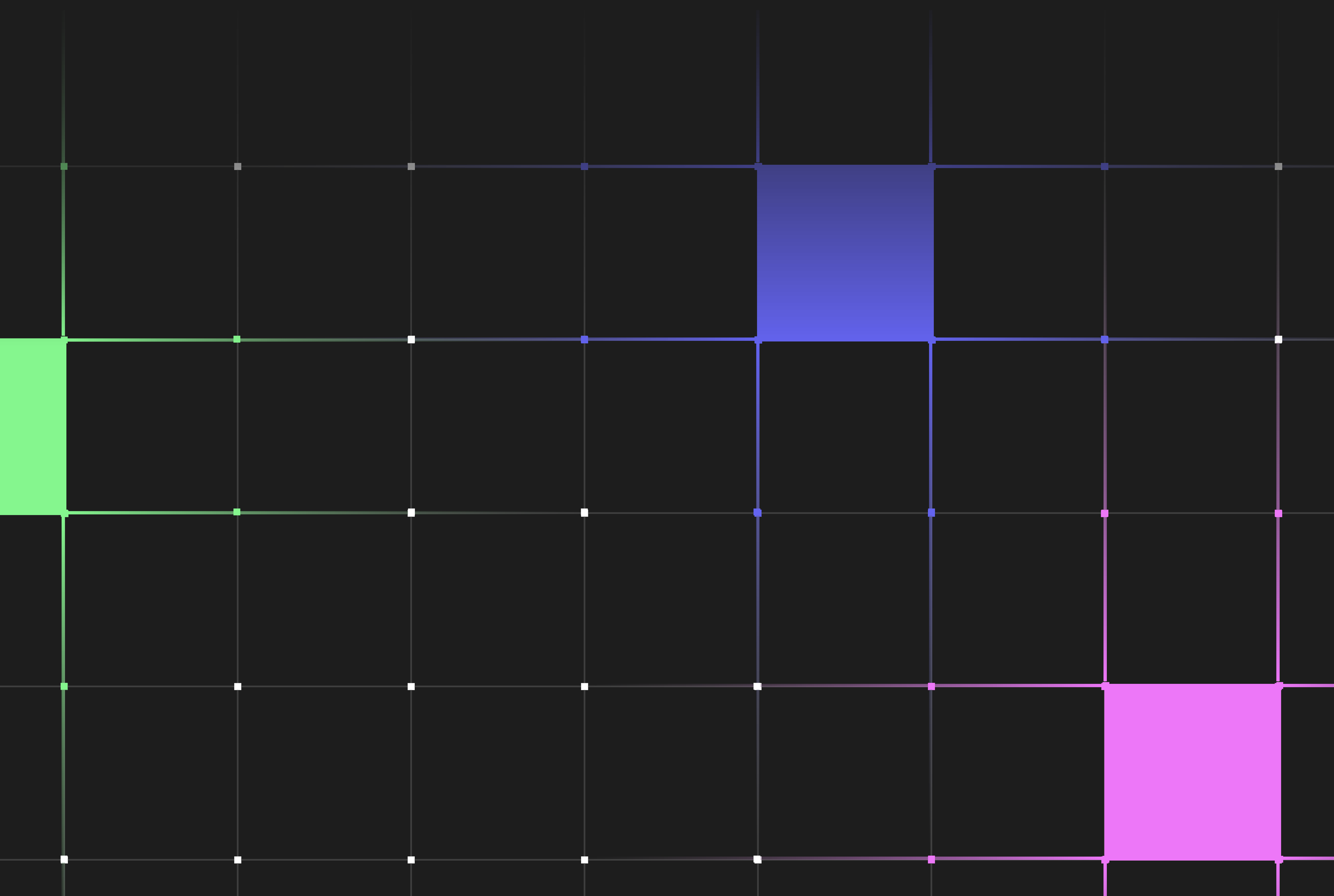
[DISTRIBUTIONAL]

Distributional's Platform



Intro

Distributional has updated and continues to update our core functionality.

See docs.dbnl.com for the latest on our data pipeline, analytics workflow, and enterprise platform.

