



**hazelcast** **JET**

Rahul Gupta

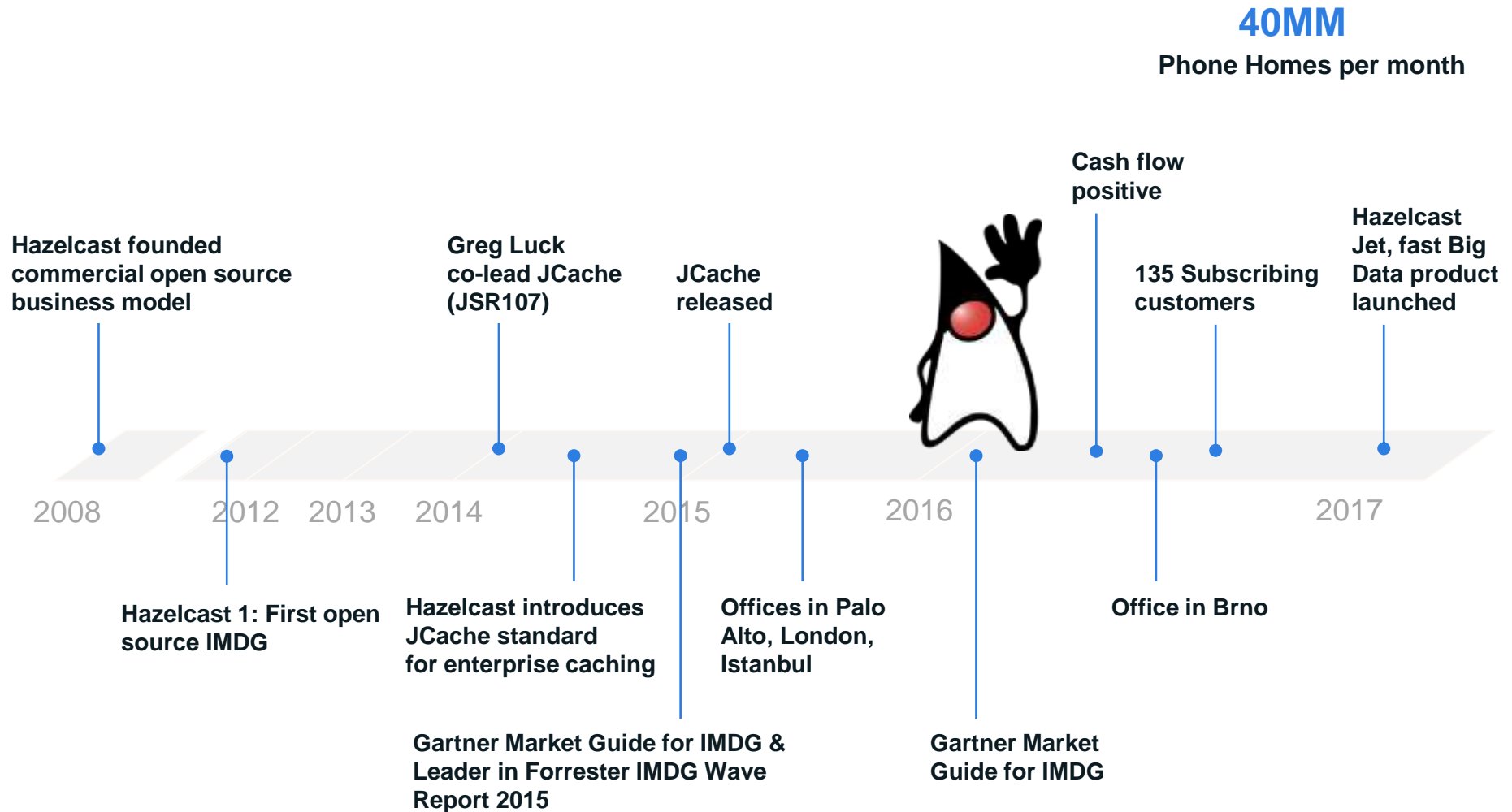


Hazelcast Jet is a distributed computing platform for fast processing of big data sets. Jet is based on a parallel core engine allowing data-intensive applications to operate at near real-time speeds.

**DISTRIBUTED COMPUTING. SIMPLIFIED.**

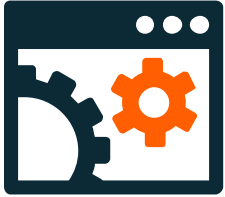


# Hazelcast Company Snapshot





# Why Hazelcast Jet?



**Performance – Low-latency at scale**



**Highly optimized reading from and writing to IMDG**



**Embeddable and light-weight**



# Hazelcast Jet Key Differentiators

## HIGH PERFORMANCE

- Distributed computing platform enables fast processing of big data.
- Parallel and low-latency core engine allows data-intensive applications to operate at real-time speeds.
- Cooperative multi-threading architecture allows operation of thousands of jobs simultaneously.
- Scales both vertically and horizontally.

## PRODUCTIVE

- One solution provides both batch and stream processing.
- Based on Hazelcast legendary ease-of-use.
- Variety of connectors enable easy integration into data processing pipeline.
- Scaling, failure handling and recovery are all automated for ease of operational management.
- Exactly-Once or At-Least Once guarantees.
- Jet team provides quick response to new feature requests.



## FLEXIBLE INTEGRATION

- Operates as shared application infrastructure or is embedded directly in applications as a JAR- making big data processing an application concern.
- Ideal for microservices with light-weight footprint --each Jet processing job is launched within its own cluster maximizing service isolation.
- Light-weight makes manipulation with Jet easy for both developers and DevOps.
- Cloud and container ready. Pivotal Cloud Foundry, OpenShift, Docker, Azure, AWS, Kubernetes, ...

## ENTERPRISE GRADE STORAGE

- Embeds Hazelcast IMDG as an operational storage for enterprise-grade reliability and resilience.
- Most widely deployed IMDG with tens of thousands of deployments in production worldwide.
- High-Performance integration eliminates network transit latency penalties from moving data back and forth.
- Eliminates the integration issues between processing and storage.



