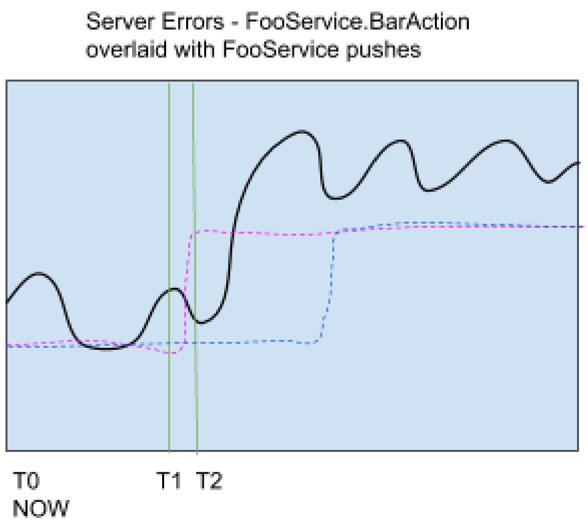
User Journey Tool - Change Over Time Design

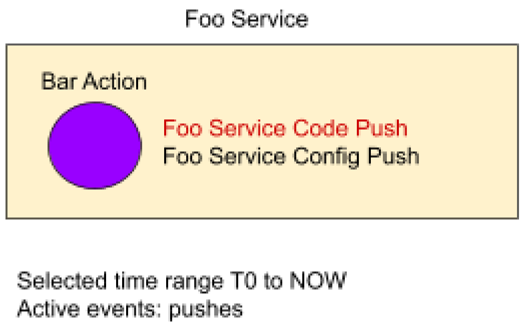
Context

The user journey tool currently displays the current status of services in a system based on their SLI values and the status of their dependencies. However, it may be desirable to view the historical status of services in order to identify the source of current errors or outages.

Initial Idea



This is the graph we might see in monitoring. The black line represents error rate for an action, Bar Action, in Foo Service. The pink dotted line represents a push of a new binary for that service (it isn't instantaneous because different servers might complete the rollout in different amounts of time). The green vertical lines are timestamps at which the push begins and ends. The Blue dotted line represents a config push for the service; I didn't bother drawing a time range around it but there is one.



This is an example of what we might see in the UI. We have selected a time range of T0 to NOW. The Bar Action is highlighted in purple, to indicate that the failure rate of that action got worse over this time period. The Code Push tag is correlated with that drop, because the average rate for a period of time before the event is better than the average rate for a period of time after the event.

The example above illustrates the connection between an example code push and config push, the existing graph in monitoring, and the desired at-a-glance information from the UJT. Per the example above, we would like the UJT to display the change over time of a specific SLI type:

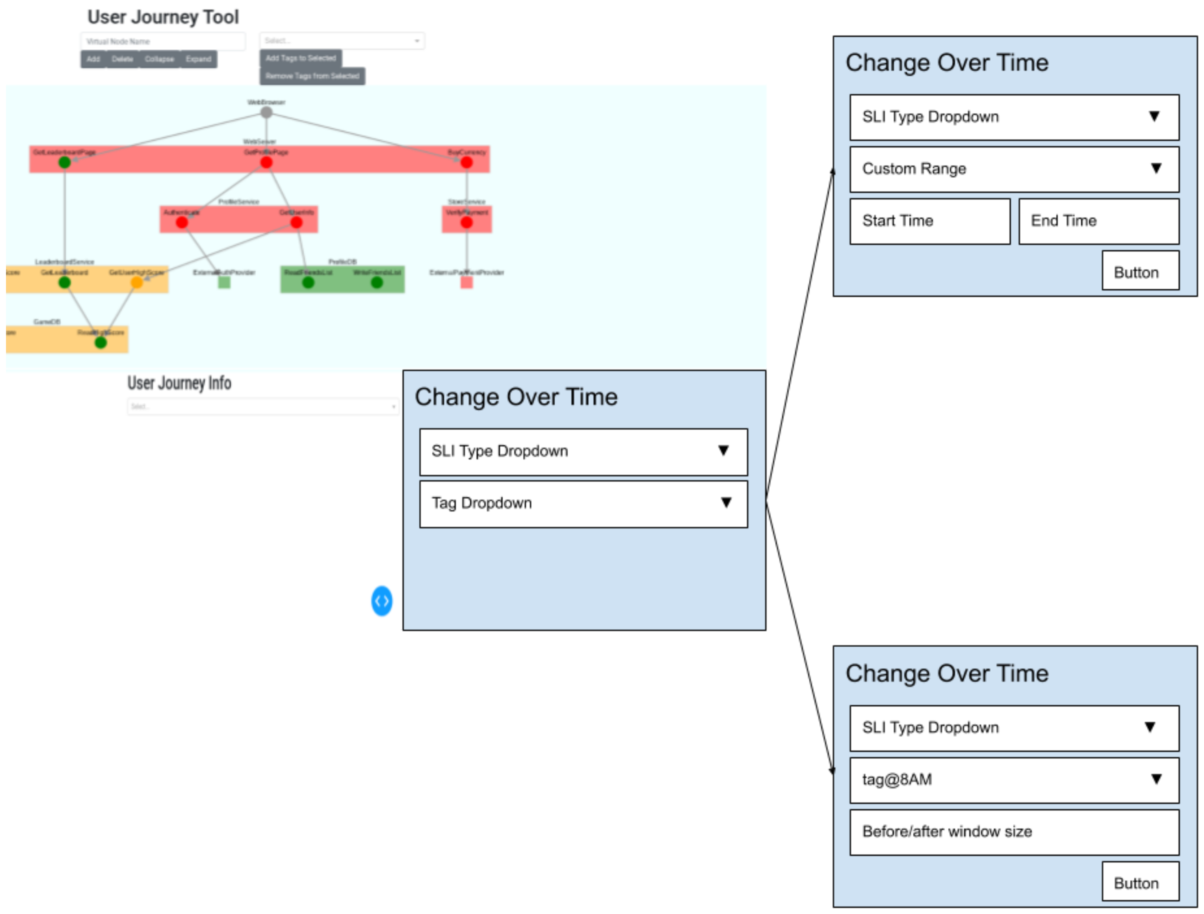
- over a user-selected (arbitrary) time range
- over a time range around a significant event

A difficulty is posed by the fact that Dash doesn't support JavaScript functions as arguments. Ideally, we would specify a function in the Cytoscape stylesheet to dynamically generate labels. As a workaround, I believe it's possible to generate the labels in a callback, save the labels to the element's data dictionary, and access the data dictionary through the data mapper.

Unfortunately, displaying multiple labels with custom styling for each (or a single label with custom styling with each line) doesn't seem to be possible without hacking or additional extensions that aren't supported in Dash-Cytoscape. Of course, we could simply use a text indicator like ↑ to indicate that the SLI increased after the timestamp. However, I tend to lean away from this approach as it could become unclear as the number of labels increases.

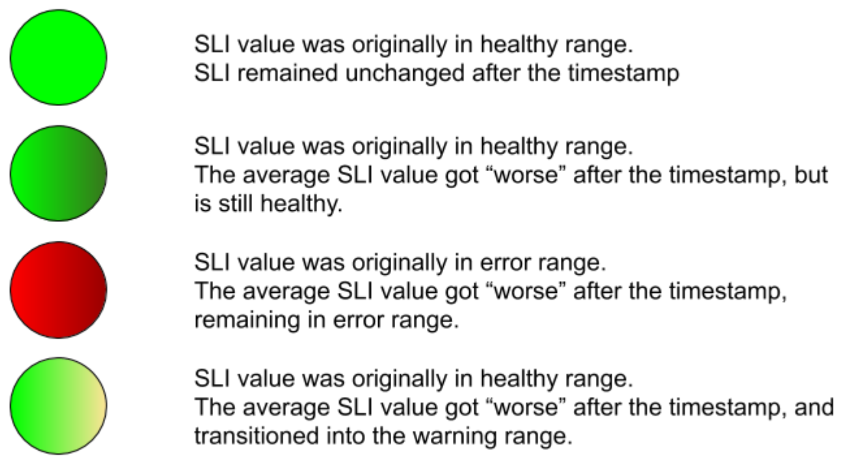
Solution

I plan to add a Display Change Over Time panel.



