opsclarity

Monitoring and Troubleshooting Data-First Applications

Table of Contents

Introduction

The last few years have witnessed an upheaval in how applications are built. Applications can now handle massive volumes of data to address the growing needs of digital businesses that are trying to gain competitive leverage by extracting real-time insights from data. Today, the industry is undergoing a monumental shift in the way it develops applications to what we call “Data-first” applications.

Data-first apps start with the data at core of their application architecture and focus on what is called a data pipeline instead of transactions. They typically leverage open source data processing frameworks like Apache Spark, Apache Kafka, Apache Storm and Elasticsearch. They use a multitude of new programming languages and frameworks to construct dynamic applications that continuously interact with and process huge amounts of data.

While these data-first applications have become a key component, the highly complex, interconnected mesh of different services designed to optimize performance of the application as a whole has created a completely new set of challenges when it comes to monitoring, troubleshooting and maintaining the health of the application. This architectural transformation demands purpose-built solutions that bring significantly more domain depth and visibility than the legacy technologies and generic approaches previously used.

Understanding the Evolution of “Data-first” Applications

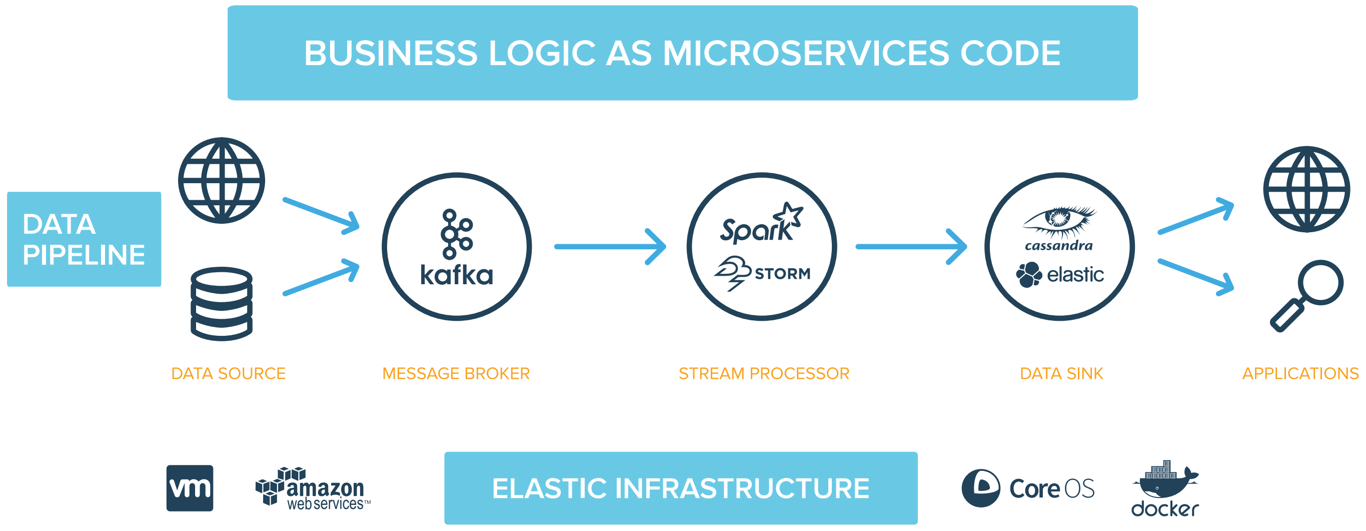
Over the past decade, rapid advances in information technology have radically shifted expectations for both consumers and business users. We now live in a world where almost every activity from shopping to finding new customers has made a digital transformation. The digital transformation of our personal lives and businesses has led to a creation of a constant and massive stream of data. This enormous data stream has in turn opened up new opportunities for businesses to extract intelligent insights from data in near real-time to create new user experiences quickly and engage with their customers more responsively than ever before. Simply put, data processing has become a key component of modern business.