# Hazelcast Jet Key Competitive Differentiators

- High Performance | *Industry Leading Performance*
- Works great with Hazelcast IMDG | *Source, Sink, Enrichment*
- Very simple to program | *Leverages existing standards*
- Very simple to deploy | *Embed 10MB jar or Client Server*
- Works in every Cloud | *Same as Hazelcast IMDG*
- For Developers by Developers | *Code it*



# Hazelcast Jet Product Overview



# Hazelcast Jet Architecture







# Hazelcast Jet Use Cases

## Real-time Stream processing



- Big Data in near real-time
- Plugged into data processing pipeline
- Distributed, in-memory computation
- Aggregating, joining multiple sources, filtering, transforming, enriching
- Elastic scalability
- Super fast
- High availability
- Fault tolerant

## Distributed In-Memory Computation



- Simple, modern API
- Distributed, in-memory computation
- Transforming and querying distributed data structures
- Computation close to the data
- Distributed clustering
- High availability
- Fault tolerance

## Data-Processing Microservices



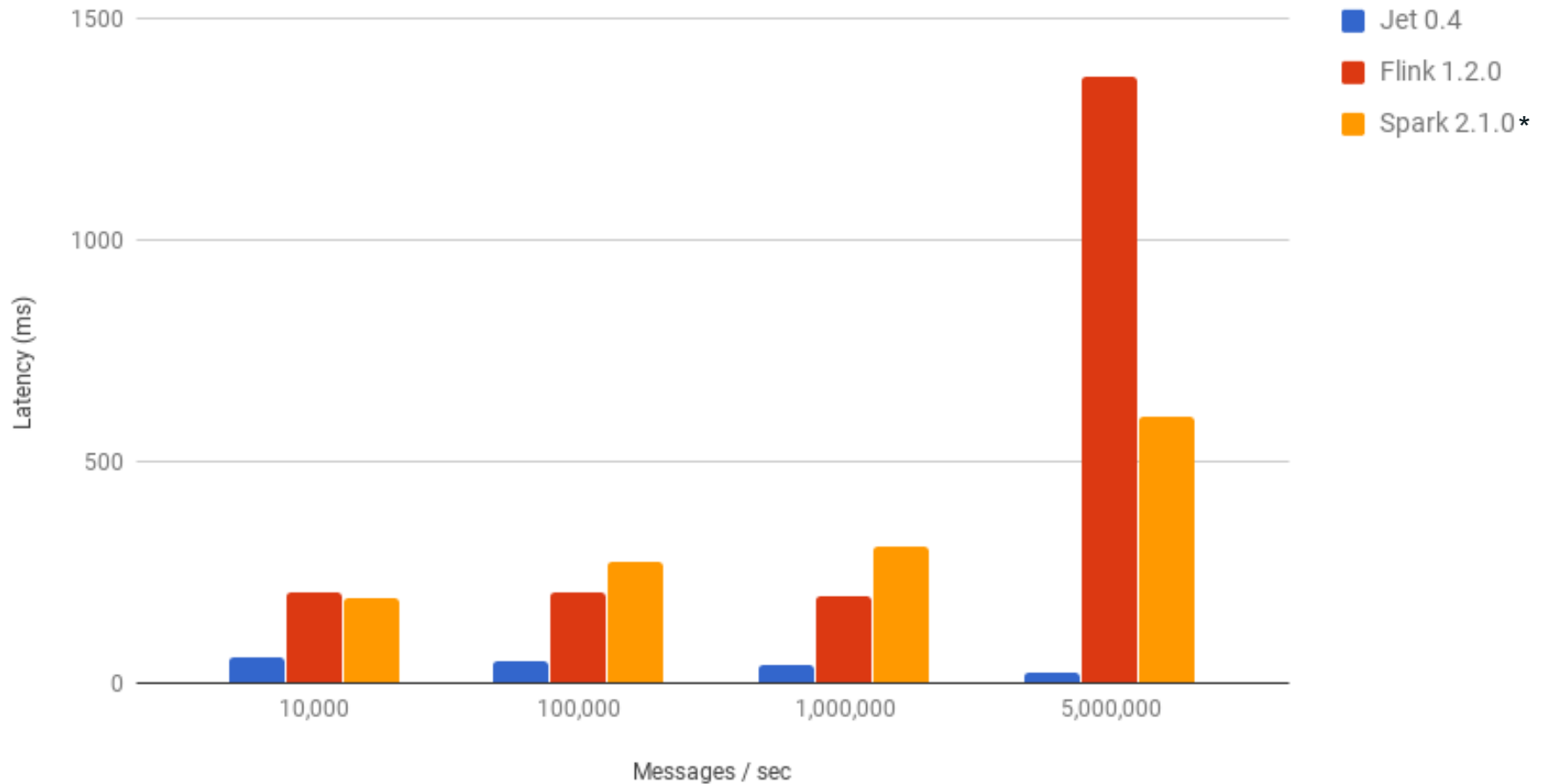
- Data-processing microservices
- Isolation of services with many, small clusters
- Service registry
- Network discovery
- Inter-process messaging
- Fully embeddable
- Spring Cloud, Boot Data Services



# Jet Streaming Performance

Streaming Trade Monitor - Average Latency (lower is better)

