Team Intellect's Approach

1. Prototype Name

Adverse Drug Effects Research System (ADERS)

2. Use Cases addressed by ADERS

After interviewing few sample users, we came up with two use cases that would be very useful to the end users:

- I was told to take certain drug, but I am not sure if I should take that drug. I would like to find out adverse effects of the drugs suggested to me and confirm that I have all the correct label and warning information for the drugs I am using.

Additionally our users also stressed importance on maintaining privacy of their search.

3. Introduction

Our site is designed to provide users with ability to research on adverse side effects of the branded and generic drugs and medical devices. While building the prototype, Adverse Drug Effects Research System (ADERS), Intellect followed all the 13 plays from the US Digital Services Playbook to ensure we deliver successful Agile Service prototype to GSA 18F.

Below are the details of each play and how we followed them and customized to 18F's requirements.

4. Requirements Development Phase

This section discusses how we followed following play:

- *Play 1: Understand what people need*

Our approach began with studying the prototype requirements and performing research of the APIs provided by open.fda.gov. After reviewing the APIs we identified some sample users who might use that information and conducted interviews and meetings. GSA was also identified as a stakeholder and based on the requirements in the RFQ.

After these meetings, our team developed user stories in Taiga.Io that addressed following information:

- Functionality
- Usability and 508 compliance

- Information displayed

- Security

- Technical designs and considerations

5. Prototype Design Phase

This section discusses how we followed following plays:

- *Play 2: Address the whole experience, from start to finish*

- *Play 3: Make it simple and intuitive*

Based on the requirements gathered from our sample users, Intellect developed high level user stories that describe end-to-end prototype behavior. We engaged an experienced team member to serve in the capacity of Usability expert and front-end developer. We developed mock-ups, created user flows, and addressed 508 compliant elements in our design. We researched different websites that are top-ranked in terms of usability such as http://www.rxlist.com/script/main/hp.asp; http://www.drugs.com/; and https://www.myprime.com/ to understand latest design elements. Additionally, we also reviewed concepts used by 18F on their consulting projects and developed the user experience.

Our prototype includes other elements such as gather user feedback using Google forms, and help for users to easily perform search.

6. Technical Design Phase

This section discusses how we followed following plays:

- *Play 5: Structure budgets and contracts to support delivery*

- *Play 8: Choose a modern technology stack*

- *Play 9: Deploy in a flexible hosting environment*

- *Play 13: Default to open*

To align with US Digital Services Playbook, industry standard tools, industry leading practices, and 18F's approach to software development, Intellect chose all the sub-components that make up our prototype to meet following criteria:

- Use latest and reliable tools used in the industry

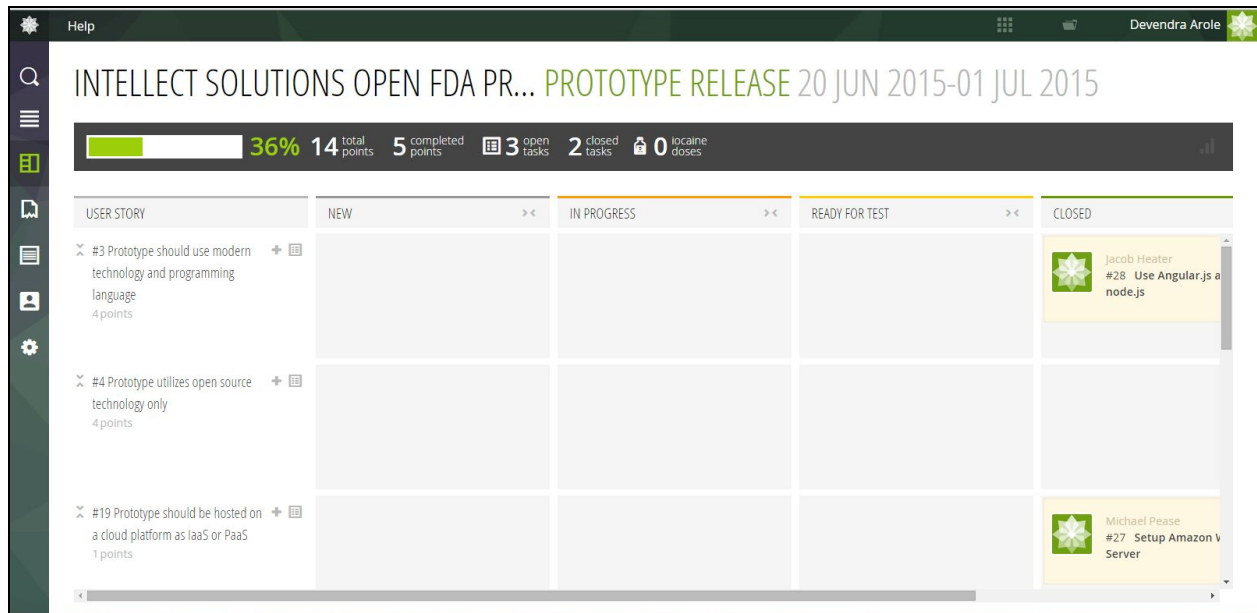- Increase reusability

- Open source and low startup cost

- Fast turn-around time

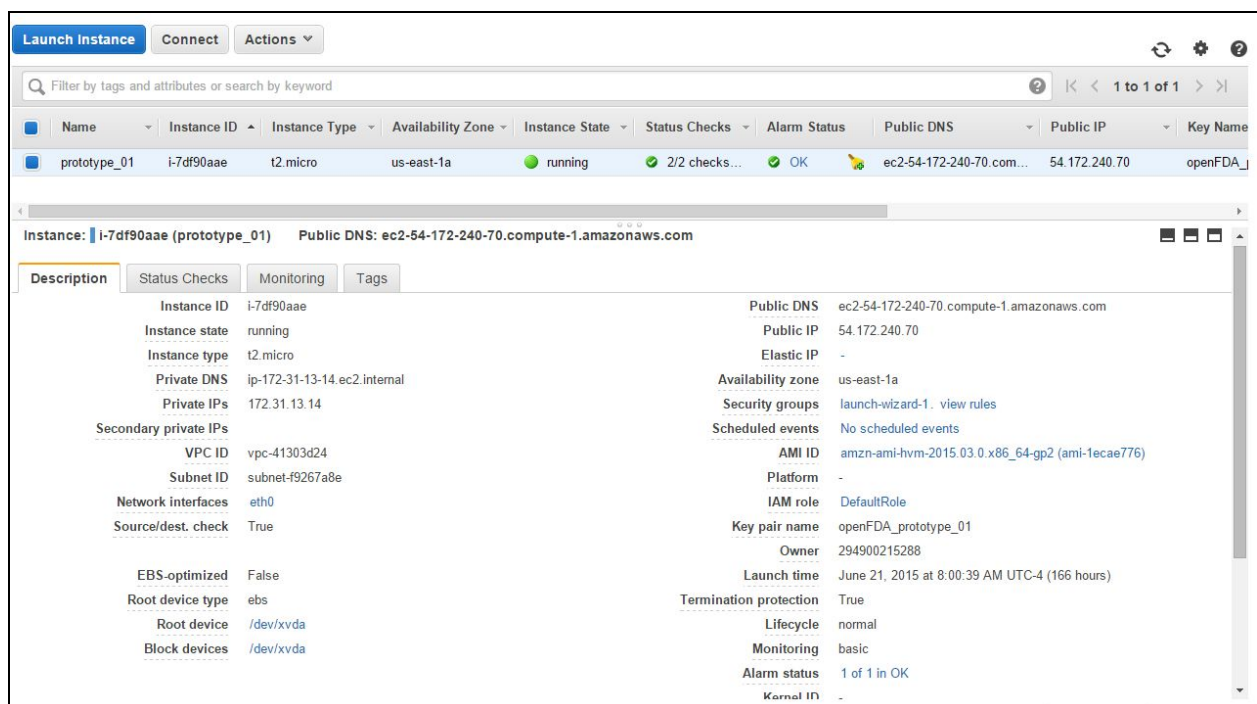Team Intellect used following open-source and modern technologies:

- SaaS hosting provider selected to host the prototype is Amazon Web Services

- Source control repository is Github

- Project management tool is open source and free Taiga.Io

- CentOS virtual server hosted on Amazon Web Services

- Kali Linux platform used for vulnerability testing

- Open VAS for vulnerability testing

- Front-end development framework is angular.js

- Back-end development framework is node.js

- JSON API specifications

GitHub Repository

Tiaga.io Collaboration Site

Amazon Web Service Prototype Instance Configuration Summary

Login Screen to AWS Instance with AMI Release Information Reference

7. Prototype Development Phase

This section discusses how we followed following plays:

- *Play 4: Build the service using agile and iterative practices*
- *Play 6: Assign one leader and hold that person accountable*
- *Play 7: Bring in experienced teams*
- *Play 10: Automate testing and deployments*

Team Intellect is adept at Agile development methodology, having done this for other government clients. After reviewing the requirements, gathering user stories by interacting with sample users via interviews and meetings, Team Intellect quickly organized itself to be lean and identified core team members required to develop the prototype.
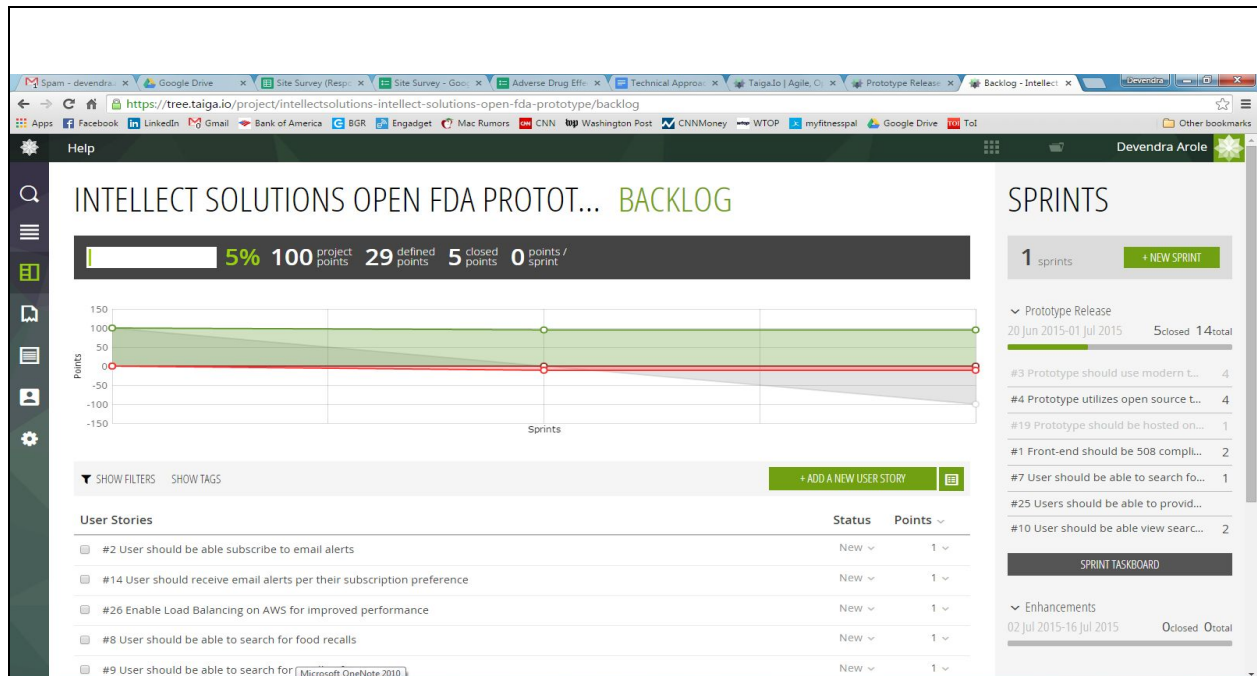
Sprints
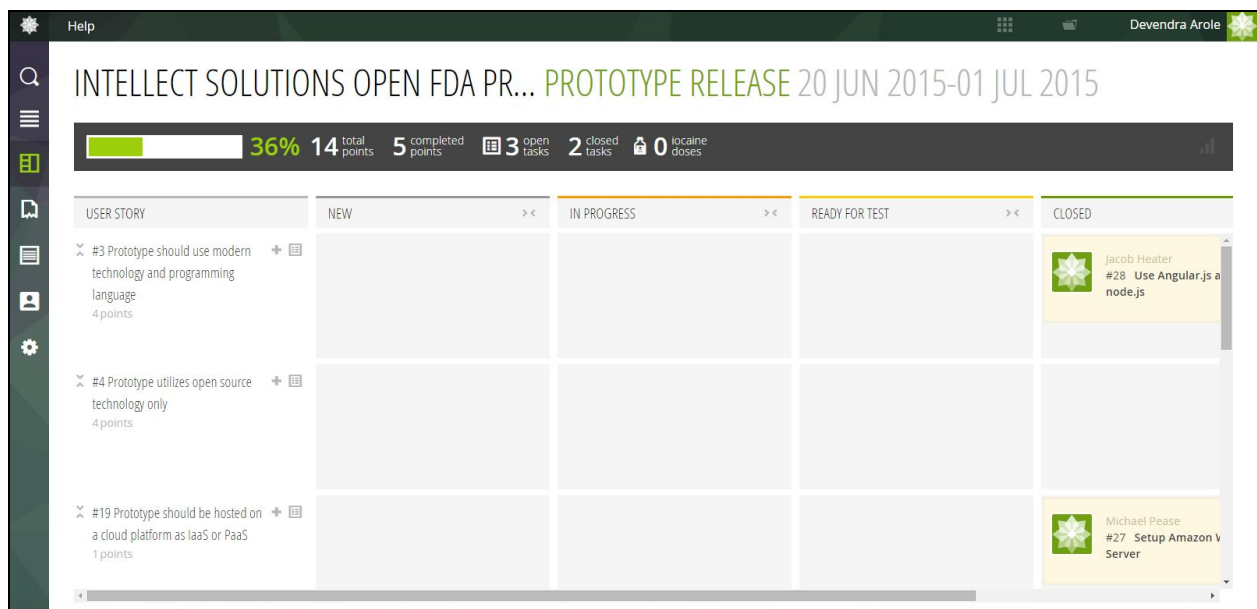
User Stories developed in Taiga.Io