The data pipelines are meant to extract insights from data rather than process a specific business transaction. Code and data frameworks are interleaved with each other in multiple layers. The focus is on processing data. Developers may manage and access this ever-changing, heterogeneous mass of data through a microservices and/or containerized architecture.

Application Architecture

Data-first application architectures are significantly different from traditional 3-tier or n-tier applications. Data sits at the core of data-first applications. The data pipeline, rather than the individual transaction, is the critical component that developers and architects of these data-first applications tend to focus on. A typical data pipeline ingests either data-at-rest from NoSQL data stores, or captured through streams coming from click-data, sensor networks, logs or network traffic. Data is ingested and processed, leveraging data-pipelines that might consist of high efficiency queues such as Kafka or Flume, data processing frameworks such as Apache Spark or Apache Storm, and persisted in unstructured data stores like Elasticsearch, Cassandra, Hadoop or MongoDB.

A Data-First Application Architecture



The data pipelines are meant to extract insights from data rather than process a specific business transaction. Code and data frameworks are interleaved with each other in multiple layers. The focus is on processing data. Developers may manage and access this ever-changing, heterogeneous mass of data through a microservices and/or containerized architecture.

To provide a concrete example of this shift in application architectures, lets compare how e-commerce applications were built about a decade ago. In the early 2000s, most development was focused on processing and optimizing the shopping cart and checkout experience, integrating payments, merchandising, inventory management etc. Developers and architects were primarily focused on building applications that provided faster transactions, optimized business processes and provided better integration with other sub-systems. Fast forward to modern-day e-commerce architectures and the above are considered mostly solved problems and for the most part are commonplace. Increasingly, cutting-edge e-commerce vendors are now focusing on driving traffic to their site, personalizing customer experience, providing recommendations based on real-time trends, and providing dynamic pricing based on real-time analysis of competitor prices.

The new business differentiator is to convince the customer to buy, rather than processes the checkout transaction. Achieving a better conversion requires them to ingest and process data in data pipelines and make split second decisions on what to show the customer, maximizing conversion and share of wallet. We call the underlying application architectures that enable businesses to generate these real or near-real-time insights as “data-first” application architectures.

Benefits of the Data-First Approach

Several B2B domains, including marketing analytics, networking, security and IoT are leveraging real-time, or near-real-time insights to produce transformative business impact for their customers through:

- Achieving continuous visibility
- Providing more up-to-date information
- Enabling faster decision-making

Adapting quickly to evolving data sets makes it possible for an organization to stay on top of market trends and maintain a live, relevant experiential relationship with their customers. Like the code version of the Transformers® toys, data-first applications can be reshaped to serve new requirements in a rapid cycle. As such, data-first applications are rapidly becoming a core tenet of any digital business.

Challenges of the Data-First Approach

With the growing popularity of new open-source frameworks such as Kafka, Spark, Elasticsearch, Cassandra and Mesos, we now have a complete set of components available to rapidly build powerful data processing applications. However, like anything new in IT that delivers benefits, data-first applications also create a number of challenges.

- **Fragility:** The many moving parts and the relative newness of these systems means they they can be brittle.

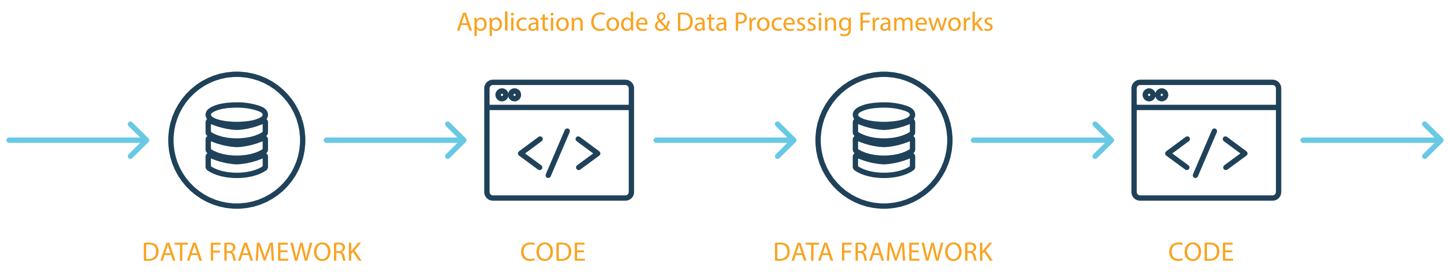
- **Management:** The development process may involve managing multiple teams across the application in production.