Adaptive testing is critical to support production-grade AI applications at scale. This enables AI teams to leverage behavioral distributions to create a comprehensive definition of an app's desired behavior that can be refined over time. Thus enabling teams to quantifiably test and investigate when there are deviations from that desired behavior.

Distributional provides an enterprise platform for adaptive testing, which is designed to meet the scale and performance needs for AI teams to continuously test, understand, and refine application behavior. Using Distributional's platform, customers are able to shorten the development cycles for AI, productize higher value applications, and keep them in production. All while minimizing risk to the business with the confidence that these applications are behaving and will continue to behave as desired.

Distributional's platform was designed to easily integrate with existing tools and workflows, and to scale alongside AI teams and their projects. The architecture is intentionally streamlined to result in efficiencies and simplified management in the long run.

This guide was written to help AI teams quickly get up to speed on Distributional's architecture so they can hit the ground running. In the following pages, we'll share key terminology, take a deep dive into the technical architecture, and review how it can integrate into existing architecture.

Table Of Contents

Distributional Workflow & Terminology

Before diving into how the Distributional platform is designed, it's important to understand some key terms related to how Distributional implements adaptive testing for AI apps. These help to break down the types of computation in the system done to create a statistical representation of behavior for AI applications.

THE FIRST STEP WITH DISTRIBUTIONAL IS FOR A USER TO SEND DATA TO THE PLATFORM.

DATASET

A dataset includes examples of running the application (such as inputs and outputs for a question / answer app), and can be augmented with as few or as many metrics as available.

As long as the dataset fits in a pandas DataFrame, Distributional is able to start working with it.

METRIC

A metric is a measurable property of the app. These could include: response time, word count, token count, LLM-as-judge metrics, etc. Metrics can be both user-created (such as existing evaluation metrics) and generated by Distributional automatically.

WHAT MAKES A GOOD EXAMPLE OF APP USAGE?

An example is the data related to a single instance use of your app. If additional context is added, Distributional is able to compute a more complete definition of app behavior. Here's an example of some common columns and attributes for a question / answer application:

- **Question:** The question inputted into your app by the user.
- **Answer:** The answer outputted by your app.
- **Response time:** The time it took the app to produce the answer.
- **Context:** The context that was used to generate the answer (e.g. documents retrieved from a knowledge base).
- **Eval(s):** Performance metric(s) already being calculated.

ONCE THE DATA IS COLLECTED, DISTRIBUTIONAL PROCESSES THIS DATA TO GENERATE A RUN.

RUN

A run is the result of uploading a dataset to Distributional. It includes a set of results derived from the examples in the original dataset augmented with metrics and other data computed by Distributional, used to create a measurable definition of application behavior.

BY QUANTIFYING THE DEFINITION OF BEHAVIOR AS A RUN, IT CAN NOW BE TESTED FOR CHANGES TO THAT BEHAVIOR OVER TIME.

STATISTIC

A statistic is an aggregate quantity computed over metrics in one or more runs. Statistics can be used to measure changes in the distribution of a metric across runs, for example by computing a Kolmogorov-Smirnov test.

TEST

A test defines the acceptable range of values compared to a baseline run. These definitions can be automatically generated by Distributional based on known behavior, or defined by the user, creating a definition of desired behavior. A test can check for behavioral change between two runs, or check for behavioral constraints of a single run.

TEST SESSION

A test session compares two runs to test for behavioral similarity over time. A test session executes a set of predefined tests over a run comparing it to a known baseline run and compares the run statistics against a set of thresholds.

TEST RESULT

A test result is the outcome of executing a test as part of a test session. For each test, the test result notes whether the test assertion passed or failed when evaluating the run compared to the baseline run.

SIMILARITY INDEX

A Similarity Index (Sim Index) is a numerical value — between 0 and 100 — that quantifies how much an application or subsets of an application (at the column- and metric-level) has changed between runs.