The work was based around sprints or releases. Initial release focused on developing an alpha prototype in the short time frame as requested by the government. We also identified subsequent releases to address feedback from unit testing and end user testing. We also

identified some nice to have features that included functionality enhancements and UI improvements.

Team Intellect identified a diverse team with varied experience to develop the prototype. Team members that supported the development of prototype represented following labor categories under the **development pool**:

- Technical Architect
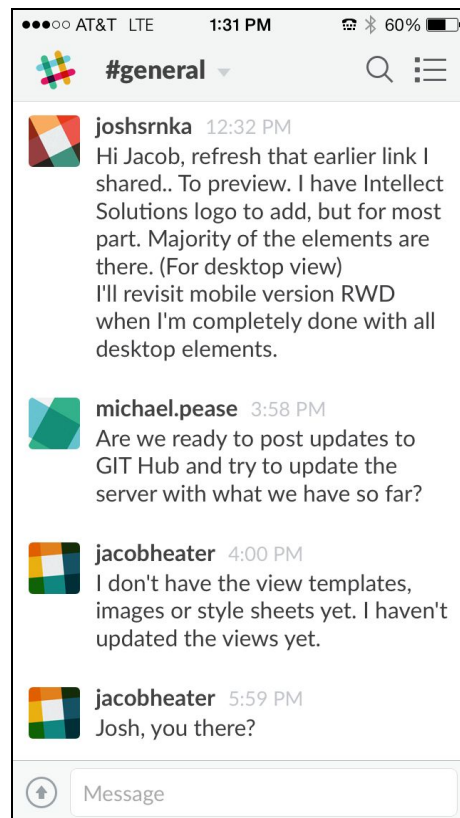- Front End Developer
- Back End Developer
- DevOps Engineer

Technical Architect played the role of the leader who was responsible for setting the tone of the engagement, execution and overall quality. This team came together, shared their experience, reviewed the RFQ requirements and quickly developed a prototype.