1 sec Tumbling Windows

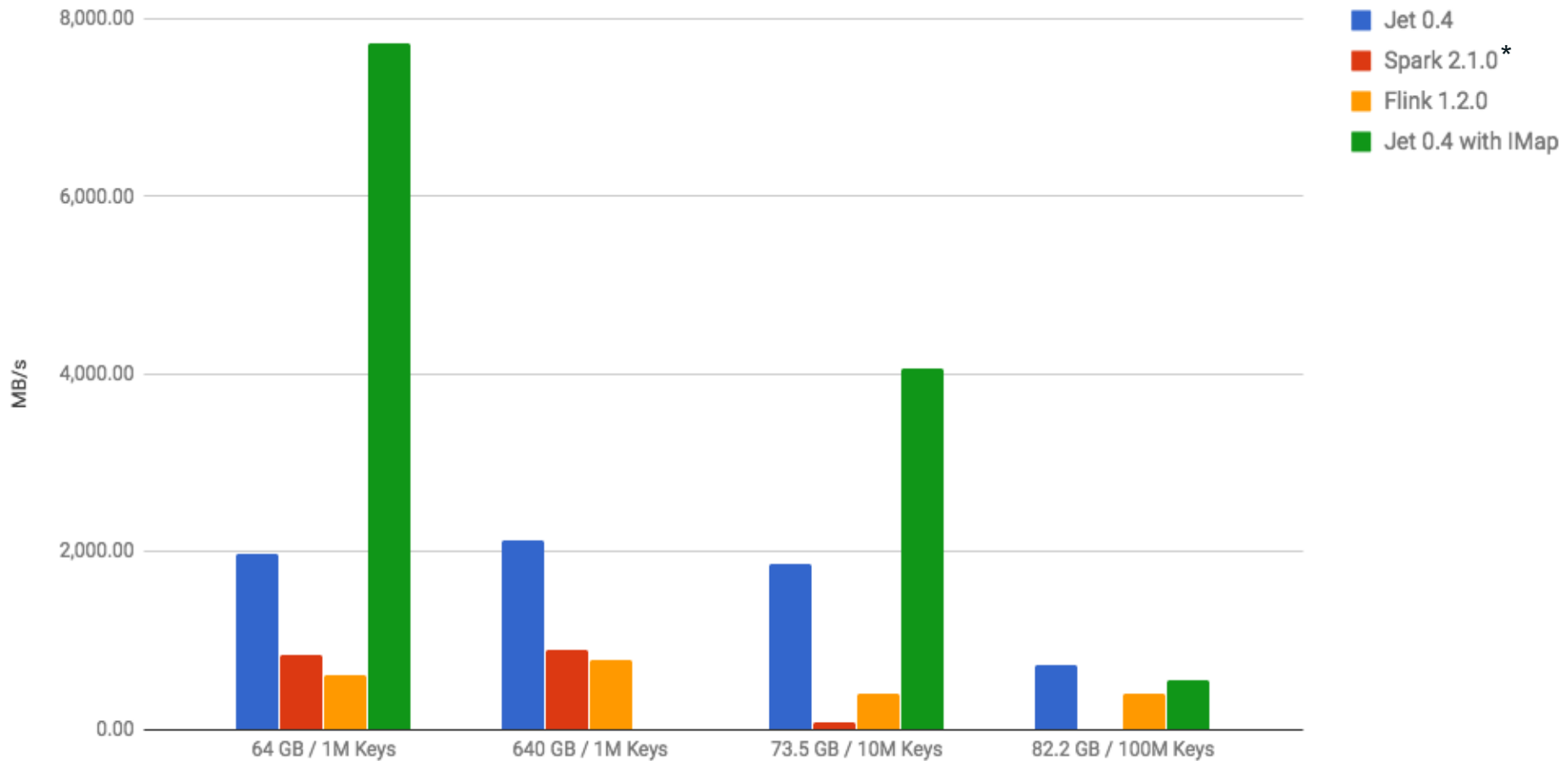


\* Spark had all performance options turn on including Tungsten



# Performance - Throughput

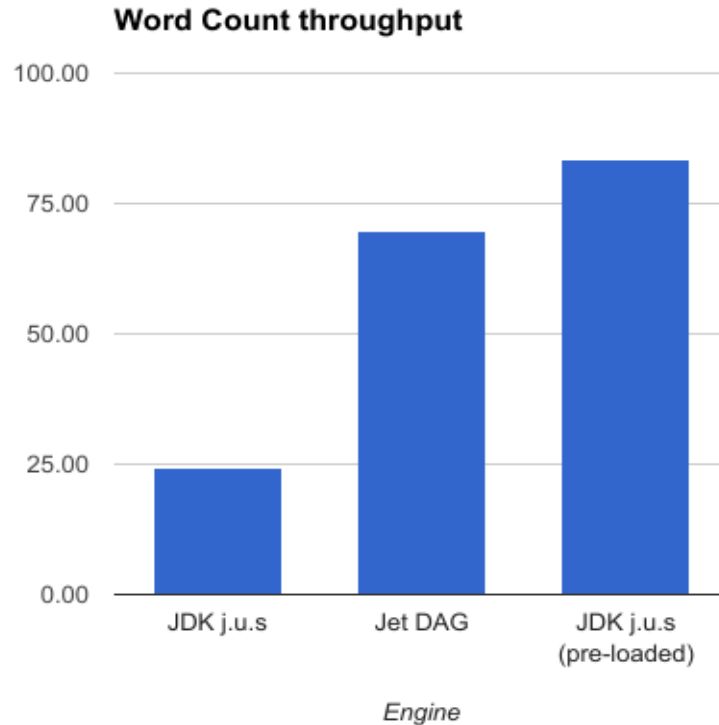
Word Count Benchmark - Throughput (MB/s)



\* Spark had all performance options turn on including Tungsten



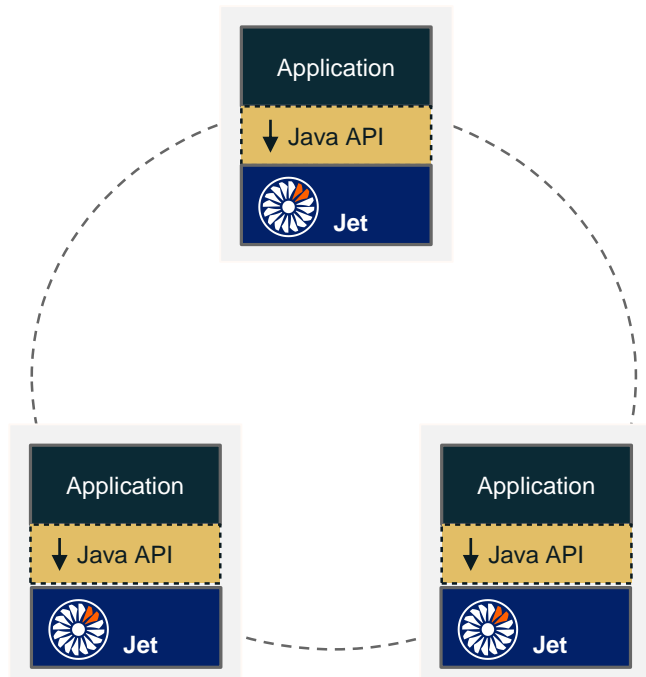
# Performance - ForkJoin



- Jet is faster than JDK's j.u.s implementation. The issue is in the JDK the character stream reports "unknown size" and thus it cannot be parallelized
- If you first load the data into a List in RAM then run the JDK's j.u.s it comes out a little faster