- **Communication:** With so many people involved, often working separately, it may be difficult to coordinate analyses and share knowledge about how the app is performing.

- **Complexity:** The number of moving parts and the interconnectedness of those parts make monitoring and troubleshooting a difficult and time consuming undertaking.

Why are Data-First Applications Challenging to Monitor?

Data-first applications are comprised of a combination of application code, data frameworks and the underlying infrastructure, which is increasingly becoming containerized. The application code and the underlying data frameworks are closely intertwined with each other. As a result, the line is blurring between your application and data processing tier. Monitoring and troubleshooting these complex and distributed data-first applications can be challenging due to the following factors:



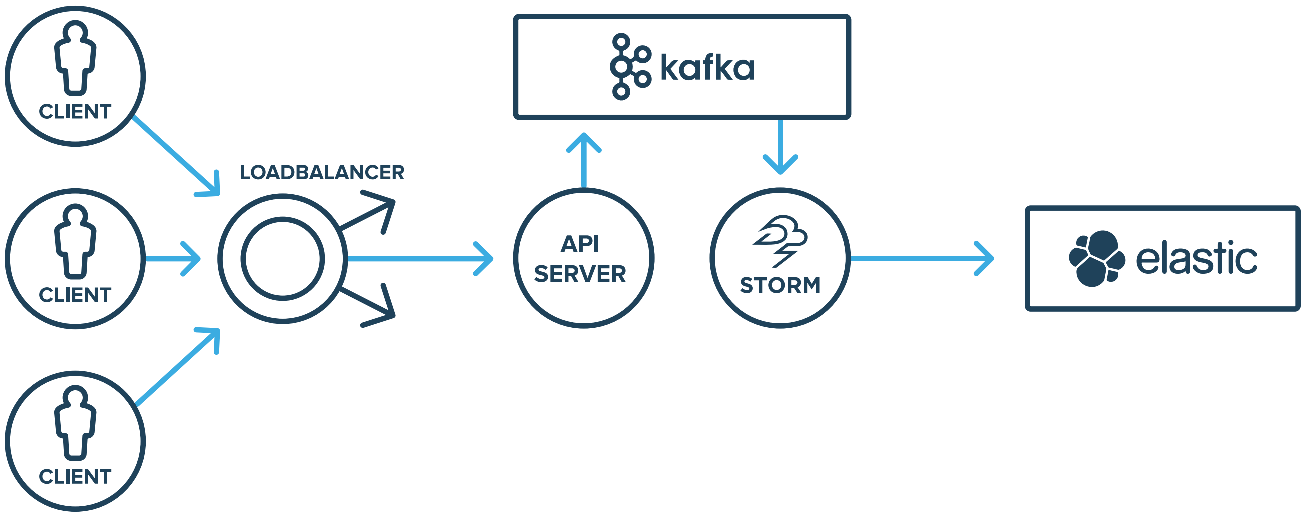
Steep Learning Curve: In a short-time, numerous data processing frameworks have appeared on horizon, each constantly changing with new ones popping up every day. Domain knowledge for these is relatively scarce. Identifying which metrics to collect and understanding the behavior of these metrics is by no means an easy undertaking. It can take you weeks or months to learn and configure this against generic monitoring tools.

Understanding the data pipeline: The core architectural tenet of a data-first application is the data pipeline. A data pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next. It is critical to understand the flow of data through each stage of the pipeline. Key performance concerns that matter across the pipeline include throughput, error rate, latency, backpressure, lag and data loss. Things can go wrong in any stage, it is therefore imperative that these be computed both per stage AND across the entire pipeline to have a holistic understanding of the health and performance of the pipeline.