From this panel, users can select the SLI type they want to examine via the topmost dropdown menu.
From the tag dropdown, users can select either custom range, or a tag with an associated timestamp.
If the user selects "custom range", they are presented with start time and end time input fields to specify the time range that they would like to examine.
If the user selects a tag, they are presented with an input field to specify the window size (the time to examine before and after the offset).
The "over time status" computed from the specified time range will be displayed both visually in the graph, and in text below the dropdowns and inputs. As a crude initial approximation, we can

compare the average SLI value over (start time, midpoint or timestamp) with the average value over (midpoint or timestamp, end time).
The following diagram illustrates some examples of the visual indicators for different types of "over time statuses".


SLI value was originally in healthy range.
SLI remained unchanged after the timestamp
SLI value was originally in healthy range.
The average SLI value got "worse" after the timestamp, but is still healthy.
SLI value was originally in error range.
The average SLI value got "worse" after the timestamp, remaining in error range.
SLI value was originally in healthy range.
The average SLI value got "worse" after the timestamp, and transitioned into the warning range.

We also note that this panel takes precedence over all other styling in the graph. Nodes without the relevant SLI type or tag will be shown in grey. To disable this feature, we require users to clear the tag dropdown.

Data Structure Changes

Currently, SLIs have a warning range and an error range. If the SLI's value is within the warning range, the SLI is healthy (admittedly, this convention is a little confusing -- maybe this should be the named healthy range instead). If the SLI's value is outside the warning range but within the error range, the SLI is in a warning state. Finally, if the SLI's value is outside the error range, the SLI status is error.

This design allows us to account for different types of SLIs that target higher or lower values.

1. For example, if the SLI measures uptime percentage, a possible warning range could be (99, 100), and a possible error range could be (98, 100).
2. Alternatively, for a SLI measuring request latency, a possible warning range could be (0ms, 10ms), and a possible error range could be (0ms, 100ms).

In these cases, it's clear if an increase in the SLI value indicates that the service is performing better or worse.

However, for SLIs that should remain within a certain range, the SLI value moving towards the center of the range should indicate that a service is performing better.

To determine the meaning of a change in SLI, I see two possible solutions:

- Introduce a "target value" field in the SLI

    - If the SLI moves towards the target value, then it indicates that the service is performing better
    - This is inelegant for representing unbounded SLIs (e.g. target throughput could be infinite)[1]
- Use the SLI bounds to infer the target value
    - If the error range upper bound = the warning range upper bound, we can infer this is the target value. This corresponds to case 1 above.
    - We can apply the same logic for SLIs having the same lower bound for the error and warning ranges. This corresponds to case 2 above.
    - For SLIs that don't fall into the cases above, we can infer the target value to be the average of the warning bounds.
    - This could be somewhat unclear (?)
    - We also face the same problem of representing unbounded SLIs here.

I'm leaning towards introducing a target value to make the semantics more explicit.

SLI Querying

To reduce complexity on the UJT side, we decided to have the UJT query the reporting server with a time range, and have the reporting server return data at an arbitrary granularity/interval. This eliminates the need to store historical data on the Dash server. However, this does come at the cost of responsiveness, as the UI needs to wait for a network call to complete before being able to render the data.

Alternatively, we could save historical SLI data each time the UJT queries the server to refresh the SLIs. This involves some additional complexity on the Dash server, but could be done. We could also back-fill some historical data on startup. This approach would reduce the latency in querying intervals within (histroical backfill start time, current time). However, for intervals outside of this range, the UJT would still need to make a network call to the reporting server.

[1] A possibility is to monitor some function of the SLI of interest. For example, instead of measuring throughput, we could measure 1/throughput and have the target value be 0.