TTS NextGen Alerts

James Cameron, CS
Kate Hanson, CS
Muhammad Umair, CS
Andrew Wang, CS

Sponsor: Tufts Technology Services

Background

- Tufts Technology Services (TTS) handles technology solutions for the entirety of Tufts campus, including wifi networks and many on-campus devices.
- TTS has traditionally used enterprise services for alerts and monitoring in relation to their user logs.
  - Premium Elasticsearch services, specifically Elastic Alerting.
- Within the past 2 years, TTS has changed to a free license and as a result has lost its previous alerting capability.
  - Open source projects exist, but open source had supportability issues with Kibana and Elasticsearch.
- Effective and efficient alerting is a core part of how TTS successfully maintains a proper security posture.
  - Without such alerting features, it is easy for potential issues to go unnoticed.

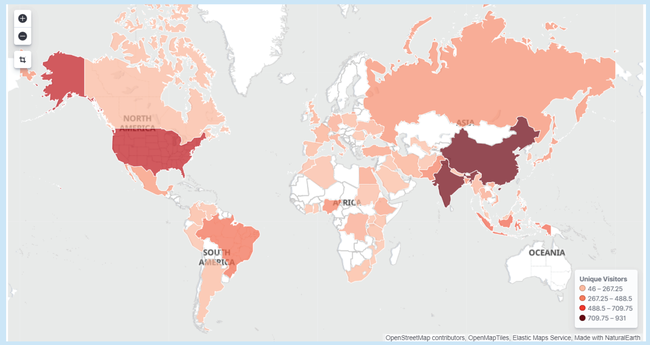


Figure 1: Unique user heatmap for TTS network over the course of a month, showing both the variety of user of the TTS network and the scale of the information that TTS logs store.

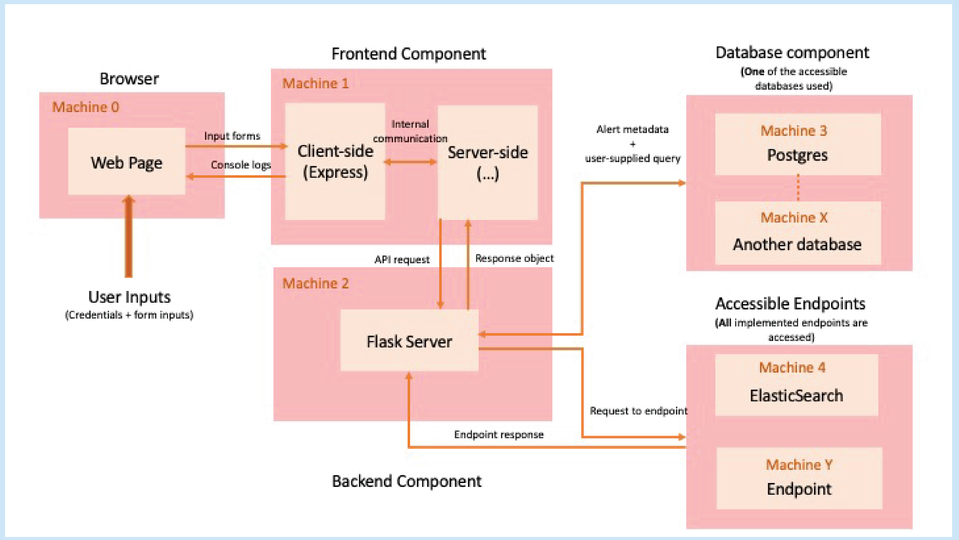
Goal

To design and develop an alert system that interfaces with TTS's ElasticSearch clusters; the alert system will have a user-friendly frontend, a moddable backend, and will incorporate standard enterprise security features.

Observations

- Implemented Features
  - Input format by query type
  - Single Sign-On (SSO) metadata hosting
  - Internal alert manager
  - HTTPS hosting
- Next Steps
  - SSO integration
  - Alert histories
  - Flask server in production mode
  - "Graceful" start, stop, and restart of backend server

Engineering Diagram



Reflection and Conclusion

Industry Takeaways

- Security concerns are a lot more serious on the enterprise level.
  - Previous projects, even within webdev, did not prepare us for real security requirements.
- Scalability is needed for data and usage of this magnitude.
  - Many design decisions were made around supporting multiple users at once or speed optimizations.
- Work needs to be presented to non-technical people when working with an organization.
  - Getting our descriptions high-level enough to be useful took time.

Project Takeaways

- The scale and scope of our project required professional enterprise-level knowledge and comprehensive understanding of non-technical specifications. We were unprepared for either and as a result did not meet our goals.
- Modular design allows for easier modification and extensibility for future development.
  - The API of the flask server's internal modules will remain relatively consistent.
- Knowing which features to prioritize was difficult and relied on multiple influences.
  - We had to juggle immediate asks from our sponsors while considering what order of implementation made the most sense.

Technologies

Design

Initial Frontend: React

Hosted on Node server

Advantages

- Modular design allowed for code reuse.
- Lots of supporting documentation.
- Easy integration with Firebase.

Disadvantages

- Not easily SSO-compatible.

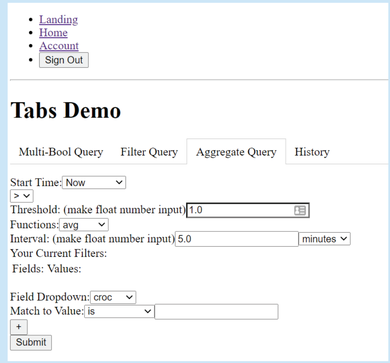


Figure 2: React UI demo

Final Frontend: Express

Hosted on Node server

Advantages

- Easy SSO integration.
- Lightweight, uses base HTML.

Disadvantages

- Less modular in it's design.

Figure 3: Express UI demo

Backend: Flask

Server written in Python

Advantages

- Large collection of usable libraries.
  - Elasticsearch included.
- Deep knowledge within group.

Disadvantages

- Difficult to convert to production.

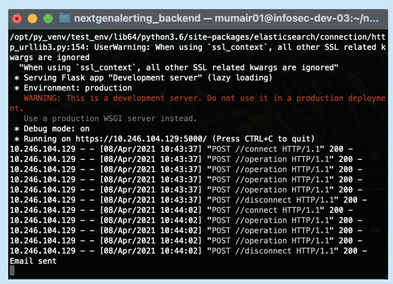


Figure 4: Backend demo