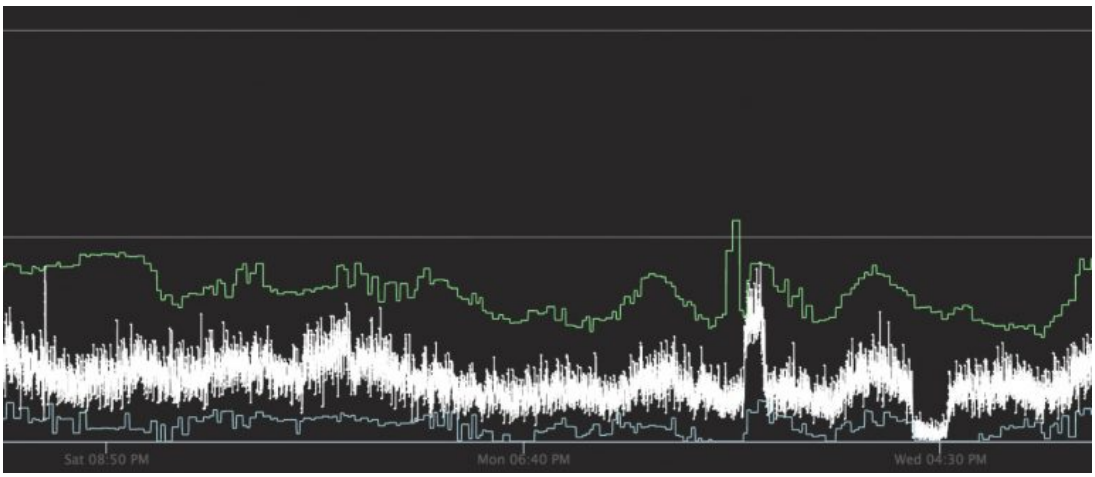
Implementing Model-Agnosticism in Uber's Real-Time Anomaly Detection Platform

February 14, 2018



By Jonathan Jin

Real-time alerting and monitoring systems contribute to our goal of achieving 24/7 reliability. To further this mission, Uber Engineering built an anomaly detection platform to find and flag deviations in system metrics and notify the on-call engineers responsible for addressing them.

Our platform was built to support a single forecasting model, but our growth necessitated new ones, each with their own distinct timeframes. Rather than starting from scratch for each new design, we opted to make our existing platform model-agnostic. Moreover, we were able to successfully integrate this extension into production with zero downtime and full backward compatibility.

Here, we discuss how we rolled out model-agnosticism for our existing anomaly detection platform, leading to more robust and extensible real-time alerting systems and an overall smoother on-call experience for engineers.

Anomaly detection at Uber

Uber Engineering's Observability team built the first iteration of our anomaly detection platform in late 2015. Since then, the platform has grown to encompass multiple discrete parts, including both forecasting and storage layers. The forecasting layer (F3) generates periodic forecasts, in other words, time-bounded sets of floating-point values corresponding to varying degrees of deviance from the predicted norm. The platform then transfers forecasts to our storage layer (Forecaststore) where users can retrieve both historical and near-future forecasts for more accurate predictions.