Design Sessions

Design Sessions

Team Collaboration using Slack tool

Testing

We are performing following type of testing as part of our methodology:

1. Functionality - Utilize JSFilddle.Net to create test AJAX scripts for submitting and validating Node.JS functionality to query and return data from OpenFDA. Additionally showed the working prototype to additional Intellect team members.
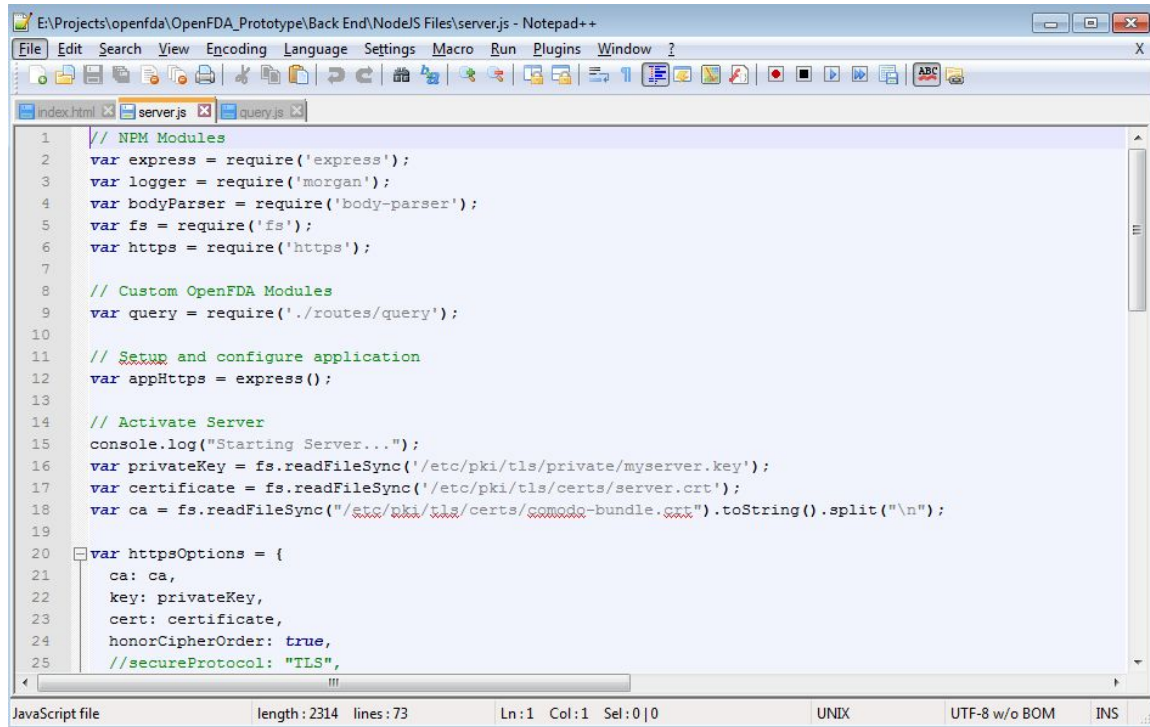

Demo to the internal team

Feedback from internal team members

2. Performance - Review logs to monitor query and page response times and adjust code and configurations based on log analysis results. (Further described below)

3. Security (through vulnerability management and penetration testing) - Utilized Kali Linux and Open VAS platform tools to test exposed services for vulnerabilities; SSL Labs and Comodo for validating SSL configurations.

Kali Linux Penetration Testing tool

4. Development - Used text editors to edit code files; preliminary functional testing was performed on local web services running on development workstations; resulting updates were submitted to the Github Repository.

Using Notepad++ to Edit Files

Configuration Management and Source Code Deployment

- Taiga.Io integrates with Github to check-in, check-out source code files and collaboration.