Distributed and clustered: Each of these frameworks itself is composed of several components, usually deployed in a distributed environment. Apache Spark, for instance, consists of master, worker, application, driver and executor. The complexity of the pipelines grows exponentially as you stack up multiple data frameworks together, intertwined with your custom application code. To monitor this, you are collecting metrics and checks from several data frameworks, custom code and dozens or hundreds of hosts. Correlating issues across all of these to understand dependencies and analyzing root-cause requires a purpose-built solution that understands and deliberately ties this data together. Today, logs are the primary mechanism to identify and trace these issues. However, logs can be hard to read or query, lacking user context, spread across multiple servers and maybe even void of helpful details making this a laborious and slow process.

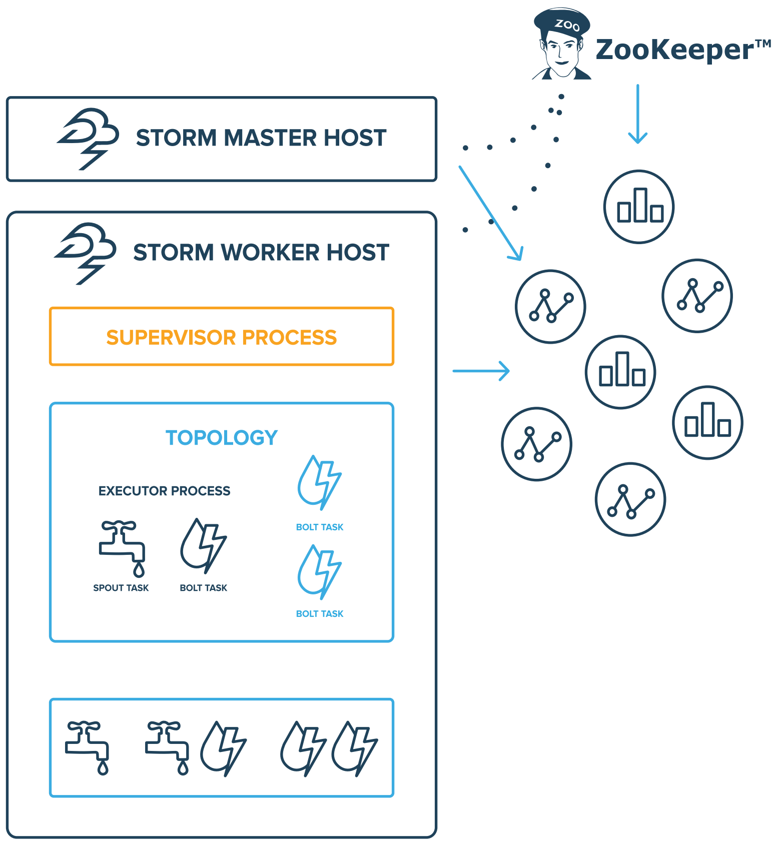
Intricately interconnected: By almost any measure, a data-first application is a complex system. The figure below shows a typical data pipeline. It is comprised of many parts - responsible for host management, process management, job management, data routing, resource allocation and recovery. These applications are deployed on a cluster of machines which may serve different functions, and have dependencies on other systems. Systems such as Storm, Spark and Kafka all fall under this category, as do Hadoop and Mesos for that matter. Problems that manifest in one part of the system can often originate in a completely different place. For example, let's say you detected that your Storm job is reporting a drop in throughput of data processed. Where do you start to look? Is the problem upstream in Kafka or the system that is feeding Kafka? Is it downstream in Elasticsearch where there's a problem writing data to the store? Is there something wrong with the Storm job itself? All of these are questions you might ask because each is a very real possibility! Because of the interconnected nature of these systems it is hard to tell where you should even start to look when something goes wrong.

A Typical Pipeline with Kafka, Storm and Elasticsearch



Ineffective existing tools: Even if you've isolated the issue to Storm, do you know if it's something wrong with your application code, or is something wrong the Storm cluster infrastructure or the ZooKeeper cluster which it depends on? Is it something wrong with one of the hosts that is a part of the Storm cluster, or one the storm executor processes running your application code? Again, all very real possibilities! Does your monitoring solution help you effectively figure this out?