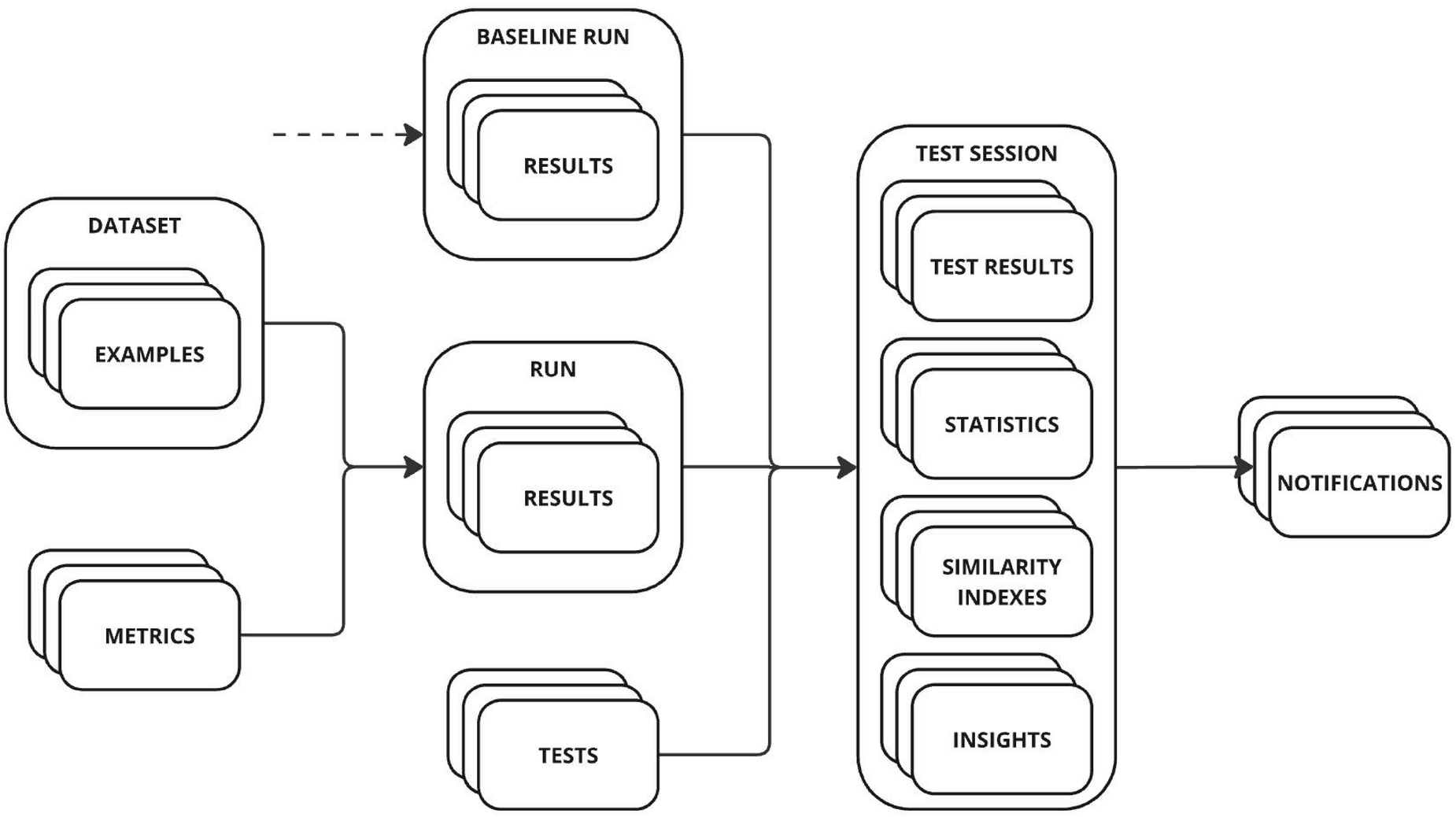
ONCE A CHANGE IS DETECTED, USERS CAN BE NOTIFIED, UNDERSTAND THE CHANGE, AND TAKE ACTION.

NOTIFICATION

A notification can be triggered on the outcome of a test session. Users can use their preferred notification channels (such as PagerDuty or Slack), to create custom notifications based on the properties of the test results in a test session.

INSIGHT

An insight is a piece of human readable evidence explaining the outcome of a test result. While Distributional calculates metrics and statistics to test against, it also maintains the lineage to the raw dataset. This allows users to explore evidence from test results to understand and pinpoint specific properties or values that caused the change in behavior. Users then are able to assess whether there is an underlying issue in the application that needs to be resolved, or whether the test needs to be recalibrated.