- Automated scripts are manually executed to update the server with the latest versions of the front and back end files.
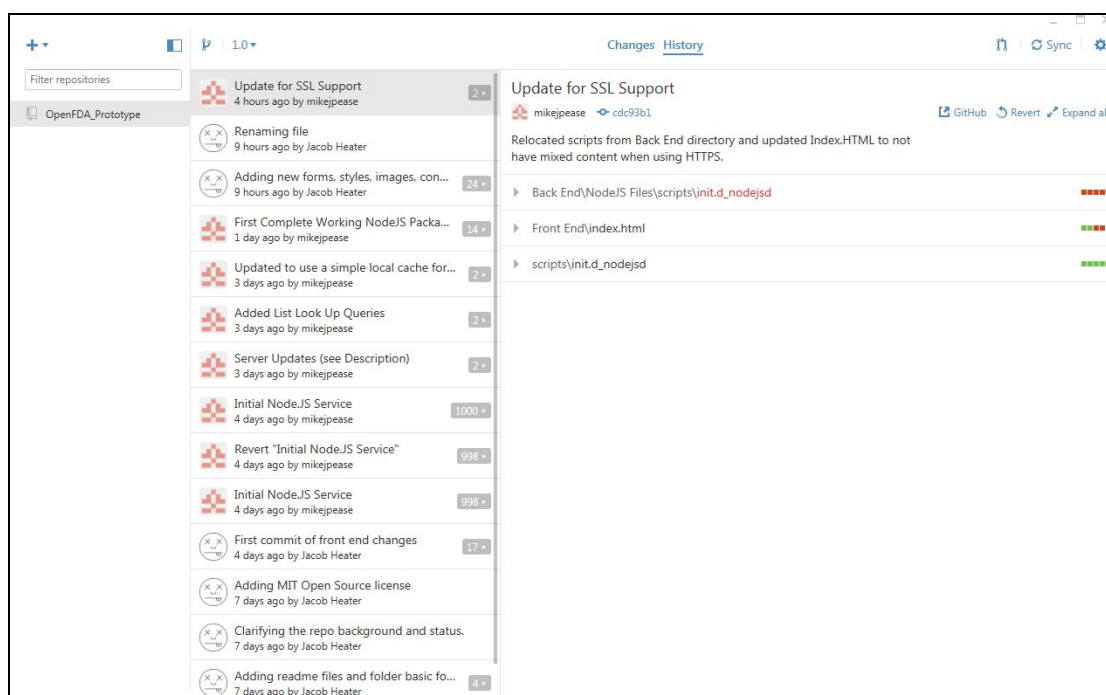
Synchronizing Changes to GitHub Repository

8. System Security and Ongoing Maintenance

This section discusses how we followed following plays:

- *Play 11: Manage security and privacy through reusable processes*
- *Play 12: Use data to drive decisions*

Security and Privacy

Security and privacy has played an important role in our approach. Unlike traditional software development projects where security is an after thought or nice to have, Intellect designed its prototype by placing huge emphasis on security.

Our prototype is deployed on Amazon Web Services hosting provider which is FedRAMP certified - http://aws.amazon.com/compliance/fedramp-faqs/. All the connections made to our website use HTTPS TLS v1.0, v1.1, and v1.2 with Cipher modes that place preferences on TLSv1.2 and only use the TLSv1.0 and 1.1 for backward compatibility. Additionally, we use OpenVAS to perform weekly vulnerability scans and address the critical and high vulnerabilities exploitable from the internet within 15 days.