Odds are that it does not, because most existing monitoring solutions were not designed to help you navigate such complexities. The problems we're dealing with here are a matter of domain depth, context, and correlation. Because such systems are so interdependent, it's important to correlate horizontally and understand how different parts of the pipeline relate to each other. The additional complexity of the individual components of the system make it important to correlate vertically as well, seeking an understanding of how the application, the service infrastructure and the system infrastructure affect each other.
Existing monitoring tools tend to try to solve the problem of collecting a whole bunch of data and presenting a whole bunch of data (usually in the form of graph dashboards), but they do a poor job of helping you navigate, much less understand, your data-first application, either vertically or horizontally. That leaves you to do all the heavy lifting and correlating of various graphs and charts to try and understand what is going on. Developers and operations engineers usually want to understand common performance concerns about the application, such as throughput, overall health, consumer lag or queue length, error rate and latency. They also usually want to know if the application has overcapacity or has headroom. Is there back pressure? Is there data loss? Can they onboard one more customer before adding more capacity? All of these data points are essential for delivering a high quality user experience.



STORM CLUSTER INFRASTRUCTURE

Solving the Monitoring Puzzle in Data-First Applications

When monitoring data processing frameworks in general, it is very easy to get lost in the sheer amount of data that's available. Every aspect of these complex systems generates telemetry, which is on top of the custom metrics their own applications generate. It is often unclear where to start looking when things go wrong. The first step in trying to solve this problem is properly organizing the information available. It is helpful to organize the information into a hierarchy of concerns, which you then investigate in order. Below we look at the case of Apache Storm, but this approach can be generalized to other data processing frameworks as well.

Storm Monitoring Concerns Hierarchy

Concern	Questions to Ask
Data Health (for a given application)	• Throughput: are we processing data at the expected rate? • Latency: are we processing data within the expected timeframe? • Error/quality: are there problems with the data being produced? • Input data: are input data streams flowing into storm behaving normally? For instance, what are the throughput rates for Kafka topics feeding into my Storm job?
Dependency Health	• Are systems feeding input into my storm job (such as Kafka) healthy? • Are the systems that my application is dependent on, such as Memcached or other API endpoints, healthy? • Is ZooKeeper healthy, given the Storm framework depends on it for job coordination?
Service Health	• Is the Storm master operating normally? If not, we will not be able to re-balance workloads or restart jobs.
Application Health	• Are my application KPI's within normal operating parameters?
Topology Health	• Are there resources assigned to the given Storm topology? • Are the Storm tasks and executors well distributed amongst the Storm cluster? • Are the performance counters (emitted, acked, failed, latency) for the given Storm topology normal?
Service Node Health	• Are the slotsUsed and slotsTotal on this given Storm node steady? • Are the aggregated performance counters across all Storm topologies on the given node normal?
Node System Health	• Are the key system metrics (load, cpu, memory, net-i/o, disk-i/o, disk free) operating normally?

Bringing it together

Retrieving all that data, configuring and calculating the desired aggregations, creating the desired dashboards and setting up the desired monitors is a monumental task. In the case of Apache Storm, it is quite a bit more difficult because there aren't a lot of options available when it comes to monitoring. Ideally, you'd like to have your application health and metrics, Storm metrics, and system metrics all in one place so you don't have to jump from tool to tool to find out the information you need. This also makes configuration of metric aggregations and monitoring easier. However, as your data workloads change, either by adding new data pipelines or growing the size of the clusters that support your pipelines, you'd like your monitoring solution to be able to keep up automatically, this continues to be a time consuming effort because you must repeat the entire setup process every time unless it can be automated.

