace identifier and the user's account. Logs are captured by **fluent-bit** and distributed to **Graylog** and **CloudWatch** via **fluentd**.

Deploying

The deployment of new microservice releases is automatically managed by **Keel**. The developers push new commits to **GitHub** and/or make releases, which triggers unit testing via **Travis** continuous integration and image building via **docker hub**. When a new image is built, Keel deploys it directly to our development environment (each pushed commit) or to our quality assurance environment (each new release). Confirmation to deploy a release in production is provided via **slack**, where Keel reports its operations and reacts to developers approvals.

Future plans

Several services still require manual intervention in order to deploy new releases, Keel does not cover all our development cases and we are working on a new custom tool to meet our needs (after discarding other tools available in the market). We seek to fully automate the deployment process, while ensuring traceability and easy roll-backs based on automatic functional tests from our monitoring tool. Additionally, to reduce the required resources and simplify operations, we will evaluate other engines for searching through our logs such as Kibana via ElasticSearch (provided by AWS).

Sergi Blanco-Cuaresma
sblancocuaresma@cfa.harvard.edu
http://www.blancocuaresma.com/s/

Accomazzi, Alberto; Kurtz, Michael J.; Henneken, Edwin; Grant, Carolyn S.; Thompson, Donna M.; Chyla, Roman; McDonald, Steven; Shapurian, Golnaz; Hostetler, Timothy W.; Templeton, Matthew R.; Lockhart, Kelly E.; Bukovi, K.