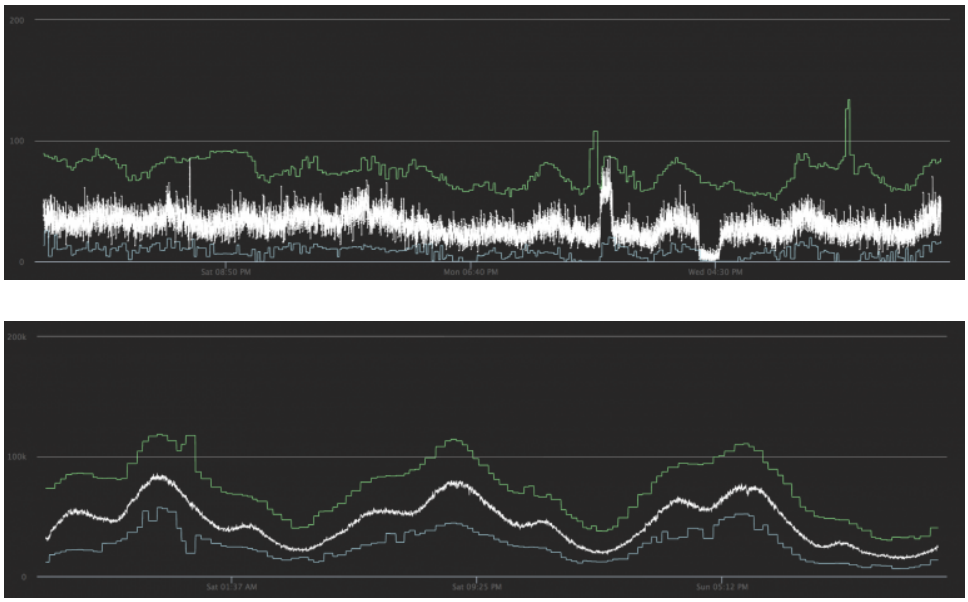


Figure 1: Powered by our F3 forecasting layer, Uber's anomaly detection platform tracks dynamic thresholds for seasonal metrics.

Our original anomaly detection algorithm, TimeTravel, generated metrics in 30-minute intervals, and was implemented as a self-contained forecasting model within our existing forecasting service. Originally deployed in Argos, TimeTravel assumes seasonality when forecasting for its underlying time series. Consequently, it is an insufficient method for detecting anomalies in more static metrics, such as those monitoring CPU utilization or the amount of open threads in a given service.

In early 2017, Uber's Observability Applications and Intelligent Decision Systems teams decided to address this limitation by expanding the platform's forecasting mechanism to encompass a broader suite of distinct models, thereby expanding the range of valid uses cases for our anomaly detection platform.

One of these new models was AutoStatic, a design that accommodates 24-hour forecasts; by paving the way for long-term metrics, AutoStatic allows users to rely on the model itself to determine optimal "static" threshold values.

However, as we began integrating AutoStatic into F3, we quickly discovered several fundamental shortcomings in the underlying architecture. First, F3's forecasting pipeline was unable to account for variations in model mechanisms. For example, a model that produced forecasts valid for the next 30 minutes was unable to co-exist with a peer model whose forecasts remained valid for the next five hours—F3's pipeline simply had no native way to account for these variations.

Additionally, the data model underlying Forecaststore stored all threshold values in a single top-level mapping that was keyed only by a threshold identifier. To store values produced by different models corresponding to the same threshold, we had to create and use specialized threshold keys that contain the model name itself, e.g. ModelNameExtremelyHigh or SomeOtherModelNameExtremelyLow, to prevent confusion and clobbering other models' values. An early version of the AutoStatic model was implemented exactly this way, and the result was expectedly cumbersome for both F3 developers and consumers, as demonstrated in Figure 2 below:

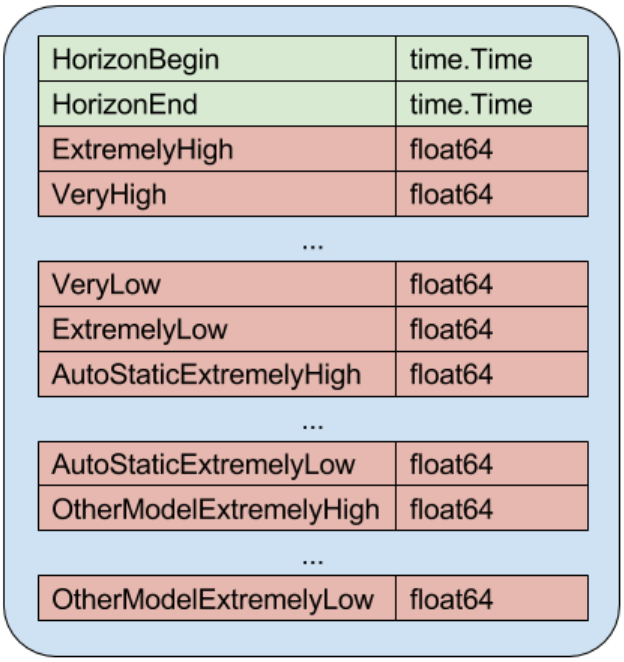


Figure 2: Under the pre-existing data model, integrating a new model required growing the threshold keyspace linearly. Consumers' threshold selection becomes implicitly coupled to model selection, and only one model's expiration time can be recorded at a time.

