Automated Troubleshooting of Live Site Issues

Sriram Srinivasan

PayPal SRE

May 23, 2017

About Me

- MTS 2, Software Engineer @ PayPal
- Site Reliability Engineer

Agenda

- A bit about PayPal SRE
- Troubleshooting Challenges
- Manual Troubleshooting Process
- Requirements of Automated Troubleshooting Platform
- Evolution of the Architecture
- Architecture in Detail
- Major features of the Automated Troubleshooting Platform
- How to troubleshoot any type of Issues through Workflows?
- Future Plans

A bit about PayPal SRE

- Focus on the Key Aspects of Site Reliability:
  - Availability
  - Performance
  - Change Management
  - Monitoring and Alerting
  - Incident Management

- To troubleshoot and drive resolution of Live issues (from every domain) across the company.

Troubleshooting Challenges

- Manual:
  - Not having enough data to troubleshoot
  - Knowledge of the area/domain
  - Multiple signal generators
  - Inherent urgency in resolving
  - Takes time (due to human intervention)
  - Past troubleshooting knowledge not always leveraged
- Landscape:
  - Newer Products/Flows
  - People, Product & Bugs changing teams
  - Growing number of issues as we grow
  - Growing signal generators
  - Troubleshooting system generated Alerts not scalable
  - Low priority issues don’t get enough attention
  - Expiry of the logs

Manual Troubleshooting Process

- Issue Comprehension
- Categorize Issue (System vs Application)
- Look for Samples
- Tag Samples with the corresponding logs (spanning multiple applications)
- Check further in:
  - Stack Trace (Logs from the point of entry)
  - Recent Pushes (pertaining to the application/service)
  - Deployment Logs
  - Databases
  - In-house Alerting & Monitoring tools
  - In-house Admin tool
  - Code base
  - Production box
  - Bug Tracker/Ticketing Systems
  - …

Requirements

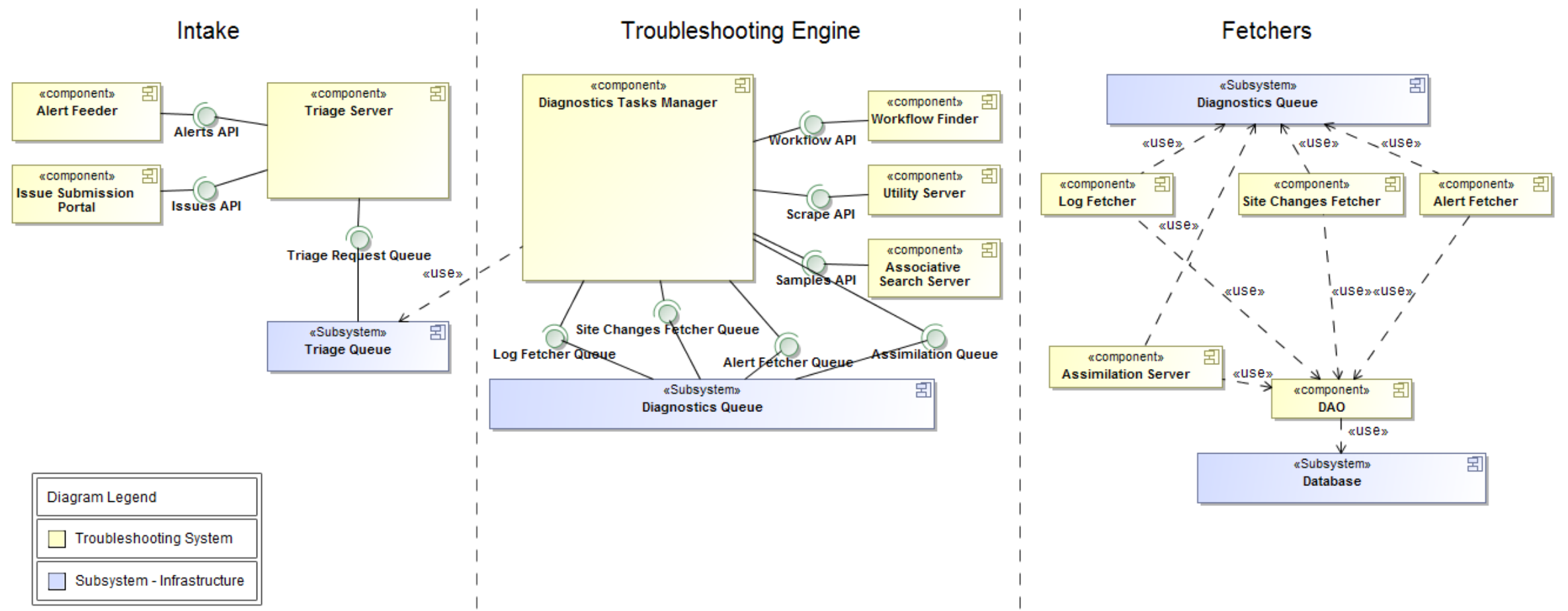
- Explicit Functional Requirements:
  - Automate the troubleshooting process

