White Paper

# Using Apache Spark™, Apache Kafka®, And Apache Cassandra® To Power Intelligent Applications

## Overview

Apache Cassandra® is well known as the database of choice for powering the most scalable, reliable architectures available. Apache Spark™ is the state-of-the-art advanced and scalable analytics engine. Apache Kafka® is the leading stream processing engine for scale and reliability. Deployed together, these technologies give developers the building blocks needed to build reliable, scalable, and intelligent applications that adapt based on the data they collect.

This paper discusses the use cases, architectural pattern, and operations considerations for deploying these technologies to deliver intelligent applications.

instaclustr

# Use Cases

## Internet of Things

At the core of an IoT application there is a stream of regular observations from (potentially) a large number of device or items with embedded electronics (e.g. switches, sensors, tags). A stream of IoT data is just "big data", but analyzing that big data in a way that drives actions or recommendations or provides information is where the application delivers value.

Apache Cassandra is extremely well suited to receiving and storing streams of data. It's always-on availability matches the constant stream of data sent by devices to ensure your application is always able to store data. In addition, its native storage formats are well suited to efficient storing and using time series data such as that produced by IoT devices. The scalability of Apache Cassandra means you can be assured that your datastore will smoothly scale as the number of devices and stream of data grows.

The powerful analytics capabilities and distributed architecture of Apache Spark is the perfect engine to help you make sense and make decisions based on the data you're receiving from your IoT devices. Spark's stream processing can quickly determine answers from short-term views of your data as it's received. For analysis running over longer time periods, the Spark Cassandra connector enables Spark to efficiently access data stored in Cassandra to perform analysis.

In this context, Apache Kafka is often used as a reliable message buffer. In many IoT scenarios, the flow of data from devices is constant and the devices have very limited capacity to buffer data in the event the central processing service is unavailable. Events from the devices can be written to Kafka when first received and then picked up and processed by the downstream applications. This ensures events are not lost even if the processing elements for the central system become backed up or suffer downtime. In addition, use of Kafka in this manner easily allows additional consumers of the event stream to be added to the system. For example, your initial implementation may have a simple application that just saves data to Cassandra for later use but you then you add a second application that performs real time processing on the event stream. Kafka Streams may also be used as an alternative to Spark Streaming for real time stream processing.

## Financial Services

The pressures for financial services companies to gain a technological edge in data processing are coming not only from the competition but also from consumers. Gaining a competitive edge requires systems that can collect and quickly analyze vast streams of data. Consumers expect that the systems they interact with will be instantly up to date, always available and, increasingly, be aware of the context of all their previous interactions and related information.

Addressing these joint pressures, while containing technology costs, requires the adoption of new generation of architectural patterns and technologies. Apache Cassandra, Apache Kafka, and Apache Spark are technologies that are ideally placed to form the core of such an architecture. The applicability of these technologies in financial services has been proven many times by leading organisations such as ING and UBS.

One common application we see for Cassandra in financial services is as a persistent cache to support high volume client requests. In particular, we see this requirement with banks implementing the Payment Services Directive (PSD2) in the EU. This leverages Cassandra's extreme reliability and built-for-the-cloud architecture to enable financial services organization to deliver always-on service and avoid the high cost of scaling their legacy (often mainframe) architectures to meet increased client interactions needs. Spark is often included in this architecture to enrich the cached data with sophisticated analysis of trends and patterns in the data enabling user-facing applications to make this analysis with interactive response times. Kafka often sits in this picture as a message bus to connect the core processing system to multiple downstream consumers.

## Others

The two use cases above are great examples where we see regular adoption of Spark and Cassandra. However, there are many other business problems where the two technologies can combine to provide an ideal solution. Some examples that we have seen include:

- **Ad-tech**: Relying on the low-latency (low double digit ms) responsiveness and always-on availability of Cassandra to make online advertising placement decisions backed by deep analysis calculated with Spark. Massive flows of inbound events and information can be managed with Kafka.

- **Application Monitoring**: We use a combination of Spark and Cassandra in our own monitoring system that monitors close to 1500 servers. Cassandra seamlessly handles a steady stream of writes with metrics data while Spark is used to calculate regular roll-ups to allow viewing summarized data over long time periods. Kafka acts as a centralization point for the messages and also a message buffer.

- **Inventory Management, particularly in travel**: Uses Cassandra to track inventory records and Spark to analyse available inventory to determine dynamic pricing, capacity trends, etc.

# Architectural Patterns

## Batch Updating

At a more rudimentary level, many Cassandra applications have a need for periodic batch processing for data maintenance. While this can include summarization, it can also include requirements like implementing complex data expiry rules. Running these batches through a single threaded (or single machine) batch engine will not scale to the same extent your Cassandra cluster will. Implementing these batch jobs in Spark not only provides a pre-built set of libraries to assist with development of the data processing functionality but also the frameworks to automatically scale the jobs and scale and execute processing logic on the same servers where the data is stored.

## Stream Enrichment

For most applications, a strong design will store in a single Cassandra table all of the information required to services a particular read request (i.e. the data will be highly denormalized). In some cases this denormalization process will require calculating or looking up additional data to add to a stream before the stream of data is saved. Using Spark Streaming to process data before saving to Cassandra provides a scalable and reliable technology base to implement this pattern. Kafka Streams is an alternative engine for implementing this form of stream enrichment.

## Lambda Architecture

The Lambda Architecture is an increasingly popular architectural pattern for handling massive quantities of data through both a combination of stream and batch processing. With the Lambda Architecture, you maintain a short-term, volatile view of your data (the speed layer) and a longer term, more permanent view (the batch layer) with a service to join views across the two (the serving layer).