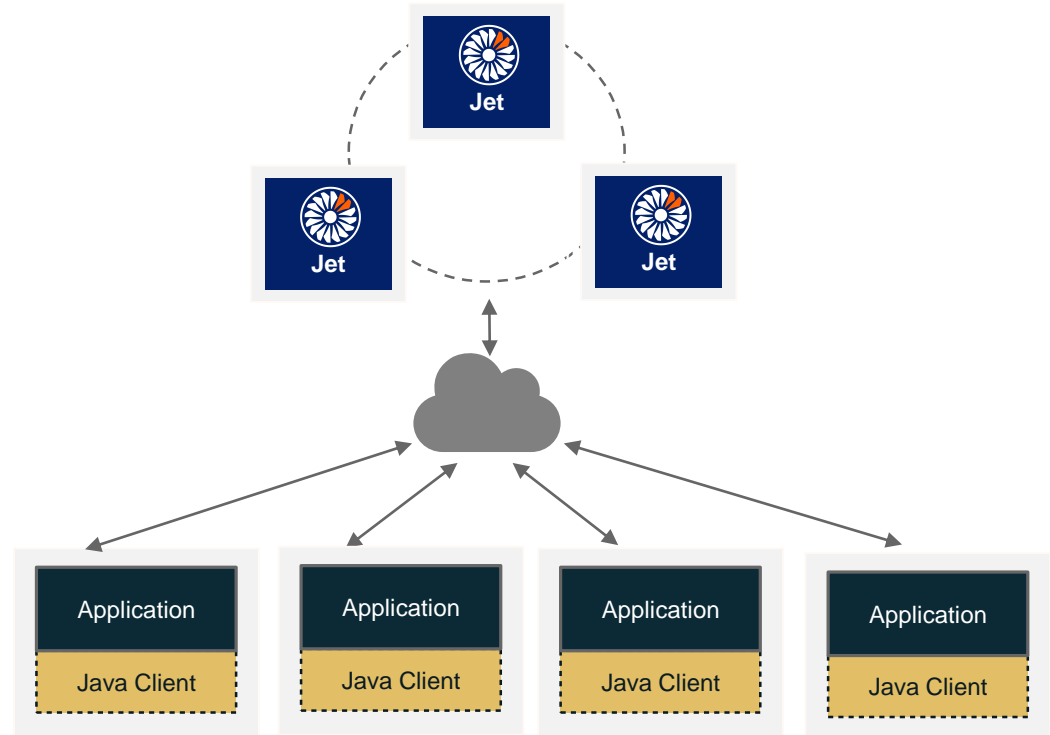
# Jet Application Deployment Options

## Embedded



- No separate process to manage
- Great for microservices
- Great for OEM
- Simplest for Ops – nothing extra

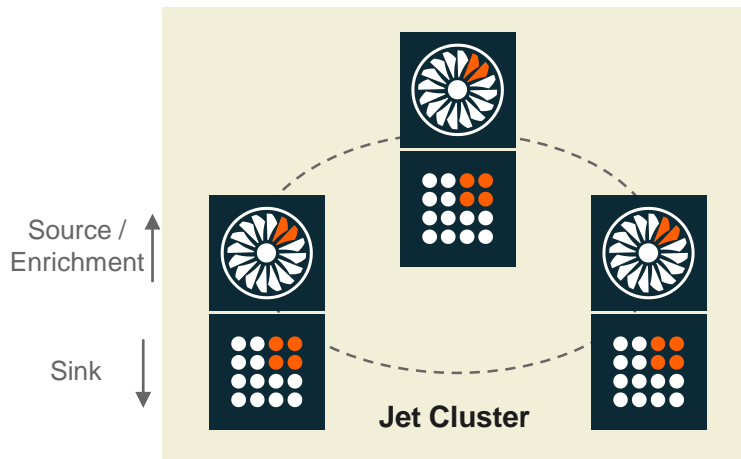
## Client-Server



- Separate Jet Cluster
- Scale Jet independent of applications
- Isolate Jet from application server lifecycle
- Managed by Ops

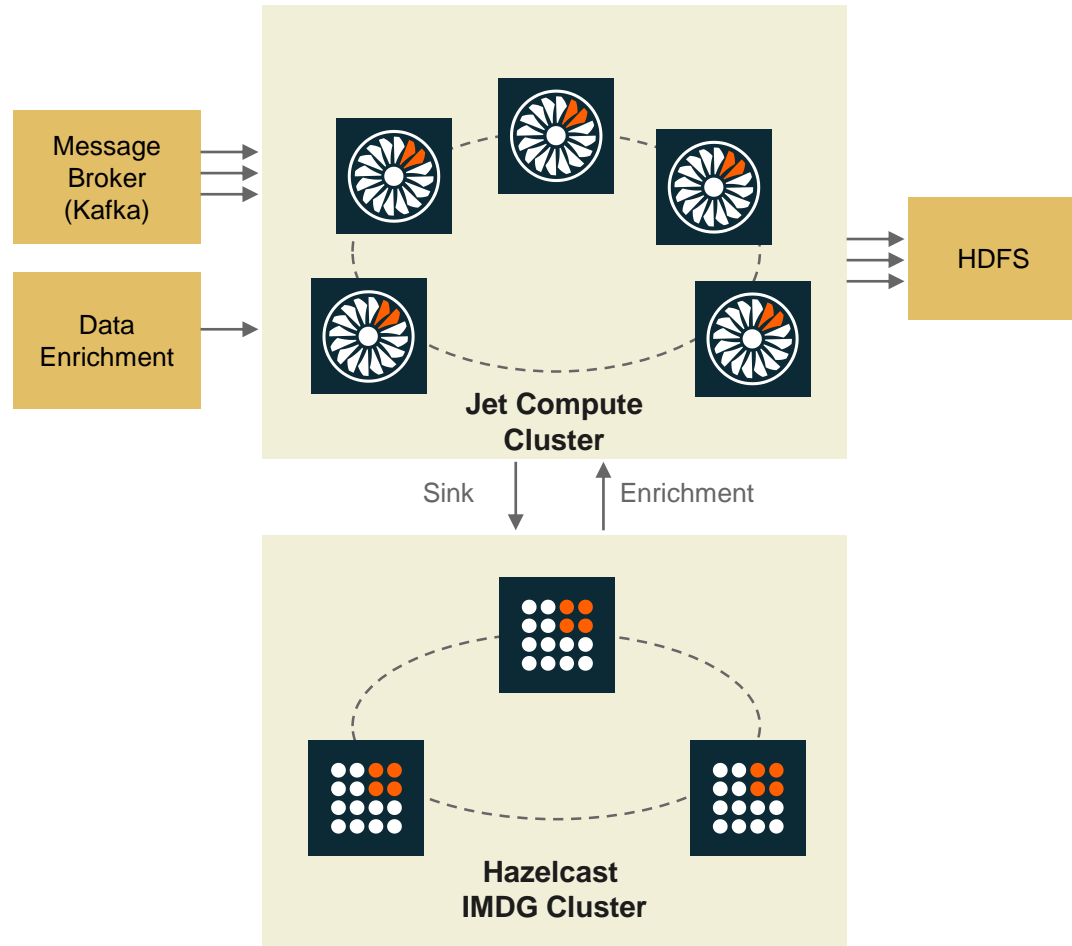


# Jet with Hazelcast Deployment Choices



## Good when:

- Where source and sink are primarily Hazelcast
- Jet and Hazelcast have equivalent sizing needs