DISTRIBUTIONAL'S PLATFORM

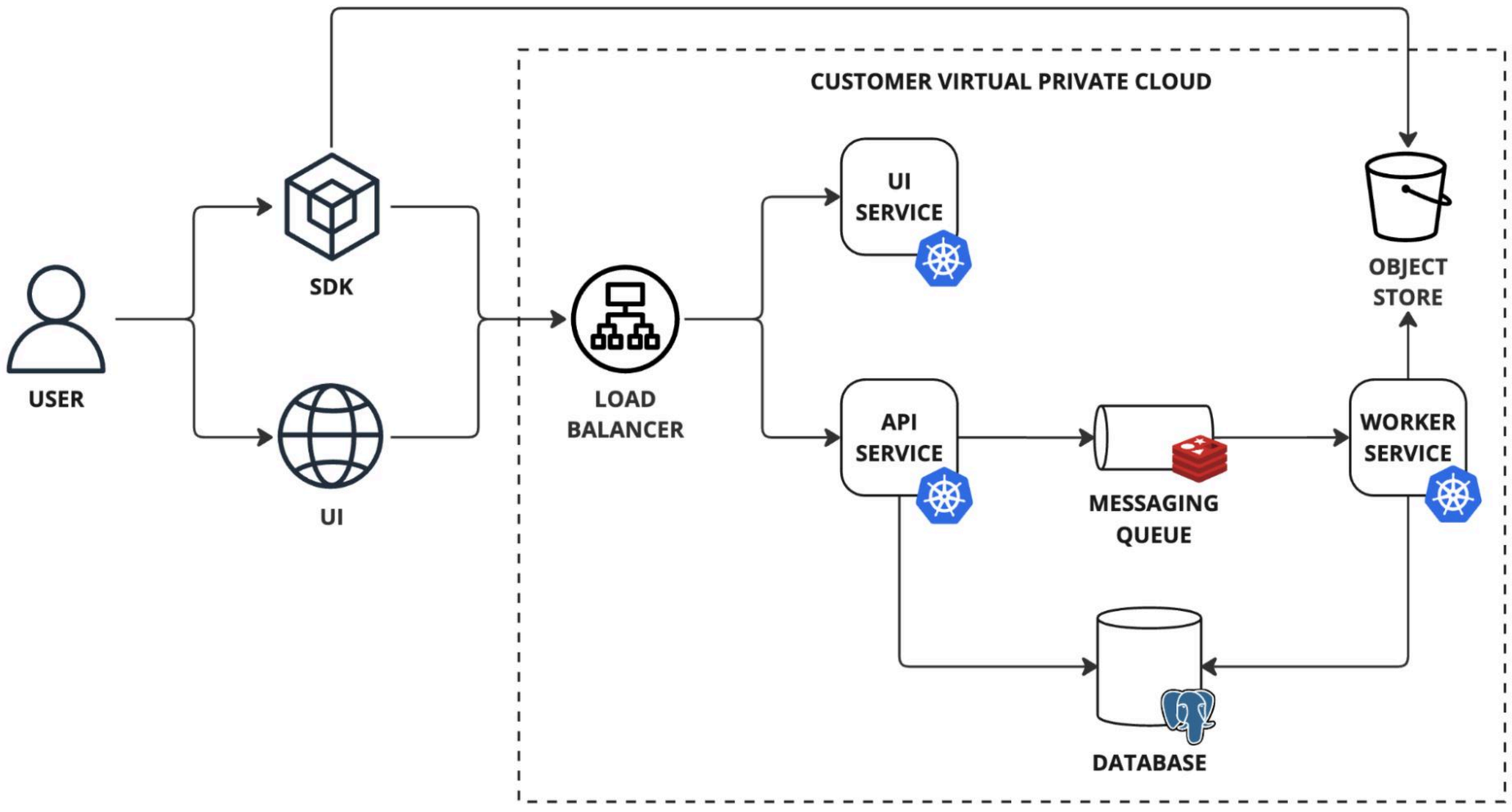
Distributional's Platform Architecture

For the platform to support this workflow and continuously test for behavior changes over time, it needed to both be able to handle large, context-rich datasets as well as perform fast, analytic-style processing of that data. Let's go into more detail about how the platform is architected to support this.