Rethinking Monitoring

Monitoring data-first applications means rethinking the traditional approaches to monitoring. The existing monitoring tools are not able to organize or centralize the information needed to monitor and troubleshoot these data-first applications much less provide deep domain level expertise about the individual components, whether they be the data processing frameworks, associated applications and code, containerized infrastructure or the underlying schedulers and orchestrators for the data frameworks and microservices. Today's data-first applications are stretching application monitoring solutions, but they are really just the start of a new trend that is going to present DevOps practioners with far more complex applications over time. The pace of innovation in containerization, data processing and agile development methodologies is accelerating, not slowing down. As each successive wave of change hits the IT field, application monitoring will be that much more challenged to keep up.

A New Solution

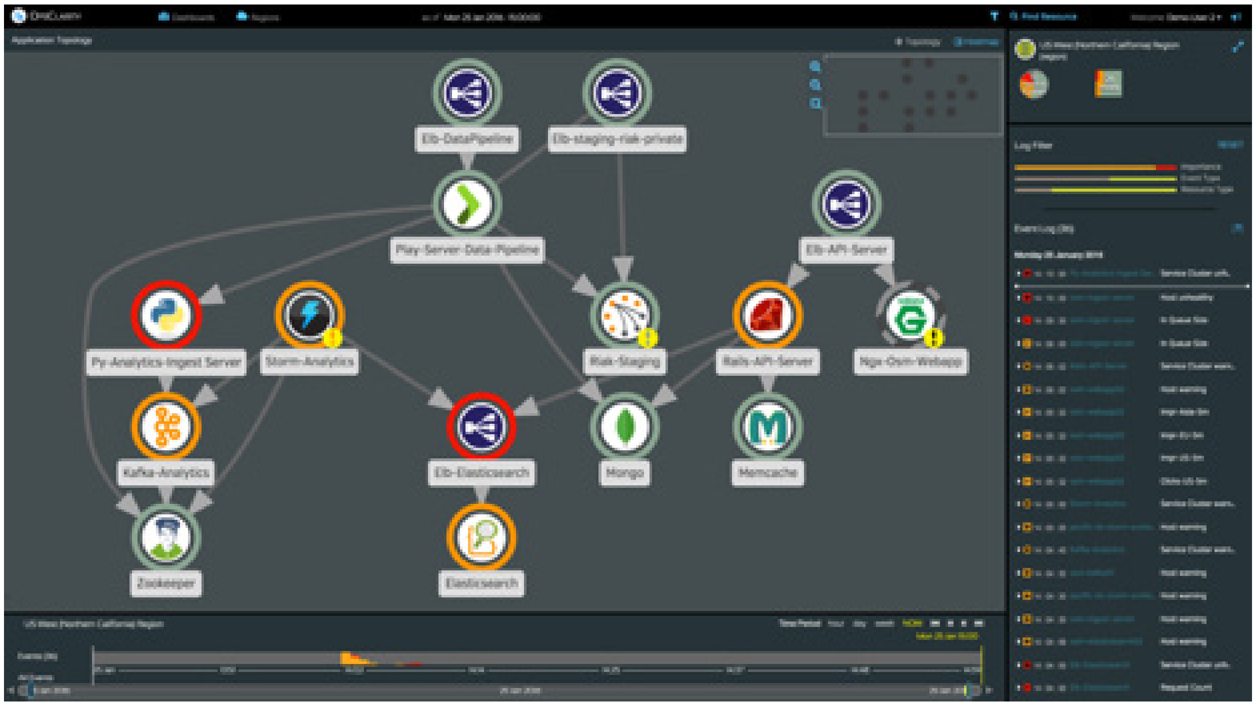
OpsClarity acknowledges that the application universe is in a constant state of change. By providing DevOps teams with a flexible, automated monitoring solution, OpsClarity offers a path to monitoring the Data-first application future, no matter how different it looks from today's already complex application landscape. We believe a monitoring solution should :

- Ingest huge amount of metrics from these systems in real time to provide the information you need, when you need it.
- Provide you with deep domain expertise which can then be codified in the monitoring tool as a combination of data science and advanced anomaly detection.
- Present you with an understanding of the specific failure patterns associated with the individual components and logical data pipeline as a whole.
- Automate the discovery of components that need monitoring.
- Automate the setup of monitoring itself to provide and efficient, effective and useful solution to for monitoring and troubleshooting data-first applications.