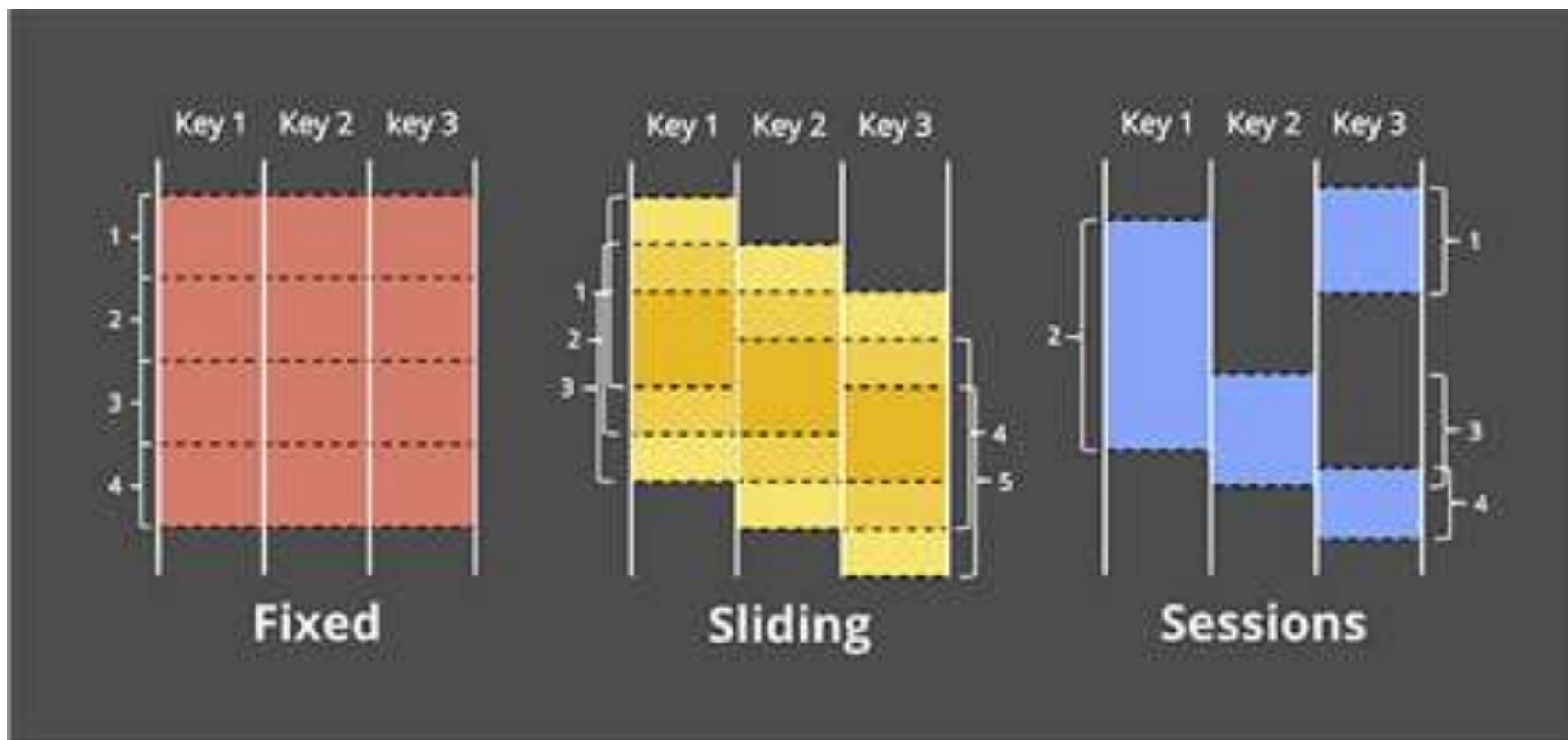
## Good when:

- Where source and sink are primarily Hazelcast
- Where you want isolation of the Jet cluster



# Stream Processing

- Support for events arriving out of order via Watermarks
- Sliding, Tumbling and Session window support





# Fault Tolerance – Distributed State Snapshots



**Exactly-Once, At-Least Once or No Guarantee** to optimize between performance and correctness



**Distributed State Snapshots** to back-up running computations



**Resilience** with backups distributed and replicated across the cluster to prevent losing data when member fails



**Simplicity** as the snapshots are stored in embedded in-memory structures. No further infrastructure is necessary





# Job Management & Fault Tolerance

- Job state and lifecycle saved to IMDG IMaps and benefit from their performance, resilience, scale and persistence
- Automatic re-execution of part of the job in the event of a failed worker
- Tolerant of loss of nodes, missing work will be recovered from last snapshot and re-executed
- Cluster can be scaled without interrupting jobs – new jobs benefit from the increased capacity
- State and snapshots can be persisted to resume after cluster restart



# Processing Guarantees

Guarantee	Snapshots	Performance
None	No	Fastest
At-Least Once	Yes	Slower
Exactly-Once	Yes	Slower