During the initial architecture evaluation and design security review, we noted that while the OpenFDA site allows HTTPS for secure data transmission, the service does only supports GET requests. As a result, a risk was identified that internet service providers (ISP's) and other man-in-the-middle situations could result in obtaining information regarding users of the service

12

by allowing them to capture the query parameters and identify drugs, devices, or other medical information that users are researching. Our architecture attempts to mitigate this vulnerability utilizing Node.JS to provide a server side query capability. Users will utilize encrypted connections to the ADERS server platform which will create and submit the query request to OpenFDA for the user and then return the results to the requestor. While this does not eliminate the vulnerability, it anonymizes the connection by preventing the attribution of the query parameters to the actual individual. To further protect the user, the web services are not configured to capture query parameters with requesting IP addresses to ensure the anonymity of the user query requests to OpenFDA.
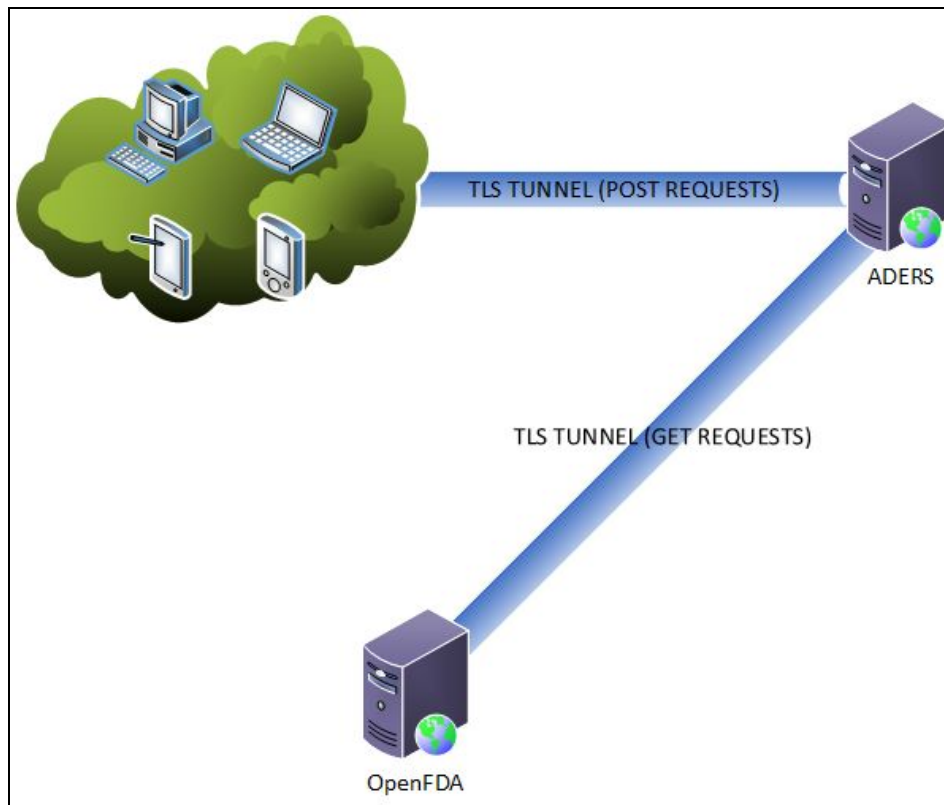

High-Level Architecture Diagram

As part of our commitment to maintaining privacy of user information, we only capture information necessary to meet functionality of the system. Apache captures IP addresses as part of normal website logging for statistical, performance and security monitoring.