Intelligent Monitoring for Data-first Applications with OpsClarity

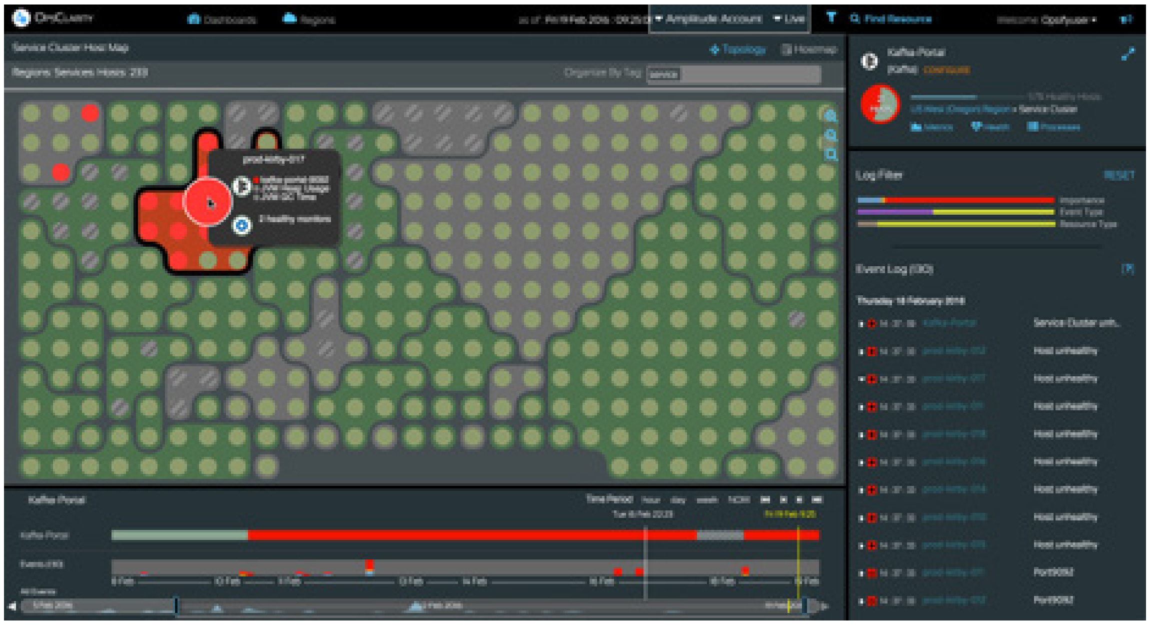
OpsClarity is a purpose-built monitoring solution for data-first applications. It provides real-time understanding of the health of your data pipelines and presentation of end-to-end performance monitoring of your apps, data frameworks and infrastructure in a single view. OpsClarity completely automates metric and metadata collection and leverages its deep domain expertise about the individual data frameworks to apply data science constructs such as anomaly detection and event correlation to rapidly troubleshoot issues.

Visual topology of the entire data-first application



OpsClarity provides a visual topology of the entire data-pipeline, overlaid with synthesized health of each service and cluster, enabling early identification and quick resolution of common concerns such as throughput, latency, error rate, back pressure etc.

Infrastructure hostmap of the data-first application



The OpsClarity Data Science Platform combines automated topology discovery with automated anomaly detection. The setup and configuration stage, which can take months with traditional monitoring technologies, can be accomplished in a matter of hours with OpsClarity's automated topology discovery significantly shortening the "time to value". A large-scale data visualization capability renders the performance metrics comprehensible to users without the need for a time-consuming metadata annotation and dash-board setup. Further, OpsClarity learns how to monitor a Data-first application better over time, detecting the signal from the "noise" of application processes and enabling a fast time to issue resolution.

OpsClarity Provides

Service-aware data collection for:

- Auto-discovery of the complete topology of the Data-first application, including operating systems, network and data.
- Auto-configuration of metric and metadata collection, with awareness of data streaming, code, containers and deployments.