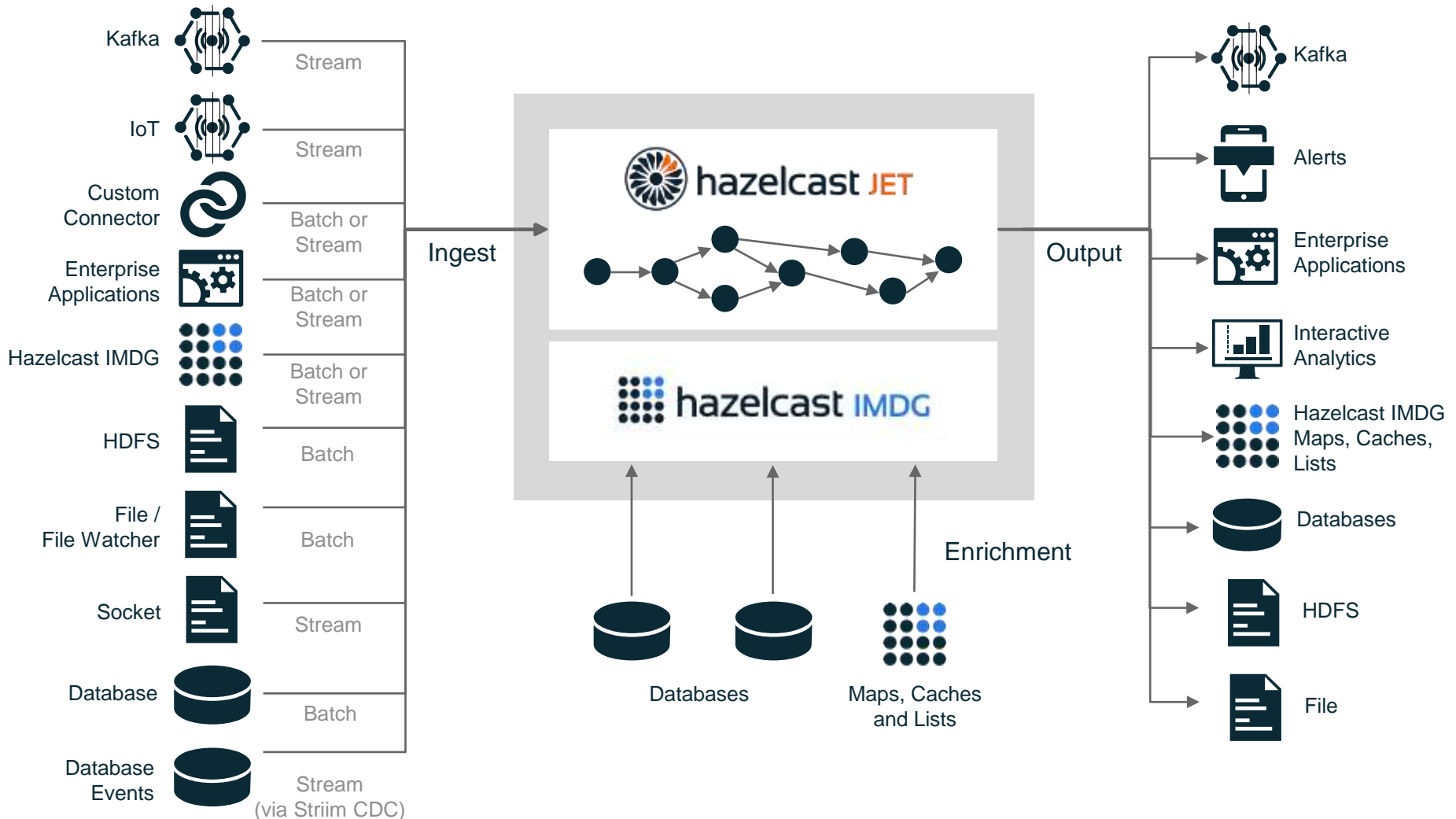


# Data Input and Output

- Hazelcast **ICache (JCache)**, (batch and streaming of changes)
- Hazelcast **IMap** (batch and streaming of changes)
- Hazelcast **IList** (batch)
- HDFS (batch)
- Kafka (streaming)
- Socket (text encoding) (streaming)
- File (batch)
- FileWatcher (streaming – as new files appear)
- Custom, as sources and sinks are blocking **Processors**



# Stream and Batch Processing





# Business Use Cases



# Field Equipment Monitoring and Analytics

## Hazelcast Jet Enables Real-time Monitoring of Sensors and Processing of Streaming Data from Field Equipment

Industries such as Oil & Gas use sensor data to automate decision making. Events reported from oil rig sensors are evaluated in the field so that rig settings are adjusted in real time. Hazelcast Jet is ideal for applications that require near real-time insight. Deployment simplicity makes it a optimal choice for both remote field and data center applications.

### Why most processing solutions fall short...

Traditional data processing frameworks don't allow direct embedding into an application or independent system, forcing users to install them as a secondary standalone server complicating deployment of the solution.

Field data processing spans from sensors to on-rig processing, and from there to processing within the data center --involving multiple execution environments and hardware architectures. Traditional tools require different runtimes (single-node, multi-node, different architectures), resulting in higher system TCO.

Traditional complex event processing solutions are built for single machine/server deployments and cannot be scaled out easily.

Most processing solutions experience growing latencies as the workload throughput increases.

### Why is Hazelcast Jet ideal?

Jet can be used as an embedded library --avoiding need for separate server tier. Jet doesn't have any dependencies, so no further infrastructure necessary to run a cluster. Jet allows users to build self-contained applications.

Jet is a lightweight tool with a small memory footprint. There is just one distribution to be used among all deployments. Running on a JVM, Jet can be used on a variety of hardware platforms.

Jet can be scaled out easily, as it's built on top of a massively parallel, distributed computing core (Hazelcast IMDG). It's designed to scale out.

Jet is designed for low latency under the most grueling workloads with latency remaining relatively flat even up to 5,000,000 aggregations per second.



# Field Equipment Monitoring and Analytics Case Study



## CUSTOMER SUCCESS

Leading oil & gas system integrator, specializing in acquisition, persistence, secure transportation and dissemination of high-frequency sensor data

10's of 1000's of events per second are gathered throughout the process



## CHALLENGE

- Use sensor from oil wells data to automate decision-making:
  - Keep latency low and consistent at scale, while system turns insights into decisions executed in a closed loop
  - Simplify system deployment and maintenance in remote locations with limited bandwidth
  - Use standards that extend system viability and maintenance lowering TCO
- Requires high application availability/performance for early issue detection to avoid production loss and optimize well productivity

## SOLUTION

- Hazelcast Jet is the processing backbone of application monitoring well sensors. API connects multiple sensor data streams with varying formats and frequency. After joining and filtering, sliding and session windows compute data insights to decisions
- Embedded Hazelcast IMDG is operational data store for easy scaling, whether on bare metal or in AWS
- Real-time analysis and correlation of event data from various devices and sensors throughout the process allow immediate action. Jet adjusts rig settings in real time.

## WHY HAZELCAST JET

- Performance and scalability
- In-memory data store with parallel processing enables scalable real-time analytics
- Open source, standards-based avoids vendor lock-in



# Payment Processing

## Hazelcast Jet Modernizes Payment Processing, Accelerating Bank Performance and Improving Customer Experience

Hazelcast Jet is an ideal technology for payment processing systems as it offers the lowest end-to-end processing latencies on a consistent basis. This is the result of Jet combining data processing and messaging in a single solution with elastically scalable storage for caching data from 3rd party systems.

### Why most processing solutions fall short...

Most processing solutions experience growing latencies as the workload throughput increases.

Most data processing solutions rely on 3rd party tools for messaging and storage increasing deployment and maintenance complexity.

Traditional solutions for service orchestration are built for single machine/server deployments and cannot be scaled out easily.

### Why is Hazelcast Jet ideal?

Jet is designed for low latency under the most grueling workloads with latency remaining relatively flat even up to 5,000,000 aggregations per second.

Jet embeds Hazelcast IMDG, which provides messaging and storage capabilities in a single solution simplifying deployment and maintenance while lowering TCO.

Jet scales easily, as it's built on top of a massively parallel, distributed computing core (Hazelcast IMDG). It's designed to scale out. The elasticity of a Jet cluster allows adding and removing computing and storage resources in accordance with changing processing demand.