These shortcomings made further expansion of the anomaly detection platform's model fleet difficult. We knew that further development would only exacerbate the problem, so we took a fundamentally different approach. Decoupling F3 from TimeTravel made the service model-agnostic and extensible.

To accomplish this restructuring, we first defined the updated platform's requirements, outlined below. Primarily, we needed to support distinct:

- **Update schedules**. TimeTravel, for instance, updates its forecasts every half-hour; another model might do so every 24 hours. Our final design would need to provide support for models that update on differing schedules.
- **Forecast threshold values.** All models need to output thresholds corresponding to the same severity levels, e.g., ExtremelyHigh or SlightlyLow, without clobbering values output by another model.

With these specifications in mind, we arrived at a design that repurposed our existing data models as a "collection," scoped by individual models. This new paradigm afforded us greater extensibility, thereby enabling forecasting models to output threshold value sets with their own validity durations. This provides F3, as well as any consumers of its produced forecasts, with a native ability to disambiguate between models and their corresponding outputs.

Explicitly scoping threshold values by model also lets consumers decouple model-selection logic from threshold-selection logic; threshold semantics, after all, remain the same across models. Finally, this data model—outlined below—lets F3 update model forecasts as they expire, while leaving the others untouched.

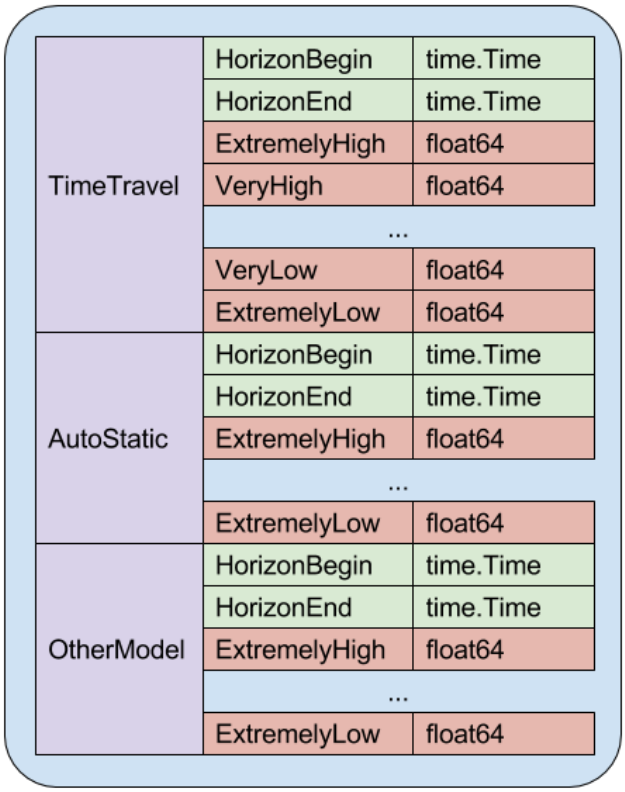


Figure 3: Under the new data model, integration of a new model only requires adding a new "model identifier" symbol. The threshold keyspace remains constant, and the data model itself directly reflects the separation of distinct concepts.

Implementation and rollout

Before we could execute, however, we had to ensure that our new design was interoperable with our legacy platform by providing support for top-level fields and backwards compatibility, a functionality that would let platform developers continue to iterate, independent of any downstream dependencies.