With Spark and Cassandra, you have the key architectural building blocks you need to implement the Lambda Architecture. Spark Streaming is an ideal engine for implementing the speed layer of the architecture (potentially with results stored in TTL'd tables in Cassandra) while Spark can also be used to perform the longer-term batch calculations and store results in Cassandra.

## Kappa Architecture

The Kappa Architecture takes the next step from the Lambda Architecture, removing the batch layer and treating the stream of events as the immutable record of system state. Stream processing maintains summary views as the stream is processed. If the logic of summary views needs to change then the stream processing logic is updated and the saved streams reprocessed. The Kappa Architecture removes the need to maintain separate stream and batch logic that is required for the Lambda Architecture.

Once again, the combination of Spark and Cassandra gives you the architectural components you need to implement Kappa Architecture. Spark Streaming is an ideal processing engine to undertake the calculations needed on the stream of data. Apache Cassandra can be used both as the long term, immutable store of the data stream, and as a store for the results of the stream calculations that are used by the serving layer. An alternative is to use Apache Kafka as your immutable event store and Apache Cassandra as the store for the materialized views calculated based on these events.

## Operations

Operating as part of a mission-critical application is the normal mode of operation for Cassandra and there is a well established body of knowledge about how to operate Cassandra to achieve the highest levels of availability. Although Kafka is a little newer it is also widely operated at the highest levels of scale and reliability.

Spark, on the other hand, is often run to provide an analytics environment for use by a small number of data scientists. In this situation, reliability and predictable performance are not as critical as when Spark is deployed as a component of a production application. This section of the paper describes some of the considerations to be applied when deploying Spark for production usage.

### Management Environment

The key to reliable operations of any technology is to have a solid overall management environment including aspects such as:

- Automated (or at least well controlled) deployment and configuration management;
- High quality testing of new configurations prior to deployment;
- Backup and disaster recovery procedures;
- Appropriate monitoring and systems and people that are paying attention to what is being reported by that monitoring;
- Rigorous incident response procedures and well-trained staff.

None these items is specific to Kafka, Spark, or Cassandra. However, introducing production usage of these technologies will require examination of each of these areas to ensure they are fit for purpose with introduction of new architectural components and applications.

## High Availability

One specific area to be considered is high availbility architecture (ensuring your overall service continues to run even when components fail). Cassandra is effectively high-availability by default—if you use multiple machines and a basic, competent setup you will have a high-availability cluster. Of course, there is more you can do for the absolute high level of availability. Kafka follows a somewhat similar architecture and has similar considerations in terms of distributing data across multiple replicas and placing replicas in multiple availability zones.

For Spark, more detailed consideration is required. Spark by default is resilient to the failure of worker processes with work being automatically redistributed to running workers should a worker fail. However, the Spark Master and Driver require further consideration. Apache Spark has built-in capability to make the Spark Master highly available by using an Apache Zookeeper™ cluster to control the election of which machine will be the active Master at any point in time.

For the Driver component (that submits jobs to the cluster), it is possible to configure Spark to automatically retry jobs that fail. To enable this, the job must be submitted in cluster mode (--deploy-mode:cluster) and with the --supervise flag set. As this will restart failed jobs from scratch, it is necessary to ensure your jobs are idempotent when using this functionality.

## Monitoring

It is important for any production system to have quality monitoring in place to help detect and diagnose problems. This starts with basic operating-system level monitoring of metrics such as CPU load and free disk space. It should then extend to monitoring that the expected system processes are running.

For Cassandra and Kafka, a broad range of metrics are available out of the box and are sufficient to monitor usage of Cassandra and Kafka for the vast majority of use cases. It will likely be necessary to tune alerting thresholds for your application, but the important metrics to monitor are fairly standard and well known.

Spark also provides built-in monitoring capabilities including a UI to allow to you to review the progress of your jobs. However, given the extremely diverse nature of workloads that Spark can handle it will also likely be necessary to implement error handling and reporting as part of your production jobs as well as relying on the native Spark metrics.

### Workload Isolation

One of the unique advantages of Cassandra is its ability to provide workload isolation through its native multi-data center architecture support. By setting up two Cassandra "data centers" in the same physical data center (or cloud provider region) you can isolate the loads of your Spark analytic reads to a single data center, ensuring processing capacity and response times of your online process are minimally impacted when batch processing runs in Spark.

## Conclusion

Cassandra, Kafka, and Spark form a powerful combination for many use cases. However, architecting and running distributed technologies at scale and with the highest levels of reliability and security requires a specialist environment including tools such as monitoring, management processes, and skilled and experienced staff.

Instaclustr's focus is the provision of the world's best managed environment for running open-source, distributed technologies reliably, at scale. We bring to your application a proven management platform and more than 100 million node hours of experience running these technologies in production.

## About Instaclustr

Instaclustr helps organizations deliver applications at scale through its managed platform for open source technologies such as **Apache Cassandra®**, **Apache Kafka®**, **Apache Spark™**, **Redis™**, **OpenSearch®**, **PostgreSQL®**, and **Cadence**.

Instaclustr combines a complete data infrastructure environment with hands-on technology expertise to ensure ongoing performance and optimization. By removing the infrastructure complexity, we enable companies to focus internal development and operational resources on building cutting edge customer-facing applications at lower cost. Instaclustr customers include some of the largest and most innovative Fortune 500 companies.