# Payment Processing Case Study



## CUSTOMER SUCCESS

**A global information  
technology solutions  
company**

**Processing 10's of 1,000's  
of payments per second  
today.**

**Built-in scalability to  
support future business.**

## CHALLENGE

- Before settling a transaction, payment processing systems check the merchant details by forwarding them to the card's issuing bank or association for verification, and carry out anti-fraud measures
- Each step in this pipeline requires the lowest possible latency to deliver a positive customer experience
- With 24/7 global operations and hard SLAs, resiliency and automatic recovery are a must-have

## SOLUTION

- Within the payment processing application, Jet acts as the pipeline for each payment process step
- The payment management application orchestrates XML payment instructions and forwards them to the respective card's issuing bank or association for verification, then carries out anti-fraud measures before settling transactions
- Multiple Jet processing jobs are pipeline components. Hazelcast IMDG distributed IMaps are used for transaction ingestion and messaging

## WHY HAZELCAST JET

- High-performance connectors between Jet and IMDG enable low-latency operations; consistent low latency of the Hazelcast platform keeps the CGI payment management application within strictest SLA requirements
- Automatic recovery of the Jet cluster achieves high-availability even during failures
- Open source, standards-based avoids vendor lock-in



# APIs



# APIs

	Pipeline API (High-Level API)	java.util.stream	Core API (DAG API)
Use for	<ul style="list-style-type: none"><li>General purpose High-Level API for processing both bounded and unbounded data.</li></ul>	<ul style="list-style-type: none"><li>Filter-map-reduce operations on top of a bounded data set.</li><li>Fast adoption, as j.u.stream is a well-known Java 8 API.</li></ul>	<ul style="list-style-type: none"><li>Computations modelled as DAGs.</li><li>If the use case is too specific to match the Pipeline API.</li><li>For fine-tuned performance.</li><li>For building DSLs.</li></ul>
Declarative (what) x Imperative (how)	Declarative	Declarative	Imperative
Map/FlatMap/Filter	✓	✓	✓
Aggregations	✓	✓	✓
Joins	✓	✗	✓
Processing bounded data (batch)	✓	✓	✓
Processing unbounded data (streaming)	*✓	✗	✓



# Pipeline API

- General purpose, declarative API: both powerful and simple to use
- Recommended as the best place to start using Jet
- Supports fork, join, cogroup, map, filter, flatmap, reduce, groupby
- Works with all sinks and sources
- Is a DSL which is put through a planner and converted to DAG plan for execution
- Batch and Streaming (window support in 0.6)

See <https://github.com/hazelcast/hazelcast-jet-code-samples>



# Pipeline API Code Sample

```
JetInstance jet = Jet.newJetInstance();
Pattern delimiter = Pattern.compile("\\W+");
Pipeline p = Pipeline.create();
p.drawFrom(Sources.<Long, String>readMap(BOOK_LINES))
    .flatMap(e -> traverseArray(delimiter.split(e.getValue().toLowerCase()))
        .filter(word ->
!word.isEmpty()))
    .groupBy(wholeItem(), counting())
    .drainTo(Sinks.writeMap(COUNTS));
Job job = jet.newJob(p);
job.join();
```



# Distributed `java.util.stream` API

- Jet adds distributed support for the `java.util.stream` API for Hazelcast Map, List and Cache
- Supports all j.u.s. operations such as:
  - `map()`, `flatMap()`, `filter()`, `reduce()`, `collect()`, `sorted()`, `distinct()`
- Lambda serialization is solved by creating **Serializable** versions of the interfaces
- j.u.s streams are converted to Processor API (DAG) for execution
- Strictly a batch processing API
- Easiest place to start, but we recommend the Pipeline API to exploit all features of Jet

See <https://github.com/hazelcast/hazelcast-jet-code-samples>



# j.u.s API

```
JetInstance jet = Jet.newJetInstance();  
Jet.newJetInstance();  
IStreamMap<Long, String> lines = jet.getMap("lines");  
  
Map<String, Long> counts = lines  
    .stream()  
    .flatMap(m -> Stream.of(PATTERN.split(m.getValue().toLowerCase())))  
    .filter(w -> !w.isEmpty())  
    .collect(DistributedCollectors.toMap("counts", w -> w,  
w -> 1L, (left, right) -> left + right));
```



# DAG API: Powerful, Low Level API

DAG describes how vertices are connected to each other:

```
DAG dag = new DAG();
// nil -> (docId, docName)
Vertex source = dag.newVertex("source", readMap(DOCID_NAME));
// (docId, docName) -> lines
Vertex docLines = dag.newVertex("doc-lines",
    nonCooperative(flatMap((Entry<?, String> e) ->
        traverseStream(docLines(e.getValue())))))
);
// line -> words
Vertex tokenize = dag.newVertex("tokenize",
    flatMap((String line) -> traverseArray(delimiter.split(line.toLowerCase()))
        .filter(word -> !word.isEmpty()))
);
// word -> (word, count)
Vertex accumulate = dag.newVertex("accumulate", accumulateByKey(wholeItem(),
    AggregateOperations.counting()));
// (word, count) -> (word, count)
Vertex combine = dag.newVertex("combine", combineByKey(AggregateOperations.counting()));
// (word, count) -> nil
Vertex sink = dag.newVertex("sink", writeMap("counts"));

return dag.edge(between(source.localParallelism(1), docLines)
    .edge(between(docLines.localParallelism(1), tokenize))
    .edge(between(tokenize, accumulate).partitioned(wholeItem(), HASH_CODE))
    .edge(between(accumulate, combine).distributed().partitioned(entryKey()))
    .edge(between(combine, sink));
```





# Building Custom Processors

- Unified API for sinks, sources and intermediate steps
- Not required to be thread safe
- Each Processor has an **Inbox** and **Outbox** per inbound and outbound edge
- Two main methods to implement:

**boolean** tryProcess(**int** ordinal, **Object** item)

- Process incoming item and emit new items by populating the outbox

**boolean** complete()

- Called after all upstream processors are also completed. Typically used for sources and batch operations such as **group by** and **distinct**
- Non-cooperative processors may block indefinitely
- Cooperative processors must respect **Outbox** when emitting and yield if **Outbox** is already full



# Hazelcast Jet Professional Support



# Meet Your Business SLAs with Hazelcast Jet Professional Support

- Minimized downtime with 24/7 response to support tickets, continuous delivery of patches, and Hot Fix Patches
- Lower cost of operation of DevOps for your Hazelcast Jet apps with simulation/configuration testing
- Certified IP Compliance guarantees avoiding legal exposure from open source

## 100% Success Rate on Customer Issues

“As usual, the response was timely beyond expectations, and very good technical content returned. Exemplary support, hard to find in any company...”

—Fortune 100 Financial Services customer

### FEATURE COMPARISON

### HAZELCAST JET PROFESSIONAL SUPPORT

24x7 Support Window  
Service-Level Agreement

**4 hours**

Quarterly Review of Feature Requests



Quarterly Review of the Hazelcast Jet Roadmap



Email, IM, Phone Support Contacts

**2**

Hazelcast Jet Production Assurance for Initial Deployment/Charge Control with Hazelcast Simulator



Maintenance Patches



Access to Hot Fix Patches



IP Compliance Assurance with Black Duck ® Protex™





# Hazelcast Jet Production Assurance for Initial Deployment/Change Control with Hazelcast Simulator

Production simulation of your Hazelcast Jet environment including hardware, virtualization, operating system, configuration, feature usage, topology, sizing and load

Includes assistance with ongoing simulation and testing of your Hazelcast Jet platform and deployed applications using [Hazelcast Simulator](#)

- Simulate the expected throughput/latency with your specific requirements during pre-production
- Test if Hazelcast Jet behaves as expected when implementing new functionality, load characteristics or expansion
- Standardize your application deployment process with a battery of rigorous tests that we will run for you before initial deployment and on each upgrade

## Hazelcast Simulator

Watch 39 Star 28 Fork 39

Last year, we developed a production simulator and stress testing tool to test our Hazelcast releases and reproduce gnarly bugs. We are now releasing that to the community as Hazelcast Simulator. Now you can test Hazelcast for your specific hardware, OS, Java version, network configuration, Hazelcast configuration, feature use and load characteristics. Hazelcast Simulator can be run on-premise, on Amazon EC2 and on Google Compute Engine.