Data Driven Decisions

AWS provides system level continuous monitoring. This helps us setup alerts, monitor system performance, and make necessary adjustments to the prototype behavior in a timely manner.
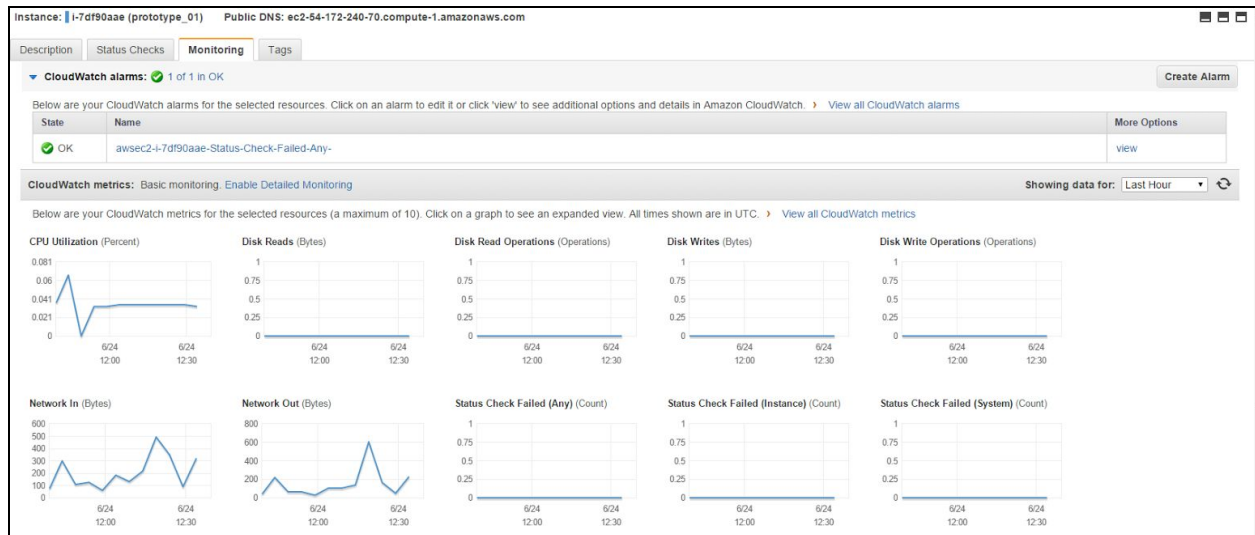
AWS Performance Monitoring Features

During development, performance monitoring for query response and page displays were analyzed as per normal functional testing and evaluation. During the test result review process and expected user interaction on the pages, the team identified that there are four queries required to display the drop down lists for users to select a drug. These are: Generic Drug List of Over the Counter Medication; Generic Drug List for Prescription Medication; Brand Name Drug List of Over the Counter Medication; and Brand Name Drug List for Prescription Medication.

The screen shot below shows that during initial testing, the OpenFDA query response times ranged from 65 ms to 186 ms.

Performance before optimization

Based on this information, the team concluded that these queries for multiple users and multiple times per user could be cached on the server to reduce the number of queries submitted to OpenFDA and improve the response time for page displays.

Once the caching capability was added to the service, the response times dropped to 2-9ms as shown in the monitoring screen below.

```
Listening on 8000
OpenFDA Query Response: 200
Caching BrandName OTC List
OpenFDA Query Response: 200
Caching Generic OTC List
OpenFDA Query Response: 200
Caching BrandName Prescription List
OpenFDA Query Response: 200
Caching Generic Prescription List
Using Cached BrandName OTC Data
GET /openfda/listBrandNameOTCDrugs 200 9.360 ms - -
Using Cached BrandName Prescription Data
GET /openfda/listBrandNamePresDrugs 304 3.533 ms - -
Using Cached Generic Prescription Data
GET /openfda/listGenericPresDrugs 304 2.696 ms - -
Using Cached Generic OTC Data
GET /openfda/listGenericOTCDrugs 304 9.900 ms - -
GET /openfda/clearcache 304 1.187 ms - -
Brand Name OTC List Request: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=
openfda.brand_name.exact
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.brand_name.exac
t
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listBrandNameOTCDrugs 304 26.032 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=PRESCRIPTION&count=openfda.brand_
name.exact
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listBrandNamePresDrugs 304 100.049 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.generic_name.ex
act
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listGenericPresDrugs 200 52.571 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.generic_name.ex
act
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listGenericOTCDrugs 304 47.172 ms - -
Using Cached Generic Prescription Data
GET /openfda/listGenericPresDrugs 304 2.757 ms - -
Using Cached Generic OTC Data
GET /openfda/listGenericOTCDrugs 304 0.819 ms - -
```
○ GitHub

Performance after optimization

Load Balancer

One of the future enhancement planned is to analyze the location of the traffic to determine load balancing requirements. AWS provides load balancing services at its servers across the Continental US and internal locations in Europe, Asia, etc. Once set up appropriately, the load balancer will automatically re-distribute connections across multiple locations and instances. We plan to leverage the IP addresses geocoding feature to identify locations where additional AWS EC2 instances and load balancing would provide the most significant performance increase based on actual user activity.