- Implicit Functional Requirements:
  - Provision to talk to disparate signal generators/data sources (like log servers, DB, …) synchronously/asynchronously
  - Adaptable to the growing signal generators/data sources
  - Ability to troubleshoot any type of issue/alert
  - Troubleshooting data augmentation/enrichment
  - Assimilation of the results from various data sources
  - Retain concerned Logs/troubleshooting info forever
  - Single place to view the auto-troubleshooting result
  - Build a Platform

Evolution of the Architecture

- Key Abstractions:
  - Identify the Type of Issue/Alert (Workflow)
  - Workflow has the say on how to troubleshoot (control strategy)
  - Augment the Troubleshooting Data
  - Invoke various Fetchers in the order prescribed (diverse specialized modules)
  - Gather results in a common place
  - Assimilate / Solution is incrementally constructed
- Architecture Patterns:
  - Multitier / n-tier architecture
  - Service-oriented architecture (SOA)
  - Presentation–abstraction–control / (MVC)
  - Blackboard system

Architecture



Workflows

- Workflow has details on how to enrich the troubleshooting data and what Fetchers would be required (including the order of invocation).
- Workflows are described in JSON and is nothing but a union of various Sections (or Directives).

```
"directives" : [
    {
        "_id" : "CF",
        "triageDataList" : [
            "POOLS"
        ],
        "pools" : [
            "servername1"
        ],
        "priority" : 0
    },
    {
        "_id" : "ASE",
        "triageDataList" : [
            "SELLER_TRAN_ID"
        ],
        "priority" : -1
    },
],
```

Major features of the Automated Troubleshooting Platform

- Pluggable:
  - Fetchers are Pluggable. We can add as many Fetchers (for Data Sources) as we want.
  - Language for the development of the Fetcher is not fixed.
- Expandable:
  - Add as many Workflows (Products/Flows) as possible. Workflow says what Fetchers to be invoked and in what order.
  - Issues and various types of Alerts can be triaged.
- Scalable by Design:
  - Asynchronous invocation of Fetchers.
  - Underlying technologies will also help.

Benefits

- Fast Triaging of all Issues & Alerts:
  - All issues and alerts are auto-triaged in minutes.
  - Reduces MTTT (Mean Time to Triage) and thus reduced MTTR (Mean Time to Resolve)
- Less Cost to Company:
  - Reduces the Sustaining Budget of teams. Teams can expend their effort on building other cool features.
- Customer Satisfaction:
  - Better Customer Satisfaction as logs are available forever and we don't need to go back to our customers.
- Better Insights:
  - As a single platform, it has gotten all the issues and their resolution. Thus this data platform can provide various insights.
  - Past triaging knowledge is leveraged for future troubleshooting.

Future Plans

- Platform Usage:
  - Continuously evolve the platform by adding more Fetchers
- Disposition:
  - Smart Issue Classification & Intelligent Issue Routing
- Data Platform:
  - Cataloguing the Issue with additional information (Resolution details, additional Notes)
  - Insights generation sliced by products, flows, root cause
- Proactive Measures:
  - Where more issues are coming and invest there by leveraging the data

Questions

?