You can test Hazelcast for real-life production problems like network problems, overloaded CPU and failing nodes. It also provides a benchmarking and performance-testing platform. Finally it integrates with various out-of-the-box profilers.

Hazelcast Simulator lets you:

- Simulate the expected throughput/latency of Hazelcast with your specific requirements during the pre-production phase of your deployment.
- Test if Hazelcast behaves as expected when implementing new functionality in your project.
- Standardize your application deployment process with a battery of rigorous tests.
- Ease Hazelcast upgrades between versions using simulations to identify and remediate issues in advance.

Simulator 0.8.4 10/14/2016

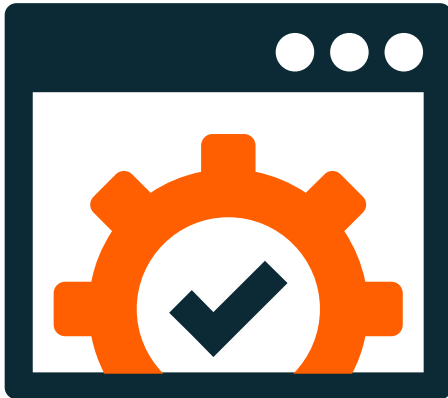
[ZIP, TAR](#)

[Reference Manual](#)

[Show More +](#)



# Hot Fix Patches



Personalized release of Hazelcast Jet software dedicated to addressing your specific reported issue

Made to the exact version of Hazelcast Jet that you are running with only the specific bug fix applied

Hot Fix Patches can thus be applied to your production system with no or minimal testing

We reproduce the problem, and apply the patch in our Simulator production simulation tool, to ensure we go red before we go green



# IP Compliance Assurance with Black Duck® Protex™

- Full insight into Hazelcast Jet IP compliance with Black Duck® Protex™ reporting to help you understand license obligations, conflicts and risks
- Black Duck Protex is the leading solution for managing open source compliance
- Regular code scans before minor and major releases
- We provide you a copy of the report on demand

Code Label	
hazelcast-os	
Code Base 70.55MB	
	% content
Open Source 70.55MB	100.0%
Reciprocal as Components <0.01MB	<0.1%
Reciprocal as Files 0.00MB	0.0%
Permissive 15.22MB	21.6%
Owned Open Source 55.32MB	78.4%
Proprietary <0.01MB	<0.1%
Licensed 3rd Party <0.01MB	<0.1%
Owned Proprietary 0.00MB	0.0%
Multi-Licensed 0.00MB	0.0%
Reciprocal and Proprietary 0.00MB	0.0%
Reciprocal and Owned 0.00MB	0.0%
Reciprocal and Permissive 0.00MB	0.0%
Licensed 3rd Party and Owned 0.00MB	0.0%
Other Multi-Licensed 0.00MB	0.0%
Unknown 0.00MB	0.0%
Unknown 0.00MB	0.0%
• Apache License 2.0	100.0%
• Creative Commons Public Domain Dedication License	<0.1%
• Eclipse Public License 1.0	<0.1%
• Unspecified	<0.1%
• MIT License	<0.1%
The code in this project cannot be used for purposes beyond Distribution	
Created: Jul 29, 2016 9:18:24 PM	


Components:  
Agrona (UNSPECIFIED)  
Apache License 2.0 (2.0)  
A tiny async implementation for Java (Performance Test) (1.7.0)  
castmapr (UNSPECIFIED)  
hazelcast (UNSPECIFIED)  
hazelcast-all (UNSPECIFIED)  
hazelcast-build-utils (UNSPECIFIED)  
hazelcast-client (UNSPECIFIED)  
hazelcast-client-legacy (UNSPECIFIED)  
hazelcast-client-new (UNSPECIFIED)  
Hazelcast Simulator Tests (0.4)  
hazelcast-spring (UNSPECIFIED)  
hazelcast-tuplespace (master-20101204)  
json (UNSPECIFIED)  
JSR-166 Concurrency Utilities (20160130, UNSPECIFIED)  
JUnit (UNSPECIFIED)  
OPS4J Pax Exam - JUnit Probe Invoker (UNSPECIFIED)

Furnished by:  
Powered by Black Duck Software

This report was created using Black Duck® products and services. Recipient of the results bears all of the risks relating to use of, or reliance upon, such results or any other content contained in this report. Black Duck makes no representation or warranty to any party regarding the contents of this report, its accuracy, completeness or correctness, and Black Duck hereby disclaims any and all warranties (both express and implied) with respect thereto.



# Hazelcast Jet Product Roadmap



# Hazelcast Jet 0.6 Features (03/2018)

Features	Description
<b>Distributed Computation</b>	Distributed data processing framework. It's based on directed acyclic graphs (DAGs) and in-memory computation
<b>Robust stream processing</b>	Fault tolerance based on distributed snapshots. When there is a failure (network failure or split, node failure), Jet uses the latest state snapshot and automatically restarts
<b>Ex-Once and At-least Once</b>	Jet can be configured to at-least-once or exactly-once semantics to favor performance or consistency
<b>Windowing with Event-time</b>	Improved streaming support including windowing support with event-time semantics. Out of the box support for tumbling, sliding and session window aggregations
<b>High-Level Pipeline API</b>	Convenient API to implement data processing pipeline
<b>Pivotal Cloud Foundry Tile</b>	Hazelcast Jet is available in Pivotal Cloud Foundry
<b>Spring Integration</b>	Integration with Spring Framework and Spring Boot
<b>High-Performance Hazelcast Integrations</b>	Integrates with both embedded and non-embedded Hazelcast instances to be used for reading and writing data for distributed computation. Change event reader allows streaming from Hazelcast IMDG
<b>File and Socket Connector</b>	Ability to read and write unbounded data from text sockets and files
<b>Kafka and HDFS Connector</b>	Reads and writes data streams to the Kafka message broker and Hadoop Distributed File System





# Hazelcast 2018 Jet Roadmap

Features	Description
Elasticity	Dynamic rescaling of the computation
Diagnostics and management	Management and monitoring features for Jet
More Connectors	JMS   JDBC
1.0 Maturity	To reach 1.0 maturity in 2018



**Thank you**