Our new design made accounting for these needs straightforward. Since each of the deprecated top-level fields are implicitly associated with one model—TimeTravel—they each have one corresponding accessor with respect to the new, model-specific fields. As a result, even if the forecasting layer is done populating legacy fields, the storage layer can impute the values that belong in those fields using their accessors and inject them into the corresponding object payload at query time. This strategy allows forecast consumers to onboard new accessor fields according to their own schedules, while also enabling the F3 team to continue growing the platform on the backend.

Not only does this "impute-and-inject" pattern ensure backward compatibility, but it also lets us refit older forecast results (i.e., those that were persisted to storage prior to the introduction of these new fields) for the new data model. In other words, we use the same strategy to implement not just backward compatibility, but also forward compatibility. The end result: a platform extension that seamlessly enables increased flexibility for producers and consumers alike.

Raw stored payload	User-facing read value

<pre>{
  "Thresholds": null,
  "ModelThresholds": {
  "TimeTravel": {
  "Thresholds": {
  "ExtremelyHigh": 9.0,
  "ExtremelyLow": -9.0
  }
  }
  }
}</pre>
	<pre>{
	  "Thresholds": {
	  "ExtremelyHigh": 9.0,
	  "ExtremelyLow": -9.0
	  },
	  "ModelThresholds": {
	  "TimeTravel": {
	  "Thresholds": {
	  "ExtremelyHigh": 9.0,
	  "ExtremelyLow": -9.0
	  }
	  }
	  }
	}</pre>

<pre>{
  "Thresholds": {
  "ExtremelyHigh": 9.0,
  "ExtremelyLow": -9.0,
  "AutoStaticHigh": 100.0,
  "AutoStaticLow": -100.0
  },
  "ModelForecasts": null,
}</pre>
	<pre>{
	  "Thresholds": {
	  "ExtremelyHigh": 9.0,
	  "ExtremelyLow": -9.0,
	  "AutoStaticHigh": 100.0,
	  "AutoStaticLow": -100.0
	  },
	  "ModelForecasts": {
	  "TimeTravel": {
	  "Thresholds": {
	  "ExtremelyHigh": 9.0,
	  "ExtremelyLow": -9.0,
	  }
	  },
	  "AutoStatic": {
	  "Thresholds": {
	  "ExtremelyHigh": 100.0,
	  "VeryHigh": 100.0,
	  "VeryLow": -100.0,
	  "ExtremelyLow": -100.0,
	  }
	  },
	  }
	}</pre>

Figure 4: An illustration of the "impute-and-inject" pattern used as part of the storage layer's backwards-compatibility logic. Also shown is one of the "workarounds" that initial new model development employed to circumvent the data model's limitations, as well as this pattern's ability to take that workaround into account as part of the pre-existing API.

Key takeaways

Our project illustrates the unique challenges that crop up at the intersection of mathematical modeling and platform engineering. Moreover, it suggests that general engineering best practices remain widely applicable irrespective of domain, whether applied to time series forecasting or service-oriented architecture development.

Specifically, from a platform perspective, our research has shown that:

- There is value to viewing your architecture not as a static, unshifting limitation, but as an organic entity that can adapt and evolve in the face of changing external demands.
- It can be worthwhile to extend seemingly unscalable systems instead of redesigning them from scratch, making it easier to support historical use patterns.

From a modeling perspective, this project draws attention to the challenges of taking an isolated model and integrating it into a pre-existing production system. In particular, assumptions must be broken down into low-level constructs, at which point they must either be reworked to integrate with existing architecture, or—as shown with our anomaly detection platform—extended to account for new requirements.

Next steps

Our anomaly detection platform's newfound extensibility has drastically reduced the barrier-to-entry for forecast model integrations, allowing the platform to scale to meet more complex use cases. Over time, this extensibility will provide more accurate, intelligent, and actionable real-time alerts for engineers across Uber.

While there is more work in the pipeline for our anomaly detection platform, the new extensions outlined above will pave the way to future optimizations across our real-time alerting and monitoring systems.

If you would like to empower engineers at Uber to monitor their production systems, the Observability Applications team would love to speak with you.