Importantly, Distributional's platform does not try to replace customers' existing data storage systems. It is purposefully designed to integrate with existing data infrastructure, where all the raw logs for the AI applications already are stored. Based on the customer-defined ingestion schedule, Distributional is then able to efficiently process this data in batch, with inherently no limits on scale. This enables the platform to run much more complex analytics against the data, while maintaining its rich context, to derive a comprehensive set of metrics and statistics about the applications, as well as calculate changes to these over time.

Additionally, unlike systems designed for pure logging and monitoring, users can update or change the data provided to Distributional. This gives them the ability to do things like adding in missing data points and retrying the processing job, expanding the type of data and context provided, or even recalculating metrics based on data from a past point in time.

Since access to this data is critical, customers deploy and manage the Distributional platform in their private cloud environment. This prevents Distributional from becoming yet another technology silo within a customer's overall architecture and allows the platform to live where the data already resides, preventing duplicate systems of record or concerns about moving data outside of the existing secure environment. To support this though, we purposefully architected it with as few dependencies as possible and leveraged industry-standard systems to make it as easy to deploy and manage as possible.



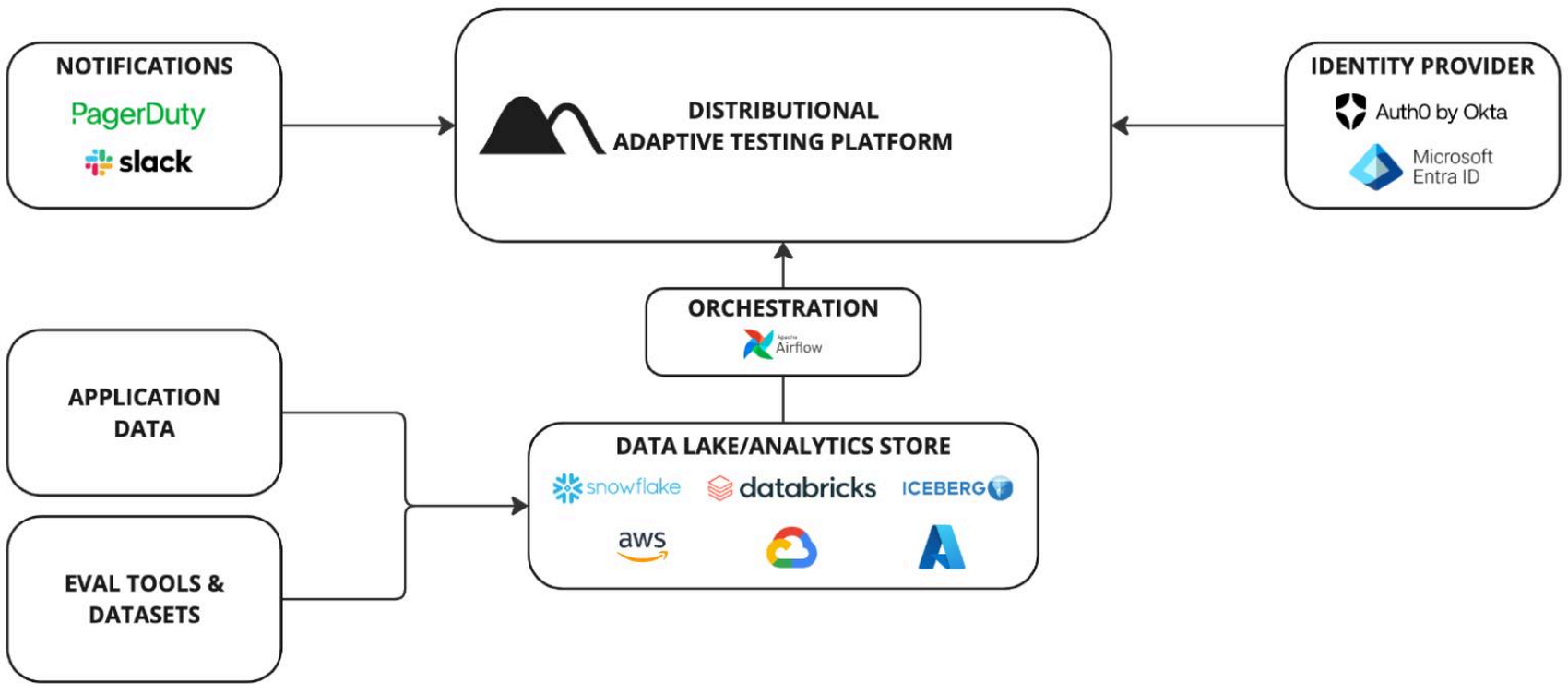
At a high level, the platform consists of an API service, a UI service, a worker service, a messaging queue, and a database. All three services run on Kubernetes with Redis and PostgreSQL being used for the messaging queue and database, respectively. The data is stored in an object store with support for AWS S3, Google Cloud Storage (GCS), and Azure Blob Storage. This results in only four external dependencies that all leverage industry-standards our customers are already familiar with.