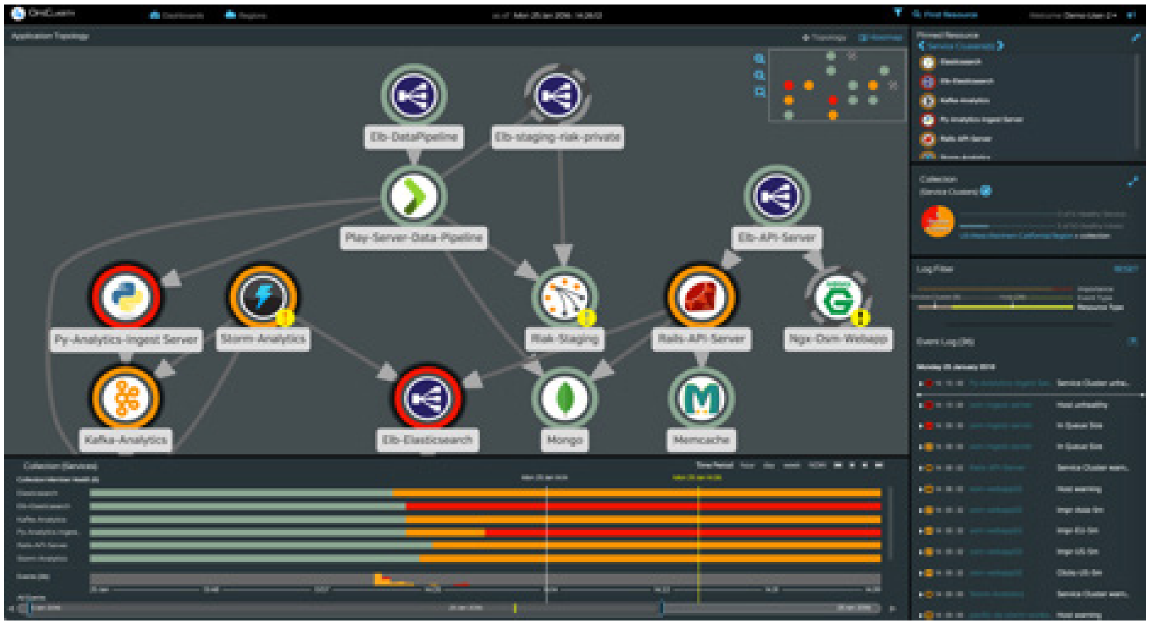
Intelligent health tracking and unified visibility to:

- Synthesize health for every service and infrastructure component in the Data-first application. Catch performance issues early on using its anomaly detection capabilities
- Centralize multiple signals into a single picture of pipeline an data-first application health.

Event correlation across services to:

- Keep administrators informed about application performance with proactive alerts and dashboards.
- Signal problems with smart notifications as they come into view, typically before they affect user experience.
- Enable rapid troubleshooting, with visibility across multiple service clusters and event replay for root cause analysis.

Correlated troubleshooting across the data-pipeline



Key Features

Automated Topology Discovery

Every data framework and associated application is automatically discovered and clusted to create a logical application topology.

Infrastructure Hostmap

Infasturcture map with real-time view of hosts and application services and data framework clusters.

Automated Performance Monitors

Understand system and data pipeline performance at a glance with real-time health information overlaid.

Event Timeline and Replay

Interactive timeline to replay system and pipeline state like a DVR.

Multi-Service Event Correlation

Data-science driven event correlation eliminates noise and drives focus.

Dashboards

Automatically curated dashboards of service, system and custom metrics for each resource and data framework.

Alerts

Get notified about performance issues that affects your business, application or data pipeline.

Instrument Your Applications

Full API access to capture any custom metrics that are special to your application or data pipeline.

If you would like to learn more about monitoring and troubleshooting data-first applications using OpsClarity, please visit us at www.opsclarity.com or contact us at info@opsclarity.com