The core of the platform is the API service, with users able to interact with it via the user interface or SDK for more programmatic access. When the API service is tasked with a workload such as computing metrics on raw data or evaluating different tests, the messaging queue relays the work to the worker service for compute processing and can scale out these resources or adjust the type of compute as needed depending on the size of job or concurrent requests. For the users, this design abstracts away the complexity of managing storage and compute resources, so they can focus on getting the answers they need from their application's data.

Integrating Within A Customer's Environment

As mentioned earlier, the Distributional Platform is designed to sit within a customer's existing data infrastructure, rather than be a separate silo to manage and sync. It can easily be deployed in their existing cloud environment and integrate with their existing storage system. This makes it easy to fit seamlessly within any AI platform. By leveraging industry-standard components, it also ensures that customers can take advantage of the managed cloud service versions of each for even easier management.

In addition to being fairly agnostic with regard to where the data lives, the platform is also agnostic on what the data looks like. Any existing logs, traces, or other evaluation metrics can be used as inputs – as long as it's structured and fits in a pandas DataFrame, that's enough for the platform. The more data provided, the more context the platform has to develop a comprehensive understanding of behavior for testing.



Customers can then use their preferred orchestration tool to define the ingest schedule for how often new data is sent to Distributional. For example, they could schedule daily updates through Airflow to ensure their models haven't drifted by feeding a fixed set of inputs into their AI app to generate a dataset of examples to be uploaded to Distributional as a run and tested for change.

For customers to get alerted when there are key changes to their AI app's behavior over time, they can also integrate Distributional with PagerDuty or Slack for real-time notifications.

For authentication, the platform uses OpenID Connect (OIDC), again ensuring seamless integration with any preferred identity providers. In addition to authentication, the platform also has a native permission model using role-based access controls for further security.

Overall, the SDK is designed for broad extensibility. It's built in Python, making it flexible for customers to integrate with other preferred tools or existing processes. Distributional is also continuing to expand the native integrations built into the platform to make this even more seamless for customers. This allows customers to seamlessly integrate Distributional with their AI platforms, while still allowing for portability and adaptability as these platforms continue to mature.

Ultimately resulting in a platform that is easy to deploy and manage, integrates within an existing environment and preferred tooling, and is built for enterprise scale and secure usage.

Conclusion

With Distributional's adaptive testing platform, customers no longer need to build AI applications in a vacuum, and are able to account for the uncertainty of production usage while constantly adapting applications incrementally over time. This leads to fewer production surprises and the ability to catch gradual shifts before users do. Plus, with a shared comprehensive view of desired behavior, the silos between development to production break down, resulting in faster and more predictable updates. Ultimately this unlocks the confidence needed to ship higher value AI products faster, while minimizing business risk.

For additional resources, please visit:

- Distributional Docs
- Distributional Blog

[DISTRIBUTIONAL]

About Distributional

Distributional is building the modern enterprise platform for adaptive testing to make AI safe, secure and reliable. As the power of AI applications grows, so does the risk of harm. By taking a proactive, adaptive testing approach with Distributional, AI teams can deploy AI applications with more confidence and catch issues before they cause significant damage in production.

Learn more at distributional.com

Follow us on:

- LinkedIn
- YouTube